

Bölüm 8: Ana Bellek

Bölüm 8: Ana Bellek

- Arka Plan Bilgisi
- Yer Değiştirme (Swapping)
- Bitişik Bellek Yerleşimi (Contiguous Memory Allocation)
- Sayfalama (Paging)
- Sayfa Tablosunun (Page Table) Yapısı
- Bölütleme (Segmentation)
- Örnek: Intel Pentium

Hedefler

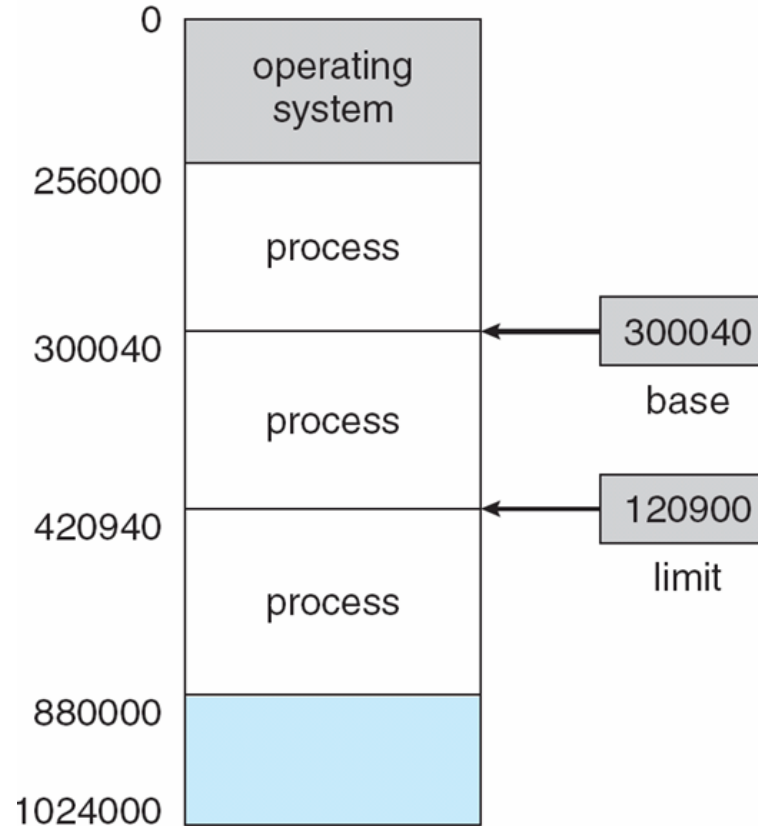
- Hafıza donanımının farklı şekillerde nasıl organize edilebileceğini detaylı bir şekilde anlatmak
- Sayfalama (**paging**) ve bölütleme (**segmentation**) gibi pek çok hafıza yönetim mekanizmasının açıklanması
- Hem saf bölütleme hem de sayfalama ile bölütleme desteği sunan Intel Pentium'un incelenmesi

Arka Plan Bilgisi

- Çalıştırılmak istenen program öncelikle diskten belleğe alınmalı ve bir işleme dönüştürülmelidir
- CPU'nun direk olarak erişebileceği kayıt birimleri yalnızca ana bellek (main memory) ve yazmaçlardır (registers)
- CPU'nun yazmaçlara erişimi bir CPU birim zamanı (veya daha az) sürer
- Ana belleğe erişim pek çok CPU birim zamanı sürebilir
- **Ön bellek (cache)** ana bellek ile CPU yazmaçları arasında yer almaktadır
- Ana belleğin korunması sistemin doğru çalışması için şarttır

Taban ve Sınır Yazmaçları

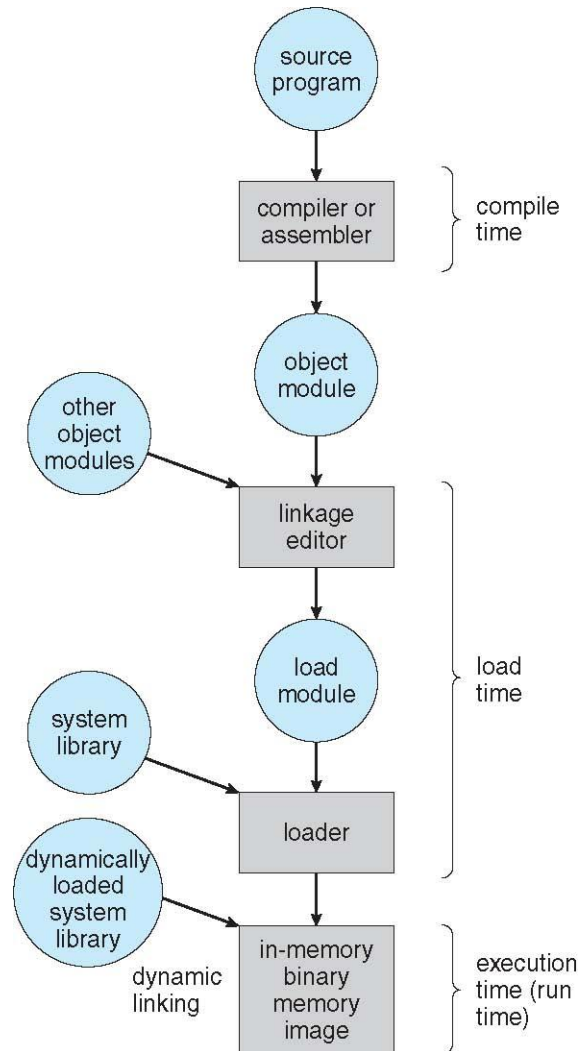
- **Taban (base)** ve **sınır (limit)** yazmaçları mantıksal adres uzayını tanımlar



Komutların ve Verilerin Hafızaya Bağlanması

- Programlardaki komutların ve verilerin adreslerinin hafızadaki adreslere bağlanması üç farklı aşamada gerçekleşebilir
 - **Derleme zamanı (compile time):** Eğer ana bellekteki pozisyon önceden biliniyorsa, **mutlak kod (absolute code)** oluşturulabilir. Başlangıç adresi değişirse kodun yeniden derlenmesi gerekir
 - **Yükleme zamanı (load time):** Eğer derleme zamanında ana bellek pozisyonu bilinmiyorsa, **yeniden yerleştirilebilir kod (relocatable code)** oluşturulmalıdır
 - **Çalışma zamanı (execution time):** Eğer işlem çalışırken bir hafıza bölümünden diğerine taşınabiliyorsa, bağlama çalışma zamanına kadar ertelenir
 - Adres haritaları (address maps) için donanım desteği gerekir (örn., taban ve sınır yazmaçları)

Kullanıcı Programının Çok Adımlı İşletimi



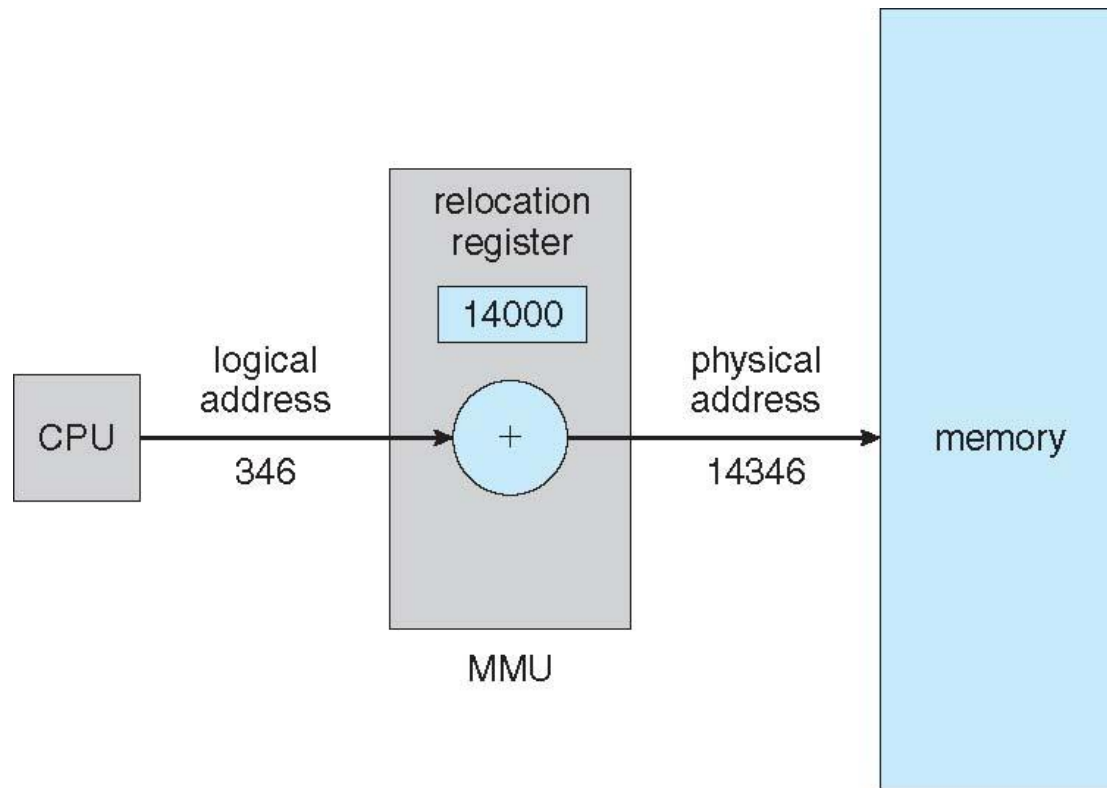
Mantıksal veya Fiziksel Adres Uzayı

- Mantıksal bir adres uzayının ayrı bir fiziksel adres uzayına bağlanması düzgün bir hafıza yönetimi için önşarttır
 - **Mantıksal adres** (logical address) –CPU tarafından oluşturulur. **Sanal adres** (virtual address) olarak da adlandırılır
 - **Fiziksel adres** (physical address) – hafıza birimi tarafından bilinen adrestir
- Derleme-zamanı ve Yükleme-zamanı adres bağlama yaklaşımlarında mantıksal ve fiziksel adresler aynıdır
- Çalışma zamanı adres bağlama yaklaşımında mantıksal ve fiziksel adresler farklılık gösterir

Hafıza Yönetim Birimi (MMU)

- Sanal adresleri fiziksel adreslere çeviren donanım
- MMU birimine kullanıcı programı tarafından gönderilen her adrese **yeniden yerleştirme yazmacındaki** (relocation register) değer eklenir
- Elde edilen adres hafızaya gönderilir
- Kullanıcı programları mantıksal adresleri kullanırlar; gerçek fiziksel adresleri *asla* görmezler

Yeniden Yerleştirme Yazmacını Kullanarak Dinamik Yeniden Yerleştirme



Dinamik Yükleme (Dynamic Loading)

- Metot çağırılmadığı sürece yüklenmez
- Daha iyi hafıza uzayı yönetimi; kullanılmayan metot asla yüklenmez
- Nadiren gerçekleşen durumlara karşılık gelen büyük miktarda kod olduğunda faydalı
- İşletim sisteminden özel destek gerektirmiyor – program tasarımına dikkat edilmesi yeterli

Dinamik Bağlama (Dynamic Linking)

- Bağlama çalışma zamanına kadar ertelenir
- Dinamik bağlama özellikle kütüphaneler için faydalıdır
- **Stub:** hafızadaki kütüphane metodunun yerini bulmakta kullanılan küçük kod parçası
- Stub kendini kütüphane metodunun adresi ile değiştirir ve metodu çalıştırır
- İşletim sistemi, kütüphane metodunun, işlemin bellek adresi içinde olduğunu kontrol etmelidir
- Bu mekanizma aynı zamanda **paylaşımlı kütüphane (shared libraries)** olarak bilinmektedir

