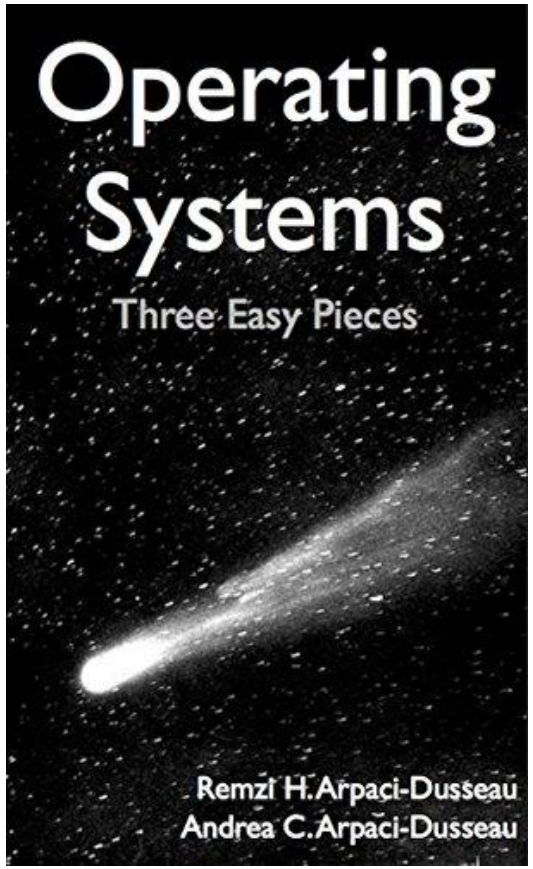
İşletim Sistemleri



5. Ders

Prof. Dr. Kemal Bıçakcı

Alıştırma Sorusu

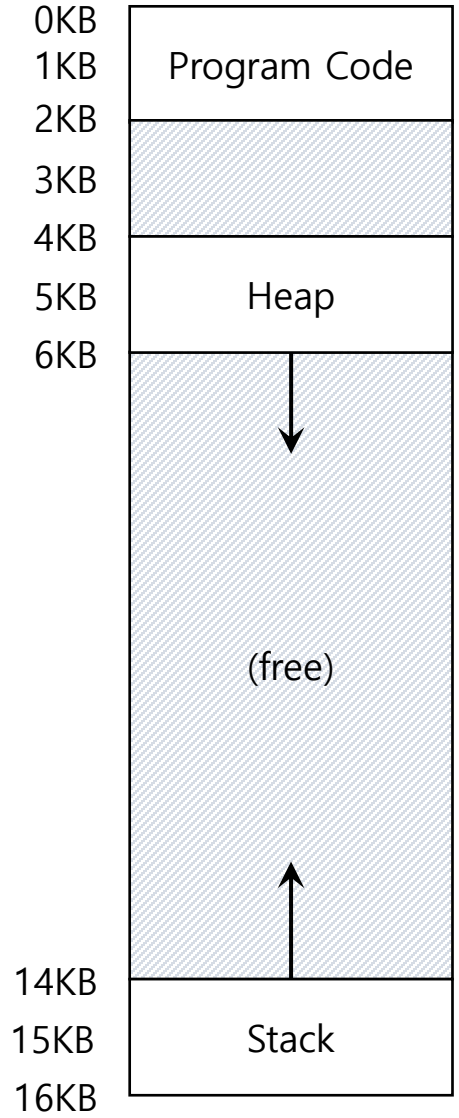
Aşağıdakilerden hangisi memory API kullanılırken yapılan programlama hatalarından biri değildir?

a) Yeterli bellek ayırmamak
b) Tahsis edilen bellekte başlangıç değerleri atamayı unutmak
c) Boşaltılacak belleğin boyutunu belirtmeyi unutmak
d) İş bitmeden belleği boşaltmak

16. Bölütleme (Segmentation)

Operating System: Three Easy Pieces

Taban ve Sınır Yaklaşımının Verimsizliği

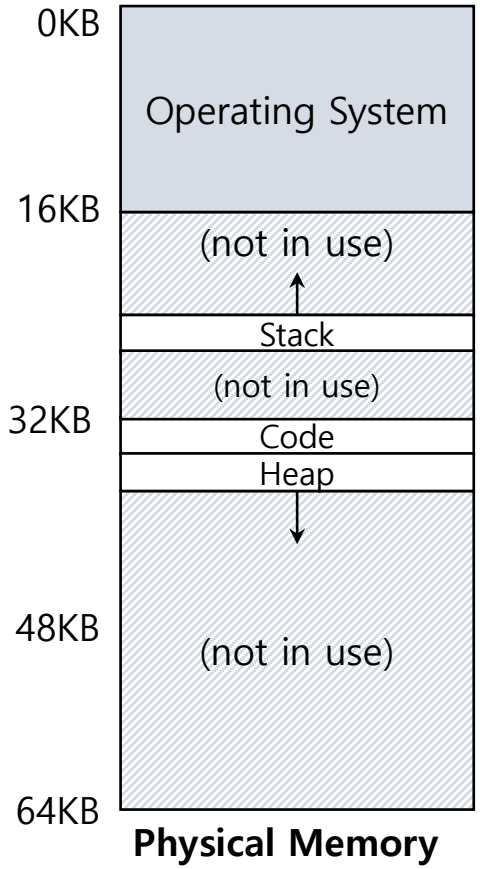


- Büyük bir "boş" alan var.
- "boş" alan fiziksel belleği de kaplar.
- Bir adres alanı fiziksel belleğe sığmayabilir.

Bölütleme (Segmentation)

- Bölüt (Segment): belirli uzunluktaki adres alanının yalnızca bitişik bir kısmına verilen ad.
- Mantıksal olarak farklı bölütler: kod, öbek, yığın.
- Her bölüt, fiziksel belleğin farklı bölümlerine yerleştirilebilir.
- Her bölüt için taban ve sınırlar mevcuttur.

