Bölüm 9: Sanal Bellek

İşletim Sistemleri

Talebe bağlı sayfalama · Sayfa hatası · Etkin erişim süresi · Yazınca kopyalama · mmap · FIFO ve Belady · OPT ve LRU · Saat, NRU, yaşlandırma · Çalışma kümesi, WSClock · Tahsis ve thrashing · Sayfa boyutu, TLB erimi · Buddy, slab · Linux ve Windows

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/os/slides/Bolum_09_Sanal_Bellek.html

GİRİŞ

Öğrenme Çıktıları

Talebe bağlı sayfalama

Sanal belleğin amacını açıklamak; geçerli–geçersiz biti, sayfa hatası işleme adımları ve komutun yeniden başlatılmasını anlatmak.

Performans

Sayfa hatası olasılığıyla etkin erişim süresini (EAT) hesaplamak; HDD, SSD ve sıkıştırılmış takas arasındaki farkı yorumlamak.

COW ve mmap

`fork()` sonrasında yazınca kopyalamayı ve bellek eşlemeli dosyaların sayfa önbelleğiyle ilişkisini izlemek.

Yer değiştirme

FIFO, OPT, LRU, ikinci şans, saat, NRU, yaşlandırma, çalışma kümesi ve WSClock'u bir referans dizisi üzerinde elle çalıştırmak.

Tahsis ve thrashing

Eşit/orantılı ve yerel/genel tahsisi karşılaştırmak; thrashing'i çalışma kümesi ve PFF ile açıklamak.

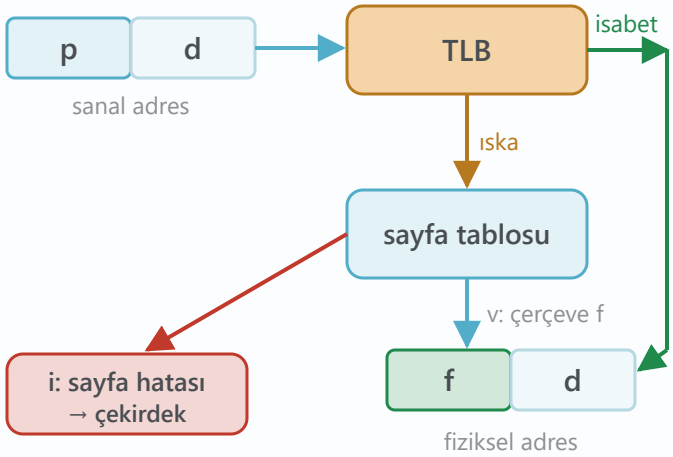
Tasarım ve gerçek sistemler

Sayfa boyutu, TLB erimi, program yapısı, sabitleme, buddy/slab; Linux (MGLRU, zswap) ve Windows yaklaşımlarını tanımak.

GİRİŞ

Sayfalama ve TLB

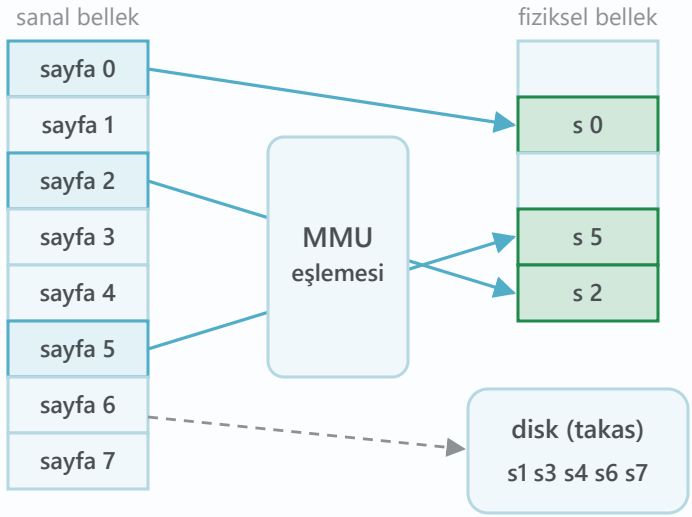
- Sanal adres = **(p, d)**; sayfa tablosu p'yi bir **çerçeveye** çevirir, bağıl konum d aynen kalır.
- **TLB** son çevirileri önbellekler; isabet oranı %99'un üstündedir, ıskada sayfa tablosu (x86-64'te 4 ya da 5 düzey) yürünür.
- Her girdi: çerçeve no, **geçerli** (present), **R** (accessed), **M** (dirty), koruma bitleri (R/W, U/S, NX).
- Girdi *geçersizse* sayfa hatası oluşur; sayfa **bellekte değilse** yedekleme deposundan getirilir ve gerekirse seçilen bir kurban sayfanın yerine yüklenir.



Etkileşimli simülatör: Sayfalama ve Adres Çevirisi Simülatörü: farklı adreslerle sanal adresten fiziksel adrese çeviri, TLB isabet/ıskası ve sayfa hataları.

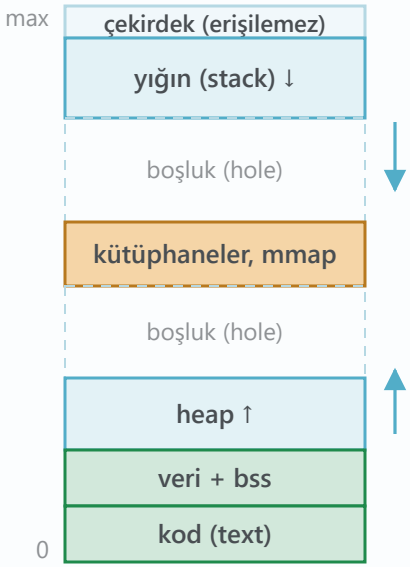
Neden Sanal Bellek?

- Programın tamamı nadiren kullanılır: hata işleme kodu, seyrek seçenekler, gereğinden büyük ayrılmış diziler.
- **Takas (swapping):** sürecin *tamamını* belleğe getirir/diske götürür; büyük süreçte pahalıdır.
- **Sanal bellek:** mantıksal belleği fiziksel bellekten ayırır; süreç **kısmen bellekteyken** çalışır.
- Kazançlar: program fiziksel bellekle sınırlı değildir; süreç başına daha az çerçeve → **daha çok süreç**; yükleme ve takas için daha az G/Ç; paylaşım kolaylaşır.
- Şişkin (bloat) yazılımı elle parçalara bölmek (eski *overlay* yöntemi) gerekmez; bunu İS sayfa düzeyinde yapar.



Sanal Adres Alanı

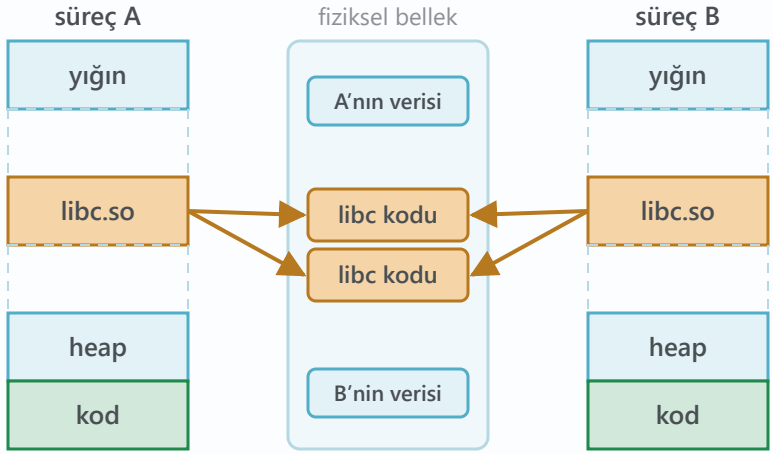
- Sürecin bellekteki **mantıksal görünümü**: 0'dan başlayan bitişik adresler. Fiziksel bellek ise dağınık **çerçevelerden** oluşur; eşlemeyi MMU yapar.
- Yığın en yüksek kullanıcı adresinden **aşağı**, heap **yukarı** büyür; aradaki kullanılmayan alan **boşluktur** ve sayfa ya da çerçeve harcamaz.
- Yığın/heap yeni bir sayfaya taşınana kadar o sayfa için fiziksel bellek gerekmez (*seyrek* adres alanı).
- x86-64: 48 bit sanal adres (kullanıcıya 128 TiB); 5 düzeyli sayfalamayla 57 bit. ASLR bölgelerin yerini her çalıştırmada rastgeleler.
- Sanal bellek iki yolla kurulabilir: **talebe bağlı sayfalama** (bugün evrensel) ve talebe bağlı kesimleme (tarihsel: Burroughs, OS/2).



Linux'ta bir sürecin bölgeleri: `cat /proc/self/maps`; her satır bir sanal bellek alanıdır (VMA).

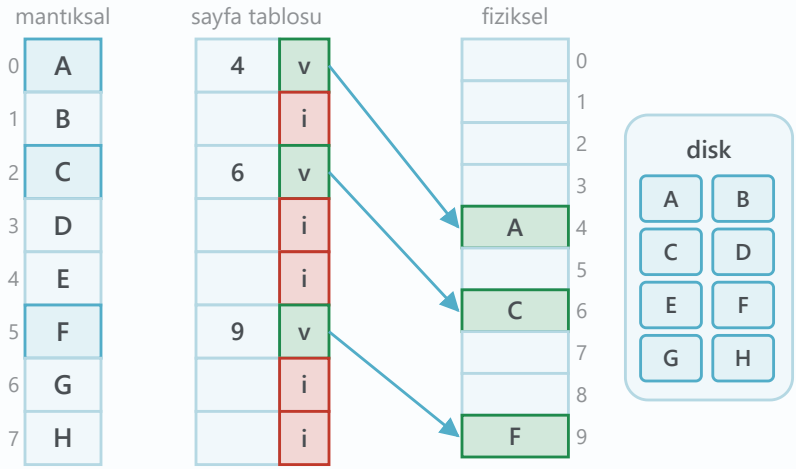
Paylaşımlı Sayfalar ve Kütüphaneler

- Sanal bellek, sayfaları birden çok süreç arasında **paylaştırmayı** kolaylaştırır: aynı çerçeve birden çok sayfa tablosunda görünür.
- **Paylaşımlı kütüphane** (libc, libstdc++): salt okunur kod sayfaları her süreçte kendi adresinde eşlenir, fiziksel kopya bir tanedir.
- **Paylaşımlı bellek** ile süreçler arası iletişim: iki süreç aynı bölgeyi eşler.
- `fork()` sırasında sayfalar kopyalanmadan paylaşılır (yazınca kopyalama).
- Yazılabilir veri sayfaları ise her süreçte **özeldir**.



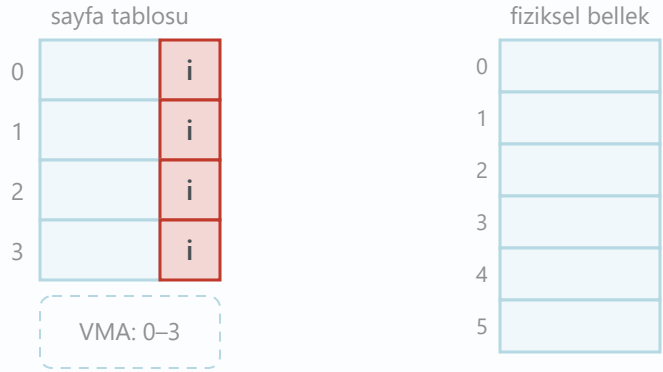
Talebe Bağlı Sayfalama

- Süreç başlarken hiçbir sayfa yüklenmez (ya da çok az sayfa yüklenir); bir sayfa **yalnızca erişildiğinde** getirilir (*tembel* yükleme).
- Daha az G/Ç, daha az bellek, daha hızlı başlatma, daha çok kullanıcı.
- Erişimde: adres **geçersizse** (hiçbir bölgede değilse) süreç sonlandırılır; geçerli ama **bellekte değilse** sayfa getirilir.
- Bütün süreçle değil tek tek sayfalarla uğraşan birime **sayfalayıcı** (pager) denir; "tembel takasçı" (lazy swapper) eski ve yanıltıcı bir addır.
- **Saf** talebe bağlı sayfalamada ilk komut bile sayfa hatasıyla başlar; yerellik sayesinde hata oranı kısa sürede düşer.



Geçerli–Geçersiz Biti

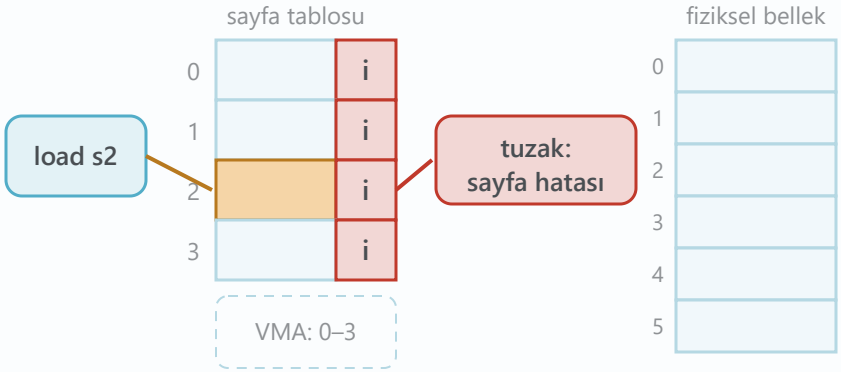
- Her girdide bir bit: **v** = geçerli ve bellekte, **i** = bellekte değil *ya da* adres alanında değil. İkisini ayırmak İS'nin işidir.



Başlangıçta tablonun **tüm girdileri** `i` (geçersiz/bellekte değil) yapılır; hiçbir çerçeve ayrılmamıştır.

Geçerli–Geçersiz Biti

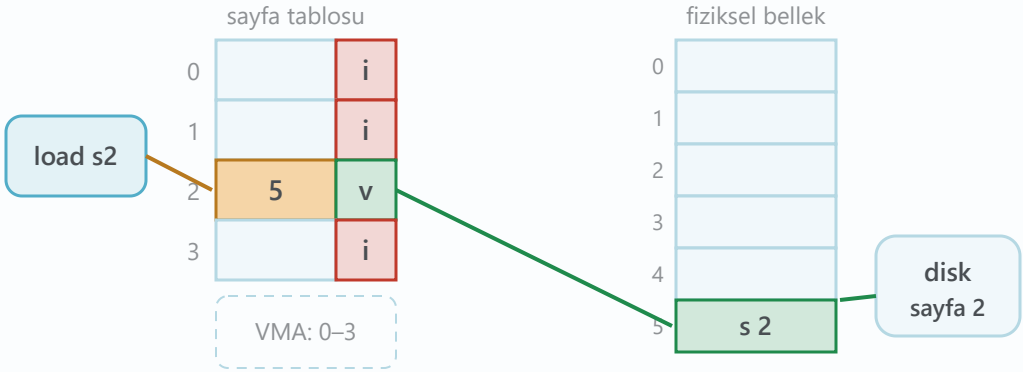
- Her girdide bir bit: **v** = geçerli ve bellekte, **i** = bellekte değil *ya da* adres alanında değil. İkisini ayırmak İS'nin işidir.



CPU sayfa 2'ye erişir; MMU girdide `i` görür ve **sayfa hatası** tuzağı üretir.

Geçerli–Geçersiz Biti

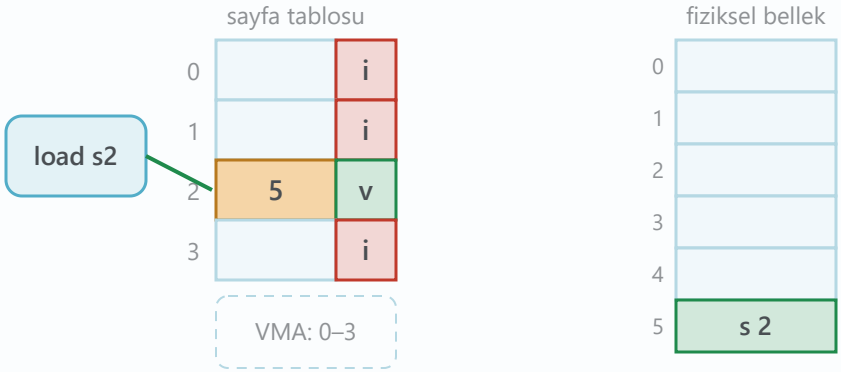
- Her girdide bir bit: **v** = geçerli ve bellekte, **i** = bellekte değil *ya da* adres alanında değil. İkisini ayırmak İS'nin işidir.



Çekirdek adresin geçerli bir bölgede (VMA) olduğunu doğrular, sayfayı boş çerçeve 5'e okur, girdiyi `5, v` yapar; komut yeniden çalışır.

Geçerli–Geçersiz Biti

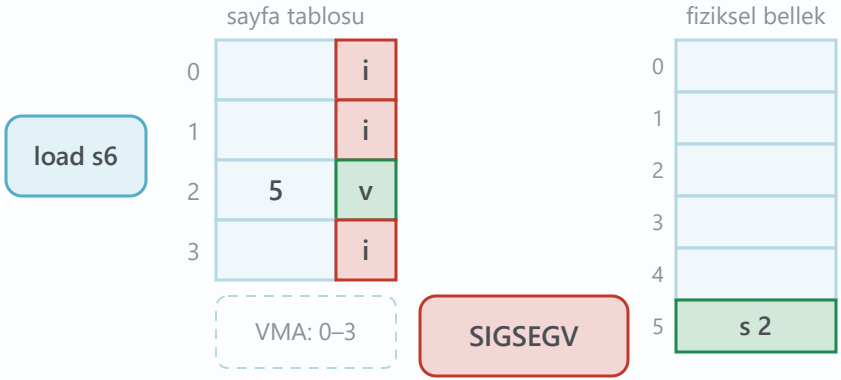
- Her girdide bir bit: **v** = geçerli ve bellekte, **i** = bellekte değil *ya da* adres alanında değil. İkisini ayırmak İS'nin işidir.



Aynı sayfaya sonraki erişimler artık doğrudan çevrilir (TLB'ye de yüklenir).

Geçerli–Geçersiz Biti

- Her girdide bir bit: **v** = geçerli ve bellekte, **i** = bellekte değil *ya da* adres alanında değil. İkisini ayırmak İS'nin işidir.



Sayfa 6 hiçbir bölgede değildir: bu bir **geçersiz referanstır**; çekirdek süreci `SIGSEGV` ile sonlandırır.

Sayfa Hatasının İşlenmesi

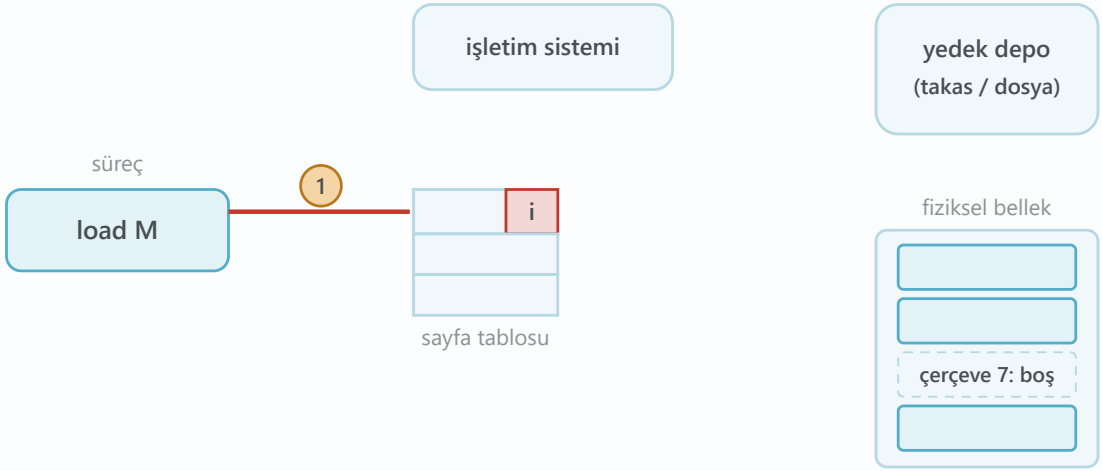


Süreç M adresine erişir; sayfa tablosunda girdinin biti `i`.

Adım 1 / 7

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

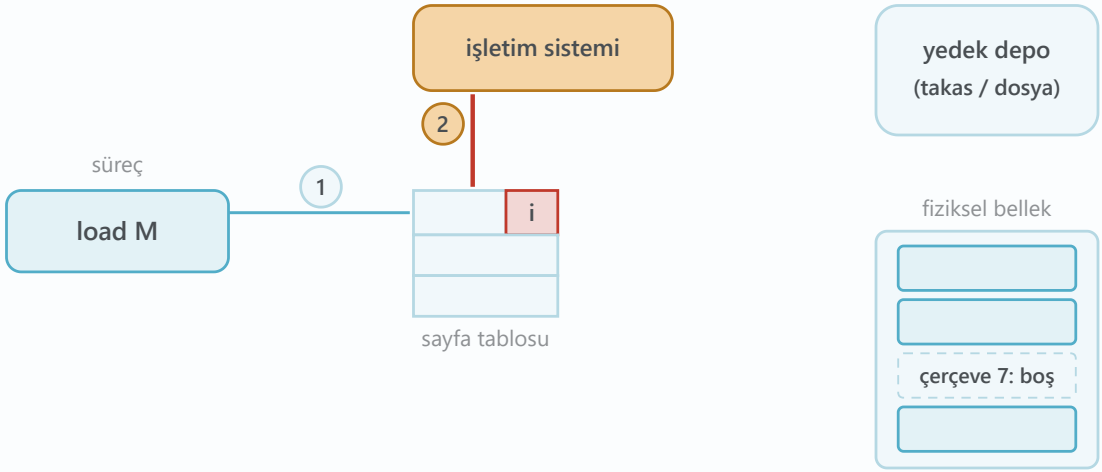
Sayfa Hatasının İşlenmesi



1. Referans: MMU girdiyi okur, sayfa bellekte değil.

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

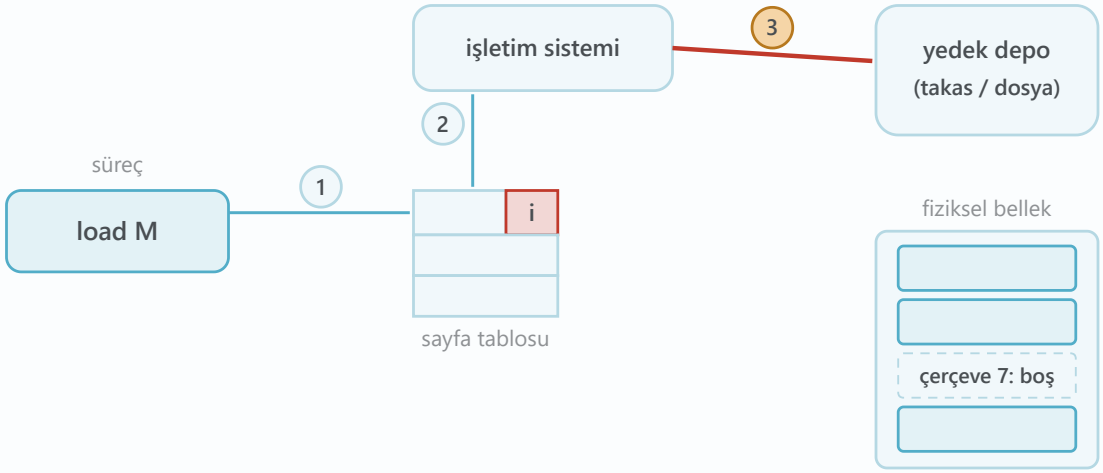
Sayfa Hatasının İşlenmesi



2. Tuzak: donanım çekirdeğe geçer; çekirdek referansın geçerli olduğunu doğrular (değilse süreci sonlandırır).

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

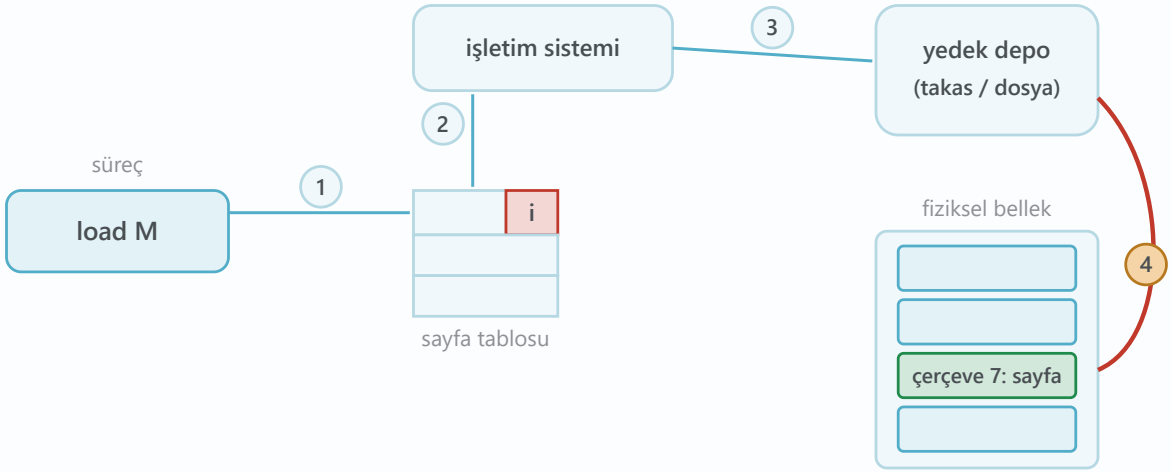
Sayfa Hatasının İşlenmesi



3. Sayfa yedek depoda: sayfanın diskteki yeri bulunur (takas alanı ya da dosya), boş bir çerçeve seçilir.

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

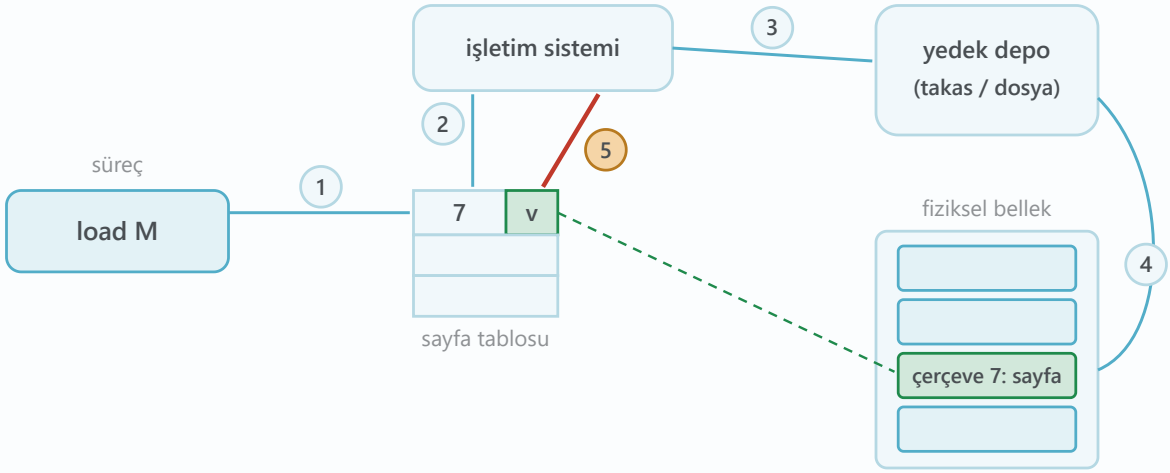
Sayfa Hatasının İşlenmesi



4. Getir: disk okuma işi çizelgelenir; sayfa çerçeve 7'ye okunur. Bu sırada CPU başka sürece verilir.

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

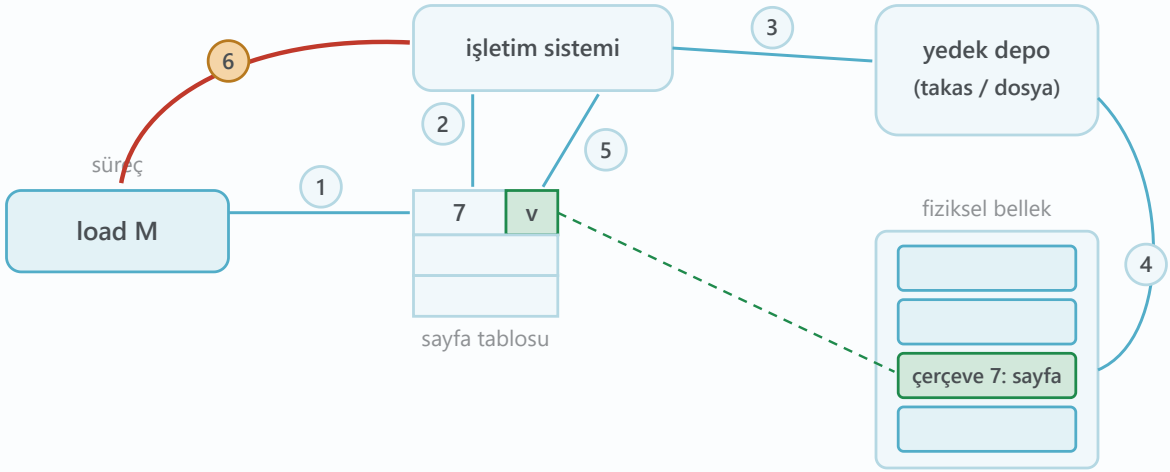
Sayfa Hatasının İşlenmesi



5. Tabloyu güncelle: girdiye çerçeve 7 yazılır, bit `v` yapılır.

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

Sayfa Hatasının İşlenmesi



6. Yeniden başlat: hataya yol açan komut baştan çalışır; süreç sayfa hiç eksik değilmiş gibi devam eder.

Silberschatz Şekil 10.5'in yeniden çizimi. Sayfa hatası bir *hata (fault)* türü tuzaktır: kaydedilen PC hatalı komutu gösterir, böylece komut yeniden çalıştırılabilir. x86'da hatalı adres `CR2` yazmacına yazılır. Kirli kurban önce diske yazılır; bu sırada çerçeve meşgul işaretlenir.

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
▶ int ceviri(int va) {
2   int p = va >> 12;       /* sayfa no */
3   int d = va & 0xFFF;     /* bağıl konum */
4   int f = tlb_ara(p);     /* -1: ıska */
5   if (f < 0) {
6     if (!pt[p].gecerli)   /* bellekte yok */
7       sayfa_hatasi(p);    /* diskten al */
8     f = pt[p].cerceve;
9     tlb_ekle(p, f);
10  }
11  pt[p].R = 1;
12  return (f << 12) | d;
13 }
```



TLB'de sayfa 0 ve 1'in çevirisi var; sayfa 2 bellekte (çerçeve 5) ama TLB'de değil, sayfa 3 diskte. İlk çağrı: `ceviri(0x2A10)`.

Adım 1 / 12

DEĞİŞKENLER

va = **0x2A10** | p = **–** | d = **–** | f = **–**

ÇIKTI

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



Üst bitler sayfa numarası, alt 12 bit bağıl konum: 0x2A10 → p = 2, d = 0xA10 (4 KB sayfa).

Adım 2 / 12

DEĞİŞKENLER

va = 0x2A10 | p = 2 | d = 0xA10 | f = –

ÇIKTI

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
▶    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



TLB'de p = 2 yok: **ıska**, f = −1. Çeviri için sayfa tablosuna gidilir (bellek erişimi).

Adım 3 / 12

DEĞİŞKENLER

va = 0x2A10 | p = 2 | d = 0xA10 | f = −1

ÇIKTI

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
▶    if (f < 0) {
▶      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```

VA | p = 2 | d = 0xA10 | → PA | –

TLB

p	f
0	1
1	4

Sayfa tablosu

p	bit	f	R
0	v	1	0
1	v	4	0
2	v	5	0
3	i	–	0

Disk

s3

Girdi 2'nin biti `v` : sayfa bellekte, sayfa hatası **yok**; 7. satır atlanır.

Adım 4 / 12

DEĞİŞKENLER

`va = 0x2A10` `p = 2` `d = 0xA10` `f = -1`

ÇIKTI

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
▶      f = pt[p].cerceve;
▶      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



f = 5 tablodan okunur ve (2, 5) TLB'ye eklenir: aynı sayfaya bir sonraki erişim tablo yürümeden çevrilir.

Adım 5 / 12

DEĞİŞKENLER

va = 0x2A10 | p = 2 | d = 0xA10 | f = 5

ÇIKTI

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
▶    pt[p].R = 1;
▶    return (f << 12) | d;
13 }
```



R biti 1 olur (yer değiştirme algoritmaları bunu kullanır); fiziksel adres (5 << 12) | 0xA10 = **0x5A10**.

Adım 6 / 12

DEĞİŞKENLER

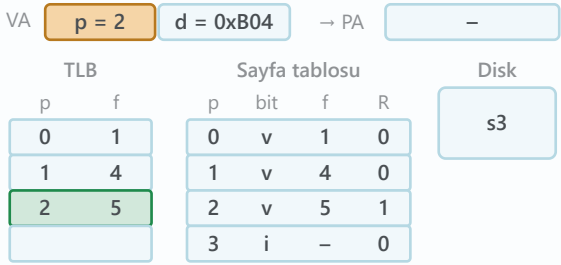
`va = 0x2A10` `p = 2` `d = 0xA10` `f = 5`

ÇIKTI

```
0x2A10 -> 0x5A10
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
▶    int p = va >> 12;       /* sayfa no */
▶    int d = va & 0xFFF;     /* bağıl konum */
▶    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



`ceviri(0x2B04)`: p = 2, d = 0xB04. TLB'de **isabet**, f = 5; sayfa tablosuna hiç gidilmez.

Adım 7 / 12

DEĞİŞKENLER

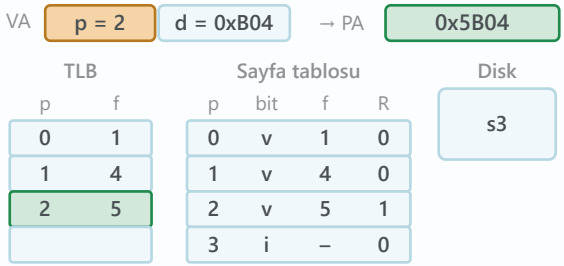
va = 0x2B04 | p = 2 | d = 0xB04 | f = 5

ÇIKTI

```
0x2A10 -> 0x5A10
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
▶    pt[p].R = 1;
▶    return (f << 12) | d;
13 }
```



Aynı sayfadaki ikinci erişim: if bloğu atlandı, sonuç **0x5B04**. Yerellik sayesinde çoğu erişim bu yolu izler.

Adım 8 / 12

DEĞİŞKENLER

`va = 0x2B04` `p = 2` `d = 0xB04` `f = 5`

ÇIKTI

```
0x2A10 -> 0x5A10
0x2B04 -> 0x5B04
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
▶    int p = va >> 12;       /* sayfa no */
▶    int d = va & 0xFFF;     /* bağıl konum */
▶    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



`ceviri(0x3008)`: p = 3, d = 0x008. TLB'de yok: ıska, f = −1.

Adım 9 / 12

DEĞİŞKENLER

`va = 0x3008` `p = 3` `d = 0x008` `f = -1`

ÇIKTI

```
0x2A10 -> 0x5A10
0x2B04 -> 0x5B04
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
▶      if (!pt[p].gecerli)   /* bellekte yok */
▶        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



Girdi 3'ün biti `i`: **sayfa hatası**. Çekirdek sayfayı diskten boş çerçeve 7'ye okur; süreç bu sırada bekler.

Adım 10 / 12

DEĞİŞKENLER

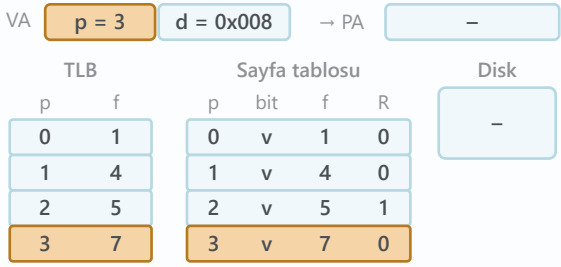
`va = 0x3008` `p = 3` `d = 0x008` `f = -1`

ÇIKTI

```
0x2A10 -> 0x5A10
0x2B04 -> 0x5B04
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
▶      f = pt[p].cerceve;
▶      tlb_ekle(p, f);
10   }
11   pt[p].R = 1;
12   return (f << 12) | d;
13 }
```



Hata işleyicisi girdiyi (v, 7) yaptı; komut yeniden çalışınca f = 7 okunur ve TLB'ye eklenir.

Adım 11 / 12

DEĞİŞKENLER

va = **0x3008** | p = **3** | d = **0x008** | f = **7**

ÇIKTI

```
0x2A10 -> 0x5A10
0x2B04 -> 0x5B04
```

Adres Çevirisi ve Sayfa Hatası: Kod İzleme

```
1  int ceviri(int va) {
2    int p = va >> 12;       /* sayfa no */
3    int d = va & 0xFFF;     /* bağıl konum */
4    int f = tlb_ara(p);     /* -1: ıska */
5    if (f < 0) {
6      if (!pt[p].gecerli)   /* bellekte yok */
7        sayfa_hatasi(p);    /* diskten al */
8      f = pt[p].cerceve;
9      tlb_ekle(p, f);
10   }
▶    pt[p].R = 1;
▶    return (f << 12) | d;
13 }
```

VA | p = 3 | d = 0x008 → PA | 0x7008

TLB

p	f
0	1
1	4
2	5
3	7

Sayfa tablosu

p	bit	f	R
0	v	1	0
1	v	4	0
2	v	5	1
3	v	7	1

Disk: –

Sonuç **0x7008**. Üç çeviride: 1 TLB isabeti, 2 ıska, bunlardan biri sayfa hatası (disk G/Ç'si).

Adım 12 / 12

DEĞİŞKENLER

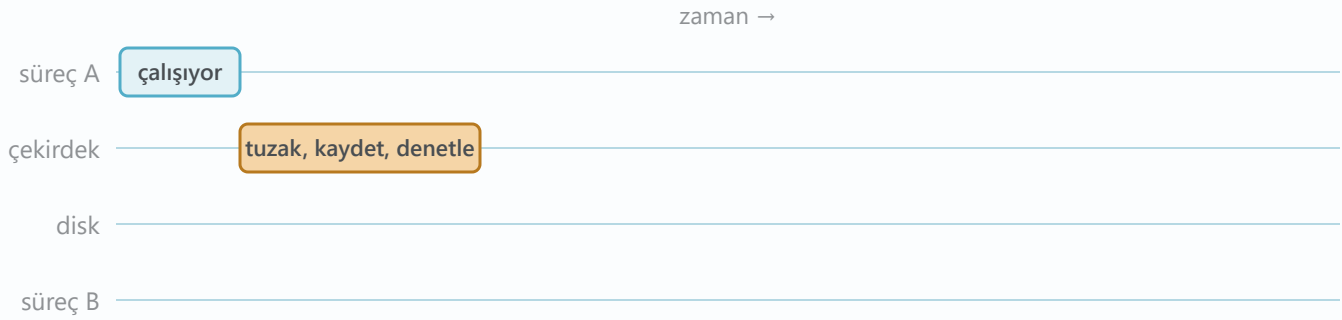
va = **0x3008** | p = **3** | d = **0x008** | f = **7**

ÇIKTI

```
0x2A10 -> 0x5A10
0x2B04 -> 0x5B04
0x3008 -> 0x7008
```

Sayfa Hatası: Zaman Çizelgesi

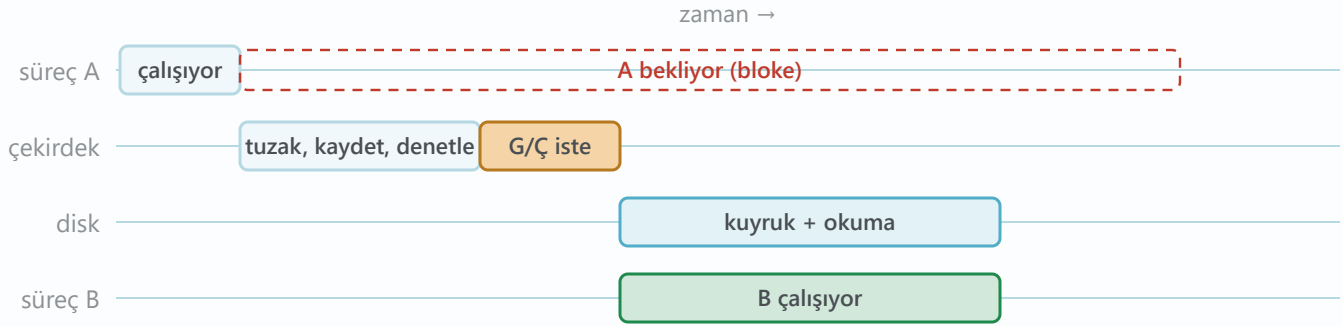
Ders kitabındaki 15 adım üç büyük maliyete ayrılır: **tuzağı işlemek** (µs), **sayfayı okumak** (HDD'de ms, NVMe'de 10–100 µs), **süreci yeniden başlatmak** (µs).



1–5. Çekirdeğe tuzak; yazmaçlar ve süreç durumu kaydedilir; kesmenin sayfa hatası olduğu ve referansın geçerli olduğu belirlenir; sayfanın diskteki yeri bulunur.

Sayfa Hatası: Zaman Çizelgesi

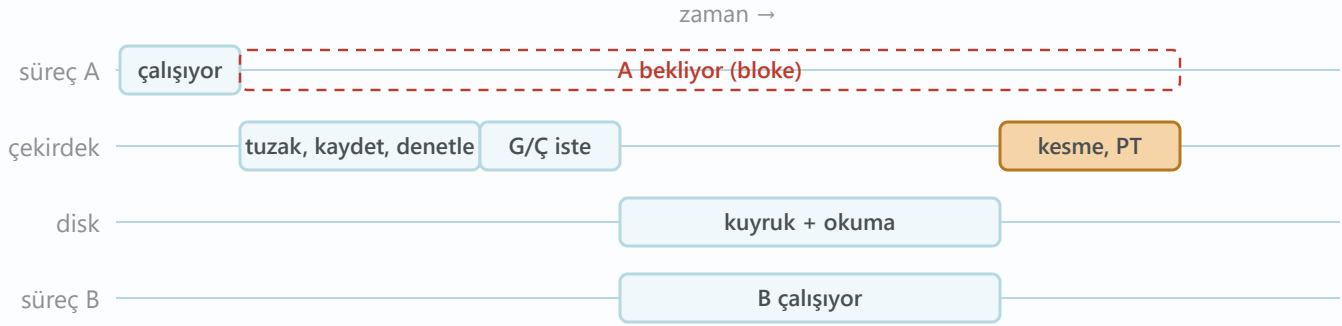
Ders kitabındaki 15 adım üç büyük maliyete ayrılır: **tuzağı işlemek** (µs), **sayfayı okumak** (HDD'de ms, NVMe'de 10–100 µs), **süreci yeniden başlatmak** (µs).



6–9. Disk/SSD kuyruğuna okuma isteği konur; aygıt beklenir (HDD'de arama + dönme gecikmesi). Bu arada **CPU başka bir sürece** (B) verilir.

Sayfa Hatası: Zaman Çizelgesi

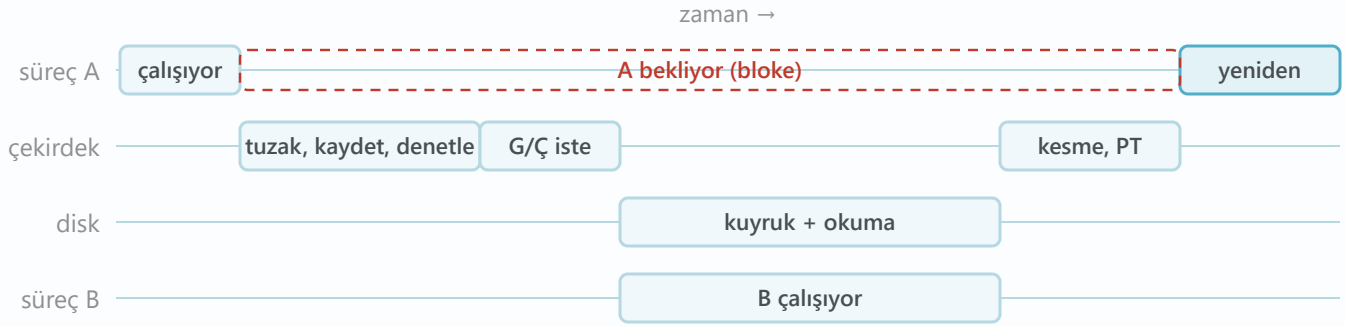
Ders kitabındaki 15 adım üç büyük maliyete ayrılır: **tuzağı işlemek** (µs), **sayfayı okumak** (HDD'de ms, NVMe'de 10–100 µs), **süreci yeniden başlatmak** (µs).



10–13. G/Ç tamamlandı kesmesi gelir; B'nin durumu kaydedilir, kesmenin diskten geldiği anlaşılır, sayfa tablosu güncellenir, A hazır kuyruğuna alınır.

Sayfa Hatası: Zaman Çizelgesi

Ders kitabındaki 15 adım üç büyük maliyete ayrılır: **tuzağı işlemek** (µs), **sayfayı okumak** (HDD'de ms, NVMe'de 10–100 µs), **süreci yeniden başlatmak** (µs).



14–15. A'ya yeniden CPU verilince yazmaçları ve sayfa tablosu yüklenir; yarıda kalan komut **baştan** çalışır.

Komutu Yeniden Başlatma ve Hata Türleri

Yeniden başlatma sorunu

- Hata komut *getirilirken* ya da işlenen okunurken olabilir; çoğu durumda komutu baştan çalıştırmak yeter.
- Zor durum: komut bir kısmını **değiştirdikten sonra** hata alır. Örn. IBM 360 `MVC` blok taşıma sayfa sınırını aşar; otomatik artırmalı adresleme yazmacı değiştirmiştir.
- Motorola 680x0'da üç sözcüklü `MOVE.L 6(A1),2(A0)` işlenen okunurken hata alırsa İS komutun nerede başladığını bilemez; bazı CPU'lar bu yüzden her komuttan önce PC'yi gizli bir yazmaca kopyalar.
- Çözümler: komut başlamadan iki ucu da yoklamak; eski değerleri geçici yazmaçlarda tutup geri almak. Modern CPU'larda **kesin istisnalar** (precise exceptions) bunu donanımda garanti eder.

Hata türleri (Linux)

Tür	Ne olur?
Küçük (minor)	Sayfa zaten bellekte (sayfa önbelleği, paylaşımlı kütüphane, sıfır sayfa, COW): yalnızca tablo güncellenir, G/Ç yok.
Büyük (major)	Sayfa diskten/takastan okunmalı: G/Ç ve bekleme var.
Geçersiz	Bölge yok ya da izin ihlali: `SIGSEGV`.

```
$ /usr/bin/time -v ./prog | grep -i fault
$ ps -o min_flt,maj_flt -p PID
$ perf stat -e page-faults,major-faults ./prog
```

PERFORMANS

Etkin Erişim Süresi (EAT)

Formül

EAT = (1 − p) × *ma* + p × *sayfa hatası süresi*

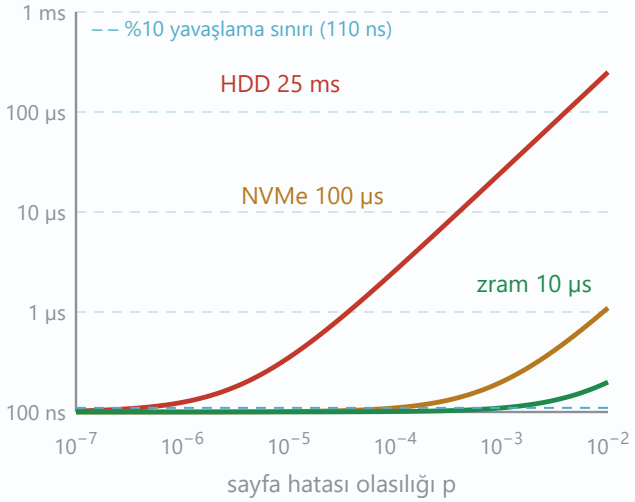
ma: bellek erişimi (100 ns); hata süresi: tuzak + okuma + yeniden başlatma. p = 0 → hiç hata yok.

HDD: 25 ms = 25.000.000 ns

EAT = 100(1 − p) + 25.000.000 p = 100 + 24.999.900 p

p = 0,001 → EAT = **25.099,9 ns** ≈ 25 µs

Bin erişimde bir hata, belleği yaklaşık **251 kat** yavaşlatır.



Adım 1 / 3

Silberschatz 10e aynı hesabı 200 ns ve 8 ms ile yapar: p = 0,001 → 8,2 µs (41 kat). zram/zswap: sayfa RAM'de sıkıştırılmış durur, "okuma" bir sıkıştırma çözme işlemidir.

PERFORMANS

Etkin Erişim Süresi (EAT)

Formül

EAT = (1 − p) × *ma* + p × *sayfa hatası süresi*

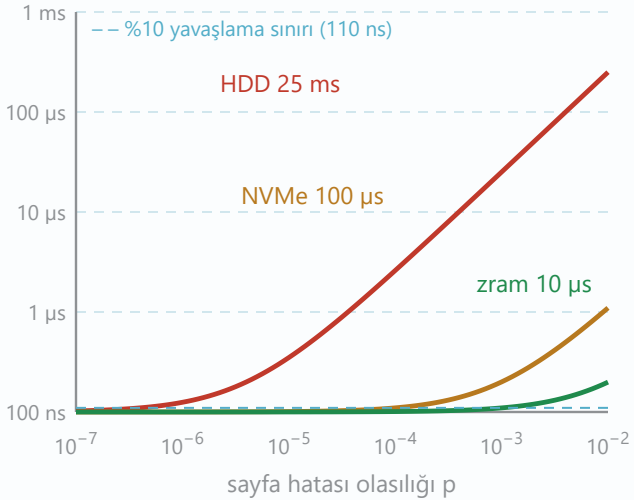
ma: bellek erişimi (100 ns); hata süresi: tuzak + okuma + yeniden başlatma. p = 0 → hiç hata yok.

Hedef: en çok %10 yavaşlama

110 > 100 + 24.999.900 p → p < 4,0 × 10^{-7}

yani **2,5 milyon erişimde birden az** hata

p < 0,0000004: hata oranı son derece düşük tutulmalıdır.



Adım 2 / 3

Silberschatz 10e aynı hesabı 200 ns ve 8 ms ile yapar: p = 0,001 → 8,2 µs (41 kat).

zram/zswap: sayfa RAM'de sıkıştırılmış durur, "okuma" bir sıkıştırma çözme işlemidir.

PERFORMANS

Etkin Erişim Süresi (EAT)

Formül

EAT = (1 − p) × *ma* + p × *sayfa hatası süresi*

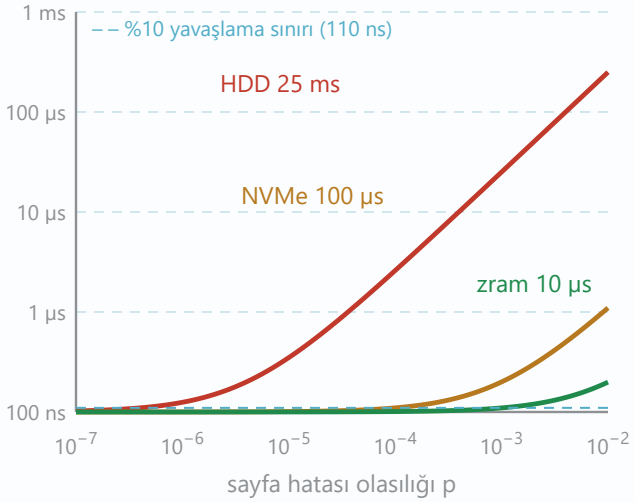
ma: bellek erişimi (100 ns); hata süresi: tuzak + okuma + yeniden başlatma. p = 0 → hiç hata yok.

2026: NVMe 10–100 µs, zram ≈ 10 µs

NVMe (100 µs ile), p = 0,001 → EAT ≈ **199,9 ns**; %10 için $p < 1{,}0 \times 10^{-4}$

zram, p = 0,001 → EAT ≈ **109,9 ns**

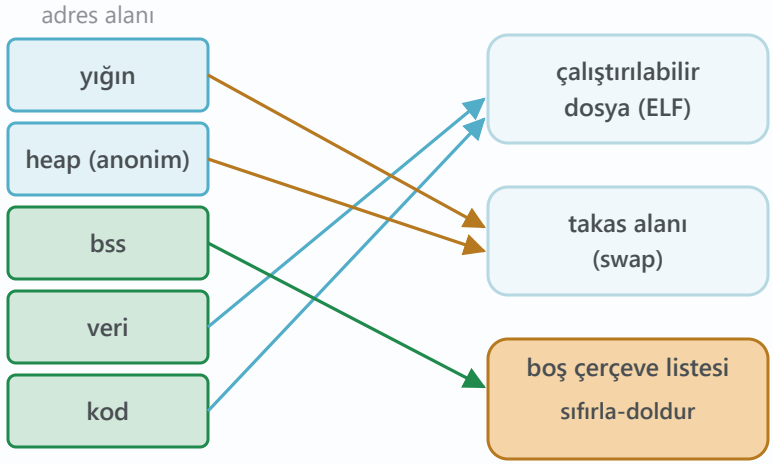
Hızlı depolama cezayı iki üç büyüklük mertebesi (250–2500 kat) küçültür ama bellek hâlâ **yüzlerce kat** hızlıdır; bu yüzden yer değiştirme algoritmaları önemini korur.



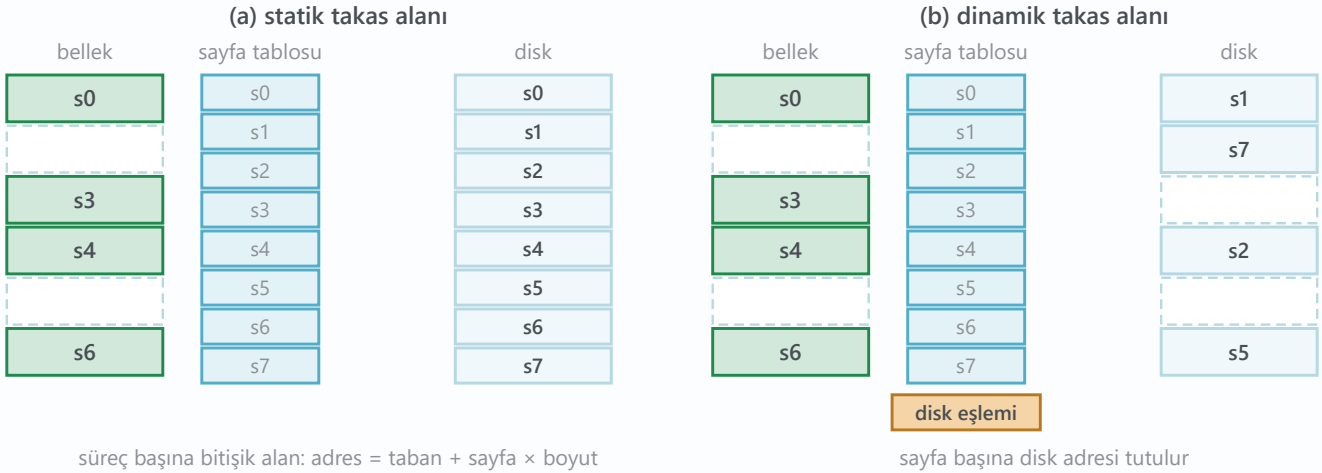
Silberschatz 10e aynı hesabı 200 ns ve 8 ms ile yapar: p = 0,001 → 8,2 µs (41 kat). zram/zswap: sayfa RAM'de sıkıştırılmış durur, "okuma" bir sıkıştırma çözme işlemidir.

Sayfalar Nereden Gelir?

- **Dosya destekli** sayfalar (kod, salt okunur veri, `mmap` edilmiş dosya): temizse atılır, gerektiğinde dosyadan *yeniden okunur*.
- **Anonim** sayfalar (heap, yığın, değiştirilmiş veri): dosyası yoktur; atılmadan önce **takas alanına** yazılmalıdır.
- **Sıfırla-doldur** (zero-fill-on-demand): bss ve yeni heap sayfaları ilk erişimde sıfırlanmış bir çerçeveyle karşılanır; eski verinin sızmaması için şarttır.
- İS bir **boş çerçeve havuzu** tutar; hata anında arama yapmak gerekmez. Havuz azalınca geri kazanım başlar.
- Takas alanı ayrı bölüm ya da dosya olabilir; Linux'ta `swapon --show`.



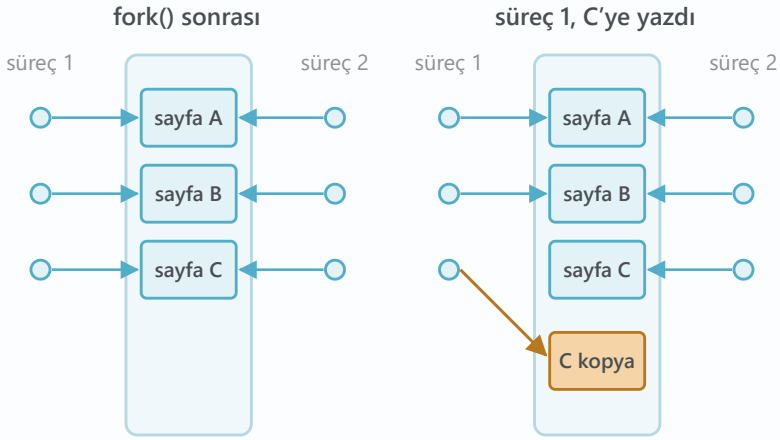
Yedekleme Deposu (Takas Alanı)



- **Statik**: süreç başlarken tüm boyu kadar bitişik takas alanı ayrılır; başlangıç adresi süreç tablosunda. Kod, veri ve yığın için ayrı alanlar olabilir; yığın büyüyebilir.
- **Dinamik**: önceden yer ayrılmaz; sayfa diske çıkarken bir yuva bulunur, adresi sayfa başına **disk eşleminde** tutulur. Değişmemiş kod sayfaları hiç takas edilmez, çalıştırılabilir dosyadan yeniden okunur.
- Yer: ayrı bir disk/bölüm (Linux takas bölümü) ya da dosya sistemindeki dosya (`swapfile` , Windows `pagefile.sys`).

Yazınca Kopyalama (Copy-on-Write)

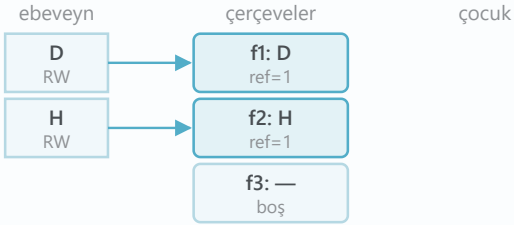
- `fork()` ebeveynin bütün sayfalarını kopyalamaz; ebeveyn ve çocuk **aynı sayfaları paylaşır**, yazılabilir sayfalar iki tarafta da **salt okunur + COW** işaretlenir.
- Biri yazınca koruma hatası oluşur; çekirdek o **tek sayfayı** kopyalar ve yazana yazılabilir kopyayı eşler.
- Hiç değiştirilmeyen sayfalar (kod, salt okunur veri) sonsuza dek paylaşılır; `fork()` + `exec()` neredeyse bedavaya gelir.
- Kopya için çerçeve, sıfırla-doldur havuzundan alınır.
- `vfork()` sayfa tablosunu bile kopyalamaz (ebeveyn bekler); bugün `posix_spawn()` tercih edilir.



Paylaşılan birim dosya değil **sayfadır**: yalnızca yazılan sayfa kopyalanır, diğerleri ortak kalır.

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
▶    char *b = malloc(4096);   /* heap: H */
▶    b[0] = 'a';
5    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
7      printf("%d\n", x);      /* okuma */
8      x = 2;                  /* yazma */
9      _exit(0);
10   }
11   wait(NULL);
12   b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



Ebeveyn D (veri) ve H (heap) sayfalarını yazılabilir olarak kullanıyor.

Adım 1 / 6

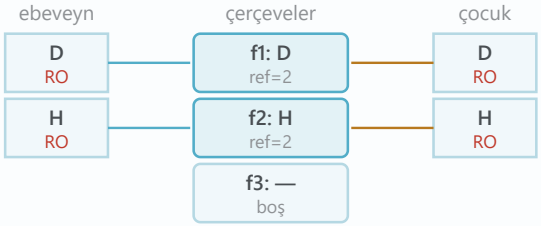
DEĞİŞKENLER

`ebeveyn.pid = –` `ebeveyn.x = 1` `kopya = 0`

ÇIKTI

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
3    char *b = malloc(4096);   /* heap: H */
4    b[0] = 'a';
▶    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
7      printf("%d\n", x);      /* okuma */
8      x = 2;                  /* yazma */
9      _exit(0);
10   }
11   wait(NULL);
12   b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



`fork()` : yalnızca sayfa tabloları kopyalanır; iki taraf da **salt okunur** (RO), referans sayıları 2.

Adım 2 / 6

DEĞİŞKENLER

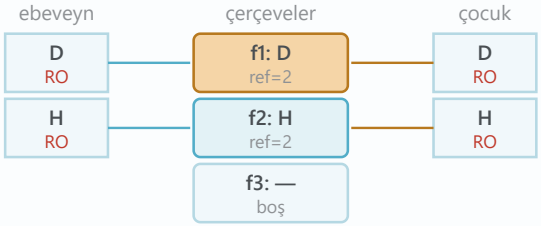
`çocuk.pid = 0` `ebeveyn.pid = 4242`

`çocuk.x = 1` `ebeveyn.x = 1` `kopya = 0`

ÇIKTI

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
3    char *b = malloc(4096);   /* heap: H */
4    b[0] = 'a';
5    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
▶      printf("%d\n", x);      /* okuma */
8      x = 2;                  /* yazma */
9      _exit(0);
10   }
11   wait(NULL);
12   b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



Çocuk `x`'i okur: okuma izinli olduğundan **hata yok**, kopya yok.

Adım 3 / 6

DEĞİŞKENLER

`çocuk.pid = 0` `çocuk.x = 1`

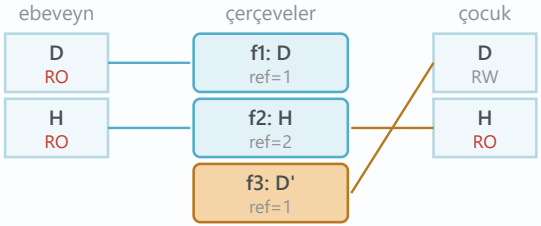
`ebeveyn.x = 1` `kopya = 0`

ÇIKTI

```
1
```

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
3    char *b = malloc(4096);   /* heap: H */
4    b[0] = 'a';
5    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
7      printf("%d\n", x);      /* okuma */
▶      x = 2;                  /* yazma */
9      _exit(0);
10   }
11   wait(NULL);
12   b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



Çocuk `x = 2` yazar → koruma hatası. Çekirdek COW'u tanır: f3'ü ayırır, D'yi kopyalar, çocuğa **RW** eşler.

Adım 4 / 6

DEĞİŞKENLER

`çocuk.pid = 0` `çocuk.x = 2`

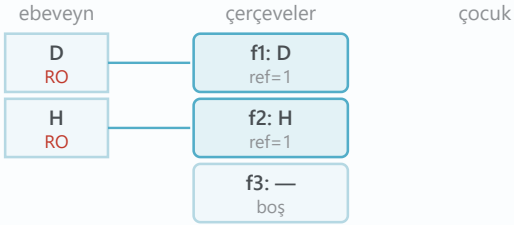
`ebeveyn.x = 1` `kopya = 1`

ÇIKTI

```
1
```

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
3    char *b = malloc(4096);   /* heap: H */
4    b[0] = 'a';
5    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
7      printf("%d\n", x);      /* okuma */
8      x = 2;                  /* yazma */
▶      _exit(0);
10   }
11   wait(NULL);
12   b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



Çocuk çıkar: eşlemeleri kaldırılır, f3 boşa düşer, f2'nin referans sayısı da 1'e iner.

Adım 5 / 6

DEĞİŞKENLER

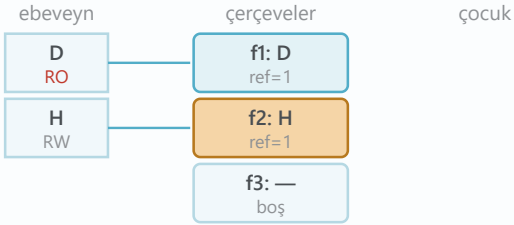
`çocuk.pid = 0` `çocuk.x = 2` `ebeveyn.x = 1` `kopya = 1`

ÇIKTI

```
1
```

fork() ve COW: Adım Adım

```
1  int x = 1;                  /* veri: D */
2  int main(void) {
3    char *b = malloc(4096);   /* heap: H */
4    b[0] = 'a';
5    pid_t pid = fork();
6    if (pid == 0) {           /* çocuk */
7      printf("%d\n", x);      /* okuma */
8      x = 2;                  /* yazma */
9      _exit(0);
10   }
11   wait(NULL);
▶    b[0] = 'b';               /* ebeveyn */
13   return 0;
14 }
```



Ebeveyn H'ye yazar → hata; referans sayısı 1 olduğundan **kopyalamaya gerek yok**, sayfa yalnızca yazılabilir yapılır.

Adım 6 / 6

DEĞİŞKENLER

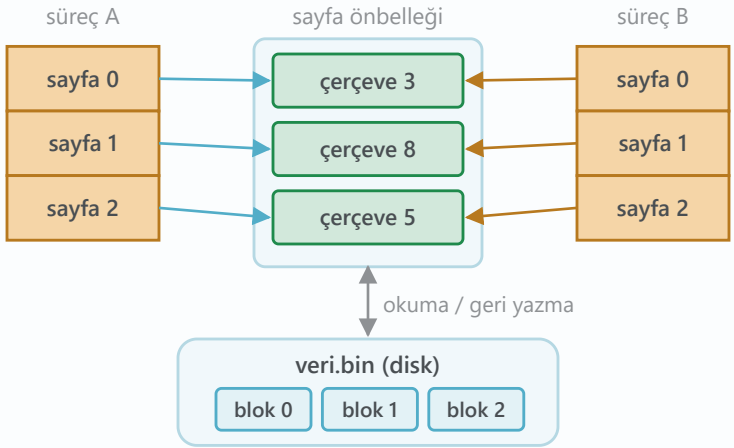
`ebeveyn.pid = 4242` `ebeveyn.x = 1` `kopya = 1`

ÇIKTI

```
1
```

Bellek Eşlemeli Dosyalar

- `mmap()` bir dosyayı adres alanının bir bölgesine eşler; dosya `read()/write()` yerine **bellek erişimiyle** okunup yazılır.
- Sayfalar talebe bağlı getirilir; ilk erişim (küçük ya da büyük) sayfa hatası üretir. Kirli sayfalar arka planda ya da `msync()` / `munmap()` ile dosyaya yazılır.
- `MAP_SHARED` : değişiklikler dosyaya ve aynı dosyayı eşleyen **herkese** görünür → paylaşımlı bellek ile IPC.
- `MAP_PRIVATE` : yazmalar **COW** ile özel kopyaya gider, dosya değişmez (çalıştırılabilirlerin veri bölümü böyle yüklenir).
- `MAP_ANONYMOUS` : dosyasız bölge (`malloc` büyük bloklar). Windows: `MapViewOfFile` .



mmap: Adım Adım

```
▶ int fd = open("veri.bin", O_RDWR);
▶ struct stat st;
▶ fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4 char *p = mmap(NULL, st.st_size,
5      PROT_READ | PROT_WRITE,
6      MAP_SHARED, fd, 0);
7 close(fd);               /* eşleme sürer */
8 char c = p[0];           /* sayfa 0 */
9 p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



Dosya açılır ve boyutu öğrenilir: 12 KiB, yani 3 sayfa.

