



Bölüm 8: Bellek Yönetimi

İşletim Sistemleri



Bellek Yönetimi

- Sonsuz bellek yok
- Bir bellek hiyerarşisi var
 - Önbellek (cache)(hızlı)
 - Anabellek (orta)
 - Disk (yavaş)
- Bellek yöneticisi, kolayca erişilebilen bir bellek soyutlaması (illusyonu) yaratmak için bu hiyerarşiyi kullanır.



Bellek Yönetimi

- Bellek (RAM) önemli ve nadir bulunan bir kaynaktır
 - Programlar, genişleyerek kendilerine sunulan belleği doldururlar
- Programcının istediği
 - Bellek gizli, sonsuz büyük, sonsuz hızlı, ve kalıcı olmalıdır...
- Gerçekte olan
 - İnsanların aklına gelen en iyi şey: bellek hiyerarşisi
 - Yazmaç, önbellek, bellek, disk, teyp
- Bellek yöneticisi
 - Belleği verimli bir şekilde yönetir
 - Boşalan bellek alanlarını takip eder, programlara bellek tahsis eder



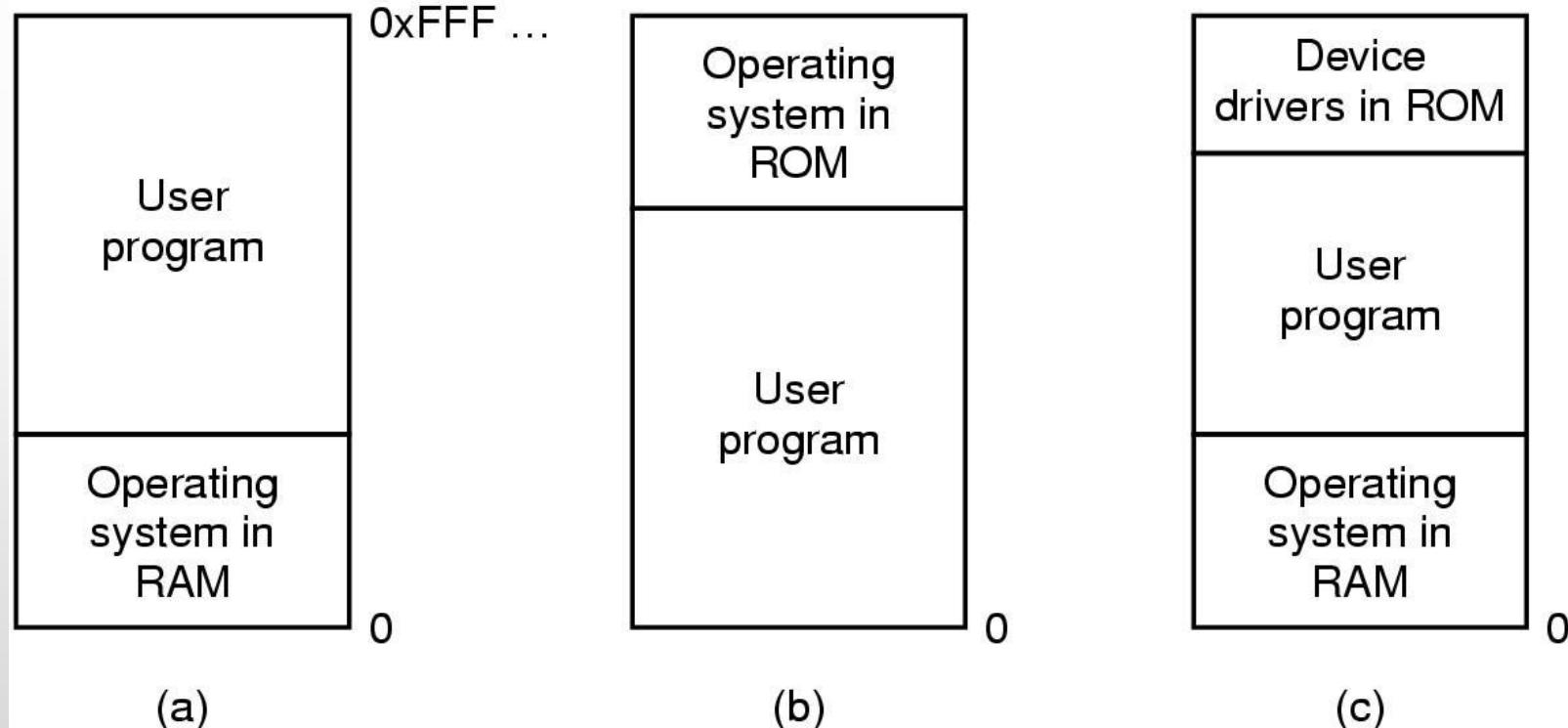
Soyutlama Yapılmadığında

- Eski anabilgisayar, mini bilgisayarlar, kişisel bilgisayarlarda bellek soyutlaması yoktu...
 - MOV REGISTER1, 1000
 - fiziksel bellek adresi 1000'in içeriğini yazmaca taşır
- Bellekte aynı anda iki program yer alamaz



Soyutlama Yapılmadığında

- İşletim sistemi ve bir süreç ile belleğin düzenlenmesi





Soyutlama Yapılmadığında

- Bellekte aynı anda yalnızca bir program olabilir.
- Kullanıcı programındaki hata işletim sistemini çöp yapabilir (a ve c)
- Bazı gömülü sistemlerde (b)
- MS-DOS (c) - ROM'da BIOS adı verilen bölüm

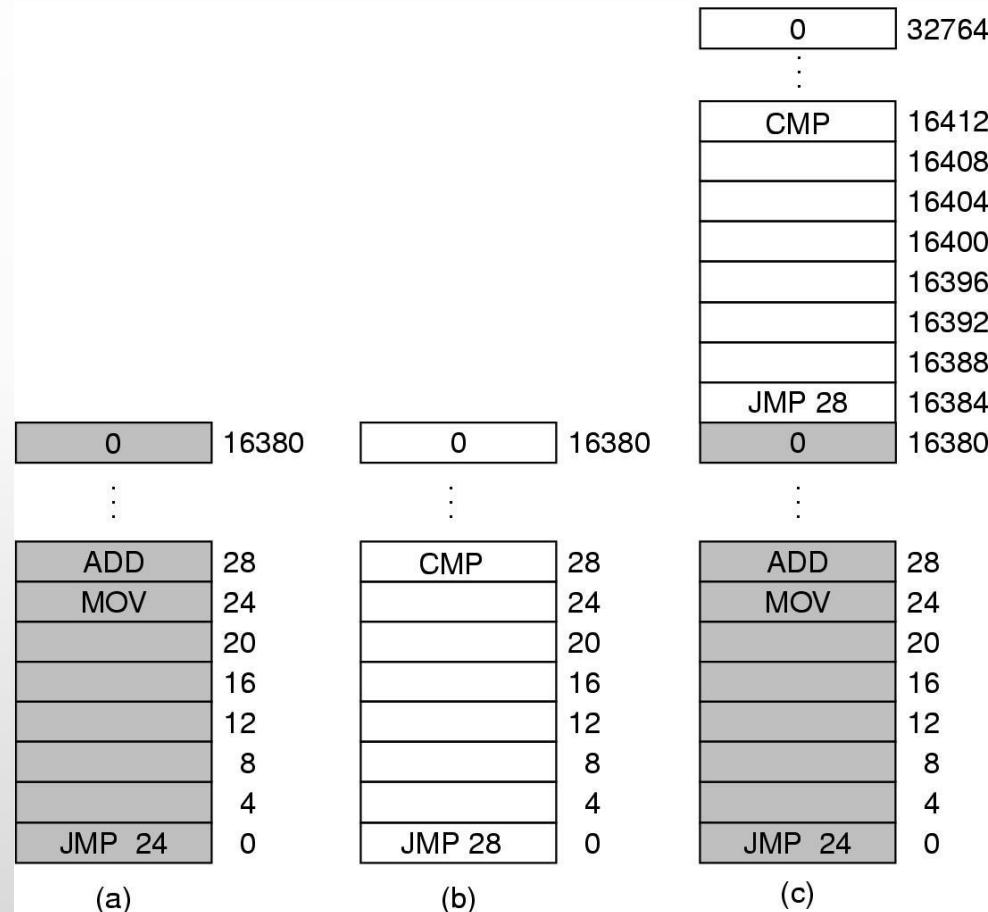


Soyutlama Yapılmadığında Problemler

- IBM 360 - 2 KB bloklara bölünmüş bellek ve her biri 4 bit koruma anahtarına sahip
- PSW 4 bitlik bir koruma anahtarına sahip
- Donanım, PSW anahtarından farklı bir koruma koduyla belleğe erişme girişimlerini yakalar.



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



Adres Uzayı

- Programın var olması için soyut bellek alanı oluşturulur
- Her programın kendi adres kümesi vardır
- Adresler her program için farklıdır
- Programın adres alanı olarak adlandırılır



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 çökmesi
 - Paralelleştirmek zor
- Çözülmlesi gereken iki problem:
 - Koruma
 - Yer değiştirme
- Adres alanı:
 - Belleği adreslemek için bir dizi bellek süreci kullanılabilir
 - Her sürecin birbirinden bağımsız kendi adres alanı vardır.
 - Nasıl?



Dinamik Yer Değiştirme

- İşlemciye iki özel yazmaç:
 - taban ve limit
- Program ardışık bir boşluğa 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



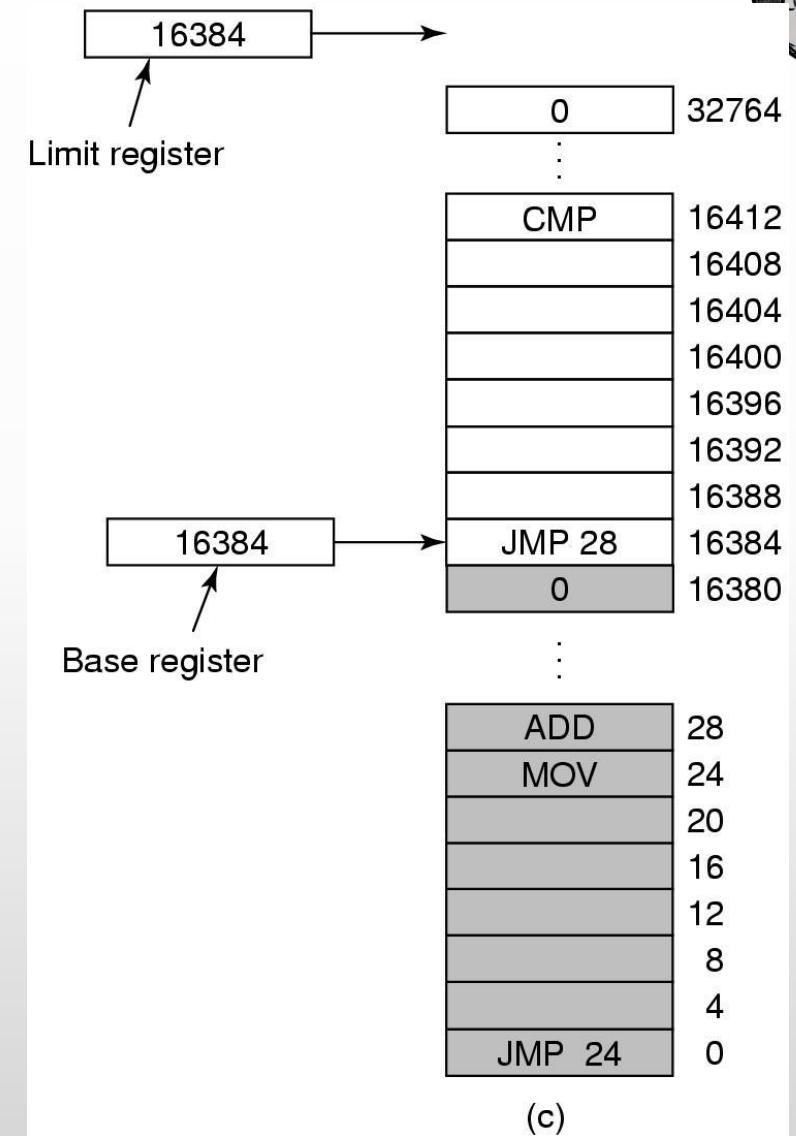
Taban ve Limit Yazmaçları

- Bir tür dinamik yer değiştirme
- Taban, programın başlangıç adresini içerir
- Limit programın uzunluğunu içerir
- Program belleğe başvurur, işlem 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





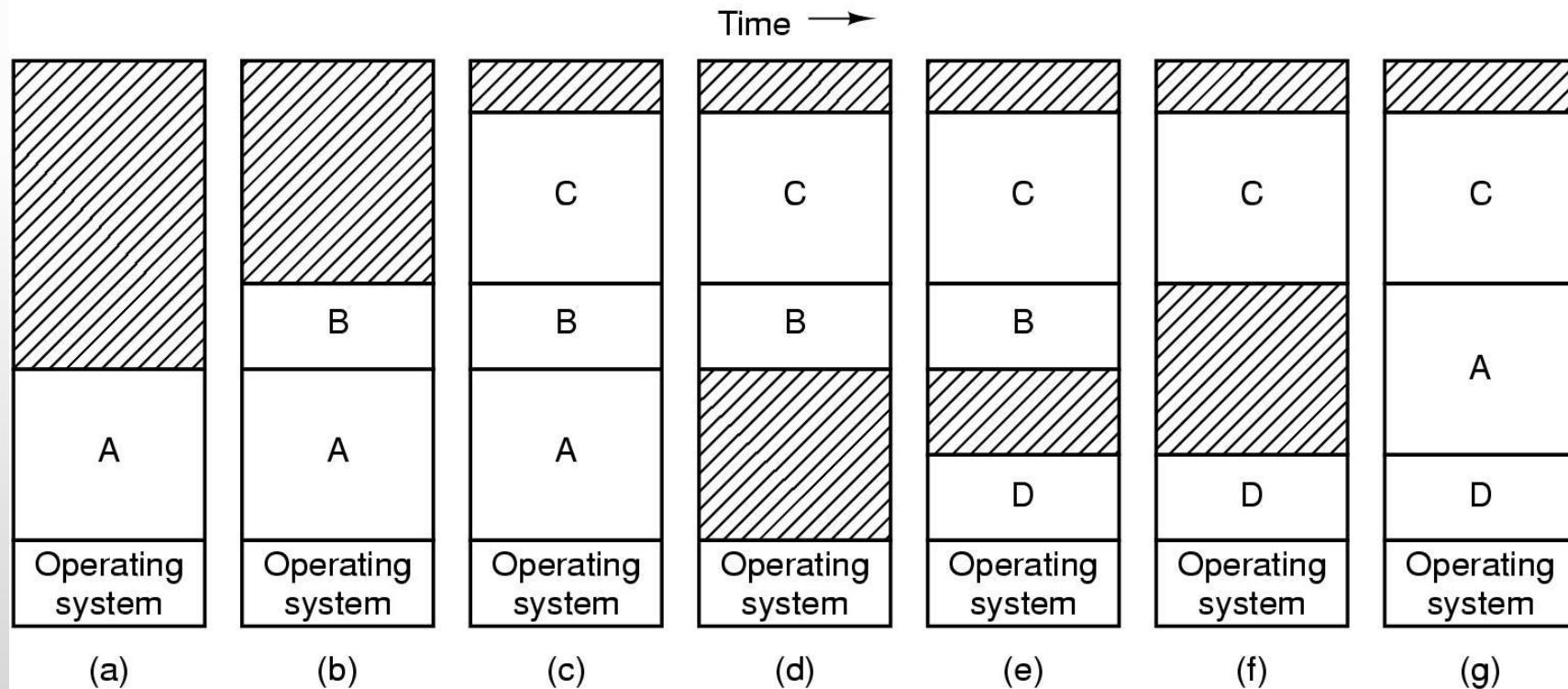
Değiş Tokuş Yapmak (swap)

- Sistemde arka planda çalışan bir çok sunucu süreci vardır
- Fiziksel bellek tüm programları tutacak kadar büyük değildir
 - Değiş tokuş
 - Programları belleğe getir, değişim yapıp götür
 - Sanal bellek
 - Programları kısmen bellekte olsalar bile çalıştır



Bellek Düzeni Değişimi

- Süreçler belleğe girip çıktıktan sonra bellek tahsisini değiştir. 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 gereklidir
 - 20 ns'de 4 byte taşıır, ardından 1 GB'ı sıkıştırmak için 5 saniye
- Bir program için ayrılan bellek miktarı ne kadar?
 - Programlar büyümeye eğilimindedir
 - Hem veri kesimi (segment) hem de yığın



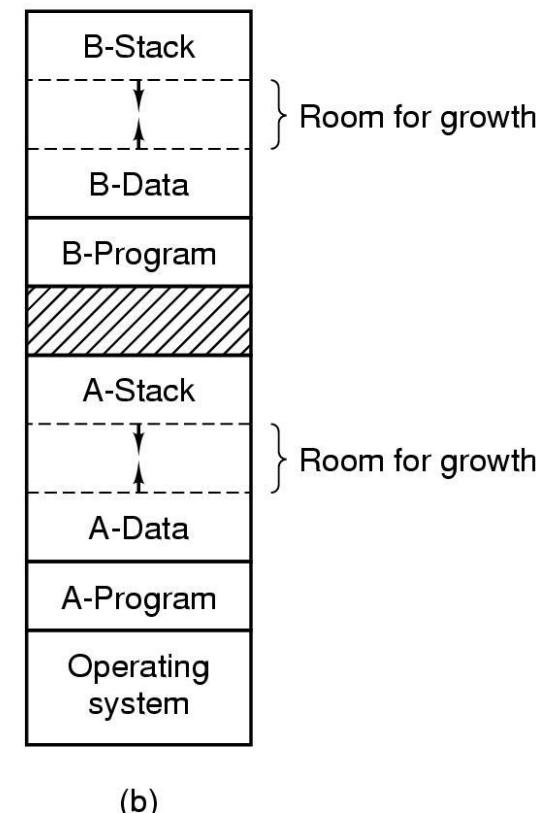
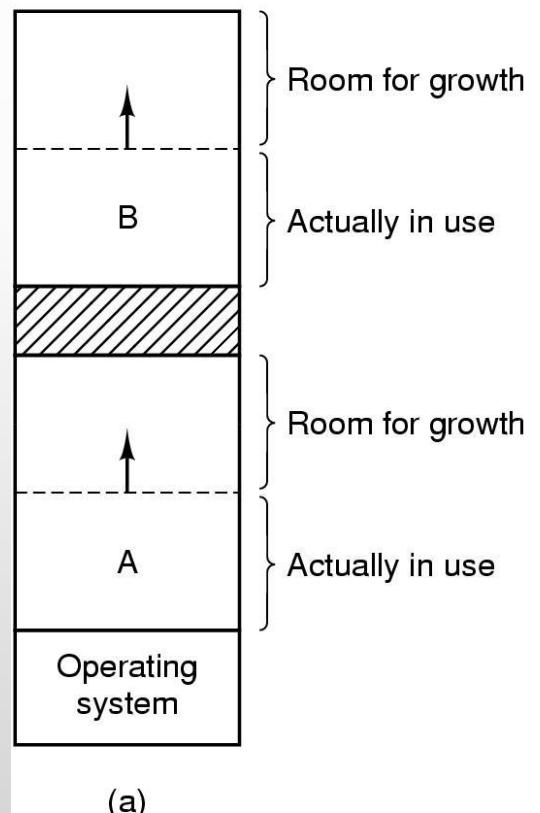
Programlar Koştukça Büyür

- Yığın (dönüş adresleri ve yerel değişkenler)
- Veri segmenti (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!



