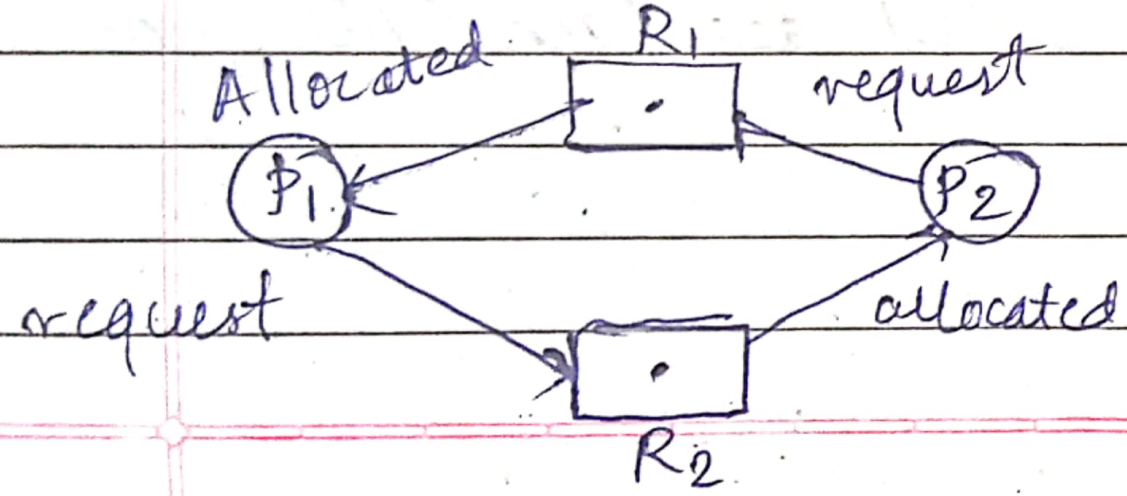
DEADLOCK

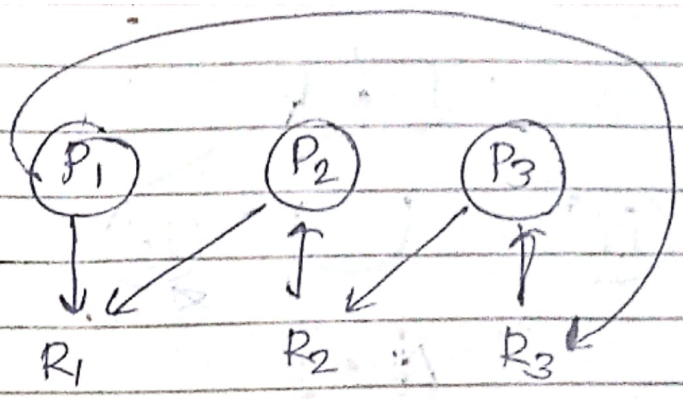
If 2 or more processes are waiting on happening of some event which never happened we say that these processes are on a deadlock state.



