



# Bölüm 8: Bellek Yönetimi

## İşletim Sistemleri



# Bellek Yönetimi

- Bir programın çalışabilmesi için;
  - diskten belleğe getirilmesi ve
  - bir süreç içerisine yerleştirilmesi gereklidir.
- Ana bellek ve yazmaçlar, CPU'nun doğrudan erişebildiği alanlardır.
- Bellek birimi şu akışları görür:
  - adres + okuma isteği,
  - adres + veri + yazma isteği.
- Yazmaç erişimi bir CPU döngüsünde yapılır.
- Ana bellek erişimi, duraklamaya neden olacak kadar birçok döngü alabilir.
- Ön bellek (*cache*), ana bellek ile yazmaçlar arasında bulunur.



# Bellek Yönetimi

- Bellek, kısıtlı bir kaynak.
- Bu nedenle bellek hiyerarşisi var.
  - Önbellek (*cache*)(**hızlı**)
  - Ana bellek
  - Disk (**yavaş**)
- Bellek yönetici, bellek soyutlaması yapmak için *hiyerarşi* kullanır.



# Bellek Yönetimi

- Bellek (RAM) önemli ve kısıtlı bulunan bir kaynaktır.
  - Programlar, genişleyerek kendilerine sunulan belleği doldururlar.
- Programcının istediği,
  - Belleğin korumalı, sonsuz büyük, sonsuz hızlı, ve kalıcı olması..😊
- Gerçekte olan,
  - Akla gelen en iyi çözüm: bellek hiyerarşisi. 😞
  - Yazmaç, önbellek, bellek, disk, teyp.
- Bellek yöneticisi,
  - Belleği verimli bir şekilde yönetir.
  - Serbest bırakılan bellek alanlarını takip eder.
  - İhtiyaç halinde programlara tahsis eder.



# Soyutlama Olmazsa

- Eskiden bilgisayarlarda bellek soyutlaması yoktu... 😞
  - MOV REGISTER1, [1000]
  - Fiziksel bellek adresi 1000'in içeriğini yazmaca taşıır.
- Bellekte aynı anda iki süreç yer alamaz. Neden?



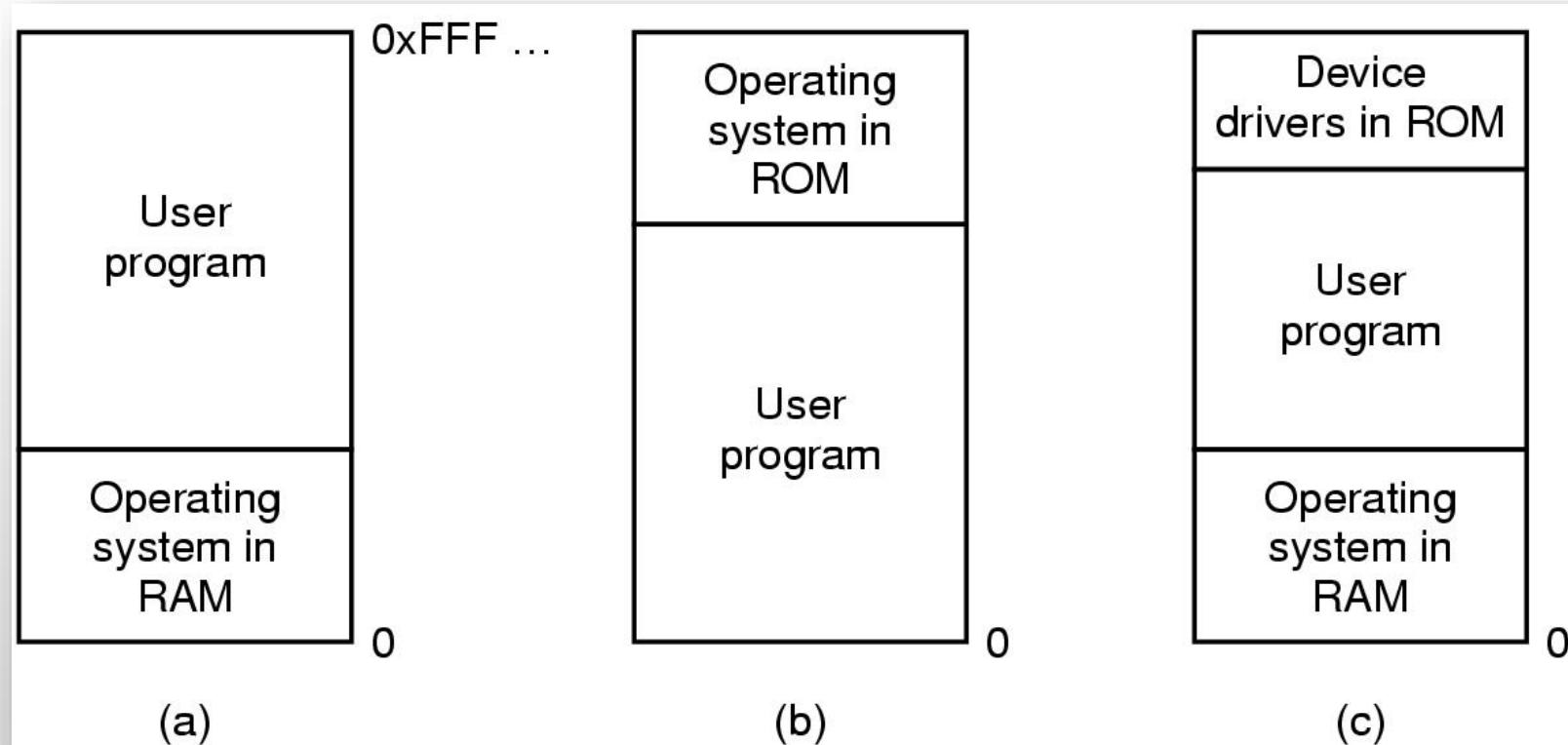
# Soyutlama Olmazsa

- Eskiden bilgisayarlarda bellek soyutlaması yoktu... 😞
  - MOV REGISTER1, [1000]
  - Fiziksel bellek adresi 1000'in içeriğini yazmaca taşıır.
- Bellekte aynı anda iki süreç yer alamaz. Neden?
- Fiziksel adresler kullanıldığı için,
  - Süreçler aynı adreslere erişebilirler.
  - O anda hangi süreç hangi adresi kullanıyor bilinemez.



# Soyutlama Olmazsa

- İşletim sistemi ve tek bir süreç ile belleğin düzenlenmesi.



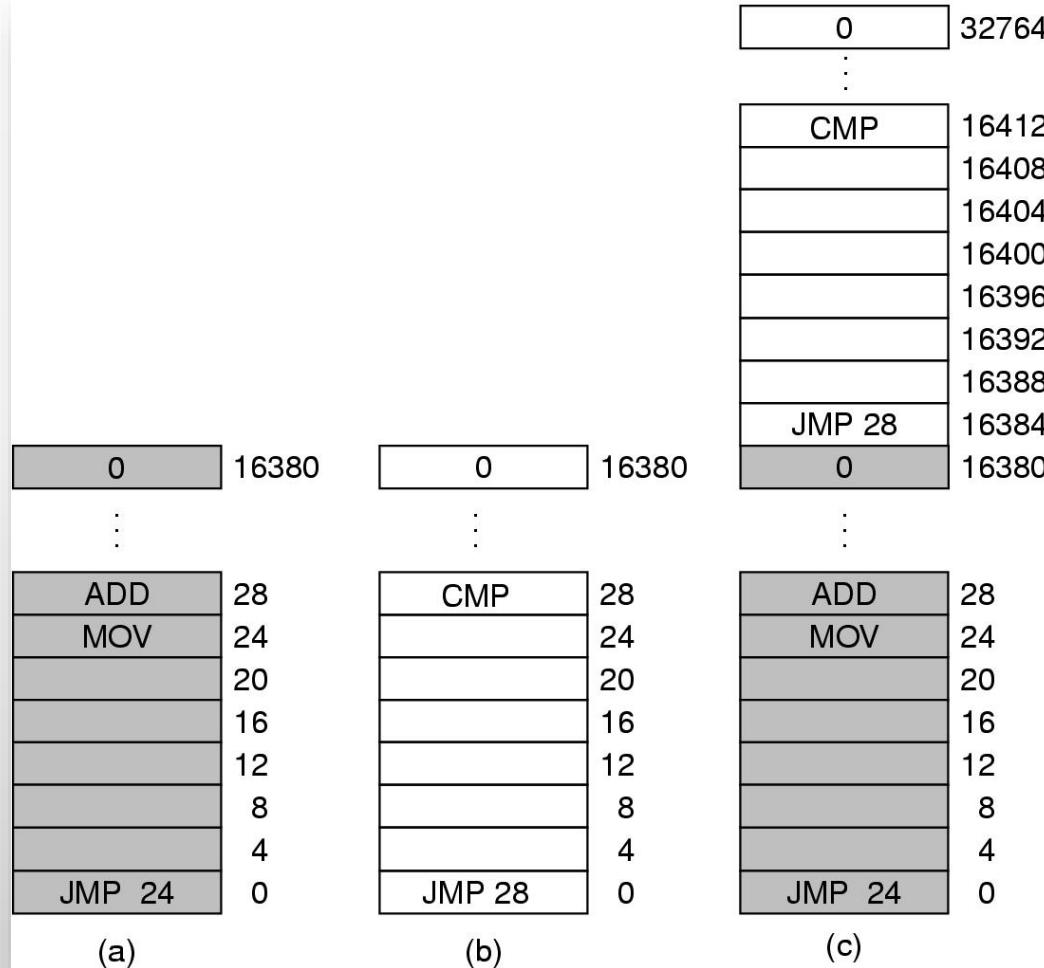


# Soyutlama Olmazsa

- Bellekte aynı anda yalnızca bir süreç olabilir.
- Kullanıcı programındaki hata işletim sistemini çökertebilir (a ve c).
- Bazı gömülü sistemlerde salt okunur bellekte tutulabilir (b).
- MS-DOS (c) – işletim sistemi bellekte, BIOS salt okunur bellekte tutulur.



# Yer Değiştirme (Relocation) Problemi





# Statik Yer Değiştirme

- Sorun, her iki programın da *mutlak fiziksel belleğe* referans vermesidir.
- Statik yer değiştirme:
  - Programın ilk komutu x adresine yüklenir,
  - Yükleme sırasında sonraki her adrese x değeri eklenir.
  - Çok maliyetli ve yavaş bir işlem.
  - Tüm adresler değiştirilemez.
    - MOV REGISTER 1,28 değiştirilemez.



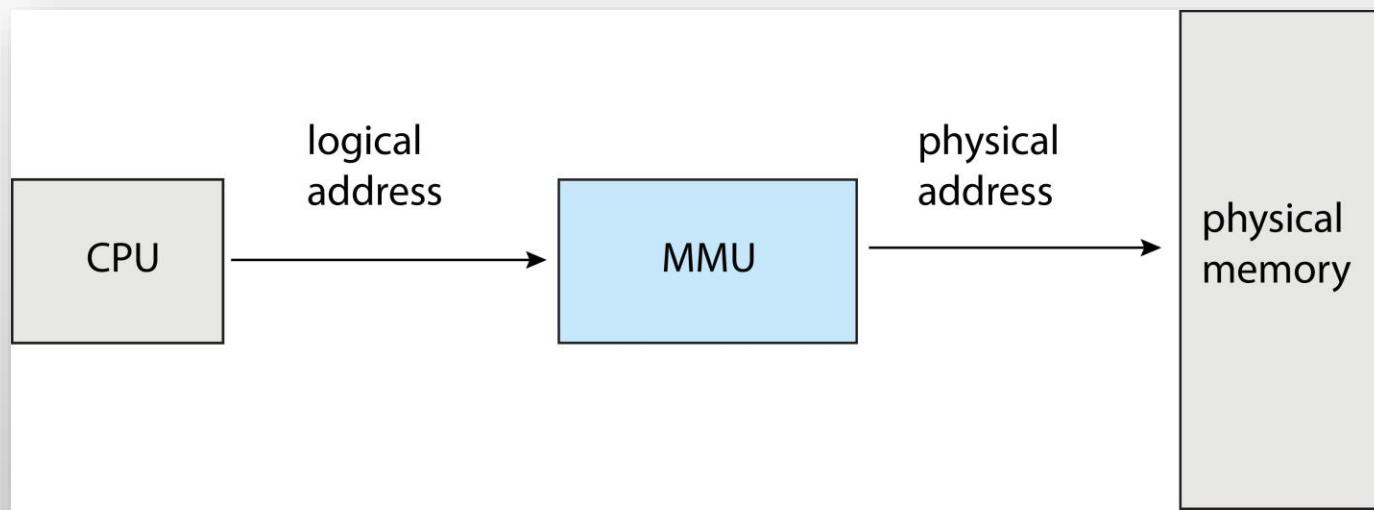
# Soyutlama Olmamasının Dezavantajı

- Sorun, her iki programın da mutlak fiziksel belleğe referans vermesidir.
- İnsanlar mahrem bir alana, programlar yerel adreslere sahip olmak ister.
- IBM 360
  - İkinci program belleğe yüklenirken adresler değiştirilir.
  - Statik yer değiştirme (*static relocation*).
    - 16384'e bir program yüklenirken, her adrese bir sabit değer eklenir.
    - Yüklemeyi yavaşlatır, ek bilgi gerektirir.
- Gömülü sistemlerde soyutlama olmadan bellek yönetimi var.



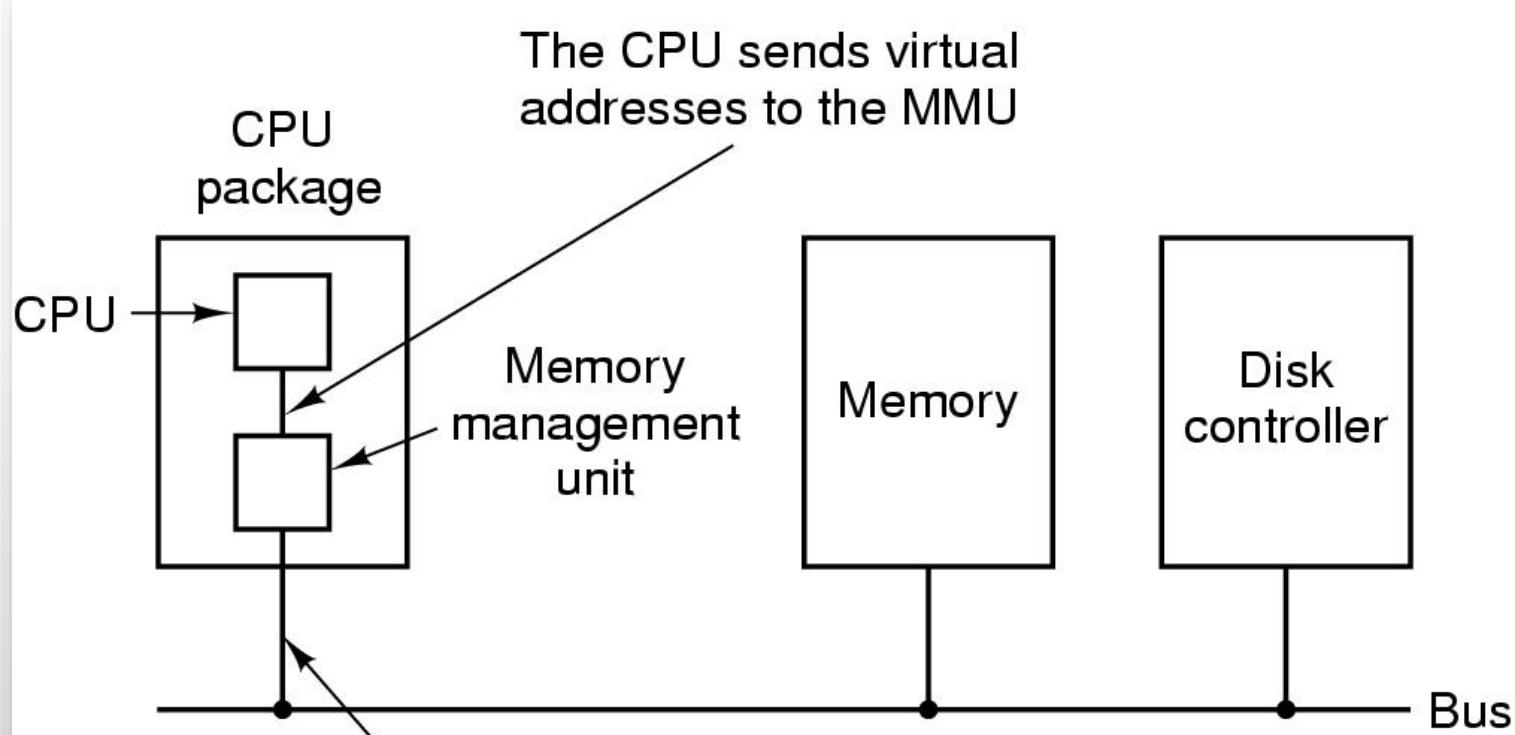
# Bellek Yönetim Birimi

- MMU (*memory management unit*) mantıksal adresi fiziksel adrese çevirir.
- Mantıksal ve fiziksel adresler,
  - Derleme ve yükleme aşamasında aynıdır.
  - Yürütme zamanında, adres eşleme şemasında farklılık gösterir.





