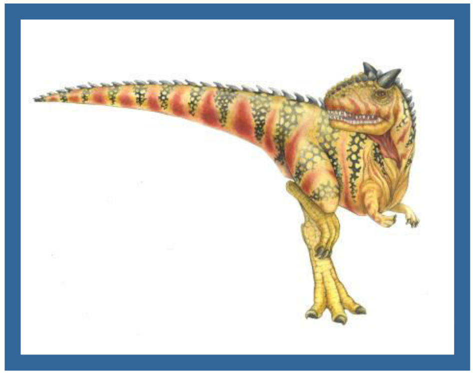
Deadlocks



Contents

Deadlock in Layman Terms

Imagine we have two friends, Alice and Bob. ***Alice has a laptop and wants to borrow Bob's tablet, while Bob has the tablet and wants to borrow Alice's laptop.***

Now, let's say they make a deal: *Alice agrees to lend her laptop to Bob if he lends her the tablet, and Bob agrees to lend his tablet to Alice if she lends him the laptop.*

The problem is that neither of them wants to give up their item first because they're waiting to receive the other item they need. So, they're stuck in a situation where they're both holding onto their items and not willing to let go until they get what they want from the other person.

Resource Use

- *Under normal mode of operation, a process may utilize the resource in only the following sequence.*

1. ***Requests** a resource*
2. ***Use** the resource*
3. ***Releases** the resource*

Deadlock

- *In multiprogramming environment, several processes may compete for a* ***finite number*** *of resources. A process requests resources, if the resources are not available at that time, the process enters into the waiting state.*
- ***Sometimes a waiting process is never again able to change state,*** **because** ***the resources it has requested are held by other processes,*** *this situation is called a* ***deadlock*****.**
- *Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.*

Deadlock

A deadlock in an operating system occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource or complete a specific task.

Deadlock Example

Processes:

1. Process P1
2. Process P2

Resources:

1. Resource 1
2. Resource 2

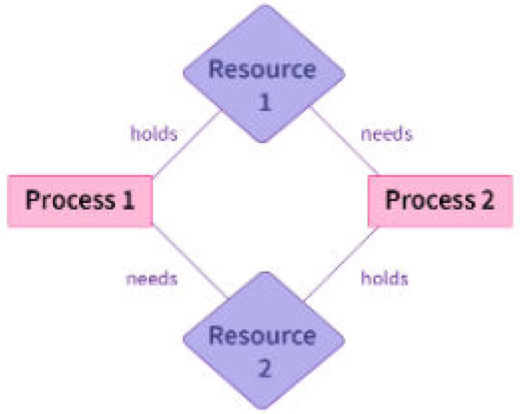
Sequence of Events:

1. Process P1 requests Resource 1 and gains exclusive access to it.
2. Process P2 requests Resource 2 and gains exclusive access to it.
3. Process P1 now wants Resource 2 but has to wait because it's currently held by P2.
4. Process P2 wants Resource 1 but has to wait because it's currently held by P1.

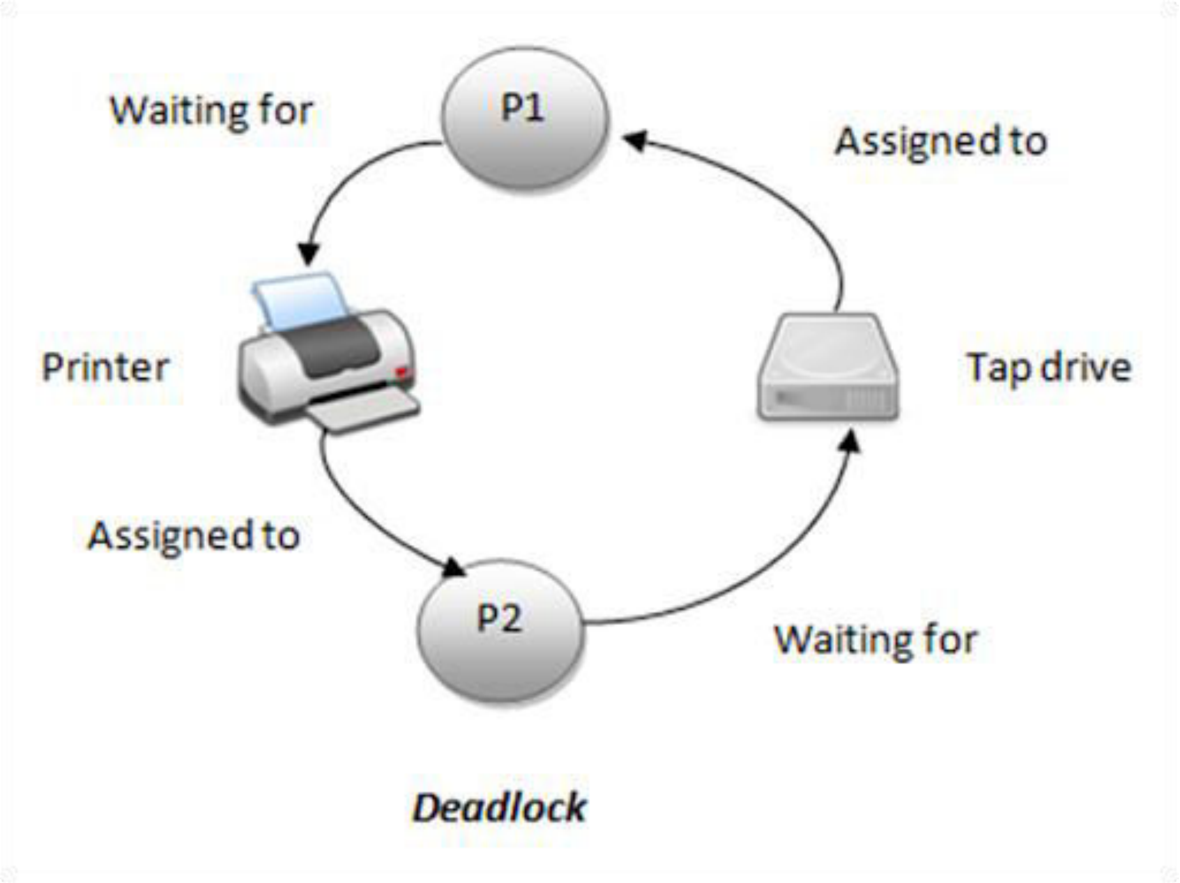
At this point, both processes are stuck in a waiting state. P1 is waiting for Resource 2, which is held by P2, and P2 is waiting for Resource 1, which is held by P1. Since neither process can release the resource it currently holds (1 or 2) and cannot proceed without the other resource, they are deadlocked.

In this deadlock scenario:

- ❑ P1 is blocking P2.
- ❑ P2 is blocking P1.

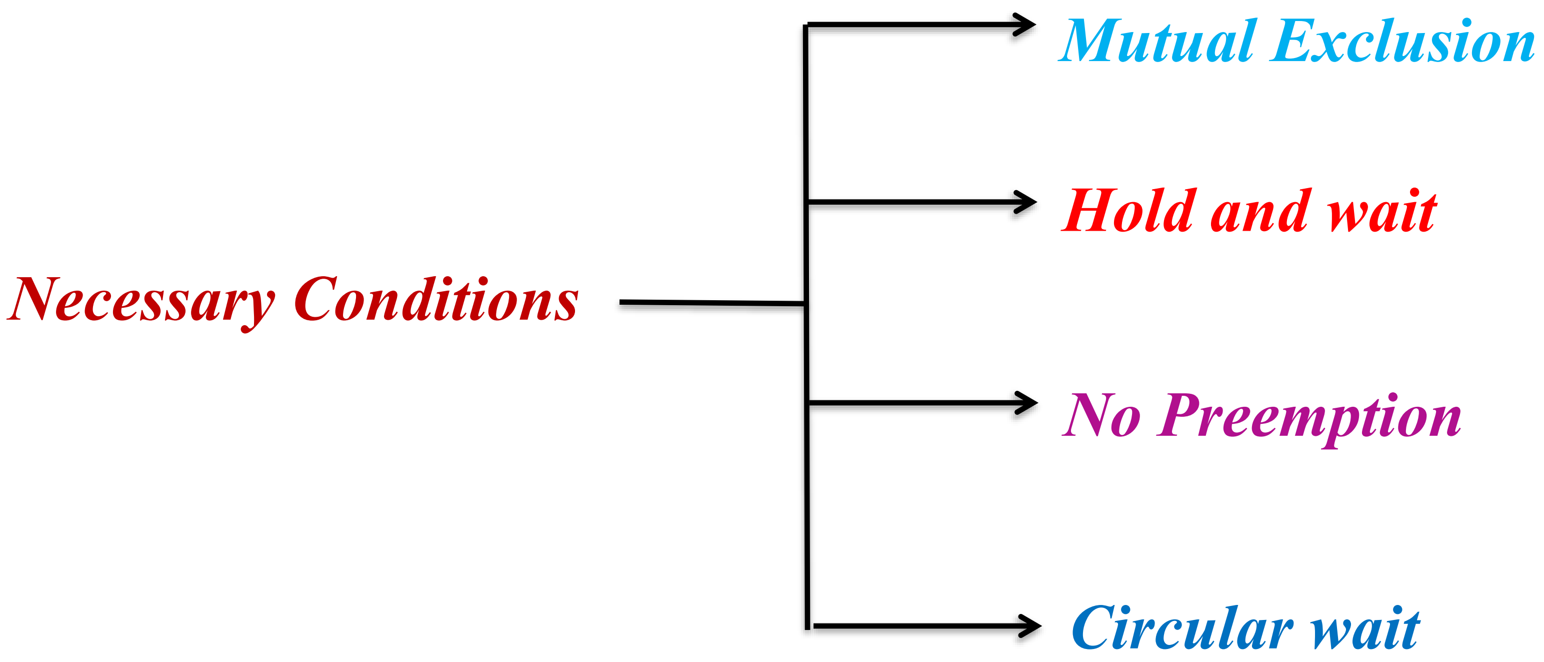


Deadlock


Waiting for
P1
Assigned to
Printer
Tap drive
Assigned to
P2
Waiting for
Deadlock

Necessary Conditions

- *In deadlock, process* ***never finish executing*** *and system resources are tied up, preventing other jobs from starting.*
- *Deadlock can arise if* ***four conditions*** *hold simultaneously. All four conditions must hold for a deadlock to occur*



Mutual Exclusion: In Real Life

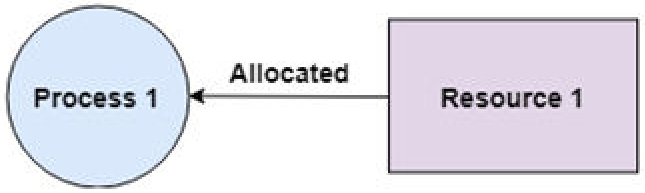
- *Traffic Intersection*

- *ATM Machines*

Mutual Exclusion (Never Share Resource)

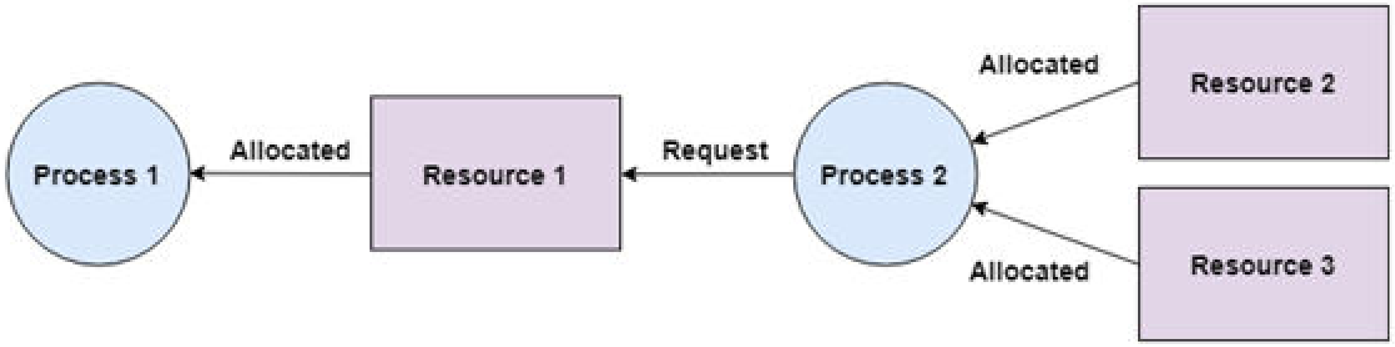
- There must exist at least one resource in the system which can be used by only one process at a time. If another process request the resource, the requested process must be delayed until the resource has been released.
- If there exists no such resource, then deadlock will never occur.
- Printer, CPU, Certain memory regions is an example of a resource that can be used by only one process at a time.

