Tishk International University
Science Faculty
IT Department

Operating Systems

Lecture 6 Deadlock

3rd Grade - Fall Semester

Instructor: Alaa Ghazi

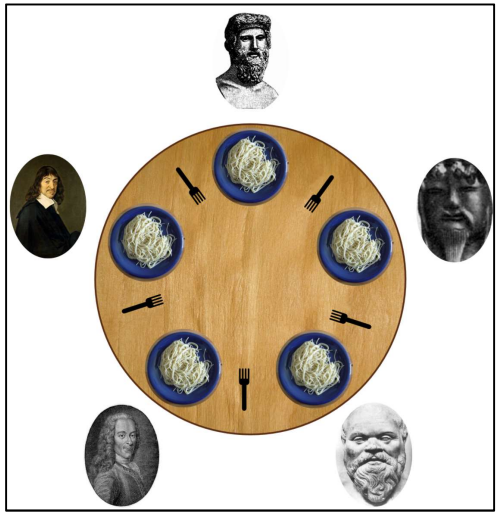
Lecture 6: Deadlock Agenda

- What is a Deadlock?
- Deadlock Conditions
- Ways for Handling Deadlocks
- Resource Allocation Graph
- What is a Livelock?
- What is a Zombie Process?

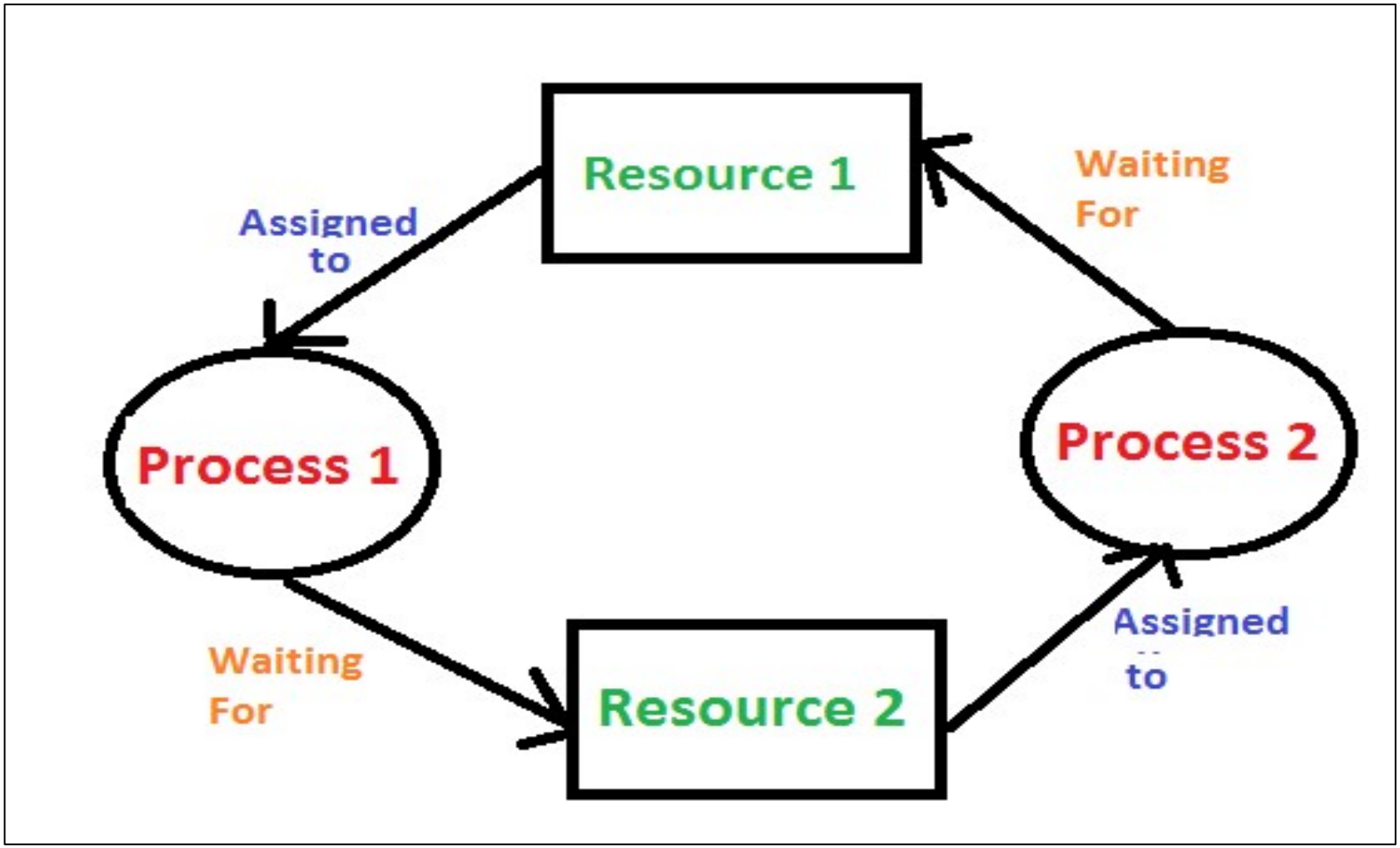
What is a Deadlock?

