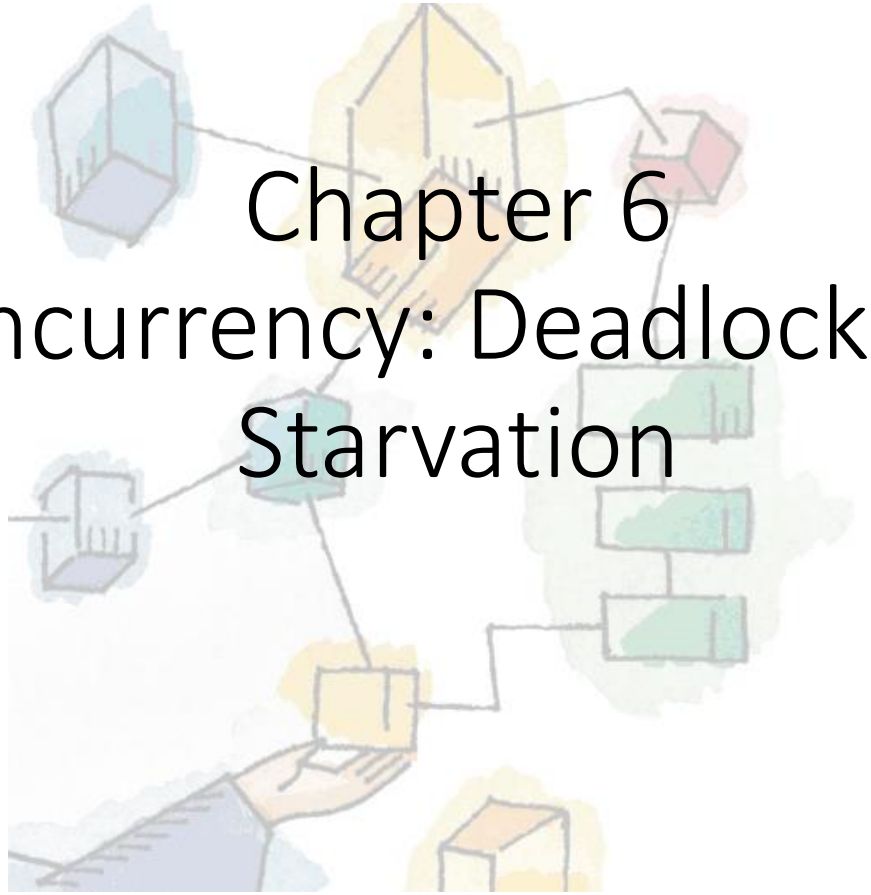
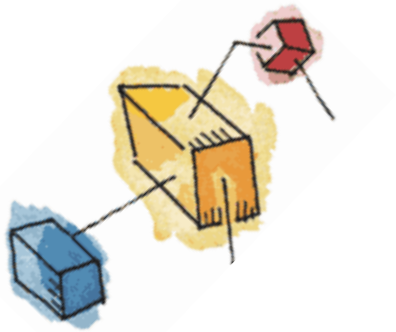
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



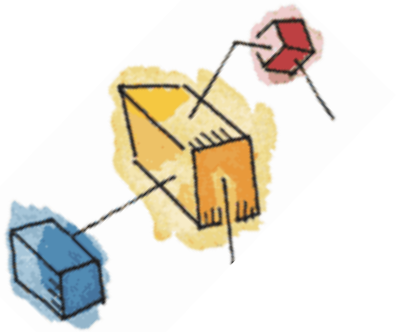
Chapter 6
Concurrency: Deadlock and Starvation

Dave Bremer
Otago Polytechnic, N.Z.



Roadmap

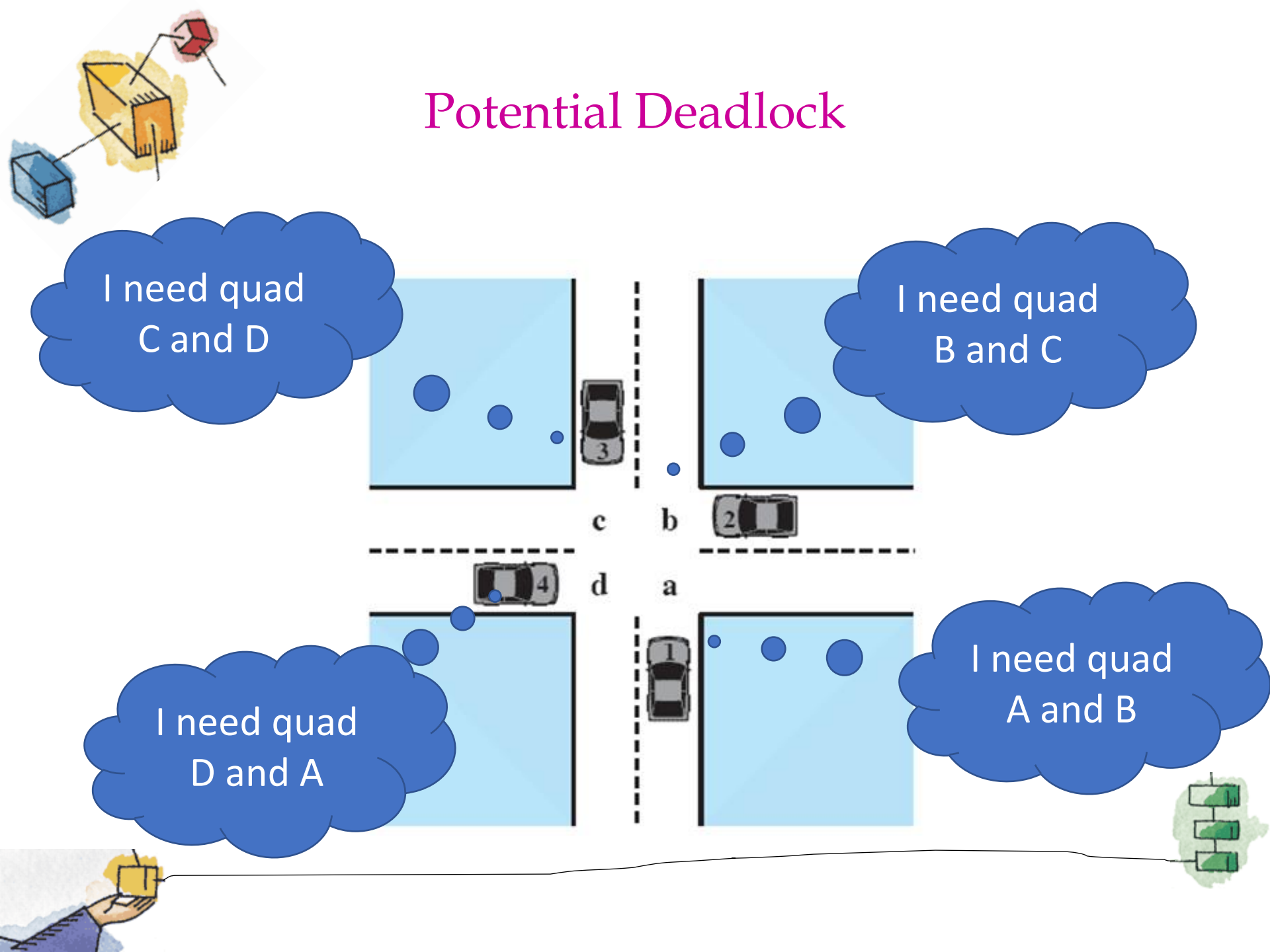
- Principals of Deadlock
  - Deadlock prevention
  - Deadlock Avoidance
  - Deadlock detection