Adım 1 / 7

DEĞİŞKENLER

`fd = 3` `p = –` `hata = 0`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
▶  char *p = mmap(NULL, st.st_size,
▶       PROT_READ | PROT_WRITE,
▶       MAP_SHARED, fd, 0);
7  close(fd);               /* eşleme sürer */
8  char c = p[0];           /* sayfa 0 */
9  p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



`mmap` yalnızca bir **VMA** oluşturur: tablo girdileri boş, henüz hiçbir sayfa okunmadı.

Adım 2 / 7

DEĞİŞKENLER

`fd = 3` `p = 0x7f3a…000` `hata = 0`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4  char *p = mmap(NULL, st.st_size,
5       PROT_READ | PROT_WRITE,
6       MAP_SHARED, fd, 0);
▶  close(fd);               /* eşleme sürer */
8  char c = p[0];           /* sayfa 0 */
9  p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



`close(fd)` eşlemeyi bozmaz; VMA dosyaya kendi referansını tutar.

Adım 3 / 7

DEĞİŞKENLER

`fd = kapalı` `p = 0x7f3a…000` `hata = 0`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4  char *p = mmap(NULL, st.st_size,
5       PROT_READ | PROT_WRITE,
6       MAP_SHARED, fd, 0);
7  close(fd);               /* eşleme sürer */
▶  char c = p[0];           /* sayfa 0 */
9  p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



`p[0]` → sayfa hatası; blok 0 sayfa önbelleğine okunur (büyük hata), girdi geçerli olur.

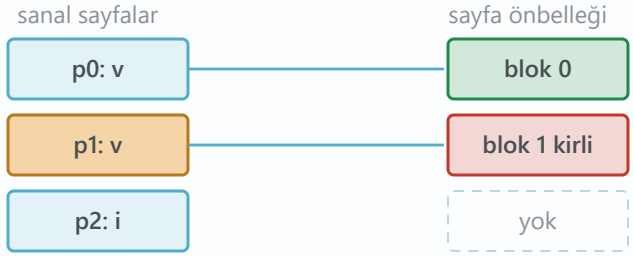
Adım 4 / 7

DEĞİŞKENLER

`c = '…'` `hata = 1`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4  char *p = mmap(NULL, st.st_size,
5       PROT_READ | PROT_WRITE,
6       MAP_SHARED, fd, 0);
7  close(fd);               /* eşleme sürer */
8  char c = p[0];           /* sayfa 0 */
▶  p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



`p[5000]` : 5000 / 4096 → sayfa 1; hata, okuma, yazma → sayfa **kirli** (M=1).

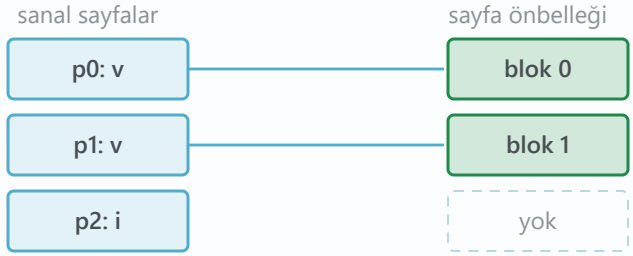
Adım 5 / 7

DEĞİŞKENLER

`c = '…'` `hata = 2`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4  char *p = mmap(NULL, st.st_size,
5       PROT_READ | PROT_WRITE,
6       MAP_SHARED, fd, 0);
7  close(fd);               /* eşleme sürer */
8  char c = p[0];           /* sayfa 0 */
9  p[5000] = 'x';           /* sayfa 1 */
▶  msync(p, st.st_size, MS_SYNC);
11 munmap(p, st.st_size);
```



`msync` kirli sayfayı dosyaya yazar; M temizlenir. Sayfa 2'ye hiç dokunulmadı, hiç okunmadı.

Adım 6 / 7

DEĞİŞKENLER

`hata = 2` `yazılan = 1 sayfa`

mmap: Adım Adım

