System Architecture

15 Priority Inversion

Gerd Liefländer
December 22 2008
Winter Term 2008/09

Slides made by Kevin Elphinstone
EMail: kevine@cse.unsw.edu.au

Agenda

- Introduction
- Basic Example
- Resource Contention
- Resource Allocation Protocols
  - Non-preemptive critical sections (NPCS)
  - Priority Inheritance (PI)
  - Priority-ceiling protocol (PCP)
  - Stacked priority-ceiling protocol (SPCP)
- Summary

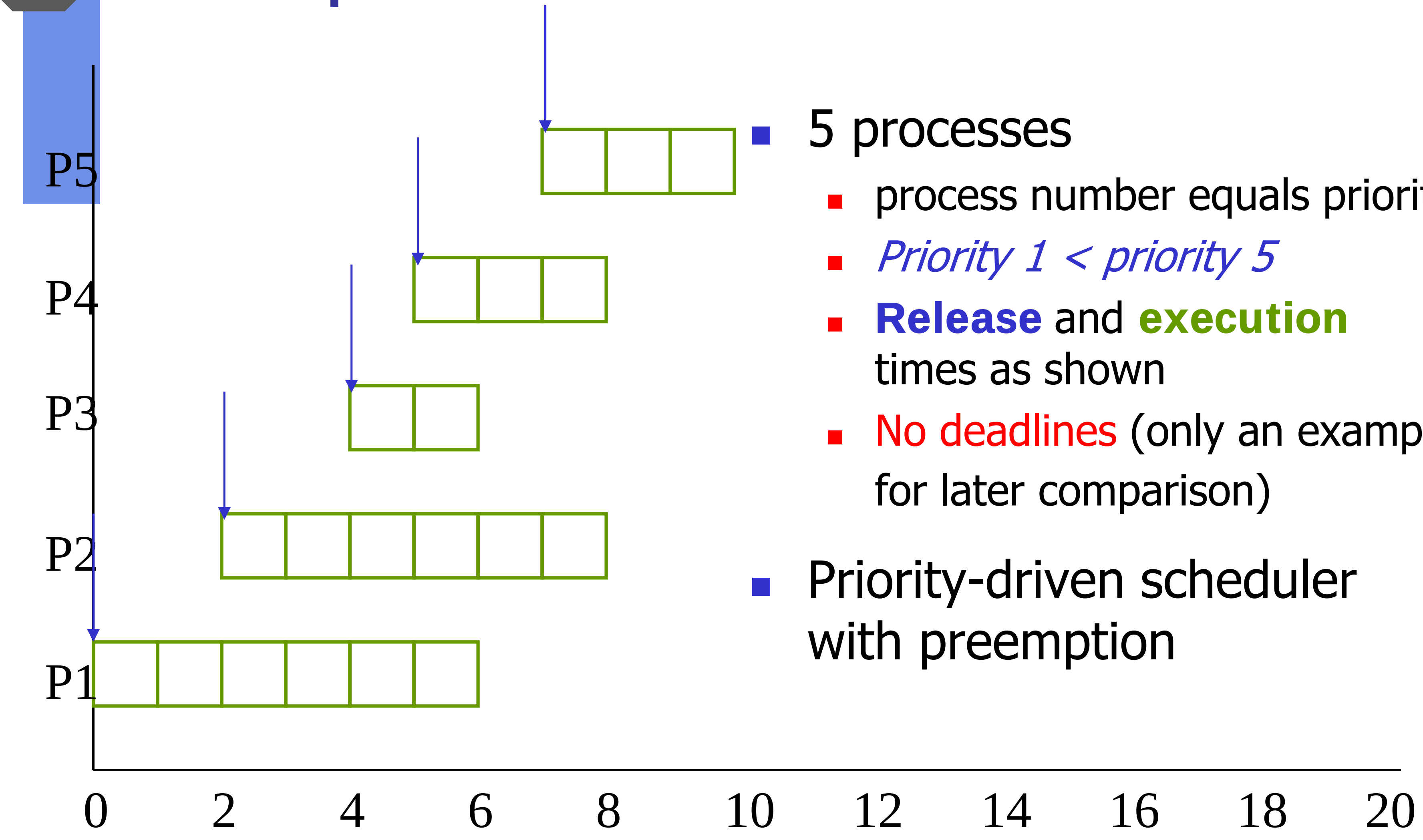
Real-Time Processes

- Process = unit of work being scheduled and executed on the system.
- Processes have:
  - Release time or available time
  - *Worst-case execution time*
  - (Relative) Deadline
  - Sporadic or periodic characteristic
- Processes are scheduled such that deadlines are always met (*hard real time*).

Scheduling

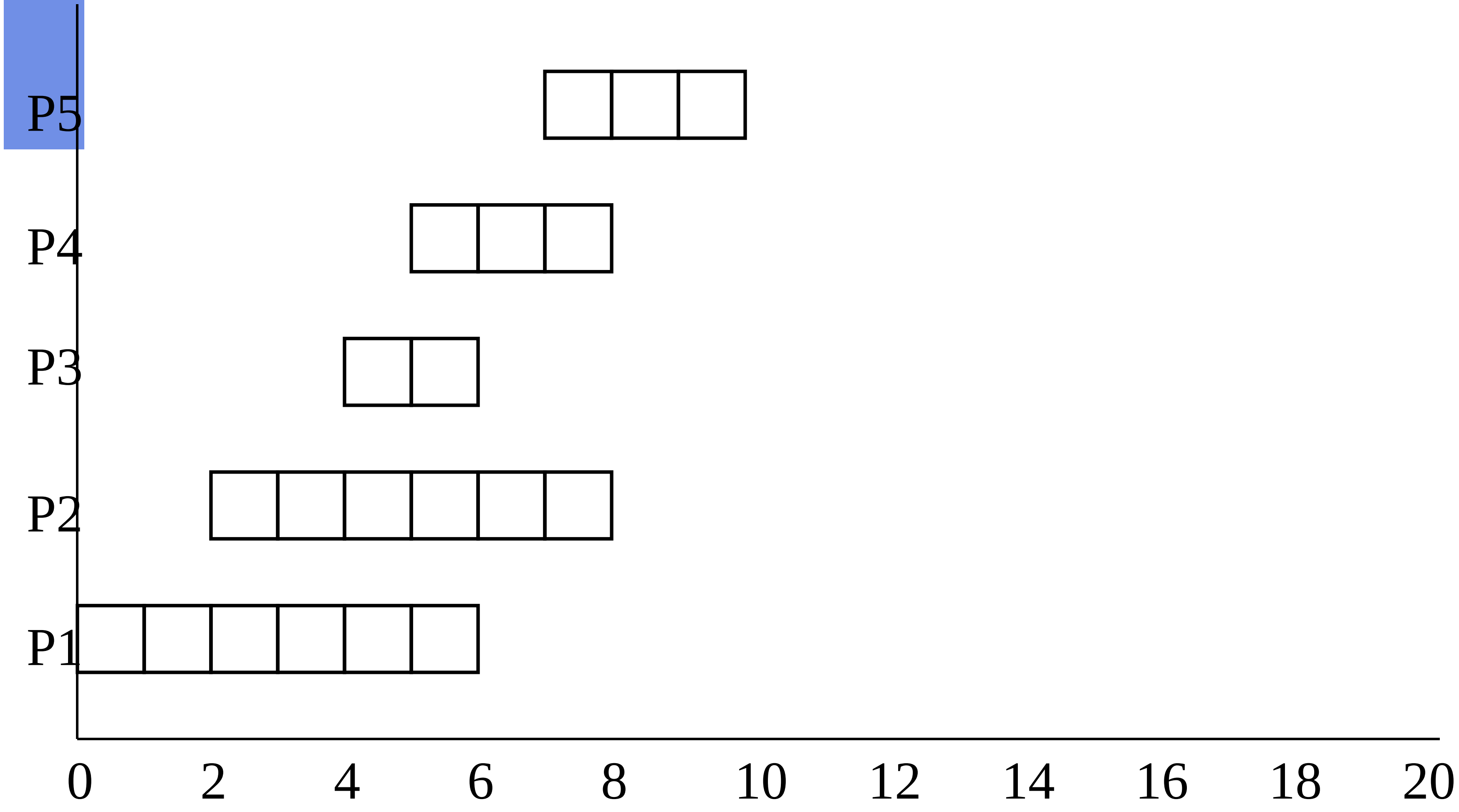
- Common scheduling policy
  - *Priority driven preemptive scheduling*
    - High priority process is always scheduled in preference to low priority process
    - High priority value = high priority
  - Priorities can be assigned according to some algorithm
    - Rate monotonic
    - Earliest deadline first
  - We will focus on *static priorities*

Example

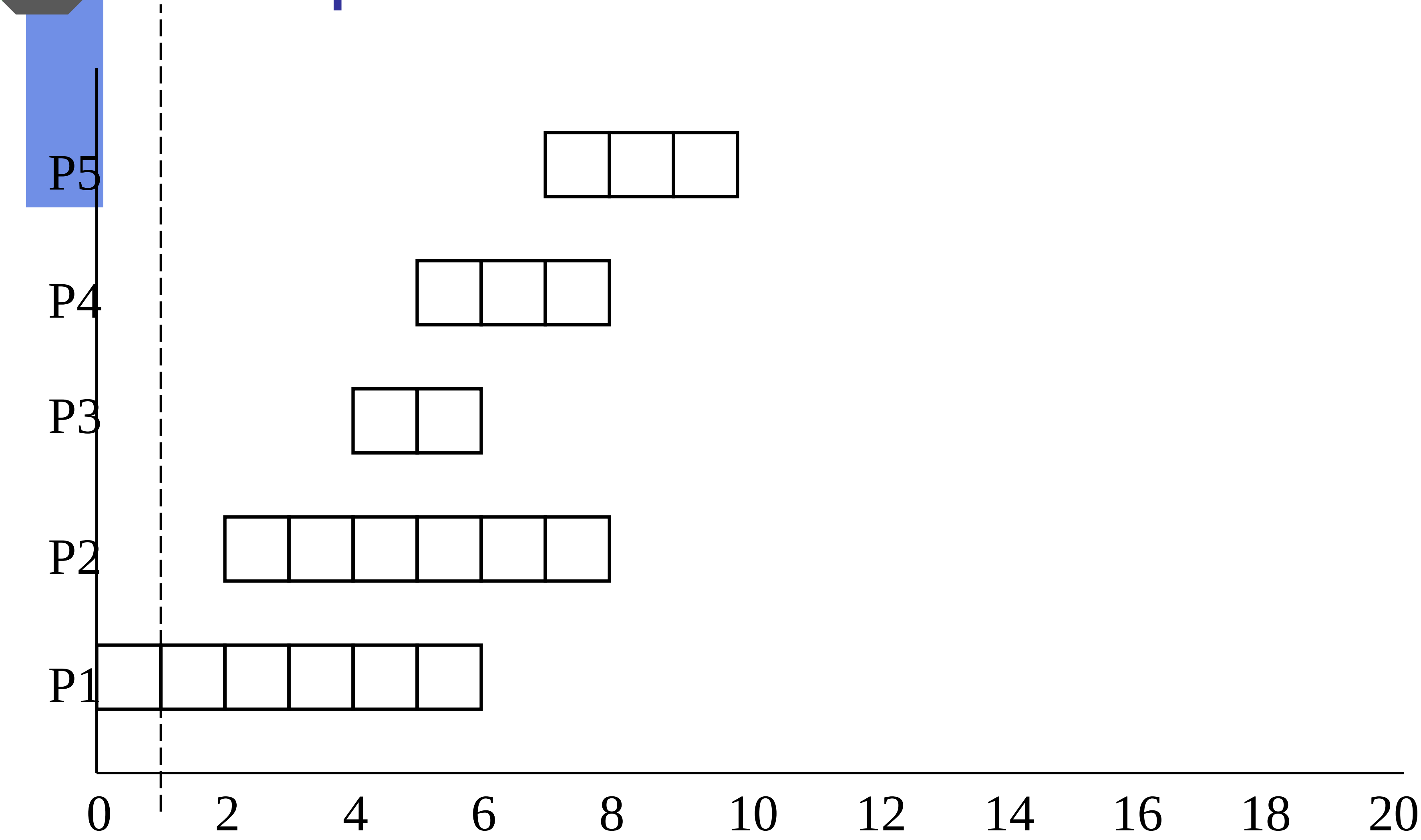


- 5 processes
  - process number equals priority
  - *Priority 1 < priority 5*
  - **Release** and **execution** times as shown
  - No deadlines (only an example for later comparison)
- Priority-driven scheduler with preemption

