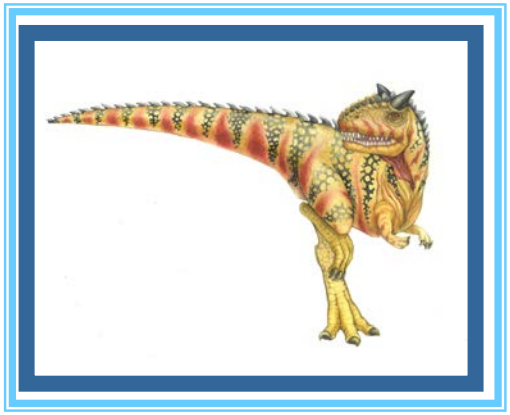
Bölüm 9:
Sanal Bellek Yönetimi



Bölüm 9: Sanal Bellek Yönetimi

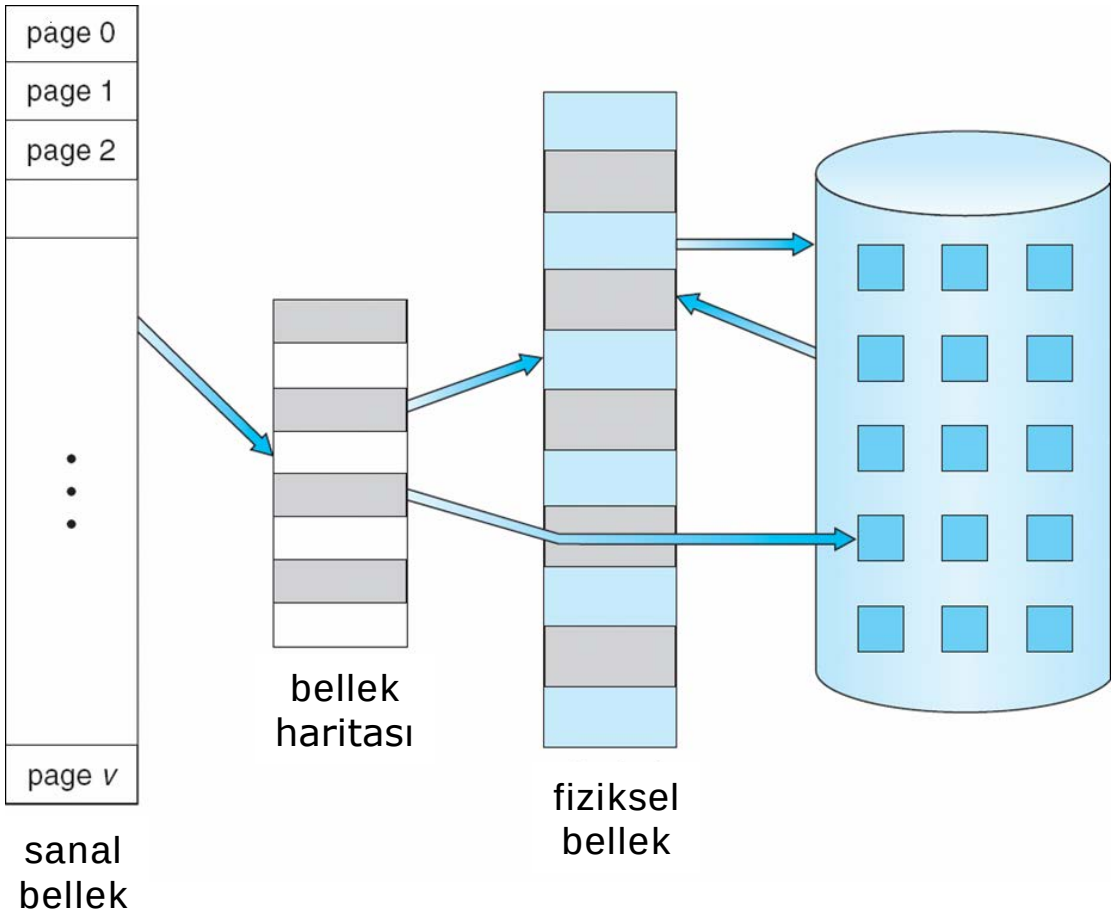
- Altyapı
- İstek Sayfalama (Demand Paging)
- Yazarken Kopyala (Copy-on-Write)
- Sayfa Değiştirme (Page Replacement)
- Çerçevelerin Tahsisi (Allocation of Frames)
- Boşuna Çalışma (Thrashing)
- Bellekte Haritalanan Dosyalar (Memory-Mapped Files)
- Çekirdek Belleğinin Tahsisi (Allocating Kernel Memory)
- Diğer Durumlar

Altyapı

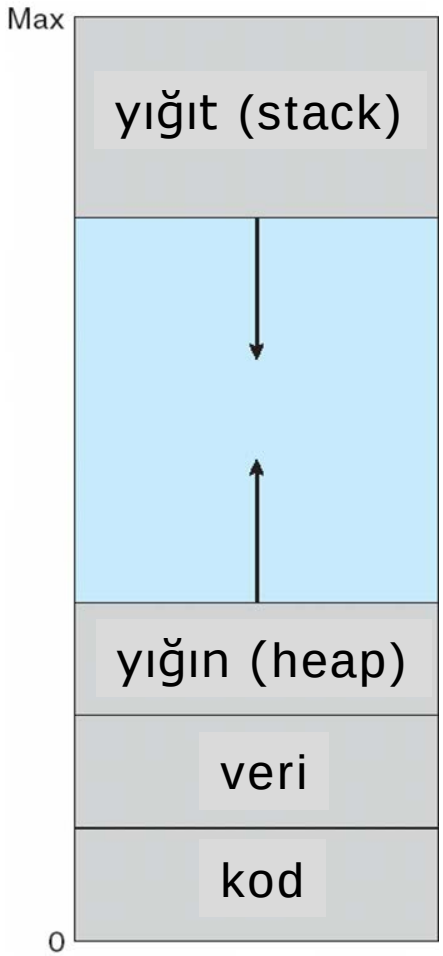
- **Sanal Bellek (Virtual memory)** – kullanıcı mantıksal belleğinin fiziksel bellekten ayrılması.
  - Programın sadece bir parçası çalıştırılmak üzere bellekte bulunmalıdır
  - Bu yüzden mantıksal adres alanı fiziksel adres alanından çok daha büyük olabilir
  - Adres alanlarının çeşitli süreçler tarafından paylaşılmasına izin verir
  - Daha verimli süreç yaratılmasına izin verir

- Sanal Bellek:
  - İstek safyalama (Demand paging)
  - İstek bölümleme (Demand segmentation) yollarından biri ile gerçeklenebilir

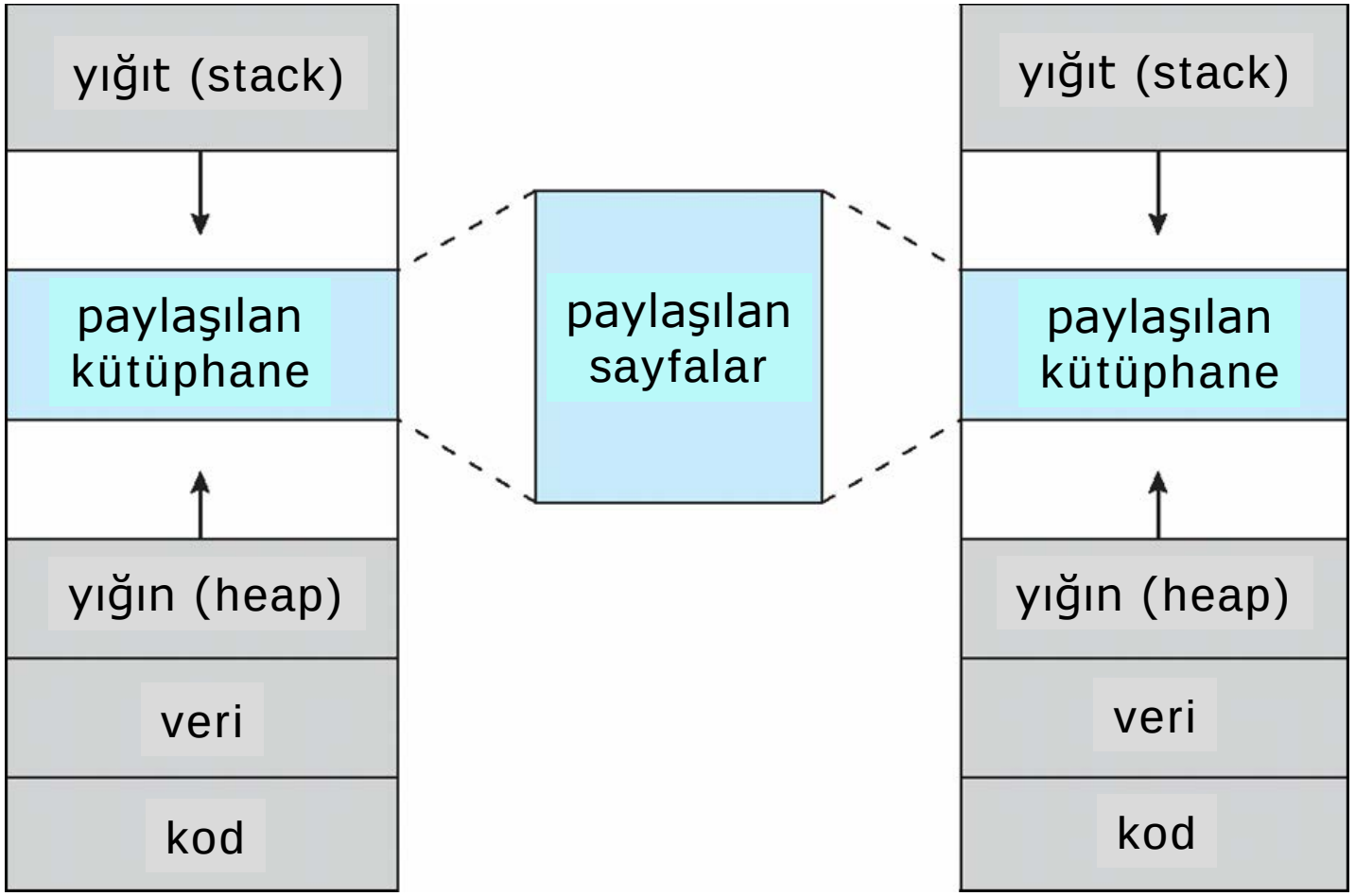
Fiziksel Bellekten Daha Büyük Olan Sanal Bellek



