Operating Systems
CS 415

Lecture 10:
Deadlocks

Suyash Gupta
Assistant Professor
Distopia Labs and ONRG
Dept. of Computer Science
(E) suyash@uoregon.edu
(W) gupta-suyash.github.io

UNIVERSITY OF OREGON

Announcements

- **Suyash Gupta**
  - Office Hours: Thursday, 10-11am, Deschutes 334

- **Nihal Balivada (TA)**
  - Office Hours: Wednesday/ Friday, 11-12pm, Deschutes 335

- **Ranjitha Rani (TA)**
  - Office Hours: Monday /Tuesday, 1-2pm, Deschutes 229

Assignment 3 is will be out today!

- **Deadline → June 3, 2026 at 11:59pm PST**
- Please start working and talk to us if you are stuck.

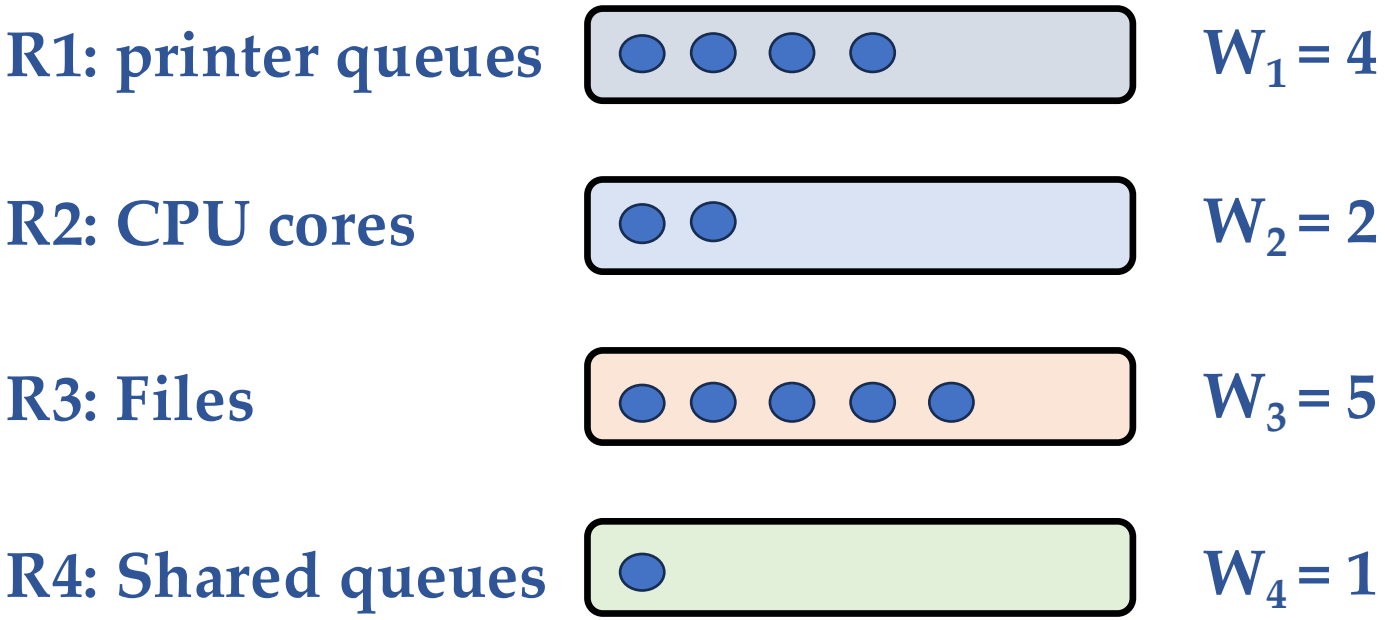
- **Final → June 10, 2026 at 12:30pm PST, STB 145**
  - Closed book, no cheat sheets, no discussions.

Last Class

- Synchronization (Chapter 6 + some parts of Chapter 7)
- Next, we move to Chapter 8!

Shared Resources

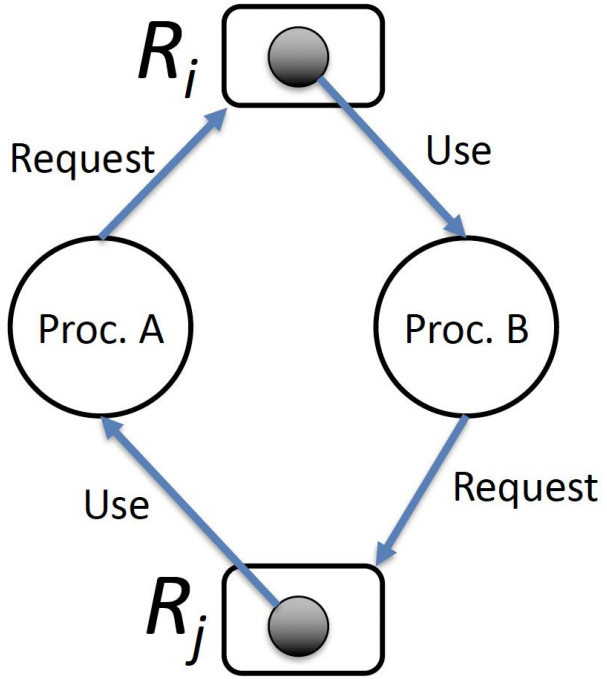
- System consists of resources.
- Resource types R1, R2, . . ., Rm.
- Each resource type Ri has Wi instances.



Deadlock

Deadlock

- Given a set of processes running concurrently, a deadlock is a condition where:
  - The computation can not make any progress because all processes are blocked while waiting for resources currently owned by other processes in the set.
- Deadlock can prevent processes from completion!

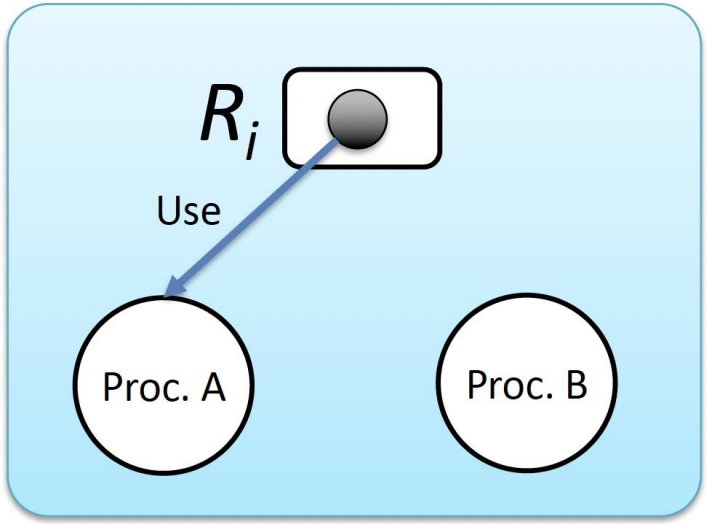


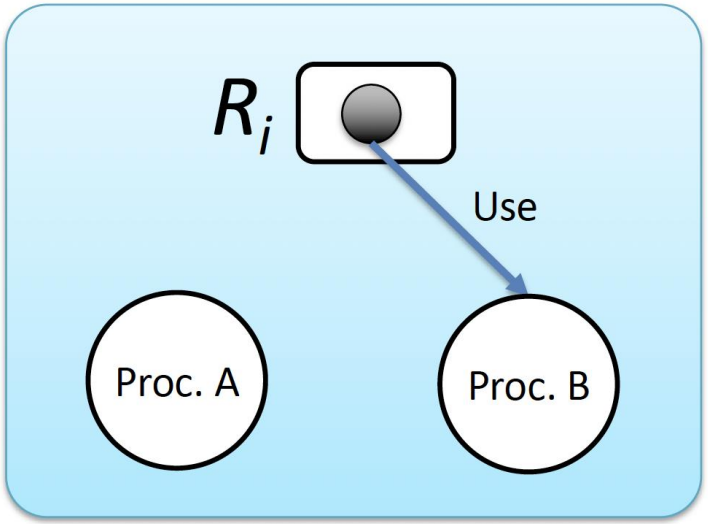
Deadlock Characterization

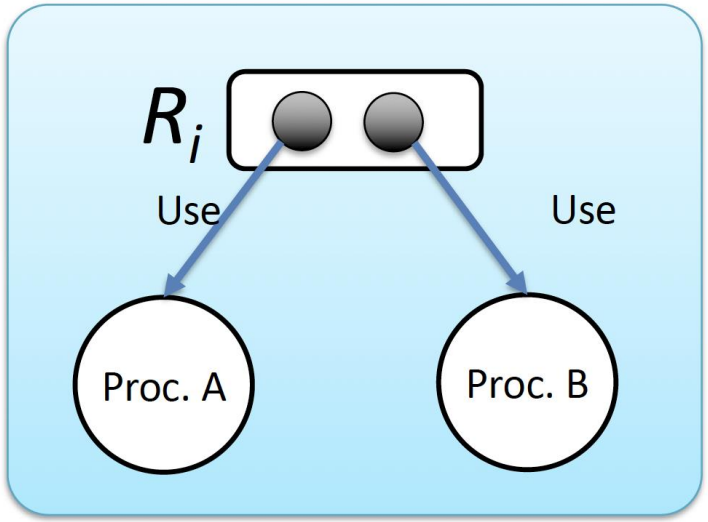
- Deadlock can arise if four conditions hold simultaneously:
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait

Mutual Exclusion

- Only one process at a time can use a resource instance.

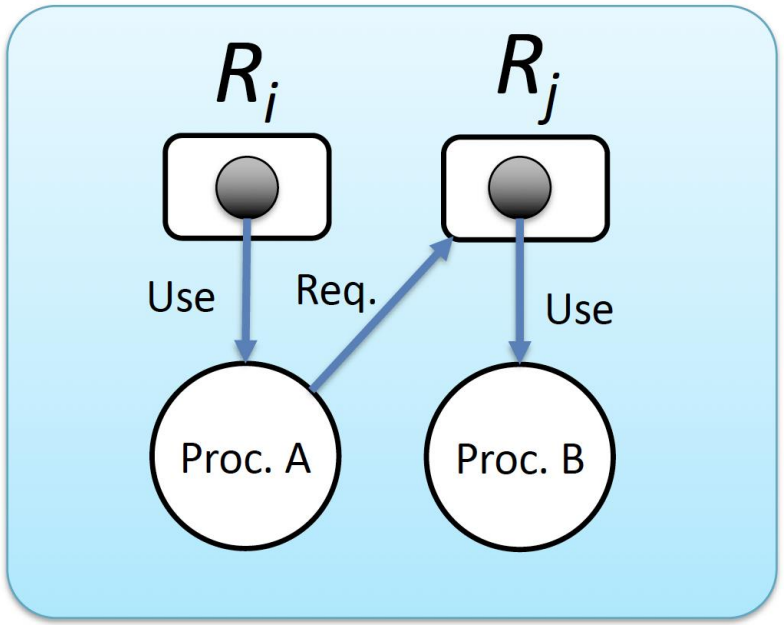






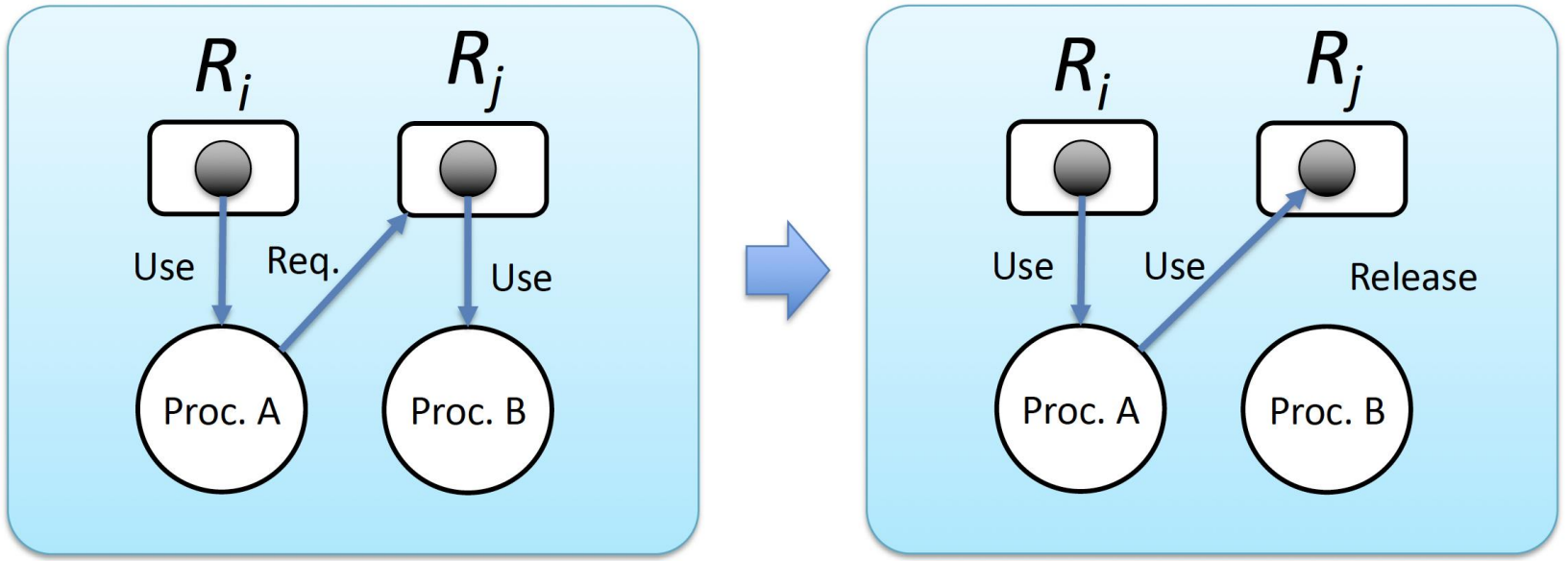
Hold and Wait

- A process holding at least one resource is waiting to acquire additional resources held by other processes.



