STEVENS INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 492: Operating Systems

Deadlocks

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

Deadlocks

Goals for Today

- Deadlock
  - Concepts
  - How to deal with deadlocks?
    - Ignoring them: ostrich algorithm
    - Detection & recovery
    - Prevention
    - Avoidance

Resources

- Resource: objects granted to the processes

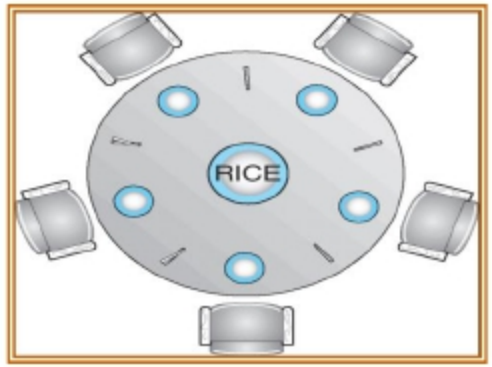
- Two types of resources
  - Preemptable resources
    - can be taken away from a process with no ill effects (e.g., memory)
  - Nonpreemptable resources
    - will cause the process to fail if taken away (e.g. printer)

Deadlocks

The cause of deadlocks: Each process needing what another process has. This results from sharing resources such as memory, devices, links.

- Example
  - System has 2 disk drives.
  - P_1 and P_2 each holds one disk drive and each needs another one.

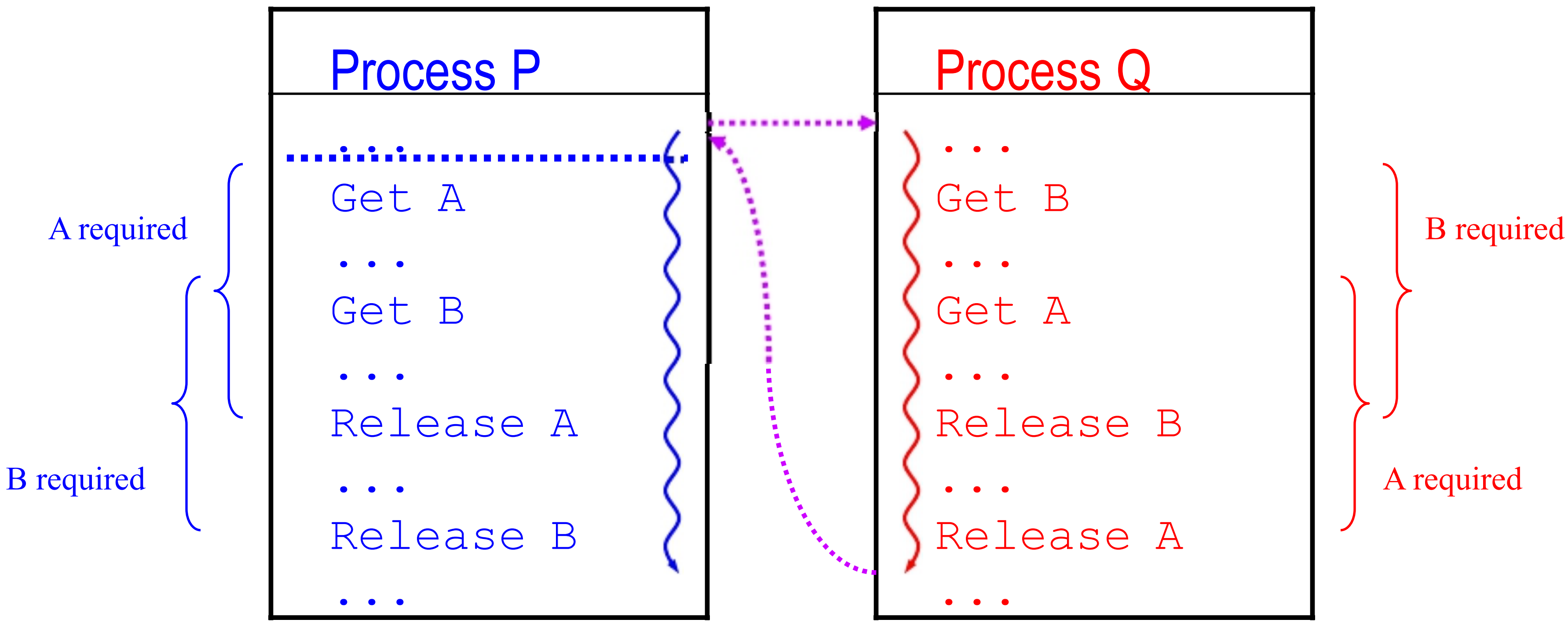
Dining Lawyers Problem Example

- Five chopsticks/Five lawyers (really cheap restaurant)
  - Free-for all: Lawyer will grab any one they can
  - Need two chopsticks to eat
- What if all grab at same time and wait for the one on the left (right)?
  - Deadlock!