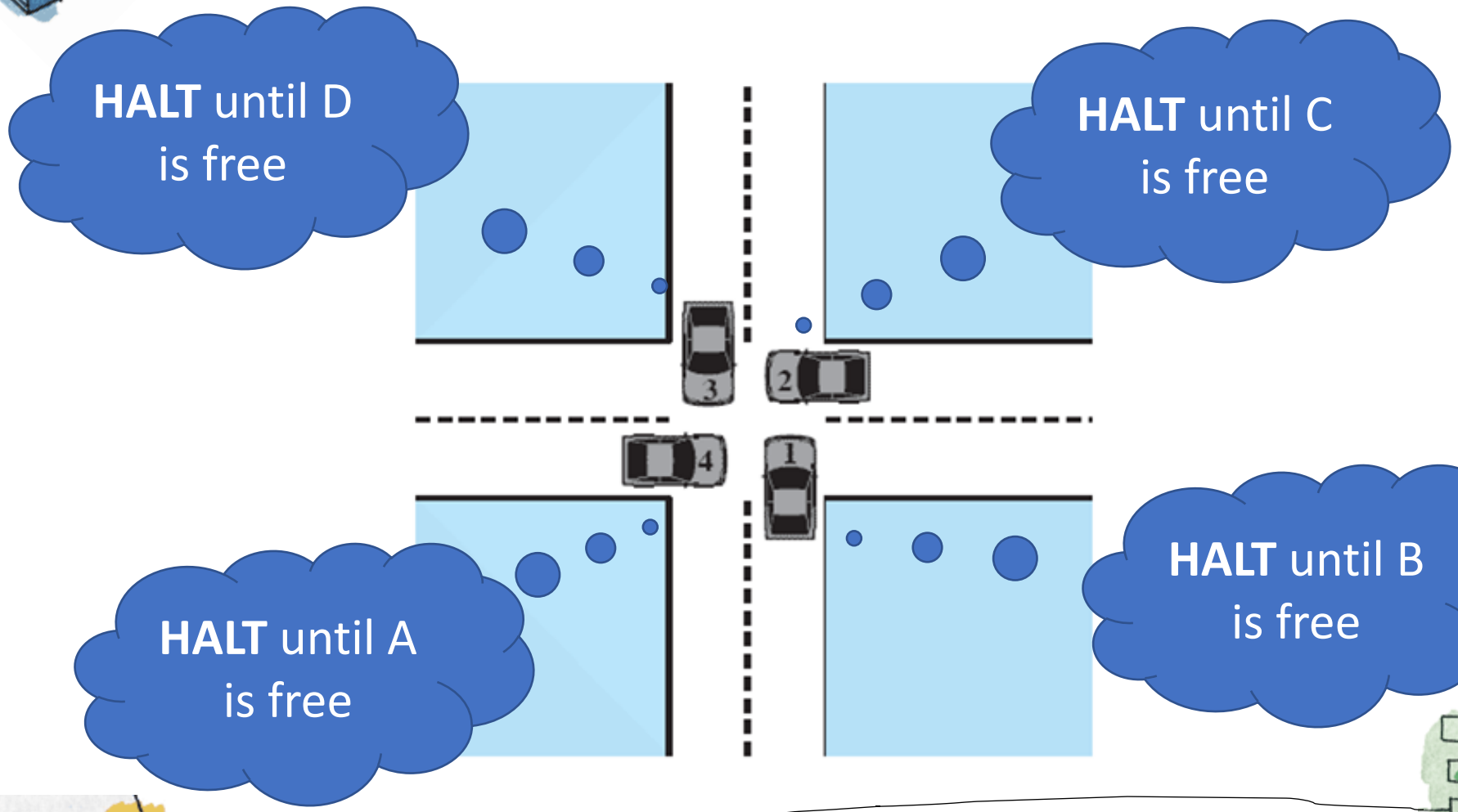
Deadlock

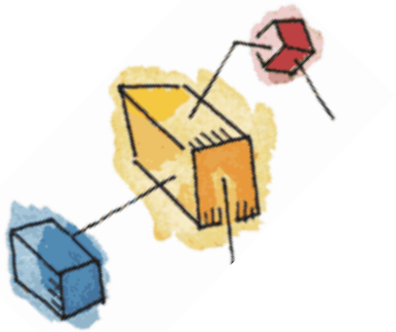
- A set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set
  - Typically involves processes competing for the same set of resources
- No efficient solution

Potential Deadlock



Actual Deadlock





Deadlock involving Two Processes P and Q

- Lets look at this with two processes P and Q
- Each needing exclusive access to a resource A and B for a period of time
- There can be six different way of execution
- 1. Q require B , then A and then release B , and A . When q resume execution it will be acquired by another process p .

Process P	Process Q
• • •	• • •
Get A	Get B
• • •	• • •
Get B	Get A
• • •	• • •
Release A	Release B
• • •	• • •
Release B	Release A
• • •	• • •

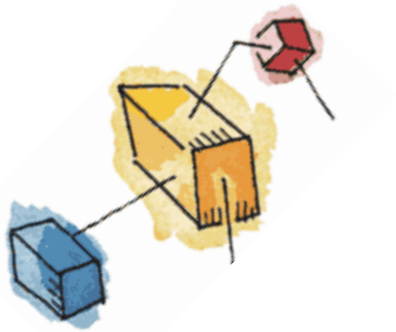
2 . Q acquires B and then A. P executes and blockes on request A . Q release B and A then P can execute and acquire resources.