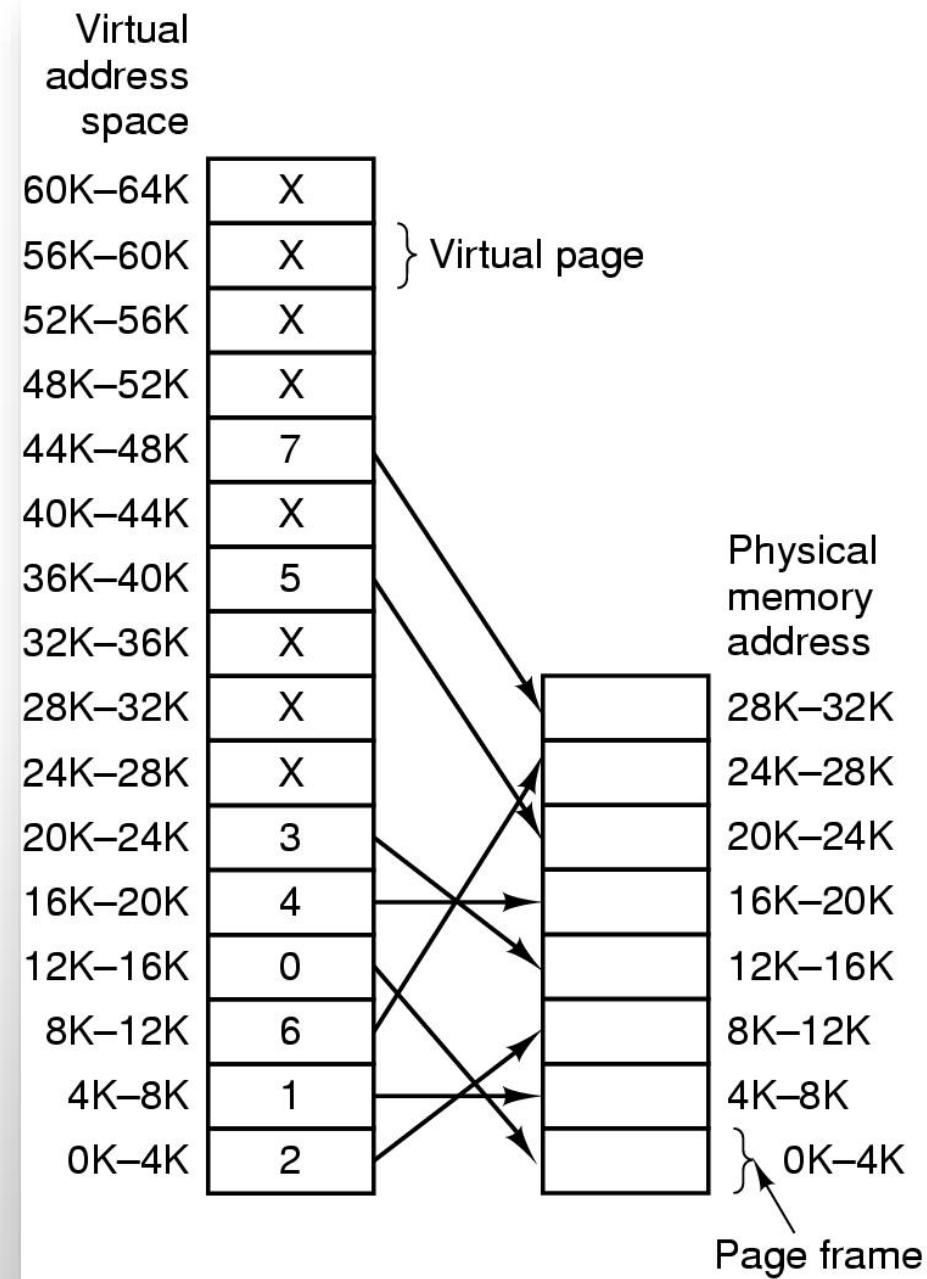
# MMU Konum ve İşlevi





# Bellek Yönetim Birimi

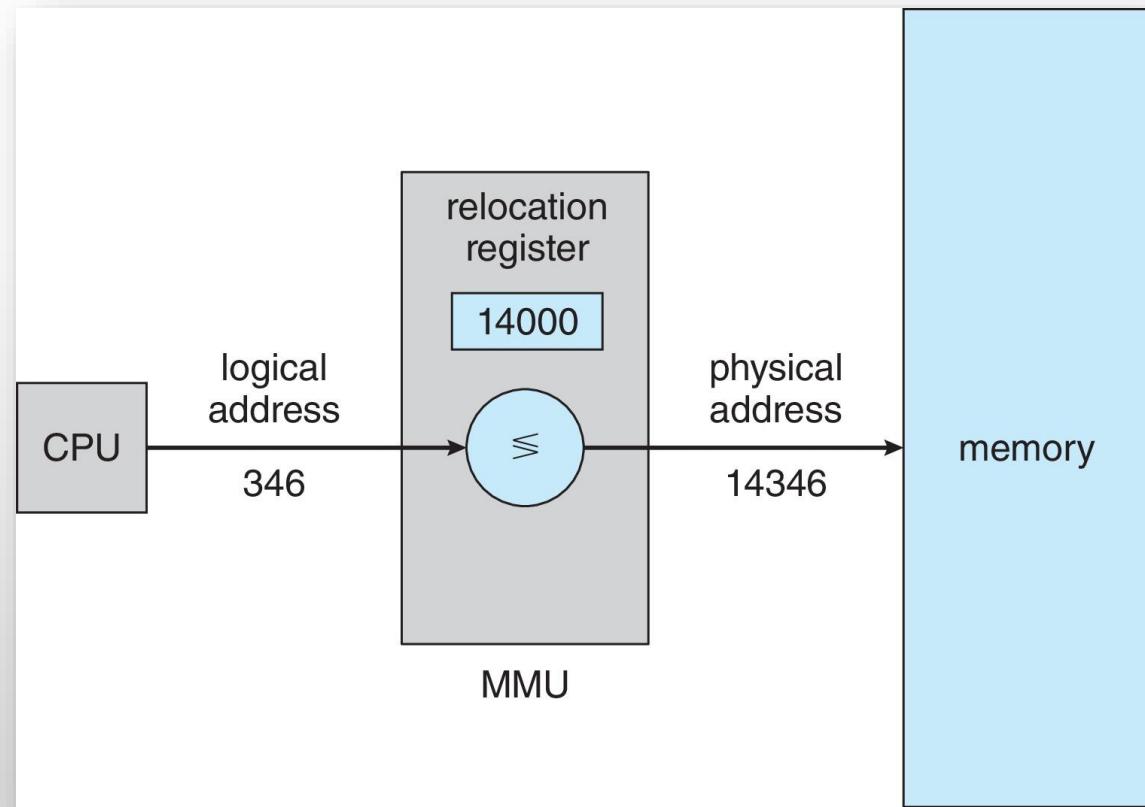
- MMU (*memory management unit*)
  - CPU: MOV REG, 0
  - MMU: MOV REG, 8192
  - CPU: MOV REG 8192
  - MMU: MOV REG 24567
  - CPU:MOV REG 20500
  - MMU:MOV REG 12308
  - CPU: MOV REG 32780
  - MMU: page fault





# Bellek Yönetim Birimi

- Taban yazmacı (*base register*) yer değiştirme yazmacı (*relocation register*) olarak adlandırılıyor.





# Adres Uzayı

- Süreç için kendisine ait soyut bellek alanı oluşturulur.
- Her sürecin kendi adres kümesi vardır.
- Adresler her süreç için farklıdır.



# Soyutlama: Adres Uzayı

- Fiziksel adres programcılara gösterilmez (*not expose*).
  - İşletim sistemi çökertilebilir.
  - Süreçleri paralel yürütmek zor.
- Çözülmesi gereken iki problem:
  - Koruma,
  - Yer değiştirme.
- Adres alanı:
  - Bir dizi süreç belleği kullanabilir.
  - Her sürecin, birbirinden bağımsız, kendi adres alanı vardır.



# Dinamik Yer Değiştirme

- İşlemciye iki özel yazmaç gereklidir:
  - taban ve limit yazmaçları (*base and limit registers*).
- Program ardışık boş alanlara yüklenir.
- Yükleme sırasında yer değiştirme yok.
- Süreç bir adrese erişmek istediğiinde,
  - CPU otomatik olarak *limit* değerini kontrol eder.
  - *Taban* değerini, adres değerine ekler.



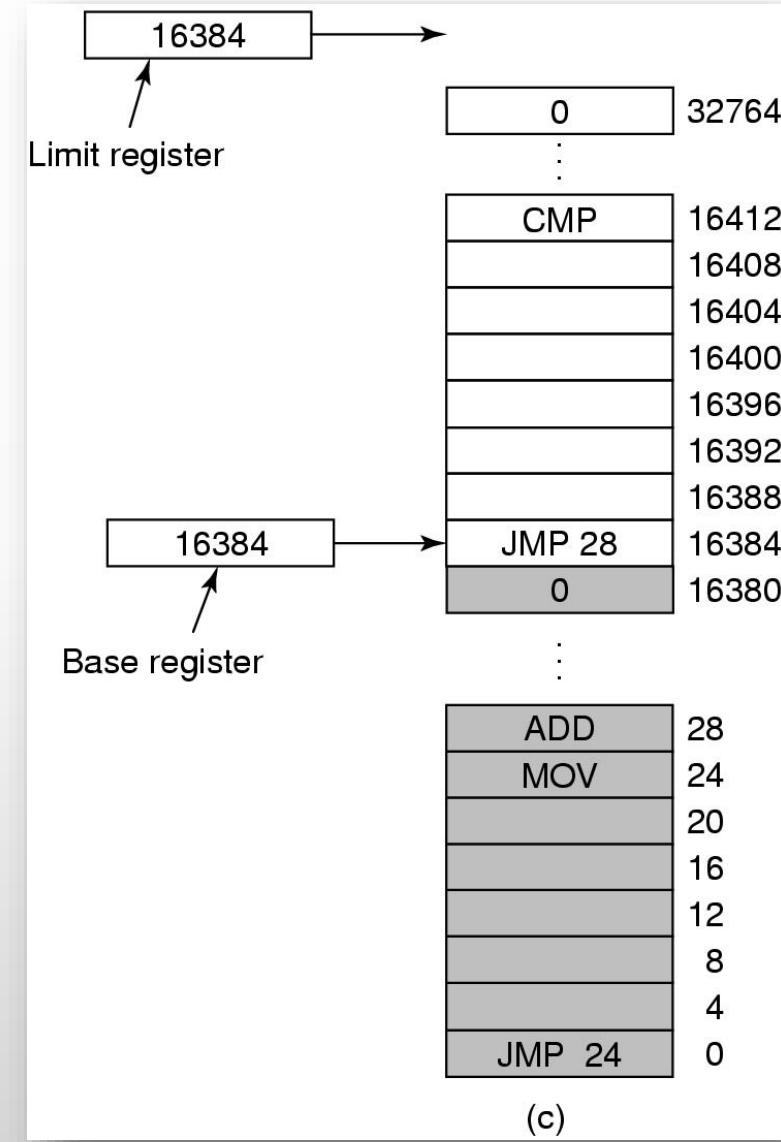
# Taban ve Limit Yazmaçları

- Bir tür dinamik yer değiştirme.
- *Taban*, programın başlangıç adresini içerir.
- *Limit*, programın uzunluğunu belirtir.
- Süreç belleğe erişeceğinde, erişilecek adrese *taban* adresi eklenir.
- Adresin *limit* değerinden büyük olup olmadığı kontrol edilir.
  - Büyükse, hata oluşturulur.
- Her adımda ekleme ve karşılaştırma yapılması maliyetlidir.
- *CDC 6600* ve *Intel 8088*'de kullanılır.



# Taban ve Limit Yazmaçları

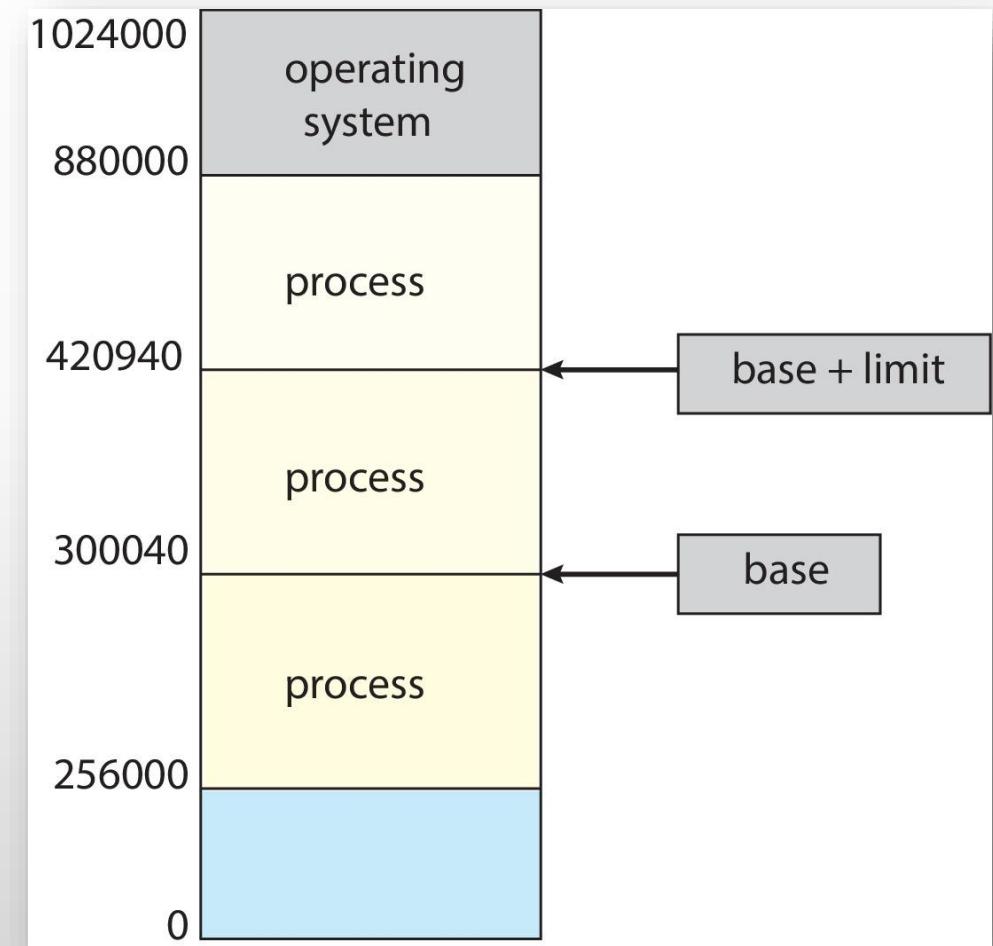
- Her bellek erişiminde bir ekleme ve karşılaştırma yapılması gerekiyor





# Taban ve Limit Yazmaçları

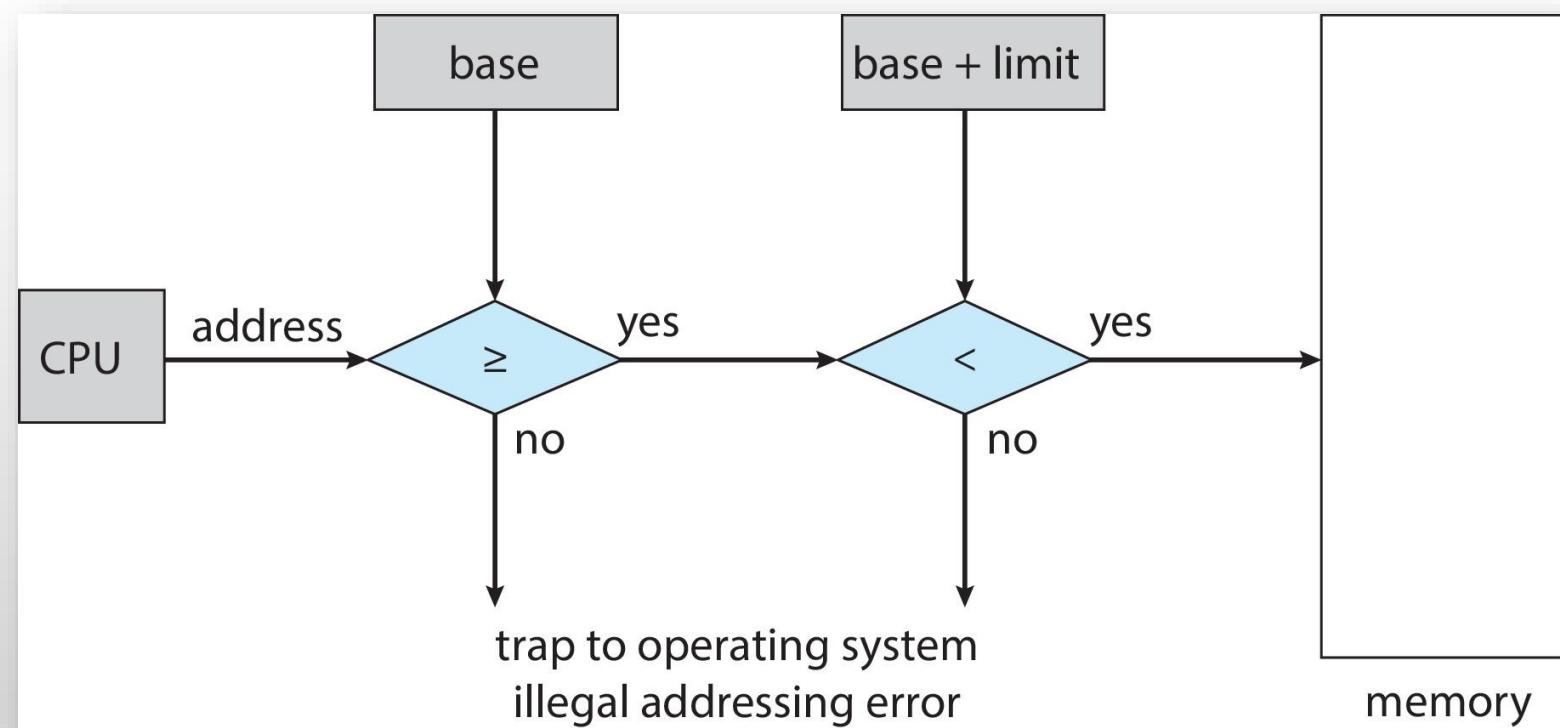
- Süreç yalnızca kendi adres uzayı içerisinde kalan alanlara erişebilir.
- Bu koruma *taban* ve *limit* yazmaçları kullanılarak sağlanır.





# Donanımsal Adres Kontrolü

- CPU, kullanıcı modunda iken, her bellek erişimini, *taban* ile *limit* arasında olup olmadığını kontrol etmelidir.





# Bellek Adresleri

- Komut ve verilerin bellek adreslerine eşlenmesi üç aşamada gerçekleşir.
- **Derleme zamanı:** Bellek konumu önceden biliniyorsa, mutlak (*absolute*) kod üretilebilir; başlangıç konumu değişirse kod yeniden derlenir.
- **Yükleme zamanı:** Derleme zamanında bellek konumu bilinmiyorsa, yeri değiştirilebilen kod üretilir.
- **Yürütme zamanı:** Süreç yürütülürken, belleğin bir bölümünden diğerine taşınabiliyorsa, adres eşlemesi yürütme zamanına kadar ertelenir.
- Adres eşlemeleri için donanım desteği gereklidir (taban ve limit yazmaçları).



# Dinamik Yükleme

- Programın yürütülebilmesi için bellekte olması gereklidir.
- Prosedür çağrılarına kadar belleğe yüklenmez.
- Daha iyi bellek alanı kullanımı sağlar.
  - Kullanılmayan prosedür asla belleğe yüklenmez.
- Tüm prosedürler, yeri değiştirilebilen yükleme biçiminde diskte tutulur.
- Büyük miktarda kodun yüklenmesi gerekiyinde kullanılmalıdır.
- Donanımsal özel bir destek gerekmez.
  - Program bu şekilde tasarlanır.
  - İşletim sistemi, dinamik yükleme için kütüphane sağlar.



