ECE437/CS481

M05A: DEADLOCKS

CHAPTER 7

Xiang Sun

The University of New Mexico

Deadlock Problem

❑ Deadlock problem

- A set of blocked processes, each of which holds a **resource** and waits to acquire another **resource** held by another process.

❑ Example of deadlock

- System has two hard disks; P1 and P2 each hold one hard disk, and each needs another one.

- Semaphores A and B, initialized to 1

P_0	P_1
wait (A);	wait(B)
wait (B);	wait(A)

Deadlock Problem

❑ Deadlock problem

- A set of blocked processes, each of which holds a resource and waits to acquire another resource held by another process.

❑ Example of deadlock

- 200K bytes memory space is available for allocation.
- Deadlock occurs if both processes progress to their second requests.

Deadlock & Starvation

Deadlock	Starvation
A deadlock occurs when a set of processes in a system is blocked waiting on requirements that can **NEVER** be satisfied. 	Starvation occurs when a process waits for a resource that continually available but is never assigned to that process because of priority or a flaw in the design of the scheduler.

❑ Difference:

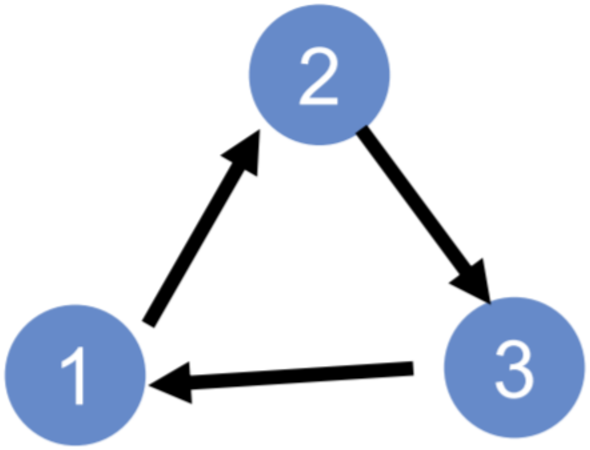
- In starvation, a process could finally get the requested resource, whereas a deadlock process is permanently blocked.
- In starvation, the resource under contention is in continuous use, whereas this is not true in a deadlock.

Deadlock Characterization

- Four conditions are necessary for deadlocks to occur:
  - **Mutual exclusion**
    - ✓ if a process is using a resource, no other process can use that resource until the first process releases it.
  - **Wait while hold**
    - ✓ processes hold previously acquired resources while waiting for additional resources
  - **No preemption**
    - ✓ A resource cannot be preempted from a process without aborting the process
  - **Circular wait**
    - ✓ ∃ a set of processes {P1, P2, … PN}, such that P1 is waiting for P2, P2 for P3, …. and PN for P1

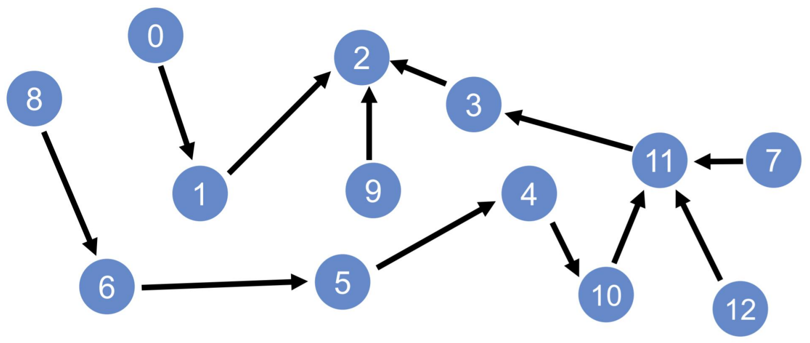
Deadlock Detection based on WFG

- Create a Wait-For Graph (WFG):
  - **Node:** process
  - **Edge with direction:** the blocking/waiting relation between processes, e.g., e(pi,pj) means that pi needs a resource currently held by pj, or pi is waiting for pj.
  - **Cycle:** indication of deadlock



