Handling Resource Sharing and Dependencies among Real Time Tasks

- In many applications, real-time tasks need to share some resources among themselves.
- Often these shared resources need to be used by the individual tasks in exclusive mode.
- If a task is preempted before it completes using the resource, then the resource can become corrupted. Example: Files, devices and certain data structures.
- These resources are also called non-preemptable resources or critical sections.

Priority Inversion

- When a lower priority task is already holding a resource, a higher priority task needing the same resource has to wait and cannot make progress with its computations.
- The higher priority task would remain blocked until the lower priority task releases the required non-preemptable resource.
- The higher priority task is said to undergo simple priority inversion on account of the lower priority task for the duration it waits while the lower priority task keeps holding the resource.

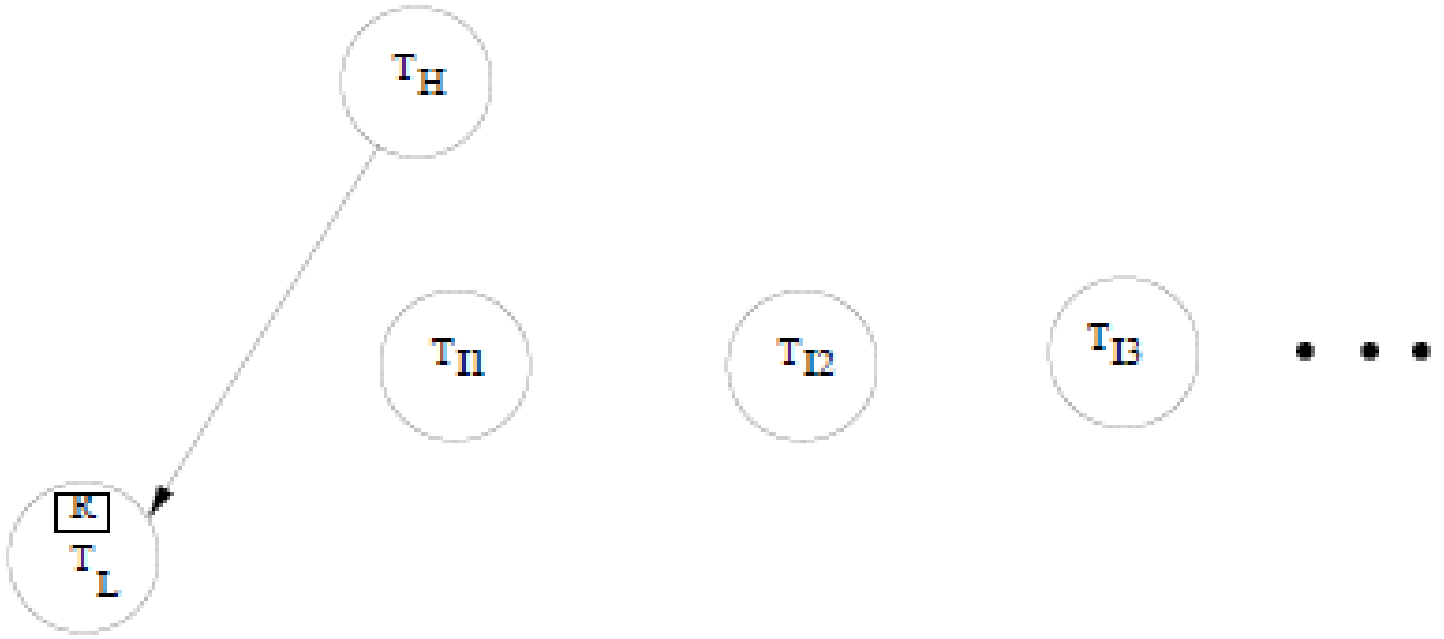
Unbounded Priority Inversion

- While even a simple priority inversion does delay a higher priority task by sometime, the duration for which a task blocks due to a simple priority inversion can be made very small if all tasks are made to restrict themselves to very brief periods of critical section usage.
- Therefore, a simple priority inversion can easily be tolerated through careful programming.
- However, a more serious problem that arises during sharing of critical resource among tasks is unbounded priority inversion.

