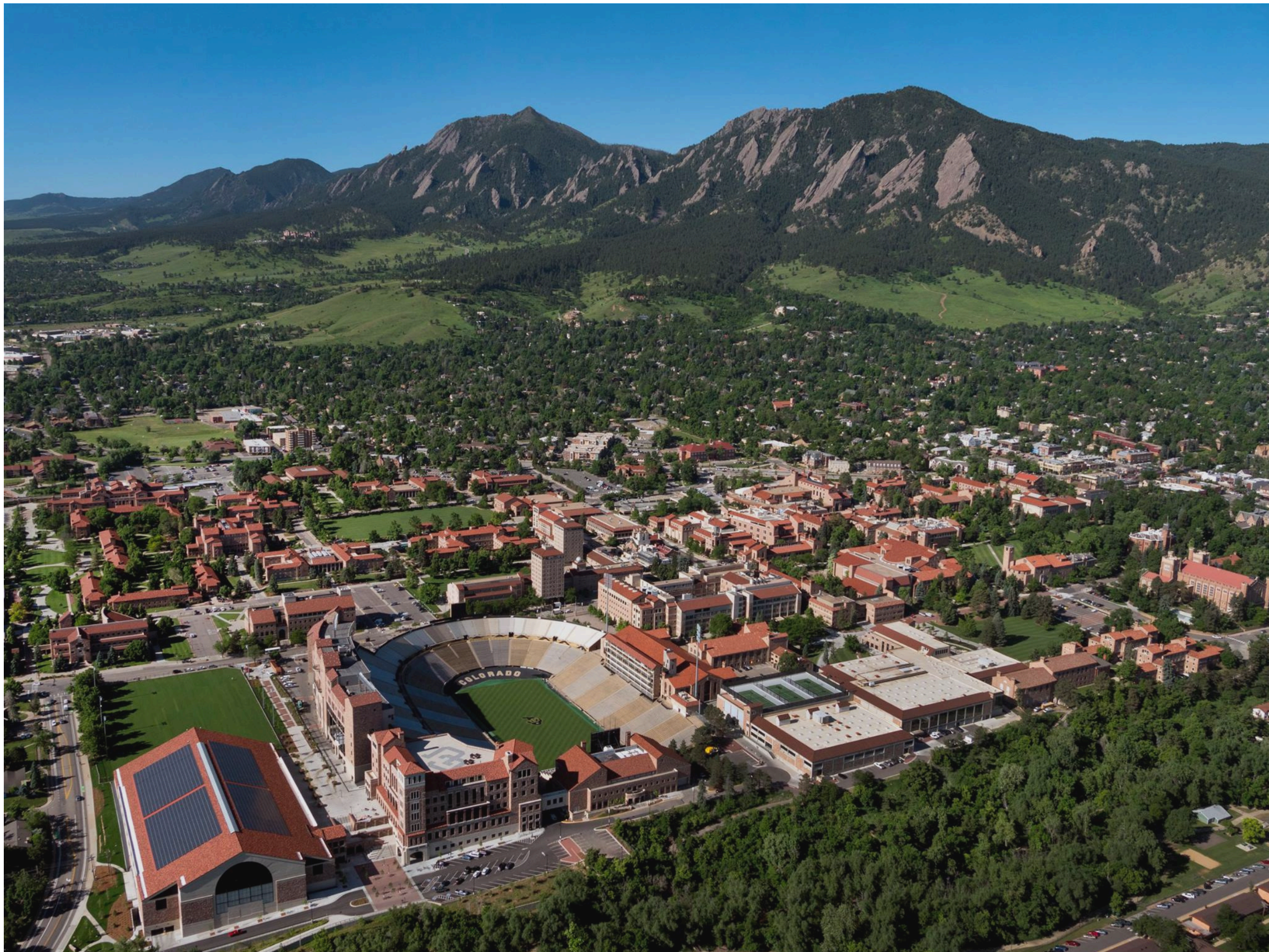
Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

These slides adapted from materials provided by the textbook authors.


