İşletim Sistemlerine Giriş

Bellek Yönetimi
(Memory Management)

Bellek Yönetimi

Bellek önemli bir kaynaktır ve dikkatli yönetilmelidir.

İşletim sistemlerinde bellek hiyerarşisini yöneten parçaya **bellek yöneticisi(memory manager)** denilir.

Bellek yöneticisinin görevi, bellein hangi parçalarının kullanımda olduğunu, hangi parçalarının kullanılmadığını izlemek, süreçlere bellek tahsis etme(allocate) , tahsis edilen belleği geri almak ve bellek ile disk arasındaki takas işlemlerini gerçekleştirmektir.

Basit Bellek Yönetimi

Bellek yönetim sistemleri iki temel sınıfa ayrılabilir:

* Çalışma zamanında süreçleri bellek ile disk arasında yer sürekli yer değiştirenler. (takaslama, sayfalama). Süreçlerin bu şekilde disk ile bellek arasında yer değiştirilmesinin nedeni, belleğin boyutunun yetersiz olmasıdır.

* Değiştirme işlemi yapmayanlar.

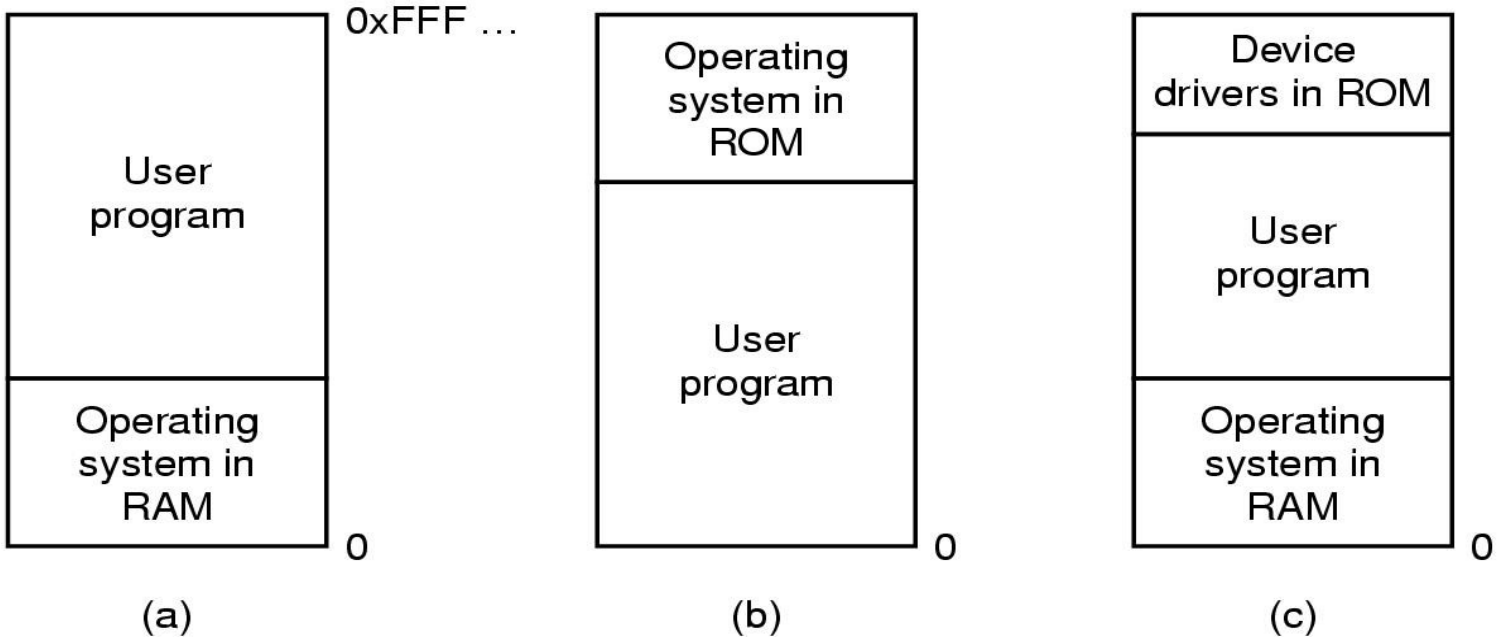
Sayfalama ve Takaslama olmadan Tekdüze programlama

En kolay bellek yönetim şeması aynı anda sadece tek bir programın çalıştırılmasıdır.

Belleği bu program ve işletim sistemi arasında paylaştırmaktır.

Bu modelin 3 değişik sürümü vardır.

Sayfalama ve Takaslama olmadan Tekdüze programlama



a) modeli mainframe sistemlerde ve minicomputer lerde kullanıldı.
b) modeli gömülü(embedded) sistemlerde kullanılır. Palm
c) modeli ilk kişisel bilgisayarlarda kullanılmıştır, sistemin ROM daki kısmına BIOS denilir.

Bu şekildeki sistemde aynı anda tek bir süreç çalışır. Belleğe yüklenir ve çalıştırılır.Yeni program gelirse eskisinin üzerine yazılır ve yenisi çalıştırılır.

Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

Çok basit gömülü sistemler haricinde tekdüze programlama artık kullanılmaz.

Modern işletim sistemleri aynı anda birden fazla sürecin çalışmasına olanak verir. Bir süreç bloklandığında başka bir süreç çalışmaya başlar. Bu şekilde işlemci kullanımı artar.

Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

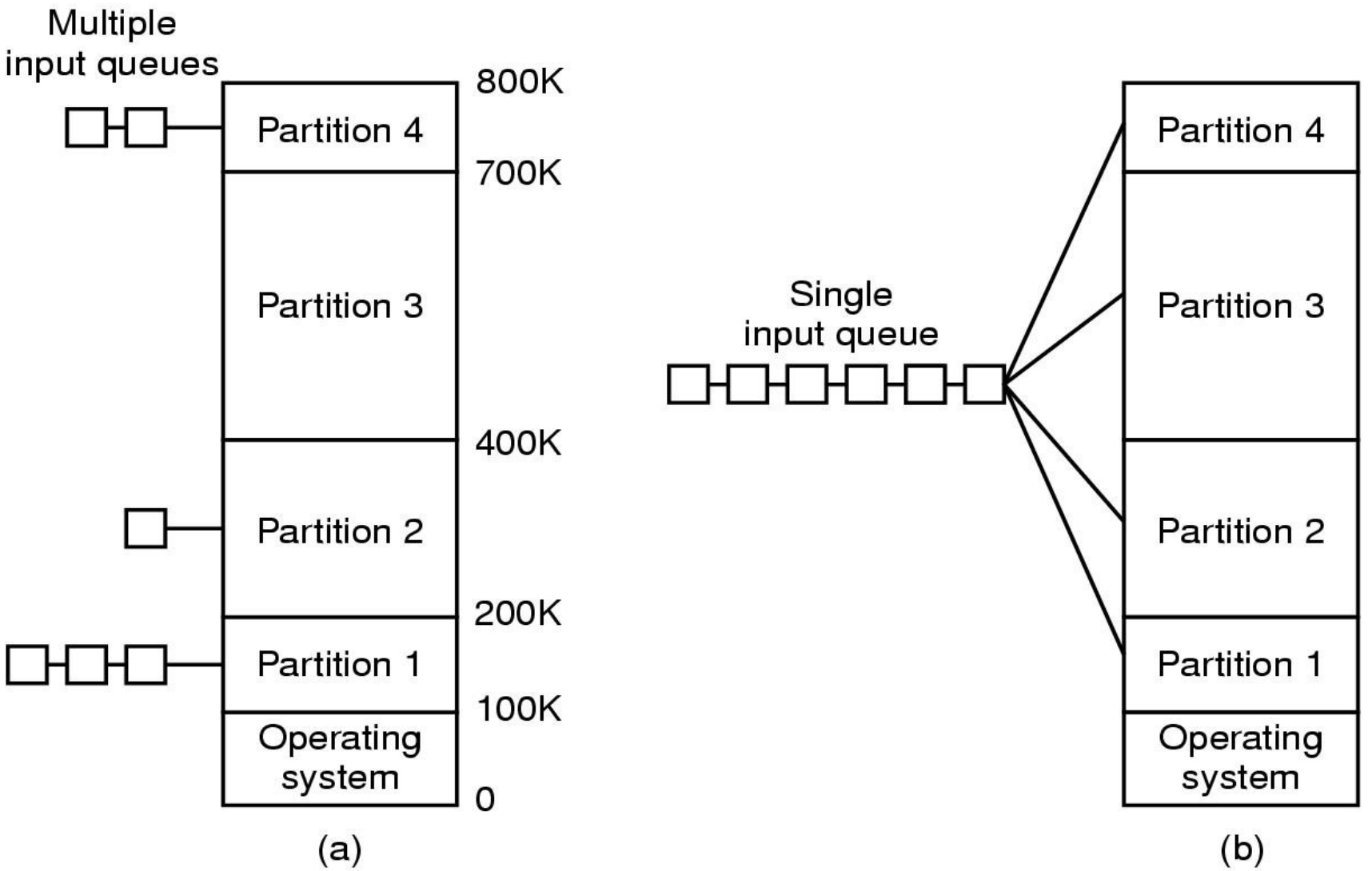
Çoklu programlamayı sağlamanın en kolay yolu *belleği **n** adet bölüme ayırmaktır.* Bu bölümler eşit ya da mümkünse farklı boyutta olmalıdır.

Ayrılan her bölüm için bir süreç kuyruğu bulunur. Bir iş geldiği zaman kendisini tutabilecek olan en küçük boyutlu bölümün girdi kuyruğuna eklenir.

