Deadlocks

Read Text Chapter 8

Description

Necessary Conditions for Deadlocks

Deadlocks as Directed Graphs

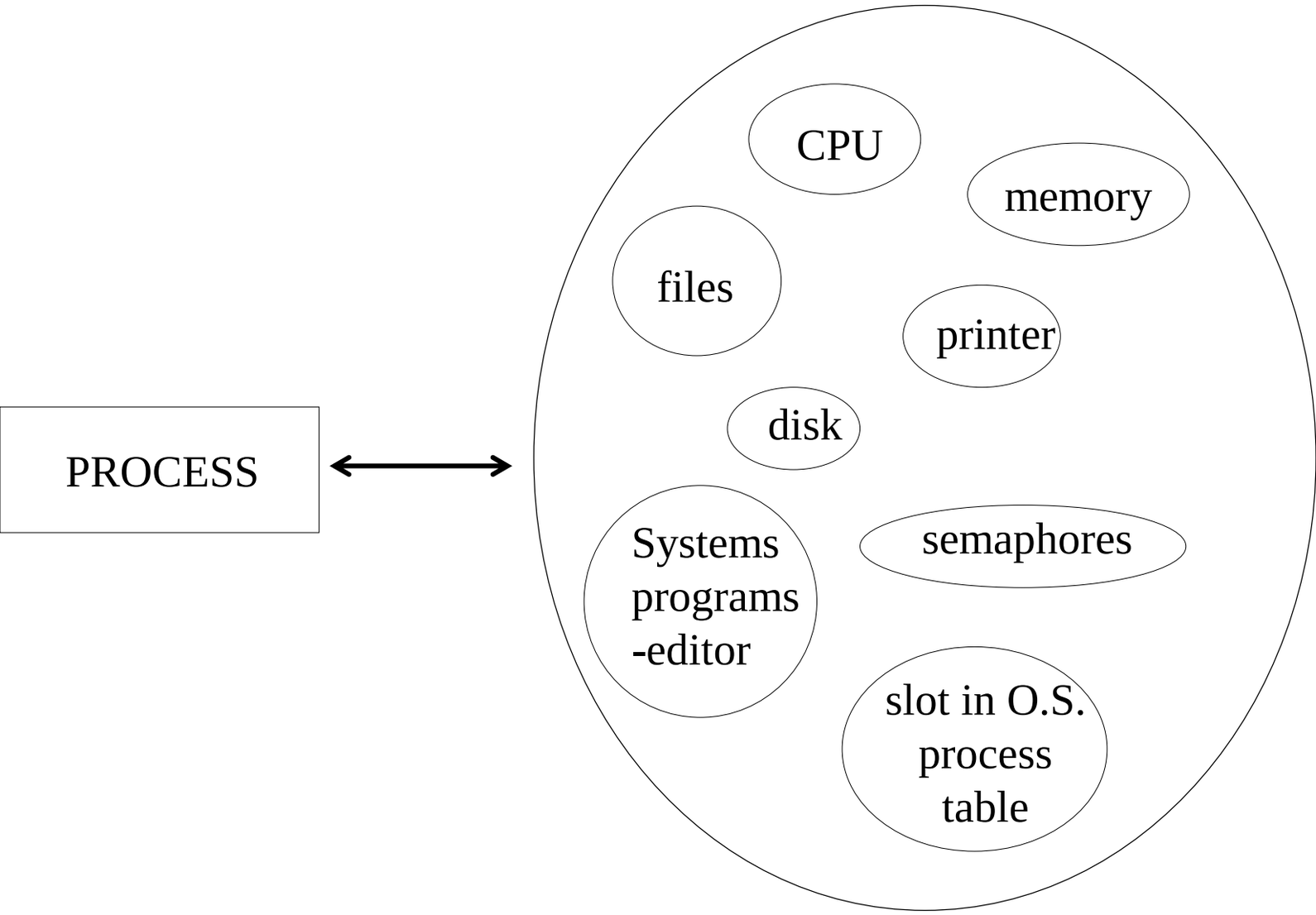
Deadlock Prevention

Deadlock Avoidance

Deadlock Detection

Recovery From Deadlock

Processes Need Resources



FINITE NUMBER of Resources must be shared among all the concurrently executing processes

Different Ways of Sharing Resources

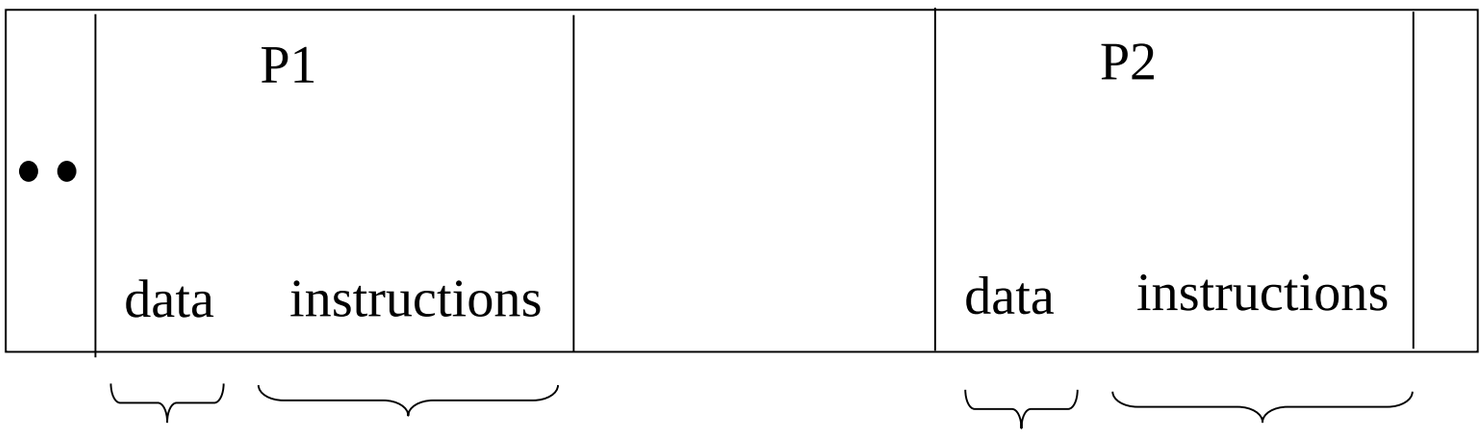
PRE-EMPTIVE

May be taken away from a process before the process is completely finished with them

