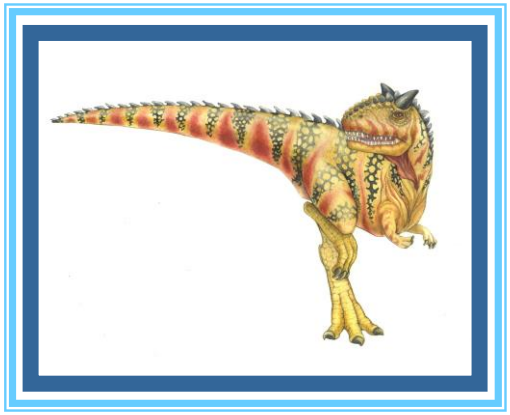
Bölüm 9: Sanal Bellek (Virtual Memory)



Bölüm 9: Sanal Bellek

- Arka plan
- Talep/İstek Sayfalama
- Copy-on-Write
- Sayfa Değiştirme
- Çerçevelerin Tahsisi
- Boşuna çalışma (Thrashing)
- Bellek Haritalı Dosyalar
- Çekirdek Belleğini Tahsis etme
- İşletim Sistemi Örnekleri

Hedefler

- Sanal belleğin tanımı ve faydaları
- İstek/Talep sayfalama kullanarak sayfaların belleğe nasıl yüklendiğinin gösterilmesi.
- FIFO, optimal ve LRU sayfa değiştirme algoritmalarının uygulanması.
- Bir prosesin çalışma kümesinin tanımı ve bunun programın yerleşimi ile nasıl ilişkili olduğunu açıklanması.
- Linux, Windows 10 ve Solaris'in sanal belleği nasıl yönettiğini açıklamak
- C programlama dilinde sanal bir bellek yöneticisi simülasyonu tasarımı

Background

- Kodun yürütülebilmesi için bellekte olması gerekir, ancak programın tamamı nadiren kullanılır
  - Hata kodu, olağandışı rutinler, büyük veri yapıları
- Tüm program kodu aynı anda gerekli değil
- Kısmen-yüklü programı yürütme yeteneğini göz önünde bulundurun
  - Program artık fiziksel hafızanın sınırlarıyla sınırlandırılmıyor
  - Her program çalışırken daha az hafıza alır -> aynı anda çalışan daha fazla program
    - Tepki süresinde veya tamamlanma süresinde bir artış olmadan artan CPU kullanımı ve iş hacmi
  - Programları belleğe yüklemek veya değiştirmek için daha az G/Ç gerekli -> her kullanıcı programı daha hızlı çalışır

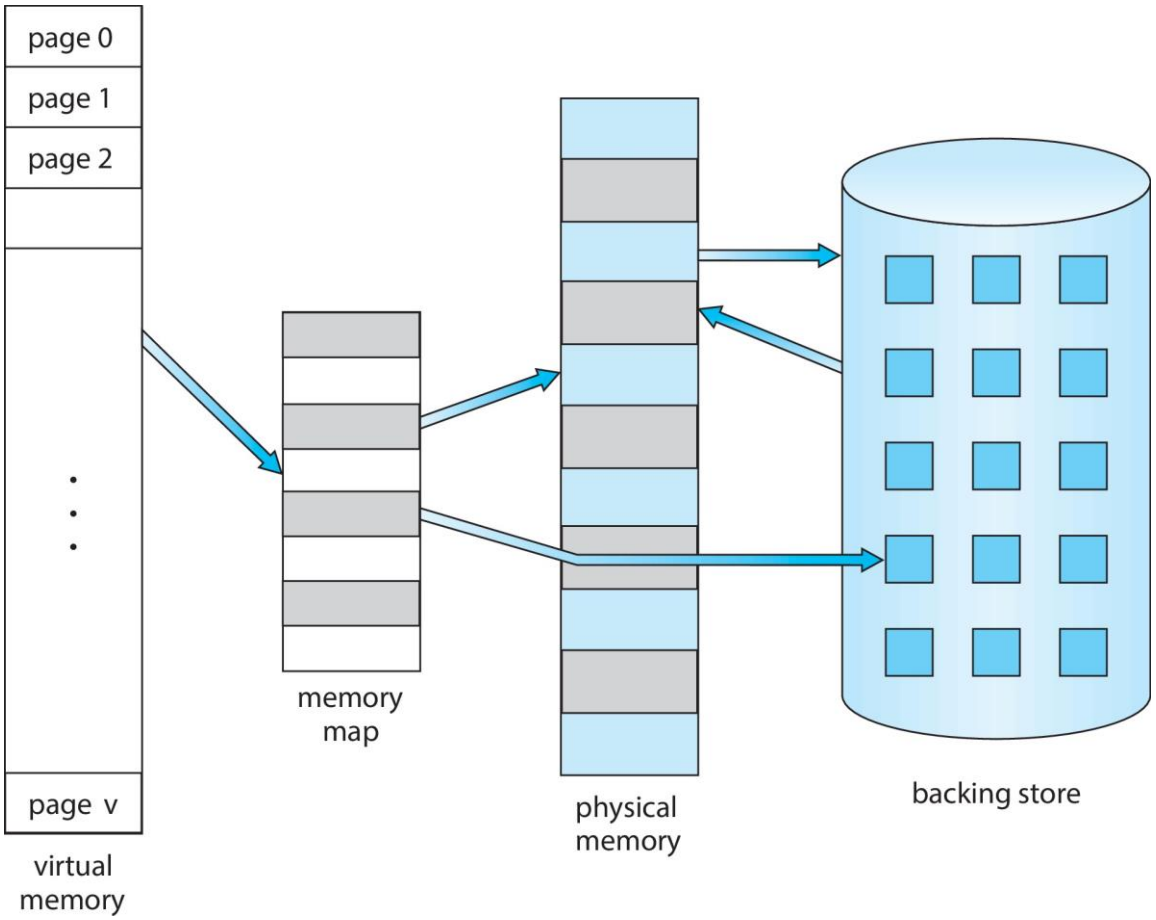
Virtual memory

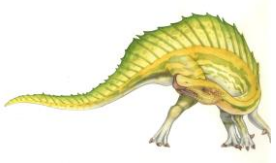
- **Sanal Bellek**–kullanıcı mantıksal belleğinin fiziksel bellekten ayrılması
  - Programın sadece bir bölümünün yürütme için bellekte olması gerekiyor
  - Mantıksal adres alanı bu nedenle fiziksel adres alanından çok daha büyük olabilir
  - Adres alanlarının birkaç proses tarafından paylaşılmasını sağlar
  - Daha verimli proses oluşturmaya izin verir
  - Aynı anda çalışan daha fazla program
  - Prosesleri yüklemek veya takas için daha az G / Ç gerekli

Virtual memory (Cont.)

- **Sanal Adres Alanı**- işlemin bellekte nasıl saklandığına dair mantıksal görünüm
  - Genellikle 0 adresinden başlar, alanın sonuna kadar bitişik adresler
  - Bu arada, fiziksel hafıza sayfa çerçeveleri/ page frames ile düzenlenmiş
  - MMU mantıksal alan ile fiziksel alanı eşlemelidir
- Sanal bellek şu şekilde gerçekleştirilebilir:
  - Talep sayfalama
  - Talep segmentasyonu

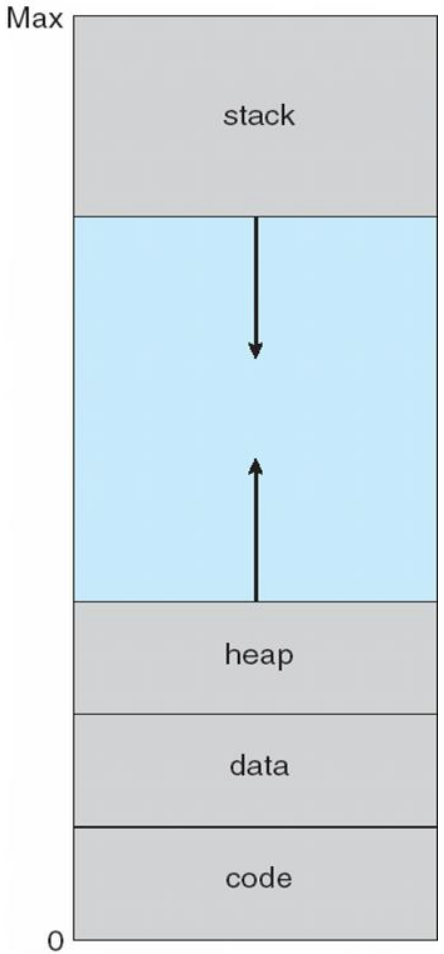
Virtual Memory That is Larger Than Physical Memory



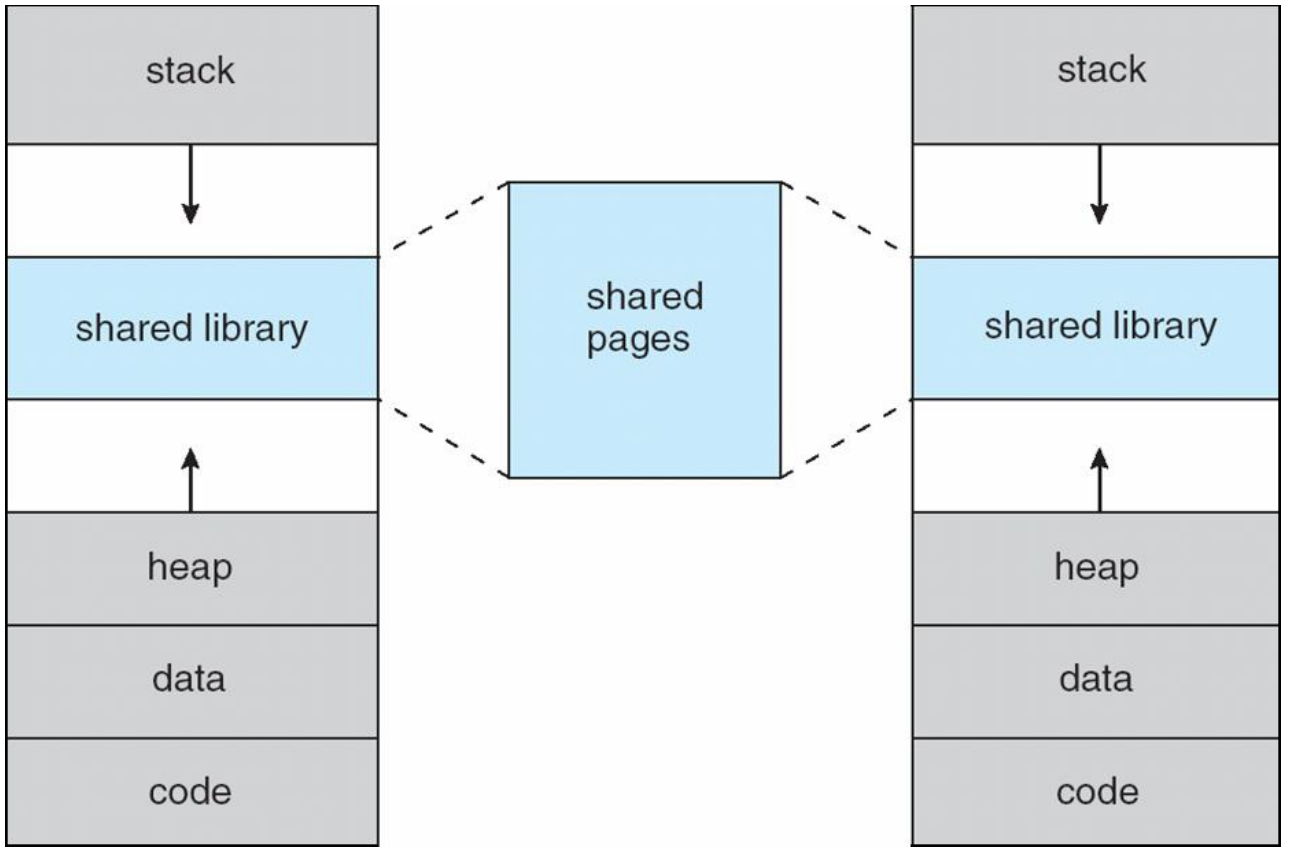


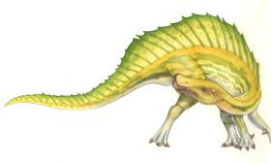
Virtual-address Space

- Genellikle yığın mantıksal adres alanında max'dan başlar ve yığıt-heap "yukarı" büyürken yığın-stack "aşağı" büyür şekilde mantıksal adres alanı tasarlanır
  - Adres alanı kullanımını maksimize eder
  - İkisi arasındaki kullanılmayan adres alanı deliktir (hole)
    - Yığın veya yığıt belirli bir yeni sayfaya büyüyene kadar fiziksel belleğe gerek yoktur
- Büyüme, dinamik olarak bağlanmış kütüphaneler, vb. için boşluk bırakılmış seyrek (**sparse)** adres boşlukları sağlar
- Sanal adres alanına eşleme yoluyla paylaşılan sistem kütüphaneleri
- Sayfaları okuma-yazma ile sanal adres alanına eşleştirerek, paylaşılan hafıza
- Sayfalar `fork()` sırasında paylaşılabilir, hızlı proses oluşturma



Shared Library Using Virtual Memory

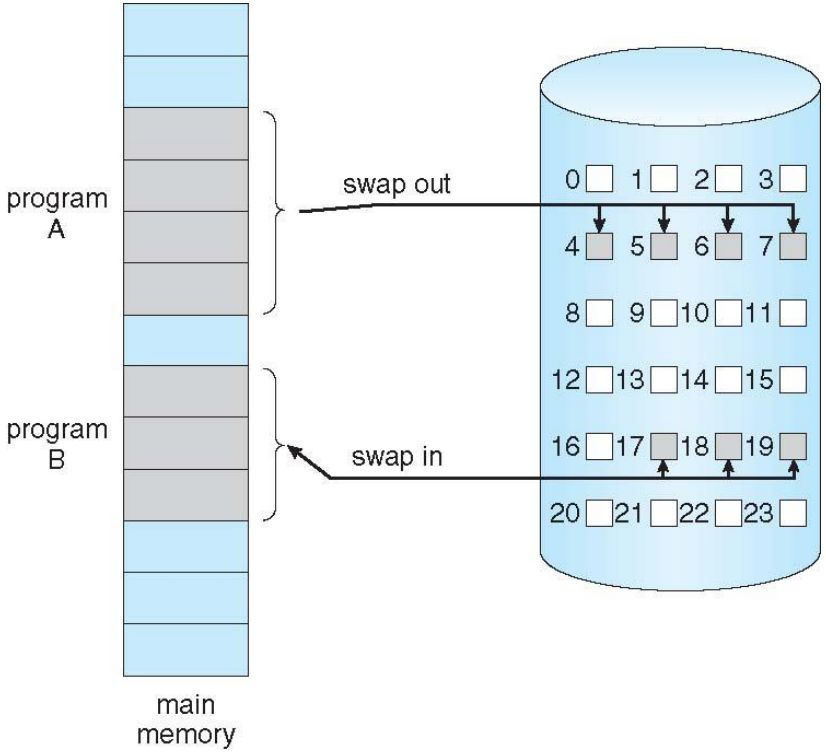




Talep Sayfalama

- Yükleme sırasında tüm proses belleğe getirilebilir
- Veya bir sayfayı yalnızca gerektiğinde belleğe getirin
  - Daha az G / Ç gerekli, gereksiz G / Ç yok
  - Daha az hafıza gerekli
  - Hızlı tepki
  - Daha fazla kullanıcı
- Takas işlemi ile sayfalama gibi (sonraki slayt diyagram)
- Sayfaya ihtiyaç oldu ⇒ referans ver
  - Geçersiz referans ⇒ iptal
  - Bellekte değilse ⇒ belleğe getir
- **Lazy swapper (Tembel Takascı)** – Gerekmedikçe bir sayfayı asla belleğe takas etmez.
  - Swapper that deals with pages is a **pager(sayfalayıcı)**

