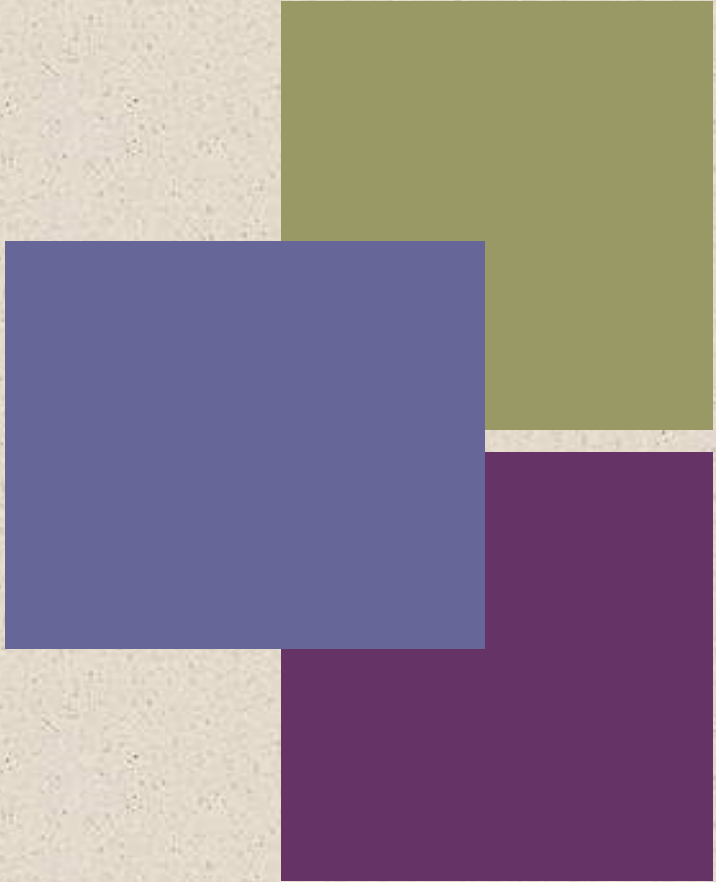


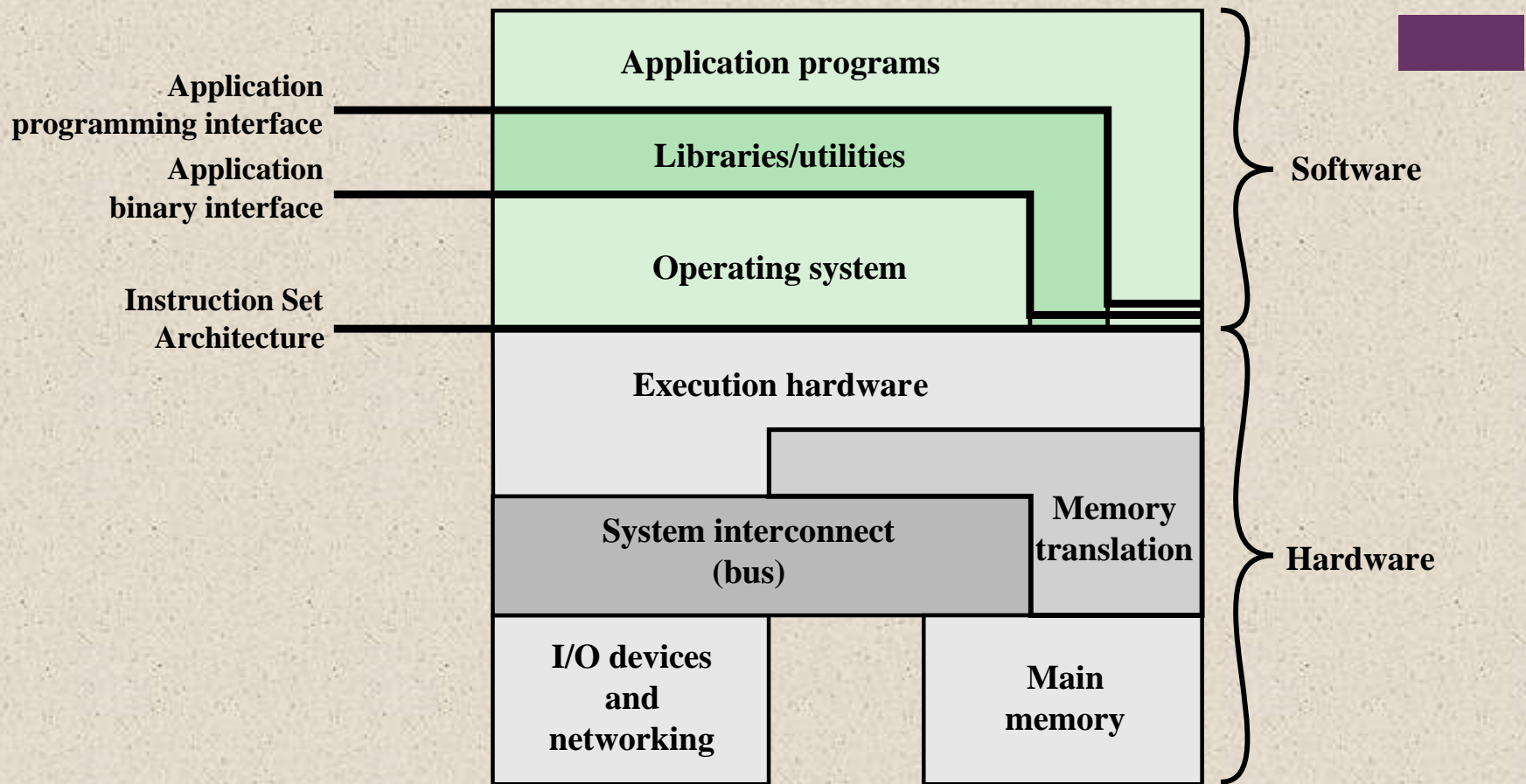
# William Stallings Computer Organization and Architecture 10<sup>th</sup> Edition

Orijinal slaytların  
çevirisidir.



# + Chapter 8

## Operating System Support



İşletim sistemi, uygulama programlarının yürütülmesini kontrol eden ve uygulamalar ile bilgisayar donanımı arasında bir arayüz görevi gören bir programdır. İki amacı olduğu düşünülebilir:

- **Kolaylık (Convenience):** İşletim sistemi, bilgisayarın kullanımını daha kolay hale getirir.
- **Verimlilik (Efficiency):** Bir işletim sistemi, bilgisayar sistemi kaynaklarının verimli bir şekilde kullanılmasına izin verir.

**Figure 8.1 Computer Hardware and Software Structure**



Bir kullanıcıya uygulama sağlamada kullanılan donanım ve yazılım, Şekil 8.1'de gösterildiği gibi katmanlı veya hiyerarşik bir şekilde görüntülenebilir.

- Bu uygulamaların kullanıcısı olan son kullanıcı (end user) genellikle bilgisayarın mimarisiyle ilgilenmez.
  - Böylelikle son kullanıcı, bir bilgisayar sistemini bir uygulama açısından (penceresinden) görür.
- Bu uygulama bir programlama dilinde ifade edilebilir ve bir uygulama programcısı tarafından geliştirilir.
- Bir uygulama programını, bilgisayar donanımını kontrol etmekten tamamen sorumlu olan bir dizi işlemci komutu olarak geliştirmek son derece karmaşık bir görev olacaktır.
- Bu görevi kolaylaştırmak için bir dizi sistem programı sağlanmıştır. Bu programlardan bazıları yardımcı programlar (**utilities**) olarak adlandırılır.
- Bunlar, program oluşturulmasına, dosyaların yönetimine ve I/O cihazlarının kontrolüne yardımcı olan sık kullanılan fonksiyonları yerine getirir.
- Bir programcı, bir uygulama geliştirirken bu olanaklardan yararlanır, ve uygulama çalışırken, belirli fonksiyonları gerçekleştirmek için yardımcı programları çağırır.





# Operating System (OS) Services



- En önemli sistem programı
- Donanımın ayrıntılarını programcıdan saklar (maskeler) ve programcıya sistemi kullanmak için uygun bir arayüz sağlar
- İşletim sistemi tipik olarak aşağıdaki alanlarda hizmetler sağlar:
  - Program creation
  - Program execution
  - Access to I/O devices
  - Controlled access to files
  - System access
  - Error detection and response
  - Accounting



İyi bir işletim sistemi, çeşitli kaynaklar için kullanım istatistiklerini toplar ve yanıt süresi gibi performans parametrelerini izler.

# Interfaces

- Tipik bir bilgisayar sistemindeki temel arayüzler:

## Instruction set architecture (ISA)

ISA, bir bilgisayarın izleyebileceği makine dili komutlarının repertuarını tanımlar.

Donanım ve yazılım arasındaki sınır

## Application binary interface (ABI)

Programlar arasında **binary** taşınabilirlik için bir standart tanımlar

ABI, işletim sistemine sistem çağrısı arayüzünü ve «user ISA» aracılığıyla bir sistemde bulunan donanım kaynaklarını ve hizmetleri tanımlar.

## Application programming interface (API)

API, yüksek seviyeli dil (HLL) kütüphane çağrıları ile desteklenen «user ISA» aracılığıyla **bir sistemde bulunan donanım kaynaklarına ve hizmetlere program erişimi sağlar.**

Bir API kullanmak, uygulama yazılımının aynı API'yi destekleyen **diğer sistemlere kolayca taşınmasını** (*to be ported*) sağlar

Hem uygulama programlarının hem de yardımcı programların ISA'ya doğrudan erişebileceğini unutmayın. Bu programlar için, komut repertuarının bir alt kümesi mevcuttur (user ISA). İşletim sistemi, sistem kaynaklarının (system ISA) yönetilmesiyle ilgilenen ek makine dili komutlarına erişime sahiptir.



# Operating System as Resource Manager

Bilgisayar; verilerin taşınması, depolanması ve işlenmesi ve bu işlevlerin kontrolü için bir kaynaklar kümesidir.

- İşletim sistemi bu kaynakların yönetiminden sorumludur

Bir kontrol mekanizması olarak işletim sistemi iki açıdan alışılmadık durum/davranış sergiler :

- İşletim sistemi, sıradan bilgisayar yazılımı ile aynı şekilde çalışır – işlemci tarafından yürütülen bir programdır
- İşletim sistemi sıklıkla kontrolü başkasına bırakır (*relinquish*) ve kontrolü yeniden kazanmasına (*regain control*) izin vermek için işlemciye güvenmesi gerekir



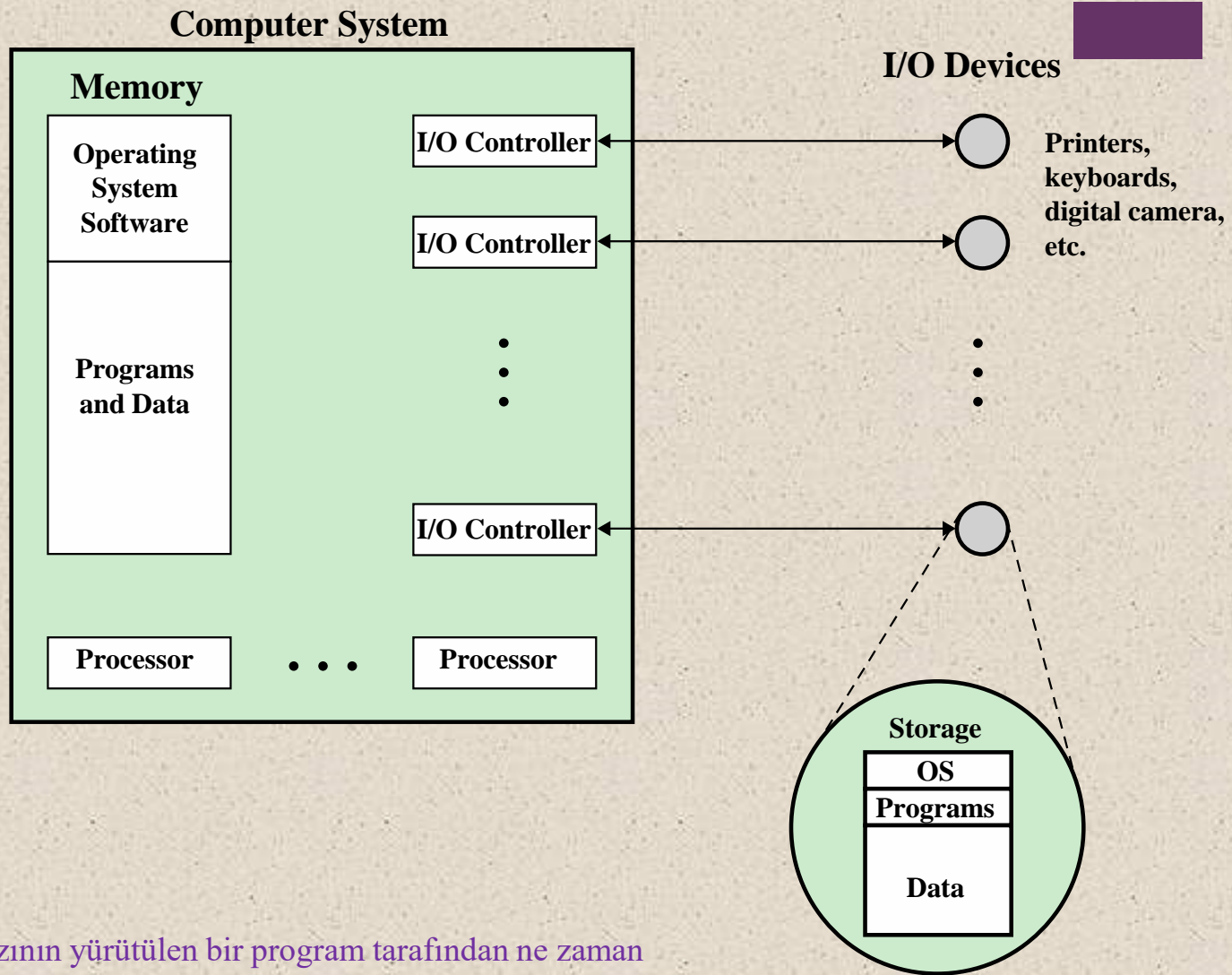
\* İşletim sistemi (OS), işlemciyi diğer sistem kaynaklarının kullanımında ve diğer programları yürütme zamanlamasında yönlendirir.

İşletim sisteminin bir kısmı ana bellektedir. Bu, işletim sisteminde en sık kullanılan fonksiyonları ve belirli bir zamanda kullanımda olan işletim sisteminin diğer bölümlerini içeren çekirdeği (**kernel/nucleus**) içerir.

Ana belleğin geri kalanı kullanıcı programlarını ve verilerini içerir.

Bu kaynağın (ana bellek) tahsisi, ileride göreceğimiz gibi, işletim sistemi ve işlemcideki bellek yönetimi donanımı (*memory-management hardware*) tarafından birlikte kontrol edilir.

İşletim sistemi (OS), bir I/O cihazının yürütülen bir program tarafından ne zaman kullanılabileceğine karar verir ve dosyalara erişimi ve bunların kullanımını kontrol eder. İşlemcinin kendisi bir kaynaktır ve işletim sistemi, belirli bir kullanıcı programının yürütülmesine ne kadar işlemci süresinin ayrılacağını belirlemelidir.



**Figure 8.2 The Operating System as Resource Manager**

Şekil 8.2, İşletim Sistemi tarafından yönetilen ana kaynakları göstermektedir.





# Types of Operating Systems



## ■ Interactive system

- Kullanıcı / programcı, bir işin yürütülmesini veya bir işlemin gerçekleştirilmesini istemek için doğrudan bilgisayarla etkileşime girer.
- Kullanıcı, uygulamanın niteliğine bağlı olarak, işin yürütülmesi sırasında bilgisayarla iletişim kurabilir.

