Deadlocks, Part II

CSD2180 Operating Systems
BSc in Computer Science (IMGD / RTIS)
Singapore Institute of Technology / DigiPen Institute of Technology
September 2021

Attendance Taking

https://forms.office.com/r/UfBi6v7u8E

- Log in to your SIT account to submit the form
- You can only submit the form once
- The codeword is **tablet**

Deadlock Avoidance

Deadlock Avoidance

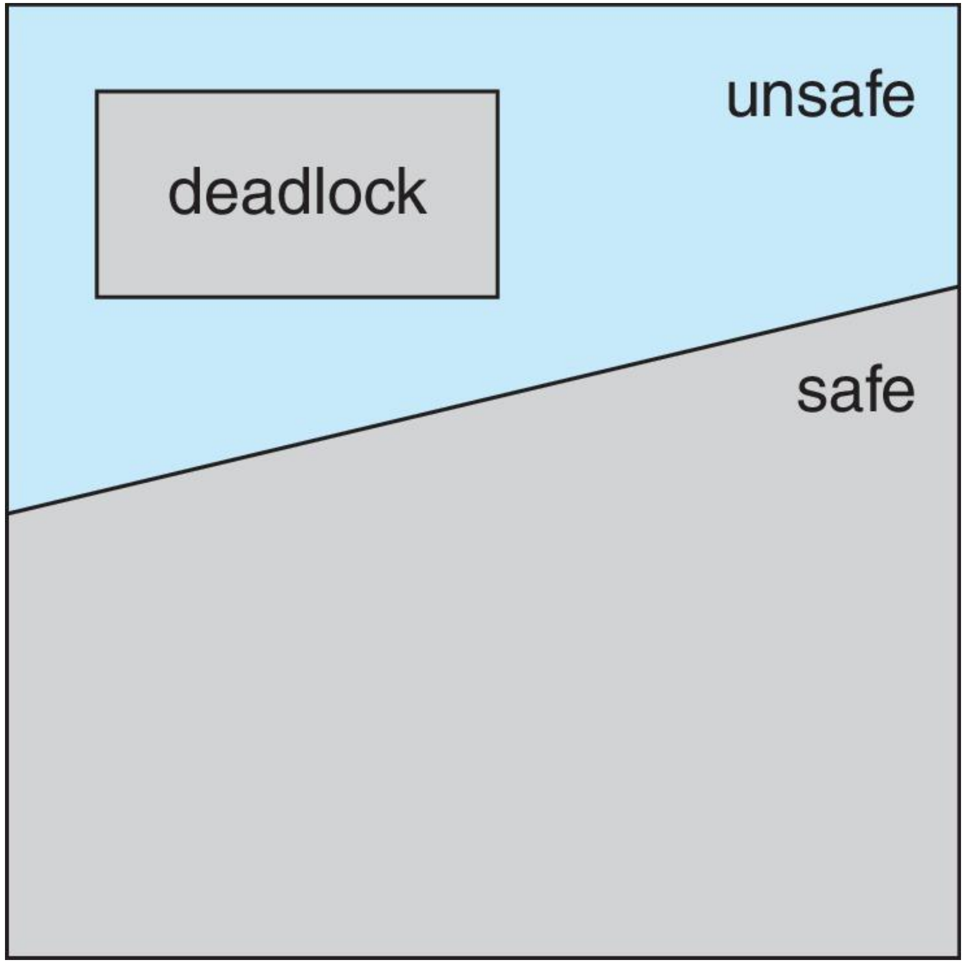
Requires that the system has some additional *a priori* information available

- Simplest and most useful model requires that each process declare the **maximum number** of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*
- System is in safe state if there exists a sequence $\langle P_1, P_2, \ldots, P_n \rangle$ of all processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all P_j, where $j < i$
- That is
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Safe State and Deadlocks



If a system is in safe state, it has no deadlocks

If a system is in unsafe state, there is possibility of deadlock

Deadlock avoidance algorithms ensure that a system will never enter an unsafe state

Safe State and Deadlocks

Consider a system with twelve units of resources and three threads

Is this system in a safe state?

Thread	Maximum Resource Needs	Current Resource Needs
T_A	10	5
T_B	4	2
T_C	9	2

Safe State and Deadlocks

Consider a system with twelve units of resources and three threads

Is this system in a safe state?

