50.005 CSE

Natalie Agus
Information Systems Technology and Design
SUTD

NECESSARY CONDITIONS

But NOT sufficient

1 **Mutex**

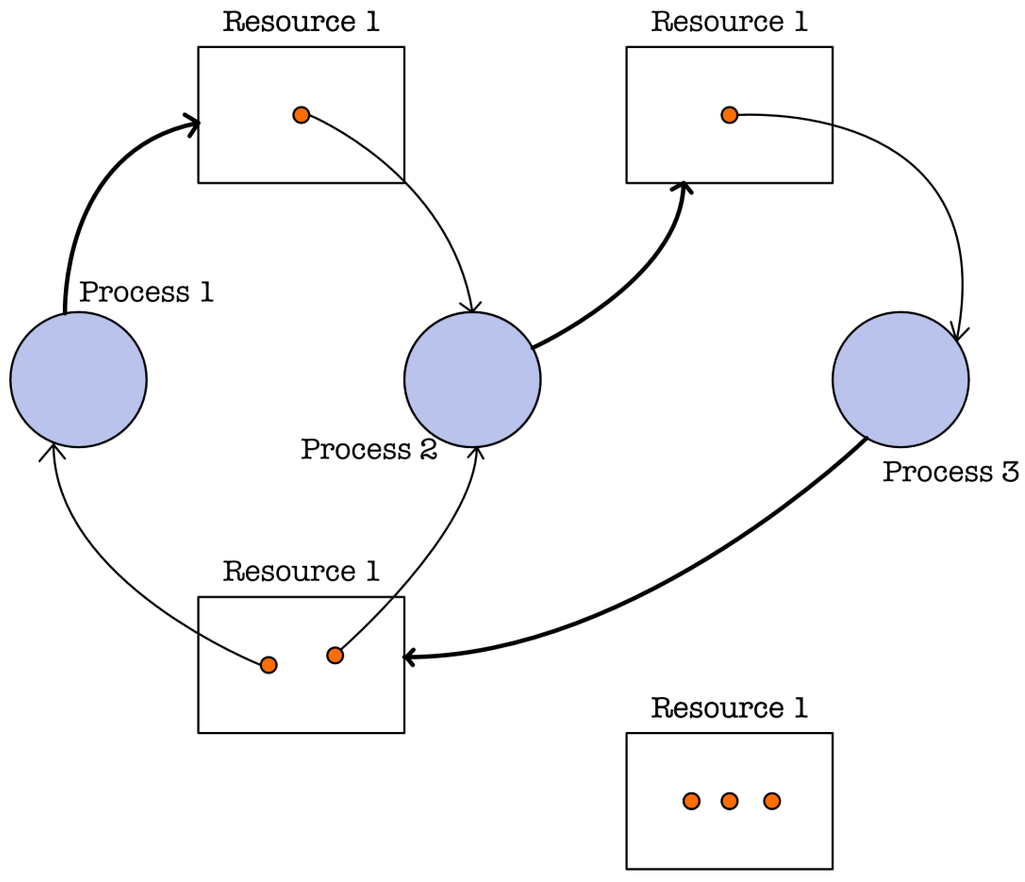
2 **Hold resource & wait**

3 **No pre-emption**

4 **Circular wait**

If all 4 happens, deadlock **might** happen
Deadlock can be prevented by removing either conditions

RESOURCE ALLOCATION GRAPH



Circular wait is a necessary but **not sufficient** condition **if there's multiple instances** per resources