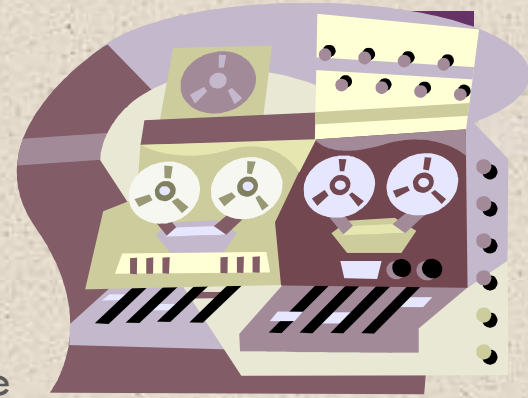
## ■ Batch system

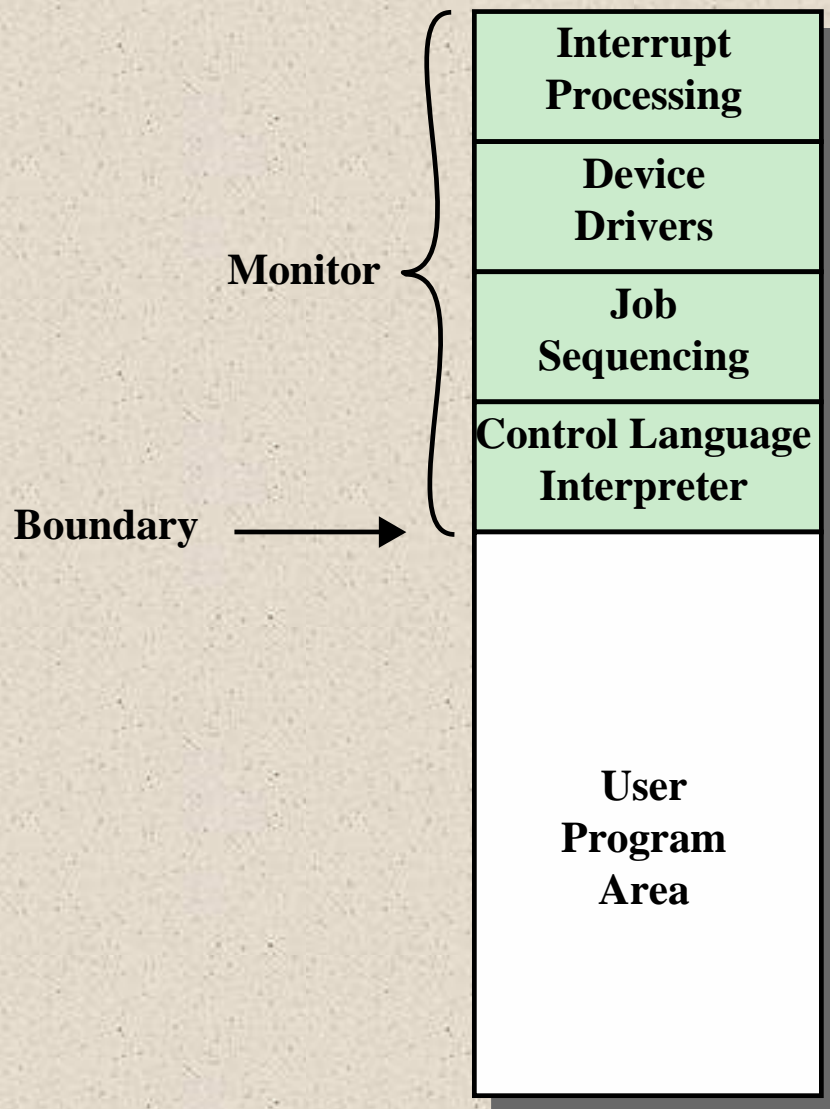
- İnteraktifin tam tersi
- Kullanıcının programı, diğer kullanıcıların programlarıyla birlikte gruplandırılır (*batched together*) ve bir bilgisayar operatörü tarafından gönderilir
- Program tamamlandıktan sonra sonuçlar kullanıcı için yazdırılır (*printed out for the user*).



# Early Systems

- 1940'ların sonlarından 1950'lerin ortalarına kadar programcı doğrudan bilgisayar donanımıyla etkileşime girdi – işletim sistemi (OS) yoktu
  - İşlemciler, ekran ledleri, toggle anahtarlar, bir tür giriş aygıtı ve bir yazıcıdan oluşan bir konsoldan çalıştırıldı.
- Problems:
  - Planlama (Scheduling)
    - İşlemci süresini rezerve etmek için kayıt sayfaları kullanıldı
      - Bu, kullanıcı erken bitirirse bilgisayarın ziyan edilen süresinin (*wasted idle time*) olmasına neden olabilir
      - Sorunlar meydana gelirse, kullanıcı problemi çözmeden önce durmaya zorlanabilir.
  - Setup time
    - Tek bir program (**job** olarak anılır) şunları kapsayabilir:
      - Derleyiciyi ve (yüksek seviyeli dilde yazılan) kaynak programı belleğe yükleme
      - Derlenmiş programı kaydetme
      - «object» programı ve ortak fonksiyonları yükleme ve birbirine bağlama (linking)

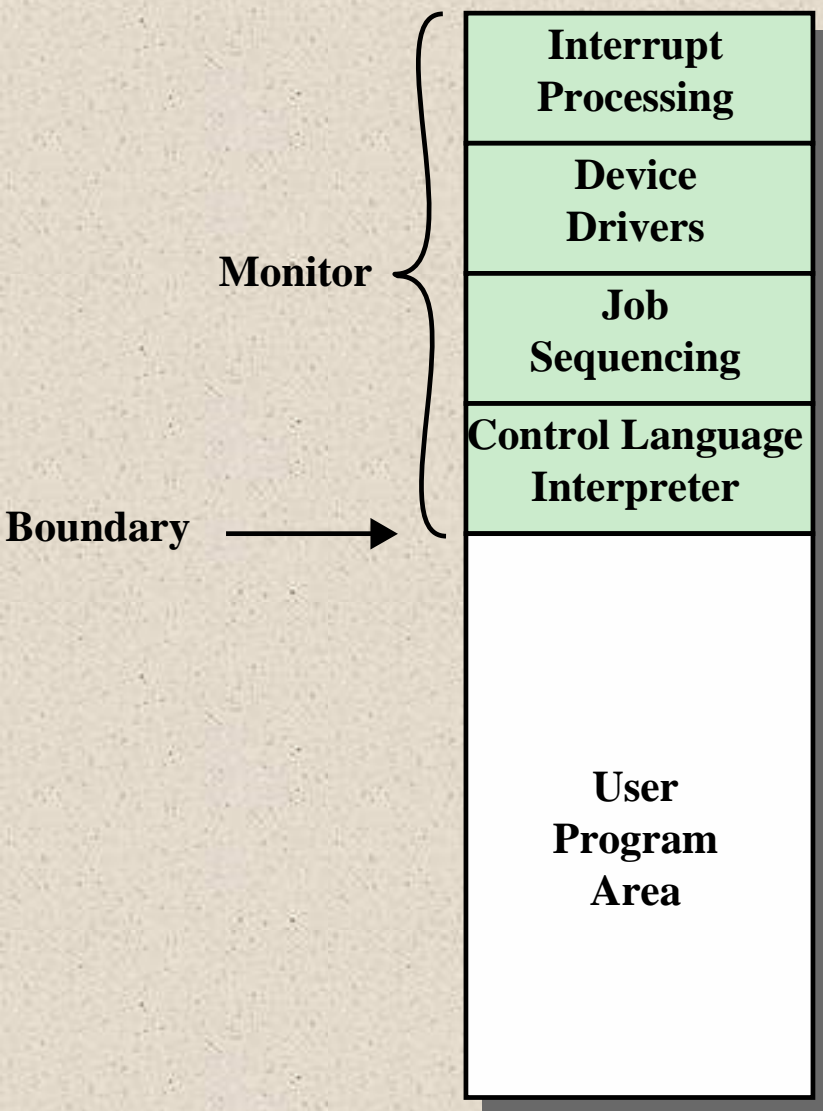




İlk işlemciler çok pahalıydı ve bu nedenle işlemci kullanımını en üst düzeye çıkarmak önemliydi. Planlama (*scheduling*) ve kurulum süresi nedeniyle boşa harcanan zaman kabul edilemezdi.

Kullanımı iyileştirmek (*improve utilization*) için basit toplu işletim sistemleri (*batch operating systems*) geliştirildi. Monitör (**monitor**) olarak da adlandırılan böyle bir sistemle, kullanıcının artık işlemciye doğrudan erişimi yoktur. Bunun yerine, kullanıcı işi kartlar veya bant üzerinde bir bilgisayar operatörüne gönderir, o da işleri sırayla bir araya getirir ve tüm partiyi monitör tarafından kullanılmak üzere bir giriş cihazına yerleştirir.

**Figure 8.3** Memory Layout for a Resident Monitor



Bu düzenin nasıl çalıştığını anlamak için, ona iki açıdan bakalım: monitör (**monitor**) ve işlemci (**processor**) açısından. Monitörün bakış açısından, monitör olayların sırasını kontrol eder. Bunun böyle olabilmesi için, monitörün büyük bir kısmının her zaman ana bellekte olması ve çalıştırılmaya hazır olması gerekir (Şekil 8.3).

Bu kısım, yerleşik monitör (**resident monitor**) olarak adlandırılır. Monitörün geri kalanı, bunları gerektiren herhangi bir işin başında kullanıcı programına alt yordamlar olarak yüklenen yardımcı programlar ve ortak fonksiyonlardan oluşur. Monitör, giriş cihazından (tipik olarak bir kart okuyucu veya manyetik teyp sürücüsü) işleri birer birer okur.

Okunduğu gibi mevcut iş kullanıcı programı alanına yerleştirilir ve kontrol bu işe geçer. İş tamamlandığında, monitöre kontrolü geri döndürür ve o da hemen sıradaki işi okur. Her işin sonuçları kullanıcıya teslim edilmek üzere yazdırılır.

**Figure 8.3 Memory Layout for a Resident Monitor**



# + From the View of the Processor . . .

- İşlemci, ana belleğin monitör programı içeren kısmından komutları yürütür
  - Bu komutlar, sonraki işin ana belleğin başka bir kısmında okunmasına neden olur
  - İşlemci, bir bitiş veya hata durumu ile karşılaşana kadar kullanıcının programındaki komutları yürütür.
  - Her iki olay da işlemcinin bir sonraki komutu monitör programından almasına neden olur
- Monitör, kurulum ve zamanlamayı (*scheduling*) yönetir
  - Bir grup/toplu iş (batch of jobs ), boşta kalma süresi olmadan mümkün olduğunca hızlı bir şekilde sıraya alınır ve yürütülür
- Job control language (JCL)
  - Monitöre komutlar sağlamak için kullanılan özel programlama dili türü
- Example:
  - \$JOB
  - \$FTN
  - ... Some Fortran instructions
  - \$LOAD
  - \$RUN
  - ... Some data
  - \$END
- Monitor, veya batch OS, sadece bir bilgisayar programıdır
  - İşlemcinin, kontrolü dönüşümlü olarak ele geçirmek ve bırakmak için ana belleğin çeşitli bölümlerinden komut alma yeteneğine dayanır.

\*\* Her bir FORTRAN komutu ve her veri ögesi ayrı bir delikli kartta veya kaset üzerinde ayrı bir kayıttadır. FORTRAN ve veri satırlarına ek olarak, iş "\$" ile gösterilen iş kontrol komutlarını (JCL) içerir.



# Desirable Hardware Features

## ■ Memory protection

- Kullanıcı programı, monitörü içeren bellek alanını değiştirmemelidir
- Böyle bir girişimde bulunulursa, işlemci donanımı hatayı algılamalı ve denetimi monitöre aktarmalıdır
- Bu durumda monitör işi iptal eder, bir hata mesajı yazdırır ve sonraki işi yükler

## ■ Timer


- Bir işin sistemi tekeline almasını önlemek için kullanılır
- Zamanlayıcının süresi dolarsa bir kesme oluşur ve kontrol monitöre döner

## ■ Privileged instructions

- Yalnızca monitör tarafından yürütülebilir
- İşlemci, bir kullanıcı programını yürütürken böyle bir komutla karşılaşursa, bir hata kesmesi (*error interrupt*) oluşur.
- I/O komutları ayrıcalıklıdır/imtiyazlıdır (*privileged*), bu nedenle monitör tüm I/O cihazlarının kontrolünü elinde tutar

## ■ Interrupts

- İşletim sistemine kontrolü bırakma ve kullanıcı programlarından kontrolü yeniden kazanma konusunda daha fazla esneklik sağlar



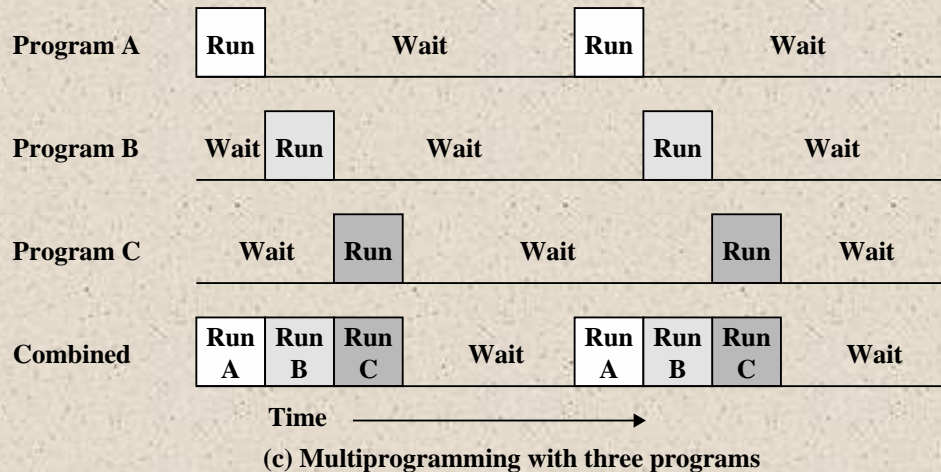
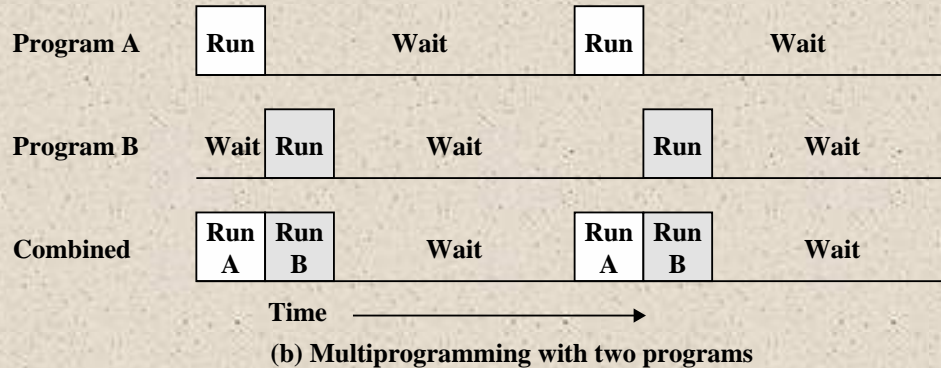
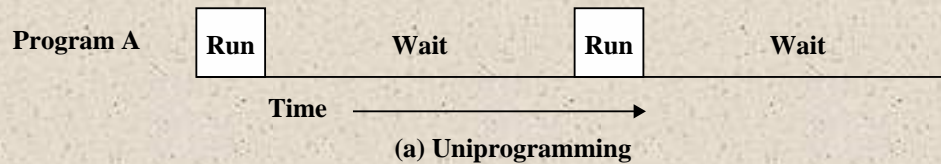
Read one record from file	15 $\mu$ s
Execute 100 instructions	1 $\mu$ s
Write one record to file	<u>15 <math>\mu</math>s</u>
TOTAL	31 $\mu$ s

$$\text{Percent CPU utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

## Figure 8.4 System Utilization Example

Basit bir «batch OS» tarafından sağlanan otomatik iş sıralamasıyla (job sequencing ) bile, işlemci genellikle boşta. Sorun, I/O cihazlarının işlemciye göre yavaş olmasıdır. Şekil 8.4, temsili bir hesaplamayı detaylandırmaktadır. Hesaplama, bir kayıt dosyasını işleyen ve kayıt başına ortalama 100 işlemci komutu gerçekleştiren bir programla ilgilidir. Bu örnekte bilgisayar, zamanının % 96'sından fazlasını I/O cihazlarının veri aktarımını bitirmesini bekleyerek geçiriyor! Şekil 8.5a bu durumu göstermektedir.





**Figure 8.5 Multiprogramming Example**

İşlemci, bir I/O komutuna ulaşana kadar belirli bir süreyi yürütmek (executing) için harcar. Devam etmeden önce I/O komutunun sona ermesini beklemesi gerekir.

Bu verimsizlik zorunlu değildir. İşletim sistemini (resident monitor) ve bir kullanıcı programını tutmak için yeterli bellek olması gerektiğini biliyoruz. İşletim sistemi ve iki kullanıcı programı için yer olduğunu varsayalım. Şimdi, bir işin I/O'yu beklemesi gerektiğinde, işlemci muhtemelen I/O'yu beklemeyen diğer işe geçebilir (Şekil 8.5b).

Ayrıca, belleği üç, dört veya daha fazla programı tutacak şekilde genişletebilir ve hepsi arasında geçiş yapılabilir (Şekil 8.5c). Bu teknik, çoklu programlama (**multiprogramming**) veya çoklu görev (**multitasking**) olarak bilinir. Modern işletim sistemlerinin ana temasıdır.



# Table 8.1

## Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	80 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Bu örnek, «multiprogramming» yararını göstermektedir. 250 Mbyte (işletim sistemi tarafından kullanılmayan) kullanılabilir belleği olan bir bilgisayar, bir disk, bir terminal ve bir yazıcı düşünün. Üç program, JOB1, JOB2 ve JOB3, Tablo 8.1'de listelenen özniteliklerle aynı anda yürütülmek üzere sunulur.

JOB2 ve JOB3 için minimum işlemci gereksinimlerini ve JOB3 tarafından sürekli disk ve yazıcı kullanımını varsayıyoruz.