Process Example

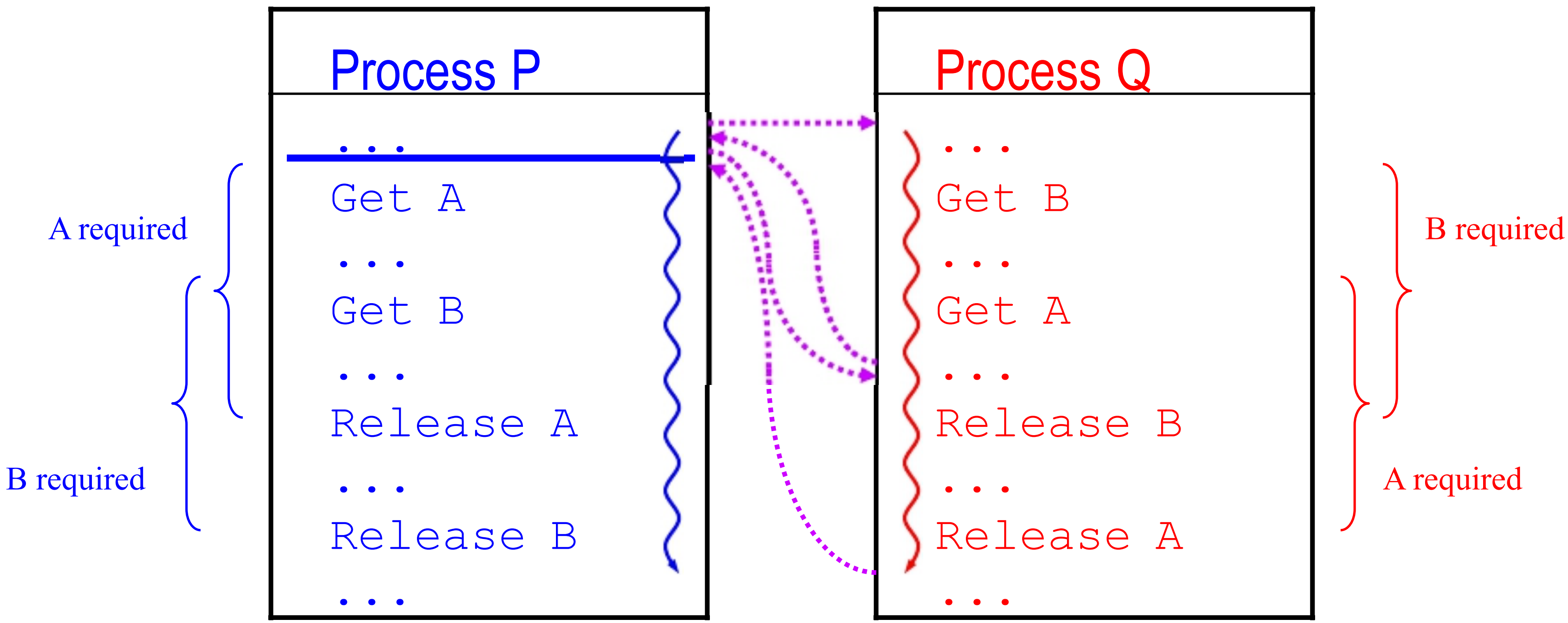
- Illustration of a deadlock — scheduling path 1 ☺
  - Q executes everything before P can ever getA
  - when P is ready, resources A and B are free and P can proceed



