Deadlocks Detection and Avoidance

Prof. Sirer

CS 4410

Cornell University

System Model

- There are non-shared computer resources
  - Maybe more than one instance
  - Printers, Semaphores, Tape drives, CPU
- Processes need access to these resources
  - Acquire resource
    - If resource is available, access is granted
    - If not available, the process is blocked
  - Use resource
  - Release resource
- Undesirable scenario:
  - Process A acquires resource 1, and is waiting for resource 2
  - Process B acquires resource 2, and is waiting for resource 1

  $\Rightarrow$ Deadlock!

Example 1: Semaphores

```
semaphore: mutex1 = 1    /* protects resource 1 */
           mutex2 = 1    /* protects resource 2 */
```

```
Process A code:
 {
    /* initial compute */
   P(file)
   P(printer)

  /* use both resources */

   V(printer)
   V(file)
}
```

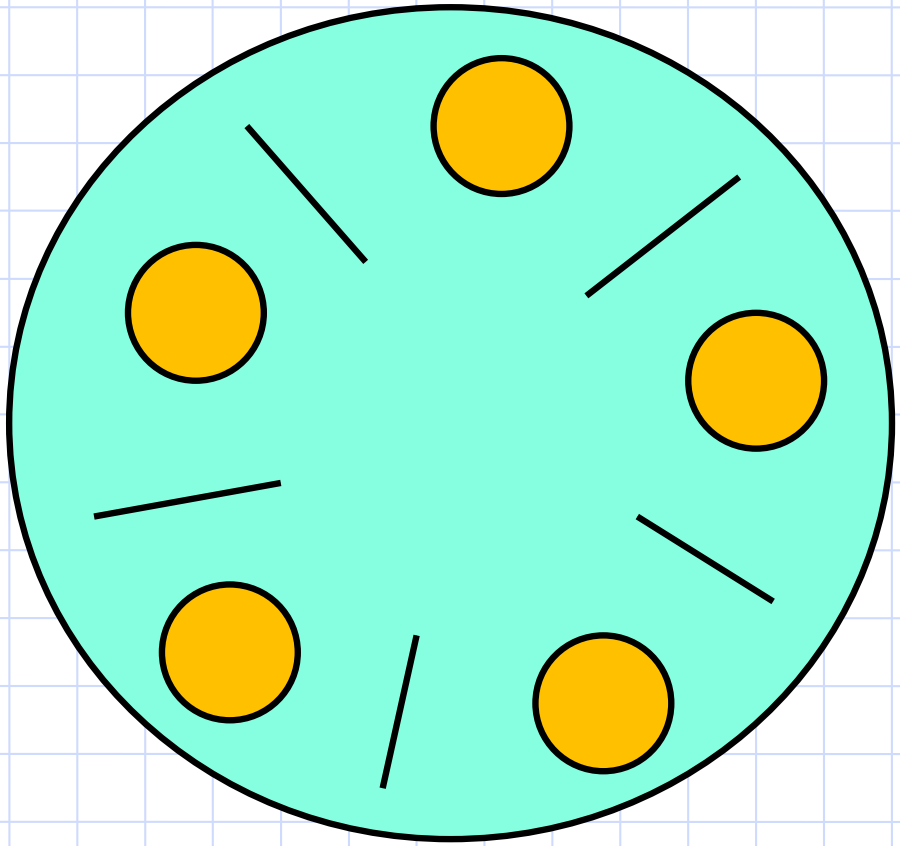
```
Process B code:
 {
    /* initial compute */
   P(printer)
   P(file)

  /* use both resources */

   V(file)
   V(printer)
}
```