Demand Paging

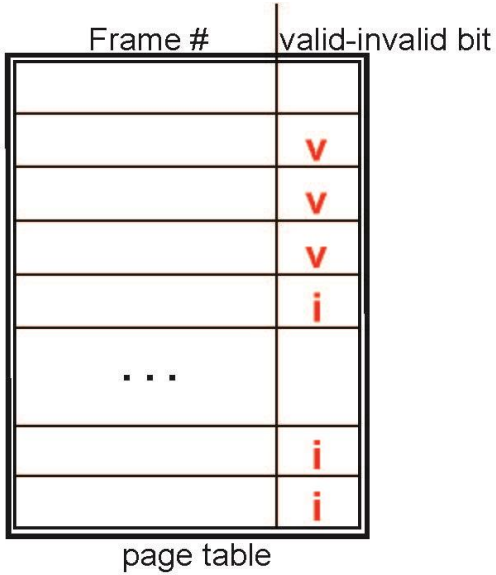


Basic Concepts

- Takas işlemi ile, Sayfalayıcı/pager tekrar değiştirmeden önce hangi sayfaların kullanılacağını tahmin eder.
- Bunun yerine, sayfalayıcı yalnızca bu sayfaları belleğe getirir
- Bu sayfa kümesini nasıl belirleyebilirim?
  - Talep sayfalamayı uygulamak için yeni MMU işlevselliğine ihtiyacınız var
- Gerektiğinde sayfalar zaten bellekte bulunursa
  - Talep-sayfalama yok gibi
- Sayfa gerekirse ve bellekte saklanmıyorsa
  - Sayfayı algılayıp depodan belleğe yüklemeniz gerekir
    - Program davranışını değiştirmeden
    - Programcının kod değiştirmesine gerek kalmadan

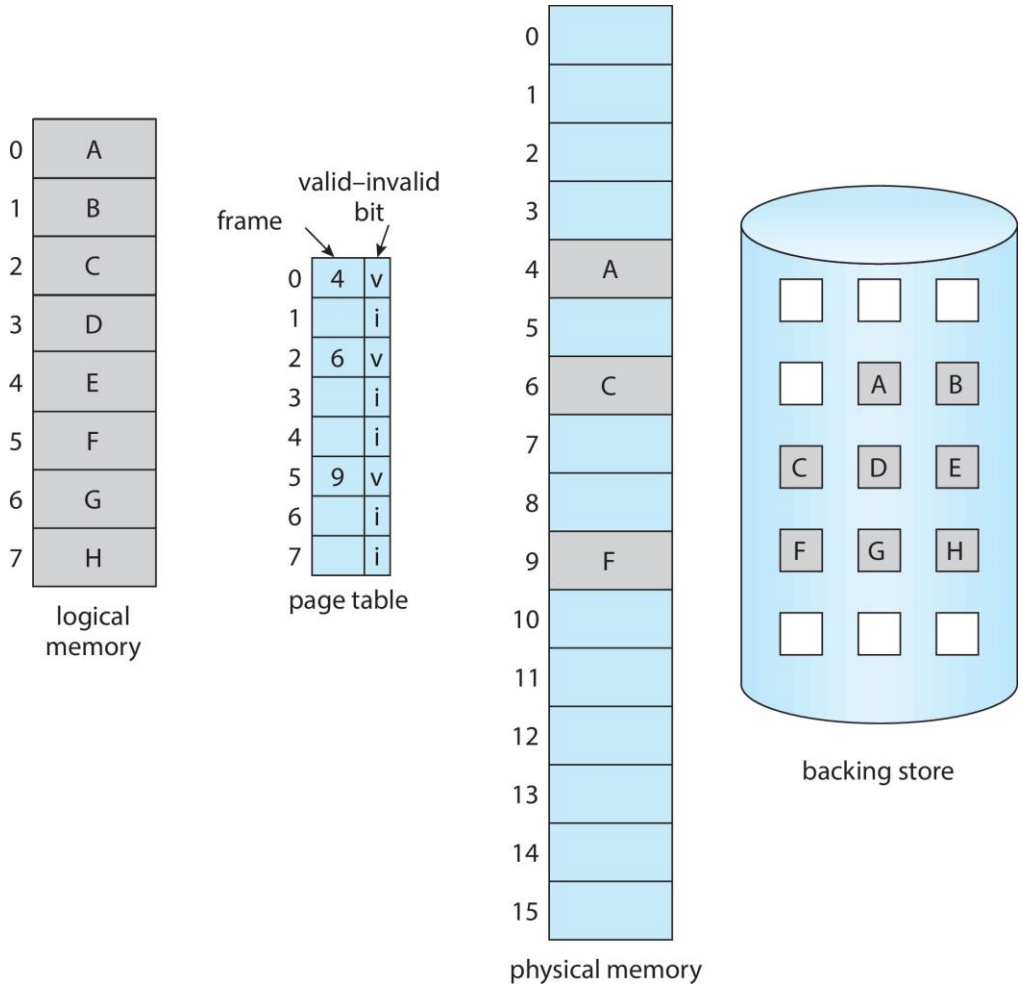
Valid-Invalid Bit

- Her sayfa tablosu girişilere geçerli / geçersiz biti ilişkilendirir (**v** ⇒ in-memory – **memory resident/ var**, **i** ⇒ not-in-memory/yok)
- Initially valid– geçerlilik biti bütün girdiler için **i**'ye ayarlanmıştır
- Sayfa tablosu örneği görünümü:



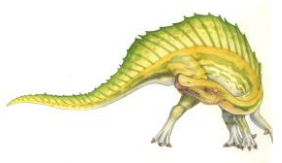
- MMU adres dönüşümü boyunca, eğer geçerli/geçersiz biti sayfa tablosunda **i** ise ⇒ sayfa hatası

Page Table When Some Pages Are Not in Main Memory

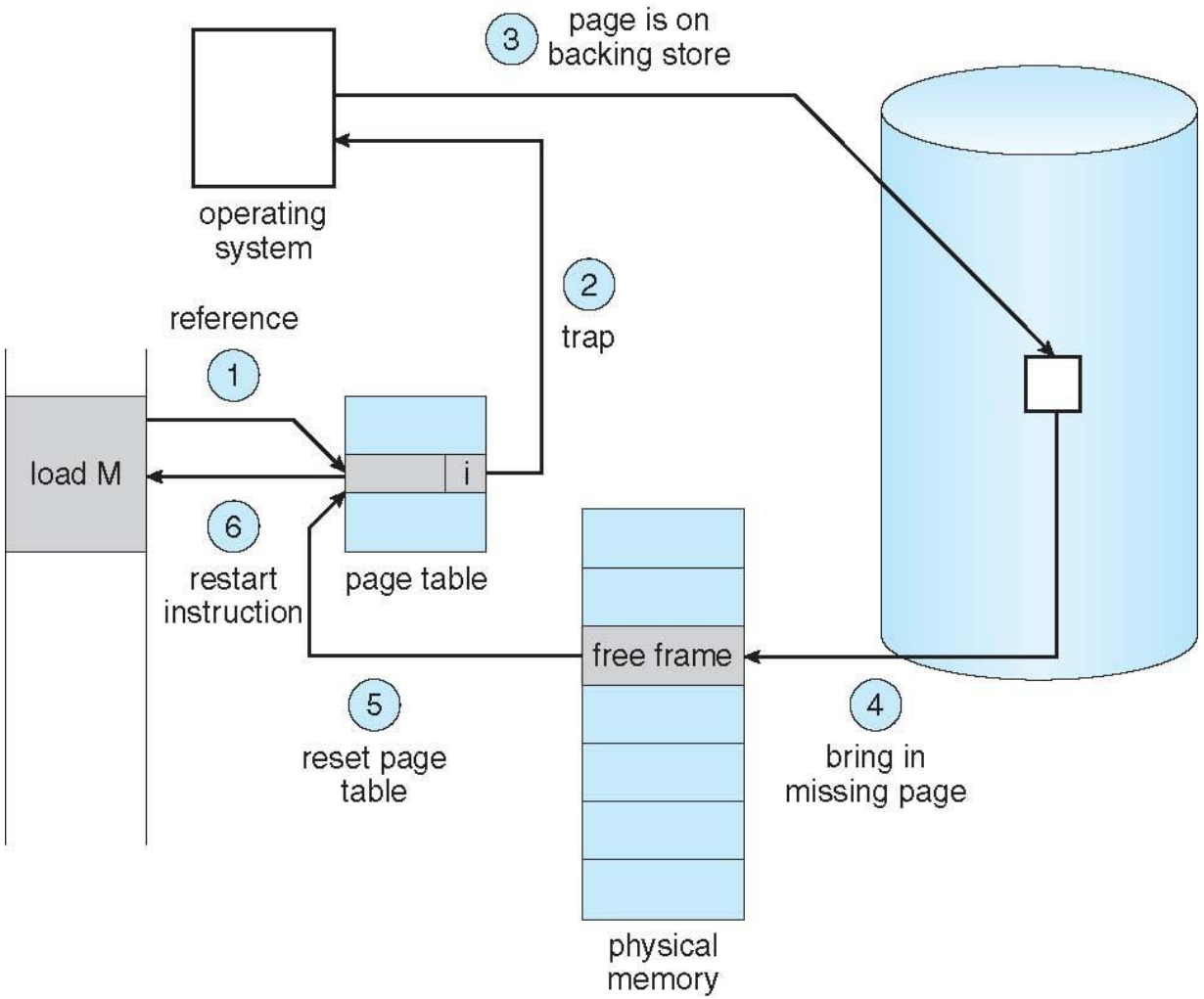


Steps in Handling Page Fault

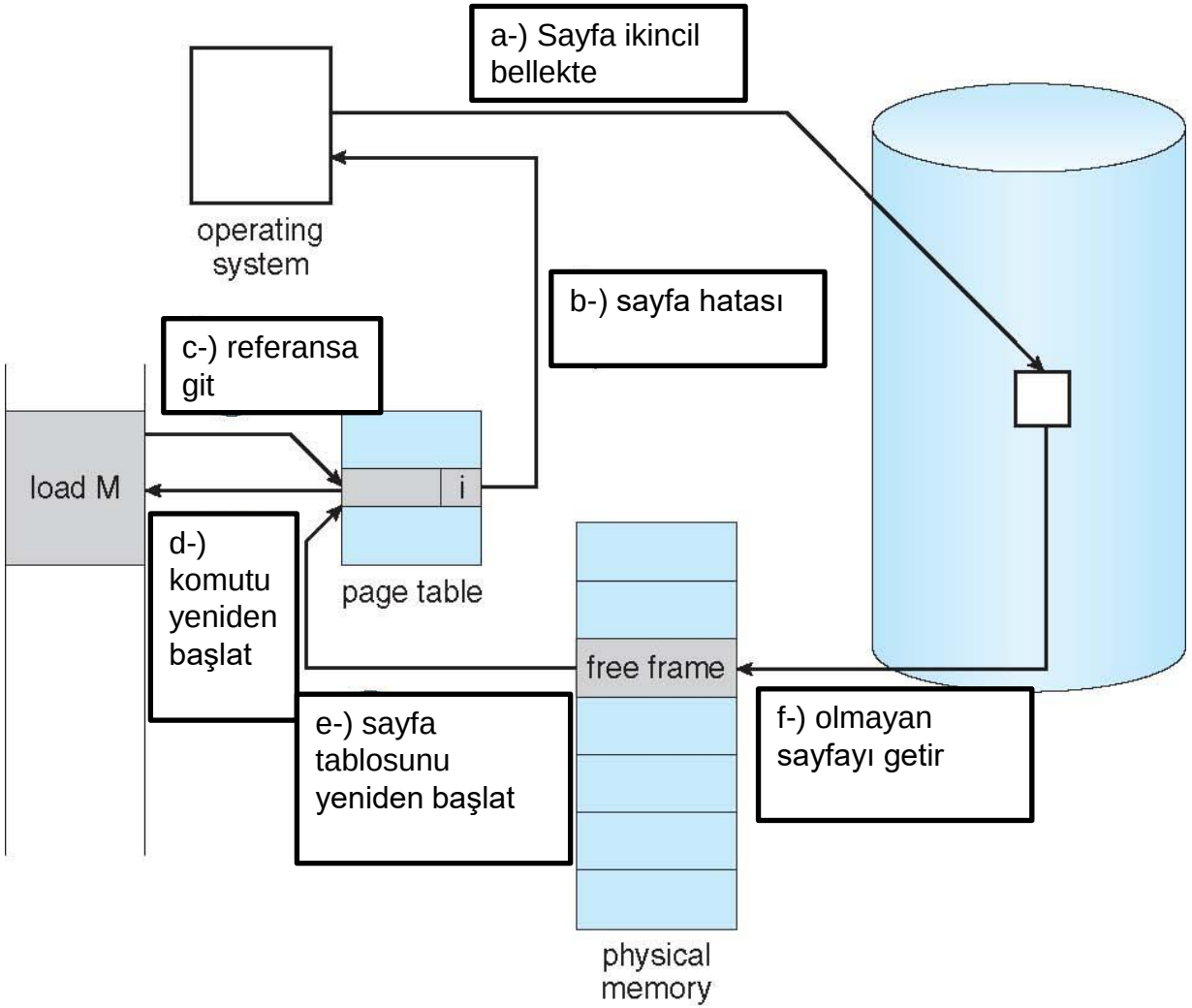
1. Bir sayfaya referans varsa, o sayfaya yapılan ilk referans işletim sistemini hataya düşürebilir.
   - Sayfa hatası
2. İşletim sistemi karar vermek için başka bir tabloya bakar:
   - Geçersiz referans ⇒ iptal
   - Şu an bellekte değil
3. Boş bir çerçeve bul
4. Çizelgelenmiş disk işlemi yerine sayfayı çerçeve ile takas et
5. Sayfayı şimdi bellekte olduğunu belirtmek için tabloları sıfırla Geçerlilik bitini ayarla = **v**
6. Sayfa hatasına neden olan komutu yeniden başlatın



Steps in Handling a Page Fault (Cont.)



Steps in Handling a Page Fault (Cont.)

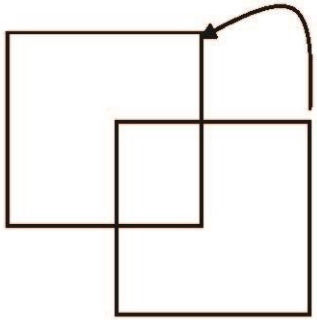


Aspects of Demand Paging

- Ekstrem durum - hafızada sayfa olmadan işlemi başlat
  - İşletim sistemi, komut işaretçisini ilk proses komutuna ayarlar. bellekte yerleşik değil -> sayfa hatası
  - Ve ilk erişimdeki diğer tüm proses sayfaları için
  - **Pure demand paging/saf talep sayfalama**
- Aslında, verilen bir komut birden fazla sayfaya erişebilir -> birden fazla sayfa hatası
  - Bellekten 2 sayı ekleyen ve sonucu belleğe geri kaydeden komutun kodunu almayı ve çözmeyi düşünün
  - **referansın yerleşimi** sebebiyle hata azalabilir
- Talep sayfalama için gerekli donanım desteği
  - Geçerli / geçersiz bit içeren sayfa tablosu
  - İkincil Bellek (**takas alanı** ile takas aygıtı)
  - Komutları yeniden başlatma/restart

Instruction Restart

- Birkaç farklı yere erişebilecek bir komut düşünün
  - Blok hareketi



  - Otomatik artış / azalış yeri
  - Tüm işlem yeniden başlatılsın mı?
    - Kaynak ve hedef üst üste binerse ne olur?