Table 8.2

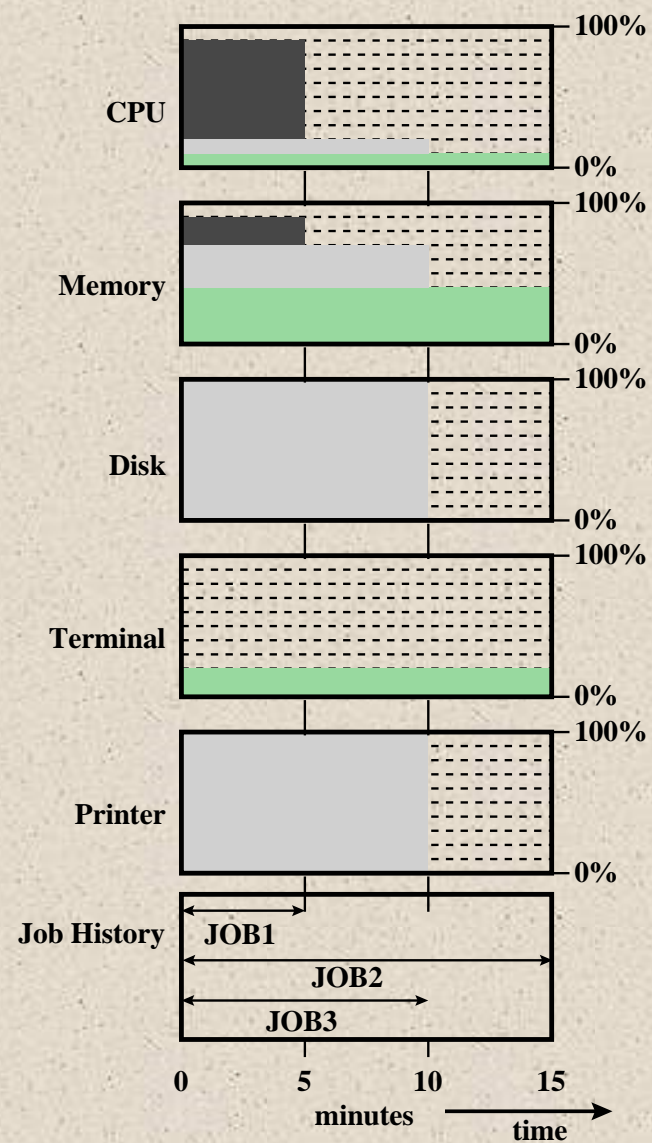
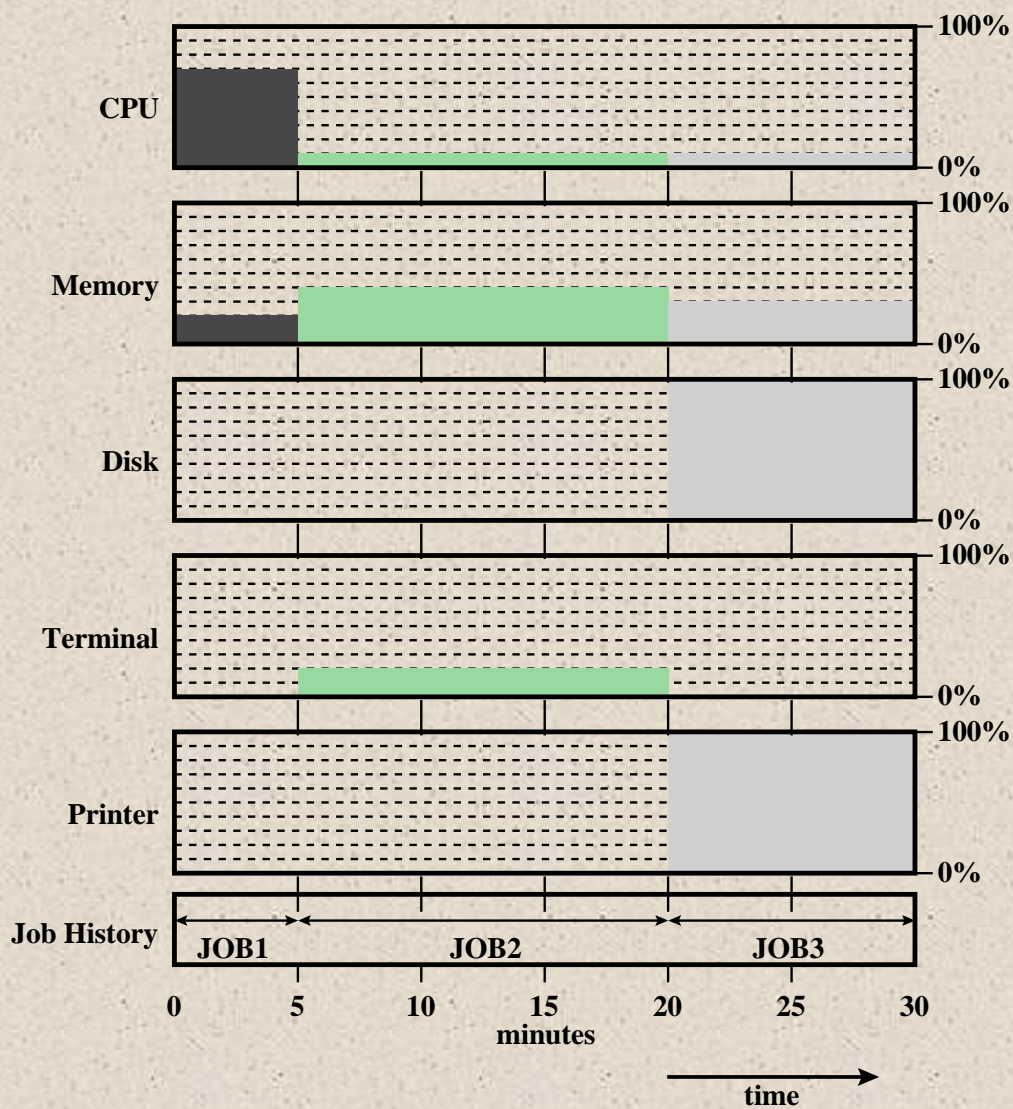
## Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput rate	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Ortalama kaynak kullanımı, ütilen iş (*throughput*) ve yanıt süreleri Tablo 8.2'nin «uniprogramming» sütununda gösterilmektedir. Cihaz bazında kullanım Şekil 8.6a'da gösterilmektedir. Gerekli 30 dakikalık sürenin ortalaması alındığında tüm kaynaklar için toptan yetersiz kullanım olduğu açıktır.

Şimdi, işlerin bir çoklu programlama işletim sistemi (multiprogramming OS) altında eşzamanlı olarak çalıştırıldığını varsayalım. İşler arasında çok az kaynak çekişmesi olduğu için, üçü de neredeyse minimum sürede çalışabilir ve diğerleriyle bilgisayarda bir arada bulunur.

Üç işin tümü 15 dakika içinde bitmiş olacak. Şekil 8.6b'de gösterilen histogramdan elde edilen Tablo 8.2'nin çoklu programlama sütununu incelerken iyileşme açıktır.



**Figure 8.6 Utilization Histograms**



# Time Sharing Systems



- Kullanıcı doğrudan bilgisayarla etkileşime girdiğinde kullanılır
- İşlemcinin zamanı birden çok kullanıcı arasında paylaşılır
- Birden çok kullanıcı, sisteme terminaller aracılığıyla eşzamanlı olarak erişir ve işletim sistemi, her kullanıcı programının yürütülmesini kısa bir hesaplama zamanı (*short burst or quantum of computation*) ile birleştirir.
- Örnek:
  - Bir seferde aktif olarak hizmet talep eden  $n$  kullanıcı varsa, her kullanıcı yalnızca efektif bilgisayar hızının ortalama  $1/n$ 'sini görecektir





## Table 8.3

# Batch Multiprogramming versus Time Sharing

<i>Ana amaç</i>	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system <i>İşletim sistemine direktif veren kaynak</i>	Job control language commands provided with the job	Commands entered at the terminal

**Table 8.4 Types of Scheduling**

<b>Long-term scheduling</b>	<i>Yürütülecek prosesler havuzuna ekleme kararı</i> The decision to add to the pool of processes to be executed
<b>Medium-term scheduling</b>	<i>Kısmen veya tamamen bellekte olan proseslerin yanına ekleme kararı</i> The decision to add to the number of processes that are partially or fully in main memory
<b>Short-term scheduling</b>	<i>İşlemci tarafından mevcut hangi prosesin yürütüleceğine ilişkin karar</i> The decision as to which available process will be executed by the processor
<b>I/O scheduling</b>	The decision as to which process's pending I/O request shall be handled by an available I/O device <i>Uygun bir I/O cihazı tarafından hangi prosesin bekleyen I/O talebinin ele alınacağı kararı</i>

«**Process**», «**job**»’dan biraz daha genel bir terimdir. Proses terimi için birçok tanım verilmiştir.

- Yürütülmekte olan bir program
- Bir programın "hareketli ruhu" (“animated spirit” of a program)
- Bir işlemcinin atandığı varlık

# Long Term Scheduling

Hangi programların işlenmek üzere gönderildiğini belirler



Bir iş (*job*), gönderildikten sonra kısa vadeli planlayıcı için bir proses haline gelir

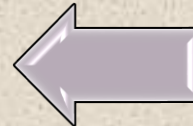


Bazı sistemlerde, yeni oluşturulan bir proses, takas durumunda (*swapped-out condition*) başlar, bu durumda orta vadeli planlayıcı için bir kuyruğa eklenir.



## Time-sharing system

- Bir kullanıcı sisteme bağlanmaya çalıştığında bir proses talebi üretilir
- İşletim sistemi, sistem doymuş hale gelene kadar tüm yetkili kişileri kabul edecektir.
- Bu noktada, sistemin dolu olduğunu ve daha sonra tekrar deneneceğini belirten bir mesajla bir bağlantı talebi karşılanır.



## Batch system

- Yeni gönderilen işler diske yönlendirilir ve bir toplu iş kuyruğunda tutulur
- Uzun vadeli planlayıcı, yapabildiğinde kuyruktan prosesler oluşturur.



# Medium-Term Scheduling and Short-Term Scheduling

## Medium-Term

- «**swapping**» işlevinin bir parçası
- «**Swapping-in**» kararı, çoklu programlamanın derecesini yönetme ihtiyacına dayanır
- «**Swapping-in**» kararı, «**swapped-out**» proseslerin bellek gereksinimlerini dikkate alacaktır.

## Short-Term

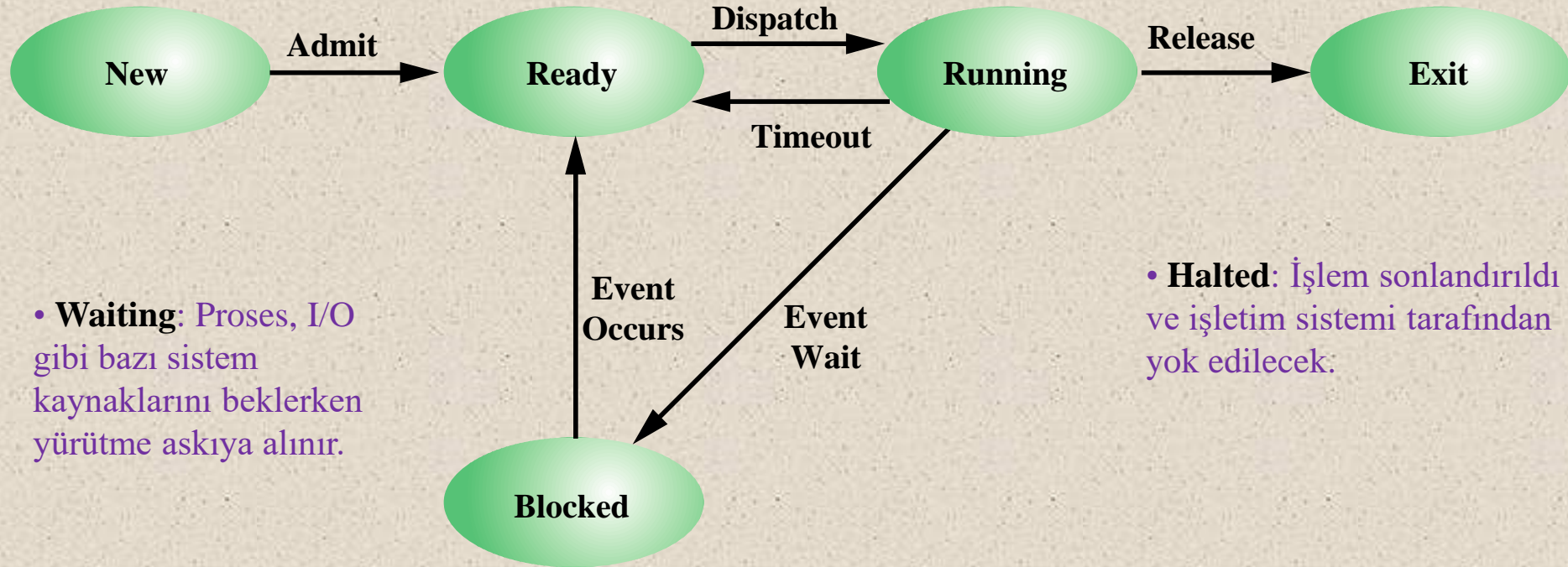
- «**Dispatcher**»  
(Sevkeden/Görev dağıtıcı) olarak da bilinir
- Sık sık icra eder ve sıradaki hangi işin yürütüleceğine ilişkin detaylı kararı verir





- **New:** Bir program, üst seviyedeki planlayıcı (*high-level scheduler*) tarafından kabul edilmiştir, ancak henüz yürütülmeye hazır değildir. İşletim sistemi, prosesi hazır hale getirerek süreci başlatacaktır.
- **Ready:** İşlem yürütülmeye hazır ve işlemciye erişim bekliyor.

- **Running:** İşlem işlemci tarafından yürütülüyor.



- **Halted:** İşlem sonlandırıldı ve işletim sistemi tarafından yok edilecek.

Bir süreç için en az beş tanımlanmış durum vardır (Şekil 8.7):

**Figure 8.7 Five-State Process Model**

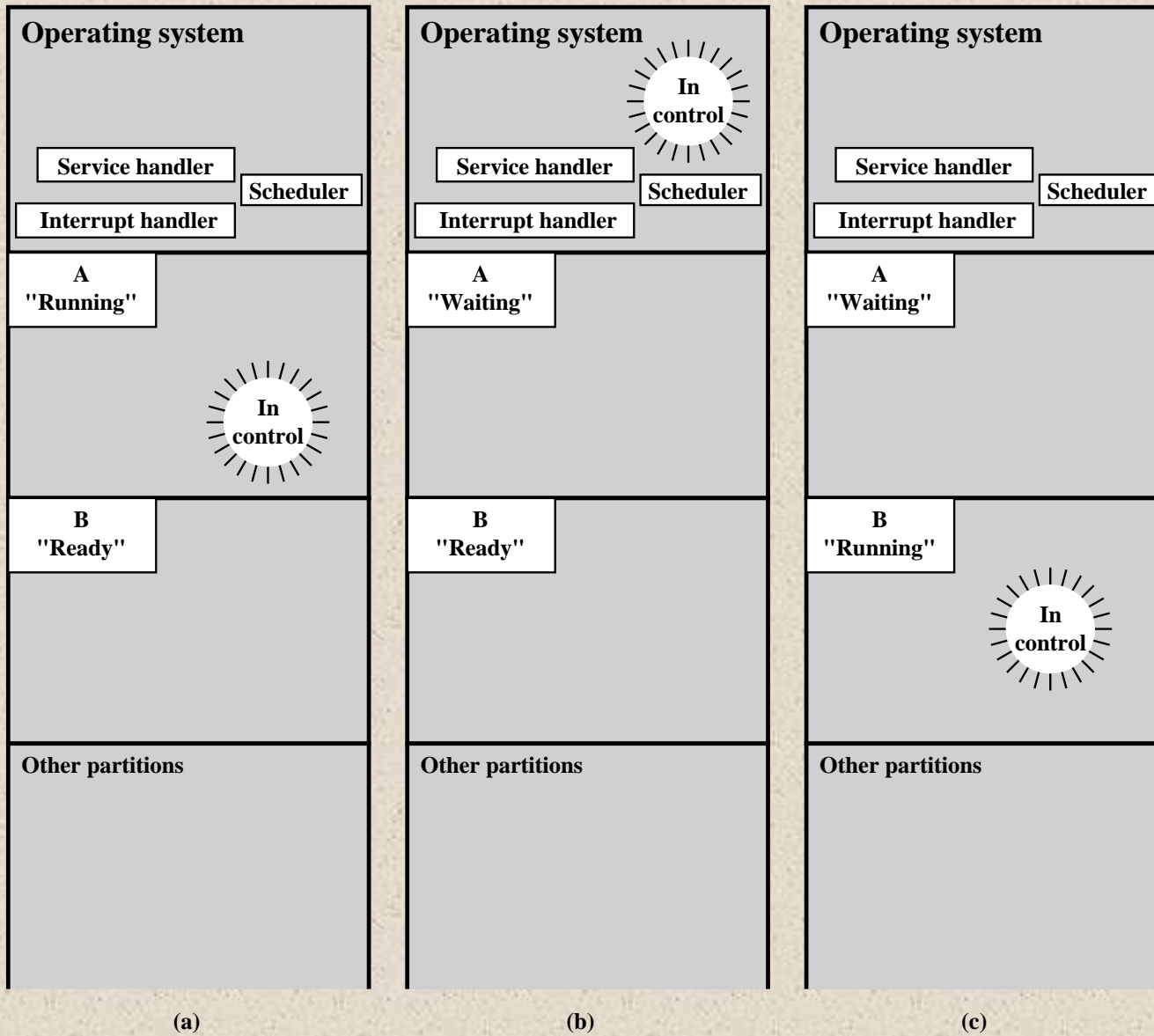
<b>Identifier</b>
<b>State</b>
<b>Priority</b>
<b>Program counter</b>
<b>Memory pointers</b>
<b>Context data</b>
<b>I/O status information</b>
<b>Accounting information</b>
• • •

**Figure 8.8 Process Control Block**

- **Accounting information** : Kullanılan işlemci zamanı ve saat zamanını, zaman limitlerini, hesap numaralarını vb. içerebilir.

Sistemdeki her proses için, İşletim Sistemi prosesin durumunu gösteren bilgileri ve prosesin yürütülmesi için gerekli diğer bilgileri muhafaza etmelidir. Bu amaçla, her proses işletim sisteminde tipik olarak aşağıdakileri içeren bir proses kontrol bloğu (PCB) ile temsil edilir (Şekil 8.8) .

