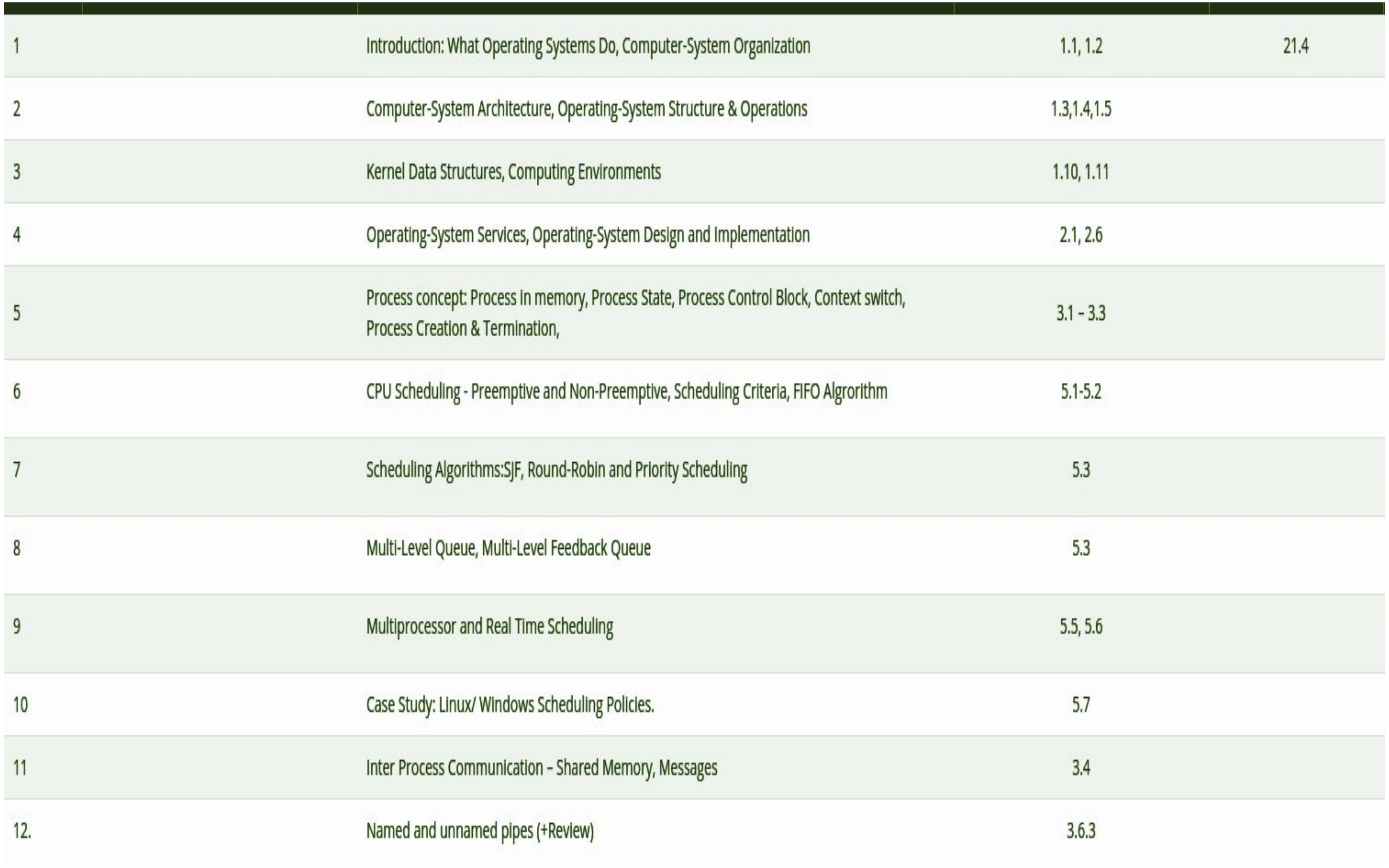
OPERATING SYSTEMS

Unit1_Unit2_Unit3: Revision Class #4

Nitin V Pujari
Faculty, Computer Science
Dean , IQAC, PES University

OPERATING SYSTEMS

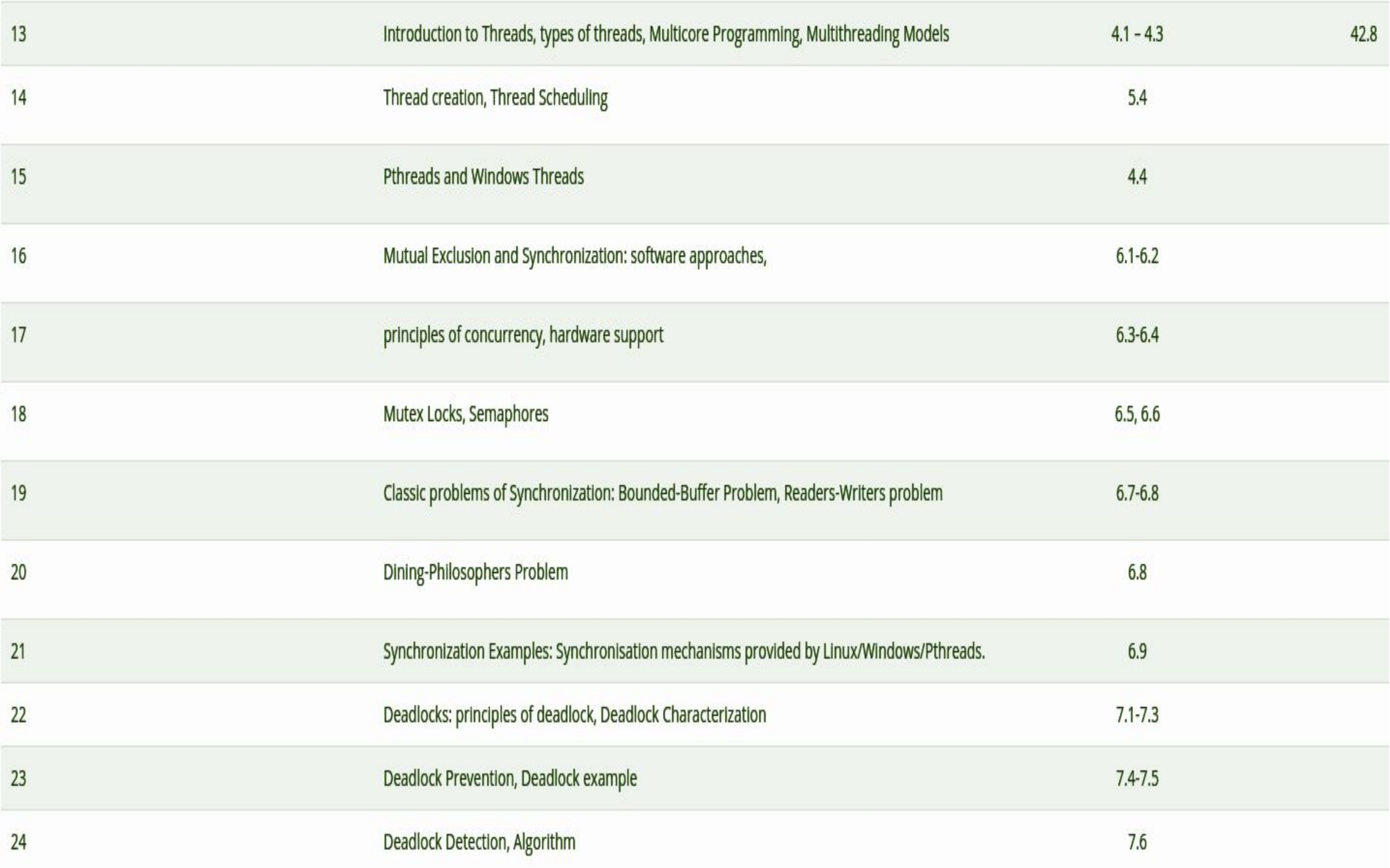
Course Syllabus => Unit 1



1		Introduction: What Operating Systems Do, Computer-System Organization	1.1, 1.2	21.4
2		Computer-System Architecture, Operating-System Structure & Operations	1.3,1.4,1.5	
3		Kernel Data Structures, Computing Environments	1.10, 1.11	
4		Operating-System Services, Operating-System Design and Implementation	2.1, 2.6	
5		Process concept: Process in memory, Process State, Process Control Block, Context switch, Process Creation & Termination,	3.1 - 3.3	
6		CPU Scheduling - Preemptive and Non-Preemptive, Scheduling Criteria, FIFO Algrorithm	5.1-5.2	
7		Scheduling Algorithms:SJF, Round-Robin and Priority Scheduling	5.3	
8		Multi-Level Queue, Multi-Level Feedback Queue	5.3	
9		Multiprocessor and Real Time Scheduling	5.5, 5.6	
10		Case Study: Linux/ Windows Scheduling Policies.	5.7	
11		Inter Process Communication - Shared Memory, Messages	3.4	
12.		Named and unnamed pipes (+Review)	3.6.3	

