

Bölüm 14: Çoklu Ortam Sistemleri

İşletim Sistemleri

Çoklu ortamın özellikleri ve veri hızları

Ses sayısallaştırma ve Nyquist

Psikoakustik ve ses sıkıştırma

Video: tarama, renk uzayı, 4:2:0

JPEG: DCT, niceleme, zikzak

MPEG: I/P/B kareleri, hareket dengeleme

H.264, HEVC, AV1 ve donanım kod çözücüler

Gerçek zamanlı çizelgeleme: RMS ve EDF

İtme/çekme sunucular, NVOD, VCR

Dosya yerleşimi, Zipf ve organ borusu

Önbellekleme ve disk çizelgeleme

HLS/DASH, CDN ve bit hızı merdiveni

Öğrenme Çıktıları

Özellikler ve hesap

Sıkıştırmasız ses ve video veri hızlarını hesaplamak; gerçek zaman, titreşim ve QoS kavramlarını açıklamak.

Ses

Örnekleme, niceleme ve Nyquist ölçütünü uygulamak; psikoakustik maskelemenin sıkıştırmadaki rolünü anlatmak.

Görüntü ve video

YCbCr ve 4:2:0, JPEG boru hattı (DCT, niceleme, zikzak, Huffman) ve I/P/B kareli MPEG kodlamayı izlemek.

Süreç çizelgeleme

Periyodik görevlerde RMS ve EDF zaman çizelgesini çıkarmak; kullanım sınırıyla çizelgelenebilirliği sınamak.

Dosya sistemi

İtme/çekme sunucuları, NVOD, blok boyutu ve indis seçimleri, Zipf ve organ borusu yerleşimini değerlendirmek.

Disk ve akış

Statik turlar ile SCAN-EDF'yi; HLS/DASH, CDN ve uyarlanabilir bit hızı merdivenini açıklamak.

Çoklu Ortam Nedir?

Metin, grafik, ses, video ve animasyonun birlikte sunulması. İşletim sistemi açısından asıl zorluk **sürekli ortam** (ses ve video): veri, belirli bir hızda ve zamanında akmalıdır.

Yüksek veri hızı

Sıkıştırmasız 1080p $\approx 1,5$ Gbit/s; sıkıştırma zorunlu. 2 saatlik 4K film 16 Mbit/s ile bile ≈ 14 GB.

Gerçek zaman

Kareler sabit aralıkla (25 fps $\rightarrow$ 40 ms) gösterilmeli; geç gelen veri işe yaramaz. Titreşim tamponla emilir.

Hizmet kalitesi (QoS)

Bant genişliği, gecikme, titreşim ve kayıp için garanti ya da öngörülebilirlik; kaynak ayırma ve kabul denetimi.

Senkronizasyon

Dudak senkronizasyonu: ses ≈ 45 ms önde ya da ≈ 125 ms geride kalırsa fark edilir (ITU-R BT.1359).

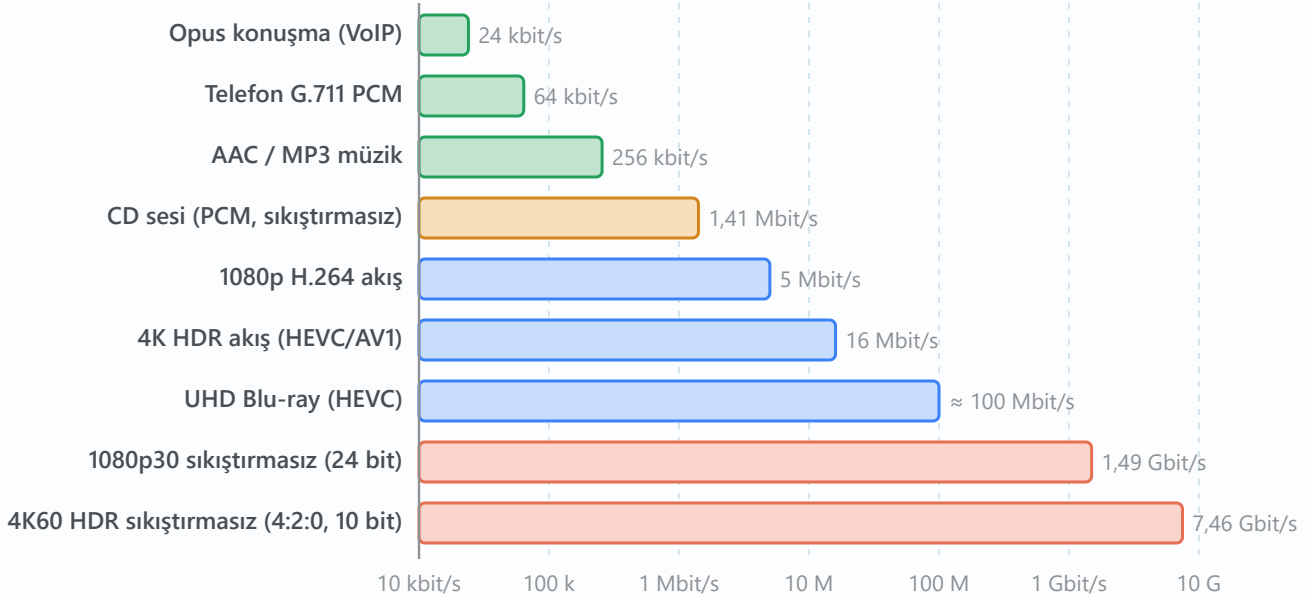
Etkileşim

Duraklat, geri/ileri sar, atla (VCR işlevleri); sunucu ve dosya sistemi tasarımını etkiler.

Uyumluluk

Farklı kodex, kapsayıcı (MP4, MKV), ekran (HDR/SDR) ve ağ; oynatıcı uygun sürümü seçer.

Veri Hızları



Akış hızları 2026 için tipik değerlerdir (Netflix HD için ≈ 5, 4K için ≈ 15 Mbit/s önerir). Kayıpsız FLAC, CD kalitesinde ≈ 0,7–1 Mbit/s; 24 bit/96 kHz ≈ 2–3 Mbit/s. Arada ≈ 3 basamak fark: sıkıştırma olmadan video akışı yoktur.

Sıkıştırmasız Veri Hızlarını Hesaplama

CD sesi

44.100 örnek/s

×

16 bit

×

2 kanal

=

1.411.200 bit/s $\approx$ 1,41 Mbit/s1 saat $\approx$ 635 MB

?

?

?

sıkıştırma oranı?

CD sesi: 44,1 kHz örnekleme $\times$ 16 bit $\times$ 2 kanal = **1,41 Mbit/s**; bir saatlik albüm $\approx$ 635 MB.

Sıkıştırmasız Veri Hızlarını Hesaplama

CD sesi

44.100 örnek/s

×

16 bit

×

2 kanal

=

1.411.200 bit/s $\approx$ 1,41 Mbit/s1 saat $\approx$ 635 MB

1080p30, RGB

1920 × 1080 piksel

×

24 bit

×

30 kare/s

=

1.492.992.000 bit/s $\approx$ 1,49 Gbit/s2 saat $\approx$ 1,34 TB

?

?

sıkıştırma oranı?

1080p, piksel başına 24 bit RGB, 30 kare/s: **1,49 Gbit/s**. İki saatlik film sıkıştırmasız 1,34 TB tutar.

Sıkıştırmasız Veri Hızlarını Hesaplama

CD sesi

44.100 örnek/s

×

16 bit

×

2 kanal

=

1.411.200 bit/s $\approx$ 1,41 Mbit/s1 saat $\approx$ 635 MB

1080p30, RGB

1920 × 1080 piksel

×

24 bit

×

30 kare/s

=

1.492.992.000 bit/s $\approx$ 1,49 Gbit/s2 saat $\approx$ 1,34 TB

1080p30, 4:2:0

1920 × 1080 piksel

×

12 bit (ort.)

×

30 kare/s

=

746.496.000 bit/s $\approx$ 746 Mbit/s

renk alt örnekleme: 1/2

?

sıkıştırma oranı?

Renk bilgisini yarı yarıya azaltan 4:2:0 alt örnekleme piksel başına ortalama 12 bit verir: 746 Mbit/s. Bu henüz kayıplı sıkıştırma değildir.

Sıkıştırmasız Veri Hızlarını Hesaplama

CD sesi

44.100 örnek/s

×

16 bit

×

2 kanal

=

1.411.200 bit/s $\approx$ 1,41 Mbit/s1 saat $\approx$ 635 MB

1080p30, RGB

1920 × 1080 piksel

×

24 bit

×

30 kare/s

=

1.492.992.000 bit/s $\approx$ 1,49 Gbit/s2 saat $\approx$ 1,34 TB

1080p30, 4:2:0

1920 × 1080 piksel

×

12 bit (ort.)

×

30 kare/s

=

746.496.000 bit/s $\approx$ 746 Mbit/s

renk alt örnekleme: 1/2

4K60 HDR, 4:2:0

3840 × 2160 piksel

×

15 bit (10 bit, 4:2:0)

×

60 kare/s

=

7.464.960.000 bit/s $\approx$ 7,46 Gbit/s2 saat $\approx$ 6,7 TB

sıkıştırma oranı?

4K, 60 kare/s, 10 bit HDR, 4:2:0 ($10 + 10/4 + 10/4 = 15$ bit/piksel): **7,46 Gbit/s**; tek bir 10 GbE bağlantıyı neredeyse doldurur.

Sıkıştırmasız Veri Hızlarını Hesaplama

CD sesi

44.100 örnek/s

×

16 bit

×

2 kanal

=

1.411.200 bit/s $\approx$ 1,41 Mbit/s1 saat $\approx$ 635 MB

1080p30, RGB

1920 × 1080 piksel

×

24 bit

×

30 kare/s

=

1.492.992.000 bit/s $\approx$ 1,49 Gbit/s2 saat $\approx$ 1,34 TB

1080p30, 4:2:0

1920 × 1080 piksel

×

12 bit (ort.)

×

30 kare/s

=

746.496.000 bit/s $\approx$ 746 Mbit/s

renk alt örnekleme: 1/2

4K60 HDR, 4:2:0

3840 × 2160 piksel

×

15 bit (10 bit, 4:2:0)

×

60 kare/s

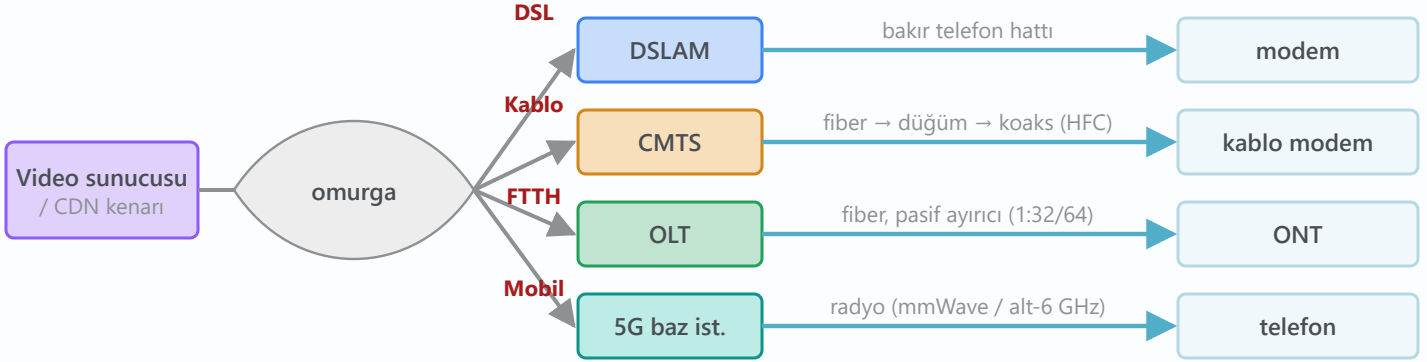
=

7.464.960.000 bit/s $\approx$ 7,46 Gbit/s2 saat $\approx$ 6,7 TB**Sıkıştırma oranı: 4K60 HDR 7,46 Gbit/s $\rightarrow$ 16 Mbit/s akış $\approx$ 467 : 1**1080p30 (4:2:0) 746 Mbit/s $\rightarrow$ 5 Mbit/s $\approx$ 149 : 1 · CD 1,41 Mbit/s $\rightarrow$ AAC 256 kbit/s $\approx$ 5,5 : 1

Sonuç: modern kodekler videoyu 150–500 kat küçültür; ses için gereken oran çok daha düşüktür. İşletim sistemi yine de sürekli, zamanında veri akışı sağlamalıdır.

GİRİŞ

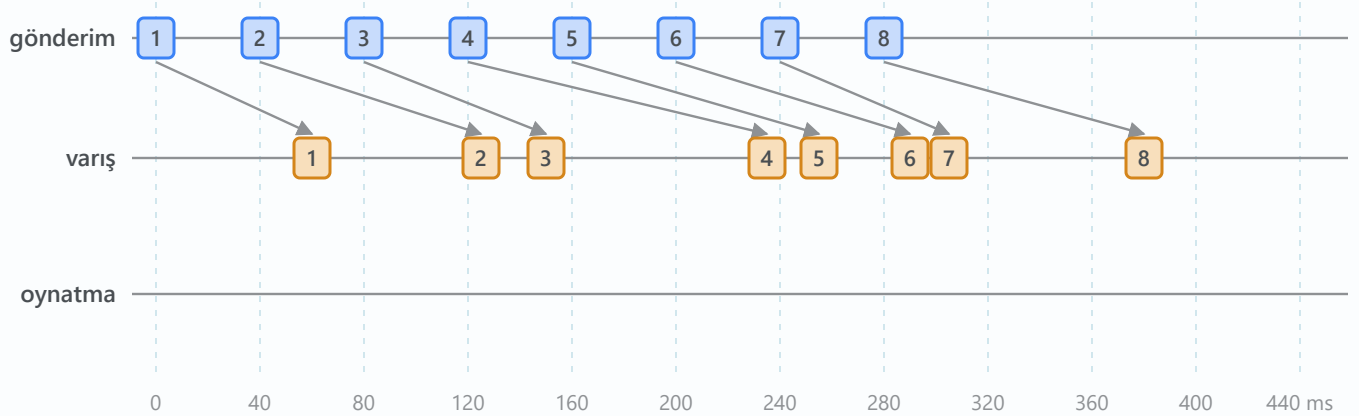
Video Dağıtım Ağları



Erişim	Erken nesil değer	Güncel tipik değer (2026)
ADSL2+	—	≤ 24 Mbit/s indirme, ≤ 1–3 Mbit/s yükleme (asimetrik)
VDSL2 (vektörleme)	VDSL: 52 / 16 Mbit/s	100 Mbit/s; 35b profili kısa hatta ≈ 250–300 Mbit/s
Kablo (DOCSIS 3.1 / 4.0)	10–100 Mbit/s	1–10 Gbit/s indirme (paylaşımlı ortam)
FTTH (GPON / XGS-PON)	—	2,5 / 1,25 Gbit/s ve 10 / 10 Gbit/s, ayırıcıdaki kullanıcılar paylaşır

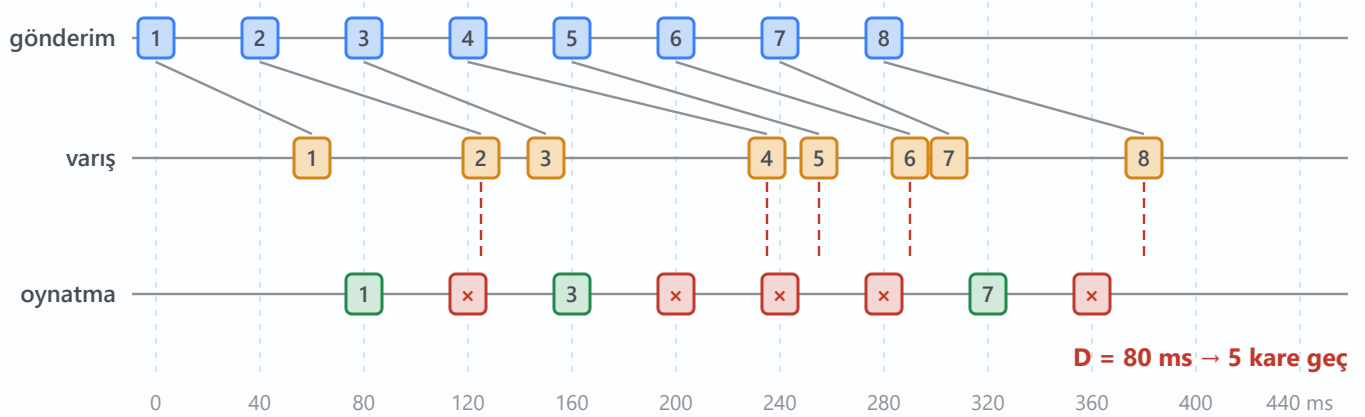
SDSL ve HDSL simetrik eski DSL türleridir. Erişim hızından çok sunucu ile kullanıcı arasındaki yol (CDN) ve titreşim belirleyicidir.

Gerçek Zaman ve Titreşim



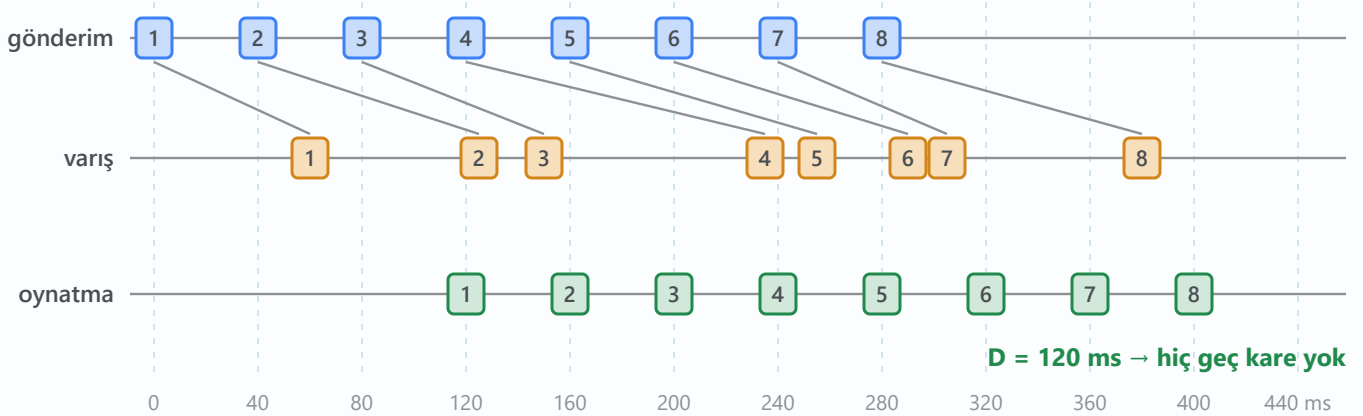
Kaynak kareleri 40 ms aralıklarla (25 fps) gönderir; ağ gecikmesi 60 ile 115 ms arasında değişir. Varış aralıkları düzensizdir: **titreşim**.

Gerçek Zaman ve Titreşim



Oynatma, gönderimden $D = 80$ ms sonra planlanırsa gecikmesi 80 ms'yi aşan 5 kare (2, 4, 5, 6, 8) zamanında yetişmez: takılma ya da atlama.

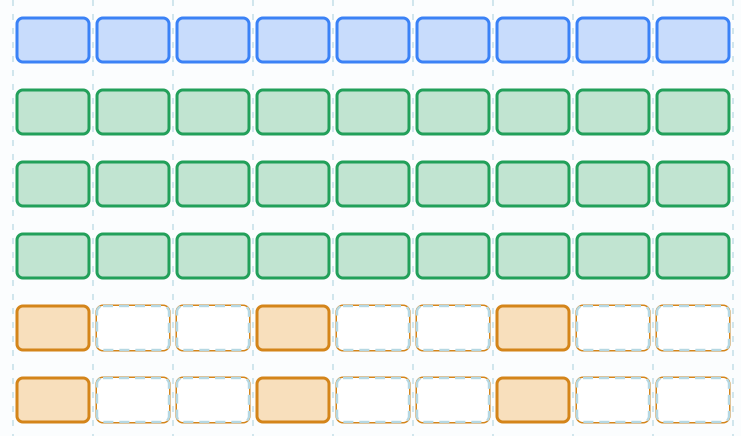
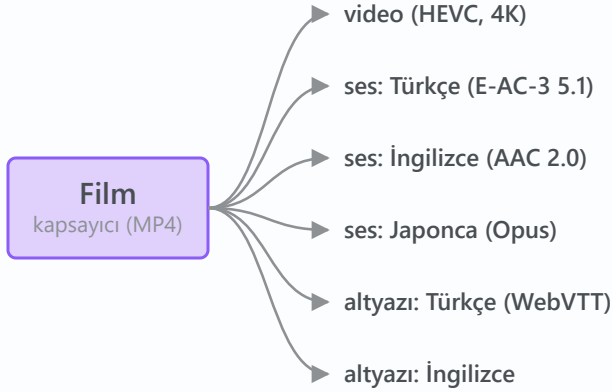
Gerçek Zaman ve Titreşim



$D = 120$ ms (en büyük gecikme 115 ms) seçilirse tüm kareler yetişir. Bedel: 120 ms ek gecikme ve en çok 3 karelik tampon. Akış hizmetleri saniyeler, görüntülü görüşme onlarca ms tampon kullanır.

GİRİŞ

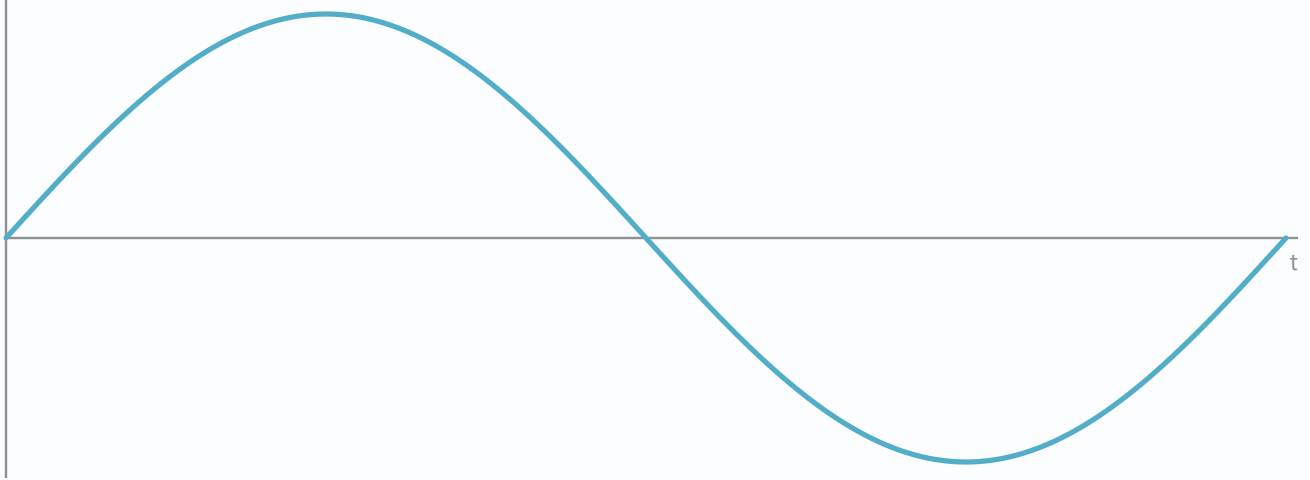
Bir Film Birkaç Dosyadan Oluşabilir



ortak zaman çizelgesi: her parça bir sunum zaman damgası (PTS) taşır

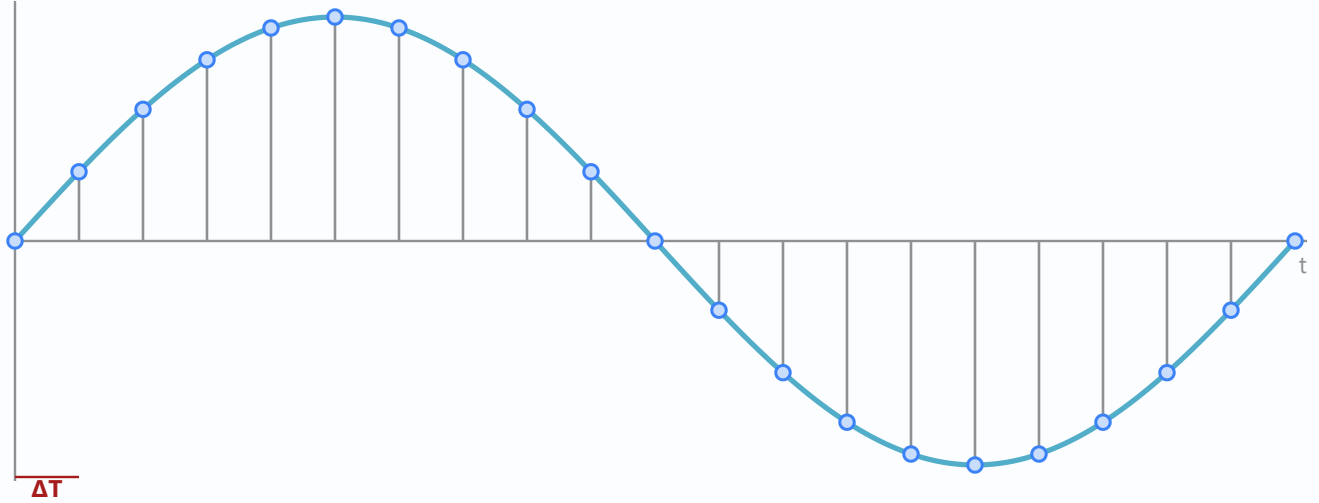
Oynatıcı yalnız seçilen dilin ses ve altıyazı izini okur. Eski VOD sistemleri izleri ayrı dosyalarda tutardı; bugün kapsayıcı içindeki izler ya da DASH/HLS'te ayrı "uyarlama kümeleri" olarak ayrı ayrı indirilen bölütler kullanılır.

Ses Sayısallaştırma: Örnekleme ve Niceleme



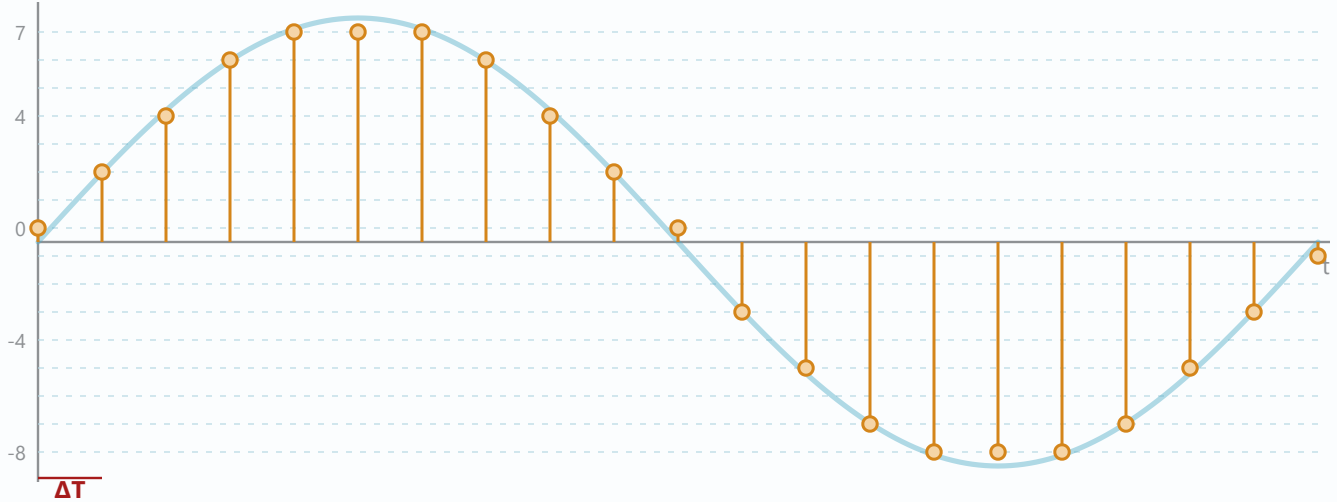
(a) Analog ses: sürekli zamanda sürekli genlikli bir dalga (burada tek periyotluk sinüs).

Ses Sayısallaştırma: Örneklem ve Niceleme



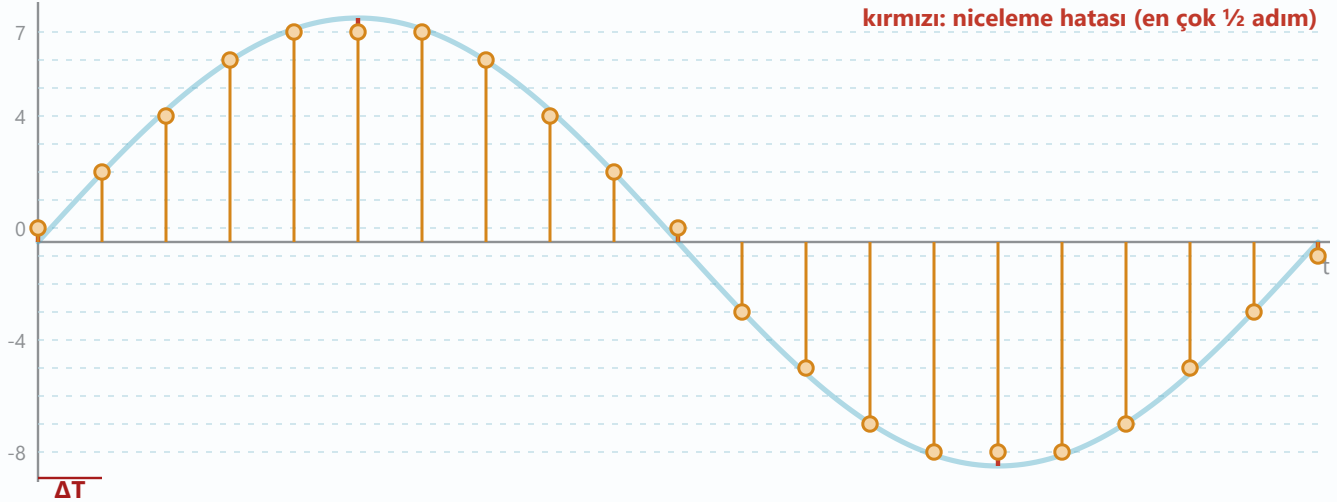
(b) **Örneklem:** her ΔT saniyede bir genlik ölçülür. Örneklem hızı $f_s = 1/\Delta T$ (CD: 44.100 örnek/s).