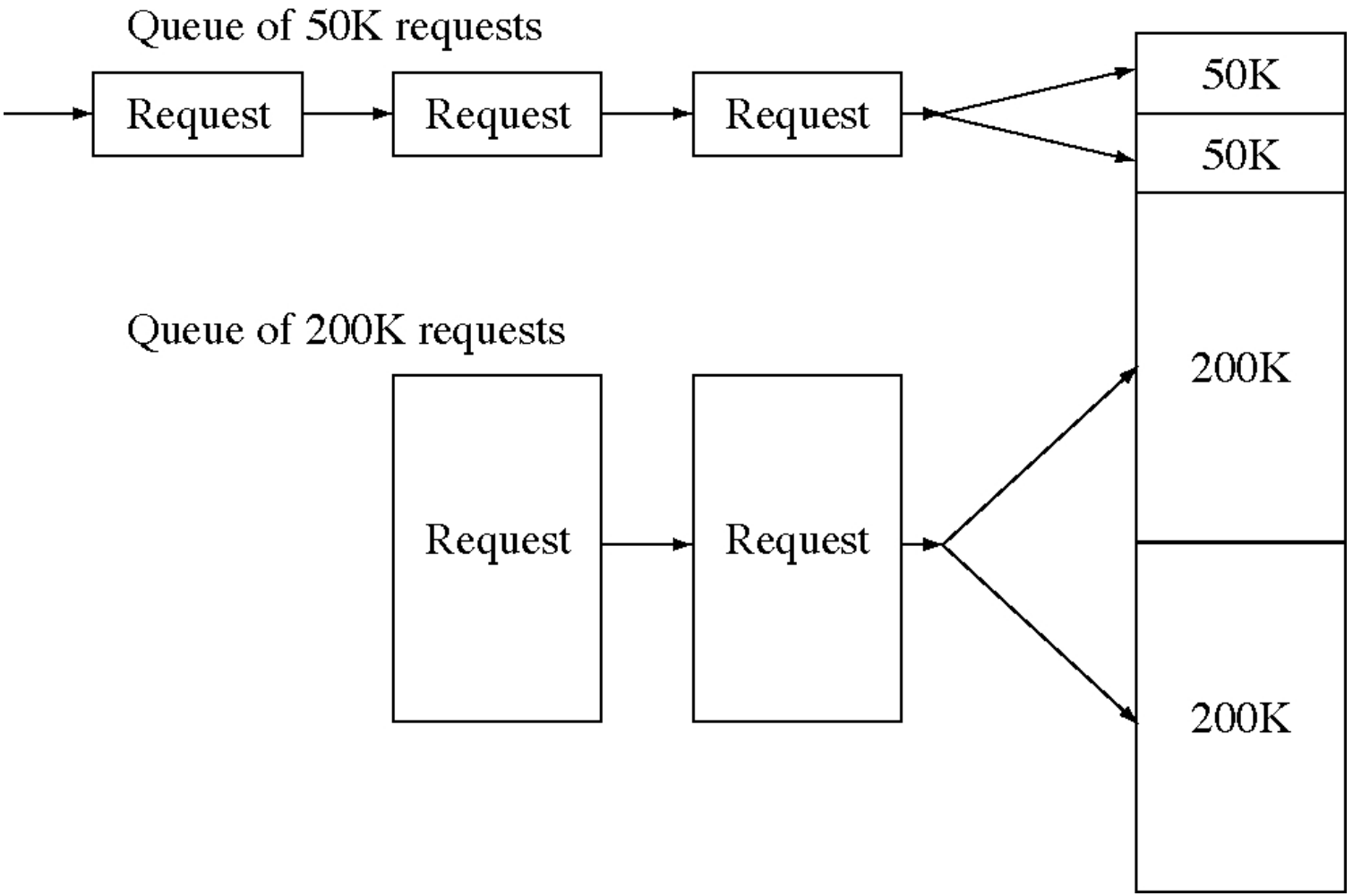
Bir süreç eklendiği bölümün hepsini kullanmaz ise bu kullanılmayan bellek boşuna kullanılmış ve israf edilmiş olur.

Sabit Bölümler ile Çoklu Programlama

(multi programming with Fixed Partitions)



Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

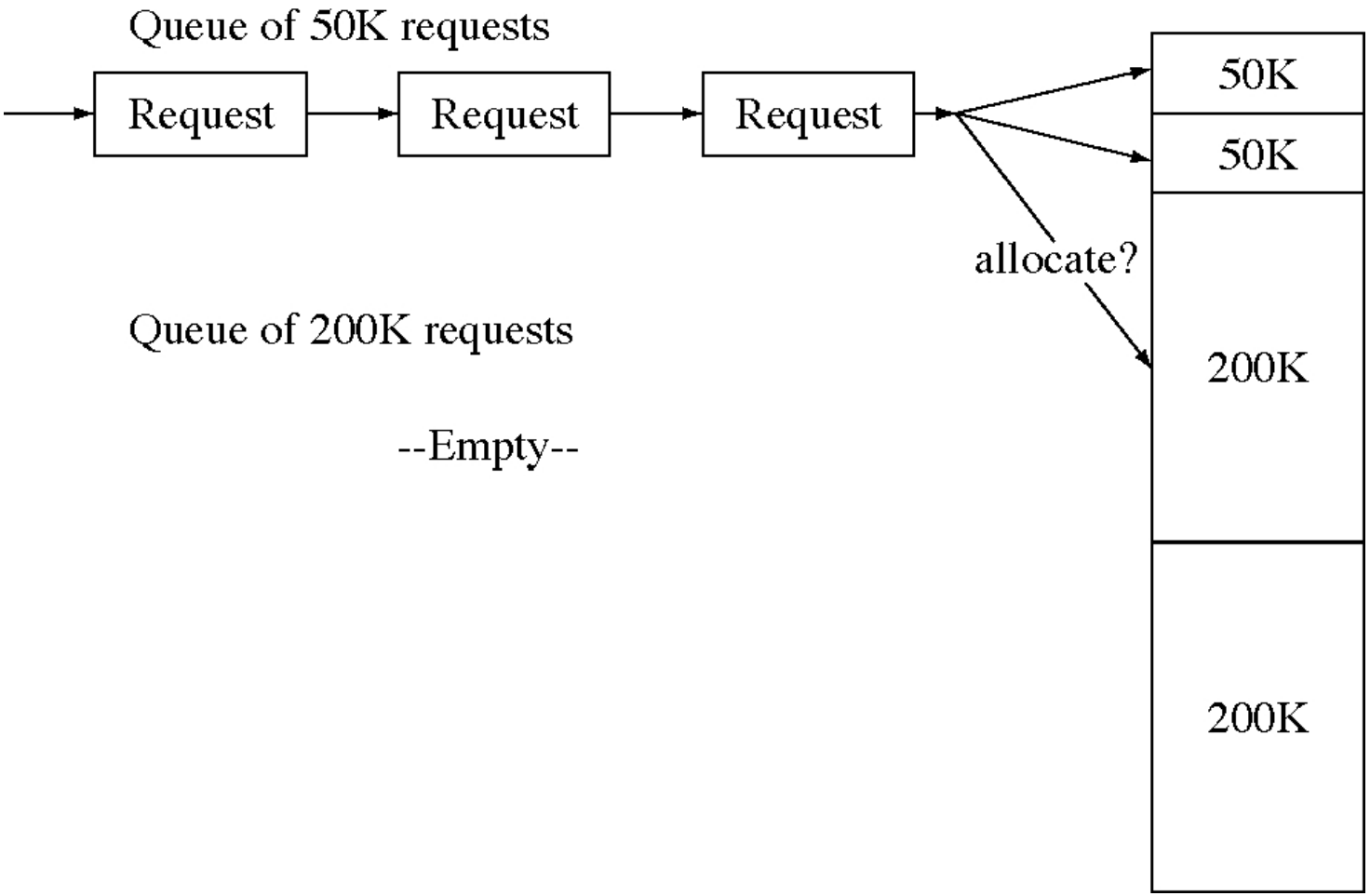


Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

Gelen işleri bu şekilde boyutlarına göre kuyruklara eklediğimizde, büyük boyutlu bellek bölgelerinin kuyrukları boş kalıp kullanılmaz iken, küçük boyutlu bellek bölgelerin kuyrukları dolu olabilir.