Sharing pieces of the resource

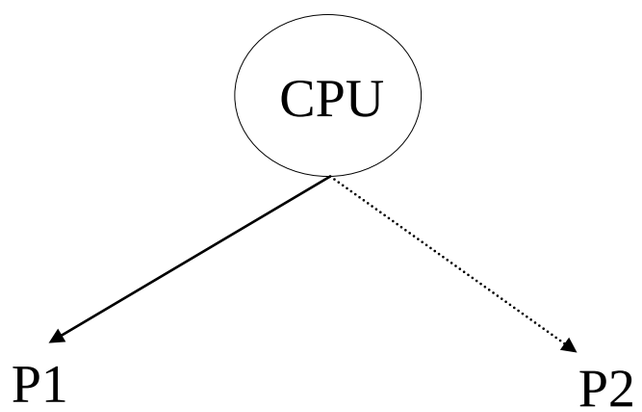
Memory can be divided into different pieces and each piece can be assigned to a process

main memory



Sharing the resource over time

CPU can be exclusively assigned to a process for a few milliseconds and then to another process, etc.



Different Ways of Sharing Resources

NON PRE-EMPTIVE

Can be taken away from a process only after the process is completely finished with them

Serially Re-usable resources

A file that a process opens and closes

A printer printing a file for a process

The disk swap space presently occupied by a blocked process

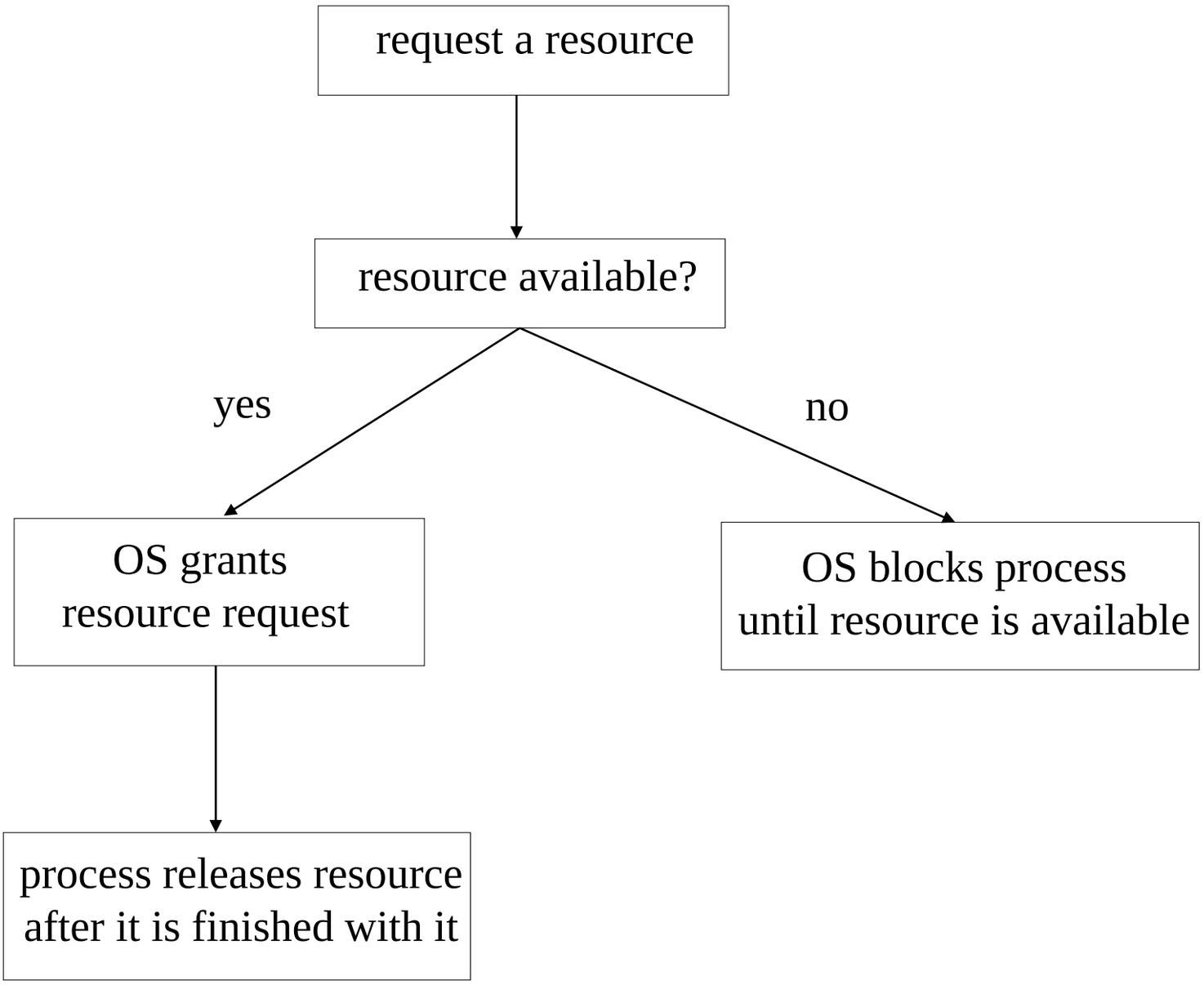
A slot in the O.S. process table

A semaphore

Deadlocks develop due to contention for non-preemptive serially reusable resources