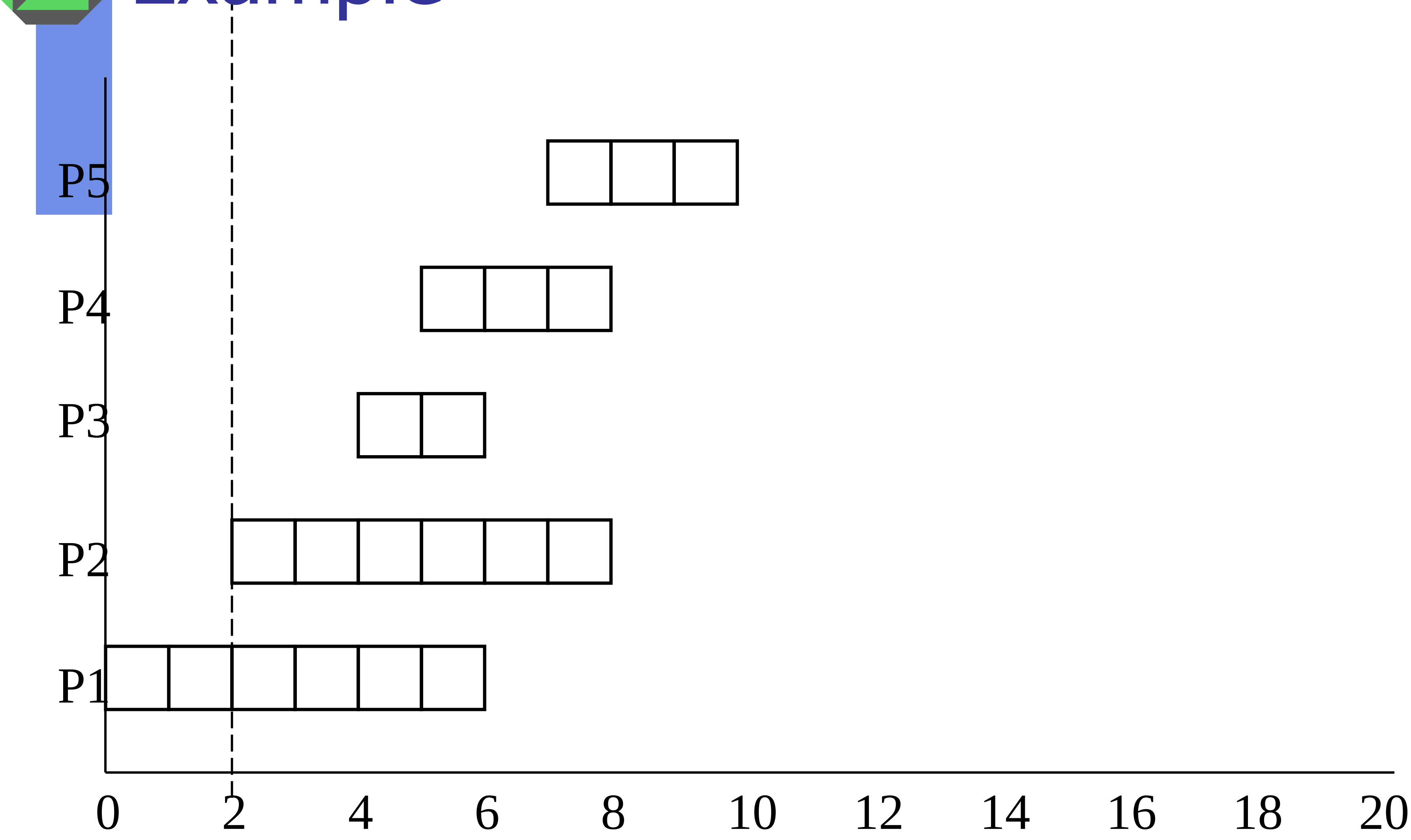
Example



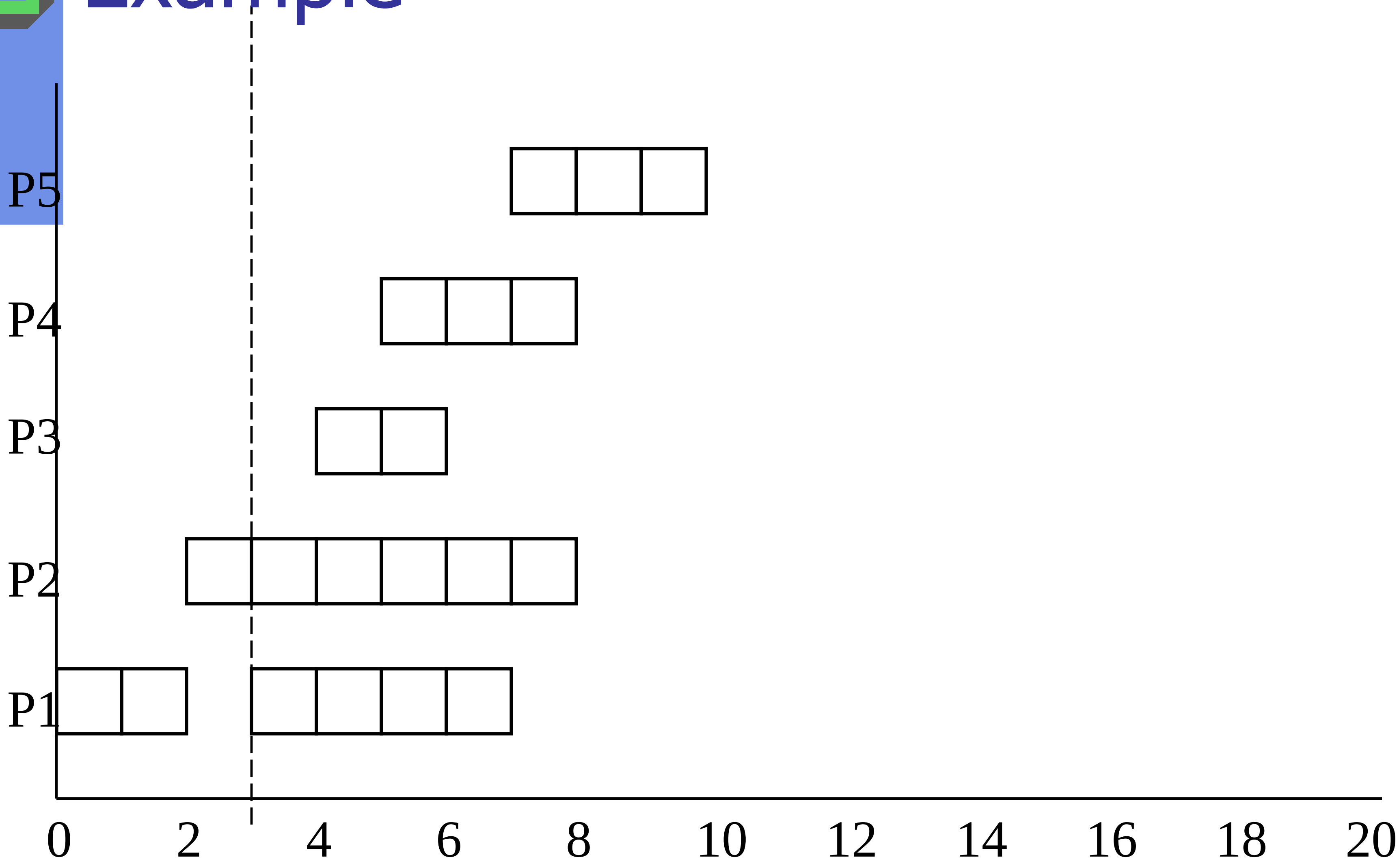
Example



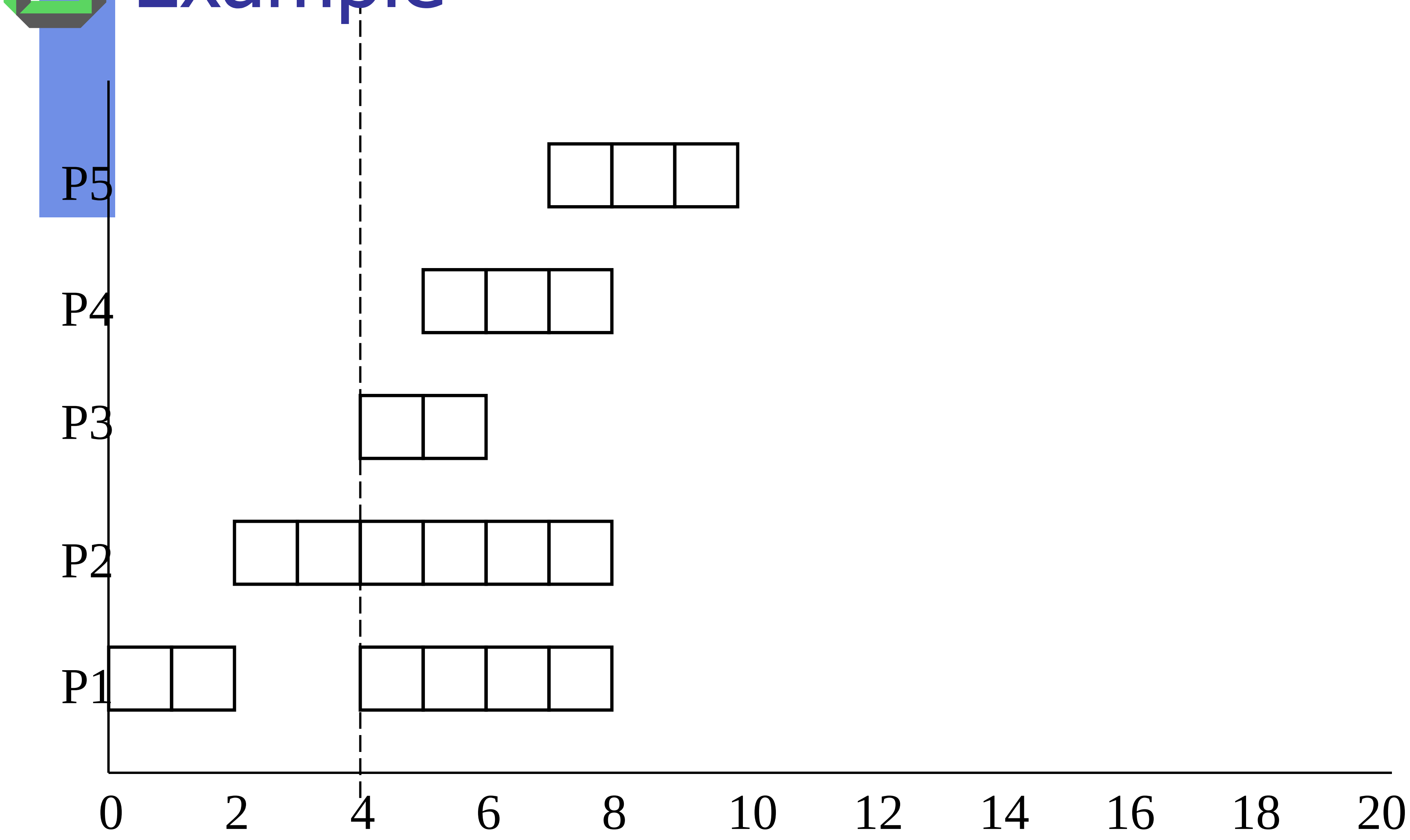
Example



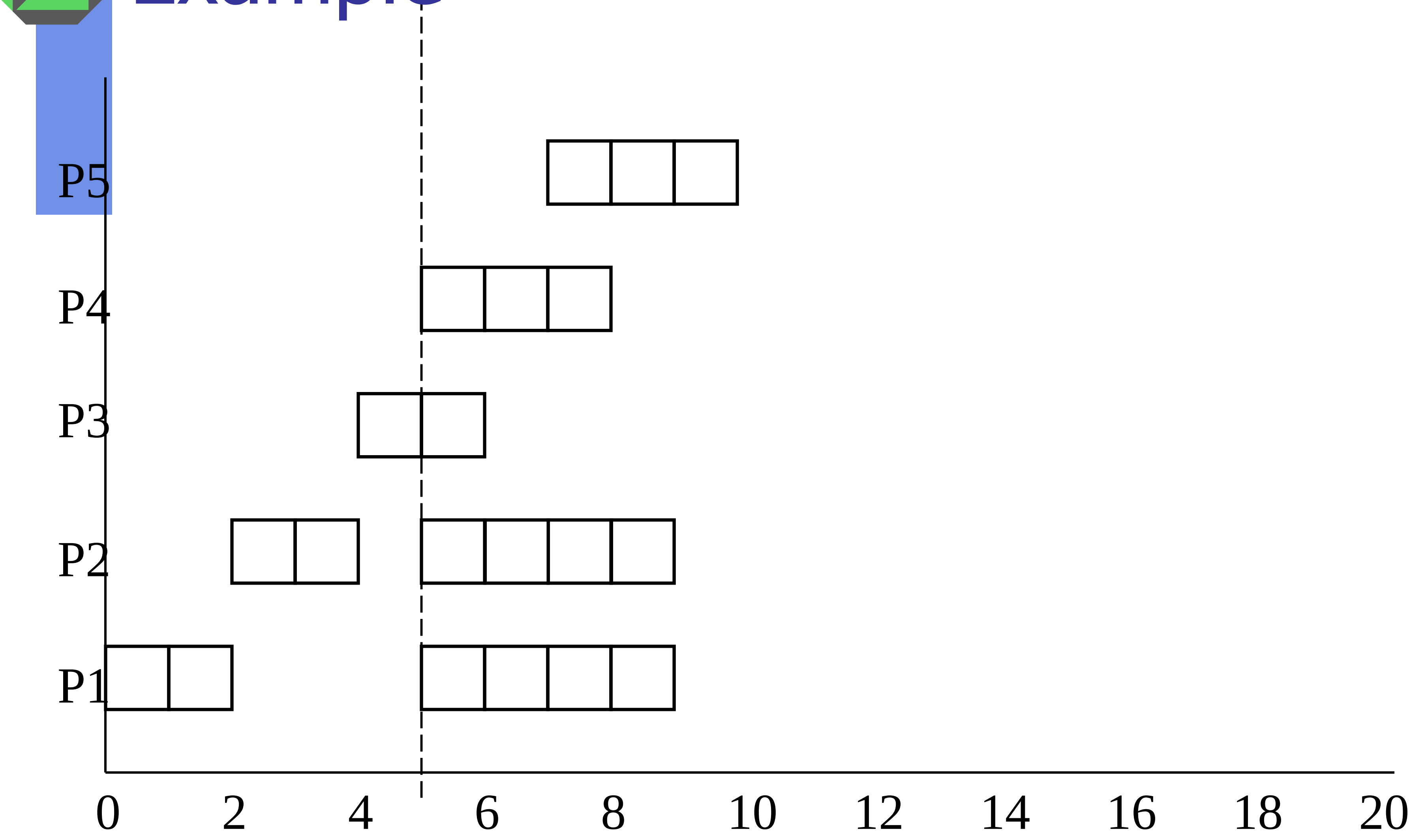
Example



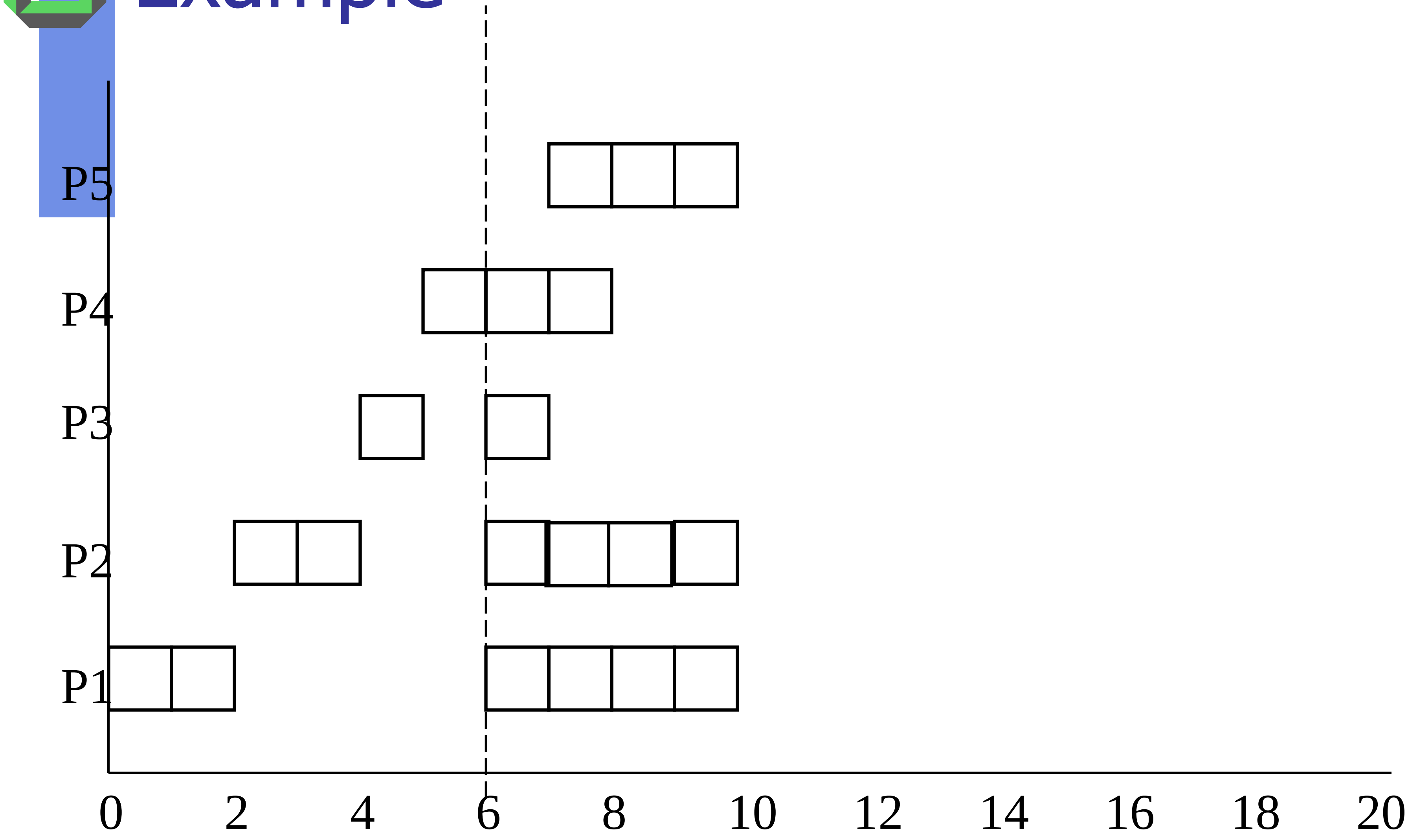
Example



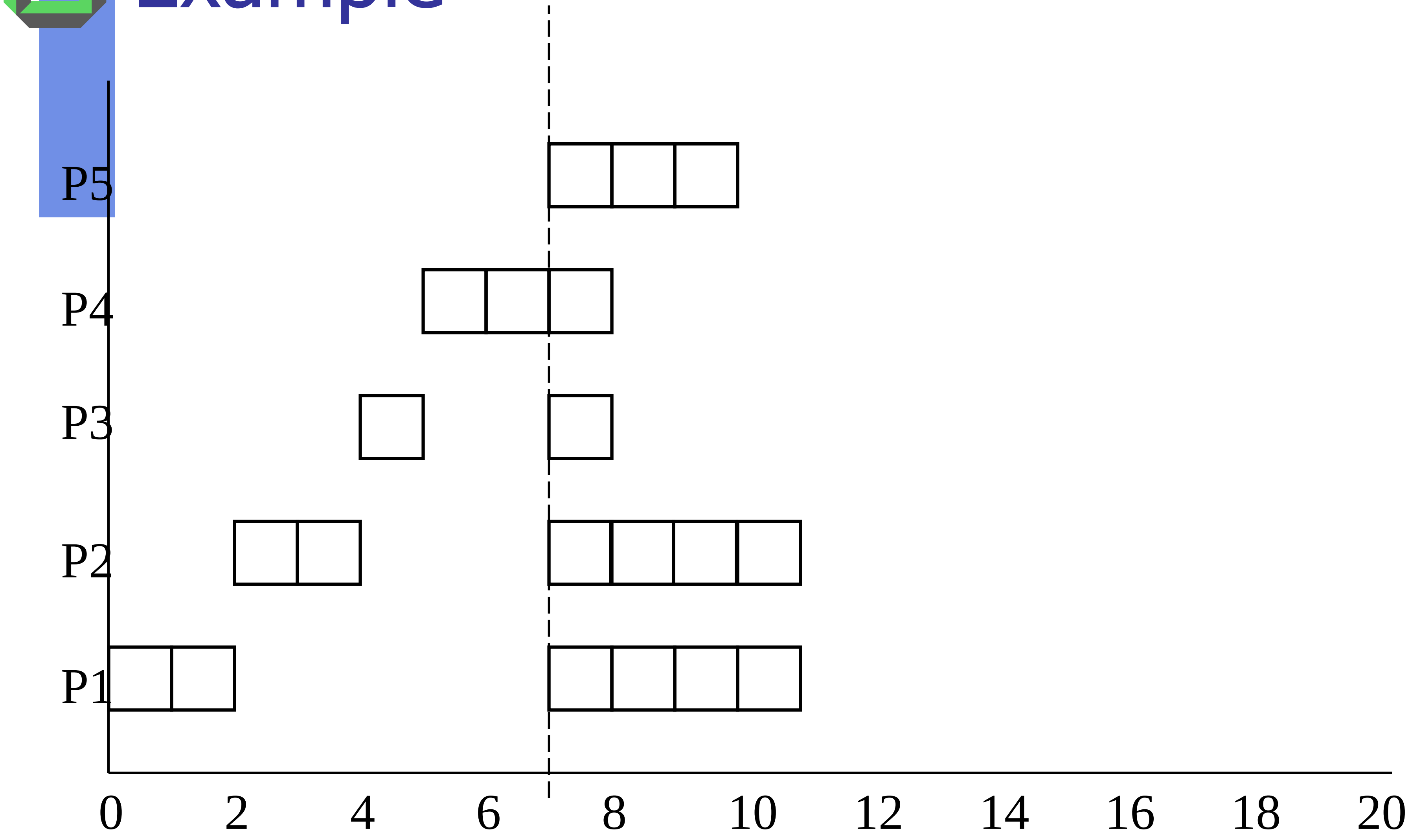
Example



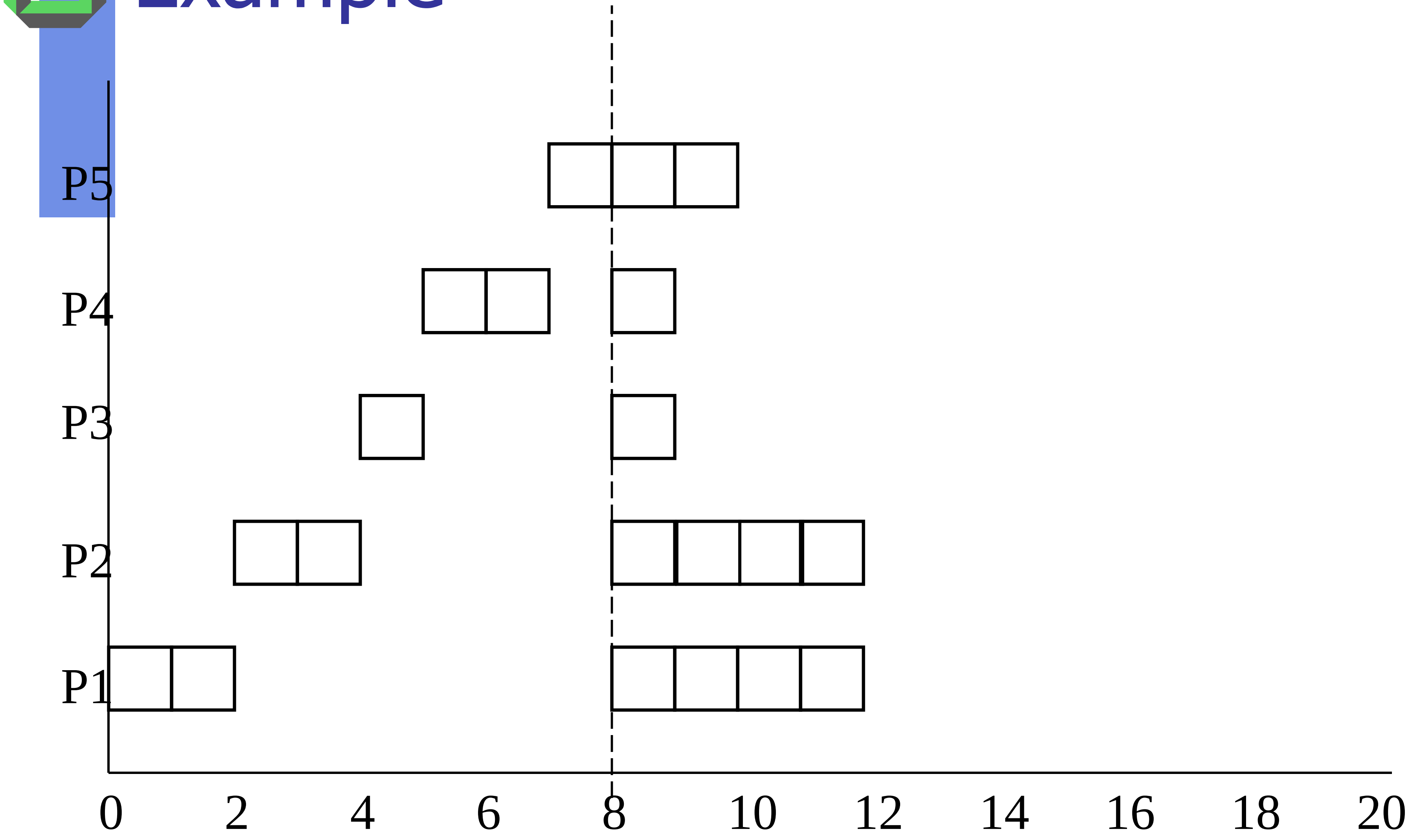
Example



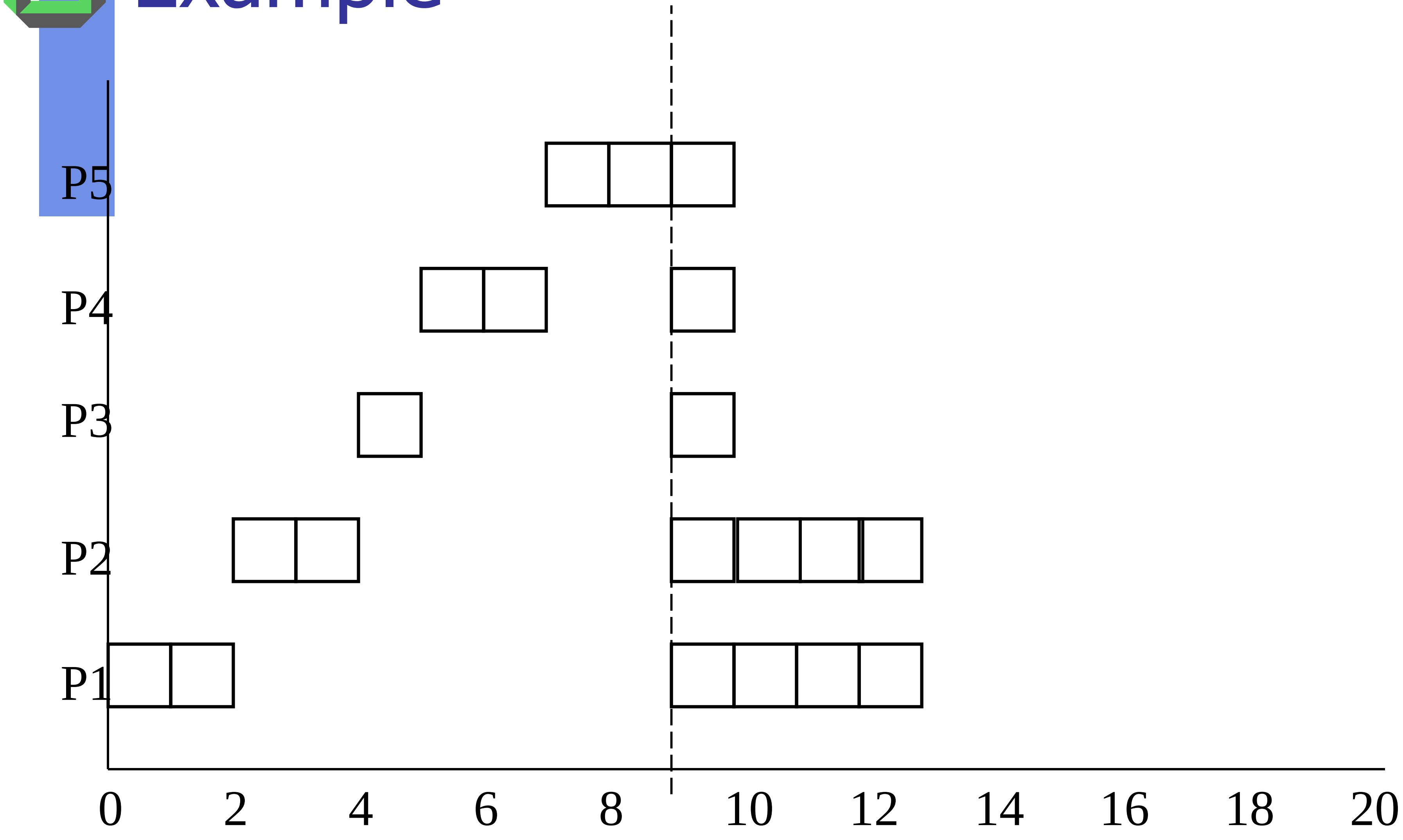
Example



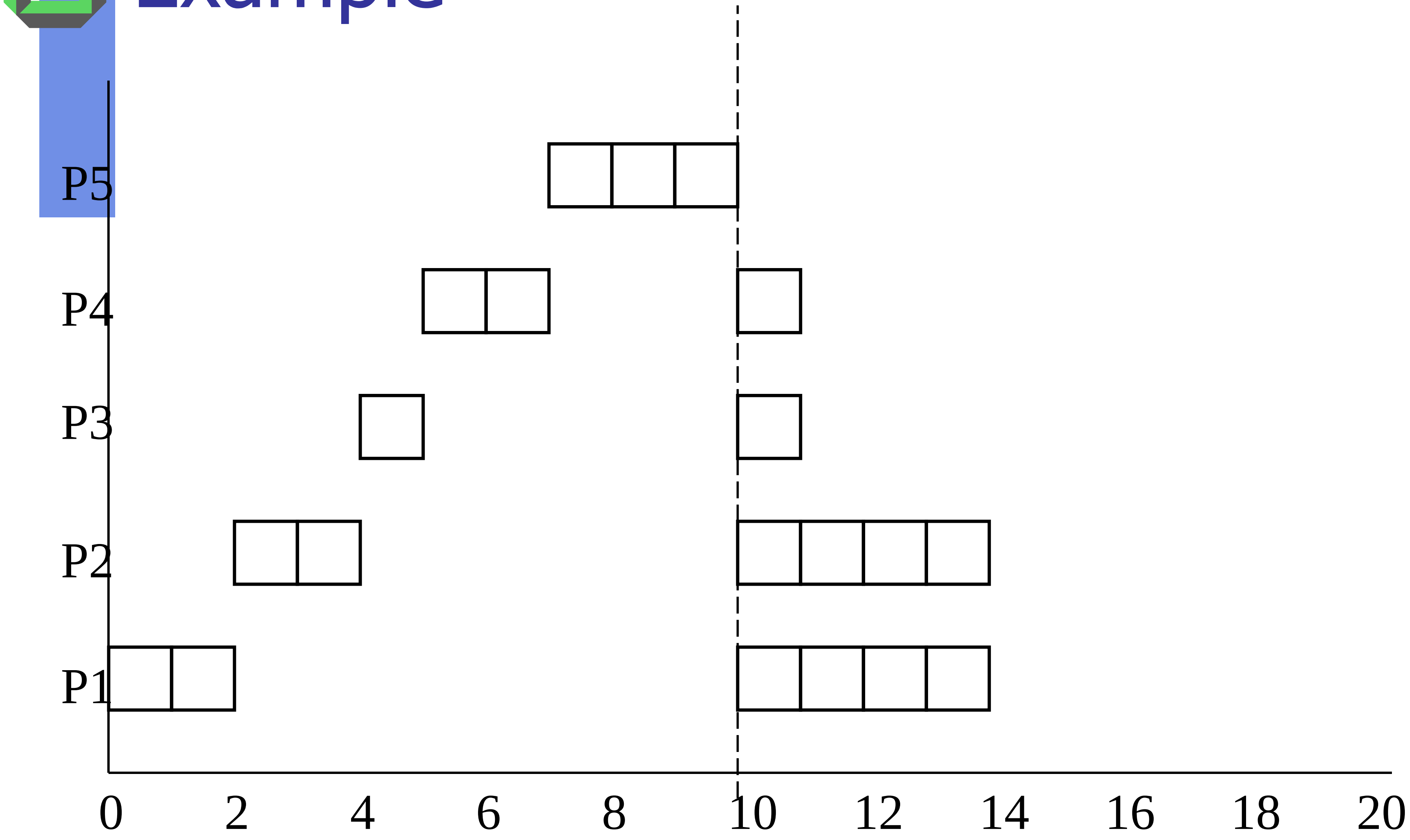
Example



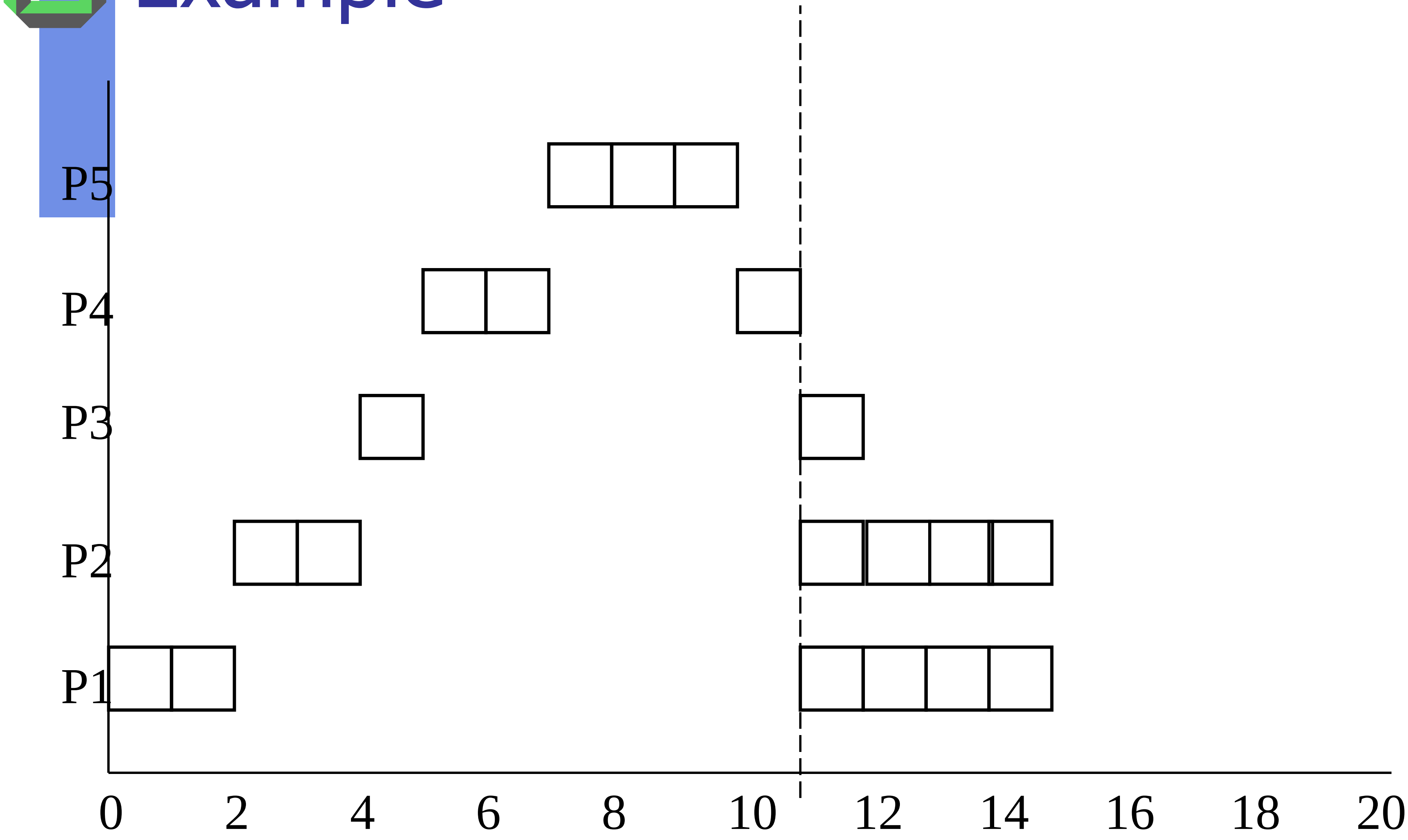
Example



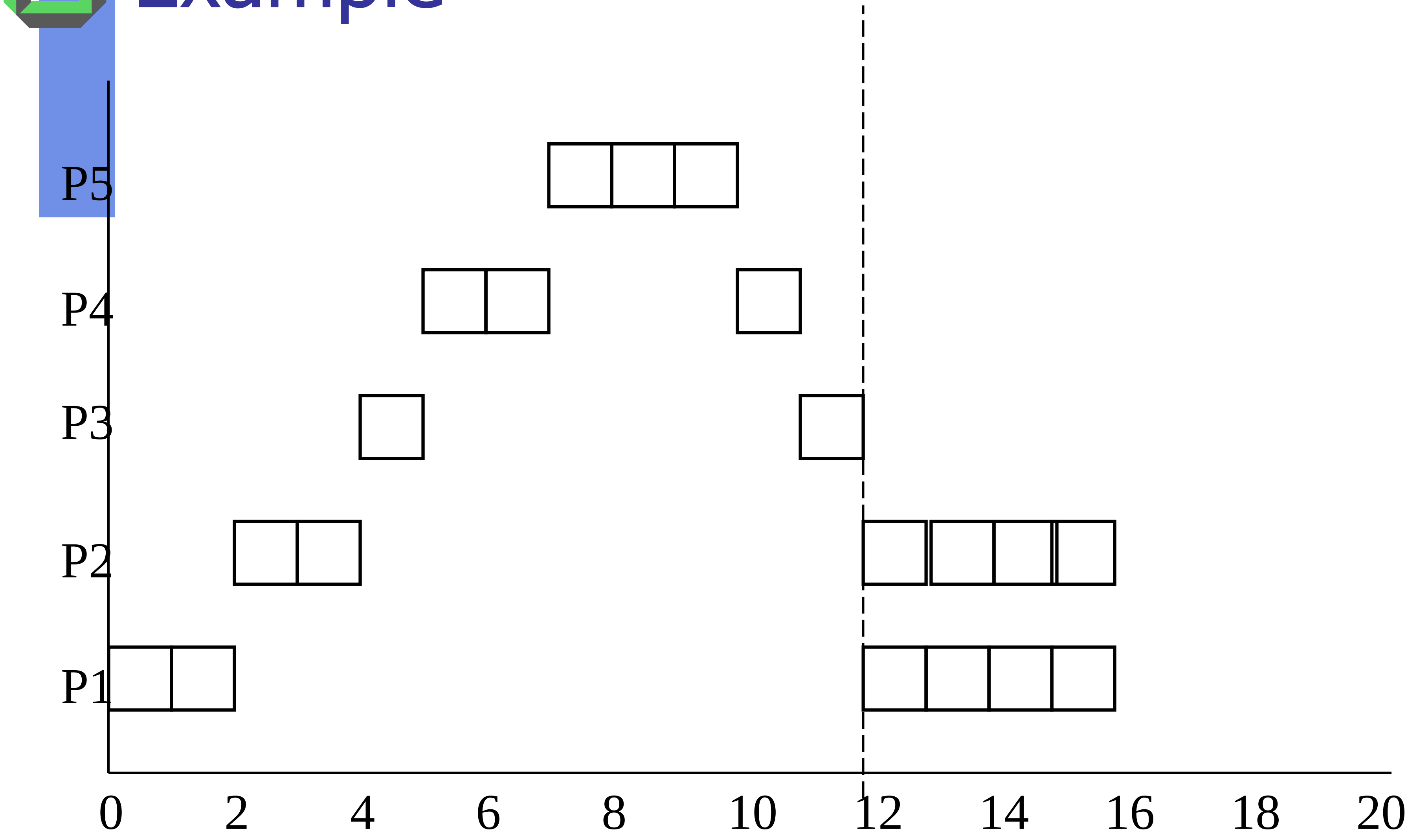
Example



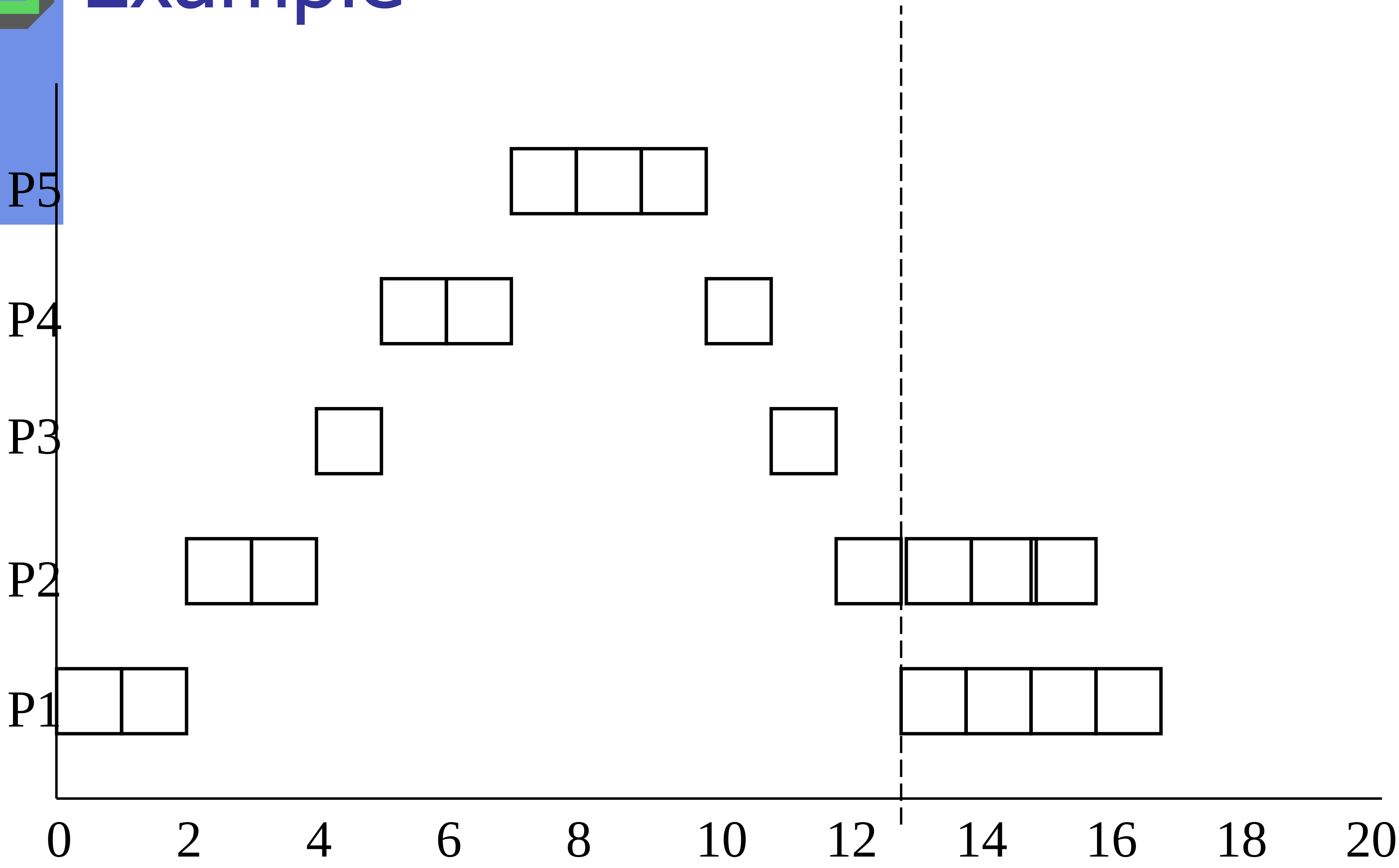