Sanal Adres Alanı



Sanal Bellek Kullanan Paylaşılan Kütüphane

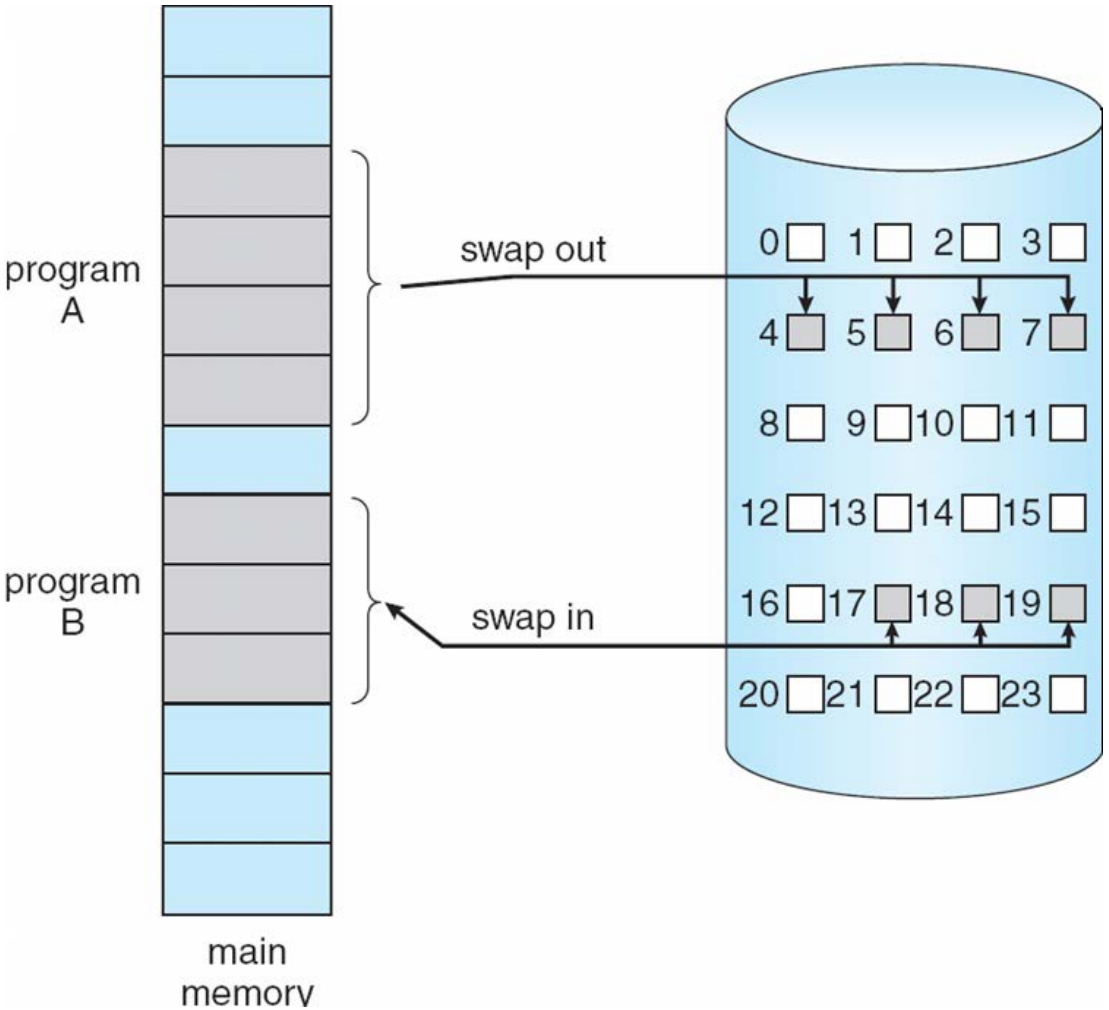


İstek Sayfalama (Demand Paging)

- Bir sayfayı yalnızca ihtiyaç duyulduğunda belleğe getirme
  - Daha az G/Ç gereklidir
  - Daha az bellek gereklidir
  - Daha hızlı tepki
  - Daha fazla kullanıcı

- Sayfa gerekli ➔ Ona referans göster
  - Geçersiz referans ➔ Terket (abort)
  - Bellekte değil ➔ Belleğe getir
- **Tembel Değiştirici (Lazy swapper)**– sayfa gerekmediği sürece asla belleğe getirilmez
  - Bir sürecin sayfalarıyla (page) ilgilenen değiştiriciye (swapper) **sayfalayıcı (pager)** adı verilir
  - İstek sayfalamada sayfalayıcı kullanılır

Sayfalanmış Bellekten Ardışık Disk Alanına Transfer



Geçerli-Geçersiz Biti (Valid-Invalid Bit)

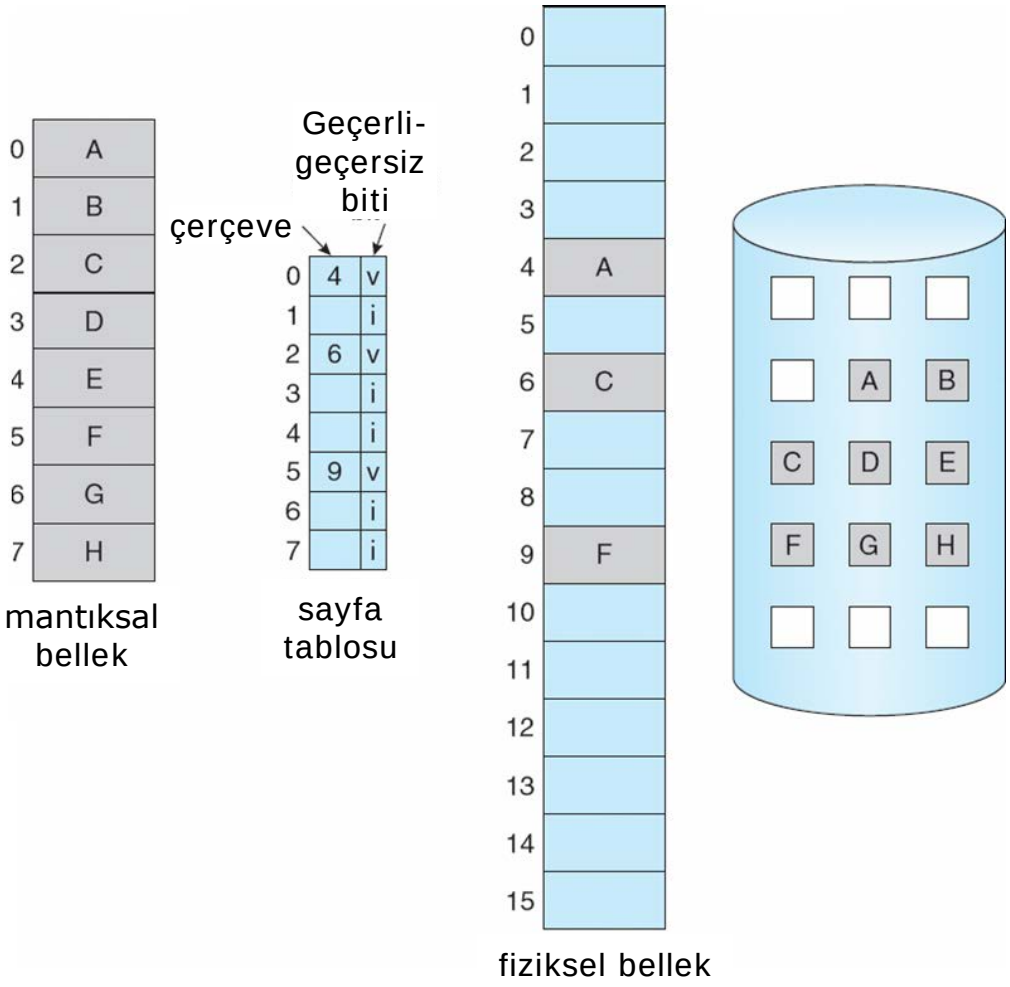
- Her sayfa tablosu girdisi ile bir geçerli-geçersiz biti ilişkilendirilmiştir (**v** ⇒ bellekte, **i** ⇒ bellekte değil)
- Başlangıçta geçerli-geçersiz biti tüm girdiler için **i** (geçersiz) olarak tanımlanmıştır
- Örnek sayfa tablosu:

Çeçeve #	Geçerli-Geçersiz Biti
	v
	v
	v
	v
	i
….	
	i
	i

Sayfa tablosu

- Adres dönüşümü sırasında, eğer geçerli-geçersiz biti sayfa tablosu girdisinde **I** ise ➔ sayfa hatası

Bazı Sayfaları Bellekte Bulunmayan Sayfa Tablosu

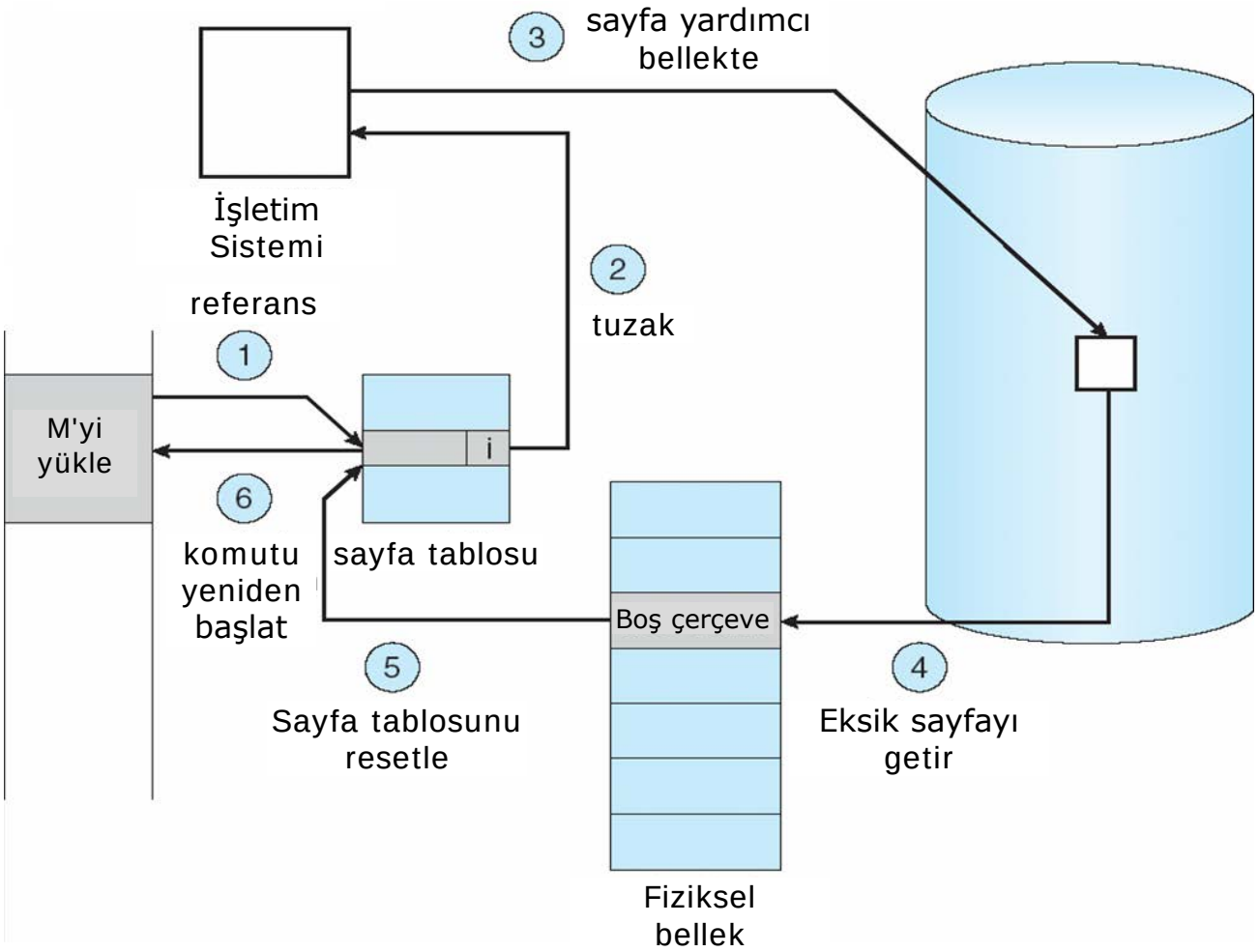


Sayfa Hatası (Page Fault)

- Geçersiz olarak işaretlenmiş bir sayfaya referans verilmesi durumunda sayfa hatası (page fault) oluşur.
- **Sayfa hata**larını çözmek için:

1. İşletim Sistemi karar vermek için başka bir tabloya (genelde süreç kontrol bloğunda yer alan) bakar:
   - Geçersiz referans ➔ terket (abort)
   - Geçerli ancak bellekte değil
2. Boş bir çerçeve bul
3. Sayfayı çerçevenin içine taşı
4. Tabloları yeniden düzenle
5. Geçerli biti = **v** yap
6. Sayfa hatasına sebep olan komutu yeniden başlat

Sayfa Hatasını İşleme Adımları



Sayfa Hatası

- Sayfa hatası oluştuğunda aşağıdaki sıralama meydana gelir:

1. İşletim sisteminde tuzak oluştur.
2. Kullanıcı kayıtçılarını ve süreç durumunu kaydet.
3. Oluşan kesmenin sayfa hatası olduğuna karar ver.
4. Sayfa referansının geçerli olduğunu kontrol et ve sayfanın diskteki yerine karar ver.
5. Diskten boş çerçeveye bir okuma işlemi gerçekleştir:
   - **a.** Okuma isteği gerçekleştirilinceye kadar bu aygıt için kuyrukta bekle.
   - **b.** Aygıt arama ve/veya gecikme süresi boyunca bekle.
   - **c.** Sayfadan boş çerçeveye doğru transfere başla.
6. Beklerken, CPU'yu başka bir kullanıcıya tahsis et.