No Preemption

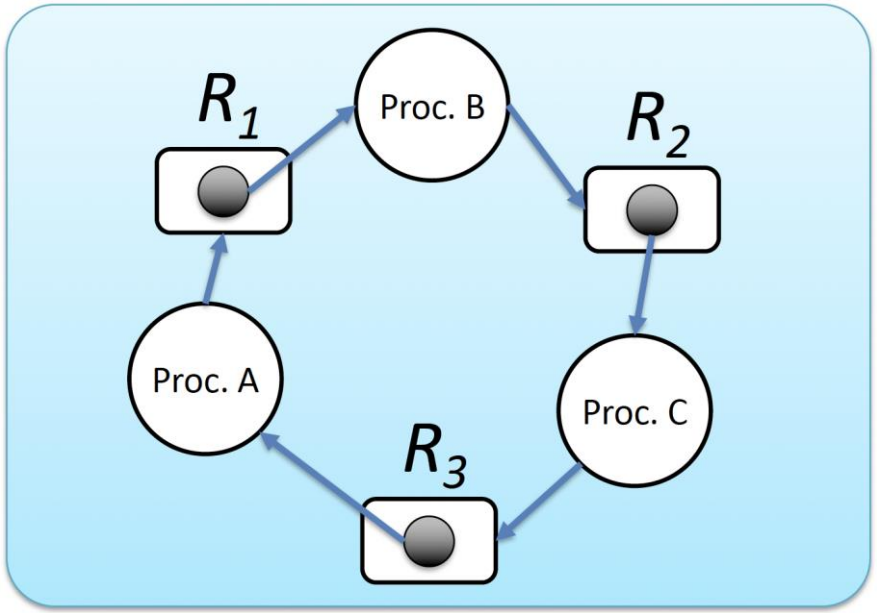
- A resource can be released only voluntarily by the process holding it, after that process has completed its task using the resource.



Rj can only be released by its current user -- process B. It cannot be forcefully taken by process A while process B is still using it.

Circular Wait

- There exists a set {P1, P2, …, Pn} of waiting processes such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3, …, Pn–1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P1. Note: all processes in the set must be involved.



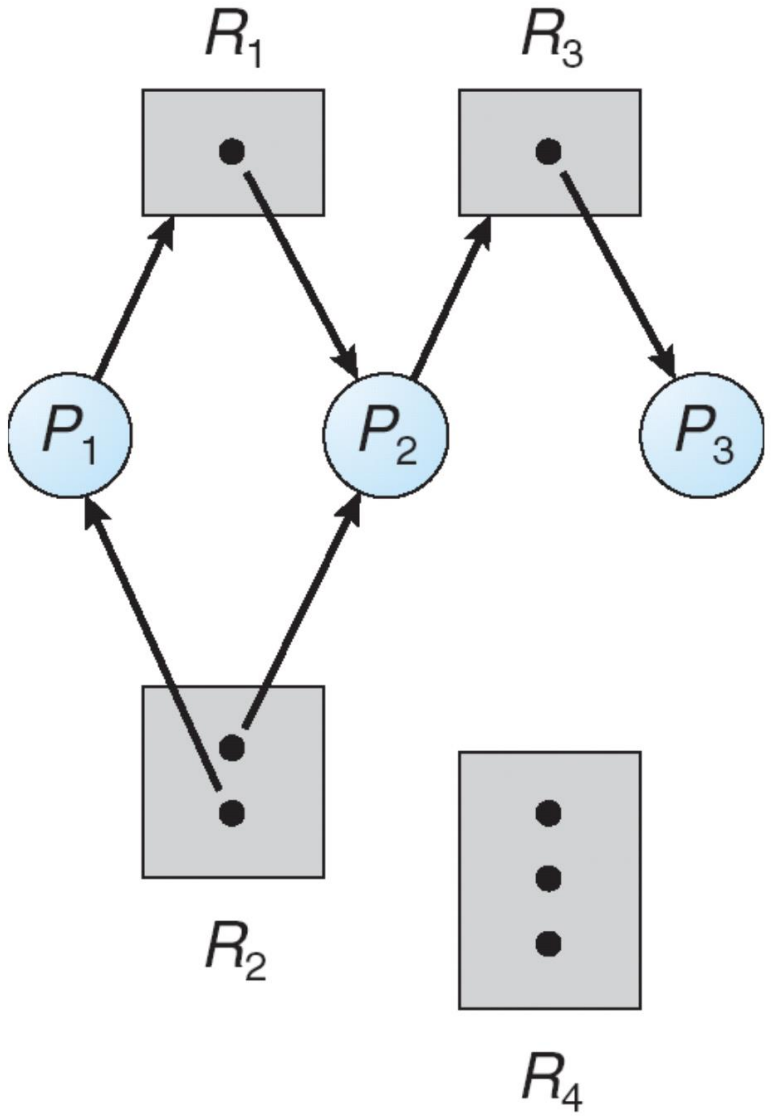
How to detect Deadlocks?

Resource Allocation Graph

- Determines state of the resource: a set of vertices V and a set of edges E.
- V is partitioned into two types:
  - P = {P1, P2, …, Pn}, set of all processes.
  - R = {R1, R2, …, Rm}, set of all resource types.
- Request edge
  - Directed edge Pi ⟹ Rj
- Assignment (Holds) edge
  - Directed edge Rj ⟹ Pi

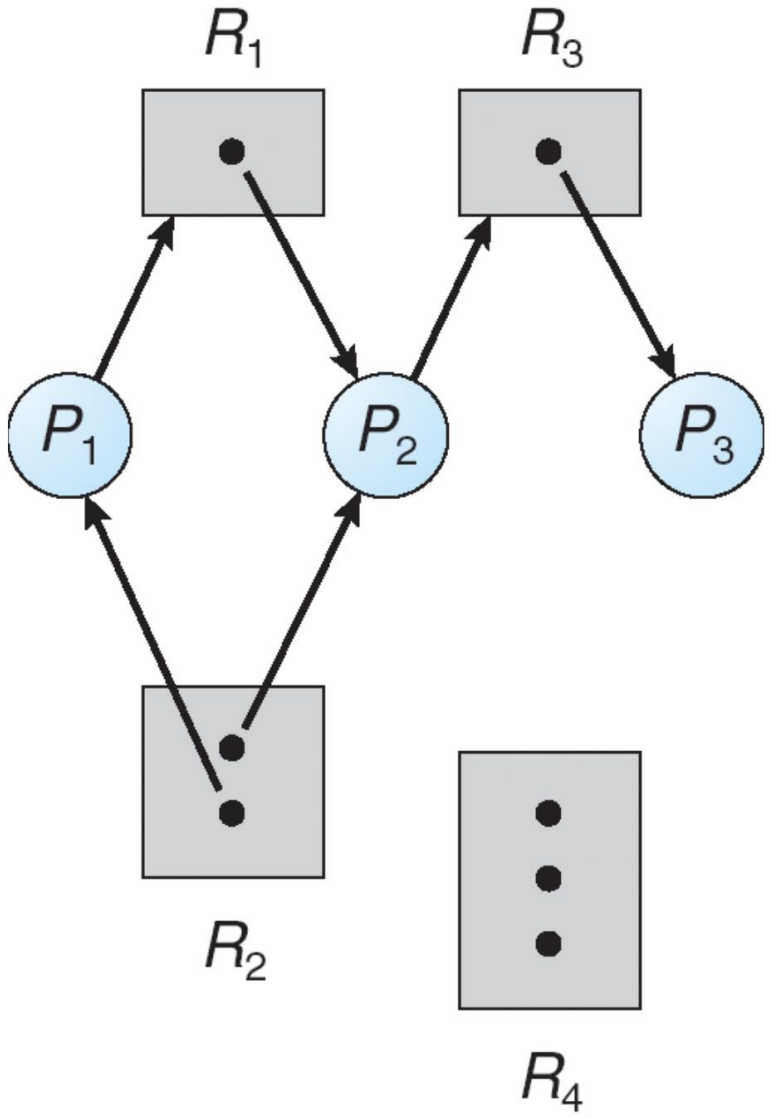
Resource Allocation Graph

Is there a deadlock?



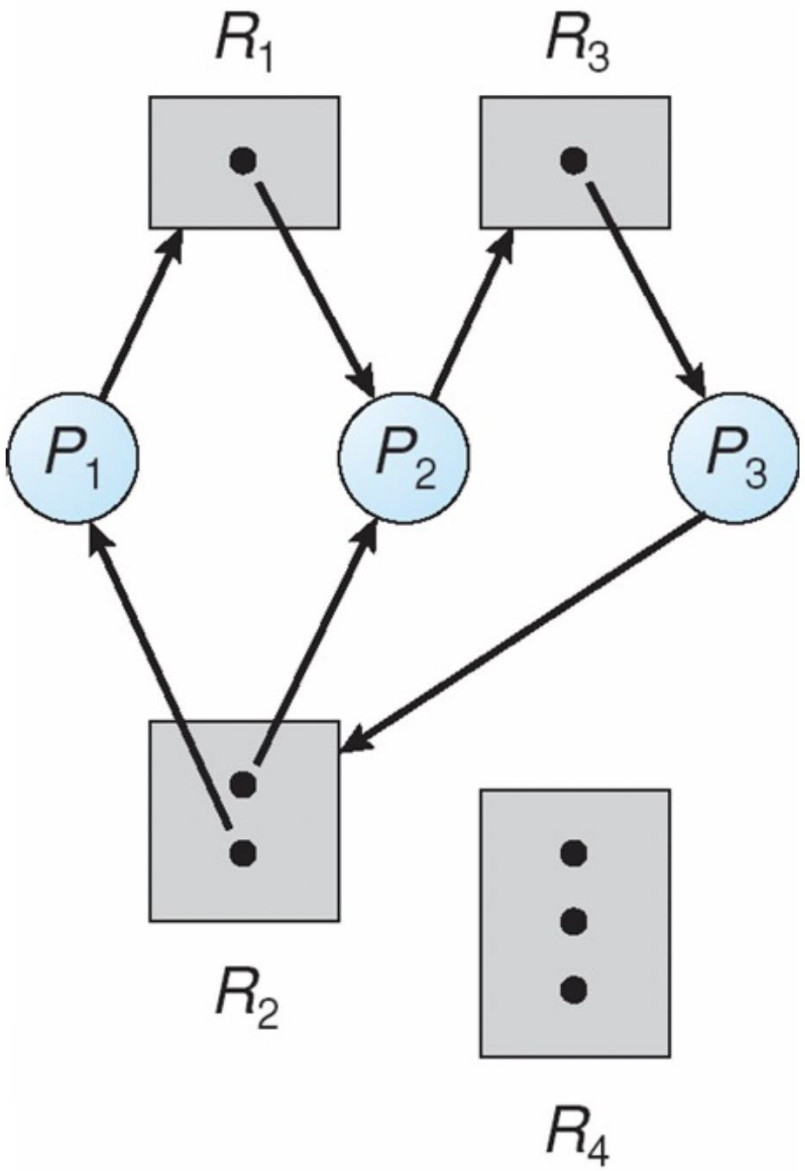
Resource Allocation Graph

Is there a deadlock?
No!



Resource Allocation Graph

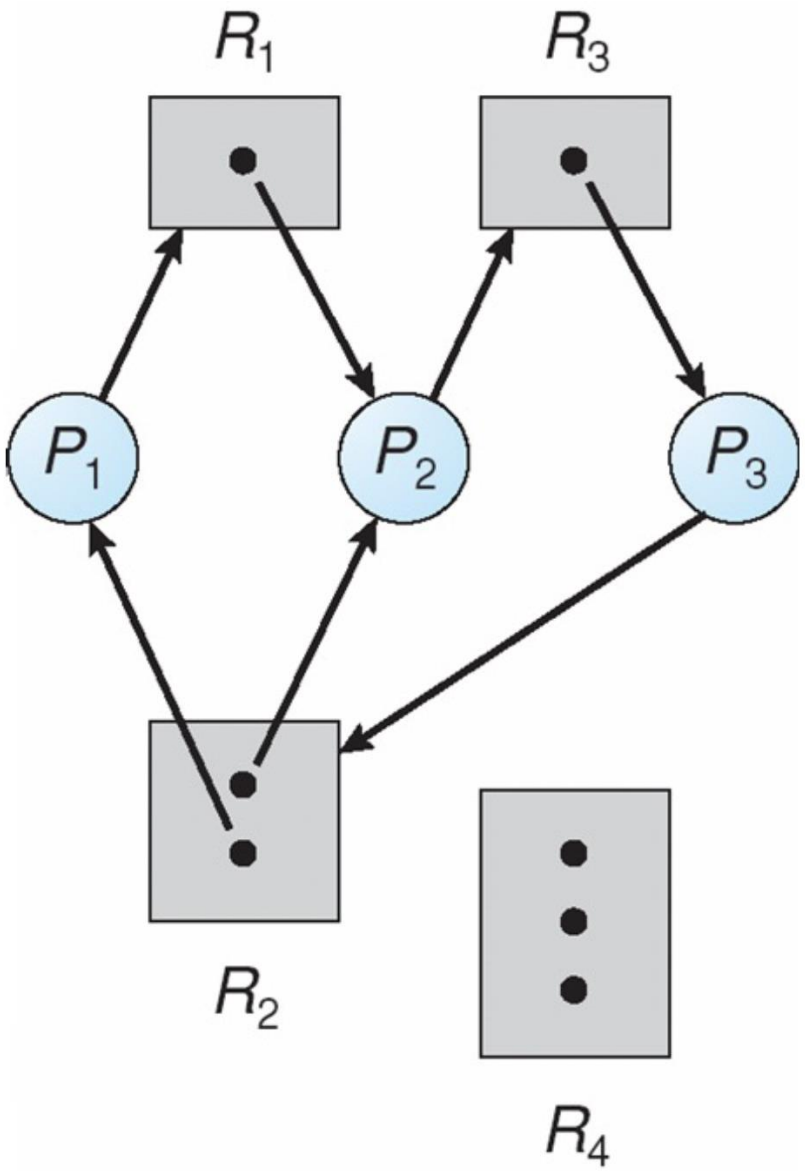
Is there a deadlock?



