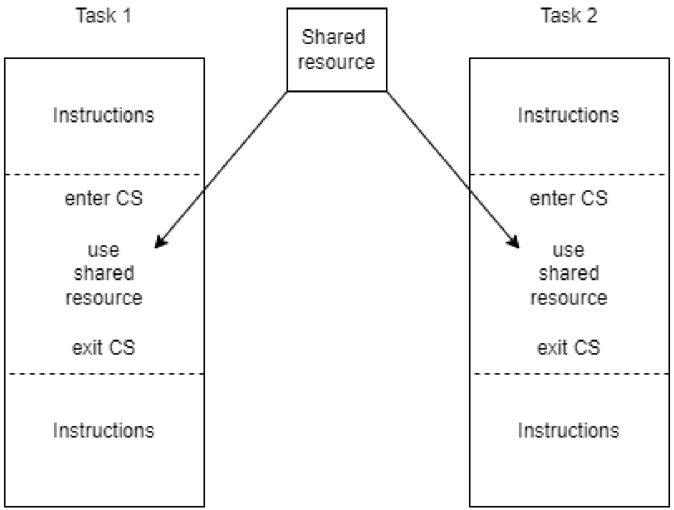
TSM_AdvEmbSof

Priority Inversion

Serge Ayer | 22.11.2024 | Cours MSE

What is the concern?

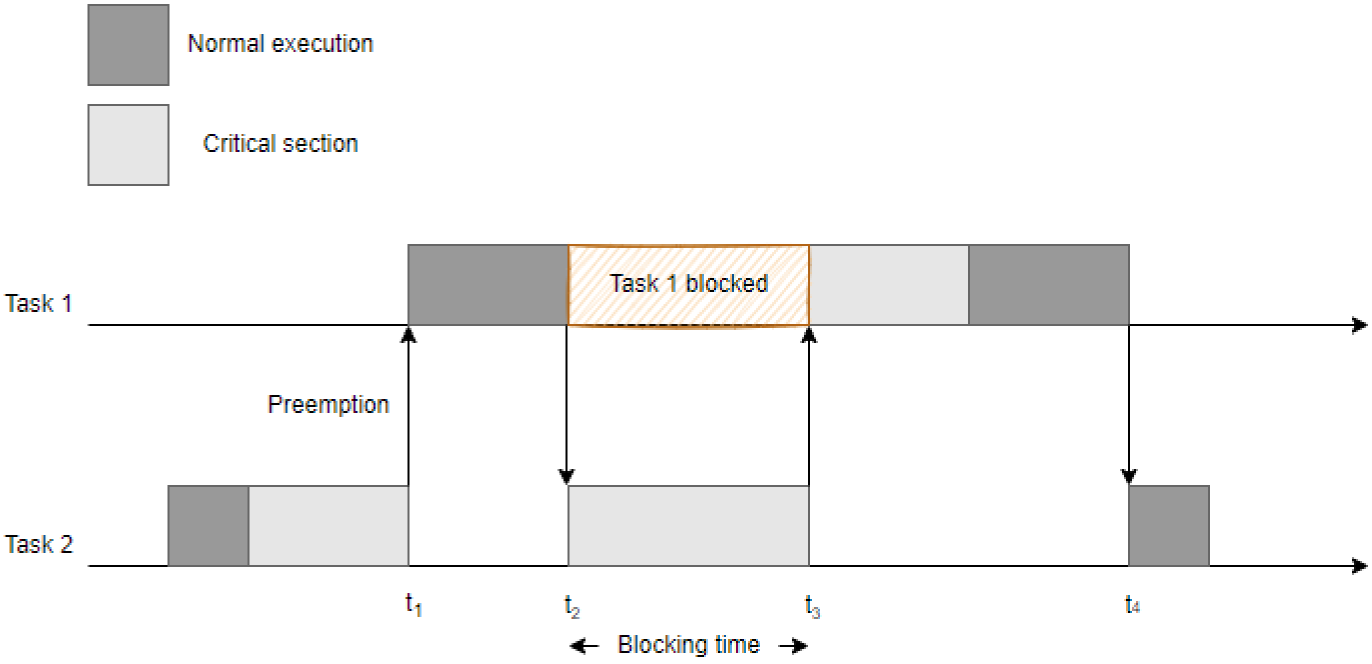
- Multi-tasking program often require to share resources among tasks
- Access to shared resources needs to be protected against concurrent access
  - Mutual exclusion among competing tasks
  - A piece of code executed under mutual exclusion is a critical section