Kuyruk dolu olduğunda gelen iş boş yer olmasına rağmen bekler.

Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

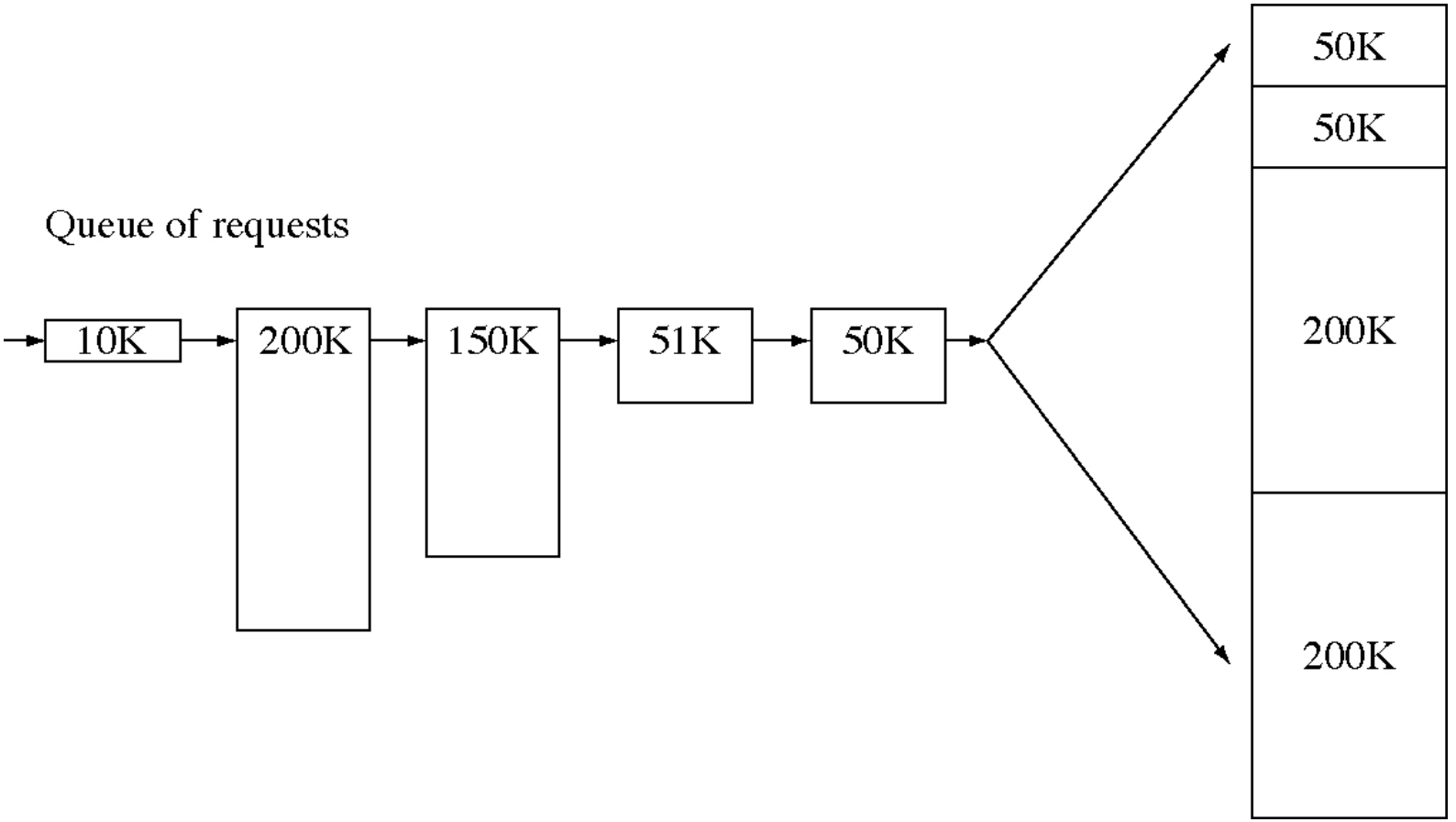


Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)

Bunun yerine tüm süreçler bir kuyruğa konulur ve sıraları geldiğinde uygun bellek gözüne konulurlar.

Bu yöntem OS/360 larda kullanılmıştır.

Sabit Bölümler ile Çoklu Programlama
(multi programming with Fixed Partitions)



Yerdeğiştirme (relocation) ve Koruma (protection)

Çoklu programlamada yerdeğiştirme ve koruma birer problemdir.

Farklı süreçler bellekte farklı adreslerde çalıştıralacaktır. Programın bağlama işlemi yapıldığında, bağlayıcının programdaki kullanıcı prosedürlerinin, kütüphane prosedürlerinin, ana programın adreslerini bilmesi gerekmektedir.

Bu bilgileri bilebilmesi için ve program kodlarını bu adreslere göre düzenleyebilmesi için programın bellekte hangi adrese konulduğunu bilmesi gereklidir.

Yerdeğiştirme (relocation) ve Koruma (protection)

Bu bilgi süreç belleğe yüklendiğinde belirli olduğu için, bağlayıcı sürecin her zaman 0 adresine konulduğunu varsayar.