Bölütü Fiziksel Belleğe Yerleştirmek

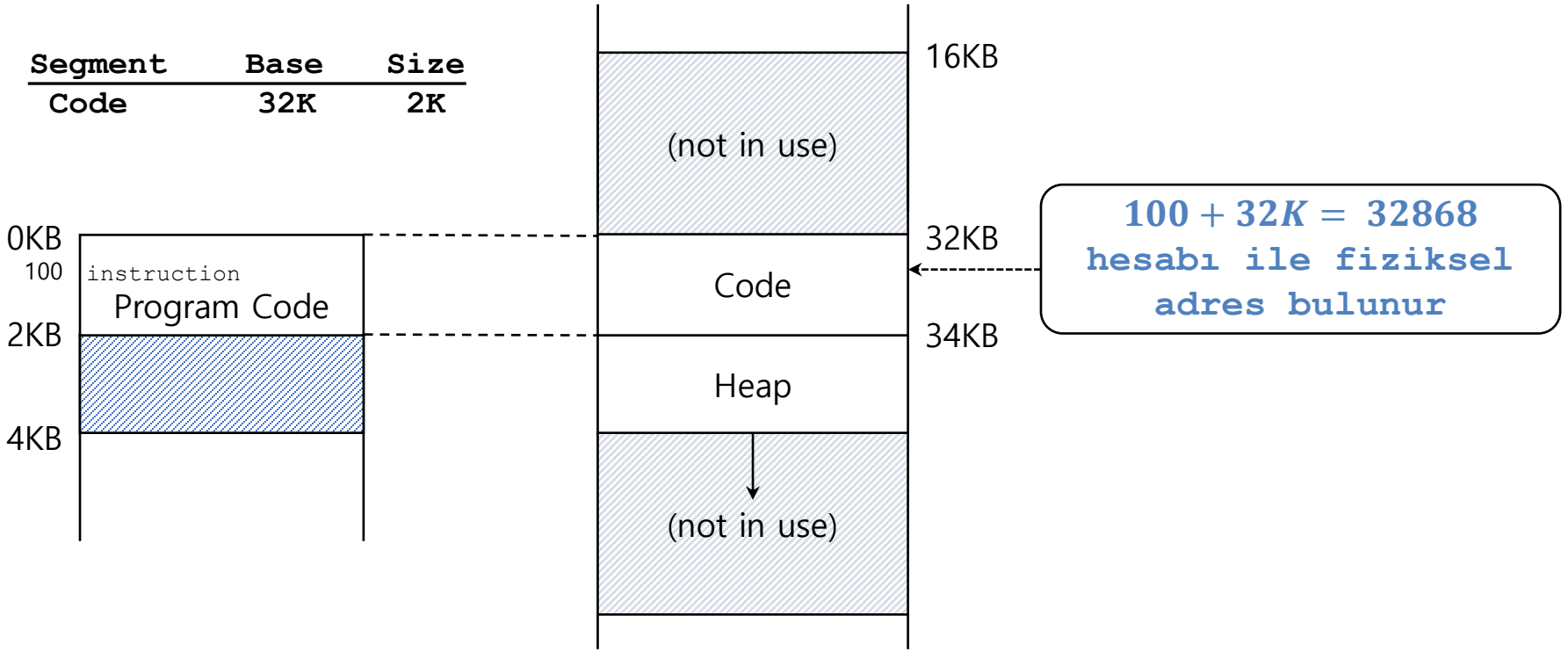


Segment	Base	Size
Code	32K	2K
Heap	34K	2K
Stack	28K	2K

Bölütlemede Adres Çevrimi

Fiziksel adres $= offset + taban$

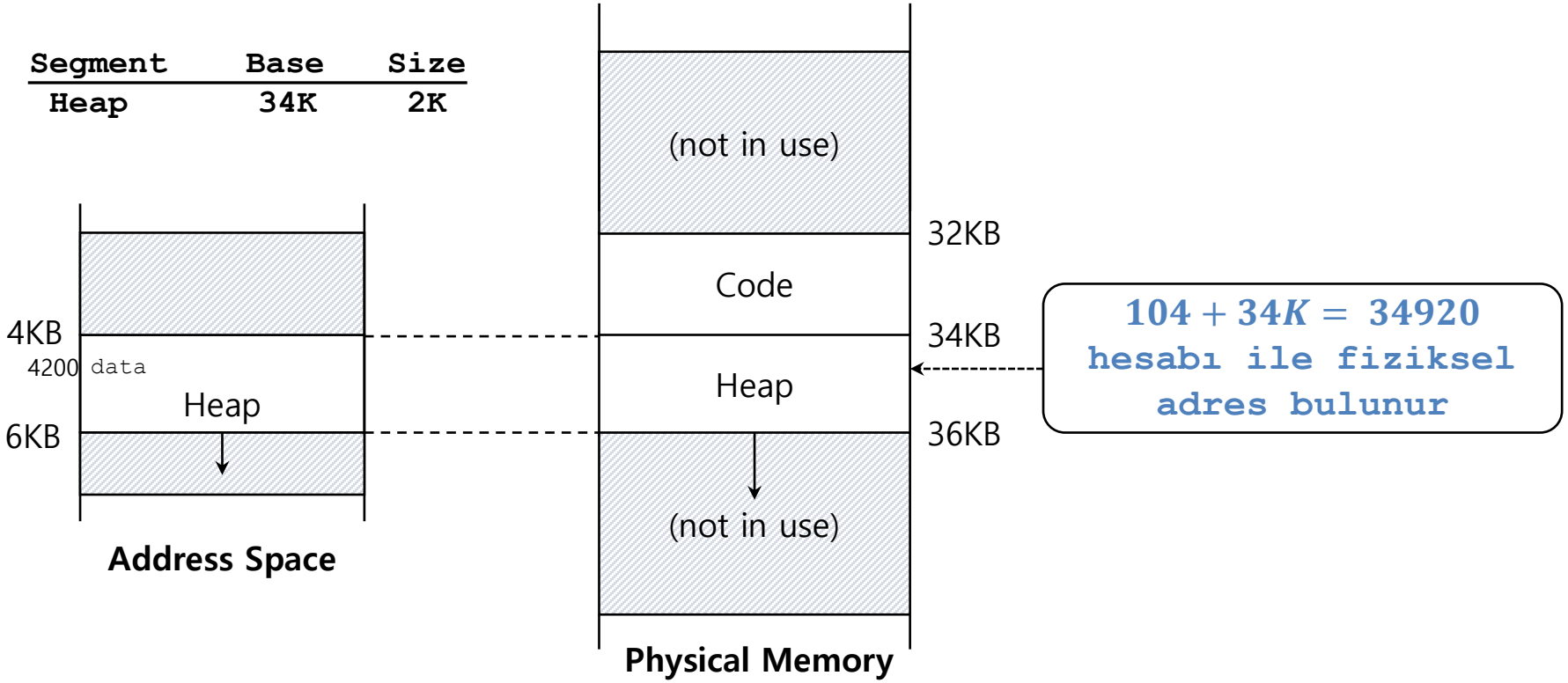
- Sanal adresin 100'ün `offset` değeri: `100`
  - Çünkü kod bölütü, adres uzayında sanal adres 0'da başlar.



Bölütlemede Adres Çevrimi (Devam)

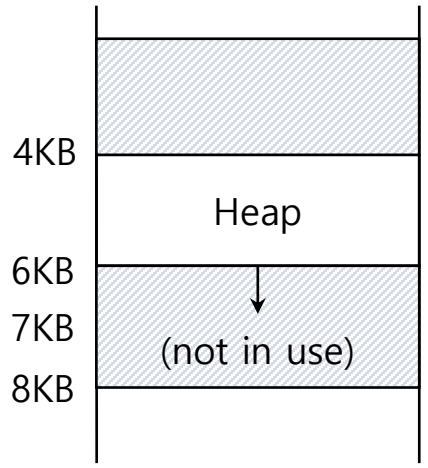
Sanal adres $+taban$ hesabı doğru fiziksel adresi vermez.

- Sanal adress 4200'un `offset` değeri: `104`
  - Öbek bölütü, adres alanında sanal adres 4096'da başlar.



«Segmentation Fault»

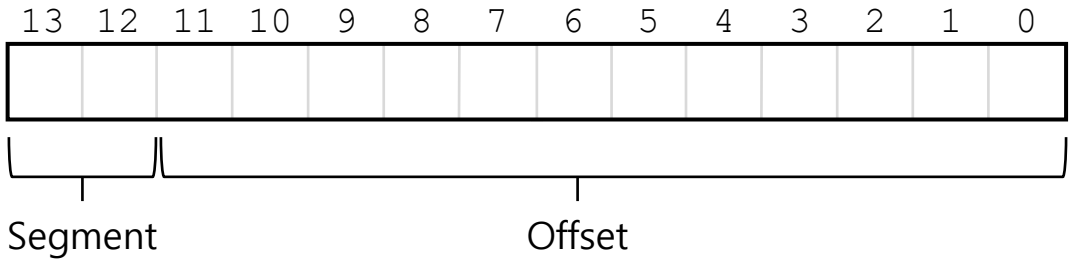
- 7KB gibi öbek sonunu aşan geçersiz bir adrese referans verilirse, işletim sisteminde segmentasyon hatası oluşur.
  - Donanım, adresin sınırların dışında olduğunu algılar.



Address Space

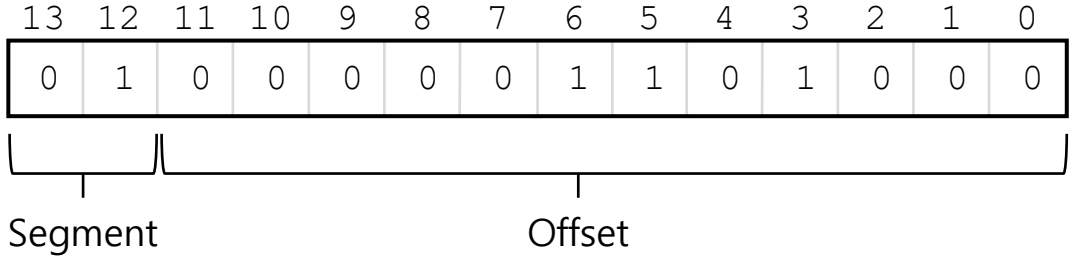
Bölüt Tespiti

- Adres uzayı, sanal adresin en üstteki birkaç bitine dayalı olarak bölütlere ayrılabilir.



- Örnek: sanal adres 4200 (0100001101000)

Segment	bits
Code	00
Heap	01
Stack	10
-	11



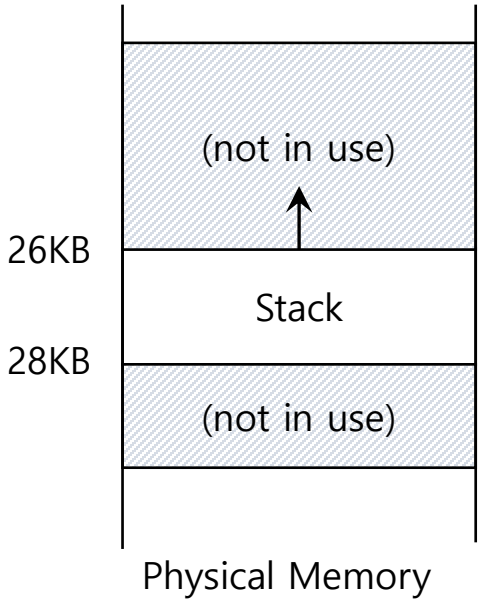
Bölüt Tespiti (Devam)

```
1   // get top 2 bits of 14-bit VA
2   Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT
3   // now get offset
4   Offset = VirtualAddress & OFFSET_MASK
5   if (Offset >= Bounds[Segment])
6        RaiseException(PROTECTION_FAULT)
7   else
8        PhysAddr = Base[Segment] + Offset
9        Register = AccessMemory(PhysAddr)
```

- SEG_MASK = 0x3000(11000000000000)
- SEG_SHIFT = 12
- OFFSET_MASK = 0xFFF (00111111111111)

Yığın Bölütünün Özel Durumu

- Yığın geriye doğru büyür.
- Bu yüzden ekstra donanım desteğine ihtiyaç vardır.
- Donanım, bölütün hangi yönde büyüdüğünü kontrol eder.
  - 1: pozitif yön, 0: negatif yön



Segment Register (with Negative-Growth Support)

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K	1
Stack	28K	2K	0

Paylaşma Desteği

- Segment, farklı adres uzayları tarafından paylaşılabilir.
- Kod paylaşımı günümüz sistemlerinde halen kullanılmaktadır.
- Koruma bitleri şeklinde ekstra donanım desteğine ihtiyaç vardır.
  - Okuma, yazma ve yürütme izinlerini belirtmek için bölüt başına yazmaçlara birkaç bit daha eklenir.