Ses Sayısallaştırma: Örnekleme ve Niceleme



(c) **Niceleme:** her örnek en yakın düzeye yuvarlanır. $4 \text{ bit} \rightarrow 2^4 = 16$ düzey ($-8 \dots 7$). CD 16 bit, yani 65.536 düzey kullanır.

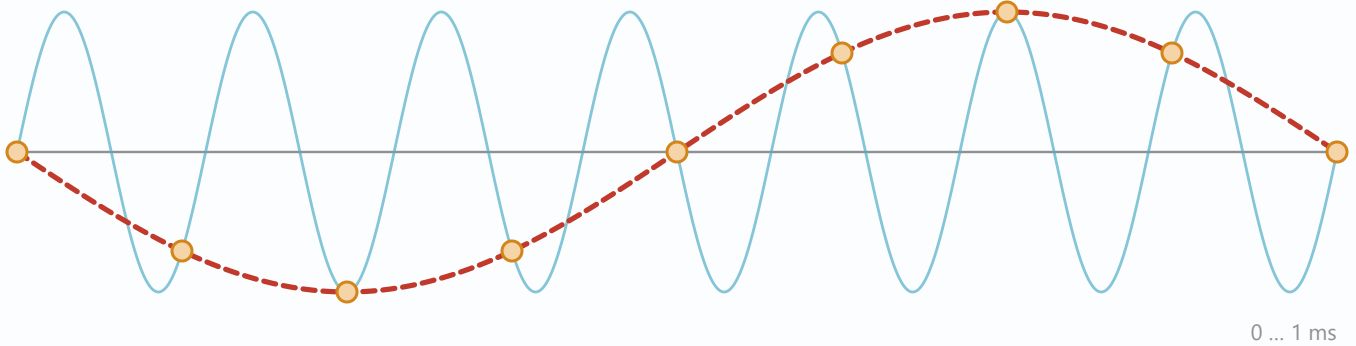
Ses Sayısallaştırma: Örnekleme ve Niceleme



(d) Gerçek değer ile kodlanan düzey arasındaki fark **niceleme gürültüsüdür**. Her ek bit gürültüyü yarıya indirir: $SNR \approx 6,02 \cdot n + 1,76$ dB (4 bit ≈ 26 dB, 16 bit ≈ 98 dB).

Nyquist Ölçütü ve Örtüşme

mavi: 7 kHz giriş · turuncu: 8 kHz örnekler · kırmızı: aynı örneklerden geri çatılan 1 kHz



Nyquist–Shannon

En yüksek frekansı $f_{\max}$ olan sinyal $f_s > 2 \cdot f_{\max}$ ile örneklenirse tam geri çatılır. Aksi hâlde yüksek frekanslar alçak frekans gibi görünür (örtüşme); kodlayıcı önce alçak geçiren süzgeç uygular.

Tipik hızlar

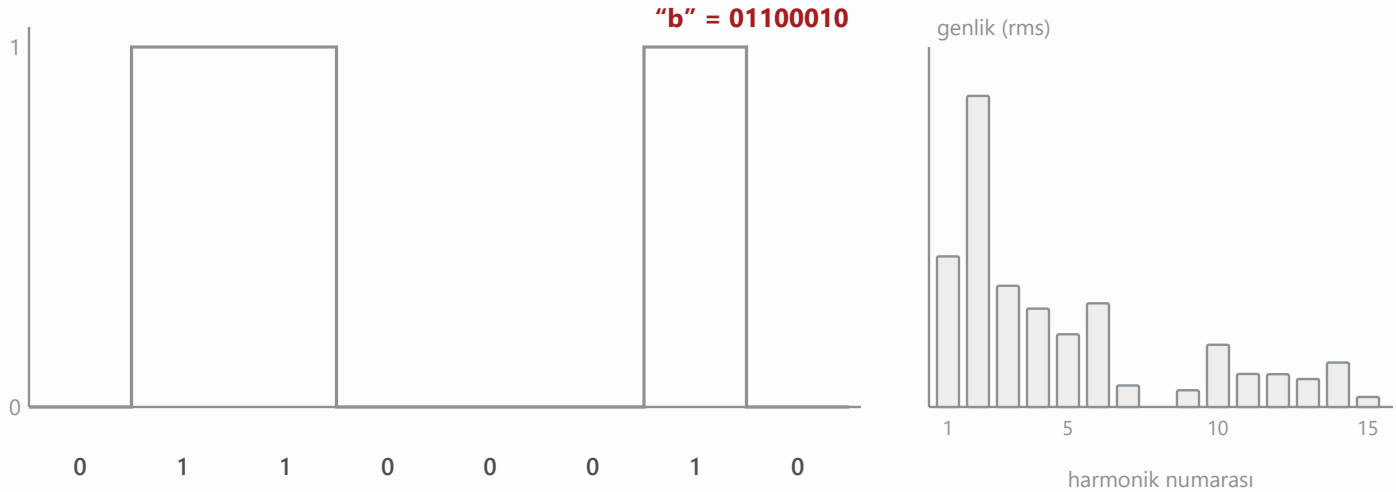
Telefon: 8 kHz ($\leq 3,4$ kHz konuşma). CD: 44,1 kHz (insan kulağı ≈ 20 kHz). Video/yayın: 48 kHz. Stüdyo: 96–192 kHz. Bit derinliği 16 bit ≈ 98 dB, 24 bit ≈ 146 dB dinamik aralık.

Ses Kodlama Türleri

Biçim	Tür	Tipik bit hızı	Kullanım
PCM (WAV, CD)	sıkıştırmasız, doğrusal	1,41 Mbit/s (44,1 kHz/16 bit/stereo)	kayıt, düzenleme, CD
MP3 (MPEG-1 Audio Layer III)	kayıplı, psikoakustik	128–320 kbit/s	eski müzik arşivleri
AAC (LC, HE-AAC)	kayıplı, MDCT	64–256 kbit/s	Apple Music, YouTube, HLS/DASH, yayın
Opus (IETF RFC 6716)	kayıplı, düşük gecikme	6–510 kbit/s (konuşma $\approx$ 16–32)	WebRTC, Discord, VoIP, YouTube
E-AC-3 / Dolby Atmos	kayıplı, çok kanallı / nesne	384–768 kbit/s	film, 5.1/7.1, uzamsal ses
FLAC, ALAC	kayıpsız (doğrusal öngörü)	$\approx$ 0,7–1 Mbit/s (CD); hi-res 2–3	arşiv, kayıpsız akış

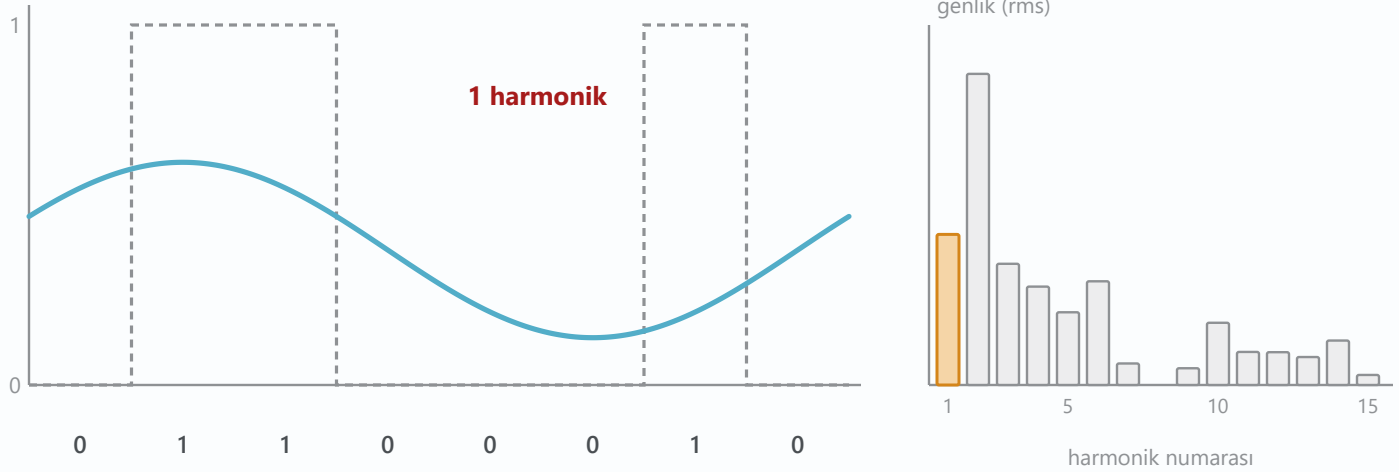
FLAC için 1–5 Mbit/s yalnız yüksek çözünürlüklü (24 bit/96–192 kHz) dosyalarda görülür; CD kalitesinde FLAC, PCM'nin $\approx$ %50–70'i kadardır. Opus, 2012'den beri konuşma ve müzik için tek kodektir ve gerçek zamanlı uygulamaların varsayılanıdır.

Fourier Ayırışımı ve Ardışık Yaklaşımlar



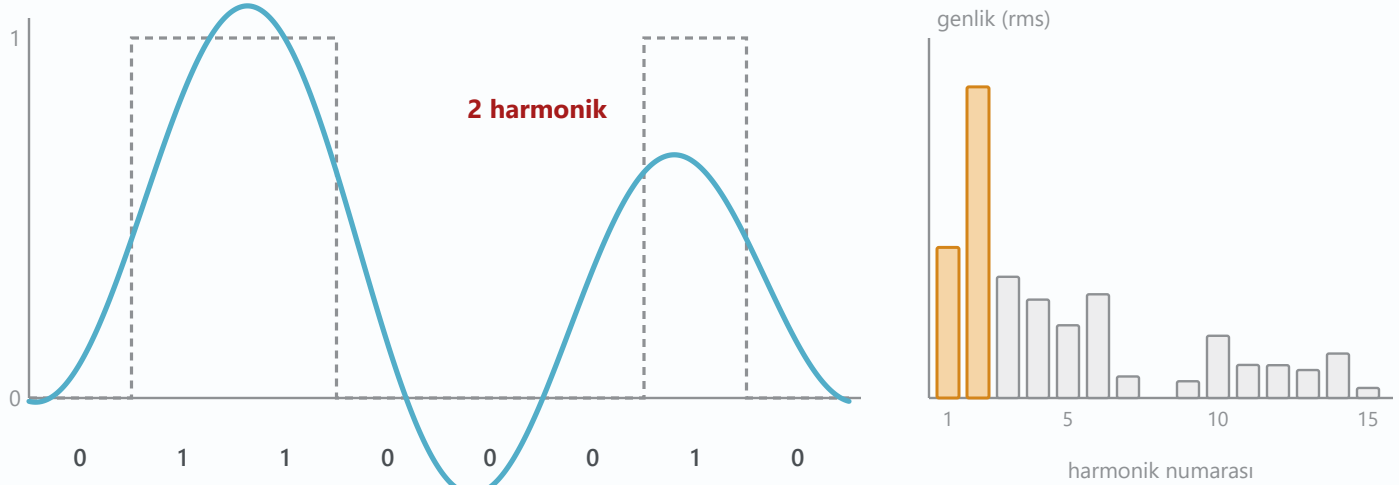
8 bitlik ASCII "b" (01100010) kare dalgası. Her periyodik sinyal, sabit + sinüs/kosinüs harmonikleri toplamı olarak yazılabilir (sağda genlikler).

Fourier Ayrışımı ve Ardışık Yaklaşımlar



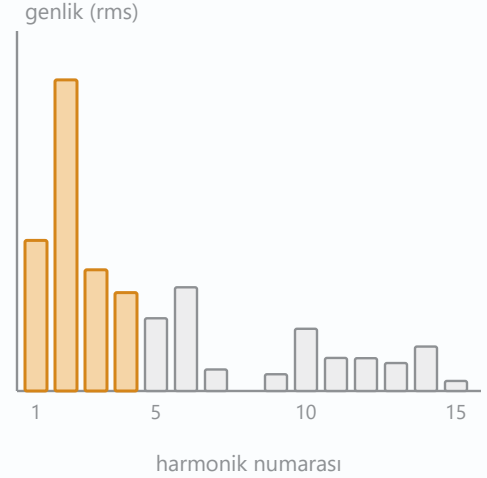
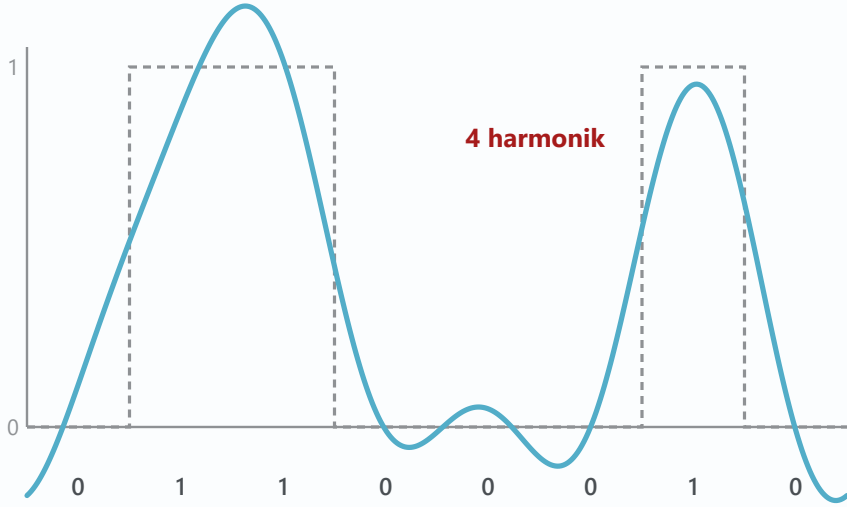
Yalnız 1. harmonik: sinyalin kaba biçimi. İletim kanalı ya da kodlayıcı yüksek frekansları atarsa elde kalan budur.

Fourier Ayrışımı ve Ardışık Yaklaşımlar



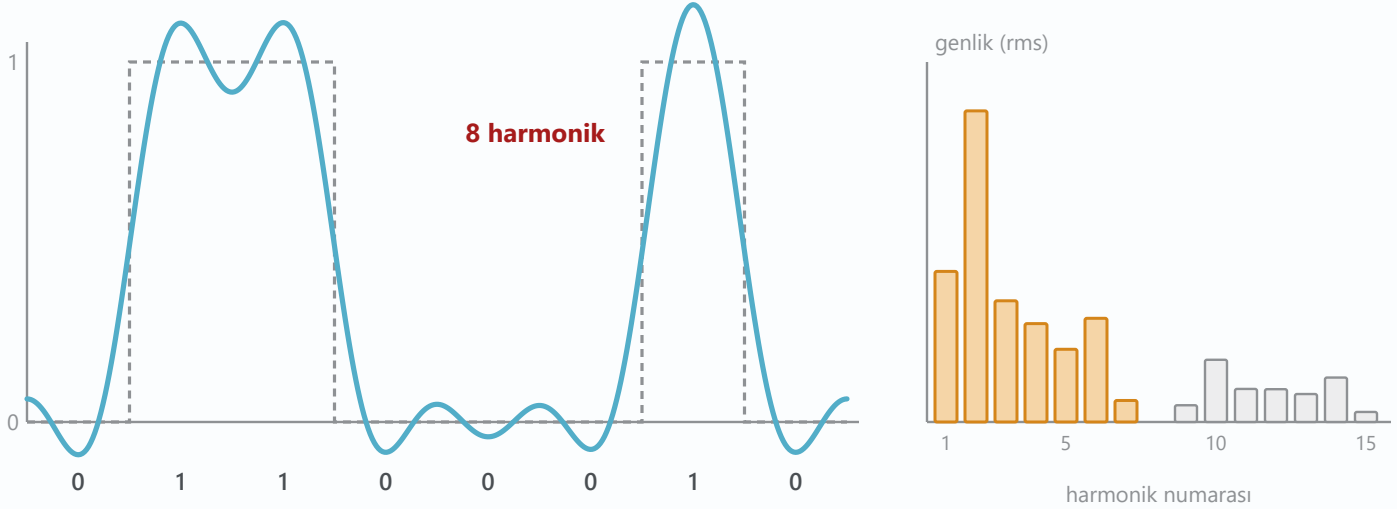
2 harmonik: geçişler belirginleşiyor. Genliklerin çoğu ilk birkaç harmonikte toplanmıştır.

Fourier Ayırışımı ve Ardışık Yaklaşımlar



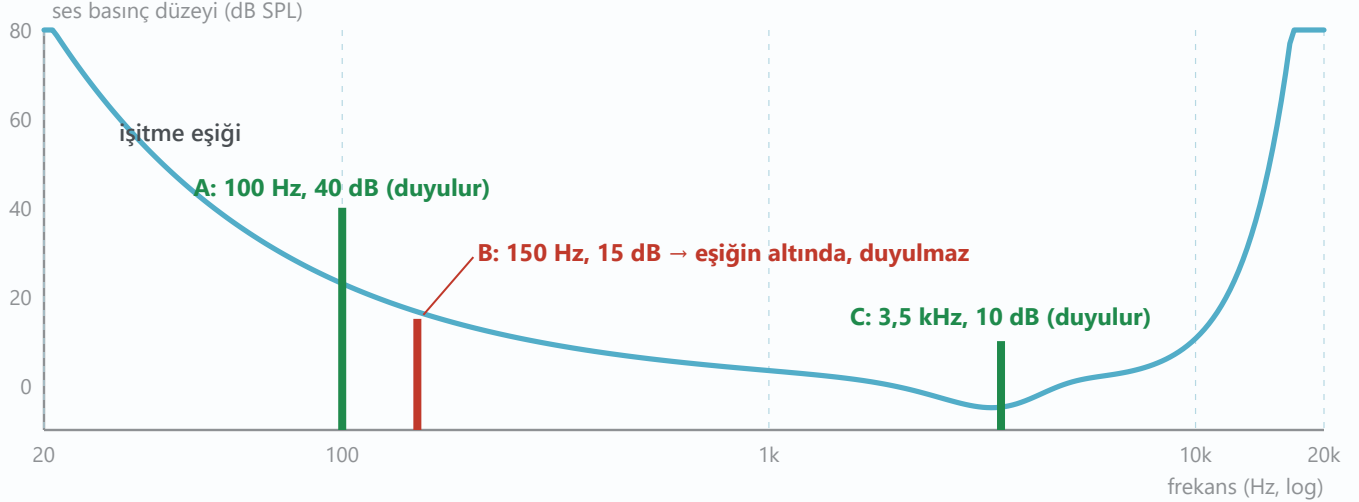
4 harmonik: bitler artık ayırt edilebilir.

Fourier Ayrışımı ve Ardışık Yaklaşımlar



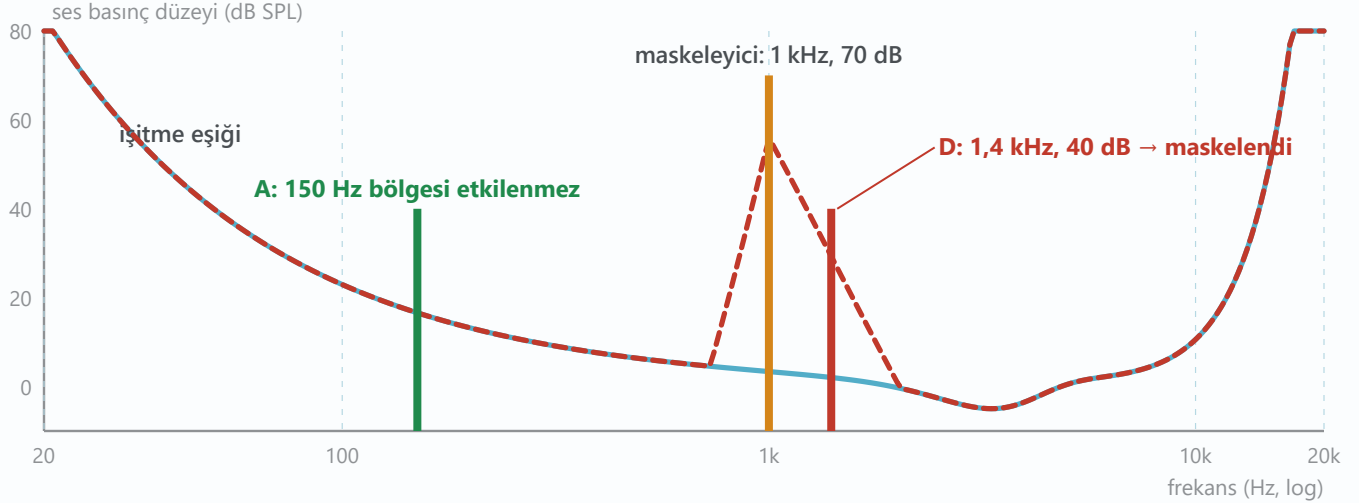
8 harmonik: orijinale çok yakın. **Dönüşüm kodlama** fikri buradan gelir: frekans uzayına geçip küçük bileşenleri kaba nicelemek ya da atmak. MP3/AAC (MDCT) ve JPEG (DCT) bunu yapar.

Psikoakustik: Eşik ve Maskeleye



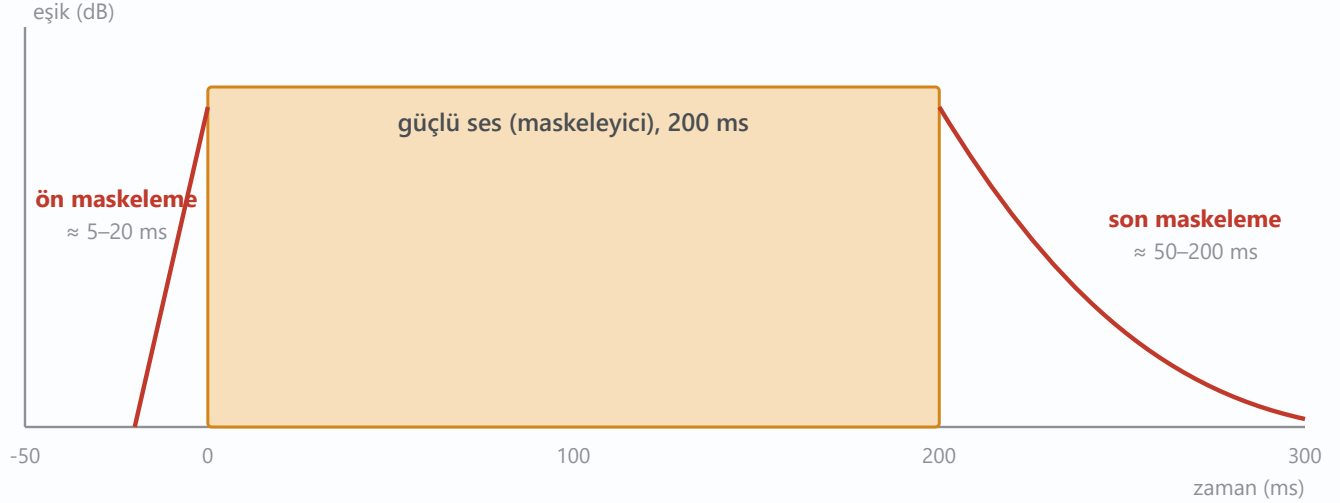
İşitme eşiği frekansa bağlıdır: kulak 2–5 kHz'te en duyarlıdır, çok alçak ve çok yüksek frekanslarda eşik yükselir. Eşiğin altındaki B tonu hiç kodlanmayabilir.

Psikoakustik: Eşik ve Maskeleme



Frekans maskelemesi: 1 kHz, 70 dB'lik güçlü ton yakınındaki eşiği yükseltir (kesikli eğri). 1,4 kHz'teki 40 dB'lik D tonu artık duyulmaz; kodlayıcı ona bit harcamaz.

Psikoakustik: Eşik ve Maskeleye



Zamansal maskeleye: güçlü bir sestten hemen sonra ($\approx 50\text{--}200$ ms) ve az da olsa önce, zayıf sesler duyulmaz. MP3, AAC ve Opus her iki etkiyi psikoakustik modelde kullanır.

Algısal Ses Kodlayıcı ve Kanal Yapılandırmaları

Not: kayıplı adımlar yalnız “niceme”dedir; geri kalanı tersinirdir.



Mono

Tek giriş akışı.

Çift mono

İki bağımsız akış (ör. Türkçe ve İngilizce anlatım).

Ayrık stereo

Sol ve sağ kanal ayrı ayrı sıkıştırılır.

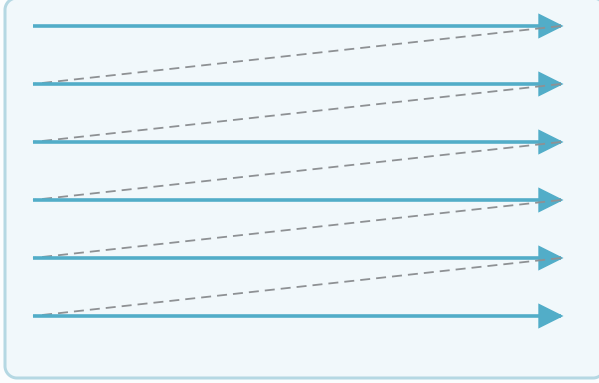
Ortak (joint) stereo

Kanallar arası artıklık kullanılır: orta/yan (M/S) ya da yoğunluk stereo.

Bugün ayrıca 5.1/7.1 çok kanallı ve nesne tabanlı (Dolby Atmos, MPEG-H) ses yaygındır; kanal yerine konumu olan ses nesneleri kodlanır.

Görüntü Tarama: Geçmeli ve Aşamalı

Geçmeli: 1. alan (tek satırlar)



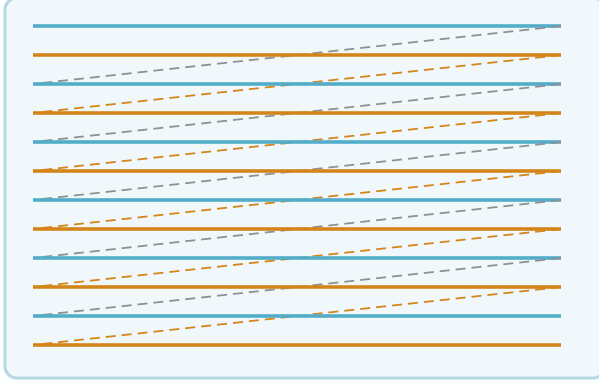
$1/50$ s (PAL) ya da $1/59,94$ s (NTSC)
içinde yarım çözünürlüklü alan

kesikli çizgiler: ışının bir sonraki satırın başına dönüşü (yatay geri dönüş)

Analog TV, enerji ve bant genişliğinden tasarruf için **geçmeli** tarar: önce tek numaralı satırlar (1. alan) çizilir.

Görüntü Tarama: Geçmeli ve Aşamalı

Geçmeli: 2. alan araya girer



iki alan = bir kare

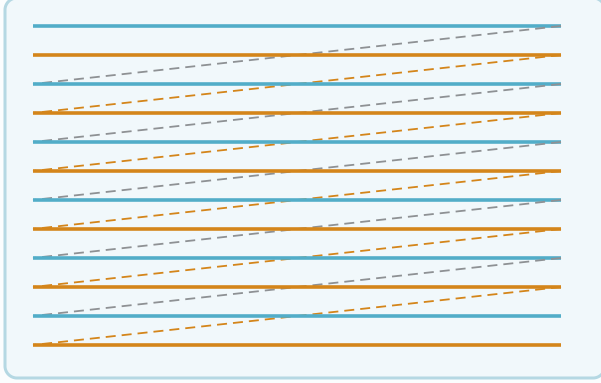
hareketli sahnede "tarak" bozulması;
dijital ekranda ayrıştırma (deinterlace) gerekir

kesikli çizgiler: ışının bir sonraki satırın başına dönüşü (yatay geri dönüş)

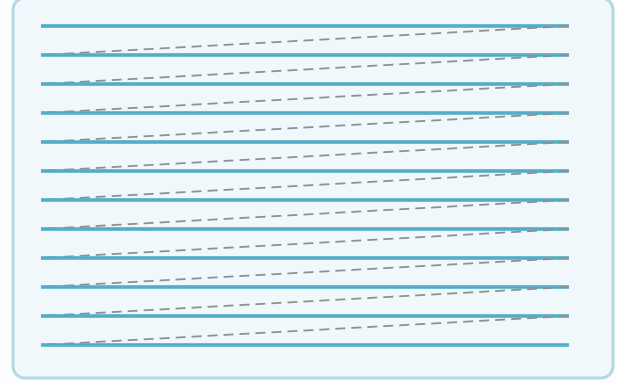
Sonra çift satırlar (2. alan) araya girer. Alan hızı 50/59,94 Hz olduğundan titreme azalır ama kare hızı 25/29,97'dir.