Örneğin, bağlayıcının oluşturduğu ikili(binary) programın ilk komutu 100 numaralı bellekteki bir prosedürü çalıştırsın. Eğer bu program 100 numaralı adresten itibaren konulmuşsa program doğru çalışmayacaktır.

100 numaralı adreste işletim sistemi bulunmaktadır ve işletim sisteminin olduğu bellek bölgesine geçilir. Bu hatalıdır. Asıl çalıştırılması gerekli olan prosedür 100+100. bellek gözündedir.

Yerdeğiştirme (relocation) ve Koruma (protection)

Bu probleme yer değiştirme problemi denilmektedir.
*Birinci çözüm; program belleğe yüklenirken tüm adres değerlerine programın başlangıç bellek adresi eklenir.

Bu işlemin yapılabilmesi için, programdaki hangi kelimelerin (word) adres, hangilerinin işleç olduğunu bağlayıcının bilmesi gereklidir bu işlem bir liste veya bitmap ile çözülmüştür.

Belleğin korunmasıda son derece önemlidir. Bu işlem için IBM belleği 2Kb lık parçalara ayırmış ve bu parçalara Koruma kodları konulur.

Yerdeğiştirme (relocation) ve Koruma (protection)

*İkinci çözüm; taban yazmacı(base register) ve sınır yazmacı (limit register) adı verilen donanım yazmacı ile çözümdür.

Bir süreç belleğe aktarıldığında, bellekteki başlangıç adresi taban yazmacına, bitiş adresi de sınır yazmacına yazılırlar.

Bir adres istenildiğinde, bu adrese taban yazmacının'ın değeri eklenir; bulunan bellek adresinin sınır yazmacı değerini geçip geçmediği kontrol edilir.
Bu iki yazmaç değeri kullanıcı programları tarafından değiştirilemez.

TAKASLAMA(SWAPPING)

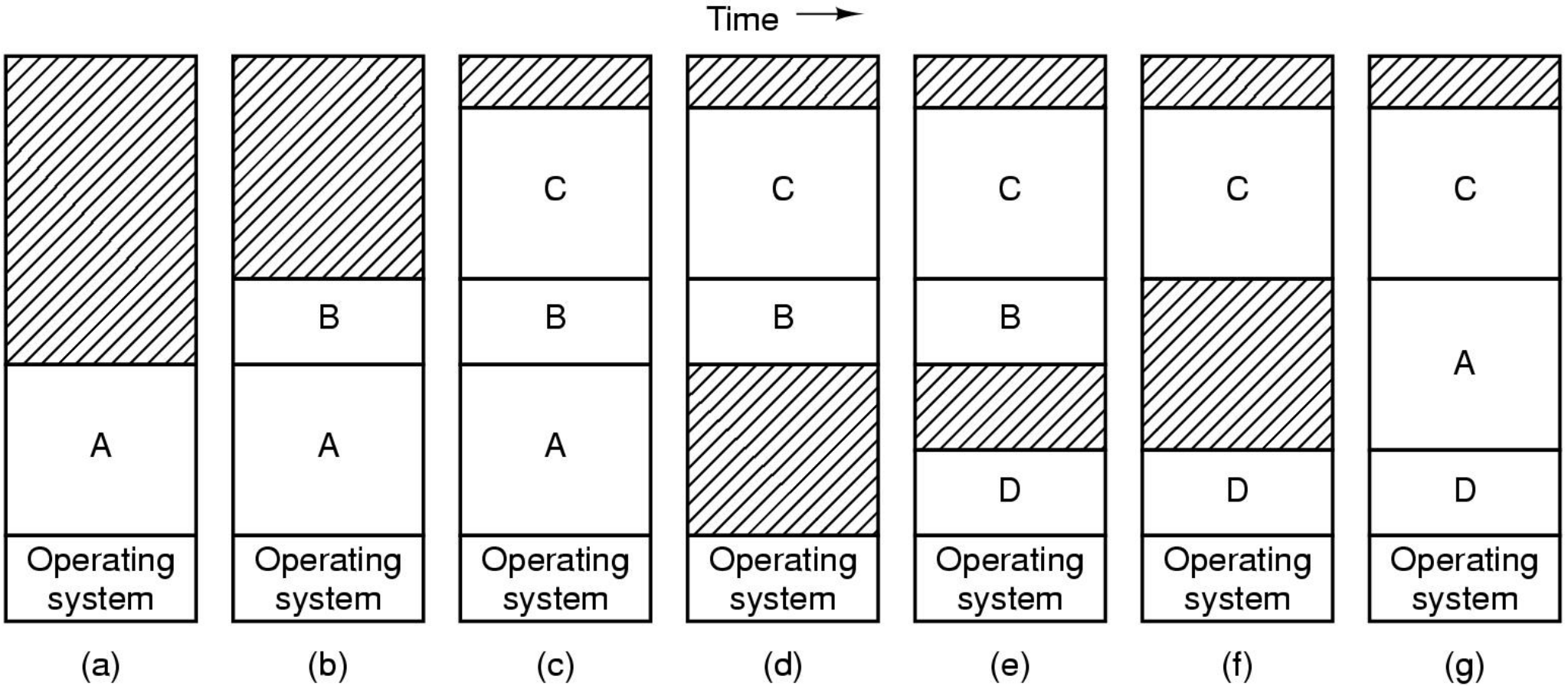
Tüm aktif süreçler bazen bellekte yer olmadığı için bellekte aynı anda bulunamazlar.

Bu süreçlerden bir kısmı diskte bulunmalıdır.

Diskte bulunanan süreçler dinamik olarak belleğe yüklenirler.

Sürecin tümü çalışırken belleğe getirilir, biraz çalışır ve sonra diske kaydedilir. Bu işleme **takaslama(swapping)** denilir.

TAKASLAMA(SWAPPING)



TAKASLAMA(SWAPPING)

Diğer bir strateji **sanal bellek(virtual memory)** dir. Programlara, sadece bir kısmı bellekte olmasına rağmen tamamı bellekteymiş gibi çalışmasına olanak sağlar.