Bellekte Alan Tahsisi

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





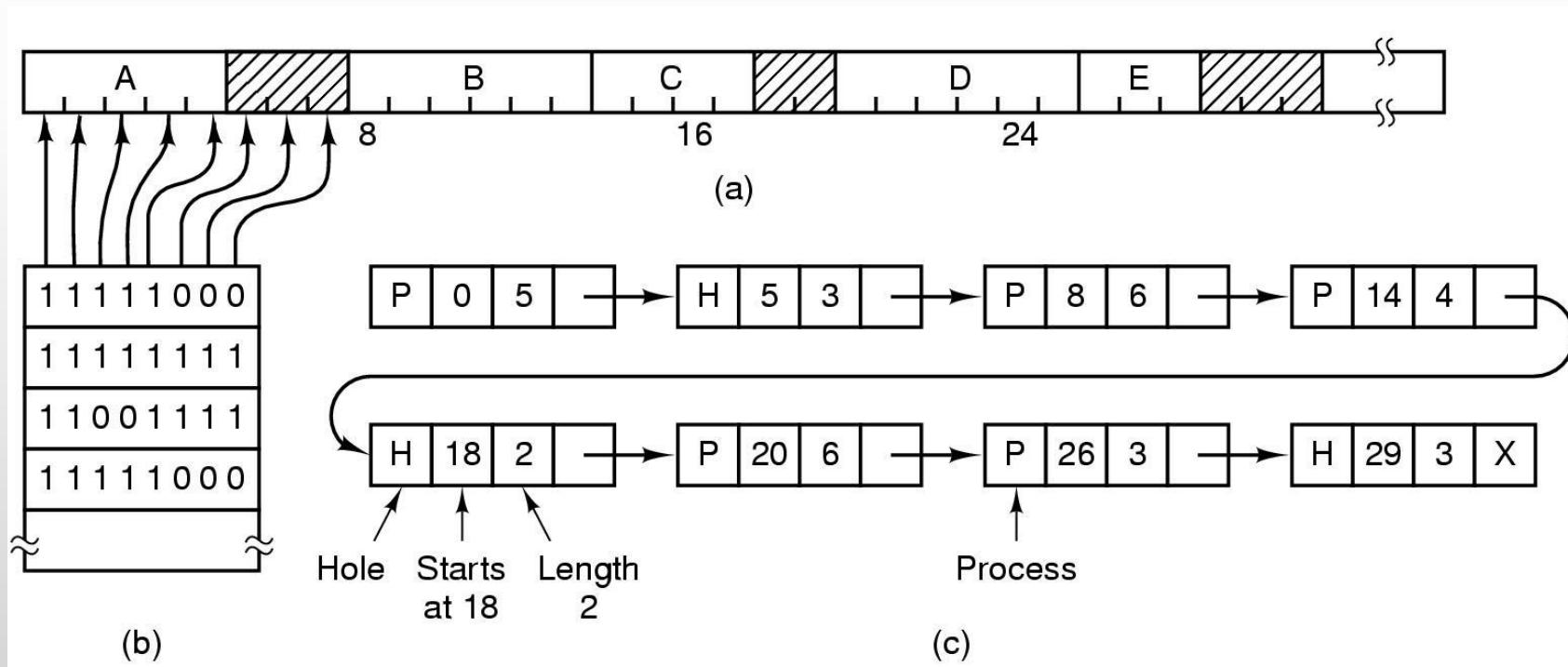
Bellekte Boş Alan Yönetimi

- Biteşlem (bitmap) ve bağlı listeler
- Biteşlem
 - Bellek, ayırma birimlerine bölünmüştür (birkaç sözcükten KB boyutuna kadar)
 - Her birime karşılık gelen, biteşlem'de bir bit var
 - 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.





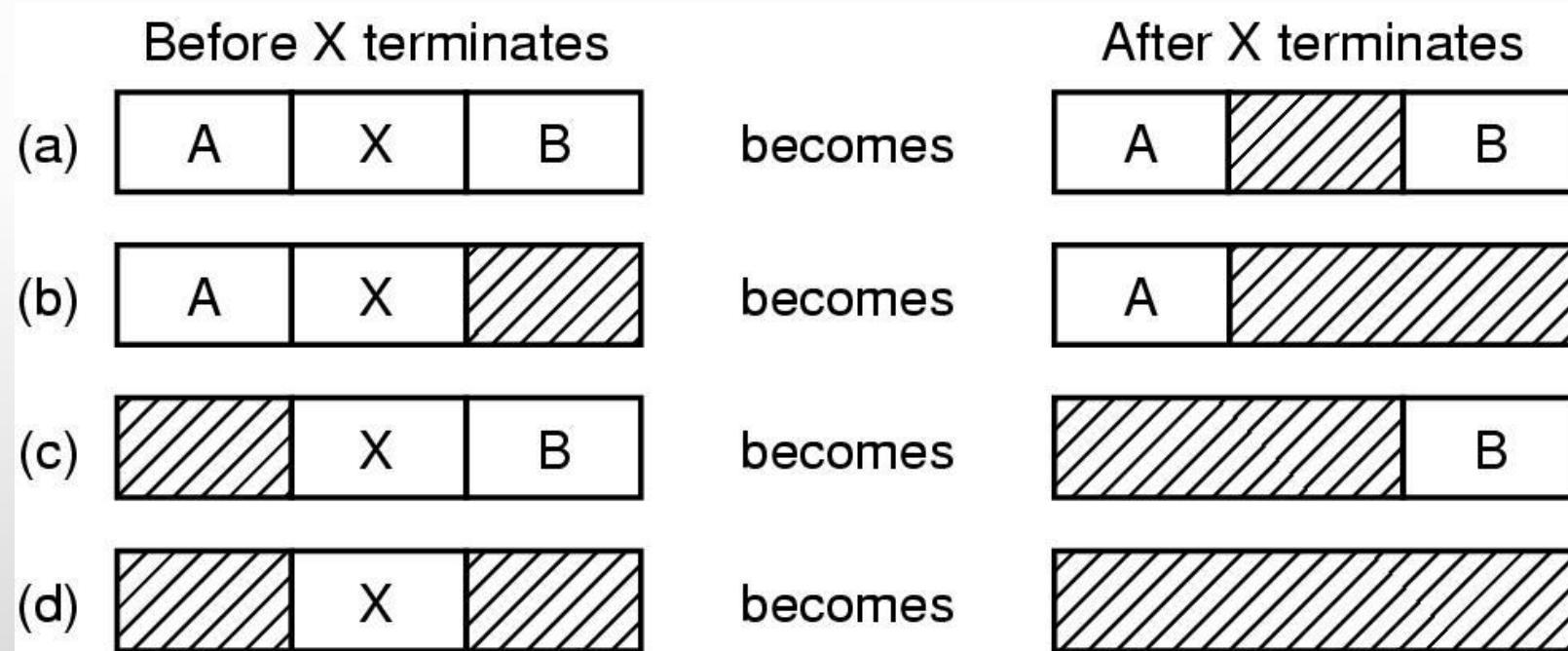
Biteşlem

- Hafızayı takip etmenin kompakt yolu
- k birim uzunluğundaki bir dosyayı getirmek için hafızada k ardışık sıfır aramak gereklidir
- Birimler bit veya bayt olabilir



Bağılı Liste ile Bellek Yönetimi

- X Sürecini sonlandırdıktan sonra oluşan bellek düzeni.





Boş Alan Yönetimi – Bağlı Liste

- Çift bağlı liste
- Programlara boş hafıza nasıl tahsis edilir?
 - İlk uyan (first fit)
 - hızlı; başlangıç daha sık kullanılır; büyük bir boş alanı kırmak
 - Sonraki uyan (next fit)
 - Her seferinde en son kullanılan yerden
 - En iyi uyan (best fit)
 - Tüm listeyi arayıp, gerekli boyuta yakın deliği bulan
 - En kötü uyan (worst fit)
 - En büyük deliği bulur
 - Hızlı uyan (quick fit)
 - İşlemler ve delikler için ayrı kuyruklarda tutulur



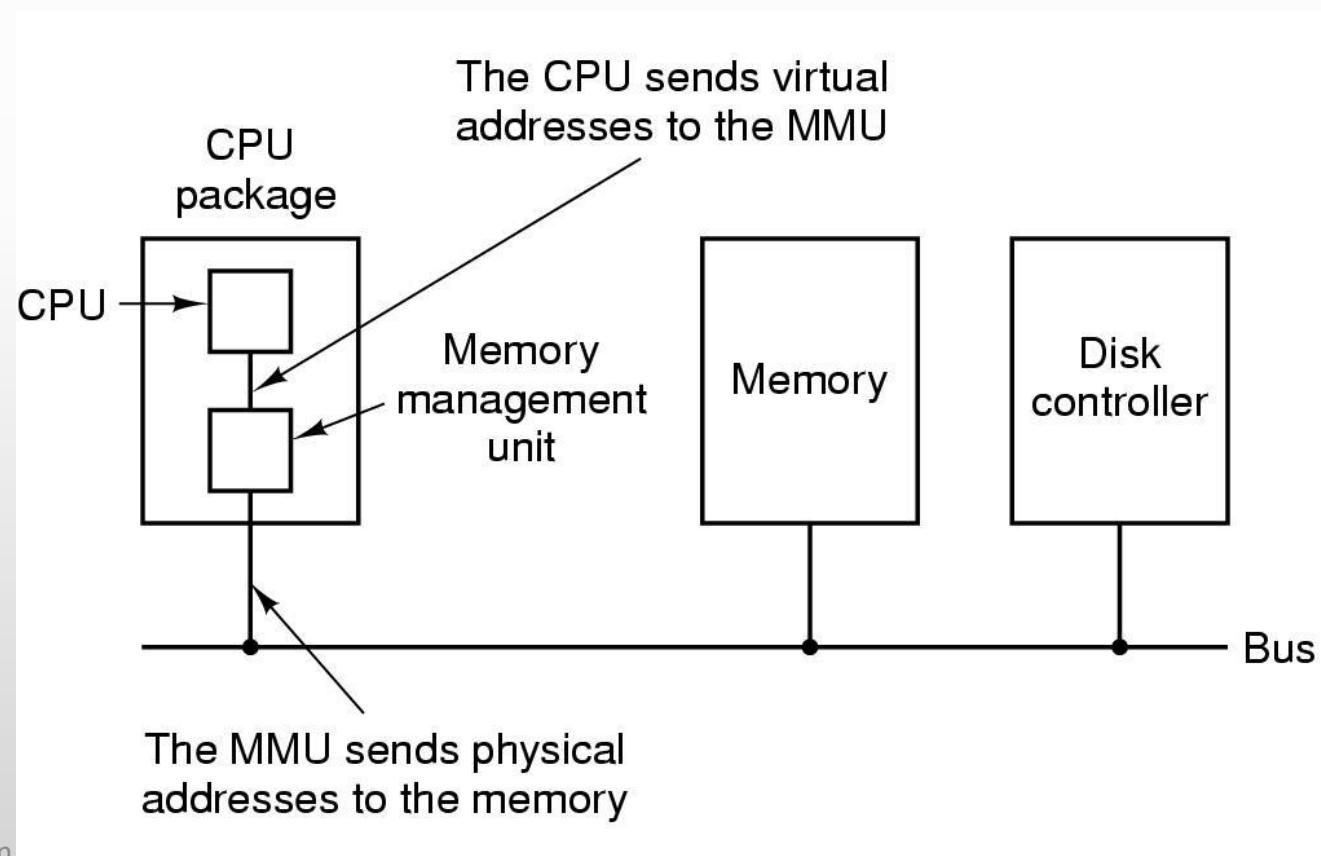
Sanal Bellek – Şişkin (bloat) Yazılım Yönetimi

- Programların belleğe sığmayacak kadar büyük olduğu yerler
- Programlar tarafından bölünmek kötü bir fikir
- Sanal bellek
 - Her programın kendi adres alanı vardır
 - Adres alanı, sayfa adı verilen parçalara bölünmüştür.
 - Her sayfa bitişik bir alandır ve fiziksel adresle eşlenir
 - Tüm sayfaların fiziksel bellekte olması gerekmek.
 - İşletim sistemi, sayfa adreslerini ve fiziksel adresleri ihtiyaç anında eşler
 - Gerekli bir sayfa bellekte olmadığında, işletim sistemi halleder
 - Her sayfa değişim tokusu ihtiyaç duyar



Sanal Bellek - Sayfalama

- MMU'nun konumu ve işlevi – işlemcinin bir parçası olarak gösterilir





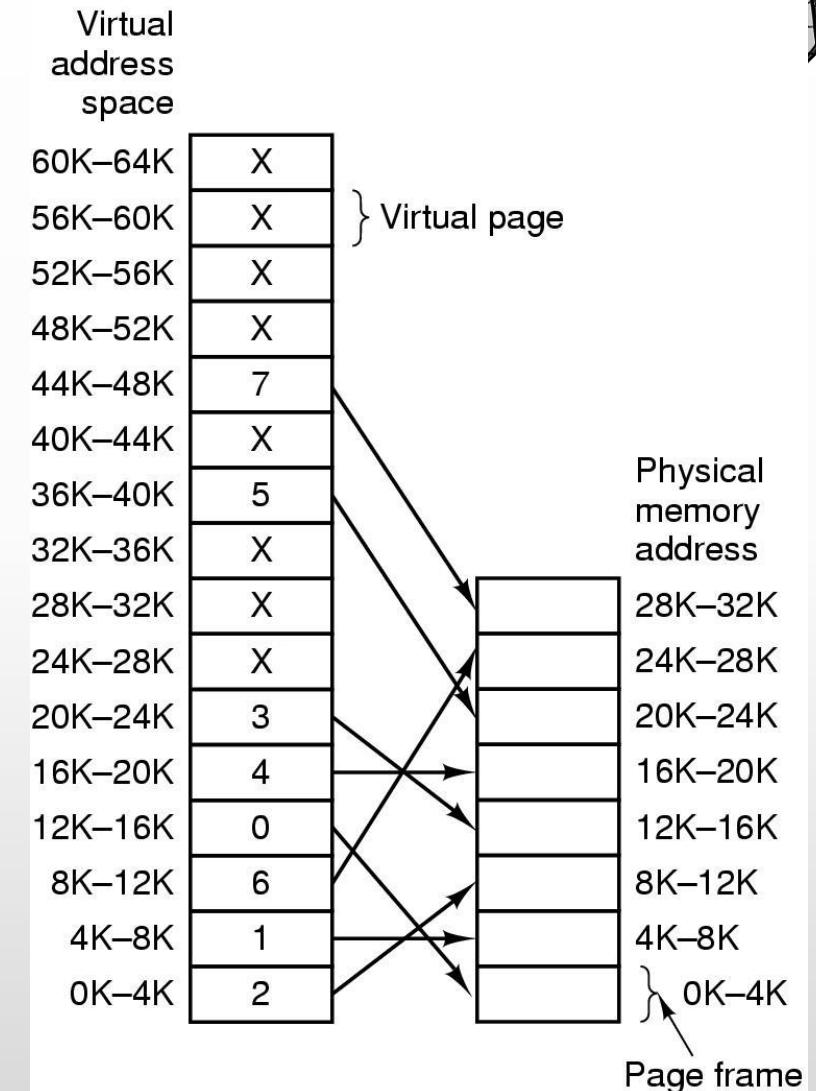
Sayfa ve Sayfa Çerçeveleri

- Sanal adresler sayfalara bölünmüştür
- 512 bayt-64 KB aralığında
- Tüm sayfalar RAM ve disk arasındaki aktarılır



Sanal Bellek – Sayfa Tablosu

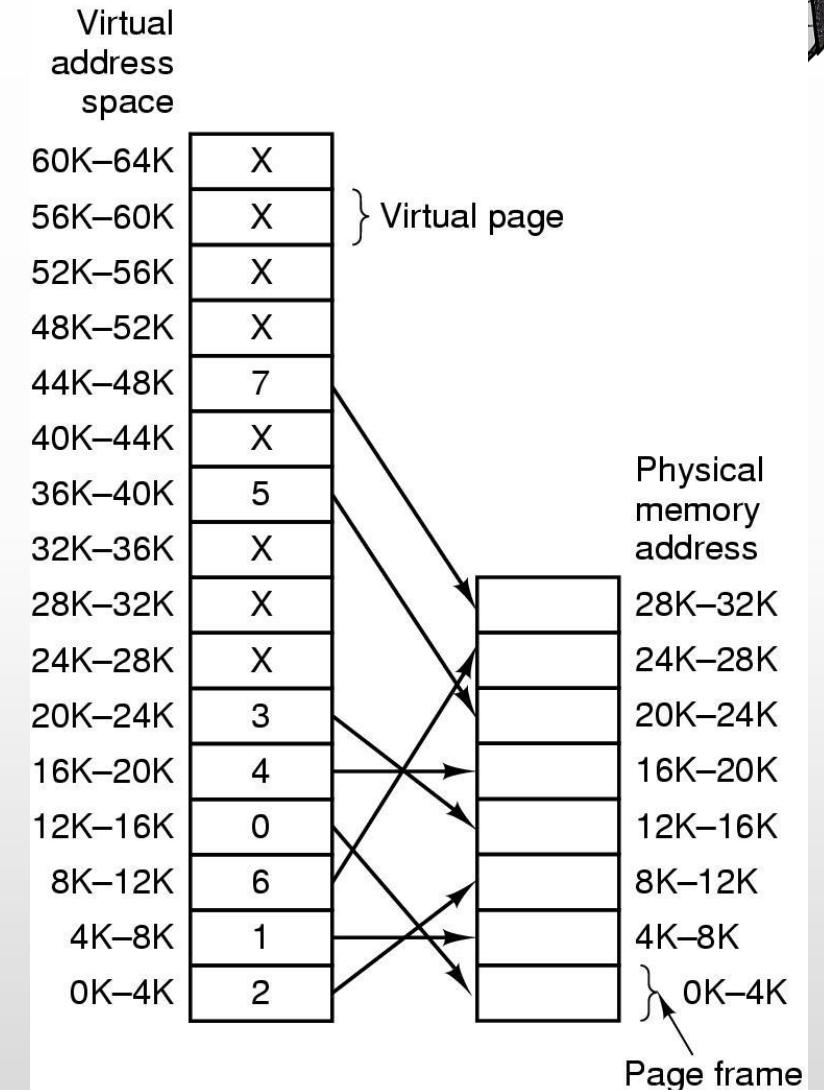
Sayfa tablosu (page table) sanal ve fiziksel bellek adresleri arasındaki ilişkiyi tutar.





