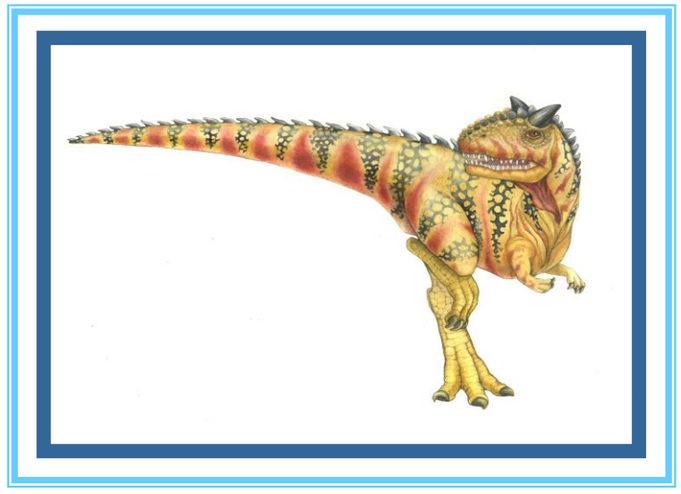
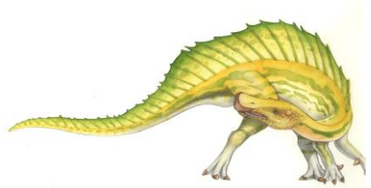
5. Bölüm: CPU Çizelgeleme (CPU Scheduling)



5. Bölüm: İş Sıralama (CPU Çizelgeleme)

- Temel Kavramlar
- Çizelgeleme Kriterleri
- Çizelgeleme Algoritmaları
- İş Parçacığı Çizelgeleme
- Çoklu-işlemci Çizelgeleme
- Gerçek-zamanlı CPU Çizelgeleme
- İşletim Sistemleri Örnekleri
- Algoritma Değerlendirme



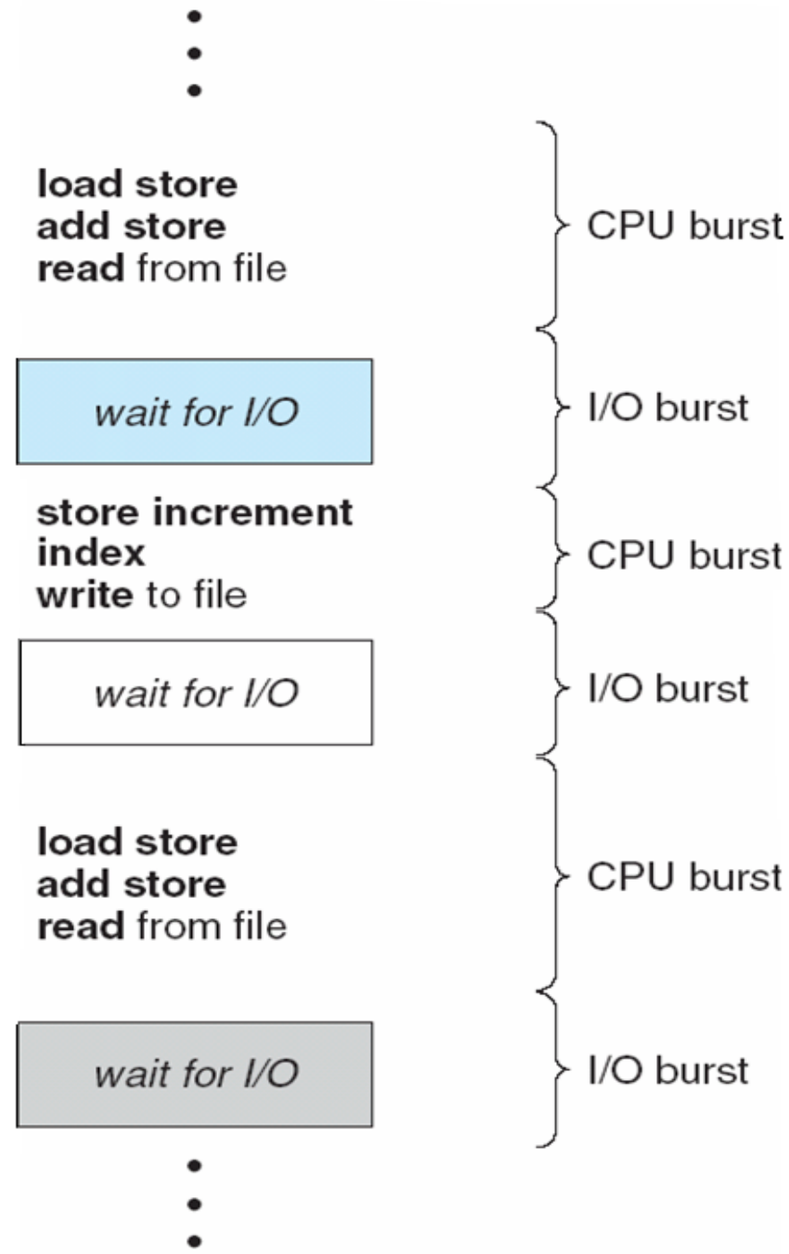
Hedefler

- Çoklu-programlı(multi-programmed) işletim sistemlerine temel teşkil eden CPU Çizelgeleme konusunu anlamak
- Çeşitli CPU Çizelgeleme algoritmalarını tanıtmak
- Belirli bir sistem için CPU Çizelgeleme algoritma seçim kriterlerini değerlendirmek

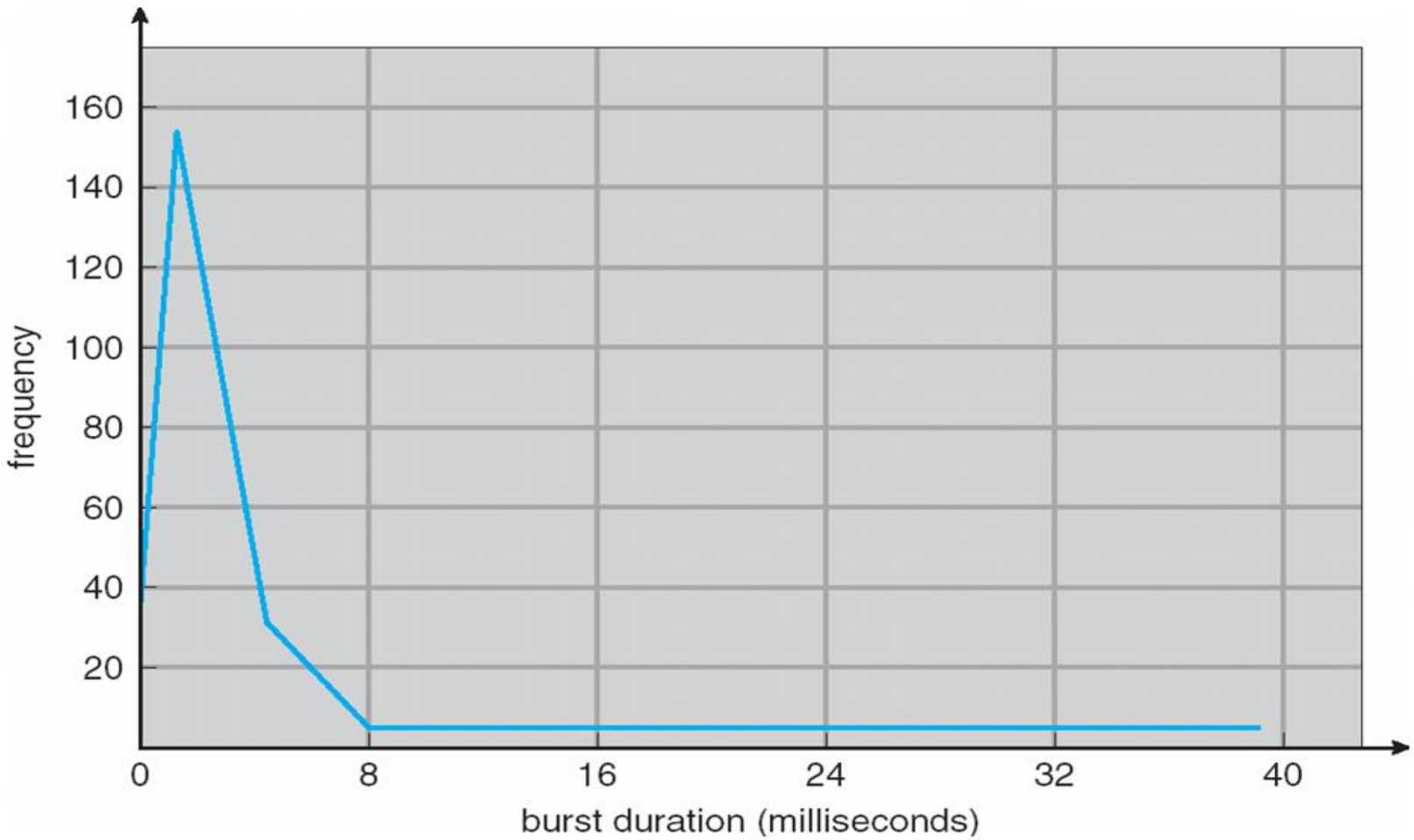
➢ Çoklu-programlamanın amacı, CPU kullanımını en üst düzeye çıkarmak için bazı işlemlerin her zaman çalışmasını sağlamaktır.

Temel Kavramlar

- Çoklu-programlama sonucu **maksimum CPU kullanımı** elde edilir
- CPU–I/O Patlama (Burst) Çevrimi – Proses yürütülürken CPU'nun yürütülmesi ve I/O beklemesinden oluşan bir çevrimdir
- **CPU patlamasının ardından I/O patlaması gelir**
- **CPU patlama** dağılımı ana ilgi noktasıdır



CPU-patlama Zamanları



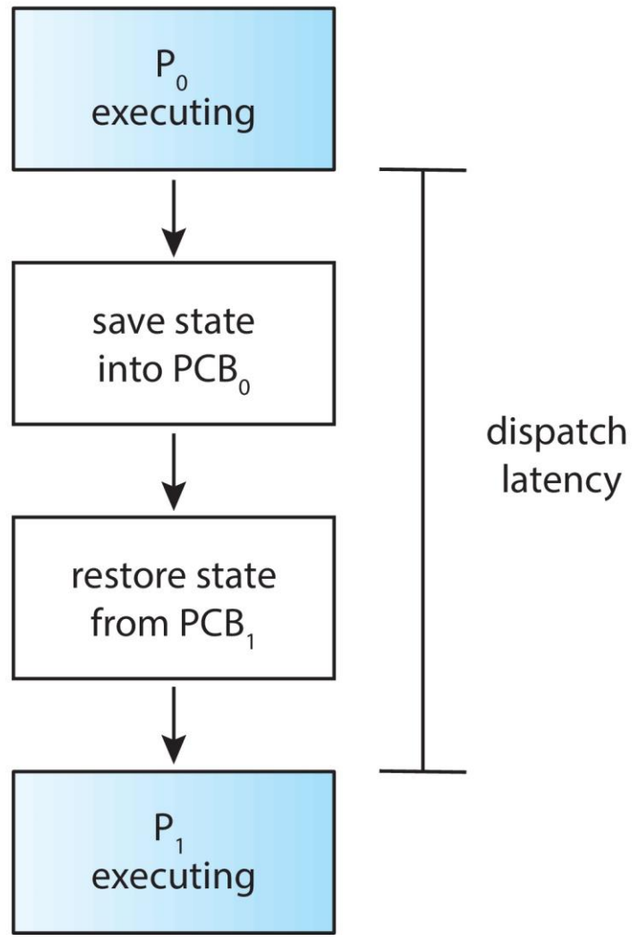
- Eğri genel olarak üstel veya hiper-üstel olarak karakterize edilir;
- Çok sayıda kısa CPU patlamaları ve az sayıda uzun CPU patlamaları.

CPU Çizelgeleyici

- Hazır kuyruğunda bekleyen prosesler arasından seçim yapar ve CPU'yu birine tahsis eder.
  - Kuyruk çeşitli şekillerde sıralanabilir
  - Bununla birlikte, kavramsal olarak hazır kuyruktaki tüm işlemler, CPU üzerinde yürütülme şansını bekleyerek sıralanır. Kuyruklardaki kayıtlar genellikle proses kontrol blokları (PCB'leri) 'dir.
- CPU Çizelgeleme kararları aşağıdaki durumlarda verilir:
  1. Çalışıyor durumundan bekleme durumuna geçerken (I/O için bekleme olabilir)
  2. Çalışıyor durumundan hazır durumuna geçerken (Kesme ile olabilir)
  3. Bekleme durumunda hazır durumuna geçerken (I/O işlemi bitmiş olabilir)
  4. Proses sonlanınca
- 1 ve 4 durumları **kesintisiz – nonpreemptive, başka seçenek yok**
- Diğer tüm çizelgeleme işlemleri **kesintili - preemptive**
  - Paylaşılan veriye erişim,
  - Çekirdek modundayken kullanıcı moduna geçiş veya
  - Önemli OS işlemleri esnasında olan kesmeler gibi

Görevlendirici - Dispatcher

- Görevlendirici modülü CPU'nun kontrolünü kısa vadeli çizelgeleyici tarafından seçilen prosese verir. Bu işlem aşağıdaki işlemleri içerir:
  - Bağlam (İçerik) değişimi
  - Kullanıcı moduna geçiş
  - Kullanıcı programını yeniden başlatmak için programdaki uygun bir konuma dallanma



- **Görevlendirici olabildiğince hızlı olmalı**
- **Görevlendirme Gecikmesi**– bir prosesi sonlandırmak ve bir başkasını çalıştırmak için geçen süre

Çizelgeleme Kriterleri

- **CPU kullanımı** – CPU'yu olabildiğince meşgul tut
- **İş çıkarma oranı (throughput)** – birim zamanda çalışmasını tamamlanan proses sayısı
- **İş bitirme zamanı** – belirli bir prosesin yürütülmesi için gerekli zaman
- **Bekleme zamanı** – hazır kuyruğundaki beklemekte olan prosesin geçirdiği süre
- **Cevap zamanı** – bir istek gönderildikten ilk cevap alınana (çıkış değil) kadarki geçen süre

Çizelgeleme Algoritması Optimizasyon Kriterleri

- Max CPU kullanımı
- Max iş çıkarma oranı
- Min iş bitirme zamanı
- Min bekleme zamanı
- Min cevap zamanı

İlk Gelen İlk Çalışır Algoritması (First-Come, First-Served - FCFS)

Proses	Patlama zamanı
P_1	24
P_2	3
P_3	3

- Proseslerin P_1 , P_2 , P_3 *sırasında geldiğini varsayalım* Gantt Diyagramı :

- *Bekleme zamanları* P_1 = 0; P_2 = 24; P_3 = 27
- Ortalama bekleme zamanı: (0 + 24 + 27)/3 = 17

FCFS Çizelgeleme (Devam)

Proseslerin aşağıdaki sırada geldiğini varsayalım:

$$P_2 , P_3 , P_1$$

- Gantt diyagramı :

- Bekleme zamanı $P_1 = 6; P_2 = 0, P_3 = 3$
- Ortalama bekleme zamanı: (6 + 0 + 3)/3 = 3
- Az önceki durumdan daha iyi. FCFS durumlardan çok etkilenir
- **Konvoy etkisi** – uzun proseslerin arkasında kısa proseslerin beklemesi
  - Bir adet CPU-bağımlı proses ve birden fazla I/O-bağımlı prosesler durumu gibi

