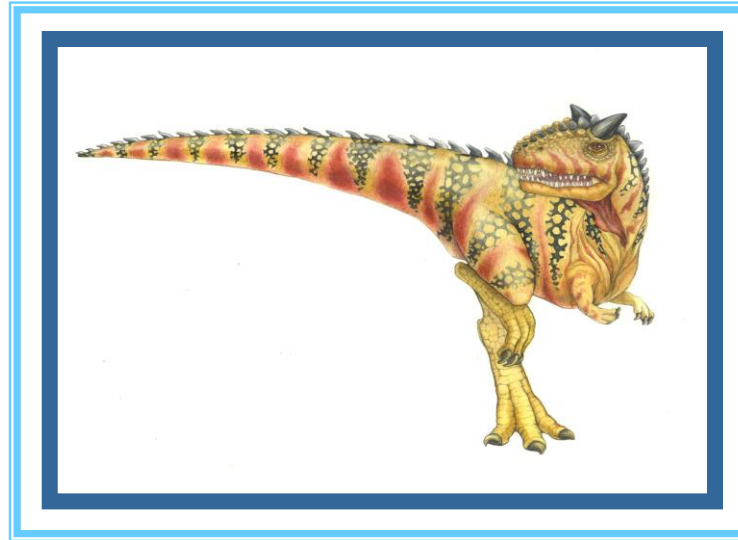


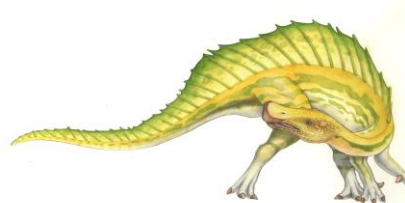
5. Bölüm: İş Sıralama (CPU Scheduling)





5. Bölüm: İş Sıralama (CPU Zamanlama / Planlama / Çizelgeleme)

- Temel Kavramlar
- Sıralama Kriterleri
- İş Sıralama Algoritmaları
- İş Parçacığı Sıralama
- Çoklu-işlemci Sıralama
- İşletim Sistemleri Örnekleri
- Algoritma Değerlendirme





Hedefler

- Çeşitli CPU iş sıralama algoritmalarını tanımlamak
- CPU iş sıralama algoritmalarını değerlendirmek
- Çok işlemcili ve çok çekirdekli iş sıralama ile ilgili sorunları açıklamak
- Çeşitli gerçek zamanlı iş sıralama algoritmalarını tanımlamak
- Windows, Linux ve Solaris işletim sistemlerinde kullanılan iş sıralama algoritmalarını açıklamak
- CPU iş sıralama algoritmalarını değerlendirmek için modelleme ve simülasyon yönteminden faydalanmak
- Belirli bir sistem için CPU İş sıralama algoritma seçim kriterlerini değerlendirmek





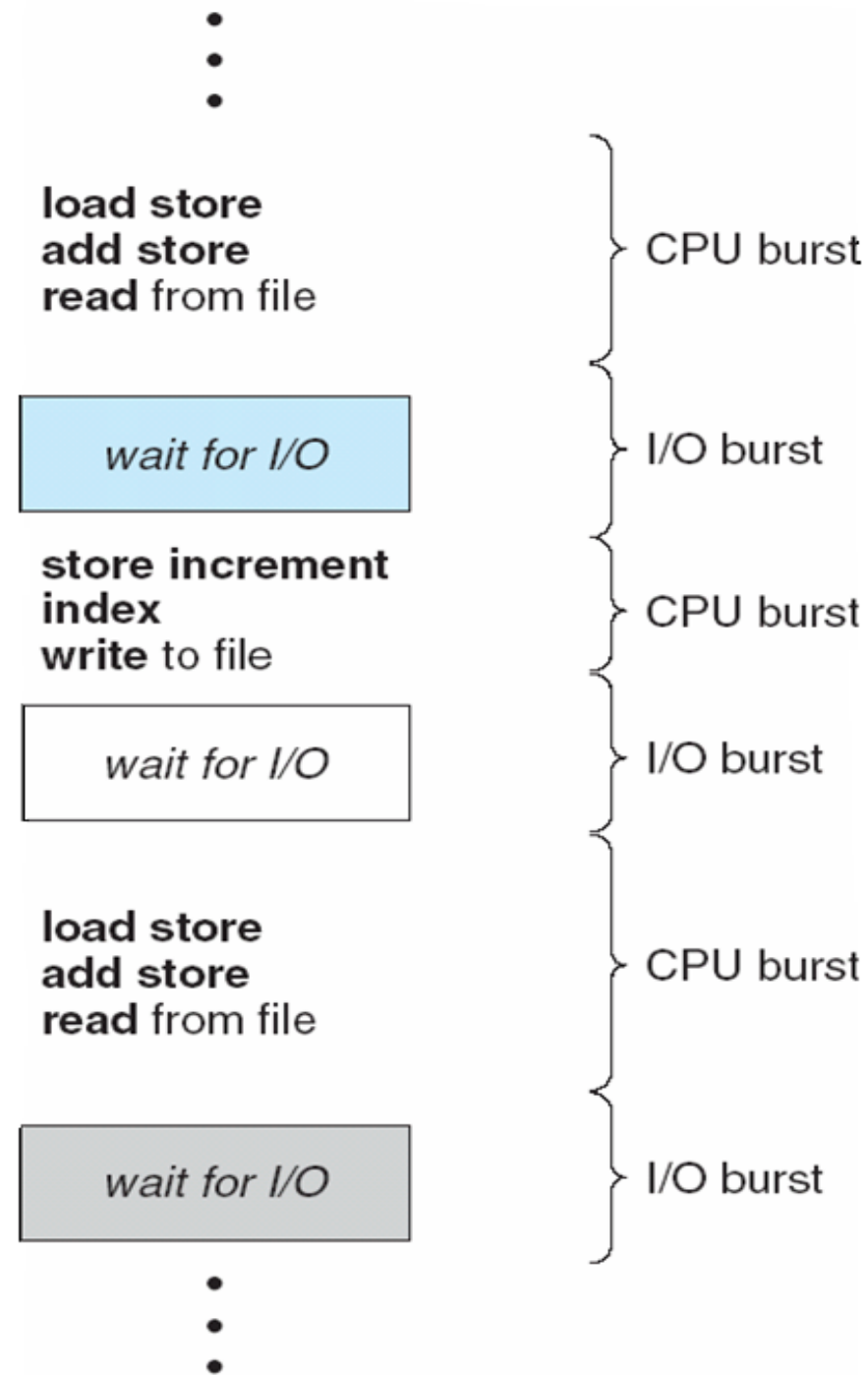
Temel Kavramlar

- Çoklu programlama ile elde edilen maksimum CPU kullanımı
- CPU–I/O Patlama (Burst) Çevrimi – Proses çalışması CPU çalışması **çevrimi** ve G/Ç beklemesinden oluşur
- CPU patlamasını G/Ç patlaması takip eder
- **CPU patlama** dağılımı ana ilgi noktasıdır





