Bölüm 8: Bellek Yönetimi

İşletim Sistemleri

Bellek Yönetimi

- Programın çalışabilmesi için diskten belleğe getirilmesi ve bir süreç içerisine yerleştirilmesi gerekir.
- Ana bellek ve yazmaçlar, CPU'nun doğrudan erişebildiği depolama alanlarıdır.
- Bellek birimi yalnızca şu akışları görür:
  - adres + okuma istekleri veya
  - adres + veri ve yazma istekleri
- Yazmaç erişimi bir CPU döngüsünde yapılır
- Ana bellek erişimi, duraklamaya neden olacak şekilde birçok döngü alabilir
- Önbellek (cache), ana bellek ile yazmaçlar arasında bulunur

Bellek Yönetimi

- Bellek kısıtlı bir kaynak
- Bu nedenle bellek hiyerarşisi var
  - Önbellek (cache)(hızlı)
  - Anabellek (orta)
  - Disk (yavaş)
- Bellek yöneticisi, kolayca erişilebilen bir bellek soyutlaması yaratmak için bu hiyerarşiyi kullanır.

Bellek Yönetimi

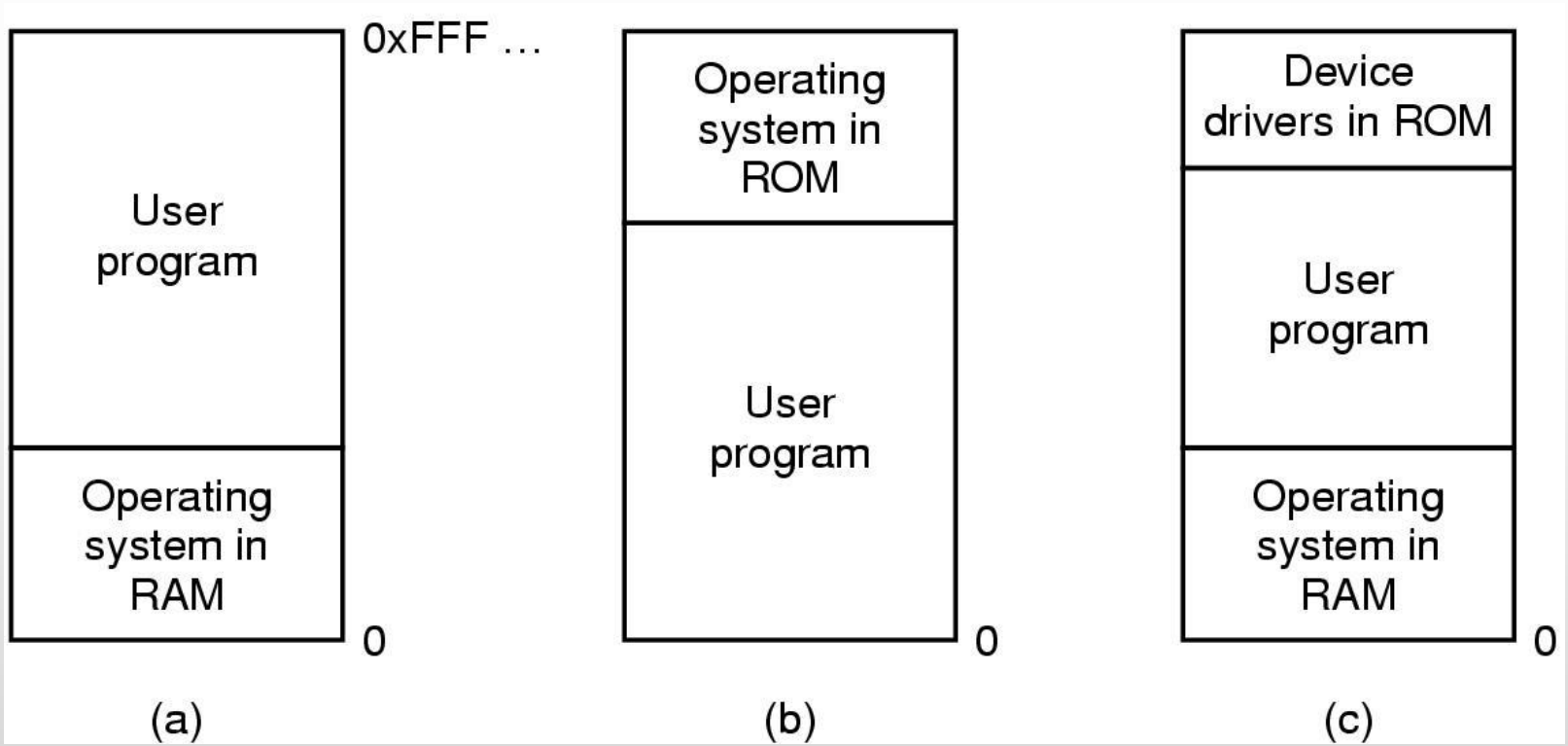
- Bellek (RAM) önemli ve kısıtlı bulunan bir kaynaktır
  - Programlar, genişleyerek kendilerine sunulan belleği doldururlar
- Programcının istediği
  - Belleğin korumalı, sonsuz büyük, sonsuz hızlı, ve kalıcı olması..
- Gerçekte olan
  - Akla gelen en iyi çözüm: bellek hiyerarşisi
  - Yazmaç, önbellek,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

Soyutlama Olmazsa

- Eskiden anabilgisayar, mini bilgisayarlar, kişisel bilgisayarlarda bellek soyutlaması yoktu…
  - MOV REGISTER1, 1000
  - Fiziksel bellek adresi 1000'in içeriğini yazmaca taşır
- Bellekte aynı anda iki süreç yer alamaz

Soyutlama Olmazsa

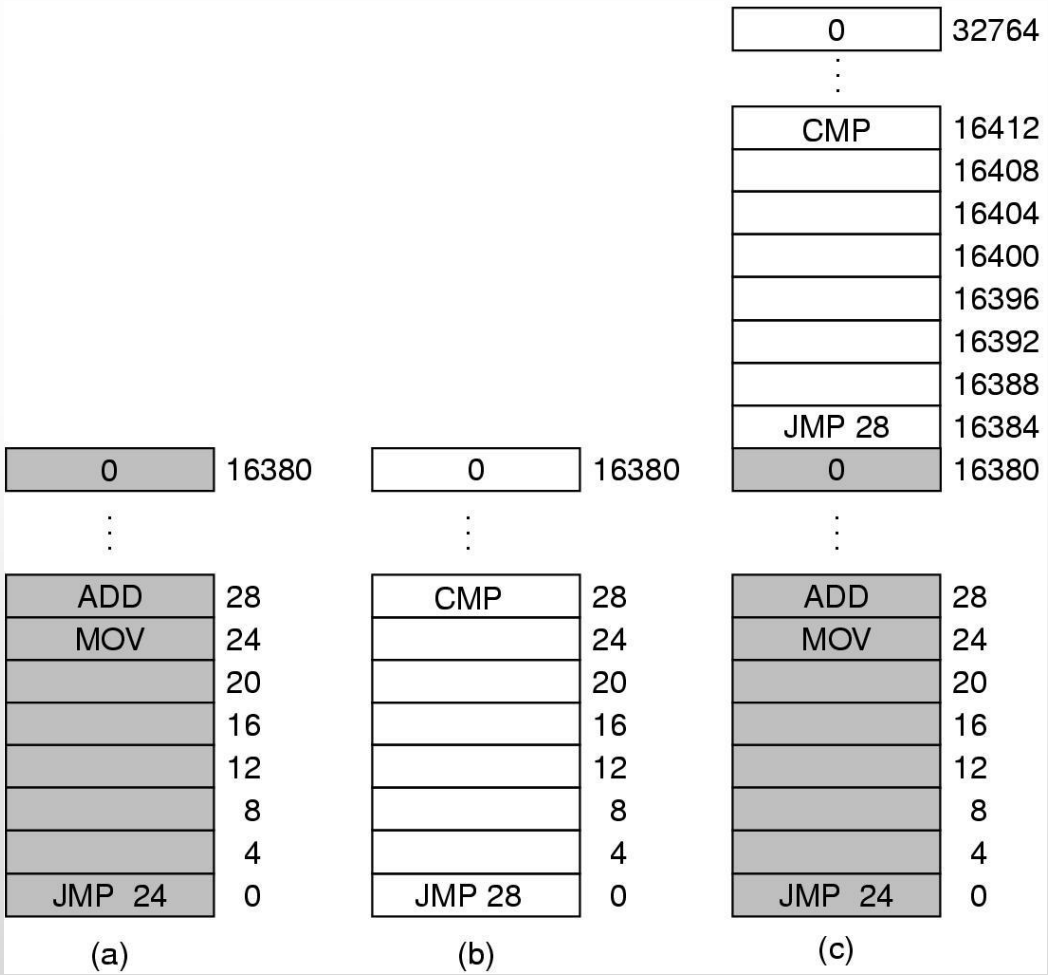
- İşletim sistemi ve bir süreç ile belleğin düzenlenmesi



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 Problemi

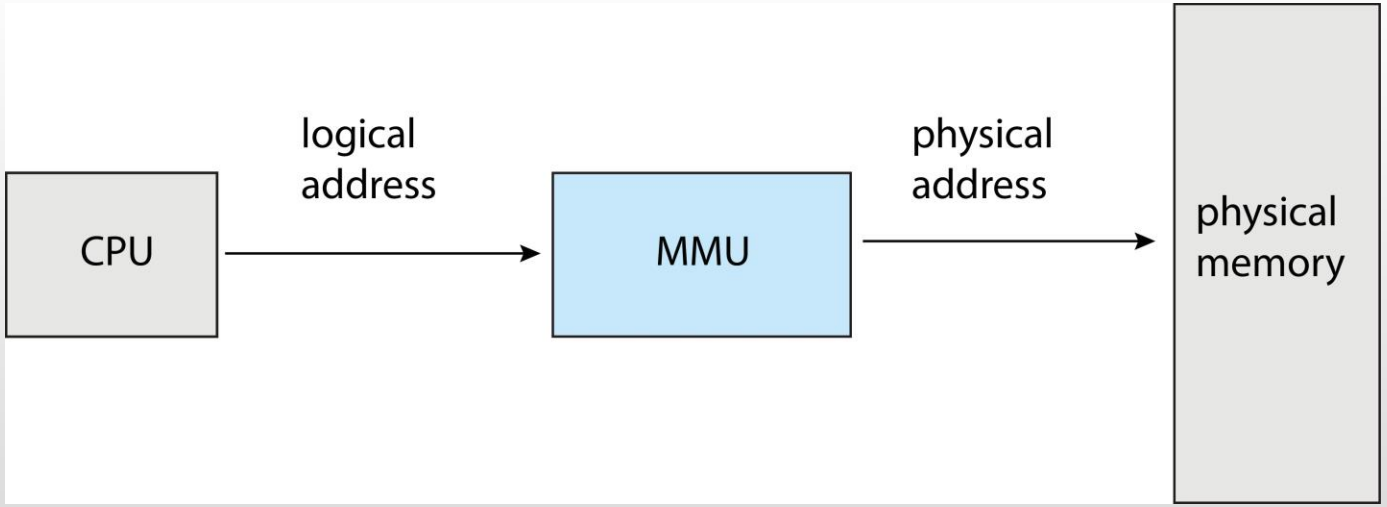


Statik Yer Değiştirme

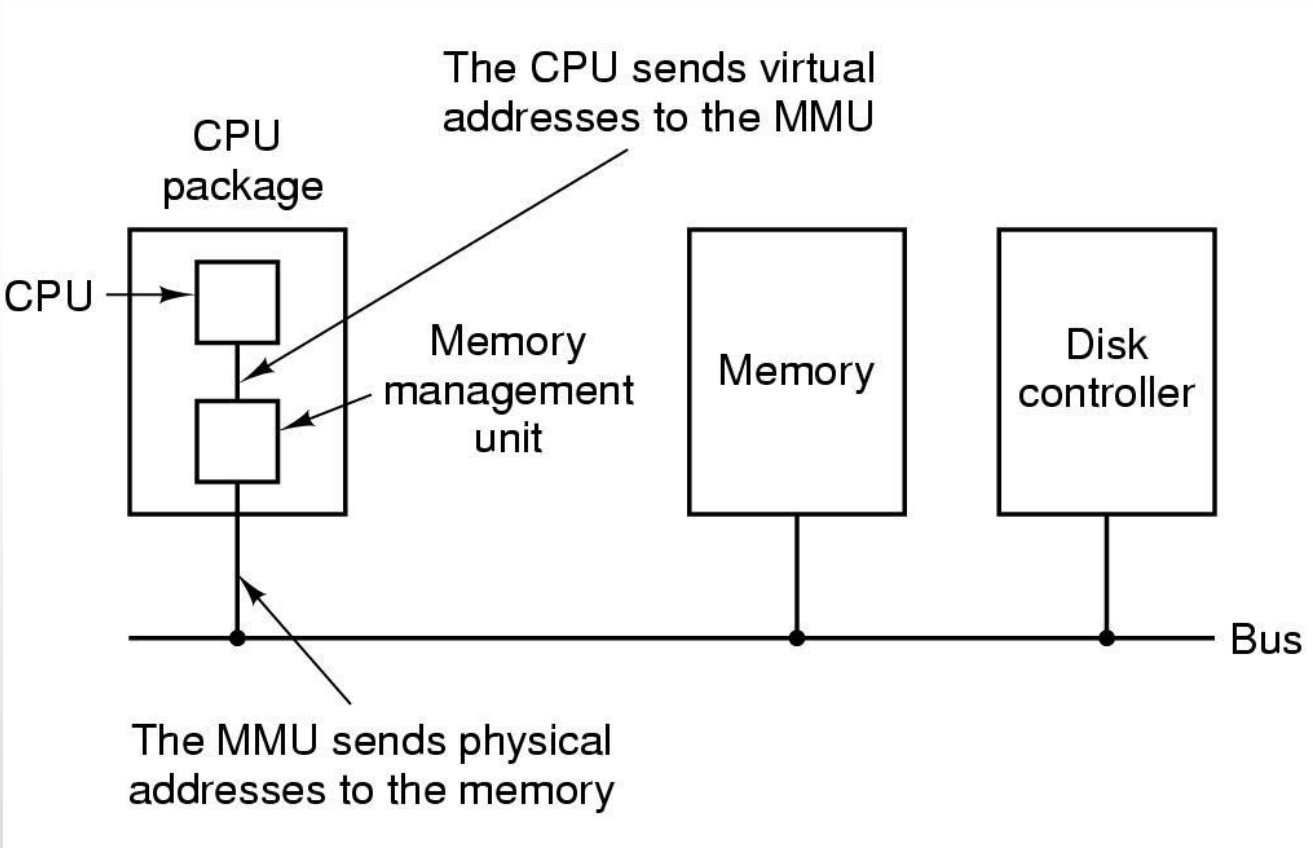
- Sorun, her iki programın da mutlak fiziksel belleğe referans vermesidir.
- Statik yer değiştirme - programın ilk komutunu x adresine yükler ve yükleme sırasında sonraki her adrese x ekler
  - Bu çok yavaş ve
  - Tüm adresler değiştirilemez
    - MOV REGISTER 1,28 değiştirilemez

Bellek Yönetim Birimi

- MMU(memory management unit) mantıksal adresi fiziksel adrese çevirir.
- Mantıksal ve fiziksel adresler, derleme zamanı ve yükleme zamanında aynıdır; yürütme zamanında adres eşleme şemasında farklılık gösterir

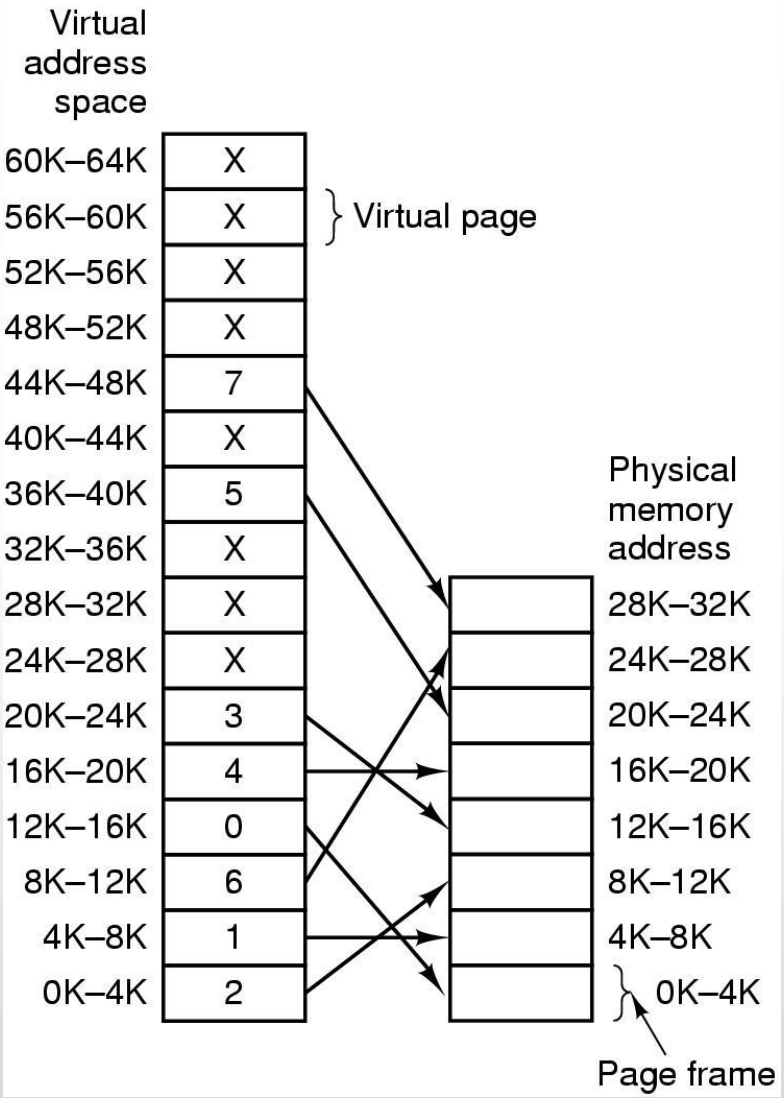


MMU Konum ve İşlevi



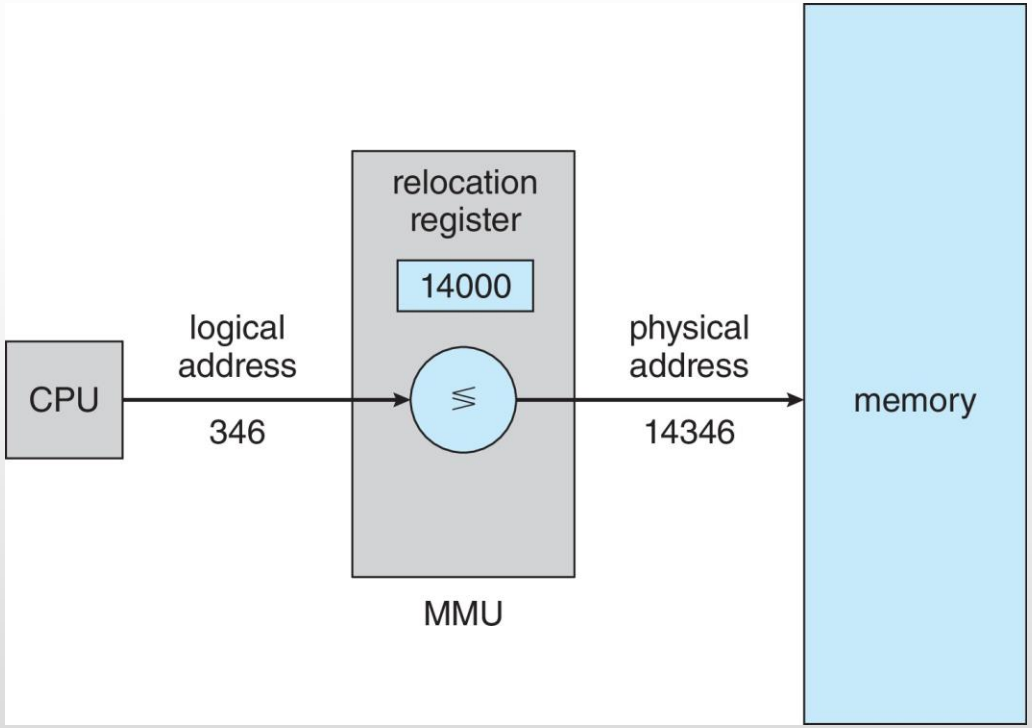
Bellek Yönetim Birimi

- 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 yazmaç artık yer değiştirme yazmacı olarak adlandırılıyor