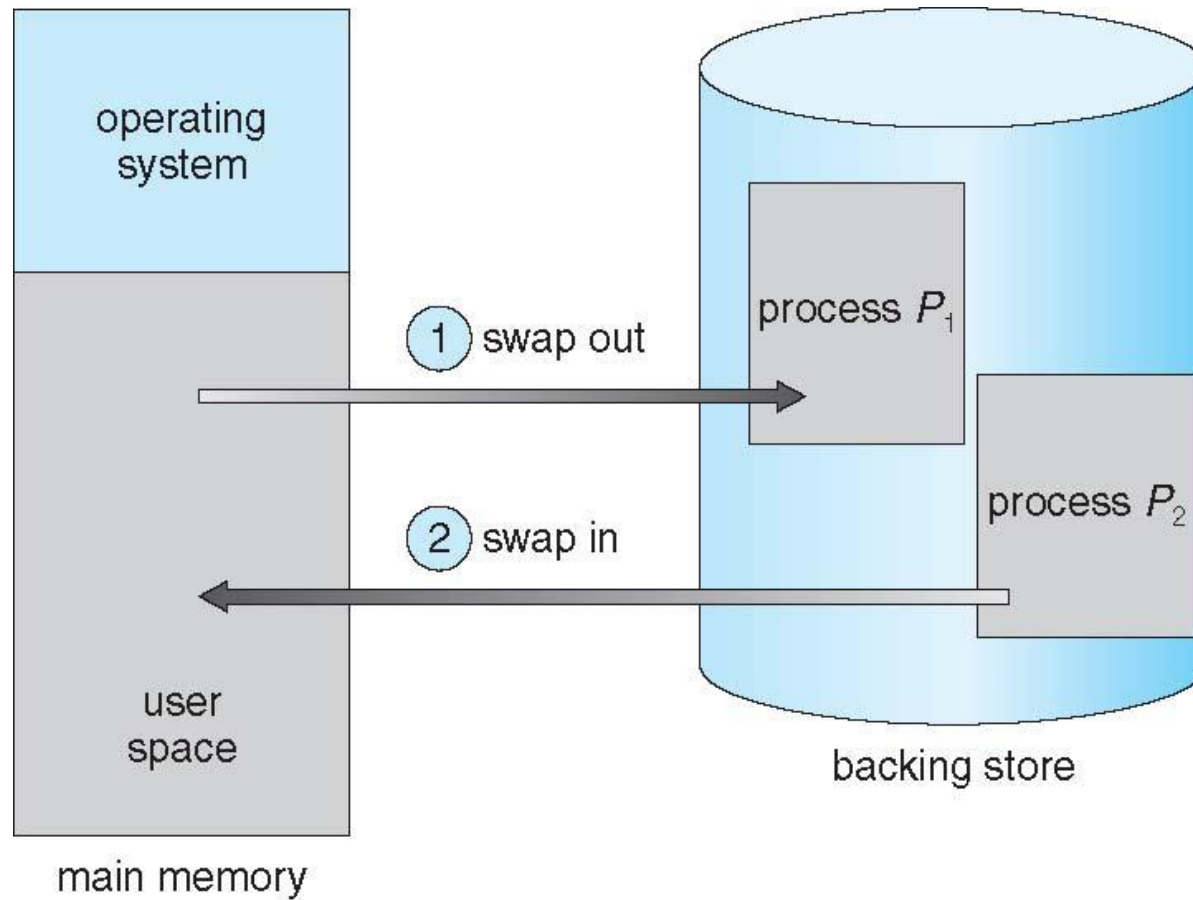
Yer Değiştirme (Swapping) (1/2)

- Bir işlem geçici olarak hafızadan çıkarılıp ikincil bir kayıt birimine alınabilir
- İşlem daha sonra çalışmaya devam etmek üzere yeniden ana belleğe alınabilir
- **Roll out, roll in** – öncelik tabanlı zamanlama algoritmaları tarafından kullanılan yer değiştirme varyasyonu
 - Düşük öncelikli işlemler, daha yüksek öncelikli işlemlerin yüklenip çalıştırılabilmesi için geçici olarak hafıza dışına alınır

Yer Değiştirme (Swapping) (2/2)

- Yer değiştirme zamanının önemli bir kısmı verinin transferinde kullanılır. Toplam transfer zamanı yer değiştirilen hafıza boyutu ile orantılıdır
- Yer değiştirmenin farklı varyasyonları pek çok işletim sisteminde bulunur (örn: UNIX, Linux ve Windows)
- Sistem çalışmaya hazır olan ancak kodu hafızada olmayan işlemleri **hazır kuyruğunda (ready queue)** tutar ve bu kuyruğu günceller

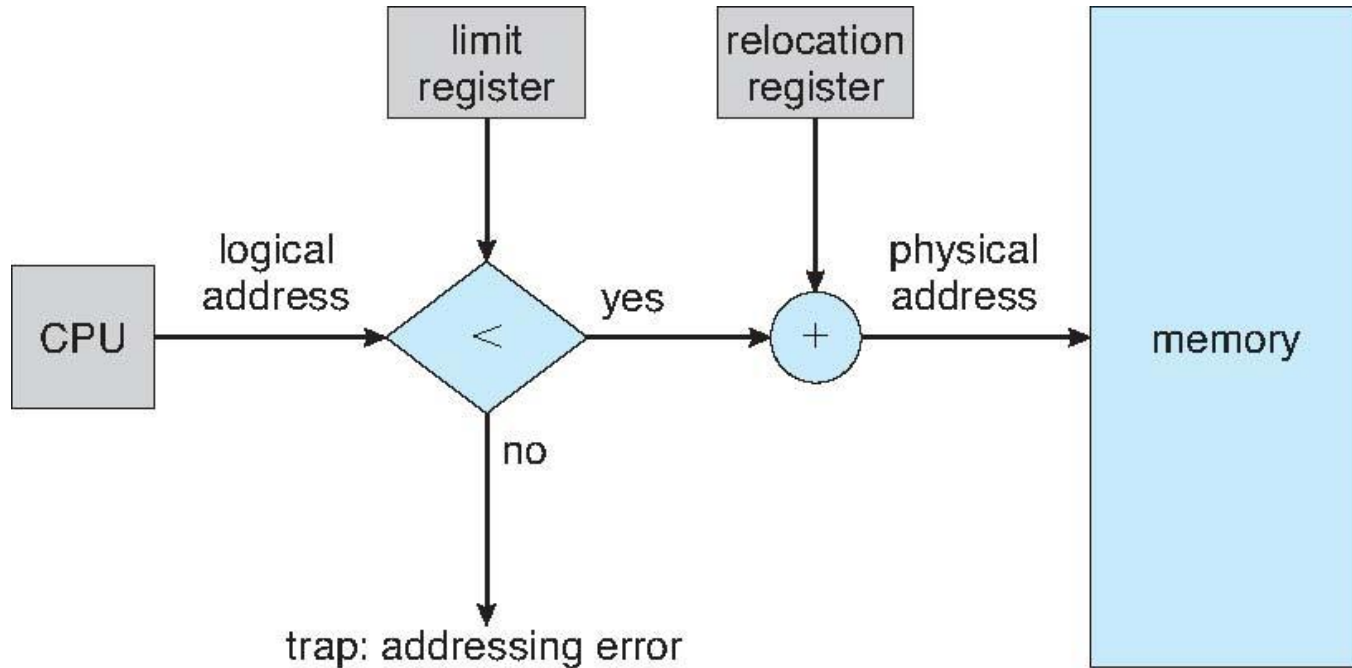
Yer Değiştirme



Bitişik (Contiguous) Bellek Yerleşimi

- Contiguous Memory Allocation
- Ana bellek genellikle iki bölüme ayrılır:
 - İşletim sistemi, genellikle kesinti vektörü ile birlikte hafızanın alt kısmında yer alır
 - Kullanıcı işlemleri, hafızanın üst kısmında yer alır
- Yeniden yerleştirme yazmaçları kullanıcı işlemlerini birbirinden korumak, işletim sistemi kodu ve verilerinin değiştirilmesini önlemek için kullanılır
 - Base register, en küçük fiziksel adresin değerini tutar
 - Limit Register, mantıksal adreslerin sınır değerini tutar – tüm mantıksal adresler limit registerında tutulan değerden daha küçüktür
 - MMU mantıksal adresleri *dinamik* olarak çevirir.

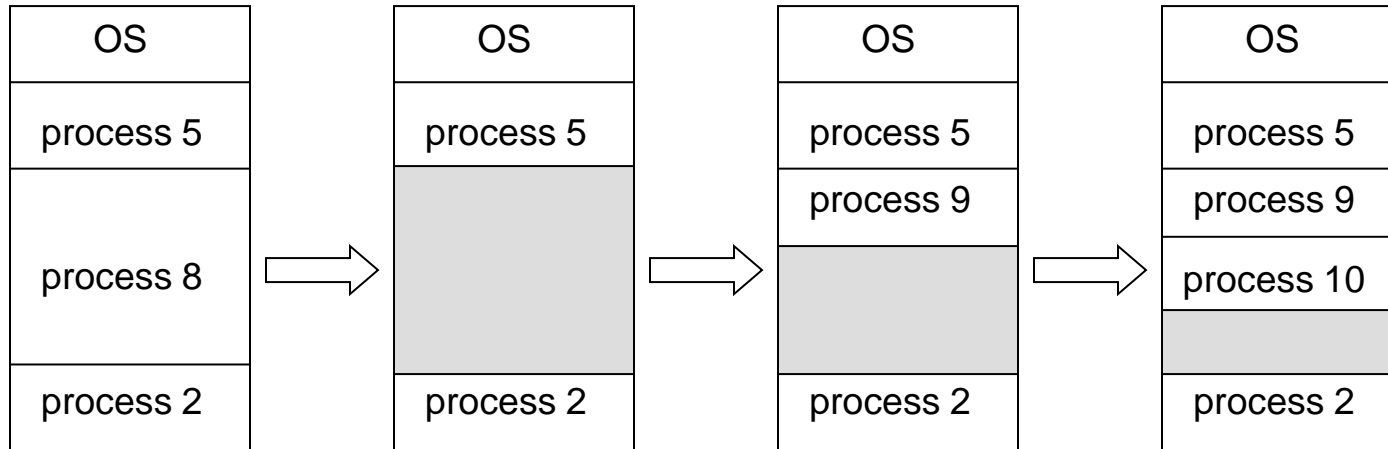
Yeniden Yerleştirme ve Sınır Yazmaçları için Donanım Desteği



Bitişik Bellek Yerleşimi (Devam)

- Çoklu-bölüm ayırımı

- Boşluk (Hole) – kullanılabilir bellek bloğu; hafıza üzerinde çeşitli boyutlarda boşluklar dağınık bir şekilde bulunur
- Bir işlem geldiğinde, bu işlemi tutabilecek kadar büyük bir hafıza boşluğuna yerleştirilir
- İşletim sistemi şu bilgileri tutup günceller:
 - a) işlemlere ayrılmış bölümler
 - b) boş bölümler (boşluklar)



Dinamik Kayıt Birimi Ayırım Problemi

Bir boş bölümler listesi elimizdeyken, n boyutunda bir işlemi nasıl bir boşluğa atayabiliriz?

