Lecture 21

Deadlocks

ECE 422: Reliable and Secure Systems Design

UNIVERSITY OF ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

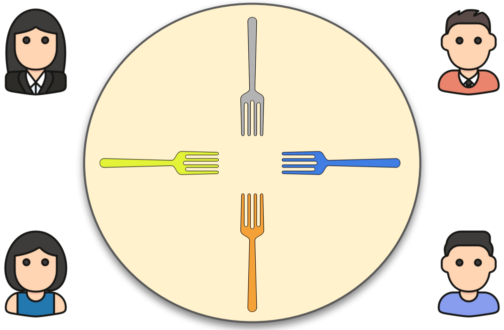
Schedule for today

- Key concepts from last classes
- Race condition on reliability
- Deadlock avoidance
  - Resource allocation graph
  - Resource allocation table
  - Resource availability

The Dining Philosophers Problem

Dining Philosophers Problem is a classical synchronization problem in the operating system.

- Philosophers can either think or eat
- To eat, philosophers needs both their left and right forks
  - Two forks will only be available when the two nearest neighbors are thinking, not eating
  - If not available, they start thinking
  - After they are done eating, they will put down both forks
- To think, philosophers does nothing but thinking



Goal: Designing a solution (algorithm) so that no philosopher starves.

Example of race condition

P1 decides to eat

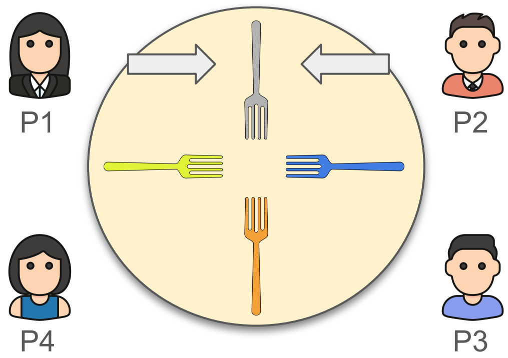
- P1 checks if the grey fork is available
- It is available, therefore P1 will try to grab the grey fork

At the same time, P2 decides to eat

- P2 checks if the grey fork is available
- Because the check happens before P1 actually grabs the grey fork, the fork will also be available to P2

Race condition detected!

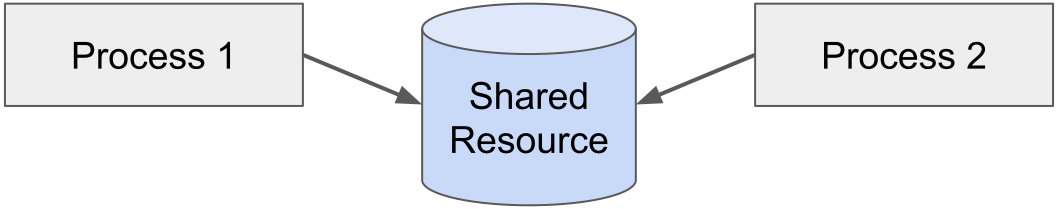
- P2 cannot grab the fork as P1 will grab the fork before