Simplest deadlock

Example 2: Dining Philosophers

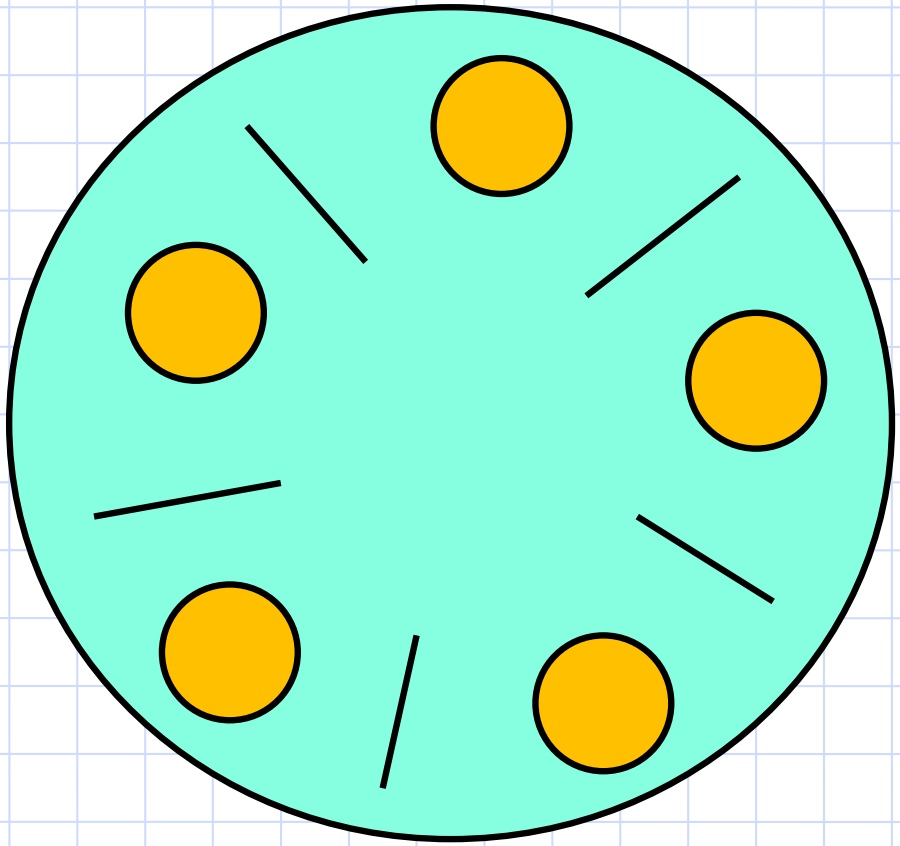


```
class Philosopher:
chopsticks[N] = [Semaphore(1),…]
Def __init__(mynum)
  self.id = mynum
Def eat():
   right = (self.id+1) % N
   left = (self.id-1+N) % N
   while True:

      # om nom nom
```

- Philosophers go out for Chinese food
- They need exclusive access to two chopsticks to eat their food

Example 2: Dining Philosophers



```
class Philosopher:
chopsticks[N] = [Semaphore(1),…]
Def __init__(mynum)
  self.id = mynum
Def eat():
   right = (self.id+1) % N
   left = (self.id-1+N) % N
   while True:
      P(left)
      P(right)
      # om nom nom
      V(right)
      V(left)
```

- Philosophers go out for Chinese food
- They need exclusive access to two chopsticks to eat their food

More Complicated Deadlock

Deadlocks

Deadlock exists among a set of processes if

- Every process is waiting for an event
- This event can be caused only by another process in the set that in turn is waiting for an event

- Typically, the event is the acquire or release of another resource

- Kansas 20th century law: "When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone"