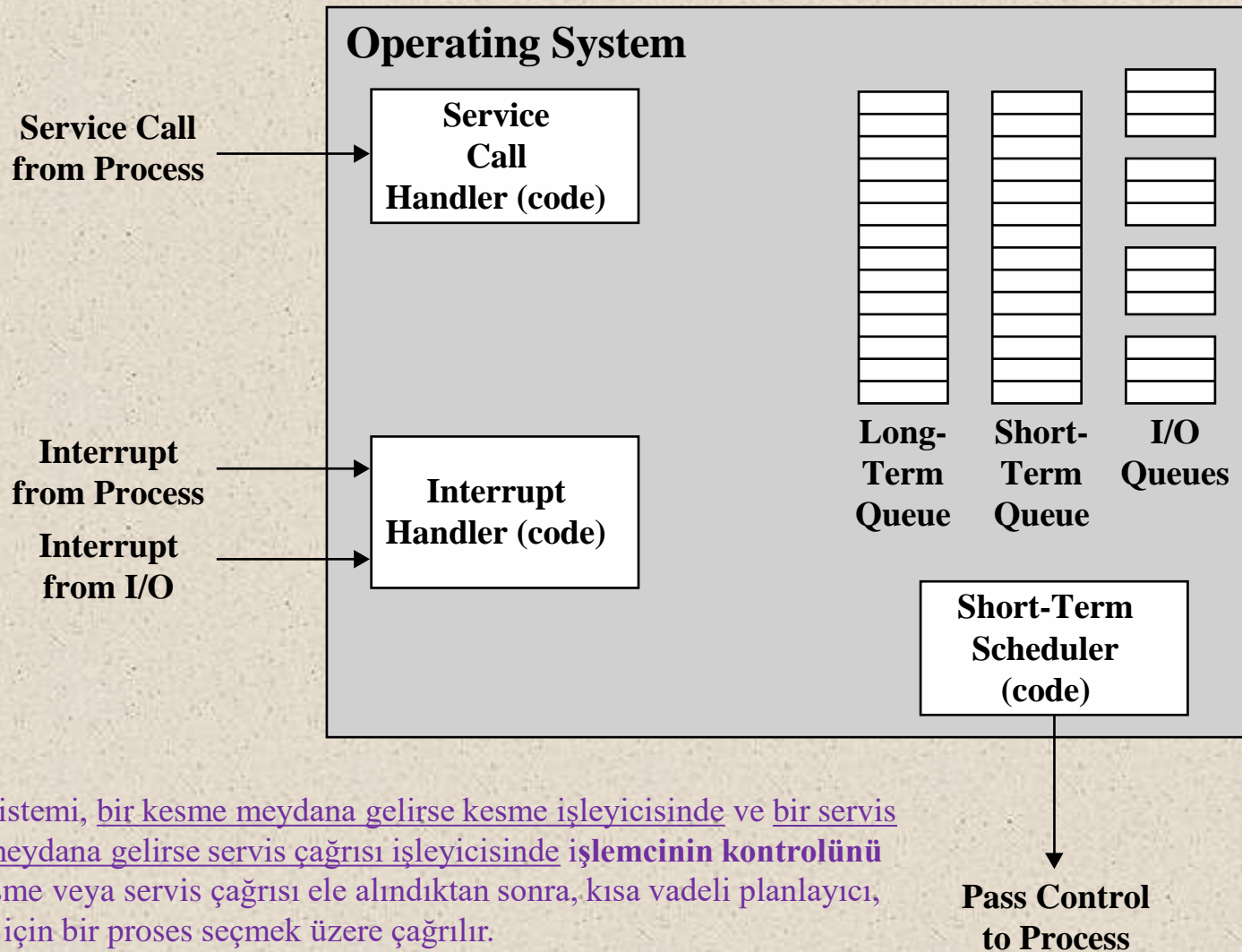
- **Identifier**: Her mevcut prosesin benzersiz bir tanımlayıcısı vardır.
- **State** : Prosesin mevcut durumu (yeni, hazır vb.).
- **Priority** : Göreceli öncelik seviyesi.
- **Program counter** : Programda yürütülecek bir sonraki komutun adresi.
- **Memory pointers** : Bellekteki prosesin başlangıç ve bitiş konumları.
- **Context data** : Bunlar, proses yürütülürken işlemciye registerlarda bulunan verilerdir. Şimdilik, bu verilerin prosesin “bağlamını” (*context*) temsil ettiğini söylemek yeterli. Context verileri artı program sayacı, proses koşma (*running*) durumundan çıktığında kaydedilir. Prosesin yürütülmesine devam ettiğinde işlemci tarafından alınırlar.
- **I/O status information** : Bekleyen I/O isteklerini, bu işleme atanmış I/O aygıtlarını (örneğin teyp sürücülerini), işleme atanan dosyaların bir listesini vb.



**Figure 8.9 Scheduling Example**

Şekil, belirli bir zamanda ana belleğin nasıl bölümlendiğini gösterir. İşletim sisteminin çekirdeği elbette her zaman yerleşiktir. Ek olarak, her biri belleğin bir kısmına tahsis edilmiş (A ve B dahil olmak üzere) bir dizi aktif proses vardır.

A prosesinin çalıştığı bir noktada başlıyoruz. İşlemci, A'nın bellek bölümünde bulunan programdaki komutları yürütmektedir. Daha sonraki bir zamanda, işlemci A'daki komutları yürütmeyi durdurur ve işletim sistemi alanında komutları yürütmeye başlar.



İşletim sistemi, bir kesme meydana gelirse kesme işleyicisinde ve bir servis çağrısı meydana gelirse servis çağrısı işleyicisinde işlemcinin kontrolünü alır. Kesme veya servis çağrısı ele alındıktan sonra, kısa vadeli planlayıcı, yürütme için bir proses seçmek üzere çağrılır.

**Figure 8.10 Key Elements of an Operating System for Multiprogramming**

Şekil 8.10, proseslerin çoklu programlanması ve planlanmasında yer alan işletim sisteminin ana unsurlarını gösterir.



Bu basit örnek, kısa vadeli planlayıcının temel işleyişini vurgular. İşletim sistemi, bir kesme meydana gelirse kesme işleyicisinde (*interrupt handler*) ve bir servis çağrısı meydana gelirse servis çağrısı işleyicisinde (*service-call handler*) işlemcinin kontrolünü alır. Kesme veya servis çağrısı ele alındıktan sonra, kısa vadeli planlayıcı, yürütmek için bir proses seçmek üzere çağrılır.

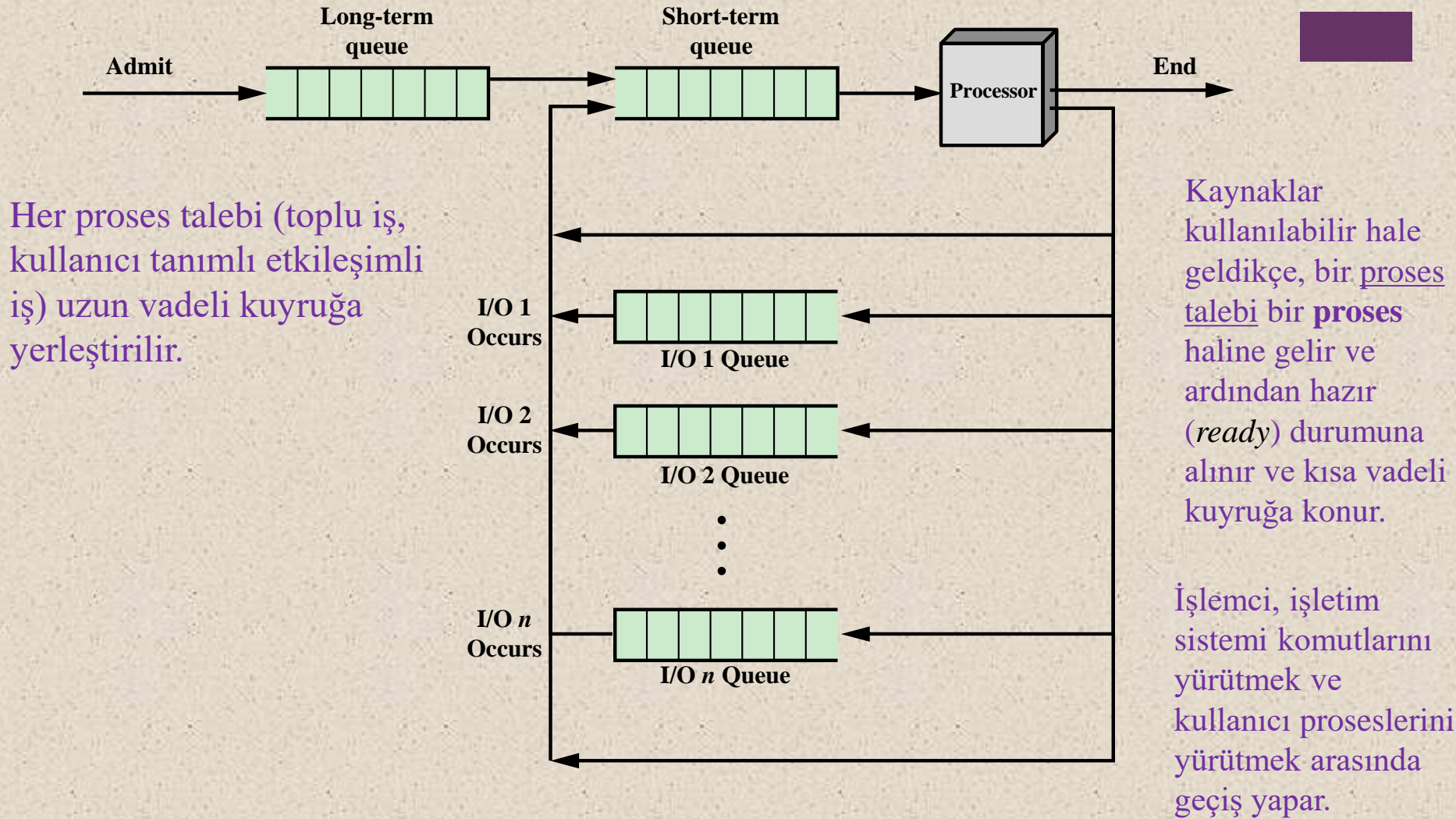
İşini yapmak için, işletim sistemi birkaç kuyruğa sahiptir. Her kuyruk, bazı kaynakları bekleyen proseslerin bir bekleme listesidir.

Uzun vadeli kuyruk (**long-term queue**), sistemi kullanmayı bekleyen işlerin bir listesidir. Koşulların izin verdiği ölçüde, üst düzey planlayıcı (*high-level scheduler*) bellek ayırarak ve bekleyen öğelerden biri için bir proses yaratacaktır.

Kısa vadeli kuyruk (**short-term queue**), hazır durumdaki tüm proseslerden oluşur. Bu proseslerden herhangi biri işlemciyi kullanan bir sonraki proses olabilir.

Birini seçmek kısa vadeli planlayıcıya bağlıdır. Genel olarak, bu, her prosese sırayla biraz zaman veren bir döngüsel (**round-robin**) algoritma ile yapılır.

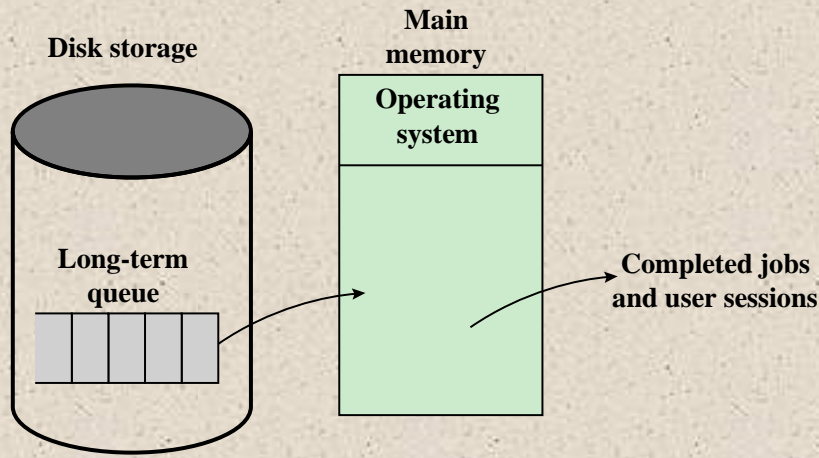
Öncelik seviyeleri de kullanılabilir. Son olarak, her bir I/O aygıtı için bir I/O kuyruğu vardır. Birden fazla proses aynı I/O cihazının kullanılmasını talep edebilir. Her bir cihazı kullanmayı bekleyen tüm işlemler, o cihazın kuyruğunda sıralanır.



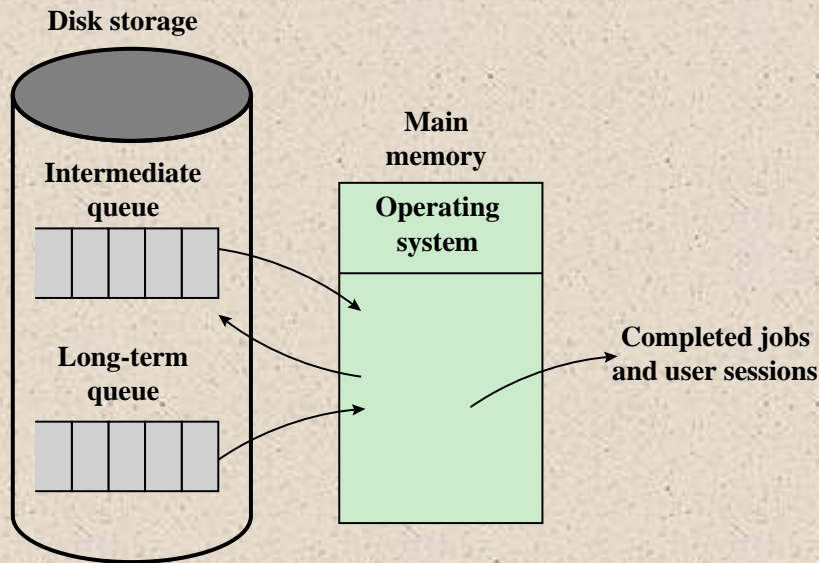
İşletim sistemi acil görevlerini bitirdiğinde, işlemciyi seçilen prosese devreder.

**Figure 8.11 Queuing Diagram Representation of Processor Scheduling**

Şekil 8.11, işletim sisteminin kontrolü altında bilgisayarda proseslerin nasıl ilerlediğini göstermektedir.



(a) Simple job scheduling



(b) Swapping

**Figure 8.12 The Use of Swapping**

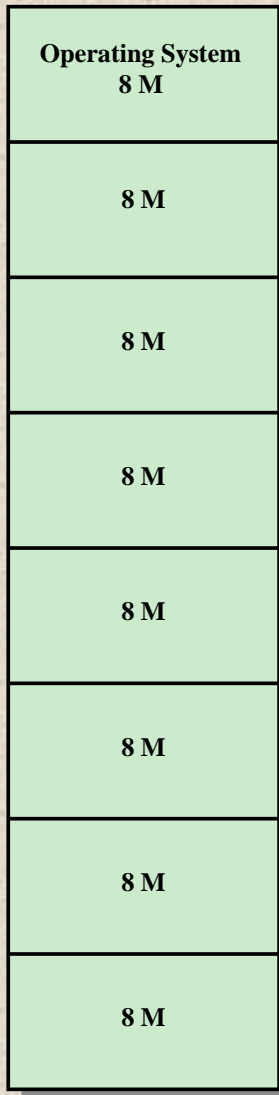
Ana bellek genişletilebilir ve böylece daha fazla işlemi barındırabilir. Ancak bu yaklaşımda iki kusur var.

- Birincisi, ana bellek bugün bile pahalıdır ve dolayısıyla sınırlıdır.
- İkincisi, programların bellek iştahı, bellek maliyeti düştükçe hızla arttı. Yani daha büyük bellek, daha fazla prosesle değil, daha büyük proseslerle sonuçlanır.

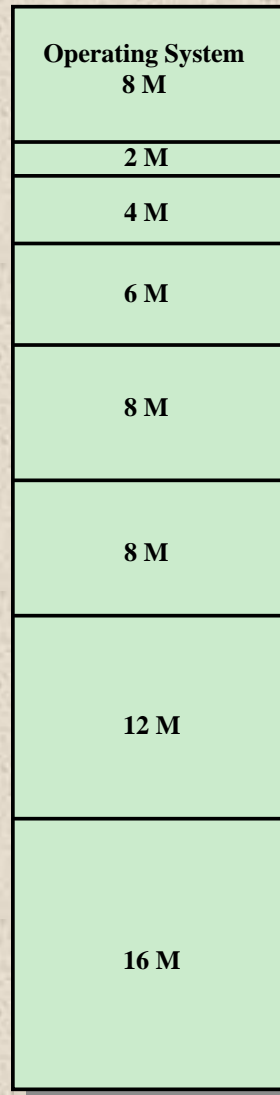
Başka bir çözüm, Şekil 8.12'de gösterilen **swapping**.