Deadlock is when a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.

If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes.



Basic Deadlock Diagram



Deadlocks Conditions

Deadlocks four conditions are required simultaneously to cause deadlock, so deadlock can be avoided by avoiding at least one of the conditions.

Deadlock occurs = Cond1 AND Cond2 AND Cond3 AND Cond4

1. **Mutual Exclusion:** only one process at a time can use a resource.
2. **Hold and Wait:** a process holding resource is waiting to acquire additional resources held by other processes
3. **No Preemption (No Interruption)** a resource can be released only by the process holding it upon its task completion.
4. **Circular Wait** it is the case when the processes are waiting circularly (infinitely). The processes are waiting circularly and forming a cycle which never breaks.

Ways for Handling Deadlocks

Generally speaking there are three ways of handling deadlocks:

- **Deadlock Prevention** - Do not allow the system to get into a deadlocked state. Prevention algorithms are difficult to implement.
- **Deadlock detection and recovery** - Abort a process or preempt some resources when deadlocks are detected.
- **Ignore deadlock** - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. **This is the approach that both Windows and UNIX take.**

Resource Allocation Graph (RAG)

• Deadlock can be described through a resource allocation graph.

• The RAG consists of a set of vertices

P={P1,P2 ,…,P n} of processes and

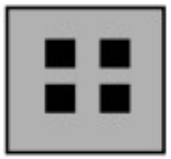
R={R1,R2,…,Rm} of resources.

• A directed edge from a processes to a resource, Pi->R j, implies that Pi has requested Rj.

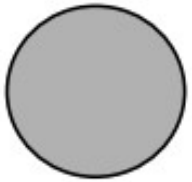
• A directed edge from a resource to a process, Rj->Pi, implies that Rj has been allocated by Pi.

• If the graph has no cycles, deadlock cannot exist. If the graph has a cycle, deadlock may exist.

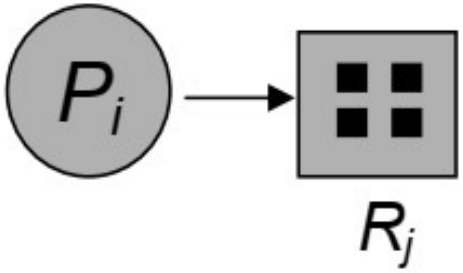
Resource Allocation Graph (RAG)



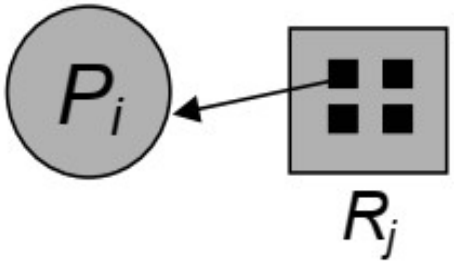
R vertex (a resource type with 4 instances)



P vertex (a process)