- **İlk-uyum (first-fit)**: yeterince büyük olan *ilk* boşluğa ata
- **En-iyi-uyum (best-fit)**: yeterince büyük olan *en küçük* boşluğa ata
 - boyuta göre sıralı değilse, tüm liste aranmalıdır
 - geriye en küçük boşluğu bırakır
- **En-kötü-uyum (worst-fit)**: mevcut *en büyük* boşluğa ata
 - gene, tüm liste aranmalıdır
 - geriye en büyük boşluğu bırakır

İlk-uyum ve en-iyi-uyum, hız ve kayıt biriminin verimli kullanımı açısından en-kötü-uyum'a göre daha iyi sonuç verir

Parçalanma (Fragmentation) (1/2)

- **Dışsal Parçalanma (External Fragmentation)** – isteği karşılamak için hafıza alanı mevcut fakat bitişik değil
- **İçsel Parçalanma (Internal Fragmentation)** – işleme ayrılan hafıza gerekenden biraz fazla
 - boyut farkı işleme ayrılan hafıza alanında oluşuyor ve bu alan kullanılamıyor

Parçalanma (Fragmentation) (2/2)

- Dışal parçalanmayı **sıkıştırma (compaction)** ile azalt
 - Hafıza bloklarını, tüm boş blokları biraraya getirecek şekilde yeniden düzenle
 - Sıkıştırma sadece, yeniden yerleştirme dinamik ise, çalışma zamanında gerçekleştirilir
 - I/O problemi
 - İşlemler I/O gerçekleştirirken hafızadaki yerlerini sabitle
 - I/O işlemlerini sadece işletim sistemine ait tampon bellekler üzerinde gerçekleştir

Sayfalama (Paging) (1/2)

- Sayfalama, bir işleme ayrılan fiziksel adres uzayının bitişik olmamasına (noncontiguous) izin veren bir hafıza yönetim şeklidir
- Fiziksel hafıza sabit büyüklükte bloklara, **çerçevelere (frames)**, bölünür (belirlenen boyut 2'nin katları halindedir, 512 bayt ile 8,192 bayt arası)
- Mantıksal hafıza alanı da aynı boyutta bloklara, **sayfalara (pages)**, bölünür
- Tüm boş çerçeveler takip edilir: **çerçeve tablosu (frame table)**
- **n** sayfalık bir programı çalıştırmak için, n tane boş çerçevenin bulunması ve bu çerçevelere programın yüklenmesi gerekir
- Mantıksal adreslerin fiziksel adreslere dönüştürülmesi için bir **sayfa tablosuna (page table)** ihtiyaç duyulur

Sayfalama (Paging) (2/2)

- Dışsal parçalanma yok fakat içsel parçalanma var
- İçsel parçalanmayı azaltmak için, sayfa boyutu küçültülmelidir
- Sayfa tablosuna her erişim sistem kaynaklarını kullanmayı gerektirdiğinden, sayfa tablosuna erişim sayısını azaltmak için sayfa boyutu olabildiğince büyük olmalıdır
- Ayrıca, disk I/O işlemleri genellikle daha büyük miktarlarda veri transfer edildiğinde daha verimli çalışır
- Ödünleşme (tradeoff)
- İşlem boyutları arttıkça, sayfa boyutları da zamanla artmıştır
- Günümüzde tipik olarak sayfa boyutları 4KB ile 8KB arasındadır
- Bazı sistemler çok daha büyük sayfa boyutlarını desteklerler
- Solaris: 8KB ve 4MB

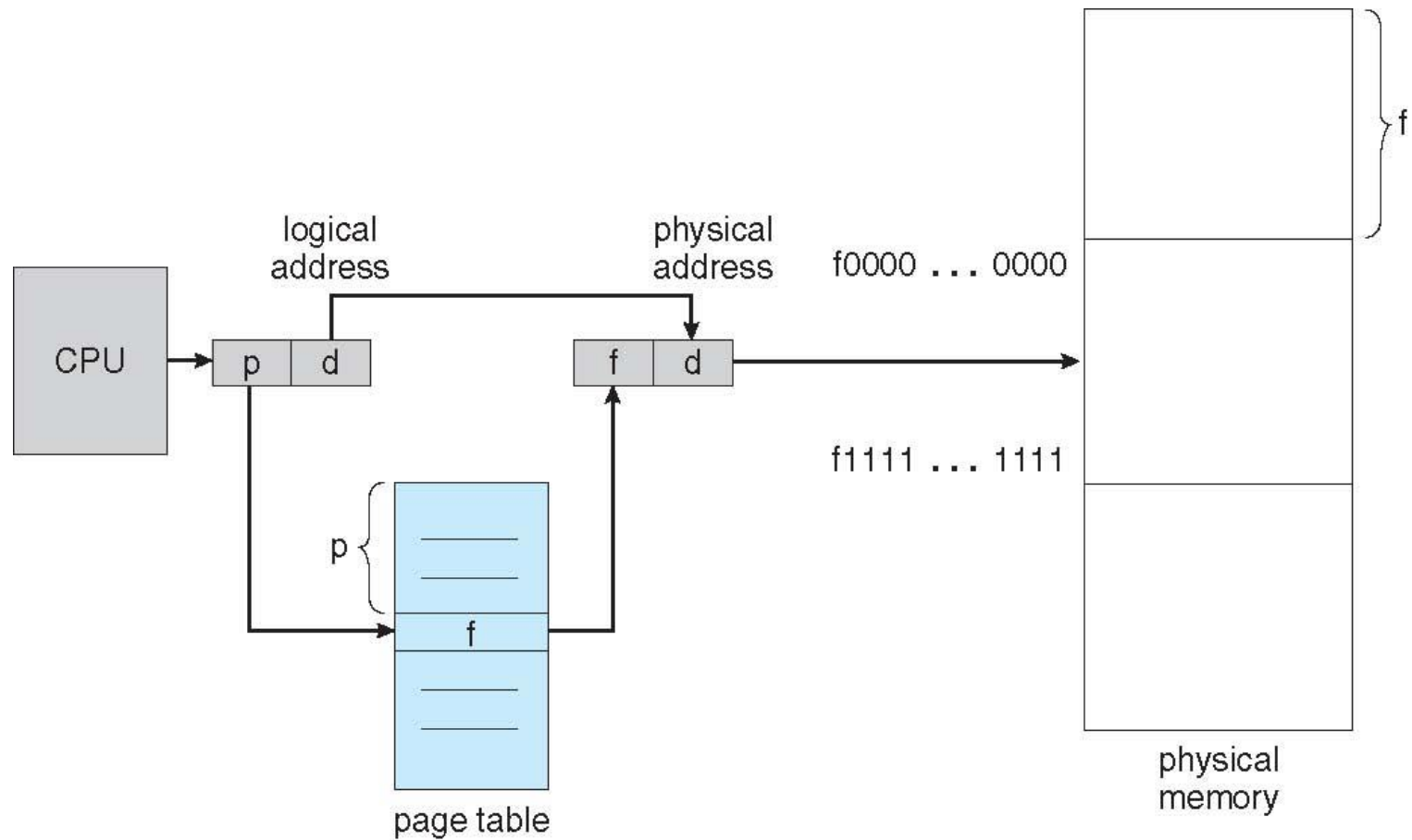
Adres Dönüşüm Mekanizması

- CPU tarafından üretilen adresle ikiye bölünür:
 - **Sayfa numarası (page number) (p)** – fiziksel hafızadaki her bir sayfanın temel adresini içeren sayfa tablosunun indeksi olarak kullanılır
 - **Sayfa ofseti (page offset) (d)** – temel adres ile birleştirilerek hafıza birimine gönderilecek fiziksel hafıza adresi elde edilir