Example



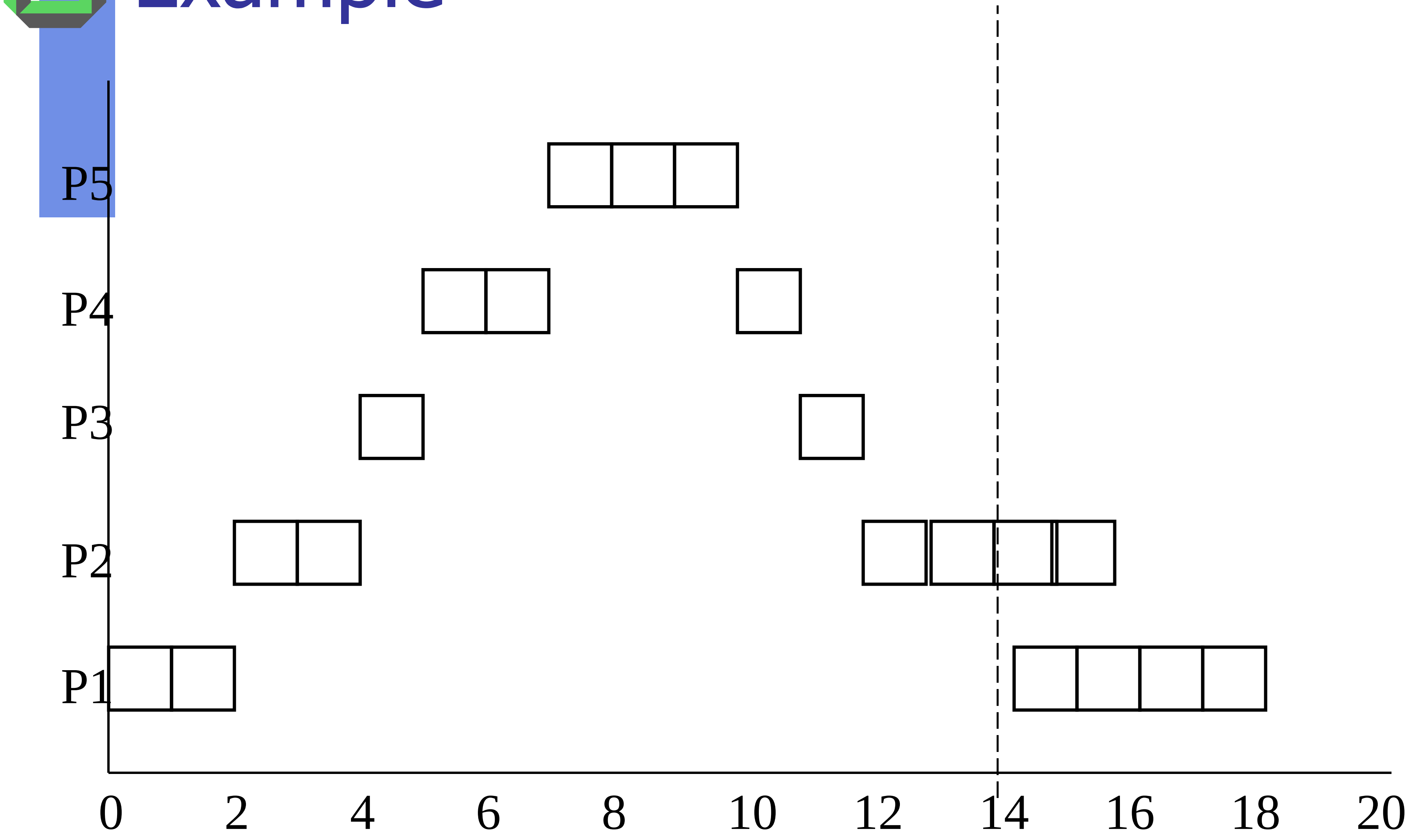
Example



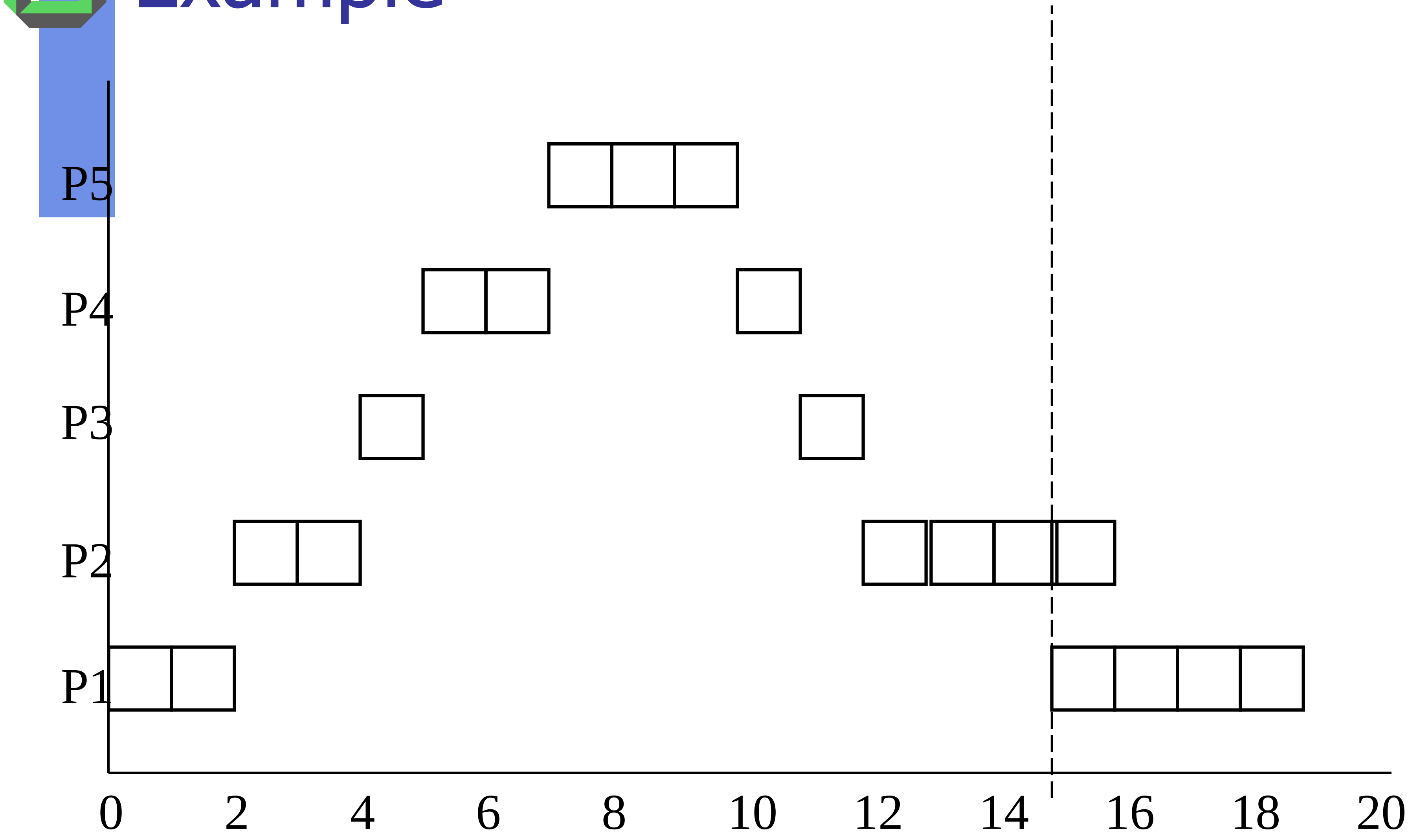
Example



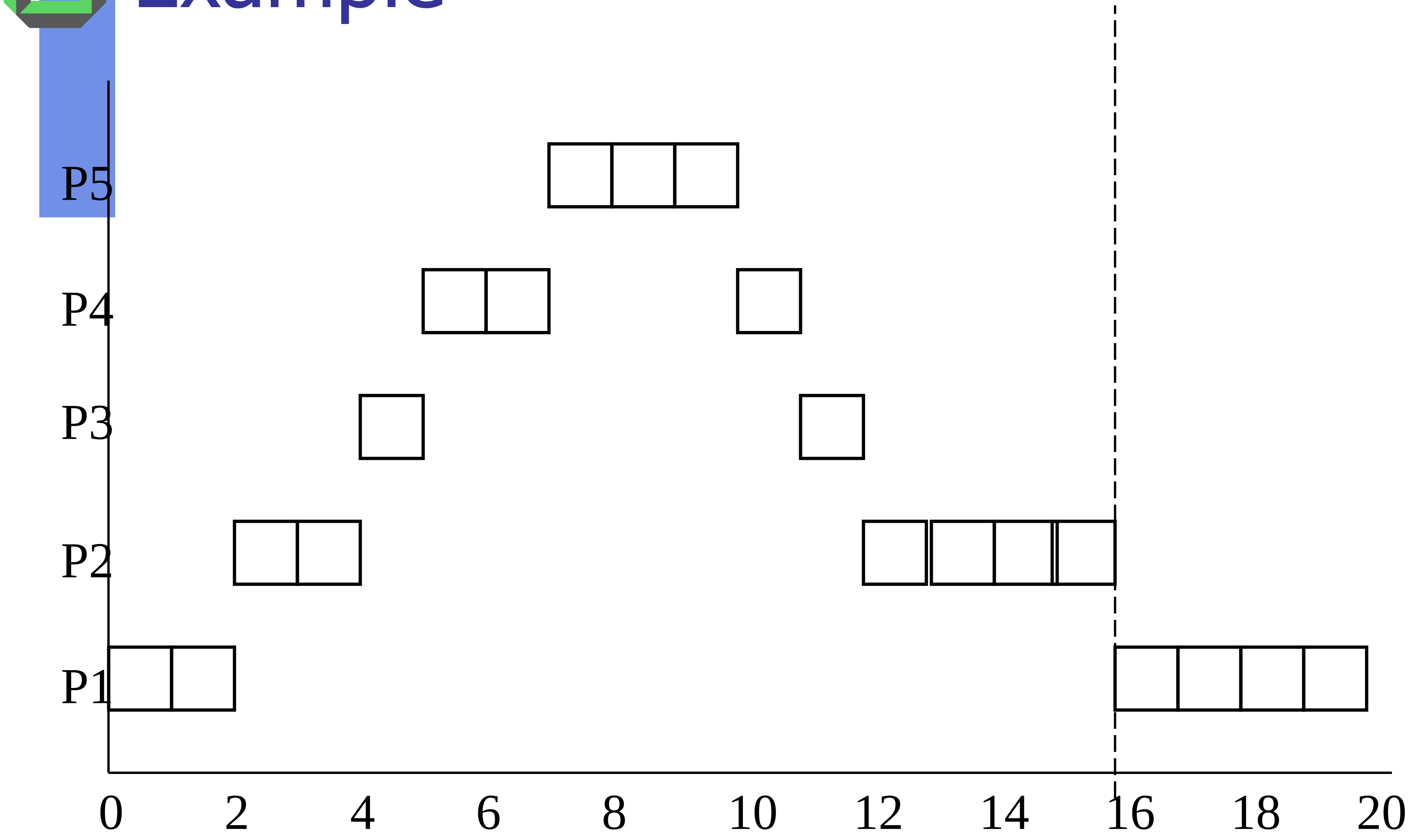
Example



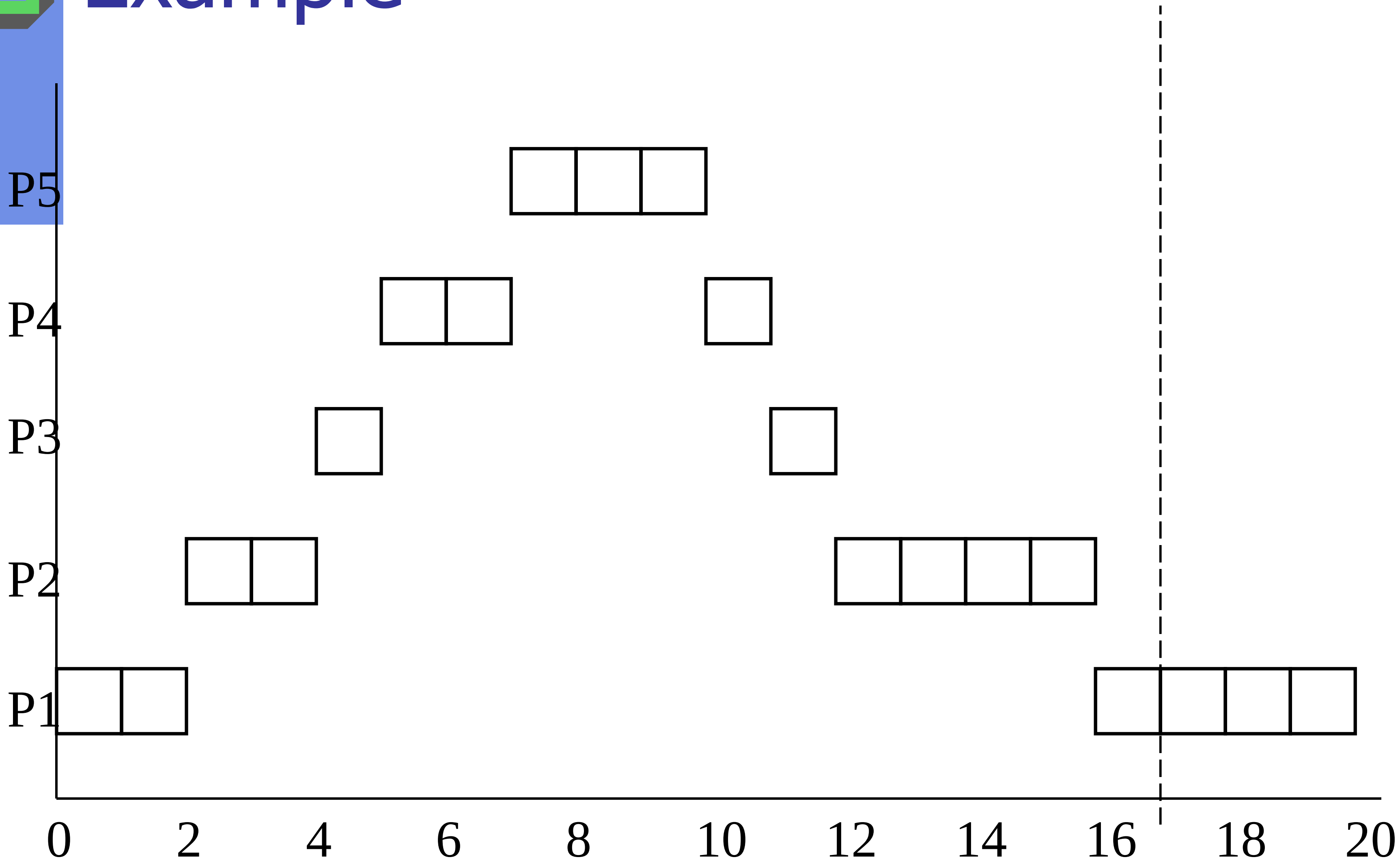
Example



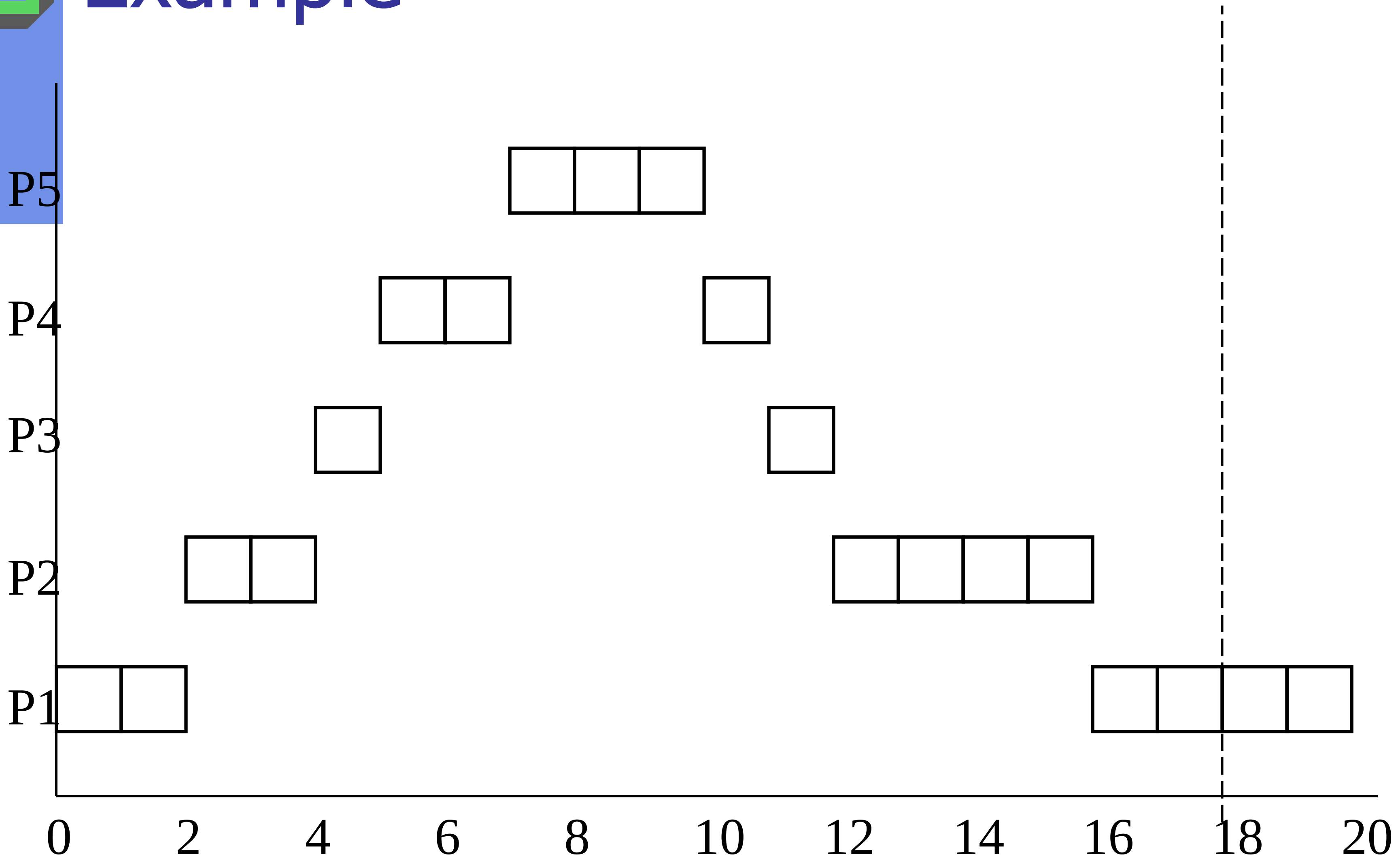
Example



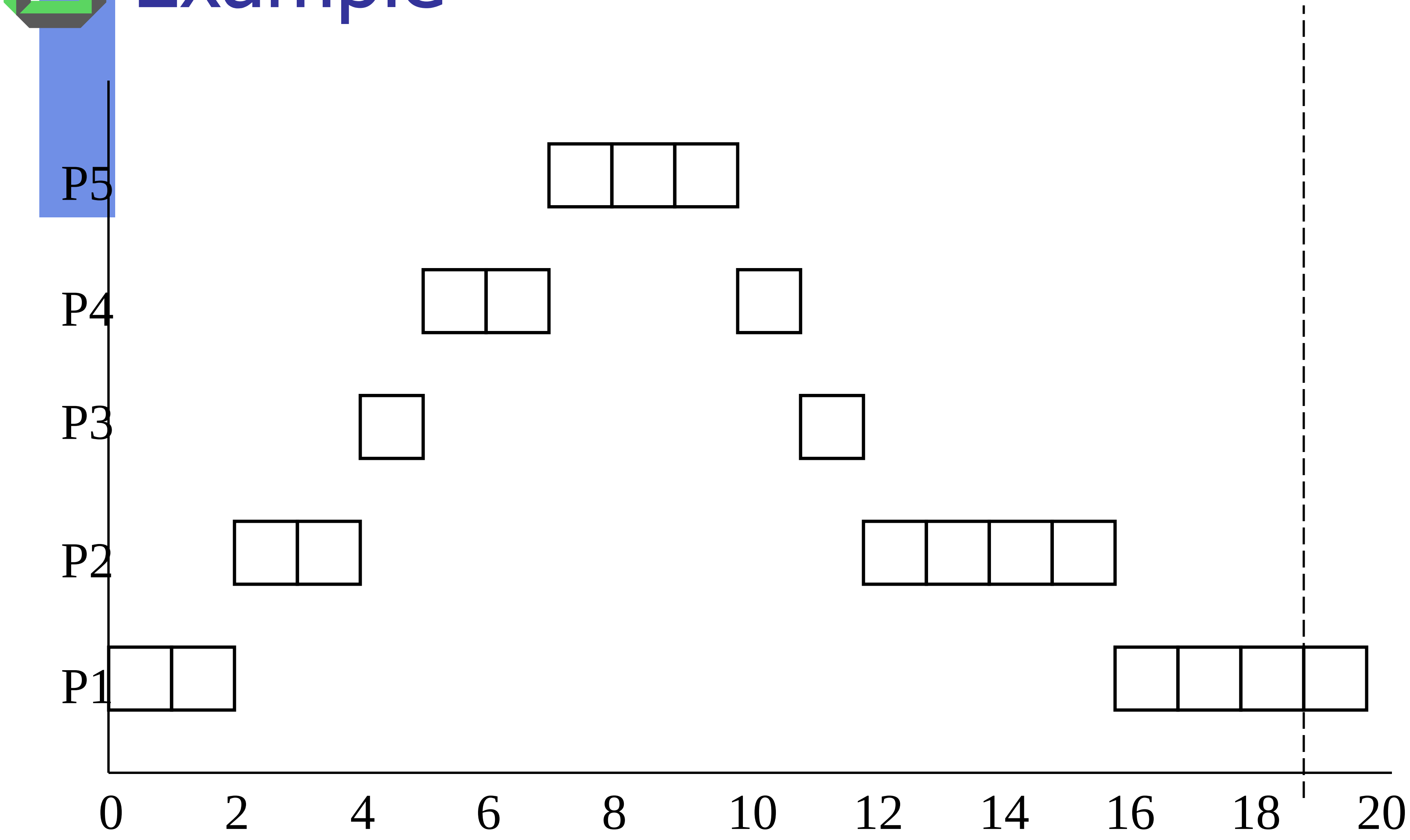
Example



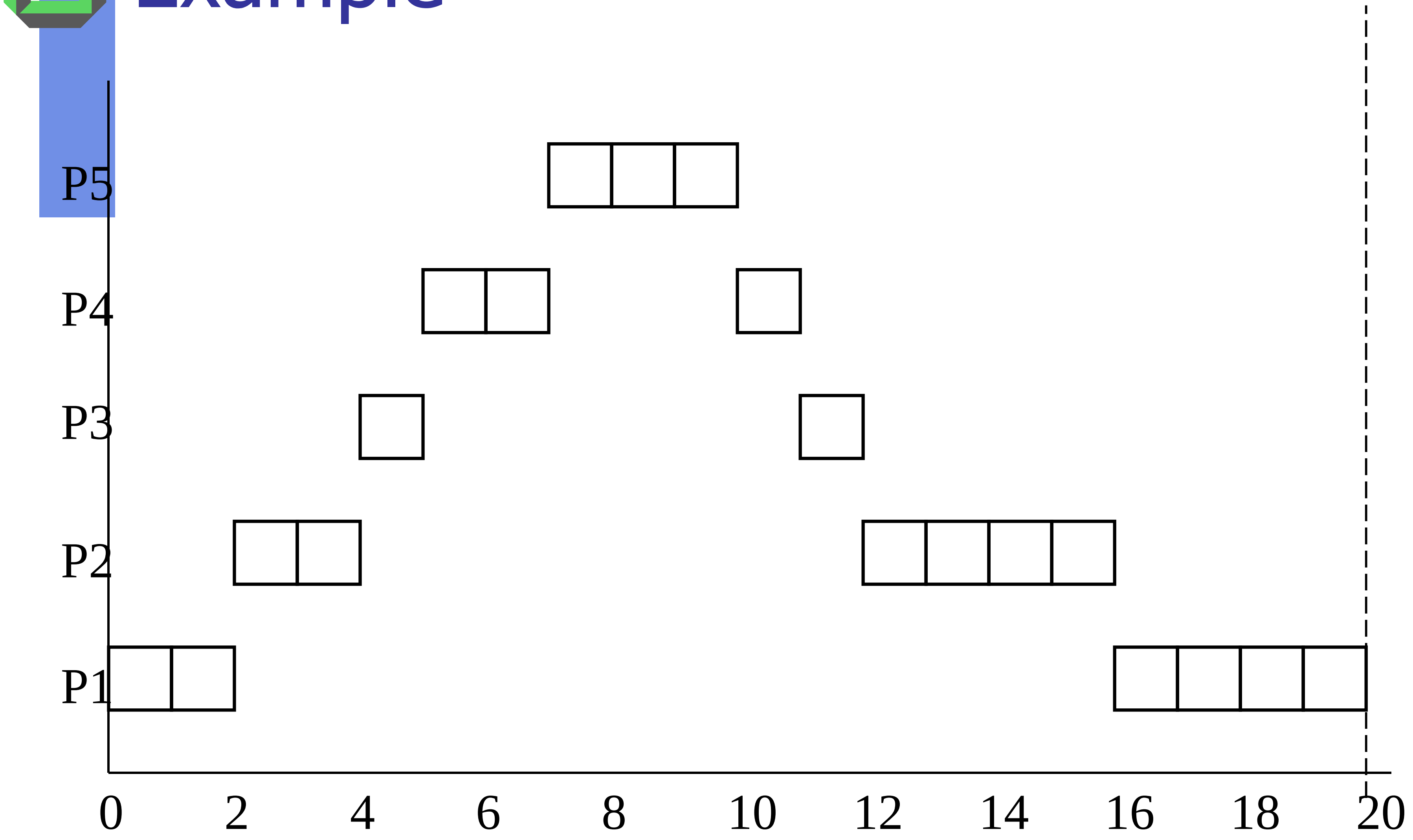
Example



Example



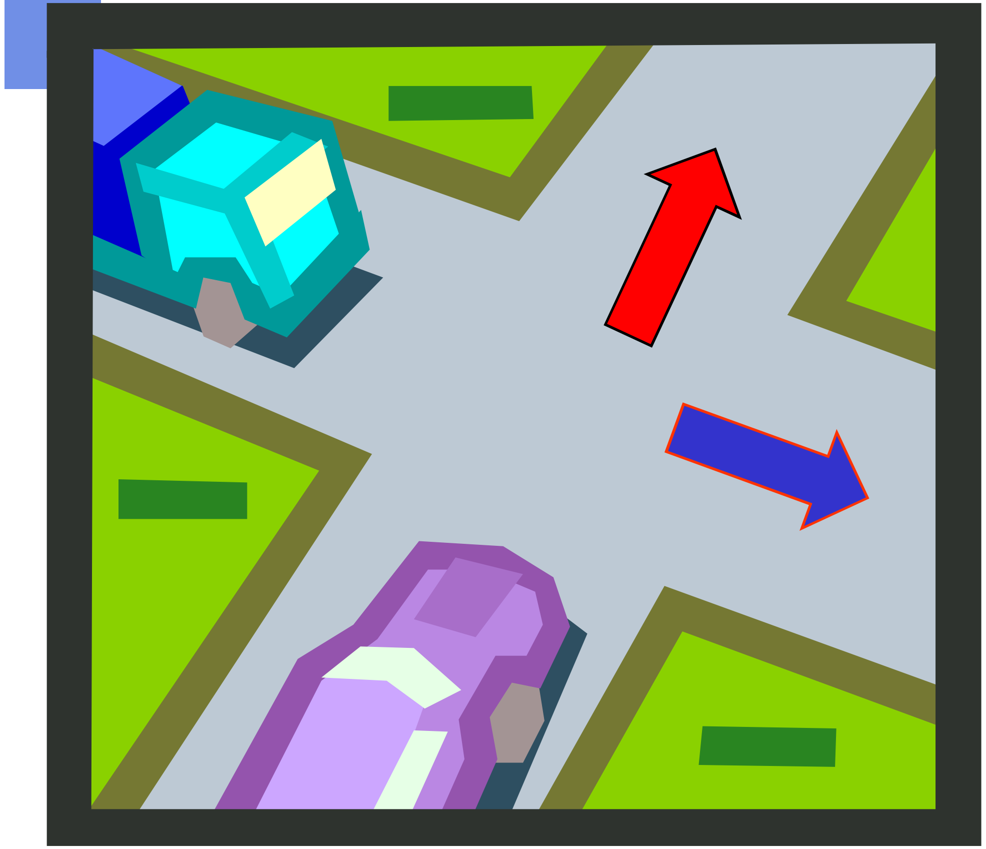
Example



Reality is more complex

- Usually processes are not independent
- They compete for resources or rely on each other’s intermediate results

