CS343: Operating System

Deadlock: Avoidance, Prevention, Detection and Recovery

Lect24 : 03rd Oct 2023

Dr. A. Sahu

Dept of Comp. Sc. & Engg.

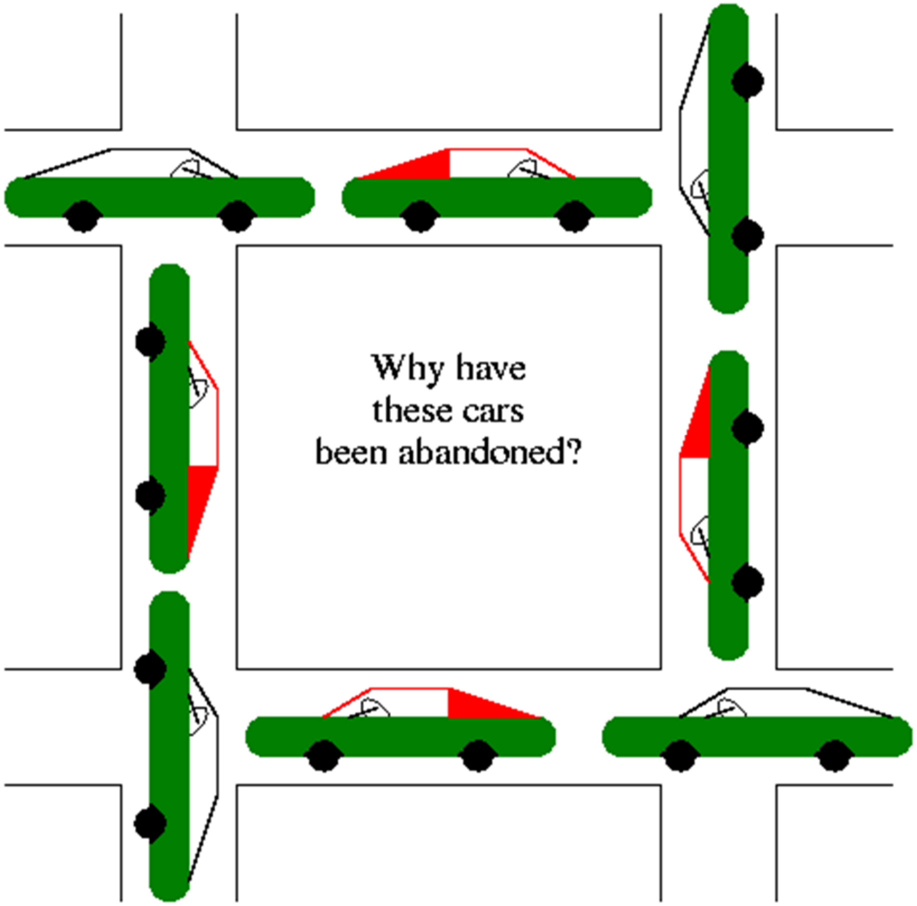
Indian Institute of Technology Guwahati

Outline

- Deadlock
  - Conditions (Why deadlock happens)
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and Recovery

Deadlock

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait



System Model

- System consists of resources
- 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** **// Similar to Lock**
  - **Use** **// Similar to CS**
  - **Release** **// Similar to Unlock**