Resource Allocation Graph

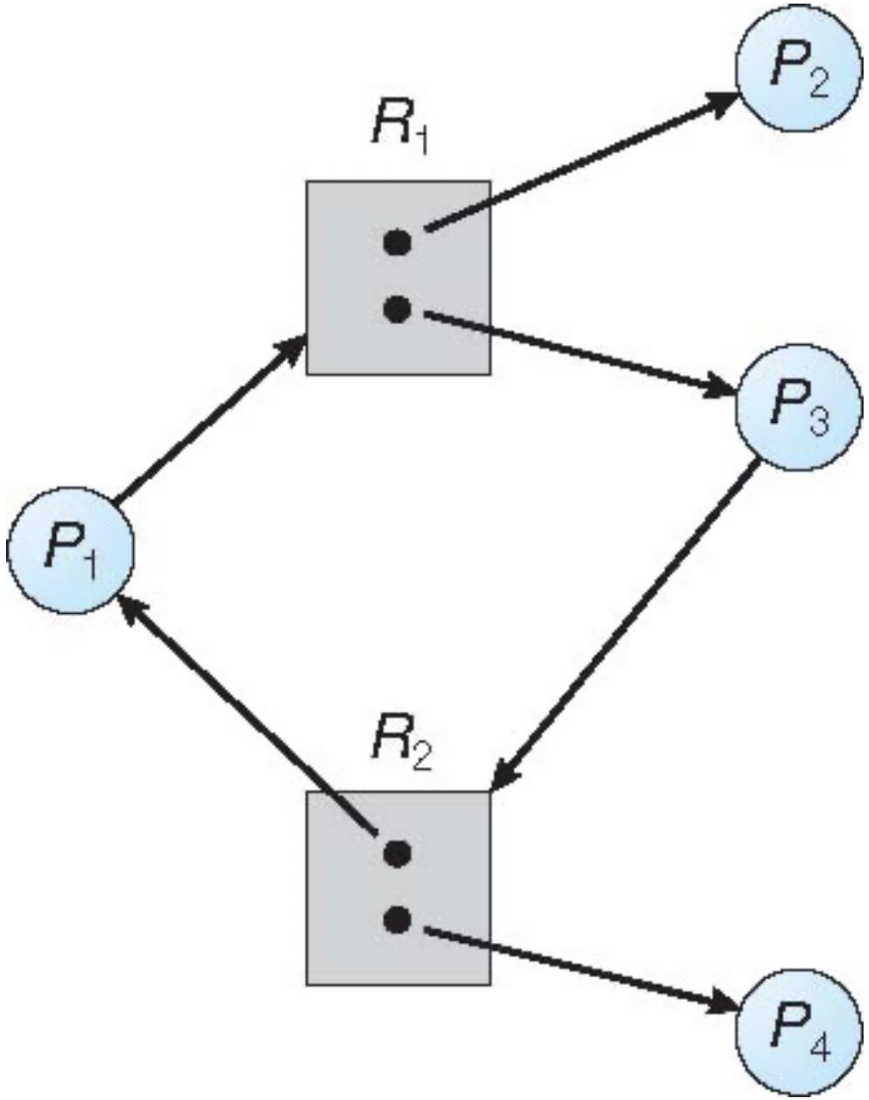
Is there a deadlock?

Yes! Circular wait



Resource Allocation Graph

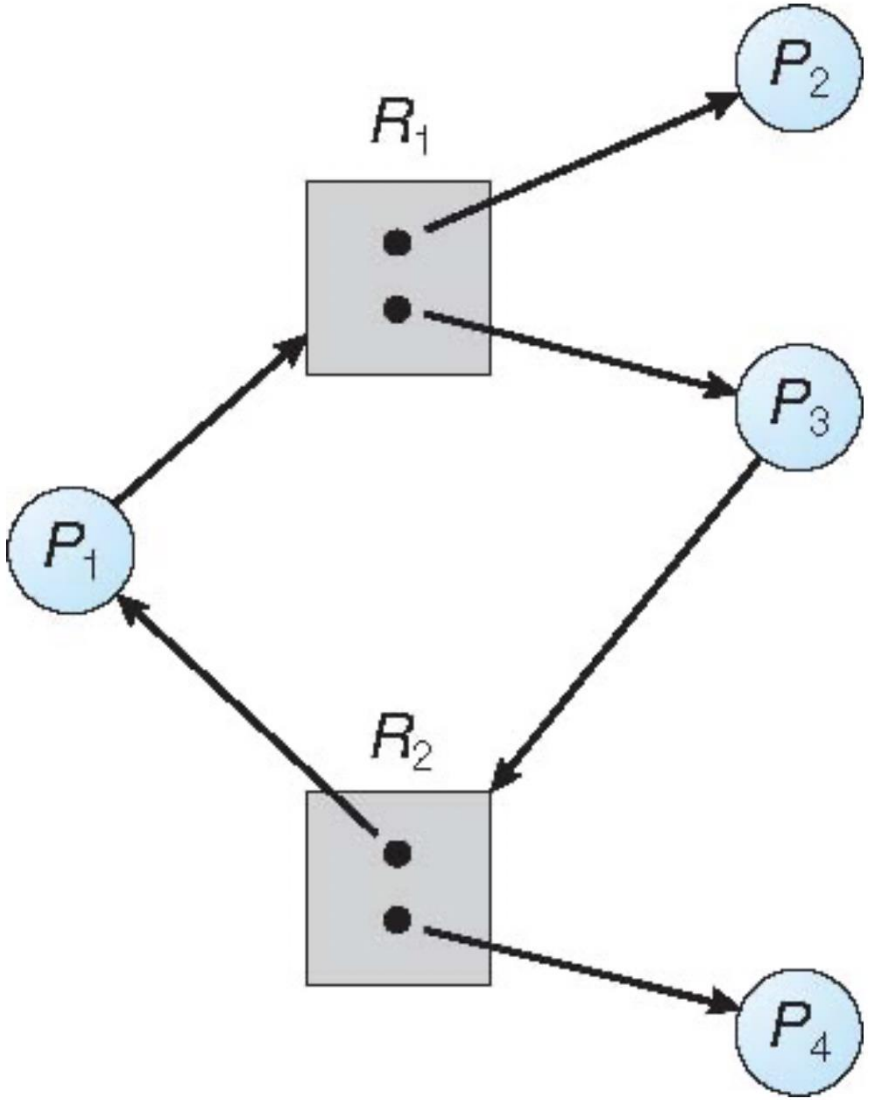
Is there a deadlock?



Resource Allocation Graph

Is there a deadlock?

Yes! Circular wait



Resource Allocation Graph

- If resource allocation graph contains no cycles:
  - There can be no deadlock.

- If resource allocation graph contains a cycle:
  - If there is only one instance per resource type → Deadlock.
  - If there are several instances per resource type → Possibility of deadlock.

Methods for Handling Deadlocks

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Recovery

Deadlock Prevention

Deadlock Prevention

- Ensuring that at least one of the following conditions do not hold:
  - Mutual Exclusion
  - Hold and Wait
  - No Preemption
  - Circular Wait

Deadlock Prevention: Mutual Exclusion

- Aim to remove unnecessary conditions for mutual exclusion.

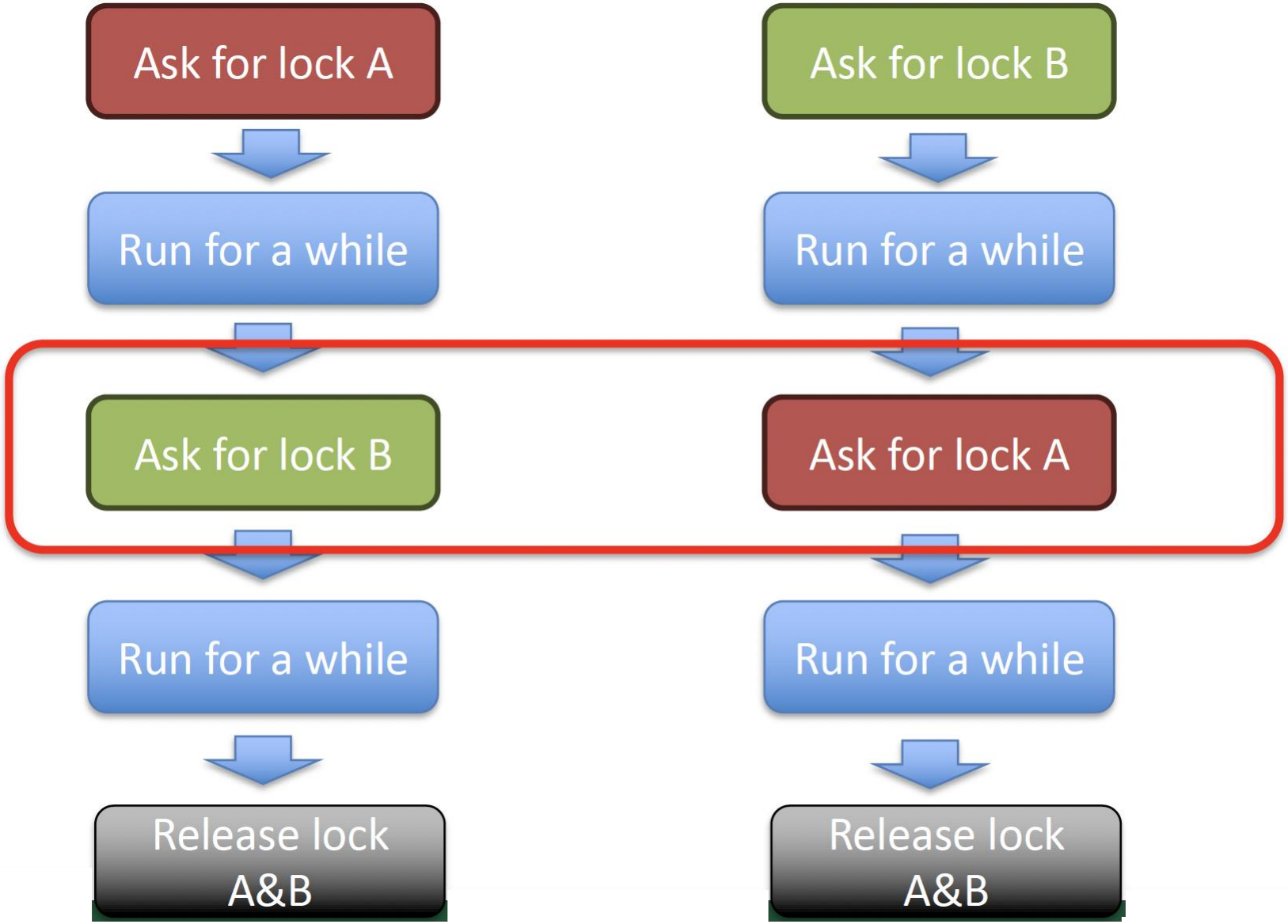
- If we know all processes will just read this file, is mutual exclusion still necessary?
  - Maybe allow Read-lock and Write-locks.
  - Multiple processes can read simultaneously.
  - POSIX provides support.

Deadlock Prevention: Hold and Wait

- Require a process to request and be allocated **All** of its resources before it begins execution.
- If it fails to acquire any resource, **release all** and start over.
- May lead to low resource utilization

Deadlock Prevention: Hold and Wait

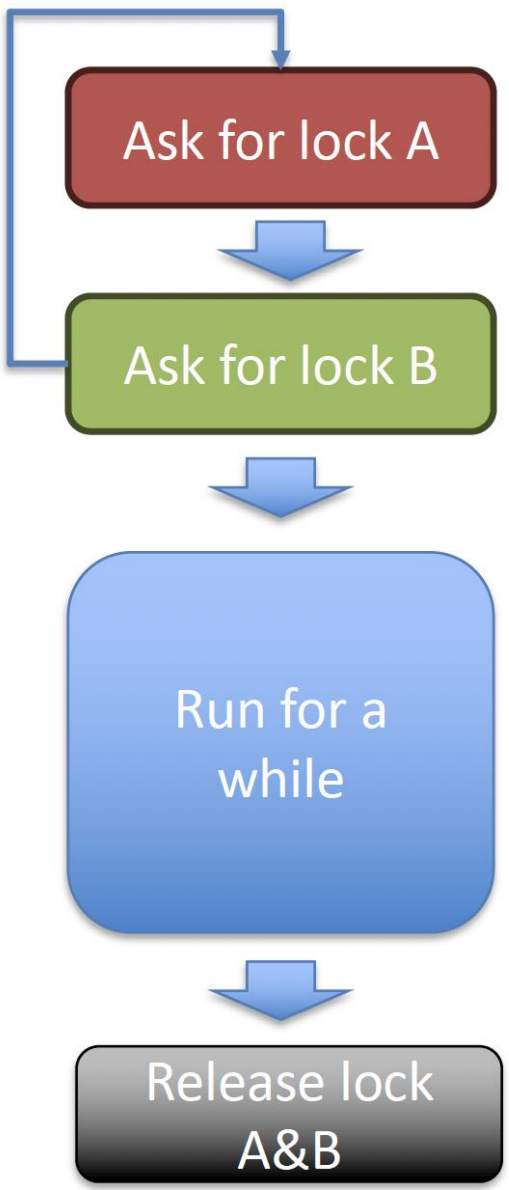
Without preventing Hold and Wait, we have a deadlock!

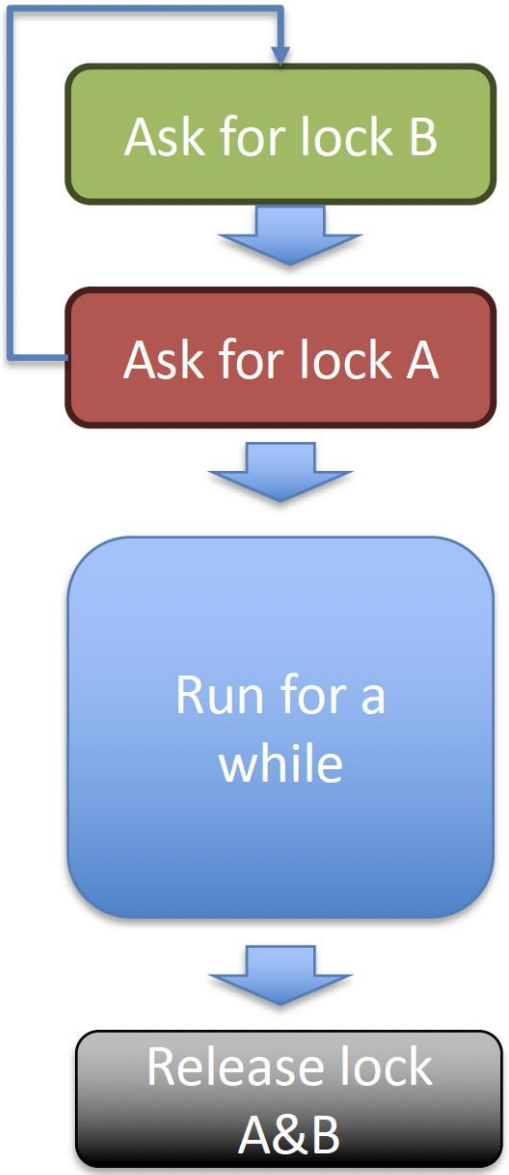


Deadlock Prevention: Hold and Wait

What it we can request for all locks ahead of time?