Thread	Maximum Resource Needs	Current Resource Needs
T_A	10	5
T_B	4	2
T_C	9	2

Situation analysis

- Nine units of resources currently used; 3 units free
- T_B can be allocated all its remaining resources (2 units) and then returns them all when done
  - 5 units will be available when T_B is done
- Then, T_A can be allocated all its remaining resources (5 units) and return them all when done
  - 10 units will be available when T_A is done
- Finally, T_C can be allocated all its remaining resources (7 units) and return them all when done
  - 12 units will be available when done

The sequence $\langle T_B, T_A, T_C \rangle$ satisfies the safety condition

Safe State and Deadlocks

Consider another system with twelve units of resources and three threads

Is this system in a safe state?

Thread	Maximum Resource Needs	Current Resource Needs
T_A	10	5
T_B	4	2
T_C	9	3

Safe State and Deadlocks

Consider another system with twelve units of resources and three threads

Is this system in a safe state?

Thread	Maximum Resource Needs	Current Resource Needs
T_A	10	5
T_B	4	2
T_C	9	3

Situation analysis

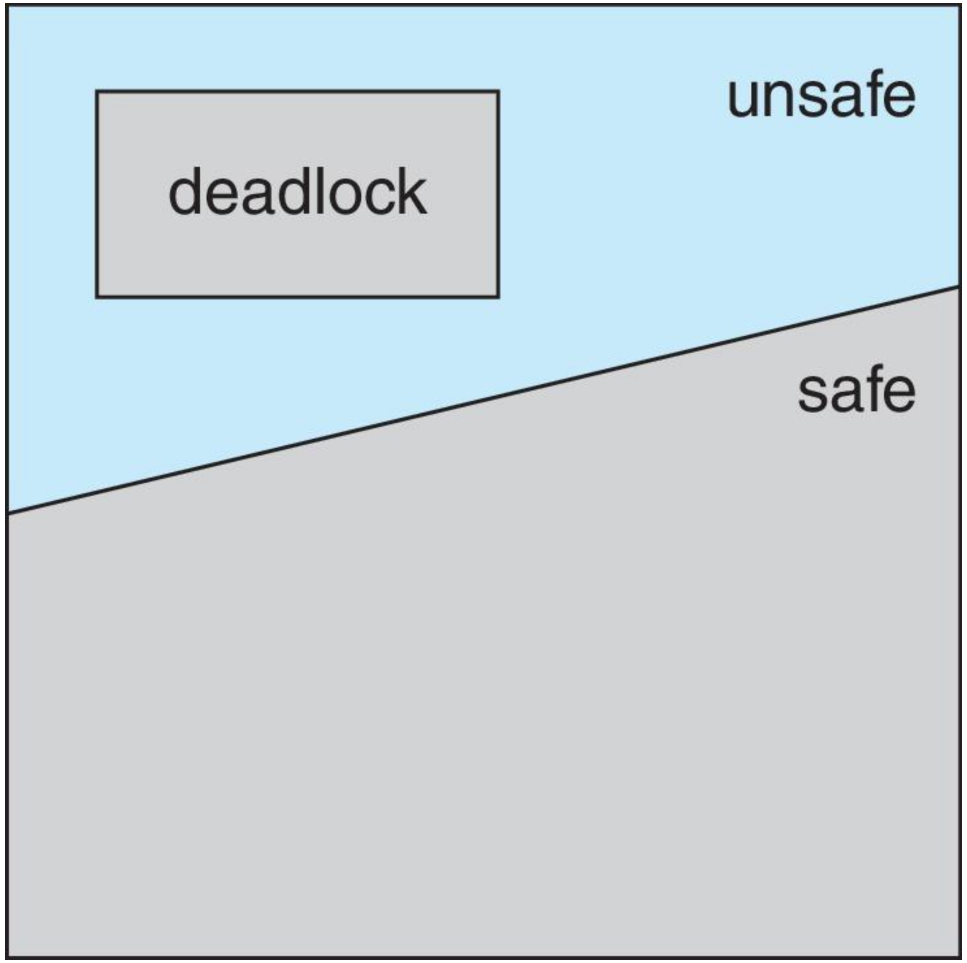
- Ten units of resources currently used; 2 units free
- T_B can be allocated all its remaining resources (2 units) and then returns them all when done
  - 4 units will be available when T_B is done
- No other threads can obtain all its resources
  - T_A requires 5 units more, while T_C requires 6 units more

The system is not in a safe state!