OPERATING SYSTEMS

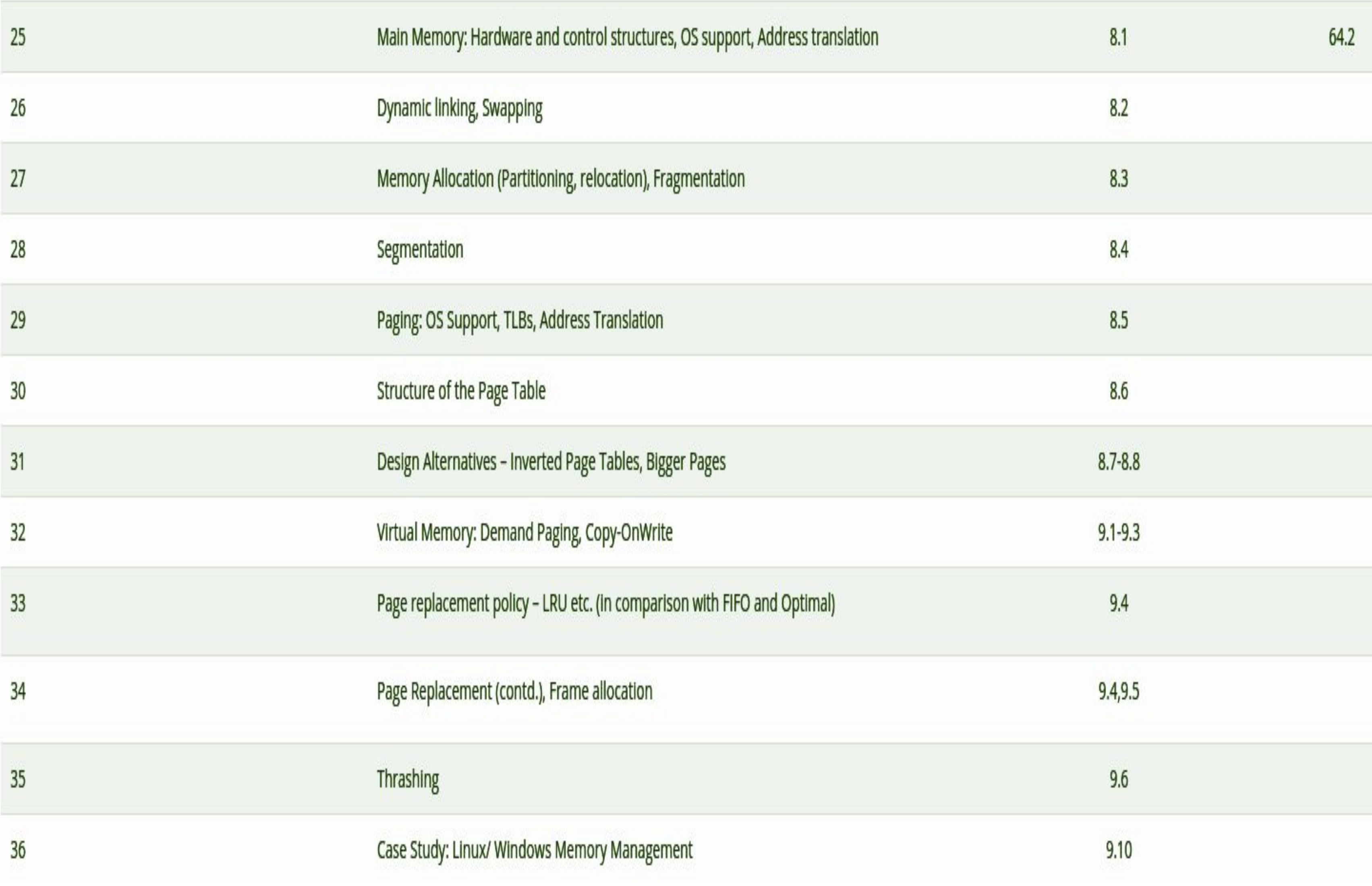
Course Syllabus => Unit 2



13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

OPERATING SYSTEMS

Course Syllabus => Unit 3



25	Main Memory: Hardware and control structures, OS support, Address translation	8.1	64.2
26	Dynamic linking, Swapping	8.2	
27	Memory Allocation (Partitioning, relocation), Fragmentation	8.3	
28	Segmentation	8.4	
29	Paging: OS Support, TLBs, Address Translation	8.5	
30	Structure of the Page Table	8.6	
31	Design Alternatives - Inverted Page Tables, Bigger Pages	8.7-8.8	
32	Virtual Memory: Demand Paging, Copy-OnWrite	9.1-9.3	
33	Page replacement policy - LRU etc. (In comparison with FIFO and Optimal)	9.4	
34	Page Replacement (contd.), Frame allocation	9.4,9.5	
35	Thrashing	9.6	
36	Case Study: Linux/ Windows Memory Management	9.10	

Revision Class Outline => Student Queries

