Deadlock (1)

Dave Eckhardt
de0u@andrew.cmu.edu

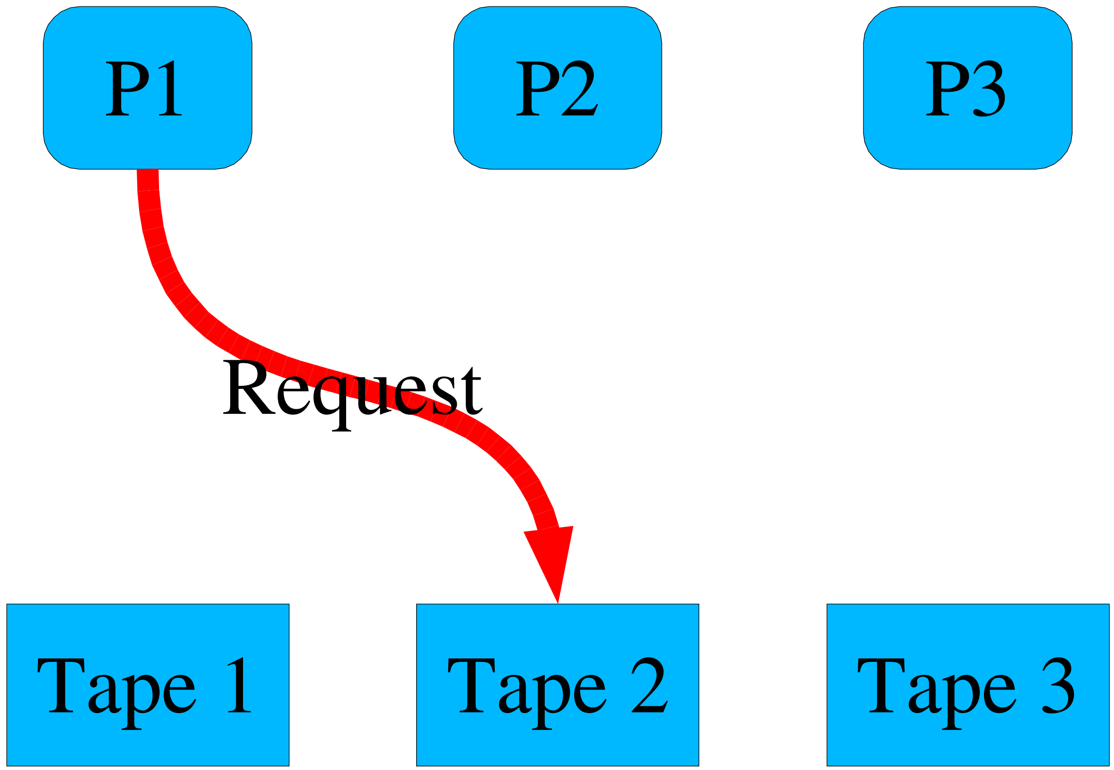
Synchronization

- P2 – You should ***really*** have
  - Made each syscall once
    - Except maybe minclone()
  - A ***detailed*** design for {thr,mutex,cond}_*()
- Readings (posted on course web)
  - Deadlock: 7.4.3, 7.5.3, Chapter 8
  - Scheduling: Chapter 6
  - Memory: Chapter 9, Chapter 10

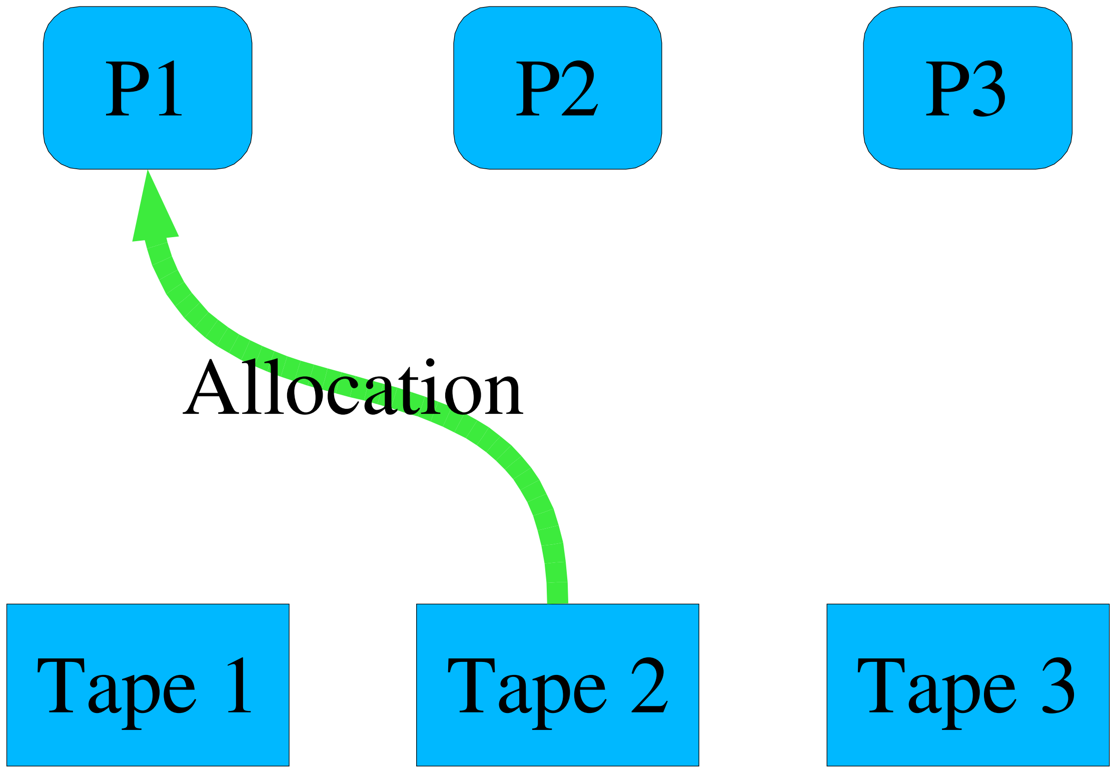
Outline

- Process resource graph
- What is deadlock?
- Deadlock ***prevention***
- Next time
  - Deadlock ***avoidance***
  - Deadlock ***recovery***