CPU ve G/Ç Patlamaları Sırası

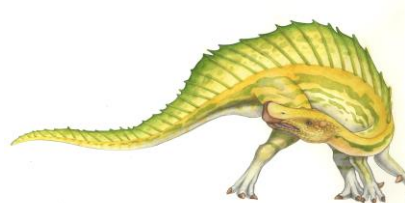
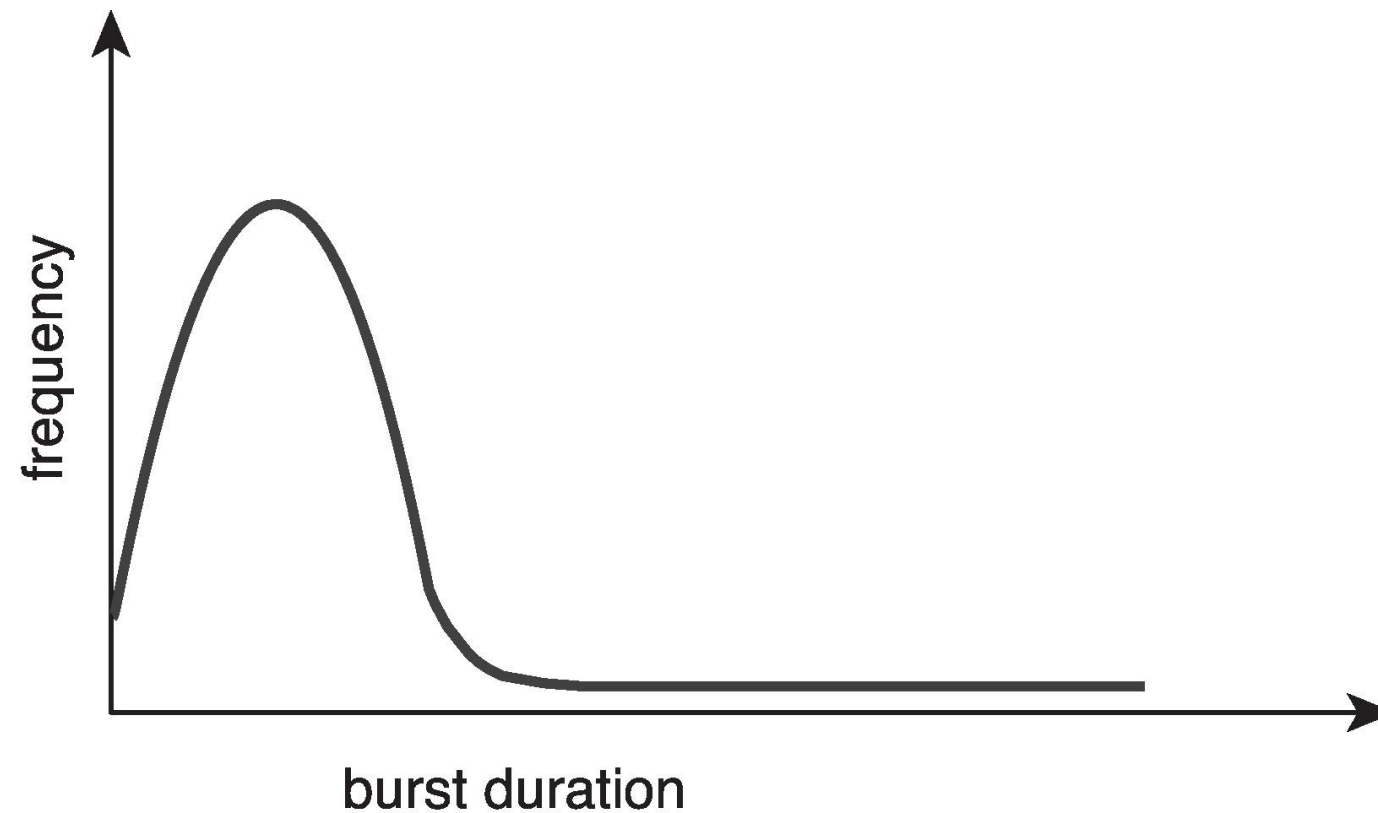