Adres Uzayı

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

Soyutlama Olmamasının Dezavantajı

- Sorun, her iki programın da mutlak fiziksel belleğe referans vermesidir.
- İnsanlar mahrem bir alana, yani yerel adreslere sahip olabilmek isterler.
- IBM 360
  - İkinci program belleğe yüklenirken adresler değiştirilir
  - Statik yer değiştirme
    - 16384'e bir program yüklenirken, her adrese bir sabit değer eklenir.
    - Yüklemeyi yavaşlatır, ek bilgi gerektirir
- Gömülü ve akıllı sistemlerde soyutlama olmadan bellek yönetimi var

Soyutlama: Adres Uzayı

- Fiziksel adresi programcılara gösterme (not expose)
  - İşletim sistemi çökertilebilir
  - Paralelleştirmek 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

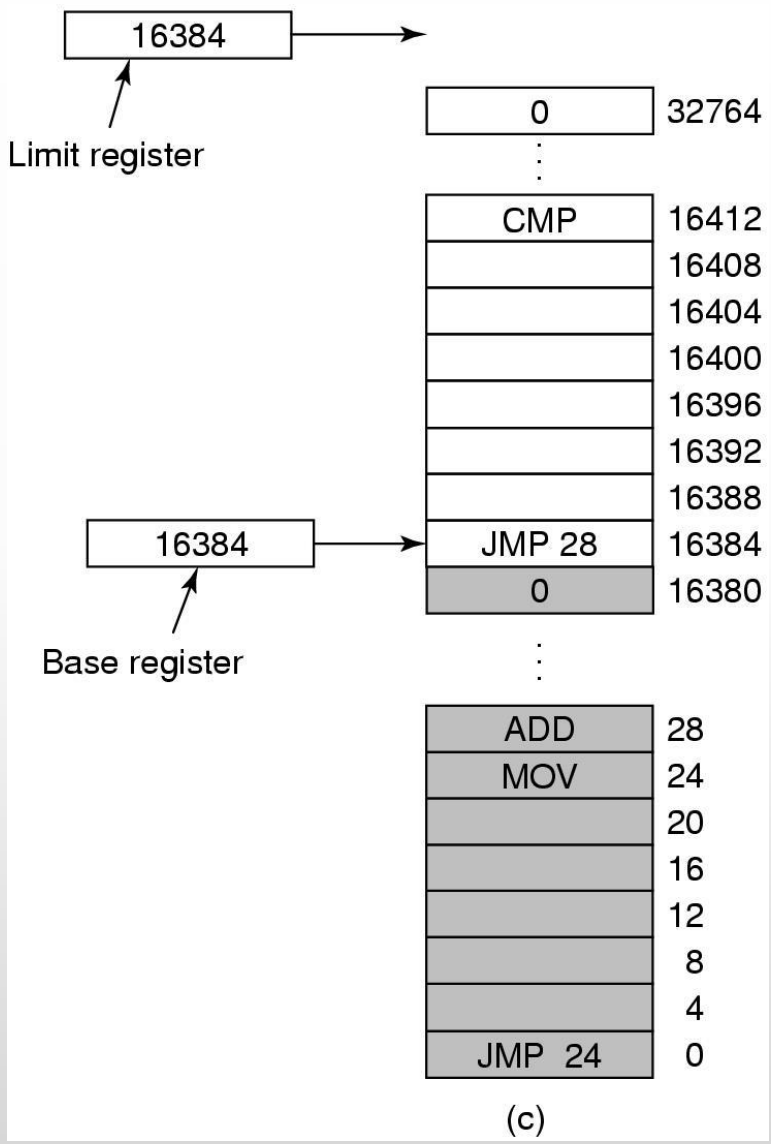
- İşlemciye iki özel yazmaç:
  - taban ve limit
- Program ardışık bir boş alanlara yüklenecek
- Yükleme sırasında yer değiştirme yok
- Süreç çalıştırıldığında ve bir adrese referans verildiğinde, CPU otomatik olarak limit değerini aşıp aşmadığını kontrol ederek taban değerini o adrese ekler

Taban ve Limit Yazmaçları

- Bir tür dinamik yer değiştirme
- Taban, programın başlangıç adresini içerir
- Limit programın uzunluğunu içerir
- Program belleğe başvurur, süreç tarafından oluşturulan adrese temel adresi ekler.
- Adresin limitten büyük olup olmadığını kontrol eder.
  - Eğer öyleyse, hata oluşturur
- Dezavantaj – her adımda ekleme ve karşılaştırma yapılmalıdır.
- CDC 6600 ve Intel 8088'de kullanılır

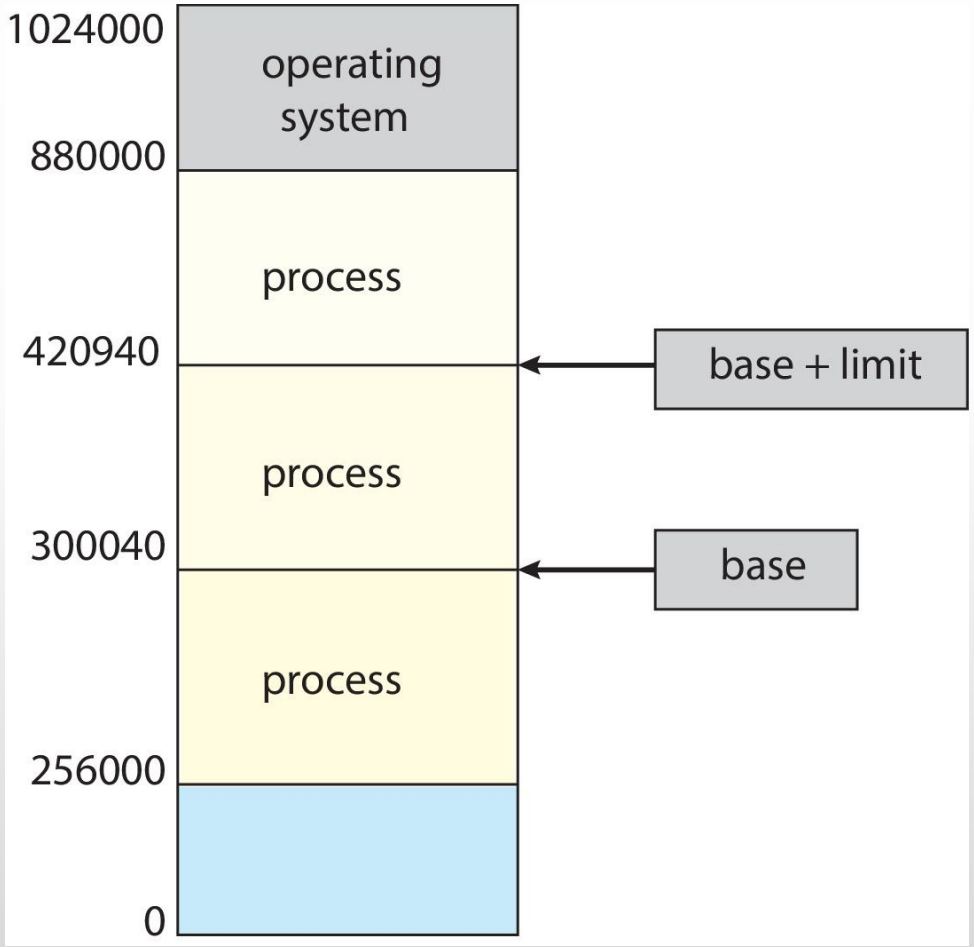
Taban ve Limit Yazmaçları

- Her bellek erişiminde bir ekleme ve karşılaştırma yapılması gerekiyor



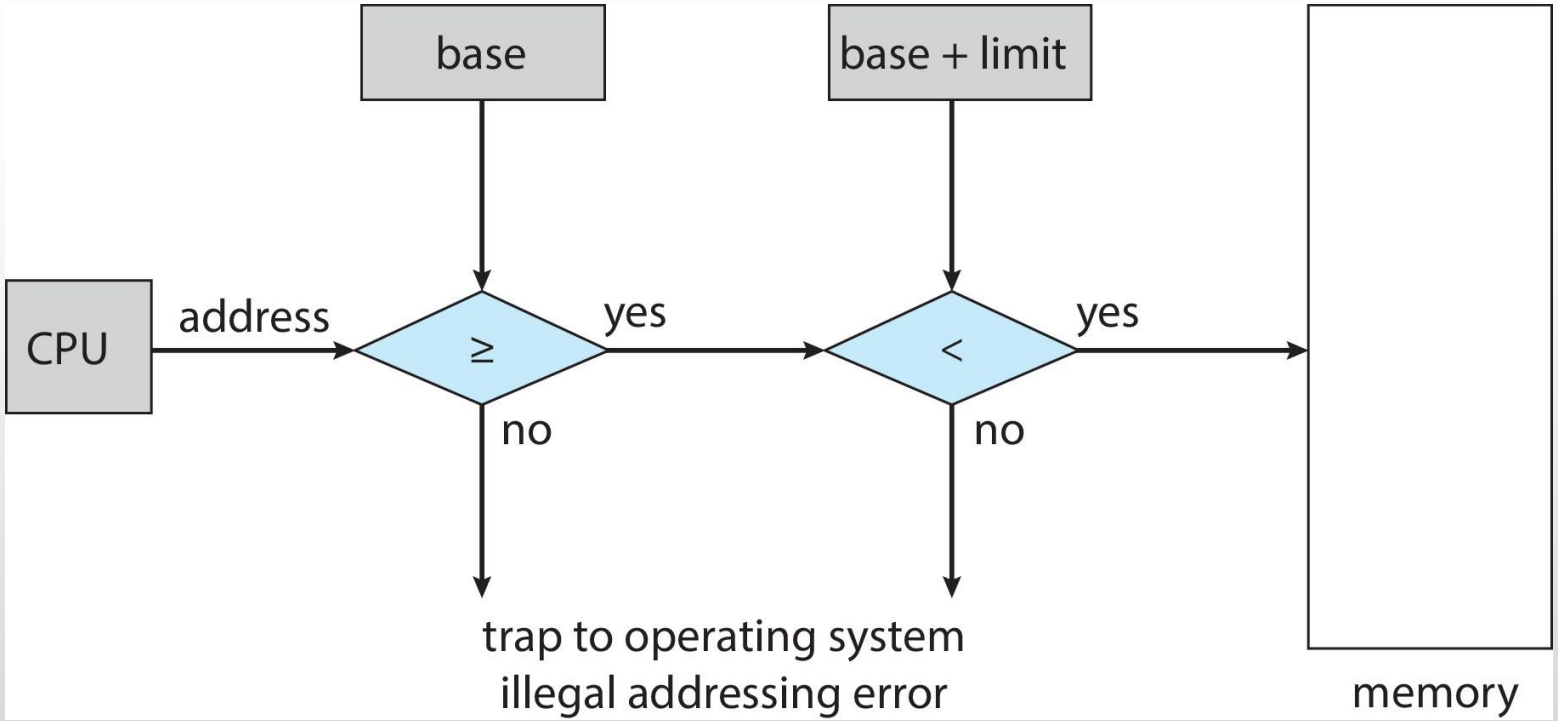
Taban ve Limit Yazmaçları

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



Donanımsal Adres Kontrolü

- CPU, kullanıcı modunda oluşturulan her bellek erişimini, o kullanıcı için taban ile sınır arasında olduğundan emin olmak için kontrol etmelidir.



Bellek Adresleri

- Talimatların ve verilerin bellek adreslerine eşlenmesi üç farklı aşamada gerçekleşebilir.
- **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 üretilmelidir
- **Yürütme zamanı**: Süreç yürütülürken bir bellek bölümünden diğerine taşınabiliyorsa, eşleme çalışma zamanına kadar ertelenir
- Adres haritaları için donanım desteği gerekli (ör. taban ve limit yazmaçları)

Dinamik Yükleme

- Tüm programın yürütülmesi için bellekte olması gerekir
- Prosedür çağrılana kadar yüklenmez
- Daha iyi bellek alanı kullanımı;
  - kullanılmayan prosedür asla yüklenmez
- Tüm prosedürler, yeri değiştirilebilen yükleme biçiminde diskte tutulur
- Büyük miktarda kodun yüklenmesi gerektiğinde kullanışlıdır.
- İşletim sisteminden özel bir destek gerekmez
  - Program tasarımı yoluyla uygulanır
  - İşletim sistemi, dinamik yüklemeyi uygulamak için kütüphane sağlayarak yardımcı olabilir

Dinamik Bağlama

- **Statik bağlama**: yükleyici tarafından ikili (binary) program imajında birleştirilen sistem kütüphaneleri ve program kodu
- **Dinamik bağlama**: bağlama yürütme zamanına kadar ertelenir
- **Stub** bellekte yerleşik uygun kütüphane yordamını bulmak için kullanılan küçük kod parçası,
- Stub, altprogramın adresiyle kendisini 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
  - Paylaşılan kütüphaneler

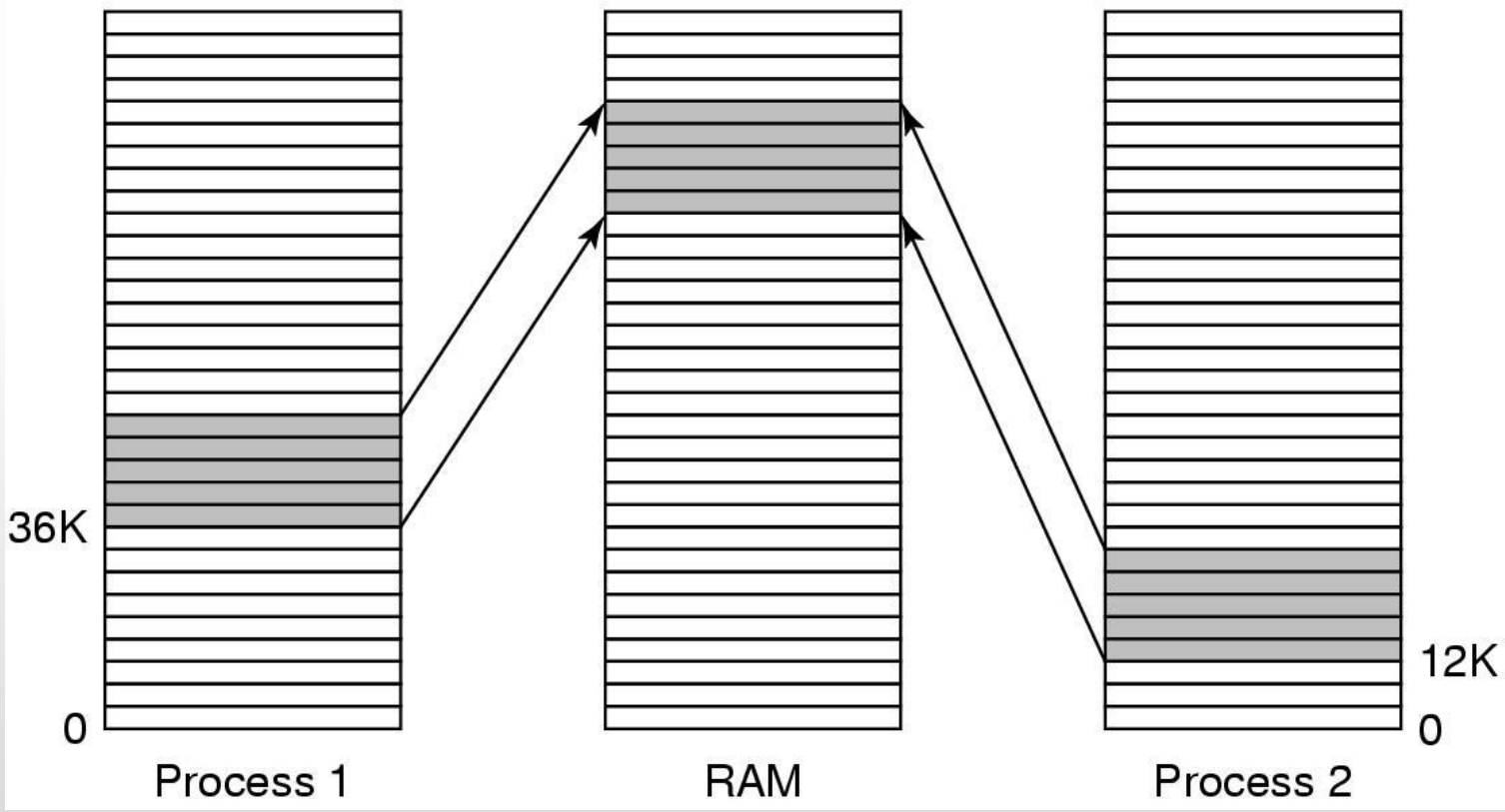
Paylaşımlı Kütüphaneler

- Birçok süreç tarafından kullanılan büyük kütüphaneler (ör. grafikler).
- Kullanmak isteyen her süreç için belleğe yüklemek çok pahalı. Bunun yerine paylaşılan kütüphaneler kullanılır.
- Unix bağlama (link): ld*.o –lc –lm . .o uzantılı dosyalarda bulunmayan dosyalar m veya c kütüphanelerinde bulunur ve ikili dosyalara dahil edilir.