Real-Time Traffic Scheduling



- Two process streams
- A high priority & a low priority

Priorities and Resource Contention

Main Reference

Pane W. S. Liu “Real-time Systems”, Chapter 8

Resources

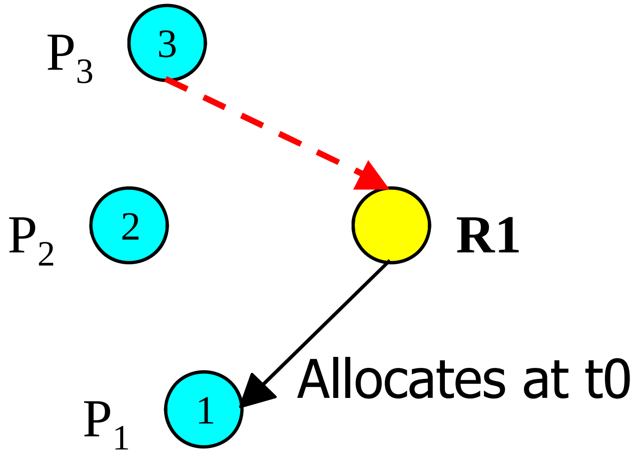
- Processes require resources in order to execute (e.g. locks, ports, memory, …)
- Resource characteristics
  - *serially reusable*
  - *mutually exclusive*

⇒ we ignore resources that

- are infinitely available or exceed demand
- or can be pre-allocated

Resource Contention Problem

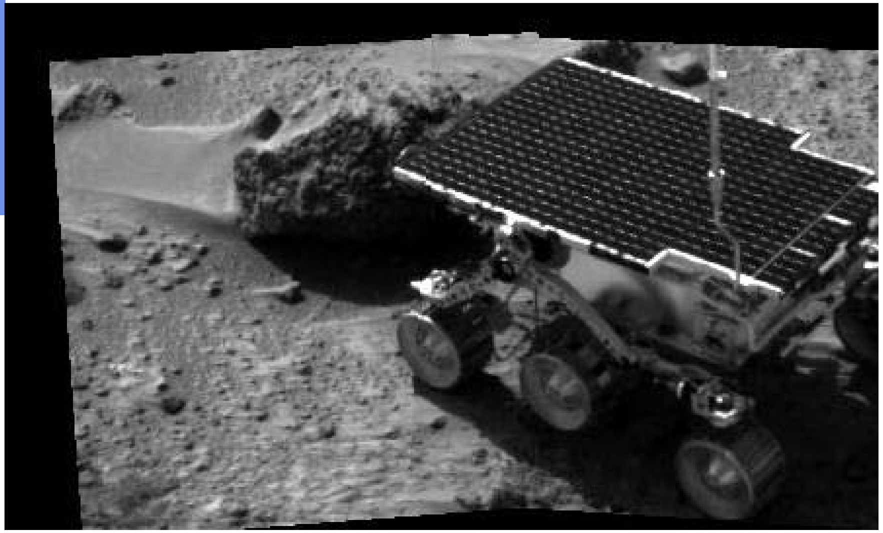
- Priority inversion, given 3 processes, and a resource R1
  - We need to, at least, *bound* the length of priority inversion
  - Preferably *minimize* the length of priority inversion



Famous example of priority inversion:

Mars Path-Finder 1997

Mars Pathfinder



Mars Path Finder and …

the famous Mars "rock" YOGI

Read the following papers:

Mick Jones: *What really happened on the Mars?*

How did they fix the problem?

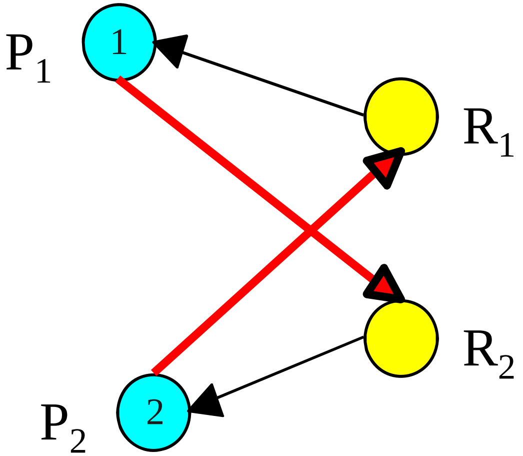
http://www.research.microsoft.com/~mbj/ and

http://www.research.microsoft.com/~mbj/Mars_Pathfinder/Authoritative_Account.html

by **Glenn Reeves**, chief of the software team of Mars-Pathfinder software

Resource Contention Problems

- Timing anomaly (e.g. convoy problem)
- Deadlock



One Class of Solutions

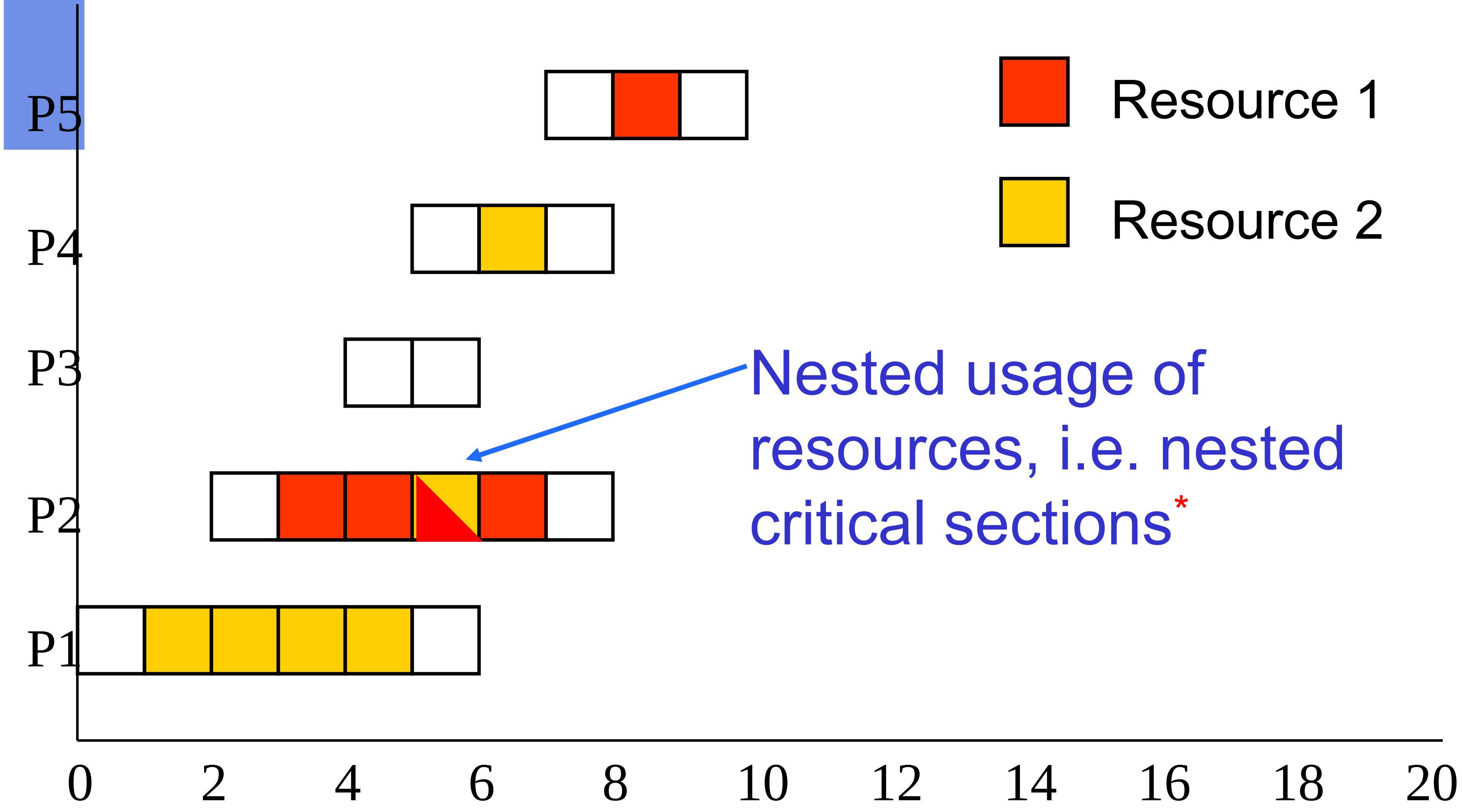
- Use a resource allocation protocol that
  1. bounds priority inversion
  2. avoids deadlock

- Estimate worst-case blocking time due to resource contention
  - Combine blocking time and execution time
    - Use in admission control

Major Assumption

- Single processor system

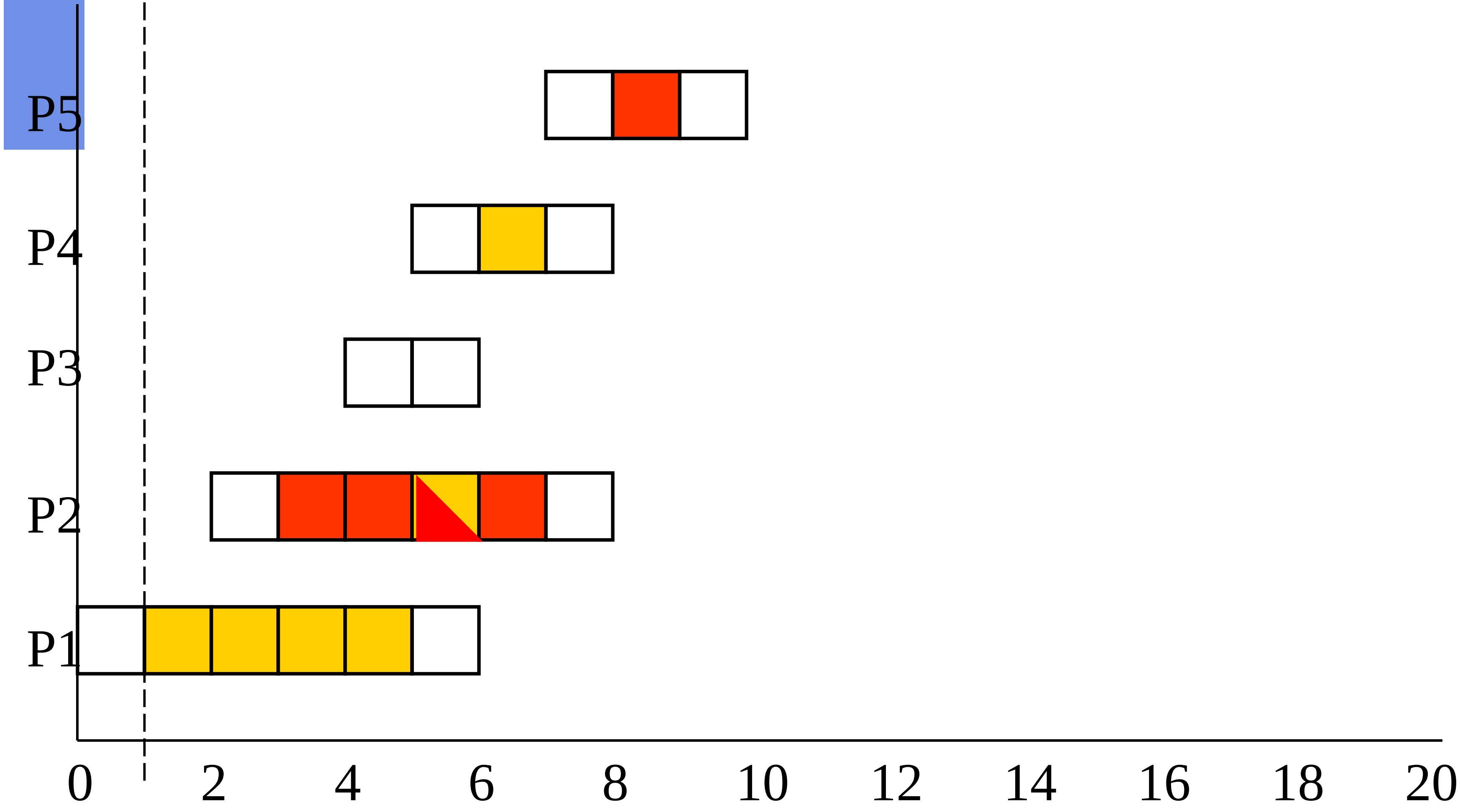
Our Example + 2 Resources



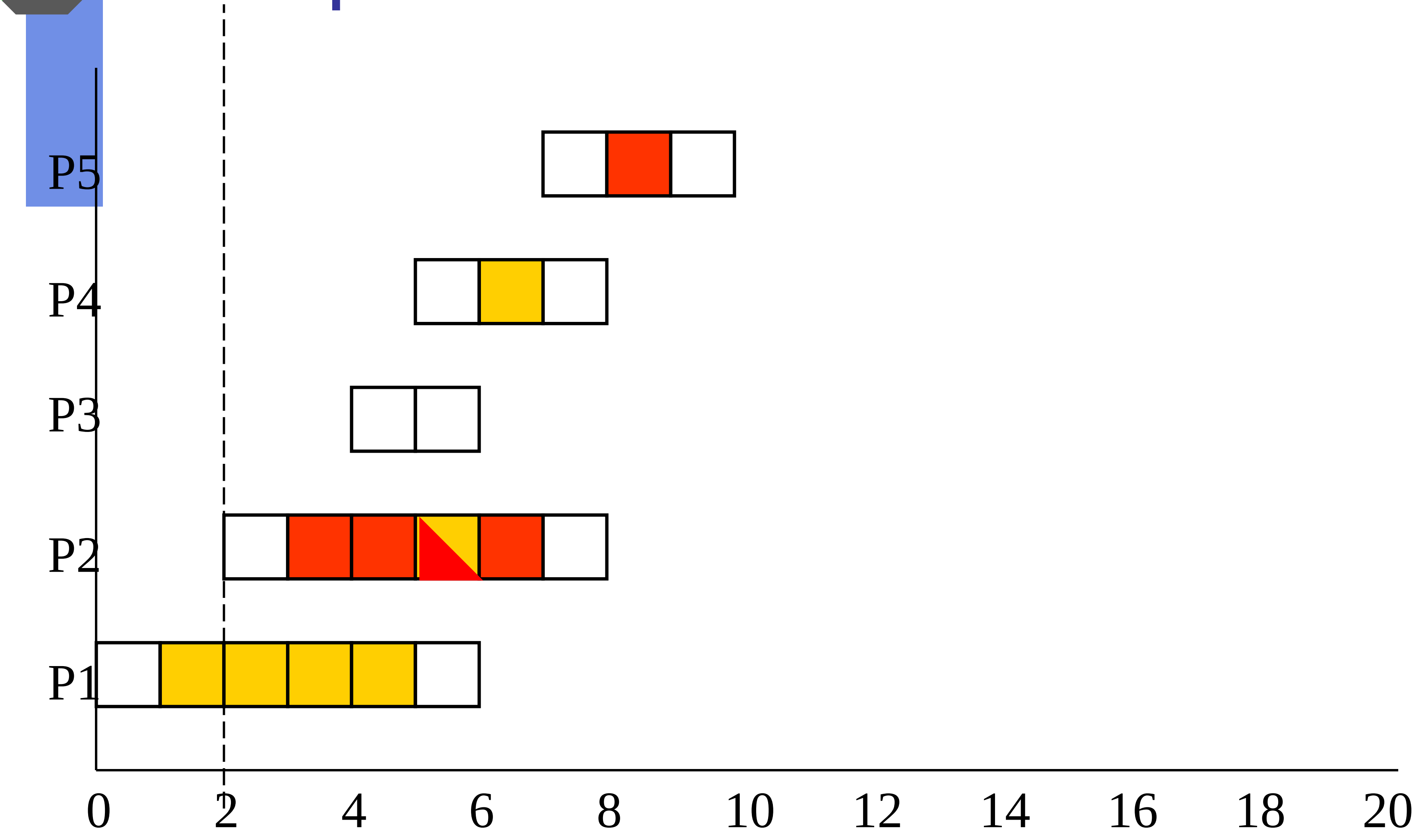
Nested usage of resources, i.e. nested critical sections*

*P2 first needs R1 and then later *additionally* R2

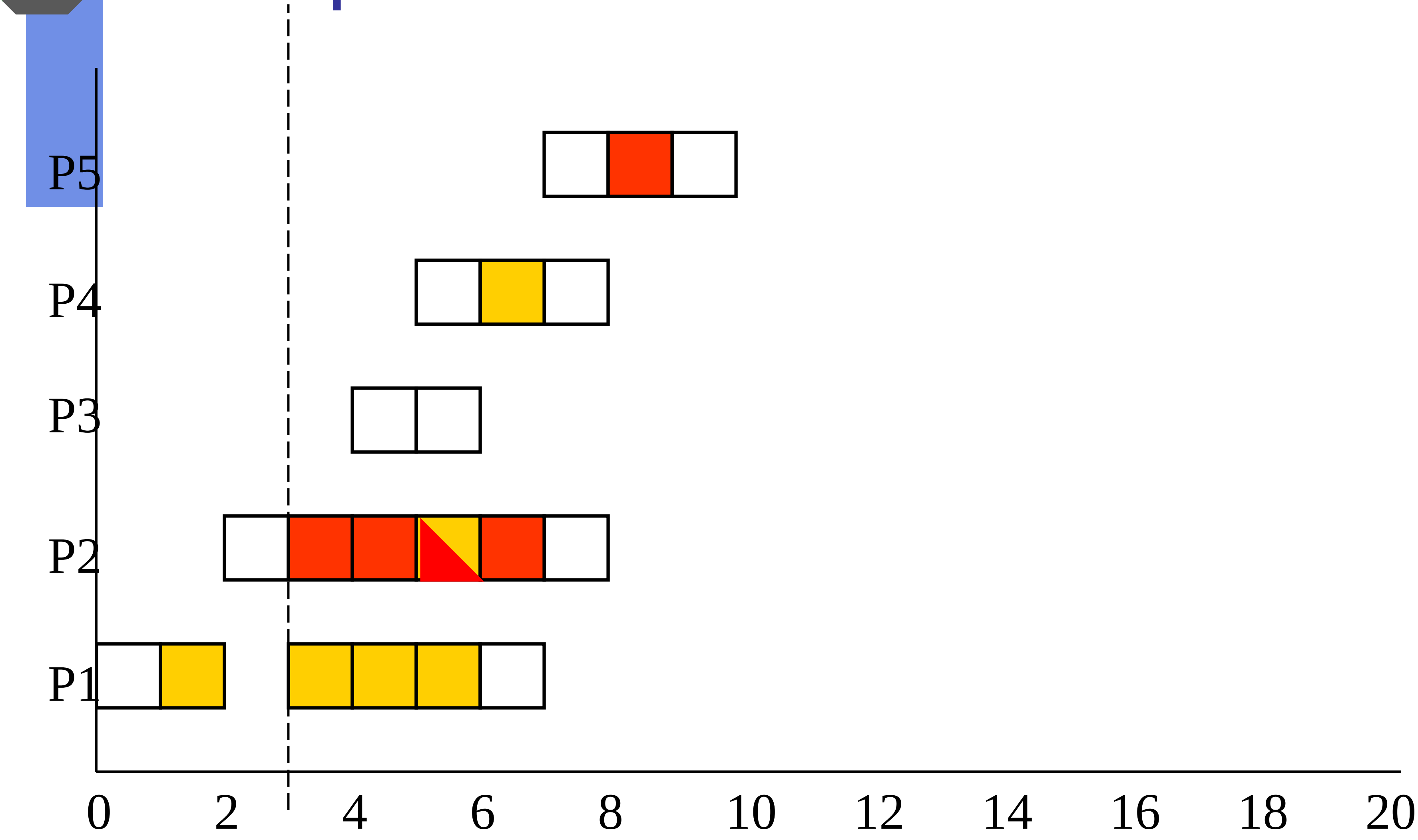
Simple Priority Driven Scheduling (SPD)