3. Q acquire B then P acquire A , deadlock is inevitable, as Q will be block on A and P will block on B.

4. P acquire A, then Q acquire B then , Deadlock is inevitable,

5. P acquires A and then B. Q execute and block on request of B. P release A and B then Q can execute.

6. P acquires A and B the release A and B. so Q can acquire both the resources..

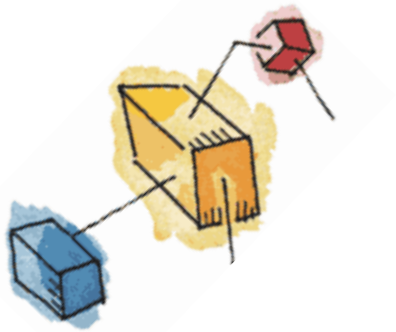


- Existence of deadlock depends on the logic of the two processes..
- Deadlock is only inevitable if the joint progress of the two processes create a path which create a deadlock.
- Deadlock occurs depends on the both the dynamics of execution and details of the application.

Alternative logic

- Suppose that P does not need both resources at the same time so that the two processes have this form.
- At this form dead lock will not occur.

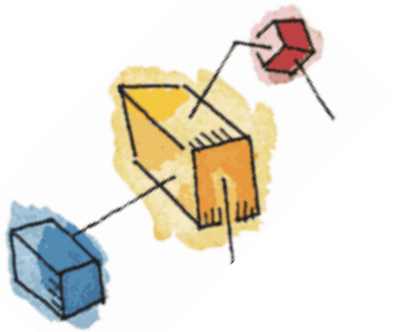
Process P	Process Q
• • •	• • •
Get A	Get B
• • •	• • •
Release A	Get A
• • •	• • •
Get B	Release B
• • •	• • •
Release B	Release A
• • •	• • •



Resource Categories

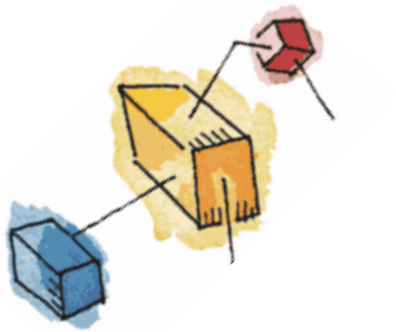
Two general categories of resources:

- Reusable
  - can be safely used by only one process at a time and ***is not depleted*** by that use.
  - Process obtain resource unit that they later release for reuse by other process.
- Consumable
  - one that can be created (***produced***) and destroyed (***consumed***).



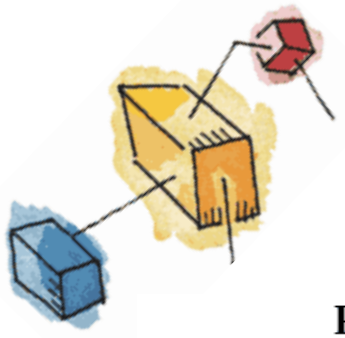
Reusable Resources

- Such as:
  - Processors, I/O channels, main and secondary memory, devices, and data structures such as files, databases, and semaphores
- Deadlock occurs if each process holds one resource and requests the other



Example of Reuse Deadlock

- Consider two processes that compete for exclusive access to a disk file D and a tape drive T.
- Deadlock occurs if each process holds one resource and requests the other.



Reusable Resources Example

Process P

Step	Action
p_0	Request (D)
p_1	Lock (D)
p_2	Request (T)
p_3	Lock (T)
p_4	Perform function
p_5	Unlock (D)
p_6	Unlock (T)

Process Q

Step	Action
q_0	Request (T)
q_1	Lock (T)
q_2	Request (D)
q_3	Lock (D)
q_4	Perform function
q_5	Unlock (T)
q_6	Unlock (D)

Figure 6.4 Example of Two Processes Competing for Reusable Resources

Deadlock involving reusable resources ,
Two process have exclusive lock on disk file D. and Tape drive T.
Deadlock occur when each process hold one resource and request for other Resource.
One way to deal with deadlock is impose system design constraint concerining The order in which resources can be requested.

Example 2: Memory Request

- Space is available for allocation of 200Kbytes, and the following sequence of events occur

P1

. . .

Request 80 Kbytes;

. . .

Request 60 Kbytes;

P2

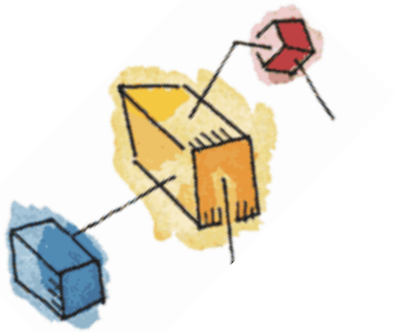
. . .

Request 70 Kbytes;

. . .

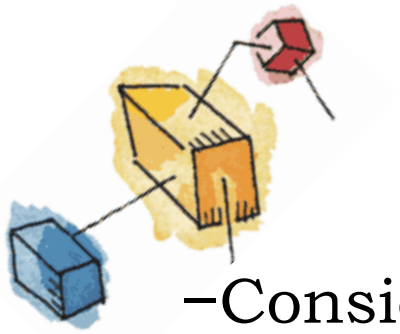
Request 80 Kbytes;

- Deadlock occurs if both processes progress to their second request.
- Amount of memory to be requested is not known ahead of time.
- It is difficult deal with such kind of deadlock.



Consumable Resources

- Such as Interrupts, signals, messages, and information in I/O buffers.
- Process may create any number of such resources.
- Deadlock may occur if a Receive message is blocking
- May take a rare combination of events to cause deadlock



Example of Deadlock

- Consider a pair of processes, in which each process attempts to receive a message from the other process and then send a message to the other process
- Deadlock occur if the Receive is blocking. poor design causes the deadlock.

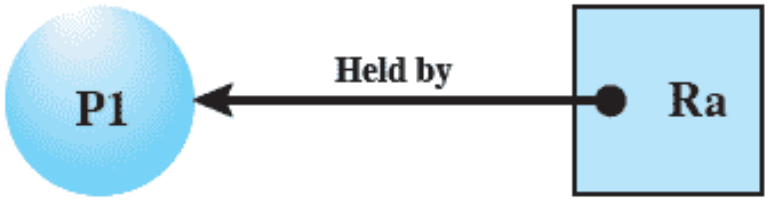
P1	P2
...	...
Receive (P2);	Receive (P1);
...	...
Send (P2, M1);	Send (P1, M2);

Resource Allocation Graphs

- Useful tool
- It is Directed graph that depicts a state of the system resources and processes
- each resource and process represented by node.
- Dot inside node is instance of resources