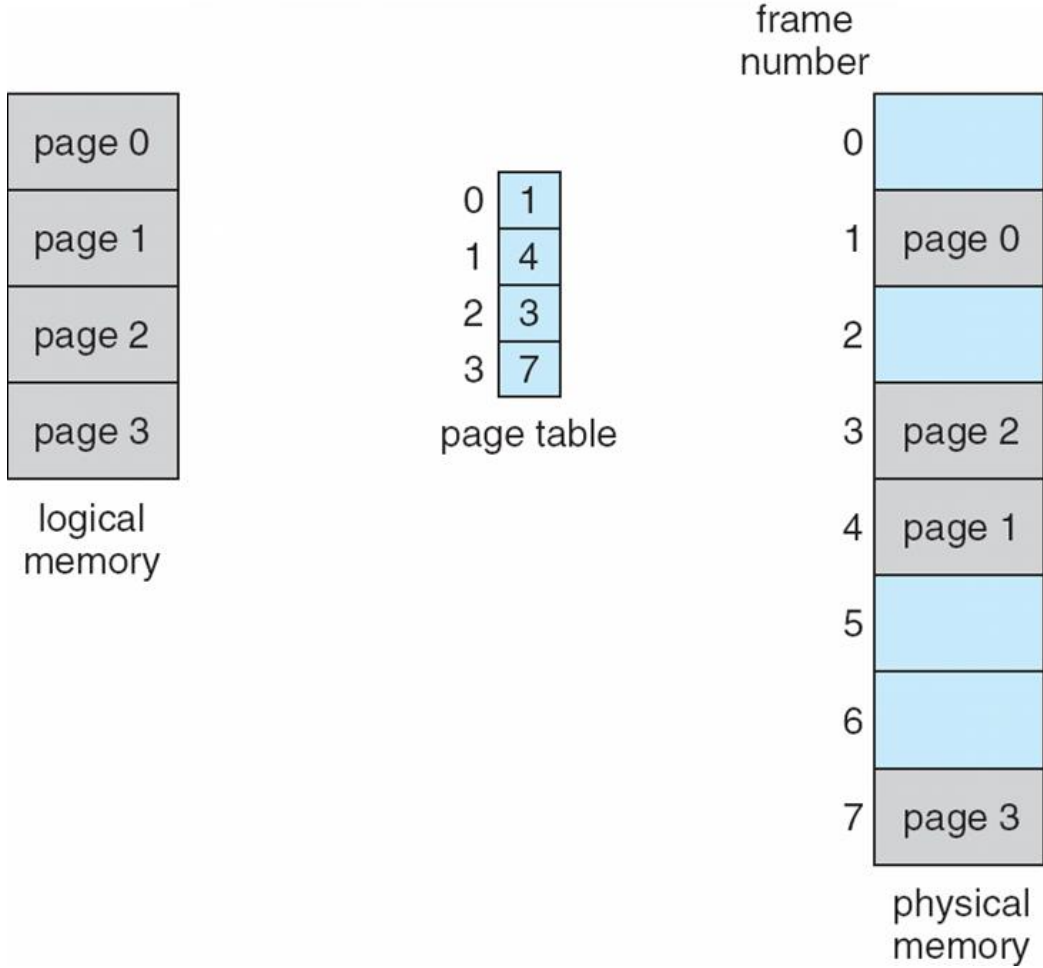
page number	page offset
p	d

- 2^m mantıksal adres uzayı ve 2^n sayfa boyutu ile
 $m - n$ n

Sayfalama Donanımı



Mantıksal ve Fiziksel Hafızanın Sayfalama Modeli



Sayfalama Örneği

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

logical memory

0	5
1	6
2	1
3	2

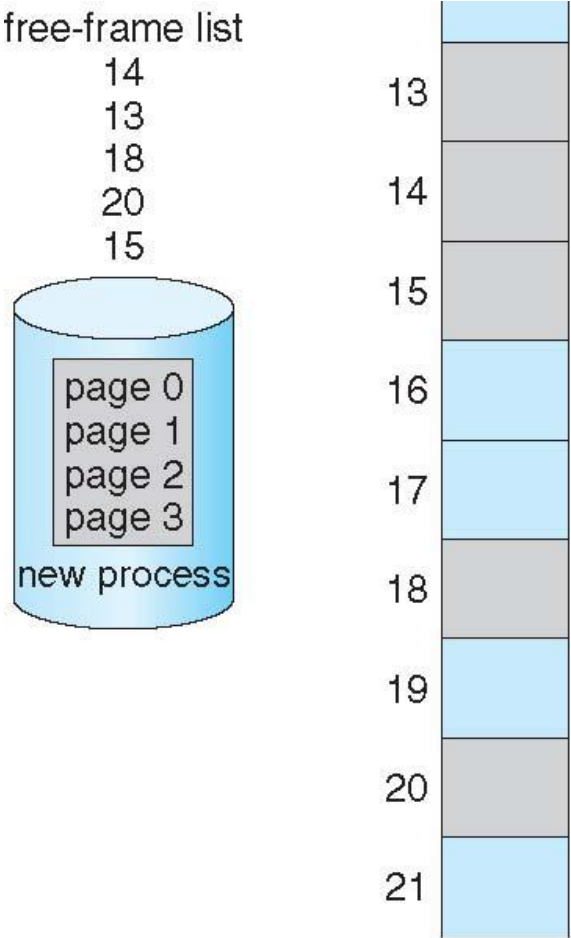
page table

0	
4	i j k l
8	m n o p
12	
16	
20	a b c d
24	e f g h
28	

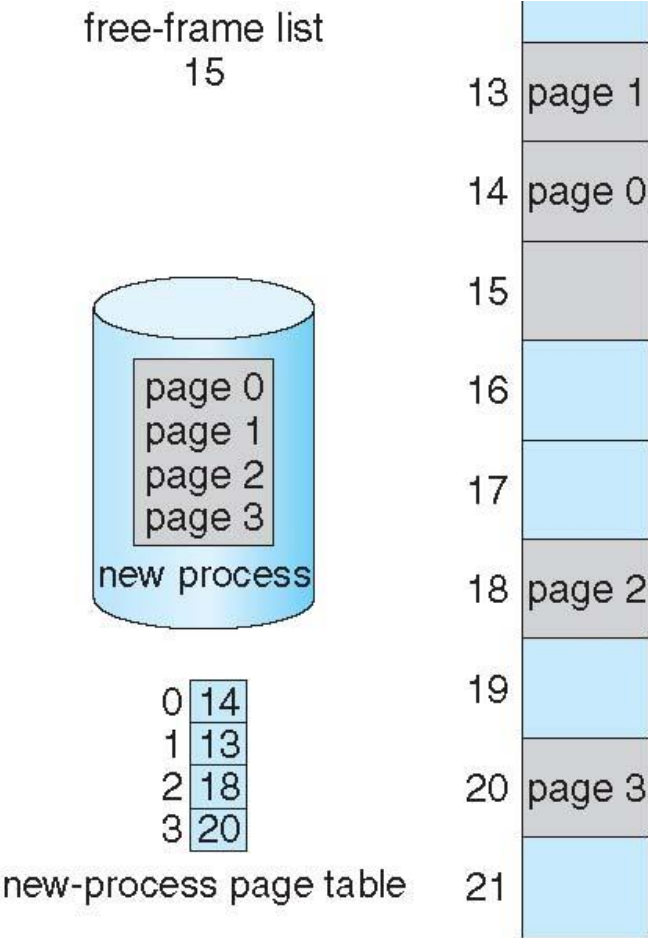
physical memory

32 bayt hafıza ve 4 baytlık sayfalar ($n=2$, $m=4$): 8 sayfa

Boş Çerçeveler (Free Frames)



(a)
Ayrım öncesi



(b)
Ayrım sonrası

Sayfa Tablosunun Gerçekleştirimi

- Sayfa tabloları hafızada tutulur
- Pek çok işletim sistemi her bir işlem için ayrı bir sayfa tablosu tutar ve tablonun adres detayları kayıtçı değerleri (register value) olarak işleme ait PCB'nin içerisinde saklanır
- **Sayfa tablosu temel sayacı - Page-table base register (PTBR)**
 - Sayfa tablosunun başlangıç adresini tutar
- **Page-table length register (PRLR)**
 - Sayfa tablosunun boyutunu tutar
- Bu yöntemde her tür veri veya komut erişimi hafızaya iki defa erişmeyi gerektirir. Bir erişim sayfa tablosu için, diğer erişim veri veya komuta erişim için

Sayfa Tablosunun Gerçekleştirimi

- Hafızaya iki kere erişim problemi, hızlı erişimi sağlayan özel önbellekler sayesinde çözülebilir:
 - çağrışımlı bellek (associative memory)
 - veya translation look-aside buffers (TLBs)
- TLB'deki her bir satır iki bileşenden oluşur: anahtar ve değer
- TLB üzerinde belirli bir sorgu yapıldığında, sorgulanan anahtar aynı anda tüm satırlardaki anahtarlar üzerinde aranır
- Sorgulanan anahtar TLB'lerde sayfa numarasıdır ve değer olarak çerçeve numarası döndürülür
- Arama hızlı ve donanım pahalı. Tipik olarak TLB'deki giriş sayısı 64 ve 1024 arasında

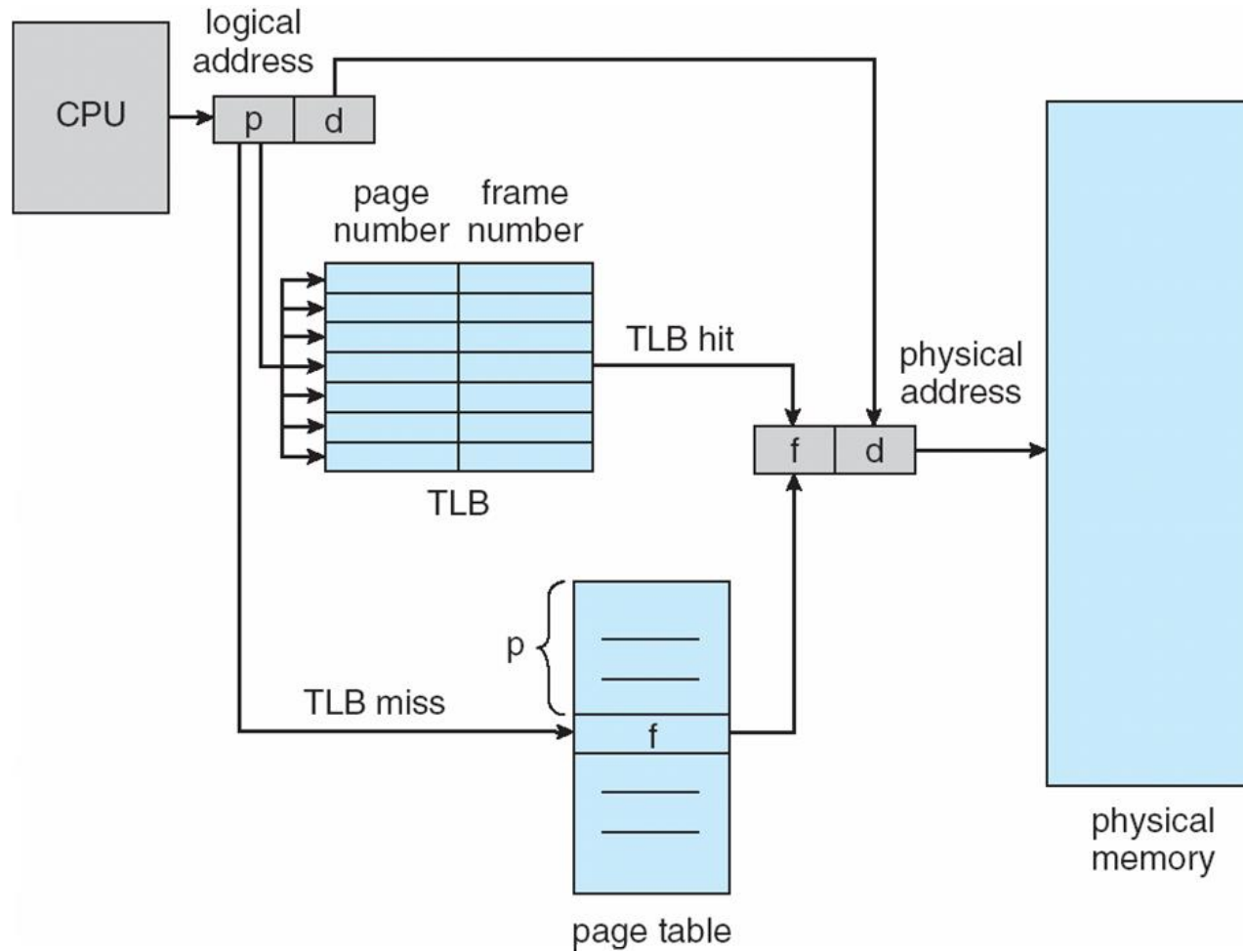
Çağrışımlı Bellek

- Çağrışımlı bellek:

Page #	Frame #

- Adres dönüşümü (p, d)
 - Eğer sayfa numarası (page #) çağrışımlı bellekte bulunduysa, çerçeve numarasını (frame #) al
 - Bulunmazsa, çerçeve numarasını hafızada tutulan sayfa tablosundan elde et
- Bazı TLB'ler her bir girişte **adres uzayı belirleyicilerini** de tutar – Böylece farklı işlemlere ait adres uzayları birbirlerinden korunmuş olur: **address-space identifiers (ASIDs)**

TLB ile Sayfalama Donanımı



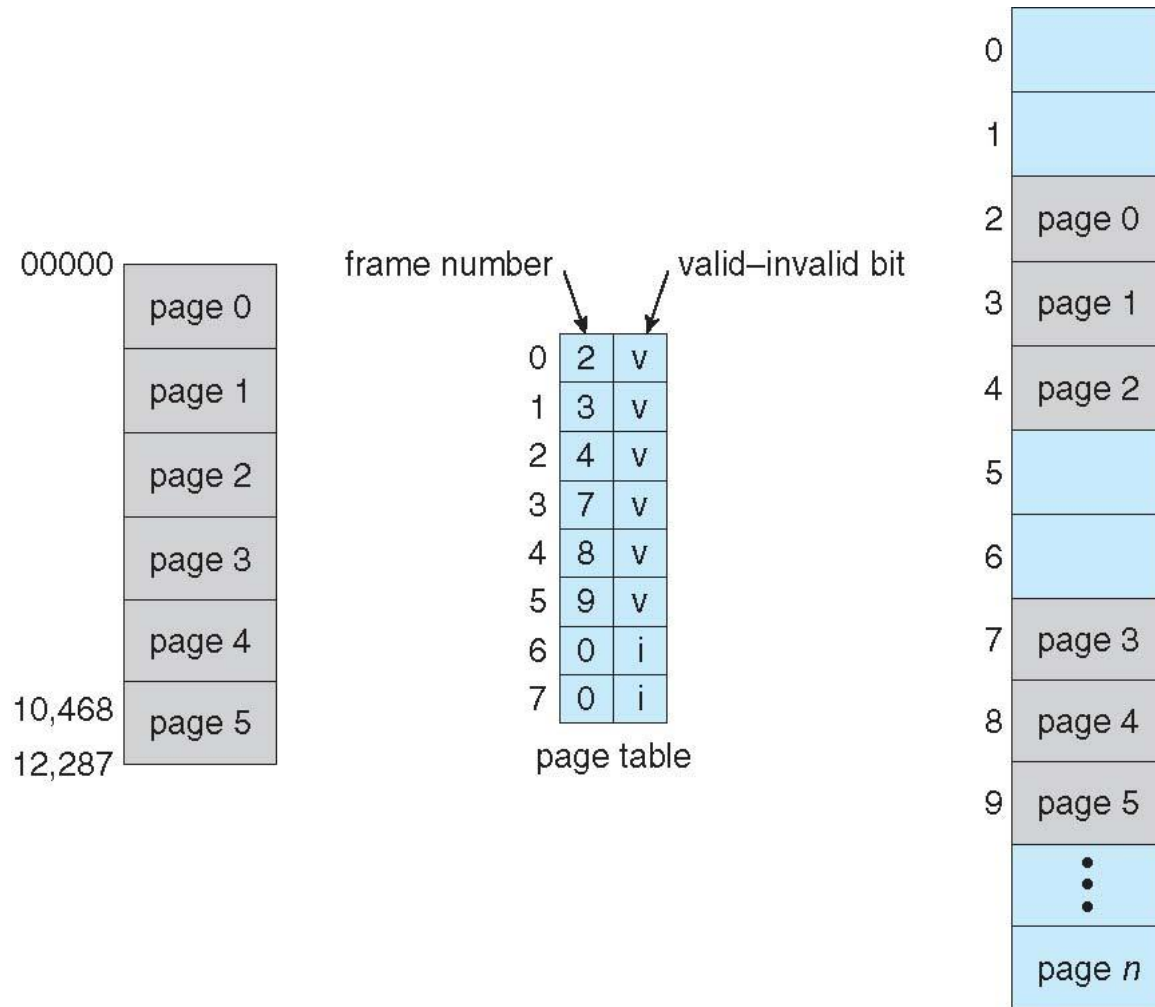
Geçerli Erişim Zamanı

- Effective Access Time
- Çağrışımlı arama (associative lookup) = ε mikrosaniye
- Hafıza erişim süresi: 1 mikrosaniye
- **Yakalama Oranı (hit ratio)** – bir sayfanın çağrışımlı bellekte bulunma olasılığı. Belirli sayıda erişim için, sayfanın çağrışımlı bellekte toplam bulunma sayısı bölü toplam erişim sayısı
- Yakalama oranı = α
- **Effective Access Time (EAT)**
$$\begin{aligned} \text{EAT} &= (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\ &= 2 + \varepsilon - \alpha \end{aligned}$$