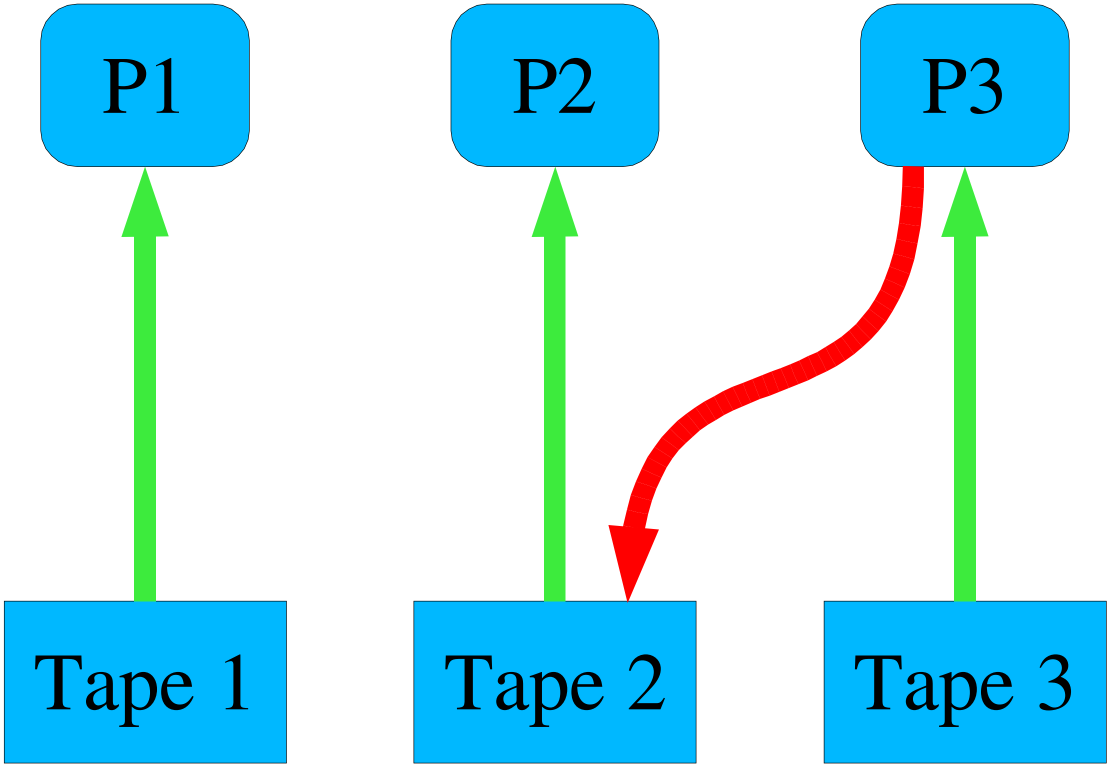
Process/Resource graph



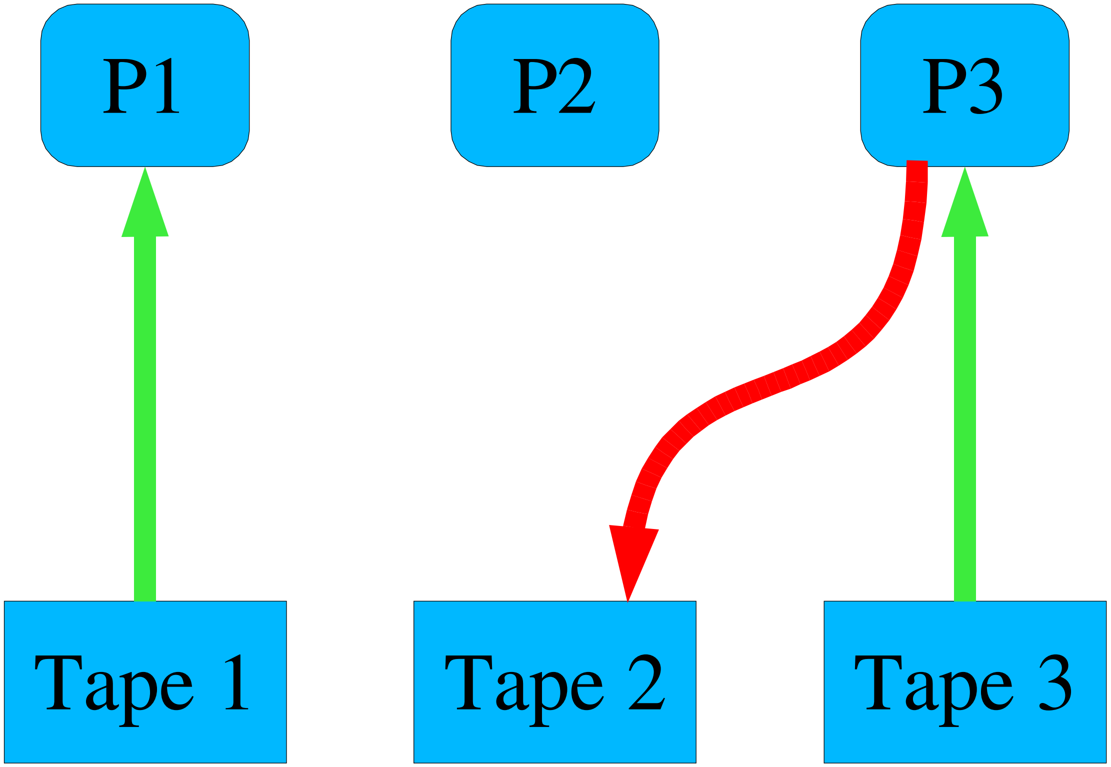
Process/Resource graph



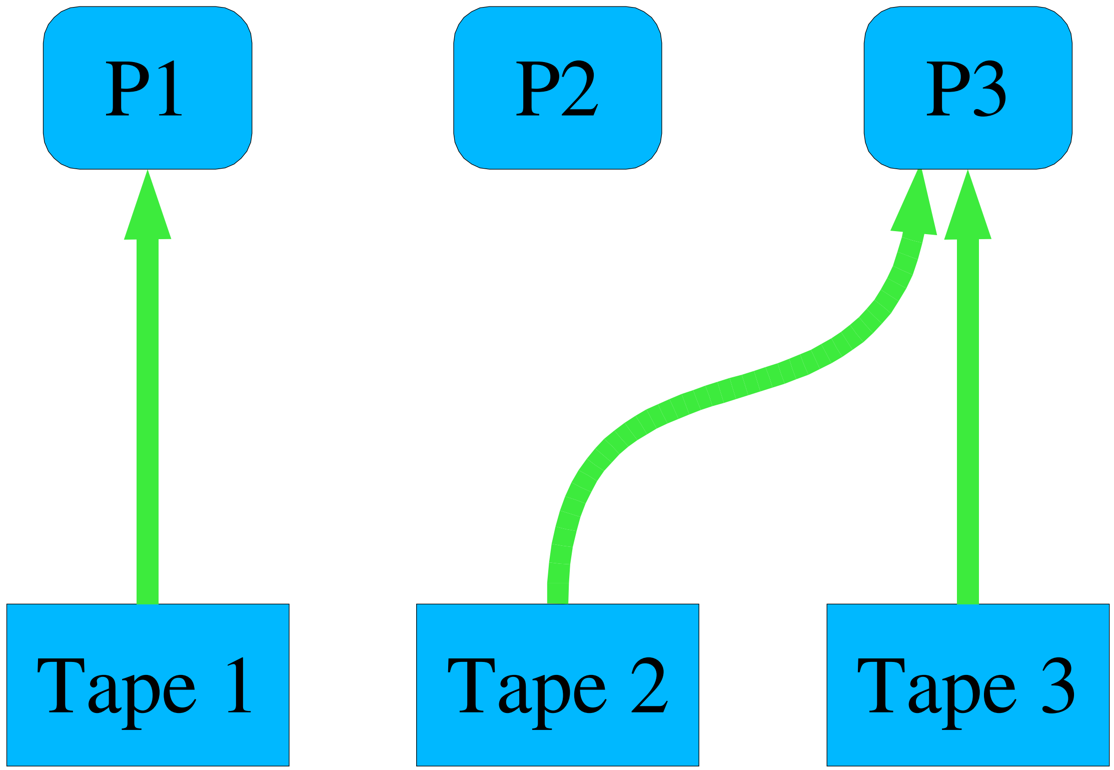
Waiting



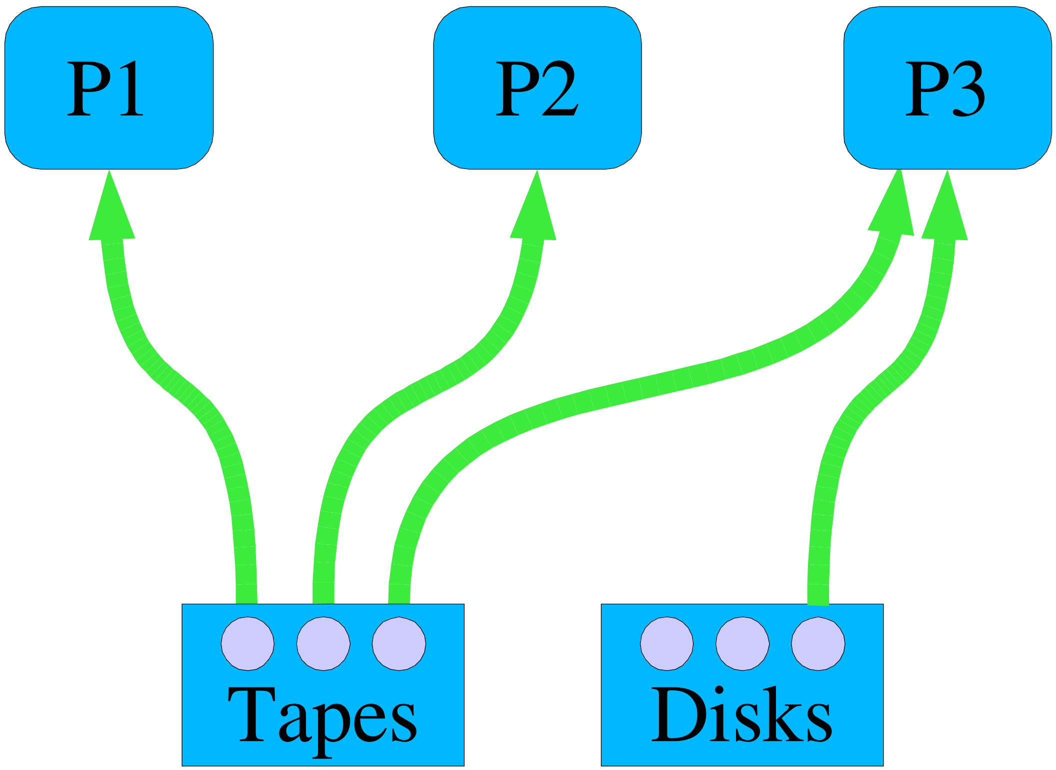
Release



Reallocation



Multi-instance Resources



Definition of Deadlock

- Deadlock
  - Set of N processes
  - Each waiting for an event
    - ...which can be caused ***only by another waiting process***
- Every process will wait forever