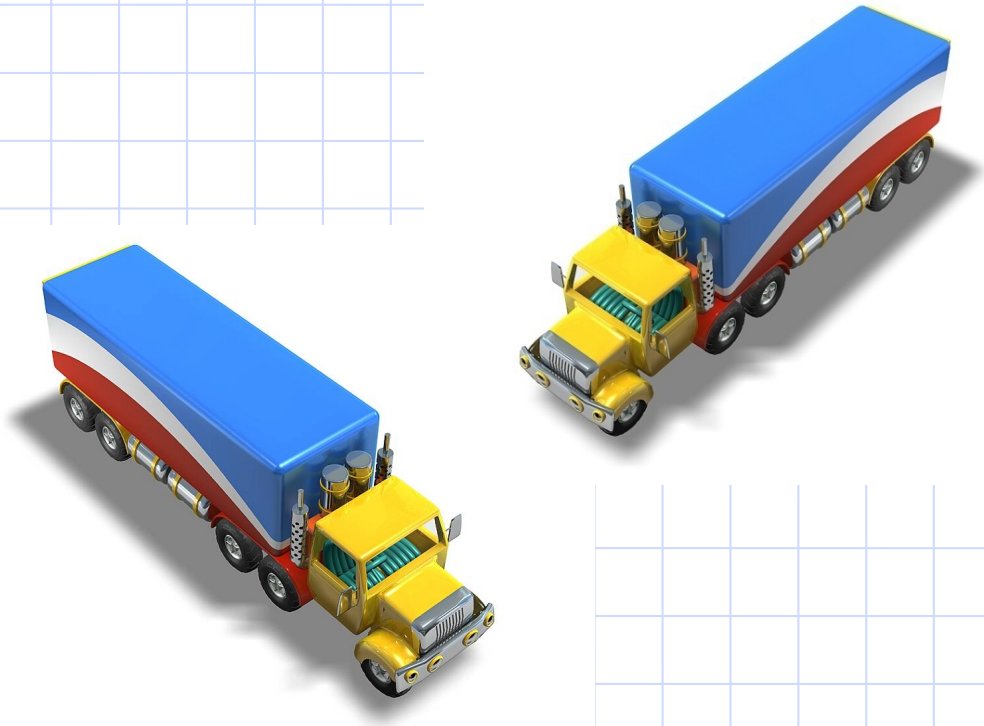
Four Conditions for Deadlock

- Necessary conditions for deadlock to exist:
  - **Mutual Exclusion**
    - At least one resource must be held in non-sharable mode
  - **Hold and wait**
    - There exists a process holding a resource, and waiting for another
  - **No preemption**
    - Resources cannot be preempted
  - **Circular wait**
    - There exists a set of processes $\{P_1, P_2, \ldots P_N\}$, such that
      - P_1 is waiting for P_2, P_2 for P_3, …. and P_N for P_1

All four conditions must hold for deadlock to occur

Real World Deadlocks?

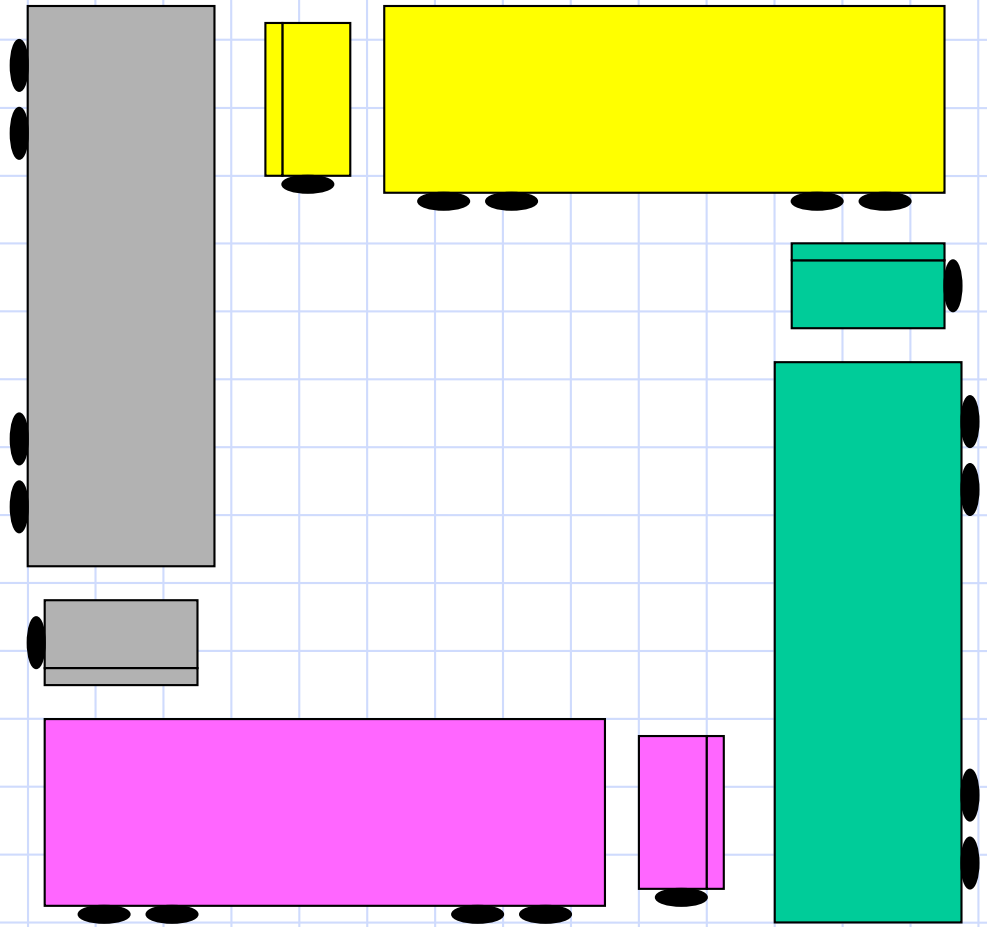
- Truck A has to wait for truck B to move