Mutual Exclusion (Never Share Resource)

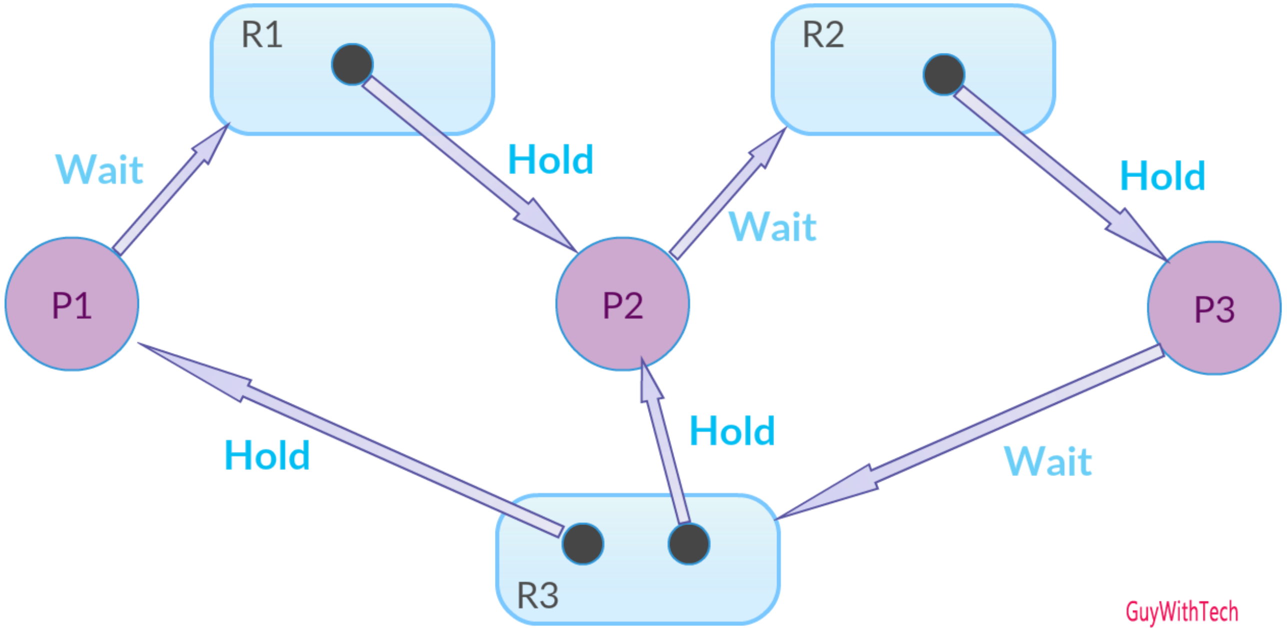


Hold and Wait

- ***Hold and wait:*** *There must exist a process which* ***holds*** *some (at least one) resource and* ***waits*** *for another resource held by some other process.*



Hold and Wait

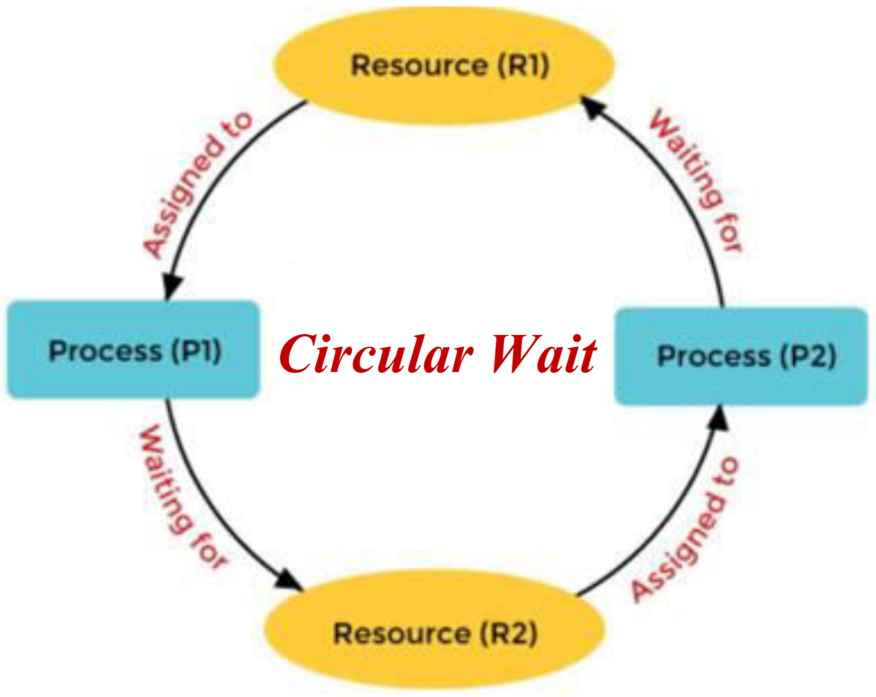


No Preemption

- *Once the resource has been allocated to the process, it can not be preempted.*
- *It means resource can not be snatched forcefully from one process and given to the other process.*
- *The process must release the resource voluntarily by itself.*

Circular wait

- *There exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.*
- *All the processes must wait for the resource in a* ***cyclic manner*** *where the last process waits for the resource held by the first process.*



P1 is waiting for P2 to release R2

P2 is waiting for P1 to release R1

- ***All these 4 conditions must hold simultaneously*** *for the occurrence of deadlock.*
- ***If any of these conditions fail****, then the system can be ensured deadlock free.*

Resource Allocation Graph

Deadlocks can be described more precisely in terms of directed graphs called a ***resource allocation graphs (RAG). RAG is the pictorial representation of the state of a system.*** *It gives complete information about the state of a system like:*

- *How many processes exist in the system?*
- *How many instances of each resource type exist?*
- *How many instances of each resource type are allocated?*
- *How many instances of each resource type are still available?*
- *How many instances of each resource type are held by each process?*
- *How many instances of each resource type does each process need for execution?*

Resource Allocation Graph

Components Of RAG -

*The graph consist of a set of vertices **V** and a set of edges **E**.*

- ***Vertices -***

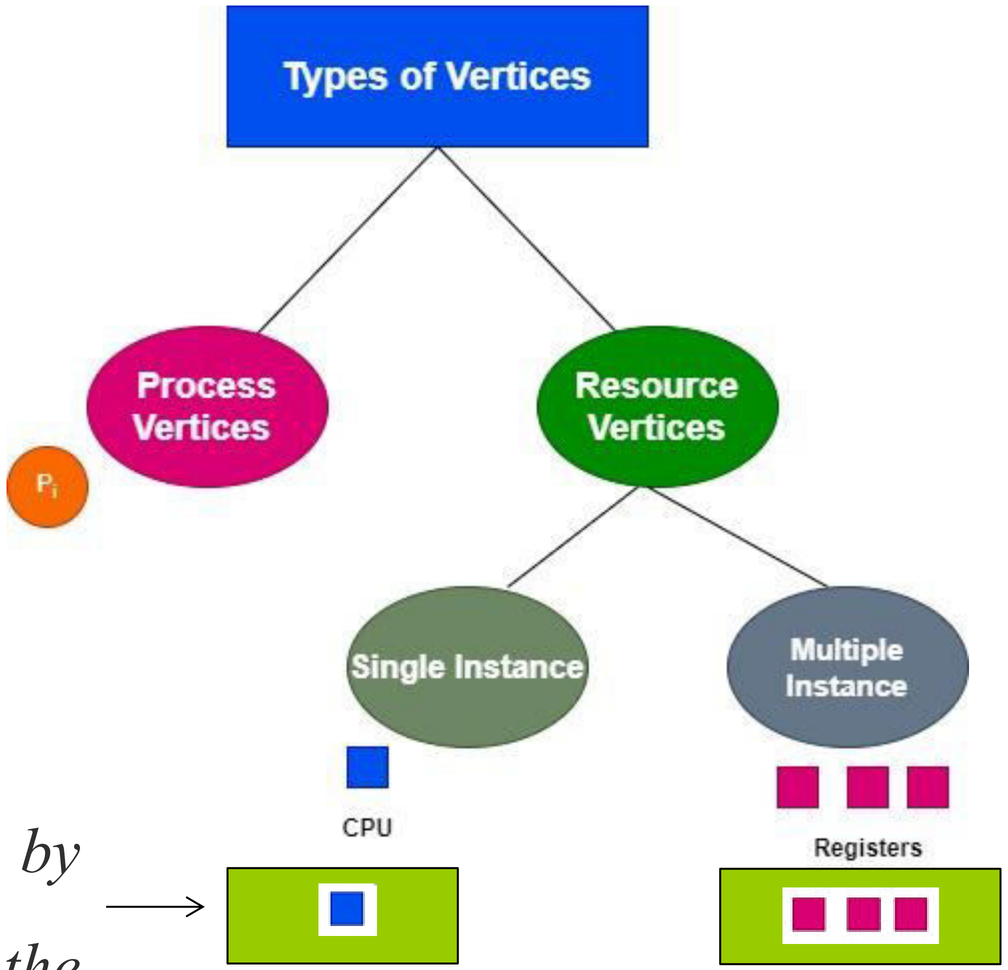
***V** is partitioned into two types:*

- ***$P = \{P_1, P_2, \ldots, P_n\}$**, the set consisting of all the processes in the system.*
- ***$R = \{R_1, R_2, \ldots, R_m\}$**, the set consisting of all resource types in the system.*

Resource Allocation Graph

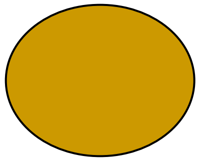
Drawn** as a **circle

***Drawn** as a rectangle by **mentioning the dots** inside the rectangle*



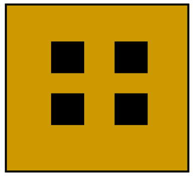
Resource Allocation Graph

Process Vertices -



- They are drawn as a **circle** by mentioning the name of process inside the circle.

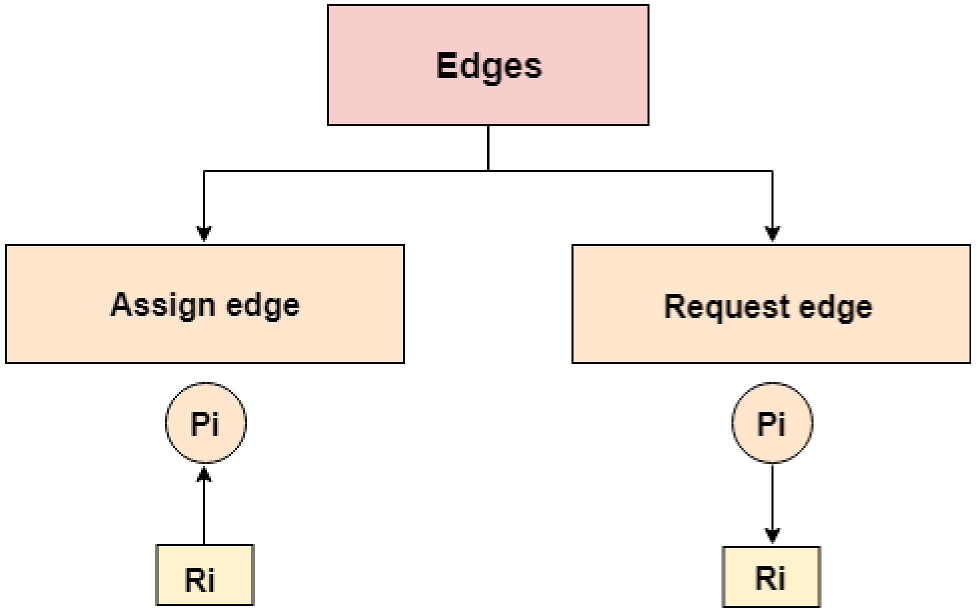
Resource Vertices -



- They are drawn as a rectangle by ***mentioning the dots*** inside the rectangle. The number of dots inside the rectangle indicates the ***number of instances*** of that resource existing in the system.