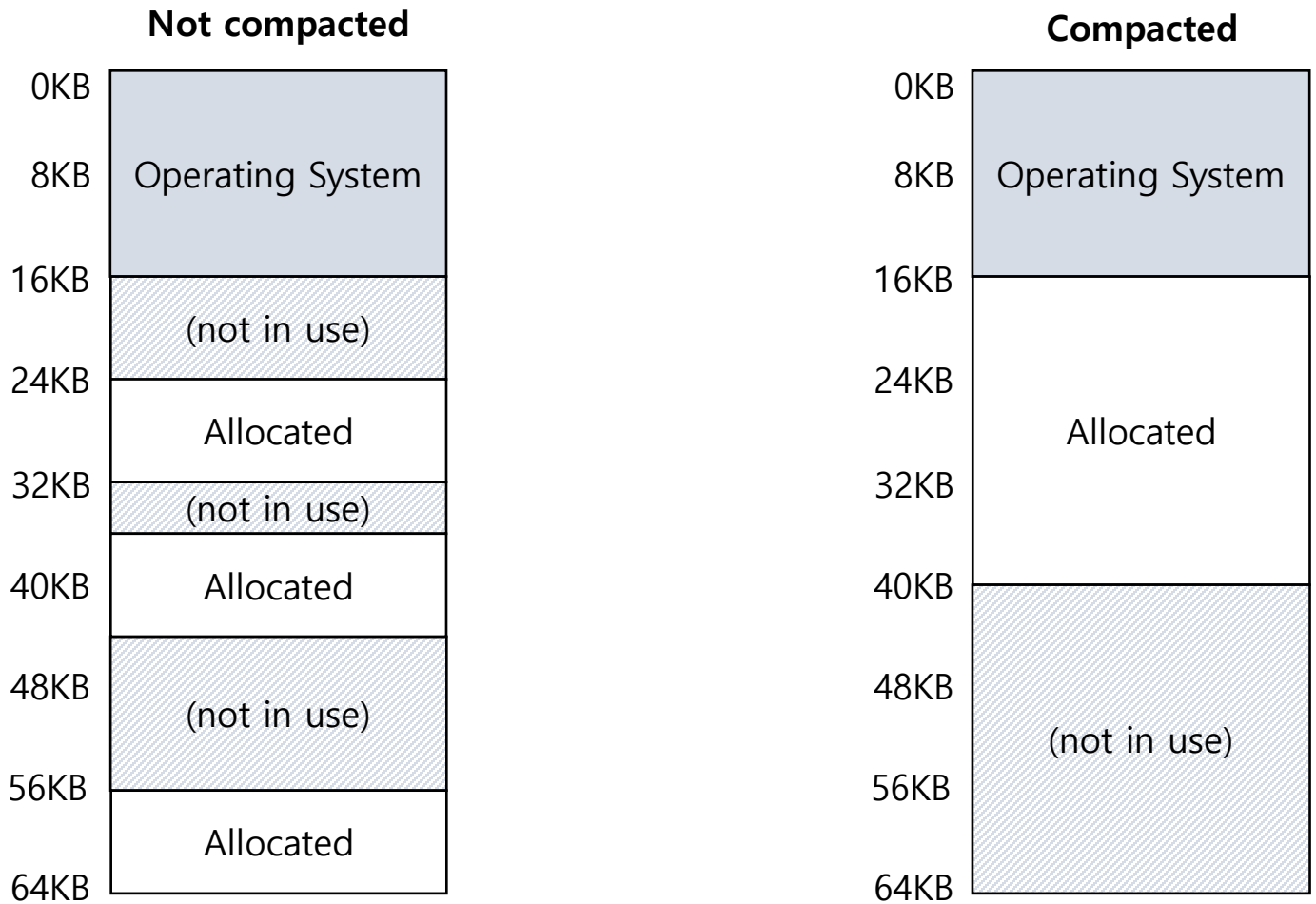
Segment Register Values (with Protection)

Segment	Base	Size	Grows Positive?	Protection
Code	32K	2K	1	Read-Execute
Heap	34K	2K	1	Read-Write
Stack	28K	2K	0	Read-Write

Parçalanma Problemine Yönelik İşletim Sistemi Desteği

- Dış Parçalanma (External Fragmentation): fiziksel bellekte yeni bölütler ayırmayı zorlaştıran küçük boş alan parçaları.
  - Toplam 24 KB boş alan var, ancak bitişik bir bölütte değil.
  - İşletim sistemi 20 KB'lık isteği karşılayamıyor.
- Sıkıştırma (Compaction): fiziksel bellekte mevcut bölütlerin yeniden düzenlenmesi.
- Sıkıştırma maliyetlidir.
  - Çalışan işlemi durdur.
  - Verileri bir yere kopyala.
  - Bölüt yazmaç değerlerini değiştir.

Bellek Sıkıştırma (Compaction)

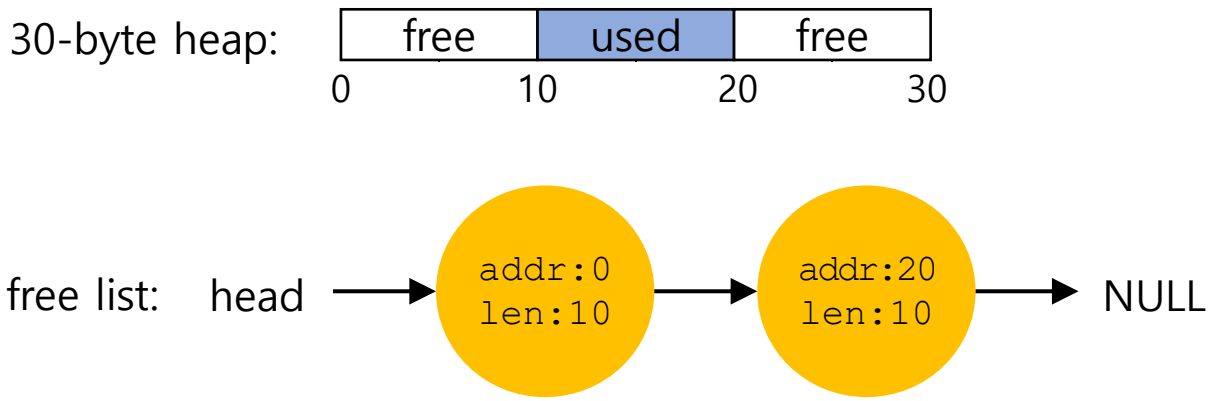


17. Boş Alan Yönetimi

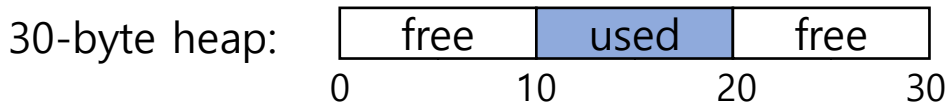
Operating System: Three Easy Pieces

Ayırma (Splitting)

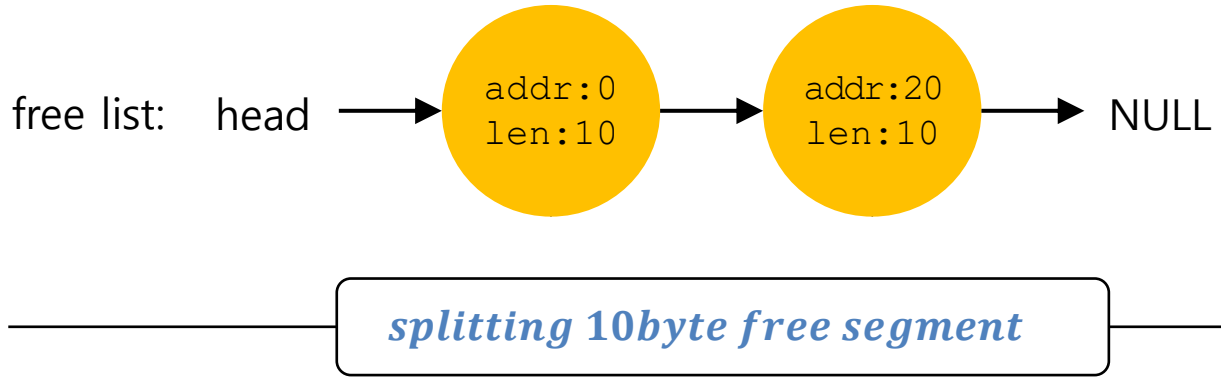
- İsteği karşılayabilecek boş bir bellek parçası bulmak ve onu ikiye bölmek.
- Bellek tahsisi talebi, boş parçaların boyutundan daha küçük olduğunda uygulanır.

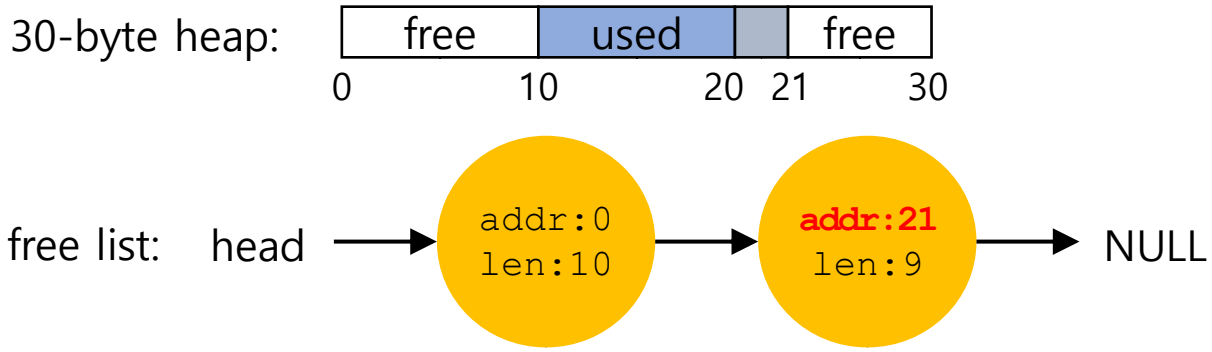


Ayırma (Devam)