Deadlock Avoidance Algorithms

- If only single instances of each resource type, use a *resource-allocation graph*

- If multiple instances of a resource type exist, use the *banker's algorithm*



Resource-Allocation Graph Approach

A **claim edge** $P_i \rightarrow R_j$ indicates that process P_j may request resource R_j

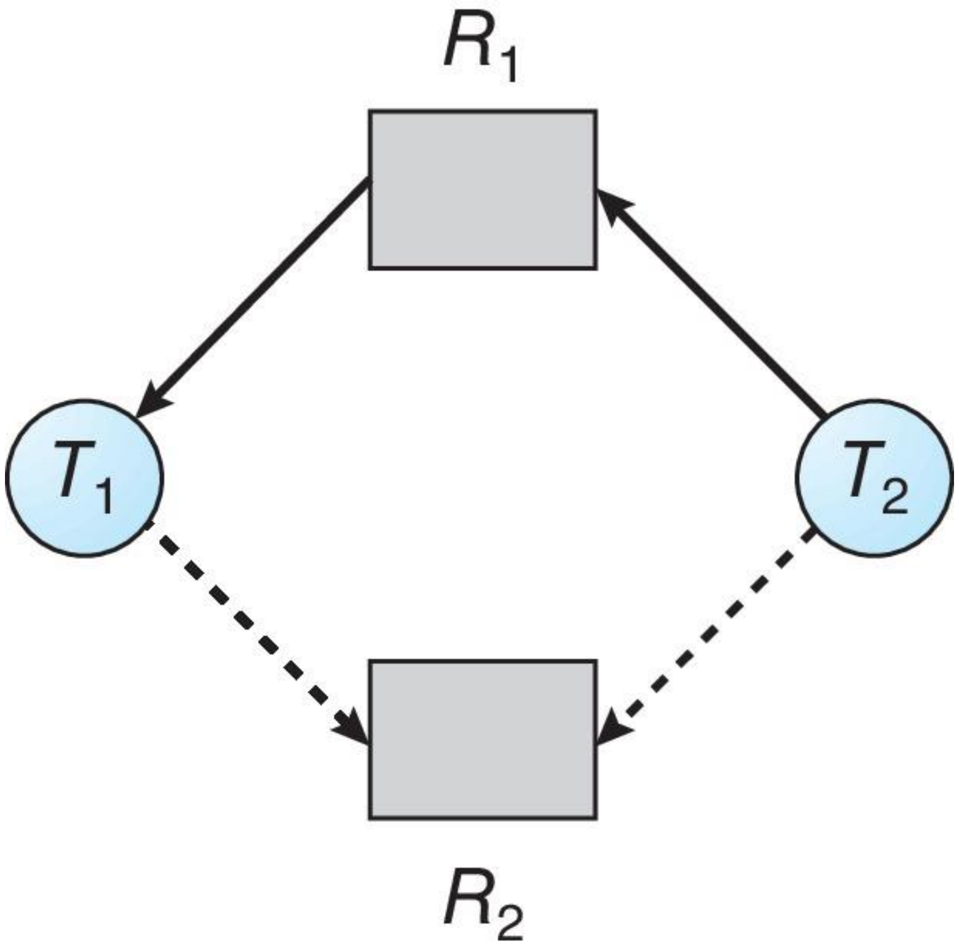
- This is represented by a dashed line

Edge characteristics

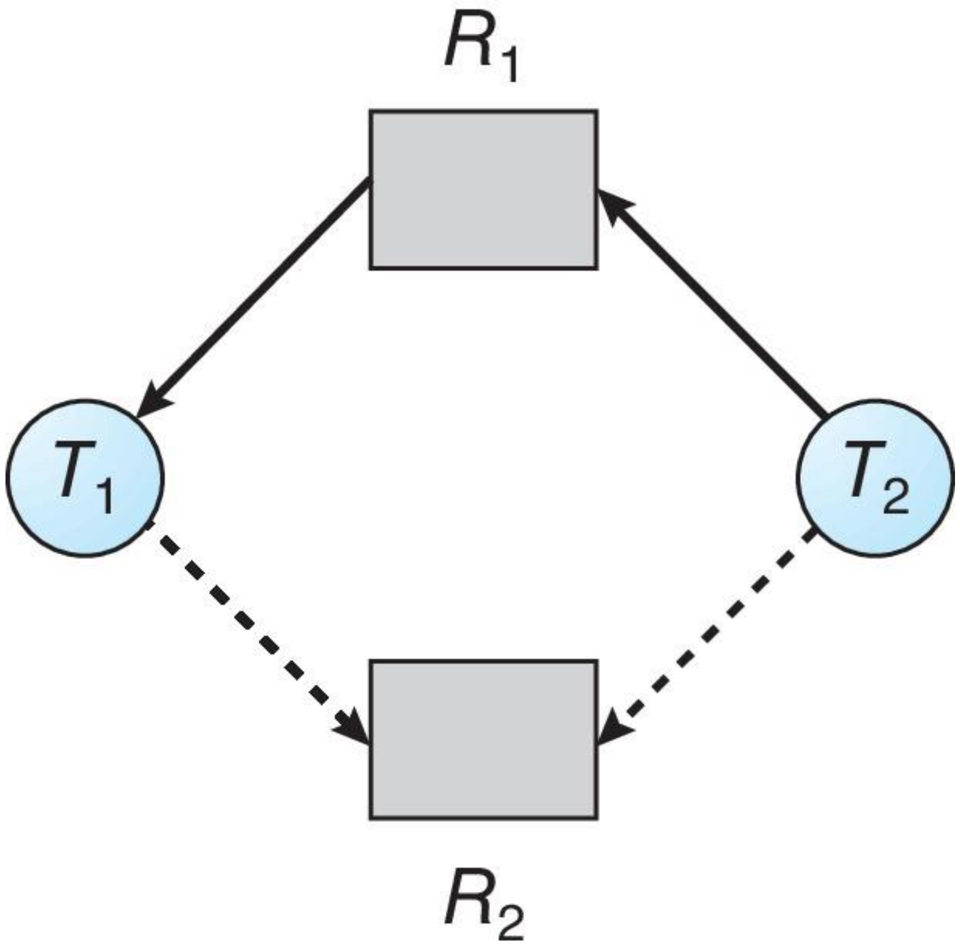
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge

Resources must be claimed *a priori* in the system

- Before process starts executing, all its claim edges must already appear in the resource-allocation graph