Free-Frame List

- Bir sayfa hatası oluştuğunda, işletim sistemi istenen sayfayı ikincil bellekten ana belleğe getirmelidir.
- Çoğu işletim sistemi, bu tür talepleri yerine getirmek için bir boş çerçeve havuzu - bir boş çerçeve listesi/ **free-frame list** tutar.

- İşletim sistemi tipik olarak talep-üzerine-sıfır-dolu/ **zero-fill-on-demand** olarak bilinen bir teknik kullanarak serbest kareleri tahsis eder - çerçevelerin içeriği tahsis edilmeden önce sıfırlanır.
- Bir sistem başlatıldığında, mevcut tüm hafıza boş çerçeve listesine yerleştirilir.

Stages in Demand Paging – Worse Case

1. İşletim sistemi hata verir
2. Kullanıcı kayıtlarını ve proses durumunu kaydedin
3. Kesmenin bir sayfa hatası olduğunu belirleyin.
4. Sayfa referansının geçerli olduğunu kontrol edin ve sayfanın diskteki yerini belirleyin.
5. Diskten bir boş çerçeveye okuma yapın:
    1. Okuma isteği hizmet verilene kadar bu cihaz için kuyrukta bekleyin.
    2. Cihaz arama ve / veya gecikme süresi için bekleyin.
    3. Sayfanın bir boş çerçeveye aktarımını başlatın

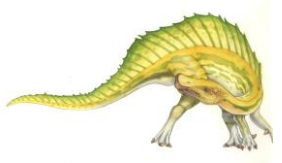
Stages in Demand Paging (Cont.)

6. Beklerken, CPU'yu başka bir kullanıcıya tahsis edin
7. Disk G/Ç alt sisteminden bir kesme al (G / Ç tamamlandı)
8. Diğer kullanıcı için kayıtları ve işlem durumunu kaydet
9. Kesmenin diskten olduğunu belirleyin.
10. Sayfanın artık bellekte olduğunu göstermek için sayfa tablosunu ve diğer tabloları düzeltin
11. CPU'nun tekrar bu işleme tahsis edilmesini bekleyin.
12. Kullanıcı kayıtlarını, işlem durumunu ve yeni sayfa tablosunu geri yükleyin ve ardından kesilen komutu devam ettirin

Performance of Demand Paging

- Üç ana faaliyet
  - Kesmeye hizmet ver - Dikkatli kodlama demek sadece birkaç yüz komut gerekli demektir
  - Sayfayı oku – çok fazla zaman
  - Prosesi yeniden başlatın - tekrar az miktarda zaman
- Sayfa hata oranı $0 \le p \le 1$
  - Eğer $p = 0$ sayfa hatası yok
  - Eğer $p = 1$, her referans hatalı
- Effective Access Time (EAT)

  EAT = (1 – p) x bellek erişimi
  + p (sayfa hatası ek yükü)
  + sayfayı dışarı takas
  + sayfayı içeri takas)



Demand Paging Example

- Bellek Erişim Zamanı= 200 nanoseconds
- Ortalama sayfa-hatası servis süresi = 8 milliseconds
- EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p) x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- Eğer 1000'de bir erişim bir sayfa hatasına neden oluyorsa;

  EAT = 8.2 microseconds.

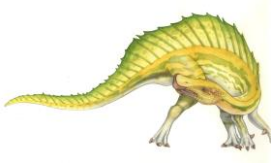
  (8,2 mikrosaniye / 200 nanosaniye = 41 kat yavaşlar)
- Performans düşüşü oranı < % 10 isteniyorsa
  - 220 > 200 + 7,999,800 x p
    20 > 7,999,800 x p
  - p < .0000025
  - <her 400.000 belleğe erişimde bir sayfa hatası

Demand Paging Optimizations

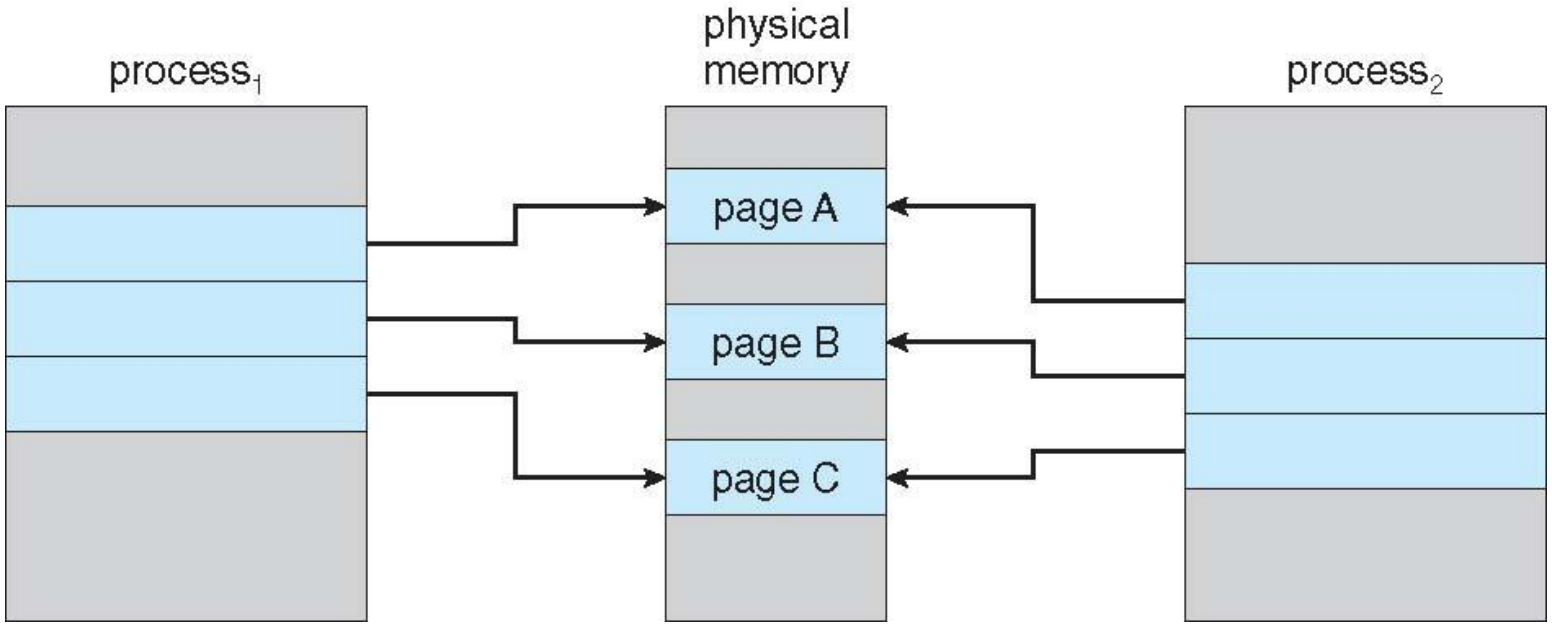
- Aynı aygıtta olsa bile, takas alanı G/Ç'si dosya sistemi G/Ç'den daha hızlıdır
  - Daha büyük parçalara ayrılan takas, dosya sisteminden daha az yönetime ihtiyaç var
- Proses yükleme zamanında alanını takas etmek için proses görüntüsünün tamamını kopyalayın
  - Takas alanını içeri ve dışarı takas yaparak gerçekleştirin
  - Eski BSD Unix'te kullanılırdı
- Diskteki ikili program dosyasından sayfa isteyin, ancak çerçeveyi boşaltırken dışarı sayfalama yerine discard edin (atın).
- Solaris'te ve mevcut BSD'de kullanılır
  - Hala takas etmek için yazmanız gerekiyor
    - Bir dosyayla ilişkilendirilmemiş sayfalar (yığın ve yığıt gibi)– **anonymous memory**
    - Bellekte değiştirilen ancak henüz dosya sistemine geri yazılmayan sayfalar
- Mobil sistemler
  - Genelde takas yapmayı desteklemez
  - Bunun yerine, dosya sisteminden sayfa isteyin ve salt okunur sayfaları geri alın (kod gibi)

Copy-on-Write

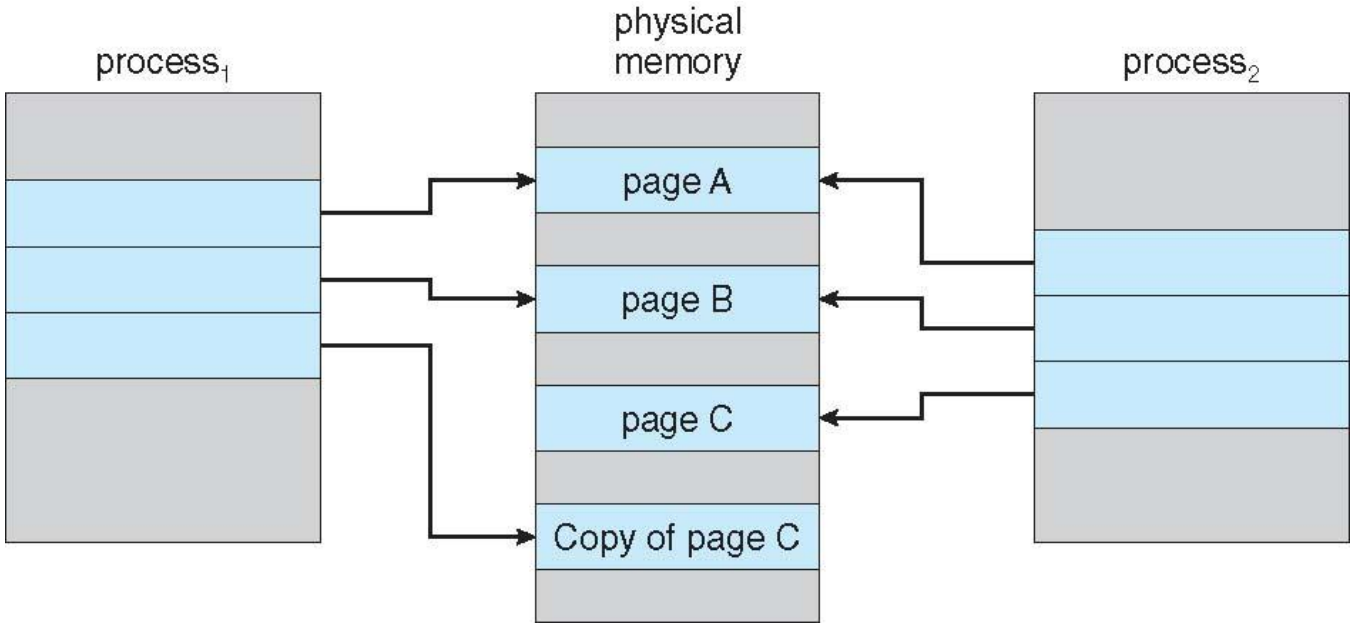
- **Copy-on-Write** (COW) hem ebeveynlerin hem de çocuk proseslerin başlangıçta aynı sayfaları bellekte paylaşmasına izin verir
  - Proseslerden herhangi biri paylaşılan bir sayfayı değiştirirse, ancak o zaman sayfa kopyalanır
- COW, yalnızca değiştirilmiş sayfalar kopyalandığından daha verimli proses oluşturma
- Genel olarak, boş sayfalar talep üzerine **zero-fill-on-demand** oluşan bir havuzdan/ **pool** dağıtılır
  - Havuz, hızlı talep sayfası yürütme için her zaman boş çerçevelere sahip olmalıdır
  - Ayrılmadan önce neden bir sayfa sıfırlanıyor?
- `vfork()` bir çeşit `fork()` sistem çağrısı ebeveyni askıya alır ve ebeveyn üzerine yazılan adres alanını kullanan çocuğa sahiptir.
  - Çocuk prosesin `exec()` çağrısını yürütmek için
  - Verimli



Before Process 1 Modifies Page C



After Process 1 Modifies Page C

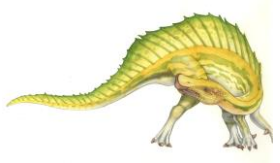


What Happens if There is no Free Frame?

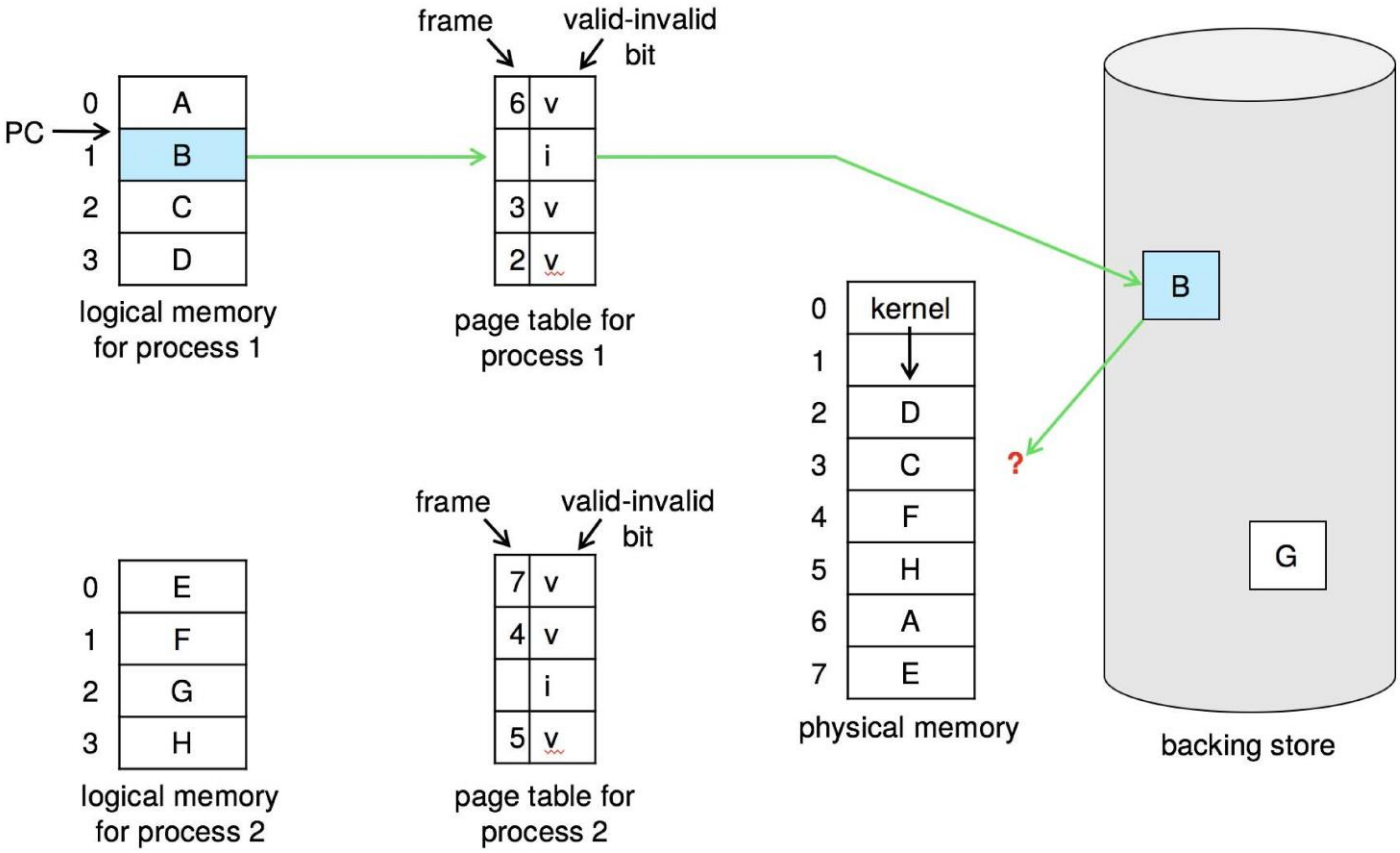
- Boş çerçeveler Proses sayfaları tarafından kullanılmış
- Ayrıca çekirdekden talep, G/Ç tamponları, vb.
- Her birine ne kadar tahsis edilmeli?
- Sayfa değiştirme - bellekte bir sayfa bulun, ancak gerçekte kullanımda değil
  - Algoritma - sonlandırmak? takas etmek? sayfa değiştirilsin mi?
  - Performans - en az sayfa hatasıyla sonuçlanacak bir algoritma isteriz
- Aynı sayfa birkaç kez belleğe getirilebilir