Sayfa Hatası

7. Disk G/Ç altsisteminden bir kesme (G/Ç tamamlandı) al.

8. Diğer kullanıcı için kayıtçıları ve süreç durumunu kaydet (eğer 6. adım gerçekleştirilmişse)

9. Kesmenin diskten geldiğine karar ver.

10. İstenen sayfanın bellekte olduğunu gösterecek şekilde sayfa tablosu ve diğer tabloları düzelt.

11. CPU'nun bu sürece yeniden tahsis edilmesi için bekle

12. Kullanıcı kayıtçıları, süreç durumu ve yeni tablo sayfasını yeniden yükle ve kesilen komuta devam et.

İstek Sayfalamanın Performansı

- Sayfa Hata Oranı $0 \le p \le 1.0$
  - Eğer $p = 0$ sayfa hatası yoktur
  - Eğer $p = 1$, her referans hatalıdır

- Verimli Erişim Zamanı (Effective Access Time (EAT))

  EAT = (1 – p) x bellek erişimi
  + p (sayfa hata maliyeti (page fault overhead)
  + sayfayı götürme (swap page out)
  + sayfayı getirme (swap page in)
  + yeniden başlatma maliyeti (restart overhead)
)

İstek Sayfalama Örneği

- Bellek Erişim Süresi = 200 nanosaniye
- Ortalama sayfa hatası servis zamanı = 8 milisaniye
- EAT = (1 – p) x 200 + p (8 milisaniye)
  = (1 – p) x 200 + p x 8,000,000
  = 200 + p x 7,999,800 nanosaniye
- Eğer 1000 erişimden bir tanesi sayfa hatasına sebep oluyorsa
  EAT = 200 + 0.001 x 7,999,800 = 8200 nanosaniye = 8.2 mikrosaniye.
  Bu da istek sayfalamadan dolayı 40 oranında bir yavaşlama faktörüdür!!
- Eğer kabul edilebilir bir yavaşlama (%10 gibi) istiyorsak sayfa hatası oranını 400000'de 1 seviyesine düşürmeliyiz.

Süreç Yaratma

- Sanal Belleğin diğer faydaları süreç yaratırken ortaya çıkar:

  - Yazarken Kopyala (Copy-on-Write)

  - Bellek Haritasındaki dosyalar (Memory-Mapped Files) (daha sonra)

Yazarken Kopyala (Copy-on-Write)

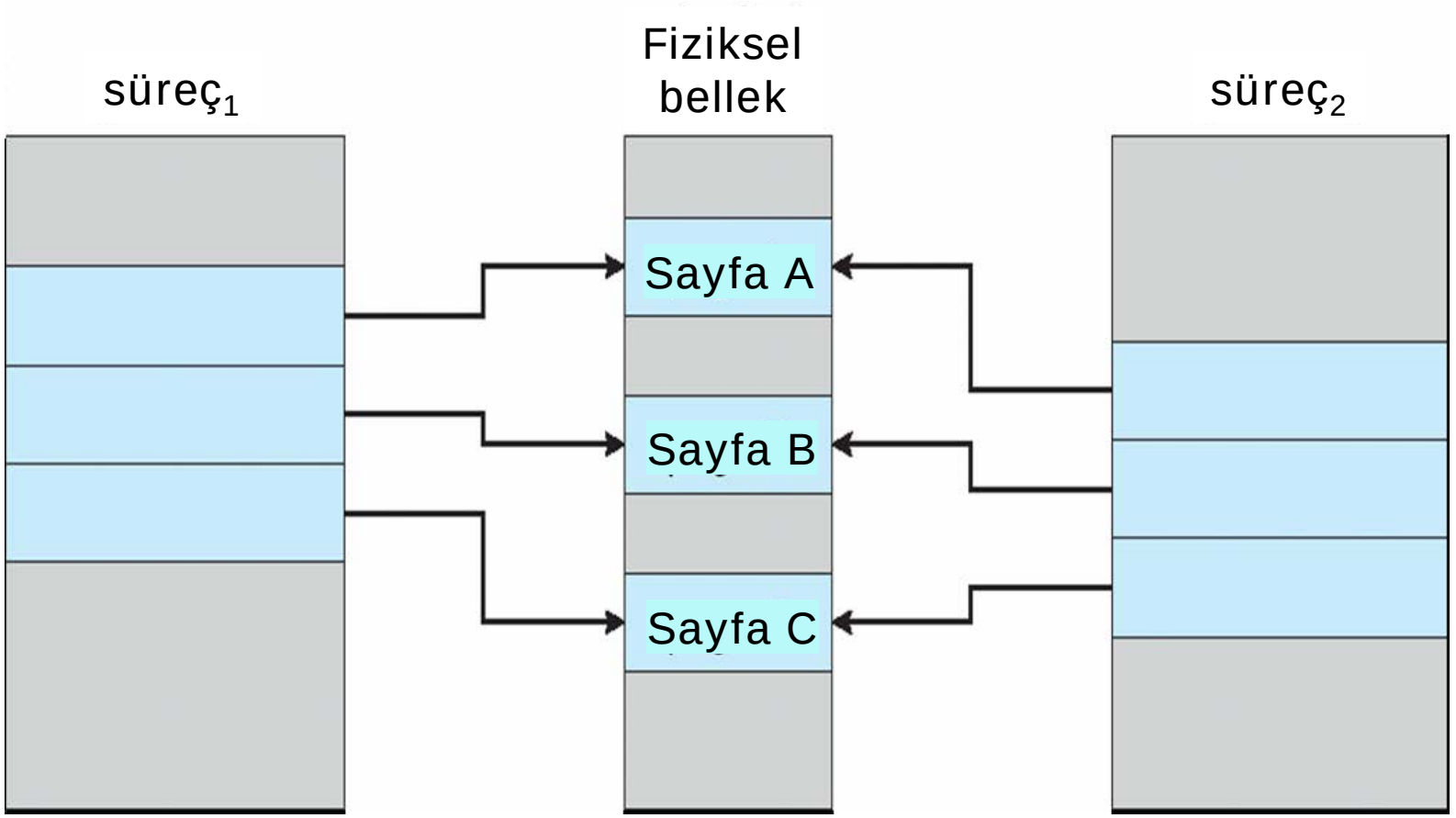
- Yazarken Kopyala (Copy-on-Write (COW)) hem ana hem de alt süreçler başlangıçta bellekte aynı sayfaları paylaşırlar

  Eğer bu süreçlerden biri paylaşılan sayfayı değiştirirse, bu durumda sayfa kopyalanır

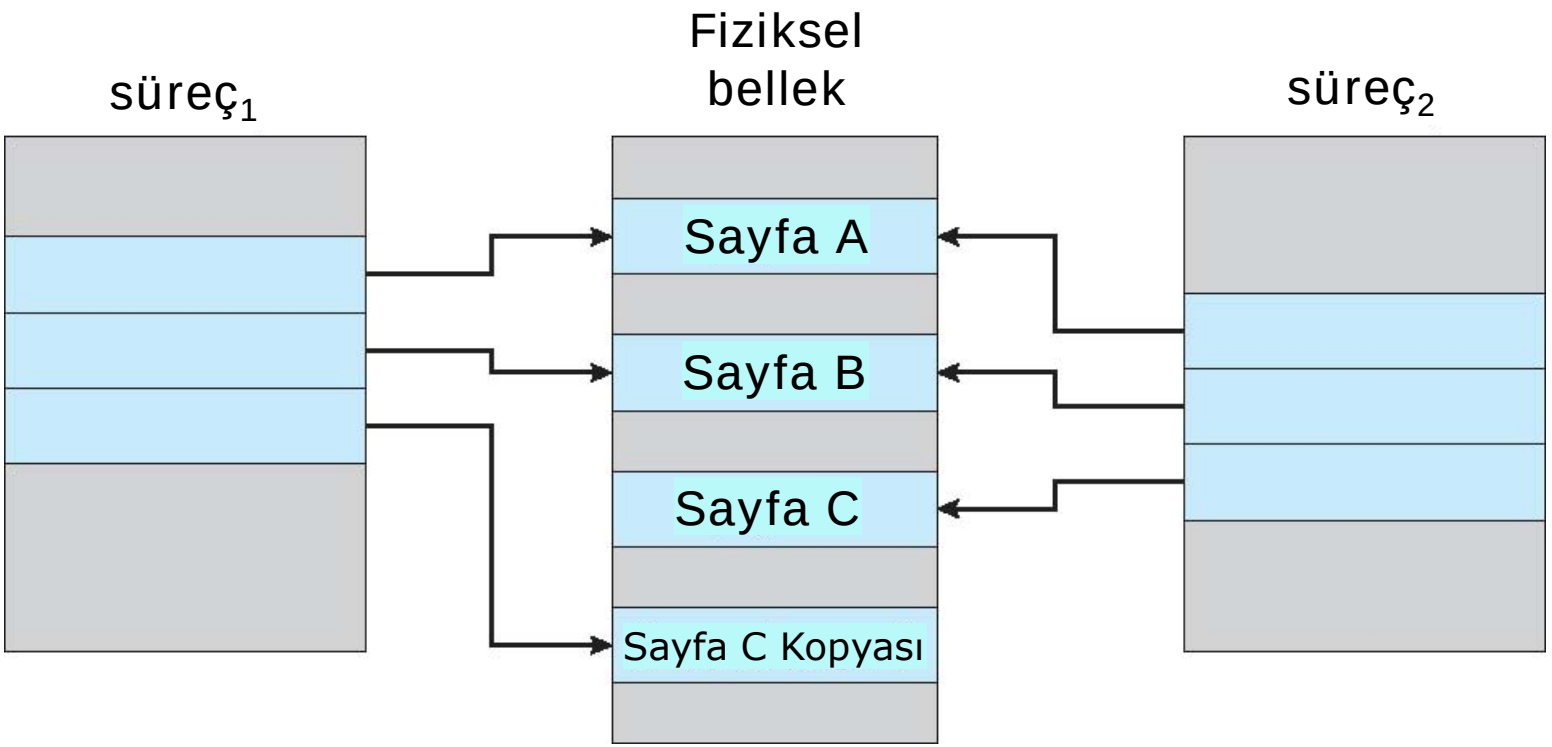
- COW sadece değiştirilen sayfaları kopyaladığı için süreç yaratımını daha verimli olarak gerçekleştirir

- Boş (serbest) sayfalar sıfırlanmış sayfalardan oluşan bir **havuz**dan tahsis edilir