- How valid is the diagram which shows the stack and heap expanding towards each other given that they can be on different segments ?

- Sir, in the textbook for shortest job first/next, they have spoken about a formula to find the CPU burst time of the next process (exponential average), could you explain that ?

- Sir, can you explain how the CPU burst prediction is done in detail ?

- Comparative study on various CPU scheduling strategies => like advantages and disadvantages

Revision Class Outline => Student Queries

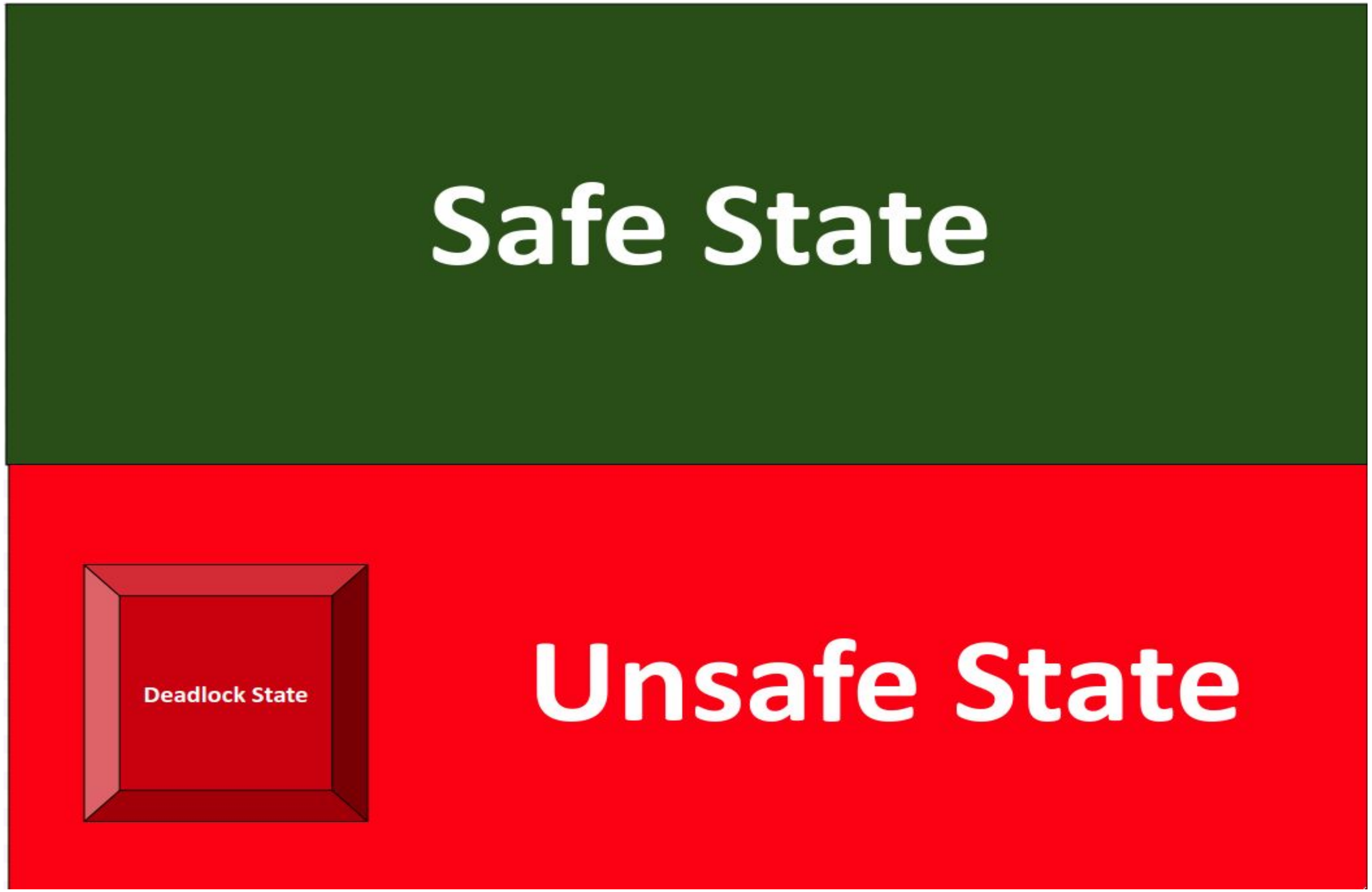
- Revision on Real time scheduling sir
- Revision for Round Robin Scheduling
- Recursive Mutexes, Semaphores, Spinlocks
- Revision of Banker's safety algorithm for detecting deadlocks
- Revision of Banker's Request Resource algorithm

Revision Class Outline => Student Queries

- Page replacement algorithm in general
- Demand paging performance calculation
- Inverted page table