Deadlock Examples

- Simplest form
  - Process 1 owns printer, wants tape drive
  - Process 2 owns tape drive, wants printer
- Less-obvious
  - Three tape drives
  - Three processes
    - Each has one tape drive
    - Each wants “just” one more
  - Can't blame anybody, but problem is still there

Deadlock Requirements

- Mutual Exclusion
- Hold & Wait
- No Preemption
- Circular Wait

Mutual Exclusion

- Resources aren't “thread-safe” (“reentrant”)
- Must be allocated to one process/thread at a time
- Can't be shared
  - Programmable Interrupt Timer
    - Can't have a different reload value for each process

Hold & Wait

- Process holds resources while waiting for more

```
mutex_lock(&m1);
mutex_lock(&m2);
mutex_lock(&m3);
```

- ***Typical*** locking behavior

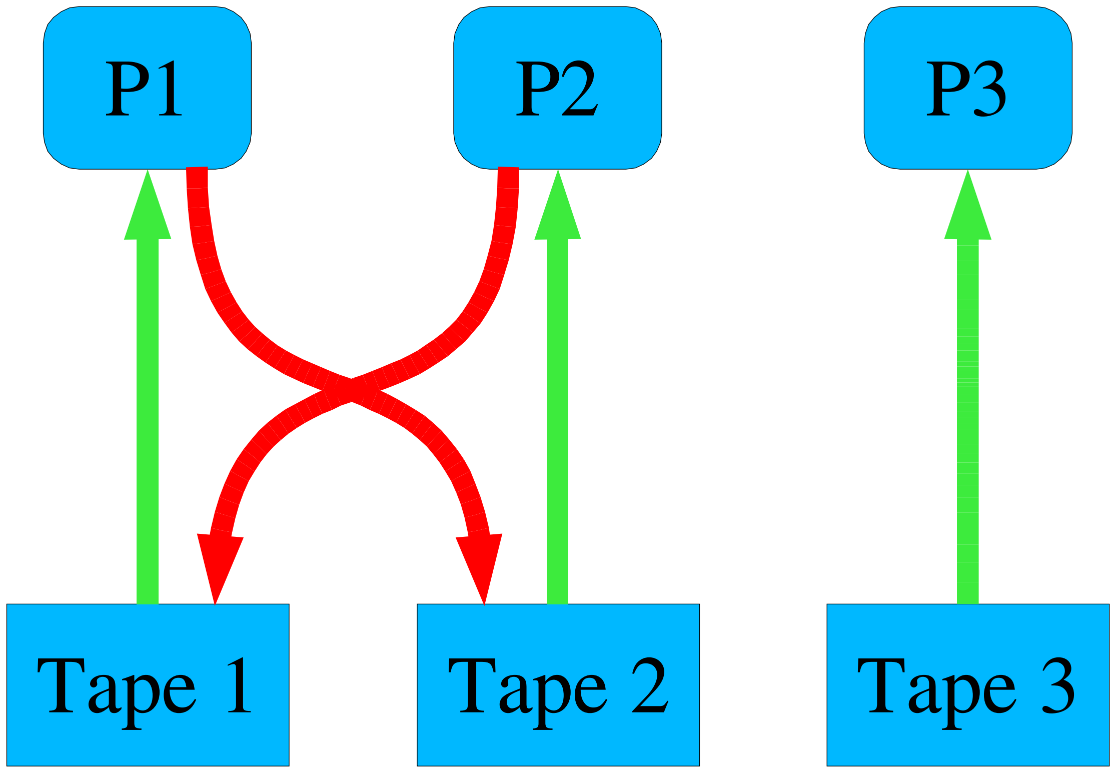
No Preemption

- Can't force a process to give up a resource
- Interrupting a CD-R write creates a “coaster”
- Obvious solution
  - CD-R device driver forbids second open()

Circular Wait

- Process 0 needs something process 4 has
- Process 4 needs something process N has
- Process N needs something process M has
- Process M needs something process 0 has