Race condition

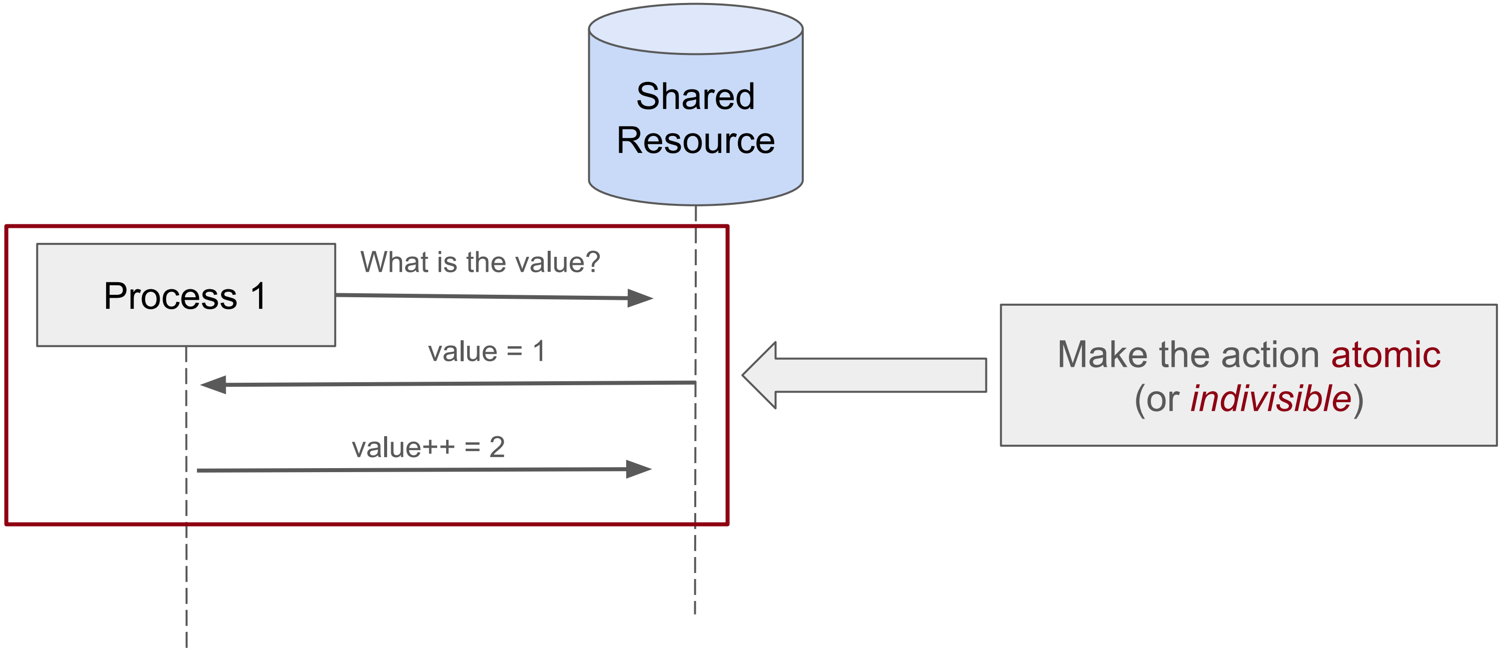
A race condition happens when two processes attempt to access the same resource at the same time. However, the timing or ordering of events affects a program's correctness.

- E.g., Two processes that attempt to increase a shared value by 1
- Instead of increasing the value twice, the value is only increased by 1 where the last modification is preserved



Avoiding race conditions

We must guarantee that no process can intervene during another process' manipulation that involves shared resources.



Race condition on reliability

Detecting and identifying race condition in practice is difficult but important.

- With race condition vulnerabilities, the system may suffer from denial-of-service attack

For example:

- Hackers could ask the program to execute multiple operations concurrently
- If a deadlock occurs, then the system risks of stop responding and eventually crashing

Example of reliability issue: Northeast Blackout of 2003

- Power outage throughout Northeastern United States and most parts of Ontario
    - 55 million people affected
    - Duration: 2 hours–4 days, depending on location
    - At the time, it was the world’s second most widespread blackout in history
- Race condition in the energy management system
    - Some unique combination of events and alarm conditions triggered the race condition
    - Because of the race condition, the system was sending an system alarm event into an infinite loop
    - Within thirty minutes, the server went down as there was too much events
    - Then, the backup server also went down

Example of deadlock

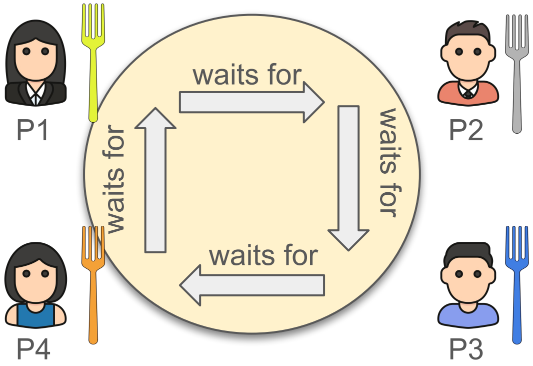
Step 1: Each philosopher grabs their right fork at the same time

- P1 grabs the yellow fork; P2 grabs the grey fork; …

Step 2: Each philosopher grabs their left fork at the same time

- P1 tries to grab the grey fork which is taken, so P1 starts to wait (think);
- P2 tries to grab the blue fork which is taken, so P2 starts to wait (think);
- …

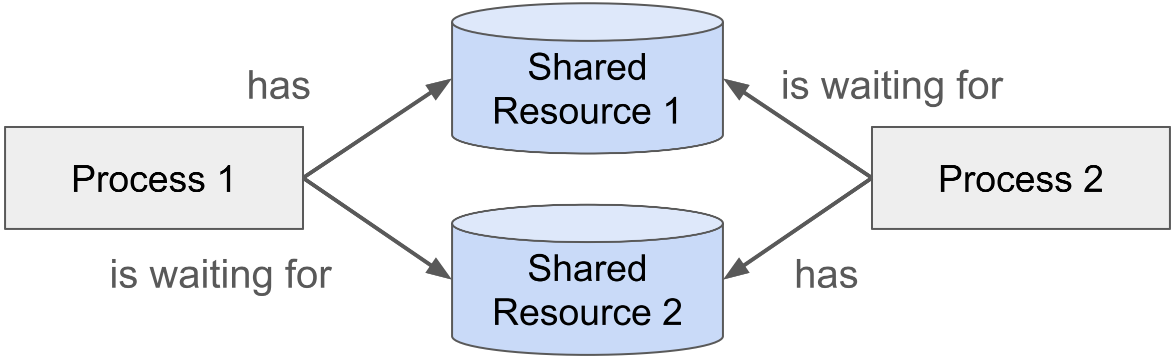
All philosophers hold a fork but are unable to eat.
They will eventually all starve.