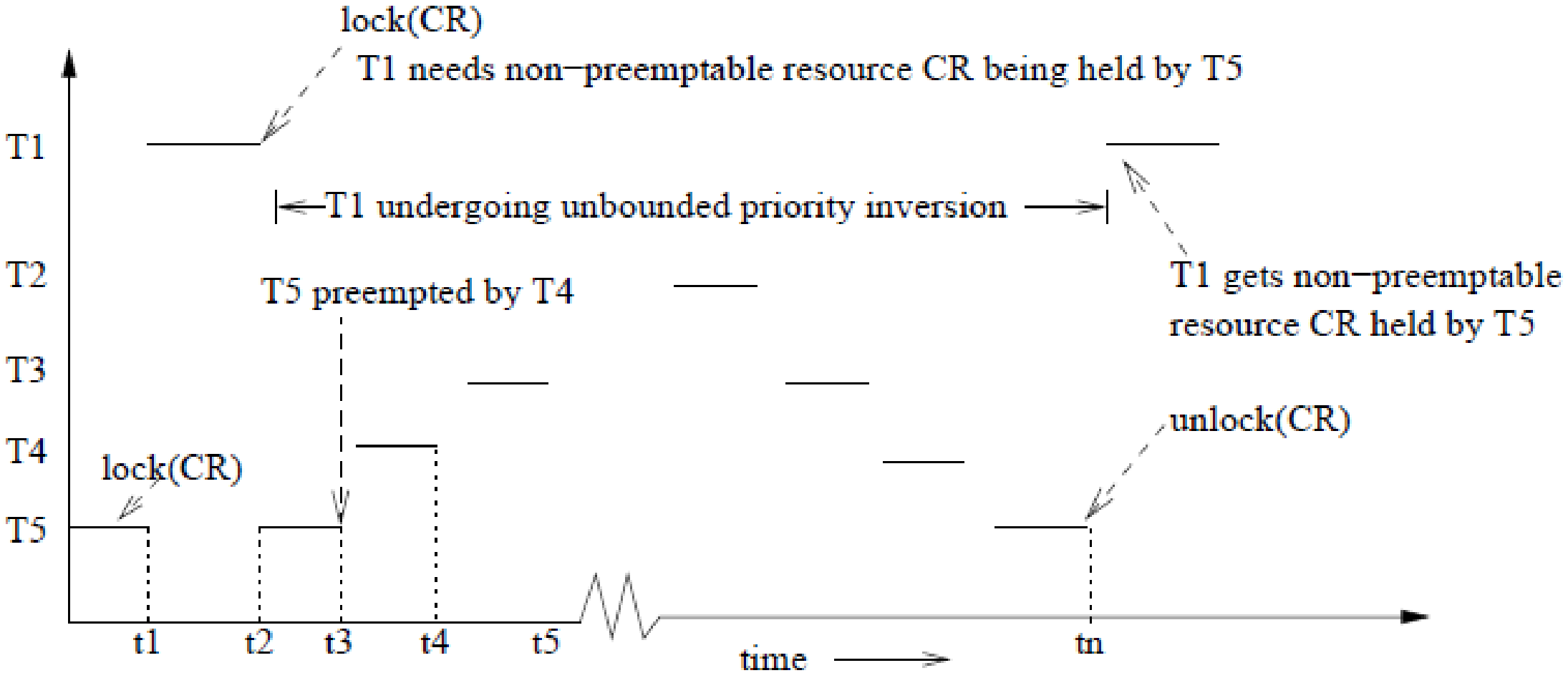
Unbounded Priority Inversion

- Unbounded priority inversion occurs when a higher priority task waits for a lower priority task to release a resource it needs, and in the meanwhile intermediate priority tasks preempt the lower priority task from CPU usage repeatedly.
- As a result, the lower priority task can not complete its usage of the critical resource and the higher priority task waits indefinitely for its required resource to be released.

Unbounded Priority Inversion



Unbounded Priority Inversion



MARS Path Finder

- Landed on the MARS surface on July 4th, 1997
- Bounced onto the Martian surface surrounded by airbags.
- Deployed the Sojourner rover.
- Gathered and transmitted voluminous data back to Earth