Revision Class Outline => Student Queries

- **Safe and Unsafe State**



Safe State

Deadlock State

Unsafe State

Safe State

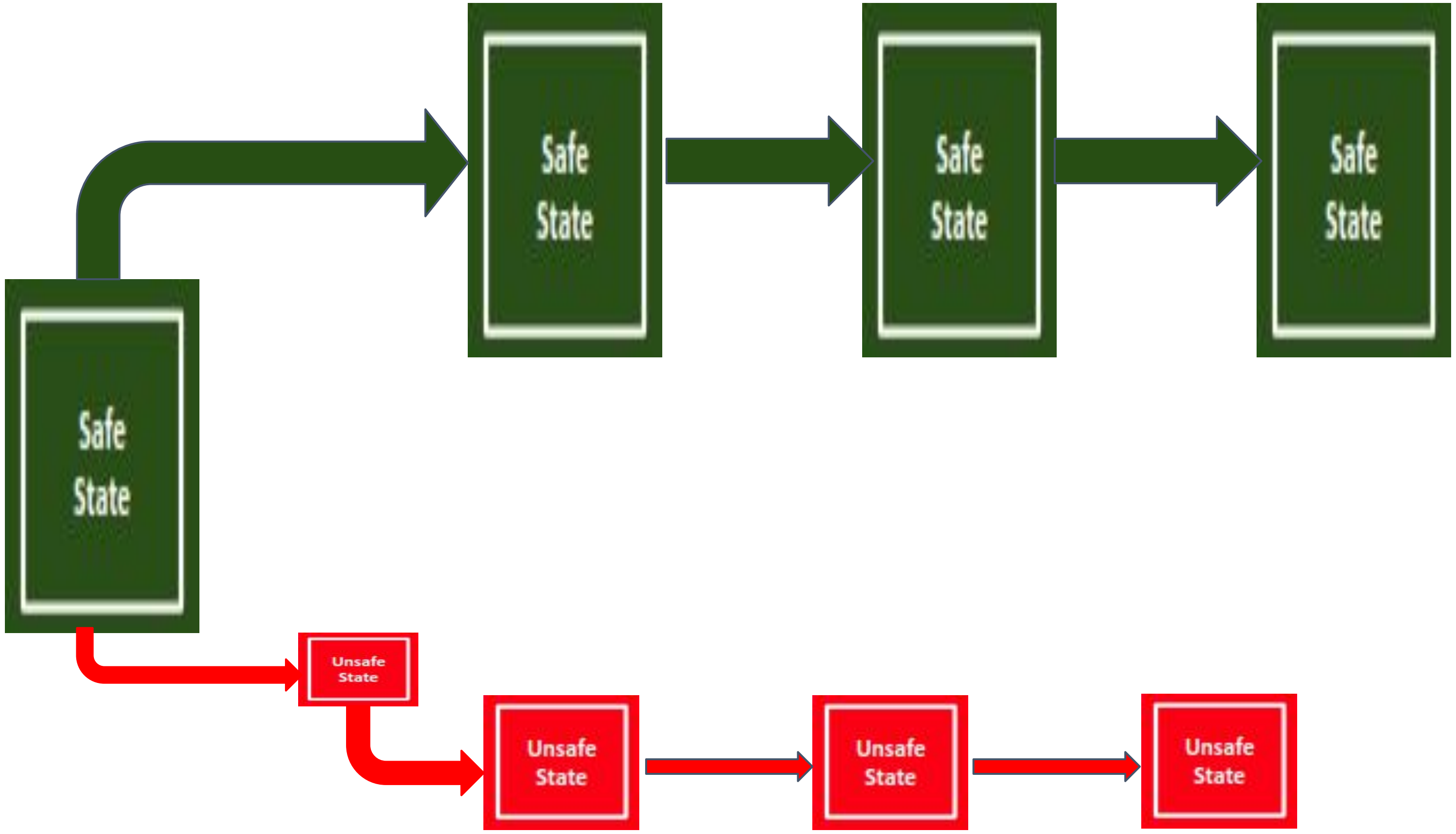
- **Safe state:** A safe state is a state in which all the processes can be executed in some arbitrary order with the available resources such that no deadlock occurs.

1. If it is a safe state, then the requested resources are allocated to the process in actual.

2. If the resulting state is an unsafe state then it rollbacks to the previous state and the process is asked to wait longer.

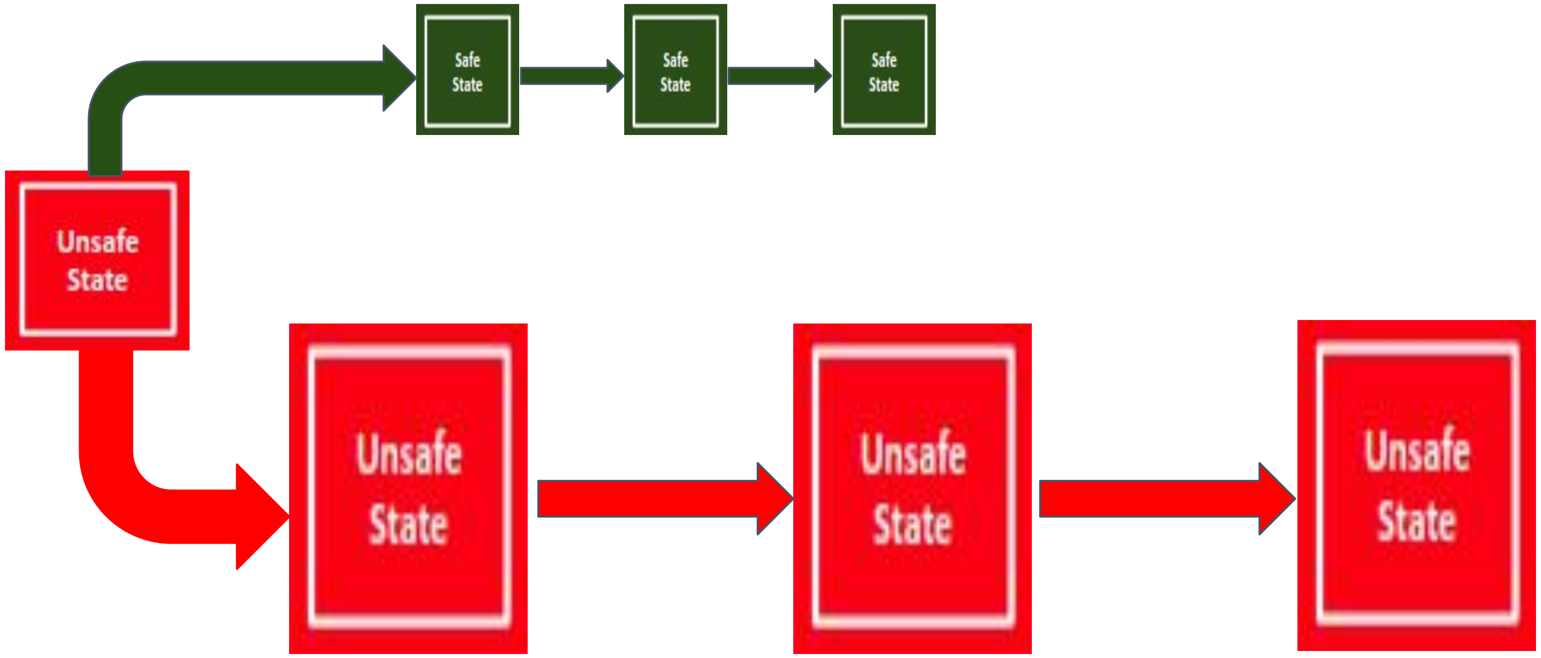
Revision Class Outline => Student Queries