Sayfalama işlemi bellekte delikler oluşturabilir. Tüm süreçleri mümkün olan en aşağıya itmek bellekte kullanılabilir bölümler açar. Bu işleme **disk sıkıştırma (disk compaction)** denilir.

TAKASLAMA(SWAPPING)

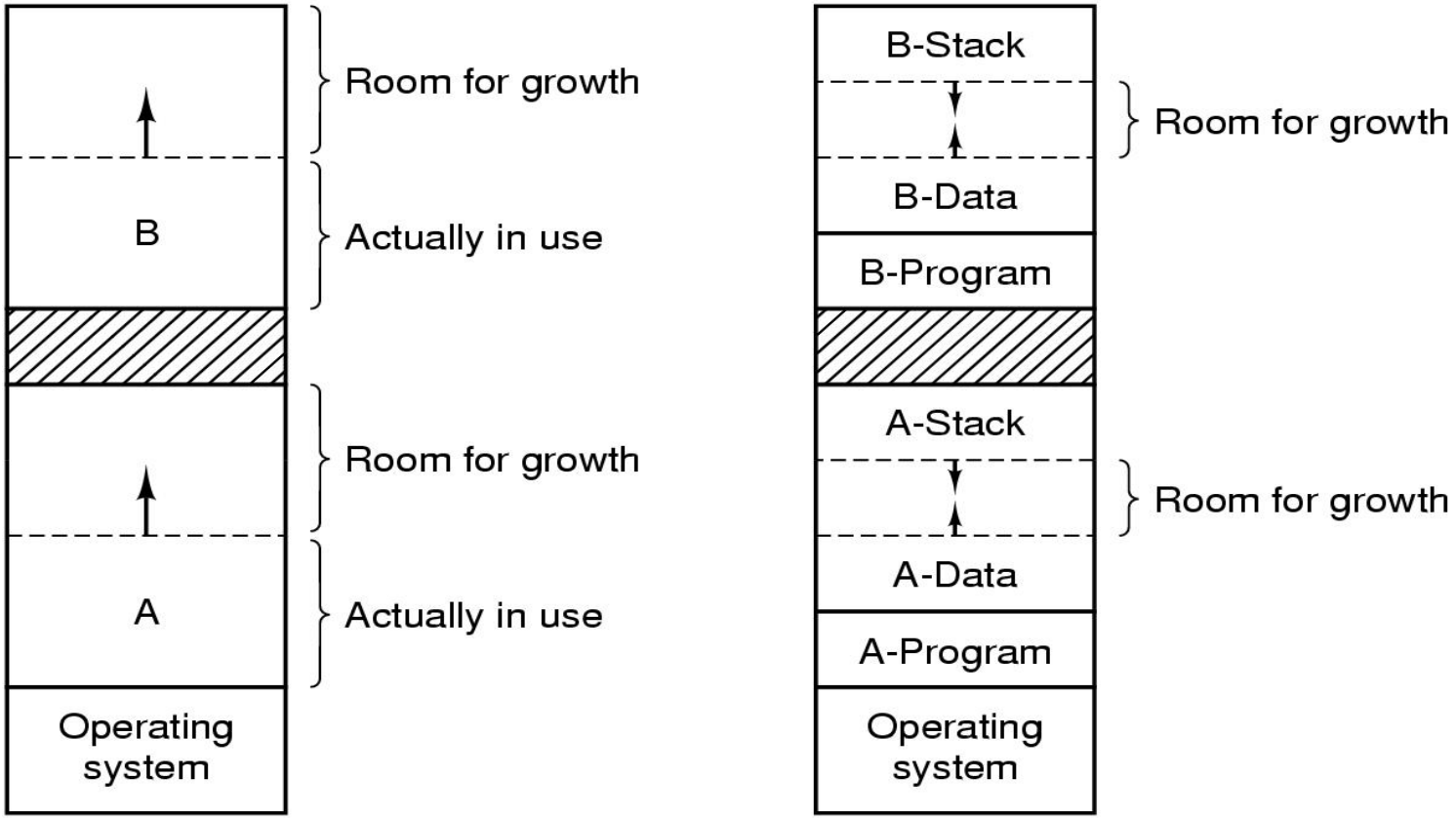
Bir süreç oluşturulduğunda ya da takas ile belleğe geldiğinde ne kadar bellek tahsis edilmelidir?

Eğer tüm süreçler sabit boyutlu olsalardı bu işlem kolay bir şekilde yapılırdır. İşletim sistemi belirli boyutta bir belek alanı tahsis eder.

Bununla birlikte bir sürecin veri bölgesi(heap) ve/veya yığın bölgesi(stack) büyür.

TAKASLAMA(SWAPPING)

Bu dinamik büyümelerde eğer büyümenin gerçekleştiği yönde başka süreç varsa mevcut süreç ya başka bir bellek bölgesine taşınacak ya da takaslama işlemi gerçekleştirilecektir.



BitMap ler ile Bellek Yönetimi

Bellek dinamik olarak tahsis ediliyorsa, işletim sistemi bu işi yönetmelidir.

Bellek kullanımını inceleyebilmek için iki yöntem bulunur:

1. Bitmap ler
2. Boş elemanlar listesi (free list)

BitMap ler ile Bellek Yönetimi

Bitmap lerde, bellek tahsis edilecek minumum birimlere parçalanır.