# Dinamik Bağlama

- **Statik bağlama:** sistem kütüphaneleri ve program kodu, yükleyici tarafından ikili (*binary*) program imajında birleştirilir. (.obj, .a, .o)
- **Dinamik bağlama:** bağlama yürütme zamanına kadar ertelenir. (.so, .dll)
- **Stub** bellekte yerleşik kütüphane yordamını bulmak için kullanılır.
- Stub, altprogramın adresiyle kendisini yer değiştirir ve yordamı yürütür.
- İşletim sistemi, prosedürün bellek adresinde olup olmadığını kontrol eder.
- Adres alanında değilse, adres alanına ekler.
- Dinamik bağlama özellikle kütüphaneler için kullanışlıdır.
  - Paylaşımlı kütüphaneler (*shared library*).



# Paylaşımı Kütüphaneler

- Birden çok süreç tarafından ortak kullanılan kütüphaneler (ör. grafik).
- Her bir süreç için kütüphaneyi belleğe yüklemek maliyetlidir.
- Bunun yerine paylaşımı kütüphaneler (*shared library*) kullanılır.
- Unix bağlama (*link*): `ld *.o -lc -lm`.
- .o uzantılı dosyalar ile, m ve c kütüphanelerinde bulunan prosedürler, ikili çalıştırılabilir (*binary executable*) dosyaya dahil edilir.

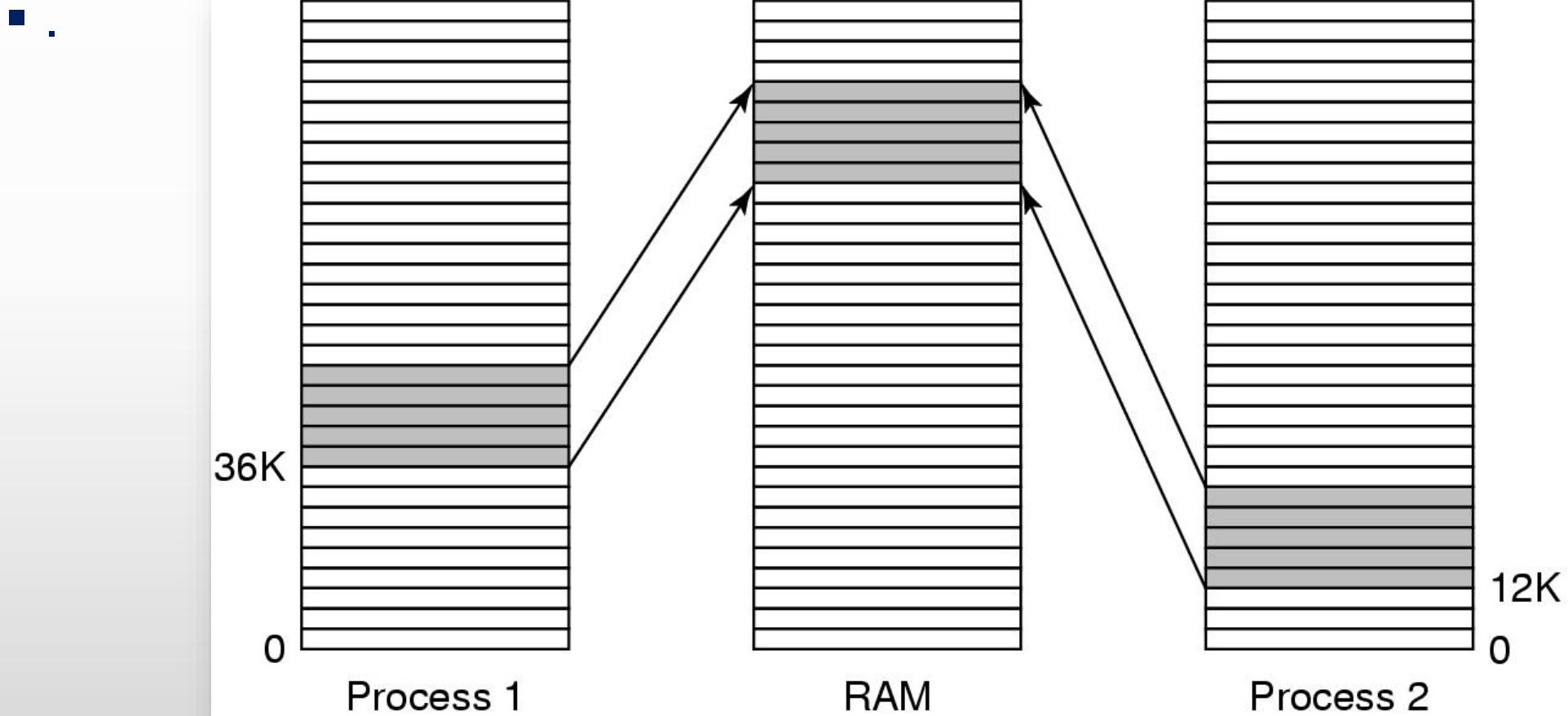


# Paylaşımı Kütüphaneler

- İlk kez içindeki bir prosedüre erişileceğinde yüklenir.
- Bağlayıcı (*linker*), yürütme zamanında, çağrılan prosedüre bağlanan bir *stub* yordamını çağrırmak için kullanır.
- Doğru adrese erişim için,
  - Konumdan bağımsız kod (*position independent code*) kullanılır.
  - Paylaşımı kütüphaneler için mutlak (*absolute*) adresler üretmez.
  - Yalnızca göreli (*relative*) adresleme yapılır.



# Paylaşımı Kütüphaneler





# Statik Kütüphane

- \$ gcc -c func.c -o func.o
- \$ ar rcs libfunc.a func.o
- \$ gcc main.c -o main -static -L. -lfunc
- \$ ./main



# Dinamik Kütüphane

- \$ gcc -fPIC -c func.c -o func.o
- \$ gcc -shared -o libfunc.so func.o
- \$ export LD\_LIBRARY\_PATH=\$(pwd)
- \$ ./main





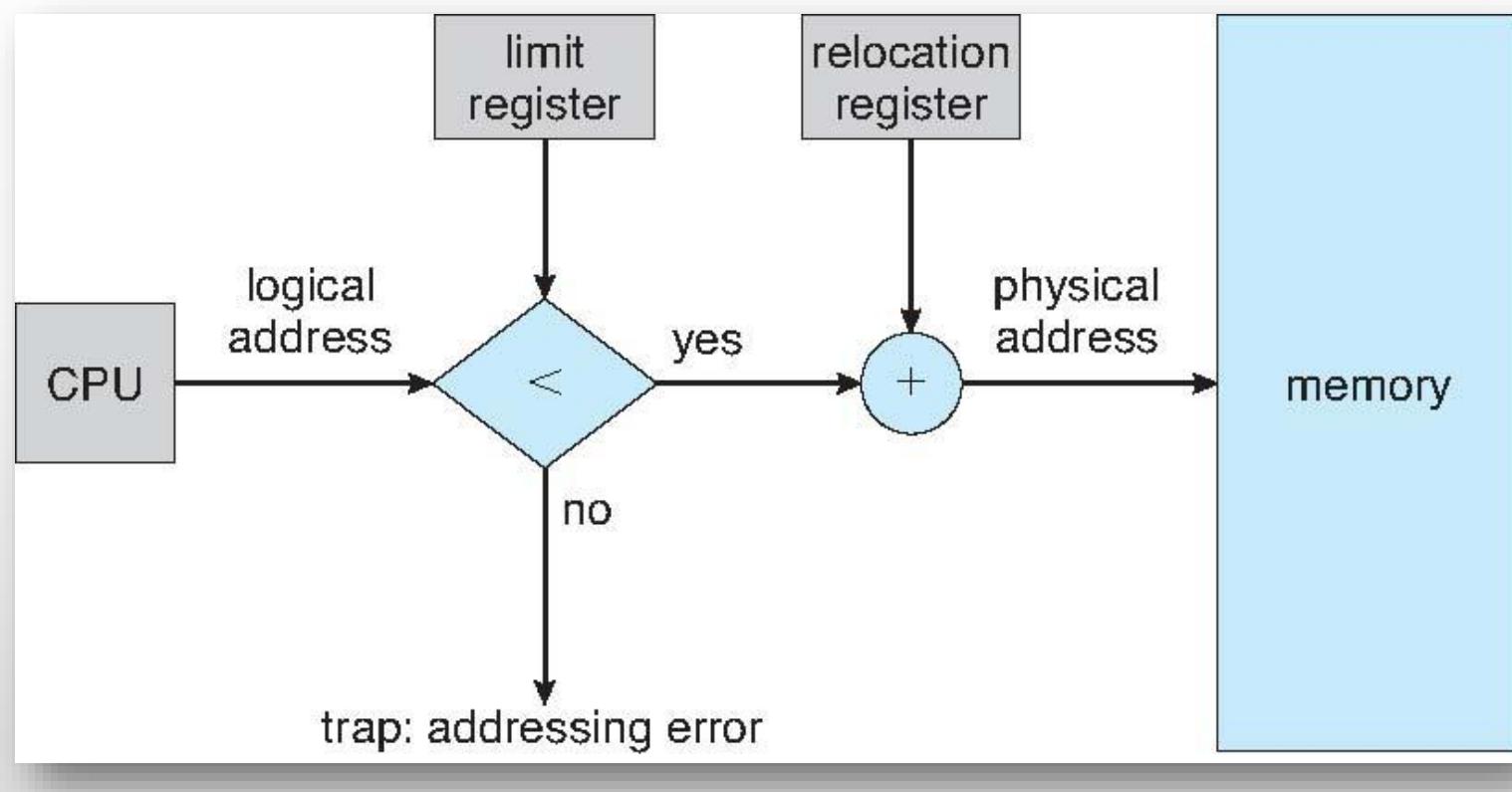
# Ardışık Yer Tahsisi

- Ana bellekte, hem işletim sistemi hem de kullanıcı süreçleri yer alır.
- Sınırlı kaynak, verimli bir şekilde tahsis edilmelidir.
- Ardışık yer tahsisi eski bir yöntemdir.
- Ana bellek genellikle iki bölüme ayrılır.
  - Yerleşik işletim sistemi, kesme vektörü ile belleğin alt kısmında tutulur.
  - Kullanıcı süreçleri belleğin üst kısmında tutulur.
- Her süreç bellekte tek bir ardışık (*bitişik, contiguous*) bölümde yer alır.



# Ardışık Yer Tahsisi

■





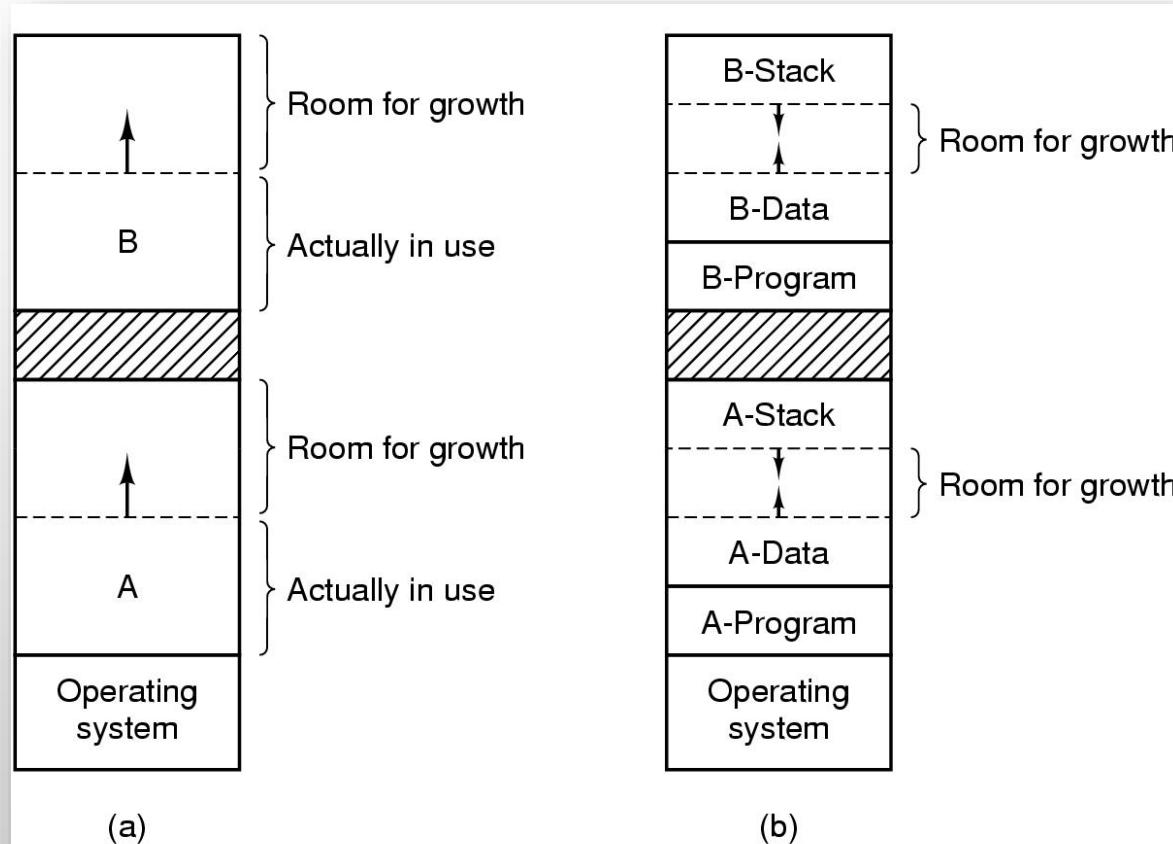
# Süreçler Koştukça Büyür

- **Yığın** kesimi (dönüş adresleri ve yerel değişkenler)
- **Veri** kesimi (dinamik olarak tahsis edilen ve serbest bırakılan değişkenler)
- Her iki kesim için, fazladan bellek ayırmak tavsiye edilir.
- Program diske gönderildikten sonra,
  - Belleğe tekrar getirilirken,
    - Boşluklar (*hole*) ile beraber getirilmez!
    - Diskte ardışık konumlarda saklanmayabilir.



# Bellekte Alan Tahsisi

- (a) Büyüyen veri kesimi. (b) Büyüyen yiğin, ve veri kesimi.





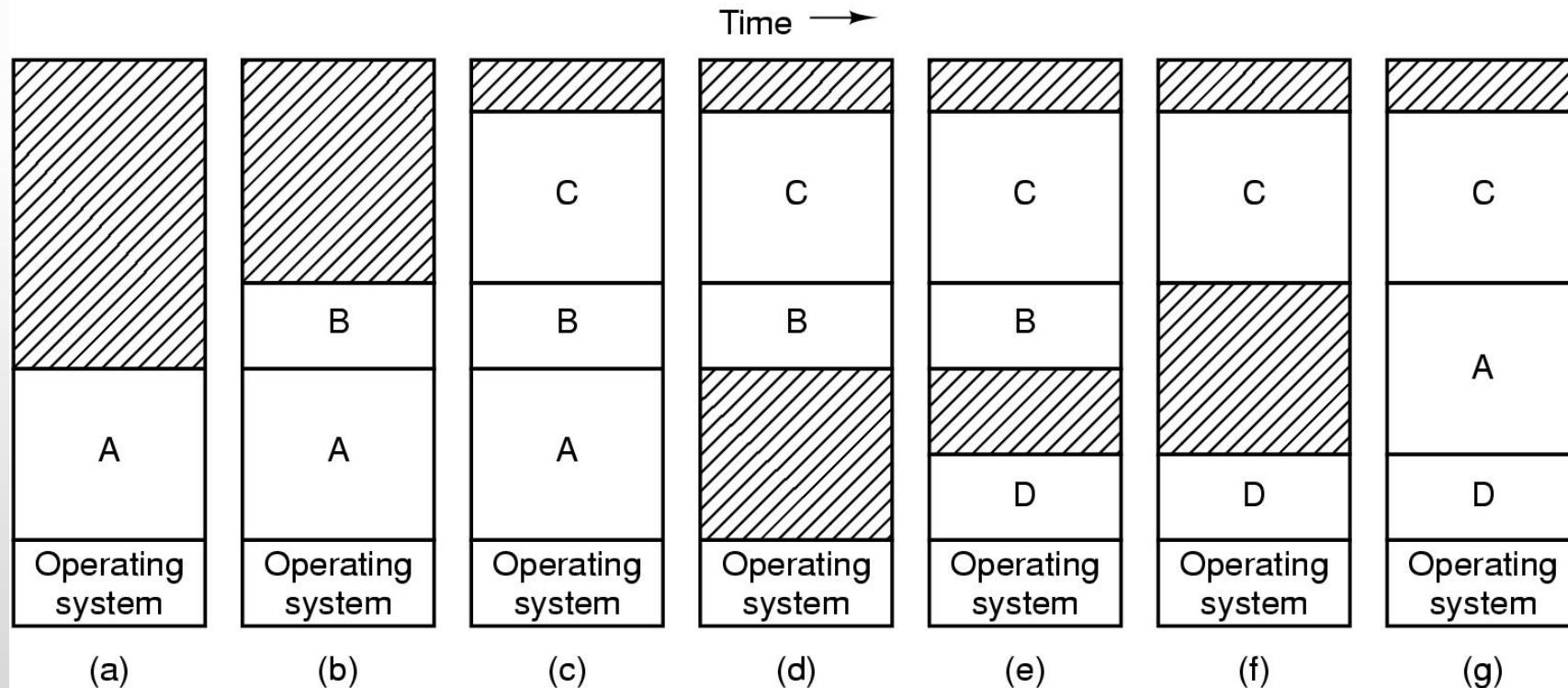
# Değişken Bölümleme

- Verimlilik için bölüm boyutları ihtiyaca göre değişkendir.
- Delik, boşluk (*hole*): kullanılabılır serbest bellek bloğu;
  - Farklı boyutlarda bellek boyunca dağınık durumda olabilir.
- Süreç yürütüleceğinde, yeterince büyük bir bellek alanı tahsis edilir.
- Süreç sonlandığında, bellek alanı boşaltılır, bitişik boş bölümler birleştirilir.
- İşletim sistemi, tahsis edilen ve boş bölümler hakkında bilgi tutar.



# Bellek Düzeni Değişimi

- Süreç, yürütüldükçe bellek tahsisini değişir.
- Gölgeli bölgeler kullanılmayan boş alanlardır.





# Problemler

- Diske yaz-Belleğe geri yükle takası (*swap*) sonrası adresler değişimdir.
  - Statik ve dinamik yer değiştirme.
- Bellek boşlukları (*hole*).
  - Bellek sıkıştırma (*compression*) yapılır.
    - İşlemci zamanı gereklidir. 4 byte için 20 ns, 1 GB için > 5 s.
- Bir süreç için ayrılan bellek miktarı ne kadar?
  - Süreçler büyümeye eğilimindedir.
  - Veri kesimi (*data segment*), yığın (*stack*)





# Boş Alan Yönetimi

- Süreçlere boş bellek alanı nasıl tahsis edilir?
  - İlk uyan (*first fit*)
    - Hızlı, baş kısımlar sık kullanılır, büyük bir alan verimsiz kullanılabilir.
  - Sonraki uyan (*next fit*)
    - En son kullanılan yerden aramaya başlanır.
  - En iyi uyan (*best fit*)
    - Tüm liste taranır, gerekli boyuta en yakın boşluk (*hole*) bulunur.
  - En kötü uyan (*worst fit*)
    - En büyük boşluk bulunur.
  - Hızlı uyan (*quick fit*)
    - Süreçler ve boşluklar ayrı kuyruklarda tutulur.



