COS 140: Foundations of Computer Science

Handling Deadlocks: Banker's Algorithm

Fall 2018

Computer
Science
Foundations

Homework, reminder

- Chapter 22 (online)
- Homework at the end of chapter
- Homework due 11/16 (later than usual!)
- Prelim II: Wednesday, 11/14

Computer Science Foundations

Operating systems as resource managers

- Example of resources
- Sharable vs non-sharable resources
- *Preemptible* vs non-preemptible resources
- Potential problem: deadlocks

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:

Computer Science Foundations

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card
  - P1: asks for CD-ROM and receives it

Computer Science Foundations

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card
  - P1: asks for CD-ROM and receives it
  - P2: needs CD-ROM and sound card

Computer Science Foundations

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card
  - P1: asks for CD-ROM and receives it
  - P2: needs CD-ROM and sound card
  - P2: asks for sound card and gets it

Copyright © 2002–2018 UMaine School of Computing and Information Science

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card
  - P1: asks for CD-ROM and receives it
  - P2: needs CD-ROM and sound card
  - P2: asks for sound card and gets it
  - P1: asks for sound card $\Rightarrow$ blocks

Computer Science Foundations

What are deadlocks?

- A deadlock occurs when each process in a set of processes is waiting for some event that only another process in the set can cause. [after Tannenbaum]
- Example:
  - P1: needs CD-ROM and sound card
  - P1: asks for CD-ROM and receives it
  - P2: needs CD-ROM and sound card
  - P2: asks for sound card and gets it
  - P1: asks for sound card $\Rightarrow$ blocks
  - P2: asks for CD-ROM $\Rightarrow$ blocks

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Conditions for deadlocks

Mutual exclusion: Resource is either available or assigned to at most one process

Copyright © 2002–2018 UMaine School of Computing and Information Science

Conditions for deadlocks

Mutual exclusion: Resource is either available or assigned to at most one process

Hold-and-wait: Process can hold one resource and then ask for others

Copyright © 2002–2018 UMaine School of Computing and Information Science

Conditions for deadlocks

Mutual exclusion: Resource is either available or assigned to at most one process

Hold-and-wait: Process can hold one resource and then ask for others

No preemption: Can't take a resource away from a process once assigned

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Conditions for deadlocks

Mutual exclusion: Resource is either available or assigned to at most one process

Hold-and-wait: Process can hold one resource and then ask for others

No preemption: Can't take a resource away from a process once assigned

Circular wait: $\geq$ 2 processes in circle in which each is waiting for resource held by next in circle

Copyright © 2002–2018 UMaine School of Computing and Information Science

Digression: Directed graphs

- Many areas of CS require us to think of objects and relationships between them; e.g., paths between locations, data dependencies, constraints in logic puzzles

Computer Science Foundations

Digression: Directed graphs

- Many areas of CS require us to think of objects and relationships between them; e.g., paths between locations, data dependencies, constraints in logic puzzles
- Can represent this formally as a *graph*:
  - *Vertices* (or nodes) represent the objects
  - *Edges* (or arcs, or links) represent the relationships