Page Replacement

- Sayfa yer değiştirmesini içerecek şekilde sayfa hatası servis yordamını değiştirerek fazla bellek ayırmayı/ **over-allocation** önleyin
- Sayfa aktarımlarının ek yükünü azaltmak için **modify** (**dirty**) **bit** kullanın - yalnızca değiştirilen sayfalar diske yazılır
- Sayfa değiştirme, mantıksal bellek ile fiziksel bellek arasındaki ayrımı tamamlar; daha küçük bir fiziksel bellekte büyük sanal bellek sağlanabilir



Need For Page Replacement



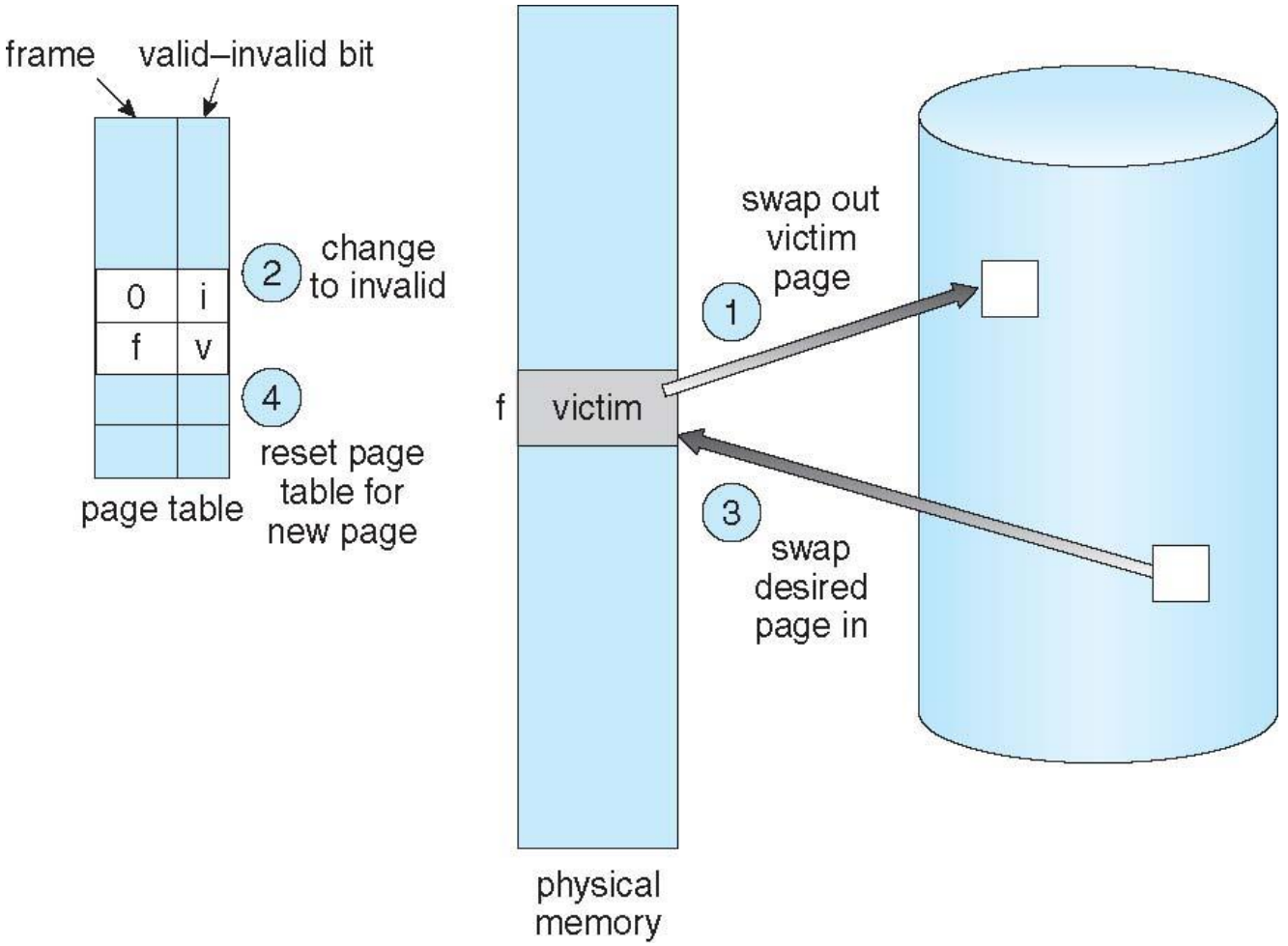
Basic Page Replacement

1. Diskte istediğiniz sayfanın yerini bulun
2. Boş bir çerçeve bulun:
   - Boş bir çerçeve varsa kullanın
   - Boş bir çerçeve yoksa, kurban çerçevesi/**victim frame** seçmek için sayfa değiştirme algoritması kullanın
     - eğer dirty/modifiye edilmiş ise kurban çerçeveyi diske yazın
3. İstediğiniz sayfayı (yeni) boş çerçeveye getirin; sayfa ve çerçeve tablolarını güncelle
4. Tuzağa/hataya neden olan komutu yeniden başlatarak işleme devam edin

sayfa hatası için potansiyel olarak 2 sayfa aktarımı

- “artan EAT”

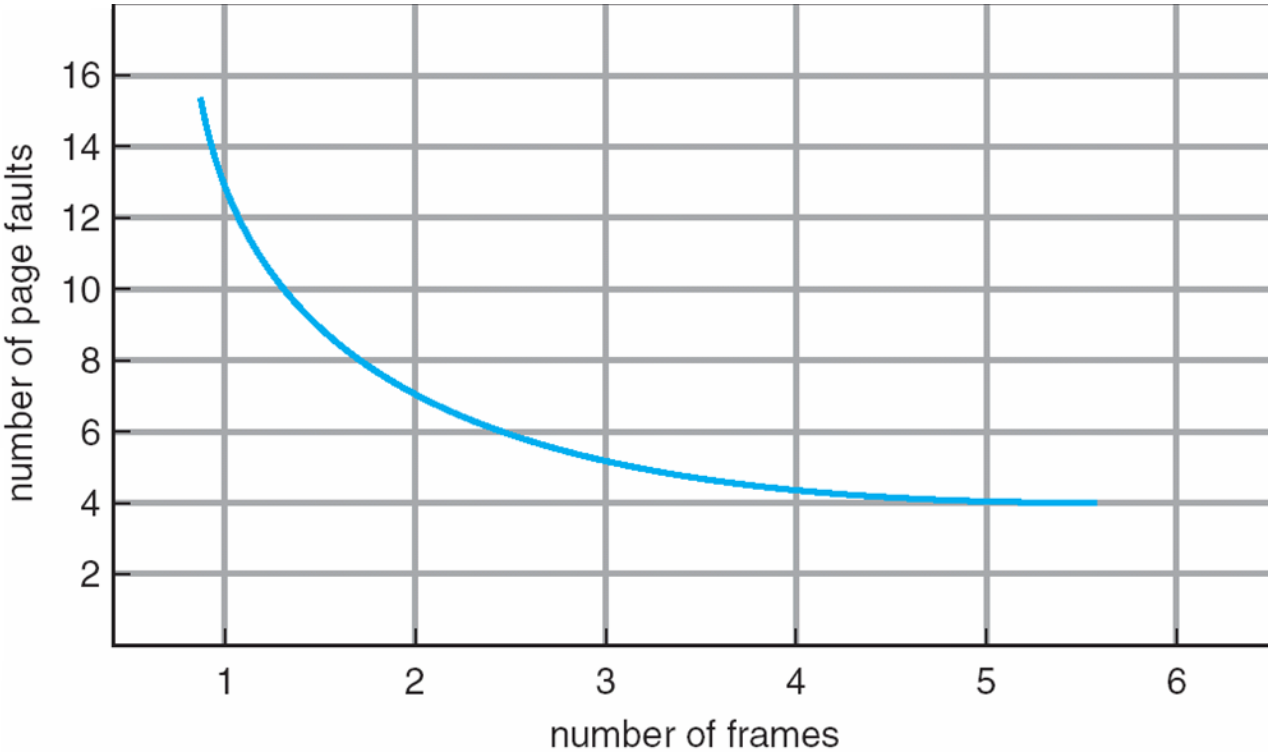
Page Replacement



Page and Frame Replacement Algorithms

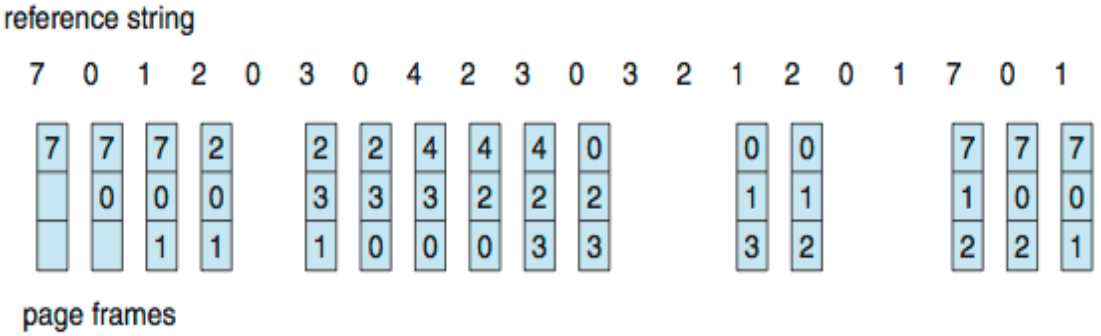
- **Frame-allocation algorithm/Çerçeve tahsis algoritması** belirler;
  - Her proses için kaç tane çerçeve verilecek
  - Değiştirilecek çerçeveler
- **Page-replacement algorithm/Sayfa değiştirme algoritması**
  - Hem ilk erişimde hem de yeniden erişimde en düşük sayfa hata oranı istenir
- Algoritmayı, belirli bir bellek başvurusu dizesinde (başvuru dizesi) çalıştırarak ve bu dizgede sayfa hatası sayısını hesaplayarak değerlendirin
  - Dize, tam adres değil, yalnızca sayfa numaralarıdır
  - Aynı sayfaya tekrar tekrar erişim, sayfa hatasına neden olmaz
  - Sonuçlar mevcut çerçevelerin sayısına bağlıdır
- Tüm örneklerimizde, referans verilen sayfa numaralarının referans dizesi/ **reference string**
- **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

Graph of Page Faults Versus The Number of Frames



First-In-First-Out (FIFO) Algorithm

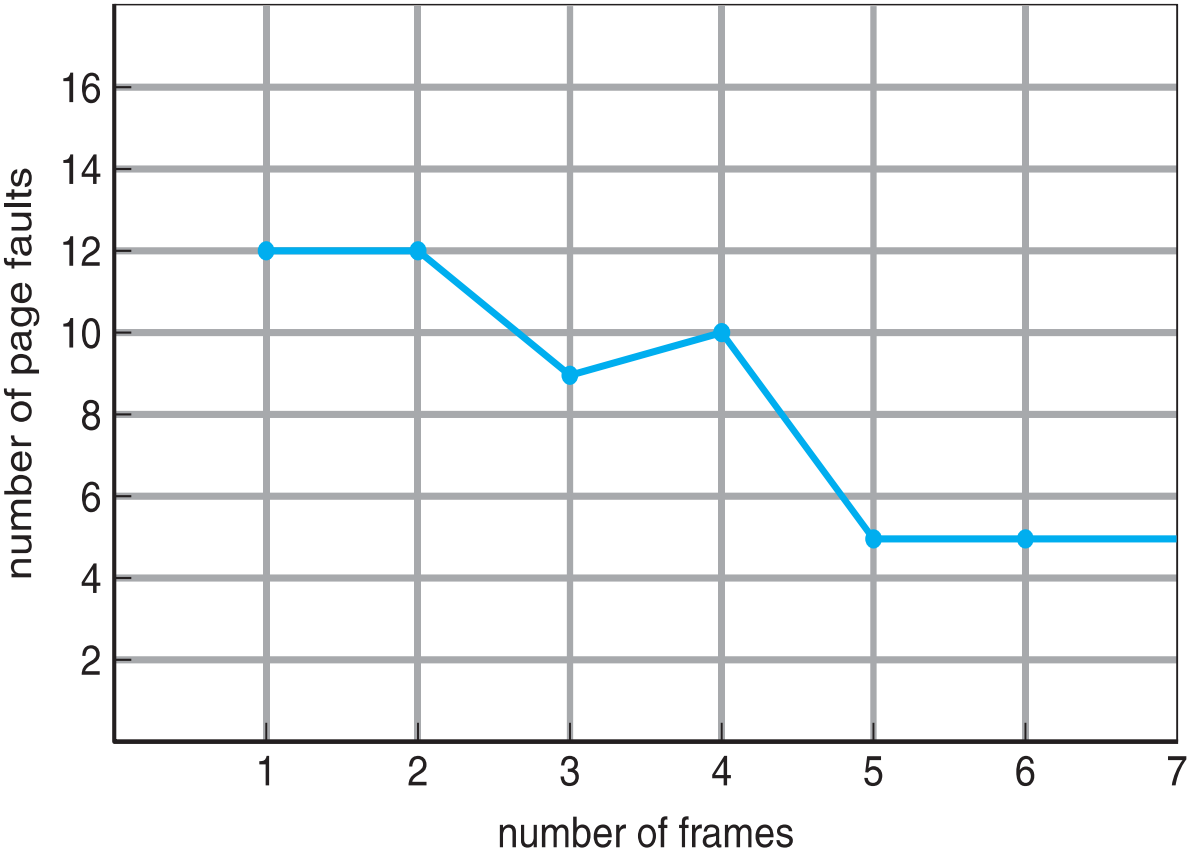
- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 çerçeve (proses başına bir anda 3 sayfa bellekte olabilir)