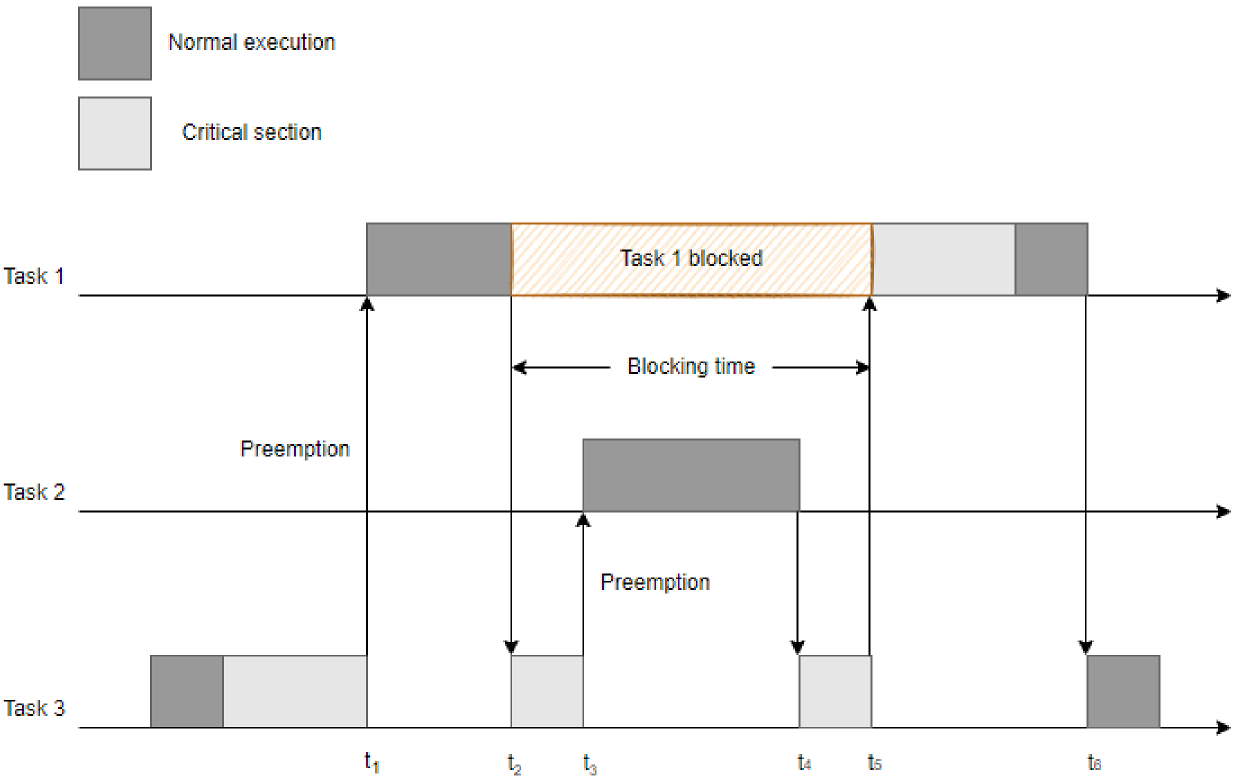
Priority Inversion

- Until now, we have considered that tasks are independent, sharing no resources, and not interacting. What if we remove this assumption?
  - Scheduling of tasks is affected!
- One main problem arises: priority inversion
  - High priority tasks may be blocked by lower priority tasks
- The main sources of priority inversion are
  - Non preemptable sections
  - Sharing resources
  - Synchronization and mutual exclusion
- In all cases, the response time (latencies) are modified

Priority Inversion (Simplest Form)



Priority Inversion (Unbounded Form)



Solutions to Priority Inversion

- Tasks are ’forced’ to follow certain rules when locking and unlocking a mutex
  - This is about requesting and releasing resources.
- The rules are often called Resource Access Protocols
  - There are several existing such protocols
- What about Mbed OS scheduling algorithm?
  - Mbed OS/RTX implements the priority inheritance mechanism

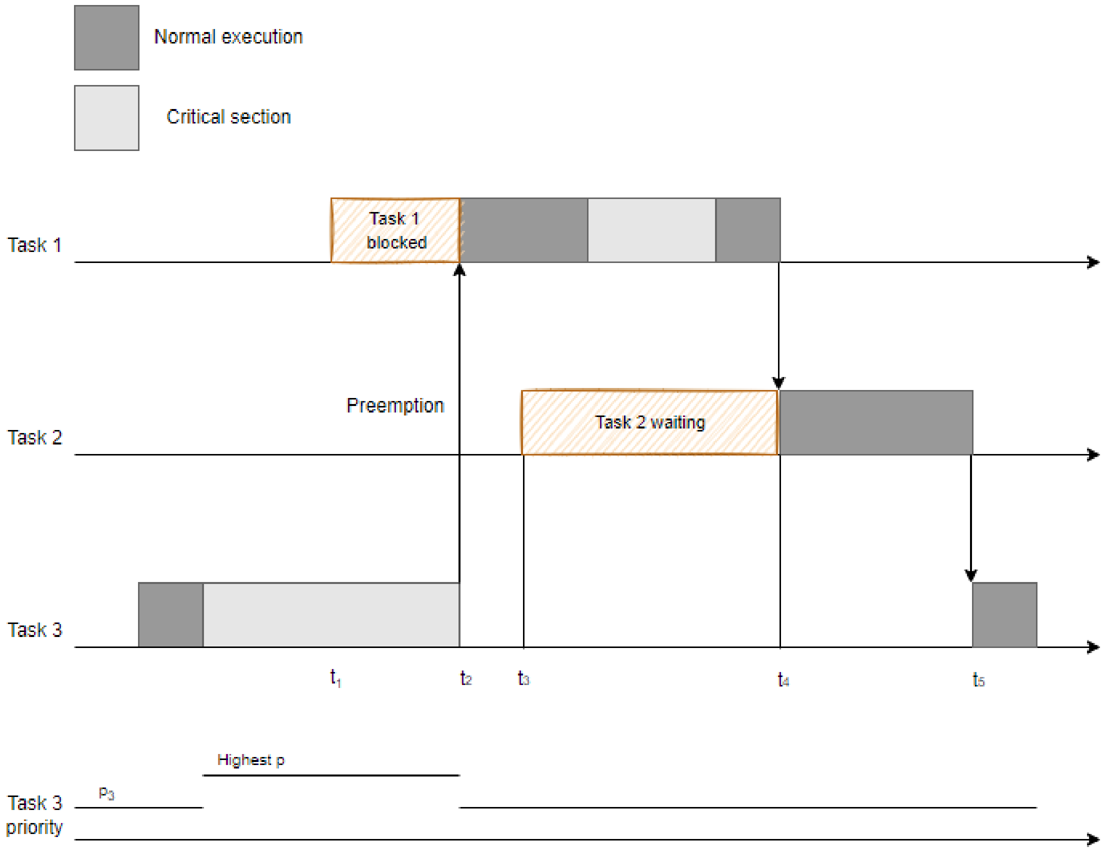
Resource Access Protocols

- Need to consider the following points
  - Is the priority inversion bounded?
  - Does the protocol avoid deadlock?
  - Does the protocol avoid unnecessary blocking?
  - Is it easy to calculate the blocking time upper bound?
  - What is the maximum number of blocking?
  - Is it easy to implement?

