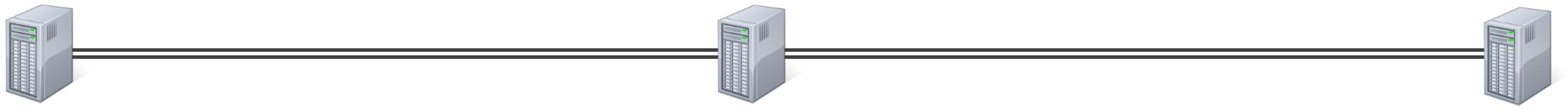
Deadlocks

(Operating System)



Deepika H V

C-DAC Bengaluru

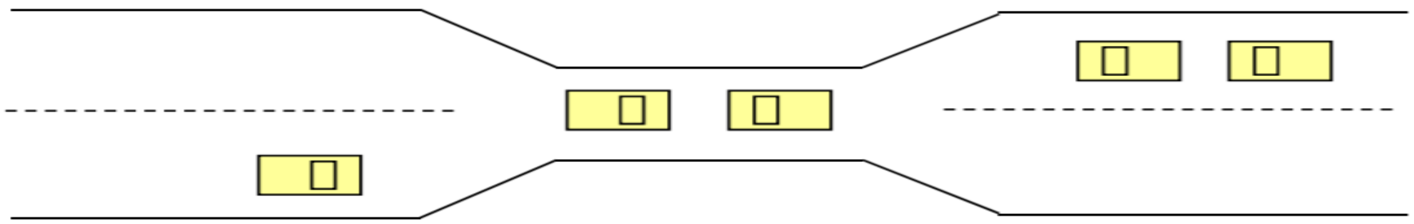
(deepikahv@cdac.in)

Agenda

- What is a Deadlock?
- Why does it happen?
- How can you detect deadlock happen?
- Consequences
- Dealing with deadlocks
- Semaphores
- Mutex
- Producer consumer problem
- Deadlock vs Starvation

What is a deadlock?

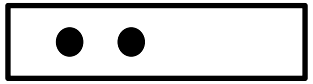
- **Definition**
  - When a waiting process is never again able to change state because the resource requested is held by other waiting process. This situation is **Deadlock**

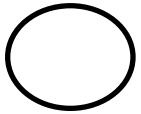


- **Resources in a system**
  - CPU cycles ; I/O devices (printers, tape drives) ; database (tables)
  - Two types – preemptable and non-premptable

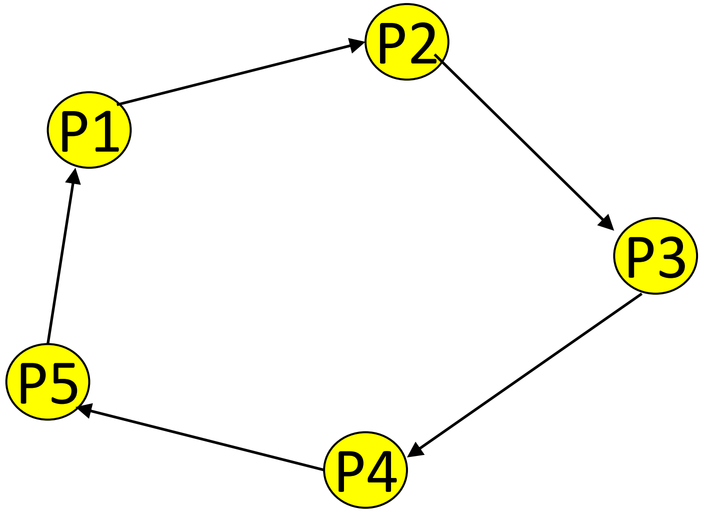
- **How to access a resource**
  - Sequence of events : (a) Request resource (b) Use resource (c) release resource
- **Must wait if resource is denied**
  - Requesting process is blocked
  - May fail with error code
- **Identify Deadlock (4 conditions)**
  - Mutual exclusion condition - each resource assigned to 1 process or is available
  - Hold and wait condition - process holding resources can request additional
  - No preemption condition - previously granted resources cannot forcibly taken away
  - Circular wait condition - must be a circular chain of 2 or more processes ; each process is waiting for resource held by next process of the chain