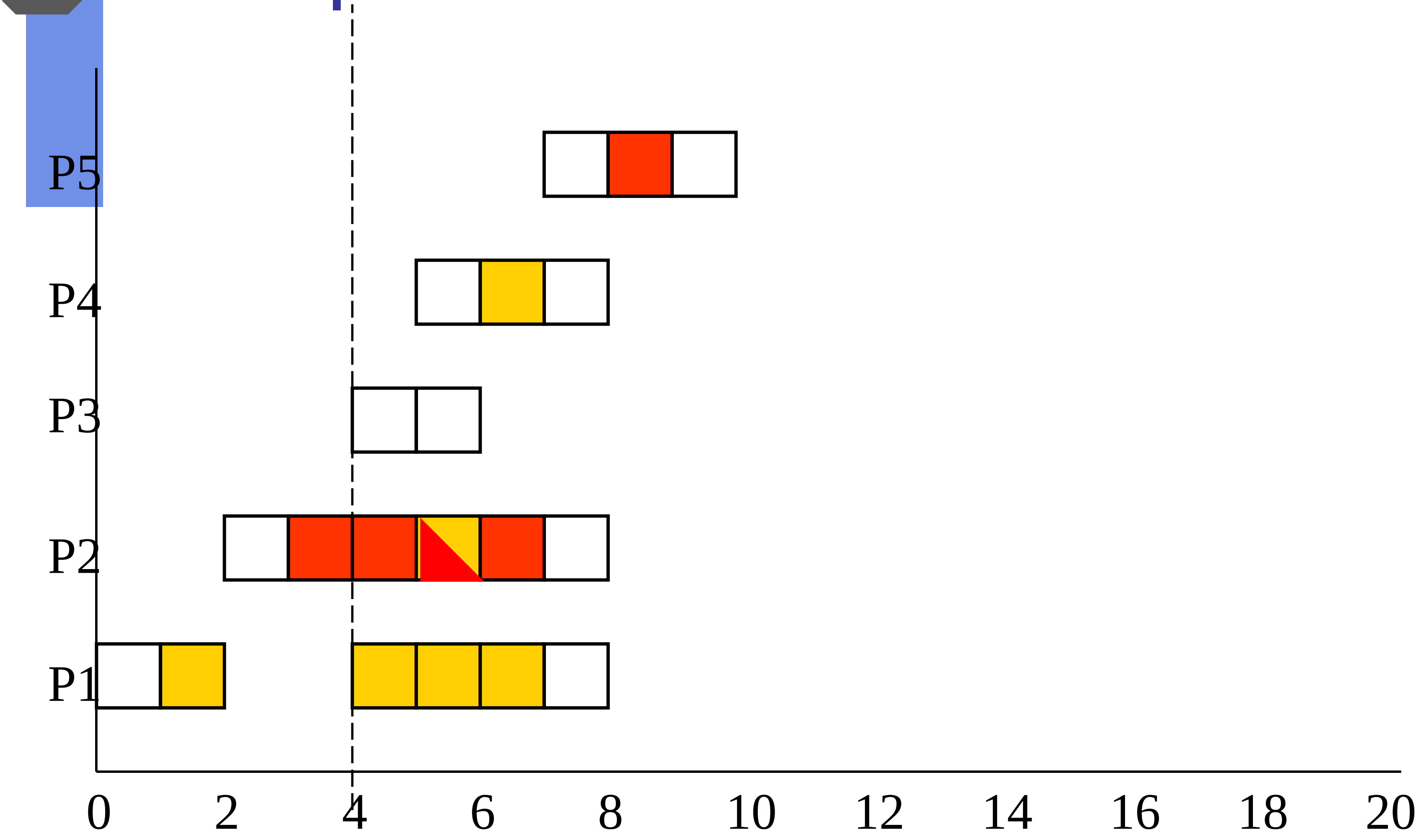
Example



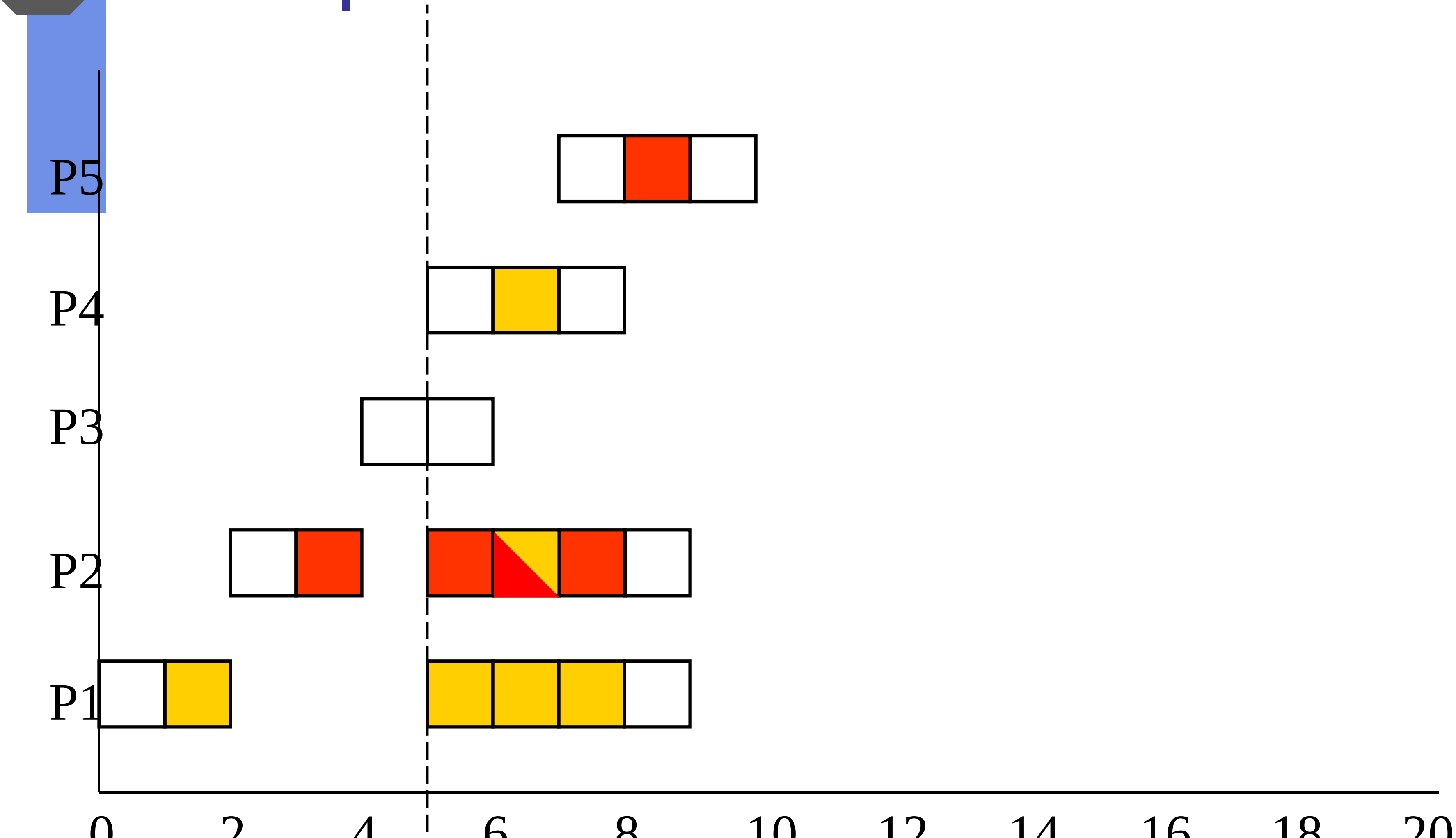
Example



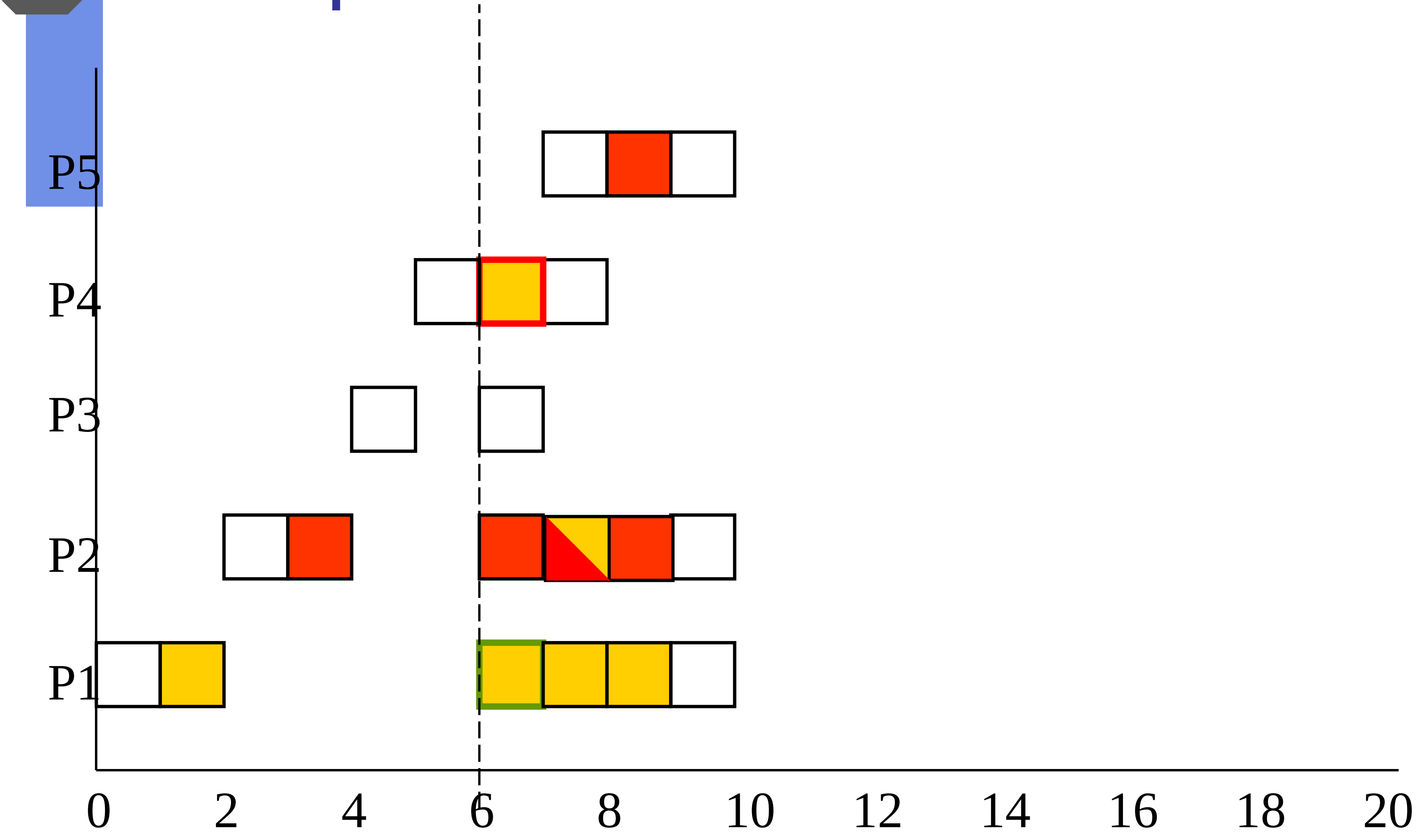
Example



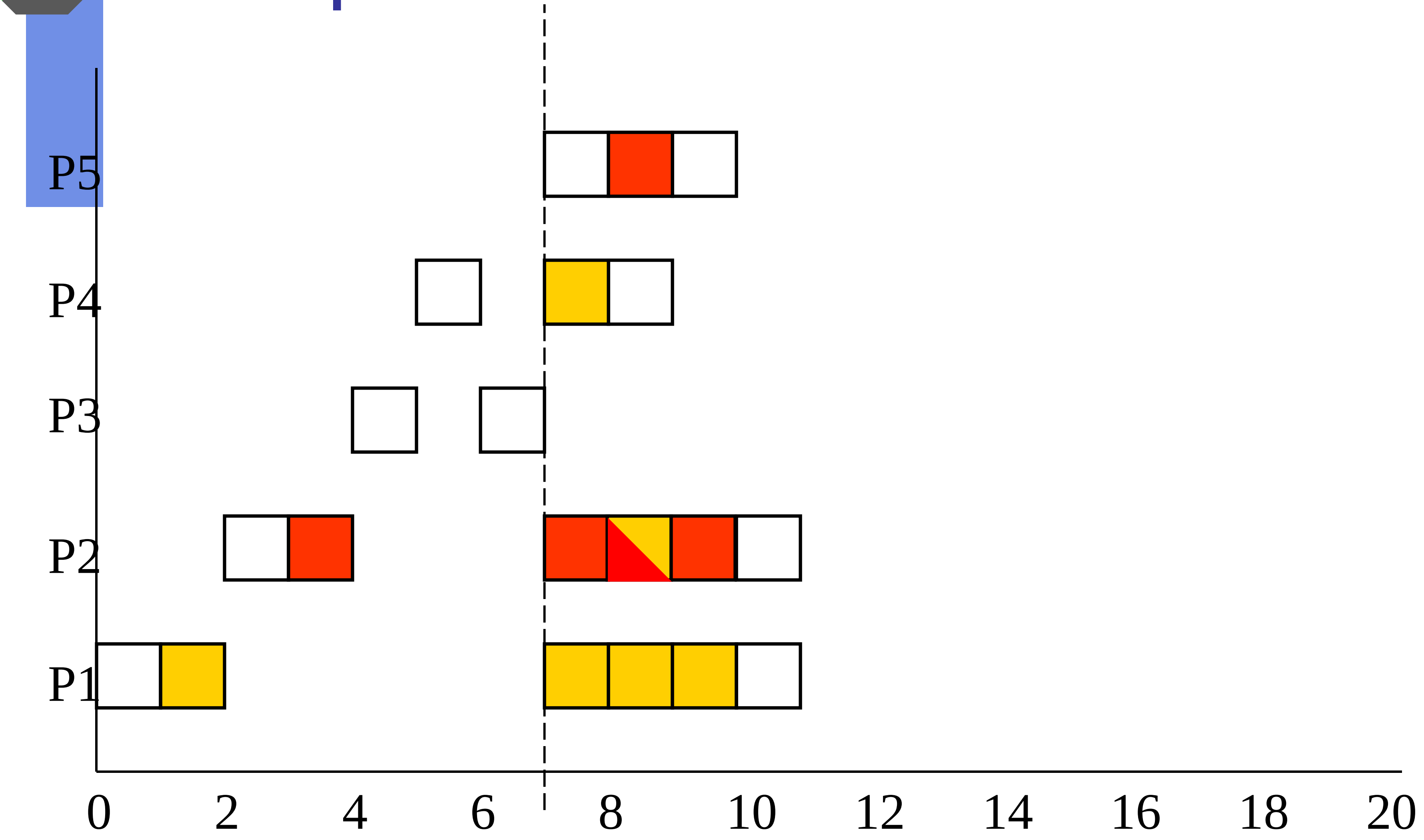
Example



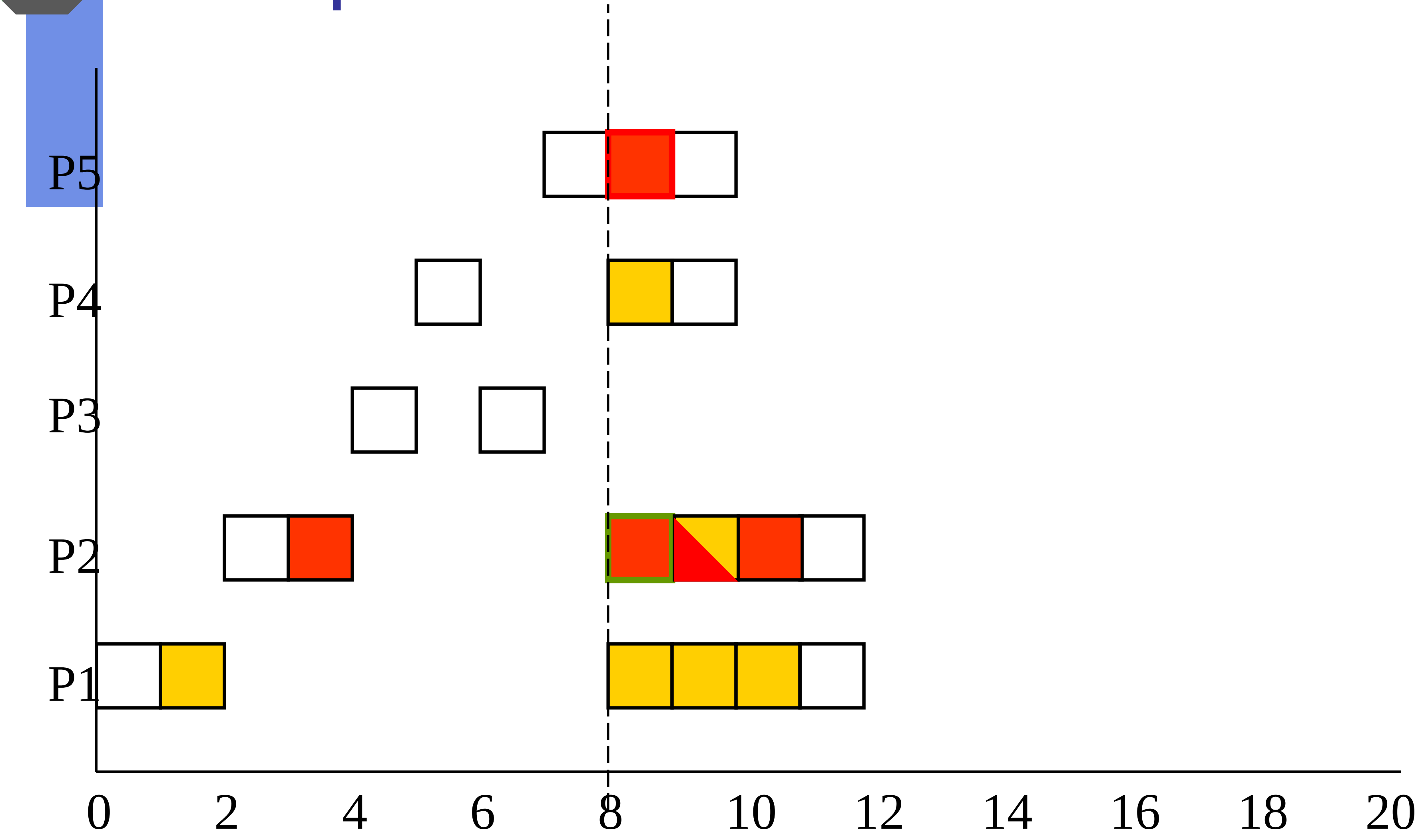
Example



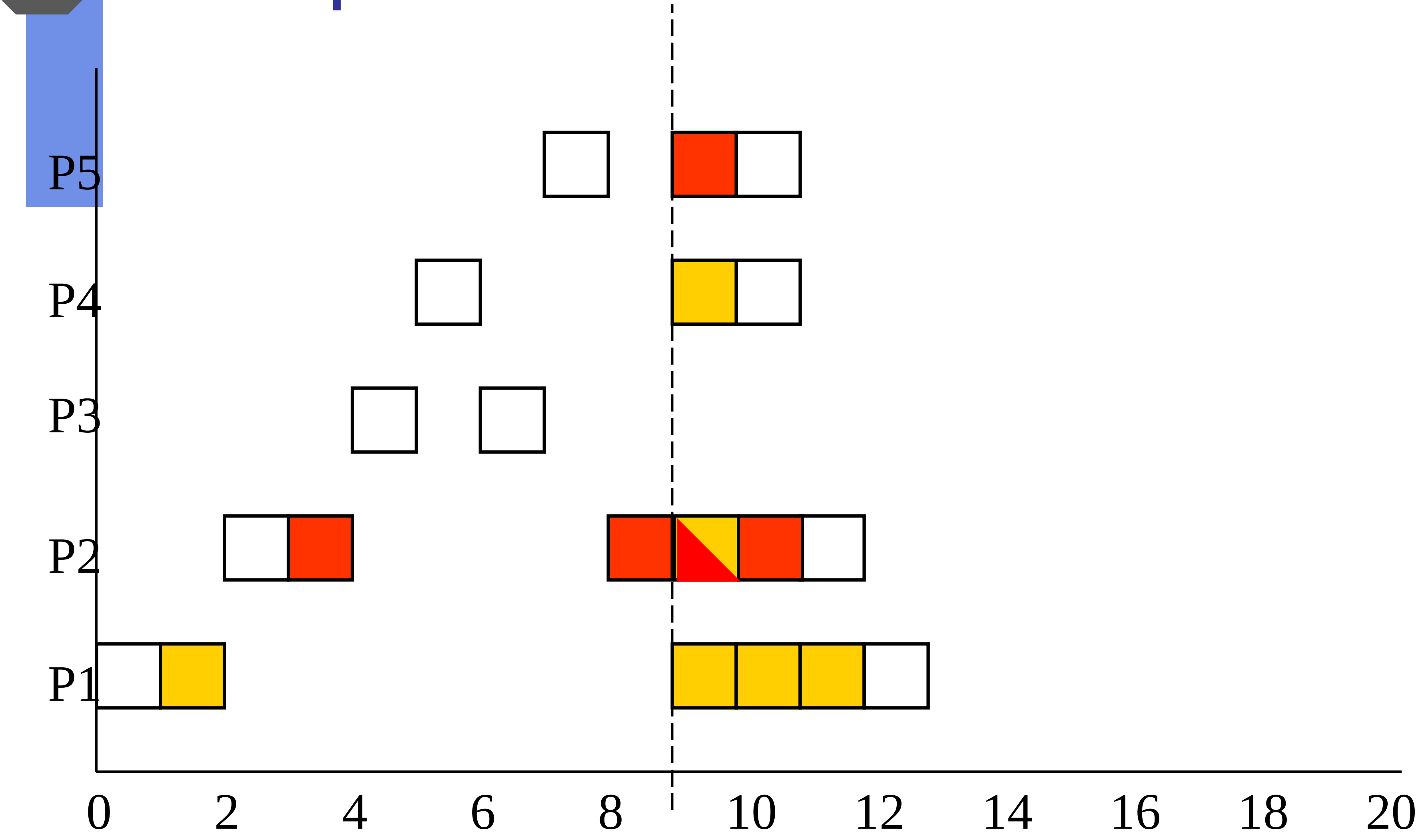
Example



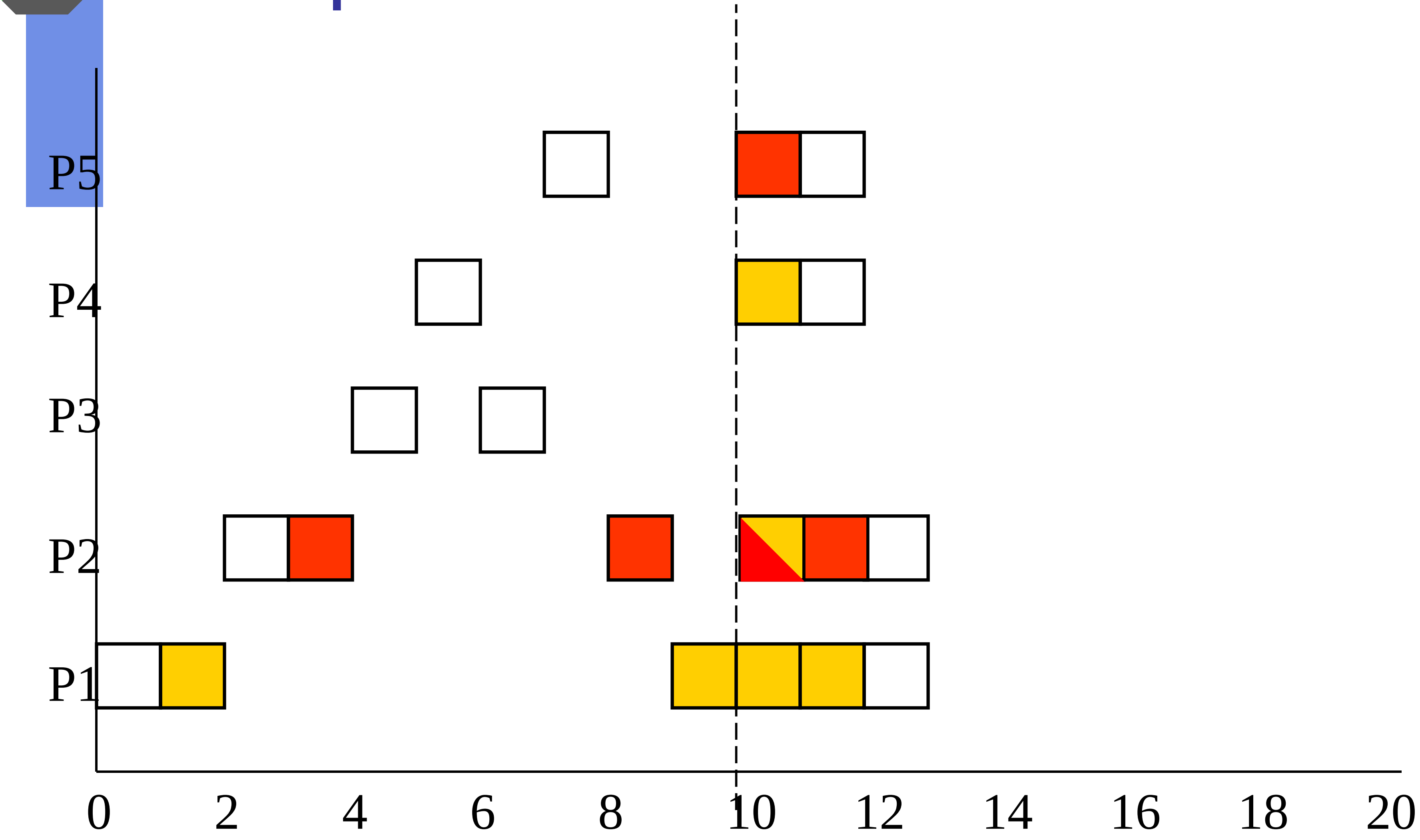
Example



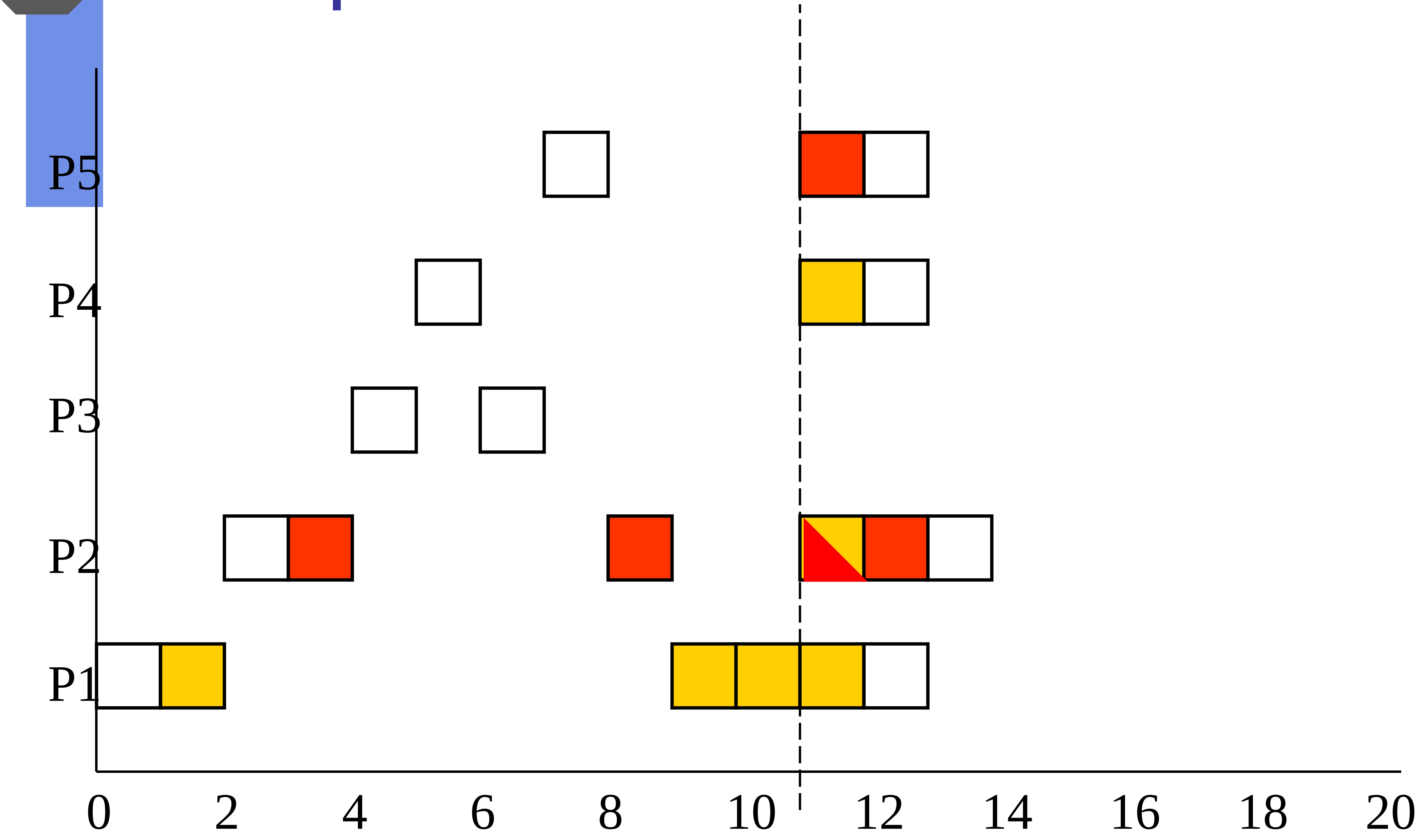
Example



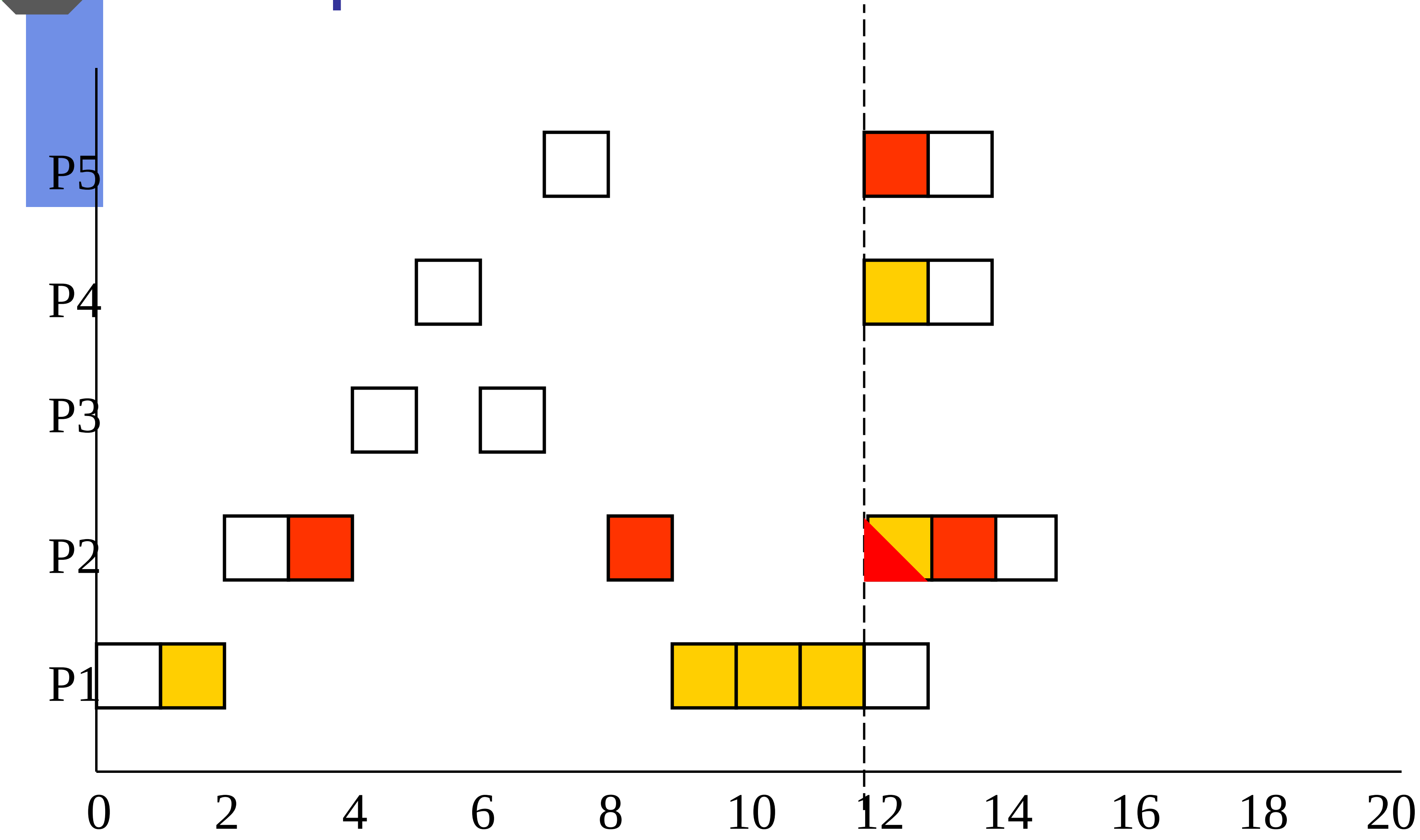
Example



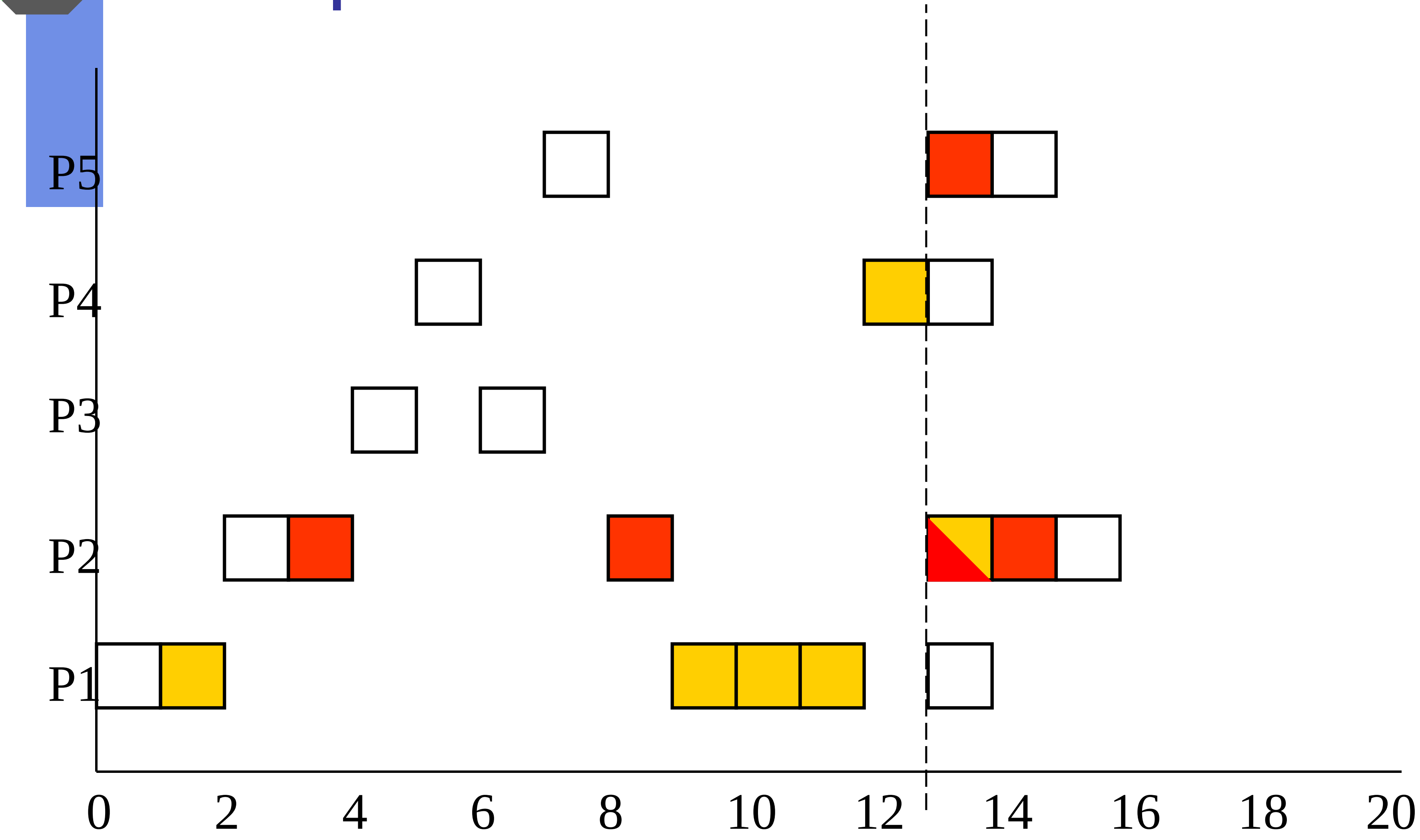
Example



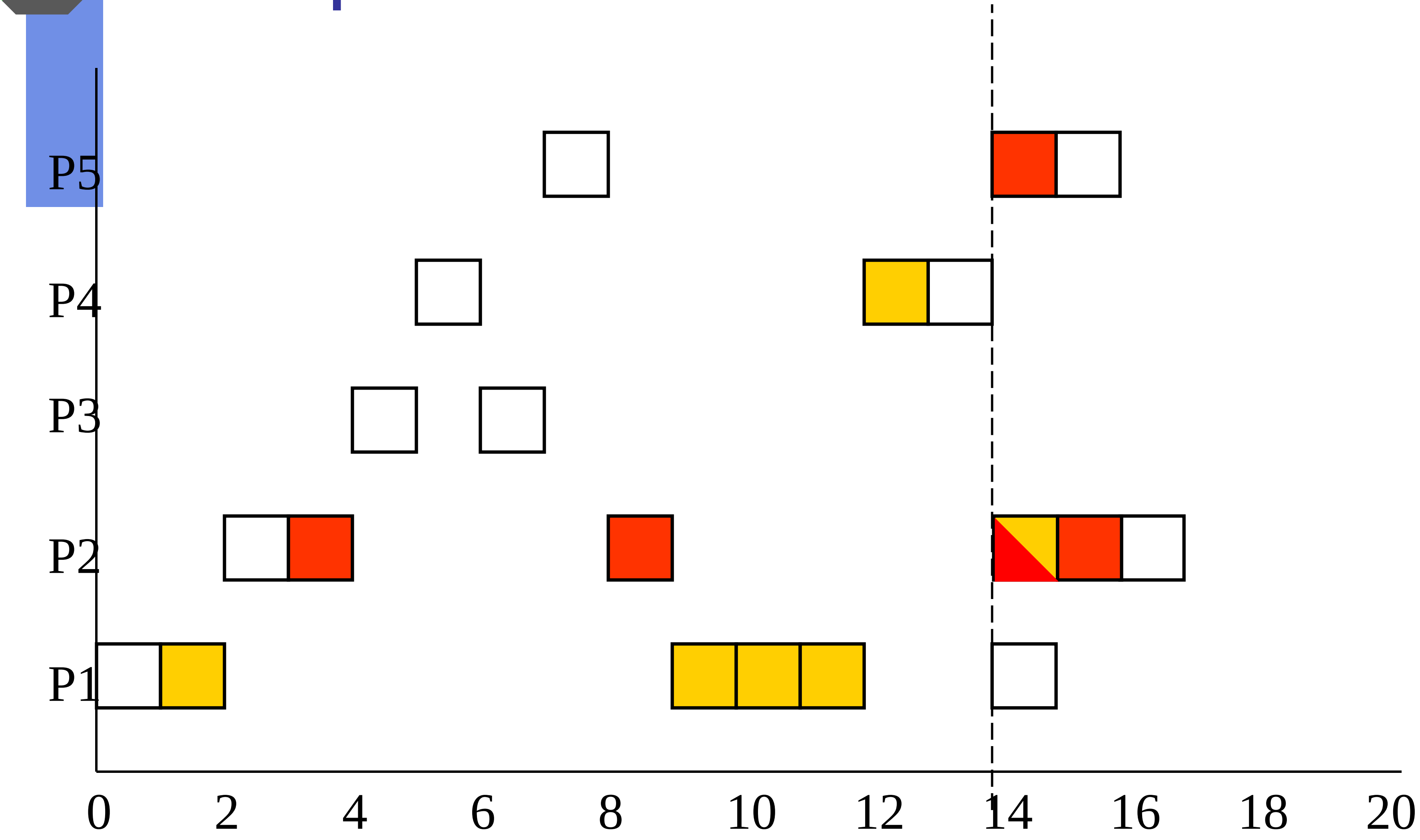
Example



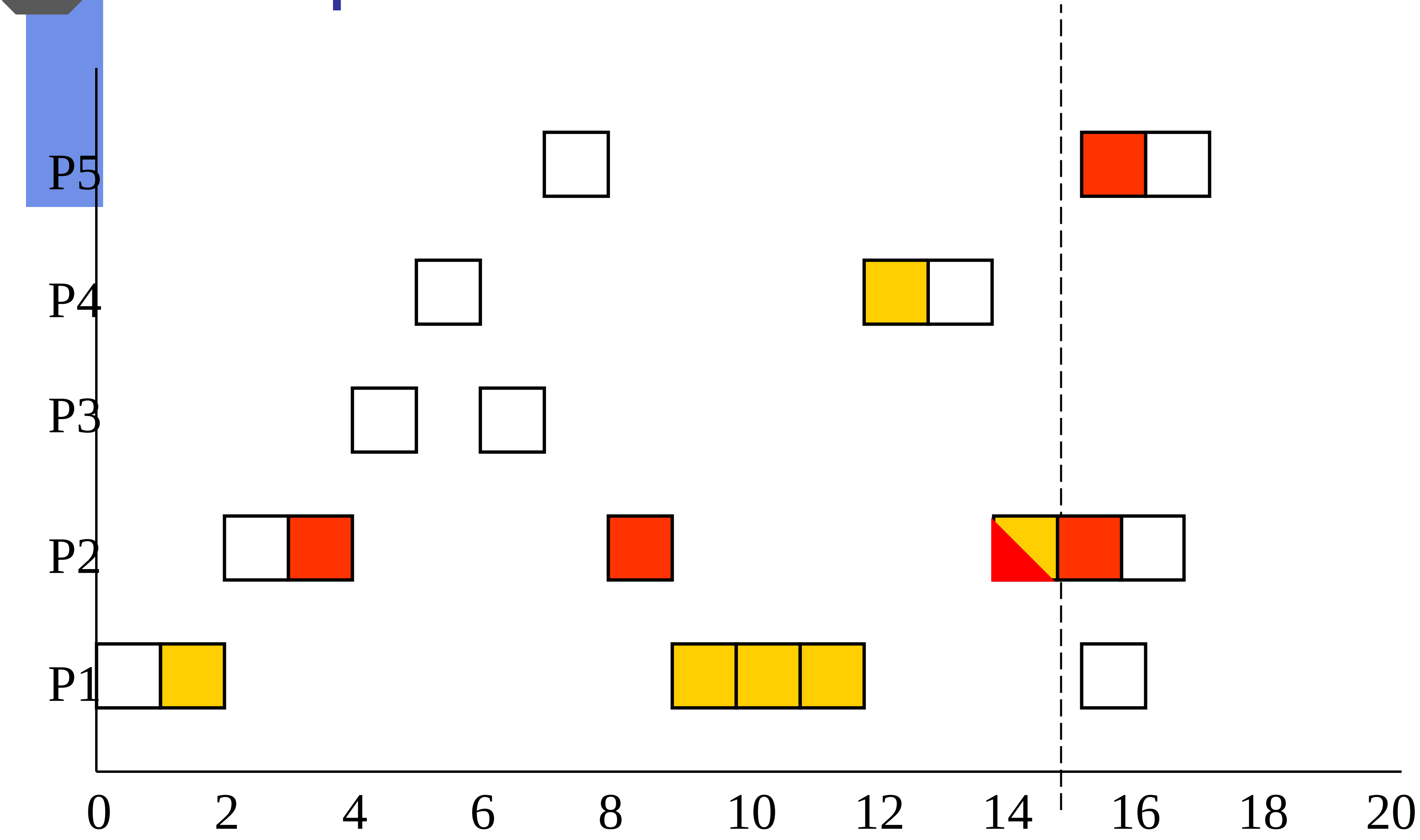
Example



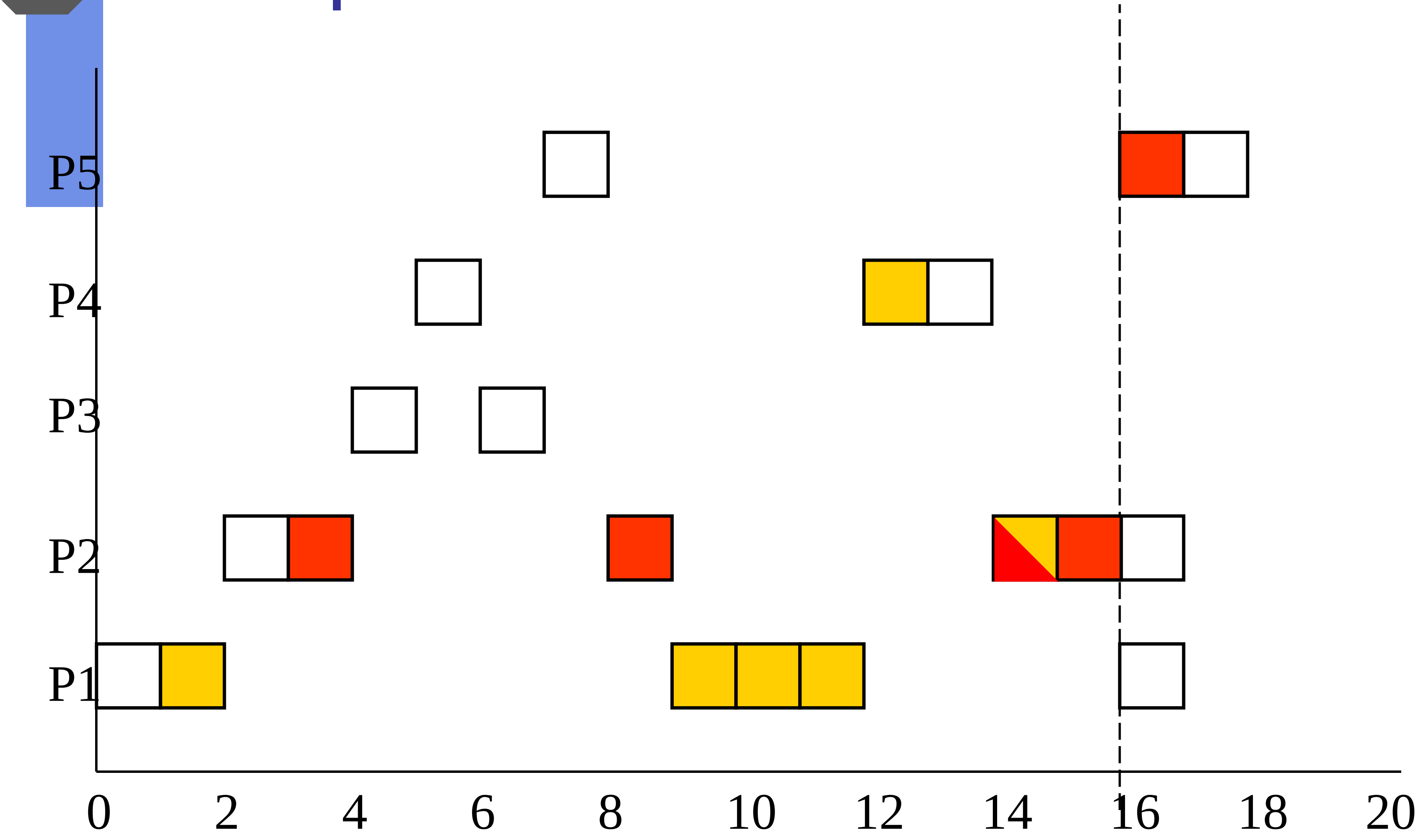
Example



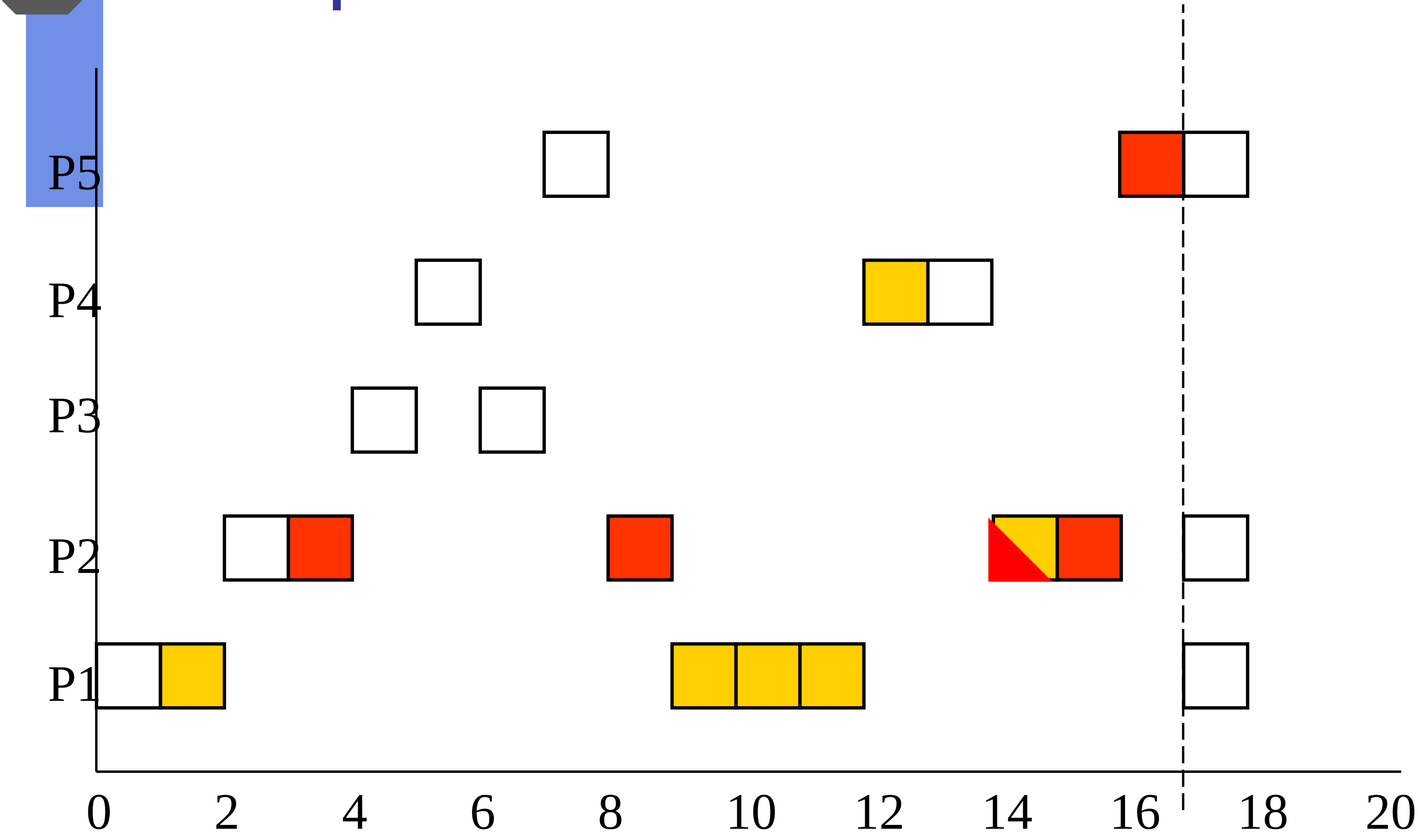
Example



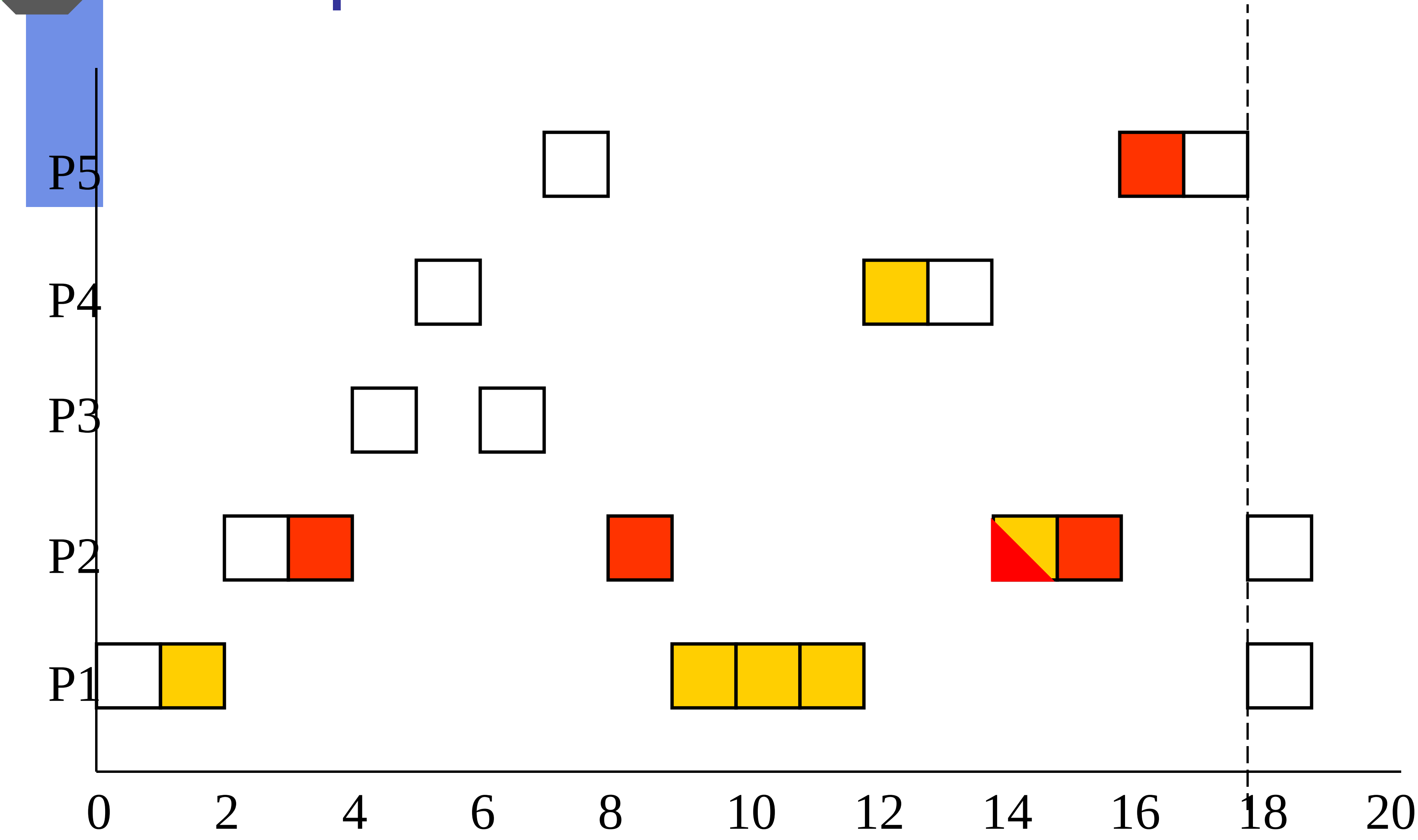
Example



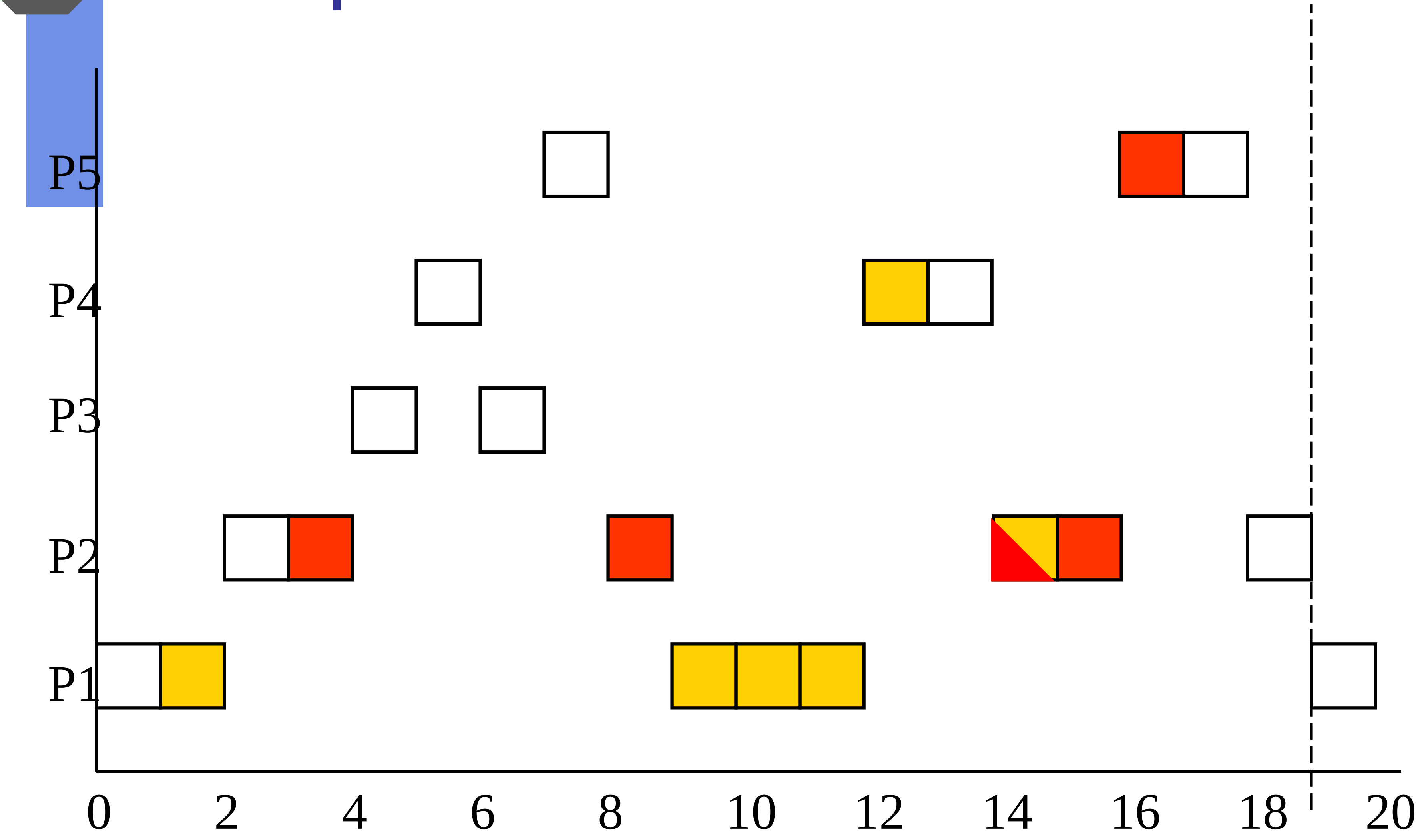
Example



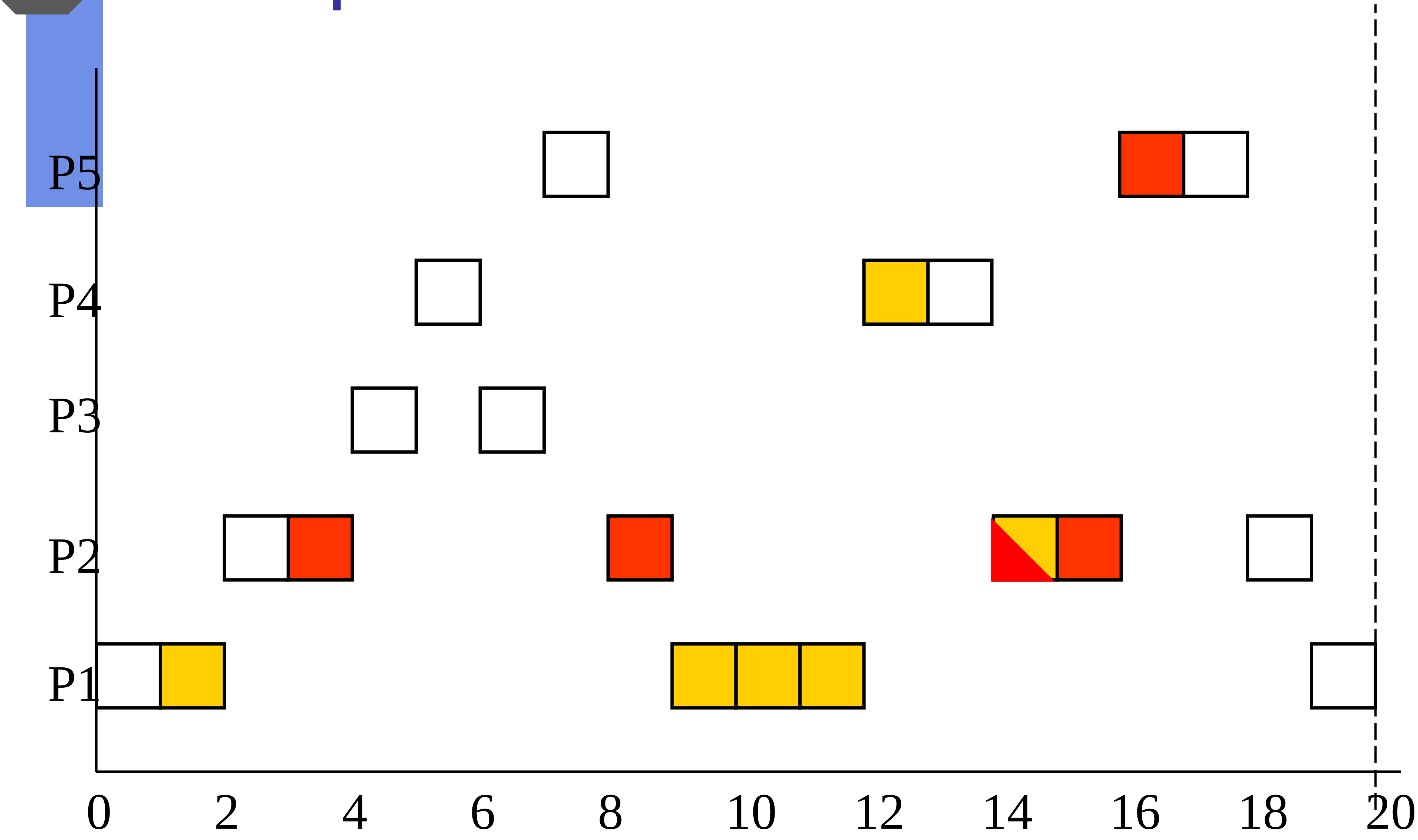
Example



Example



Example



Result

- High priority processes P5, P4 heavily delayed
- P3 is almost not delayed due to its characteristic, it does not need any resource

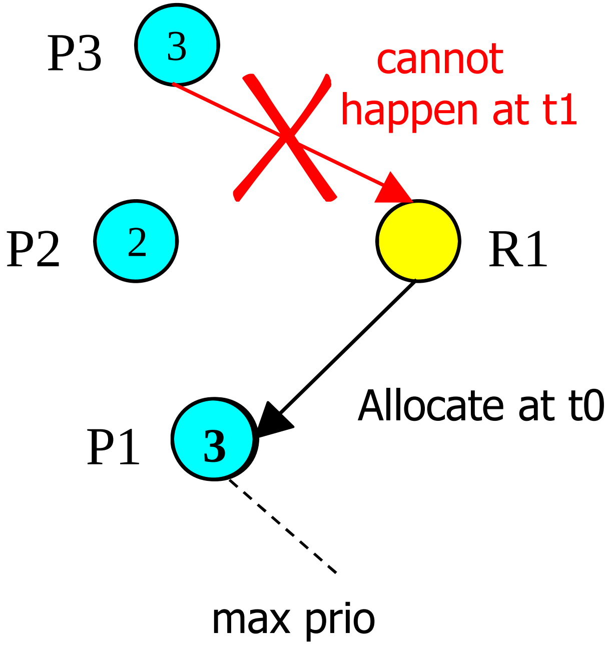
⇒ Find a better solution

4 Resource Allocation Protocols

- Non Preemptive Critical Sections (*NPCS)*
- Priority Inheritance (*PI*)
- Priority-Ceiling Protocol *(PCP)*
- Stacked Priority-Ceiling Protocol *(SPCP)*
- … and some others
  - See text book (Liu)

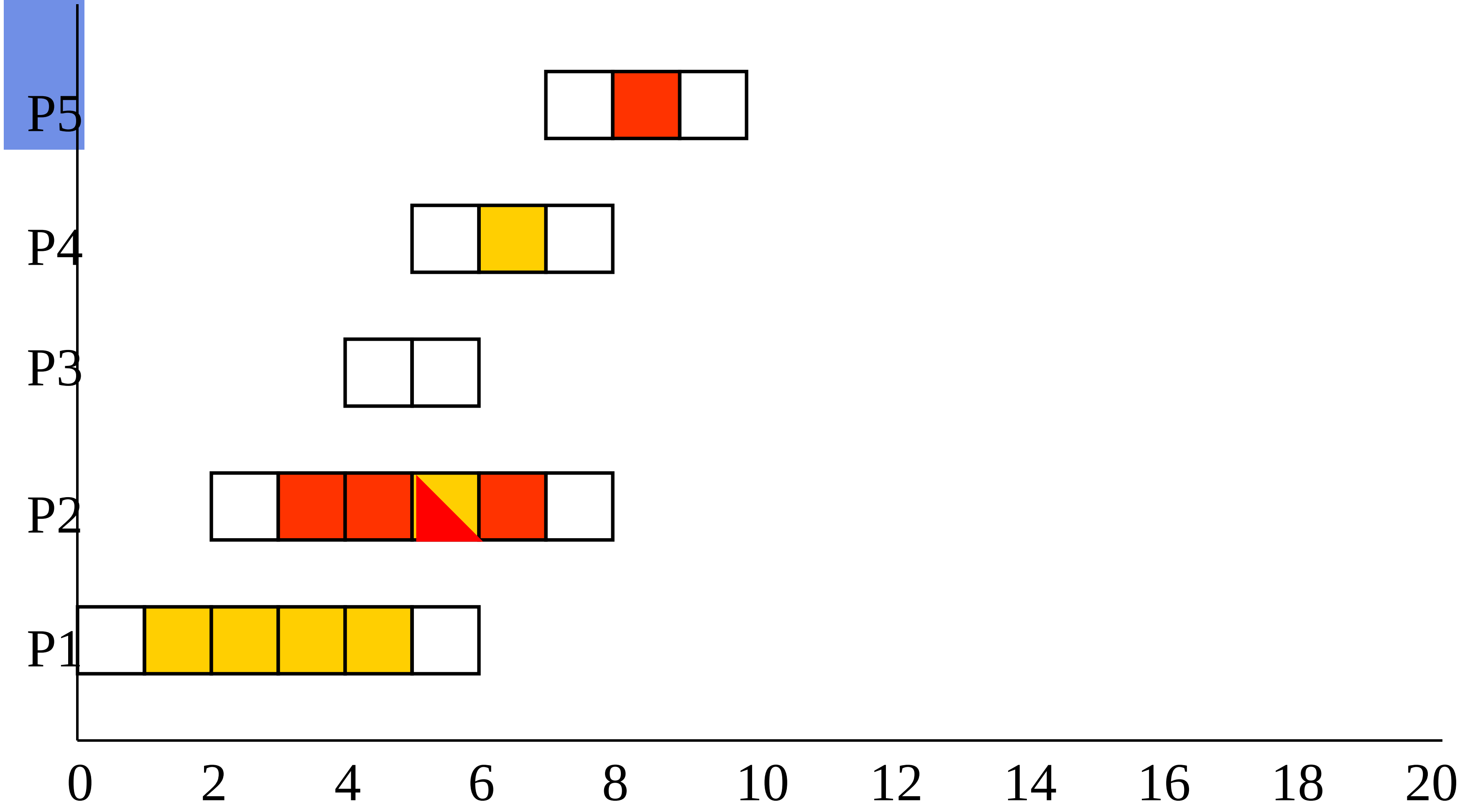
Nonpreemptive Critical Sections

- As soon as a process holds a resource it is *no longer preemptable**
- *Prevents deadlock*
- *Bounds priority inversion*
  - Max blocking time is the *maximum execution time* of the critical sections of all lower priority processes