HANDLING DEADLOCKS

Real world OS do not handle deadlocks completely all the time

Avoidance

Detection

Prevention

DEADLOCK AVOIDANCE

Process i needs to declare max Resources it **ever needs** in the beginning

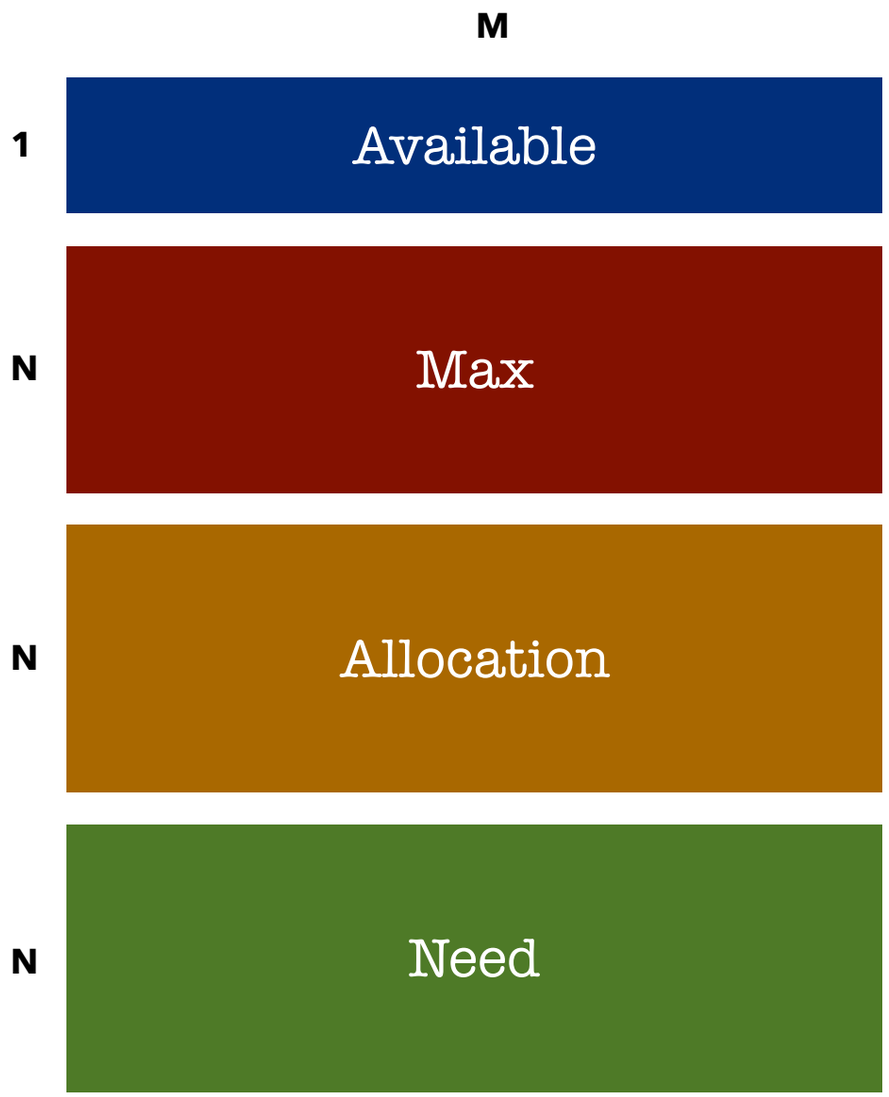
Requests that lead to unsafe state will have to wait

DON’T GIVE RESOURCES THAT MIGHT LEAD TO FUTURE DEADLOCK, EVEN IF ITS AVAILABLE NOW

Always checks if requests will lead to circular wait. Only grant requests with **safe state**

BANKER'S ALGORITHM

Future deadlock detection algorithm



- Given 4 arrays: Available, Max, Allocation, and Need
- N = number of processes / threads in the system
- M = number of resources type

1. SAFETY ALGORITHM

The first part of banker's algorithm, $O(mn^2)$

1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize *Work* = *Available* and *Finish*[i] = *false* for $i = 0, 1, ..., n - 1$.
2. Find an index i such that both
   a. *Finish*[i] == *false*
   b. $Need_i \leq Work$

   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.