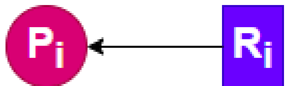
Resource Allocation Graph

- ***Edges -*** *There are two types of edges in a Resource Allocation Graph-*



- ***Assign Edges -***

- *Assign edges represent the assignment of resources to the processes.*
- *They are drawn as an arrow where the head of the arrow points to the process and tail of the process points to the instance (dot) of the resource.*

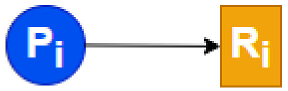


directed edge: $R_i \rightarrow P_i$

Resource Allocation Graph

- ***Request Edges-***

- *Request edges represent the* ***waiting state*** *of processes for the resources.*
- *They are drawn as an arrow where the* ***head of the arrow points to the instance*** *(rectangle) of the resource and* ***tail of the process points to the process****.*
- *If a process requires 'n' instances of a resource type, then 'n' assign edges will be drawn.*



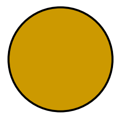
directed edge: $P_i \rightarrow R_j$

Resource Allocation Graph

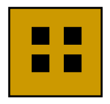
- *When process **Pi request** instance of resource type **Rj**, a **request edge** is inserted into the RAG.*
- *When request can be fulfilled the **request edge is instantaneously transformed to an assignment edge**.*
- *When process no longer needs access to the resource **it releases the resource**. As a result assignment edge is deleted*

Resource Allocation Graph

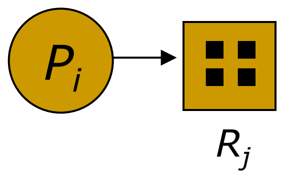
- *Process*



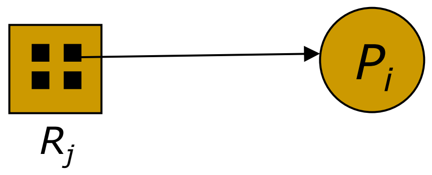
- *Resource Type with 4 instances*



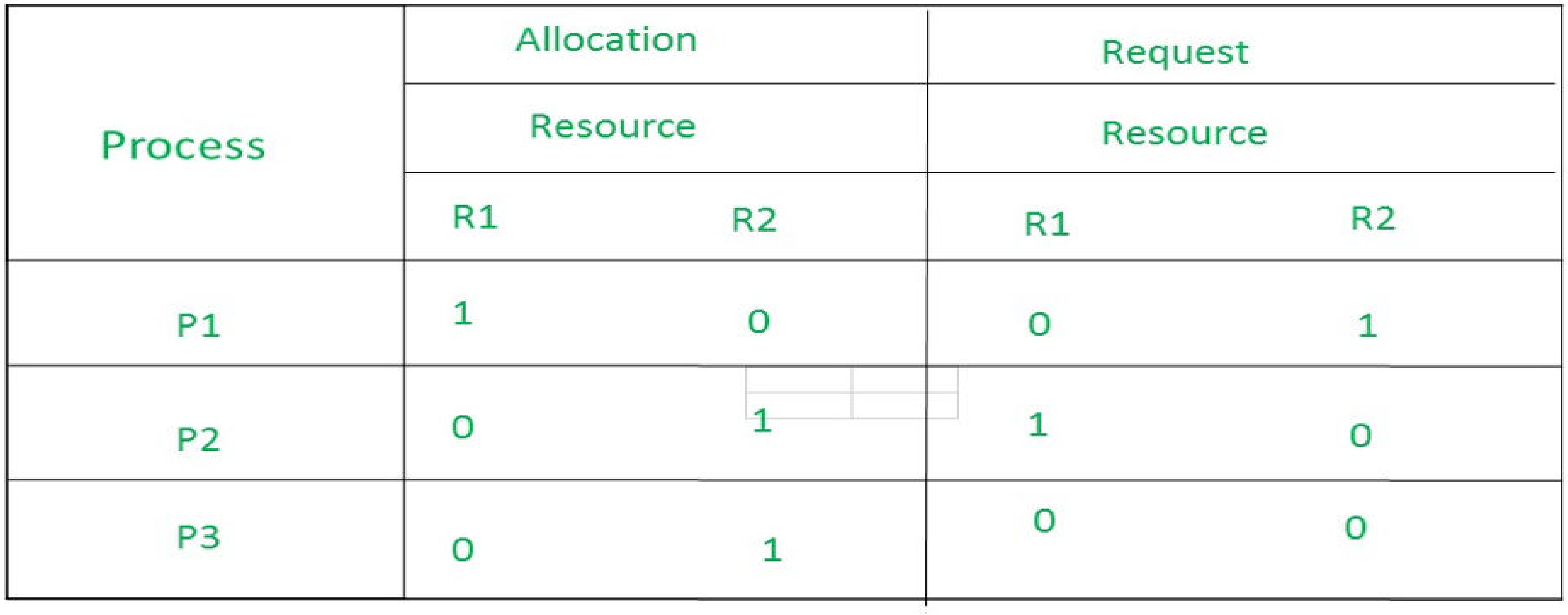
- *P_i requests instance of R_j*



- *P_i is holding an instance of R_j*



Resource Allocation Graph

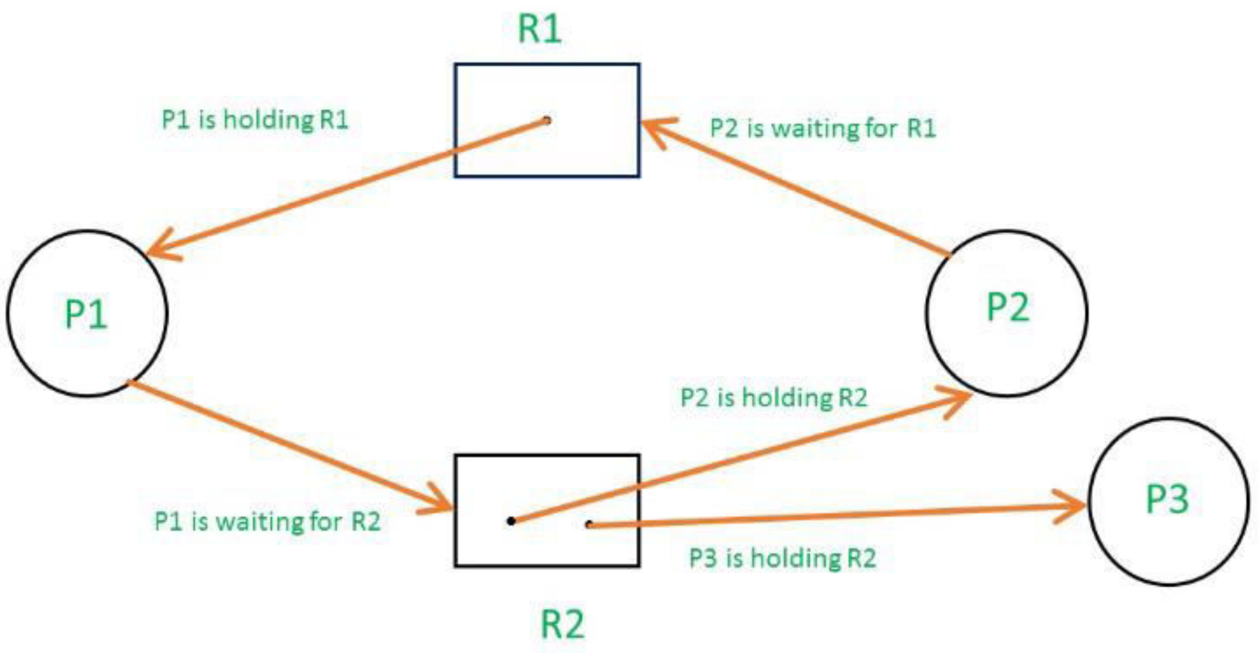


Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

Draw RAG using Allocation matrix.

Resource Allocation Graph

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0



Deadlock Detection-RAG

Using Resource Allocation Graph, it can be easily detected whether system is in a ***Deadlock state*** *or not by using two rules.*

RULE 1

In a RAG where ***all the resources are single instance***

- *If a* ***cycle*** *is being formed, then system is in a* ***deadlock state****.*
- *If* ***no cycle*** *is being formed, then system is* ***not in a deadlock state****.*
- *In this case, a* ***cycle*** *in graph is a* ***necessary and a sufficient*** *condition for the existence of deadlock.*

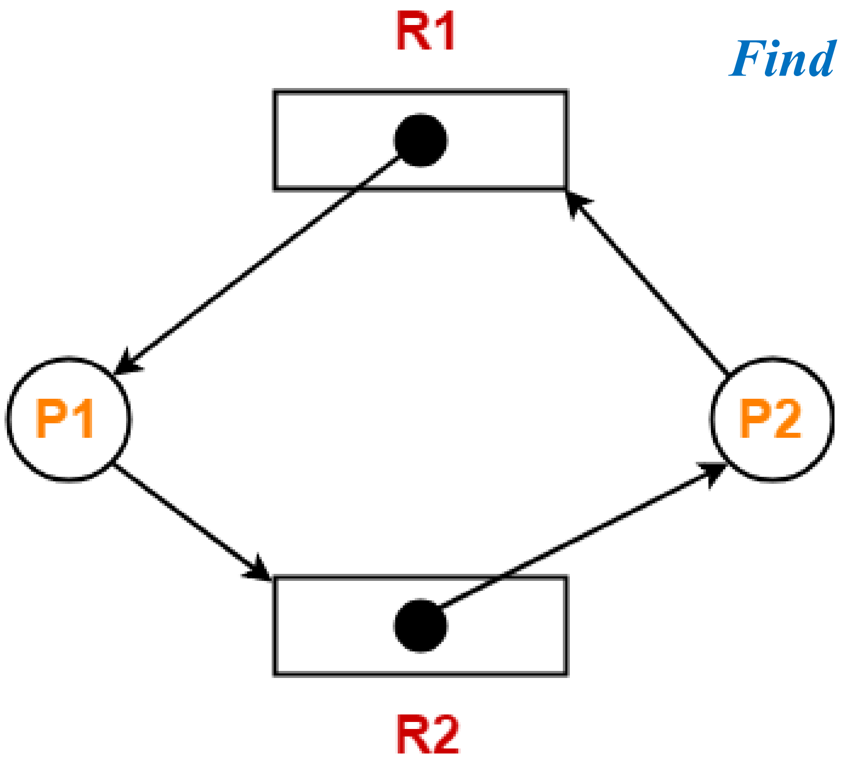
Deadlock Detection-RAG

RULE 2

In a Resource Allocation Graph ***where all the resources are NOT single instance***

- *If a* ***cycle is being formed****, then system* ***may be in a deadlock state****.*
- ***Banker's Algorithm*** *is applied to confirm whether system is in a deadlock state or not.*
- *If* ***no cycle is being formed****, then system is* ***not in a deadlock state****.*
- *Presence of a* ***cycle*** *is a* ***necessary but not a sufficient*** *condition for the occurrence of deadlock.*

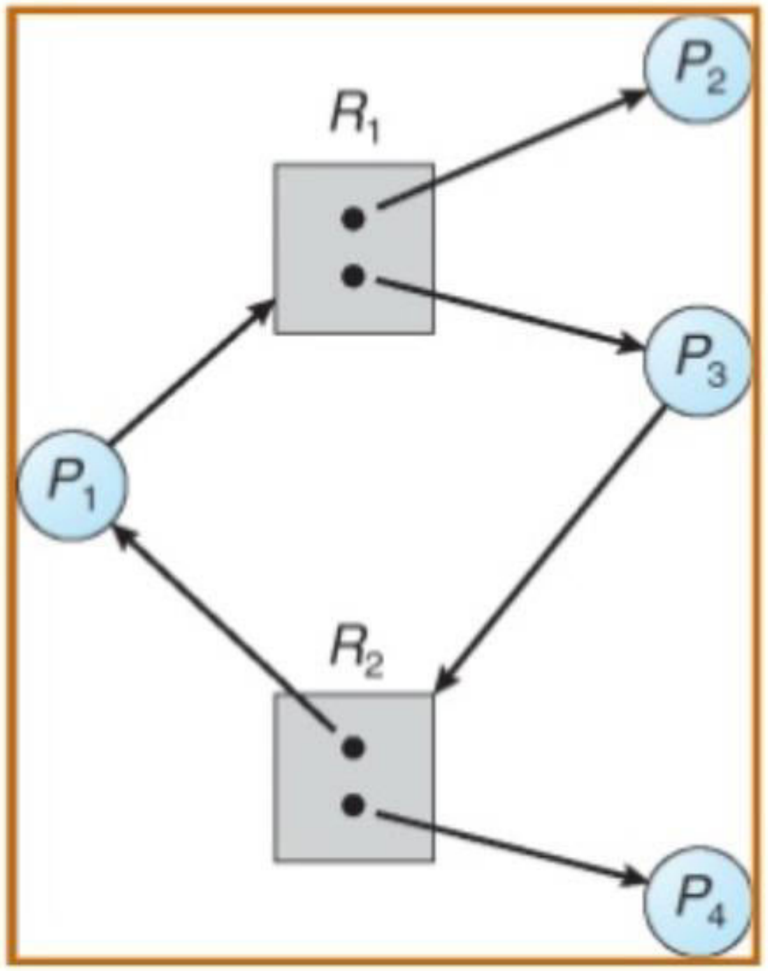
Deadlock Detection-RAG



Find if the system is in a deadlock state or not.