Görüntü Tarama: Geçmeli ve Aşamalı

Geçmeli (iki alan)



Aşamalı (progressive)



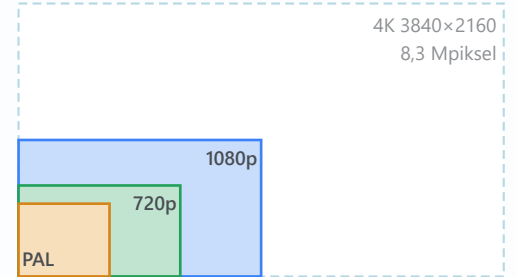
kesikli çizgiler: ışının bir sonraki satırın başına dönüşü (yatay geri dönüş)

Bilgisayar ekranları ve modern video (720p, 1080p, 2160p) **aşamalıdır**: her karede tüm satırlar sırayla. Akış hizmetlerinde geçmeli içerik kodlamadan önce ayrıştırılır.

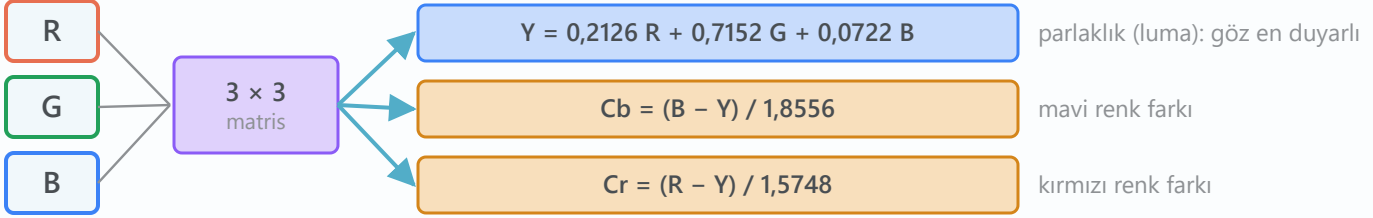
Analog Standartlardan Dijital Video'ya

Standart	Yıl / bölge	Satır, kare hızı	Dijital örnekleme (BT.601)
NTSC	1941 (s/b), 1953 (renkli); K. Amerika, Japonya	525 satır, 29,97 kare/s (59,94 alan/s)	720 × 480
PAL	1967; Avrupa, Türkiye, Asya'nın çoğu	625 satır, 25 kare/s (50 alan/s)	720 × 576
SECAM	1967; Fransa, eski SSCB	625 satır, 25 kare/s	720 × 576
HD (BT.709)	1990'lar–2000'ler	720p, 1080i/1080p; 24–60 kare/s	1280 × 720, 1920 × 1080
UHD (BT.2020/2100)	2012 →	2160p (4K), 4320p (8K); 50–120 kare/s, HDR	3840 × 2160, 7680 × 4320

SECAM, PAL'in bir sürümü değil ayrı bir renk sistemidir (PAL-M ve PAL-N ise PAL türevleridir). 720 × 480 / 720 × 576, analog yayının değil onun dijital örneklemesinin çözünürlüğüdür. Analog karasal yayın çoğu ülkede kapandı; yerini DVB-T2, ATSC 3.0, IPTV ve internet akışı aldı.



Renk Uzayları ve HDR



Neden YCbCr?

Göz parlaklık ayrıntısına renkten çok daha duyarlıdır. Y tam çözünürlükte, Cb/Cr düşük çözünürlükte saklanabilir. Katsayılar BT.709 (HD) içindir; SD BT.601, UHD BT.2020 katsayılarını kullanır.

HDR ve geniş renk gamı

10 bit örnek, BT.2020 renk gamı ve PQ (SMPTE ST 2084, 10.000 nit'e kadar) ya da HLG (yayın) aktarım eğrisi. Biçimler: HDR10, HDR10+, Dolby Vision. SDR genelde 8 bit, ≈ 100 nit.

Baskı ve görüntü dosyaları

CMYK baskı içindir, video kodlamada kullanılmaz. Fotoğrafta sRGB, Display P3 yaygındır; JPEG kodlamadan önce YCbCr'ye çevirir.

Renk Alt Örneklemme: 4:4:4, 4:2:2, 4:2:0

4:4:4

Y (parlaklık)

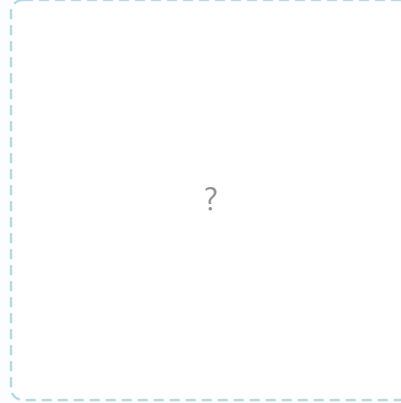
Y	Y	Y	Y
Y	Y	Y	Y

Cb, Cr (renk)

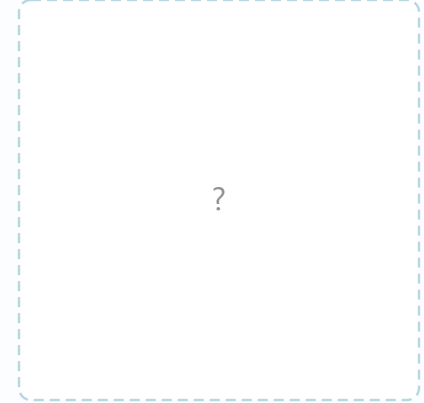
CbCr	CbCr	CbCr	CbCr
CbCr	CbCr	CbCr	CbCr

24 bit/piksel · 1080p kare 6.220.800 B

4:2:2



4:2:0



4:4:4: her piksel Y, Cb ve Cr taşır (8 bit için 24 bit/piksel). Bir 1080p kare 6,2 MB. Ekran görüntüsü ve kurgu ara biçimlerinde kullanılır.

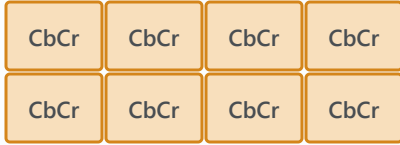
Renk Alt Örneklemme: 4:4:4, 4:2:2, 4:2:0

4:4:4

Y (parlaklık)



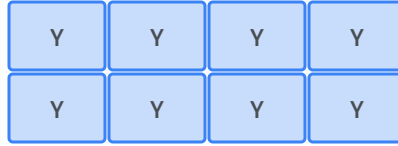
Cb, Cr (renk)



24 bit/piksel · 1080p kare 6.220.800 B

4:2:2

Y (parlaklık)

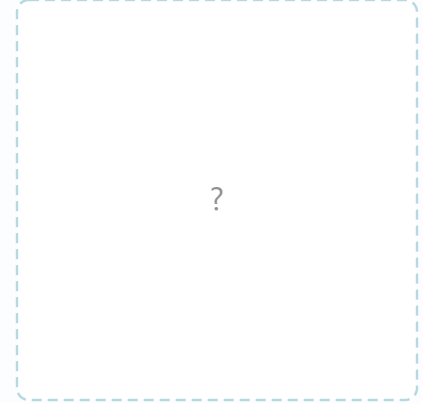


Cb, Cr (renk)



16 bit/piksel · 1080p kare 4.147.200 B

4:2:0



4:2:2: yatayda iki piksel bir renk örneğini paylaşır: 16 bit/piksel, kare 4,1 MB. Profesyonel yayın ve kamera kayıtları.

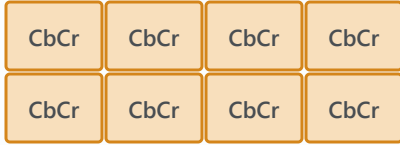
Renk Alt Örnekleme: 4:4:4, 4:2:2, 4:2:0

4:4:4

Y (parlaklık)



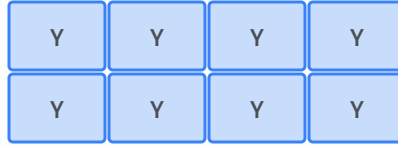
Cb, Cr (renk)



24 bit/piksel · 1080p kare 6.220.800 B

4:2:2

Y (parlaklık)



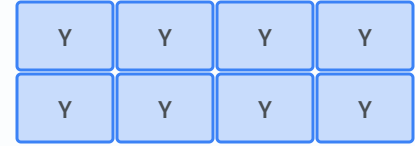
Cb, Cr (renk)



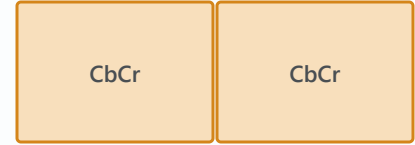
16 bit/piksel · 1080p kare 4.147.200 B

4:2:0

Y (parlaklık)



Cb, Cr (renk)



12 bit/piksel · 1080p kare 3.110.400 B

4:2:0: 2 × 2 piksel bir Cb ve bir Cr paylaşır: 12 bit/piksel, kare 3,1 MB (yarı boyut). JPEG, H.264, HEVC, AV1 akışlarının neredeyse tamamı 4:2:0'dır.

Sıkıştırmanın Temelleri

Kayıpsız ve kayıplı

Kayıpsız (PNG, FLAC, GIF'in LZW'si): bit bit geri çatılır, oran $\approx 2:1$. Kayıplı (JPEG, MP3, H.264): algıda önemsiz bilgi atılır, oran 10:1 – 500:1.

Asimetri

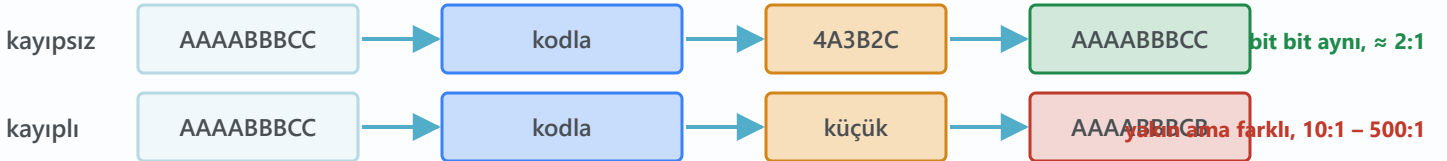
Kodlama bir kez ve yavaş yapılabilir (ör. film için saatler), kod çözme her izleyicide gerçek zamanlı olmalıdır. Canlı yayın ve görüntülü görüşmede kodlama da gerçek zamanlıdır.

Tekrar boyu kodlama (RLE)

AAAABBBCC → 4A3B2C : aynı simgenin art arda tekrarları (sayı, simge) çiftiyle yazılır. JPEG'de sıfır dizilerini kısaltır.

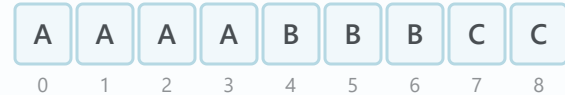
Biçimler

Raster: JPEG, PNG, GIF, WebP, AVIF, HEIC, TIFF, BMP. Vektör: SVG, PDF, EPS, AI. Üst veri: EXIF (kamera, konum), IPTC, XMP. Çözünürlük piksel boyutudur; PPI yalnız baskı ölçeğidir.



RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             char c = s.charAt(i);  
7             int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```



out =

""

encode("AAAABBBCC") çağrıldı; boş bir StringBuilder oluşturulur, i = 0.

Adım 1 / 7

DEĞİŞKENLER

i = 0

c = -

n = -

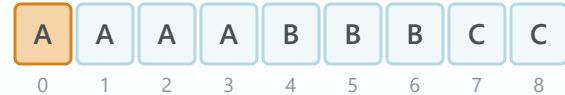
out = ""

ÇIKTI

RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             ▶ char c = s.charAt(i);  
7             ▶ int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```

c = A, n = 1



out =

""

Döngü: c = s.charAt(0) = 'A', n = 1 ile başlanır.

Adım 2 / 7

DEĞİŞKENLER

i = 0

c = A

n = 1

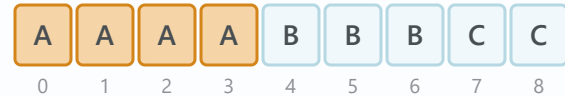
out = ""

ÇIKTI

RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             char c = s.charAt(i);  
7             int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```

c = A, n = 4



out = ""

İç döngü sonraki karakterler 'A' oldukça n'yi artırır: n = 4 (s[4] = 'B' farklı).

Adım 3 / 7

DEĞİŞKENLER

i = 0

c = A

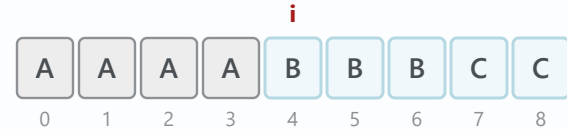
n = 4

out = ""

ÇIKTI

RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             char c = s.charAt(i);  
7             int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```



out =

"4A"

"4A" eklenir; $i += 4 \rightarrow i = 4$.

Adım 4 / 7

DEĞİŞKENLER

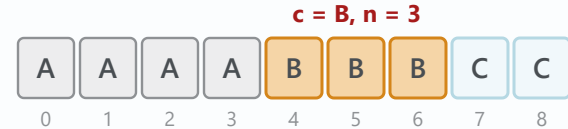
 $i = 4$ $c = A$ $n = 4$ $out = 4A$

ÇIKTI

RLE Kodlayıcıyı İzleme

```

1 public class RLE {
2     static String encode(String s) {
3         StringBuilder out = new StringBuilder();
4         int i = 0;
5         while (i < s.length()) {
6             char c = s.charAt(i);
7             int n = 1;
8             while (i + n < s.length()
9                 && s.charAt(i + n) == c) n++;
10            ▶ out.append(n).append(c);
11            ▶ i += n;
12        }
13        return out.toString();
14    }
15    public static void main(String[] a) {
16        System.out.println(encode("AAAABBBCC"));
17    }
18 }
    
```



out = "4A3B"

c = 'B', üç tekrar: n = 3; "3B" eklenir, i = 7.

Adım 5 / 7

DEĞİŞKENLER

i = 7

c = B

n = 3

out = 4A3B

ÇIKTI

RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             char c = s.charAt(i);  
7             int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```



out =

"4A3B2C"

c = 'C': i + n = 9 dizinin sonu, n = 2; "2C" eklenir, i = 9.

Adım 6 / 7

DEĞİŞKENLER**i = 9****c = C****n = 2****out = 4A3B2C****ÇIKTI**

RLE Kodlayıcıyı İzleme

```
1 public class RLE {  
2     static String encode(String s) {  
3         StringBuilder out = new StringBuilder();  
4         int i = 0;  
5         while (i < s.length()) {  
6             char c = s.charAt(i);  
7             int n = 1;  
8             while (i + n < s.length()  
9                 && s.charAt(i + n) == c) n++;  
10            out.append(n).append(c);  
11            i += n;  
12        }  
13        return out.toString();  
14    }  
15    public static void main(String[] a) {  
16        System.out.println(encode("AAAABBBCC"));  
17    }  
18 }
```



out =

"4A3B2C"

i = 9 = s.length(): döngü biter, sonuç döndürülür ve yazdırılır. 9 karakter → 6 karakter.

Adım 7 / 7

DEĞİŞKENLER

i = 9

c = C

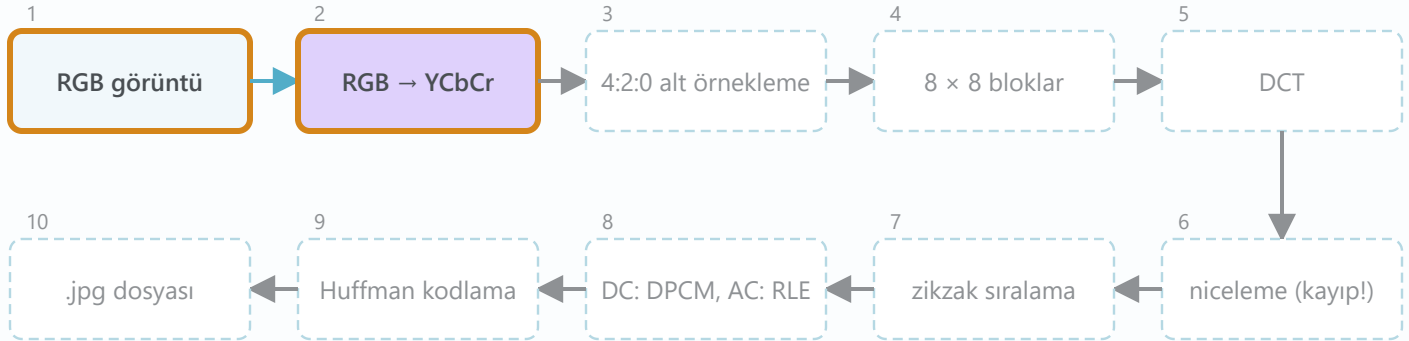
n = 2

out = 4A3B2C

ÇIKTI

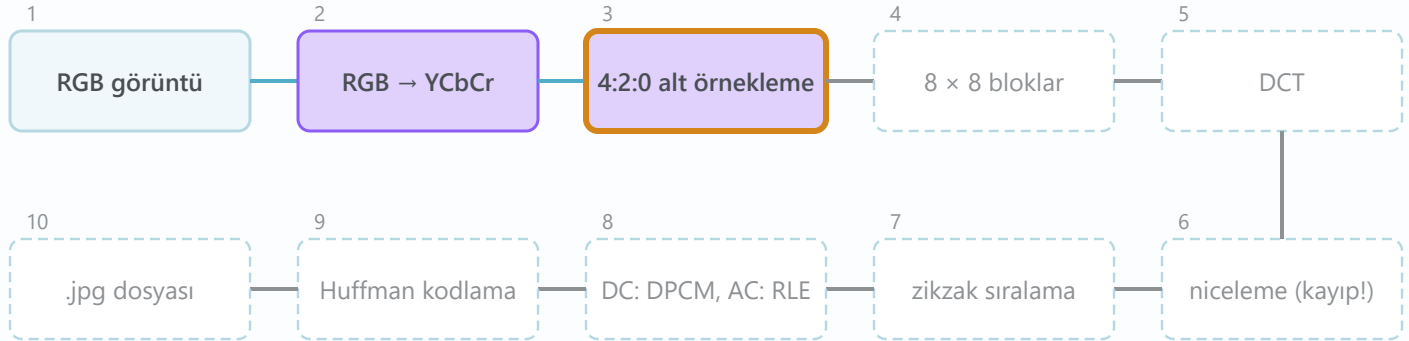
4A3B2C

JPEG Sıkıştırma Boru Hattı



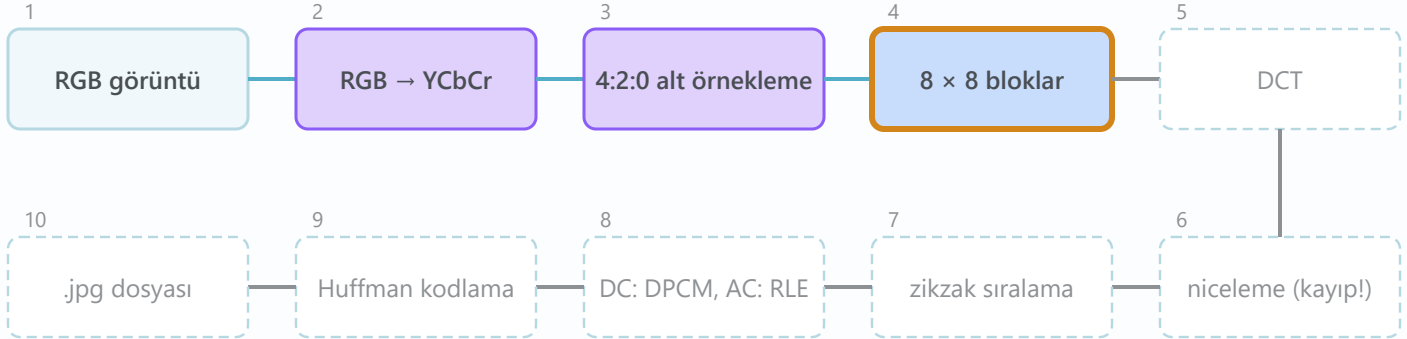
Görüntü RGB'den **YCbCr**'ye çevrilir: parlaklık ve renk ayrılır.

JPEG Sıkıştırma Boru Hattı



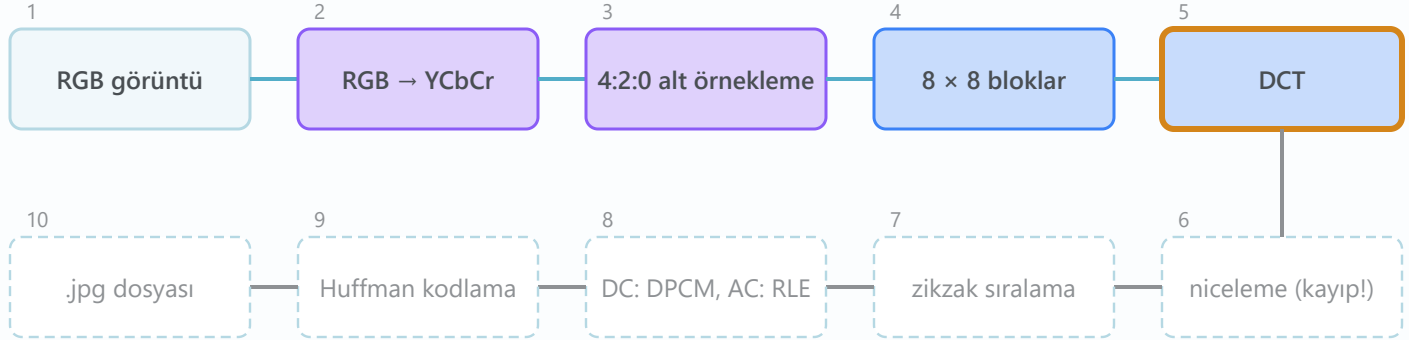
Renk bileşenleri 4:2:0 ile **alt örneklenir**: veri, görsel kayıp neredeyse olmadan yarıya iner.

JPEG Sıkıştırma Boru Hattı



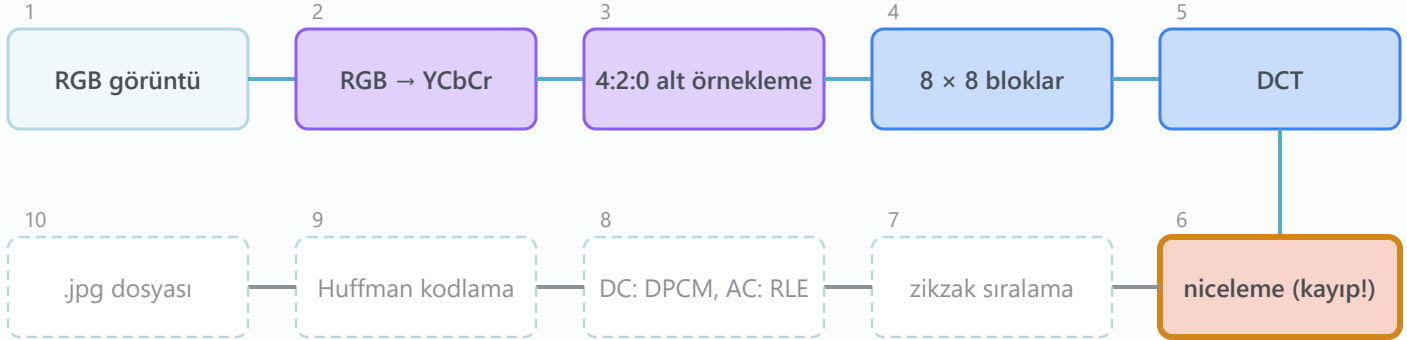
Her bileşen 8×8 piksellik **bloklara** bölünür; bloklar birbirinden bağımsız işlenir (yüksek oranda “bloklanma” bu yüzden görülür).

JPEG Sıkıştırma Boru Hattı



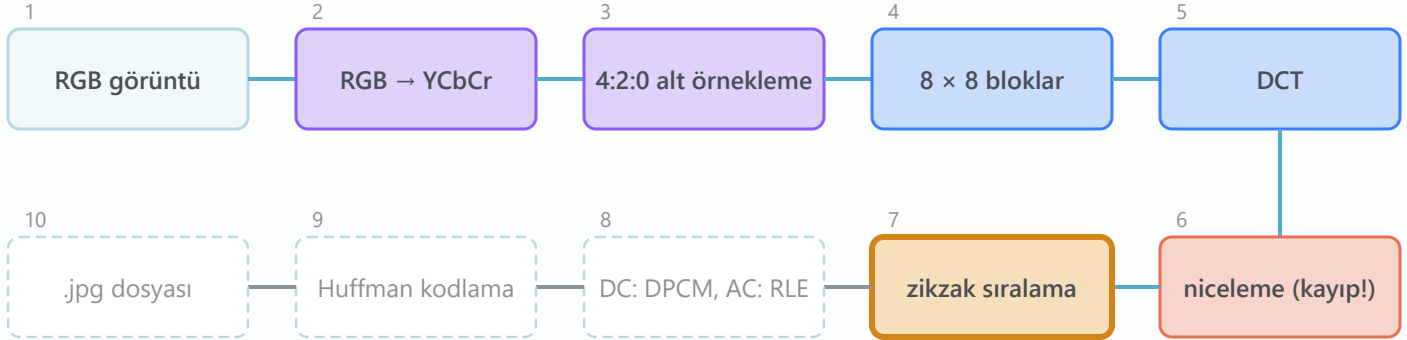
DCT her bloğu 64 frekans katsayısına dönüştürür. Enerji sol üstteki düşük frekanslarda toplanır. Bu adım tersinirdir.

JPEG Sıkıştırma Boru Hattı



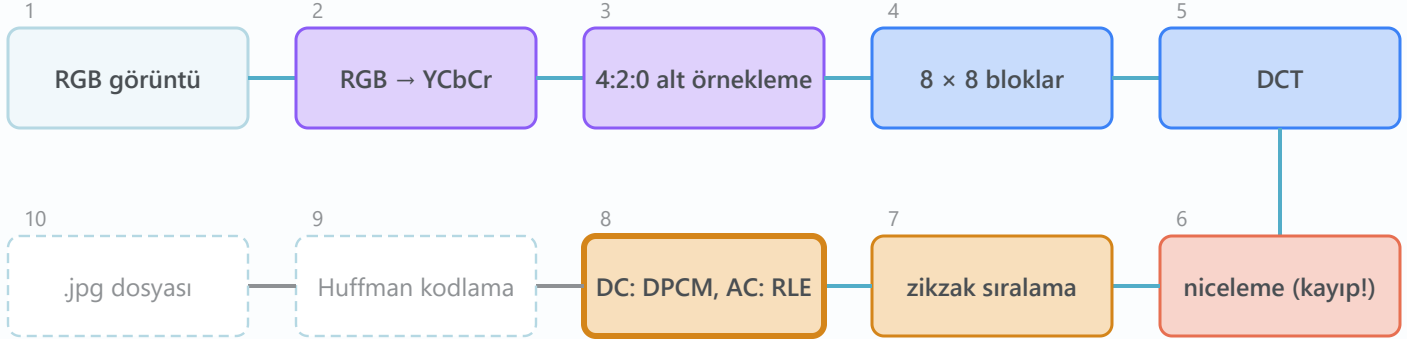
Niceleme: her katsayı tablo değerine bölünüp yuvarlanır. Yüksek frekanslar sıfırlanır. JPEG'in tek kayıplı adımı budur; kalite ayarı tabloyu ölçekler.

JPEG Sıkıştırma Boru Hattı



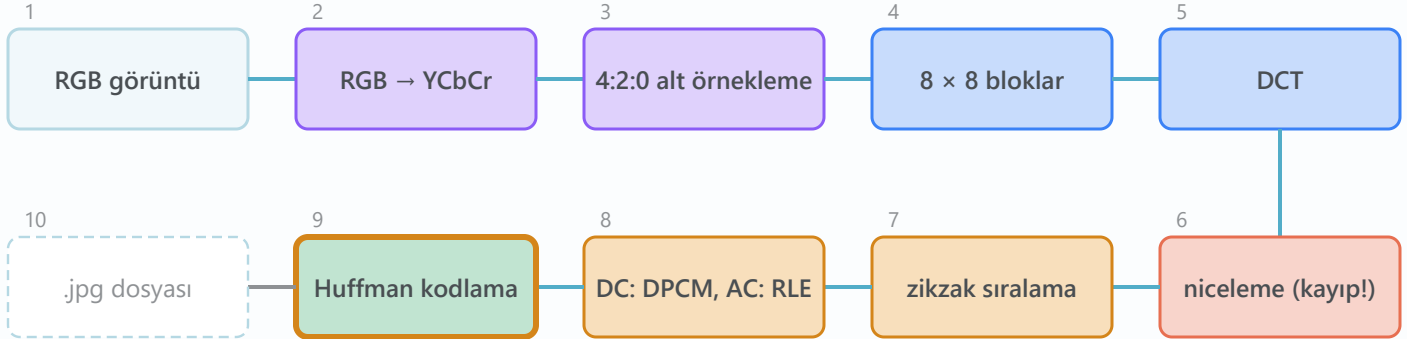
Zikzak sıra katsayıları düşükten yükseğe dizer, sondaki sıfırlar art arda gelir.