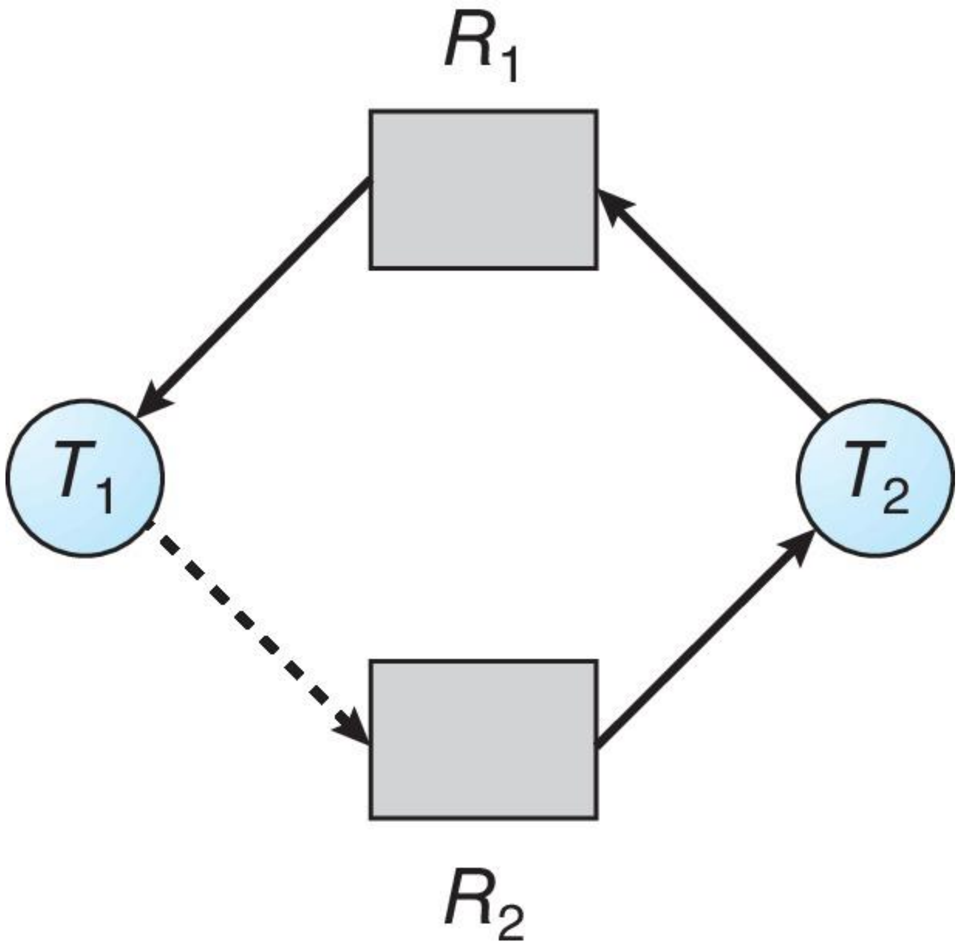
Resource-Allocation Graph Approach



Request can be granted only if converting the request edge to an assignment edge does not result in formation of a cycle

Should T_2 be assigned R_2 if it requests for it?

Resource-Allocation Graph Approach



Request can be granted only if converting the request edge to an assignment edge does not result in formation of a cycle

Should T_2 be assigned R_2 if it requests for it?

Not a good idea since a cycle will form, indicating that a deadlock will occur...

Banker's Algorithm

Multiple instances of resources

Each process must *a priori* claim maximum use

When a process requests a resource, it may have to wait

When a process gets all its resources it must return them in a finite amount of time

Think of it from the perspective of a bank lending money...

Banker's Algorithm

Data structures for the banker's algorithm

- Let
  n = number of processes
  m = number of resources types
- **Available:** Vector of length m indicating the number of available resources of each type
  - If $\text{Available}[j] = k$, there are k instances of resource type R_j available
- **Max:** $n \times m$ matrix (n rows, m columns) defining the maximum demand of each process
  - If $\text{Max}[i, j] = k$, then process P_i will request at most k instances of resource type R_j
- **Allocation:** $n \times m$ matrix defining the number of resources of each type currently allocated to each process
  - If $\text{Allocation}[i, j] = k$ then P_i is currently allocated k instances of R_j
- **Need:** $n \times m$ matrix indicating the remaining resource need of each process
  - If $\text{Need}[i, j] = k$, then P_i may need k more instances of R_j
- It can be seen that $\text{Need}[i, j] = \text{Max}[i, j] - \text{Allocation}[i, j]$