(a) Resouce is requested

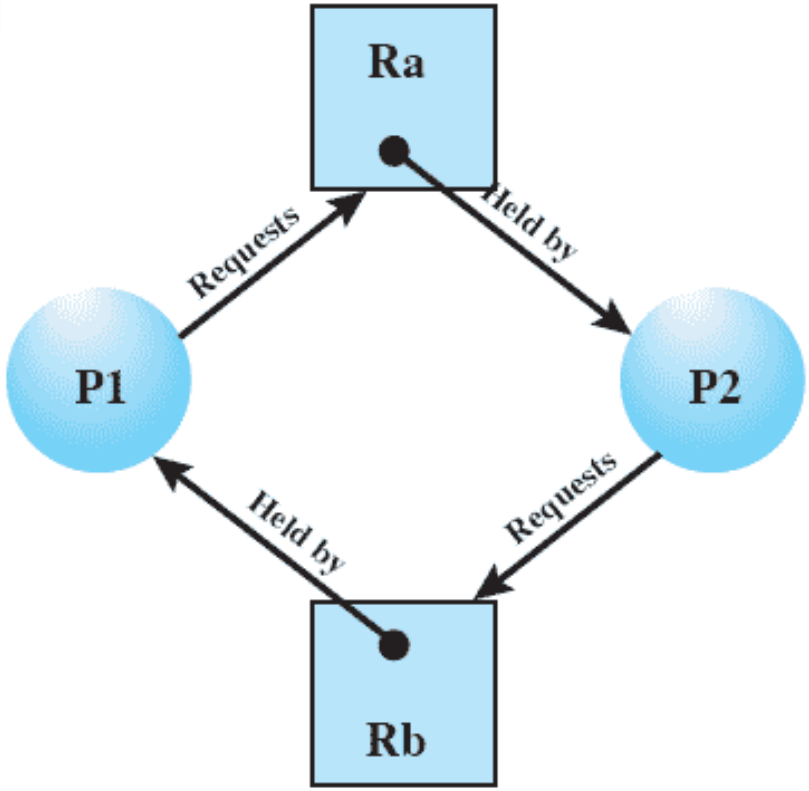
- Not granted yet



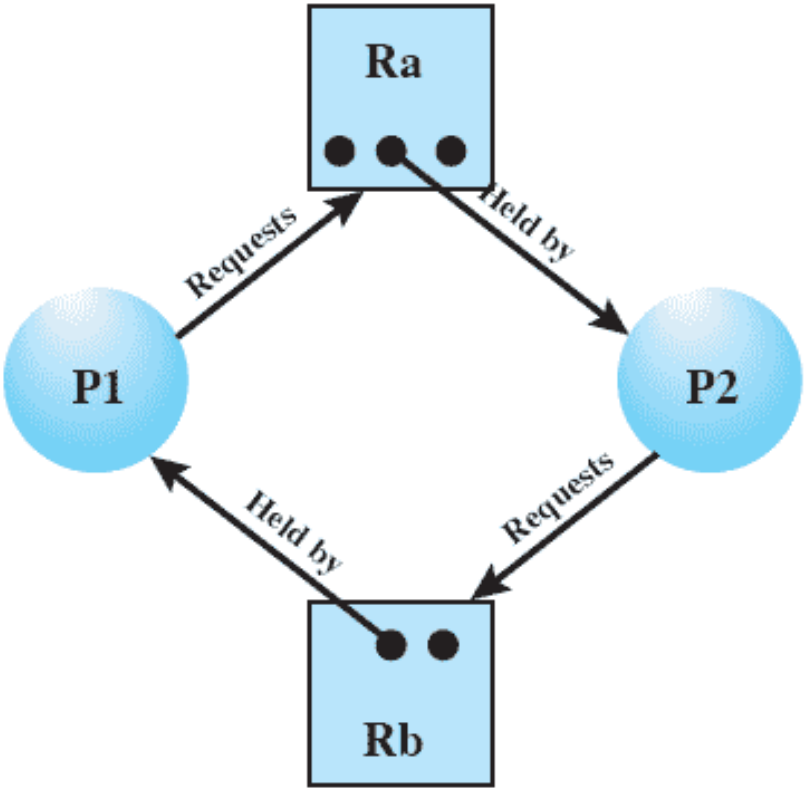
(b) Resource is held

one unit of resource assigned to process

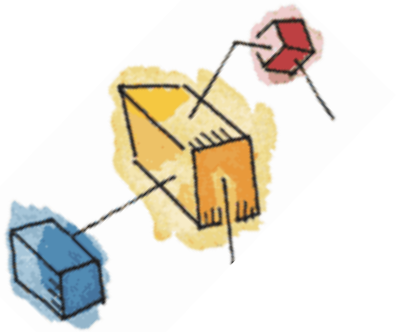
Resource Allocation Graphs of deadlock



(c) Circular wait



(d) No deadlock



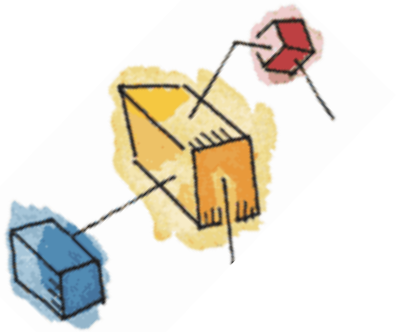
- fig c shows the deadlock. Ra assign to p2, p2 request Rb. Rb assigned to P1 . P1 requesting Ra.
- in fig d. no deadlock as multiple instance of resource Ra available.

  - If graph contains no cycles ->no deadlock
  - If graph contains a cycle -> if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Conditions for *possible* Deadlock

- Three conditions of policy must be present for a deadlock .

- Mutual exclusion
  - Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process.
- Hold-and-wait
  - A process may hold allocated resources while awaiting assignment of others process
- No pre-emption
  - No resource can be forcibly removed form a process holding it



Actual Deadlock Requires …

All previous 3 conditions plus:

4. Circular wait
   - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain.

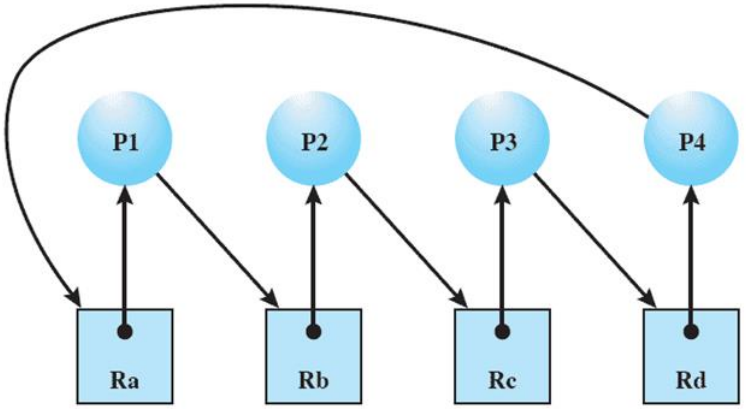


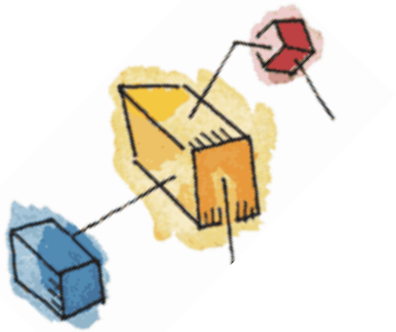
Figure 6.6 Resource Allocation Graph for Figure 6.1b