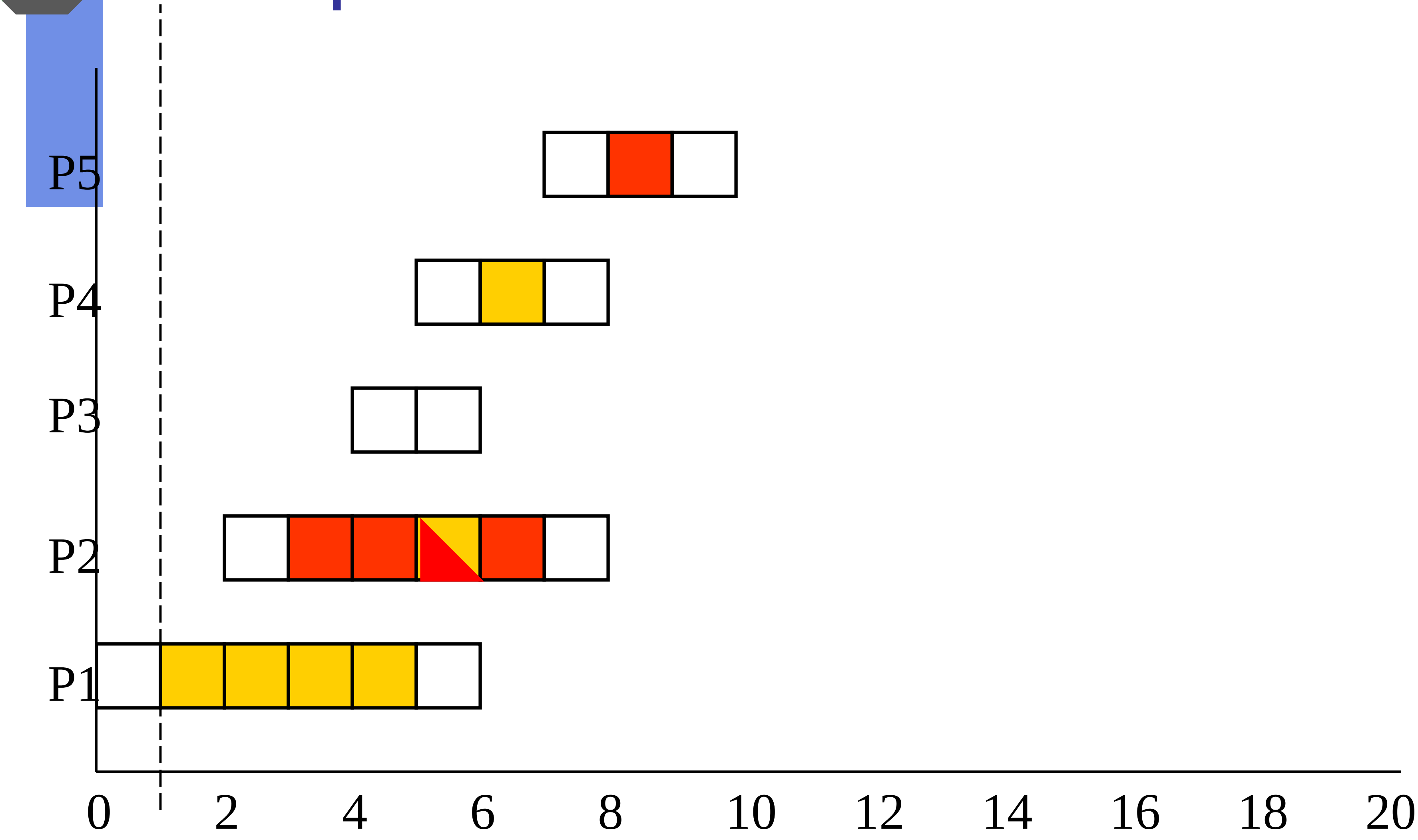


*This process gets ***highest priority*** in system

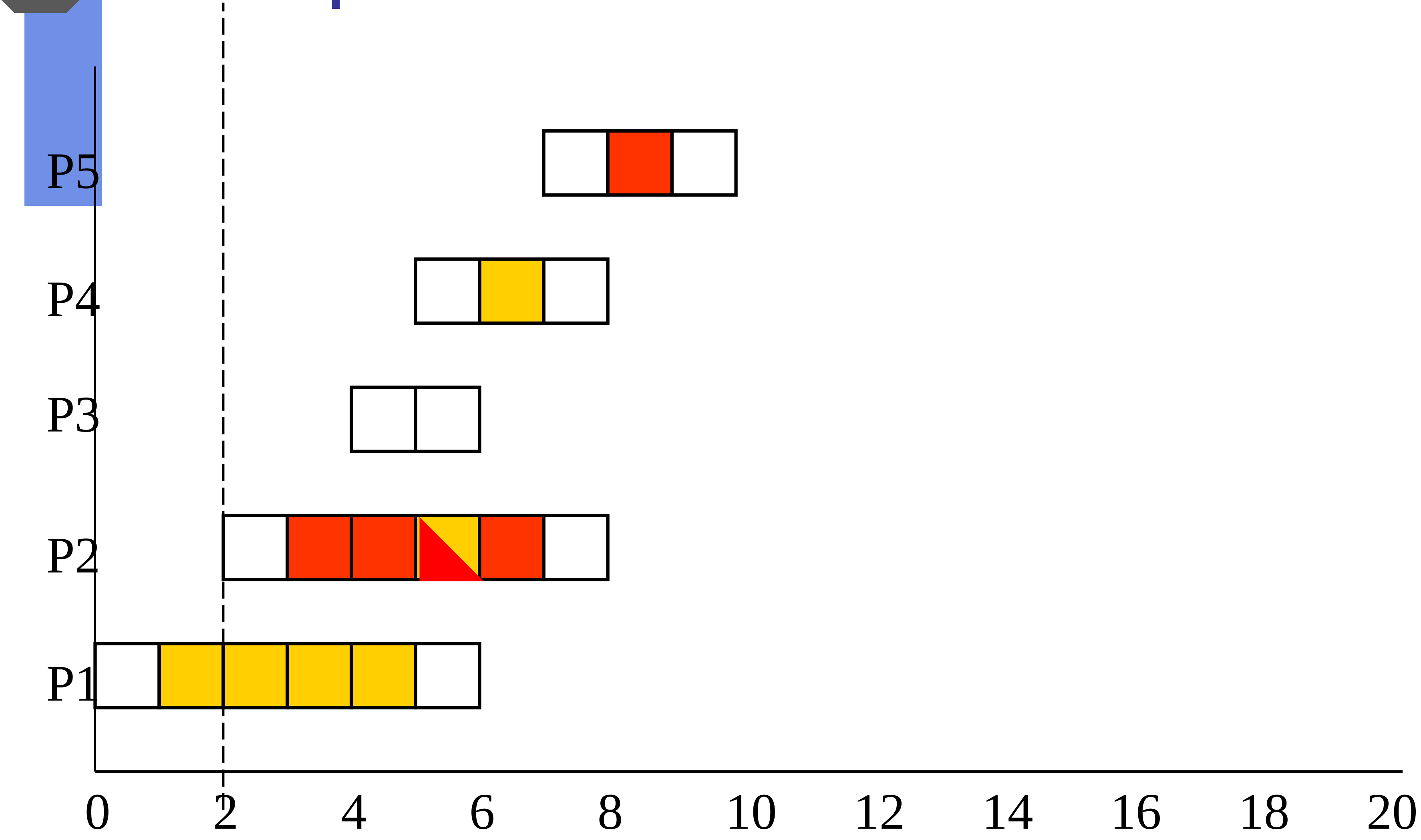
Non-Preemptive Critical Sections



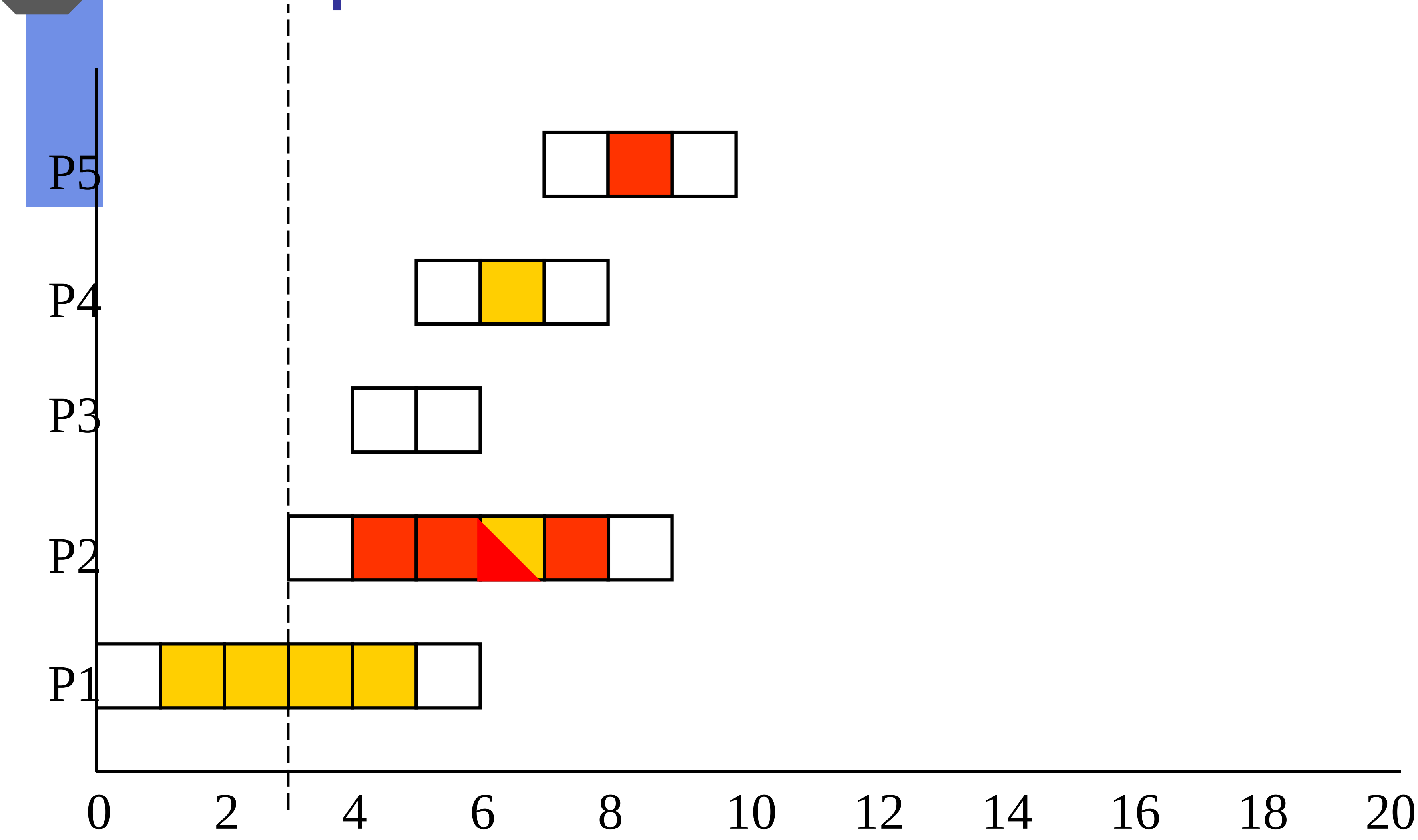
Example



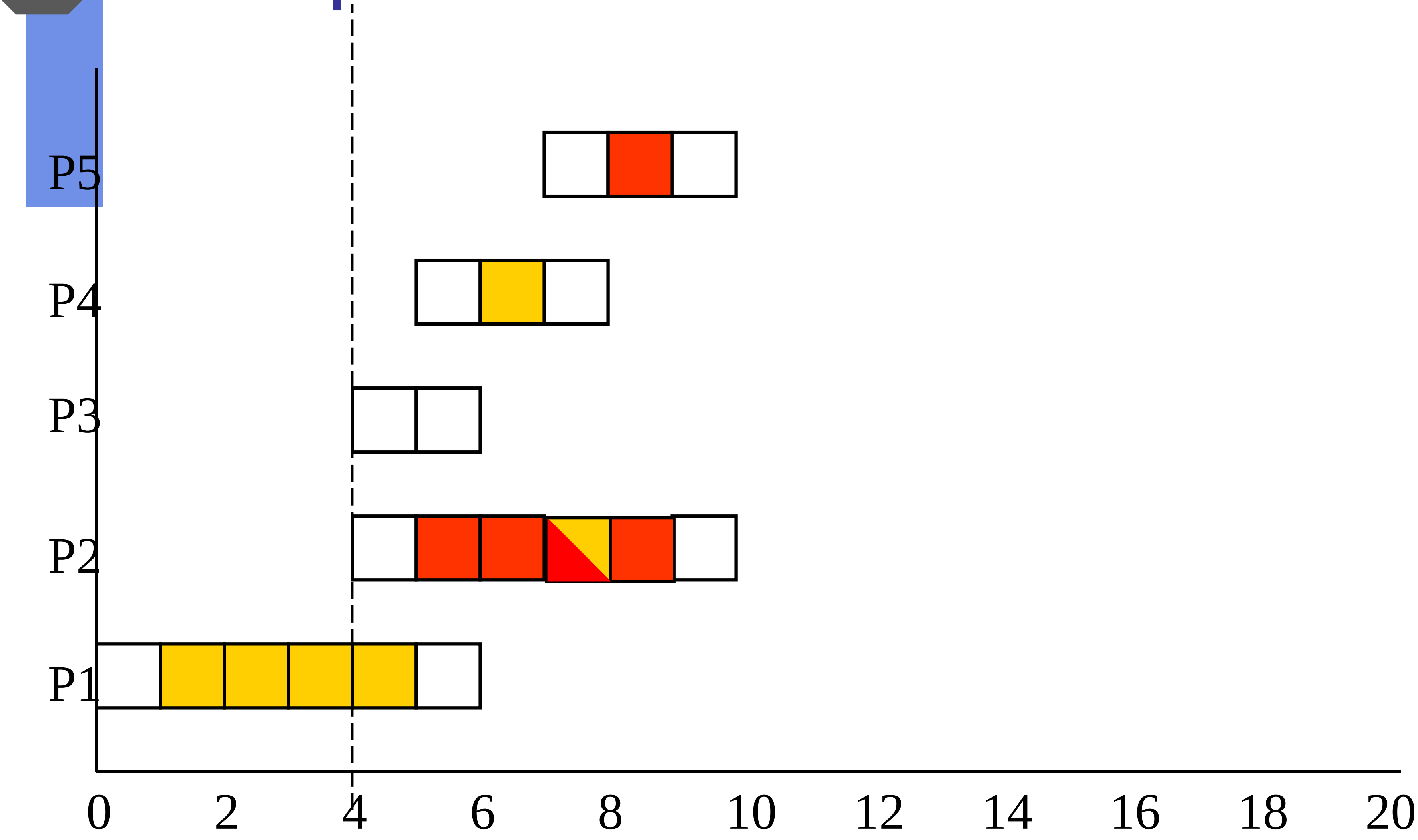
Example



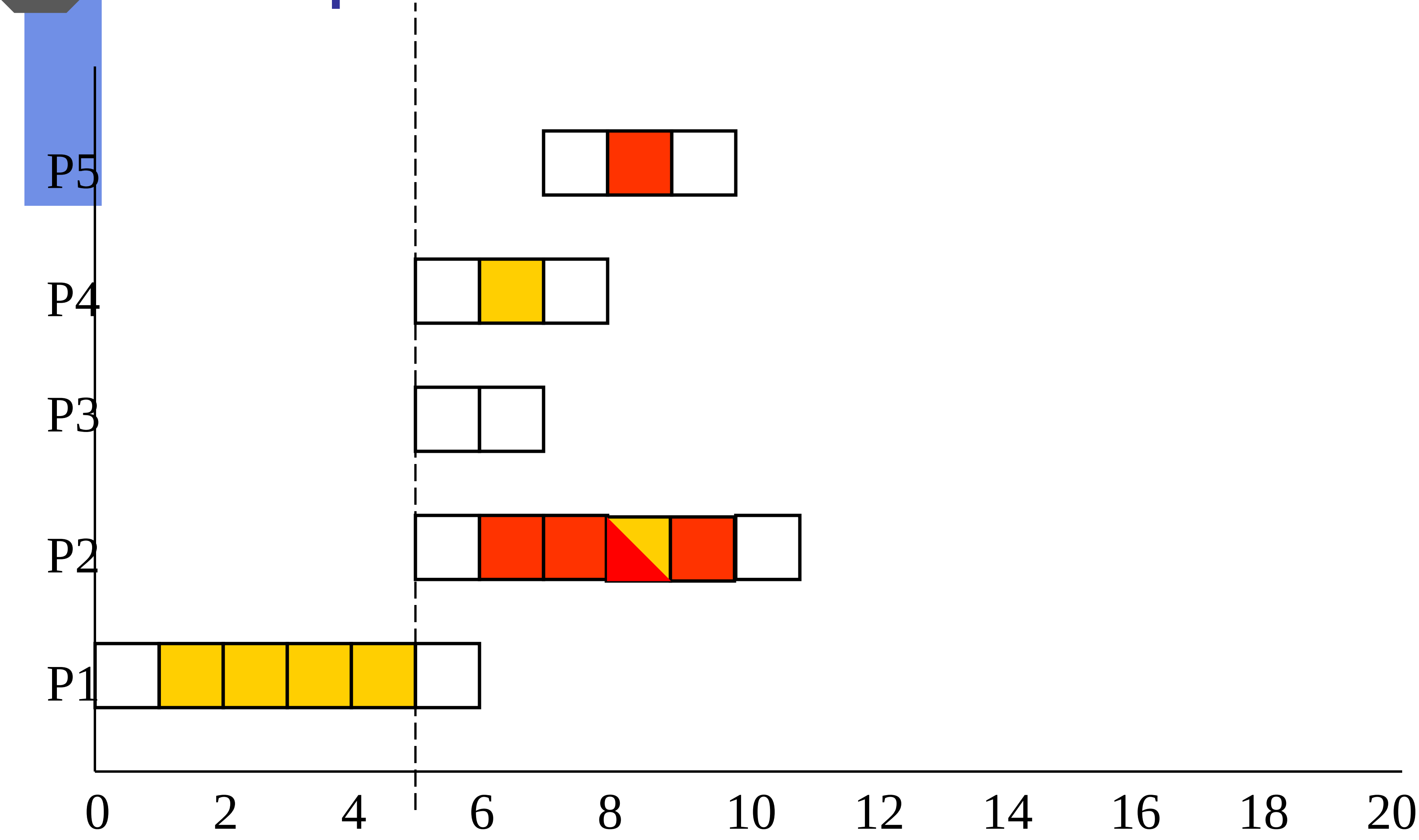
Example



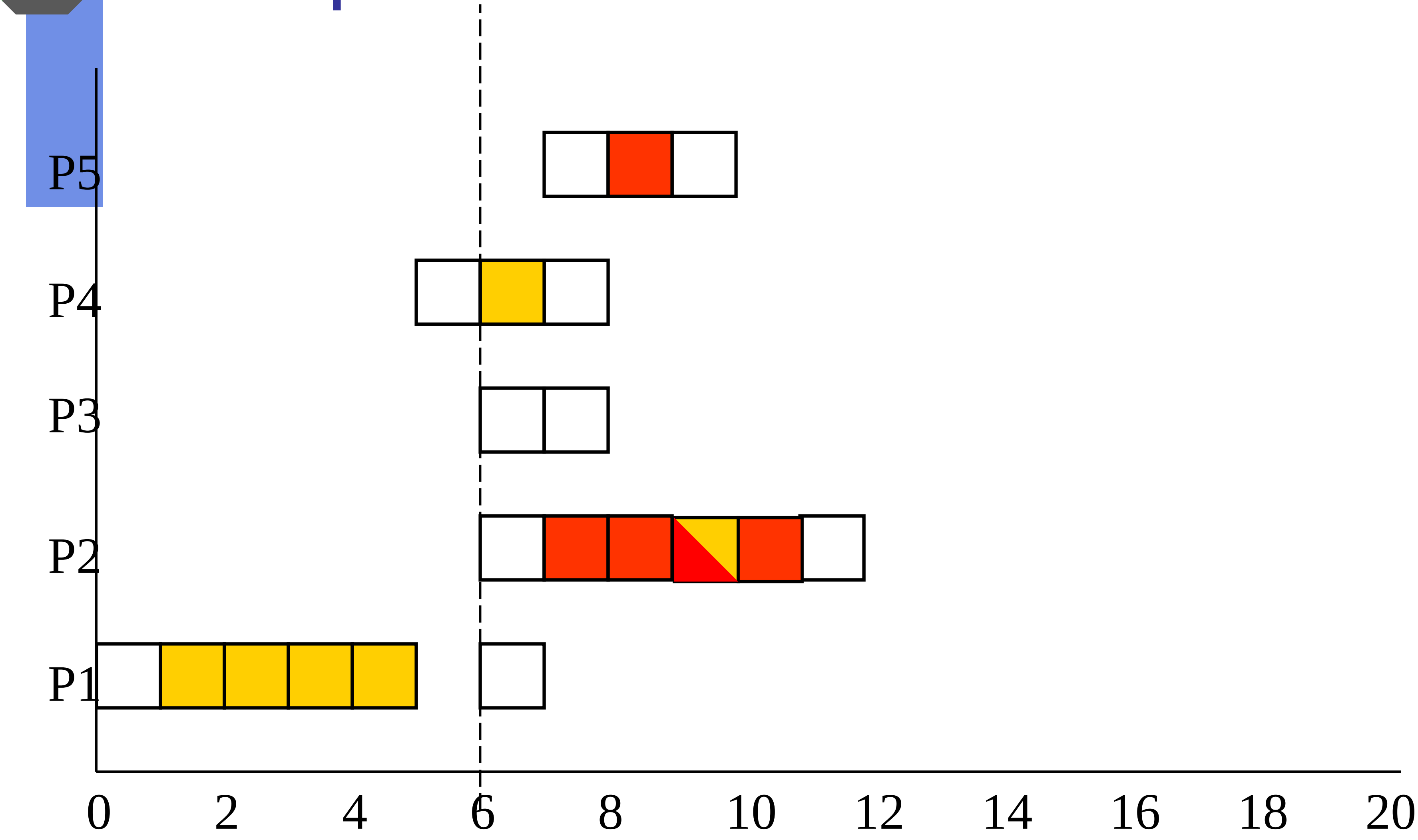
Example



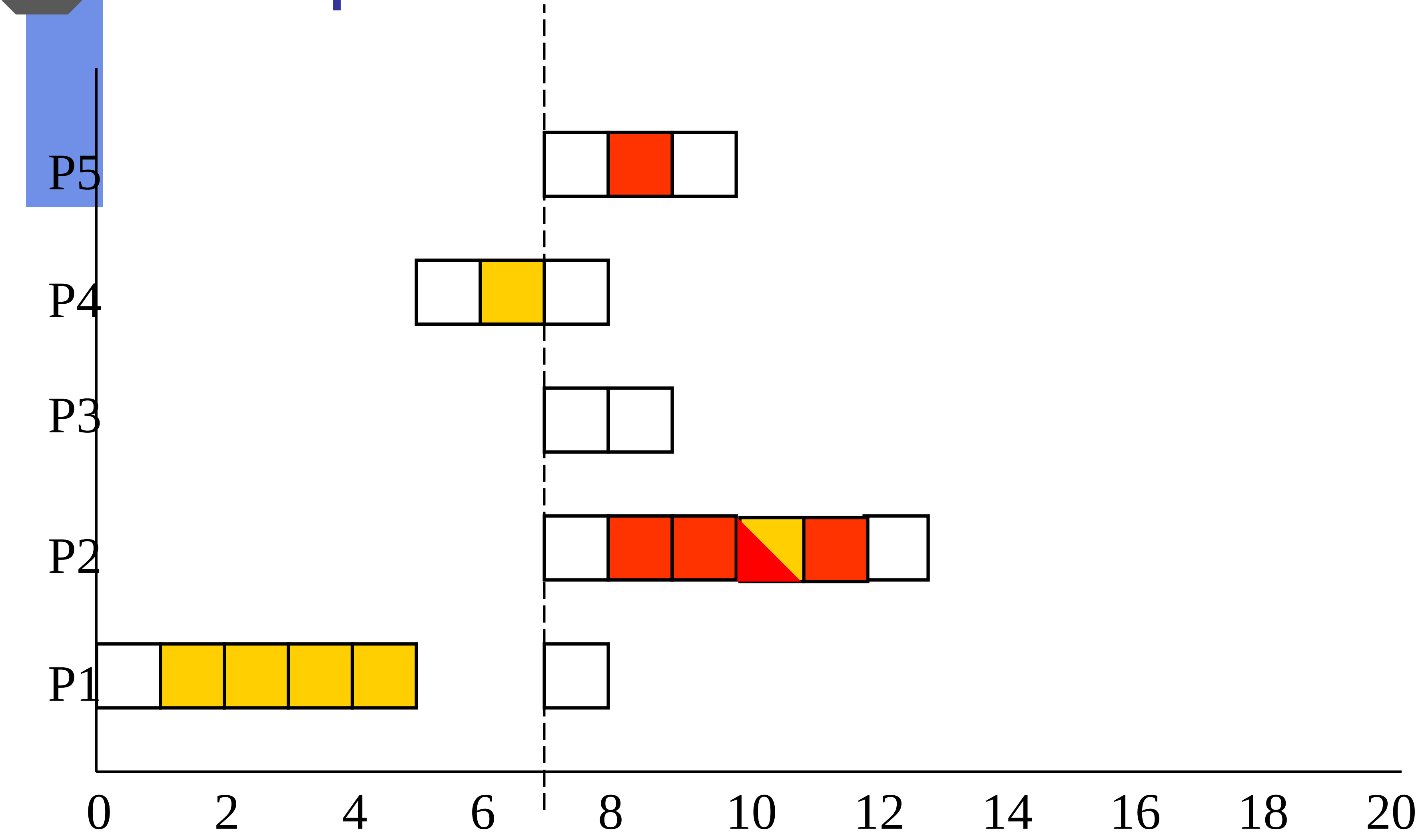
Example



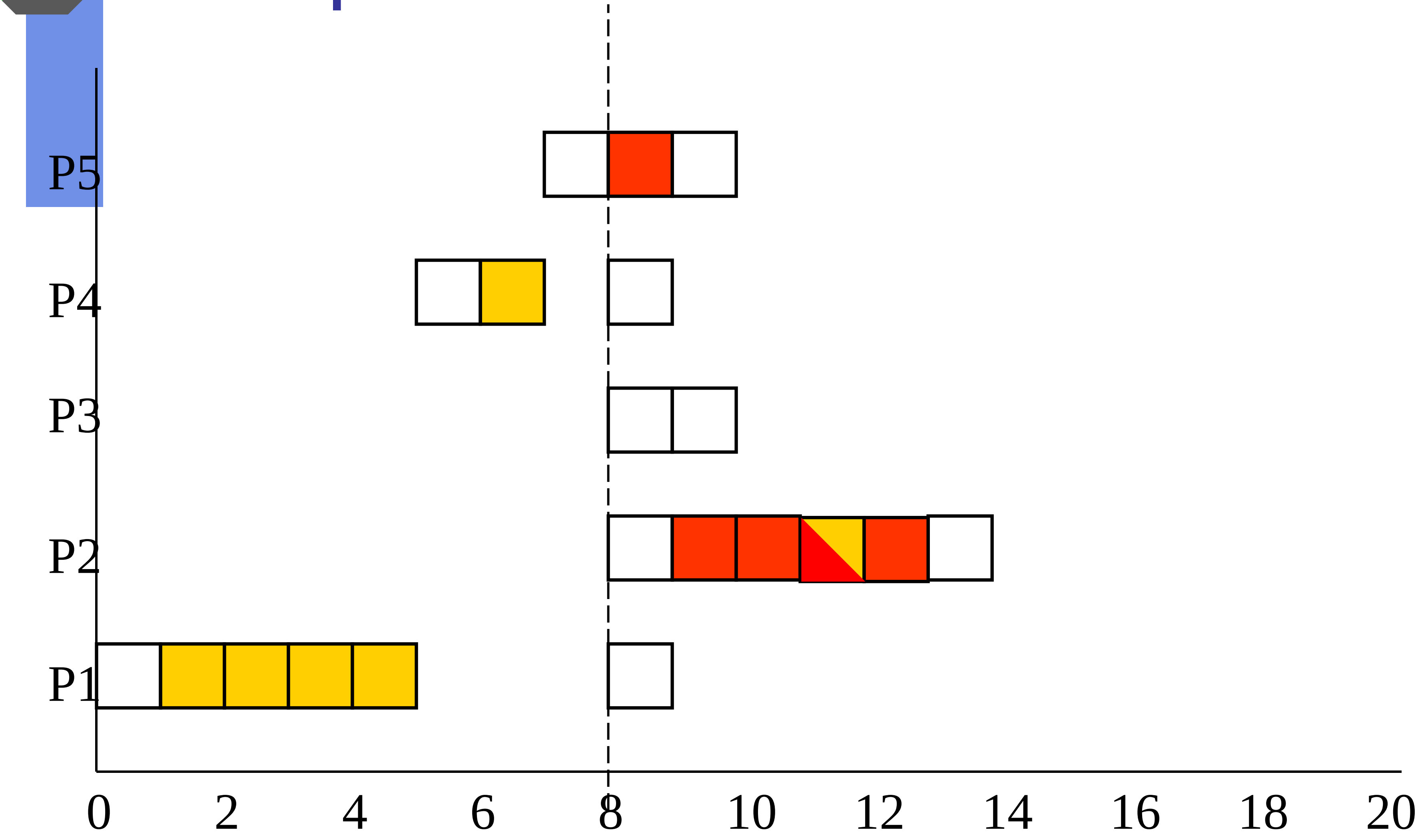
Example



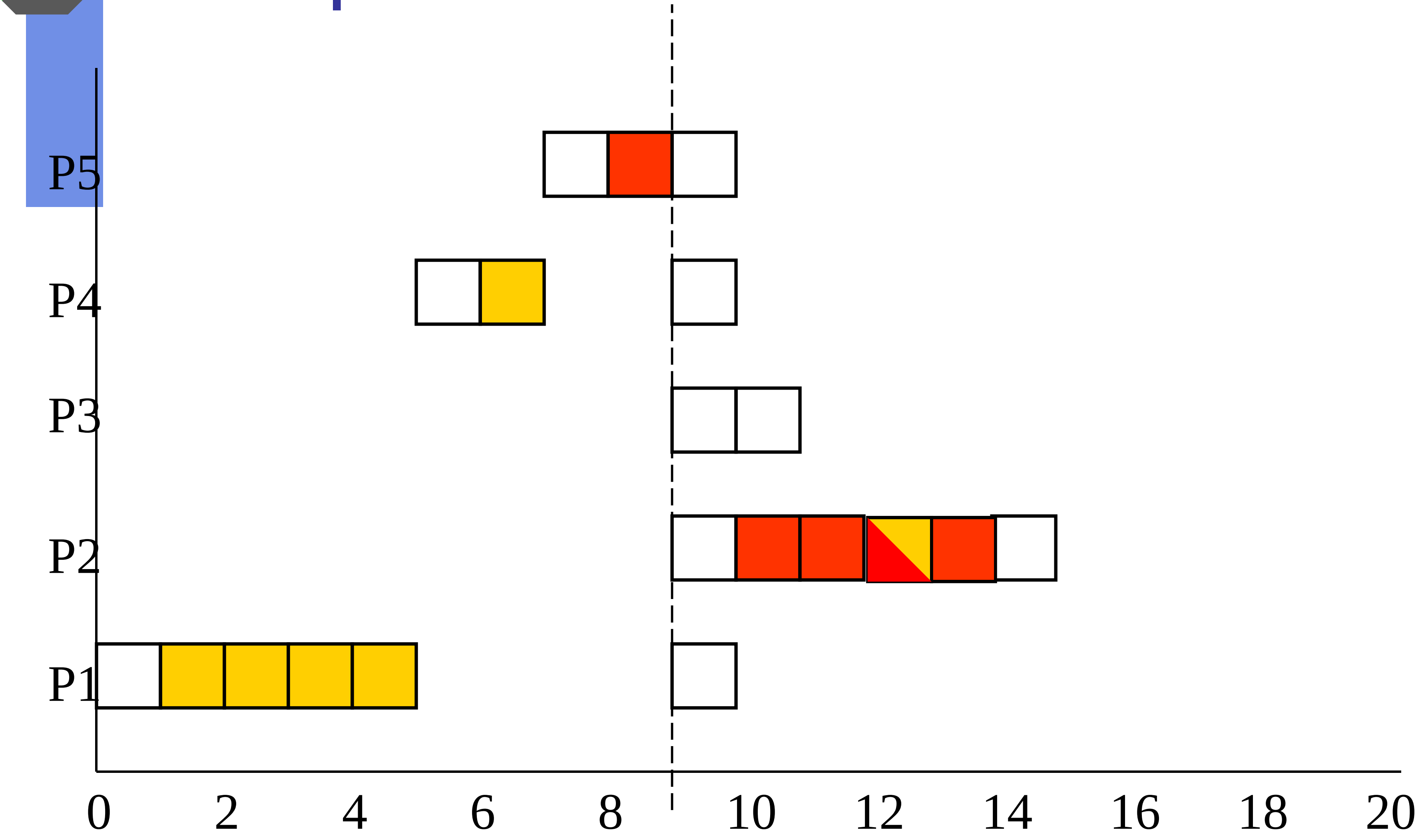
Example



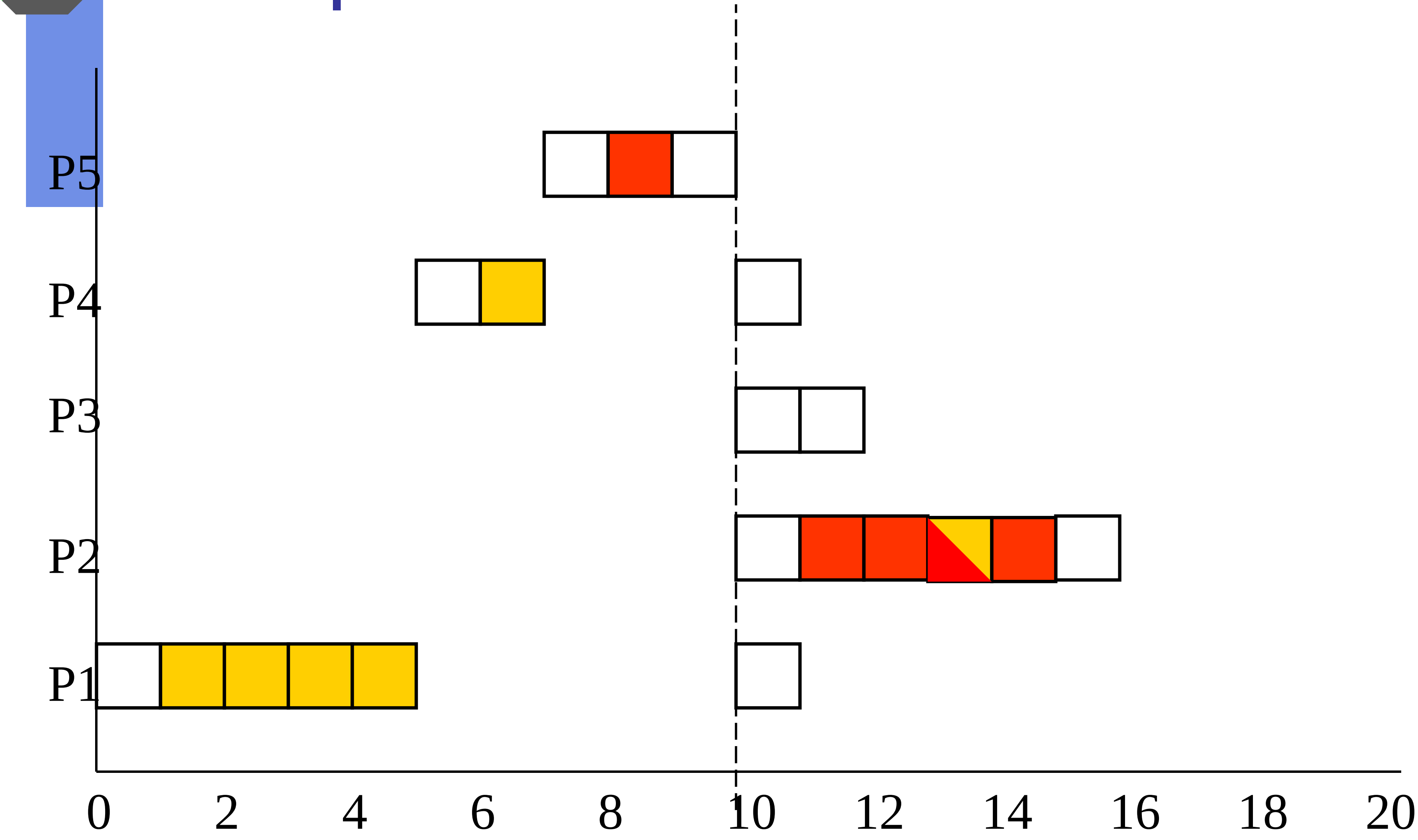
Example



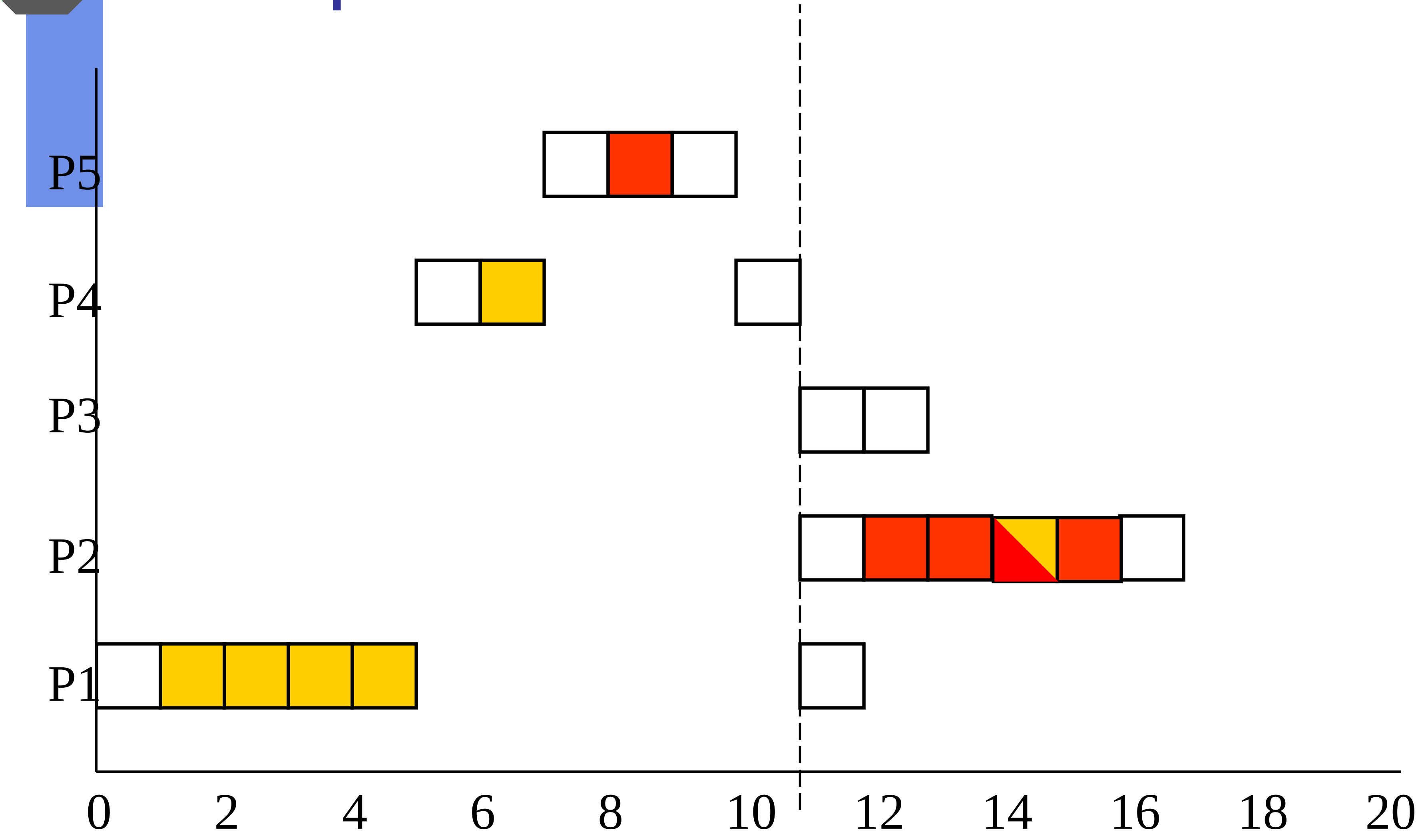
Example



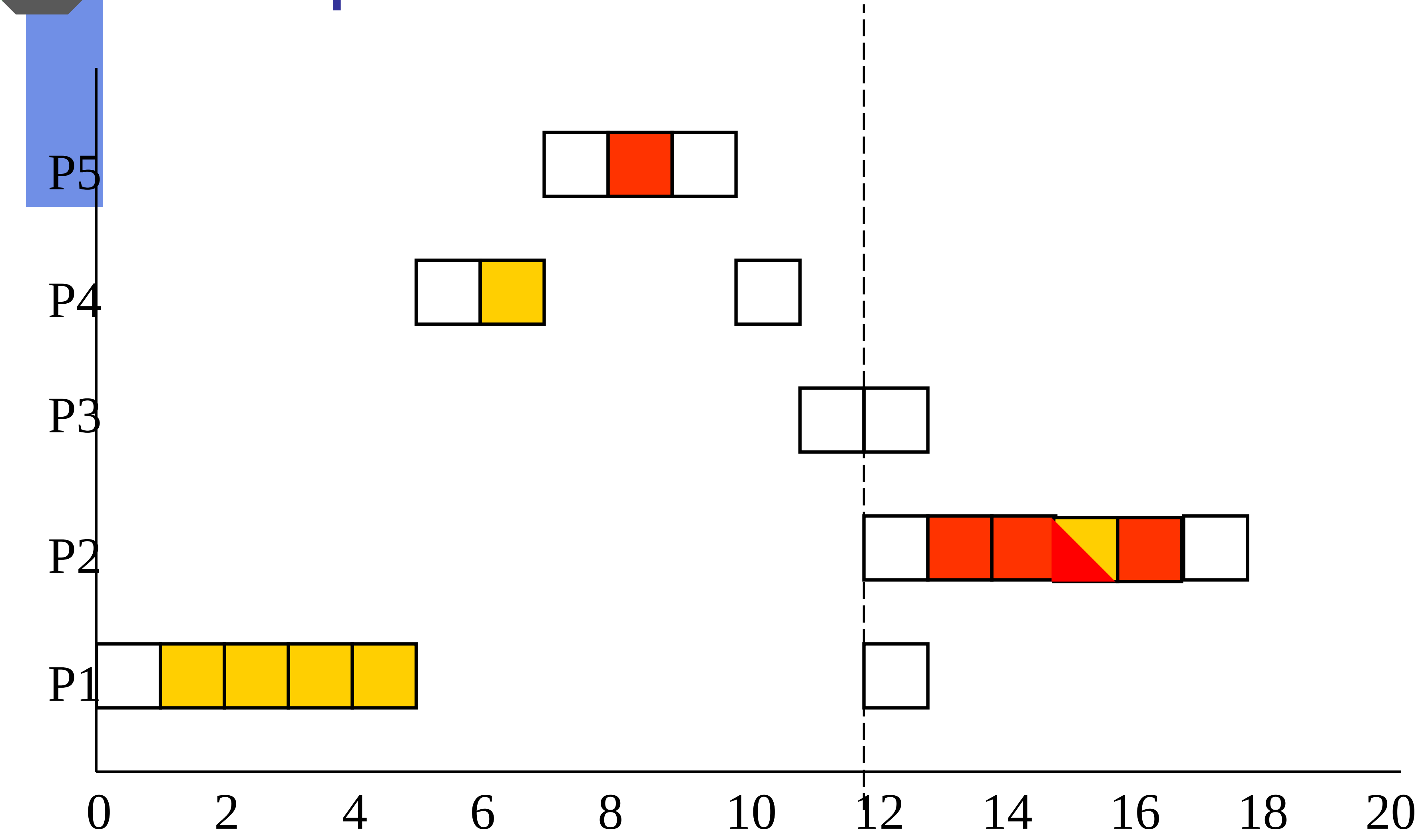
Example



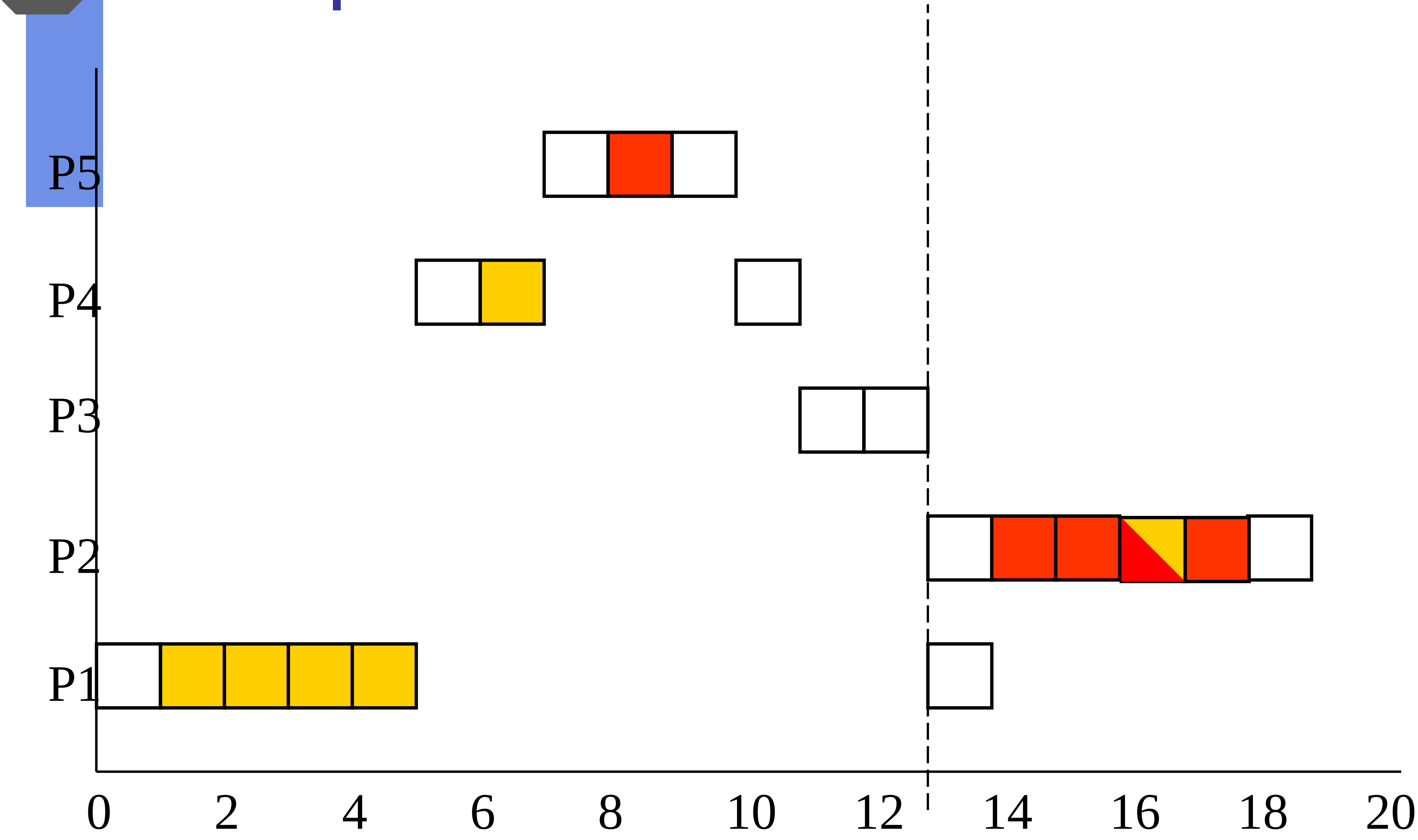
Example



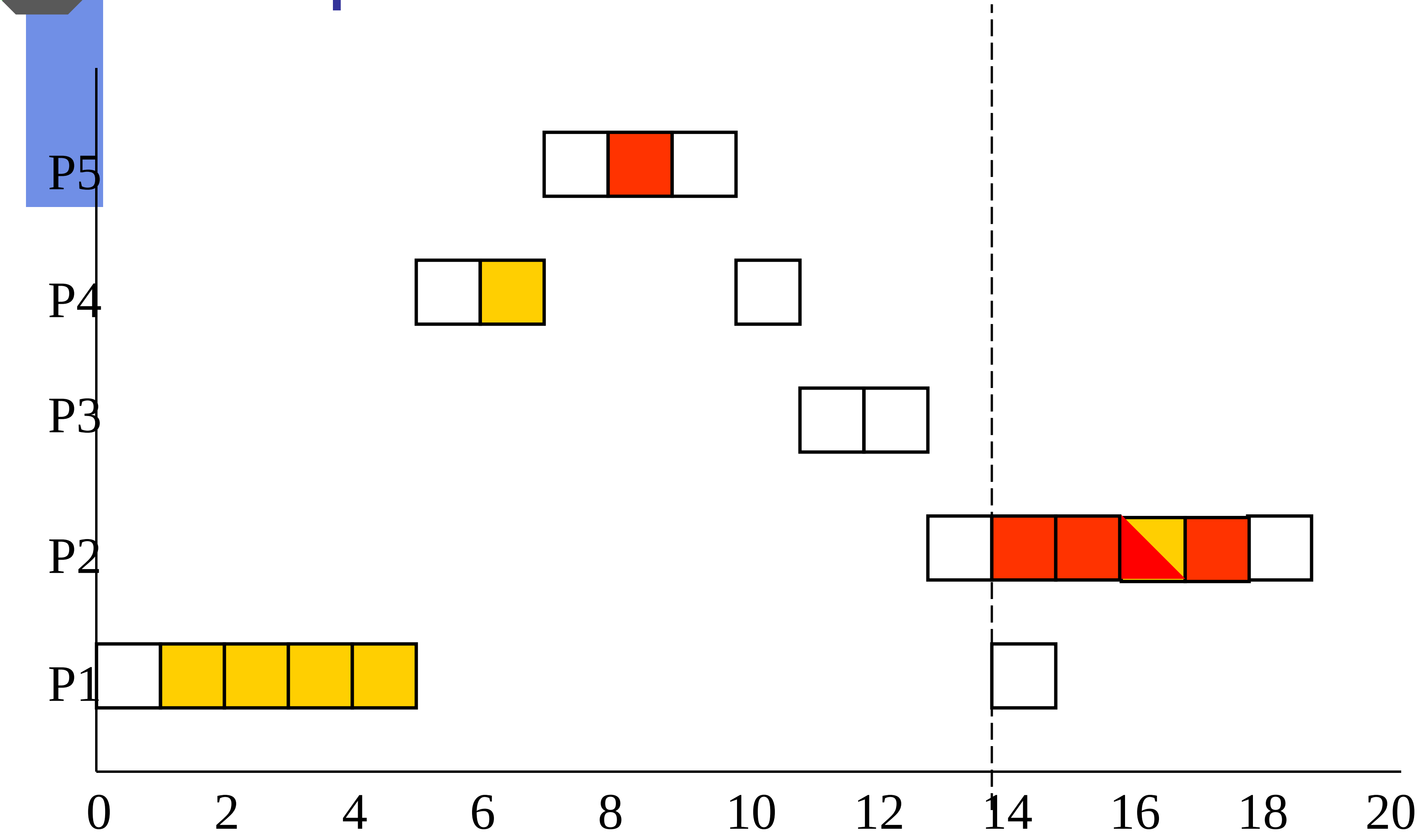
Example



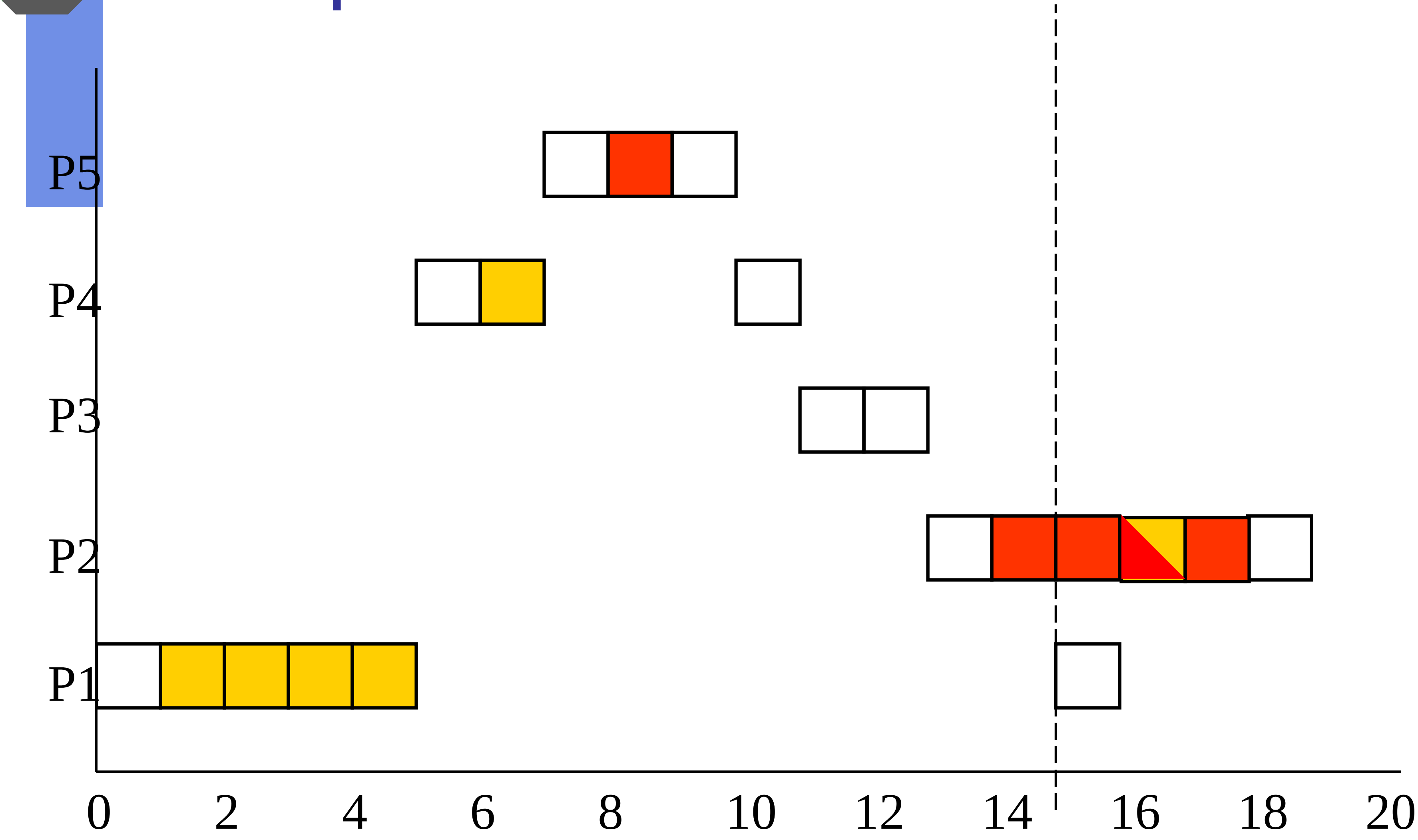
Example



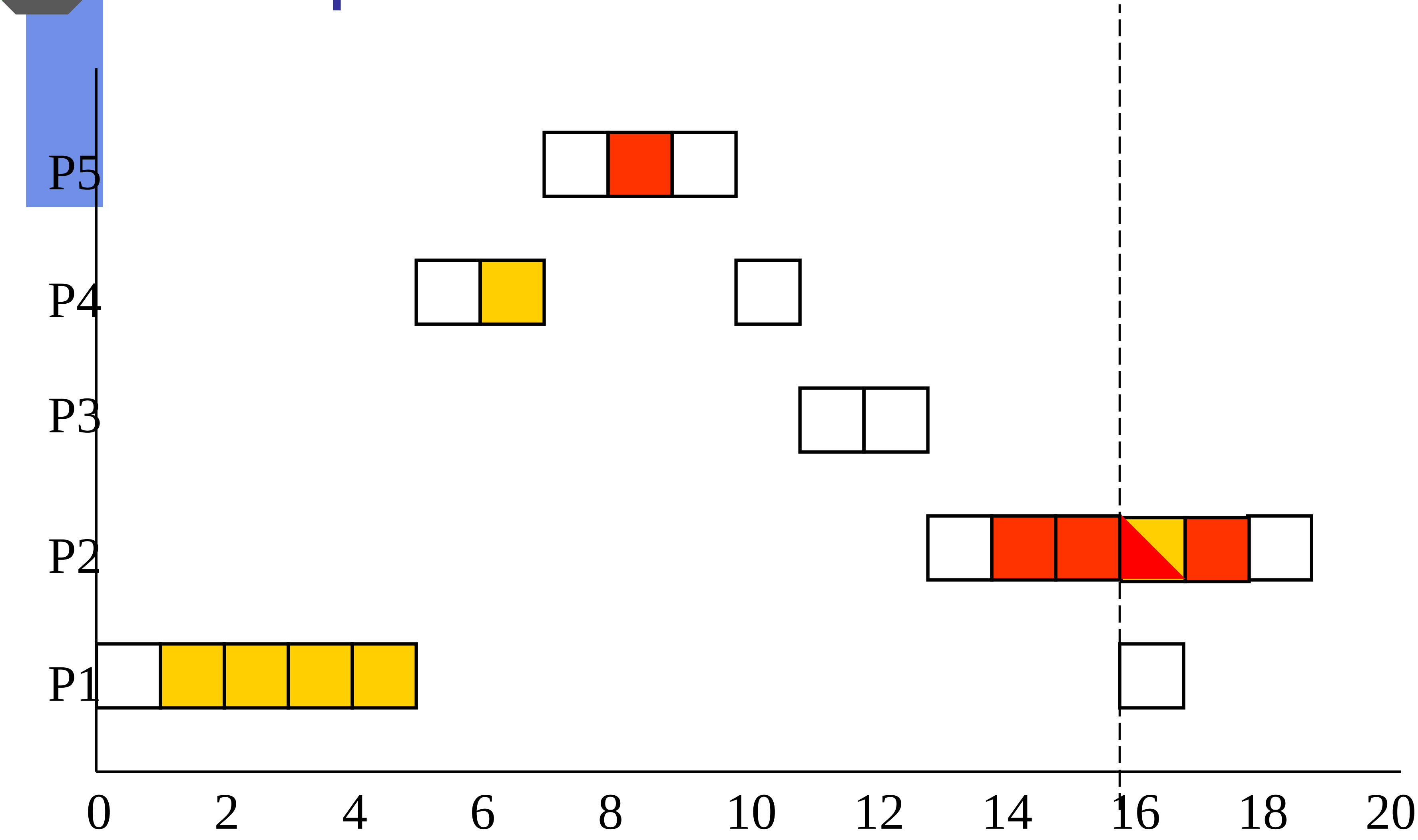
Example



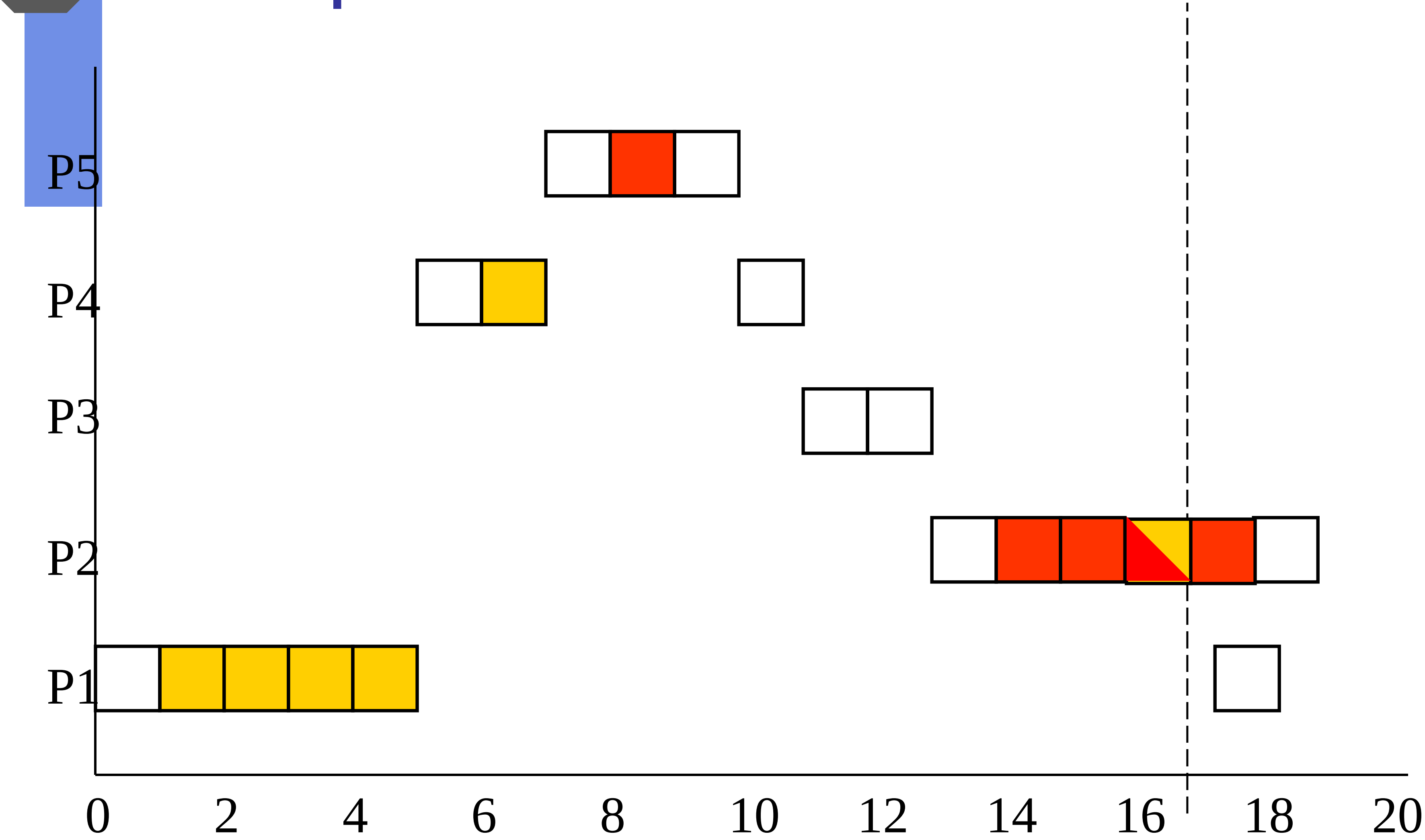
Example



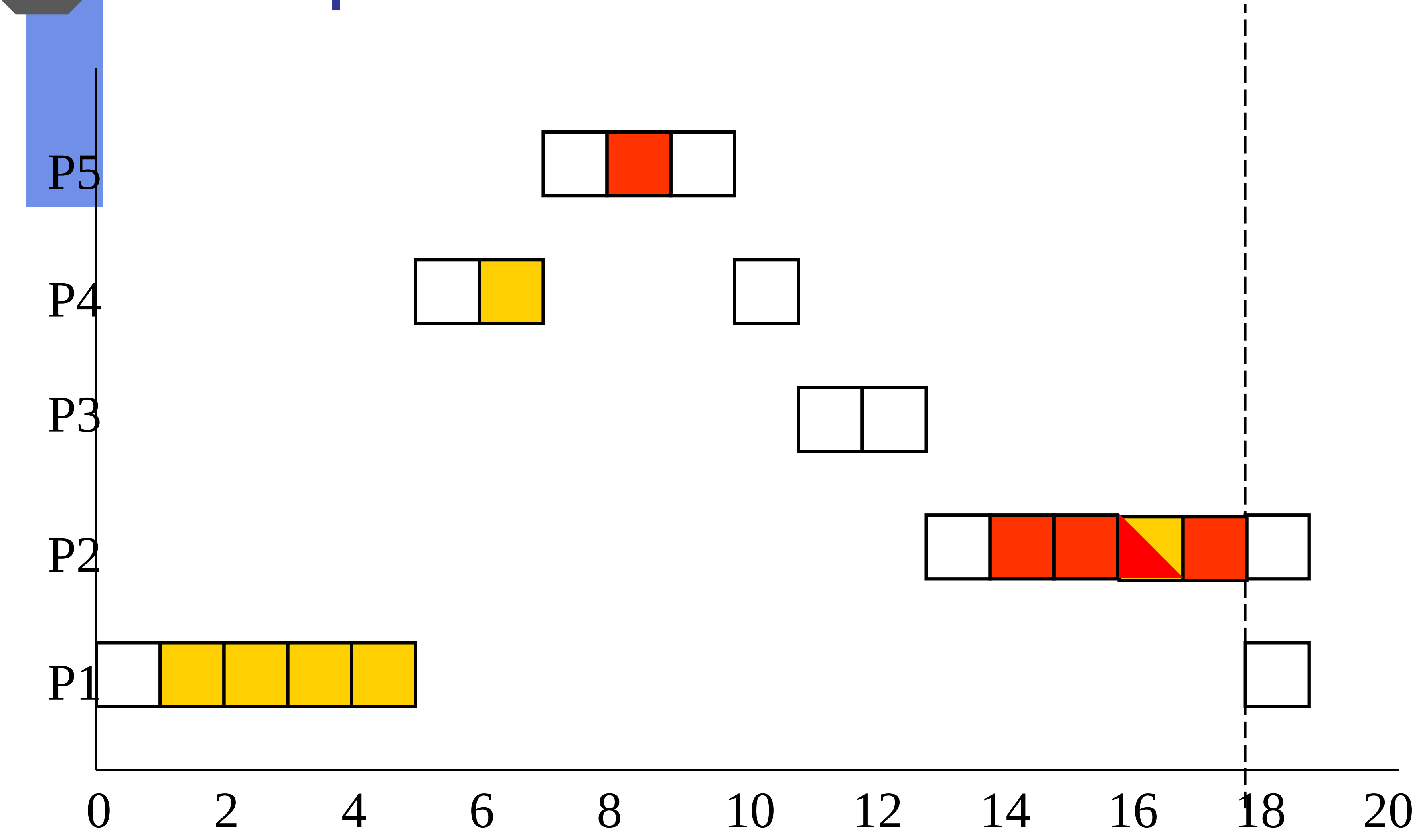
Example



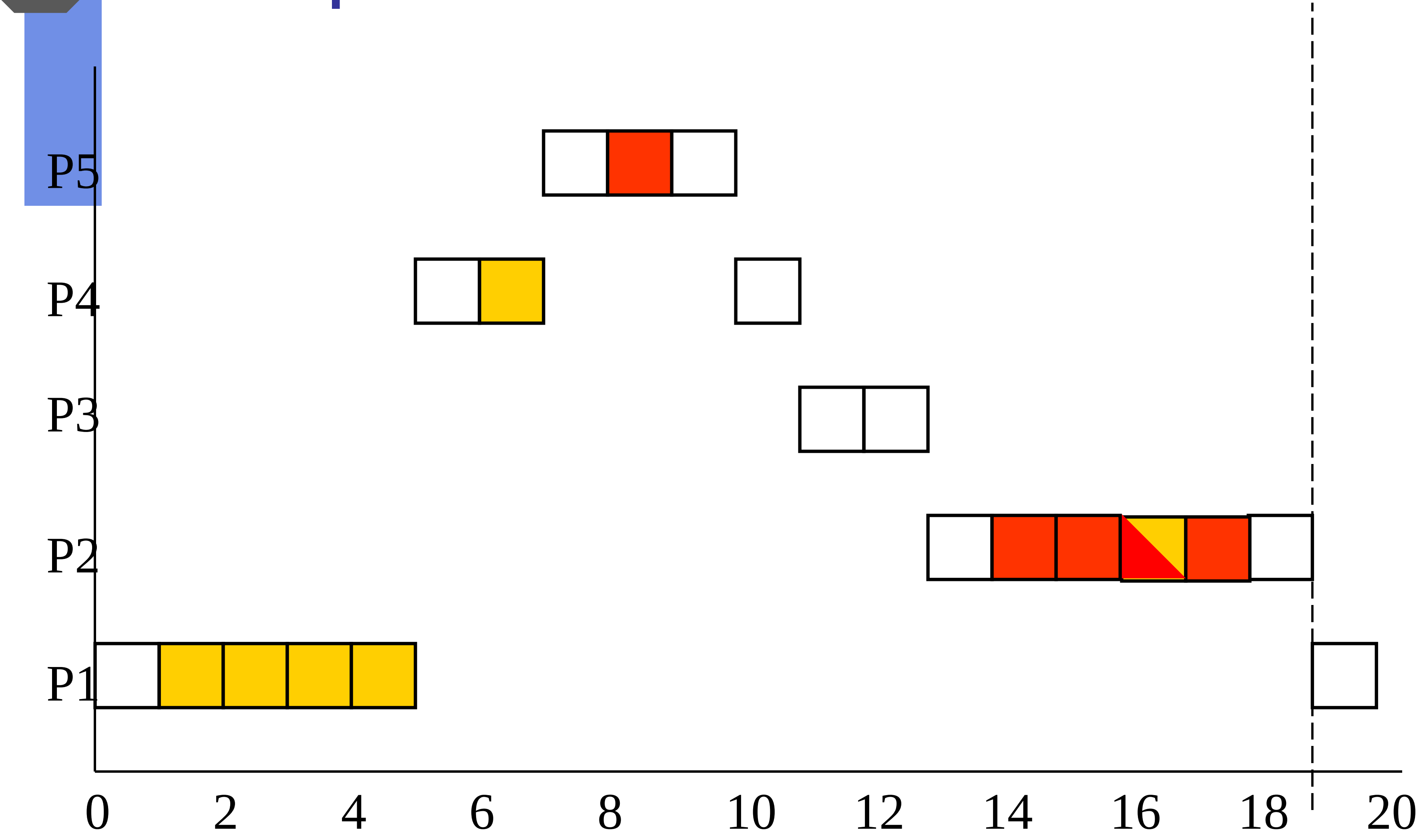