Paylaşımlı Kütüphaneler

- Bağlayıcı, çalışma zamanında çağrılan fonksiyona bağlanan bir saplama (stub) yordamını çağırmak için kullanır.
  - Paylaşılan kitaplık yalnızca bir kez yüklenir (ilk kez içindeki bir işleve erişilmek istendiğinde).
- Yanlış adrese gitmeyi önlemek için konumdan bağımsız kod (position independent code) kullanmak gerekir.
  - Derleyici, paylaşılan kitaplıkları kullanırken mutlak adresler üretmez; yalnızca göreli (relative) adresler.

Paylaşımlı 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

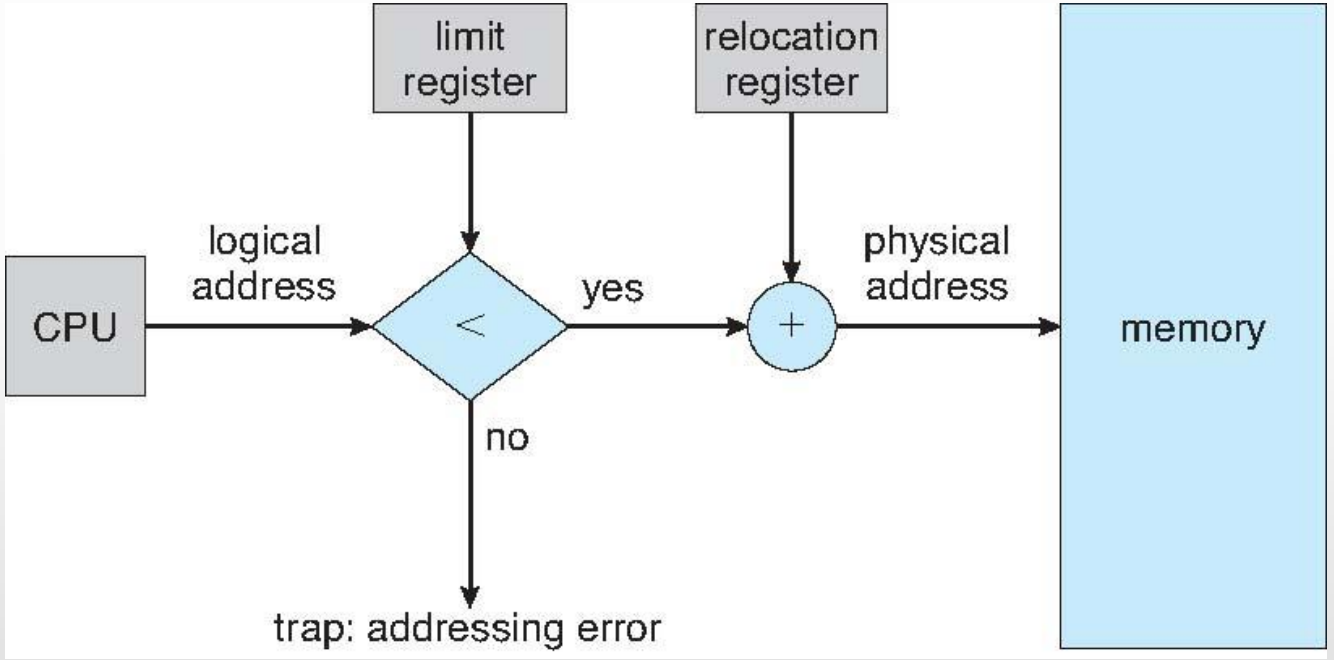
- $ gcc -fPIC -c func.c -o func.o
- $ $gcc -shared -o libfunc.so func.o
- $ export LD_LIBRARY_PATH=$(pwd)
- $ ./main

Bitişik Yer Tahsisi

- Ana bellekte hem işletim sistemi hem de kullanıcı süreçleri yer alır
- Sınırlı kaynak, verimli bir şekilde tahsis edilmelidir
- Bitişik yer tahsisi eski bir yöntemdir
- Ana bellek genellikle iki bölüme ayrılır:
  - Yerleşik işletim sistemi, kesilme vektörü ile belleğin alt kısmında tutulur
  - Kullanıcı süreçleri belleğin üst kısmında tutulur
- Her süreç bellekte tek bir bitişik bölümde yer alır

Bitişik Yer Tahsisi

- .



Programlar Koştukça Büyür

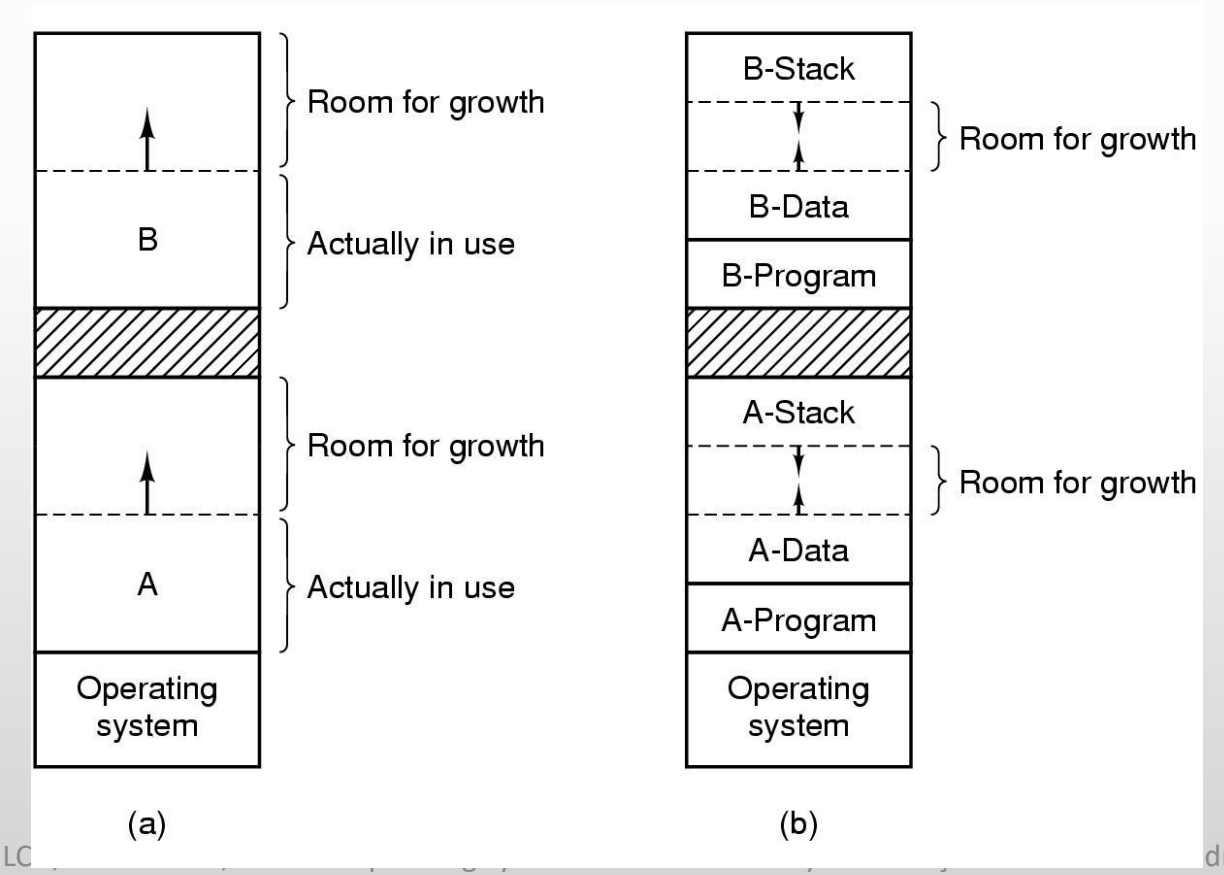
- **Yığın** kesimi (dönüş adresleri ve yerel değişkenler)
- **Veri** kesimi (dinamik olarak tahsis edilen ve serbest bırakılan değişkenler için yığın)
- Her ikisi için de fazladan bellek ayırmak iyi bir fikirdir.
- Program diske gönderildikten sonra belleğe tekrar getirilirken onunla birlikte delikler (hole) getirmeyin!

Değişken Bölümleme

- Verimlilik için değişken bölüm boyutları (belirli bir sürecin ihtiyaçlarına göre boyutlandırılmıştır)
- Delik (hole) - kullanılabilir bellek bloğu; çeşitli boyutlardaki delikler bellek boyunca dağılmış durumda
- Bir süreç geldiğinde, onu konumlandırmak için yeterince büyük bir bellek tahsis edilir.
- Sonlanan süreç, kendi bölümünü serbest bırakır, bitişik boş bölümler birleştirilir
- İşletim sistemi aşağıdakiler hakkında bilgi tutar:
  - a) tahsis edilen bölümler
  - b) boş bölümler (delik)

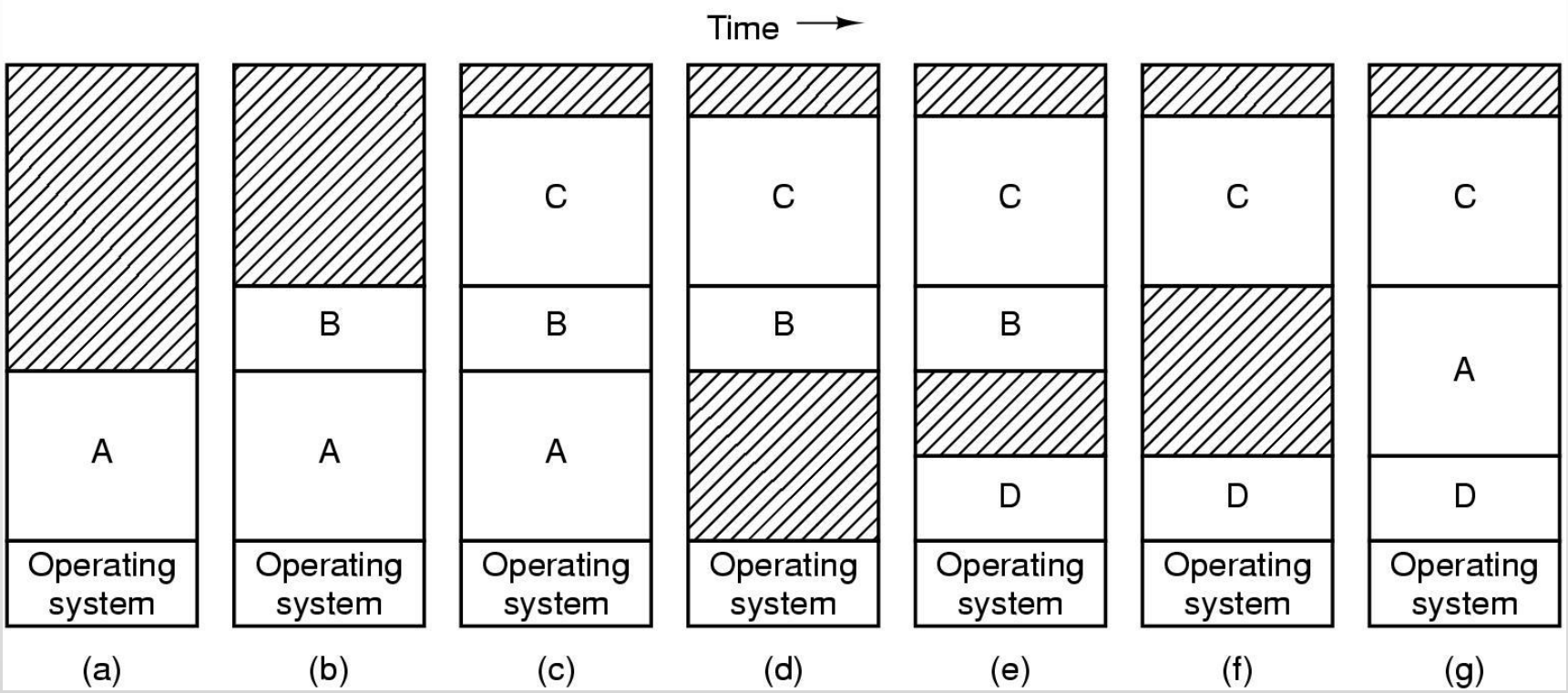
Bellekte Alan Tahsisi

- (a) Büyüyen veri segmenti için. (b) Büyüyen yığın, ve veri kesmi için.



Bellek Düzeni Değişimi

- Süreçler belleğe girip çıktıkça bellek tahsisi değişir. Gölgeli bölgeler kullanılmayan bellektir.



Problemler

- Giriş ve çıkış takası sonrası farklı adresler
  - Statik yer değiştirme/dinamik yer değiştirme
- Bellek delikleri (hole)
  - Bellek sıkıştırma
    - İşlemci zamanı gerekir
    - 20 ns'de 4 byte taşır, ardından 1 GB'ı sıkıştırmak için 5 saniye
- Bir program için ayrılan bellek miktarı ne kadar?
  - Programlar büyüme eğilimindedir
  - Hem veri kesimi (segment) hem de yığın

Boş Alan Yönetimi

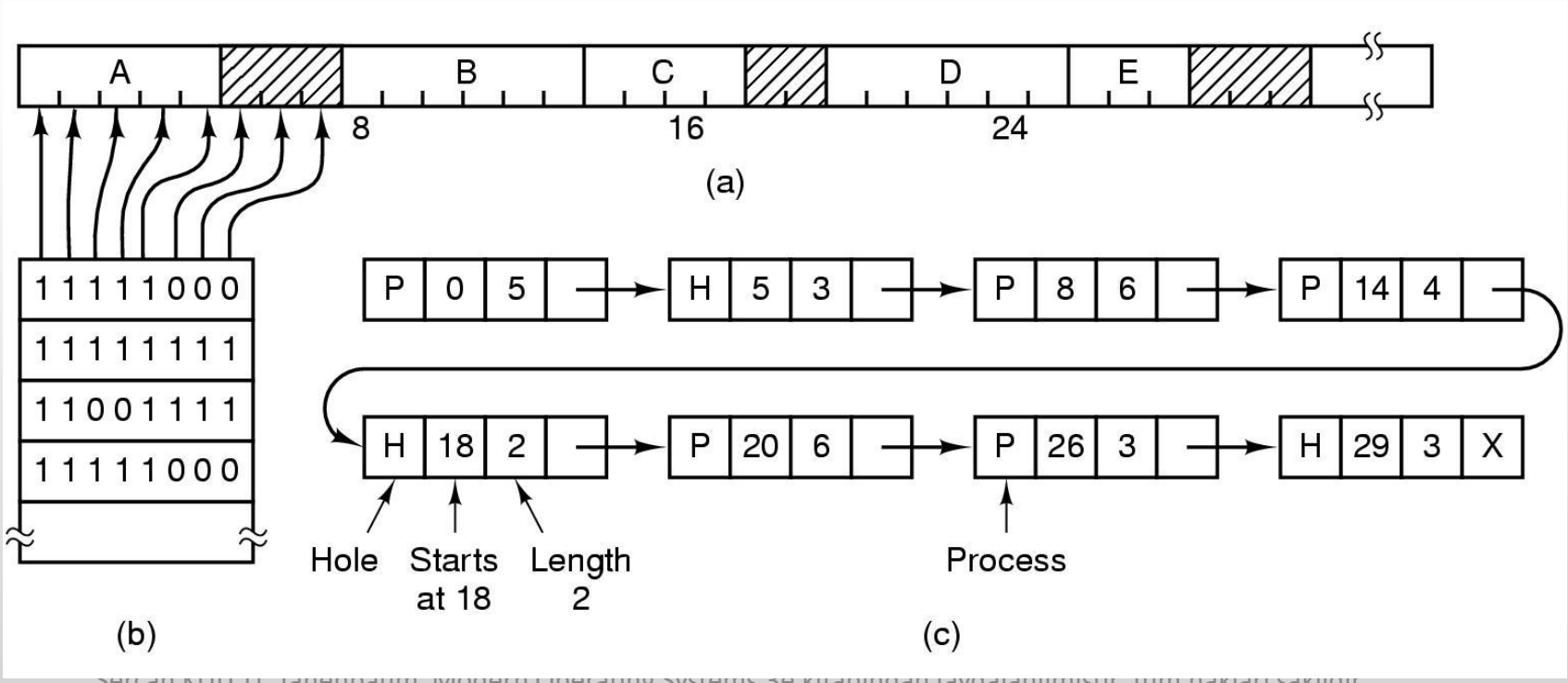
- Programlara boş bellek alanı nasıl tahsis edilir?
  - İlk uyan (first fit)
    - Hızlı; İlk kesim daha sık kullanılır; büyük bir boş alan boşa gidebilir
  - Sonraki uyan (next fit)
    - Her seferinde en son kullanılan yerden aramaya başlanır
  - En iyi uyan (best fit)
    - Tüm listeyi arar, gerekli boyuta en yakın deliği bulur
  - En kötü uyan (worst fit)
    - En büyük deliği bulur
  - Hızlı uyan (quick fit)
    - Süreçler ve delikler ayrı kuyruklarda tutulur