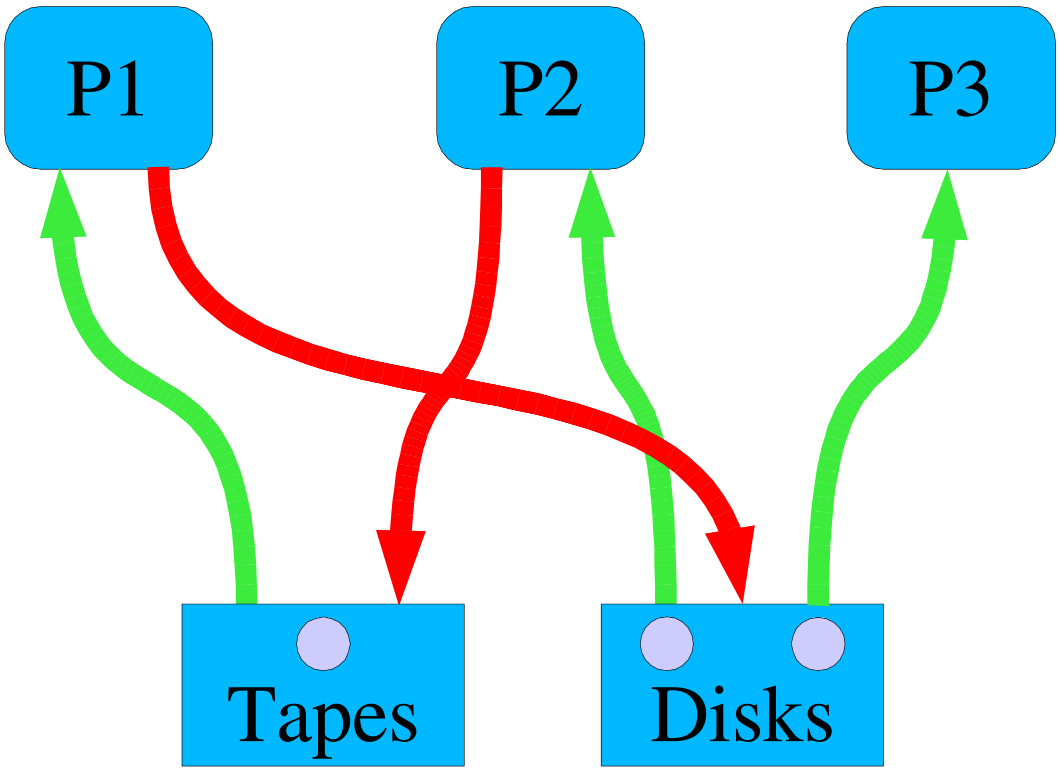
Cycle in Resource Graph



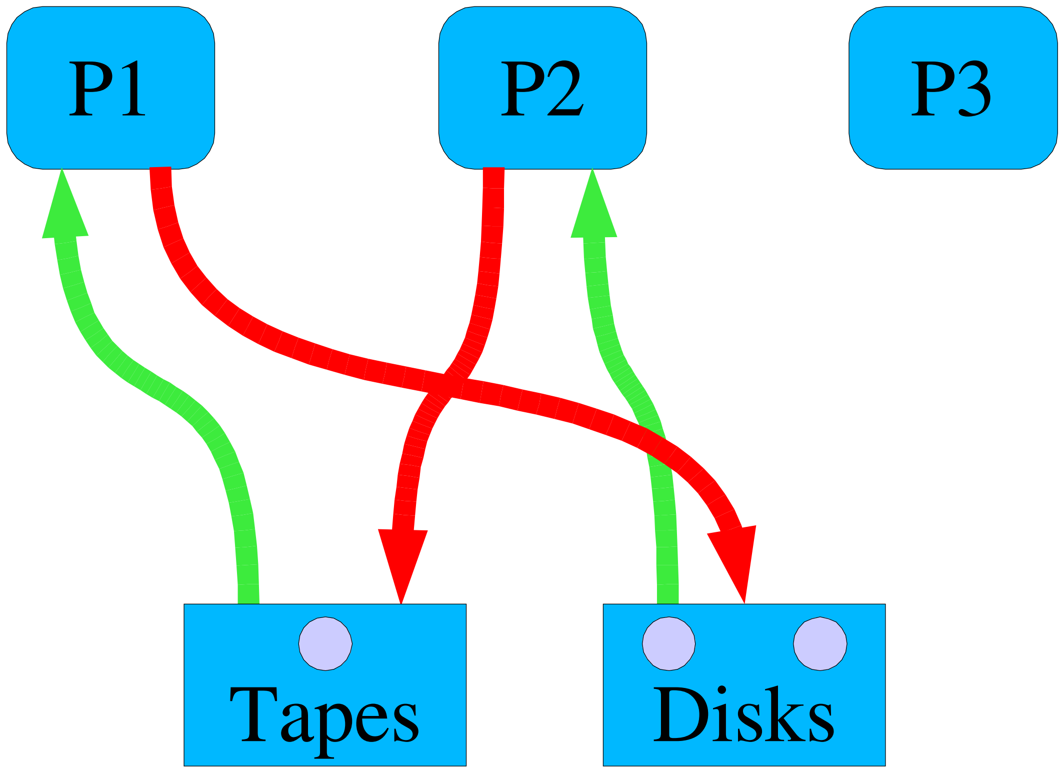
Deadlock Requirements

- Mutual Exclusion
- Hold & Wait
- No Preemption
- Circular Wait
- ***Each deadlock*** requires ***all four***