Graph Theoretic Models

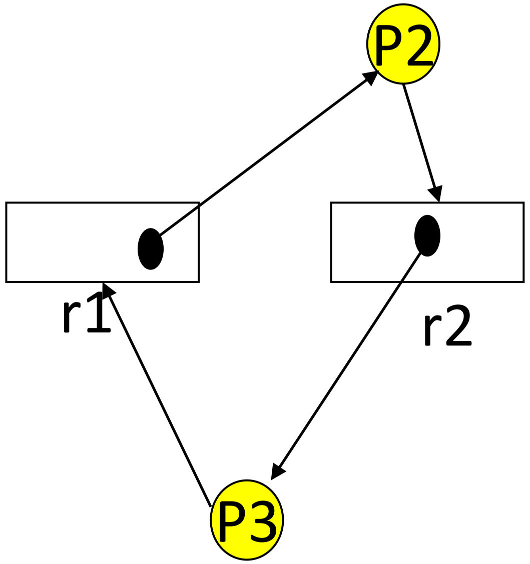
- **Resources :** 

- **Process:** 

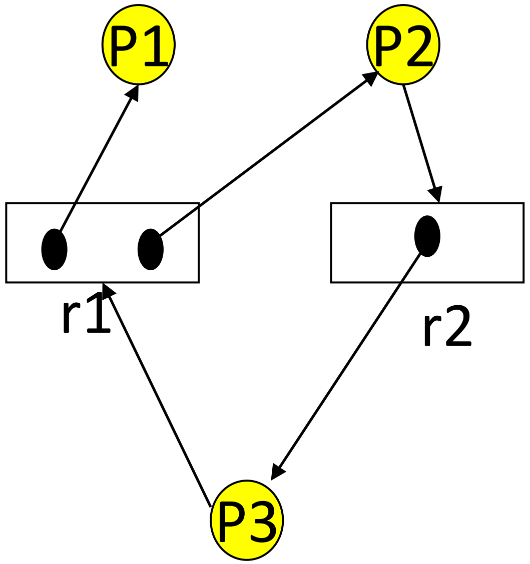
- **Wait for Graph**



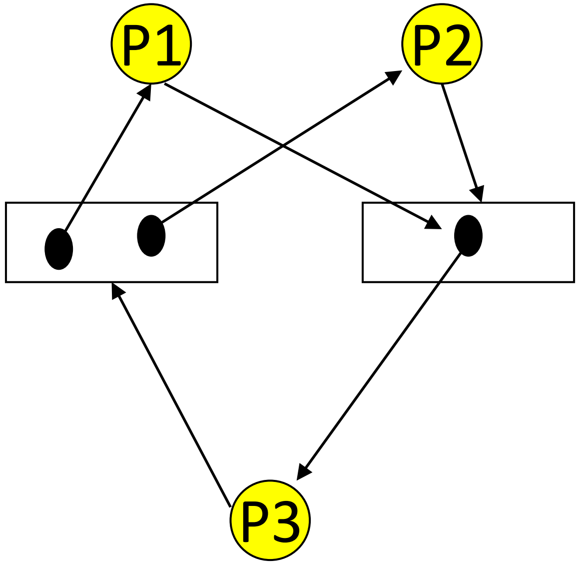
- **Resource Allocation Graph**



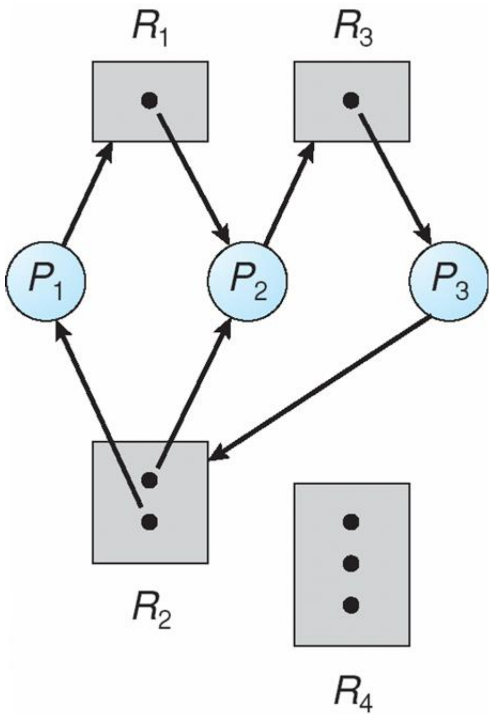
- **Resource Allocation Graph**



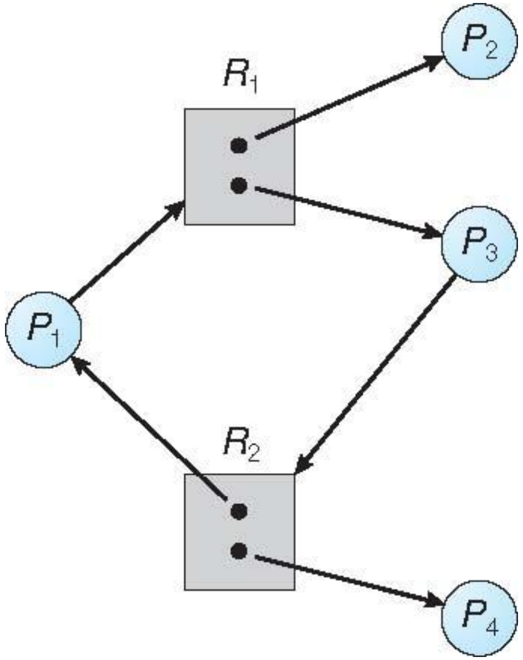
(A)



(B)



(C)



(D)

- **Which is deadlock and not?**

Strategies

- **Handling Deadlocks**
  - Ignorance
  - Prevention
  - Avoidance
  - Detection and recovery

Ignore

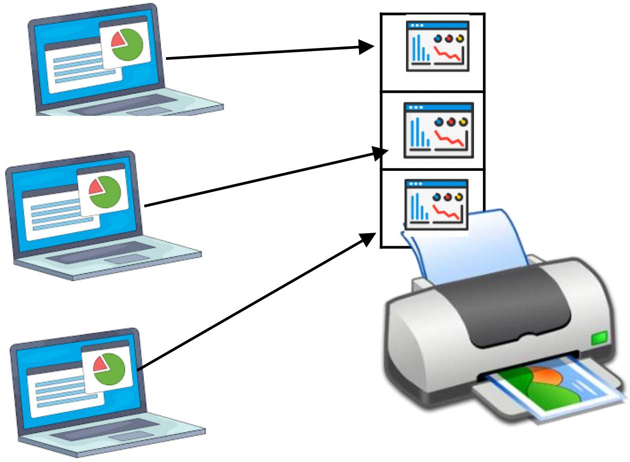
- **Ostrich Algorithm**
  - Pretend there's no problem
  - Reasonable if
    - Deadlocks occur very rarely
    - Cost of prevention is high
  - UNIX and Windows take this approach
    - Resources (memory, CPU, disk space) are plentiful
    - Deadlocks over such resources rarely occur
    - Deadlocks typically handled by rebooting
  - Trade off between convenience and correctness

Prevent

- **Ensure that at least one of the conditions for deadlock never occurs**
  - Mutual exclusion
  - No preemption
  - Hold & wait
  - Circular wait

1. Eliminate Mutual Exclusion

- **Cause for Mutual Exclusion**
  - Mutual Exclusion happens with non-sharable resources
- **How to handle mutex devices**
  - Spooling
    - Combination of buffering and queuing
    - Eg: Printer or tape drive
      - Only the printer daemon uses printer resource
      - This eliminates deadlock for printer
- **Not all devices can be spooled**