Deadlock Detection based on WFG

- Testing for cycle in WFG:

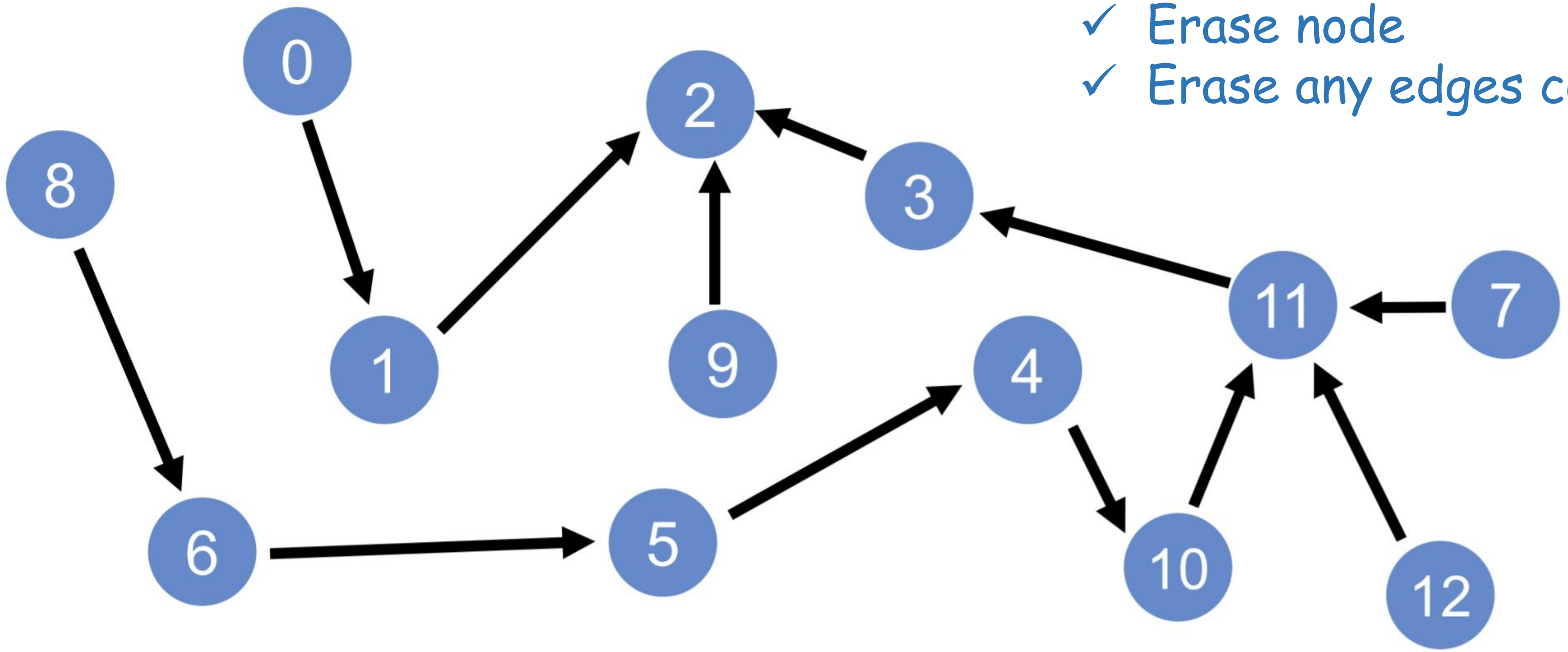


  - Find a node with no outgoing edges
    - ✓ Erase node
    - ✓ Erase any edges coming into it
  - Intuition: this was a process waiting on nothing. It will eventually finish, and anyone waiting on it will no longer be waiting.
  - Results:
    - ✓ Erase whole graph ↔ graph has no cycles
    - ✓ Graph remains ↔ deadlock