Bellekte Boş Alan Yönetimi

- Biteşlem (bitmap) ve bağlı listeler
- Biteşlem
  - Hafızayı takip etmenin kompakt yolu
  - Bellek, birimlere ayrılmıştır (birkaç sözcükten KB boyutuna kadar)
  - Her birime karşılık gelen, biteşlem'de bir bit var
  - k birim uzunluğundaki bir dosyayı getirmek için hafızada k ardışık sıfır aramak gerekir
  - İstenen uzunlukta boş alan bulmak zor

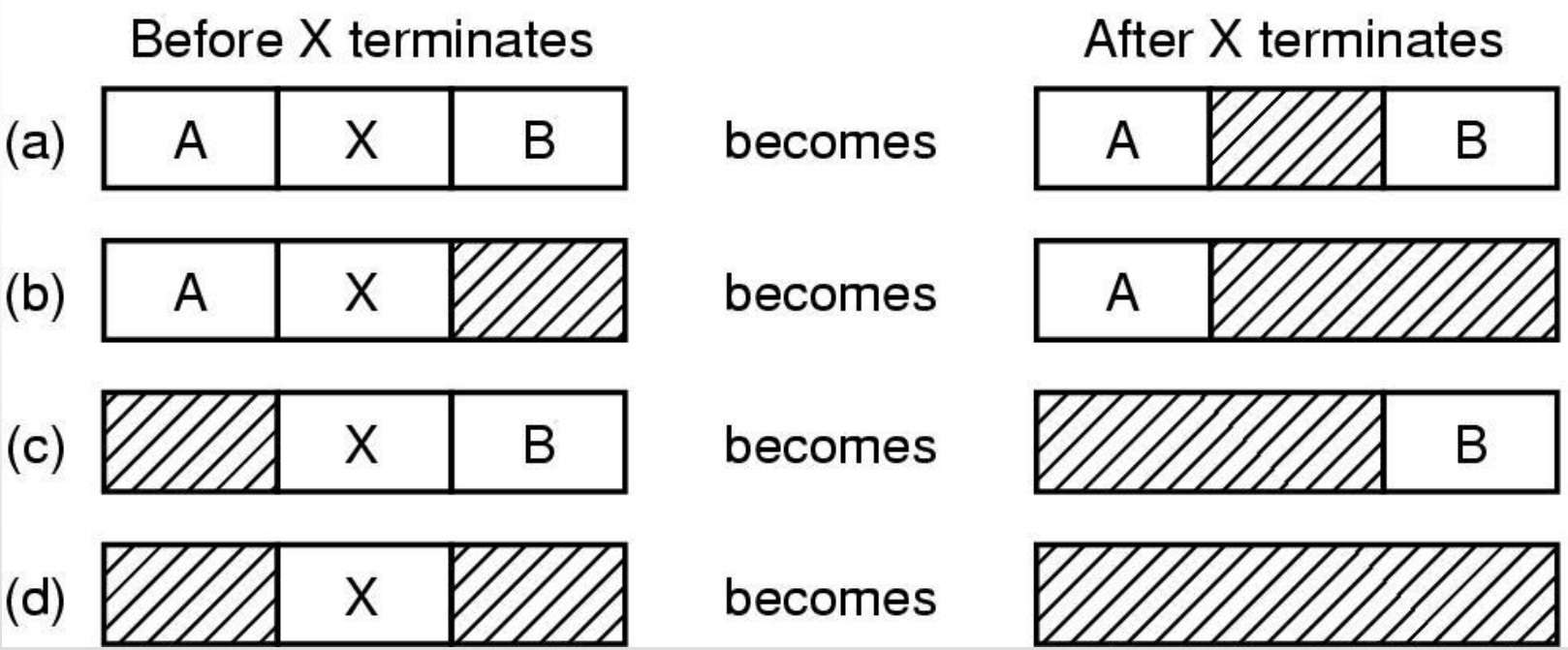
Biteşlem ile Bellek Yönetimi

(a) Belleğin bir bölümü, beş işlem ve üç delik var. İm işaretleri, bellek ayırma birimlerini gösterir. Gölgeli bölgeler (bit eşlemde 0) boştur. (b) ilgili bit eşlem. (c) bağlı liste gösterimi.



Bağlı Liste ile Bellek Yönetimi

- X Sürecini sonlandırdıktan sonra oluşan bellek düzeni.



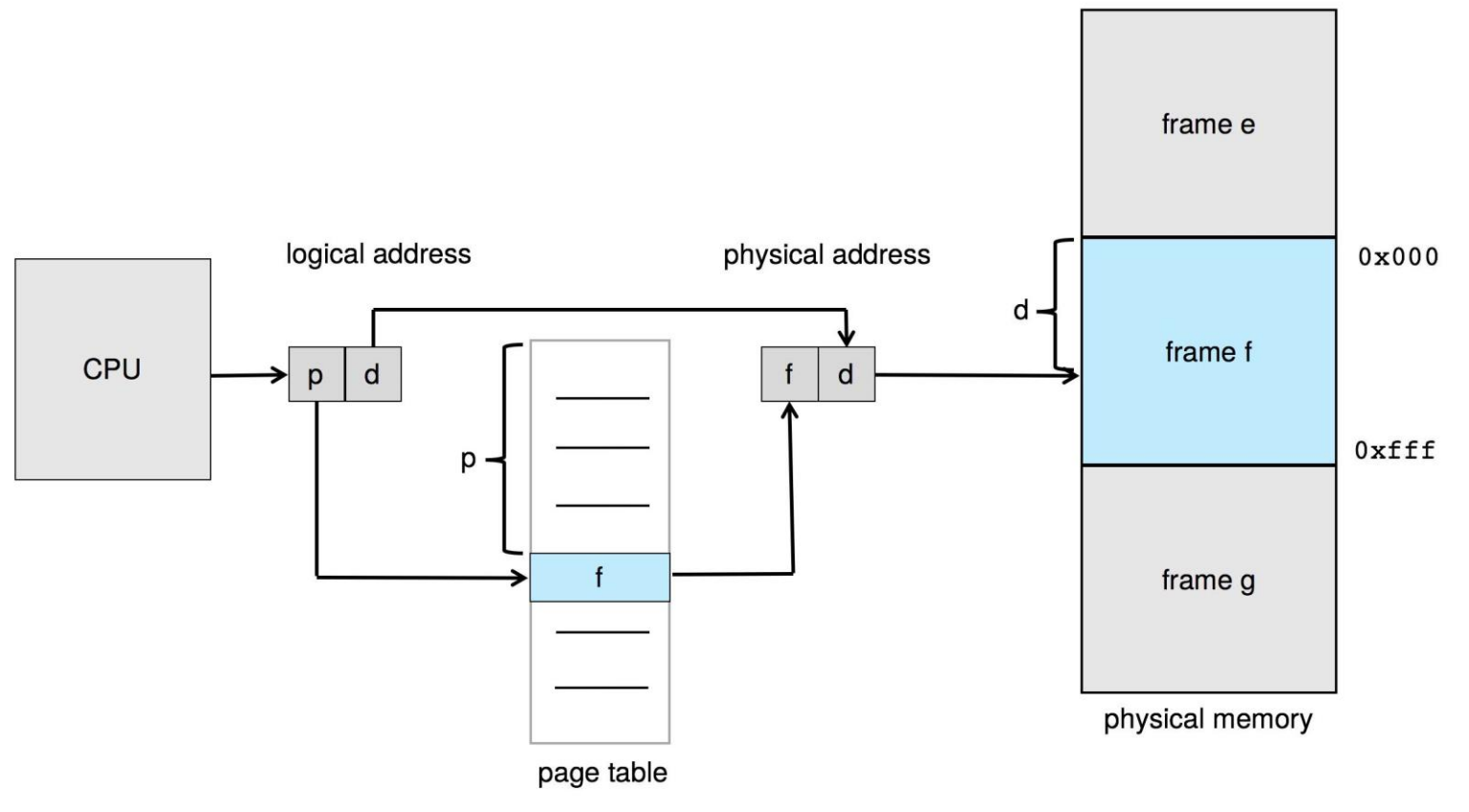
Parçalanma

- **Harici Parçalanma** - bir isteği karşılamak için toplam bellek alanı var, ancak bitişik değil
- **Dahili Parçalanma** – tahsis edilen hafıza, talep edilen hafızadan biraz daha büyük olabilir; bu boyut farkı, bir bölümün içindeki, kullanılmayan bellektir.
- Sıkıştırarak harici parçalanma azaltılabilir
  - Tüm boş belleği tek bir büyük blokta bir araya getirmek için bellek içeriğini karıştırır
  - Sıkıştırma, yer değiştirme dinamikse ve yürütme zamanında yapılırsa mümkündür

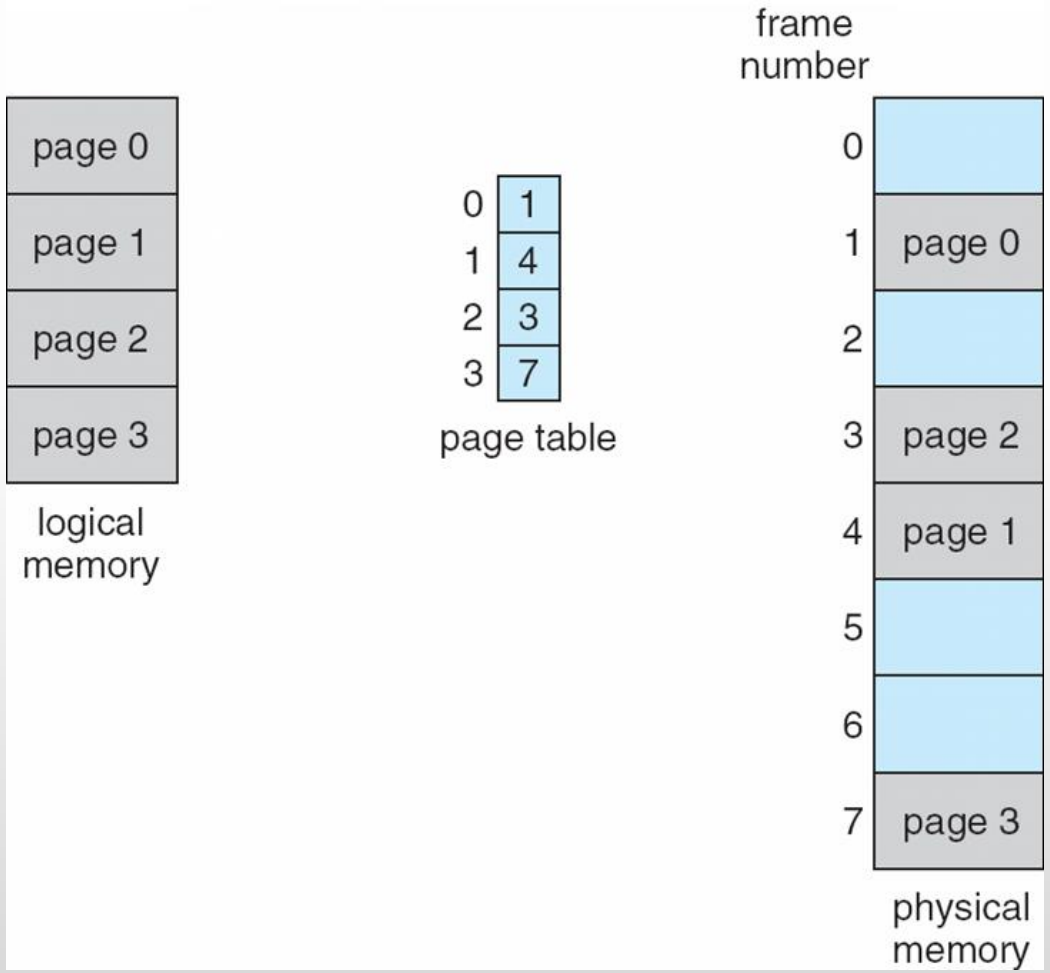
Sayfalama

- Bir sürecin fiziksel adres alanı bitişik olmayabilir
  - Harici parçalanmayı önler
  - Değişken boyutlu bellek parçaları probleminden kaçınır
- Fiziksel belleği **çerçeve** adı verilen sabit boyutlu bloklara ayırır
- Boyut, 2'nin kuvveti olacak şekilde, 512 bayt ile 16 MB arasındadır
- Mantıksal belleği, **sayfa** adı verilen aynı boyuttaki bloklara ayırır
  - Tüm boş çerçeveleri takip eder
  - N sayfa boyutunda bir programı için N adet boş çerçeve gerekir
- Mantıksal adresleri fiziksel adreslere çevirmek için **sayfa tablosu**
- Dahili parçalanma var

Sayfalama Donanımı



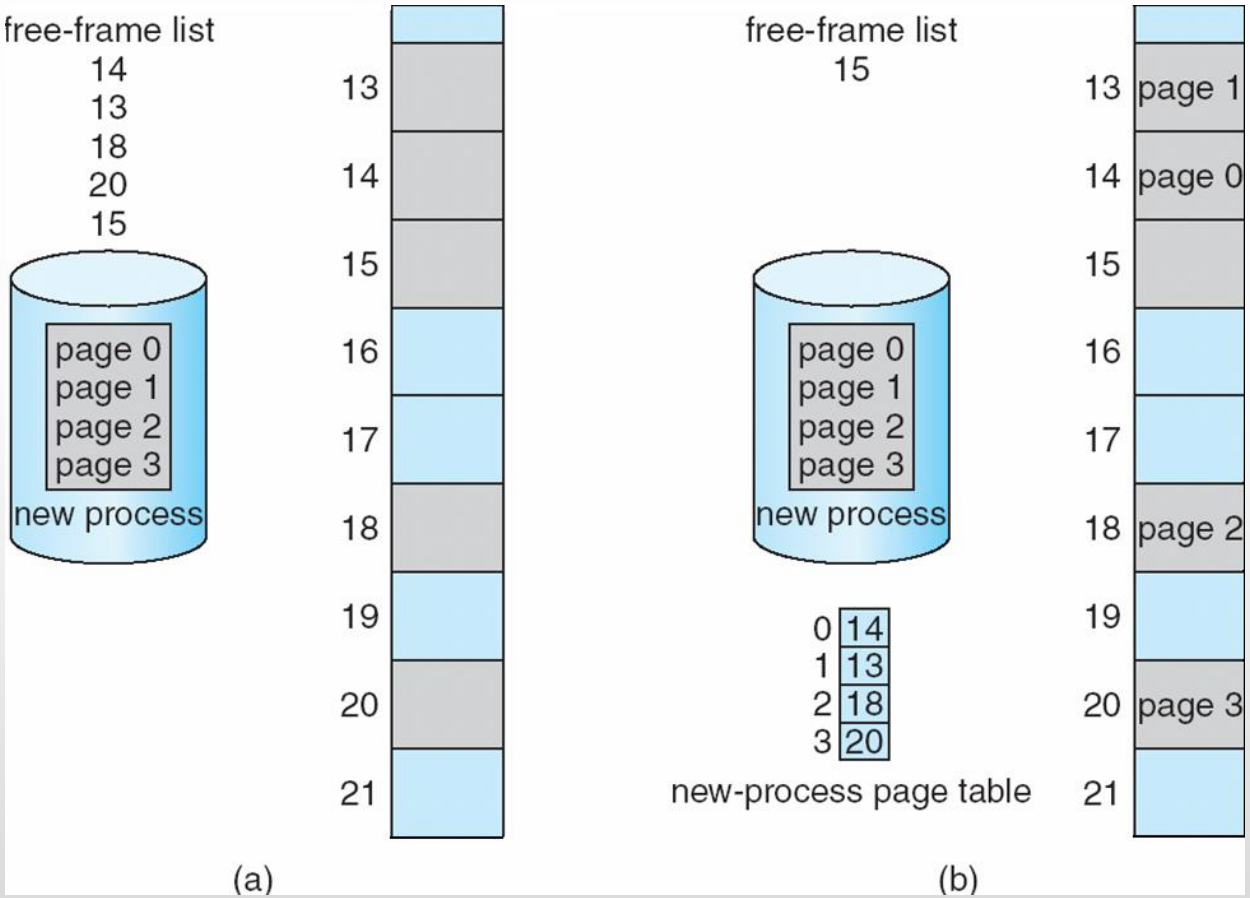
Mantıksal ve Fiziksel Bellek



Sayfalama – Dahili Parçalanma

- Sayfa boyutu = 2.048 bayt
- Süreç boyutu = 72.766 bayt
- 35 sayfa + 1.086 bayt
- 2.048 - 1.086 = 962 bayt dahili parçalanma
- En kötü durum parçalanması = 1 çerçeve – 1 bayt
- Ortalama parçalanma = 1 / 2 çerçeve boyutu
- Küçük çerçeve boyutları arzu edilir mi?
  - Her sayfa tablosu girdisi için bellek gerekir
  - Zamanla büyüyen sayfa boyutları

Boş Çerçeveler



Sayfa Tablosu

- Sanal adres={sayfa numarası, ofset}
- Sayfa çerçeve numarasını bulmak için sayfa tablosuna dizine eklemek için kullanılan sayfa numarası
- Mevcut/yok biti 1'e ayarlıysa, sayfa çerçeve numarası ofsetin önüne eklenir ve bellek veri yolunda gönderilen fiziksel adres oluşturulur.