Deadlock Characterization

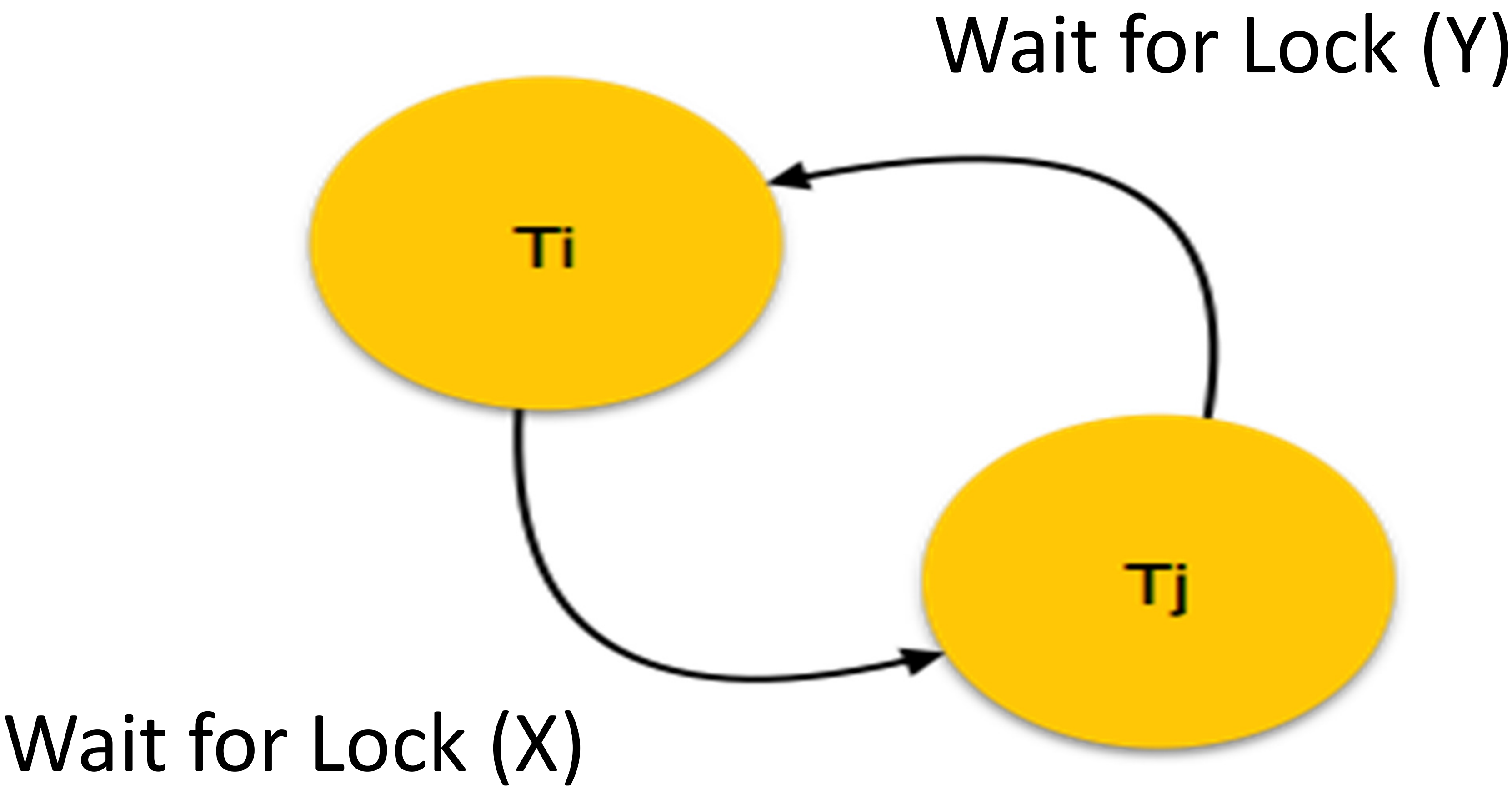
- Deadlock can arise if four conditions hold simultaneously.
- **Mutual exclusion**
  - Only one process at a time can use a resource
- **Hold and wait**
  - A process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption**
- **Circular wait**

Deadlock Characterization

- **Mutual exclusion, Hold and wait**
- **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 $\{P_0, P_1, \ldots, P_n\}$ 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.