```
1  int fd = open("veri.bin", O_RDWR);
2  struct stat st;
3  fstat(fd, &st);          /* 12 KiB = 3 sayfa */
4  char *p = mmap(NULL, st.st_size,
5       PROT_READ | PROT_WRITE,
6       MAP_SHARED, fd, 0);
7  close(fd);               /* eşleme sürer */
8  char c = p[0];           /* sayfa 0 */
9  p[5000] = 'x';           /* sayfa 1 */
10 msync(p, st.st_size, MS_SYNC);
▶  munmap(p, st.st_size);
```



`munmap` VMA'yı kaldırır; sayfalar önbellekte kalabilir, başka süreç dosyayı açarsa küçük hata ile kullanır.

Adım 7 / 7

DEĞİŞKENLER

`p = –` `hata = 2`

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.



Sayfa hatası: istenen sayfa diskte bulundu ama **boş çerçeve yok**.

Adım 1 / 6

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.



1. Yer değiştirme algoritması bir **kurban** çerçeve seçer.

Adım 2 / 6

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.



2. Kurban **kirliyse** (M = 1) takasa/dosyaya yazılır; temizse yazmaya gerek yok: G/Ç yarıya iner.

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.

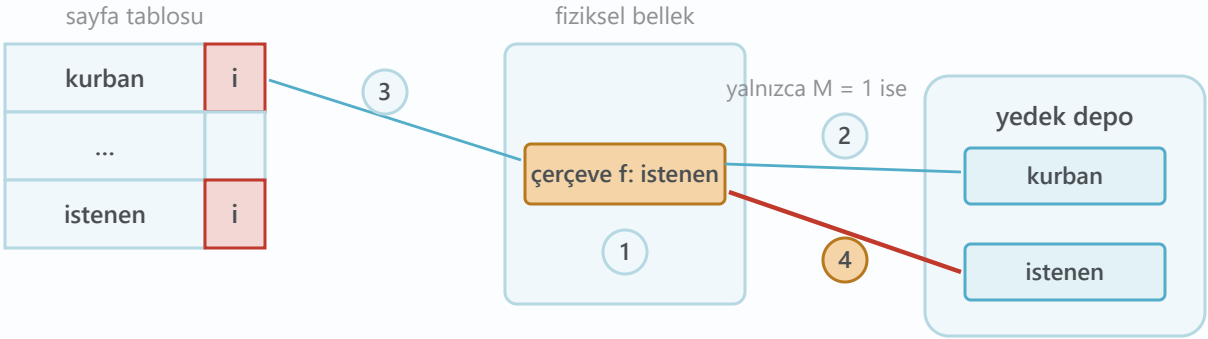


3. Kurbanın girdisi `i` yapılır (ve TLB'den silinir: çok çekirdekte *TLB shootdown*).

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.

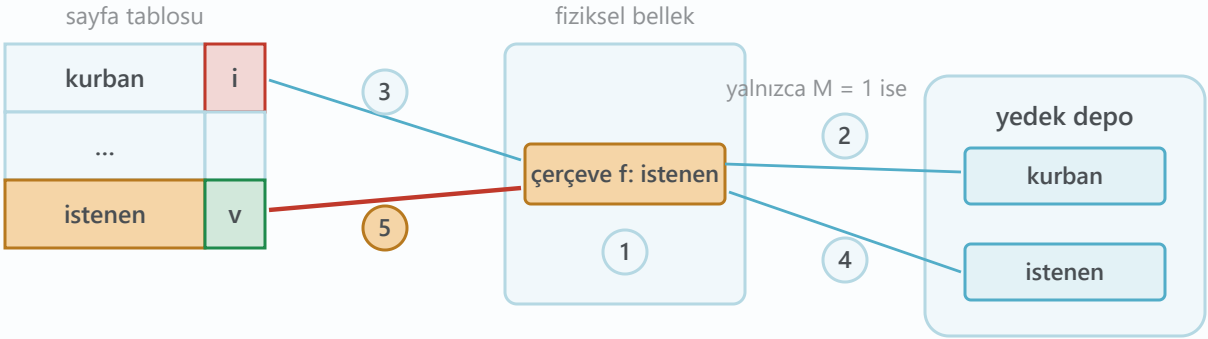


4. İstenen sayfa boşalan çerçeveye okunur.

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Boş Çerçeve Yoksa: Sayfa Yer Değiştirme

- Bellek çoğunlukla **doludur** (boş RAM boşa harcanmış RAM'dir). Seçenekler: süreci sonlandırmak (kabul edilemez), bütün süreci takaslamak ya da **tek bir sayfayı** değiştirmek.



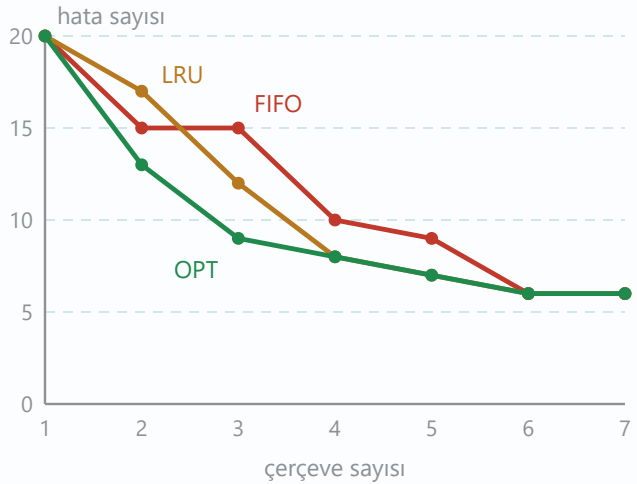
5. Yeni girdi `v` yapılır; tuzağa yol açan komut yeniden başlatılır.

İki karar var: *çerçeve tahsisi* (sürece kaç çerçeve?) ve *yer değiştirme* (hangi sayfa çıkar?). Amaç: en az sayfa hatası; aynı sayfanın tekrar tekrar gidip gelmesini önlemek.

Algoritmaları Değerlendirmek

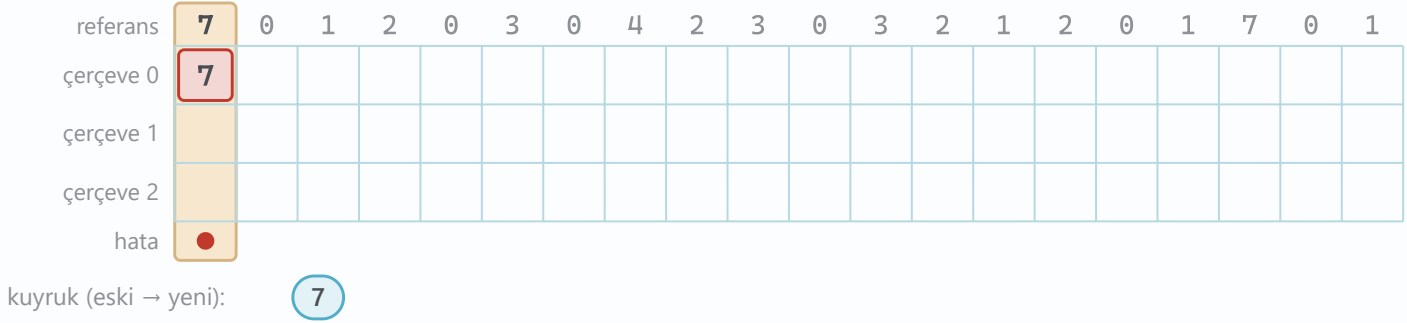
- Algoritma bir **referans dizisi** (yalnızca sayfa numaraları) üzerinde çalıştırılır, **sayfa hatası** sayılır.
- Adres izinden elde edilir; ardışık aynı sayfalar tek sayılır. 100 baytlık sayfalarla `0100 0432 0101 0612 0102 …` → `1 4 1 6 1 6 1 6 1 6 1`.
- Bu bölümün **ortak dizisi** (Silberschatz), 3 çerçeve:

  `7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1`
- Beklenen: çerçeve sayısı arttıkça hata **azalır**; ilk yüklemeler (burada 6 farklı sayfa: 7, 0, 1, 2, 3, 4) kaçınılmazdır.
- Algoritmalar: OPT, FIFO, LRU, ikinci şans, saat, NRU, NFU/yaşlandırma, çalışma kümesi, WSClock.



FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.



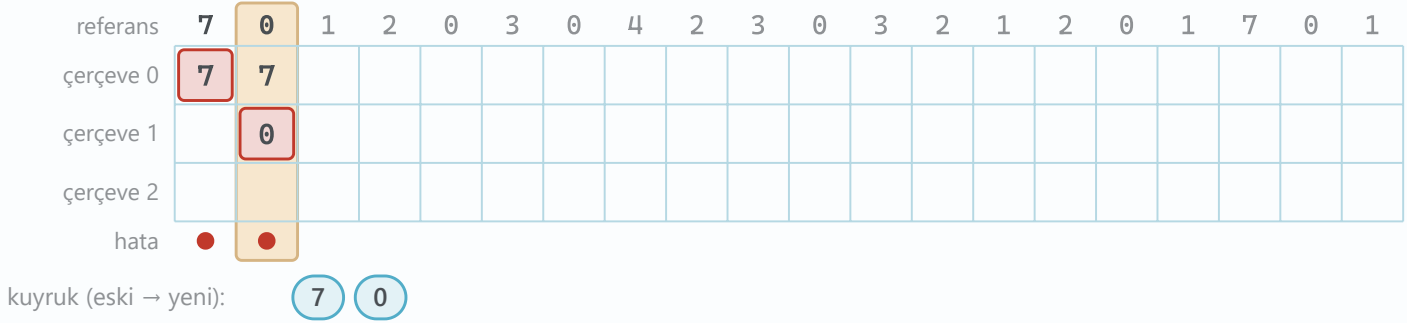
1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

Adım 1 / 20

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

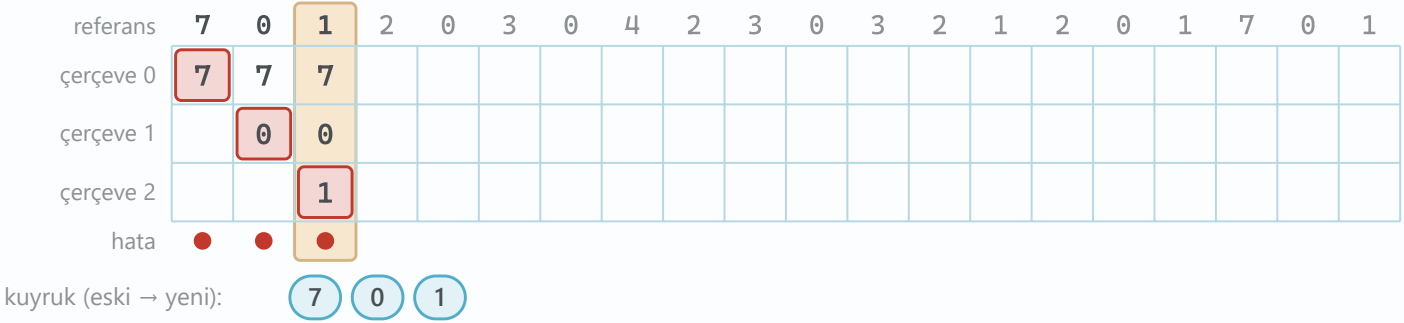


2. referans 0: hata; boş çerçeve 1'e yüklenir. Hata: 2

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

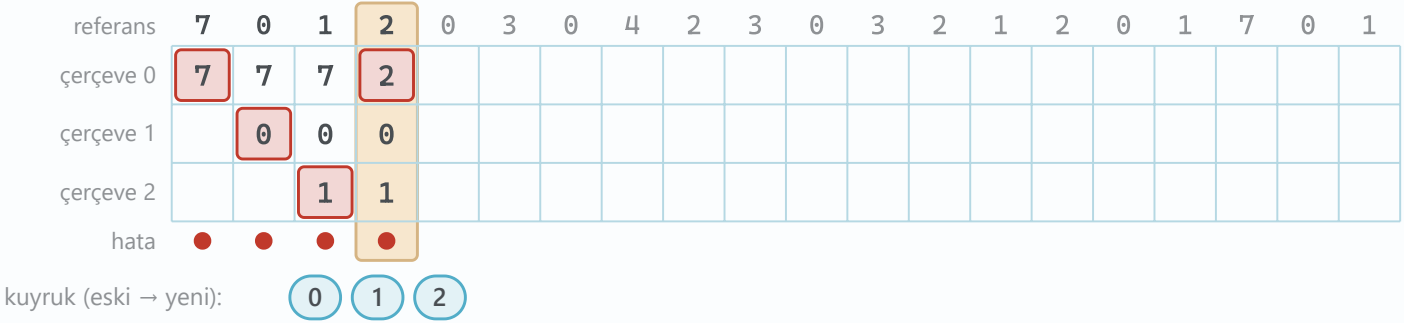


3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

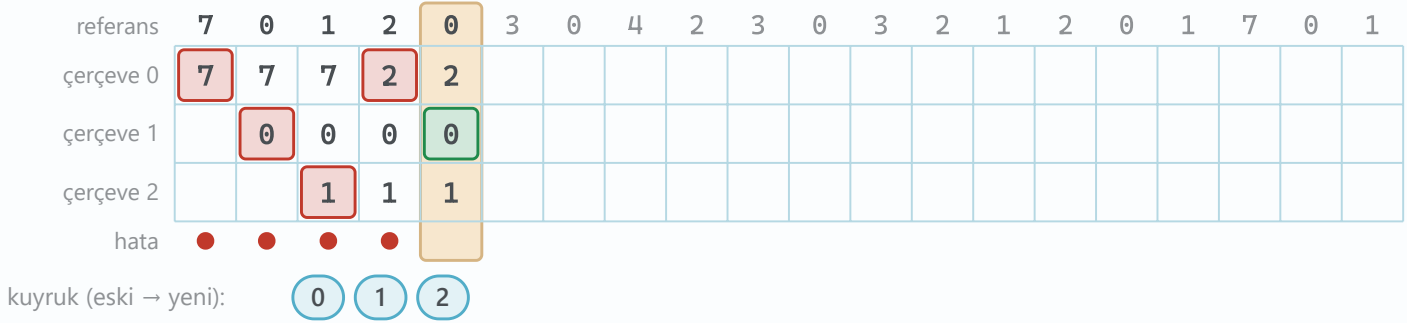


4. referans 2: hata; kuyruğun başındaki en eski sayfa **7** çıkar, 2 sona girer. Hata: 4

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

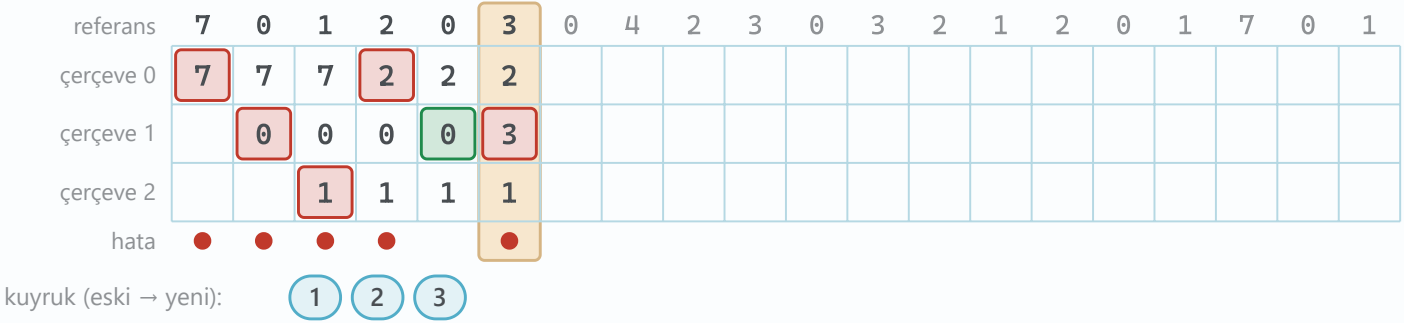


5. referans 0: bellekte → isabet; FIFO sırası değişmez.

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

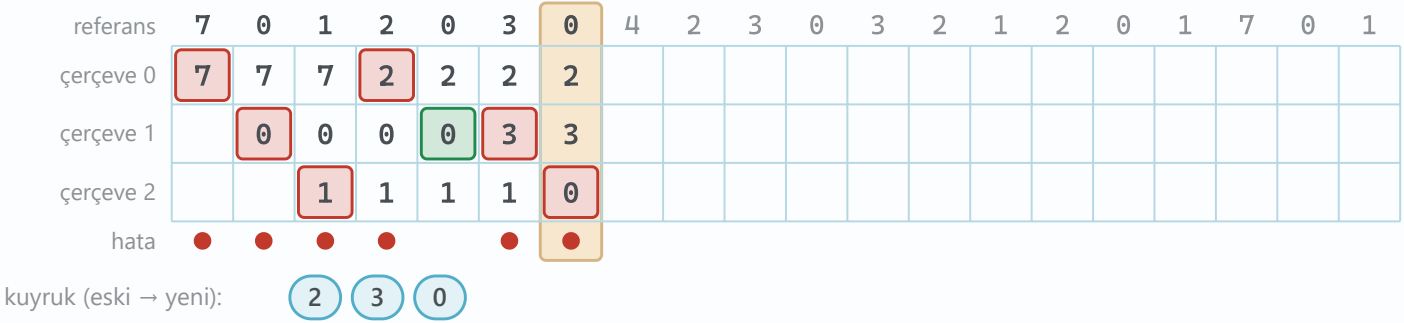


6. referans 3: hata; kuyruğun başındaki en eski sayfa **0** çıkar, 3 sona girer. Hata: 5

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

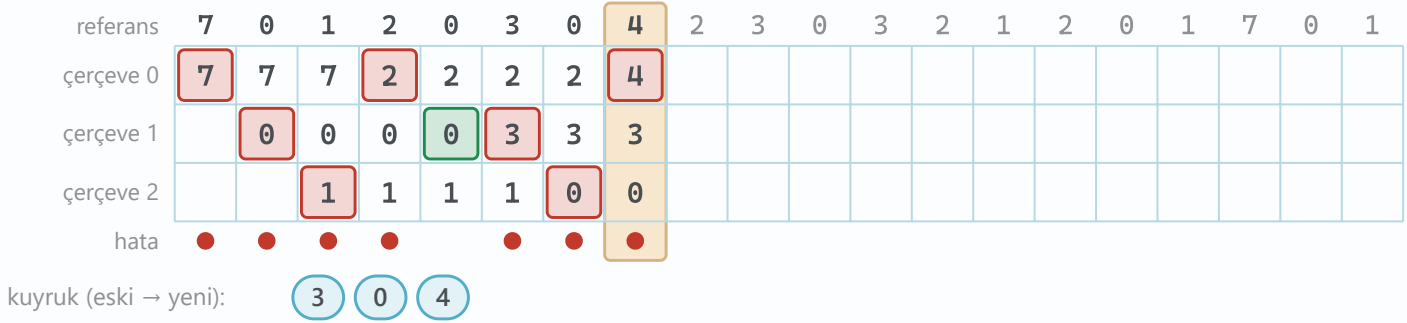


7. referans 0: hata; kuyruğun başındaki en eski sayfa **1** çıkar, 0 sona girer. Hata: 6

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

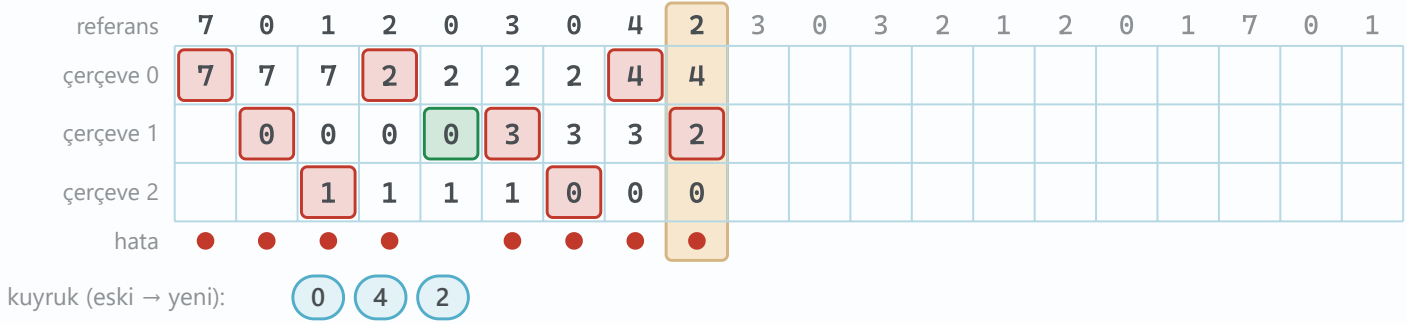


8. referans 4: hata; kuyruğun başındaki en eski sayfa **2** çıkar, 4 sona girer. Hata: 7

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

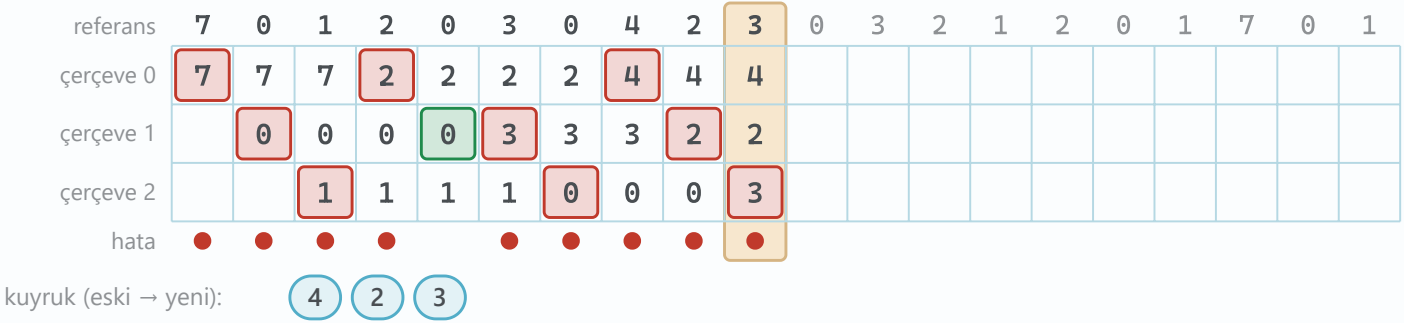


9. referans 2: hata; kuyruğun başındaki en eski sayfa **3** çıkar, 2 sona girer. Hata: 8

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

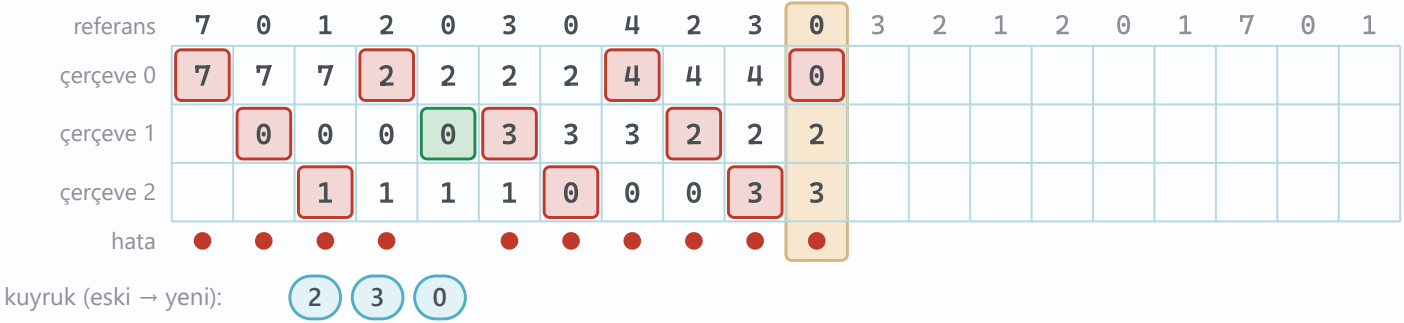


10. referans 3: hata; kuyruğun başındaki en eski sayfa **0** çıkar, 3 sona girer. Hata: 9

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

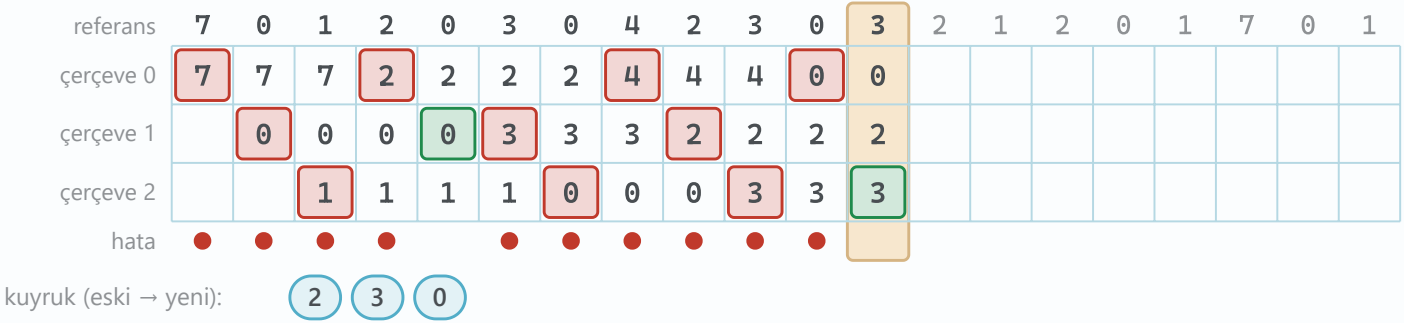


11. referans 0: hata; kuyruğun başındaki en eski sayfa **4** çıkar, 0 sona girer. Hata: 10

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

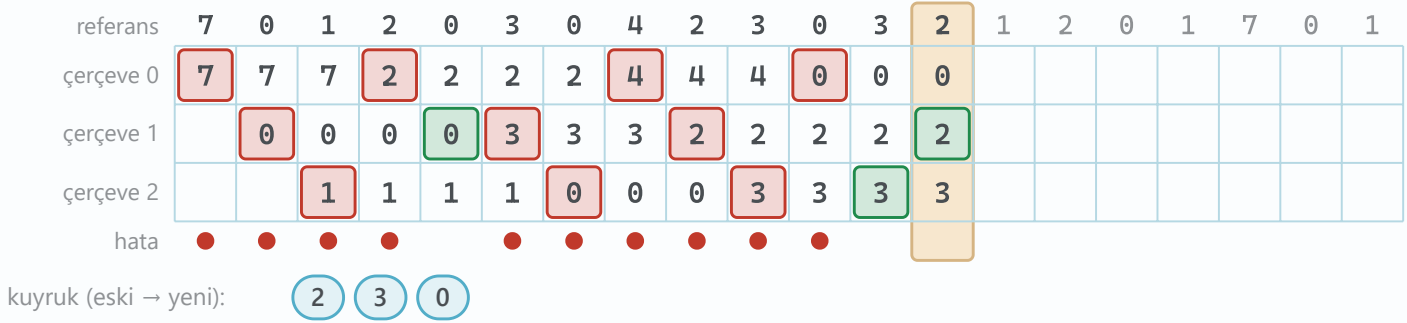


12. referans 3: bellekte → isabet; FIFO sırası değişmez.

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

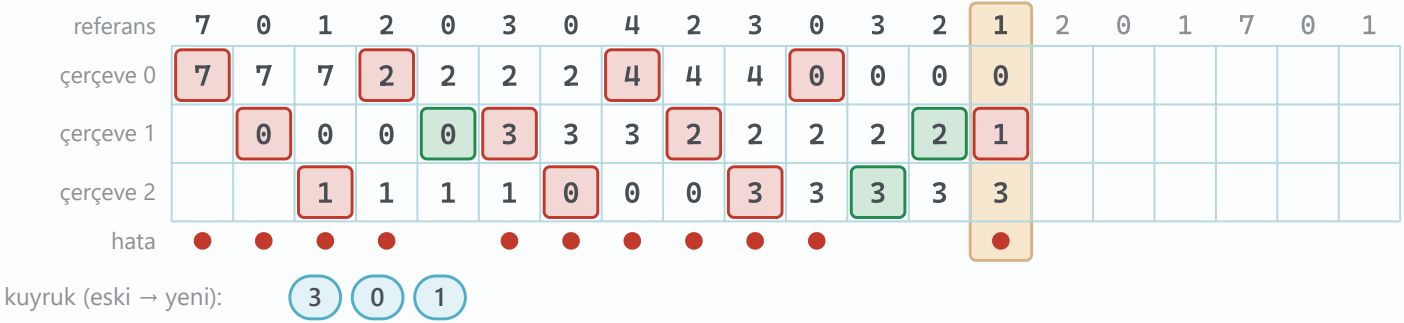


13. referans 2: bellekte → isabet; FIFO sırası değişmez.

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

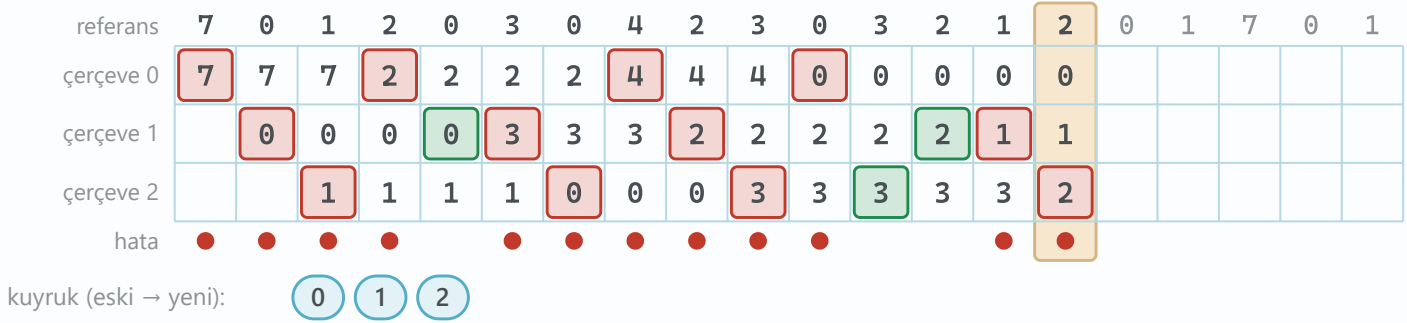


14. referans 1: hata; kuyruğun başındaki en eski sayfa **2** çıkar, 1 sona girer. Hata: 11

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

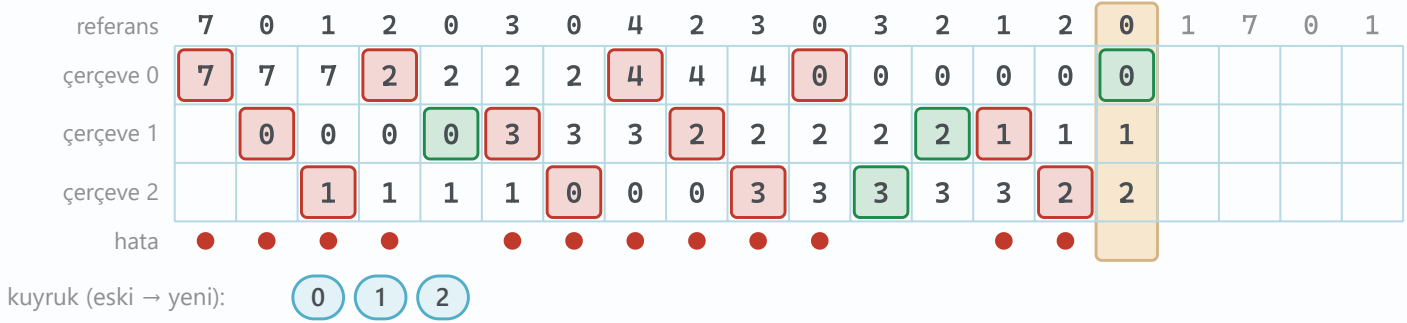


15. referans 2: hata; kuyruğun başındaki en eski sayfa **3** çıkar, 2 sona girer. Hata: 12

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

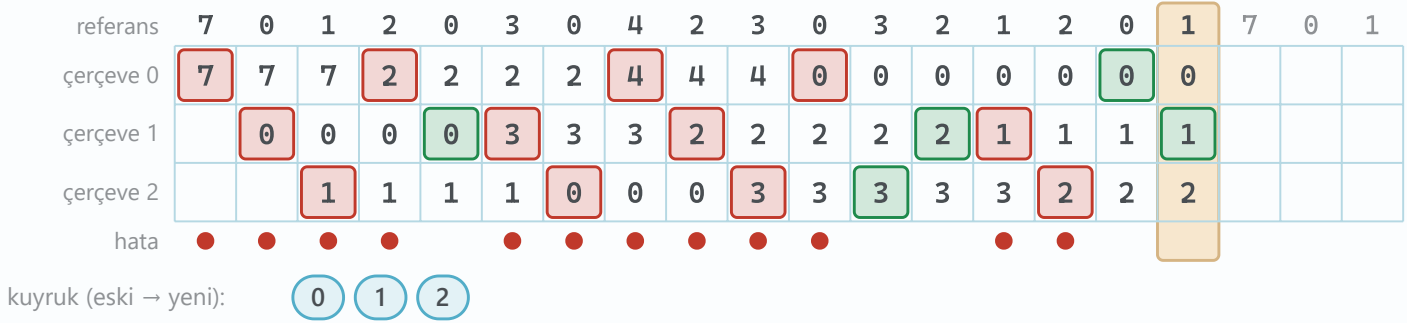


16. referans 0: bellekte → isabet; FIFO sırası değişmez.

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

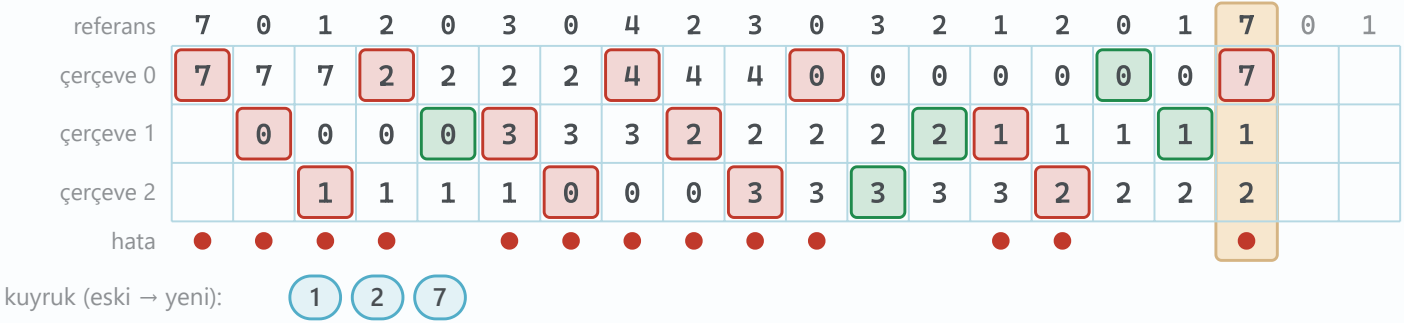


17. referans 1: bellekte → isabet; FIFO sırası değişmez.

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

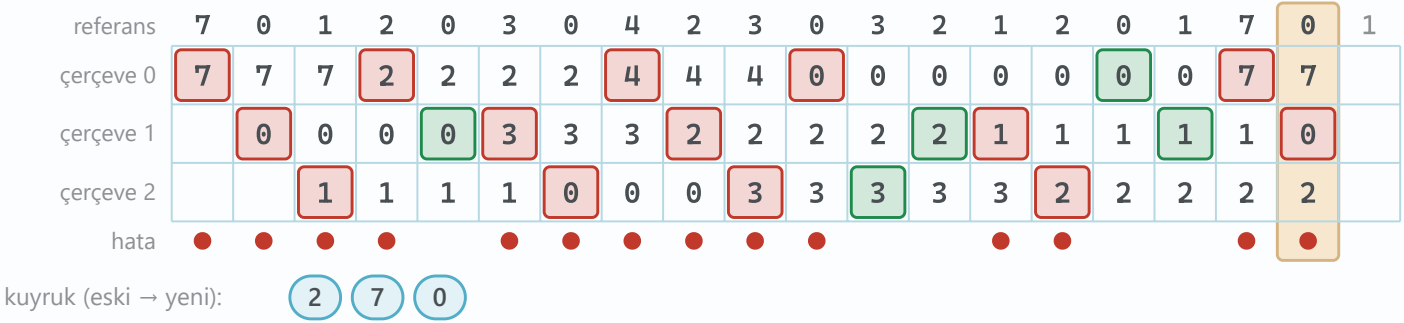


18. referans 7: hata; kuyruğun başındaki en eski sayfa **0** çıkar, 7 sona girer. Hata: 13

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.

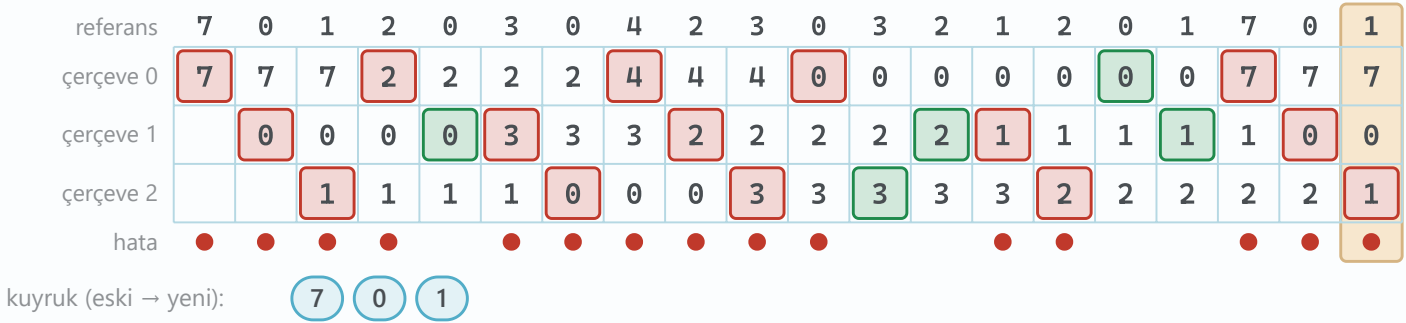


19. referans 0: hata; kuyruğun başındaki en eski sayfa **1** çıkar, 0 sona girer. Hata: 14

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

FIFO: İlk Giren İlk Çıkar

- Sayfalar **yüklenme** sırasına göre bir kuyrukta tutulur; en son *yüklenen* sondadır.
- Yer gerekince kuyruğun başındaki **en eski** sayfa çıkar.
- Uygulaması kolay, bit gerektirmez.
- Kullanımı hiç dikkate almaz: en eski sayfa en yoğun kullanılan sayfa olabilir.



20. referans 1: hata; kuyruğun başındaki en eski sayfa **2** çıkar, 1 sona girer. Hata: 15 **Toplam 15 hata** (3 ilk yükleme + 12 yer değiştirme).

Kırmızı hücre: hatayla yüklenen sayfa · yeşil: isabet · kırmızı nokta: sayfa hatası. FIFO erişimi değil yüklenmeyi izler: listenin sonunda en son *yüklenen* sayfa durur.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

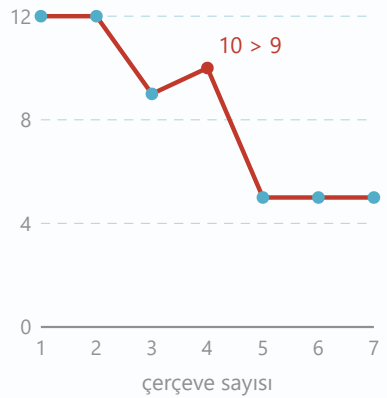
3 çerçeve — hata: 1

	1	2	3	4	1	2	5	1	2	3	4	5
referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1											
çerçeve 1												
çerçeve 2												
hata	●											

4 çerçeve — hata: 1

	1	2	3	4	1	2	5	1	2	3	4	5
referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1											
çerçeve 1												
çerçeve 2												
çerçeve 3												
hata	●											

1. referans 1 — 3 çerçeve: hata · 4 çerçeve: hata.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

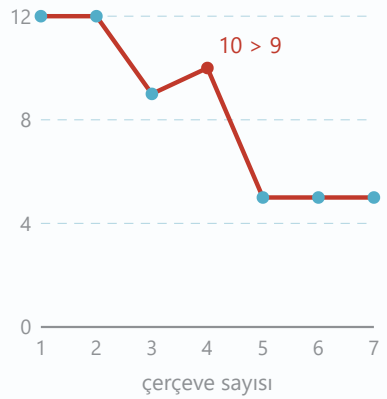
3 çerçeve — hata: 2

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1										
çerçeve 1		2										
çerçeve 2												
hata	●	●										

4 çerçeve — hata: 2

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1										
çerçeve 1		2										
çerçeve 2												
çerçeve 3												
hata	●	●										

2. referans 2 — 3 çerçeve: hata · 4 çerçeve: hata.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

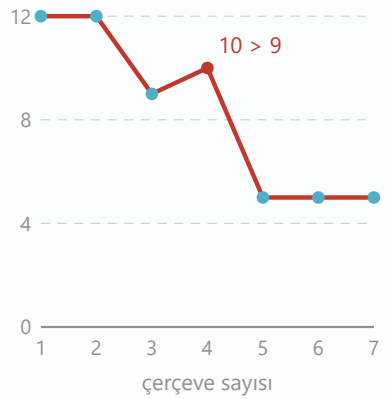
3 çerçeve — hata: 3

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1									
çerçeve 1		2	2									
çerçeve 2			3									
hata	●	●	●									

4 çerçeve — hata: 3

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1									
çerçeve 1		2	2									
çerçeve 2			3									
çerçeve 3												
hata	●	●	●									

3. referans 3 — 3 çerçeve: hata · 4 çerçeve: hata.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

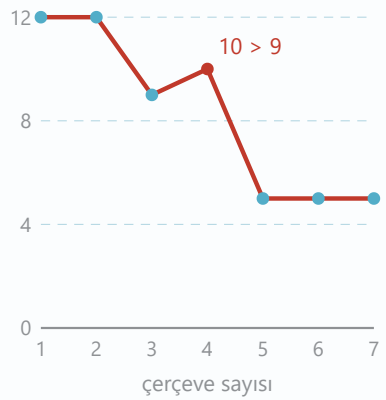
3 çerçeve — hata: 4

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4								
çerçeve 1		2	2	2								
çerçeve 2			3	3								
hata	●	●	●	●								

4 çerçeve — hata: 4

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1								
çerçeve 1		2	2	2								
çerçeve 2			3	3								
çerçeve 3				4								
hata	●	●	●	●								

4. referans 4 — 3 çerçeve: hata (1 çıkar) · 4 çerçeve: hata.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

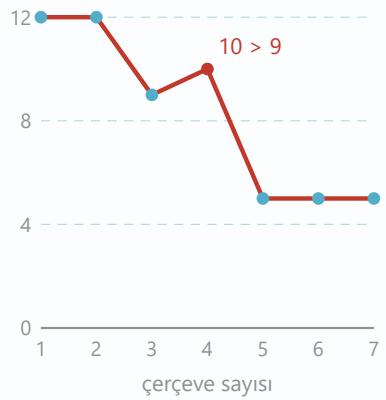
3 çerçeve — hata: 5

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4							
çerçeve 1		2	2	2	1							
çerçeve 2			3	3	3							
hata	●	●	●	●	●							

4 çerçeve — hata: 4

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1							
çerçeve 1		2	2	2	2							
çerçeve 2			3	3	3							
çerçeve 3				4	4							
hata	●	●	●	●								

5. referans 1 — 3 çerçeve: hata (2 çıkar) · 4 çerçeve: isabet.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

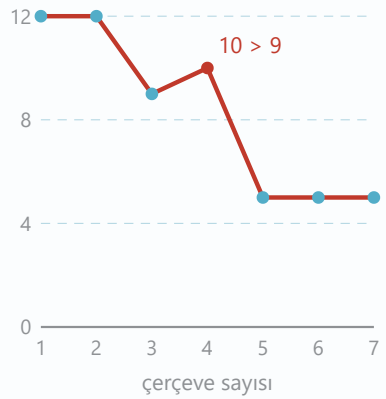
3 çerçeve — hata: 6

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4						
çerçeve 1		2	2	2	1	1						
çerçeve 2			3	3	3	2						
hata	●	●	●	●	●	●						

4 çerçeve — hata: 4

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1						
çerçeve 1		2	2	2	2	2						
çerçeve 2			3	3	3	3						
çerçeve 3				4	4	4						
hata	●	●	●	●								

6. referans 2 — 3 çerçeve: hata (3 çıkar) · 4 çerçeve: isabet.



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

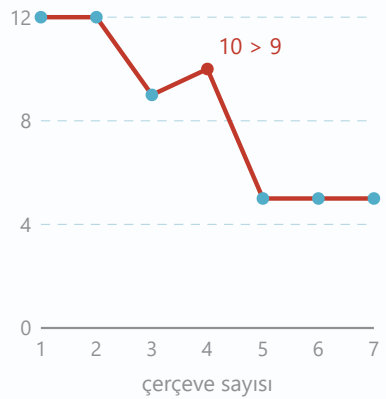
3 çerçeve — hata: 7

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5					
çerçeve 1		2	2	2	1	1	1					
çerçeve 2			3	3	3	2	2					
hata	●	●	●	●	●	●	●					

4 çerçeve — hata: 5

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5					
çerçeve 1		2	2	2	2	2	2					
çerçeve 2			3	3	3	3	3					
çerçeve 3				4	4	4	4					
hata	●	●	●	●			●					

7. referans 5 — 3 çerçeve: hata (4 çıkar) · 4 çerçeve: hata (1 çıkar).



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

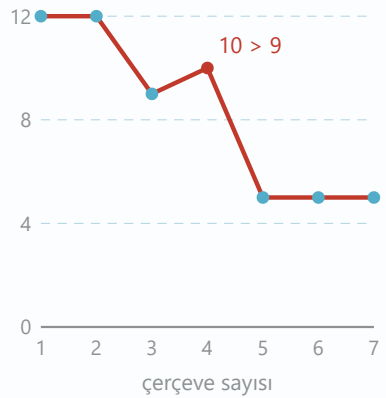
3 çerçeve — hata: 7

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5	5				
çerçeve 1		2	2	2	1	1	1	1				
çerçeve 2			3	3	3	2	2	2				
hata	●	●	●	●	●	●	●					

4 çerçeve — hata: 6

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5	5				
çerçeve 1		2	2	2	2	2	2	1				
çerçeve 2			3	3	3	3	3	3				
çerçeve 3				4	4	4	4	4				
hata	●	●	●	●			●	●				

8. referans 1 — 3 çerçeve: isabet · 4 çerçeve: hata (2 çıkar).



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

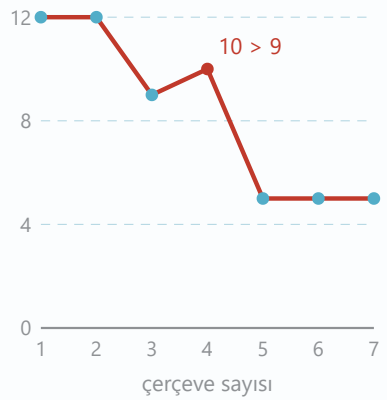
3 çerçeve — hata: 7

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5	5	5			
çerçeve 1		2	2	2	1	1	1	1	1			
çerçeve 2			3	3	3	2	2	2	2			
hata	●	●	●	●	●	●	●					

4 çerçeve — hata: 7

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5	5	5			
çerçeve 1		2	2	2	2	2	2	1	1			
çerçeve 2			3	3	3	3	3	3	2			
çerçeve 3				4	4	4	4	4	4			
hata	●	●	●	●			●	●	●			

9. referans 2 — 3 çerçeve: isabet · 4 çerçeve: hata (3 çıkar).



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

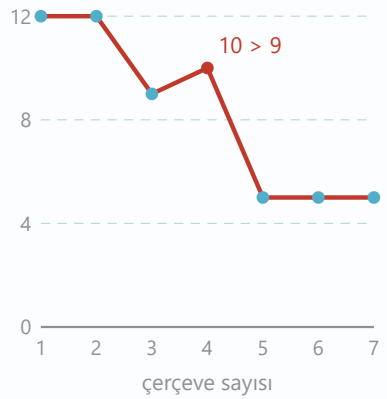
3 çerçeve — hata: 8

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5	5	5	5		
çerçeve 1		2	2	2	1	1	1	1	1	3		
çerçeve 2			3	3	3	2	2	2	2	2		
hata	●	●	●	●	●	●	●			●		

4 çerçeve — hata: 8

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5	5	5	5		
çerçeve 1		2	2	2	2	2	2	1	1	1		
çerçeve 2			3	3	3	3	3	3	2	2		
çerçeve 3				4	4	4	4	4	4	3		
hata	●	●	●	●			●	●	●	●		

10. referans 3 — 3 çerçeve: hata (1 çıkar) · 4 çerçeve: hata (4 çıkar).



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

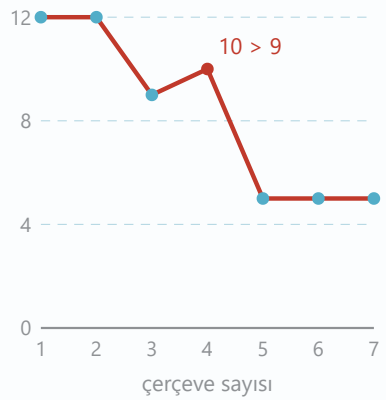
3 çerçeve — hata: 9

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5	5	5	5	5	
çerçeve 1		2	2	2	1	1	1	1	1	3	3	
çerçeve 2			3	3	3	2	2	2	2	2	4	
hata	●	●	●	●	●	●	●			●	●	

4 çerçeve — hata: 9

referans	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5	5	5	5	4	
çerçeve 1		2	2	2	2	2	2	1	1	1	1	
çerçeve 2			3	3	3	3	3	3	2	2	2	
çerçeve 3				4	4	4	4	4	4	3	3	
hata	●	●	●	●			●	●	●	●	●	

11. referans 4 — 3 çerçeve: hata (2 çıkar) · 4 çerçeve: hata (5 çıkar).



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

Belady Anomalisi

Daha çok çerçeve daha az hata getirmeli… FIFO'da her zaman değil. Dizi: `1 2 3 4 1 2 5 1 2 3 4 5`

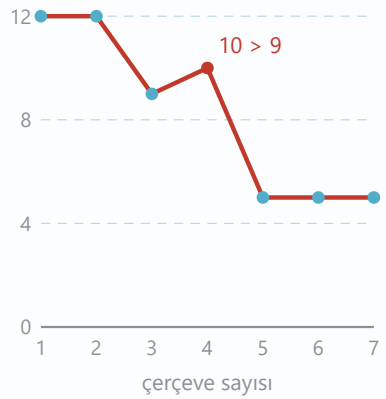
3 çerçeve — hata: 9

	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	4	4	4	5	5	5	5	5	5
çerçeve 1		2	2	2	1	1	1	1	1	3	3	3
çerçeve 2			3	3	3	2	2	2	2	2	4	4
hata	●	●	●	●	●	●	●			●	●	

4 çerçeve — hata: 10

	1	2	3	4	1	2	5	1	2	3	4	5
çerçeve 0	1	1	1	1	1	1	5	5	5	5	4	4
çerçeve 1		2	2	2	2	2	2	1	1	1	1	5
çerçeve 2			3	3	3	3	3	3	2	2	2	2
çerçeve 3				4	4	4	4	4	4	3	3	3
hata	●	●	●	●			●	●	●	●	●	●

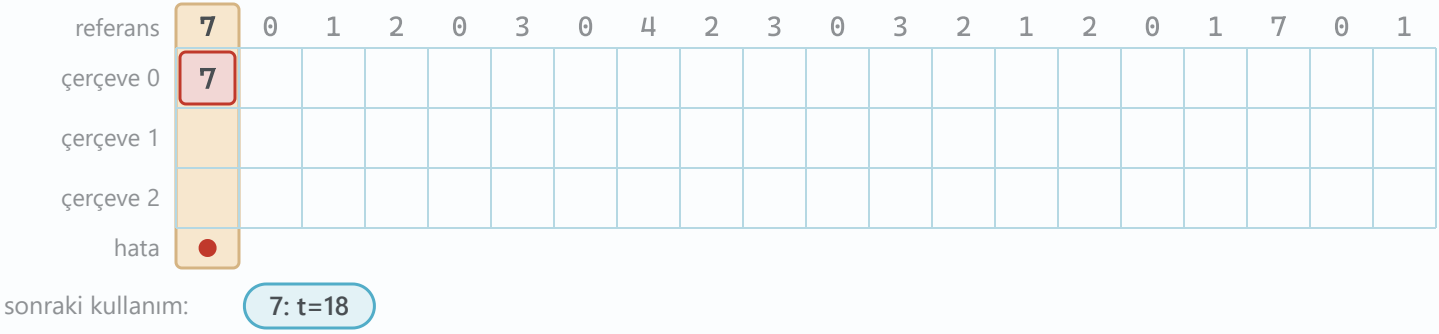
12. referans 5 — 3 çerçeve: isabet · 4 çerçeve: hata (1 çıkar). **Sonuç: 9 < 10.**



Yığın algoritmaları (OPT, LRU): n çerçevedeki sayfa kümesi her zaman n+1 çerçevedekinin alt kümesidir → anomali **olamaz**.

OPT: En Uzun Süre Kullanılmayacak Olan

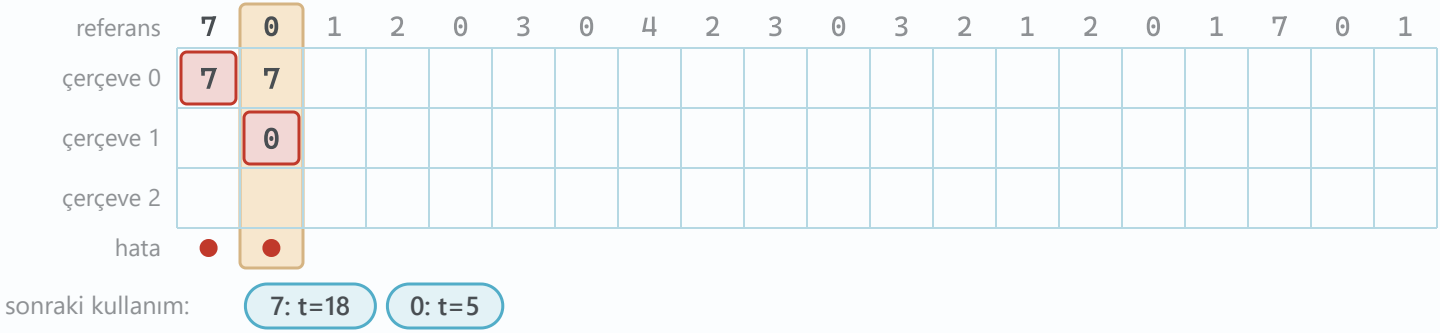
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

OPT: En Uzun Süre Kullanılmayacak Olan

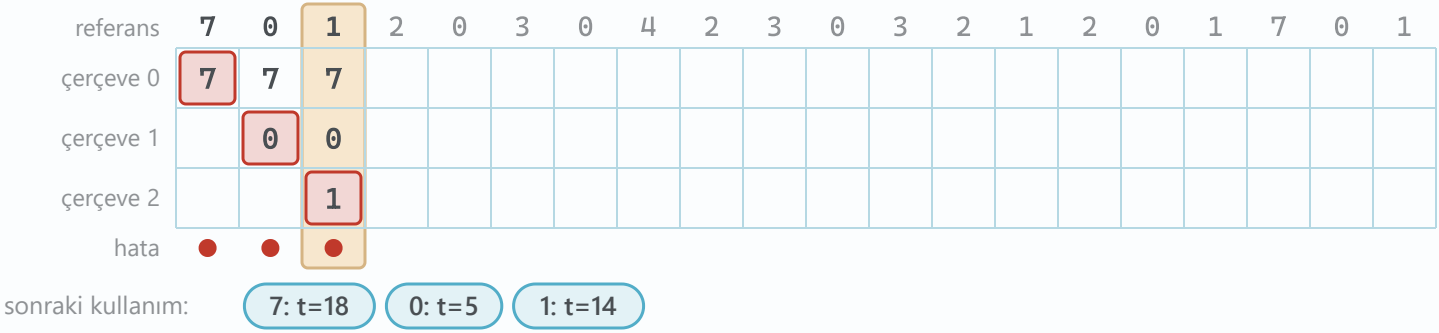
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



2. referans 0: hata; boş çerçeve 1'e yüklenir. Hata: 2

OPT: En Uzun Süre Kullanılmayacak Olan

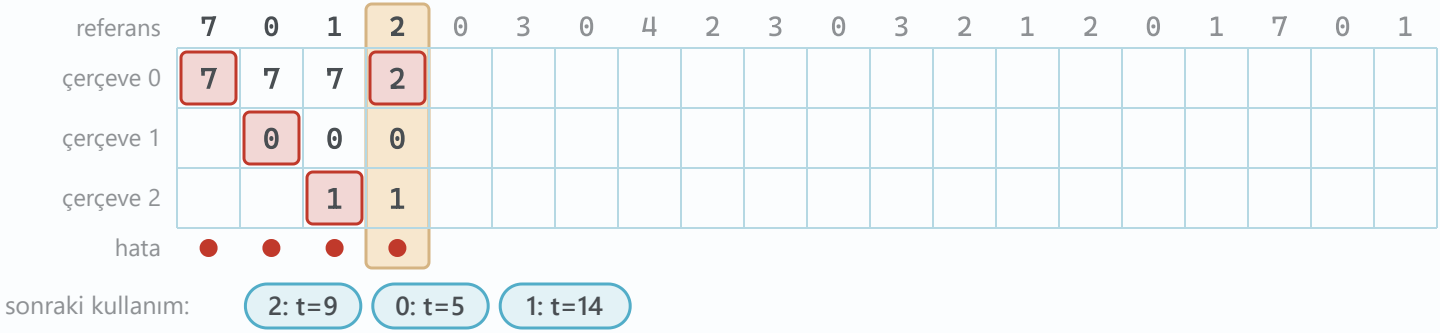
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

OPT: En Uzun Süre Kullanılmayacak Olan

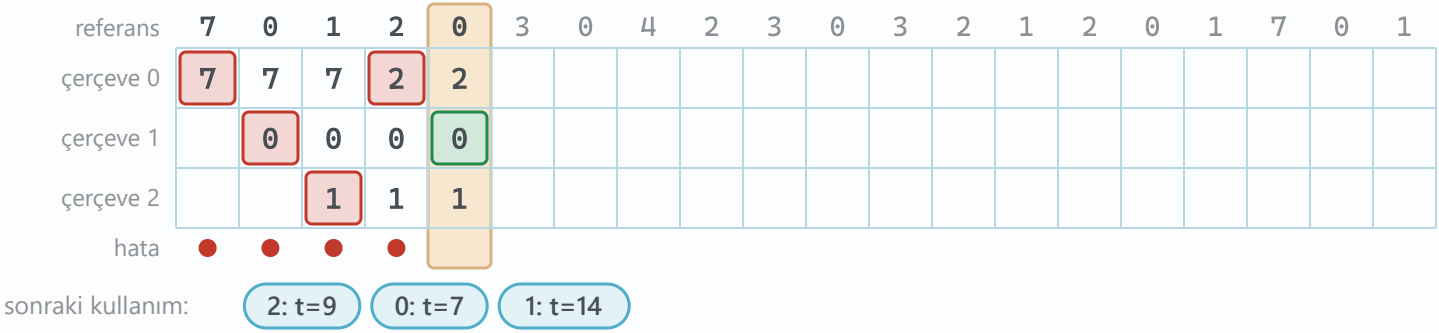
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



4. referans 2: hata; sonraki kullanımlar (7→18, 0→5, 1→14); en uzağa kalan **7** çıkar. Hata: 4

OPT: En Uzun Süre Kullanılmayacak Olan

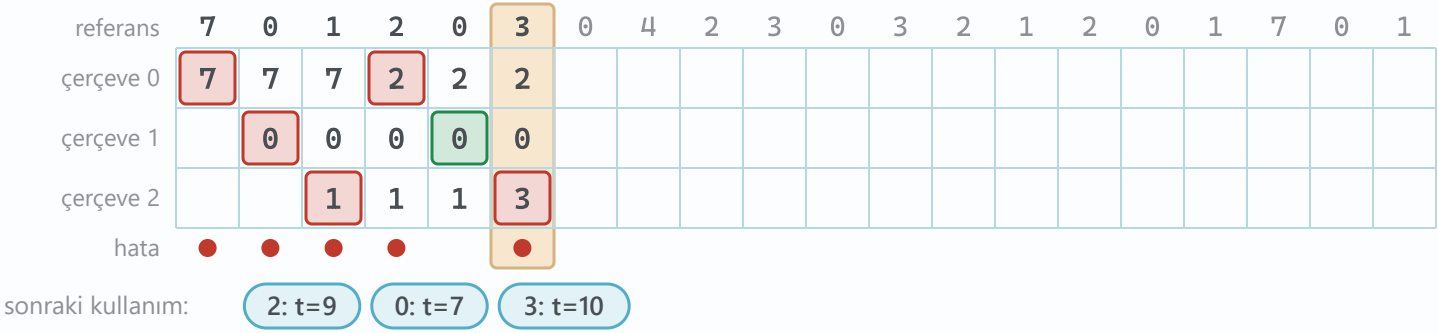
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



5. referans 0: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

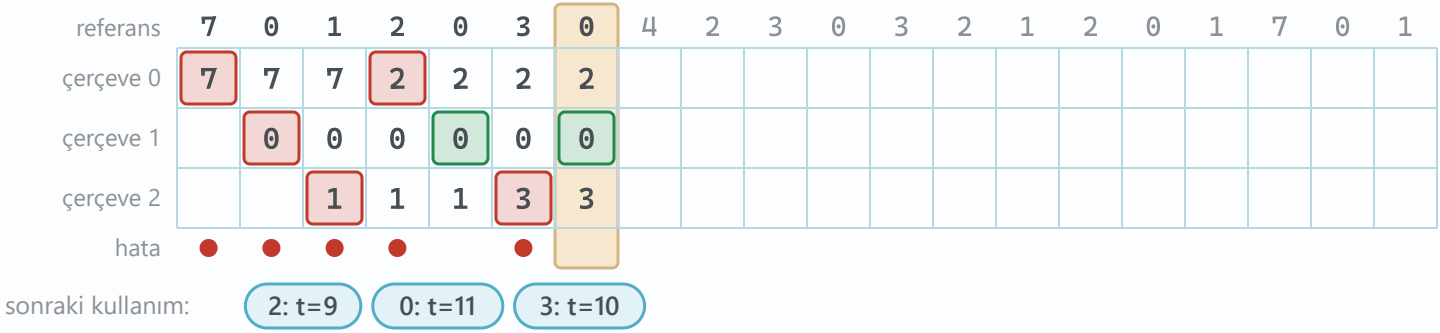
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



6. referans 3: hata; sonraki kullanımlar (2→9, 0→7, 1→14); en uzağa kalan **1** çıkar. Hata: 5

OPT: En Uzun Süre Kullanılmayacak Olan

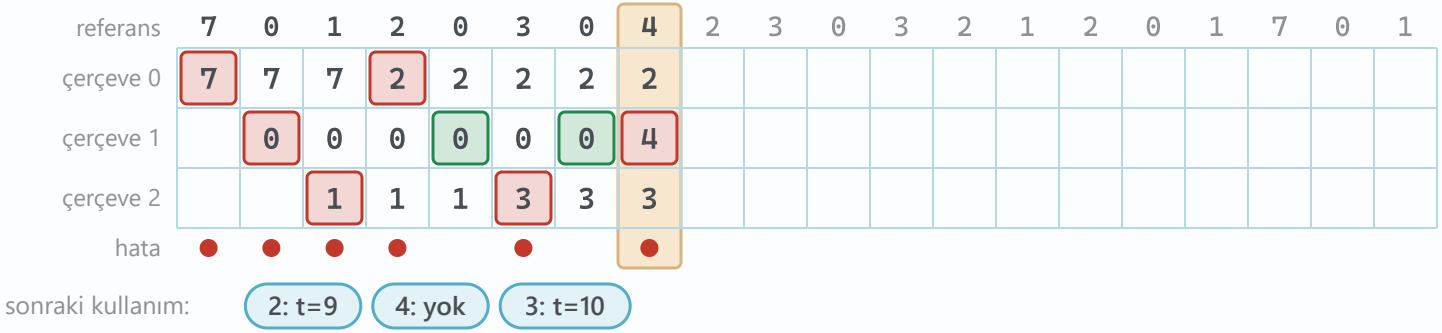
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



7. referans 0: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

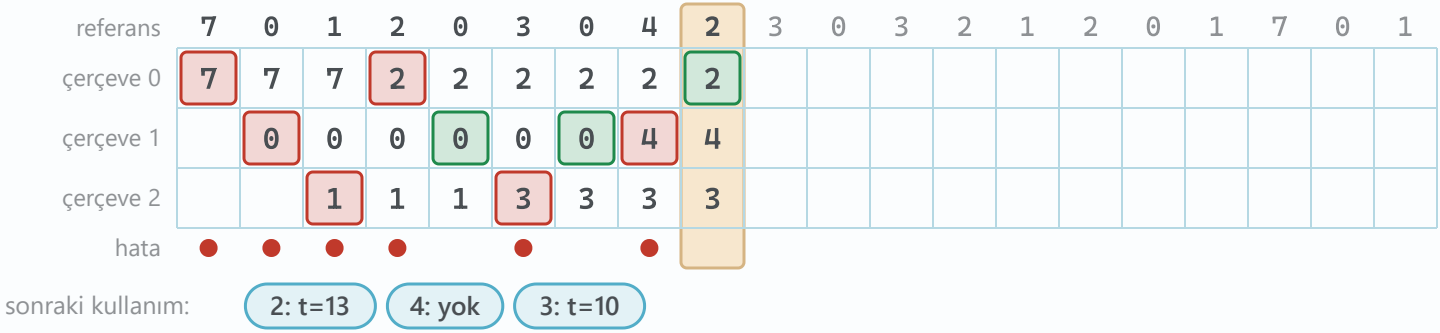
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



8. referans 4: hata; sonraki kullanımlar (2→9, 0→11, 3→10); en uzağa kalan **0** çıkar. Hata: 6

OPT: En Uzun Süre Kullanılmayacak Olan

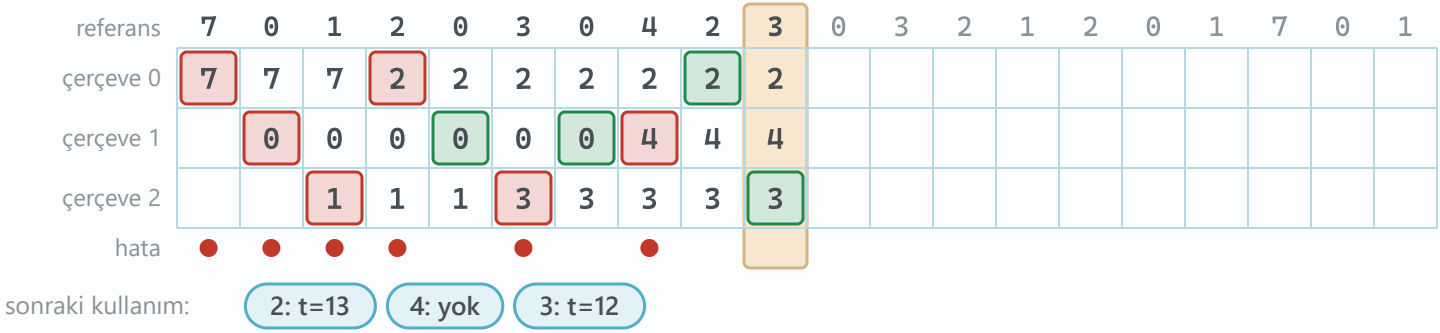
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



9. referans 2: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

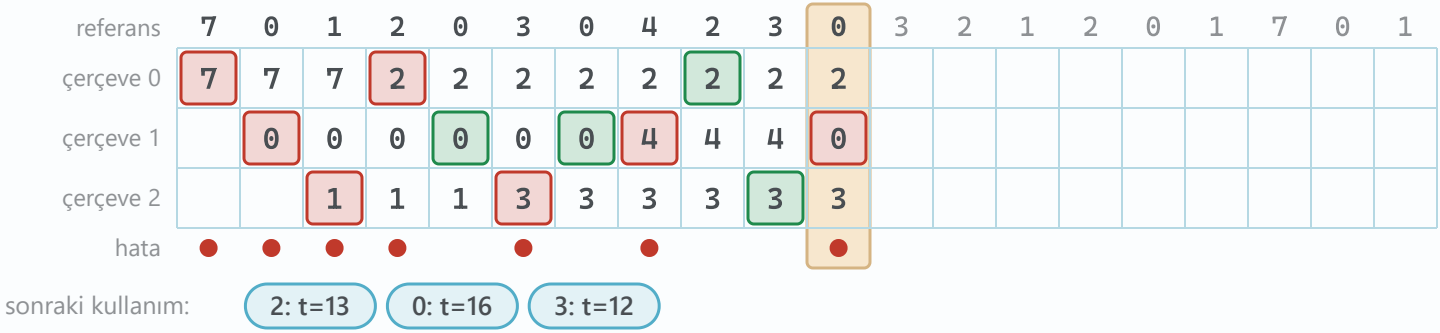
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



10. referans 3: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

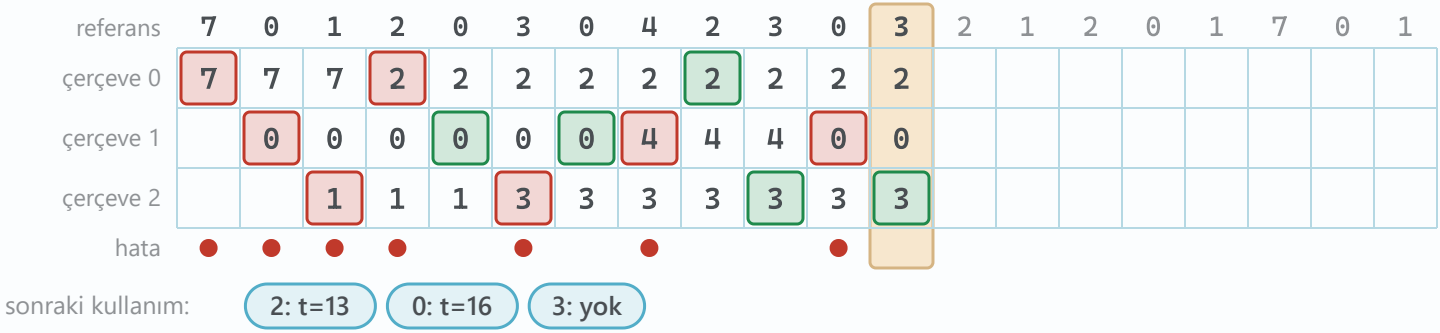
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



11. referans 0: hata; sonraki kullanımlar (2→13, 4→yok, 3→12); en uzağa kalan **4** çıkar. Hata: 7

OPT: En Uzun Süre Kullanılmayacak Olan

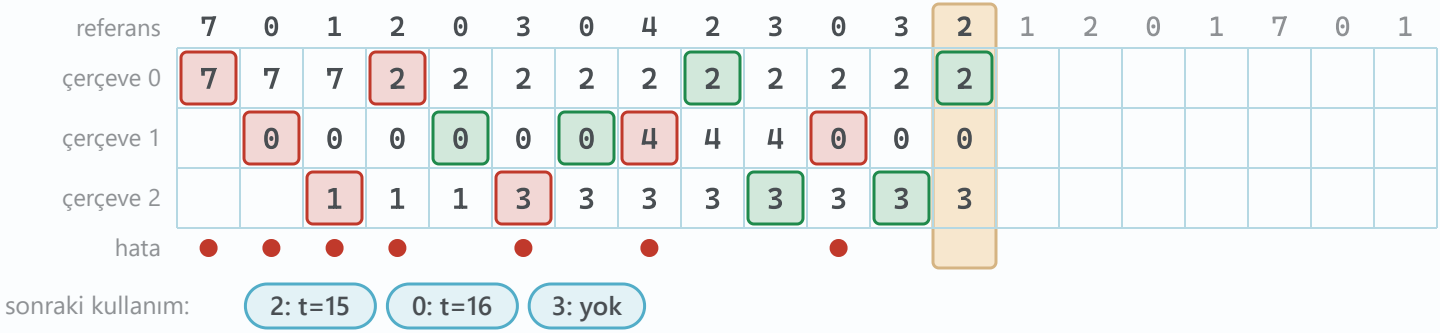
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



12. referans 3: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

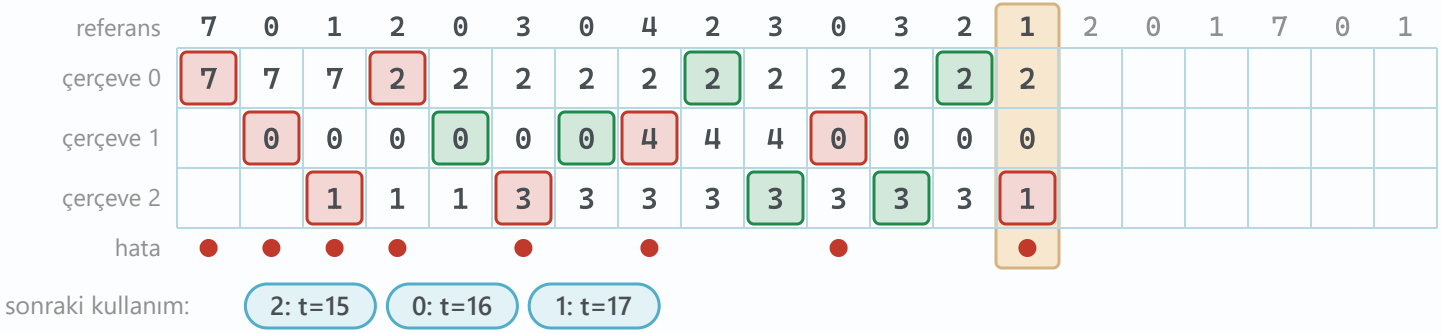
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



13. referans 2: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

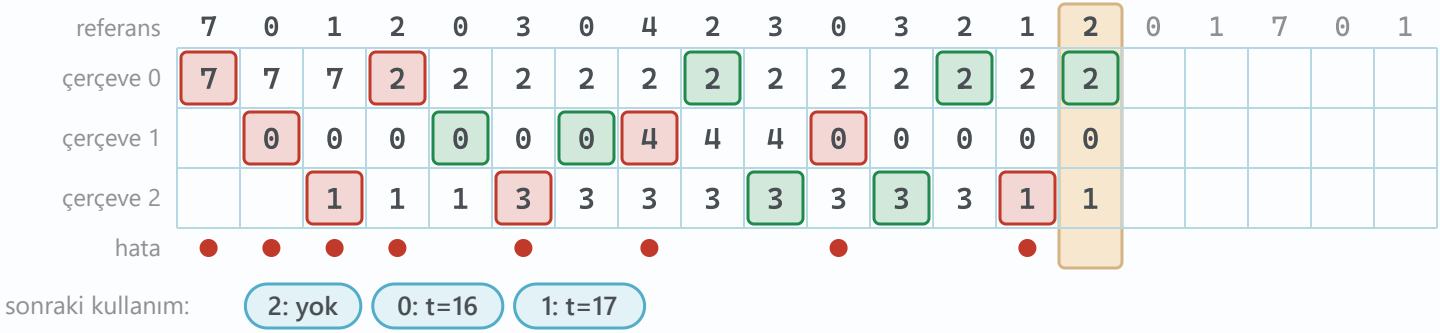
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



14. referans 1: hata; sonraki kullanımlar (2→15, 0→16, 3→yok); en uzağa kalan **3** çıkar. Hata: 8

OPT: En Uzun Süre Kullanılmayacak Olan

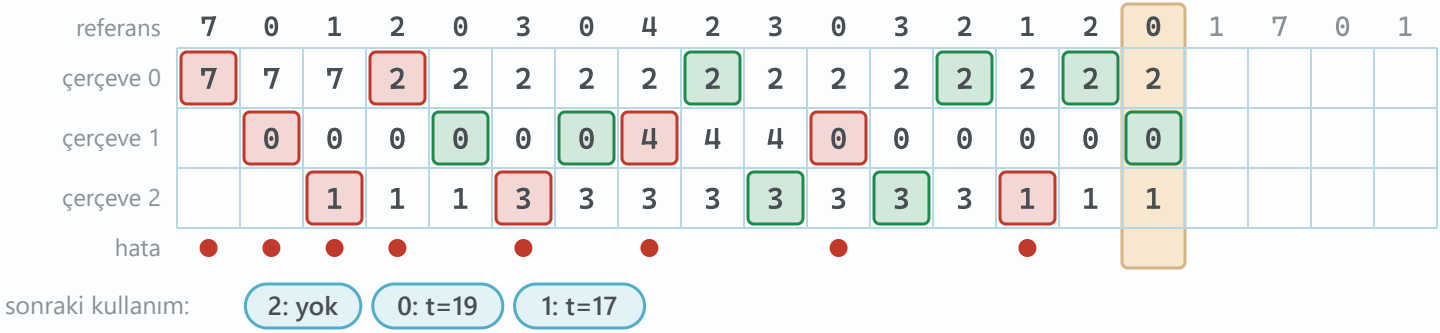
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



15. referans 2: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

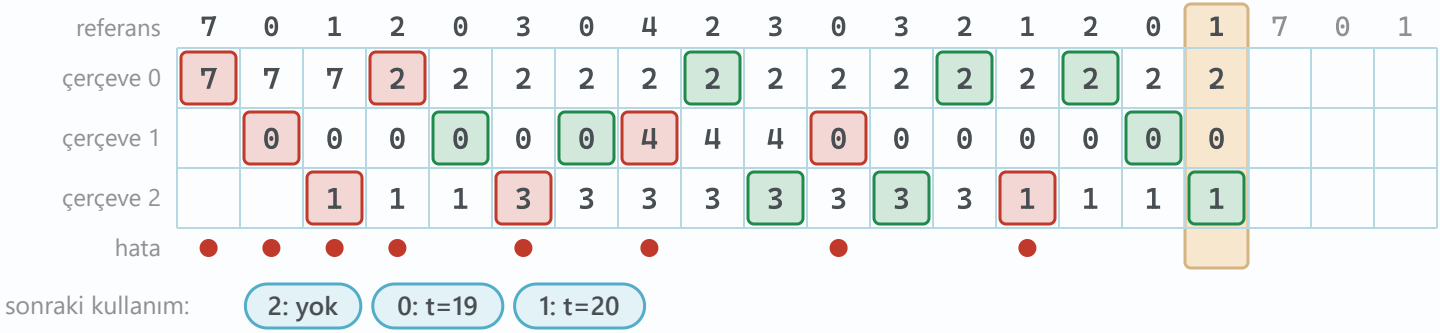
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



16. referans 0: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

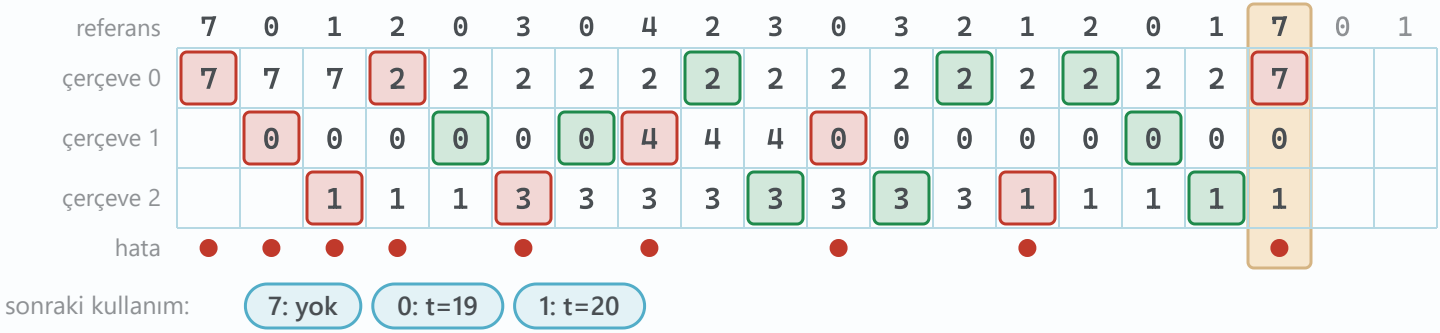
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



17. referans 1: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

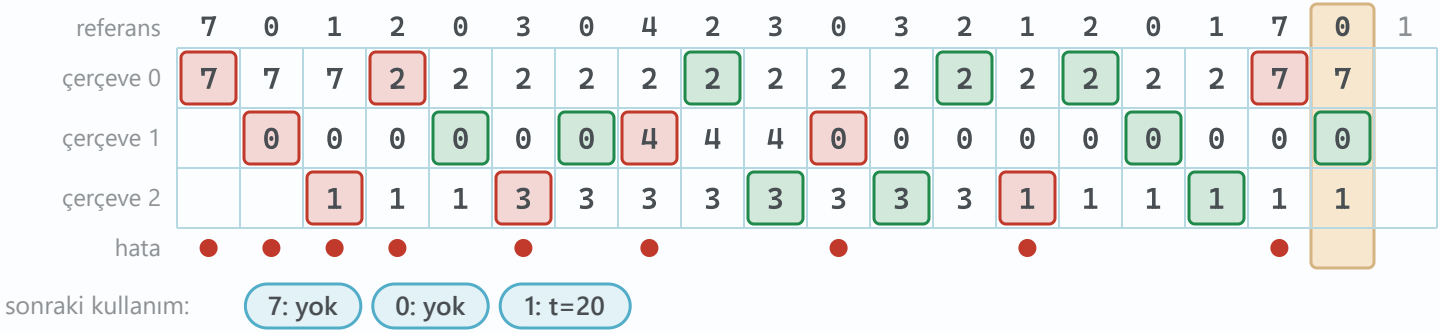
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



18. referans 7: hata; sonraki kullanımlar (2→yok, 0→19, 1→20); en uzağa kalan **2** çıkar. Hata: 9

OPT: En Uzun Süre Kullanılmayacak Olan

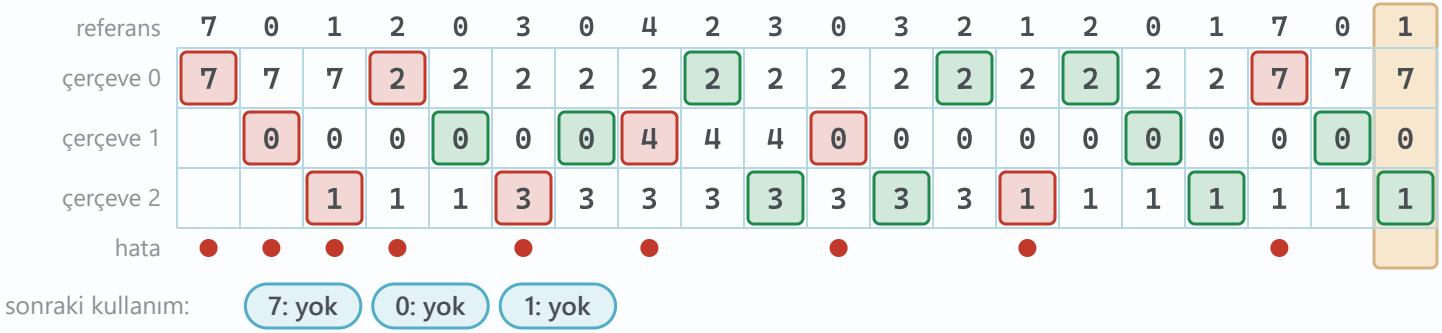
- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



19. referans 0: bellekte → isabet.

OPT: En Uzun Süre Kullanılmayacak Olan

- **Gelecekte en geç** kullanılacak (ya da hiç kullanılmayacak) sayfa çıkar (Belady, 1966).
- Tanımı kolay, uygulanması imkânsız: İS geleceği bilemez (kristal küre).
- Bir **alt sınırdır**: diğer algoritmaları kıyaslamak için kullanılır.
- Gerçek sistemde ancak bir izi iki kez çalıştırarak hesaplanabilir.



20. referans 1: bellekte → isabet. **Toplam 9 hata** (6 yer değiştirme): bu dizi ve 3 çerçeve için alt sınır.

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.

referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7																			
çerçeve 1																				
çerçeve 2																				
hata	●																			

son kullanım (yeni → eski): 7

1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.

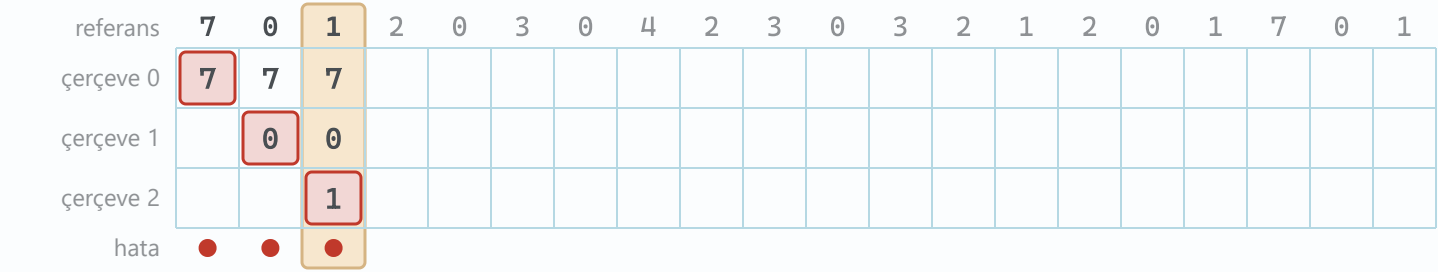
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7																		
çerçeve 1		0																		
çerçeve 2																				
hata	●	●																		

son kullanım (yeni → eski): 0 7

2. referans 0: hata; boş çerçeve 1'e yüklenir. Hata: 2

LRU: En Uzun Süredir Kullanılmayan

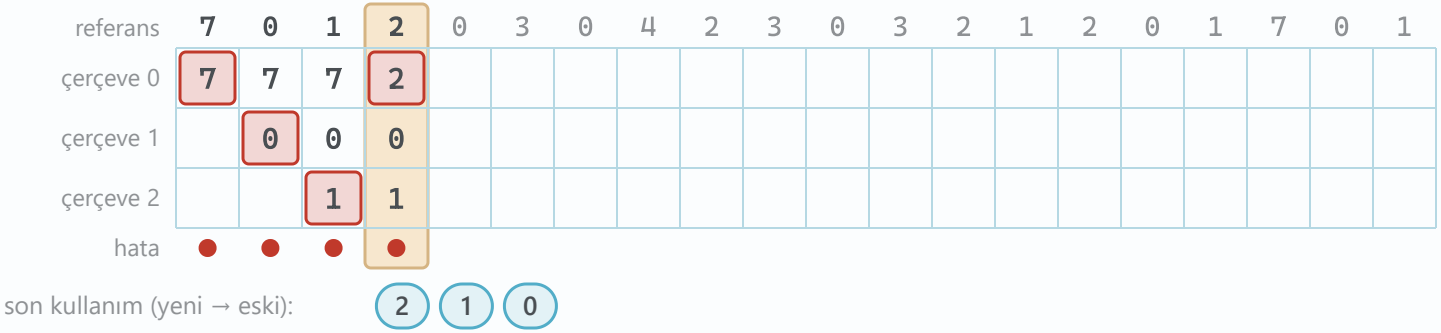
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



4. referans 2: hata; en uzun süredir kullanılmayan **7** çıkar. Hata: 4

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.

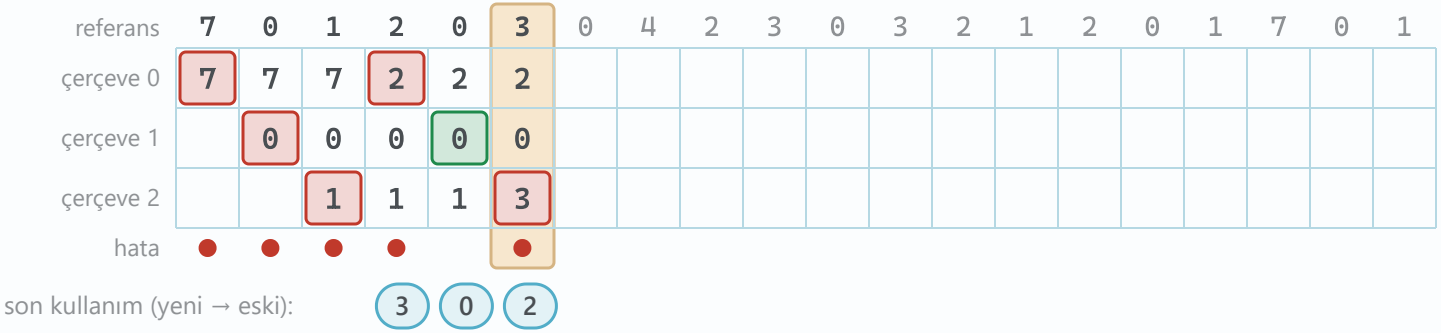
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2															
çerçeve 1		0	0	0	0															
çerçeve 2			1	1	1															
hata	●	●	●	●																

son kullanım (yeni → eski): 0 2 1

5. referans 0: bellekte → isabet; 0 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

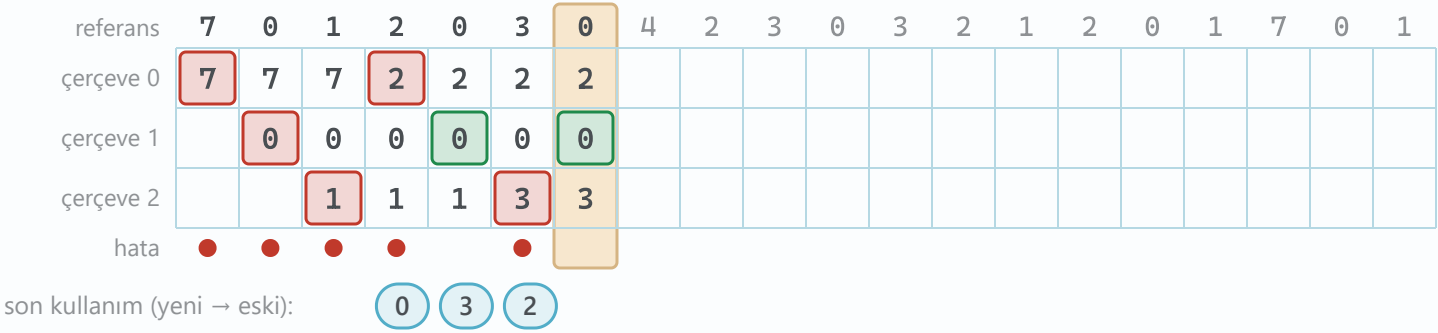
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



6. referans 3: hata; en uzun süredir kullanılmayan **1** çıkar. Hata: 5

LRU: En Uzun Süredir Kullanılmayan

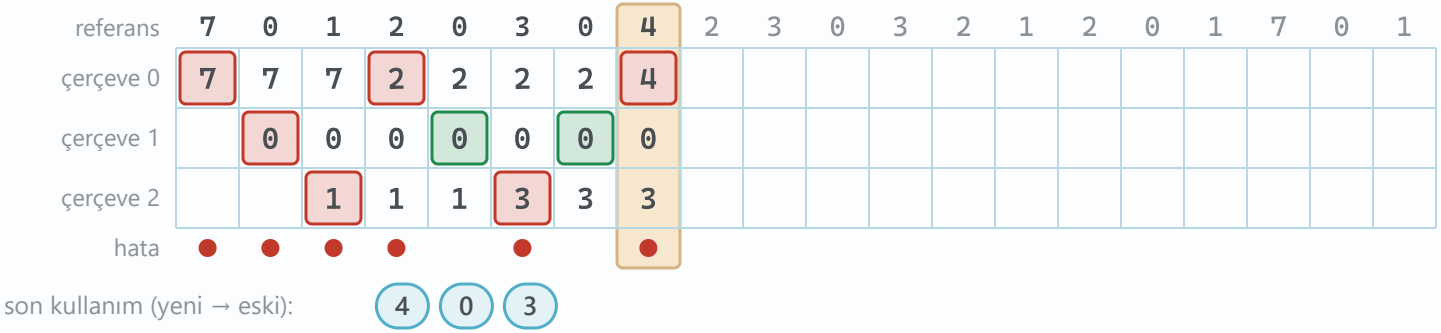
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



7. referans 0: bellekte → isabet; 0 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

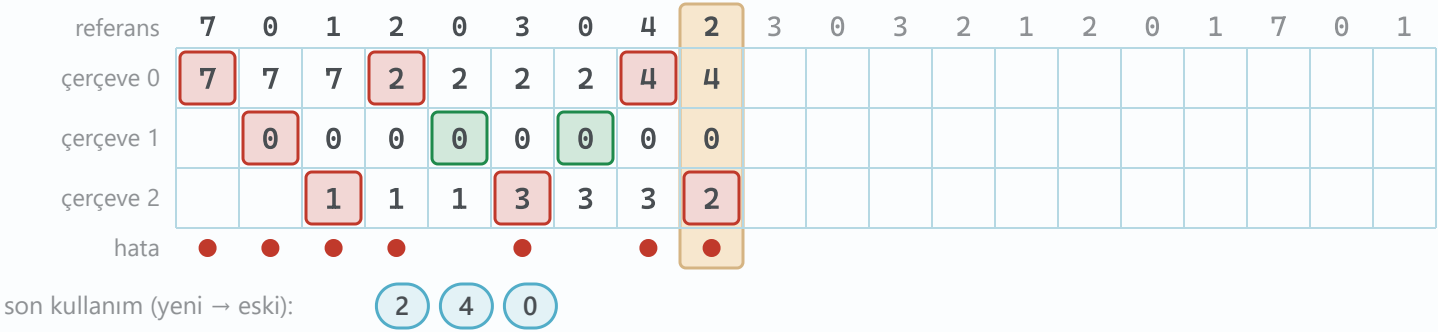
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



8. referans 4: hata; en uzun süredir kullanılmayan **2** çıkar. Hata: 6

LRU: En Uzun Süredir Kullanılmayan

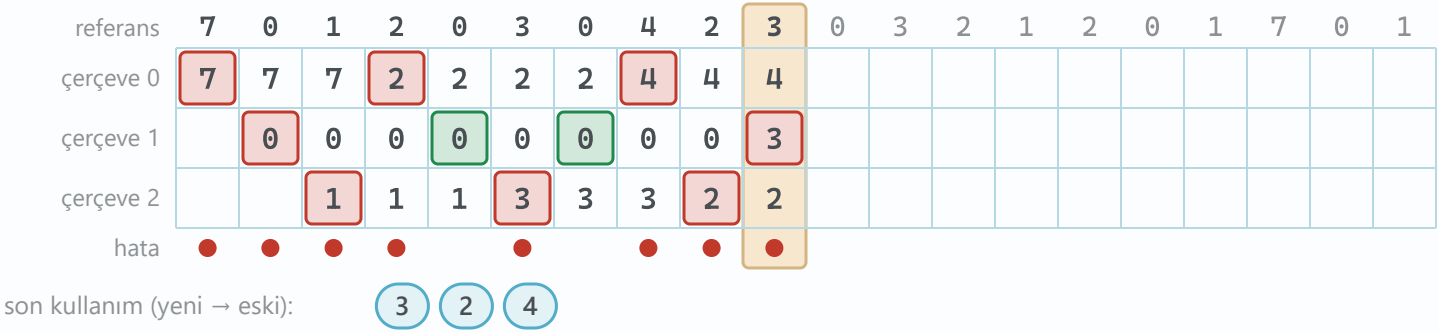
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



9. referans 2: hata; en uzun süredir kullanılmayan **3** çıkar. Hata: 7

LRU: En Uzun Süredir Kullanılmayan

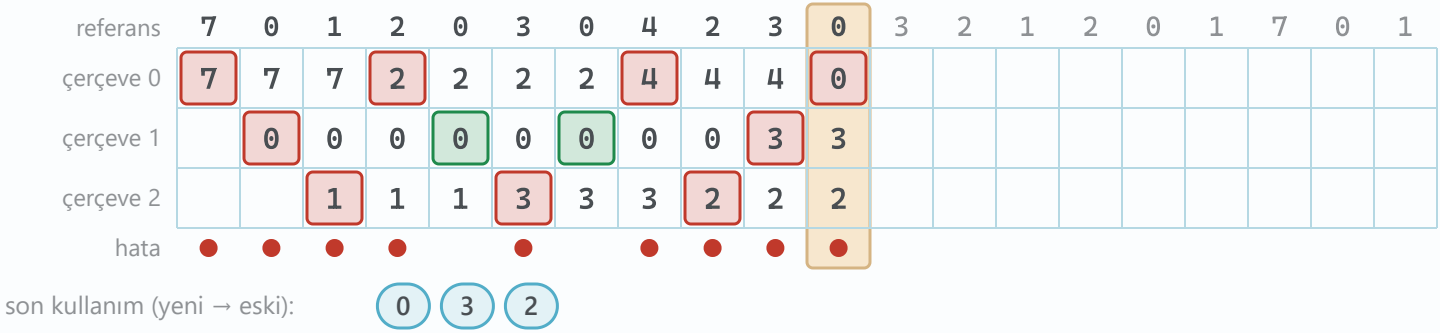
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



10. referans 3: hata; en uzun süredir kullanılmayan **0** çıkar. Hata: 8

LRU: En Uzun Süredir Kullanılmayan

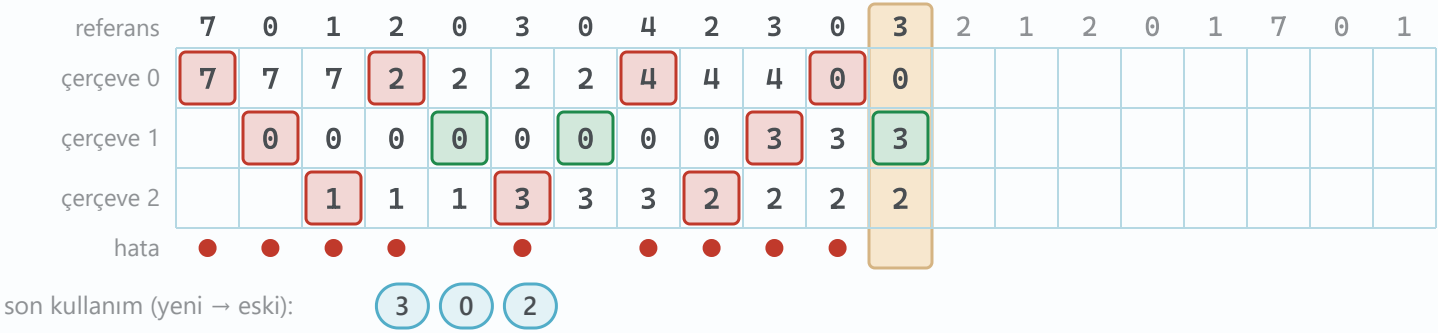
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



11. referans 0: hata; en uzun süredir kullanılmayan **4** çıkar. Hata: 9

LRU: En Uzun Süredir Kullanılmayan

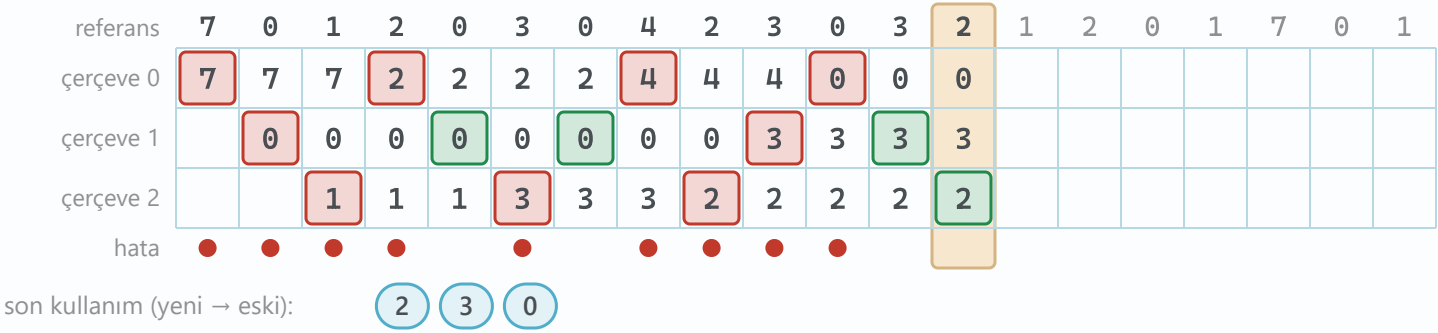
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



12. referans 3: bellekte → isabet; 3 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

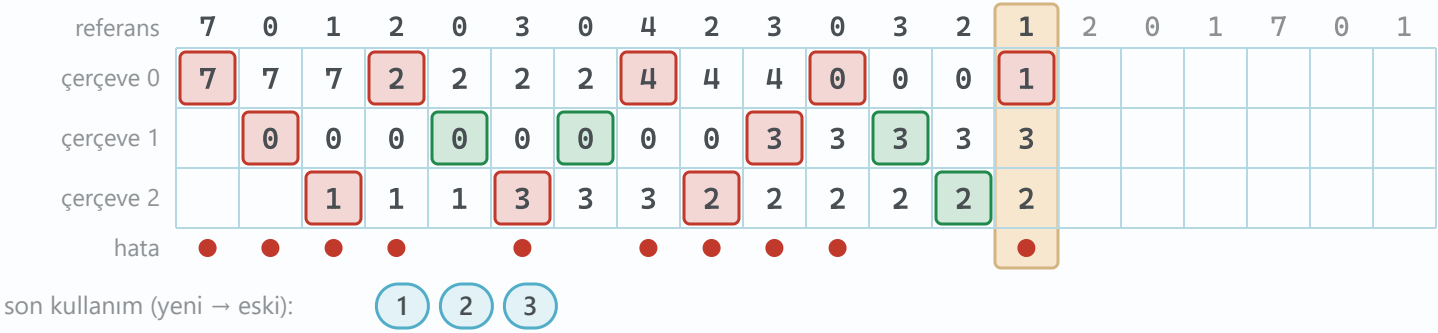
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



13. referans 2: bellekte → isabet; 2 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

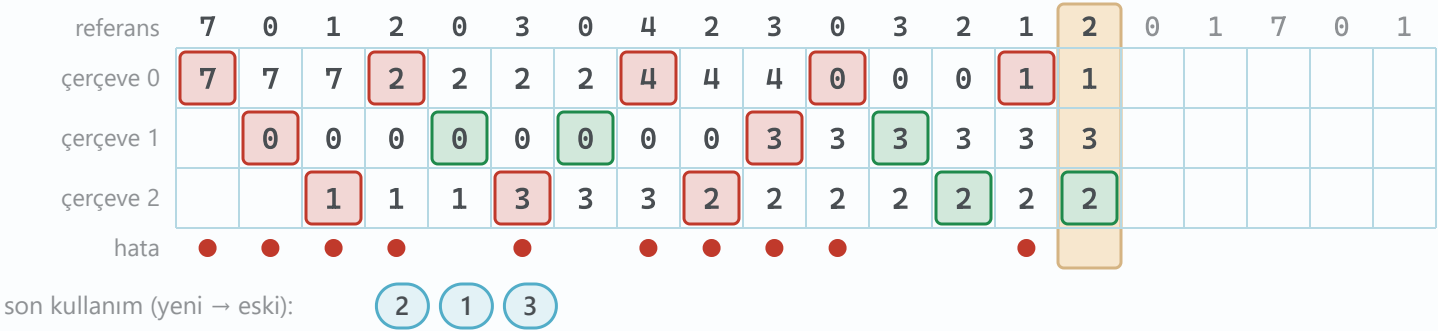
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



14. referans 1: hata; en uzun süredir kullanılmayan **0** çıkar. Hata: 10

LRU: En Uzun Süredir Kullanılmayan

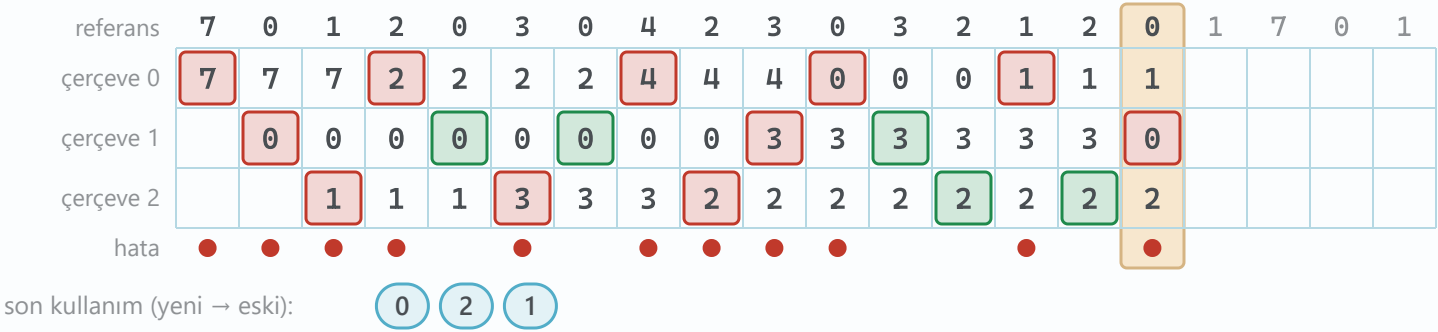
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



15. referans 2: bellekte → isabet; 2 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

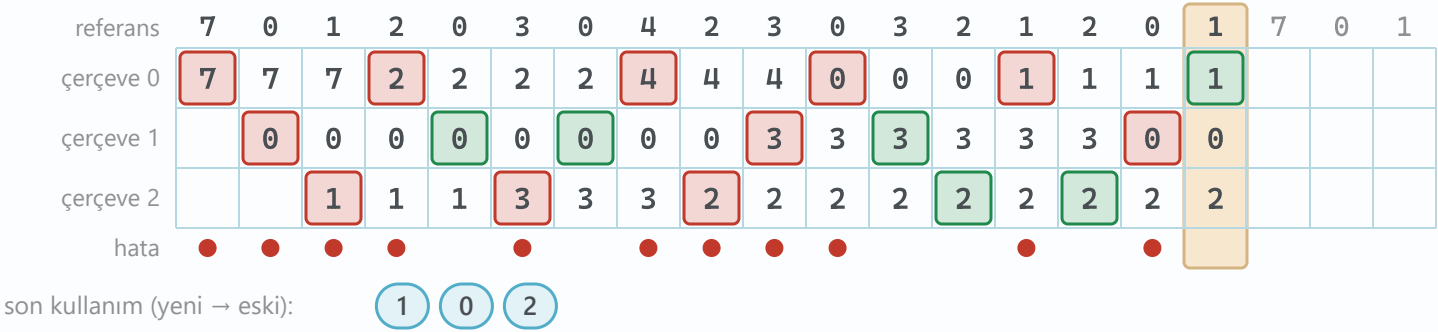
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



16. referans 0: hata; en uzun süredir kullanılmayan **3** çıkar. Hata: 11

LRU: En Uzun Süredir Kullanılmayan

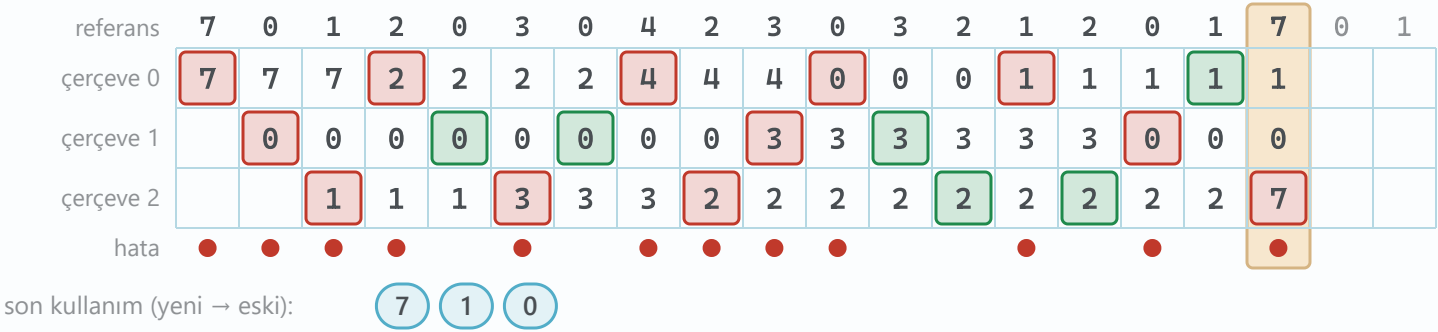
- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



17. referans 1: bellekte → isabet; 1 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.



18. referans 7: hata; en uzun süredir kullanılmayan **2** çıkar. Hata: 12

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.

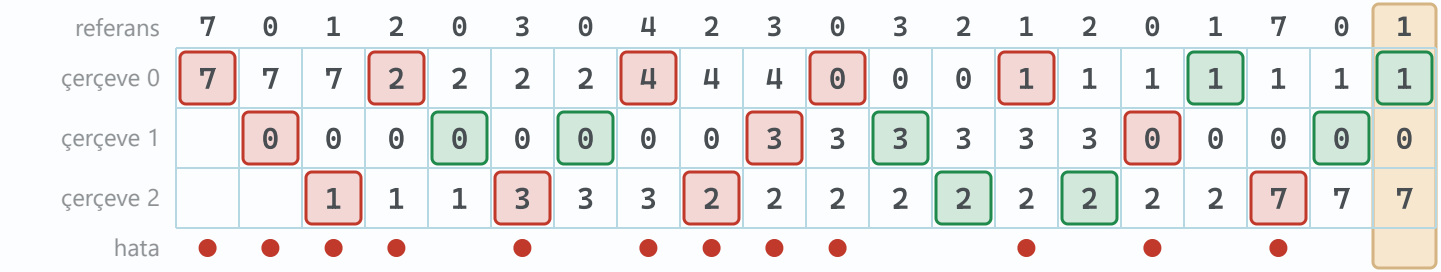
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	
çerçeve 1		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	
çerçeve 2			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	
hata	●	●	●	●		●		●	●	●	●			●		●		●		

son kullanım (yeni → eski): 0 7 1

19. referans 0: bellekte → isabet; 0 en yeni olur.

LRU: En Uzun Süredir Kullanılmayan

- **Geçmişe bakan OPT:** yakın geçmişte kullanılmayan sayfa yakın gelecekte de kullanılmayacaktır (yerellik).
- En uzun süredir **kullanılmayan** (least recently used) sayfa çıkar.
- İyi sonuç verir ama **her erişimde** kayıt güncellenmelidir: donanım desteği olmadan pahalı.
- En son kullanılanı atan algoritma MRU'dur; LRU en *eski* kullanılanı atar.

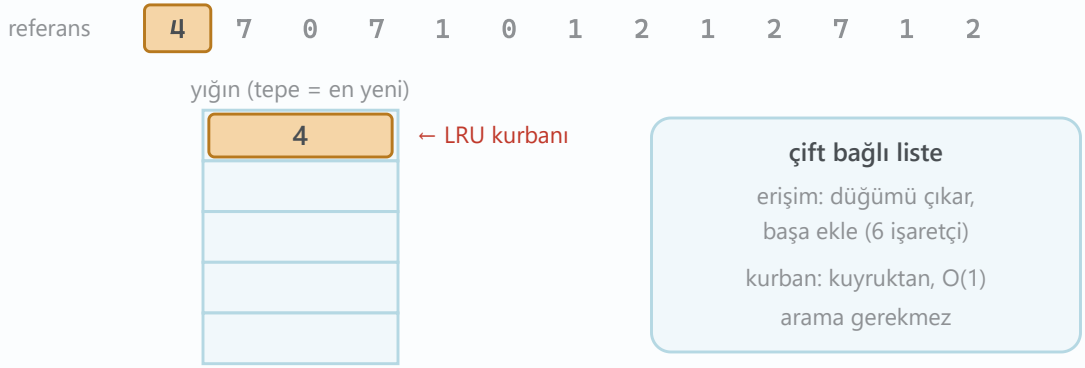


son kullanım (yeni → eski): 1 0 7

20. referans 1: bellekte → isabet; 1 en yeni olur. **Toplam 12 hata** (9 yer değiştirme): FIFO'dan iyi, OPT'den kötü.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

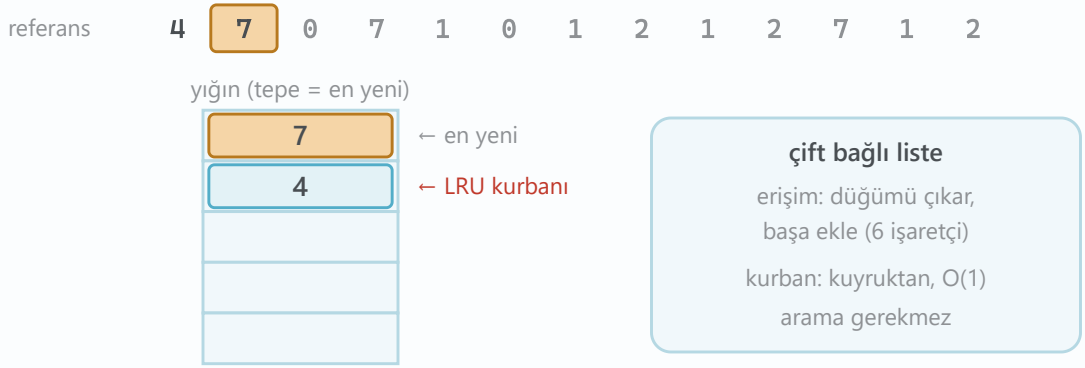
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



4 yığında yoktu: tepeye **itilir** (push).

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

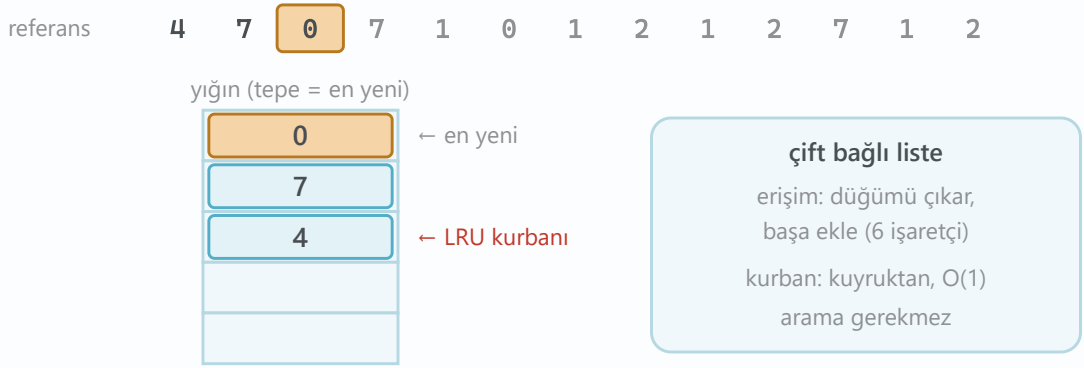
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



7 yığında yoktu: tepeye **itilir** (push).

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



0 yığında yoktu: tepeye **itilir** (push).

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

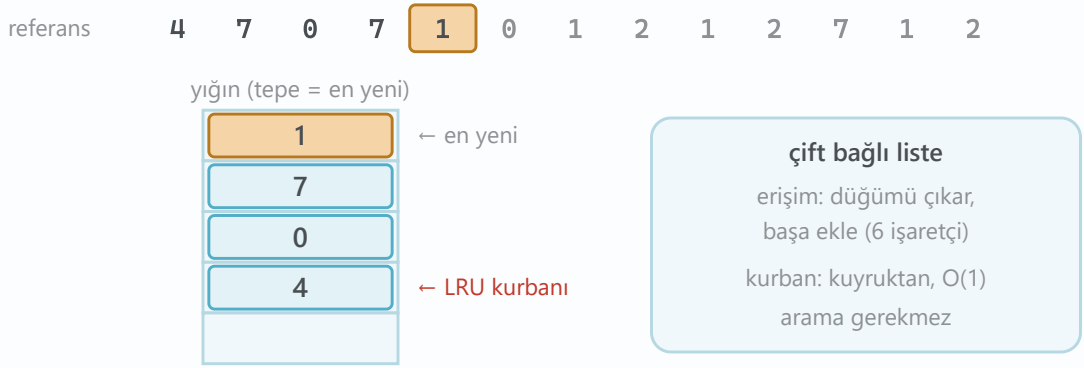
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



7 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



1 yığında yoktu: tepeye **itilir** (push).

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



0 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

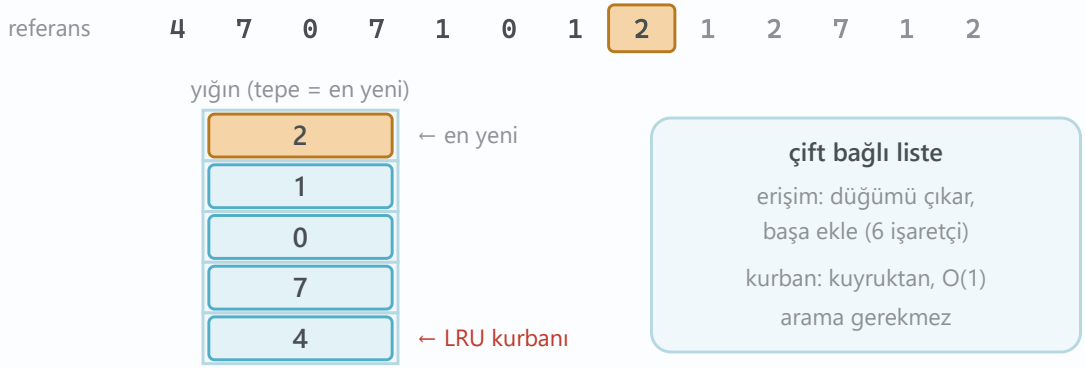
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



1 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

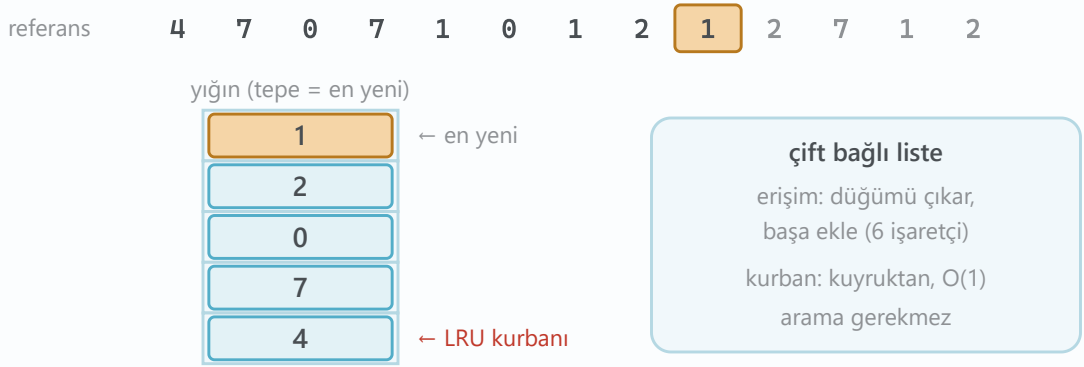
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



2 yığında yoktu: tepeye **itilir** (push).

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

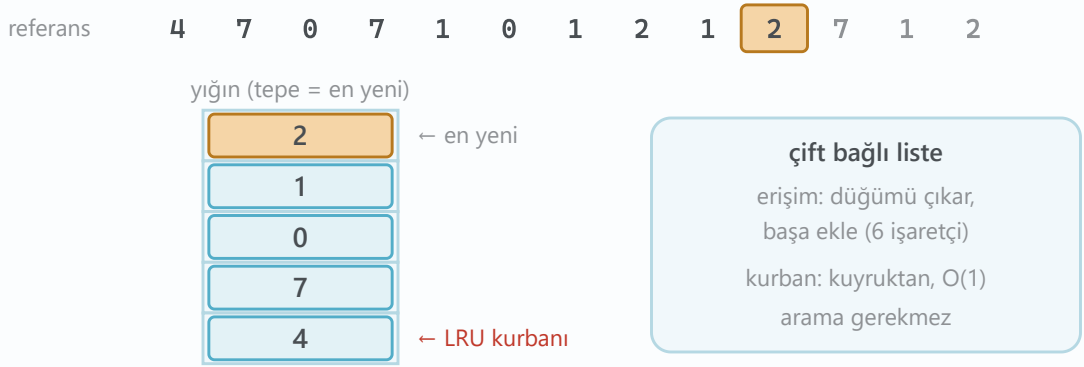
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



1 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



2 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



7 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



1 yığında vardı: **ortadan çıkarılıp tepeye** konur.

LRU'yu Gerçekleştirmek: Sayaç ve Yığın

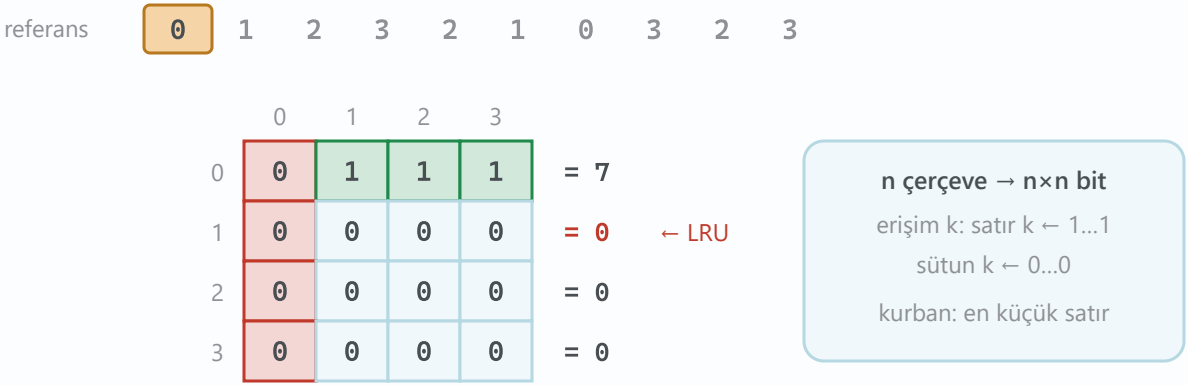
- **Sayaç (zaman damgası):** CPU'da bir mantıksal saat; her erişimde sayfanın girdisine saat değeri yazılır. Kurban: *en küçük* damga → her hatada tüm tabloyu tarama.
- Her bellek erişiminde tabloya bir yazma demektir; donanım desteği şarttır.
- **Yığın:** sayfa numaraları bir yığında; erişilen sayfa *tepeye* taşınır, kurban *en alttaki*dir.
- Güncelleme pahalı, seçim ucuz. İkisi de bugün donanımda yoktur; pratikte **yaklaşımlar** kullanılır.



2 yığında vardı: **ortadan çıkarılıp tepeye** konur. En altta **4** var: bir yer değiştirmede o çıkar.

LRU'yu Gerçekleştirmek: Matris Yöntemi

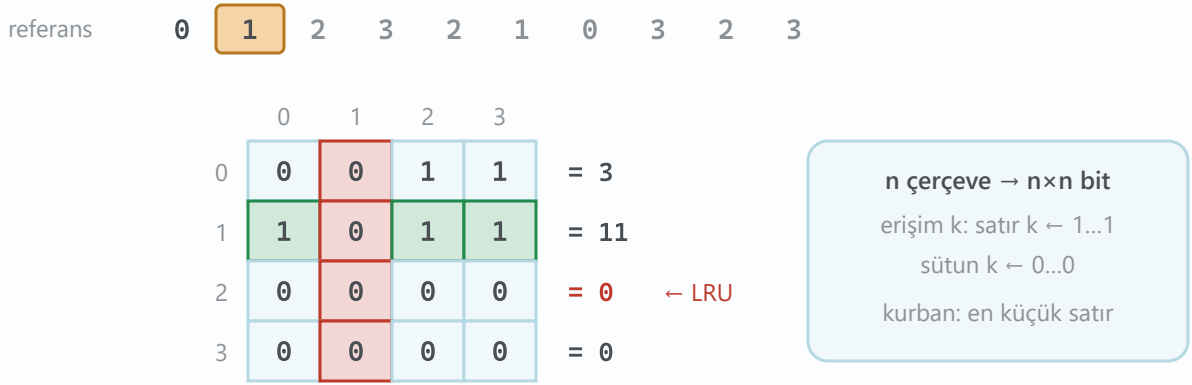
Tanenbaum'un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.



Sayfa **0**: 0. satır 1, 0. sütun 0 yapılır. Satır değerleri 7, 0, 0, 0 → LRU: **1**.

LRU'yu Gerçekleştirmek: Matris Yöntemi

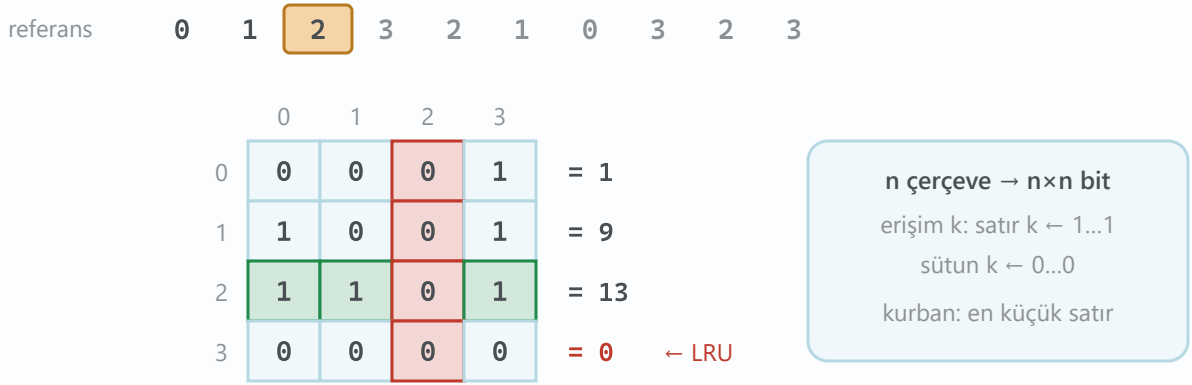
Tanenbaum'un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.



Sayfa **1**: 1. satır 1, 1. sütun 0 yapılır. Satır değerleri 3, 11, 0, 0 → LRU: **2**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.



Sayfa **2**: 2. satır 1, 2. sütun 0 yapılır. Satır değerleri 1, 9, 13, 0 → LRU: **3**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.

referans	0	1	2	**3**	2	1	0	3	2	3

	0	1	2	3		
0	0	0	0	0	= 0	← LRU
1	1	0	0	0	= 8	
2	1	1	0	0	= 12	
3	1	1	1	0	= 14	

n çerçeve → n×n bit

erişim k: satır k ← 1…1

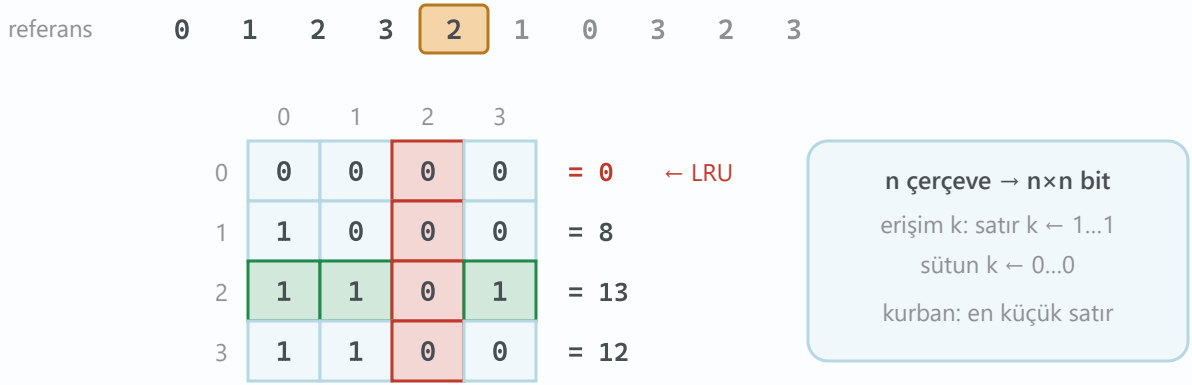
sütun k ← 0…0

kurban: en küçük satır

Sayfa **3**: 3. satır 1, 3. sütun 0 yapılır. Satır değerleri 0, 8, 12, 14 → LRU: **0**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

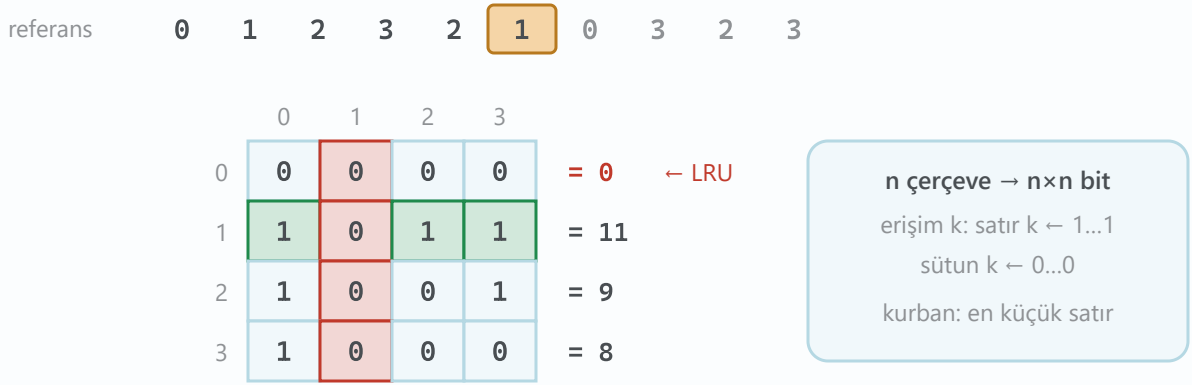
Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.



Sayfa **2**: 2. satır 1, 2. sütun 0 yapılır. Satır değerleri 0, 8, 13, 12 → LRU: **0**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.



Sayfa **1**: 1. satır 1, 1. sütun 0 yapılır. Satır değerleri 0, 11, 9, 8 → LRU: **0**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.

referans	0	1	2	3	2	1	0	3	2	3

	0	1	2	3		
0	0	1	1	1	= 7	
1	0	0	1	1	= 3	
2	0	0	0	1	= 1	
3	0	0	0	0	= 0	← LRU

n çerçeve → n×n bit

erişim k: satır k ← 1…1

sütun k ← 0…0

kurban: en küçük satır

Sayfa **0**: 0. satır 1, 0. sütun 0 yapılır. Satır değerleri 7, 3, 1, 0 → LRU: **3**.

LRU'yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum'un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.

referans	0	1	2	3	2	1	0	**3**	2	3

	0	1	2	3	
0	0	1	1	0	= 6
1	0	0	1	0	= 2
2	0	0	0	0	= 0 ← LRU
3	1	1	1	0	= 14

n çerçeve → n×n bit

erişim k: satır k ← 1…1

sütun k ← 0…0

kurban: en küçük satır

Sayfa **3**: 3. satır 1, 3. sütun 0 yapılır. Satır değerleri 6, 2, 0, 14 → LRU: **2**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.

referans	0	1	2	3	2	1	0	3	2	3

	0	1	2	3		
0	0	1	0	0	= 4	
1	0	0	0	0	= 0	← LRU
2	1	1	0	1	= 13	
3	1	1	0	0	= 12	

n çerçeve → n×n bit

erişim k: satır k ← 1…1

sütun k ← 0…0

kurban: en küçük satır

Sayfa **2**: 2. satır 1, 2. sütun 0 yapılır. Satır değerleri 4, 0, 13, 12 → LRU: **1**.

LRU’yu Gerçekleştirmek: Matris Yöntemi

Tanenbaum’un donanım çözümü: 4 çerçeve, erişim sırası `0 1 2 3 2 1 0 3 2 3`. Satırın ikili değeri ne kadar büyükse sayfa o kadar yakın zamanda kullanılmıştır.

referans	0	1	2	3	2	1	0	3	2	**3**

	0	1	2	3		
0	0	1	0	0	= 4	
1	0	0	0	0	= 0	← LRU
2	1	1	0	0	= 12	
3	1	1	1	0	= 14	

n çerçeve → n×n bit

erişim k: satır k ← 1…1

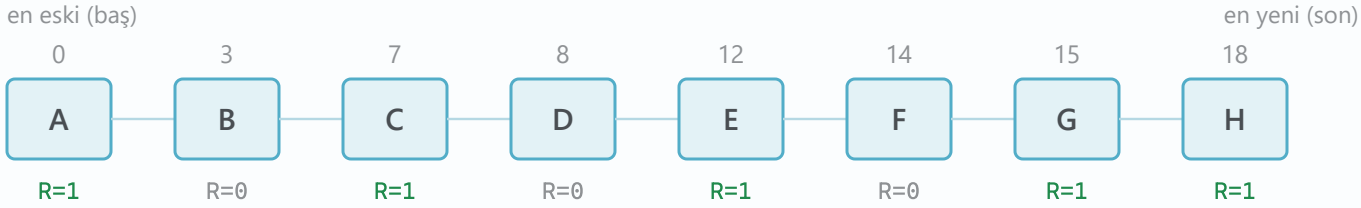
sütun k ← 0…0

kurban: en küçük satır

Sayfa **3**: 3. satır 1, 3. sütun 0 yapılır. Satır değerleri 4, 0, 12, 14 → LRU: **1**.

İkinci Şans (Second Chance)

- FIFO'nun iyileştirilmiş hâli: baştaki sayfanın **R (referans) biti**ne bakılır.
- R = 0 → sayfa eski ve kullanılmamış: hemen atılır.
- R = 1 → R temizlenir, sayfa sona taşınır (ikinci şans) ve arama sürer.
- Sorun: sayfaları liste içinde sürekli **taşımak** gereksiz iştir → saat algoritması.



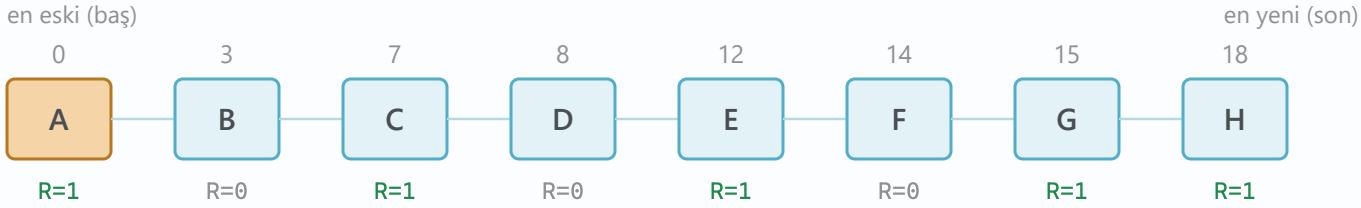
üstteki sayılar: yüklenme zamanı

Sayfalar FIFO sırasında; zaman 20'de sayfa hatası oluşur ve yer gerekir.

Tanenbaum MOS Şekil 3-15 (yeniden çizim).

İkinci Şans (Second Chance)

- FIFO'nun iyileştirilmiş hâli: baştaki sayfanın **R (referans) biti**ne bakılır.
- R = 0 → sayfa eski ve kullanılmamış: hemen atılır.
- R = 1 → R temizlenir, sayfa sona taşınır (ikinci şans) ve arama sürer.
- Sorun: sayfaları liste içinde sürekli **taşımak** gereksiz iştir → saat algoritması.

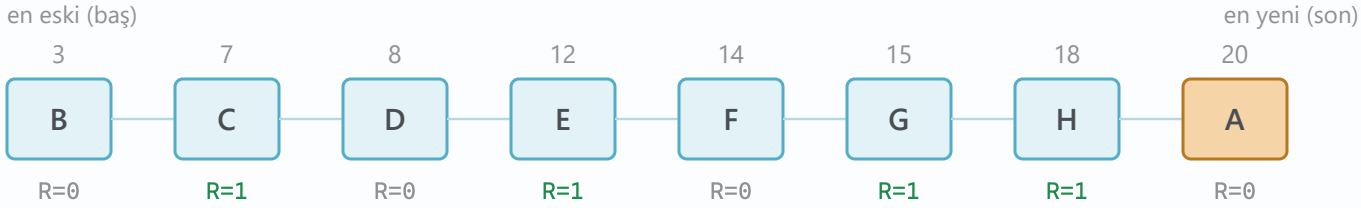


Baştaki **A**'nın R biti 1: yakın zamanda kullanılmış. Atmak yerine **ikinci şans** verilir.

Tanenbaum MOS Şekil 3-15 (yeniden çizim).

İkinci Şans (Second Chance)

- FIFO'nun iyileştirilmiş hâli: baştaki sayfanın **R (referans) biti**ne bakılır.
- R = 0 → sayfa eski ve kullanılmamış: hemen atılır.
- R = 1 → R temizlenir, sayfa sona taşınır (ikinci şans) ve arama sürer.
- Sorun: sayfaları liste içinde sürekli **taşımak** gereksiz iştir → saat algoritması.



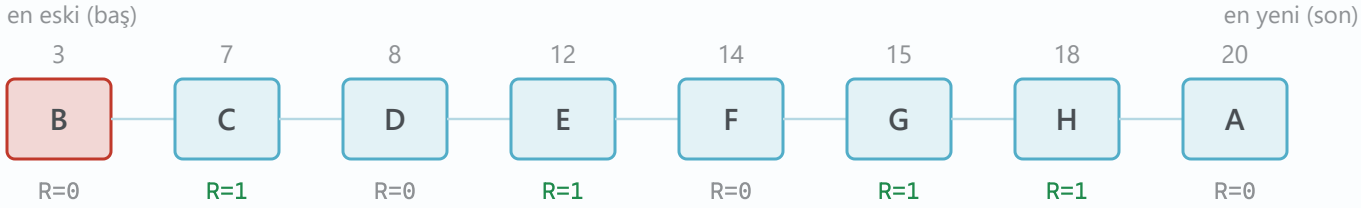
üstteki sayılar: yüklenme zamanı

A'nın R'si 0 yapılır ve yeni yüklenmiş gibi **sona** taşınır (zaman 20).

Tanenbaum MOS Şekil 3-15 (yeniden çizim).

İkinci Şans (Second Chance)

- FIFO'nun iyileştirilmiş hâli: baştaki sayfanın **R (referans) biti**ne bakılır.
- R = 0 → sayfa eski ve kullanılmamış: hemen atılır.
- R = 1 → R temizlenir, sayfa sona taşınır (ikinci şans) ve arama sürer.
- Sorun: sayfaları liste içinde sürekli **taşımak** gereksiz iştir → saat algoritması.



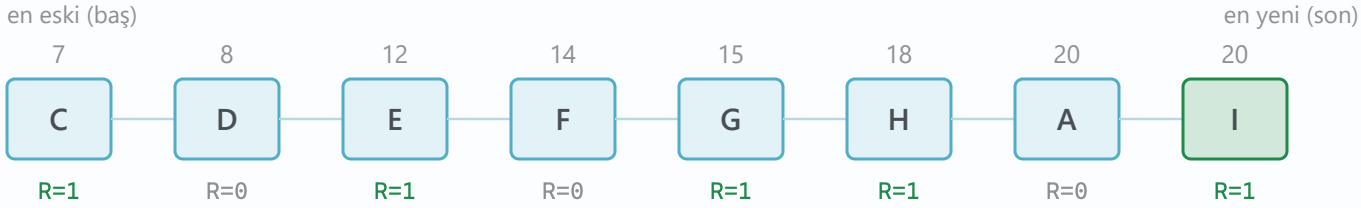
üstteki sayılar: yüklenme zamanı

Yeni baş **B**: R = 0 → hem eski hem kullanılmamış: **kurban**.

Tanenbaum MOS Şekil 3-15 (yeniden çizim).

İkinci Şans (Second Chance)

- FIFO'nun iyileştirilmiş hâli: baştaki sayfanın **R (referans) biti**ne bakılır.
- R = 0 → sayfa eski ve kullanılmamış: hemen atılır.
- R = 1 → R temizlenir, sayfa sona taşınır (ikinci şans) ve arama sürer.
- Sorun: sayfaları liste içinde sürekli **taşımak** gereksiz iştir → saat algoritması.



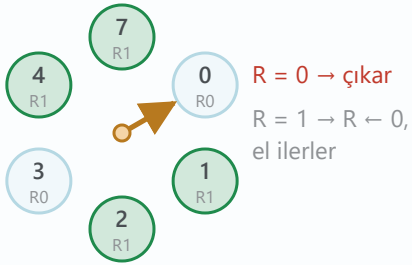
üstteki sayılar: yüklenme zamanı

B atılır, yeni sayfa I sona eklenir. Tüm R'ler 1 olsaydı liste bir tur dönüp **saf FIFO** gibi davranırdı.

Tanenbaum MOS Şekil 3-15 (yeniden çizim).

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.

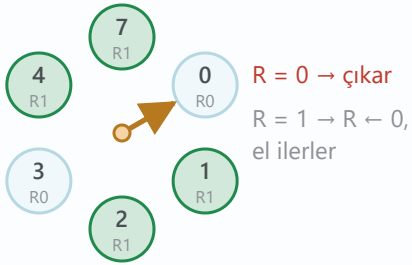


	referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	çerçeve 0	7 R1																			
	çerçeve 1																				
	çerçeve 2																				
	hata	●																			
R bitleri:		7: 1																			

1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.

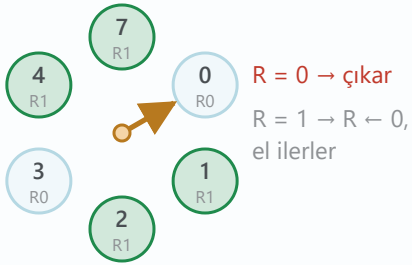


	referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	çerçeve 0	7	7 R1																		
	çerçeve 1		0 R1																		
	çerçeve 2																				
	hata	●	●																		
R bitleri:		7: 1	0: 1																		

2. referans 0: hata; boş çerçeve 1'e yüklenir. Hata: 2

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



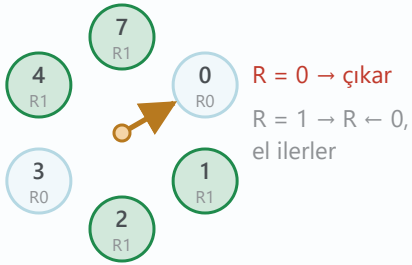
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7 R1																	
çerçeve 1		0	0 R1																	
çerçeve 2			1 R1																	
hata	●	●	●																	

R bitleri: 7: 1 | 0: 1 | 1: 1

3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



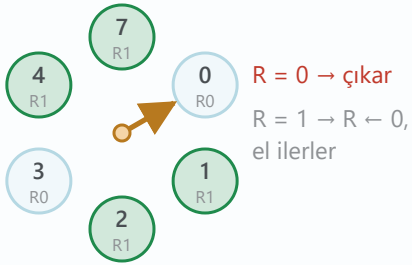
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2 R1																
çerçeve 1		0	0	0 R0																
çerçeve 2			1	1 R0																
hata	●	●	●	●																

R bitleri: 2: 1 | 0: 0 | 1: 0

4. referans 2: hata; R=1 olan 7, 0, 1 için R ← 0 ve el ilerler; elin gösterdiği **7**'nin R'si 0 → çıkar. Hata: 4

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



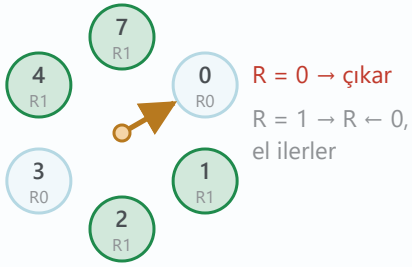
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2 R1															
çerçeve 1		0	0	0	0 R1															
çerçeve 2			1	1	1 R0															
hata	●	●	●	●																

R bitleri: 2: 1 | 0: 1 | 1: 0

5. referans 0: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



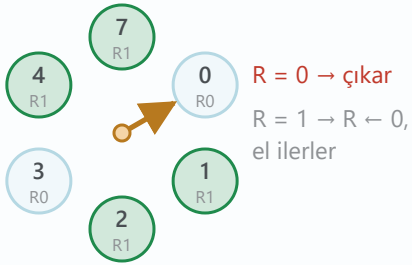
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2 R1														
çerçeve 1		0	0	0	0	0 R0														
çerçeve 2			1	1	1	3 R1														
hata	●	●	●	●		●														

R bitleri: 2: 1 | 0: 0 | 3: 1

6. referans 3: hata; R=1 olan 0 için R ← 0 ve el ilerler; elin gösterdiği **1**'in R'si 0 → çıkar. Hata: 5

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



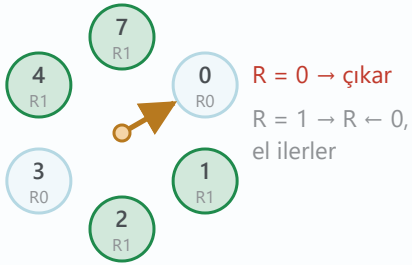
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2 R1													
çerçeve 1		0	0	0	0	0	0 R1													
çerçeve 2			1	1	1	3	3 R1													
hata	●	●	●	●		●														

R bitleri: 2: 1 | 0: 1 | 3: 1

7. referans 0: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



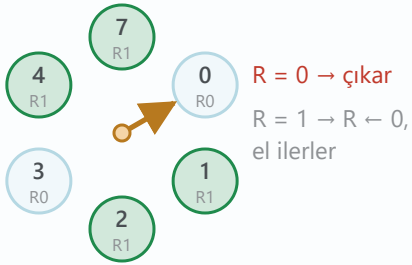
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4 R1												
çerçeve 1		0	0	0	0	0	0	0 R0												
çerçeve 2			1	1	1	3	3	3 R0												
hata	●	●	●	●		●		●												

R bitleri: 4: 1 | 0: 0 | 3: 0

8. referans 4: hata; R=1 olan 2, 0, 3 için R ← 0 ve el ilerler; elin gösterdiği **2**'nin R'si 0 → çıkar. Hata: 6

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



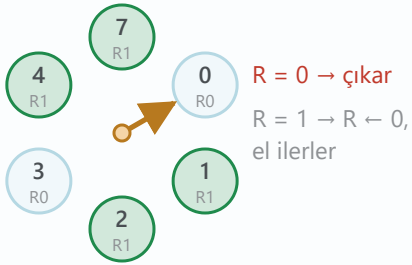
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4 R1											
çerçeve 1		0	0	0	0	0	0	0	2 R1											
çerçeve 2			1	1	1	3	3	3	3 R0											
hata	●	●	●	●		●		●	●											

R bitleri: 4: 1 | 2: 1 | 3: 0

9. referans 2: hata; elin gösterdiği **0**'ın R'si 0 → çıkar. Hata: 7

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



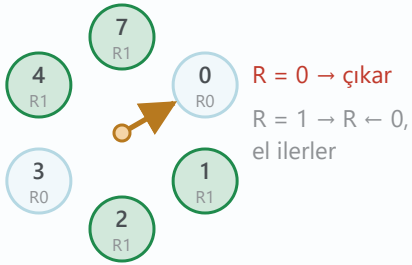
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4 R1										
çerçeve 1		0	0	0	0	0	0	0	2	2 R1										
çerçeve 2			1	1	1	3	3	3	3	3 R1										
hata	●	●	●	●		●		●	●											

R bitleri: 4: 1 | 2: 1 | 3: 1

10. referans 3: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



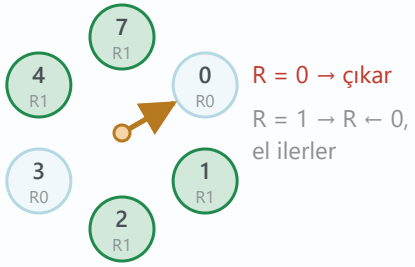
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4 R0									
çerçeve 1		0	0	0	0	0	0	0	2	2	2 R0									
çerçeve 2			1	1	1	3	3	3	3	3	0 R1									
hata	●	●	●	●		●		●	●		●									

R bitleri: 4: 0 | 2: 0 | 0: 1

11. referans 0: hata; R=1 olan 3, 4, 2 için R ← 0 ve el ilerler; elin gösterdiği **3**'ün R'si 0 → çıkar. Hata: 8

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



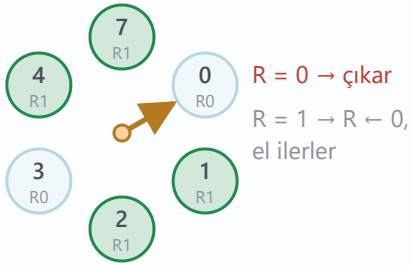
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3 R1								
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2 R0								
çerçeve 2			1	1	1	3	3	3	3	3	0	0 R1								
hata	●	●	●	●		●		●	●		●	●								

R bitleri: 3: 1 | 2: 0 | 0: 1

12. referans 3: hata; elin gösterdiği **4**'ün R'si 0 → çıkar. Hata: 9

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



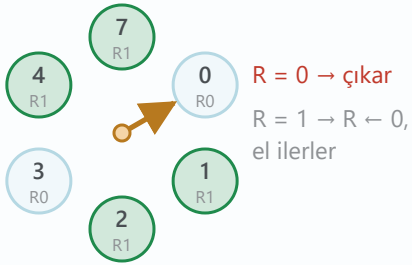
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3 R1							
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2 R1							
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0 R1							
hata	●	●	●	●		●		●	●		●	●								

R bitleri: 3: 1 | 2: 1 | 0: 1

13. referans 2: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



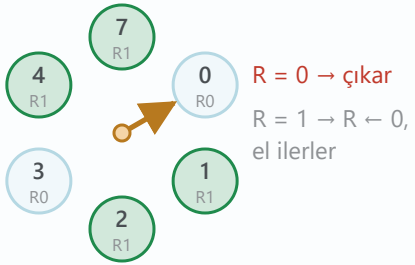
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3 R0						
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1 R1						
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0 R0						
hata	●	●	●	●		●		●	●		●	●		●						

R bitleri: 3: 0 | 1: 1 | 0: 0

14. referans 1: hata; R=1 olan 2, 0, 3 için R ← 0 ve el ilerler; elin gösterdiği **2**'nin R'si 0 → çıkar. Hata: 10

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



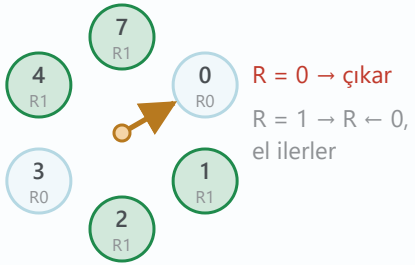
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3 R0					
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1 R1					
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2 R1					
hata	●	●	●	●		●		●	●		●	●		●	●					

R bitleri: 3: 0 | 1: 1 | 2: 1

15. referans 2: hata; elin gösterdiği **0**'ın R'si 0 → çıkar. Hata: 11

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



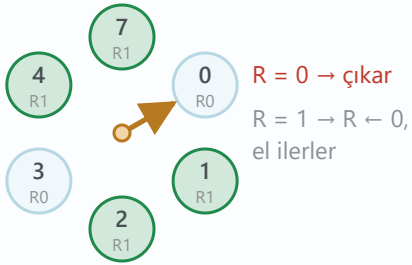
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0 R1				
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1 R1				
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2	2 R1				
hata	●	●	●	●		●		●	●		●	●		●	●	●				

R bitleri: 0: 1 | 1: 1 | 2: 1

16. referans 0: hata; elin gösterdiği **3**'ün R'si 0 → çıkar. Hata: 12

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



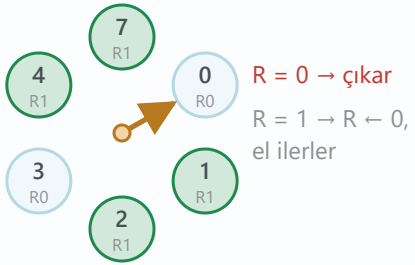
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0	0 R1			
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1 R1			
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2 R1			
hata	●	●	●	●		●		●	●		●	●		●	●	●				

R bitleri: 0: 1 | 1: 1 | 2: 1

17. referans 1: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



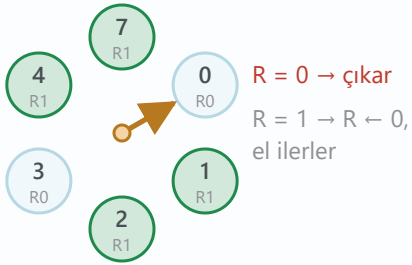
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0	0	0 R0		
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	7 R1		
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2	2 R0		
hata	●	●	●	●		●		●	●		●	●		●	●	●		●		

R bitleri: 0: 0 | 7: 1 | 2: 0

18. referans 7: hata; R=1 olan 1, 2, 0 için R ← 0 ve el ilerler; elin gösterdiği **1**'in R'si 0 → çıkar. Hata: 13

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



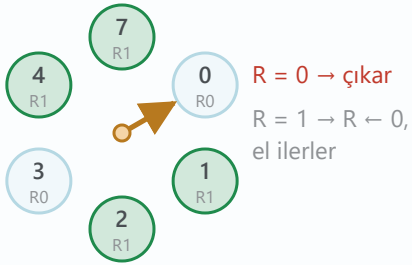
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0	0	0	0 R1	
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	7	7 R1	
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2	2	2 R0	
hata	●	●	●	●		●		●	●		●	●		●	●	●		●		

R bitleri: 0: 1 · 7: 1 · 2: 0

19. referans 0: bellekte → isabet; donanım R ← 1 yapar, el hareket etmez.

Saat (Clock) Algoritması

- İkinci şansla aynı karar, farklı yapı: çerçeveler **dairesel listede** durur, bir **el** (işaretçi) en eski sayfayı gösterir; sayfa taşınmaz.
- Yüklenen ya da erişilen sayfanın R'si 1 olur. El yalnızca hata anında döner.
- El sayfa tablosunda değil, *çerçeve listesinde* döner.



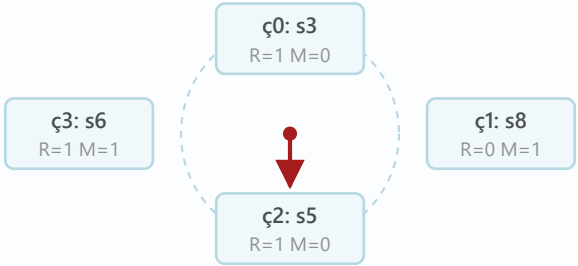
referans	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
çerçeve 0	7	7	7	2	2	2	2	4	4	4	4	3	3	3	3	0	0	0	0	0 R1
çerçeve 1		0	0	0	0	0	0	0	2	2	2	2	2	1	1	1	1	7	7	7 R1
çerçeve 2			1	1	1	3	3	3	3	3	0	0	0	0	2	2	2	2	2	1 R1
hata	●	●	●	●		●		●	●		●	●		●	●	●		●		●

R bitleri: 0: 1 | 7: 1 | 1: 1

20. referans 1: hata; elin gösterdiği **2**'nin R'si 0 → çıkar. Hata: 14 **Toplam 14 hata**: FIFO'dan (15) bir az, LRU'dan (12) çok.

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
3      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;               /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
▶    int k = kurban_sec();
▶    if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 2 · diske yazılan: 0

Dört çerçeve dolu, el çerçeve 2'de. Sayfa 9 için hata oluştu: `hata(9)` bir kurban arar.

Adım 1 / 10

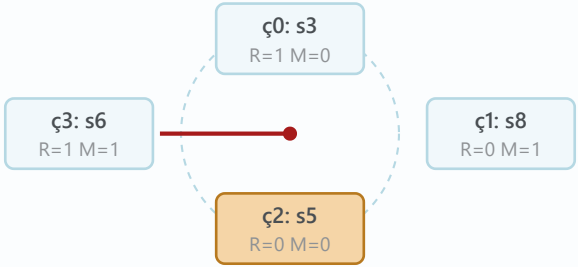
DEĞİŞKENLER

`p = 9` `el = 2` `k = –` `ç2.R = 1`

ÇIKTI

Saat Algoritması: Kod İzleme

