ECE 3140 / CS 3420
EMBEDDED SYSTEMS

LECTURE 18

Prof. José F. Martínez

TR 1:25-2:40pm in 150 Olin

Cornell University
Computer Systems Laboratory

NON-PREEMPTIVE PROTOCOL

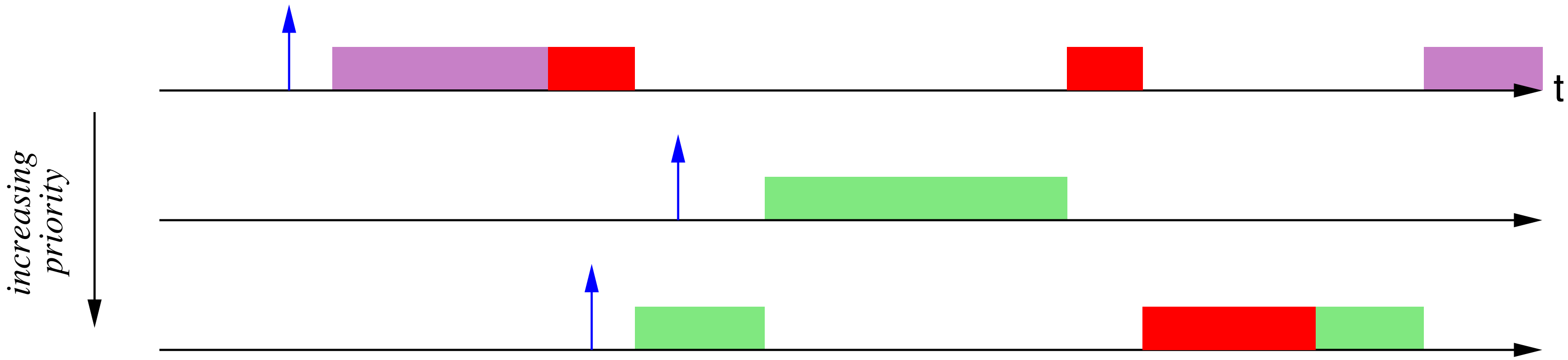
Simple modification:

- Preemption is forbidden in critical sections
- To implement: when a task enters a critical section, increase its priority to the maximum value.
- $p_{CS} = \max_i\{p_1, \ldots, p_n\}$

Drawbacks:

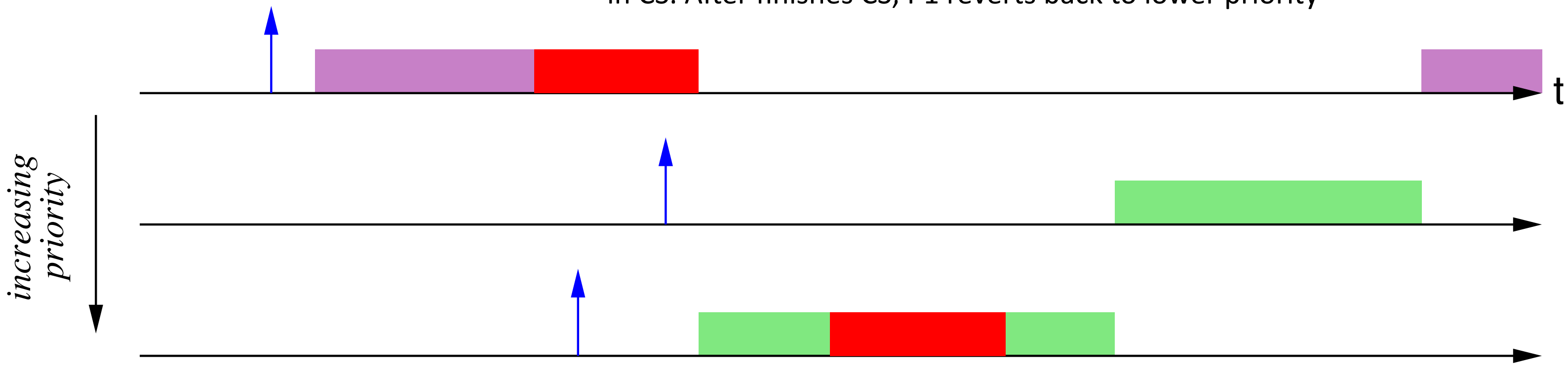
- High priority tasks that do not interfere with the critical section will be blocked