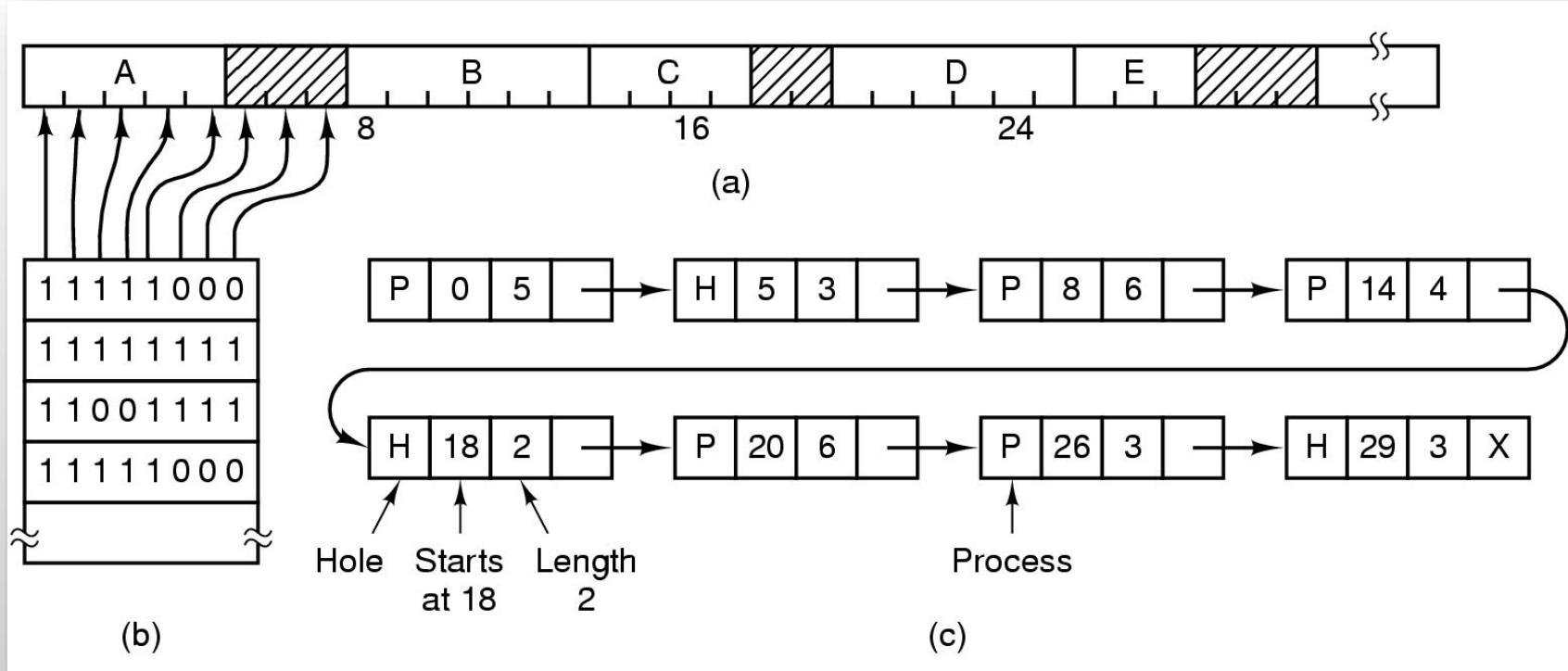
# Bellekte Boş Alan Yönetimi

- Biteşlem (*bitmap*) ve bağlı liste veri yapıları kullanılır.
- Biteşlem
  - Belleği takip etmenin basit ve sıkı (*compact*) yolu.
  - Bellek, küçük birimlerden (word, KB) oluşur.
  - Her birime biteşlem'de bir bit karşılık gelir.
  - $k$  birim dosyayı belleğe getirmek için, biteşlem'de  $k$  ardışık sıfır aranır.
  - İstenen uzunlukta boş alan bulmak zor. ☹



# Biteşlem ile Bellek Yönetimi

(a) Beş süreç ve üç boşluklu bellek kesiti. Bellek birimleri  $im$  ile ayrılmış. Taralı bölgeler (biteşlemde 0) boş. (b) Biteşlem. (c) Bağlı liste gösterimi.





# Bağılı Liste ile Bellek Yönetimi

- X süreci sonlanmadan önce ve sonrası oluşan bellek düzeni.





# Parçalanma (Fragmentation)

- **Harici Parçalanma:** isteği karşılayacak bellek alanı var, ancak bitişik değil.
- **Dahili Parçalanma:** tahsis edilen bellek, talep edilenden büyük olabilir.
  - Tahsis edilen bellek bölümünün içinde, boş kalan kısım.
- **Sıkıştırma (compression)** işlemi, harici parçalanmayı azaltır.
  - Boş bellek blokları, tek bir büyük blokta bir araya getirilir.
  - Dinamik yer değiştirme (*dynamic relocation*) kullanılıyorsa mümkün.



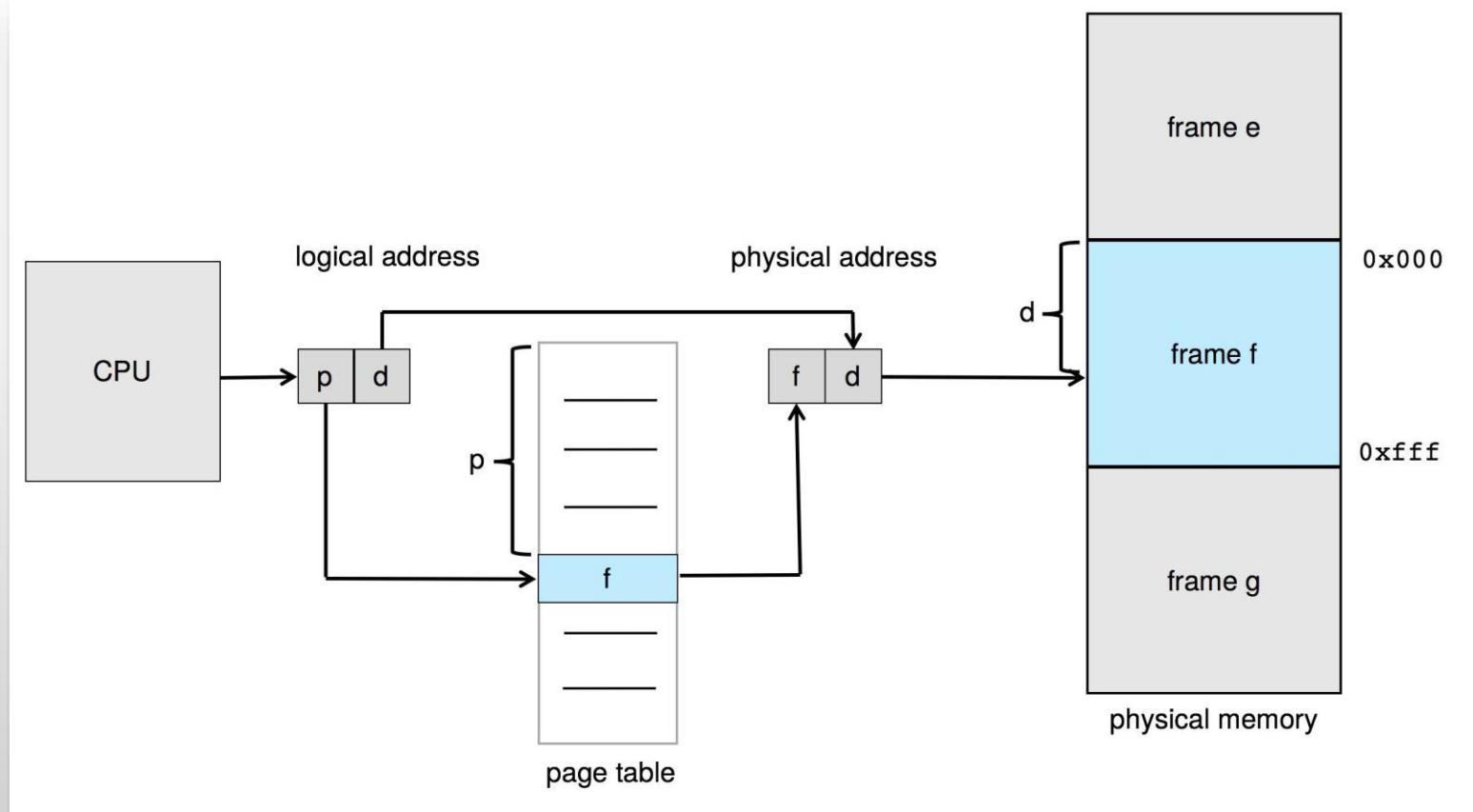
# Sayfalama (Paging)

- Bir sürecin fiziksel adres alanının bitişik olması gerekmez.
  - Harici parçalanmayı öner.
  - Değişken boyutlu bellek blokları yerine sabit boyutlu kullanılır.
- Fiziksel bellek **çerçeve (frame)** adı verilen sabit boyutlu bloklara ayrılır.
- Boyut, 2'nin kuvveti olarak, *512 bayt* ile *16 MB* arasında olur.
- Mantıksal bellek **sayfa (page)** adı verilen sabit boyutlu bloklara ayrılır.
  - Tüm boş çerçeveler takip edilir.
  - $N$  sayfa boyutunda bir süreç için,  $N$  adet boş çerçeve gereklidir.
- Mantıksal adresler, **sayfa tablosu (page table)** ile fiziksel adrese çevrilir.
- Dahili parçalanma oluşur.



# Sayfalama Donanımı

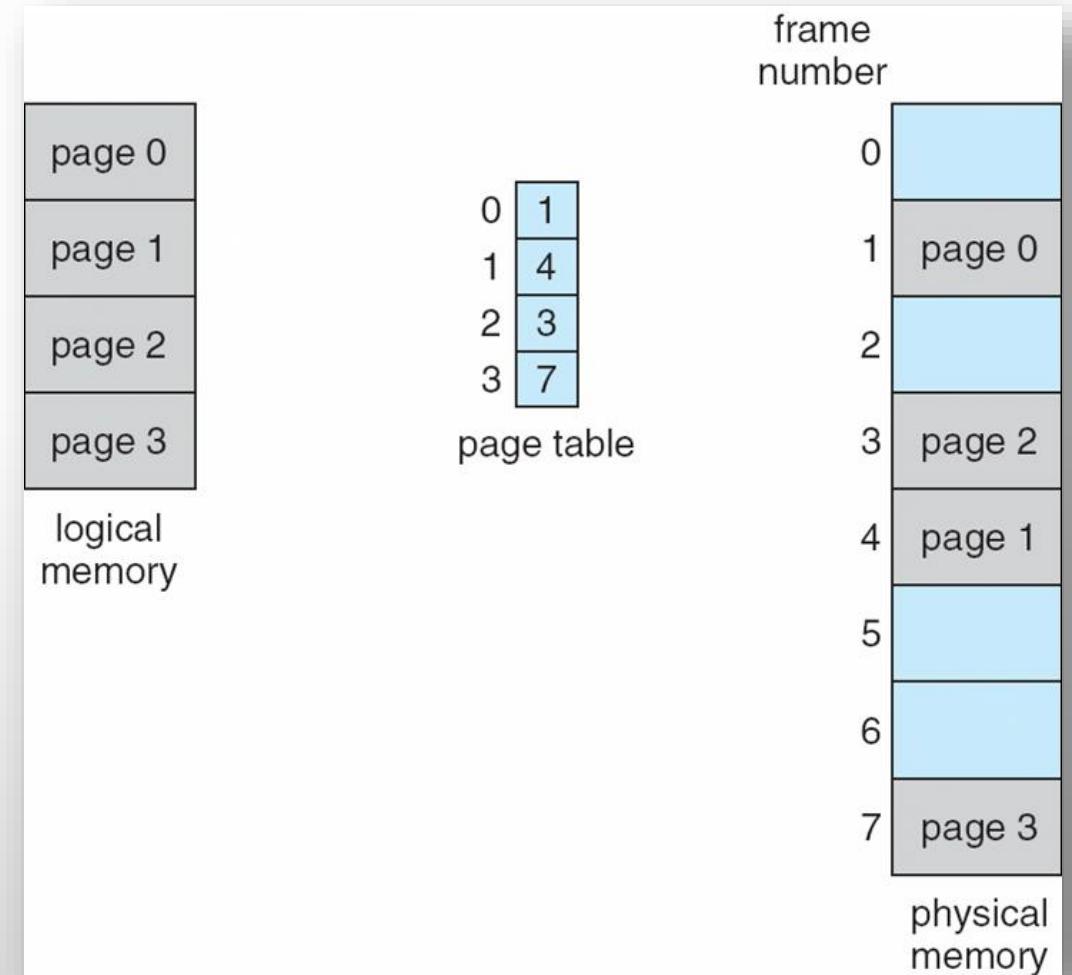
- .





# Mantıksal ve Fiziksel Bellek

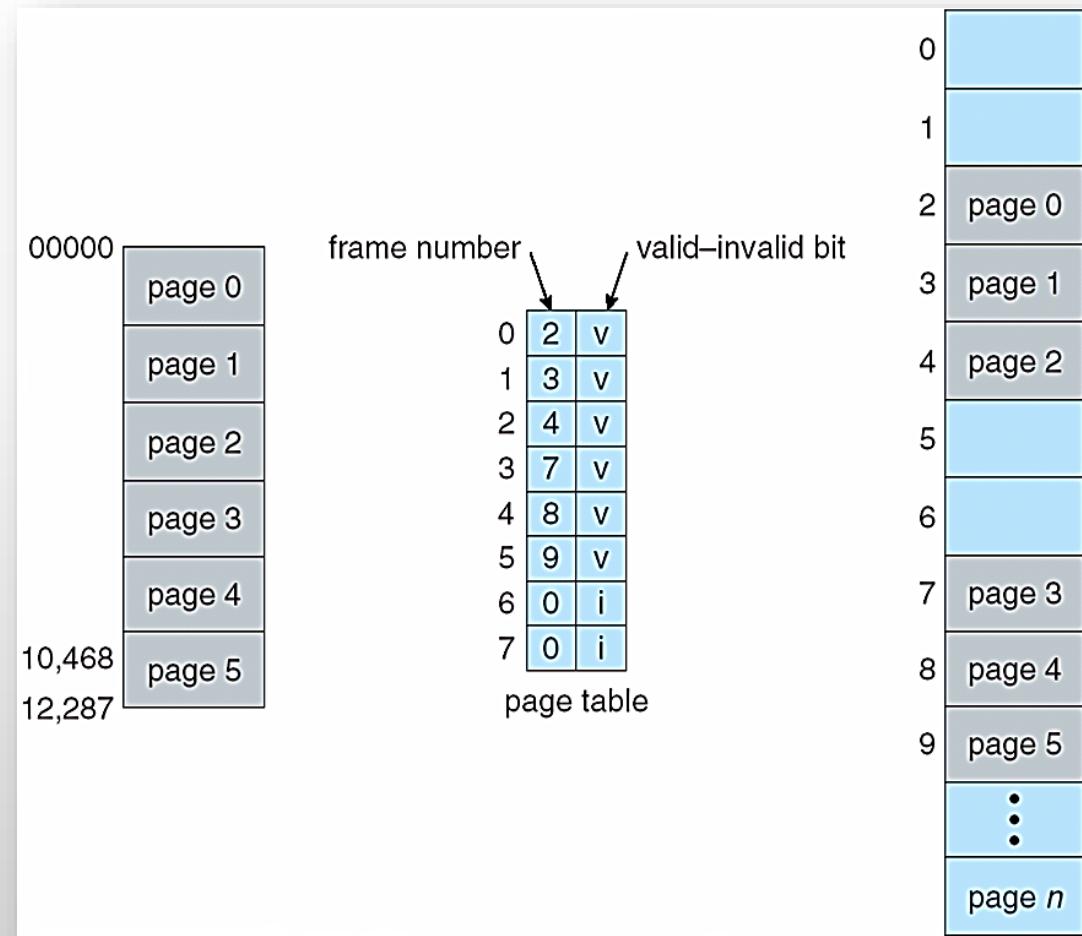
.





# Sayfa Tablosu Elemanı Yapısı

- .





# Sayfalama – Dahili Parçalanma

- Sayfa boyutu = 2.048 *bayt*
- Süreç boyutu = 72.766 *bayt*
- 35 sayfa + 1086 *bayt*
- $2048 - 1086 = 962$  *bayt* (dahili parçalanma)
- En kötü durumda parçalanma = 1 çerçeve boyutu – 1 *bayt*
- Ortalama parçalanma = çerçeve boyutu / 2
- Küçük çerçeve boyutları arzu edilir mi?
  - Her sayfa tablosunda satır için bellek gereklidir.
  - Süreç boyutları zamanla büyür.