- possibility of Deadlock

1. mutual exclusion
2. No pre-emption
3. Hold and wait

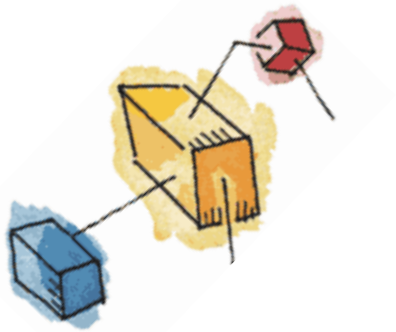
- Existence of Deadlock

1. mutual exclusion
2. No pre-emption
3. Hold and wait
4. Circular wait



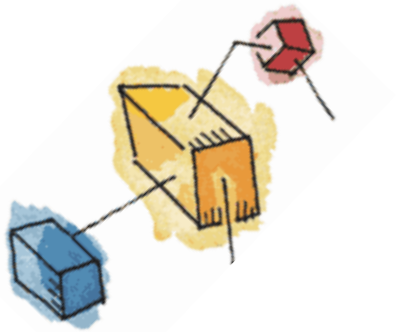
Dealing with Deadlock

- Three general approaches exist for dealing with deadlock.
  - Prevent deadlock
  - Avoid deadlock
  - Detect Deadlock



Roadmap

- Principals of Deadlock
- Deadlock prevention
  - Deadlock Avoidance
  - Deadlock detection



Deadlock Prevention Strategy

- Design a system in such a way that the possibility of deadlock is excluded.
- Two main methods
  - Indirect – prevent all three of the necessary conditions occurring at once
  - Direct – prevent circular waits

Deadlock Prevention Conditions

- Mutual Exclusion
  - Must be supported by the OS.
  - Some resource like file support multiple read at a time. But only exclusive access for writes. Deadlock can occur when two process require write permission.
- Hold and Wait
  - Require a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously.
  - This approach is insufficient .
  - 1.

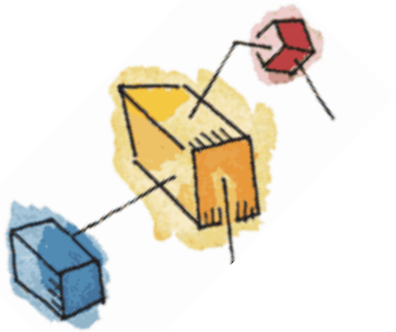
- Process may be held up for a long time waiting for all of its resources request to be filled. While it can proceed with only some of the resources.

- Second , resources allocated to a process may remain unused for considerable period, during which time they are denied to other processes.

- Another issue is process may not know in advance of resources.

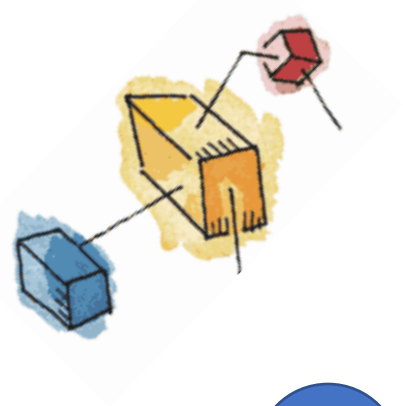
Deadlock Prevention Conditions 3 & 4

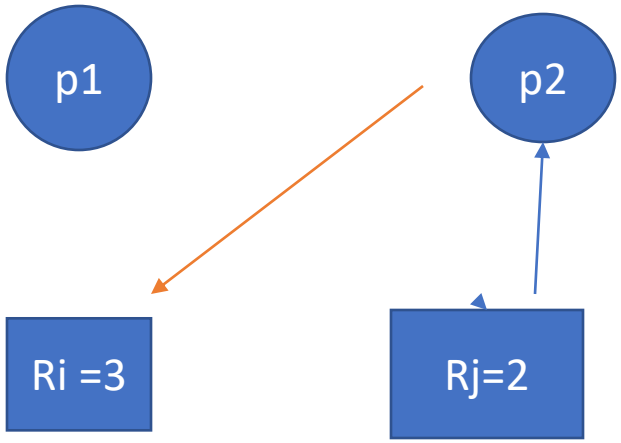
- No Preemption
  - No preemption can be prevented in
  - First ;
  - If Process must release resource and request again.
  - Second :
  - If process request resource , that is currently held by other process. OS may preempt a process to require it releases its resources.
  - It will prevent deadlock if no two process have same priority.

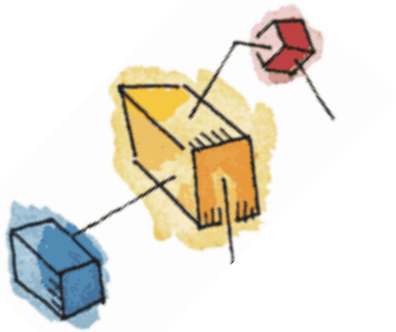


- Circular Wait
  - Circular wait condition can be prevented by defining a linear ordering of resource type.
  - If process as been allocated resources of type R , then it may request only those resources which are in lesser order than R.
  - E.g
  - Let index associated with resources. Resource Ri precede Rj in ordering I > j.
  - Two process A and B are deadlocked if A acquire Ri and request Rj . And B acquire Rj and request Ri. Which is not possible because

  i > j so Ri> Rj

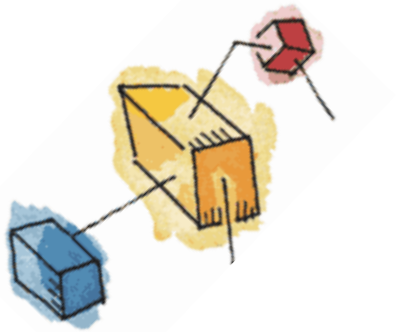



p1
p2
Ri =3
Rj=2



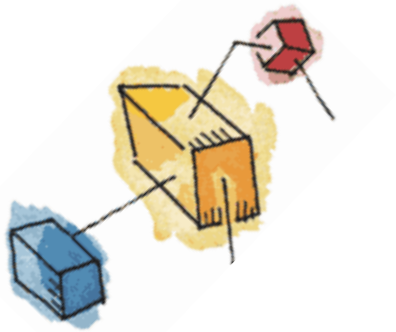
Roadmap

- Principals of Deadlock
  - Deadlock prevention
  - Deadlock Avoidance
  - Deadlock detection