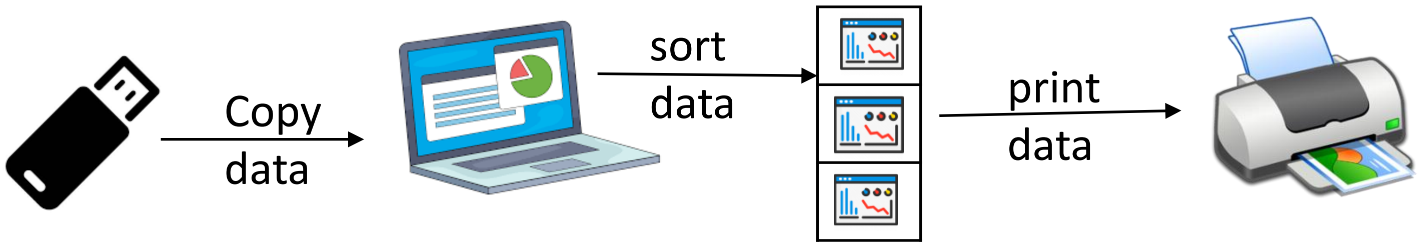
2. Preempt resources

- **Work for some resources**
  - Forcibly take away memory pages, suspending the process
  - Process may be able to resume with no ill effects
  - Eg : Round Robin scheduling of CPU cycles

- **This is not usually a viable option for all**
- **Consider a process given the printer**
  - Halfway through its job, take away the printer
  - Confusion ensues!

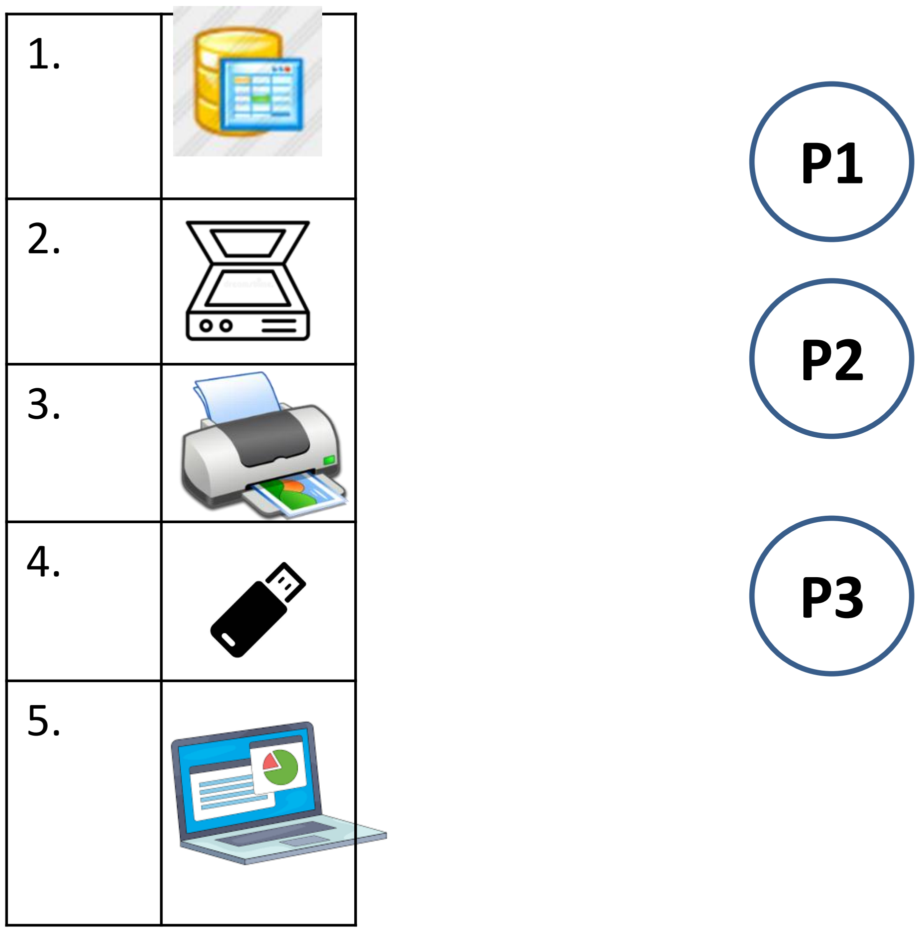
3. Stop Hold & Wait

- **Require processes to request resources before starting**
  - A process never has to wait for what it needs
- **Problem with the stratergy**
  - A process may not know required resources at start of run
    - Lower no. of resource request or over request resources
  - This also ties up resources other processes could be using
- **Variation:**
  - process must give up all held resources
  - Make a new request of resoureces required



