Real-Time Systems

Resource Access Control Protocols

Assumptions

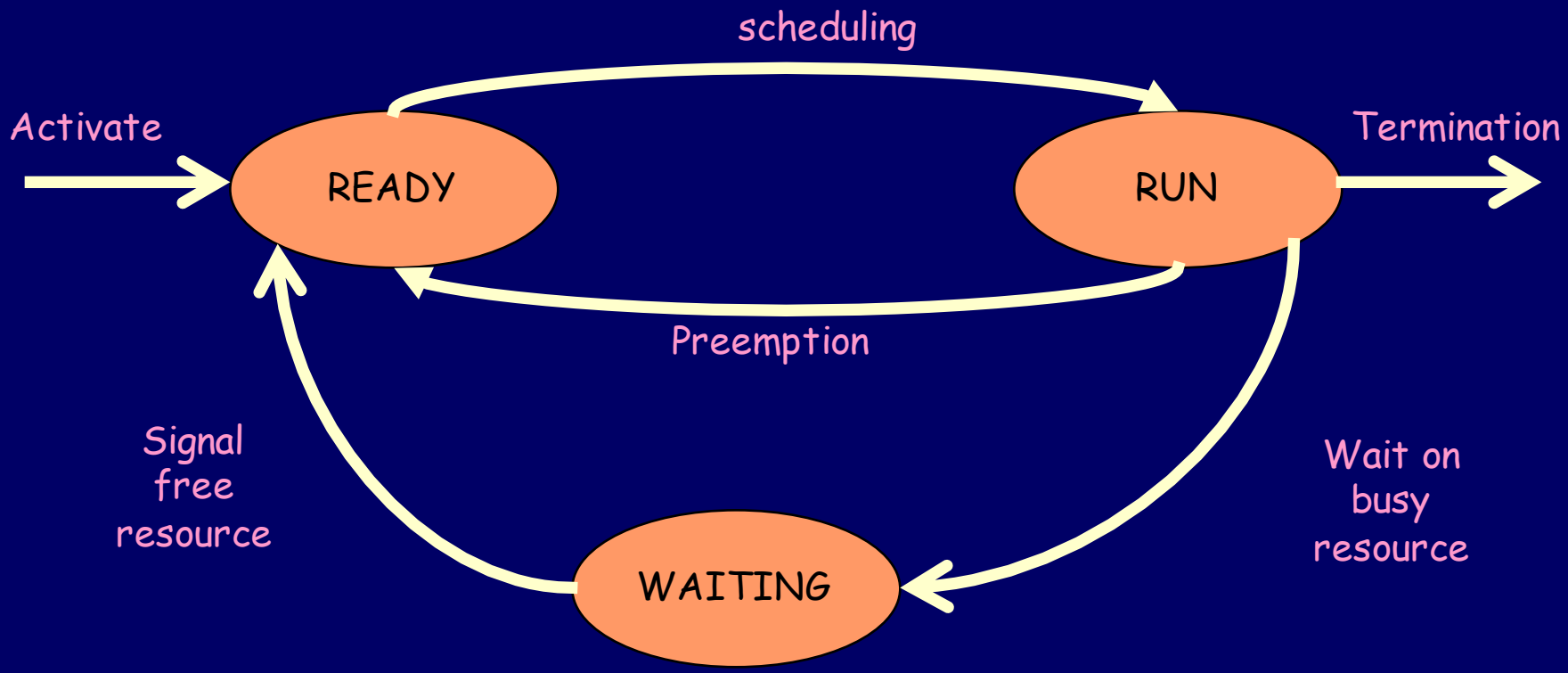
- Periodic tasks
- Task can have resource access
- Semaphore is used for mutual exclusion
- RMS scheduling

Background – Task State diagram

- Ready State: waiting in ready queue
- Running State: CPU executing the task
- Blocked: waiting in the semaphore queue until the shared resource is free

- Semaphore types – mutex (binary semaphore), counting semaphore

Task State Diagram



Process/Task state diagram with resource constraints

Priority Inversion Problem

Priority inversion is an undesirable situation in which a **higher** priority task gets **blocked** (waits for CPU) for more time than that it is supposed to, by **lower** priority tasks.