Non-Preemptive Protocol (NPP)

- Principle: disallow pre-emption during the execution of any critical section
  - A task is assigned the highest priority if it succeeds in locking a critical section
- The task is assigned its own priority when it releases the critical section

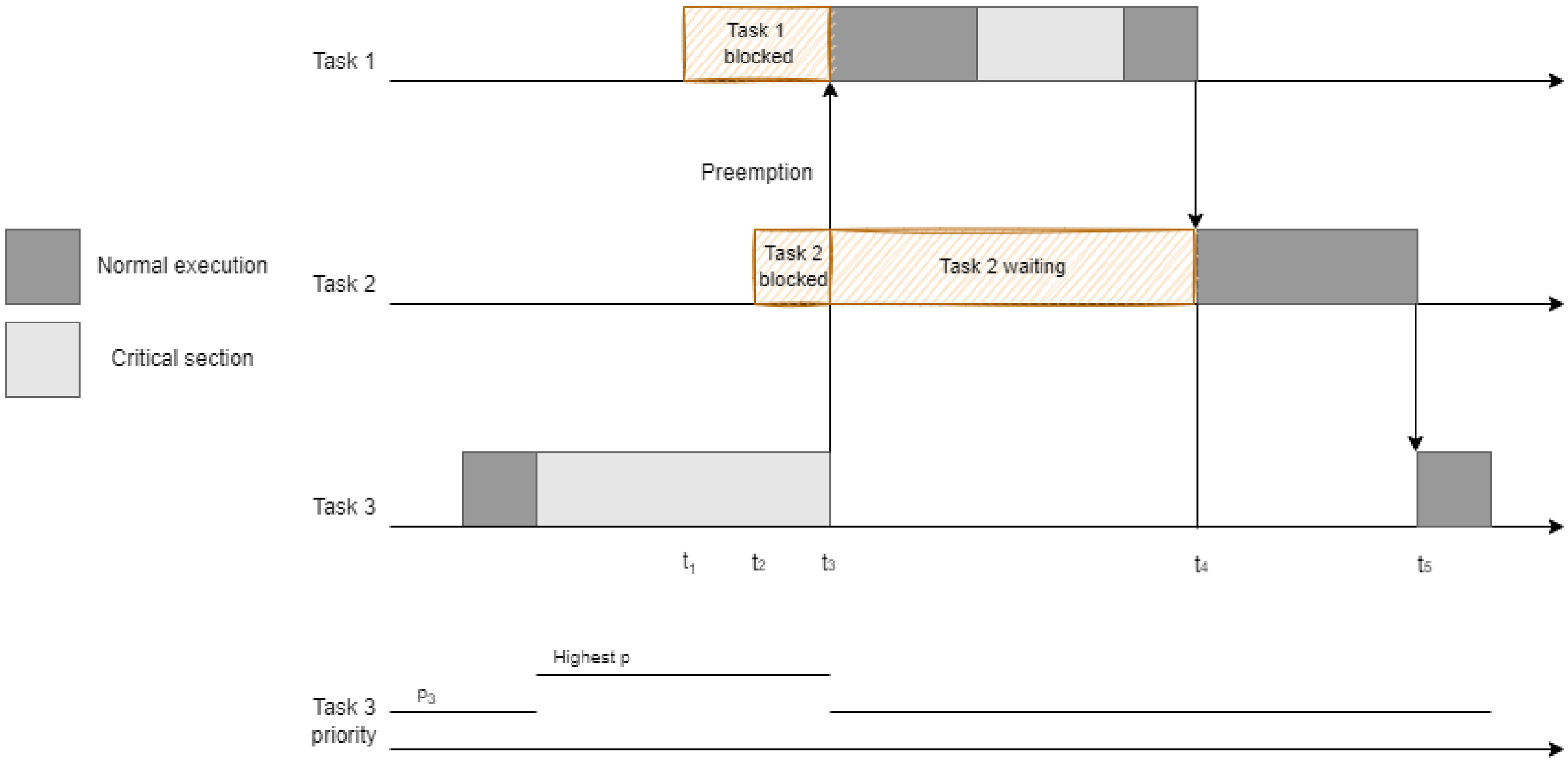
Non-Preemptive Protocol (NPP)



Non-Preemptive Protocol (NPP)

- Advantages:
  - This bounds Priority Inversion and for a given task the bound is the maximal length of any single critical section belonging to lower priority tasks
  - It is deadlock free
  - It limits the number of blocking of any task to one
  - It is easy to implement and transparent
- But:
  - It allows low priority tasks to block high priority tasks including those that do not require access to shared resources.