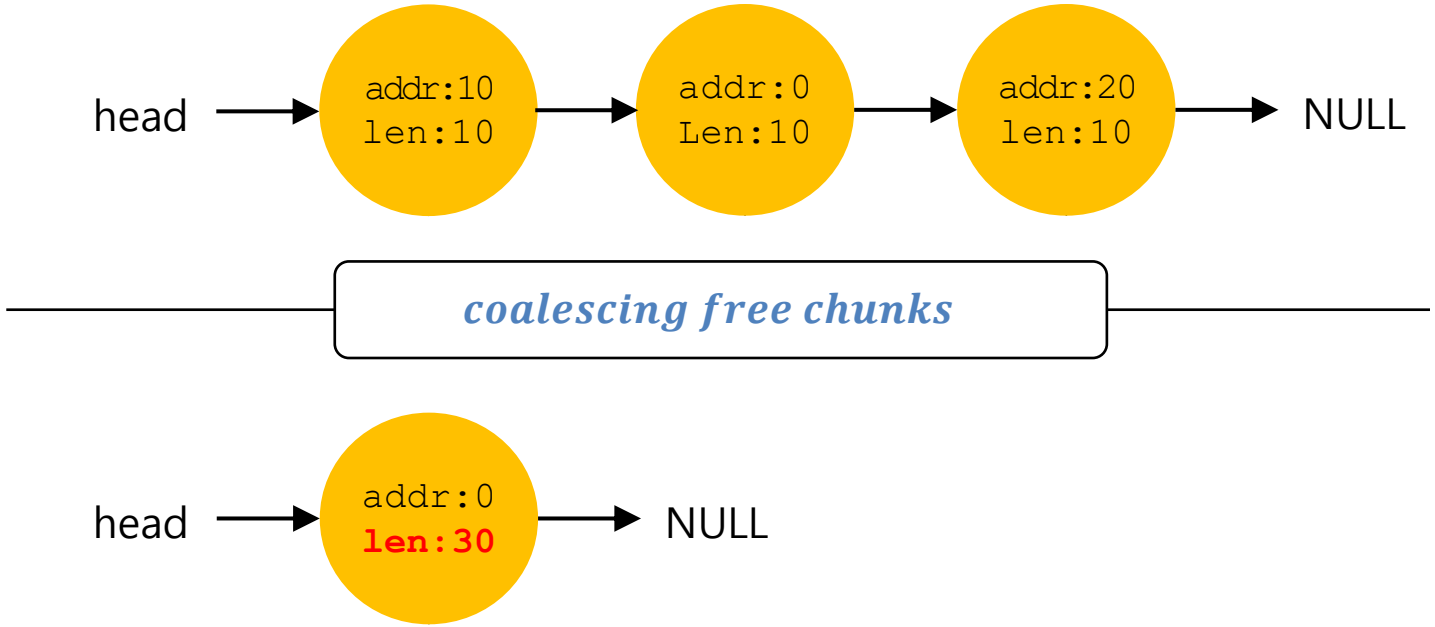
- İki tane 10 Baytlık boş bölüt var ve 1 Baytlık bir istek geliyor:





Birleştirme (Coalescing)

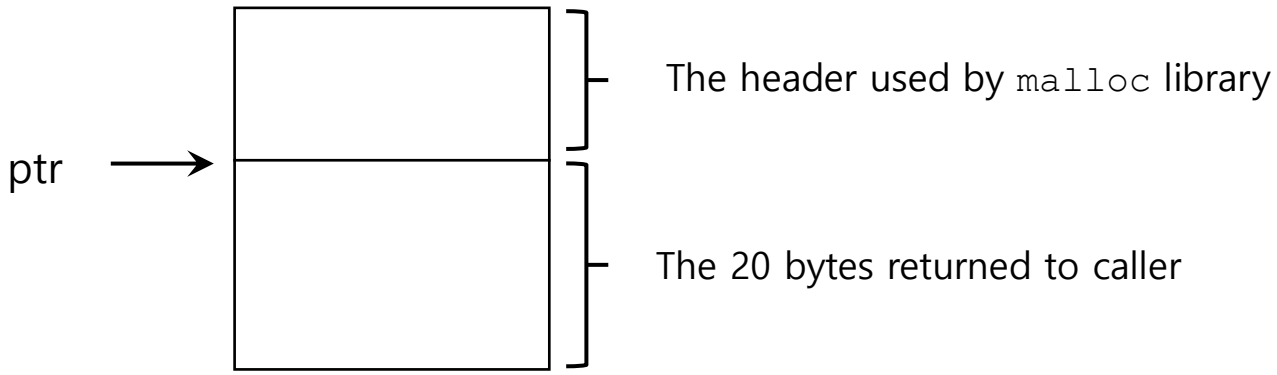
- Bir kullanıcı en büyük boş parça boyutundan daha büyük bir bellek isterse, liste böyle bir boş parça bulamaz.
- Birleştirme: Adresleri ardışıksa, boş parçaları, büyük tek bir boş parça elde etmek için bir araya getirmek.



Tahsis Edilen Alanların Büyüklüğünü İzleme

- free(void *ptr): bir boyut parametresini girdi olarak almaz.
- Bu durumda kütüphane boş listeye geri eklenecek bellek alanının boyutunu nasıl biliyor?
- Çoğu kez, bu problem ilave bilgilerin bir başlık bloğunda depolanması ile çözülür

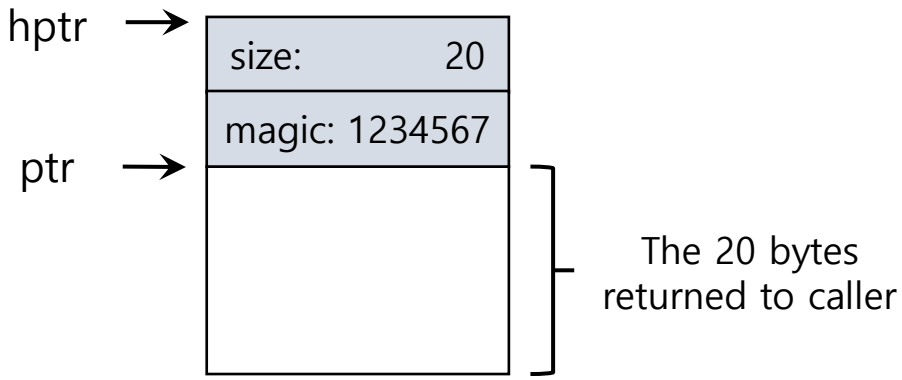
```
ptr = malloc(20);
```



An Allocated Region Plus Header

Tahsis Edilen Bellek Parçasının Başlığı

- Başlık (Header), minimumda ayrılan bellek alanının boyutunu içerir.
- Ayrıca, serbest bırakma işlemini hızlandırmak için ek işaretçiler ve bütünlük kontrolü için sihirli sayı (magic number) içerebilir.



Specific Contents Of The Header

```
typedef struct __header_t {
        int size;
        int magic;
} header_t;
```

A Simple Header

Tahsis Edilen Bellek Parçasının Başlığı (Devam)

- Boş bölgenin boyutu, başlığın boyutu artı kullanıcıya ayrılan alanın boyutudur.
- Bir kullanıcı N bayt isterse, kütüphane N + başlık boyutunda boş bir parça arar.

Başlık işaretçisini bulmak için basit işaretçi aritmetiği:

```
void free(void *ptr) {
        header_t *hptr = (void *)ptr - sizeof(header_t);
        . . .
}
```

Boş Listesini Hazırlama

- Bellek tahsis kütüphanesi, öbeği ilklendirmek ve boş listenin ilk öğesini boş alana koymakla işe başlar.
  - Kütüphane, bu amaç için `malloc()` kullanamaz.
  - Niye?

Boş Listesini Hazırlama (Devam)

- Listedeki bir düğümün tanımlanması:

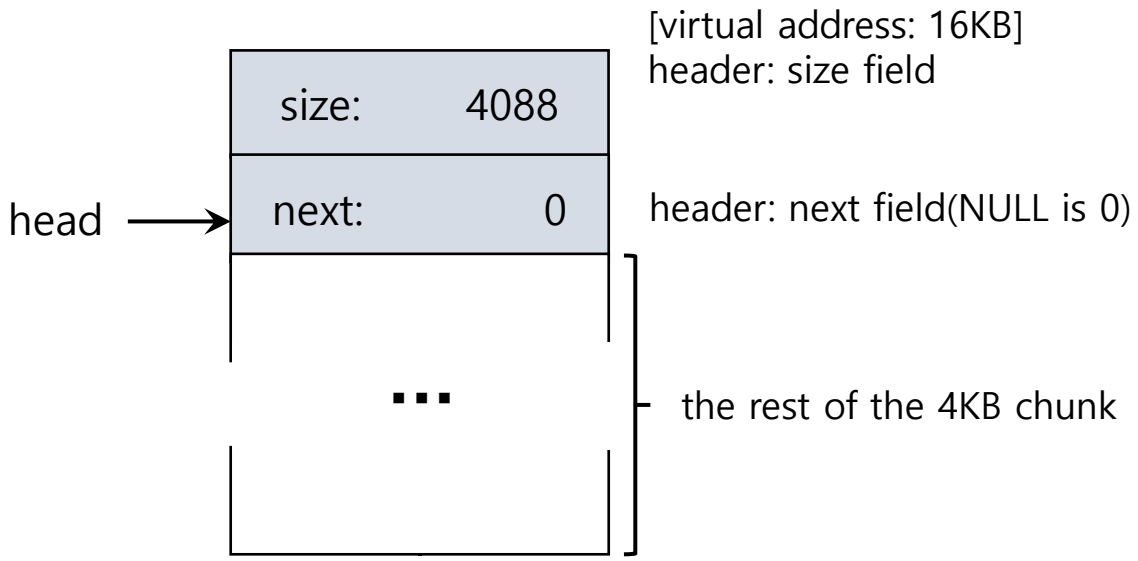
```
typedef struct __node_t {
        int size;
        struct __node_t *next;
} nodet_t;
```

- Öbeğin (heap) inşası ve boş listenin içine konulması
  - Öbeğin `mmap()` sistem çağrısı aracılığıyla oluşturulduğunu varsayalım.

```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                                MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
```

Sadece Bir Boş Parçaya Sahip Öbek

```
// mmap() returns a pointer to a chunk of free space
node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
                                MAP_ANON|MAP_PRIVATE, -1, 0);
head->size = 4096 - sizeof(node_t);
head->next = NULL;
```

