

# 8. Deadlocks

ECE30021/ITP30002 Operating Systems

# Agenda

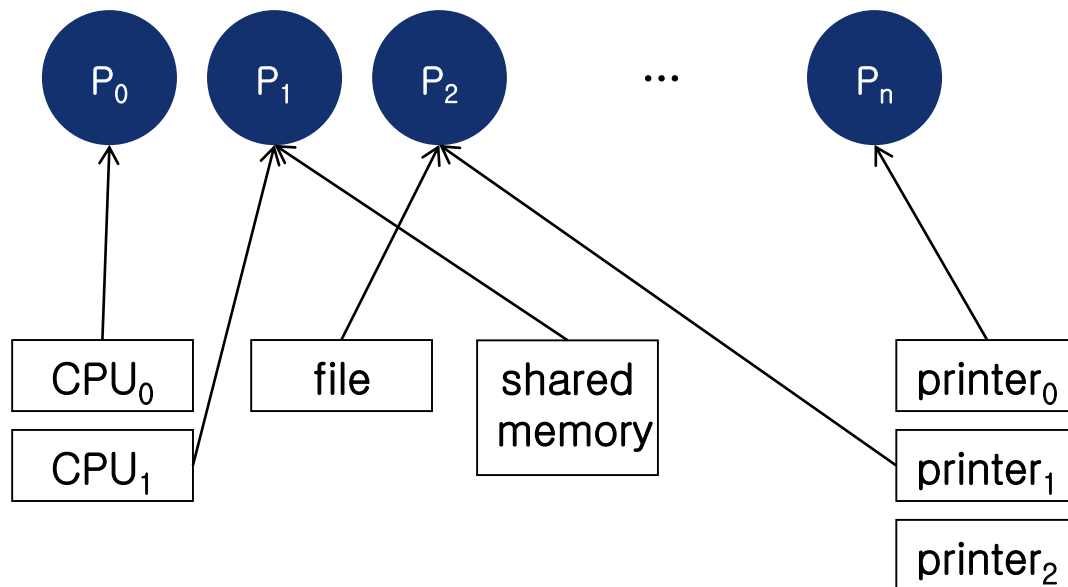
---



- Introduction
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection
- Recovery from deadlock

# System Model

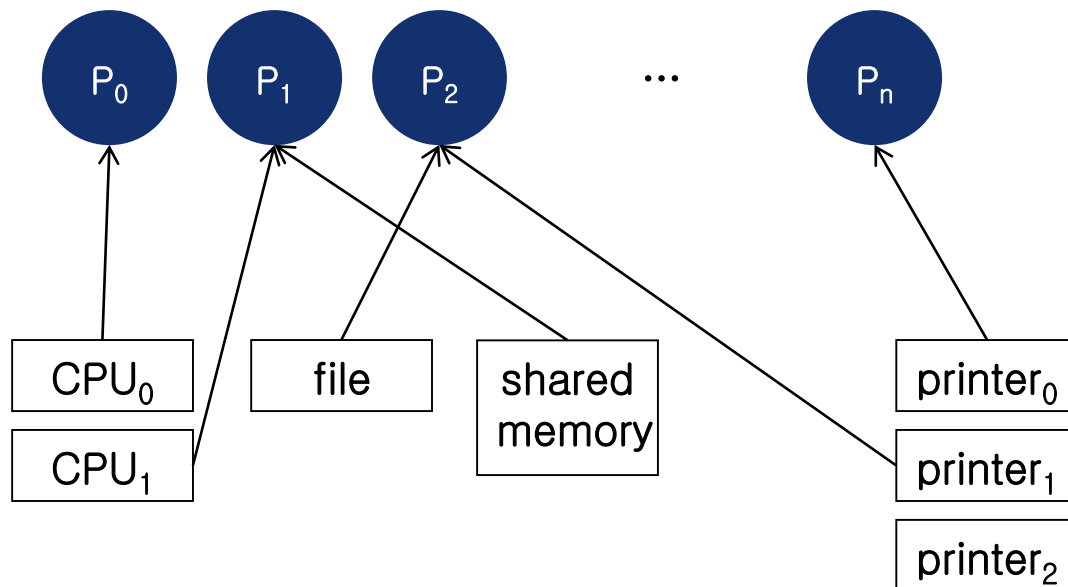
- A system consists of
  - A finite number of resources
    - Resources are partitioned into several **types**  
Ex) two identical printers
  - A number of competing processes