- *The given resource allocation graph is single instance with a cycle contained in it.*
- *Thus, the system is definitely in a deadlock state*

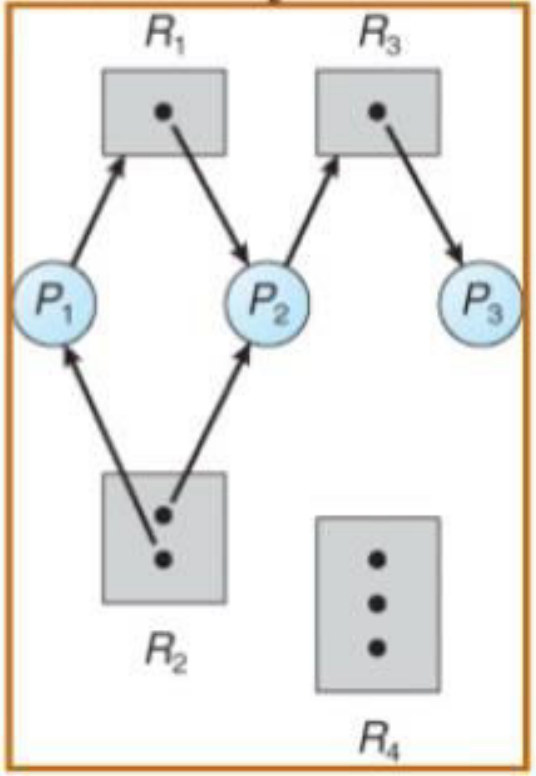
Deadlock Detection-RAG



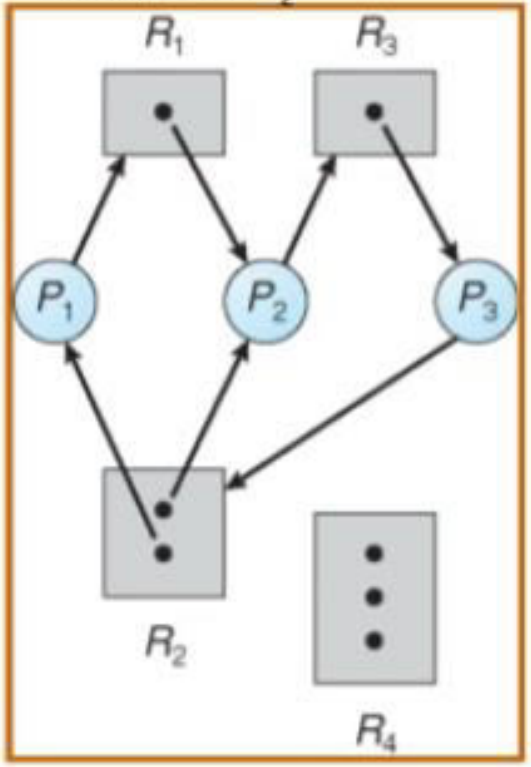
Process P4 may release its instance of resource type R2. That resource can then be allocated to P3, thereby breaking the cycle.

Deadlock Detection-RAG

Before P_3 requested an instance of R_2

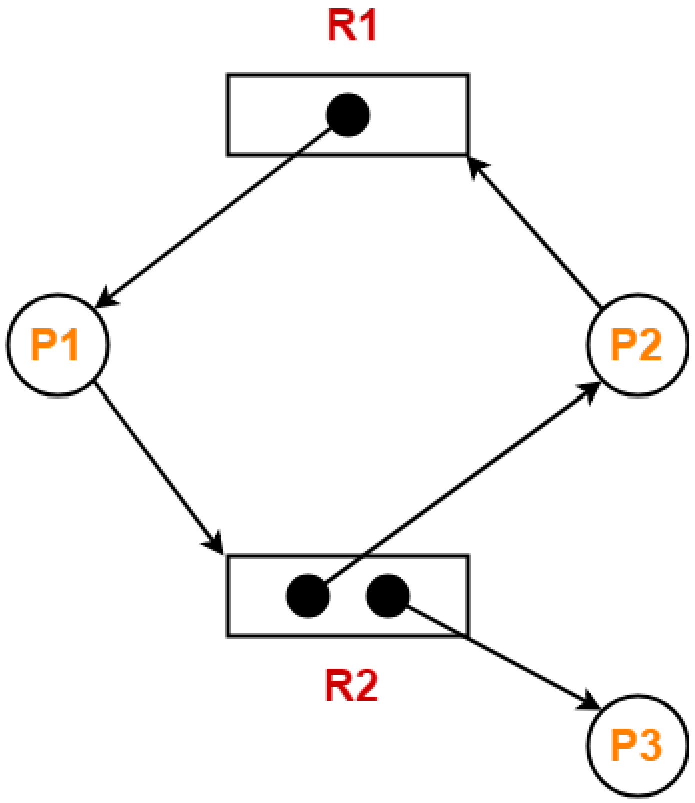


After P_3 requested an instance of R_2



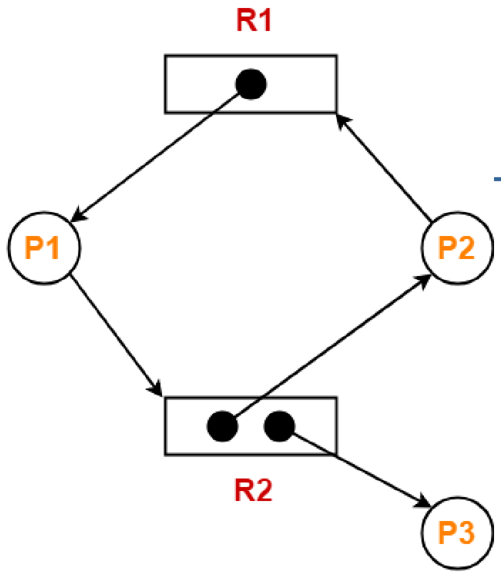
Deadlock Detection-RAG

Find if the system is in a deadlock state otherwise find a safe sequence



- *The given resource allocation graph is multi instance with a cycle contained in it.*
- *So, the system may or may not be in a deadlock state.*

Deadlock Detection-RAG

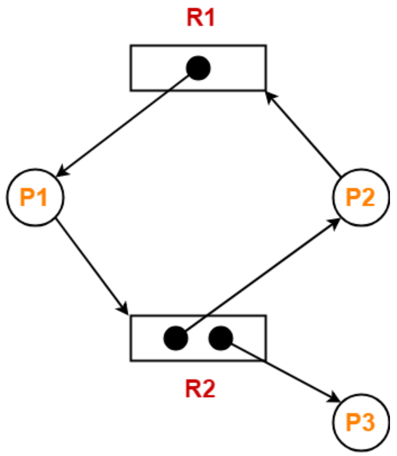


	Allocation		Need	
	R1	R2	R1	R2
Process P1	1	0	0	1
Process P2	0	1	1	0
Process P3	0	1	0	0

Available = [R1 R2] = [0 0]

Deadlock Detection-RAG

Step-01: *Since process P3 does not need any resource, so it executes.*



- *After execution, process P3 release its resources.*

  Then, Available

  = [0 0] + [0 1]

  = [0 1]

Step-02: *With the instances available currently, only the requirement of the process P1 can be satisfied.*

- *So, process P1 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

  Then- Available

  = [0 1] + [1 0] ***= [1 1]***

Deadlock Detection-RAG

Step-03: *With the instances available currently, the requirement of the process P2 can be satisfied.*

- *So, process P2 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

Then- Available

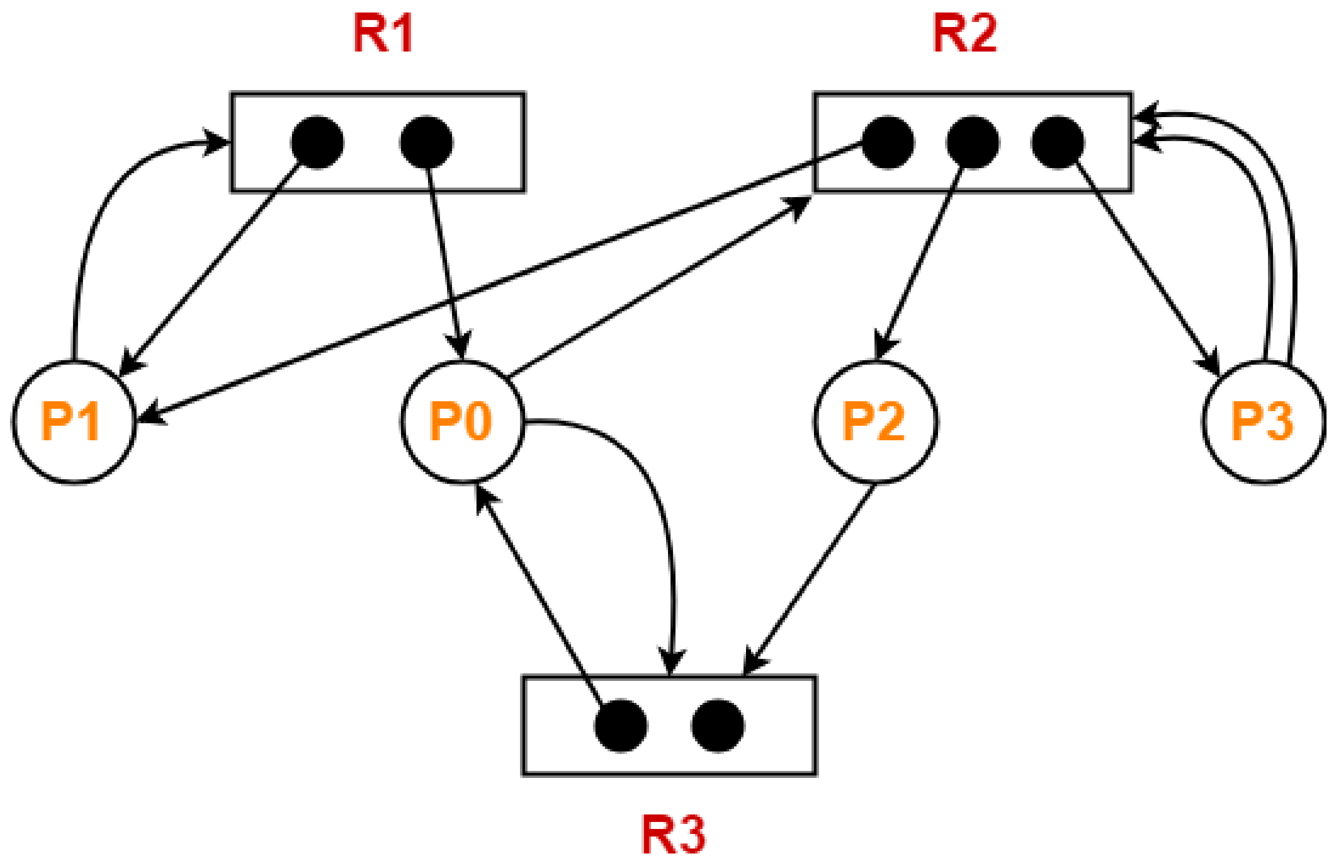
= [1 1] + [0 1]

= [1 2]

Thus, There exists a safe sequence ***P3, P1, P2*** *in which all the processes can be executed.*

So, the system is in a safe state.

Deadlock Detection-RAG



Find if the system is in a deadlock state otherwise find a safe sequence.

Deadlock Detection-RAG

Using the given resource allocation graph, we have-

	Allocation			Need		
	R1	**R2**	**R3**	**R1**	**R2**	**R3**
Process P0	1	0	1	0	1	1
Process P1	1	1	0	1	0	0
Process P2	0	1	0	0	0	1
Process P3	0	1	0	0	2	0

Available = [R1 R2 R3] = [0 0 1]

Deadlock Detection-RAG

Step - 1:

- *With the instances available currently, only the requirement of the process P2 can be satisfied.*
- *So, process P2 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

Then- Available

= [0 0 1] + [0 1 0]

= [0 1 1]

Deadlock Detection-RAG

Step-02:

- *With the instances available currently, only the requirement of the process P0 can be satisfied.*
- *So, process P0 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

Then-Available

= [0 1 1] + [1 0 1] = [1 1 2]

Deadlock Detection-RAG

Step-03:

- *With the instances available currently, only the requirement of the process P1 can be satisfied.*
- *So, process P1 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

Then- Available = [1 1 2] + [1 1 0] = [2 2 2]

Deadlock Detection-RAG

Step - 4

- *With the instances available currently, the requirement of the process P3 can be satisfied.*
- *So, process P3 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

Then-

Available

= [2 2 2] + [0 1 0] = [2 3 2]

Thus,

- *There exists a safe sequence P2, P0, P1, P3 in which all the processes can be executed.*
- *So, the system is in a safe state.*

Deadlock Practice Problem-2

A system is having 3 user processes each requiring 2 units of resource R. The minimum number of units of R such that no deadlock will occur-

a) 3

b) 5

c) 4

d) 6

Solution Deadlock Practice Problem-1

In worst case, The number of units that each process holds = ***One less than its maximum demand,*** *So,*

Process P1 holds 1 unit of resource R

Process P2 holds 1 unit of resource R

Process P3 holds 1 unit of resource R

Thus,

Maximum number of units of resource R that ensures deadlock = 1 + 1 + 1 = 3

Minimum number of units of resource R that ensures no deadlock = 3 + 1 = 4

Deadlock Practice Problem-3

A system is having 10 user processes each requiring 3 units of resource R. The minimum number of units of R such that no deadlock will occur ______?