Süreç 1 Sayfa C'yi Değiştirmeden Önce



Süreç 1 Sayfa C'yi Değiştirdikten Sonra



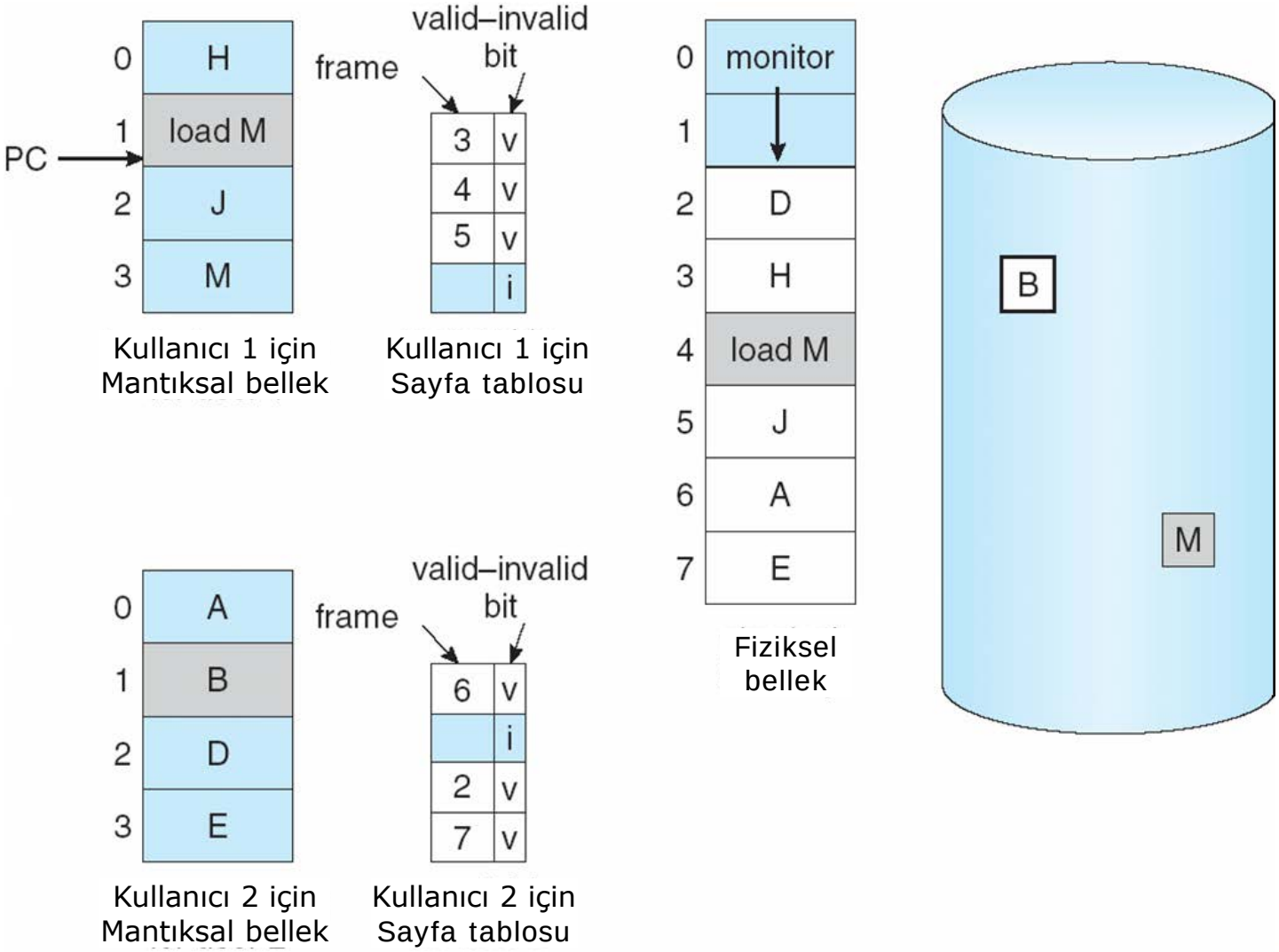
Eğer Boş Çerçeve Yoksa Ne Olur?

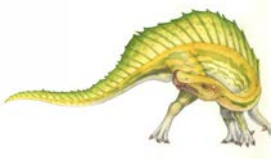
- Sayfa Değiştirme (Page Replacement) – bellekte gerçekte kullanılmayan sayfalar bul ve dışarı taşı
  - algoritma
  - performans – minimum sayıda sayfa hatasına sebep olacak bir algoritma gerektirir
- Bazı sayfaların birkaç kere belleğe getirilmesi gerekebilir

Sayfa Değiştirme (Page Replacement)

- Çoklu programlama seviyesini arttırırsak belleğe aşırı tahsis (over-allocation) gerçekleştiririz.
- Sayfa hatası servis rutinini sayfa değiştirmeyi içerecek şekilde değiştirerek belleğin aşırı tahsisini (over-allocation) engellemek
- **Değiştirme (kirli) Biti** (**modify (dirty) bit**) kullanarak sayfa transfer maliyetini azaltmak – sadece değiştirilen sayfalar diske yazılır
- Sayfa değiştirme, mantıksal ve fiziksel bellekteki ayrımı tamamlar – küçük bir fiziksel bellek üzerinde büyük bir sanal bellek kullanılabilir

Sayfa Değiştirme İhtiyacı





Basit Sayfa Değiştirme

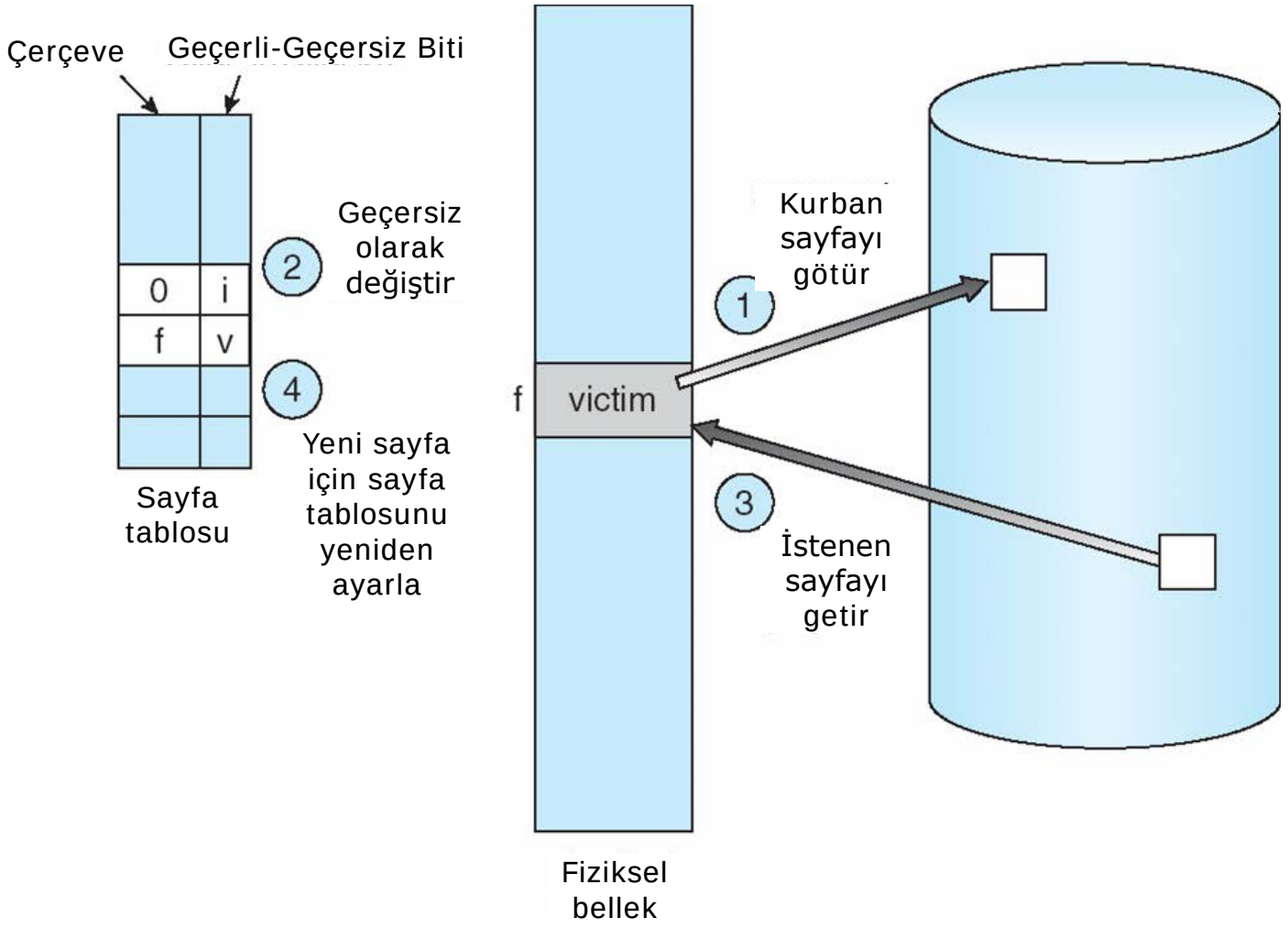
1. İstenen sayfanın yerini diskte bul

2. Boş bir çerçeve bul:
   - Eğer boş çerçeve varsa, kullan
   - Eğer boş çerçeve yoksa, bir **kurban (victim) çerçeve** seçmek için bir sayfa değiştirme algoritması çalıştır
   - Kurban çerçeveyi diske yaz; buna göre sayfa ve çerçeve tablolarını değiştir

3. İstenen sayfayı yeni boş çerçeveye getir; sayfa ve çerçeve tablolarını güncelle

4. Süreci yeniden başlat

Basit Sayfa Değiştirme

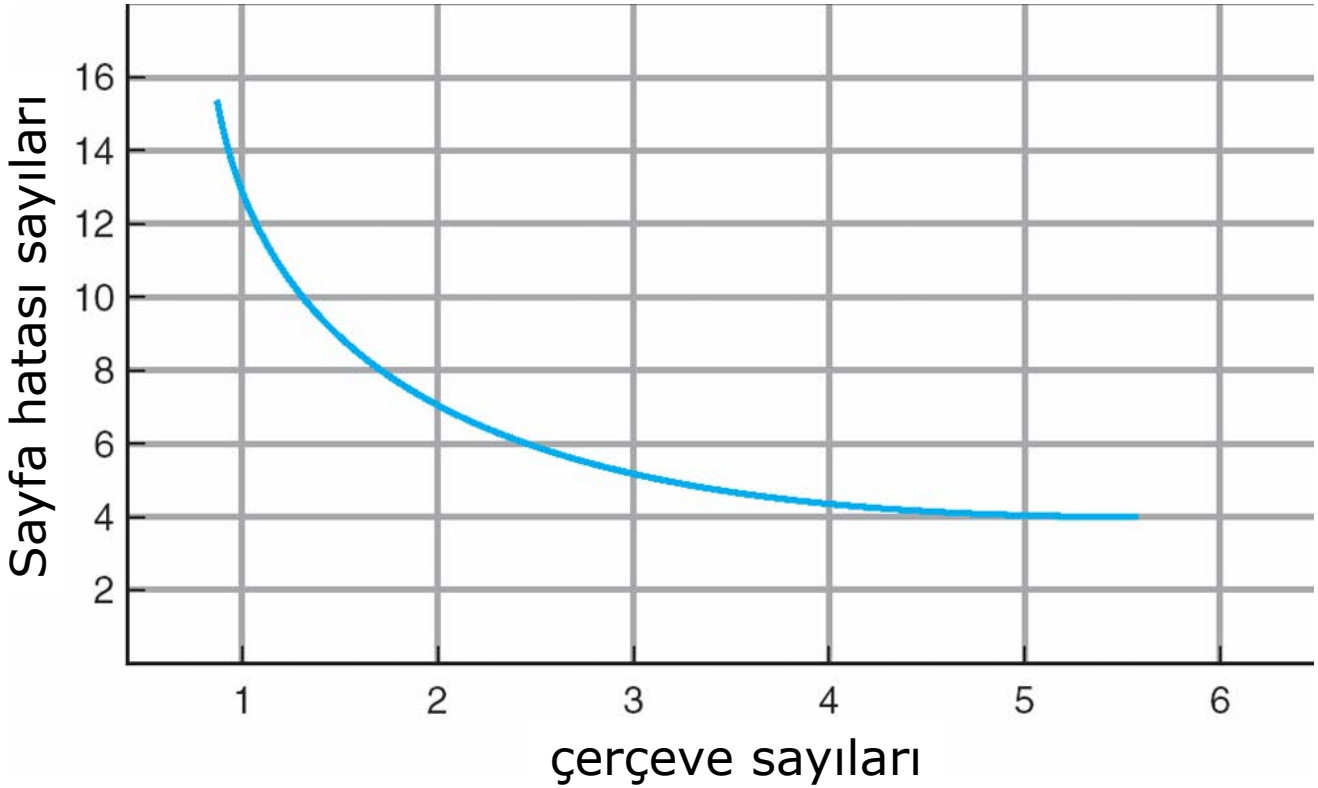


Sayfa Değiştirme Algoritmaları

- En düşük sayfa hatası oranı istenir
- Algoritmayı belirli bellek referansları dizisi üzerinde çalıştırmalı daha sonra bunları değerlendirmeli ve bu dizideki sayfa hataları hesaplanmalıdır
- Bizim örneklerimizde referans dizisi

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Sayfa Hataları – Çerçeve Sayıları Grafiği



İlk Gelen İlk Çıkar (First-In-First-Out (FIFO)) Algoritması

- Referans dizisi: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 çerçeve (Her süreç için aynı anda 3 sayfa bellekte bulunabilir)