CPU Patlama Zamanları Histogramı

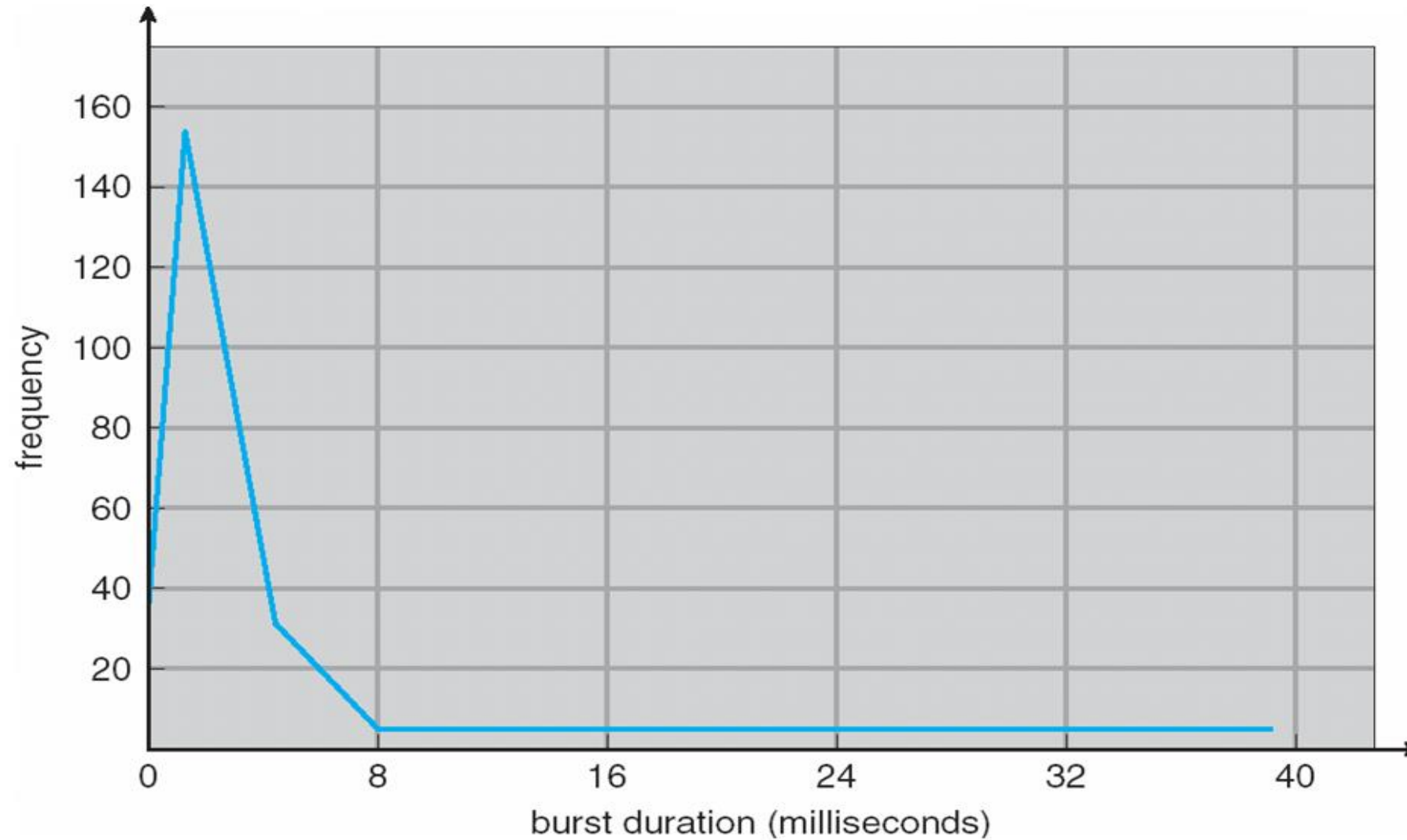
Çok sayıda kısa patlama

Az sayıda daha uzun patlama





CPU-patlama Zamanları





CPU İş Sıralayıcı

- Hazır kuyruğunda bekleyen prosesler arasından seçim yapar ve CPU çekirdeğini bir tanesine tahsis eder.
 - Kuyruk çeşitli şekillerde sıralanabilir
- CPU iş sıralama kararları aşağıdaki durumlarda verilir:
 1. Çalışıyor durumundan bekleme durumuna geçerken
 2. Çalışıyor durumundan hazır durumuna geçerken
 3. Bekleme durumunda hazır durumuna geçerken
 4. Proses sonlanınca
- 1 ve 4 durumları için iş sıralama bakımından başka seçenek yoktur. Yeni bir proses (eğer hazır kuyruğunda bulunuyorsa) çalışması için seçilir.
- 2 ve 3 nolu durumlarda bir seçenek vardır.





Kesintili (Preemptive) ve Kesintisiz(Nonpreemptive) İş Sıralama

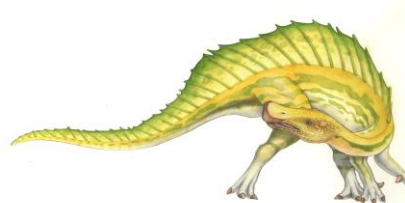
- 1 ve 4 durumları **kesintisizdir**
- Diğer tüm iş sıralama işlemleri **kesintilidir**
- Kesintisiz iş sıralamada, CPU bir prosese tahsis edildikten sonra, proses CPU'yu sonlanana veya bekleme durumuna geçene kadar elinde tutar.
- Windows, MacOS, Linux ve UNIX dahil olmak üzere neredeyse tüm modern işletim sistemleri kesintili iş sıralama algoritmaları kullanır.





Kesintili İş Sıralama ve Yarış Koşulları

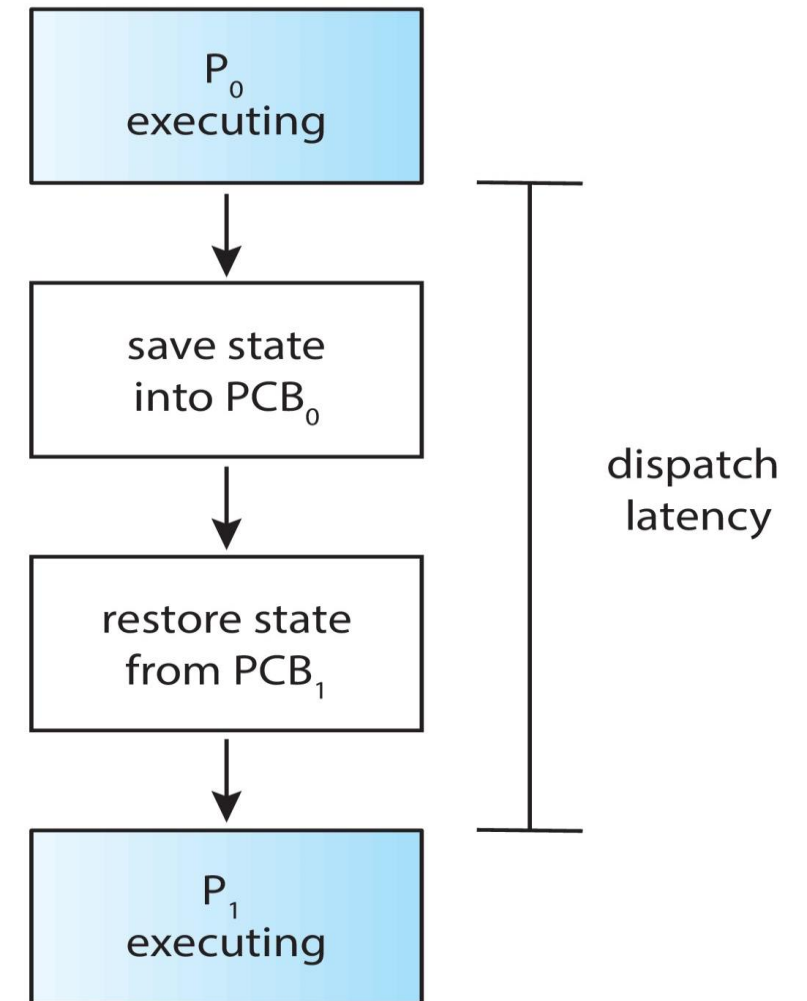
- Kesintili iş sıralama, verilerin çeşitli prosesler arasında paylaşılması durumunda yarış koşullarına neden olabilir.
- Verileri paylaşan iki proses düşünün. Bir proses verileri güncellerken, diğer bekleyen proses çalışabilsin diye çalışması yarıda kesilir. İkinci proses daha sonra tutarsız bir durumda olan verileri okumaya çalışır.