Example:

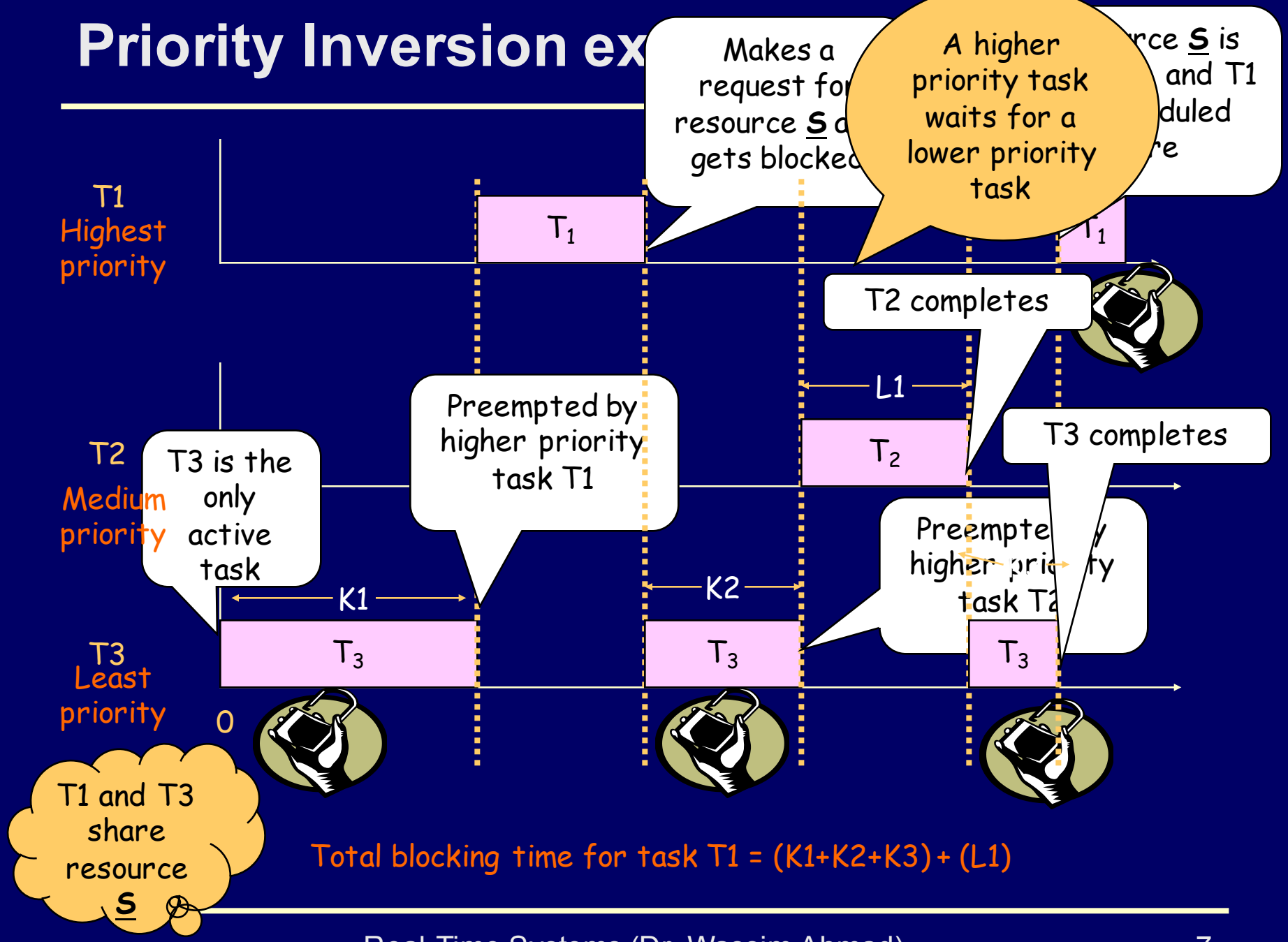
- Let T_1 , T_2 , and T_3 be the three periodic tasks with decreasing order of priorities.
- Let T_1 and T_3 share a resource “S”.

Priority Inversion -- Example

- T3 obtains a lock on the semaphore *S* and enters its critical section to use a shared resource.
- T1 becomes ready to run and preempts T3. Then, T1 tries to enter its critical section by first trying to lock *S*. But, *S* is already locked by T3 and hence T1 is blocked.
- T2 becomes ready to run. Since only T2 and T3 are ready to run, T2 preempts T3 while T3 is in its critical section.

Ideally, one would prefer that the highest priority task (T1) be blocked no longer than the time for T3 to complete its critical section. However, the duration of blocking is, in fact, **unpredictable** because task T2 got executed in between.

Priority Inversion ex



Priority Inheritance Protocol

Priority inheritance protocol solves the problem of priority inversion.

Under this protocol, if a higher priority task T_H is blocked by a lower priority task T_L, because T_L is currently executing critical section needed by T_H, T_L temporarily inherits the priority of T_H.