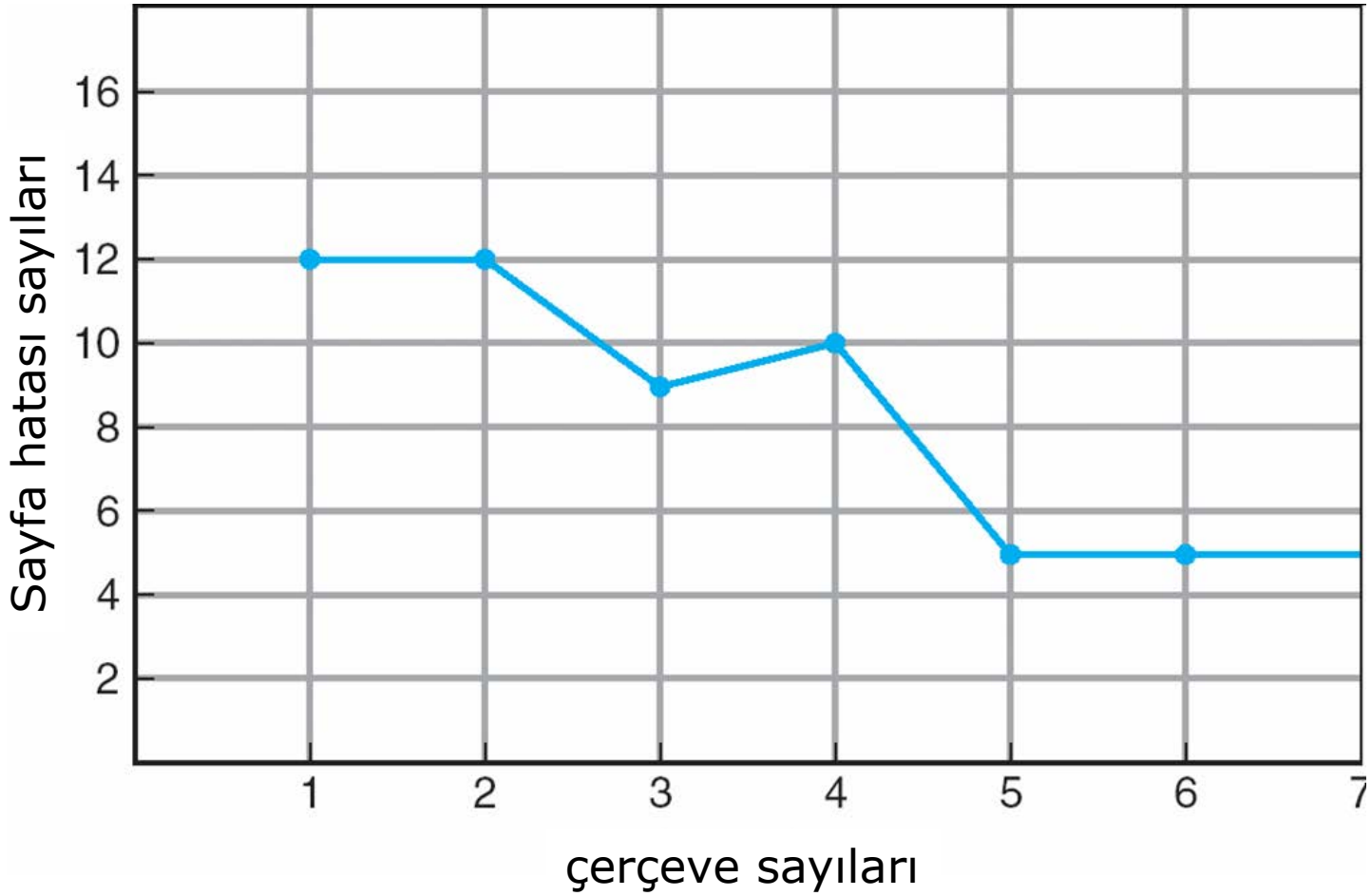
1	1	4	5	
2	2	1	3	9 sayfa hatası
3	3	2	4	

- 4 çerçeve

1	1	5	4	
2	2	1	5	10 sayfa hatası
3	3	2		
4	4	3		

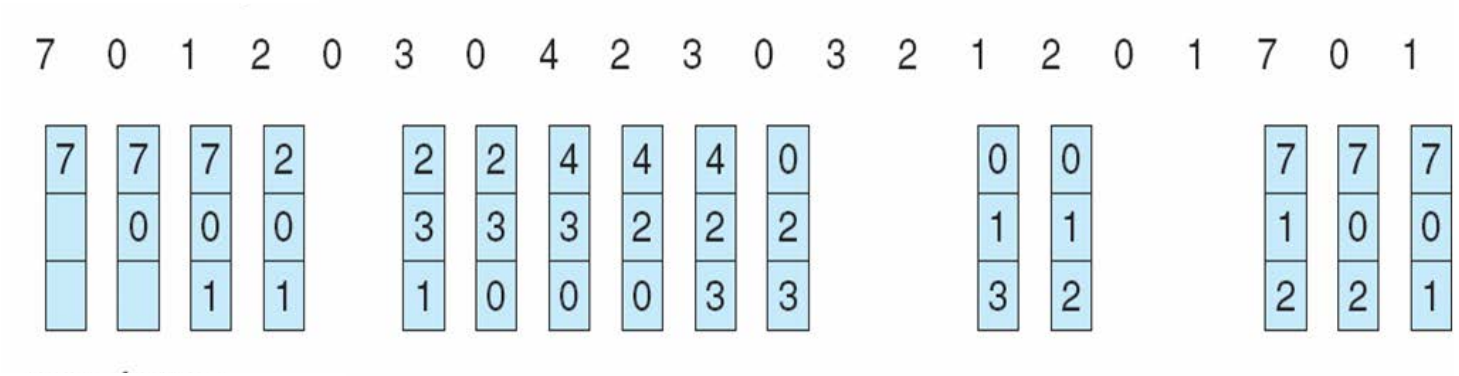
- Belady'nin Anormalliği: daha fazla çerçeve ➔ daha fazla sayfa hatası

Belady'nin Anormalliğini Gösteren FIFO



FIFO Sayfa Değiştirme

Referans dizisi



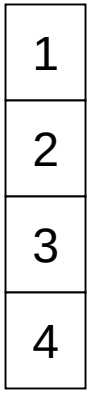
Sayfa çerçeveleri

15 sayfa hatası oluşur

Optimal Sayfa Değiştirme Algoritması

- En uzun süre için kullanılmayacak olan sayfayı değiştir
- 4 çerçeve örneği

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



4

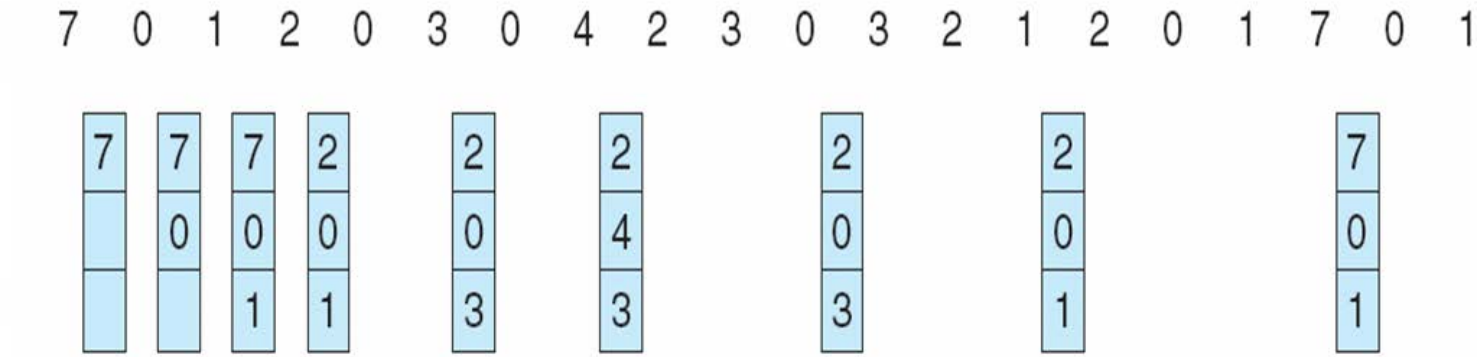
5

6 sayfa hatası

- Bunu nasıl biliyoruz?
- Algoritmanın ne kadar iyi olduğunu ölçmek için kullanılır

Optimal Sayfa Değiştirme

Referans dizisi



Sayfa çerçeveleri

9 sayfa hatası oluşur

En Uzak Geçmişte Kullanılan (Least Recently Used (LRU)) Algoritması

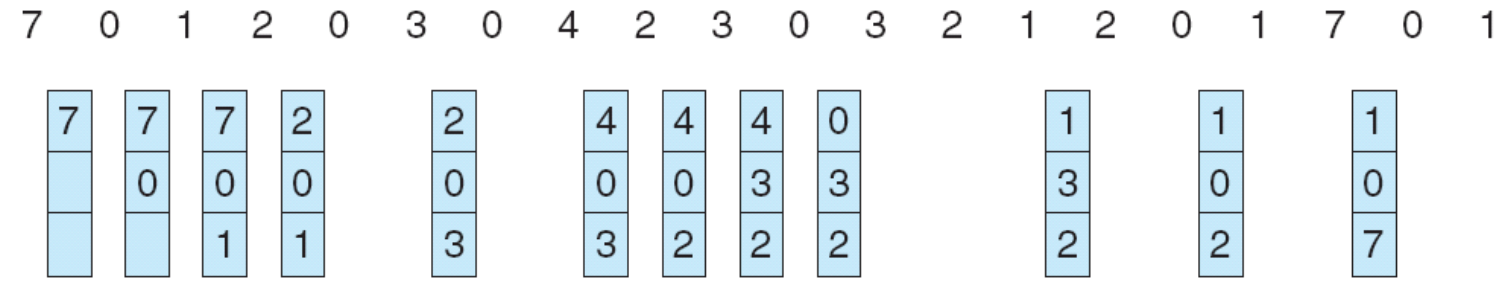
- Referans dizisi: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**

1	1	1	1	**5**
2	2	2	2	2
3	**5**	5	**4**	4
4	4	**3**	3	3

8 sayfa hatası

LRU Sayfa Değiştirme

Referans dizisi



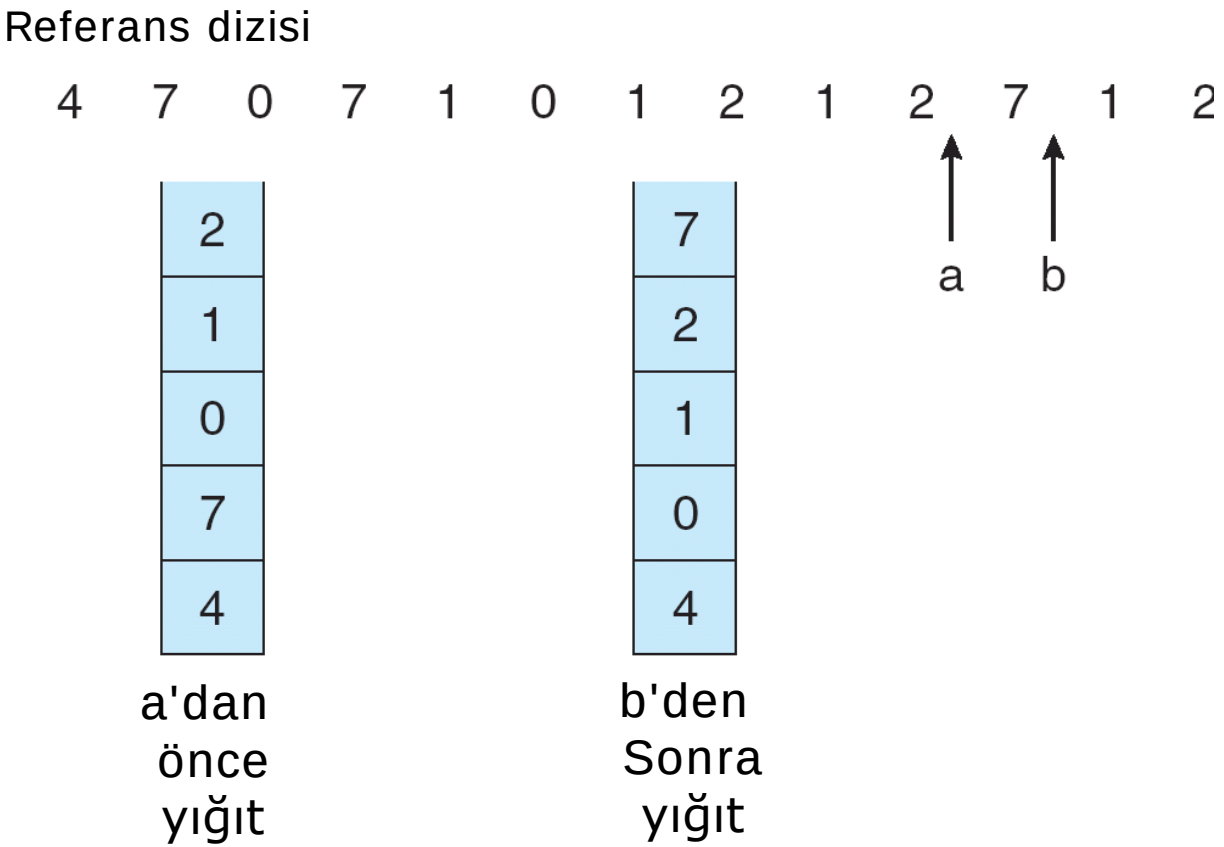
Sayfa çerçeveleri

LRU Algoritması

- Sayaç uygulaması
  - Her sayfanın bir sayacı vardır; sayfa girdi tarafından referans edildiği her zaman saat sayaca kopyalanır
  - Sayfanın değiştirilmesi gerekiyorsa, sayaçlara bakılarak hangisinin değiştirileceğine karar verilir