MARS Pathfinder Bug

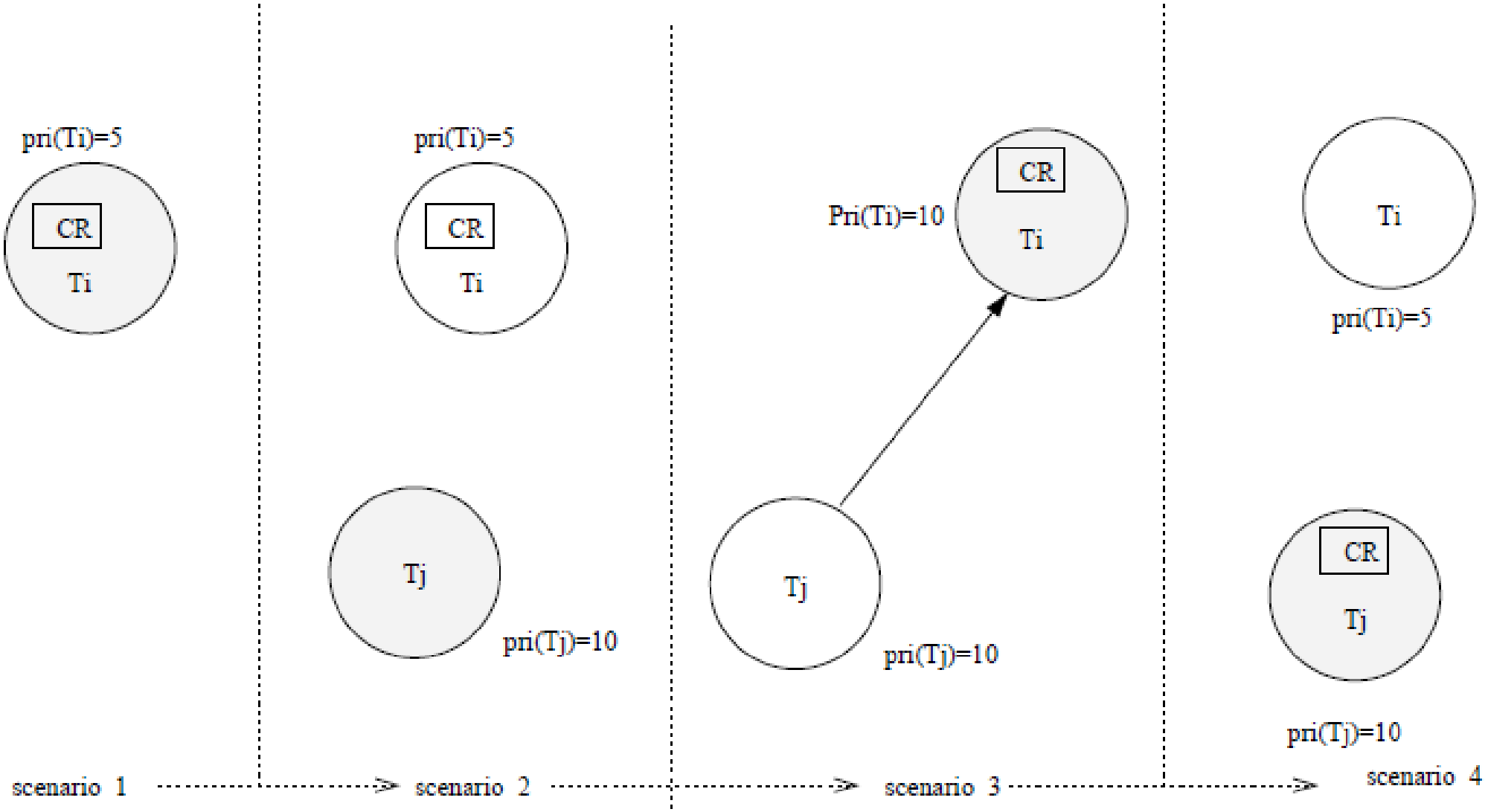
- Pathfinder began experiencing total system resets: Each resulting in loss of data
- The failures were reported as software glitches , the computers inability to handle too many things etc.
- Engineers spent hours running exact spacecraft replica in lab: replicated the precise conditions under which the reset occurred.
  - Turning on priority inheritance would prevent the resets.
  - Initialization parameters were stored in global variables.
  - A short C program was uploaded to the spacecraft