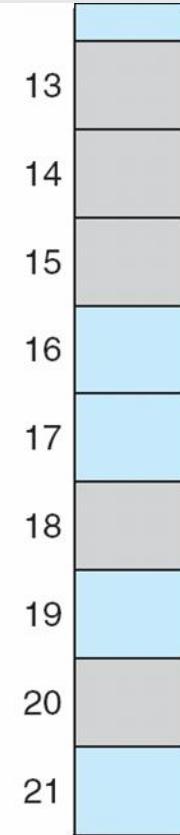
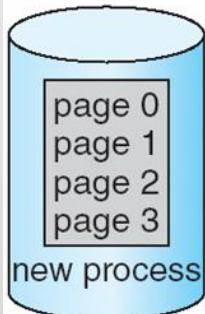


# Boş Çerçeveler

- .
- .

free-frame list

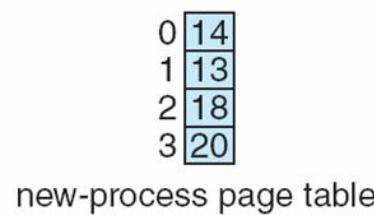
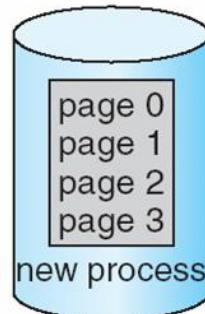
14  
13  
18  
20  
15



(a)

free-frame list

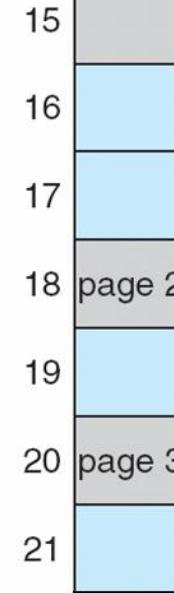
15



(b)

free-frame list

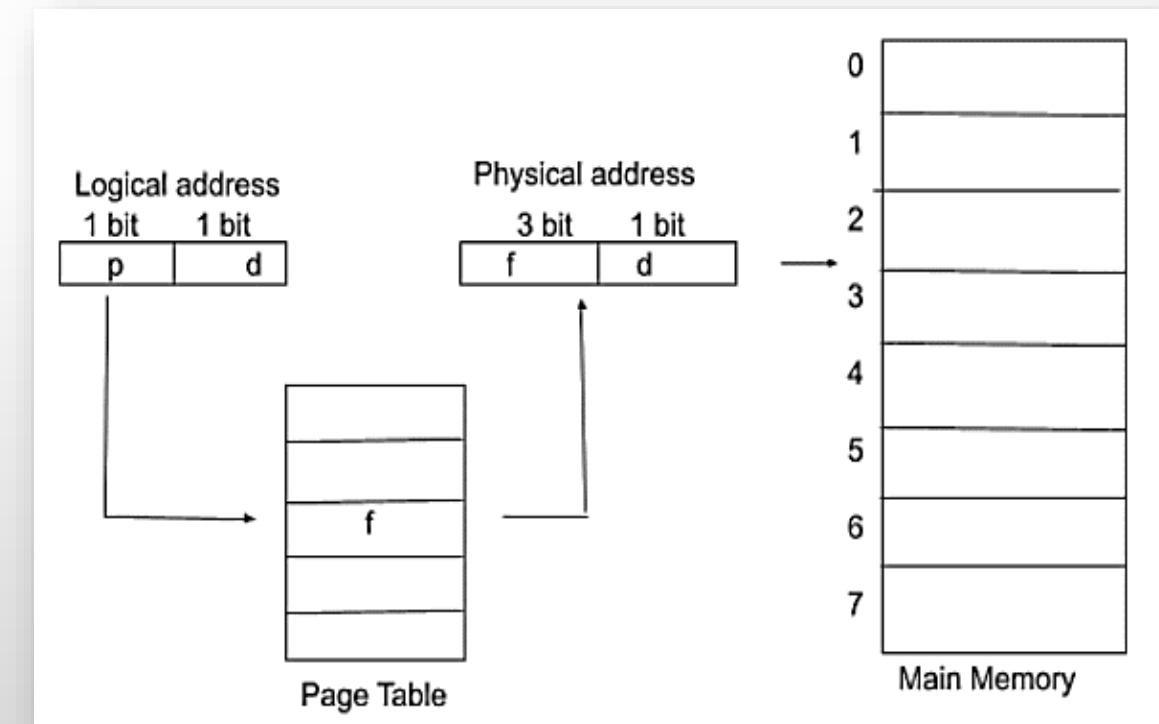
13  
page 1  
14  
page 0





# Sayfa Tablosu (Page Table)

- Mantıksal adres = {sayfa numarası, bağıl konum (offset)}
- Fiziksel adres = {çerçeve numarası, bağıl konum (offset)}
- Çerçeve, sayfa tablosunda (*page table*) sayfa ile ilgili satırda bulunur.
- *Mevcut/yok* biti 1 ise, çerçeve bağıl konumun önüne eklenir.



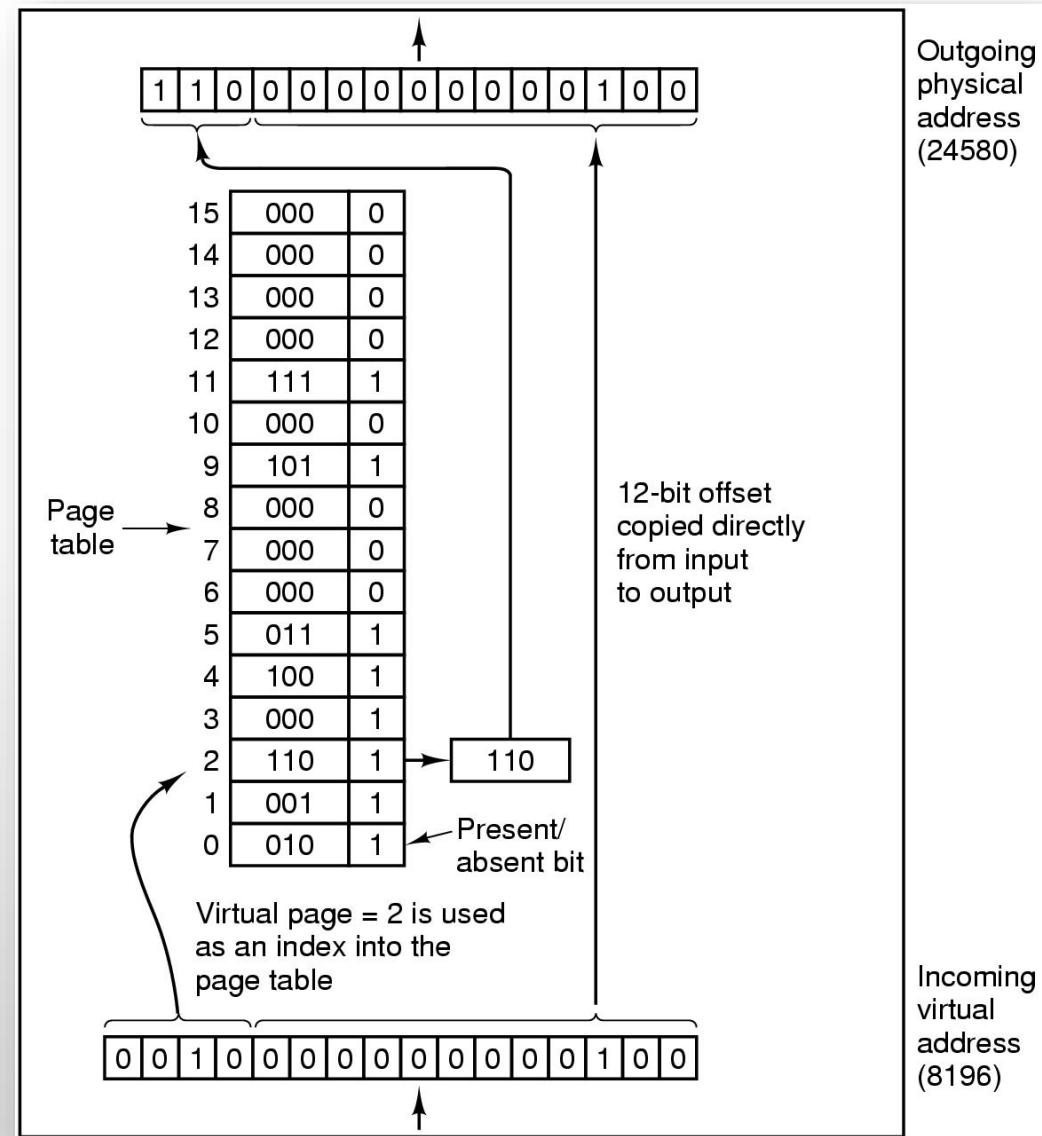


# Hatalı Sayfa İşlemi

- *Mevcut/yok* biti, sayfanın bellekte olup olmadığını söyler.
- Adres bellekte yoksa ne olur?
- İşletim sistemine tuzak (*trap*).
  - Diske göndermek için bir sayfa seçilir.
  - Gerekli olan sayfa belleğe getirilir.
  - Komut yeniden başlatılır.

# Bellek Yönetim Birimi

- Sayfa tablosunda 16 girdi var, sayfa numarası için 4 *bit* gereklidir.
- Sayfa boyutu = 4 KB, bağıl adres için 12 *bit* gereklidir.
- çerçeve numarası 3 *bit*.
- **MMU (*memory management unit*)**



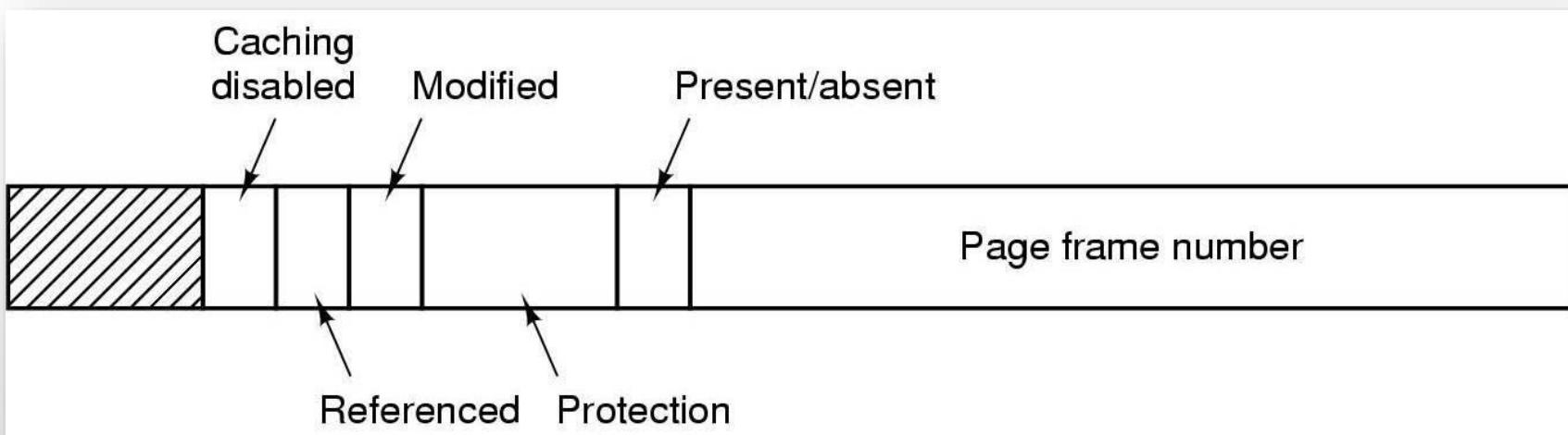


# Sanal Adres Eşleme

- Sanal adres, sayfa numarası ve bağıl adres.
- 16 *bit* adres: 4 KB sayfa boyutu (*12 bit*), 16 sayfa (*4 bit*).
- Sayfa numarası: sayfa tablosundaki indis (*index*).
- Sayfa tablosu, sayfa numarasını çerçeve numarasına eşler.



# Sayfa Tablosu Elemanı Yapısı





# Sayfa Tablosu Yapısı

- Koruma (*protection*)
  - Ne tür erişimlere (*okuma, yazma*) izin verilir?
- Değiştirildi (*modified*)
  - Yazma (*kırli (dirty page)*) yapıldığında 1 değeri atanır.
- Erişildi (*referenced*)
  - Sayfaya erişildiğinde 1 değeri atanır.
- Önbellek devre dışı (*caching disabled*)
  - Veri tutarsızlığını önlemek için önbellek devre dışı bırakılabilir.



# Sayfalama Uygulama Sorunları

- Sanal adresten fiziksel adrese eşleme hızlı olmalıdır.
- Sanal adres alanı büyükse, sayfa tablosu da büyük olacaktır (*32bit/64bit*).
- Her sürecin bellekte kendi sayfa tablosu olmalıdır.



# Sayfalamayı Hızlandırma

- Sayfa tablosu yazmaçta tutulursa?
  - Süreç koşarken bellek erişimi gerekmeyez. ☺
  - Son derece maliyetli. ☹
- Sayfa tablosu tamamen bellekte tutulursa?
  - Her sürecin kendi sayfa tablosu var.
  - Bir bellek erişimi için kaç bellek erişimi gerekir? (mantıksal → fiziksel)
  - İki kez bellek erişimi; performansı yarı yarıya azaltır.
  - Sayfa tablosuna erişim + verilere/komutlara erişim.



# Sayfalamayı Hızlandırma

- Çözüm, özel ve hızlı arama yapabilen **donanım** gereklili.
  - İlişkisel yazmaçlar (*associative registers*).
  - Adres çevirme tampon bellekleri (*translation look-aside buffers*).



# Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

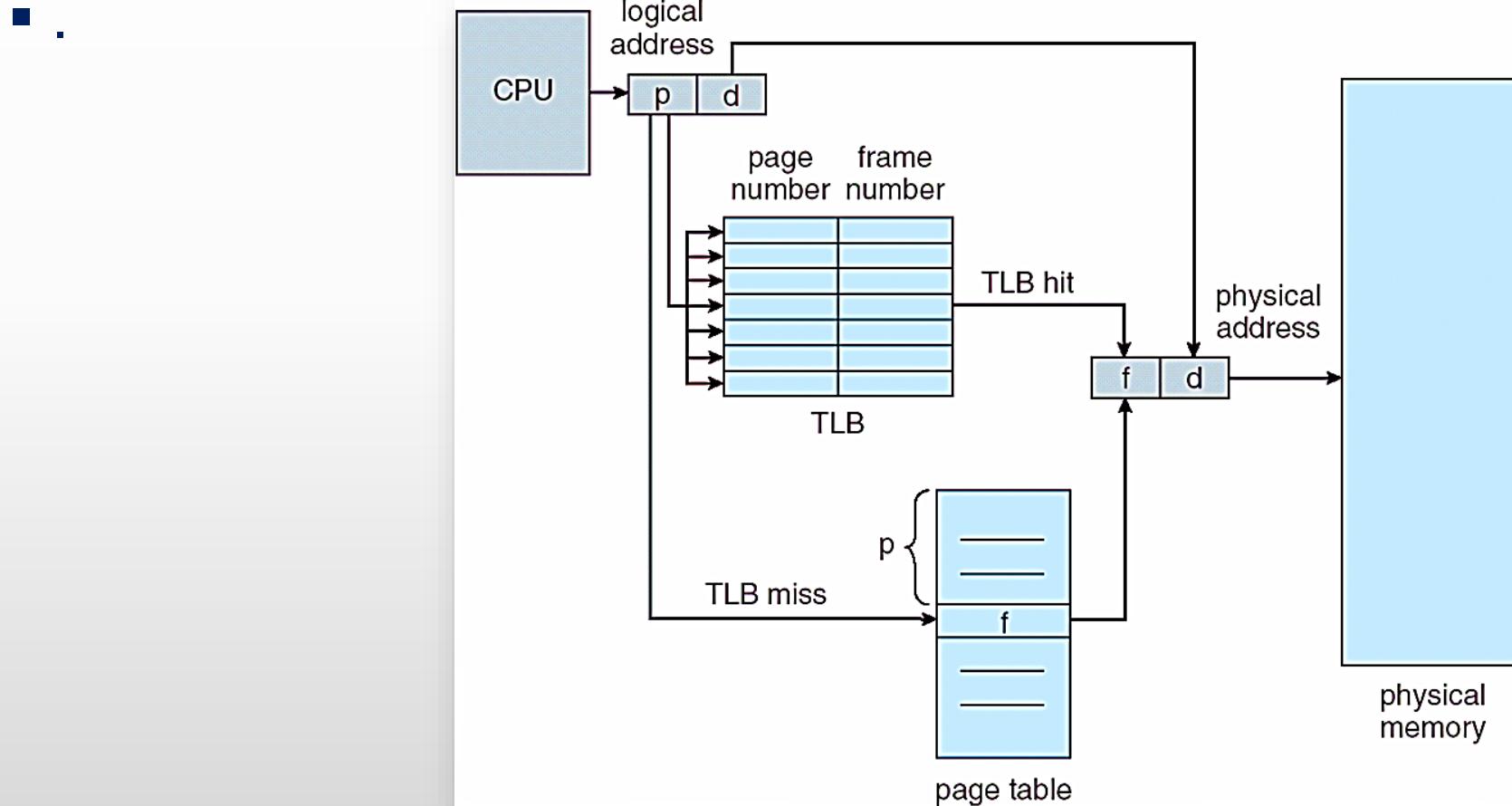


# TLB ile Sayfalama

- *TLB, MMU* içerisinde bulunur.
- Az sayıda elemandan oluşur.
- Sanal bir adres geldiğinde,
  - Sanal sayfa numarasının *TLB*'de olup olmadığı kontrol edilir.
  - Varsa, sayfa tablosuna bakmaya gerek yok. ☺
  - Yoksa, *TLB*'den bir eleman ile sayfa tablosundan bir eleman takas edilir



# TLB ile Sayfalama





# TLB Yönetimi

- RISC makineleri TLB'yi yazılım ile yönetir.
- TLB hatası, MMU donanımı yerine işletim sistemi tarafından işlenir.
- Yazılım, hangi sayfaların TLB'ye önceden yükleneceğini tahmin edebilir.
- Sık kullanılan sayfalar önbellekte tutulur.



# Etkili Erişim Süresi

- İlişkisel arama =  $\varepsilon$  zaman birimi.
- Bellek çevrim (*cycle*) süresi =  $t$  zaman birimi.
- İsabet oranı =  $\alpha$
- Etkili Erişim Süresi =  $(t + \varepsilon) \alpha + (2t + \varepsilon)(1 - \alpha) = 2t + \varepsilon - t\alpha$
- $\varepsilon$  (20 ns),  $t$  (100 ns),  $\alpha_1$  (%80),  $\alpha_2$  (%98) ise:
  - TLB'de bulundu (*hit*):  $20 + 100 = 120$  ns.
  - TLB'de bulunamadı (*miss*):  $20 + 100 + 100 = 220$  ns.
  - $EES_1 = 120 * 0,80 + 220 * 0,20 = 140$  ns.
  - $EES_2 = 120 * 0,98 + 220 * 0,02 = 122$  ns.