Görevlendirici - Dispatcher

- Görevlendirici modülü CPU'nun kontrolünü iş sıralayıcı tarafından seçilen prosese verir. Bu aşağıdaki işlemleri içerir:
 - Bağlam anahtarlama
 - Kullanıcı moduna değişim
 - Kullanıcı programını yeniden başlatmak için programdaki uygun bir konuma dallanma
- **Görevlendirme Gecikmesi**— bir prosesi sonlandırmak ve bir başkasını çalıştırmak için geçen süre





İş Sıralama Kriterleri

- **CPU kullanım oranı (utilization)**– CPU'yu olabildiğince meşgul tut
- **Çıkış (throughput)** – birim zamanda çalışmasını tamamlayan proses sayısı
- **Tamamlanma (turnaround) zamanı** – belirli bir prosesin çalışması 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





İş Sıralama Algoritması Optimizasyon Kriterleri

- Max CPU kullanımı
- Max çıkış
- Min tamamlanma zamanı
- Min bekleme zamanı
- Min cevap zamanı

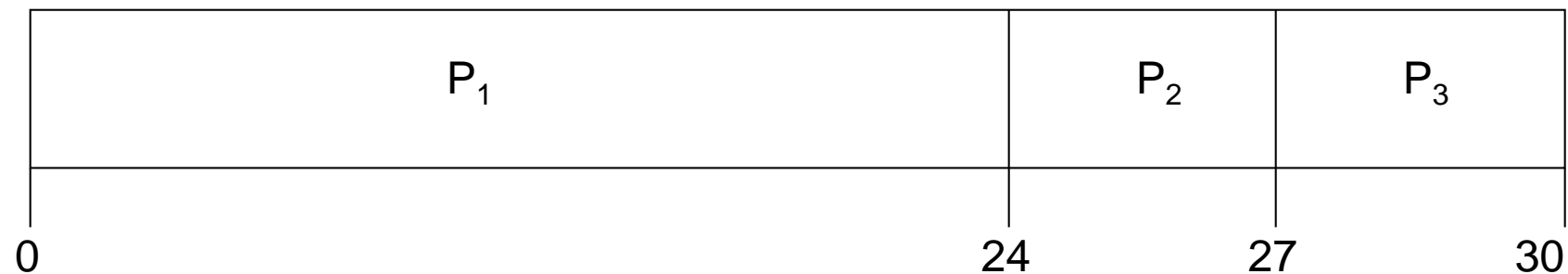




İlk Gelen İlk Çalışır Algoritması (First-Come, First-Served - FCFS)

<u>Proses</u>	<u>Patlama zamanı</u>
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 İş Sıralama (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
- **Konvoy etkisi** – uzun proseslerin kısa proseslerden önce gelmesi
 - 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 proses bir sonraki CPU patlama süresiyle ilişkilendir
 - 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
 - Veya hesaplanır
- Kesintili versiyonu **ek kısa kalan zaman önce** olarak adlandırılır

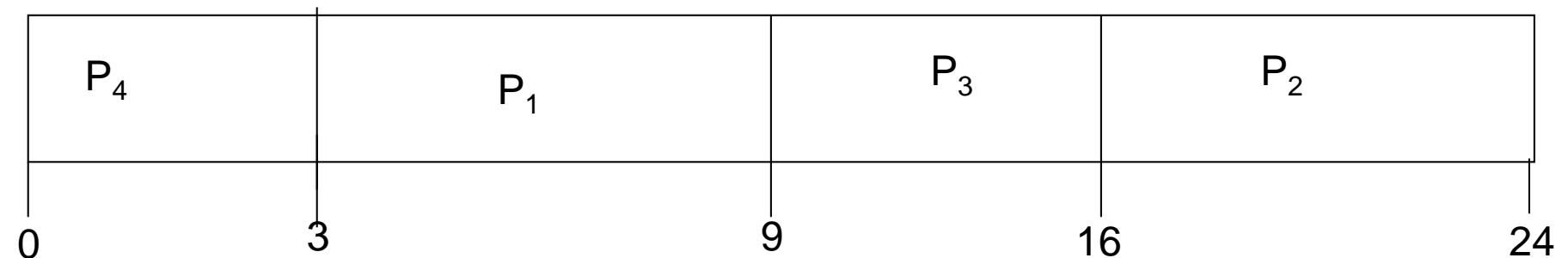




SJF örneği

<u>Proses</u>	<u>Patlama Zamanı</u>
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 Zamanının Belirlenmesi

- Süre sadece tahmin edilebilir – bir öncekine benzer olur
 - Bir sonraki CPU patlaması tahmin edilen en kısa süreli 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. $\alpha, 0 \leq \alpha \leq 1$
 4. Define :

