CIS 721 - Real-Time Systems

Lecture 12: Priority Ceiling Protocols

Mitch Neilsen
neilsen@cis.ksu.edu

Outline

- Priority Inheritance Protocols
  - Basic Priority Inheritance Protocol
  - NonPreemptive Critical Sections (NPCS)
  - Basic (Original) Priority Ceiling Protocol
  - Stack-Based (Ceiling-Priority) Priority Ceiling Protocol
- Analysis
  - Utilization-Based Test
  - Response Time Analysis

Priority Inheritance Protocols

- L. Sha, R. Rajkumar, J. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”, *IEEE Transactions on Computers*, Vol. 39, No. 9, pages 1175-1185, 1990.

Notation

- Serially reusable (accessed by one task at a time) resource types $R_1, R_2, \ldots, R_m$
- Each critical section is denoted $[R,\eta;e]$ where
  - R denotes resource type
  - η denotes number of units required (default 1)
  - e denotes execution time in the critical section
- L(R) denotes a lock request for resource R
- U(R) denotes an unlock request for resource R
- Critical sections may be **nested**

Critical Section

- L(*R*) - Obtain lock on resource *R*
  - Access data (resource *R*) in critical section
- U(*R*) - Release lock on resource *R*

Last time, LLLRR becomes code:

Local computation (3 time units)

L(R)

Access R in critical section (2 time units)

U(R)

Simple Preemptive Priority-Based Scheduling Problems

- **Schedulability** – preemption of a low priority task in its critical section can lead to priority inversion and missed deadlines.
- **Timing Anomalies** – even reducing critical section times can lead to priority inversion and more missed deadlines.
- **Note:** Traditional resource management algorithms (e.g., Banker’s Algorithm) decouple resource management decisions from scheduling decisions, so they are not applicable to real-time systems.

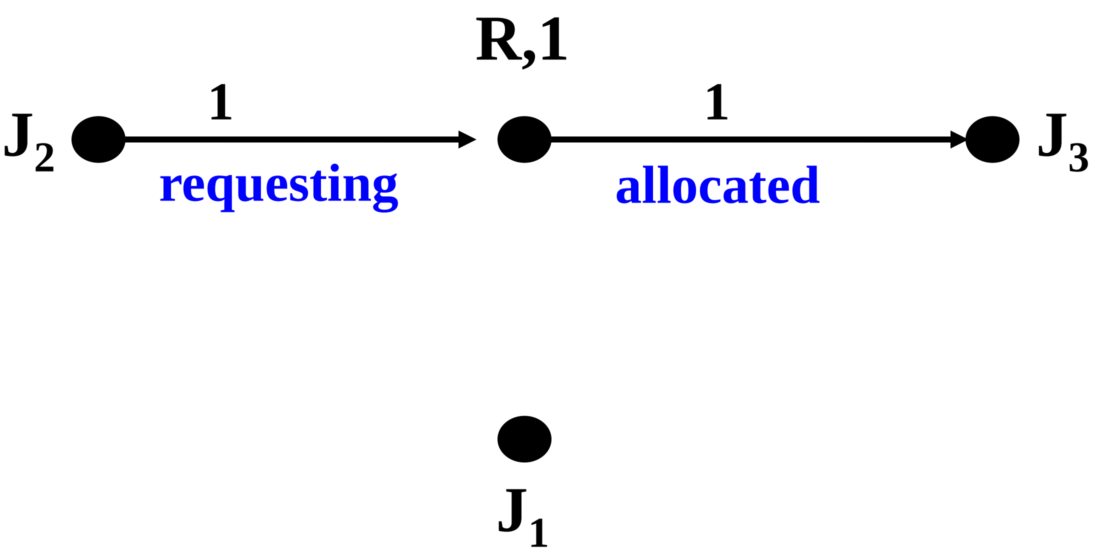
Timing Anomalies

- Does reducing the length of a critical section improve schedulability? **Not always!**
- Example: $\tau_i = (\varphi_i, p_i , e_i, [R,x;y])$, x = number of resources requested (x = 1 by default), y = length of cs, τ_1 = (6, 8, 5, [R;2]), τ_2 = (2, 22, 7, [R;4]), τ_3 = (0, 26, 6, [R;4]) is feasible.
- On the other hand, if we reduce the critical section time of τ_3 from 4 to 2.5, then the task set: τ_1 and τ_2 unchanged, τ_3 = (0, 26, **4.5**, [R;**2.5**]) is not feasible. (consider the case when τ_3 starts with R…R, and the other tasks start with L..L).

Blocking

- **Blocking** occurs when a higher priority task is waiting on a lower priority task.
- **Three types** of blocking may occur:
  - **Direct** - higher priority task attempts to lock a locked semaphore
  - **Priority-inheritance** - a medium priority task is blocked by a low priority task that has inherited the priority of a high priority task
  - **Priority-ceiling** - a task’s priority is not higher than the priority ceiling of a locked semaphore

Direct Blocking



Notation for Analysis

- Let B_i denote the **worst-case blocking time** due to lower priority tasks for any job of task τ_i.
- Let res_i denote the set of **resources** accessed by task τ_i.
- Let $cs(i,[r])$ denote the **maximum time in critical section** (accessing resource *r*) of a job in task τ_i.
- Let $ceil(r)$ denote the static ceiling priority for resource *r.*

Basic Priority Inheritance Protocol

- For each resource (semaphore), a list of blocked tasks must be stored in a priority queue.
- A task τ_i uses its assigned priority, unless it is in its critical section and blocks some higher priority tasks, in which case, task τ_i uses (**inherits**) the highest **dynamic** priority of all the tasks it blocks.
- Priority inheritance is **transitive**; that is, if task τ_i blocks τ_j and τ_j blocks τ_k , then τ_i can inherit the priority of τ_k (π_k).

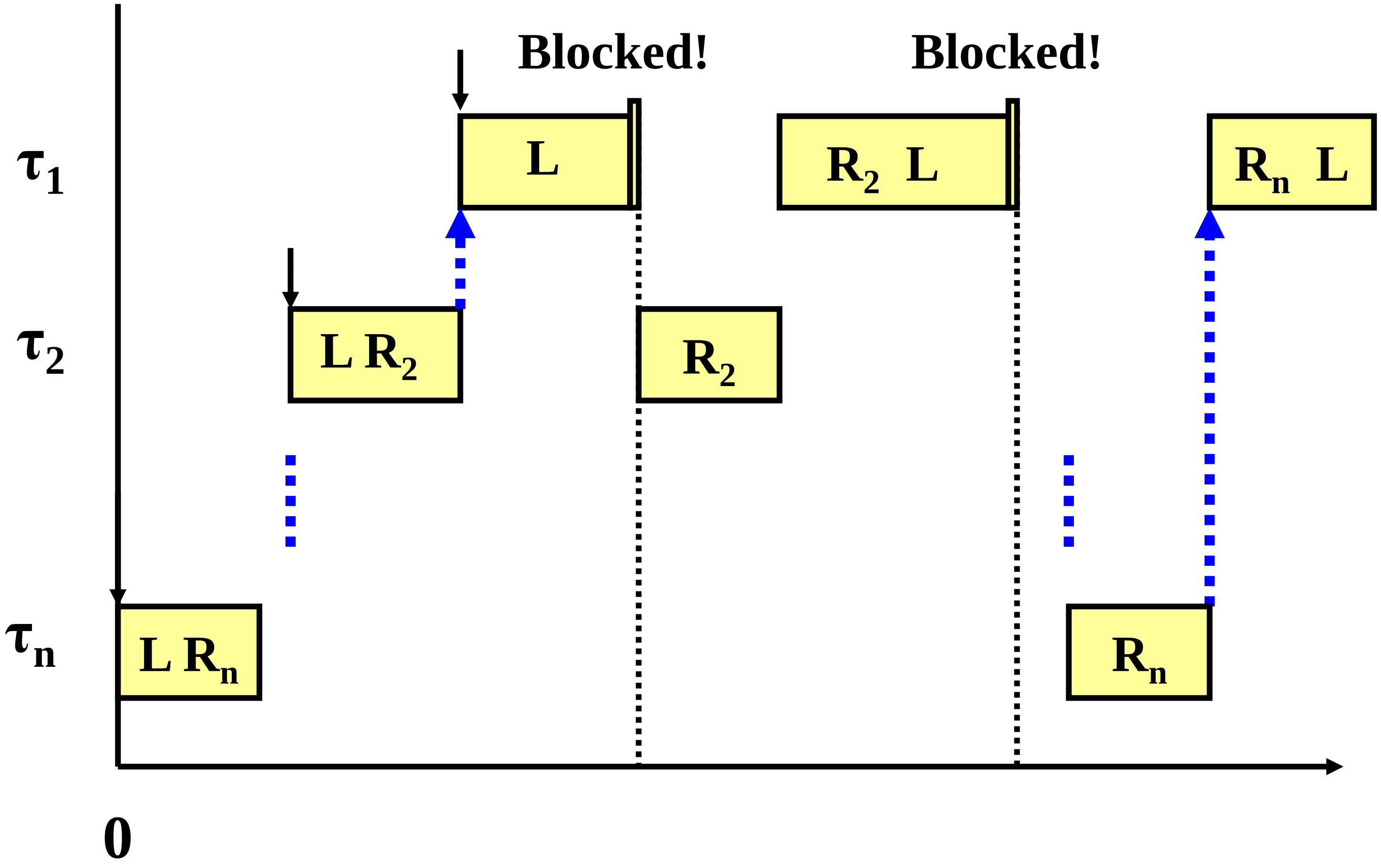
Problems

- The Basic **Priority Inheritance Protocol** has two problems:
  - **Deadlock** - two tasks need to access a pair of shared resources simultaneously. If the resources, say A and B, are accessed in opposite orders by each task, then deadlock may occur.
  - **Blocking Chain** - the blocking duration is bounded (at most the sum of critical section times), but may be substantial.