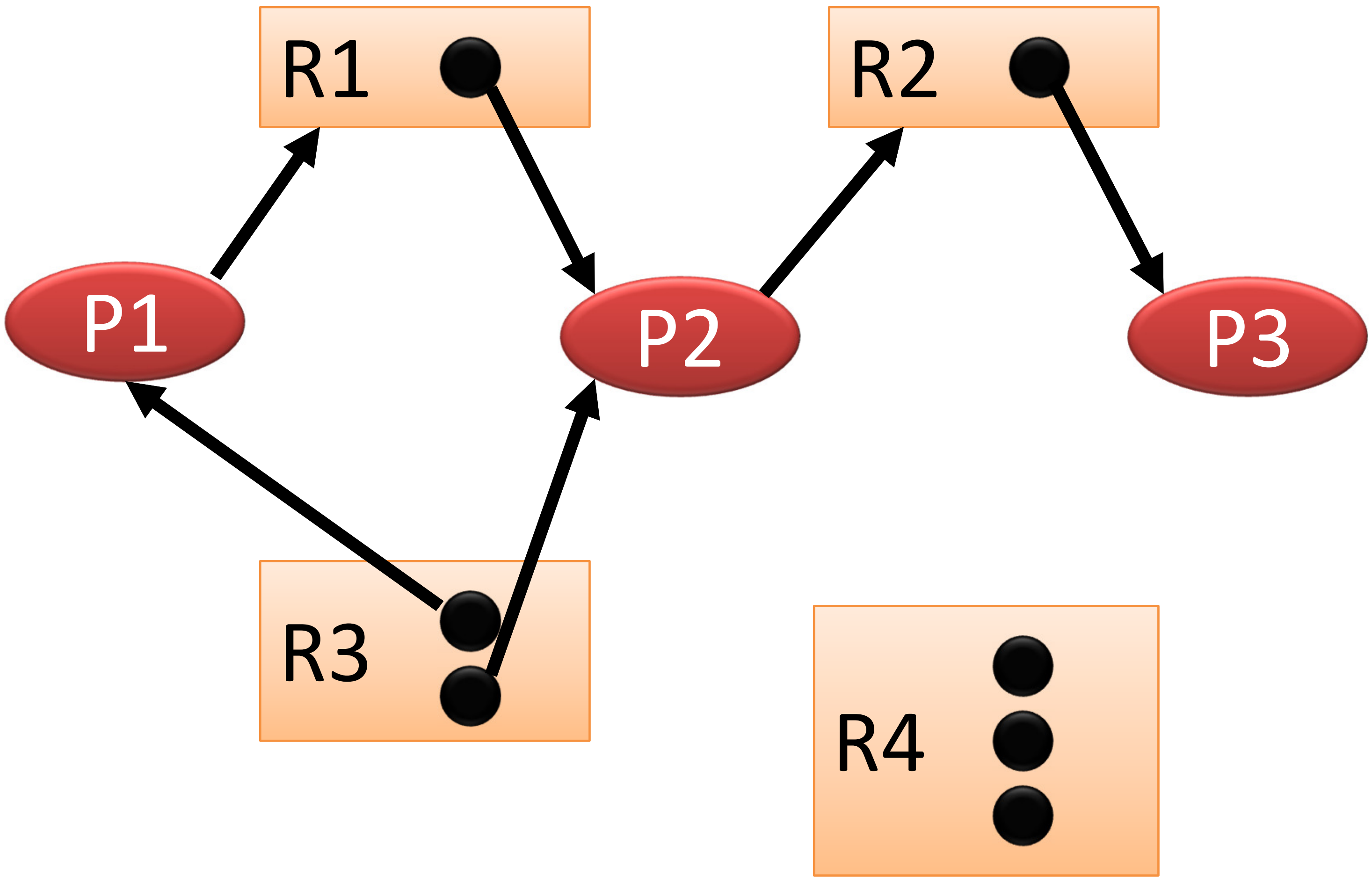
Deadlock with Mutex Locks

- Deadlocks can occur via system calls, locking, etc.



