

# Programlama Dillerinin Prensipleri

## Hafta 11 - Eş Zamanlı Programlama

Dr. Öğr. Üyesi M. Fatih ADAK

# İçerik

- Tanım
- Öncelik Grafları
- Eşzamanlık Şartları
- Fork ve Join Yapıları
- Parbegin ve Parend Yapıları
- Programlama Dillerinde Eş Zamanlılığın Gerçekleştirimi
- İşlem Durumları
- İşlem Grafları
- Kritik Bölge
- Semaforlar

# Tanım

- Programlama dillerindeki eş zamanlılık kavramı ile bilgisayar donanımındaki paralel çalışma birbirinden bağımsız kavramlardır.
- Programlama dillerinde eş zamanlılık belli kavramlar kullanılarak gerçekleştirilebilir.
- Amaç programın hesaplama hızını arttırıp verimi yükseltmektir.
- Bir programda eş zamanlılık kullanılmamışsa varsayılan olarak bir thread o programı yürütecektir.

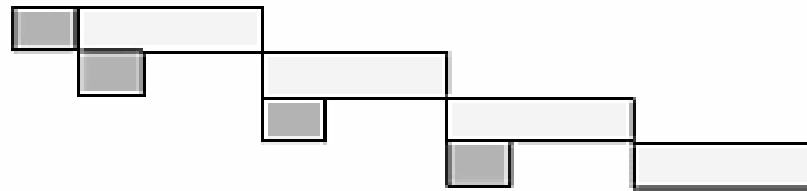


# Donanımsal Paralellik

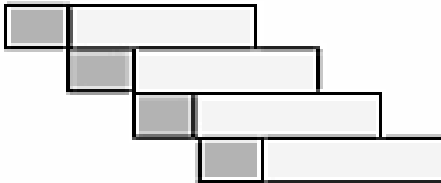
**Thread Yok**



**Çok Thread  
Tek İşlemci**

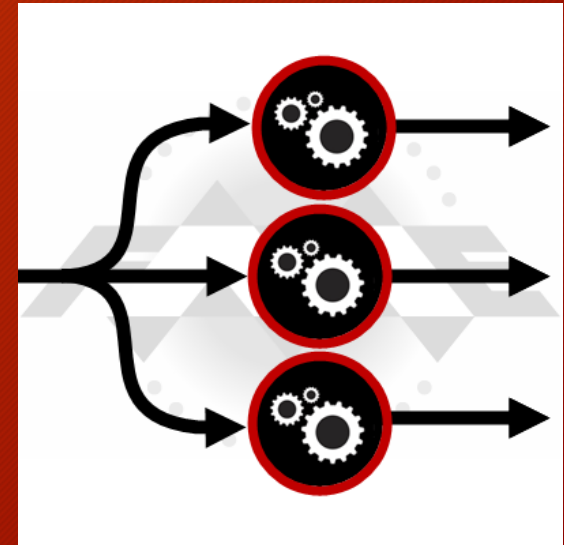


**Çok Thread  
Çok İşlemci**



# Eş Zamanlılık Türleri

- Program içerisindeki eş zamanlılık 4 farklı şekilde gerçekleşebilir.
  - **Makine komutu düzeyinde** : 2 veya daha fazla makine komutunun paralel çalıştırılması
  - **Program kod satırı düzeyinde**: 2 veya daha fazla program kod satırının paralel çalıştırılması
  - **Birim seviyesinde** : 2 veya daha fazla fonksiyonun paralel çalıştırılması
  - **Program seviyesinde** : 2 veya daha fazla programın paralel çalıştırılması

