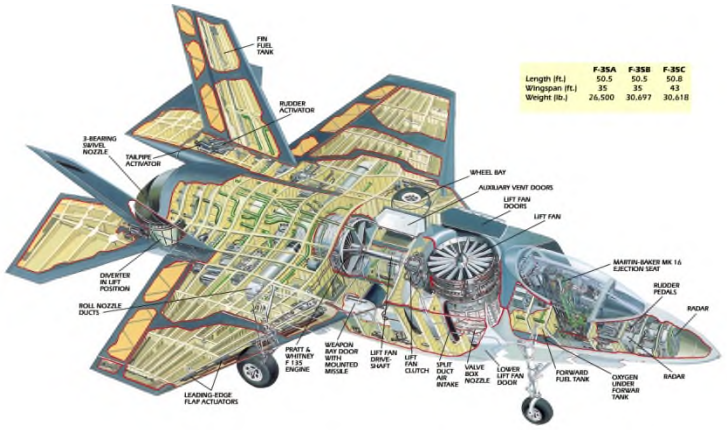
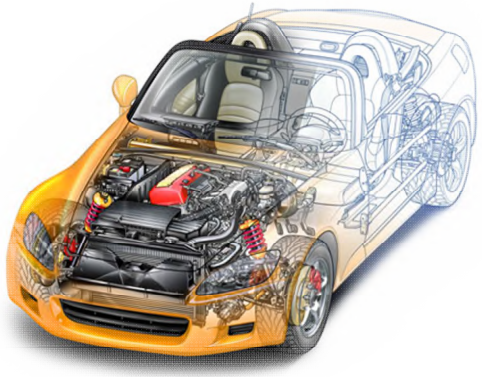
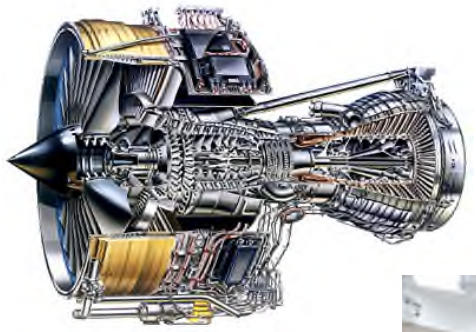
Scheduling Periodic Tasks
(in Embedded Systems)





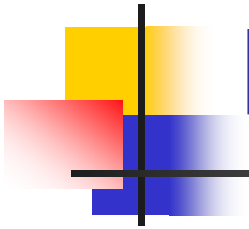


Embedded Computing Systems







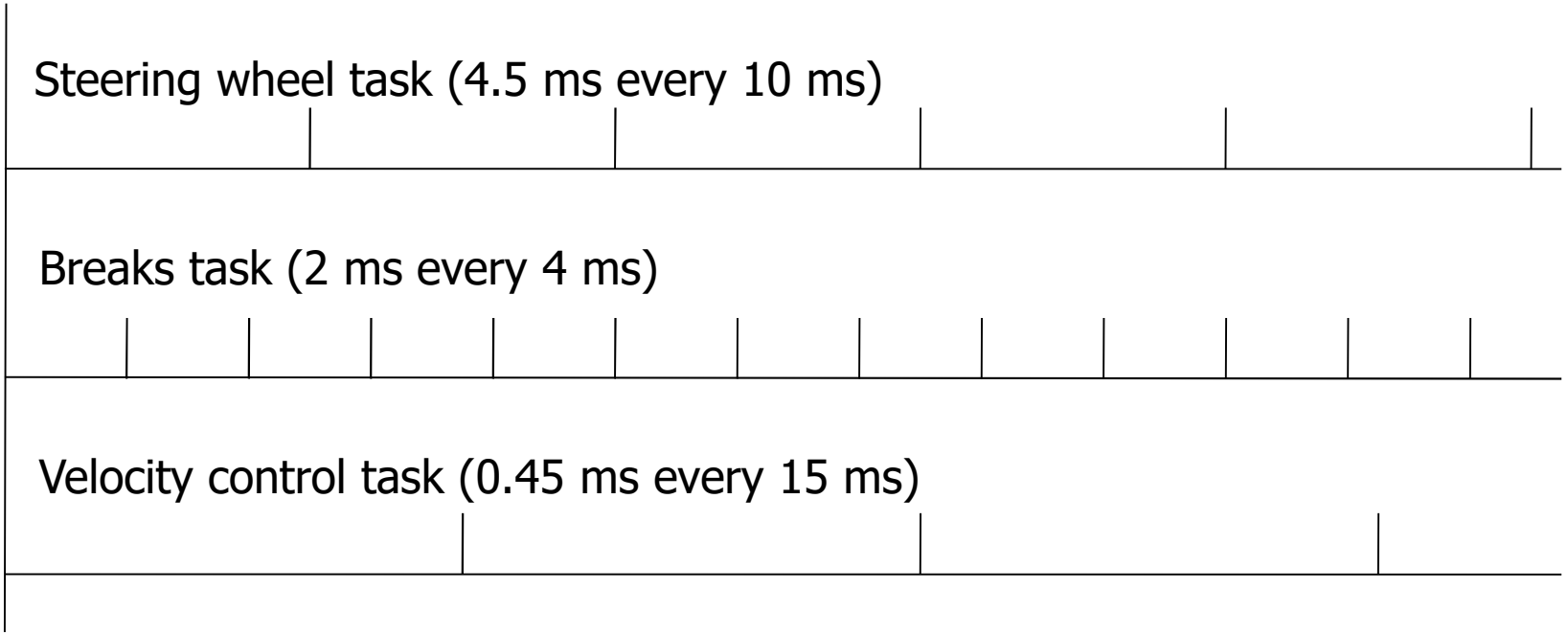


Drive-by-Wire Example

- Consider a control system in a drive-by-wire vehicle
  - Steering wheel sampled every 10 ms – wheel positions adjusted accordingly (computing the adjustment takes 4.5 ms of CPU time)
  - Breaks sampled every 4 ms – break pads adjusted accordingly (computing the adjustment takes 2ms of CPU time)
  - Velocity is sampled every 15 ms – acceleration is adjusted accordingly (computing the adjustment takes 0.45 ms)
  - For safe operation, adjustments must always be computed before the next sample is taken
- How to assign priorities?

