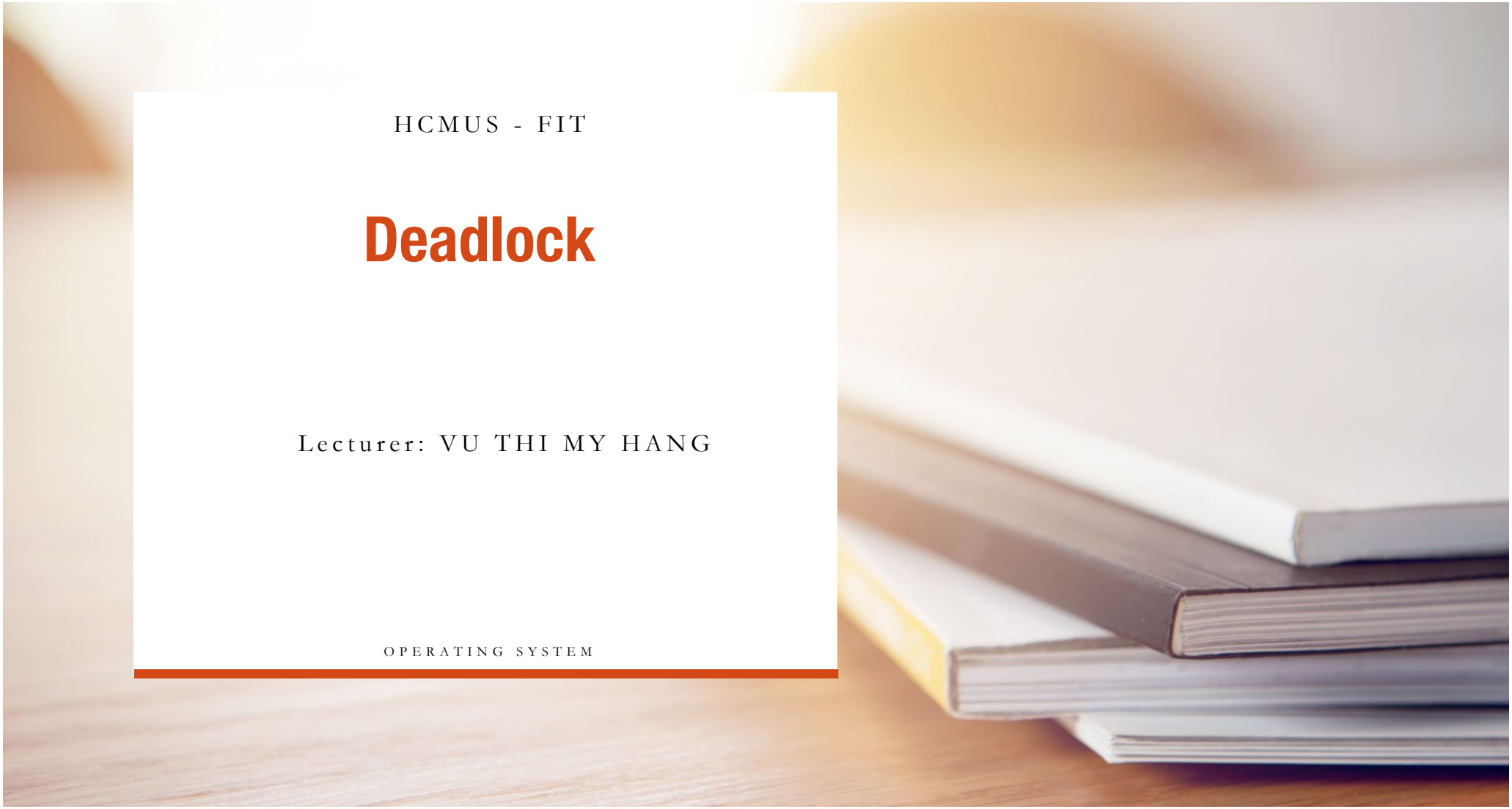


HCMUS - FIT

Deadlock

Lecturer: VU THI MY HANG

OPERATING SYSTEM

Plan

- Introduction to Deadlock
- Deadlock Handling

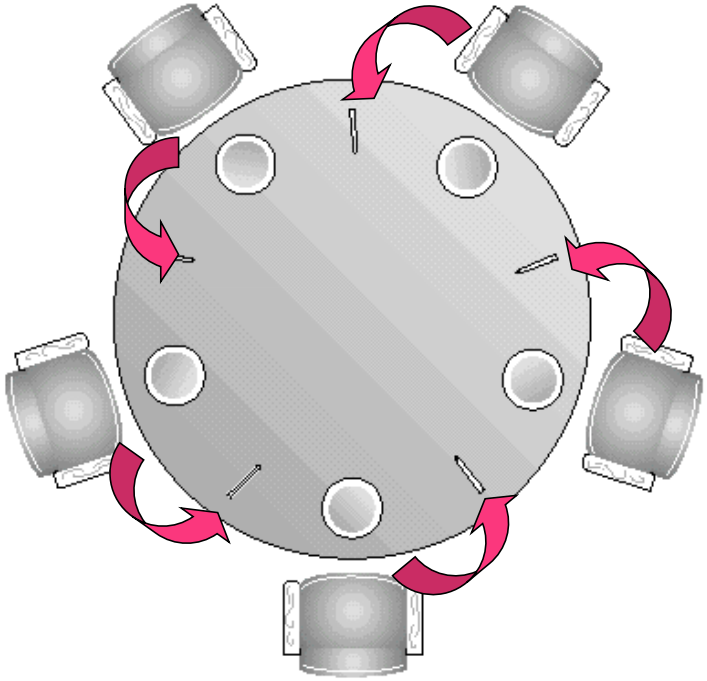
Plan

- **Introduction to Deadlock**
- Deadlock Handling

Recall: Dining-Philosophers Problem

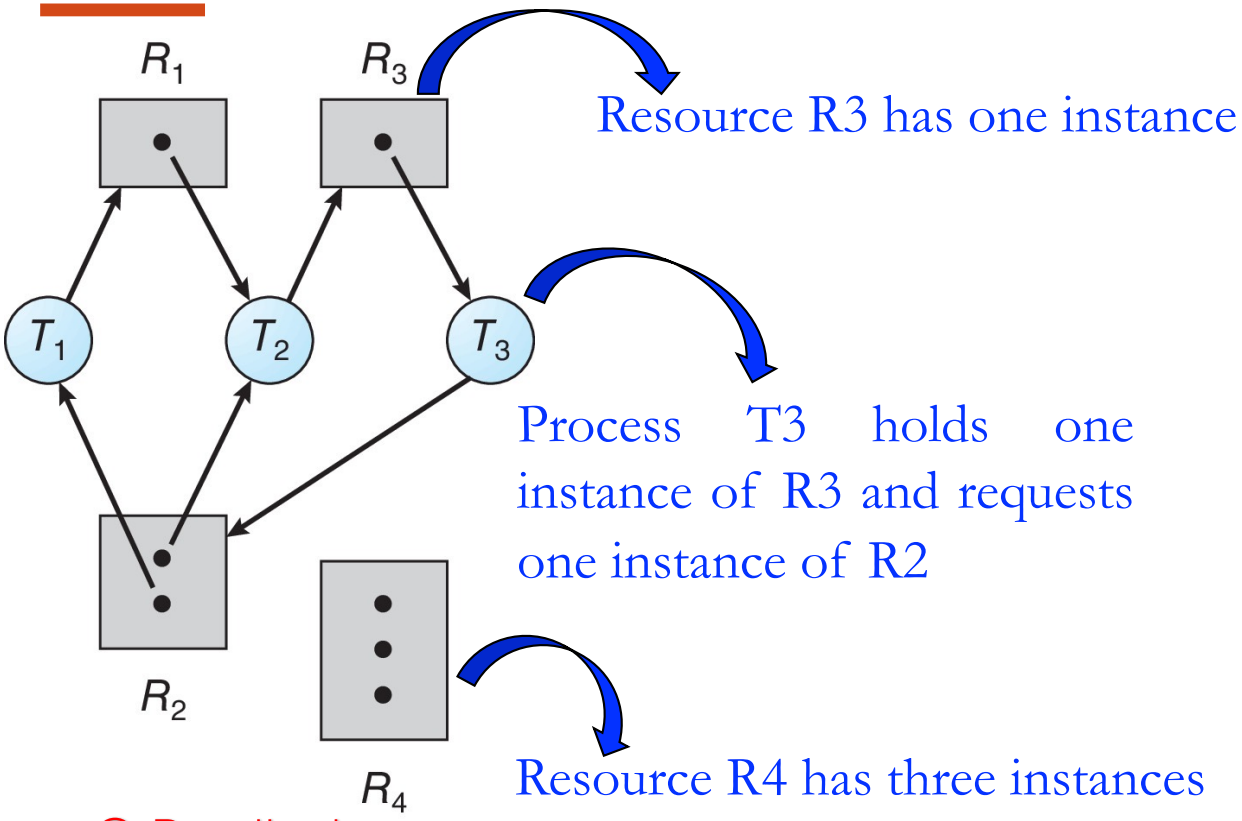
```
semaphore forks[5];
```

```
Philosopher (int i) {//i: philosopher number
    while(true) {
        down(forks[i]);              //take left fork
        down(forks[(i + 1)%5]); //take right fork
        eating( );
        up(forks[i]);                //put left fork back
        up(forks[(i + 1)%5]);       //put right fork back
    }
}
```