15 sayfa hatası\

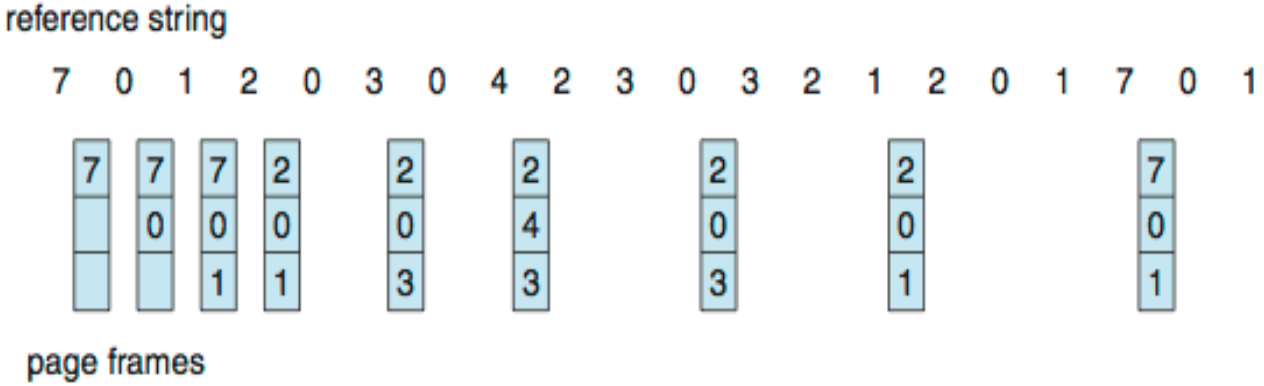
- Referans dizesine göre değişebilir: 1,2,3,4,1,2,5,1,2,3,4,5
  - Daha fazla çerçeve eklemek, daha fazla sayfa hatasına neden olabilir!
    - **Belady's Anomaly**

FIFO Illustrating Belady's Anomaly



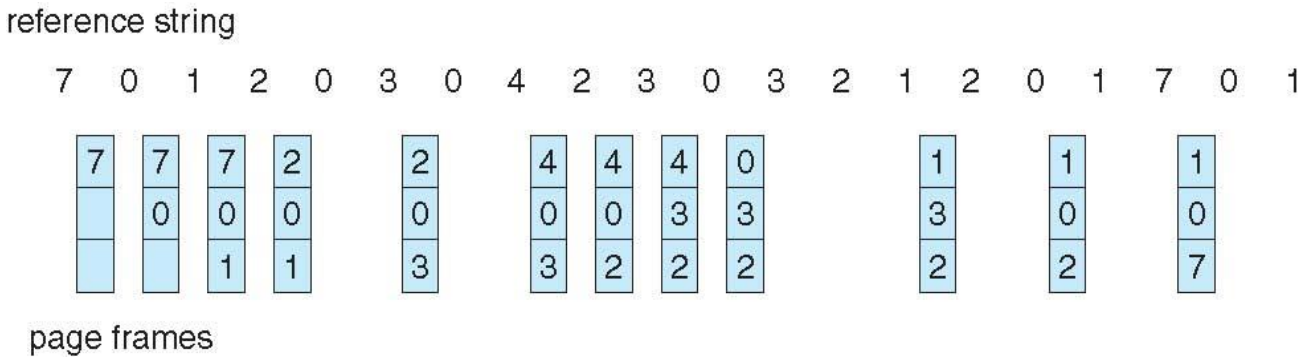
Optimal Algorithm

- En uzun süre kullanılmayacak sayfayı değiştirin
  - 9 örnek için en uygunudur
- Nasıl Bilinebilir?
  - Geleceği okuyamayız
- Algoritmanızın ne kadar iyi performans gösterdiğini ölçmek için kullanılır

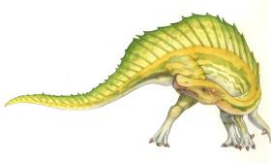


Least Recently Used (En eski kullanılan) Algorithm

- Gelecekten ziyade geçmiş bilgiyi kullanın
- En çok kullanılmayan sayfayı değiştirin
- Her sayfayla son kullanım süresini ilişkilendirme

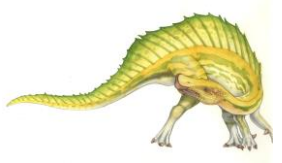


- 12 hata - FIFO'dan daha iyi, ancak OPT'den daha kötü
- Genellikle iyi bir algoritma ve sık kullanılır
- Ama nasıl uygulanır?

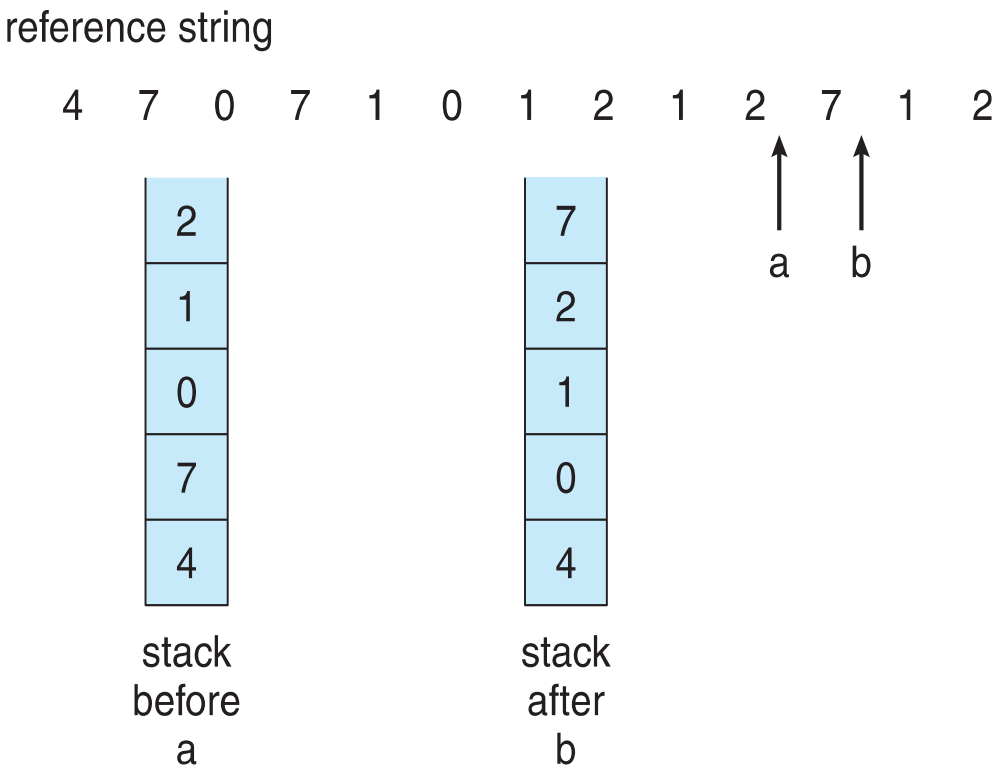


LRU Algorithm (Cont.)

- Sayaç uygulaması
  - Her sayfa girişinin bir sayacı vardır; Her sayfaya bu girişte referans verildiğinde, saati sayaca kopyalayın.
  - Bir sayfanın değiştirilmesi gerektiğinde, en küçük değeri bulmak için sayaçlara bakın
    - Gerekli tabloyu ara
- Yığın uygulaması (Bir sonraki slaytta görseli var)
  - Sayfa numaraları yığınını çift bağlantı biçiminde tutun:
  - Başvuru yapılan sayfa:
    - En üste getir
    - 6 işaretçinin değiştirilmesini gerektirir
  - Ancak her güncelleme daha masraflı
  - Yer Değiştirime için arama yok
- LRU ve OPT, Belady'nin Anomalisine sahip olmayan yığın algoritmaları/ **stack algorithms** durumlarıdır

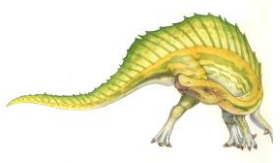


Use Of A Stack to Record Most Recent Page References

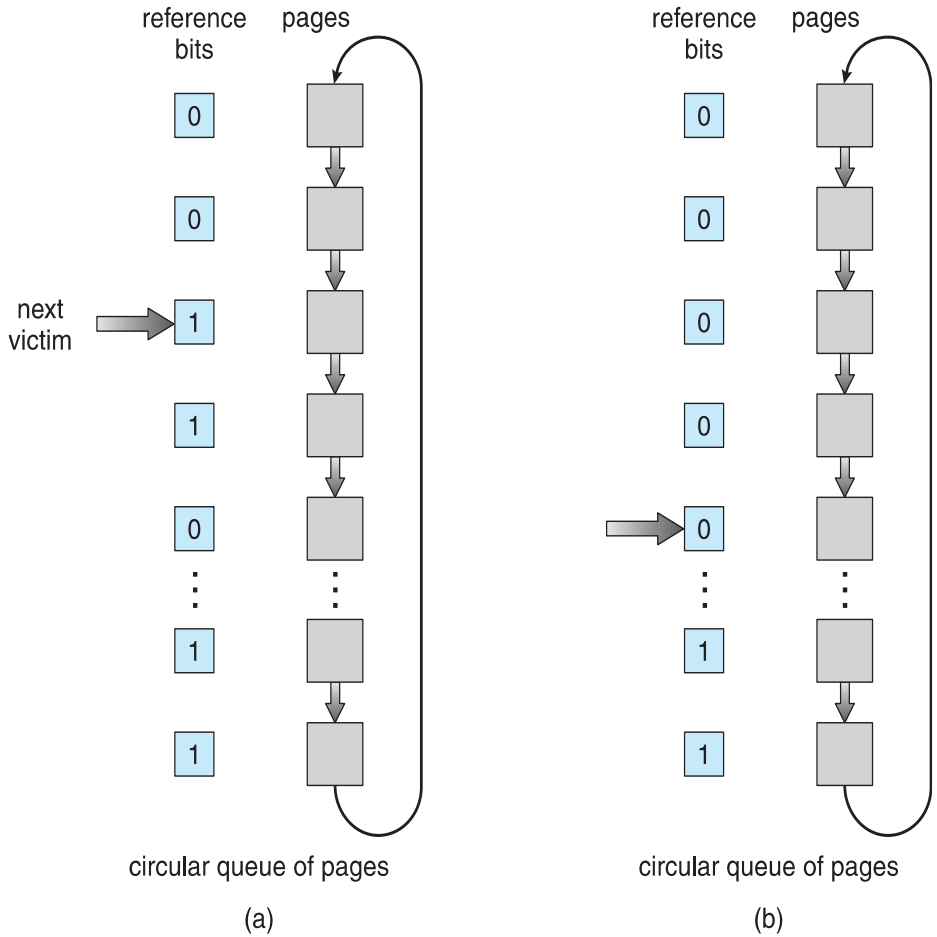


LRU Approximation Algorithms

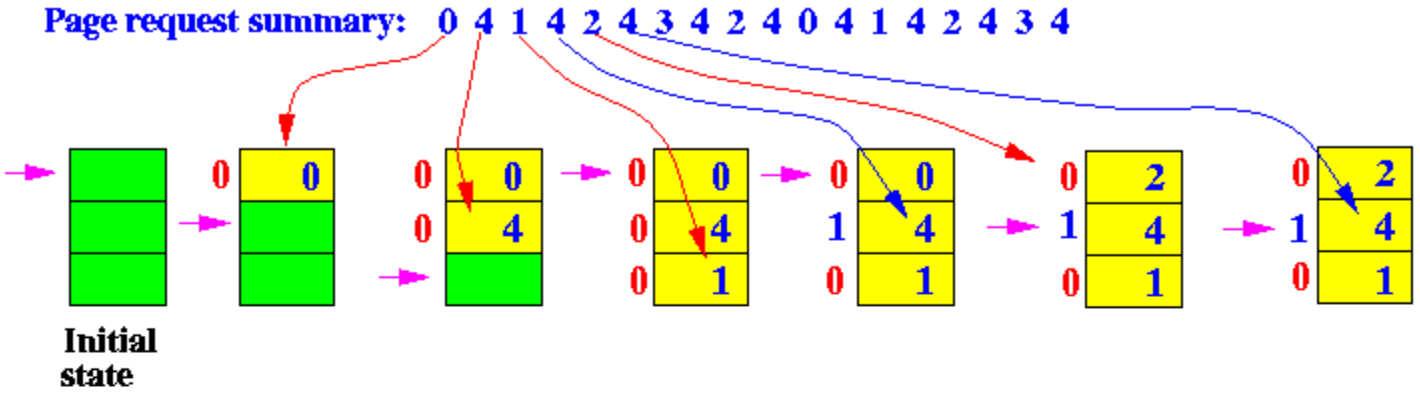
- LRU özel bir donanıma ihtiyaç duyuyor ve hala yavaş
- **Reference bit**
  - Her sayfa bir bit ile ilişkilendirilir, başlangıçta = 0
  - Sayfa referans edildiğinde bit 1 olarak ayarlanır
  - reference bit = 0 olan birisi ile değiştirin (varsa)
    - Fakat sırayı bilmiyoruz
- **Second-chance algorithm**
  - Genellikle FIFO, ayrıca donanım tarafından sağlanan referans biti
  - **Clock/saat** değiştirme
  - Değiştirilecek sayfa varsa
    - Reference bit = 0 -> değiştir
    - reference bit = 1 ise sonra:
      - referans biti 0'a ayarlayın, sayfayı belleğe bırakın
      - sonraki sayfayı değiştir, aynı kurallara tabi



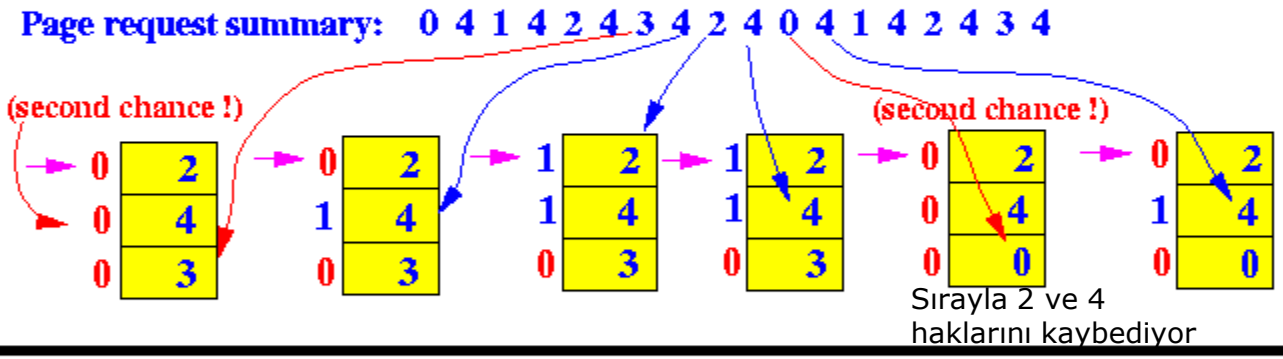
Second-Chance (clock) Page-Replacement Algorithm