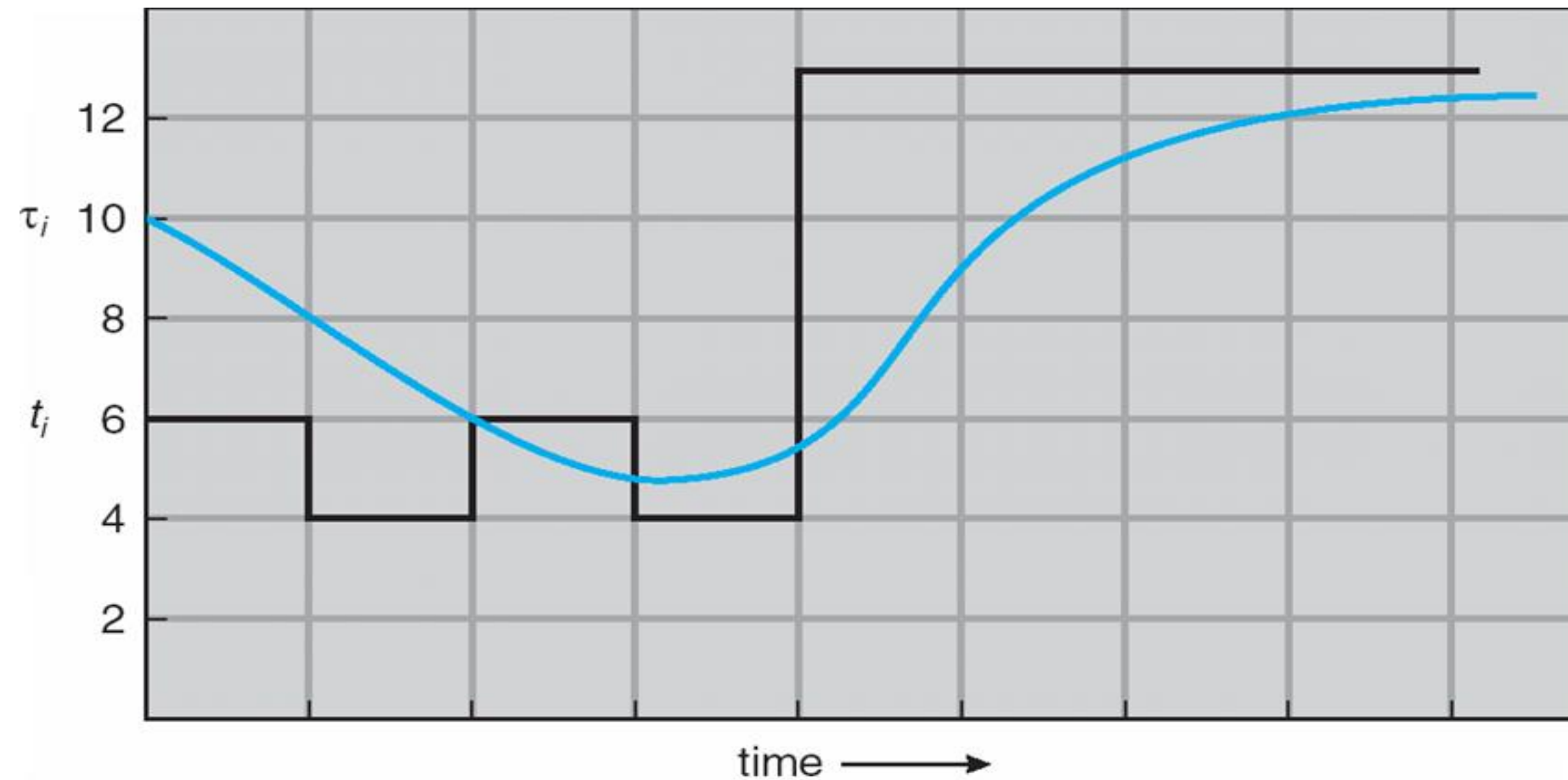
$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$$

- Genelde , α ½ seçilir





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

■ $\alpha = 0$

- $\tau_{n+1} = \tau_n$
- Son kayıtlar hesaba katılmaz

■ $\alpha = 1$

- $\tau_{n+1} = \alpha t_n$
- Sadece en son gerçek CPU patlaması hesaba katılır

■ Eğer formülü genişletirsek:

$$\begin{aligned}\tau_{n+1} = & \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots \\ & + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ & + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

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



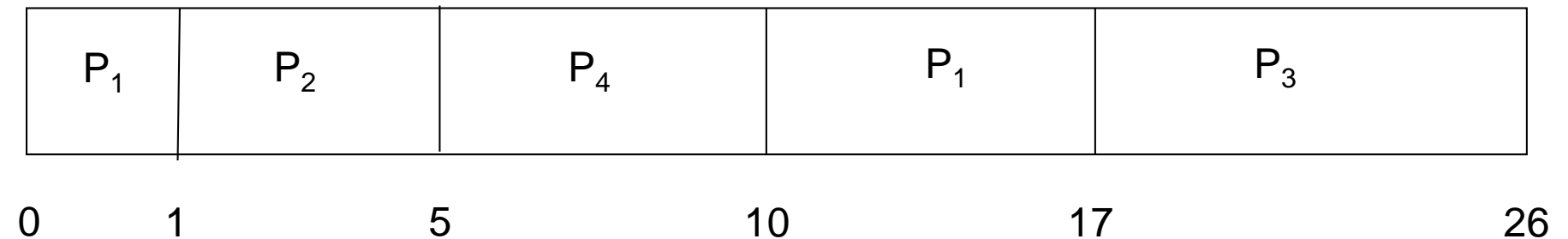


En Kısa Kalan Zaman Önce Örneği

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

<u>Proses</u>	<u>Varış Zamanı</u>	<u>Patlama Zamanı</u>
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Kesintili* SJF Gantt Diyagramı



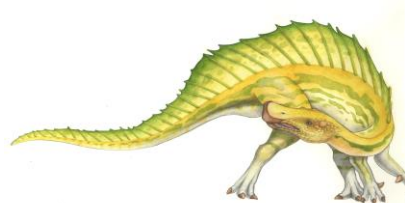
- Ortalama Bekleme zamanı= $[(10-1)+(1-1)+(17-2)+5-3]/4 = 26/4 = 6.5 \text{ msec}$





Öncelikli İş Sıralama

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

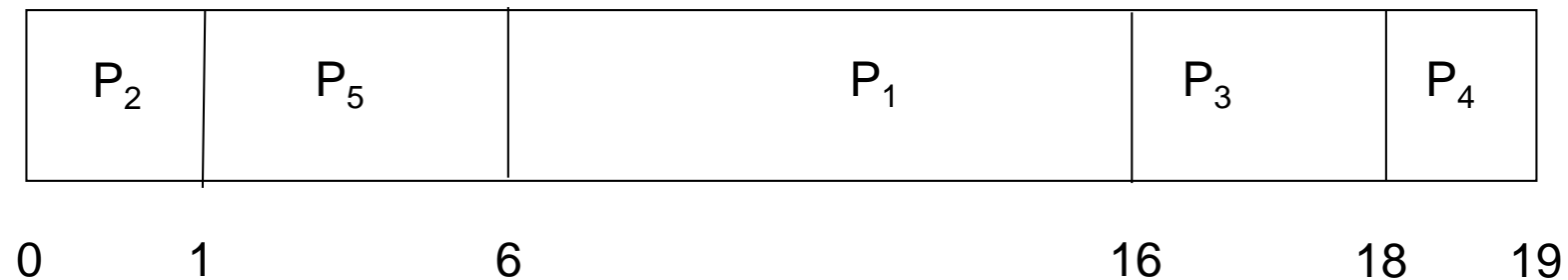




Öncelikli İş Sıralama Örneği

<u>Proses</u>	<u>Patlama Zamanı</u>	<u>Öncelik</u>
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