JPEG Sıkıştırma Boru Hattı



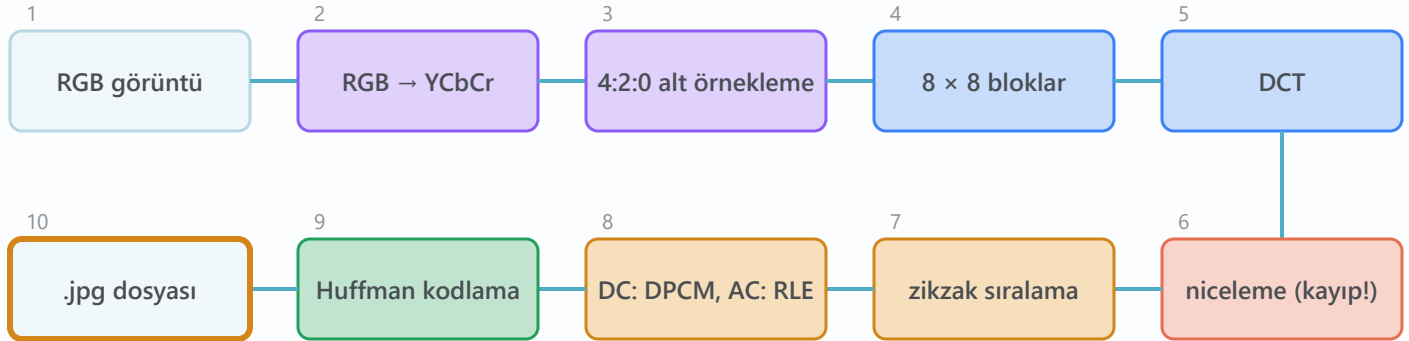
DC katsayısı önceki bloğun DC'sine göre fark (DPCM) olarak, AC katsayıları (sıfır sayısı, değer) çiftleri olarak **RLE** ile kodlanır; kalan sıfırlar tek EOB simgesidir.

JPEG Sıkıştırma Boru Hattı



Huffman kodlama sık simgelere kısa kodlar verir. Kod çözme bu adımların tersidir; tipik oran 10:1 – 20:1.

JPEG Sıkıştırma Boru Hattı



Sonuç: başlıklar, tablolar ve entropi kodlu veri içeren .jpg. Modern ardılları (WebP, AVIF, JPEG XL) aynı dönüşüm-niceleme-entropi fikrini daha gelişmiş araçlarla kullanır.

DCT ve Niceleme: Sayısal Örnek

(a) Y bloğu (0–255)

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

merkezleme: –128

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-73	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

Bir 8×8 parlaklık bloğu (değerler 0–255). DCT'den önce her değerden 128 çıkarılır; değerler –128 ... 127 aralığına gelir.

DCT ve Niceleme: Sayısal Örnek

merkezlenmiş blok

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-73	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-60	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

(b) DCT katsayıları (yuvarlanmış)

-415	-30	-61	27	56	-20	-2	0
4	-22	-61	10	13	-7	-9	5
-47	7	77	-25	-29	10	5	-6
-49	12	34	-15	-10	6	2	2
12	-7	-13	-4	-2	2	-3	3
-8	3	2	-6	-2	1	4	2
-1	0	0	-2	-1	-3	4	-1
0	0	-1	-4	-1	0	1	2

DCT sonucu: sol üst DC katsayısı (-415) bloğun ortalama parlaklığıdır; sağa ve aşağı gidildikçe frekans artar, katsayılar küçülür.

DCT ve Niceleme: Sayısal Örnek

DCT katsayıları

-415	-30	-61	27	56	-20	-2	0
4	-22	-61	10	13	-7	-9	5
-47	7	77	-25	-29	10	5	-6
-49	12	34	-15	-10	6	2	2
12	-7	-13	-4	-2	2	-3	3
-8	3	2	-6	-2	1	4	2
-1	0	0	-2	-1	-3	4	-1
0	0	-1	-4	-1	0	1	2

niceleme tablosu Q (kalite 50)

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Standart parlaklık niceleme tablosu: yüksek frekanslar (sağ alt) büyük bölenlerle cezalandırılır, çünkü göz ince ayrıntıdaki hatayı zor fark eder.

DCT ve Niceleme: Sayısal Örnek

Q

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(c) yuvarla(DCT / Q)

-26	-3	-6	2	2	-1	0	0
0	-2	-4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-3	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Nicelenmiş blok: 64 katsayıdan yalnız 20'si sıfırdan farklı; $DC = \text{yuvarla}(-415,4 / 16) = -26$.
Bilgi kaybı tam bu adımda oluşur.

Zikzak, RLE ve Huffman

nicelenmiş blok

-26	3	-6	2	2	1	0	0
0	-2	-4	1	1	0	0	0
-3	1	5	-1	-1	0	0	0
-3	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Okuma sırası: (0,0) → (0,1) → (1,0) → (2,0) → (1,1) → ...

çapraz çizgiler boyunca, yön her köşegende değişir.

İlk 28 katsayı:

-26, -3, 0, -3, -2, -6, 2, -4, 1, -3, 1, 1, 5, 1,
2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, 0, 0 ...

Son sıfırdan farklı katsayı 26. sırada; geri kalan 38 değer sıfır.

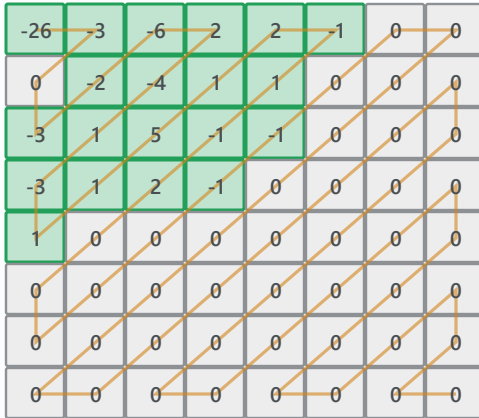
Nicelenmiş katsayılar **zikzak** sırayla okunur: düşük frekanslar önce, yüksek frekanslar (çoğu sıfır) sonra.

(0,-1) (0,2) (5,-1) (0,-1) EOB

Adım 2 / 3

Zikzak, RLE ve Huffman

nicelenmiş blok



DC = -26 → önceki bloğun DC'sine göre fark (DPCM)

AC (sıfır sayısı, değer) simgeleri:

(0,-3) (1,-3) (0,-2) (0,-6) (0,2)

(0,-4) (0,1) (0,-3) (0,1) (0,1)

(0,5) (0,1) (0,2) (0,-1) (0,1)

(0,-1) (0,2) (5,-1) (0,-1) EOB

64 katsayı → 1 DC + 19 AC simgesi + EOB

Huffman: sık görülen (0, küçük) simgelere 2–4 bitlik

kodlar → blok ≈ 93 bit (ham 512 bit, ≈ 5,5 : 1)

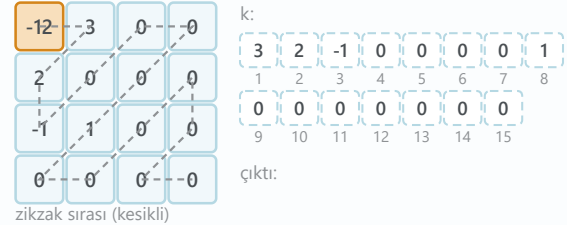
Simgeler **Huffman** ile kodlanır. Standart tablolarla örnek blok 512 bitten (64×8) ≈ 93 bite iner (DC 8 + AC 85 bit); oran ≈ 5,5:1 yalnız bu blokta, düz alanlarda çok daha yüksektir.

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
    9,12,13,10,7,11,14,15};
▶ void ac_kodla(int B[16]) {
▶   int sifir = 0;
5   for (int k = 1; k < 16; k++) { /* k=0: DC */
6       int v = B[zz[k]];
7       if (v == 0) { sifir++; continue; }
8       printf("(%d,%d) ", sifir, v);
9       sifir = 0;
10  }
11  printf("EOB");
12 }

```



4 × 4'lük nicelenmiş blok (gerçek JPEG 8 × 8). DC katsayısı (-12) önceki bloğa göre fark olarak ayrı kodlanır; geri kalan 15 AC katsayısı zikkak sırayla okunacak.

Adım 1 / 8

DEĞİŞKENLER

sifir = 0

DC = -12

ÇIKTI



zikkak sırası (kesikli)

Adım 2 / 8

k = 1 zz[k] = 1 v = 3

sifir = 0

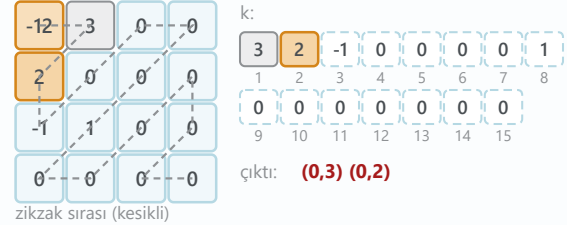
 $(0, 3)$

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     ▶ for (int k = 1; k < 16; k++) { /* k=0: DC */
6         ▶ int v = B[zz[k]];
7         if (v == 0) { sifir++; continue; }
8         ▶ printf("(%d,%d) ", sifir, v);
9         ▶ sifir = 0;
10    }
11    printf("EOB");
12 }

```



k = 2: zz[2] = 4 → v = 2. Sıfır deęil: önünde 0 sıfır var, **(0,2)** yazılır.

Adım 3 / 8

DEęİŞKENLER

k = 2

zz[k] = 4

v = 2

sifir = 0

ÇIKTI

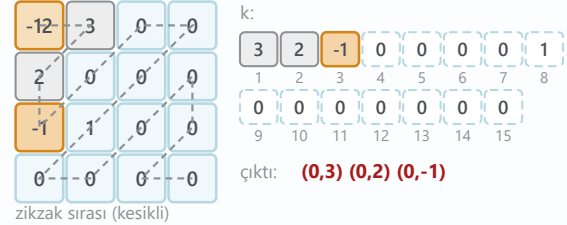
(0,3) (0,2)

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     ▶ for (int k = 1; k < 16; k++) { /* k=0: DC */
6         ▶ int v = B[zz[k]];
7         if (v == 0) { sifir++; continue; }
8         ▶ printf("(%d,%d) ", sifir, v);
9         ▶ sifir = 0;
10    }
11    printf("EOB");
12 }

```



k = 3: zz[3] = 8 → v = -1. Sıfır değil: önünde 0 sıfır var, **(0,-1)** yazılır.

Adım 4 / 8

DEĞİŞKENLER

k = 3

zz[k] = 8

v = -1

sifir = 0

ÇIKTI

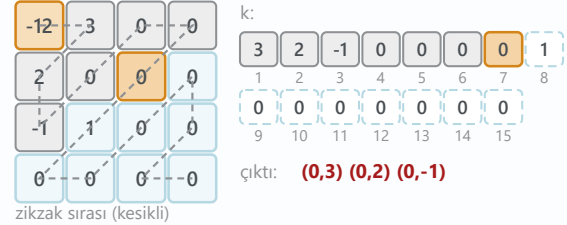
(0,3) (0,2) (0,-1)

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     ▶ for (int k = 1; k < 16; k++) { /* k=0: DC */
6         ▶ int v = B[zz[k]];
7         ▶ if (v == 0) { sifir++; continue; }
8         printf("(%d,%d) ", sifir, v);
9         sifir = 0;
10    }
11    printf("EOB");
12 }

```



k = 4 ... 7: dört katsayı da 0. Yazılmaz, yalnızca sıfır sayacı artar: **sifir = 4**.

Adım 5 / 8

DEĞİŞKENLER

k = 7

v = 0

sifir = 4

ÇIKTI

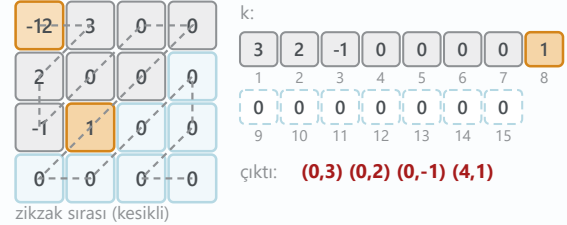
(0,3) (0,2) (0,-1)

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     ▶ for (int k = 1; k < 16; k++) { /* k=0: DC */
6         ▶ int v = B[zz[k]];
7         if (v == 0) { sifir++; continue; }
8         ▶ printf("(%d,%d) ", sifir, v);
9         ▶ sifir = 0;
10    }
11    printf("EOB");
12 }

```



k = 8: zz[8] = 9 → v = 1. Önündeki dört sıfır tek çiftte toplanır: **(4,1)**; sayaç sıfırlanır.

Adım 6 / 8

DEĞİŞKENLER

k = 8

zz[k] = 9

v = 1

sifir = 0

ÇIKTI

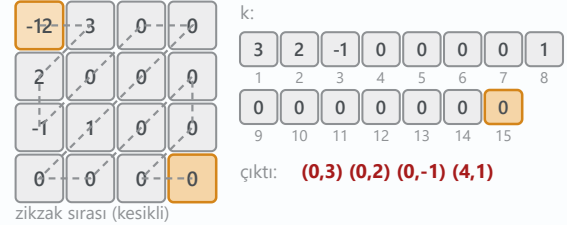
(0,3) (0,2) (0,-1) (4,1)

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     for (int k = 1; k < 16; k++) { /* k=0: DC */
6         int v = B[zz[k]];
7         if (v == 0) { sifir++; continue; }
8         printf("(%d,%d) ", sifir, v);
9         sifir = 0;
10    }
11    printf("EOB");
12 }

```



k = 9 ... 15: kalan yedi katsayı sıfır. Yüksek frekanslar nicelemede sıfırlandığı için zikkak sırada hep sonda toplanır.

Adım 7 / 8

DEĞİŞKENLER

k = 15

v = 0

sifir = 7

ÇIKTI

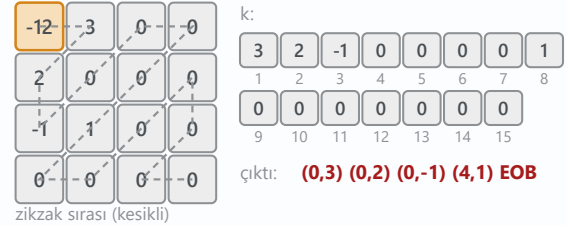
(0,3) (0,2) (0,-1) (4,1)

AC Katsayılarını Kodlama: Kod İzleme

```

1 int zz[16] = {0,1,4,8,5,2,3,6,
2             9,12,13,10,7,11,14,15};
3 void ac_kodla(int B[16]) {
4     int sifir = 0;
5     for (int k = 1; k < 16; k++) { /* k=0: DC */
6         int v = B[zz[k]];
7         if (v == 0) { sifir++; continue; }
8         printf("(%d,%d) ", sifir, v);
9         sifir = 0;
10    }
11    printf("EOB");
12 }

```



Döngü bitti: sondaki yedi sıfır tek bir **EOB** simgesiyle anlatılır. 15 sayı yerine 5 simge Huffman ile kodlanacak.

Adım 8 / 8

DEĞİŞKENLER

k = 16

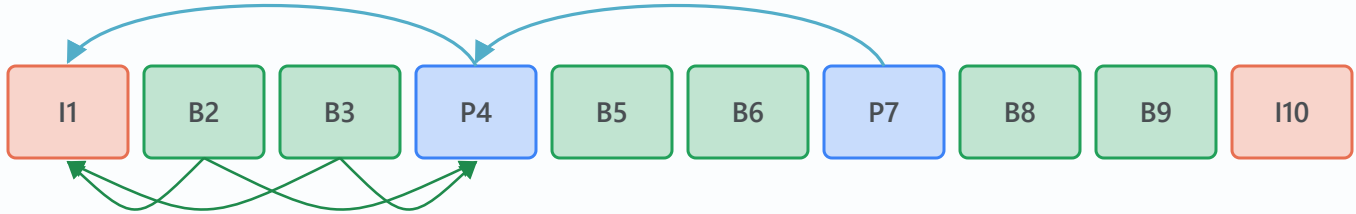
sifir = 7

ÇIKTI

(0,3) (0,2) (0,-1) (4,1) EOB

MPEG: I, P ve B Kareleri

görüntüleme sırası

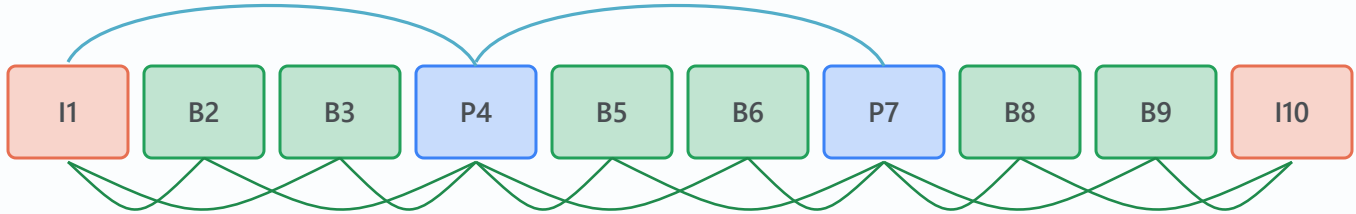


mavi: $P \rightarrow$ önceki I/P · yeşil: $B \rightarrow$ önceki ve sonraki referans

I (iç kodlu): kendi başına, JPEG benzeri. **P** (öngörölü): önceki I/P referansına göre hareket dengelemeli fark. **B** (çift yönlü): önceki ve sonraki referansa göre.

MPEG: I, P ve B Kareleri

görüntüleme sırası



mavi: P → önceki I/P · yeşil: B → önceki ve sonraki referans

kod çözme / iletim sırası



B kareleri gelecekteki referansa ihtiyaç duyduğundan kodlayıcı sırayı değiştirir: I1 P4 B2 B3 P7 B5 B6 ... Kod çözücü en az bir referans kareyi tamponda tutar; bu gecikme ekler.

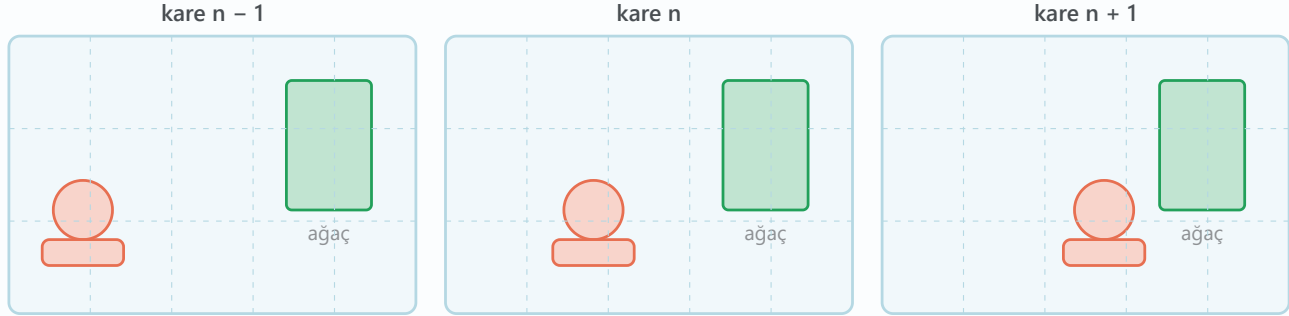
MPEG: I, P ve B Kareleri

Örnek kare boyutları (1080p, ≈ 5 Mbit/s; temsili)



I kareleri en büyüktür; P ve özellikle B kareleri çok daha küçüktür. GOP (I'den I'ye) akışta genelde 2 s'dir; atlama ve bölüt sınırları I karelerinde yapılır.

Hareket Dengeleme

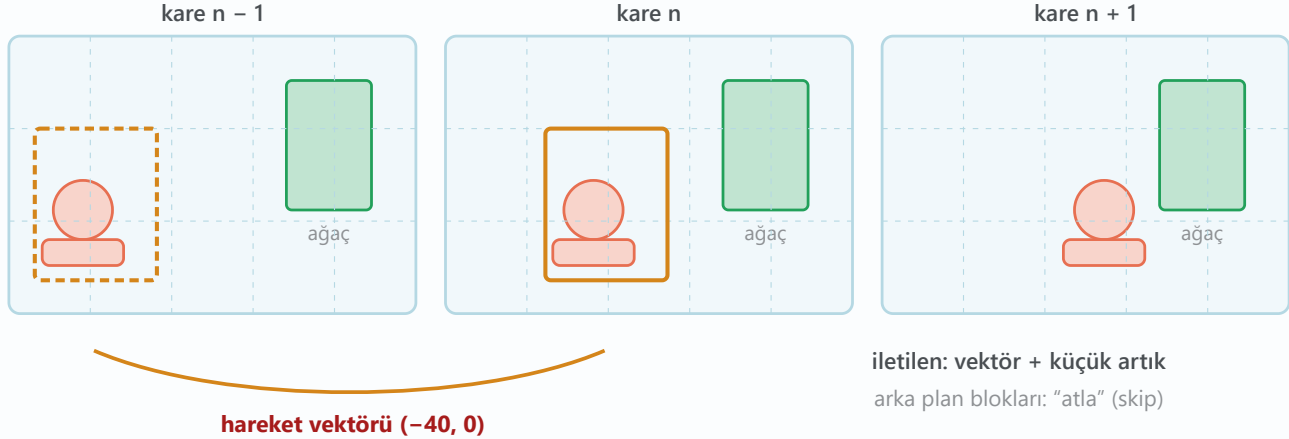


Ardışık karelerde sahnenin çoğu aynıdır (ağaç); yalnız top sağa kayar. Kareleri bağımsız kodlamak bu **zamansal artıklığı** boşa harcar.

Adım 1 / 2

MPEG-2 16×16 makro blok ve yarım piksel doğruluk kullanır; H.264 $4 \times 4 - 16 \times 16$ bölmeli ve çeyrek piksel, HEVC 64×64 'e, AV1 128×128 'e kadar esnek bloklar ve çok sayıda referans kare kullanır.

Hareket Dengeleme



Hareket dengeleme: kodlayıcı n karesindeki her blok için $n - 1$ 'de en iyi eşleşen bloğu arar; hareket vektörünü ve farkı (artık) DCT ile kodlar. Değişmeyen bloklar yalnız "atla" bayrağıyla geçer.

Adım 2 / 2

MPEG-2 16×16 makro blok ve yarım piksel doğruluk kullanır; H.264 $4 \times 4 - 16 \times 16$ bölmeli ve çeyrek piksel, HEVC 64×64 'e, AV1 128×128 'e kadar esnek bloklar ve çok sayıda referans kare kullanır.

Hareket Vektörü Arama: Kod İzleme

```
► /* cur[0..3]: x = 3'teki blok */  
► int en_iyi = 0, en_kucuk = INT_MAX;  
3 for (int d = -2; d <= 2; d++) { /* pencere */  
4     int sad = 0;  
5     for (int j = 0; j < 4; j++)  
6         sad += abs(cur[j] - prev[x + d + j]);  
7     if (sad < en_kucuk) {  
8         en_kucuk = sad;  
9         en_iyi = d;  
10    }  
11 }  
12 /* vektör = en_iyi, artık = cur - eşleşen */
```

önceki kare (n - 1)

10	10	50	80	90	60	20	10	10	10
0	1	2	3	4	5	6	7	8	9

şimdiki kare (n): blok x = 3

52	80	88	60
----	----	----	----

Tek boyutlu basit örnek: önceki karede parlak bir nesne (50–90) var; şimdiki karedeki x = 3 bloğu ona benziyor. Kodlayıcı bu bloğu önceki karede ± 2 piksel içinde arıyor.

Adım 1 / 7

DEĞİŞKENLER

x = 3

en_iyi = 0

en_kucuk = ∞

ÇIKTI



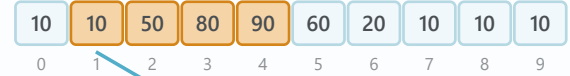
Hareket Vektörü Arama: Kod İzleme

```

1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
▶ for (int d = -2; d <= 2; d++) { /* pencere */
▶     int sad = 0;
▶     for (int j = 0; j < 4; j++)
▶         sad += abs(cur[j] - prev[x + d + j]);
▶     if (sad < en_kucuk) {
▶         en_kucuk = sad;
▶         en_iyi = d;
10     }
11 }
12 /* vektör = en_iyi, artık = cur - eşleşen */

```

önceki kare (n - 1)



şimdiki kare (n): blok x = 3



d = -2: SAD = 110

en iyi: d = -2, SAD = 110

d = -2: $|52-10| + |80-50| + |88-80| + |60-90|$
 = **110**. Şimdiye kadarki en küçük: kaydedilir.

Adım 2 / 7

DEĞİŞKENLER

d = -2

sad = 110

en_iyi = -2

en_kucuk = 110

ÇIKTI



Hareket Vektörü Arama: Kod İzleme

```

1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
▶ for (int d = -2; d <= 2; d++) { /* pencere */
▶     int sad = 0;
▶     for (int j = 0; j < 4; j++)
▶         sad += abs(cur[j] - prev[x + d + j]);
▶     if (sad < en_kucuk) {
▶         en_kucuk = sad;
▶         en_iyi = d;
10     }
11 }
12 /* vektör = en_iyi, artık = cur - eşleşen */

```

önceki kare (n - 1)



şimdiki kare (n): blok x = 3



d = -1: SAD = 4

en iyi: d = -1, SAD = 4

d = -1: $|52-50| + |80-80| + |88-90| + |60-60|$
= 4. Şimdiye kadarki en küçük: kaydedilir.

Adım 3 / 7

DEĞİŞKENLER

d = -1

sad = 4

en_iyi = -1

en_kucuk = 4

ÇIKTI



Hareket Vektörü Arama: Kod İzleme

```

1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
▶ for (int d = -2; d <= 2; d++) { /* pencere */
▶     int sad = 0;
▶     for (int j = 0; j < 4; j++)
▶         sad += abs(cur[j] - prev[x + d + j]);
▶     if (sad < en_kucuk) {
8         en_kucuk = sad;
9         en_iyi = d;
10     }
11 }
12 /* vektör = en_iyi, artık = cur - eşleşen */

```

önceki kare (n - 1)



şimdiki kare (n): blok x = 3



d = 0: SAD = 106

en iyi: d = -1, SAD = 4

d = 0: $|52-80| + |80-90| + |88-60| + |60-20| = 106$. 4'ten büyük: değişmez.

Adım 4 / 7

DEĞİŞKENLER

d = 0

sad = 106

en_iyi = -1

en_kucuk = 4

ÇIKTI



Hareket Vektörü Arama: Kod İzleme

```
1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
▶ for (int d = -2; d <= 2; d++) { /* pencere */
▶     int sad = 0;
▶     for (int j = 0; j < 4; j++)
▶         sad += abs(cur[j] - prev[x + d + j]);
▶     if (sad < en_kucuk) {
8         en_kucuk = sad;
9         en_iyi = d;
10    }
11 }
12 /* vektör = en_iyi, artık = cur - eşleşen */
```

önceki kare (n - 1)



şimdiki kare (n): blok x = 3



d = 1: SAD = 176

en iyi: d = -1, SAD = 4

d = 1: $|52-90| + |80-60| + |88-20| + |60-10| = 176$. 4'ten büyük: değişmez.

Adım 5 / 7

DEĞİŞKENLER

d = 1

sad = 176

en_iyi = -1

en_kucuk = 4

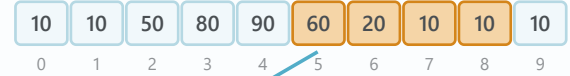
ÇIKTI



Hareket Vektörü Arama: Kod İzleme

```
1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
▶ for (int d = -2; d <= 2; d++) { /* pencere */
▶     int sad = 0;
▶     for (int j = 0; j < 4; j++)
▶         sad += abs(cur[j] - prev[x + d + j]);
▶     if (sad < en_kucuk) {
8         en_kucuk = sad;
9         en_iyi = d;
10    }
11 }
12 /* vektör = en_iyi, artık = cur - eşleşen */
```

önceki kare (n - 1)



şimdiki kare (n): blok x = 3



d = 2: SAD = 196

en iyi: d = -1, SAD = 4

d = 2: $|52-60| + |80-20| + |88-10| + |60-10| = 196$. 4'ten büyük: değişmez.

Adım 6 / 7

DEĞİŞKENLER

d = 2

sad = 196

en_iyi = -1

en_kucuk = 4

ÇIKTI



Hareket Vektörü Arama: Kod İzleme

```
1 /* cur[0..3]: x = 3'teki blok */
2 int en_iyi = 0, en_kucuk = INT_MAX;
3 for (int d = -2; d <= 2; d++) { /* pencere */
4     int sad = 0;
5     for (int j = 0; j < 4; j++)
6         sad += abs(cur[j] - prev[x + d + j]);
7     if (sad < en_kucuk) {
8         en_kucuk = sad;
9         en_iyi = d;
10    }
11 }
▶ /* vektör = en_iyi, artık = cur - eşleşen */
```

önceki kare (n - 1)

10	10	50	80	90	60	20	10	10	10
0	1	2	3	4	5	6	7	8	9

şimdiki kare (n): blok x = 3

52	80	88	60
----	----	----	----

en iyi: d = -1, SAD = 4

Hareket vektörü **-1**: blok önceki karede 1 piksel soldaydı, yani nesne sağa kaydı. Kodlanan şey vektör ve küçük artıktır (2, 0, -2, 0); 4 piksel yerine neredeyse yalnız sıfırlar.