```
 1 int kurban_sec(void) {
 2   while (cer[el].R == 1) { /* 2. şans */
▶      cer[el].R = 0;
▶      el = (el + 1) % N;
▶    }
 6   int k = el;              /* kurban */
 7   el = (el + 1) % N;
 8   return k;
 9 }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 3 · diske yazılan: 0

Çerçeve 2'de s5'in R'si 1: yakın zamanda kullanılmış. R ← 0 (ikinci şans) ve el 3'e ilerler.

Adım 2 / 10

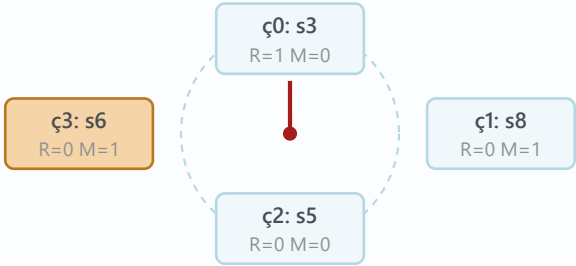
DEĞİŞKENLER

`p = 9` `el = 3` `k = –` `ç3.R = 1`

ÇIKTI

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
▶      cer[el].R = 0;
▶      el = (el + 1) % N;
▶    }
6    int k = el;              /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



Çerçeve 3'te s6'nın R'si 1: yakın zamanda kullanılmış. R ← 0 (ikinci şans) ve el 0'a ilerler.

Adım 3 / 10

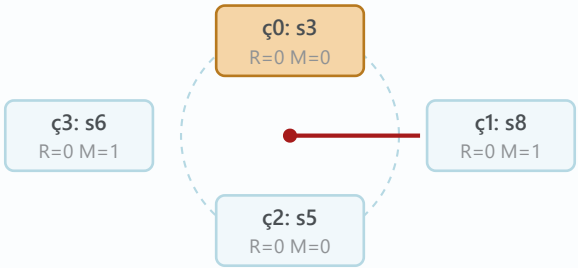
DEĞİŞKENLER

`p = 9` `el = 0` `k = –` `ç0.R = 1`

ÇIKTI

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
▶      cer[el].R = 0;
▶      el = (el + 1) % N;
▶    }
6    int k = el;              /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 1 · diske yazılan: 0

Çerçeve 0'da s3'ün R'si 1: yakın zamanda kullanılmış. R ← 0 (ikinci şans) ve el 1'e ilerler.

Adım 4 / 10

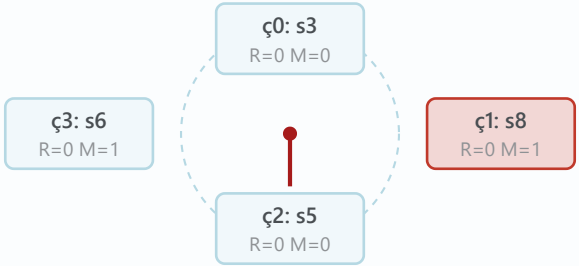
DEĞİŞKENLER

`p = 9` `el = 1` `k = –` `ç1.R = 0`

ÇIKTI

Saat Algoritması: Kod İzleme

```
 1 int kurban_sec(void) {
 2   while (cer[el].R == 1) { /* 2. şans */
 3     cer[el].R = 0;
 4     el = (el + 1) % N;
 5   }
 6   int k = el;              /* kurban */
 7   el = (el + 1) % N;
 8   return k;
 9 }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 2 · diske yazılan: 0

Çerçeve 1'de R = 0: döngü biter. k = 1 kurbandır; el bir sonraki çerçeveye (2) geçer.

Adım 5 / 10

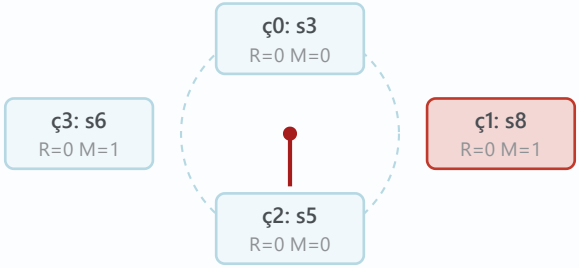
DEĞİŞKENLER

`p = 9` `el = 2` `k = 1` `ç2.R = 0`

ÇIKTI

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
3      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;              /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
▶    cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



M = 1: s8 önce diske yazılır · diske yazılan: 1

s8'in M (kirli) biti 1: bellekteki kopya diskte yok. Atılmadan önce **diske yazılır** (ek G/Ç).

Adım 6 / 10

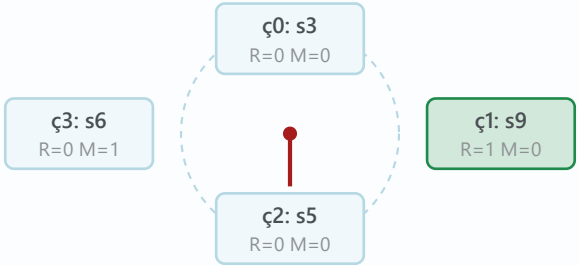
DEĞİŞKENLER

`p = 9` `el = 2` `k = 1` `ç2.R = 0`

ÇIKTI

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
3      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;               /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
▶    cer[k].M = 0;
▶  }
```



el = 2 · diske yazılan: 1

s9 çerçeve 1'e yüklenir: R = 1, M = 0. Tam bir tur dönüldü; üç sayfanın R'si artık 0.

Adım 7 / 10

DEĞİŞKENLER

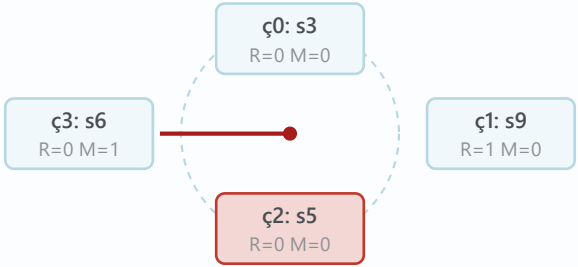
`p = 9` `el = 2` `k = 1` `ç2.R = 0`

ÇIKTI

```
hata(9): kurban ç1 (s8, kirli)
```

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
▶      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;              /* kurban */
▶    el = (el + 1) % N;
▶    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
▶    if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 3 · diske yazılan: 1

`hata(4)` : el çerçeve 2'de ve R = 0 (önceki turda sıfırlandı, o zamandan beri kullanılmadı). Kurban hemen bulunur: k = 2.

Adım 8 / 10

DEĞİŞKENLER

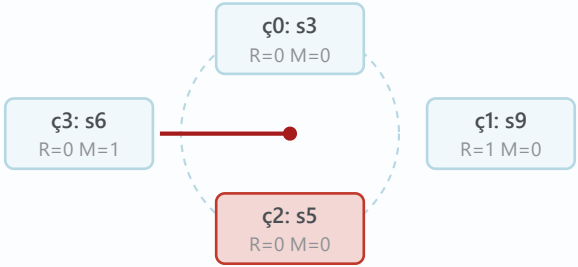
`p = 4` `el = 3` `k = 2` `ç3.R = 0`

ÇIKTI

```
hata(9): kurban ç1 (s8, kirli)
```

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
3      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;              /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
▶    cer[k].sayfa = p; cer[k].R = 1;
14   cer[k].M = 0;
15 }
```



el = 3 · diske yazılan: 1

s5 temiz (M = 0): diskteki kopya güncel, yazmaya gerek yok. Temiz kurban daha ucuzdur.

Adım 9 / 10

DEĞİŞKENLER

`p = 4` `el = 3` `k = 2` `ç3.R = 0`

ÇIKTI

```
hata(9): kurban ç1 (s8, kirli)
```

Saat Algoritması: Kod İzleme

```
1  int kurban_sec(void) {
2    while (cer[el].R == 1) { /* 2. şans */
3      cer[el].R = 0;
4      el = (el + 1) % N;
5    }
6    int k = el;              /* kurban */
7    el = (el + 1) % N;
8    return k;
9  }
10 void hata(int p) {
11   int k = kurban_sec();
12   if (cer[k].M) diske_yaz(k); /* kirli */
13   cer[k].sayfa = p; cer[k].R = 1;
▶    cer[k].M = 0;
▶  }
```



el = 3 · diske yazılan: 1

s4 çerçeve 2'ye yüklendi. İki hatada 3 R biti sıfırlandı, 1 disk yazması yapıldı; el çerçeve 3'te bekliyor.

Adım 10 / 10

DEĞİŞKENLER

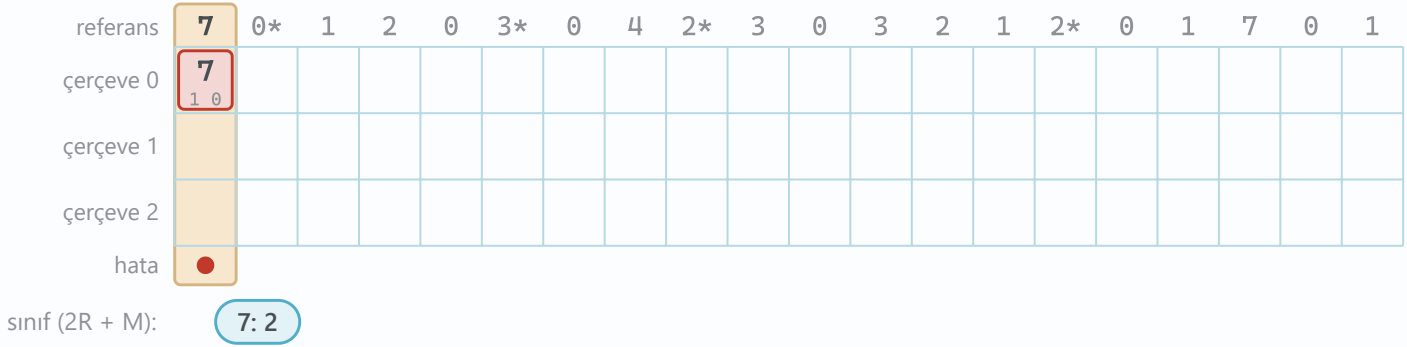
`p = 4` `el = 3` `k = 2` `ç3.R = 0`

ÇIKTI