Blocking Chain Example

- Task τ_1 : L R_2 L R_3 L R_4 L ... L R_n L, $\varphi_1 = 2(n-1)$
- Task τ_2 : L R_2 R_2, $\varphi_2 = 2(n-2)$
- Task τ_3 : L R_3 R_3, $\varphi_3 = 2(n-3)$
- Task τ_4 : L R_4 R_4, $\varphi_4 = 2(n-4)$
- ...
- Task τ_{n-1} : L R_{n-1} R_{n-1}, $\varphi_{n-1} = 2$
- Task τ_n : L R_n R_n, $\varphi_n = 0$

Priority Inheritance - Blocking Chain



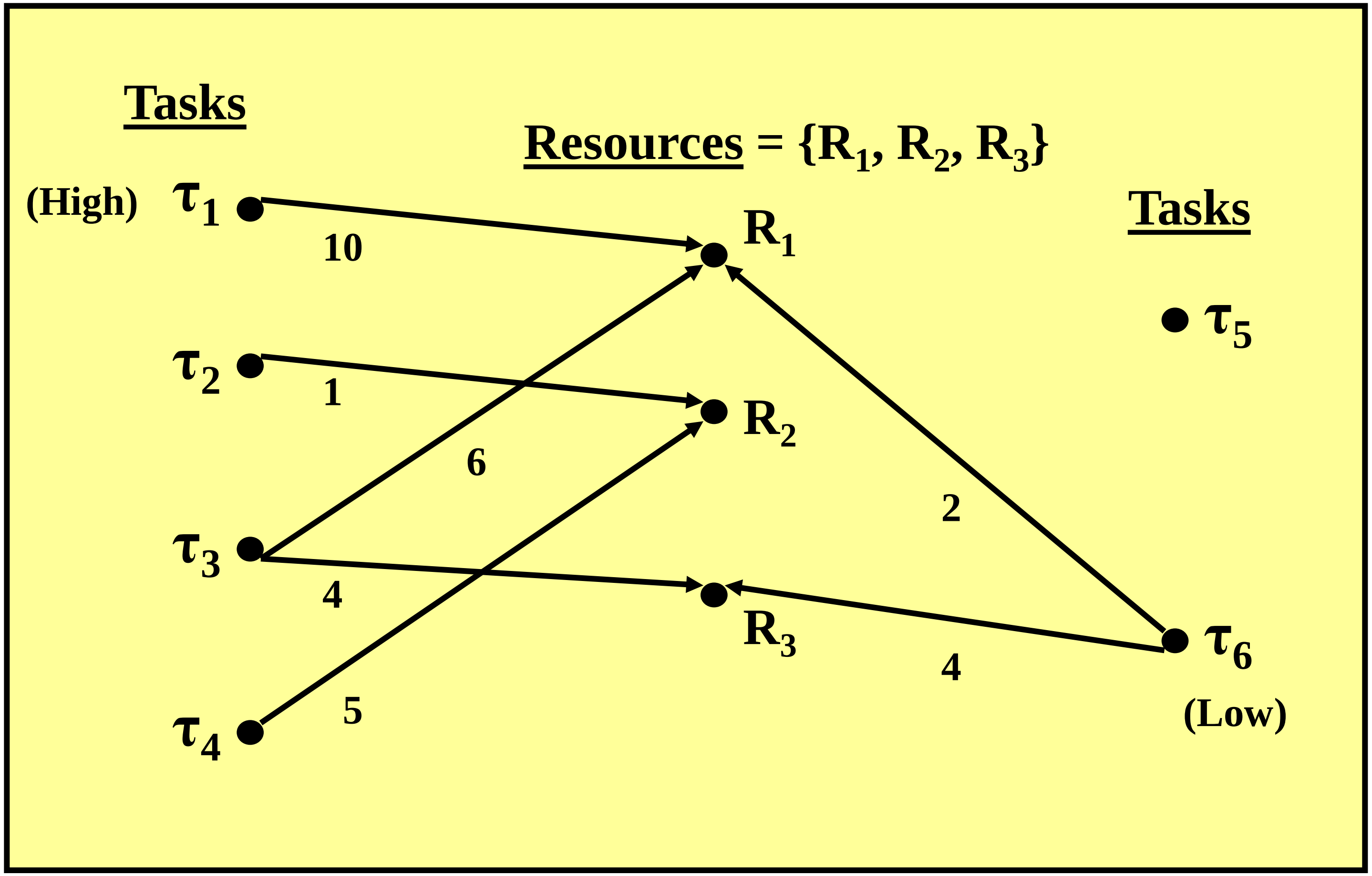
Blocking Time (Priority Inheritance)

- Under the **basic priority inheritance protocol**, if there are *m* semaphores that can block a job J in a task, then J can be directly blocked at most *m* times; e.g., on each semaphore at most once.
- **Worst-Case Blocking Time** is given by:

$$B_i = \sum_{r \in res_i} \max_{j \in lp(i)} \{cs(j,r)\} + \max_{\substack{j \in lp(i) \\ ceil(r) > \pi_i}} \{cs(j,r)\}$$

Directly blocked **Priority inheritance blocked**

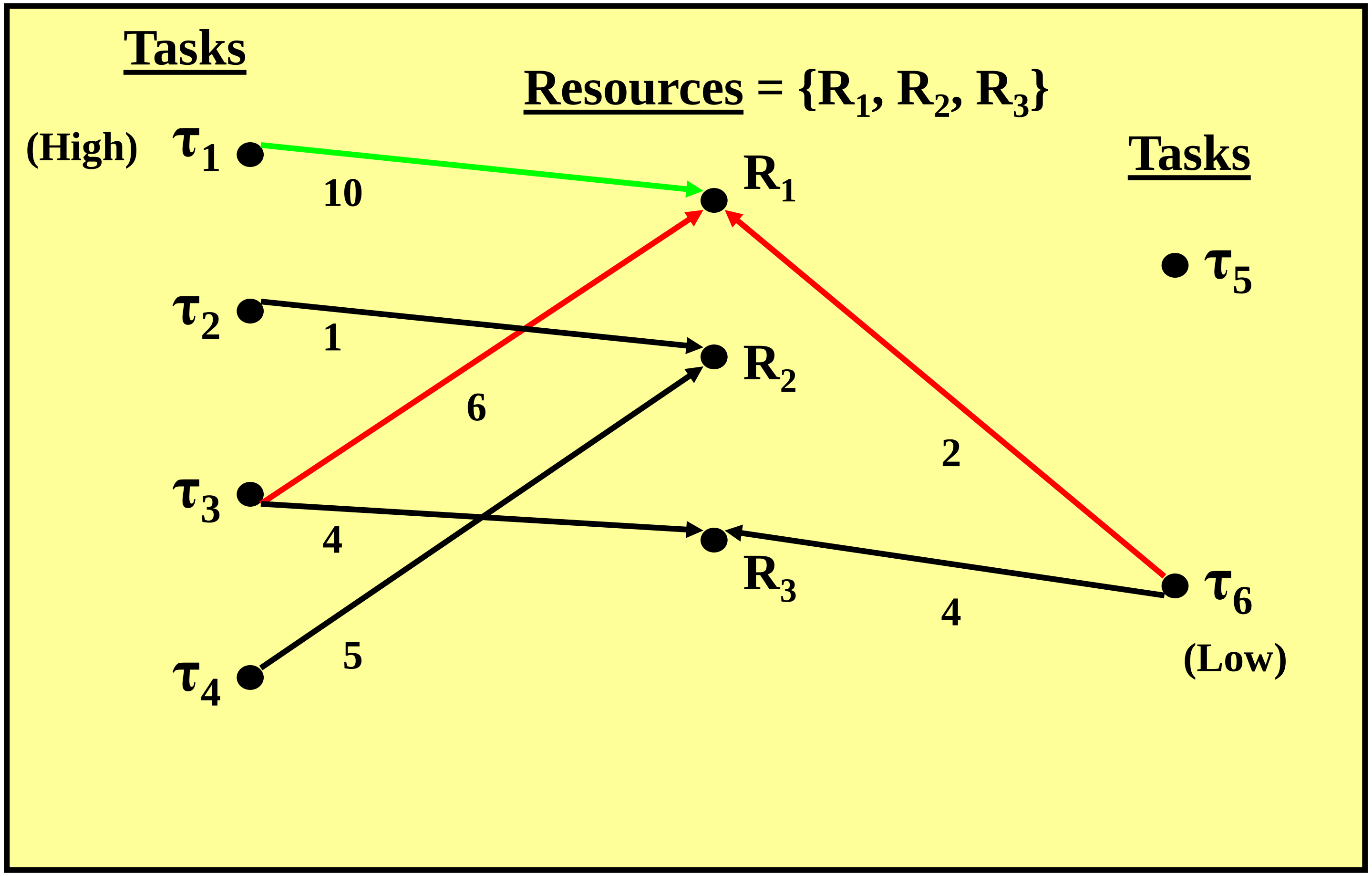
Example



Tasks
Resources = {R1, R2, R3}
Tasks
(High) τ1
10
R1
τ5
τ2
1
R2
6
2
τ3
4
R3
4
τ6
(Low)
τ4
5