# Büyük Bellek için Sayfa Tablosu

- Adres alanı: 32 *bit*.
- Sayfa boyutu: 4 *KB* ( $4096 \rightarrow 12$  *bit*).
- Sayfa numaraları:  $32 - 12 = 20$  *bit*, 1 milyon sayfa ( $2^{32} / 2^{12}$ ).
- Sayfa tablosunda her bir eleman 32 *bit* (4 *byte*),
- Sayfa tablosu boyutu 4 *MB*. ( $2^{20} * 4$  *byte*)
- 64 bit sistem için?
  - Çözüm: sayfa tablosu küçük parçalara bölünür.
    - Sıradüzensel (*hierarchical*)
    - Karma (*hashed*)
    - Ters (*inverted*)



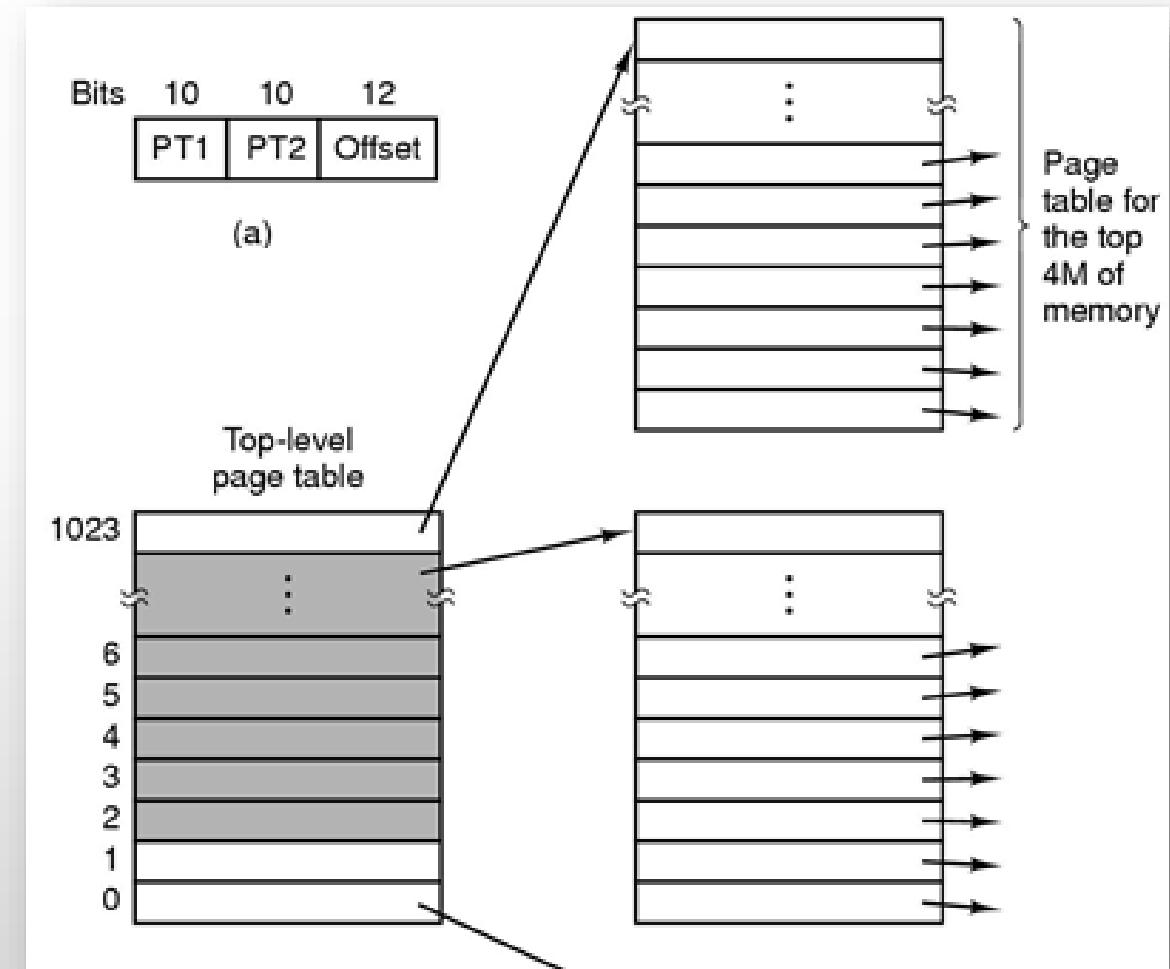
# Sıradüzensel Sayfa Tablosu

- En basit teknik.
- İki seviyeli sayfa tablosu.
- 32 *bitlik* sayfa tablosu girdisi üç bölüme ayrılır. (*PT* = *page table*)
  - 10 *bit*  $PT_1$ , 10 *bit*  $PT_2$ , 12 *bit* bağıl konum (*offset*).
- Tüm sayfa tablolarını sürekli bellekte tutmak gerekmekz.
- Sayfa tabloları da sayfalarda saklanır.
- Süreç 4 *GB* adres alanına sahip, koşmak için 12 *MB* belleğe ihtiyaç duyar.
  - 4 *MB* kod, 4 *MB* veri, 4 *MB* yığın için.



# Sıradüzensel Sayfa Tablosu

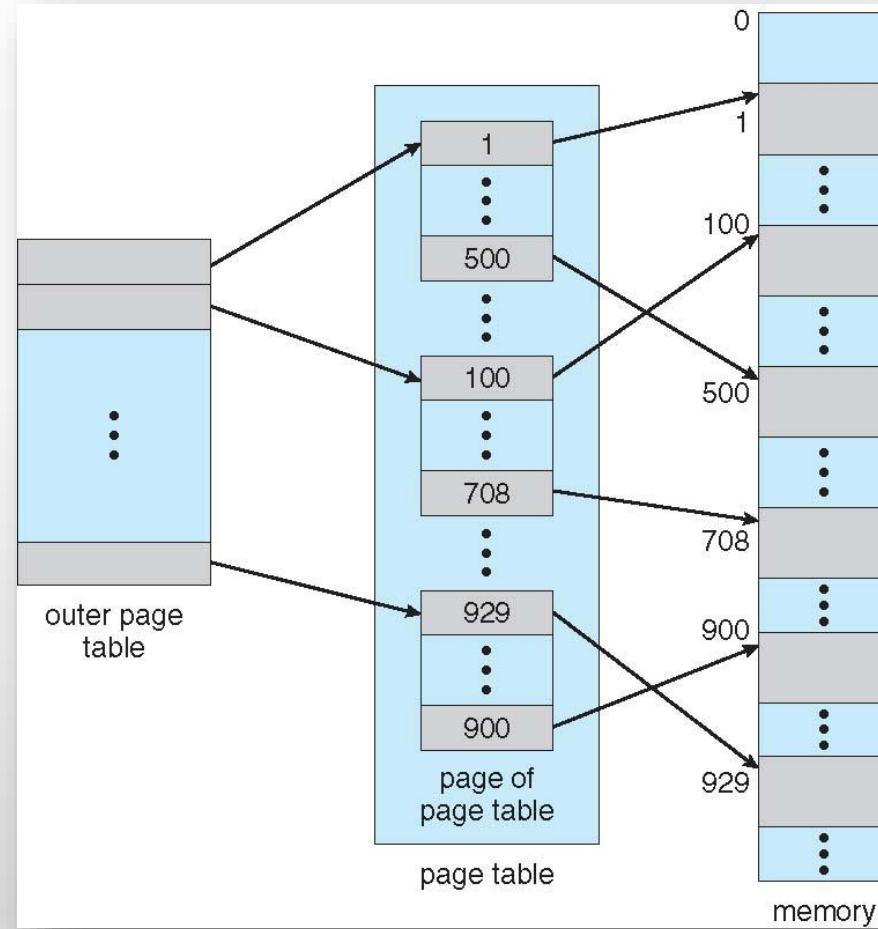
- (a) 32 bitlik adres.
- (b) İki seviyeli sayfa tabloları.





# Sıradüzensel Sayfa Tablosu

- .





# Sıradüzensel Sayfa Tablosunun Kullanımı

- Sayfa tablosunun,
  - ilk satırı, kod kesimi için sayfalara, (girdi 0)
  - ikinci satırı, veri kesimi için sayfalara, (girdi 1)
  - son satırı, yiğin kesimi için sayfalara (girdi 1023) işaret eder.



# Sıradüzensel Sayfa Tablosunun Kullanımı

- Sıradüzensel sayfa tablosu  $32\ bit$  bellek için çalışır.
- $64\ bit$  bellek için çalışmaz. 😞
- $2^{64}$  *bayt* ve  $4\ KB$  sayfa boyutu ( $12\ bit$ ) → sayfa tablosunda  $2^{52}$  satır!
- Her girdi  $8\ bayt$  ise → sayfa tablosu 30 milyon *GB*.
- Başka bir çözüme ihtiyaç var! 😞

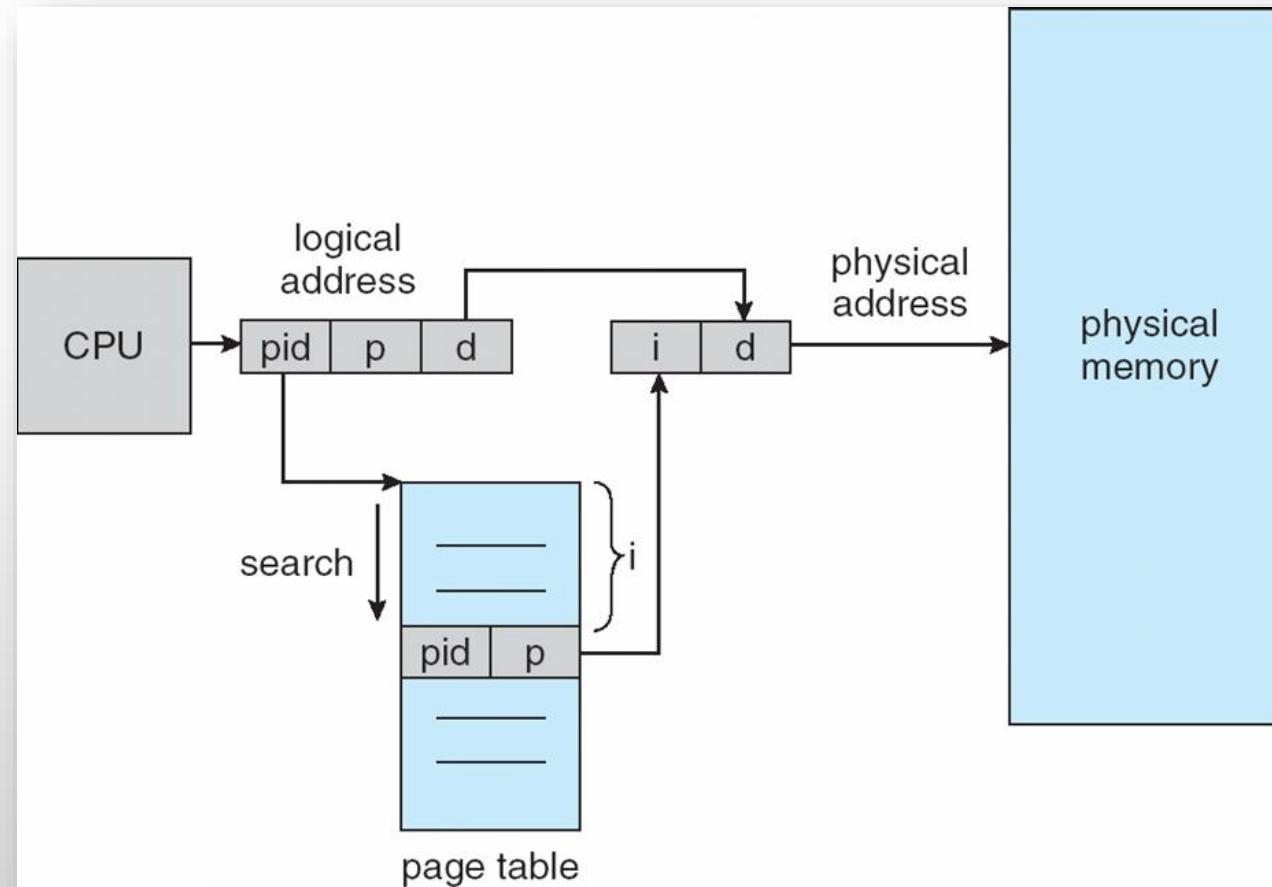


# Ters Sayfa Tablosu

- Sanal adres alanları fiziksel bellekten çok büyük olduğunda kullanılır.
- 32 *bitten* büyük adres alanlarını destekler.
- Sayfa yerine çerçeve başına bir girdi tutulur.
- Girdiler, çerçeve ile ilişkili süreç ve *sanal sayfa* takibini yapar.
- Her bellek erişimi için  $(n,p)$  ilişkili sayfa çerçevelerini arar.
- Arama işlemi daha maliyetli. Verimli bir şekilde nasıl yapılır?
  - Sık kullanılan çerçeveler *TLB*'de tutulur.
  - *Hash* tablo kullanılır.