Copyright © 2002–2018 UMaine School of Computing and Information Science

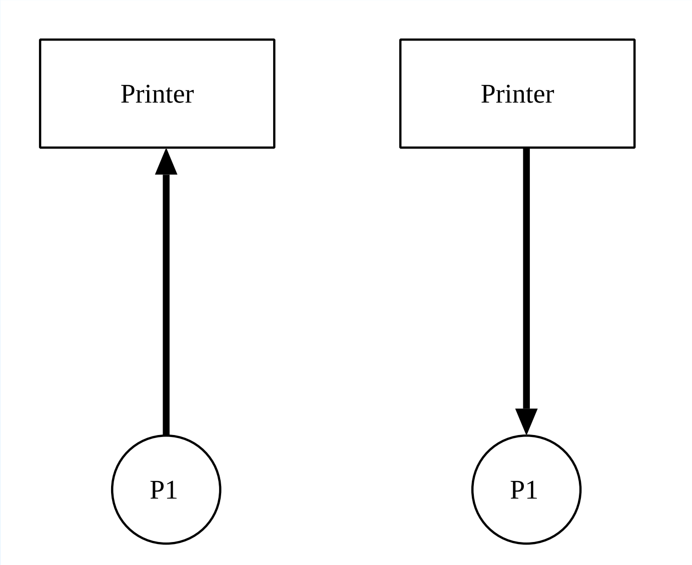
Digression: Directed graphs

- Many areas of CS require us to think of objects and relationships between them; e.g., paths between locations, data dependencies, constraints in logic puzzles
- Can represent this formally as a *graph*:
  - *Vertices* (or nodes) represent the objects
  - *Edges* (or arcs, or links) represent the relationships
- Sometimes, relationship is directional
  - Think “one-way streets”
  - Now the edges have direction, and the graph is called a *directed graph* or *digraph*

