Deadlock- 1

CS3600 Spring 2022

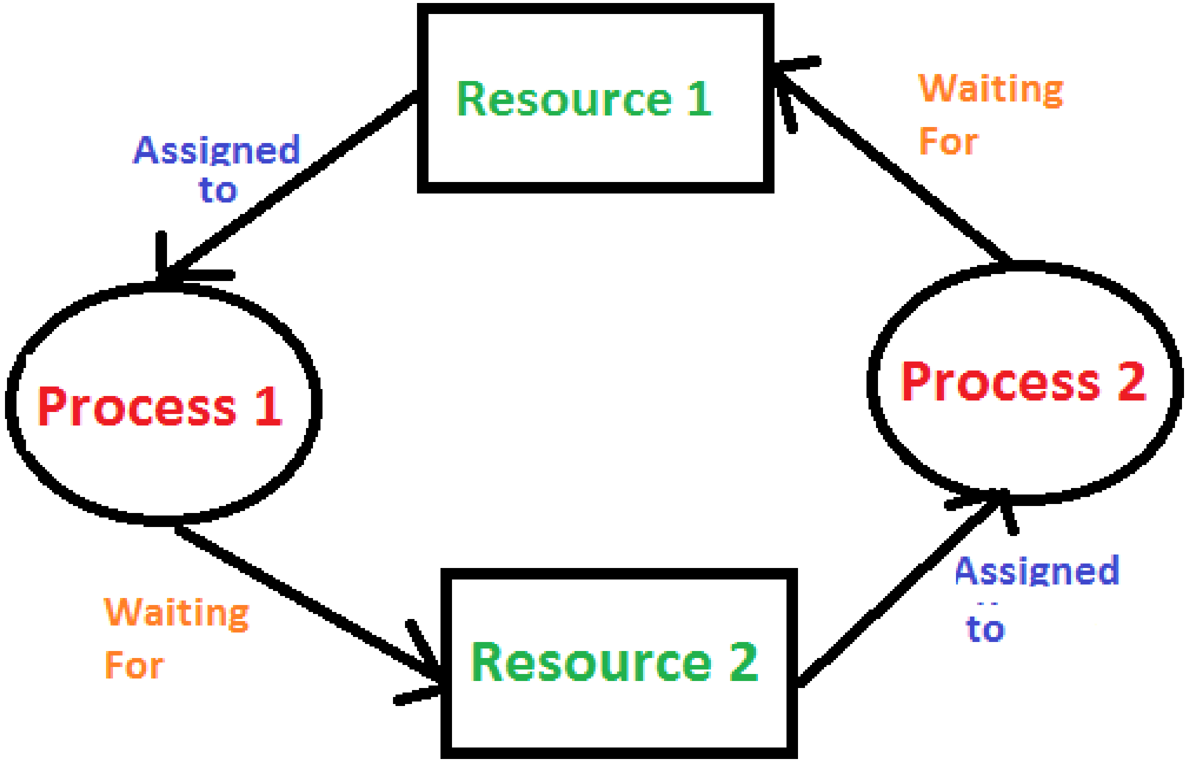
Deadlock

- A process is ***deadlocked*** in a state s if the process is blocked in s and if no matter what state transitions occur in the future, the process remains blocked.

- A state s is called a ***deadlock state*** if s contains two or more deadlocked processes.

- A state s is a ***safe state*** if no sequence of state transitions exists that would lead from s to a deadlock state.

Deadlock


Resource 1
Assigned to
Waiting For
Process 1
Process 2
Waiting For
Resource 2
Assigned to

Example for Deadlock

```
void *do_work_one(void *param) //  thread_one runs in this function
{
  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);
}
 void *do_work_two(void *param) //  thread_two runs in this function
{
  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);
}
```

Modelling using Directed Graph

- A set of vertices V and a set of edges E.

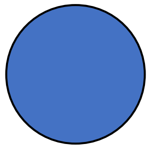
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.