COLORADO

Deadlock Avoidance

Deadlock Prevention

prevent at least 1 of necessary conditions

Mutual exclusion

Hold and wait

No preemption

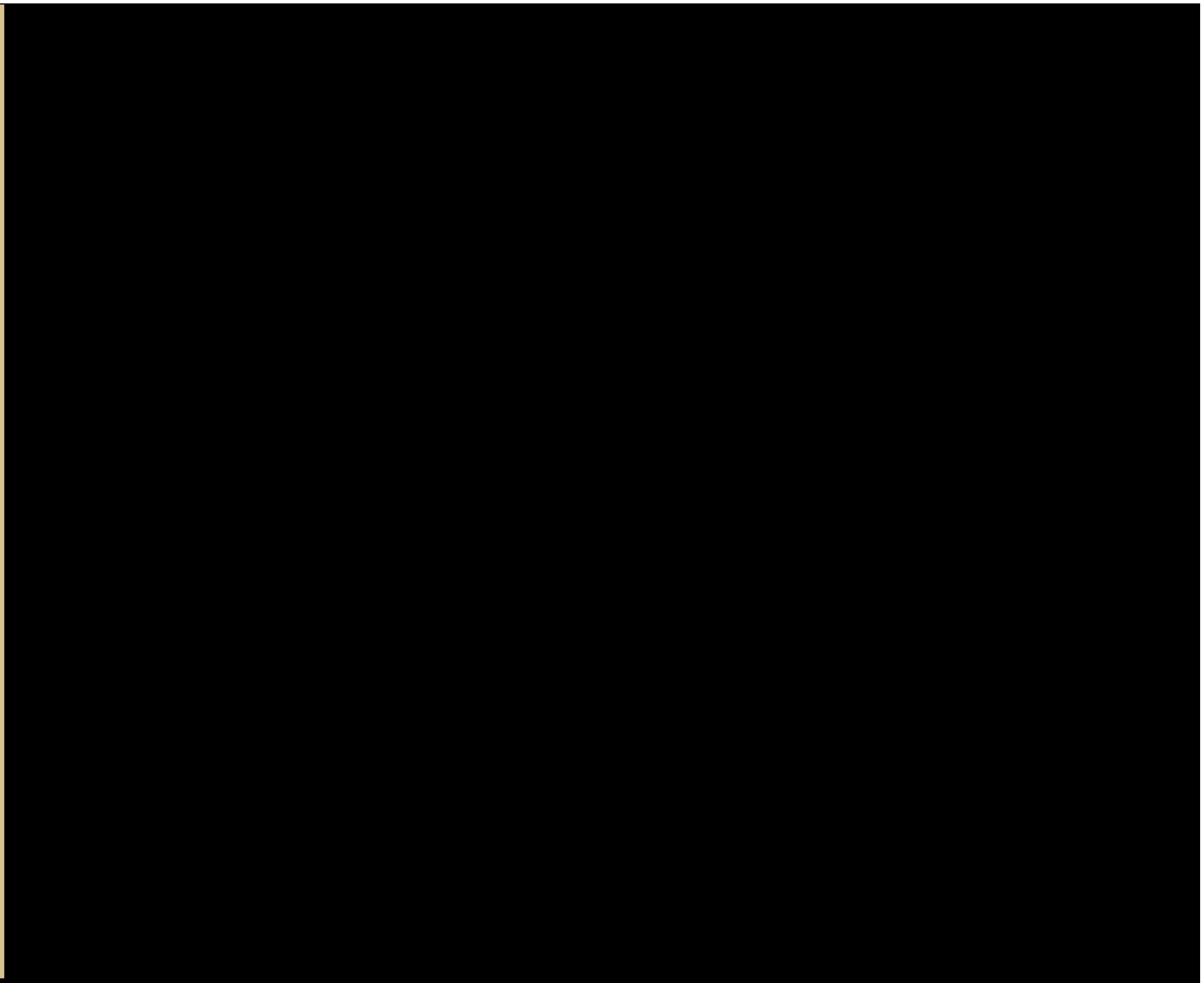
Circular dependency

Deadlock Avoidance

Can we analyze the system and detect deadlock?