Resource Allocation Graph (RAG)

RAG to Characterize Deadlock



Resource-Allocation Graph

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

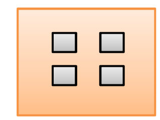
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge

$$R_j \rightarrow P_i$$

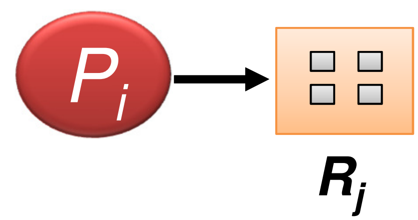
Resource-Allocation Graph (Cont.)

- Process

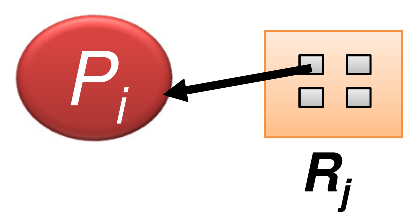
- Resource Type with 4 instances



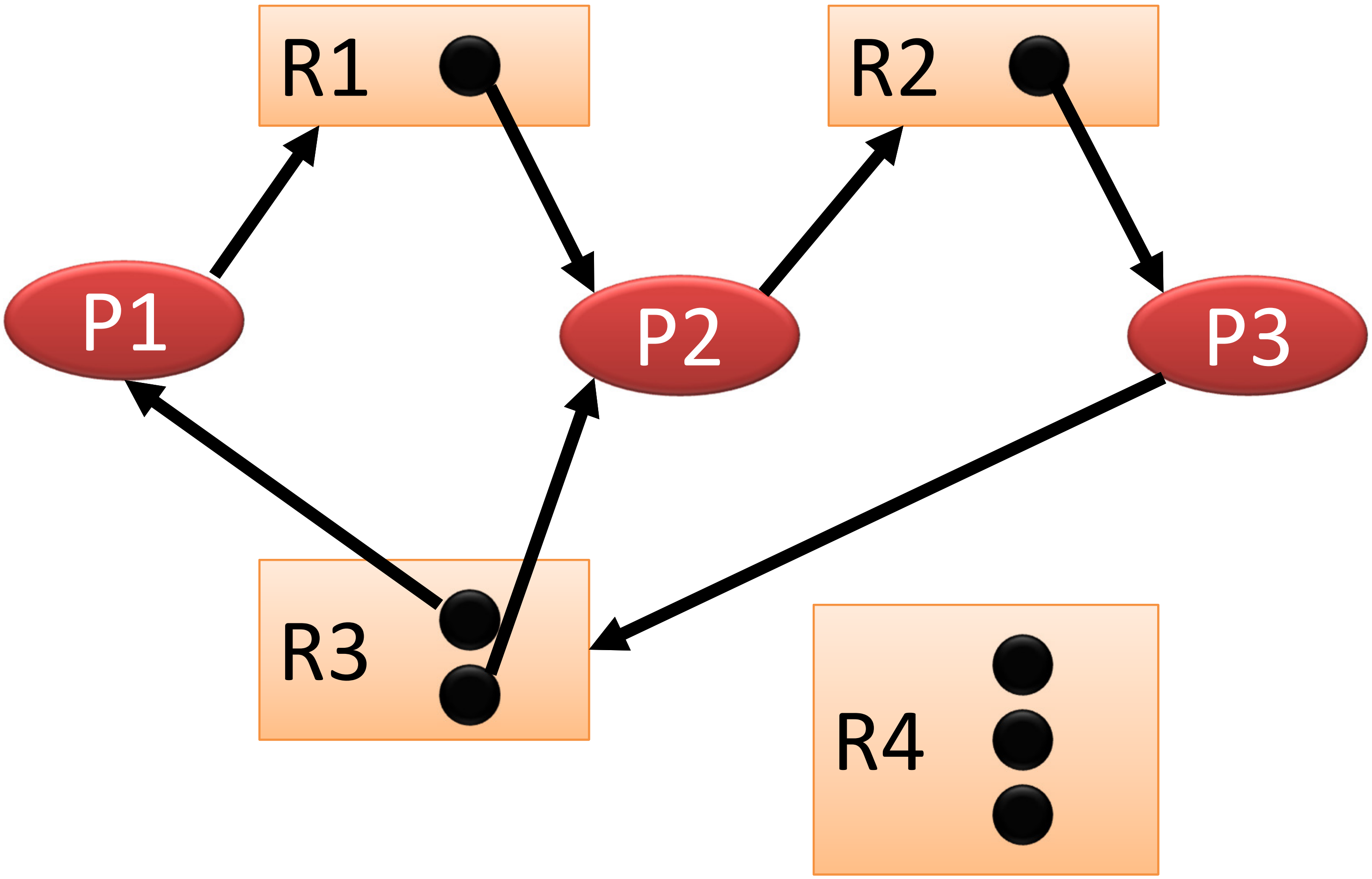