Copyright © 2002–2018 UMaine School of Computing and Information Science

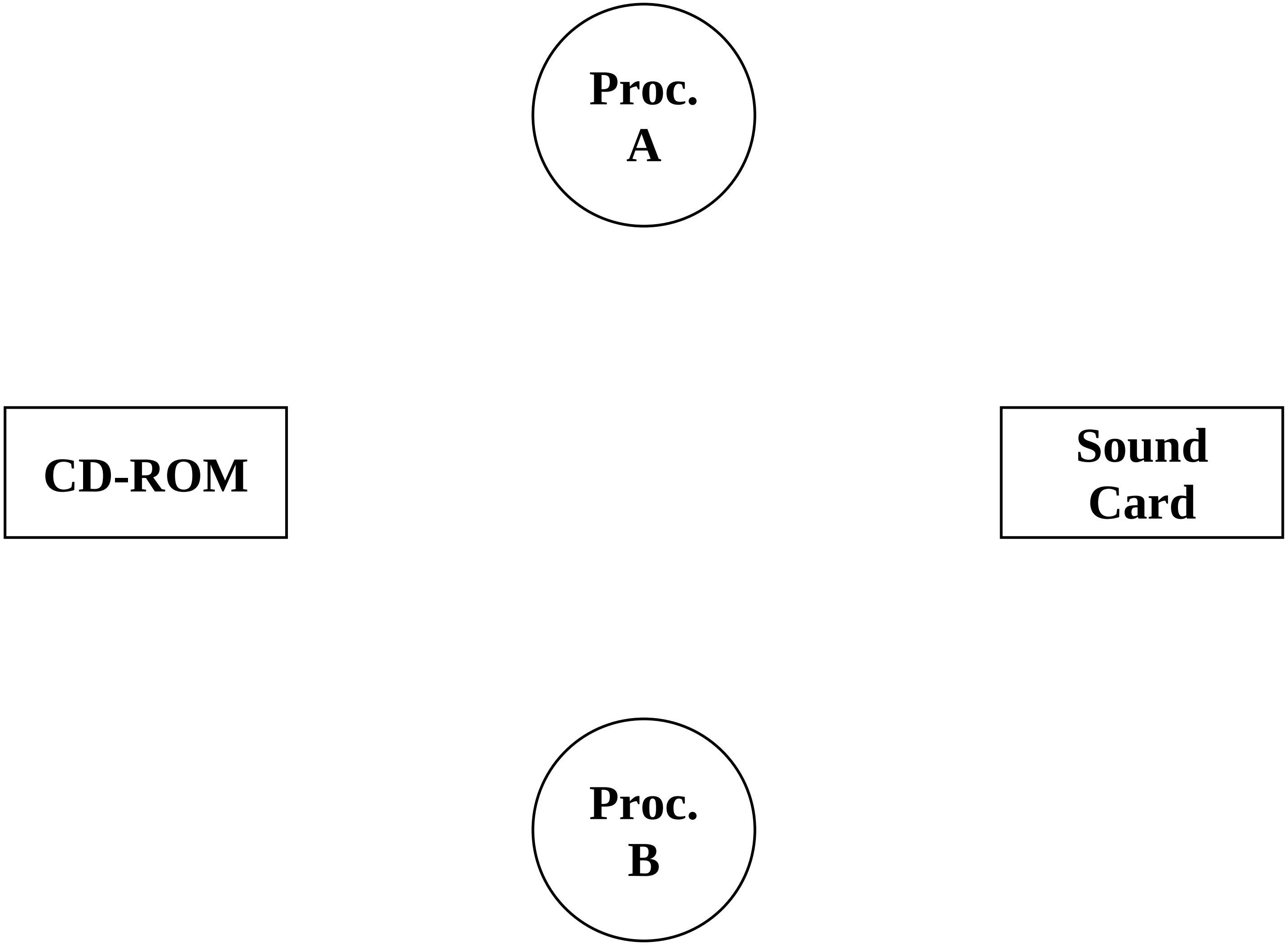
Modeling deadlocks as digraphs

- Circles: processes
- Squares: resources
- Link from process $\longrightarrow$ resource: process requests resource
- Link from resource $\longrightarrow$ process: process has control of resource