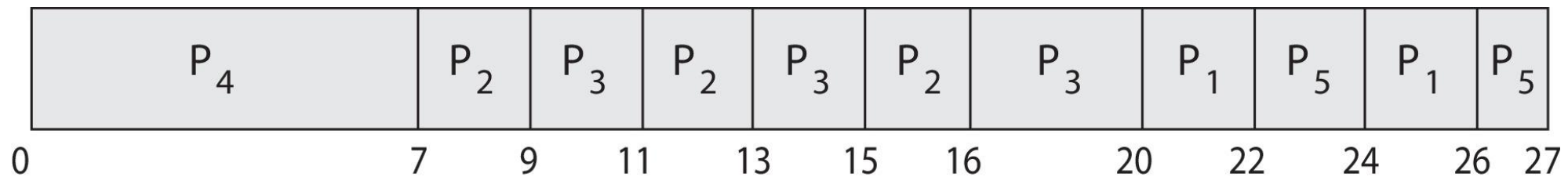




Çevrimsel Sıralı Öncelikli Sıralama

<u>Process</u>	<u>Patlama Zamanı</u>	<u>Öncelik</u>
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- En yüksek öncelikli prosesi çalıştır. Öncelik aynı ise çevrimsel sıralı
- Gantt Diyagramı, quantum = 2





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 $(n-1)q$ süresinden daha fazla beklemez
- Zamanlayıcı bir sonraki prosesi devreye almak için her bir kuantum süresi sonunda keser.
- Performans
 - q büyük \Rightarrow FIFO
 - q küçük $\Rightarrow q$ bağlam değişimine göre büyük olmalıdır aksi halde sistem kasılır

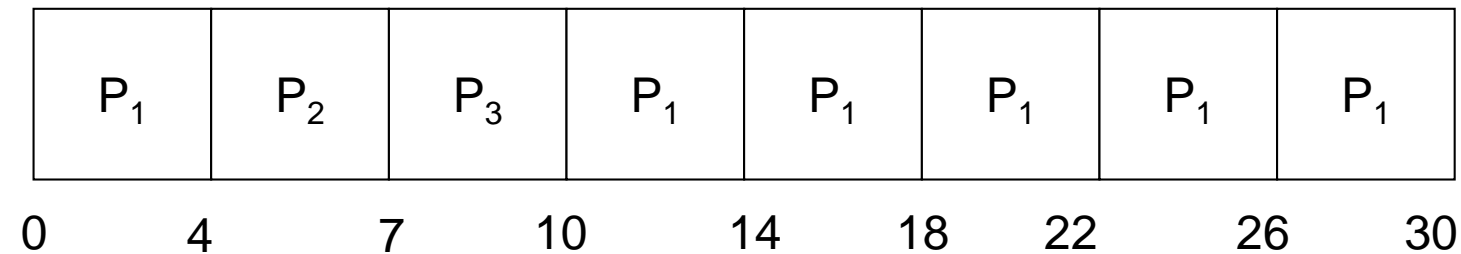




RR Örneği ($q = 4$)

<u>Process</u>	<u>Patlama Zamanı</u>
P_1	24
P_2	3
P_3	3

■ Gantt diyagramı:

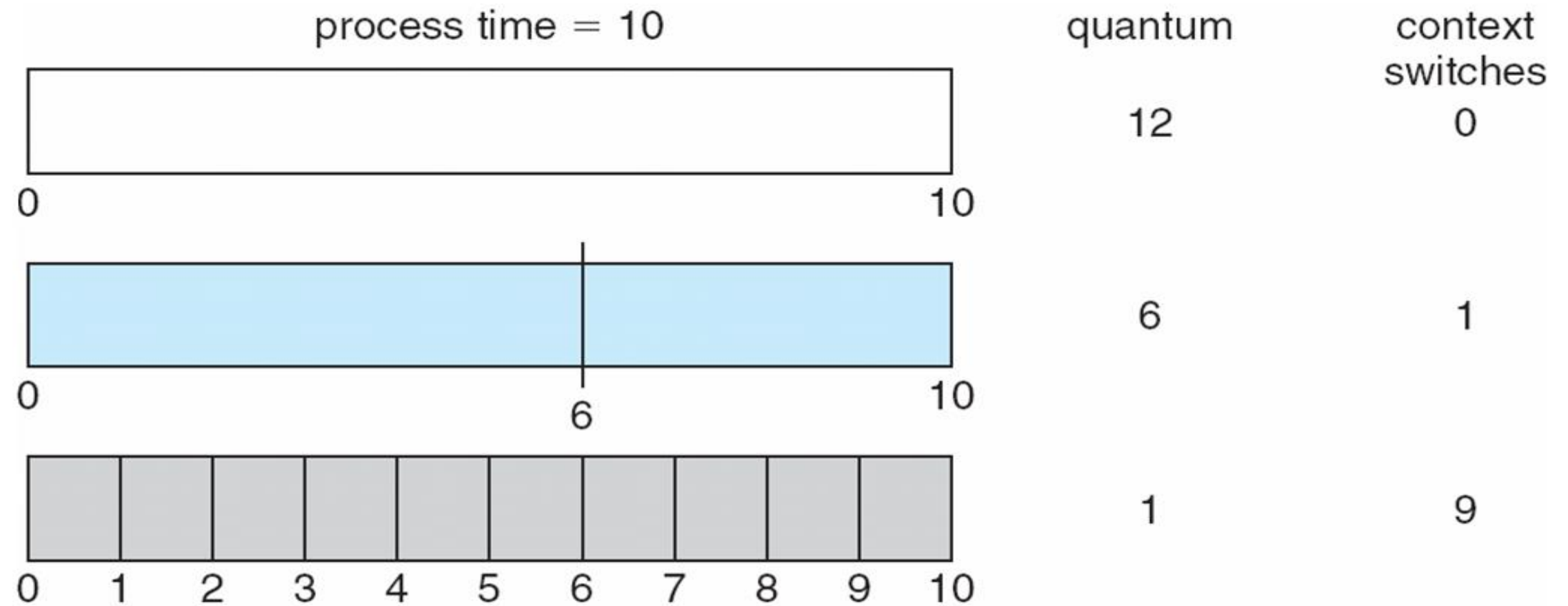


- Tipik olarak, SJF'den daha yüksek bir tamamlanma zamanı, ancak daha iyi bir cevap zamanına sahiptir.
- q bağlam değişimi zamanına göre büyük olmalıdır
- q genellikle 10ms ila 100ms,
- bağlam değişimi < 10 mikrosaniye



