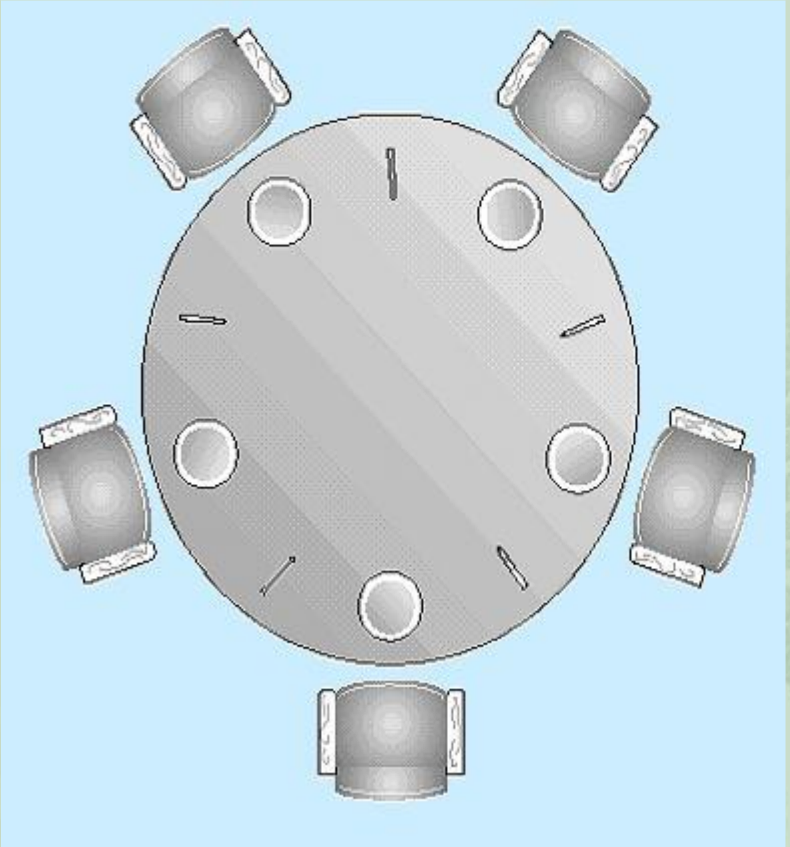
The Dining Philosophers Problem

- 5 philosophers who only eat and think.
- Each need to use 2 forks for eating.
- There are only 5 forks.
- Classical synchronization problem.
- Illustrates the difficulty of allocating resources among process without deadlock and starvation.



Solution??

```
Process Pi:
repeat
  think;
  wait(forks[i]);
  wait(forks[(i+1)%5]);
  eat;
  signal(forks[(i+1)%5]);
  signal(forks[i]);
forever
```

- Each philosopher is a process.
- One semaphore per fork:
  - forks: array[0..4] of semaphores
  - Initialization: forks[i].count:=1 for i:=0..4

- Deadlock if each philosopher starts by picking left fork!

Another Solution

- A solution: admit only 4 philosophers at a time that tries to eat
- Then 1 philosopher can always eat when the other 3 are holding 1 fork
- Introduce semaphore T that limits to 4 the number of philosophers "sitting at the table"
- Initialize: T.count:=4

```
Process Pi:
repeat
  think;
  wait(T);
  wait(forks[i]);
  wait(forks[(i+1)%5]);
  eat;
  signal(forks[(i+1)%5]);
  signal(forks[i]);
  signal(T);
forever
```

Solving Dining Philosophers

- Buy more Forks
  - Equivalent to increasing resources
- Put fork down if 2nd fork busy
  - Can produce "livelock" if philosophers stay synchronized
- Room Attendant
  - Only let 4 of the philosophers into the room at once
  - May have 4 philosophers in room, but only 1 can eat
- Left-Handed Philosophers (asymmetric solution)
  - Grab forks in the other order (right fork, then left fork)
- A philosopher may only pick up forks in pairs.
  - must allocate all resources at once

Synchronization in Windows

- Windows provides four mechanisms
  - Event
  - Mutex
  - Semaphore
  - Timer
- Each mechanism provides a slightly different behavior