Need to know resource usage required by each process (and track usage)

Let the OS deal with deadlock at runtime



Deadlock Avoidance

will need to detect deadlock within a set of processes

- *knowledge of the maximum number of resources used by each process*
- *keep track of resource usage*
- *make sure there is always a sequence of processes that could run to completion and NOT exceed the available resources even if process demands their maximum*

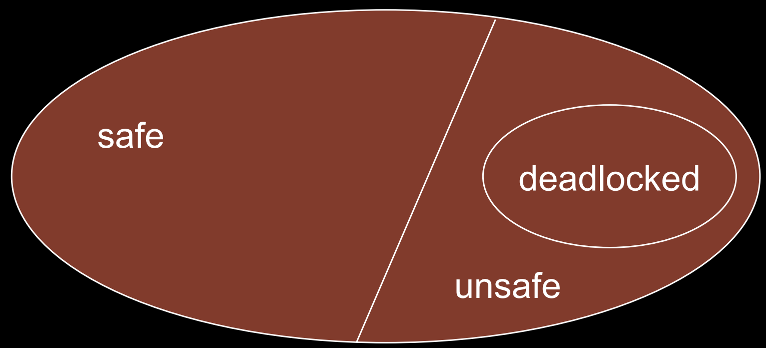
Safe State

Intuition: IF there is a way to satisfy the resource requirements for all tasks, the system cannot deadlock!

- *find a sequence of process completion that will never exceed the resources available*
- *As each process completes the resources currently allocated would be available to other processes*

Safe State

- *Most of the system states will be safe*
- *Unsafe states do not necessarily lead to deadlock*
- *Some unsafe states can lead to deadlock*



Example 1 (initial state)

processes	max needs	allocated (total=12)
P0	10	5
P1	4	2
P2	9	2

sequence <P1, P0, P2> is safe

Available

- P1 requests its max 3
  (has 2, so needs 2 more),
  now holds 4 1
- then P1 releases all 4 5

- P0 requests its max 0
  (has 5, so needs 5 more),
  now holds 10
- then P0 releases all 10 10

- P2 requests its max
  (has 2, so needs 7 more),
  holds 9 3
- then P0 releases all 9 12

Example 1 (initial state)

processes	max needs	allocated (total=12)
P0	10	5
P1	4	2
P2	9	3

Is the state still safe?

Available

- P1 requests its max 2
  (has 2, so needs 2 more),
  now holds 4 0
- then P1 releases all 4 4

- P0 requests its max ???
  (has 5, so needs 5 more)

- DEADLOCK !!!

Dijkstra's Banker's Algorithm

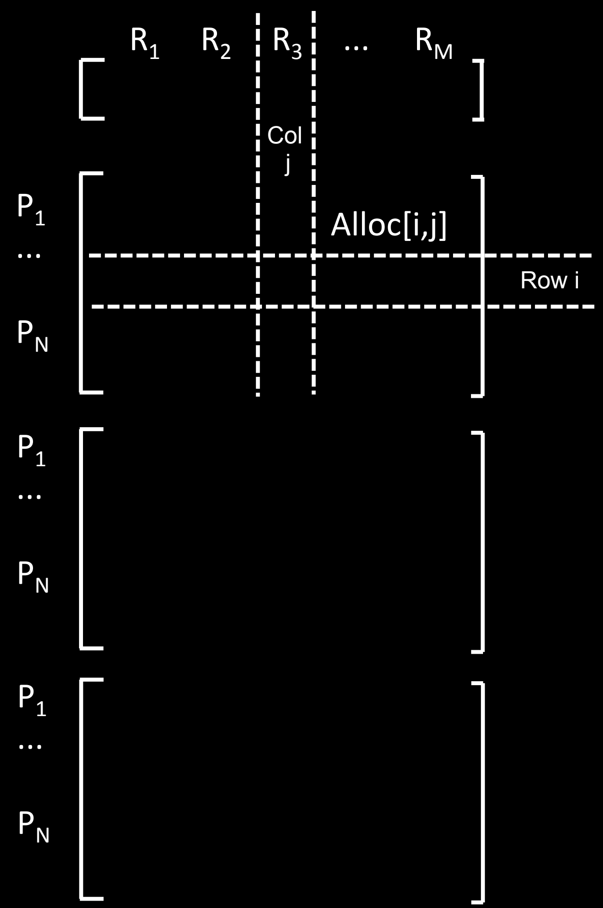
- *Generalizes deadlock avoidance to multiple resources*
  - *Determines whether the system is in a safe state*
- *Before granting a request, run Banker's Algorithm pretending as if request was granted*
  - *Does the worst-case analysis find such a hypothetical system is in a safe state?*
  - *If so, grant request.*
  - *If not, delay requestor, and wait for more resources to be freed.*

Available

Allocated

Max

Needed



$Alloc_i$ is shorthand for row *i* of matrix, i.e. the resources allocated to process *i*

Terminology

$$V1 = \begin{bmatrix} 1 \\ 7 \\ 3 \\ 2 \end{bmatrix} \qquad V2 = \begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

$$V3 = \begin{bmatrix} 0 \\ 10 \\ 2 \\ 1 \end{bmatrix}$$

$V2 \leq V1$

But,

$V3 \not\leq V1$

Banker's Algorithm

1. Let Work and Finish be vectors length m and n respectively.
   - Initialize Work = Available
   - Finish[i]=false for i=0,...,n-1

2. Find a process i such that both
   - Finish[i]==false, and
   - $Need_i \le Work$
   - If no such i exists, go to step 4.