Adım 7 / 7

DEĞİŞKENLER

en_iyi = **-1**

en_kucuk = **4**

ÇIKTI

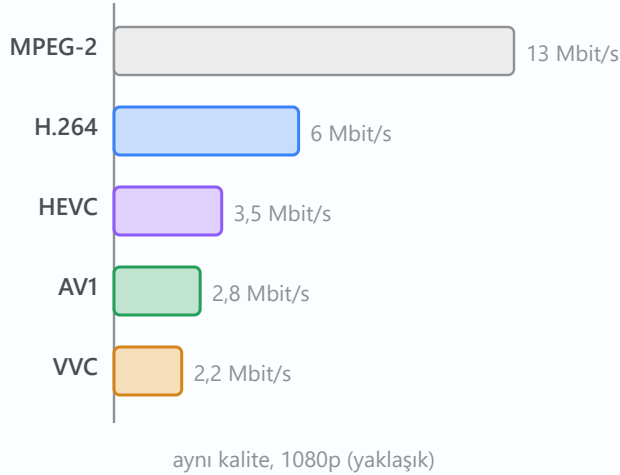


MPEG Standartları ve Modern Kodekler

Kodek	Yıl	Tipik kullanım	1080p için kabaca
MPEG-1	1993	Video CD; MP3 bu standardın ses katmanıdır	—
MPEG-2 (H.262)	1995	DVD, DVB yayın; geçmeli video desteği	≈ 12–15 Mbit/s
MPEG-4 Part 2	1999	DivX/Xvid; bugün eskimiş	≈ 8–10 Mbit/s
H.264 / AVC	2003	en yaygın uyumluluk: web, Blu-ray, görüntülü görüşme	≈ 5–8 Mbit/s
H.265 / HEVC	2013	UHD Blu-ray, 4K HDR yayın ve akış, iPhone kaydı	≈ 3–4 Mbit/s
VP9	2013	YouTube, WebRTC (Google, telifsiz)	≈ 3–4 Mbit/s
AV1	2018	Netflix, YouTube, Twitch; AOMedia, telifsiz	≈ 2,5–3 Mbit/s
H.266 / VVC	2020	8K, yayın; donanım desteği henüz sınırlı	≈ 2 Mbit/s

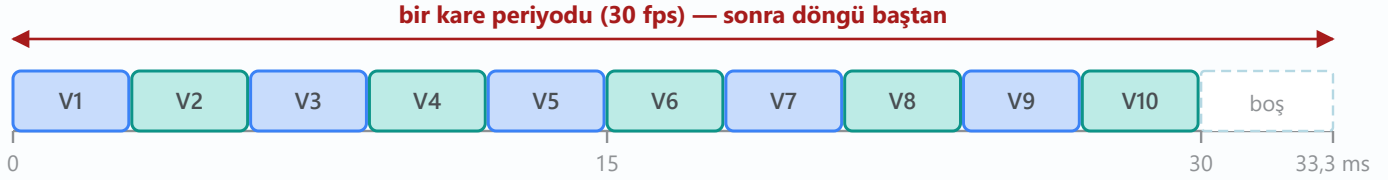
MPEG-3 hiç yayımlanmadı; HDTV için planlanmış, 1992’de MPEG-2’ye katılmıştır. MP3, MPEG-1/2 Audio Layer III’tür. Bit hızları aynı algısal kalite için yaklaşıktır ve içeriğe göre değişir.

Kodek Verimliliği ve Donanım Kod Çözücüler



- Her kuşak aynı kalite için bit hızını kabaca **%30–50** düşürür; bedeli kodlayıcıda 5–10 kat, kod çözücüde 2–3 kat daha fazla işlemdir.
- 4K/8K, HDR ve 60–120 fps kod çözme CPU yerine **sabit işlevli donanımda** yapılır: NVIDIA NVDEC/NVENC, Intel Quick Sync, AMD VCN, Apple VideoToolbox, telefon SoC'leri.
- İşletim sistemi arayüzleri: Linux VA-API / V4L2, Windows Media Foundation ve D3D11/12 Video, Android MediaCodec. Kareler GPU belleğinde kalır (dmabuf, sıfır kopya).
- Donanım yoksa yazılım kod çözme (ör. AV1 için dav1d) çok çekirdekli CPU'yu ve pili zorlar; oynatıcı desteklenen kodeği seçer.

Homojen Çoklu Ortam Süreçleri



her süreç bir kare çözer: $3 \text{ ms} \times 10 = 30 \text{ ms} \leq 33,3 \text{ ms} \rightarrow \text{en çok } \lfloor 33,3 / 3 \rfloor = 11 \text{ akış}$

Varsayım

Video sunucusundaki tüm akışlar aynı kare hızında ve kare başına aynı işlem süresiyle çalışır (ör. hepsi 30 fps).

Çizelge

Basit döngüsel (round-robin): her periyotta her süreç bir kare üretir ve bloklanır; zamanlayıcı kesmesi 33,3 ms'de bir yeni turu başlatır.

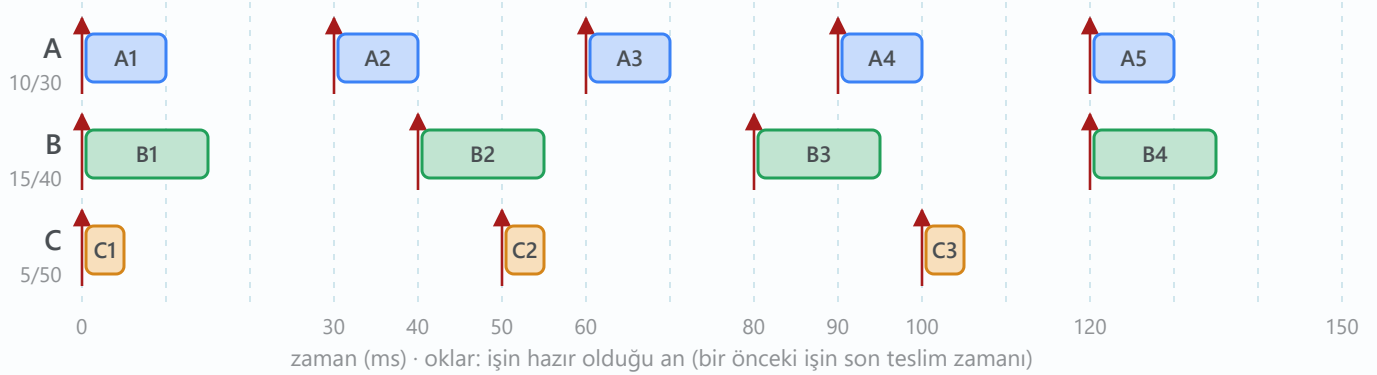
Kabul denetimi

Yeni akış ancak $N \cdot c \leq T$ ise kabul edilir (c: kare başına süre, T: periyot). Aksi hâlde tüm akışlar bozulur.

Gerçekte akışların hızları ve kare maliyetleri farklıdır (I/P/B, farklı çözünürlük); bu yüzden genel gerçek zamanlı çizelgeleme gerekir.

SÜREÇ ÇİZELGELEME

Genel Gerçek Zamanlı Çizelgeleme



Süreç	Film	Periyot P	CPU / kare C	C / P
A	33,3 fps → 30 ms (örnek)	30 ms	10 ms	0,333
B	25 fps	40 ms	15 ms	0,375
C	20 fps	50 ms	5 ms	0,100
			Toplam U	0,808

Çizelgelenebilirlik için gerekli koşul: $U = \sum C_i / P_i \leq 1$. Her karenin, bir sonraki kare gelmeden (periyot sonunda) bitmesi gerekir.

Hız Monoton Çizelgeleme (RMS)

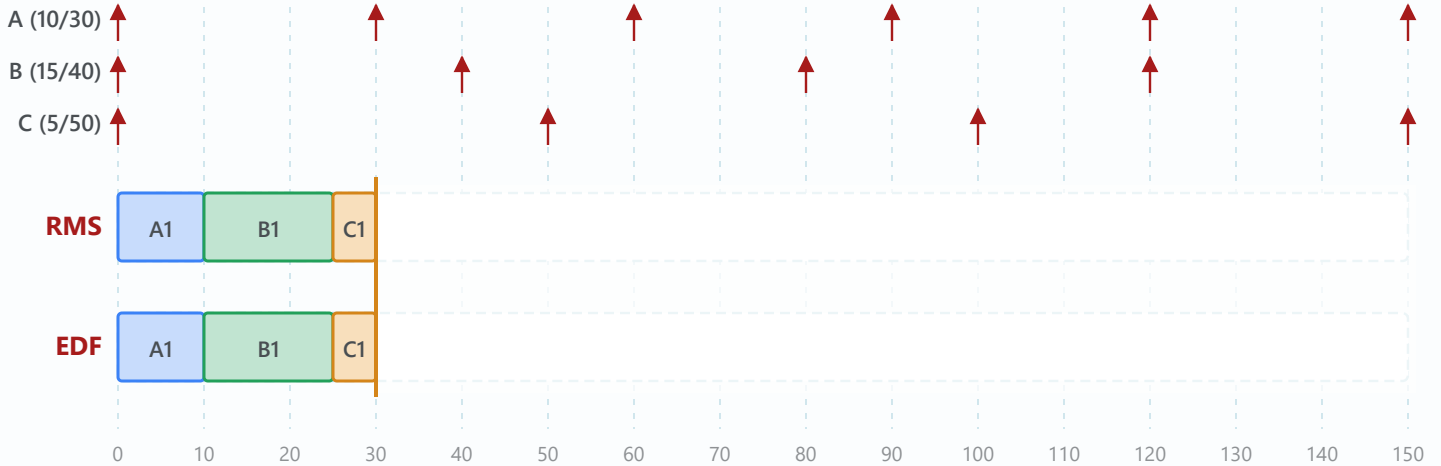
- Statik öncelik: **öncelik** $\propto 1 / \text{periyot}$ (sık çalışan süreç önceliklidir); kesmeli (preemptive).
- Varsayımlar: her süreç periyodu içinde bitmeli; süreçler birbirine bağımlı değil; her işlem patlaması aynı CPU süresini ister; periyodik olmayan işlerin son teslim zamanı yok; kesme ek maliyetsiz.
- Liu ve Layland (1973): $U \leq m(2^{1/m} - 1)$ ise RMS kesin başarılıdır. Bu *yeterli* koşuldur, gerekli değildir.
- Statik öncelikli süreçler arasında en iyisidir; uygulaması kolay (sabit öncelikli çizelgeleyici, ö. SCHED_FIFO).

m	sınır $m(2^{1/m} - 1)$
1	1,000
2	0,828
3	0,780
4	0,757
5	0,743
10	0,718
∞	$\ln 2 \approx 0,693$

Örneğimizde $U = 0,808 > 0,780$ ($m = 3$): sınır bir şey garanti etmez; çizelgeyi çizip bakmak gerekir. Bu örnekte RMS yine de başarır.

SÜREÇ ÇİZELGELEME

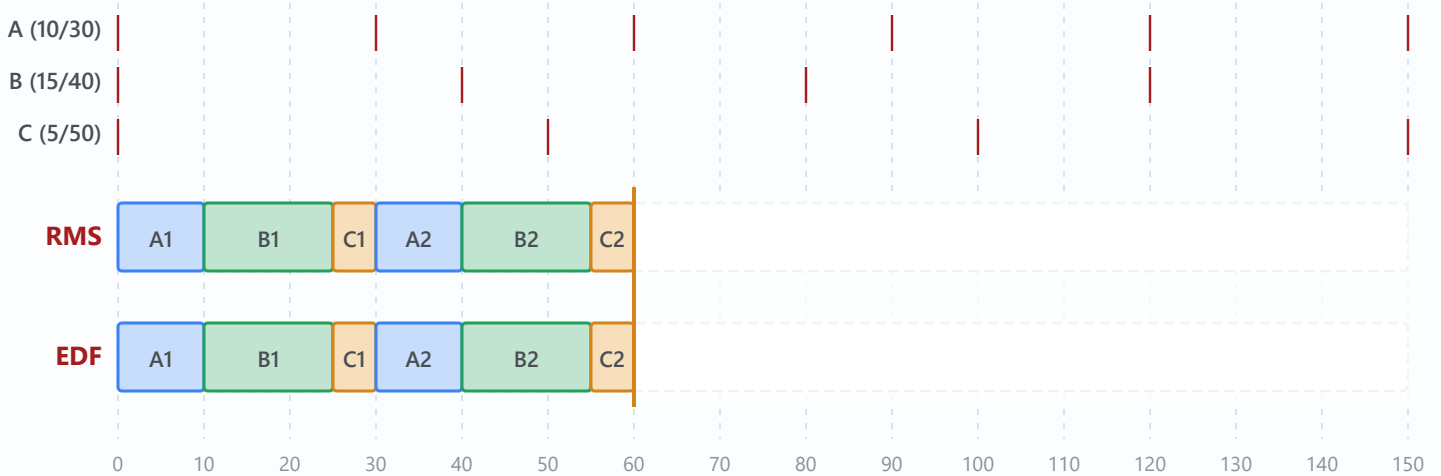
RMS ve EDF Örneği 1 ($U = 0,808$)



$t = 0$: uç süreç birden hazır. RMS önceliği periyoda göre $A > B > C$; EDF teslim zamanlarına (30, 40, 50) bakar: ikisi de A1, B1, C1 sırasını seçer.

SÜREÇ ÇİZELGELEME

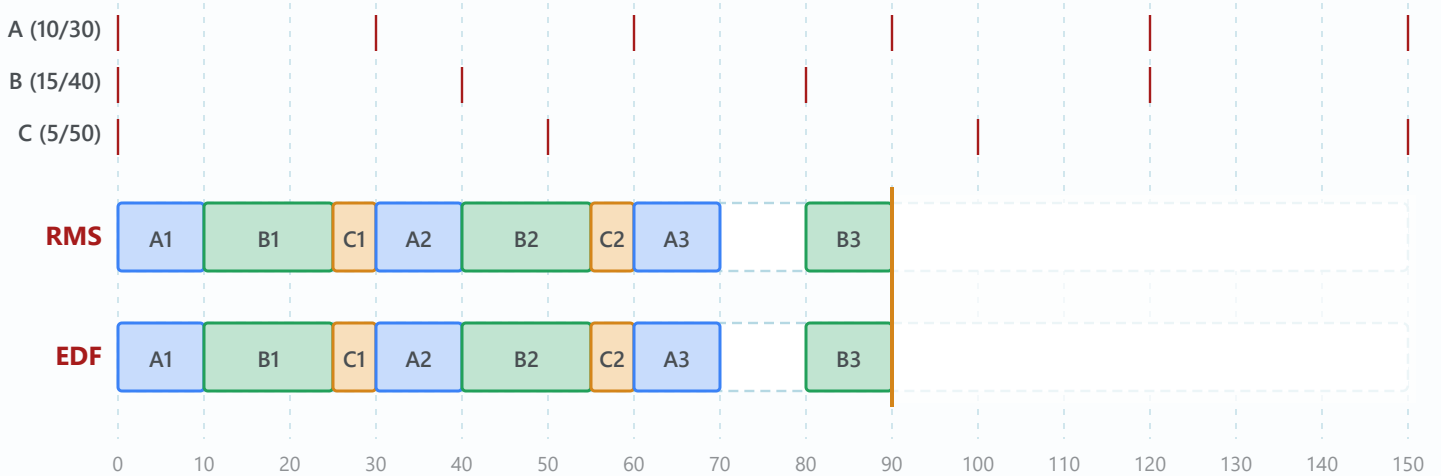
RMS ve EDF Örneği 1 ($U = 0,808$)



$t = 30-60$: A2 (teslim 60) gelir, A2 çalışır; $t = 40$ 'ta B2 hazır olur (teslim 80), $t = 50$ 'de C2 (teslim 100). İki çizelge hâlâ aynı.

SÜREÇ ÇİZELGELEME

RMS ve EDF Örneği 1 ($U = 0,808$)



$t = 60-90$: A3 çalışır; ardından B3 hazır olmayana kadar ($t = 80$) işlemci boştur; B3 $t = 80$ 'de başlar.

SÜREÇ ÇİZELGELEME

RMS ve EDF Örneği 1 ($U = 0,808$)



t = 90: A4 hazır. **RMS** B3'ü keser (A daha öncelikli). **EDF** B3'ün teslimi (120) A4'ünkinden (120) geç değil; eşitlikte çalışanı sürdürür, A4 bekler.

RMS ve EDF Örneği 1 ($U = 0,808$)



$t = 150$ 'ye kadar iki algoritma da hiçbir teslim zamanını kaçırmaz. $U = 0,808 > 0,780$ olmasına rağmen RMS başardı: sınır yalnızca yeterli koşuldur.

RMS Kabul Denetimi: Kod İzleme

```
► double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)



kabul edilen: m = 0, U = 0

Video sunucusu RMS ile çizelgeliyor ve yeni bir akışı ancak Liu–Layland sınırı bir garanti verirse kabul ediyor. Başta hiç akış yok: U = 0, m = 0.

Adım 1 / 11

DEĞİŞKENLER

U = 0

m = 0

ÇIKTI



RMS Kabul Denetimi: Kod İzleme

```

1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
▶   double Uy = U + C / P;
▶   int my = m + 1;
▶   double sinir = my * (pow(2, 1.0 / my) - 1);
6   if (Uy > sinir)
7       return 0; /* garanti yok: reddet */
8   U = Uy; m = my;
9   return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */

```

kullanım U (0 ... 1)



kabul edilen: $m = 0$, $U = 0$

aday S1: $C/P = 10/40 = 0,25$

S1 ($C = 10$, $P = 40$) için yeni kullanım $U_y = 0 + 0,25 = 0,25$; 1 akış için sınır $1(2^{1/1} - 1) = 1$.

Adım 2 / 11

DEĞİŞKENLER

aday = S1

$U_y = 0,25$

$my = 1$

sınır = 1

ÇIKTI



RMS Kabul Denetimi: Kod İzleme

```
1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)

kabul edilen: $m = 1$, $U = 0,25$

$0,25 \leq 1$: **kabul**. $U = 0,25$, $m = 1$; tüm akışlar teslim zamanlarını kaçırmaz.

Adım 3 / 11

DEĞİŞKENLER

aday = S1

U = 0,25

m = 1

ÇIKTI

S1: kabul

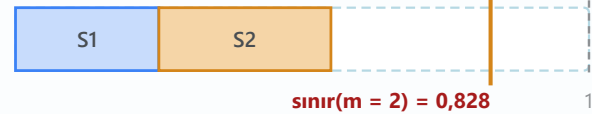
RMS Kabul Denetimi: Kod İzleme

```

1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */

```

kullanım U (0 ... 1)



kabul edilen: $m = 1$, $U = 0,25$

aday S2: $C/P = 15/50 = 0,3$

S2 ($C = 15$, $P = 50$) için yeni kullanım $U_y = 0,25 + 0,3 = 0,55$; 2 akış için sınır $2(2^{(1/2)} - 1) = 0,828$.

Adım 4 / 11

DEĞİŞKENLER

aday = S2

$U_y = 0,55$

$my = 2$

sınır = 0,828

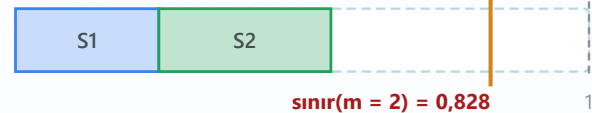
ÇIKTI

S1: kabul

RMS Kabul Denetimi: Kod İzleme

```
1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)



kabul edilen: m = 2, U = 0,55

$0,55 \leq 0,828$: **kabul**. U = 0,55, m = 2; tüm akışlar teslim zamanlarını kaçırmaz.

Adım 5 / 11

DEĞİŞKENLER

aday = S2

U = 0,55

m = 2

ÇIKTI

S1: kabul

S2: kabul

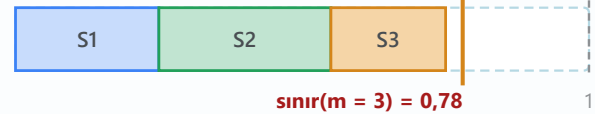
RMS Kabul Denetimi: Kod İzleme

```

1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
    ▶ double Uy = U + C / P;
    ▶ int my = m + 1;
    ▶ double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */

```

kullanım U (0 ... 1)



kabul edilen: m = 2, U = 0,55

aday S3: C/P = 20/100 = 0,2

S3 (C = 20, P = 100) için yeni kullanım $U_y = 0,55 + 0,2 = 0,75$; 3 akış için sınır $3(2^{1/3} - 1) = 0,78$.

Adım 6 / 11

DEĞİŞKENLER

aday = S3

$U_y = 0,75$

my = 3

sınır = 0,78

ÇIKTI

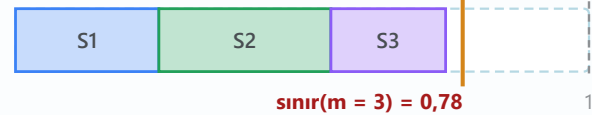
S1: kabul

S2: kabul

RMS Kabul Denetimi: Kod İzleme

```
1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)



kabul edilen: m = 3, U = 0,75

$0,75 \leq 0,78$: **kabul**. U = 0,75, m = 3; tüm akışlar teslim zamanlarını kaçırmaz.

Adım 7 / 11

DEĞİŞKENLER

aday = S3

U = 0,75

m = 3

ÇIKTI

S1: kabul
S2: kabul
S3: kabul

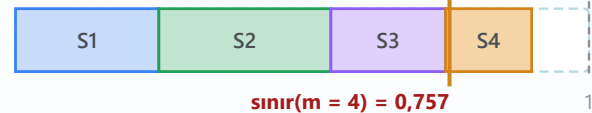
RMS Kabul Denetimi: Kod İzleme

```

1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
▶   double Uy = U + C / P;
▶   int my = m + 1;
▶   double sinir = my * (pow(2, 1.0 / my) - 1);
6   if (Uy > sinir)
7       return 0; /* garanti yok: reddet */
8   U = Uy; m = my;
9   return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */

```

kullanım U (0 ... 1)



kabul edilen: $m = 3$, $U = 0,75$

aday S4: $C/P = 30/200 = 0,15$

S4 ($C = 30$, $P = 200$) için yeni kullanım $U_y = 0,75 + 0,15 = 0,9$; 4 akış için sınır $4(2^{1/4} - 1) = 0,757$.

Adım 8 / 11

DEĞİŞKENLER

aday = S4

$U_y = 0,9$

my = 4

sınır = 0,757

ÇIKTI

S1: kabul

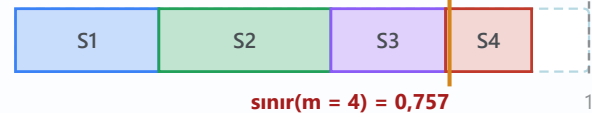
S2: kabul

S3: kabul

RMS Kabul Denetimi: Kod İzleme

```
1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)



kabul edilen: m = 3, U = 0,75

aday S4: C/P = 30/200 = 0,15

0,9 > 0,757: sınır garanti vermiyor, **ret**. U ve m değişmez. Toplam 1'in altında olsa bile RMS için güvence yok; EDF kullansaydık $U \leq 1$ yeterdi.

Adım 9 / 11

DEĞİŞKENLER

aday = S4

U = 0,75

m = 3

ÇIKTI

```
S1: kabul
S2: kabul
S3: kabul
S4: ret
```

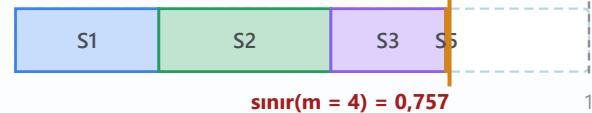
RMS Kabul Denetimi: Kod İzleme

```

1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */

```

kullanım U (0 ... 1)



kabul edilen: m = 3, U = 0,75

aday S5: C/P = 1/250 = 0,004

S5 (C = 1, P = 250) için yeni kullanım $U_y = 0,75 + 0,004 = \mathbf{0,754}$; 4 akış için sınır $4(2^{1/4} - 1) = \mathbf{0,757}$.

Adım 10 / 11

DEĞİŞKENLER

aday = S5

$U_y = 0,754$

my = 4

sınır = 0,757

ÇIKTI

```

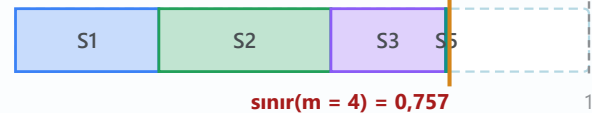
S1: kabul
S2: kabul
S3: kabul
S4: ret

```

RMS Kabul Denetimi: Kod İzleme

```
1 double U = 0; int m = 0; /* kabul edilenler */
2 int kabul(double C, double P) {
3     double Uy = U + C / P;
4     int my = m + 1;
5     double sinir = my * (pow(2, 1.0 / my) - 1);
6     if (Uy > sinir)
7         return 0; /* garanti yok: reddet */
8     U = Uy; m = my;
9     return 1;
10 }
11 /* S1(10,40) S2(15,50) S3(20,100) */
12 /* S4(30,200) S5(1,250) */
```

kullanım U (0 ... 1)



kabul edilen: m = 4, U = 0,754

$0,754 \leq 0,757$: **kabul**. U = 0,754, m = 4; tüm akışlar teslim zamanlarını kaçırmaz. Küçük S5 sığdı, büyük S4 sığmamıştı.

Adım 11 / 11

DEĞİŞKENLER

aday = S5

U = 0,754

m = 4

ÇIKTI

```
S1: kabul
S2: kabul
S3: kabul
S4: ret
S5: kabul
```

En Erken Teslim Zamanı Önce (EDF)

Kural

Dinamik öncelik: hazır işler arasında **son teslim zamanı en yakın olan** çalışır. Yeni iş geldiğinde öncelikler yeniden değerlendirilir; kesmelidir.

Optimallik

Tek işlemcide, bağımsız ve kesilebilir işler için $U \leq 1$ ise EDF her zaman başarılıdır: %100 kullanım mümkündür (RMS'de $\approx$ %69–78).

Bedel

Her olayda sıralama gerekir (öncelik kuyruğu, $O(\log n)$). Aşırı yükte ($U > 1$) hangi işin kaçıracağı öngörülemez; "domino etkisi".

Bugün

Linux `SCHED_DEADLINE` (EDF + sabit bant sunucusu, çalışma/periyot ile kabul denetimi); ses için JACK/PipeWire `SCHED_FIFO`; Windows MMCSS; RTOS'larda RMS yaygın.

hazır kuyruk:

A2 · teslim 60

C2 · teslim 70

B2 · teslim 80

A3 · teslim 90

← C2 yeni geldi (teslim 70):
A2'den sonra, B2'den önce

baştaki iş çalışır; kuyruk öncelik kuyruğu (öbek) ile $O(\log n)$

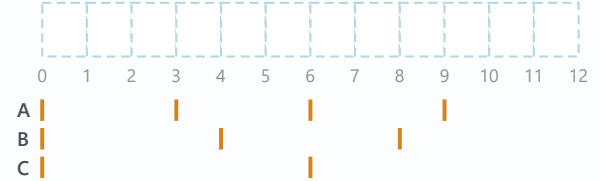
EDF ve RMS analizlerinin ikisi de bağımsız, kaynak paylaşmayan işler varsayar; paylaşımda öncelik tavanı/kalıtımı gerekir.

EDF Çizelgeleyici: Kod İzleme

```

▶ /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
2 for (int t = 0; t < 12; t++) {
3     for (int i = 0; i < n; i++)
4         if (t % P[i] == 0) {          /* yeni iş */
5             kalan[i] = C[i];
6             teslim[i] = t + P[i];
7         }
8     int s = -1;
9     for (int i = 0; i < n; i++)
10        if (kalan[i] > 0 &&
11            (s < 0 || teslim[i] < teslim[s]))
12            s = i;
13    if (s >= 0) kalan[s]--;          /* 1 ms */
14 }

```



Üç periyodik iş, $U = 1/3 + 1/4 + 2/6 = 0,917 \leq 1$. Ortak periyot 12 ms; her milisaniye bir karar verilir. Oklar iş gelişlerini gösterir.

Adım 1 / 10

DEĞİŞKENLER

$U = 0,917$

ÇIKTI



EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```

▶ for (int t = 0; t < 12; t++) {
  ▶ for (int i = 0; i < n; i++)
    ▶ if (t % P[i] == 0) {          /* yeni iş */
      ▶ kalan[i] = C[i];
      ▶ teslim[i] = t + P[i];
    }
  ▶ int s = -1;
  ▶ for (int i = 0; i < n; i++)
    ▶ if (kalan[i] > 0 &&
      ▶ (s < 0 || teslim[i] < teslim[s]))
      ▶ s = i;
  ▶ if (s >= 0) kalan[s]--;        /* 1 ms */
}
14 }
```



Adım 2 / 10

DEĞİŞKENLER

t = 0

kalan = [0,1,2]

teslim = [3,4,6]

s = A

ÇIKTI



EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```
▶ for (int t = 0; t < 12; t++) {
3   for (int i = 0; i < n; i++)
4     if (t % P[i] == 0) {      /* yeni iş */
5       kalan[i] = C[i];
6       teslim[i] = t + P[i];
7     }
▶   int s = -1;
▶   for (int i = 0; i < n; i++)
▶     if (kalan[i] > 0 &&
▶         (s < 0 || teslim[i] < teslim[s]))
▶       s = i;
▶   if (s >= 0) kalan[s]--;      /* 1 ms */
14 }
```



Hazır: B, C; teslim zamanları B 4, C 6 → en yakını **B** çalışır.

Adım 3 / 10

DEĞİŞKENLER

t = 1

kalan = [0, 0, 2]

teslim = [3, 4, 6]

s = B

ÇIKTI



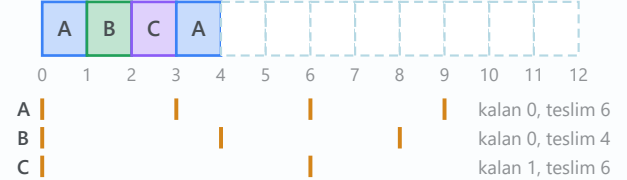
EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```

▶ for (int t = 0; t < 12; t++) {
  ▶ for (int i = 0; i < n; i++)
    ▶ if (t % P[i] == 0) {      /* yeni iş */
      ▶ kalan[i] = C[i];
      ▶ teslim[i] = t + P[i];
7    }
  ▶ int s = -1;
  ▶ for (int i = 0; i < n; i++)
    ▶ if (kalan[i] > 0 &&
      ▶ (s < 0 || teslim[i] < teslim[s]))
      ▶ s = i;
  ▶ if (s >= 0) kalan[s]--;    /* 1 ms */
14 }

```



t = 2–3: C → A. Gelen: t = 3’de A. Her milisaniyede en yakın teslimli hazır iş seçilir.