Unix Semaphores

- Generalization of the counting semaphores (more operations are permitted).
- A semaphore includes:
  - the current value S of the semaphore
  - number of processes waiting for S to increase
  - number of processes waiting for S to be 0
- Uses queues of processes that are blocked on a semaphore
- The system call ***semget*** creates an array of semaphores
- The system call ***semop*** performs a list of operations: one on each semaphore (atomically)

Conclusion

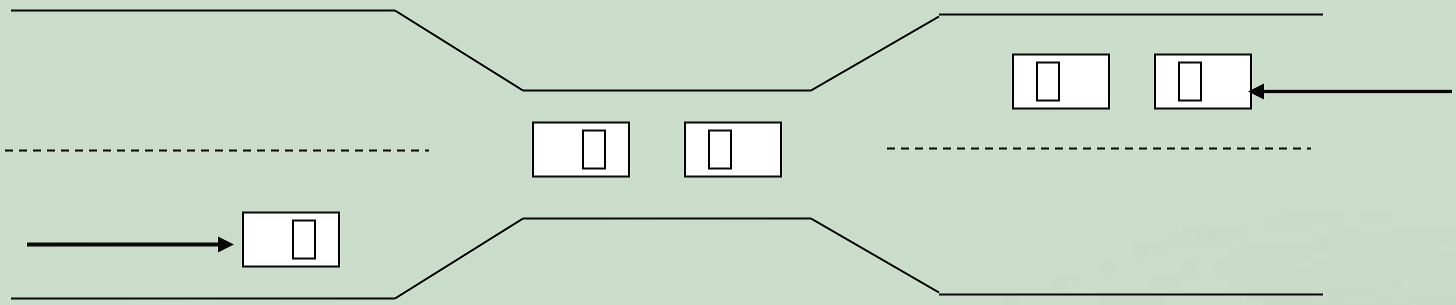
- Semaphores are a powerful tool for enforcing mutual exclusion and to coordinate processes
- But wait/signal, P/V, lock/unlock operations are complicated to use.
- Usage must be correct in all the processes
  - One bad (or malicious) process can fail the entire collection of processes

Deadlock

Deadlock

- System Model
  - Process must request a resource before using
  - Process must release the resource when done
- Deadlock
  - A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set.

Bridge Crossing Example

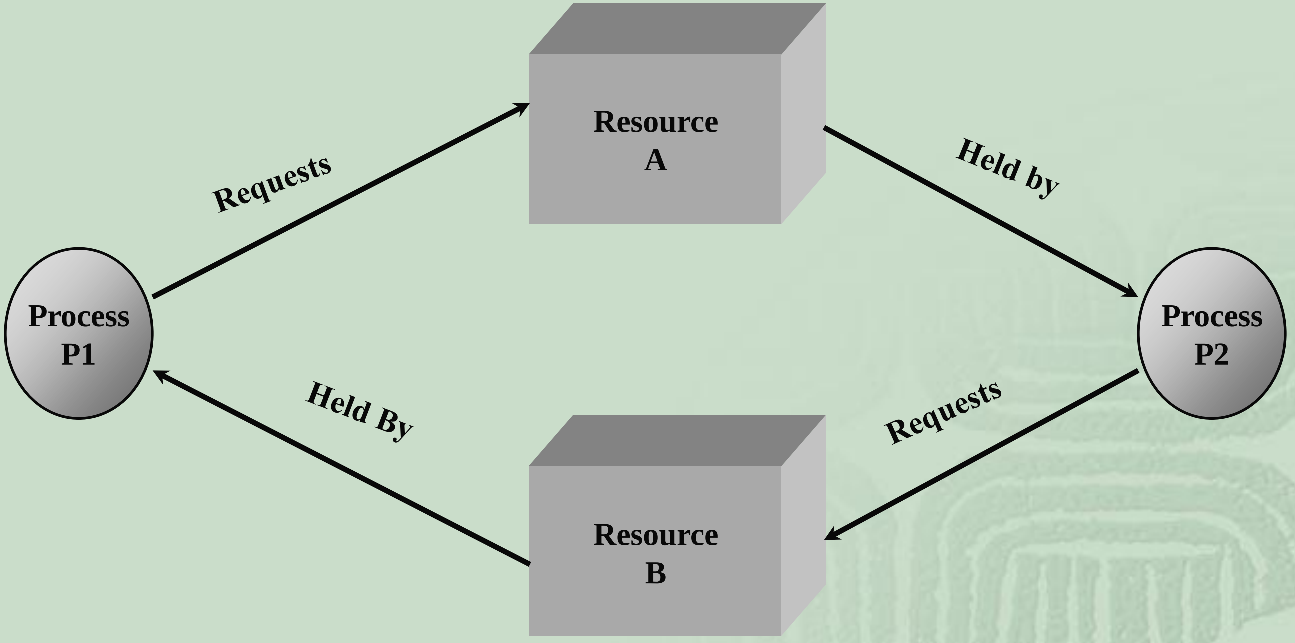


- Traffic only in one direction (Niska Rd).
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