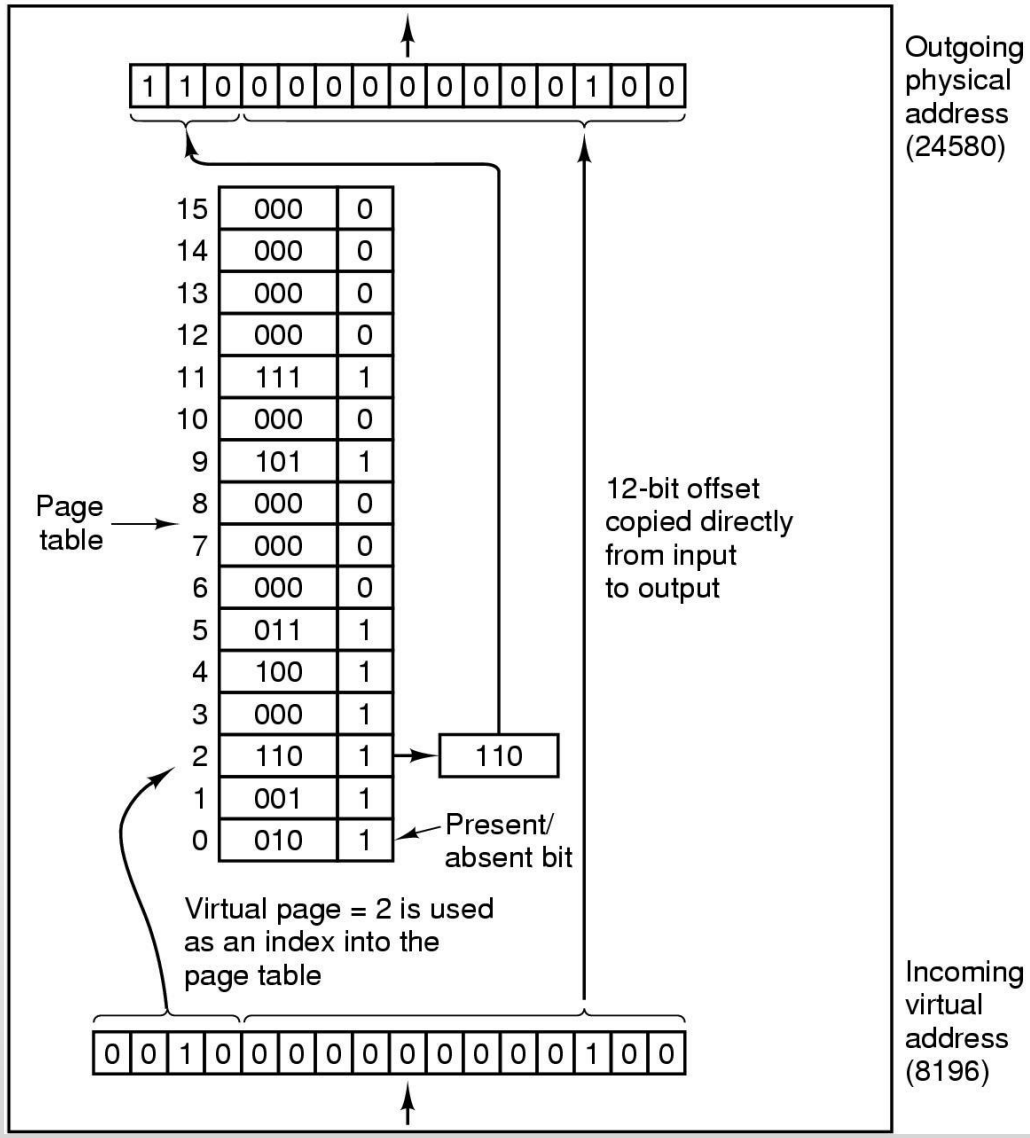
Page #	Frame #

Hatalı Sayfa İşlemi

- Mevcut/yok biti, sayfanın bellekte olup olmadığını söyler
- Adres bellekte yoksa ne olur?
- İşletim sistemine tuzak (trap)
  - İşletim sistemi diske yazmak için sayfayı seçer
  - (Gerekli) adresi olan sayfayı belleğe getirir
  - Talimatı yeniden başlatır

Bellek Yönetim Birimi

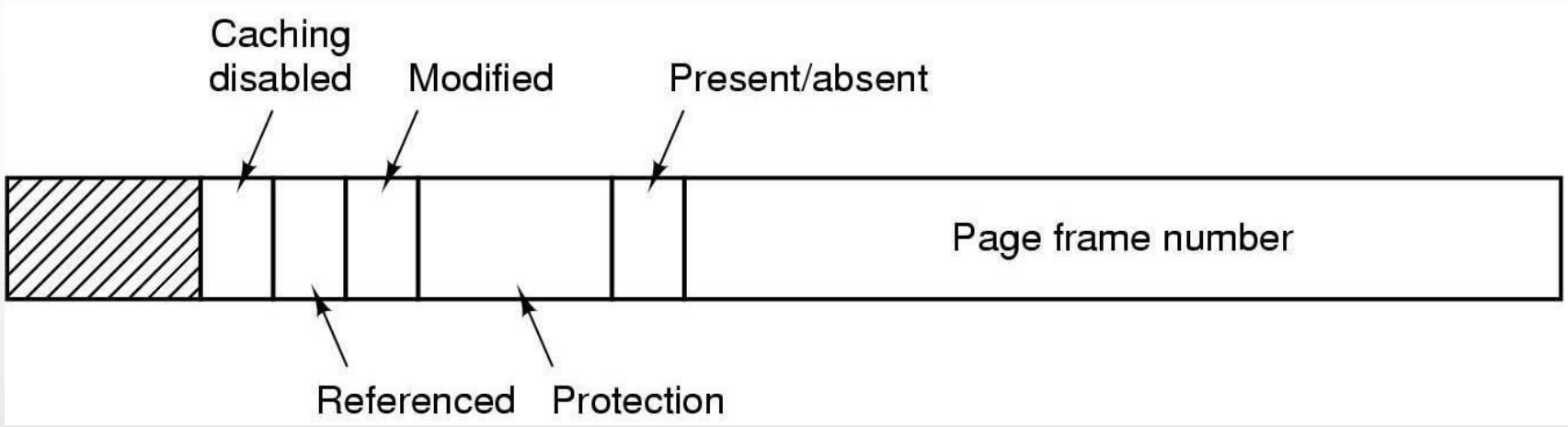
- 16 adet 4 KB sayfalı MMU'nun dahili çalışması



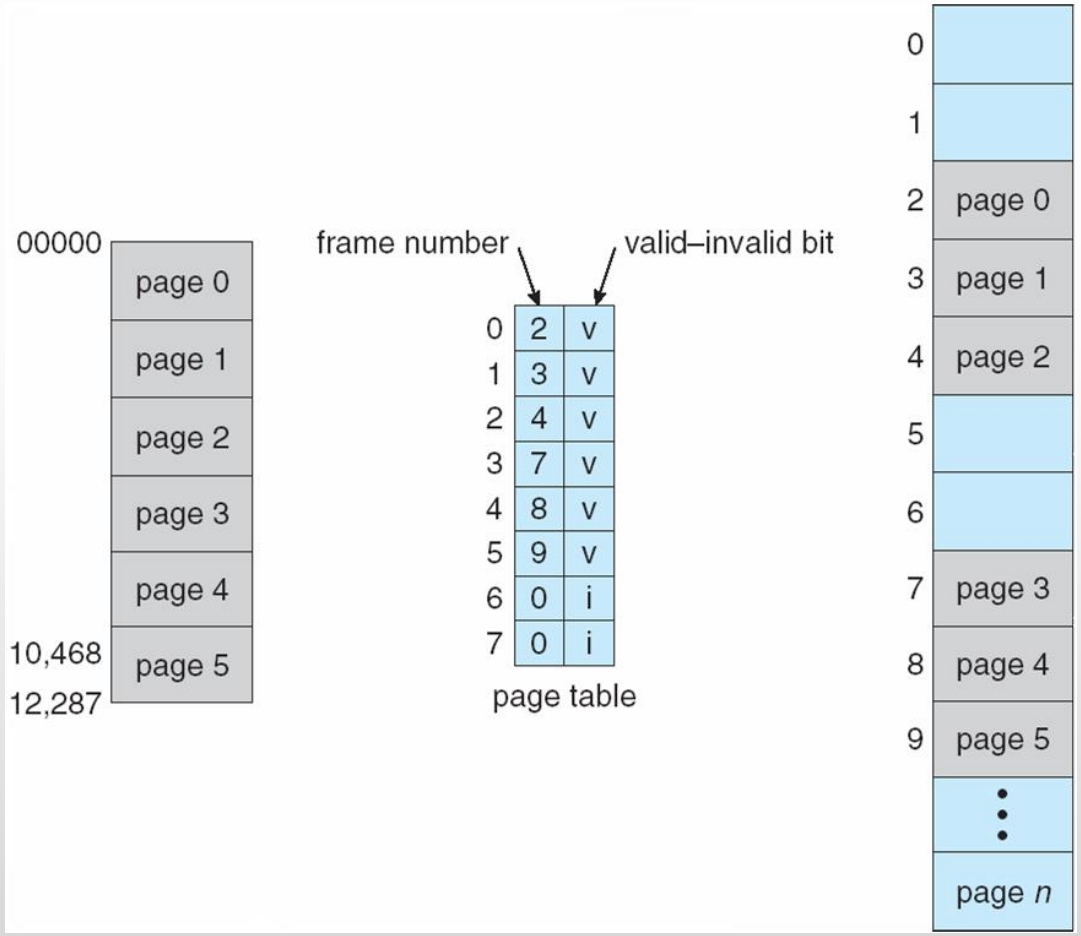
Sanal Adres Eşleme

- Sanal adres, sanal sayfa numarası ve ofset
- 16 bit adres: 4 KB sayfa boyutu; 16 sayfa
- Sanal sayfa numarası: sayfa tablosundaki indis (index)
- Sayfa tablosunun amacı
  - Sanal sayfaları sayfa çerçevelerine eşleme

Sayfa Tablosu Elemanı Yapısı



Sayfa Tablosu Elemanı Yapısı



Sayfa Tablosu Yapısı

- Koruma
  - Ne tür erişimlere izin verilir? Okuma-yazma
- Değiştirilmiş
  - Bir sayfa yazıldığında (kirli)
- Erişilen
  - Bir sayfa referans verildiğinde
- Önbellek devre dışı bırakma
  - Veri tutarsızlığı

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 tablosunu yazmaçta tutmak?
  - Süreç koşarken bellek erişimi gerekmez
  - Karşılanmayacak derecede pahalı
- Sayfa tablosunu tamamen bellekte tutmak?
  - Her sürecin kendi sayfa tablosu vardır
  - Sayfa tablosu bellekte tutulur
  - Mantıksal bir bellek erişimi gerçekleştirmek için kaç bellek erişimi gerekir?

Sayfalamayı Hızlandırma

- Etkili bellek erişim süresi, her veri/komut erişimi için gereken süre
  - İki kez bellek erişim süresi; performansı yarı yarıya azaltır
  - Sayfa tablosuna eriş → verilere/komutlara eriş

- Çözüm:
  - İlişkili yazmaçlar (associative registers) veya çeviri arama arabellekleri (translation look-aside buffers) (TLB'ler) adı verilen özel hızlı arama yapan donanım

Translation Lookaside Buffers

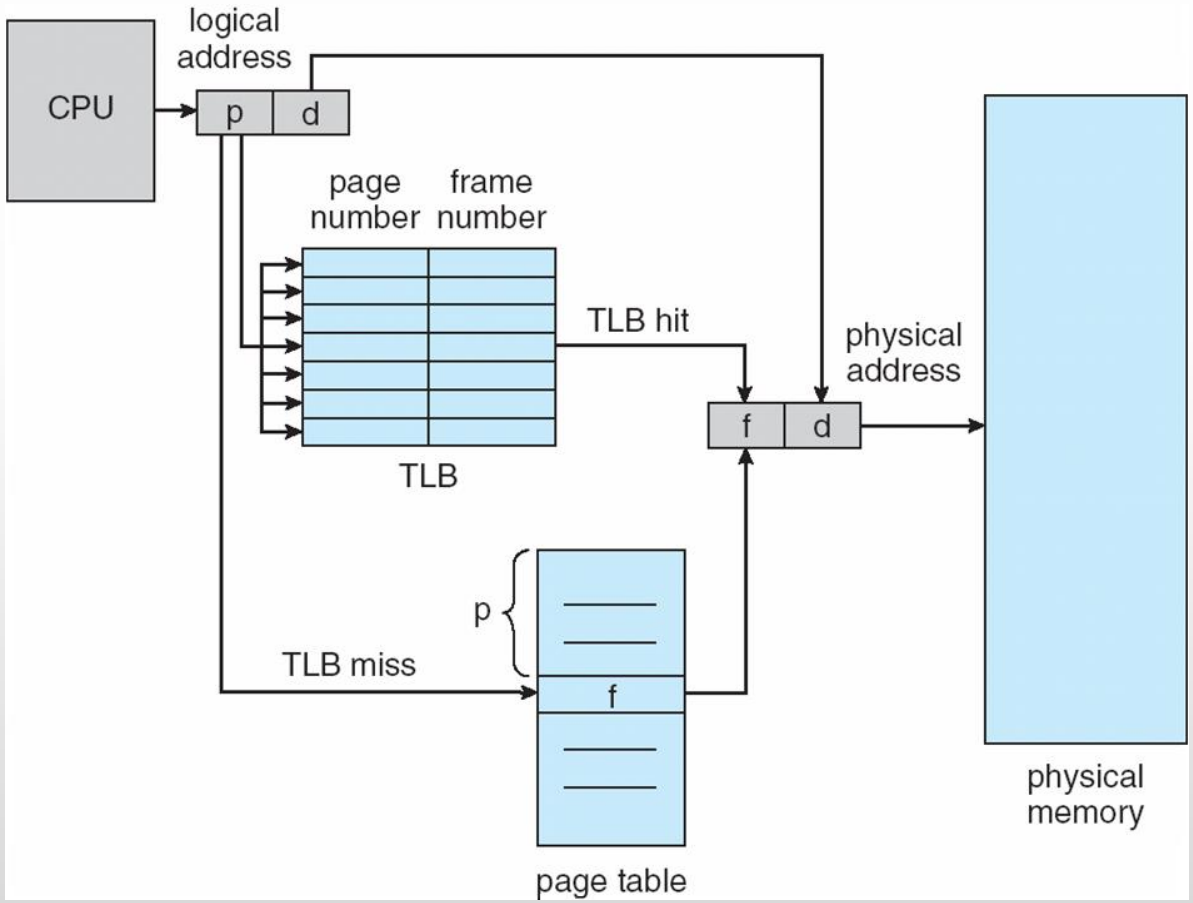
- .

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 genellikle MMU'nun içindedir ve az sayıda elemandan oluşur
- Sanal bir adres alındığında
  - MMU öncelikle sanal sayfa numarasının TLB'de olup olmadığını kontrol eder
  - TLB’de ise, sayfa tablosunu ziyaret etmeye gerek yok
  - değilse, TLB'den bir elemanı çıkarır ve yeni elemanı sayfa tablosunda bir eleman ile değiştirir.

TLB ile Sayfalama



TLB Yönetimi

- RISC makineleri TLB'yi yazılımda yönetir
- TLB hatası MMU donanımı yerine işletim sistemi tarafından işlenir
- MMU'da daha az donanım ihtiyacı
- Yazılım, hangi sayfaların TLB'ye önceden yükleneceğini anlayabilir (örn. İstemci isteğinden sonra sunucuyu yükle)
- Sık kullanılan sayfaların önbelleğini tutar

Etkili Erişim Süresi

- İlişkili Arama = ε zaman birimi;
- Bellek çevrim (cycle) süresi = *t* zaman birimi;
- İsabet oranı = α
- Etkili Erişim Süresi
  - EES = $(t + \varepsilon)\,\alpha + (2t + \varepsilon)(1 - \alpha) = 2t + \varepsilon - t\,\alpha$
- ε(20 ns), *t*(100 ns), α 1(%80), α 2(%98) ise:
  - TLB hit: 20+100=120 ns
  - TLB miss: 20+100+100=220 ns
  - EES1 = 120*0,8 + 220 * 0,2 = 140 ns
  - EES2 = 120*0,98 + 220 * 0,02 = 122 ns

Büyük Bellek için Sayfa Tablosu

- Adres alanı: 32 bit
- Sayfa boyutu: 4 KB (4096 -> 12 bit)
- Sayfa Numaraları: 20 bit, 1 milyon sayfa (2^{32} / 2^{12})
- Sayfa eleman başına 32 bit (4 byte), sayfa tablosunu tutmak için 4 MB
- 64 bit sistem için?
- Çözüm sayfa tablosunu küçük parçalara bölmek
  - Çoklu seviye (hierarchical)
  - Karma (hashed)
  - Ters (inverted)