Banker's Algorithm

Notation for banker's algorithm

- Let X and Y be vectors of length n
  - We say that $X \leq Y$ if and only if $\forall i = 1, 2, \ldots, n,\ X[i] \leq Y[i]$
  - E.g., If $S = (1,7,3,2)$ and $T = (0,3,2,1)$, then $T \leq S$
- Let each row in the matrices **Allocation** and **Need** be vectors, and refer to them as $\mathrm{Allocation}_i$ and Need_i

Safety Algorithm (Banker's Algorithm)

Let **Work** and **Finish** be vectors of length m and n, respectively

1. Initialize

$$\text{Work} = \text{Available}$$
$$\text{Finish}[i] = \text{false}, \qquad \forall i = 1, 2, \dots, n$$

2. Find an i such that

$$(\text{Finish}[i] = \text{false}) \land (\text{Need}_i \leq \text{Work})$$

   - If no such i exists, go to **Step 4**

3. Run the following, then go to **Step 2**

$$\text{Work} = \text{Work} + \text{Allocation}_i$$
$$\text{Finish}[i] = \text{true}$$

4. If $\forall i,\ \text{Finish}[i] == \text{true}$, then the system is in a safe state

Resource-Request Algorithm for Process P_i (Banker's Algorithm)

Request_i = request vector for process P_i

- If $\text{Request}_i[j] = k$, then process P_i wants k instances of resource type R_j

1. If $\text{Request}_i \leq \text{Need}_i$ go to **Step 2**
   - Otherwise, raise error condition, since process has exceeded its maximum claim

2. If $\text{Request}_i \leq \text{Available}$, 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

$$\begin{aligned} \text{Available} &= \text{Available} - \text{Request}_i \\ \text{Allocation}_i &= \text{Allocation}_i + \text{Request}_i \\ \text{Need}_i &= \text{Need}_i - \text{Request}_i \end{aligned}$$

4. Check for safety
   - If safe, the resources are allocated to P_i
   - If unsafe, P_i must wait, and the old resource-allocation state is restored