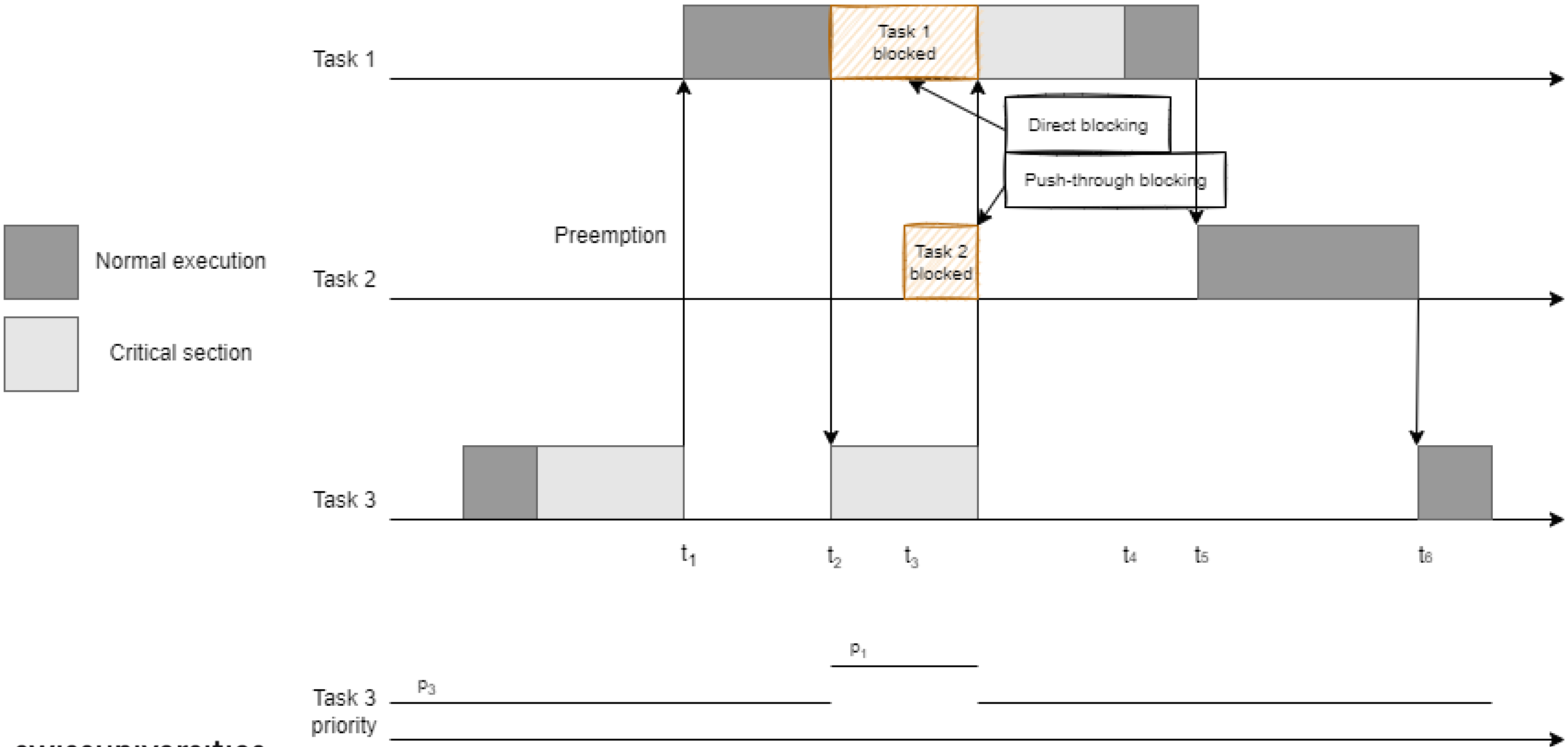
Non-Preemptive Protocol (NPP)



Priority Inheritance Protocol

- The need is to again to avoid unbounded priority inversion
- Principle:
  - The idea is to elevate the priority of a low priority task to the highest priority of tasks blocked by it.
  - And resume its original priority when it exits the critical section.
  - This prevents medium-priority tasks from preempting lower priority tasks and thus prolonging the blocking duration experienced by the higher-priority tasks.

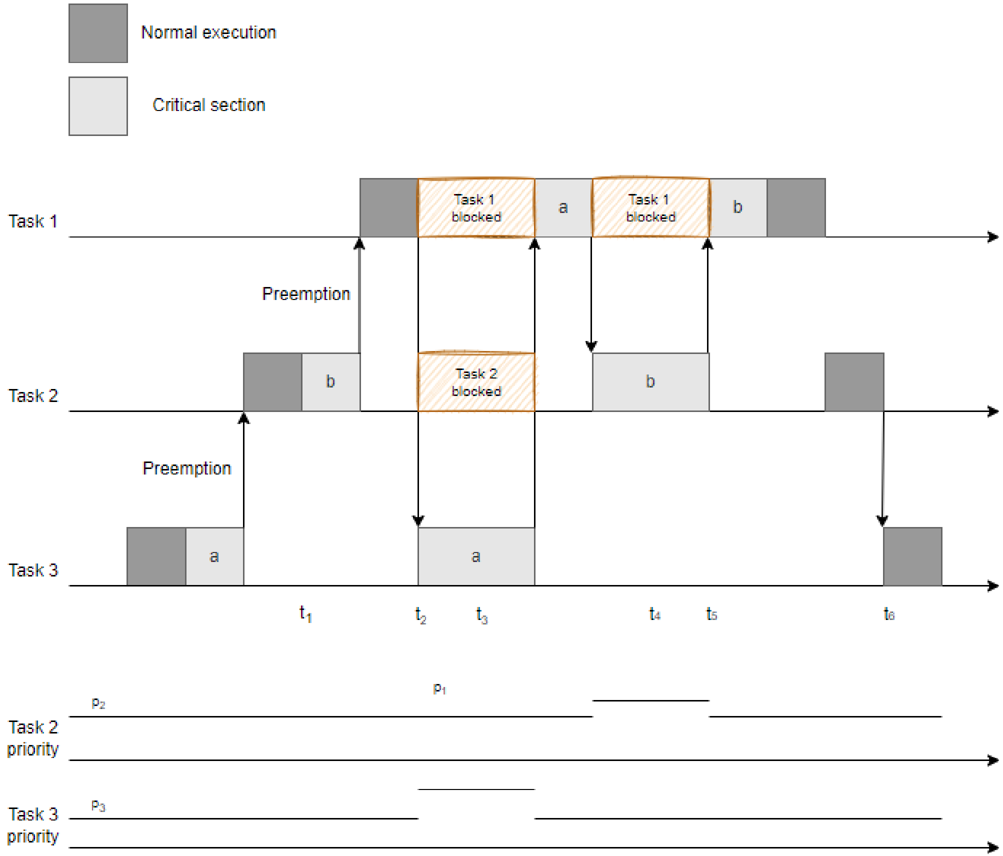
Priority Inheritance Protocol (PIP)