En Kısa İş Önce (Shortest-Job-First - SJF) Algoritması

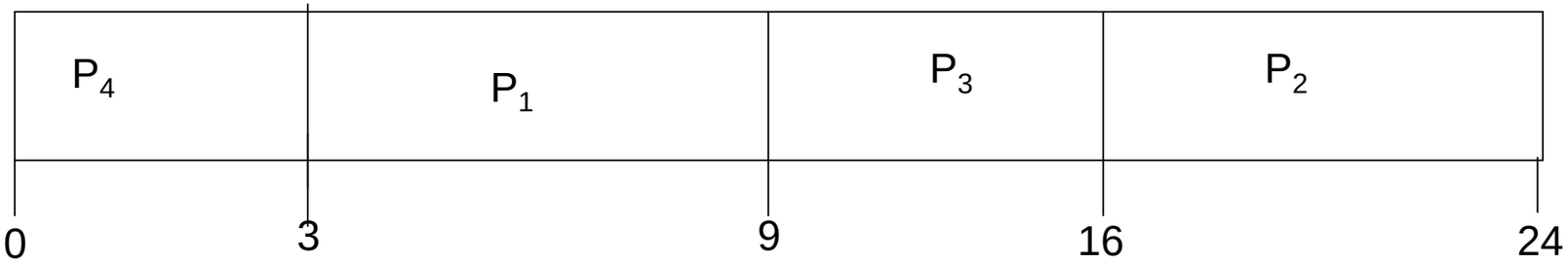
- Her bir prosesin bir sonraki CPU patlama süresiyle ilişkilendirilir
  - en kısa zamanlı prosesi tespit etmek için bu zaman değerlerini kullan

- SJF en uygundur – verilen bir grup proses için minimum ortalama bekleme zamanına neden olur
  - (Zorluk) bir sonraki CPU isteğinin patlama süresinin hesaplanmasındadır
  - Kullanıcıdan istenir

SJF örneği

Proses	Patlama Zamanı
P_1	6
P_2	8
P_3	7
P_4	3

- SJF iş sıralama diyagramı



- Ortalama bekleme zamanı= (3 + 16 + 9 + 0) / 4 = 7

Bir Sonraki CPU Patlama Boyutunun Belirlenmesi

- Süre sadece tahmin edilebilir – bir öncekine benzer olur
  - Bir sonraki CPU patlaması en kısa tahmin edilen prosesi al

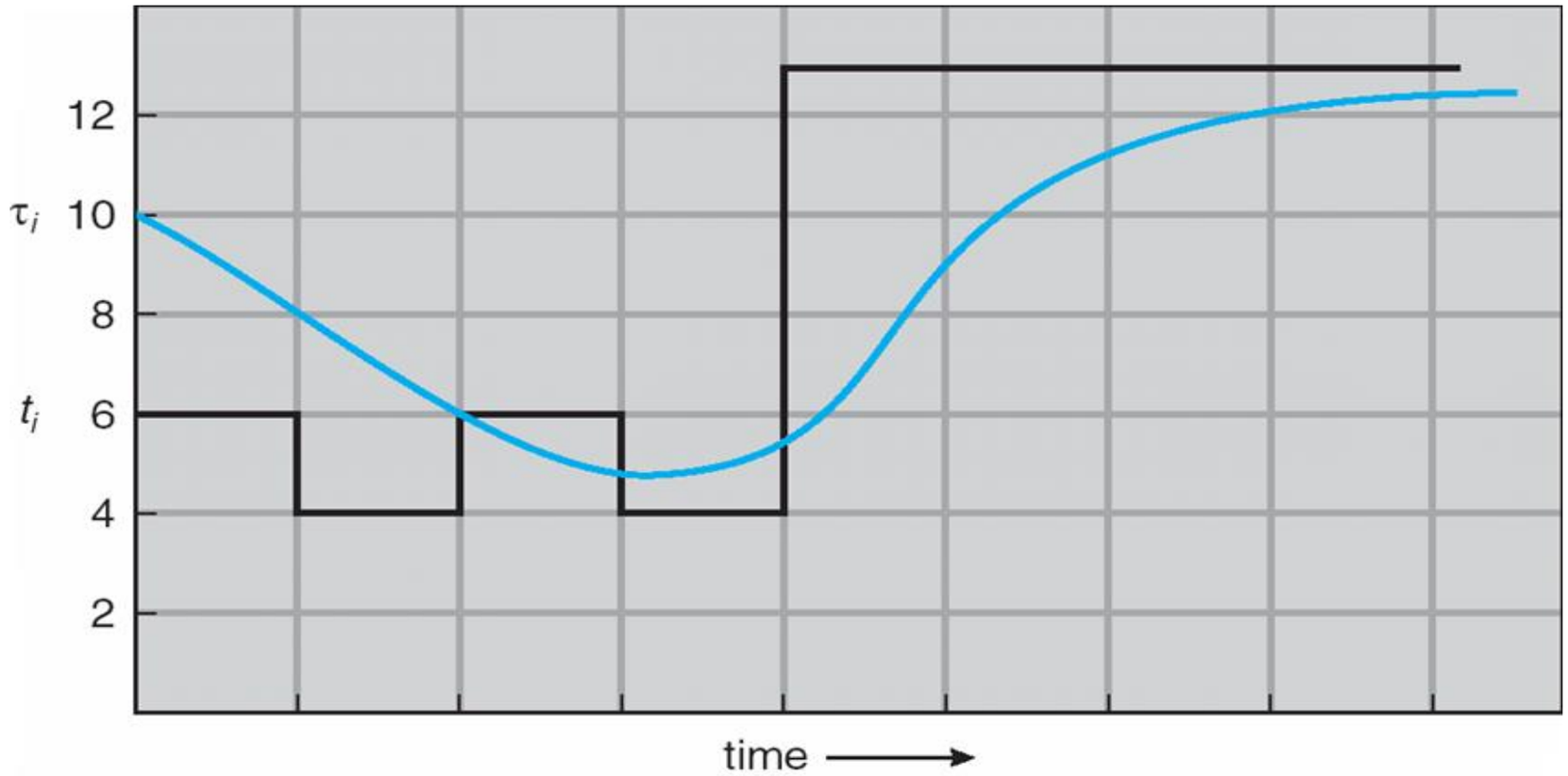
- Üstel ortalama ve bir önceki CPU patlama süresi kullanılarak hesaplanabilir

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define:

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n$$

- Genelde , α = ½ seçilir. α parametresi tahminimizdeki en son ve geçmişteki işlemin bağlı ağırlığını kontrol eder.
- preemptive versiyonu **en kısa kalan zaman önce** olarak adlandırılır

Bir Sonraki CPU Patlamasının Süresinin Tahmini



CPU burst (t_i)		6	4	6	4	13	13	13 ...
"guess" (τ_i)	10	8	6	6	5	9	11	12 ...

Üstel Ortalama Örnekleri

- α =0
  - $\tau_{n+1} = \tau_n$
  - Son kayıtlar hesaba katılmaz
- α =1
  - $\tau_{n+1} = \alpha\, t_n$
  - Sadece en son gerçek CPU patlaması hesaba katılır
- Eğer formülü genişletirsek:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

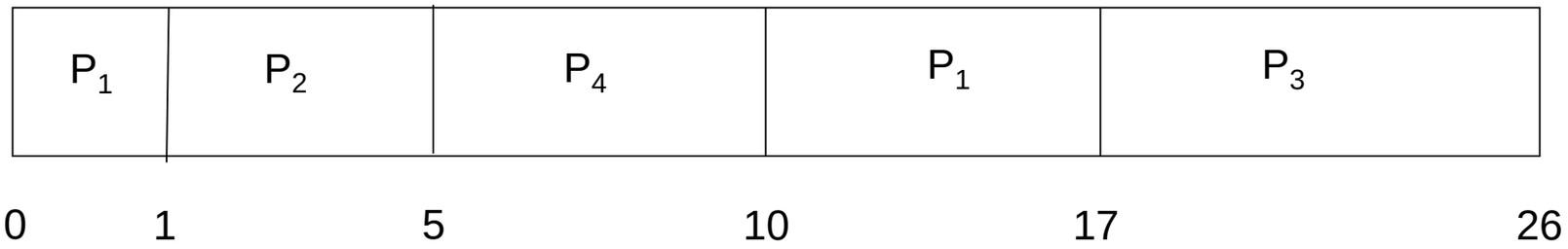
- α ve (1 - α) 1 e eşit veya daha küçük olduğu için, her bir terim kendisinden bir öncekine göre daha az ağırlığı vardır

En Kısa Kalan Zaman Önce Örneği

- Bu örnekte farklı varış zamanı ve kesintili olan prosesleri analiz ederiz.

Proses	*Varış Zamanı*	Patlama Zamanı
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Kesintili* SJF Gantt Diyagramı



- Ortalama Bekleme zamanı= [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

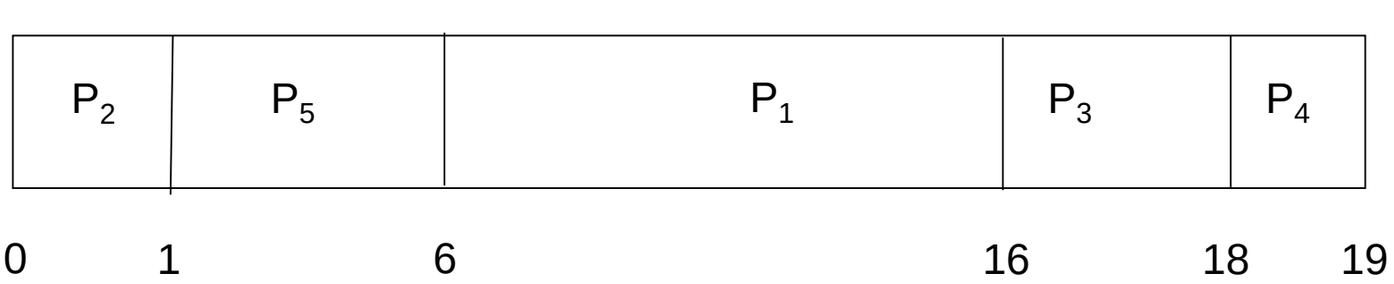
Öncelikli Çizelgeleme

- Bir öncelik sayısı (tamsayısı) her bir prosese atanır
- CPU en yüksek öncelik değerine sahip prosese tayin edilir (en küçük tamsayı ≡ en yüksek öncelik)
- SJF bir öncelikli çizelgelemedir (tahmini bir sonraki CPU patlama zamanının tersinin öncelik olduğu bir öncelikli iş sıralama yaklaşımıdır)
- Problem ≡ **Açlıktan Ölme** (**Starvation)** – düşük öncelikli prosesler hiç çalışmayabilir
- Çözüm ≡ **Yaşlandırma** – zaman ilerlerken proses önceliğini artır

Öncelikli İş Sıralama Örneği

Proses	Patlama Zamanı	Öncelik
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

- Öncelikli İş Sıralama Gantt Diyagramı



- Ortalama bekleme zamanı = 8.2 msec

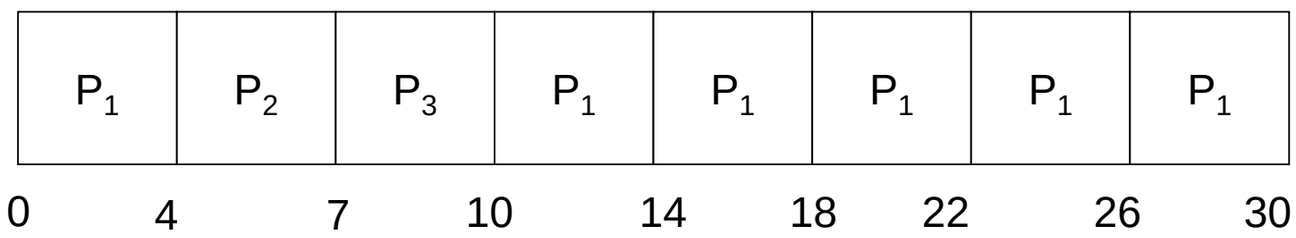
Çevrimsel Sıralı (Round Robin - RR)

- Her bir proses genellikle 10-100 milisaniye arası küçük bir zaman süresince CPU'ya sahip olur .
- Bu zaman değerine **kuantum zamanı** denir ve q ile gösterilir.
- Bu süre tamamlandıktan sonra proses kesilir ve hazır kuyruğunun sonuna eklenir.
- Eğer hazır kuyruğunda n adet proses varsa ve kuantum zamanı değeri q ise her bir proses 1/n kadar CPU 'yu elde eder. Hiçbir proses $(n\text{-}1)q$ süresinden daha fazla beklemez
- Zamanlayıcı bir sonraki prosesi devreye sokmak için her bir kuantum sonunda keser.
- Performans
  - q büyük olursa $\Rightarrow$ FIFO gibi çalışır
  - q küçük$\Rightarrow$ q bağlam-içerik değişimine göre büyük olmalıdır aksi halde sistem kasılır

RR Örneği (q = 4)

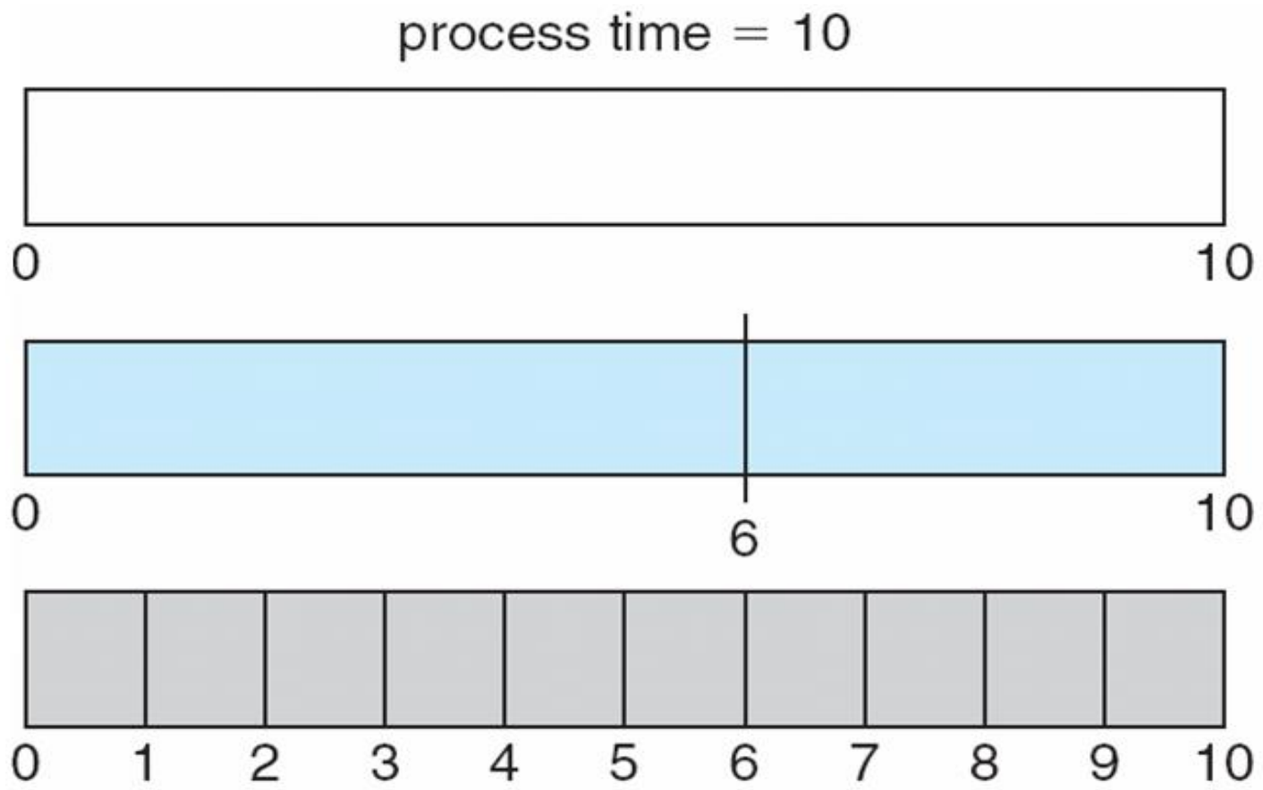
Process	Patlama Zamanı
P_1	24
P_2	3
P_3	3