Priority Inheritance Protocol

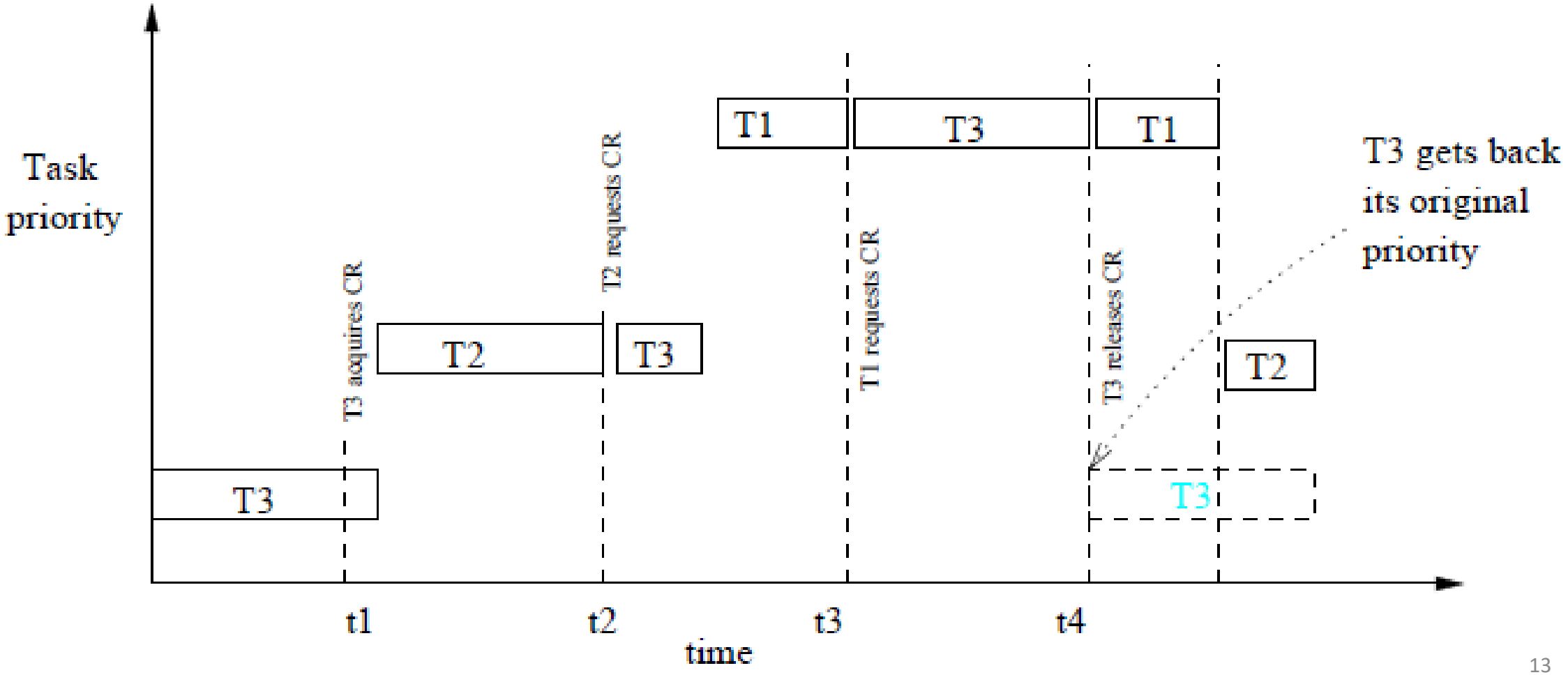
- Whenever a task suffers priority inversion, the priority of the lower priority task holding the resource is raised through a priority inheritance mechanism.
- When many tasks are waiting for a resource, the task holding the resource inherits the highest priority of all tasks waiting for the resource.
- Since the priority of the low priority task holding the resource is raised to equal the highest priority of all tasks waiting for the resource being held by it, intermediate priority tasks can not preempt it and unbounded priority inversion is avoided.

- As soon as a task that had inherited the priority of a waiting higher priority task releases the resource it gets back its original priority value if it is holding no other critical resources.
- In case it is holding other critical resources, it would inherit the priority of the highest priority task waiting for the resources being held by it.

Working of Priority Inheritance Protocol



Priority changes Under PIP



PIP pseudocode

```
if the required resource is free then
        grant it.
if the required resource is being held by a higher priority task then
        wait for the resource
if the required resource is held by a lower priority task then
        {
                wait for the resource
                the low priority task holding the
                resource acquires the highest priority of
                tasks waiting for the resource.
        }
```