Graph reduction algorithm

Deadlock Detection based on WFG

❑ Graph Reduction in WFG: Example 1

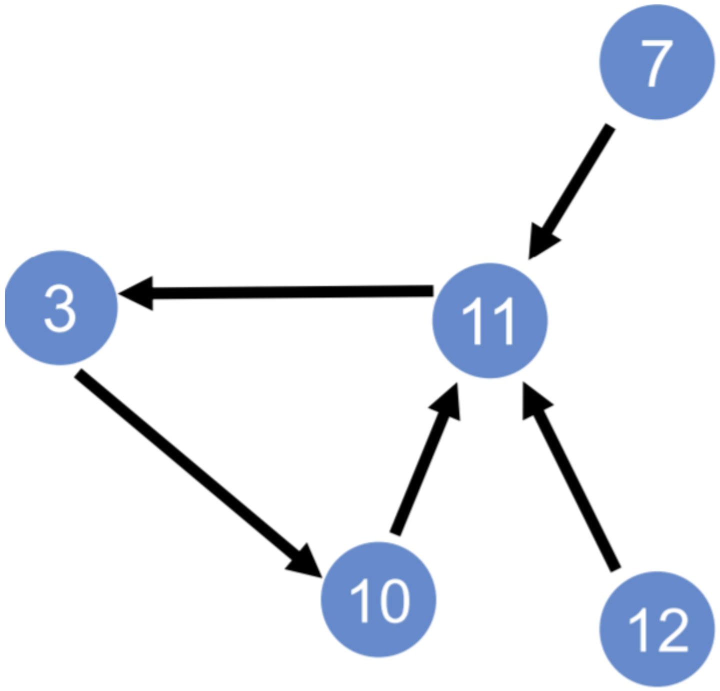


- ➢ Find a node with no outgoing edges
  - ✓ Erase node
  - ✓ Erase any edges coming into it

✓ Graph can be fully reduced, hence there was no deadlock at the time the graph was drawn.

Deadlock Detection based on WFG

❑ Graph Reduction in WFG: Example 2



- Find a node with no outgoing edges
  - ✓ Erase node
  - ✓ Erase any edges coming into it

✓ No node with no outgoing edges… Irreducible graph, contains a cycle ↔ deadlock.

Deadlock Detection based on RAG

❑ Create a Resource Allocation Graph (RAG):

➢ Two types of Nodes

node circles: $P = \{P_1, P_2, \ldots, P_n\}$

- the set consisting of all the processes in the system

node squares: $R = \{R_1, R_2, \ldots, R_m\}$

- the set consisting of all resource types in the system
- (dots within a square --- resource instances)

➢ Two types of edges

request edge:

directed edge $P_i \rightarrow R_j$

- ❖ edge from a process node to a resource node

assignment edge (granting edge):

directed edge $R_j \rightarrow P_i$

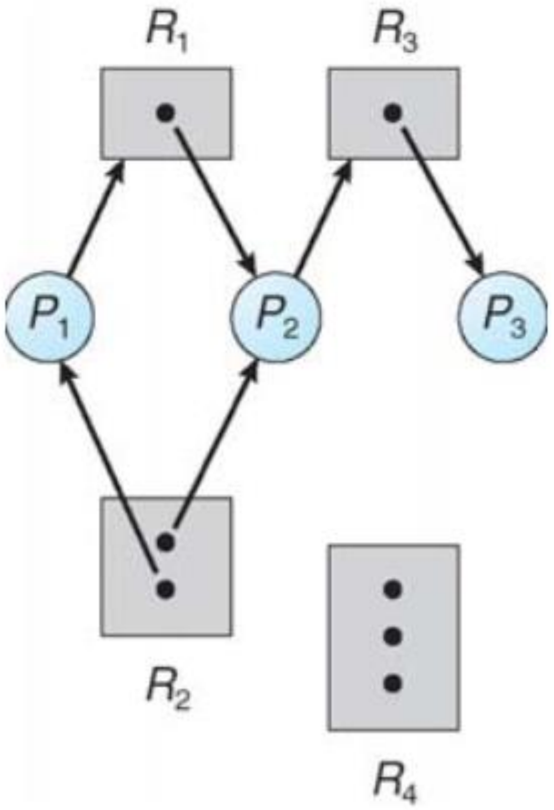
- ❖ edge from a resource instance to a process node