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





Hatalı Sayfa İşlemi

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



Sayfa Tablosu

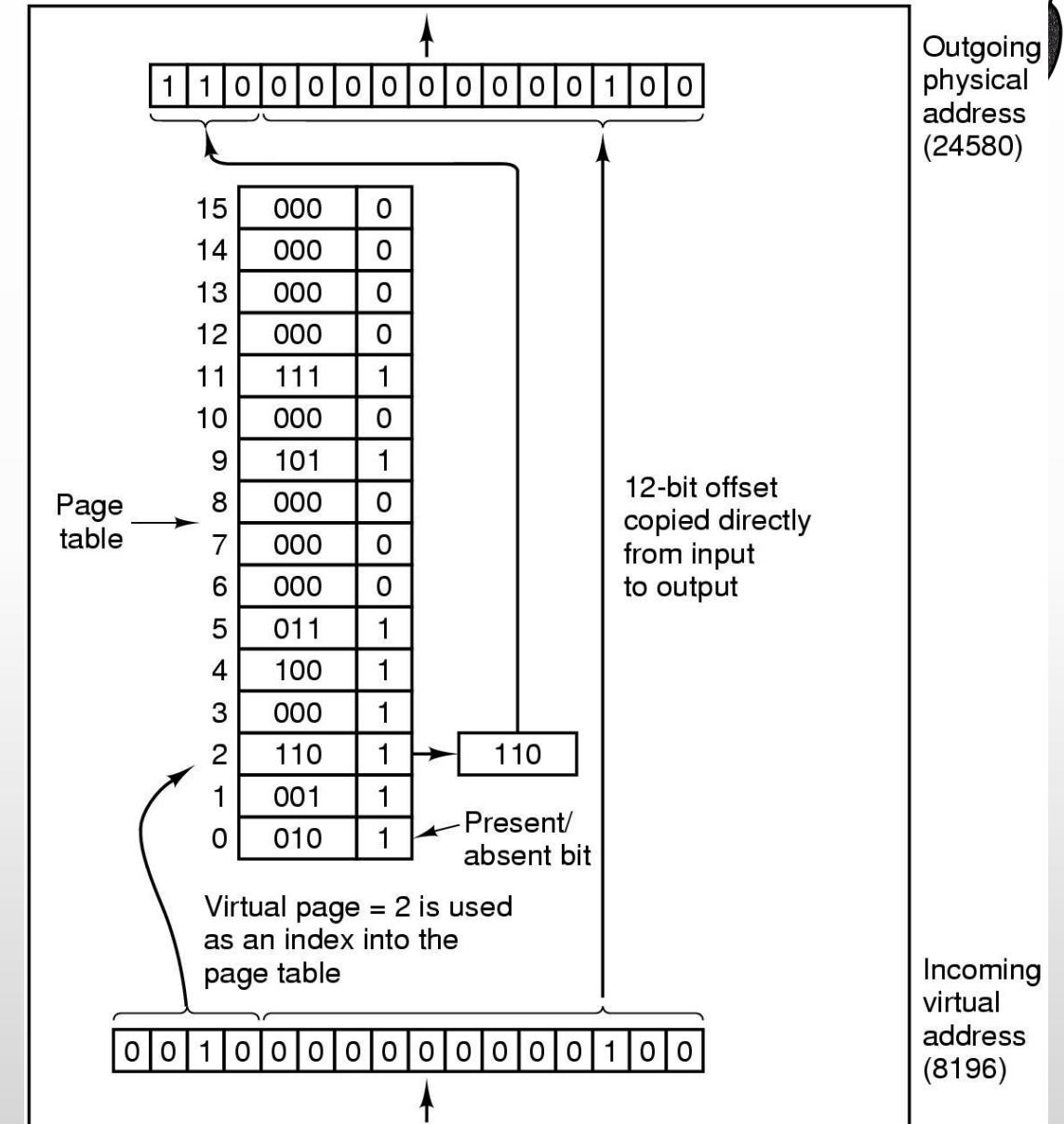
- Sanal adres={sanal sayfa numarası, ofset}
- Sayfa çerçeve numarasını bulmak için sayfa tablosuna dizine eklemek için kullanılan sanal sayfa numarası
- Mevcut/yok biti 1'e ayarlıysa, sayfa çerçeve numarası ofsetin önüne eklenir ve bellek veri yolunda gönderilen fiziksel adres oluşturulur.



Outgoing physical address (24580)

Bellek Yönetim Birimi

- 16 adet 4 KB sayfalı MMU'nun dahili çalışması



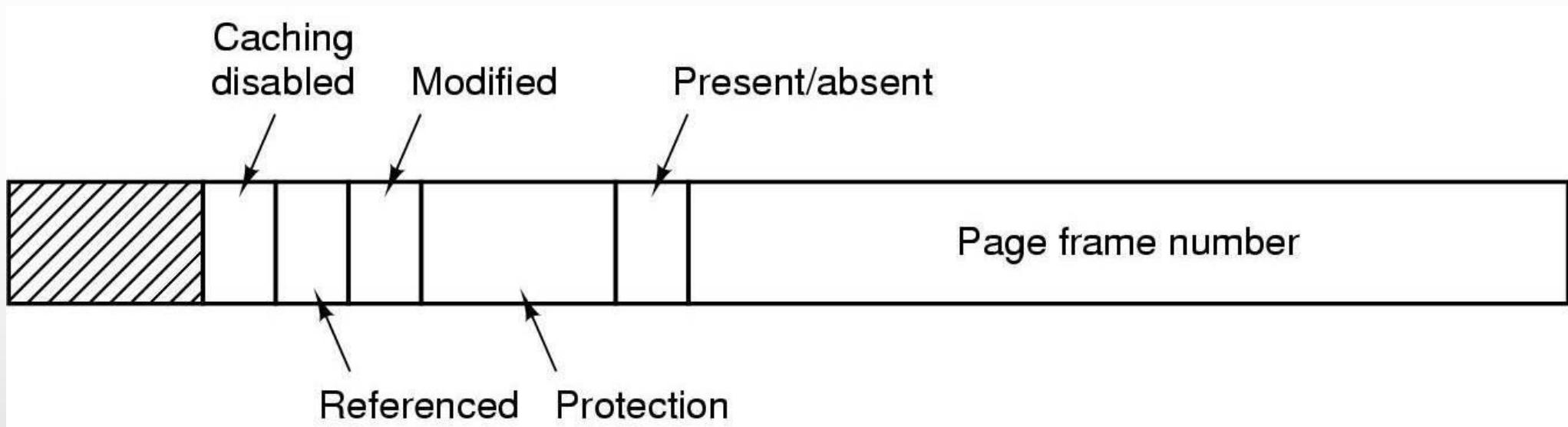


Sanal Adres Eşleme

- 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 Yapısı

- Koruma
 - Ne tür erişimlere izin verilir?
- Değiştirilmiş:
 - Bir sayfa yazıldığındá (kirli)
- Erişilen:
 - Bir sayfa referans alındığındá
- 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 gerekmek
 - 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 erişin & verilere/komutlara erişin
- Çözüm:
 - İlişkili yazmaçlar (associative registers) veya çeviriye bakma arabellekleri (translation look-aside buffers) (TLB'ler) adı verilen özel hızlı arama yapan donanım önbelleği



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



- 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 Yönetimi

- RISC makineleri TLB'yi yazılımda yönetir
- TLB hatası MMU donanımı yerine işletim sistemi tarafından işlenir
- MMU'da daha az donanım ihtiyacı ve OK performansı
- Yazılım, hangi sayfaların TLB'ye önceden yükleneceğini anlayabilir (örn. İstemci isteğinden sonra sunucuya 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
 - $EES = (t + \varepsilon) \alpha + (2t + \varepsilon)(1 - \alpha) = 2t + \varepsilon - t\alpha$
- $\varepsilon(20 \text{ ns})$, $t(100 \text{ ns})$, $\alpha 1(\%80)$, $\alpha 2(\%98)$ ise:
 - TLB hit: $20+100=120 \text{ ns}$
 - TLB miss: $20+100+100=220 \text{ ns}$
 - $EES1 = 120 * 0,8 + 220 * 0,2 = 140 \text{ ns}$
 - $EES2 = 120 * 0,98 + 220 * 0,02 = 122 \text{ ns}$



Büyük Bellek için Sayfa Tablosu

- Adres alanı: 32 bit
- Sayfa boyutu: 4 KB ($4096 \rightarrow 12$ bit)
- Sayfa Numaraları: 20 bit, 1 milyon sayfa
- Sayfa eleman başına 32 bit, sayfa tablosunu tutmak için 4 MB
- 64 bit sistem için?



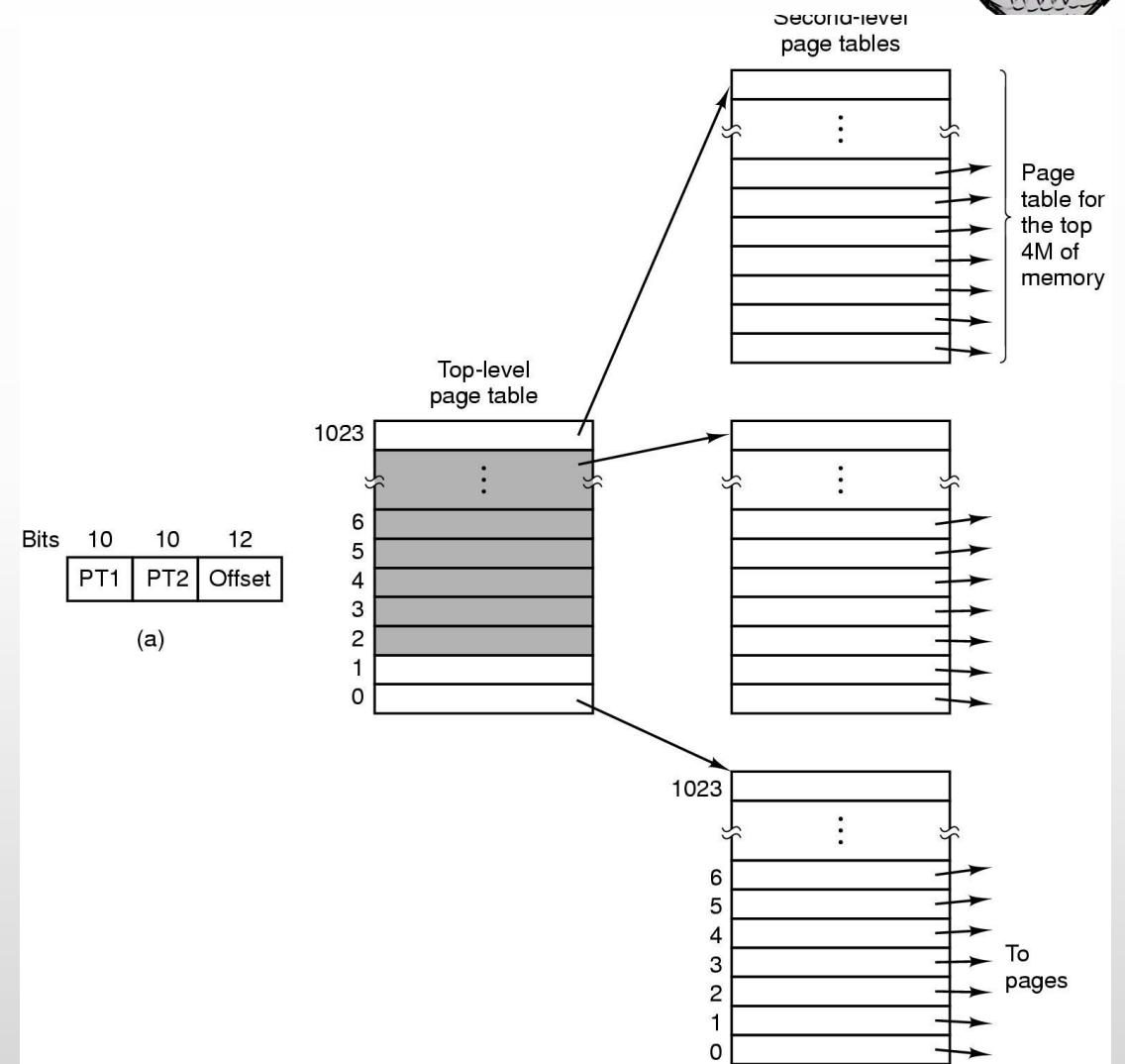
Çoklu Seviye Sayfa Tablosu

- 32 bit sanal bellek üç bölüme ayrılmıştır
 - 10 bit PT1, 10 bit PT2, 12 bit ofset
- Çoklu seviye sayfa tablosu
 - Tüm sayfa tablolarını daima bellekte tutmak gerekmek
 - 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 Tablosunun Kullanımı

- Sayfa tablosunun en üst düzeyi şunları içerir:
- Giriş 0, program metni için sayfalara işaret eder
- Giriş 1, veriler için sayfalara işaret eder
- Giriş 1023 yığın için sayfalara işaret ediyor



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} giriş
- Her giriş 8 bayt ise=> sayfa tablosu için 30 milyon GB
- Yeni bir çözüme ihtiyaç var



Ters Sayfa Tablosu

- "Ters" tabloda (gerçek) sayfa çerçevesi başına bir giriş tutun
- Girişler, sayfa çerçevesiyle ilişkili (sureç, sanal sayfa) takibini yapar
- Her bellek referansı için (n,p) ile ilişkili çerçeveyi bulma ihtiyacı



Ters Sayfa Tablosu

- Her bellek erişiminde sayfa çerçevelerini arar
- Bu verimli bir şekilde nasıl yapılır?
 - Yoğun olarak kullanılan çerçeveleri TLB'de tut
 - Eksikse, sanal sayfadan çerçeveveye eşlemeyi bulmak için ilişkisel aramayı kullanabilir
 - hash tablo kullan

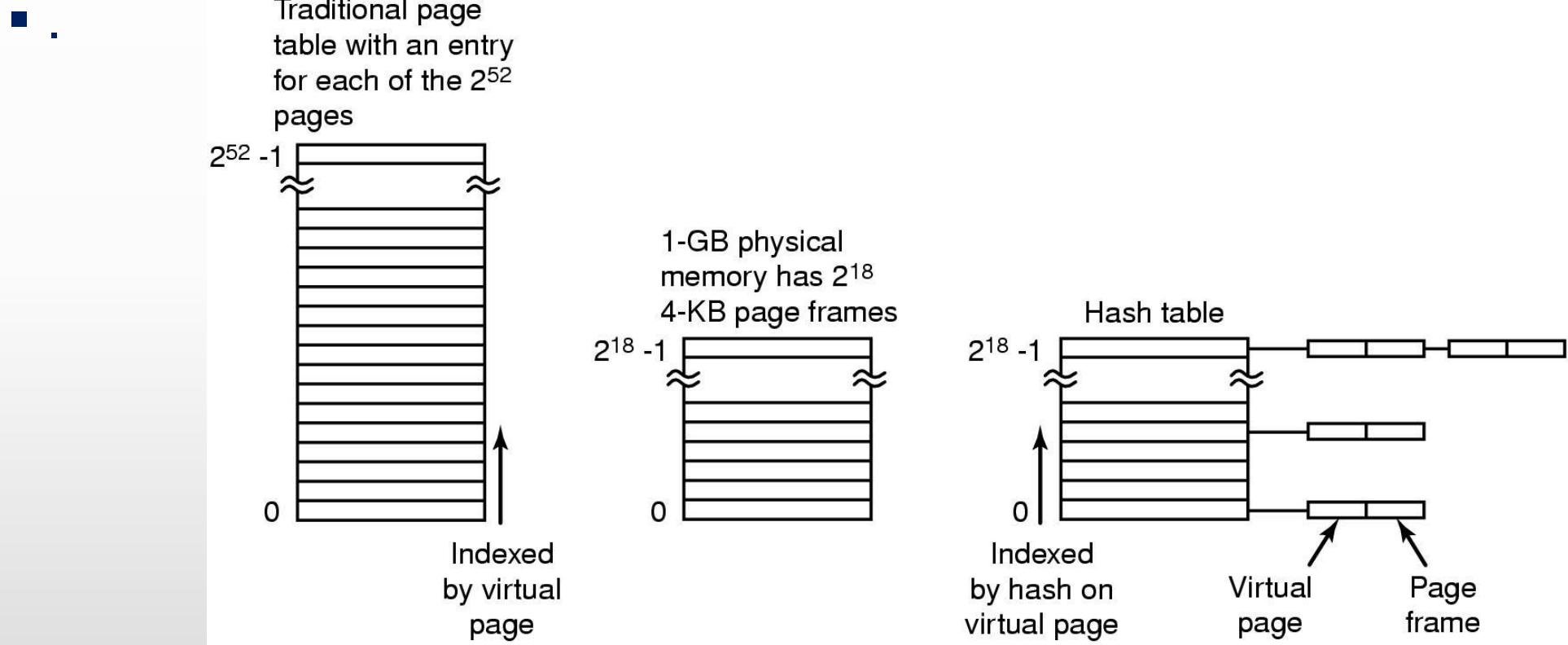


Ters Sayfa Tablosu

- Sanal adres alanları fiziksel bellekten çok daha büyük olduğunda
- Ters sayfa tablosu: sayfa yerine sayfa çerçevesi başına eleman
- Arama çok daha zor
 - TLB
 - Hash
- Ters sayfa tablosu 64 bit sistemlerde yaygındır



Geleneksel ve Test Sayfa Tablosu





Sayfa Değiştirme Algoritmaları

- Optimum (optimal)
- Yakın zamanda kullanılmayan (not recently used)
- İlk Giren İlk Çıkar (first-in first-out)
- İkinci şans (second chance)
- Saat (clock)
- En son kullanılan (least recently used)
- Çalışma kümesi (working set)
- Çalışma kümesi saat (working set clock)



Sayfa Hatasının Etkisi

- Sayfa hatası süresi = 25ms (page fault)
- Bellek erişim süresi = 100ns (memory access)
- Sayfa kayıp oranı p olsun: (miss rate)
- $EAT = 100(1-p) + 25*10^6*p = 100 + 24.999.900*p$
- $p = 1/1000$ ise, $EAT = 25.099,9$ ns
- $EAT < 110$ ns olması gerekiyorsa, o zaman $100 + 24.999.900*p < 110$ yani
 $p < 10/24.999.900 < 10/25.000.000 = 1/2.500.000 = 4 \times 10^{-7}$
- Sayfa hatası oranı p , 4×10^{-7} 'den küçük olmalıdır



Sayfa Değişimi

- Bir sayfa hatası oluştuğunda, bazı sayfaların bellekten çıkarılması gereklidir.
- Çıkarılacak sayfa bellekteyken değiştirilmişse, diske geri yazılması gereklidir.
- Çok kullanılan bir sayfa taşınırsa büyük ihtimalle kısa süre sonra geri getirilecektir.
- Değiştirilecek sayfa nasıl seçilmeli?