- P_i requests instance of R_j



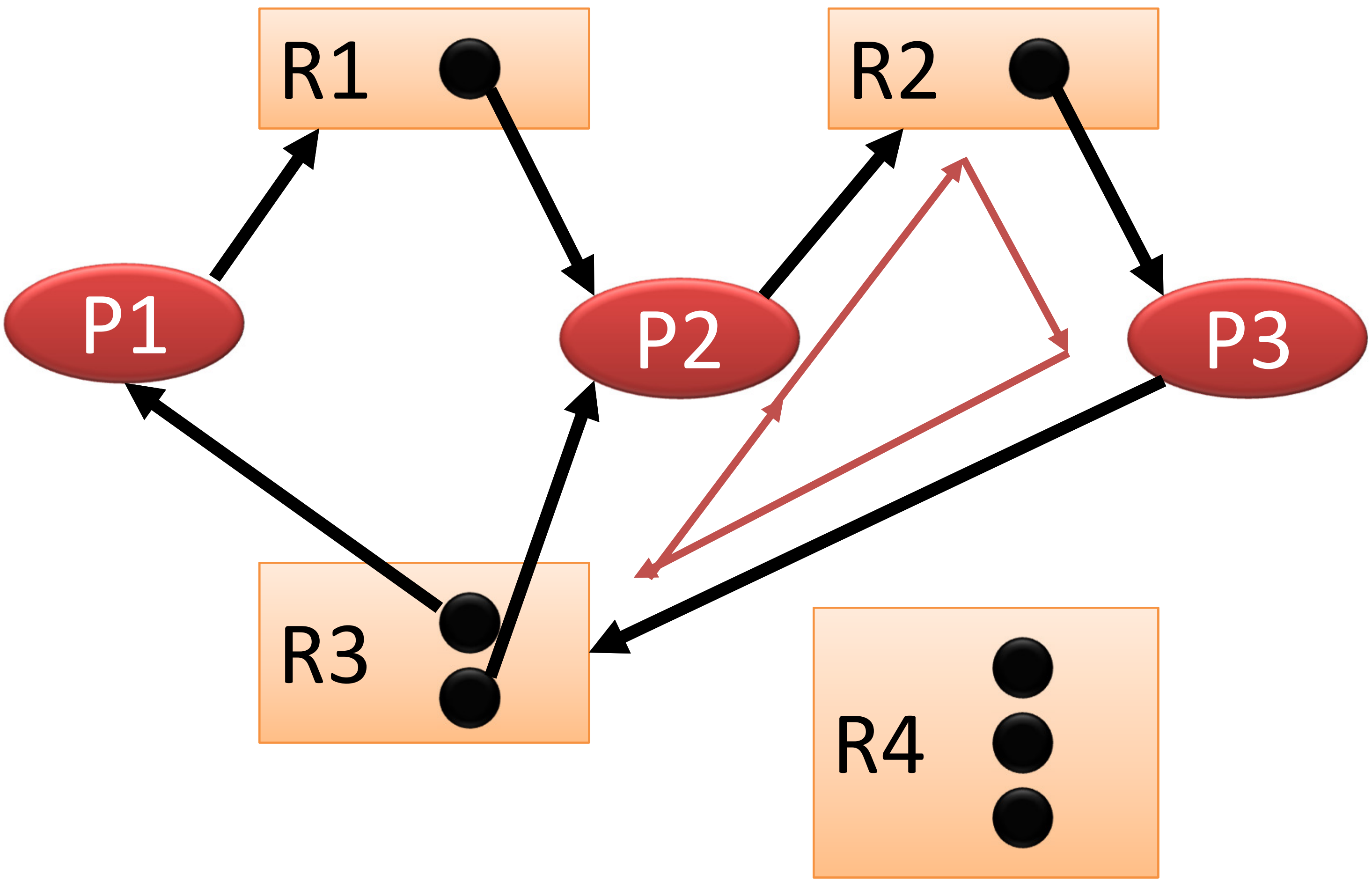
- P_i is holding an instance of R_j



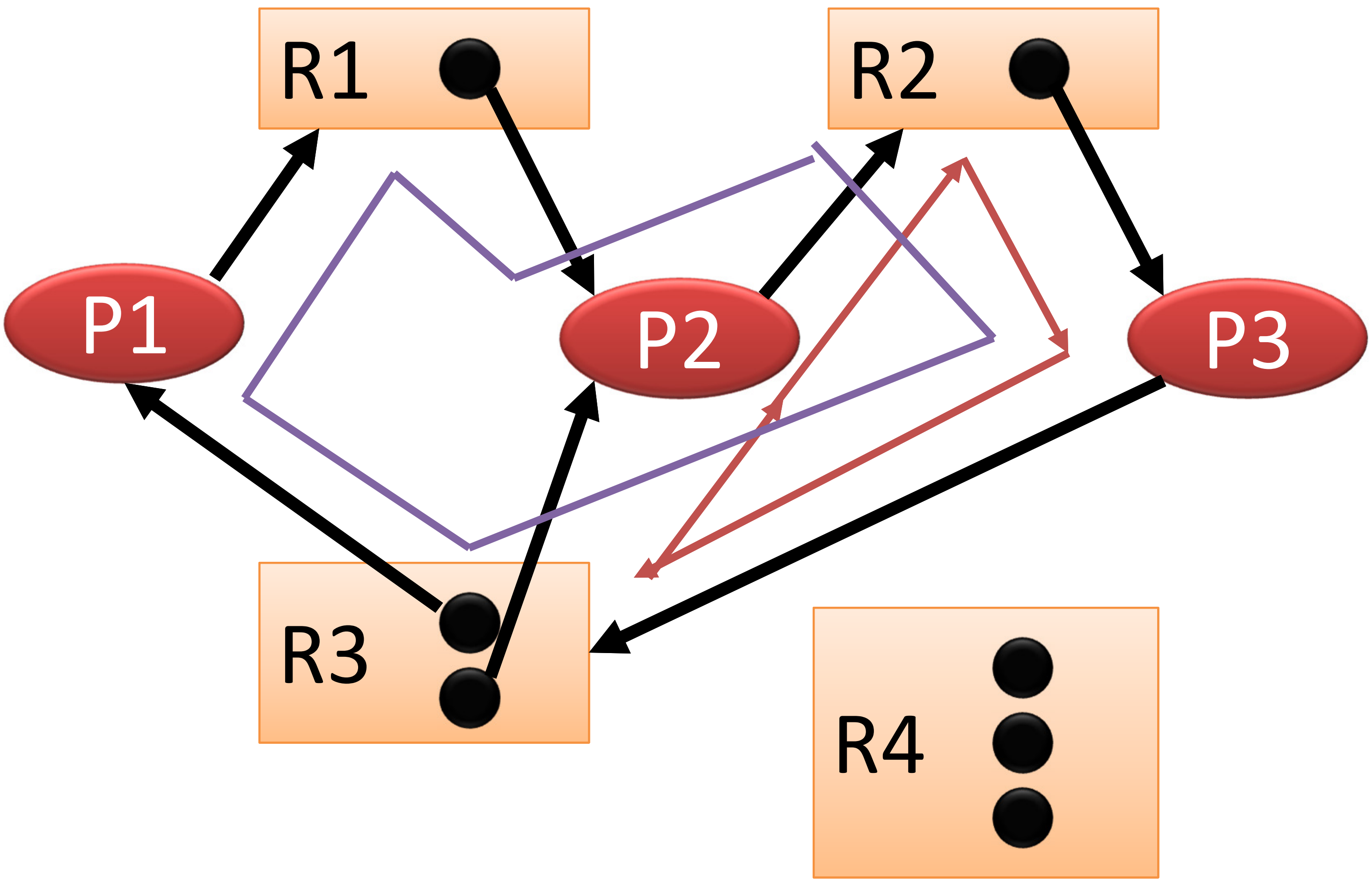
Example of a Resource Allocation Graph with a Deadlock