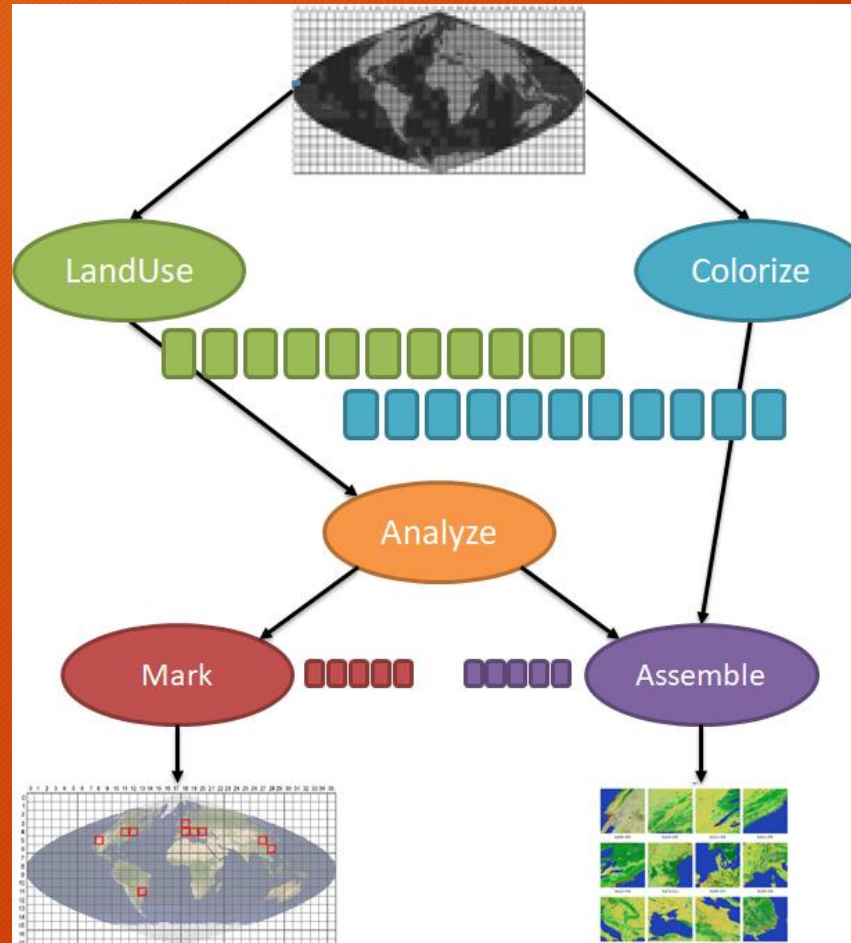


# Programlama Dillerinde Gerçekleştirimi

- Bazı dillerde kütüphaneler yardımıyla gerçekleştirilir.
  - Örnek: OpenMP - C/C++ ve Fortran
- Diğer bazı diller bunu kendi içerisinde sağlayabilir.
- Bu şekilde ilk destek PL/I programlama dili ile başlamıştır.
- Daha sonra bunu, Ada95, Java, C#, Python ve Ruby takip etmiştir.



# Python Paralleleştirme Örneği



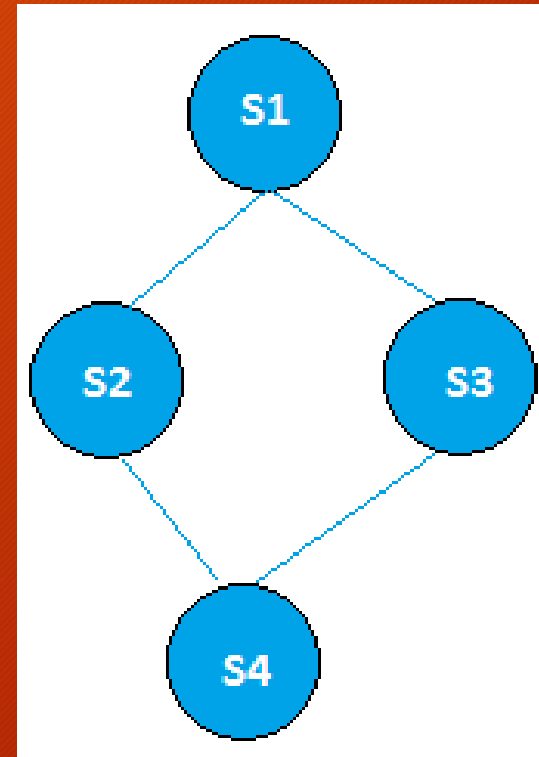
# Öncelik Grafları

- Bağımlı ve bağımsız değişkenlerin tespit edilip hangi işlerin eş zamanlı çalıştırılabileceği belirlenir.
- Bir öncelik grafi hangi işlemlerin eş zamanlı hangi işlemlerin seri olarak çalıştırılması gerekeceğini gösterir.
- Hangi işlemlerin hangi işlemleri bekleyeceği görülebilir.

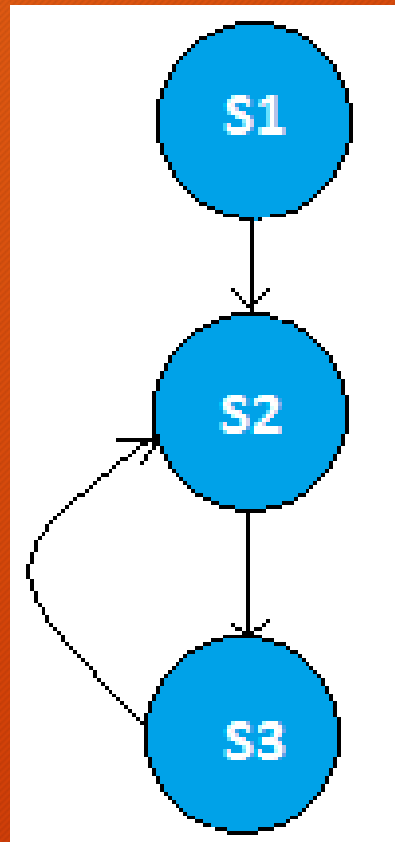


# Öncelik Grafları

- S2 ve S3 çalıştırılabilmesi için S1 ifadesi işlemini bitirmelidir.
- S2 ve S3 eş zamanlı çalıştırılabilir.
- S4 çalışabilmesi için hem S2 hem de S3 işlemini bitirmelidir.



# Hatalı Öncelik Grafi



# Eşzamanlık Şartları

$R(S_i) = \{a_1, a_2, \dots, a_n\}$ :  $S_i$  için “oku” kümesi.

$W(S_i) = \{b_1, b_2, \dots, b_n\}$ :  $S_i$  için “yaz” kümesi.

$S_1$  ve  $S_2$  eşzamanlı çalışması için aşağıdaki 3 şart gerçekleşmelidir.