- PIP suffers from two important problems
  - Deadlock
  - Chain Blocking

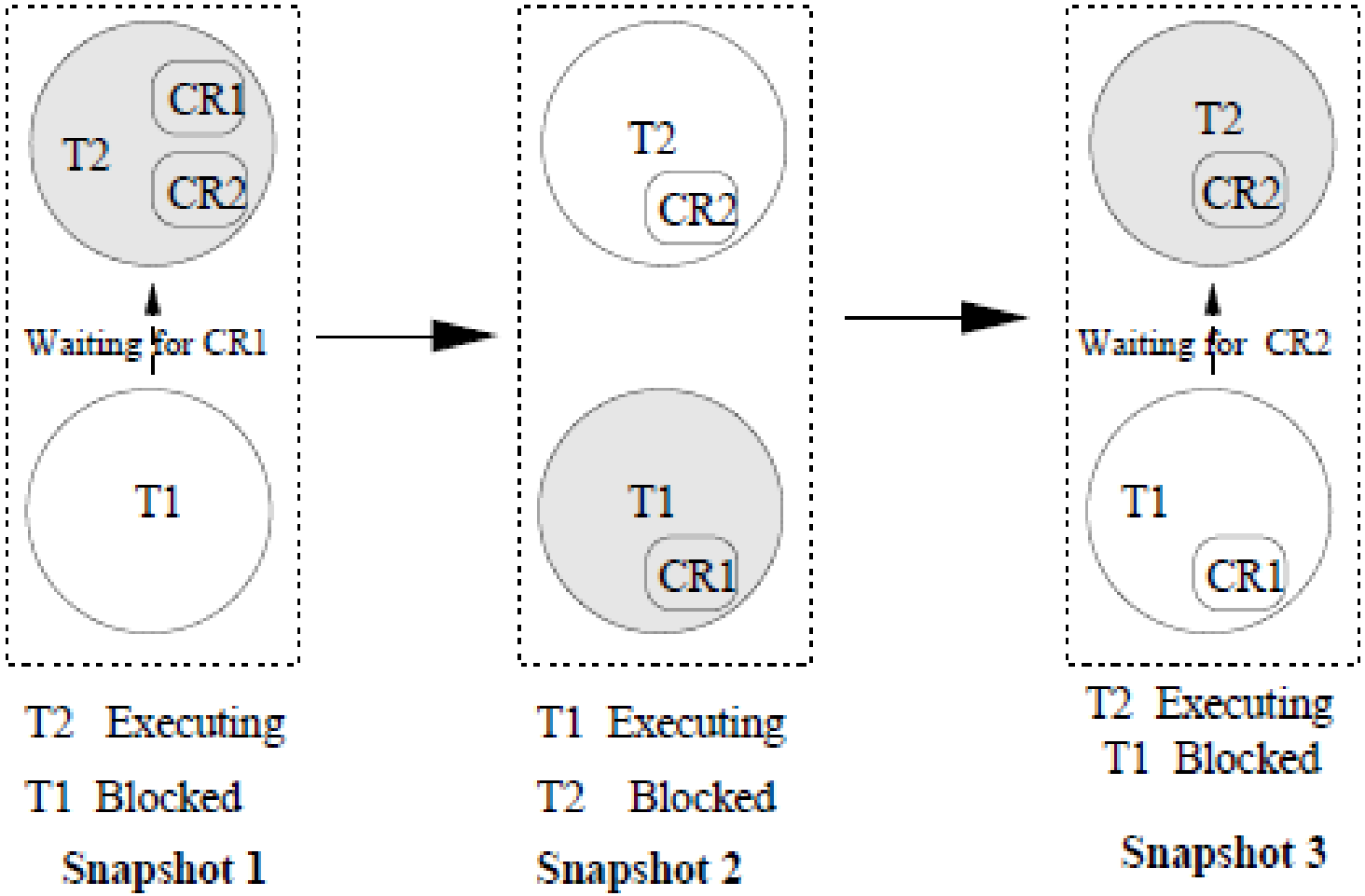
Deadlock

- The basic Priority Inheritance Protocol (PIP) leaves open the possibility of deadlocks.
- Consider the following sequence of actions of two tasks T1 and T2 which need access to two shared critical resources CR1 and CR2.
- T1 : Lock CR1, Lock CR2, Unlock CR2, Unlock CR1
- T2 : Lock CR2, Lock CR1, Unlock CR1, Unlock CR2

Chain Blocking

- A task is said to undergo chain blocking, if each time it needs a resource, it undergoes priority inversion.
- If a task needs n resources for its computations, it might have to undergo priority inversions n times to acquire all its resources.

Chain Blocking in PIP



Highest Locker Protocol (HLP)

- In HLP every critical resource is assigned a ceiling priority value.
- The ceiling priority of a critical resource (CR) is defined as the maximum of all those tasks which may request to use this resource.
- Under HLP, as soon as a task acquires a resource, its priority is set equal to the ceiling priority of the resource.
- If a task holds multiple resources, then it inherits the highest ceiling priority of all its locked resources.
- Consider a scheduler that follows FCFS scheduling policy among equal priority tasks. Let the ceiling priority of a resource R_i be denoted by Ceil (R_i) and the priority of a task T_i be denoted as pri (T_i). Then ceil (R_i) is defined as
  - Ceil (R_i) = max ({pri (T_i)/ T_i needs R_i })
- The ceiling priority of a critical resource R_i is the maximum of the priority of all tasks that may use R_i

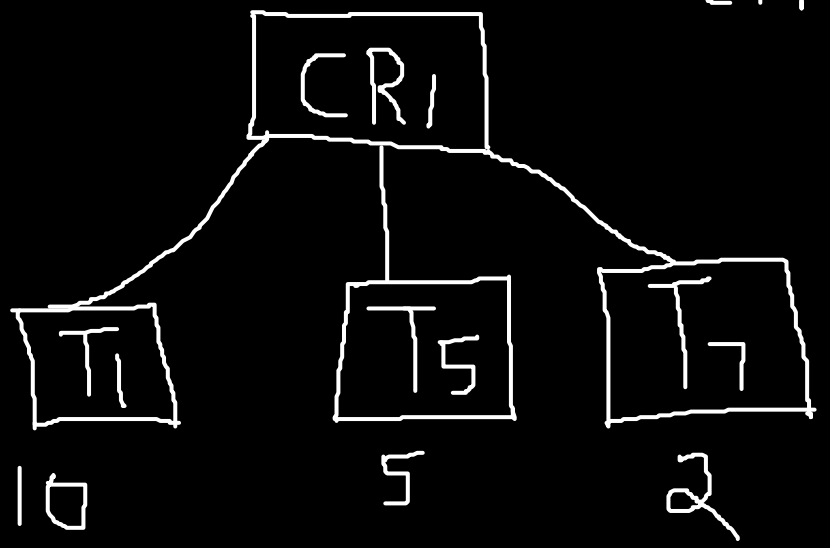
- If higher priority values indicate lower priorities (Unix), then the ceiling priority would be the minimum priority of all tasks needing the resource. That is
  - Ceil (R_i) = min ({pri (T_i)/ T_i needs R_i })
- For operating systems supporting time sliced round robin scheduling among equal priority tasks and larger priority value indicates higher priority, the rule for computing the ceiling priority is :
  - Ceil (R_i) = max ({pri (T_i)/ T_i needs R_i }) + 1
- For the case where larger priority value indicates lower priority (and time sliced round robin scheduling among equal priority tasks), the rule for computing the ceiling priority is :
  - Ceil (R_i) = min ({pri (T_i)/ T_i needs R_i }) + 1