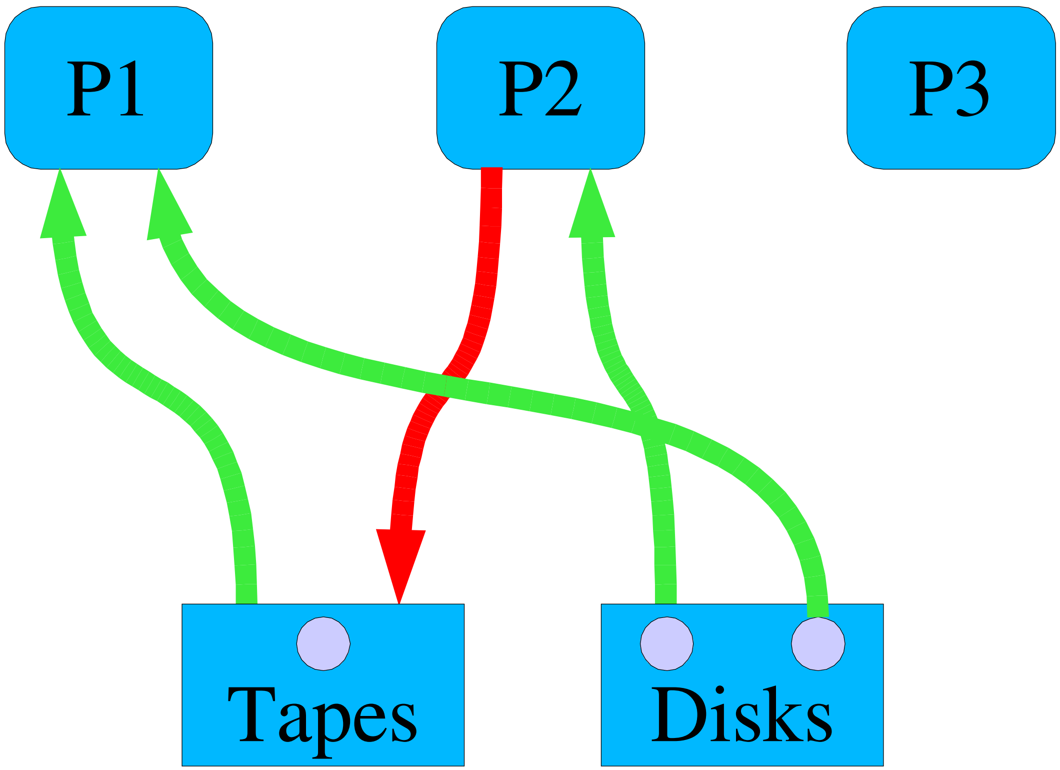
Multi-Instance Cycle



Multi-Instance Cycle *(With Rescuer!)*



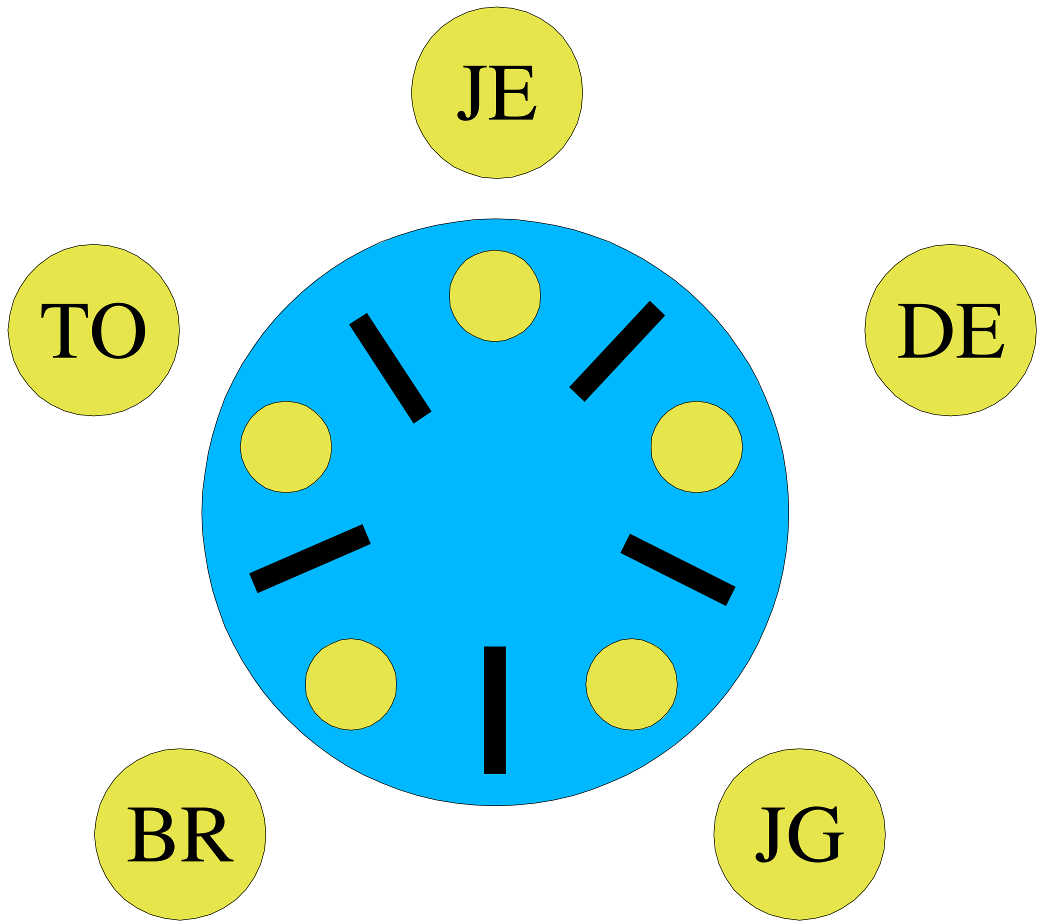
Cycle Broken



Dining Philosophers

- The scene
  - 410 staff at a Chinese restaurant
  - A little short on utensils

Dining Philosophers



Dining Philosophers

- Processes
  - 5, one per person
- Resources
  - 5 bowls (dedicated to a diner: ignore)
- 5 chopsticks
  - 1 between every adjacent pair of diners
- Contrived example?
  - Illustrates contention, starvation, deadlock

Dining Philosophers – State

```
int stick[5] = { -1 }; /* owner */
condition avail[5]; /* now avail. */
mutex table = { available };

/* Right-handed convention */
right = diner;
left = (diner + 4) % 5;
```

start_eating(int diner)

```
mutex_lock(table);
while (stick[right] != -1)
  condition_wait(avail[right], table);
stick[right] = diner;
while (stick[left] != -1)
  condition_wait(avail[left], table);
stick[left] = diner;
mutex_unlock(table);
```