1.  $R(S_1) \cap W(S_2) = \{\}$
2.  $W(S_1) \cap R(S_2) = \{\}$
3.  $W(S_1) \cap W(S_2) = \{\}$



# Eşzamanlık Şartları

S1  $a := x + y;$        $R(S1) = \{x, y\}$   
S2  $b := z + 1;$        $R(S2) = \{z\}$   
S3  $c := a - b;$        $R(S3) = \{a, b\}$   
                          $W(S1) = \{a\}$   
                          $W(S2) = \{b\}$   
                          $W(S3) = \{c\}$

S1 ve S2 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S1) \cap W(S2) = \{x, y\} \cap \{b\} = \{\}$

Koşul 2.  $W(S1) \cap R(S2) = \{a\} \cap \{z\} = \{\}$

Koşul 3.  $W(S1) \cap W(S2) = \{a\} \cap \{b\} = \{\}$



S1 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S1) \cap W(S3) = \{x, y\} \cap \{c\} = \{\}$

Koşul 2.  $W(S1) \cap R(S3) = \{a\} \cap \{a, b\} = \{a\}$

Koşul 3.  $W(S1) \cap W(S3) = \{a\} \cap \{c\} = \{\}$



S2 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?

Koşul 1.  $R(S2) \cap W(S3) = \{z\} \cap \{c\} = \{\}$

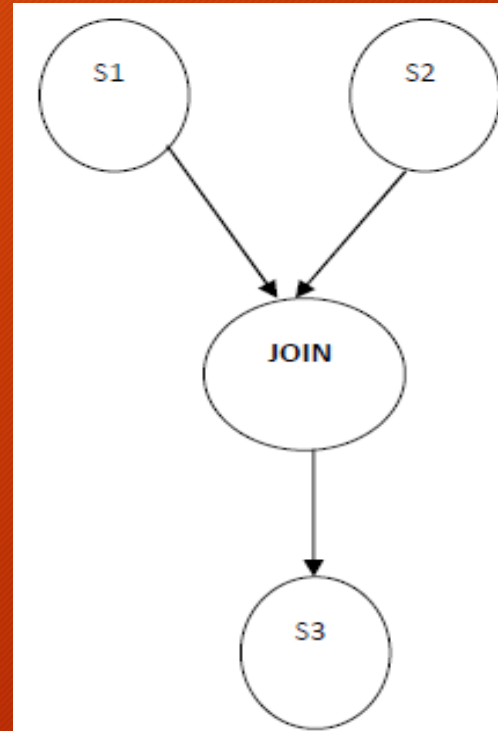
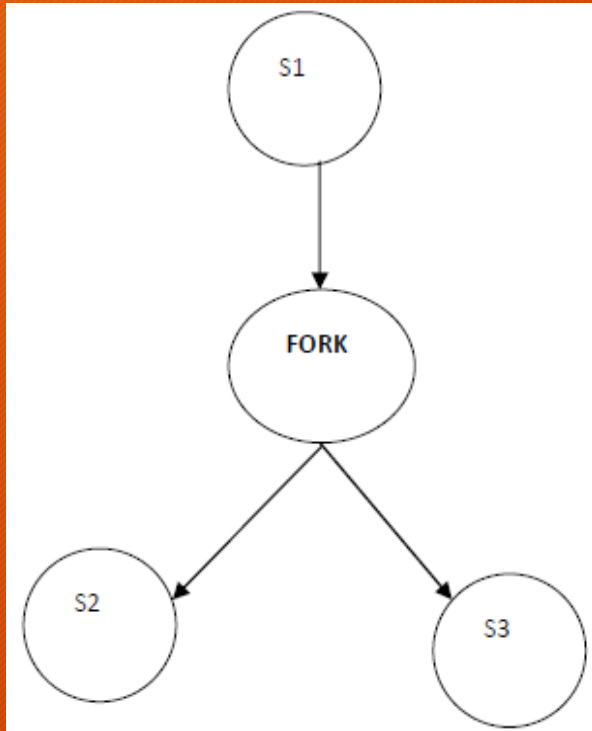
Koşul 2.  $W(S2) \cap R(S3) = \{b\} \cap \{a, b\} = \{b\}$

Koşul 3.  $W(S2) \cap W(S3) = \{b\} \cap \{c\} = \{\}$



# Fork ve Join Yapıları

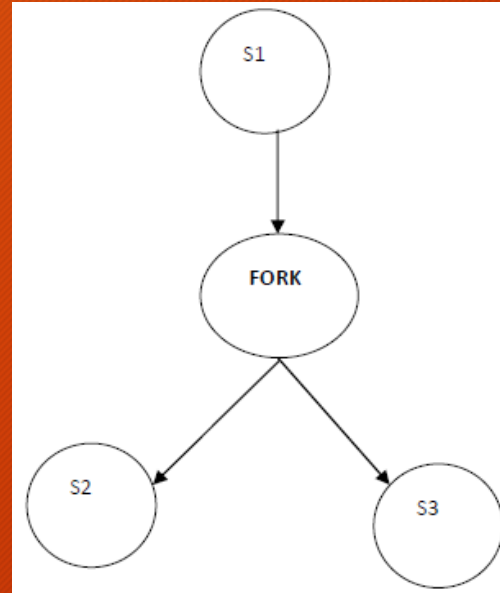
- Eş zamanlılığı tanımlayan ilk programlama dili notasyonlarından biridir.



# Fork Yapısı

- İki eş zamanlı eşleme üretir.