If lock request fails at any time, release all the locks!





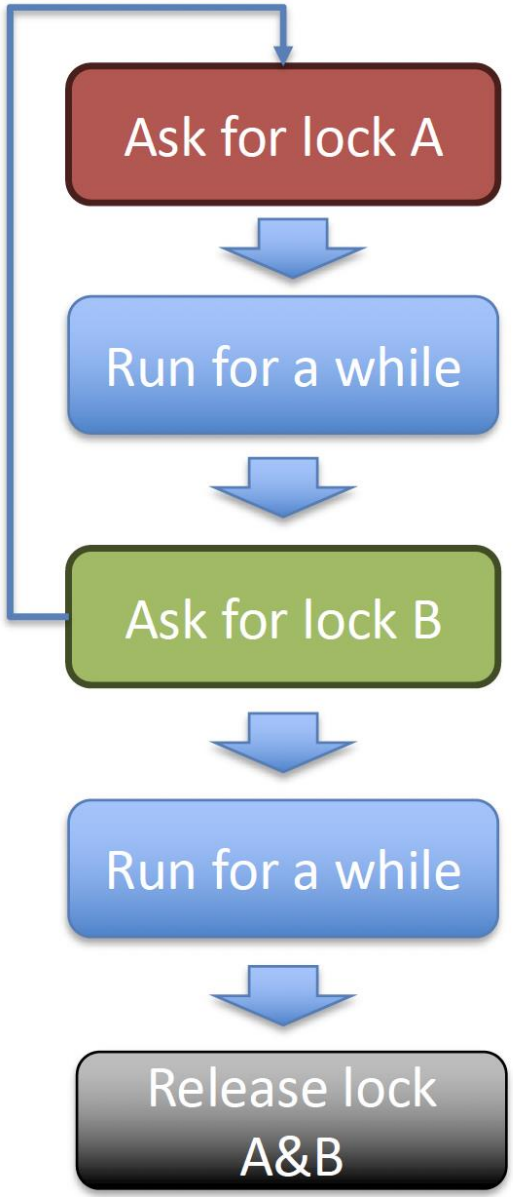
Deadlock Prevention: No Preemption

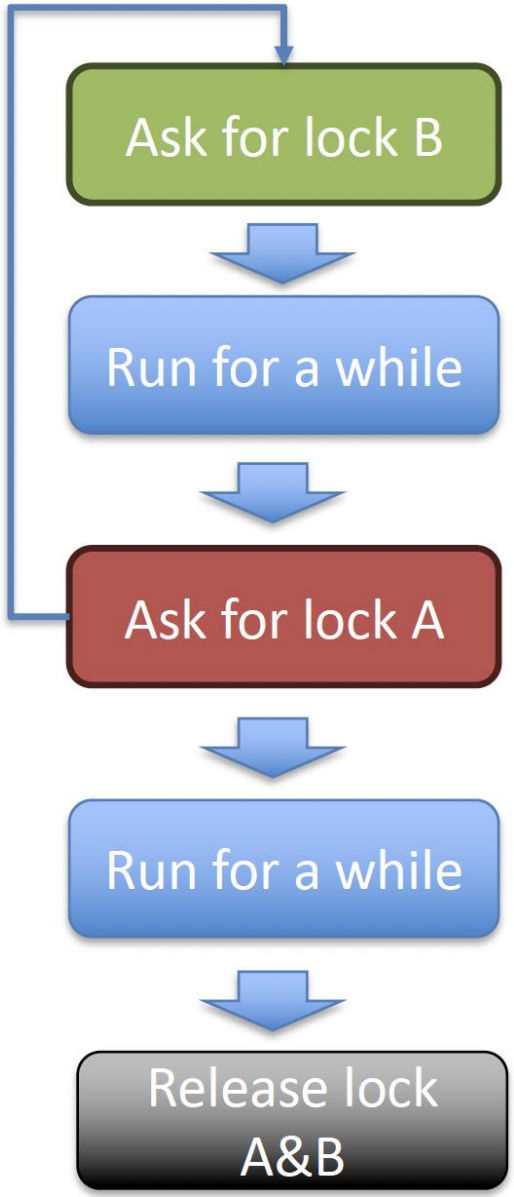
- Execute normally: lock → run → lock → execution

- If a lock is unavailable, instead of blocking and waiting → preempt all acquired resources.

Deadlock Prevention: No Preemption

If lock request fails at any time, release all the locks!





Deadlock Prevention: Circular wait

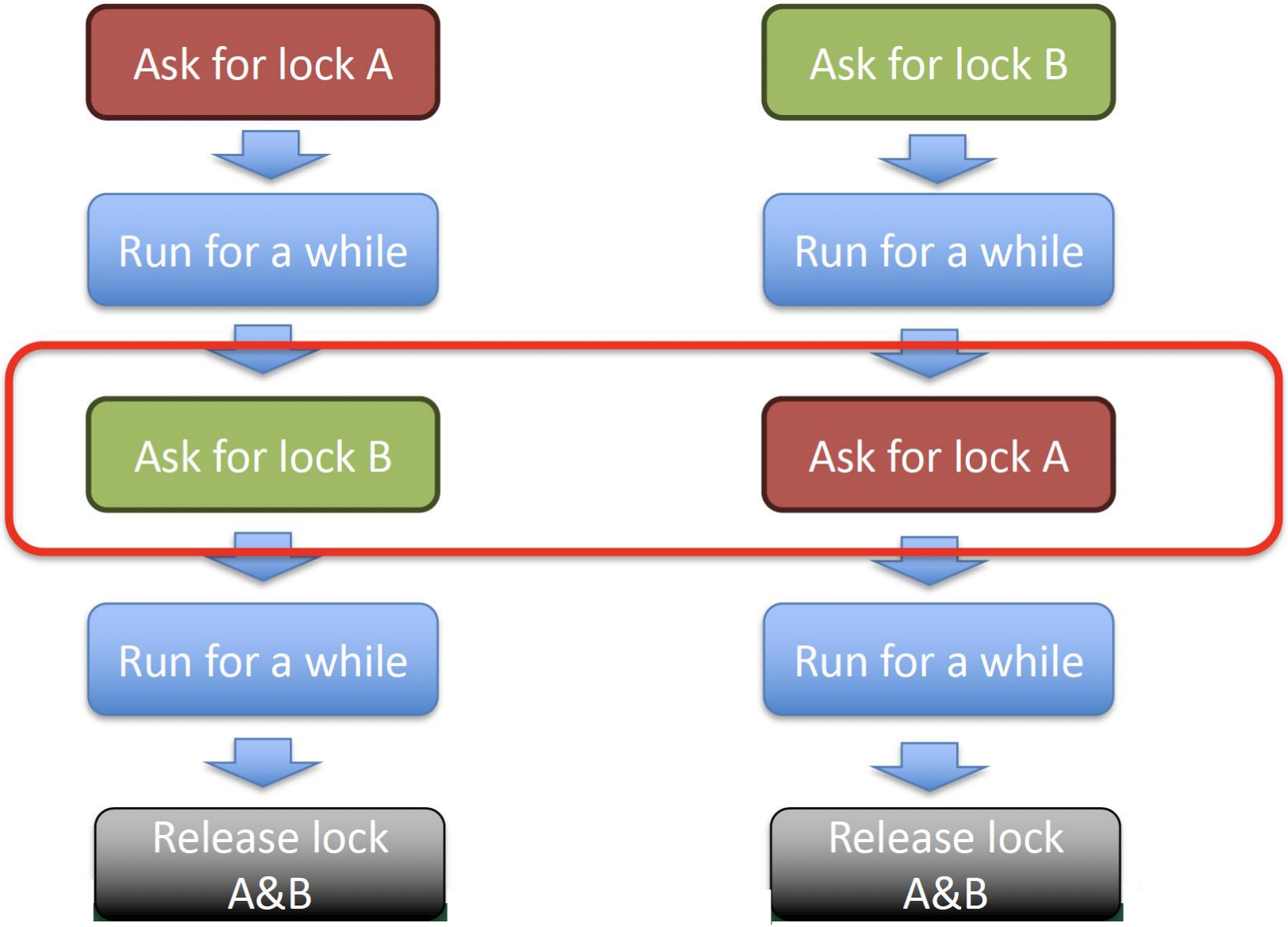
- Assign each resource with a unique integer number.

- Require that each process request resources in an increasing order.

Deadlock Prevention: Circular wait

Enforce the lock order.

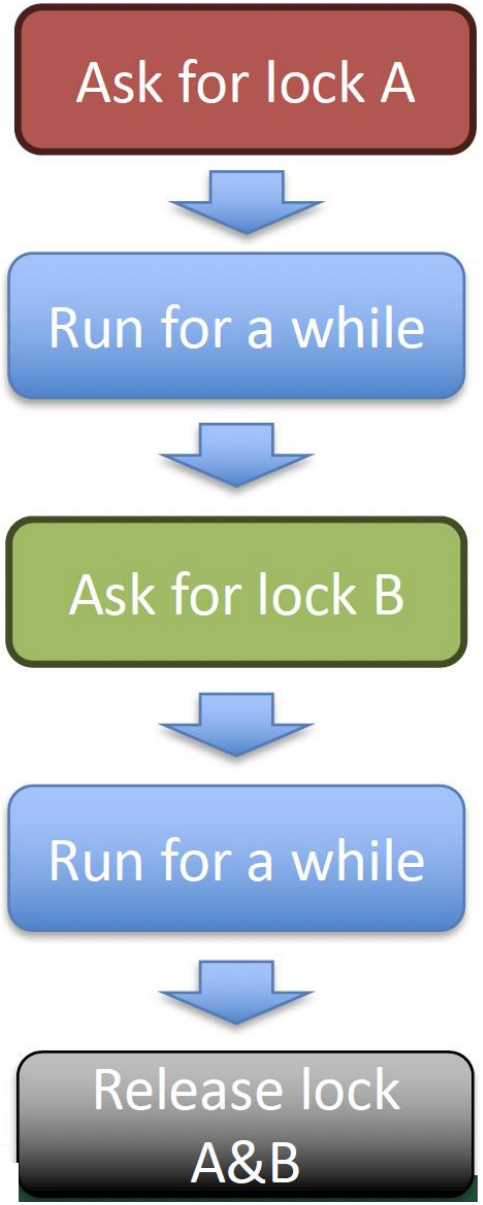
Lock A = 1
Lock B = 2

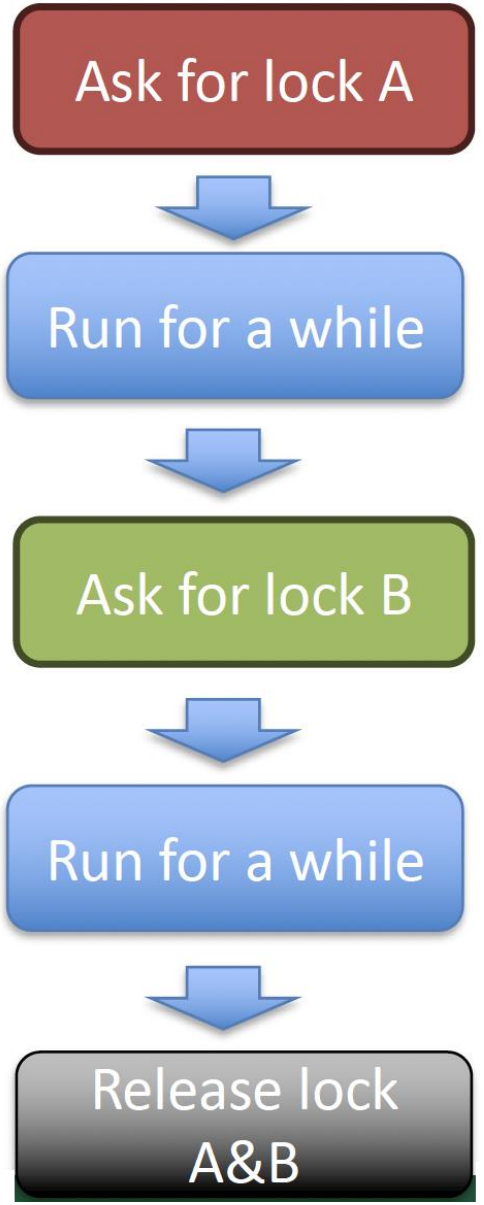


Deadlock Prevention: Circular wait

Enforce all process to request resources only in an increasing order.

Lock A = 1
Lock B = 2





Challenges

- Hold and Wait → May require re-ordering instructions!
- No Preempt → Wastes work due to restarting the process!
- Circular Wait → Requires application developers to write programs that follow the ordering.

Deadlock Avoidance

Deadlock Avoidance

- More proactive than deadlock prevention.
- Requires that the system has some additional **a priori** information available.
- The simplest and most useful information:
  - Each process declares the maximum number of resources of each type it needs.

Deadlock Avoidance Algorithm

- Dynamically examines the **resource allocation state** to ensure that there can never be a circular-wait condition.

- Resource allocation state is determined by:
  - number of available resource.
  - number of allocated resources.
  - the maximum possible demands of the processes.

Safe State

- System is in a safe state if there exists a safe sequence:
  - **< P1, P2, …,Pn >** of all the processes in the system,
  - such that for each **Pi**, the resources **Pi** needs can be satisfied by
  - currently available resources and the resources held by all **Pj** with **j < i.**

- A state is ***safe*** if the system can allocate resources to each thread (up to its maximum) in some order and still avoid a deadlock.

Safe State

- System is in a safe state if there exists a safe sequence:
  - **< P1, P2, …,Pn >** of all the processes in the system,
  - such that for each **Pi**, the resources **Pi** needs can be satisfied by
  - currently available resources and the resources held by all **Pj** with **j < i.**

- A state is ***safe*** if the system can allocate resources to each thread (up to its maximum) in some order and still avoid a deadlock.

- Deadlock avoidance is achieved by ensuring that we never enter an unsafe state.