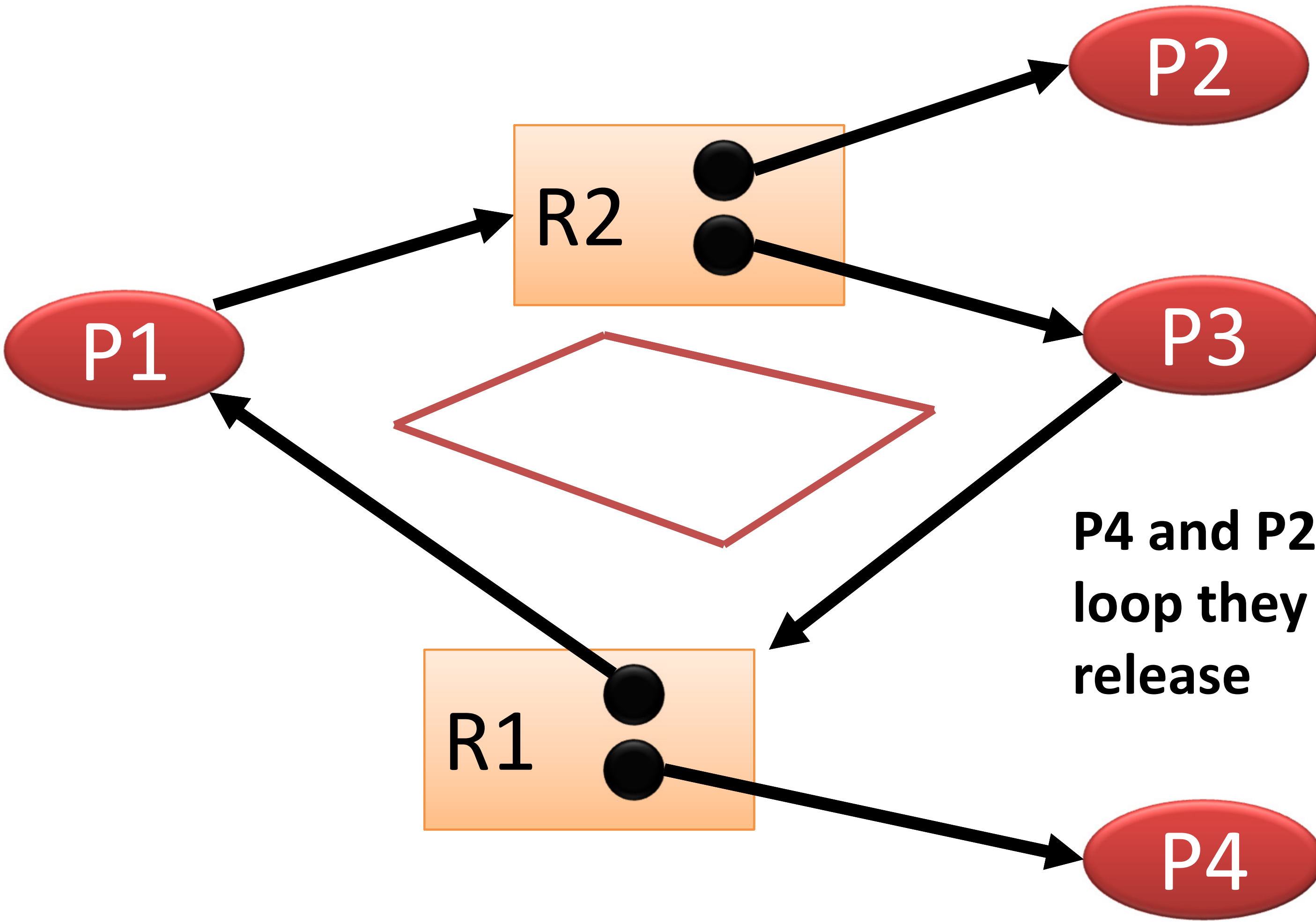
Example of a Resource Allocation Graph with a Deadlock