Solution Deadlock Practice Problem-2

In worst case, The number of units that each process holds = ***One less than its maximum demand So,***

Process P1 holds 2 units of resource R

Process P2 holds 2 units of resource R

Process P3 holds 2 units of resource R and so on.

Process P10 holds 2 units of resource R

Thus,

Maximum number of units of resource R that ensures deadlock = 10 x 2 = 20

Minimum number of units of resource R that ensures no deadlock = 20 + 1 = 21

Deadlock Practice Problem-3

A system is having 3 user processes P1, P2 and P3 where P1 requires 2 units of resource R, P2 requires 3 units of resource R, P3 requires 4 units of resource R. The minimum number of units of R that ensures no deadlock is ______?

Deadlock Practice Problem-3

- *Process P1 holds 1 unit of resource R*
- *Process P2 holds 2 units of resource R*
- *Process P3 holds 3 units of resource R*
- *Maximum number of units of resource R that ensures deadlock = 1 + 2 + 3 = 6*
- *Minimum number of units of resource R that ensures no deadlock = 6 + 1 = 7*

Deadlock Practice Problem-4

If there are 6 units of resource R in the system and each process in the system requires 2 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

Solution Deadlock Practice Problem-4

In worst case, The number of units that each process holds = One less than its maximum demand So,

Process P1 holds 1 unit of resource R

Process P2 holds 1 unit of resource R

Process P3 holds 1 unit of resource R

Process P4 holds 1 unit of resource R

Process P5 holds 1 unit of resource R

Process P6 holds 1 unit of resource R

Thus,

Minimum number of processes that ensures deadlock = 6

Maximum number of processes that ensures no deadlock = 6 – 1 = 5

Deadlock Practice Problem-5

Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C which have peak demands of 3, 4 and 6 respectively. For what value of m, deadlock will not occur?

a) 7

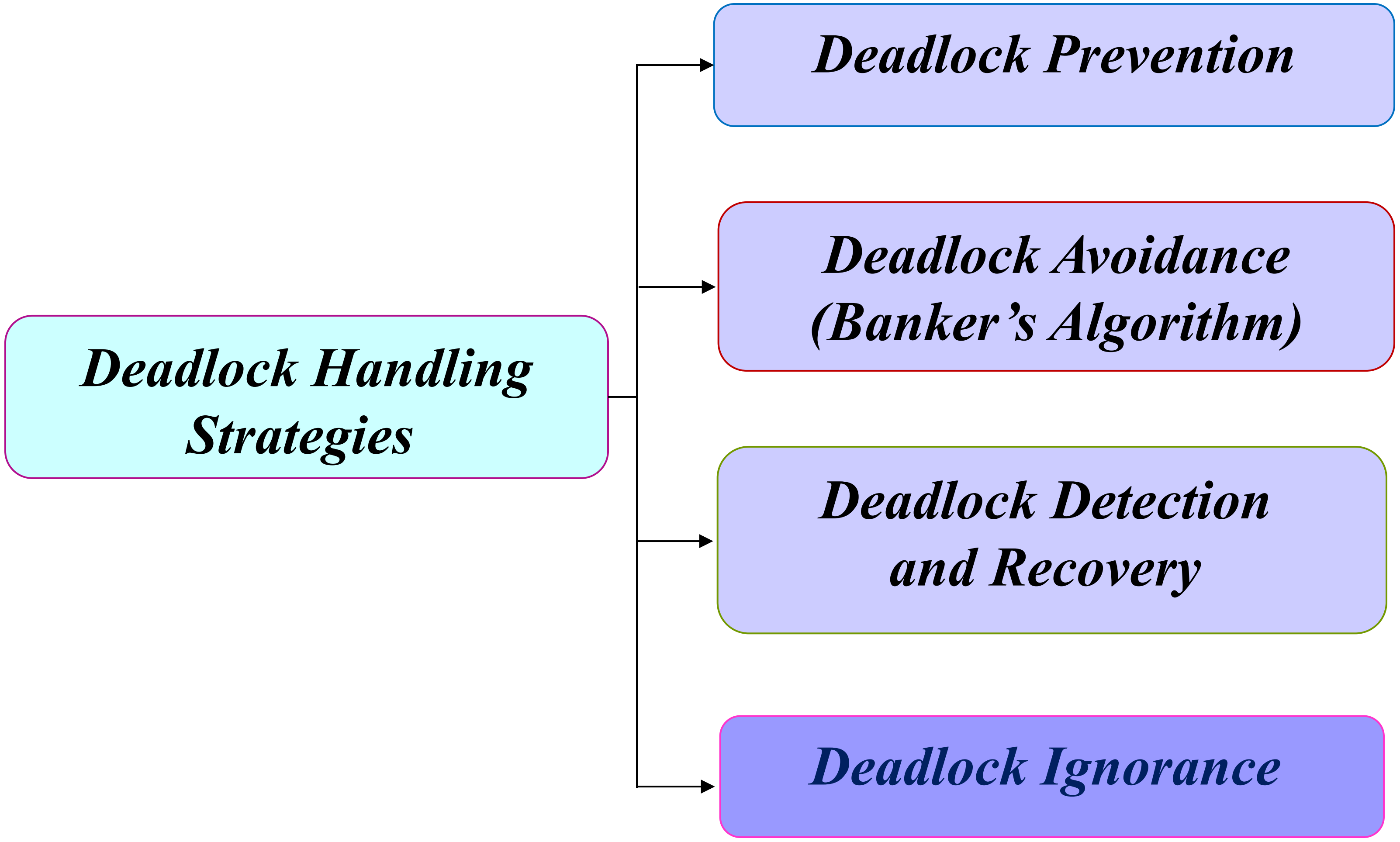
b) 9

c) 10

d) 13

Deadlock Handling Methods

There are four approaches to deal with deadlocks.


Deadlock Handling Strategies
Deadlock Prevention
Deadlock Avoidance (Banker's Algorithm)
Deadlock Detection and Recovery
Deadlock Ignorance

Methods for Handling Deadlocks

- ***Deadlock Prevention:***
- ***Deadlock Avoidance:***

*Use protocols to **prevent and avoid** deadlock to ensure that the system will never enter a deadlock state*

- ***Detection and recovery:** Allow the **system to enter a deadlock state**, detect it and recover.*
- ***Ignore the problem** and pretend that deadlocks never occur in the system, used by most operating systems, including UNIX, Linux and windows.*

Deadlock Prevention

- *Provides a set of methods to* ***ensure*** *that* ***at least one of necessary condition cannot hold****.*
- *This strategy involves designing a system that* ***violates*** *one of the four necessary conditions required for the occurrence of deadlock.*
- *This ensures that the system remains* ***free*** *from the deadlock.*

Elimination of Mutual Exclusion

- *Elimination of Mutual Exclusion – To violate this condition, all the system resources must be such that they can be used in a **shareable mode**.*

- *Sharable resource do not require mutually exclusive access thus can not involved in a deadlock. A process never needs to wait for a shareable resource.*

- *It is not possible to **dis-satisfy** the mutual exclusion because some resources are intrinsically non-sharable. For example the tape drive, mutex lock and printer, are inherently non-shareable*

Elimination of Hold and Wait

- ***Elimination of Hold and Wait*** – *To violate this condition we must guarantee that,* ***whenever a process request a resource it does not hold any other resource****.*

There are ***some protocols*** *that can be used in order to ensure that the Hold and Wait condition never occurs:*

- ***Protocol 1-*** *Each process must request and gets all its resources before the beginning of its execution.*
- ***Protocol 2-*** *Allows a process to request resources only when it does not occupy any resource.* ***(Request resources only when it has none).***

Elimination of Hold and Wait

Let us illustrate the difference between these two protocols: ***We will consider a process that mainly copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer.***

Protocol 1: *Process requests the DVD drive, disk file, and printer initially. It will hold the printer during its entire execution, even though the printer is needed only at the end.*

Disadvantage:

- *Resource utilization will be low*

Elimination of Hold and Wait

Protocol 2: *Process request initially only the DVD drive and disk file. It copies the data from the DVD drive to the disk and then releases both the DVD drive and the disk file. The process must then again request the disk file and printer. After copying the disk file to the printer, the process releases these two resources as well and then terminates.*

Disadvantage: *The process will make a new* ***request*** *for resources* ***after releasing*** *the current set of resources. This solution may lead to* ***starvation*** *(if process needs several popular resources because at least one of the resources that it needs is always allocated to some other process).*

Elimination of Hold and Wait

Drawbacks

- *The drawbacks of this approach are-*
- *It is less efficient.*
- *It is not implementable since it is not possible to predict in advance which resources will be required during execution.*

Elimination of No Preemption

This condition can by violated by forceful preemption.

- ***Protocol 1:*** *Consider a process is holding some resources and request other resources that can not be immediately allocated to it.* ***Then, all resources the process is currently holding are preempted.***

- *Preempted resources are added to the list of resources for which the process is waiting. Process will be restarted only when* ***it can regain its old resources, as well as the new ones*** *that it is requesting.*

Elimination of No Preemption

- ***Protocol 2:*** *When a process requests some resources, if they are available, then allocate them.*
- *If in case the requested resource is not available then we will check whether it is being used or is allocated to some other process waiting for other resources.*
- *If so, the OS preempts it from the waiting process and allocate it to the requesting process. And if that resource is being used, then the requesting process must wait".*

Elimination of No Preemption

- *Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.*
- *This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become* ***inconsistent/ineffective.***
- *Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes* ***performance. inefficiency.***

Elimination of Circular Wait

This condition can be violated by ***not allowing the processes to wait for resources in a cyclic manner.***

To violate this condition, the following approach is followed-

- *Impose a* ***total ordering of all resource types****, and require that each process requests resources in an* ***increasing order of enumeration****.*
- *Each resource will be assigned with a* ***numerical number****. A process can request the resources increasing/decreasing order of numbering.*

Elimination of Circular Wait

Example: *if P1 process is allocated R5 resources, now next time if P1 ask for R4 and R3, lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.*

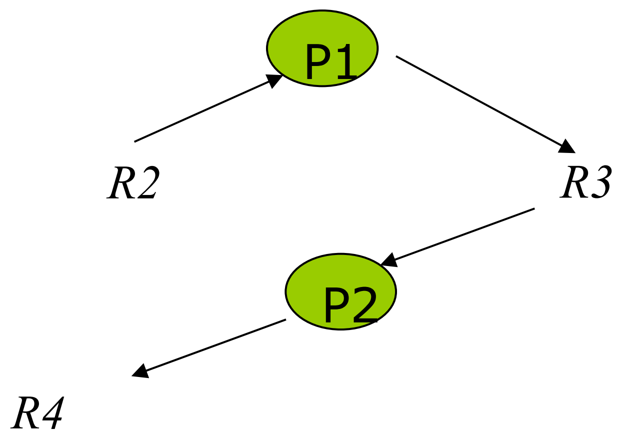
R1 – 1

R2 - 2

R3 - 3

R4 - 4

R5 - 5



Elimination of Circular Wait

Example: *If the set of resource types R includes tape drives, disk drives, and printers, then the function F might be defined as follows:*

F(tape drive) = 1

F(disk drive) = 5

F(printer) = 12

Protocol: *Each process can request resources only in an increasing order of enumeration. That is, a process can initially request any number of instances of a resource type—say, Ri . After that, the process can request instances of resource type Rj if and only if* ***F(Rj) > F(Ri).***

Elimination of Circular Wait

- *For example, using the function defined previously, a process that wants to use the tape drive and printer at the same time must first request the tape drive and then request the printer.*
- ***Alternatively**, we can require that a process requesting an instance of resource type Rj must have released any resources Ri such that **F(Ri) ≥ F(Rj).***
- *Note also that if several instances of the same resource type are needed, a single request for all of them must be issued.*

Deadlock Avoidance

- *Some* ***additional a priori information*** *needed about how resources are to be requested.*
- *The deadlock Avoidance method is used by the operating system in order to* ***check*** *whether the system is in a* ***safe state*** *or in an* ***unsafe state*** *and in order to* ***avoid the deadlocks****.*
- *The process must need to tell the operating system about the* ***maximum number*** *of resources a process can request in order to complete its execution.*

Deadlock Avoidance

In this method, *the request for any resource will be granted only if the resulting state of the system* ***doesn't cause any deadlock*** *in the system.*

- *This method* ***checks every step*** *performed by the operating system. Any process* ***continues*** *its* ***execution*** *until the system is in a* ***safe state***. *Once the system enters into an unsafe state, the operating system has to take a step back.*

- *With the help of a deadlock-avoidance algorithm, you can dynamically assess the resource-allocation state so that there can* ***never be a circular-wait situation***.