Priority Inheritance Protocol

- Advantages:
  - Blocking time is bounded.
  - Blocking time can be computed.
- But:
  - It is not deadlock free.
  - Chained blocking is still possible (push-through blocking)
  - PIP must implement transitive inheritance (with nested CSs)
- Applied in many RTOS including Mbed OS/RTX.

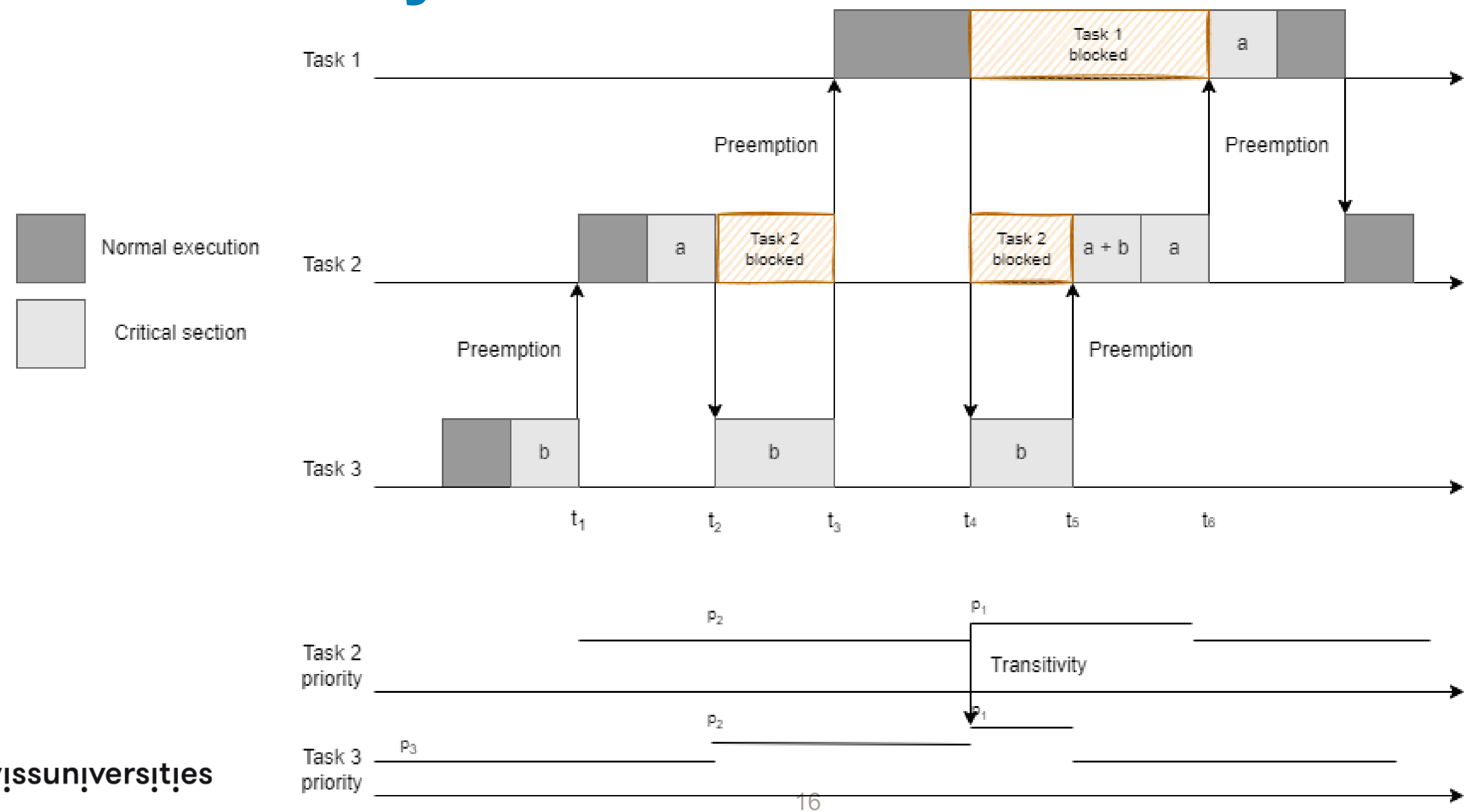
Chained blocking with Priority Inheritance