Deadlock Detection based on RAG

❑ RAG

- If each resource type has exactly one instance
  - ✓ a cycle in the graph is both a necessary and a sufficient condition for the existence of deadlock
- If each resource type has several instances
  - ✓ a cycle in a graph is a necessary but not a sufficient condition for the existence of deadlock

P_1 hold R_2 request R_1	R_1 --- m=1
P_2 hold R_1 & R_2 request R_3	R_2 --- m=2
P_3 hold R_3	R_3 --- m=1
	R_4 --- m=3



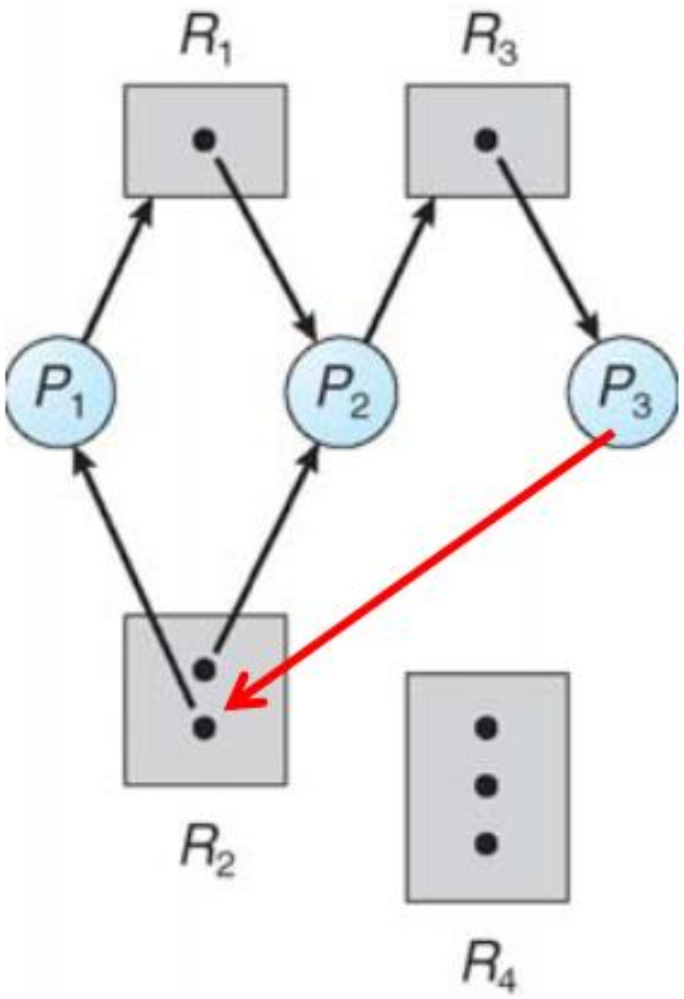
Deadlock Detection based on RAG

- RAG: a circle incurs a deadlock

P_1 hold R_2 request R_1 P_2 hold R_1 & R_2 request R_3 P_3 hold R_3	R_1 --- m=1 R_2 --- m=2 R_3 --- m=1 R_4 --- m=3

P_3 request R_2

Deadlock!



Deadlock Detection based on RAG

- RAG: a circle does not incur a deadlock

P_1 hold R_2 request R_1 P_2 hold R_1 R_3 hold R_1 request R_2 P_4 hold R_2	R_1 --- m=2 R_2 --- m=2

With a cycle,
But no deadlock!