Hafıza Koruma

- Hafıza koruması, her bir tablo girişi ile bir koruma bitinin ilişkilendirilmesi ile mümkündür
- **Geçerli-Geçersiz biti (Valid-invalid bit):**
 - “geçerli” (valid), ilişkili sayfanın işlemin mantıksal adres uzayında olduğunu ve erişim hakkının geçerli olduğunu gösterir
 - “geçersiz” (invalid), ilişkili sayfanın işlemin mantıksal adres uzayında olmadığını gösterir

Sayfa Tablosundaki Geçerli (v) veya Geçersiz (i) Bitler



Paylaşımlı Sayfalar

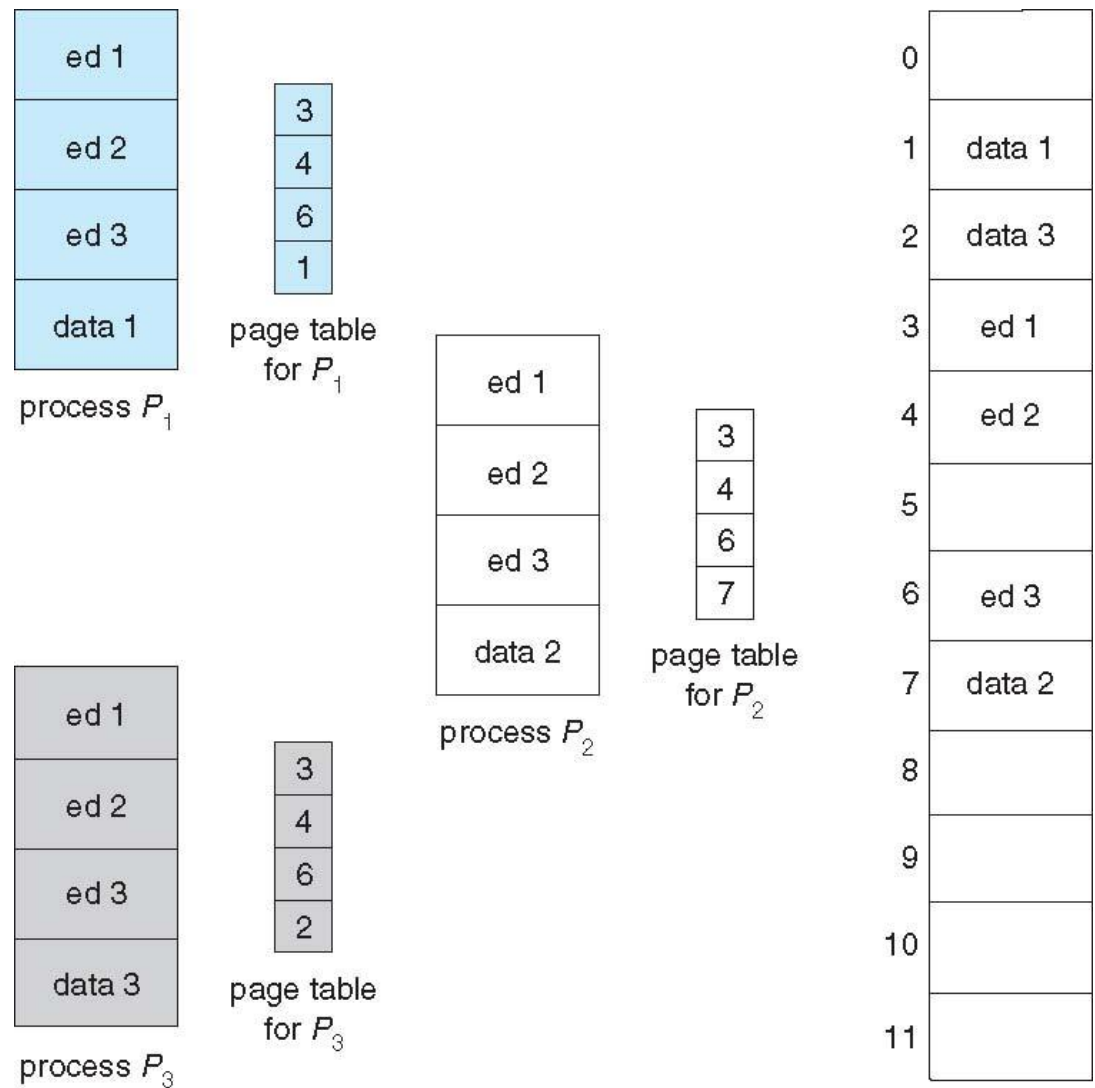
- **Kod paylaşımı**

- Sadece okunabilir olan kodun bir kopyası işlemler arasında paylaştırılır (örn: metin editörleri, derleyiciler, pencere sistemleri)
- Paylaşılan kod, tüm işlemlerin mantıksal adres uzaylarında aynı konumda bulunmalıdır

- **Özel kod ve veri**

- Her bir işlem kod ve verilerin ayrı bir kopyasını tutarlar
- Özel kod ve veriye ait sayfalar mantıksal adres uzayının herhangi bir yerinde bulunabilir

Paylaşımlı Sayfa Örneği



Sayfa Tablosunun Yapısı

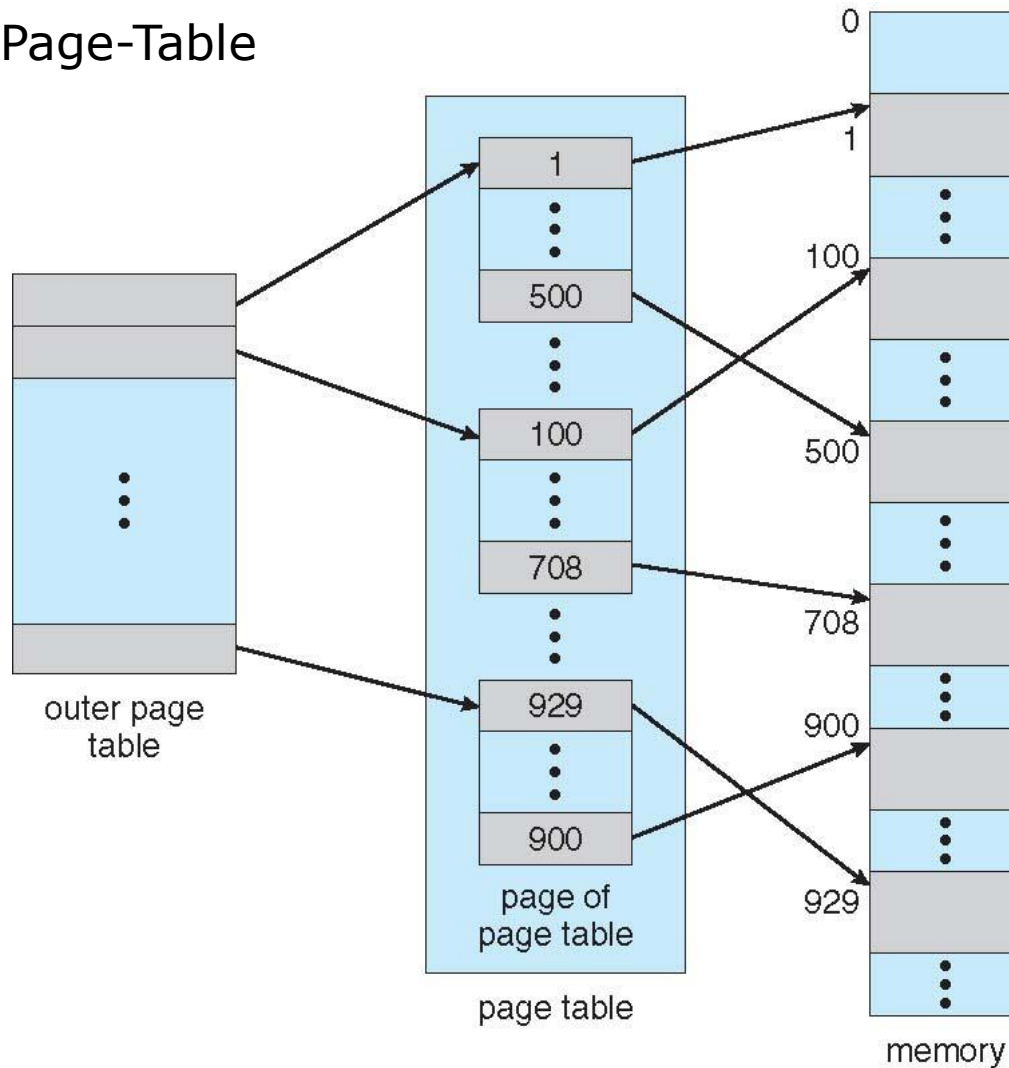
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hiyerarşik Sayfa Tabloları