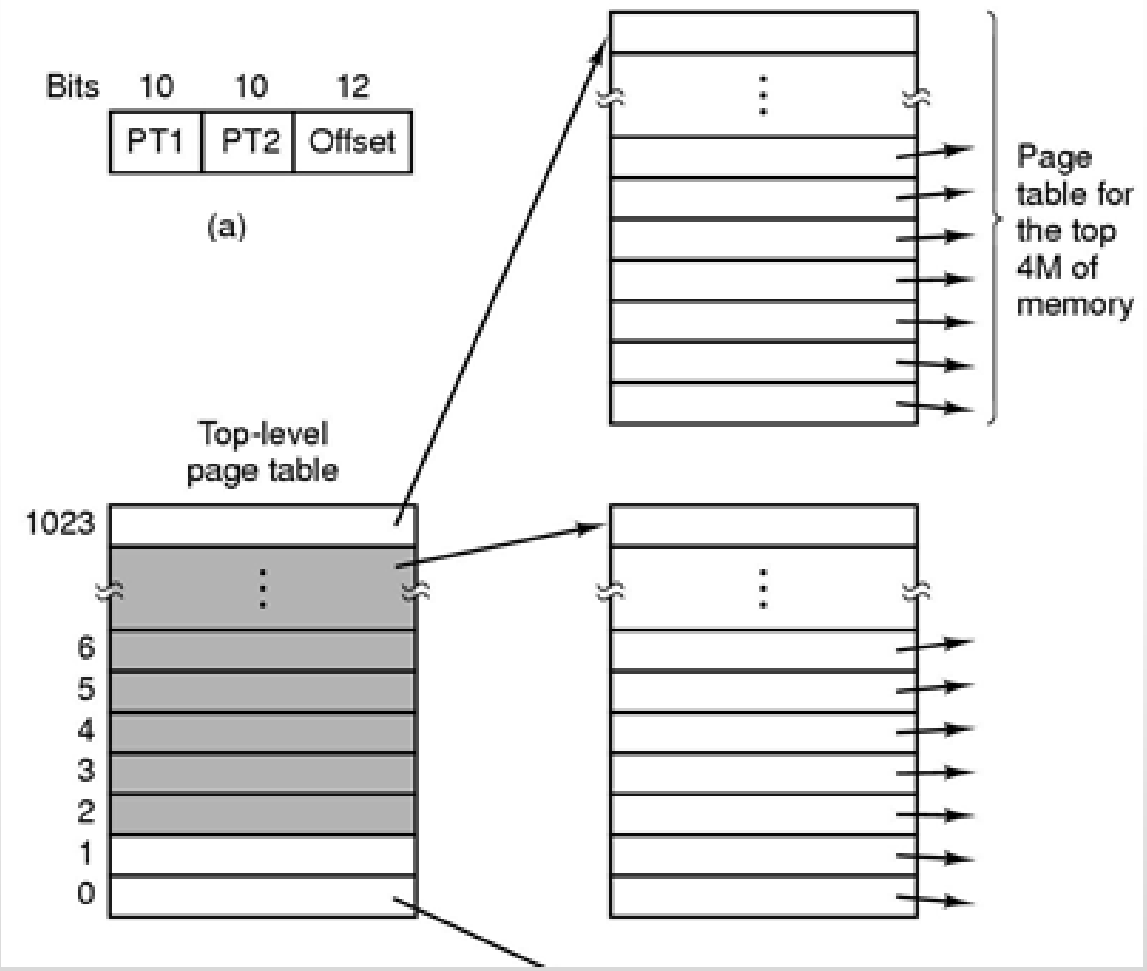
Çoklu Seviye Sayfa Tablosu

- En basit teknik iki seviyeli sayfa tablosudur
- 32 bit sanal bellek üç bölüme ayrılmıştır (PT = page table)
  - 10 bit PT1, 10 bit PT2, 12 bit ofset
- Çoklu seviye sayfa tablosu
  - Tüm sayfa tablolarını daima bellekte tutmak gerekmez
  - Sayfa tabloları da sayfalarda saklanır
  - Örnek: bir program 4G adres alanına sahiptir, koşmak için 12M'ye ihtiyaç duyar: 4M kod, 4M veri, 4M yığın için

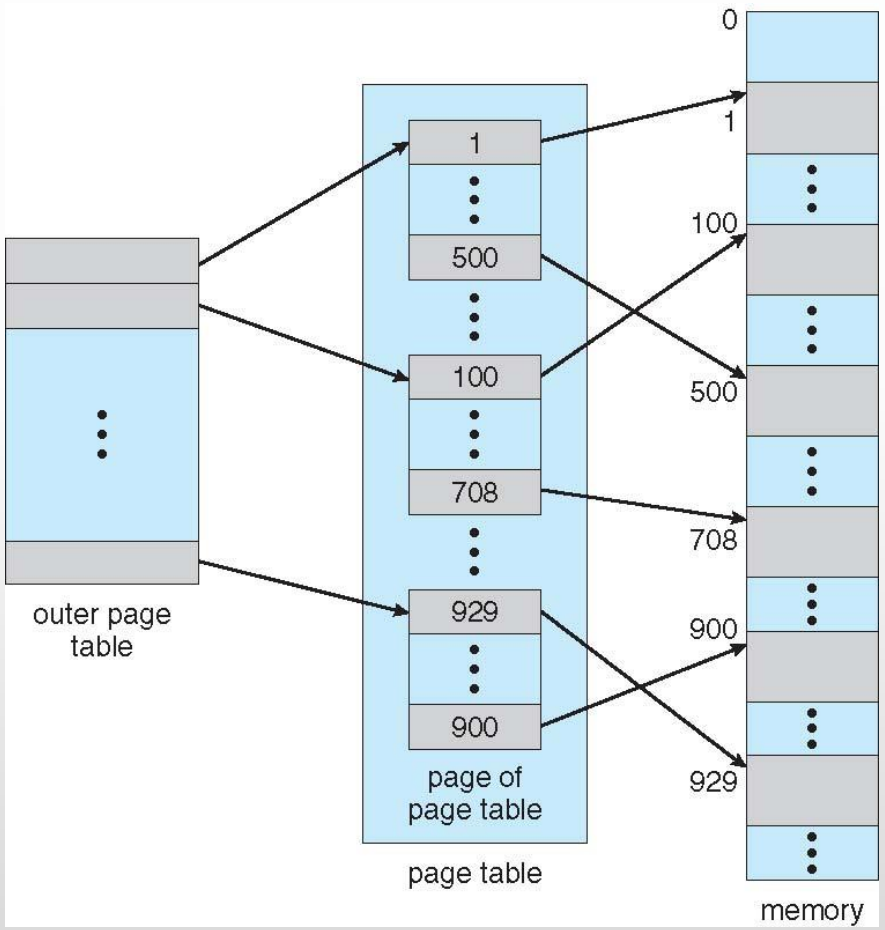
Çoklu Seviye Sayfa Tablosu

(a) İki sayfa tablo alanına sahip 32 bitlik bir adres.

(b) İki seviyeli sayfa tabloları.



Çoklu Seviye Sayfa Tablosu



Çoklu Seviye Sayfa Tablosunun Kullanımı

- Sayfa tablosunun en üst düzeyi şunları içerir:
  - Girdi 0, program metni için sayfalara işaret eder
  - Girdi 1, veri kısmı için sayfalara işaret eder
  - Girdi 1023 yığın kesimi için sayfalara işaret eder

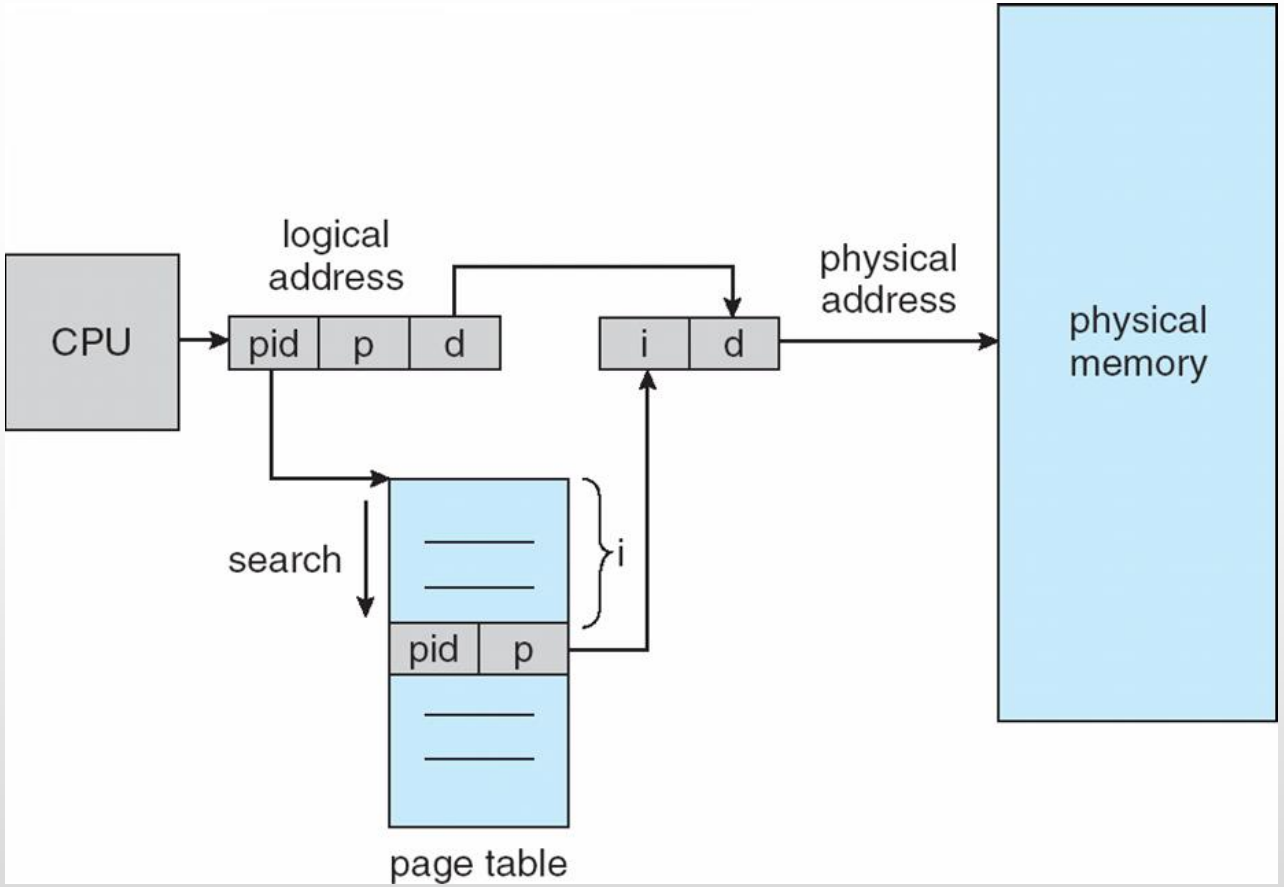
Çoklu Seviye Sayfa Tablosunun Kullanımı

- Çok düzeyli sayfa tablosu 32 bit bellek için çalışır
- 64 bit bellek için çalışmıyor
- 2^{64} bayt ve 4 KB sayfa (12 bit) => sayfa tablosunda 2^{52} girdi!
- Her giriş 8 bayt ise => sayfa tablosu için 30 milyon GB
- Başka bir çözüme ihtiyaç var

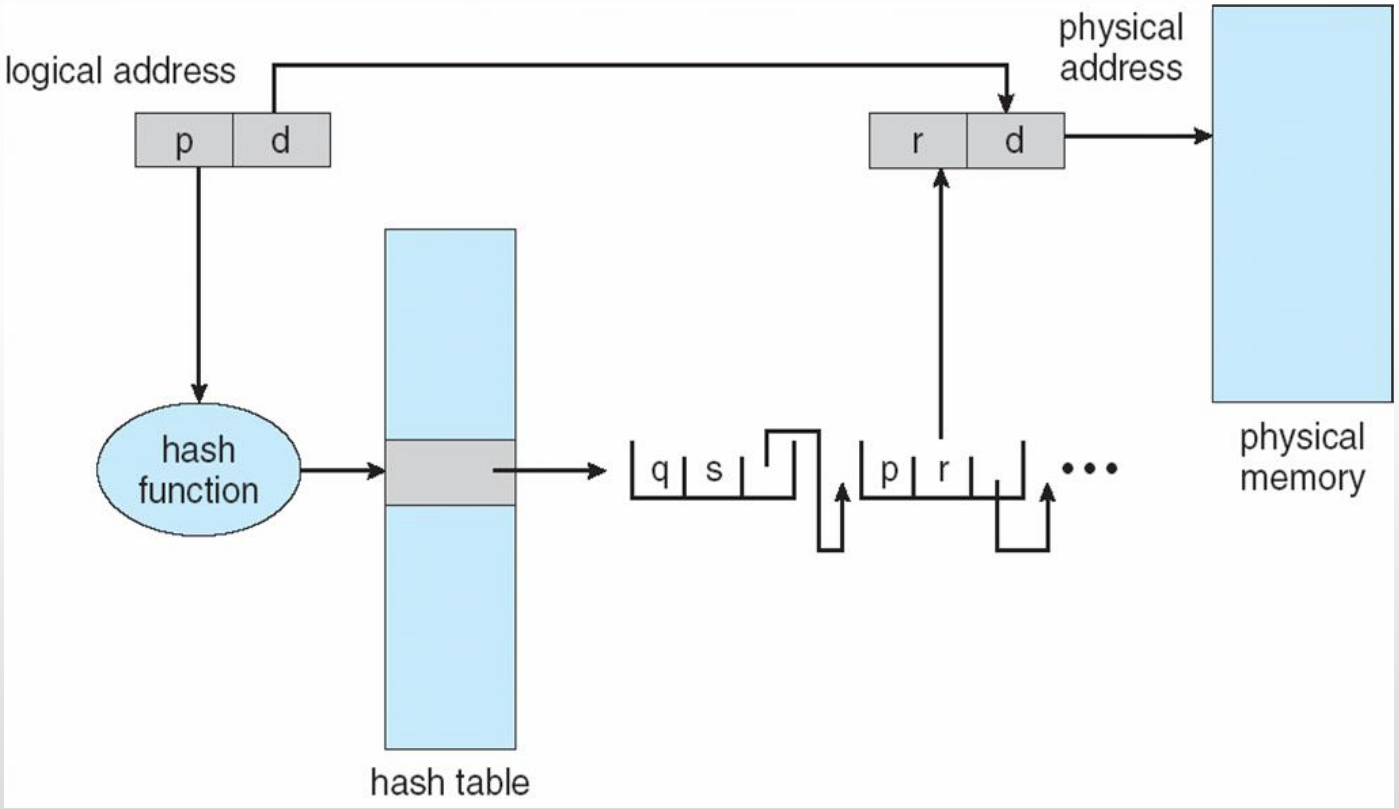
Ters Sayfa Tablosu

- Sanal adres alanları fiziksel bellekten çok daha büyük olduğunda (>32)
- "Ters" tabloda, sayfa yerine sayfa çerçevesi başına bir girdi tutulur
- Girdiler, sayfa çerçevesiyle ilişkili (süreç, sanal sayfa) takibini yapar
- Her bellek erişimi için (n,p) ilişkili sayfa çerçevelerini arar
- Arama daha zordur. Bu verimli bir şekilde nasıl yapılır?
  - Yoğun olarak kullanılan çerçeveleri TLB'de tut
  - Hash tablo kullan