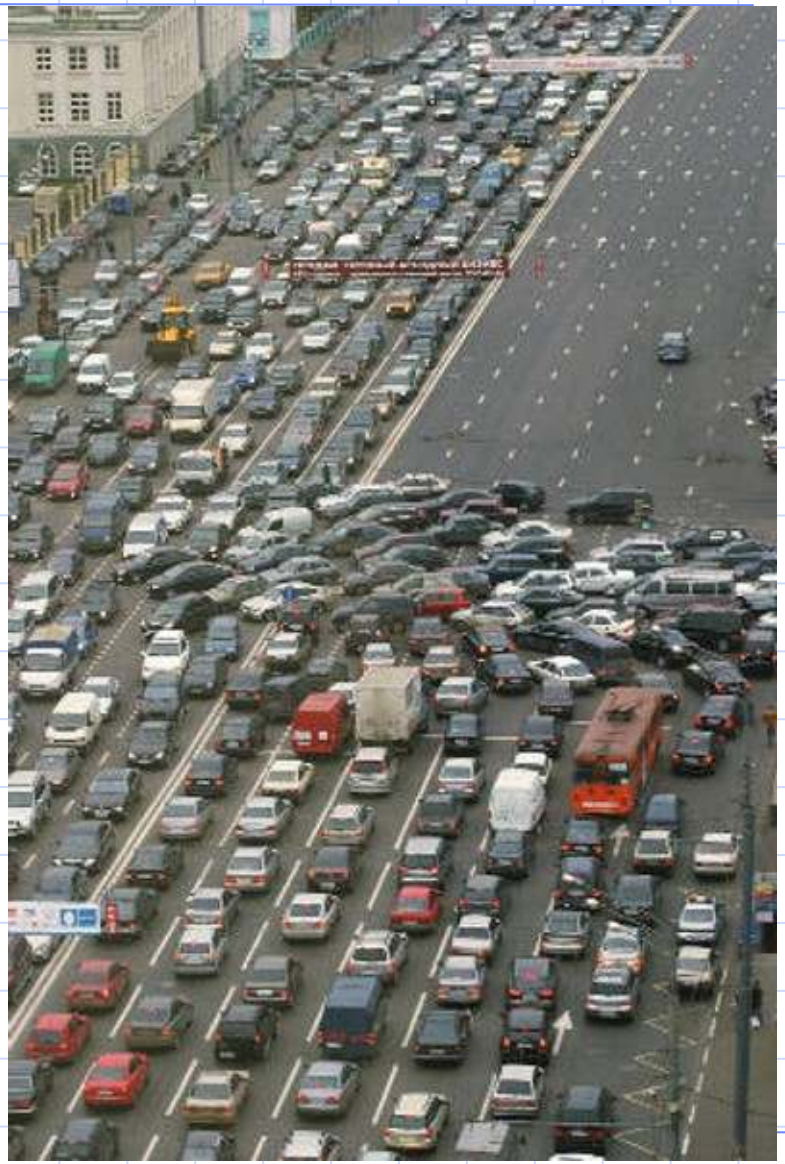
- Not deadlocked

Real World Deadlocks?