Non-Preemptive Protocol



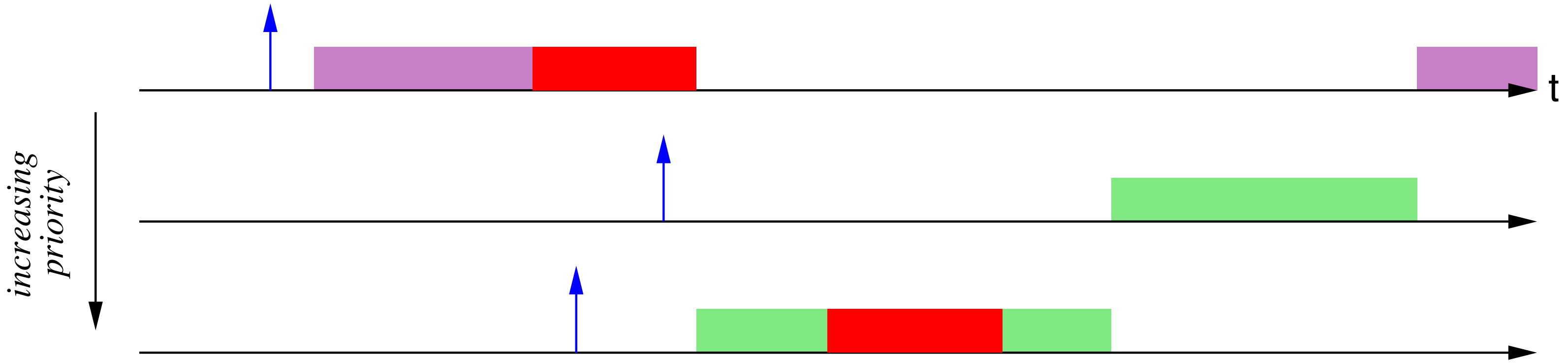
the top has highest priority and goes until completion of tis CS
It then goes to the bottom process and finishes
Then goes to middle process
Despite P2 and P3 arrive at P1’s CS, will not execute bc P1 has the highest priority
in CS. After finishes CS, P1 reverts back to lower priority

With NPP:



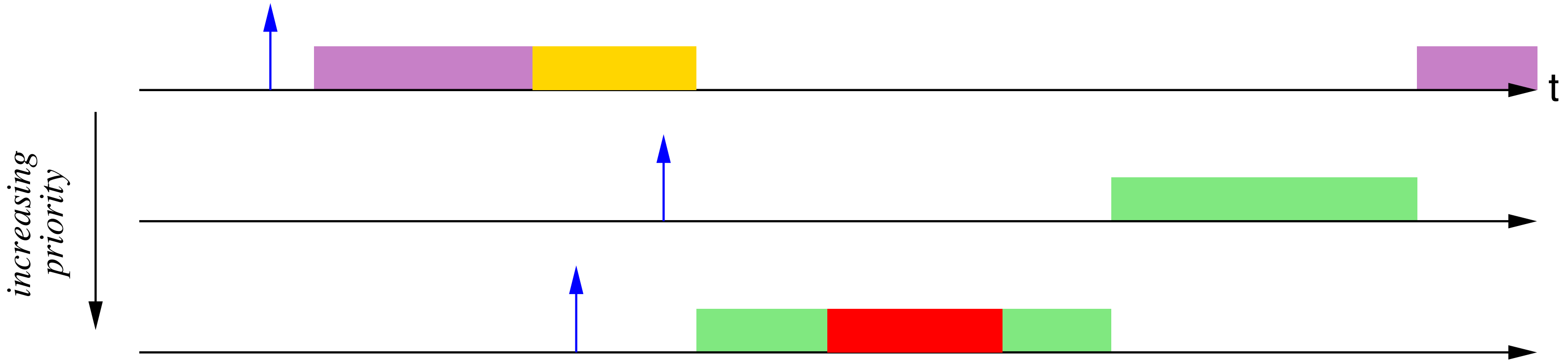
Non-Preemptive Protocol

NPP schedule:



RED; same critical section

. . . even for critical sections that don’t matter



Even with different CS, they still respect priority scheduling, which is inefficient

HIGHEST LOCKER PRIORITY

Bump priority above everyone else that uses the SAME critical

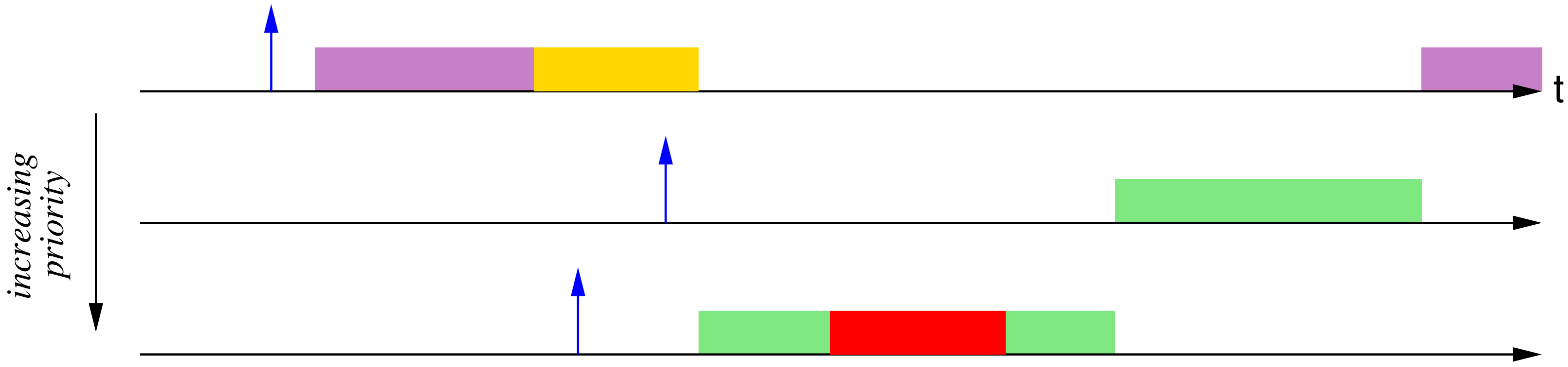
- A task in the critical section gets the highest priority among the tasks that use the critical section.
- To implement: when a task enters a critical section, increase its priority to the maximum value of the tasks that may access the critical section.
- $p_{CS} = \max_i \{p_i \mid \tau_i \text{ uses CS}\}$

Drawbacks:

- A task could be blocked because it *might* enter the critical section, not because it is in the critical section.