Computer Science Foundations

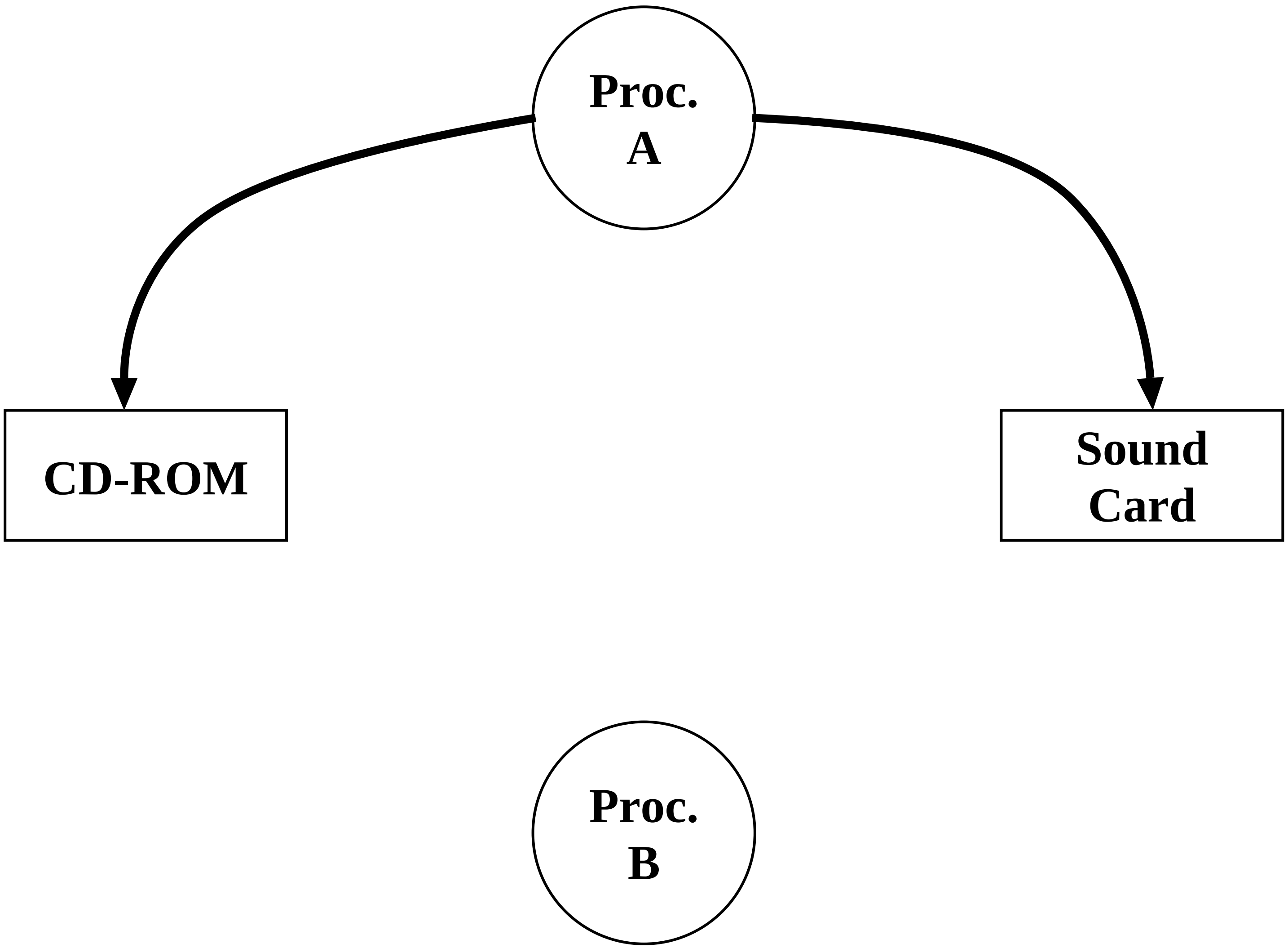
Modeling deadlocks as digraphs



Computer Science Foundations

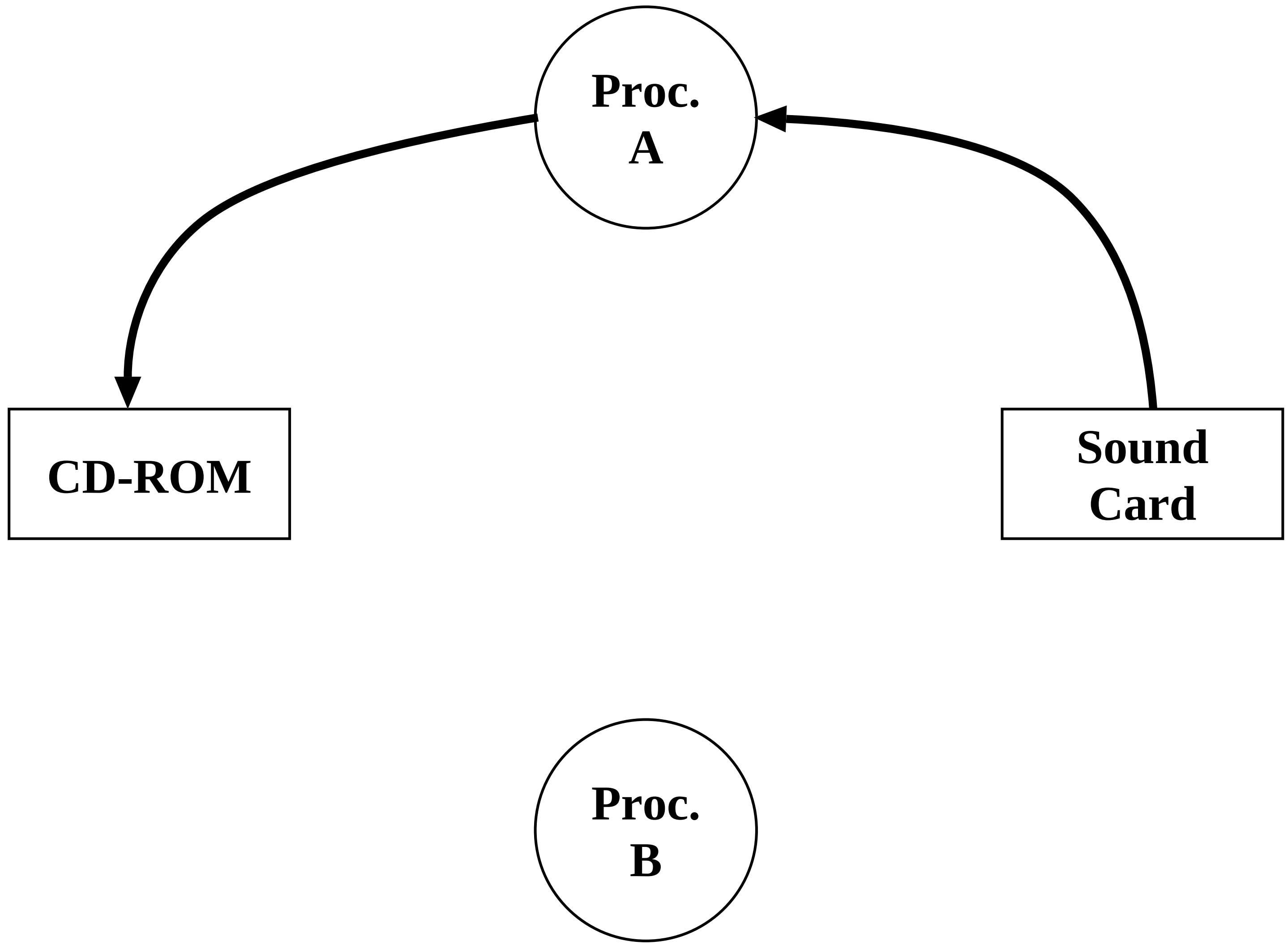
Copyright © 2002–2018 UMaine School of Computing and Information Science

