ECS150 Discussion Section

Sophie Engle

(*February 11/13 2004*)

Announcements

- Website updated!
  - Added information on compiling the Minix kernel
  - Added more hints & tips for using Minix
  - Added more common compile errors
  - Updated grades
- Will have to know how to compile the Minix kernel for next programming assignment!

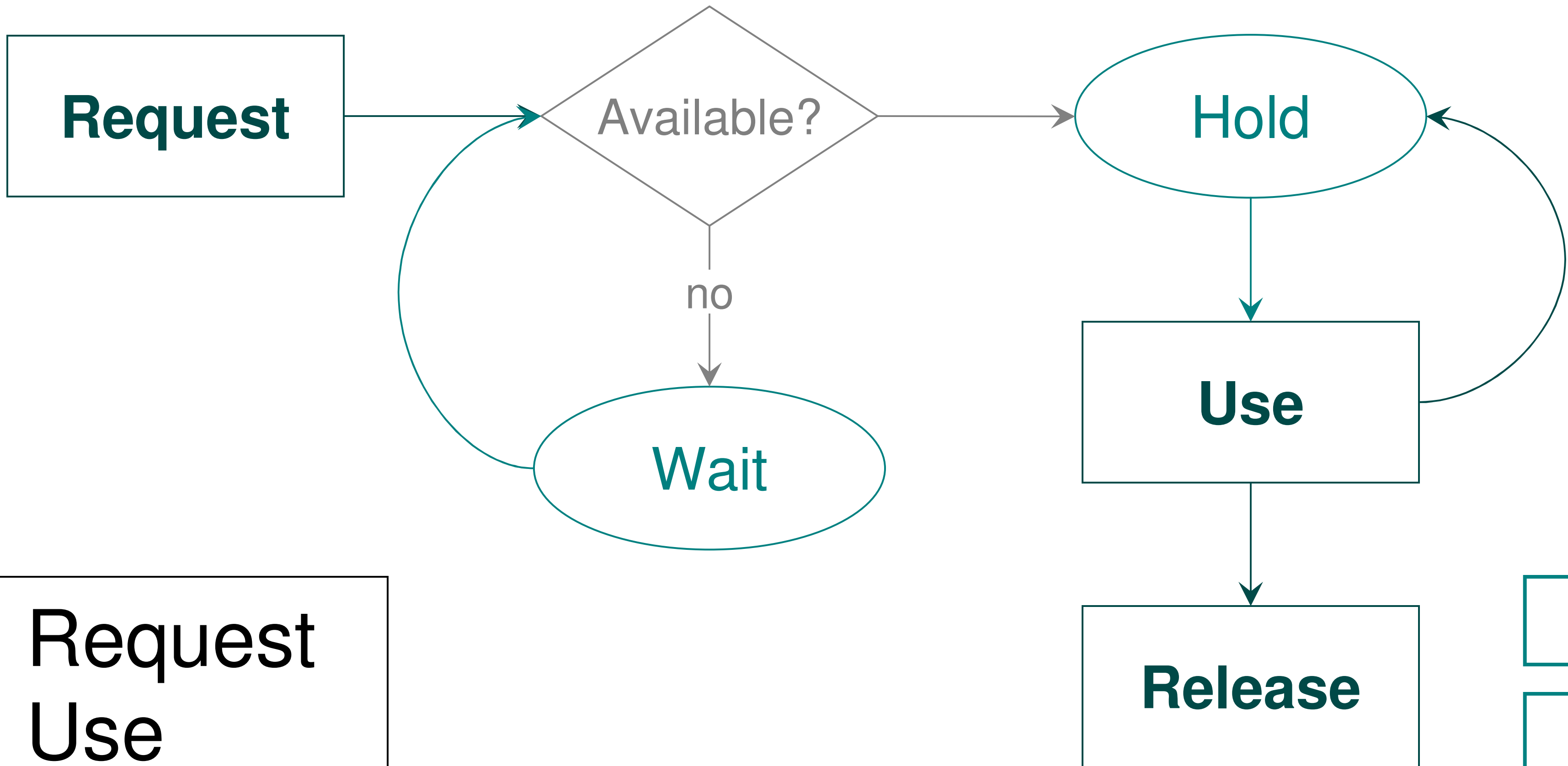
Deadlocks

- Resources
  - Tanenbaum p75 – 77
  - Tanenbaum p166 – 179

Resources

- **Resource**
  - Anything that can only be used by a single process at any instant
- Types
  - **Preemptable**
    - Resource can be taken away
    - Example: Memory
  - **Nonpreemptable***
    - Resource can NOT be taken away
    - Example: Printer