# System Model

## ■ Steps to utilize a resource

1. Request
  - Process should wait until the request is granted
2. Use
3. Release



# Deadlock State

- A set of processes is in deadlock state
  - Each process is waiting for an event that can be caused only by another process in the set.
  - Mainly concerned with resource acquisition and release



# Deadlock Characterization

- Necessary conditions for deadlock
  1. Mutual exclusion
  2. Hold and wait
  3. No preemption
  4. Circular wait



# Resource-Allocation Graph

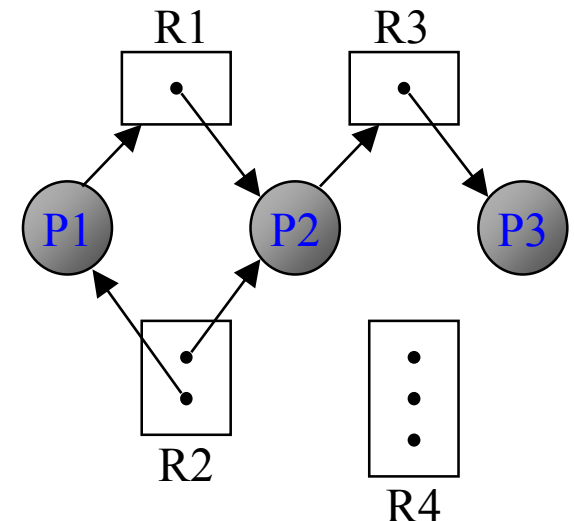
## ■ $G(V, E)$

### ■ $V$ : two types of vertices

- $P = \{P_1, P_2, \dots, P_n\}$ : a set of processes
- $R = \{R_1, R_2, \dots, R_m\}$ : a set of resource types

### ■ $E$ : two types of directed edges

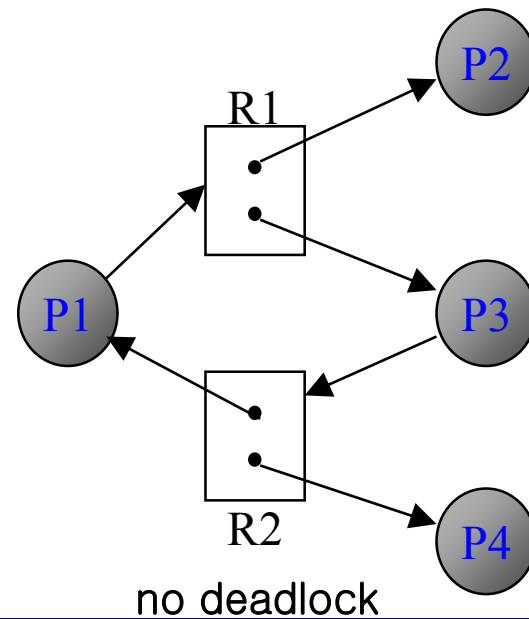
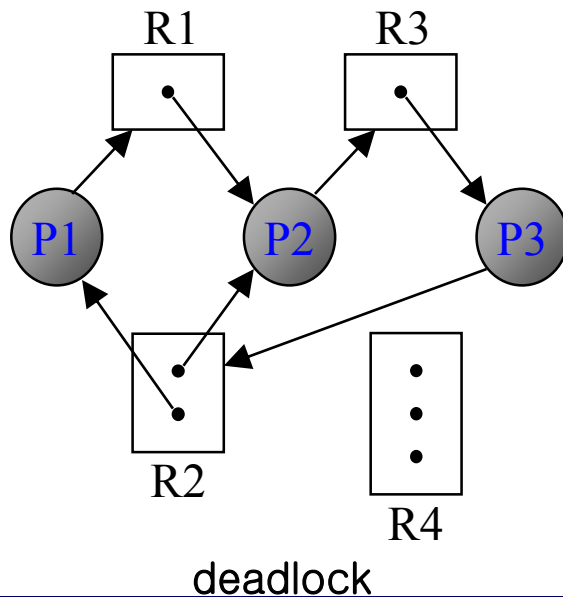
- $P_i \rightarrow R_j$ : **request edge**
- $R_j \rightarrow P_i$ : **assignment edge**
- If a request edge is fulfilled, it becomes an assignment edge



# Resource-Allocation Graph

## ■ Cycle vs. deadlock state

- No cycle  $\rightarrow$  no deadlock
- Cycle  $\rightarrow$  deadlock (?)
  - Only one instance per resource type – deadlock occurred
  - Multiple instances per resource type – possibility of deadlock





# Methods for Handling Deadlock



- Three methods to handle deadlock
  - Prevent or avoid deadlock
  - Detect deadlock and recover
  - Ignore
    - Most common case
    - Deadlock is very infrequent (once a year)