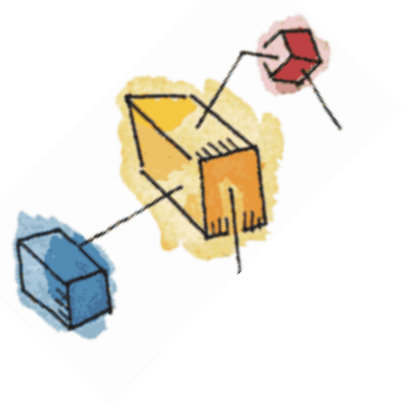
Deadlock Avoidance

- In deadlock prevention , preventing conditions to be occur(mutual exclusion, hold-wait,circular wait,no preemption)
- Dead lock avoidance allowed three necessary conditions but assure that deadlock point never occur.
- A decision is made dynamically whether the current resource allocation & request will, if granted, potentially lead to a deadlock
- Requires knowledge of future process requests



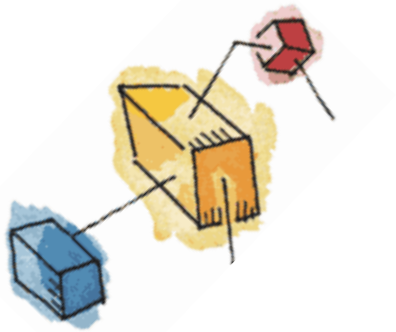
Two Approaches to Deadlock Avoidance

- Process Initiation Denial
  - Do not start a process if its demands might lead to deadlock
- Resource Allocation Denial
  - Do not grant an incremental resource request to a process if this allocation might lead to deadlock



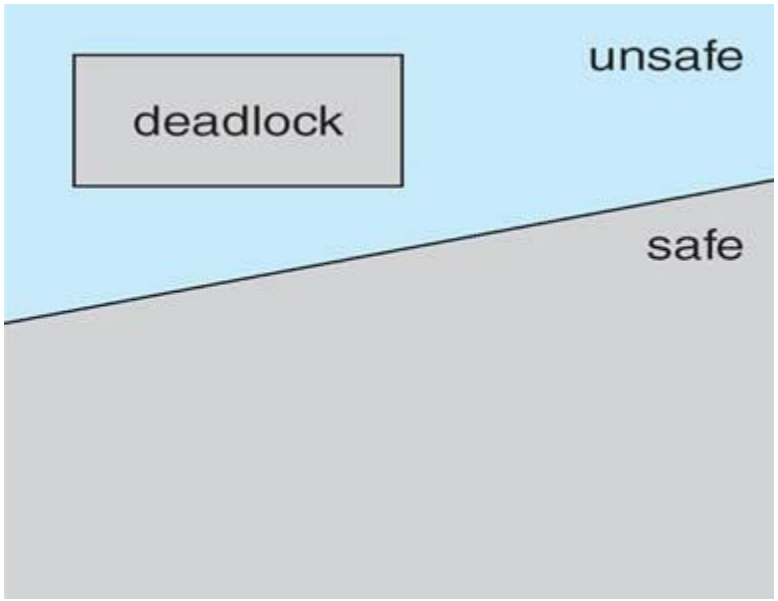
Process Initiation Denial

- A process is only started if the maximum claim of all current processes plus those of the new process can be met.
- Not optimal,
  - Assumes the worst: that all processes will make their maximum claims together.
- Information of resources required declared in advance by a process for deadlock avoidance to work
- It refuse to start a process if its resources requirements might lead to deadlock.



Resource Allocation Denial

- Referred to as the banker's algorithm
  - A strategy of resource allocation denial
- Consider a system with fixed number of resources
  - ***State*** of the system is the current allocation of resources to process.
  - state consists of two vectors, Resource an Available. And two matrices claim , & allocation.
  - ***Safe state*** is where there is at least one sequence that does not result in deadlock.
  - ***Unsafe state*** is a state that is not safe



If a system is in safe state -> no deadlocks

If a system is in unsafe state -> possibility of deadlock

Metrices and Vectors

- Resource R
  - R = [R1,R2,R3…Rn]
  - Total amount of each resource in the system.
- Available = V = [v1, v2,v3..Vn]
  - Total amount of each resource not allocated to any process (no of free resources)
- Claim = C /

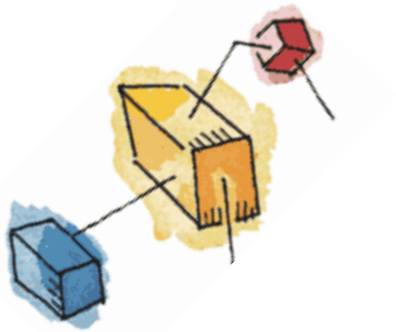
$$\begin{bmatrix} c11 & c12 & c13\ldots & c1m \\ c21 & c22 & c23\ldots & c2m \\ c31 & c32 & c33\ldots & c3m \\ & & & \\ cn1 & cn2 & cn3\ldots & cnm \end{bmatrix}$$

cij = requirement of process i for j resources

- Allocation A

$$\begin{bmatrix} A11 & A12 & A13 & \ldots A1m \\ A21 & A22 & A23 & \ldots A2m \\ A31 & A32 & A33 & \ldots A3m \\ An1 & An2 & An3 & \ldots Anm \end{bmatrix}$$

Aij = current allocation to process i of resource j



Determination of Safe State

- A system consisting of four processes and three resources.
- Allocations are made to processors
- ***Is this a safe state?***

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

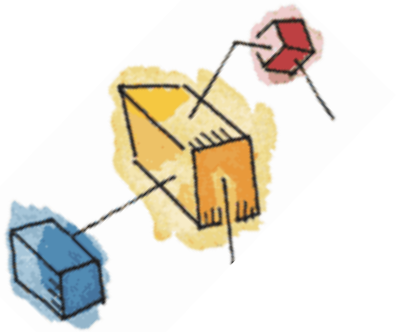
R1	R2	R3
9	3	6

Resource vector **R**

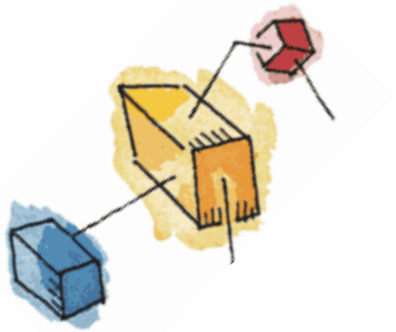
R1	R2	R3
0	1	1

Available vector **V**

(a) Initial state



- Allocation matrix : currently resources allocated to process.
- calim matrix / max matrix : total resource required by process
- C-A : (remaining)required resources for process execution (request matrix)
- Resource vector :: total unit available into system for each resources
  - R1 has 9 resource , R2 has 6 resource , R3 has 3 total resources into system.
- Available vector : resources available(free) at current time { Resource vector – allocated vector]