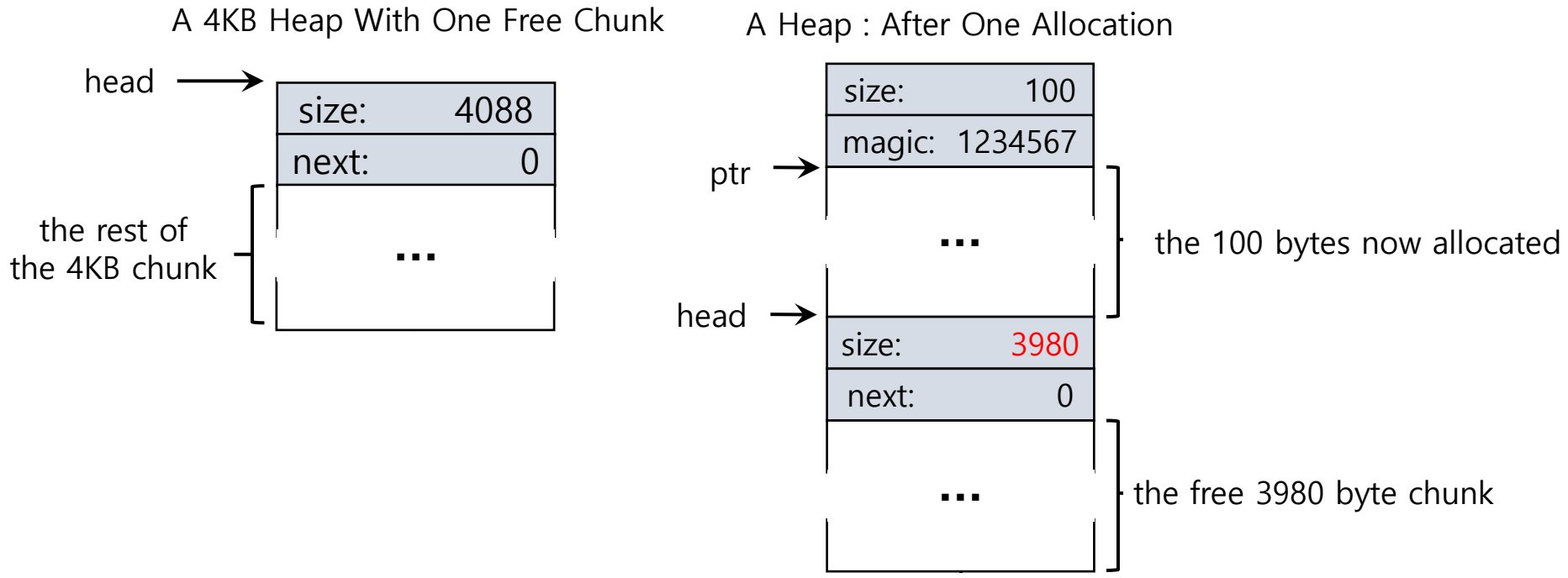


Boş Listesinden Tahsis

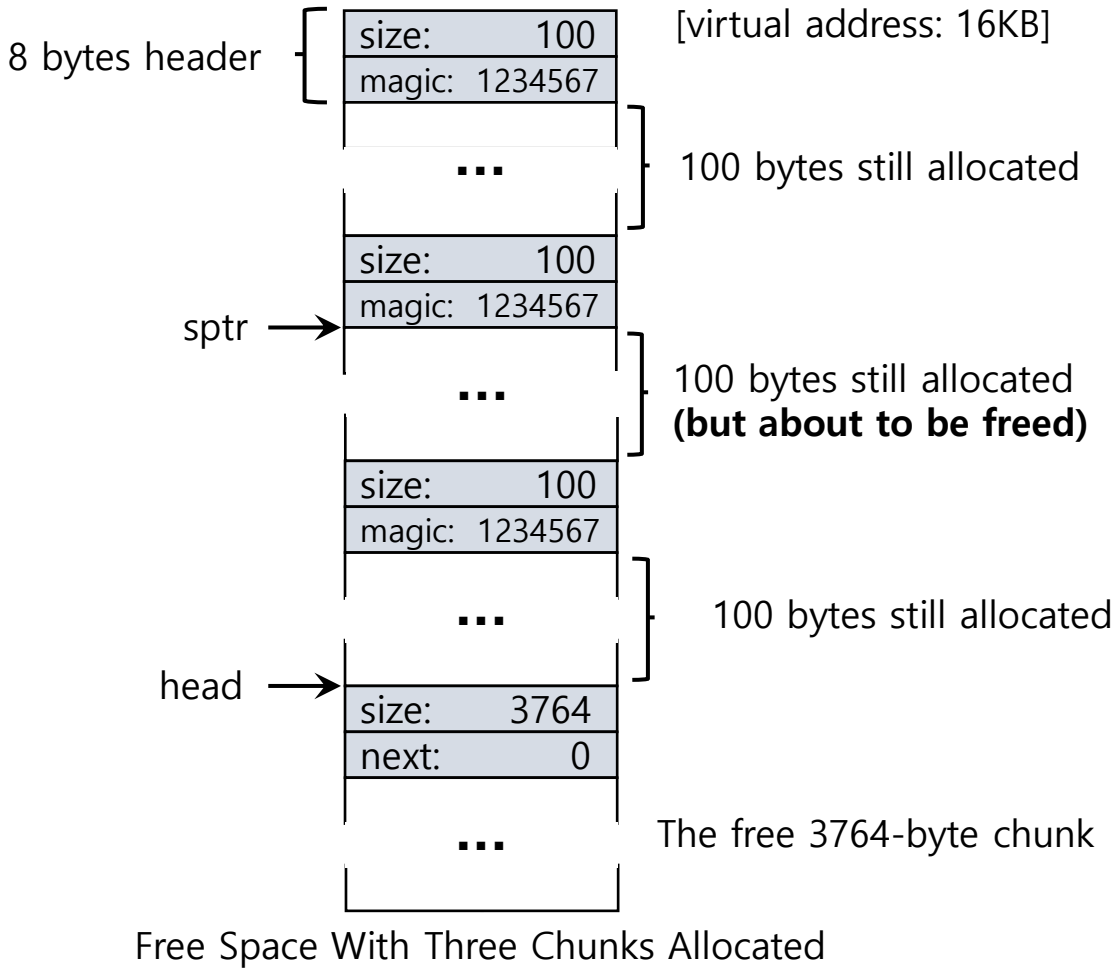
- Belli bir büyüklükte bellek tahsisi isteği gelince, kütüphane önce isteği karşılayacak kadar büyük bir parça bulur.
- Ardından, büyük boş parça ikiye bölünür:
  - Bir parça istek için tahsis edilir.
  - Kalan parça boş listede kalır. Listedeki bu boş parçanın boyutu küçültülür.

Boş Listesinden Tahsis (Devam)

- Örnek: `ptr = malloc(100)` şeklinde 100 baytlık bir istek.
- Mevcut boş parçadan 108 bayt tahsis edilir.
- Boş parça büyüklüğü 3980'e (4088 - 108) indirilir.