Requesting and Releasing Resources

The sequence of events required to use a resource

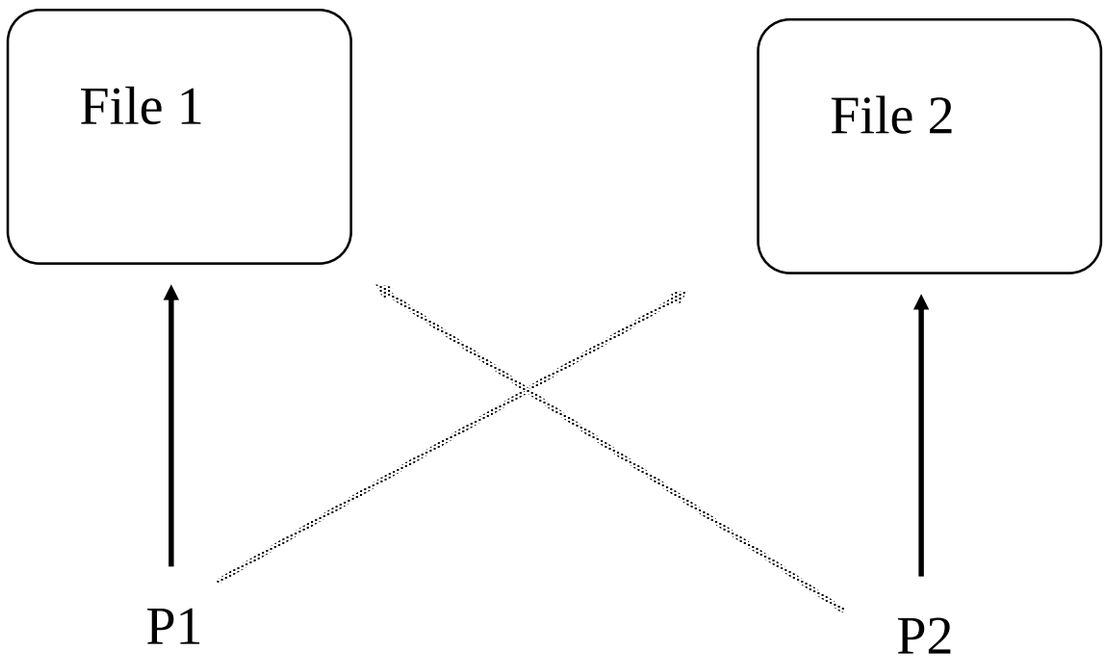


In some cases some OS will return an error when a process requests a resource that is not available but will not block the process. Programmers could implement their own waiting loops when the process encounters this condition. Depending on the programmer's solution, deadlock may or may not occur

Example of Deadlock

While P1 is accessing file 1 it also requests file 2 and it starts waiting for it. While P2 is accessing file 2 it also requests file 1 and it starts waiting for it.

Deadlock occurs !



A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause

Tanenbaum

Example of Deadlock

If THREAD 1 acquires the lock on A, and THREAD 2 acquires the Lock on B before THREAD 1 acquires it ……

Deadlock occurs !

NOTICE THAT DEADLOCK MAY OR MAY NOT OCCUR DEPENDING ON HOW THREADS ARE SCHEDULED.

THREAD 1	THREAD 2
Pthread_mutex_lock(&A)	Pthread_mutex_lock(&B)
Pthread_mutex_lock(&B)	Pthread_mutex_lock(&A)
Critical Section	Critical Section
Pthread_mutex_unlock(&B)	Pthread_mutex_unlock(&A)
Pthread_mutex_unlock(&A)	Pthread_mutex_unlock(&B)

> A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
>
> Tanenbaum

Necessary Conditions for Deadlocks

The following four conditions must be true for deadlock to be possible

1. Mutual Exclusion

   At least two resources in the system must be the type that only one process at a time can use it

2. Hold and Wait

   A process using one resource (at least) is waiting to use another resource (at least) which is being used by another process