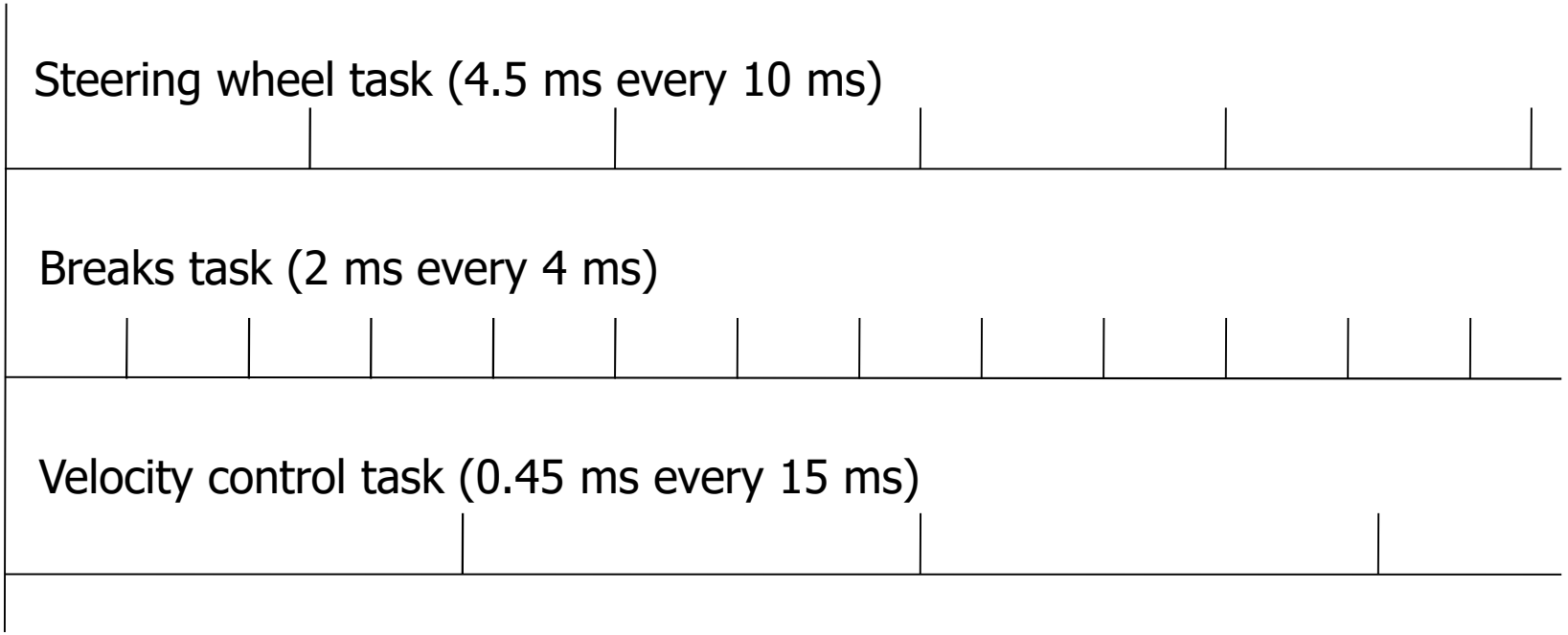
Drive-by-Wire Example

- Find a schedule that makes sure all task invocations meet their deadlines



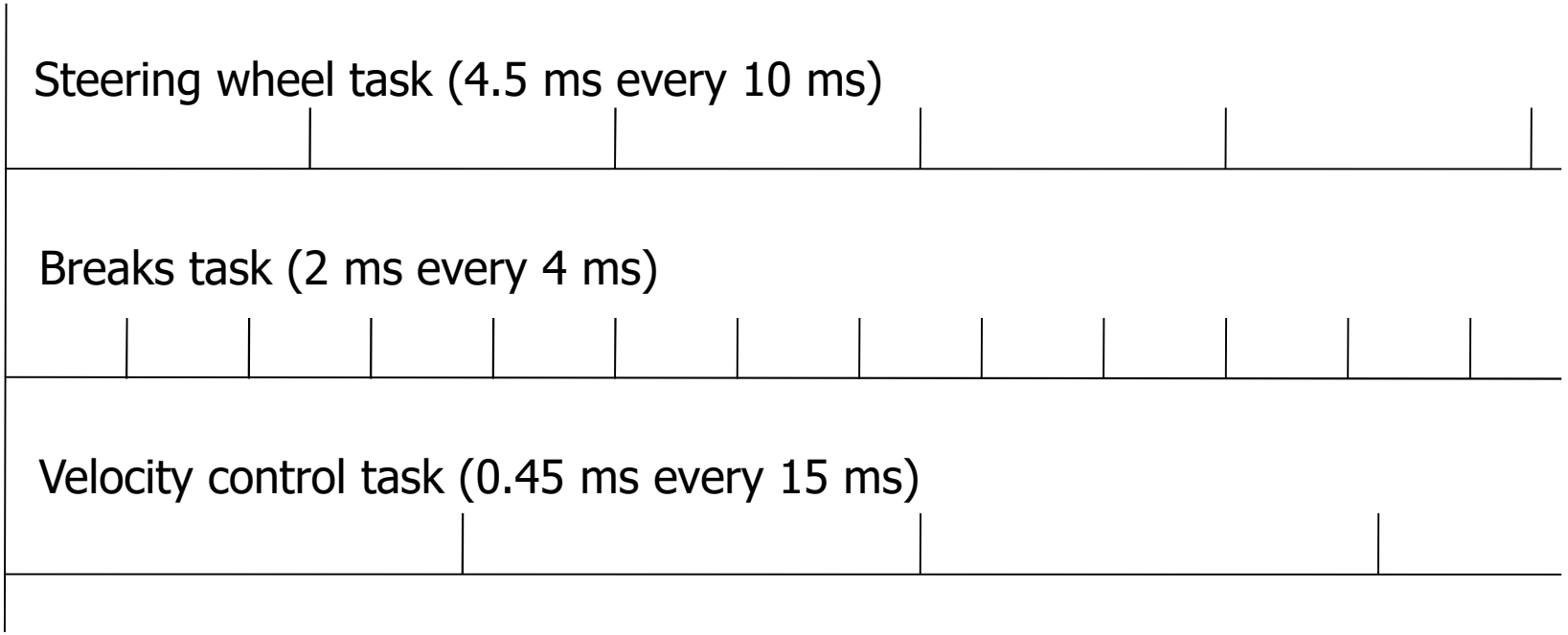
Drive-by-Wire Example

- Sanity check: Is the processor over-utilized? (e.g., if you have 5 homeworks due this time tomorrow, each takes 6 hours, then 5x6 = 30 > 24 → you are overutilized)



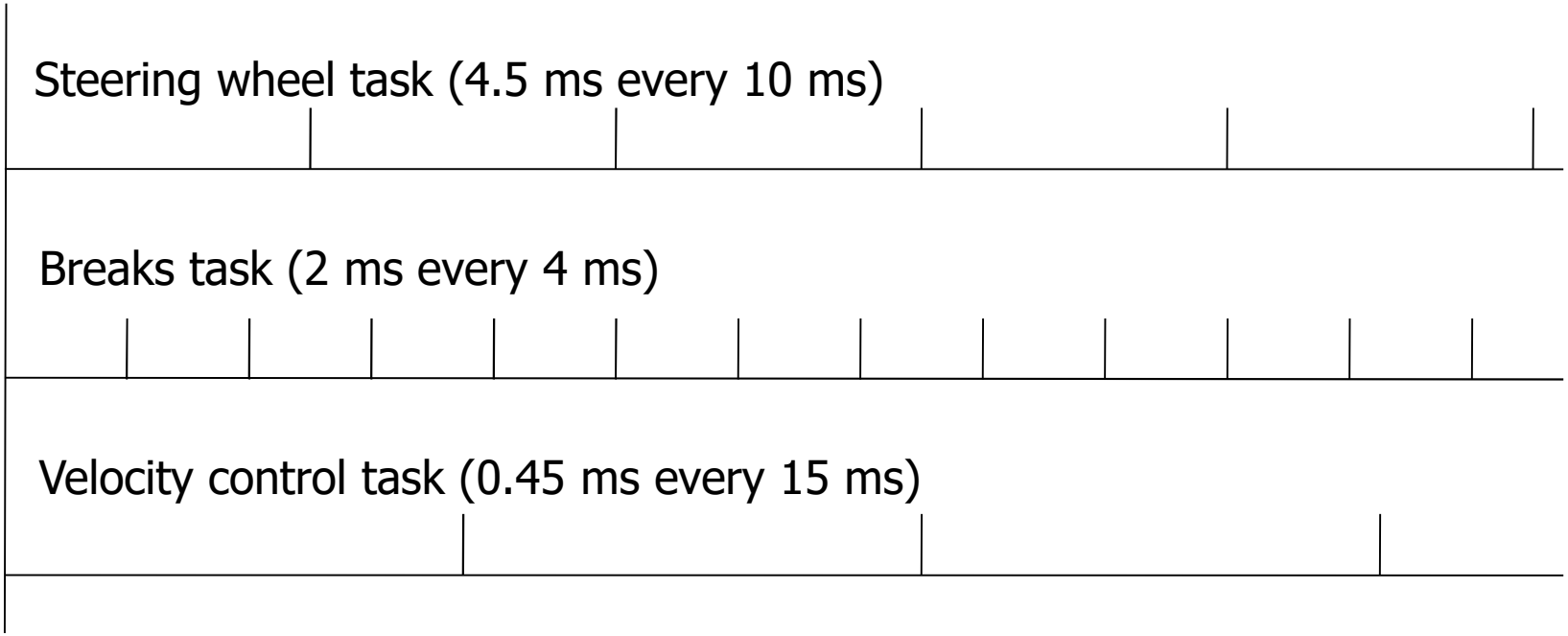
Drive-by-Wire Example

- Sanity check: Is the processor over-utilized? (e.g., if you have 5 homeworks due this time tomorrow, each takes 6 hours, then 5x6 = 30 > 24 → you are overutilized)
  - Hint: Check if processor utilization > 100%



Task Scheduling

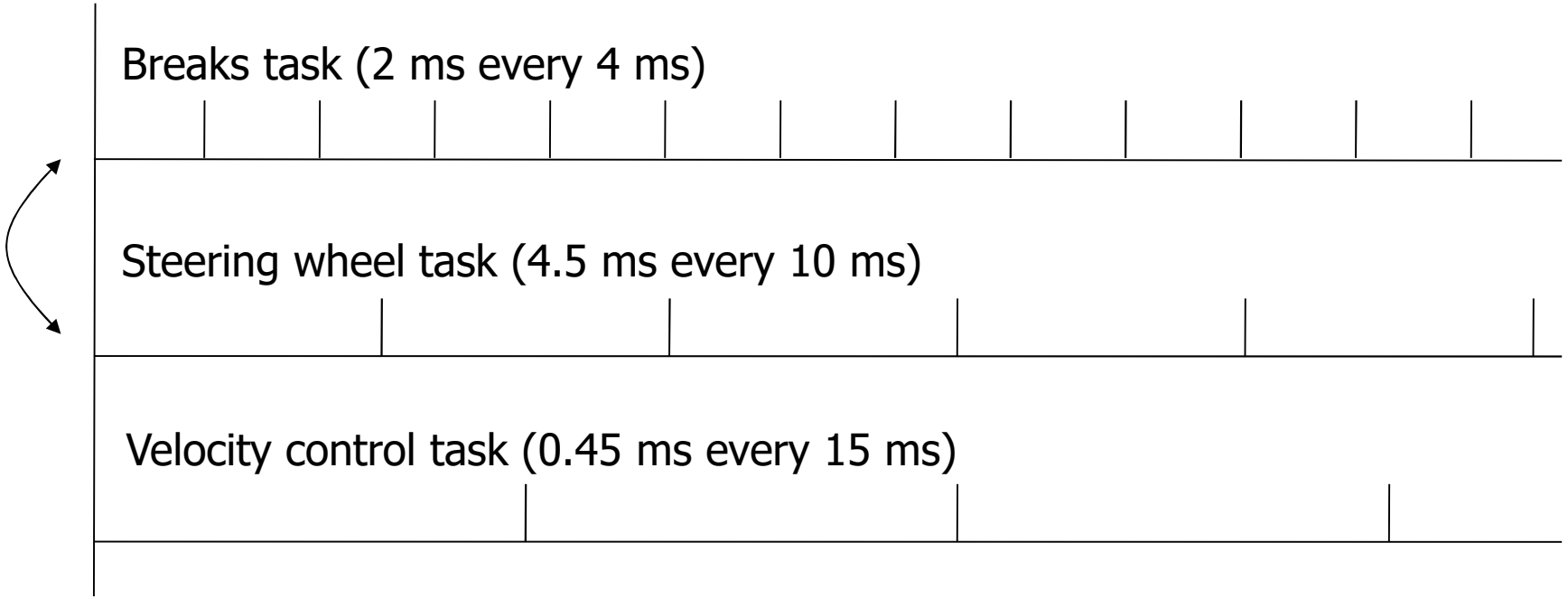
- How to assign task priorities?