Boş Listesinden Tahsise Devam

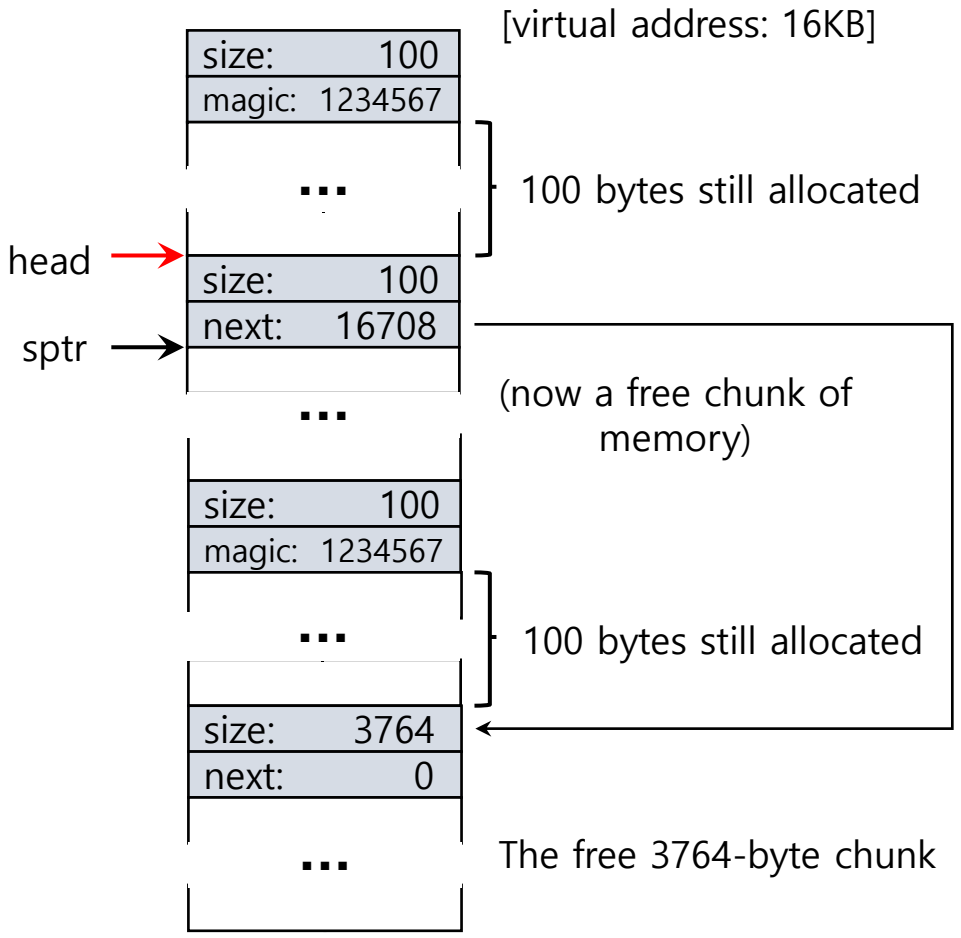


Free Space With Three Chunks Allocated

`free()` çağrılırsa

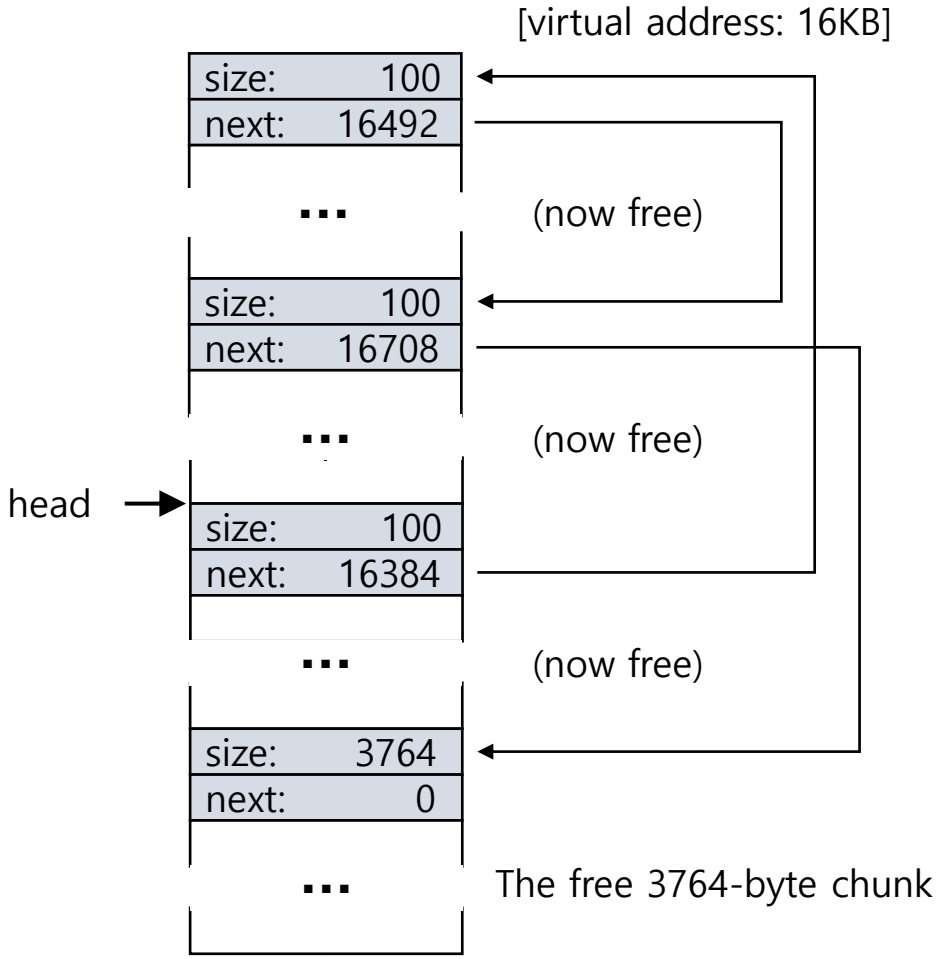
Örnek: `free(sptr)`

- 100 baytlık parça tekrar boş listeye alınır.
- Boş liste bu küçük parça ile başlar.
- Liste başlığı küçük parçayı işaret etmektedir.
- Bu ilk düğümün işaretçisi ise bir sonraki büyük parçayı gösterir.



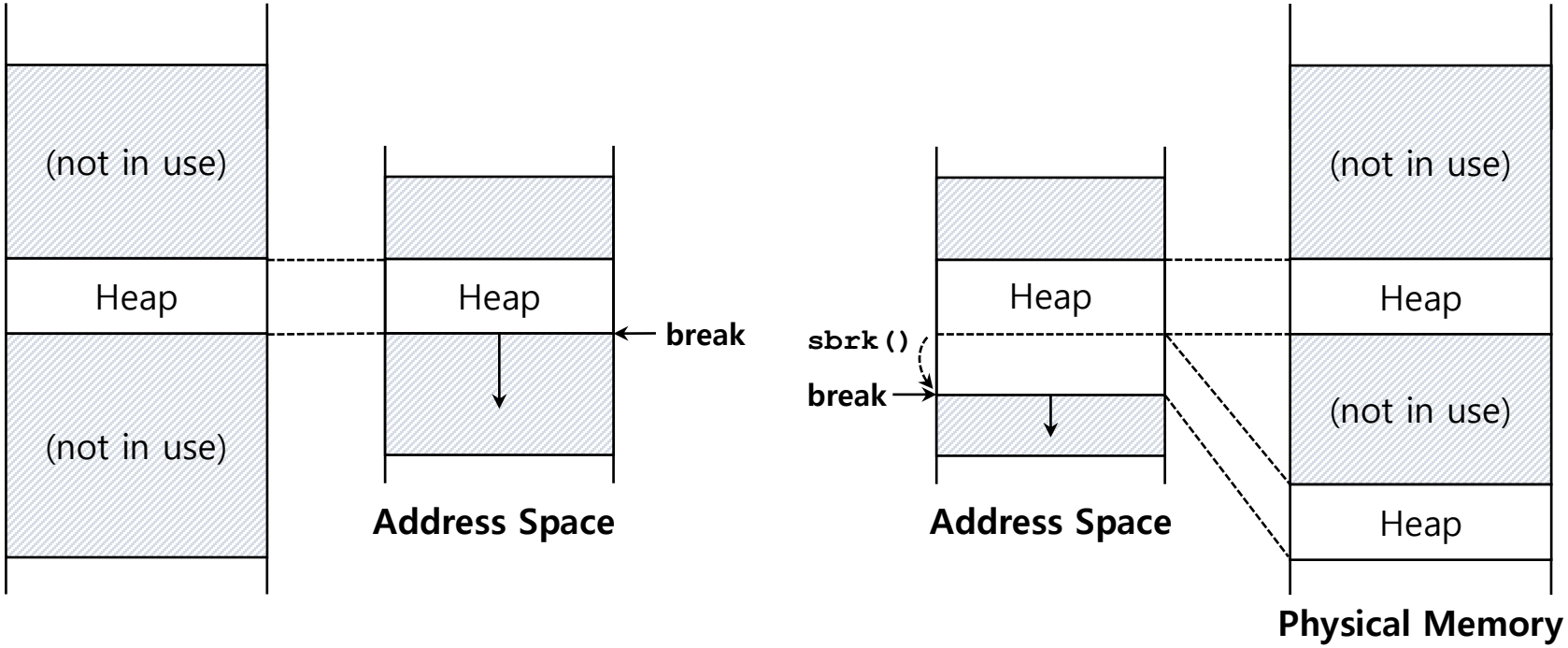
Boş Parçaların Durumu

- Kullanımdaki son iki parçanın da serbest kaldığını varsayalım.
- Dış Parçalanma meydana gelir.
- Listede birleştirme gereklidir.



Öbeğin Büyütülmesi

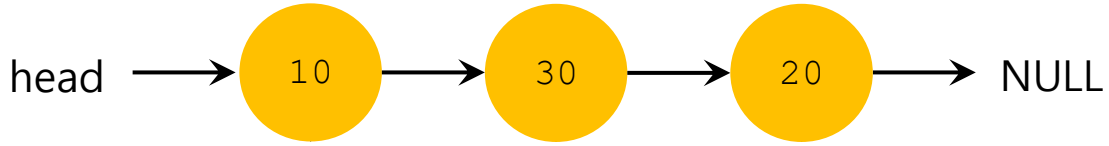
- Çoğu durumda, küçük boyutlu bir öbekle başlanır ve bu öbekte yer bittiğinde işletim sisteminden daha fazla bellek istenir.
- Örnek: Çoğu UNIX sisteminde `sbrk()`, `brk()` sistem çağrıları ile.



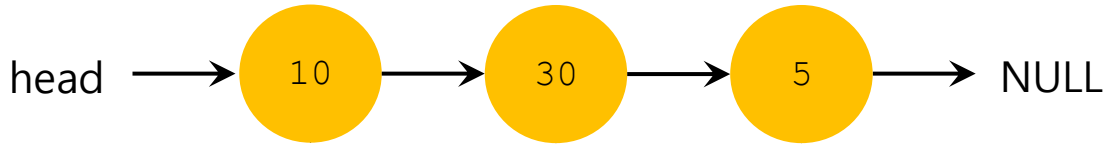
Boş Alanın Yönetilmesi: Temel Stratejiler

- En İyi Uyum (Best Fit): İstek ile aynı büyüklükte veya İstekten daha büyük olan boş parçaları bul ve aday grubundaki parçalardan en küçüğünü seç.
- En Kötü Uyum (Worst Fit): En büyük boş parçayı bul ve isteğe tahsis et, boş kalan parçayı boş listesinde tutmaya devam et.
- İlk Uyum (First Fit): İstek için yeterince büyük olan ilk parçayı kullan.
- Sonraki Uyum (Next Fit): İstek için yeterince büyük olan ilk parçayı bulurken daha önce listede nerede kalınmışsa oradan başla.

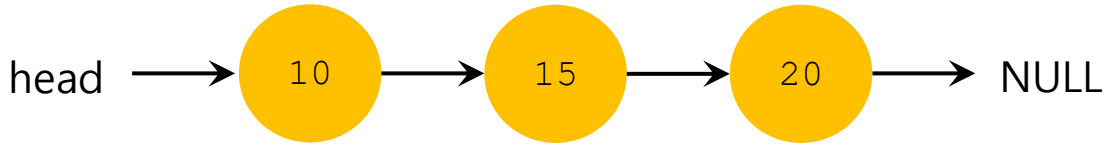
Temel Stratejilere Örnekler



- 15B büyüklüğünde istek gelirse
- En İyi Uyum ile Sonuç:



- En Kötü Uyum ile Sonuç:



Diğer Yaklaşımlar: Ayrılmış Liste

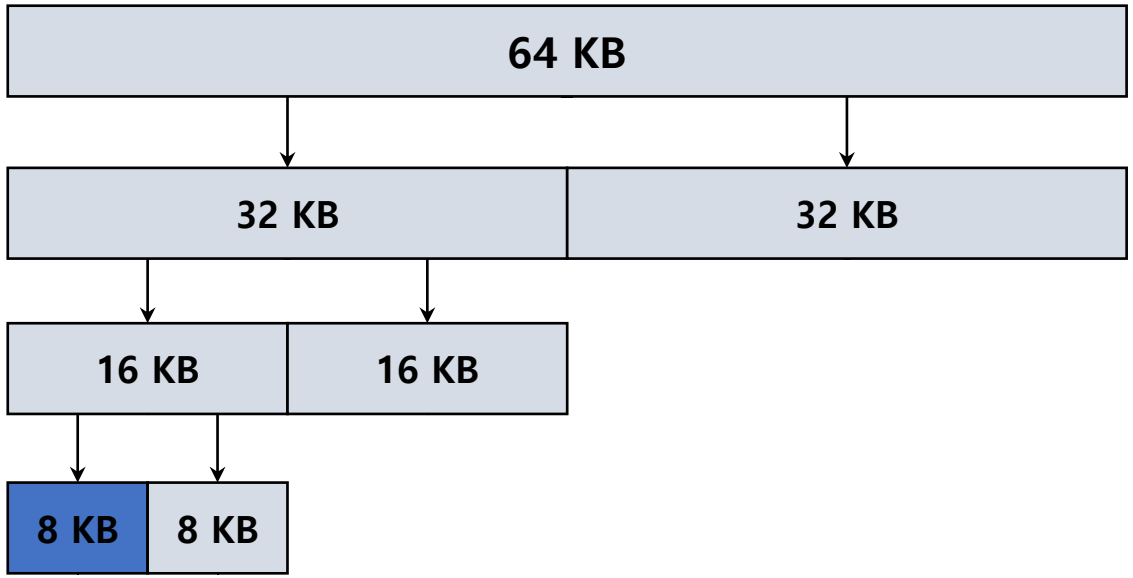
- Ayrılmış Liste (Segregated List): Popüler istek boyutları için boş parçaları ayrı bir listede tutmak.