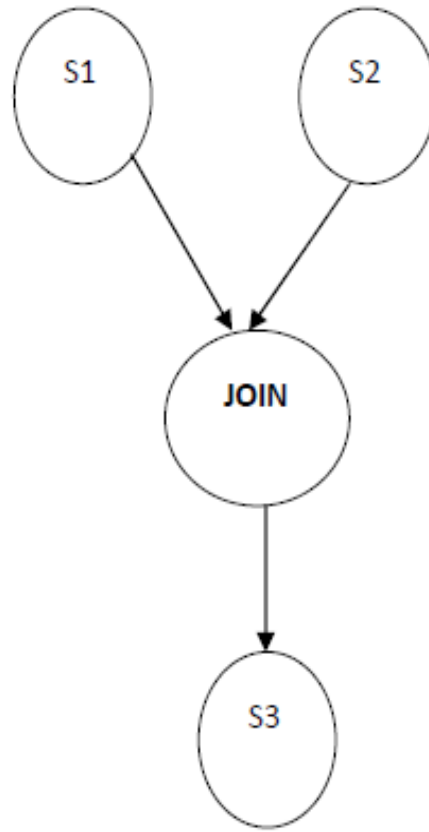
```
S1;  
FORK L  
S2;  
...  
...  
L:S3;
```





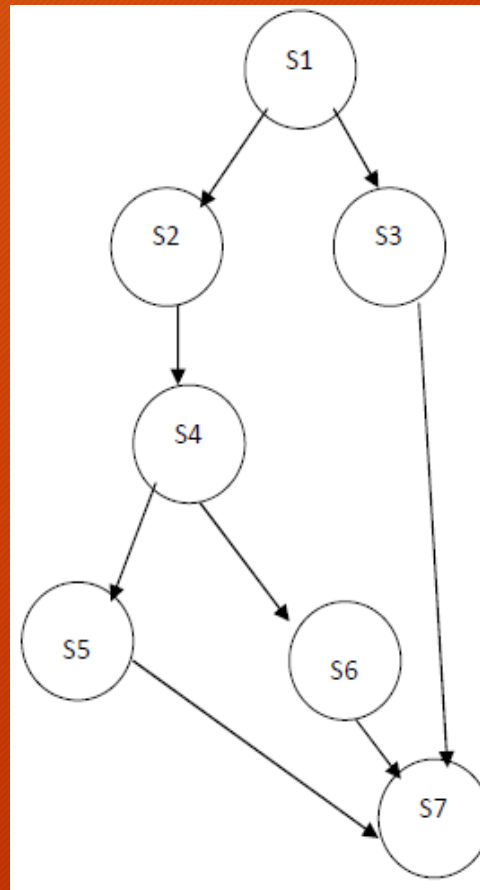
# Join Yapısı

```
Count:=2;  
FORK L1;  
...  
...  
S1;  
Go to L2;  
  
L1:S2;  
L2:JOIN count;  
S3;
```



# Fork Join Örneği

```
S1  
Count:=3;  
FORK L1;  
S2;  
S4;  
FORK L2;  
S5;  
Goto L3;  
L2:S6;  
Goto L3;  
L1:S3;  
L3: JOIN count;  
S7;
```



# Parbegin ve Parend Yapıları

Parbegin

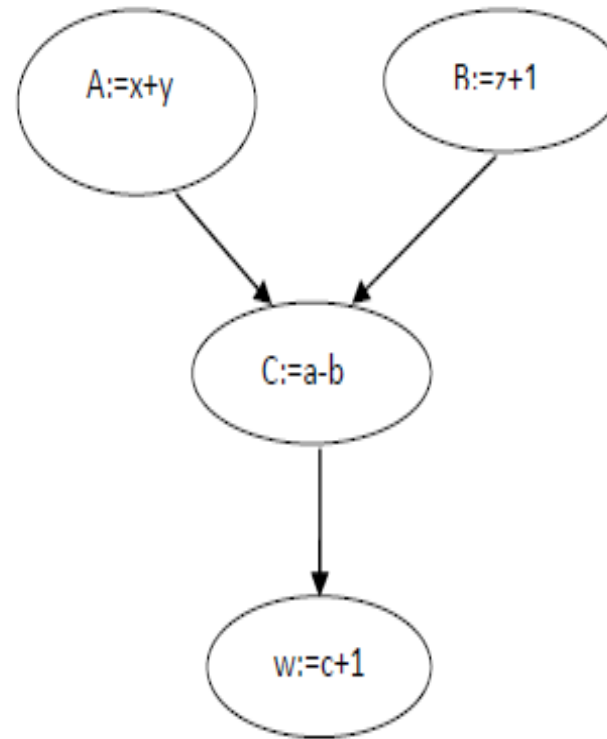
a:= x + y;

b:= z + 1;

parend;

c:= a - b;

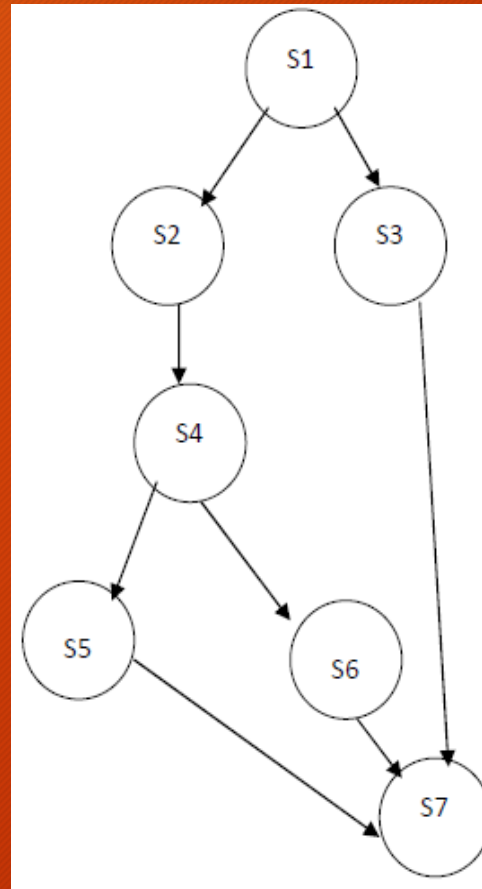
w:=c + 1;