Process *i*

- $C_{ij} - A_{ij} \leq V_j$, for all j
- This is not possible for P1,
  - which has only 1 unit of R1 and requires 2 more units of R1, 2 units of R2, and 2 units of R3.
- If we assign one unit of R3 to process P2,
  - Then P2 has its maximum required resources allocated and can run to completion and return resources to 'available' pool

After P2 runs to completion

- Can any of the remaining processes can be completed?

	R1	R2	R3
P1	3	2	2
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

Note P2 is completed

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
6	2	3

Available vector **V**

(b) P2 runs to completion

After P1 completes

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	1	0	3
P4	4	2	0

C – A

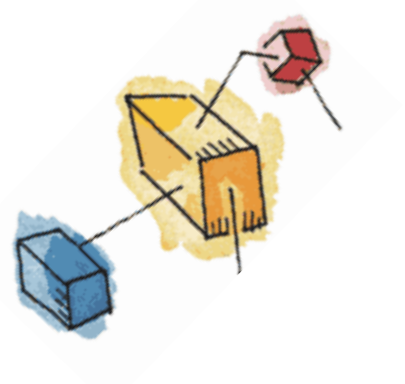
R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
7	2	3

Available vector **V**

(c) P1 runs to completion



P3 Completes

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

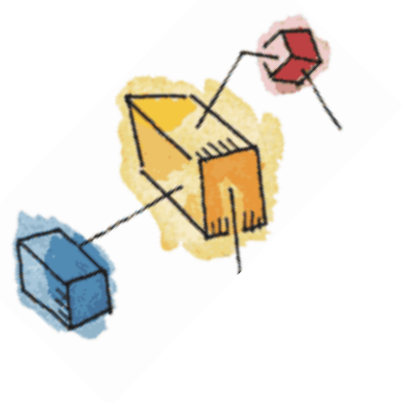
Resource vector **R**

R1	R2	R3
9	3	4

Available vector **V**

(d) P3 runs to completion

Thus, the state defined originally is a safe state.



	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Allocation matrix **A**

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
9	3	6

Available vector **V**

(d) P3 runs to completion

P2,p1,p3,p4-> safe sequence
P2,p3,p4,p1

- If with same example p1 request more one resources of R1 and R3 then
- Claim for P1 [4 2 3]
- If that request of one extra is granted then allocation for p1 [2 0 1] so now matrix become

• Claim c = Allocation matrix c-A

0

•

p1	4	2	3
p2	6	1	3
P3	3	1	4
p4	4	2	2

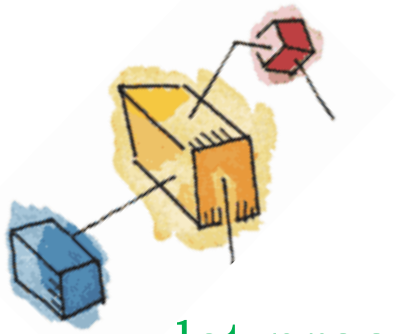
P1	2	0	1
P2	5	1	1
p3	2	1	1
P4	0	0	2
total	9	2	5

P1	2	2	2
P2	1	0	2
P3	1	0	3
P4	4	2	0

Now

available is [9-9,3-2,6-5] = [0,1,1]

all process required R1 resource and no R1 resource is available so this state is unsafe state.



exercise

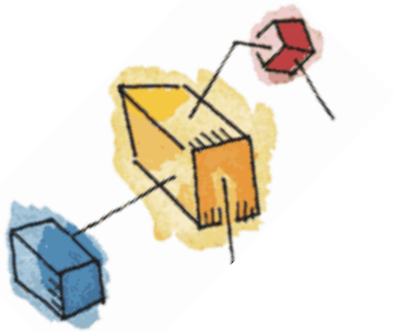
- let process p1, p2 ,p3 and x,y,z are resources.
- Is system is in safe state or not?
- C / Max matrix allocation matrix resource vector

	X	Y	Z
P1	7	5	3
P2	3	2	2
P3	9	0	2
p4	2	2	2
p5	4	3	3

	X	Y	Z
P1	0	1	0
P2	2	0	0
P3	3	0	2
p4	2	1	1
P5	0	0	2

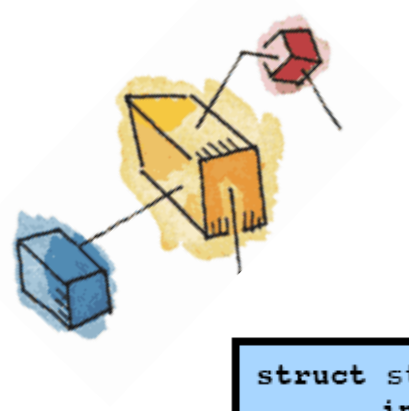
X	Y	Z
10	5	7

- If p2 grants two resource more of R2 and one resource of R3 will system in a safe state?



Deadlock Avoidance

- When a process makes a request for a set of resources,
  - assume that the request is granted,
  - Update the system state accordingly,
- Then determine if the result is a safe state.
  - If so, grant the request and,
  - if not, block the process until it is safe to grant the request.