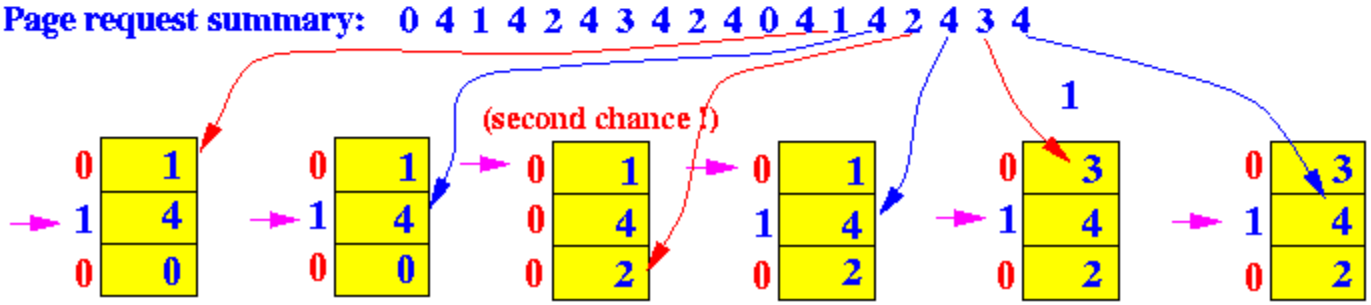
Second-Chance (clock) Page-Replacement Algorithm



FIFO

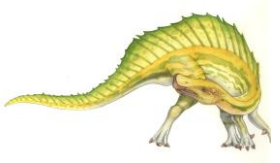


Sırayla 2 ve 4 haklarını kaybediyor



Enhanced Second-Chance Algorithm

- Referans bitini kullanarak algoritmayı geliştirin ve bit'i (varsa) değiştirin
- Sipariş edilen çifti al (referans, değiştir):
  - (0, 0) son zamanlarda hiçbiri değiştirilmedi - değiştirilecek en iyi sayfa
  - (0, 1) son zamanlarda kullanılmamış ancak değiştirilmemiş - çok iyi değil, değiştirmeden önce yazmalı
  - (1, 0) son zamanlarda kullanılmış, ancak temiz - muhtemelen kısa süre sonra tekrar kullanılacak
  - (1, 1) son zamanlarda kullanılmış ve değiştirilmiş - muhtemelen kısa süre sonra tekrar kullanılacak ve değiştirilmeden önce yazmanız gerekecek
- Sayfa değiştirme çağrıldığında, saat(clock) şemasını kullanın, ancak boş olmayan en düşük sınıftaki dört sınıf değiştirme sayfasını kullanın
- Dairesel kuyruğu birkaç kez aramanız gerekebilir



Counting Algorithms

- Her sayfaya yapılan referans sayısının bir sayacını saklayın
  - Yaygın değil
- **Lease Frequently Used** (**LFU**) **Algorithm**: Sayfayı en küçük sayıyla değiştirir
- **Most Frequently Used** (**MFU**) **Algorithm**: en küçük sayıma sahip sayfanın muhtemelen henüz kullanılmaya başlandığı ve henüz kullanılmamış olduğu argümanına dayanarak

Page-Buffering Algorithms

- Her zaman boş çerçeveleri bir havuzda tutun
  - Ardından gerektiğinde çerçeve kullanılabilir, hata anında bulunmaz
  - boş çerçeveden sayfayı okuyun ve kurban seç ve boş havuza ekle
  - Uygun olduğunda, kurbanı serbest bırak
- Bir ihtimalle, değiştirilen sayfaların listesini tut
  - Depolama birimi boşsa, sayfayı oraya yaz ve non-dirty/kullanılmamış olarak ayarla
  - Bir ihtimalle, boş çerçeve içeriklerini sağlam tutun ve içlerinde ne olduğunu not edin.
  - Tekrar kullanmadan önce tekrar başvuruda bulunulursa, içeriği tekrar diskten yüklemeye gerek yoktur
  - Yanlış kurban çerçevesi seçildiğinde cezayı azaltmak için genellikle yararlı

Applications and Page Replacement

- Bu algoritmaların tümü gelecekteki sayfa erişimi hakkında işletim sistemi tahminine sahiptir.
- Bazı uygulamalar daha iyi bilgiye sahiptir - yani veritabanları
- Hafıza yoğun uygulamalar çift tamponlamaya neden olabilir
  - İşletim sistemi sayfanın kopyasını G/Ç tamponu olarak bellekte tutar
  - Uygulama, kendi çalışması için sayfayı bellekte tutar
- İşletim sistemi, uygulamaların dışına çıkarak diske doğrudan erişim sağlayabilir
  - **Raw disk** modu
- Bypasses buffering, locking, etc

Allocation of Frames

- Her proses **minimum** çerçeve sayısına ihtiyaç duyar
- Örnek: IBM 370 - SS MOVE komutunu işlemek için 6 sayfa:
  - Komut6 bayttır, 2 sayfaya kadar yayılabilir
  - 2 pages to handle *from*
  - 2 pages to handle *to*
- Tabii maksimum sistemdeki toplam çerçevelerdir
- İki ana tahsis şeması
  - sabit tahsis
  - öncelikli tahsisi

Fixed Allocation/Sabit Tahsis

- Eşit tahsisi - Örneğin, 100 çerçeve (işletim sistemi için çerçeveler ayırıldıktan sonra) ve 5 proses varsa, her işlem için 20 çerçeve verin
  - Bazılarını boş çerçeve tampon havuzu olarak sakla
- Oransal tahsis - Proses büyüklüğüne göre tahsis
  - Çoklu programlama derecesine göre dinamik, proses boyutları değişgen
    - s_i = size of process p_i
    - $S = \sum s_i$
    - m = total number of frames
    - a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 4$

$a_2 = \frac{127}{137} \times 62 \approx 57$

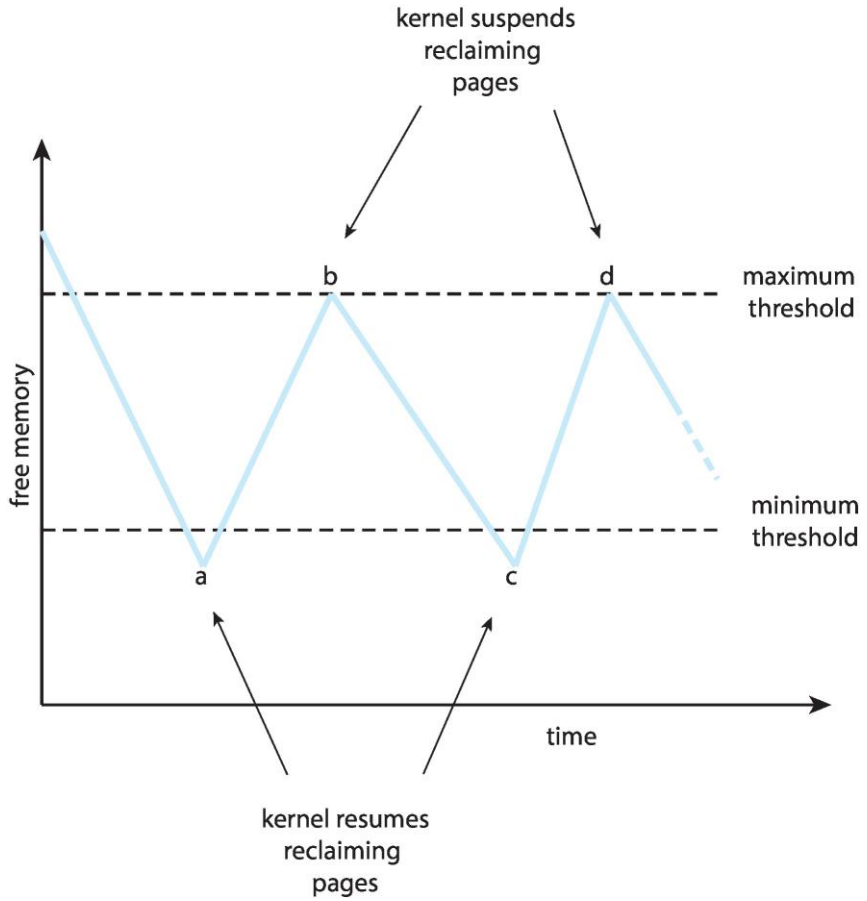
Global vs. Local Allocation

- **Global replacement** – Global değiştirme - proses, tüm çerçeve kümesinden bir değiştirme çerçevesi seçer; bir proses diğerinden bir çerçeve alabilir
  - Ancak proses yürütme süresi büyük ölçüde değişebilir
  - Fakat daha büyük iş hacmi çok daha yaygın
- **Local replacement** – Yerel değiştirme - her işlem yalnızca kendine ayrılmış çerçeve setinden seçebilir
  - Proses başına daha tutarlı performans
  - Ancak muhtemelen yetersiz bellek

Reclaiming Pages/Sayfaları geri almak

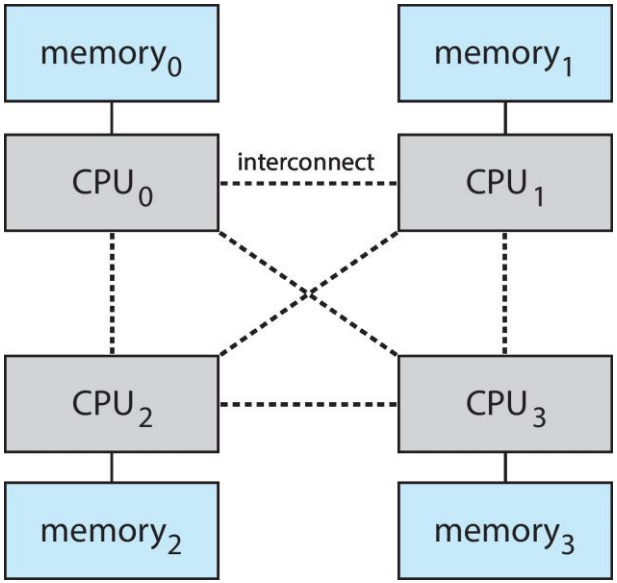
- Global sayfa değiştirme politikasını uygulama stratejisi
- Tüm bellek istekleri, değiştirilecek sayfaları seçmeye başlamadan önce listenin sıfıra düşmesini beklemek yerine serbest çerçeve listesinden yerine getirilir.
- Sayfa değiştirme, liste belirli bir eşiğin altına düştüğünde tetiklenir.
- Bu strateji, yeni istekleri karşılamak için her zaman yeterli boş hafıza olduğundan emin olmaya çalışır.

Reclaiming Pages Example



Non-Uniform Memory Access

- Şimdiye kadar tüm hafızalara erişim eşitti
- Birçok sistem **NUMA** 'dır - belleğe erişim hızı değişir
  - Sistem veriyolu üzerinden birbirine bağlı CPU ve bellek içeren sistem kartlarını göz önünde bulundurun
- NUMA çokişlemli/ multiprocessing mimarisi

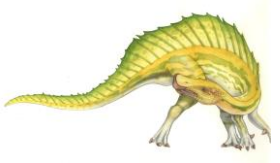


Non-Uniform Memory Access (Cont.)

- En iyi performans, iş parçacığının programlandığı CPU'ya "yakın" bir bellek tahsisi ile olur
  - Ve iş parçacığı, mümkün olduğunda aynı sistem kartındaki iş parçacığını çizelgeleyicisi ile çizelgelenmeli
  - Solaris tarafından **lgroups** oluşturarak çözüldü.
    - Structure to track CPU / Memory low latency groups
    - Used my schedule and pager
    - When possible schedule all threads of a process and allocate all memory for that process within the lgroup

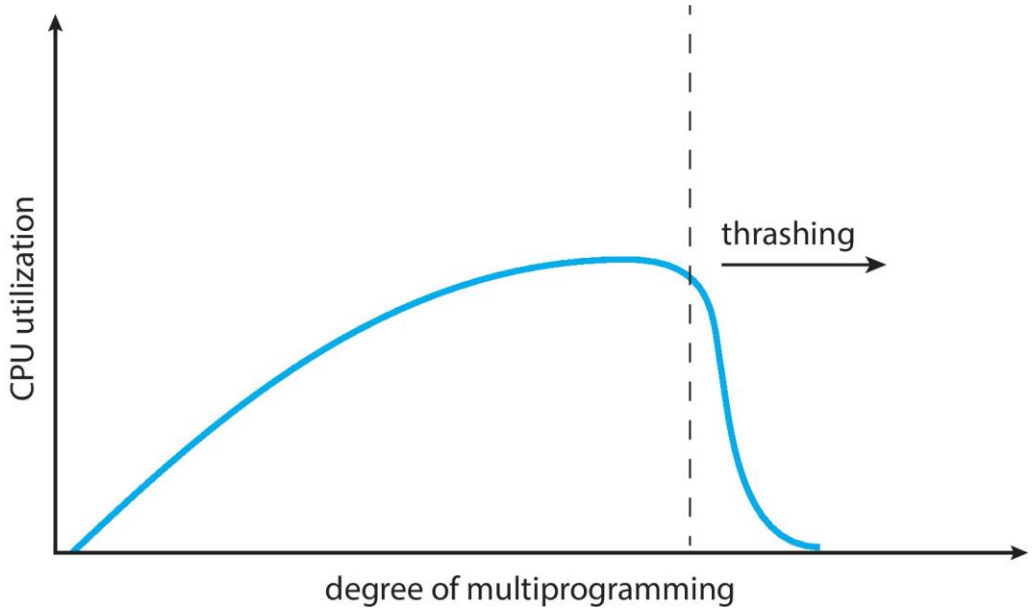
Thrashing/Boşuna çalışma

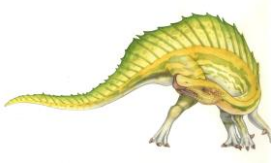
- Bir proses “yeterli” sayfalara sahip değilse, sayfa-hata oranı çok yüksektir
  - Sayfa almak için sayfa hatası
  - Mevcut çerçeveyi değiştir
  - Ancak çabucak değiştirilen çerçeveyi geri almak gerekir
  - This leads to:
    - Düşük CPU kullanımı
    - İşletim sistemi, çoklu programlama derecesini arttırması gerektiğini düşünür
    - Sisteme bir başka proses eklenir



Thrashing (Cont.)

- **Thrashing**. Bir proses, sayfaları içeri ve dışarı takasla meşgul





Demand Paging and Thrashing

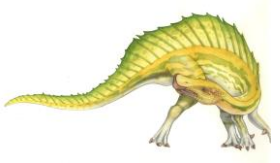
- Neden talep sayfalama çalışır?