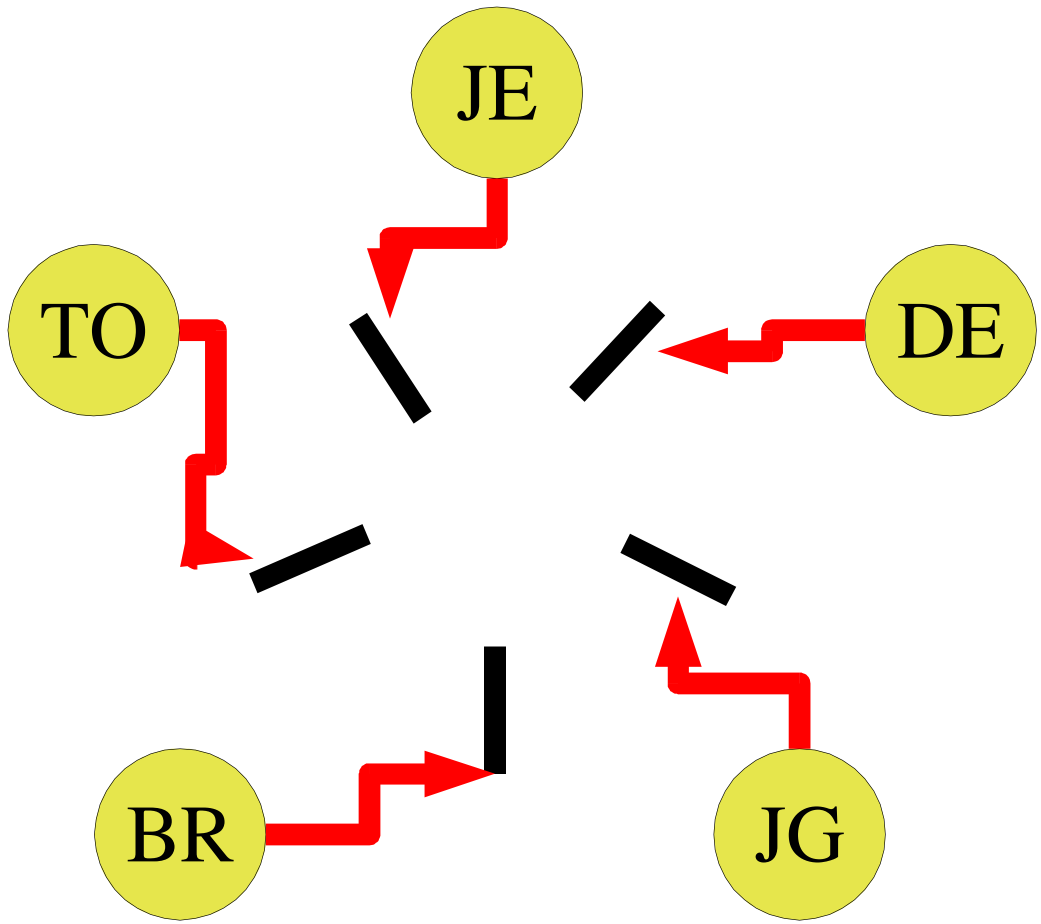
done_eating(int diner)

```
mutex_lock(table);
stick[left] = stick[right] = -1;
condition_signal(want[right]);
condition_signal(want[left]);
mutex_unlock(table);
```

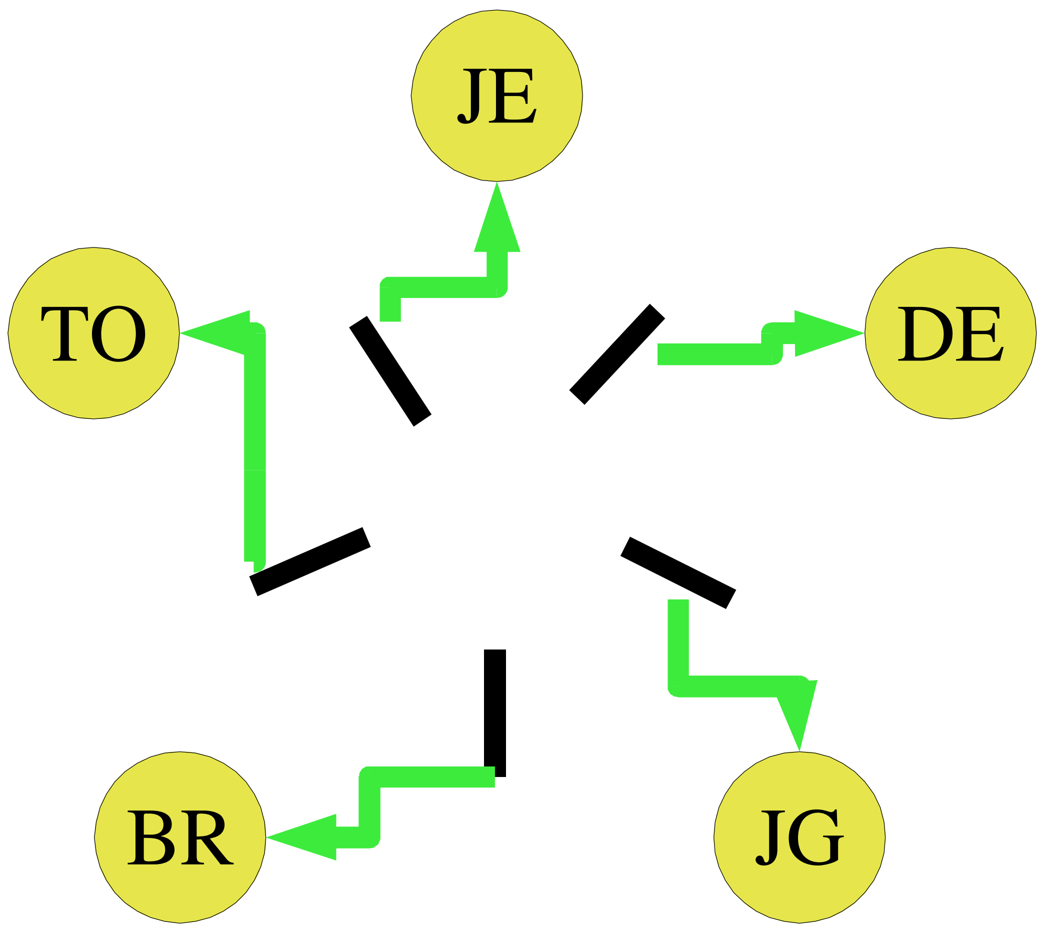
Dining Philosophers Deadlock

- Everybody reaches clockwise...
  - ...at the same time?