Happy scheduling 1

Process Example

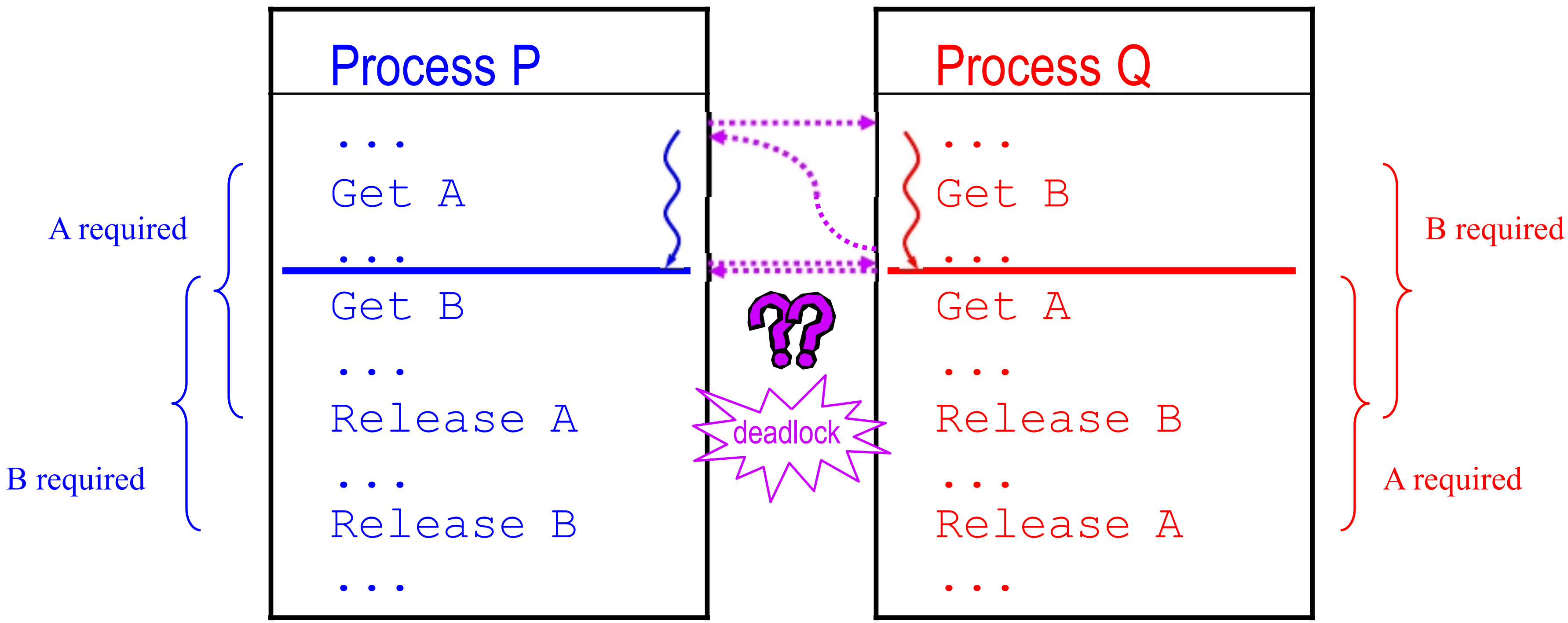
- Illustration of a deadlock — scheduling path 2 ☺
  - Q gets B and A, then P is scheduled; P wants A but is blocked by A's mutex; so Q resumes and releases B and A; P can now go



Happy scheduling 2

Process Example

- Illustration of a deadlock — scheduling path 3 ☹
  - Q gets only B, then P is scheduled and gets A; now both P and Q are blocked, each waiting for the other to release a resource



Bad scheduling → deadlock

Semaphore Example

semaphore:

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

Process A code:

```
{
   /* initial compute */
  down(mutex1)
  down(mutex2)

  /* use both resources */

  up(mutex2)
  up(mutex1)
}
```

Process B code:

```
{
   /* initial compute */
  down(mutex2)
  down(mutex1)

  /* use both resources */

  up(mutex2)
  up(mutex1)
}
```

Deadlock Definition

- Formal definition :

A deadlock is a situation in which two or more competing processes are each waiting for the other to finish, and thus neither ever does.

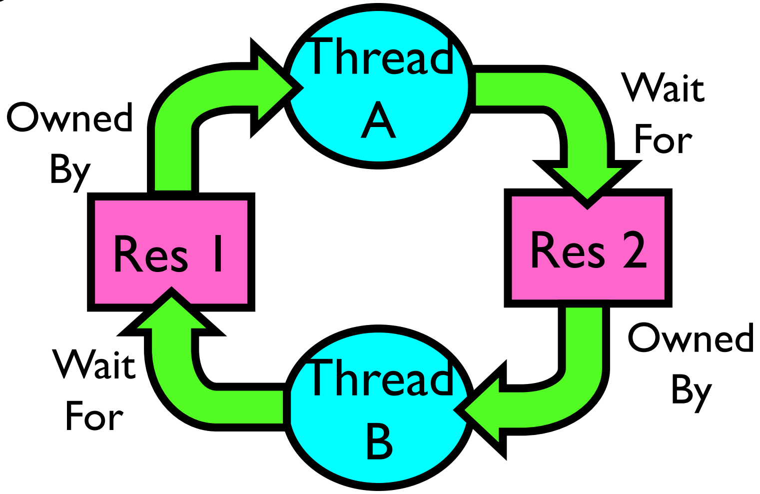
  - Usually the event waiting for is the release of a currently held resource
- In deadlock, none of the processes can …
  - run
  - release resources
  - be awakened
- The number of processes and resources is unimportant