How to assign priorities to tasks?

Task Scheduling

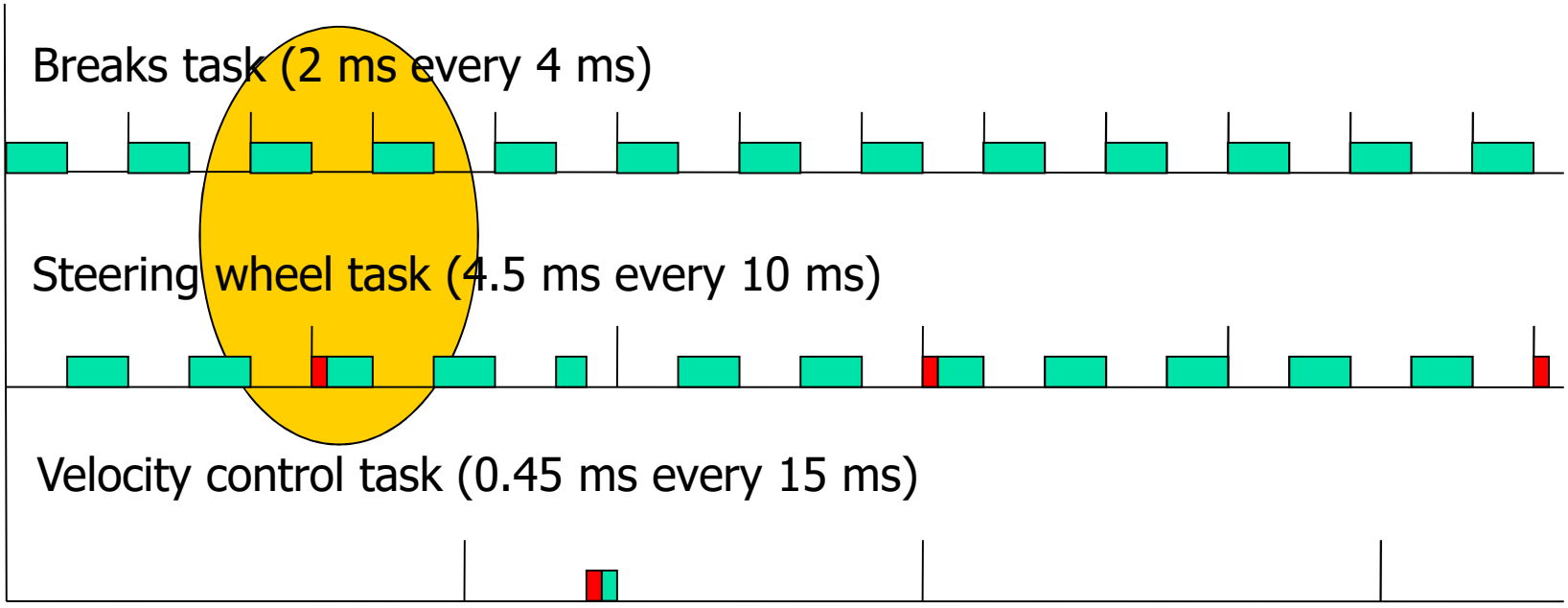
- How to assign task priorities?
  - Rate Monotonic (large rate = higher priority)



Intuition: Urgent tasks should be higher in priority

Problem?

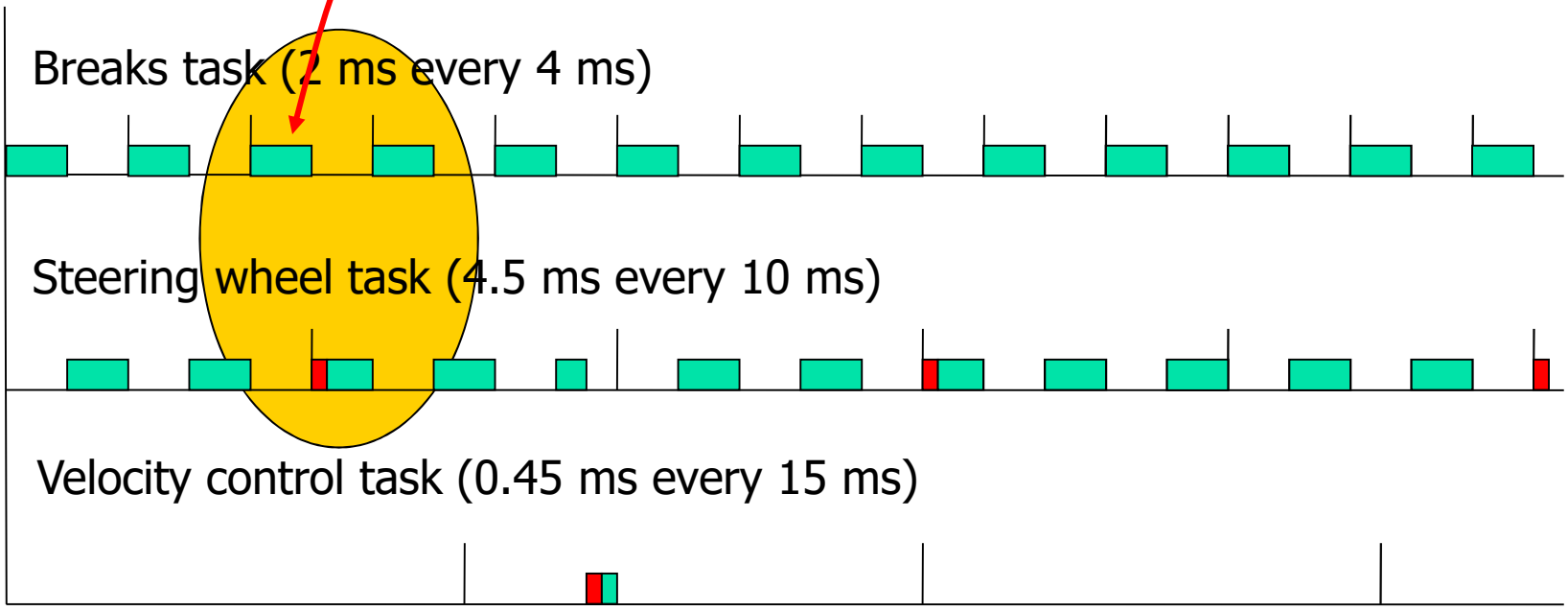
- Deadlines are missed!
- Average Utilization < 100%



Problem?

Fix:
Give this task invocation a lower priority

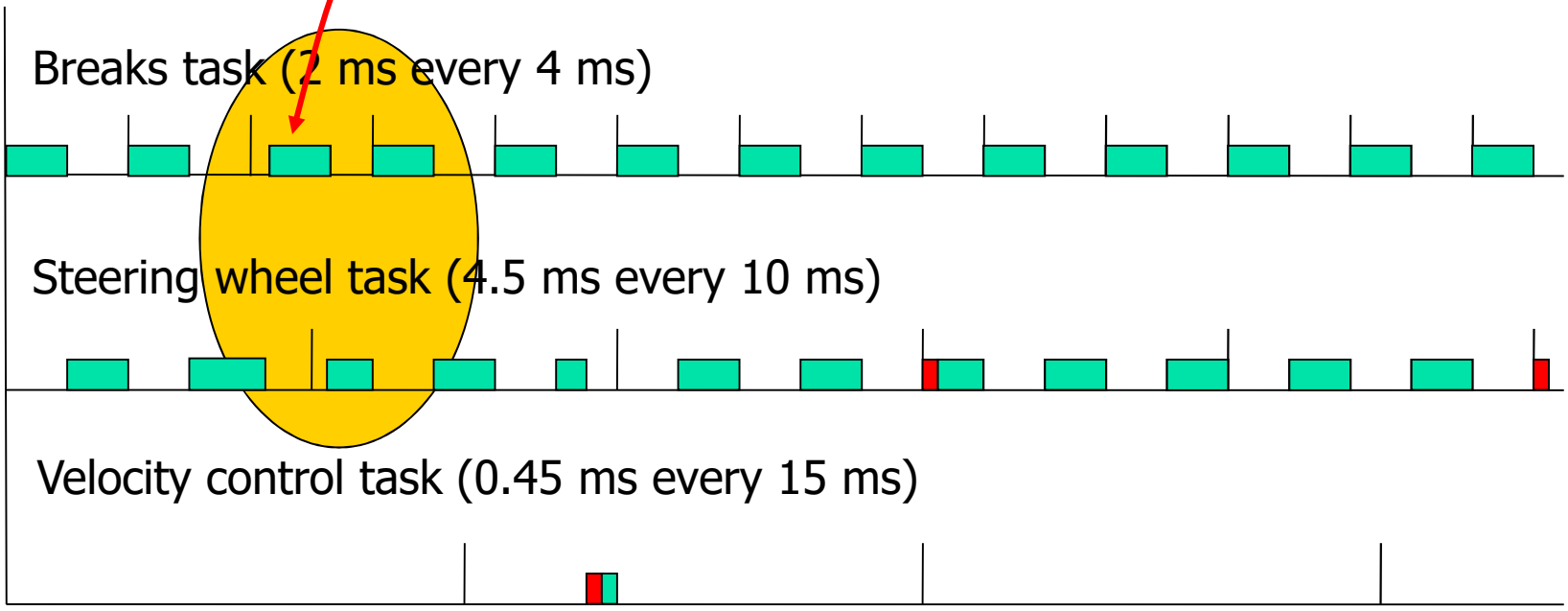
- Deadlines are missed!
- Average Utilization < 100%



