Amirkabir University of Technology
(Tehran Polytechnic)

Operating Systems

Deadlocks-Part1

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Spring 2021

Outline

- Liveness
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance

Chapter Objectives

- Illustrate how deadlock can occur when mutex locks are used
- Define the four necessary conditions that characterize deadlock
- Identify a deadlock situation in a resource allocation graph
- Evaluate the four different approaches for preventing deadlocks
- Apply the banker's algorithm for deadlock avoidance

مهم و امتحانی

Liveness

- Processes may have to wait indefinitely while trying to acquire a synchronization tool such as a mutex lock or semaphore.

- Waiting indefinitely violates the progress and bounded-waiting criteria.

Liveness

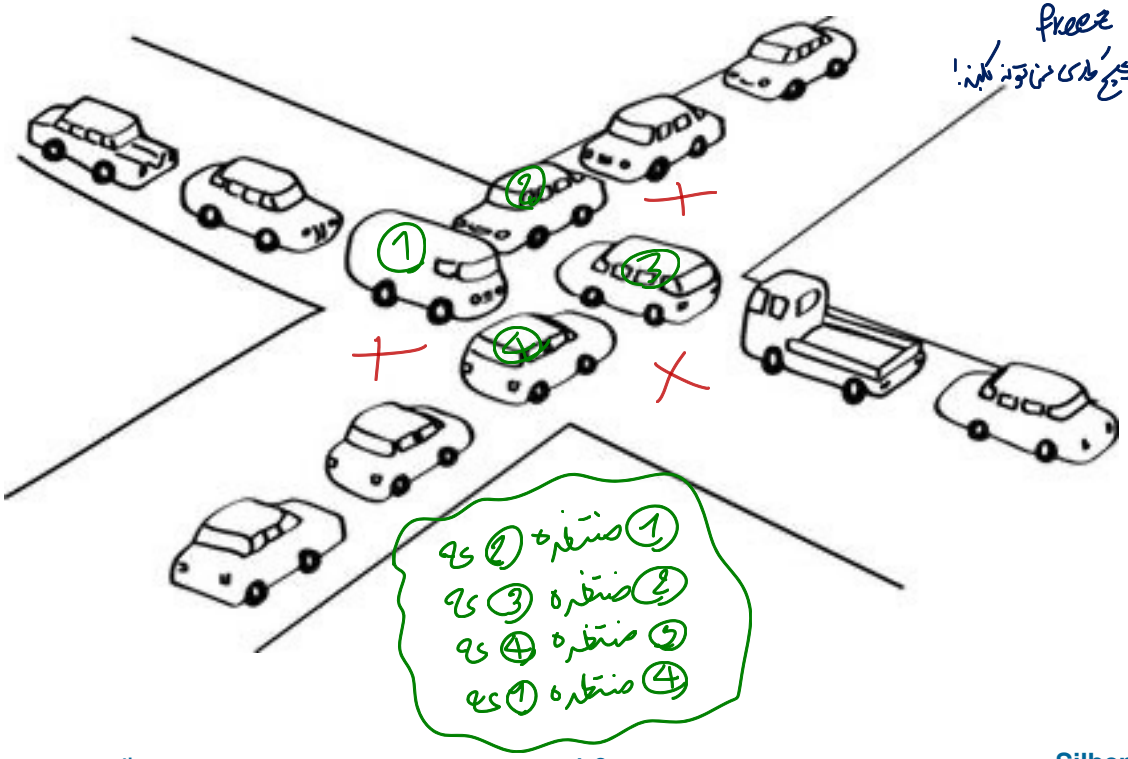
- **Liveness** refers to a set of properties that a system must satisfy to ensure processes make progress.

- **Indefinite waiting** is an example of a **liveness failure**.

Deadlock

* two or more processes are **waiting indefinitely** for an event

that can be caused **by only one of the waiting processes**.



Liveness (cont.)

- Let S and Q be two semaphores initialized to **1**

P_0	P_1
`wait(S);`	`wait(Q);`
`wait(Q);`	`wait(S);`
...	...
`signal(S);`	`signal(Q);`
`signal(Q);`	`signal(S);`

P_0 must wait until P_1 executes signal(Q).

P_1 is waiting until P_0 execute signal(S)

Since these signal() operations will never be executed, P_0 and P_1 are **deadlocked**.

Other Forms of Deadlock

- **Starvation** – indefinite blocking
  - A process may never be removed from the semaphore queue in which it is suspended.

- **Priority Inversion** – Scheduling problem when lower-priority process holds a lock needed by higher-priority process.
  - We do not cover this.