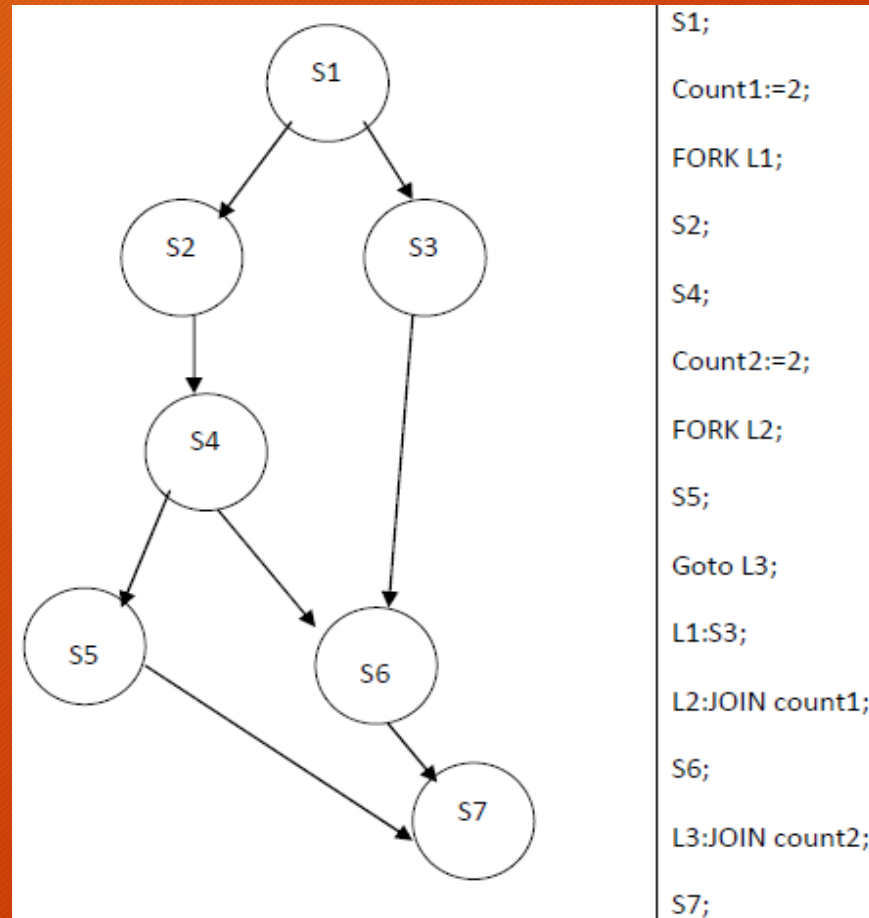


# Parbegin ve Parend Yapıları

```
S1;  
Parbegin  
S3;  
Begin  
S2;  
S4;  
Parbegin  
S5;  
S6;  
Parend;  
End;  
Parend;  
S7;
```



# Fork Join ile Yapılıp Parbegin Parend ile Yapılamayan Örnek



# Fonksiyonel Dillerde Eş Zamanlılık

- Multi-LISP

- 1985 Yılında tanıtılan bu dil program parçalarının eş zamanlı çalışmasına izin veriyordu.
- pcall yapısı ile bu gerçekleştiriliyordu.

(fonk x y z)



Eş Zamanlılık olmayan normal bir fonksiyon çağırımı

(pcall fonk x y z)



Eş Zamanlılık içeren fonksiyon çağırımı

- pcall ile çağırılması x, y ve z parametrelerinin eş zamanlı çalışmasını sağlayacaktır. x y ve z parametreleri yine bir fonksiyon olabilir bu durumda bu fonksiyonlar eş zamanlı çalıştırılır.



# Thread Yield Metodu

- yield metodu geçici olarak diğer threadlere zaman verir.
- `Thread.yield();` şeklinde kullanılır.
- O satıra gelen her thread bu komutu uygulayacaktır.

# Thread Sleep Metodu

- Belirtilen süre boyunca thread uykuya geçer.
- Bu tepki süresinin uzun olduğu bazı durumlarda zorunlu olarak kullanılabilmektedir.
- `Thread.sleep(1);` 1 milisaniye uykuya geçirir.
- Java dilinde sleep metodu yakalanması zorunlu bir hata fırlatma durumu olduğu için try catch bloklarında kullanımı zorunludur.