- Yığıt uygulaması – sayfa numaralarının bulunduğu çift bağlı formda bir yığıt (stack) tututlur:
  - Referans gösterilen sayfa:
    - En üste taşı
    - 6 işaretçinin değiştirilmesi gerekir
  - Değiştirme için arama yok

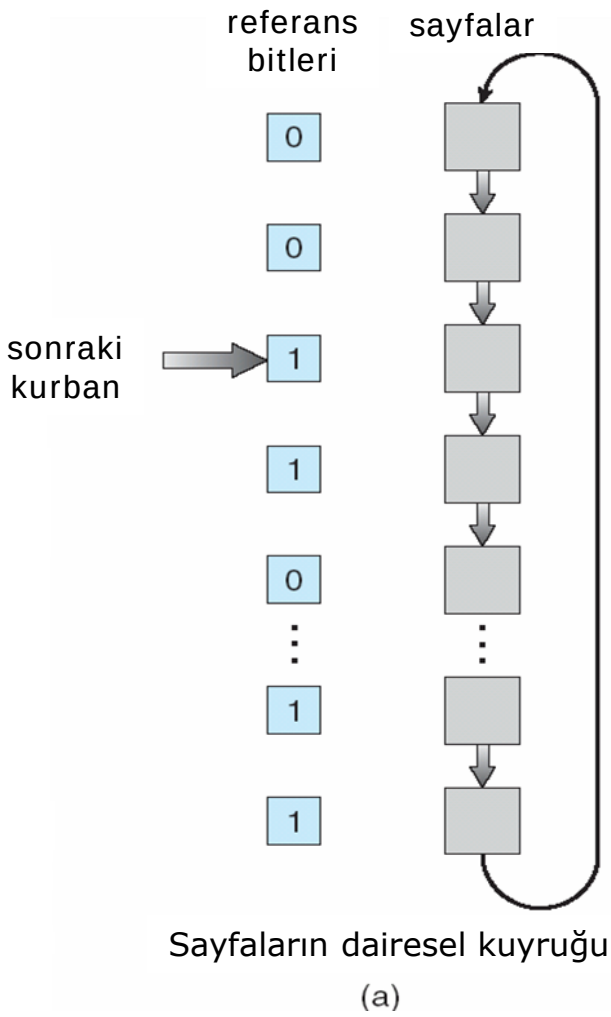
En Güncel Sayfa Referanslarını Kaydetmek İçin Yığıt Kullanımı

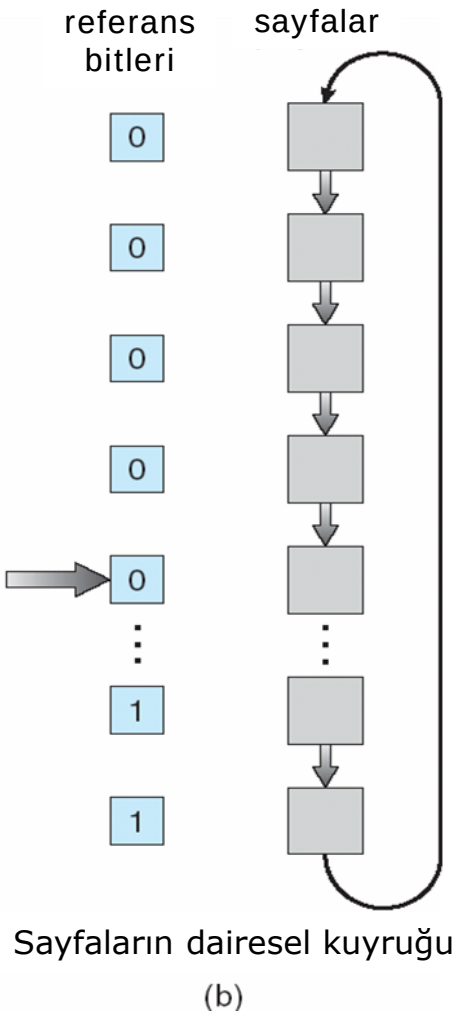


LRU-Yakınsama Algoritmaları

- Ek Referans Biti Algoritması
  - Her sayfa ile bir bit ilişkilendirilir, başlangıçta 0'dır.
  - Sayfaya referans yapılırsa bit 1 yapılır
  - 0 olan bit değiştirilir (eğer varsa)
    - Buna rağmen sırayı bilmiyoruz
- İkinci Şans Algoritması
  - Referans bitine ihtiyaç var
  - Saat kullanılır
  - Eğer değiştirilecek sayfada (saat sırası ile) referans biti = 1 ise:
    - Referans biti 0 yapılır
    - Sayfa bellekte bırakılır
    - Saat sırasında sonraki sayfa aynı kurallara bağlı olarak değiştirilir

İkinci Şans (saat) Sayfa Değiştirme Algoritması





Sayma algoritmaları

- Her sayfaya yapılan referans sayısı bir sayaçta tutulur
- **LFU (Least Frequently Used) Algoritması:** en düşük sayılı sayfayı değiştir
- **MFU (Most Frequently Used) Algoritması:** en düşük sayılı sayfanın getirildiği ve hala kullanıldığı varsayımına dayanır

Çerçevelerin Tahsisi

Minimum Çerçeve Sayısı

- Her süreç *minimum* sayıda sayfaya ihtiyaç duyar
- Örnek: IBM 370 – SS MOVE komutunun işlenmesi için 6 sayfaya ihtiyaç vardır:
  - komut 6 byte'tır, 2 sayfa kaplayabilir
  - 2 sayfa *nereden* 'i (from) işlemek için
  - 2 sayfa da *nereye* 'yi (to) işlemek için
- İki ana tahsis şeması vardır
  - Sabit tahsis (fixed allocation)
  - Öncelik tahsisi (priority allocation)

Sabit Tahsis (Fixed Allocation)

- Eşit tahsis – Örneğin, eğer 100 çerçeve ve 5 süreç varsa, her sürece 20 çerçeve verilir.
- Orantılı tahsis (Proportional allocation) – Sürecin boyutuna göre tahsisin yapılması
- Örneğin: 1KB çerçeve boyutu olan bir sistemde, 10 KB'lık ve 127 KB'lık iki süreç olduğunu ve 62 boş çerçeve olduğunu düşünelim. Herbirine 31'er çerçeve veremeyiz çünkü birinci süreçte 21 adet çerçeve boşa gider. Bu durumda orantılı tahsis kullanılır

$s_i = p_i$ sürecinin boyutu

$S = \sum s_i$

$m =$ çerçeveler toplam sayı

$a_i = p_i = \frac{s_i}{S} \times m$ için tahsis

$m = 62$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 5$

$a_2 = \frac{127}{137} \times 62 \approx 57$

Öncelik Tahsisi (Priority Allocation)

- Boyut yerine öncelikleri kullanan bir orantılı tahsis şeması kullanılır
- Eğer P_i süreci bir sayfa hatası yaratırsa,
  - Bu çerçevelerden biri değiştirilmek için seçilir
  - Daha düşük öncelik sayısına sahip süreçlerden bir çerçeve değiştirmek için seçilir

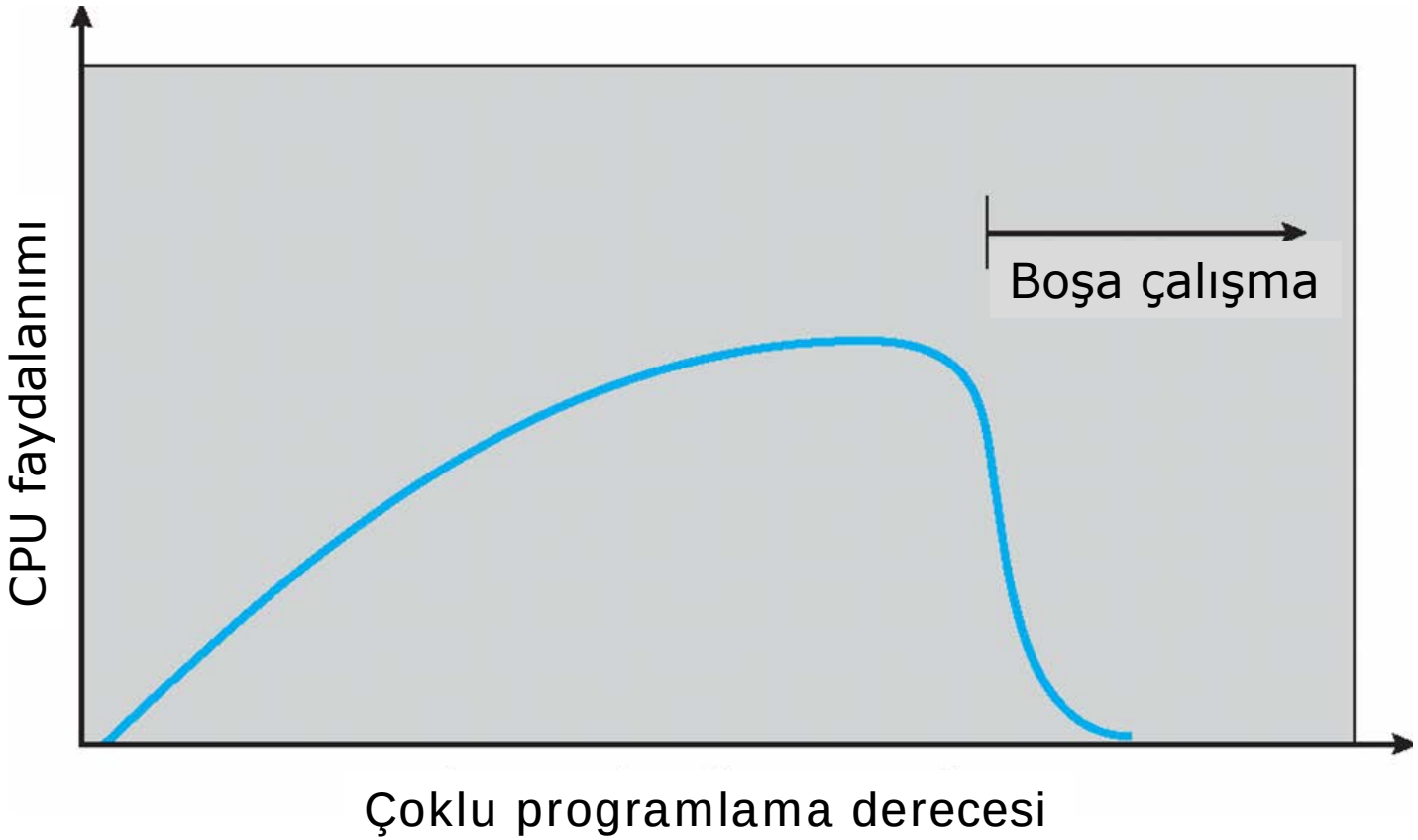
Global – Yerel Tahsis

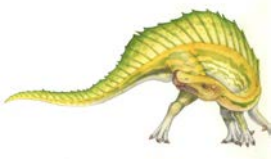
- **Global değiştirme** – süreç mevcut çerçevelerin arasından değiştirmek üzere bir çerçeve seçer; bir süreç diğerinden bir çerçeve seçebilir
- **Yerel değiştirme** – her süreç sadece kendi tahsis edilen çerçeveleri arasından bir çerçeve seçer

Boşuna Çalışma (Thrashing)

- Eğer bir sürecin "yeterli" miktarda sayfası yoksa, sayfa hata oranı çok yüksek olur. Bu durumda:
  - Düşük CPU faydalanımı olur
  - İşletim sistemi çoklu programlama derecesini arttırması gerektiğini düşünür
  - Sisteme diğer bir süreç eklenir
- **Boşa çalışma (Thrashing)** ≡ bir süreç sürekli sayfaları değiş tokuş etmekle meşguldür. Bir süreç sayfalamaya; çalışmadan daha çok zaman harcıyorsa bu duruma boşa çalışma denir.