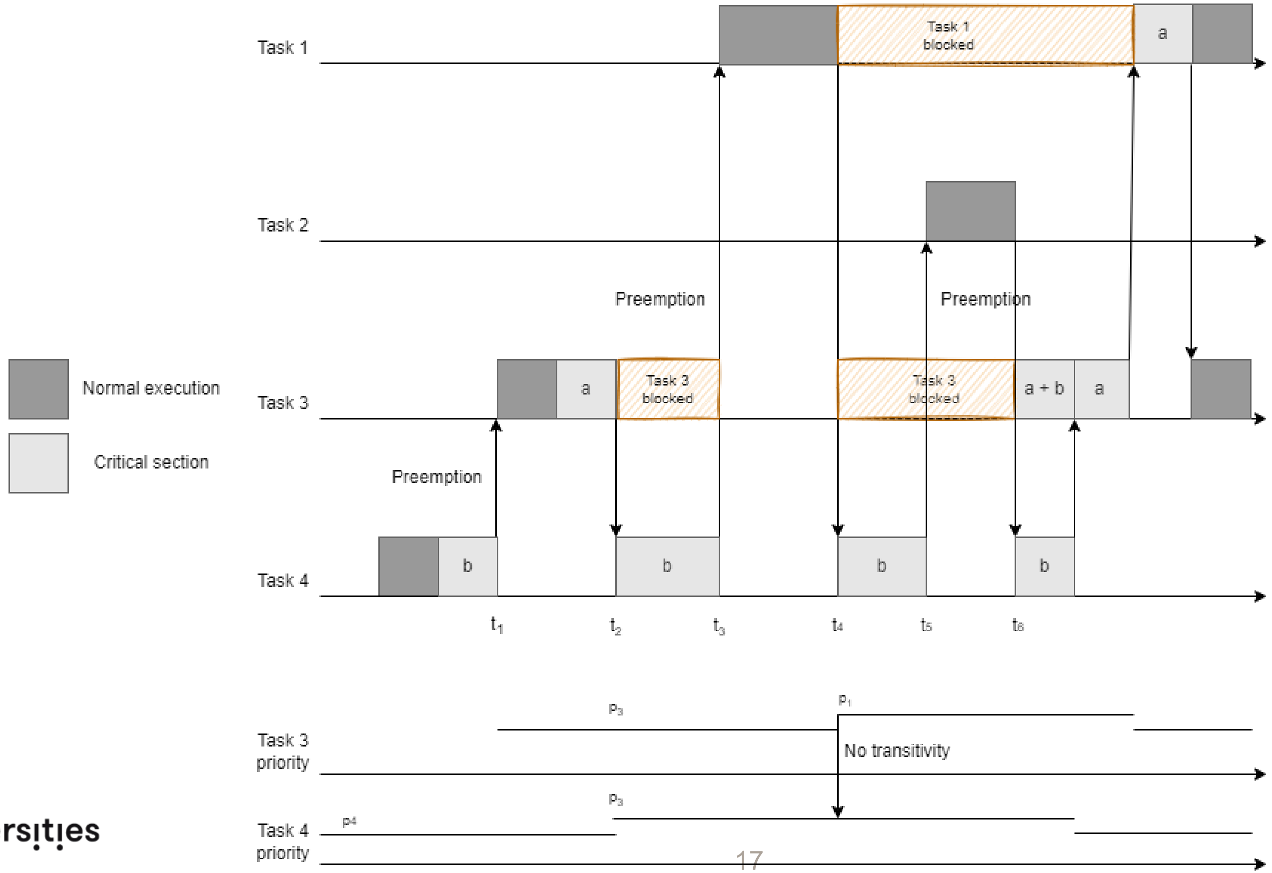
swissuniversities

Hes·so Master

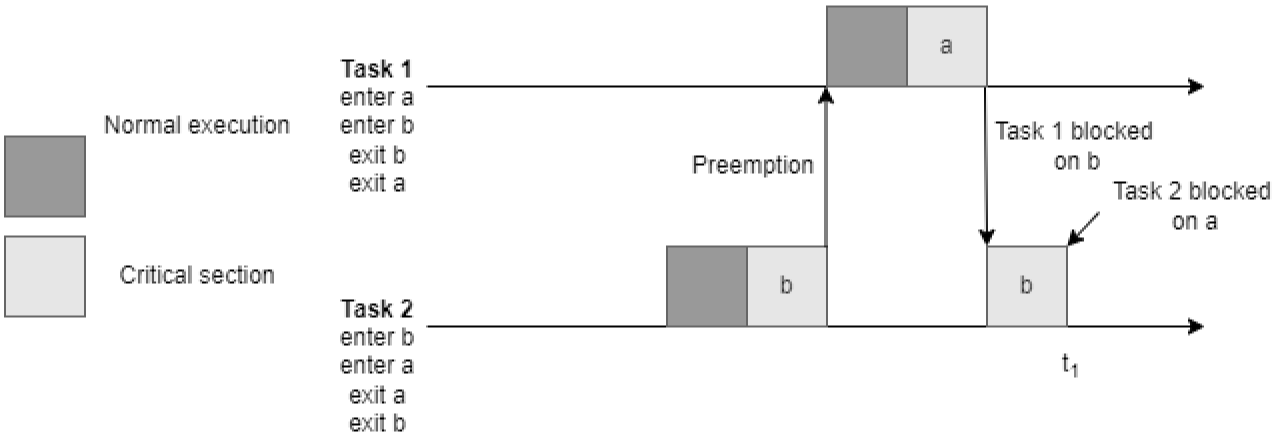
Transitivity for PIP

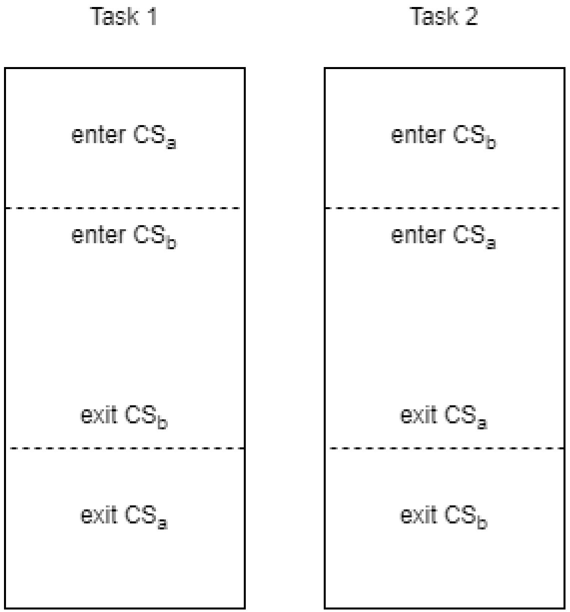


Transitivity is needed



Deadlock possible with PIP





PIP does not prevent deadlock

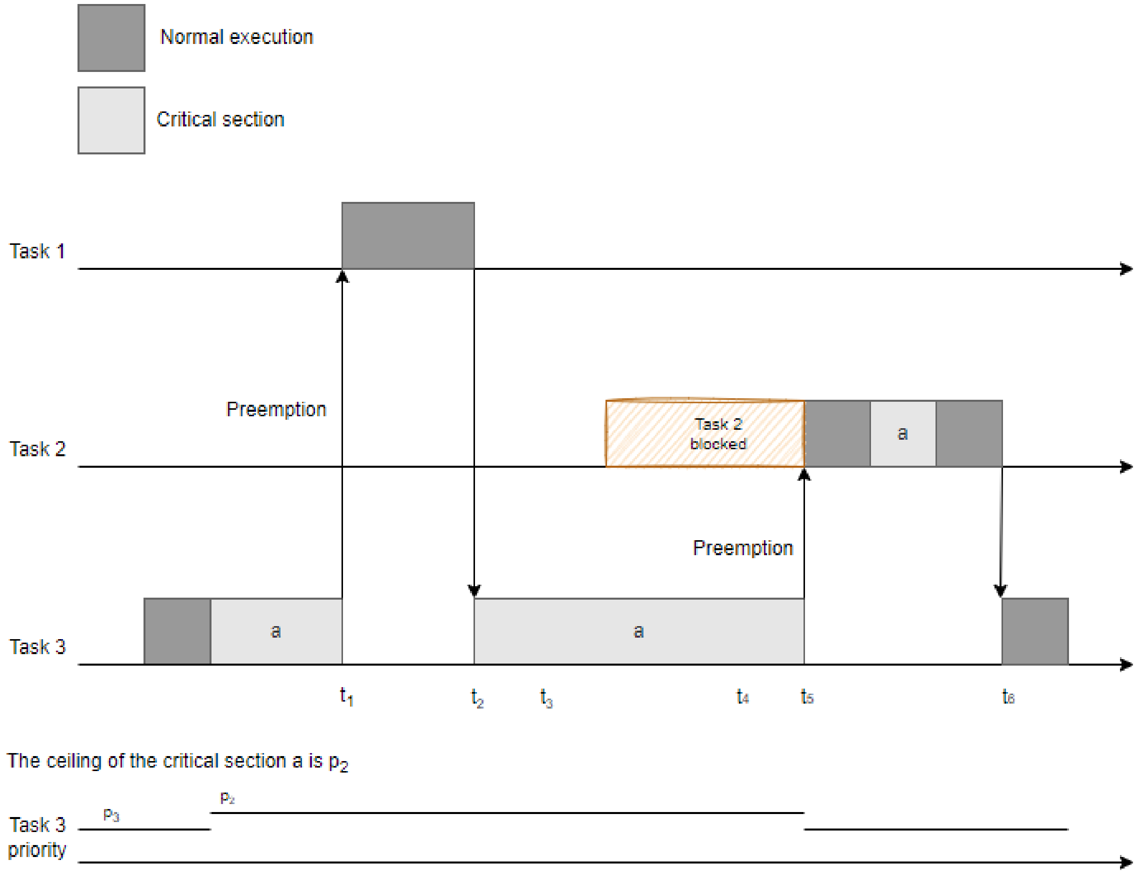
Highest Locker's Priority Protocol

- Idea: define the ceiling C(S) of a critical section S to be the highest priority of all tasks that use S during execution. Note that C(S) can be calculated statically (off-line).
- Whenever a task succeeds in holding a critical section S, its priority is changed dynamically to the maximum of its current priority and C(S)
- When it finishes with S, it sets its priority back to what it was before.