Fix

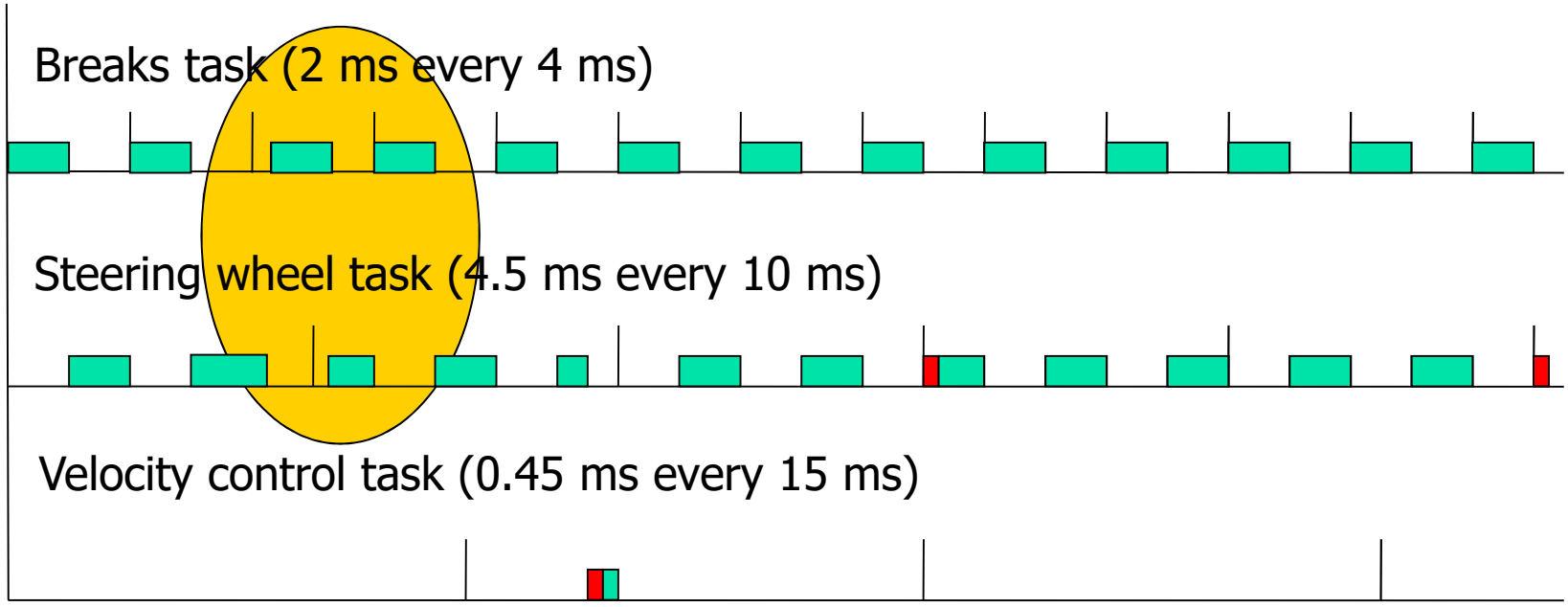
Fix:
Give this task invocation a lower priority

- Deadlines are missed!
- Average Utilization < 100%



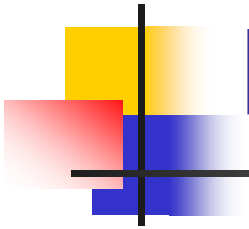
Task Scheduling

- Static versus Dynamic priorities?
  - Static: Instances of the same task have the same priority
  - Dynamic: Instances of same task may have different priorities

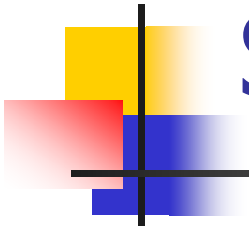


Intuition: Dynamic priorities offer the designer more flexibility and hence are more capable to meet deadlines

Real-time Scheduling of Periodic Tasks

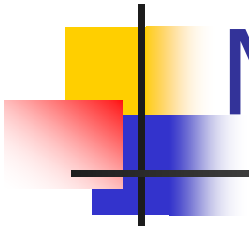


- Result #1: Earliest Deadline First (EDF) is the optimal dynamic priority scheduling policy for independent periodic tasks (meets the most deadlines of all dynamic priority scheduling policies)
- Result #2: Rate Monotonic Scheduling (RM) is the optimal static priority scheduling policy for independent periodic tasks (meets the most deadlines of all static priority scheduling policies)



Semaphores in POSIX

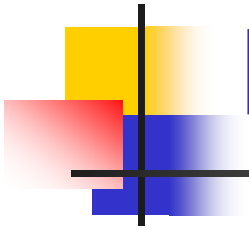
- int sem_init(sem_t *sem, int pshared, unsigned value);
- int sem_destroy(sem_t *sem);
- int sem_trywait(sem_t *sem);
- int sem_wait(sem_t *sem);
- int sem_post(sem_t *sem);



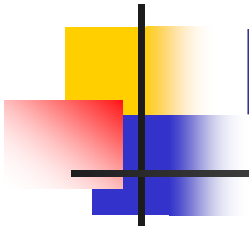
Mutex (Lock)

- Simplest and most efficient thread synchronization mechanism
- A special variable that can be either in
  - locked state: some thread *holds* or *owns* the *mutex;* or
  - unlocked state: no thread holds the *mutex*
- When several threads compete for a *mutex,* the losers block at that call
  - The *mutex* also has a queue of threads that are waiting to hold the *mutex.*

POSIX Mutex-related Functions



- int **pthread_mutex_init**(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr);
- int **pthread_mutex_destroy**(pthread_mutex_t *mutex);
- int **pthread_mutex_lock**(pthread_mutex_t *mutex);
- int **pthread_mutex_trylock**(pthread_mutex_t *mutex);
- int **pthread_mutex_unlock**(pthread_mutex_t *mutex);

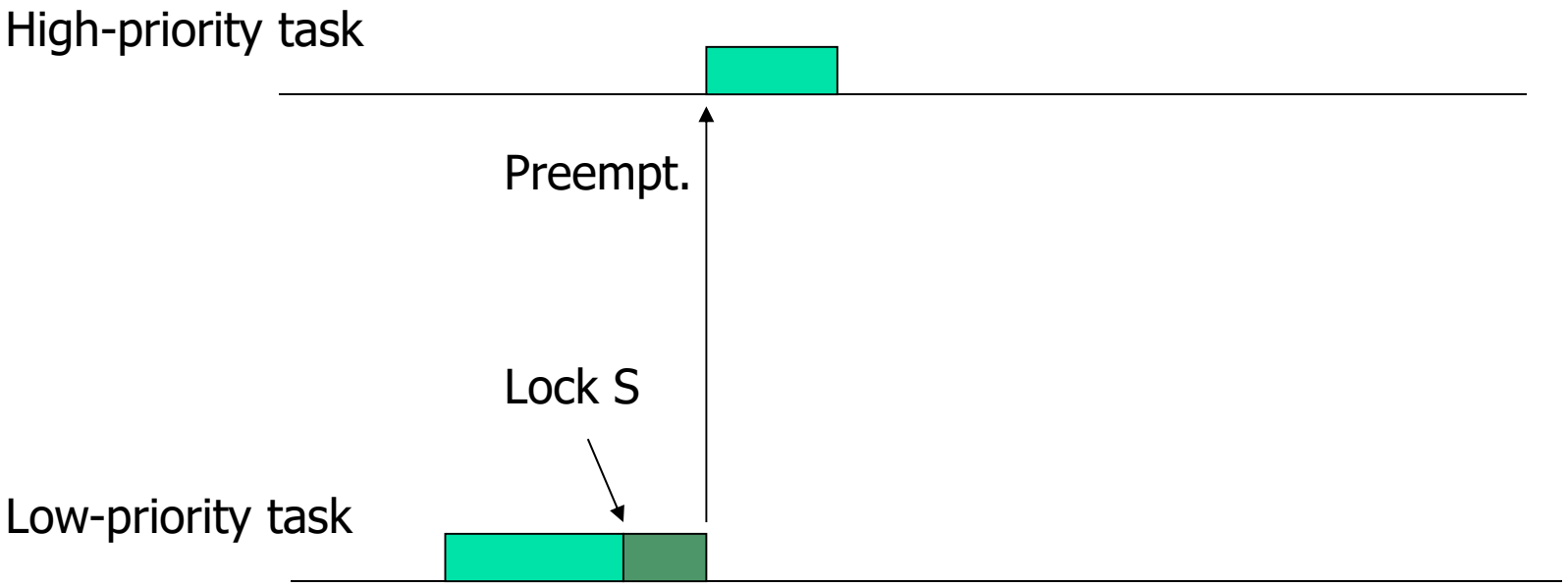


Locking and Priority Inversion

- What if a higher-priority process needs a resource locked by a lower-priority process?
  - How long will the higher priority process have to wait for lower-priority execution?