☹ Deadlock

Resource-Allocation Graph (RAG)

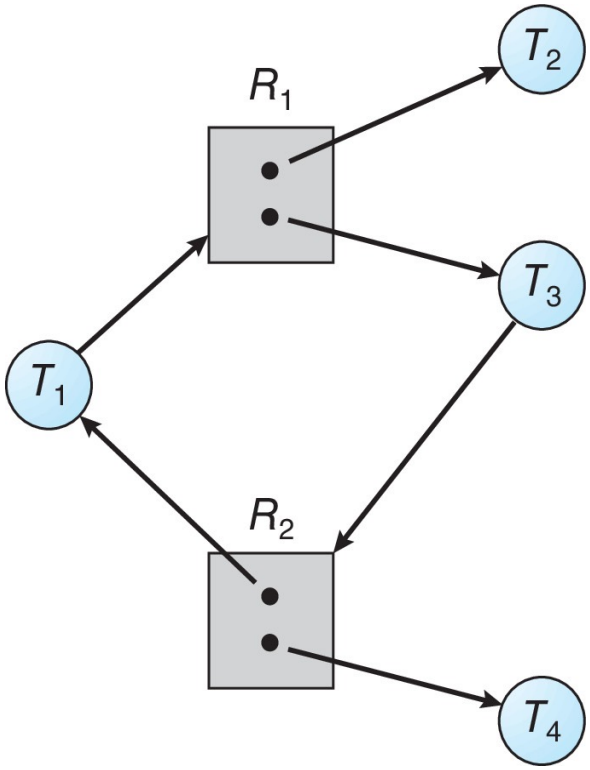


Resource R3 has one instance

Process T3 holds one instance of R3 and requests one instance of R2

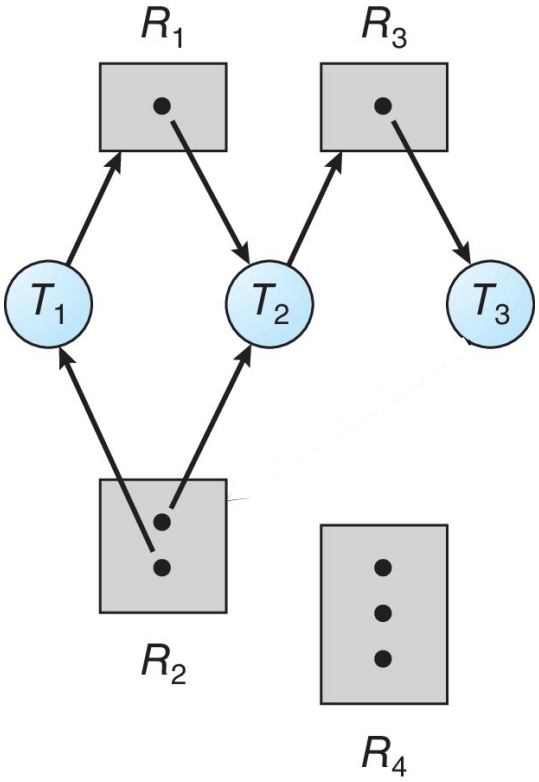
Resource R4 has three instances

☹ Deadlock



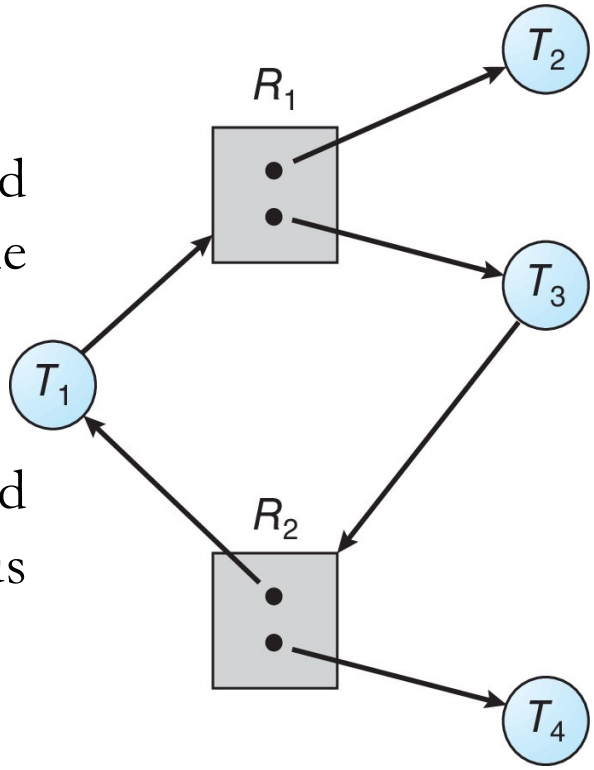
☺ No Deadlock

Resource-Allocation Graph (RAG)



☺ No Deadlock

- Graph with no cycle
  → *no deadlock*
- Graph with a cycle and resources having only one instance
  → *deadlock*
- Graph with a cycle and resources having various instances
  → *deadlock possibility*



☺ No Deadlock

Deadlock Definition

A set of processes is in a **deadlock** situation when each of them is waiting for resources allocated to other waiting processes in the set, therefore none of them can get all necessary resources to continue executing.

Conditions for Deadlock

All four following conditions must take place to cause deadlock

- **Mutual Exclusion**
  - ✓ Only one process can hold the resource at a time
- **Hold and Wait**
  - ✓ Process holds some resources while waiting for others, which are held by other waiting processes
- **No preemption**
  - ✓ A resource can be released only by the process holding it
- **Circular Wait**
  - ✓ There is a wait-cycle between at least two processes

Plan

- Introduction to Deadlock
- **Deadlock Handling**
  - *Ignorance*
  - *Prevention*
  - *Avoidance*
  - *Detection and Recovery*

Deadlock Ignorance

Ostrich Algorithm: pretend that deadlocks do not cause any problem

- Deadlocks occur rarely
- Deadlock handling is much more costly

Deadlock Prevention

Eliminate the occurrence of one of the four conditions causing deadlocks

- Mutual Exclusion
  - ☹ Not practical
- Hold and Wait
  - ✓ Process must require all necessary resources when starting executing
  - ✓ Or release currently holding resources before requesting others
  - ☹ Difficult to know all necessary resources at the beginning
  - ☹ Low resource utilization
  - ☹ Starvation possibility

Deadlock Prevention (cont.)