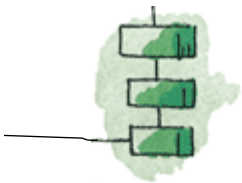
Deadlock Avoidance Logic

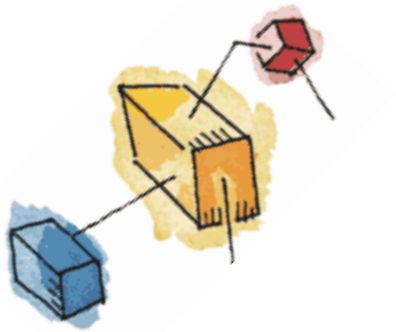
```
struct state {
        int resource[m];
        int available[m];
        int claim[n][m];
        int alloc[n][m];
}
```

(a) global data structures

```
if (alloc [i,*] + request [*] > claim [i,*])
     < error >;                                  /* total request > claim*/
else if (request [*] > available [*])
     < suspend process >;
else {                                                /* simulate alloc */
     < define newstate by:
     alloc [i,*] = alloc [i,*] + request [*];
     available [*] = available [*] - request [*] >;
}
if (safe (newstate))
     < carry out allocation >;
else {
     < restore original state >;
     < suspend process >;
}
```

(b) resource alloc algorithm





Determination of Safe State

- A system consisting of four processes and three resources.
- Allocations are made to processors
- ***Is this a safe state?***

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Claim matrix **C**

	R1	R2	R3
P1	1	0	0
P2	6	1	2
P3	2	1	1
P4	0	0	2

Allocation matrix **A**

	R1	R2	R3
P1	2	2	2
P2	0	0	1
P3	1	0	3
P4	4	2	0

C – A

R1	R2	R3
9	3	6

Resource vector **R**

R1	R2	R3
0	1	1

Available vector **V**

(a) Initial state

Deadlock Avoidance Logic

```
boolean safe (state S) {
    int currentavail[m];
    process rest[<number of processes>];
    currentavail = available;
    rest = {all processes};
    possible = true;
    while (possible) {
        <find a process Pk in rest such that
            claim [k,*] - alloc [k,*] <= currentavail;>
        if (found) {                          /* simulate execution of Pk */
            currentavail = currentavail + alloc [k,*];
            rest = rest - {Pk};
        }
        else possible = false;
    }
    return (rest == null);
}
```

(c) test for safety algorithm (banker's algorithm)

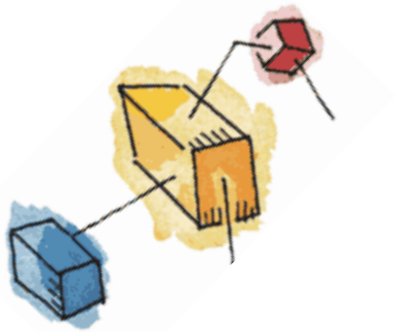
Figure 6.9 Deadlock Avoidance Logic

Deadlock Avoidance Advantages

- It is not necessary to preempt and rollback processes, as in deadlock detection,
- It is less restrictive than deadlock prevention.

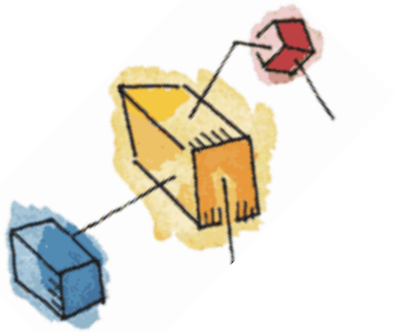
Deadlock Avoidance Restrictions

- Maximum resource requirement must be stated in advance
- Processes under consideration must be independent and with no synchronization requirements
- There must be a fixed number of resources to allocate
- No process may exit while holding resources



Roadmap

- Principals of Deadlock
  - –Deadlock prevention
  - –Deadlock Avoidance
  - –Deadlock detection

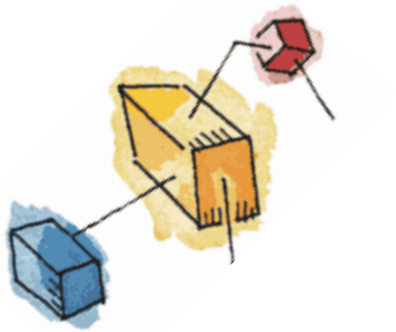


Deadlock Detection

- Deadlock prevention strategies are very conservative;
  - limit access to resources and impose restrictions on processes.
- Deadlock detection strategies do the opposite
  - Resource requests are granted whenever possible.
  - Regularly check for deadlock

A Common Detection Algorithm

- Use a Allocation matrix and Available vector as previous (deadlock detection)
- Also use a request matrix ***Q (claim-allocation)***
  - Where ***Qij*** indicates that an amount of resource ***j*** is requested by process ***I***
- *Algorithm proceed by marking processes that are not deadlocked.*
- First 'un-mark' all processes that are not deadlocked
  - Initially that is all processes



Detection Algorithm

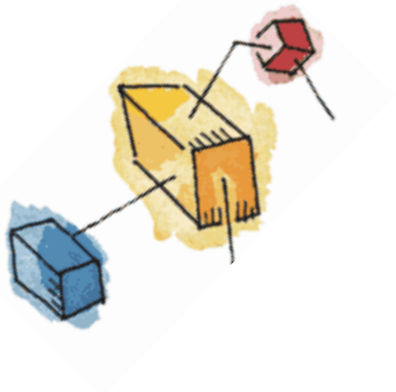
1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector **W** to equal the Available vector.
3. Find an index ***i*** such that process ***i*** is currently unmarked and the ***i***th row of Q is less than or equal to ***W***.
   - i.e. $Q_{ik} \leq W_k$ for $1 \leq k \leq m$.
   - If no such row is found, terminate

Detection Algorithm cont.

4. If such a row is found,
   - mark process *i* and add the corresponding row of the allocation matrix to W.
   - i.e. set $W_k = W_k + A_{ik}$, for $1 \le k \le m$

Return to step 3.

- A deadlock exists if and only if there are unmarked processes at the end
- Each unmarked process is deadlocked.



Deadlock Detection

	R1	R2	R3	R4	R5
P1	0	1	0	0	1
P2	0	0	1	0	1
P3	0	0	0	0	1
P4	1	0	1	0	1

Request matrix Q

	R1	R2	R3	R4	R5
P1	1	0	1	1	0
P2	1	1	0	0	0
P3	0	0	0	1	0
P4	0	0	0	0	0

Allocation matrix A

R1	R2	R3	R4	R5
2	1	1	2	1

Resource vector

R1	R2	R3	R4	R5
0	0	0	0	1

Available on vector

Figure 6.10 Example for Deadlock Detection

Detection Algorithm

- 1. Mark P4, because P4 has no allocated resources.
- 2. Set **W** = (0 0 0 0 1).
- 3. The request of process P3 is less than or equal to **W**, so mark P3 and set W=W + (0 0 0 1 0) = (0 0 0 1 1).
- 4. No other unmarked process has a row in Q that is less than or equal to W.
  - Therefore, terminate the algorithm.
- The algorithm concludes with P1 and P2 unmarked, indicating that these processes are deadlocked.

Recovery Strategies Once Deadlock Detected

- Abort all deadlocked processes
  - Back up each deadlocked process to some previously defined checkpoint, and restart all process
    - Risk of deadlock recurring
- Successively abort deadlocked processes until deadlock no longer exists
- Successively preempt resources until deadlock no longer exists

Advantages

Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates online handling	•Inherent preemption losses