Deadlock

Deadlock describes a situation in which two processes, sharing the same resources, are preventing each other from accessing the resources.

- Each process is holding the resource the other is waiting for
- But no one is releasing the resource until the other releases it



Deadlock Prevention

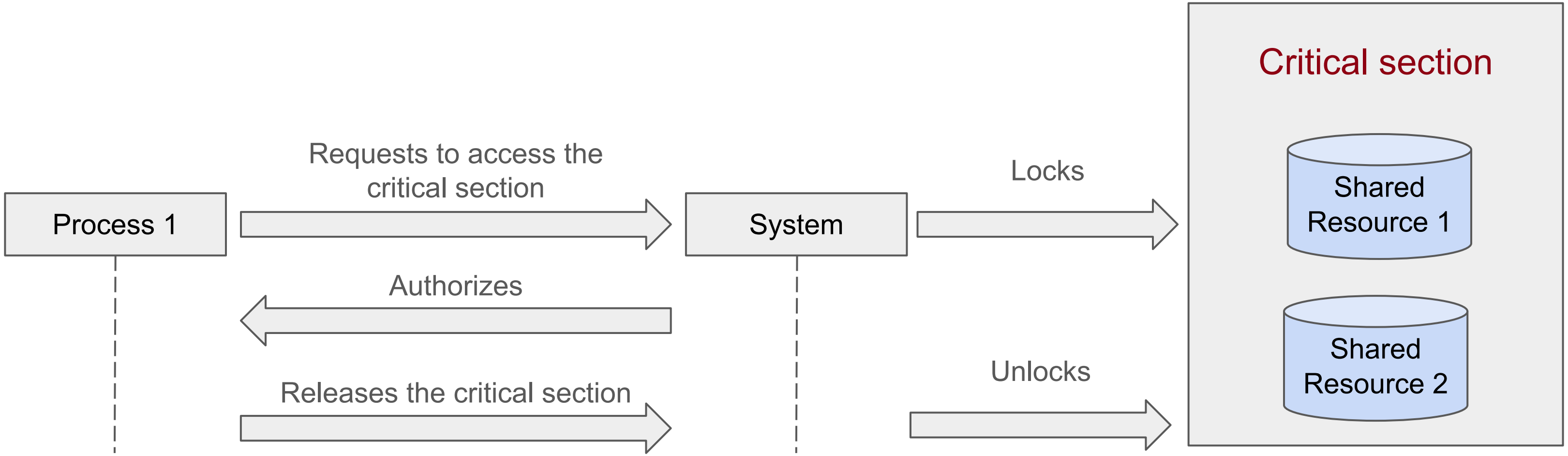
To prevent deadlock, we only need to ensure one of the conditions does not hold:

- Mutual Exclusion
  - Make the resource sharable
  - However, not all resources are sharable, e.g., editing files, analogous to forks
- Hold and Wait
  - Make processes request all resources at once and make them release all resources once done
- No Preemption
  - Make processes release all resources if the request cannot be proceed immediately
  - However, not all resources can be easily preempted, e.g., printing jobs
- Circular Wait
  - Impose an ordering on the resources and processes can only request them in order
  - However, waste of resources for such turn-based solution

Solution: atomic locking

Introducing atomic locks to the resources:

The atomic property guarantees that the resource remains locked until Process 1 finishes all its execution.



Schedule for today

- Key concepts from last classes
- Race condition on reliability
- Deadlock avoidance
  - Resource allocation graph
  - Resource allocation table
  - Resource availability

Deadlock avoidance

To avoid deadlock, we can examine the processes and resources to guarantee there can never be a circular-wait condition.

- System has access to information in advance about what resources will be needed by which processes
- System only approves resource requests if the process can obtain all resources it needs in future requests

However, it is tough to avoid deadlock in practice

- Hard to determine all the needed resources in advance
- Good problem statement in theory, but difficult to apply in practice

Deadlock avoidance

Deadlock avoidance requires that the system knows a priori information about the requests and needed resources.