Example

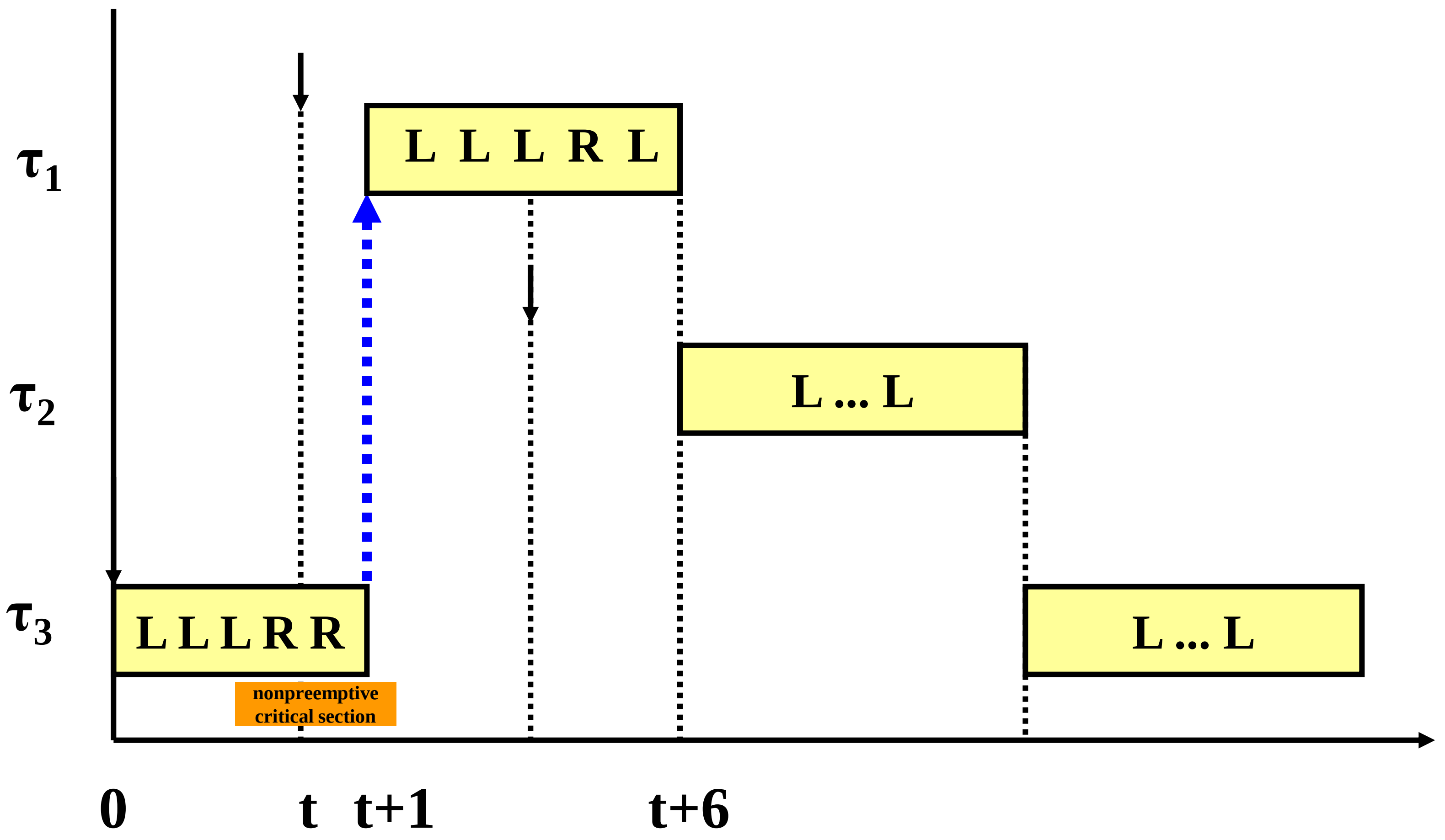
$B_1 = \max\{6,2\} = 6$, $B_2 = \max\{5\} + \max\{6,2\} = 11$, etc.



NonPreemptive Critical Sections (NPCS) (A. Mok)

- All critical sections are executed nonpreemptively.
- When a job requests a resource, it is always allocated the resource.
- When a job holds a resource (in a critical section), it executes at a priority higher than any other task.
- Since no job is ever preempted when it holds a resource, deadlock can never occur.
- The **blocking time, B_i,** due to resource conflicts is the maximum critical section time of **any** lower priority tasks.

NonPreemptive Critical Sections



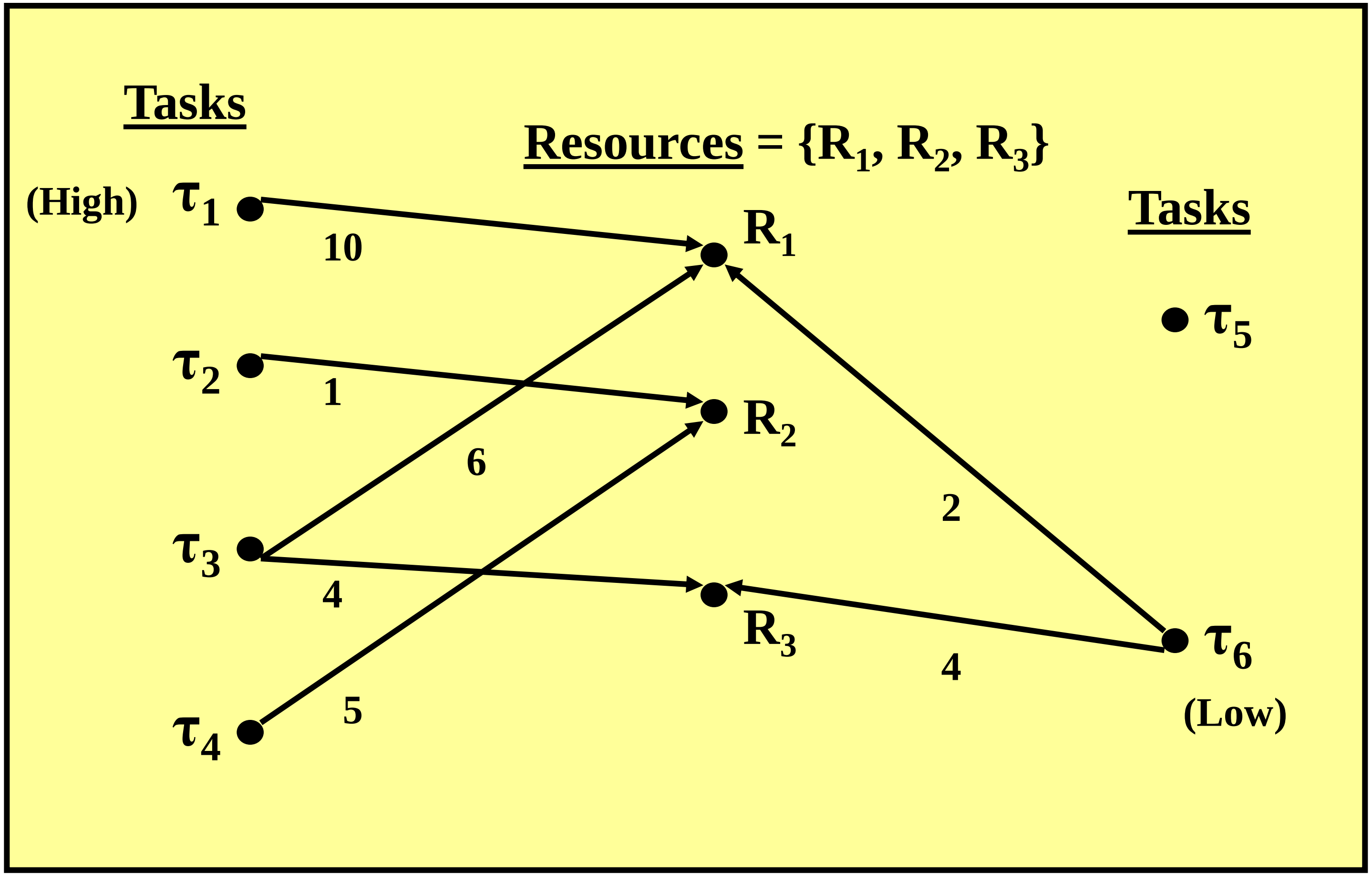
Blocking Time (NPCS)