Adım 4 / 10

DEĞİŞKENLER

t = 3

kalan = [0,0,1]

teslim = [6,4,6]

s = A

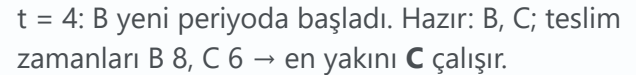
ÇIKTI



```

> for (int t = 0; t < 12; t++) {
>     for (int i = 0; i < n; i++)
>         if (t % P[i] == 0) {           /* yeni iş */
>             kalan[i] = C[i];
>             teslim[i] = t + P[i];
7         }
>     int s = -1;
>     for (int i = 0; i < n; i++)
>         if (kalan[i] > 0 &&
>             (s < 0 || teslim[i] < teslim[s]))
>             s = i;
>     if (s >= 0) kalan[s]--;           /* 1 ms */
14 }

```



Adım 5 / 10

DEĞİŞKENLER

kalan = [0, 1, 0]

$$S = C$$

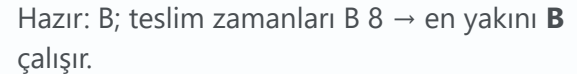
ÇIKTI



```

▶ for (int t = 0; t < 12; t++) {
3     for (int i = 0; i < n; i++)
4         if (t % P[i] == 0) {           /* yeni iş */
5             kalan[i] = C[i];
6             teslim[i] = t + P[i];
7         }
▶     int s = -1;
▶     for (int i = 0; i < n; i++)
▶         if (kalan[i] > 0 &&
▶             (s < 0 || teslim[i] < teslim[s]))
▶             s = i;
▶     if (s >= 0) kalan[s]--;           /* 1 ms */
14 }

```



Adım 6 / 10

DEĞİŞKENLER

 $t = 5$

```
kalan = [0, 0, 0]
```

```
teslim = [6, 8, 6]
```

$$S = B$$

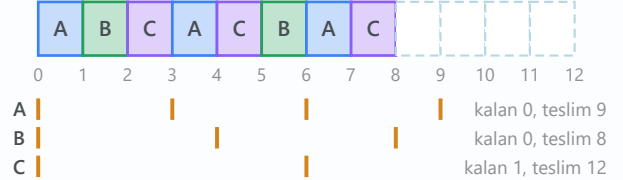
ÇIKTI



EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```
▶ for (int t = 0; t < 12; t++) {
3   for (int i = 0; i < n; i++)
4     if (t % P[i] == 0) {      /* yeni iş */
5       kalan[i] = C[i];
6       teslim[i] = t + P[i];
7     }
▶   int s = -1;
▶   for (int i = 0; i < n; i++)
▶     if (kalan[i] > 0 &&
▶       (s < 0 || teslim[i] < teslim[s]))
▶       s = i;
▶   if (s >= 0) kalan[s]--;    /* 1 ms */
14 }
```



t = 6–7: A → C. Gelen: t = 6'de A, C. Her milisaniyede en yakın teslimli hazır iş seçilir.

Adım 7 / 10

DEĞİŞKENLER

t = 7

kalan = [0,0,1]

teslim = [9,8,12]

s = C

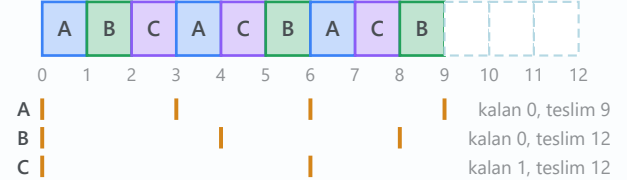
ÇIKTI



EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```
▶ for (int t = 0; t < 12; t++) {
  ▶ for (int i = 0; i < n; i++)
    ▶ if (t % P[i] == 0) {      /* yeni iş */
      ▶ kalan[i] = C[i];
      ▶ teslim[i] = t + P[i];
    }
  ▶ int s = -1;
  ▶ for (int i = 0; i < n; i++)
    ▶ if (kalan[i] > 0 &&
      ▶ (s < 0 || teslim[i] < teslim[s]))
      ▶ s = i;
  ▶ if (s >= 0) kalan[s]--;      /* 1 ms */
14 }
```



t = 8: B yeni periyoda başladı. Hazır: B, C; teslim zamanları B 12, C 12 → en yakını **B** çalışır.

Adım 8 / 10

DEĞİŞKENLER

t = 8

kalan = [0,0,1]

teslim = [9,12,12]

s = B

ÇIKTI



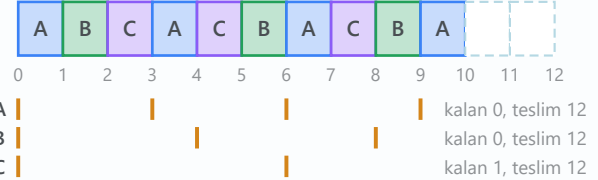
EDF Çizelgeleyici: Kod İzleme

```
1 /* A(P=3,C=1) B(P=4,C=1) C(P=6,C=2) */
```

```

▶ for (int t = 0; t < 12; t++) {
  ▶ for (int i = 0; i < n; i++)
    ▶ if (t % P[i] == 0) {      /* yeni iş */
      ▶ kalan[i] = C[i];
      ▶ teslim[i] = t + P[i];
7    }
  ▶ int s = -1;
  ▶ for (int i = 0; i < n; i++)
    ▶ if (kalan[i] > 0 &&
      ▶ (s < 0 || teslim[i] < teslim[s]))
      ▶ s = i;
  ▶ if (s >= 0) kalan[s]--;    /* 1 ms */
14 }

```



t = 9: A yeni periyoda başladı. Hazır: A, C;
 teslim zamanları A 12, C 12 → en yakını **A**
 çalışır.

Adım 9 / 10

DEĞİŞKENLER

t = 9

kalan = [0,0,1]

teslim = [12,12,12]

s = A

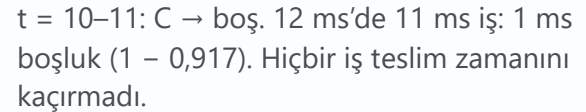
ÇIKTI



```

▶ for (int t = 0; t < 12; t++) {
3     for (int i = 0; i < n; i++)
4         if (t % P[i] == 0) {           /* yeni iş */
5             kalan[i] = C[i];
6             teslim[i] = t + P[i];
7         }
▶     int s = -1;
▶     for (int i = 0; i < n; i++)
▶         if (kalan[i] > 0 &&
▶             (s < 0 || teslim[i] < teslim[s]))
▶             s = i;
▶     if (s >= 0) kalan[s]--;           /* 1 ms */
14 }

```



DEĞİŞKENLER

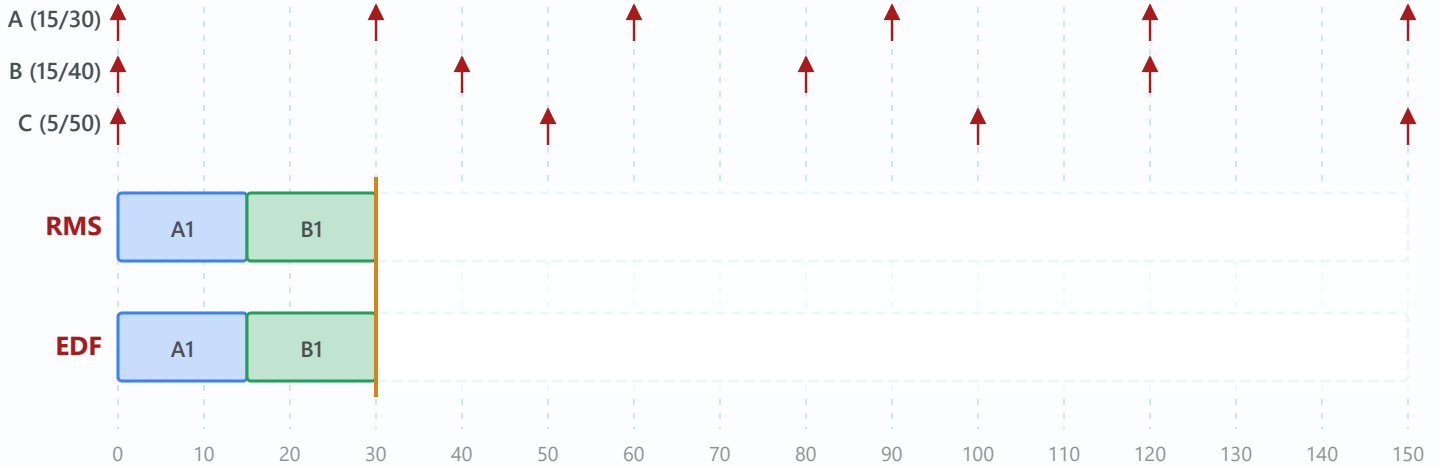
```
teslim = [12,12,12] s = -1
```

ÇIKTI



SÜREÇ ÇİZELGELEME

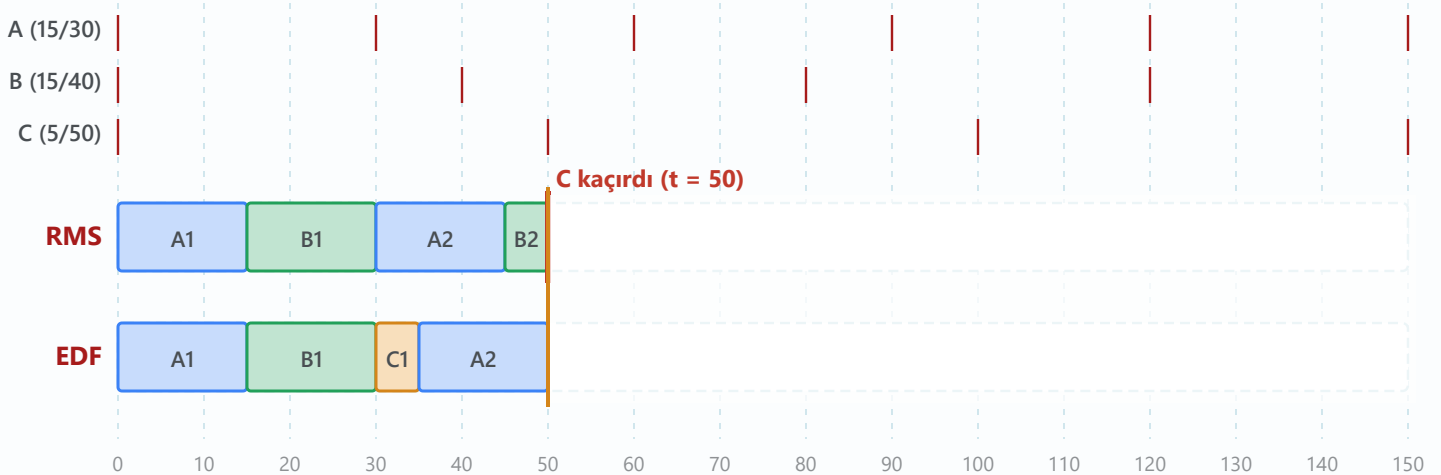
RMS Başarısız, EDF Başarılı (U = 0,975)



A'nın kare başına süresi 15 ms'ye çıktı: $U = 0,5 + 0,375 + 0,1 = \mathbf{0,975}$. $t = 0-30$ aralığında iki çizelge aynı: A1, B1.

SÜREÇ ÇİZELGELEME

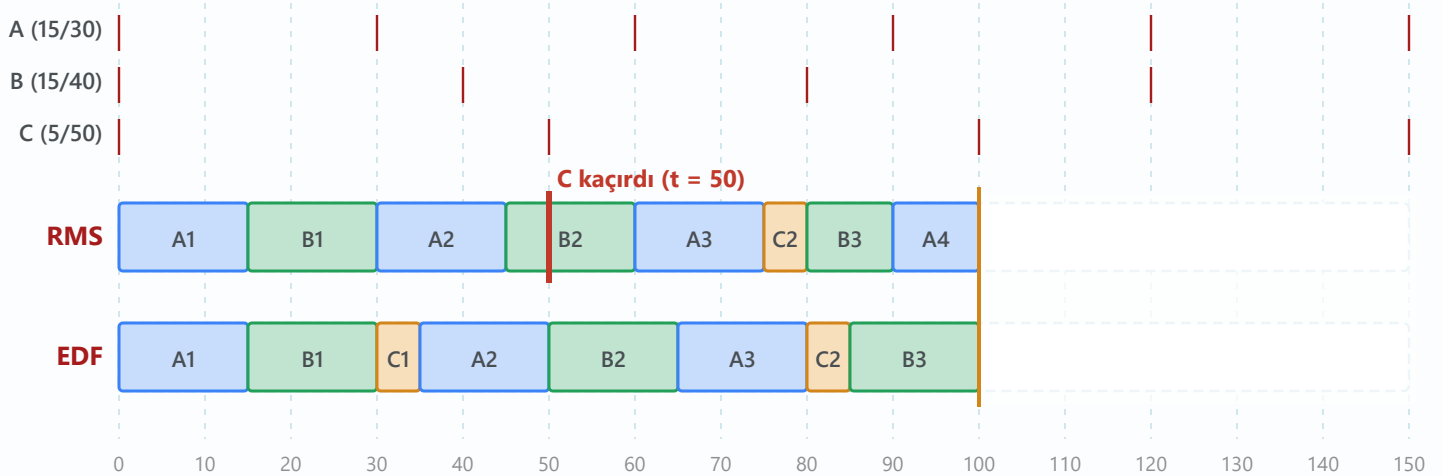
RMS Başarısız, EDF Başarılı (U = 0,975)



t = 30: RMS A2'yi C1'e tercih eder (A daha öncelikli). C1 hiç çalışmadan **t = 50'de teslim zamanını kaçıır**. EDF ise C1'i (teslim 50) A2'den (teslim 60) önce çalıştırır.

SÜREÇ ÇİZELGELEME

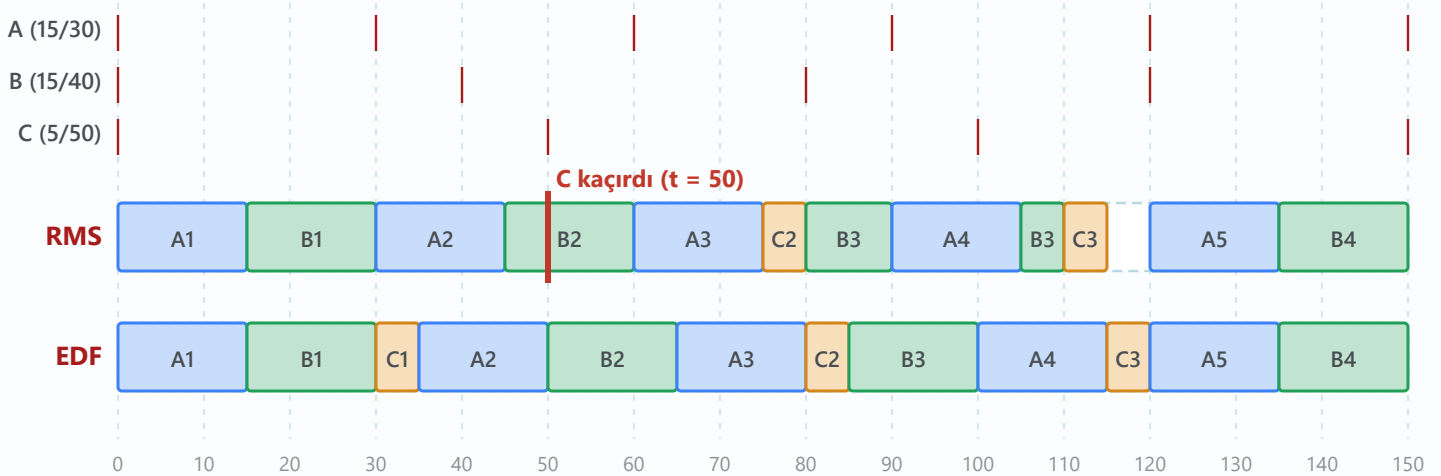
RMS Başarısız, EDF Başarılı (U = 0,975)



$t = 50-100$: EDF teslim zamanlarına göre B2 (80), A3 (90), C2 (100), B3 (120) sırasıyla devam eder; hiçbir iş geç kalmaz.

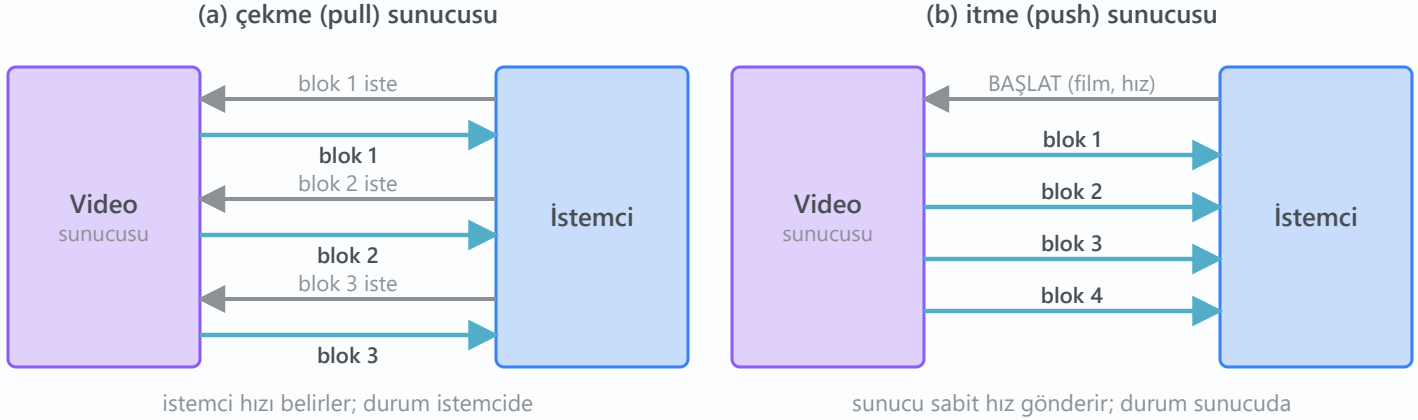
SÜREÇ ÇİZELGELEME

RMS Başarısız, EDF Başarılı (U = 0,975)



t = 150'ye kadar EDF hiçbir teslim zamanını kaçırmaz ($U \leq 1$). RMS ise ilk 50 ms içinde başarısız oldu: statik öncelik bu yükü taşıyamaz.

Çoklu Ortam Dosya Sistemi Paradigmaları



Klasik video sunucuları (ve RTP/çoklu yayın) itme modelini kullanır. İlginç biçimde günümüzün HLS/DASH akışı **çekme** modelidir: oynatıcı bölütleri sıradan HTTP GET ile ister; durum ve hız kararı istemcidedir, sunucu durumsuz ve önbelleklenebilir kalır.

VCR Kontrol İşlevleri

Duraklat

İtme sunucusunda sunucu akışı durdurmalı ve nerede kaldığını hatırlamalıdır; ayrılan disk ve ağ kaynakları boşa kalır, kabul denetimi bunu hesaba katmalı.

Geri sar / ileri sar

Ham olarak 4x hızda okumak 4x bant genişliği ister. P/B kareleri atlamak mümkün değildir (referanslara bağlılar); çözüm yalnız I kareleri göstermek ya da ayrı bir hızlı sarma sürümü tutmaktır.

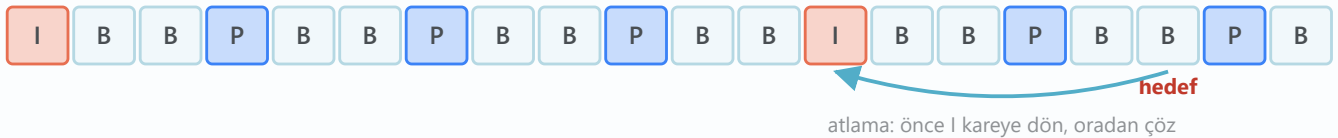
Atlama (seek)

Rastgele konuma gitmek en yakın I kareye (GOP başına) inmeyi gerektirir; dosya bu yüzden kare/blok indisi tutar.

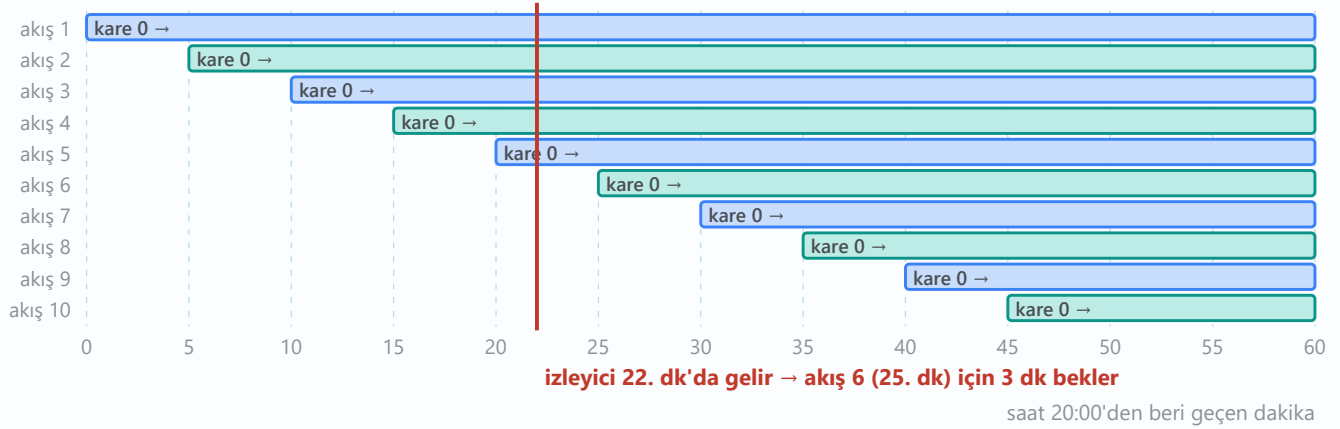
Bugünkü çözüm

HLS/DASH'te sarma, istemcinin başka bölütü istemesidir. Önizleme için yalnız I karelik oynatma listeleri (EXT-X-I-FRAMES-ONLY) ve küçük resim (thumbnail) dosyaları kullanılır.

kareler (görüntüleme sırası)



Yakın İsteğe Bağlı Video (NVOD)



Kademeli başlangıç

Film sabit aralıkla ($\Delta = 5$ dk) yeniden başlatılır. 30 fps'de 5 dk = **9.000 kare**: her akış bir öncekinin 9.000 kare gerisindedir.

Akış ve bant genişliği

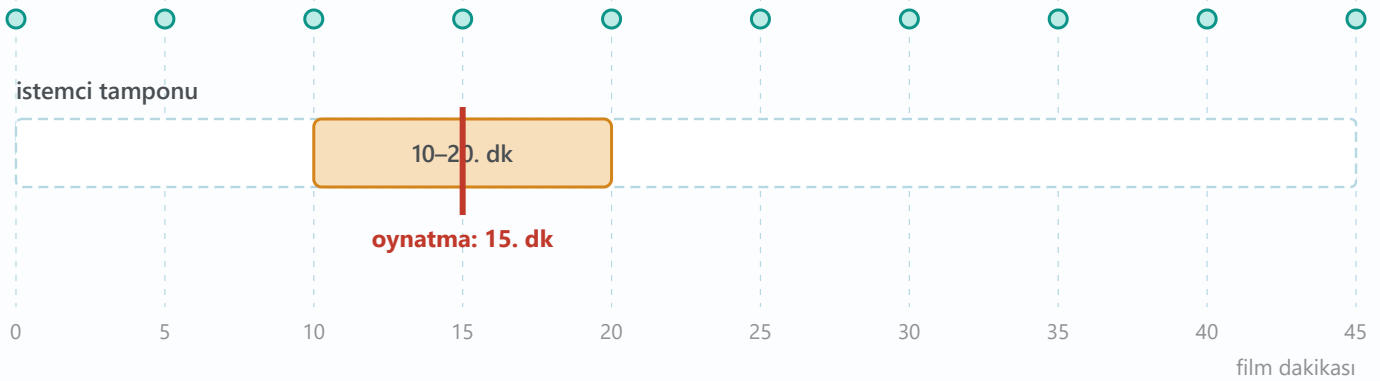
120 dk'lık film → $120 / 5 = 24$ **akış**, çoklu yayınla (multicast) sınırsız izleyiciye: 24×8 Mbit/s = 192 Mbit/s. Gerçek VOD'da 1.000 izleyici = 8 Gbit/s.

Bekleme

Ortalama $\Delta / 2 = 2,5$ dk, en kötü 5 dk. Kablo TV'deki "her çeyrek saatte başlayan film" bu modeldi.

NVOD ile VCR İşlevleri

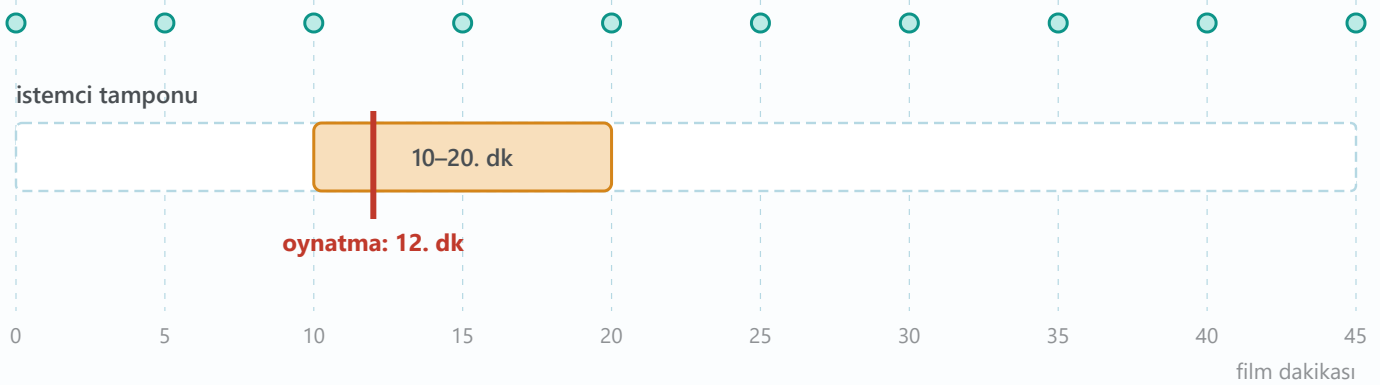
yayındaki akışların konumu (t = 0 dk):



(a) İzleyici 15. dakikada. İstemci 5 dk geriye ve 5 dk ileriye tamponda tutar (10–20); ileri kısmı, 5 dk önce başlamış akışı kaydederek doldurur. İki akışı birden alır.

NVOD ile VCR İşlevleri

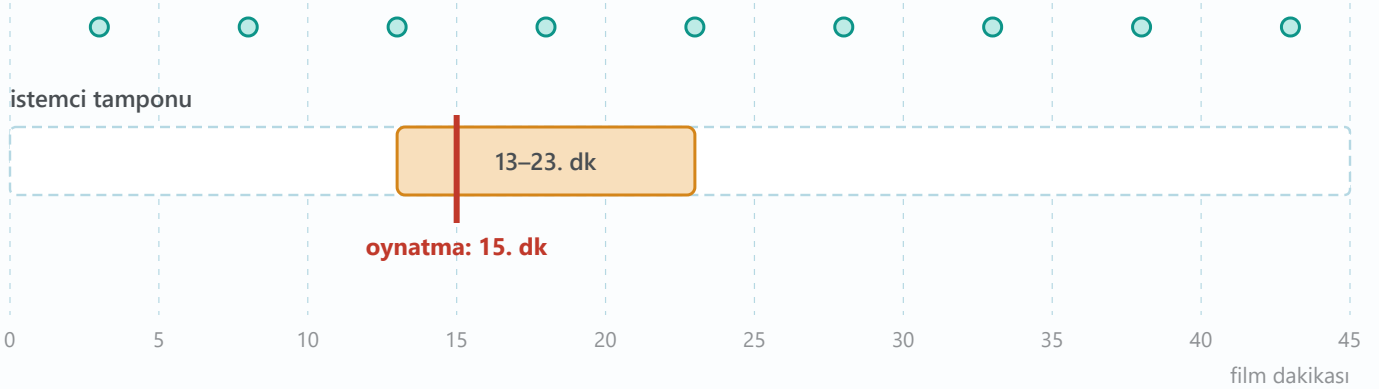
yayındaki akışların konumu ($t = 0$ dk):



(b) 12. dakikaya **geri sarma**: tampon içinde, anında. Sunucunun haberi bile olmaz.