Deadlock Characterization

- Necessary but not sufficient conditions
  - Mutual exclusion
  - Hold and wait
  - No preemption
- Required condition
  - Circular wait - a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain
  - Unresolvable circular wait is the definition of deadlock!
- All four conditions must hold for deadlock

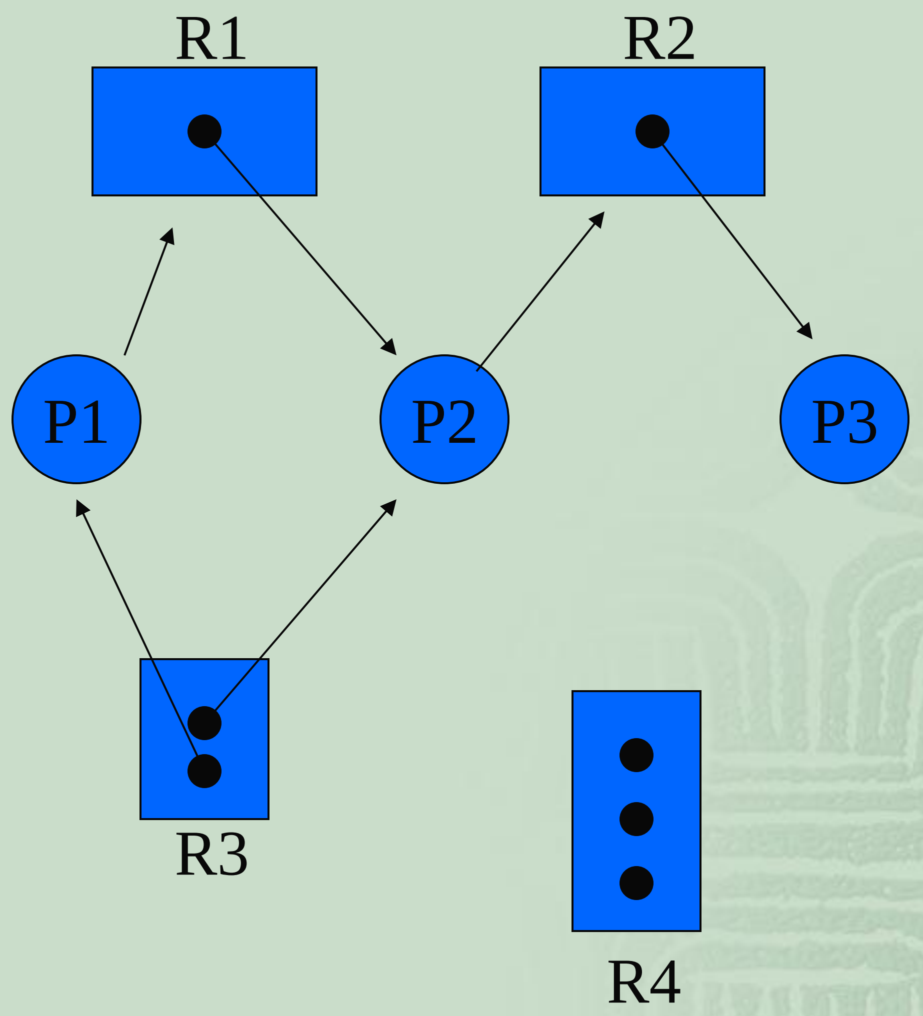
Circular Wait



Describing Deadlock

- Deadlocks can be described using resource allocation graph
  - Vertices
    - Active processes {P1, P2, … }
    - Resources {R1, R2, … }
  - Edges
    - A directed edge from Pi to Rj
      - Process Pi requested an instance of resource Rj
    - A directed edge from Rj to Pi
      - Resource Rj has been allocated to process Pi
  - Process are circles, Resources are rectangles