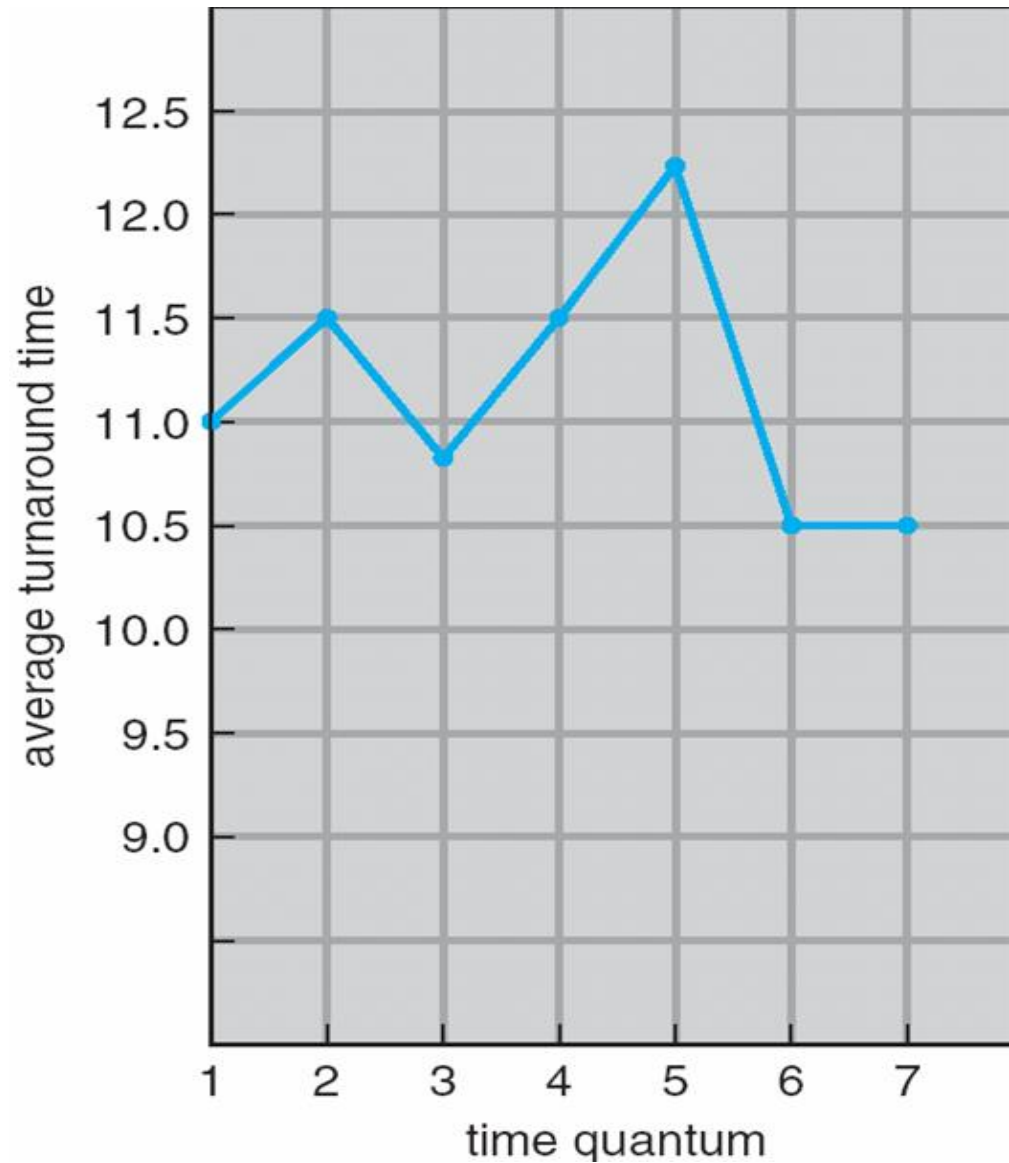


Kuantum and Bağlam Değişim Zamanı





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**





Çok Seviyeli Kuyruk

- Her öncelik için ayrı kuyruğun olduğu öncelikli sıralama
- En yüksek öncelikli kuyruktaki prosesler en önce sıralanır

priority = 0

T_0	T_1	T_2	T_3	T_4
-------	-------	-------	-------	-------

priority = 1

T_5	T_6	T_7
-------	-------	-------

priority = 2

T_8	T_9	T_{10}	T_{11}
-------	-------	----------	----------

●

●

●

priority = n

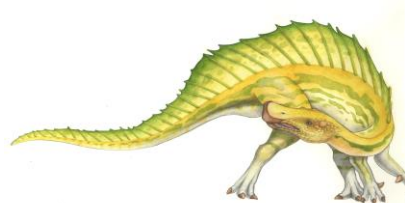
T_x	T_y	T_z
-------	-------	-------





Çok Seviyeli Kuyruk

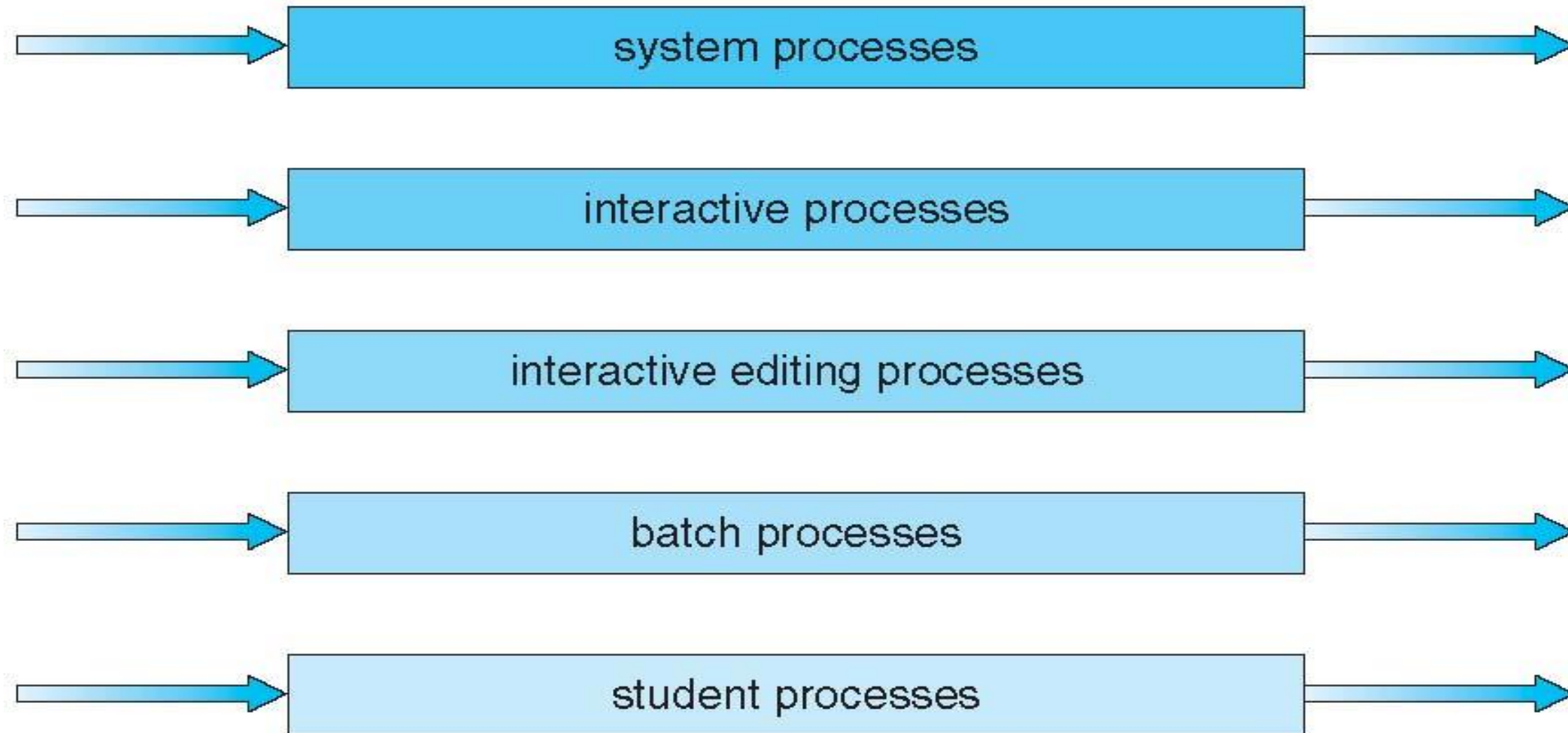
- Hazır kuyruğu ayrı kuyruklar halinde düzenlenir, ör:
 - Ön plan (interaktif)
 - Arkaplan (toplu iş - batch)
- Proses sürekli bir kuyruktadır
- Her kuyruk kendi iş sıralama algoritmasına sahiptir.
- ön plan – çevrimsel sıralı - RR
- arka plan – FCFS
- İş sıralama kuyruklar arasında yapılmalıdır:
 - 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