- **Worst-Case Blocking Time** is given by:

$$B_i = \max_{j \in lp(i)} \{cs(j, r)\}$$

- Every task can be blocked by any lower priority task, **even when there is no resource sharing!**

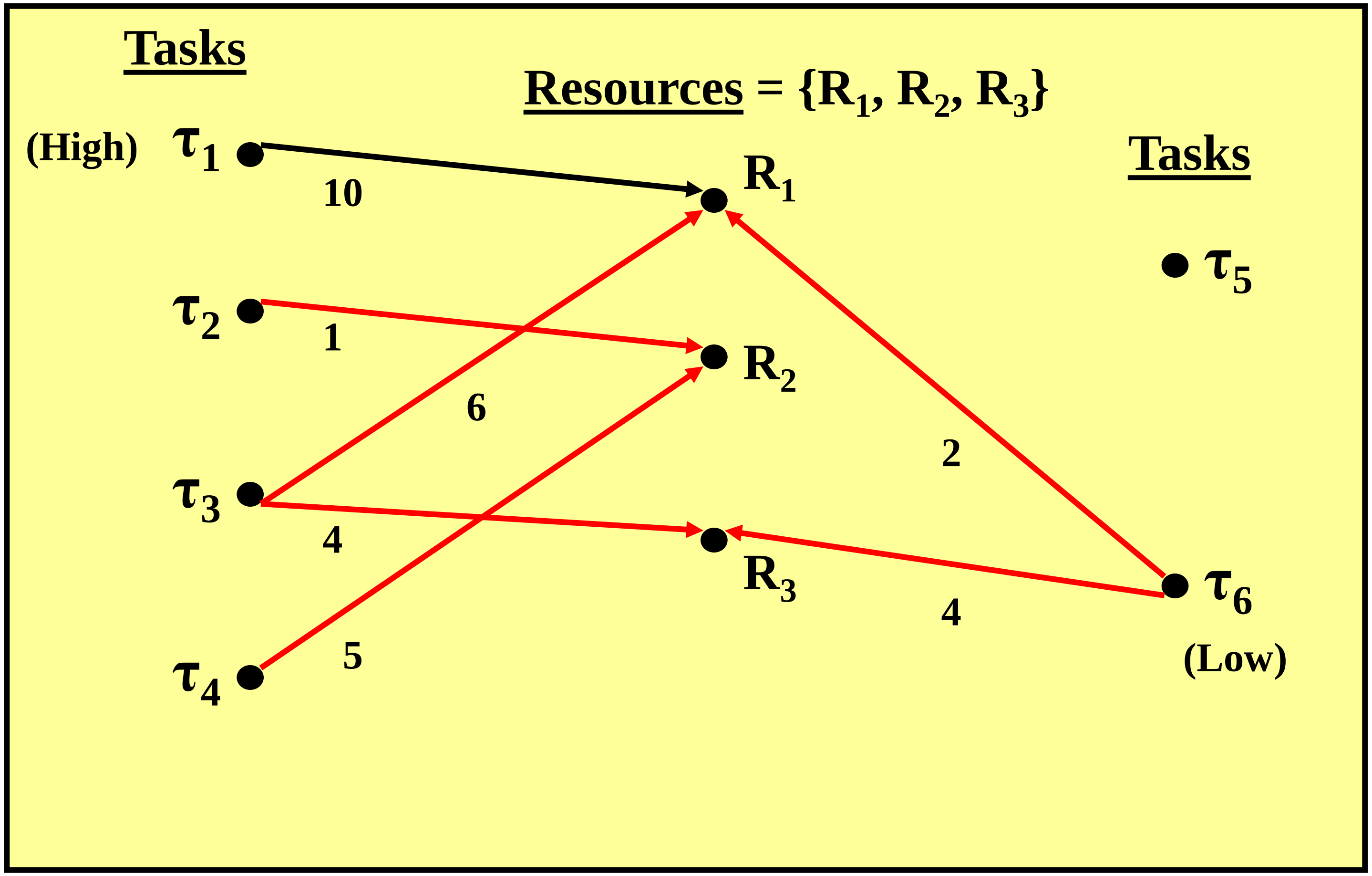
Example



Tasks
Resources = {R1, R2, R3}
Tasks
(High)
τ1
τ2
τ3
τ4
R1
R2
R3
τ5
τ6
(Low)
10
1
6
4
5
2
4

Example

$B_1 = \max\{1,6,4,5,2,4\} = 6$, $B_3 = \max\{5,2,4\} = 5$, $B_5 = \max\{2,4\} = 4$, etc.



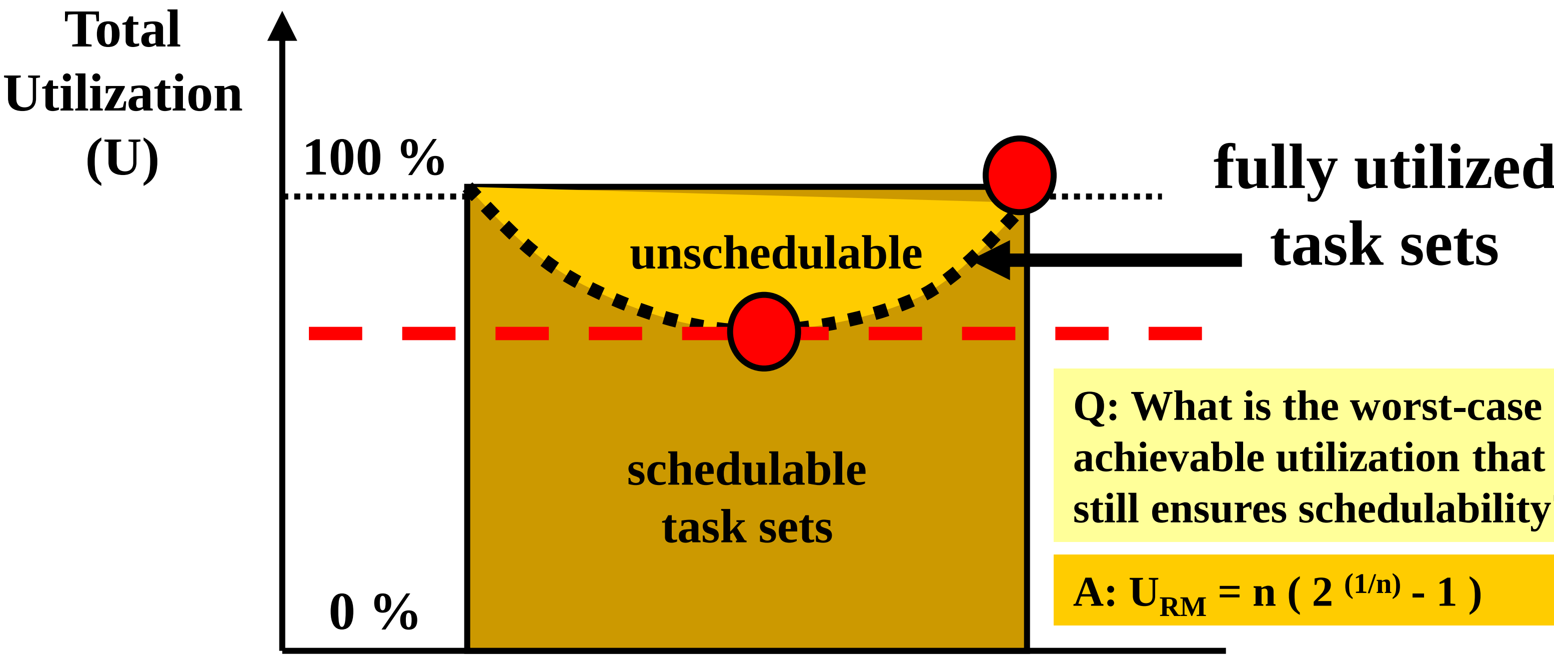
Utilization-Based Test

- Given a periodic task τ_i , the ratio $u_i = C_i / T_i$ is called the **utilization of task** τ_i .
- The **total utilization U** of all tasks in a system is the sum of the utilizations of all individual tasks:

$$\boxed{U = \sum_{i=1}^{n} \frac{C_i}{T_i}}$$

Utilization-Based Test

If $U + \mathbf{max}\{\boldsymbol{B_1 / T_1}, \boldsymbol{B_2 / T_2}, \ldots, \boldsymbol{B_n / T_n}\} \leq n\,(2^{1/n} - 1)$,
then the task set is feasible.



Q: What is the worst-case achievable utilization that still ensures schedulability?