Nonpreemptable Resources



1. Request
2. Use
3. Release

Deadlock

- **Definition**:
  - A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- **Example**:
  - Process 1 is holding resource A, needs B
  - Process 2 is holding resource B, needs A

Dining Philosophers

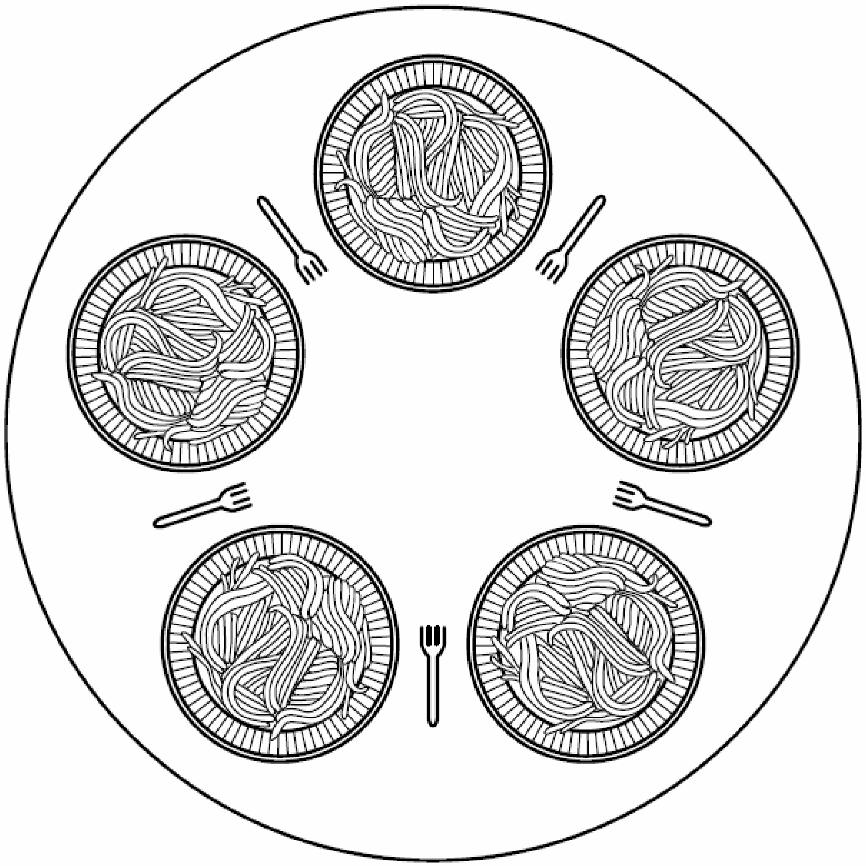


Figure 2-16. Lunch time in the Philosophy Department.

- Must have 2 forks to eat
- Everyone picks up left fork at same time and waits for right fork
- Nobody is able to eat, deadlock occurs

Deadlock Conditions

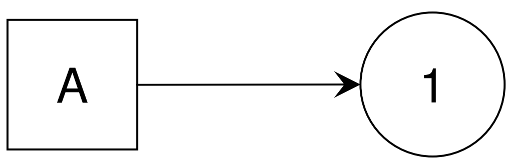
- **Necessary Conditions**:
  - Mutual exclusion condition
  - Hold and wait condition
  - No preemption condition
  - Circular wait condition

Deadlock Conditions

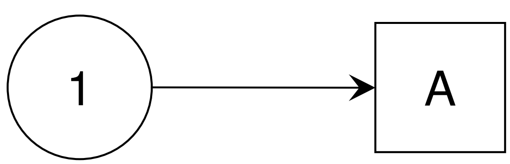
- **Necessary Conditions**:
  - Mutual exclusion condition
    - Resource assigned to exactly one process (or none)
  - Hold and wait condition
    - Can request resource at any time
  - No preemption condition
    - Granted resources can not be forcibly taken away
  - Circular wait condition
    - Must be cycle of processes waiting on each other