Yer müsait oldukça bunlar birer birer getirilir. Prosesler tamamlandıkça ana belleğin dışına taşınırlar. Şimdi, bellekteki proseslerin hiçbirinin hazır durumda olmadığı (örneğin, tümü bir I/O işlemini beklediği) ortaya çıkacaktır. İşlemci boşa kalmak yerine, bu işlemlerden birini diskten geri getirip (*swap*) bir ara kuyruğa (intermediate queue) koyar. Bu, geçici olarak bellekten atılan mevcut proseslerin bir kuyruğudur. İşletim sistemi daha sonra ara kuyruktan başka bir işlem getirir veya uzun vadeli kuyruktan yeni bir işlem talebini kabul eder. Yürütme daha sonra yeni gelen prosesle devam eder.





(a) Equal-size partitions



(b) Unequal-size partitions

Kullanılabilir belleği bölümlemek (*partitioning*) için en basit şema, Şekil 8.13'te gösterildiği gibi sabit boyutlu bölümler (*fixed-size partitions*) kullanmaktır. Bölümler sabit boyutta olmasına rağmen, eşit boyutta olmaları gerekmez. Bir proses belleğe alındığında, onu tutacak mevcut en küçük bölüme yerleştirilir.

Eşit olmayan sabit boyutlu bölümlerin kullanılmasıyla bile, boşa bellek israfı olacaktır.

Daha verimli bir yaklaşım, değişken boyutlu bölümler (*variable-size partitions*) kullanmaktır. Bir proses belleğe alındığında, tam olarak gerektirdiği kadar bellek tahsis edilir ve daha fazlası olmaz.

**Figure 8.13** Example of Fixed Partitioning of a 64-Mbyte Memory



### Logical address

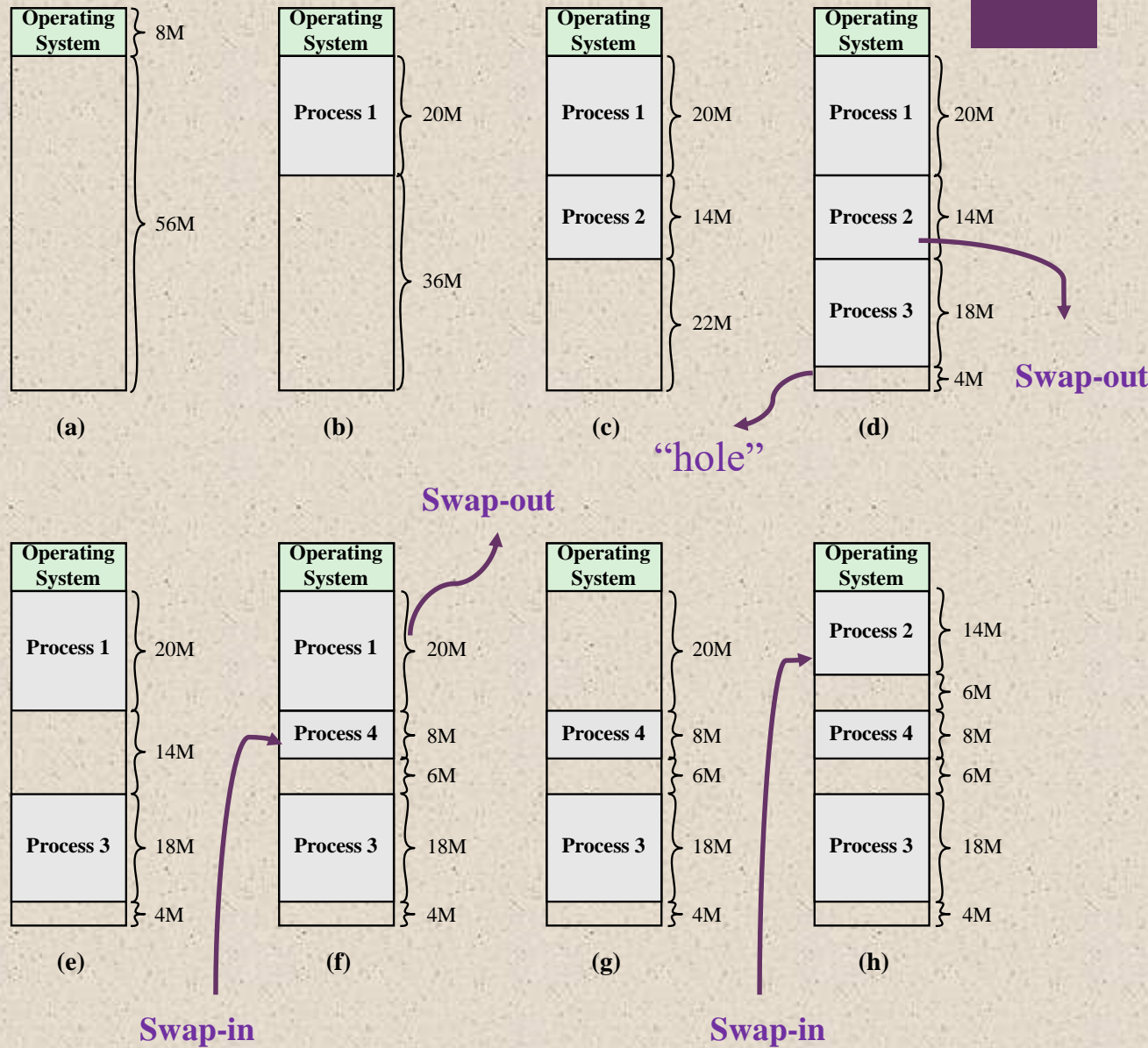
- programın başlangıcına göre bir konum olarak ifade edilir

### Physical address

- ana bellekte gerçek bir konum

### Base address

- prosesin mevcut başlangıç konumu



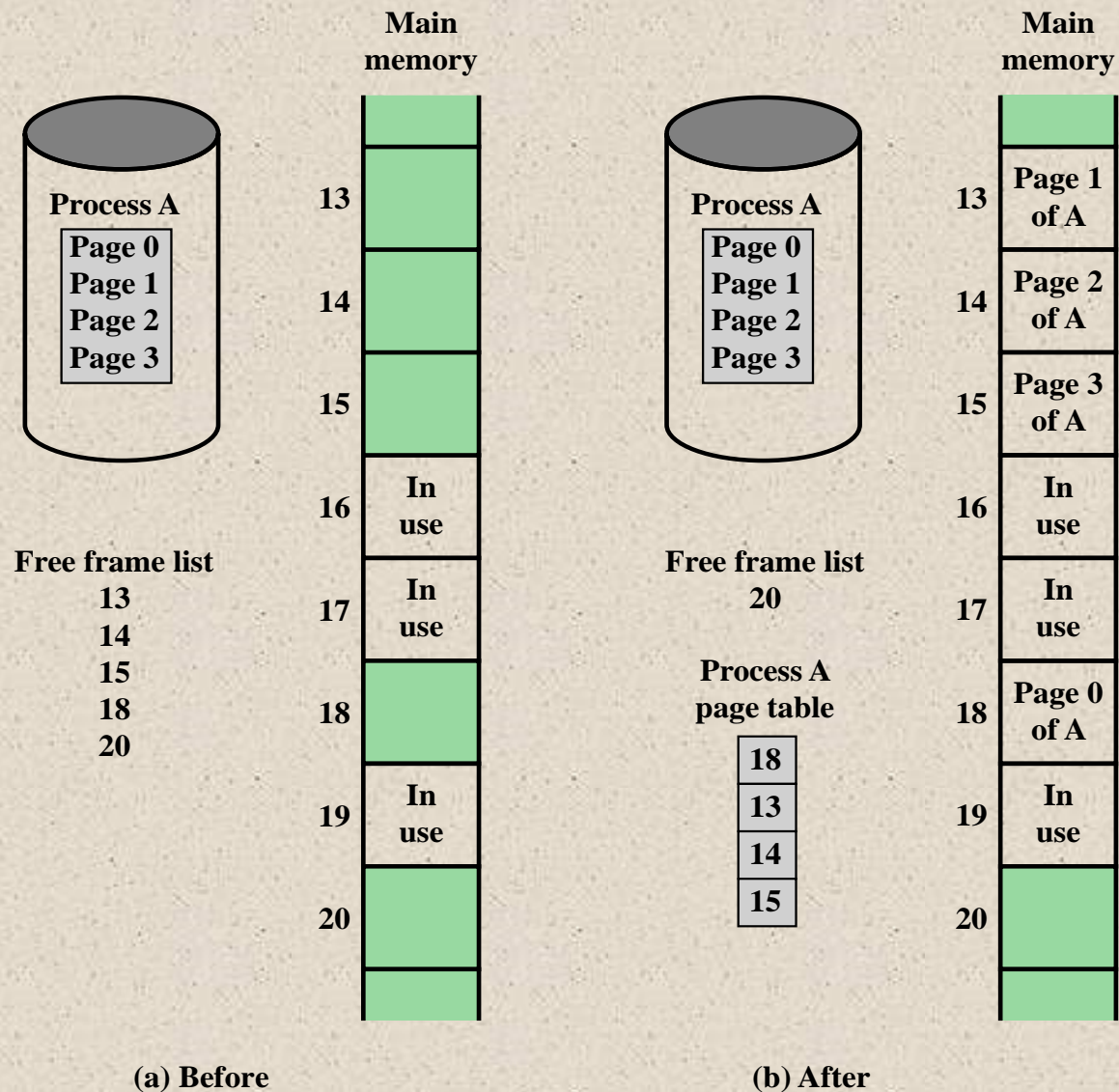
Şekil 8.14'te görüldüğü gibi; bir prosesin her «swap» işleminde ana bellekte aynı yere yüklenmesinin muhtemel olmadığı aşıkardır.

Figure 8.14 The Effect of Dynamic Partitioning

«page» olarak bilinen bir programın parçaları, «frame» veya «page frame» olarak bilinen kullanılabilir bellek parçalarına atanabilir.

Şekil 8.15, sayfaların ve çerçevelerin kullanımına bir örnek göstermektedir. Belirli bir zamanda, bellekteki çerçevelerin bazıları kullanımdadır ve bazıları boşta. Boş çerçevelerin listesi işletim sistemi tarafından tutulur.

Diskte depolanan İşlem A, dört sayfadan oluşur. Bu prosesi yükleme zamanı geldiğinde, işletim sistemi dört boş çerçeve bulur ve A prosesinin dört sayfasını dört çerçeveye yükler.

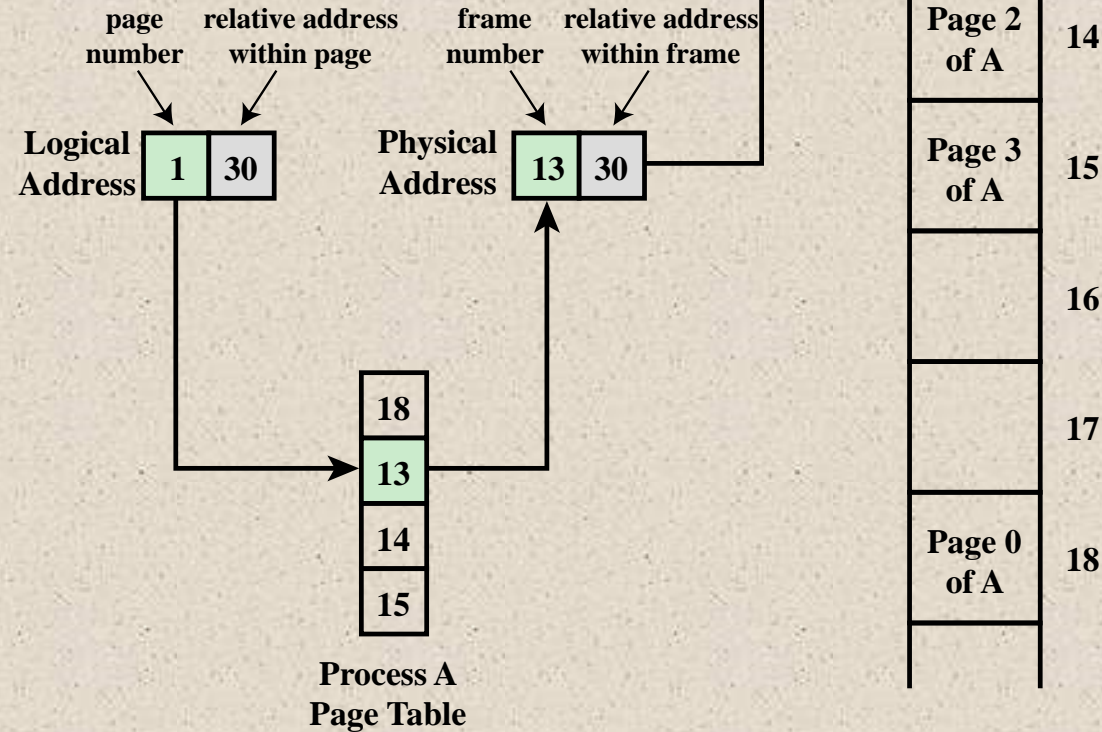


**Figure 8.15 Allocation of Free Frames**

OS, her proses için bir sayfa tablosu (**page table**) tutar. Sayfa tablosu, prosesin her sayfası için «**frame**» konumunu gösterir. Program içinde, her mantıksal adres, bir sayfa numarası (*page number*) ve sayfa içindeki göreceli bir adresten (*relative address*) oluşur.

«paging» ile, mantıksal-fiziksel adres dönüşümü işlemci donanımı tarafından yapılır. İşlemci, geçerli prosesin sayfa tablosuna (*page table*) nasıl erişeceğini bilmelidir.

Mantıksal bir adres (page number, relative address) sunulan işlemci, fiziksel bir adres (frame number, relative address) üretmek için sayfa tablosunu kullanır. Şekil 8.16'da buna ilişkin bir örnek gösterilmektedir.



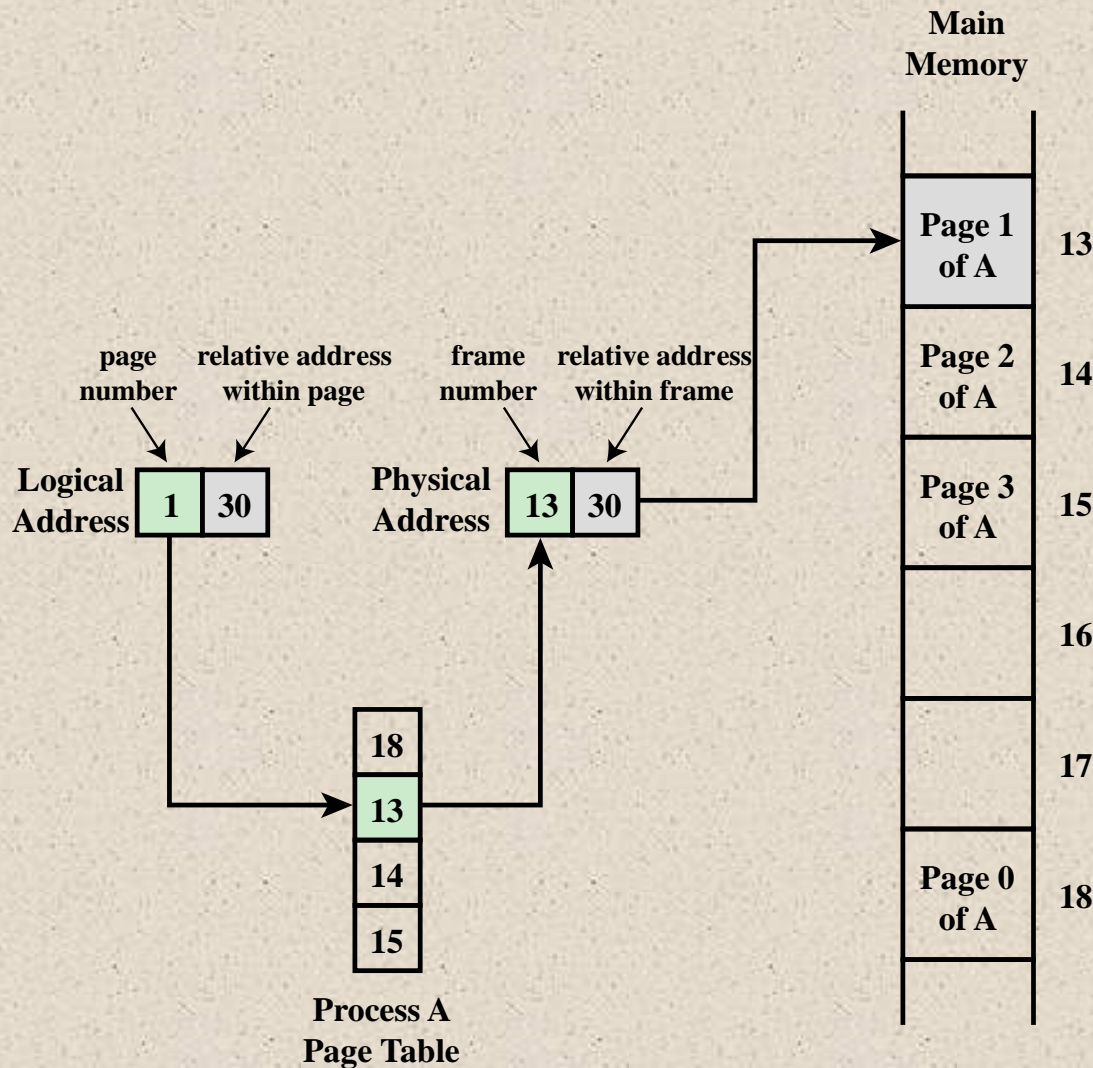
**Figure 8.16 Logical and Physical Addresses**

# + Virtual Memory

## Demand Paging

- Bir prosesin her sayfası yalnızca ihtiyaç duyulduğunda getirilir
- Principle of locality
  - Büyük bir prosesle çalışırken, bir programın küçük bir bölümü (alt yordam) ile sınırlı olabilir.
  - Sadece birkaç sayfanın (pages) belleğe yüklenmesi daha iyidir
  - Program, ana bellekte olmayan bir sayfadaki verilere veya bir komuta dallanmaya başvurursa, işletim sistemine istenen sayfayı getirmesini söyleyen bir sayfa hatası (**page fault**) tetiklenir.
- Avantajlar:
  - Bellekte daha fazla proses tutulabilir
  - Kullanılmayan sayfalar belleğe girip çıkmadığı için zamandan tasarruf edilir
- Dezavantajlar :
  - Bir sayfa getirildiğinde, başka bir sayfa atılmalıdır (sayfa değiştirme-*page replacement*)
  - Birazdan kullanılacak olan bir sayfa atılırsa, işletim sisteminin sayfayı tekrar alması gerekir.
  - *Thrashing*
    - İşlemci, zamanının çoğunu komutları yürütmek yerine sayfaları değiştirerek geçirdiğinde (*swapping pages rather than executing*)





Bellekten bir kelimeyi okumak için temel mekanizma, sayfa numarası ve ofsetten oluşan sanal veya mantıksal (*virtual*, or *logical*) bir adresin, bir sayfa tablosu kullanılarak çerçeve numarası ve ofsetten (*frame number* and *offset*) oluşan fiziksel bir adrese çevrilmesini içerir.