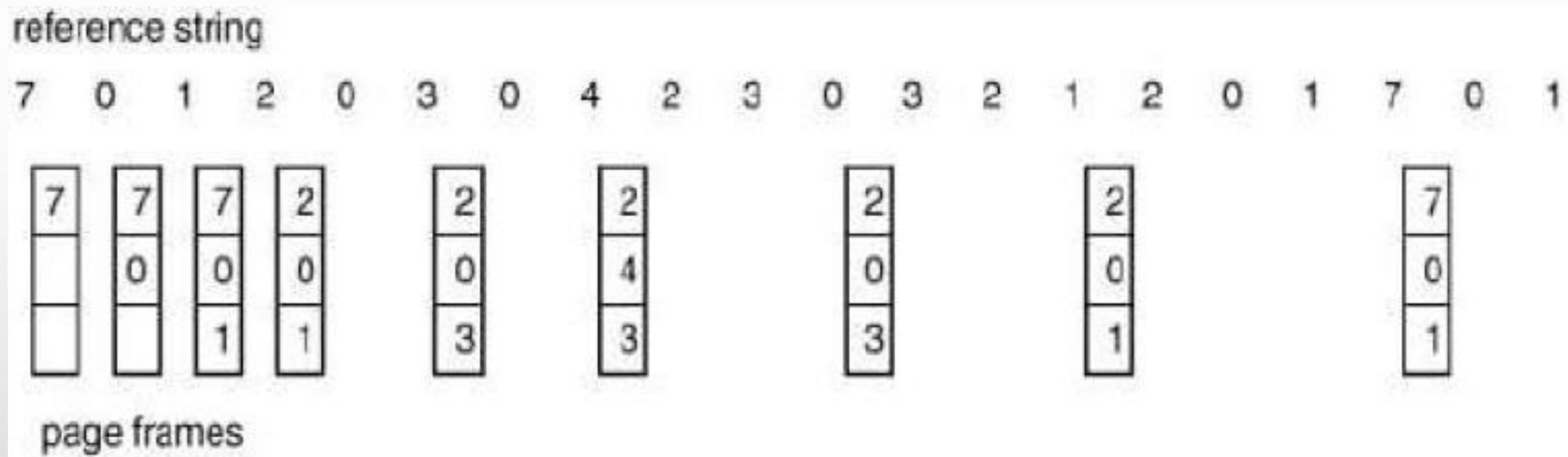
Optimum Sayfa Değişimi

- Tarif etmesi kolay ama uygulaması imkansız
- Her sayfa, o sayfaya ilk erişimden önce çalıştırılacak komutların sayısı ile etiketlenir.
- İhtiyaç duyulduğunda en yüksek etikete sahip sayfa kaldırılır
- En uzun süre kullanılmayacak olanı seçilir
- İşletim sistemi hangi sayfaya ne zaman erişileceğini bilemez (crystal ball)
- Ancak; gerçekleştirilebilir algoritmaların performansını karşılaştırmak için kullanılabilir



Optimum Sayfa Değişimi

Sayfa hatası 9 kez, değiştirme 6 kez





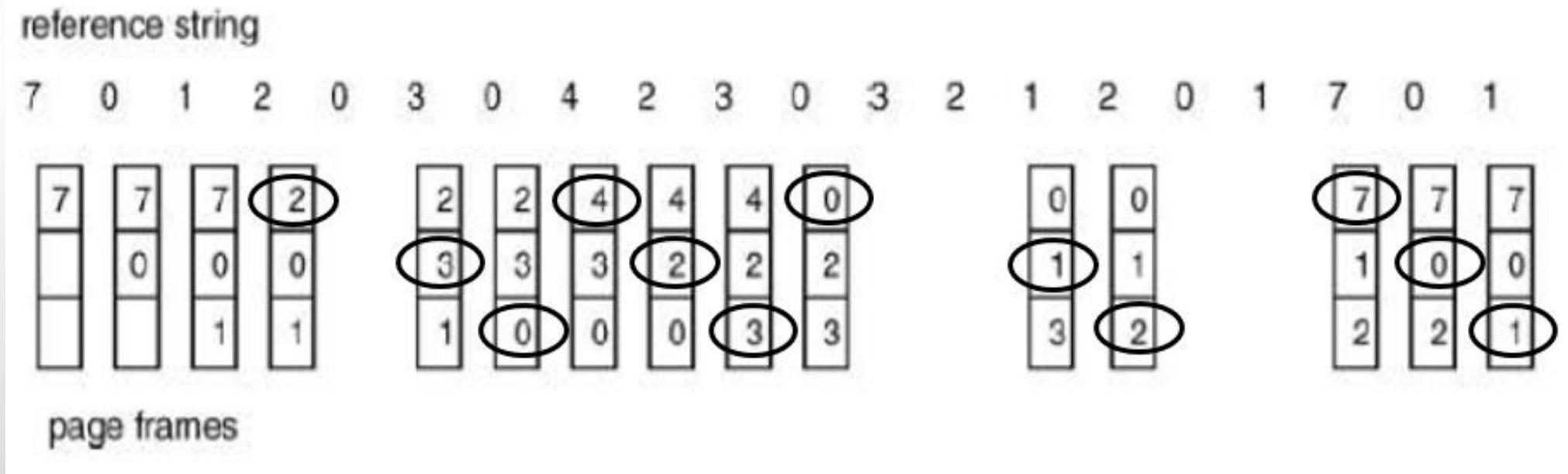
Yakın Zamanda Kullanılmayan Sayfa Değişimi

- Sayfaları kullanımlarına göre değiştirme, Çıkarmak için en düşük öncelikli sayfayı seç
- Sayfalarda R ve M biti bulunur (referenced, modified)
- Bir işlem başlatıldığında, her iki sayfa bitine de 0 atanır, periyodik olarak, R biti temizlenir
- Dört farklı durum oluşabilir
 - Sınıf 0: erişilmedi, değiştirilmedi
 - Sınıf 1: erişilmedi, değiştirildi
 - Sınıf 2: erişildi, değiştirilmedi
 - Sınıf 3: erişildi, değiştirildi



İlk Giren İlk Çıkar Sayfa Değişimi

- "En eski olanı" değiştirir, Tatmin edici olmayan performans ama basit, Sayfa hatası 15, değiştirme sayısı 12





İlk Giren İlk Çıkar Sayfa Değişimi

- Listeyi zamana göre sıralı tut (en sonucusu listenin sonuna gelir)
- En yaşlıyı, yani sıranın başını tahliye et
- Uygulaması kolay
- En eskisi en yoğun şekilde kullanılmış olabilir!
- FIFO'ya hiçbir kullanım bilgisi dahil değildir



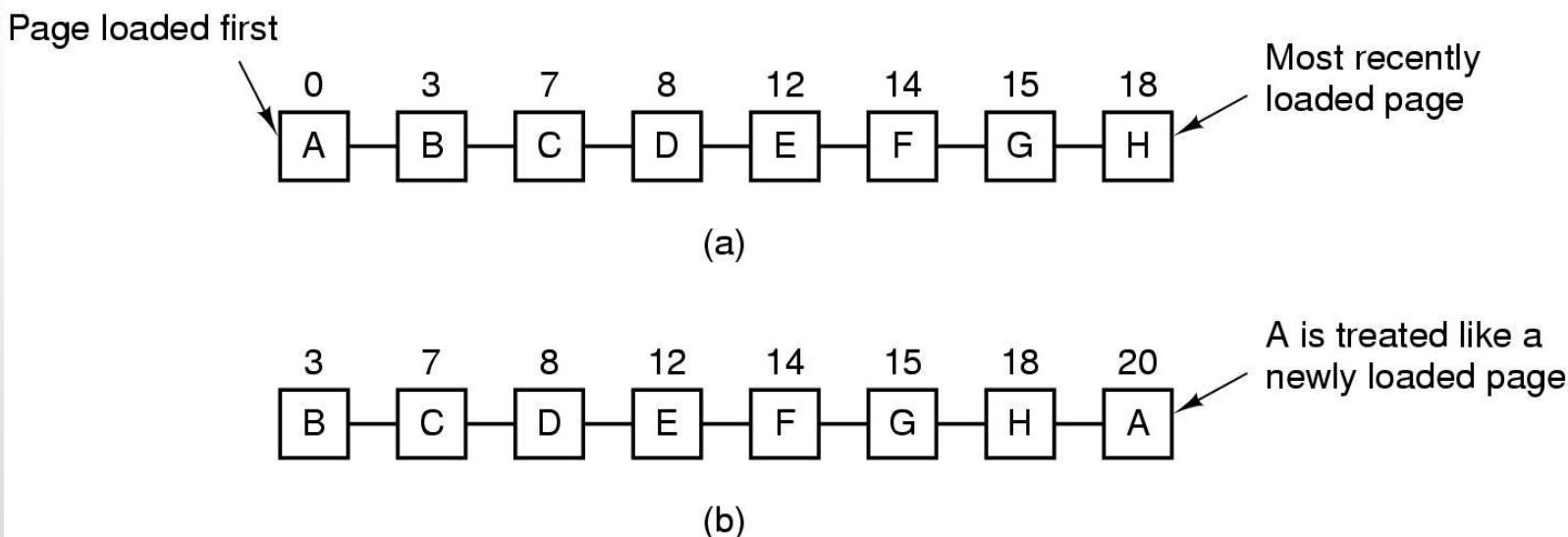
İkinci Şans Sayfa Değişimi

- «İlk giren ilk çıkar» yöntemine basit bir değişiklik yapılarak çok kullanılan bir sayfanın değiştirilmesi önlenmiştir
 - R biti incelenir
 - Eğer R değeri 0 ise, sayfa eskidir ve kullanılmamıştır
 - Eğer R değeri 1 ise sayfaya ikinci şansı ver
 - Eğer tüm sayfalara erişim olduysa, «İlk giren ilk çıkar» gibi çalışır



İkinci Şans Algoritması

(a) FIFO sırasına göre sıralanmış sayfalar. (b) Zaman 20'de bir sayfa hatası oluşursa ve A'nın R biti 1 ise sayfa listesi. Sayfaların üzerindeki sayılar yükleme süreleridir.



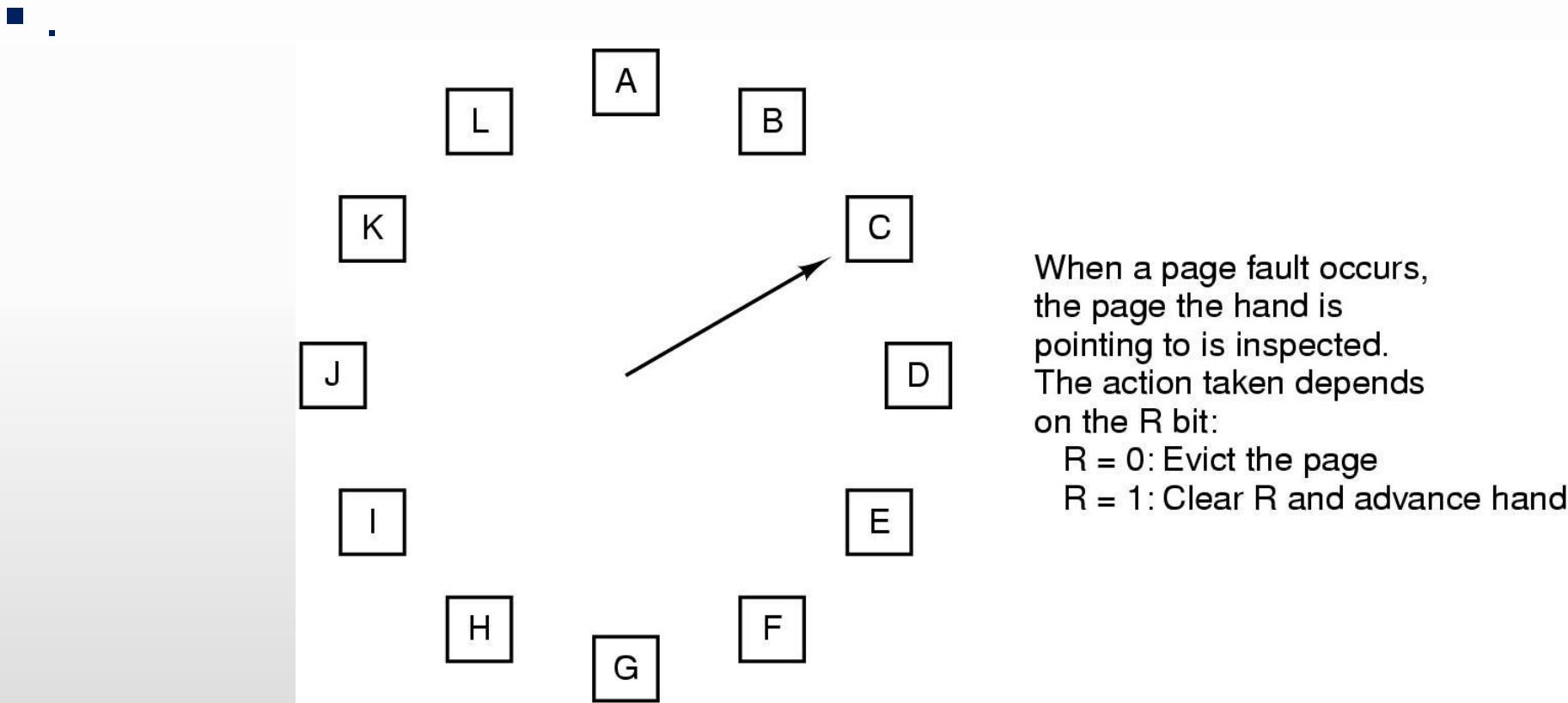


Saat Sayfa Değiştirme

- İkinci şans algoritması verimsizdir
 - Sayfaları listesinde sürekli olarak hareket ettirir
- Alternatif
 - Tüm sayfa çerçevelerini saat şeklinde dairesel bir listede tut



Saat Sayfa Değiştirme Algoritması





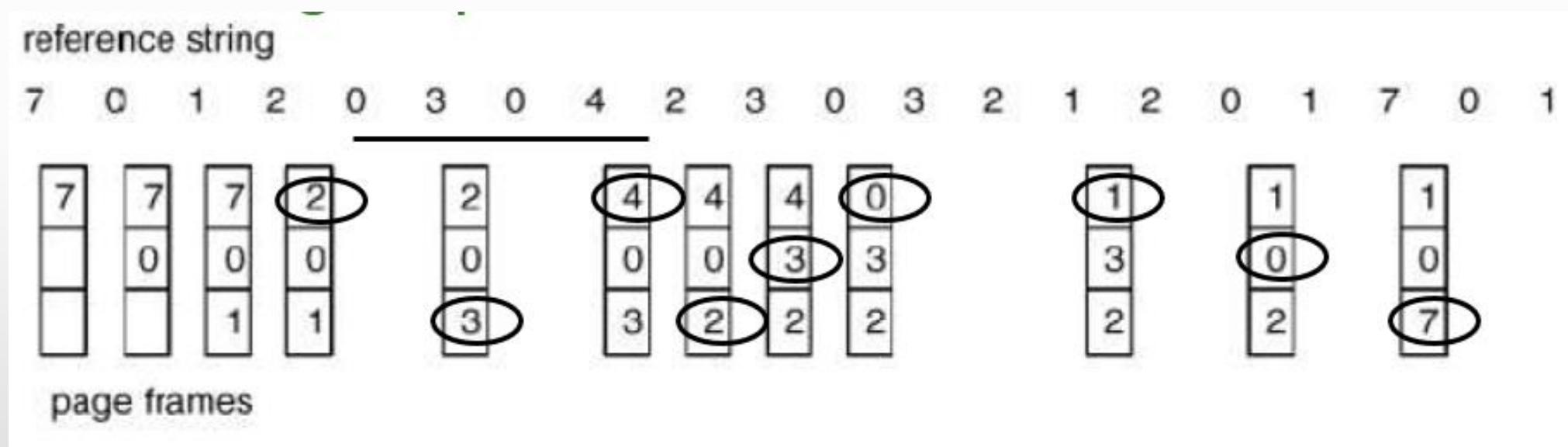
En Son Kullarılanı Değiştirme Algoritması

- Optimum algoritmaya yakın bir yaklaşım
 - Uzun zamandır kullanılmayan sayfalar muhtemelen uzun süre daha kullanılmayacaktır
 - Bir sayfa hatası oluştuğunda, en uzun süre kullanılmayan sayfayı at
- LRU'yu gerçekletemek için
 - Zamanı kaydetmek için bellekteki tüm sayfaları tutan bağlı liste kullan
 - Yada donanım sayacı veya bir matris kullan



En Son Kullarılanı Değiştir Algoritması

- Sayfa hata sayısı: 12, Sayfa değiştirme sayısı: 9





En Son Kullarılanı Değiştir Algoritması

- Verimli bir şekilde gerçekleştirmesi zor
- Sayfada fazladan bir sayaç (kullanım süresi) alanı kullan
 - En eski sayfayı değiştir
 - Sayaçlı donanım gerektirir
 - Sayfa hatasında, en düşük olanı bulmak için tüm sayaçları kontrol eder
- Bir yığında sayfa numaralarını sakla (yazılımsal)
 - En alttaki sayfayı değiştir
 - Yığının boyutu, fiziksel çerçevelerin boyutu kadar
 - Eğer sayfa yığının içindeyse çıkar ve geri koy,
 - Değilse, yığına koy (push)

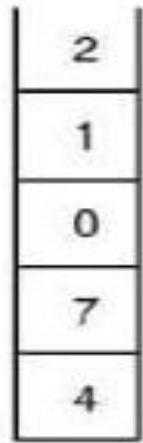


En Son Kullanılanı Değiştir (yığın)

- .

reference string

4 7 0 7 1 0 1 2 1 2 7 1 2



stack
before
a



stack
after
b

a b



En Son Kullanılanı Değiştir (donanım)

- Sayfalara 0, 1, 2, 3, 2, 1, 0, 3, 2, 3 sırasıyla erişildiğinde kullanılan matrisin içeriğinin değişimi.