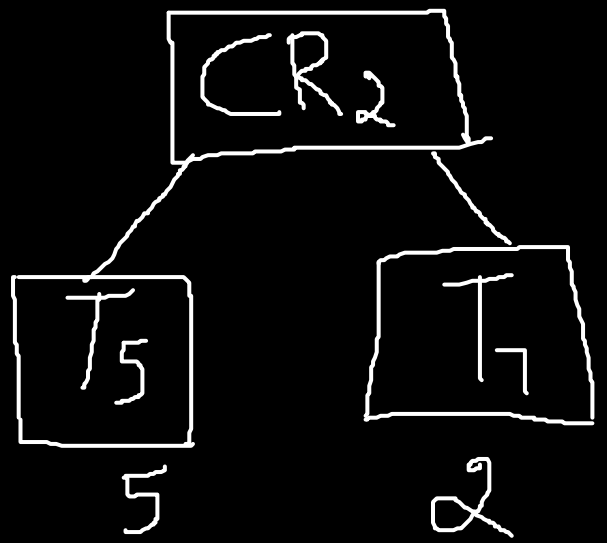
Example

- A resource CR1 is shared by the tasks T1, T5 and T7 and CR2 is shared by T5 and T7.
- Assume that the priority of T1 = 10, that of T5 = 5, priority of T7 = 2.
- Then, the priority of CR1 will be the maximum of the priorities of T1, T5 and T7. Then the ceiling priority of CR1 is Ceil(CR1) = max({10, 5, 2}) = 10.
- Therefore, as soon as either of T1, T5 or T7 acquires CR1, its priority will be raised to 10.
- The rule of inheritance of priority is that any task that acquires the resource inherits the corresponding ceiling priority

- If a task is holding more than one resource, its priority will become maximum of the ceiling priorities of all the resources it is holding.
- For example, Ceil (CR2)==max{5,2}=5.
- A task holding both CR1 and CR2 would inherit the larger of the two ceiling priorities i.e 10.

