CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Introduction to Deadlock

Ch 8.1 – 8.3

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Potential Deadlock Example

```
/* thread one runs in this function */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /** * Do some work */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);
   pthread_exit(0);
}
/* thread two runs in this function */
void *do_work_two(void *param)
{
   pthread_mutex_lock(&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /** * Do some work */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);
   pthread_exit(0);
}
```

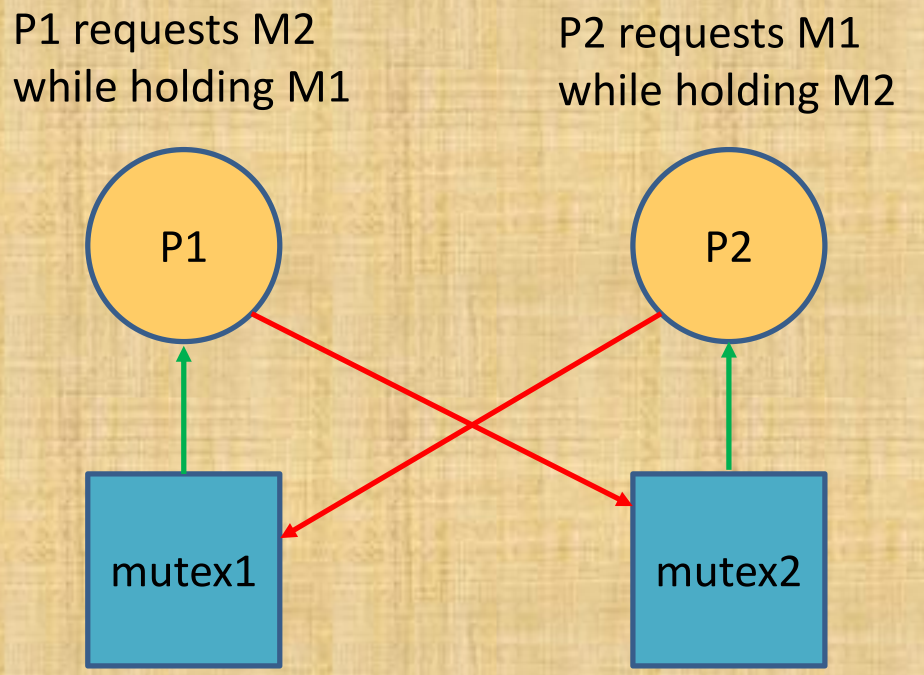
Why "potential"?

The code may not cause deadlock if one thread grabs both locks before the other.

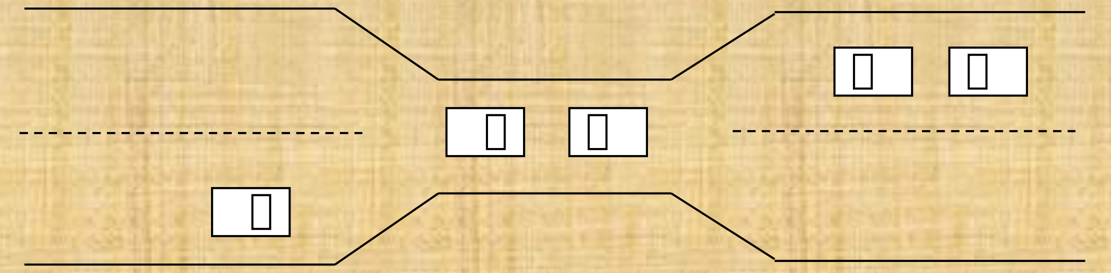
If both threads hold on the one lock before trying the second lock, a deadlock will occur.

http://www.eg.bucknell.edu/~cs315/F2021/meng/code/deadlock/deadlock.c

Deadlock for Two Processes



Deadlock: Bridge Crossing Example



- Traffic only in one direction at a time.
- 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.