- request edge – directed edge $P_1 \rightarrow R_j$

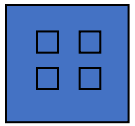
- assignment edge – directed edge $R_j \rightarrow P_i$

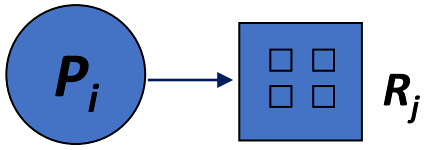
Resource Allocation Graph

- Process

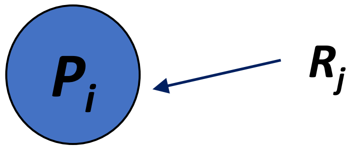


- Resource Type with 4 instances



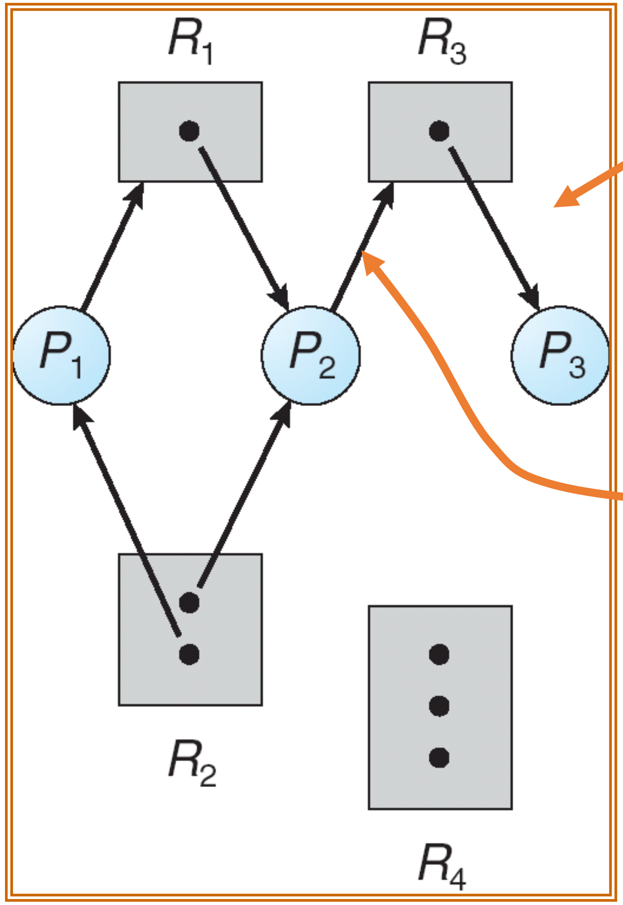


- P_i requests instance of R_j