		Page			
		0	1	2	3
0	0	1	1	1	
1	0	0	0	0	
2	0	0	0	0	
3	0	0	0	0	

(a)

		Page			
		0	1	2	3
0	0	1	1		
1	0	1	1		
2	0	0	0		
3	0	0	0		

(b)

		Page			
		0	1	2	3
0	0	0	0	1	
1	0	0	0	1	
2	1	1	0	1	
3	0	0	0	0	

(c)

		Page			
		0	1	2	3
0	0	0	0	0	
1	0	0	0	0	
2	1	0	0	0	
3	1	1	0	0	

(d)

		Page			
		0	1	2	3
0	0	0	0	0	
1	0	0	0	0	
2	1	1	0	1	
3	1	1	0	0	

(e)

0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	0

(f)

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

(g)

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

(h)

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

(i)

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

(j)



Sık Kullanılmayanı Değiştir

- «En son kullanılanı değiştir» yöntemini yazılımda simüle etmek için, Not frequently used.
- Sayfaların erişim zamanları kaydedilir
- Bir sayfa hatası oluştuğunda, değeri en düşük olan çıkarılır
- Sorun: uzun bir log tutar



Sık Kullanılmayanı Değiştir

Altı sayfa ve beş zaman dilimi için algoritma çıktısı

	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
Page	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10010000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000

(a) (b) (c) (d) (e)



LRU ile Farklar

- Bir saatteki erişimleri ayırt edemez
- Sayaçlar sınırlı sayıda, sınırlı bir geçmişi (log, history) vardır



En Son Kullarılanı Değiştir

- LRU'nun Avantajı:
 - istatistiksel analiz ve kanıtlama, performansının optimalin altında olduğunu gösteriyor
- MRU'nun tercih edildiği durumlar:
 - LRU havuzunda N sayfa varsa, $N + 1$ sayfalık bir dizi üzerinde bir döngü koşturan uygulama, her erişimde bir sayfa hatasına neden olur.
 - Örn: erişim sırası 1,2...,501'dir ve çerçeve büyülüğu 500'dür



Çalışma Kümesi Sayfa Değiştirme Algoritması

- Sayfalama talep:
 - Sayfaları önceden değil talep üzerine yükle
- Erişimler yerel:
 - Süreç koşarken, sayfalarının yalnızca nispeten küçük bir kısmına erişir
- Çalışma seti:
 - Bir sürecin herhangi bir anda kullanmakta olduğu sayfalar kümesi
 - Çalışma kümesi varsa, bir sonraki aşamaya kadar çok fazla sayfa hatasına neden olmaz



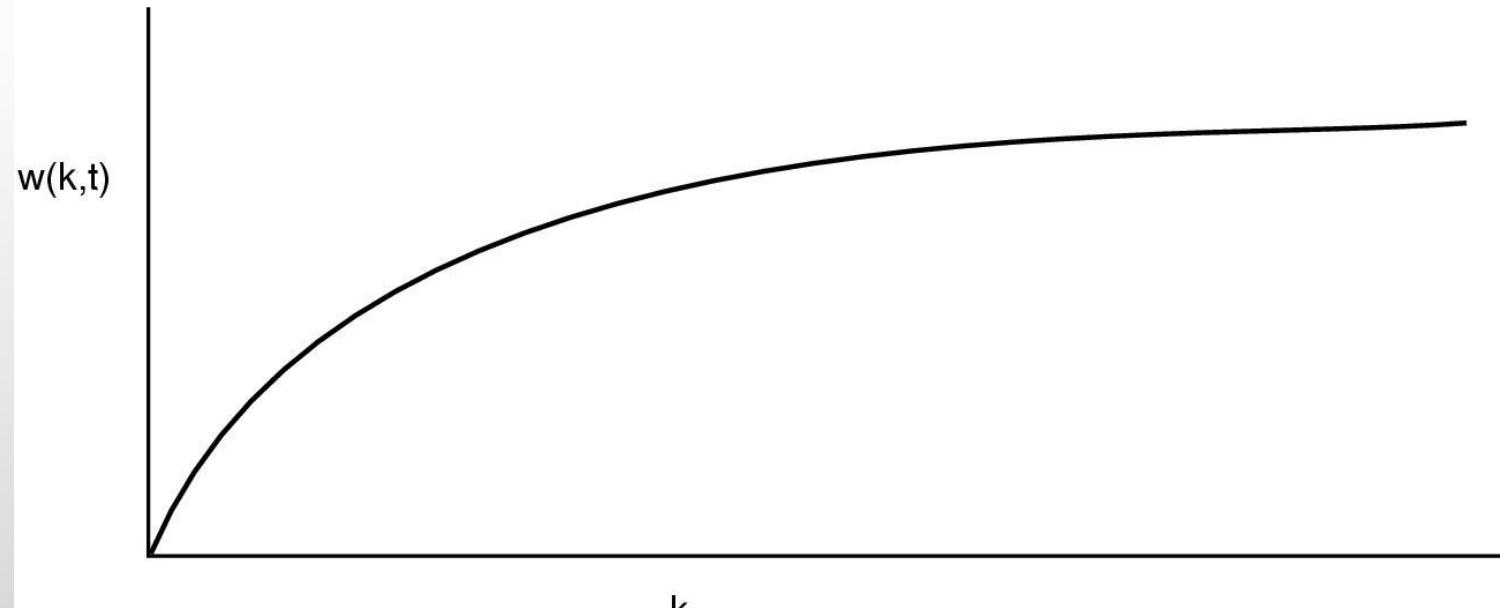
Çalışma Kümesi Modeli

- Bir süreç yer değiştirilip bellekten çıkarıldıktan ve daha sonra yer değiştirilip tekrar belleğe alındığında, önce çalışma kümesi yüklenir, böylece sayfa hatası sayısı büyük ölçüde azaltılır
- Ön sayfalama: bir sürecin çalışmasına izin vermeden önce sayfalarının yüklenmesine ön sayfalama denir



Çalışma Kümesi Sayfa Değiştirme Algoritması

- Çalışma kümesi, k adet en son bellek erişimi tarafından kullanılan sayfa setidir. $w(k,t)$ fonksiyonu, t zamanında çalışan kümenin boyutudur.





Çalışma Kümesi Yer Değiştirme

- Sayfa hatası oluştuğunda, çalışma kümesinde olmayan sayfa çıkarılır
- Çalışan bir küme elde etmek için: bir yazmaç tutulur ve maliyetli olacak her erişimde değeri bir kaydırılır (shift)
- Yakınsamalar
 - K adet bellek erişimi yerine koşma süresini (τ) kullan
- Her bellek erişiminde bellekteki sayfalar takip edilebilir.
- Her k erişim, çalışan bir kümeyle sonuçlanır.
- Yazmaç kaydır uygulaması. Her erişimde sayfa numarasını girilir.
- Masraflı



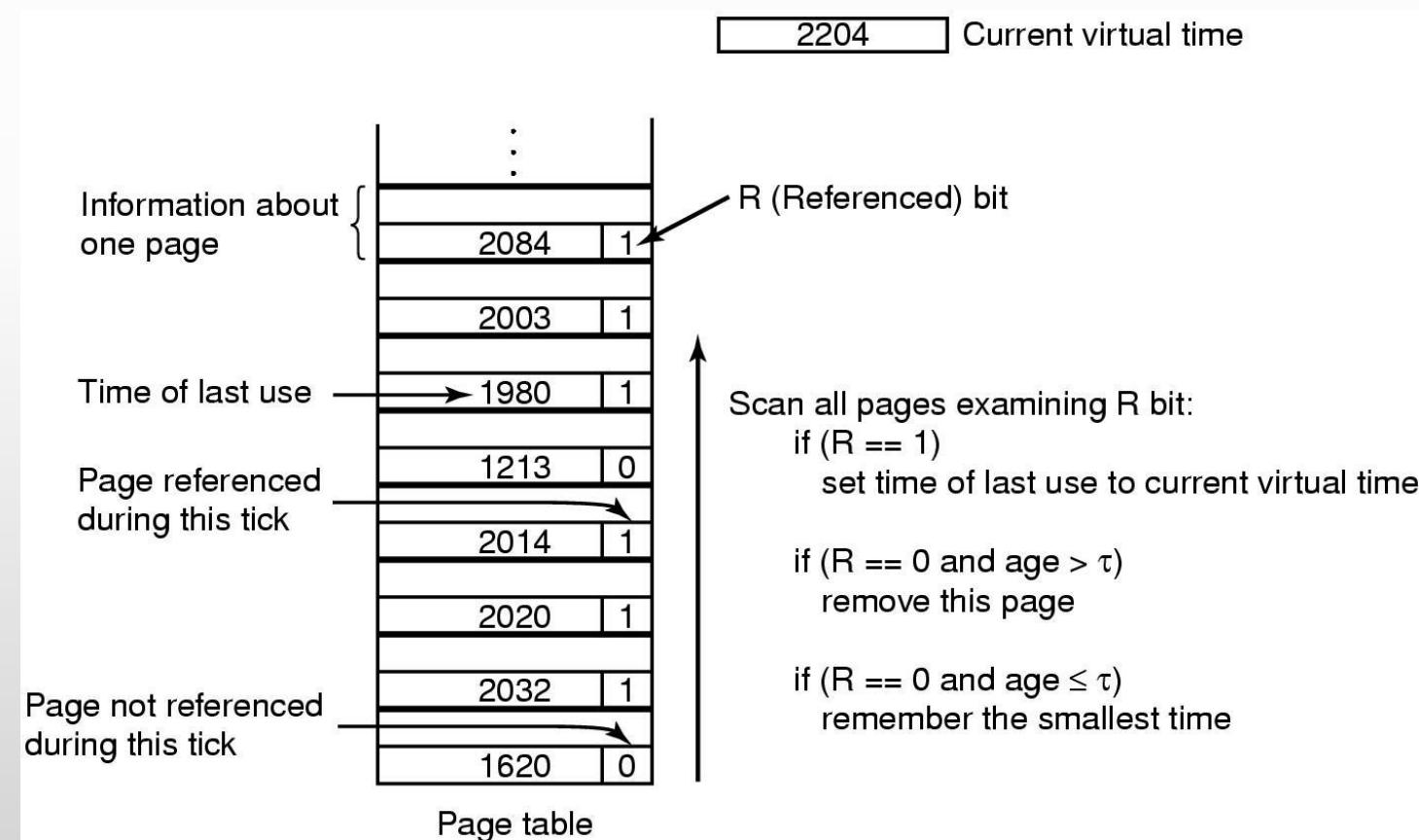
Erişim Sayısı Yerine Sanal Zaman Kullan

- t koşma (CPU) süresi boyunca erişilen son k sayfanın kaydı tutulur
- Sanal zaman, bir süreç için başladığından beri kullandığı işlemci zamanı miktarıdır.
- Bir sürecin ne kadar iş yaptığıının ölçüsü



Çalışma Kümesi Yer Değiştirme

- R biti periyodik olarak temizlenir



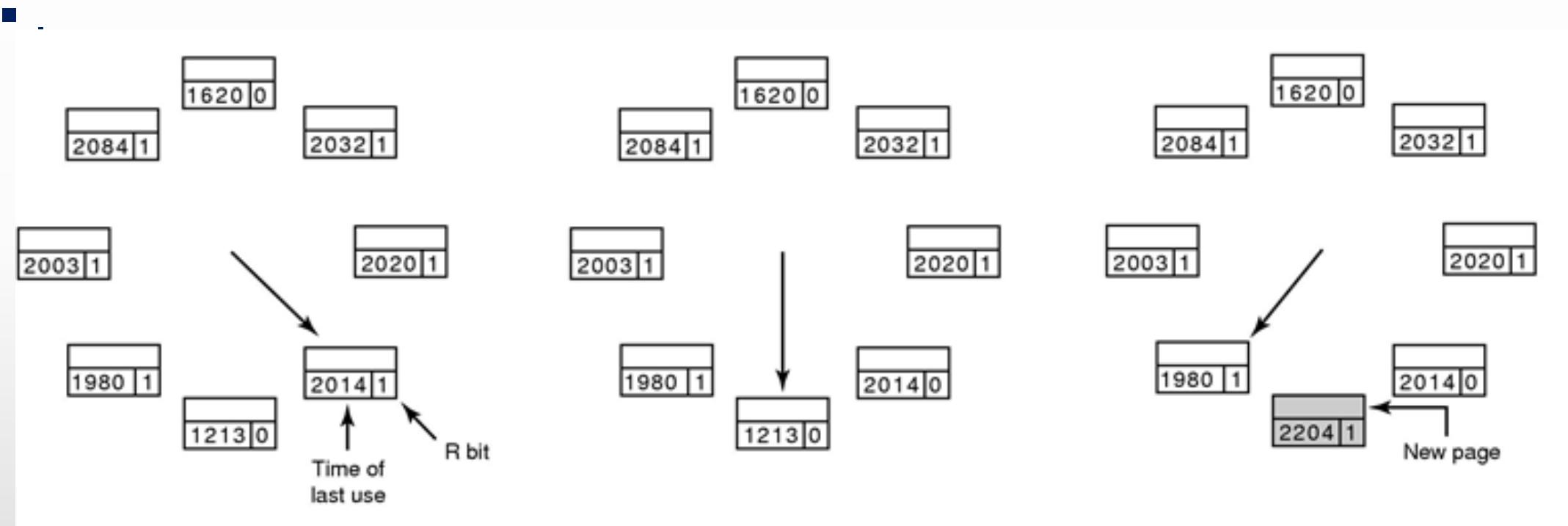


Çalışma Kümesi Saat

- Temel çalışma kümesi algoritmasının her sayfa hatasında tüm sayfa tablosunu taraması gereklidir
- WSClock: basit ve etkili
 - Sayfa çerçevelerinin dairesel listesi
 - R 1 ise, sayfaya geçerli tıklama sırasında erişilmiştir; R'ye 0 ata ve eli ilerlet
 - R 0 ise: ve M 0 ise ve yaş > tau ise yer değiştir; M 1 ise, eli ilerlet ve geri yazmayı çizelgele



Çalışma Kümesi Saat





En Son Kullarılanı Değiştir

- İki durumda el başlangıç noktasına kadar gelir:
 - En az bir yazma çizelgelendi: ibre temiz bir sayfa bulana kadar hareket etmeye devam eder
 - Hiçbir yazma çizelgelenmedi: tüm sayfalar çalışma kümesinde, temiz bir sayfa seç veya temiz sayfa yoksa, geçerli sayfa kurbandır ve diske geri yaz

Özet



Algoritma	Yorum
Optimum	Uygulanabilir değil, ancak kıyaslama olarak kullanışlı
NRU (Yakın Zamanda Kullanılmayan)	Çok kaba
FIFO (İlk Giren İlk Çıkar)	Önemli sayfaları atabilir
İkinci şans	FIFO'ya göre büyük gelişme
Saat	Gerçekçi
LRU (En Son Kullanılanlar)	Mükemmel, ancak tam olarak uygulanması zor
NFU (Sık Kullanılmayan)	LRU'ya yakınsayan adilce kaba
Yaşlanma	LRU'ya iyi yakınsayan verimli algoritma
Çalışma seti	Uygulanması biraz pahalı
WSClock	İyi verimli algoritma



Yerel ve Küresel Tahsis Politikaları

(a) Orijinal konfigürasyon. (b) Yerel sayfa değişimi. (c) Küresel.

	Age
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	4
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

(a)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(b)

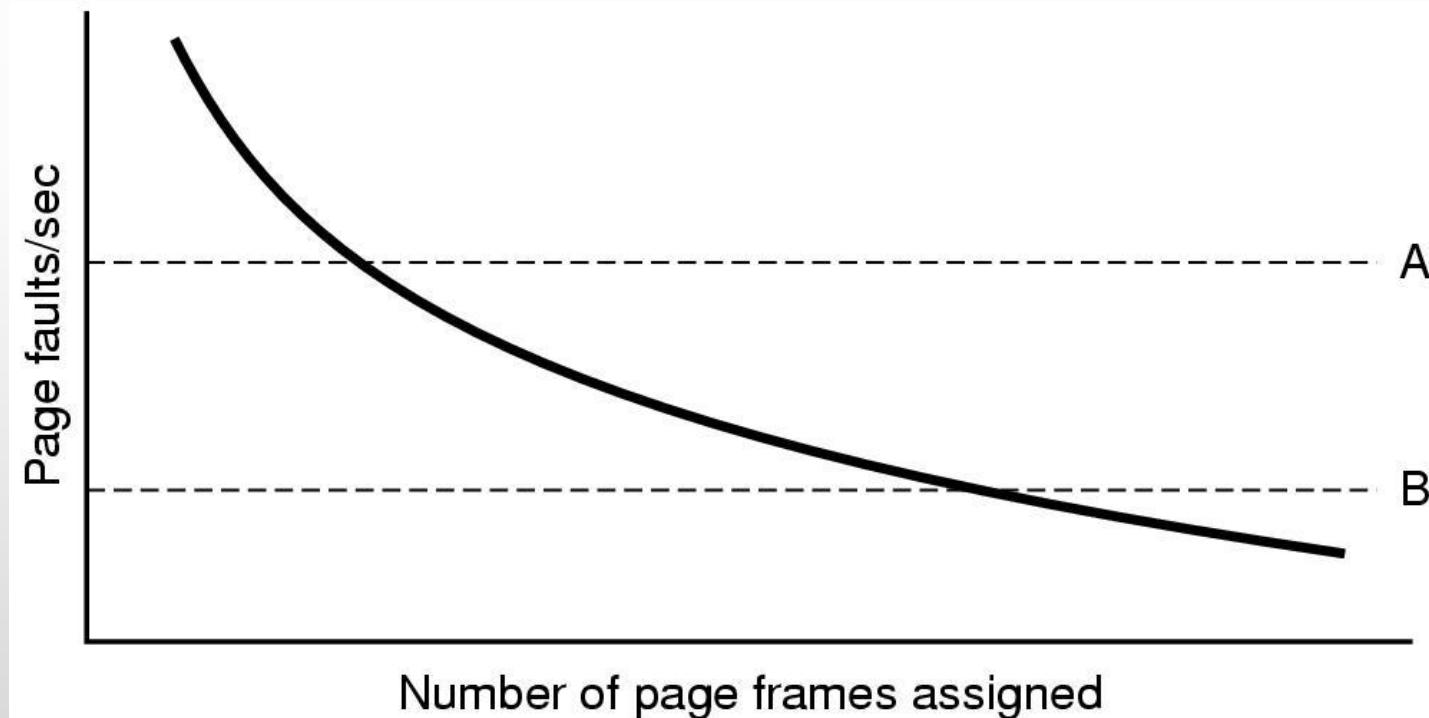
A0
A1
A2
A3
A4
A5
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

(c)



Yerel ve Küresel Tahsis Politikaları

■ ■ ■





Belady Anomalisi

- Çerçeve büyüklükleri,
- Erişim sıralaması: 1 2 3 4 1 2 5 1 2 3 4 5
- 4 çerçeve: sayfa hatası: 10, yer değiştirme 6

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

- 3 çerçeve: sayfa hatası: 9, yer değiştirme 6

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



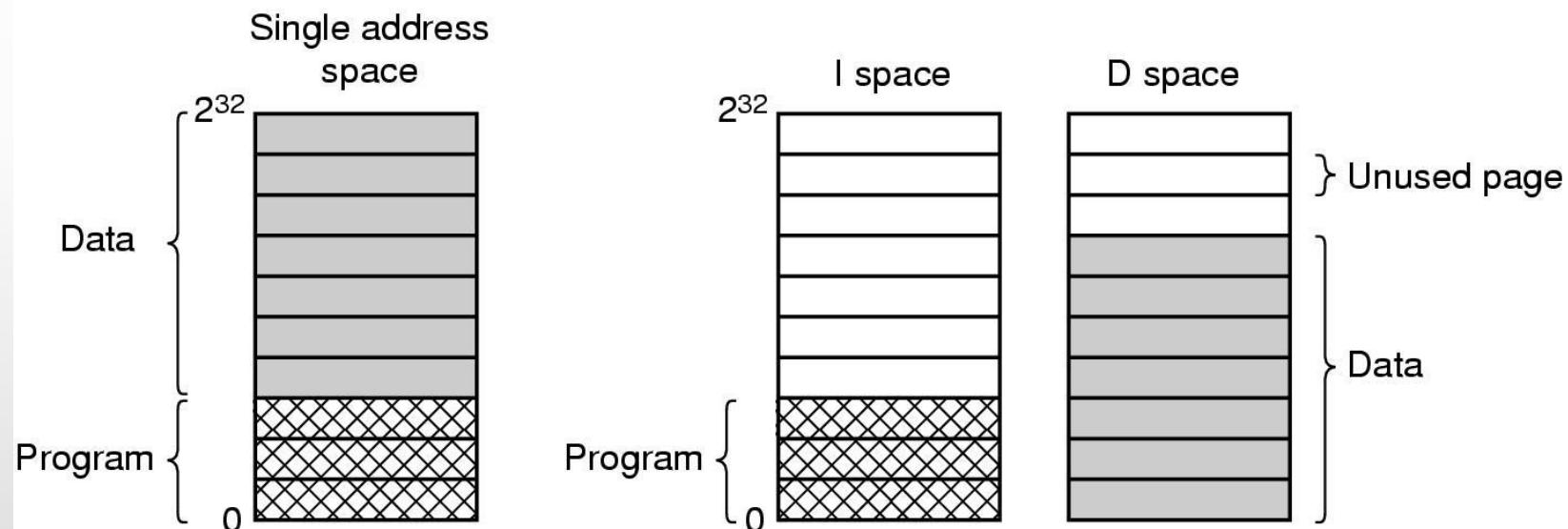
Sayfa Boyutu

- Ek yük = $s^*e/p + p/2$ [sayfa girişlerinin boyutu + parça]
 - p sayfa boyutudur,
 - s süreç boyutudur,
 - e, sayfa girişinin boyutudur (sayfa tablosunda)
- Türev al, sıfır ata => $p = \sqrt{2s^*e}$
 - $s= 1 \text{ MB}$, $e=8 \text{ bayt}$, $p=4 \text{ KB}$ en uygun
 - 1 KB tipik
 - 4-8 KB yaygın
 - bu kaba bir yaklaşım