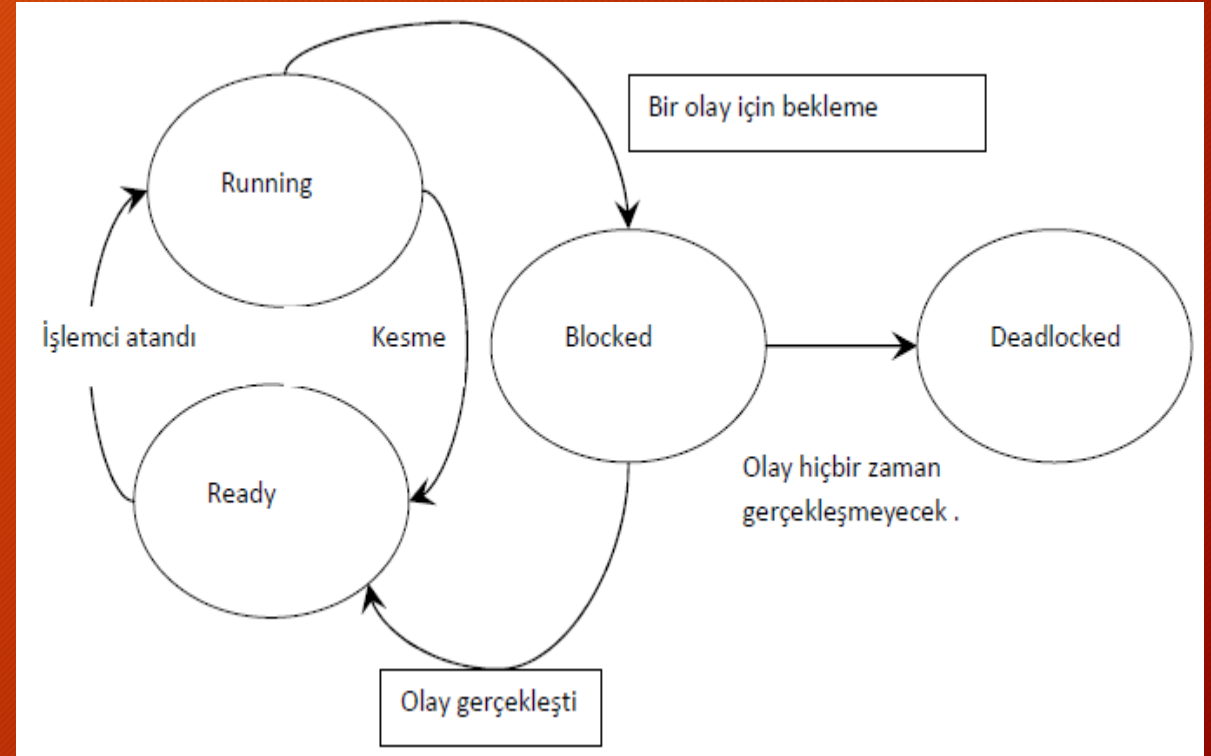
# Sleep ve Yield Metotları

- Bu metotların kullanımı için eş zamanlılık şart değildir.
- Programa atanan varsayılan thread bu komutlara denk geldiğinde işleyişi uygular.



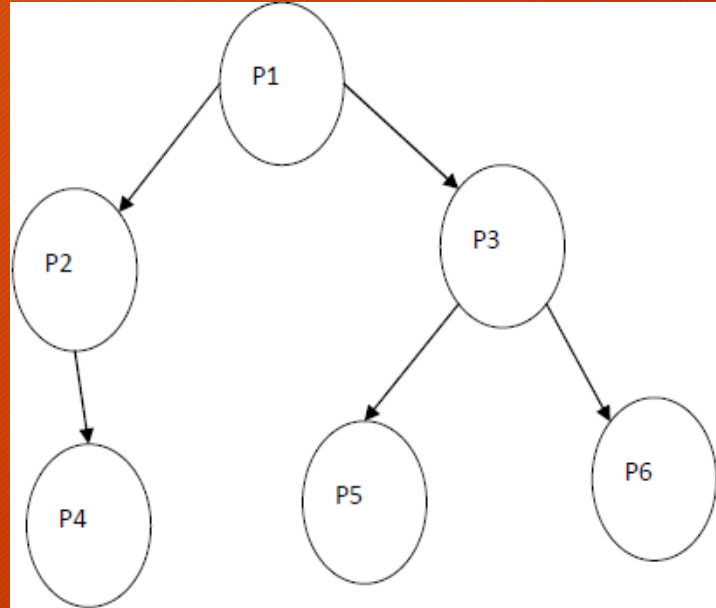
# İşlem Durumları

- Running: Komutlar işletiliyor.
- Blocked: Sistem bazı durumlar için bekletiliyor.
- Ready: İşlem bir işlemciye atanmak için hazır durumda bekletiliyor.
- Deadlock: İşlem hiç bir zaman gerçekleşmeyecek olayları bekliyor.



# İşlem Grafları

- Bir işlem grafi düğümleri işlemlere karşılık gelen yönlendirilmiş köklü bir graftır.
- $P_i$  düğümünden  $P_j$  düğümüne gelen ok işareti  $P_i$ 'nin  $P_j$ 'yi oluşturduğunu ifade eder.