# Methods for Handling Deadlock



## ■ Deadlock prevention

- A set of method for ensuring at least one of necessary conditions cannot hold
- Constraint on request for resources

## ■ Deadlock avoidance

- Keep the system in **safe state** in which deadlock cannot occur (**using additional information**)
  - Resources currently available or allocated to each process
  - **Additional information about future requests and release of each process.**

# Agenda

---



- Introduction
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection
- Recovery from deadlock

# Deadlock Prevention

## ■ Deadlock

$P_0$	$P_1$
<i>acquire(S);</i>	<i>acquire(Q);</i>
<i>acquire(Q);</i>	<i>acquire(S);</i>
$\vdots$	$\vdots$
<i>release(S);</i>	<i>release(Q);</i>
<i>release(Q);</i>	<i>release(S);</i>

# Deadlock Prevention



- **Mutual exclusion**
  - Possible only for sharable resources
    - Ex) read-only file
  - ➔ Many resources are intrinsically non-sharable
  
- **Hold and wait: guarantee whenever a process requests a resource, it does not hold other resources**
  - Allocate all required resources before it begins execution
  - Or allow a process to request resources only when it has none
  - ➔ Inefficient utilization of resource
  - ➔ Can cause starvation

# Deadlock Prevention



## ■ No Preemption

- If a process cannot allocate a resource immediately, it should release all resources it hold.
- The process restarts when it regains its old resources and the new resource it requested.

## ■ Circular wait

- Impose total ordering of all resource types
- All resources should be requested in that order.

Ex)  $F(\text{tape drive}) = 1$

$F(\text{disk drive}) = 5$

$F(\text{printer}) = 12$

# Deadlock Prevention

---



- Problems of deadlock prevention
  - Low device utilization
  - Reduce system throughput

# Deadlock Avoidance



- Require **additional information** about how resources are to be requested.
  - Ex) Each process declares maximum # of resources of each type it may request.
- Deadlock avoidance algorithm dynamically examines **resource-allocation state** to ensure a circular-wait condition can never exist.
- Resource-allocation state is defined by
  - # of available resources
  - # of allocated resources
  - **Maximum demand of processes**



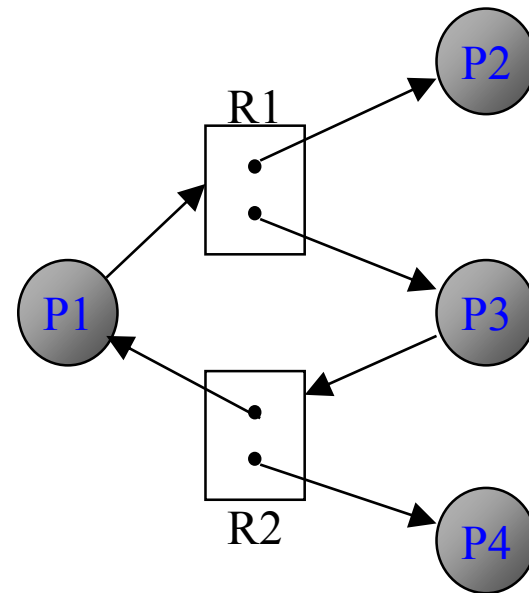
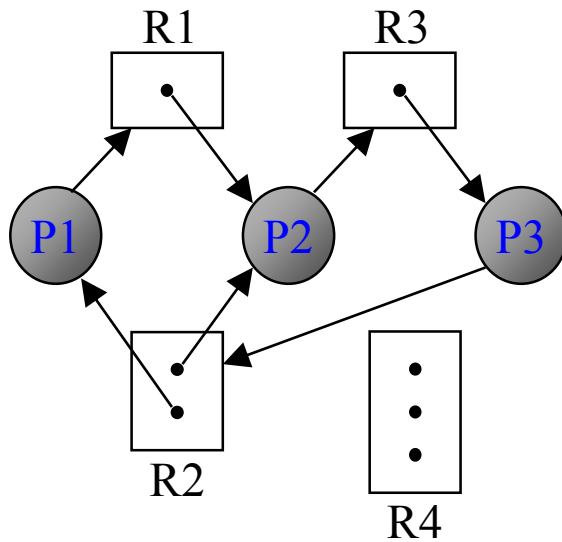
# Safe State



- When a process requests an available resource, system must decide if immediate allocation leaves the system in **safe state**.
- **Safe state**: there exists a **safe sequence** of all process
- **Safe sequence**: a sequence  $\langle P_1, P_2, \dots, P_n \rangle$  is safe if for each  $P_i$ , the resource requests of  $P_i$  can be satisfied by ...
  - Currently available resources, and
  - Resources held by  $P_j$ , where  $j < i$