Ayrı Komut ve Veri Alanları

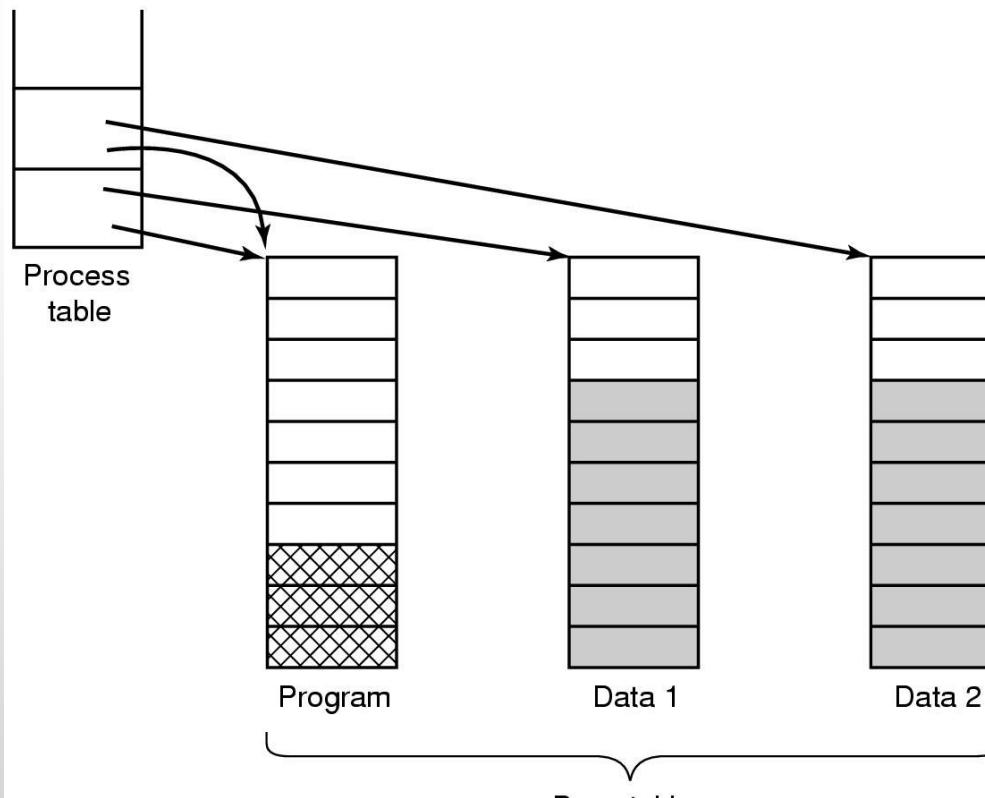
(a) Tek adres alanı. (b) Ayrı I ve D alanları. (instruction, data)





Paylaşımılı Sayfalar

- İki süreç aynı programı ve sayfa tablosunu paylaştığında





Paylaşımılı Sayfalar

- Süreç, sayfaları hala kullanımda olmadıklarından emin olmadan, çıktıgı zaman bırakamaz
 - Paylaşilan 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 gerçekleşse, her işlem kendi sayfasını alır.



Paylaşımı Kütüphaneler

- Birçok işlem tarafından kullanılan büyük kütüphaneler (ör. grafikler).
- Bunları kullanmak isteyen her sürece bağlanmak 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.
- Nesne programını diske yaz.



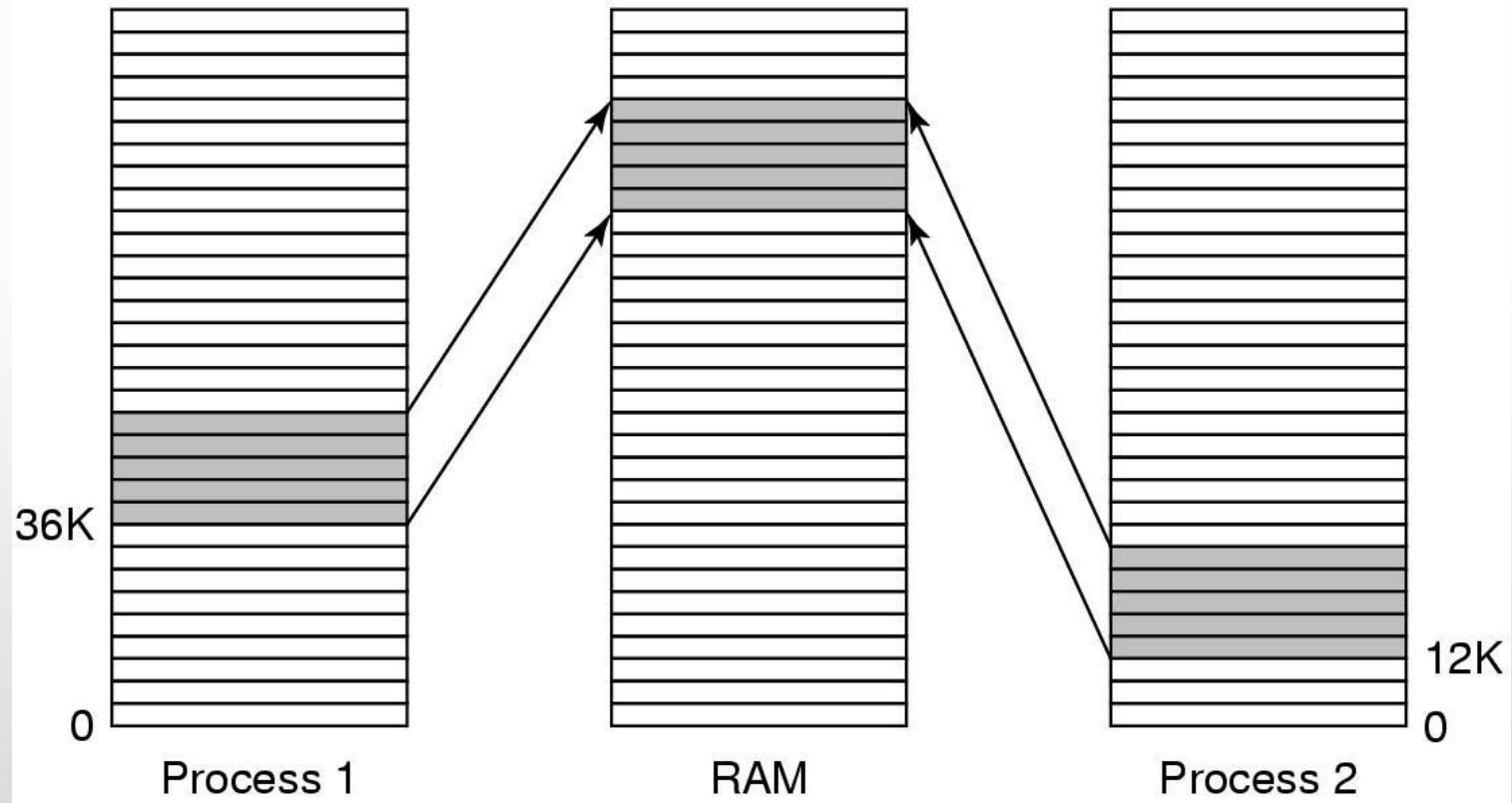
Paylaşımı Kütüphaneler

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



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



Bellek Eşlemeli Dosyalar

- Süreç, bir dosyayı sanal adres alanının bir parçasına eşlemek için sistem çağrıları 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 olduklarından emin olunmalı



Sanal Bellek Arayüzü

- 2 program fiziksel belleği paylaşmak isteyebilir
- Paylaşımılı 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



Gerçekleme

- 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

- Donanım çalışmayı çekirdeğe bırakarak, program sayacını yığına kaydeder.
- Assembler kod parçası genel yazmaçları ve diğer uçucu bilgileri kaydetmeye başlar.
- İşletim sistemi bir sayfa hatasının olduğunu anlar ve hangi sanal sayfanın gerektiğini bulmaya çalışır.
- Hataya neden olan sanal adres bulunduğuanda, sistem bu adresin geçerli olup olmadığını ve korumanın erişimle tutarlı olup olmadığını kontrol eder.



Sayfa Hatasını Ele Alma

- Seçilen sayfa çerçevesi diskten okunduktan sonra değiştirilmişse (dirty, modify), sayfanın diske aktarılması çizelgelenir ve bir içerik değiştirmesi (context switch) gerçekleşir.
- Sayfa çerçevesi temiz ise, işletim sistemi gerekli sayfanın bulunduğu disk adresini arar ve belleğe getirmek için bir disk işlemi planlar.
- Sayfa belleğe taşındığında disk kesmesi oluşur, sayfa tabloları konumu yansıtacak şekilde güncellenir, çerçeve normal durumda olarak işaretlenir.



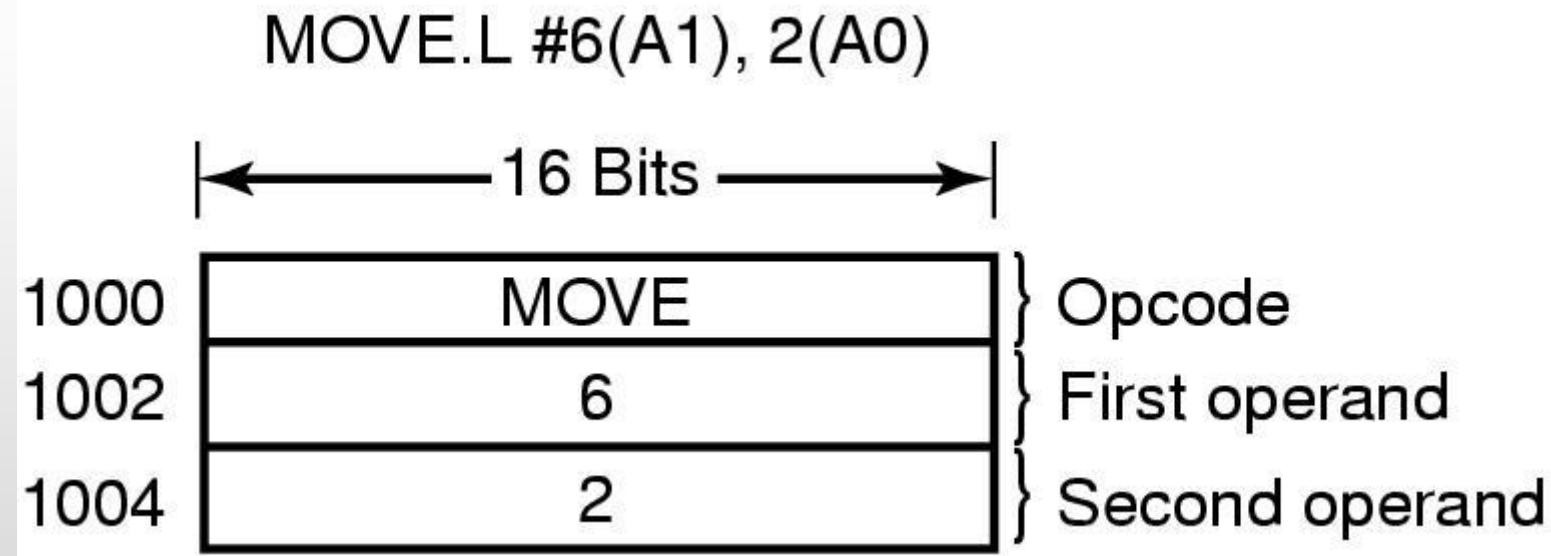
Sayfa Hatasını Ele Alma

- Hatalı kod, başladığı andaki durumuna yedeklenir ve bu komutu işaret etmek için program sayacı sıfırlanır.
- Hataya neden olan süreç çizelgelenir, işletim sistemi onu çağırılan (assembler dili) rutine geri döner.
- 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

- Daha da kötüsü: Otomatik artırma yazmaçları, komut yürütülmeden önce veya sonra yükler. Önce yüklerse, işlemin geri alınması gereklidir. Sonra yüklenirse, hiç yapılmaması gereklidir.
- Komutun yedeklenmesi için donanım çözümü – komut yürütülmeden önce mevcut komutu bir yazmaca kopyala
- Aksi takdirde işletim sistemi bataklığının derinliklerindedir



Bellekte Sayfa Kilitleme

- İşlem, 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 kilitleyin



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 yönetin. Süreç için yeterince büyük parça atayın
- Süreç tablosunda tutulan bölümün başlangıç adresi tutulur. Sanal adres uzayındaki sayfa ofseti, diskteki adrese karşılık gelir.
- Veri, metin, ve yığın için farklı alanlar atanabilir, yığın zamanla genişleyebilir



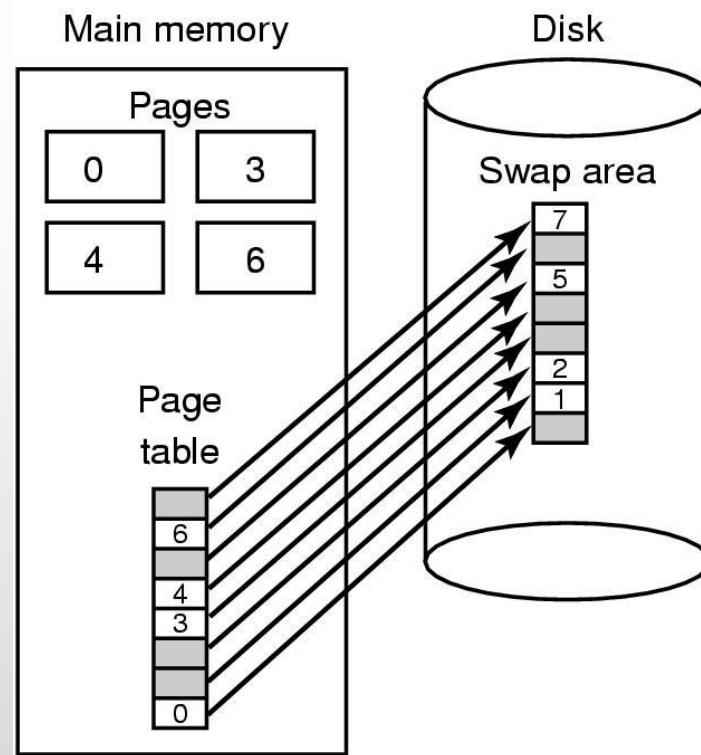
Backing Store – Dinamik Yaklaşım

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

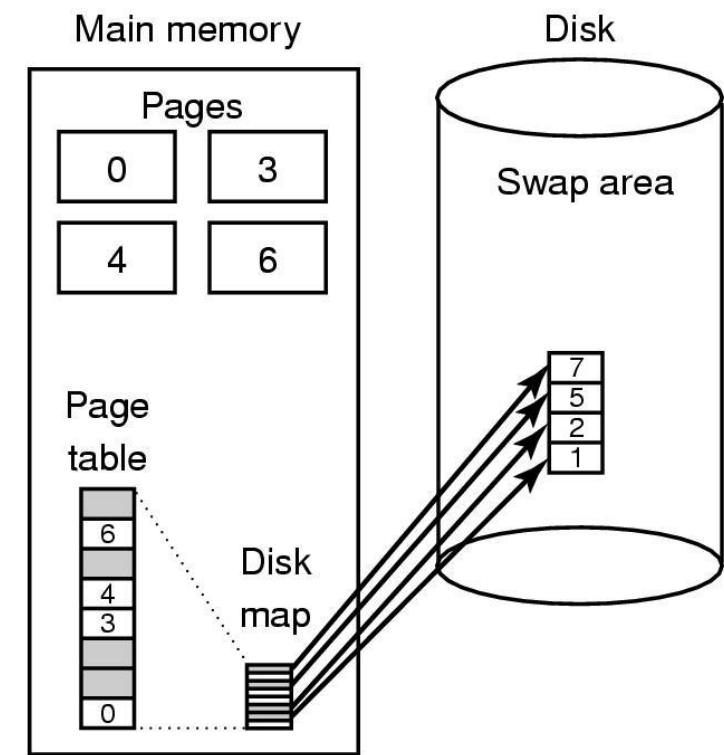


Takas Alanına Sayfalama

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



(a)



(b)

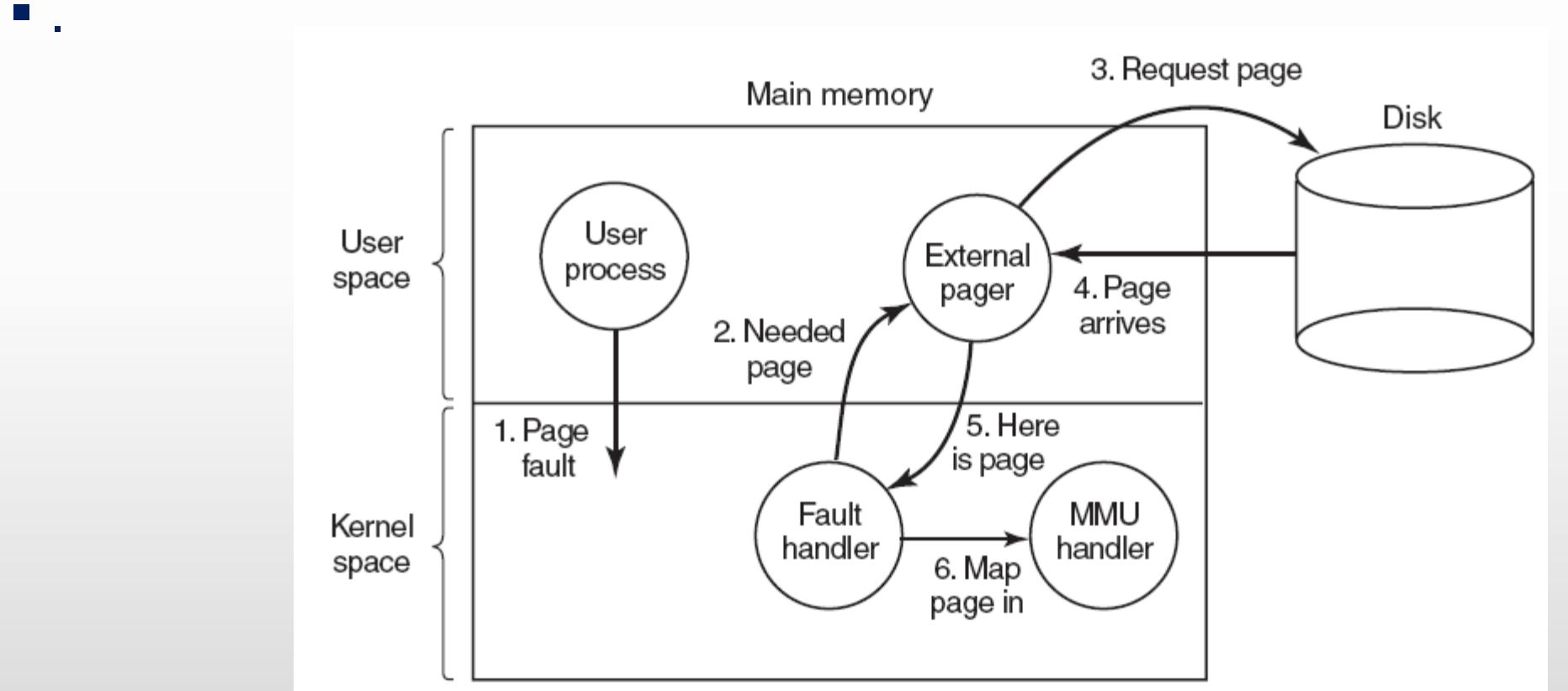


İlke ve Mekanizma Ayrımı (policy, mechanism)