Resource Graph

- Nodes:
  - Processes: Circle
  - Resources: Square
- Arcs:
  - Resource A → Process 1: P1 holds A



  - Process 1 → Resource A: P1 waiting for A



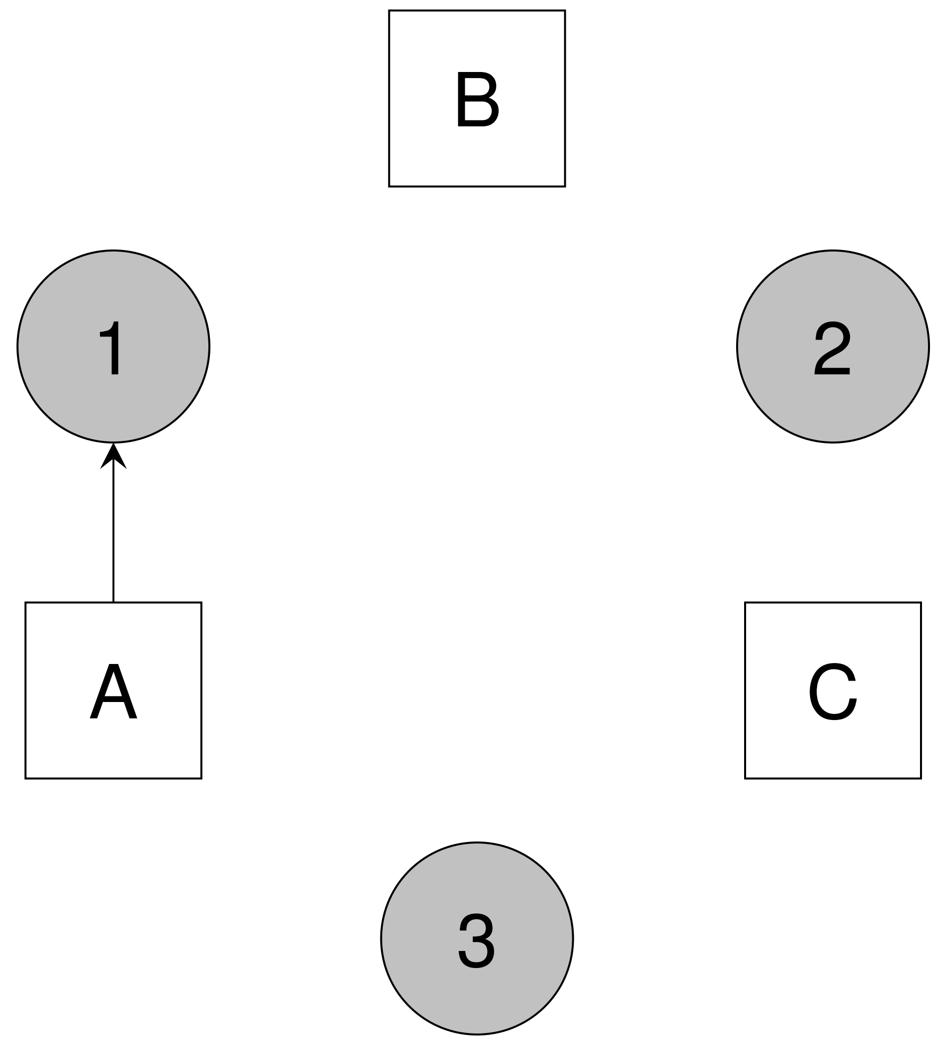
Deadlock Example

Process #	Required Resources
Process 1	A and B
Process 2	B and C
Process 3	C and A

Deadlock Example

Events:

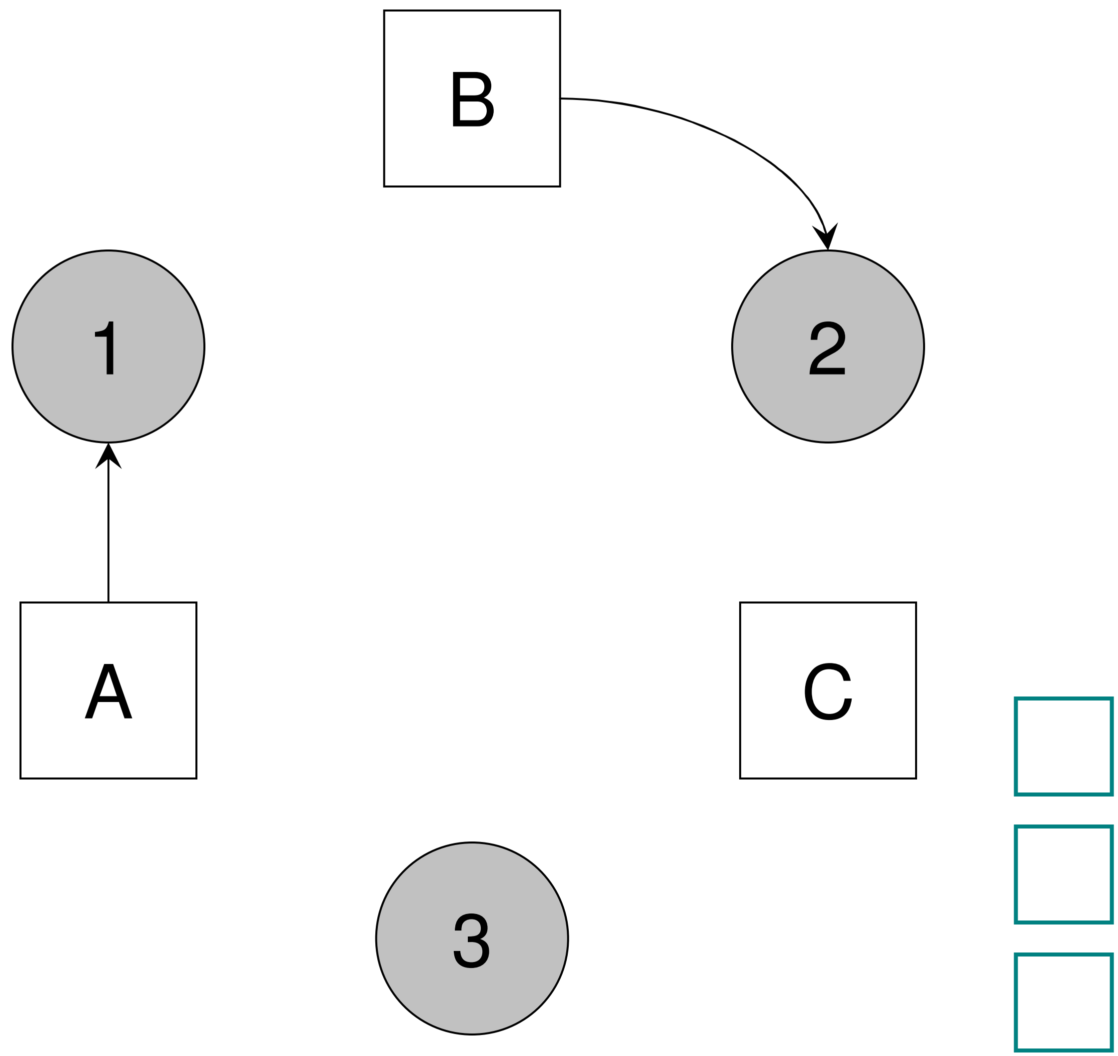
P1 requests A



Deadlock Example

Events:

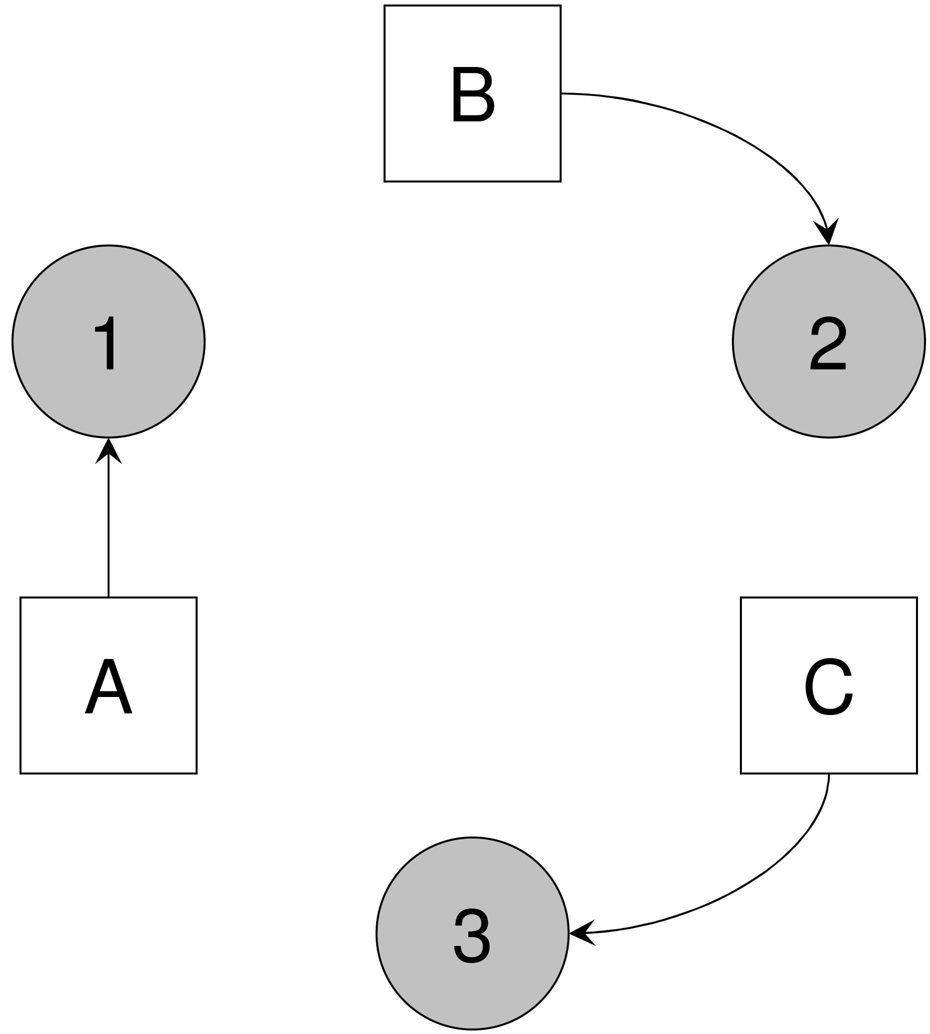
P1 requests A
P2 requests B



Deadlock Example

Events:

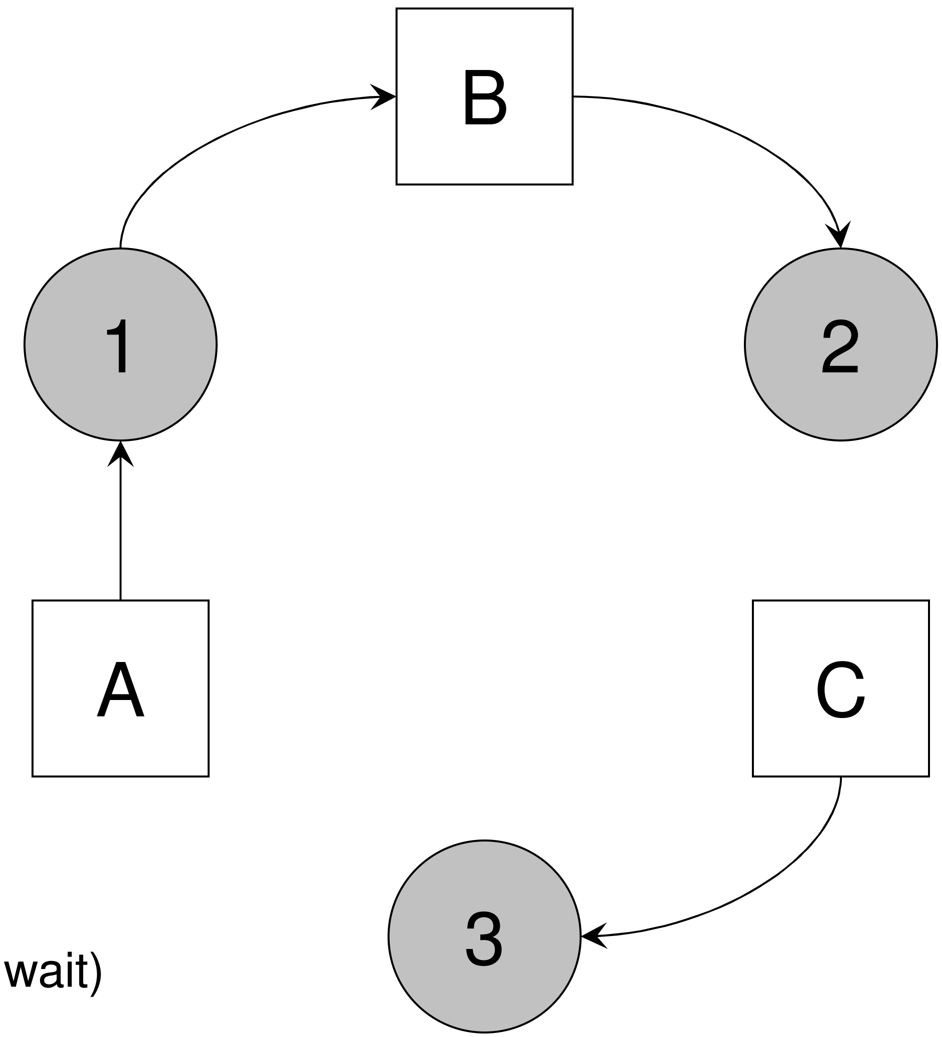
P1 requests A
P2 requests B
P3 requests C