Ceil $(CR_1) = 10$



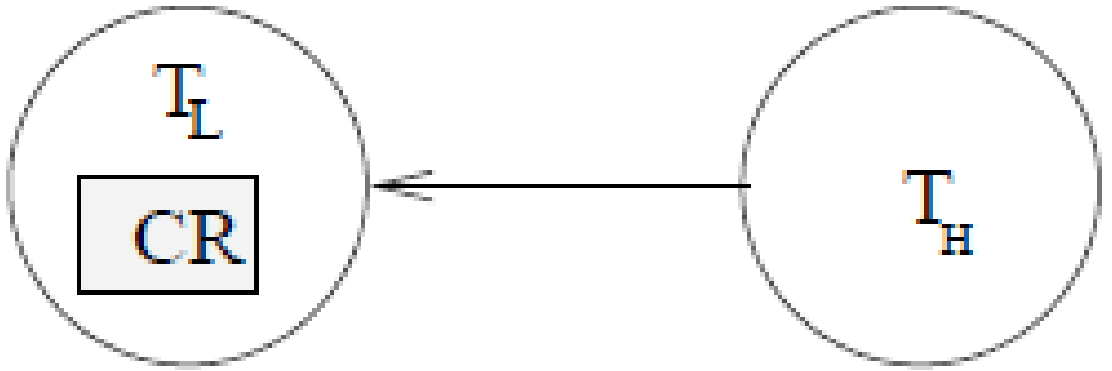


Ceil $(CR_2) = 5$

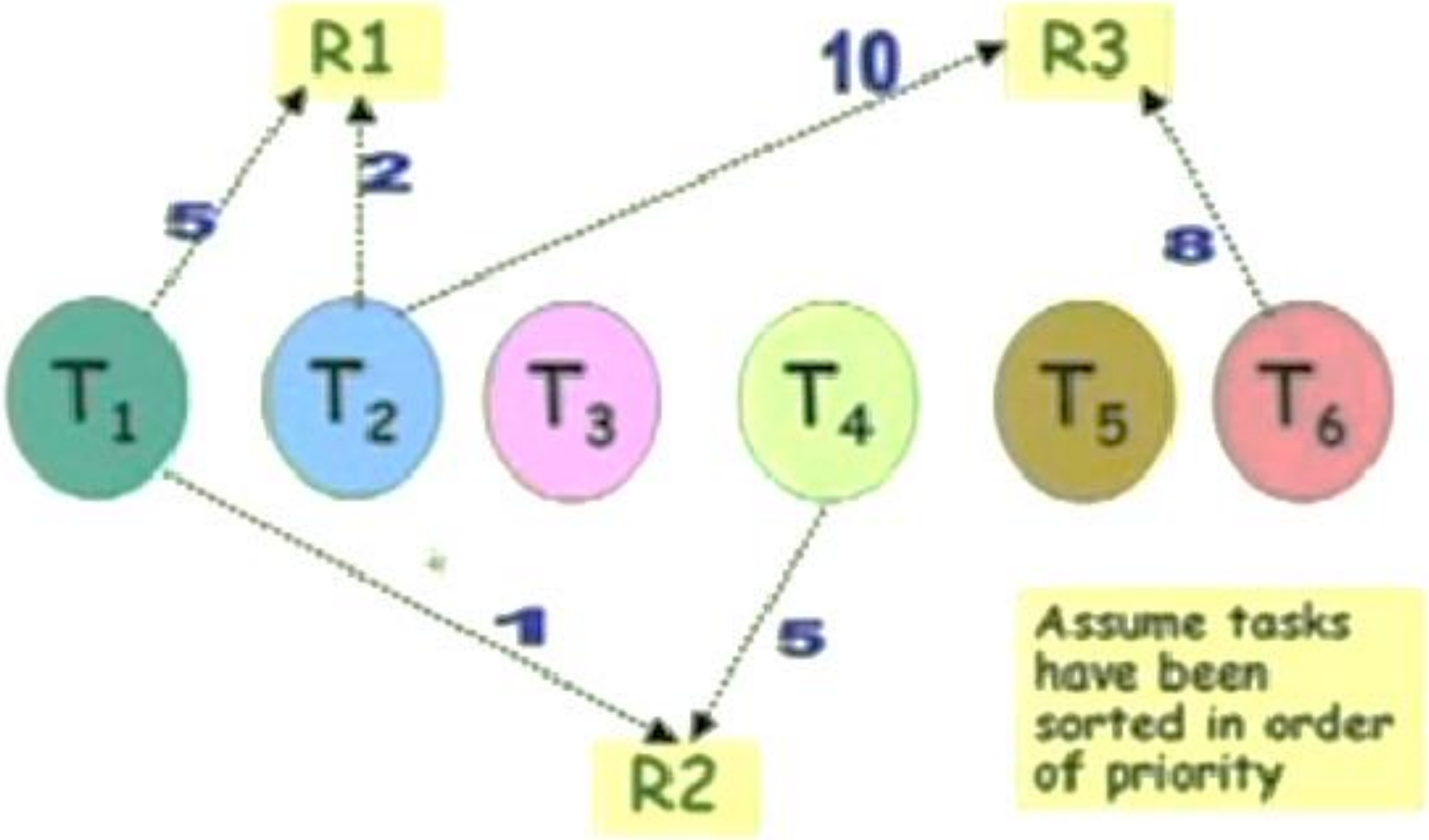
- Shortcomings of HLP
  - Possibility of Inheritance related inversion.
  - Inheritance related inversion occurs when the priority of a low priority task holding a resource is raised to a high value by the ceiling rule, the intermediate priority tasks not needing the resource can not execute and are said to undergo inheritance related inversion

Different types of priority inversions under PCP

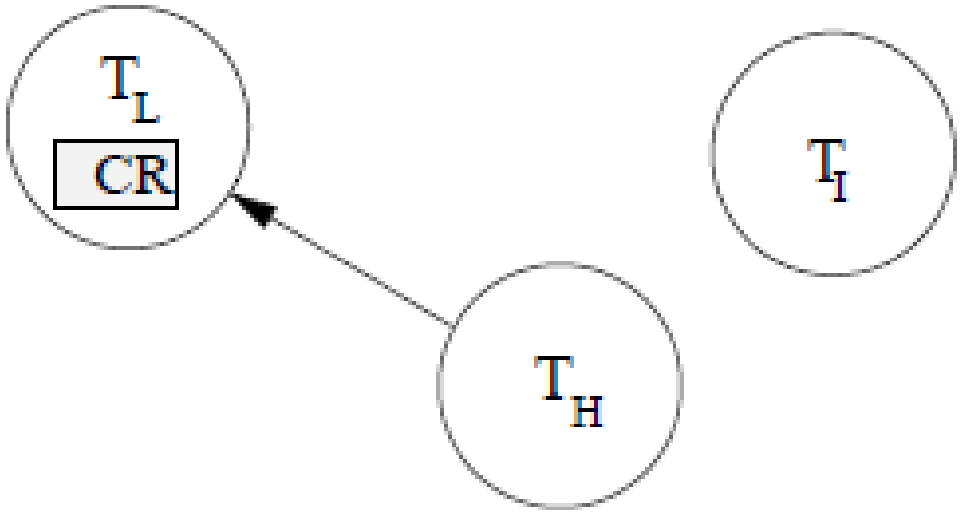
- Direct Inversion
  - Occurs when a higher priority task waits for a lower priority task to release a resource that it needs.