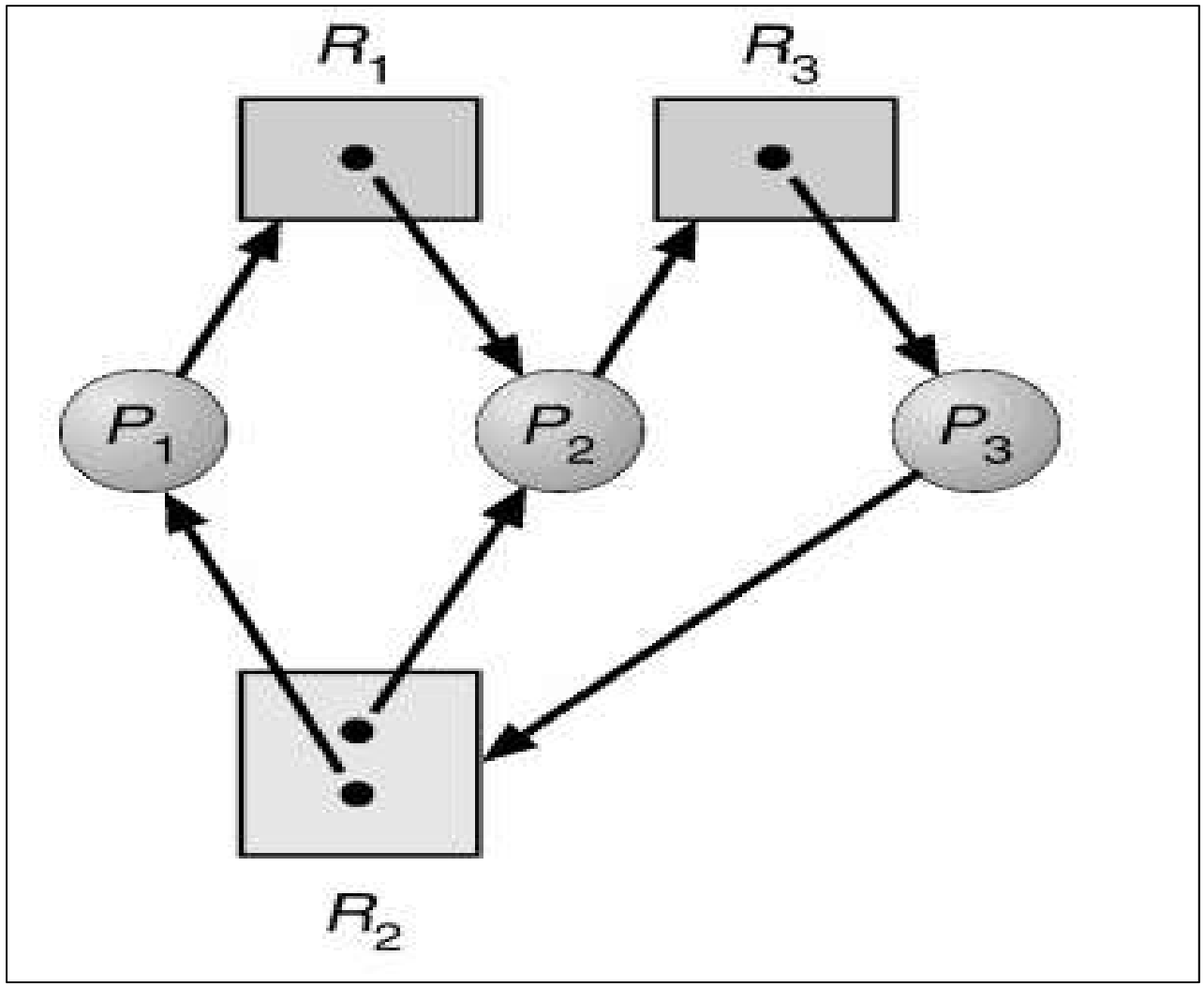


Request edge: P_i requests instance of R_j

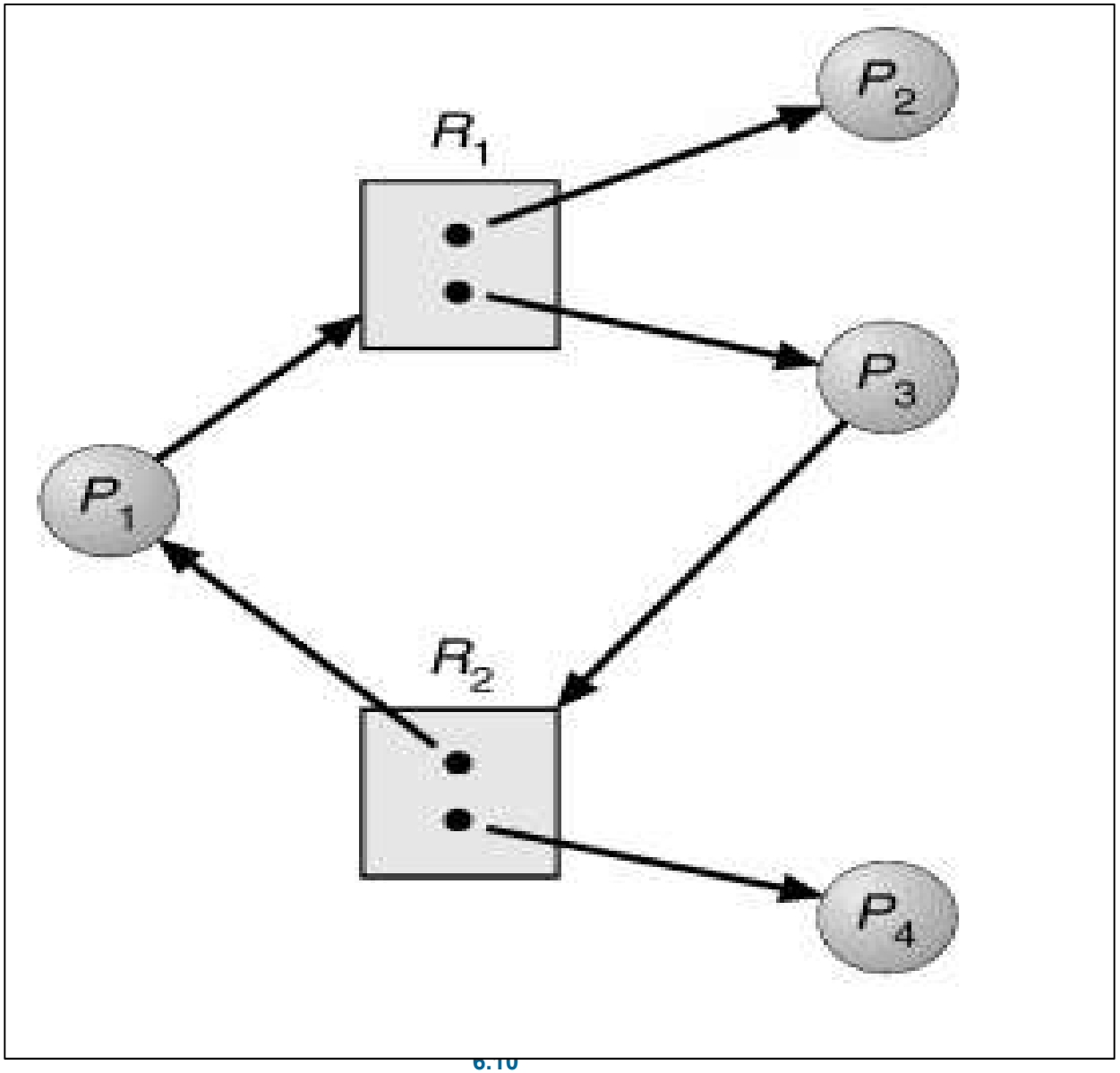


Assignment edge: P_i is holding an instance of R_j

RAG with cycle and deadlock



RAG with a cycle but no deadlock



RAG –Example1

In the example below, answer below questions:

Q1\ Which resources are assigned to p1?