System Model

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$ این منابع من توانن از نوع های زیر باشند؟
  - *CPU cycles, memory space, I/O devices*
- Each resource type R_i has W_i instances. (صفوا زیاد کارت شبکه داریم!)
- Each process utilizes a resource as follows:

1. **request**
2. **use**
3. **release**

Deadlock with Semaphores

- Data:
  - A semaphore **S1** initialized to 1
  - A semaphore **S2** initialized to 1
- Two processes P1 and P2
- **P1:**

```
wait(s1)
wait(s2)
```

→ اینجا گیر می‌کند و منتظر s2 می‌ماند

- **P2:**

```
wait(s2)
wait(s1)
```

→ اینجا گیر می‌کند و منتظر s1 می‌ماند

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

1. **Mutual exclusion**
2. **Hold and wait**
3. **No preemption**
4. **Circular wait**

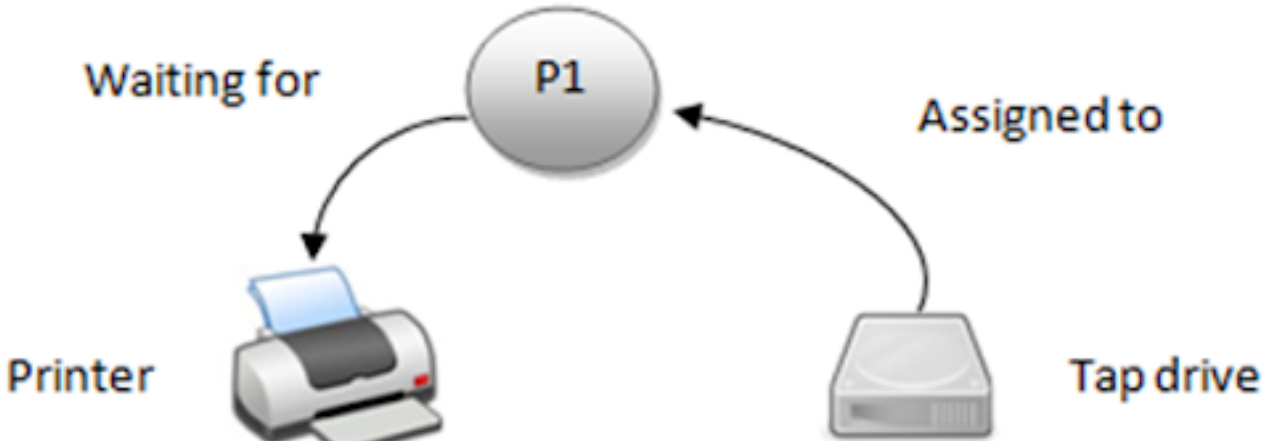
1-Mutual Exclusion

- Only one process at a time can use a resource.

* چون اگر لازم نبود که یکی مستقل بدونه که دیگری کاری نکنه اصلاً deadlock رخ نمی داد ! ت

2-Hold and Wait

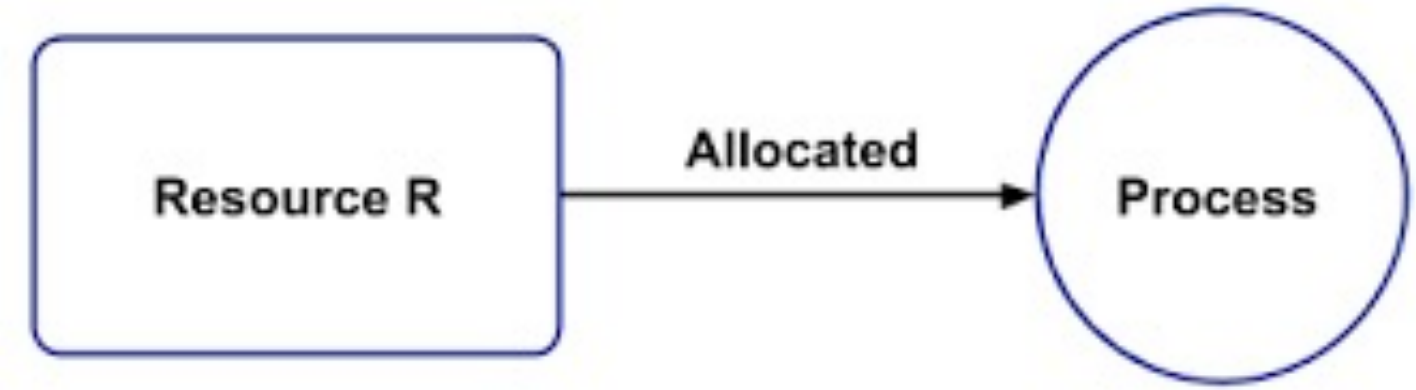
- A process holding at least one resource is waiting to acquire additional resources held by other processes.