  Locality model

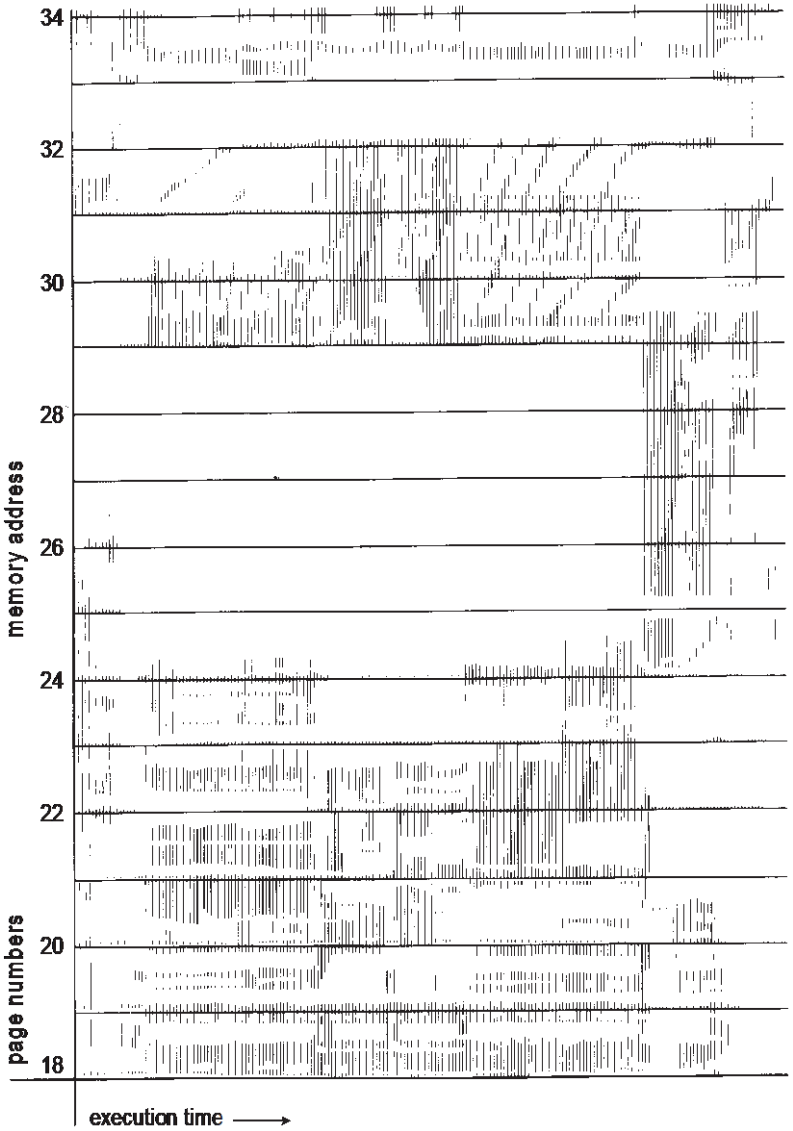
  - Proses bir konumdan diğerine geçiş yapıyor
  - Yerler üst üste gelebilir
- Thrashing neden oluyor?

  Σ size of locality > total memory size

- Yerel veya öncelikli sayfa değiştirmeyi kullanarak etkisini sınırlandırın

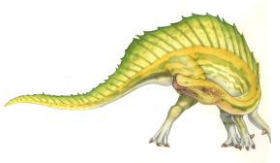


Locality In A Memory-Reference Pattern



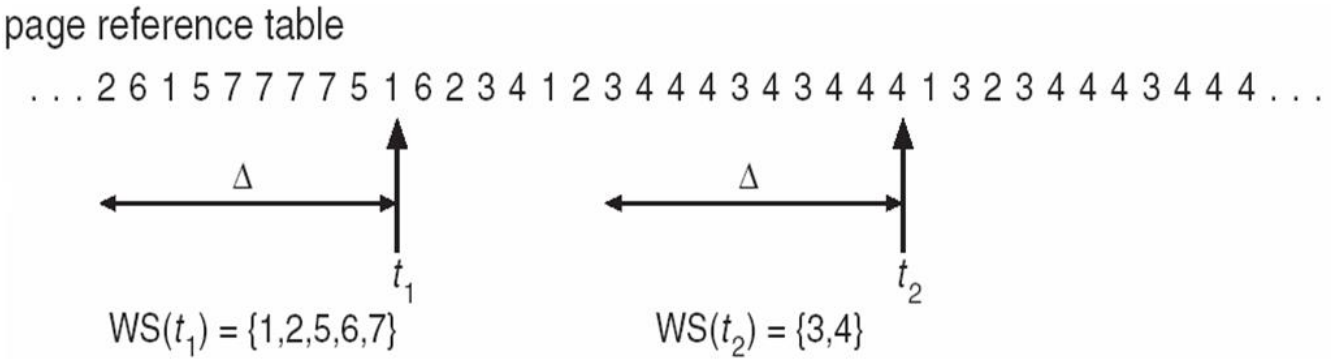
Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small, tüm bölgeyi kapsamayacak
  - if Δ too large, birkaç yerleşim birimini kapsayacak
  - if $\Delta = \infty \Rightarrow$ tüm programı kapsayacak
- $D = \Sigma\ WSS_i \equiv$ total demand frames
  - Approximation of locality



Working-Set Model (Cont.)

- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, ardından işlemlerden birini askıya alın veya değiştirin

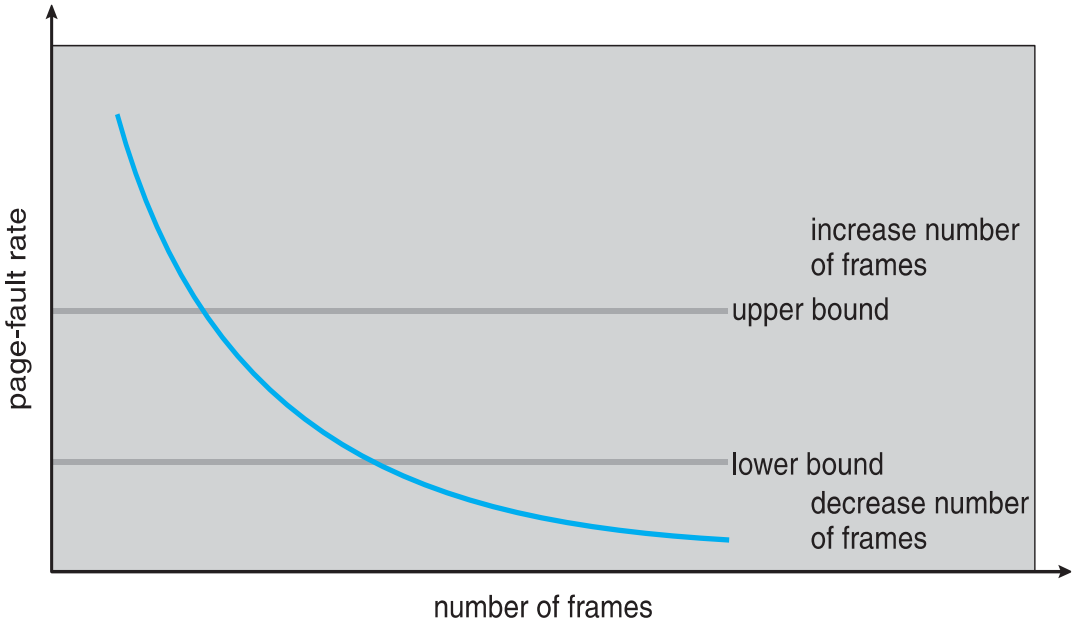


Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

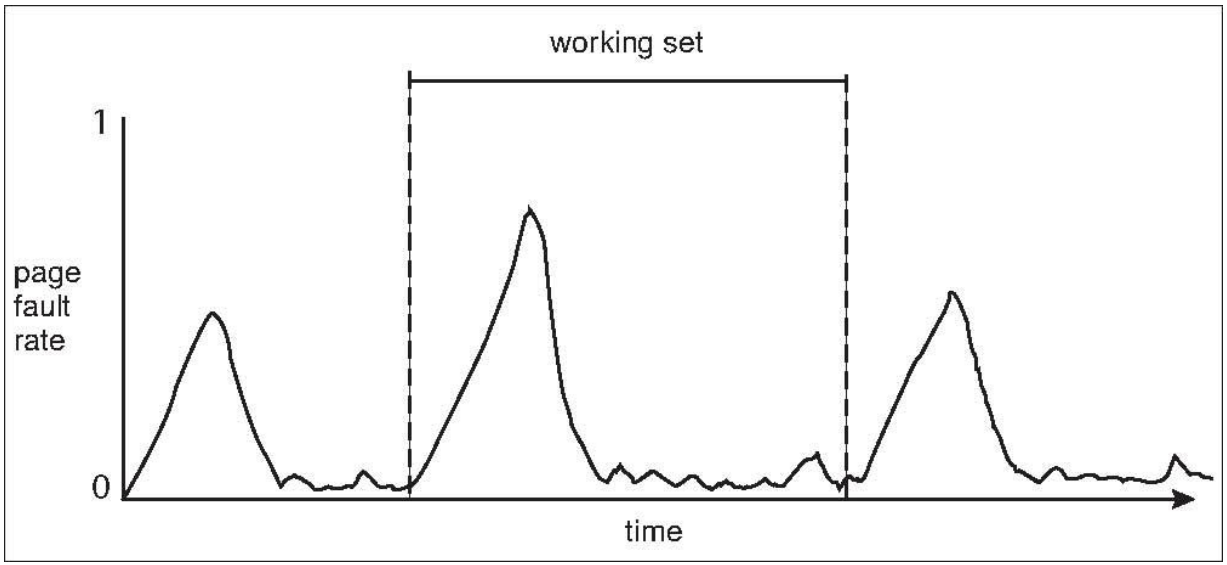
Page-Fault Frequency

- More direct approach than WSS
- Establish "acceptable" **page-fault frequency** (**PFF**) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



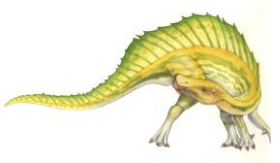
Working Sets and Page Fault Rates

- Bir işlemin çalışma grubu ile sayfa hata oranı arasındaki doğrudan ilişkisi
- Çalışma grubu zamanla değişir
- Zamanla yükselir ve alçalır



Allocating Kernel Memory

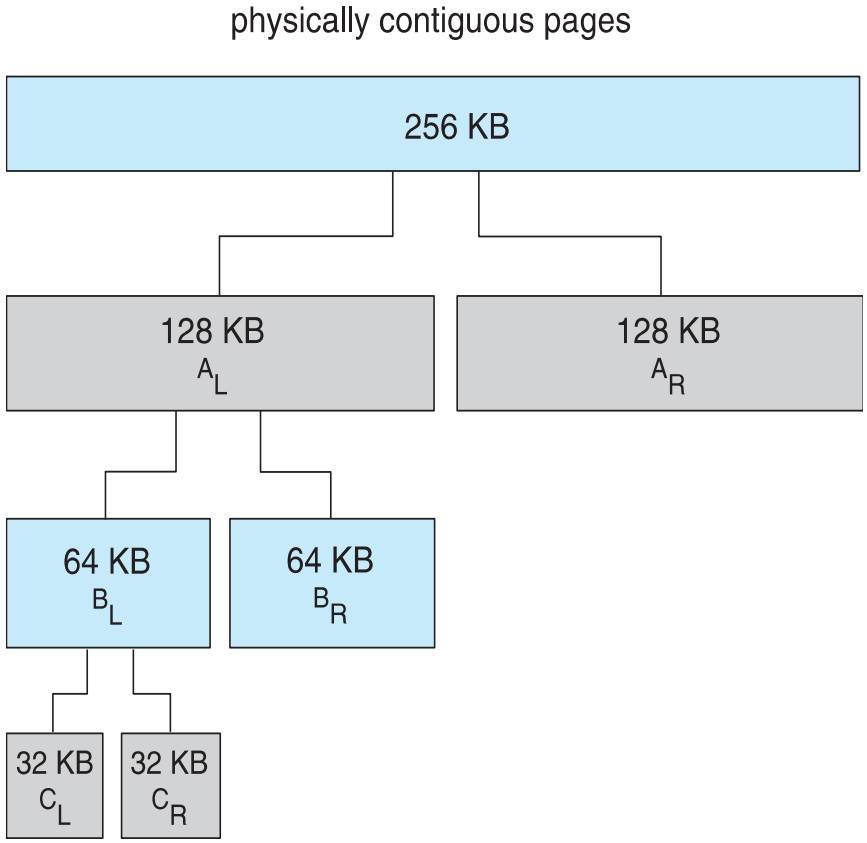
- Treated differently from user memory
- Often allocated from a free-memory pool
  - Kernel requests memory for structures of varying sizes
  - Some kernel memory needs to be contiguous
    - I.e. for device I/O



Buddy System

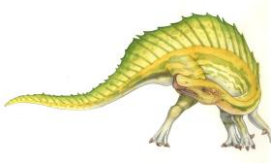
- Allocates memory from fixed-size segment consisting of physically-contiguous pages
- Memory allocated using **power-of-2 allocator**
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available
- For example, assume 256KB chunk available, kernel requests 21KB
  - Split into A_L and A_R of 128KB each
    - One further divided into B_L and B_R of 64KB
      - One further into C_L and C_R of 32KB each – one used to satisfy request
- Advantage – quickly **coalesce** unused chunks into larger chunk
- Disadvantage - fragmentation

Buddy System Allocator

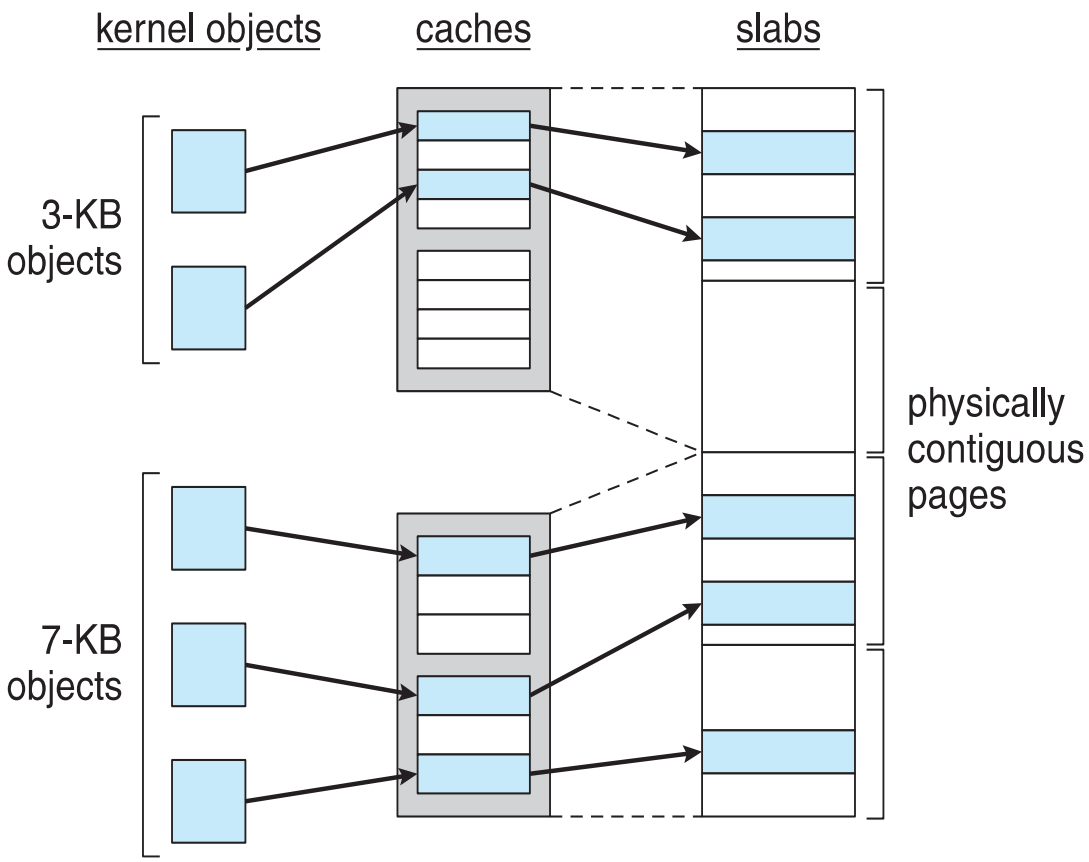


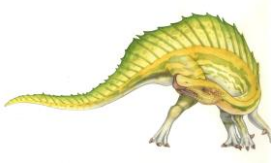
Slab Allocator

- Alternate strategy
- **Slab** is one or more physically contiguous pages
- **Cache** consists of one or more slabs
- Single cache for each unique kernel data structure
  - Each cache filled with **objects** – instantiations of the data structure
- When cache created, filled with objects marked as **`free`**
- When structures stored, objects marked as **`used`**
- If slab is full of used objects, next object allocated from empty slab
  - If no empty slabs, new slab allocated
- Benefits include no fragmentation, fast memory request satisfaction



Slab Allocation



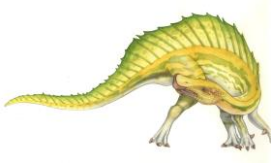


Slab Allocator in Linux

- For example process descriptor is of type `struct task_struct`
- Approx 1.7KB of memory
- New task -> allocate new struct from cache
  - Will use existing free `struct task_struct`
- Slab can be in three possible states
  1. Full – all used
  2. Empty – all free
  3. Partial – mix of free and used
- Upon request, slab allocator
  1. Uses free struct in partial slab
  2. If none, takes one from empty slab
  3. If no empty slab, create new empty

Slab Allocator in Linux (Cont.)