A: $U_{RM} = n\,(2^{(1/n)} - 1)$

Priority Ceiling Protocol

- The priority ceiling of any resource is the highest priority of all the tasks requiring that resource.
- The current priority ceiling of the system is the highest priority ceiling of the resources currently locked.
- A task that requires no critical section resources proceeds according to the traditional approach

Original Priority Ceiling Protocol

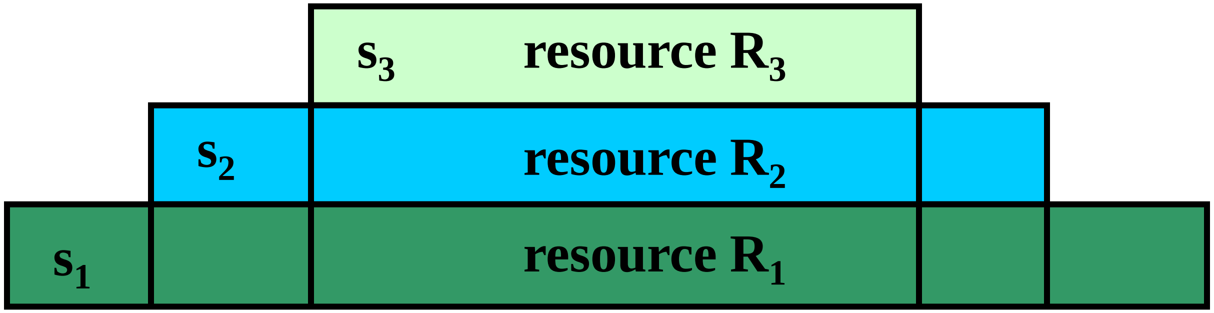
- When task A requests resource R,
  - If R is held by another task, it is blocked.
- If R is free,
  - If A’s priority is greater than the current system priority ceiling, A is granted access to R
  - If A’s priority is not greater than the current system priority ceiling, then it is blocked unless A holds resources whose priority ceiling equals the system priority ceiling.
  - Blocking tasks inherit the priority of the tasks they block (as in the priority inheritance protocol)

Priority Ceiling Protocols (PCP)

- A higher priority task can be blocked **at most once** during each job by a lower priority task.
- Deadlocks are prevented.
- Transitive blocking is prevented.
- Mutually exclusive access to shared resources is supported.

Semaphore Requirements

- Tasks cannot hold locks on a semaphore between invocations of a job.
- Tasks must lock and unlock semaphores in a "nested" or "pyramid" fashion:
  - Example: $P(s_1);P(s_2);P(s_3);...;V(s_3);V(s_2);V(s_1);$
  - **Notation: $[R_1; 6\ [R_2; 4\ [R_3; 2]\ ]\ ]$**



Original PCP

- The protocol uses the notion of a system-wide semaphore ceiling.
- Each task has a static default priority assigned.
- Each resource (semaphore) has a static ceiling priority defined to be the maximum static priority of any task that uses it.
- A task has a dynamic priority equal to the maximum of its own default priority and any priority it inherits due to blocking a higher priority task.

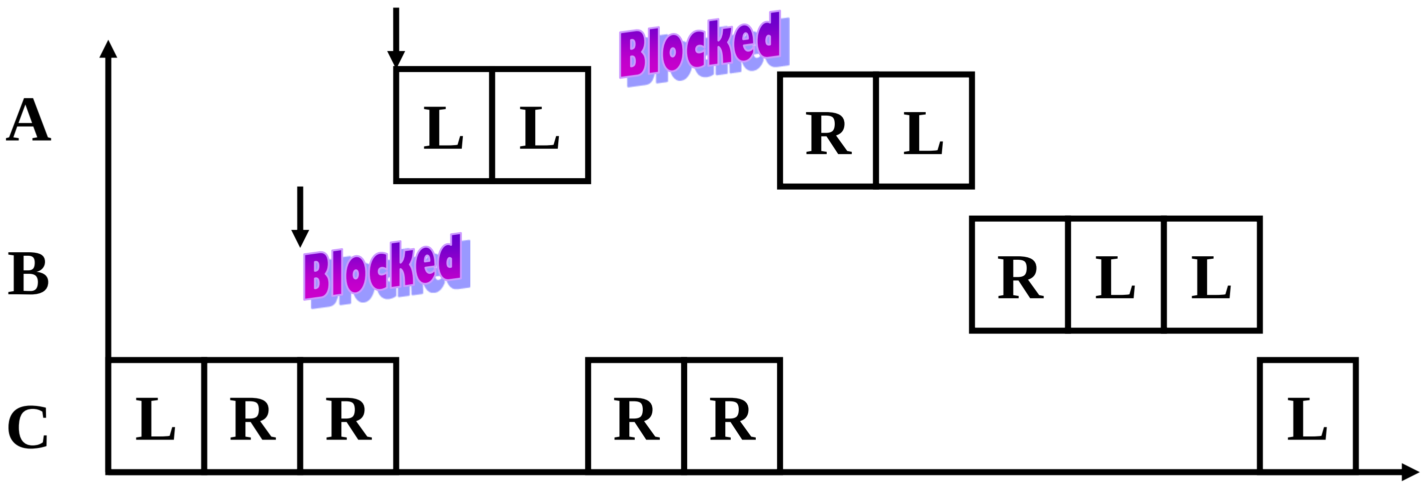
Original PCP

- At run-time, if a task wants to lock a semaphore s, its priority must be strictly higher than the ceilings of all semaphores currently locked by other tasks (unless it is the task holding the lock on the semaphore with the highest ceiling).
- If this condition is not satisfied, then the task is blocked on s.
- When a task is blocked on semaphore s, the task currently holding s inherits the priority of the blocked task.

Example #1

Task A: LLRL, priority = 3 (high), arrival time = 3
Task B: RLL, priority = 2 (medium), arrival time = 2
TaskC: LRRRRL, priority = 1 (low), arrival time = 0

L = local computation, R = access shared resource



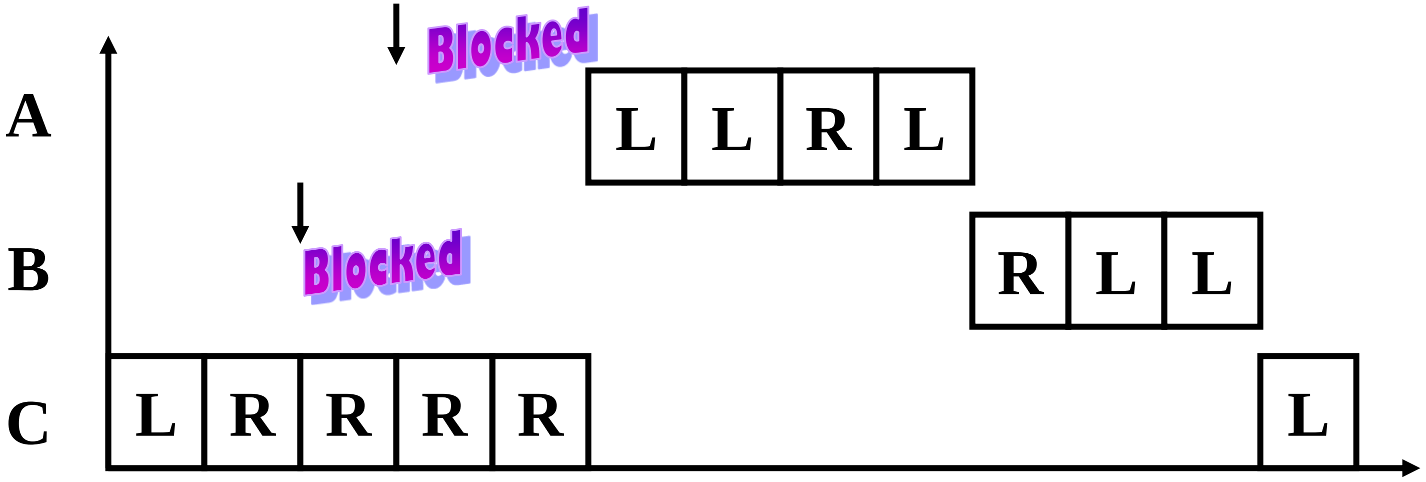
Immediate PCP (IPCP) (Ceiling-Priority Protocol)

- Each task has a static default priority assigned.
- Each resource has a static ceiling value defined.
- Each task has a dynamic priority that is the maximum of its own static priority and the ceiling value of **any resource it has locked**.
- Blocking occurs **prior to the initial execution** of a task during each invocation. Tasks should not voluntarily suspend themselves while holding a resource because no job is ever blocked once its execution begins.

Example #2 (IPCP)

Task A: LLRL, priority = 3 (high), arrival time = 3
Task B: RLL, priority = 2 (medium), arrival time = 2
TaskC: LRRRRL, priority = 1 (low), arrival time = 0

L = local computation, R = access shared resource



Priority Ceiling Protocols

- Each task has a static default priority.
- Each resource has a static ceiling priority equal to the maximum priority of all tasks that use it.
- A task has a dynamic priority equal to the maximum of its own default priority and any priority it inherits due to **blocking** a higher priority task.
- The difference is **when does** the dynamic priority change:
  - **immediately (before execution)** (IPCP), or
  - **when blocking occurs** (OPCP).