- Gridlock

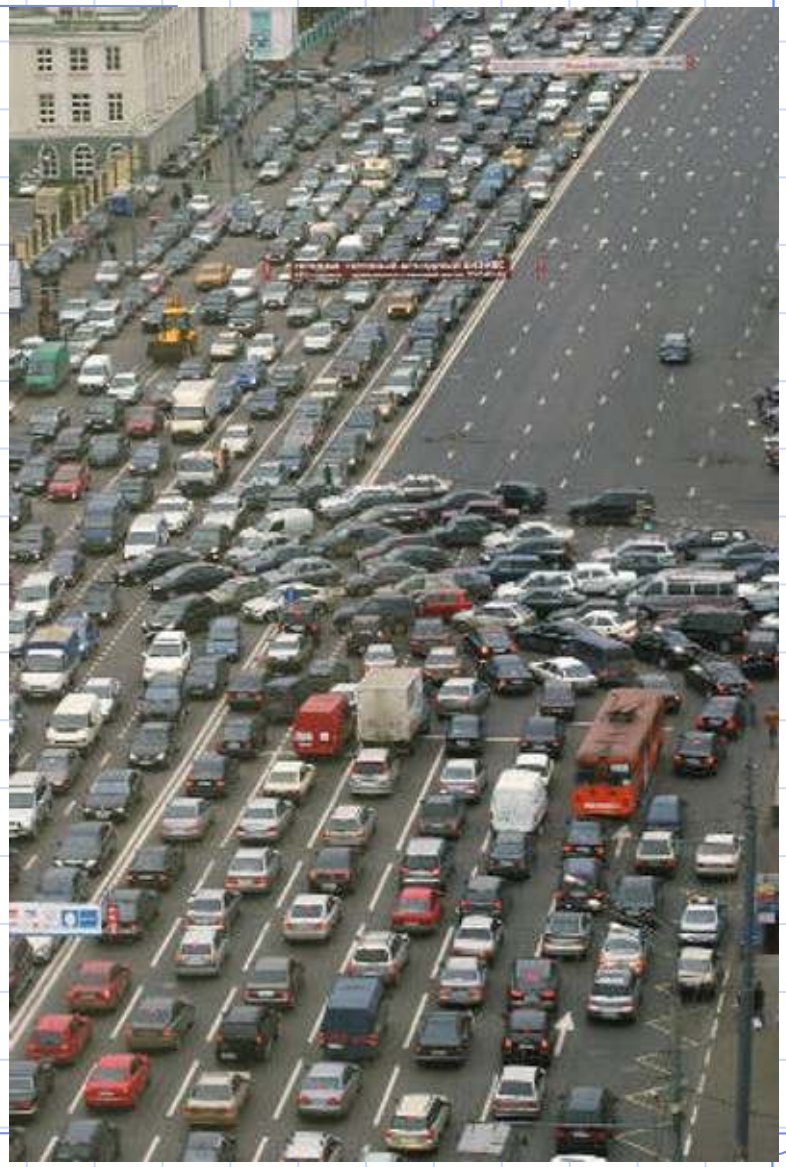


Deadlock in Real Life?



Deadlock in Real Life?

- No circular wait!
- Not a deadlock!
  - At least, not as far as we can see from the picture
- Will ultimately resolve itself given enough time



Deadlock in Real Life

Deadlock Prevention

Deadlock Prevention

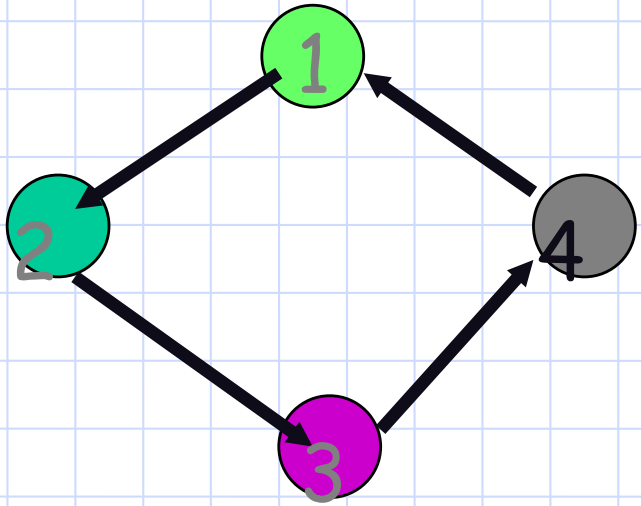
- Can the OS prevent deadlocks?
- Prevention: Negate one of necessary conditions
  - Mutual exclusion:
    - Make resources sharable
    - Not always possible (printers?)
  - Hold and wait
    - Do not hold resources when waiting for another
    - ⇒ Request all resources before beginning execution
    - Processes do not know what all they will need
    - Starvation (if waiting on many popular resources)
    - Low utilization (Need resource only for a bit)
    - Alternative: Release all resources before requesting anything new
      - Still has the last two problems

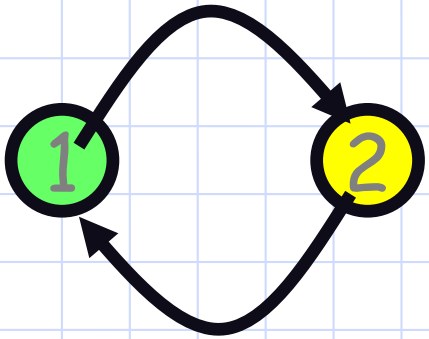
Deadlock Prevention

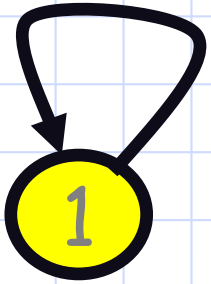
- Prevention: Negate one of necessary conditions
  - No preemption:
    - Make resources preemptable (2 approaches)
      - Preempt requesting processes' resources if all not available
      - Preempt resources of waiting processes to satisfy request
    - Good when easy to save and restore state of resource
      - CPU registers, memory virtualization
  - Circular wait: (2 approaches)
    - Single lock for entire system? (Problems)
    - Impose partial ordering on resources, request them in order

Deadlock Prevention

- Prevention: Breaking circular wait
  - Order resources (lock1, lock2, …)
  - Acquire resources in strictly increasing/decreasing order
  - When requests to multiple resources of same order:
    - Make the request a single operation
  - Intuition: Cycle requires an edge from low to high, and from high to low numbered node, or to same node







  - Ordering not always possible, low resource utilization