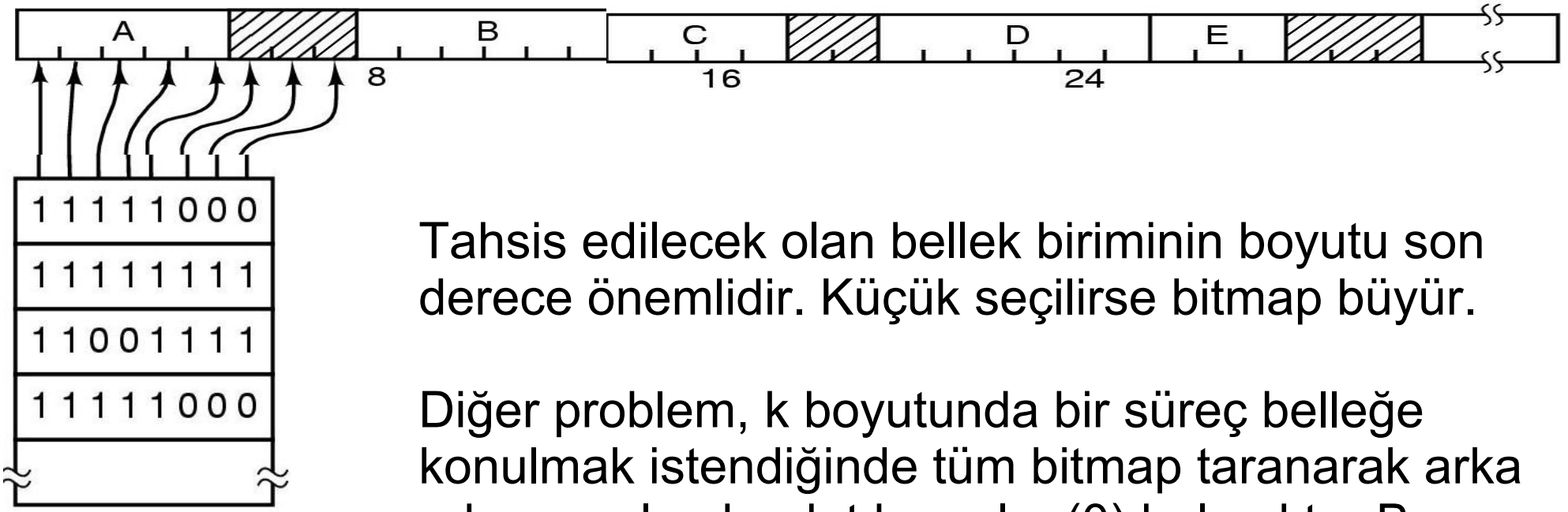
Her birimin sabit boyutu vardır.

Her birim birkaç 8lik(word) kelime olabileceği gibi Kb. larda olabilir.

Bitmap de bellekte bulunanan ve tahsis edilecek olan her birim için 1 bit olur.

Bu bit 0 ise bellek birimi boş, 1 ise bu bellek birimi doludur.

BitMap ler ile Bellek Yönetimi

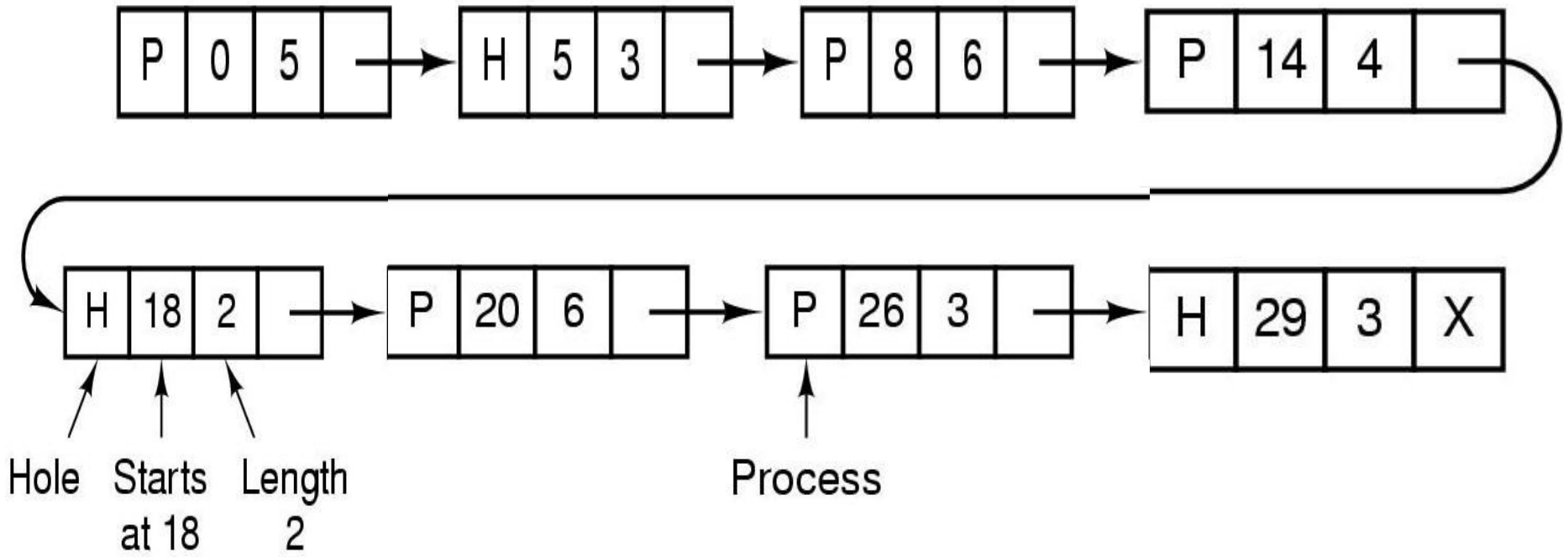


Tahsis edilecek olan bellek biriminin boyutu son derece önemlidir. Küçük seçilirse bitmap büyür.

Diğer problem, k boyutunda bir süreç belleğe konulmak istendiğinde tüm bitmap taranarak arka arkaya gelen k adet boş alan(0) bulmaktır. Bu işlem uzun sürer.