- 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)





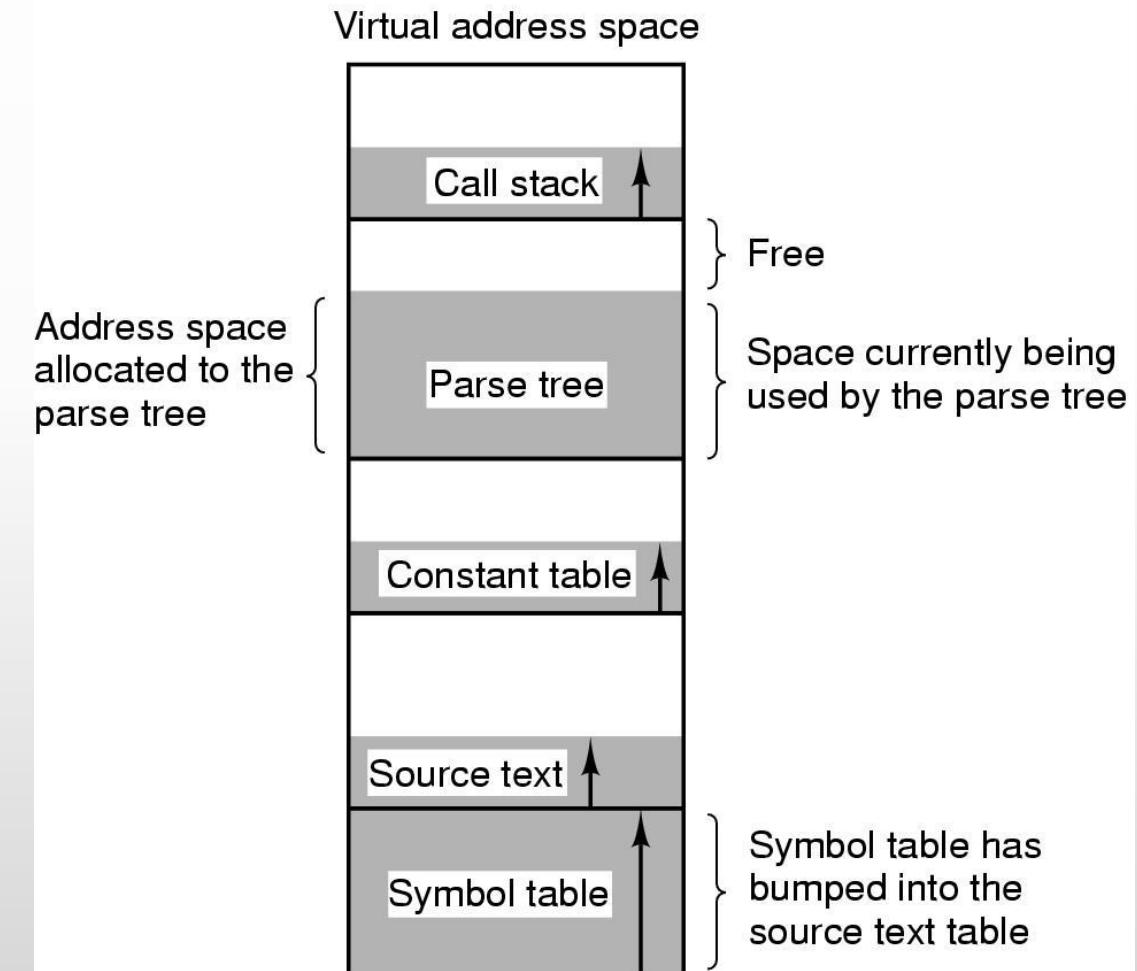
Kesimleme (segmentation)

- 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.
- Ayristirma (parse) ağacı, programın sözdizimsel (syntactic) analizi.
- Derleyici içinde prosedür çağrıları için kullanılan yığın.



Kesimleme (segmentation)

- tek boyutlu adres uzayı.





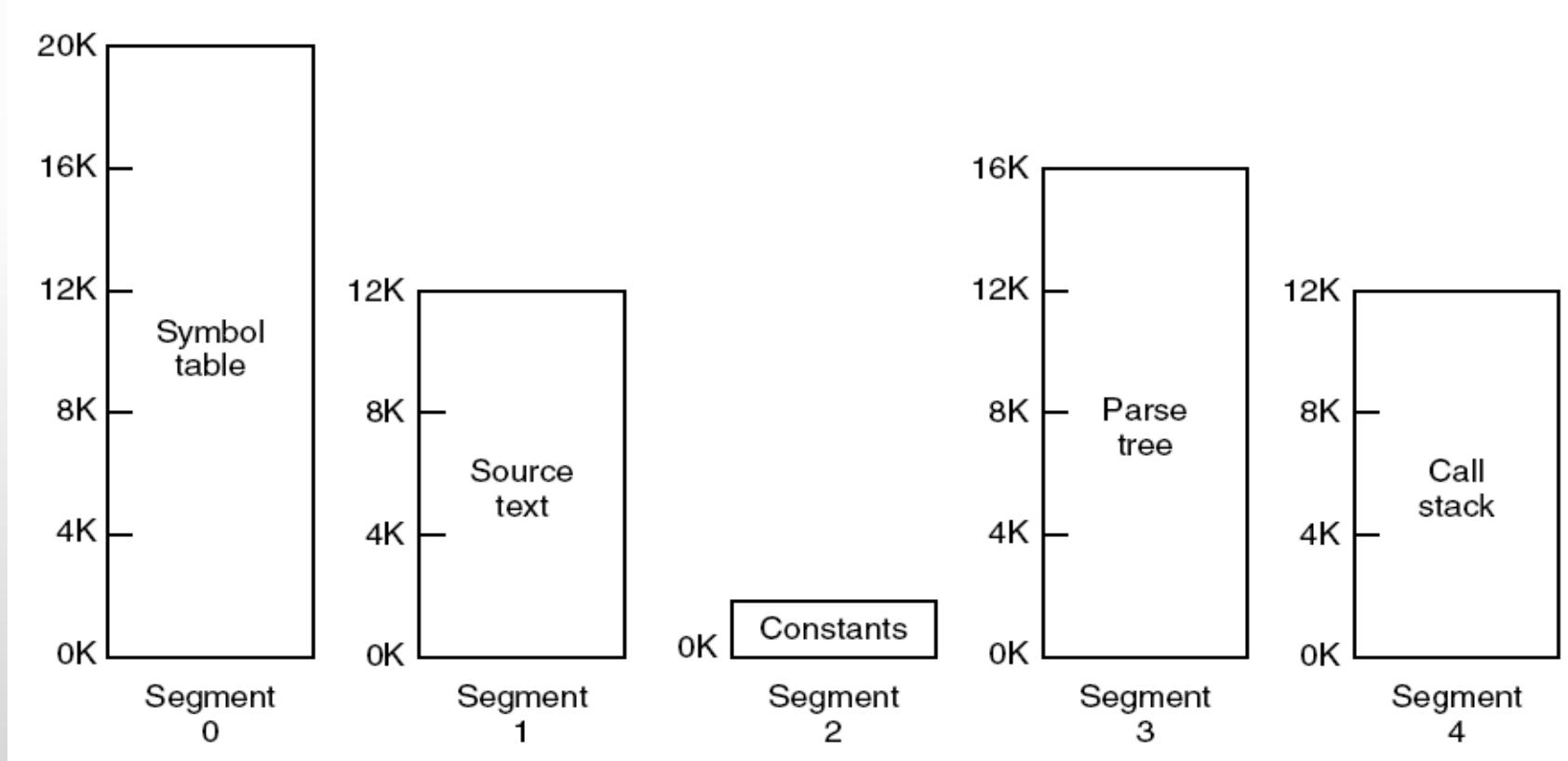
Kesimleme Avantajları

- Büyüyen ve küçülen veri yapılarının ele alınmasını basitleştirir
- Segment n'nin adres alanı (n , yerel adres) biçimindedir, burada $(n,0)$ başlangıç adresidir
- Segmentleri diğer segmentlerden ayrı olarak derleyebilir
- Kütüphaneyi bir segmente koyabilir ve paylaşabilir
- Farklı segmentler için farklı korumalara (r, w, x) sahip olabilir



Kesimleme (segmentation)

- bölümlere ayrılmış bellek





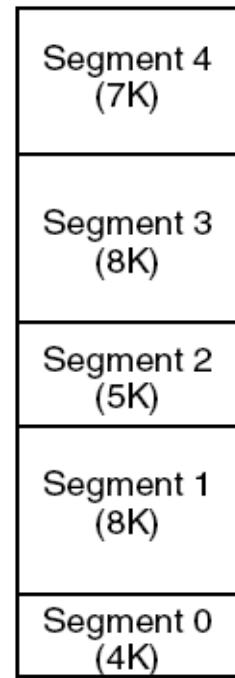
Sayfalama Ve Kesimleme Karşılaştırılması

Durum	Sayfalama	Kesimleme
Programcı bu tekniğin kullanıldığından 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

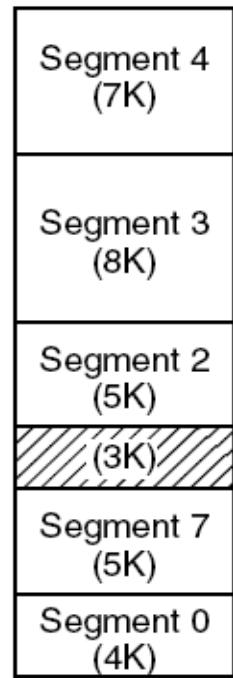


Sayfalama ile Kesimleme: MULTICS

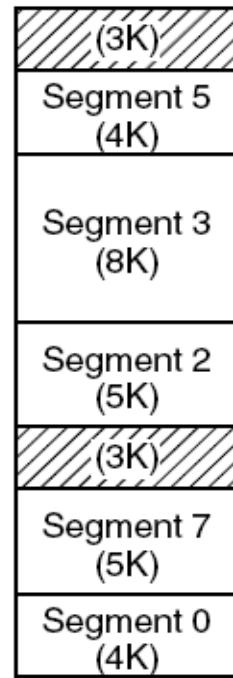
- (a)-(d) Dama tahtası deseni oluşması. (e) Sıkıştırma ile kaybolması.



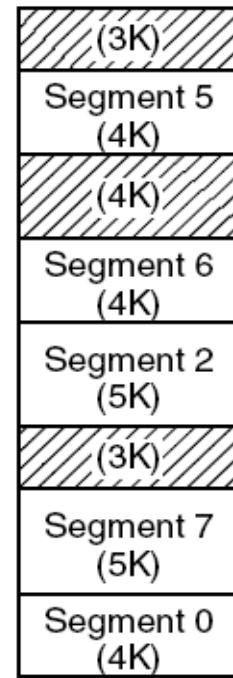
(a)



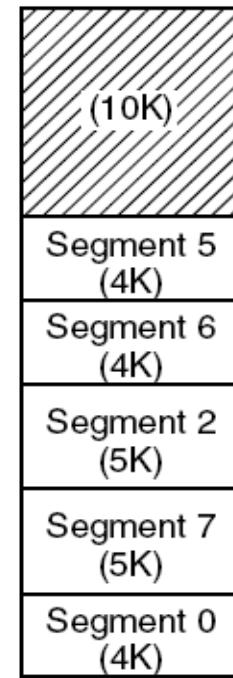
(b)



(c)



(d)

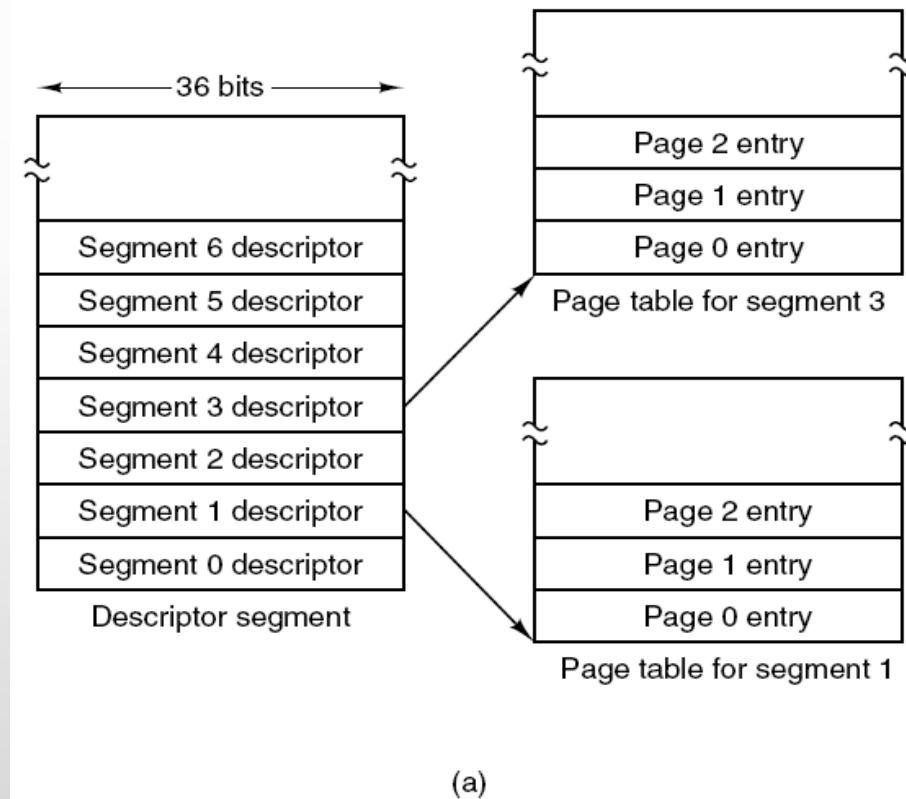


(e)



Sayfalama ile Kesimleme: MULTICS

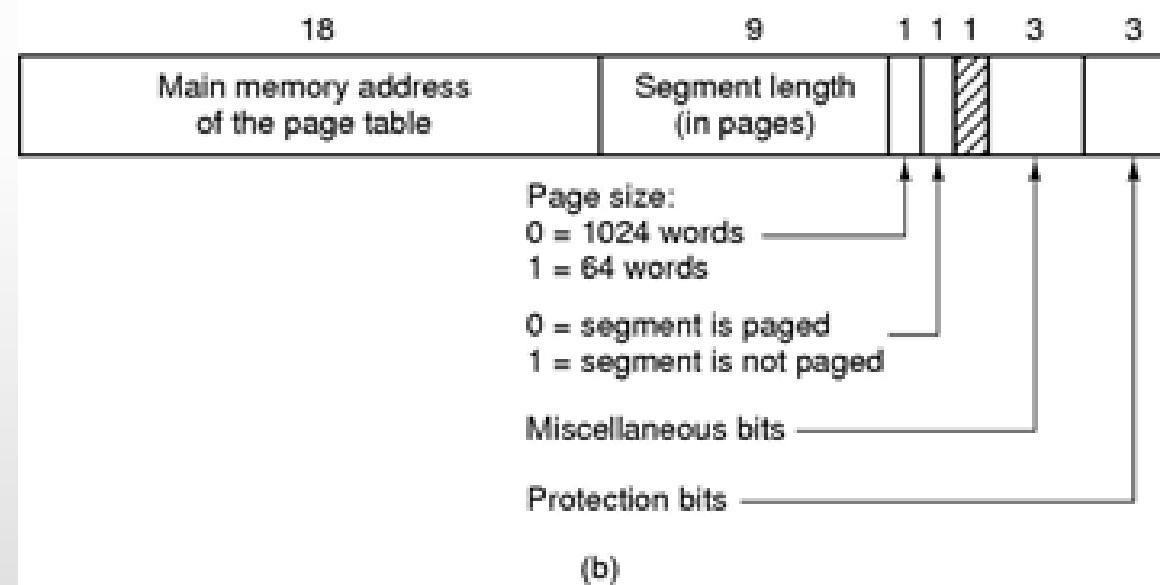
- MULTICS sanal belleği. (a) Tanımlayıcı bölüm, sayfa tablolarına işaret eder.





Sayfalama ile Kesimleme: MULTICS

- MULTICS sanal belleği. (b) Bir segment tanımlayıcısı. Sayılar alan uzunluklarıdır.





Sayfalama ile Kesimleme: MULTICS

- Bir bellek erişimi olduğunda, aşağıdaki algoritma çalıştırılır:
- Segment tanımlayıcısını bulmak için segment numarası kullanılır.
- Segmentin sayfa tablosunun bellekte olup olmadığı kontrol edilir.
- Değilse, segment hatası oluşur.
- Bir koruma ihlali varsa, bir hata (tuzak) oluşur.



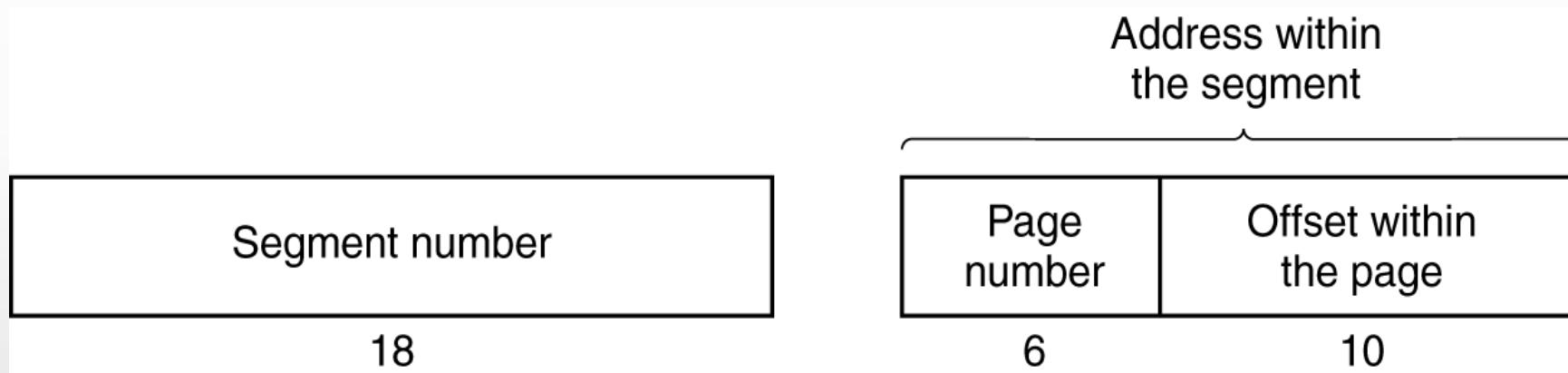
Sayfalama ile Kesimleme: MULTICS

- İstenen sanal sayfa için sayfa tablosu girdisi incelenir.
- Sayfanın kendisi bellekte değilse, bir sayfa hatası tetiklenir.
- Hafızada ise, sayfanın başlangıcının ana bellekteki adresi sayfa tablosu girdisinden okunur.
- Ofset, kelimenin bulunduğu ana bellek adresini vermek için sayfa adresine eklenir.
- Sonunda okuma veya saklama gerçekleşir.