Boşa Çalışma

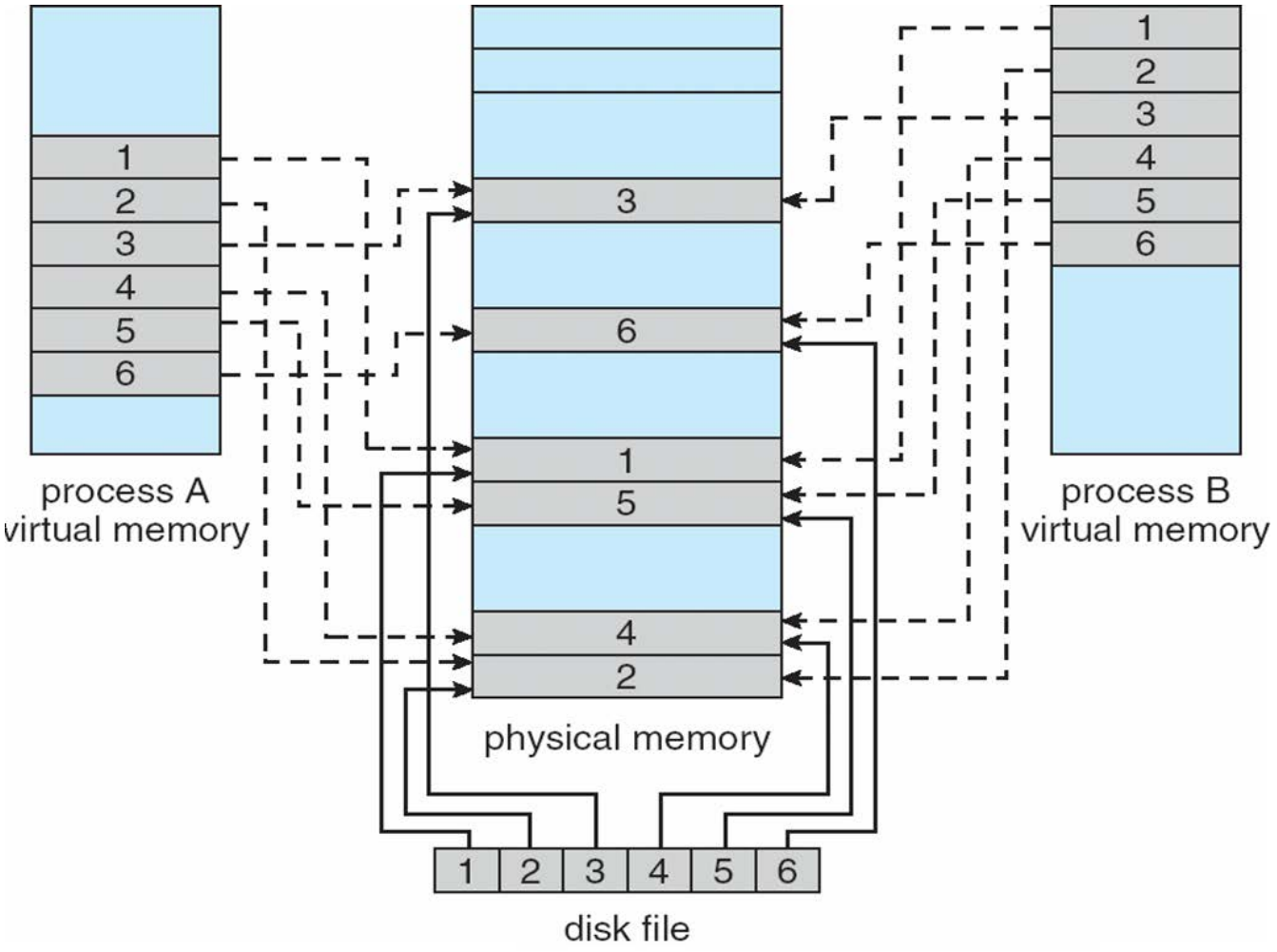




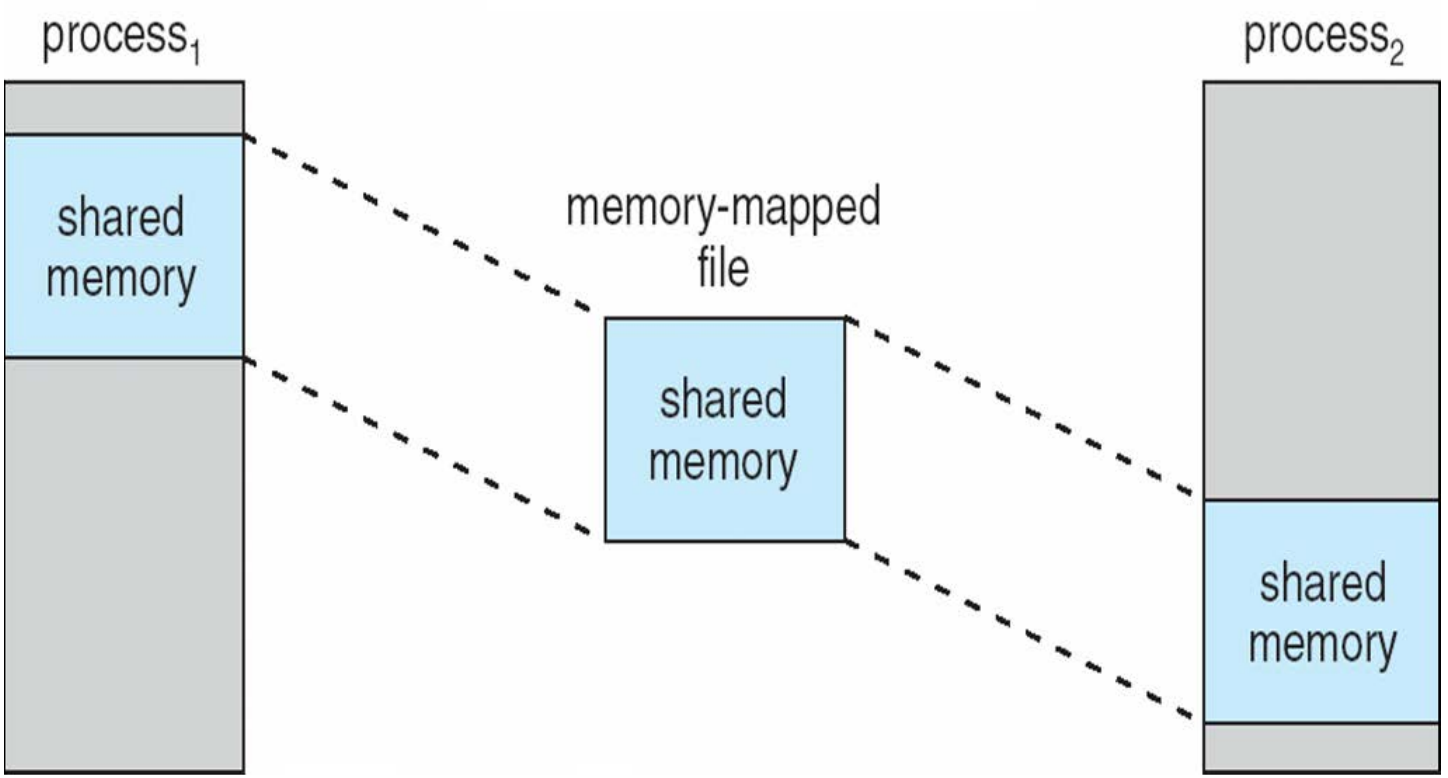
Bellekte Haritalanan Dosyalar

- Bellekte haritalanan bir dosyanın G/Ç'ı, dosya G/Ç'nın (bir disk bloğunun bellekteki bir sayfaya eşlenmesi gibi) rutin bir bellek erişimi olarak ele alınmasına izin verir.
- Bir dosya başlangıçta istek sayfalama kullanılarak okunur. Dosyanın bir sayfa boyutu kadar olan kısmı dosya sisteminden fiziksel sayfaya okunur. Sonraki dosyadan/dosyaya okuma/yazmalar sıradan bellek erişimi olarak ele alınırlar.
- **`read()` `write()`** sistem çağrıları yerine dosya G/Ç'nı bellek aracılığıyla gerçekleştirerek dosya erişimini basitleştirir.
- Bellekteki sayfaların paylaşılmasına izin vererek farklı süreçlerin aynı dosyayı kapsamasına izin verir.

Bellekte Haritalanan Dosyalar



Windows'da Bellekte Haritalanan Paylaşılan Bellek

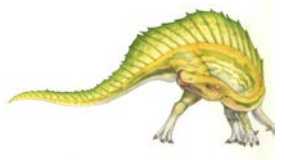


Çekirdek Belleğinin Tahsisi

- Kullanıcı belleğinden faklı olarak ele alınır
- Genellikle bir boş-bellek havuzundan tahsis edilir
  - Çekirdek değişik boyutlarda yapılar için bellek ister
  - Bazı çekirdek bellek kısımları ardışık olmak zorundadır

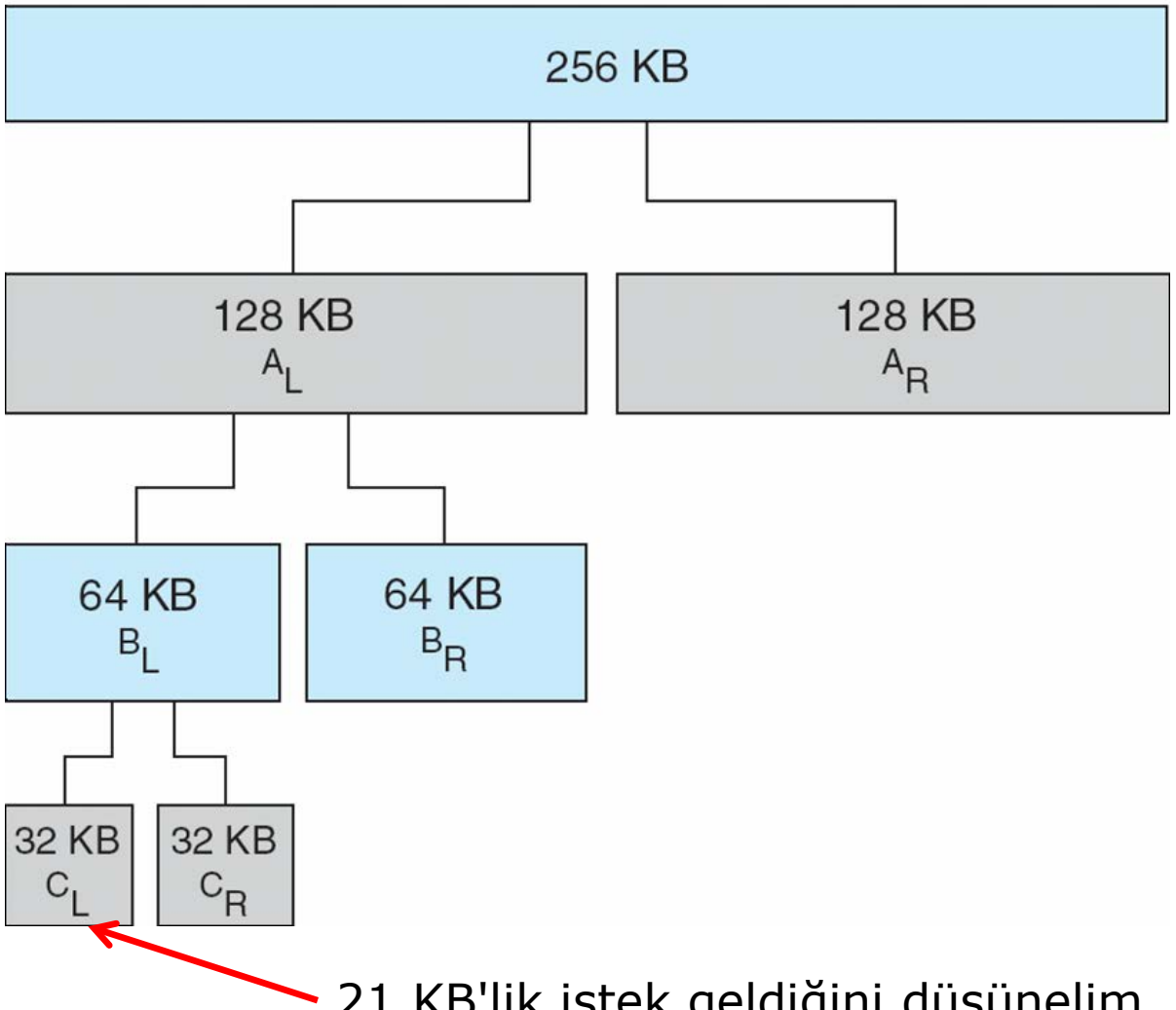
Arkadaş Sistem (Buddy System)