NVOD ile VCR İşlevleri

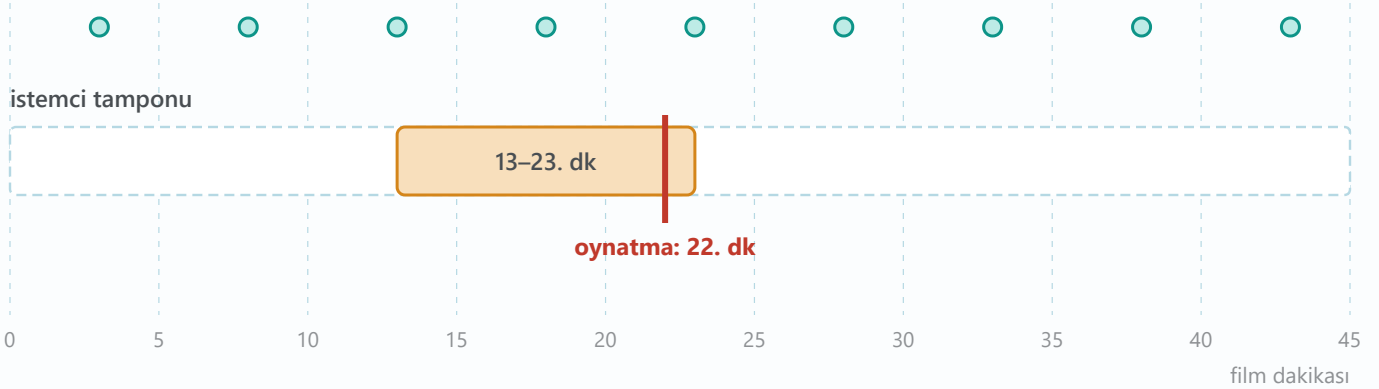
yayındaki akışların konumu ($t = 3$ dk):



(c) 3 dk sonra oynatma 15. dakikada; ileri akış 23'e ulaştı. Tampon 10 dk sığıdığından en eski kısım atıldı: 13-23.

NVOD ile VCR İşlevleri

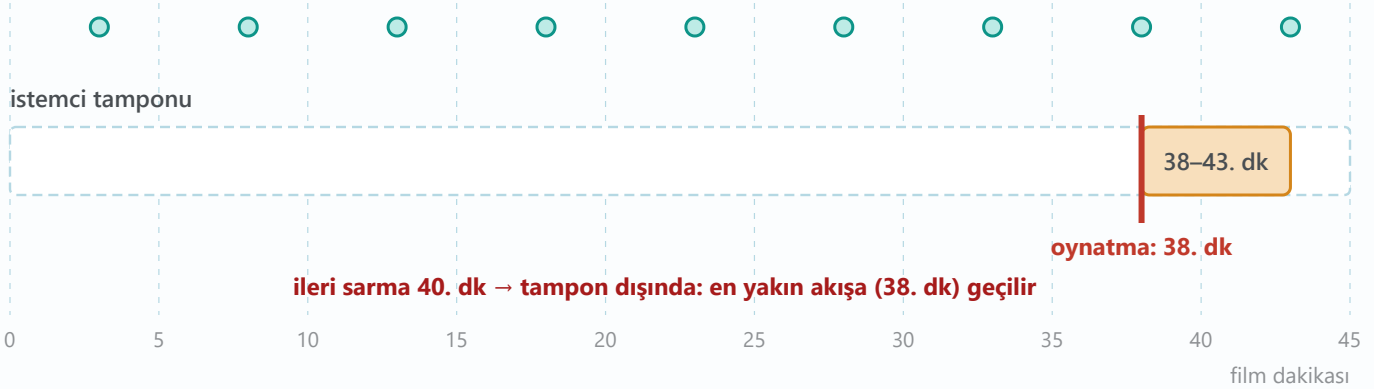
yayındaki akışların konumu ($t = 3$ dk):



(d) 22. dakikaya **ileri sarma**: yine tampon içinde, anında. Ancak ileri pay 1 dk'ya düştü; istemci daha önde bir akışı kaydetmeye başlar.

NVOD ile VCR İşlevleri

yayındaki akışların konumu ($t = 3$ dk):



(e) 40. dakikaya atlama tamponun dışında: istemci o an 38. dakikada olan akışa geçer (2 dk sapma). NVOD'da VCR işlevleri bu yüzden yaklaşıktır.

Dosyayı Tek Diske Yerleştirme

disk üzerinde ardışık sektörler →



çok dilli film: her karede 3 ses izi



gri: izleyicinin seçmediği dil → yine de okunur, bant genişliği boşa gider

V: video karesi · A: ses · T: altyazı

Avantaj

Bir kare için tek arama (seek): video, ses ve metin birlikte okunur; sıralı akış en verimli disk erişimidir.

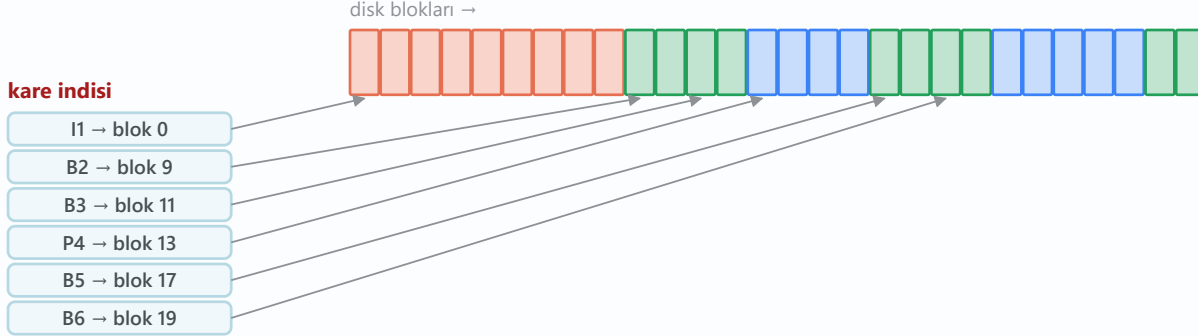
Dezavantaj

Dosya büyür ve birleştirmesi zordur; kullanılmayan izler okunur; değişken boyutlu kareler ($I > P > B$) yer ayırmayı karmaşılaştırır.

Bugünkü MP4/MKV kapsayıcıları da izleri parçalar hâlinde iç içe (interleaved) saklar; uyarlanabilir akışta ise her iz ve kalite ayrı bölüt dosyalarıdır.

Küçük ve Büyük Disk Blokları

Küçük bloklar (ör. 4 KB) + kare indisi (RAM'de)



Küçük bloklar + kare indisi: kare art arda küçük bloklara yazılır, her kare için bir indis girdisi. Disk israfı az, ama indis büyük: 2 saatlik film, 30 fps, 8 B/girdi → 216.000 girdi = 1,73 MB RAM.

Adım 1 / 2

Strateji	Bellek (indis)	Disk israfı	Arama
Kare indisi, küçük blok	yüksek (1,73 MB/film)	az	kare başına ≥ 1
Blok indisi, kareler bölünmez	düşük (57,6 KB/film)	blok sonunda $\approx \frac{1}{2}$ kare	blok başına 1
Blok indisi, kareler bölünür	düşük	yok	bölünen karede ek arama

Küçük ve Büyük Disk Blokları

Büyük bloklar (ör. 1 MB) + blok indisi



blok indisi: blok → ilk kare numarası

blok 0 → kare 1

blok 1 → kare 4

blok 2 → kare 8

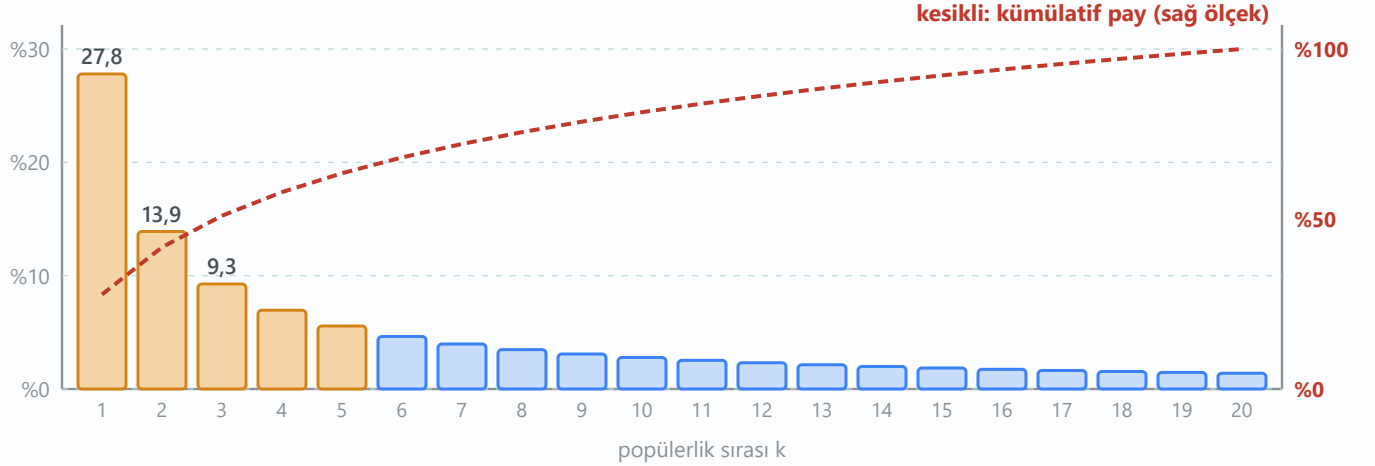
kareler bölünmezse blok sonunda boşluk; bölünürse boşluk yok ama ek arama
ortalama kare 33 KB (8 Mbit/s, 30 fps) → 1 MB blokta $\approx \frac{1}{2}$ kare $\approx \%1,7$ kayıp

Büyük bloklar + blok indisi: bir blokta birçok kare. İndis küçük: 1 MB blok ve 8 Mbit/s ile 7.200 blok $\times 8 \text{ B} = 57,6 \text{ KB}$. Kare bloklara bölünmezse israf, bölünürse ek arama.

Adım 2 / 2

Strateji	Bellek (indisi)	Disk israfı	Arama
Kare indisi, küçük blok	yüksek (1,73 MB/film)	az	kare başına ≥ 1
Blok indisi, kareler bölünmez	düşük (57,6 KB/film)	blok sonunda $\approx \frac{1}{2}$ kare	blok başına 1
Blok indisi, kareler bölünür	düşük	yok	bölünen karede ek arama

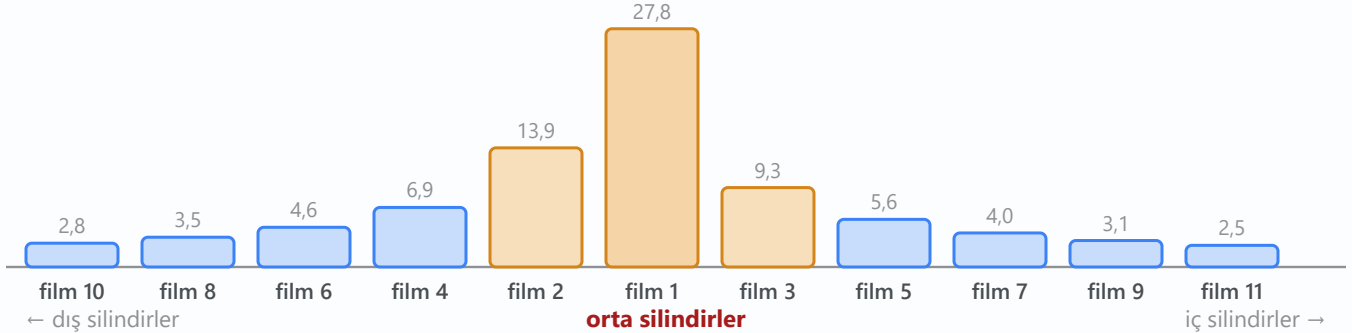
Zipf Yasası: Film Popülerliği



k. sıradaki film $P(k) = C / k$ olasılıkla seçilir; $C = 1 / H_N$. $N = 20$ için $H_{20} = 3,598$, $C = 0,278$: ilk film isteklerin %27,8'i, ilk 5 film %63,5'i. Gerçek video kataloglarında üs genelde 0,6–1 arasında ölçülür.

Organ Borusu Yerleşimi

yüzdelar: Zipf (N = 20) ile istek olasılığı



Fikir

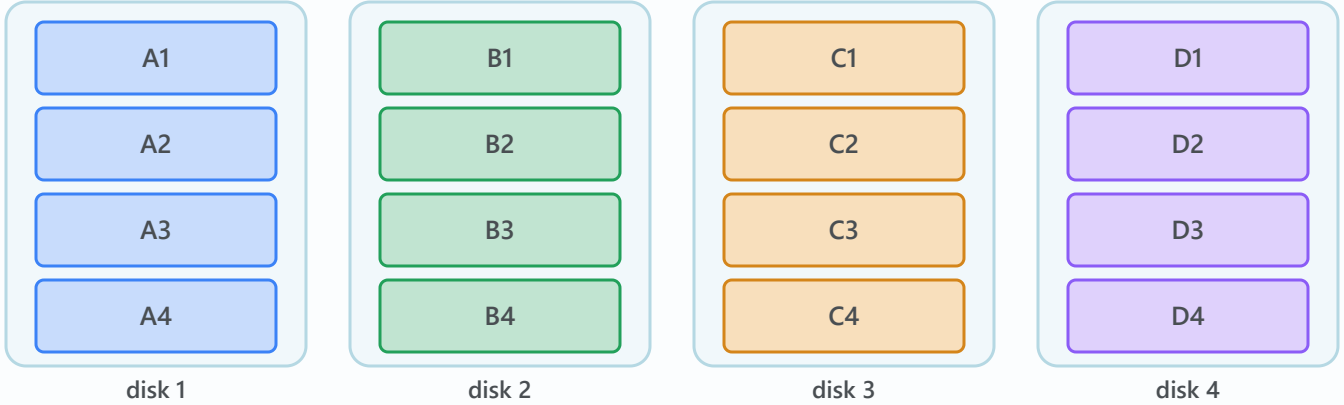
Kafa çoğu zaman popüler filmlerin üzerinde olacağından onları ortaya koymak ortalama arama mesafesini en aza indirir (Grossman ve Silverman). Histogram org borularına benzer.

Bugün

SSD'de arama yoktur, yerleşim önemsizleşir; ama aynı fikir katmanlara taşınır: popüler içerik RAM'de ve kenar SSD'lerinde, uzun kuyruk ucuz HDD/nesne deposunda.

Birden Çok Dosyayı Çoklu Diske Yerleştirme

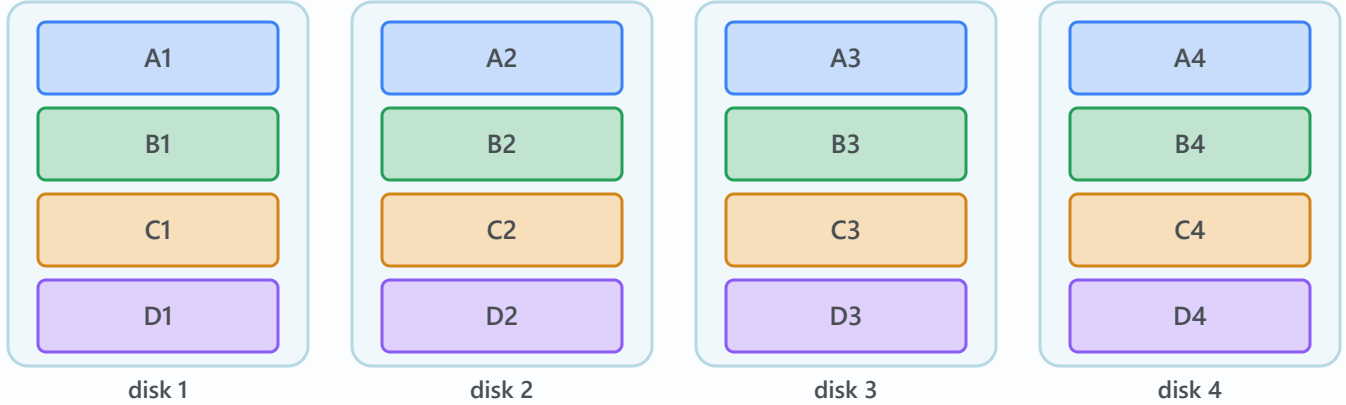
(a) şerit yok



(a) Her film tek diskte: basit, ama popüler film A'nın diski darboğaz olur, diğerleri boşta bekler.

Birden Çok Dosyayı Çoklu Diske Yerleştirme

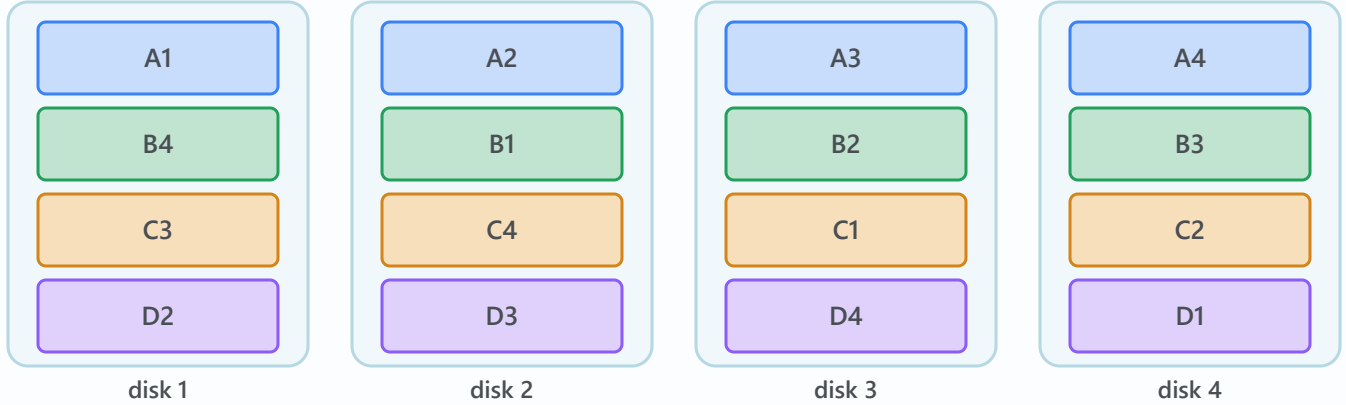
(b) aynı şeritleme



(b) Her filmin k. bloğu k. diske: yük dağılır. Ancak tüm filmler disk 1'den başladığından film başlangıçlarında disk 1 yığılır.

Birden Çok Dosyayı Çoklu Diske Yerleştirme

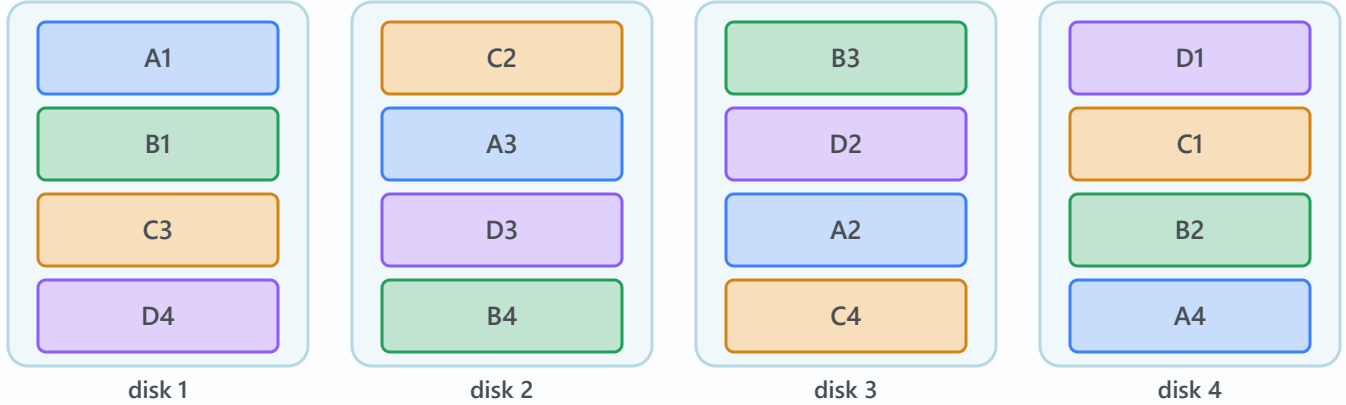
(c) kademeli şeritleme



(c) Kademeli: her film bir sonraki diskten başlar (A disk 1, B disk 2, ...); başlangıç yükü de dağılır.

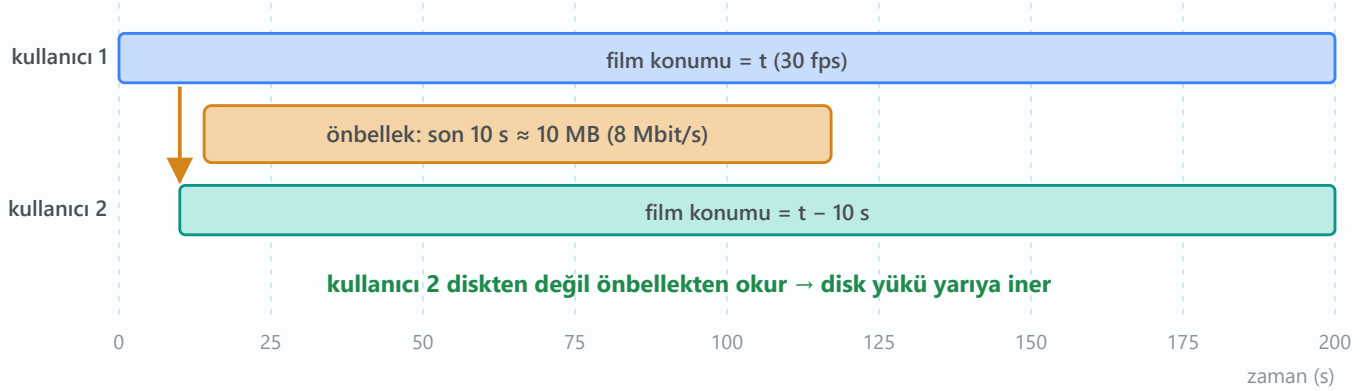
Birden Çok Dosyayı Çoklu Diske Yerleştirme

(d) rastgele şeritleme



(d) Rastgele: bloklar rastgele disklere; uzun vadede yük dengeli, yerleşim tablosu gerekir. Bugünkü dağıtık depolar (Ceph, nesne depoları) hash tabanlı rastgele yerleşim kullanır.

Blok Önbellekleme ve Akış Birleştirme

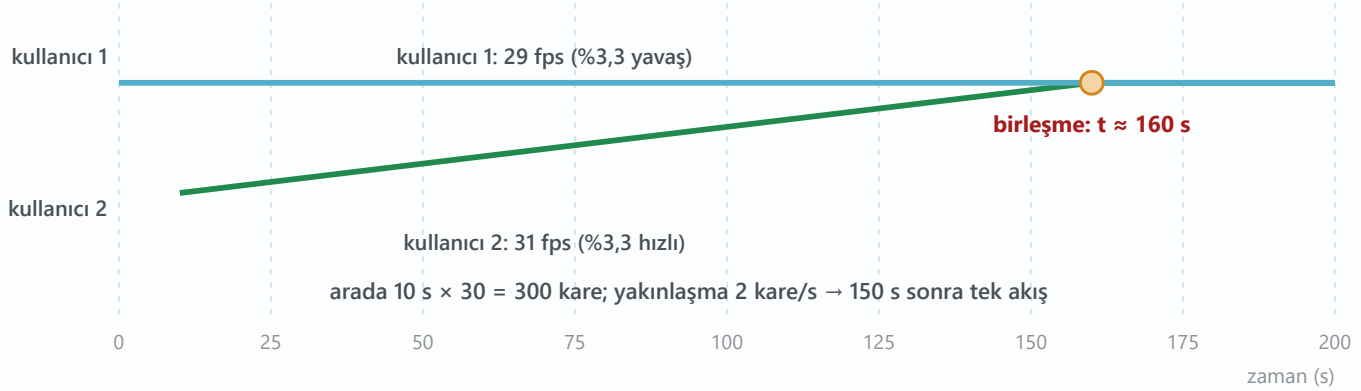


(a) İki kullanıcı aynı filmi 10 s arayla başlattı. Sunucu birinci kullanıcı için okuduğu blokları 10 s tutarsa, ikinci kullanıcı aynı blokları önbellekten alır.

Adım 1 / 2

Dosya önbellekleme: popüler filmler bütün olarak hızlı katmanda (eskiden teyp $\rightarrow$ disk, bugün HDD $\rightarrow$ SSD/RAM ve CDN kenarı) tutulur; az izlenenler yavaş katmandan istendiğinde getirilir.

Blok Önbellekleme ve Akış Birleştirme



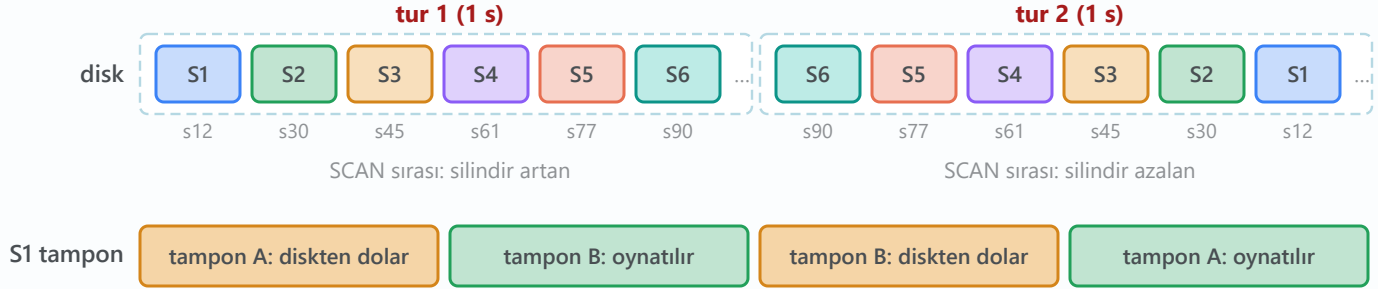
(b) Önbellek yoksa **akışları birleştirme**: önde olan biraz yavaşlatılır, arkadaki biraz hızlandırılır (fark edilmez). 300 karelik fark 150 s'de kapanır; sonra tek akış iki kullanıcıya gider.

Adım 2 / 2

Dosya önbellekleme: popüler filmler bütün olarak hızlı katmanda (eskiden teyp $\rightarrow$ disk, bugün HDD $\rightarrow$ SSD/RAM ve CDN kenarı) tutulur; az izlenenler yavaş katmandan istendiğinde getirilir.