Sorun: Belirli boyuttaki özel isteklere hizmet eden bellek havuzuna ne kadar bellek ayrılmalıdır?

Çözüm: Slab Allocator: Önbellekte boş alan azaldığında, daha genel bir bellek ayırıcıdan biraz daha bellek iste.

Diğer Yaklaşımlar: Arkadaş Tahsisi

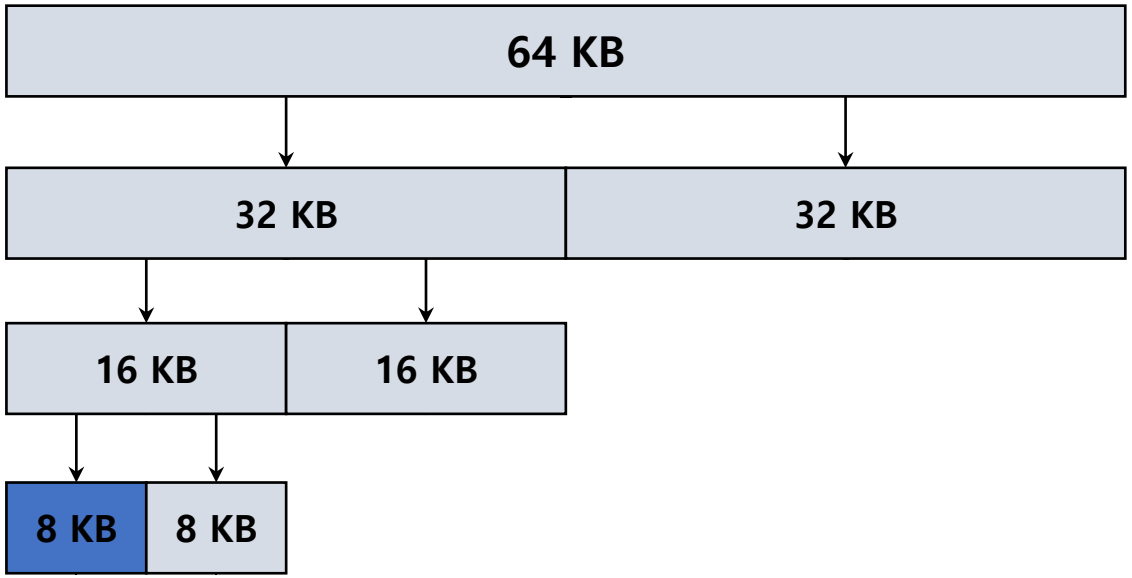
- İkili Arkadaş Tahsisi (Binary Buddy Allocation)
  - Talebi karşılamak için yeterince büyük bir blok bulunana kadar boş alanı ikiye böl.



64KB free space for 7KB request

Diğer Yaklaşımlar: Arkadaş Tahsisi (Devam)

- Arkadaş tahsisi, iç parçalanma (internal fragmentation) problemine sahiptir.
- Öte yandan, birleştirme işlemi kolaylaşır.
- İki bloğu bir sonraki blok seviyesine birleştirmek kolaydır. Çünkü arkadaş iki blok sadece bir bit değerinde farklıdır.



18. Sayfalama: Giriş

Operating System: Three Easy Pieces

Sayfalama Kavramı

- Sayfalama, adres uzayını her biri sayfa (page) adı verilen sabit boyutlu birimlere böler.
- Bölütlemede mantıksal bölütlerin değişken boyutu (kod, öbek, yığın, vb.) sözkonusu iken Sayfalama ile, fiziksel bellek de sayfa çerçevesi (page frame) adı verilen sayfa büyüklüğündeki parçalara bölünür.
- Sanal adresi fiziksel adrese çevirmek için her işlem için ayrı bir sayfa tablosu (page table) gereklidir.

Sayfalamanın Avantajları

Esneklik: Adres uzayı soyutlanmasını etkin bir şekilde destekler.

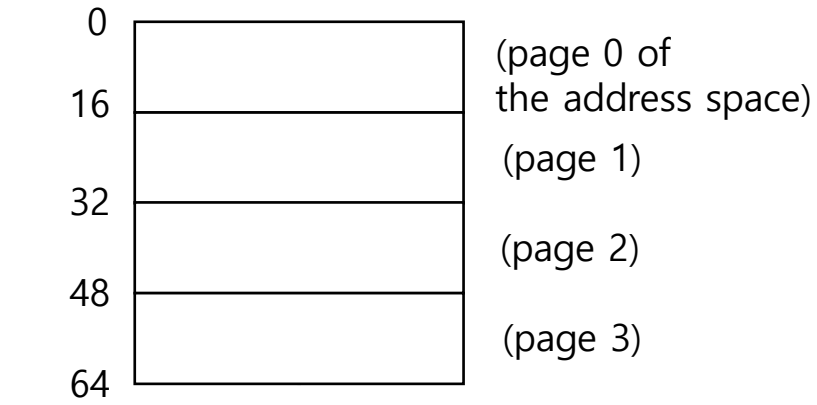
- Yığın ve öbeğin nasıl büyüdüğü ve kullanıldığına dair bir varsayıma gerek yok.

Basitlik: Boş alan yönetimi kolaydır.

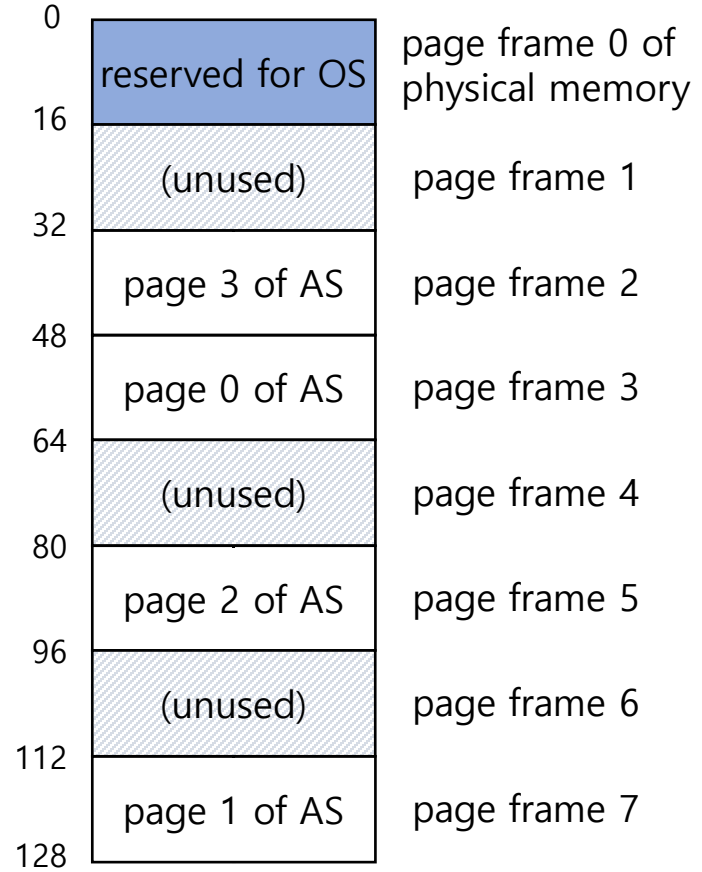
- Adres uzayındaki sayfa ve sayfa çerçevesi aynı boyuttadır.
- Tahsis ve boş liste tutulması kolaylığı.

Örnek

- 128B fiziksel bellek – 16B sayfa çerçeveleri
- 64B adres uzayı – 16B sayfalar



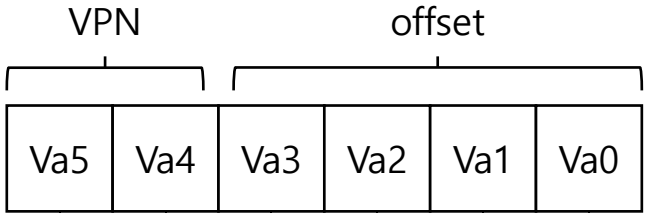
A Simple 64-byte Address Space



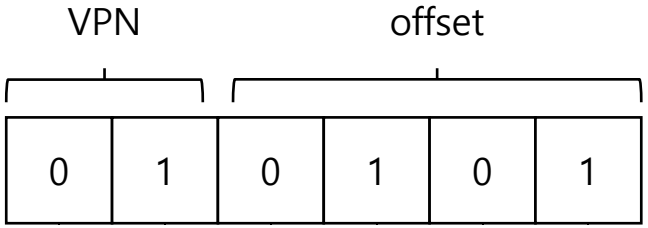
64-Byte Address Space Placed In Physical Memory

Adres Çevrimi

- Sanal adresin iki bileşeni
  - VPN: sanal sayfa numarası (virtual page number)
  - Offset: sayfa içindeki ofset değeri