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

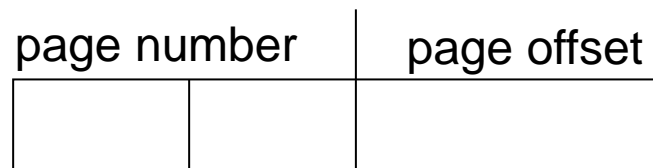
İki Seviyeli Sayfa Tabloları

Two-Level Page-Table



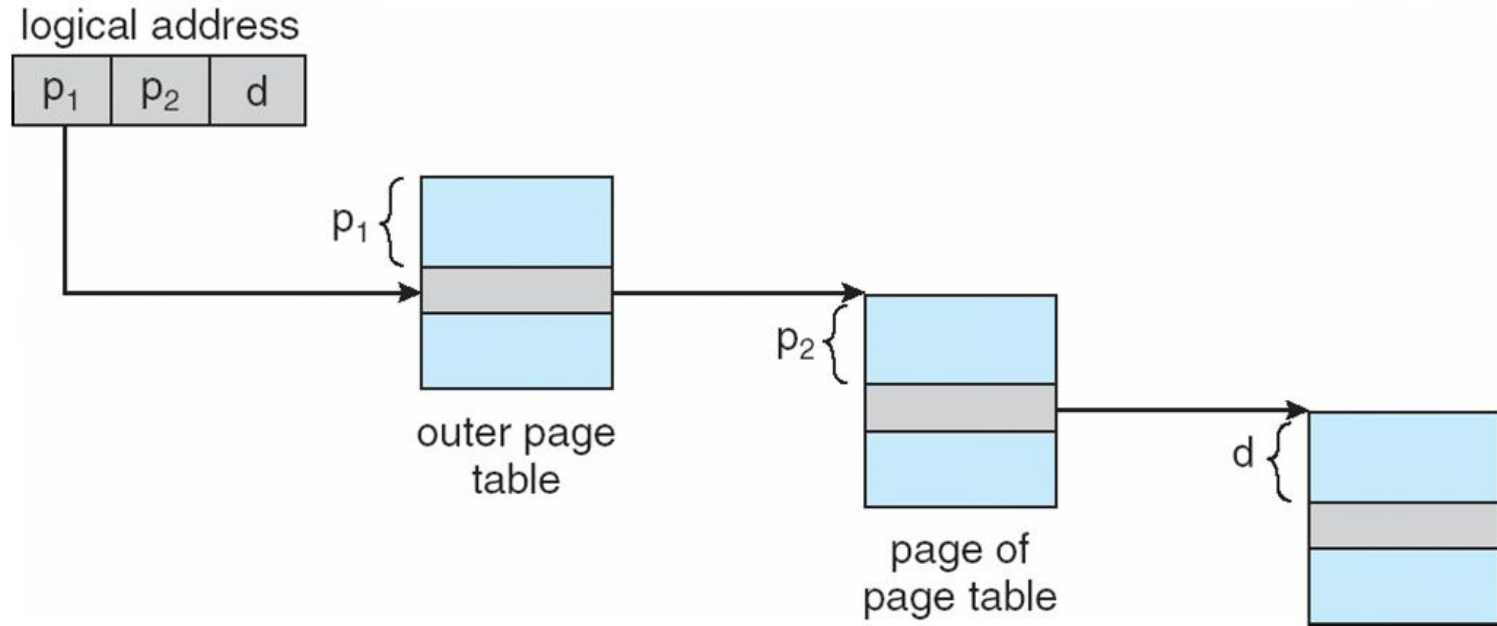
İki Seviyeli Sayfalama Örneği

- A logical address (on 32-bit machine with 1K page size) is divided into:
 - a page number consisting of 22 bits
 - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
 - a 12-bit page number
 - a 10-bit page offset
- Thus, a logical address is as follows:



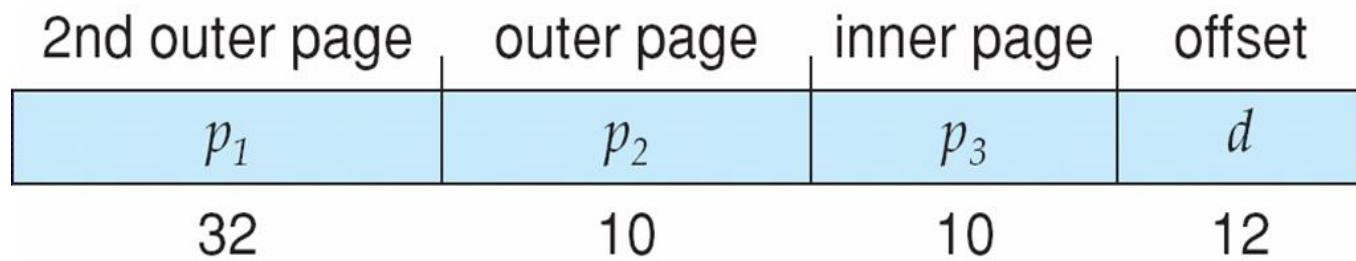
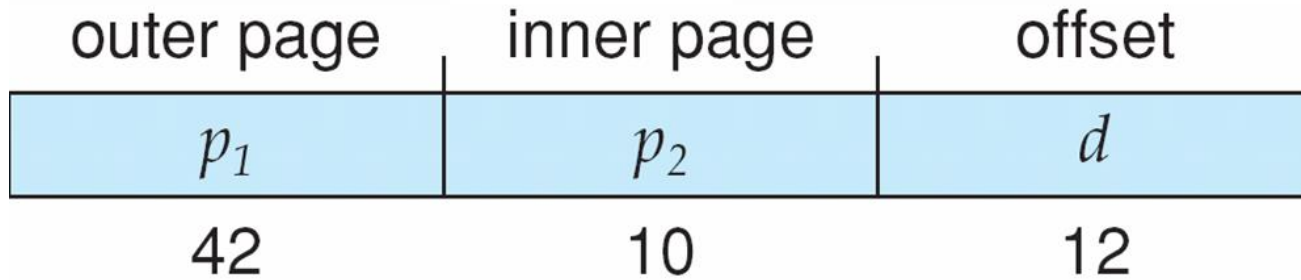
where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Adres Çevrimi



Üç Seviyeli Sayfalama

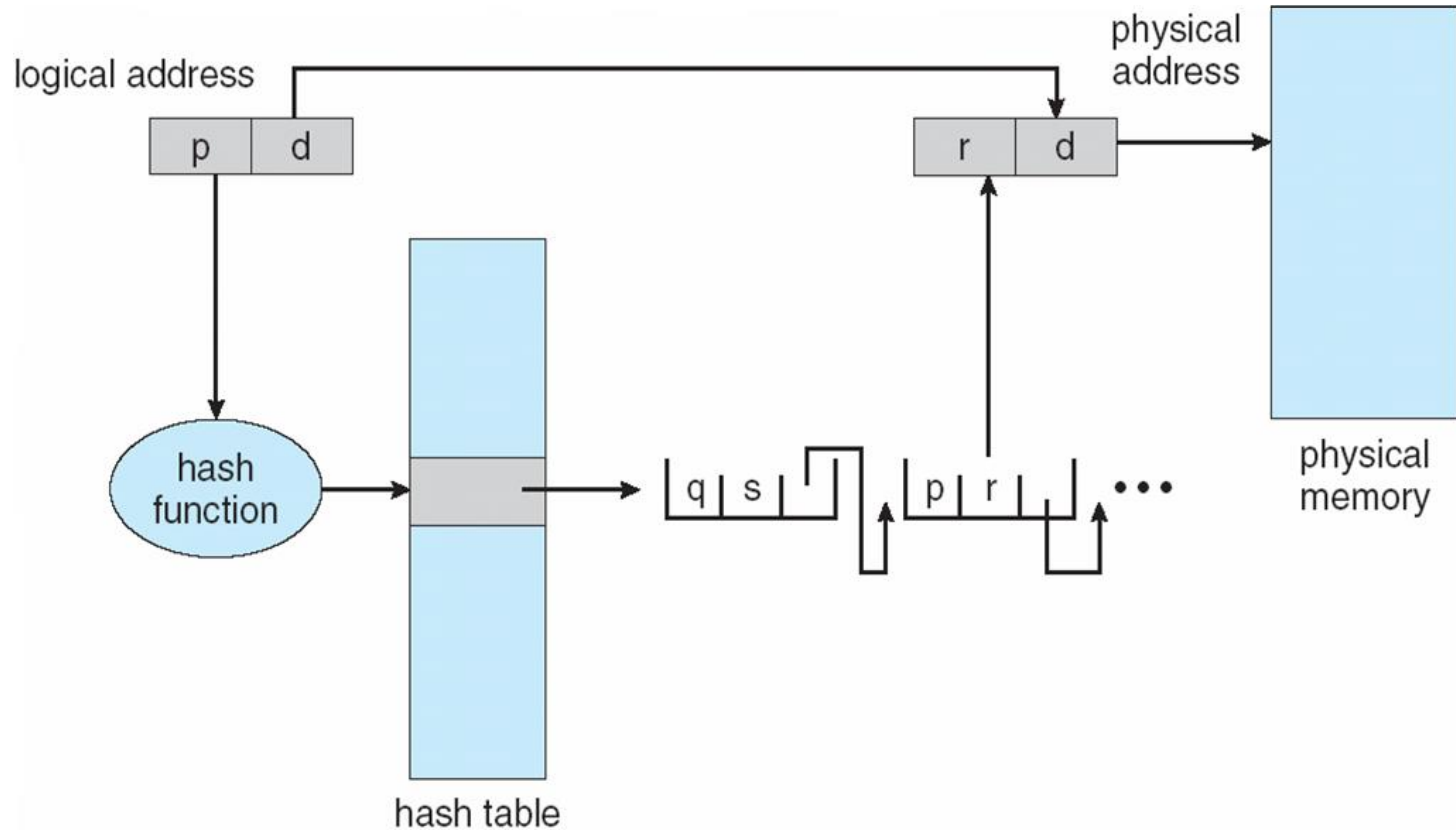
Three-level Paging



Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table.
 - This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match.
 - If a match is found, the corresponding physical frame is extracted.

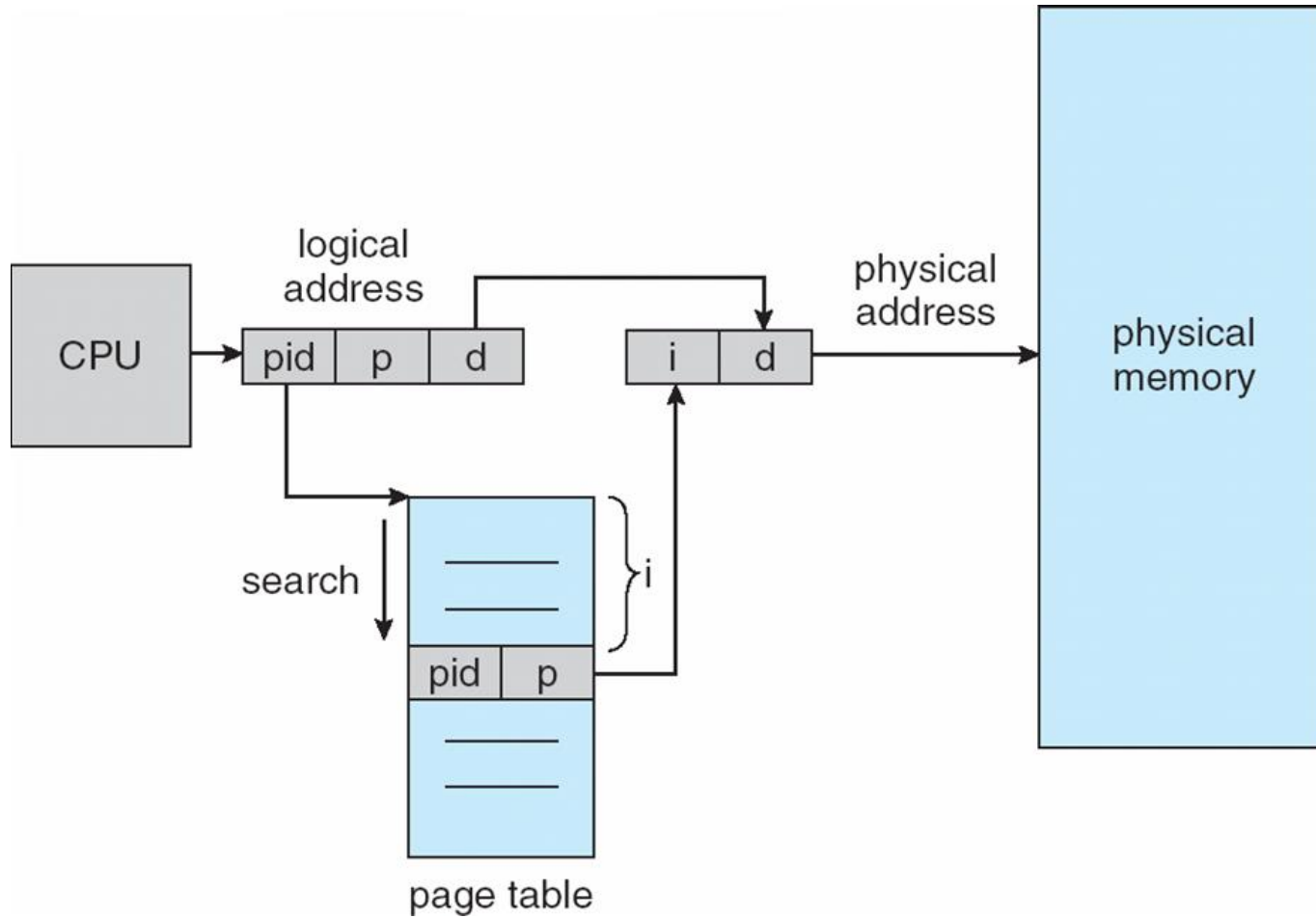
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.