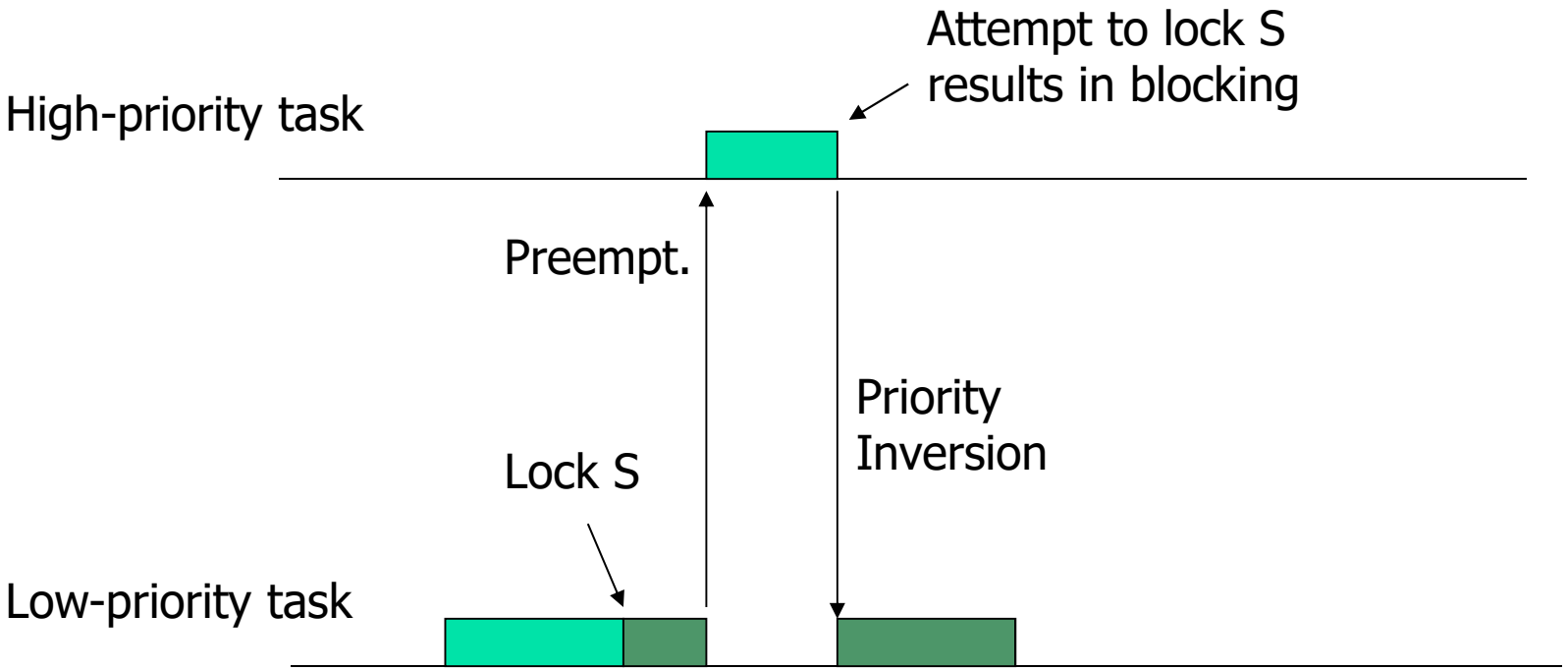
Priority Inversion

- Locks and priorities may be at odds. Locking results in priority inversion



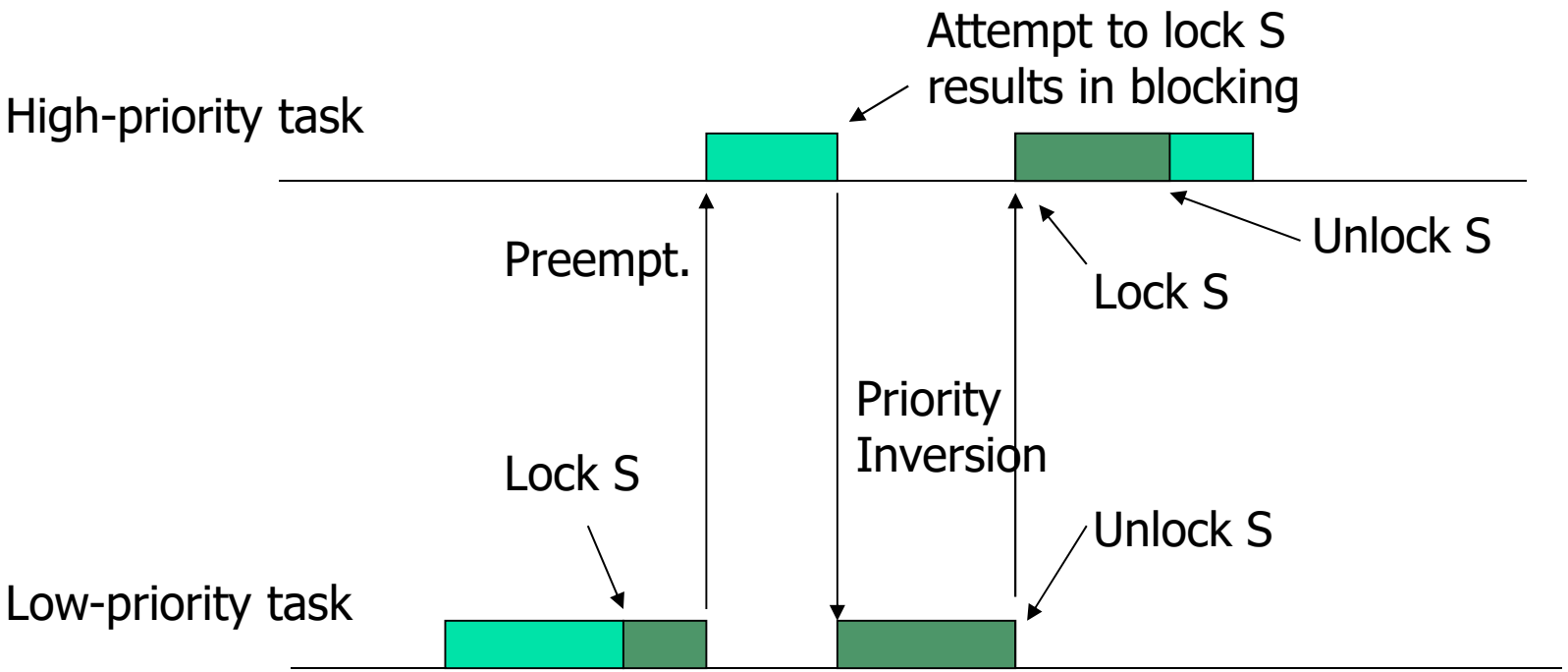
Priority Inversion

- Locks and priorities may be at odds. Locking results in priority inversion



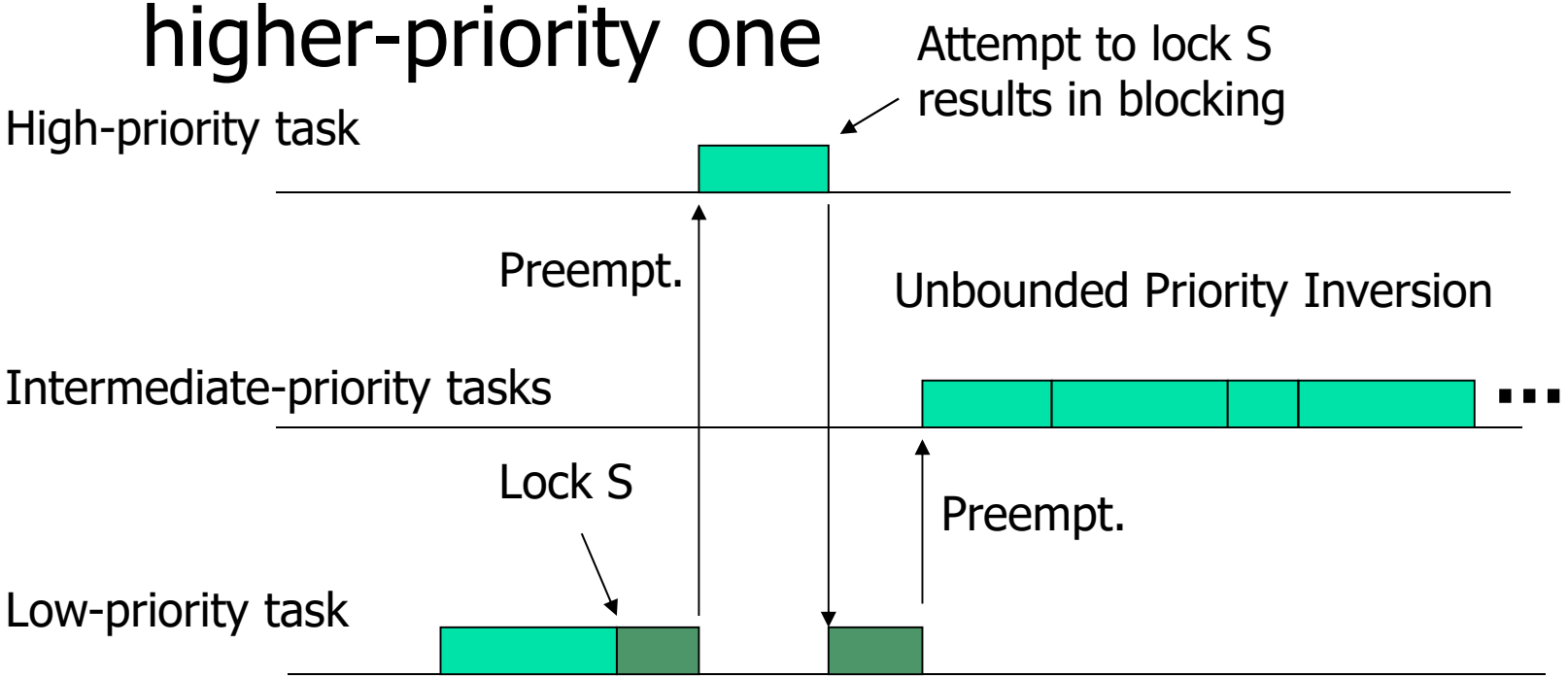
Priority Inversion

- How to account for priority inversion?