Deadlock Example

Events:

P1 requests A
P2 requests B
P3 requests C
P1 requests B

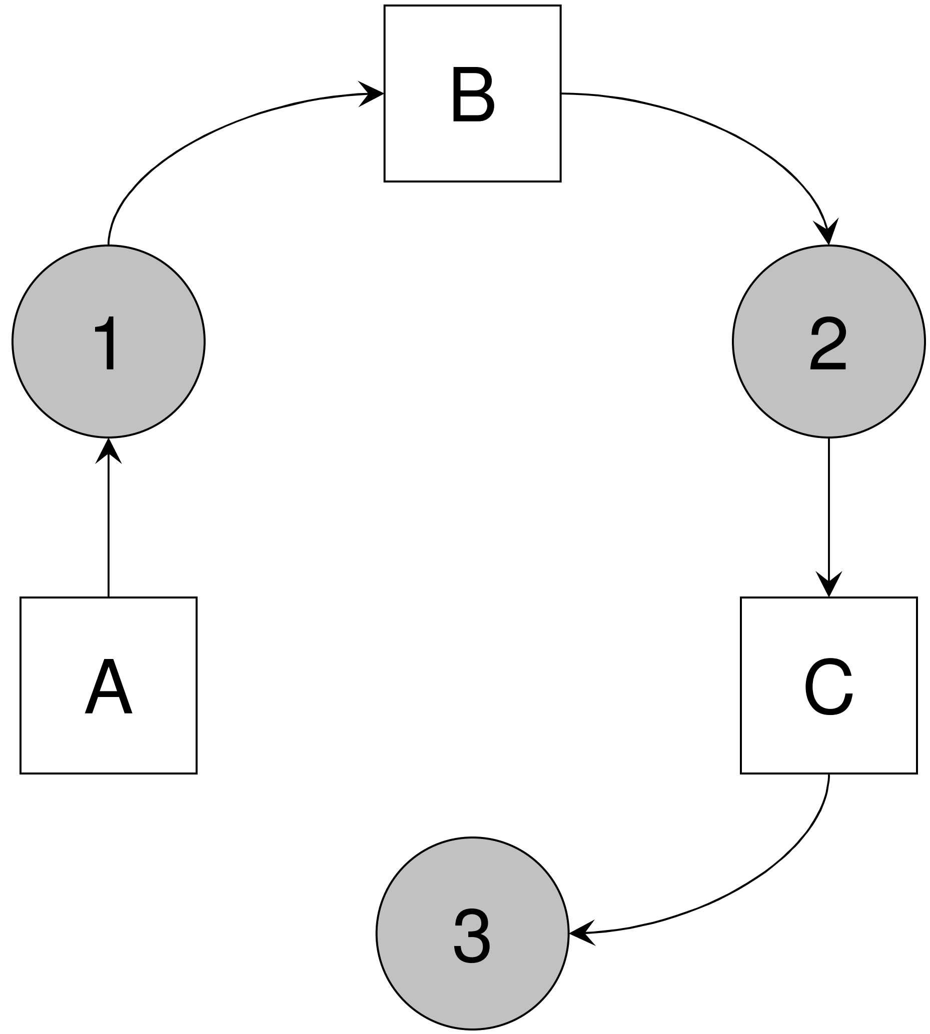


(notice example of hold and wait)