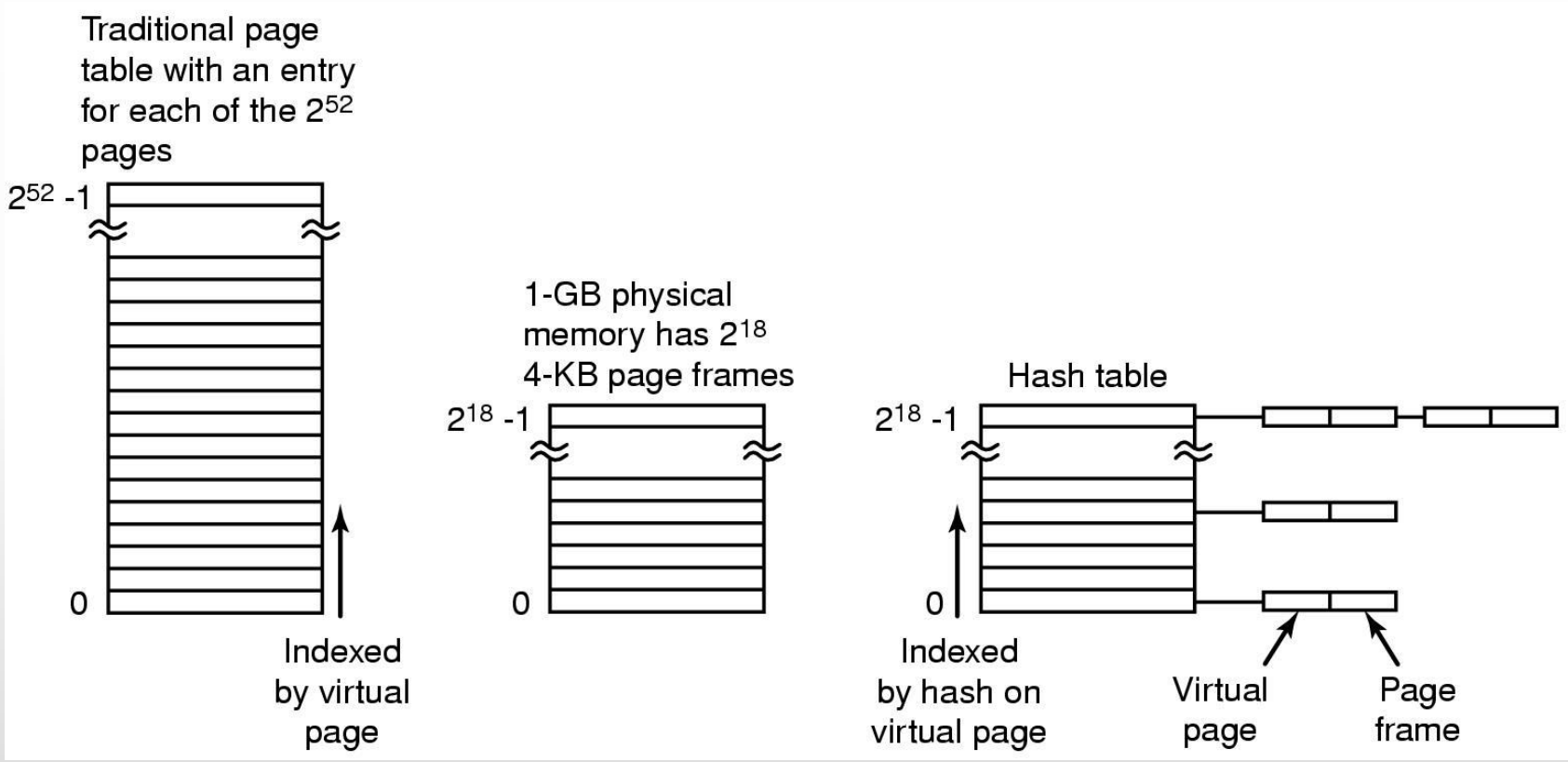
Ters Sayfa Tablosu



Hash Sayfa Tablosu

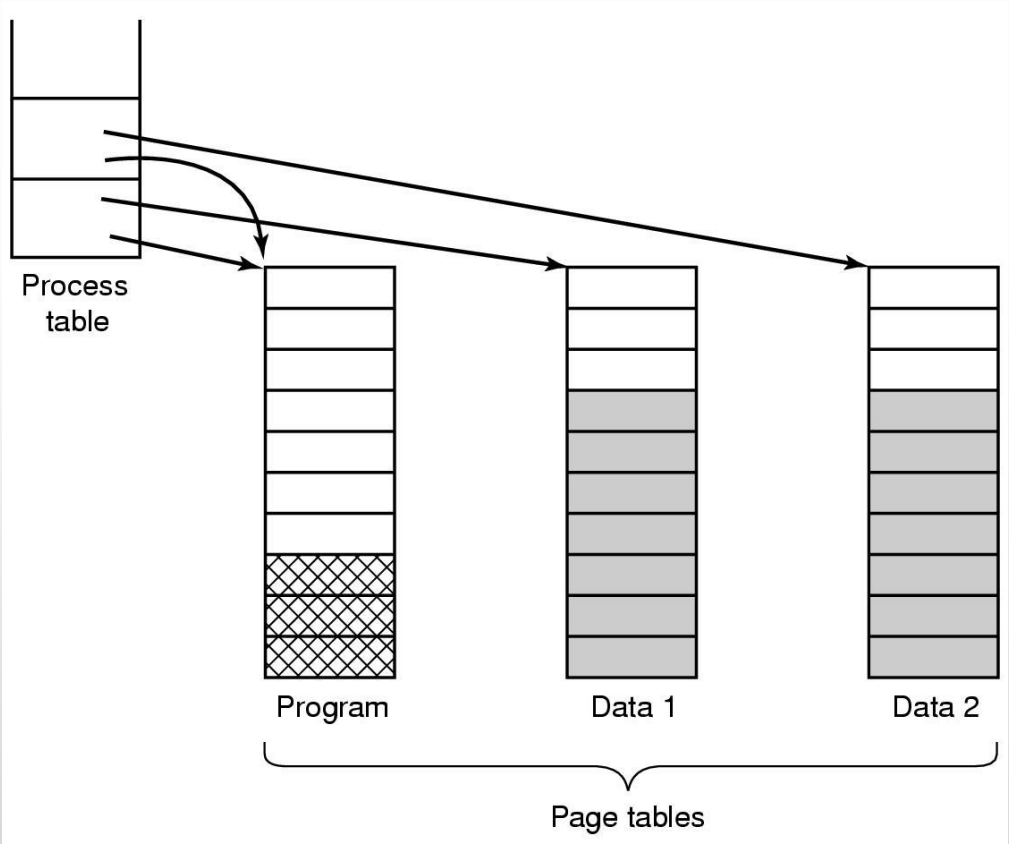


Geleneksel ve Hash Sayfa Tablosu



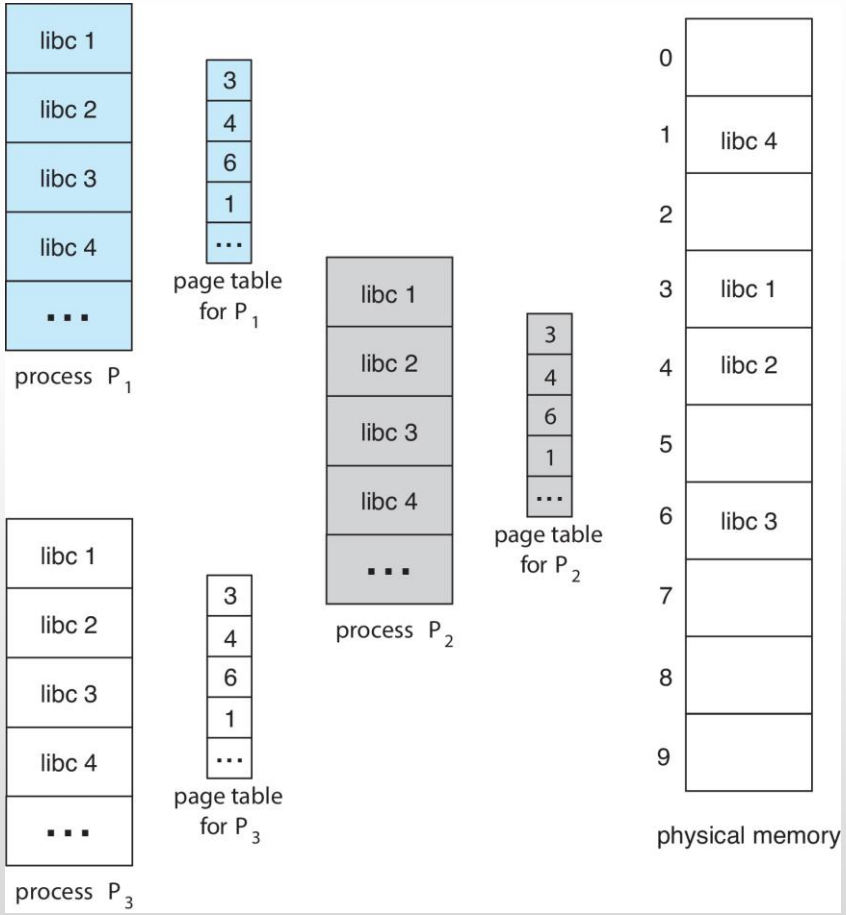
Paylaşımlı Sayfalar

- İki süreç aynı programı ve sayfa tablosunu paylaştığında



Paylaşımlı Sayfalar

- .



Paylaşımlı Sayfalar

- Süreç, sayfaları hala kullanımda olmadıklarından emin olmadan, çıktığı zaman bırakamaz
  - Paylaşılan sayfaları izlemek için özel veri yapısı kullanılır
- Sayfaya yazmalar nedeniyle veri paylaşımı sancılıdır (örn. Unix fork, ebeveyn ve çocuk süreç metin ve verileri paylaşır)
  - (Yazarken kopyala) çözümü, verileri salt okunur sayfalara eşlemektir. Yazma gerçekleşirse, her işlem kendi sayfasını alır.

Bellek Eşlemeli Dosyalar

- Süreç, bir dosyayı sanal adres alanının bir parçasına eşlemek için sistem çağrısı yapar. (memory mapped file)
- Paylaşımlı bellek (shared memory) yoluyla iletişim kurmak için kullanılabilir.
- Süreçler aynı dosyayı paylaşır.
- Okumak ve yazmak için kullanılır.

Temizleme İlkesi

- İhtiyaç duyulduğunda kurban aramak yerine, ihtiyaç duymadan önce tahliye edilecek sayfaları bulmak için bir arka plan (daemon) programı olmalı
- Daemon çoğu zaman uyur, periyodik olarak uyanır
- Eğer "çok az" çerçeve varsa, çerçeveleri atar
- Tahliye etmeden önce temiz (değiştirilmemiş) olduklarından emin olunmalı

Sanal Bellek Arayüzü

- 2 program fiziksel belleği paylaşmak isteyebilir
- Paylaşımlı bellek (shared memory) mesaj transferinin kolay yolu
  - Bellek kopyalama yaklaşımından kaçınır
- Dağıtılmış (distributed) paylaşılan bellek sayfası hata işleyicisi, sayfayı ihtiyacı olan makineye gönderen farklı makinedeki sayfayı bulur

Uygulama Sorunları

- İşletim sistemi, süreç oluşturulduğunda, yürütüldüğünde, sayfa hatası olduğunda, sonlandırıldığında sayfalamaya çok fazla dahil olur
- Özel sorun ve problemler
  - Sayfa hatası işleme
  - Talimat yedekleme
  - Bellekteki sayfaları kilitleme
  - Yedekleme deposu - sayfaların diskte yerleştirileceği yer

Sayfa Hatasını Ele Alma

1. Donanım çalışmayı çekirdeğe bırakır, program sayacını yığına kaydeder.
2. Assembler kod parçası genel yazmaçları ve geçici bilgileri kaydeder.
3. İşletim sistemi bir sayfa hatasının oluştuğunu anlar ve hangi sanal sayfaya ihtiyaç olduğunu bulur.
4. Hataya neden olan sanal adres bulunduğunda, sistem bu adresin geçerli olup olmadığını ve korumanın erişimle tutarlı olup olmadığını kontrol eder.
5. Seçilen sayfa çerçevesi diskten okunduktan sonra değiştirilmişse (dirty, modify), sayfanın diske aktarılması çizelgelenir ve bağlam anahtarlaması (context switch) gerçekleşir.
6. Sayfa çerçevesi temiz ise, işletim sistemi gerekli sayfanın bulunduğu disk adresini arar ve belleğe getirmek için bir disk işlemi planlar.

Sayfa Hatasını Ele Alma

7. Sayfa belleğe taşındığında disk kesilmesi oluşur, sayfa tabloları konumu yansıtacak şekilde güncellenir, çerçeve normal durumda olarak işaretlenir.

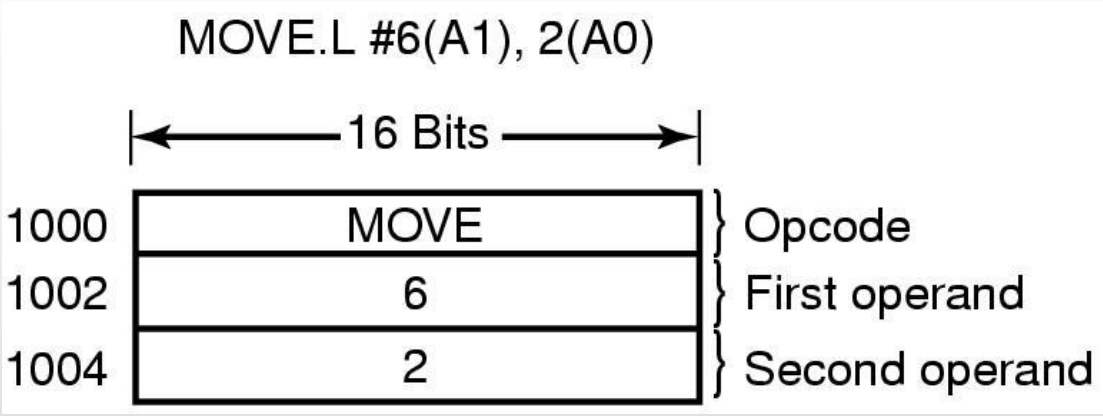
8. Hatalı kod, başladığı andaki durumuna yedeklenir ve bu komutu işaret etmek için program sayacı sıfırlanır.

9 Hataya neden olan süreç çizelgelenir, işletim sistemi onu çağıran (assembler dili) rutine geri döner.

10. Bu rutin, yazmaçları ve diğer durum bilgilerini yeniden yükler ve sanki hiçbir hata meydana gelmemiş gibi koşmaya devam etmek için kullanıcı alanına geri döner.

Sayfa Hatasına Neden Olan Bir Komut

- Talimat yeniden nereden başlatılır?
  - PC, talimatın hangi bölümünün gerçekten hatalı olduğuna bağlıdır.
- 1002'de hata verirse OS, komutun 1000'de başladığını nereden biliyor?



Komut Yedekleme

- Otomatik arttırma yazmaçları, komut yürütülmeden önce veya sonra yükler.
  - Önce yüklerse, işlemin geri alınması gerekir.
  - Sonra yüklenirse, hiç işlem yapılmaması gerekir.
- Komutun yedeklenmesi için donanım çözümü
  - Komut yürütülmeden önce mevcut komutu bir yazmaca kopyalar
- Aksi takdirde işletim sistemi bataklığın derinliklerindedir

Bellekte Sayfa Kilitleme

- Süreç, G/Ç çağrısı yapar, verileri bekler
- Beklerken askıya alınır, yeni süreç okunur, ve yeni süreç sayfa hataları alır
- Global sayfalama algoritması => gelen veriler yeni sayfanın üzerine yazılır
- Çözüm: G/Ç'de devreye giren sayfaları kilitle

Backing Store

- Sayfa takas edildiğinde diskte nereye konur?
- İki yaklaşım
  - Ayrı disk
  - Diskte ayrı bir bölüm (Üzerinde dosya sistemi olmayan)

Backing Store - Statik Bölme

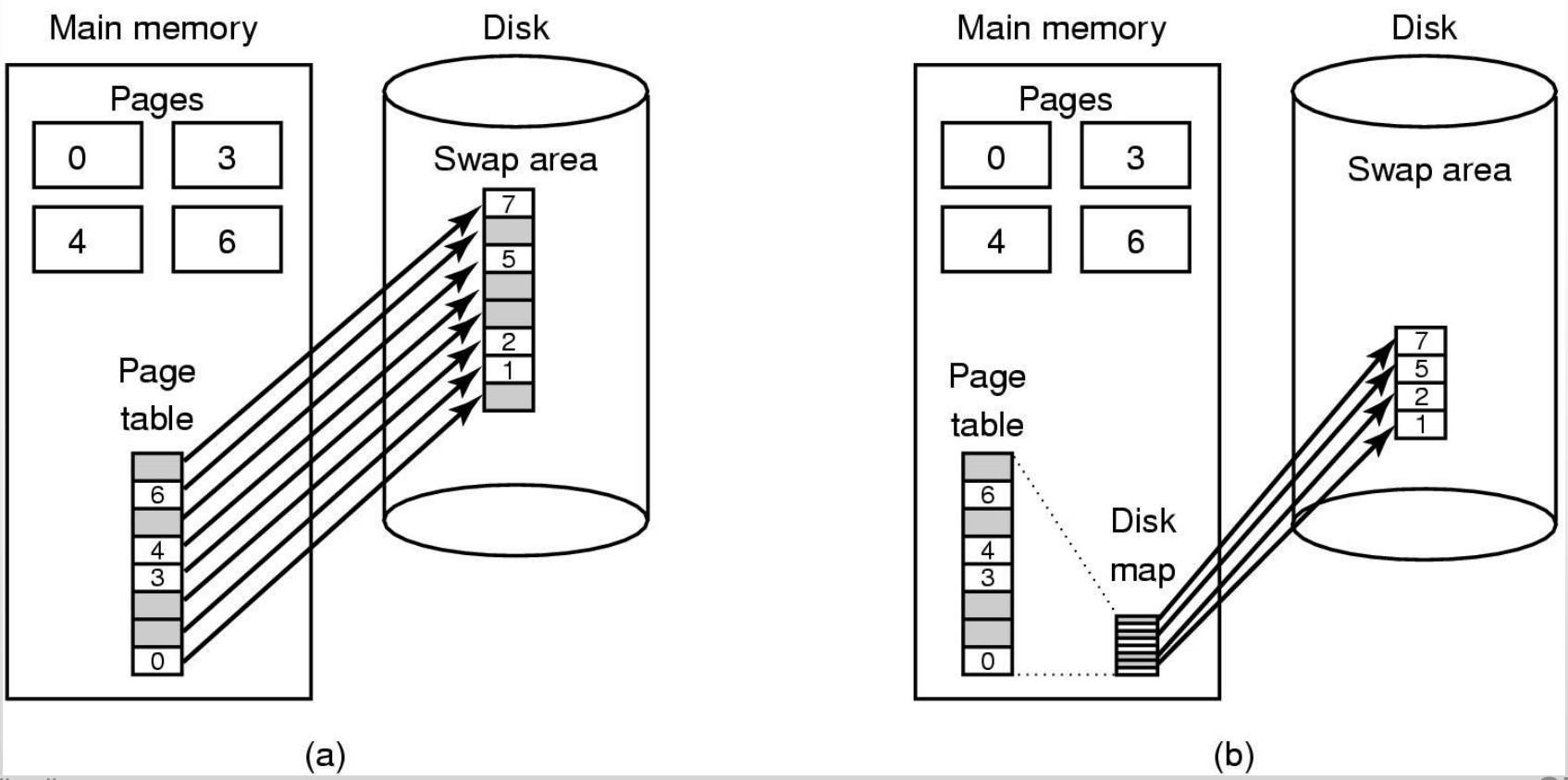
- Süreç başladığında sabit bir bölüm tahsis edilir
- Boş parçaların listesi olarak yönetim.
  - Süreç için yeterince büyük parça atanır
- Süreç tablosunda tutulan bölümün başlangıç adresi tutulur.
  - Sanal adres uzayındaki sayfa ofseti, diskteki adrese karşılık gelir.
- Veri, metin, ve yığın kesimleri için farklı alanlar atanabilir, yığın zamanla genişleyebilir.

Backing Store – Dinamik Yaklaşım

- Önceden disk alanı ayrılmaz.
- Gerektiğinde sayfaları içeri ve dışarı takas edilir.
- Bellekte disk haritasına ihtiyaç vardır

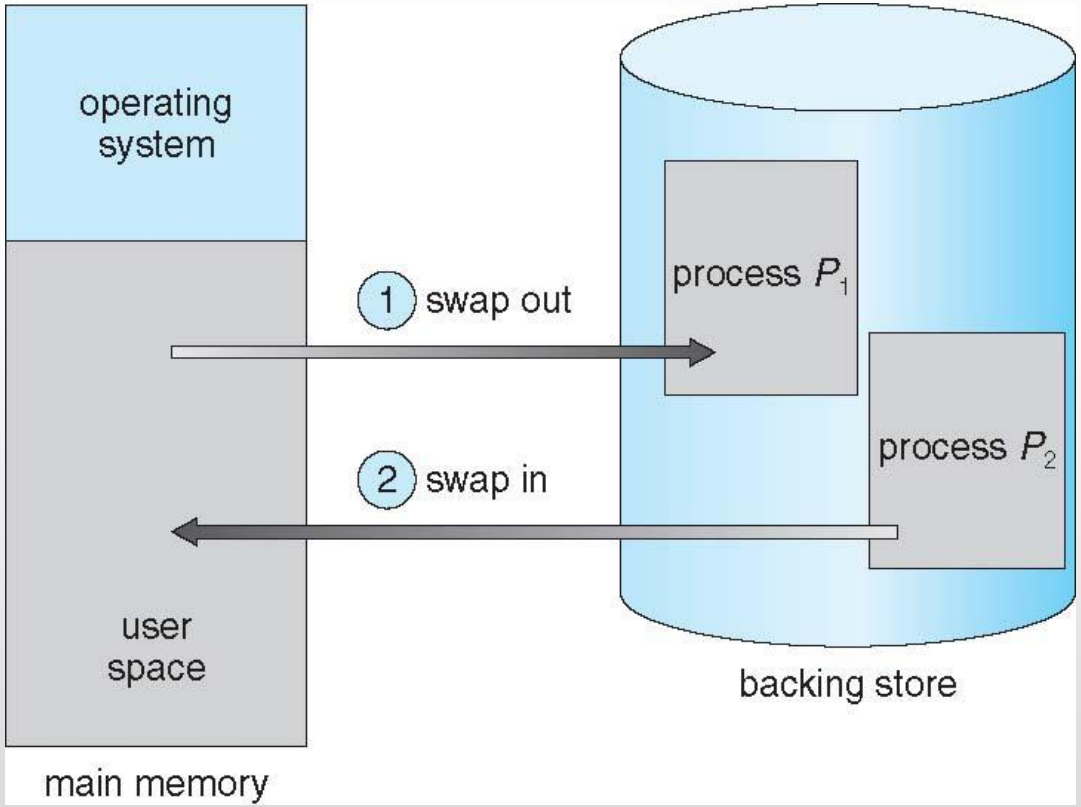
Takas Alanına Sayfalama

- (a) Statik takas alanına sayfalama (b) Sayfaları dinamik olarak yedekleme.