There are two methods for deadlock avoidance:

- Resource Allocation Graph (RAG)
- Banker's Algorithm

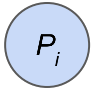
Resource Allocation Graph (RAG)

Resource Allocation Graph (RAG) represents the state of the system as graphs. The graph contain a set of vertices and edges to describe the state of the system:

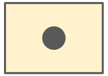
- Vertices (nodes) can either represent a process or resource
  - Process: $P = \{P_1, P_2, \ldots, P_n\}$ is the set of processes in the system
  - Resource: $R = \{R_1, R_2, \ldots, R_m\}$ is the set of resources in the system
- Edges can either represent a request or assignment
  - Request edge (directed edge): $P_i \rightarrow R_j$
  - Assignment edge (directed edge): $R_j \rightarrow P_i$

Resource Allocation Graph

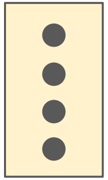
- P_i, Process



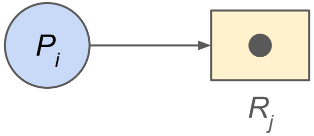
- R_j, Resource with instances
    - Resource with a single instance



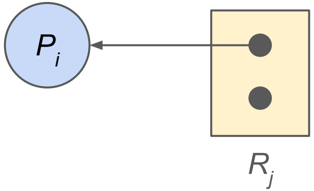
    - Resource with 4 instances



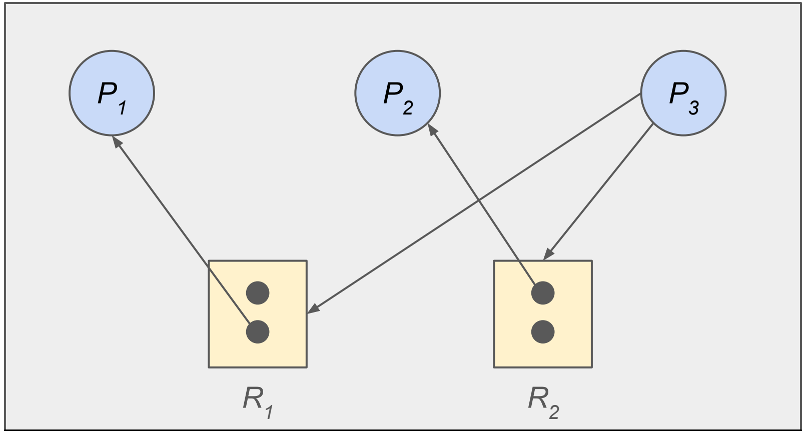
- P_i requests instance of R_j



- P_i is holding an instance of R_j



Example of Resource Allocation Graph



Vertices

- P_i, Process
- R_j, Resource with instances

Edges

- P_i requests instance of R_j
- P_i is holding an instance of R_j

Dependencies:

- P1 is holding an instance of R1
- P2 is holding an instance of R2
- P3 is waiting for R1 and R2.

Basic facts

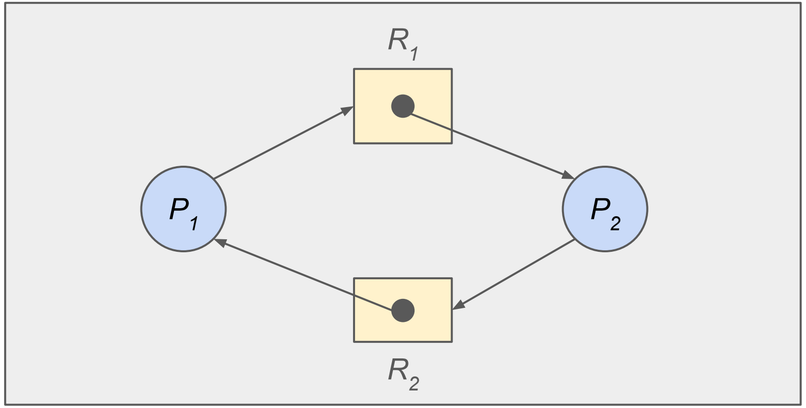
To determine if a resource allocation graph contains deadlock:

- If graph contains no cycle, then no deadlock
- If graph contains at least one cycle
  - If only one instance per resource, then deadlock
  - If several instances per resource, then possible deadlock

Basic facts

To determine if a resource allocation graph contains deadlock:

- If graph contains no cycle, then no deadlock
- If graph contains at least one cycle
  - If only one instance per resource, then deadlock
  - If several instances per resource, then possible deadlock

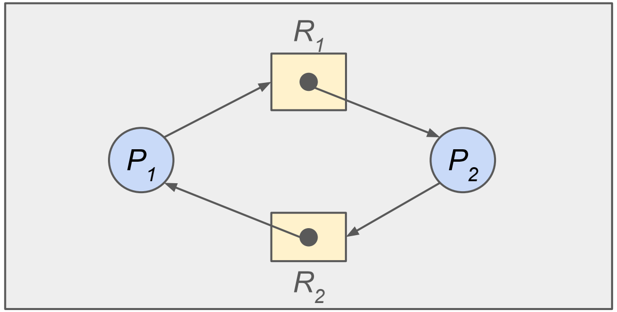


Example: processes in deadlock

- P1 holds R2
- P1 waits for R1
- P2 holds R1
- P2 waits for R2

Resource allocation table

The resource allocation table provides an overview of the allocated and requested resources.