```
hata(9): kurban ç1 (s8, kirli)
hata(4): kurban ç2 (s5, temiz)
```

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.



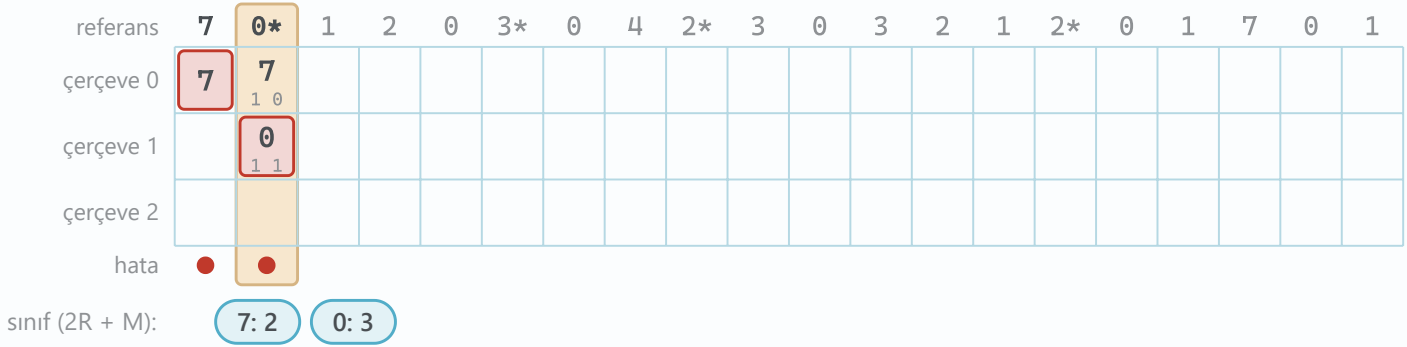
1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

Adım 1 / 20

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

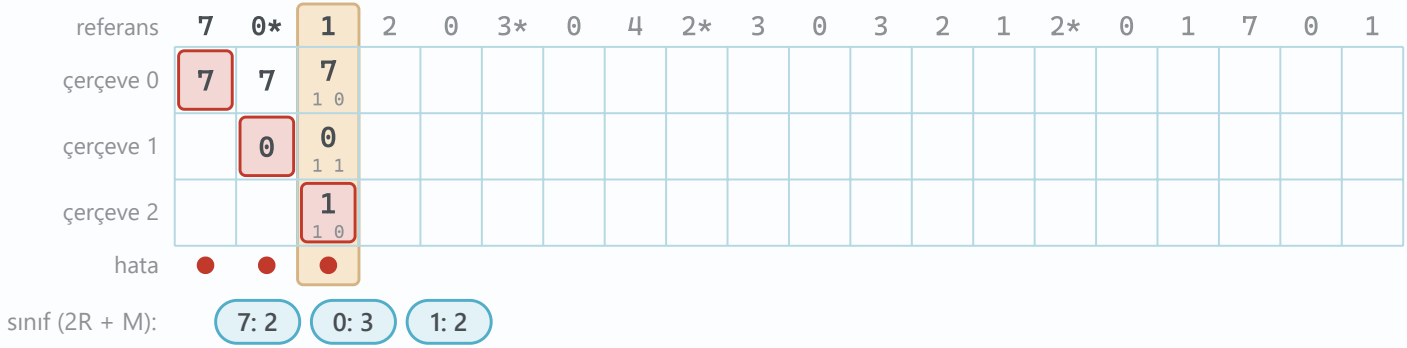


2. referans 0*: hata; boş çerçeve 1'e yüklenir. Hata: 2

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

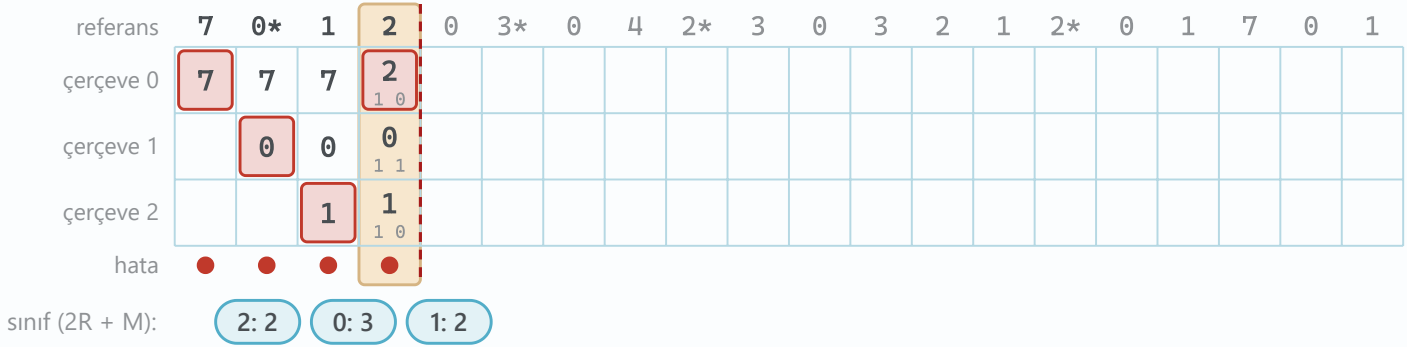


3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

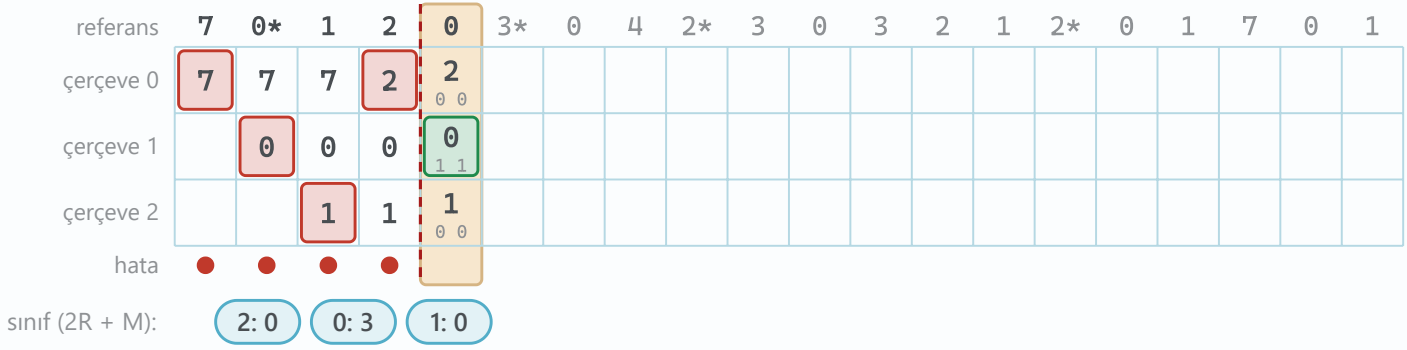


4. referans 2: hata; sınıflar (7→2, 0→3, 1→2); en düşük sınıftan **7** çıkar. Hata: 4 Saat tıkı: tüm R ← 0.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

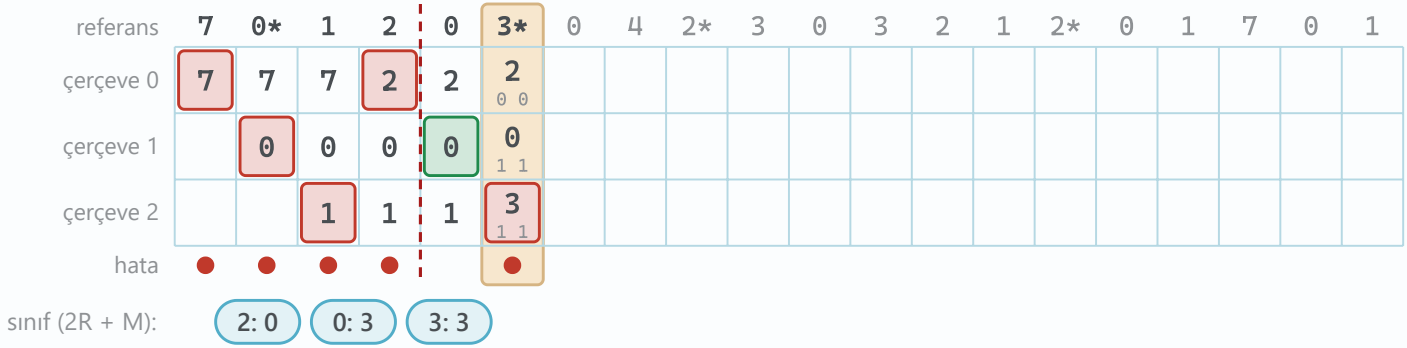


5. referans 0: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

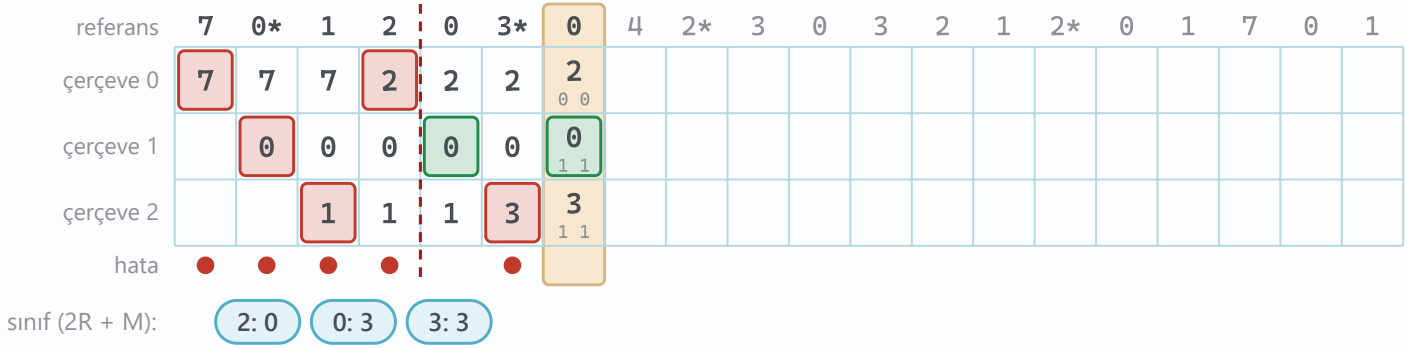


6. referans 3*: hata; sınıflar (2→0, 0→3, 1→0); en düşük sınıftan **1** çıkar. Hata: 5

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

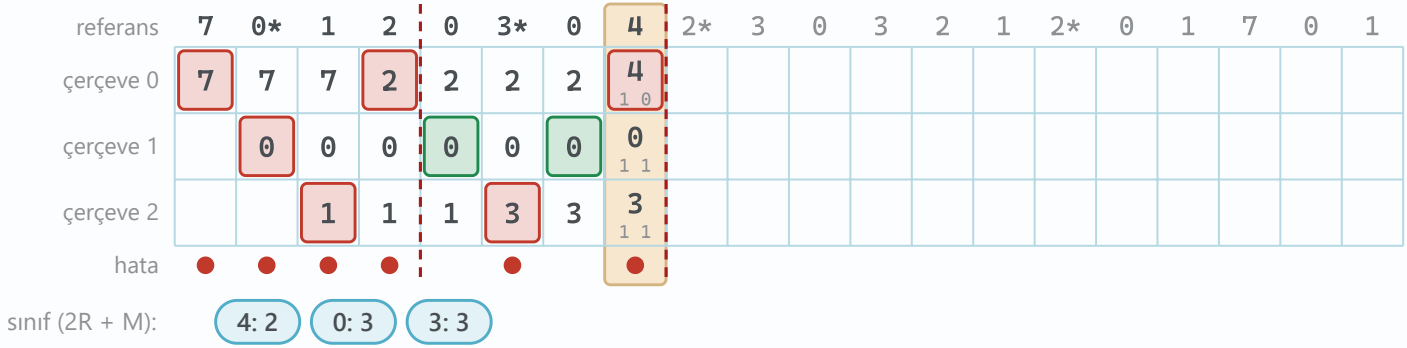


7. referans 0: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

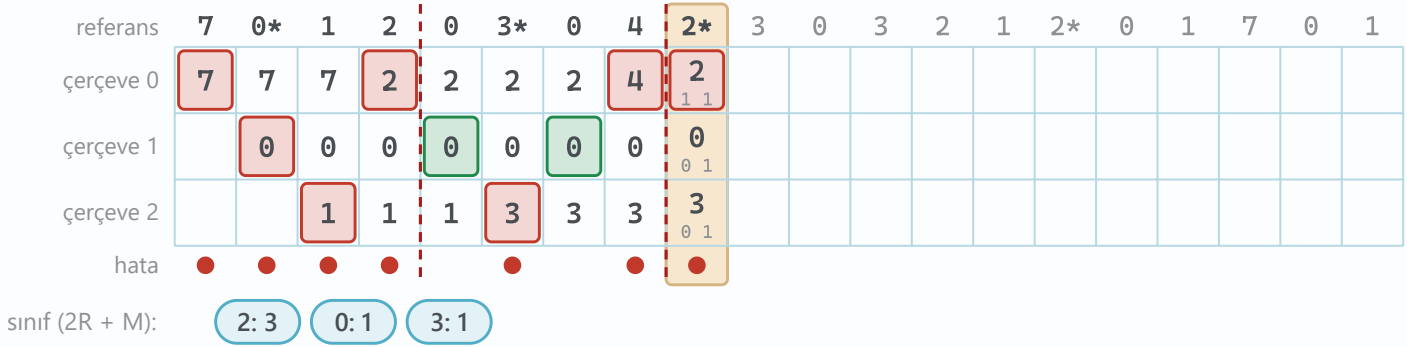


8. referans 4: hata; sınıflar (2→0, 0→3, 3→3); en düşük sınıftan **2** çıkar. Hata: 6 Saat tıkı: tüm R ← 0.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

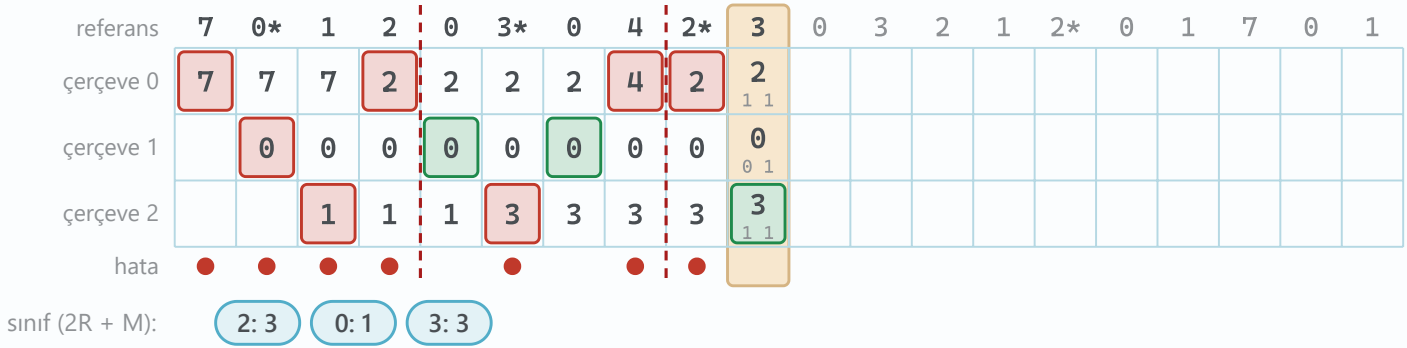


9. referans 2*: hata; sınıflar (4→0, 0→1, 3→1); en düşük sınıftan **4** çıkar. Hata: 7

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

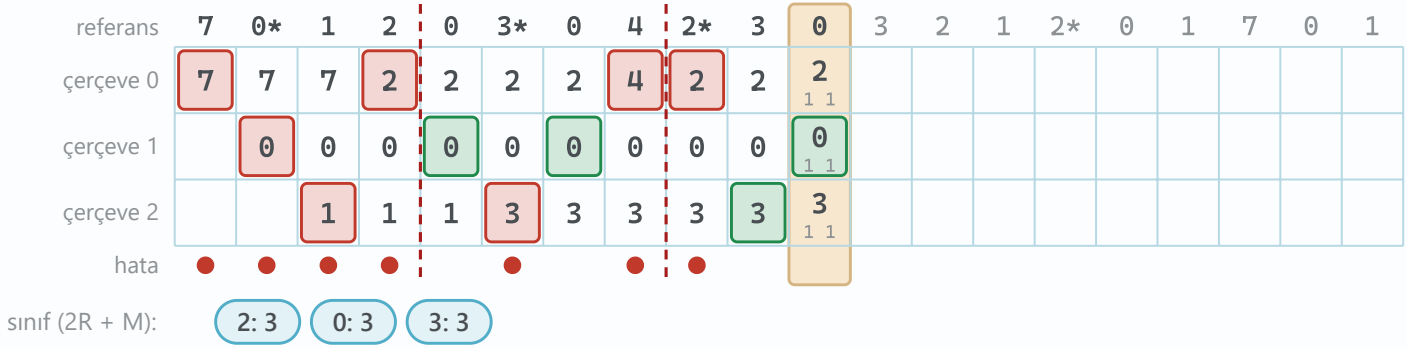


10. referans 3: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

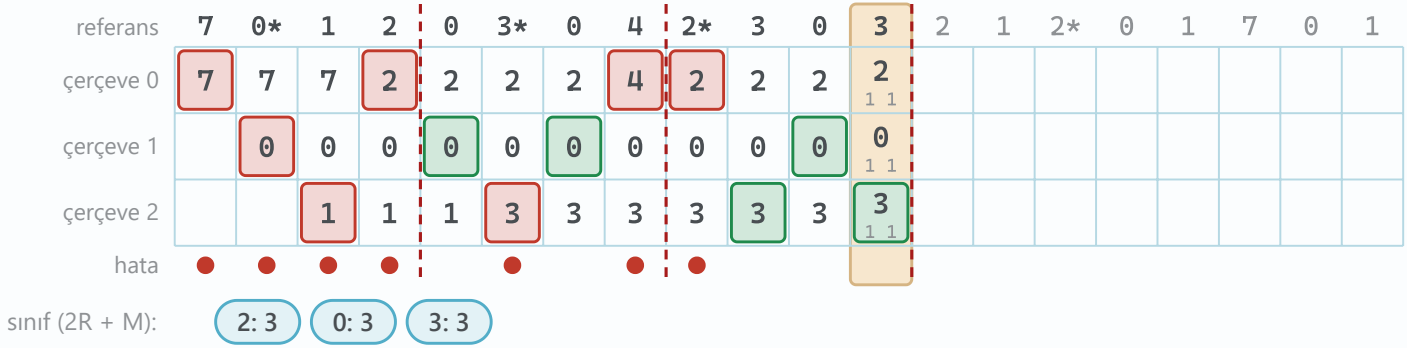


11. referans 0: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

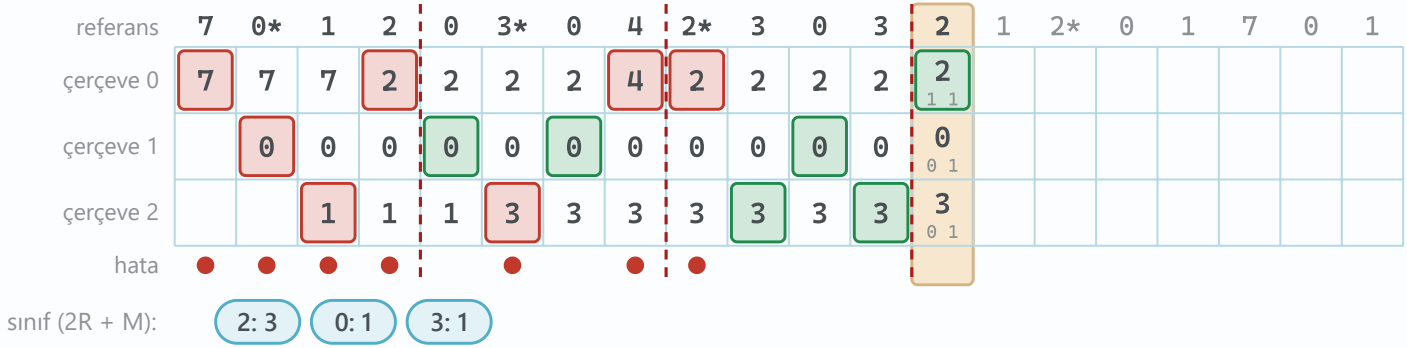


12. referans 3: isabet; R ← 1. Saat tıkı: tüm R ← 0.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

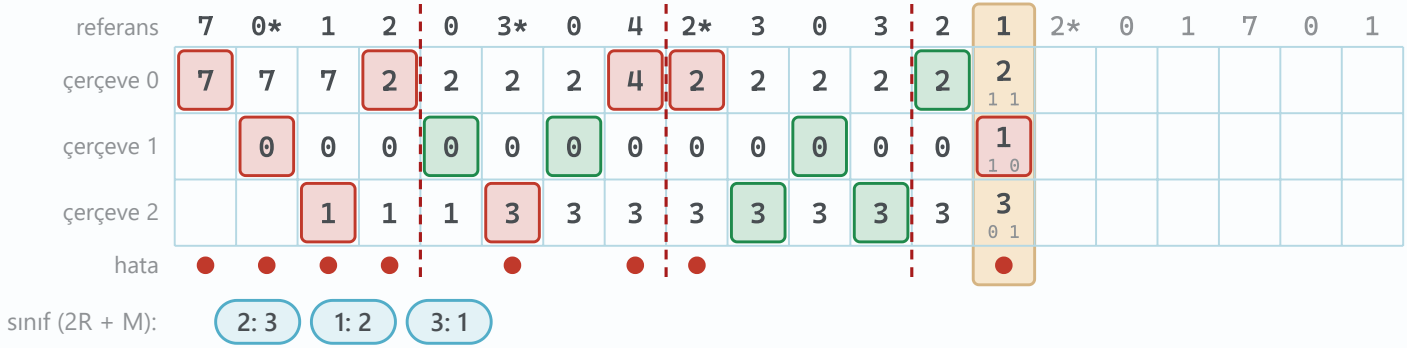


13. referans 2: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

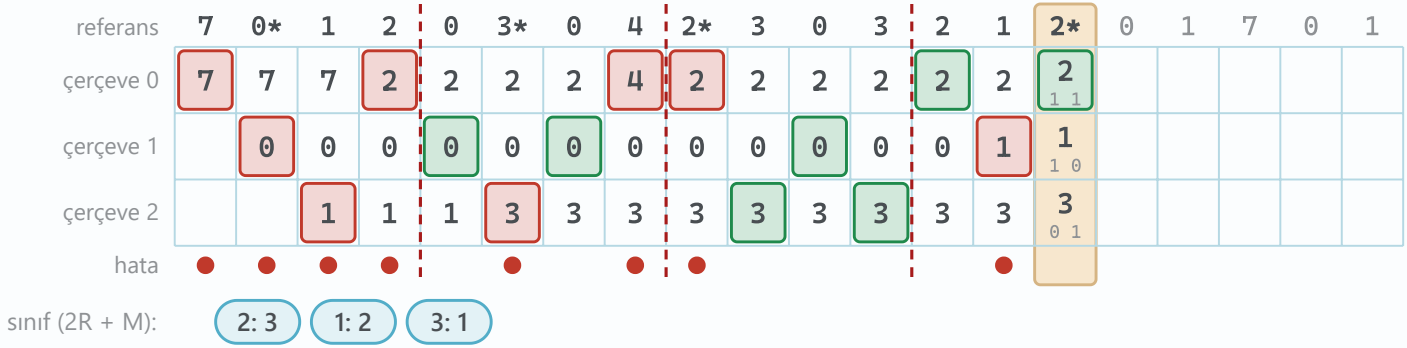


14. referans 1: hata; sınıflar (2→3, 0→1, 3→1); en düşük sınıftan **0** çıkar (kirli → önce diske yazılır). Hata: 8

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

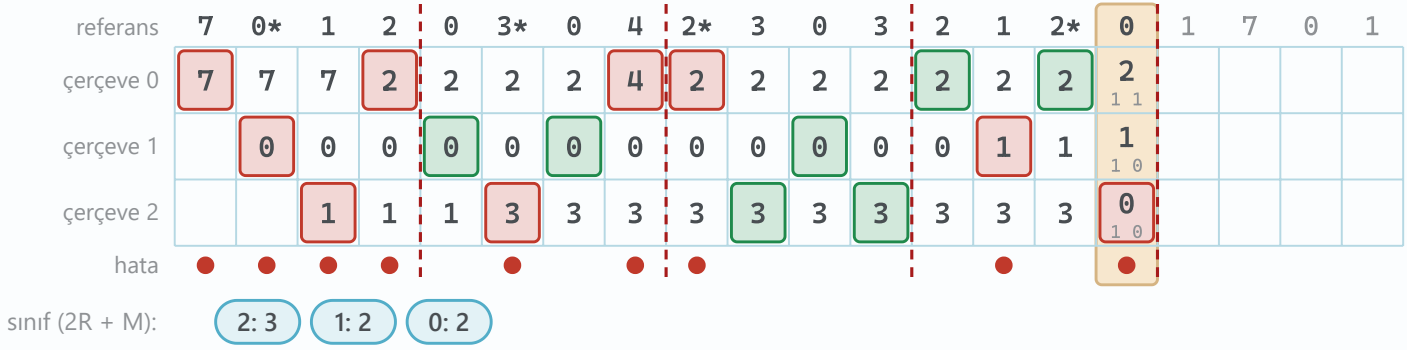


15. referans 2*: isabet; R ← 1, yazma → M ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

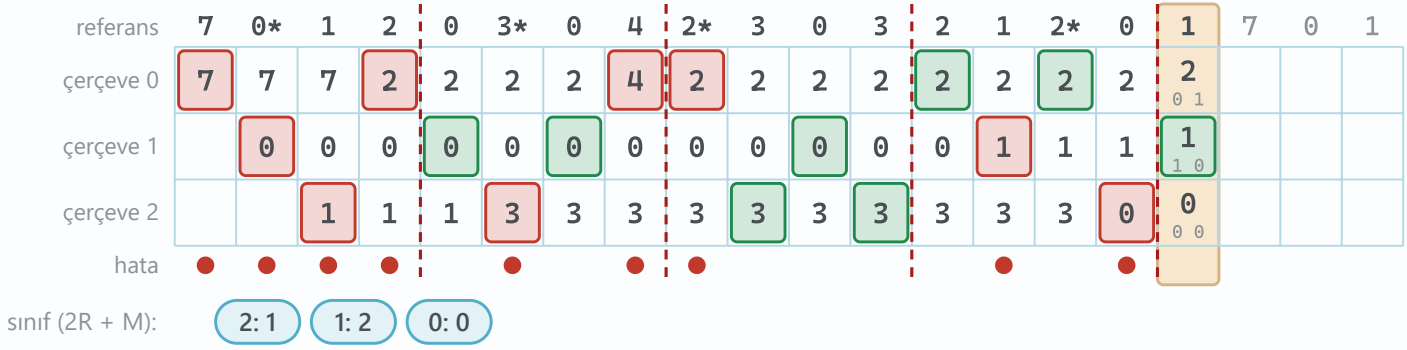


16. referans 0: hata; sınıflar (2→3, 1→2, 3→1); en düşük sınıftan **3** çıkar (kirli → önce diske yazılır). Hata: 9 Saat tıkı: tüm R ← 0.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

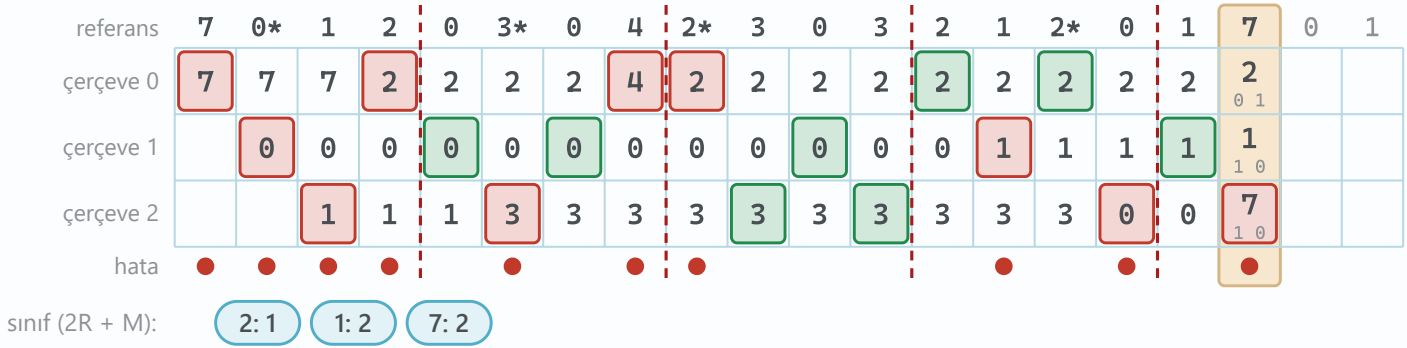


17. referans 1: isabet; R ← 1.

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

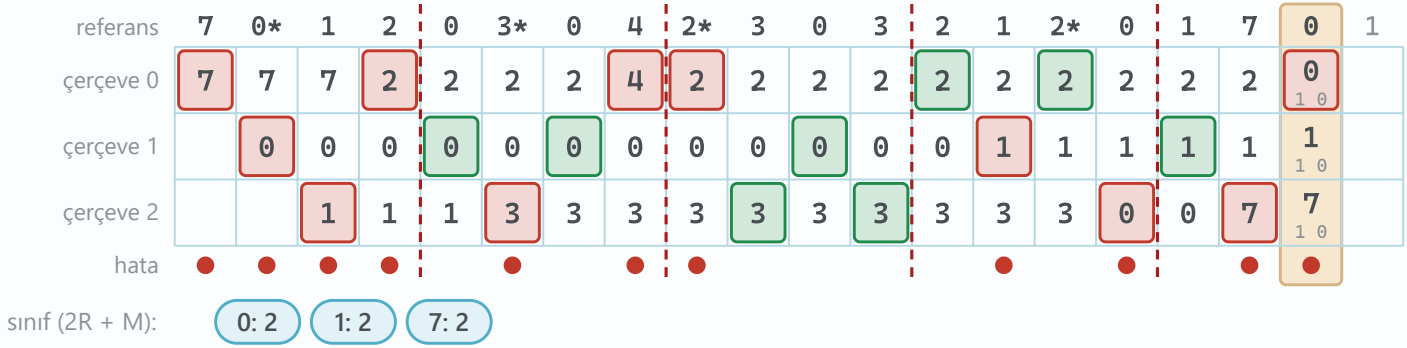


18. referans 7: hata; sınıflar (2→1, 1→2, 0→0); en düşük sınıftan **0** çıkar. Hata: 10

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

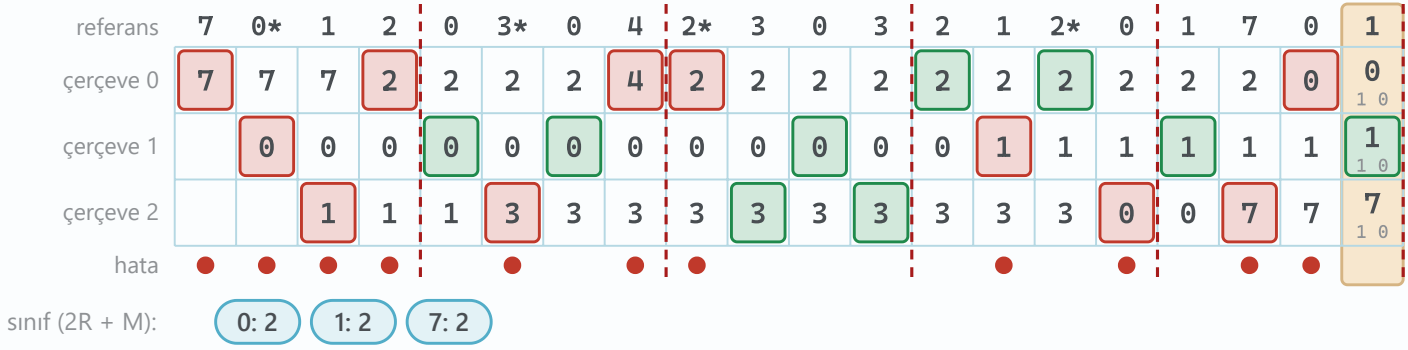


19. referans 0: hata; sınıflar (2→1, 1→2, 7→2); en düşük sınıftan **2** çıkar (kirli → önce diske yazılır). Hata: 11

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

NRU: Yakın Zamanda Kullanılmayan

- Her sayfada **R** (erişildi) ve **M** (değişti) biti; saat kesmesi R'leri periyodik temizler.
- Sınıflar: 0 = (0,0), 1 = (0,1), 2 = (1,0), 3 = (1,1). **En düşük** boş olmayan sınıftan bir sayfa atılır (eşitlikte burada en eski yüklenen).
- Sınıf 1 mümkündür: değişmiş sayfanın R'si tıkla silinmiştir.
- **LRU vs NRU:** LRU tam kullanım *sırasını* izler; NRU yalnızca son saat aralığında kullanıldı mı ve kirli mi diye bakar: kaba ama ucuz.

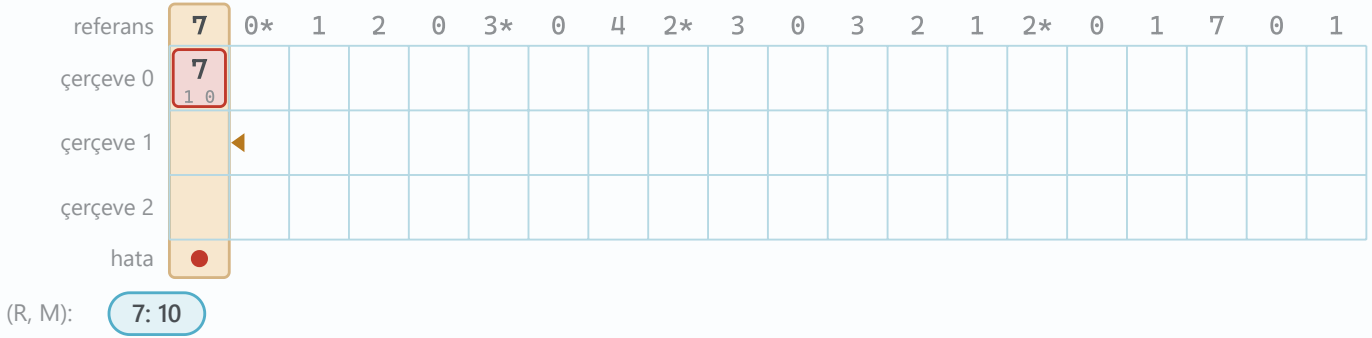


20. referans 1: isabet; R ← 1. Saat tıkı: tüm R ← 0. **Toplam 11 hata, 3 kirli sayfa yazımı.**

* yazma erişimi · alt satırdaki küçük sayılar: R M · tık her 4 referansta bir.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

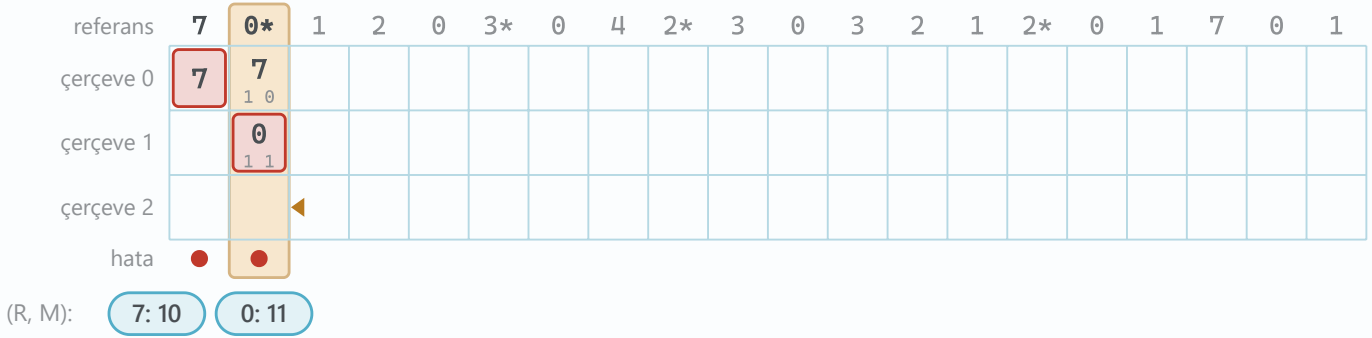


1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

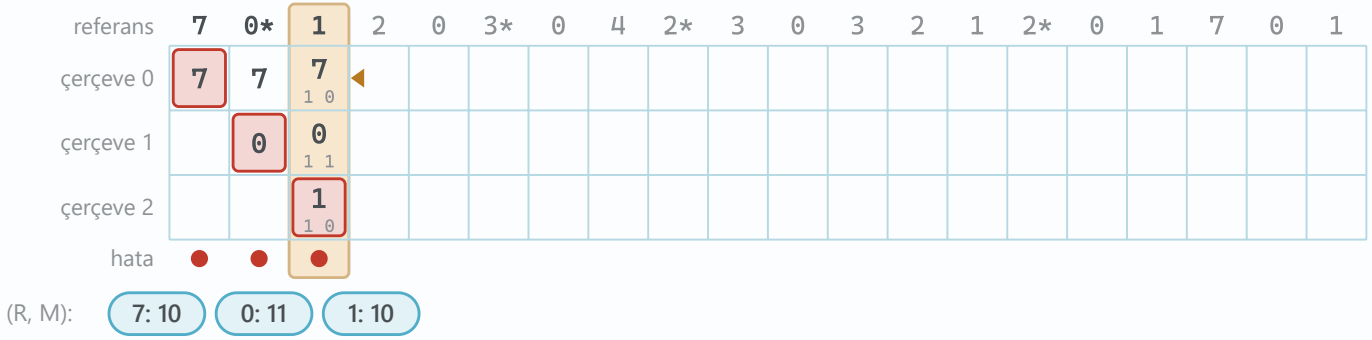


2. referans 0*: hata; boş çerçeve 1'e yüklenir. Hata: 2

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

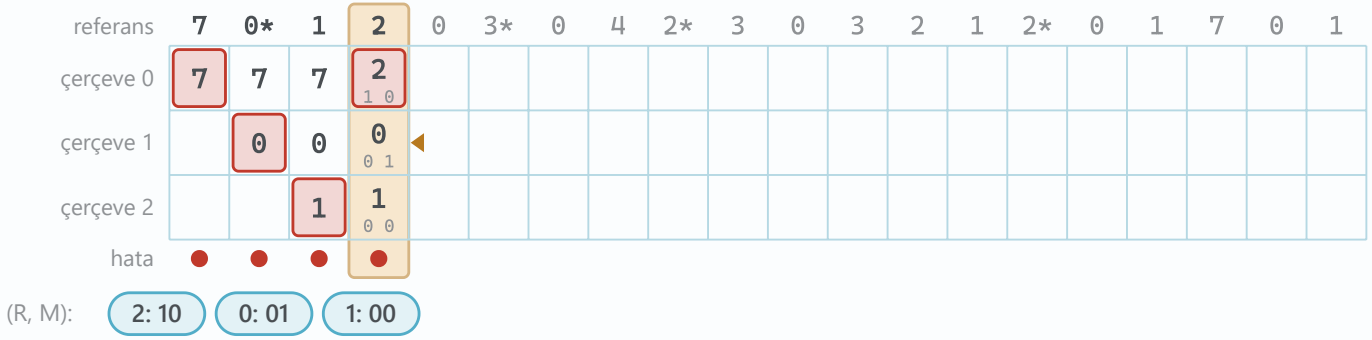


3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

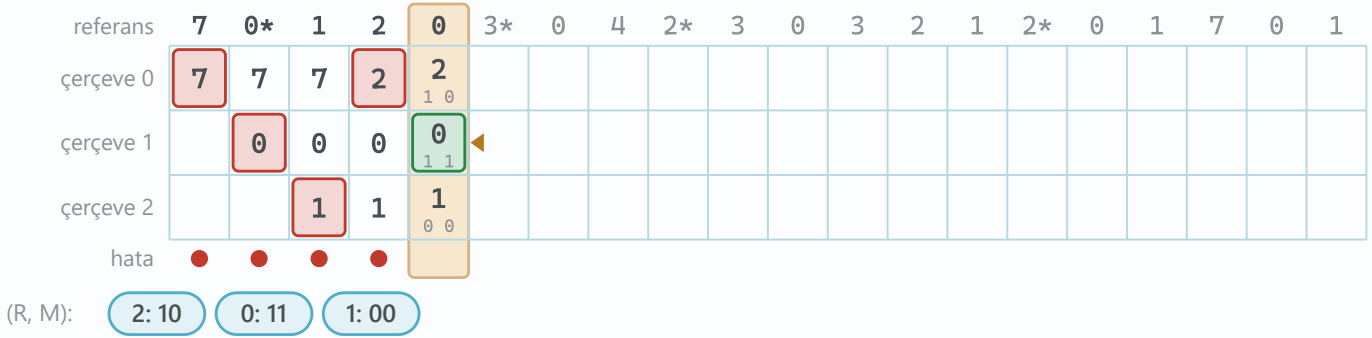


4. referans 2: hata; tarama (2 tur) en düşük sınıftaki **7**'yi seçer. Hata: 4

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

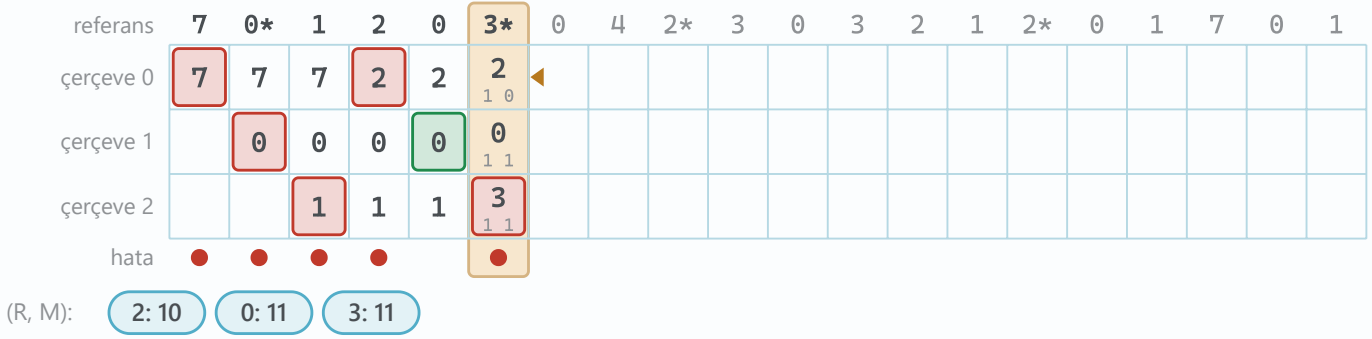


5. referans 0: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

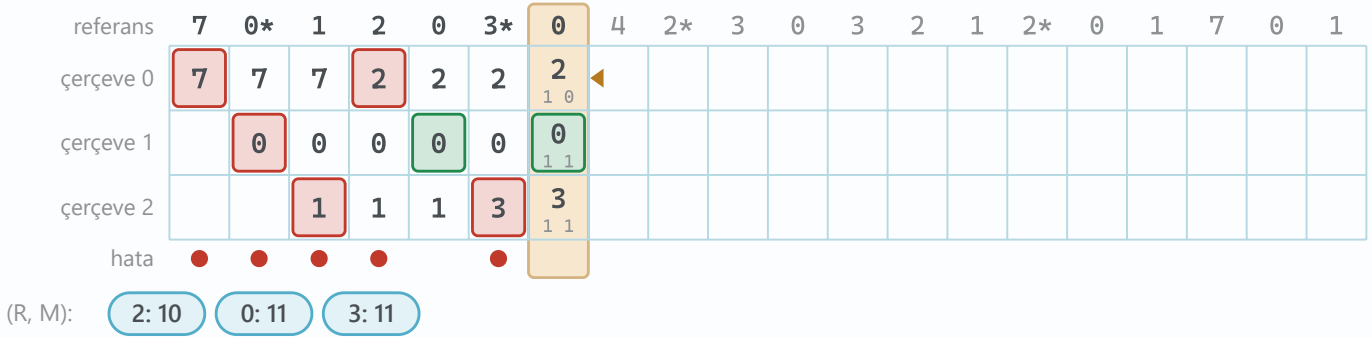


6. referans 3*: hata; tarama (1 tur) en düşük sınıftaki **1**'i seçer. Hata: 5

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

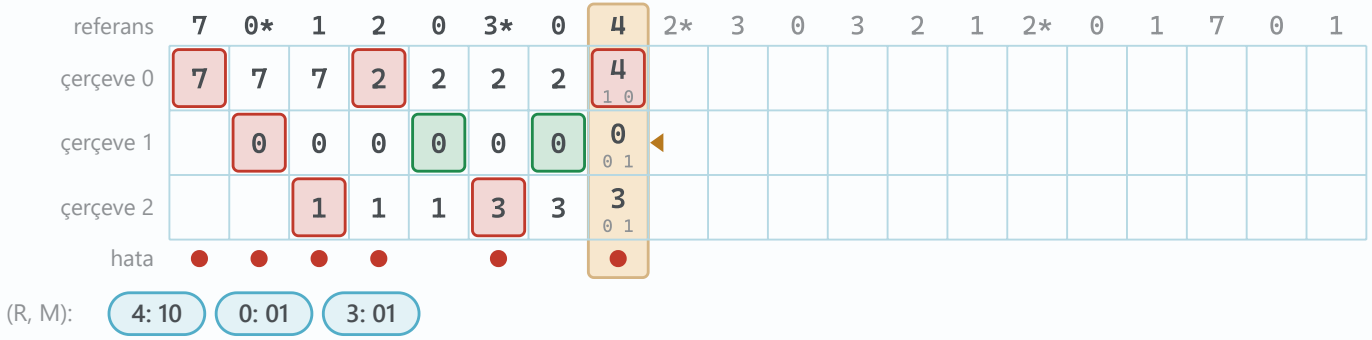


7. referans 0: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

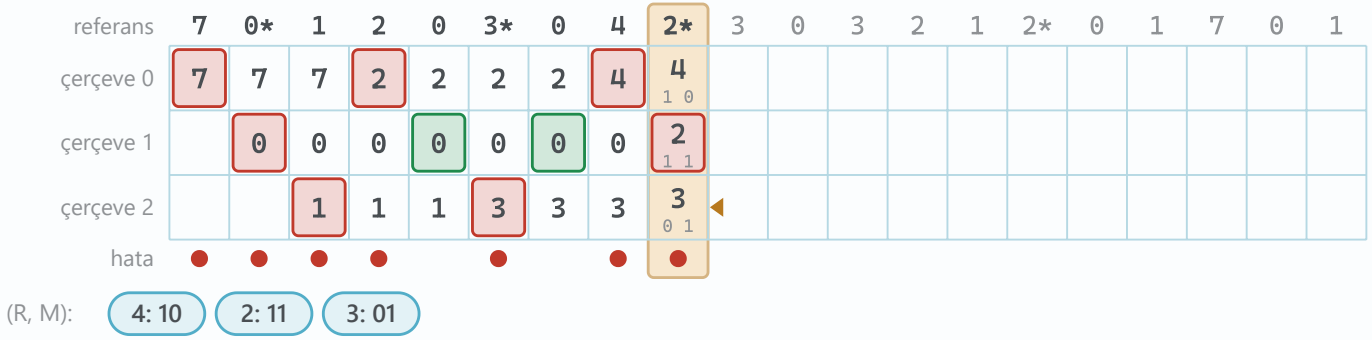


8. referans 4: hata; tarama (2 tur) en düşük sınıftaki **2**'yi seçer. Hata: 6

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

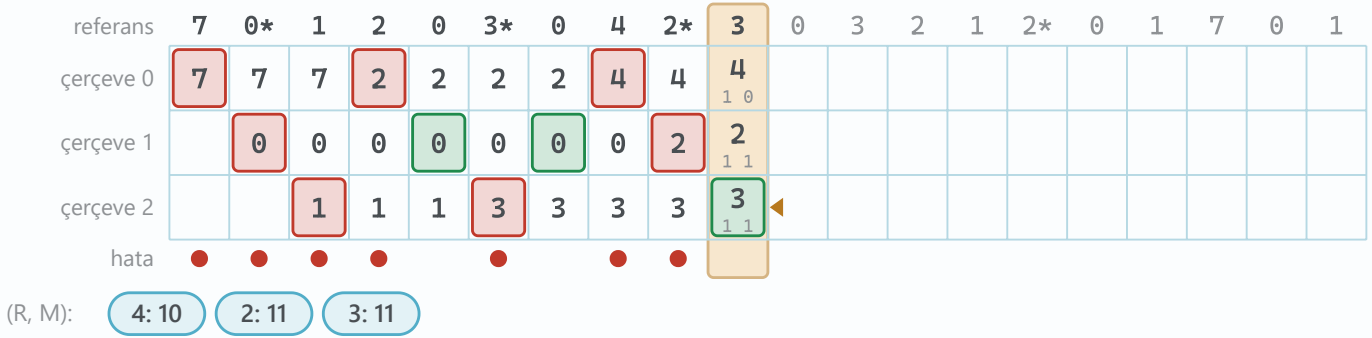


9. referans 2*: hata; tarama (1 tur) en düşük sınıftaki **0**'ı seçer; kirli olduğu için önce diske yazılır. Hata: 7

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

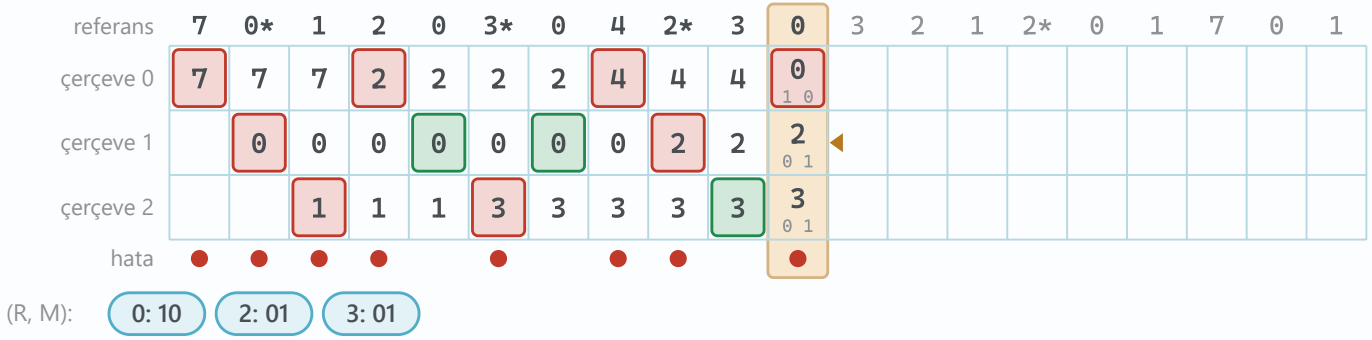


10. referans 3: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

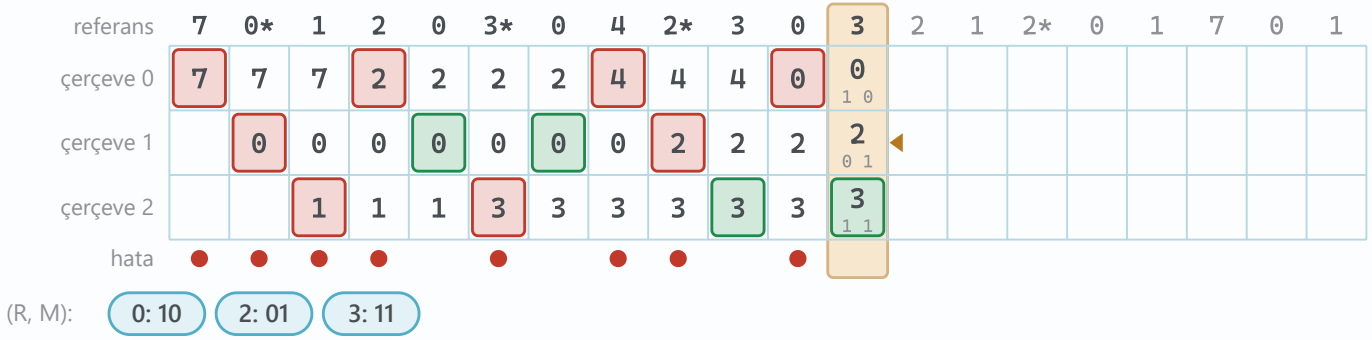


11. referans 0: hata; tarama (2 tur) en düşük sınıftaki **4**'ü seçer. Hata: 8

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

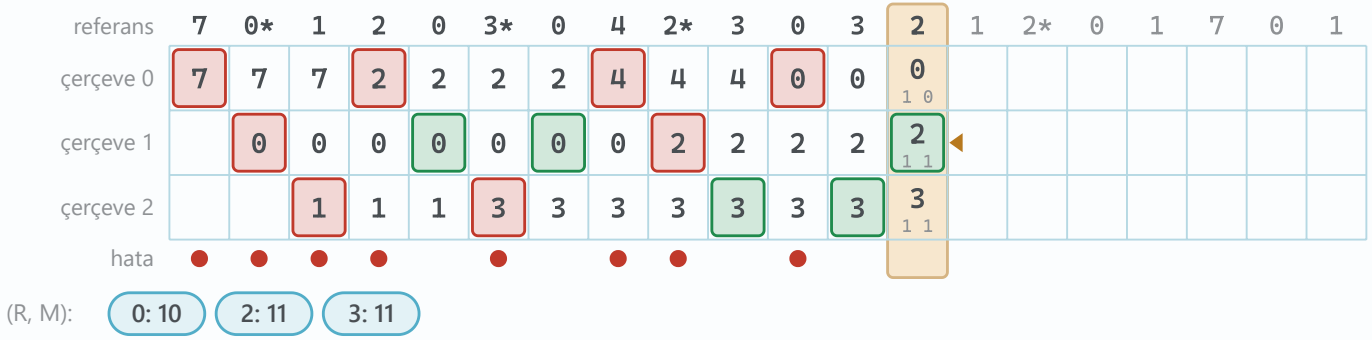


12. referans 3: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

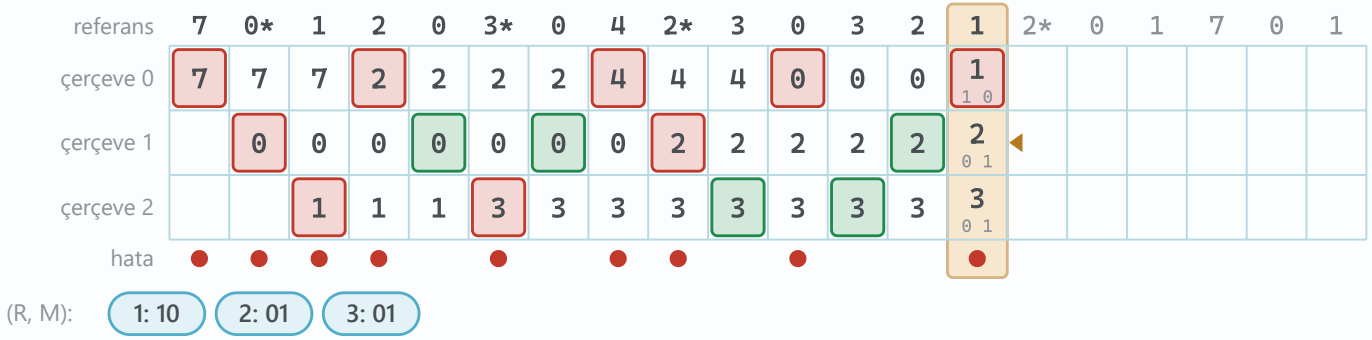


13. referans 2: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

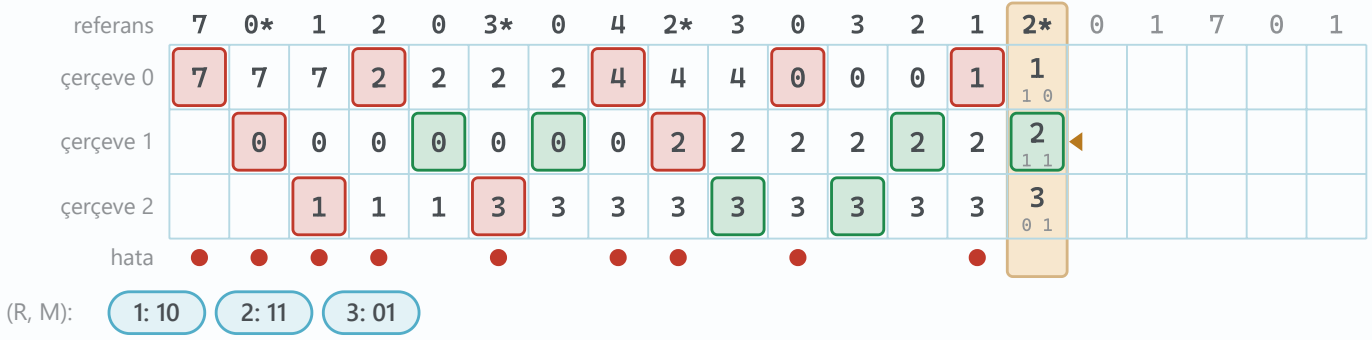


14. referans 1: hata; tarama (2 tur) en düşük sınıftaki **0**'ı seçer. Hata: 9

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

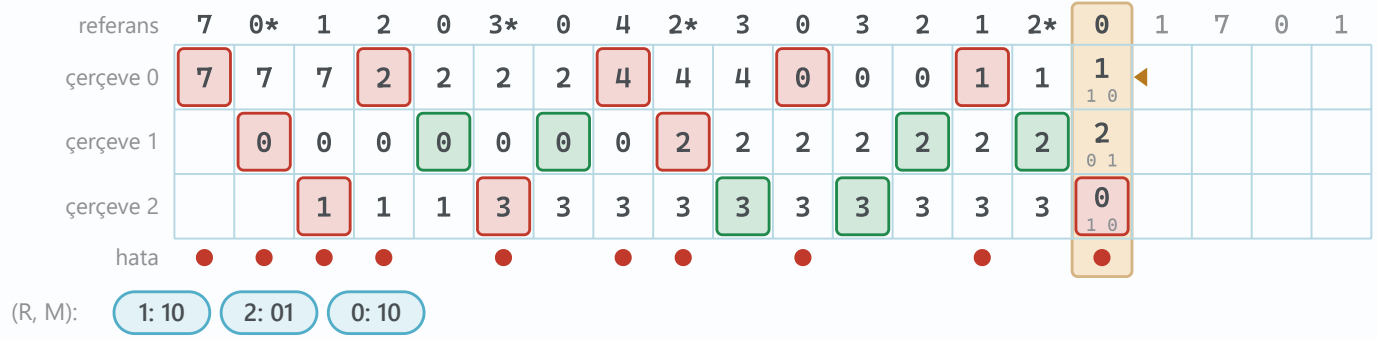


15. referans 2*: isabet; R ← 1, yazma → M ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

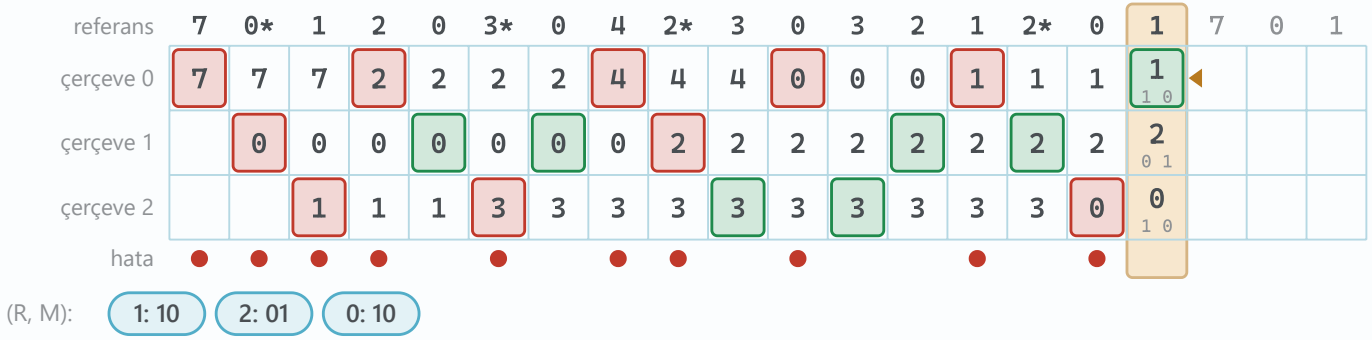


16. referans 0: hata; tarama (1 tur) en düşük sınıftaki **3**'ü seçer; kirli olduğu için önce diske yazılır. Hata: 10

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

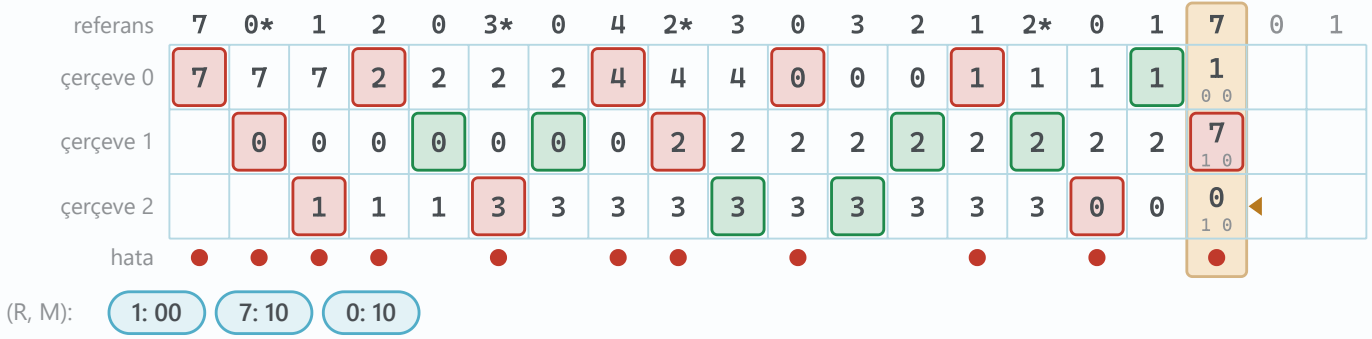


17. referans 1: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

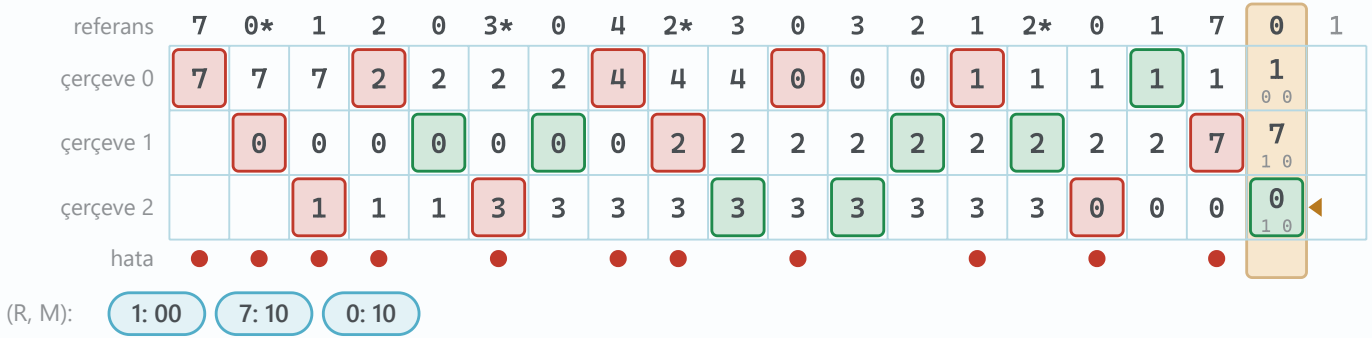


18. referans 7: hata; tarama (1 tur) en düşük sınıftaki **2**'yi seçer; kirli olduğu için önce diske yazılır. Hata: 11

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

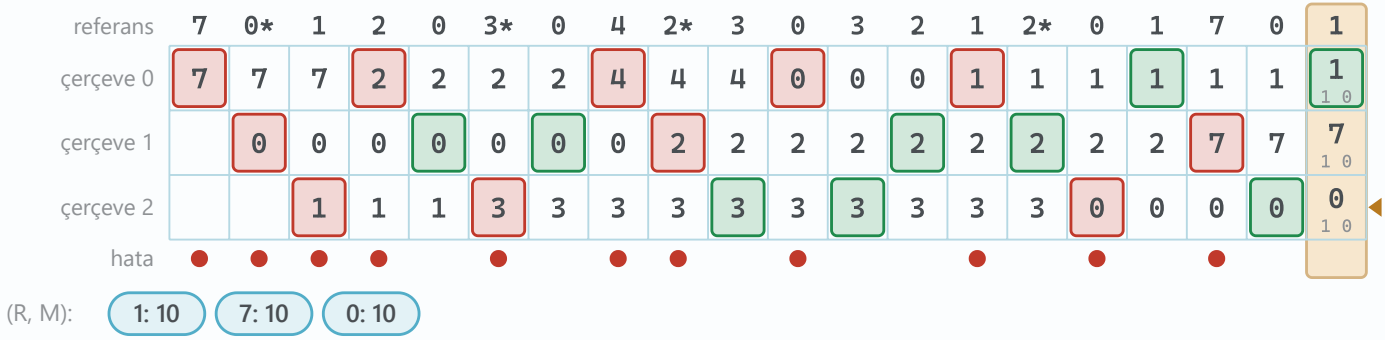


19. referans 0: isabet; R ← 1.

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

Geliştirilmiş Saat: (R, M) Sınıfları

- Saat, R ile birlikte **M**'ye de bakar; temiz sayfayı atmak G/Ç gerektirmez.
- **Tur 1:** elden başlayarak (0,0) aranır, bitlere dokunulmaz.
- **Tur 2:** (0,1) aranır; el geçtiği her sayfanın R'sini 0 yapar. Bulunamazsa tur 1'e dönülür (artık hepsi R = 0).
- Eski macOS ve Silberschatz'ın "enhanced second chance" algoritması.

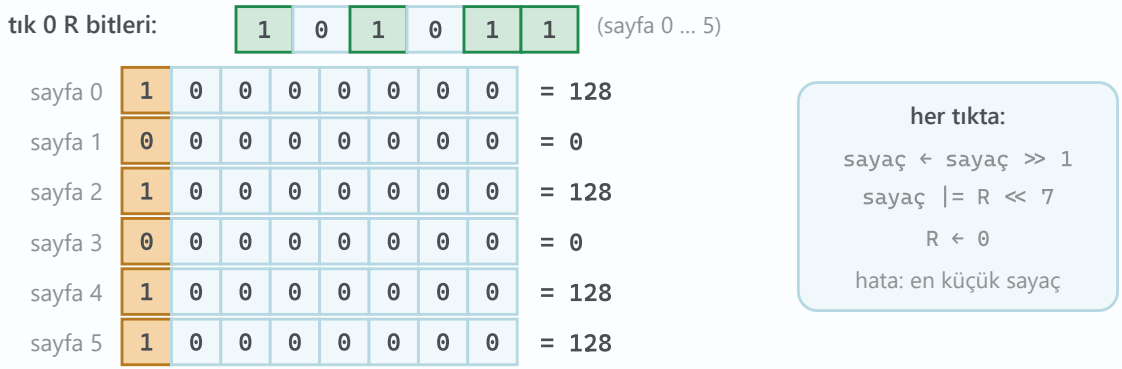


20. referans 1: isabet; R ← 1. **Toplam 11 hata, 3 kirli yazım.**

* yazma erişimi · küçük sayılar: R M. NRU ile aynı dizi; burada R yalnızca el geçerken temizlenir, saat tıkı yoktur.

NFU ve Yaşlandırma (Aging)

- **NFU** (not frequently used): her sayfaya bir yazılım sayacı; her tıkta sayaç += R. En küçük sayaçlı atılır.
- NFU **hiç unutmaz**: eskiden çok kullanılan sayfa sonsuza dek kalır. Erişim zamanı kaydedilmez, yalnızca sayaç tutulur.
- **Yaşlandırma:** önce sağa kaydır, sonra R'yi *en sola* ekle → yakın geçmiş daha ağır basar.
- LRU'ya iyi yaklaşır; farklar: tık içindeki sıra bilinmez, 8 bit yalnızca 8 tık hatırlar.

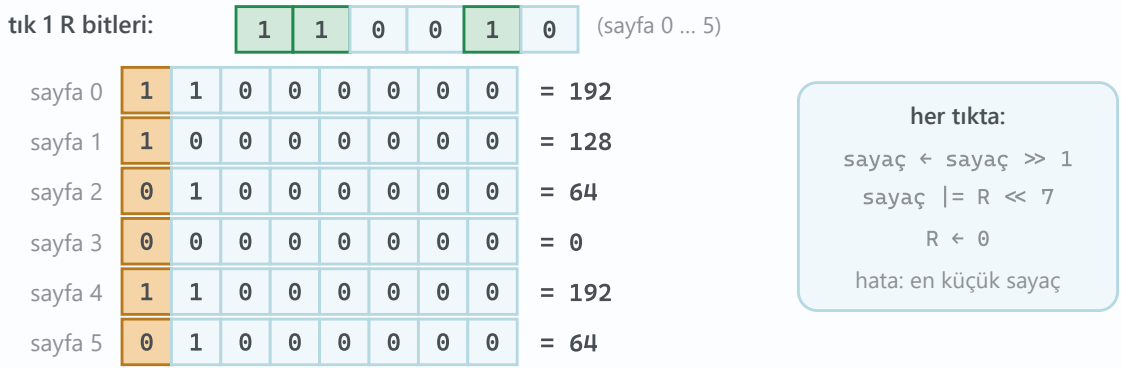


Tık 0: R bitleri 1 0 1 0 1 1 sola eklendi.

Tanenbaum MOS Şekil 3-17; sayaç değerleri R bitlerinden yeniden hesaplandı.

NFU ve Yaşlandırma (Aging)

- **NFU** (not frequently used): her sayfaya bir yazılım sayacı; her tıkta sayaç += R. En küçük sayaçlı atılır.
- NFU **hiç unutmaz**: eskiden çok kullanılan sayfa sonsuza dek kalır. Erişim zamanı kaydedilmez, yalnızca sayaç tutulur.
- **Yaşlandırma:** önce sağa kaydır, sonra R'yi *en sola* ekle → yakın geçmiş daha ağır basar.
- LRU'ya iyi yaklaşır; farklar: tık içindeki sıra bilinmez, 8 bit yalnızca 8 tık hatırlar.

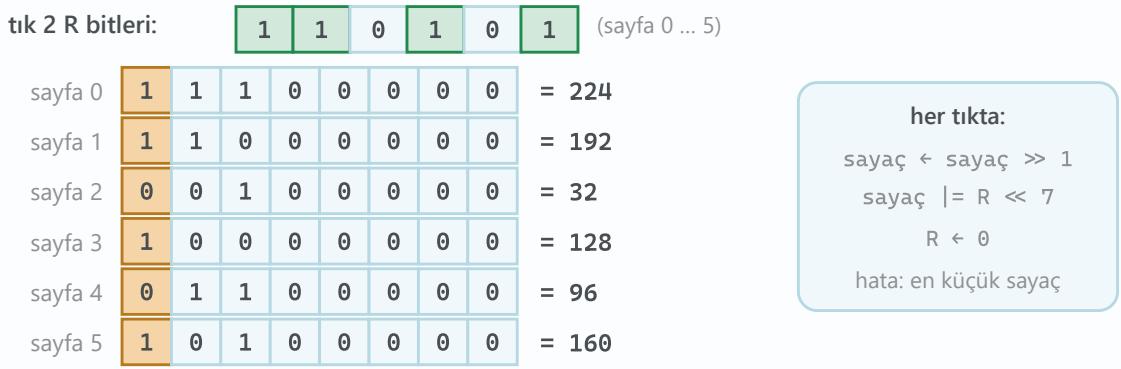


Tık 1: R bitleri 1 1 0 0 1 0 sola eklendi.

Tanenbaum MOS Şekil 3-17; sayaç değerleri R bitlerinden yeniden hesaplandı.

NFU ve Yaşlandırma (Aging)

- **NFU** (not frequently used): her sayfaya bir yazılım sayacı; her tıkta sayaç += R. En küçük sayaçlı atılır.
- NFU **hiç unutmaz**: eskiden çok kullanılan sayfa sonsuza dek kalır. Erişim zamanı kaydedilmez, yalnızca sayaç tutulur.
- **Yaşlandırma:** önce sağa kaydır, sonra R'yi *en sola* ekle → yakın geçmiş daha ağır basar.
- LRU'ya iyi yaklaşır; farklar: tık içindeki sıra bilinmez, 8 bit yalnızca 8 tık hatırlar.

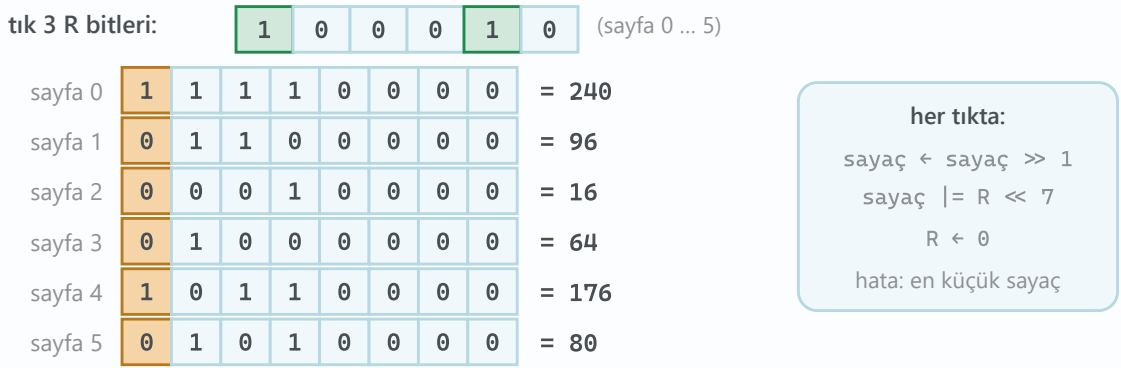


Tık 2: R bitleri 1 1 0 1 0 1 sola eklendi.

Tanenbaum MOS Şekil 3-17; sayaç değerleri R bitlerinden yeniden hesaplandı.

NFU ve Yaşlandırma (Aging)

- **NFU** (not frequently used): her sayfaya bir yazılım sayacı; her tıkta sayaç += R. En küçük sayaçlı atılır.
- NFU **hiç unutmaz**: eskiden çok kullanılan sayfa sonsuza dek kalır. Erişim zamanı kaydedilmez, yalnızca sayaç tutulur.
- **Yaşlandırma:** önce sağa kaydır, sonra R'yi *en sola* ekle → yakın geçmiş daha ağır basar.
- LRU'ya iyi yaklaşır; farklar: tık içindeki sıra bilinmez, 8 bit yalnızca 8 tık hatırlar.

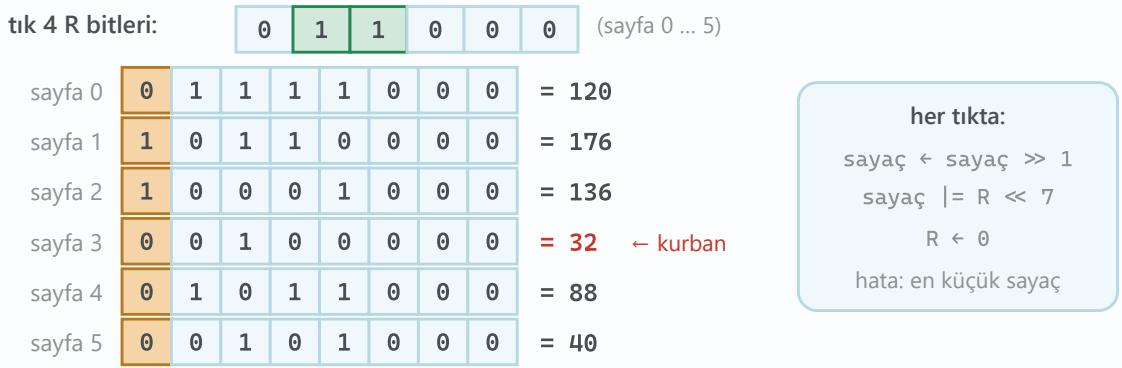


Tık 3: R bitleri 1 0 0 0 1 0 sola eklendi.

Tanenbaum MOS Şekil 3-17; sayaç değerleri R bitlerinden yeniden hesaplandı.

NFU ve Yaşlandırma (Aging)

- **NFU** (not frequently used): her sayfaya bir yazılım sayacı; her tıkta sayaç += R. En küçük sayaçlı atılır.
- NFU **hiç unutmaz**: eskiden çok kullanılan sayfa sonsuza dek kalır. Erişim zamanı kaydedilmez, yalnızca sayaç tutulur.
- **Yaşlandırma:** önce sağa kaydır, sonra R'yi *en sola* ekle → yakın geçmiş daha ağır basar.
- LRU'ya iyi yaklaşır; farklar: tık içindeki sıra bilinmez, 8 bit yalnızca 8 tık hatırlar.

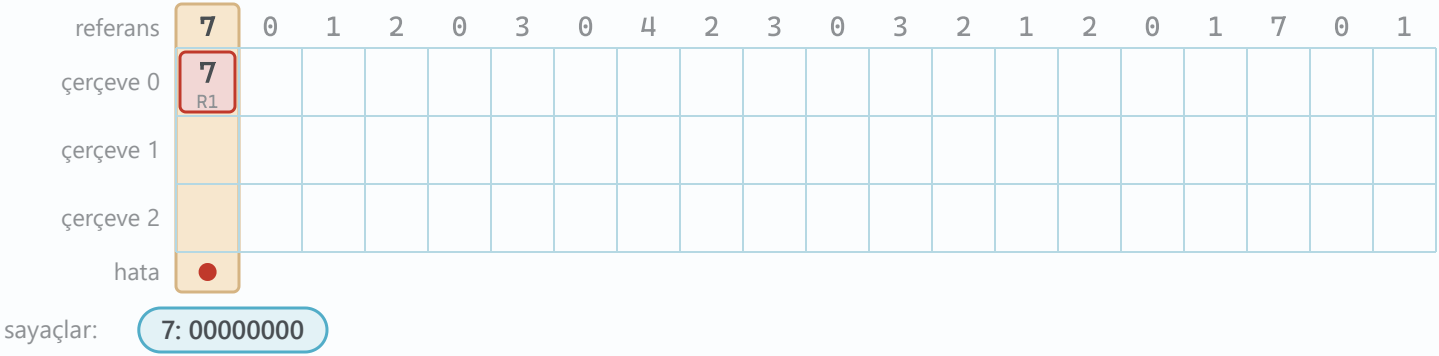


Tık 4: R bitleri 0 1 1 0 0 0 sola eklendi. Şimdi bir hata olsa sayaç değeri en küçük olan **sayfa 3** atılır.

Tanenbaum MOS Şekil 3-17; sayaç değerleri R bitlerinden yeniden hesaplandı.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

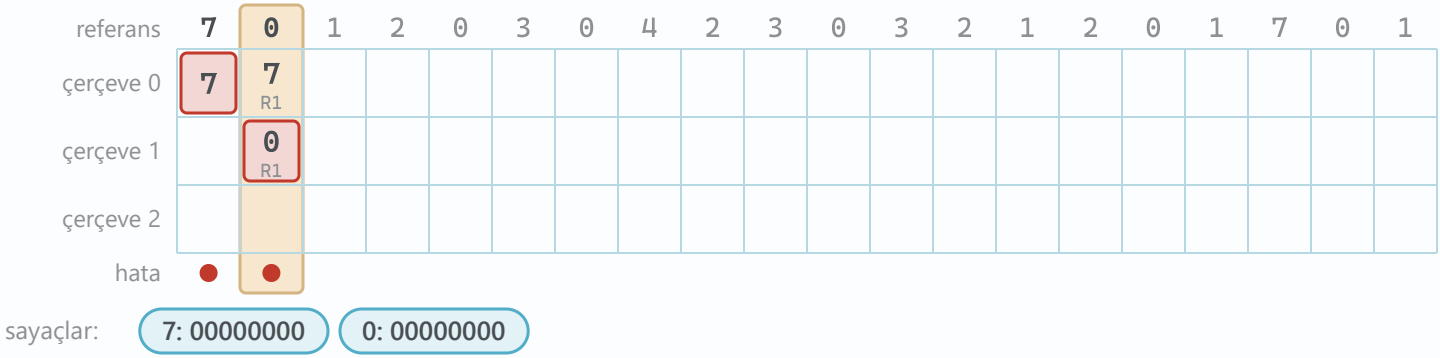


1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

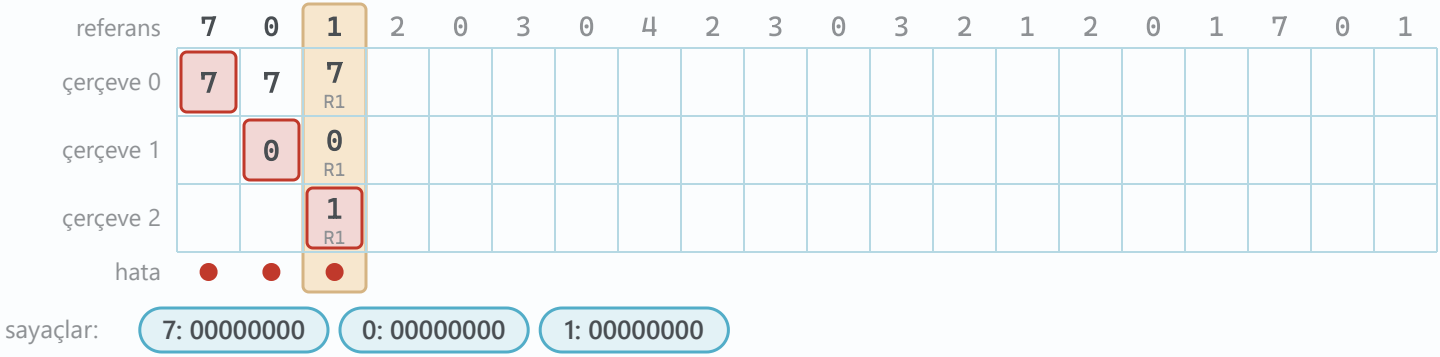


2. referans 0: hata; boş çerçeve 1'e yüklenir. Hata: 2

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

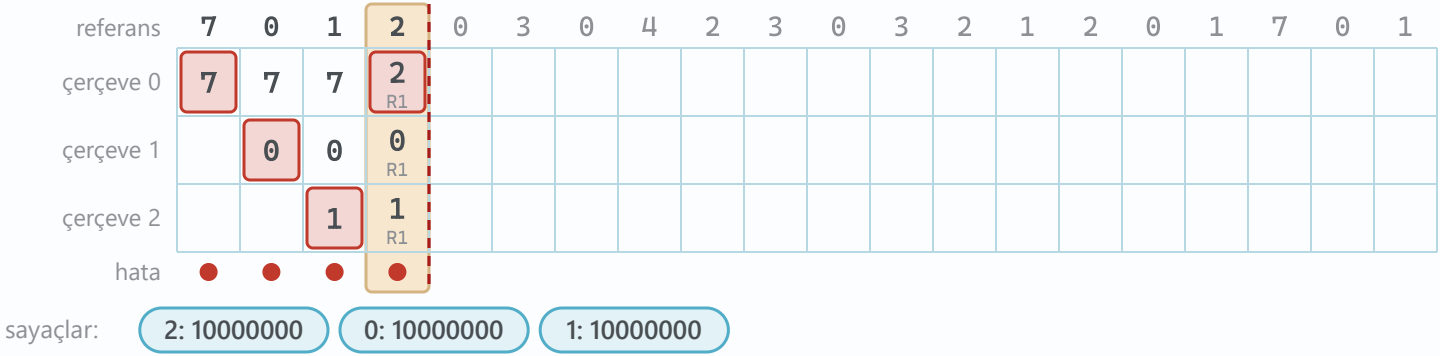


3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

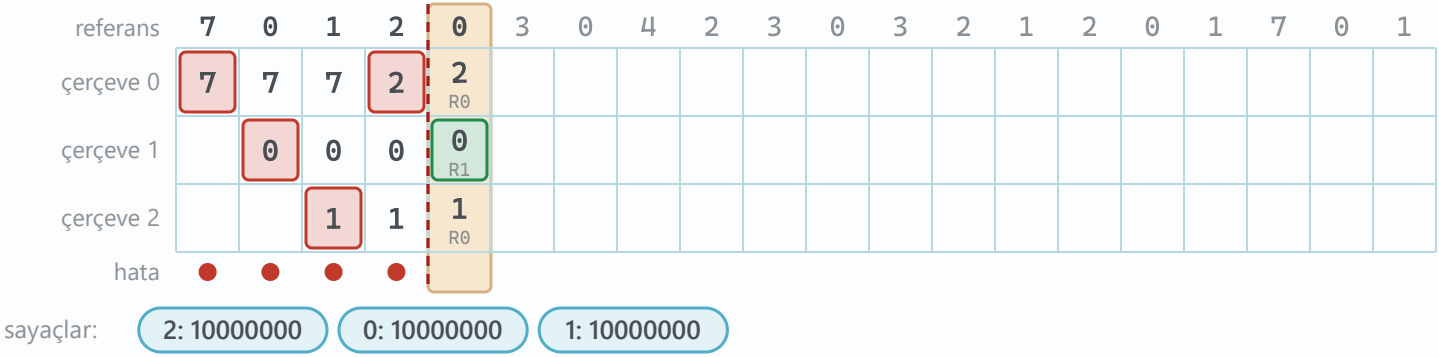


4. referans 2: hata; R ve sayaç birlikte (R en anlamlı bit): 7: 100000000, 0: 100000000, 1: 100000000 → en küçük **7** çıkar. Hata: 4 Tık: sayaçlar kaydırıldı, R'ler sola eklendi.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

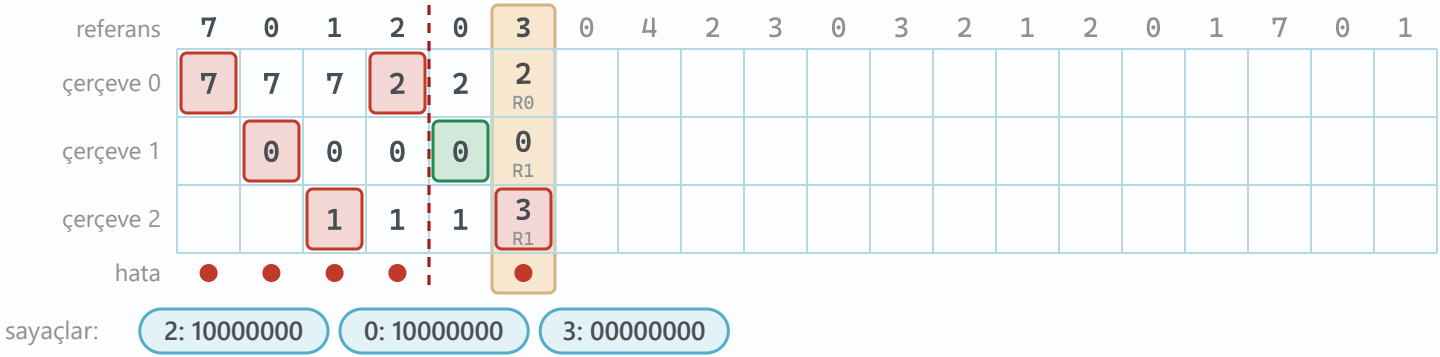


5. referans 0: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

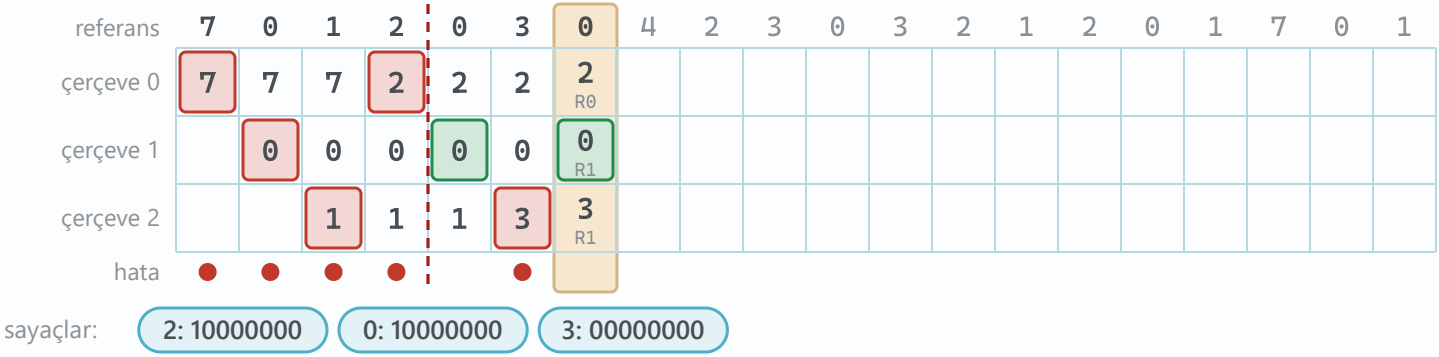


6. referans 3: hata; R ve sayaç birlikte (R en anlamlı bit): 2: 010000000, 0: 110000000, 1: 010000000 → en küçük **1** çıkar. Hata: 5

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

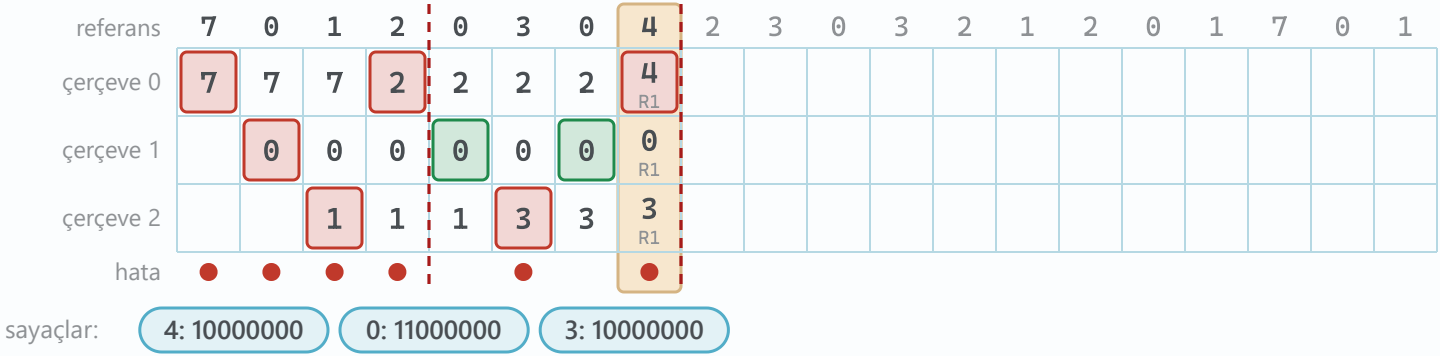


7. referans 0: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

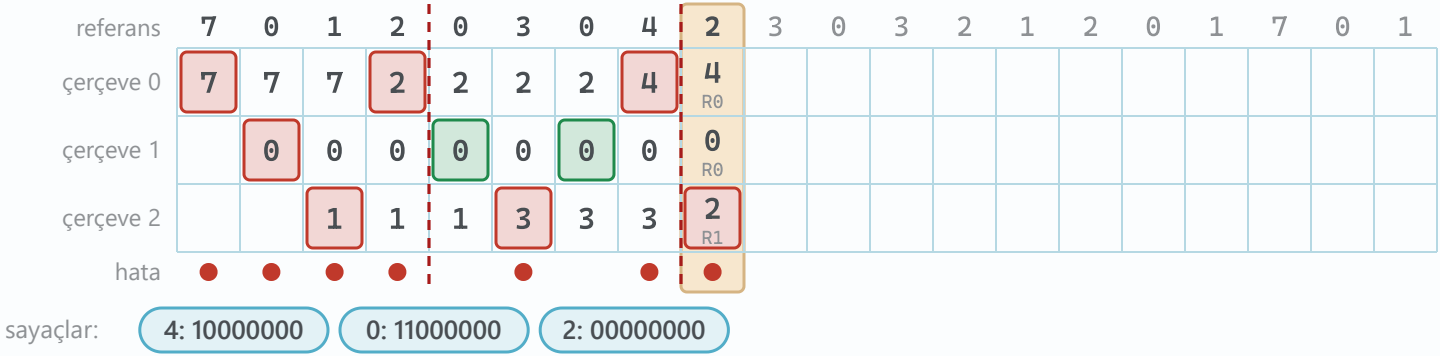


8. referans 4: hata; R ve sayaç birlikte (R en anlamlı bit): 2: 010000000, 0: 110000000, 3: 100000000 → en küçük **2** çıkar. Hata: 6 Tık: sayaçlar kaydırıldı, R'ler sola eklendi.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

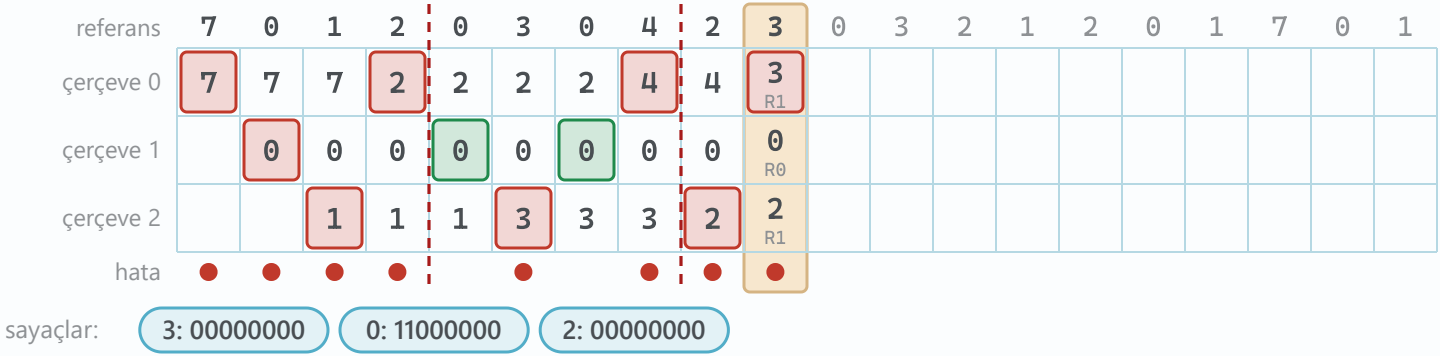


9. referans 2: hata; R ve sayaç birlikte (R en anlamlı bit): 4: 010000000, 0: 011000000, 3: 010000000 → en küçük **3** çıkar. Hata: 7

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

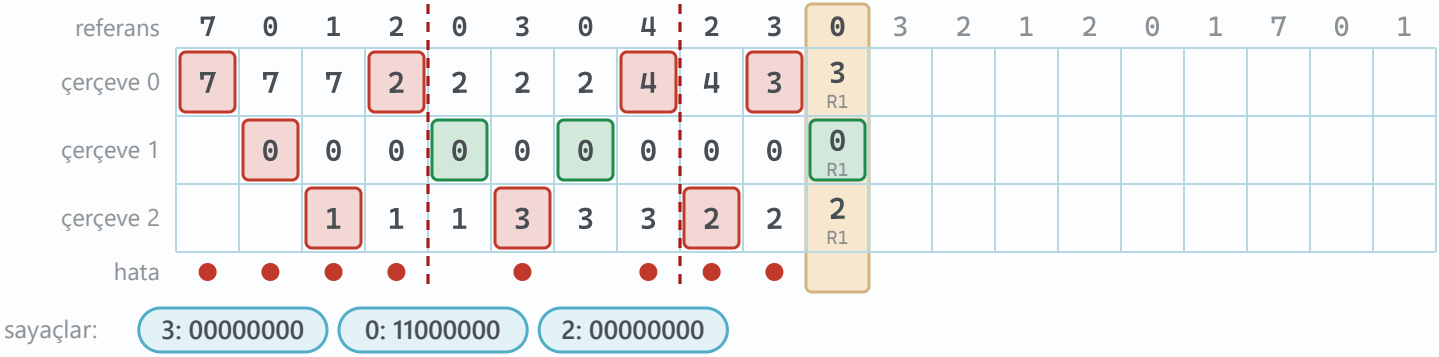


10. referans 3: hata; R ve sayaç birlikte (R en anlamlı bit): 4: 010000000, 0: 011000000, 2: 100000000 → en küçük **4** çıkar. Hata: 8

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

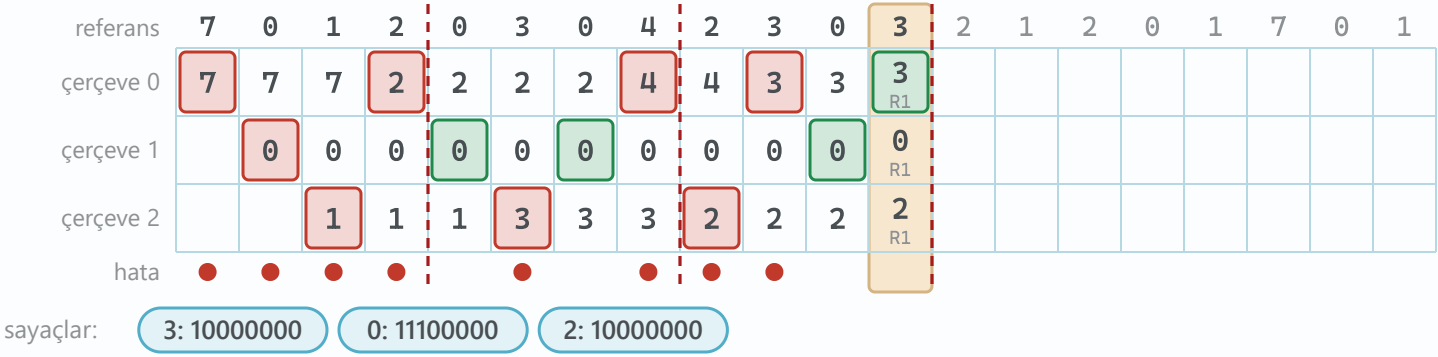


11. referans 0: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

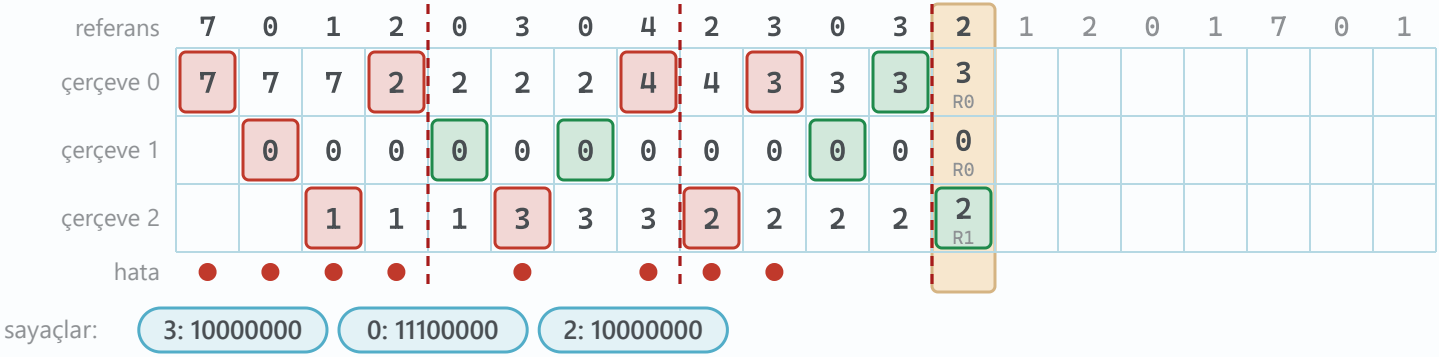


12. referans 3: isabet; R ← 1. Tık: sayaçlar kaydırıldı, R'ler sola eklendi.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

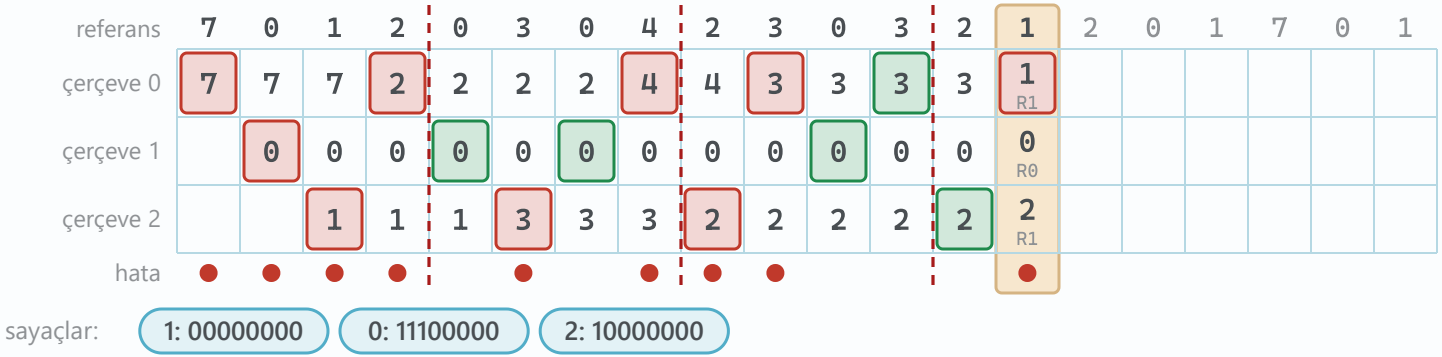


13. referans 2: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

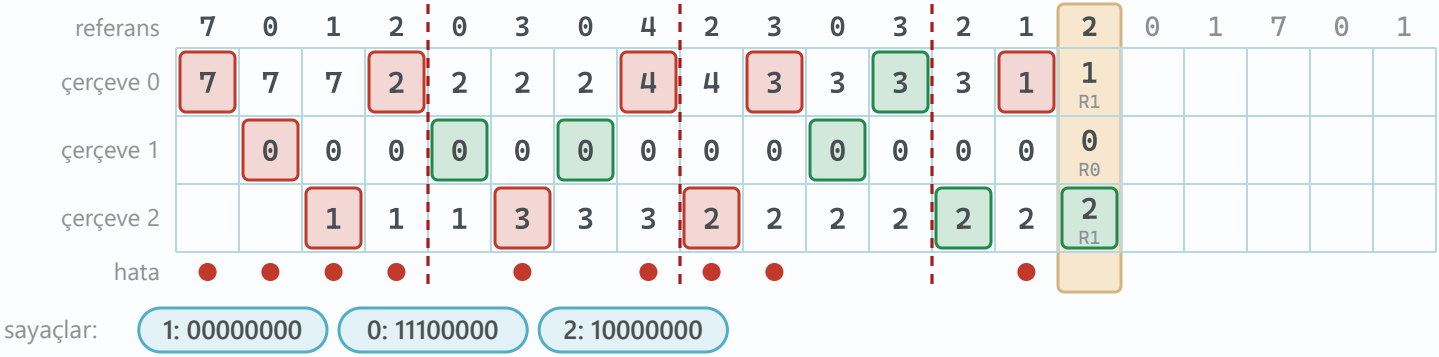


14. referans 1: hata; R ve sayaç birlikte (R en anlamlı bit): 3: 010000000, 0: 011100000, 2: 110000000 → en küçük **3** çıkar. Hata: 9

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

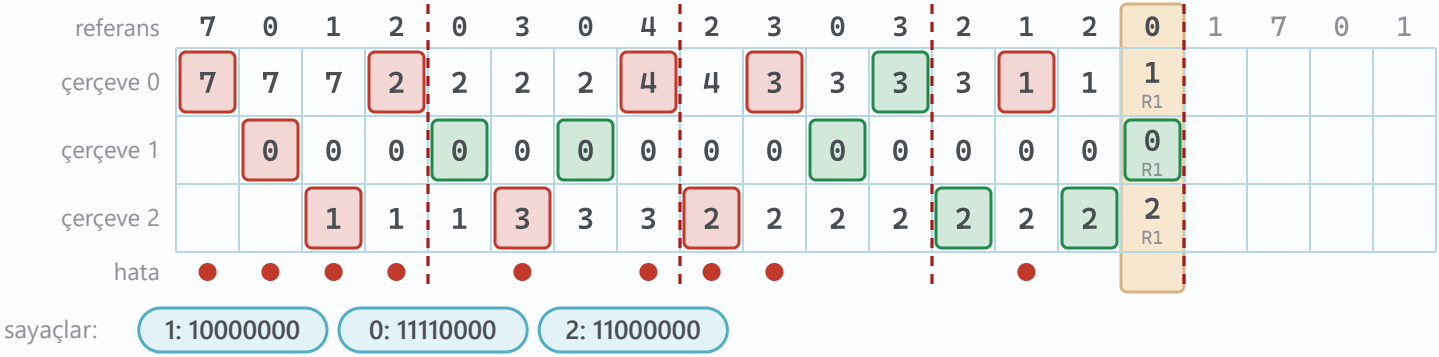


15. referans 2: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

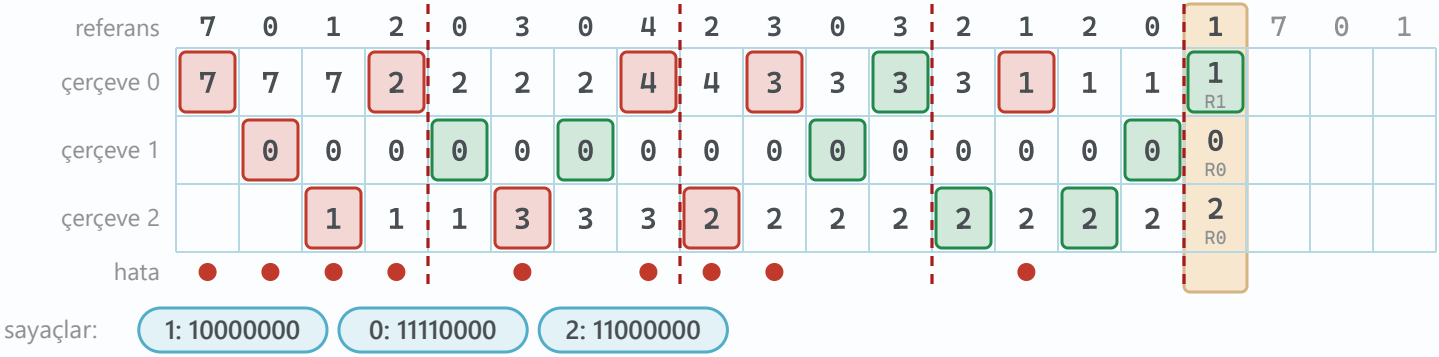


16. referans 0: isabet; R ← 1. Tık: sayaçlar kaydırıldı, R'ler sola eklendi.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

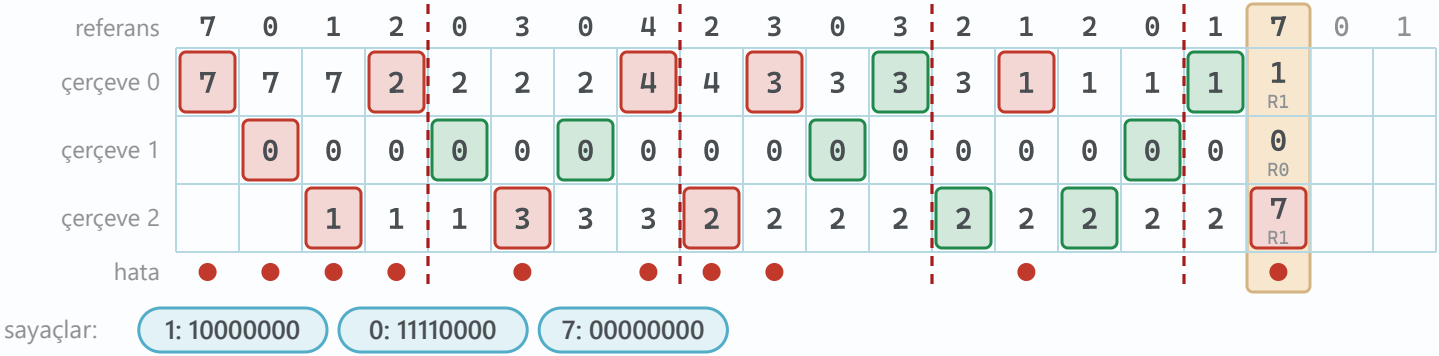


17. referans 1: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

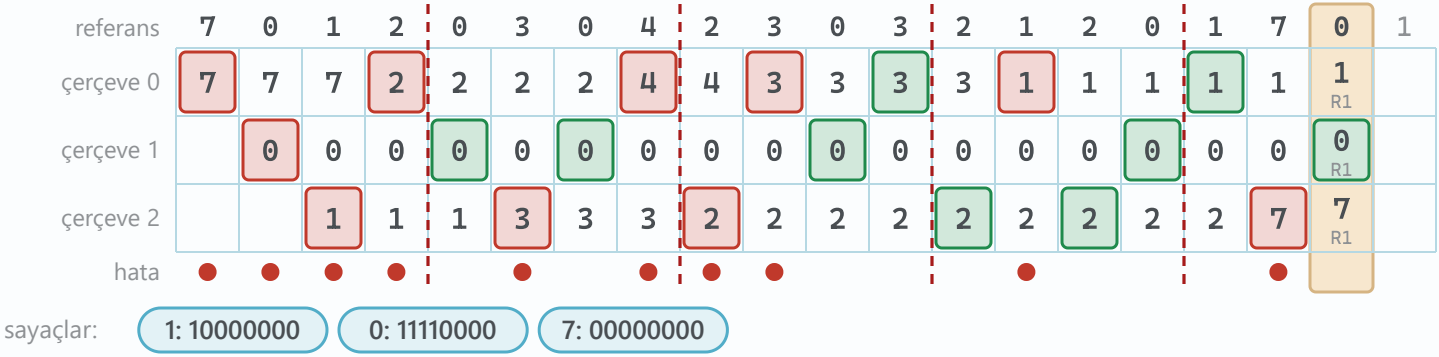


18. referans 7: hata; R ve sayaç birlikte (R en anlamlı bit): 1: 110000000, 0: 011110000, 2: 011000000 → en küçük **2** çıkar. Hata: 10

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

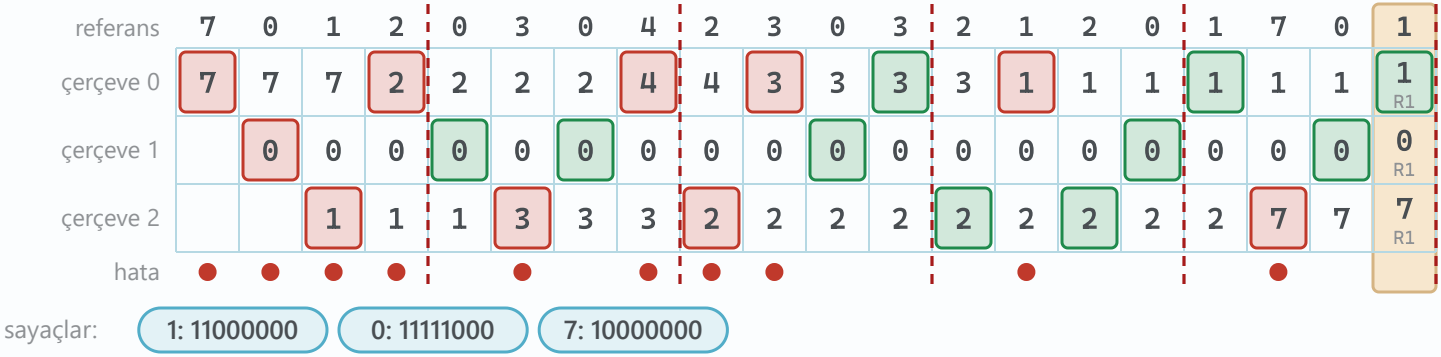


19. referans 0: isabet; R ← 1.

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Yaşlandırma: Ortak Dizide

Aynı dizi, 3 çerçeve, her 4 referansta bir saat tıkı. Yeni yüklenen sayfanın sayacı 0, R'si 1'dir; karşılaştırmada **bu aralığın R biti en anlamlı bit** sayılır (yoksa yeni sayfa hemen atılırdı).

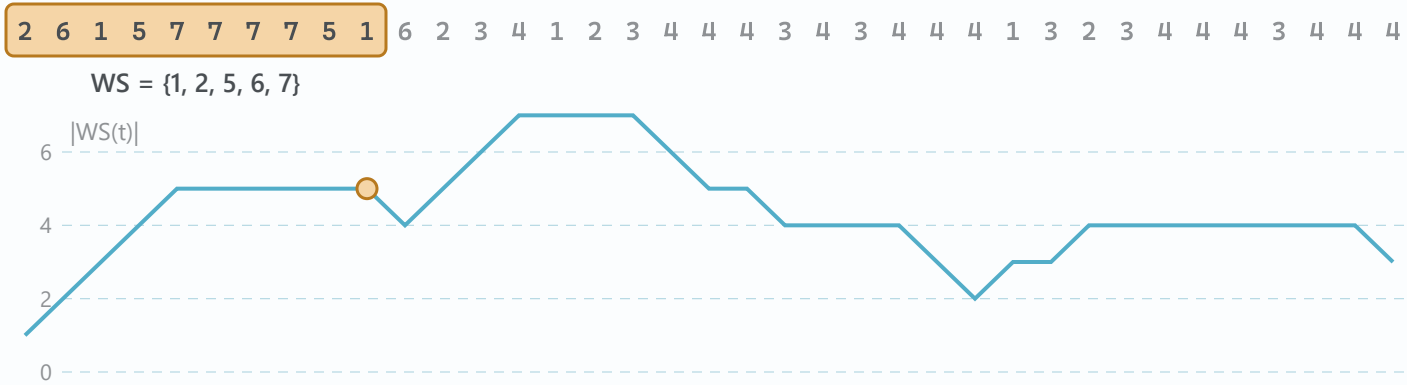


20. referans 1: isabet; R ← 1. Tık: sayaçlar kaydırıldı, R'ler sola eklendi. **Toplam 10 hata.**

Kesikli çizgi: saat tıkı. Tek bir kısa dizi algoritmaları sıralamaya yetmez; tık sıklığı ve eşitlik kuralı sonucu değiştirir.

Çalışma Kümesi Modeli

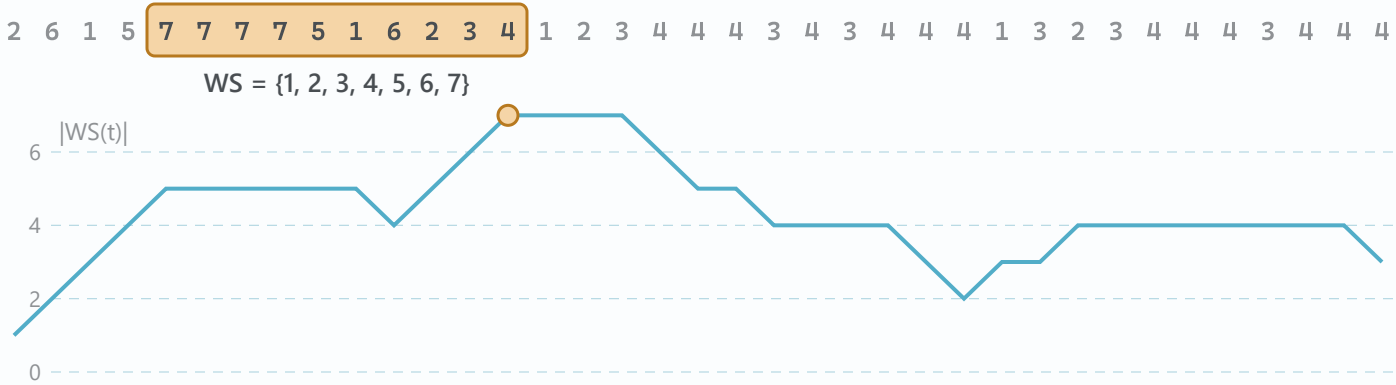
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 10: pencere 1–10. referanslar; çalışma kümesi {1, 2, 5, 6, 7}, boyut **5**. (Silberschatz'taki t_1)

Çalışma Kümesi Modeli

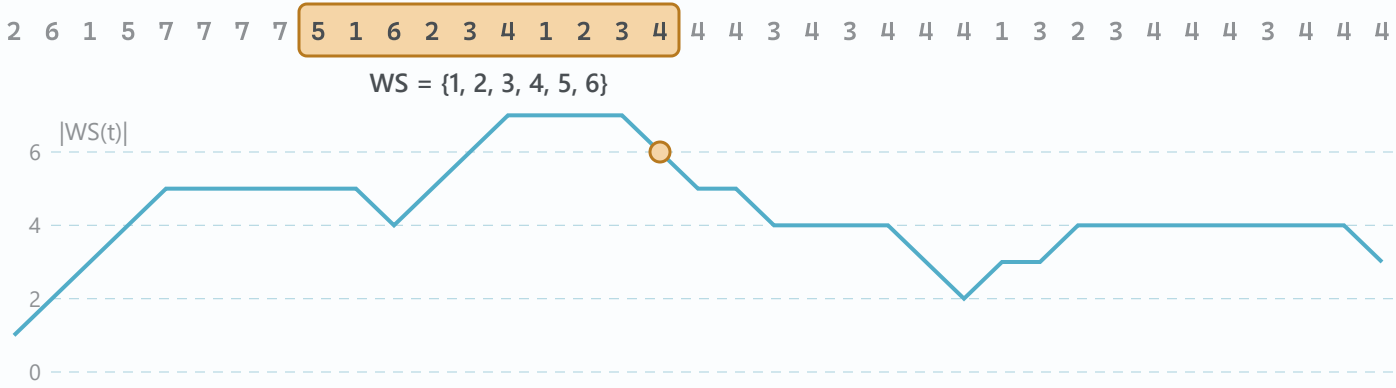
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 14: pencere 5–14. referanslar; çalışma kümesi {1, 2, 3, 4, 5, 6, 7}, boyut **7**.

Çalışma Kümesi Modeli

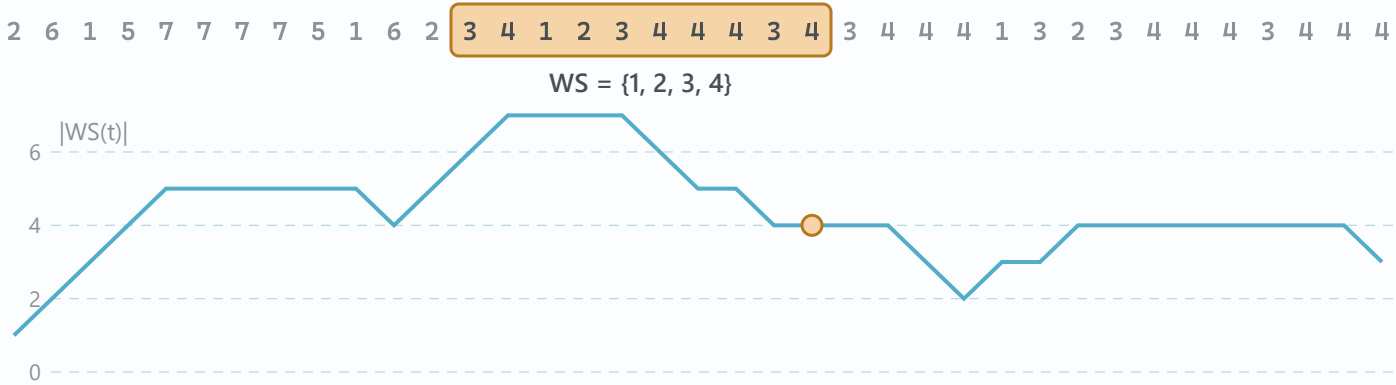
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 18: pencere 9–18. referanslar; çalışma kümesi {1, 2, 3, 4, 5, 6}, boyut **6**.

Çalışma Kümesi Modeli

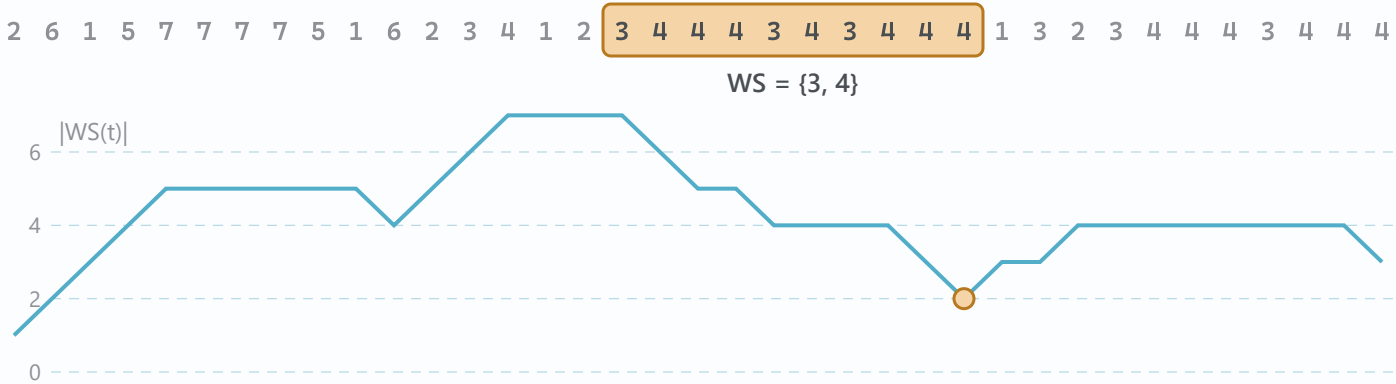
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 22: pencere 13–22. referanslar; çalışma kümesi {1, 2, 3, 4}, boyut **4**.

Çalışma Kümesi Modeli

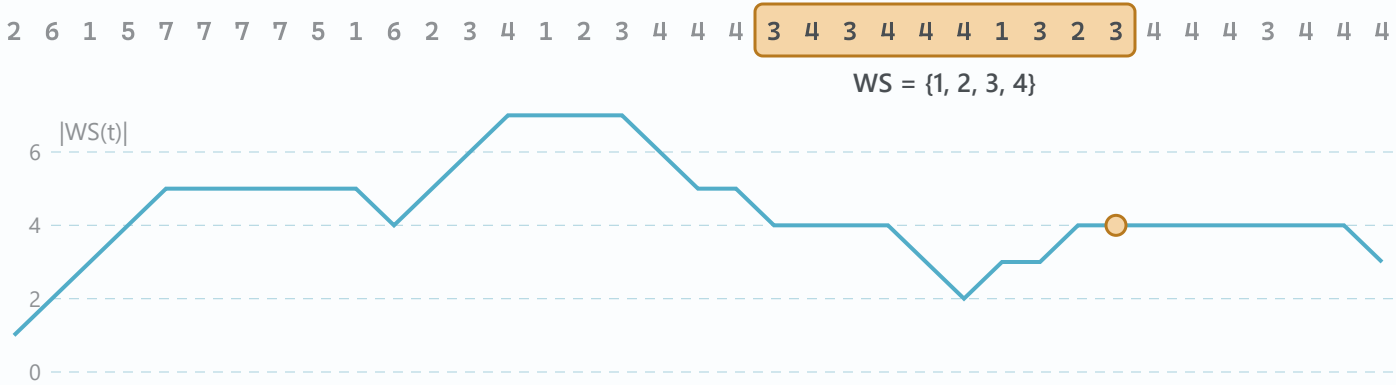
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 26: pencere 17–26. referanslar; çalışma kümesi {3, 4}, boyut **2**. (t_2): süreç yeni bir **yerelliğe** geçti.

Çalışma Kümesi Modeli

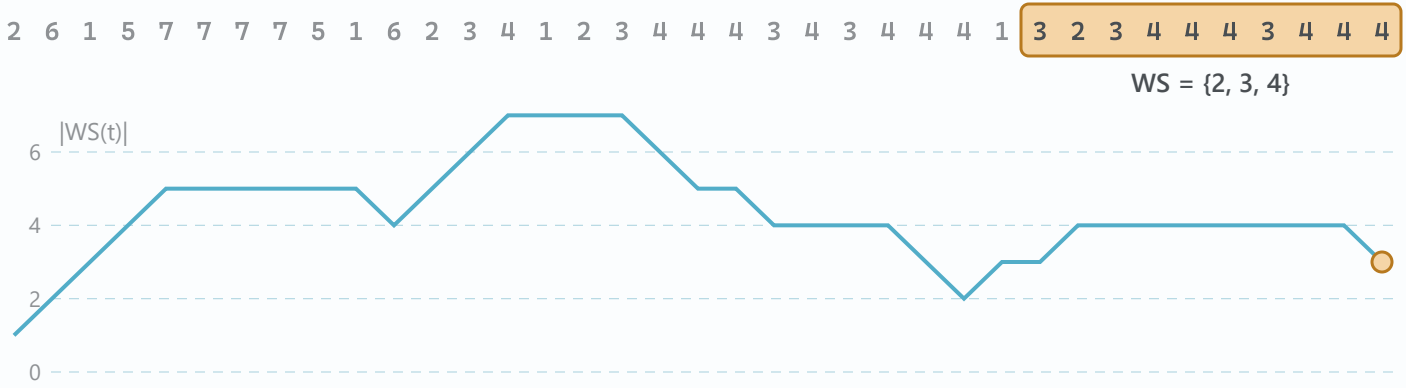
- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 30: pencere 21–30. referanslar; çalışma kümesi {1, 2, 3, 4}, boyut **4**.

Çalışma Kümesi Modeli

- **Yerellik:** süreç bir süre küçük bir sayfa grubunu kullanır, sonra başka gruba geçer.
- **Çalışma kümesi** WS(t, Δ): son Δ *bellek referansında* kullanılan sayfaların kümesi (Denning, 1968).
- Δ küçükse yerellik kaçırılır, büyükse birkaç yerellik üst üste biner; Δ = ∞ tüm program.
- Çalışma kümesi bellekteyse az hata olur; değilse **thrashing**.



t = 37: pencere 28–37. referanslar; çalışma kümesi {2, 3, 4}, boyut **3**.

Çalışma Kümesi Algoritması

- Kesin tanım (son Δ referans) her erişimde kaydırılan bir yazmaç ister: çok pahalı. Yaklaşım: **son τ sanal zaman** içinde kullanılan sayfalar.

son kullanım	R
2084	1
2003	1
1980	1
1213	0
2014	1
2020	1
2032	1
1620	0

2204 şimdiki sanal zaman

tüm sayfaları tara:
R = 1 → zaman ← şimdi, R ← 0
R = 0, yaş > τ → çıkar
R = 0, yaş ≤ τ → en eskiyi hatırla
hiçbiri yoksa: en eski (tercihen temiz) sayfa

Her girdide **son kullanım zamanı** ve R biti. Zaman **sanal** zamandır: sürecin kendi CPU süresi (tıklarla ölçülür). τ = 500 alalım.

Tanenbaum MOS Şekil 3-20 (yeniden çizim). Dezavantaj: her hatada *tüm* tabloyu taramak.

Çalışma Kümesi Algoritması

- Kesin tanım (son Δ referans) her erişimde kaydırılan bir yazmaç ister: çok pahalı. Yaklaşım: **son τ sanal zaman** içinde kullanılan sayfalar.

son kullanım	R
2204	0
2204	0
2204	0
1213	0
2204	0
2204	0
2204	0
1620	0

2204 şimdiki sanal zaman

tüm sayfaları tara:

R = 1 → zaman ← şimdi, R ← 0

R = 0, yaş > τ → çıkar

R = 0, yaş ≤ τ → en eskiyi hatırla

hiçbiri yoksa: en eski (tercihen temiz) sayfa

Sayfa hatasında tablo taranır: R = 1 olan sayfalar bu tıkta kullanılmış → zamanları **2204** yapılır, R'leri temizlenir.

Tanenbaum MOS Şekil 3-20 (yeniden çizim). Dezavantaj: her hatada *tüm* tabloyu taramak.

Çalışma Kümesi Algoritması

- Kesin tanım (son Δ referans) her erişimde kaydırılan bir yazmaç ister: çok pahalı. Yaklaşım: **son τ sanal zaman** içinde kullanılan sayfalar.

son kullanım	R	
2204	0	
2204	0	
2204	0	
1213	0	yaş 991 > τ
2204	0	
2204	0	
2204	0	
1620	0	yaş 584 > τ

2204 şimdiki sanal zaman

tüm sayfaları tara:
R = 1 → zaman ← şimdi, R ← 0
R = 0, yaş > τ → çıkar
R = 0, yaş ≤ τ → en eskiyi hatırla
hiçbiri yoksa: en eski (tercihen temiz) sayfa

R = 0 olanların yaşı hesaplanır: 2204 − 1213 = 991 ve 2204 − 1620 = 584; ikisi de τ'dan büyük → çalışma kümesinde **değiller**.

Tanenbaum MOS Şekil 3-20 (yeniden çizim). Dezavantaj: her hatada *tüm* tabloyu taramak.

Çalışma Kümesi Algoritması

- Kesin tanım (son Δ referans) her erişimde kaydırılan bir yazmaç ister: çok pahalı. Yaklaşım: **son τ sanal zaman** içinde kullanılan sayfalar.

son kullanım	R	
2204	0	
2204	0	
2204	0	
1213	0	yaş 991 > τ → çıkar
2204	0	
2204	0	
2204	0	
1620	0	yaş 584 > τ

2204 şimdiki sanal zaman

tüm sayfaları tara:

R = 1 → zaman ← şimdi, R ← 0

R = 0, yaş > τ → çıkar

R = 0, yaş ≤ τ → en eskiyi hatırla

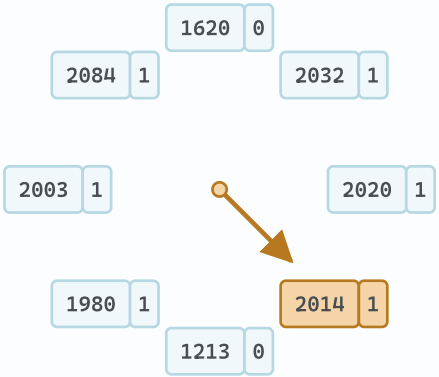
hiçbiri yoksa: en eski (tercihen temiz) sayfa

Bulunan ilk uygun sayfa (1213) çıkarılır; tarama diğer girdileri güncelleyerek sürer. Hepsi kümedeyse en eski R = 0 sayfa, o da yoksa rastgele (tercihen temiz) bir sayfa seçilir.

Tanenbaum MOS Şekil 3-20 (yeniden çizim). Dezavantaj: her hatada *tüm* tabloyu taramak.

WSClock

- Çalışma kümesi fikri + saatin **dairesel listesi** (Carr ve Hennessy, 1981): tüm tabloyu taramak gerekmez.
- Kirli eski sayfa hemen atılmaz: **yazması çizelgelenir**, el temiz bir sayfa arar.
- El tam tur atarsa: en az bir yazma çizelgelendiyse, ilk temizlenen sayfaya kadar döner.
- Hiç yazma yoksa tüm sayfalar kümededir: herhangi bir temiz sayfa; o da yoksa elin altındaki sayfa yazılıp atılır.



el bir sayfaya gelince:

R = 1 → R ← 0, zaman ← şimdi, ilerle

R = 0, yaş > τ, temiz → çıkar

R = 0, yaş > τ, kirli → yazmayı çizelgele, ilerle

R = 0, yaş ≤ τ → ilerle

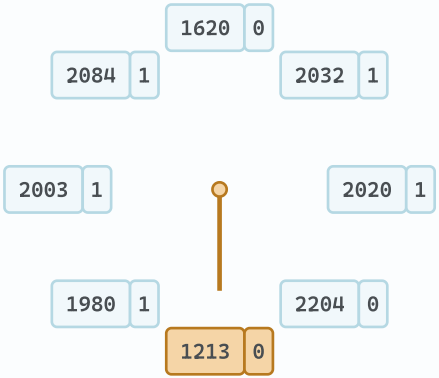
τ = 500, şimdi = 2204

Sayfa hatası: el 2014 zamanlı sayfada; R = 1 → bu tıkta kullanılmış, iyi bir kurban değil.

Tanenbaum MOS Şekil 3-21 (yeniden çizim). Basit, verimli ve pratikte yaygın bir tasarım.

WSClock

- Çalışma kümesi fikri + saatin **dairesel listesi** (Carr ve Hennessy, 1981): tüm tabloyu taramak gerekmez.
- Kirli eski sayfa hemen atılmaz: **yazması çizelgelenir**, el temiz bir sayfa arar.
- El tam tur atarsa: en az bir yazma çizelgelendiyse, ilk temizlenen sayfaya kadar döner.
- Hiç yazma yoksa tüm sayfalar kümededir: herhangi bir temiz sayfa; o da yoksa elin altındaki sayfa yazılıp atılır.



el bir sayfaya gelince:

R = 1 → R ← 0, zaman ← şimdi, ilerle

R = 0, yaş > τ, temiz → çıkar

R = 0, yaş > τ, kirli → yazmayı çizelgele, ilerle

R = 0, yaş ≤ τ → ilerle

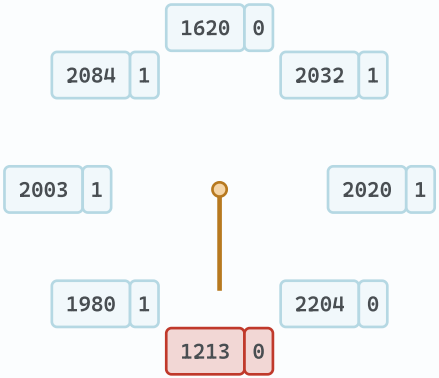
τ = 500, şimdi = 2204

R ← 0, zamanı 2204 yapılır, el ilerler. Sıradaki sayfa: zaman 1213, R = 0.

Tanenbaum MOS Şekil 3-21 (yeniden çizim). Basit, verimli ve pratikte yaygın bir tasarım.

WSClock

- Çalışma kümesi fikri + saatin **dairesel listesi** (Carr ve Hennessy, 1981): tüm tabloyu taramak gerekmez.
- Kirli eski sayfa hemen atılmaz: **yazması çizelgelenir**, el temiz bir sayfa arar.
- El tam tur atarsa: en az bir yazma çizelgelendiyse, ilk temizlenen sayfaya kadar döner.
- Hiç yazma yoksa tüm sayfalar kümededir: herhangi bir temiz sayfa; o da yoksa elin altındaki sayfa yazılıp atılır.



el bir sayfaya gelince:

R = 1 → R ← 0, zaman ← şimdi, ilerle

R = 0, yaş > τ, temiz → çıkar

R = 0, yaş > τ, kirli → yazmayı çizelgele, ilerle

R = 0, yaş ≤ τ → ilerle

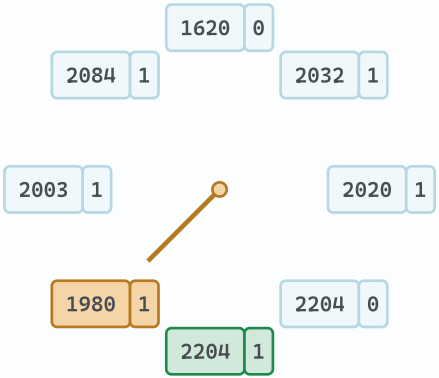
τ = 500, şimdi = 2204

Yaş = 2204 − 1213 = 991 > τ: çalışma kümesinde değil. Temizse **kurban**; kirli olsaydı yazması çizelgelenir ve el ilerlerdi.

Tanenbaum MOS Şekil 3-21 (yeniden çizim). Basit, verimli ve pratikte yaygın bir tasarım.

WSClock

- Çalışma kümesi fikri + saatin **dairesel listesi** (Carr ve Hennessy, 1981): tüm tabloyu taramak gerekmez.
- Kirli eski sayfa hemen atılmaz: **yazması çizelgelenir**, el temiz bir sayfa arar.
- El tam tur atarsa: en az bir yazma çizelgelendiyse, ilk temizlenen sayfaya kadar döner.
- Hiç yazma yoksa tüm sayfalar kümededir: herhangi bir temiz sayfa; o da yoksa elin altındaki sayfa yazılıp atılır.



el bir sayfaya gelince:

R = 1 → R ← 0, zaman ← şimdi, ilerle

R = 0, yaş > τ, temiz → çıkar

R = 0, yaş > τ, kirli → yazmayı çizelgele, ilerle

R = 0, yaş ≤ τ → ilerle

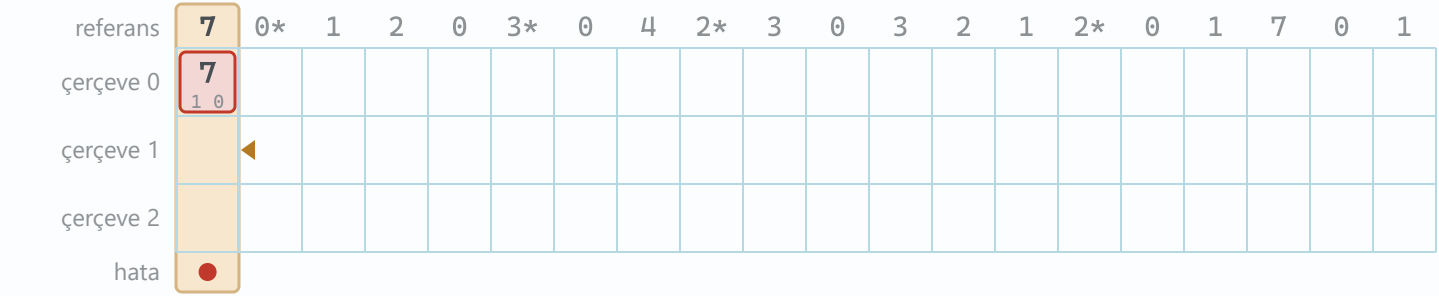
τ = 500, şimdi = 2204

Yeni sayfa bu çerçeveye yüklenir: zaman 2204, R = 1. El bir sonraki çerçeveye geçer.

Tanenbaum MOS Şekil 3-21 (yeniden çizim). Basit, verimli ve pratikte yaygın bir tasarım.

WSClock: Ortak Dizide

Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.

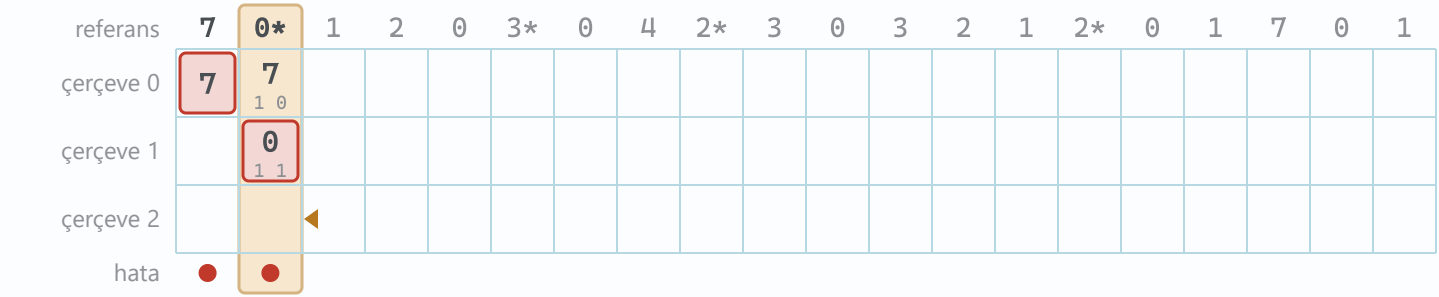


son kullanım (sanal zaman t = 1): 7: 1

1. referans 7: hata; boş çerçeve 0'a yüklenir. Hata: 1

WSClock: Ortak Dizide

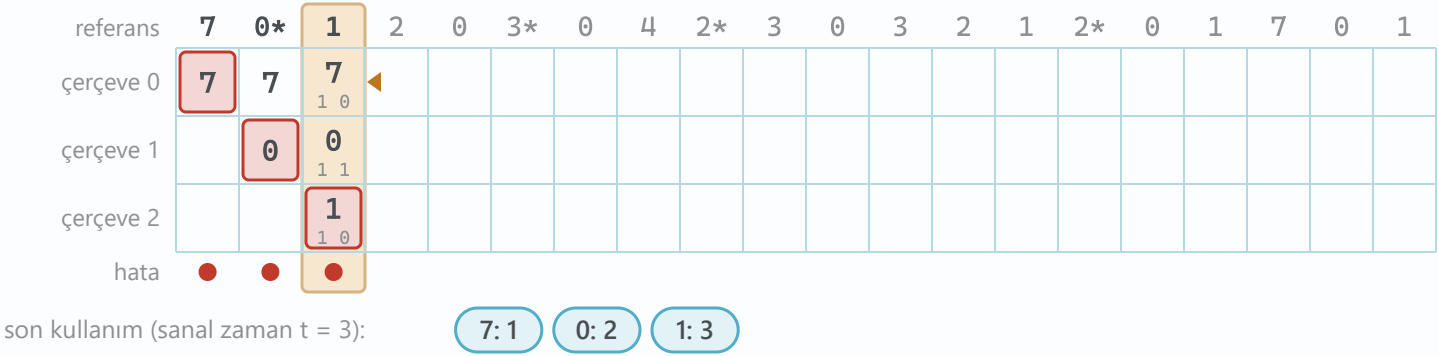
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



2. referans 0*: hata; boş çerçeve 1'e yüklenir. Hata: 2

WSClock: Ortak Dizide

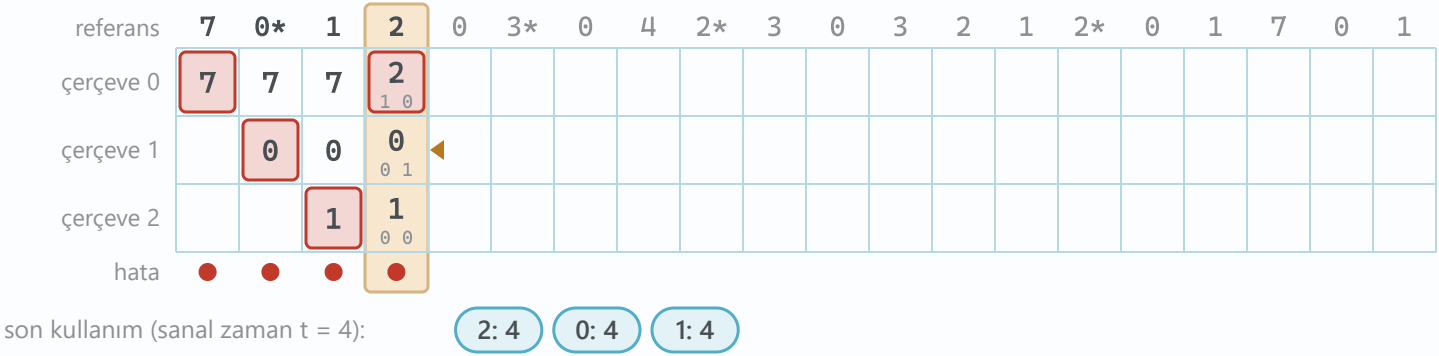
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



3. referans 1: hata; boş çerçeve 2'ye yüklenir. Hata: 3

WSClock: Ortak Dizide

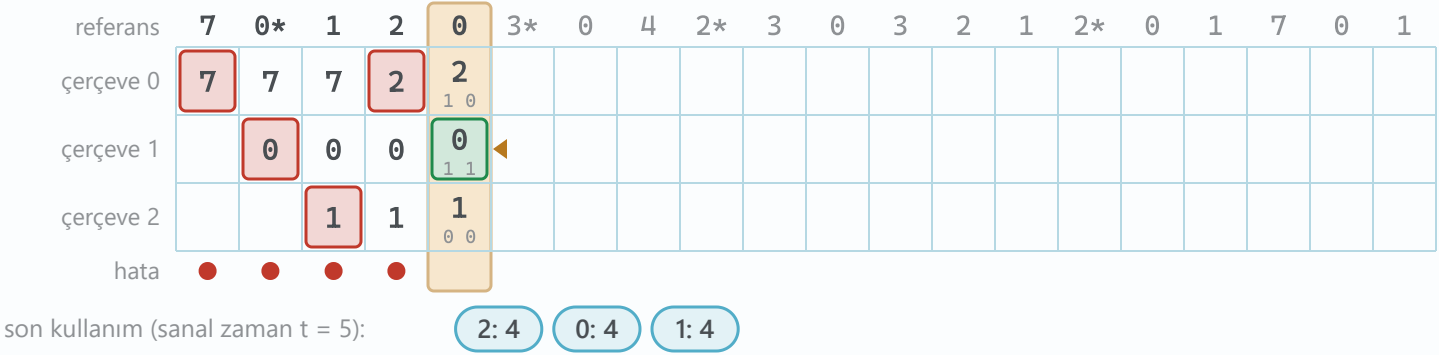
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



4. referans 2: hata; el taraması: 7: R=1 → R=0, zaman=4; 0: R=1 → R=0, zaman=4; 1: R=1 → R=0, zaman=4; tur bitti, yazma yok: ilk temiz sayfa 7 → kurban. Hata: 4

WSClock: Ortak Dizide

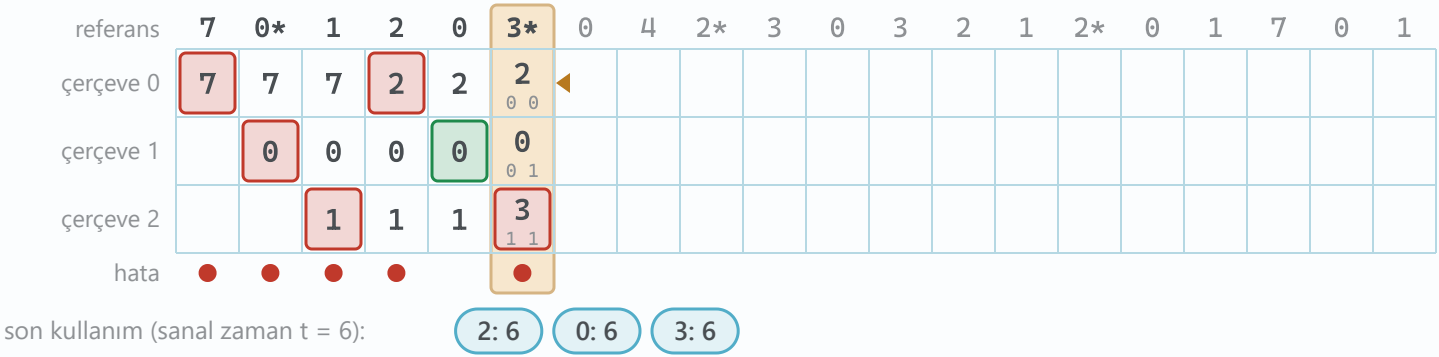
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



5. referans 0: isabet; R ← 1.

WSClock: Ortak Dizide

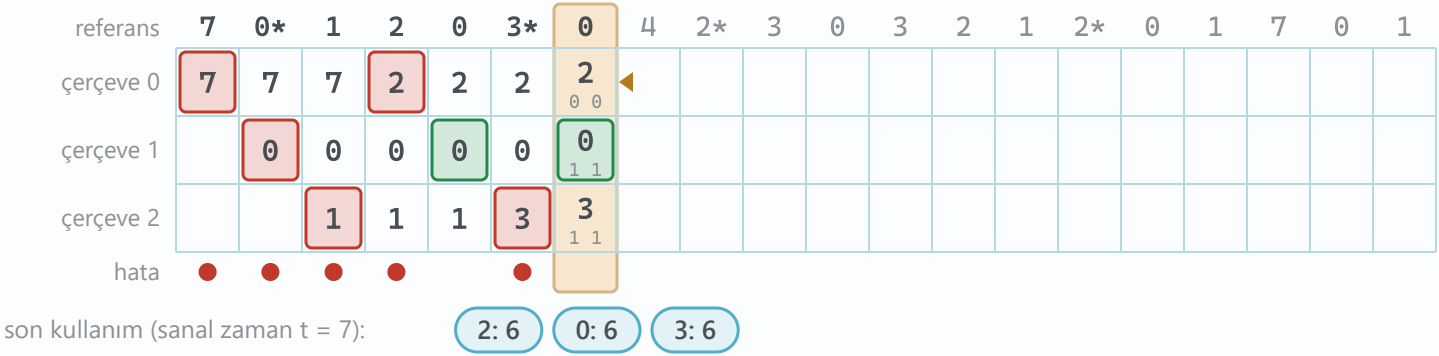
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



6. referans 3*: hata; el taraması: 0: R=1 → R=0, zaman=6; 1: R=0, yaş 2 ≤ τ → çalışma kümesinde; 2: R=1 → R=0, zaman=6; tur bitti, yazma yok: ilk temiz sayfa 1 → kurban. Hata: 5

WSClock: Ortak Dizide

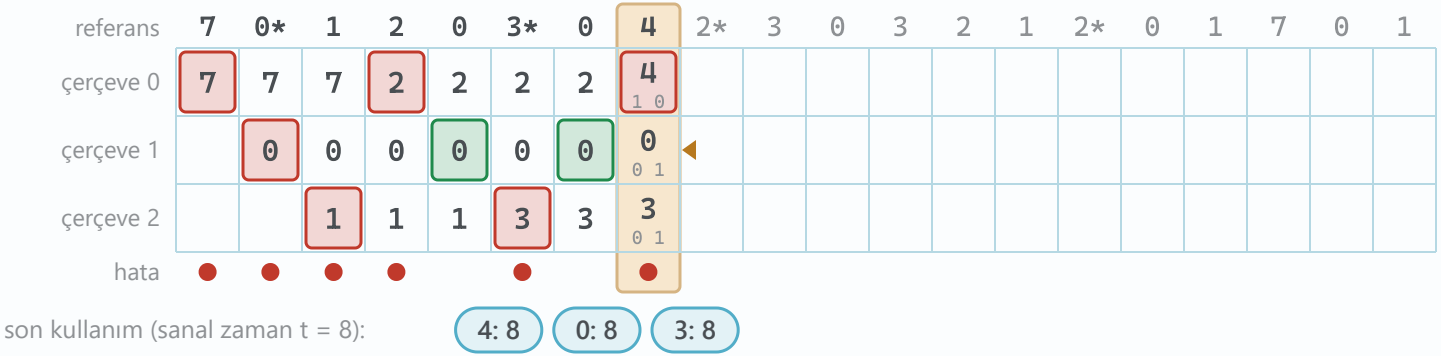
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



7. referans 0: isabet; R ← 1.

WSClock: Ortak Dizide

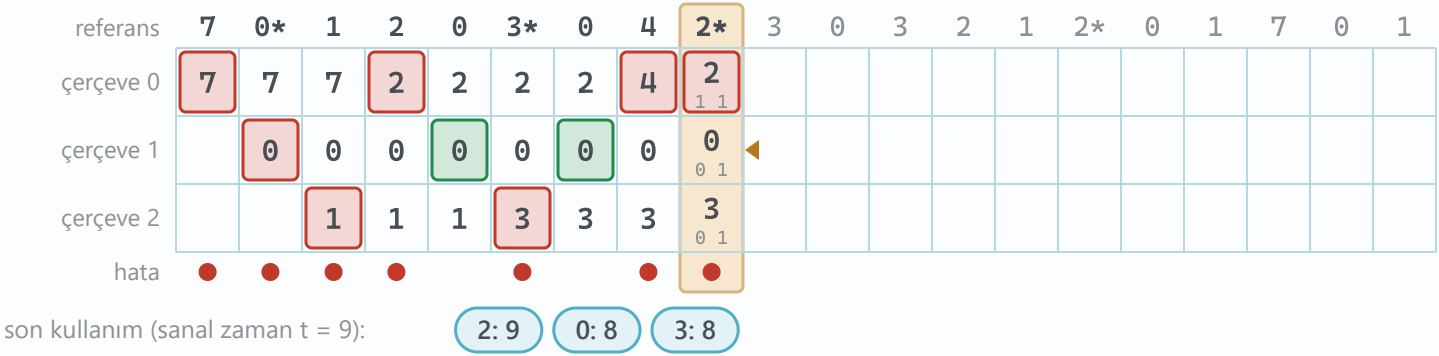
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



8. referans 4: hata; el taraması: 2: R=0, yaş 2 ≤ τ → çalışma kümesinde; 0: R=1 → R=0, zaman=8; 3: R=1 → R=0, zaman=8; tur bitti, yazma yok: ilk temiz sayfa 2 → kurban. Hata: 6

WSClock: Ortak Dizide

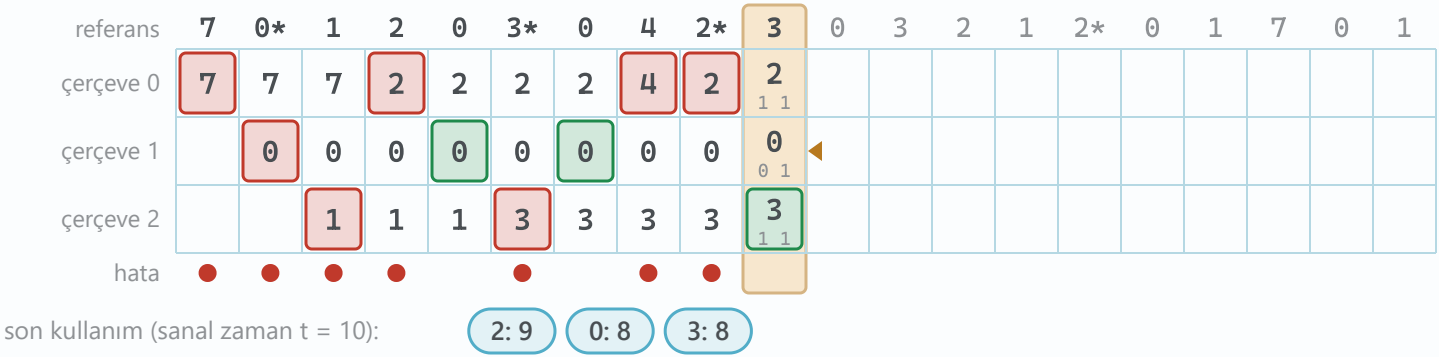
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



9. referans 2*: hata; el taraması: 0: R=0, yaş 1 ≤ τ → çalışma kümesinde; 3: R=0, yaş 1 ≤ τ → çalışma kümesinde; 4: R=1 → R=0, zaman=9; tur bitti, yazma yok: ilk temiz sayfa 4 → kurban. Hata: 7

WSClock: Ortak Dizide

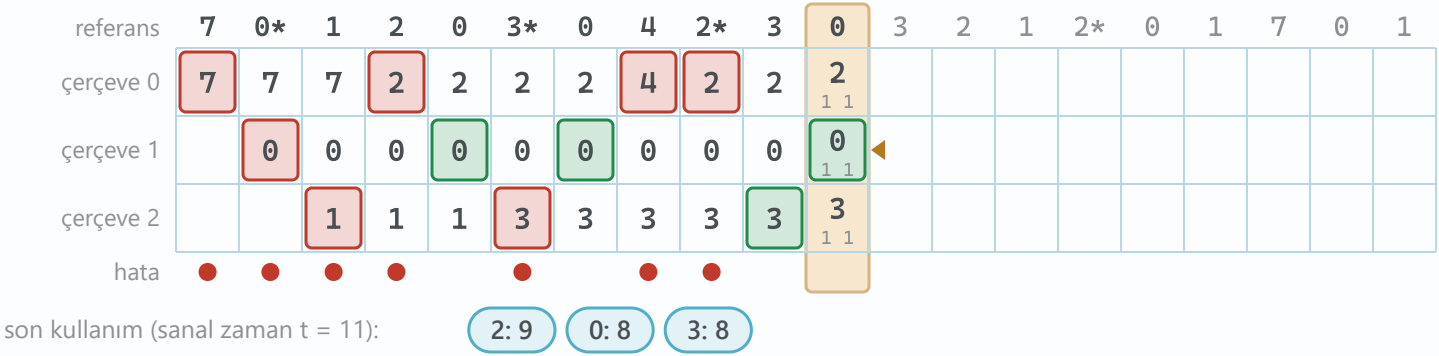
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



10. referans 3: isabet; R ← 1.

WSClock: Ortak Dizide

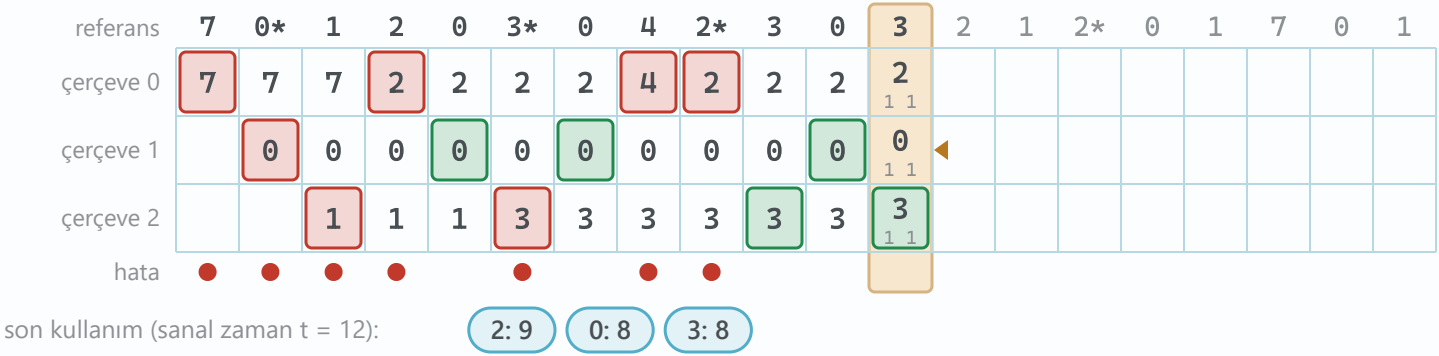
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



11. referans 0: isabet; R ← 1.

WSClock: Ortak Dizide

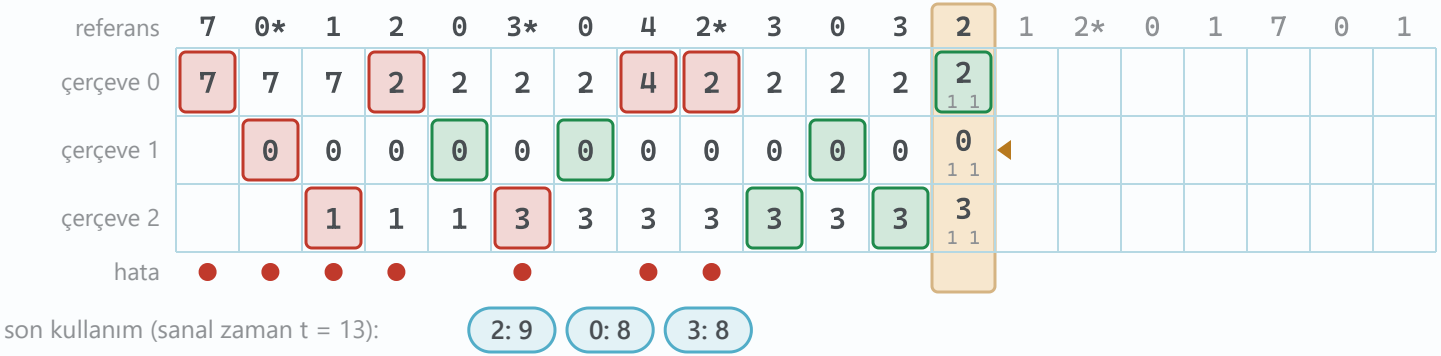
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



12. referans 3: isabet; R ← 1.

WSClock: Ortak Dizide

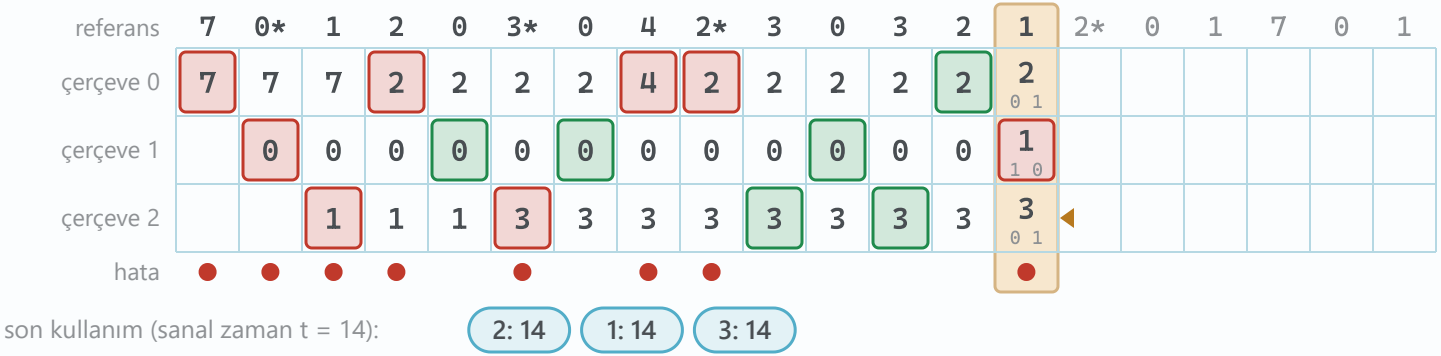
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



13. referans 2: isabet; R ← 1.

WSClock: Ortak Dizide

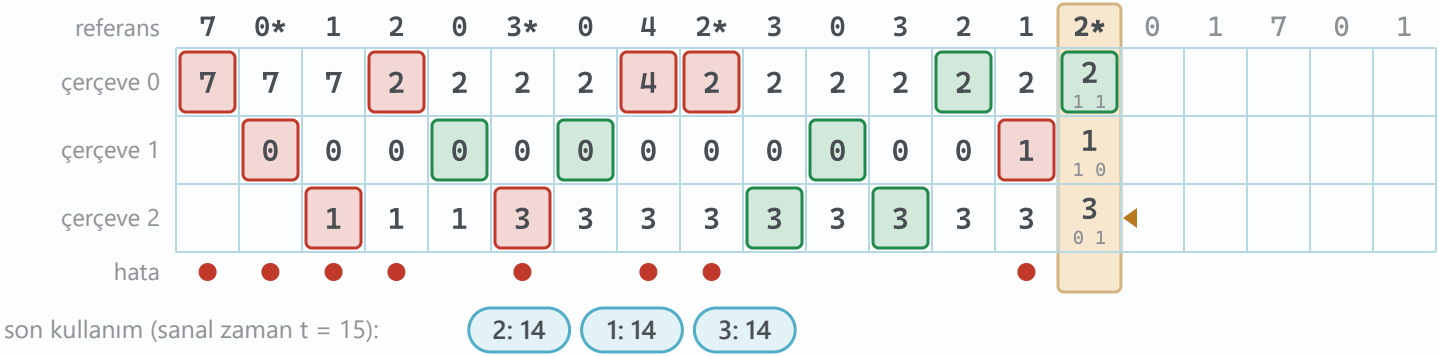
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



14. referans 1: hata; el taraması: 0: R=1 → R=0, zaman=14; 3: R=1 → R=0, zaman=14; 2: R=1 → R=0, zaman=14; tur bitti, temiz yok: 0 yazılıp atılır. Hata: 8

WSClock: Ortak Dizide

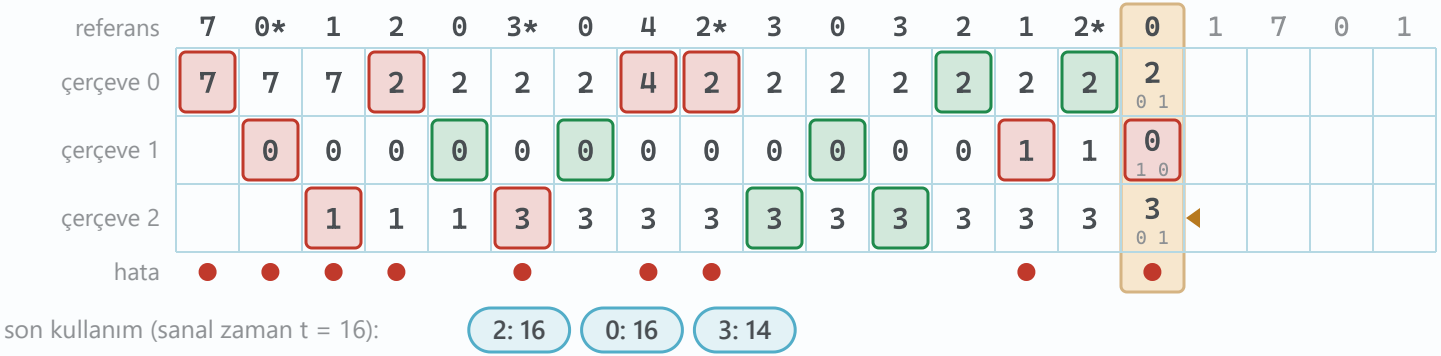
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



15. referans 2*: isabet; R ← 1, yazma → M ← 1.

WSClock: Ortak Dizide

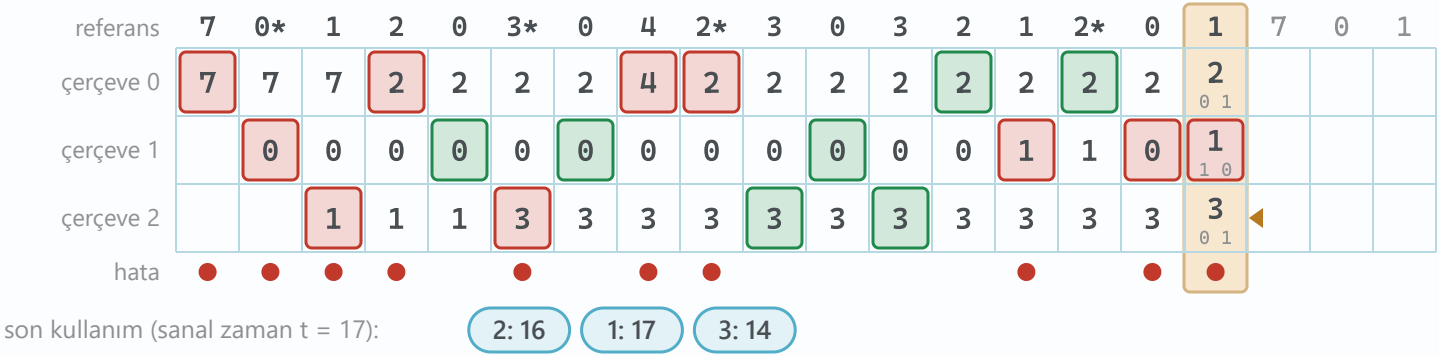
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



16. referans 0: hata; el taraması: 3: R=0, yaş 2 ≤ τ → çalışma kümesinde; 2: R=1 → R=0, zaman=16; 1: R=1 → R=0, zaman=16; tur bitti, yazma yok: ilk temiz sayfa 1 → kurban. Hata: 9

WSClock: Ortak Dizide

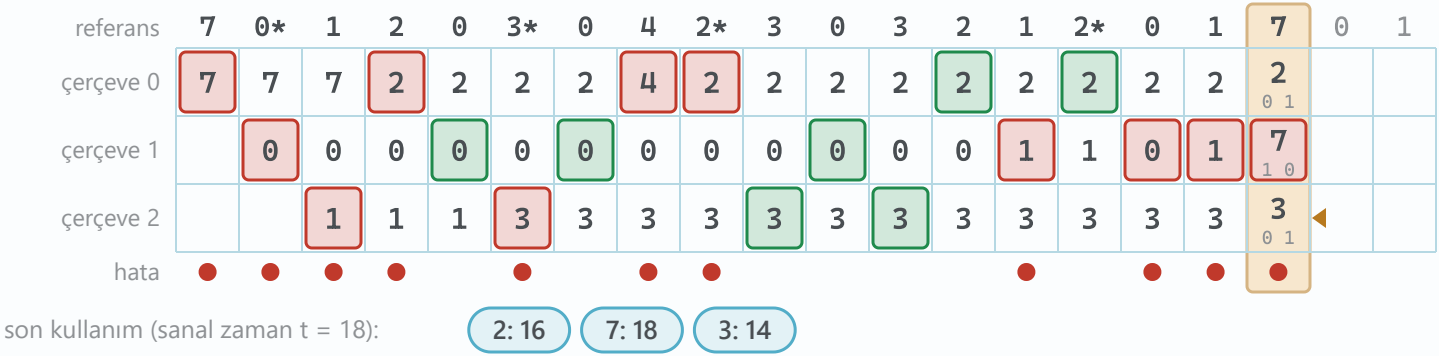
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



17. referans 1: hata; el taraması: 3: R=0, yaş 3 ≤ τ → çalışma kümesinde; 2: R=0, yaş 1 ≤ τ → çalışma kümesinde; 0: R=1 → R=0, zaman=17; tur bitti, yazma yok: ilk temiz sayfa 0 → kurban. Hata: 10

WSClock: Ortak Dizide

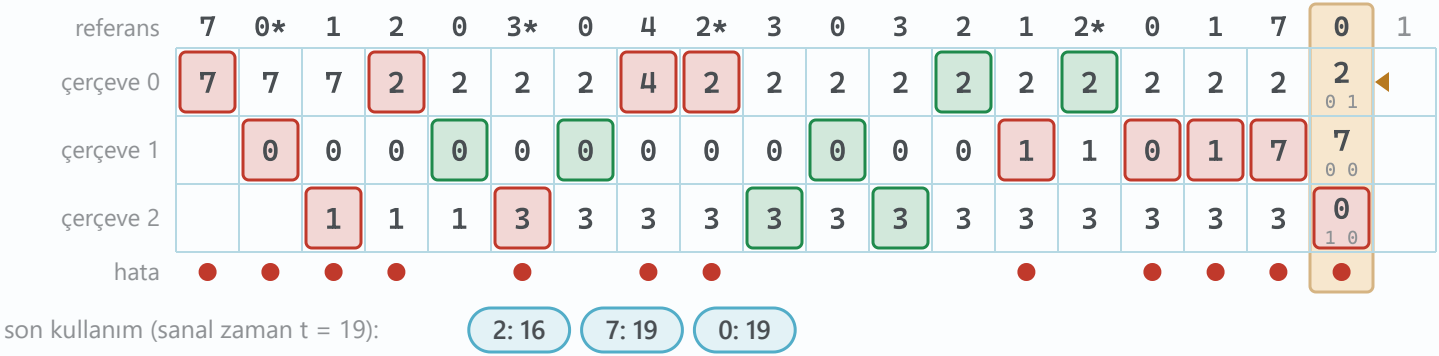
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



18. referans 7: hata; el taraması: 3: R=0, yaş 4 ≤ τ → çalışma kümesinde; 2: R=0, yaş 2 ≤ τ → çalışma kümesinde; 1: R=1 → R=0, zaman=18; tur bitti, yazma yok: ilk temiz sayfa 1 → kurban. Hata: 11

WSClock: Ortak Dizide

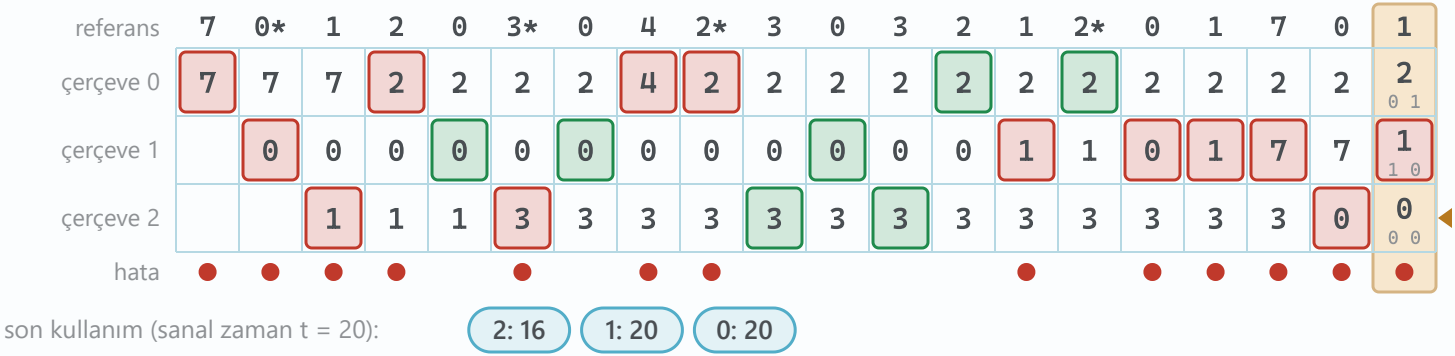
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



19. referans 0: hata; el taraması: 3: R=0, yaş 5 > τ, kirli → yazma çizelgelendi; 2: R=0, yaş 3 ≤ τ → çalışma kümesinde; 7: R=1 → R=0, zaman=19; tur bitti: yazılan 3 artık temiz → kurban. Hata: 12

WSClock: Ortak Dizide

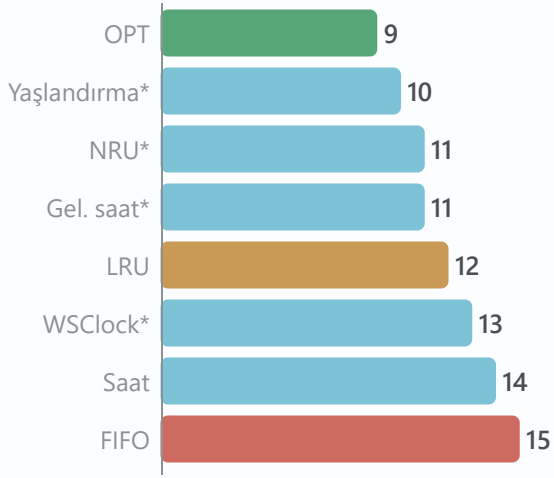
Sanal zaman = referans numarası, τ = 4. Dizi ve yazmalar (*) NRU ile aynı. Yazma çizelgelenen sayfa, tur sonunda temizlenmiş kabul edilir.



20. referans 1: hata; el taraması: 2: R=0, yaş 4 ≤ τ → çalışma kümesinde; 7: R=0, yaş 1 ≤ τ → çalışma kümesinde; 0: R=1 → R=0, zaman=20; tur bitti, yazma yok: ilk temiz sayfa 7 → kurban. Hata: 13 **Toplam 13 hata, 2 kirli yazım.** Yalnızca 3 çerçeve ve kısa bir dizide çalışma kümesi fikri avantaj sağlamaz.

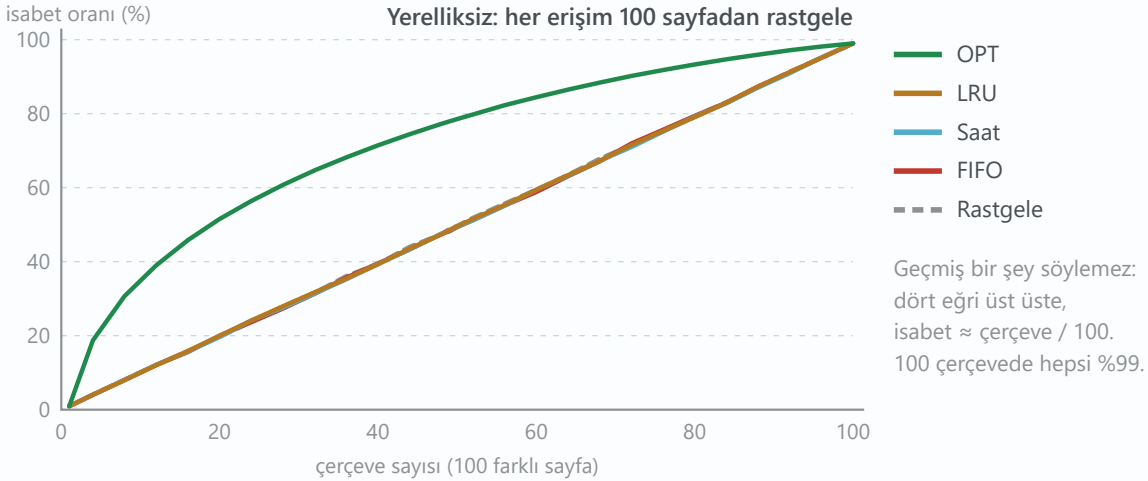
Algoritmaların Karşılaştırması

Algoritma	Yorum
OPT	Uygulanamaz; kıyaslama için alt sınır.
NRU	LRU'ya çok kaba yaklaşım; ucuz.
FIFO	Önemli sayfaları atabilir; Belady anomalisi.
İkinci şans	FIFO'ya göre büyük iyileşme.
Saat	İkinci şansın gerçekçi gerçeklemesi.
LRU	Çok iyi ama tam hâli donanım ister.
NFU	LRU'ya oldukça kaba yaklaşım (unutmaz).
Yaşlandırma	LRU'ya iyi yaklaşan verimli algoritma.
Çalışma kümesi	İyi ama uygulaması pahalı.
WSClock	İyi ve verimli.



Farklı referans dizileri: Sayfa Yer Değiştirme Simülatörü (FIFO, OPT, LRU, LFU, ikinci şans). * R/M bitli algoritmalar yazma ve tık bilgisi de kullanır; kısa tek bir dizi genel sıralama vermez. NFU hiç unutmadığı için LRU'ya iyi yaklaşmaz; iyi yaklaşım yaşlandırmadır (Tanenbaum).

İş Yükü Politikayı Belirler

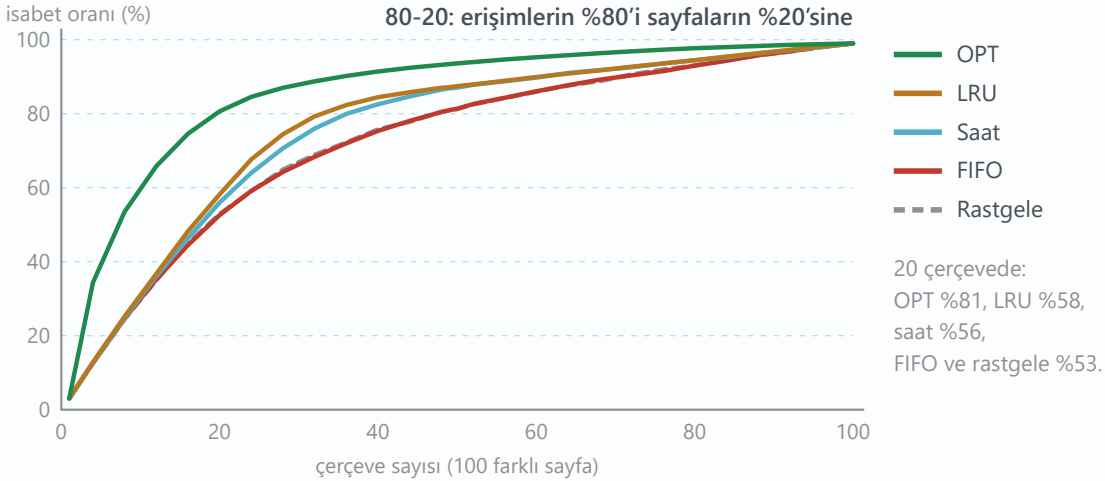


Yerellik yoksa hangi sayfanın tutulduğu fark etmez: LRU, FIFO, saat ve rastgele aynı doğruyu izler. Yalnız geleceği bilen OPT öndedir; bellek tüm sayfaları alınca politika önemsizleşir.

Adım 1 / 3

10.000 erişimlik benzetim (rastgele için 5 denemenin ortalaması). Ardışık tarama LRU'yu gerçek sistemlerde de vurur: büyük bir dosyayı bir kez okumak sıcak sayfaları kovabilir. Linux bu yüzden yeni sayfayı önce etkisiz listeye koyar, ancak ikinci erişimde etkin listeye taşır.

İş Yükü Politikayı Belirler

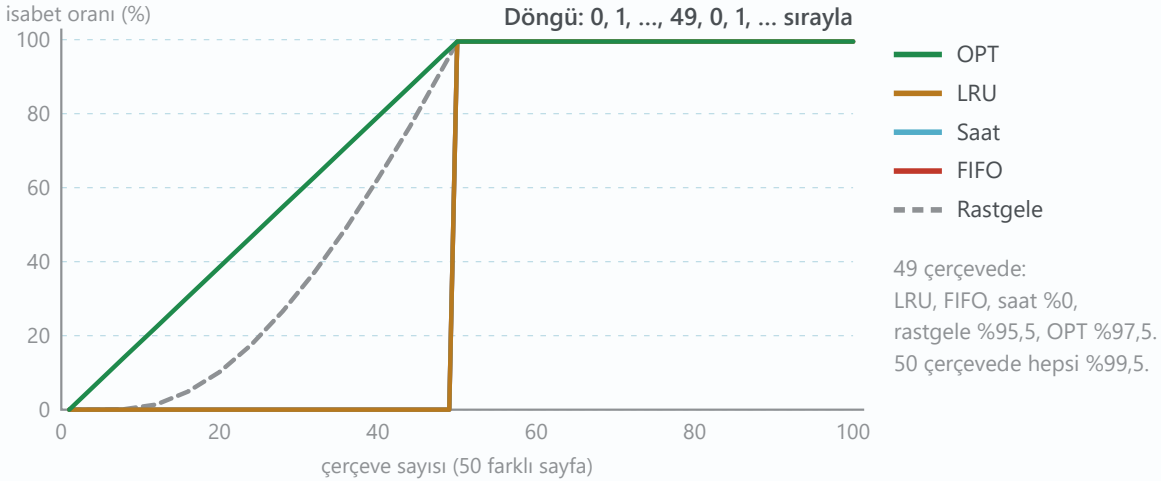


Sıcak sayfalar varsa geçmişi kullanan politikalar kazanır: LRU sıcak 20 sayfayı bellekte tutar. Saat, LRU'ya çok yakın sonuç verir; FIFO rastgeleden iyi değildir.

Adım 2 / 3

10.000 erişimlik benzetim (rastgele için 5 denemenin ortalaması). Ardışık tarama LRU'yu gerçek sistemlerde de vurur: büyük bir dosyayı bir kez okumak sıcak sayfaları kovabilir. Linux bu yüzden yeni sayfayı önce etkisiz listeye koyar, ancak ikinci erişimde etkin listeye taşır.

İş Yükü Politikayı Belirler



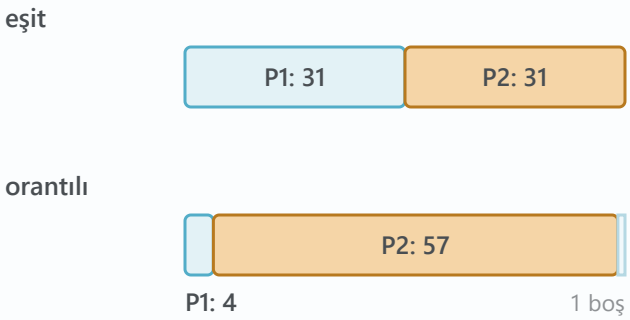
Bellekten biraz büyük döngü LRU'nun en kötü durumudur: atılan sayfa hep en yakında gerekecek olandır, her erişim ıskalar. Kör rastgele seçim burada LRU'yu ezer.

Adım 3 / 3

10.000 erişimlik benzetim (rastgele için 5 denemenin ortalaması). Ardışık tarama LRU'yu gerçek sistemlerde de vurur: büyük bir dosyayı bir kez okumak sıcak sayfaları kovabilir. Linux bu yüzden yeni sayfayı önce etkisiz listeye koyar, ancak ikinci erişimde etkin listeye taşır.

Çerçeve Tahsisi: Eşit ve Orantılı

- **Alt sınır:** bir komutun dokunabileceği tüm sayfalar aynı anda bellekte olmalıdır (komut + işlenenler + dolaylı adresler sayfa sınırını aşabilir). Bunu *mimari* belirler.
- **Üst sınır:** boş fiziksel bellek.
- **Eşit:** m çerçeve, n süreç → her birine ⌊m / n⌋.
- **Orantılı:** $a_i = s_i / S \times m$. Örnek: 10/137 × 62 ≈ **4**, 127/137 × 62 ≈ **57**.
- **Öncelikli:** oran boyut yerine (ya da yanında) önceliğe göre kurulur; yüksek öncelikli süreç düşük öncelikliden çerçeve alabilir.
- NUMA'da bir çerçevenin *nerede* olduğu da önemlidir: süreç, çalıştığı düğüme yakın bellekten beslenmelidir.

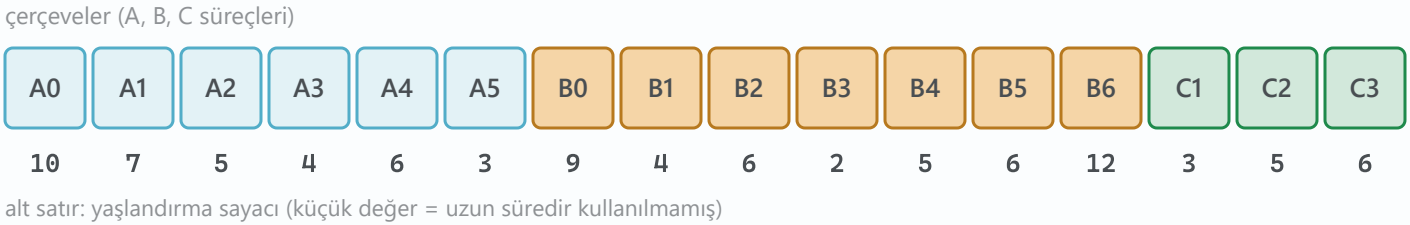


s_1 = 10 sayfa, s_2 = 127 sayfa, m = 62

$a_i = s_i / S \times m$ (aşağı yuvarlanır)

Yerel ve Genel Yer Değiştirme

- **Genel:** kurban tüm çerçevelerden seçilir; süreç başkasından çerçeve alabilir. Daha yüksek verim, ama bir sürecin süresi diğerlerinin davranışına göre çok değişebilir.
- Linux ve Windows genel geri kazanım kullanır (cgroup/iş nesnesi sınırlarıyla).
- **Yerel:** süreç yalnızca kendi kümesinden seçer; daha tutarlı performans, ama boşta duran çerçeveler başkasına geçmez.
- Yerel politika çalışma kümesi büyüyüp küçülürken çerçeve sayısını ayarlayan bir mekanizma ister (PFF).

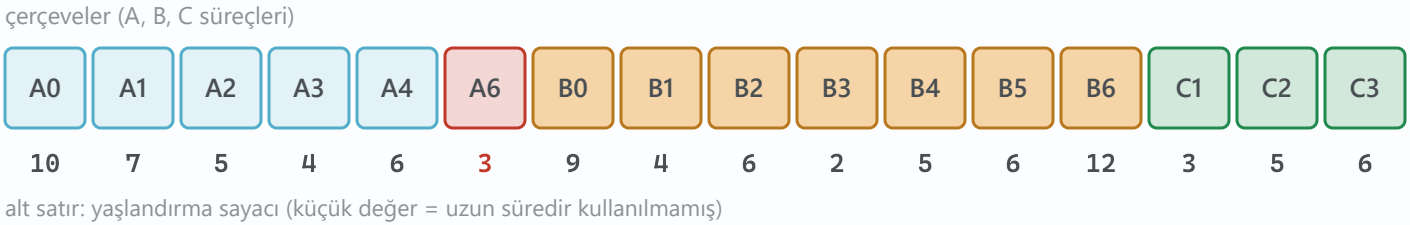


Üç süreç 16 çerçeveyi paylaşıyor; sayılar yaşlandırma sayacı değeridir (küçük = uzun süredir kullanılmamış). A, A6 sayfasında hata yapar.

Tanenbaum MOS Şekil 3-22 (yeniden çizim).

Yerel ve Genel Yer Değiştirme

- **Genel:** kurban tüm çerçevelerden seçilir; süreç başkasından çerçeve alabilir. Daha yüksek verim, ama bir sürecin süresi diğerlerinin davranışına göre çok değişebilir.
- Linux ve Windows genel geri kazanım kullanır (cgroup/iş nesnesi sınırlarıyla).
- **Yerel:** süreç yalnızca kendi kümesinden seçer; daha tutarlı performans, ama boşta duran çerçeveler başkasına geçmez.
- Yerel politika çalışma kümesi büyüyüp küçülürken çerçeve sayısını ayarlayan bir mekanizma ister (PFF).

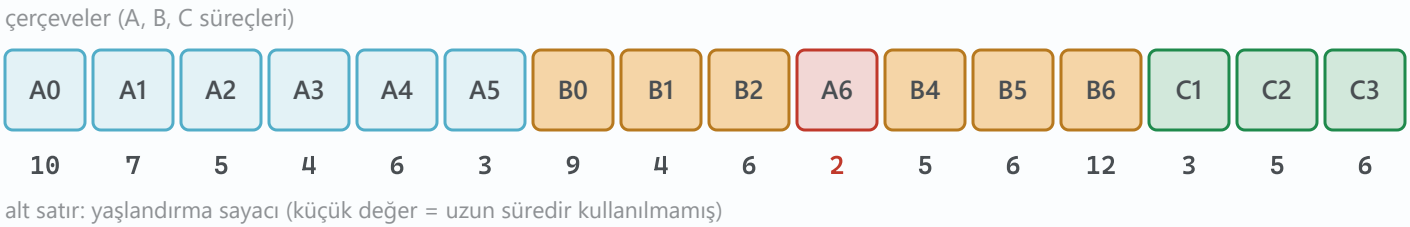


Yerel: kurban yalnızca A'nın çerçevelerinden seçilir → en küçük değerli **A5**. A'nın çerçeve sayısı sabit kalır (6).

Tanenbaum MOS Şekil 3-22 (yeniden çizim).

Yerel ve Genel Yer Değiştirme

- **Genel:** kurban tüm çerçevelerden seçilir; süreç başkasından çerçeve alabilir. Daha yüksek verim, ama bir sürecin süresi diğerlerinin davranışına göre çok değişebilir.
- Linux ve Windows genel geri kazanım kullanır (cgroup/iş nesnesi sınırlarıyla).
- **Yerel:** süreç yalnızca kendi kümesinden seçer; daha tutarlı performans, ama boşta duran çerçeveler başkasına geçmez.
- Yerel politika çalışma kümesi büyüyüp küçülürken çerçeve sayısını ayarlayan bir mekanizma ister (PFF).

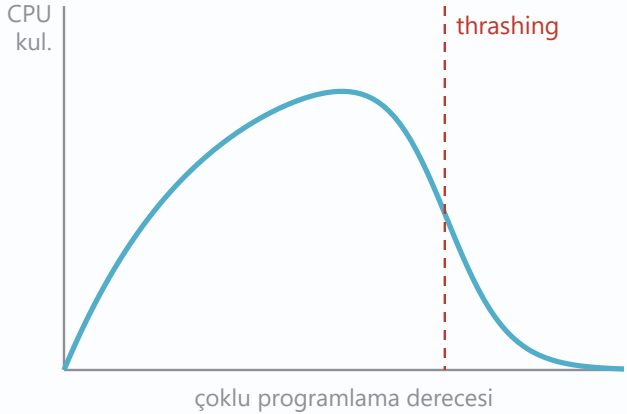


Genel: tüm çerçeveler aday → en küçük değerli **B3** çıkar. A büyür, B küçülür: tahsis dinamik olarak değişir.

Tanenbaum MOS Şekil 3-22 (yeniden çizim).

Thrashing (Boşa Çabalama)

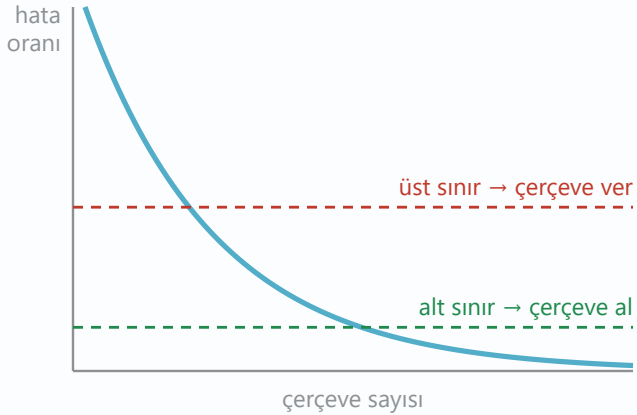
- Süreçlerin etkin sayfaları (çalışma kümeleri) belleğe sığmazsa sayfa hatası oranı tırmanır: atılan sayfa **hemen yeniden gerekir**.
- Süreçler zamanlarının çoğunu sayfa beklemekle geçirir; disk kuyruğu dolar, **CPU kullanımı düşer**.
- Eski uzun vadeli çizelgeleyiciler bunu "CPU boşta" diye yorumlayıp **yeni süreç** ekler → durum daha da kötüleşir (geri besleme döngüsü).
- Neden: $\Sigma |WS_i| >$ bellek. Yerel yer değiştirme thrashing'i *bir süreçle sınırlar* ama onu çözmez.
- Çare: çoklu programlama derecesini düşürmek (süreç askıya almak/takaslamak), daha çok RAM, ya da sistemin OOM öldürücüsü.



Linux'ta belirti: `vmstat 1` çıktısında yüksek `si/so` ve `wa`; `/proc/pressure/memory` (PSI) bellek baskısını yüzde olarak verir.

Thrashing'i Önlemek: Çalışma Kümesi ve PFF

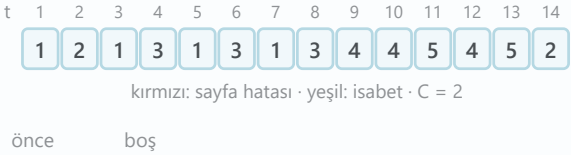
- **Çalışma kümesi stratejisi:** $D = \Sigma |WS_i|$ toplam talebi verir. $D > m$ ise bir süreç **askıya alınır**; D yeterince küçükse yeni süreç başlatılır.
- Askıdan dönen süreç için önce çalışma kümesini yüklemek (ön sayfalama) hata fırtınasını önler.
- **PFF** (sayfa hatası sıklığı): daha doğrudan bir geri besleme. Hata oranı üst sınırı aşarsa sürece çerçeve verilir, alt sınırın altına inerse çerçeve geri alınır.
- Boş çerçeve yokken oran yüksekse süreç askıya alınır ve çerçeveleri dağıtılır.



Sayfa hataları ile yerellikler arasında bağ vardır: yeni bir yerelliğe geçişte hata oranında **tepe** oluşur, çalışma kümesi yüklenince oran düşer.

PFF: Kod İzleme