Deadlock Avoidance Algorithm

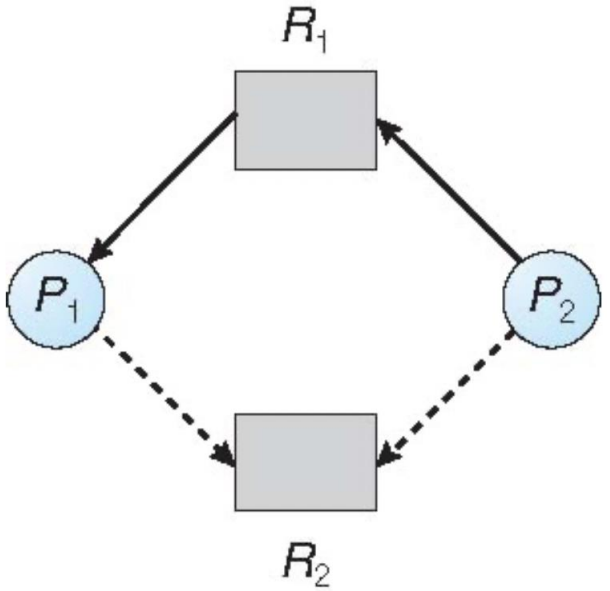
- Only one instance of each resource type:
  - Resourced Allocation Graph Algorithm

- Multiple instances of each resource type:
  - Bankers Algorithm

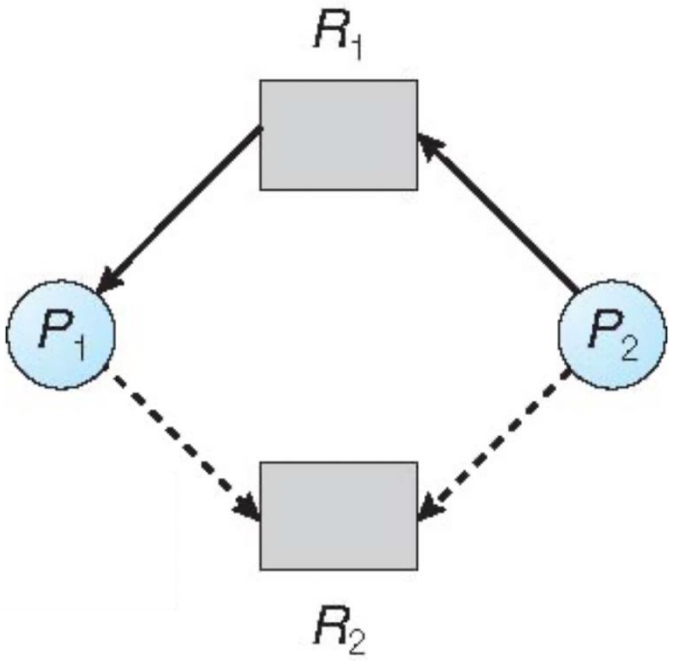
Resource Allocation Graph Algorithm

Resource Allocation Graph Algorithm

- Claim edge **Pi --> Rj** indicates that process **Pi** may request resource **Rj.**
  - Represented by a dashed line
- Claim edge converts to a **request edge** when a process requests a resource.
- Request edge converts to an assignment edge when the resource is allocated to the process.

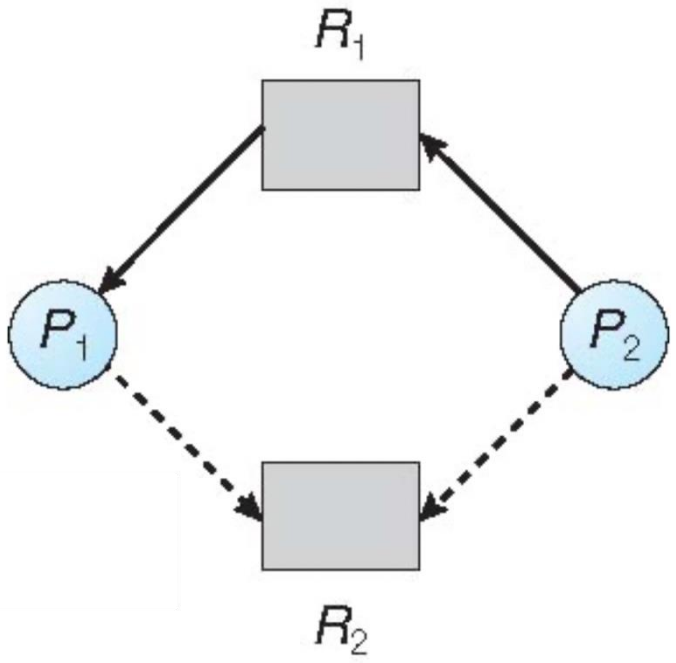


Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

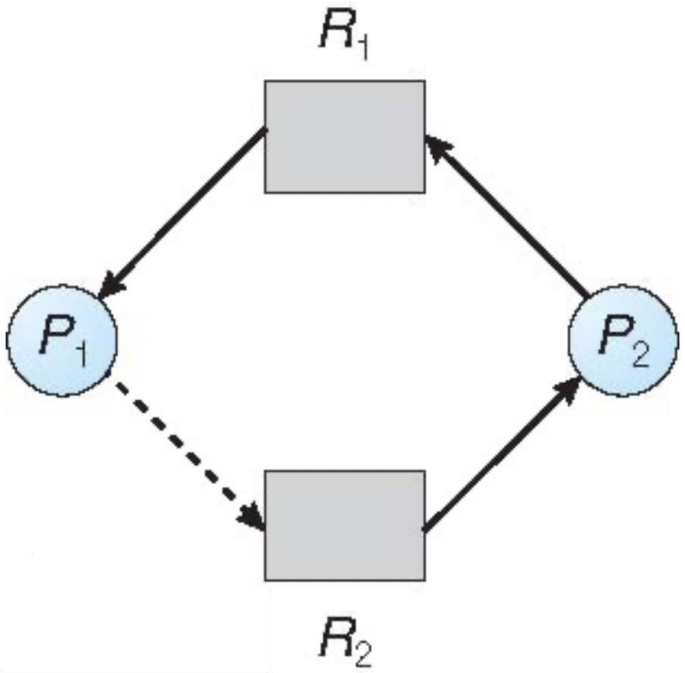
Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

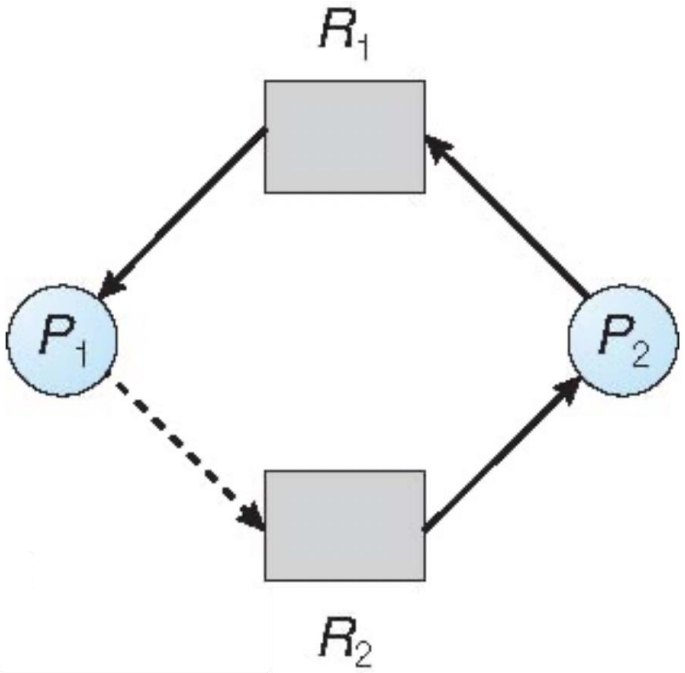
- Yes → [P1, P2]
- No → [P2, P1]

Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

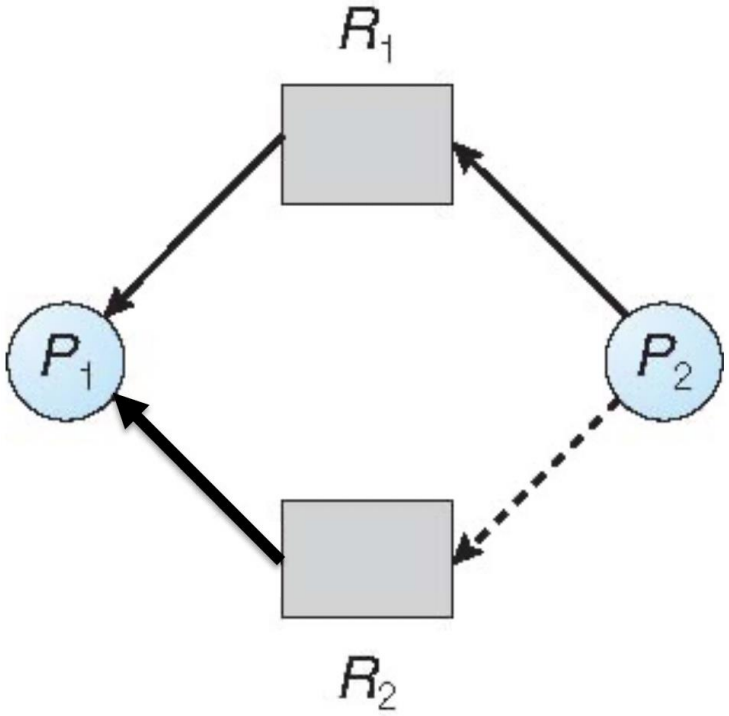
Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

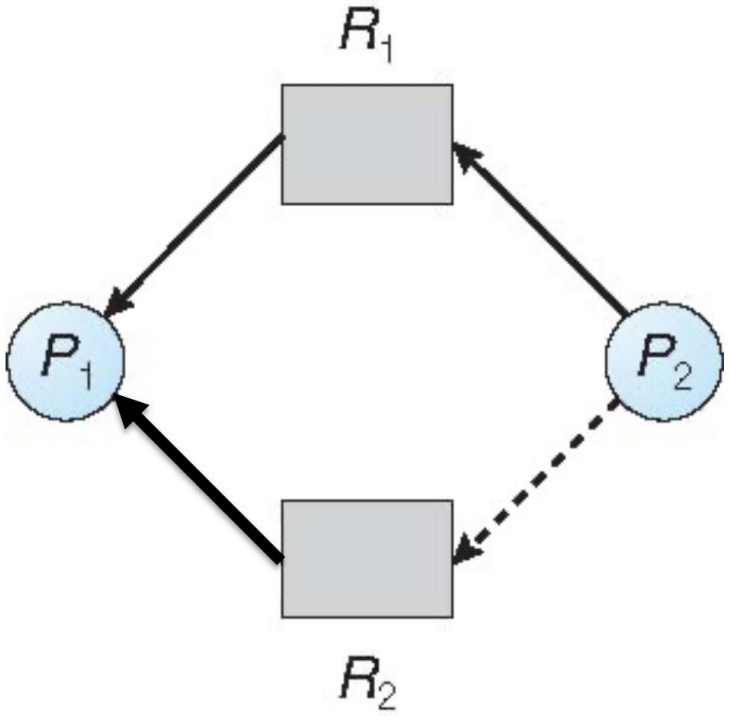
- No!

Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

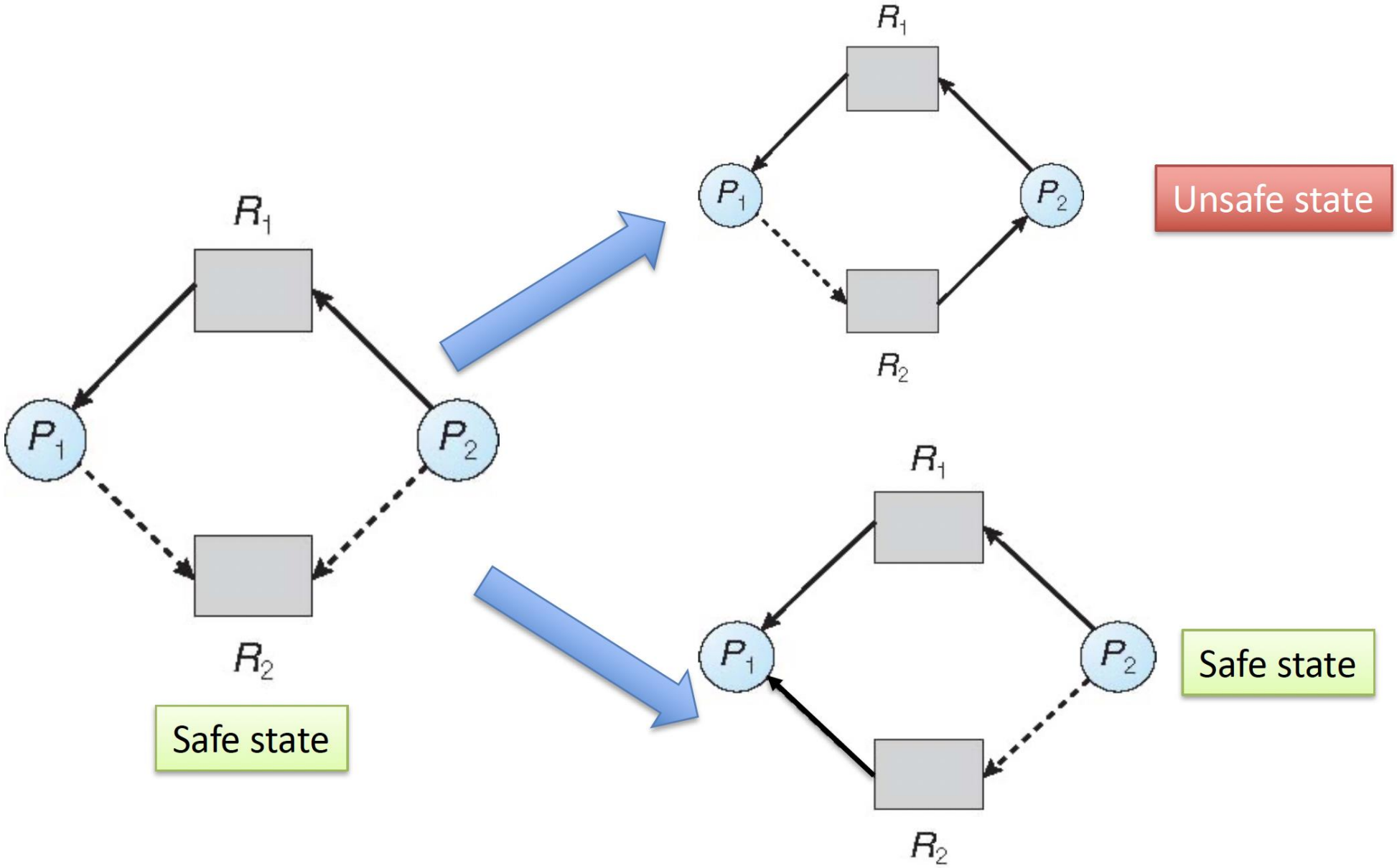
Resource Allocation Graph Algorithm



- Is this in a safe state? Can you find a sequence such that all processes can get the resources they need to finish?