Identify the direct inversions

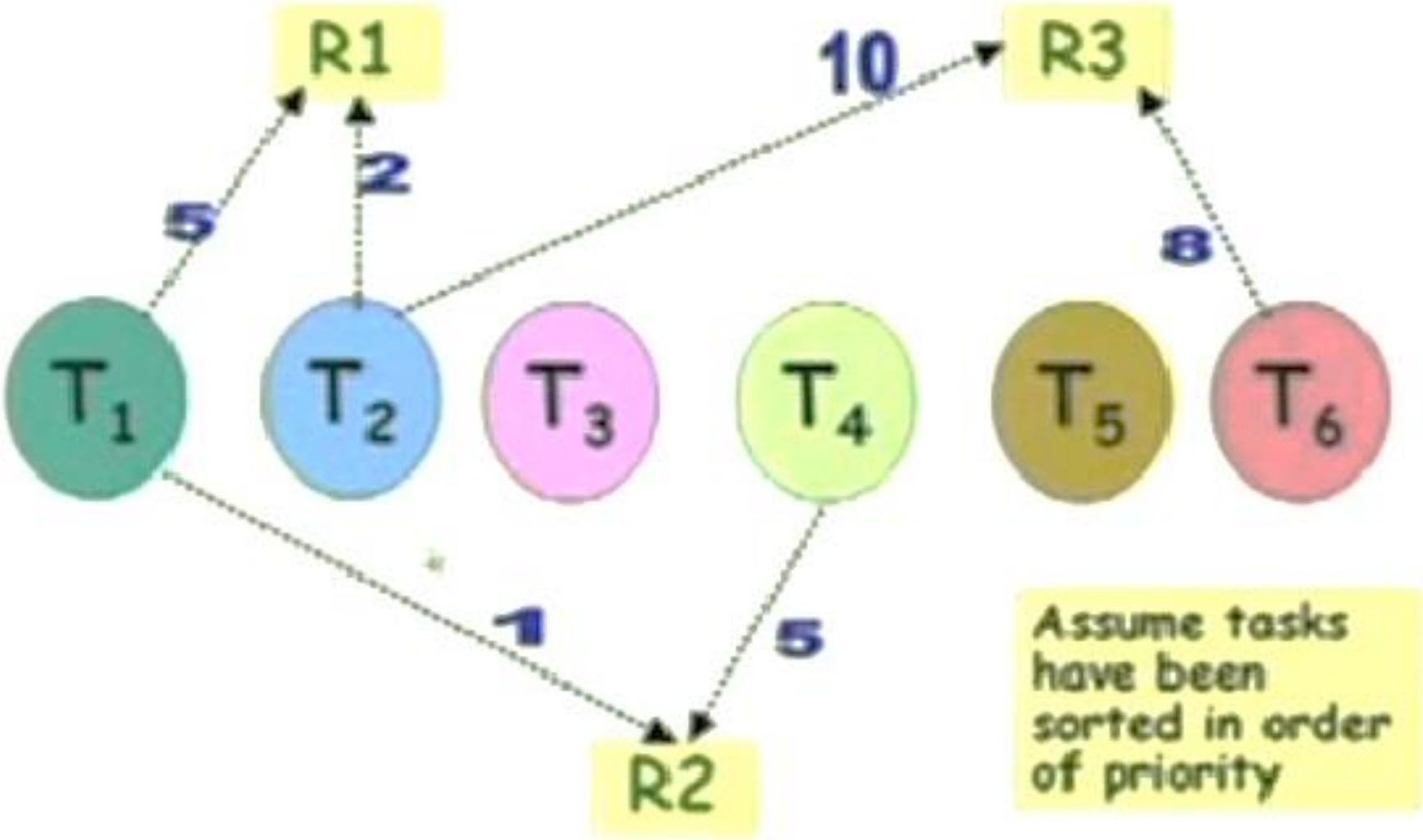


Inheritance related Inversion



- Inheritance related Inversion
  - Consider a situation where a lower priority task is holding a resource and a higher priority task is waiting for it.
  - Then, the priority of the lower priority task is raised to that of the waiting higher priority task.
  - As a result, the intermediate priority tasks not needing the resource undergo inheritance related inversion.

Identify the inheritance related inversions



Inversion Analysis