- Gantt diyagramı:



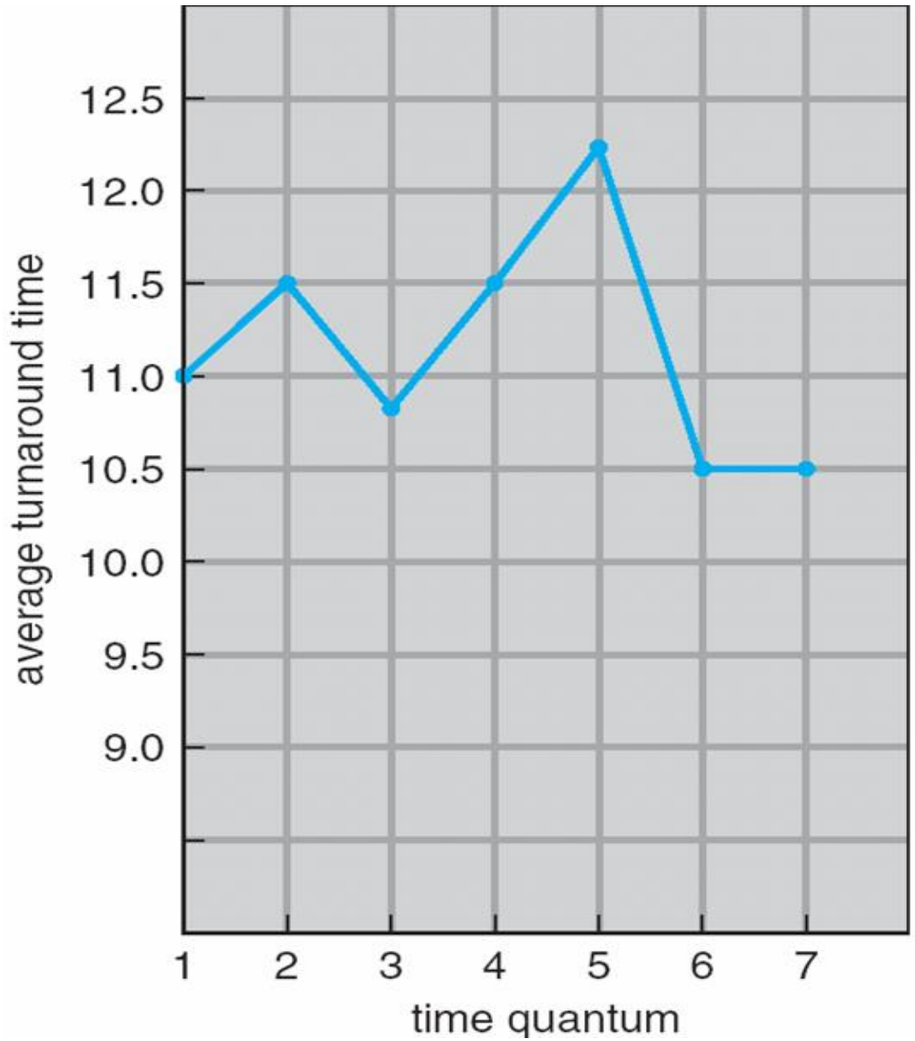
- Tipik olarak, SJF'den daha yüksek bir iş bitirme zamanı, ancak daha iyi bir cevap zamanına sahiptir.
- q bağlam değişimi zamanına göre büyük olmalıdır
- q genellikle 10ms ila 100ms, bağlam değişimi < 10 usec

Kuantum Zamanı and Bağlam Değişim Zamanı



quantum	context switches
12	0
6	1
1	9

Tamamlanma Zamanının Kuantum Zamanıyla Değişimi



process	time
P_1	6
P_2	3
P_3	1
P_4	7

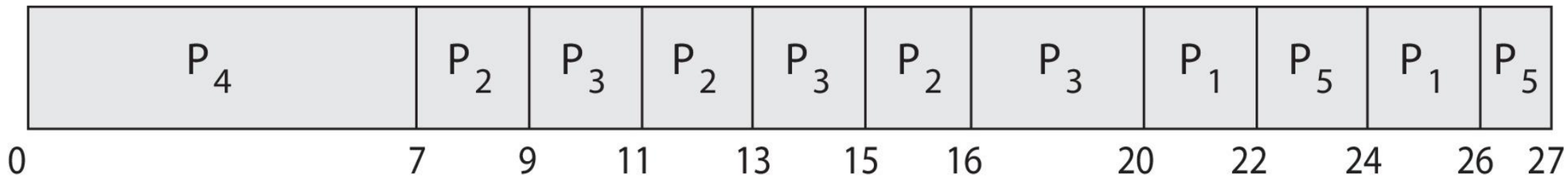
CPU patlamalarının 80% i
q dan daha küçük olmalıdır

Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

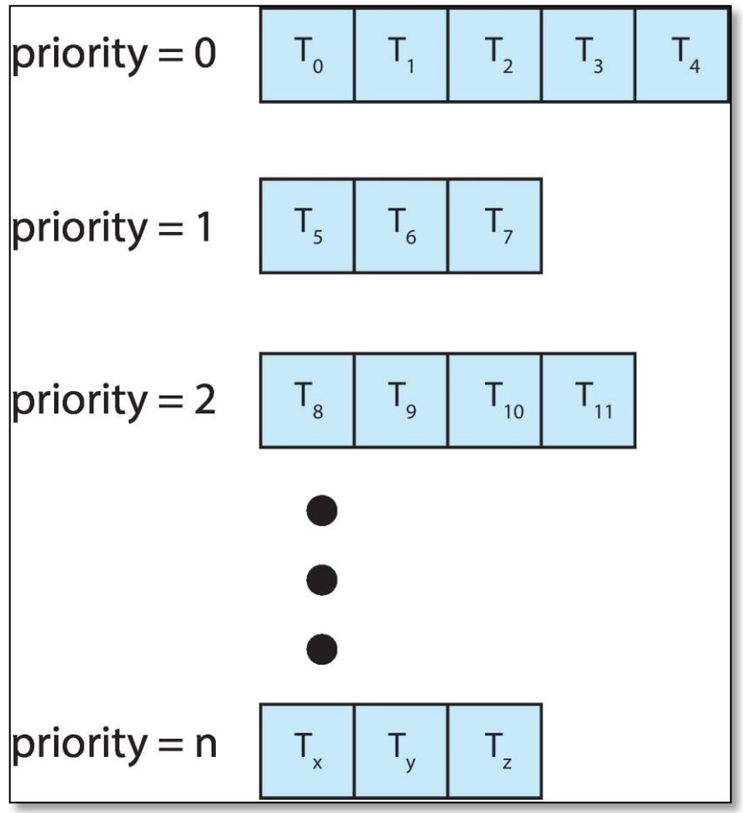
- Prosesi en yüksek önceliğe sahip olarak çalıştırın. Aynı önceliğe sahip prosesler round robinle çalışır

- Gantt Grafiği 2 ms süreli kuantum

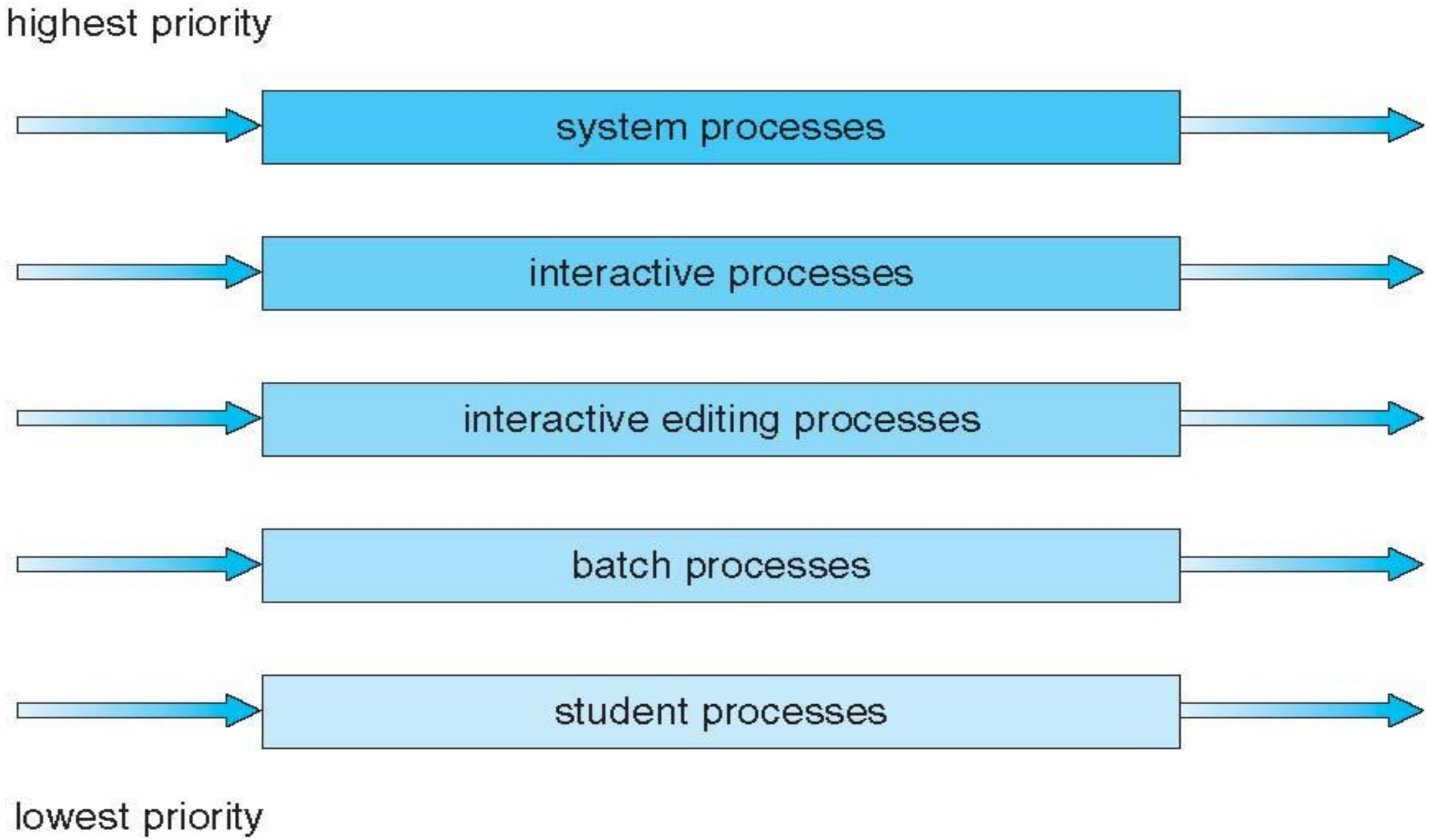


Çok Seviyeli Kuyruk

- Hazır kuyruğu ayrı kuyruklar halinde düzenlenir, ör:
  - Ön plan (interaktif)
  - Arkaplan (toplu iş - batch)
- Herbir kuyruğun farklı yanıt süresine ihtiyaçları vardır
- Proses sürekli bir kuyruktadır
- Her kuyruk kendi çizelgeleme algoritmasına sahiptir.
- ön plan – çevrimsel sıralı - RR
- arka plan – FCFS
- İş sıralama kuyruklar arasında yapılmalıdır:
  - Sabit öncelikli iş sıralama (ön plandakilerin tümü bittikten sonra arka plandakilere hizmet edilir). Ölüm olasılığı.
  - Zaman dilimli iş sıralama– her kuyruk belirli bir CPU zamanını elde eder ve prosesleri arasında paylaştırır, RR' de ön plana kadar 80%
  - FCFS'de arka plan için 20%



Çok Seviyeli Kuyruk Çizelgeleme



Çok Seviyeli Geri Beslemeli Kuyruk

- Bir proses çeşitli kuyruklar arasında hareket edebilir; yaşlandırma bu şekilde uygulanabilir

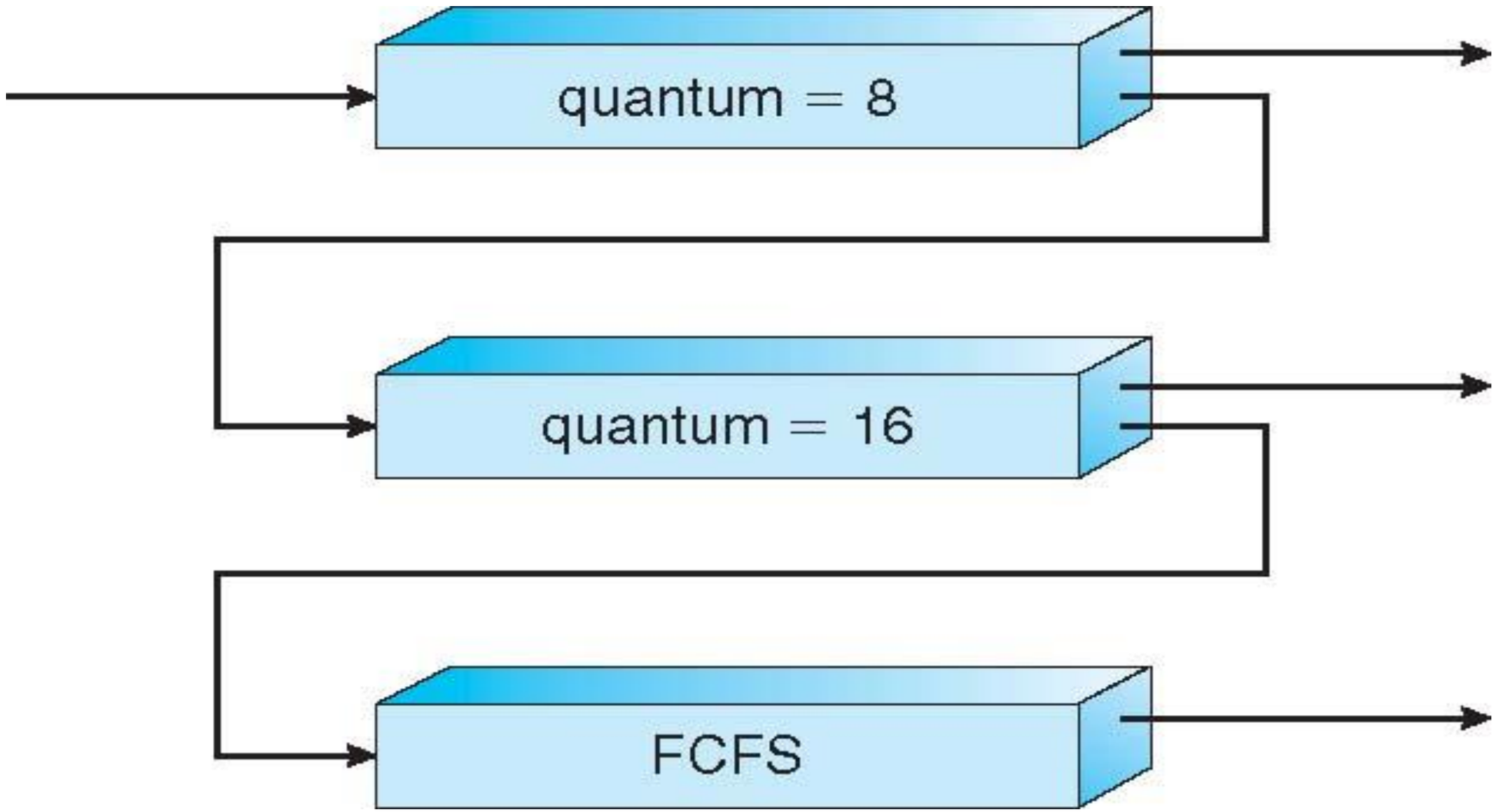
- Çok seviyeli-geri besleme kuyruğu iş sıralama işlemi aşağıdaki parametreler ile tanımlanır :
  - Kuyruk sayısı
  - Her kuyruğun iş sıralama algoritması
  - Bir prosesin ne zaman bir üst kuyruğa geçeceğini belirleme yöntemi
  - Bir prosesin ne zaman bir alt kuyruğa geçeceğini belirleme yöntemi
  - Bir prosesin çalışmaya ihtiyaç duyduğunda hangi kuyruğa ekleneceğini belirleme yöntemi