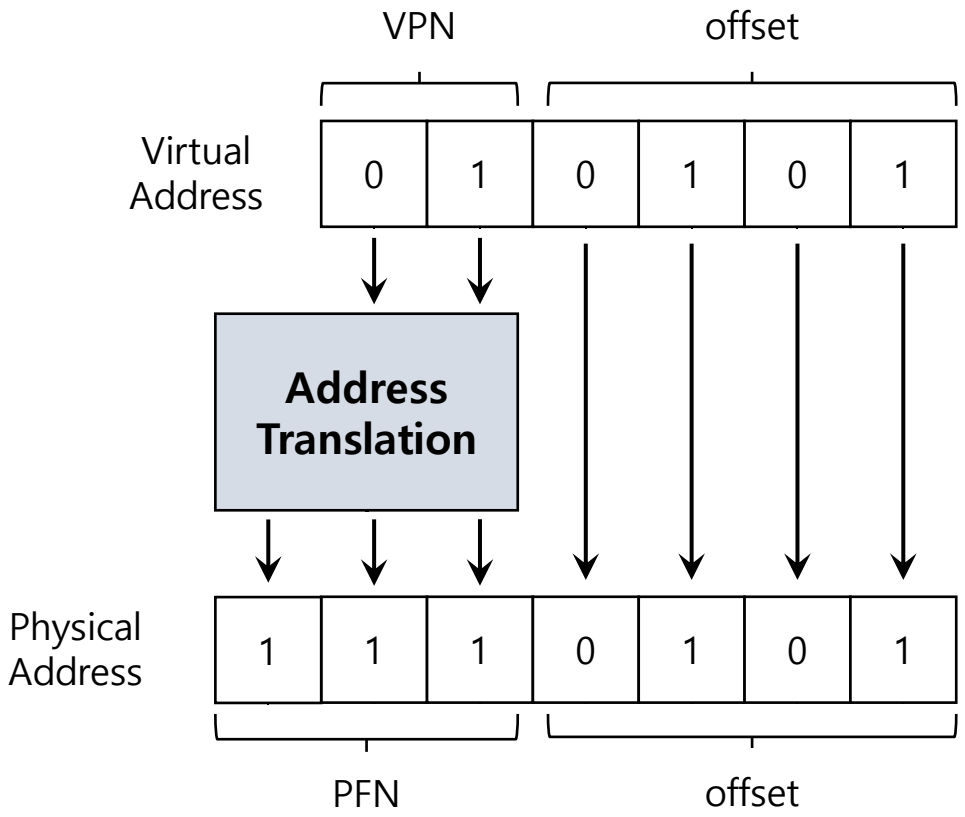


- Örnek: 64B adres uzayında yer alan 21 sanal adresi



Adres Çevrimi (Devam)

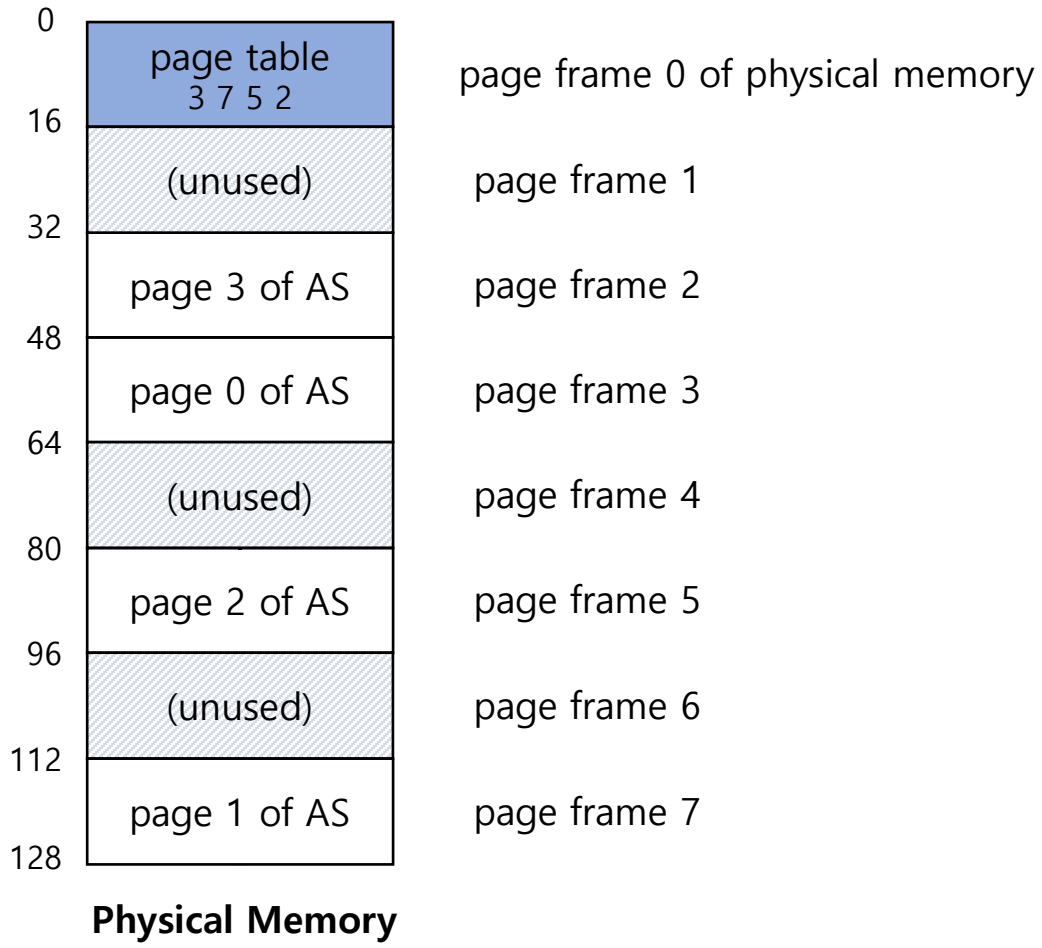
Sanal adres: 21



Sayfa Tabloları Nerededir?

- Bellekte her işlem için ayrı sayfa tabloları tutulur.
- Sayfa tabloları çok büyük olabilir
- 32-bit adres uzayı ve sayfalar 4KB olsun:
  - VPN için 20 bit ayrılır.
  - $4MB = 2^{20}\ girdi\ * 4\ Bayt\ her\ sayfa\ tablosu\ girdisi\ için$

Örnek: Kernel Fiziksel Bellekte Tutulan Sayfa Tablosu



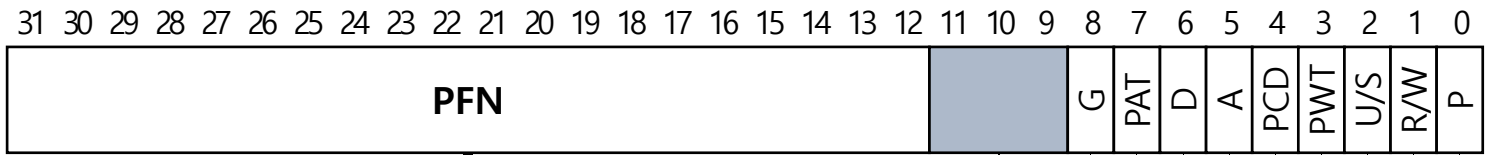
Sayfa Tablosunda Neler Var?

- Sayfa tablosu, sanal adresi fiziksel adrese eşlemek için kullanılan bir veri yapısıdır.
- En basit biçim: doğrusal sayfa tablosu, dizi
- İşletim sistemi, diziyi VPN ile indeksler, böylelikle doğru sayfa tablosu girdisi bulunur.

Bir Sayfa Tablosu Girdisindeki Bayraklar

- Geçerli (Valid) Bit: Belirli bir çevrimin geçerli olup olmadığını gösterir.
- Koruma (Protection) Biti: Sayfanın okunabileceğini, yazılabileceğini veya yürütülebileceğini gösterir.
- Mevcut (Present) Biti: Bu sayfanın fiziksel bellekte mi yoksa diskte mi olduğunu gösterir (takas edilme durumu)
- Kirli (Dirty) Bit: Sayfanın belleğe alındığından beri değiştirilip değiştirilmediğini gösterir.
- Erişilen (Reference) (Accessed) Bit: Bir sayfaya erişilmiş olduğunu gösterir.

Örnek: x86 Sayfa Tablo Girdisi



An x86 Page Table Entry(PTE)

- P: present: mevcut
- R/W: read/write bit: okuma/yazma
- U/S: supervisor: kullanıcı/kernel
- A: accessed bit: erişilen
- D: dirty bit: kirli
- PFN: sayfa çerçeve numarası (page frame number)

Sayfalama: Çok Yavaş

- İstenen sayfa tablo girdisinin (PTE) konumunu bulmak için sayfa tablosunun başlangıç konumuna ihtiyaç vardır.
- Sayfalama, her bellek referansı için, işletim sisteminin fazladan bir bellek referansı gerçekleştirmesini gerektirir.

Sayfalama ile Bellek Erişimi

```
1       // Extract the VPN from the virtual address
2       VPN = (VirtualAddress & VPN_MASK) >> SHIFT
3
4       // Form the address of the page-table entry (PTE)
5       PTEAddr = PTBR + (VPN * sizeof(PTE))
6
7       // Fetch the PTE
8       PTE = AccessMemory(PTEAddr)
9
10      // Check if process can access the page
11      if (PTE.Valid == False)
12              RaiseException(SEGMENTATION_FAULT)
13      else if (CanAccess(PTE.ProtectBits) == False)
14              RaiseException(PROTECTION_FAULT)
15      else
16              // Access is OK: form physical address and fetch it
17              offset = VirtualAddress & OFFSET_MASK
18              PhysAddr = (PTE.PFN << PFN_SHIFT) | offset
19              Register = AccessMemory(PhysAddr)
```

Bellek İzi

- Örnek: Basit Bir Bellek Erişimi

```
int array[1000];
...
for (i = 0; i < 1000; i++)
        array[i] = 0;
```

- Derle ve Çalıştır

```
prompt> gcc –o array array.c –Wall –o
prompt>./array
```

- Sonuçta Ortaya Çıkan Assembly Kodu

```
0x1024 movl $0x0,(%edi,%eax,4)
0x1028 incl %eax
0x102c cmpl $0x03e8,%eax
0x1030 jne 0x1024
```

Sanal (Ve Fiziksel) Bellek İzi