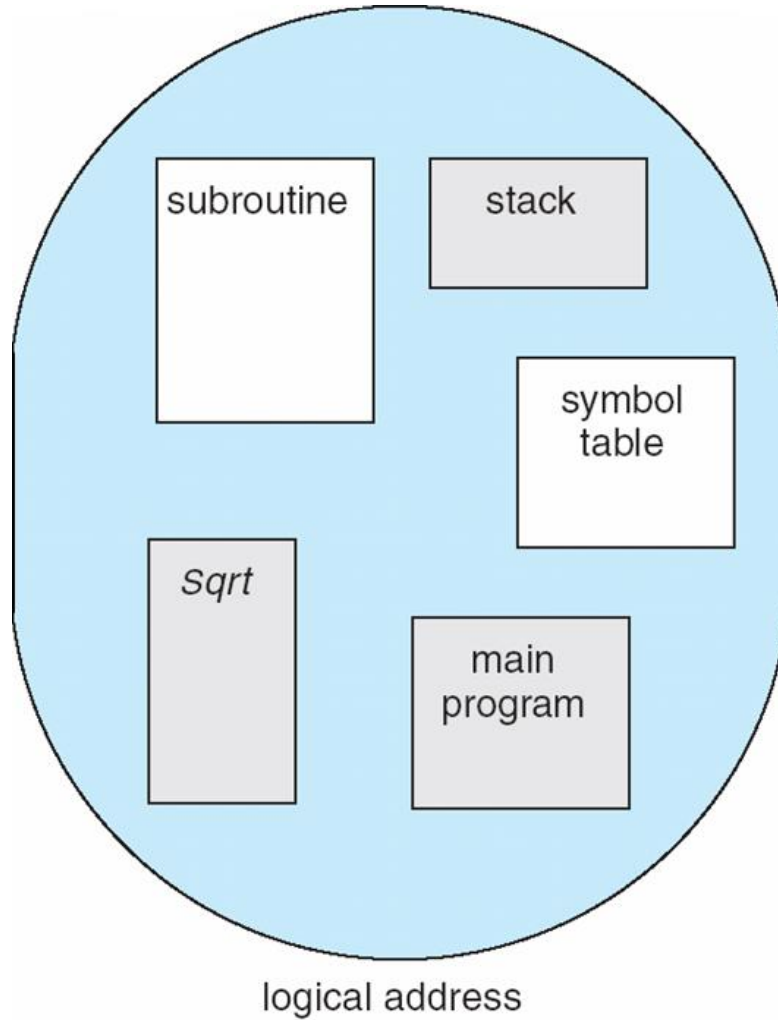
Inverted Page Table Architecture



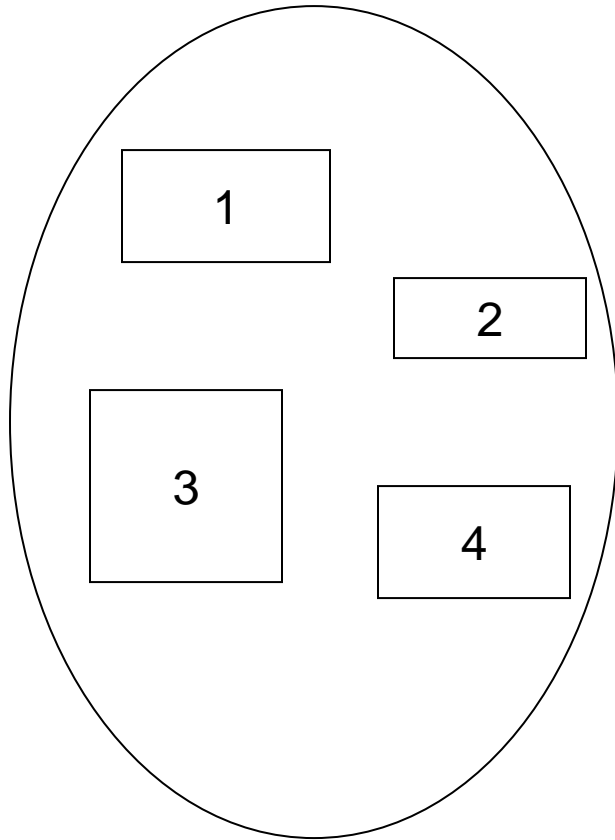
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
 - A segment is a logical unit such as:
 - main program
 - procedure
 - function
 - method
 - object
 - local variables, global variables
 - common block
 - stack
 - symbol table
 - arrays

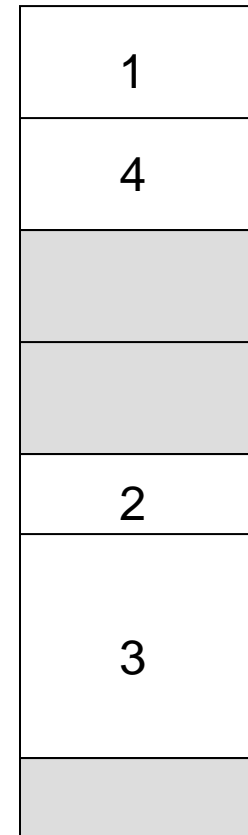
User's View of a Program



Logical View of Segmentation



user space



physical memory space

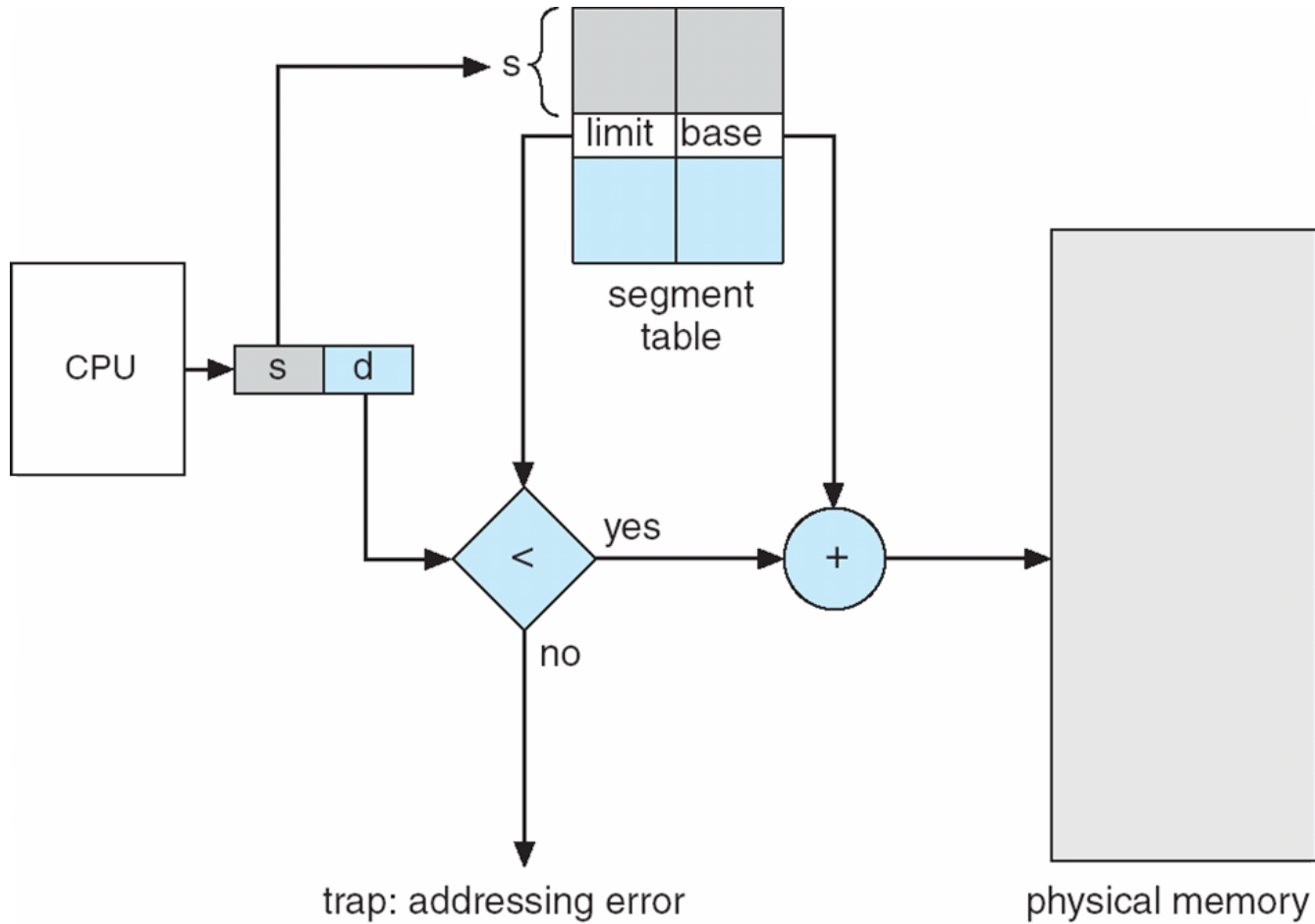
Segmentation Architecture

- Logical address consists of a two tuple:
 <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
 - **base** – contains the starting physical address where the segments reside in memory
 - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;
 segment number **s** is legal if **s** < **STLR**

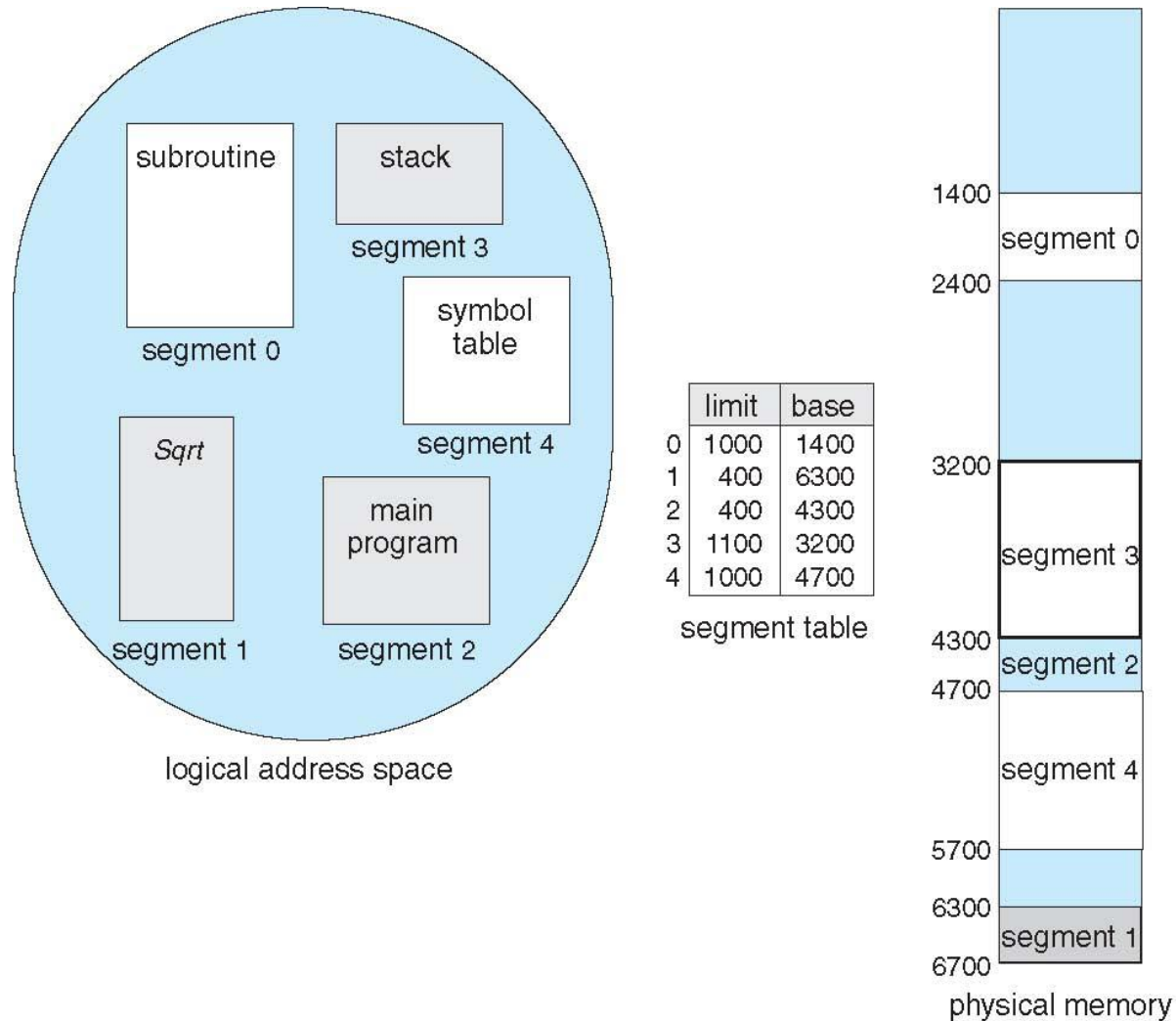
Segmentation Architecture (Cont.)