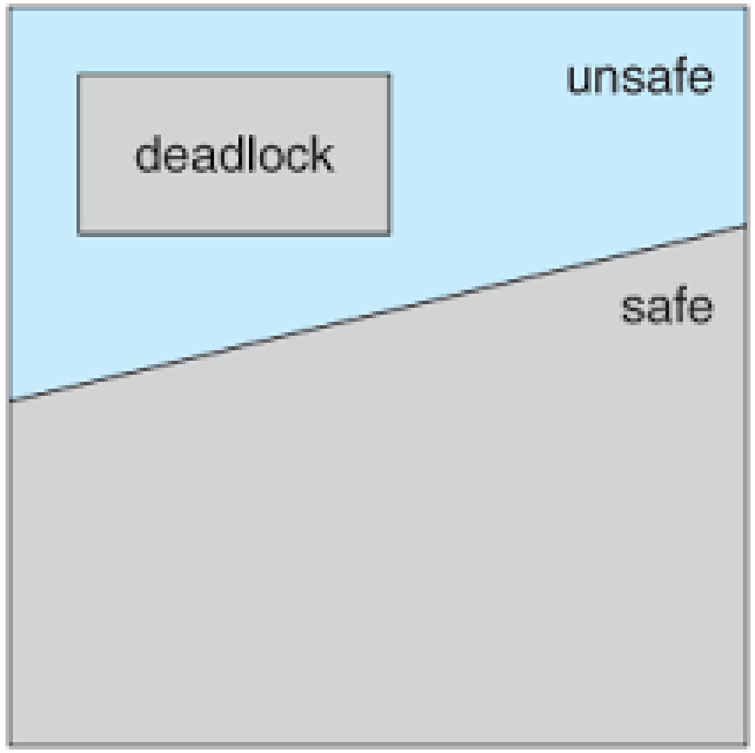
- *Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes*

Safe State and Unsafe State

- ***Safe State***: *A state is safe if the* ***system can allocate resources to each process****(up to its maximum requirement)* ***in some order*** *and* ***still avoid a deadlock****. Formally, a system is in a safe state only, if there exists a* ***safe sequence****..*

- ***Unsafe State:*** *In an Unsafe state, the operating system* ***cannot prevent processes from requesting resources in such a way that any deadlock occurs****. It is not necessary that all unsafe states are deadlocks; an unsafe state may lead to a deadlock.*

Safe State and Unsafe State



Safe State and Unsafe State

Let us consider a system having ***13 magnetic tapes*** *and three processes P1, P2, P3. Process* ***P1 requires 10 magnetic tapes****,* ***process P2 may need as many as 4 tapes****, process* ***P3 may need up to 9 tapes****. Suppose at a time* t_0*, process P1 is holding 5 tapes, process P2 is holding 2 tapes and process P3 is holding 2 tapes.* ***(There are 3 free magnetic tapes)***

Processes	Maximum Needs	Current Needs
P1	10	5
P2	4	2
P3	9	3

Safe State and Unsafe State

- ***at time t_0***, *the system is in a safe state. The sequence is* ***<P2,P1,P3>*** *satisfies the safety condition. Process P2 can immediately be allocated all its tape drives and then return them. After the return the system will have 5 available tapes, then process P1 can get all its tapes and return them (the system will then have 10 tapes); finally, process P3 can get all its tapes and return them (The system will then have 12 available tapes).*
- *A system can go from a safe state to an unsafe state. Suppose* ***at time t_1, process P3 requests and is allocated one more tape****. The system is no longer in a safe state. At this point, only process P2 can be allocated all its tapes. When it returns them the system will then have only 4 available tapes. Since P1 is allocated five tapes but has a maximum of ten so it may request 5 more tapes. If it does so, it will have to wait because they are unavailable. Similarly, process P3 may request its additional 6 tapes and have to wait which then results in a deadlock.*

Safe State and Unsafe State

- *The mistake was granting the request from P3 for one more tape*
- ***The idea*** *is simply to ensure that the* ***system will always remain in a safe state****. Initially, the system is in a safe state. Whenever a process requests a resource that is currently available, the system must decide whether the* ***resource can be allocated immediately or whether the process must wait.*** *The request is granted only if the allocation leaves the system in a safe state.*

Deadlock Avoidance

- *If a system is in **safe state** $\Rightarrow$ **no deadlocks**.*
- *If a system is in **unsafe state** $\Rightarrow$ **possibility of deadlock**.*
- ***Avoidance** $\Rightarrow$ ensure that a system will never enter an unsafe state.*

Resource-Allocation-Graph Algorithm

- ***Claim edge ($P_i \rightarrow R_j$):*** *A claim edge* ***$P_i \rightarrow R_j$*** *indicates that process P_i may request resource R_j at some time in the* ***future****.*
- *This edge resembles a request edge in direction but is represented in the graph by a* ***dashed line****.*
- *When process* ***P_i*** *requests resource* ***R_j****, the claim edge* ***$P_i \rightarrow R_j$*** *is converted to a request edge.*
- *Similarly, when a resource R_j is released by P_i , the assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge $P_i \rightarrow R_j$.*
- *Resources must be* ***claimed a priori*** *in the system. That is, before process P_i starts executing, all its claim edges must already appear in the resource-allocation graph.*

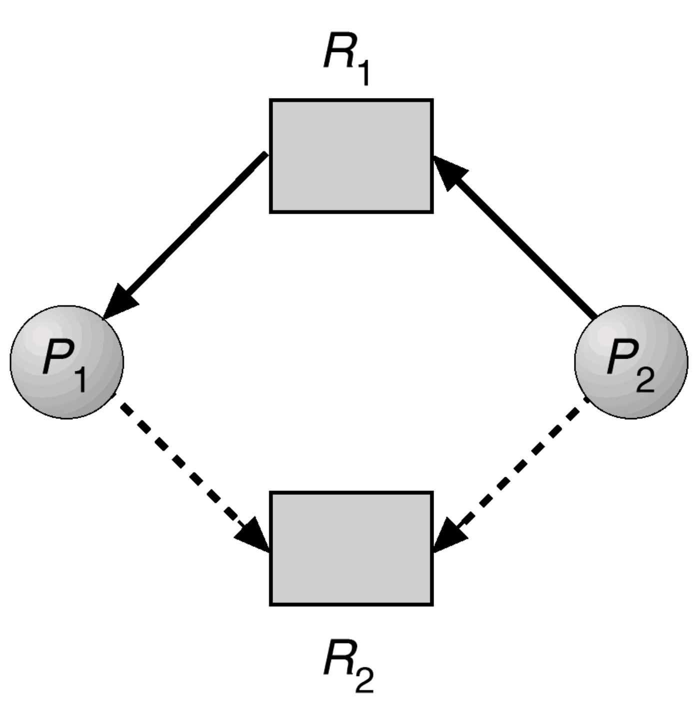
Resource-Allocation-Graph Algorithm

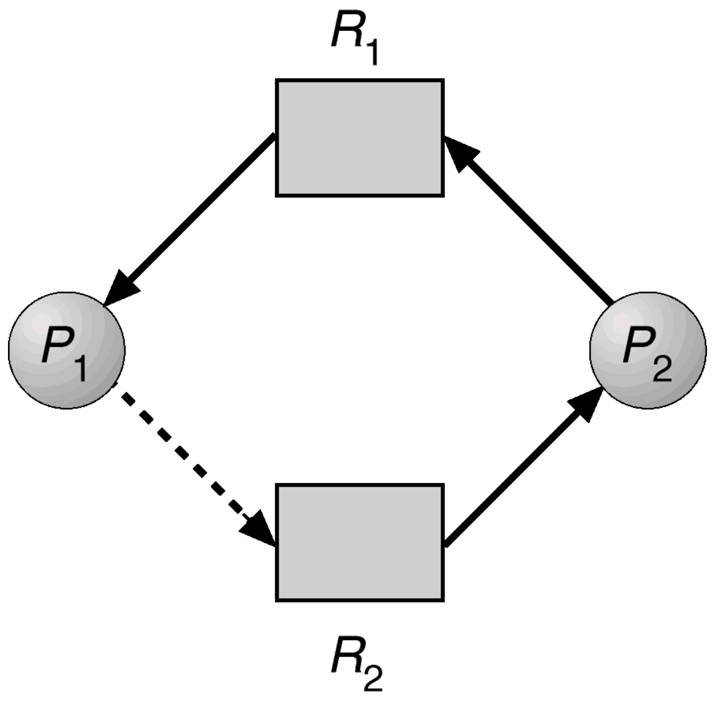
- *Now suppose that process P_i requests resource R_j. The request can be granted only if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$* ***does not result in the formation of a cycle*** *in the resource-allocation graph.*

- *If no cycle exists, then the allocation of the resource will leave the system in a* ***safe state.***

- *If a cycle is found, then the allocation will put the system in an* ***unsafe state****.*

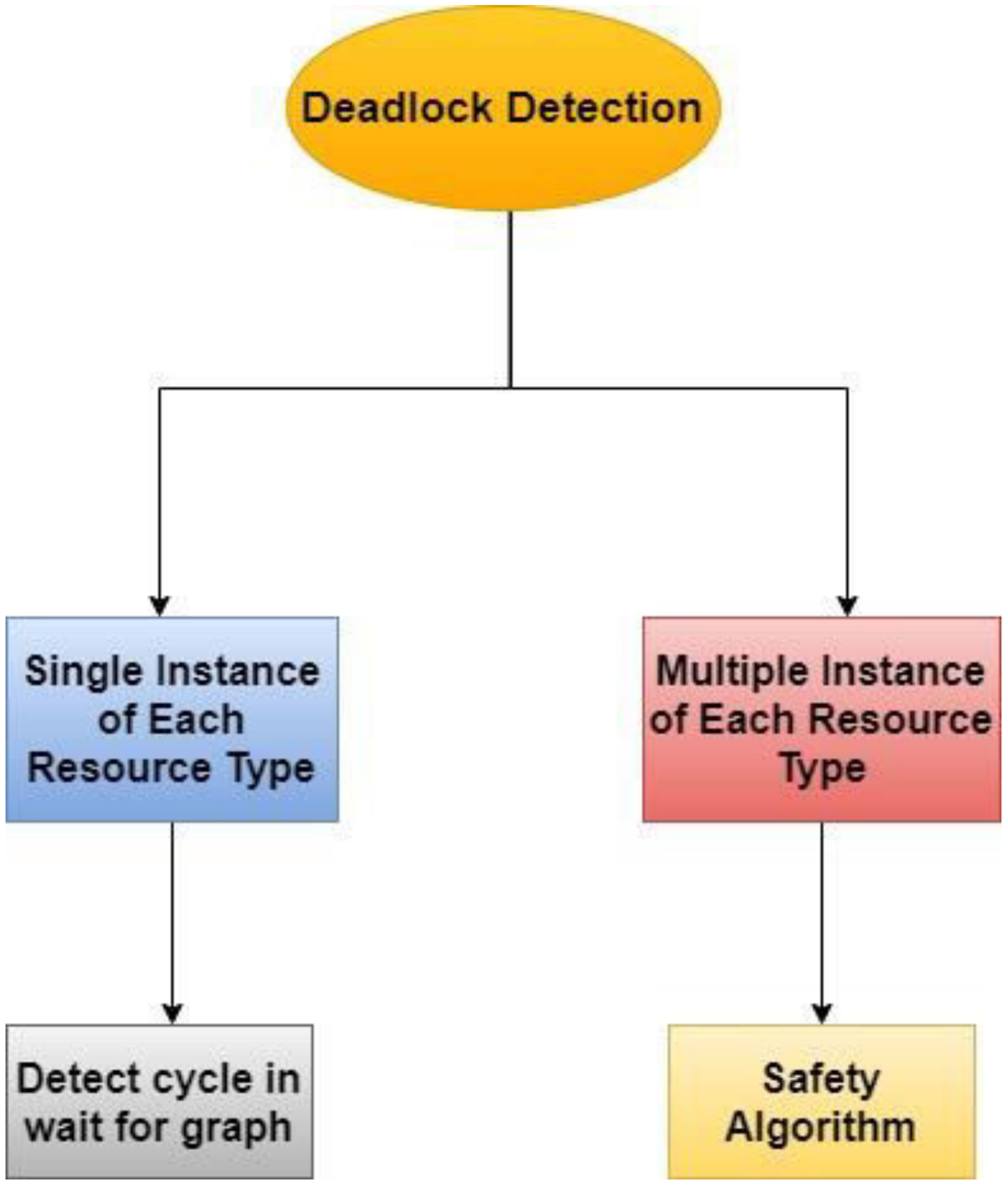
Resource-Allocation-Graph Algorithm





Suppose that P_2 requests R_2. Although R_2 is currently free, we cannot allocate it to P_2, since this action will create a cycle in the graph. A cycle, as mentioned, indicates that the system is in an unsafe state. If P_2 requests R_2, and P_2 requests R_1, then a deadlock will occur.