- P_i is holding an instance of R_j

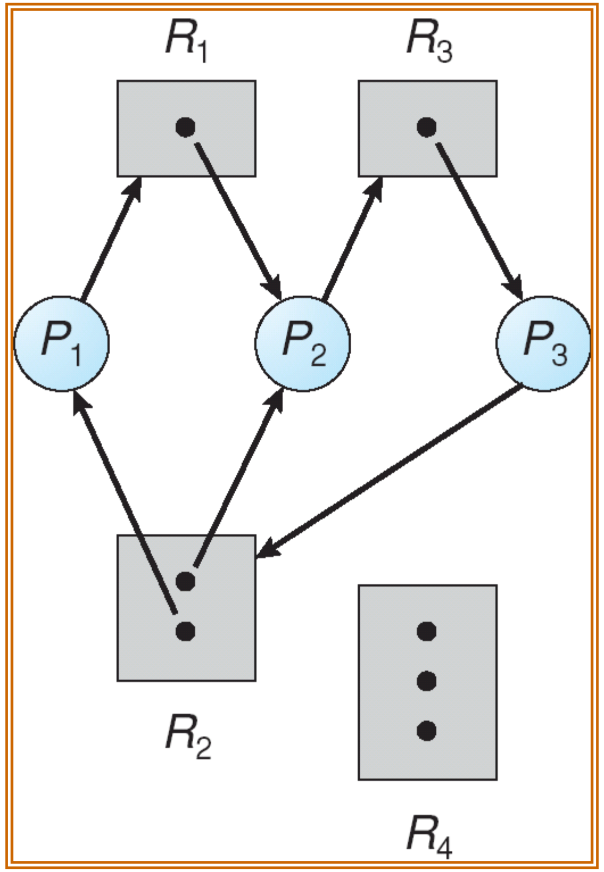
Example of a Resource Allocation Graph



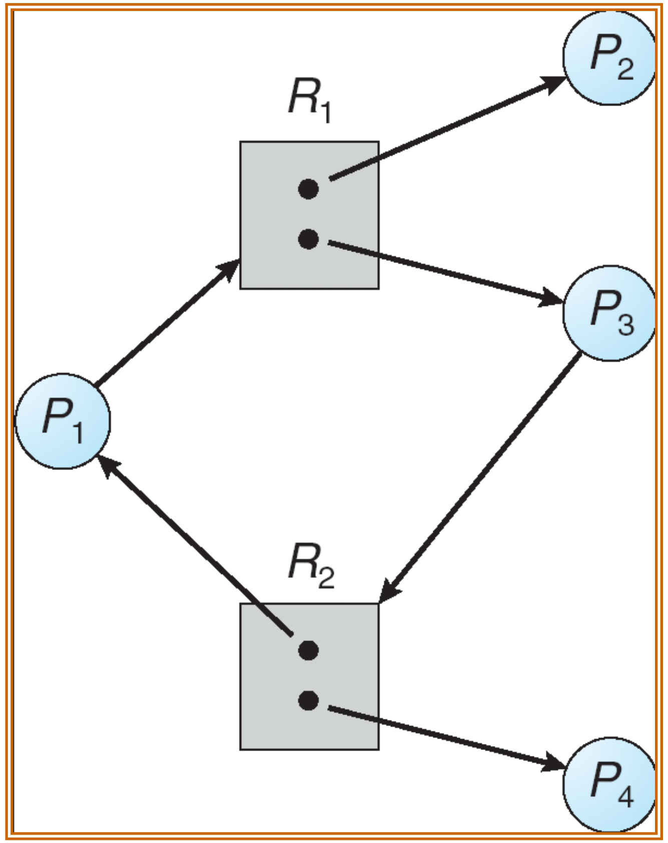
R3 Assigned to P3

P2 Requests P3

Resource Allocation Graph With A Deadlock



Check for deadlock ????



Conditions for Deadlock

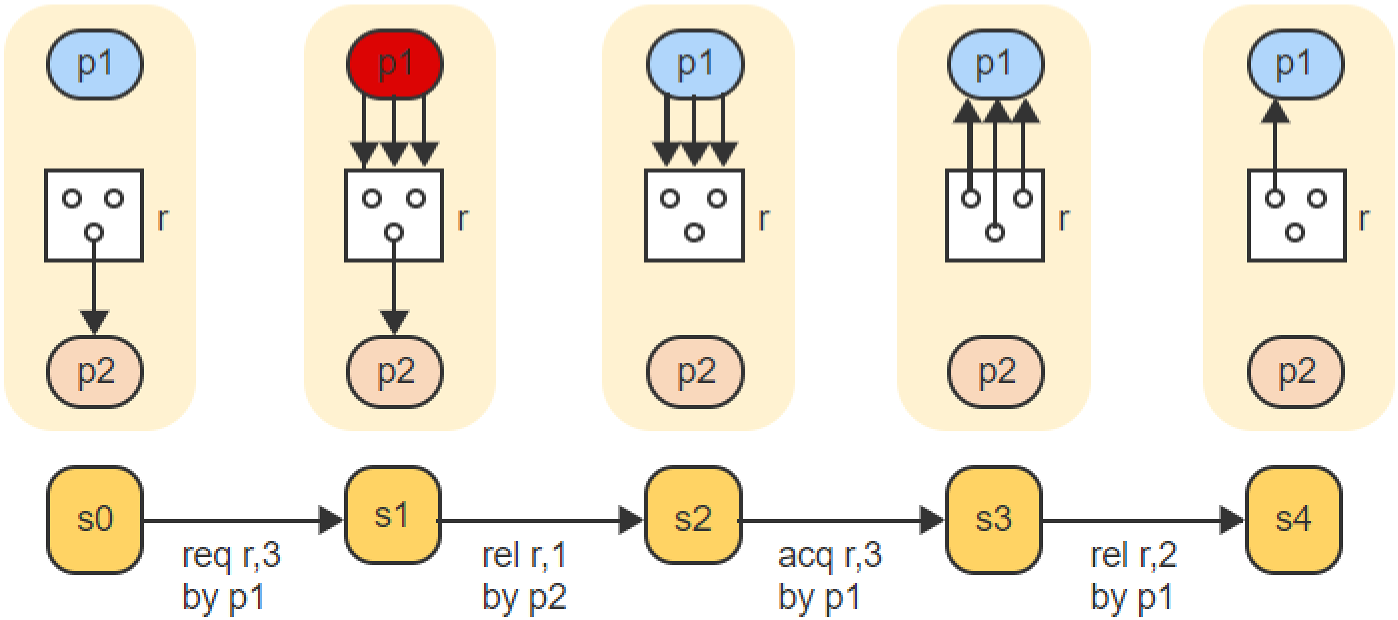
- If graph contains no cycles $\Rightarrow$ no deadlock.