When blocking ceases (i.e., T_L exits the critical section), T_L resumes its original priority.

Unfortunately, priority inheritance may lead to deadlock.

Priority Inheritance Protocol – Deadlock

Assume T2 > T1 (i.e., T2 has high priority)

task	operation sequence on critical section
T_1	$Lock(CS_2)$ $Lock(CS_1)$ $Unlock(CS_1)$ $Unlock(CS_2)$
T_2	$Lock(CS_1)$ $Lock(CS_2)$ $Unlock(CS_2)$ $Unlock(CS_1)$

time	task	action
t_0	T_1	starts execution
t_1	T_1	locks CS_2
t_2	T_2	activated and preempts T_1 due to its higher priority
t_3	T_2	locks CS_1
t_4	T_2	attempts to lock CS_2, but is blocked because T_1 has a lock on it
t_5	T_1	inherits the priority of T_2 and starts executing
t_6	T_1	attempts to lock CS_1, but is blocked because T_2 has a lock on it
$\geq t_7$	-	both the tasks cannot proceed (deadlocked)

Figure 1: Example for priority inheritance protocol resulting in deadlock

Priority Ceiling Protocol

- Priority ceiling protocol solves the priority inversion problem without getting into deadlock.

- For each semaphore, a *priority* ***ceiling*** is defined, whose value is the **highest** priority of all the tasks that may lock it.

- When a task T_i attempts to execute one of its critical sections, it will be **suspended** unless its priority is **higher** than the priority ceiling of **all** semaphores **currently** locked by tasks other than T_i.

Priority Ceiling Protocol (Contd.)

- If task T_i is unable to enter its critical section for this reason, the task that holds the lock on the semaphore **with** the highest priority ceiling is said to be blocking T_i and hence **inherits** the priority of T_i.
- As long as a task T_i is **not** attempting to enter one of its critical sections, it will **preempt** every task that has a lower priority.

Priority Ceiling Protocol -- properties

- This protocol is the same as the priority inheritance protocol, **except** that a task T_i can also be blocked from entering a critical section if any other task is currently holding a semaphore whose priority ceiling is greater than or equal to the priority of task T_i.

- **Prevents** mutual **deadlock** among tasks
- A task can be blocked by lower priority **tasks at most once**

Priority Celiling Protocol - Example

- For the previous example, the priority ceiling for both CS_1 and CS_2 is the priority of T_2.
- From time t_0 to t_2, the operations are the same as before.
- At time t_3, T_2 attempts to lock CS_1, but is blocked since CS_2 (which has been locked by T_1) has a priority ceiling equal to the priority of T_2.
- Thus T_1 inherits the priority of T_2 and proceeds to completion, thereby preventing deadlock situation.

Priority Inversion - Real-world Example

- Mars Pathfinder mission (July 4, 1997)
- VxWorks (real-time OS), preemptive priority scheduling of threads (e.g., RMS)

- Priority inversion involving three threads:
  - Information bus task (T1), meteorological data gathering task (T3), communication task (T2). Priority order: T1>T2>T3
  - Shared resource: information bus (used mutex)

- Same situation as described in the previous example had occurred
- Findings: Priority ceiling protocol was found to be disabled initially, then it was enabled online and the problem was corrected

Priority Ceiling Emulation

- Once a task locks a semaphore, its priority is immediately raised to the level of the priority ceiling of the semaphore.
- Deadlock avoidance and block at-most-once result of priority ceiling protocol still holds.
- Restriction: A task cannot suspend its execution within the critical section.

Modeling Blocking Time and Earlier Deadline

- **Blocking time** (Bi) encountered by task Ti by lower priority tasks can be modeled by increasing Ti′s **utilization** by Bi/Pi.
- Earlier deadline (Di < Pi) can also be modeled as blocking time for **Ei = Pi – Di**.

- Net increase in task Ti′s utilization is

  (Bi + Ei) / Pi.

Modeling Blocking and Earlier Deadline (Cont.)

- Schedulability Check (sorted order T1 > T2 > ... > Tn) -- sufficient, but not necessary

$$\forall i, 1 \le i \le n, \frac{C_1}{p_1} + \frac{C_2}{p_2} + \cdots + \frac{(C_i + B_i + E_i)}{p_i} \le i(2^{1/i} - 1)$$

- Completion time Test (Exact analysis)
  - Earlier deadline (di < pi) case: same as DMS exact analysis
  - Blocking time (Bi) case:
    - Let Ci′ = Ci + Bi
    - While calculating Wi(t) for task Ti, use Ci′ for task Ti and for all other higher priority tasks Tj simply use Cj