Example of a Resource Allocation Graph with a Deadlock



Graph with a Cycle but No Deadlock



P4 and P2 are not in loop they will release

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**

Methods for Handling Deadlocks

- Ensure that the system will ***never*** enter a deadlock state:
  - Deadlock prevention
  - Deadlock avoidance
- 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

Prevention, Avoidance & Detection

- Cold wave in December
- Prevention
  - Don’t go outside: it is too restrictive
- Avoidance
  - Go to outside but wear sweeter/jacket
- Recovery : Got cold : Take medicine

Prevention, Avoidance & Detection

- Diabetes and Sugar
- Prevention
  - Don’t take sugar, fruits, rice, patato, Cake, Rasogola, Laddu
  - With out having any symptoms of diabetes: it is too restrictive
- Avoidance
  - Take all food but care fully, if you are symptom is boundary case
- Recovery : Got diabetes : Take medicine

Deadlock Prevention

Deadlock Prevention

Restrain the ways request can be made

- **Mutual Exclusion**
  - Not required for sharable resources (e.g., read-only files);
  - Must hold for non-sharable resources

Deadlock Prevention

Restrain the ways request can be made

- **Hold and Wait**
  - Must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution
  - Or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible

Deadlock Prevention (Cont.)

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

Deadlock Prevention (Cont.)

- **Circular Wait**
  - Impose a **total ordering** of all resource types
  - And require that each process requests resources in an increasing order of enumeration

Deadlock Example

```
#define pthread_mutex PM
/* thread one runs in this function */
void *do_work_one(void *param) {
  PM_lock(&Mutex1);      PM_lock(&Mutex2);
            /** * Do some work */
  PM_unlock(&Mutex2);   PM_unlock(&Mutex1);
}
```

```
/* thread two runs in this function */
void *do_work_two(void *param){
  PM_lock(&Mutex2);     PM_lock(&Mutex1);
           /** * Do some work */
  PM_unlock(&Mutex1);   PM_unlock(&Mutex2);
}
```

Thread Lock are not in Orders

Thread 1 : Lock1 $\xrightarrow{\text{Increasing}}$ Lock2

Thread 2 : Lock2 $\xrightarrow{\text{Decreasing}}$ Lock1

At this point Both may hold one lock and wait for other

Deadlock Example with Lock Ordering

```
void transaction(Account from, Account to, double amount) {
  mutex lock1, lock2;
  lock1 = get_lock(from);  lock2 = get_lock(to);
  acquire(lock1);
    acquire(lock2);
      withdraw(from, amount);  deposit(to, amount);
    release(lock2);
  release(lock1);
}
```

Transactions 1 and 2 execute concurrently. Transaction 1 transfers Rs 25,000 from account A to account B, and Transaction 2 transfers Rs 50,000 from account B to account A

Thread Locks are in Orders: but Still Deadlock…?

Transaction 1 : LockS → Increasing → LockD

Transaction 2 : LockS → Increasing → LockD

Let source of T1 is A and source of T2 is B

And

Dest of T1 is B and dest T2 is A

At this point Both may hold one lock and wait for other

Deadlock Avoidance