- Fiziksel olarak ardışık sayfalardan oluşan sabit boyutlu segmentlerden oluşan bellek tahsisi
- Bellek **2'nin üsleri tahsis edici** kullanılarak tahsis edilir
  - İstekleri 2'nin üsleri boyutundaki birimler aracılığıyla karşılar
  - İstek bir sonraki 2'nin üstüne yuvarlanır
  - Eğer ihtiyaç duyulandan mevcuttan daha az ise yığın 2'nin bir küçük üssü boyutlu olacak şekilde iki arkadaşa bölünür
    - Bu şekilde uygun boyutlu yığın mevcut oluncaya kadar devam edilir



Arkadaş Sistem Tahsis Edici

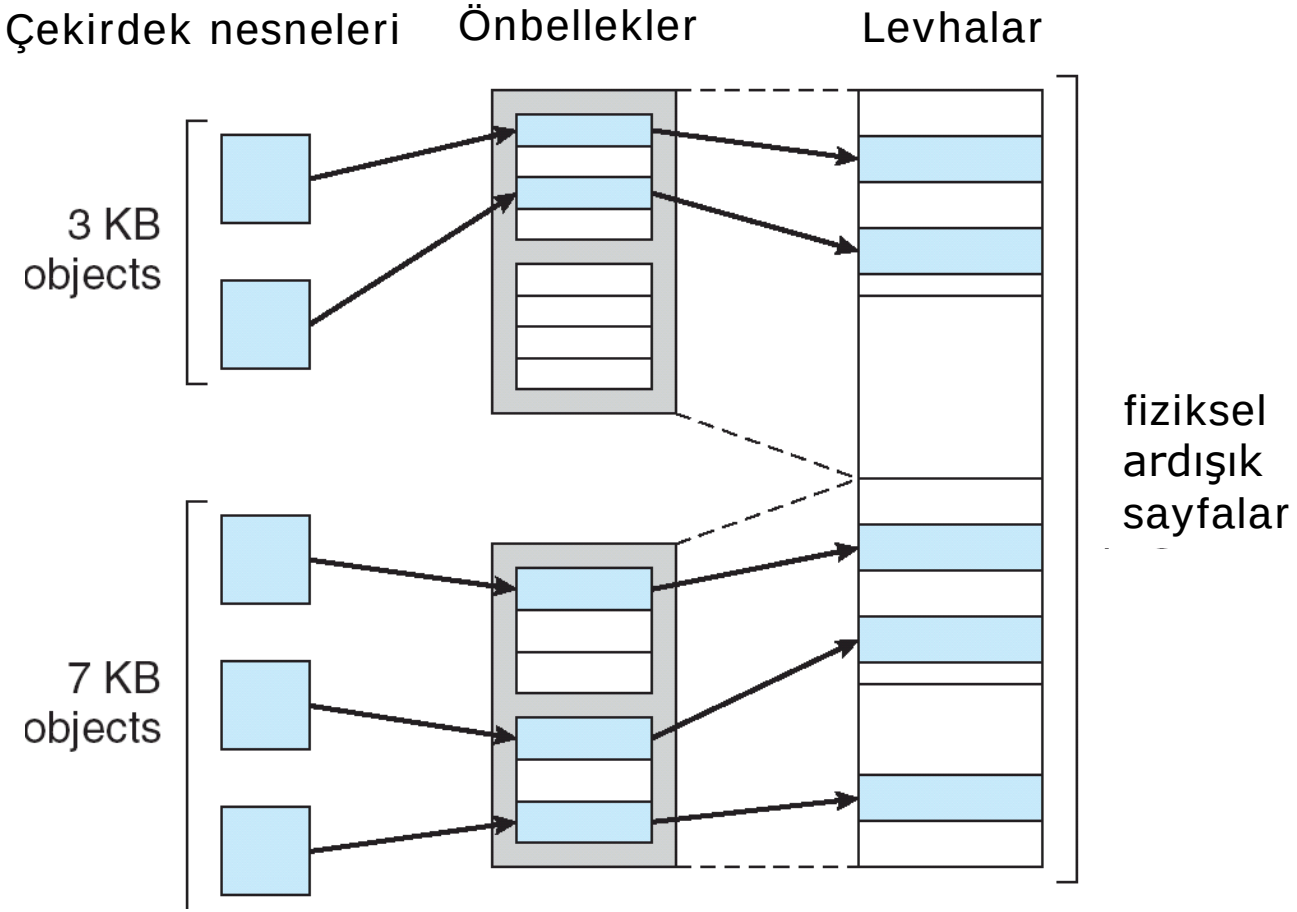
Fiziksel olarak ardışık sayfalar



Levha Tahsis Edici (Slab Allocator)

- Alternatif bir strateji
- **Levha (Slab)** bir veya daha fazla fiziksel olarak ardışık sayfalardır
- **Önbellek (Cache)** bir veya daha fazla levhadan oluşur
- Her çekirdek veri yapısı için tek önbellek
  - Her önbellek **nesneler (objects)** ile doludur – veri yapısının başlatımları (ör., semafor, dosya nesneleri, vb.)
- Önbellek yaratıldığında (nesnelerle dolu olarak) **boş (free)** olarak işaretlenir
- Yapılar kaydedildiğinde ise, nesneler **kullanıldı (used)** olarak işaretlenir
- Eğer levha kullanılmış nesneler ile dolu ise, bir sonraki nesneye boş levhadan tahsis gerçekleşir
  - Boş levha yoksa, yeni bir levha tahsis edilir
- Faydaları arasında parçalanma (fragmentation) olmaması, ve hızlı bellek istek memnuniyeti vardır

Levha Tahsisi (Slab Allocation)



Diğer Konular – Önceden Sayfalama (Prepaging)

- Önceden sayfalama (Prepaging)
  - Süreç başlangıçlarında oluşan büyük sayıdaki sayfa hatalarını azaltmak içindir
  - Bir sürecin ihtiyaç duyacağı bütün veya bazı sayfaları bunlara referans yapılmadan önce sayfalamak
  - Ancak bu önceden sayfalanan sayfalar kullanılmazsa, G/Ç ve bellek boşuna kullanılmış olur
  - Düşünelim ki *s* adet sayfa önceden sayfalanmış ve bu sayfalardan *α* yüzdesi kullanılmış
    - *s * α* adet sayfa hatasından kurtulmanın maliyeti, *s * (1- α)* adet gereksiz sayfayı önceden sayfalamanın maliyetinden büyük müdür yoksa küçük müdür?
    - *α* sıfıra yakınsa ⇒ önceden sayfalama kaybeder
    - *α* 1'e yakınsa ⇒ önceden sayfalama kazanır

Diğer Konular – Sayfa Boyutları

- Sayfa boyutları (4,096 (2^{12})'den 4,194,304 (2^{22}) byte) arasındadır:
  - Parçalanma (fragmentation)
  - Tablo boyutu
  - G/Ç maliyeti (I/O overhead) ve
  - Lokalite dikkate alınmalıdır

Diğer Konular – TLB Erimi (Reach)

- TLB Erimi – TLB tarafından erişilebilir olan bellek miktarıdır
- TLB erimi = (TLB Boyutu) X (Sayfa Boyutu)
- İdeal olarak her sürecin çalışma kümesi TLB'de tutulmalıdır
  - Diğer durumda büyük derecede sayfa hatası olur
- Sayfa Boyutunu Büyütürsek
  - Bu parçalanmada bir artışa sebep olabilir, çünkü tüm uygulamalar büyük sayfa boyutuna ihtiyaç duymazlar
- Farklı Sayfa Boyutları Sağlamak
  - Daha büyük sayfa boyutları talep eden uygulamaların parçalanmaya sebep olmadan bunu kullanmalarına izin verir

Diğer Konular – Program yapısı

- Program yapısı
  - `Int[128,128] data;`
  - Her satır bir sayfada tutulur, eğer 128 çerçeveden az bir yer tahsis edlirse:
  - Program 1 (i satır, j sütun)

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 sayfa hatası

  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

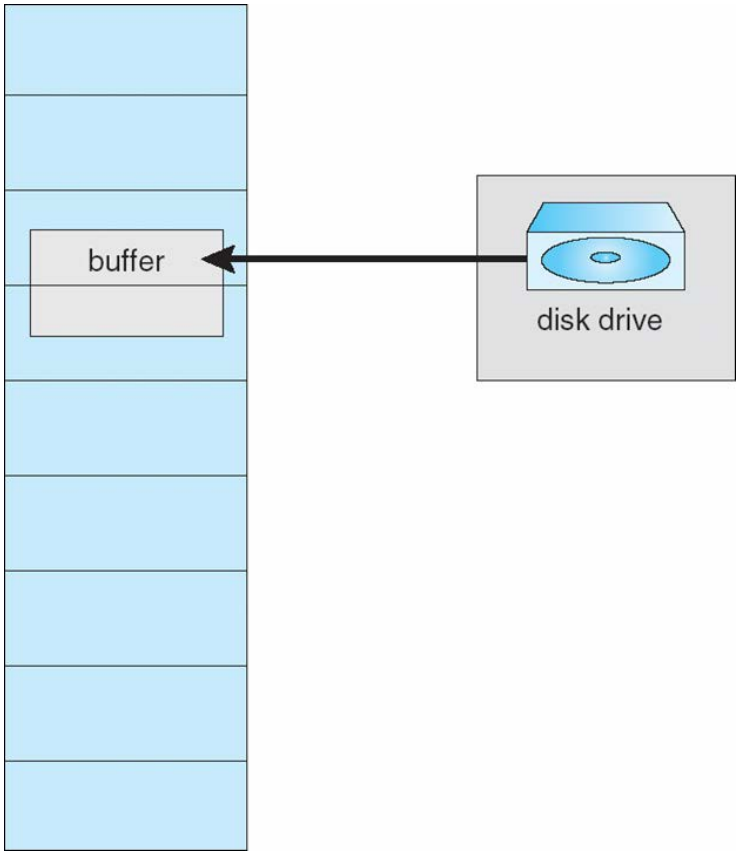
128 sayfa hatası

Diğer Konular – G/Ç Kilitleri (interlock)

- **G/Ç Kilitleri (I/O Interlock)** – Sayfaların bazen bellekte kilitlenmeleri gereklidir

- G/Ç'ı düşünelim – Bir dosyayı bir aygıttan kopyalamak için kullanılan sayfaların bir sayfa değiştirme algoritması tarafından seçilmesi engellenecek şekilde kilitlenmesi gereklidir.

G/Ç için Kullanılan Çerçeveler Niçin Bellekte Olmalıdır?



Bölüm 9 Son