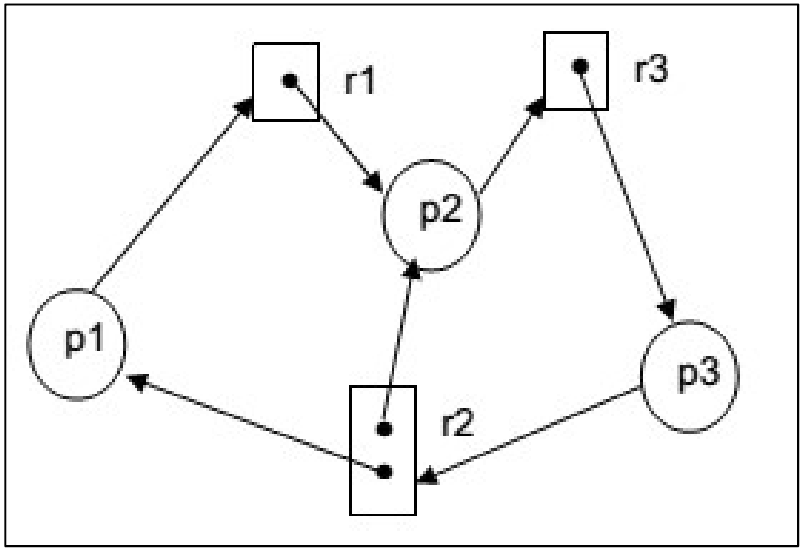
A1\ r2

Q2\ Which resources are requested by p1?

A2\ r1

Q3\ Which processes are using resource r1?

A3\ p2



RAG –Example2

In the example below, answer below questions:

Q1\ Which resources are assigned to P4?

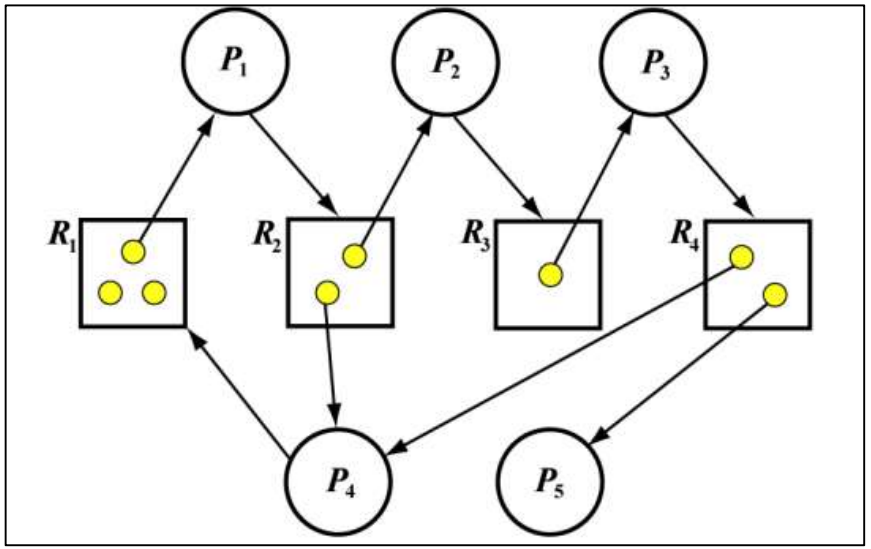
A1\ R2 and R4

Q2\ Which resources are requested by P3?

A2\ R4

Q3\ Which processes are using resource R4?

A3\ P4 and P5



What is a Livelock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work. This is similar to deadlock in that no progress is made but differs in that neither process is blocked or waiting for anything.

A human example of livelock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

What is a Zombie Process?

A **zombie process** is the process that has been completed, but its PID and process entry remains in the Linux process table.

Zombie processes don't use up any system resources. (Actually, each one uses a very tiny amount of system memory to store its process descriptor.) However, each zombie process retains its process ID (PID). Linux systems have a finite number of process IDs – 32767 by default on 32-bit systems. If zombies are accumulating at a very quick rate (for example, if improperly programmed server software is creating zombie processes under load) the entire pool of available PIDs will eventually become assigned to zombie processes, preventing other processes from launching.