4. No Circular Wait

- **Assign an order to resources**
- **Always acquire resources in numerical order**
  - Need not acquire them all at once!
- **Circular wait is prevented**



Prevention Summary

- **Mutual exclusion**
  - Spool everything
- **Hold and wait**
  - Request all resources initially
- **No preemption**
  - Take resources away
- **Circular wait**
  - Order resources numerically

Deadlock Avoidance

- **Do not grant a resource request if this allocation might lead to deadlock**
- **2 approaches**
  - do not start a process if its total demand might lead to deadlock: (“Process Initiation Denial”)
  - do not grant an incremental resource request if this allocation could lead to deadlock: (“Resource Allocation Denial”)
- **In both cases: maximum requirements of each resource must be stated in advance**
- **A Better Approach: Resource Allocation Denial**
  - Grant <u>incremental resource requests</u> if we can prove that this leaves the system in a state in which <u>deadlock cannot occur</u>.
  - Based on the concept of a “safe state”
  - Eg: Banker’s Algorithm

Banker's Algorithm

- **Steps involved in Banker's Algorithm**
  - Tentatively grant each resource request
  - Analyze resulting system state to see if it is "safe".
  - If safe, grant the request
  - if unsafe refuse the request (undo the tentative grant)
  - block the requesting process until it is safe to grant it.

The two states defined in Deadlock Avoidance

① Safe State

A system is in "safe state" if there is at least one assignment schedule to complete all the processes without deadlock and without any process releasing their resources

(deadlock will never happen even in the worst case)

➡ If a system is in safe state, deadlock can always be avoided

② Unsafe State

If a system is not in safe state, the system must be in "unsafe state"

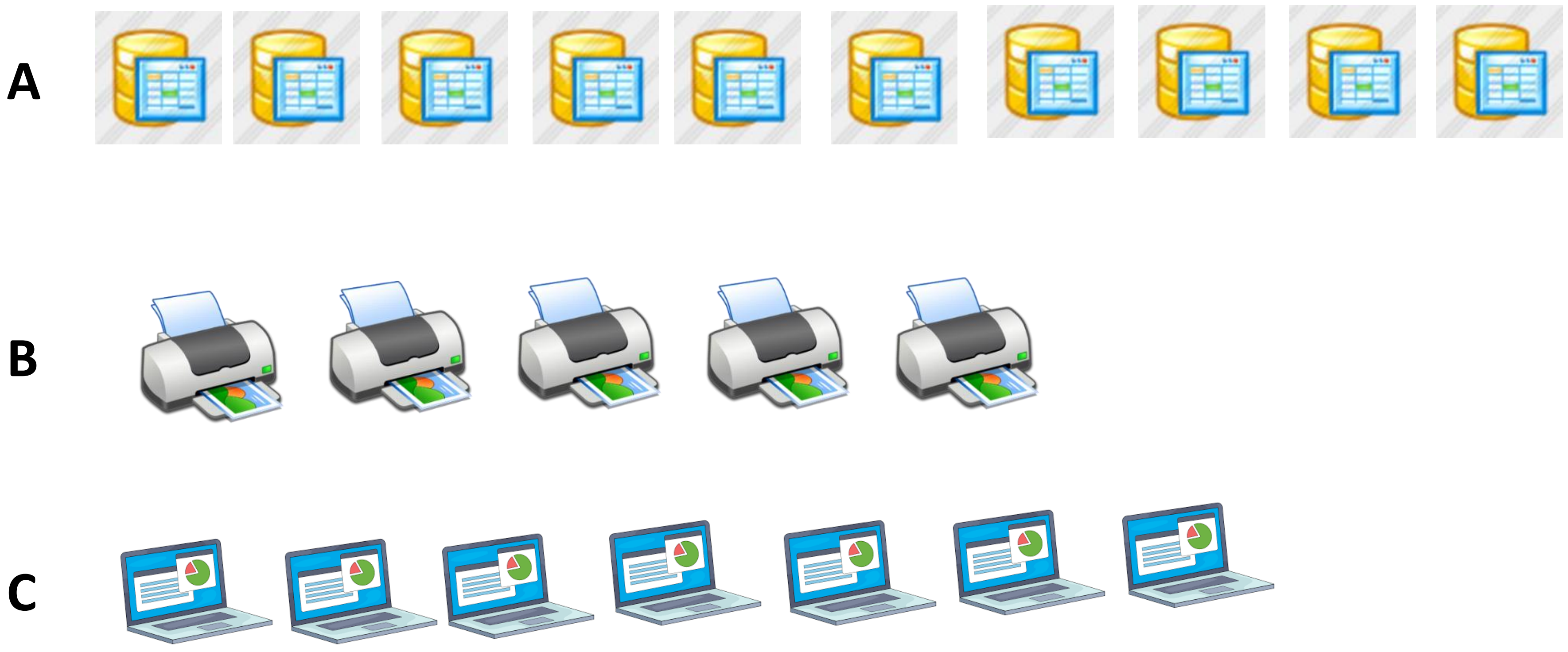
(deadlock can happen in the worst case)