3. No Preemption

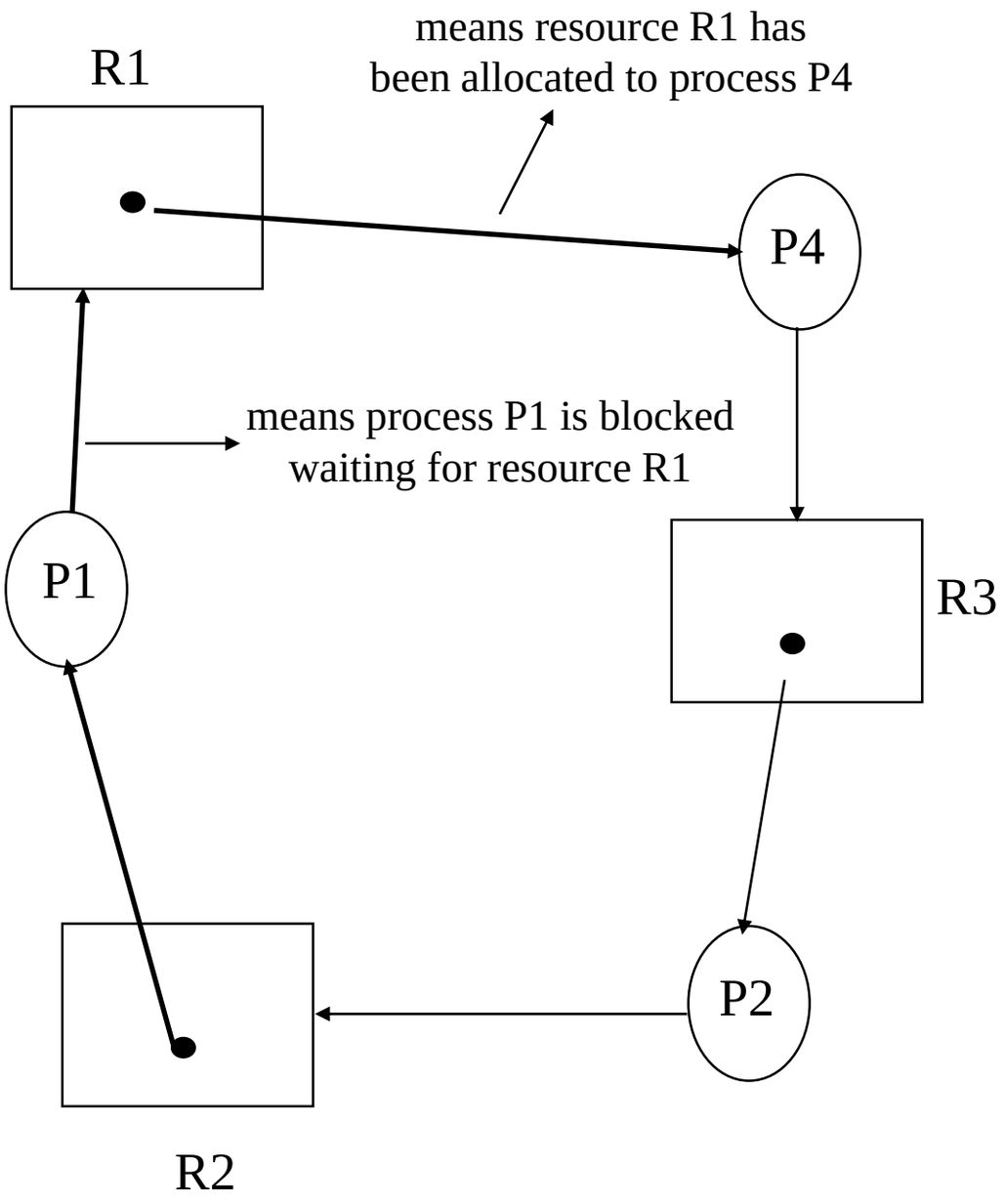
   A resource can only be released voluntarily by the process using it

4. Circular Wait

   There is a set of waiting processes { P0, P1, ... Pn } such that P0 is waiting for a resource being used by P1, P1 for P2, ... Pn for P0

Deadlock As a Directed Graph

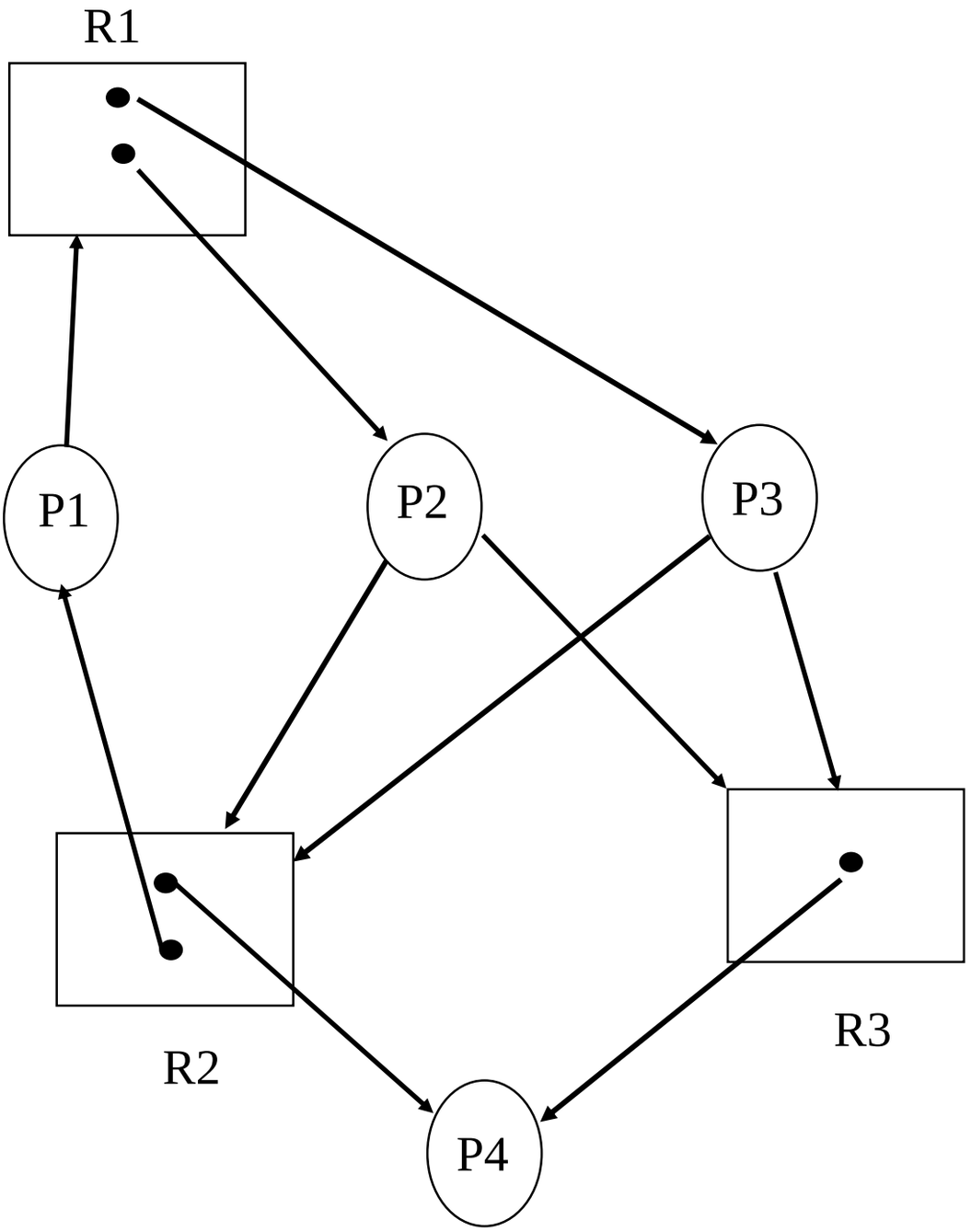
" If each resource has only one instance, then a cycle implies that a deadlock has occurred "