Conditions for Resource Deadlocks

- Mutual exclusion
  - Only one thread at a time can use a resource
- Hold and wait
  - Thread holding at least one resource is waiting to acquire additional resources held by other threads
- No preemption
  - Resources are released only voluntarily by the thread holding the resource, after thread is finished with it
- Circular wait
  - There exists a set $\{T_1, \ldots, T_n\}$ of waiting threads
    - T_1 is waiting for a resource that is held by T_2
    - T_2 is waiting for a resource that is held by T_3
    - …
    - T_n is waiting for a resource that is held by T_1

Starvation Vs Deadlock

- Starvation vs. Deadlock
  - Starvation: thread waits indefinitely
    - Example, low-priority thread waiting for resources constantly in use by high-priority threads
  - Deadlock: circular waiting for resources
    - Thread A owns Res 1 and is waiting for Res 2 Thread B owns Res 2 and is waiting for Res 1



  - Deadlock $\Rightarrow$ Starvation but not vice versa

Goals for Today

- Deadlock
  - Concepts
  - How to deal with deadlocks?

Strategies Dealing with Deadlocks

Allow deadlock to happen. This requires using either

I. Ostrich algo: ignore the problem altogether
II. Deadlock ***detection and recovery***: allow deadlock, detect it, break it

Ensure deadlock **never** occurs using either

I. Deadlock ***prevention***: negate one of the four necessary conditions
II. Dynamic ***avoidance***: careful resource allocation – each resource request is analyzed and denied if deadlock might result

Outline

- Deadlock
  - Concepts
  - How to deal with deadlocks
    - Ignoring them: ostrich algorithm
    - Detecting & recovering from deadlock
    - Preventing deadlock
    - Avoiding deadlock

The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- UNIX and Windows take this approach
- It is a trade-off between
  - convenience
  - correctness

Outline

- Deadlock
  - Concepts
  - How to deal with deadlocks
    - Ignoring them: ostrich algorithm
    - Detecting & recovering from deadlock
    - Preventing deadlock
    - Avoiding deadlock

Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

System Model

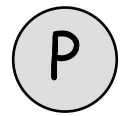
- Resource types
  - $R_1, R_2, \ldots. R_m$
  - E.g., printers, disks,
- Each resource type R_i has W_i instances
  - E.g., 3 printers, 5 disks, etc.
- Assume serially reusable resources
  - request -> use -> release

Resource-Allocation Graph

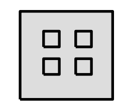
- A set of vertices V and a set of edges E
- V is partitioned into two types:
  - *Process vertices* $P = \{P_1, P_2, \ldots, P_n\}$, the set of processes
  - *Resource vertices* $R = \{R_1, R_2, \ldots, R_m\}$, the set of resource types (not resources!)
- E is partitioned into two types:
  - *request edge* – directed edge $P_i \rightarrow R_j$
  - *assignment edge* – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph

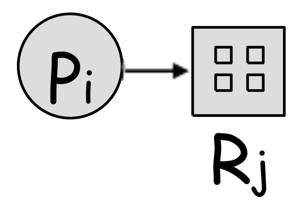
- Process node



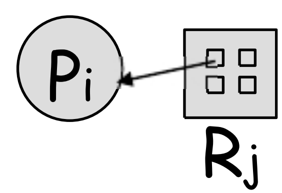
- Resource node with 4 instances



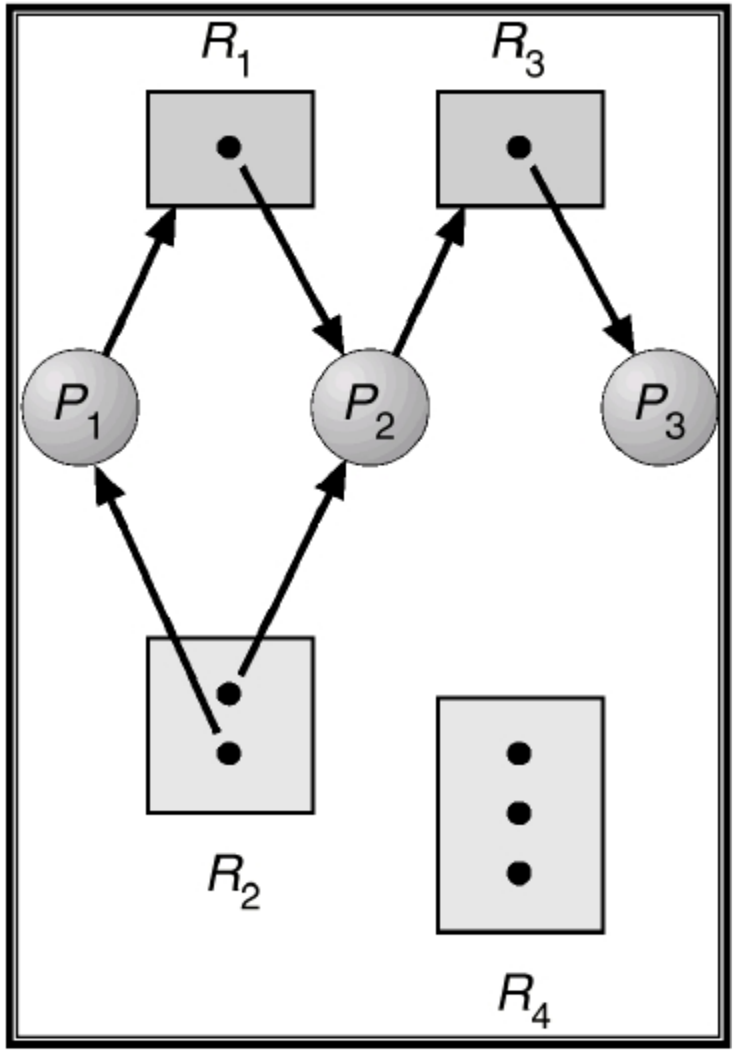
- P_i requests instance of R_j



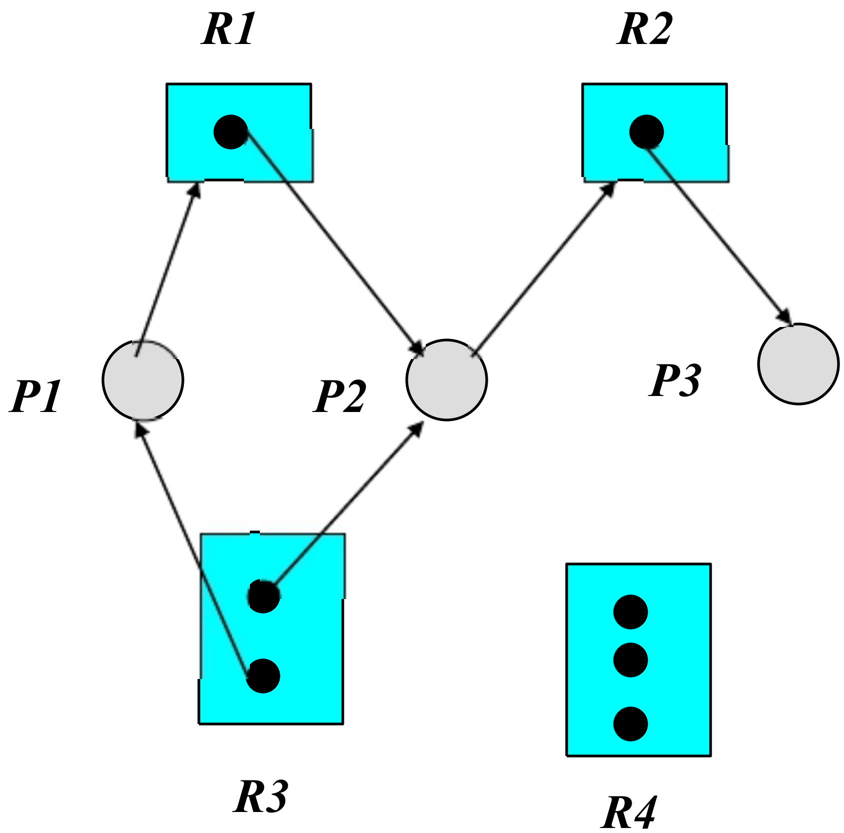