➡ If a system is in unsafe state, deadlock may not be avoided

Courtesy :DR. Hiroshi Fujinoki, Southern Illinios Univerity Edwardsville

Sample Example

- **At Time T0**
  - Total Resources
    - A : 10
    - B : 5
    - C : 7



Sample Example

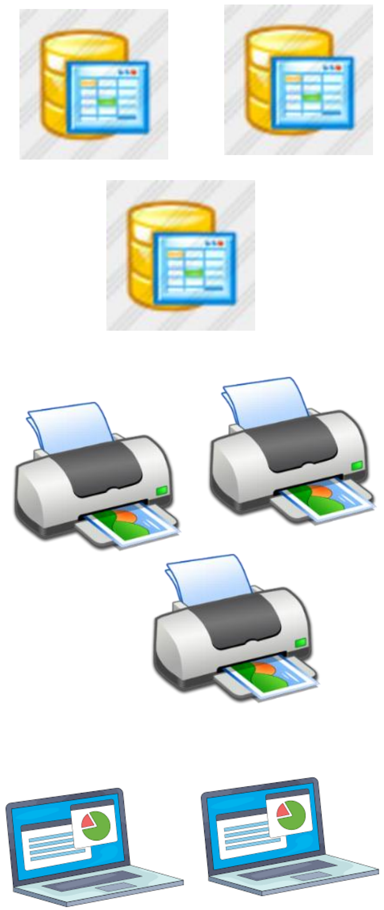
- **At Time T0**
  - Total Resources
    - A : 10
    - B : 5
    - C : 7
- **At Time T6**
  - Resource Avail
    - A : 3
    - B : 3
    - C : 2

AVAILABLE: 3 3 2		
Process	**Allocation**	**Max**
	A B C	**A B C**
P0	*0 1 0*	*7 5 3*
P1	*2 0 0*	*3 2 2*
P2	*3 0 2*	*9 0 2*
P3	*2 1 1*	*2 2 2*
P4	*0 0 2*	*4 3 3*