Modeling deadlocks as digraphs



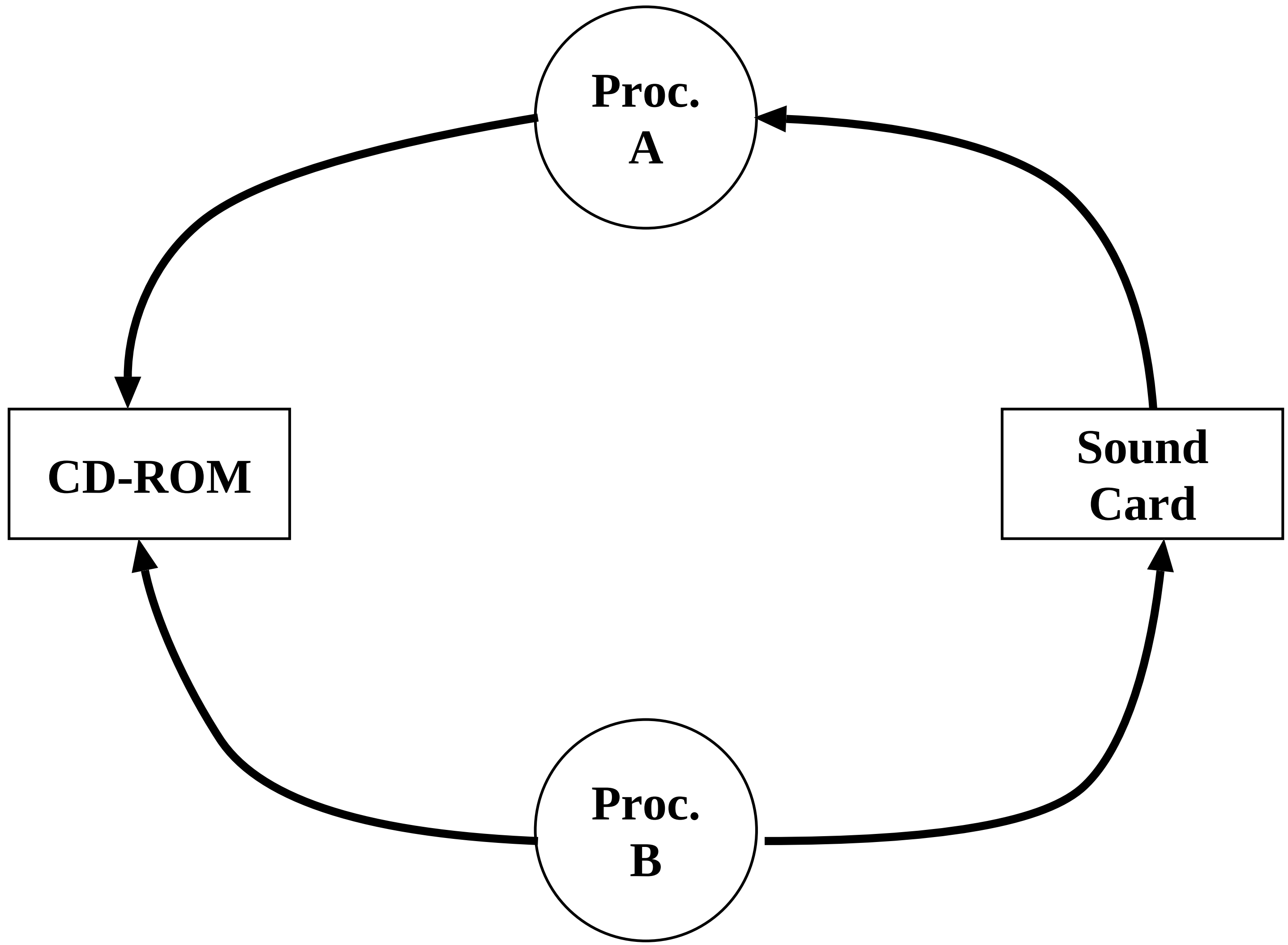
Computer Science Foundations

Modeling deadlocks as digraphs



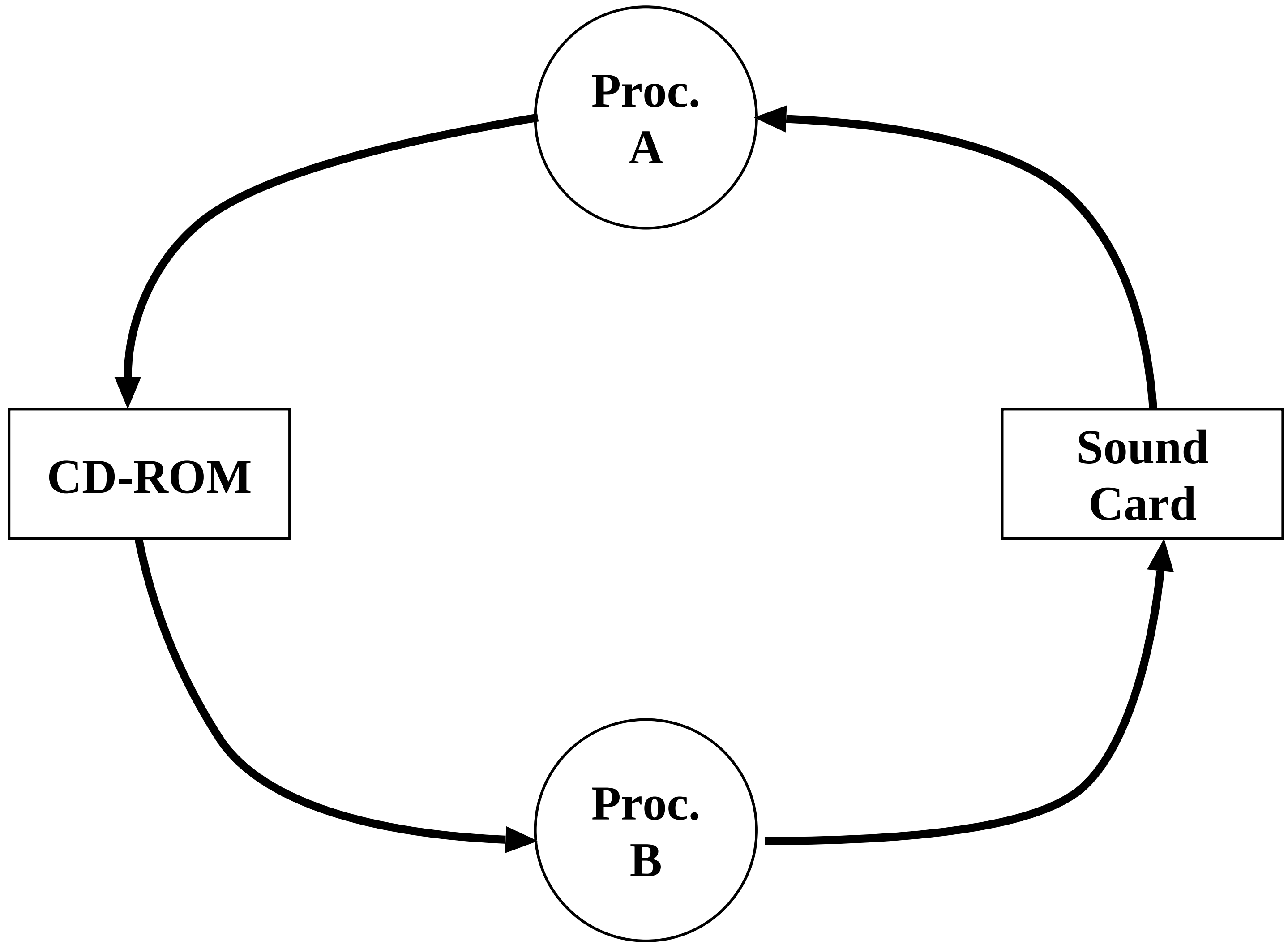
Computer Science Foundations

Modeling deadlocks as digraphs



Computer Science Foundations

Modeling deadlocks as digraphs



Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

What do we do about deadlocks?