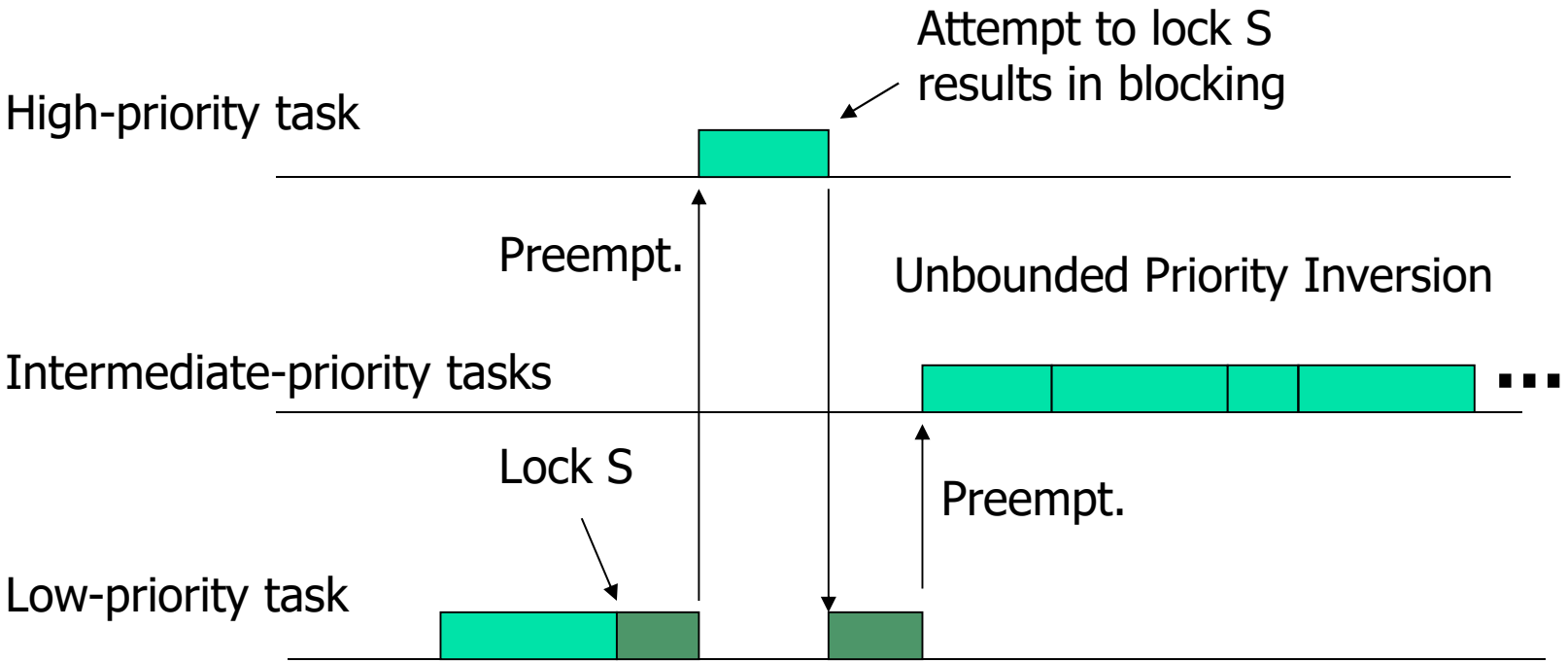
Unbounded Priority Inversion

- Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one



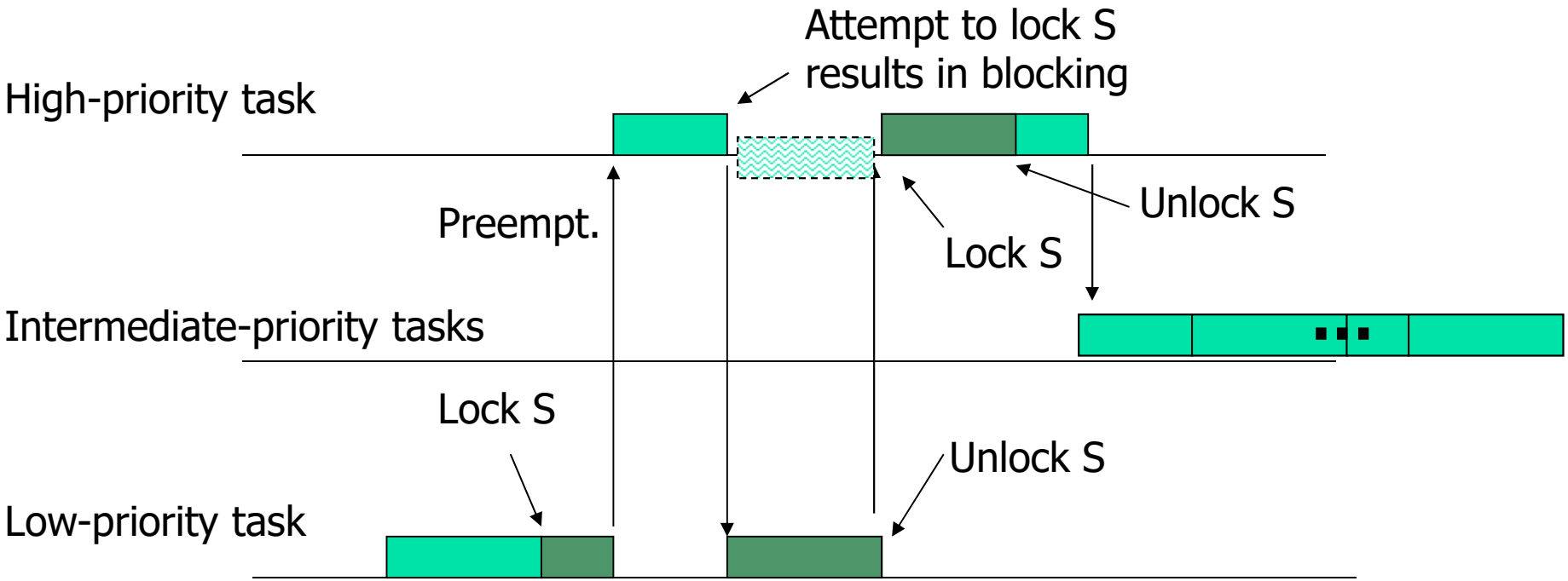
Unbounded Priority Inversion

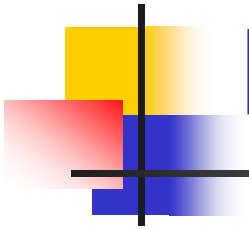
- How to prevent unbounded priority inversion?



Priority Inheritance Protocol

- Let a task inherit the priority of any higher-priority task it is blocking



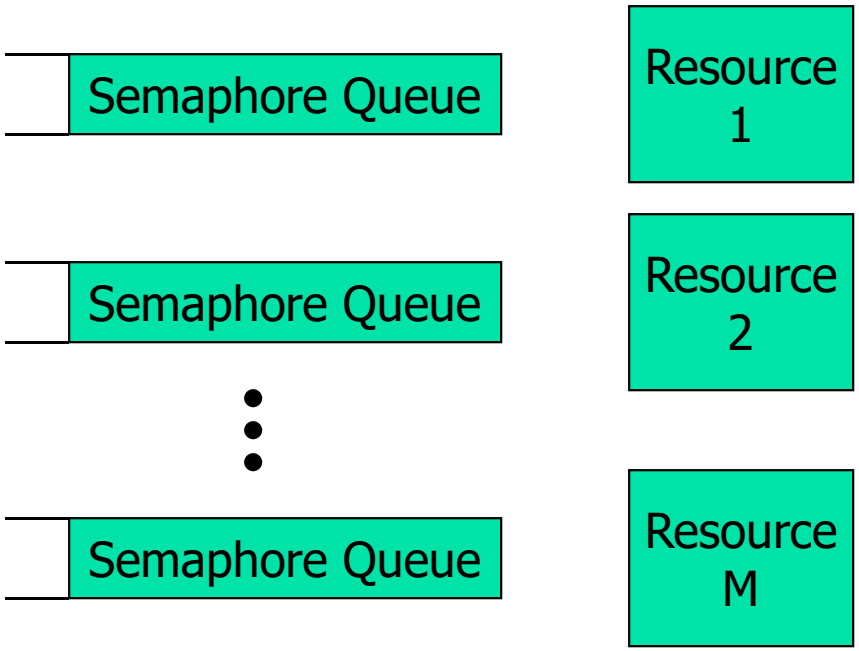


Priority Inheritance Protocol

- Question: What is the longest time a task can wait for lower-priority tasks?
  - Let there be N tasks and M locks
  - Let the largest critical section of task i be of length B_i
- Answer: ?