Blocking Time (PCP)

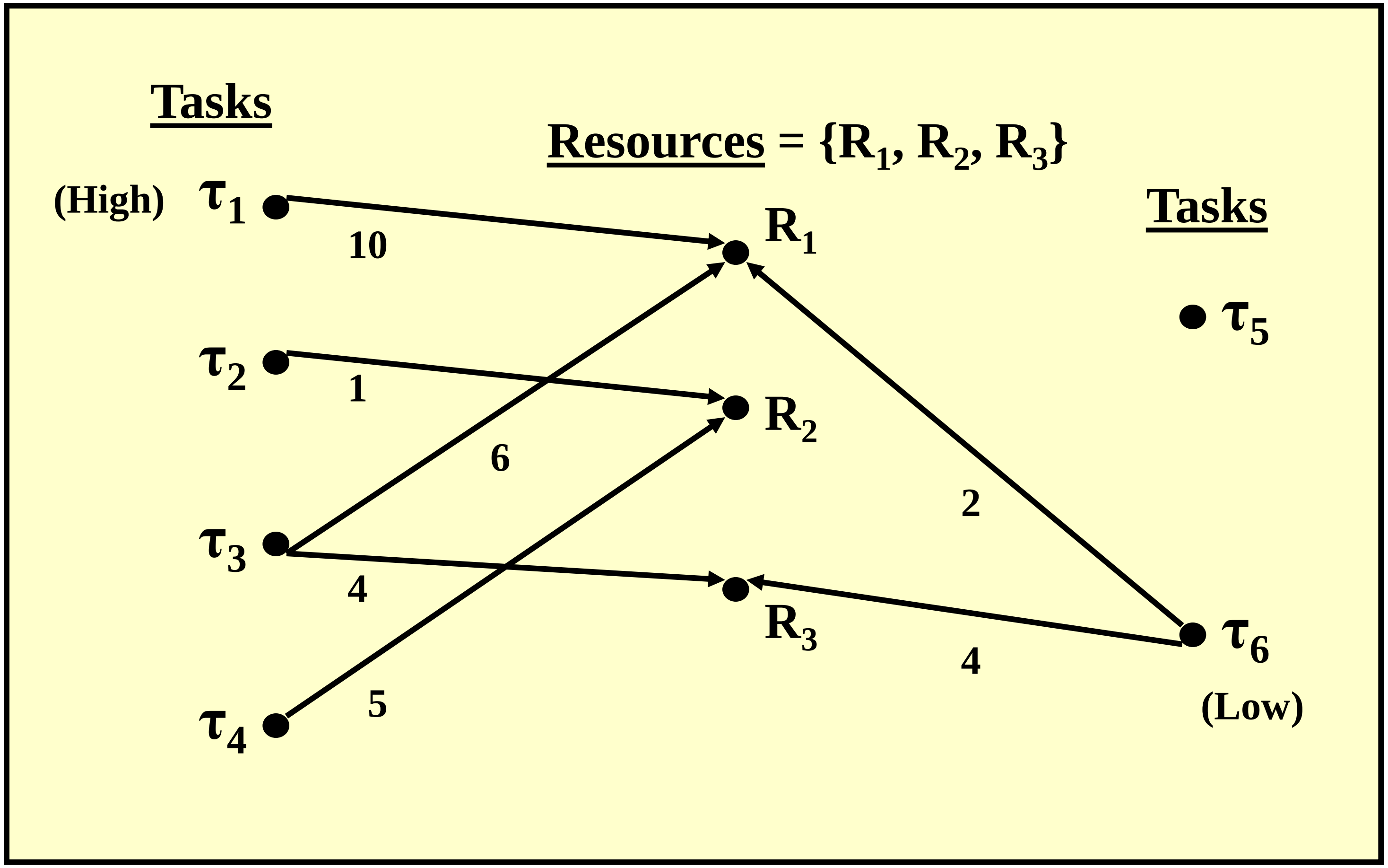
- The worst-case blocking time is at most the duration of one critical section time:

$$B_i = \max_{\substack{j \in lp(i) \\ r \in res_j \\ ceil(r) \geq \pi_i}} \{cs(j, r)\}$$

Types of Blocking

- Direct Blocking
- Priority Inheritance Blocking
- Priority-Ceiling Blocking

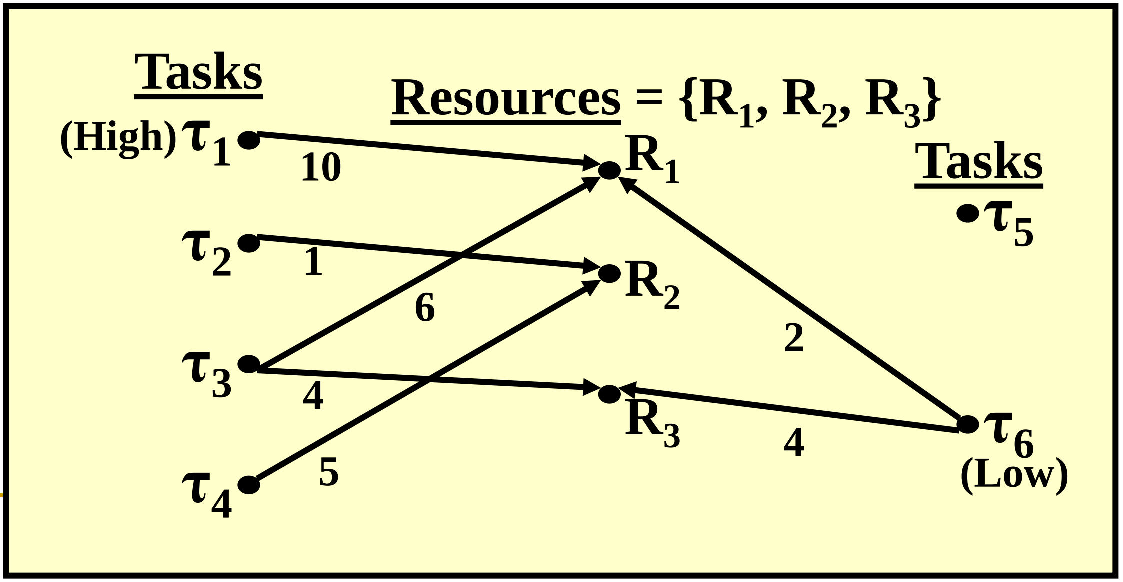
Example



Tasks
Resources = {R_1, R_2, R_3}
Tasks
(High)
τ_1
τ_2
τ_3
τ_4
τ_5
τ_6
(Low)
R_1
R_2
R_3
10
1
6
4
5
2
4

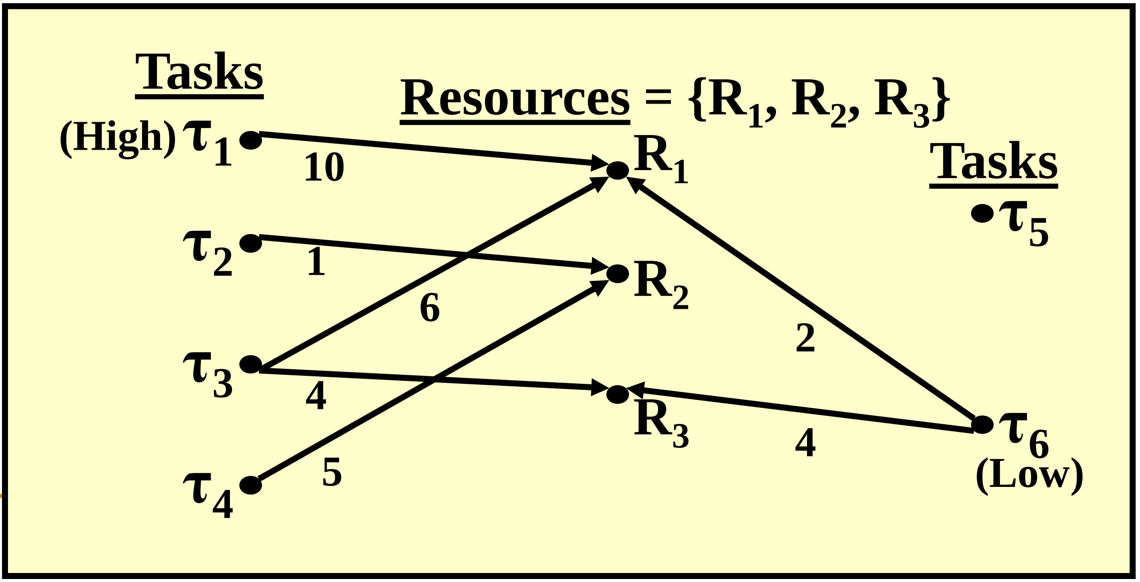
Direct Blocking

Blocked By:		τ_2	τ_3	τ_4	τ_5	τ_6
Task:	τ_1		6			2
	τ_2			5		
	τ_3					4
	τ_4					
	τ_5					