Eliminate the occurrence of one of the four conditions causing deadlocks

- No preemption
  - ✓ OS may preempt resources from a process in some circumstance
  - ☹ May cause errors for some resources (e.g., printer, files)
- Circular Wait
  - ✓ Allocate different resource types according to a priority order

Deadlock Avoidance

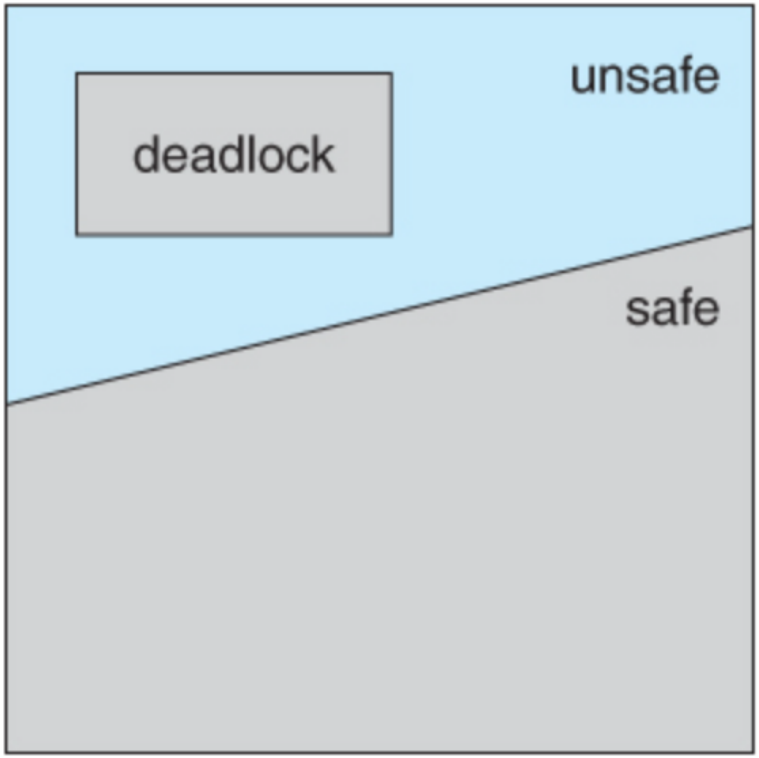
Test deadlock possibility before granting resources to avoid a future deadlock

- OS requires processes to provide additional information (e.g., maximum requirement) to decide on resource allocation

- Whenever a process requests a resource, OS tests deadlock possibility by using ***deadlock avoidance algorithms***
  - ✓ If a resource request may lead to a future deadlock, the resource can NOT BE GRANTED
  - ✓ One instance of resource type: RAG
  - ✓ Multiple instances of resource type: Banker's algorithm

Deadlock Avoidance (cont.)

- Safe state: no deadlock
  - ✓ System is safe if there exists a ***safe sequence*** for resource allocation
  - ✓ All resource requests will be granted without entering a deadlock situation
- Unsafe state: may lead to deadlock
- Deadlock avoidance: ensure that the system will be in a safe state after resource allocation



Deadlock Avoidance (cont.)

- Banker's algorithm
  - ✓ Also referred to as safety algorithm, developed by Edsger Dijkstra
- Notions used in Banker's algorithm
  - ✓ Available: available resources
  - ✓ Max: maximum resource requirements of each process
  - ✓ Allocation: resources allocated to each process
  - ✓ Need: remaining resources needed for each process
    - *Need = Max – Allocation*
  - ✓ Request: resource request chain of a process
    - *For a request, test if the system will be in a safe state*

Deadlock Avoidance (cont.)

Example of Banker's algorithm

	Allocation			Max			Need		
	C	D	E	C	D	E	C	D	E
P_0	0	1	0	7	5	3	7	4	3
P_1	2	0	0	3	2	2	1	2	2
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available (Work)		
C	D	E
10	5	7

The system is safe?

One possible safe sequence: (P1, P3, P0, P2, P4)

Deadlock Avoidance (cont.)

Example of Banker's algorithm

	Allocation			Max			Need		
	C	D	E	C	D	E	C	D	E
P_0	0	1	0	7	5	3	7	4	3
P_1	2	0	0	3	2	2	1	2	2
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available (Work)		
C	D	E
3	3	2

$Request_{P1}$ = (1, 0, 2) will be granted safely?

$Request_{P1} < Need_{P1} < Available$

➔ $Request_{P1}$ granted ➔ Update $Allocation_{P1}$, $Need_{P1}$, Available

Deadlock Avoidance (cont.)