Deadlock Example

Events:

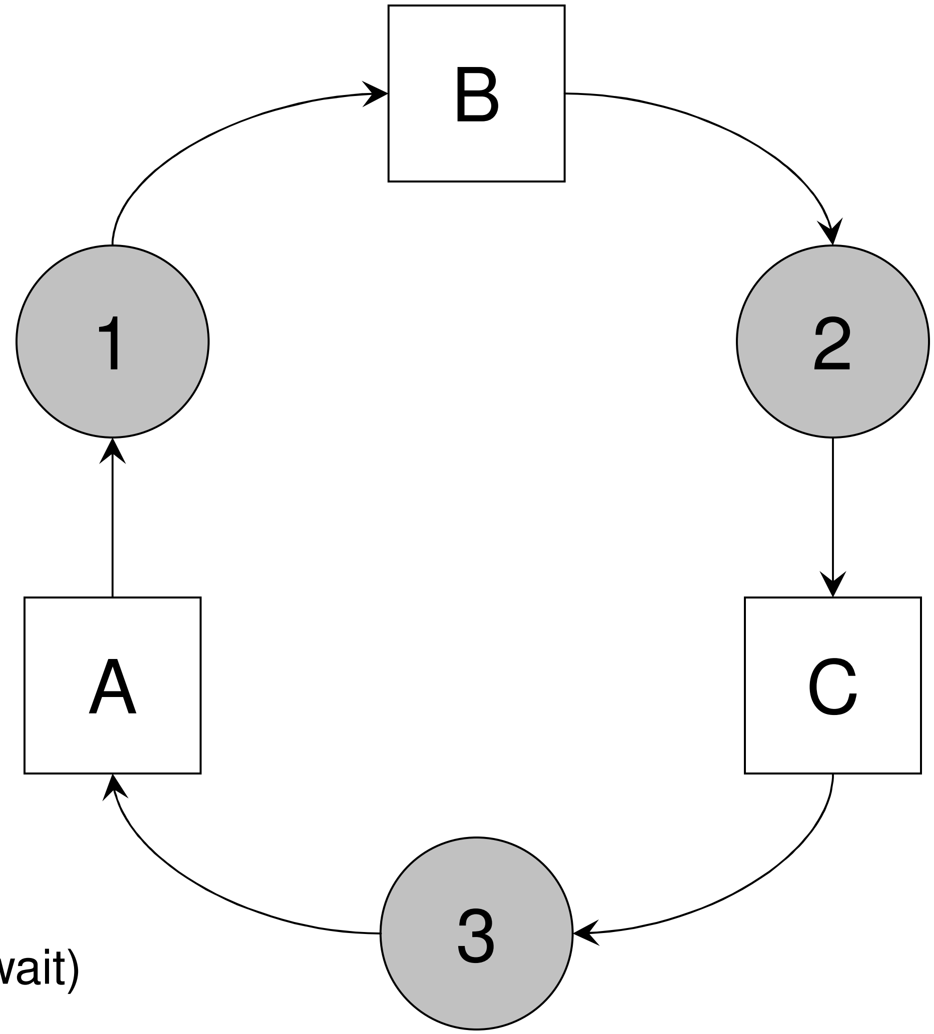
P1 requests A
P2 requests B
P3 requests C
P1 requests B
P2 requests C



Deadlock Example

Events:

P1 requests A
P2 requests B
P3 requests C
P1 requests B
P2 requests C
P3 requests A



(notice example of circular wait)

Handling Deadlocks

- Possible responses:
  - Ignore
  - Detect (and recover)
  - Avoid
  - Prevent

Handling Deadlocks

- Possible responses:
  - Ignore
    - Are deadlocks such a bad problem?
  - Detect (and recover)
    - Watch for deadlocks, fix when happens
  - Avoid
    - Make good decisions!
  - Prevent
    - Eliminate one condition required for deadlocks