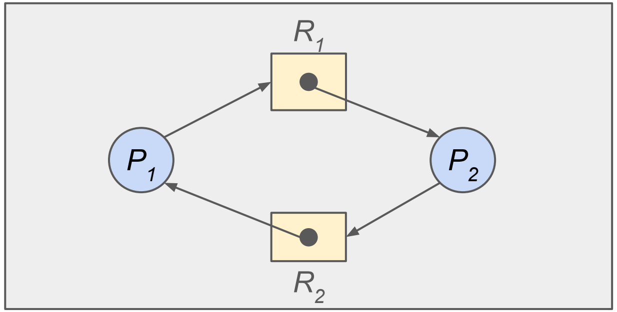


	Allocated		Requested	
	R1	R2	R1	R2
P1				
P2				

Example: Check for deadlock

Resource allocation table

The resource allocation table provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1		
P2	1	0		

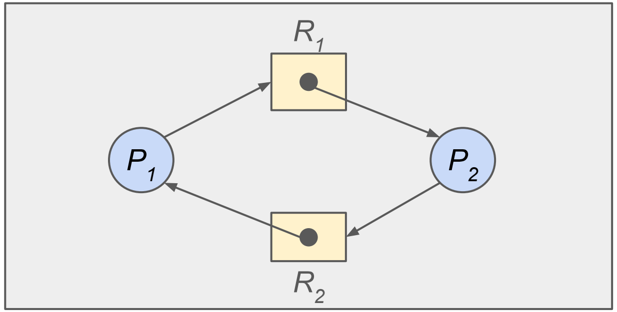
Example: Check for deadlock

Step 1: Fill in the allocated resources

- P1 is holding R2
- P2 is holding R1

Resource allocation table

The resource allocation table provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

Example: Check for deadlock

Step 1: Fill in the allocated resources

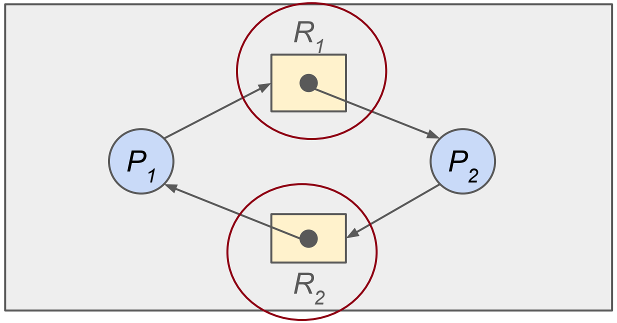
- P1 is holding R2
- P2 is holding R1

Step 2: Fill in the requested resources

- P2 is requesting for R2
- P1 is requesting for R1

Resource allocation table

The resource allocation table provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

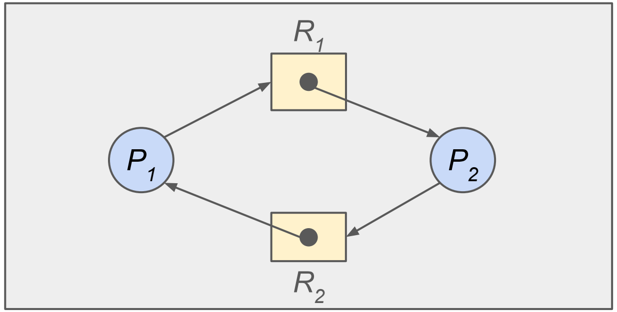
Example: Check for deadlock

Step 3: Check the availability of the resources

- Availability (R1, R2) = (0, 0)

Resource allocation table

The resource allocation table provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

Example: Check for deadlock

Step 3: Check the availability of the resources

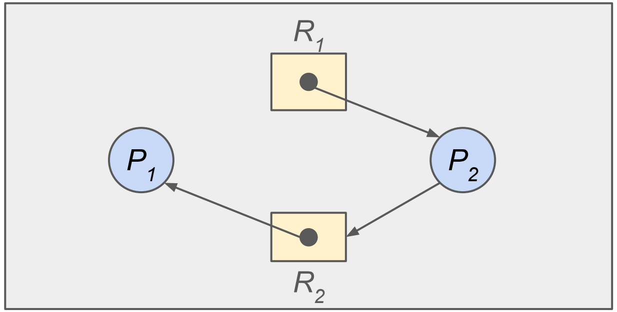
- Availability (R1, R2) = (0, 0)