Safe to Unsafe State



Revision Class Outline => Student Queries

Unsafe State to Safe State



Banker's Algorithm

- Banker’s Algorithm is a **Deadlock Avoidance algorithm**.
- It is also used for **Deadlock Detection**.
- This algorithm tells that if any system can go into a deadlock or not by analyzing the currently allocated resources and the resources required by it in the future.
- This algorithm can be used when several instance of resources are present, which is typically cannot be handled by RAG
- There are various data structures which are used to implement this algorithm

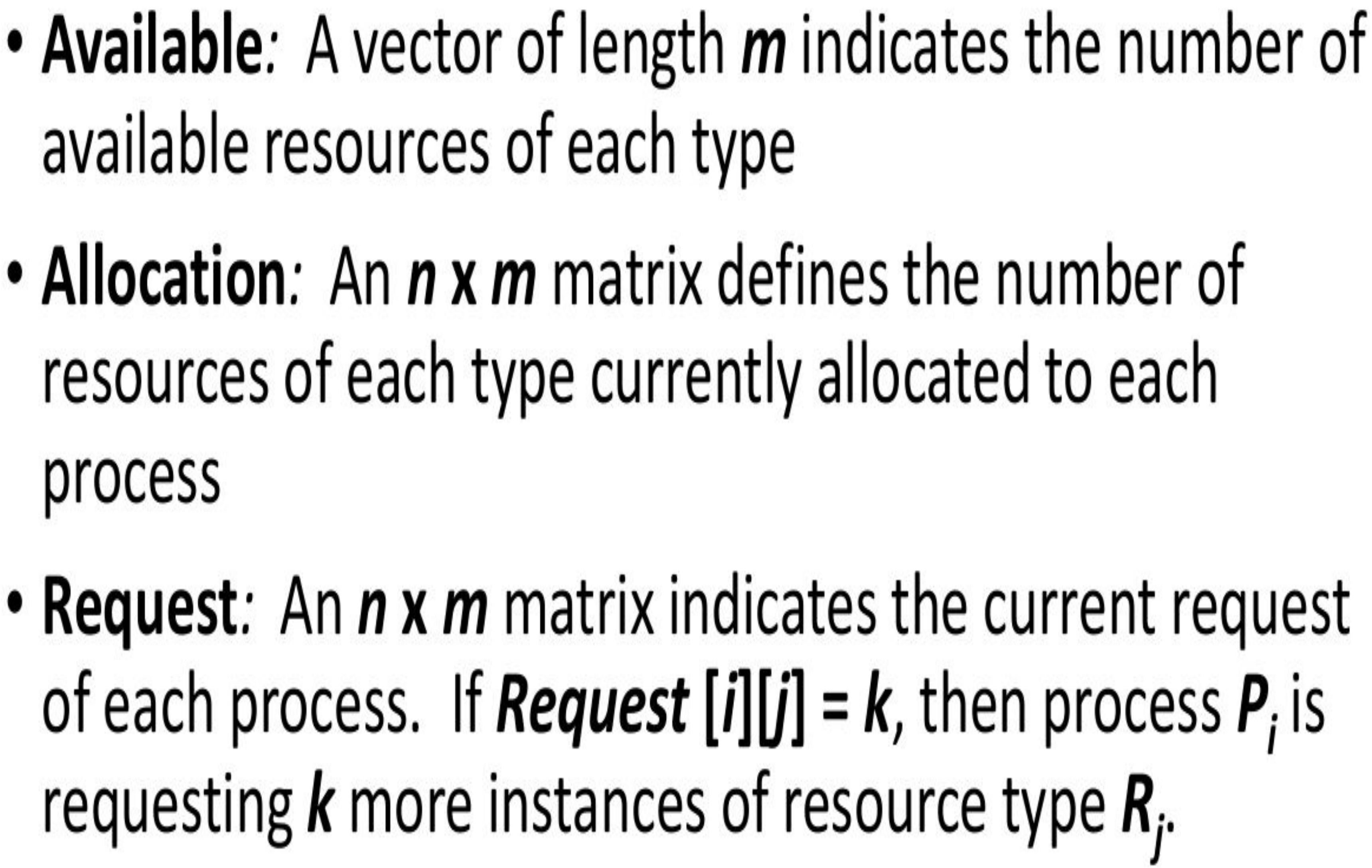
Banker’s Algorithm

The Bankers Algorithm consists of the following two algorithms

1. Safety Algorithm

2. Request-Resource Algorithm / Resource-Request Algorithm

Revision Class Outline => Student Queries



- **Available**: A vector of length ***m*** indicates the number of available resources of each type
- **Allocation**: An ***n* x *m*** matrix defines the number of resources of each type currently allocated to each process
- **Request**: An ***n* x *m*** matrix indicates the current request of each process. If ***Request* [*i*][*j*] = *k***, then process P_i is requesting ***k*** more instances of resource type R_j.