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

State transitions

- A ***resource request*** (req r, m) by a process p for m units of a resource r creates m new edges directed from p to r.
- A ***resource acquisition*** (acq r, m) by a process p of m units of a resource r reverses the direction of the corresponding request edges to point from the units of r to p.
- A ***resource release*** (rel r, m) operation by a process p of m units of a resource r deletes m allocation edges between p and r.

State transitions



Deadlock states and safe states

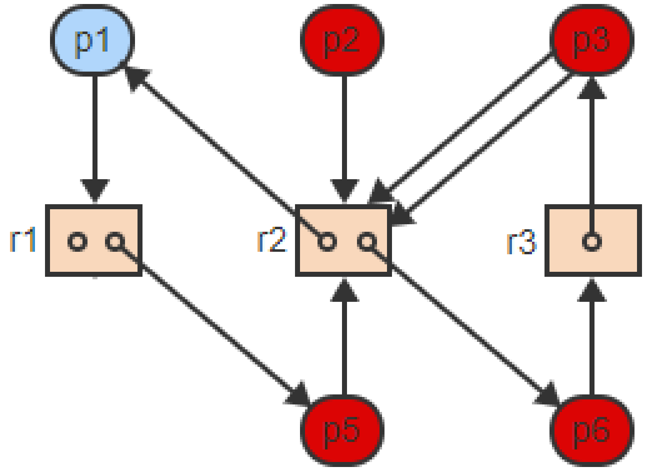
- A process is ***deadlocked*** in a state s if the process is blocked in s and if no matter what state transitions occur in the future, the process remains blocked.

- A state s is called a ***deadlock state*** if s contains two or more deadlocked processes.

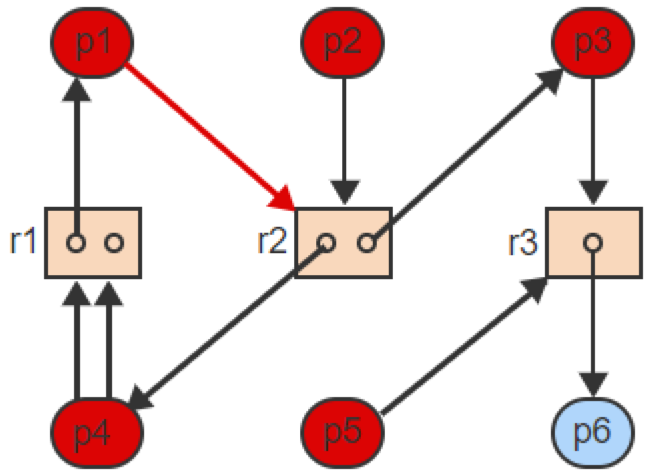
- A state s is a ***safe state*** if no sequence of state transitions exists that would lead from s to a deadlock state.

Graph Reduction Detection

- Repeat until no unblocked process remains in the graph:
  - Select an unblocked process p.
  - Remove p, including all request and allocation edges connected to p



Graph Reduction Detection



Classwork

- Complete Worksheet 05

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.

- Allow the system to enter a deadlock state and then recover.

- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Do not use the 3rd method for solutions in your exam questions

To Do as of 03/15/2022

- Complete Worksheet 05.
- Weekly Quiz
- Next Class: Resource allocation Algorithms