DİSK ÇİZELGELEME

Statik Disk Çizelgeleme



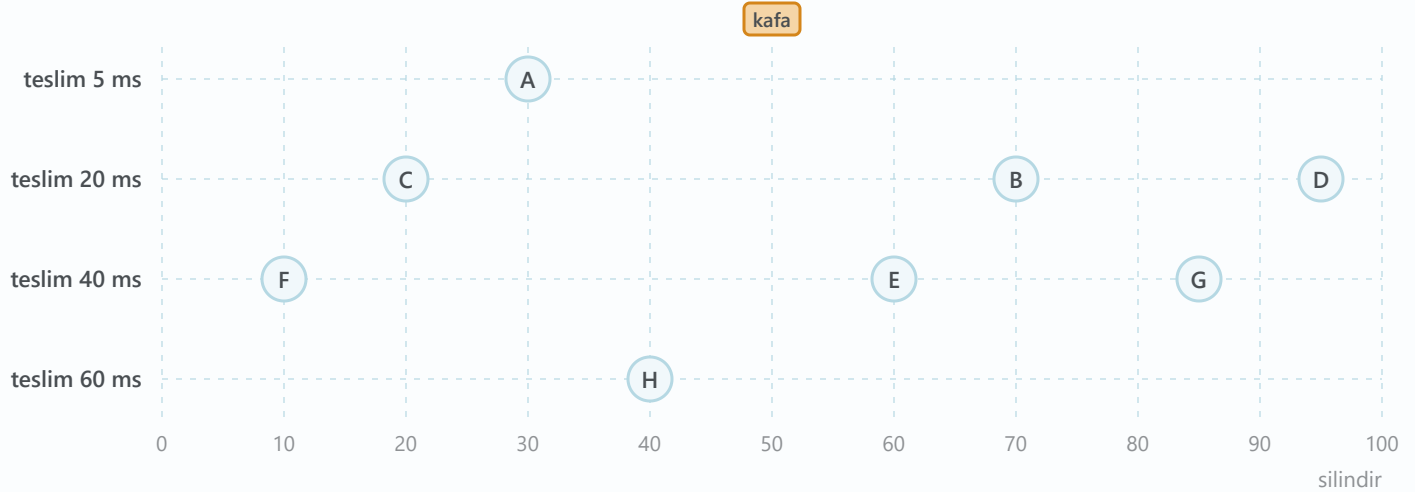
tur süresi = 1 s; okunan blok = her akışın 1 s'lik verisi (8 Mbit/s → 1 MB)

her blok tur sonuna kadar hazır olmalı; bu turda dolan tampon bir sonraki turda oynatılır

S1 ... S6: akışlar · s: silindir numarası

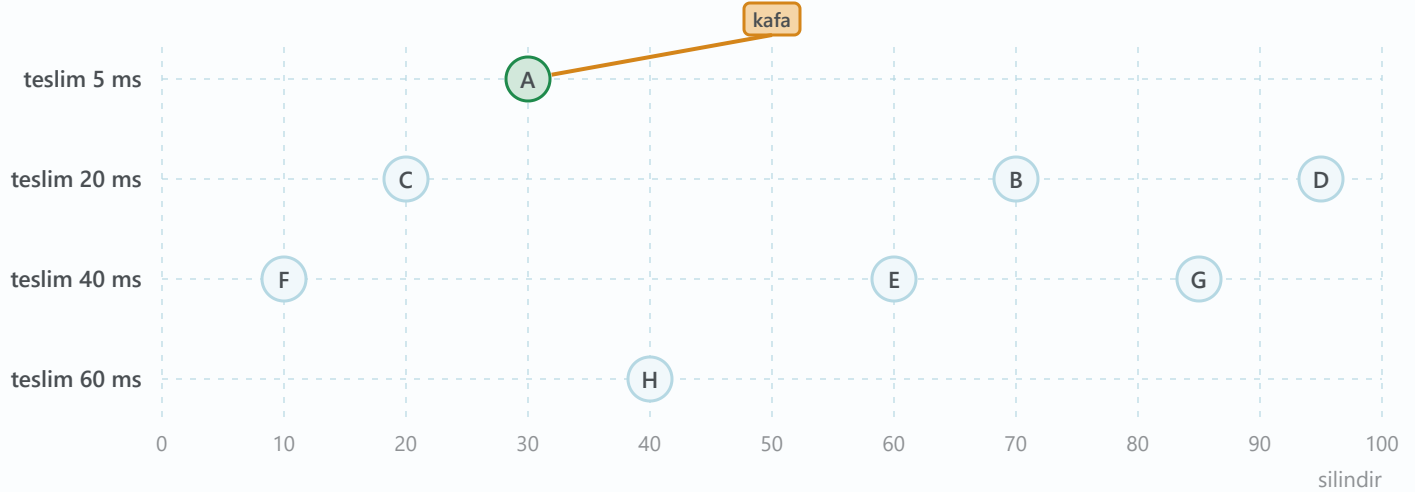
Neden büyük blok? HDD'de ortalama arama + dönme ≈ 8 ms, aktarım 200 MB/s. Kare başına (33 KB) okuma $\approx 8,2$ ms → 33,3 ms'lik turda yalnız **4 akış**. Turda 1 MB (1 s) okunursa istek ≈ 13 ms → saniyede $\approx$ **76 akış**. SCAN sıralaması arama süresini daha da kısaltır. NVMe SSD'de arama yoktur; tur fikri IOPS ve kuyruk derinliği planlamasına dönüşür.

Dinamik Disk Çizelgeleme: SCAN-EDF



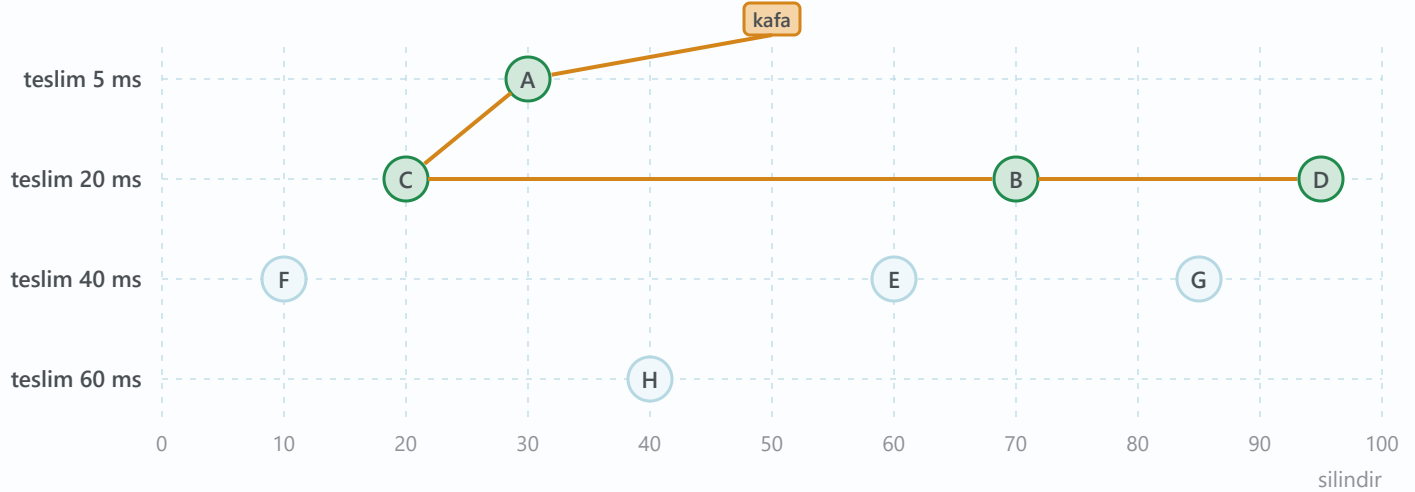
Kafa 50. silindirde, yukarı yönde. Sekiz istek, dört farklı son teslim zamanında (5, 20, 40, 60 ms). Salt SCAN teslim zamanlarını kaçırabilir; salt EDF kafayı zikzak çizdirir.

Dinamik Disk Çizelgeleme: SCAN-EDF



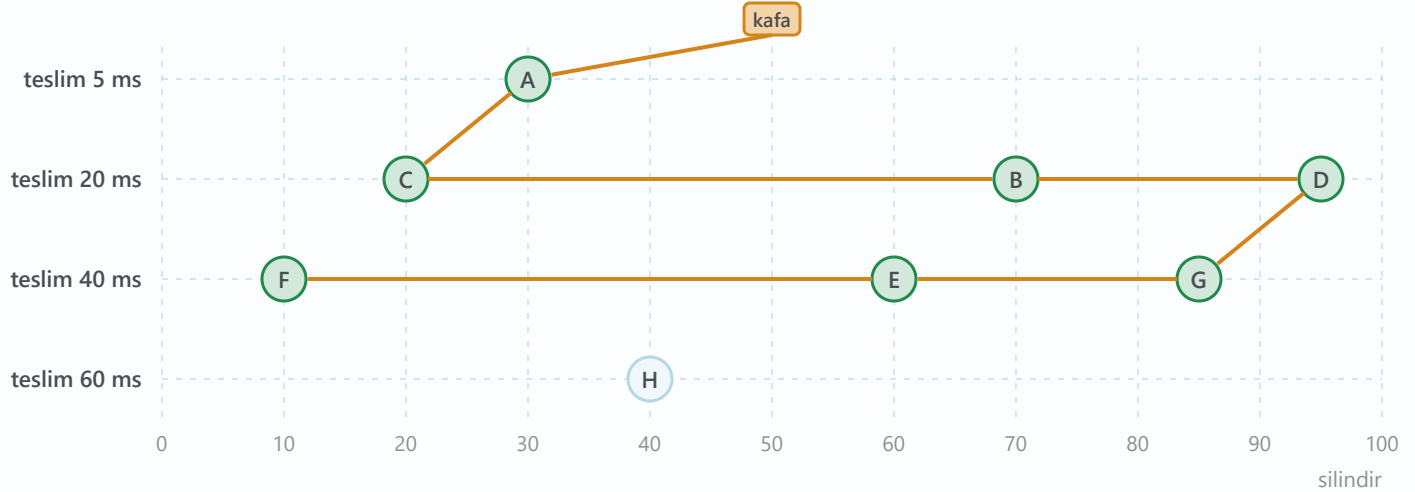
Önce teslimi en yakın grup: yalnız A (silindir 30). Kafa 50 → 30 iner; yön artık aşağı.

Dinamik Disk Çizelgeleme: SCAN-EDF



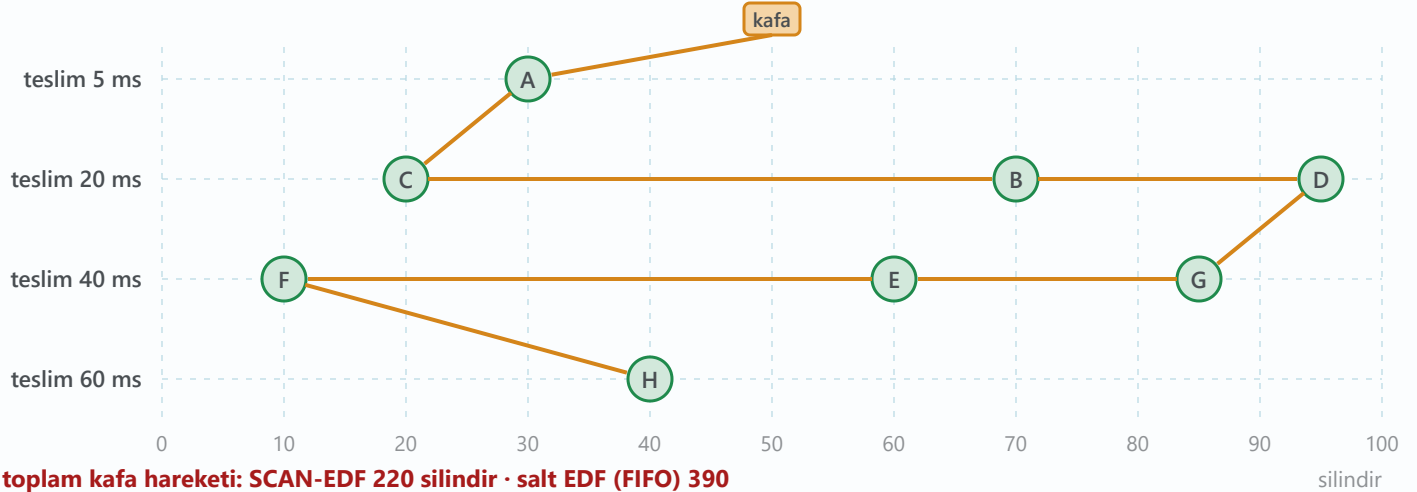
Teslim 20 ms grubu {B 70, C 20, D 95}: grup içinde SCAN, mevcut yönde (aşağı) önce C, sonra yukarı B, D: C(20) → B(70) → D(95).

Dinamik Disk Çizelgeleme: SCAN-EDF



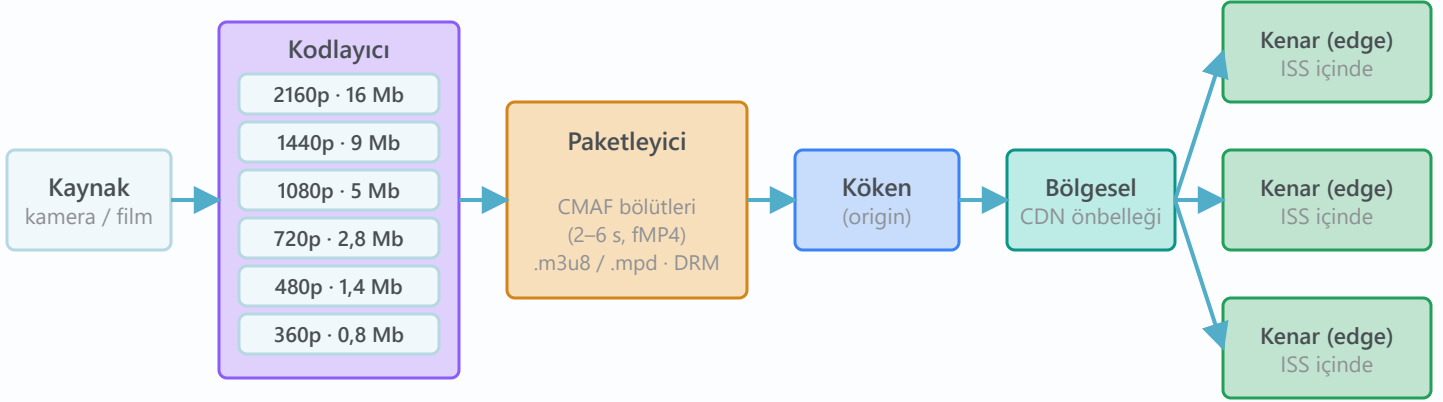
Teslim 40 ms grubu: kafa 95'te, yukarıda istek yok; aşağı doğru $G(85) \rightarrow E(60) \rightarrow F(10)$.

Dinamik Disk Çizelgeleme: SCAN-EDF



Son olarak H (40). Toplam hareket 220 silindir; aynı istekler salt EDF ile (grup içinde geliş sırası) 390 silindir. Gerçek sistemler “yakın” teslim zamanlarını da aynı gruba alarak daha fazla toplu tarama yapar.

Modern Akış Mimarisi: HLS, DASH ve CDN

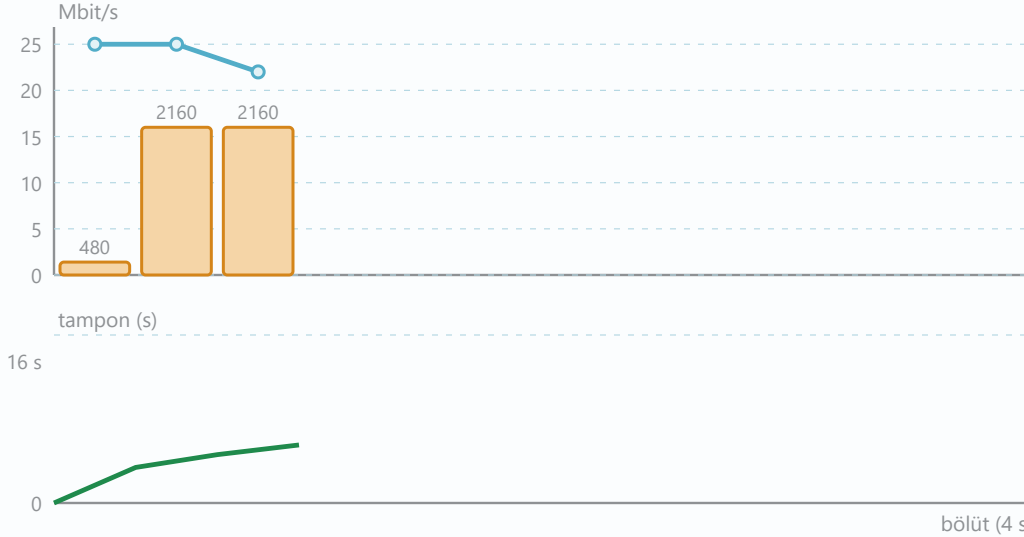


istek kenarda yoksa (ıska) bir üst katmana gider

↑ oynatıcılar: HTTP GET (çekme)

HLS (Apple) ve MPEG-DASH aynı CMAF bölütlerini paylaşabilir. Gecikme: klasik HLS/DASH 10–30 s, düşük gecikmeli LL-HLS/LL-DASH 2–5 s, etkileşimli WebRTC < 0,5 s. Video bugün internet trafiğinin yaklaşık üçte ikisini oluşturur.

Uyarlanabilir Bit Hızı (ABR) Merdiveni



kalite merdiveni

2160p · 16 Mbit/s

1440p · 9 Mbit/s

1080p · 5 Mbit/s

720p · 2,8 Mbit/s

480p · 1,4 Mbit/s

360p · 0,8 Mbit/s

240p · 0,4 Mbit/s

mavi: ölçülen hız

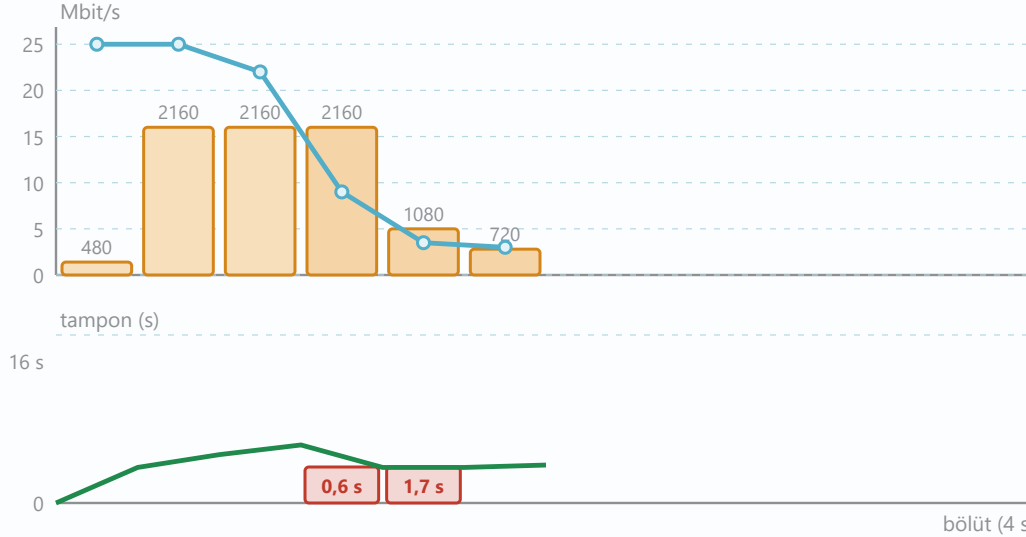
turuncu: seçilen bit hızı

yeşil: tampon doluluğu

kırmızı: takılma süresi

Başlangıç: ağ hızı bilinmiyor, oynatıcı güvenli seçer (480p). İlk ölçüm 25 Mbit/s → kural: ölçülen hızın %80'ine sığan en yüksek basamak → 2160p (16 Mbit/s).

Uyarlanabilir Bit Hızı (ABR) Merdiveni



kalite merdiveni

2160p · 16 Mbit/s

1440p · 9 Mbit/s

1080p · 5 Mbit/s

720p · 2,8 Mbit/s

480p · 1,4 Mbit/s

360p · 0,8 Mbit/s

240p · 0,4 Mbit/s

mavi: ölçülen hız

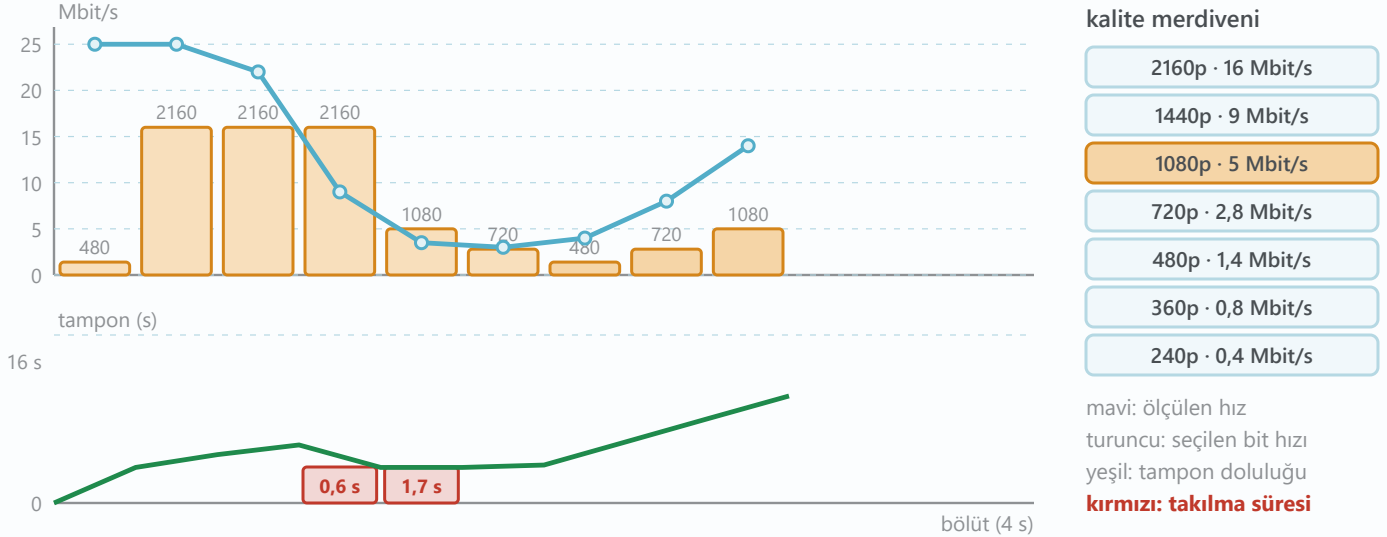
turuncu: seçilen bit hızı

yeşil: tampon doluluğu

kırmızı: takılma süresi

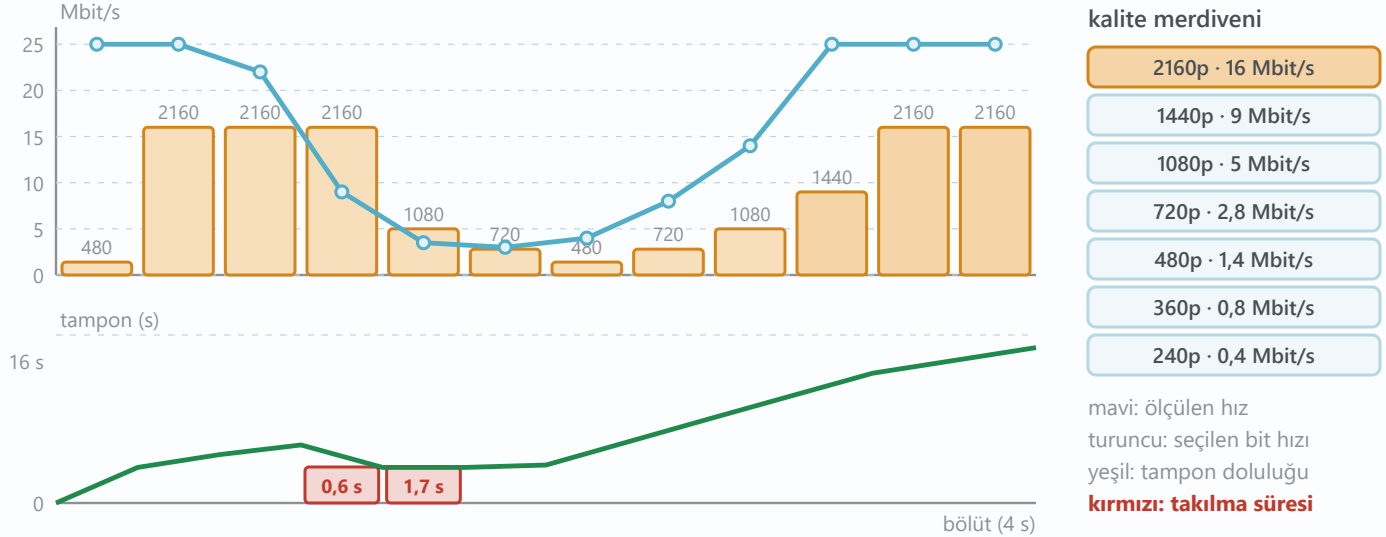
Ağ hızı 9, sonra 3,5 Mbit/s'ye düşer. Kural bir önceki ölçüme baktığı için 2160p bölütü 7,1 s sürer: tampon boşalır, **0,6 s takılma**; ardından 1080p de yetiştir (1,7 s).

Uyarlanabilir Bit Hızı (ABR) Merdiveni



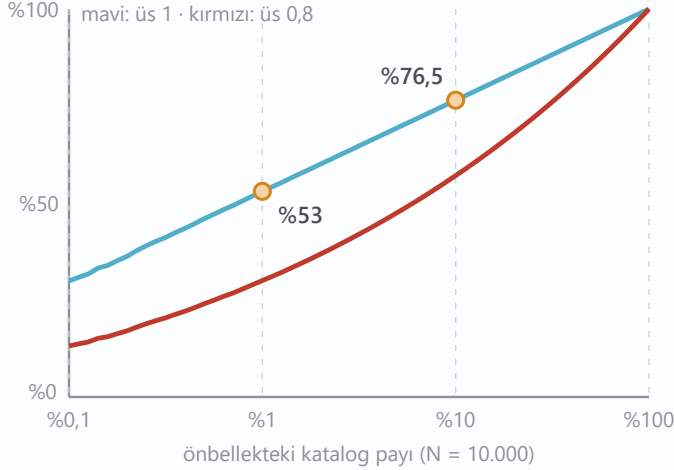
Oynatıcı 720p ve 480p'ye iner; indirme süresi bölüt süresinin altına düşünce tampon yeniden dolar ve kalite 720p, 1080p'ye çıkar.

Uyarlanabilir Bit Hızı (ABR) Merdiveni



Hız toparlanınca kalite kademeli olarak 1440p ve 2160p'ye çıkar, tampon ≈ 17 s. Gerçek oynatıcılar (BOLA, hibrit kurallar) tampon düzeyine de bakarak bu takılmaları önler.

CDN Önbellediği ve Zipf



- Katalogun en popüler **%1**'i (100 başlık) isteklerin **%53**'ünü, **%10**'u **%76,5**'ini karşılar (Zipf, $\alpha = 1$, $H_{100} / H_{10.000}$).
- $\alpha = 0,8$ ise kuyruk daha kalındır: $\%1 \rightarrow \%30$, $\%10 \rightarrow \%57$. Bu yüzden kenar önbellekleri büyük SSD'ler kullanır.
- Katmanlar: oynatıcı tamponu $\rightarrow$ ISS içindeki kenar sunucu (ör. Netflix Open Connect) $\rightarrow$ bölgesel önbellek $\rightarrow$ köken. Gece boş saatte popüler içerik önceden doldurulur.
- Değiştirme: LRU basit ama tek seferlik isteklerle kirlenir; LFU/ARC türevleri ve "ikinci istekte önbelleğe al" kuralları tercih edilir.

ÖZET

Özet

Özellikler

Sürekli ortam: yüksek veri hızı, gerçek zaman, titreşim, QoS ve senkronizasyon. Sıkıştırmasız 4K60 HDR 7,46 Gbit/s, akışı 16 Mbit/s.

Kodlama

Örnekleme (Nyquist), niceleme, psikoakustik maskeleme; YCbCr, 4:2:0, DCT + niceleme + entropi kodlama; I/P/B kareler ve hareket dengeleme.

Kodekler

MPEG-2 → H.264 → HEVC/VP9 → AV1/VVC: her kuşak $\approx$ %30–50 tasarruf; çözme donanımda. Ses: AAC, Opus, FLAC.

Süreç çizelgeleme

Homojen akıslarda döngü; genelde RMS (statik, $U \leq m(2^{1/m} - 1)$ yeterli) ve EDF (dinamik, $U \leq 1$ yeterli ve gerekli).

Dosya sistemi

İtme/çekme, NVOD ve VCR, iç içe yerleşim, blok boyutu ve indis, Zipf + organ borusu, şeritleme, blok ve dosya önbelleği.

Disk ve akış

Statik turlar + SCAN, SCAN-EDF; bugün HLS/DASH bölütleri, CDN katmanları ve uyarlanabilir bit hızı merdiveni.

Kontrol Soruları

1. 48 kHz, 24 bit, 6 kanallı (5.1) PCM sesin veri hızı nedir?
2. 1280×720 , 60 fps, 8 bit 4:2:0 sıkıştırmasız videonun veri hızı nedir?
3. 12 kHz bileşen içeren bir sinyal 16 kHz ile örneklenirse ne olur?
4. JPEG'de hangi adımlar kayıplı, hangileri tersinirdir?
5. Görüntüleme sırası I1 B2 B3 P4 B5 B6 P7 olan karelerin kod çözme sırası nedir? Neden?
6. (P, C) = (20, 5), (50, 10), (100, 30) ms görevleri RMS ile kesin çizelgelenebilir mi?
7. 90 dk'lık film her 10 dk'da bir başlatılırsa NVOD kaç akış ister, ortalama bekleme nedir?
8. Zipf (N = 20) ile ilk 3 filmin istek payı nedir? Organ borusunda nereye konurlar?
9. Kare indisi ile blok indisi arasındaki bellek-disk israfı ödünleşimi nedir?
10. Ölçülen hız 6 Mbit/s, kural "hızın %80'ine sığan en yüksek basamak": slayttaki merdivenden hangisi seçilir?

► Cevaplar

Kaynaklar

- A. S. Tanenbaum, *Modern Operating Systems*, 3. baskı, Pearson, 2008 — Bölüm 7: Multimedia Operating Systems (bu sunumun temel kaynağı).
- A. Silberschatz, P. B. Galvin, G. Gagne, *Operating System Concepts*, 8. baskı, Wiley, 2009 — Bölüm 20: Multimedia Systems (9. baskıda, 2012, bu bölüm çıkarılmış; gerçek zamanlı çizelgeleme Bölüm 6.6'dadır).
- C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", JACM, 1973; A. L. N. Reddy, J. Wyllie, "Disk Scheduling in a Multimedia I/O System" (SCAN-EDF), ACM Multimedia, 1993.
- ITU-T T.81 (JPEG); ITU-T H.264, H.265, H.266; AOMedia AV1 Bitstream Specification; IETF RFC 6716 (Opus); ISO/IEC 23000-19 (CMAF).
- IETF RFC 8216 ve draft-pantos-hls-rfc8216bis (HLS, LL-HLS); ISO/IEC 23009-1 (MPEG-DASH); ITU-R BT.601, BT.709, BT.2020, BT.2100 (HDR, PQ/HLG).
- Linux çekirdek belgeleri: "Deadline Task Scheduling" (SCHED_DEADLINE), VA-API ve V4L2 kodek arayüzleri.

Son

Bölüm 14: Çoklu Ortam Sistemleri

Sonraki bölüm: İşletim Sistemi Tasarımı · [Terim sözlüğü](#)