Banker’s Algorithm: Resource Request Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:

   (a) ***Work = Available***

   (b) For ***i* = 1,2, …, *n***, if $\boldsymbol{\mathit{Allocation}_i \neq 0}$, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

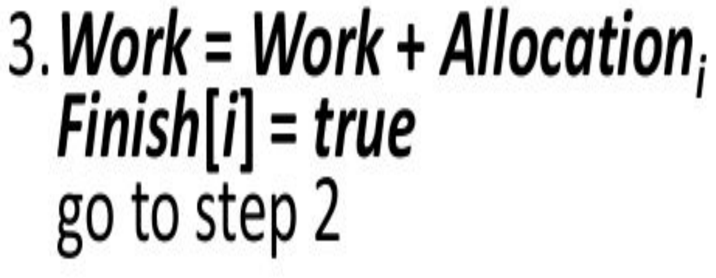
2. Find an index ***i*** such that both:

   (a) ***Finish*[*i*] == *false***

   (b) $\boldsymbol{\mathit{Request}_i \leq \mathit{Work}}$

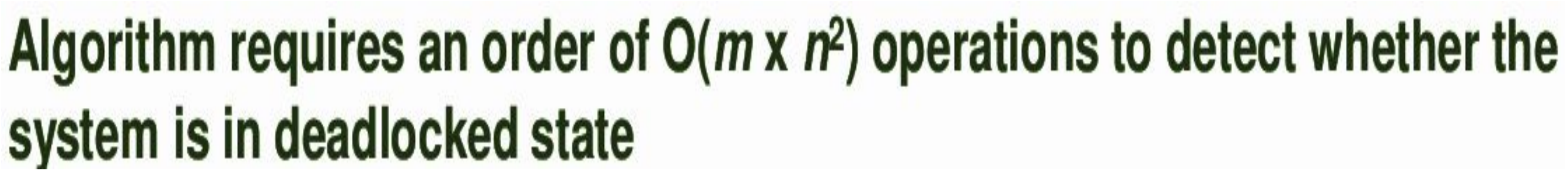
   If no such ***i*** exists, go to step 4

Banker’s Algorithm: Resource Request Algorithm



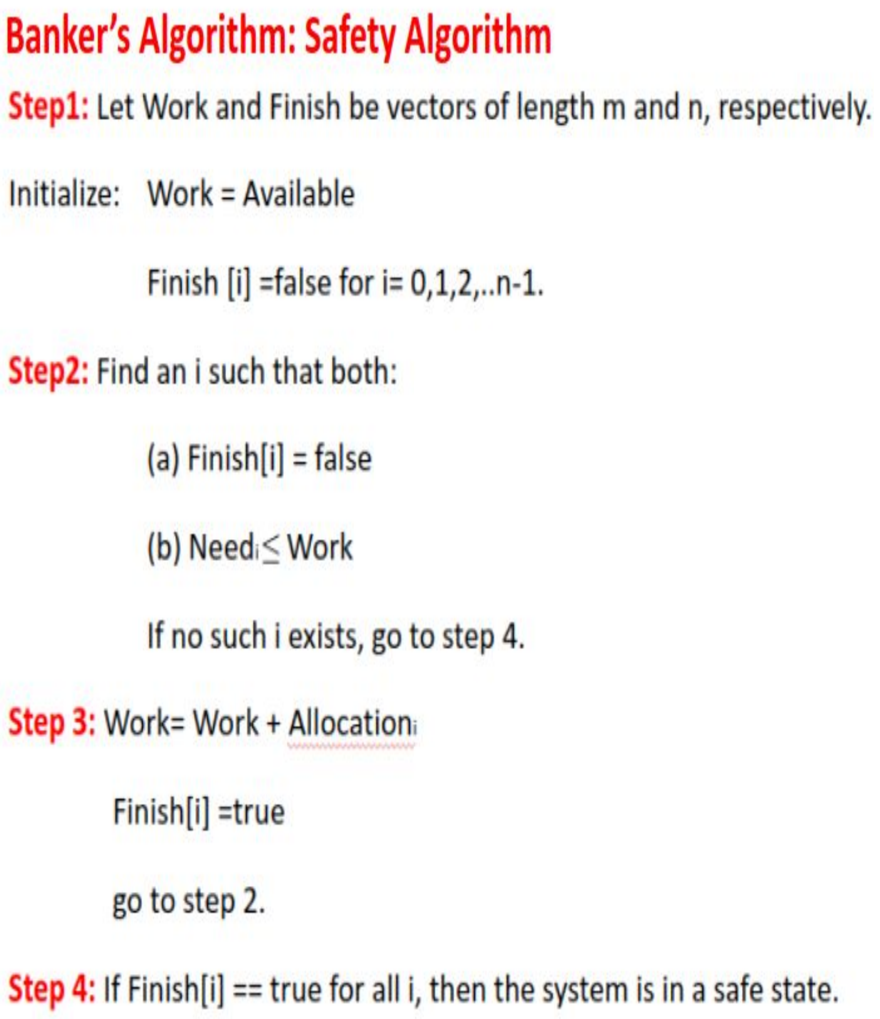
3. *Work* = *Work* + $Allocation_i$
   Finish[*i*] = *true*
   go to step 2

4. If ***Finish[i] == false***, for some ***i***, 1 ≤ ***i*** ≤ ***n***, then the system is in deadlock state. Moreover, if ***Finish***[***i***] **==** ***false***, then P_i is deadlocked



Algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state

Revision Class Outline => Student Queries



Banker's Algorithm: Safety Algorithm

Step1: Let Work and Finish be vectors of length m and n, respectively.

Initialize: Work = Available

Finish [i] =false for i= 0,1,2,..n-1.