Highest Locker Priority

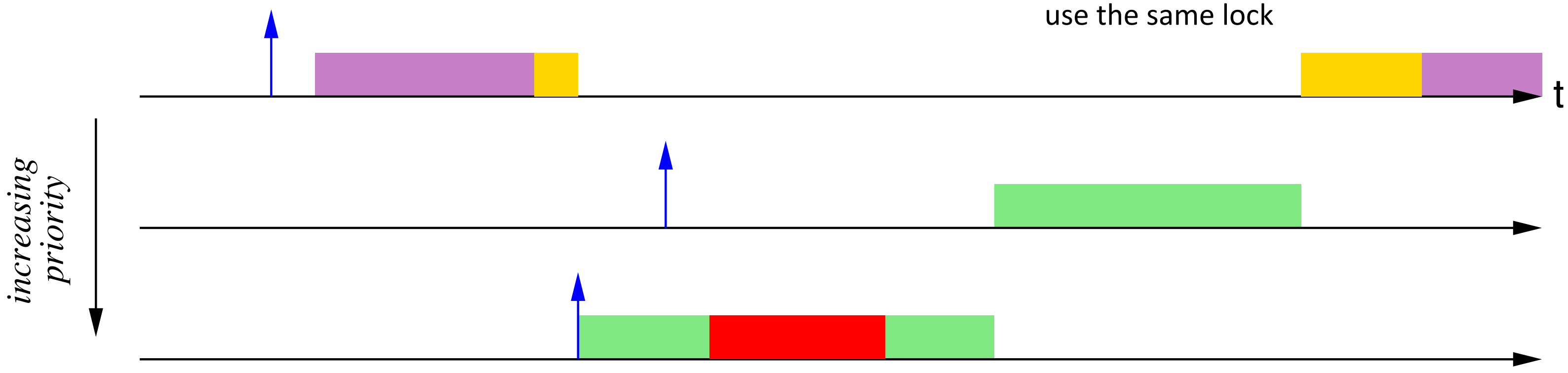
NPP:



The yellow CS gets broken up because there are different CS for the processes
Yellow is the only type of CS so it can be broken up; so the priority doesn't change
Information on different CS must be known offline
Look at code to see if anything would every use the same lock

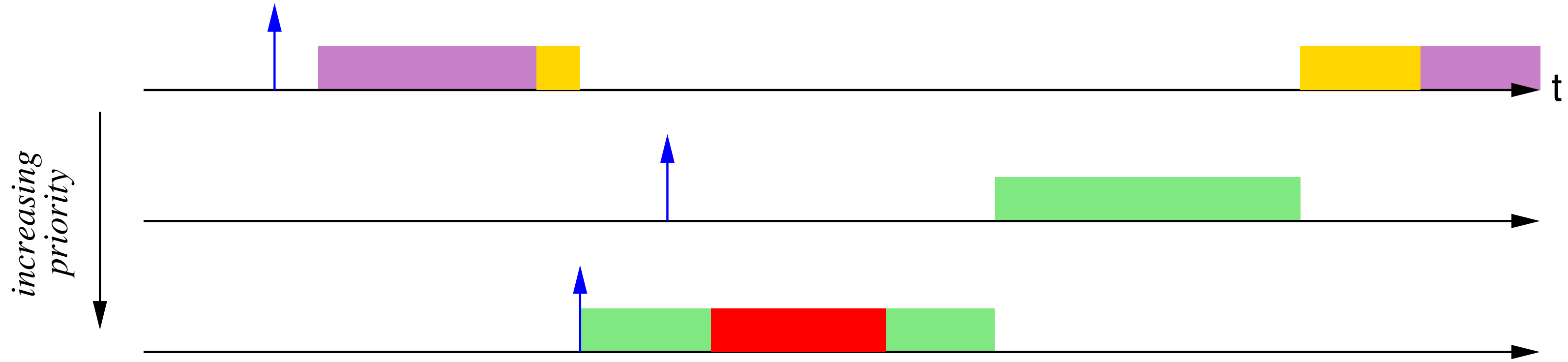
In NPP the CS cannot be broken up even though the CS are different

HLP:



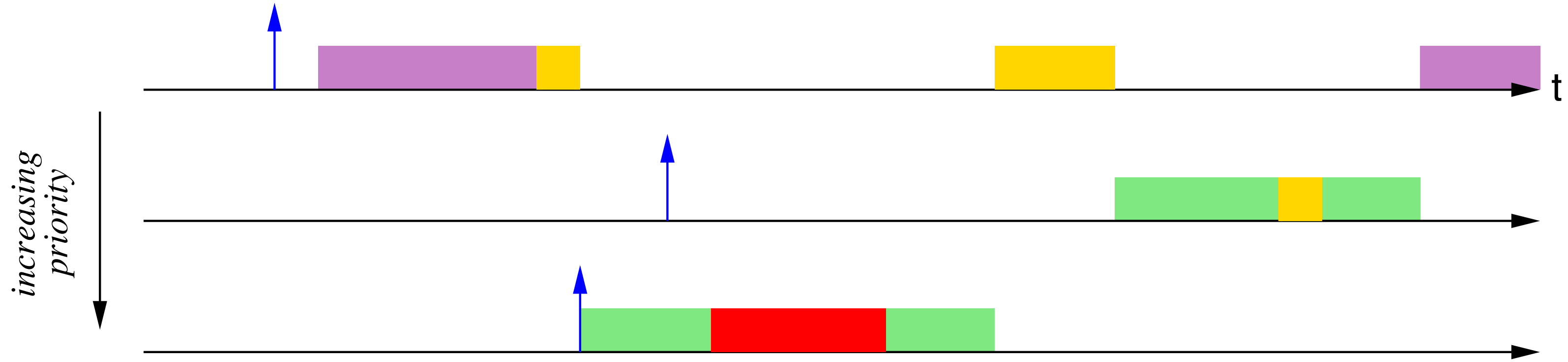
Highest Locker Priority

HLP:



PROBLEM: the second process cannot start the green
section until the first process finishes (overkil)
Only needs that both processes cannot be in the
same CS at the same time

If the middle task might use the yellow lock:



Priority Inheritance Protocol

Further efficiency

- A task in a critical section increases its priority only if it blocks other tasks.

Alter order only when you need to block

- A task in a critical section inherits the highest priority among those tasks that it blocks.

block when try to access a CS when another thread owns that CS

- $p_{CS} = \max_i\{p_i \mid \tau_i \text{ blocked on CS}\}$

The guy who has lower priority switched
Increase prioirty when activley blocking somebody
Inhearity the proitiy of the one's that are actively blocked

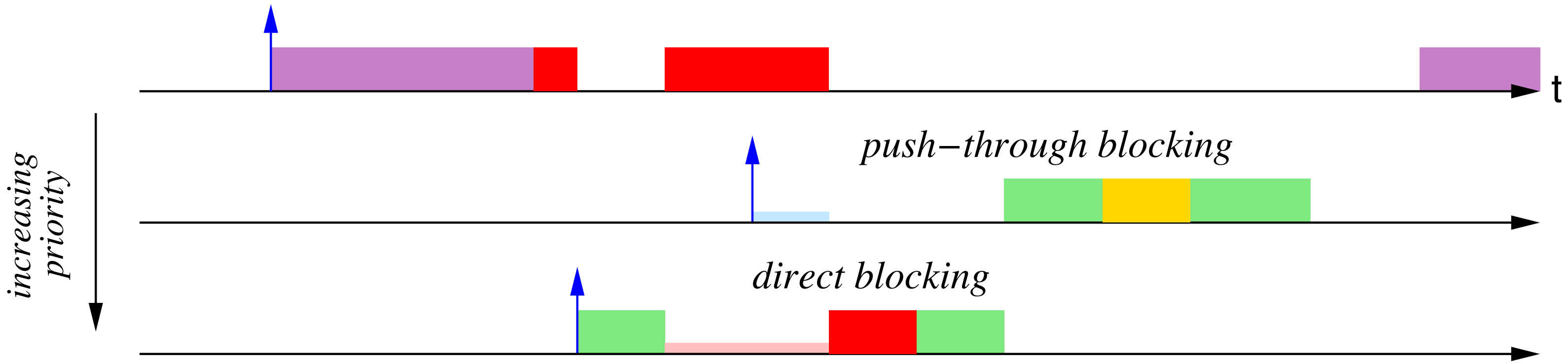
Two types of blocking:

- Direct: task blocked on a lock
- Push-through: task blocked because a lower priority task inherited a higher priority