Banker's Algorithm Example

5 processes, P_0 through P_4

3 resources

- 10 instances of resource A
- 5 instances of resource B
- 7 instances of resource C
- **Available** = $(3, 3, 2)$

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3			
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Banker's Algorithm Example

5 processes, P_0 through P_4

3 resources

- 10 instances of resource A
- 5 instances of resource B
- 7 instances of resource C
- **Available** = $(3, 3, 2)$

	Allocation			Max			Need		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P_0	0	1	0	7	5	3	7	4	3
P_1	2	0	0	3	2	2	1	2	2
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Banker's Algorithm Example

5 processes, P_0 through P_4

3 resources

- 10 instances of resource A
- 5 instances of resource B
- 7 instances of resource C
- **Available** = $(3, 3, 2)$

	Allocation			Max			Need		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P_0	0	1	0	7	5	3	7	4	3
P_1	2	0	0	3	2	2	1	2	2
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available = **Work** = $(3, 3, 2)$

P_1: $(1, 2, 2) \leq (3, 3, 2)$
- When done, Work = $(5, 3, 2)$

P_3: $(0, 1, 1) \leq (5, 3, 2)$
- When done, Work = $(7, 4, 3)$

P_4: $(4, 3, 1) \leq (7, 4, 3)$
- When done, Work = $(7, 4, 5)$

P_2: $(6, 0, 0) \leq (7, 4, 5)$
- When done, Work = $(10, 4, 7)$

P_0: $(7, 4, 3) \leq (10, 4, 7)$
- When done, Work = $(10, 5, 7)$

The system is in a **safe state** since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria

Banker's Algorithm Example

Suppose now at this safe state, P_1 requests for $(1, 0, 2)$

Can this be fulfilled?

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3			
P_1	3	0	2	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Banker's Algorithm Example

Suppose now at this safe state, P_1 requests for $(1, 0, 2)$

Perform checks

- $\text{Request}_1 = (1, 0, 2) \leq (1, 2, 2)$, hence $\text{Request}_i \leq \text{Need}_i$
- $\text{Request}_1 = (1, 0, 2) \leq (3, 3, 2)$, hence $\text{Request}_i \leq \text{Available}$

Pretend that request has been fulfilled to arrive at the following new state, with **Available** = $(2, 3, 0)$

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3			
P_1	3	0	2	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Banker's Algorithm Example

Suppose now at this safe state, P_1 requests for $(1, 0, 2)$

Perform checks

- $\text{Request}_1 = (1, 0, 2) \leq (1, 2, 2)$, hence $\text{Request}_i \leq \text{Need}_i$
- $\text{Request}_1 = (1, 0, 2) \leq (3, 3, 2)$, hence $\text{Request}_i \leq \text{Available}$

Pretend that request has been fulfilled to arrive at the following new state, with **Available** = $(2, 3, 0)$

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3
P_1	3	0	2	3	2	2	0	2	0
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Banker's Algorithm Example

Suppose now at this safe state, P_1 requests for $(1, 0, 2)$

Perform checks

- $\text{Request}_1 = (1, 0, 2) \leq (1, 2, 2)$, hence $\text{Request}_i \leq \text{Need}_i$
- $\text{Request}_1 = (1, 0, 2) \leq (3, 3, 2)$, hence $\text{Request}_i \leq \text{Available}$

Pretend that request has been fulfilled to arrive at the following new state, with **Available** = $(2, 3, 0)$

	Allocation			Max			Need		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3
P_1	3	0	2	3	2	2	0	2	0
P_2	3	0	2	9	0	2	6	0	0
P_3	2	1	1	2	2	2	0	1	1
P_4	0	0	2	4	3	3	4	3	1

Available = **Work** = $(2, 3, 0)$

P_1: $(0, 2, 0) \leq (2, 3, 0)$; Work = $(5, 3, 2)$
P_3: $(0, 1, 1) \leq (5, 3, 2)$; Work = $(7, 4, 3)$
P_4: $(4, 3, 1) \leq (7, 4, 3)$; Work = $(7, 4, 5)$
P_0: $(7, 4, 3) \leq (7, 4, 5)$; Work = $(7, 5, 5)$
P_2: $(6, 0, 0) \leq (7, 5, 5)$; Work = $(10, 5, 7)$

The system is in a **safe state** since the sequence $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfies safety criteria

Continuing from here

- Can a request for $(3, 3, 0)$ by P_4 be granted?
- Can a request for $(0, 2, 0)$ by P_0 be granted?