2. RESOURCE REQ ALGORITHM

The second part of banker's algorithm

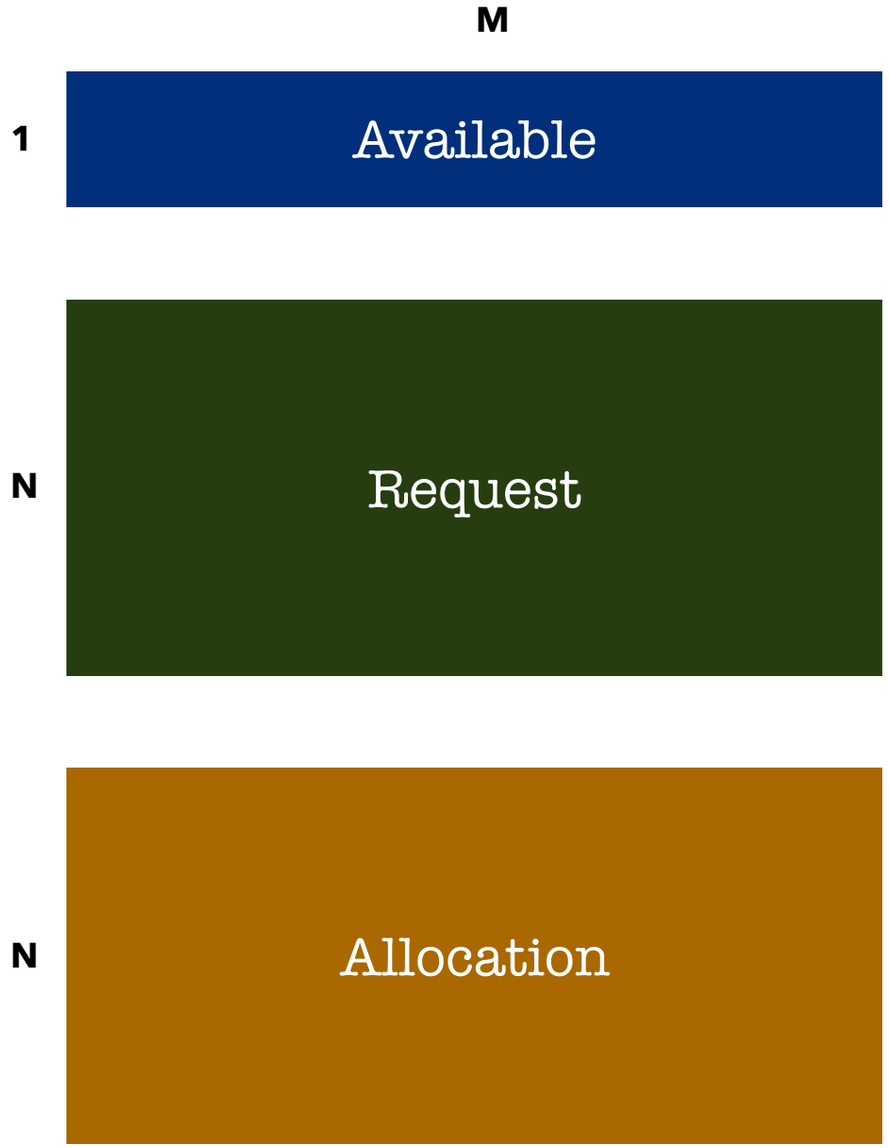
1. If $Request_i \leq Need_i$, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process 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 the resulting resource-allocation state is safe, the transaction is completed, and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$, and the old resource-allocation state is restored.

DEADLOCK DETECTION

System allows deadlock to happen and then detects with this algorithm



- Given 3 arrays: Available, Request, and Allocation
- N = number of processes / threads in the system
- M = number of resources type

DEADLOCK DETECTION ALGORITHM

Complexity of $O(mn^2)$

1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize $Work = Available$. For $i = 0, 1, ..., n\text{-}1$, 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.
3. $Work = Work + Allocation_i$
   $Finish[i] = true$
   Go to step 2.
4. If $Finish[i] == false$ for some i, $0 \leq i < n$, then the system is in a deadlocked state. Moreover, if $Finish[i] == false$, then process P_i is deadlocked.

DEADLOCK DETECTION

What happens after deadlock is detected? How to recover from deadlock?

1. **Abort** all deadlock processes, this allow them to **pre-empt** resources (get them back)
2. Can also abort all deadlock processes **one by one** until there's no more deadlock: by priority, time exec, resources used
3. Restart all aborted processes

DEADLOCK PREVENTION

Avoids potential deadlock situations by design, which is to disallow either <u>one of the four necessary conditions</u> for deadlock to happen

No resource hold and wait:

- Must get all resources before process execution
- Only allow request for resources if the process has none

Disallow circular wait:

- Must obtain resources in certain order

Allows pre-emption:

- Process must release all resources its already holding if it needs another resources that require wait
- Restart process, must wait for every resources again