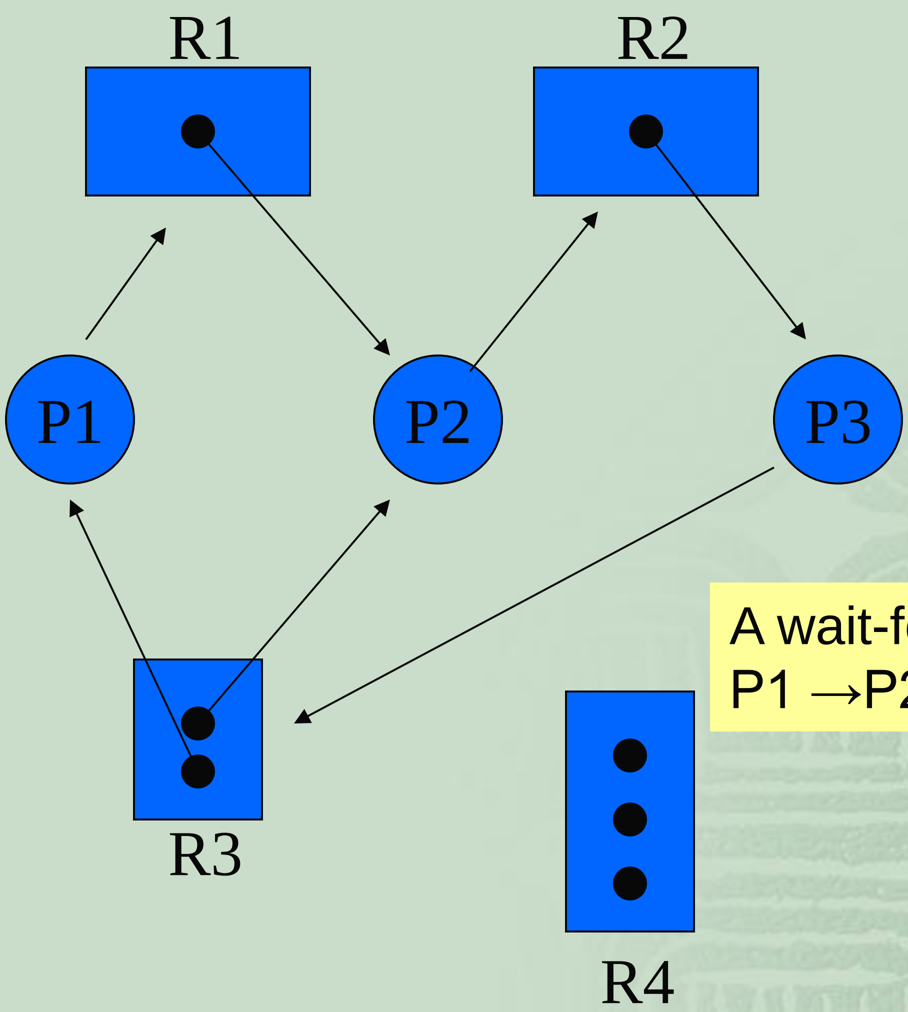
Resource Allocation Graph



So?

- If a graph contains no cycles, then no process in the system is deadlocked
- If the graph contains a cycle, deadlock MAY exist
- If each resource has only one instance, then a cycle implies deadlock

Deadlock?



Is there a cycle?

Is there deadlock?

A wait-for circle:
P1 →P2 → P3 → (P1 or P2)

Deadlock

- Reusable Resources
  - Can only be used by one process at a time. After use, can be reassigned to another process (printer, memory, files, etc.)
  - Deadlock can occur with two processes copying from disk to tape
  - Deadlock can occur with memory allocation when there is 200K available, both processes want 80K, then make second request for 70K
- Consumable Resources
  - Can be created or destroyed (signals, messages)
  - No fixed limit on # of resources
  - Can deadlock if both waiting for a message from the other

Examples of Deadlock

- 200K bytes is available for allocation…

Process 1
Request 80K bytes
…
Request 60K bytes
…

Process 2
…
Request 70K bytes
…
Request 80K bytes

- Deadlock occurs if receive blocks…

Process 1
Receive(P2)
…
Send(P2)
…

Process 2
…
Receive(P1)
…
Send(P1)