However if a resource has several instances, then a cycle is a necessary but not a sufficient condition for a deadlock

Deadlock As a Directed Graph

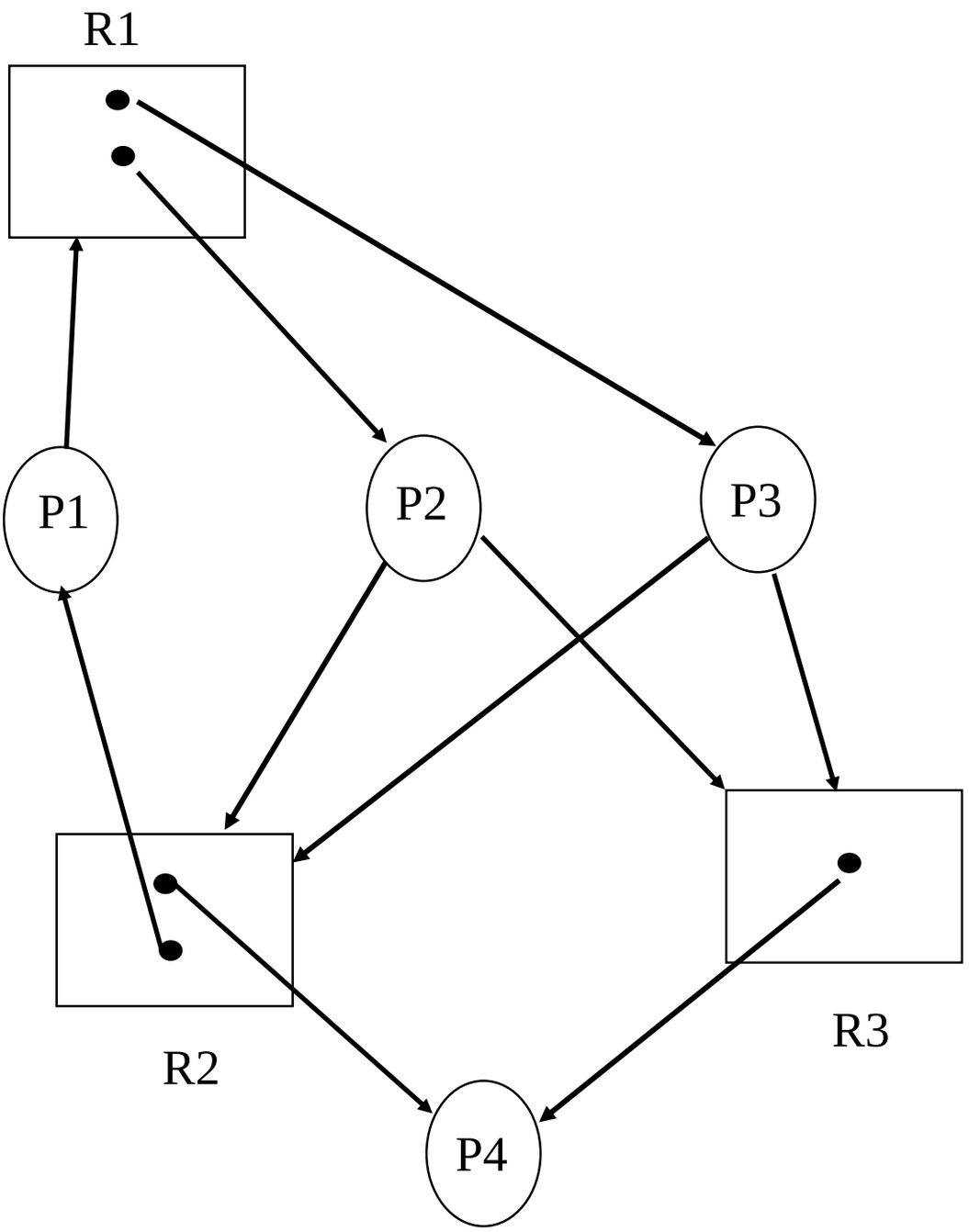
There are cycles here - but is there deadlock?



Look at every resource being used and see if it can be released by the process using it. This can only happen if that process is not waiting