- P_i is holding an instance of R_j



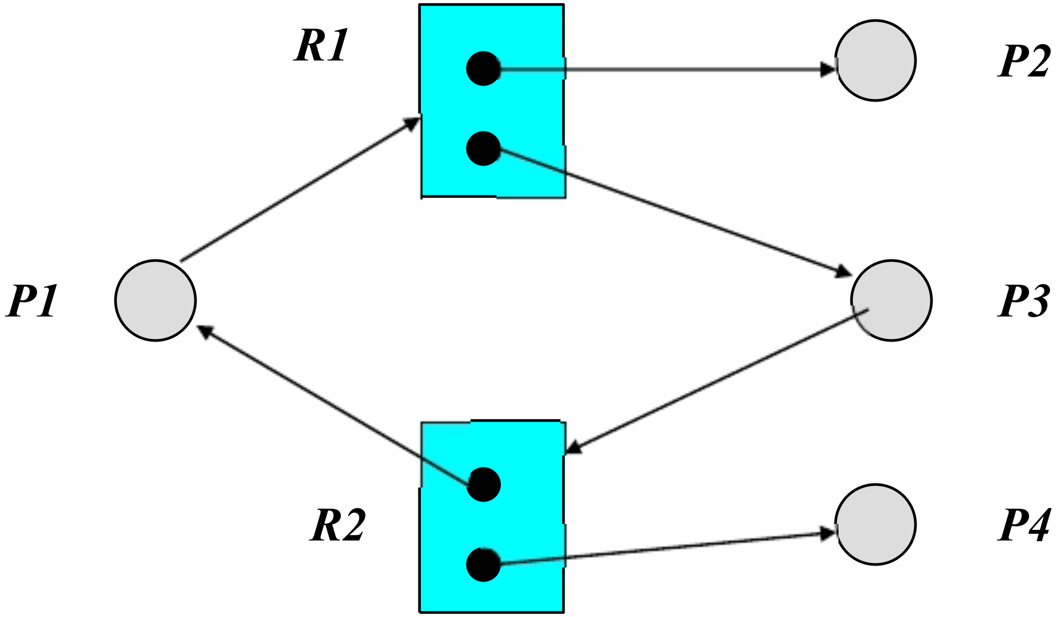
Example of a ResourceAllocation Graph



Graph with no cycles

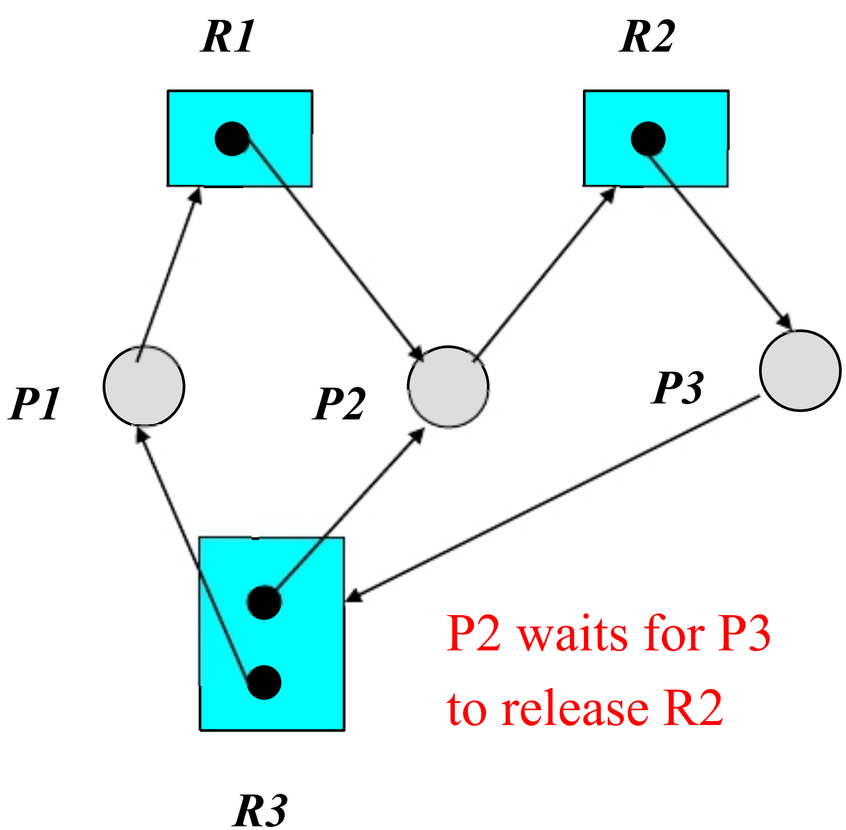


Graph with cycles



Cycles in Resource Allocation Graph