Sayfa tablosu (*page table*), prosesin boyutuna bağlı olarak değişken uzunlukta olduğundan, onu registerlarda tutmayı bekleyemeyiz.

Bunun yerine, erişilebilmesi için ana bellekte olması gerekir. Şekil 8.16, bu düzendeki bir donanım uygulamasını göstermektedir.

Belirli bir proses çalışırken, bir register o proses için sayfa tablosunun başlangıç adresini tutar. Bir sanal adresin sayfa numarası (*page number*), bu tabloyu indekslemek ve ilgili çerçeve numarasını (*frame number*) aramak için kullanılır. Bu, istenen gerçek adresi üretmek için sanal adresin ofset kısmı ile birleştirilir.

**Figure 8.16 Logical and Physical Addresses**

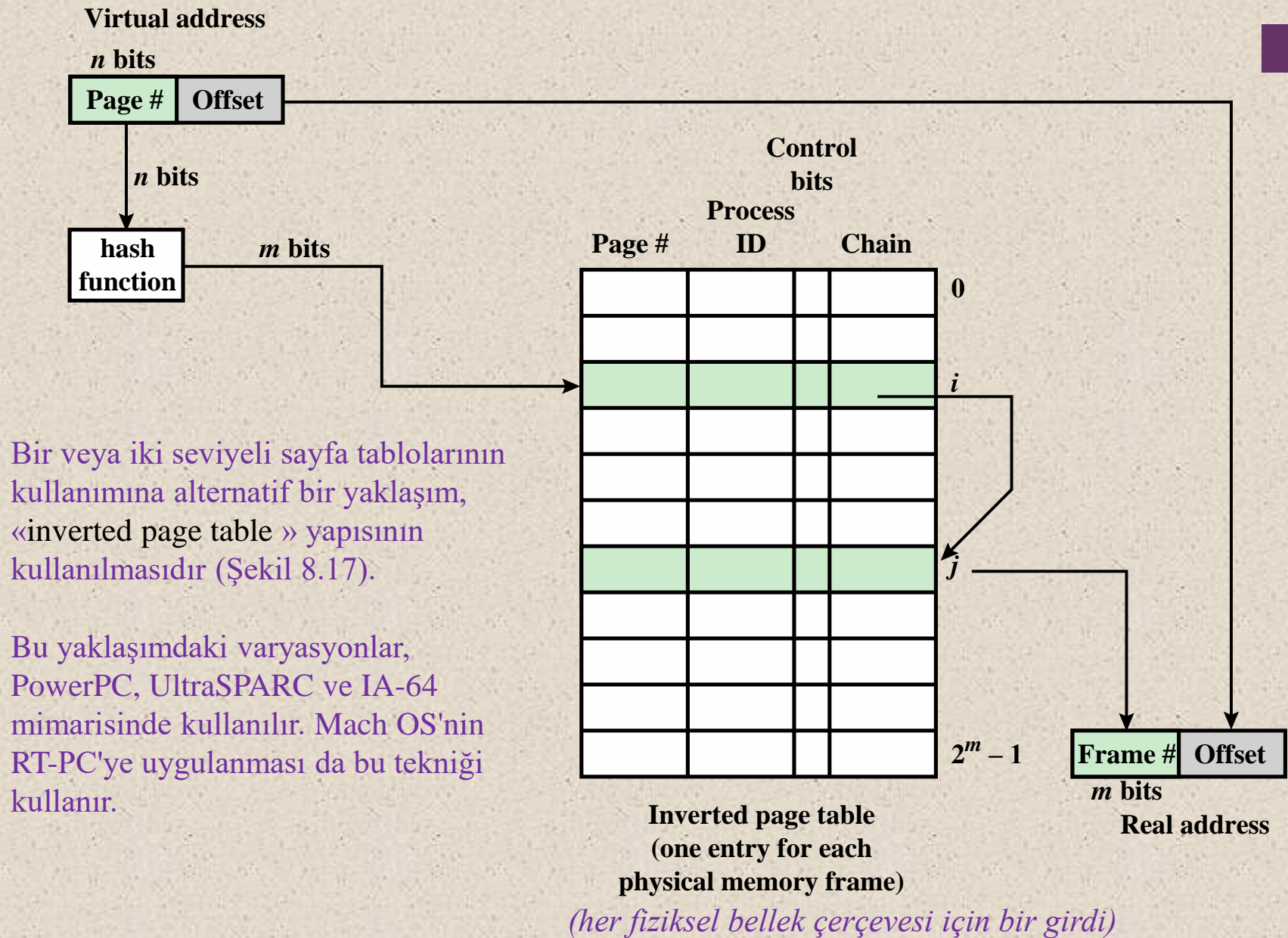


Çoğu sistemde, proses başına bir sayfa tablosu (*page table*) vardır. Ancak her proses büyük miktarda sanal bellek (*virtual memory*) işgal edebilir. Örneğin, VAX mimarisinde, her proses  $2^{31} = 2$  Gbyte'a kadar sanal belleğe sahip olabilir.  $2^9 = 512$ -bayt sayfaların kullanılması, proses başına  $2^{22}$  sayfa tablosu girdisi (*entry*) gerektiği anlamına gelir.

Açıkça görülüyor ki, tek başına sayfa tablolarına ayrılan bellek miktarı kabul edilemez derecede yüksek olabilir. Bu sorunun üstesinden gelmek için çoğu sanal bellek düzeni, sayfa tablolarını gerçek bellekten ziyade sanal bellekte depolar. Bu, diğer sayfalarda olduğu gibi sayfa tablolarının da sayfalamaya (*paging*) tabi olduğu anlamına gelir

Bir proses çalışırken, o anda yürütülmekte olan sayfanın sayfa tablosu girdisi (*page table entry*) dahil olmak üzere, sayfa tablosunun en azından bir kısmı ana bellekte olmalıdır.

Bazı işlemciler, büyük sayfa tablolarını düzenlemek için iki seviyeli bir düzenden (*two-level scheme*) yararlanır. Bu düzende, her girdinin (*entry*) bir sayfa tablosunu işaret ettiği bir sayfa dizini (*page directory*) vardır. Bu nedenle, sayfa dizininin uzunluğu  $X$  ise ve bir sayfa tablosunun maksimum uzunluğu  $Y$  ise, o zaman bir proses en fazla  $X * Y$  sayfadan oluşabilir. Tipik olarak, bir sayfa tablosunun maksimum uzunluğu bir sayfaya eşit olacak şekilde sınırlandırılmıştır.

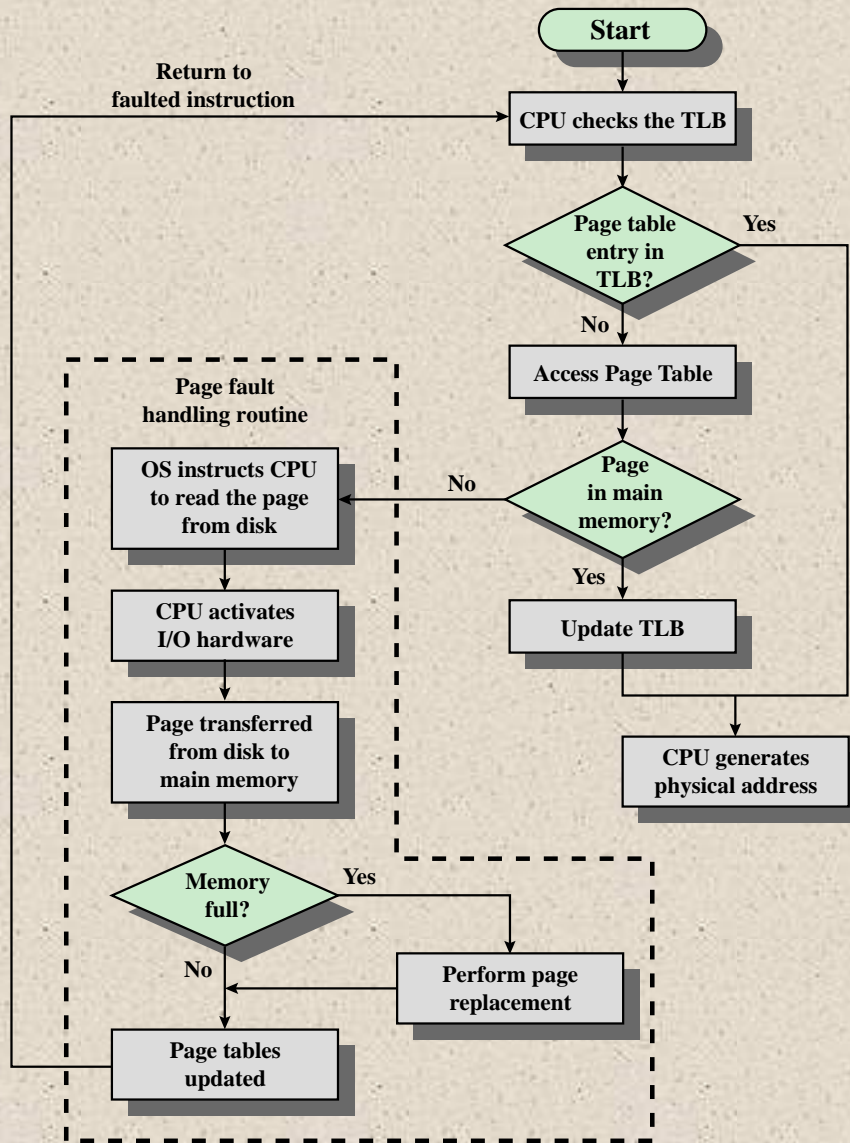


**Figure 8.17 Inverted Page Table Structure**



- Bu yaklaşımda, bir sanal adresin sayfa numarası kısmı, basit bir *hash* fonksiyonu kullanılarak bir *hash* değeriyle eşleştirilir.
- *Hash* değeri, sayfa tablosu girdilerini içeren ters çevrilmiş sayfa tablosunun işaretçisidir. Ters çevrilmiş sayfa tablosunda sanal sayfa başına bir girdi (*entry*) yerine **her gerçek bellek sayfa çerçevesi için bir girdi** vardır.
  - Bu nedenle, desteklenen proseslerin veya sanal sayfaların sayısından bağımsız olarak tablolar için sabit bir gerçek bellek oranı gereklidir.
- Birden fazla sanal adres aynı *hash* tablo girişiyle eşleşebileceğinden, taşmayı yönetmek için bir zincirleme tekniği (*chaining technique*) kullanılır. Hashing tekniği, genellikle kısa olan zincirlerle sonuçlanır - bir ile iki girdi arasında.
- Sayfa tablosunun yapısı, sayfa tablosu girdilerini sanal sayfa numarası yerine çerçeve numarasına göre indekslediği için ters çevrilmiş (*inverted*) olarak adlandırılır.





Prensip olarak, her sanal bellek başvurusu (*virtual memory reference*) **iki fiziksel bellek erişimine** neden olabilir: biri uygun sayfa tablosu girdisini almak ve diğeri istenen verileri almak için.

Bu nedenle, basit bir sanal bellek düzeni, bellek erişim süresini iki katına çıkarma etkisine sahip olacaktır.

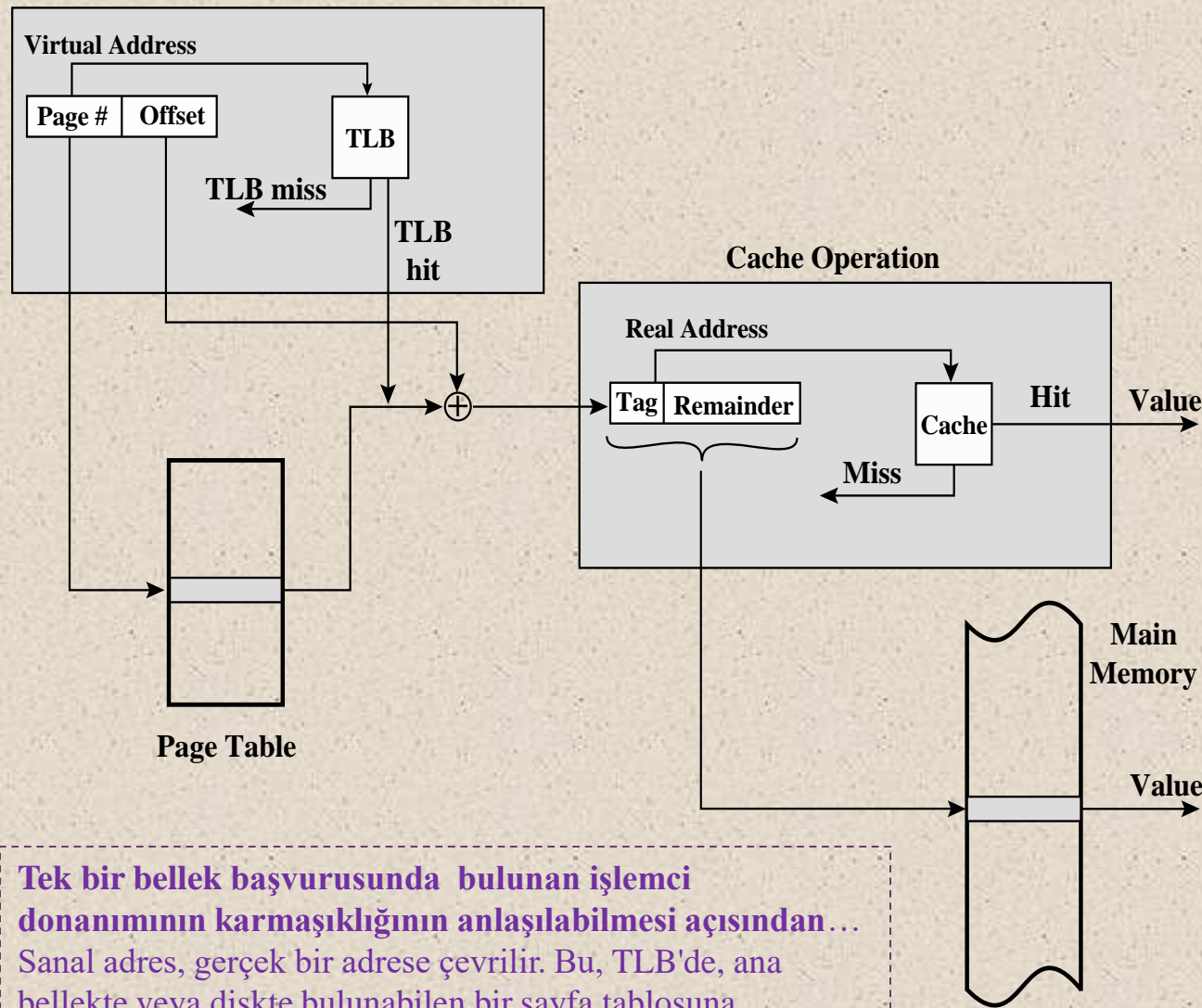
Bu sorunun üstesinden gelmek için, sanal bellek düzenlerinin çoğu, sayfa tablosu girdileri (*page table entries*) için özel bir önbellekten yararlanır, bu genellikle bir «**translation lookaside buffer (TLB)**» olarak adlandırılır.

Bu «cache», bir «cache memory» ile aynı şekilde çalışır ve en son kullanılan sayfa tablosu girdilerini içerir.

Şekil 8.18, TLB'nin kullanımını gösteren bir akış şemasıdır.

Figure 8.18 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

## TLB Operation



**Tek bir bellek başvurusunda bulunan işlemci donanımının karmaşıklığının anlaşılabilirliği açısından...** Sanal adres, gerçek bir adrese çevrilir. Bu, TLB'de, ana bellekte veya diskte bulunabilen bir sayfa tablosuna başvuruyu içerir. Başvuruda bulunulan kelime önbellekte, ana bellekte veya diskte olabilir.

**Figure 8.19 Translation Lookaside Buffer and Cache Operation**

Sanal bellek mekanizmasının önbellek sistemiyle (TLB önbelleği değil, ana bellek önbelleği) etkileşime girmesi gerektiğini unutmayın.

Bir sanal adres genellikle bir sayfa numarası, ofset şeklinde olacaktır.