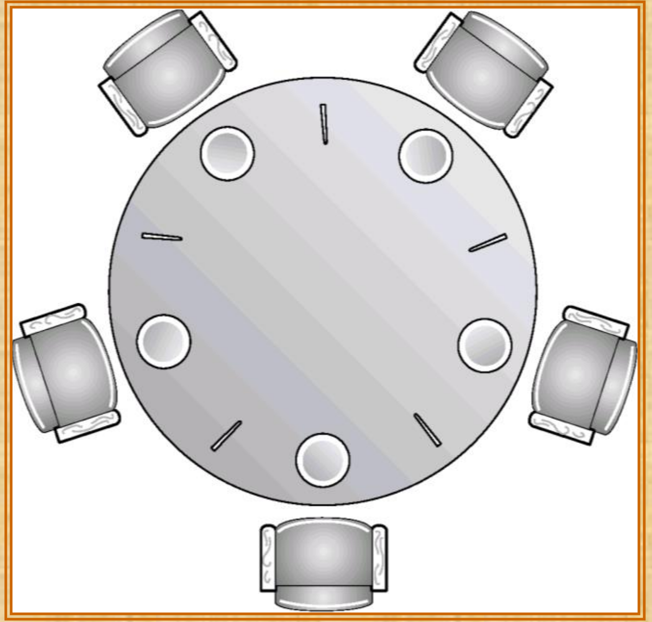
Deadlock: Dining-Philosophers Example

All philosophers start out **hungry** and that they all pick up their left chopstick at the same time.

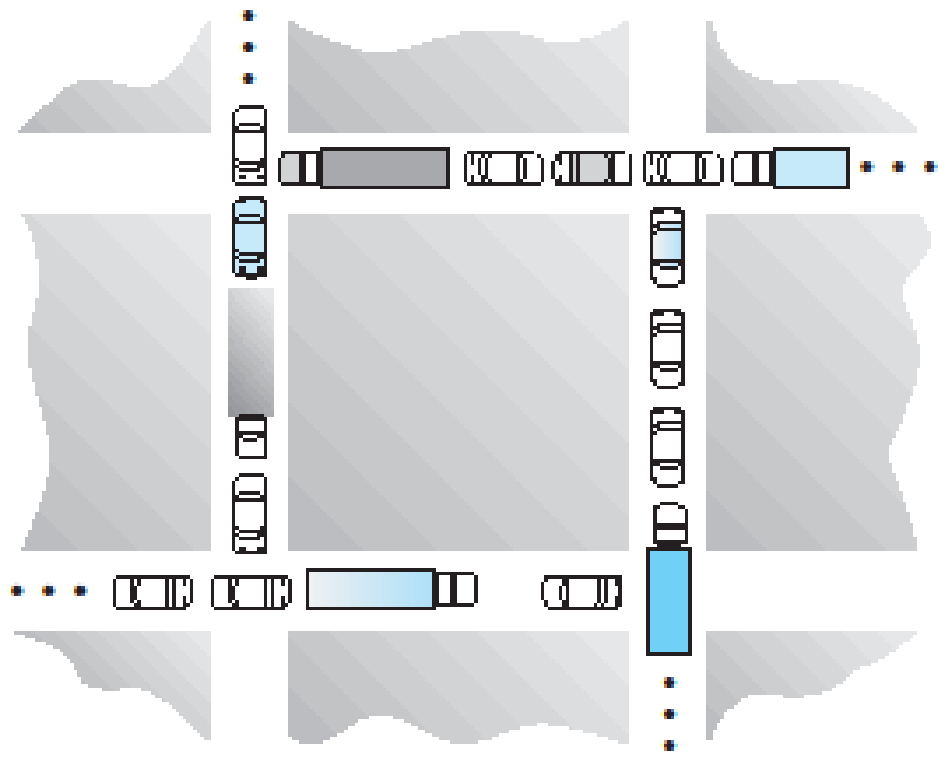
When a philosopher manages to get a chopstick, it is not released until a second chopstick is acquired and the philosopher has eaten his share.