- Ignore them
- Detect them and (try to) recover
- Prevent them altogether
- Predict and avoid them

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Ignoring deadlocks: The Ostrich Algorithm

- Sounds stupid, but...
- Consider:
  - How often will a deadlock happen?
  - How severe will it be if it does happen?
  - How hard would it be to avoid/prevent/detect?

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Deadlock detection/recovery

- Detection:
  - Monitor resource allocation using (e.g.) a digraph
  - If detect a cycle $\Rightarrow$ deadlock has occurred
- Recovery:
  - Kill one of the processes
  - If that doesn't work: kill another, etc.
- Another alternative: just look for processes that have been idle for a long time and kill them
- May be okay when aborting and restarting is okay (e.g., batch jobs)

Computer Science Foundations

Deadlock prevention

- Set things up so that deadlocks cannot occur at all
- Done by attacking one of the deadlock conditions
- Attacking mutual exclusion condition:
  - Don't let non-sharable resources be assigned to anyone
  - E.g., spooling

Computer Science Foundations

Deadlock prevention

- Attacking hold-and-wait condition:
  - Process can't request a resource if holding any
  - One way: processes request all resources up front
  - Problem: may not know ahead of time what you need!
  - Problem: hold resources too long in general
  - Another approach: release all you're holding momentarily to request another
- Attacking no preemption condition: not realistic

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Deadlock prevention

- Attacking the circular wait condition:
  - Stupid way: processes can only hold a single resource at a time

Computer Science Foundations

Deadlock prevention

- Attacking the circular wait condition:
  - Stupid way: processes can only hold a single resource at a time
  - Better way:
    - Number the resources
    - Process can request whatever it wants, whenever it wants...as long as the requests are in numerical order

Computer Science Foundations

Deadlock prevention

- Attacking the circular wait condition (cont'd):
  - Resource allocation graph can't have cycles in this scheme – why not?

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Deadlock prevention

- Attacking the circular wait condition (cont'd):
  - Resource allocation graph can't have cycles in this scheme – why not?
    - Consider the case where process A holds resource i and B holds j – deadlock only possible if A requests j and B requests i

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Deadlock prevention

- Attacking the circular wait condition (cont'd):
  - Resource allocation graph can't have cycles in this scheme – why not?
    - Consider the case where process A holds resource i and B holds j – deadlock only possible if A requests j and B requests i
    - If $i > j$, then A can't request j

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Deadlock prevention

- Attacking the circular wait condition (cont'd):
  - Resource allocation graph can't have cycles in this scheme – why not?
    - Consider the case where process A holds resource i and B holds j – deadlock only possible if A requests j and B requests i
    - If $i > j$, then A can't request j
    - If $j > i$, then B can't request i

Computer Science Foundations

Deadlock prevention

- Attacking the circular wait condition (cont'd):
  - Resource allocation graph can't have cycles in this scheme – why not?
    - Consider the case where process A holds resource i and B holds j – deadlock only possible if A requests j and B requests i
    - If $i > j$, then A can't request j
    - If $j > i$, then B can't request i
  - Problem – may not be able to find an ordering that satisfies everyone!

Computer Science Foundations

Deadlock avoidance

- Idea: predict when some action $\longrightarrow$ deadlock, avoid it
- Dijkstra's Banker's Algorithm (single resource version)
  - Modeled on the way a banker might deal with lines of credit to customers
  - Deadlock if there is no way to guarantee that all customers can borrow up to their maximum resource limit at some point in time

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Dijkstra's Banker's Algorithm

- Safety:

  A state is *safe* if some sequence of other possible states exists that allows all customers (processes) to get up to their maximum resource limit at some time

- Keep track of maxmimum and current allocation for each customer
- Start in a safe state
- When process requests additional amount of resource, make sure that next state will also be safe
- If so, allow request, else disallow it

Computer Science Foundations

Banker's Algorithm Example

- Initial state:

Process	Current	Maximum
A	0	7
B	0	3
C	0	2
D	0	4

Remaining: 8

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	2	3
C	0	2
D	2	4

Remaining: 1

- Safe or not?