$R_1 \rightarrow P_1$

R_1 alloc. to P_1

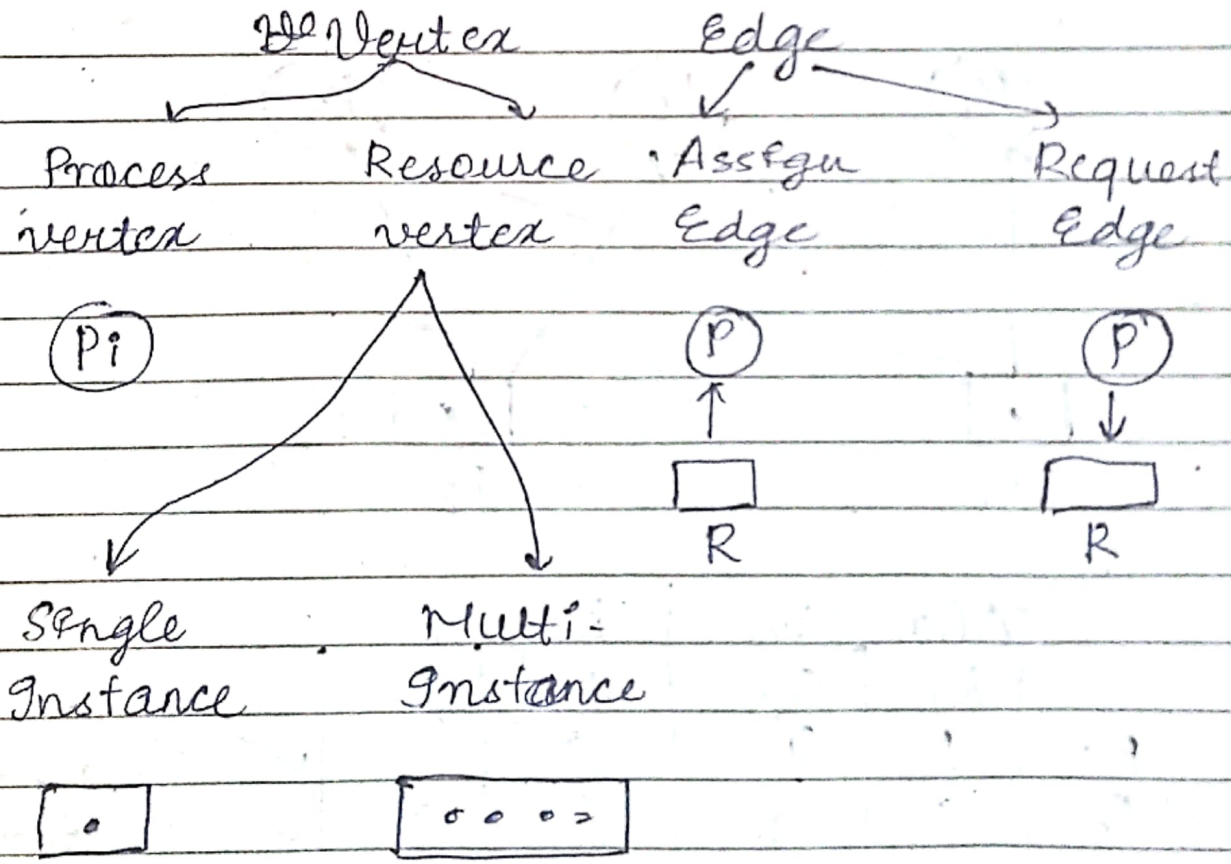
P_2 req. for R_1

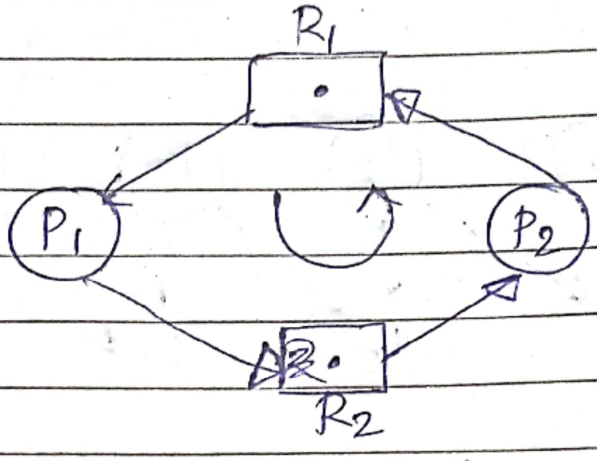


Necessary conditions for deadlock.

1) Mutual Exclusion
2) No pre-emption
3) Hold & Wait
4) Circular Wait

Resource Allocation graph on Deadlock (RAG)





Circular wait

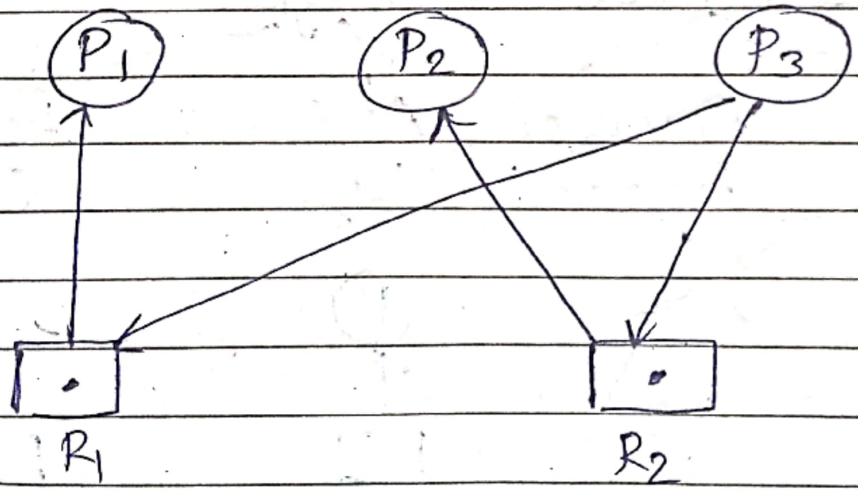
Single Instance

	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

Availability (Deadlock method)

(0, 0)