Step 4: Free the requested resources (if applicable)

- With current availability, none of the requests can be fulfilled:
  - P1 Request on Availability (R1, R2) = (1, 0)
  - P2 Request on Availability (R1, R2) = (0, 1)
- Therefore, there is a deadlock

Another resource allocation table

One more example: Check for deadlock



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	0	0
P2	1	0	0	1

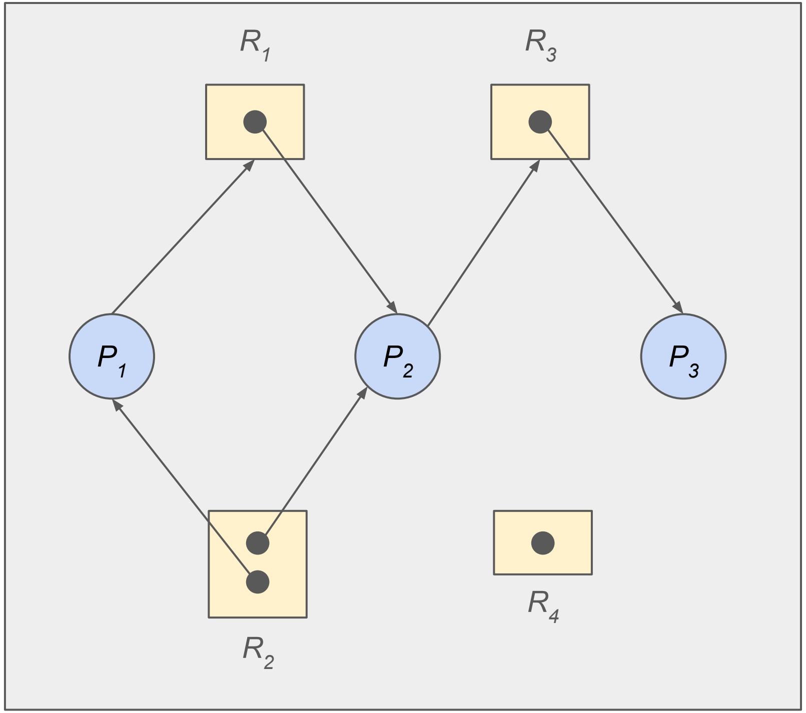
Step 3: Check the availability of the resources

- Availability (R1, R2) = (0, 0)

Step 4: Free the requested resources (if applicable)

- With current availability, none of the requests can be fulfilled:
  - P1 Request on Availability (R1, R2) = (0, 0), free P1
  - New Availability (R1, R2) = (0, 1)
  - P2 Request on Availability (R1, R2) = (0, 1), free P2
- Therefore, there is no deadlock

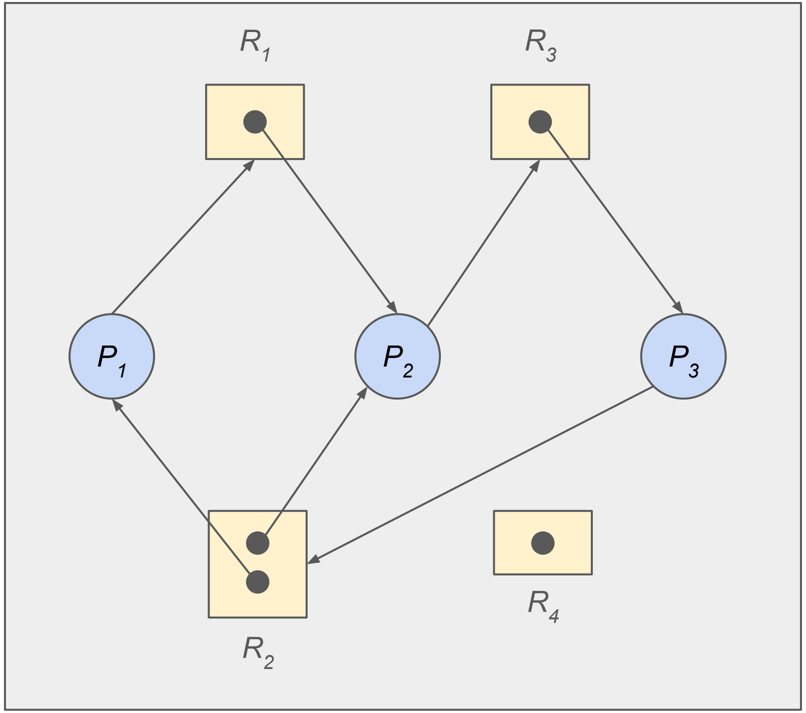
Example 1: Resource Allocation Graph



Question: Is there a deadlock?