- Slab started in Solaris, now wide-spread for both kernel mode and user memory in various OSes
- Linux 2.2 had SLAB, now has both SLOB and SLUB allocators
  - SLOB for systems with limited memory
    - Simple List of Blocks – maintains 3 list objects for small, medium, large objects
  - SLUB is performance-optimized SLAB removes per-CPU queues, metadata stored in page structure

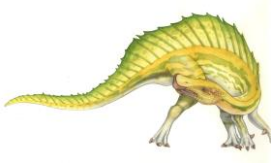


Other Considerations

- Prepaging
- Page size
- TLB reach
- Inverted page table
- Program structure
- I/O interlock and page locking

Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory was wasted
- Assume *s* pages are prepaged and *α* of the pages is used
  - Is cost of ***s * α*** save pages faults > or < than the cost of prepaging
    s * (1- α) unnecessary pages?
  - ***α*** near zero ⇒ prepaging loses



Page Size

- Sometimes OS designers have a choice
  - Especially if running on custom-built CPU
- Page size selection must take into consideration:
  - Fragmentation
  - Page table size
  - **Resolution**
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 2^{12} (4,096 bytes) to 2^{22} (4,194,304 bytes)
- On average, growing over time

TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

Program Structure

- Program structure
  - `int[128,128] data;`
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

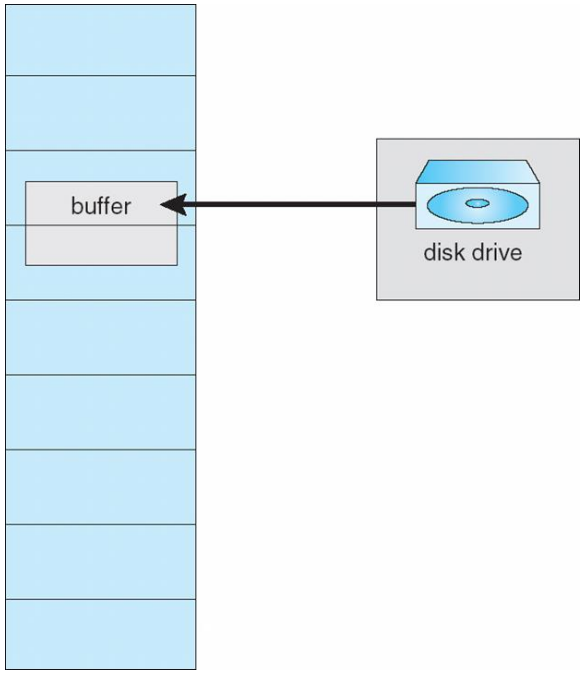
  - Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults

I/O interlock

- **I/O Interlock** – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- **Pinning** of pages to lock into memory



Operating System Examples

- Windows
- Solaris

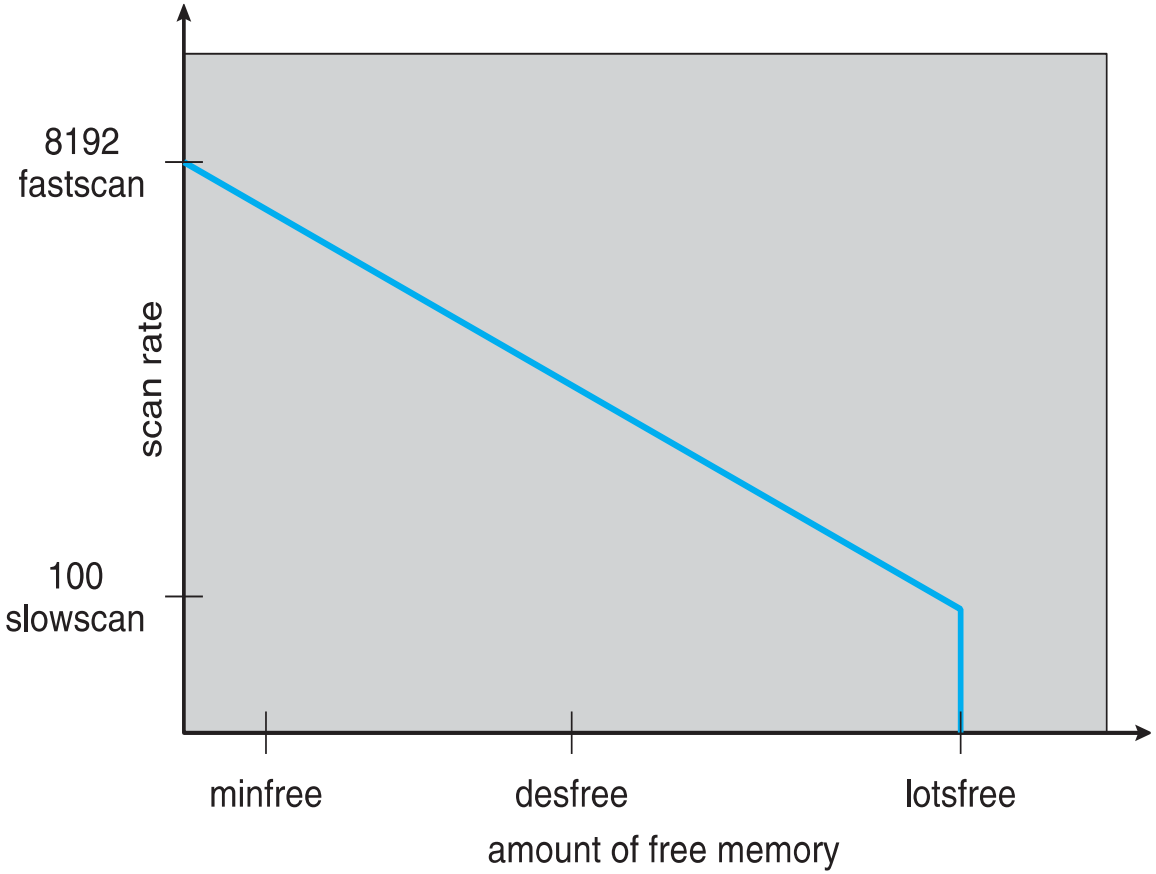
Windows

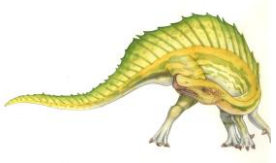
- Uses demand paging with **clustering**. Clustering brings in pages surrounding the faulting page
- Processes are assigned **working set minimum** and **working set maximum**
- Working set minimum is the minimum number of pages the process is guaranteed to have in memory
- A process may be assigned as many pages up to its working set maximum
- When the amount of free memory in the system falls below a threshold, **automatic working set trimming** is performed to restore the amount of free memory
- Working set trimming removes pages from processes that have pages in excess of their working set minimum

Solaris

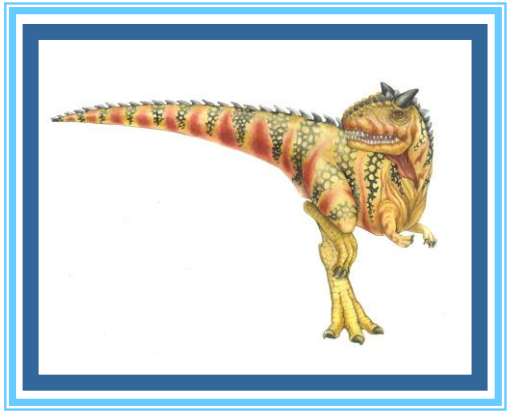
- Maintains a list of free pages to assign faulting processes
- **`Lotsfree`** – threshold parameter (amount of free memory) to begin paging
- **`Desfree`** – threshold parameter to increasing paging
- **`Minfree`** – threshold parameter to being swapping
- Paging is performed by **`pageout`** process
- **`Pageout`** scans pages using modified clock algorithm
- **`Scanrate`** is the rate at which pages are scanned. This ranges from **`slowscan`** to **`fastscan`**
- **`Pageout`** is called more frequently depending upon the amount of free memory available
- **Priority paging** gives priority to process code pages

Solaris 2 Page Scanner





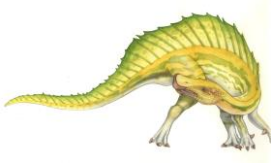
End of Chapter 10



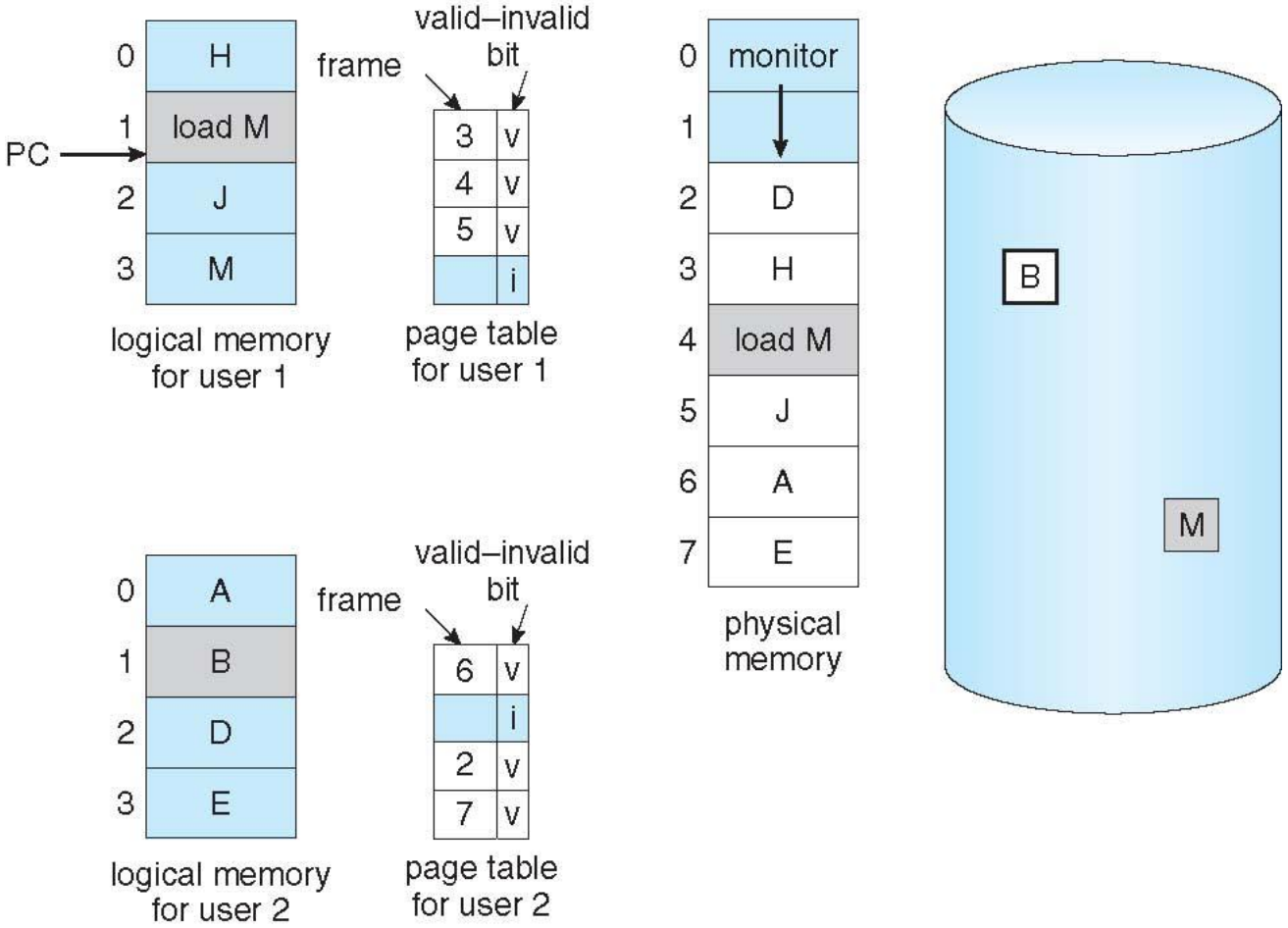
Performance of Demand Paging

- Stages in Demand Paging (worse case)

1. Trap to the operating system
2. Save the user registers and process state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    1. Wait in a queue for this device until the read request is serviced
    2. Wait for the device seek and/or latency time
    3. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user
7. Receive an interrupt from the disk I/O subsystem (I/O completed)
8. Save the registers and process state for the other user
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show page is now in memory
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction

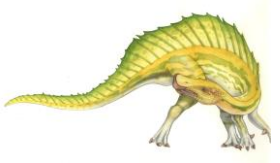


Need For Page Replacement



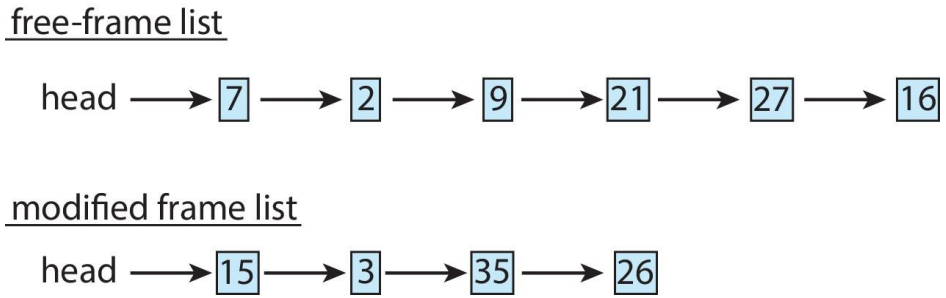
Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

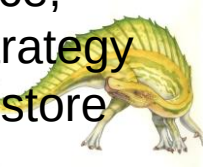


Memory Compression

- **Memory compression** -- rather than paging out modified frames to swap space, we compress several frames into a single frame, enabling the system to reduce memory usage without resorting to swapping pages.
- Consider the following free-frame-list consisting of 6 frames



- Assume that this number of free frames falls below a certain threshold that triggers page replacement. The replacement algorithm (say, an LRU approximation algorithm) selects four frames -- 15, 3, 35, and 26 to place on the free-frame list. It first places these frames on a modified-frame list. Typically, the modified-frame list would next be written to swap space, making the frames available to the free-frame list. An alternative strategy is to compress a number of frames{\mdash}say, three{\mdash}and store their compressed versions n a single page frame.



Memory Compression (Cont.)

- An alternative to paging is **memory compression**.
- Rather than paging out modified frames to swap space, we compress several frames into a single frame, enabling the system to reduce memory usage without resorting to swapping pages.