A

B

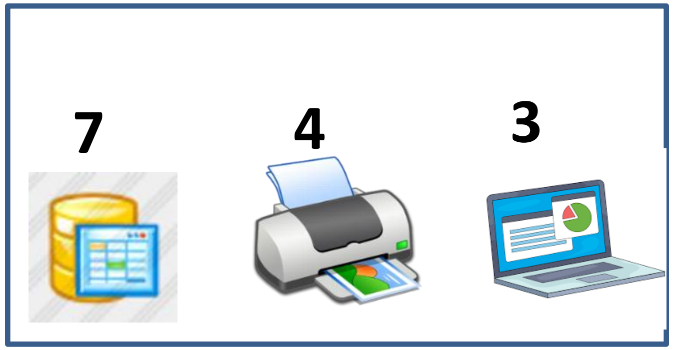
C



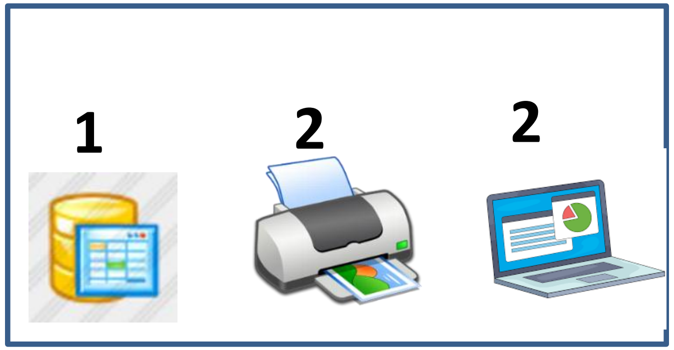
- **Need = Max – Allocation**
- **Pick P0**

AVAILABLE: *3 3 2*			
Process	**Allocation**	**Max**	**Need**
	A B C	**A B C**	**A B C**
P0	*0 1 0*	*7 5 3*	*7 4 3*
P1	*2 0 0*	*3 2 2*	*1 2 2*
P2	*3 0 2*	*9 0 2*	*6 0 0*
P3	*2 1 1*	*2 2 2*	*0 1 1*
P4	*0 0 2*	*4 3 3*	*4 3 1*

AVAILABLE: 3 3 2			
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

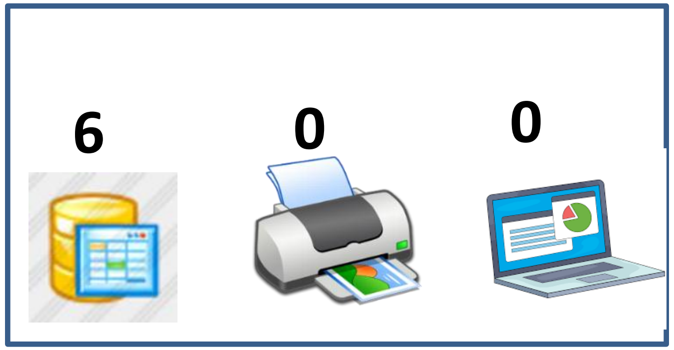


AVAILABLE: 3 3 2			
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	2 0 0	3 2 2	1 2 2
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1



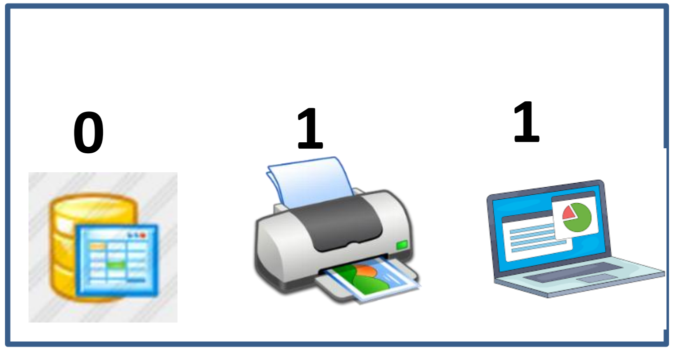
	3	3	2
-	1	2	2
Rem	2	1	0
Complete P1			
Rem	2	1	0
Ret +	3	2	2
Avail	5	3	2

AVAILABLE: 5 3 2			
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	0 0 0	3 2 2	0 0 0
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1



AVAILABLE: 5 3 2

Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	0 0 0	3 2 2	0 0 0
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1