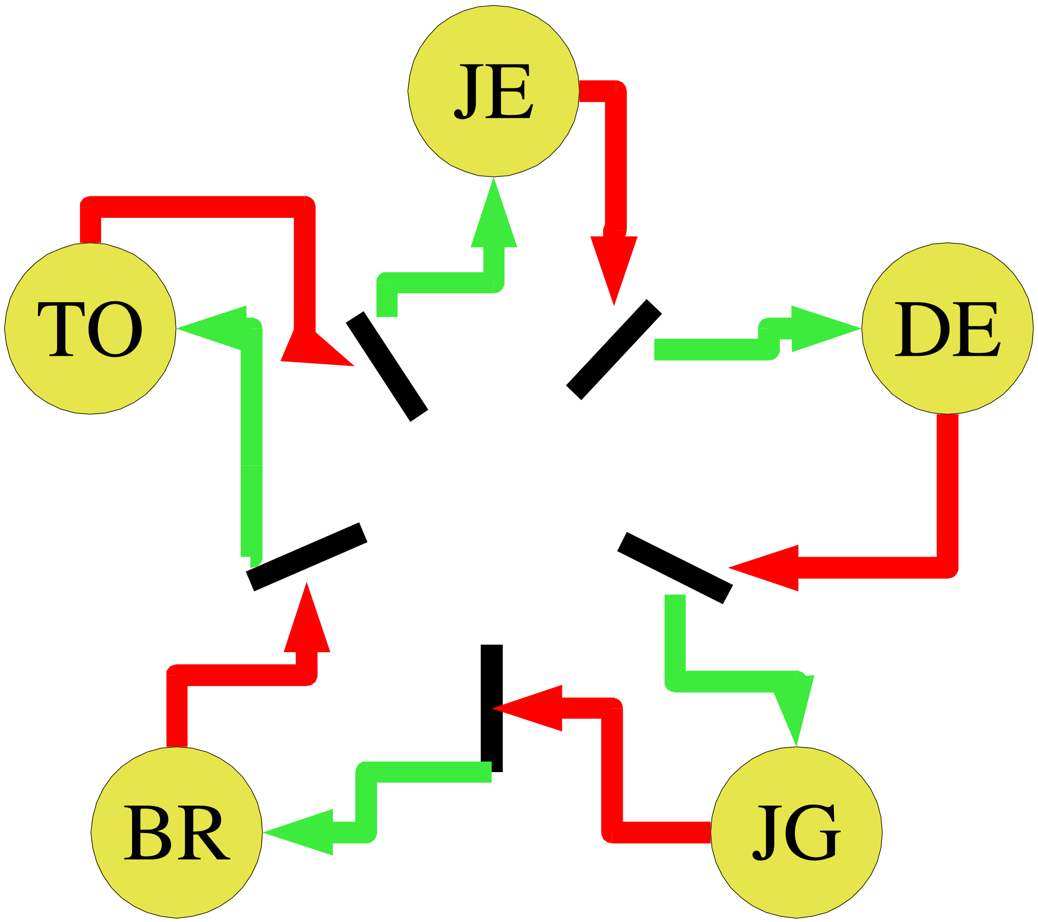
Reaching Right



Process graph



Deadlock!



Deadlock - What to do?

- Prevention
- Avoidance
- Detection/Recovery
- Just reboot when it gets “too quiet”

Prevention

- Restrict behavior or resources
  - Find a way to violate one of the 4 conditions
    - To wit...?
- What we will talk about today
  - 4 conditions, 4 possible ways

Avoidance

- Processes ***pre-declare*** usage patterns
- Dynamically examine requests
  - Imagine what other processes could ask for
  - Keep system in "safe state"

Detection/Recovery

- Maybe deadlock won't happen today...
- ...Hmm, it seems quiet...
- ...Oops, here is a cycle...
- ***Abort some process***
  - Ouch!

Reboot When It Gets “Too Quiet”

- Which systems would be so simplistic?

Four Ways to Forgiveness

- ***Each deadlock*** requires ***all four***
  - Mutual Exclusion
  - Hold & Wait
  - No Preemption
  - Circular Wait
- Prevention
  - ***Pass a law*** against one (pick one)
  - Deadlock only if somebody ***transgresses!***

Outlaw Mutual Exclusion

- ***Don't have*** single-user resources
  - Require all resources to “work in shared mode”
- Problem
  - Chopsticks???
  - Many resources don't work that way

Outlaw Hold&Wait

- Acquire resources ***all-or-none***

```
start_eating(int diner)

mutex_lock(table);
while (1)
  if (stick[lt] == stick[rt] == -1)
    stick[lt] = stick[rt] = diner
    mutex_unlock(table)
    return;
  condition_wait(released, table);
```

Problem – ***Starvation***

- Larger resource set makes grabbing harder
  - No guarantee a diner eats in bounded time
- Low utilization
  - Must allocate 2 chopsticks and waiter
  - Nobody else can use waiter while you eat

Outlaw Non-preemption

- Steal resources from sleeping processes!

```
start_eating(int diner)
right = diner;   rright = (diner+1)%5;
mutex_lock(table);
while (1)
  if (stick[right] == -1)
    stick[right] = diner
  else if (stick[rright] != rright)
    /* right can't be eating: take! */
    stick[right] = diner;
...same for left...
mutex_unlock(table);
```

Problem

- Some resources cannot be cleanly preempted
  - CD burner

Outlaw Circular Wait

- Impose ***total order*** on all resources
- Require acquisition in ***strictly increasing order***
  - Static: allocate memory, then files
  - Dynamic: ooops, need resource 0; drop all, start over

Assigning a Total Order

- Lock order: 4, 3, 2, 1, 0: right, then left
  - Issue: (diner == 0) $\Rightarrow$ (left == 4)
  - would lock(0), lock(4): ***left, then right!***

```
if diner == 0
  right = (diner + 4) % 5;
  left = diner;
else
  right = diner;
  left = (diner + 4) % 5;
...
```

Problem

- May not be possible to force allocation order
  - Some trains go east, some go west

Deadlock Prevention problems

- Typical resources require mutual exclusion
- Allocation restrictions can be painful
  - All-at-once
    - Hurts efficiency
    - May starve
  - Resource needs may be unpredictable
- Preemption may be impossible
  - Or may lead to starvation
- Ordering restrictions may not be feasible

Deadlock Prevention

- Pass a law against one of the four ingredients
  - Great if you can find a tolerable approach
- ***Very*** tempting to just let processes try their luck

Next Time

- Deadlock Avoidance
- Deadlock Recovery