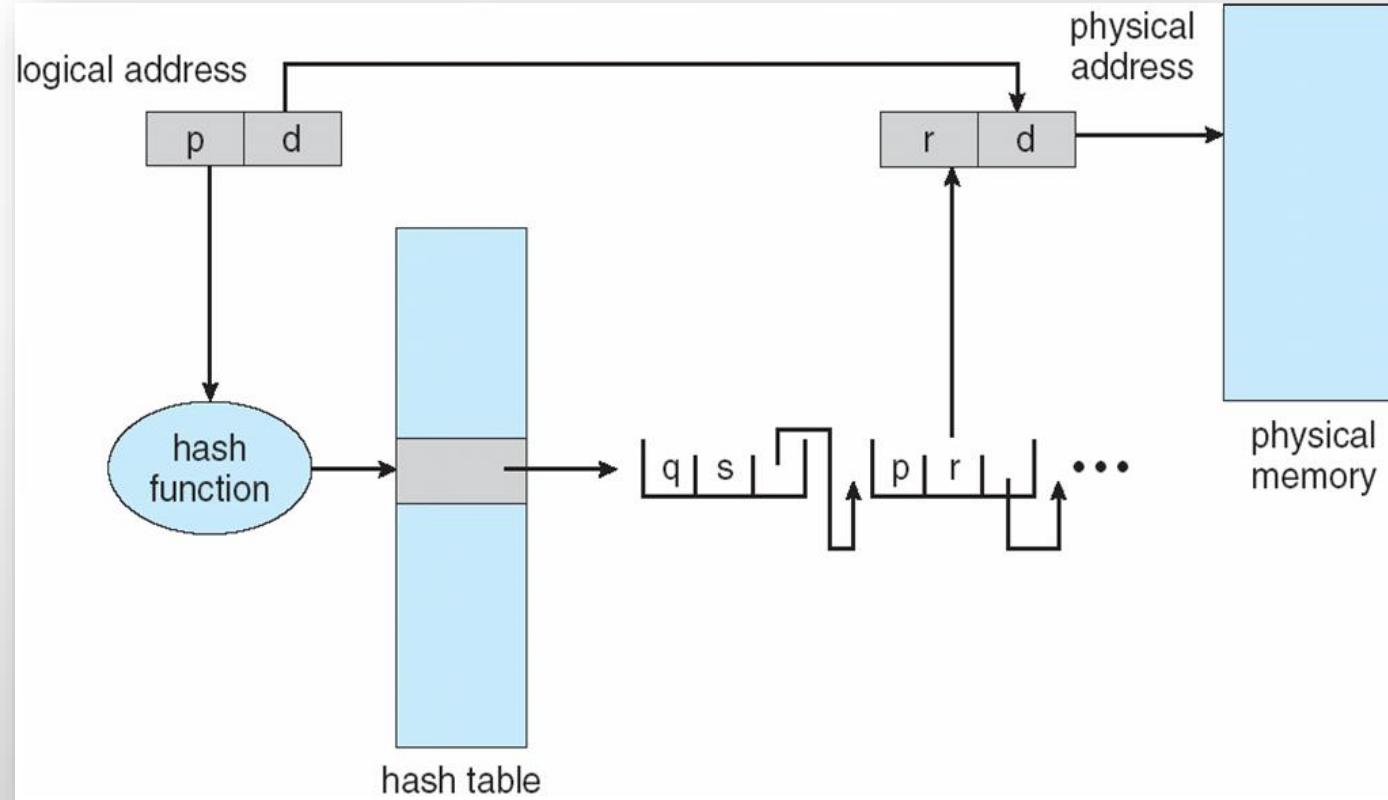


# Ters Sayfa Tablosu



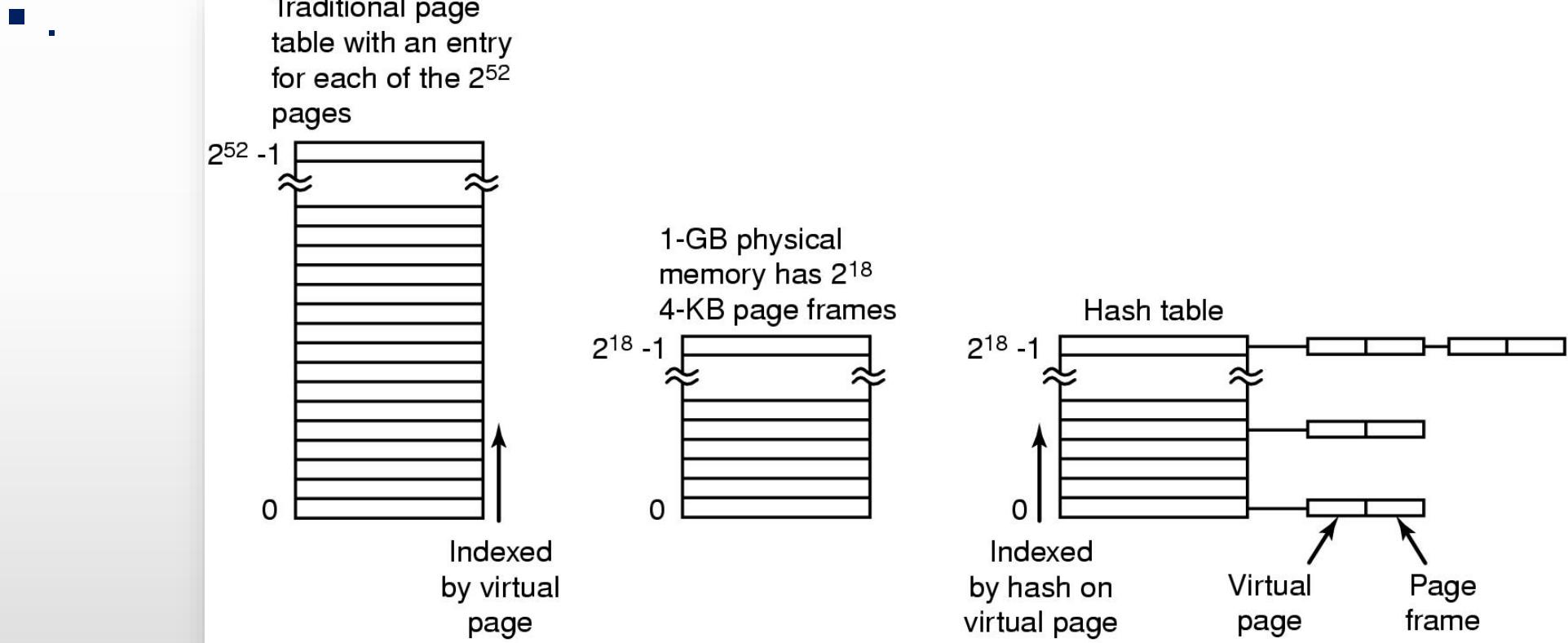


# Hash Sayfa Tablosu





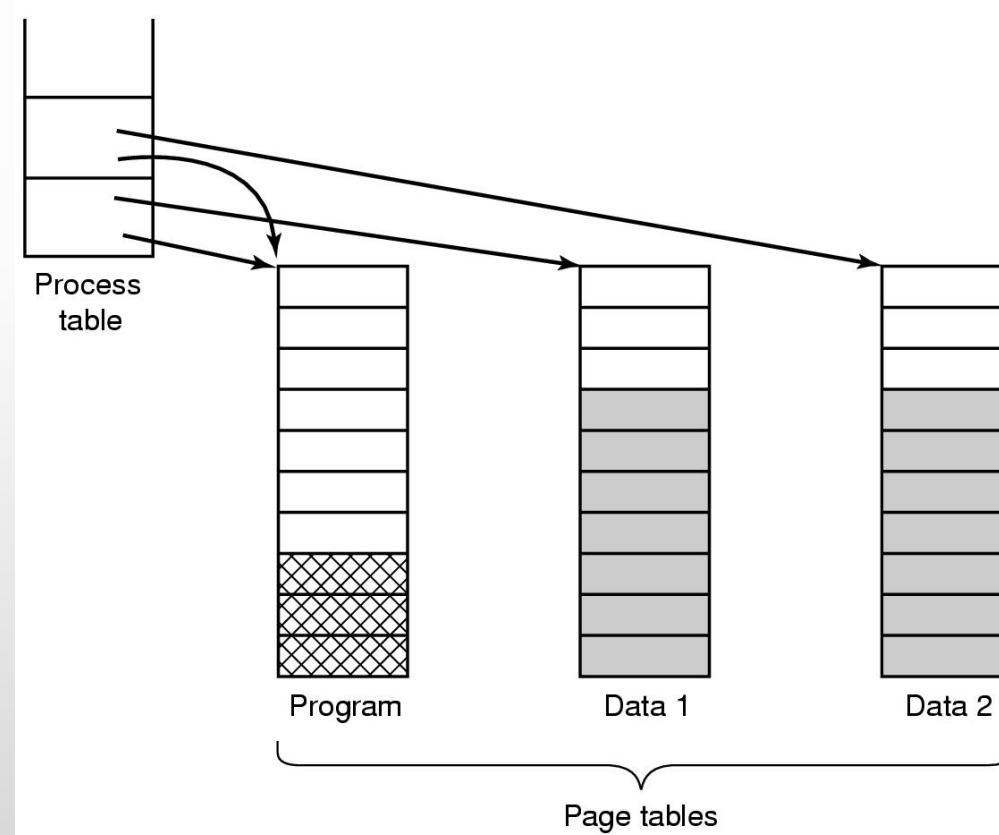
# Geleneksel ve Hash Sayfa Tablosu



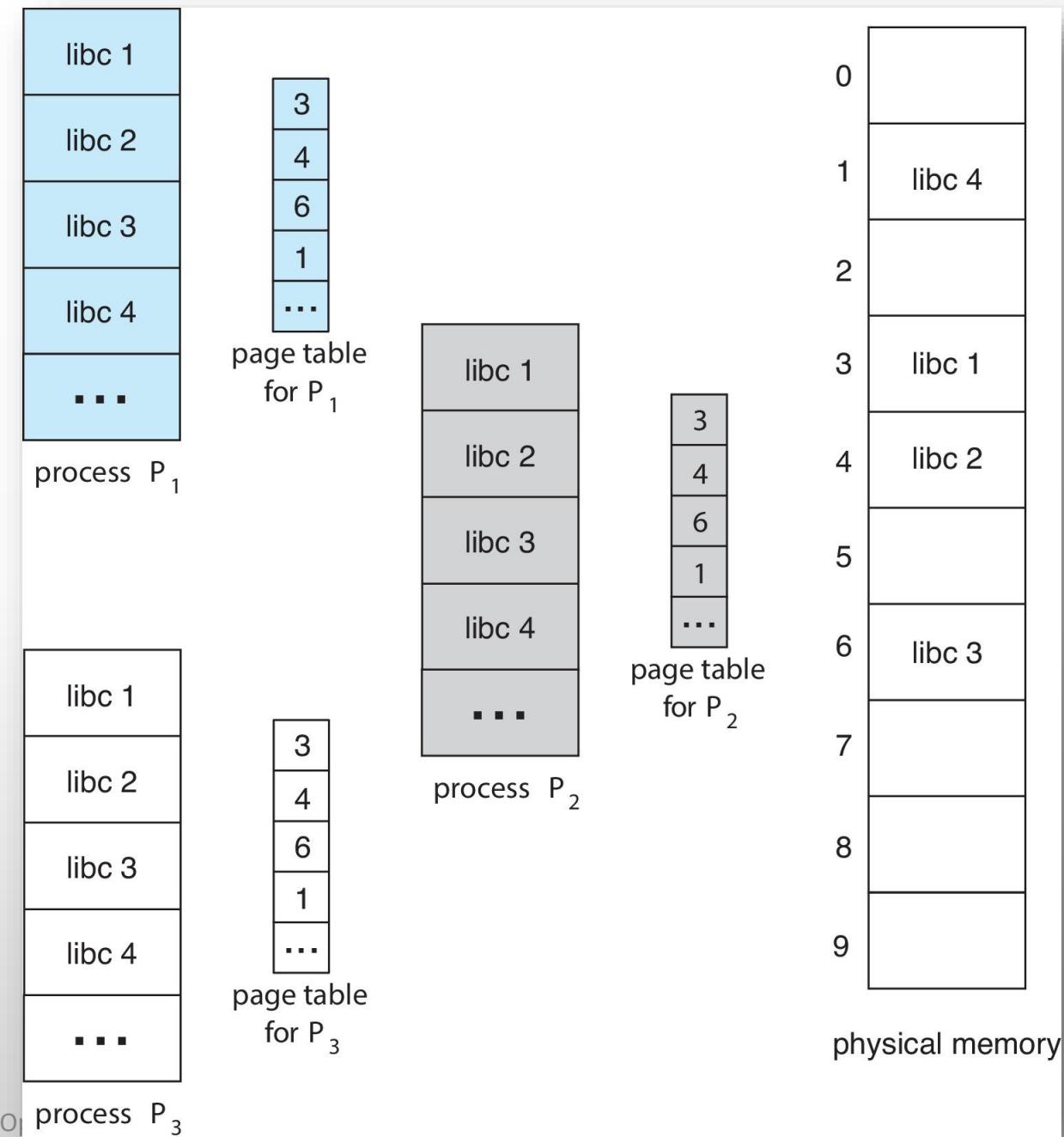


# Paylaşımı Sayfalar (Shared Pages)

- Bir programdan iki farklı süreç oluşturulursa.



# Paylaşımı Sayfalar





# Paylaşımılı Sayfalar

- Süreç sonlanırken paylaşımılı sayfaları serbest bırakamaz.
  - Başka bir süreç kullanıyor olabilir.
  - Bu sayfaları izlemek için özel veri yapısı kullanılır.
- Sayfaya yapılan yazmalar nedeniyle veri paylaşımı sıkıntılı.
  - *fork()*, ata ve çocuk süreçler, *kod* ve *veri* kesimlerini paylaşırlar.
  - Yazarken kopyala (*copy-on-write*) çözümü,
    - Veriler ilk başta salt okunur (*read-only*) sayfalara eşlenir.
    - Yazma işlemi, kopyalanan sayfa üzerinde gerçekleştirilir.



# Bellek Eşlemeli (Memory Mapped) Dosyalar

- Süreç, bir dosyayı sanal adres alanının bir parçasına eşleyebilir.
- Paylaşımılı bellek (*shared memory*) yoluyla iletişim için kullanılır.
- Süreçlerin aynı dosyayı paylaşabilmesi sağlanır.
- Okuma ve yazma işlemleri yapılabilir.



# Temizleme İlkesi

- Bellekte yer kalmadığında, çıkarılacak sayfa aramak yerine;
  - Bir arka plan (*daemon*) süreci sayfaları yönetmeli.
- Arka plan süreci, periyodik olarak uyanır ve çalışır.
- Çıkarılacak sayfalar temiz (*değiştirilmemiş*) olmalı.
  - Kirli (*dirty bit*) ise değişiklikler diske yansıtılmalı.



# Sanal Bellek Arayüzü

- 2 farklı süreç fiziksel belleği paylaşmak isteyebilir.
- Paylaşımılı bellek (*shared memory*) en kolay iletişim yöntemi.
  - Bellek kopyalama yaklaşımından kaçınır.
- Dağıtık (*distributed*) paylaşımılı bellek,
  - Sayfalar farklı makinelerde olabilir.



# Uygulama Sorunları

- İşletim sistemi, süreç oluşturulduğunda, yürütülürken, sonlandırıldığında ve hata durumunda, sayfalamaya çok fazla dahil olur.
- Sorunlar
  - Sayfa hatalarını ele alma,
  - Komut yedekleme,
  - Bellekteki sayfaları kilitleme,
  - Yedekleme deposu: sayfaların diskte yerleştirileceği yer.



# Sayfa Hatasını Ele Alma

1. Donanım çalışmayı çekirdeğe bırakır, program sayacını yiğina kaydeder.
2. Assembly kod parçası genel yazmaçları ve geçici bilgileri kaydeder.
3. İşletim sistemi sayfa hatasına neden olan adresi belirler.
4. Adresin geçerliliğini ve koruma ile erişimin tutarlığını kontrol eder.
5. Çıkarılacak sayfa çerçevesi kirli (*dirty*) ise,
  - 5.1. Sayfanın diske aktarılması çizelgelenir ve
  - 5.2. Bağlam anahtarlaması (*context switch*) gerçekleşir.
6. Temiz ise, gerekli sayfanın bulunduğu disk adresi aranır ve
  - 6.1. Belleğe getirilmek için bir disk işlemi çizelgelenir.



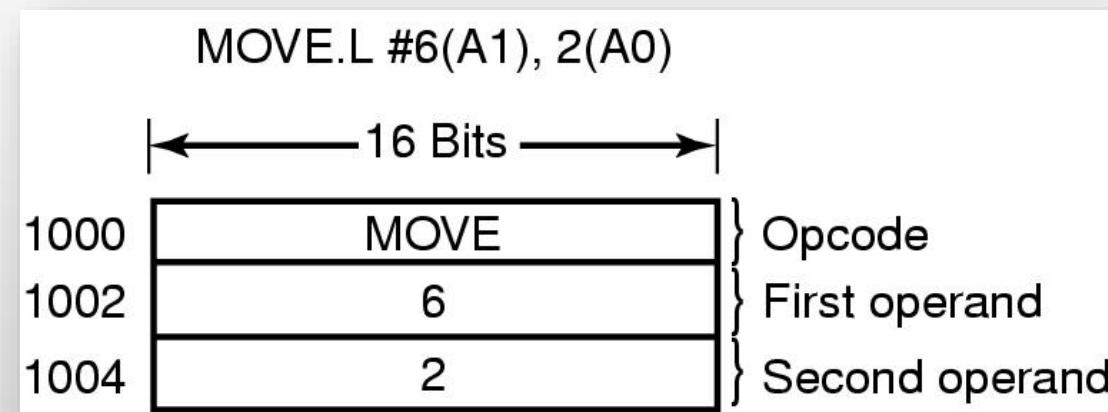
# Sayfa Hatasını Ele Alma

7. Sayfa belleğe taşındığında disk kesmesi oluşur,
  - 7.1. Sayfa tablosu güncellenir, çerçeve normal durumda olarak işaretlenir.
8. Hatalı komut, ilk çalıştırıldığı andaki durumuna yedeklenir ve
  - 8.1. Bu komutu işaret etmek için program sayacı sıfırlanır.
- 9 Hataya neden olan süreç çizelgelenir,
  - 9.1. İşletim sistemi onu çağrıran (*assembly*) kod parçasına geri döner.
10. Yazmaç ve diğer durum bilgileri yeniden yüklenir ve
  - 10.1 Hiçbir hata meydana gelmemiş gibi koşmaya devam eder.



# Sayfa Hatasına Neden Olan Komut

- Komut yeniden hangi adresten başlatılır?
  - Komutun hangi bölümünün hatalı olduğuna bağlıdır.
- 1002'de hata verirse,
  - İşletim sistemi komutun 1000'de başladığını nereden biliyor?





# Komut Yedekleme

- Donanım çözümü
  - Komut yürütülmeden önce mevcut komut bir yazmaca kopyalanır.
  - Aksi takdirde, işletim sisteminin durumu vahim 😞
- Komut yürütülmeden önce veya sonra, yazmaca yüklenir.
  - Önce yüklenirse, işlem geri alınır.
  - Sonra yüklenirse, işlem yapılmaz.



# Bellekte Sayfa Kilitleme

- Süreç, G/Ç çağrısı yapar, verileri bekler.
- Süreç beklerken askiya (*suspend*) alınır.
- Yeni süreç başlatılır, ve yeni süreçte ait sayfa hataları alınır.
- Global sayfalama algoritması
  - Gelen veriler yeni sayfanın üzerine yazılır.
  - Çözüm: G/Ç'de devreye giren sayfaları kilitle.



# Yedekleme Deposu (Backing Store)

- Sayfa takas edildiğinde diskte nereye konur?
- İki yaklaşım.
  - Ayrı bir diske.
  - Diskte ayrı bir bölüme.



# Yedekleme Deposu - Statik Yaklaşım

- Süreç başladığında *sabit bir alan* tahsis edilir.
- Boş parçaların listesi olarak yönetilir.
  - Süreç için yeterince büyük parça atanır.
- Süreç tablosunda *tahsis edilen alanın* başlangıç adresi tutulur.
  - Sanal adres uzayındaki *bağıl konum*, diskteki adrese karşılık gelir.
- Kod, veri, ve yığın kesimleri için farklı alanlar atanabilir.
  - Yığın zamanla genişleyebilir.



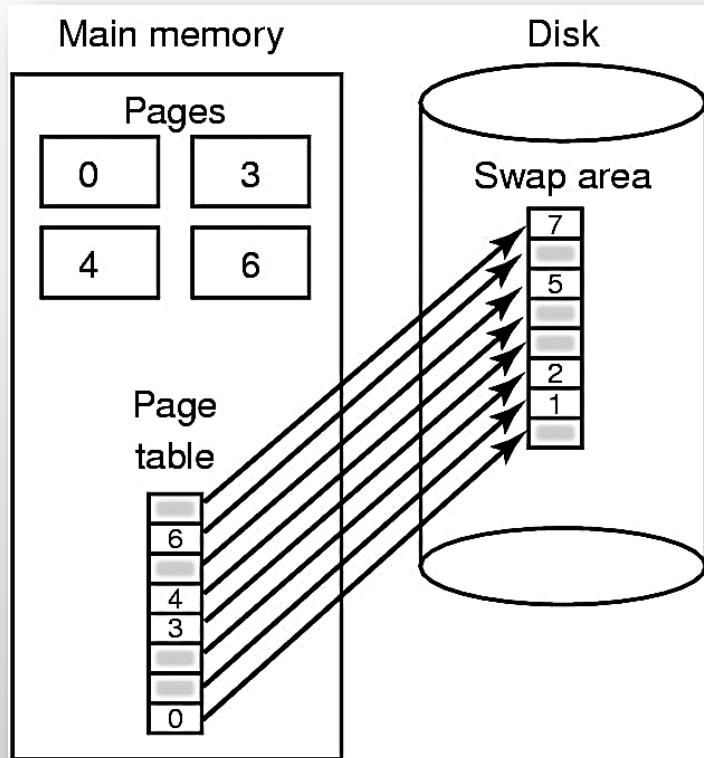
# Yedekleme Deposu – Dinamik Yaklaşım

- Süreç için önceden disk alanı ayrılmaz.
- Gerektiğinde sayfalar bellek ve disk arasında takas edilir.
- Bellek ile disk arasında eşlemeye (*mapping*) ihtiyaç vardır.

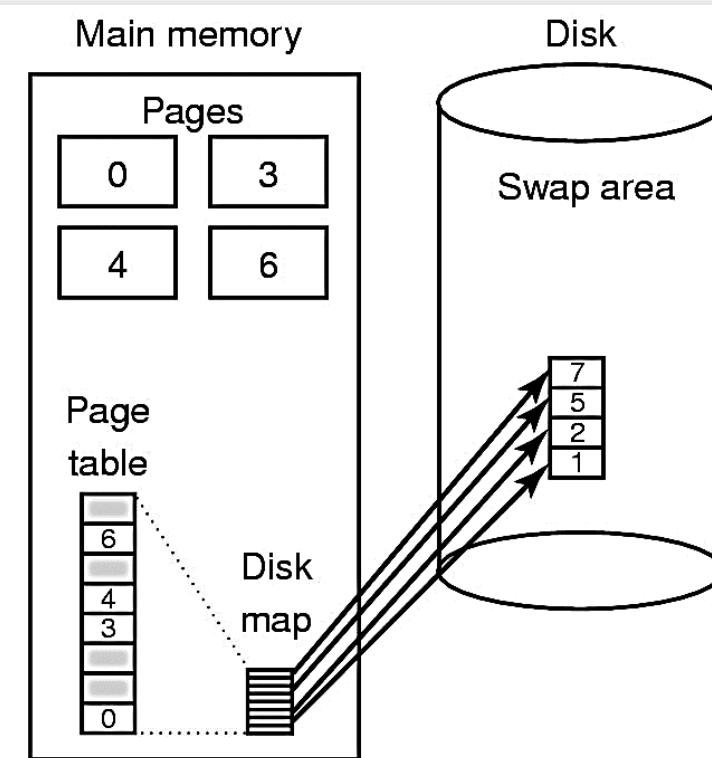


# Statik ve Dinamik Takas Alanı

- (a) Statik takas alanına sayfalama (*paging*) (b) Dinamik yaklaşım.