```
▶ void pff(Surec *s, int p, int t) {
▶   if (t - s->son > C) {   /* seyrek hata */
3     for (int i = s->n - 1; i >= 0; i--)
4       if (s->kume[i].R == 0)
5         cikar(s, i);    /* kullanılmadı */
6   }
7   for (int i = 0; i < s->n; i++)
8     s->kume[i].R = 0;     /* sıfırla */
9   ekle(s, p);             /* R = 1 ile */
10  s->son = t;
11 }
```



Süreç belleksiz başlıyor; eşik C = 2 referans. Her sayfa hatasında `pff` çağrılır ve son hatadan bu yana geçen süreye bakar.

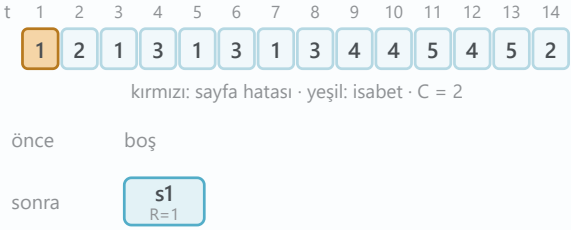
Adım 1 / 8

DEĞİŞKENLER

`t = 1` `p = 1` `son = 0` `t−son = −` `n = 0`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
2    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
5          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
8      s->kume[i].R = 0;     /* sıfırla */
9    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



aralık = 1 ≤ C

t = 1, ilk hata (s1): son = 0, aralık 1 ≤ C. Çıkarılacak sayfa yok; s1 R = 1 ile eklenir.

Adım 2 / 8

DEĞİŞKENLER

`t = 1` `p = 1` `son = 0` `t−son = 1` `n = 1`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
▶    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
5          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
▶      s->kume[i].R = 0;     /* sıfırla */
▶    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



t = 2, s2 hatası: aralık 1 ≤ C, hatalar **sık**. Hiçbir sayfa atılmaz, R bitleri sıfırlanır ve küme 2 sayfaya büyür.

Adım 3 / 8

DEĞİŞKENLER

`t = 2` `p = 2` `son = 1` `t−son = 1` `n = 2`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
▶    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
5          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
▶      s->kume[i].R = 0;     /* sıfırla */
▶    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



aralık = 2 ≤ C

t = 4, s3 hatası: aralık 2 ≤ C, hatalar **sık**. Hiçbir sayfa atılmaz, R bitleri sıfırlanır ve küme 3 sayfaya büyür.

Adım 4 / 8

DEĞİŞKENLER

`t = 4` `p = 3` `son = 2` `t−son = 2` `n = 3`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
▶    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
▶          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
8      s->kume[i].R = 0;     /* sıfırla */
▶    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



aralık = 5 > C

t = 9, s4 hatası: aralık 5 > C, hatalar **seyrek**. Son hatadan beri kullanılmayan (R = 0) s2 çıkarılır; küme küçülür.

DEĞİŞKENLER

`t = 9` `p = 4` `son = 4` `t−son = 5` `n = 3`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
▶    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
5          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
▶      s->kume[i].R = 0;     /* sıfırla */
▶    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



t = 11, s5 hatası: aralık 2 ≤ C, hatalar **sık**. Hiçbir sayfa atılmaz, R bitleri sıfırlanır ve küme 4 sayfaya büyür.

Adım 6 / 8

DEĞİŞKENLER

`t = 11` `p = 5` `son = 9` `t−son = 2` `n = 4`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
▶    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
▶          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
8      s->kume[i].R = 0;     /* sıfırla */
▶    ekle(s, p);             /* R = 1 ile */
10   s->son = t;
11 }
```