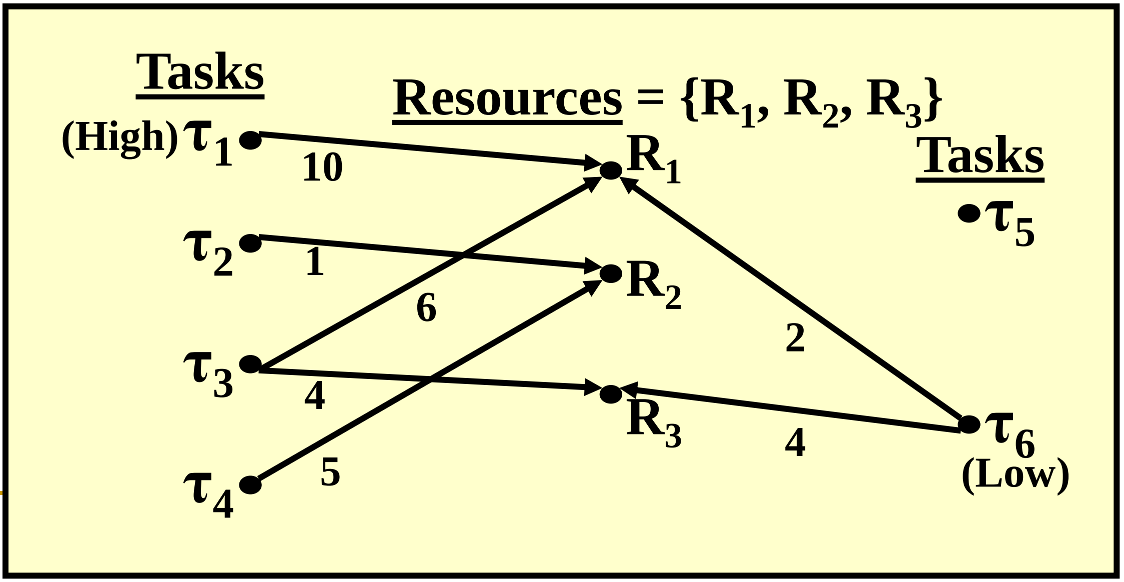
Priority Inheritance Blocking

Blocked By:		τ_2	τ_3	τ_4	τ_5	τ_6
Task:	τ_1					
	τ_2		6			2
	τ_3			5		2
	τ_4					4
	τ_5					4



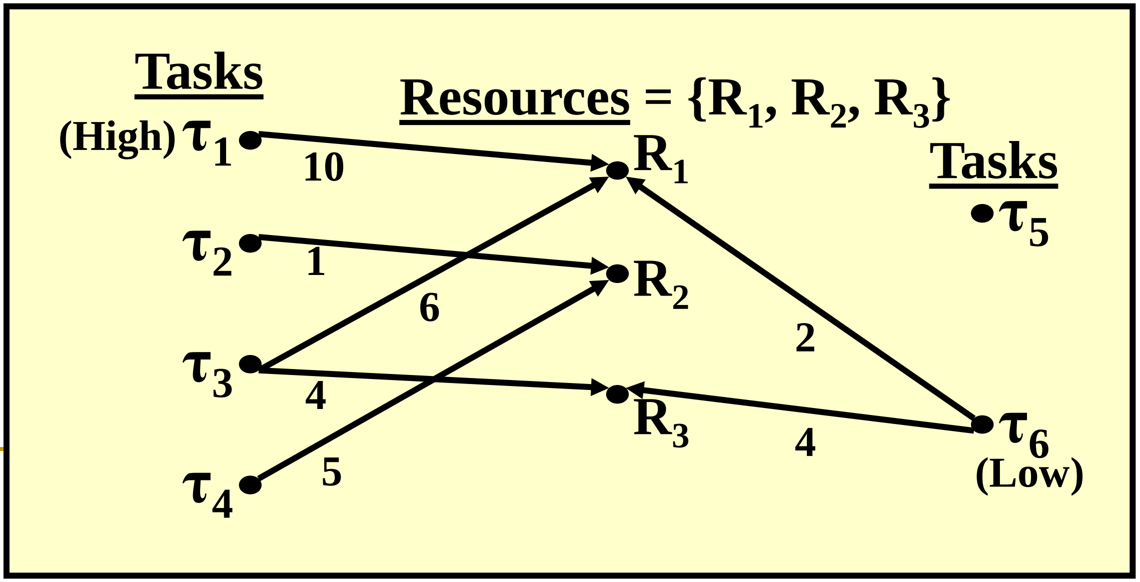
Priority Ceiling Blocking

Blocked By:		τ_2	τ_3	τ_4	τ_5	τ_6
Task:	τ_1					
	τ_2		6			2
	τ_3			5		2
	τ_4					4
	τ_5					



B_i = maximum row value

Task	Worst-Case Blocking Time
τ_1	6
τ_2	6
τ_3	5
τ_4	4
τ_5	4



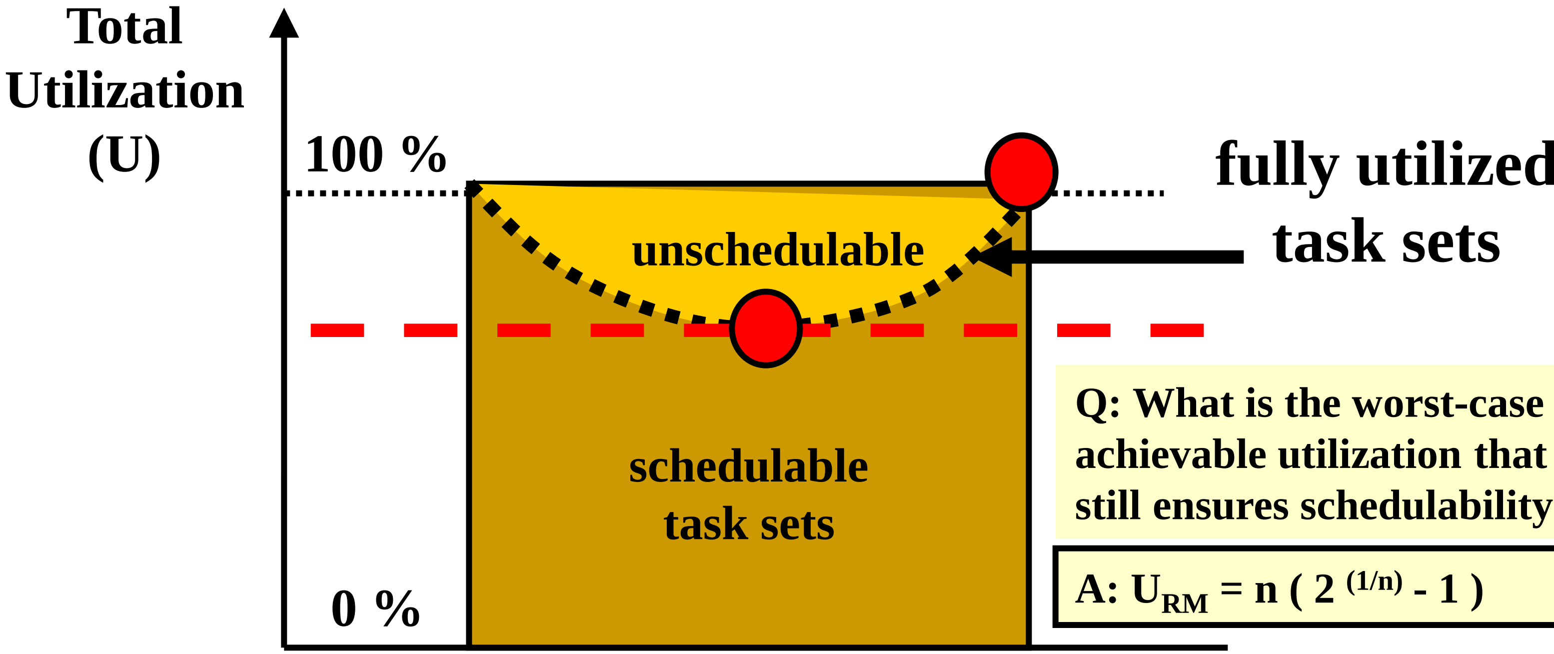
Utilization-Based Test

- Given a periodic task τ_i , the ratio $u_i = C_i / T_i$ is called the **utilization of task** τ_i .
- The **total utilization U** of all tasks in a system is the sum of the utilizations of all individual tasks:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

Utilization-Based Test

If $U + \max\{B_1/T_1, B_2/T_2, \ldots, B_n/T_n\} \le n(2^{1/n} - 1)$, then the task set is feasible.



Q: What is the worst-case achievable utilization that still ensures schedulability?