3-No Preemption

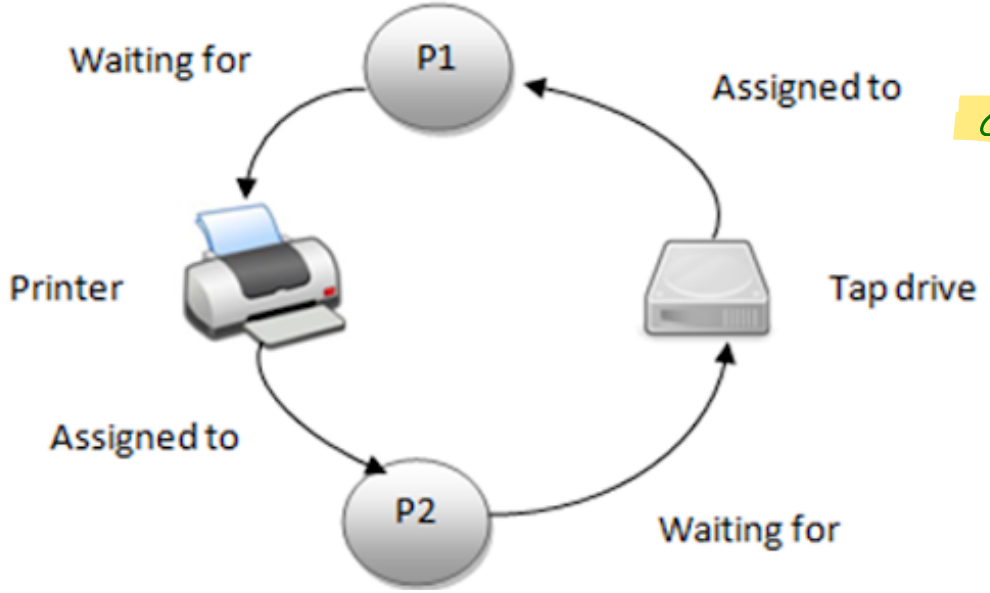
- A resource can be released only ***voluntarily*** by the process holding it, after that process has completed its task.

یعنی وقتی یک منبع رو دادیم به یک پردازه تا زمانی که کارش با این منبع تموم نشده ازش نمی‌گیریم!



4-Circular Wait

- There exists a set $\{P_0, P_1, \ldots, P_n\}$ 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, …,
  - 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

→ یک گراف بدیم و بر اساس اون تحلیل انجام بدیم !

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.

. . . printers , locks

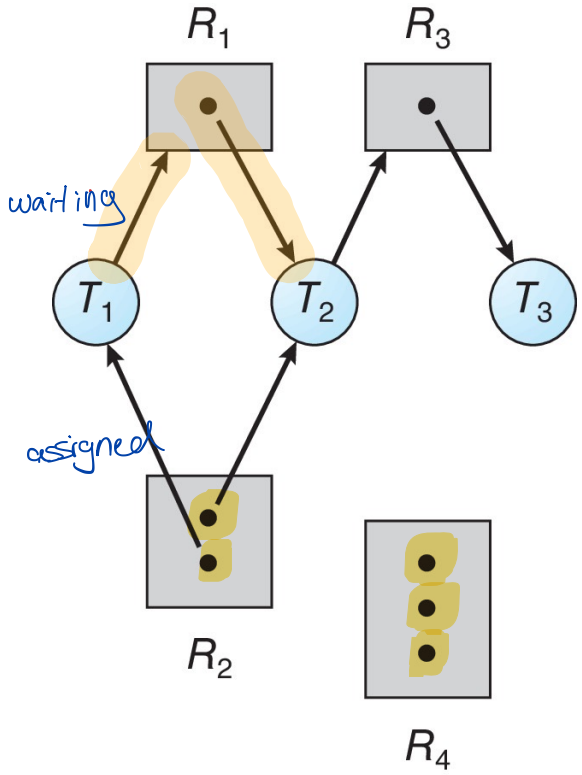
Resource-Allocation Graph

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

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

Resource Allocation Graph Example

- One instance of R1
- Two instances of R2
- One instance of R3
- Three instance of R4