- Yes → [P1, P2]

Resource Allocation Graph Algorithm



Resource Allocation Graph Algorithm

- Suppose that process Pi requests a resource Rj:
  - The request can be granted only if converting to an assignment edge does not result in a cycle in the resource allocation graph.
  - Cycles are evaluated using all types of edges, including claim edges.

Banker's Algorithm

Banker's Algorithm

- Banker's algorithms is a bookkeeping method for tracking and assigning resources.
- We have multiple instances of a resource.

Banker's Algorithm

- Let **n** = number of processes, and **m** = number of resources types.
- **Available**: Vector of length **m**. If **Available[j] = k**, there are **k** instances of resource type **Rj** available.
- **Max**: n x m matrix. If **Max[i,j] = k**, then process **Pi** may request at most **k** instances of resource type **Rj**.
- **Allocation**: n x m matrix. If **Allocation[i,j] = k,** then **Pi** is currently allocated **k** instances of **Rj**.
- **Need**: n x m matrix. If **Need[i,j] = k**, then **Pi** needs k more instances of **Rj** to finish.
  - **Need [i,j] = Max[i,j] – Allocation [i,j]**
- **Request**: n x m matrix. If **Request[i,j] = k**, then Pi requests **k** more instances of **Rj**.

Banker's Algorithm: Safety Check

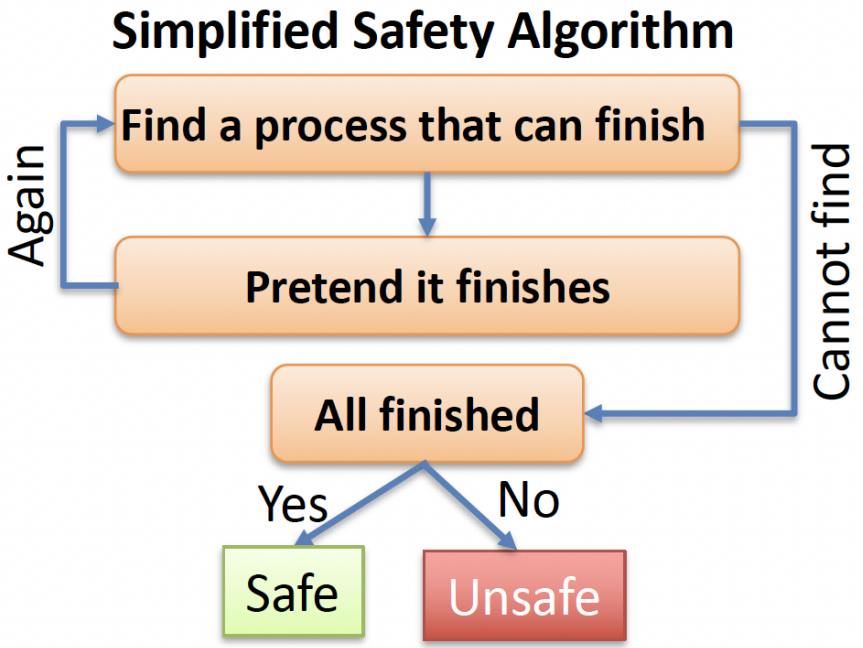
- 5 processes P0 through P4.
- 3 resource types:
  - A (10 instances), B (5 instances), C (7 instances)
- Snapshot at time Tk → Is this safe?

Process	*Allocation*	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
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	

Banker's Algorithm: Safety Check

Process	*Allocation*	*Max*	*Work*	*Need*	Finish
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	7 5 3	3 3 2	7 4 3	*No*
P_1	2 0 0	3 2 2		1 2 2	*No*
P_2	3 0 2	9 0 2		6 0 0	*No*
P_3	2 1 1	2 2 2		0 1 1	*No*
P_4	0 0 2	4 3 3		4 3 1	*No*

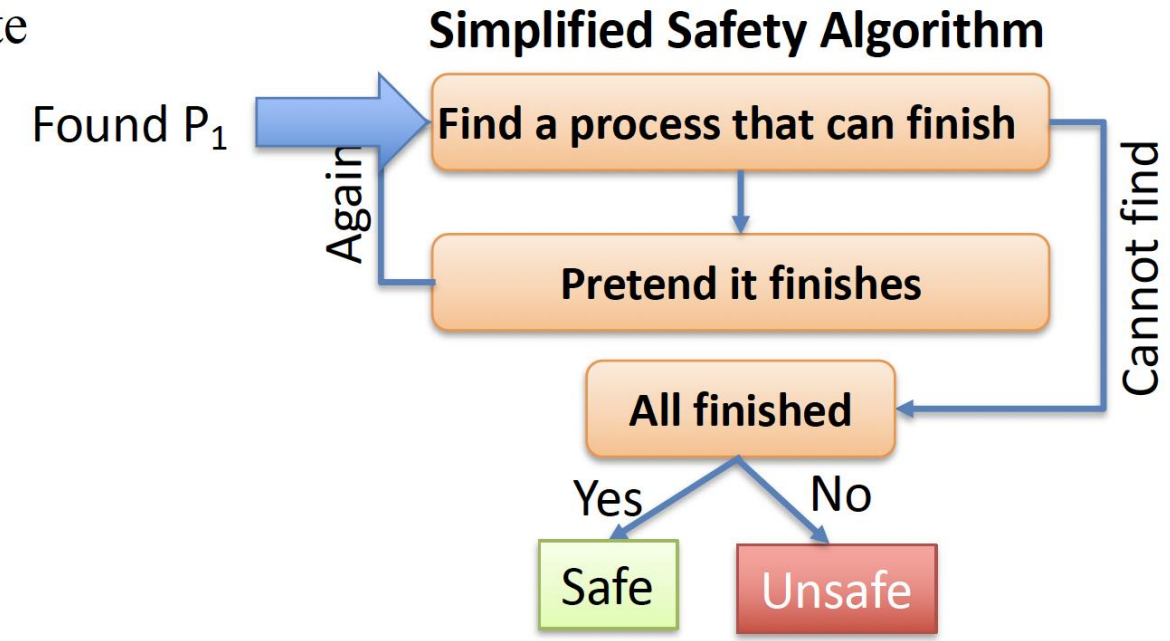
Find a sequence for processes to complete (one after the other)



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	3 3 2	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	No
P_2	3 0 2	9 0 2		6 0 0	No
P_3	2 1 1	2 2 2		0 1 1	No
P_4	0 0 2	4 3 3		4 3 1	No

Find a sequence for processes to complete (one after the other)

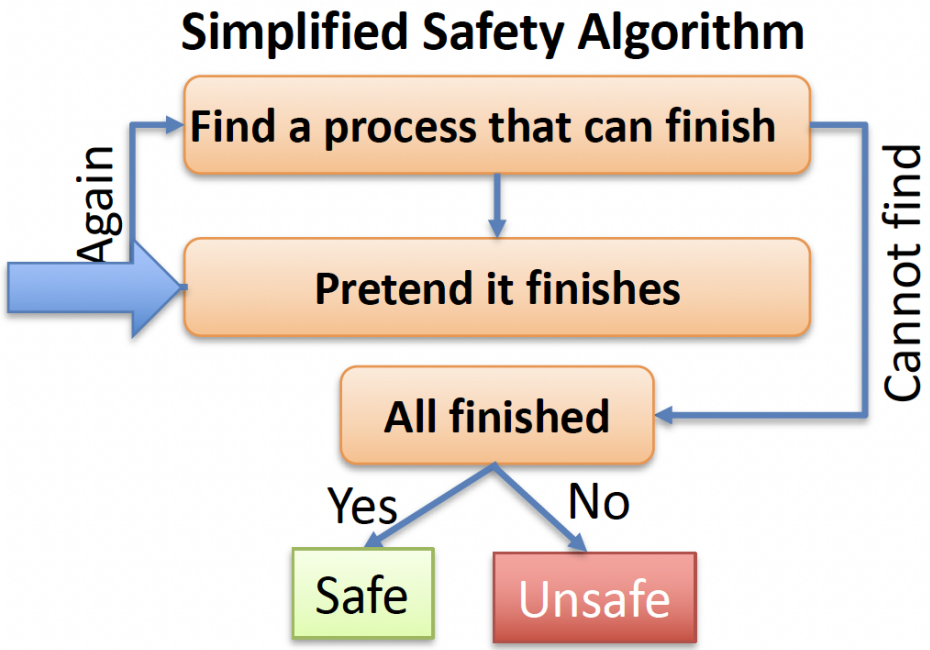


Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	5 3 2	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	No
P_3	2 1 1	2 2 2		0 1 1	No
P_4	0 0 2	4 3 3		4 3 1	No

Find a sequence for processes to complete (one after the other): P_1

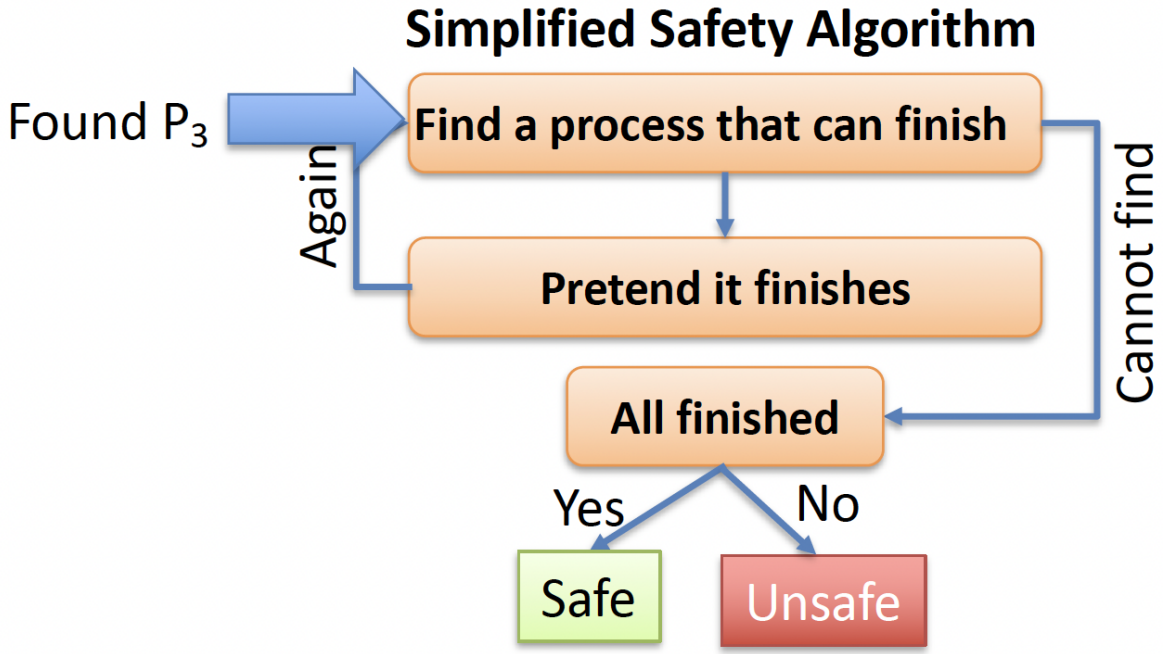
Pretend P_1 finishes



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	5 3 2	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	No
P_3	2 1 1	2 2 2		0 1 1	No
P_4	0 0 2	4 3 3		4 3 1	No

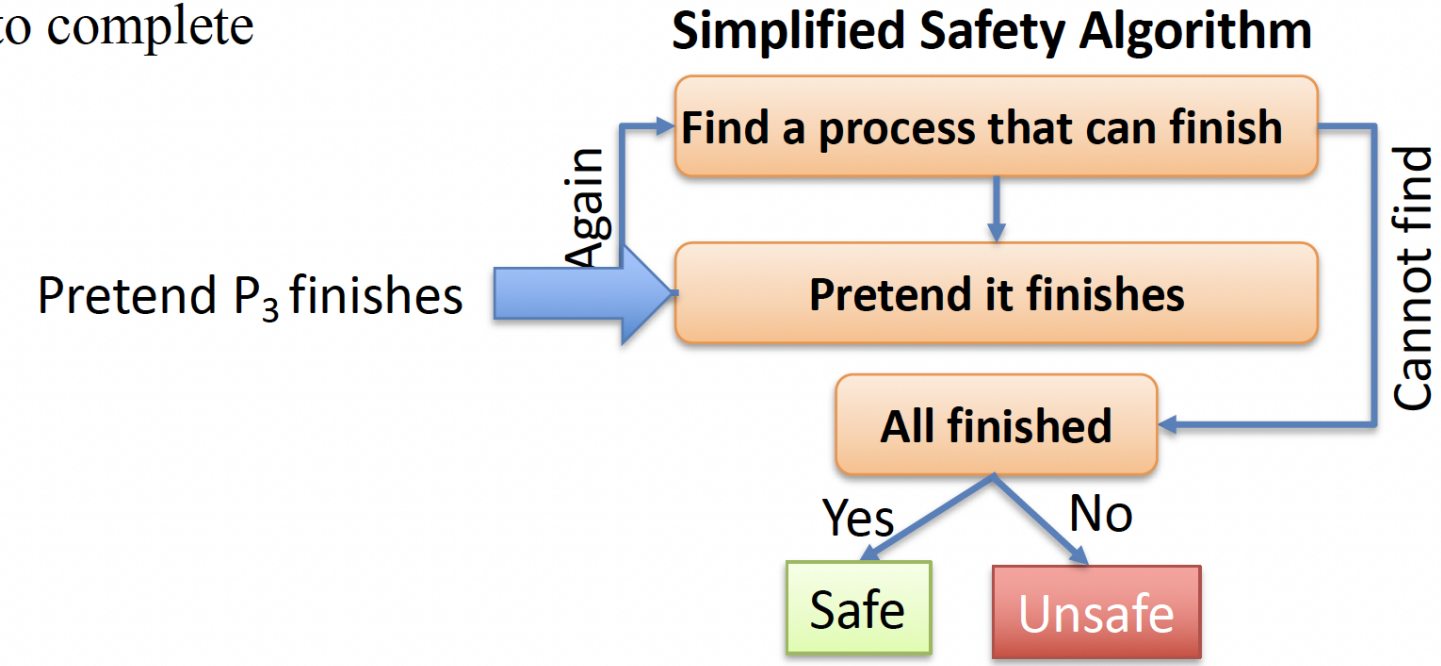
Find a sequence for processes to complete (one after the other): P_1