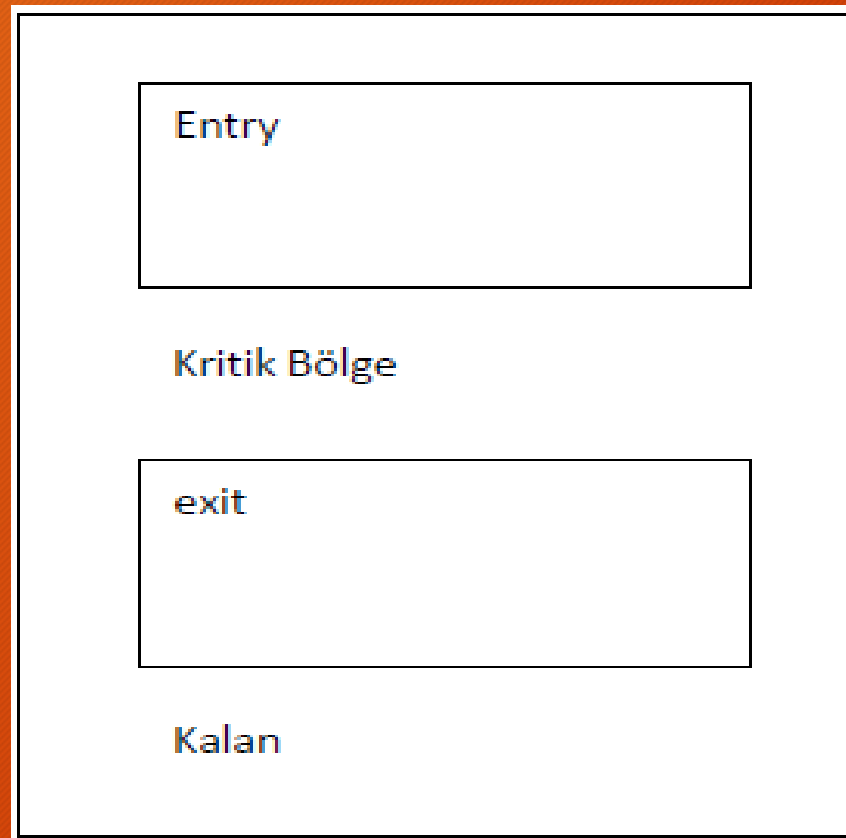


# Kritik Bölge

- Birlikte çalışan  $n$  adet işlemden  $\{P1, P2, \dots, Pn\}$  oluşan bir sistem olduğu düşünülürse.
- Her bir işlem ortak değişkenleri okuyan bir tabloyu güncelleyen, bir dosyayı yazan vb. işlemleri içerebilir.
- Bu bölümlere kritik bölge ismi verilir.
- Bu bölgelere aynı anda sadece bir thread girmelidir.



# Kritik Bölgenin Yapısı



# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 1

Repeat

While turn  $\neq$  i do skip;

Kritik Bölge

Turn=j;

Kalan

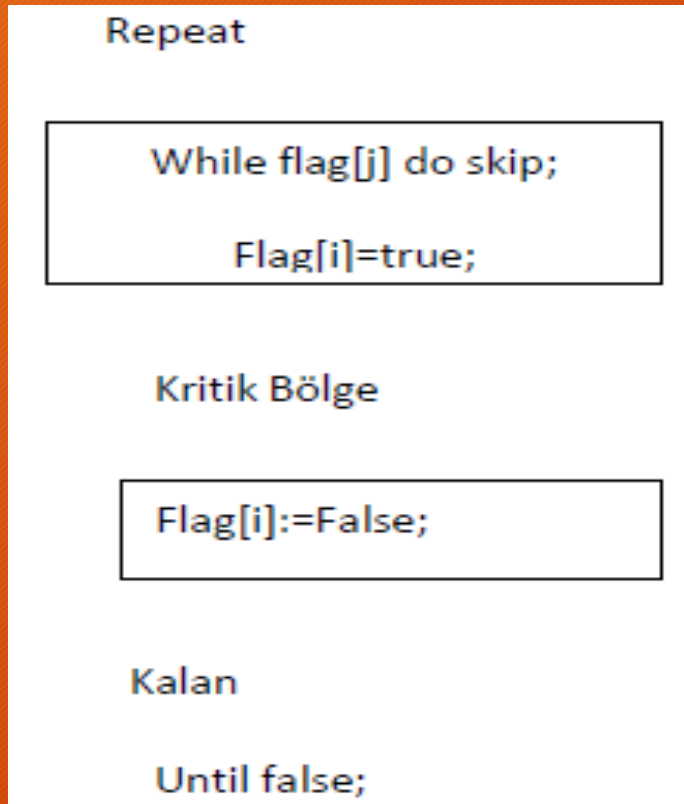
Until false;

Analiz:

- Algoritma hangi işlemin kritik bölgesine girmesine izin verdiğini hatırlar
- İşlemin hangi aşamada olduğunu hatırlayamaz.

# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 2



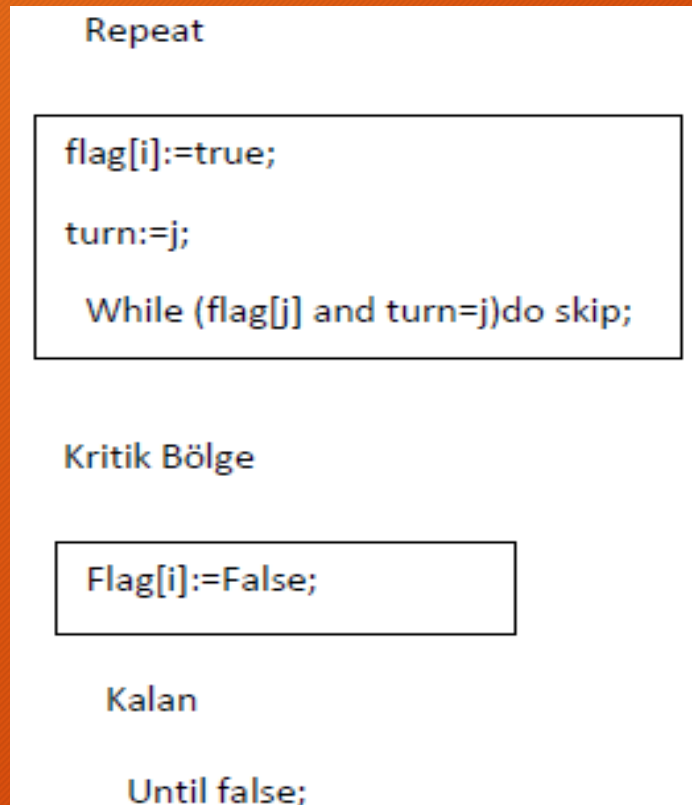
### Analiz:

- Algoritma hangi işlemin kritik bölgesine girmesine izin verdiğini hatırlar
- İşlemin hangi aşamada olduğunu hatırlar.
- Fakat bir dizi kontrol edildiği için aynı anda birden fazla thread'in kritik bölgeye girme ihtimali vardır.



# Kritik Bölge Gerçekleştirimi için Yaklaşımlar

## Örnek Algoritma 3



### Analiz:

- Algoritma hangi işlemin kritik bölgesine girmesine izin verdiğini hatırlar
- İşlemin hangi aşamada olduğunu hatırlar.
- Birden fazla thread'in kritik bölgeye girmesine ihtimal yoktur.

# Semaforlar

- Karşılıklı hariç bırakma (mutual exclusuion) problemi için yapılan çözümleri daha kompleks problemler için genelleştirmek kolay değildir.
- Bu zorluğun üstesinden gelebilmek için semaforlar olarak adlandırılan bir senkronizasyon aracı kullanılabilir.

# Semaforlar

Repeat

P(Mutex)

Kritik Bölge

V(Mutex)

Kalan

Until false;

S1;

V(synch);

P(synch);

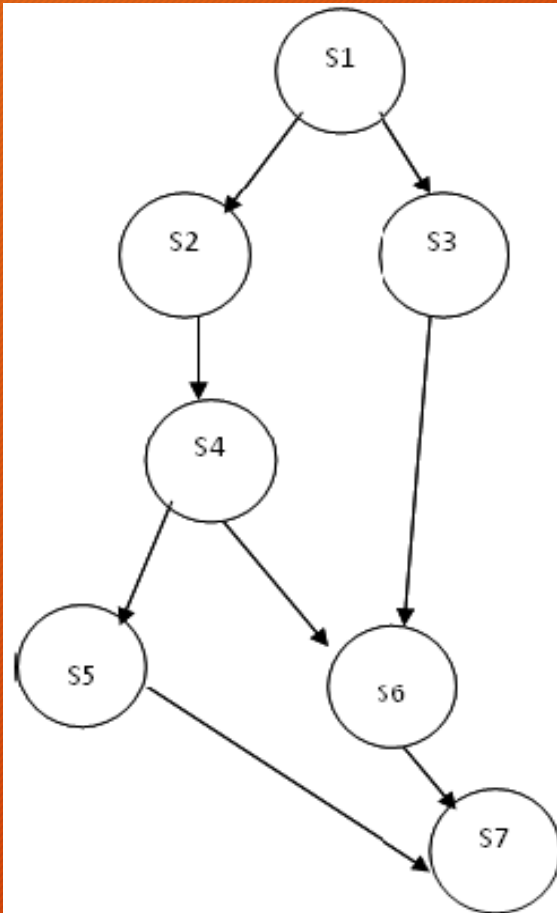
S2;

P1 işlemine konulur.

P2 işlemine konulur.



# Semaforların Örnek Üzerinde Kullanımı



```
Var a, b, c, d, e, f, g: semaphore;  
  
Begin  
  Parbegin  
    Begin S1; V(a); V(b) end;  
    Begin P(a); S2; S4; V(c); V(d); end;  
    Begin P(b); S3; V(e); end;  
    Begin P( c); S5; V(f); end;  
    Begin P(d); P(e); S6; V(g); end;  
    Begin P(f); P(g); S7; end  
  Parend;  
End;
```

# Kaynaklar

- Yumusak N., Adak M.F. *Programlama Dillerinin Prensipleri*. 1. Baskı, Seçkin Yayıncılık, 2018
- Sebesta, Robert W. *Concepts of programming languages*. 11 ed. Pearson Education Limited, 2016.
- Sethi, Ravi. *Programming languages: concepts and constructs*. Addison Wesley Longman Publishing Co., Inc., 1996.
- Watt, David A. *Programming language design concepts*. John Wiley & Sons, 2004.
- Malik, D. S., and Robert Burton. *Java programming: guided learning with early objects*. Course Technology Press, 2008.
- Waite, Mitchell, Stephen Prata, and Donald Martin. *C primer plus*. Sams, 1987.
- Hennessey, Wade L. *Common Lisp*. McGraw-Hill, Inc., 1989.
- Liang, Y. Daniel. *Introduction to Java programming: brief version*. pearson prentice hall, 2009.
- Yumusak N., Adak M.F. *C/C++ ile Veri Yapıları ve Çözümlü Uygulamalar*. 2. Baskı, Seçkin Yayıncılık, 2016