	priority	use
Task 1	H	S3
Task 2	M	S1, S
Task 3	L	S1, S2
Task 4	Lower	S2, S

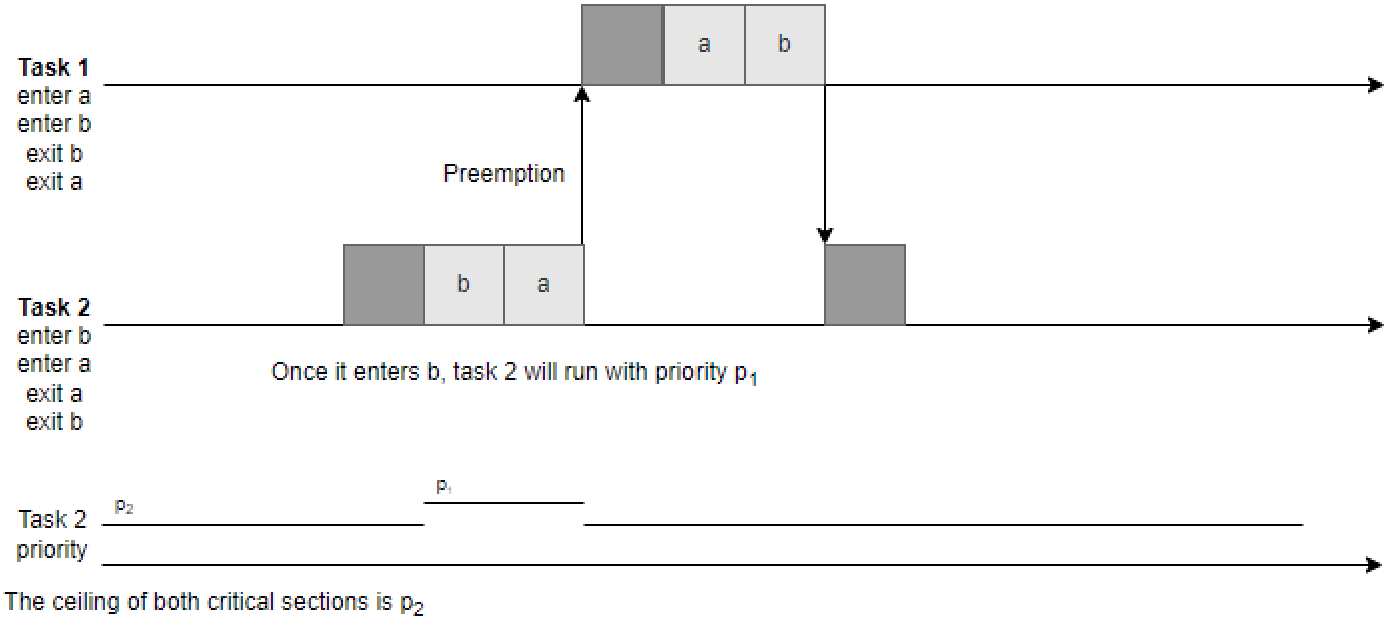
C(S1)=M
C(S2)=L
C(S3)=H
C(S)=M

Highest Locker's Priority Protocol

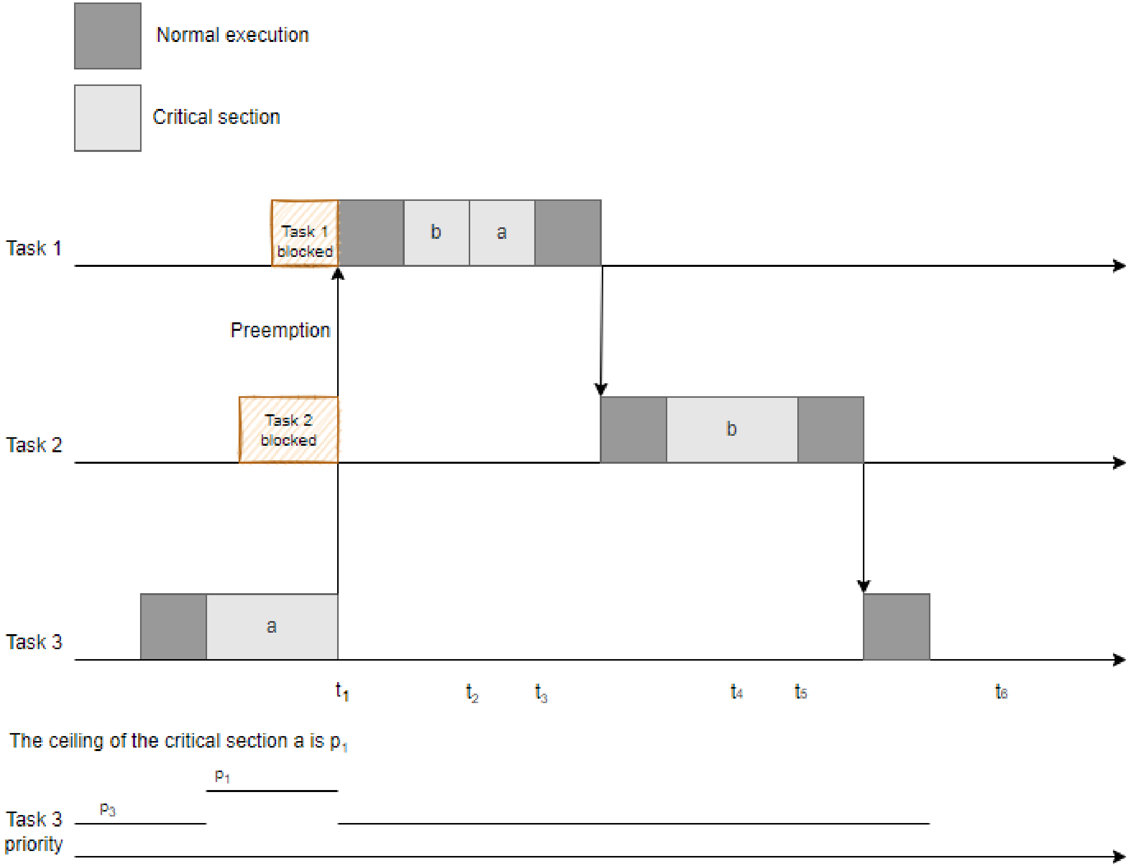


No deadlock with HLP

- Once task 2 gets b, it runs with priority p_1, task 1 will be blocked and cannot get a before task 2



No chained blocking with HLP



Other resource access protocols

- Priority Ceiling Protocol
  - Extend the Priority Inheritance Protocol
  - Rule for granting a lock request on a free mutex.
  - Avoid multiple blocking by not allowing a task to enter a critical section if there are locked mutexes that could block it.
  - This means that once a task enters its first critical section, it can never be blocked by lower-priority tasks until its completion.
- Stack Resource Policy
  - Allows dynamic scheduling

Summary

Algorithm	Chained blocking	Unnecessary Blocking	Blocking instant	Deadlock prevention	Transparency	Implementation
NPP	no	yes	on arrival	yes	yes	easy
PIP	yes	limited	on access	no	yes	medium
HLP	no	yes	on arrival	yes	no	medium