- Check for cycles
  - If none, then no deadlock
- Check for the number of instances per resource in the cycle
  - If only one instance per resource, then deadlock
- Solve the resource allocation table

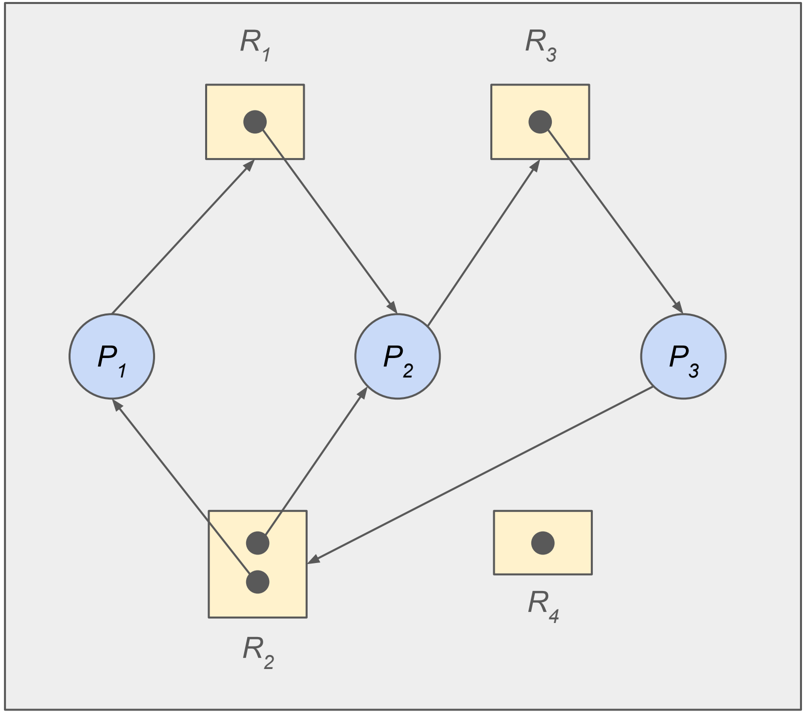
Example 2: Resource Allocation Graph



Question: Is there a deadlock?

- Check for cycles
  - If none, then no deadlock
- Check for the number of instances per resource in the cycle
  - If only one instance per resource, then deadlock
- Solve the resource allocation table

Example 2: Resource Allocation Graph



Question: Is there a deadlock?

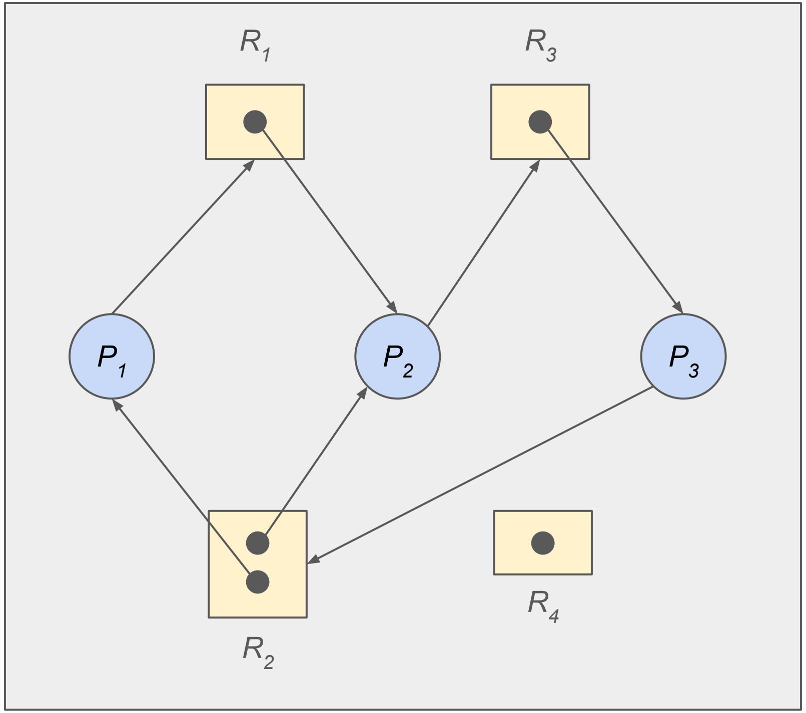
Step 1: Fill in the allocated resources

Step 2: Fill in the requested resources

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1						
P2						
P3						

Hint: allocated = R → P; requested = P → R

Example 2: Resource Allocation Graph



Question: Is there a deadlock?

Step 3: Check the availability of the resources

Example 2: Resource Allocation Graph

From Step 1 & 2:

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

From Step 3:

- Availability = (0, 0, 0)

Question: Is there a deadlock?