Computing the Maximum Priority Inversion Time

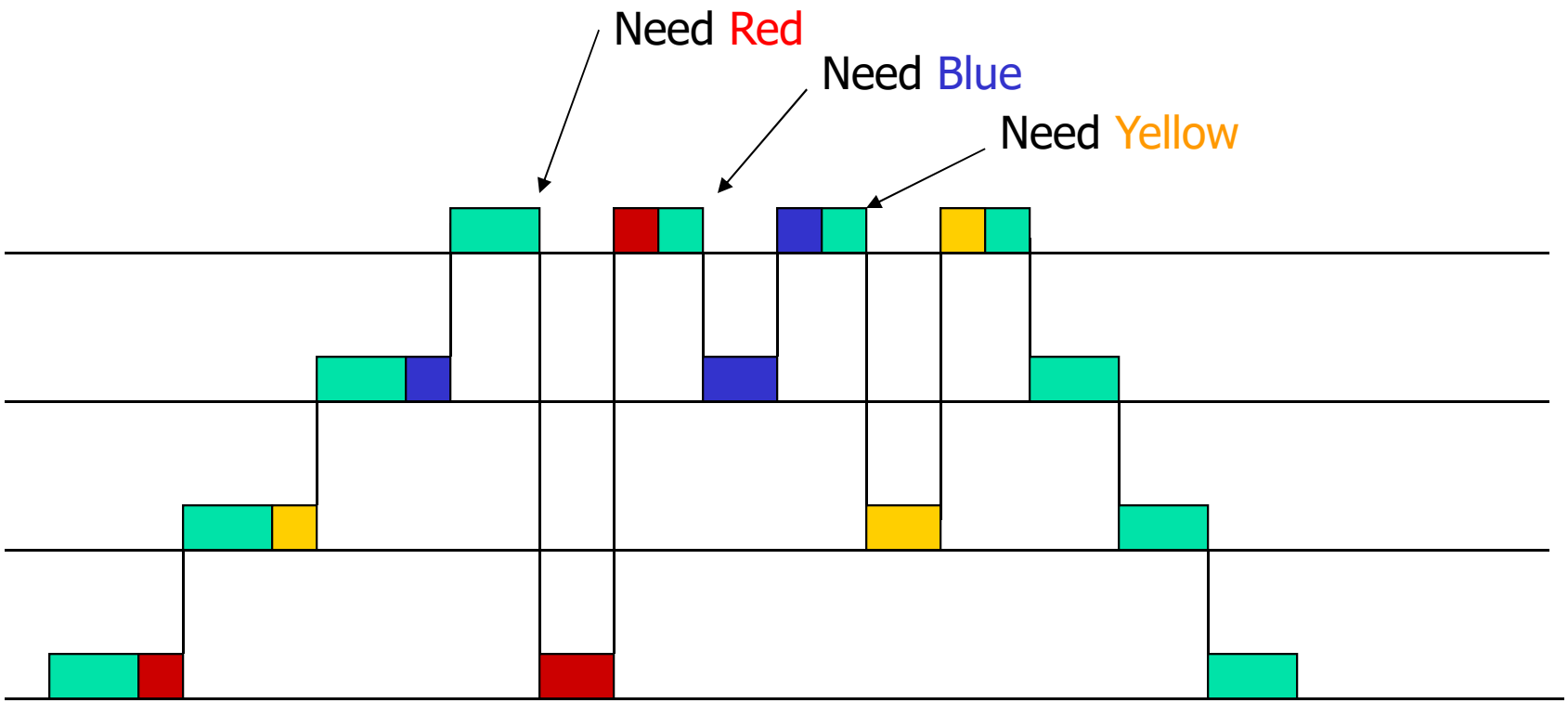
- Consider the instant when a high-priority task that arrives.
  - What is the most it can wait for lower priority ones?

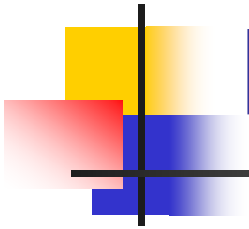


If I am a task, priority inversion occurs when
(a) Lower priority task holds a resource I need (direct blocking)
(b) Lower priority task inherits a higher priority than me because it holds a resource the higher-priority task needs (push-through blocking)