Multiple Resources of Each Type



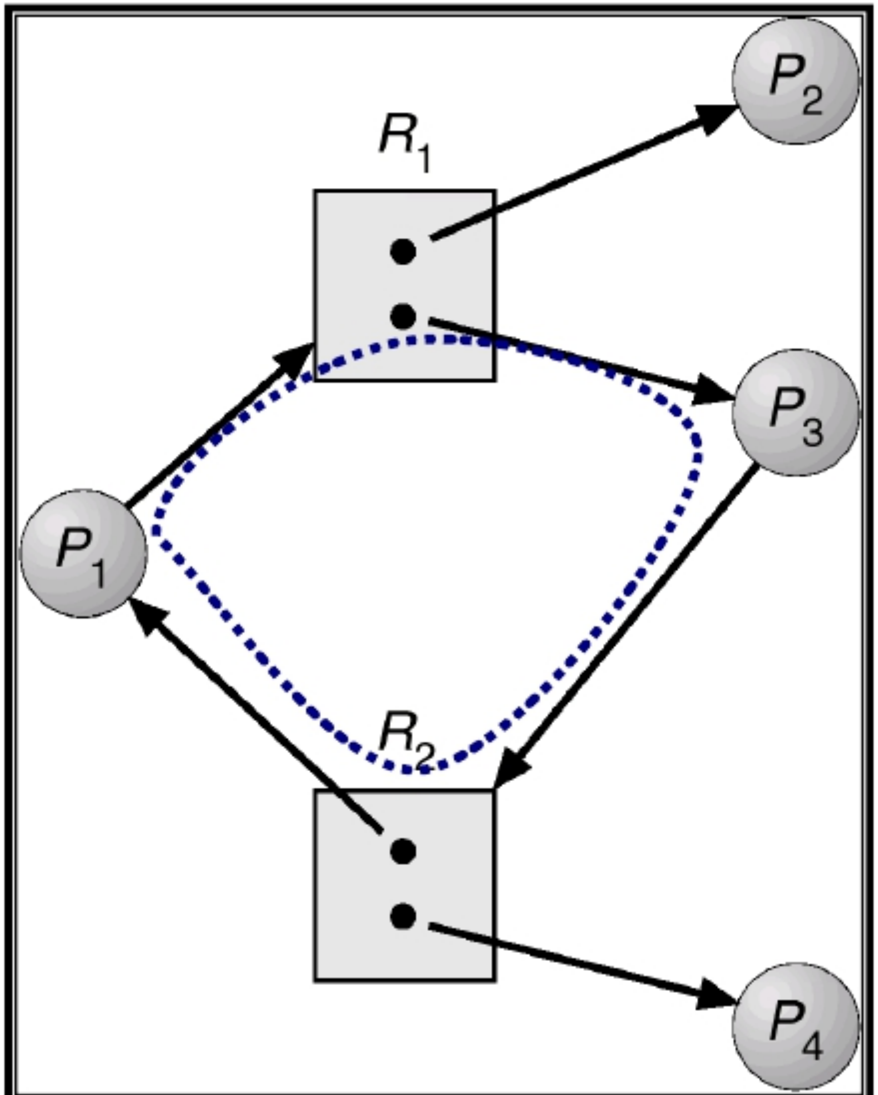
P1 waits for P2 to release R1

P3 waits for either P1 or P2 to release R3

P2 waits for P3 to release R2

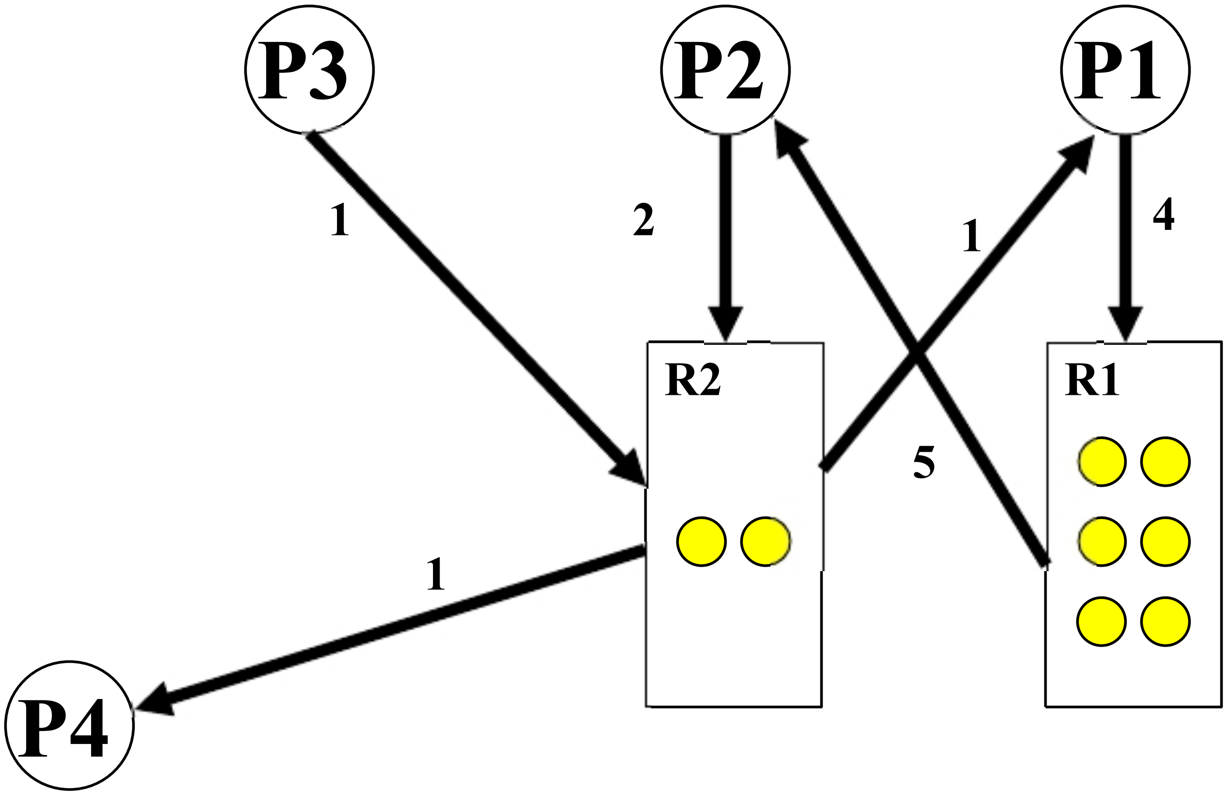
Deadlock!