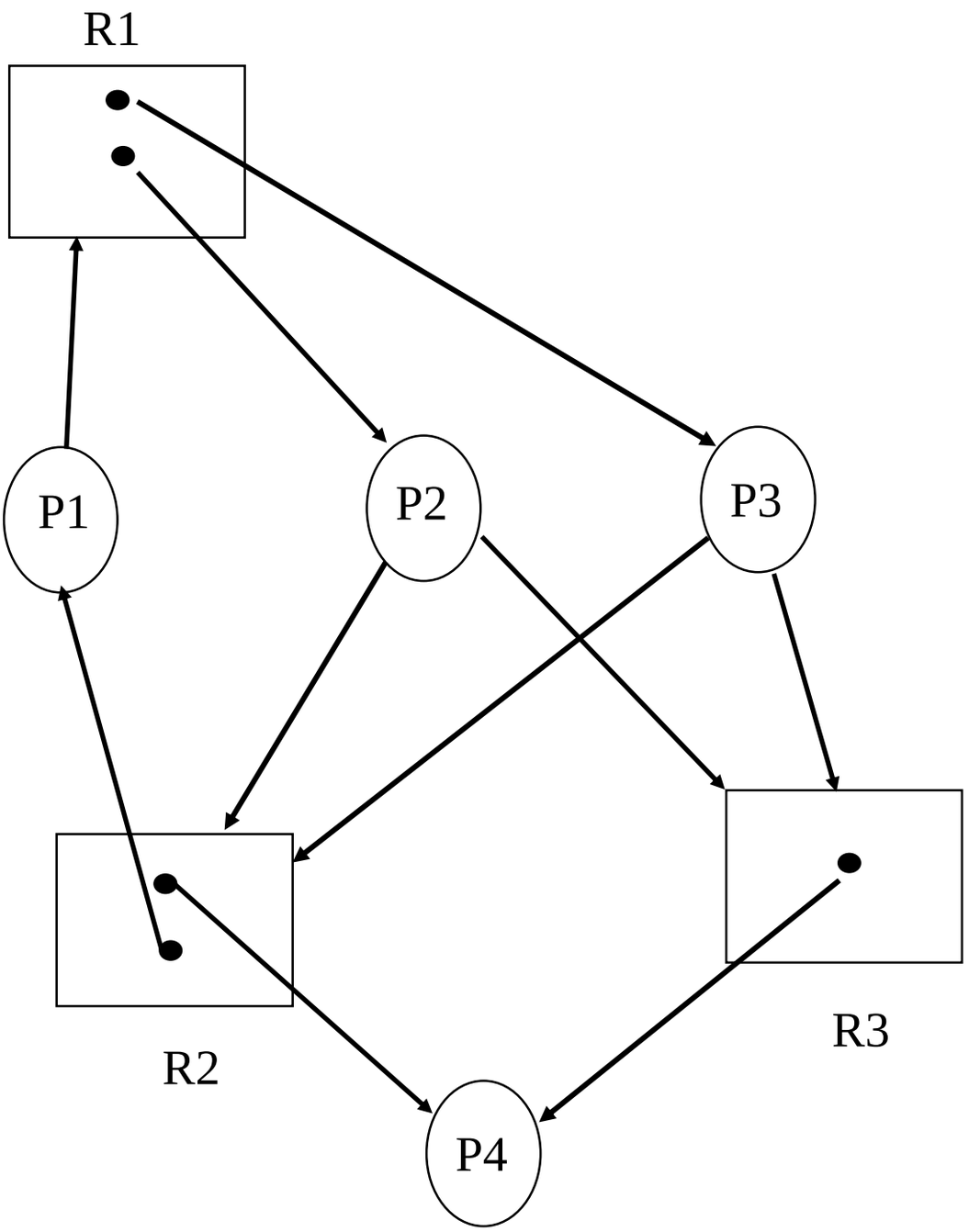
Deadlock As a Directed Graph

How could deadlock **NOT DEVELOP** from the present state?