Question: Why did deadlock happen? Enumerate all the conditions that have to be satisfied for deadlock to occur.

Question: What can be done to guarantee that deadlock won't happen?



Traffic Deadlock



Concepts to discuss

- Deadlock
- Livelock
- Spinlock vs. Blocking

A System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock Characterization

Deadlock can arise if four conditions hold *simultaneously*:

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by

  $P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

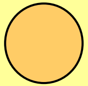
Resource Allocation Graph

Graph: G=(V,E)

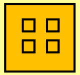
- The nodes in V can be of two types (partitions):
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- Request edge – directed edge $P_1 \rightarrow R_j$
- Assignment edge – directed edge $R_j \rightarrow P_i$

Resource Allocation Graph

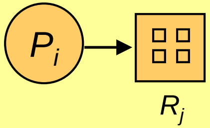
- Process



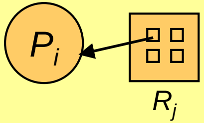
- Resource Type with 4 instances



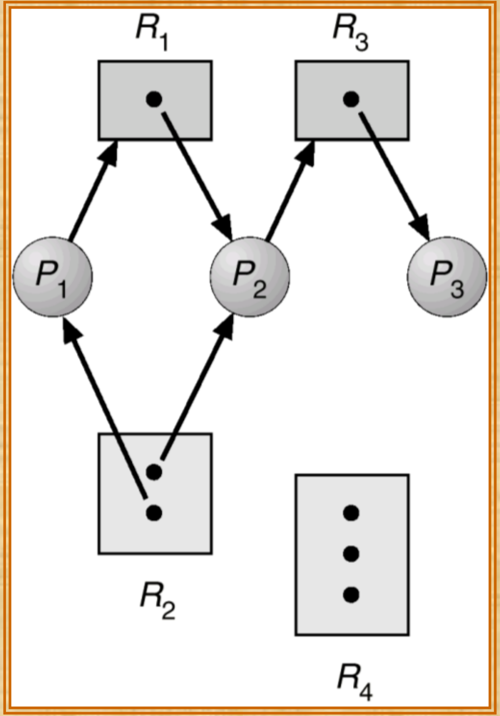
- P_i requests instance of R_j



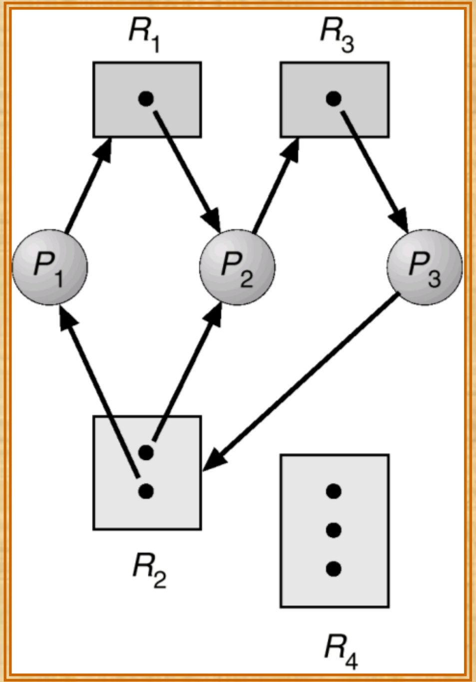
- P_i is holding an instance of R_j



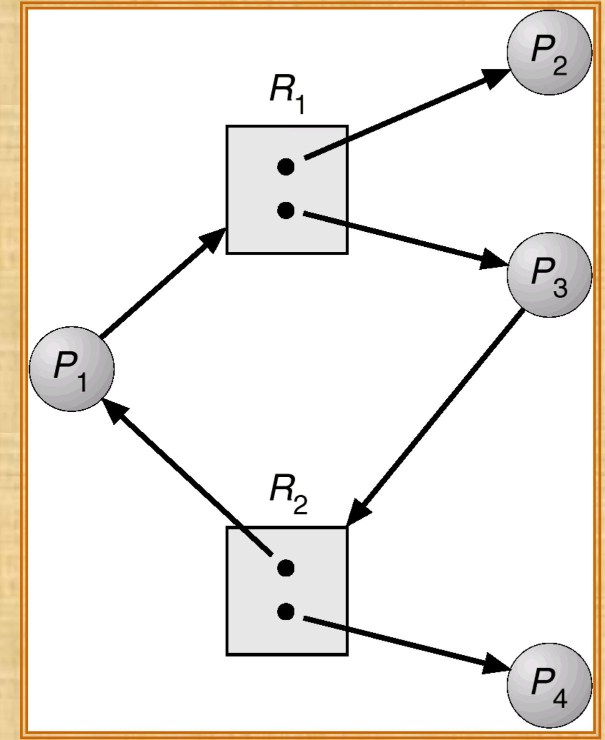
Example of a Resource Allocation Graph



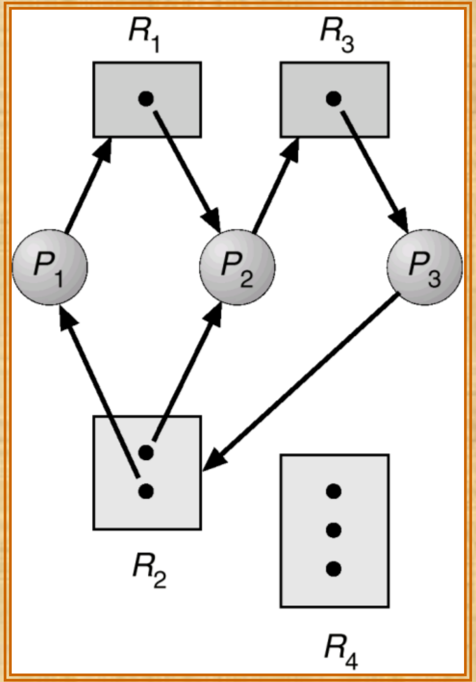
Resource Allocation Graph With A Deadlock



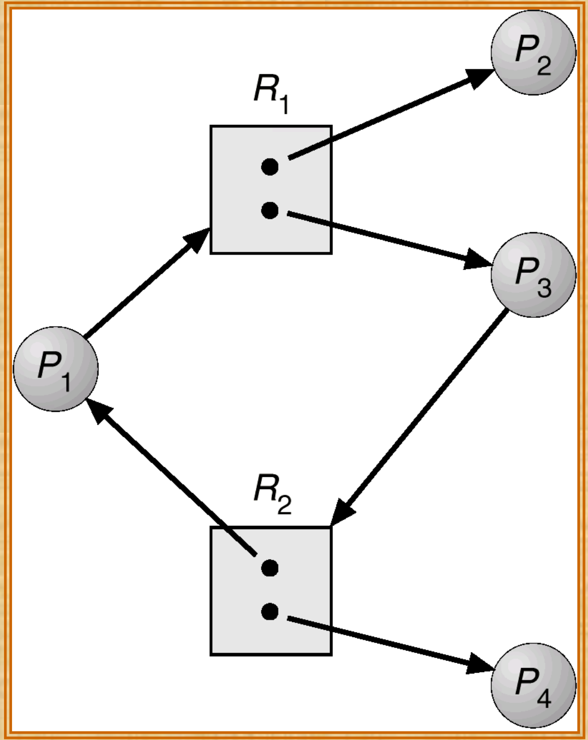
Resource Allocation Graph With A Cycle But No Deadlock



Resource Allocation Graph Example 1



Resource Allocation Graph Example 2



Basic Facts

- **If graph contains no cycles** $\Rightarrow$ no deadlock.

- **If graph contains a cycle** $\Rightarrow$
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.