Step2: Find an i such that both:

(a) Finish[i] = false

(b) $\text{Need}_i \leq$ Work

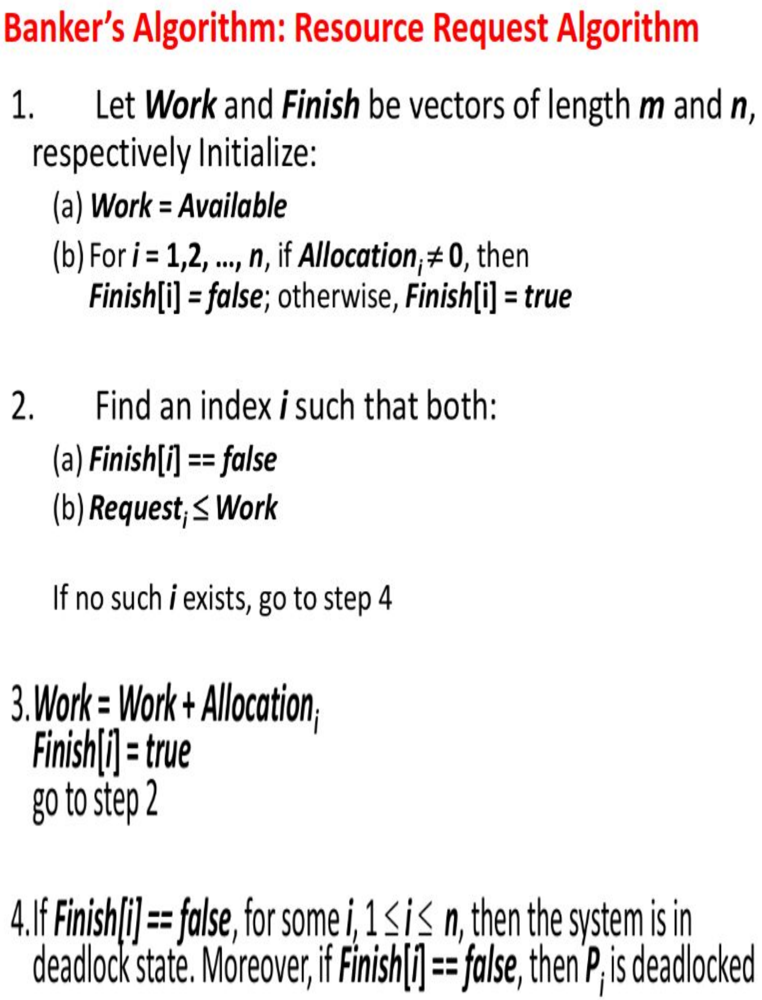
If no such i exists, go to step 4.

Step 3: Work= Work + Allocation_i

Finish[i] =true

go to step 2.

Step 4: If Finish[i] == true for all i, then the system is in a safe state.



Banker's Algorithm: Resource Request Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:

   (a) ***Work = Available***

   (b) For ***i* = 1,2, …, *n***, if $\textbf{\textit{Allocation}}_i \neq \mathbf{0}$, then ***Finish***[i] = ***false***; otherwise, ***Finish***[i] = ***true***

2. Find an index ***i*** such that both:

   (a) ***Finish***[***i***] == ***false***

   (b) $\textbf{\textit{Request}}_i \leq \textbf{\textit{Work}}$

   If no such ***i*** exists, go to step 4

3. $\textbf{\textit{Work}} = \textbf{\textit{Work}} + \textbf{\textit{Allocation}}_i$
   Finish[***i***] = ***true***
   go to step 2

4. If ***Finish***[***i***] == ***false***, for some ***i***, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if ***Finish***[***i***] == ***false***, then P_i is deadlocked

Example - Banker's Request Resource Algorithm

RMax		
A	B	C
5	2	4

Available=>Rmax-Allocated		
A	B	C
1	0	2

n	4
i	1 to n

Process	Allocation			Max		
	A	B	C	A	B	C
P1	2	0	1	3	1	1
P2	0	1	0	0	1	1
P3	1	0	1	3	1	1
P4	1	1	0	1	1	3
Total	4	2	2	4	4	6

Process	Need = Max(i) - Allocated(i)		
	A	B	C
P1	1	1	0
P2	0	0	1
P3	2	1	0
P4	0	0	3
Total	3	2	4

Example - Banker's Request Resource Algorithm

RMax		
A	B	C
5	2	4

Available=>Rmax-Allocated		
A	B	C
1	0	2

n	4
i	4

Work = Available		
A	B	C
1	0	2

Process	Need = Max(i) - Allocated(i)		
	A	B	C
P1	1	1	0
P2	0	0	1
P3	2	1	0
P4	0	0	3
Total	3	2	4

Process	Allocation			Request		
	A	B	C	A	B	C
P1	2	0	1	1	1	0
P2	0	1	0	0	0	0
P3	1	0	1	1	1	0
P4	1	1	0	5	0	0
Total	4	2	2	3	2	0

Process	Flag
P1	False
P2	False
P3	False
P4	False

Safe Sequence			

Example - Banker's Request Resource Algorithm

RMax		
A	B	C
5	2	4

Available=>Rmax-Allocated		
A	B	C
1	0	2

n	4
i	4

Work = Available		
A	B	C
4	1	4

Process	Need = Max(i) - Allocated(i)		
	A	B	C
P1	1	1	0
P2	0	0	1
P3	2	1	0
P4	0	0	3
Total	3	2	4

Process	Allocation			Request		
	A	B	C	A	B	C
P1	2	0	1	1	1	0
P2	0	1	0	0	0	0
P3	1	0	1	1	1	0
P4	1	1	0	5	0	0
Total	4	2	2	3	2	0

Process	Flag
P1	True
P2	True
P3	True
P4	False

Safe Sequence			

Error !!
because Request4
>Need4

Example - Banker’s Request Resource Algorithm

RMax		
A	B	C
5	2	4

Available=>Rmax-Allocated		
A	B	C
1	0	2

n	4
i	4

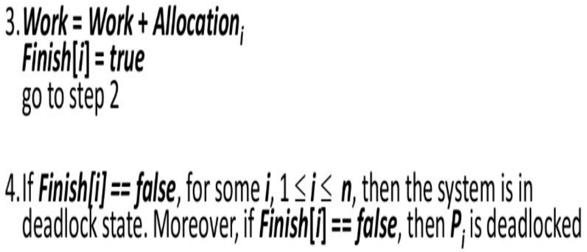
Work = Available		
A	B	C
5	2	4

Process	Need = Max(i) - Allocated(i)		
	A	B	C
P1	1	1	0
P2	0	0	1
P3	2	1	0
P4	0	0	3
Total	3	2	4

Process	Allocation			Request		
	A	B	C	A	B	C
P1	2	0	1	1	1	0
P2	0	1	0	0	0	0
P3	1	0	1	1	1	0
P4	1	1	0	0	0	1
Total	4	2	2	3	2	0