Example



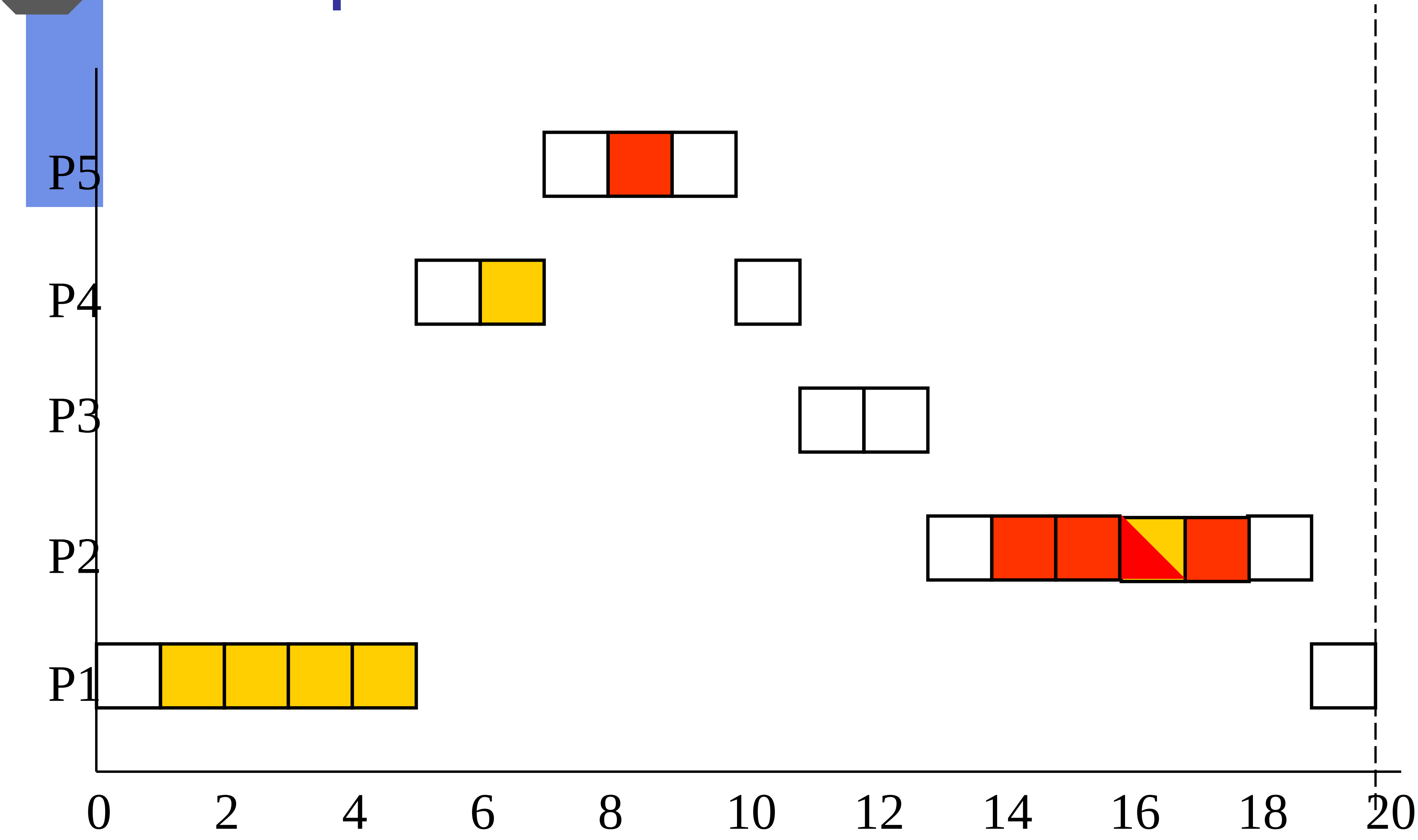
Example



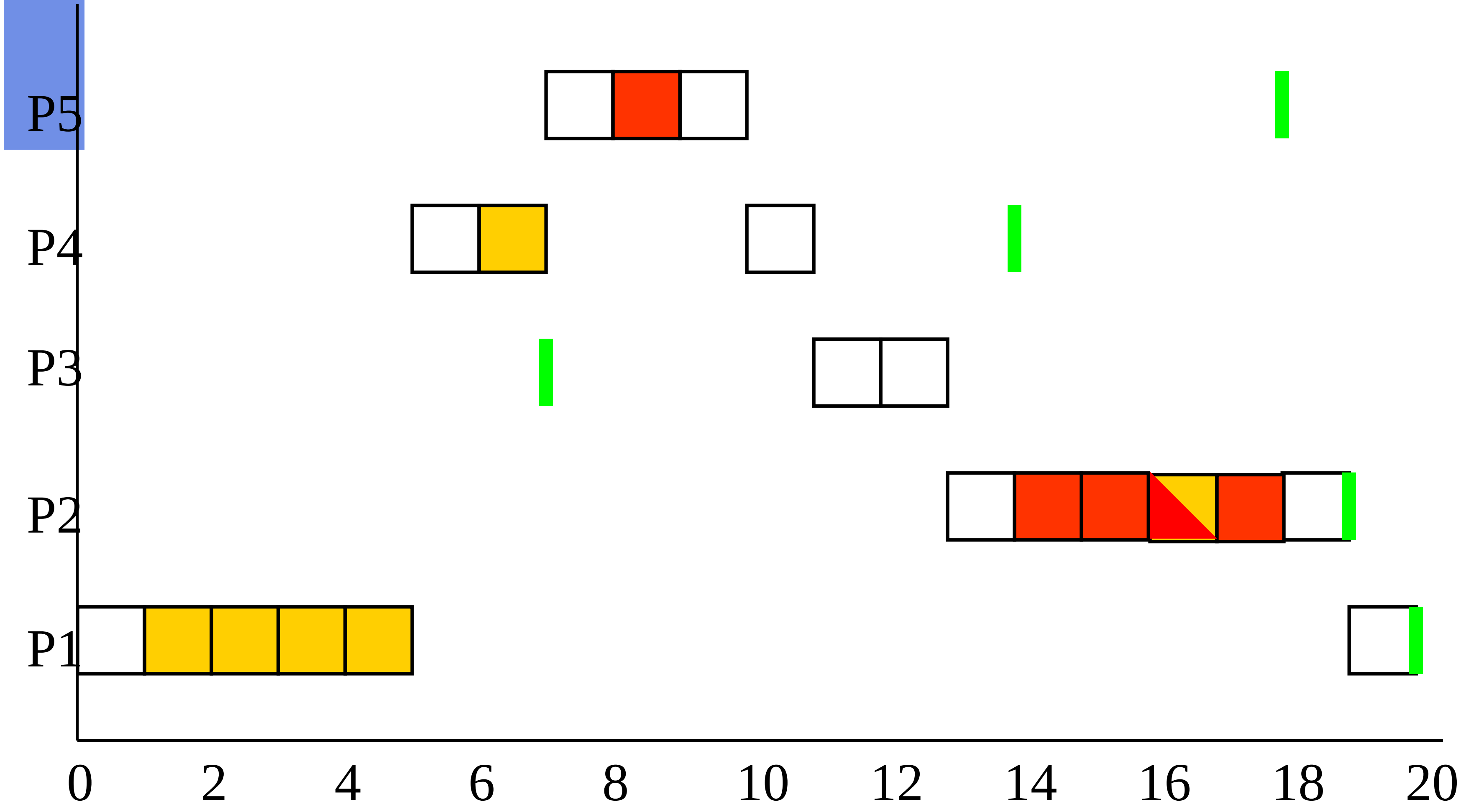
Example



Example



Comparison with *SPD*-Scheduling



Analysis: Nonpreemptive Critical Sections

- Pros
  - Simple
  - No prior knowledge of resource requirements needed
  - Prevents deadlock
- Cons
  - Low priority process blocks high priority process even when there are no resource conflicts
  - Protocol only suitable for trusted software
    - Usually implemented by *interrupt disabling*
  - In CS there is no system calls otherwise *CPU wasting* in case of a *"blocking"* system call

Worst-Case Blocking Time

- Longest lower-priority critical section:

$$bt_i(rc) = \max_{i+1 \leq k \leq n} \{cst_k\}$$

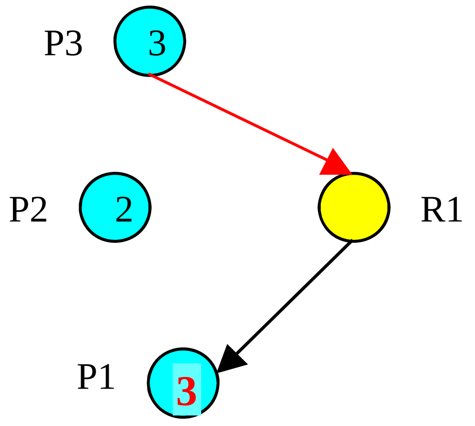
bt = blocking time

cst = critical section time

Not that realistic

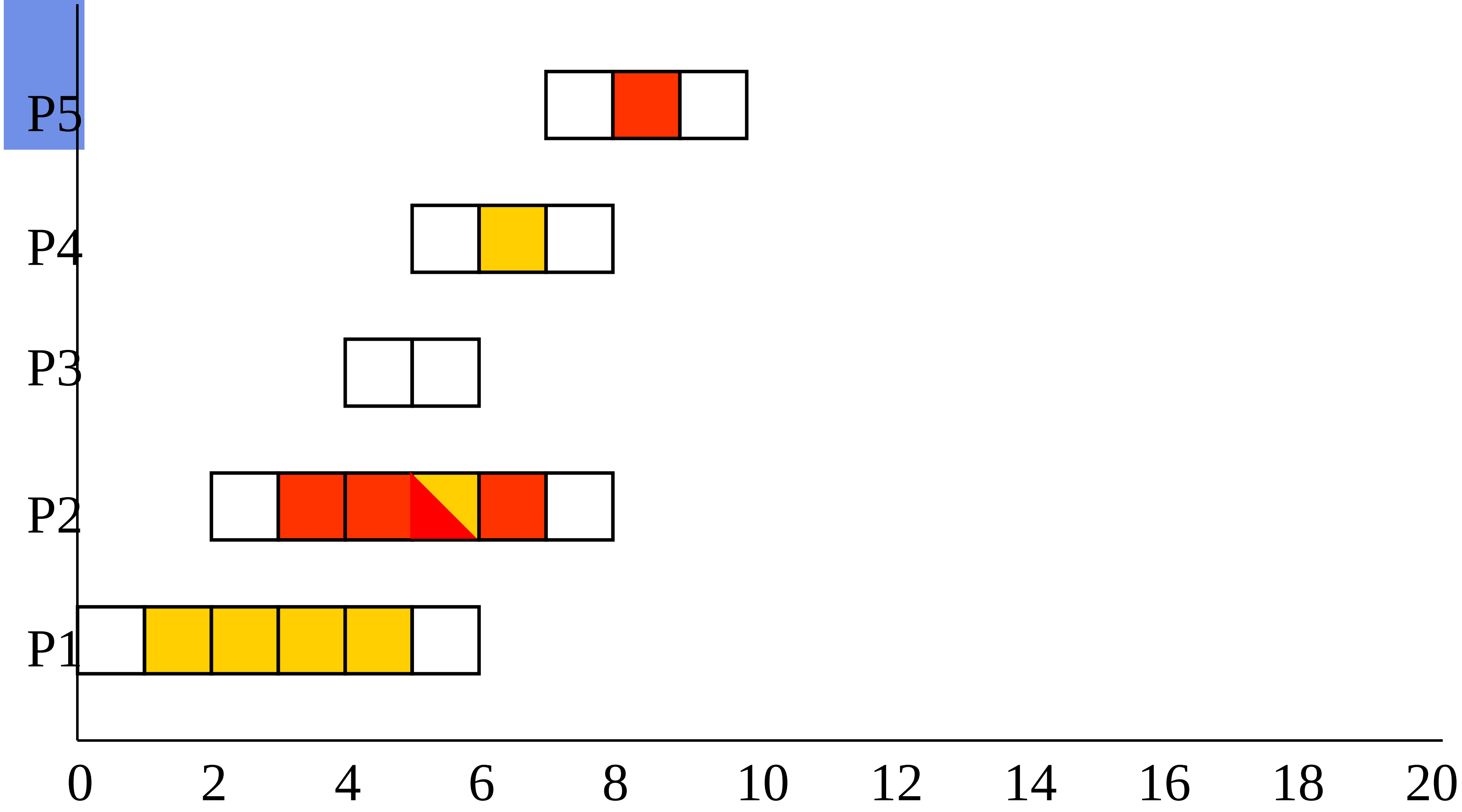
Priority Inheritance (PI)

- When a *high-priority process* (P3) blocks, the low-priority process (P1) inherits the *current priority* of the blocking process