# Safe Sequence

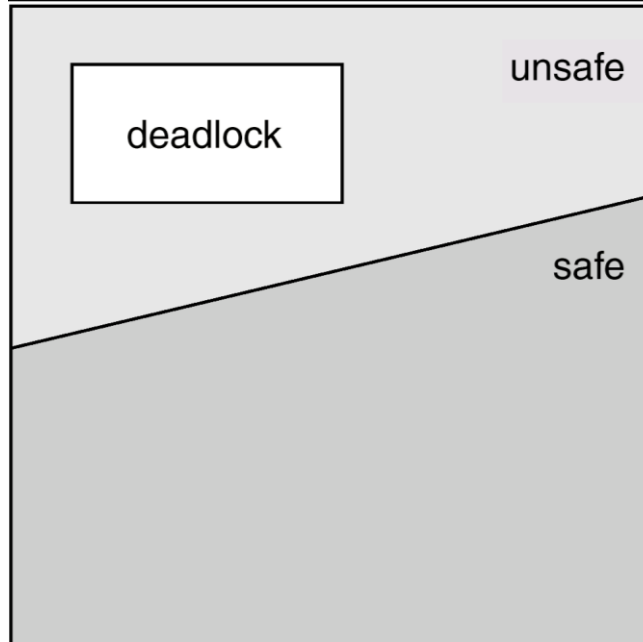
- Which state is safe?



P2→P1→P4→P3

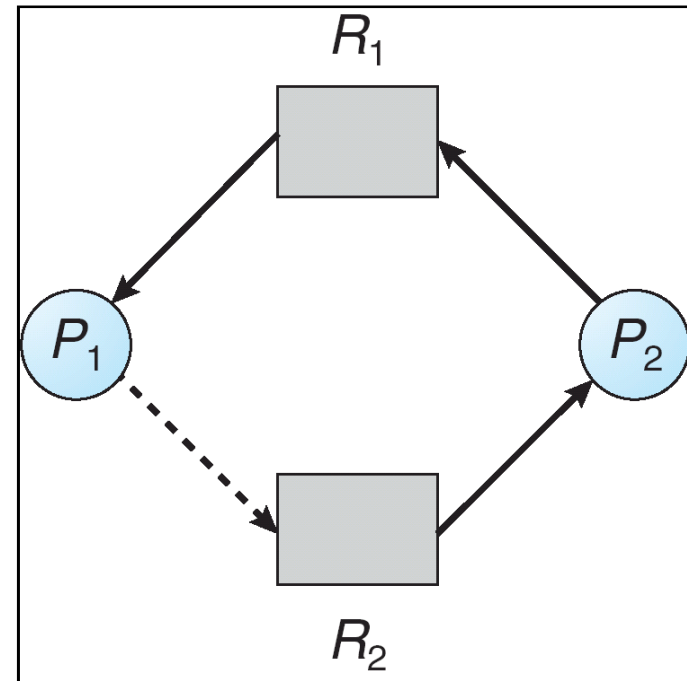
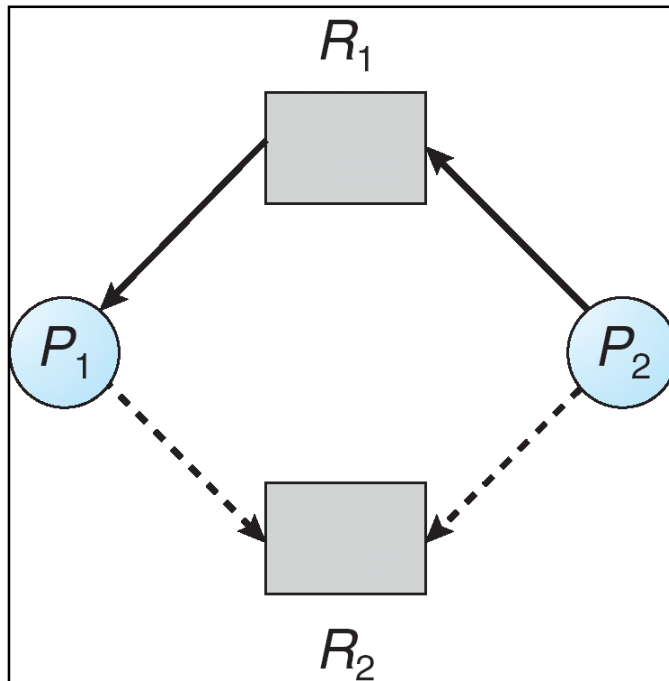
# Safe State and Deadlock

- If a system is in safe state → no deadlock
- If a system is in unsafe state → possibility of deadlock
- Deadlock avoidance is keeping a system in safe state.