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	7 4 3	7 4 3	*No*
P_1	2 0 0	3 2 2		1 2 2	*Yes*
P_2	3 0 2	9 0 2		6 0 0	*No*
P_3	2 1 1	2 2 2		0 1 1	*Yes*
P_4	0 0 2	4 3 3		4 3 1	*No*

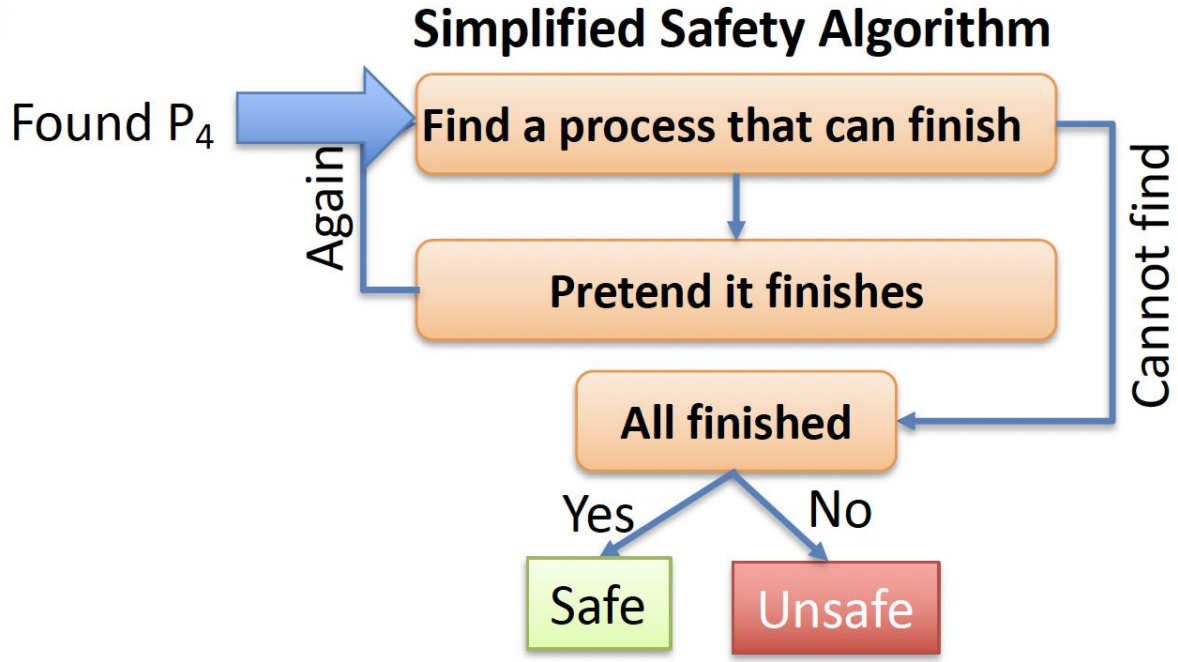
Find a sequence for processes to complete (one after the other): P_1 P_3



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	7 4 3	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	No
P_3	2 1 1	2 2 2		0 1 1	Yes
P_4	0 0 2	4 3 3		4 3 1	No

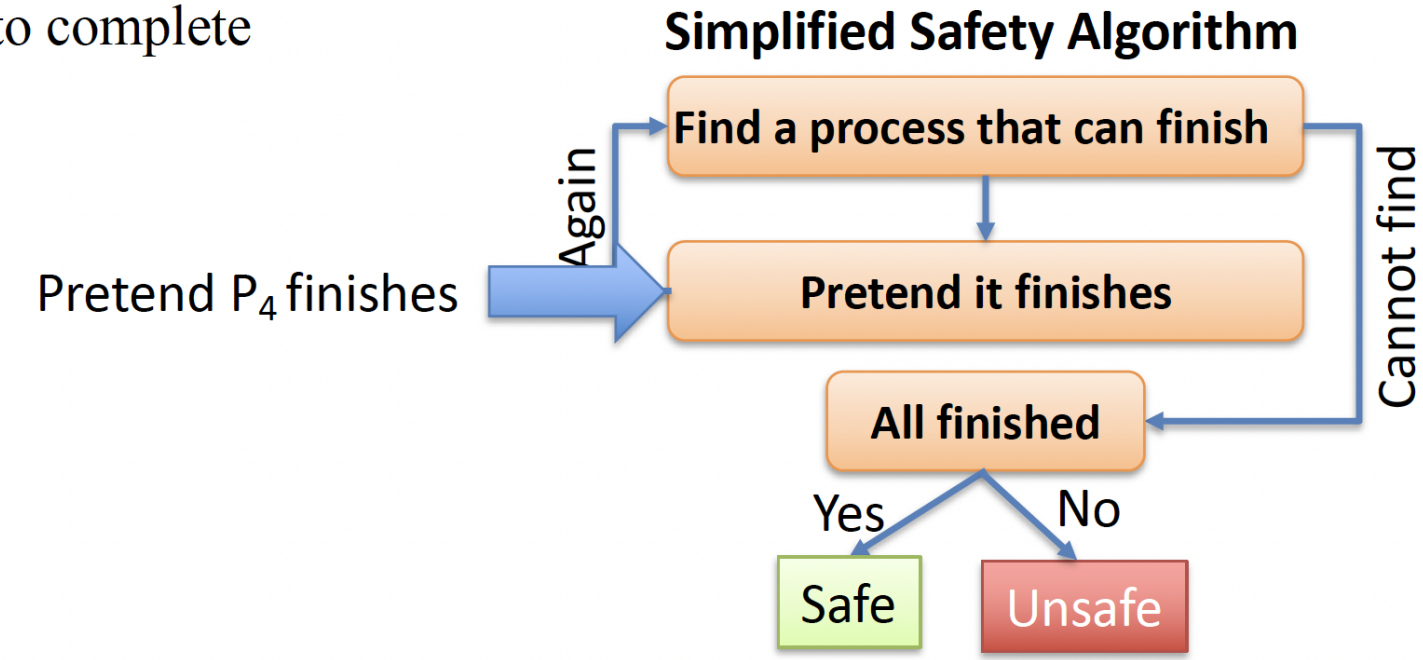
Find a sequence for processes to complete (one after the other): P_1 P_3



Banker's Algorithm: Safety Check

Process	*Allocation*	*Max*	*Work*	*Need*	Finish
	A B C	*A B C*	*A B C*	*A B C*	
P_0	0 1 0	7 5 3	7 4 5	7 4 3	*No*
P_1	2 0 0	3 2 2		1 2 2	*Yes*
P_2	3 0 2	9 0 2		6 0 0	*No*
P_3	2 1 1	2 2 2		0 1 1	*Yes*
P_4	0 0 2	4 3 3		4 3 1	*Yes*

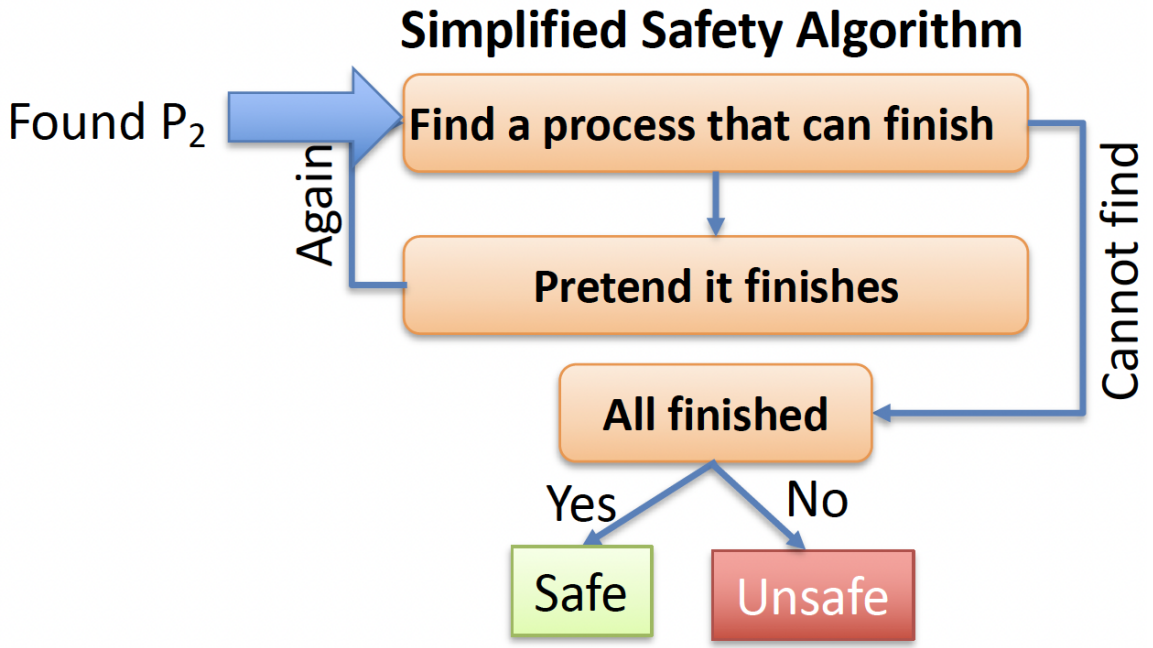
Find a sequence for processes to complete (one after the other): P_1 P_3 P_4



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	7 4 5	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	No
P_3	2 1 1	2 2 2		0 1 1	Yes
P_4	0 0 2	4 3 3		4 3 1	Yes

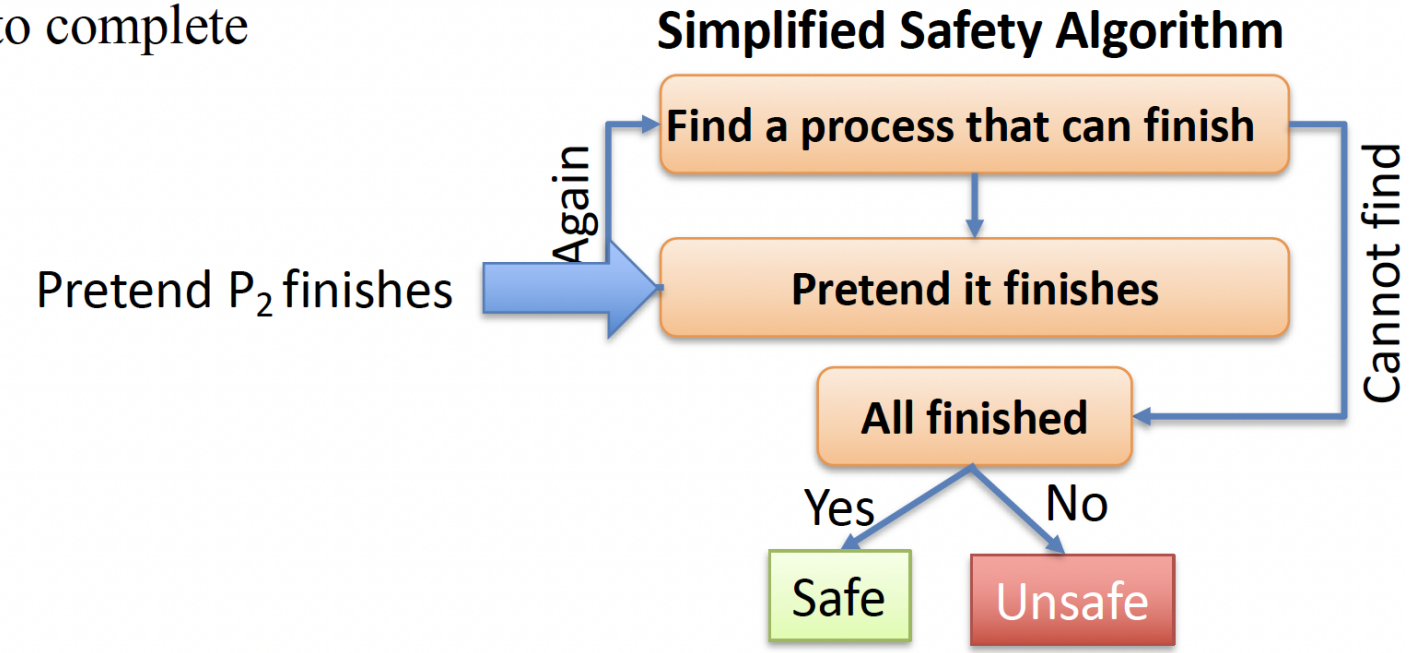
Find a sequence for processes to complete (one after the other): P_1 P_3 P_4



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	10 4 7	7 4 3	*No*
P_1	2 0 0	3 2 2		1 2 2	*Yes*
P_2	3 0 2	9 0 2		6 0 0	*Yes*
P_3	2 1 1	2 2 2		0 1 1	*Yes*
P_4	0 0 2	4 3 3		4 3 1	*Yes*