Çok Seviyeli Geri Besleme Kuyruğu Örneği

- Üç adet kuyruk:
  - Q_0 – 8 milisaniye kuantum değerine sahip RR
  - Q_1 – 16 milisaniye kuantumlu RR
  - Q_2 – FCFS

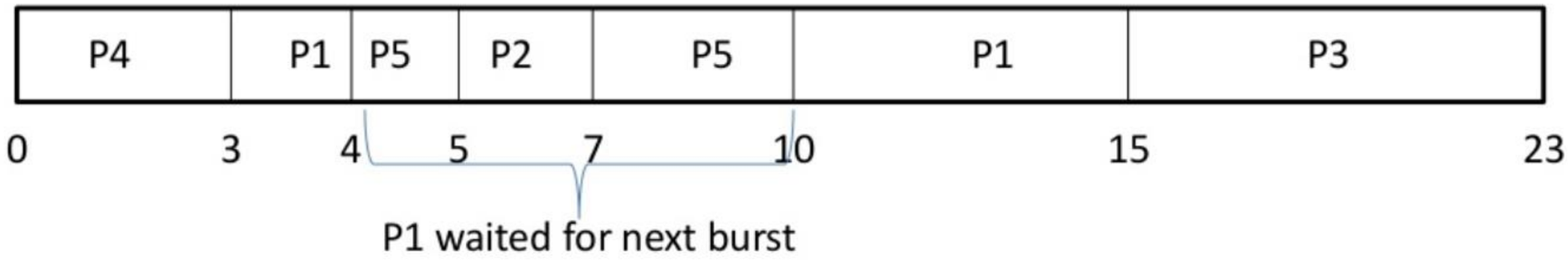
- İş Sıralama
  - Yeni bir görev Q_0 kuyruğuna girer ve FCFS olarak hizmet görür
    - CPU'yu ele geçirdiğinde 8 milisaniyesi vardır
    - 8 milisaniyede işini bitiremezse, Q_1 kuyruğuna geçer
  - Q_1 kuyruğunda görev yine FCFS gibi işlenir ve ilave 16 milisaniye alır
    - hala işini tamamlayamazsa kesilir ve Q_2 ye iletilir.

Çok Seviyeli Geri Besleme Kuyruğu



Quiz

process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4



Waiting time = Start time – Arrival time + wait time for next burst

P4 = 0 - 0 = 0

P1 = (3 - 2) +6 = 7

P2 = 5 - 5 = 0

P5 = 4 – 4+2 = 2

P3 = 15 - 1 = 14

Av. Waiting Time = $\frac{0+7+0+2+14}{5} = \frac{23}{5} = 4.6$

Çok Seviyeli Geri Besleme Kuyruğu

process	Burst Time
p1	53
p2	17
p3	68
p4	24

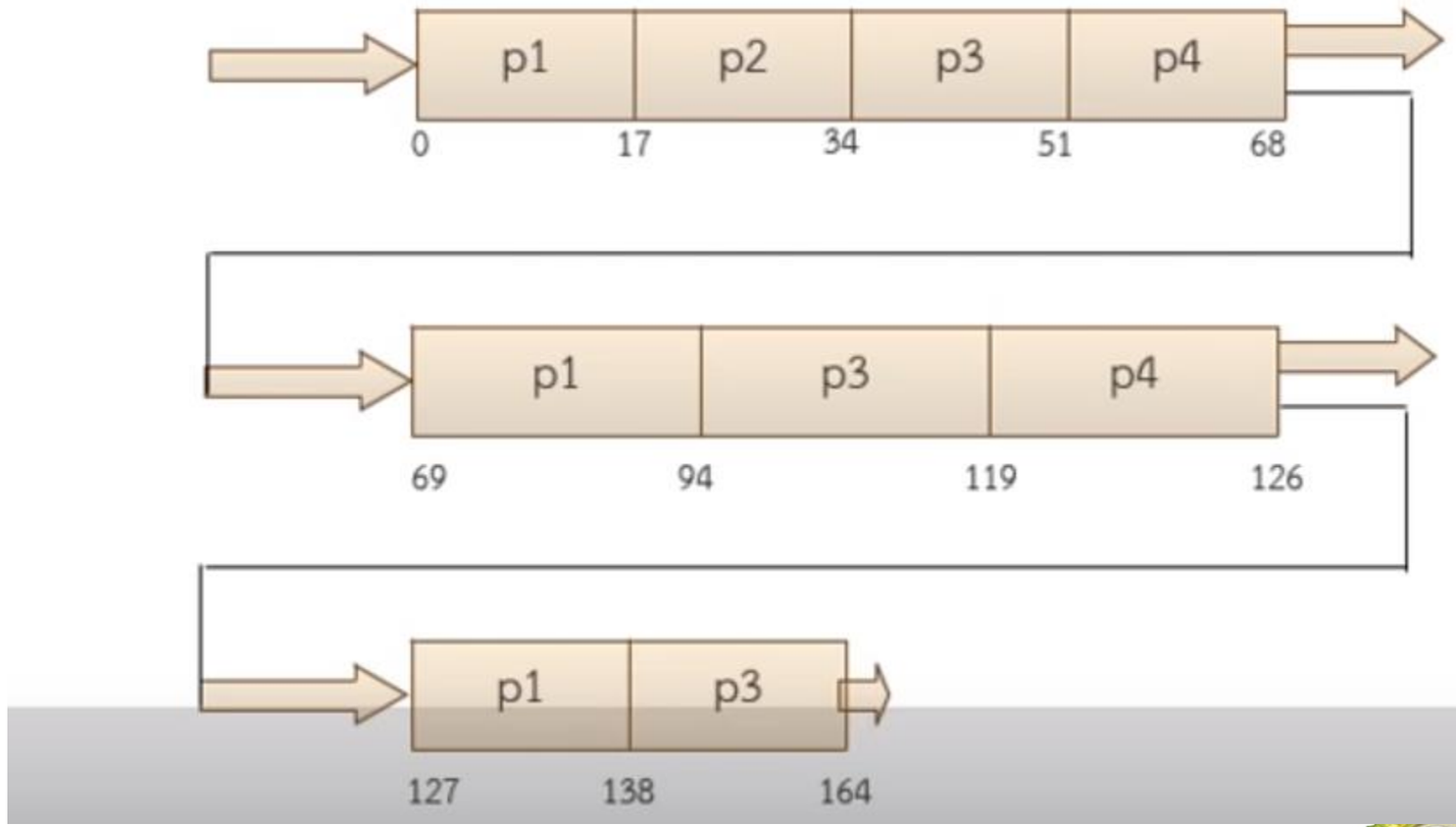
Qo = RR ; Quantum= 17
Q1 = RR ; Quantum= 25
Q2 = FCFS ;

Turnaround time : p1 = 138 ; p2 = 17 ; p3 = 130 ; p4 = 75 ;

Average Turnaround time : (138 + 17 + 130 + 75)/4 = 90

Waiting time : p1 = 85 ; p2 = 17 ; p3 = 96 ; p4 = 102 ;

Average Waiting time : (85 + 17 + 96 + 102)/4 = 75

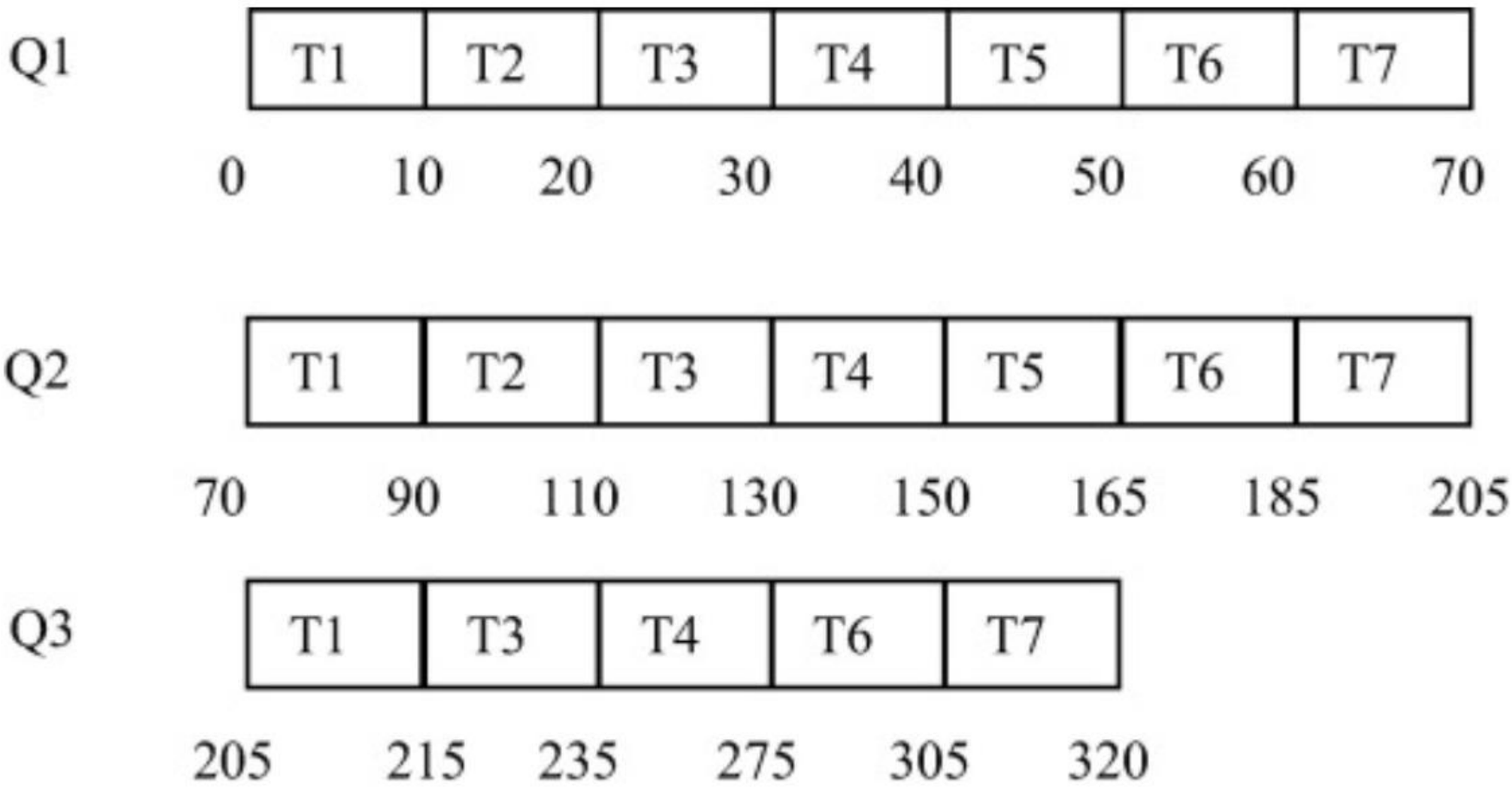


Çok Seviyeli Geri Besleme Kuyruğu

Task name	Arrival time (ms)	Burst time (ms)
T1	0	40
T2	0	30
T3	0	50
T4	2	70
T5	4	25
T6	6	60
T7	7	45

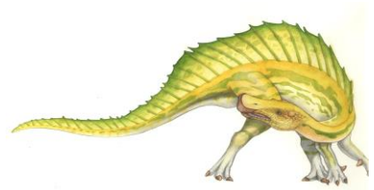
Q0 =RR Quantum=10
Q1 =RR Quantum=20
Q2 =FCFS

Çok Seviyeli Geri Besleme Kuyruğu



İş Parçacığı Sıralama

- Kullanıcı-seviyeli ve çekirdek-seviyeli iş parçacıkları arasında fark vardır. Çekirdek-seviyeli iş parçacıklarını işletim sist. sıralar. Kullanıcı-seviyeli iş parçacıkları ise iş parçacığı Kütüphaneleri tarafından yönetilir.
- İş parçacıkları desteği varsa prosesler değil iş parçacıkları sıralanır
- Çoktan-teke ve çoktan-çoka modelleri, iş parçacığı kütüphanesi kullanıcı seviyeli iş parçacıklarını sıralar, (LWP üzerinde çalışması için)
- İş sıralama çekişmesi proses içinde olduğu için proses çekişme kapsamı (**process-contention scope-PCS) olarak bilinir**
  - Programcı tarafından öncelik ayarlamasıyla yapılır
- Mevcut CPU üzerinde sıralanan çekirdek iş parçacığı sistem çekişme kapsamı (**system-contention scope-SCS)** dır– sistemdeki tüm iş parçacıkları arasında çekişme
- Bire bir modeli kullanan sistemler, örneğin Windows, Linux ve Solaris, yalnızca SCS kullanarak iş parçacıkları çizelgelenir.