Deadlock Detection and Recovery

Deadlock Detection and Recovery

Goal: Allow system to enter deadlock state

Need to have

- Detection algorithm
- Recovery scheme

If only single instances of a resource type, use a *wait-for graph*

If multiple instances of a resource type exist, use a *variation of the safety algorithm* from the banker's algorithm earlier

Wait-For Graph

Nodes in a **wait-for graph** are processes

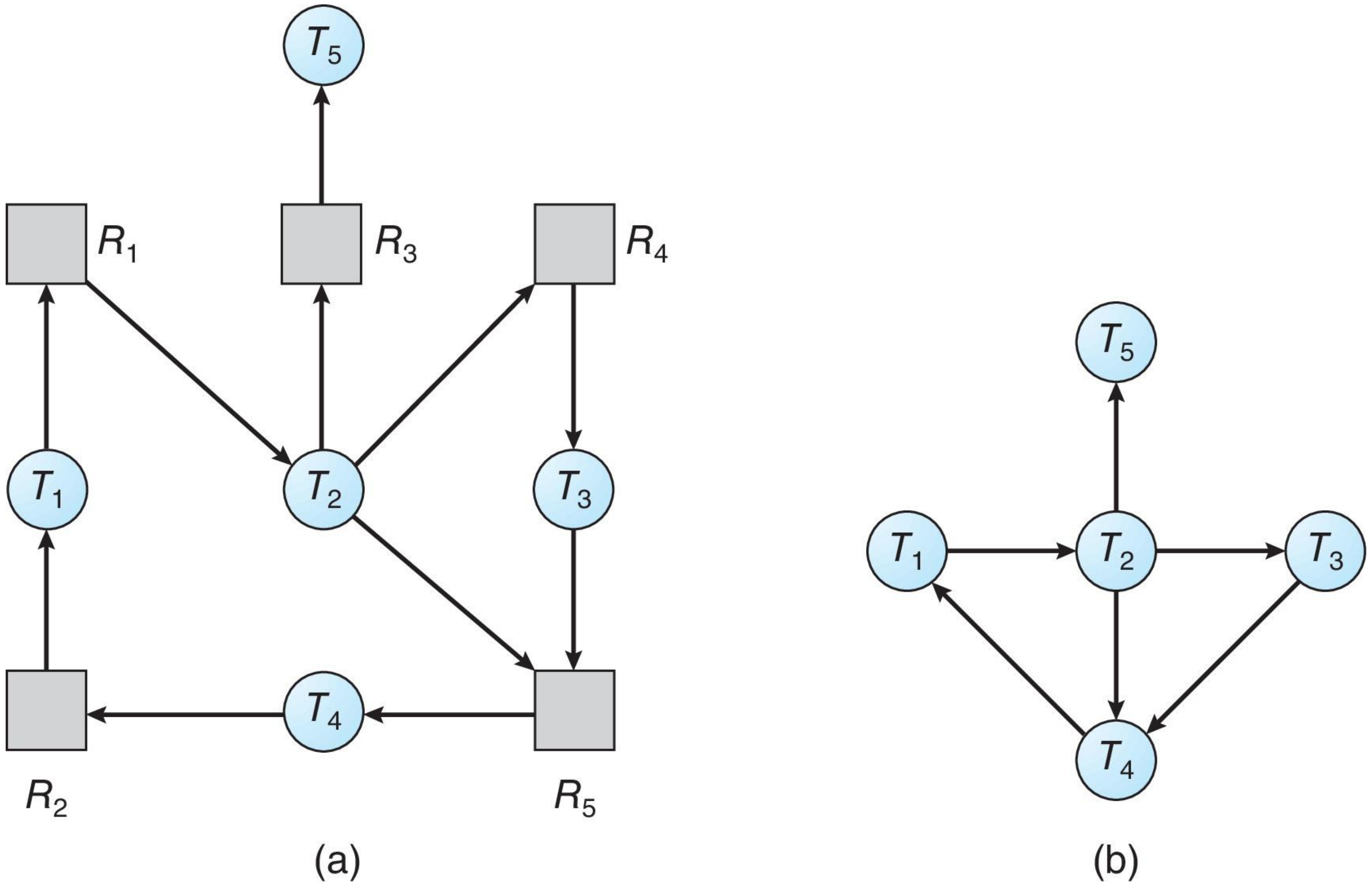
$P_i \rightarrow P_j$ if P_i is waiting for P_j