Resource-Allocation-Graph Algorithm


Deadlock Detection
Single Instance of Each Resource Type
Multiple Instance of Each Resource Type
Detect cycle in wait for graph
Safety Algorithm

Banker's Algorithm

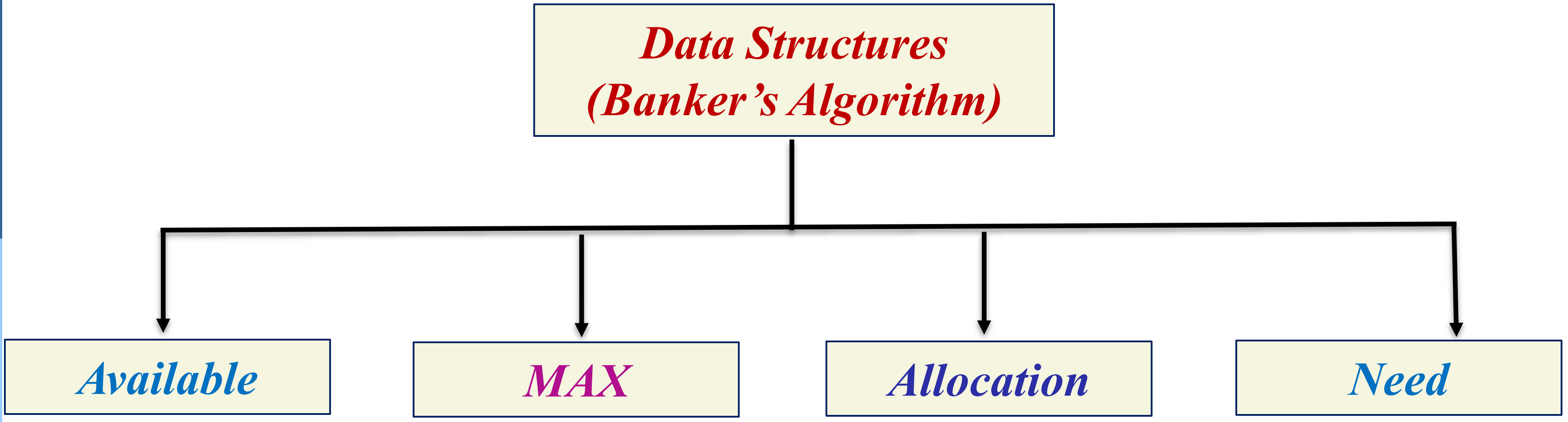
- *The resource-allocation-graph algorithm is* ***not applicable*** *to a resource allocation system* ***with multiple instances*** *of each resource type.*

- *The name banker's algorithm was chosen because the algorithm could be used in a banking system to ensure that the bank never allocate its available cash in such a way that it could no longer satisfy the needs of all its customers.*

Banker's Algorithm

- *Banker's Algorithm is a* ***deadlock avoidance*** *algorithm.*
- *It maintains a set of data using which it decides* ***whether to entertain the request of any process or not.***
- *It follows the safety algorithm to check whether the system is in a safe state or not.*

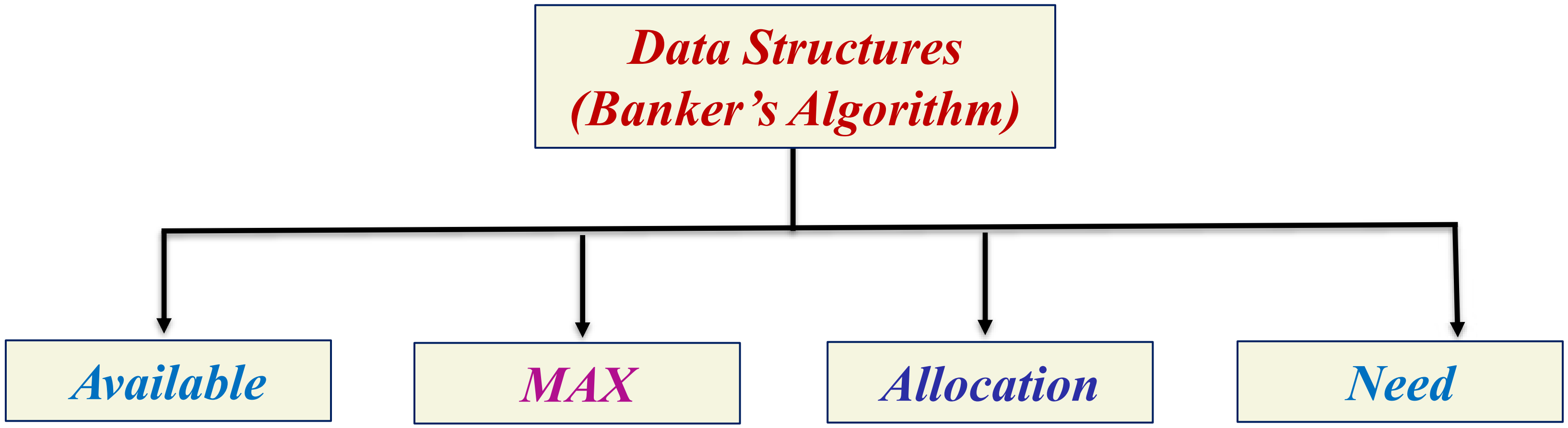
Data Structures - Banker's Algo.



*Let **n = number of processes**, and **m = number of resources types**.*

- ***Available:*** *If **available [j] = k**, there are **k** instances of resource type R_j are available.*
- ***Max:*** *An $n \times m$ matrix defines the maximum demand of each process. If **Max [i, j] = k,** then process P_i may request at most k instances of resource type R_j.*

Data Structures - Banker's Algo.



- ***Allocation:*** *An* $\mathbf{n} \times \mathbf{m}$ *matrix defines the number of resources of each type currently allocated to each process. If* $\mathbf{Allocation[i, j] = k}$ *then* $\mathbf{P_i}$ *is currently allocated* $\mathbf{k}$ *instances of* $\mathbf{R_j}$*.*

- ***Need:*** *An* $\mathbf{n} \times \mathbf{m}$ *matrix indicates the remaining resource need of each process. If* $\mathbf{Need[i, j] = k}$*, then* $\mathbf{P_i}$ *may need* $\mathbf{k}$ *more instances of* $\mathbf{R_j}$ *to complete its task.*

$$Need[i][j] = Max[i][j] - Allocation[i][j].$$

Safety Algorithm

Use to find out whether or not a system is in a safe state.

1. Let **Work** and **Finish** be vectors of length **m** and **n**, respectively. Initialize:

   Work = Available

   Finish [i] = false for i = 1,2,3, …, n.

2. Find an index i such that both:

   (a) Finish [i] == false

   (b) $Need_i \le Work$

   If no such i exists, go to step 4.

3. ***$Work = Work + Allocation_i$***

   Finish[i] := true

   go to step 2.

4. If ***Finish [i] = true*** for all i, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

Algorithm is used to determine whether requests can be safely granted

Let ***$Request_i$*** *is the request vector for process* ***P_i****. If* ***$Request_i[j] = k$*** *then process P_i wants k instances of resource type* ***R_j****. When a request for resources is made by process Pi , the following actions are taken:*

1. *If* ***$Request_i \leq Need_i$*** *go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.*
2. *If* ***$Request_i \leq Available$****, go to step 3. Otherwise P_i must wait, since resources are not available.*
3. *Pretend to allocate requested resources to P_i by modifying the state as follows:*

$$Available = Available - Request_i$$

$$Allocation_i = Allocation_i + Request_i$$

$$Need_i = Need_i - Request_i$$

- ***If safe*** *$\Rightarrow$ the resources are allocated to P_i.*
- ***If unsafe*** *$\Rightarrow$ P_i must wait, and the old resource-allocation state is restored*

Banker's Algorithm Example-1

Example: *Consider a system that contains **five processes P1, P2, P3, P4, P5** and the three resource types **A, B** and **C**. Following are the resources types: **A has 10, B has 5** and the resource type **C has 7** instances.*

Process	*Allocation*			*Max*			*Available*			*Need*		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P1	0	1	0	7	5	3						
P2	2	0	0	3	2	2						
P3	3	0	2	9	0	2						
P4	2	1	1	2	2	2						
P5	0	0	2	4	3	3						

1. What is the reference of the need matrix?

2. Determine if the system is safe or not.

3. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Banker's Algorithm Example-1

Example: *Consider a system that contains **five processes P1, P2, P3, P4, P5** and the three resource types **A, B** and **C**. Following are the resources types: **A has 10**, **B has 5** and the resource type **C has 7** instances.*

Process	*Allocation*			*Max*			*Available*			*Need*		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

1. What is the reference of the need matrix?

2. Determine if the system is safe or not.

3. What will happen if the resource request (1, 0, 0) for process P1 can the system accept this request immediately?

Safety Algorithm

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

Step 1: work = [3 3 2],

Finish [1] = False, Finish [2] = false, Finish [3] = false, Finish [4] = false, Finish [5] = false

Step 2: *Find an index i such that* ***Finish [i] == false, Need$_i$ <= Work***

For: i = 1, [7 4 3] <= [3 3 2]

For: i = 2, [1 2 2] <= [3 3 2]

Work = [3 3 2] + [2 0 0] = [5 3 2], Finish[2] = True

For: i = 3, [6 0 0] <= [5 3 2]

For: i = 4, [0 1 1] <= [5 3 2]

Work = [5 3 2] + [2 1 1] = [7 4 3], Finish[4] = True

For: i = 5, [4 3 1] <= [7 4 3]

Work = [7 4 3] + [0 0 2] = [7 4 5], Finish[5] = True

For: i = 1, [7 4 3] <= [7 4 5]

Work = [7 4 5] + [0 1 0] = [7 5 5], Finish[1] = True

For: i = 3, [6 0 0] <= [7 5 5]

Work = [7 5 5] + [3 0 2] = [10 5 7],

Finish[3] = True

System is in safe state and safe sequence is <P2 P4 P5 P1 P3>

Safety Algorithm

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

Resource-Request Algorithm for Process P_i

*What will happen if the process **P1** request **(1, 0, 0)** resources, can the system accept this request immediately?*

Step 1: *If $\mathbf{Request_1 \le Need_1}$ go to step 2. $\mathbf{[1\ 0\ 0] \le [7\ 4\ 3]}$*

Step 2: *If $\mathbf{Request_1 \le Available}$, go to step 3. $\mathbf{[1\ 0\ 0] \le [3\ 3\ 2]}$*

Step 3: ***Pretend*** *to allocate requested resources to P_1 by modifying the state as follows:*

$$Available = Available - Request_1$$

$$Allocation_i = Allocation_1 + Request_1$$

$$Need_i = Need_1 - Request_1$$

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	1	1	0	7	5	3	2	3	2	6	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

Safety Algorithm

Process	*Allocation*			*Max*			*Available*			*Need*		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P1	1	1	0	7	5	3	2	3	2	6	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

Step 1: ***work = [2 3 2],***

Finish [1] = False, Finish [2] = false, Finish [3] = false, Finish [4] = false, Finish [5] = false

Step 2: Find an index i such that ***Finish [i] == false, Need$_i$ <= Work***

For: i = 2, [1 2 2] <= [2 3 2]

For: i = 1, [6 4 3] <= [2 3 2]

Work = [2 3 2] + [2 0 0] = [4 3 2], Finish[2] = True

For: i = 4, [0 1 1] <= [4 3 2]

For: i = 3, [6 0 0] <= [4 3 2]

Work = [4 3 2] + [2 1 1] = [6 4 3], Finish[4] = True

For: i = 5, [4 3 1] <= [6 4 3]

Work = [6 4 3] + [0 0 2] = [6 4 5], Finish[5] = True

For: i = 1, [6 4 3] <= [6 4 5]

Work = [6 4 5] + [1 1 0] = [7 5 5], Finish[1] = True

For: i = 3, [6 0 0] <= [7 5 5]

Work = [7 5 5] + [3 0 2] = [10 5 7],

Finish[3] = True

System is in safe state and safe sequence

is <P2 P4 P5 P1 P3>

Banker's Algorithm

An operating system uses the banker's algorithm for deadlock avoidance when managing the allocation of three resource types ***X, Y and Z*** *to three processes* ***P0, P1 and P2****. The table given below presents the current system state. Here, the Allocation matrix shows the current number of resources of each type allocated to each process and the Max matrix shows the maximum number of resources of each type required by each process during its execution.*

	Allocation			**Max**		
	X	**Y**	**Z**	**X**	**Y**	**Z**
P0	0	0	1	8	4	3
P1	3	2	0	6	2	0
P2	2	1	1	3	3	3

There are ***3 units of type X, 2 units of type Y and 2 units of type Z*** *still* ***available****. The system is currently in safe state. Consider the following independent requests for additional resources in the current state-*

REQ1: *P0 requests 0 units of X, 0 units of Y and 2 units of Z*

REQ2: *P1 requests 2 units of X, 0 units of Y and 0 units of Z*

Banker’s Algorithm (Request-1)