Pthread İş Sıralama

- API iş parçacığının oluşturulması sırasında PCS veya SCS olup olmamasını belirtmeyi olanaklı kılar
  - PTHREAD_SCOPE_PROCESS PCS ile iş parçacıklarını sıralar
  - PTHREAD_SCOPE_SYSTEM SCS ile iş parçacıklarını sıralar
- OS tarafından sınırlandırılabilir – Linux ve Mac OS X sadece PTHREAD_SCOPE_SYSTEM i kullanır

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

Pthread Scheduling API

```
/* set the scheduling algorithm to PCS or SCS */
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
/* create the threads */
for (i = 0; i < NUM_THREADS; i++)
   pthread_create(&tid[i],&attr,runner,NULL);
/* now join on each thread */
for (i = 0; i < NUM_THREADS; i++)
   pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

Çok İşlemcili İş Sıralama

- Birden fazla CPU varsa iş sıralama daha karmaşık olur.
- Tek bir blok içerisinde **homojen işlemciler**
  - Multicore CPUs
  - Multithreaded cores
  - NUMA systems
  - Heterogeneous multiprocessing
- **Asimetrik çok işlemcili yapı** – veri paylaşımını azaltmak için sadece bir adet işlemci sistem aktivitelerini(çizelgeleme, I/O işlemleri) yapar. Diğer işlemciler kullanıcı kodlarını çalıştırır.
- **Simetrik Çok İşlemcili Yapı (SMP)** – her işlemci kendi kendine iş sıralar, tüm prosesler ortak bir hazır kuyruğundadır veya her biri kendi özel hazır kuyruğuna sahiptir
  - Şu anda en yaygın

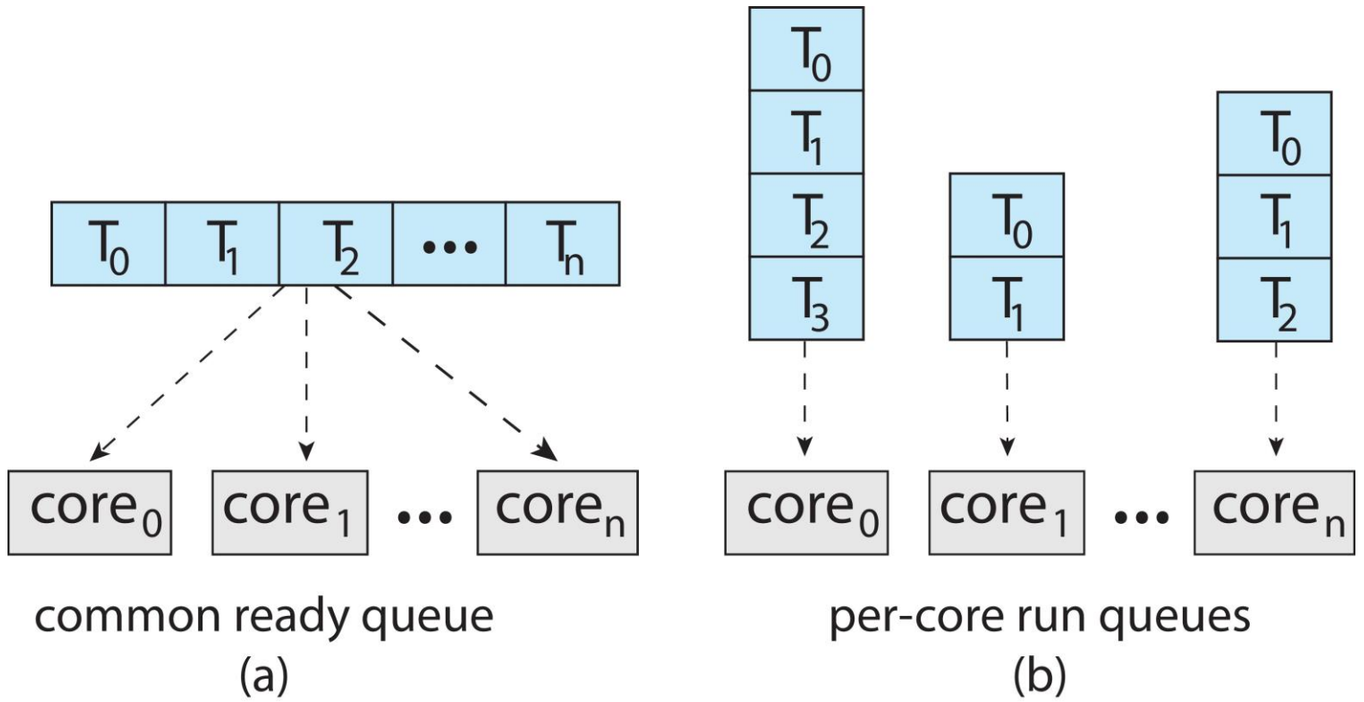
Çok İşlemcili İş Sıralama

- Bir işlemcide çalışan prosesin sık kullandığı veriler cache bellekte saklanır. Eğer bu proses başka işlemciye geçerse o işlemcini cache belleğine bu verilerin yazılması veya o belleğe yavaş erişim sağlanır. Bu yüzden proseslerin işlemciler arası geçişleri çok istenilmez(**İşlemci İlişkisi**)

- **İşlemci İlişkisi**– proses üzerinde çalıştığı işlemci ile ilişkilidir
  - **Hafif ilişki:** Aynı işlemcide tutmaya çalışır ama garanti edemez
  - **Sert ilişki:** Sadece tanımlanmış belirli işlemcide çalışabilir

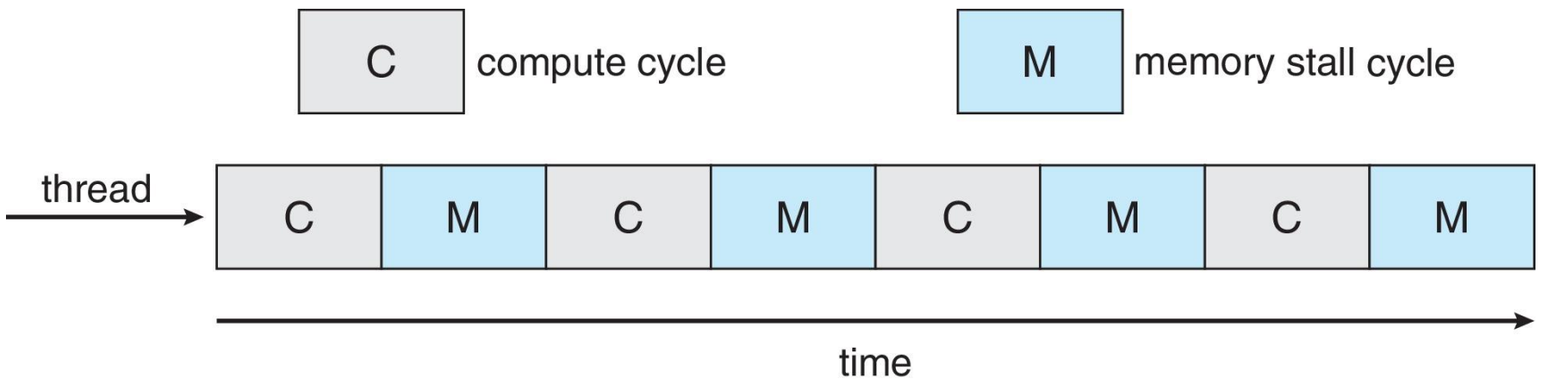
Multiple-Processor Scheduling

- **Simetrik Çok İşlemcili Yapı**(SMP), her işlemcinin kendini zamanlamasıdır.
- Tüm iş parçacıkları ortak bir hazır sırada olabilir (a)
- Her işlemci kendi özel kuyruğuna sahip olabilir (b)



Multicore Processors

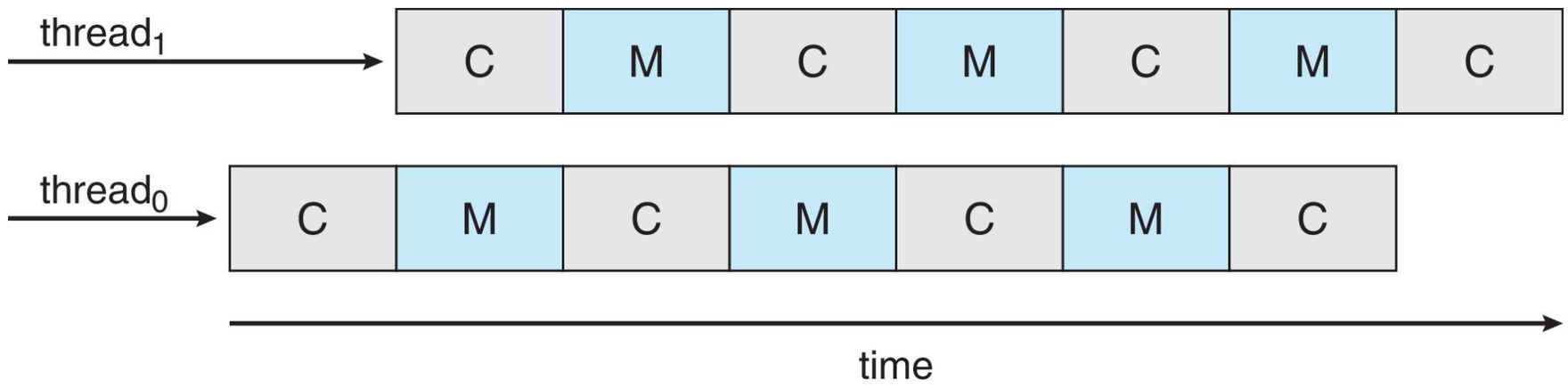
- Aynı fiziksel çipe birden çok işlemci çekirdeği yerleştirme eğilimi
- Daha hızlı ve daha az güç tüketir
- Çekirdek başına birden çok iş parçacığı popüler oluyor
- Bellek geri çekilirken başka bir iş parçacığında ilerleme sağlamak için bellek duraklamasında yararlanır



Multithreaded Multicore System

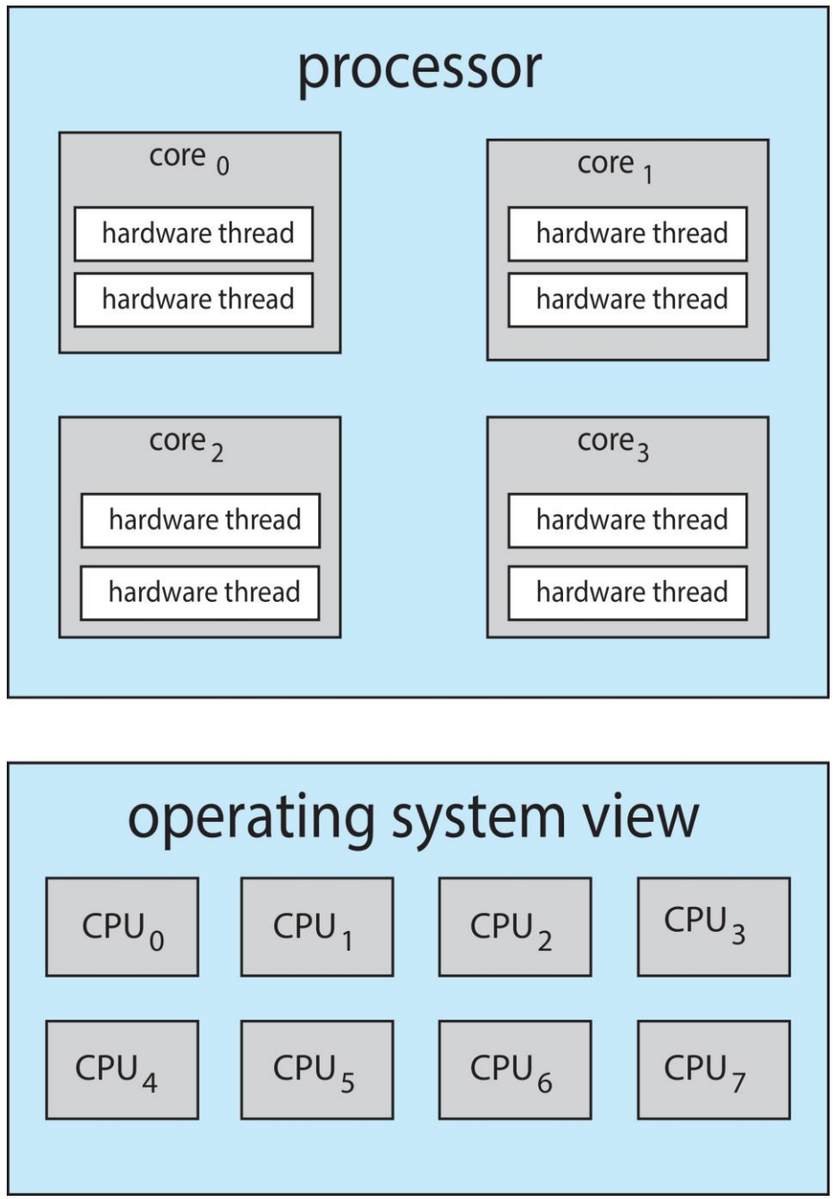
Her çekirdeğin> 1 donanım iş parçacığı vardır.

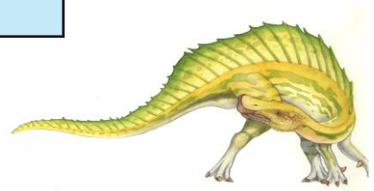
Bir iş parçacığının bir bellek duraklaması varsa, başka bir iş parçacığına geçin!



Multithreaded Multicore System

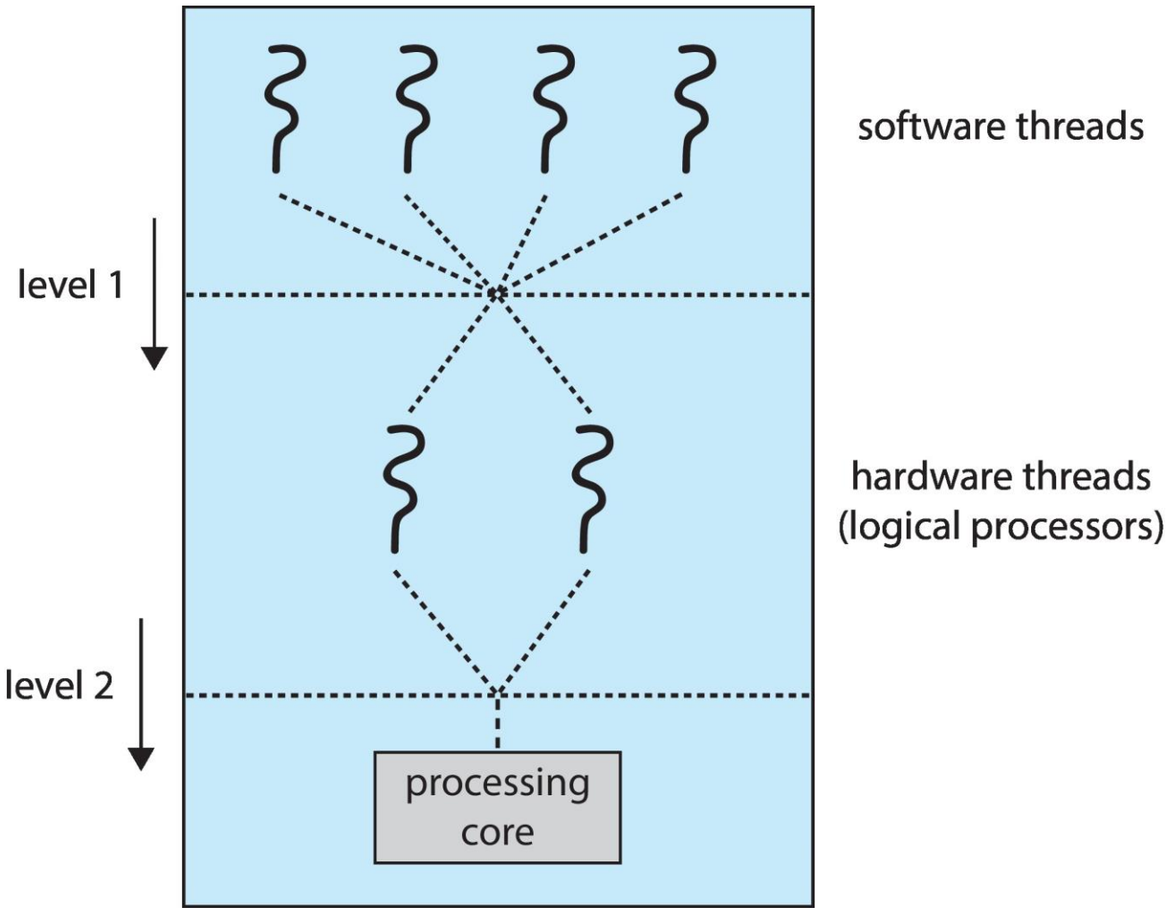
- Yonga-çoklu iş parçacığı (CMT/ **Chip-multithreading**), her bir çekirdek çoklu donanım iş parçacığına atanır. (Intel bunu **hyperthreading** olarak ifade eder.)
- Çekirdek başına 2 donanım iş parçasına sahip dört çekirdekli bir sistemde, işletim sistemi 8 mantıksal işlemci görür.



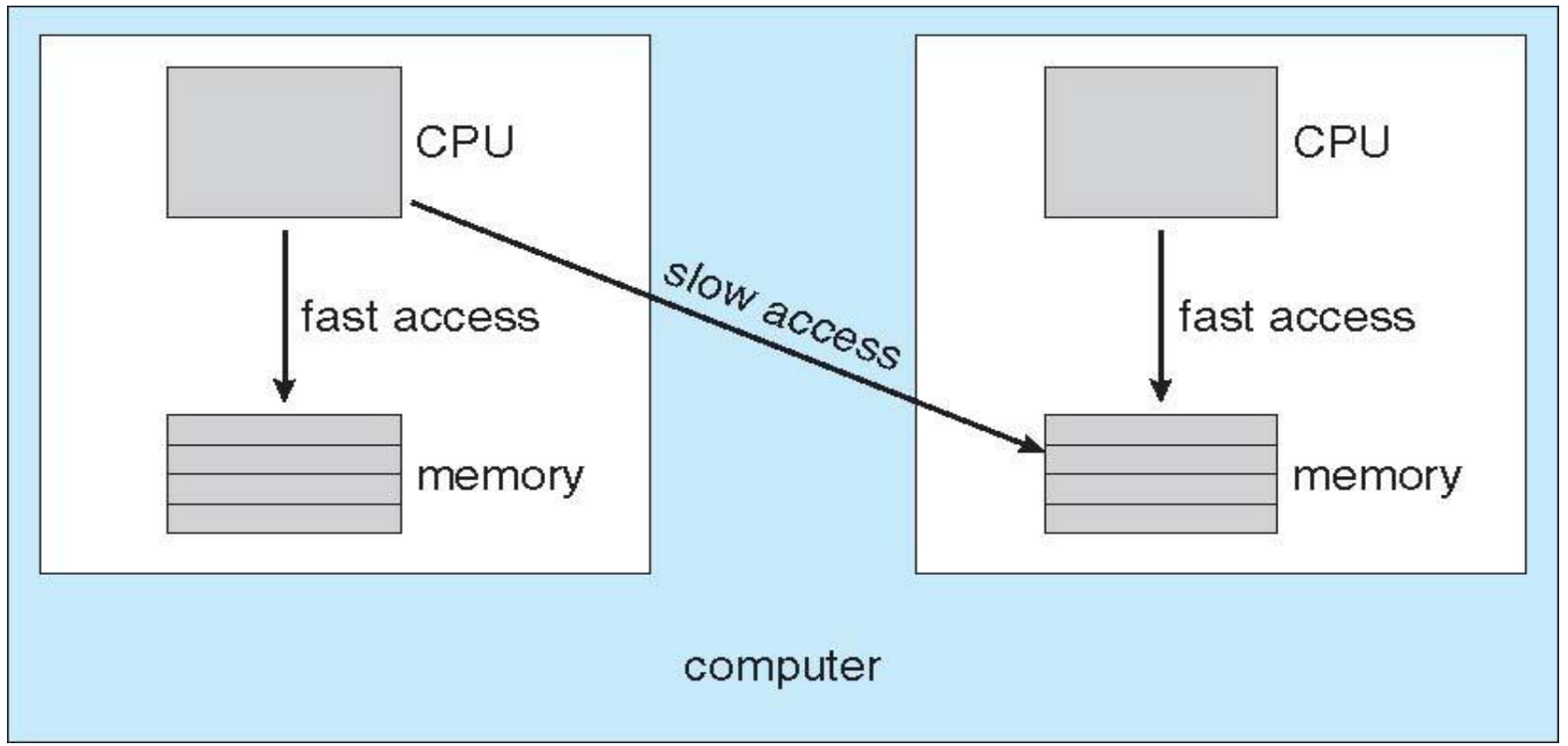


Multithreaded Multicore System

- İki aşamalı programlama:
- İşletim sistemi mantıksal bir CPU üzerinde hangi yazılım iş parçacığının çalışacağına karar verir
- Her bir çekirdek, fiziksel çekirdek üzerinde hangi donanım iş parçacığını çalışacağına karar verir.



NUMA(Non-uniform memory access) & CPU Scheduling



- CPU ana belleğin bazı bölümlerine diğer bölümlere göre daha hızlı erişime sahiptir. Tipik olarak CPU ve bellek aynı devre kartında ise olur. Diğer devre kartındaki belleğe daha yavaş erişir.
- CPU çizelgeleyicisi ve bellek yerleşim algoritmaları birlikte çalışırsa, belirli bir CPU'ya tayin edilmiş bir işleme, CPU'nun bulunduğu kart üzerinde bellek tahsis edilebilir.
- Bellek yerleşim algoritmalarının da CPU ilişkisini(affinity) göz önünde bulundurduğuna dikkat ediniz

Çok Çekirdekli İşlemcilerde Çizelgeleme- Yük dengesi

- SMP (**Symmetric multiprocessing**) ise, tüm CPU'ları verimlilik için yüklü tutmak gerekir
- Yük dengeleme, iş yükünün eşit dağıtılmasını sağlamaya çalışıyor
- Belirli bir görev (Specific task) periyodik olarak her işlemci üzerindeki yükü kontrol eder

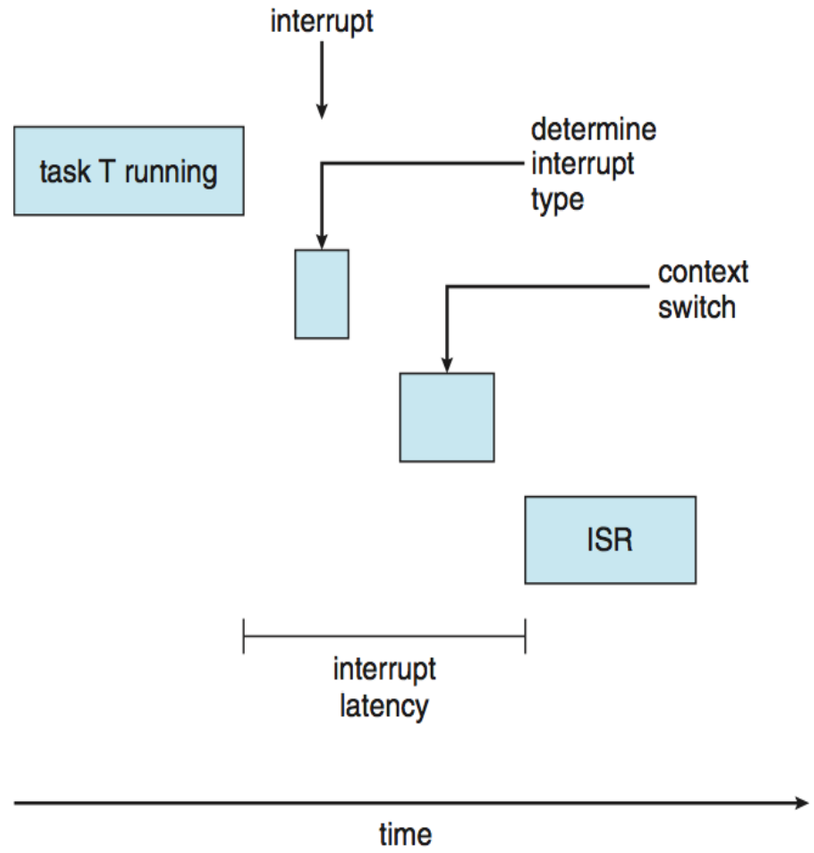