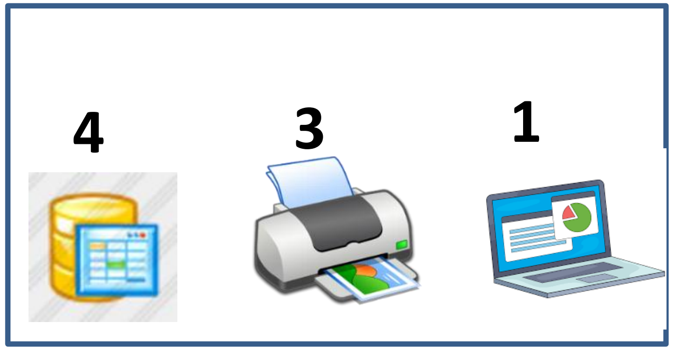
	5	3	2
-	0	1	1
Rem	5	2	1

Complete P3

Rem	5	2	1
Ret +	2	2	2
Avail	7	4	3

AVAILABLE: 7 4 3

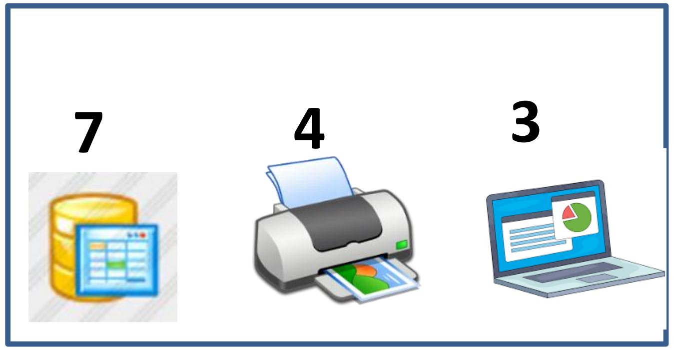
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	0 0 0	3 2 2	0 0 0
P2	3 0 2	9 0 2	6 0 0
P3	0 0 0	2 2 2	0 0 0
P4	0 0 2	4 3 3	4 3 1



7 4 3
\- 4 3 1
Rem 3 1 2
Complete P4
Rem 3 1 2
Ret + 4 3 3
Avail 7 4 5

AVAILABLE: 7 4 5

Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3
P1	0 0 0	3 2 2	0 0 0
P2	3 0 2	9 0 2	6 0 0
P3	0 0 0	2 2 2	0 0 0
P4	0 0 0	4 3 3	0 0 0

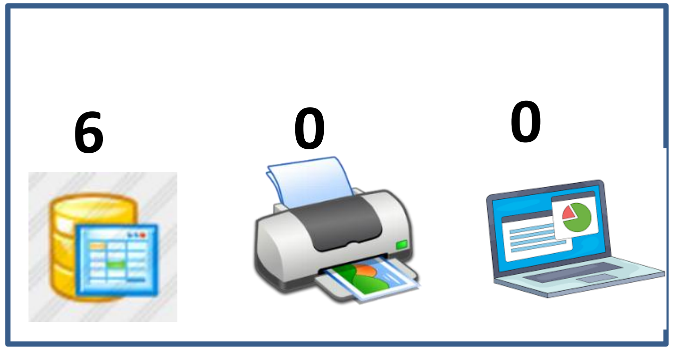


	7	4	5
-	7	4	3
Rem	0	0	2

Complete P0

Rem	0	0	2
Ret +	7	5	3
Avail	7	5	5

AVAILABLE: 755			
Process	Allocation	Max	Need
	A B C	A B C	A B C
P0	0 0 0	7 5 3	0 0 0
P1	0 0 0	3 2 2	0 0 0
P2	3 0 2	9 0 2	6 0 0
P3	0 0 0	2 2 2	0 0 0
P4	0 0 0	4 3 3	0 0 0



7 5 5
\- 6 0 0
Rem 1 5 5
Complete P2
Rem 1 5 5
Ret + 9 0 2
Avail 10 5 7

Four resources ABCD. A has 6 instances, B has 3 instances, C Has 4 instances and D has 2 instances.

Process	Allocation	Max
	ABCD	ABCD
P1	3011	4111
P2	0100	0212
P3	1110	4210
P4	1101	1101
P5	0000	2110

Is the current state safe?

If P5 requests for (1,0,1,0), can this be granted?

Deadlock Detection and Recovery

- **Allow the system to enter deadlock ; then detect and recover**
- **Detection**
  - For single instance of the resource using wait-for-graph
  - For multiple instance of resource use Banker’s Algorithm
- **Recovery**
  - Abort
  - Preempt

Thank You