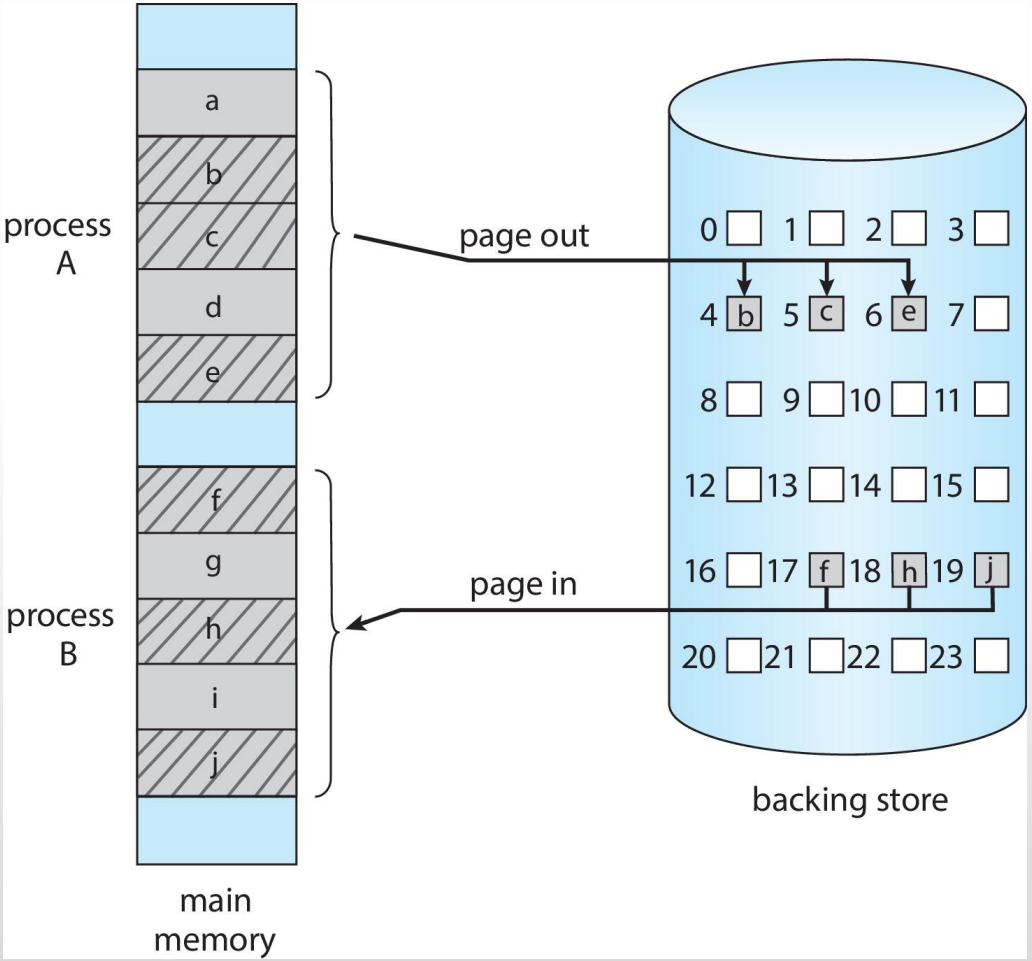


Sayfalama Olmadan Takas

- .



Sayfalama İle Takas

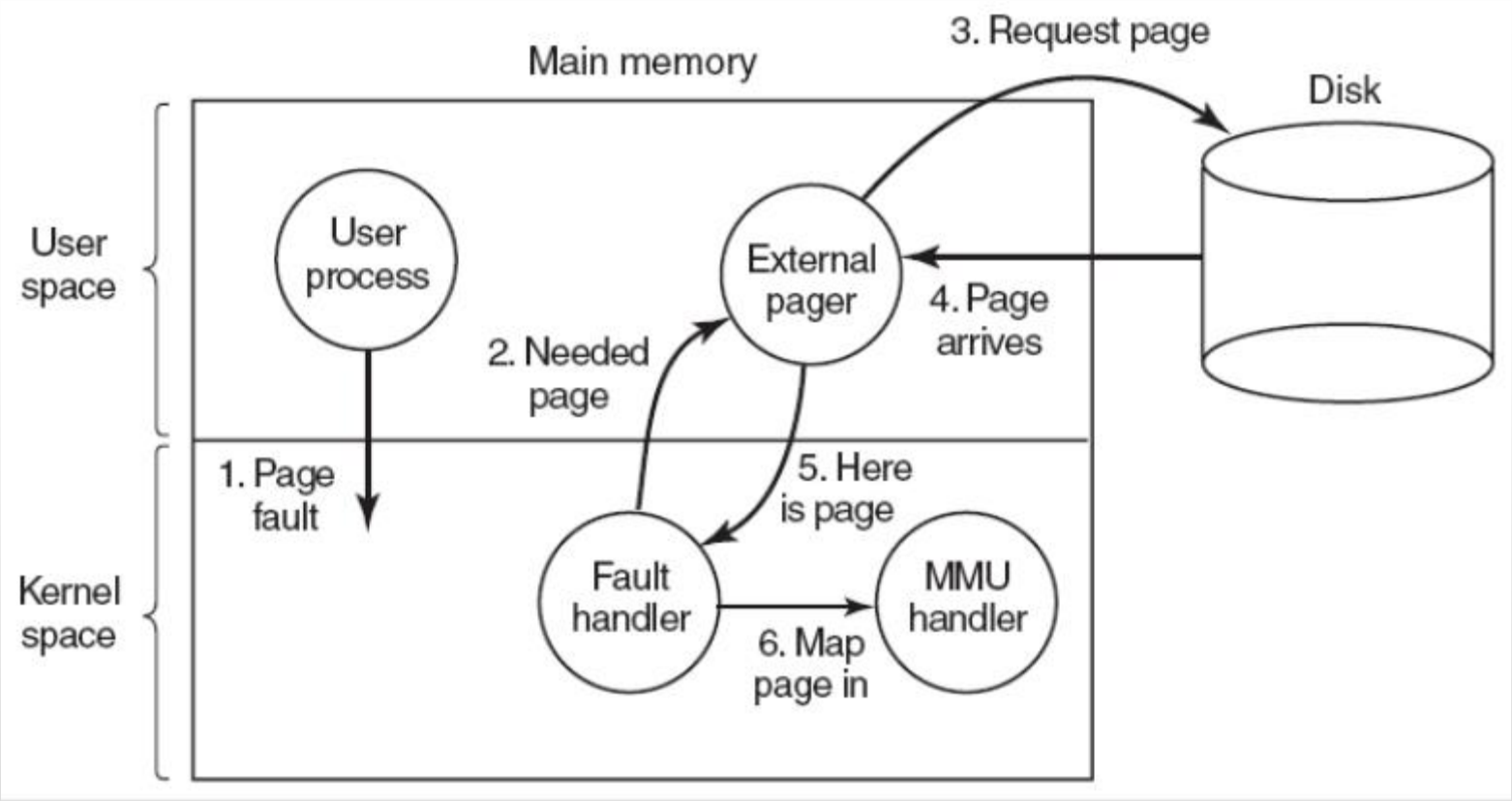


İlke ve Mekanizma Ayrımı (policy, mechanism)

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

İlke ve Mekanizma Ayrımı (policy, mechanism)

- .

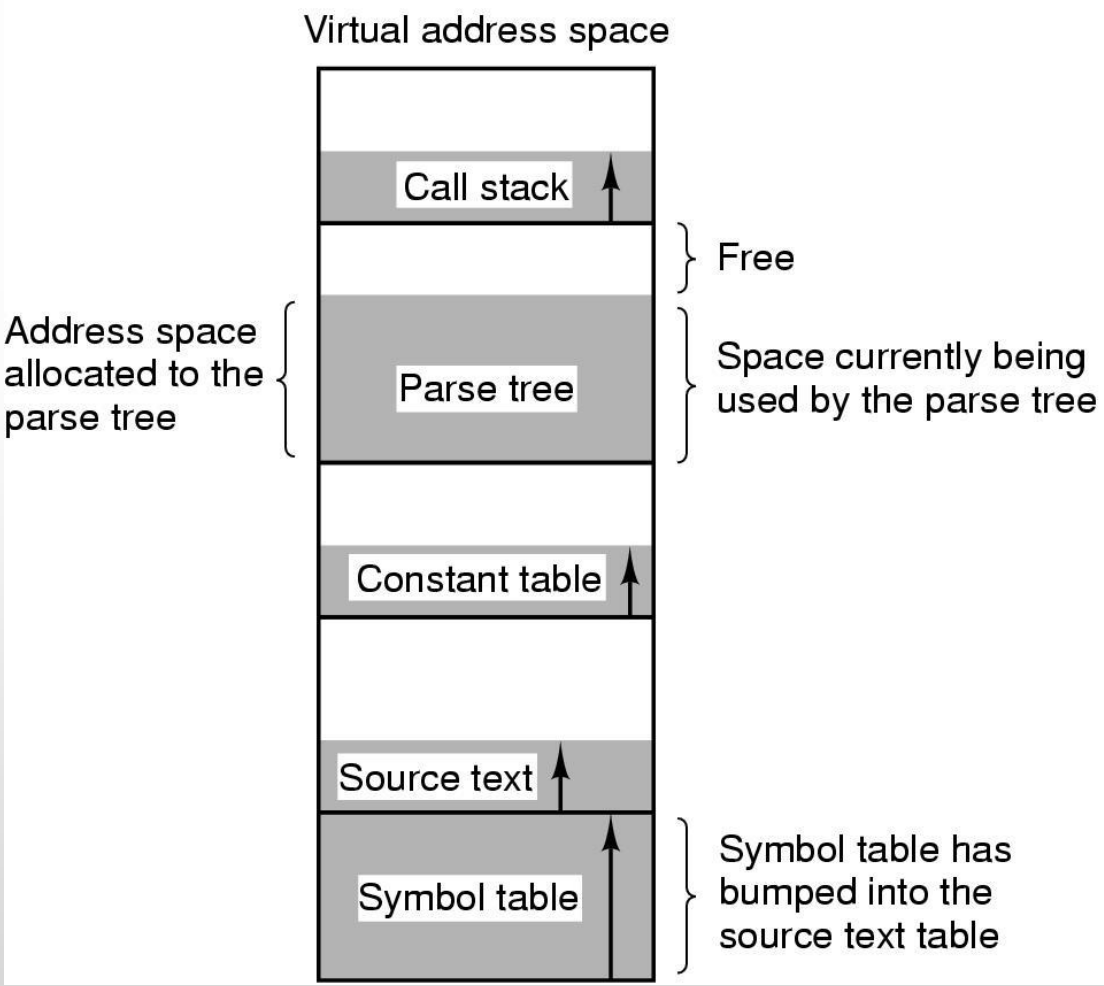


Kesimleme (segmentation)

- Bir derleyici, derleme ilerledikçe oluşturulan, aşağıdakileri içeren birçok tabloya sahiptir:
- Basılı listeleme için kaydedilen kaynak metin (toplu sistemlerde)(batch).
- Sembol tablosu – değişkenlerin adları ve nitelikleri.
- Kullanılan tamsayı, kayan noktalı sabitleri içeren tablo.
- Ayrıştırma (parse) ağacı, programın sözdizimsel (syntactic) analizi.
- Derleyici içinde prosedür çağrıları için kullanılan yığın.

Kesimleme (segmentation)

- Tek boyutlu adres uzayı.

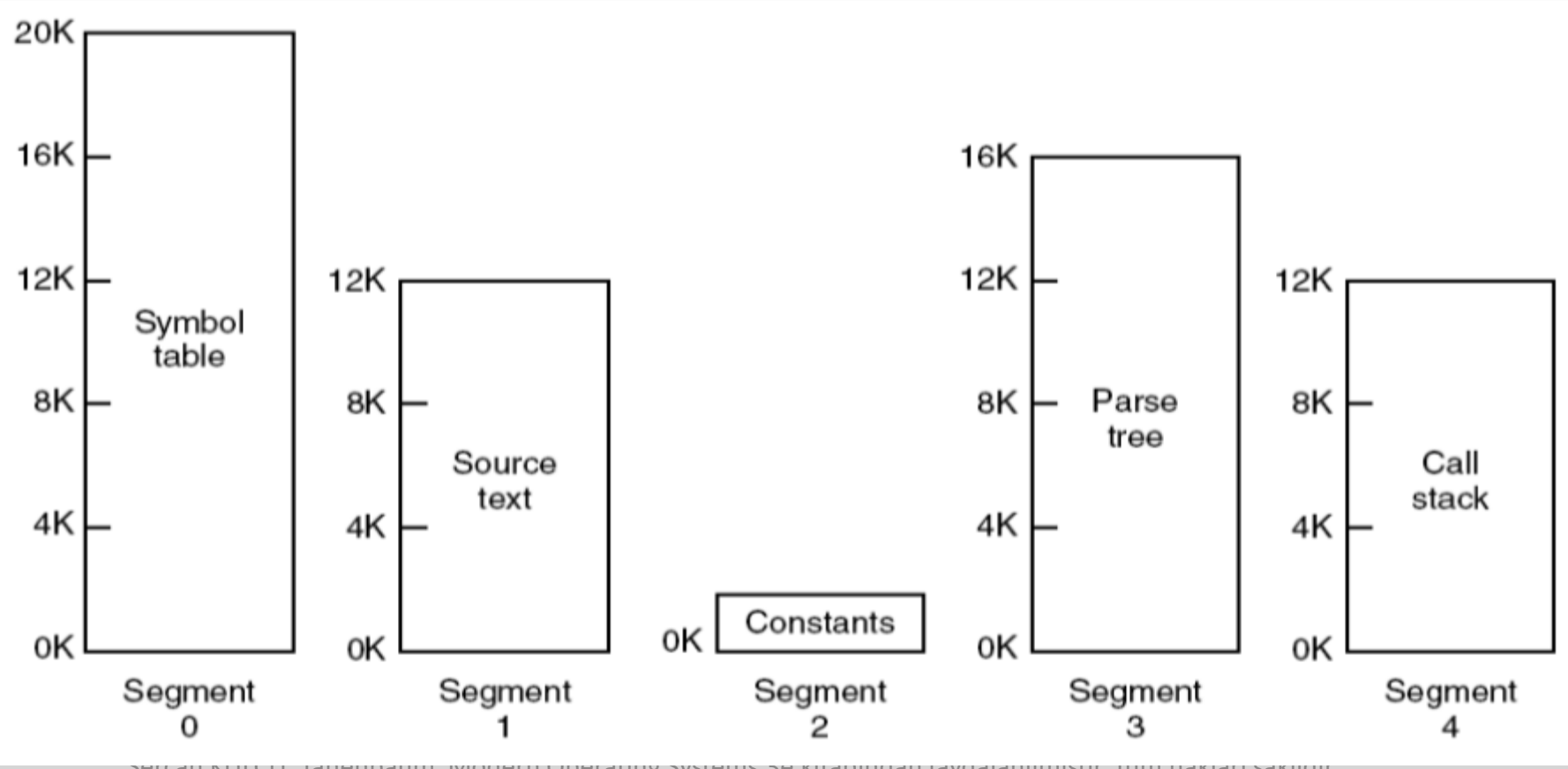


Kesimleme Avantajları

- Büyüyen ve küçülen veri yapılarının ele alınmasını basitleştirir
- Kesim n'nin adres alanı (n, yerel adres) biçimindedir, burada (n,0) başlangıç adresidir
- Kesimleri birbirinden ayrı olarak derleyebilir
- Kütüphaneyi bir kesime koyabilir ve paylaşabilir
- Farklı kesimler için farklı korumalara (r,w,x) sahip olabilir

Kesimleme (segmentation)

- Bölümlere ayrılmış bellek



Sayfalama Ve Kesimleme Karşılaştırılması

Durum	Sayfalama	Kesimleme
Programcı bu tekniğin kullanıldığının farkında olmalı mı?	Hayır	Evet
Kaç tane doğrusal adres alanı var?	1	Çok
Toplam adres alanı, fiziksel belleğin boyutunu aşabilir mi?	Evet	Evet
Prosedürler ve veriler ayırt edilebilir ve ayrı ayrı korunabilir mi?	Hayır	Evet
Boyutları değişkenlik gösteren tablolar kolayca yerleştirilebilir mi?	Hayır	Evet
Prosedürlerin kullanıcılar arasında paylaşılması kolaylaştırılmış mı?	Hayır	Evet
Bu teknik neden icat edildi?	Daha fazla fiziksel bellek almadan geniş bir doğrusal adres alanı elde etmek için	Programların ve verilerin mantıksal olarak bağımsız adres alanlarına ayrılmasına izin vermek için

SON