Sayfalama ile Kesimleme: MULTICS

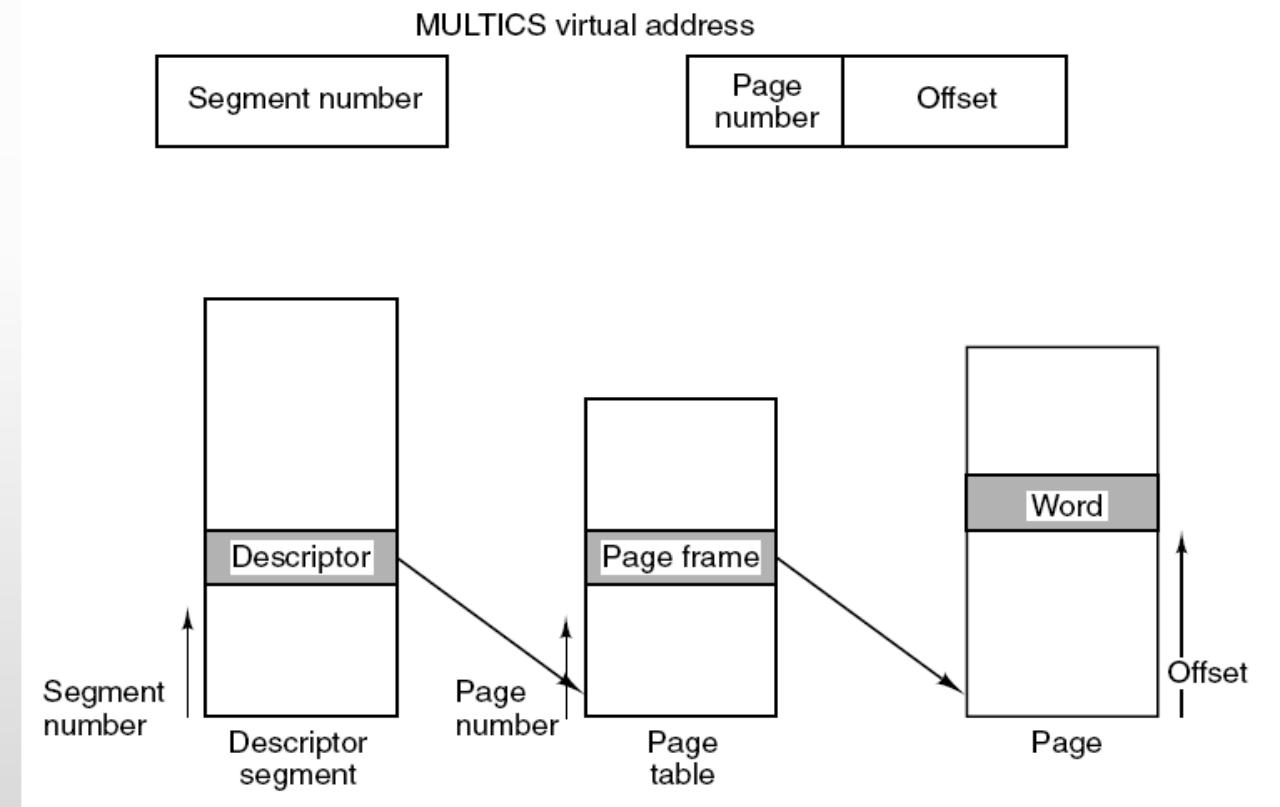
- 34 bit MULTICS sanal adresi.





Sayfalama ile Kesimleme: MULTICS

- İki parçalı bir MULTICS adresinin bir ana bellek adresine dönüştürülmesi.





Sayfalama ile Kesimleme: MULTICS

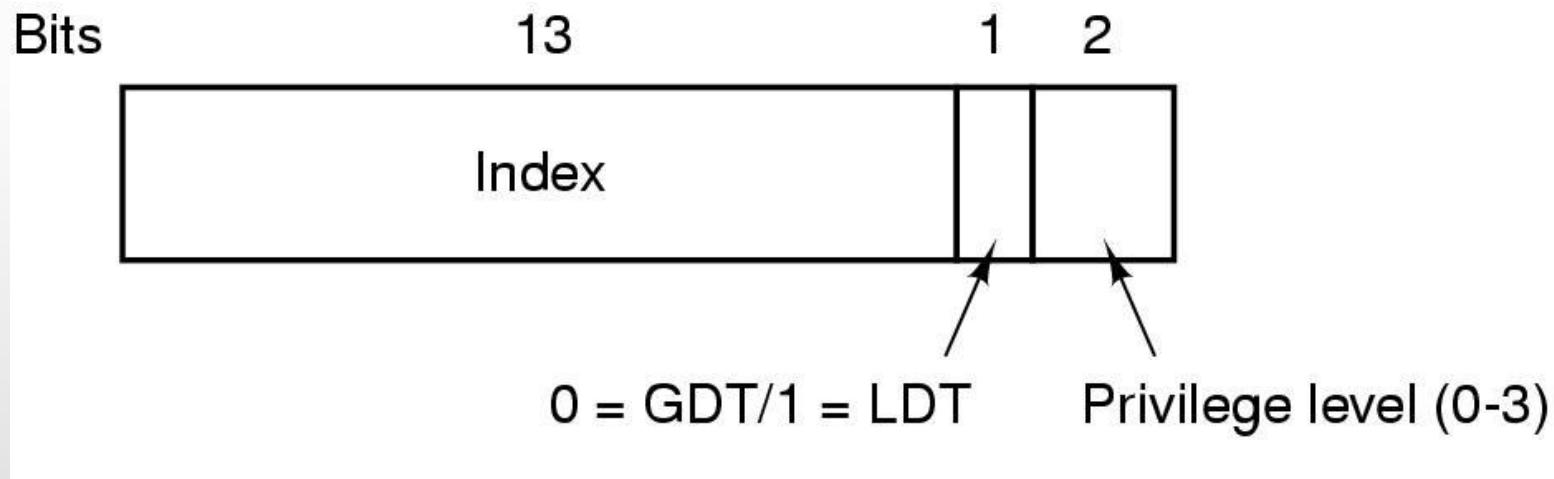
- MULTICS TLB'nin basitleştirilmiş bir versiyonu. İki sayfa boyutunun varlığı, gerçek TLB'yi daha karmaşık hale getirir.

Comparison field					Is this entry used?	
Segment number	Virtual page	Page frame	Protection	Age		
4	1	7	Read/write	13	1	
6	0	2	Read only	10	1	
12	3	1	Read/write	2	1	
					0	
2	1	0	Execute only	7	1	
2	2	12	Execute only	9	1	



Sayfalama ile Kesimleme: PENTIUM

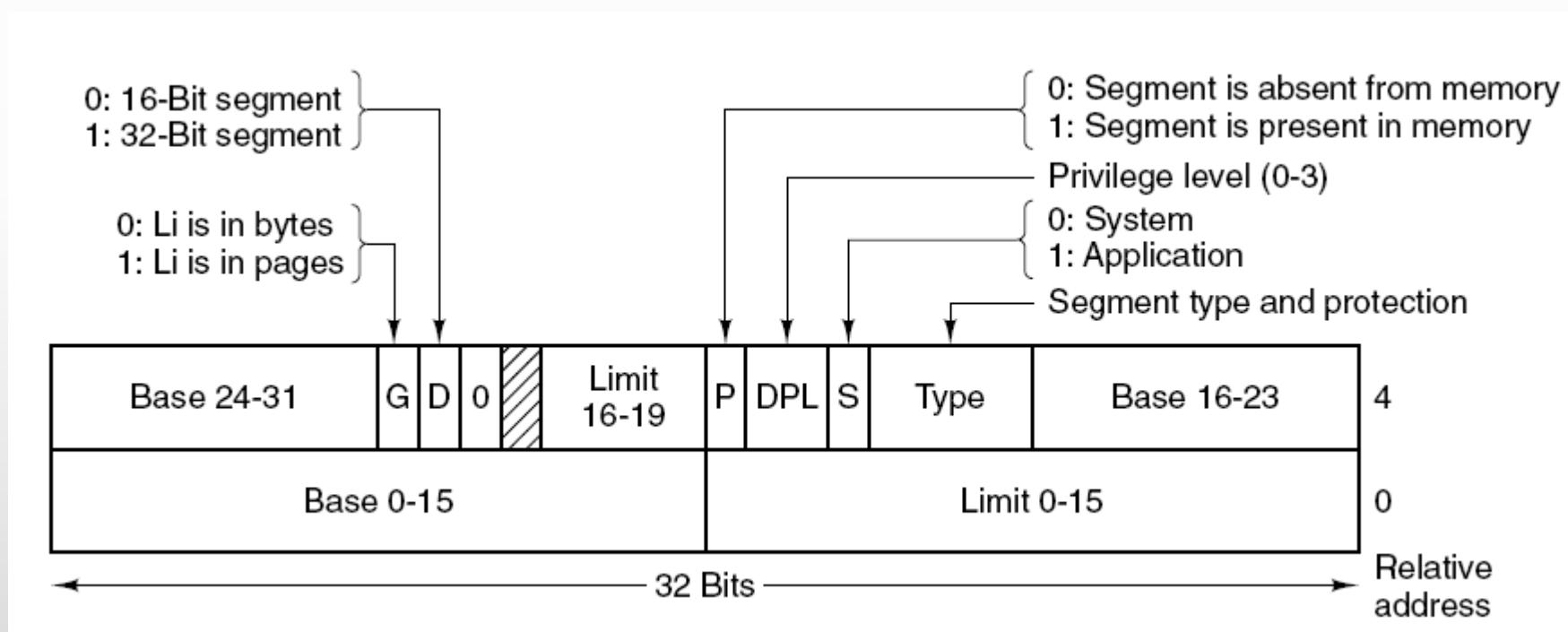
- Bir Pentium seçici.





Sayfalama ile Kesimleme: PENTIUM

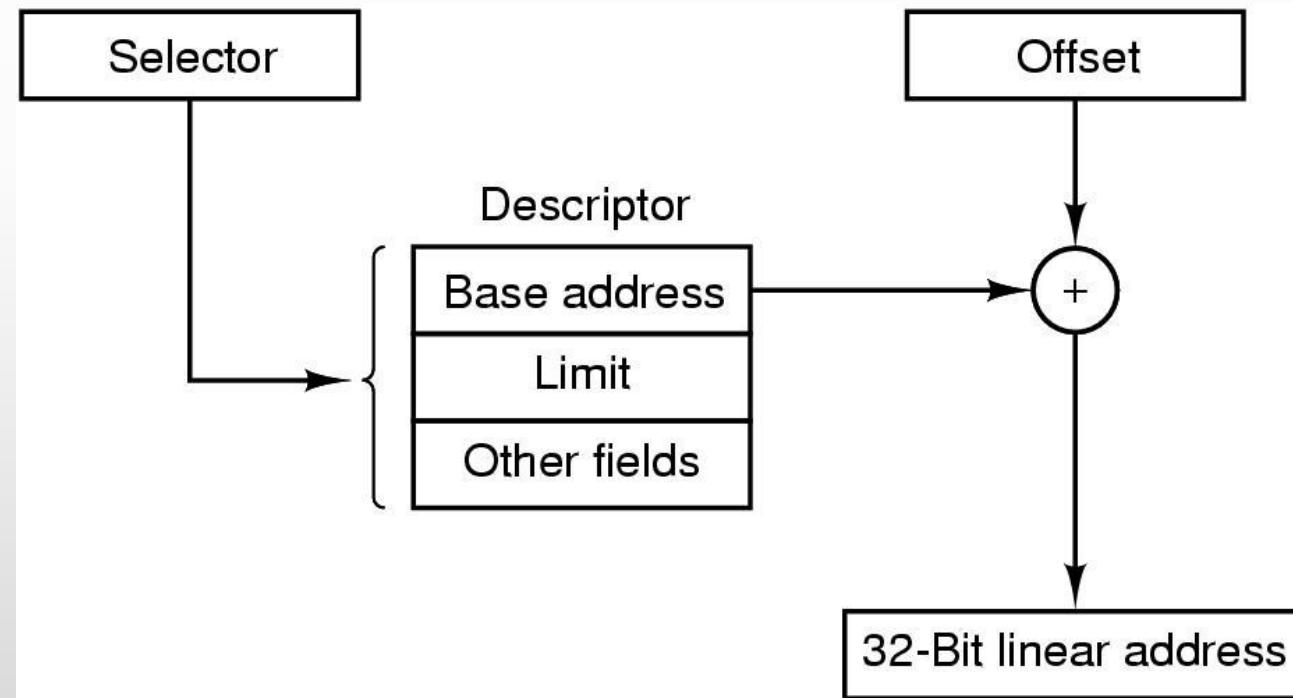
- Pentium kod segment tanımlayıcısı. Veri segmentleri biraz farklıdır.





Sayfalama ile Kesimleme: PENTIUM

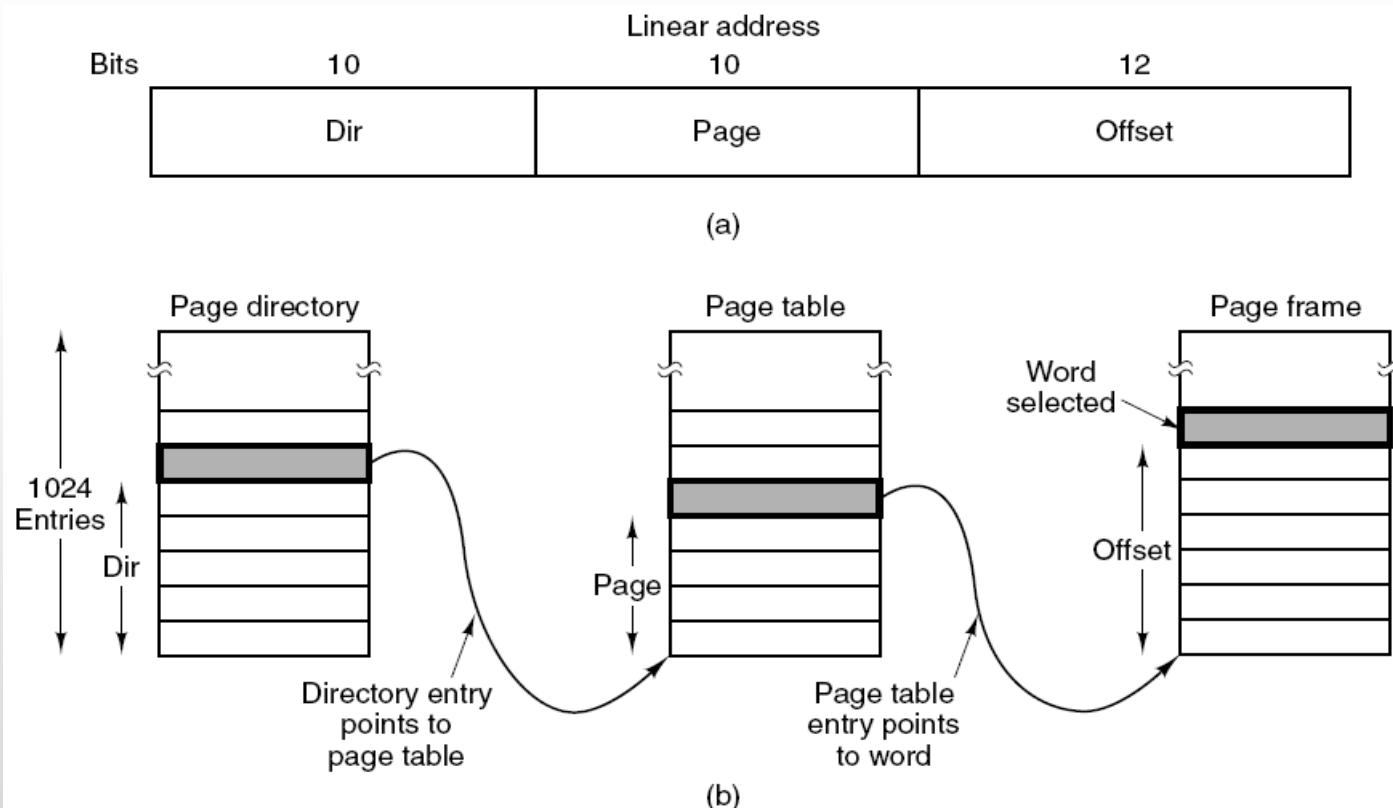
- Bir (seçici, ofset) çiftinin doğrusal bir adrese dönüştürülmesi.





Sayfalama ile Kesimleme: PENTIUM

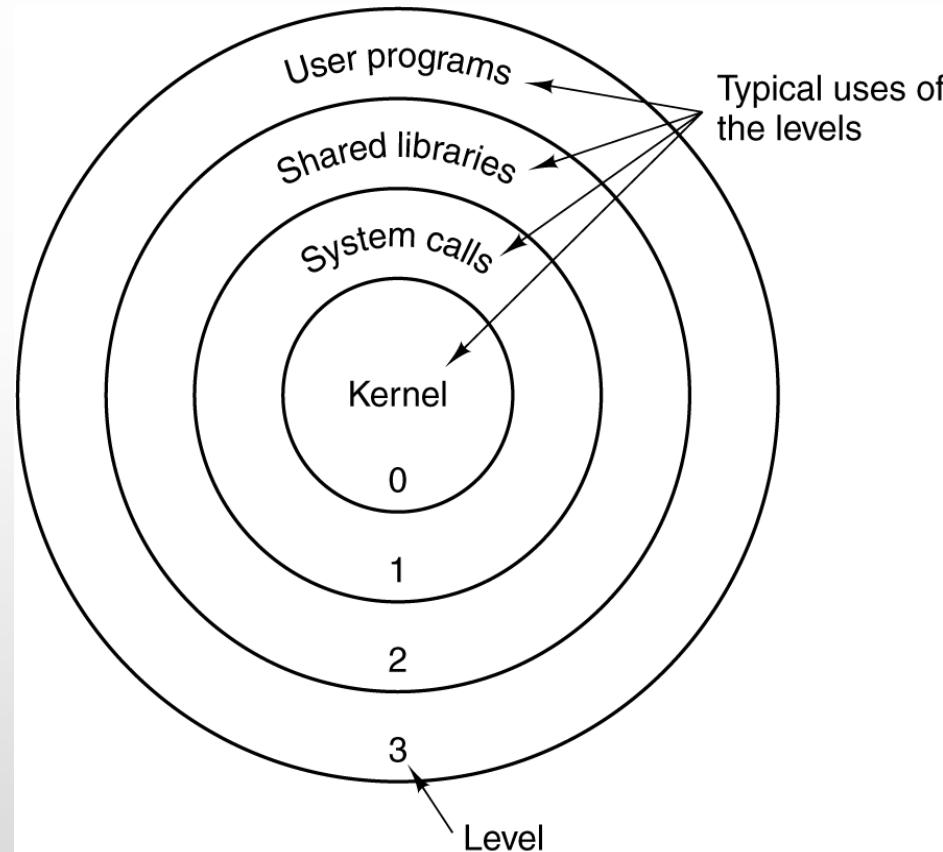
- Doğrusal bir adresin fiziksel bir adrese eşlenmesi.





Sayfalama ile Kesimleme: PENTIUM

- Pentium'da koruma.





SON