	Allocation			*Max*			*Need*		
	X	***Y***	***Z***	***X***	***Y***	***Z***	***X***	***Y***	***Z***
P0	*0*	*0*	*1*	*8*	*4*	*3*	*8*	*4*	*2*
P1	*3*	*2*	*0*	*6*	*2*	*0*	*3*	*0*	*0*
P2	*2*	*1*	*1*	*3*	*3*	*3*	*1*	*2*	*2*

Checking Whether REQ1 Can Be Entertained-

- *Need of P0 = [0 0 2], Available = [3 2 2]*

Clearly,

- *With the instances available currently, the requirement of REQ1 can be satisfied.*
- *So, banker's algorithm assumes that the request REQ1 is entertained.*

	Allocation			**Max**			**Need**		
	X	**Y**	**Z**	**X**	**Y**	**Z**	**X**	**Y**	**Z**
P0	0	0	3	8	4	3	8	4	**0**
P1	3	2	0	6	2	0	3	0	0
P2	2	1	1	3	3	3	1	2	2

Available = [3 2 2] – [0 0 2]

= [3 2 0]

- *Now, it follows the safety algorithm to check whether this resulting state is a safe state or not.*
- *If it is a safe state, then REQ1 can be permitted otherwise not.*

Checking Whether REQ1 Can Be Entertained-

- *With the instances available currently, only the requirement of process **P1** can be satisfied.*
- *So, process P1 is allocated the requested resources.*
- *It completes its execution and then free up the instances of resources held by it.*

  Then- Available

  = [3 2 0] + [3 2 0]

  = [6 4 0]
- *It is not possible to entertain any process.*
- *The system has entered the unsafe state.*
- *Thus, REQ1 will not be permitted.*

Checking Whether REQ2 Can Be Entertained-

- Need of P1 = [2 0 0]
- Available = [3 2 2]

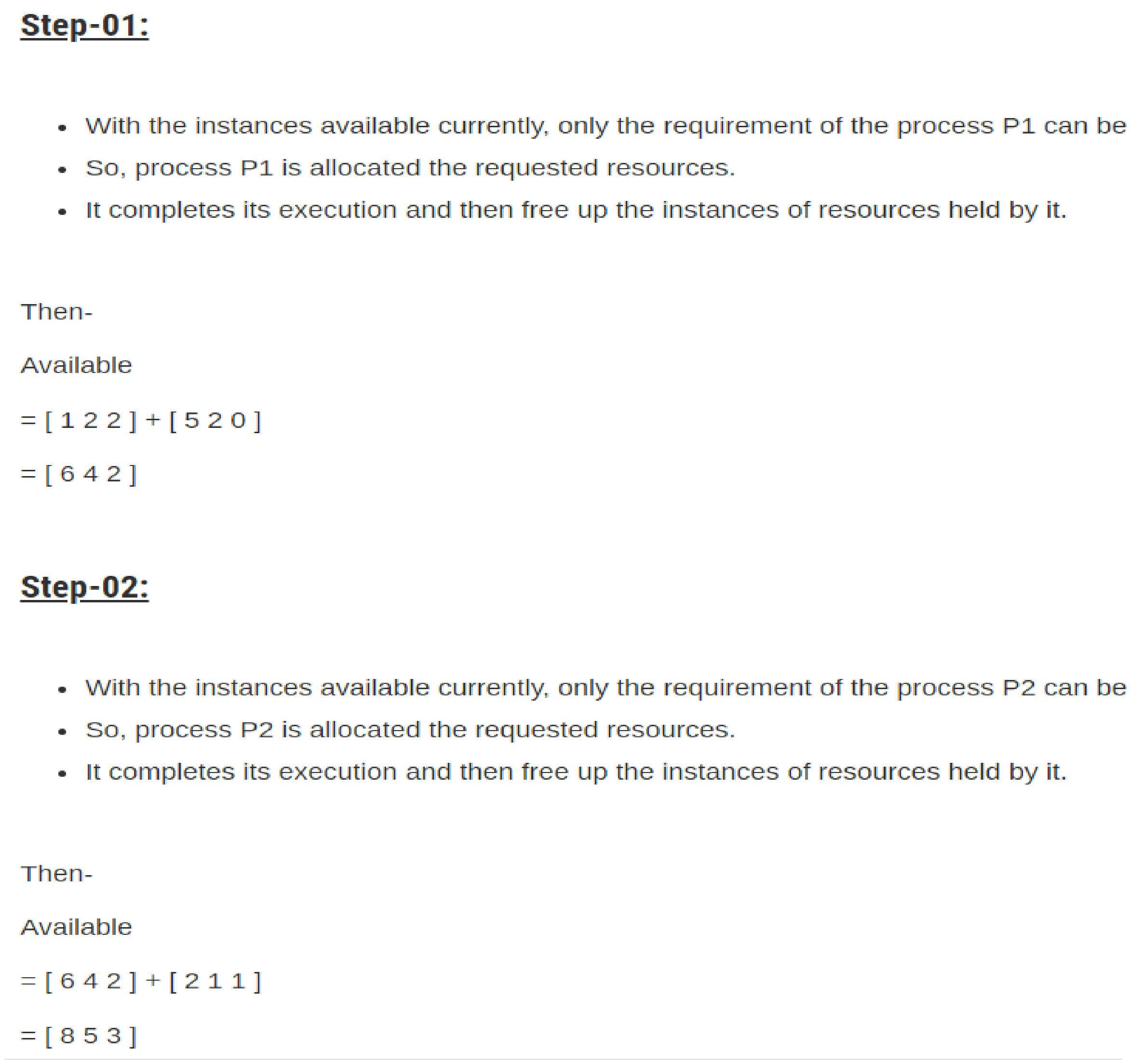
	Allocation			Max			Need		
	X	Y	Z	X	Y	Z	X	Y	Z
P0	0	0	1	8	4	3	8	4	2
P1	**5**	2	0	6	2	0	**1**	0	0
P2	2	1	1	3	3	3	1	2	2

Available

= [3 2 2] – [2 0 0]

= [1 2 2]

Banker's Algorithm (Request-2)



Step-01:

- With the instances available currently, only the requirement of the process P1 can be satisfied.
- So, process P1 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [1 2 2] + [5 2 0]

= [6 4 2]

Step-02:

- With the instances available currently, only the requirement of the process P2 can be satisfied.
- So, process P2 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [6 4 2] + [2 1 1]

= [8 5 3]

Banker's Algorithm (Request-2)

Step-03:

- With the instances available currently, the requirement of the process P0 can be satisfied.
- So, process P0 is allocated the requested resources.
- It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [8 5 3] + [0 0 1]

= [8 5 4]

Thus,

- There exists a safe sequence P1, P2, P0 in which all the processes can be executed.
- So, the system is in a safe state.

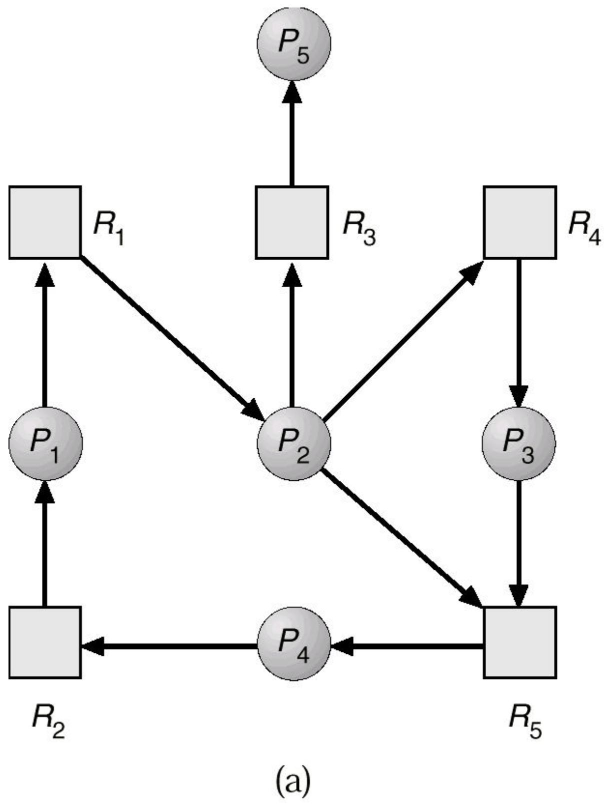
Deadlock Detection

- *Allow system to enter deadlock state*
- *Detection algorithm*
- *Recovery scheme*

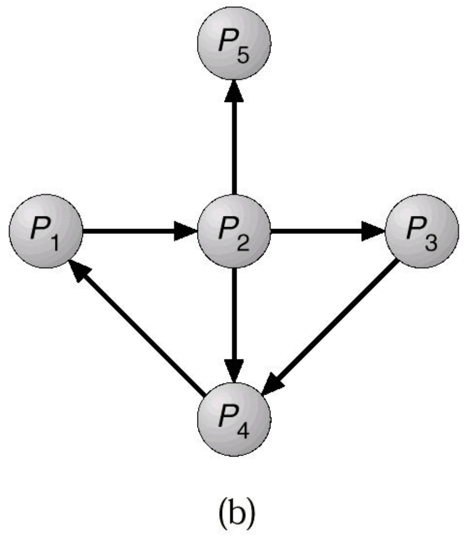
Single Instance of Each Resource Type

- *Maintain* ***wait-for graph***
  - *Nodes are processes.*
  - $P_i \rightarrow P_j$ *if* P_i *is waiting for* P_j.
- *Periodically invoke an algorithm that searches for a cycle in the graph. If cycle is there in the graph it means deadlock exist.*
- *An algorithm to detect a cycle in a graph requires an order of* n^2 *operations, where* n *is the number of vertices in the graph.*

Resource-Allocation Graph And Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

Several Instances of a Resource Type

- ***Available:*** *A vector of length m indicates the number of available resources of each type.*
- ***Allocation:*** *An n * m matrix defines the number of resources of each type currently allocated to each process.*
- ***Request:*** *An n * m matrix indicates the current request of each process. If Request [i] [j] = k, then process P_i is requesting k more instances of resource type. R_j.*

Detection Algorithm

1. *Let Work and Finish be vectors of length m and n, respectively Initialize:*

   (a) *Work = Available*

   (b) *For i = 1, 2, …, n, if* $Allocation_i \neq 0$*, then*

   Finish[i] := false; *otherwise, Finish[i] := true.*

2. *Find an index i such that both:*

   (a) *Finish[i] = false*

   (b) $Request_i \leq Work$

   If no such i exists, go to step 4.

Detection Algorithm (Cont.)

3. *Work* := *Work* + $Allocation_i$

   Finish[*i*] := *true*

   go to step 2.

4. ***If Finish[i] = false, for some i, 1 ≤ i ≤ n, then the system is in deadlock state.***

   Moreover, if Finish[i] = false, then P_i is deadlocked.

Algorithm requires an order of m x n^2 operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- *Five processes **P_0** through **P_4**; three resource types A (7 instances), B (2 instances), and C (6 instances).*
- *Snapshot at time T_0:*

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- *Sequence <P_0, P_2, P_3, P_4, P_1> will result in Finish[i] = true for all i.*

Example (Cont.)

- *P_2 requests an additional instance of type C.*

	Allocation	*Request*
	A B C	*A B C*
P_0	0 1 0	*0 0 0*
P_1	2 0 0	*2 0 1*
P_2	3 0 3	*0 0 1*
P_3	2 1 1	*1 0 0*
P_4	0 0 2	*0 0 2*

- *State of system?*
  - *Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests.*
  - *Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4.*

Detection-Algorithm Usage

- *When, and how often, to invoke depends on:*
  - *How often a deadlock is likely to occur?*
  - *How many processes will need to be rolled back?*
    - *one for each disjoint cycle*
- *If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.*

Deadlock Recovery: Process Termination

- *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.*
  - *Whether the process is interactive or batch.*

Recovery from Deadlock:Resource Preemption

- ***Selecting a victim*** *– minimize cost.*
- ***Rollback*** *– return to some safe state, restart process from that state.*
- ***Starvation*** *– same process may always be picked as victim, include number of rollback in cost factor.*

Combined Approach to Deadlock Handling

- *Combine the three basic approaches*
  - ***prevention***
  - ***avoidance***
  - ***detection***
- *allowing the use of the optimal approach for each of resources in the system.*
- *Partition resources into hierarchically ordered classes.*
- *Use most appropriate technique for handling deadlocks within each class.*

Combined Approach to Deadlock Handling

A system has 4 processes and 5 allocatable resource. The current allocation and maximum needs are as follows-

	Allocated					**Maximum**				
A	1	0	2	1	1	1	1	2	1	3
B	2	0	1	1	0	2	2	2	1	0
C	1	1	0	1	1	2	1	3	1	1
D	1	1	1	1	0	1	1	2	2	0

If Available = [0 0 X 1 1], what is the smallest value of x for which this is a safe state?