(a)

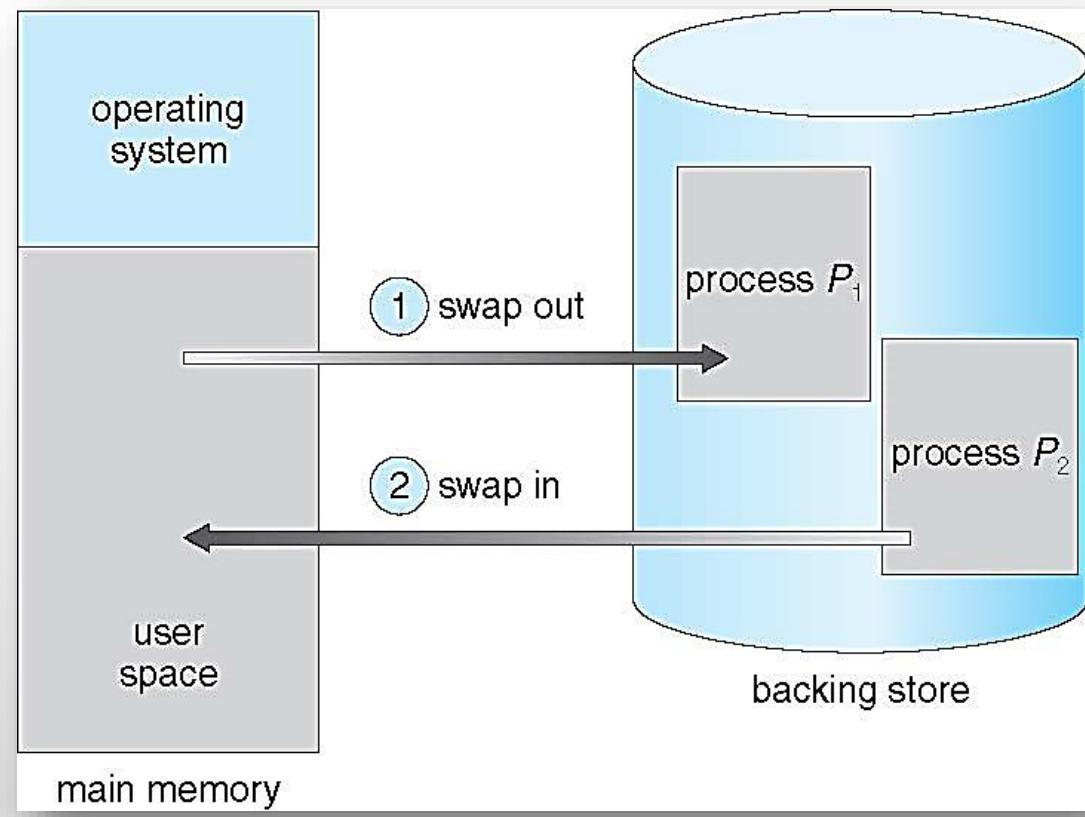


(b)



# Sayfalama Olmadan Takas İşlemi

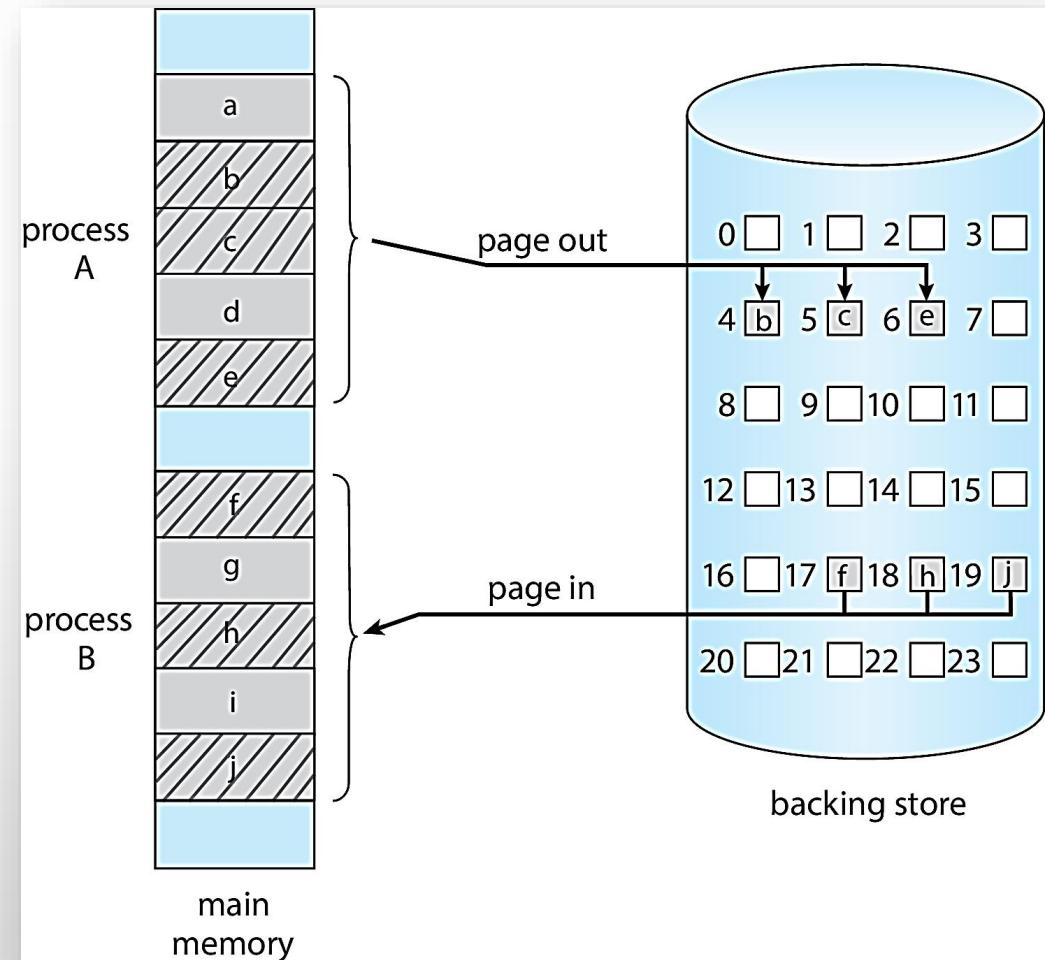
■





# Sayfalama İle Takas İşlemi

- .



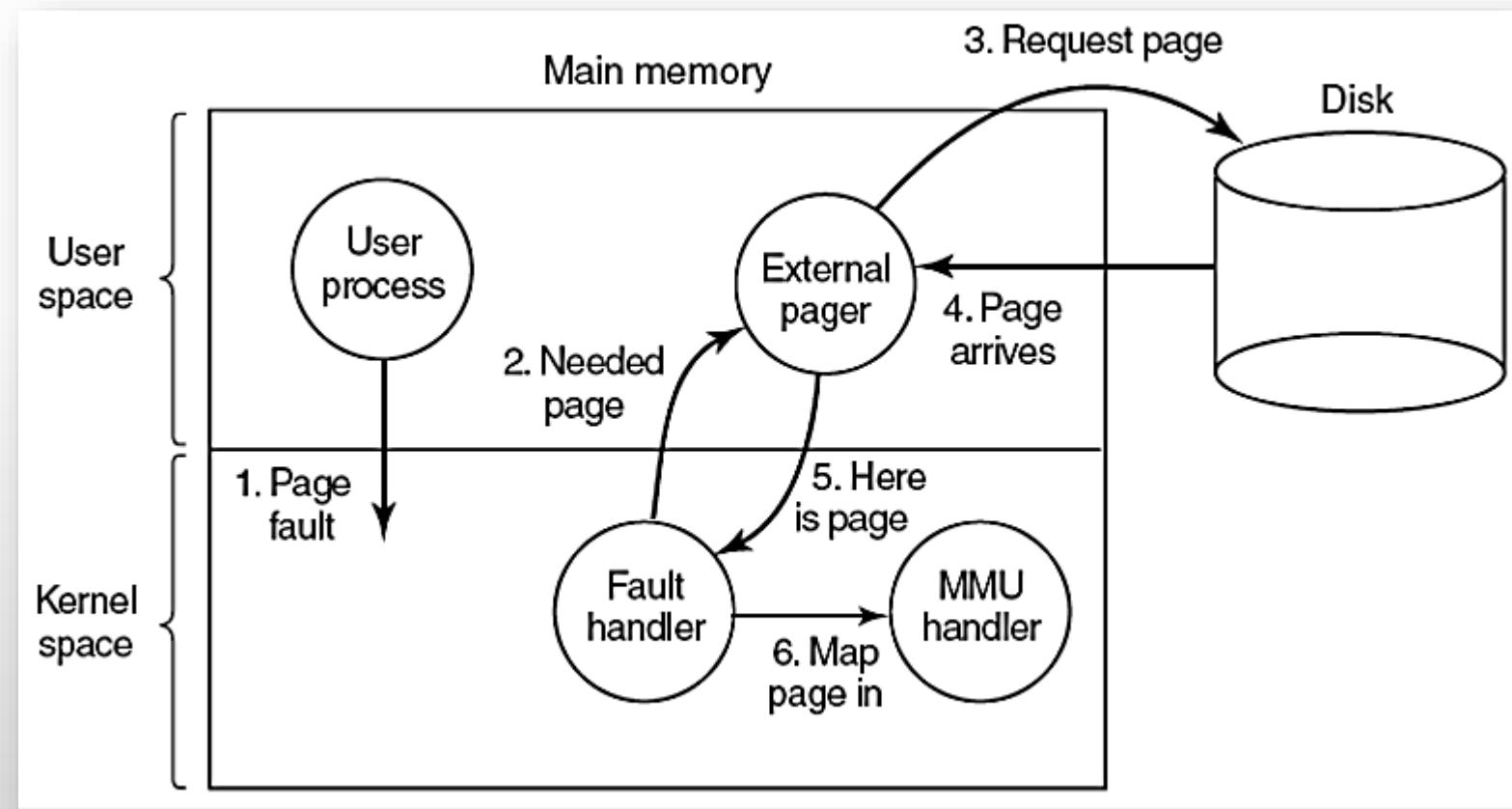


# İlke ve Mekanizma Ayrımı (Policy and Mechanism)

- Bellek yönetim sistemi üç bölüme ayrılır:
  - Alt düzey (*low level*) çalışan MMU işleyicisi (*handler*).
  - Çekirdeğin parçası olarak sayfa hatası (*page fault*) işleyicisi.
  - Kullanıcı alanında (*user space*) çalışan harici sayfalayıcı (*pager*).



# İlke ve Mekanizma Ayrımı (Policy and Mechanism)





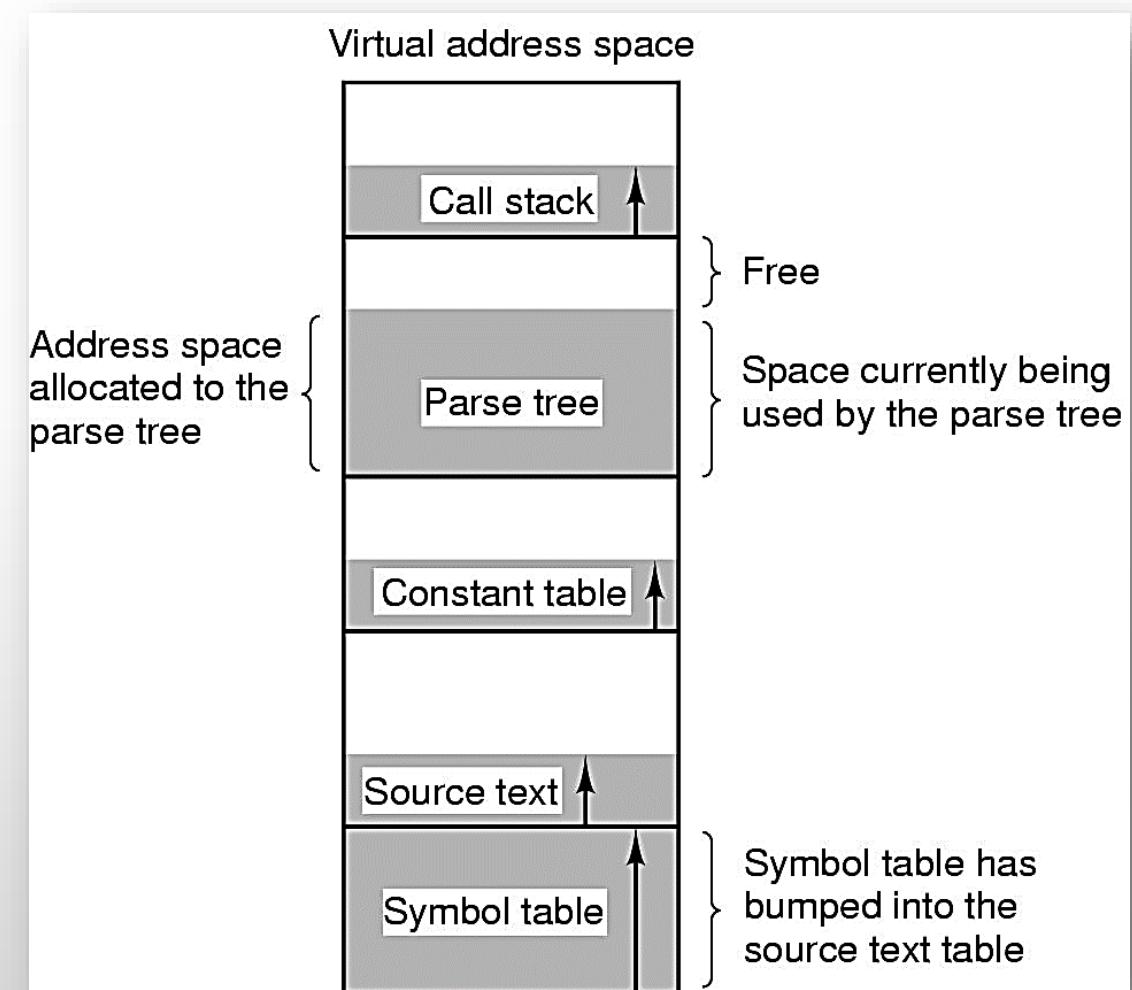
# Kesimleme (segmentation)

- Derleyici (*compiler*), derleme aşamasında birçok tablo oluşturur:
  - Toplu (*batch*) sistemlerde listeleme için *kaynak kod*.
  - Değişkenlerin adları ve niteliklerini tutan *sembol tablosu*.
  - Kullanılan tamsayı, kayan noktalı *sabitleri* (*constants*) içeren tablo.
  - Sözdizimsel (*syntactic*) analizi içeren *ayrıştırma* (*parse*) ağıacı.
  - Derleyici içinde *prosedür çağrıları* için kullanılan yığın.



# Kesimleme (segmentation)

- Tek boyutlu adres uzayı.





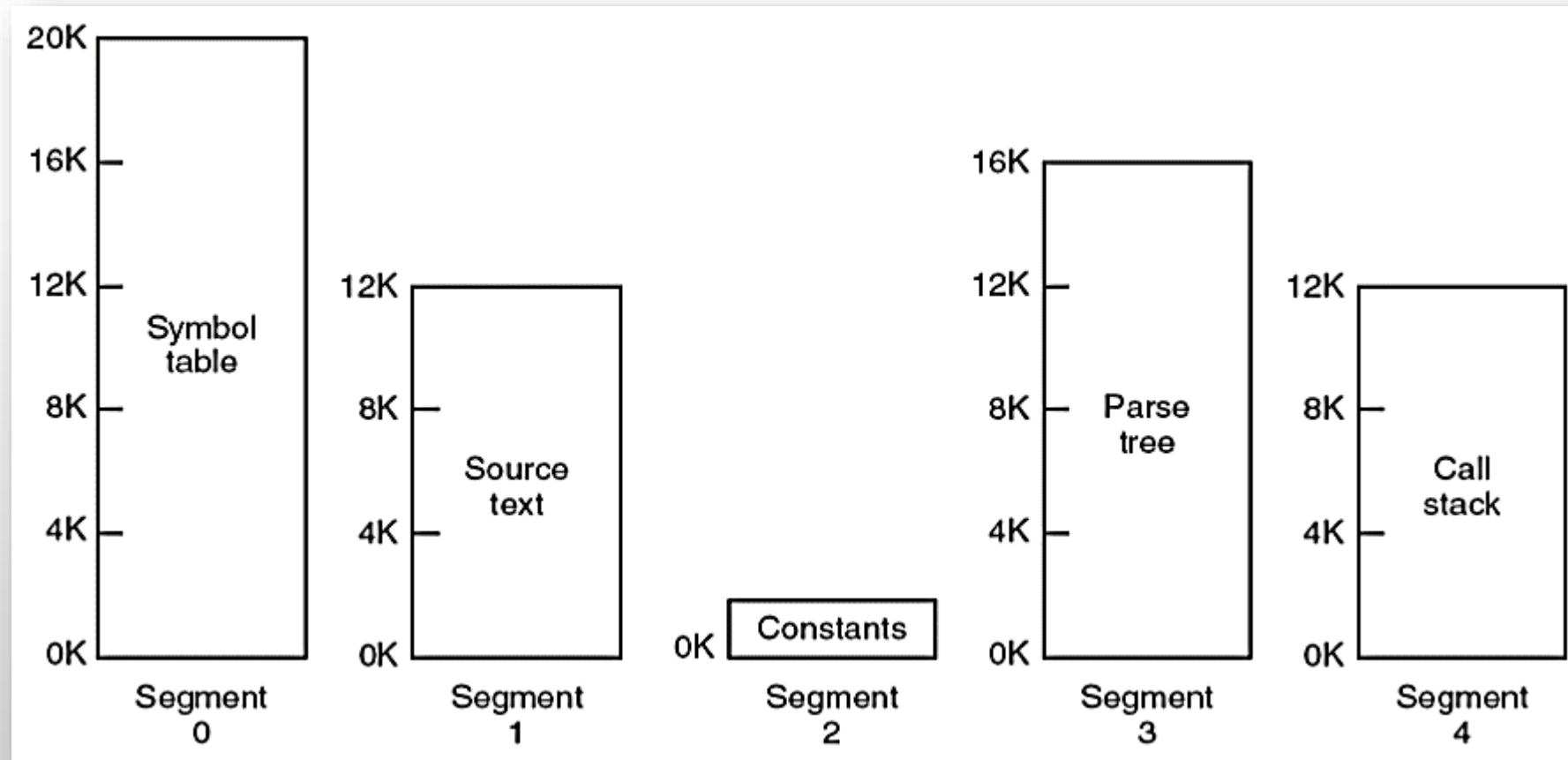
# Kesimleme Avantajları

- Büyüyen ve küçülen veri yapılarının ele alınmasını basitleştirir.
- *Kesim n'nin adres alanı ( $n$ , yerel adres) biçimindedir,*
  - $(n,0)$  başlangıç adresidir.
- Kesimler birbirinden ayrı olarak derlenebilir.
- Kütüphane bir kesime konabilir ve paylaşılabilir.
- Farklı kesimler için farklı koruma ( $r,w,x$ ) izinleri verilebilir.



# Kesimleme (segmentation)

- Kesimlere ayrılmış bellek yapısı.





# Sayfalama ve Kesimleme Karşılaştırması

Durum	Sayfalama	Kesimleme
Programcı tekniğin kullanıldığından farkında olmalı mı?	Hayır	Evet
Kaç tane doğrusal adres alanı var?	1	Çok
Toplam adres alanı, fiziksel belleğin boyunu aşabilir mi?	Evet	Evet
Prosedür ve veriler ayrıt edilip, ayrı ayrı korunabilir mi?	Hayır	Evet
Değişken boyutlu tablolar kolayca yerleştirilebilir mi?	Hayır	Evet
Prosedürler kullanıcılar arasında kolayca paylaşılabilir mi?	Hayır	Evet
Neden icat edildi?	Fiziksel belleğe ihtiyaç duymadan geniş bir doğrusal adres alanı elde etmek	Program ve verilerin mantıksal olarak bağımsız adres alanlarına ayrılabilmesi



# SON