- Exist if and only if corresponding resource-allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q

If there is a **cycle** in the wait-for graph, there exists a deadlock

Example: Resource-allocation graph (a) and corresponding wait-for graph (b)

- **Is there a deadlock?**



Variation of Safety Algorithm (Banker's Algorithm) for Deadlock Detection

Available and **Allocation** as per original algorithm

Request: $n \times m$ matrix indicating the current resource request of each process

- If $\text{Request}[i, j] = k$, then P_i is need k more instances of R_j

Let **Work** and **Finish** be vectors of length m and n, respectively

1. Initialize

$$\text{Work} = \text{Available}$$
$$\text{Finish}[i] = \text{false}, \qquad \forall i = 1, 2, \dots, n$$

2. Find an i such that
   $(\text{Finish}[i] = \text{false}) \wedge (\text{Request}_i \leq \text{Work})$
   - If no such i exists, go to **Step 4**

3. Run the following, then go to **Step 2**

$$\text{Work} = \text{Work} + \text{Allocation}_i$$
$$\text{Finish}[i] = \text{true}$$

4. If $\exists i,\ \text{Finish}[i] == \text{false}$, then the system is in a deadlocked state
   - In particular, if $\text{Finish}[i] == \text{false}$, then P_i is deadlocked

Deadlock Detection Using Variation of Safety Algorithm Example

5 processes, P_0 through P_4

3 resources
- 7 instances of resource A
- 2 instances of resource B
- 6 instances of resource C
- **Available** = $(0, 0, 0)$

	Allocation			Request		
	A	**B**	**C**	**A**	**B**	**C**
P_0	0	1	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

Deadlock Detection Using Variation of Safety Algorithm Example

5 processes, P_0 through P_4

3 resources

- 7 instances of resource A
- 2 instances of resource B
- 6 instances of resource C
- **Available** = $(0, 0, 0)$

	Allocation			Request		
	A	**B**	**C**	**A**	**B**	**C**
P_0	0	1	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

Available = **Work** = $(0, 0, 0)$

P_0: $(0, 0, 0) \leq (0, 0, 0)$
- When done, Work = $(0, 1, 0)$

P_2: $(0, 0, 0) \leq (0, 1, 0)$
- When done, Work = $(3, 1, 3)$

P_3: $(1, 0, 0) \leq (3, 1, 3)$
- When done, Work = $(5, 2, 4)$

P_1: $(2, 0, 2) \leq (5, 2, 4)$
- When done, Work = $(7, 2, 4)$

P_4: $(0, 0, 2) \leq (7, 2, 4)$
- When done, Work = $(7, 2, 6)$

The system is **not in a deadlocked state** since the sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ satisfies safety criteria

Deadlock Detection Using Variation of Safety Algorithm Example

5 processes, P_0 through P_4

3 resources

- 7 instances of resource A
- 2 instances of resource B
- 6 instances of resource C
- **Available** = $(0, 0, 0)$

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

Deadlock Detection Using Variation of Safety Algorithm Example

5 processes, P_0 through P_4

3 resources

- 7 instances of resource A
- 2 instances of resource B
- 6 instances of resource C
- **Available** = $(0, 0, 0)$

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

Available = **Work** = $(0, 0, 0)$

P_0: $(0, 0, 0) \leq (0, 0, 0)$, Work = $(0, 1, 0)$

Cannot continue, no suitable process available

The system is **in a deadlocked state**

- In particular, P_1, P_2, P_3, and P_4 are deadlocked

Recovery from Deadlock

Process termination

- Abort *all* deadlocked processes?
  - Surefire way to break deadlock, but at what expense?
- Abort one process at a time until the deadlock cycle is eliminated?
  - Need to run deadlock detection after every abort
- 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
  - Is process interactive or batch?

Resource preemption

- Selecting a victim
  - Which resources and which processes are to be preempted?
- Rollback
  - What should be done with that process?
  - Return to some safe state, then restart process from that state?
  - Restart from beginning?
- Starvation
  - How to guarantee that resources will not always be preempted from the same process?

Questions?

Thank You!

Weihan.Goh {at} Singaporetech.edu.sg

https://www.singaporetech.edu.sg/directory/faculty/weihan-goh

https://sg.linkedin.com/in/weihan-goh