Process	Flag
P1	True
P2	True
P3	True
P4	True

Safe Sequence			
P2	P1	P3	P4



3. *Work* = *Work* + *Allocation*$_i$
   Finish[*i*] = *true*
   go to step 2

4. If *Finish*[*i*] == *false*, for some *i*, $1 \le i \le n$, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked

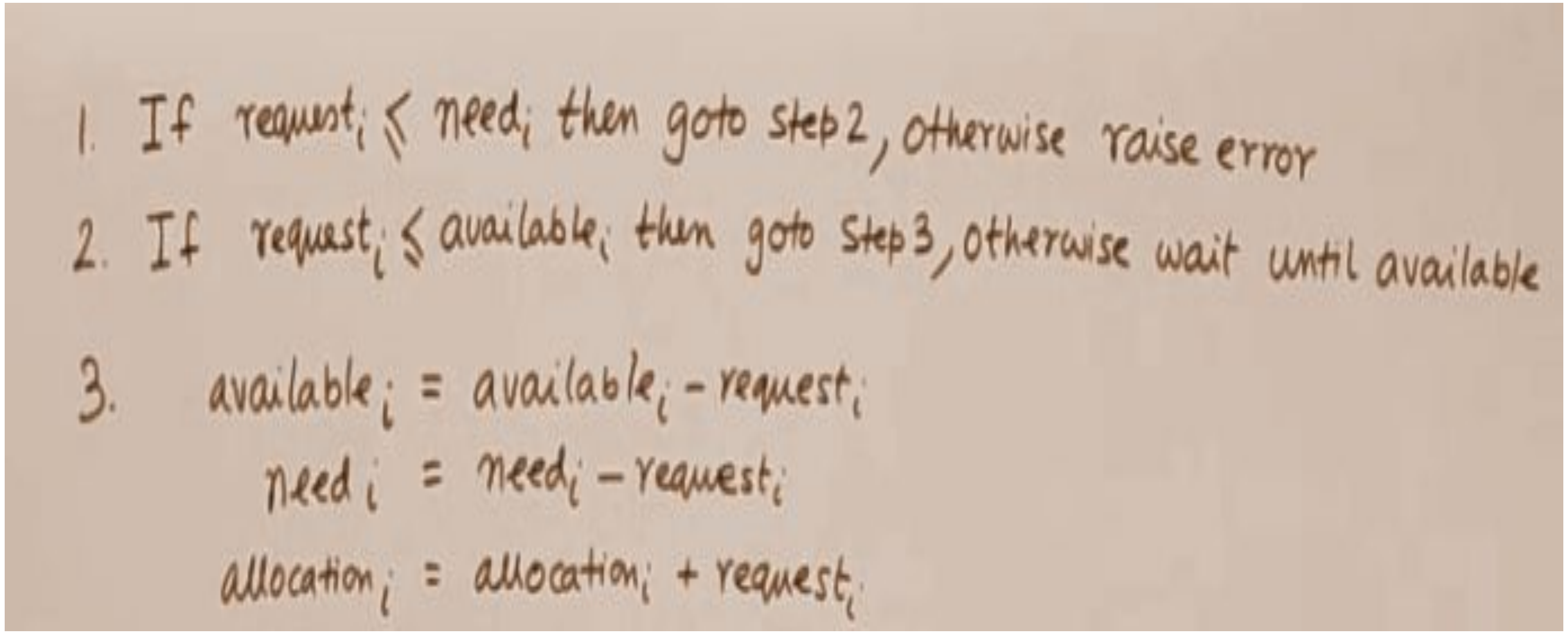
Banker’s Algorithm: Resource Request Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:
   (a) ***Work = Available***
   (b) For ***i* = 1,2, …, *n***, 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***$_i \le$ ***Work***

   If no such ***i*** exists, go to step 4

Revision Class Outline => Student Queries



1. If $request_i \leq need_i$ then goto step 2, otherwise raise error

2. If $request_i \leq available_i$ then goto step 3, otherwise wait until available

3. $available_i = available_i - request_i$

   $need_i = need_i - request_i$

   $allocation_i = allocation_i + request_i$

OPERATING SYSTEMS

Revision Class Outline => Student Queries

Example - Banker's Request Resource Algorithm

RMax		
A	B	C
5	2	4

Available=>Rmax-Allocated		
A	B	C
1	0	2

n	4
i	1

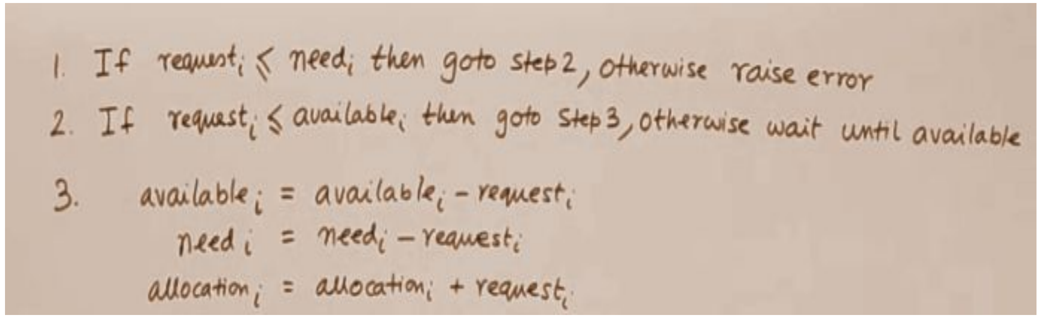
Work = Available		
A	B	C
0	1	2

Process	Need = Max(i) - Allocated(i)		
	A	B	C
P1	0	0	0
P2	0	0	1
P3	2	1	0
P4	0	0	3
Total	3	2	4

Process	Allocation			Request		
	A	B	C	A	B	C
P1	3	1	0	1	1	0
P2	0	1	0			
P3	1	0	1			
P4	1	1	0			
Total	4	2	2			

Process	Flag
P1	False
P2	False
P3	False
P4	False

Safe Sequence			



1. If $request_i \leqslant need_i$ then goto step 2, otherwise raise error
2. If $request_i \leqslant available_i$ then goto Step 3, otherwise wait until available
3. $available_i = available_i - request_i$
   $need_i = need_i - request_i$
   $allocation_i = allocation_i + request_i$

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean, IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com