R_1, R_2

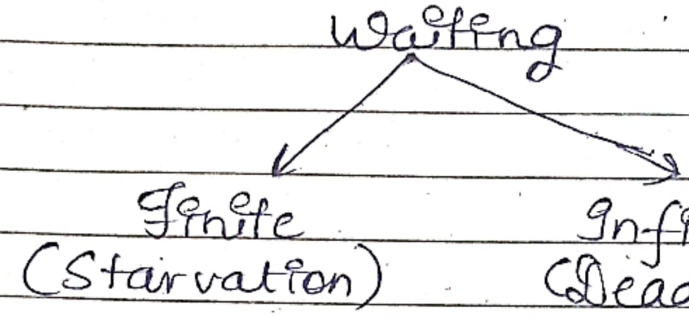


	Allocation		Request		
	R_1	R_2	R_1	R_2	
P_1	1	0	0	0	✓ Terminate
P_2	0	1	0	0	✓ —"—
P_3	0	0	1	1	

Page No.:
Date: / /

Availability (0, 0)
(1, 0) → after P_1 terminates
(1, 1) → —"— P_2 —"—

No deadlock occurs.

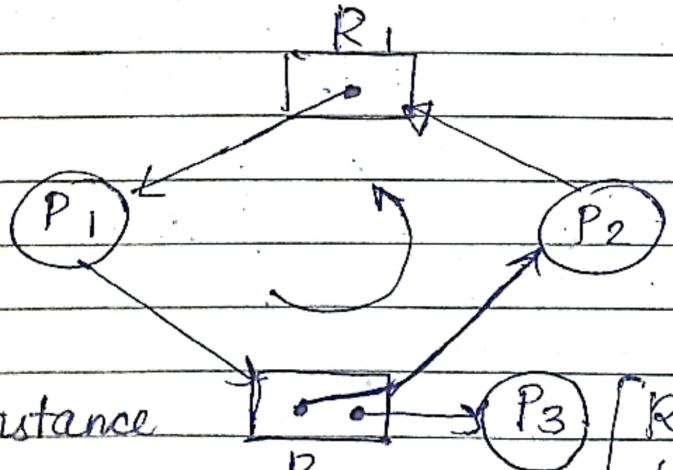


If RAG Circular wait → Always in Deadlock
↓
Single Instance

Single Instance + Circular weight = Deadlock

Mult-Instance + CW = Possible (May or may not be)

Multi-Instance RAG



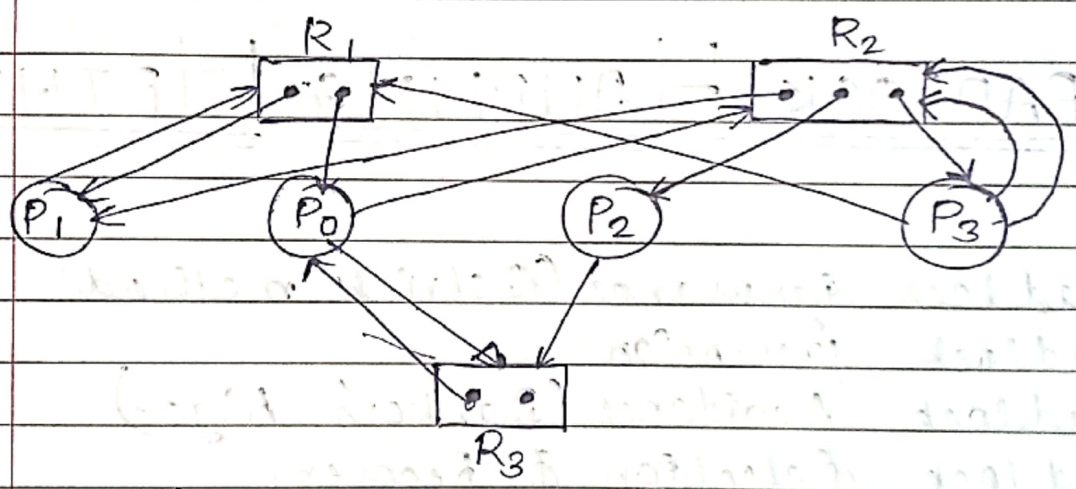
Mult-Instance RAG

Resource shared by multiple instances

Process	Allocate		Request		
	R_1	R_2	R_1	R_2	
P_1	1	0	0	1	✓
P_2	0	1	1	0	
P_3	0	1	0	0	✓

Current availability → (0, 0)
(0, 1)
(1, 0)

No deadlock



	Allocate			Request		
	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1
P_1	1	1	0	1	0	0
P_2	0	1	0	0	0	1
P_3	0	1	0	1	2	0

(R_1, R_2, R_3)

cur. avail. = (0 0 1)