3. $Work = Work + Alloc_i$
   - Finish[i] = true
   - Go to step 2.

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

Available

	A	B	C
	3	3	2

Alloc[i,j]

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

Max[i,j]

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

Work = Available
while (find unfinished P_i
where $Need_i \leq Work$)
Work = Work + $Alloc_i$
Finish[i] = true

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

Is *<P1, P3, P4, P2, P0>* a safe sequence?

Complexity

M resources

N processes

Algorithm must look at each process and consider all process sequences

Each search must look at all resources

$O(N^2 * M)$

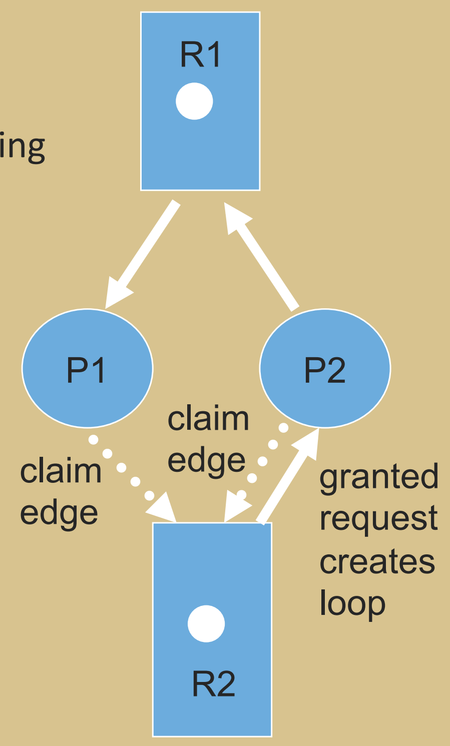
Algorithm:

Work = Available
while (find unfinished P_i
where $Need_i \leq Work$)
Work = Work + $Alloc_i$
Finish[i] = true

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

Banker's Algorithm

- Adapt to handle requests from processes
  - check to see if request would be safe
- Similar to resource allocation graph checking
  - See if new edge would generate a cycle



Solutions to Handling Deadlocks

1. Prevention by OS
   - provide methods to guarantee that at least 1 of the 4 necessary conditions for deadlock does not hold
2. Avoidance by OS
   - the OS is given advanced information about process requests for various resources, use the Banker's algorithm
   - Use resource allocation graphs
3. Detection and Recovery by OS
   - Analyze existing system resource allocation, and see if there is a sequence of releases that satisfies every process' needs.
4. Application-level solutions (OS Ignores and Pretends)
   - it's up to the application programmer to implement mechanisms that prevent, avoid, detect and deal with application-level deadlock

Deadlock Detection

- When/how often should the detection algorithm run?
  - Depends on how often deadlock is likely to occur
  - Depends on how quickly deadlock grows after it occurs, i.e. how many processes get pulled into deadlock and on what time scale
  - Could check at each resource request – this is costly
  - Could check periodically – but what is a good time interval?
  - Could check if CPU utilization suddenly drops – this might be an indication that there's deadlock, and processes are no longer executing, but what's a good threshold?
  - Could check if resource utilization exceeds some threshold, but what's a good threshold?

Deadlock Recovery

- After OS has detected which processes are deadlocked:
  - Terminate all processes - draconian
  - Terminate one process at a time until the deadlock cycle is eliminated
    - Check if there is still deadlock after each process is terminated. If not, then stop.
  - Preempt some processes – temporarily take away a resource from current owner and give it to another process but don't terminate process
    - e.g. give access to a laser printer – this is risky if you're in middle or printing a documents
  - Rollback some processes to a checkpoint – assuming that processes have saved their state at some checkpoint

Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Design and Analysis of Operating Systems CSCI 3753

Dr. David Knox
University of Colorado Boulder

These slides adapted from materials provided by the textbook authors.