- **İtme yer değişimi (Push migration)** - periyodik görev, her bir işlemcide yükü kontrol eder ve fazla yük bulursa, görevi aşırı yüklü CPU'dan diğer CPU'ya iter
- **Çekme yer değişimi (Pull migration)** - boş işlemciler bekleyen görevi meşgul işlemciden çeker

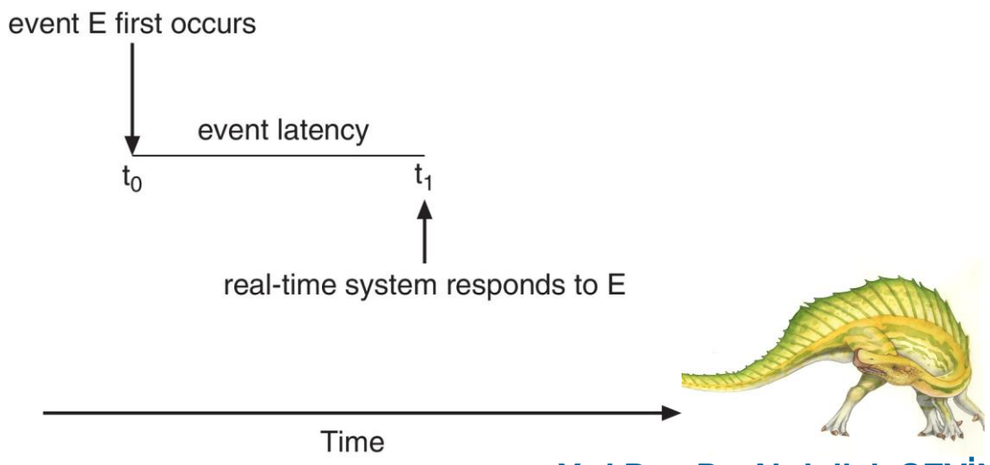
Çok Çekirdekli İşlemciler

- Son eğilim birden fazla işlemci çekirdeğini tek bir fiziksel çip üzerine yerleştirmektir
- Her işlemcinin kendine ait çipi olması durumuna göre Daha hızlı ve daha az güç harcar
- Çekirdek başına birden fazla iş parçacığı
  - Bellekten veri alınırken (**Memory stall-Bellek oyalanması**) harcanan zamanda bir başka iş parçacığı üzerinde devam etmek avantajını kullanır
- **Memory stall-Bellek oyalanması,** birden fazla sebepten dolayı olabilir. İstenilen veri cache de yok olabilir

Real-Time CPU Scheduling

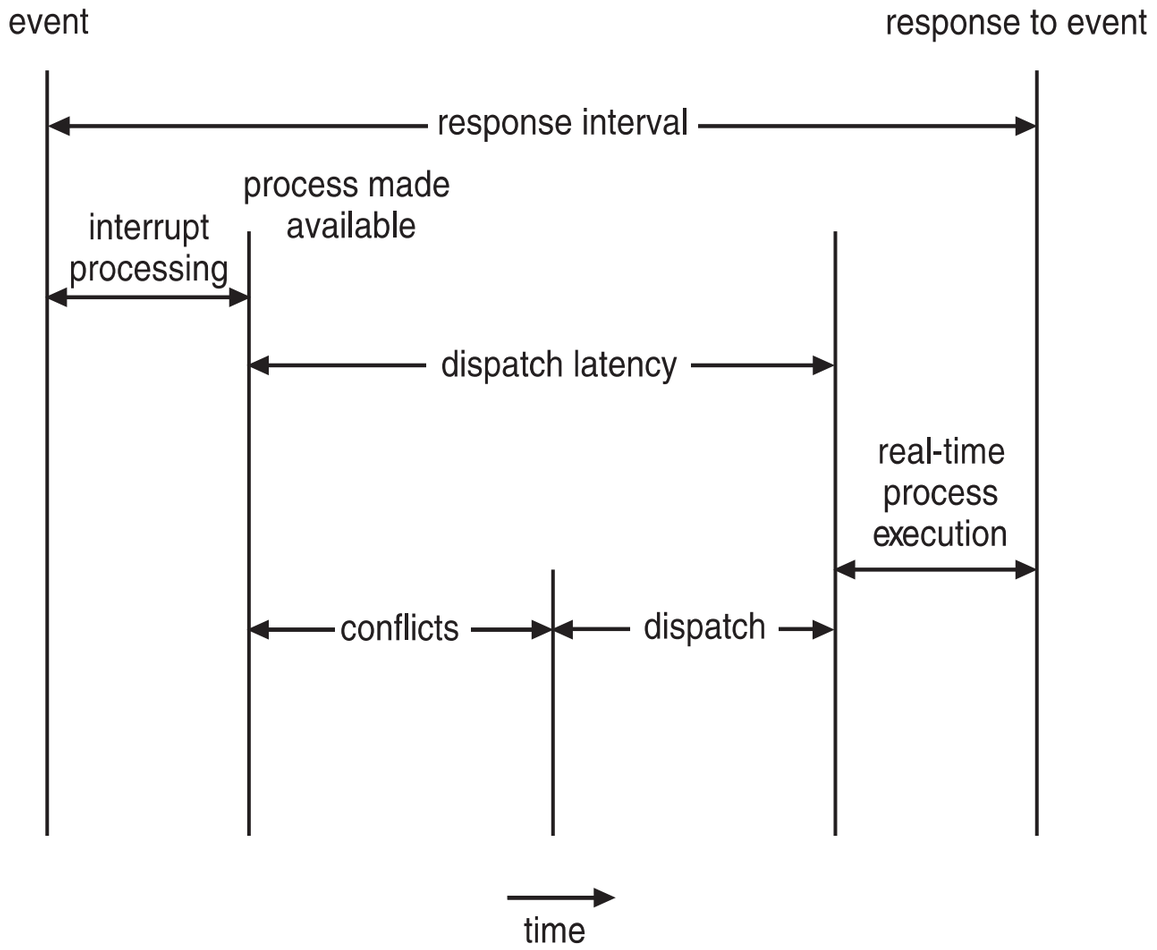
- Belli başlı zorlukları vardır
- **Hafif gerçek-zaman sist.**– gerçek zamanlı kritik bir proses çizelgelendiği zaman gerçekleştirme garantisi yoktur.
- **Katı gerçek-zaman sist.**– verilen görev istenilen zamanda-deadline gerçekleştirilmeli.
- İki tip gecikme performansı etkiler
  1. Kesme gecikmesi – Kesmenin varış zamanı ile kesmeye hizmet verilene kadarki gecikme
  2. Görevlendirme (Dispatch) gecikmesi – Çizelgeleyicinin mevcut işlemi CPU'dan alıp başka bir prosese geçirme zamanı





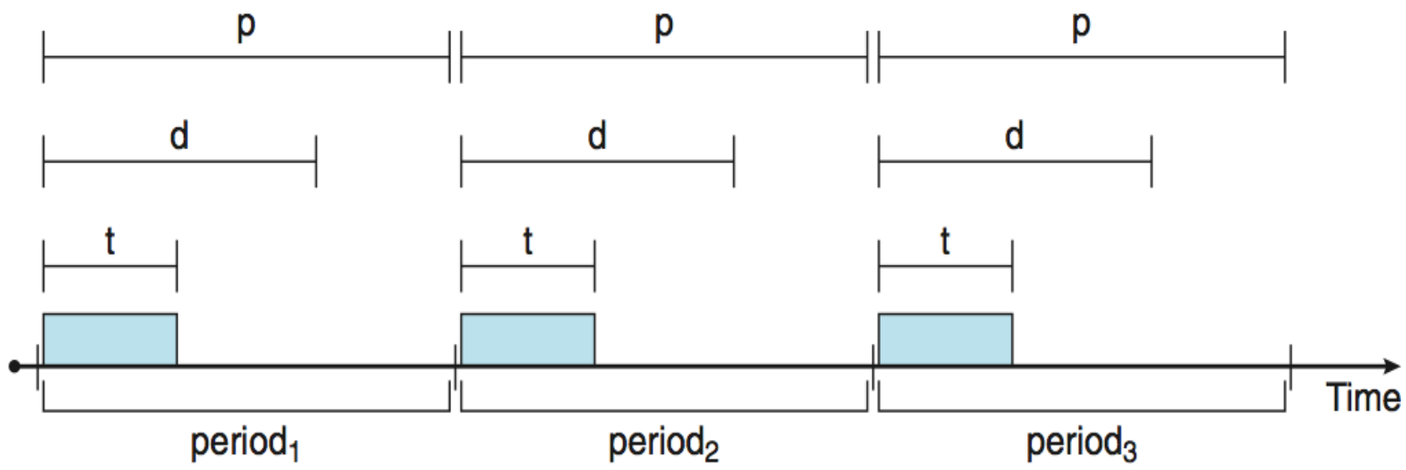
Real-Time CPU Scheduling (Cont.)

- Görevlendirme gecikmesini düşük tutmak için en etkili teknik kesintili yapının kullanılmasıdır.
- Görevlendirme gecikmesine ait çekişme fazı:
  1. Çekirdekte çalışan herhangi bir prosesin önceliklendirilmesi
  2. Yüksek öncelikli bir prosese ihtiyaç duyduğu kaynakların düşük öncelikli prosesler tarafından serbest bırakılması



Öncelik tabanlı çizelgeleme

- Gerçek-zamanlı çizelgeleme için çizelgeleme kesintili ve öncelik tabanlı olmalı
  - Fakat sadece hafif gerçek-zaman garanti edilebilir
- Katı gerçek-zaman için verilen son işlem zamanına kadar gerçekleşmesi sağlanmalıdır
- Proseslere ait yeni karakteristikler: **periyodik** zaman aralıkları ile CPU'da işlem gerekir
  - Proses zamanı *t*, deadline *d*, period *p*
  - $0 \leq t \leq d \leq p$
  - Periyodik görev **hızı** 1/p

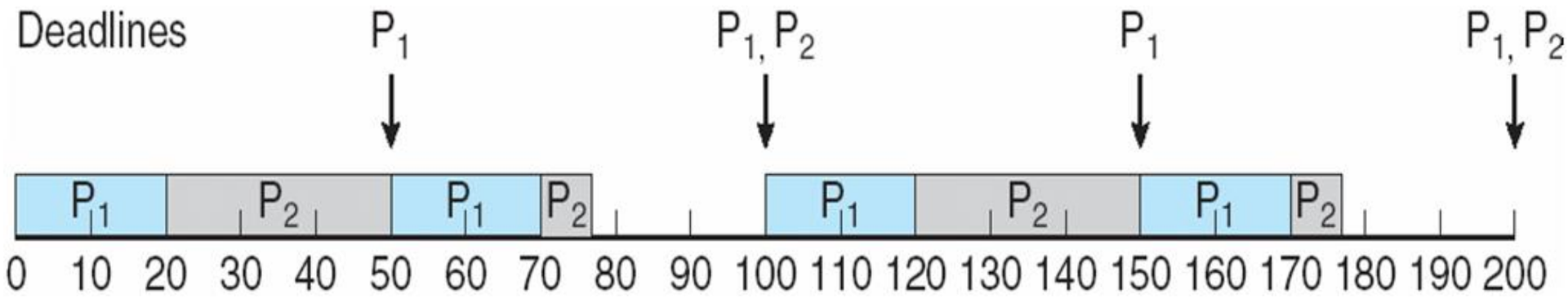


Sanallaştırma ve İş Sıralama

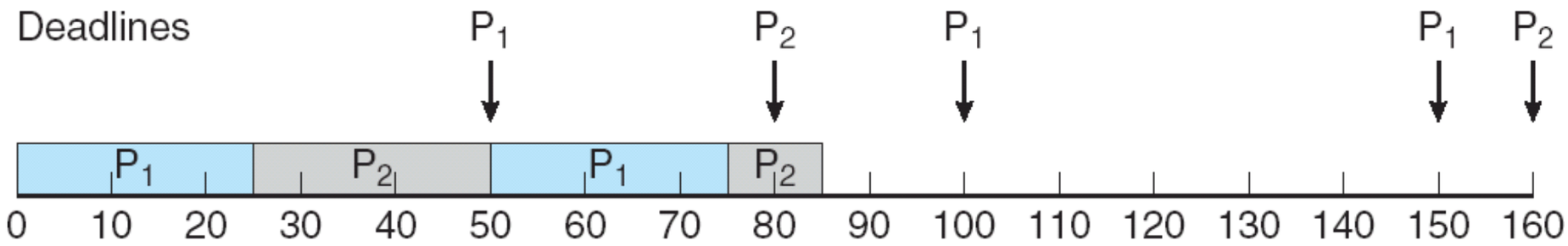
- Sanallaştırma yazılımı, birden fazla sistemi CPU üzerinde çizelgeler
- Her bir konuk kendi iş sıralama işlemini yapar
  - CPU' ya sahip olup olmadığını bilmeyerek
  - Düşük bir cevap zamanına neden olabilir
  - Konukların sistem saati etkilenebilir
- Konukların iyi iş sıralama algoritması çabaları bozulabilir

Rate Montonic Scheduling

- Öncelik periyot zamanının tersine göre atanır
- Kısa periyot = yüksek öncelik;
- Uzun periyot= düşük öncelik
- P_1 ‘ e daha yüksek öncelik atanmıştır P_2’ye göre.

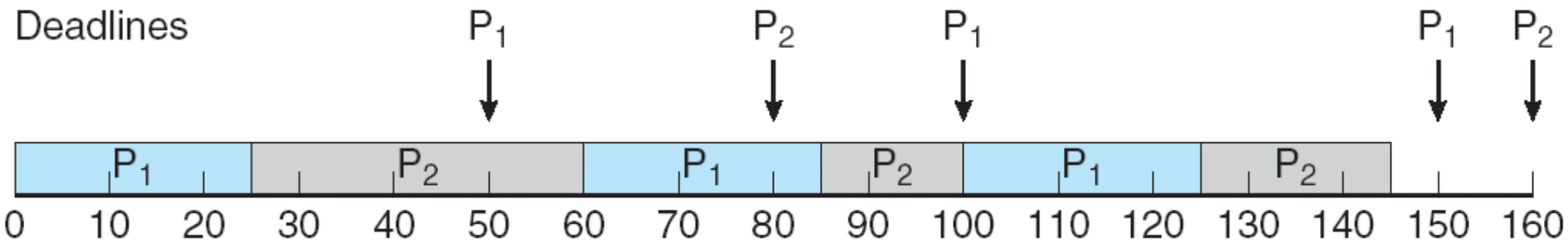


Missed Deadlines with Rate Monotonic Scheduling



Earliest Deadline First Scheduling (EDF)

- Öncelikler, süre bitimi zamanına göre atanır:
  - süre bitimi zamanı düşük olursa, öncelik artar;
  - süre bitimi zamanı fazla ise öncelik düşer



Oransal Pay Çizelgeleme

- *T pay sistemdeki tüm işlemler arasında tahsis edilir*
- *N < T olduğunda uygulama N tane pay alır*
- Bu, her uygulamanın toplam işlemci zamanının N / T'sini almasını sağlar

POSIX Real-Time Scheduling

- The POSIX.1b standard
- API, gerçek-zamanlı iş parçacıklarını yönetmek için fonksiyonlar sunar
- Gerçek-zamanlı iş parçacıklarını çizelgelemek için iki sınıf tanımlanmıştır:

1. SCHED_FIFO – İş parçacıkları FCFS ile çizelgelenir. Eşit önceliğe sahip iş parçacıkları için Zaman dilimleri yoktur
2. SCHED_RR - SCHED_FIFO'ya benzer fakat eşit önceliğe sahip iş parçacıkları için Zaman dilimleri oluşur

- Çizelgelemeyi ayarlama ve okuma için iki fonk. Tanımı var :

1. `pthread_attr_getsched_policy(pthread_attr_t *attr, int *policy)`
2. `pthread_attr_setsched_policy(pthread_attr_t *attr, int policy)`

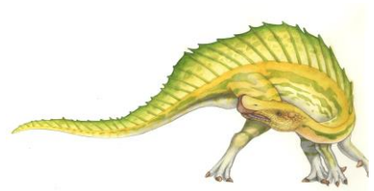
POSIX Real-Time Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[])
{
   int i, policy;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* get the current scheduling policy */
   if (pthread_attr_getschedpolicy(&attr, &policy) != 0)
      fprintf(stderr, "Unable to get policy.\n");
   else {
      if (policy == SCHED_OTHER) printf("SCHED_OTHER\n");
      else if (policy == SCHED_RR) printf("SCHED_RR\n");
      else if (policy == SCHED_FIFO) printf("SCHED_FIFO\n");
   }
```