# Detecting Unsafe State

- Claim edge  $P_i \rightarrow R_j$  (dashed line): process  $P_i$  **may request** resource  $R_j$  at some time
  - Claim edge can be converted into request edge



**unsafe state !**

# Deadlock Detection

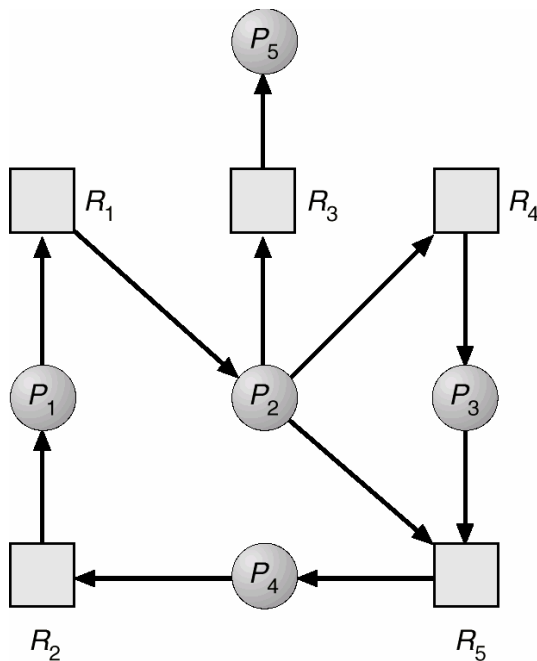
---



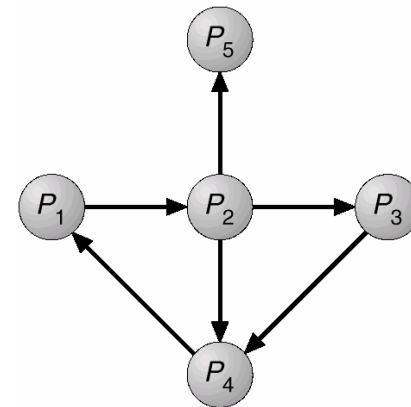
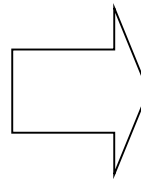
- Single instance of each resource type
- Multiple instances of a resource type (skip)

# Single Instance of Each Resource Type

- **Wait-for graph**: removing resource nodes from resource-allocation graph and collapsing the appropriate edges



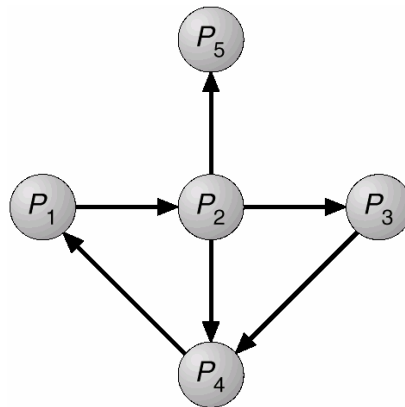
Resource-allocation graph



Wait-for graph

# Single Instance of Each Resource Type

- A deadlock exists if and only if the wait-for graph contains a cycle.
- To detect deadlock, system should
  - Maintain wait-for graph
  - Invoke an algorithm to detect a cycle –  $O(n^2)$



Wait-for graph

# Detection–Algorithm Usage



- When should we invoke the detection algorithm?
  - How often is a deadlock likely to occur?
  - How many processes will be affected by deadlock when it happens?
- We may invoke detection algorithm whenever a request for allocation cannot be granted immediately.
  - Deadlocks occur only when some processes makes a request that cannot be granted immediately
  - We can find the process which caused deadlock.



# Recovery from Deadlock

---



- Process termination
- Resource preemption

# Process Termination



- Abort all deadlocked processes
  - Too expensive
- Abort one process at a time until the deadlock is eliminated
  - Order of priority
    - Time from start / time to completion
    - Resources the process has used / needs to complete
    - How many processes will need to be terminated?
    - Is the process interactive or batch?

# Resource Preemption



## ■ Selecting a victim

- Minimizing cost (# of resources a process has, amount of time consumed so far, ...)

## ■ Rollback

- The selected process should return to some safe state and restart it.

## ■ Starvation

- How can we guarantee that resources will not always be preempted from the same process?