- Protection
 - With each entry in segment table associate:
 - validation bit = 0 \Rightarrow illegal segment
 - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level.
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.
- A segmentation example is shown in the following diagram.

Segmentation Hardware



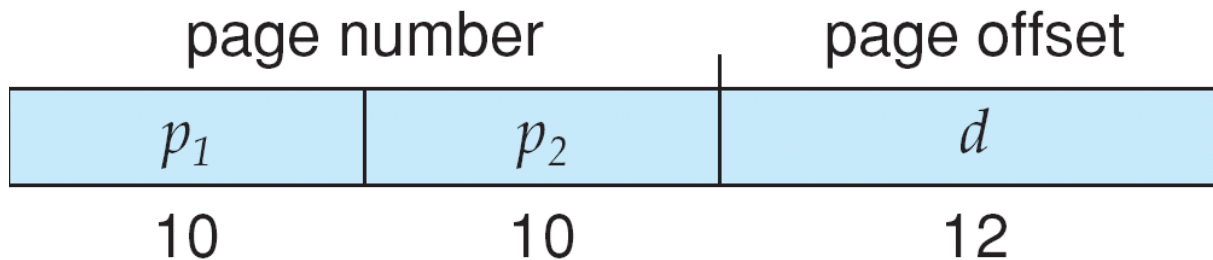
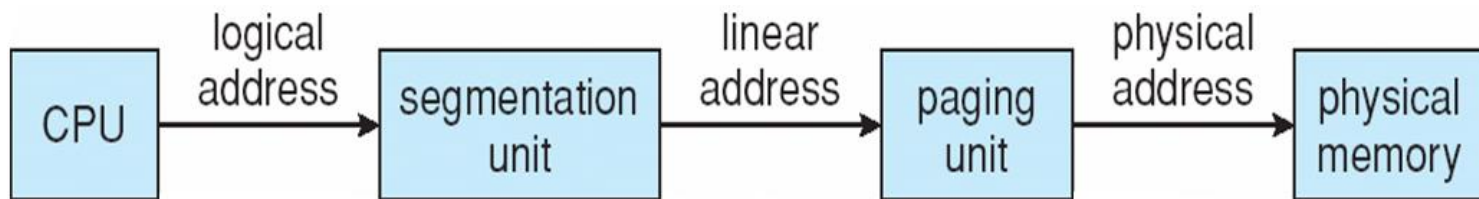
Example of Segmentation



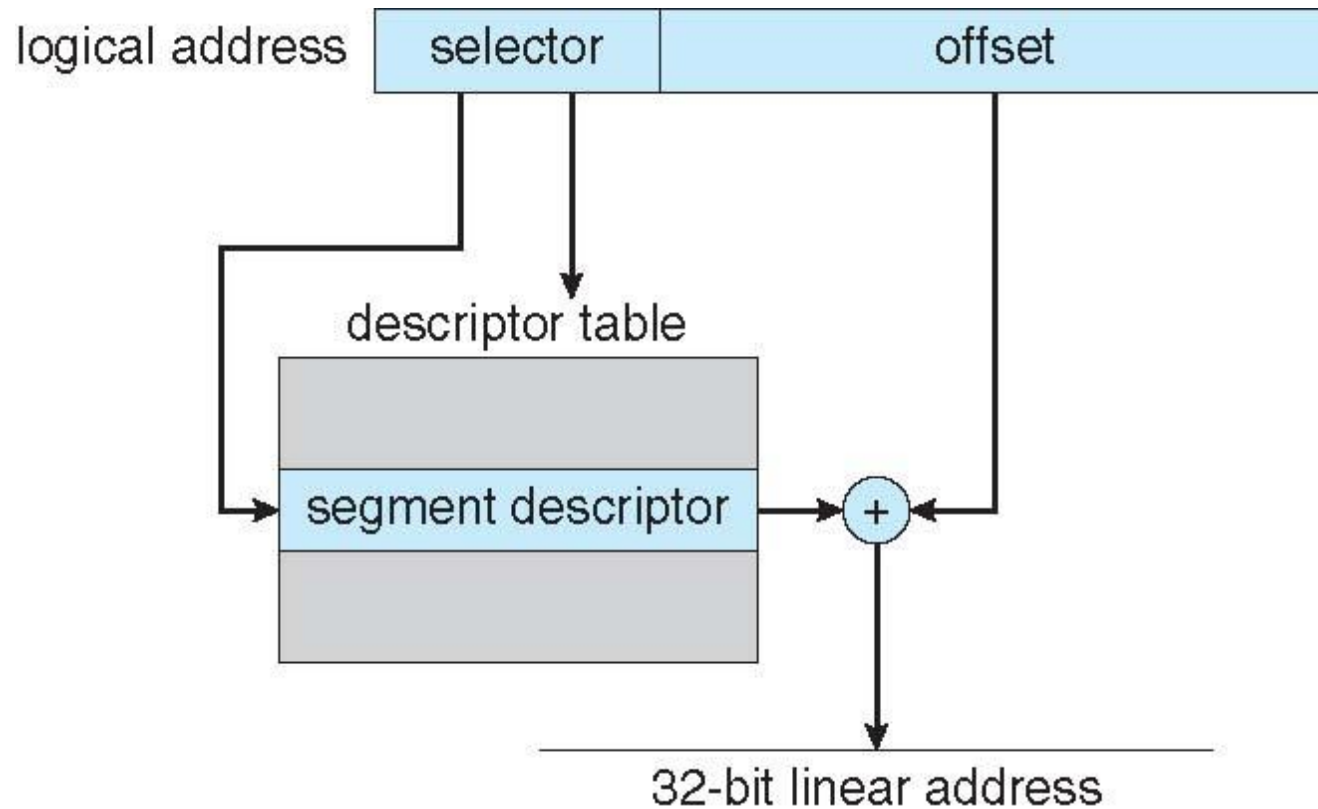
Example: The Intel Pentium

- Supports both segmentation and segmentation with paging
- CPU generates logical address
 - Given to segmentation unit
 - Which produces linear addresses
 - Linear address given to paging unit
 - Which generates physical address in main memory
 - Paging units form equivalent of MMU

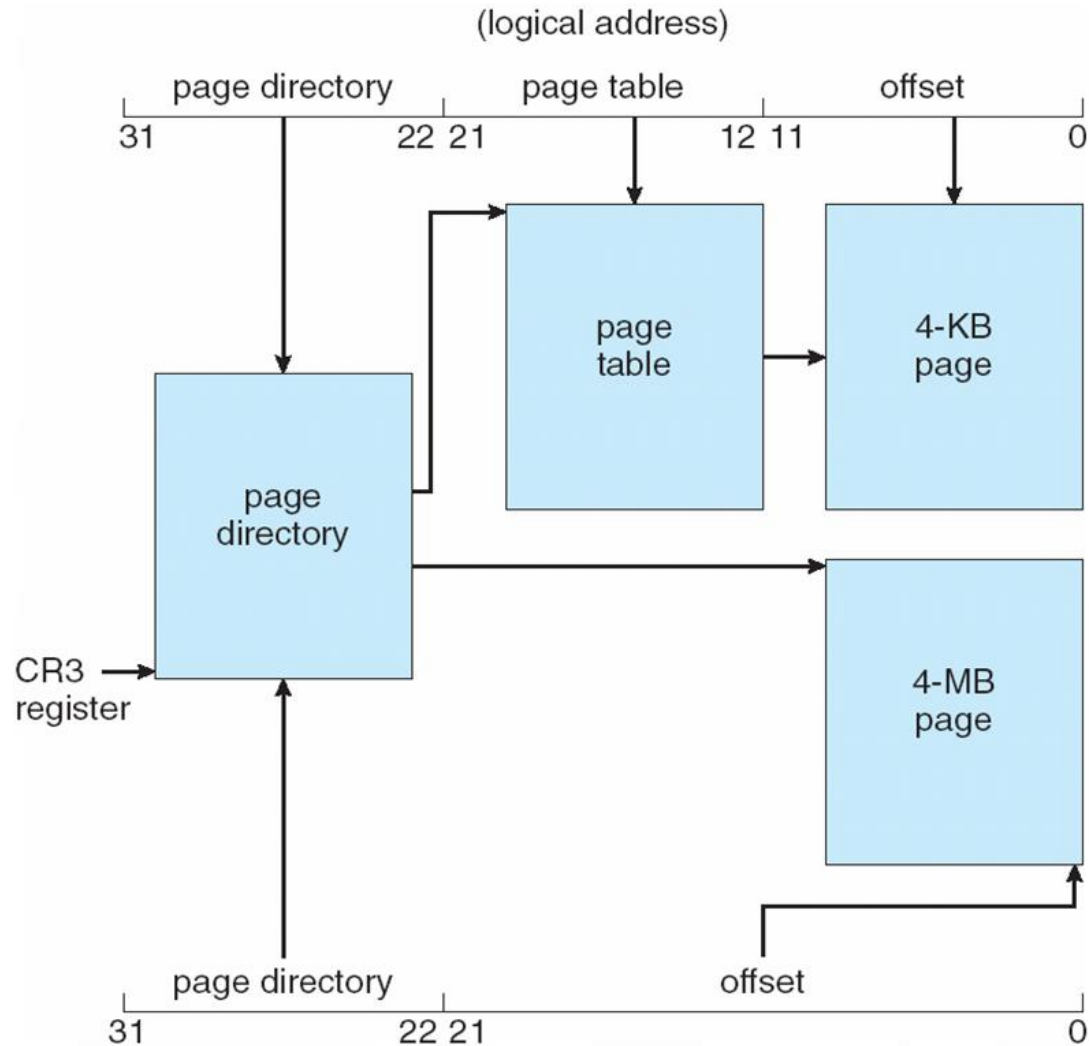
Logical to Physical Address Translation in Pentium



Intel Pentium Segmentation



Pentium Paging Architecture



Linear Address in Linux

Broken into four parts:

global directory	middle directory	page table	offset
---------------------	---------------------	---------------	--------

Three-level Paging in Linux