Graph with Cycles But No Deadlocks

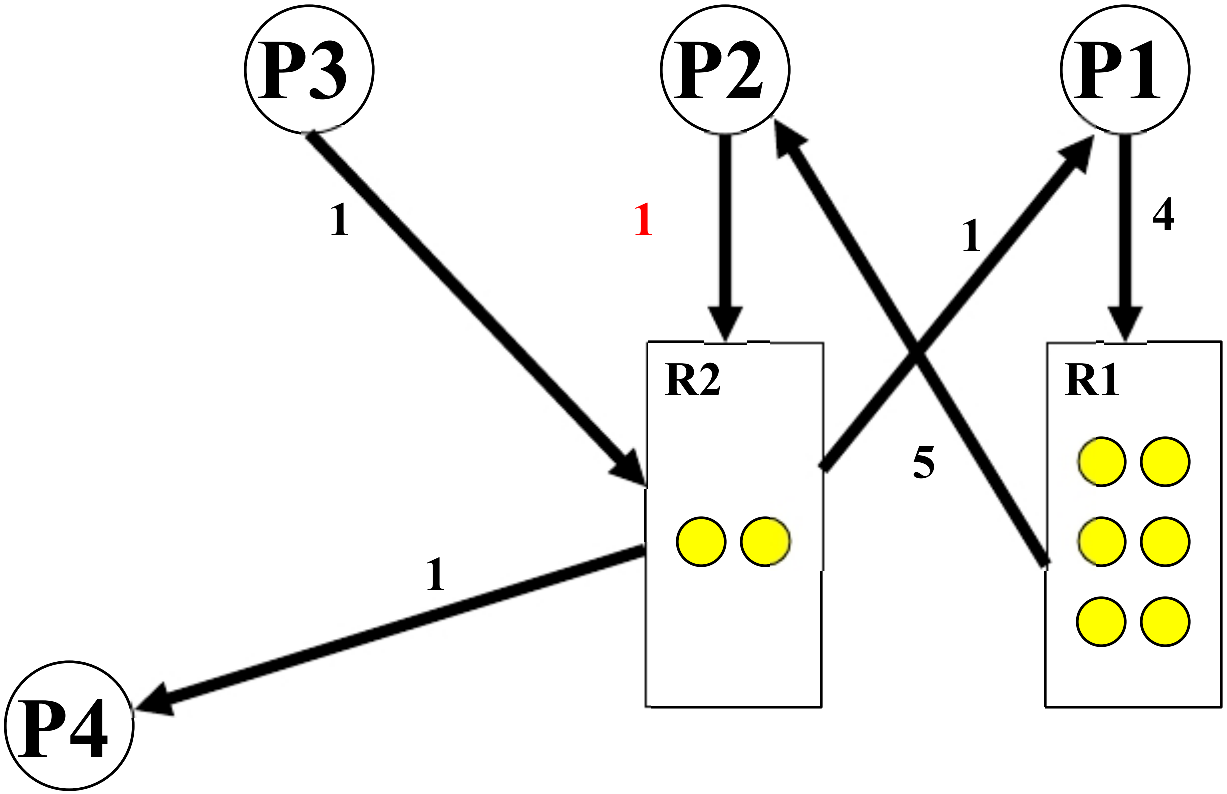


Cycle But No Deadlock!

A cycle is not sufficient to imply a deadlock



Is there a deadlock?



Is there a deadlock?