highest priority



lowest priority





Çok Seviyeli Geri Beslemeli Kuyruk

- Bir proses çeşitli kuyruklar arasında hareket edebilir;
- Bir tür yaşlandırma - aging uygulamasıdır
- 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





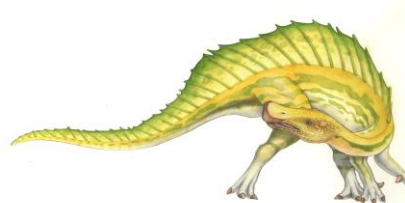
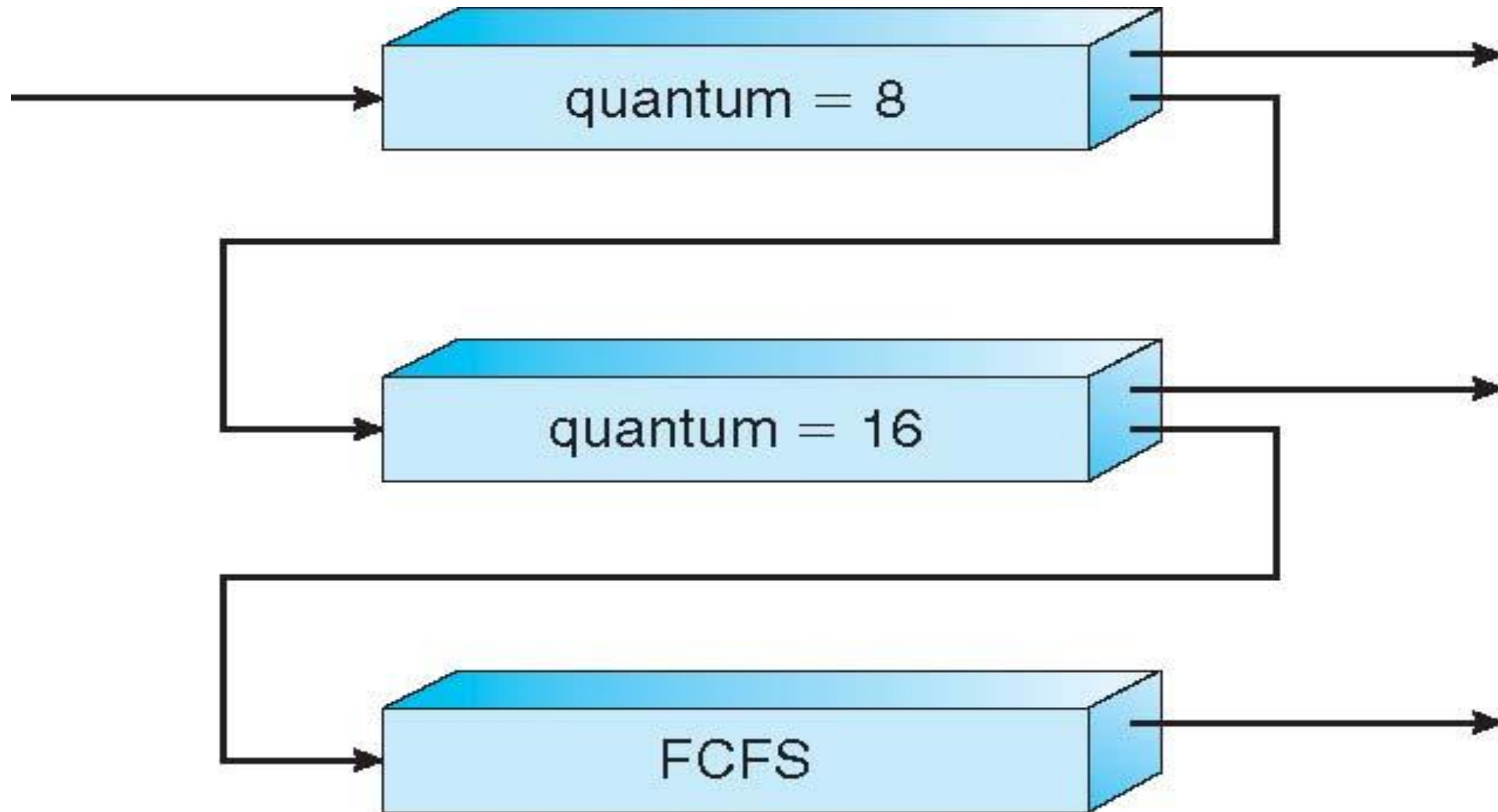
Çok Seviyeli Geri Beslemeli Kuyruk Örneği

- 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



End of Chapter 5a