İlk olarak, bellek sistemi, eşleşen «**page table**» girdisinin mevcut olup olmadığını görmek için TLB'ye başvurur.

- Öyleyse, gerçek (fiziksel) adres, «**frame number**» ile «**offset**» birleştirilerek oluşturulur.
- Değilse, girdi bir «**page table**» dan erişilir.

Bir «**tag**» ve bir «**remainder**» şeklinde olan gerçek adres oluşturulduktan sonra, bu kelimeyi içeren bloğun mevcut olup olmadığını görmek için önbelleğe bakılır.

- Eğer orada mevcutsa, işlemciye iade edilir. Değilse, kelime ana bellekten alınır.

# + Segmentation

Adreslenebilir belleğin alt bölümlere ayrılabilceği başka bir yol daha vardır, bölümleme (*segmentation*) olarak bilinir. Sayfalama (*paging*) programcı tarafından görünmezken ve programcıya daha geniş bir adres alanı sağlama amacına hizmet ederken, bölümleme genellikle programcı tarafından görülebilir.



- Genellikle programcı tarafından görülebilir
- Programları ve verileri düzenlemek için bir kolaylık olarak ve ayrıcalık ve koruma özelliklerini (*privilege and protection attributes*) komutlar ve verilerle ilişkilendirmek için bir araç olarak sağlanmıştır
- Programcının hafızayı birden çok adres alanı veya bölümden oluşuyormuş gibi görüntülemesine izin verir
- Advantages:
  - Büyüyen veri yapılarının işlenmesini basitleştirir
  - Tüm program setinin yeniden bağlanmasına ve yeniden yüklenmesine (*re-linked and re-loaded*) gerek kalmadan programların bağımsız olarak değiştirilmesine ve yeniden derlenmesine izin verir
  - Prosesler arasındaki paylaşıma olanak tanır
  - Korumaya (*protection*) olanak tanır



- X86, hem segmentasyon hem de sayfalama için donanım içerir. Her iki mekanizma da devre dışı bırakılarak kullanıcının dört farklı bellek görünümünden seçim yapmasına izin verilebilir:
- Unsegmented unpaged memory
  - Sanal adres, fiziksel adresle aynıdır
  - Düşük karmaşıklık, yüksek performanslı denetleyici uygulamalarında kullanışlıdır
- Unsegmented paged memory
  - Bellek, sayfalı doğrusal bir adres alanı olarak görülür
  - Belleğin korunması ve yönetimi sayfalama yoluyla yapılır
  - Bazı işletim sistemleri tarafından tercih edilmektedir
- Segmented unpaged memory
  - Bellek, mantıksal adres alanlarının bir koleksiyonu olarak görülür
  - Tek bayt düzeyine kadar koruma sağlar
  - Segment bellekтейken gerekli çeviri tablosunun çip üzerinde olmasını garanti eder
  - Öngörülebilir erişim süreleriyle sonuçlanır
- Segmented paged memory
  - Bölümleme (Segmentation), erişim kontrolüne tabi mantıksal bellek bölümlerini tanımlamak için kullanılır ve bölümler içinde bellek tahsisini yönetmek için sayfalama (paging) kullanılır.
  - UNIX System V gibi işletim sistemleri bu görünümü destekler

Intel  
x86



Memory  
Management





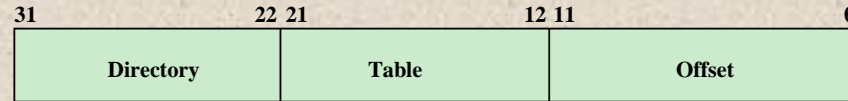
# Segmentation



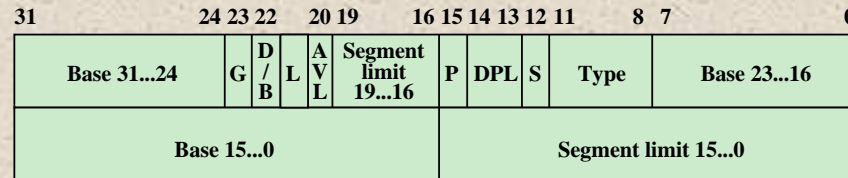
- Her sanal adres, 16-bitlik bir segment referansı ve 32-bitlik bir ofsetten oluşur
  - Segment referansının iki biti koruma mekanizmasıyla ilgili
  - 14 bit segmenti belirtir
- «Unsegmented virtual memory»  $2^{32} = 4\text{Gbytes}$
- «Segmented virtual memory»  $2^{46} = 64\text{ terabytes (Tbytes)}$
- Fiziksel adres alanı, maksimum 4 Gbayt için 32 bitlik bir adres kullanır
- Sanal adres alanı iki bölüme ayrılmıştır
  - Yarısı globaldir ve tüm prosesler tarafından paylaşılır (8K segments \* 4 Gbytes)
  - Geri kalanı yereldir ve her proses için başkadır



TI — Table indicator  
 RPL — Requestor privilege level  
 (a) Segment selector

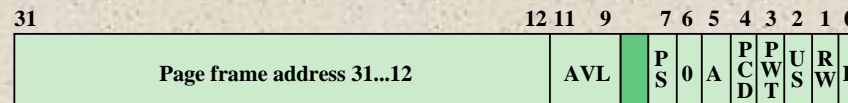


(b) Linear address



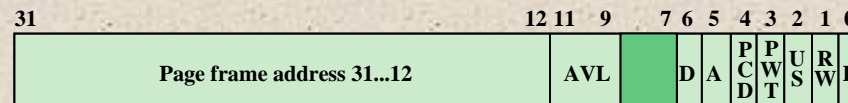
AVL — Available for use by system software  
 Base — Segment base address  
 D/B — Default operation size  
 DPL — Descriptor privilege size  
 G — Granularity  
 L — 64-bit code segment (64-bit mode only)  
 P — Segment present  
 Type — Segment type  
 S — Descriptor type

(c) Segment descriptor (segment table entry)



AVL — Available for systems programmer use  
 P — Page size  
 A — Accessed  
 PCD — Cache disable  
 PWT — Write through  
 US — User/supervisor  
 RW — Read-write  
 P — Present  
 = reserved

(d) Page directory entry



D — Dirty

(e) Page table entry

**Figure 8.20 Intel x86 Memory Management Formats**

## Table 8.5

### x86 Memory Management Parameters (page 1 of 2)

#### Segment Descriptor (Segment Table Entry)

**Base**

Defines the starting address of the segment within the 4-GByte linear address space.

**D/B bit**

In a code segment, this is the D bit and indicates whether operands and addressing modes are 16 or 32 bits.

**Descriptor Privilege Level (DPL)**

Specifies the privilege level of the segment referred to by this segment descriptor.

**Granularity bit (G)**

Indicates whether the Limit field is to be interpreted in units by one byte or 4 KBytes.

**Limit**

Defines the size of the segment. The processor interprets the limit field in one of two ways, depending on the granularity bit: in units of one byte, up to a segment size limit of 1 MByte, or in units of 4 KBytes, up to a segment size limit of 4 GBytes.

**S bit**

Determines whether a given segment is a system segment or a code or data segment.

**Segment Present bit (P)**

Used for nonpaged systems. It indicates whether the segment is present in main memory. For paged systems, this bit is always set to 1.

**Type**

Distinguishes between various kinds of segments and indicates the access attributes.

# Table 8.5

## x86 Memory Management Parameters (page 2 of 2)



### Page Directory Entry and Page Table Entry

#### **Accessed bit (A)**

This bit is set to 1 by the processor in both levels of page tables when a read or write operation to the corresponding page occurs.

#### **Dirty bit (D)**

This bit is set to 1 by the processor when a write operation to the corresponding page occurs.

#### **Page Frame Address**

Provides the physical address of the page in memory if the present bit is set. Since page frames are aligned on 4K boundaries, the bottom 12 bits are 0, and only the top 20 bits are included in the entry. In a page directory, the address is that of a page table.

#### **Page Cache Disable bit (PCD)**

Indicates whether data from page may be cached.

#### **Page Size bit (PS)**

Indicates whether page size is 4 KByte or 4 MByte.

#### **Page Write Through bit (PWT)**

Indicates whether write-through or write-back caching policy will be used for data in the corresponding page.

#### **Present bit (P)**

Indicates whether the page table or page is in main memory.

#### **Read/Write bit (RW)**

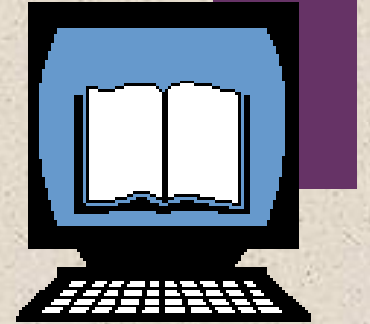
For user-level pages, indicates whether the page is read-only access or read/write access for user-level programs.

#### **User/Supervisor bit (US)**

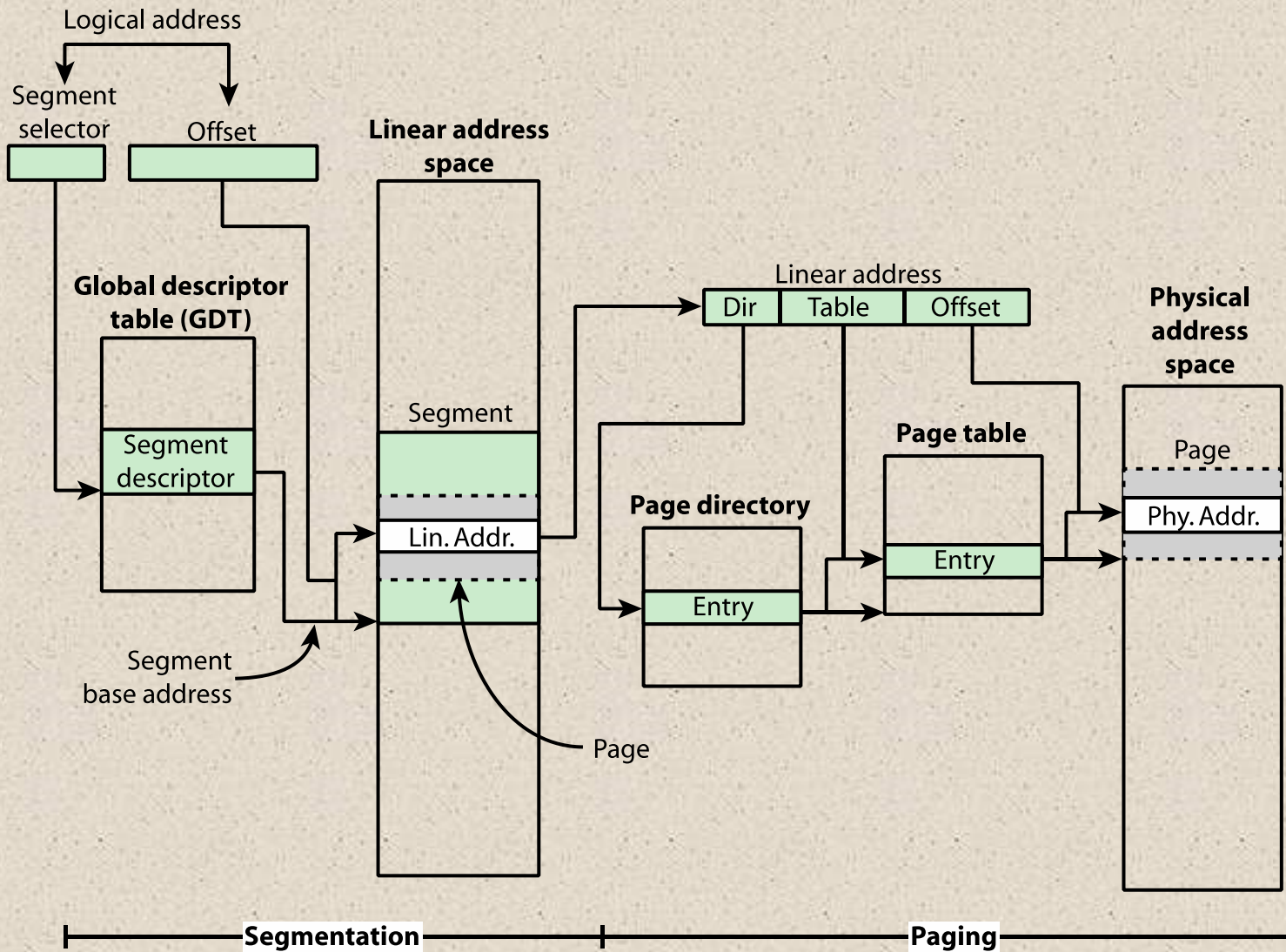
Indicates whether the page is available only to the operating system (supervisor level) or is available to both operating system and applications (user level).



# + Paging




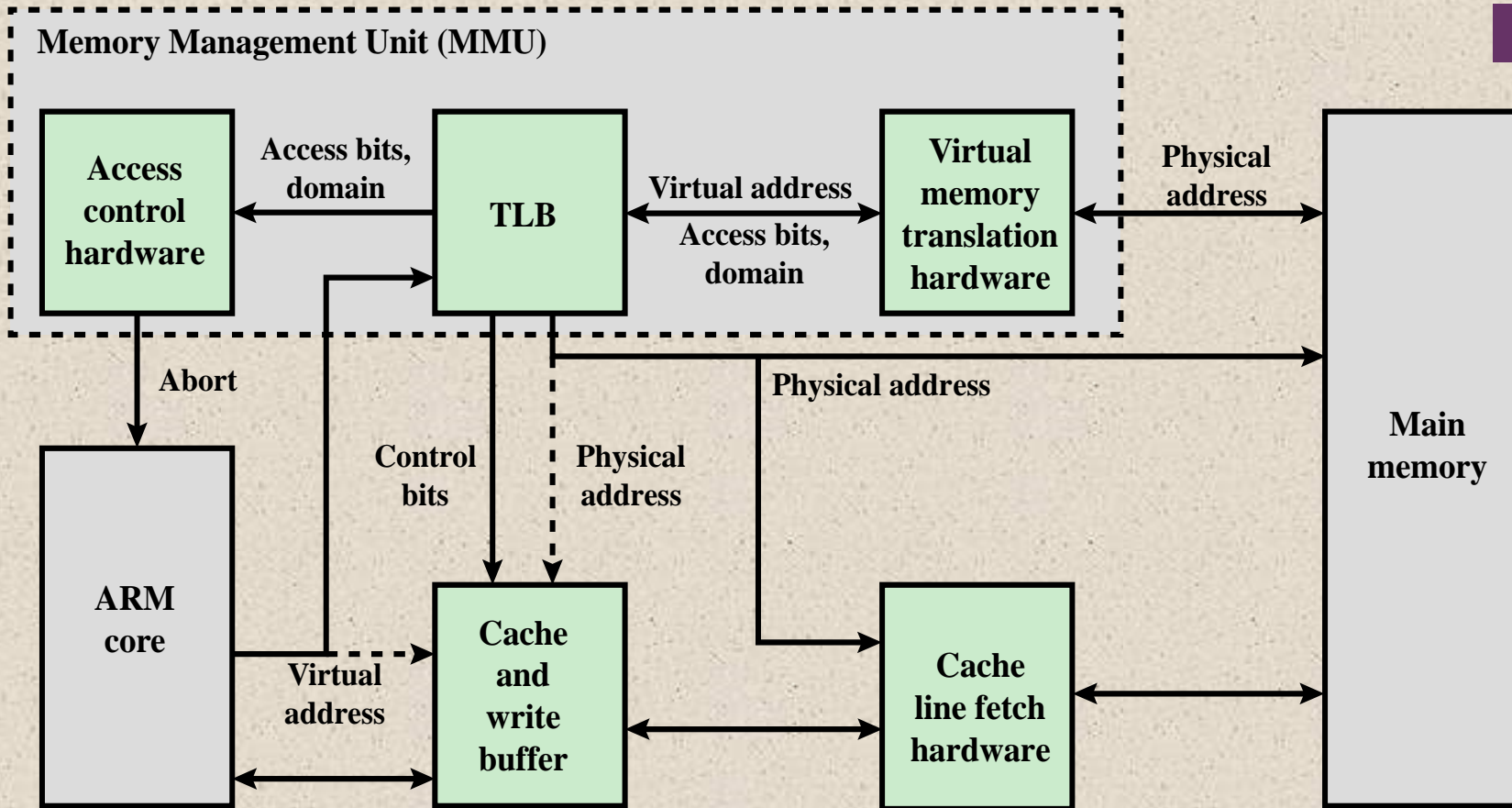
- Segmentasyon devre dışı bırakılabilir
  - Bu durumda lineer adres alanı kullanılır
- Two level page table lookup
  - İlk olarak, sayfa dizini (page directory)
    - 1024 entries max
    - 4 Gbayt lineer belleği 4 Mbaytlık 1024 sayfa grubuna böler
    - Her sayfa tablosunda 4 Kbyte sayfasına karşılık gelen 1024 girdi (*entry*) vardır
    - Tüm prosesler için, proses başına bir sayfa dizini (*page directory*) kullanabilir
    - Geçerli proses için sayfa dizini her zaman bellekte
  - 32 sayfalık tablo girdilerini tutan TLB'yi kullanır
  - İki sayfa boyutu (*page sizes*) mevcuttur, 4k veya 4M



**Figure 8.21 Intel x86 Memory Address Translation Mechanisms**

Şekil 8.21, bölümlleme ve sayfalama mekanizmalarının kombinasyonunu göstermektedir. Netlik sağlamak için TLB ve bellek cache mekanizmaları gösterilmemiştir.