$P_2 \rightarrow$ 0 1 0

0 1 1

$P_0 \rightarrow$ 1 0 1

1 1 2

	R_1	R_2	R_3
	1	1	2
$P_1 \rightarrow$	1	1	0
	2	2	2
$P_3 \rightarrow$	0	1	0
	2	3	2 → Curr. availability

No deadlock in the system:

$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

DEADLOCK HANDLING METHODS

1) Deadlock ignorance (Ostrich method)
2) Deadlock prevention
3) Deadlock Avoidance (Banker's Algo.)
4) Deadlock Detection & Recovery

Deadlock ignorance:

(Just ignore the deadlock)

Deadlock occurs very rarely. ~~In windows~~ Windows has lots of code in its OS. We want more & more speed so we don't write any code for deadlock, so its easy to avoid as it's rare.

Why is this method called Ostrich method? Whenever there's a sandstorm Ostrich puts its head in the sand assuming no sand. In the same way we ignore deadlock so that the performance & speed doesn't get degraded.

Deadlock prevention

Before the deadlock occurs find the prevention of necessary conditions for deadlock

1) Mutual Exclusion
2) No preemption
3) Hold & wait
4) Circular wait

~~Eg:~~ Either remove all the conditions or try to false any one situation.

1) Mutual Exclusion (There should be no sharing bet^n diff. processes at the same time)

So if we make all the resources shareable we can remove mutual exclusion but some resources like printer can't be made shareable

2) No premption (There shouldn't be any pre-emption in bet^n the resources)

So, if we make the process pre-empted using timestamp or TO we can prevent deadlock.

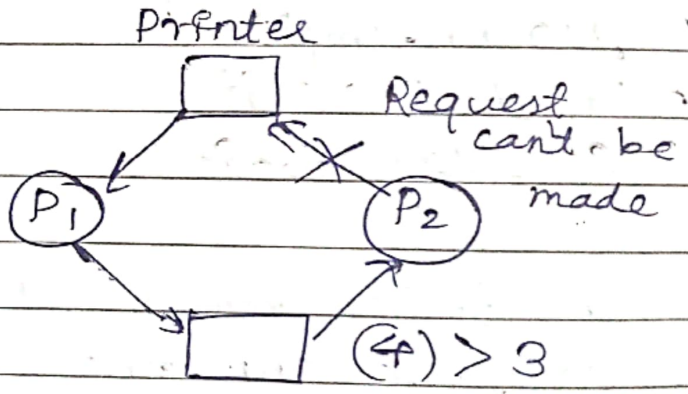
3) Hold & wait (Holding some resource & waiting for some resource)

Before the process starts, give all the resources to the process.

4) Circular wait

Order all the resources (like numbering) whenever a process request for a resource it would be ordered in increasing order. A process can request in increasing ~~order~~ order.

1) Printer
2) Scanner
3) CPU
4) Register



Deadlock Avoidance

While giving the resources to the process check whether it's safe or not

Deadlock detection & recovery

S-1 : First check whether there's a deadlock or not using RAG (resource allocation graph)

S-2 : After detection do some recovery
1) Kill the processes or a process (lower priority resources)
2) ~~Resource~~ Resource preemption (pre-empt the resource it's holding)

4.5 Deadlock Avoidance

(Banker's Algorithm)

Deadlock Avoidance Algorithm

While providing the resources to the process we check if deadlock occurs or not. It is also called Deadlock prevention.

	Max instances	Already Allocated
CPU(A)	= 10	7
Memory(B)	= 5	2
Printer(C)	= 7	5

(Total AT) Allocation → how much is allocated
Max need → how much is needed

(Total AT - A) availability → how much resource is available

(Max need - Allocation) Remaining need

Process	Allocation			Max Need			Available			Remaining-need Max - Allocation		
							10-7	5-2	7-5			
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	$3_{(2)}$	$3_{(0)}$	$2_{(0)}$	7	4	3
P_2	2	0	0	3	2	2	$5_{(2)}$	$3_{(1)}$	$2_{(1)}$	1	2	2
P_3	3	0	2	9	0	2	$7_{(0)}$	$4_{(0)}$	$3_{(2)}$	6	0	0
P_4	2	1	1	7	2	2	$7_{(0)}$	$4_{(1)}$	$5_{(0)}$	2	1	1
P_5	0	0	2	5	3	3	$7_{(3)}$	$5_{(0)}$	$5_{(2)}$	5	3	1
+												
	7	2	5				10	5	7			

If we can't fulfil the need of every process it's called deadlock

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ (Safe sequence)

(Sequence having no deadlock)

(Not possible on real life)