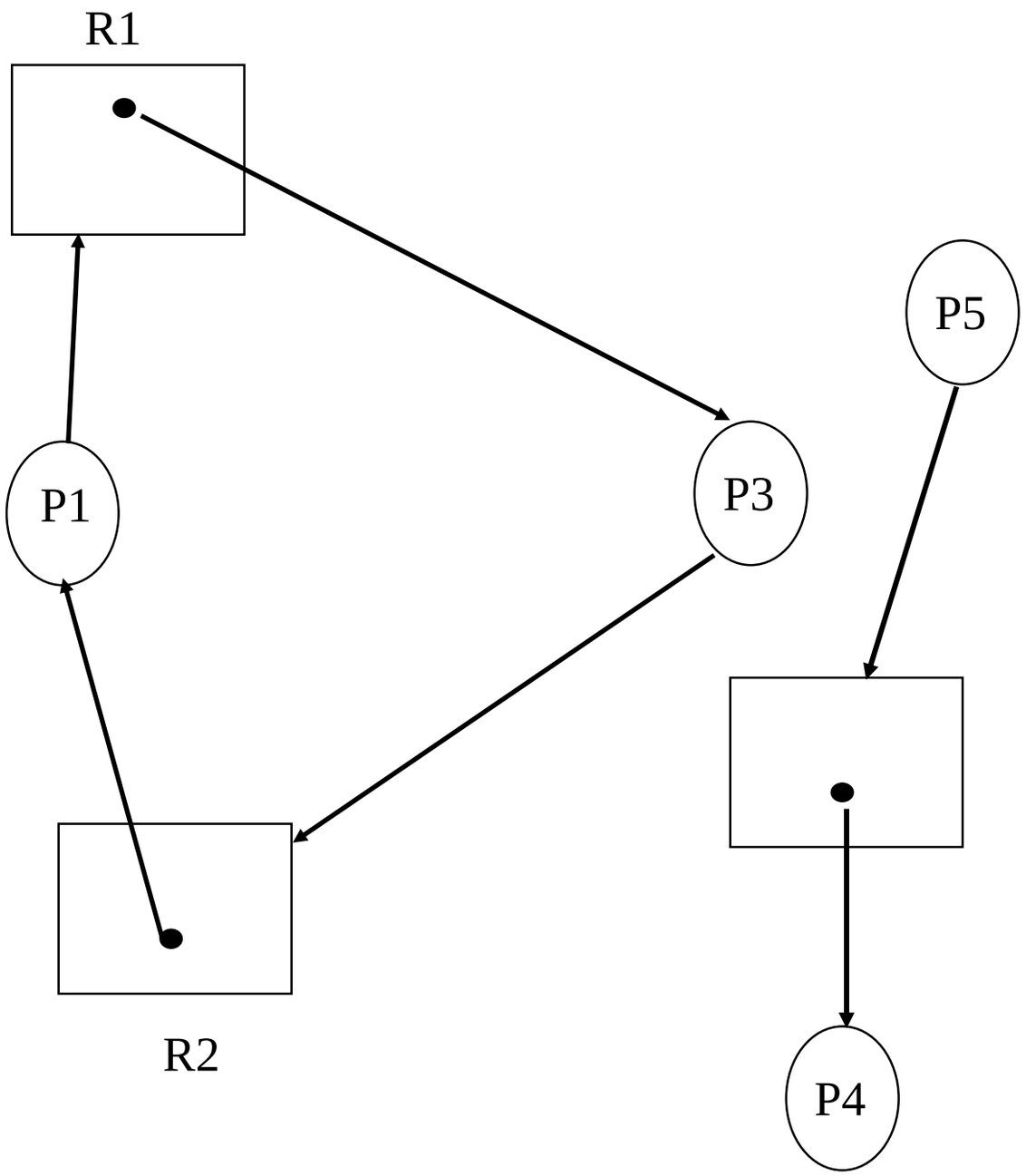
Deadlock As a Directed Graph

How could deadlock **DEVELOP** from the present state?



Deadlock As a Directed Graph

Not all processes need to be involved in deadlock



Which processes are deadlocked -- and which are not ?

Strategies for Dealing with Deadlock

1. Ignore it

   Ostrich Algorithm - may be a good solution when deadlock happens infrequently

2. Prevent it

   Make sure that at least one of the necessary conditions for deadlock does not exist in the system

3. Avoid it

   Allow the four necessary conditions for deadlock but avoid dangerous situations

4. Detect it

   Allow deadlock to occur but be able to detect it when it happens and recover from it

Deadlock Prevention

Make sure that at least one of the necessary conditions for deadlock does not occur

1. Eliminate Mutual Exclusion

--> Allow resources to be accessed simultaneously

Problem: Some resources (slots in the O.S. process table, semaphores,) are non-sharable so that this mutual exclusion must exist by necessity

2. Eliminate Hold and Wait

--> Whenever a process request a resource, make sure that it does not hold other resources doing 1 or 2 below

1. Assign all resources to a process before it begins execution

Problem - low utilization of system resources

2. A process can request a resource only when it has none

Problem - Some processes will need more than one resource at a time

3. Eliminate Preemption

-->Allow preemption. If P1 request a resource and this resource is being used by P2 and P2 is waiting for other resources, preempt P2 and allocate the resource to P1

Problem: the state of some resources cannot be easily saved and restored later

4. Circular Wait

Order the resources and then require that each process requests resources in an increasing order

Problem: too restrictive

Deadlock Avoidance

If deadlock can occur, avoid it by satisfying some resource request but not other!!

An algorithm exists - The Banker's Algorithm - that avoids deadlock if we know the following

1. The maximum number of resources of each type that a process will use
2. The number and type of resources currently being used
3. The number and type of resources currently available
4. The number and type of resources currently requested

Deadlock Avoidance

Example using only one resource type

A system has 9 instances of a resource

There are 3 processes running
- P1 will at some time require a max of 2 instances of the resource
- P2 will at some time require a max of 5 instances of the resource
- P3 will at some time require a max of 7 instances of the resource

Process	Max requirements
P1	2
P2	5
P3	7

Basic assumptions:

Processes can request any amount of resources at any time

number requested + number held ≤ Max requirement

Processes can release any or all of the resources they hold

Process	Max requirements	number held	Possible action
P1	2	0	request 1
P2	5	2	release 1
P3	7	3	request 4

Deadlock Avoidance

Safe State: One from which there exists a safe sequence

Safe Sequence: The order in which the OS will satisfy the max resource needs of each process from the current state

Process	Max requirements	number held
P1	2	0
P2	5	2
P3	7	3

Total # = 9

Safe Sequences from the present state

P1 P2 P3

P1 P3 P2

P2 P1 P3

P2 P3 P1

P3 P1 P2

P3 P2 P1

Is present state a safe state?

Deadlock Avoidance

Process	Max requirements	number held	
P1	2	0	Total # = 9
P2	5	3	
P3	7	5	

Which are the Safe Sequences from the present state ?

Is this a safe state ?

How did we arrive at this state from the safe state in previous page?

1. P3 requested 2 and OS satisfied request - new state
   Is this state safe? if so name safe sequence(s)

2. P2 requested 1 and OS satisfied request - new state

   Is this state safe? if so name safe sequence(s)

Which of the two previous request should the OS not have granted?

Bankers Algorithm

Can you think of requests from the state below that the OS should satisfy ?

Process	Max requirements	number held
P1	2	0
P2	5	3
P3	7	3

Total # = 9

Can you think of requests from the state below that the OS should not satisfy ?

The Bankers algorithm consists of only satisfying resource requests that leave the system in a safe state.

If used by the OS this algorithm can avoid deadlocks. It can be extended to multiple types of resources!

Sample Deadlock questions

Question A system has a total of 6 resources of a particular type. There are 4 processes running with their maximum resource needs and current allocation of resources as shown below.