- 
- x86, iki sayfa boyutu sağlayan, 80386 veya 80486'da bulunmayan yeni bir uzantı içerir. Register 4'teki PSE (*page size extension*) biti 1'e ayarlanmışsa, «paging unit», işletim sistemi programlayıcısının bir sayfayı 4 Kbyte veya 4 Mbyte boyutunda tanımlamasına izin verir.
  - 4 Mbyte sayfaların kullanılması, büyük ana bellekler için bellek yönetimi depolama gereksinimlerini azaltır.
  - 4 Kbayt sayfalarda, tam 4 Gbayt'lık bir ana bellek, yalnızca sayfa tabloları için yaklaşık 4 Mbyte bellek gerektirir.
  - 4 Mbyte sayfalarda ise, 4 Kbyte uzunluğunda tek bir tablo, sayfa belleği yönetimi için yeterlidir.

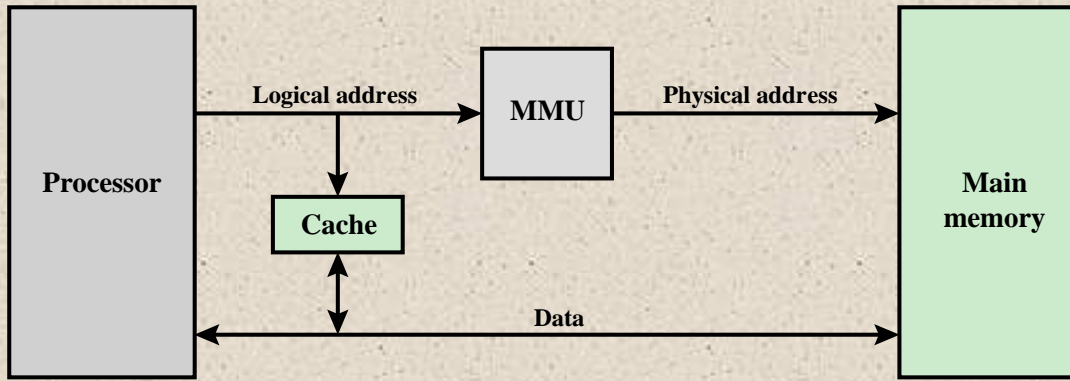


Şekil 8.22, sanal bellek için ARM'daki bellek yönetimi donanımına genel bir bakış sağlar. Sanal bellek çeviri donanımı (*virtual memory translation hardware*), sanaldan fiziksel adreslere çeviri için bir veya iki seviyeli tablo kullanır. TLB, son dönemde kullanılan sayfa tablosu entry'lerinin bir önbellediğidir.

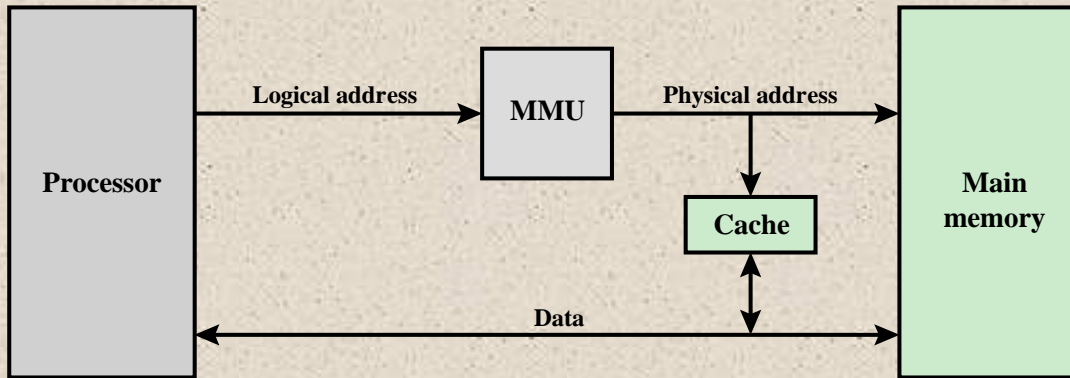
TLB'de bir entry varsa, TLB, okuma veya yazma işlemi için doğrudan ana belleğe bir fiziksel adres gönderir. Bölüm 4'te açıklandığı gibi, veri, işlemci ve ana bellek arasında önbellek aracılığıyla değiştirilir. Mantıksal bir önbellek organizasyonu kullanılıyorsa (Şekil 4.7a), bu durumda ARM, doğrudan önbellege adres vermenin yanı sıra, bir «cache miss» meydana geldiğinde bunu TLB'ye sağlar. Fiziksel bir önbellek organizasyonu kullanılıyorsa (Şekil 4.7b), TLB fiziksel adresi önbellege sağlamalıdır.

**Figure 8.22 ARM Memory System Overview**





(a) Logical Cache



(b) Physical Cache

**Figure 4.7 Logical and Physical Caches**

Sanal adresler kullanıldığında, sistem tasarımcısı önbellege şekilde görüldüğü gibi işlemci ile MMU arasına veya MMU ile ana bellek arasına yerleştirmeyi seçebilir

Sanal önbellek (**logical cache**) olarak da bilinen mantıksal önbellek (**virtual cache**), verileri sanal adresleri kullanarak depolar. İşlemci, MMU'dan geçmeden önbellege doğrudan erişir. Fiziksel bir önbellek ise, ana bellek fiziksel adreslerini (**physical addresses**) kullanarak verileri depolar.

Mantıksal önbellegen bariz bir **avantajı**, önbellek erişim hızının fiziksel bir önbellekten daha hızlı olmasıdır, çünkü önbellek MMU bir adres çevirisi gerçekleştirmeden önce yanıt verebilir.

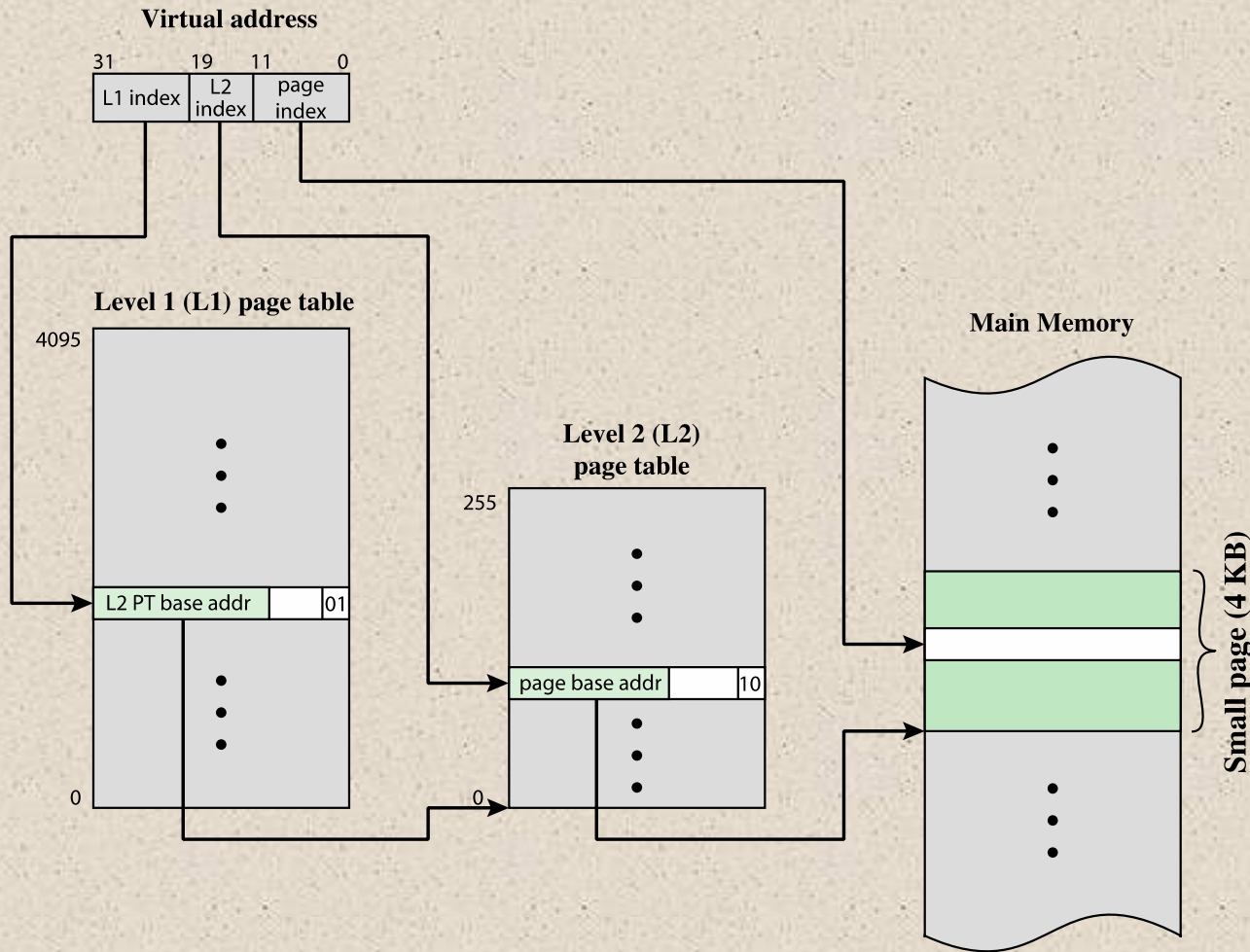
**Dezavantaj**, çoğu sanal bellek sisteminin her uygulamaya aynı sanal bellek adres alanını sağlaması gerçeğiyle ilgilidir. Yani, her uygulama 0 adresinde başlayan bir sanal bellek görür. Bu nedenle, iki farklı uygulamadaki aynı sanal adres, iki farklı fiziksel adresi ifade eder. Bu nedenle, önbellek her bir «application context switch»te tamamen temizlenmeli veya bu adresin hangi sanal adres alanını ifade ettiğini belirlemek için önbellegen her satırına fazladan bit eklenmelidir.



# Virtual Memory Address Translation



- ARM, bölümlere (*sections*) veya sayfalara dayalı bellek erişimini destekler
- Supersections (optional)
  - 16-MB ana bellek bloklarından oluşur
- Sections
  - 1-MB ana bellek bloklarından oluşur
- Large pages
  - 64 kB ana bellek bloklarından oluşur
- Small pages
  - 4 kB ana bellek bloklarından oluşur
- TLB'de yalnızca tek bir entry kullanılırken büyük bir bellek bölgesinin haritalanmasına izin vermek için «**Sections** and **supersections** » desteklenir
- Ana bellekte tutulan çeviri/dönüşüm (*translation* ) tablosunun iki seviyesi vardır :
  - First-level table
    - «section» ve «supersection» dönüşümlerini ve ikinci seviye tabloya işaretçileri tutar
  - Second-level tables
    - Hem büyük hem de küçük sayfa dönüşümlerini tutar



Şekil 8.23, küçük sayfalar için iki seviyeli adres dönüşüm sürecini göstermektedir.

**Figure 8.23 ARM Virtual Memory Address Translation for Small Pages**

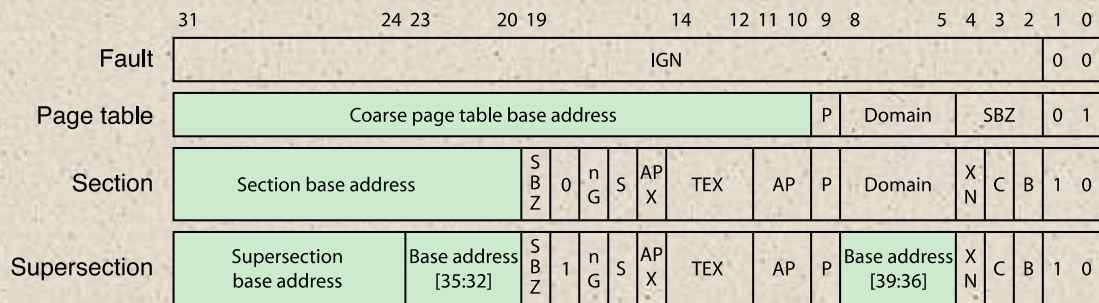
4K tane 32 bit girdili (entry) bir tane seviye 1 (L1) sayfa tablosu vardır.

Her L1 girdisi, 255 tane 32-bit girdili bir seviye 2 (L2) sayfa tablosuna işaret eder.

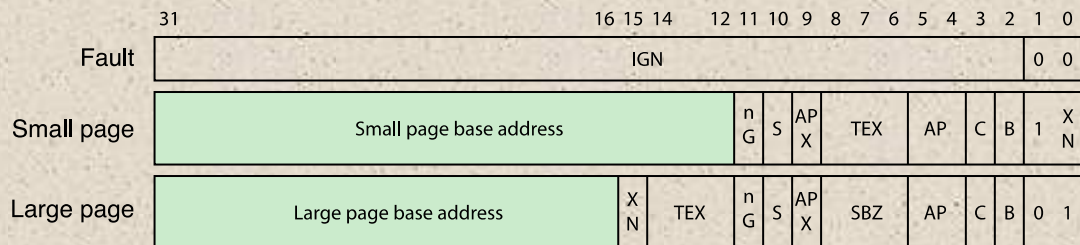
L2 girdilerini her biri, ana bellekte 4 kB'lık bir sayfayı işaret eder.

32 bit sanal adres (virtual address ) şu şekilde yorumlanır:

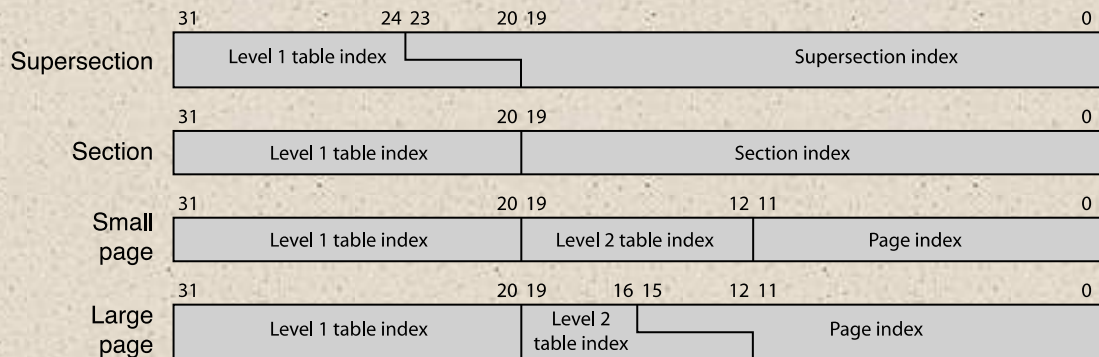
- En değerli 12 bit, L1 sayfa tablosundaki bir indekstir.
- Sonraki 8 bit, ilgili L2 sayfa tablosuna bir indekstir.
- En düşük değerli 12 bit, ana bellekteki ilgili sayfadaki bir baytı indeksler.



(a) Alternative first-level descriptor formats



(b) Alternative second-level descriptor formats



(c) Virtual memory address formats

**Figure 8.24 ARM Memory Management Formats**



**Table 8.6 ARM Memory-Management Parameters**

**Access Permission (AP), Access Permission Extension (APX)**

These bits control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, a Permission Fault is raised.

**Bufferable (B) bit**

Determines, with the TEX bits, how the write buffer is used for cacheable memory.

**Cacheable (C) bit**

Determines whether this memory region can be mapped through the cache.

**Domain**

Collection of memory regions. Access control can be applied on the basis of domain.

**not Global (nG)**

Determines whether the translation should be marked as global (0), or process specific (1).

**Shared (S)**

Determines whether the translation is for not-shared (0), or shared (1) memory.

**SBZ**

Should be zero.

**Type Extension (TEX)**

These bits, together with the B and C bits, control accesses to the caches, how the write buffer is used, and if the memory region is shareable and therefore must be kept coherent.

**Execute Never (XN)**

Determines whether the region is executable (0) or not executable (1).

# + Access Control

- Her tablo girdisindeki AP erişim kontrol bitleri, belirli bir prosesle bir bellek bölgesine erişimi kontrol eder.
- Bir bellek bölgesi şu şekilde belirlenebilir:
  - No access
  - Read only
  - Read-write
- Bölge yalnızca ayrıcalıklı erişim (*privileged access*) olabilir, uygulamalar tarafından değil, işletim sistemi tarafından kullanılmak üzere ayrılabilir
- ARM, bir etki alan kavramını kullanır (ARM employs the concept of a domain)
  - Belirli erişim izinlerine sahip bölümlerin ve / veya sayfaların koleksiyonu
  - ARM mimarisi 16 alanı destekler
  - Birbirinden bazı korumalar sağlarken birden fazla prosesin aynı dönüşüm tablolarını kullanmasına izin verir
- İki tür etki alanı erişimi desteklenir::
  - Clients
    - Bu etki alanını (domain) oluşturan ayrı bölümlerin ve / veya sayfaların erişim izinlerine uyması gereken alanların kullanıcıları
  - Managers
    - Etki alanının davranışını kontrol eder ve bu etki alanındaki tablo girdileri için erişim izinlerini atlar

# + Summary

## Chapter 8

- Operating system overview
  - Operating system objectives and functions
  - Types of operating systems
- Scheduling
  - Long-term scheduling
  - Medium-term scheduling
  - Short-term scheduling
- Intel x86 memory management
  - Address space
  - Segmentation
  - paging

## Operating System Support

- Memory management
  - Swapping
  - Partitioning
  - Paging
  - Virtual memory
  - Translation lookaside buffer
  - Segmentation
- ARM memory management
  - Memory system organization
  - Virtual memory address translation
  - Memory-management formats
  - Access control