t = 14, s2 hatası: aralık 3 > C, hatalar **seyrek**. Son hatadan beri kullanılmayan (R = 0) s3, s1 çıkarılır; küme küçülür.

Adım 7 / 8

DEĞİŞKENLER

`t = 14` `p = 2` `son = 11` `t−son = 3` `n = 3`

PFF: Kod İzleme

```
1  void pff(Surec *s, int p, int t) {
2    if (t - s->son > C) {   /* seyrek hata */
3      for (int i = s->n - 1; i >= 0; i--)
4        if (s->kume[i].R == 0)
5          cikar(s, i);    /* kullanılmadı */
6    }
7    for (int i = 0; i < s->n; i++)
8      s->kume[i].R = 0;     /* sıfırla */
9    ekle(s, p);             /* R = 1 ile */
▶    s->son = t;
11 }
```



14 referansta 6 hata. Yerellik değişince (1–3'ten 4–5'e) önce hatalar sıklaştı ve küme büyüdü, sonra eski sayfalar seyrek bir hatada bırakıldı.

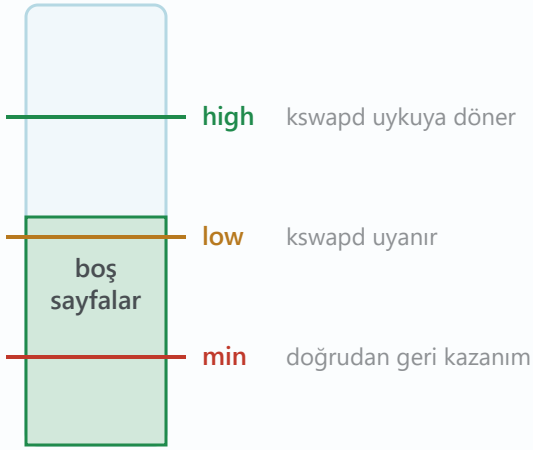
Adım 8 / 8

DEĞİŞKENLER

`t = 14` `p = 2` `son = 14` `t−son = 3` `n = 3`

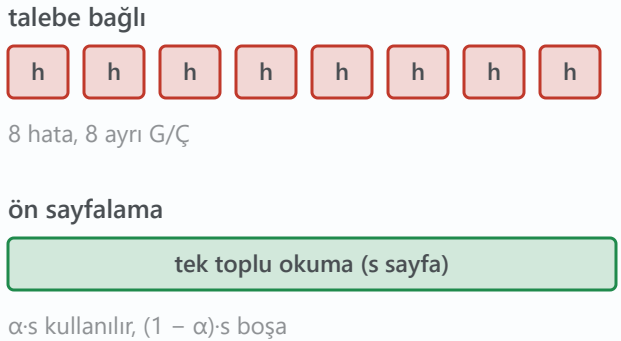
Temizleme Politikası ve Sayfalama Daemonu

- Kurbanı hata anında aramak yerine bir **sayfalama daemonu** periyodik uyanır; boş çerçeve azsa sayfaları önceden serbest bırakır.
- Kirli sayfaları **önceden diske yazar**: tahliye anında sayfa temizdir, hata yalnızca bir okuma bekler.
- Serbest bırakılan çerçeve hemen silinmez; yeniden gerekirse havuzdan **geri alınabilir** (küçük hata).
- **İki elli saat:** öndeki el kirli sayfaları yazdırır / R'yi temizler, arkadaki el uygun sayfaları serbest bırakır.
- Linux: `kswapd` bölge başına *min / low / high* su düzeyleriyle çalışır; kirli dosya sayfalarını `writeback` iş parçacıkları yazar.



Ön Sayfalama (Prepaging)

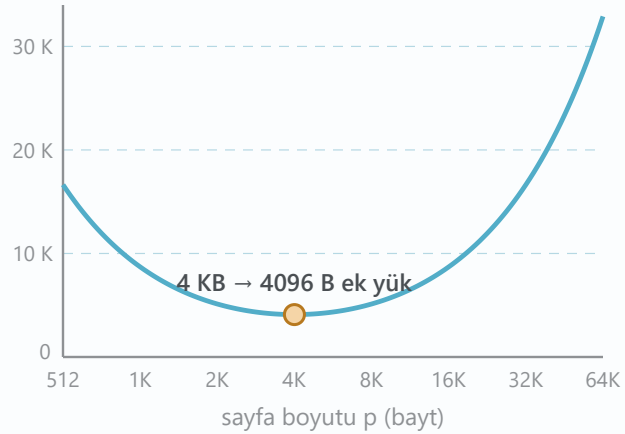
- Talebe bağlı sayfalama başlangıçta (ve askıdan dönüşte) **hata fırtınası** yaşar. Ön sayfalama, gerekecek sayfaları *önceden* toplu getirir.
- s sayfa önceden getirilir, α oranı kullanılır: kazanç **α·s hata**, bedel **(1 − α)·s gereksiz sayfa**. α 0'a yakınsa ön sayfalama kaybettirir.
- Askıya alınan sürecin **çalışma kümesini** hatırlayıp geri dönüşte yüklemek klasik örnektir.
- Linux: dosyalar için *readahead*, takas için `page-cluster` ; uygulama `madvise(MADV_WILLNEED)` ya da `mmap(MAP_POPULATE)` ile ipucu verir.
- Windows: önceden getirme (Prefetcher) ve SysMain (eski adı SuperFetch) sık kullanılan sayfaları boştayken yükler.



Sayfa Boyutu

- Ek maliyet = **se/p** (sayfa tablosu) + **p/2** (son sayfadaki iç parçalanma); s: süreç boyutu, e: girdi boyutu.
- Türev sıfır: $-se/p^2 + 1/2 = 0 \rightarrow$ **p = √(2se)**. s = 1 MB, e = 8 B → p = $\sqrt{(2^{24})}$ = **4 KB**.

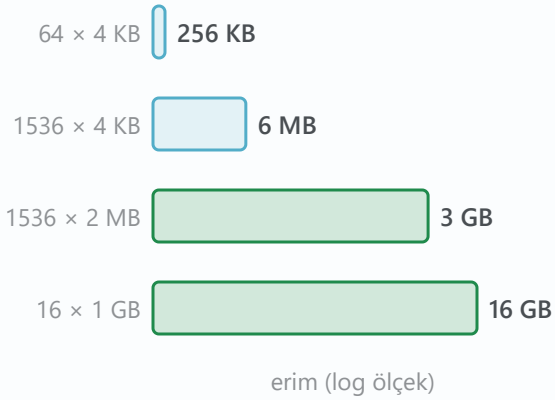
Küçük sayfa	Büyük sayfa
az iç parçalanma	küçük sayfa tablosu
yerelliği ince izler, az gereksiz veri	az hata, verimli G/Ç (arama/kurma maliyeti bir kez)
çok TLB ıskası	büyük TLB erimi



2026: x86-64 4 KB (+2 MB, 1 GB); Apple Silicon 16 KB; Android 15+ 16 KB çekirdekleri destekler; bazı ARM64 sunucular 64 KB. 1 KB sayfa yalnızca eski sistemlerde görülür.

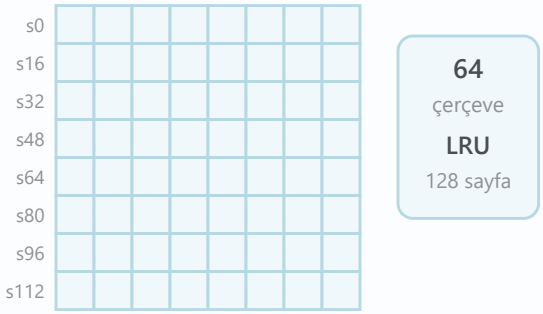
TLB Erimi ve Büyük Sayfalar

- **TLB erimi** (TLB reach) = TLB girdi sayısı × sayfa boyutu: TLB'den ıskasız erişilebilen bellek miktarı.
- Çalışma kümesi erime sığmıyorsa zaman **sayfa yürüyüşüne** gider. Girdi sayısını artırmak pahalıdır (ilişkisel bellek); sayfayı büyütmek daha etkilidir.
- **Büyük sayfalar:** x86-64'te 2 MB ve 1 GB. Linux: `hugetlbfs` ve *şeffaf büyük sayfalar* (THP, `khugepaged`); 6.x çekirdeklerinde çok boyutlu *folio*/mTHP desteği. Windows: büyük sayfa ayrıcalığı.
- Bedeli: iç parçalanma, büyük bitişik bölge bulma (sıkıştırma), sayfa başına daha uzun kopyalama/sıfırlama.
- Veritabanları ve sanal makineler büyük sayfadan en çok yararlanır.



Program Yapısı ve Yerellik

```
▶ int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
5     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

128 × 128 `int`, sayfa 128 sözcük: her **satır** ayrı bir sayfa. Sürece 64 çerçeve, hiçbiri dolu değil.

Adım 1 / 8

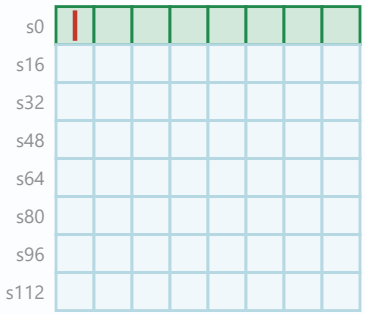
DEĞİŞKENLER

`hata = 0`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
▶     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

A, sütun 0'da: data[0][0] → sayfa 0'da hata.

Adım 2 / 8

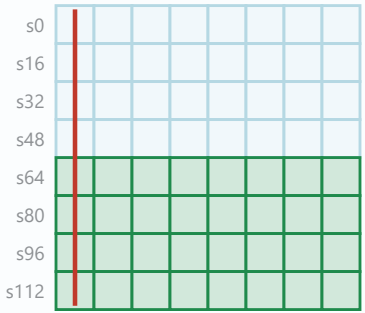
DEĞİŞKENLER

`j = 0` `i = 0` `sayfa = 0` `hata = 1`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
▶     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

Sütun 0 boyunca her `i` farklı bir sayfa: 128 hata; 64 çerçeveye sığmadığından sayfa 0–63 çoktan atıldı.

Adım 3 / 8

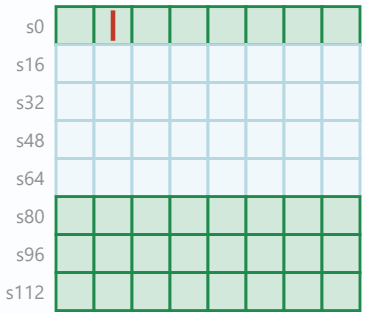
DEĞİŞKENLER

`j = 0` `i = 127` `sayfa = 127` `hata = 128`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
▶     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

Sütun 1: data[0][1] yine sayfa 0'da ama o **atıldı** → yeniden hata.

Adım 4 / 8

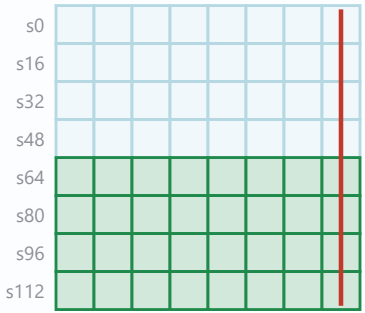
DEĞİŞKENLER

`j = 1` `i = 0` `sayfa = 0` `hata = 129`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
▶ for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
5     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64
çerçeve
LRU
128 sayfa

Her erişim hata: 128 × 128 = **16.384 hata**.

Adım 5 / 8

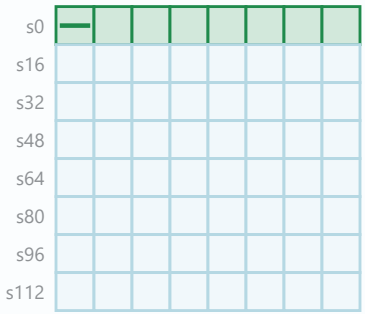
DEĞİŞKENLER

`j = 128` `hata = 16384`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
5     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
▶     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

B, satır 0: data[0][0] → sayfa 0'da hata, ardından data[0][1…127] aynı sayfada: **isabet**.

Adım 6 / 8

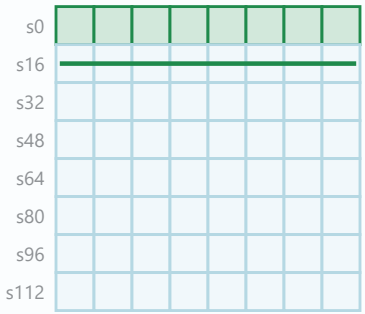
DEĞİŞKENLER

`i = 0` `j = 0..127` `sayfa = 0` `hata = 1`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
5     data[i][j] = 0;
6 /* B: satır satır */
7 for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
▶     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

Satır 1 → sayfa 1'de tek hata; satır başına bir hata.

Adım 7 / 8

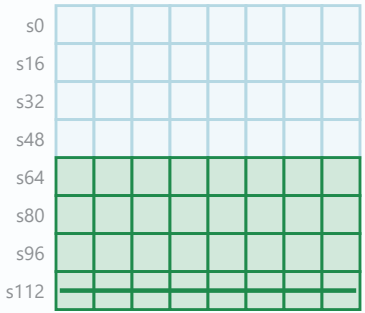
DEĞİŞKENLER

`i = 1` `j = 0` `sayfa = 1` `hata = 2`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

Program Yapısı ve Yerellik

```
1 int data[128][128];   /* satır = 1 sayfa */
2 /* A: sütun sütun */
3 for (int j = 0; j < 128; j++)
4   for (int i = 0; i < 128; i++)
5     data[i][j] = 0;
6 /* B: satır satır */
▶ for (int i = 0; i < 128; i++)
8   for (int j = 0; j < 128; j++)
9     data[i][j] = 0;
```



yeşil: bellekteki satırlar (sayfalar)

64 çerçeve
LRU
128 sayfa

Toplam **128 hata**: aynı iş, 128 kat daha az hata.

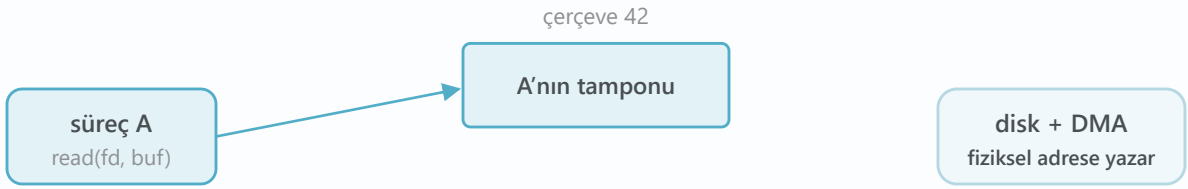
Adım 8 / 8

DEĞİŞKENLER

`i = 128` `hata = 128`

C satır öncelikli (row-major), Fortran sütun öncelikli saklar. Yığın iyi, hash tablosu ve işaretçi kovalayan yapılar kötü yerellik gösterir; derleyiciler döngü değiştirme (loop interchange) yapabilir.

G/Ç Kilitleme ve Sayfa Sabitleme

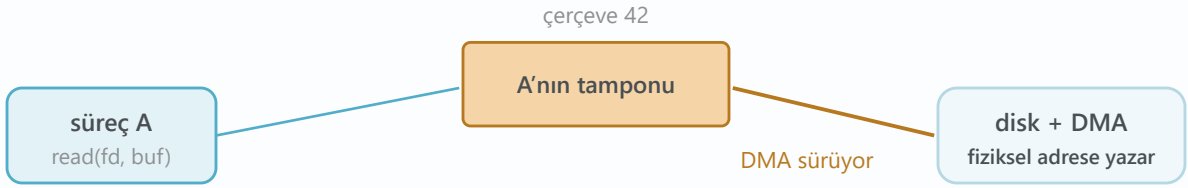


A, diskten okuma ister; çekirdek DMA'yı tamponun bulunduğu **çerçeve 42**'nin fiziksel adresiyle programlar.

Adım 1 / 4

- Sabitlenen başka sayfalar: çekirdek kodu ve verisinin bir kısmı, gerçek zamanlı süreçlerin belleği (`mlock()` , `mlockall()`).
- Linux sürücüleri kullanıcı sayfalarını `pin_user_pages()` ile sabitler (RDMA, GPU, io_uring sabit tamponları). IOMMU, aygıtların da çevrilmiş adres kullanmasını sağlar.
- Yeni yüklenmiş ama henüz hiç çalışmamış düşük öncelikli sürecin sayfası da bir kez kullanılana kadar korunabilir.
- Risk: kilitli sayfa unutulursa çerçeve sonsuza dek kullanılamaz; bu yüzden kilitleme sınırlıdır (`ulimit -l`).

G/Ç Kilitleme ve Sayfa Sabitleme



A bloke olur, aygıt kuyrukta/çalışıyor. Çerçeve 42 bu sırada sıradan bir sayfa gibi görünür.

- Sabitlenen başka sayfalar: çekirdek kodu ve verisinin bir kısmı, gerçek zamanlı süreçlerin belleği (`mlock()` , `mlockall()`).
- Linux sürücüleri kullanıcı sayfalarını `pin_user_pages()` ile sabitler (RDMA, GPU, io_uring sabit tamponları). IOMMU, aygıtların da çevrilmiş adres kullanmasını sağlar.
- Yeni yüklenmiş ama henüz hiç çalışmamış düşük öncelikli sürecin sayfası da bir kez kullanılana kadar korunabilir.
- Risk: kilitli sayfa unutulursa çerçeve sonsuza dek kullanılamaz; bu yüzden kilitleme sınırlıdır (`ulimit -l`).

G/Ç Kilitleme ve Sayfa Sabitleme

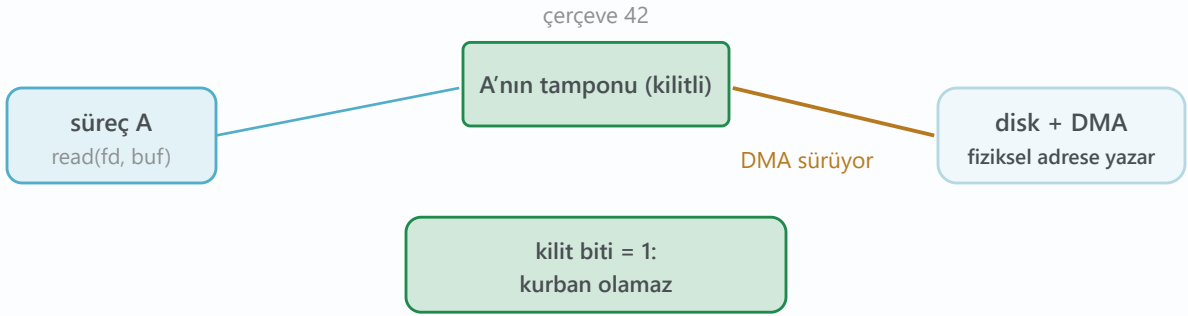


B sayfa hatası yapar; genel politika 42'yi kurban seçer ve B'nin sayfasını yükler. DMA tamamlanınca **B'nin sayfasının üstüne** yazar: bozulma.

Adım 3 / 4

- Sabitlenen başka sayfalar: çekirdek kodu ve verisinin bir kısmı, gerçek zamanlı süreçlerin belleği (`mlock()` , `mlockall()`).
- Linux sürücüleri kullanıcı sayfalarını `pin_user_pages()` ile sabitler (RDMA, GPU, io_uring sabit tamponları). IOMMU, aygıtların da çevrilmiş adres kullanmasını sağlar.
- Yeni yüklenmiş ama henüz hiç çalışmamış düşük öncelikli sürecin sayfası da bir kez kullanılana kadar korunabilir.
- Risk: kilitli sayfa unutulursa çerçeve sonsuza dek kullanılamaz; bu yüzden kilitleme sınırlıdır (`ulimit -l`).

G/Ç Kilitleme ve Sayfa Sabitleme



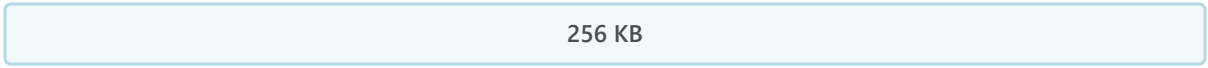
Çözüm: G/Ç süresince çerçeve **kilitlenir** (pinned) ve kurban seçilmez; ya da G/Ç çekirdek tamponlarına yapılıp sonra kopyalanır.

Adım 4 / 4

- Sabitlenen başka sayfalar: çekirdek kodu ve verisinin bir kısmı, gerçek zamanlı süreçlerin belleği (`mlock()`, `mlockall()`).
- Linux sürücüleri kullanıcı sayfalarını `pin_user_pages()` ile sabitler (RDMA, GPU, io_uring sabit tamponları). IOMMU, aygıtların da çevrilmiş adres kullanmasını sağlar.
- Yeni yüklenmiş ama henüz hiç çalışmamış düşük öncelikli sürecin sayfası da bir kez kullanılana kadar korunabilir.
- Risk: kilitli sayfa unutulursa çerçeve sonsuza dek kullanılamaz; bu yüzden kilitleme sınırlıdır (`ulimit -l`).

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.



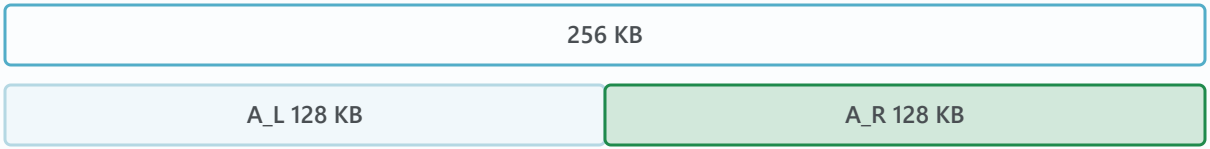
Çekirdeğin elinde fiziksel olarak **bitişik** 256 KB'lik bir bölge var; 21 KB'lik istek geliyor.

Adım 1 / 6

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.

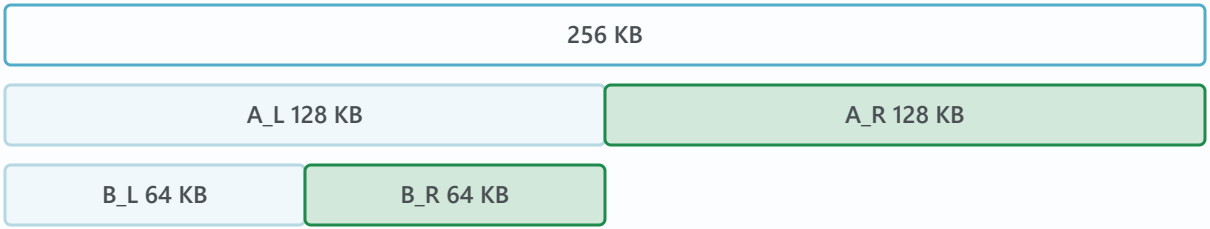


Blok iki eş **kardeşe** (buddy) bölünür: A_L ve A_R, 128'er KB.

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.

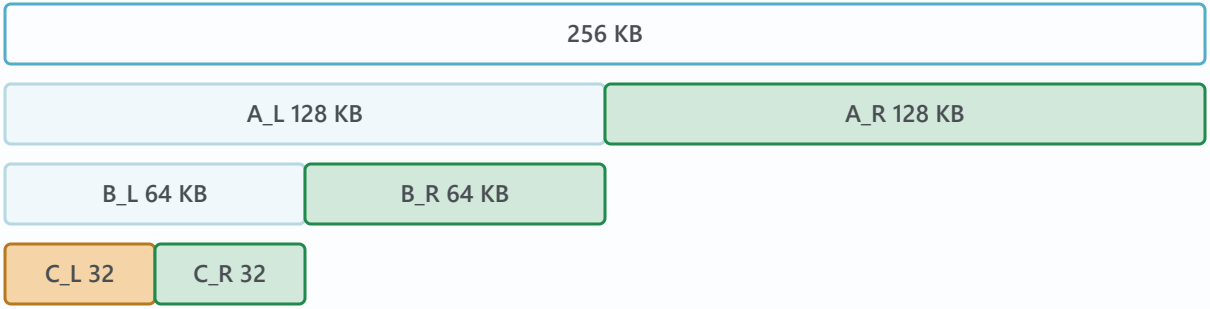


A_L yine bölünür: B_L, B_R (64 KB).

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.

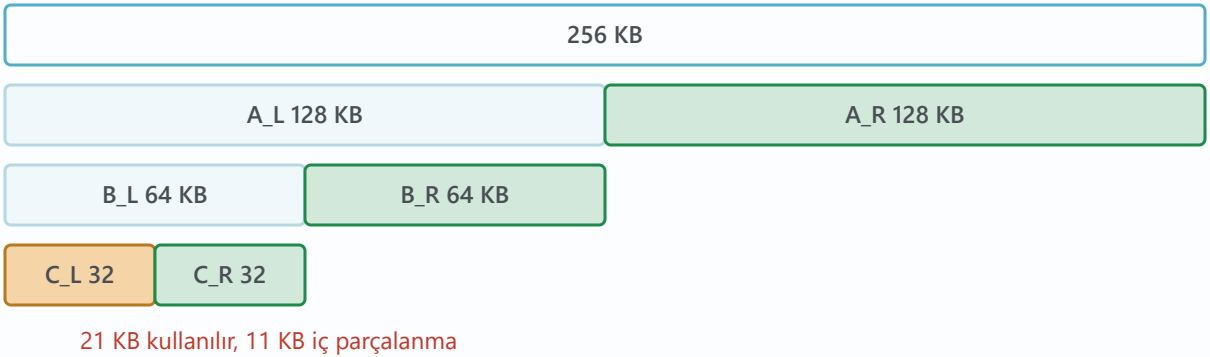


B_L bölünür: C_L, C_R (32 KB). 21 KB'yi karşılayan en küçük 2'nin kuvveti 32 KB'dir → C_L verilir.

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.

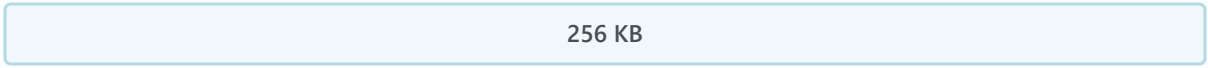


Bedeli iç parçalanma: istenen 21, verilen 32 KB.

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Çekirdek Belleği: Buddy Sistemi

Çekirdek belleği kullanıcı belleğinden farklıdır: çoğu zaman **sayfalanmaz**, farklı boyutlu yapılar ister ve DMA için **fiziksel olarak bitişik** olmalıdır.



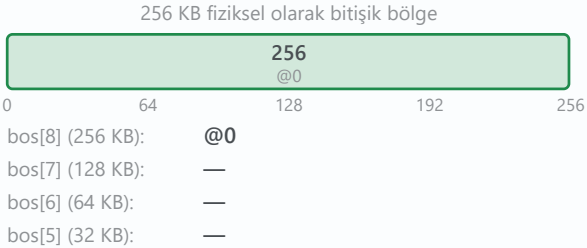
C_L + C_R → B_L, B_L + B_R → A_L, A_L + A_R → 256 KB

Bırakılınca kardeşler boşsa **hızla birleşir** (coalescing): kardeşin adresi tek bir bit değiştirerek bulunur.

Linux sayfa ayırıcısı buddy'dir: 0'dan 10'a kadar mertebe (4 KB … 4 MB) listeleri, `cat /proc/buddyinfo`.

Buddy Ayırıcı: Kod İzleme

```
▶ int ayir(int k) {          /* 2^k KB */
▶   int j = k;
3   while (bos[j] == NULL) j++;
4   int a = cek(j);
5   while (j > k) {   /* böl, sağ yarı boş */
6     j--; ekle(j, a + (1 << j));
7   }
8   return a;
9 }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



Tek boş blok: @0'da 256 KB (mertebe 8). 21 KB'lik istek 32 KB'ye (2^5) yuvarlanır: `A = ayir(5)`.

Adım 1 / 12

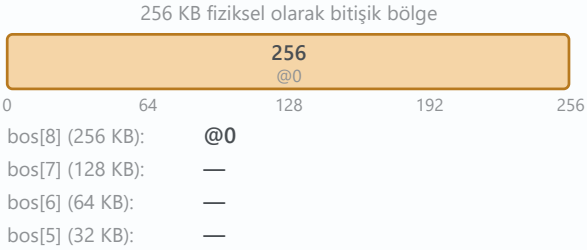
DEĞİŞKENLER

`k = 5` `j = 5` `a = –`

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
▶    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



bos[5], bos[6], bos[7] boş; ilk dolu liste bos[8]: j = 8. Daha büyük bir blok bölünecek.

Adım 2 / 12

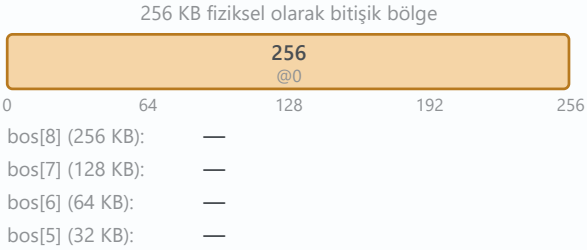
DEĞİŞKENLER

k = 5 | j = 8 | a = –

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
▶    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



256 KB'lik blok listeden alınır: a = 0.

Adım 3 / 12

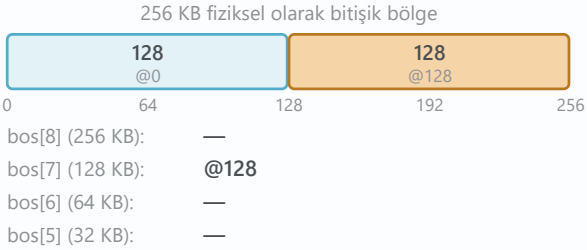
DEĞİŞKENLER

`k = 5` `j = 8` `a = 0`

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
▶    while (j > k) {   /* böl, sağ yarı boş */
▶      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



j = 7: blok ikiye bölünür; sağ yarı @128 (128 KB) bos[7]'ye eklenir, sol yarı bölünmeye devam eder.

Adım 4 / 12

DEĞİŞKENLER

`k = 5` `j = 7` `a = 0`

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
▶    while (j > k) {   /* böl, sağ yarı boş */
▶      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



bos[8] (256 KB): —
bos[7] (128 KB): @128
bos[6] (64 KB): @64
bos[5] (32 KB): —

j = 6: blok ikiye bölünür; sağ yarı @64 (64 KB) bos[6]'ya eklenir, sol yarı bölünmeye devam eder.

Adım 5 / 12

DEĞİŞKENLER

`k = 5` `j = 6` `a = 0`

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
▶    while (j > k) {   /* böl, sağ yarı boş */
▶      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



bos[8] (256 KB): —
bos[7] (128 KB): @128
bos[6] (64 KB): @64
bos[5] (32 KB): @32

j = 5: son bölme; @32 bos[5]'e gider. j = k olduğu için döngü biter ve sol yarı (@0, 32 KB) **A**'ya verilir.

Adım 6 / 12

DEĞİŞKENLER

`k = 5` `j = 5` `a = 0`

ÇIKTI

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
▶    while (bos[j] == NULL) j++;
▶    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
▶    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



bos[8] (256 KB): —
bos[7] (128 KB): @128
bos[6] (64 KB): —
bos[5] (32 KB): @32

`B = ayir(6)` : bos[6]'da @64 hazır; bölme gerekmez, hemen döner. İç parçalanma: A'da 32 − 21 = 11 KB.

Adım 7 / 12

DEĞİŞKENLER

`k = 6` `j = 6` `a = 64`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
▶ void birak(int a, int k) {
▶   int b = a ^ (1 << k);    /* kardeş */
12   while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;     /* birleş */
14     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



bos[8] (256 KB): —
bos[7] (128 KB): @128
bos[6] (64 KB): —
bos[5] (32 KB): @32

`birak(0, 5)` : A geri veriliyor. Kardeşin adresi tek bit çevrilerek bulunur: 0 ⊕ 32 = **32**.

Adım 8 / 12

DEĞİŞKENLER

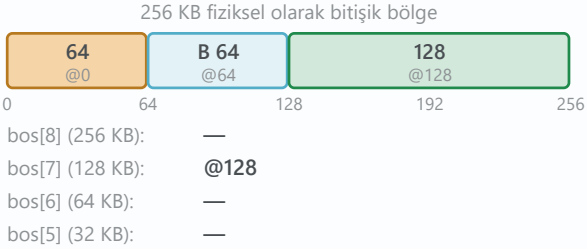
`a = 0` `k = 5` `b = 32`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
▶    while (k < MAX && sil(k, b)) {
▶      a = a < b ? a : b;     /* birleş */
▶      k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



@32 bos[5]'te: kardeş boş, **birleşirler** → @0'da 64 KB. Yeni kardeş 0 ⊕ 64 = 64.

Adım 9 / 12

DEĞİŞKENLER

`a = 0` `k = 6` `b = 64`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);     /* kardeş */
▶    while (k < MAX && sil(k, b)) {
13     a = a < b ? a : b;      /* birleş */
14     k++; b = a ^ (1 << k);
15   }
▶    ekle(k, a);
17 }
```



bos[8] (256 KB): —
bos[7] (128 KB): @128
bos[6] (64 KB): @0
bos[5] (32 KB): —

@64 bos[6]'da değil (B kullanıyor): birleşme durur, @0 (64 KB) bos[6]'ya eklenir.

Adım 10 / 12

DEĞİŞKENLER

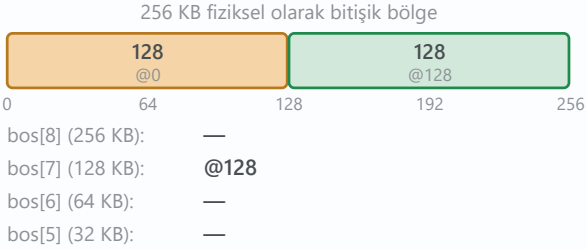
`a = 0` `k = 6` `b = 64`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
▶ void birak(int a, int k) {
▶   int b = a ^ (1 << k);    /* kardeş */
▶   while (k < MAX && sil(k, b)) {
▶     a = a < b ? a : b;     /* birleş */
▶     k++; b = a ^ (1 << k);
15   }
16   ekle(k, a);
17 }
```



`birak(64, 6)` : kardeşi 64 ⊕ 64 = 0 boş → 128 KB'lik @0 oluşur; yeni kardeş @128.

Adım 11 / 12

DEĞİŞKENLER

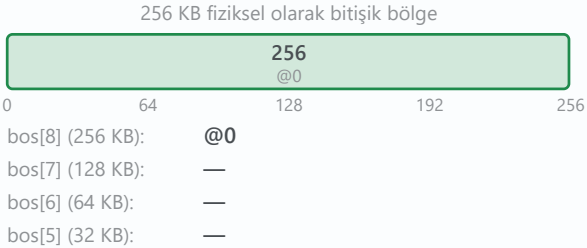
`a = 0` `k = 7` `b = 128`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

Buddy Ayırıcı: Kod İzleme

```
1  int ayir(int k) {          /* 2^k KB */
2    int j = k;
3    while (bos[j] == NULL) j++;
4    int a = cek(j);
5    while (j > k) {   /* böl, sağ yarı boş */
6      j--; ekle(j, a + (1 << j));
7    }
8    return a;
9  }
10 void birak(int a, int k) {
11   int b = a ^ (1 << k);    /* kardeş */
▶    while (k < MAX && sil(k, b)) {
▶      a = a < b ? a : b;     /* birleş */
▶      k++; b = a ^ (1 << k);
15   }
▶    ekle(k, a);
17 }
```



@128 de boş: birleşip 256 KB olur; k = MAX olduğundan durur. Bölge başlangıçtaki gibi **tek parça**.

Adım 12 / 12

DEĞİŞKENLER

`a = 0` `k = 8` `b = –`

ÇIKTI

```
A = @0 (32 KB)
B = @64 (64 KB)
```

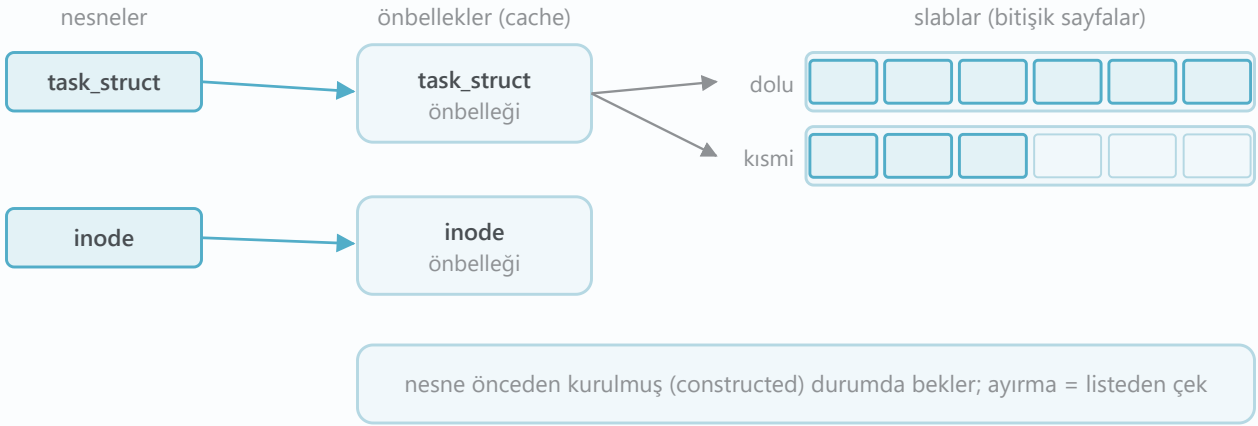
Buddy: İstek Dizisi Örneği

1 MiB (1024 K) tek boş blokla başlanır; her istek 2'nin bir üst kuvvetine yuvarlanır. Kardeşin adresi *adres XOR boyut* ile bulunur.

Olay	Blok	Bölme ya da birleşme	İç parçalanma
A = 70 K	128 K	1024 → 512 + 512 → 256 + 256 → 128 + 128	58 K
B = 35 K	64 K	128 → 64 + 64	29 K
C = 80 K	128 K	256 → 128 + 128	48 K
A bırakılır	—	kardeşi (128–256) bölünmüş, içinde B var → birleşme yok	—
D = 60 K	64 K	B'nin boş kardeşi verilir, bölme gerekmez	4 K
B bırakılır	—	kardeşi D dolu → birleşme yok	—
D bırakılır	—	64 + 64 → 128, A'nın boş 128'iyle → 256	—
C bırakılır	—	128 + 128 → 256 → 512 → 1024: bellek yine tek blok	—

A, B ve C için toplam iç parçalanma 58 + 29 + 48 = 135 K. Buddy tahsis ve birleştirmede hızlıdır (boyut başına ayrı boş liste, O(log n)), ama en kötü durumda bloğun yaklaşık yarısı boşa gider.

Slab Ayırıcı

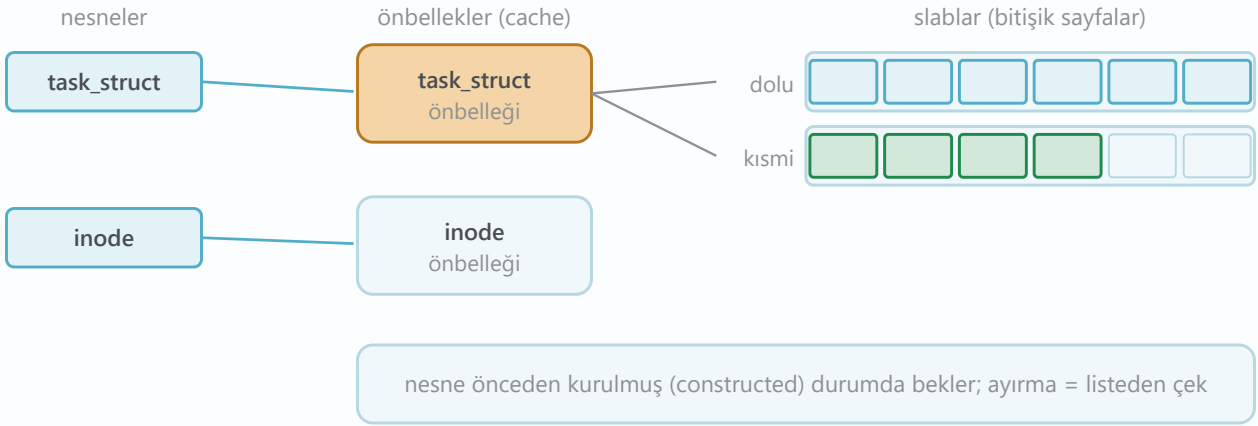


Her sık kullanılan çekirdek nesnesi için bir **önbellek**; önbellek bir ya da daha çok **slab**'dan, slab bitişik sayfalardan oluşur.

Adım 1 / 4

- Bonwick (Solaris, 1994). Yararları: iç parçalanma yok, ayırma/bırakma O(1), önbellekte sıcak ve hazır nesneler, CPU başına listelerle kilitsiz hızlı yol.
- Linux: SLOB 6.4'te, klasik SLAB 6.8'de (2024) kaldırıldı; bugün tek ayırıcı **SLUB**'dır. `kmalloc()` boyut sınıflı slab önbelleklerinden gelir; izleme: `slabtop`, `/proc/slabinfo`.

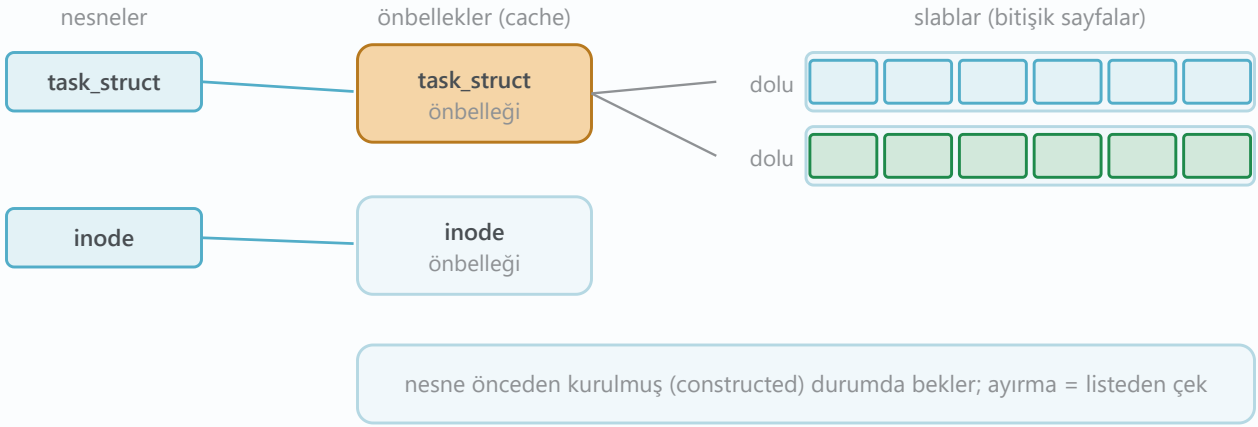
Slab Ayırıcı



Yeni `task_struct` isteği (ör. `fork()`): önce **kısmi** slabdaki boş nesneye bakılır; bulunur, verilir.

- Bonwick (Solaris, 1994). Yararları: iç parçalanma yok, ayırma/bırakma O(1), önbellekte sıcak ve hazır nesneler, CPU başına listelerle kilitsiz hızlı yol.
- Linux: SLOB 6.4'te, klasik SLAB 6.8'de (2024) kaldırıldı; bugün tek ayırıcı **SLUB**'dır. `kmalloc()` boyut sınıflı slab önbelleklerinden gelir; izleme: `slabtop`, `/proc/slabinfo`.

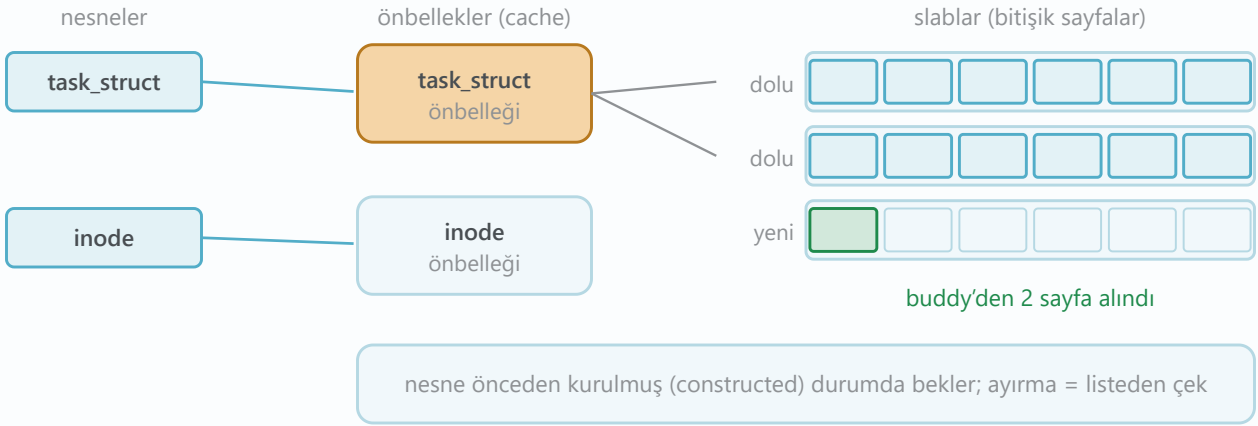
Slab Ayırıcı



Kısmi slab doldu. Parçalanma yok: nesneler tam boyutludur ve sayfa içinde yan yana paketlenir.

- Bonwick (Solaris, 1994). Yararları: iç parçalanma yok, ayırma/bırakma O(1), önbellekte sıcak ve hazır nesneler, CPU başına listelerle kilitsiz hızlı yol.
- Linux: SLOB 6.4'te, klasik SLAB 6.8'de (2024) kaldırıldı; bugün tek ayırıcı **SLUB**'dır. `kmalloc()` boyut sınıflı slab önbelleklerinden gelir; izleme: `slabtop`, `/proc/slabinfo`.

Slab Ayırıcı



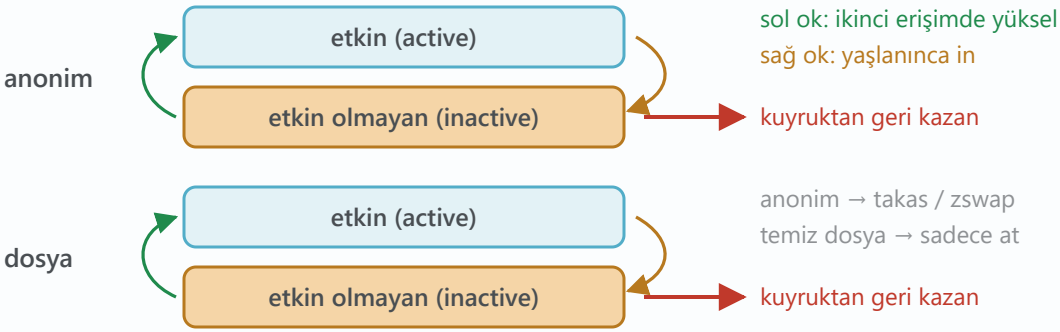
Kısmi ya da boş slab kalmadı: buddy ayırıcıdan yeni sayfalar alınır, yeni slab oluşturulur. Boşalan slablar bellek baskısında geri verilir.

- Bonwick (Solaris, 1994). Yararları: iç parçalanma yok, ayırma/bırakma O(1), önbellekte sıcak ve hazır nesneler, CPU başına listelerle kilitsiz hızlı yol.
- Linux: SLOB 6.4'te, klasik SLAB 6.8'de (2024) kaldırıldı; bugün tek ayırıcı **SLUB**'dır. `kmalloc()` boyut sınıflı slab önbelleklerinden gelir; izleme: `slabtop`, `/proc/slabinfo`.

GERÇEK SİSTEMLER

Linux: Sayfa Geri Kazanımı

- Boş sayfalar *low* su düzeyinin altına inince **kswapd** uyanır; *min* altında ayırma yapan süreç **doğrudan geri kazanım** yapar (gecikme!).
- Klasik LRU, saat benzeri iki liste çiftidir: **etkin / etkin olmayan**, anonim ve dosya için ayrı (ayrıca kilitli sayfalar için *unevictable*).
- `vm.swappiness` (0–200) anonim ile dosya sayfaları arasındaki tercihi ayarlar. Bellek tükenirse **OOM killer** bir süreci öldürür.



GERÇEK SİSTEMLER

Linux: MGLRU, zswap ve zram

MGLRU (6.1, 2022)

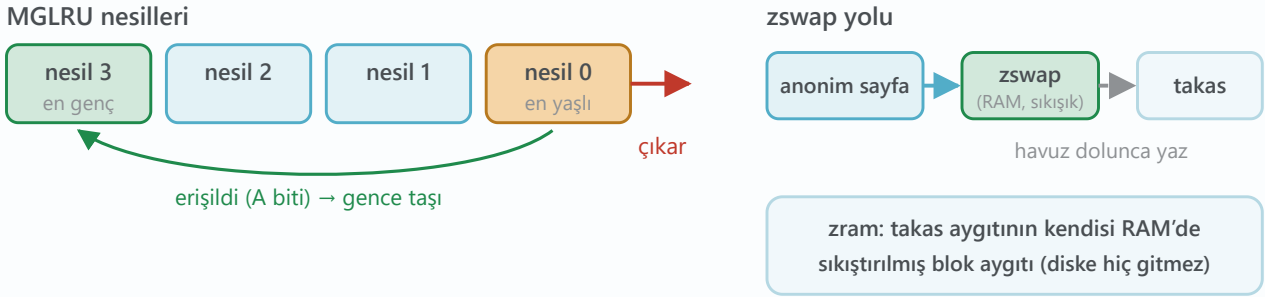
Çok nesilli LRU: sayfalar 4 nesle ayrılır; erişim bitleri sayfa tabloları yürünerek toplanır, kurban en yaşlı nesilden seçilir. Daha az CPU, daha iyi kurban seçimi. `/sys/kernel/mm/lru_gen/enabled`

zswap

Takasın önünde *sıkıştırılmış önbellek*: anonim sayfa diske gitmeden RAM'de sıkıştırılır; havuz dolunca en eski girdiler gerçek takasa yazılır.

zram

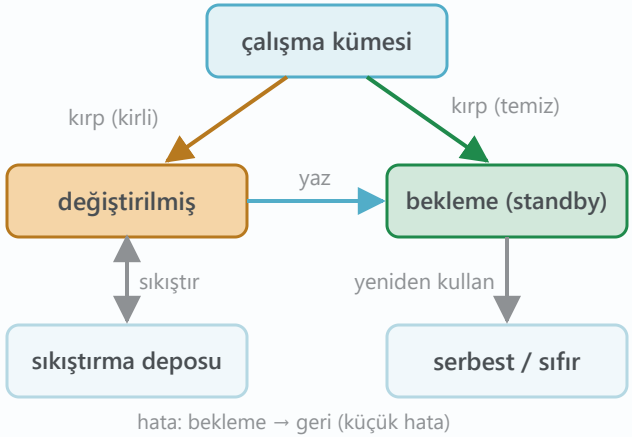
RAM'de sıkıştırılmış blok aygıtı, takas olarak kullanılır. Fedora (33'ten beri), Android ve ChromeOS varsayılan olarak kullanır.



MGLRU ChromeOS ve Android'de varsayılandır; birçok dağıtım çekirdeği derleme seçeneğiyle açar. Sıkıştırma CPU harcar ama diske gitmekten yüzlerce kat ucuzdur.

Windows: Çalışma Kümeleri ve Sıkıştırma

- Her sürecin bir **çalışma kümesi** vardır (en az / en çok sınırlarıyla); hata kümede yer açar ya da kümeyi büyütür.
- Bellek azalınca **çalışma kümesi yöneticisi** (balance set manager) kümeleri *kırpar*; kırpılan sayfa hemen silinmez, **bekleme** ya da **değiştirilmiş** listeye gider ve küçük hatayla geri alınabilir.
- **Bellek sıkıştırma** (Windows 10, 2015): kırpılan sayfalar sistem sürecindeki sıkıştırma deposunda tutulur; `pagefile.sys`'ye yazmak son çaredir.
- Önceden getirme: Prefetcher ve SysMain; büyük sayfalar `VirtualAlloc(MEM_LARGE_PAGES)`.
- macOS/iOS da benzer: sıkıştırılmış bellek, iOS'ta takas yok, baskıda uygulama sonlandırma (Jetsam).



Özet

Mekanizma

Talebe bağlı sayfalama: geçersiz girdi → sayfa hatası → sayfayı getir → tabloyu güncelle → komutu yeniden başlat. COW ve mmap aynı mekanizmayı kullanır.

Performans

EAT, hata oranına aşırı duyarlı: HDD'de p = 0,001 belleği 251 kat yavaşlatır. Hızlı SSD ve sıkıştırma cezayı küçültür, yok etmez.

Politika

Ortak dizide (3 çerçeve) OPT 9, LRU 12, saat 14, FIFO 15 hata. Pratikte saat, yaşlandırma ve WSClock türevleri; Linux'ta MGLRU.

Sistem

Tahsis (eşit/orantılı, yerel/genel), thrashing'e karşı çalışma kümesi ve PFF, sayfa boyutu ve TLB erimi, sabitleme, buddy + slab.

Yerellik sayesinde süreçlerin yalnızca **çalışma kümeleri** bellekte olmalıdır; İS'nin işi, bu kümeyi az G/Ç ile bellekte tutmaktır.

ÖZET

Kontrol Soruları

1. `1 2 3 4 1 2 5 1 2 3 4 5` dizisinde FIFO 3 ve 4 çerçeveyle kaç hata yapar? LRU neden bu anomaliyi göstermez?
2. Bellek erişimi 200 ns, hata hizmeti 8 ms. p = 0,001 için EAT nedir? %10 yavaşlama için p en çok ne olmalı?
3. 62 çerçeve, 10 ve 127 sayfalık iki süreç: orantılı tahsis ne verir?
4. 64 girdili TLB, 4 KB ve 2 MB sayfa için TLB erimi nedir?
5. `fork()` sonrası çocuk 100 sayfadan 3'üne yazıp `exec()` çağırırsa kaç sayfa kopyalanır?
6. Küçük (minor) ve büyük (major) sayfa hatasını birer örnekle ayırın.
7. Thrashing neden CPU kullanımını düşürür? Çalışma kümesi modeli nasıl önler?
8. s = 4 MB, e = 8 B için en uygun sayfa boyutu nedir?
9. G/Ç sırasında bir tampon sayfası neden kilitlenmelidir?
10. 1 MiB buddy belleğinde 70 K'lık istek hangi boyutta blok alır? İç parçalanma ne kadardır?

▶ **Cevaplar**

Kaynaklar

- A. Silberschatz, P. B. Galvin, G. Gagne, *Operating System Concepts*, 10. baskı, Wiley, 2018 — Bölüm 10: Virtual Memory.
- A. S. Tanenbaum, H. Bos, *Modern Operating Systems*, 4. baskı (2014) / 5. baskı (2023), Pearson — Bölüm 3: Memory Management.
- R. H. Arpaci-Dusseau, A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces* (OSTEP), ostep.org — "Swapping: Mechanisms", "Swapping: Policies", "Complete VM Systems".
- L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Systems Journal* 5(2), 1966; P. J. Denning, "The Working Set Model for Program Behavior", *CACM* 11(5), 1968.
- R. W. Carr, J. L. Hennessy, "WSCLOCK — A Simple and Effective Algorithm for Virtual Memory Management", SOSP 1981; J. Bonwick, "The Slab Allocator", USENIX Summer 1994.
- Linux çekirdek belgeleri: `admin-guide/mm/multigen_lru.rst`, `admin-guide/mm/zswap.rst`, `admin-guide/mm/transhuge.rst`, `admin-guide/blockdev/zram.rst`.
- P. Yosifovich, A. Ionescu, M. Russinovich, D. Solomon, *Windows Internals*, 7. baskı, Kısım 1, Bölüm 5: Memory management.

Son

Bölüm 9: Sanal Bellek

Sonraki bölüm: Dosya Sistemleri · Terim sözlüğü