PRIORITY INHERITANCE PROTOCOL

Example with PIP:

this is push-through bc P1 has inherited higher priority



Direct block because P1 holds the lock for the red CS

Process 1 switches to highest priority

Process 3: Inheritys lower prioiry of P1 priority

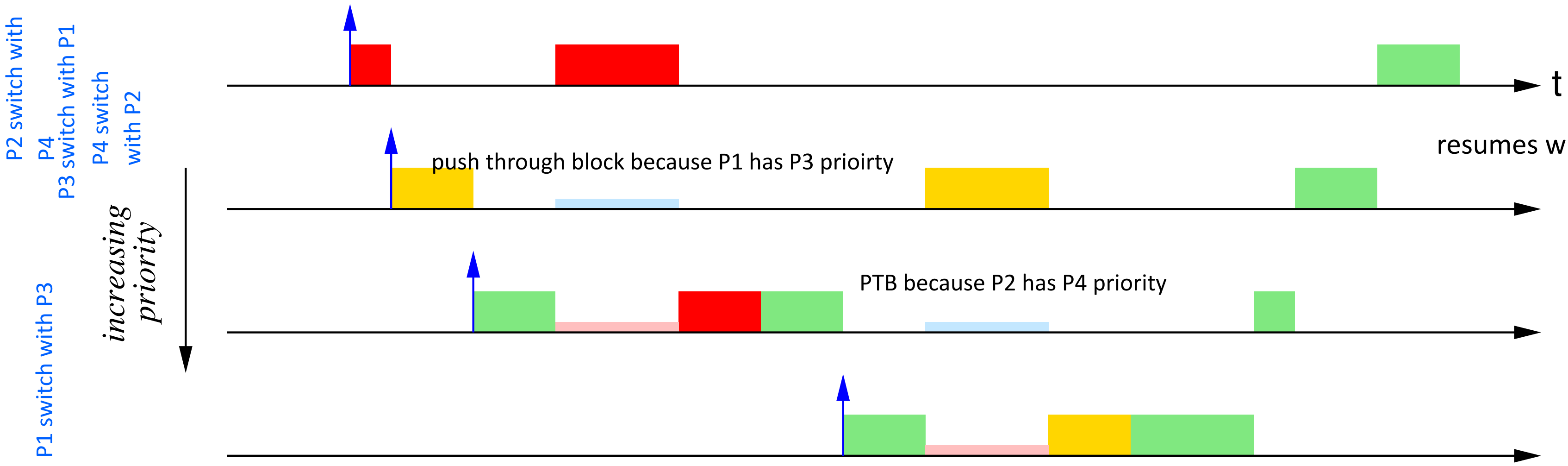
When P1 is done with CS it loses its higher priority

Priority Inheritance Protocol

Even though P2 is in CS, it switches to P3, because P3>P2 and red is now most important even though P1 > P3

push-through block
prevent priority inversion

Example with PIP:



P2 switch with P4

P3 switch with P1

P4 switch with P2

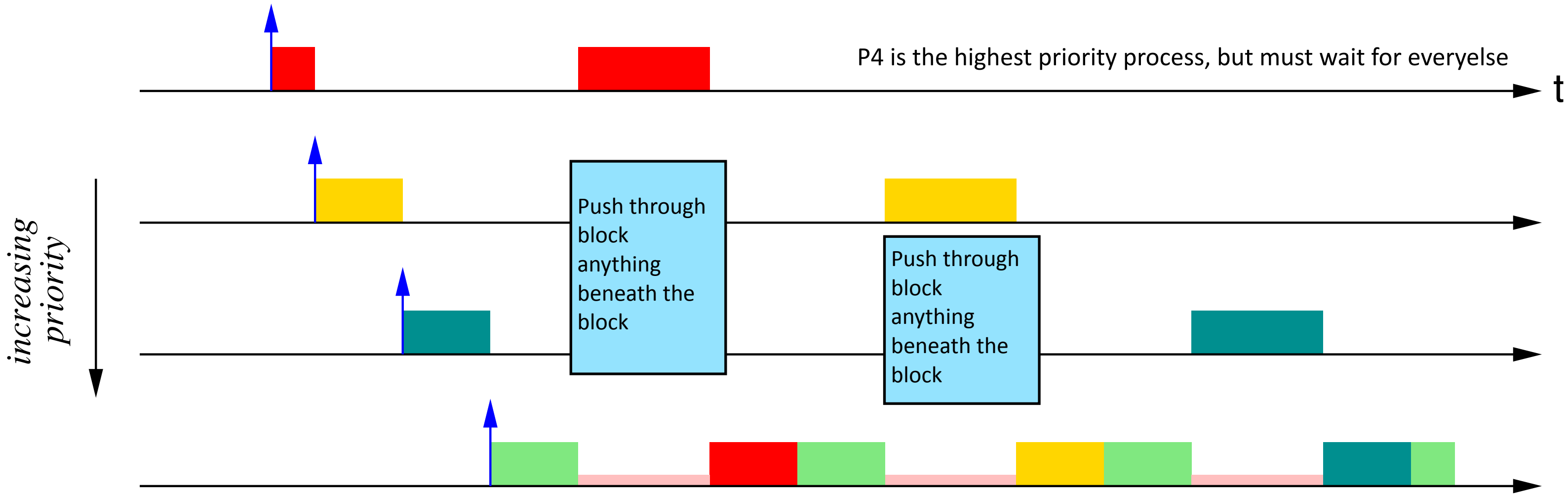
P1 switch with P3

push through block because P1 has P3 prioirty

PTB because P2 has P4 priority

resumes with original priorities

CHAINED BLOCKING


P4 is the highest priority process, but must wait for everyelse
t
increasing priority
Push through block anything beneath the block
Push through block anything beneath the block

the other preempts the other

PRIORITY CEILING PROTOCOL

Attempts to reduce chained blocking

- A modification of the PIP protocol
- Each lock is assigned a *ceiling*
  - For a lock l_k,

LAST ONE! The most fancy

$$C(l_k) = \max_i \{p_i \mid \tau_i \text{ uses } l_k\}$$

  - A task τ_i can enter the critical section only if

Text

$$p_i > \max_k \{C(l_k) \mid l_k \text{ is locked by tasks} \neq \tau_i\}$$

hence p is < NOT ≤

  - As in PIP, tasks inherit the (highest) priority of the task(s) they block

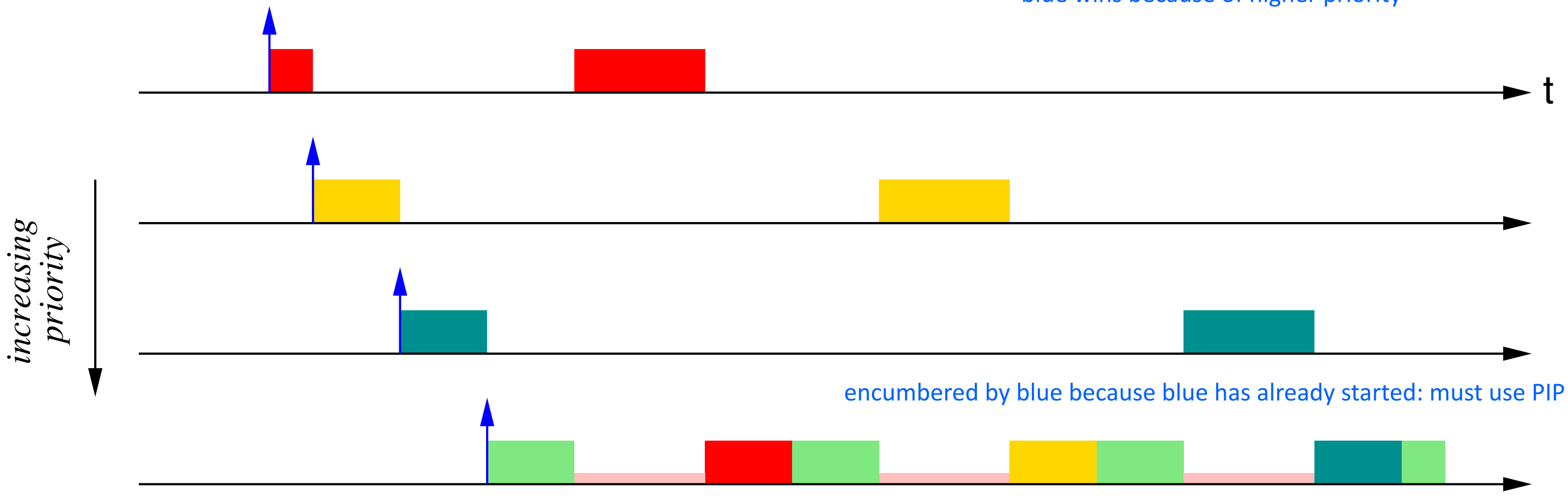
**IDEA: in order to avoid chain block, push the CS to either side of me