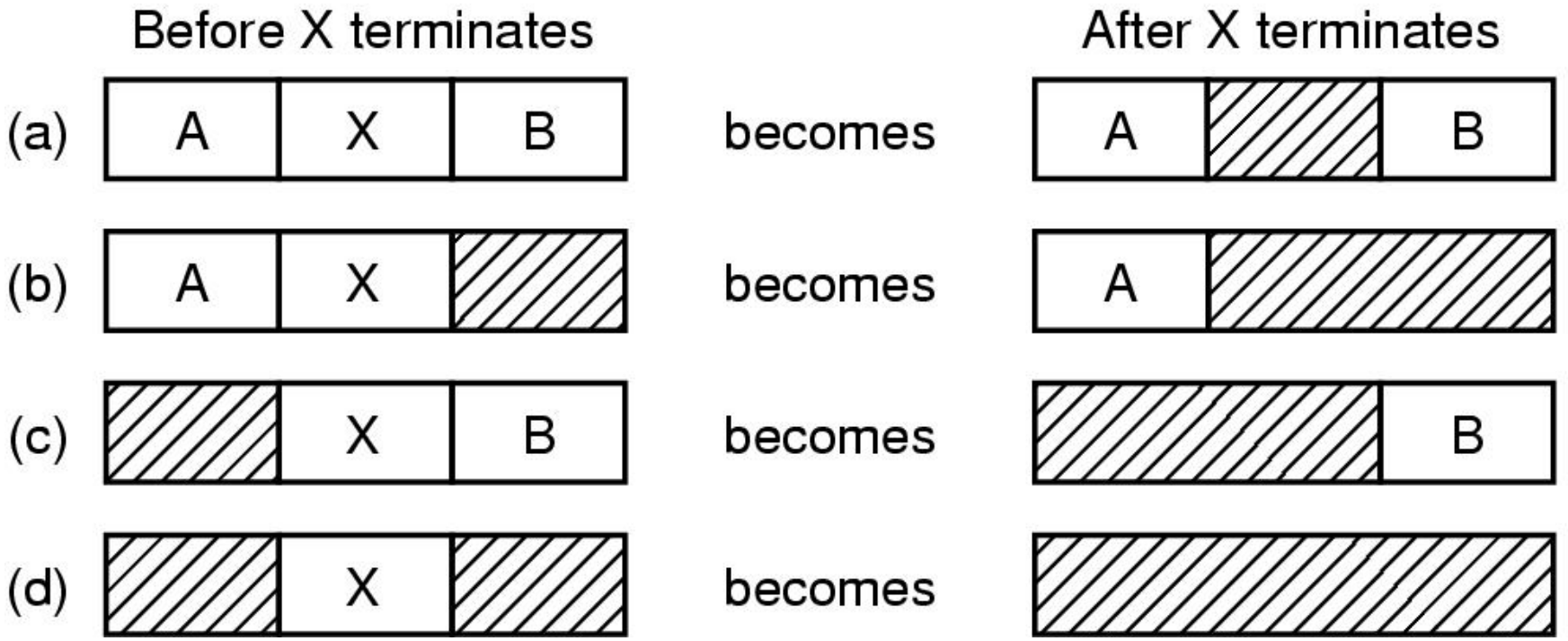
Bağlı Listeler ile Bellek Yönetimi

Diğer yöntem bellekteki dolu ve boş alanların bağlı liste yapısı şeklinde gösterilmesidir.



Bağlı Listeler ile Bellek Yönetimi

Süreç sonlandığında ya da takaslandığında bu algoritma boş alanların birleştirilmesine olanak verdiği için yararlıdır.



Bağlı Listeler ile Bellek Yönetiminde Bellek tahsis etme yöntemleri

Bellek yöneticisinin ne kadar bellek tahsis etmesi gerektiğini bildiğini varsayarak, birkaç algoritmaya bakalım.

1. İlk Uygun Yer Algoritması (First Fit)

Bellek yöneticisi bağlı listeyi başından sonuna doğru tarar. Süreç için gerekli olan bellek boyutunu sağlayan ilk bulduğu boş bellek bölümünü sürece tahsis eder.

Eğer bellek bölgesi sürecin istediğinden daha büyük ise belleği ikiye ayırır. Birinci parça sürecin istediği miktar, ikinci parça kalan kısımdır.

Bağlı Listeler ile Bellek Yönetiminde Bellek tahsis etme yöntemleri

2. Sonraki Uygun Yer Algoritması (Next Fit)

İlk uygun yer algoritması ile aynı şekilde çalışır. Ancak bu algoritma en son bulduğu uygun yer bilgisini saklar.

Bir sonraki aramada bağlı listenin başından değil, saklamış olduğu düğümden sona doğru aramaya başlar.

Bağlı Listeler ile Bellek Yönetiminde Bellek tahsis etme yöntemleri

3. En Uygun Yer Algoritması (Best Fit)

Bu algoritma başından sonuna kadar tüm listeyi tarar. Süreç için gerekli olan bellek boyutuna en uygun olan en küçük hacimli boş bellek alanını listeden bulur. Bulmuş olduğu alanı sürece tahsis eder.

En uygun yer algoritması, ilk uygun yer algoritmasına göre yavaştır ve geride bıraktığı boş bellek bölgeleri ilk uygun yer algoritmasına göre daha kullanışsızdır.

Bağlı Listeler ile Bellek Yönetiminde Bellek tahsis etme yöntemleri

4. Hızlı Uygun Yer Algoritması (Quick Fit)

Bu algoritma en fazla istenilen bellek boyutları için ayrı ayrı bağlı listeler tutar.

Örneğin, bir tabloda 1..n arası girdi vardır. 1. girdi 4KB lık boşlukların listesini göstersin, 2. girdi 8 KB lık, 3. girdi 12 KB lık,...

İşletim Sistemlerine Giriş

Bellek Yönetimi
(Memory Management)