POSIX Real-Time Scheduling API (Cont.)

```
/* set the scheduling policy - FIFO, RR, or OTHER */
if (pthread_attr_setschedpolicy(&attr, SCHED_FIFO) != 0)
   fprintf(stderr, "Unable to set policy.\n");
/* create the threads */
for (i = 0; i < NUM_THREADS; i++)
   pthread_create(&tid[i],&attr,runner,NULL);
/* now join on each thread */
for (i = 0; i < NUM_THREADS; i++)
   pthread_join(tid[i], NULL);
}

/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```



İşletim Sistemi Örnekleri

- Solaris iş sıralama
- Windows XP iş sıralama
- Linux iş sıralama

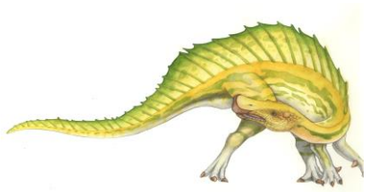
Linux İş Sıralama

- Sabit dereceli karmaşıklık *O*(1) iş sıralama zamanı
- Kesintili ve öncelik tabanlı
- İki adet öncelik aralığı: zaman paylaşımı ve gerçek zamanlı
- **Gerçek zamanlı** aralık 0 dan 99 a dır (Öncelik no)
- Daha yüksek önceliği gösteren nümerik değer düşüktür (global önceliği var)
- Daha yüksek öncelik daha büyük q (quantum zamanı) ya neden olur
- Zaman aralığında kalan zaman süresi kadar görev çalışabilir (**active**)
- Eğer zaman kalmamışsa (**expired**), diğer görevler kendi zaman aralıklarını kullanana kadar çalışamaz
- Tüm çalışabilir görevler CPU başına bir adet çalışır kuyruğunda tutulur
  - İki adet öncelikli dizi (active, expired)
  - Görevler önceliğe göre sıralanır
  - Aktif görev kalmadığında diziler yer değişir

Öncelikler ve Zaman Süresi

numeric priority	relative priority		time quantum
0	highest	real-time tasks	200 ms
•			
•			
•			
99			
100		other tasks	
•			
•			
•			
140	lowest		10 ms

Önceliklere Göre Sıralanmış Görev Listesi



Windows İş Sıralama

- Windows öncelik tabanlı kesintili bir iş sıralama yaklaşımını kullanır
- En yüksek öncelikli iş parçacığı bir sonra çalışır
- *Görevlendirici (Dispatcher) iş sıralayıcıdır/Çizelgeleyicidir*
- İş parçacıkları (1) bloke olana kadar çalışır, (2) zaman periyodu bitimine kadar, (3) daha yüksek öncelikli bir iş parçacığı tarafından kesilene dek çalışır
- Gerçek zamanlı iş parçacıkları olmayanları RT iş parçacıkları kesebilir
- 32-seviyeli öncelik şeması
- **Değişken sınıf** 1-15, **gerçek zamanlı sınıf** 16-31
- 0 önceliği bellek yönetimi iş parçacığıdır
- Her öncelik için kuyruk vardır
- Eğer çalışabilir bir iş parçacığı yoksa, **boşta iş parçacığı** çalışır

Windows Öncelik Sınıflar

- Win32 API prosesin ait olduğu öncelik sınıflarını tanımlar
  - REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
  - REALTIME harici tümü değişkendir
- Belirli bir öncelik sınıfındaki bir iş parçacığı bağıl bir önceliğe sahiptir
  - TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE
- Öncelik sınıfı ve bağıl öncelik ile bir nümerik önceliği oluşturulur
- Temel öncelik NORMAL dir
- Eğer kuantum zamanı aşarsa, öncelik tabanın altına inmeyecek şekilde düşürülür
- Eğer bekleme olursa, ne için beklediğine göre öncelik artırılır

Windows XP Öncelikleri

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

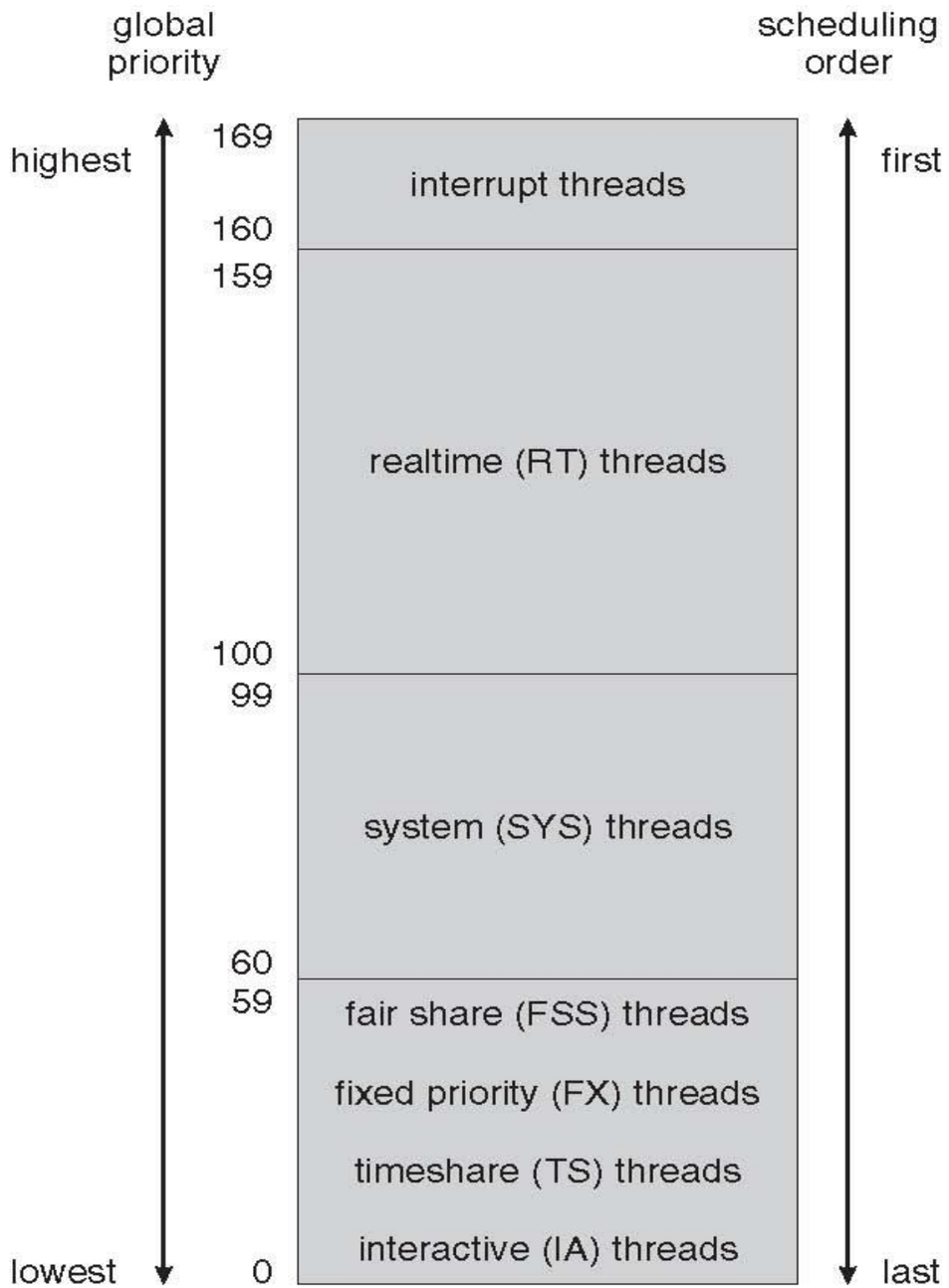
Solaris

- Öncelik tabanlı iş sıralama
- Altı adet sınıf mevcuttur
  - Zaman paylaşımı (varsayılan) (TS)
  - İnteraktif(IA)
  - Gerçek zamanlı(RT)
  - Sistem(SYS)
  - Açık paylaşım (Fair Share (FSS))
  - Sabit öncelik (Fixed priority (FP))
- Belirli bir iş parçacığı herhangi bir zamanda sadece bir sınıfta olabilir
- Her bir sınıf kendi iş sıralama Algoritmasına sahiptir
- Zaman paylaşımı çok seviyeli geri beslemeli kuyruktur
  - Sistem yöneticisi tarafından «yüklenebilir tablo» konfigüre edilebilir

Solaris Görevlendirme Tablosu

priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

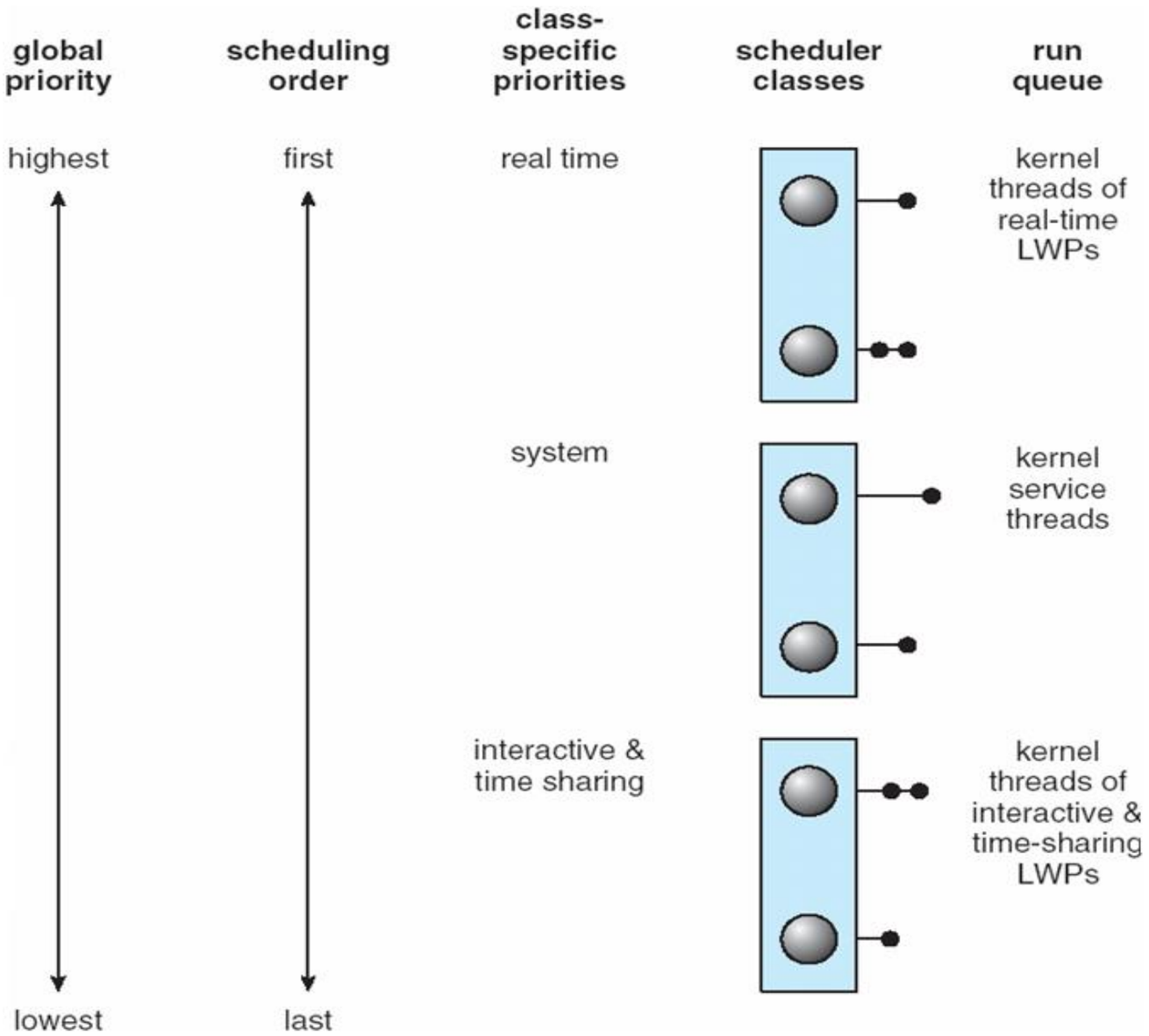
Solaris İş Sıralama



Solaris İş Sıralama(devam)

- İş sıralayıcı sınıfa-özgü öncelikleri iş parçacığına-özgü global önceliğe dönüştürür
  - En yüksek öncelikli iş parçacığı bir sonrakinde çalışır
  - (1) bloke olana kadar çalışır, (2) zaman periyodu kullanılır, (3) daha yüksek öncelikli iş parçacığı tarafından kesilebilir
  - Aynı öncelikli birden fazla iş parçacığı RR'ye göre çalışır

Solaris 2 Scheduling



Java Thread Scheduling

- JVM kesintili ve öncelikli sıralama
- Aynı Öncelikli Birden Çok iş parçacığı var ise, FIFO Kuyruğu kullanılır.

Java Thread Scheduling (Cont.)

JVM Schedules a Thread to Run When:

1. The Currently Running Thread Exits the Runnable State
2. A Higher Priority Thread Enters the Runnable State

Thread Priorities

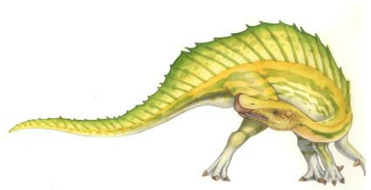
Priority	**Comment**
Thread.MIN_PRIORITY	Minimum Thread Priority
Thread.MAX_PRIORITY	Maximum Thread Priority
Thread.NORM_PRIORITY	Default Thread Priority

Priorities May Be Set Using setPriority() method:

```
setPriority(Thread.NORM_PRIORITY + 2);
```

Algoritma Değerlendirme

- Bir işletim sistemi için CPU iş sıralama algoritması nasıl seçilir ?
- Kriterleri belirle ve daha sonra algoritmaları değerlendir
- Deterministik modelleme
  - Bir tür analitik değerlendirme
  - Belirli bir iş yükünü alır ve o iş yükü için herbir algoritmanın performansını hesaplar

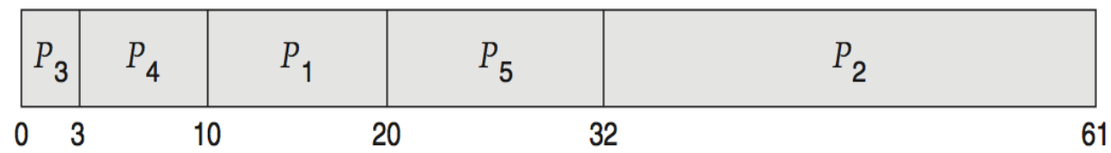


Deterministic Evaluation

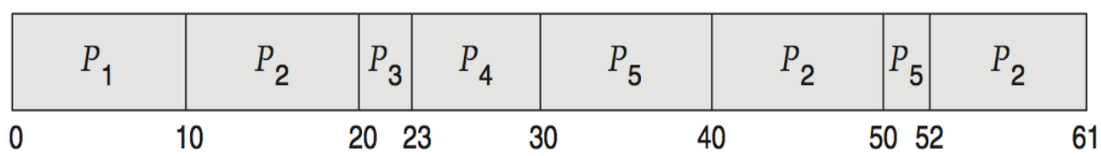
- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs

  - FCS is 28ms:

  - Non-preemptive SFJ is 13ms:



  - RR is 23ms:



Kuyruk Modelleri

- Proseslerin varışını, CPU ve I/O patlamalarını olasılıksal olarak tanımlar
  - Genelde belli bir ortalamaya sahip üstel dağılım tanımlanır
  - Ortalama iş çıkarma oranı(throughput), kullanım oranı, bekleme zamanını hesaplar
- Bilgisayar sistemini her biri bekleyen prosesler kuyruğuna sahip sunucular ağı olarak tanımlar
  - Varış oranı ve hizmet oranı bilinir
  - Kullanım oranını, ortalama kuyruk boyutunu ve ortalama bekleme zamanını hesaplar

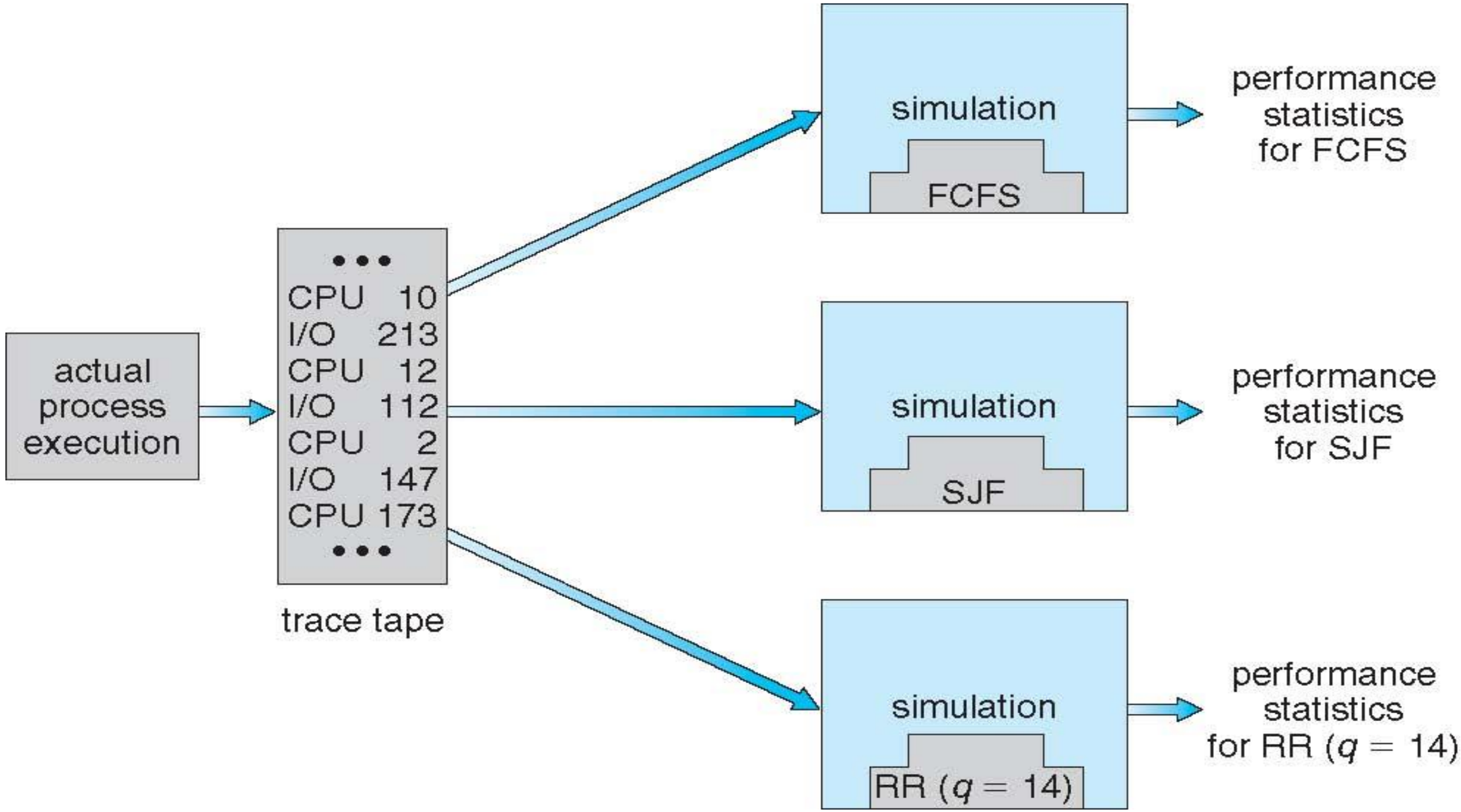
Little Formülü

- *n* = ortalama kuyruk boyutu
- *W* = kuyruktaki ortalama bekleme zamanı
- *λ* = ortalama kuyruğa varış oranı
- Little'ın kuralı – kararlı durumda, kuyruğu terk eden prosesler kuyruğa varanlarla eşit olmalıdır, bu nedenle
- $n = \lambda \times W$
  - Herhangi bir iş sıralama algoritması ve geliş dağılımı için geçerlidir
- Mesela, eğer saniyede ortalama 7 proses varıyorsa ve normalde kuyrukta 14 proses varsa prosesler için ortalama bekleme zamanı = 2 saniye olur.

Simülasyonlar

- Kuyruk modelleri sınırlıdır
- **Simülasyonlar** daha doğru sonuç verir
  - Bilgisayar sistemi modeli
  - Saat değişkendir
  - Algoritma performansını gösteren istatistikler üret
  - Simülasyondaki veriler:
    - Rasgele sayı üreteci
    - Matematiksel dağılımlar
    - Gerçek sistemlerdeki gerçek olayların sırasını tutan kayıtlar

CPU İş Sıralayıcıların Simülasyon ile Değerlendirilmesi



Uygulama

- Simülasyonlar da sınırlı doğruluğa sahiptir
- Sadece yeni iş sıralayıcıyı uyarla ve gerçek sistemlerde test et
  - Yüksek maliyet, yüksek risk
  - Ortamlar değişebilir
- Çok esnek iş sıralayıcılar ortama veya sisteme göre değiştirilebilir
- Veya API'ler öncelikleri değiştirmek maksadıyla geliştirilebilir

5. Bölümün Sonu