Example of Banker's algorithm

	Allocation			Max			Need		
	C	D	E	C	D	E	C	D	E
P_0	0	1	0	7	5	3	7	4	3
P_1	3	0	2	3	2	2	0	2	0
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available (Work)		
C	D	E
2	3	0

One possible safe sequence: (P1, P3, P0, P2, P4)

$Request_{P1}$ = (1, 0, 2) will be granted safely? ✓

$Request_{P1} < Need_{P1} < Available$

➔ $Request_{P1}$ granted ➔ Update $Allocation_{P1}$, $Need_{P1}$, Available

Deadlock Detection and Recovery

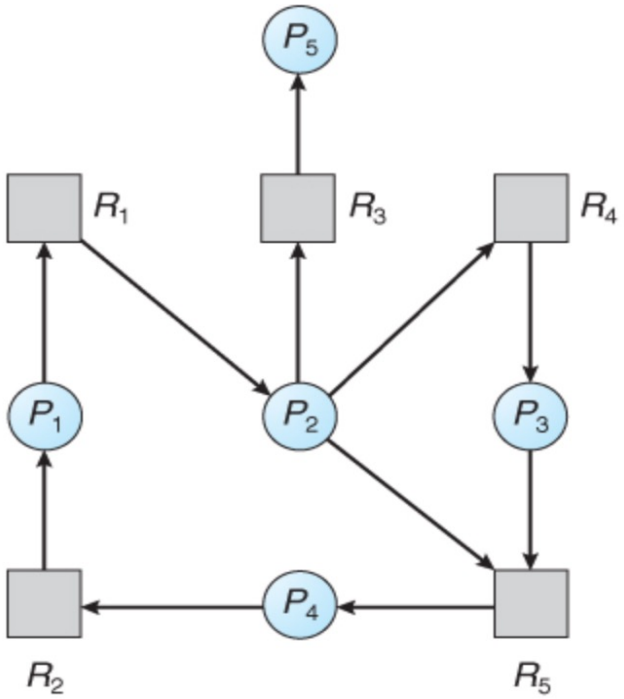
Allow deadlock occurrence and then recover the system

- **Deadlock Detection**
  - ✓ One instance of resource type: *wait-for graph* (i.e., a variation of RAG)
  - ✓ Multiple instances of resource type: *matrix-based detection algorithm* (also referred to as a variation of Banker's algorithm)

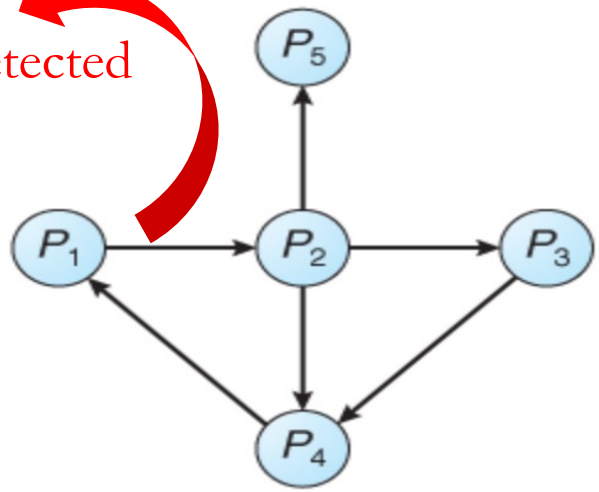
- **Recovery**
  - ✓ Terminate deadlocked processes
  - ✓ Rollback system to the previous safe checkpoint
  - ✓ Preempt resources

Deadlock Detection and Recovery (cont.)

Example of using a wait-for graph to detect deadlocks



Cycles existing in the wait-for graph → deadlock detected



Resource-Allocation Graph → *Wait-For Graph*

Deadlock Detection and Recovery (cont.)

Example of using matrix-based detection algorithm

	Allocation			Request (Claim)		
	C	D	E	C	D	E
P_0	0	1	0	7	5	3
P_1	2	0	0	3	2	2
P_2	3	0	2	9	0	2
P_3	2	1	1	2	2	2
P_4	0	0	2	4	3	3



Available		
C	D	E
7	4	5

Finish				
P_0	P_1	P_2	P_3	P_4
0	1	0	1	1

P1, P3 and P4 can finish and release holding resources

☹ P0 and P2 could not finish → Deadlock detected