Process	Max Need	Current Allocation
P1	4	2
P2	1	0
P3	3	0
P4	5	2

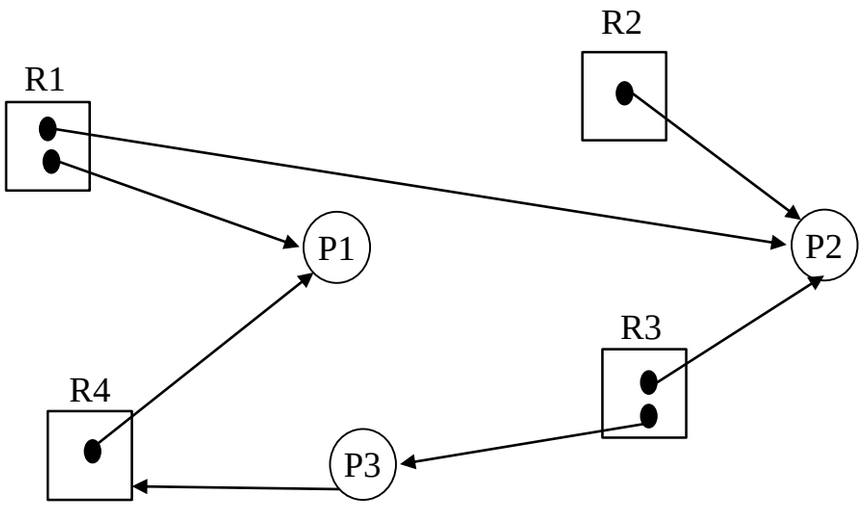
From the state above, specify all safe sequences

Question A system has a total of 5 resources of a particular type. There are 3 processes running with their maximum resource needs and current allocation of resources as shown below.

Process	Max Need	Current Allocation
P1	3	0
P2	2	1
P3	4	3

From the state above, specify all safe sequences

Question Given the "resource allocation" graph below, **describe the SHORTEST sequence of steps that will leave all processes deadlocked.** Each step in your sequence should be one of the following:
a)A specific process requests a specific resource and the OS satisfies the request
b)A specific process requests a specific resource and the OS blocks the process
c)A specific process releases a specific resource.



Bankers Algorithm

Weaknesses of the Banker's algorithm:

1. Reliable upper bounds are required on the resources to be used by each process. A user in an interactive environment may not know this in advance

2. The algorithm must be executed every time a process requests a resource. As the number of processes grow its running time increases. This overhead may be too large

Deadlock Detection

Examine the state of the system to see if deadlock exists now!

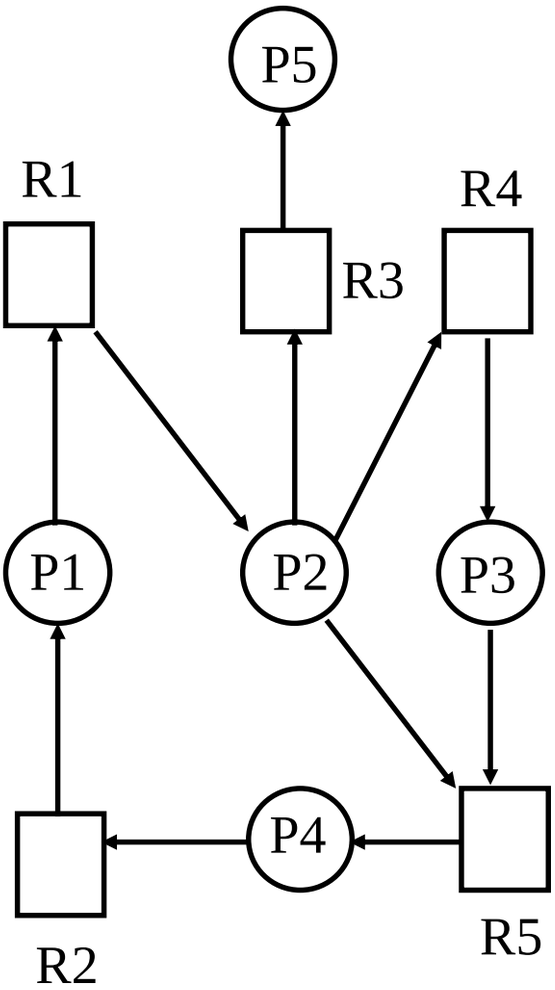
Difficulties:

- When to examine ?
  - every time requests are made?
  - when CPU utilization drops below a threshold?
  - When a number of processes are blocked for too long?

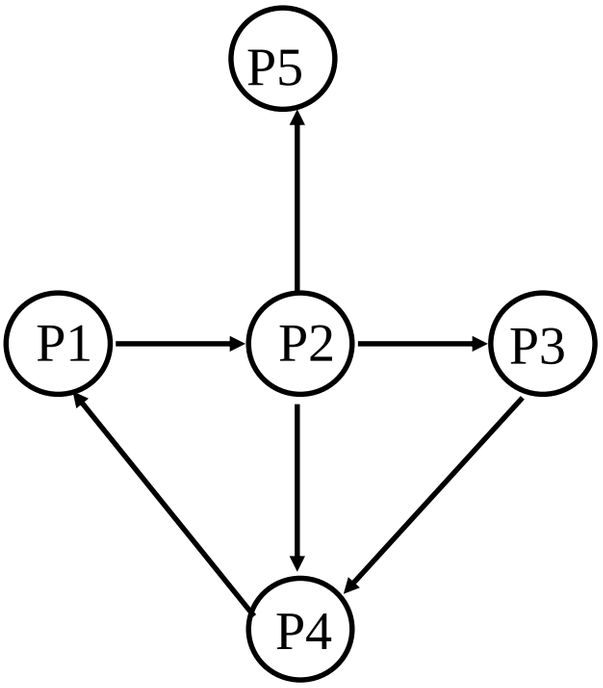
- Overhead of running algorithm

Deadlock Detection

For single instances of different resource types



Resource Allocation Graph



Wait-for Graph

The wait-for graph is obtained from the resource allocation graph by removing the resource nodes and collapsing the edges.

A deadlock exists if and only if the wait-for graph contains a cycle

Recovery From Deadlock

When deadlock is detected - what do you do?

1. Terminate all processes involved in deadlock
   drawback - wasted previous execution time

2. Terminate one process at a time
   drawback – how do we select the processes to be terminated

3. Preempt resources without terminating processes
   drawback - which process do we start with

   - can we rollback the process to a state from which it can restart again