A: $U_{RM} = n(2^{(1/n)} - 1)$

Example

Task τ_i	Period T_i	Blocking Time B_i	Run-Time C_i
τ_1	100	20	40
τ_2	150	30	40
τ_3	350	0	100

$U = 40 / 100 + 40 / 150 + 100 / 350$
$= 0.4 + 0.267 + 0.286 = 0.953$, so
$U + \max\{B_1 / T_1 , B_2 / T_2 , \ldots , B_n / T_n \}$
$= 0.953 + 0.2 > 1.0$.
Consequently, the test is **inconclusive**.

Response Time Analysis

- The **response time** (R_i) for task τ_i is given by the implicit equation:

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil * C_j$$

- Solve by forming a recurrence relation:

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil * C_j$$

$$w_i^0 = C_i + B_i$$

Note: Arbitrary phasing is assumed.

Solving The Recurrence

- The sequence $w_i^0, w_i^1, w_i^2, \ldots, w_i^n$

  is clearly non-decreasing:

  - If $w_i^{n+1} = w_i^n$, then a fixed point (solution) has been found.

  - If $w_i^{n+1} > T_i$, then no solution exists.

Algorithm

Input: $C_1, C_2, \ldots, C_m, B_1, B_2, \ldots, B_m, T_1, T_2, \ldots, T_m$

Output: $R_1, R_2, \ldots, R_m$

for $i = 1$ to m

$n = 0$

$w_i^n = C_i + B_i$

loop

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil * C_j$$

if $w_i^{n+1} = w_i^n$ then

$R_i = w_i^n$

break out of loop {solution found}

if $w_i^{n+1} > T_i$ then

break out of loop {no solution}

$n = n + 1$

end loop

end for

Example

Task τ_i	Period T_i	Blocking Time B_i	Run-Time C_i
τ_1	100	20	40
τ_2	150	30	40
τ_3	350	0	100

$R_1 = C_1 + B_1 = 40 + 20 = 60 \leq D_1 = T_1 = 100.$

$R_2 = C_2 + B_2 + \ldots = 40 + 30 + \ldots = 150 \leq T_2 = 150.$

$R_3 = C_3 + B_3 + \ldots = 100 + 0 + \ldots = 300 \leq T_3 = 350.$

Consequently, the task set is feasible.

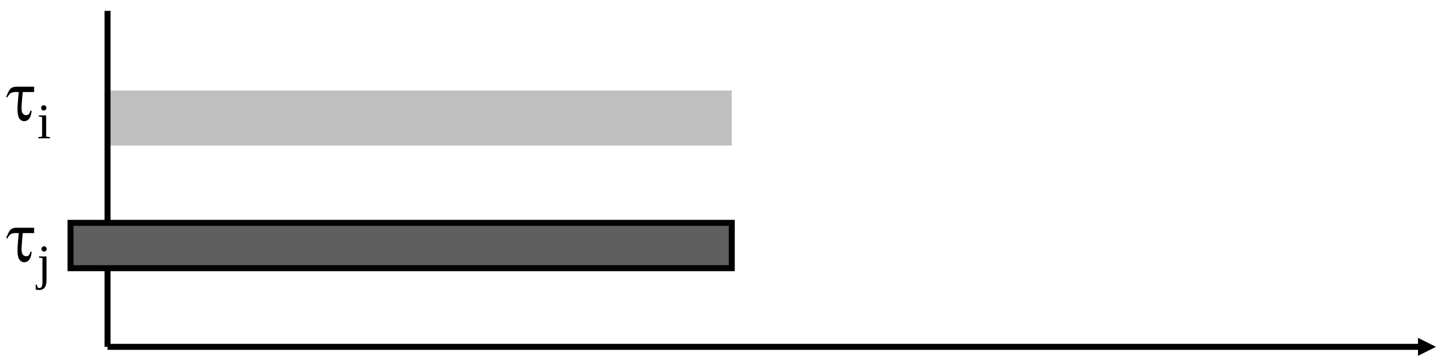
Preemption Thresholds: Three Step Process

- **Response Time Analysis**
  - Given assignment { (π_i, γ_i) | i = 1, 2, .. , n }, compute the worst-case response time (R_i or $WCRT_i$) for each task τ_i.
  - A task set Γ is schedulable iff $R_i \leq D_i$ for all i.
- Given a priority assignment { π_i | i = 1, .. , n }, determine a feasible set of **preemption thresholds**, if such a set exists.
- Use a branch and bound algorithm to search for a feasible assignment set of **priorities** (and preemption thresholds).

Response Time Analysis

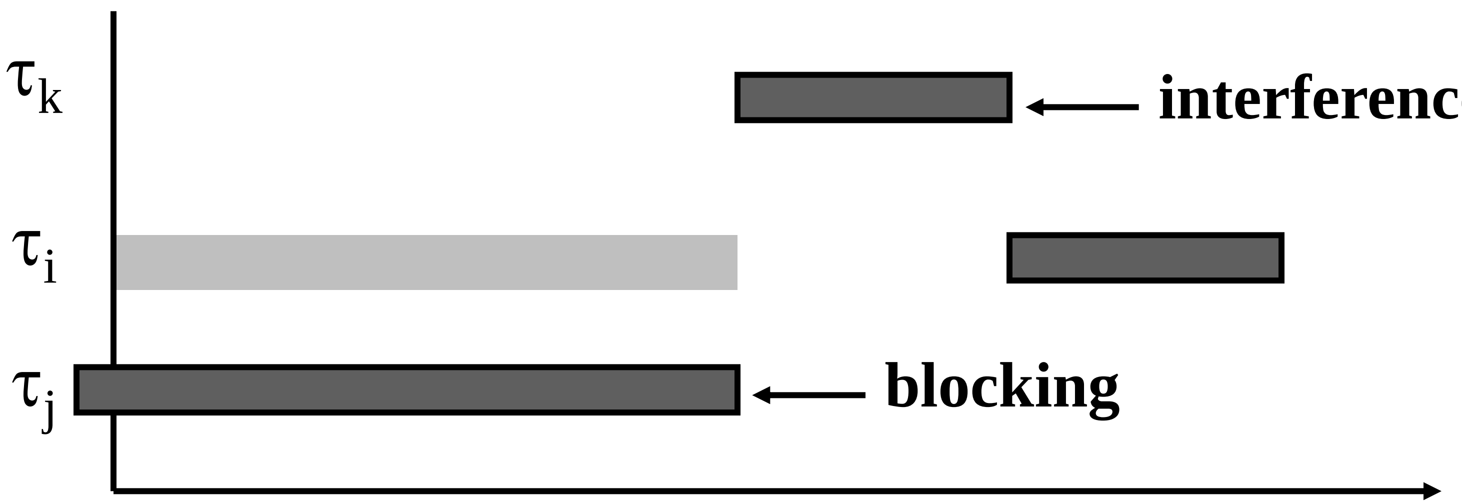
- The **blocking time** of task τ_i is denoted $B(\tau_i)$. Blocking occurs if a lower priority task is running and task τ_i cannot preempt it.

$$B(\tau_i) = \max_j \{ C_j | \gamma_j \geq \pi_i > \pi_j \}$$



Busy Period Analysis

- A **critical instant** occurs when all higher priority tasks arrive at the same time, and the task that contributes to the maximum blocking arrives at the critical instant - ε.



Divide Busy Period

- Divide the busy period for τ_i into two parts:
  - the length of time from the critical instant (time 0) to the point when τ_i starts executing its q^{th} job ($\mathbf{S_i(q)}$).
  - the length of time from the time τ_i starts executing its q^{th} job until it finishes executing its q^{th} job ($\mathbf{F_i(q)-S_i(q)}$).
- Let q = 1, 2, ... , m until we reach q = m s.t. $F_i(m) \leq m\ T_i$ that is, the m^{th} job completes before the next job is released.
- Then,

$$R_i = \max_{q\in\{1,..,m\}}\{F_i(q)-(q-1)T_i\}$$

Worst-Case Start Time ($\mathbf{S_i(q)}$)

$$S_i(q) = B(\tau_i) + (q\text{-}1)C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \pi_i}} \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor\right) C_j$$

Worst-Case Finish Time ($\mathbf{F_i(q)}$)

$$F_i(q) = S_i(q) + C_i + \sum_{\substack{j \in \{1,..,n\} \\ \pi_j > \gamma_i}} \left(\left\lceil \frac{F_i(q)}{T_j} \right\rceil - \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) \right) C_j$$

$$WCRT(\pi_i, \gamma_i)$$

Algorithm to compute R_i

Input: $C_1, \ldots, C_m, T_1, \ldots, T_m, \pi_1, \ldots, \pi_m, \gamma_1, \ldots, \gamma_m$

Output: $R_1, R_2, \ldots, R_m$

done = FALSE

q = 1

while (not done)

  compute S_i(q) and F_i(q)

  if $F_i(\mathrm{q}) \le \mathrm{q}\, T_i$ then

    done = TRUE

    m = q

  else

    q = q + 1

  end if

end while

$$R_i = \max_{\mathrm{q} \in \{1,..,\mathrm{m}\}} (F_i(\mathrm{q}) - (\mathrm{q}-1)\, T_i)$$

Summary

- Read Ch. 8.
- Read Sha's paper on priority inheritance protocols.