PRIORITY CEILING PROTOCOL

All CS have a ceiling of priority of 3
i CAN ONLY GO INTO THE priority if my priority is higher then the active locks

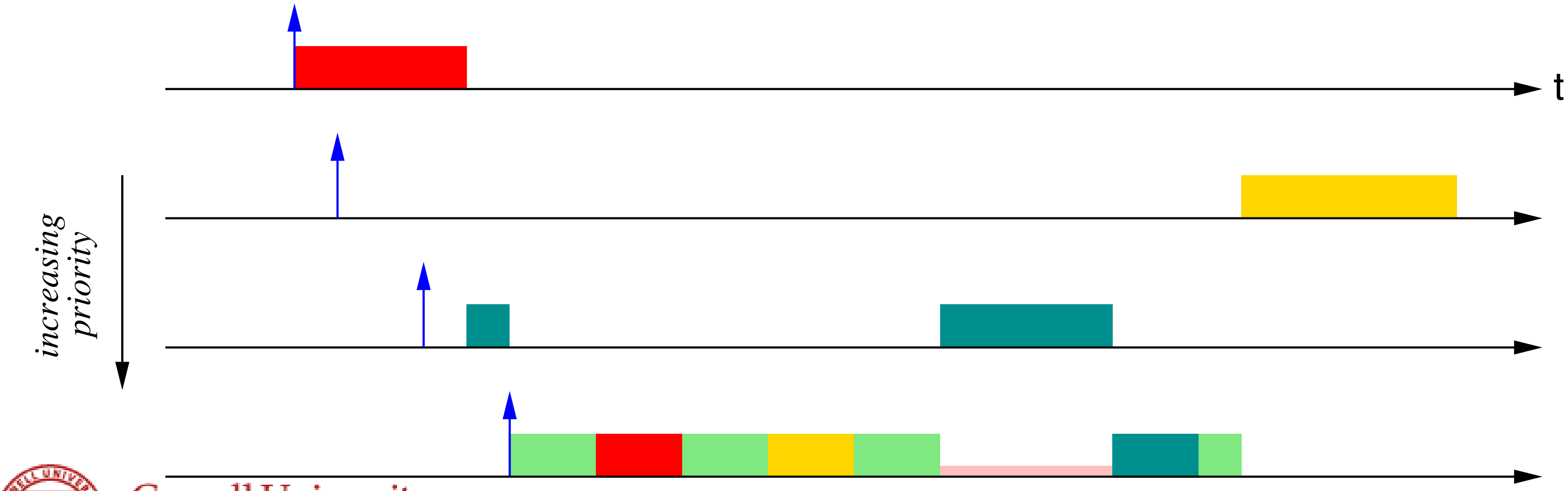
PIP:

Once red CS finishes, then everyone wants to enter CS,
blue wins because of higher priority



encumbered by blue because blue has already started: must use PIP

PCP:


t
increasing priority