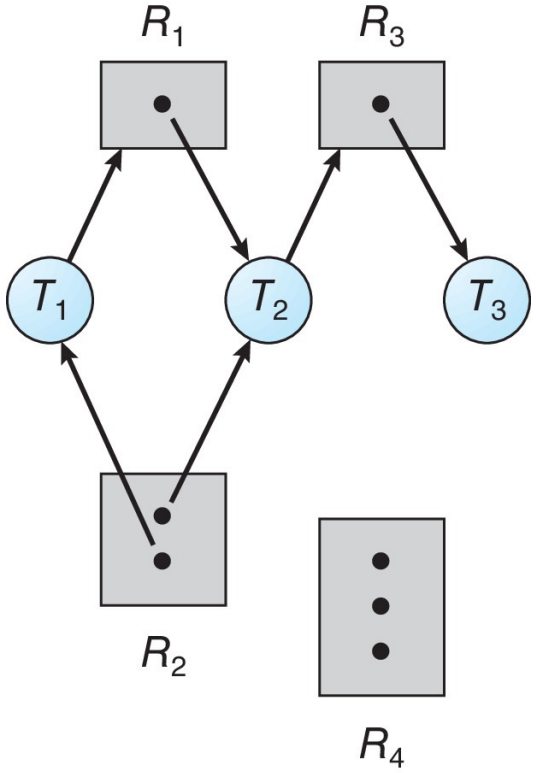


Resource Allocation Graph Example

- T1 holds one instance of R2 and is waiting for an instance of R1

- T2 holds one instance of R1, one instance of R2, and is waiting for an instance of R3.