Maximum Blocking Time

Priority Inheritance Protocol

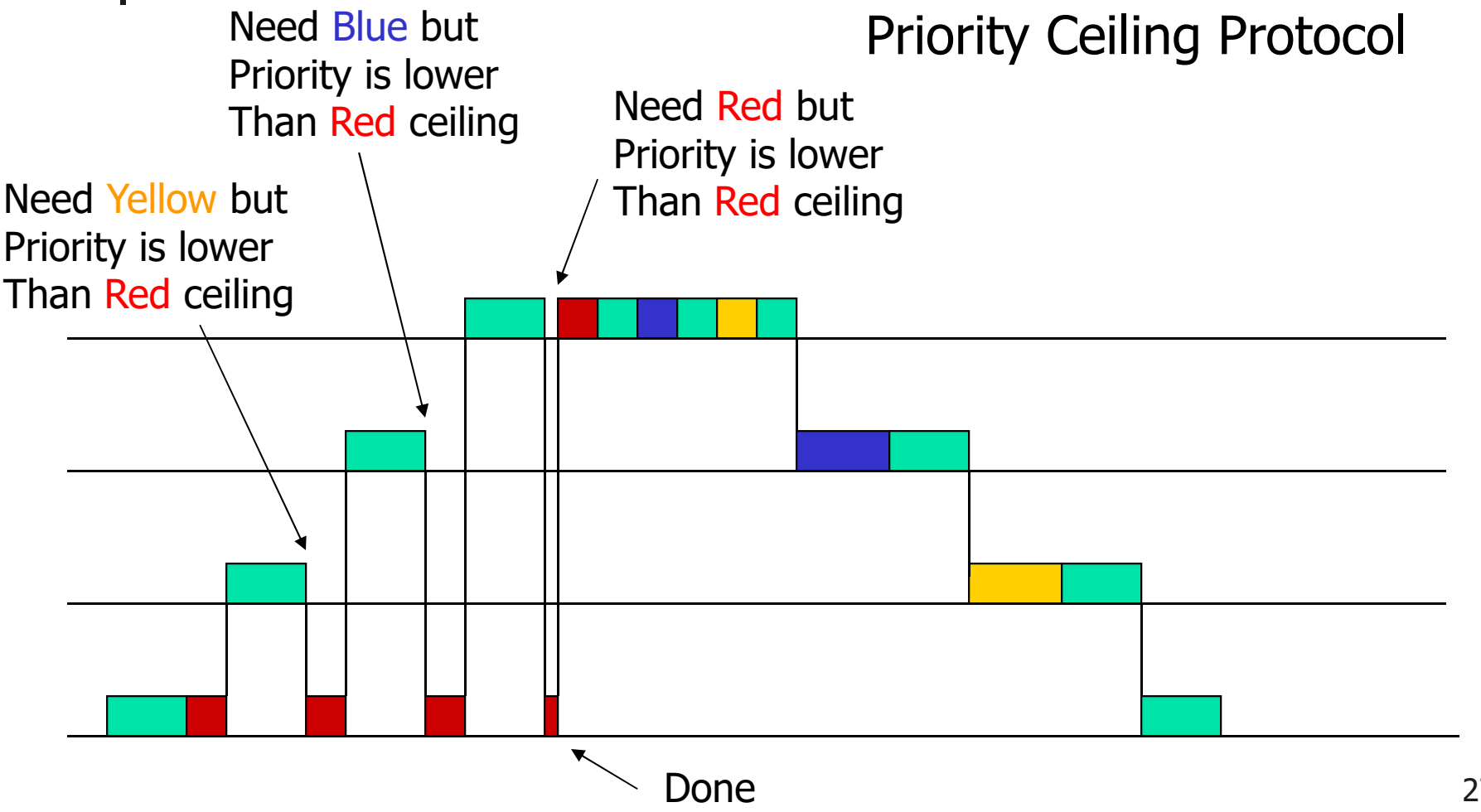




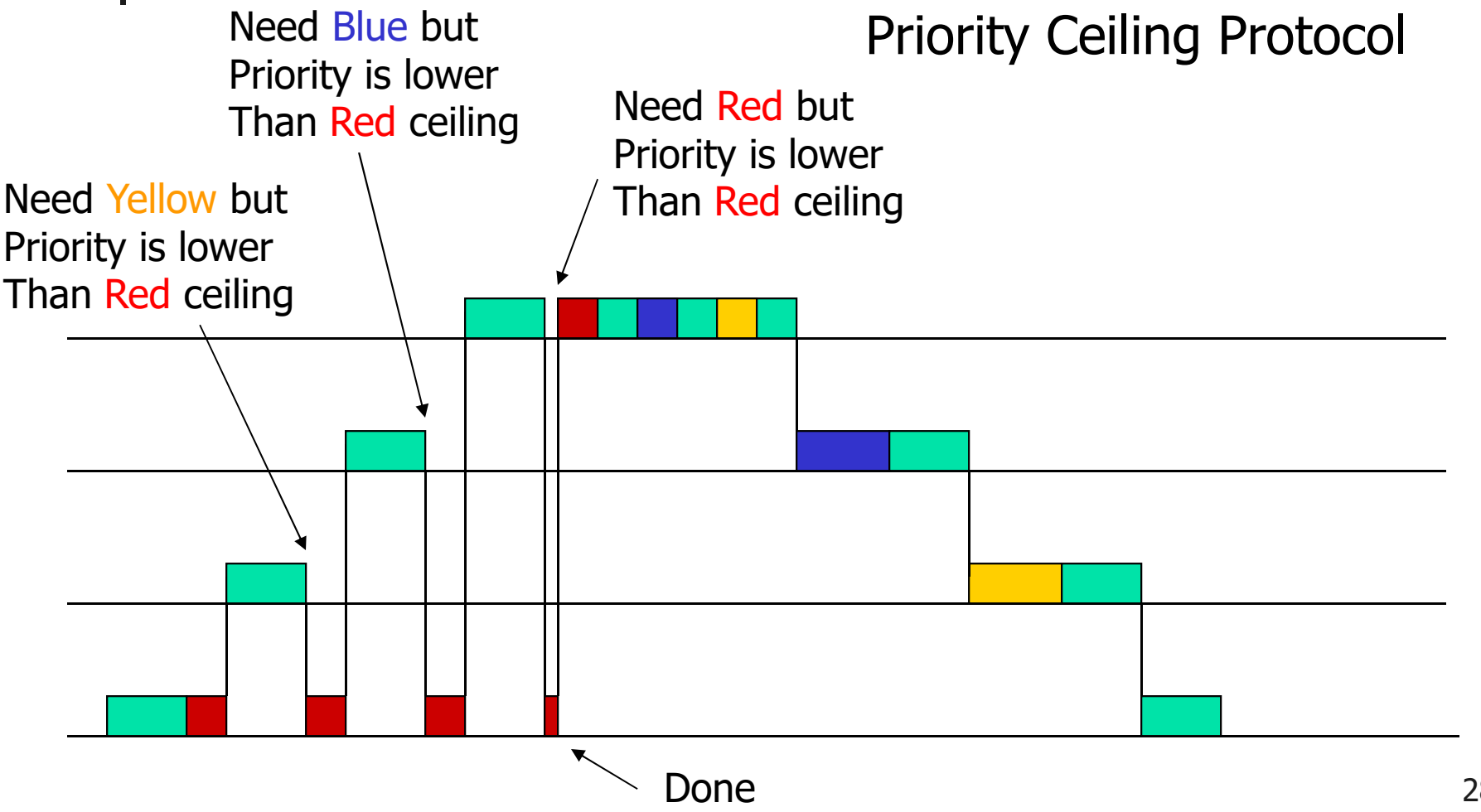
Priority Ceiling Protocol

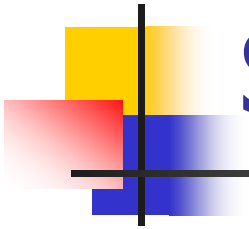
- Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it
- A task that requests a lock R_k is denied if its priority is not higher than the highest priority ceiling of all semaphores currently locked by other tasks (say it belongs to semaphore R_h)
  - The task is said to be blocked by the task holding lock R_h
- A task inherits the priority of the top higher-priority task it is blocking

Maximum Blocking Time



Maximum Blocking Time



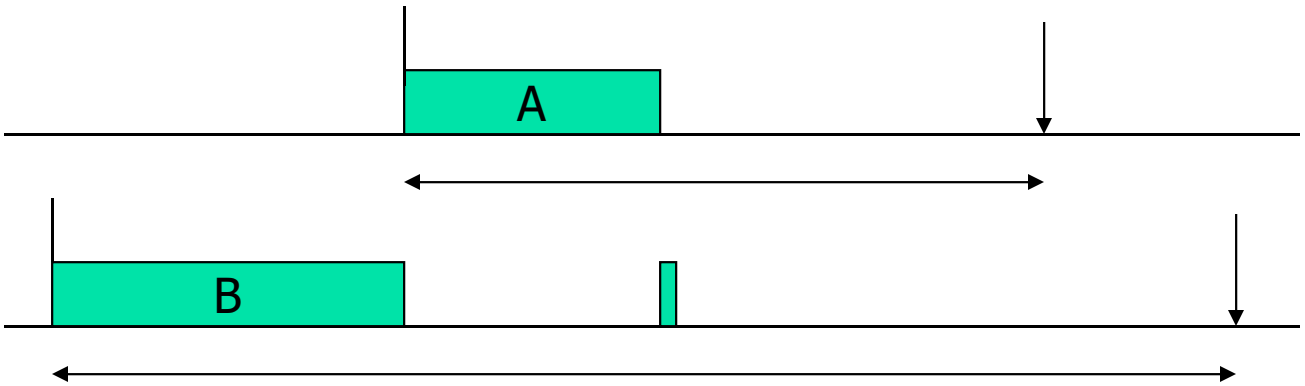


Slack Resource Policy

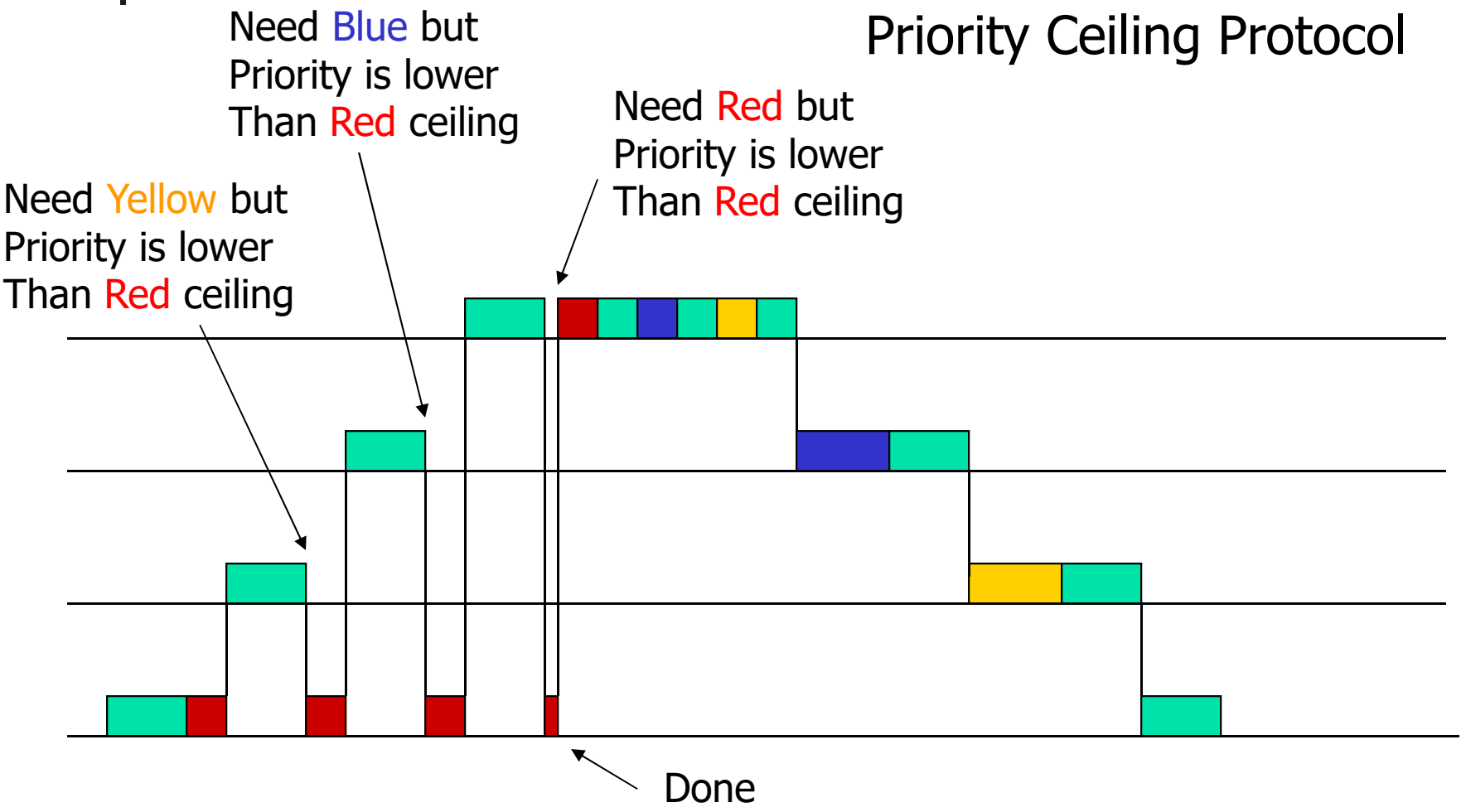
- Priority:
  - Any static or dynamic policy (e.g., EDF, RM, …)
- Preemption Level
  - Any *fixed value* that satisfies: If A arrives after B and Priority (A) > Priority (B) then PreemptionLevel (A) > PreemptionLevel (B)
- Resource Ceiling
  - Highest preemption level of all tasks that might access the resource
- System Ceiling
  - Highest resource ceiling of all currently locked resources
- A task can preempt another if:
  - It has the highest priority
  - Its preemption level is higher than the system ceiling

Example: EDF

- Priority is proportional to the absolute deadline
- Preemption level is proportional to the relative deadline.
- Observe that:
  - If A arrives after B and Priority (A) > Priority (B) then PreemptionLevel (A) > PreemptionLevel (B)



Maximum Blocking Time



Maximum Blocking Time

Slack Resource Policy

Can't preempt
Preemption level is not
higher than ceiling