Step 4: Free the requested resources (if applicable)

Example 2: Resource Allocation Graph

From Step 1 & 2:

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

From Step 3:

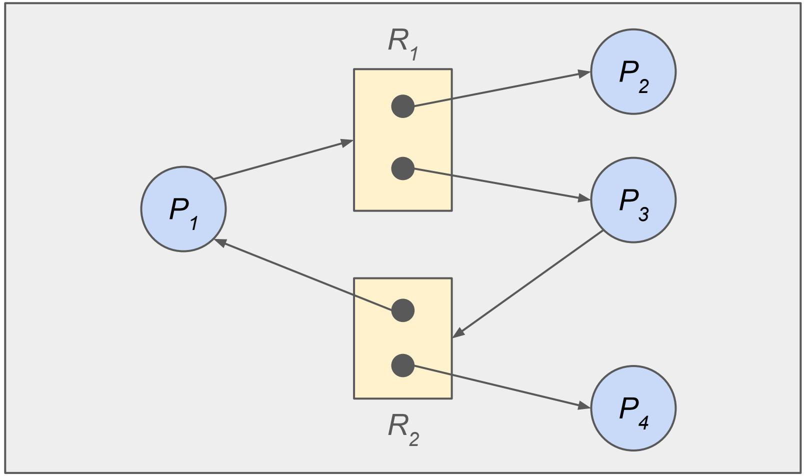
- Availability = (0, 0, 0)

Question: Is there a deadlock?

Step 4: Free the requested resources (if applicable)

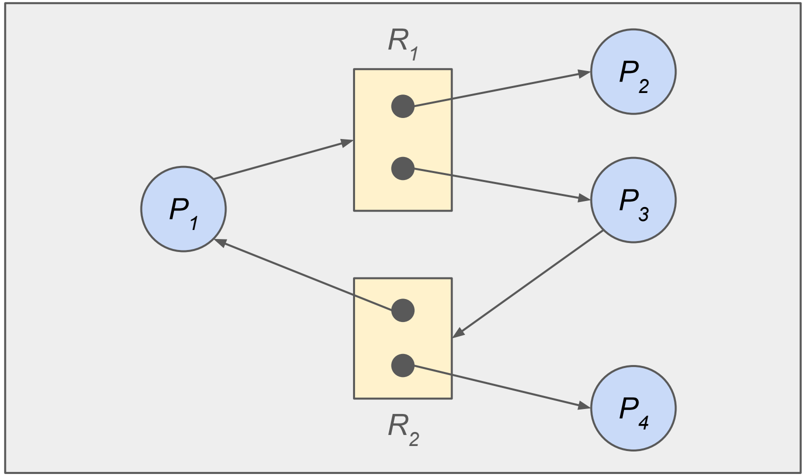
- Cannot solve any process with Availability = (0, 0, 0)
- Therefore, all three processes are deadlocked

Can we have a cycle but no deadlock?



	Allocated		Requested	
	R1	R2	R1	R2
P1				
P2				
P3				
P4				

Can we have a cycle but no deadlock?



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	0
P3	1	0	0	1
P4	0	1	0	0

Step 3: Availability = (R1, R2) = (0, 0)

Step 4: P2 and P4 break the cycle

- Availability after P2 = (1, 0)
- Availability after P4 = (1, 1)

Cycle with no deadlock

Question on deadlock from last lecture

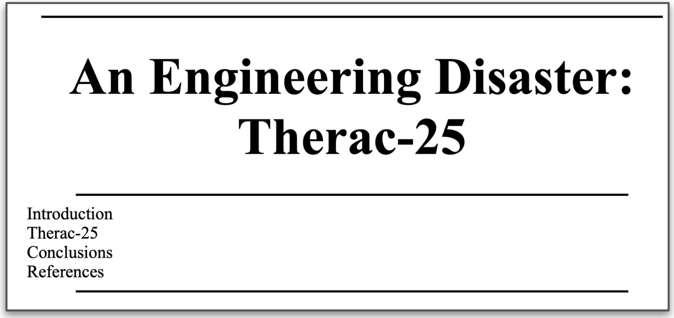
Multiple-choice question: A system that meets the four deadlock conditions will always/sometimes/never result in deadlock?

Answer: Sometimes, meeting four deadlock conditions is necessary for a deadlock to happen, but not sufficient.

- In the last example, the Circular Wait condition happened but no deadlock.

More examples of system failure due to race conditions

- Therac-25 x-ray machine (developed in Canada)
  - Between 1985 and 1987
  - Race condition caused 100 times the normal dose of radiation
  - Consequences: six patients injured, three deaths



An Engineering Disaster: Therac-25

Introduction
Therac-25
Conclusions
References

- NASA Mars-Rover
  - In January, 2004
  - Race condition identified in the file system
  - Consequences: infinite reboot loop