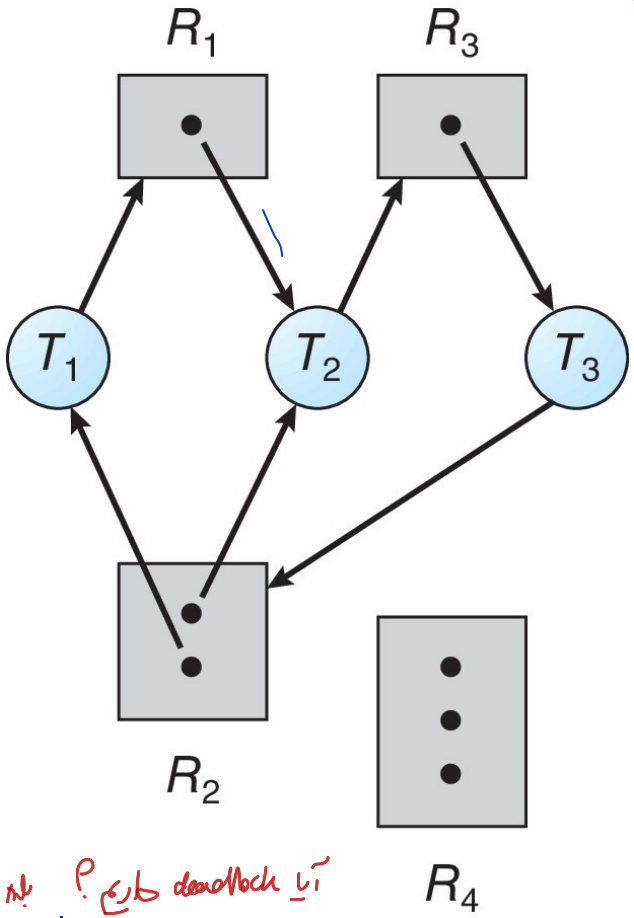
- T3 is holds one instance of R3



waiting ✗

✓ Mutual exclusion
✗ Hold and wait
✓ No preemte
✗ circular

Resource Allocation Graph with a Deadlock



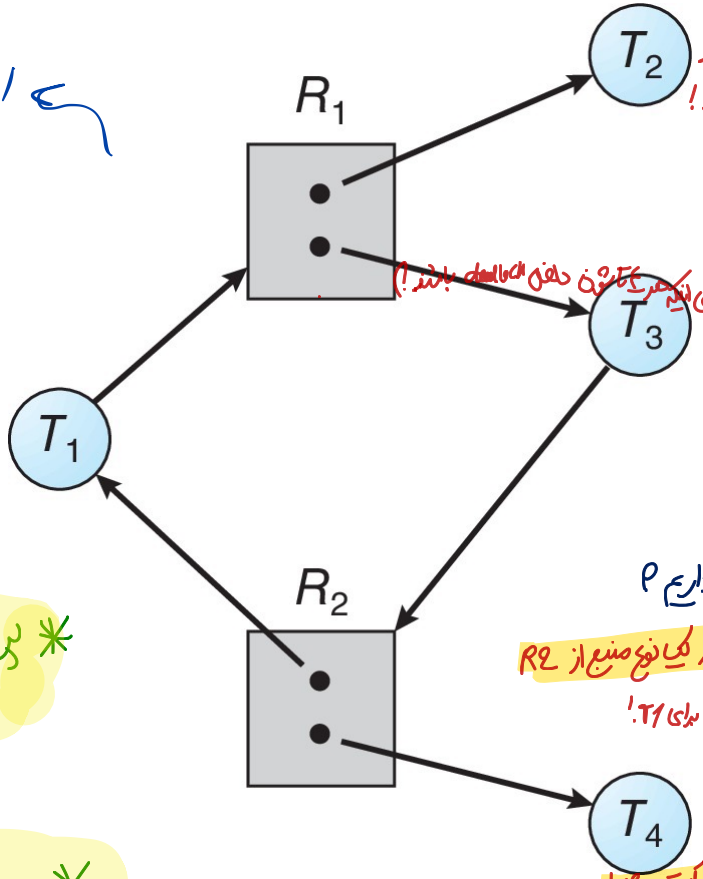
$T_1 \rightarrow R_1 \rightarrow T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$

$T_2 \rightarrow R_3 \rightarrow T_3 \rightarrow R_2 \rightarrow T_2$

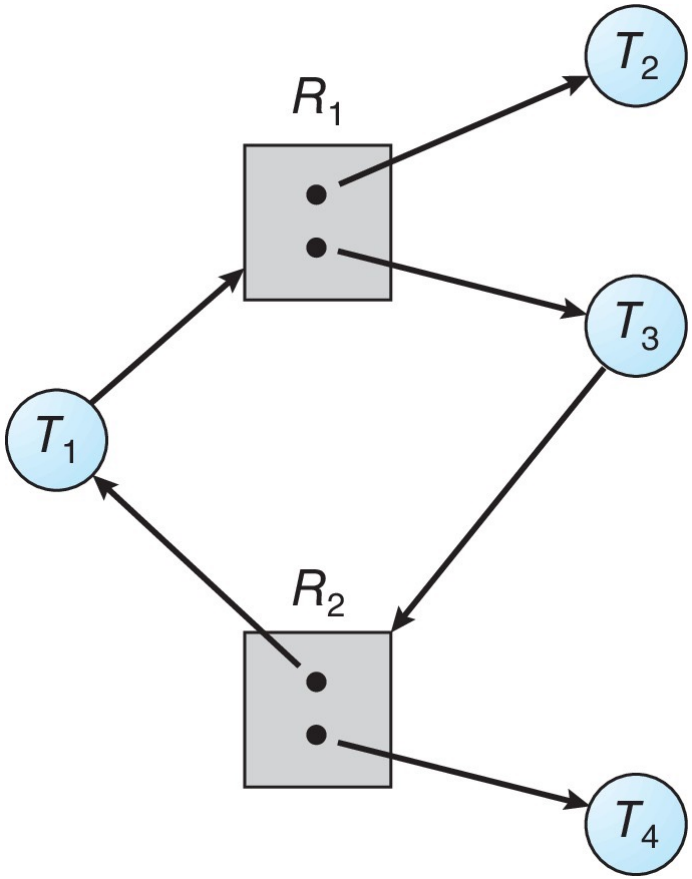
Threads T1, T2, and T3 are deadlocked.

- Thread $\mathbf{T_2}$ is waiting for the resource $\mathbf{R_3}$, which is held by thread $\mathbf{T_3}$.
- Thread $\mathbf{T_3}$ is waiting for either thread $\mathbf{T_1}$ or thread $\mathbf{T_2}$ to release resource $\mathbf{R_2}$.
- In addition, thread T1 is waiting for thread $\mathbf{T_2}$ to release resource $\mathbf{R_1}$.

Is there a Deadlock?



Graph with a Cycle But no Deadlock



$$T_1 \rightarrow R_1 \rightarrow T_3 \rightarrow R_2 \rightarrow T_1$$

There is no deadlock. Observe that thread T4 **may release** its instance of resource type R2. That resource can then be allocated to T3, breaking the cycle.

* این وجود یک حلقه در گراف دلیلی بر وجود circular wait نیست! (تستی *)

Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, **then deadlock**
  - if several instances per resource type, **possibility of deadlock**

if the graph contains **no cycles** → **no thread in the system is deadlocked**

If the graph **does contain a cycle** → **a deadlock may or may not exist.**