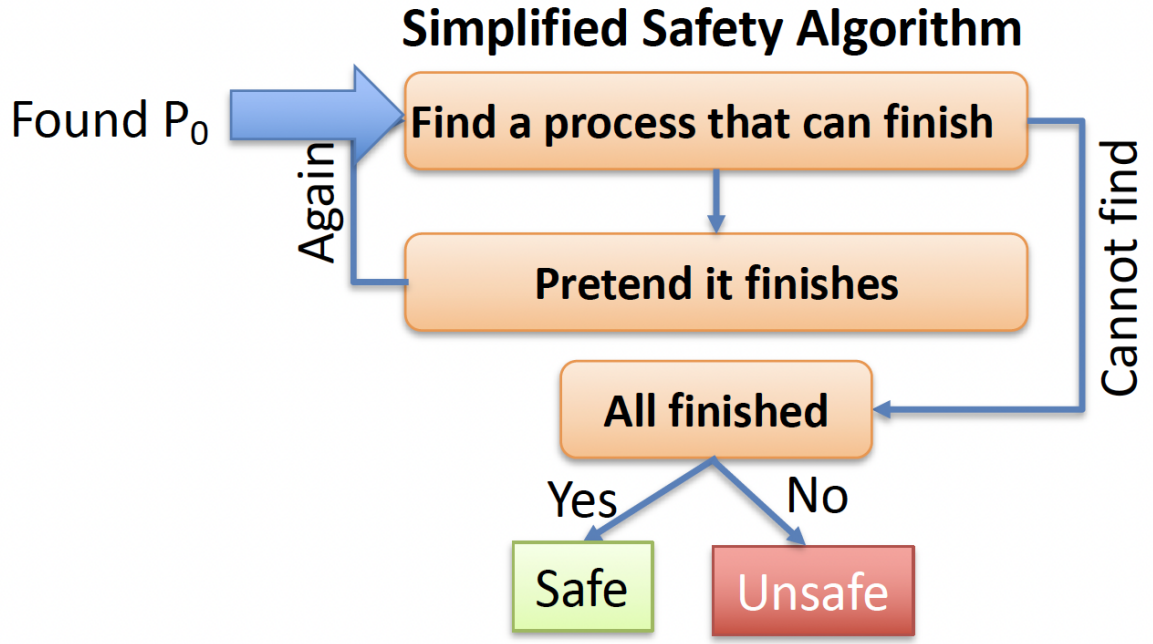
Find a sequence for processes to complete (one after the other): $P_1\ P_3\ P_4\ P_2$



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	10 4 7 ←	7 4 3	No
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	Yes
P_3	2 1 1	2 2 2		0 1 1	Yes
P_4	0 0 2	4 3 3		4 3 1	Yes

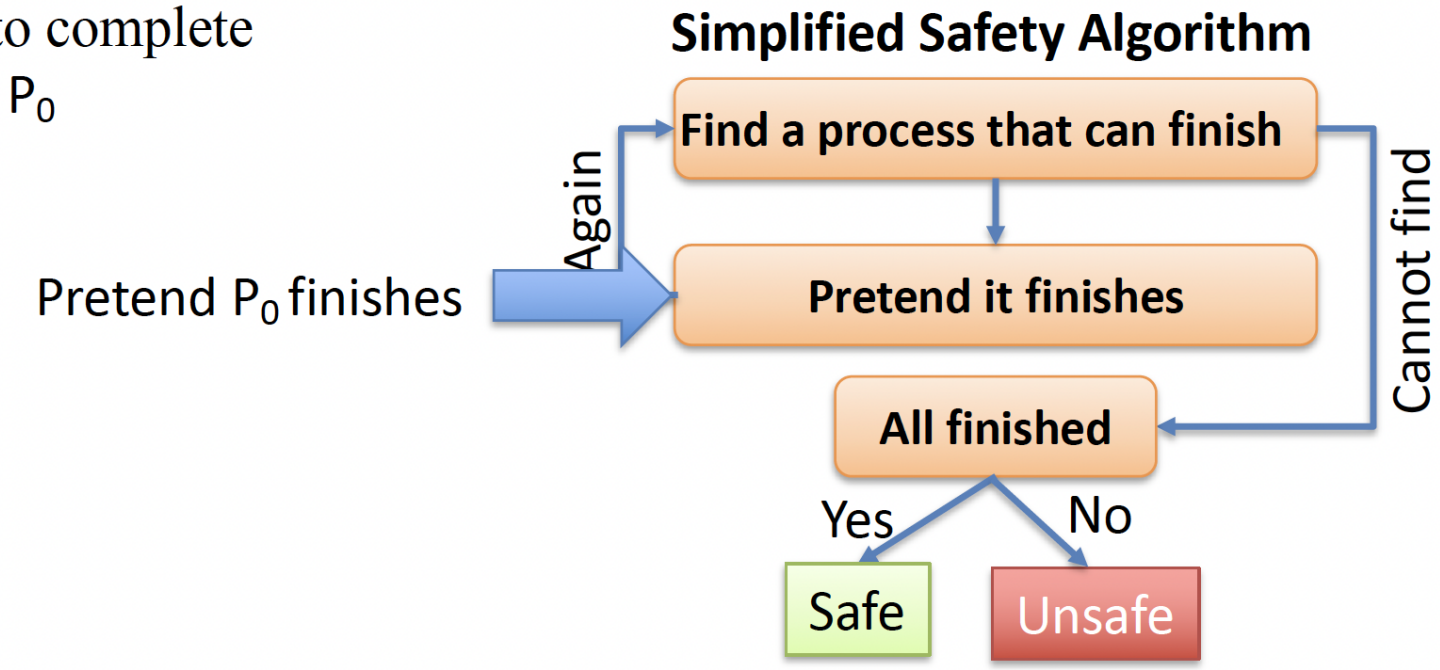
Find a sequence for processes to complete (one after the other): P_1 P_3 P_4 P_2



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	10 5 7	7 4 3	Yes
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	Yes
P_3	2 1 1	2 2 2		0 1 1	Yes
P_4	0 0 2	4 3 3		4 3 1	Yes

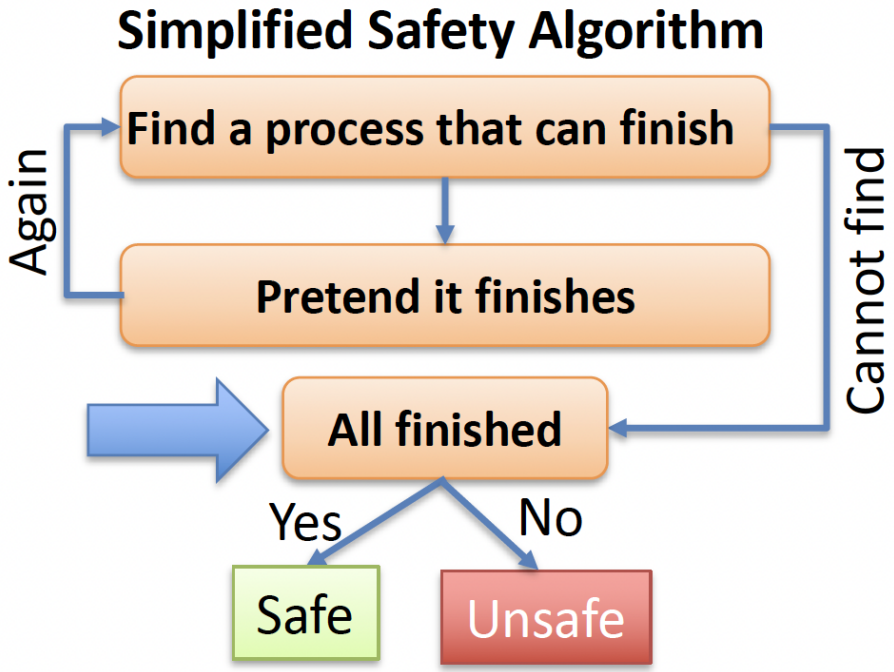
Find a sequence for processes to complete (one after the other): $P_1\ P_3\ P_4\ P_2\ P_0$



Banker's Algorithm: Safety Check

Process	Allocation	Max	Work	Need	Finish
	A B C	A B C	A B C	A B C	
P_0	0 1 0	7 5 3	10 5 7	7 4 3	Yes
P_1	2 0 0	3 2 2		1 2 2	Yes
P_2	3 0 2	9 0 2		6 0 0	Yes
P_3	2 1 1	2 2 2		0 1 1	Yes
P_4	0 0 2	4 3 3		4 3 1	Yes

Find a sequence for processes to complete (one after the other): $P_1\ P_3\ P_4\ P_2\ P_0$



Banker's Algorithm: Safety Check

- Is the safe sequence unique?

- What is the runtime of Banker's algorithm?

Banker's Algorithm: Safety Check

- Is the safe sequence unique?
  - No!

- What is the runtime of Banker's algorithm?
  - $O(m \times n^2)$

Limitations of Banker's Algorithm

- Prior knowledge of resources.
  - Processes must declare their maximum resource needs in advance.
- Fixed number of processes.
  - The algorithm assumes the total number of active processes remains constant.
- Fixed number of resources.
  - The total pool of available resources must remain static.

Deadlock Detection and Recovery

Deadlock Detection and Recovery

- Suppose we allow system to enter a deadlock state (if any).

- Then we need a deadlock detection algorithm.
  - Single instance of each resource type → Wait-for Graph.
  - Multiple instances of each resource type → Modified Banker's Safety Algorithm.

- Finally, we need to perform Deadlock recovery.

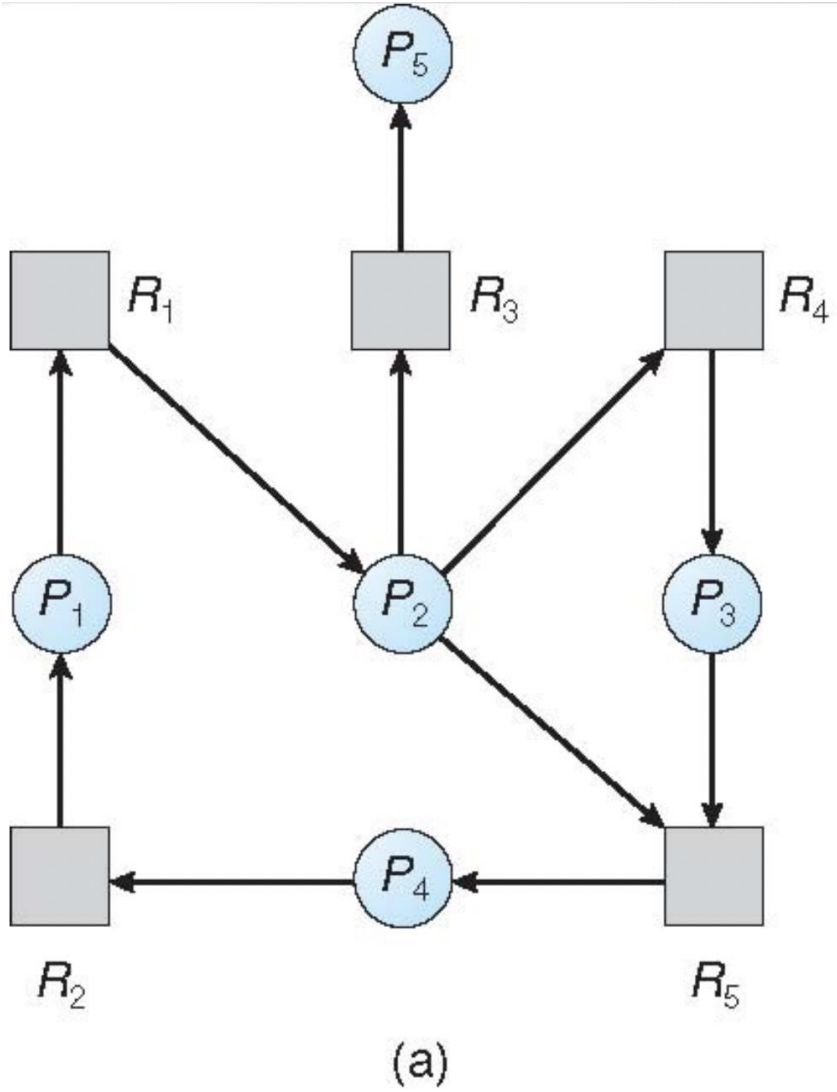
Wait-for-Graph

- Nodes are processes.
  - Have an edge **Pi → Pj** if **Pi** is waiting for **Pj**.

- Periodically invoke an algorithm that searches for a cycle in the wait-for graph.
  - If there is a cycle, there exists a deadlock

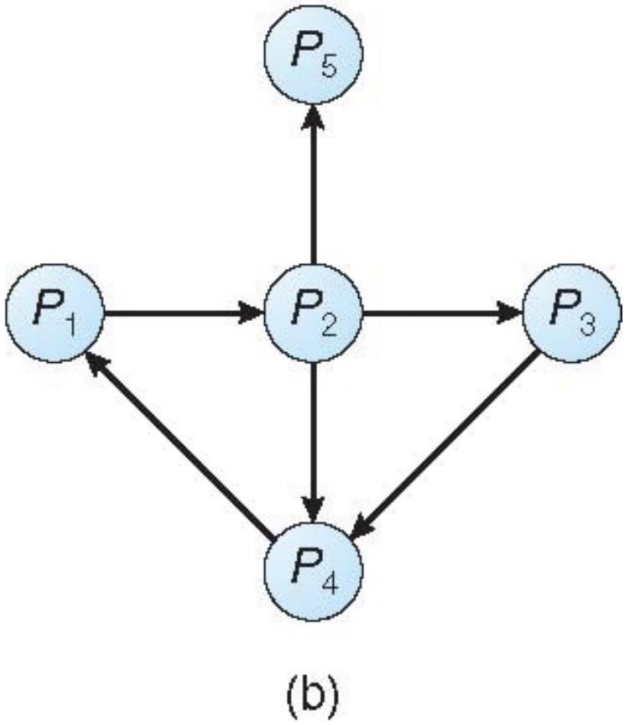
- An algorithm to detect a cycle in a graph requires an order of $\mathbf{n^2}$ operations, where **n** is the number of vertices in the graph.

Wait-for-Graph



Resource-Allocation Graph

If there is a single instance of every resource, then can just draw the processes



Corresponding wait-for graph

Deadlock Recovery

- Now that you have discovered a deadlock, you should recover the system to allow progress.

Deadlock Recovery

- Now that you have discovered a deadlock, you should recover the system to allow progress.

- Two methods:
  - Process Termination
  - Resource Preemption

Process Termination

- Aborting one or more processes will release their resources.

- Different designs:
  - Abort all deadlocked processes
  - Abort one process at a time until the deadlock cycle is eliminated.

- In which order should we choose to abort?
  - Priority of the process
  - How long process has computed, and how much longer to completion.
  - Resources the process has used
  - Resources process needs to complete
  - How many processes will need to be terminated.

Resource Preemption

- Successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

- **Selecting a victim**
  - Which resources and which processes are to be preempted?
  - Cost factors: the number of resources a deadlocked process is holding and the amount of time the process has thus far consumed.

- **Rollback**
  - If we preempt a resource from a process, what should be done with that process.
  - We must roll back the process to some safe state and restart it from that state.

- **Starvation**
  - How do we ensure that starvation will not occur?