Ignore Deadlocks

- Why ignore deadlocks?
  - How often will deadlocks occur?
  - How often will system crash anyway?
  - Is deadlock detection, prevention, or avoidance really cost effective?
- Ostrich Algorithm
  - Ignore problem of deadlocks completely

Detection and Recovery

Method	Detection	Recovery
1	Keep resource graph, check for cycles	Kill a process in the cycle until cycle is broken
2	Check how long process has been continuously blocked	Kill process that has been continuously blocked for long time

Avoidance

- Decide if resource should be granted
  - Want to keep system in a “safe” state
  - Requires certain information in advance
  - Great in theory, but often impractical

- Banker’s Algorithm for Multiple Resources
  - Often discussed avoidance algorithm
  - Rarely used in practice

Banker's Algorithm

- Tracks current **state** of system
- Prevents **state** from becoming **unsafe**
  - A state is **safe** if there exists some sequence that allows every process to run to completion

Banker's Algorithm

- State tracked by:
  - 2 Process by Resource Matrices
    - Resources assigned
    - Resources still needed
  - 3 Resource Vectors
    - (E)xisting resources
    - (P)ossessed resources
    - (A)vailable resources

Banker's Algorithm

To check if initial state safe:

1. Find process whose needs are less than the available resources
   - If none exist, then state not safe (deadlock possible)
2. Assume process completes, and add its resources as available
3. Repeat steps 1 and 2 until all processes are terminated
   - If possible, then initial state safe

Banker's Algorithm

	assigned					still needed					existing	possessed	available
	A	B	C	D	E	A	B	C	D	E			
tape drives	3	0	1	1	0	1	0	3	0	2	6	5	1
plotters	0	1	1	1	0	1	1	1	0	1	3	3	0
printers	1	0	1	0	0	0	1	0	1	1	4	2	2
cd-roms	1	0	0	1	0	0	2	0	0	0	2	2	0

Banker's Algorithm

Process = D

	assigned					still needed					existing	possessed	available
	A	B	C	D	E	A	B	C	D	E			
tape drives	3	0	1	1	0	1	0	3	0	2	6	5	1
plotters	0	1	1	1	0	1	1	1	0	1	3	3	0
printers	1	0	1	0	0	0	1	0	1	1	4	2	2
cd-roms	1	0	0	1	0	0	2	0	0	0	2	2	0

Banker's Algorithm

Process = A

	assigned					still needed					existing	possessed	available
	A	B	C	~~D~~	E	A	B	C	~~D~~	E			
tape drives	3	0	1	~~1~~	0	1	0	3	~~0~~	2	6	4	2
plotters	0	1	1	~~1~~	0	1	1	1	~~0~~	1	3	2	1
printers	1	0	1	~~0~~	0	0	1	0	~~1~~	1	4	2	2
cd-roms	1	0	0	~~1~~	0	0	2	0	~~0~~	0	2	1	1

Banker's Algorithm

Eventually, all will be able to run. Initial state = safe

	assigned					still needed					existing	possessed	available
	~~A~~	B	C	~~D~~	E	~~A~~	B	C	~~D~~	E			
tape drives	~~3~~	0	1	~~1~~	0	~~1~~	0	3	~~0~~	2	6	2	4
plotters	~~0~~	1	1	~~1~~	0	~~1~~	1	1	~~0~~	1	3	2	1
printers	~~1~~	0	1	~~0~~	0	~~0~~	1	0	~~1~~	1	4	2	2
cd-roms	~~1~~	0	0	~~1~~	0	~~0~~	2	0	~~0~~	0	2	0	3

Deadlock Prevention

- Impose restrictions that make deadlocks structurally impossible
  - Prevent at least one of four conditions required for deadlock to occur
- How prevent deadlock?
  - Prevent mutual exclusion?
  - Prevent hold and wait condition?
  - Prevent no preemption condition?
  - Prevent circular wait condition?

Deadlock Conditions

- **Necessary Conditions**:
  - Mutual exclusion condition
    - Resource assigned to exactly one process (or none)
  - Hold and wait condition
    - Can request resource at any time
  - No preemption condition
    - Granted resources can not be forcibly taken away
  - Circular wait condition
    - Must be cycle of processes waiting on each other