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	2	3
C	0	2
D	2	4

Remaining: 1

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A

Computer Science Foundations

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	3	3
C	0	2
D	2	4

Remaining: 0

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	–	–
C	0	2
D	2	4

Remaining: 3

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	–	–
C	0	2
D	4	4

Remaining: 1

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	–	–
C	0	2
D	–	–

Remaining: 5

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	–	–
C	2	2
D	–	–

Remaining: 3

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	3	7
B	–	–
C	–	–
D	–	–

Remaining: 5

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\longrightarrow$ D $\longrightarrow$ C $\longrightarrow$ A

Computer Science Foundations

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	7	7
B	–	–
C	–	–
D	–	–

Remaining: 1

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A

Computer Science Foundations

Banker's Algorithm Example 2

- Initial state:

Process	Current	Maximum
A	–	–
B	–	–
C	–	–
D	–	–

Remaining: 8

- Safe or not?
  - Safe
  - Possible sequence of processes running to completion: B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 3

Proc	Curr	Max
A	3	7
B	2	3
C	0	2
D	2	4

Remaining: 1

$\overset{B \text{ wants } 1}{\Longrightarrow}$

Proc	Curr	Max
A	3	7
B	3	3
C	0	2
D	2	4

Remaining: 0

- Allow the request?

Computer Science Foundations

Banker's Algorithm Example 3

Proc	Curr	Max
A	3	7
B	2	3
C	0	2
D	2	4

Remaining: 1

$\stackrel{B \text{ wants } 1}{\Longrightarrow}$

Proc	Curr	Max
A	3	7
B	3	3
C	0	2
D	2	4

Remaining: 0

- Allow the request?
  - Yes.
  - Possible sequence of processes running to completion: B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A

Computer Science Foundations

Banker's Algorithm Example 4

Proc	Curr	Max
A	6	7
B	0	3
C	0	2
D	1	4

Remaining: 1

$\xrightarrow{A \text{ wants } 1}$

Proc	Curr	Max
A	7	7
B	0	3
C	0	2
D	1	4

Remaining: 0

- Allow the request?

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 4

Proc	Curr	Max
A	6	7
B	0	3
C	0	2
D	1	4

Remaining: 1

$\xrightarrow{A \text{ wants } 1}$

Proc	Curr	Max
A	7	7
B	0	3
C	0	2
D	1	4

Remaining: 0

- Allow the request?
  - Yes.
  - Possible sequence of processes running to completion: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D

Computer Science Foundations

Banker's Algorithm Example 5

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	1	4

Remaining: 2

$\xrightarrow{D \text{ wants } 1}$

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	2	4

Remaining: 1

- Allow the request?

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm Example 5

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	1	4

Remaining: 2

$\xrightarrow{D \text{ wants } 1}$

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	2	4

Remaining: 1

- Allow the request?
  - NO!
  - No sequence possible where they all can finish

Computer Science Foundations

Banker's Algorithm: Critique

- Is it too strong?
- After all – no guarantee that in the previous example:

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	1	4

Remaining: 2

$\overset{D \text{ wants } 1}{\Longrightarrow}$

Proc	Curr	Max
A	4	7
B	1	3
C	0	2
D	2	4

Remaining: 1

D might not give back 1 immediately, moving back into safe state

- But we're interested in *guarantee* that there will be no deadlock, so this is what we need.

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Banker's Algorithm: Critique

- Scales to multiple processes/resources
- Problems:
  - Need to know maximum resources needed per process – often (usually?) impossible for multiprocess system
  - Number of processes constantly changes
  - Resources can disappear
- But: Really no better general-purpose algorithm exists for this

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science

Summary

- Handling deadlocks is difficult
- No best general solution
- How you choose to handle it depends on your situation: *trade-offs*

Computer Science Foundations

Copyright © 2002–2018 UMaine School of Computing and Information Science