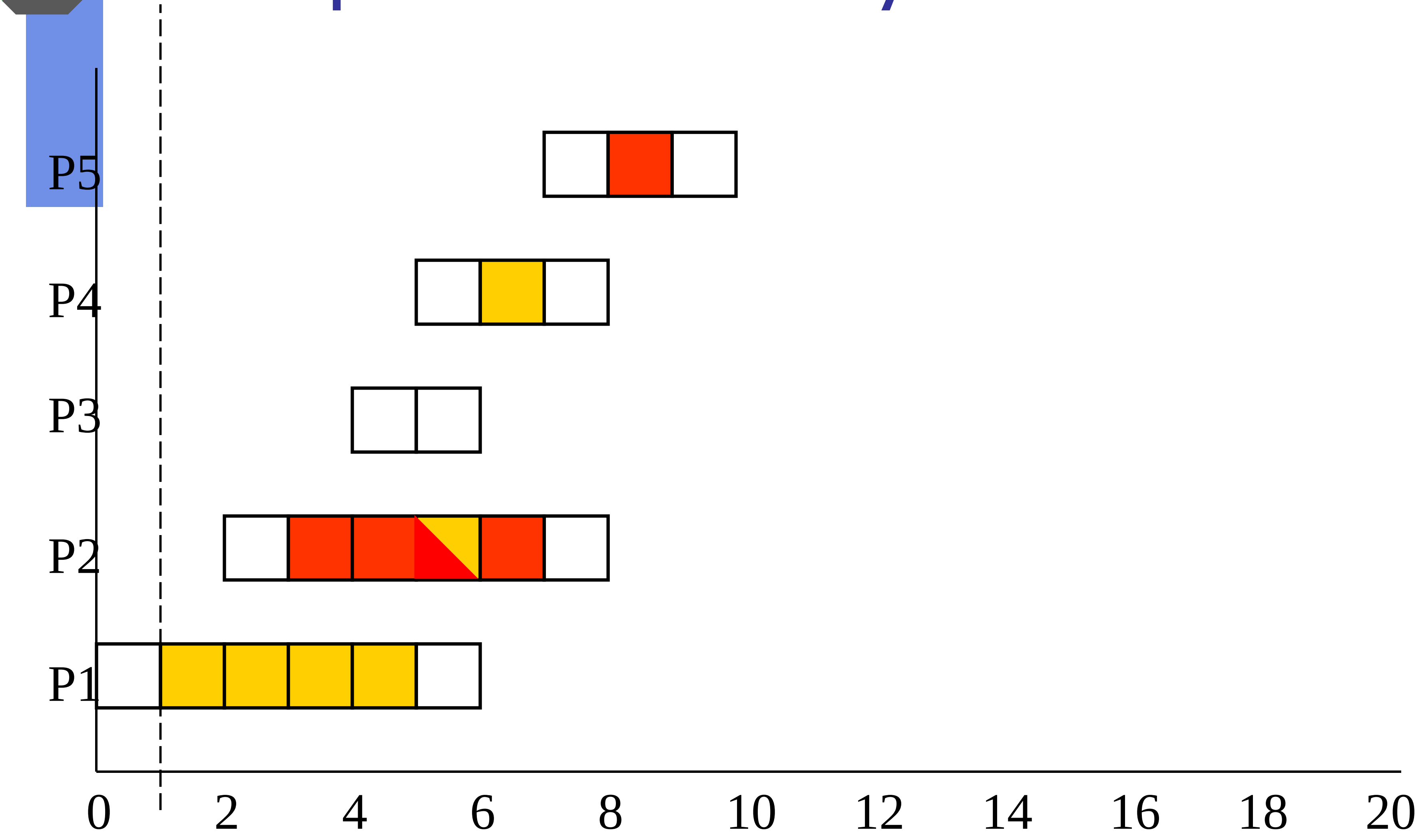


- PI bounds *priority inversion*

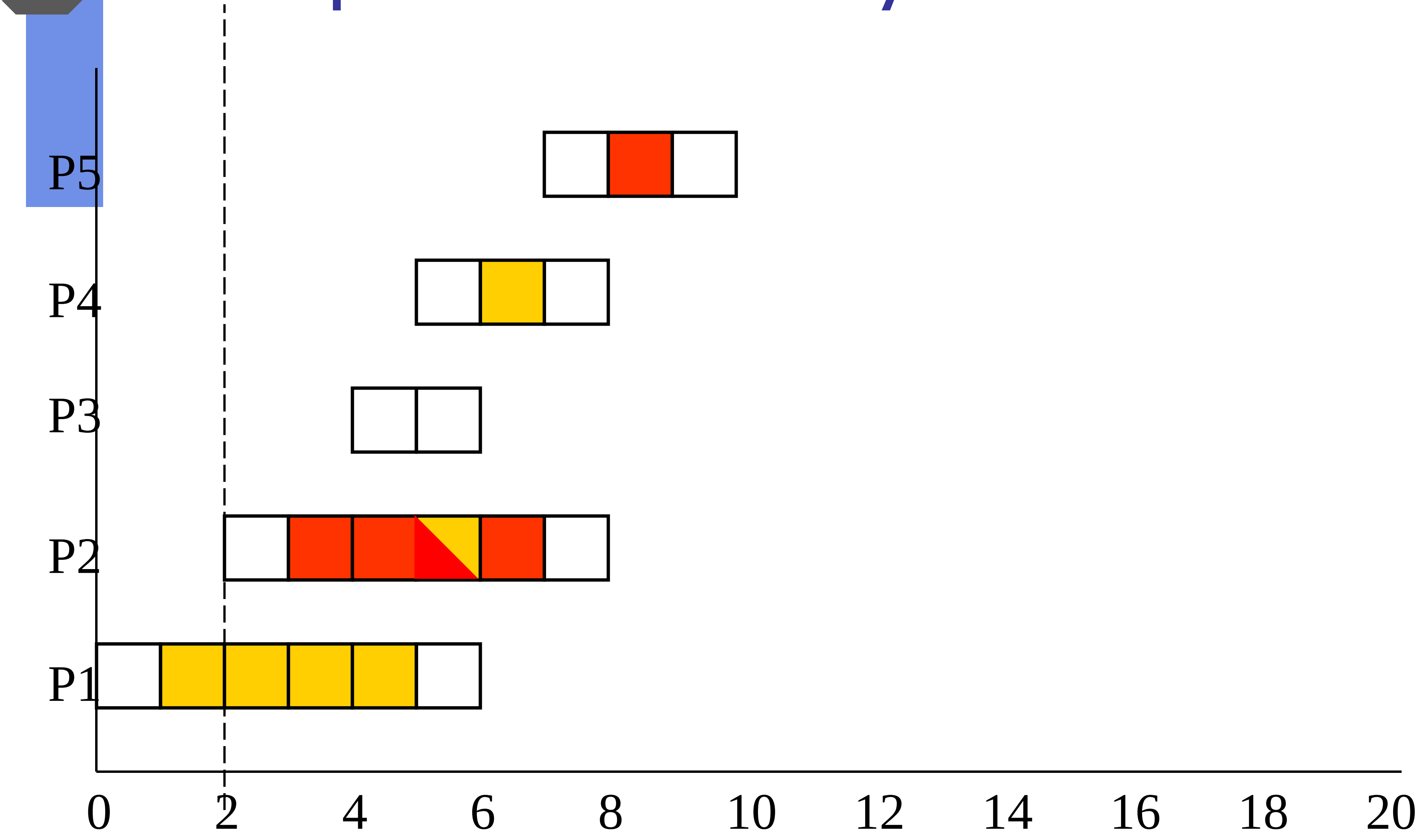
Example with Priority Inheritance



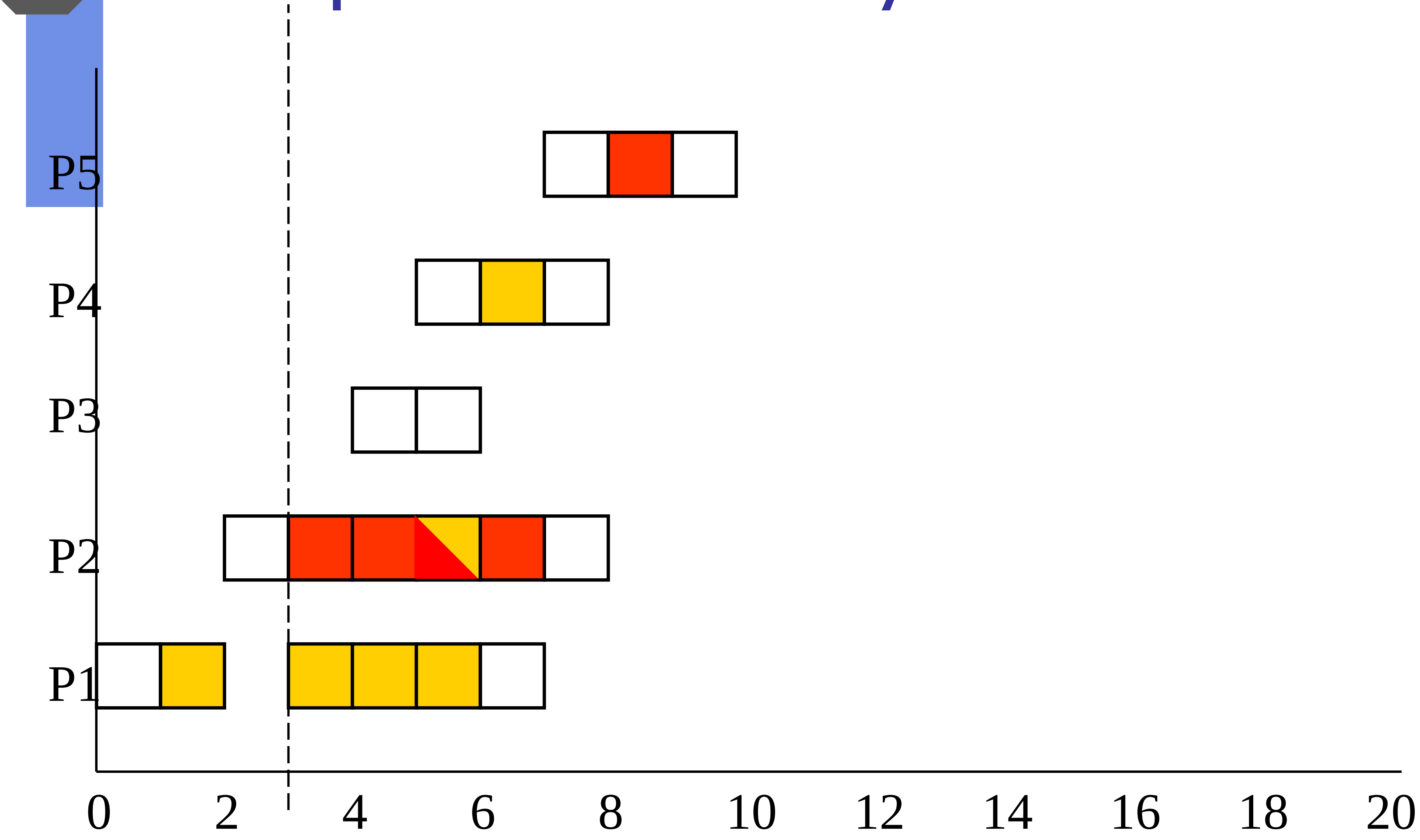
Example with Priority Inheritance



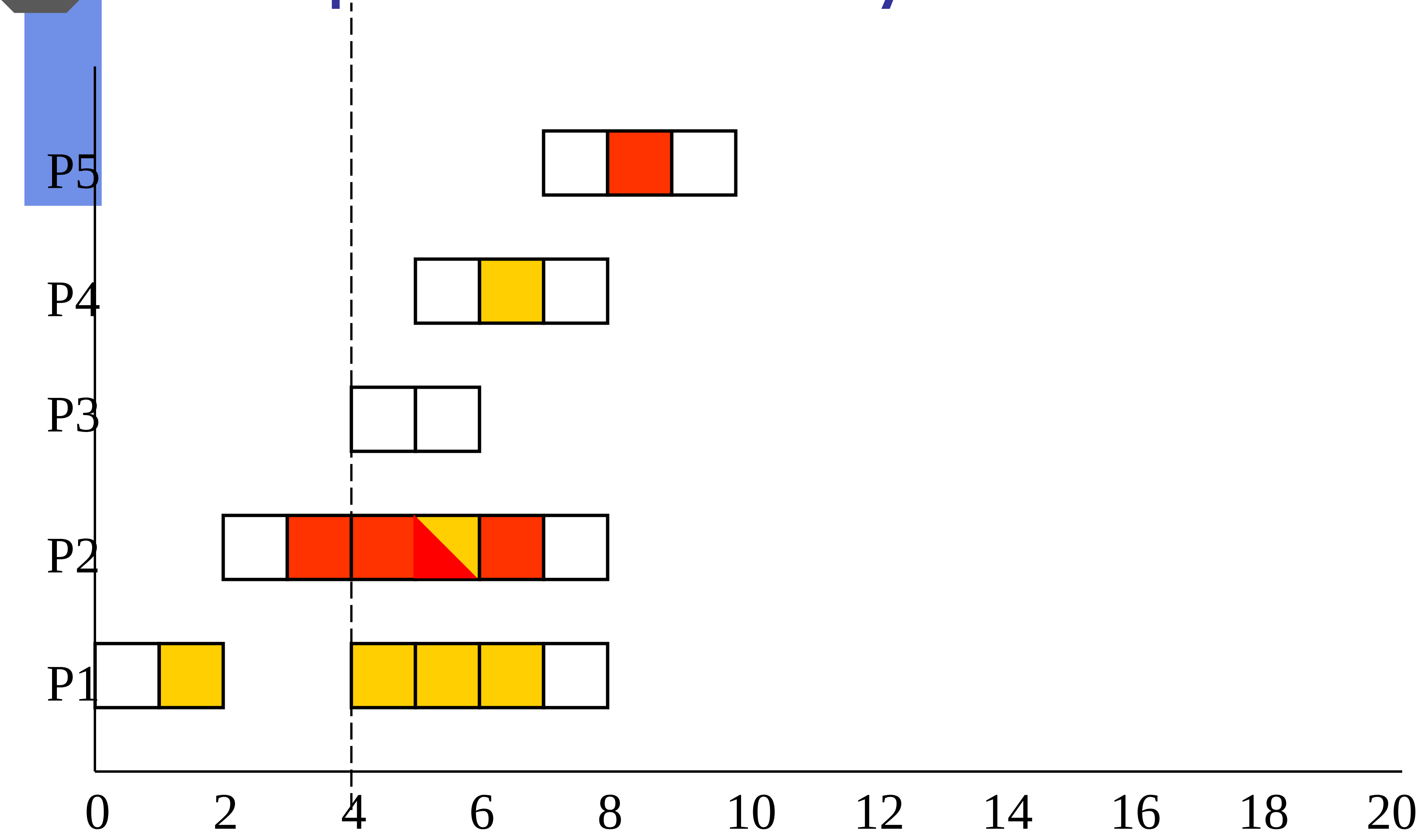
Example with Priority Inheritance



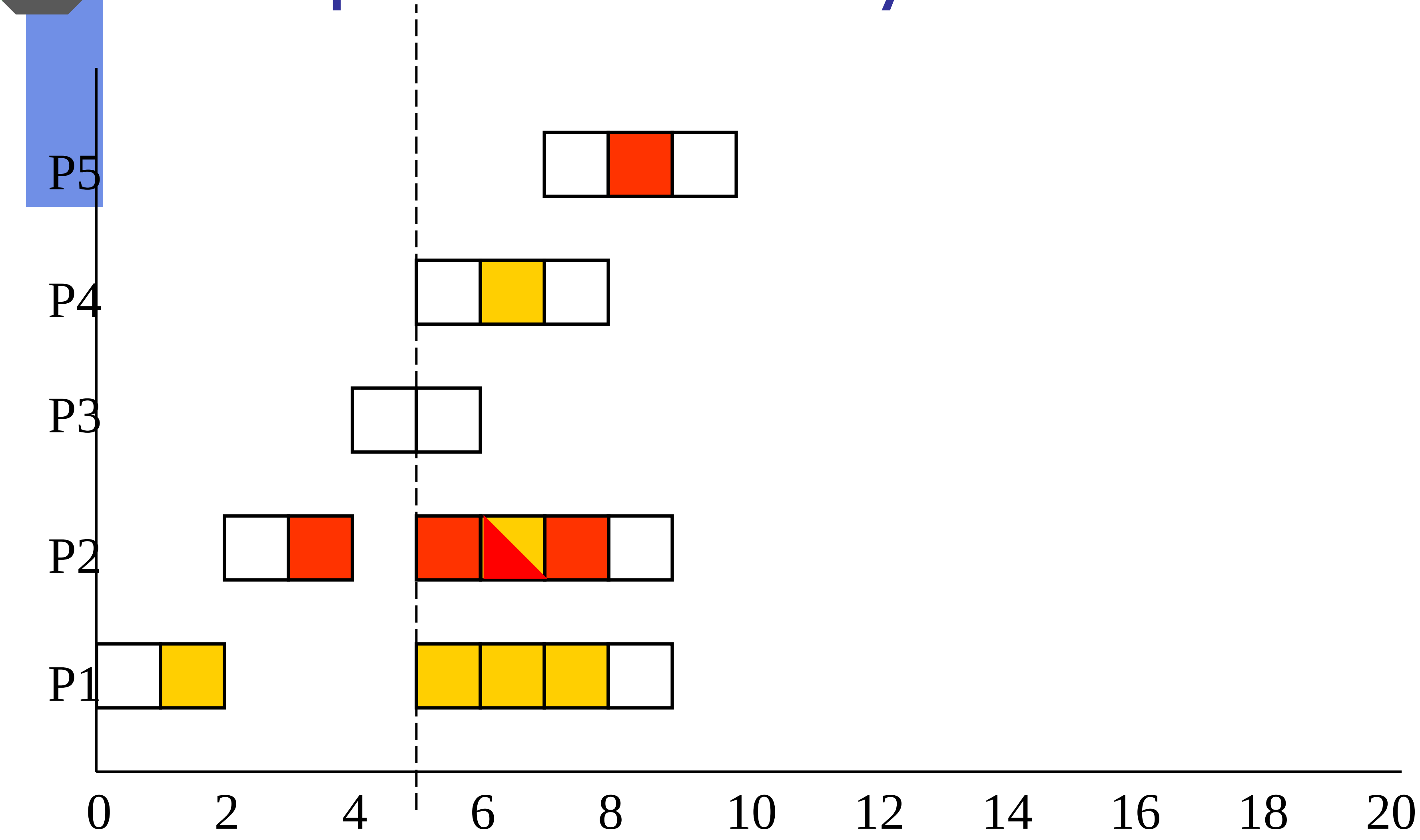
Example with Priority Inheritance



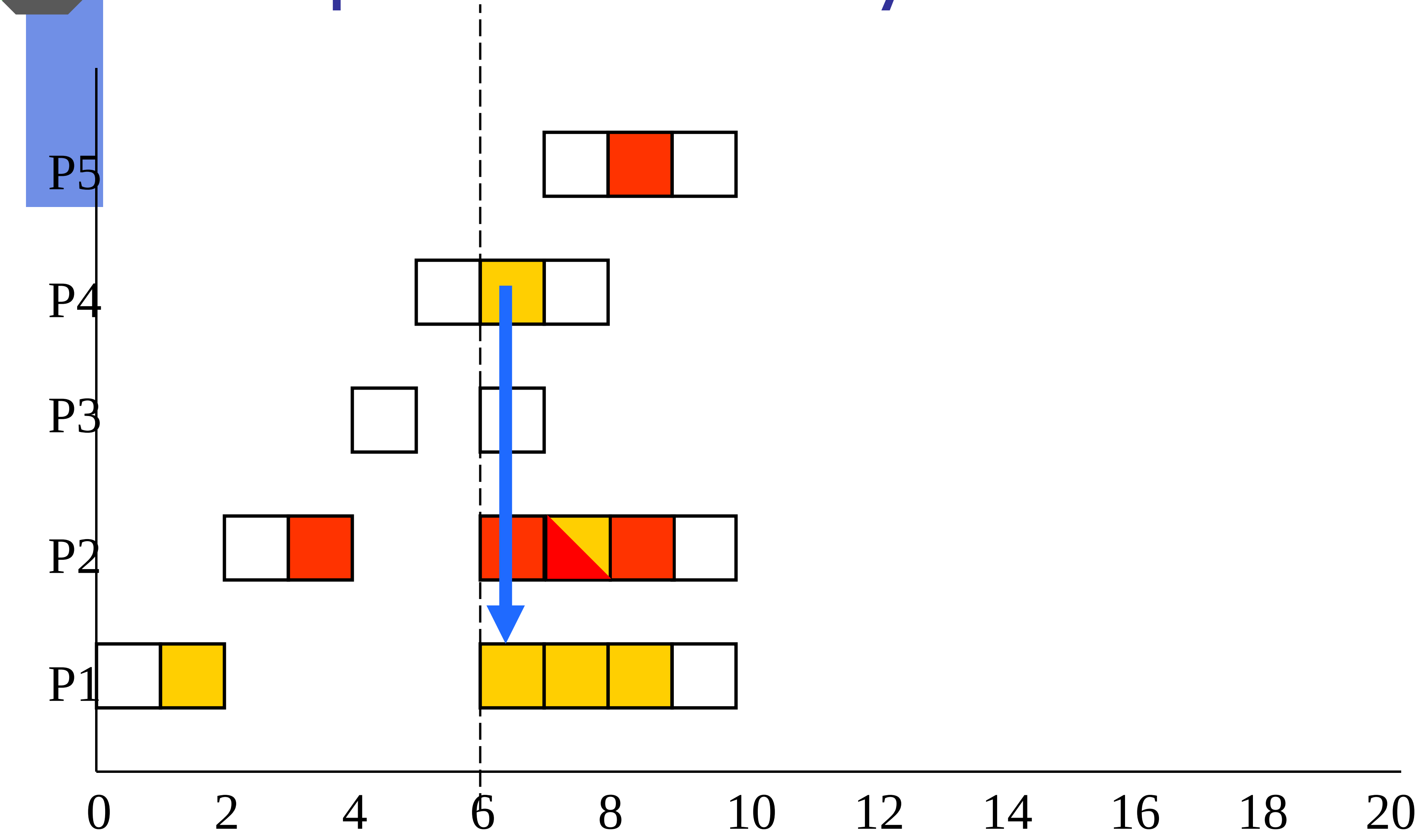
Example with Priority Inheritance



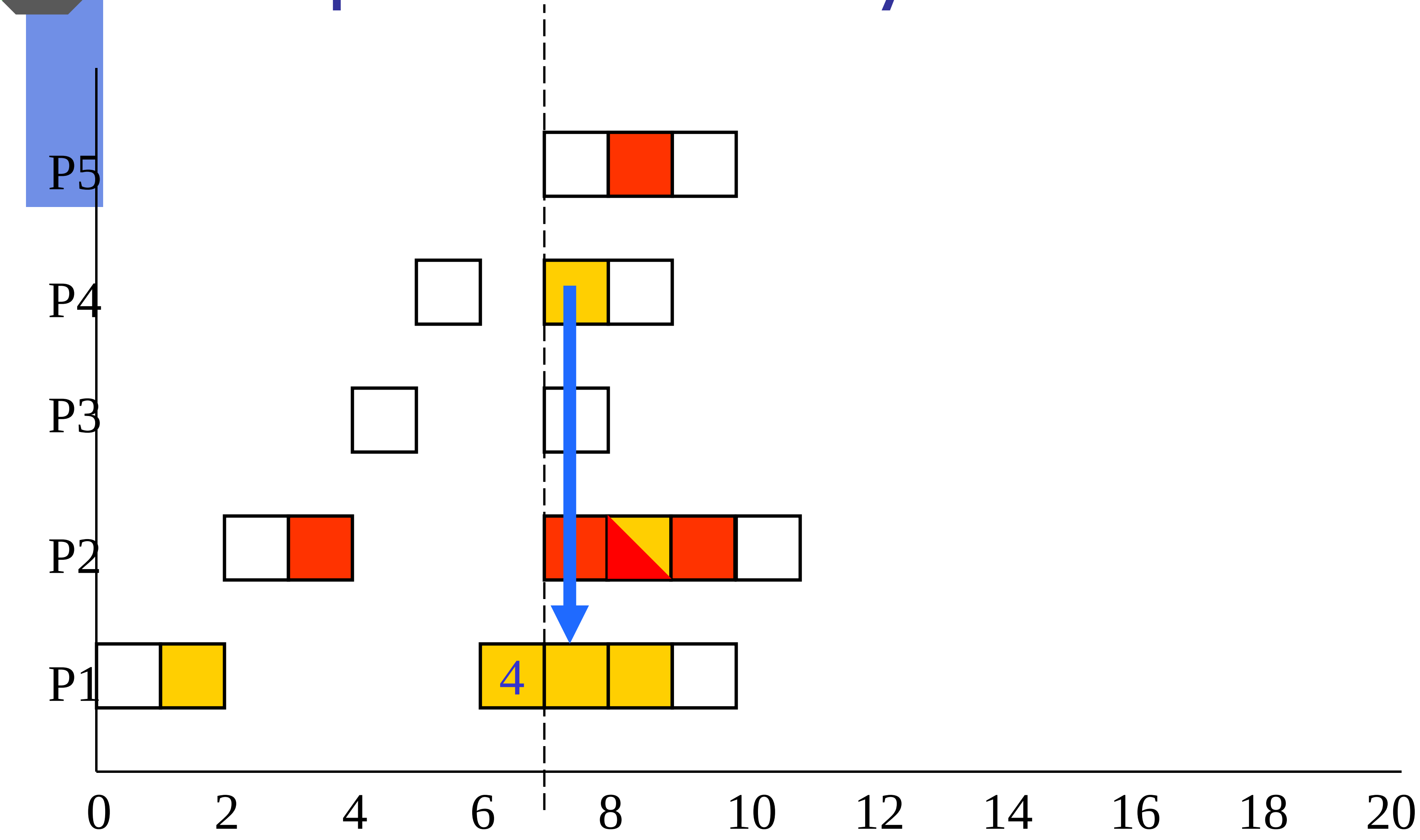
Example with Priority Inheritance



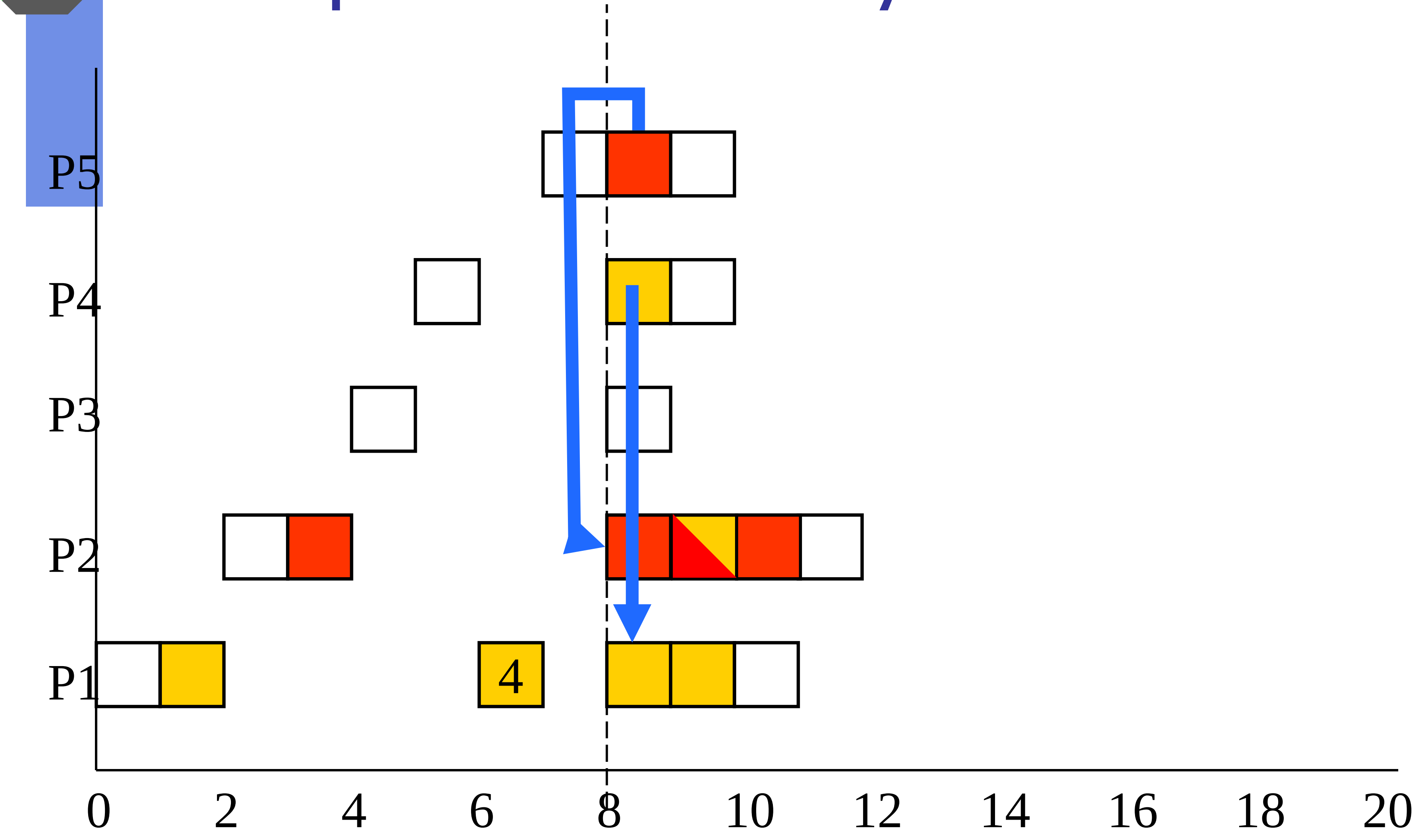
Example with Priority Inheritance



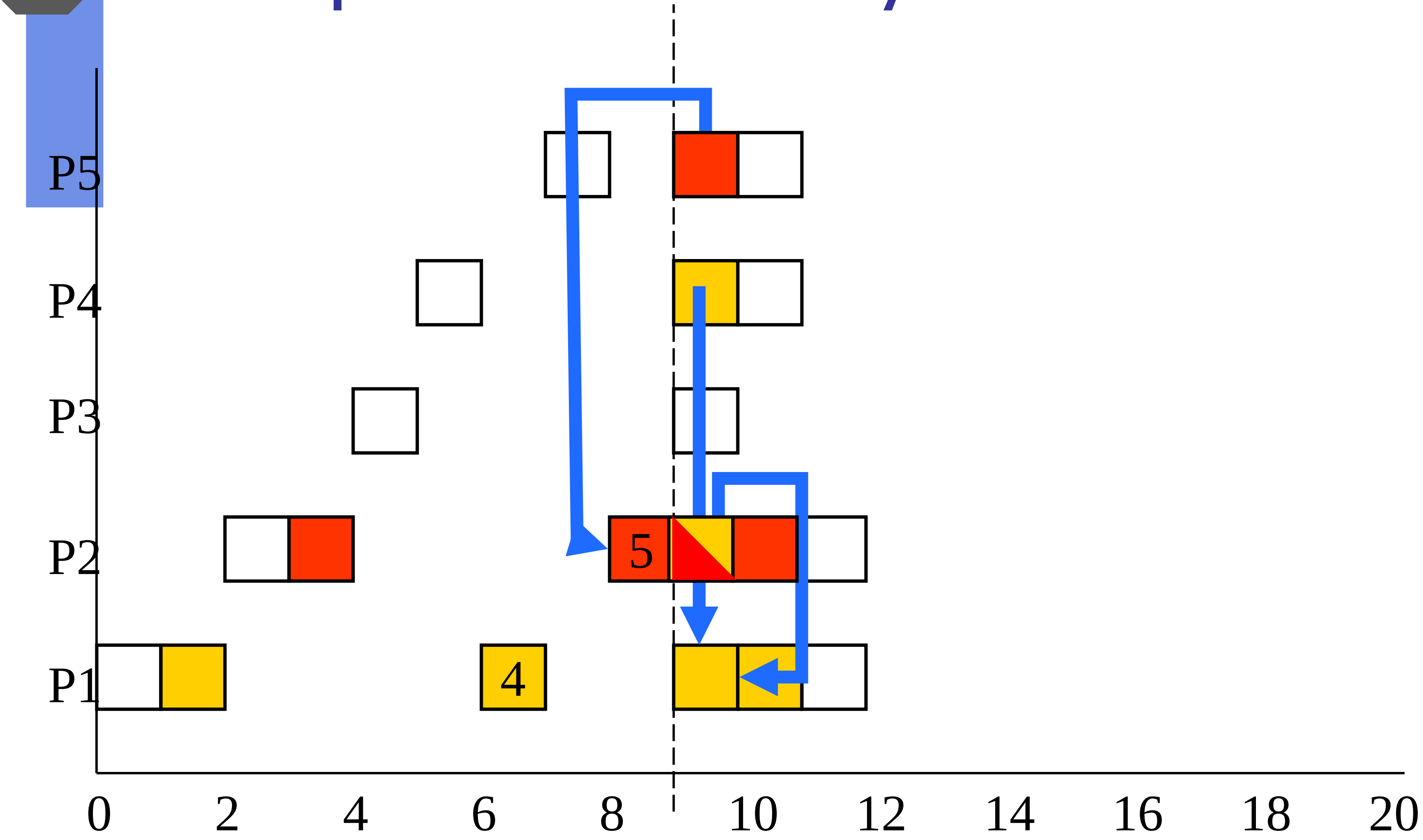
Example with Priority Inheritance



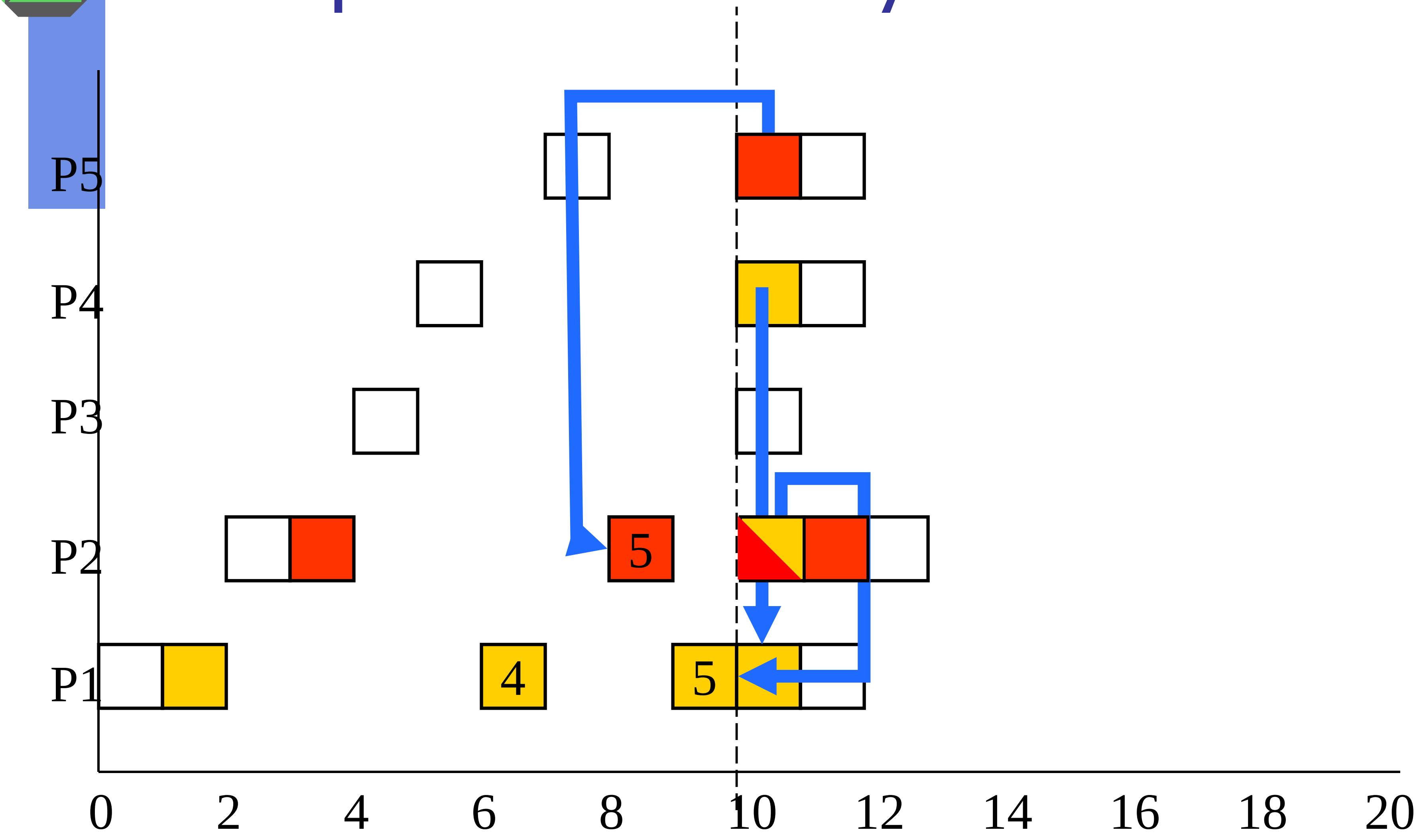
Example with Priority Inheritance



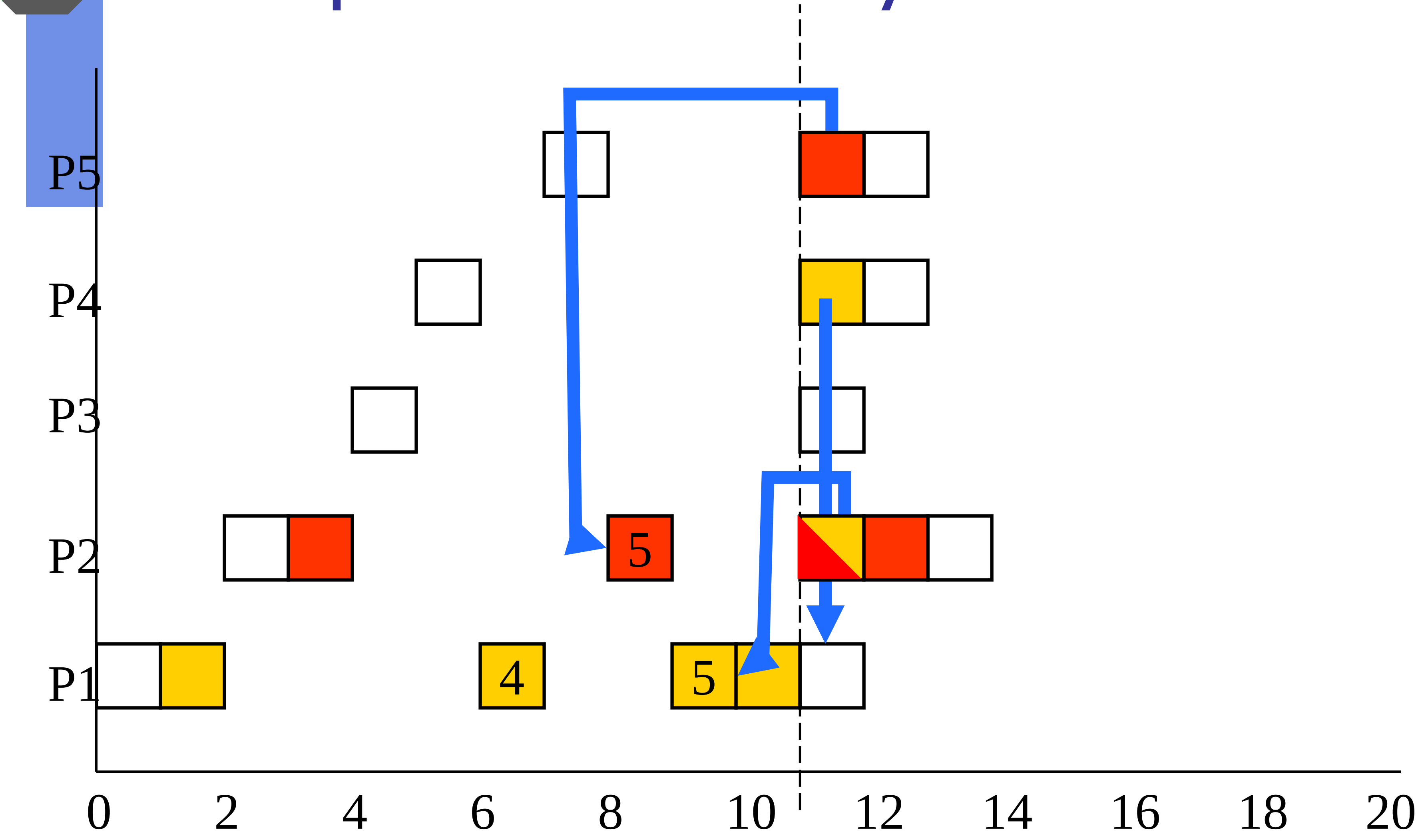
Example with Priority Inheritance



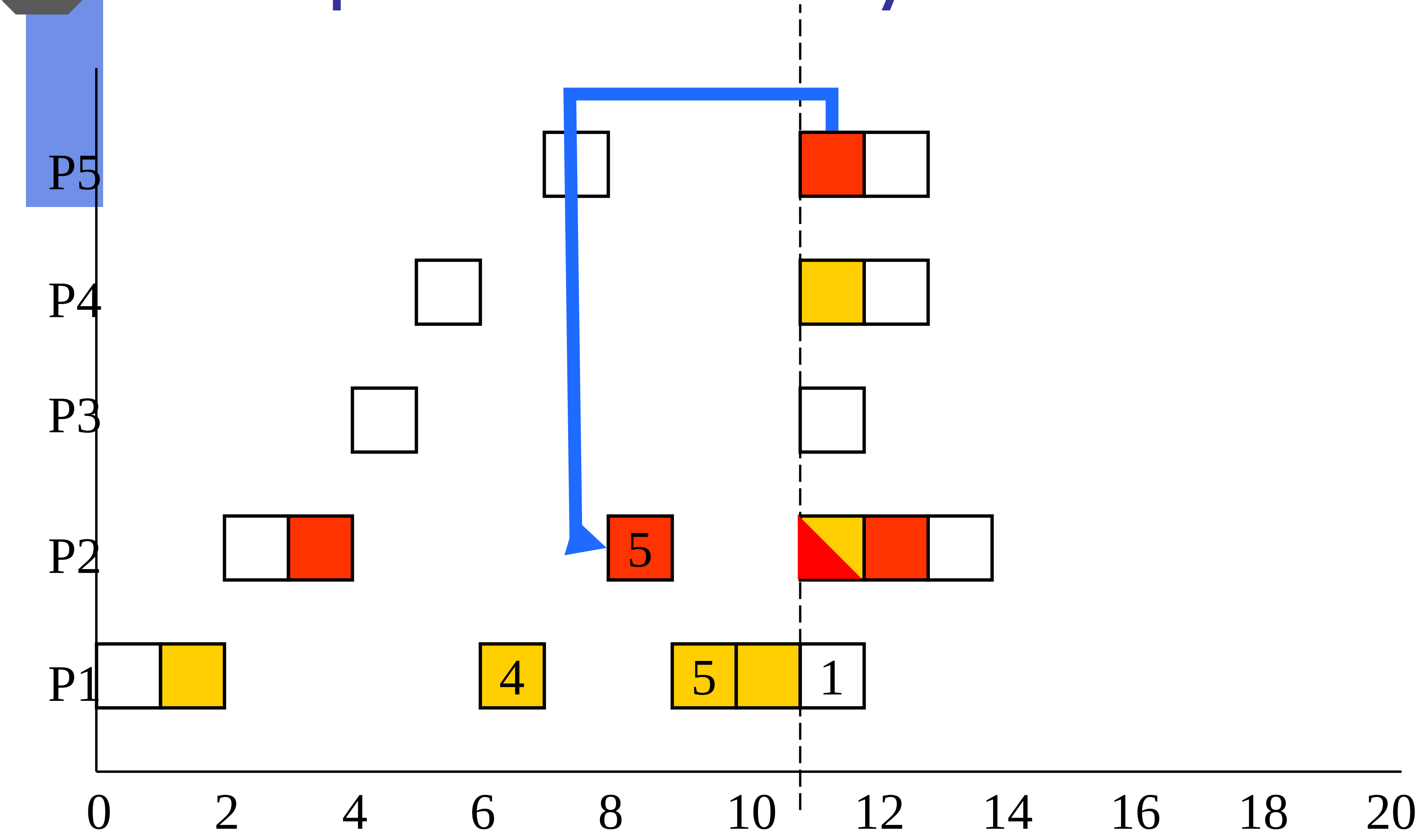
Example with Priority Inheritance



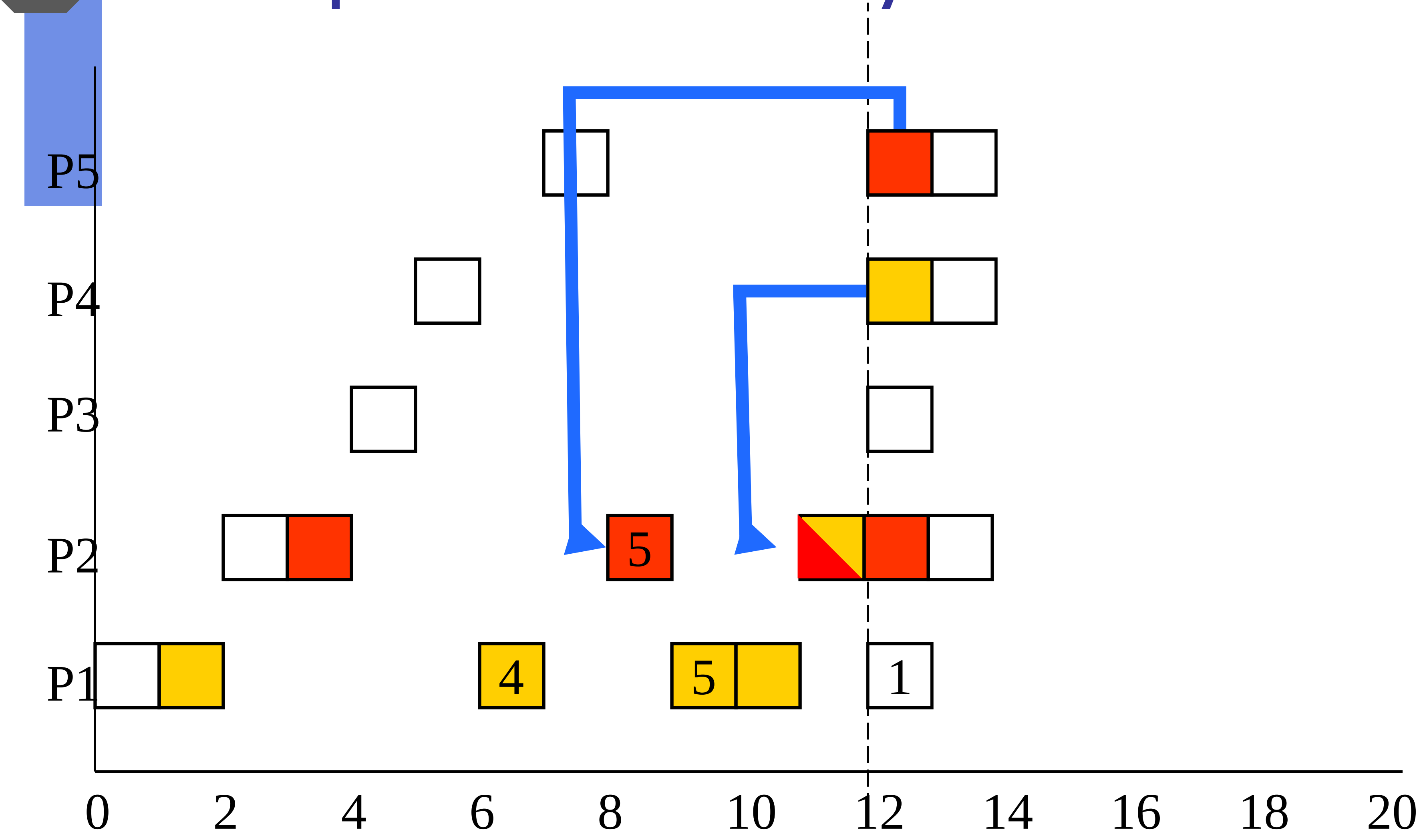
Example with Priority Inheritance



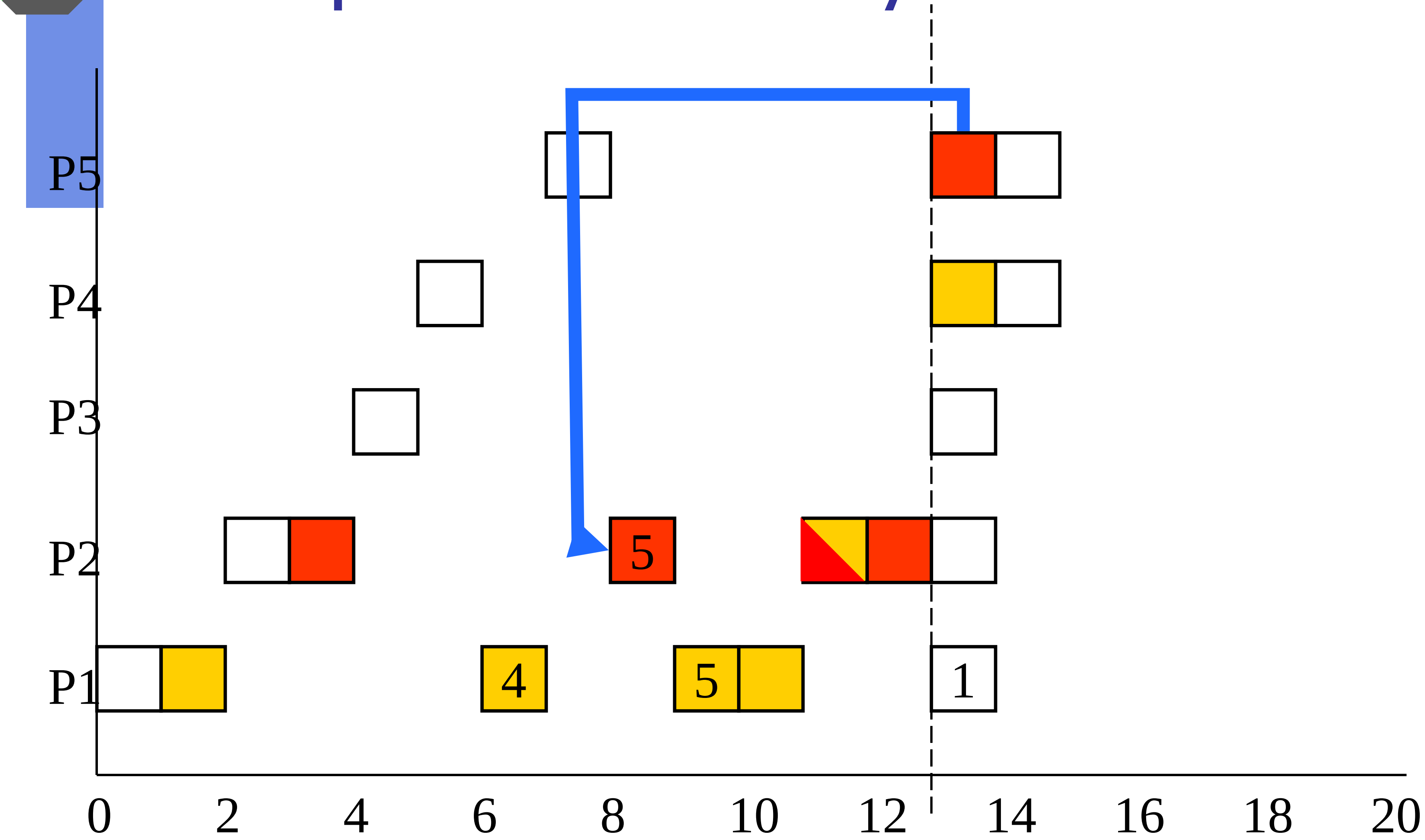
Example with Priority Inheritance



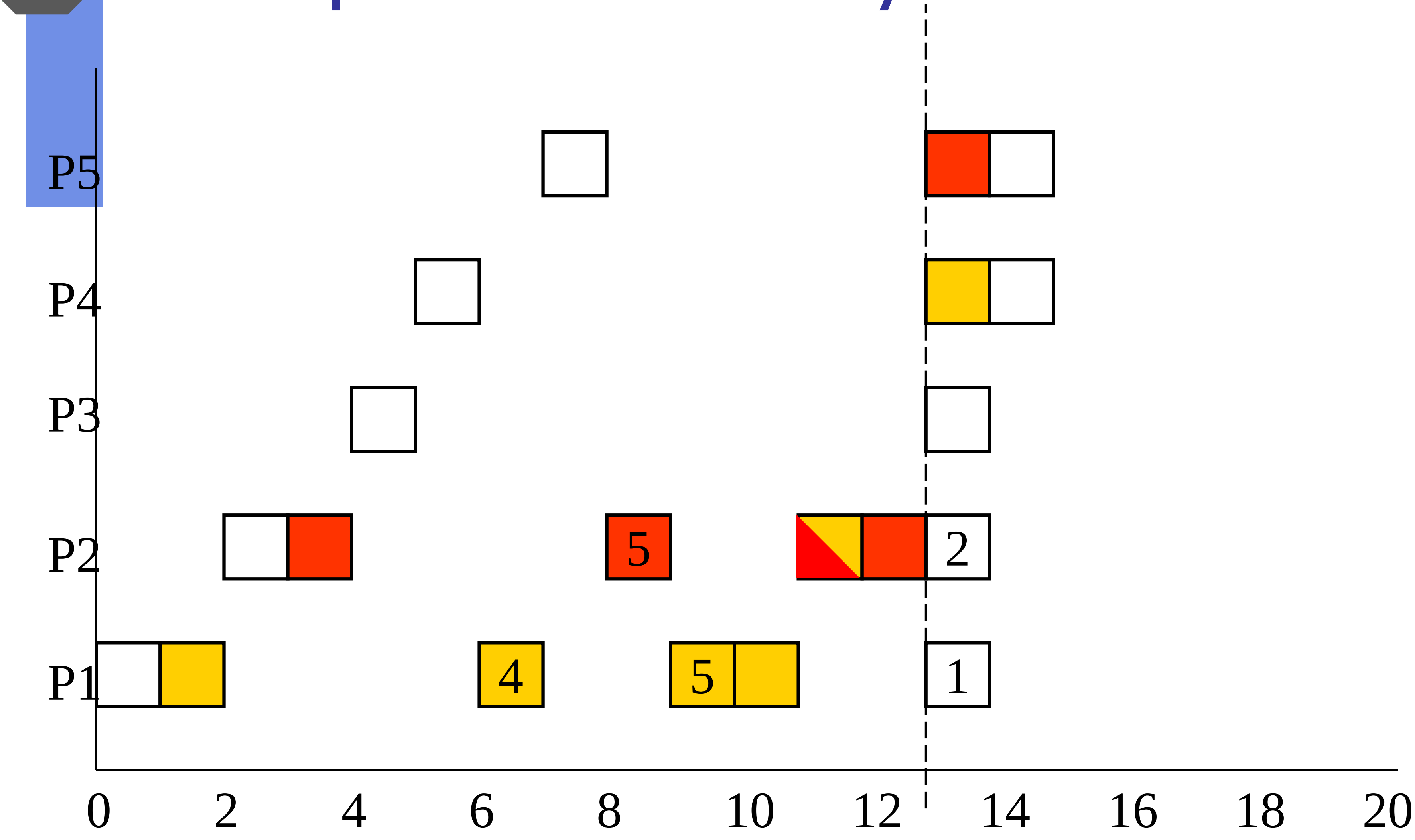
Example with Priority Inheritance



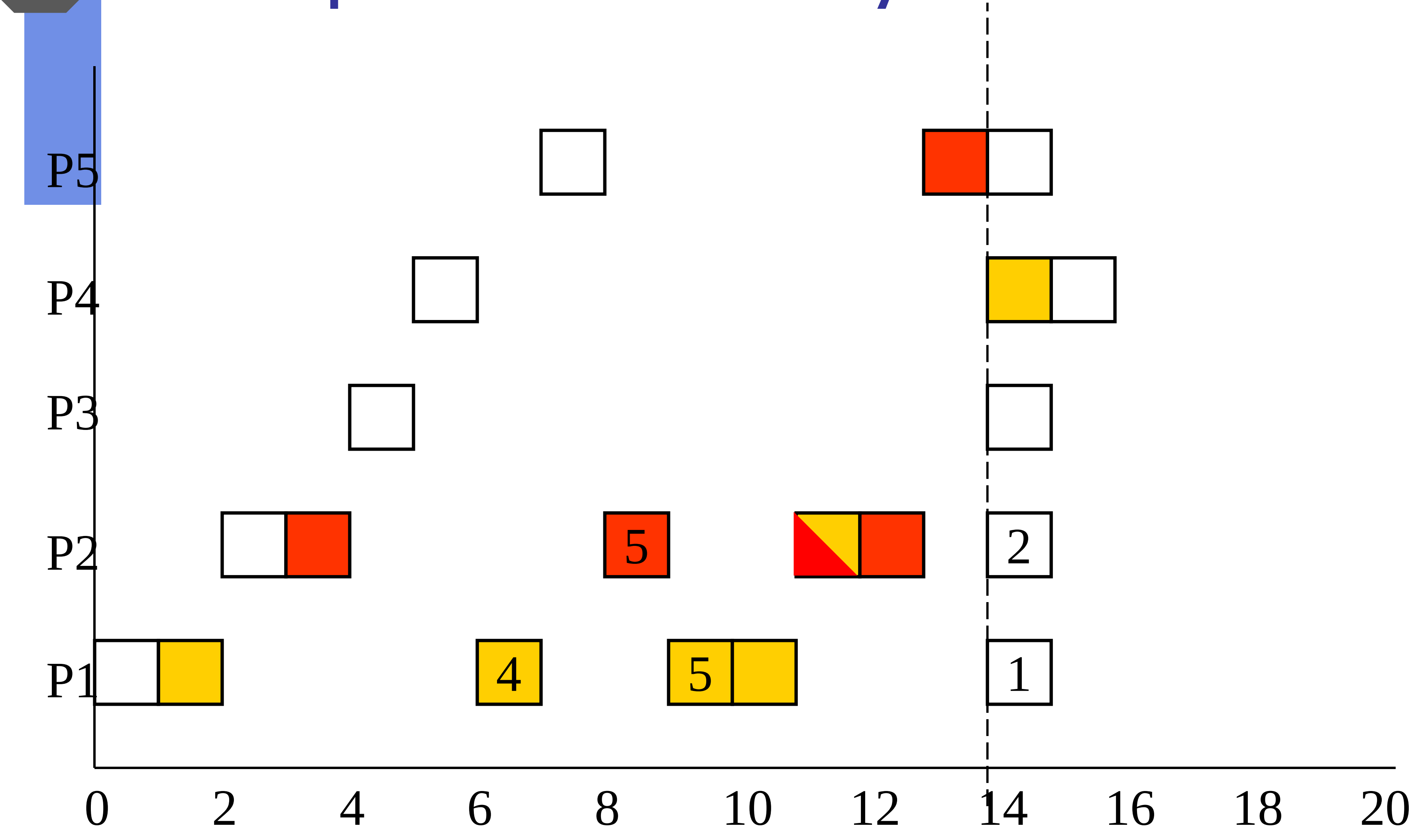
Example with Priority Inheritance



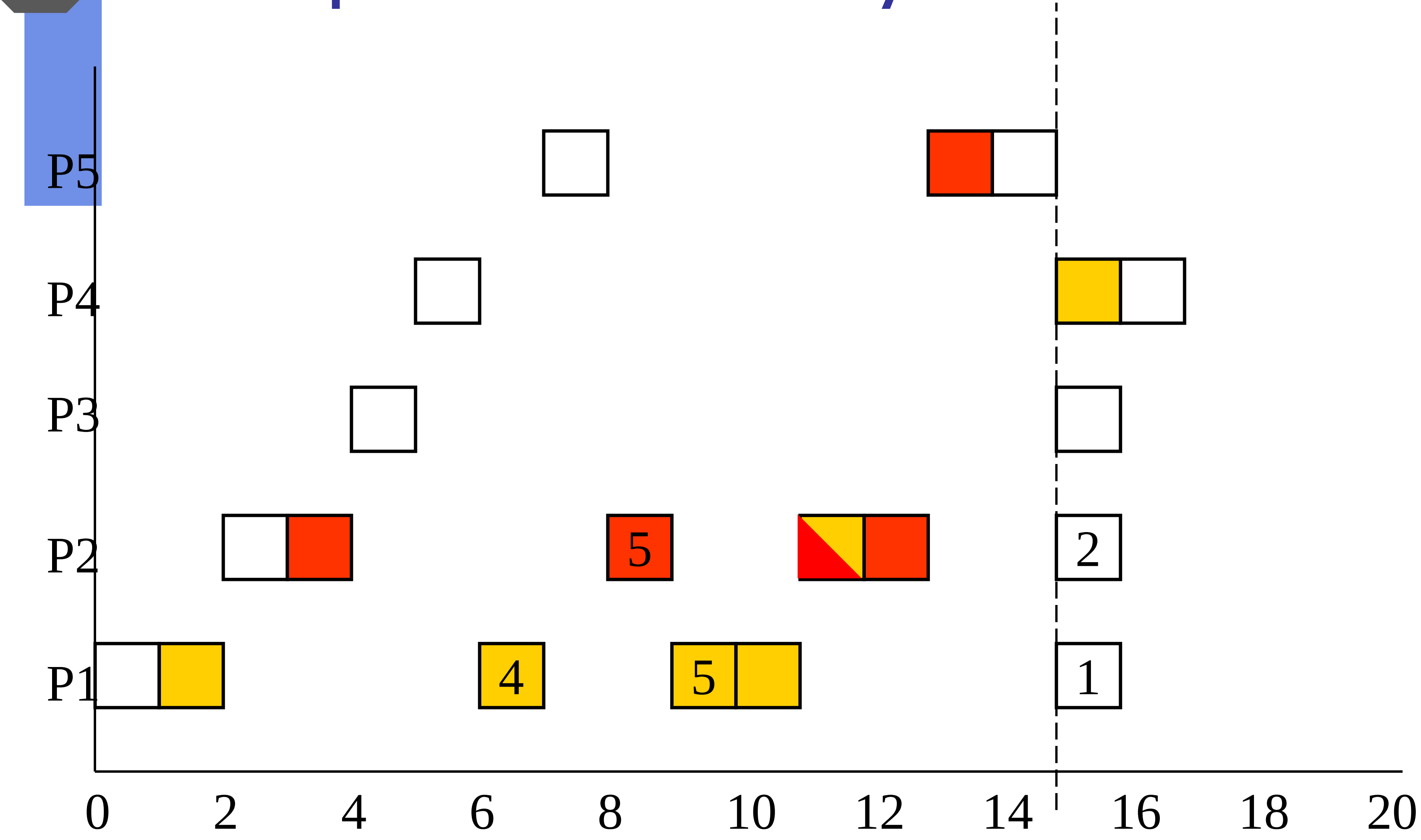
Example with Priority Inheritance



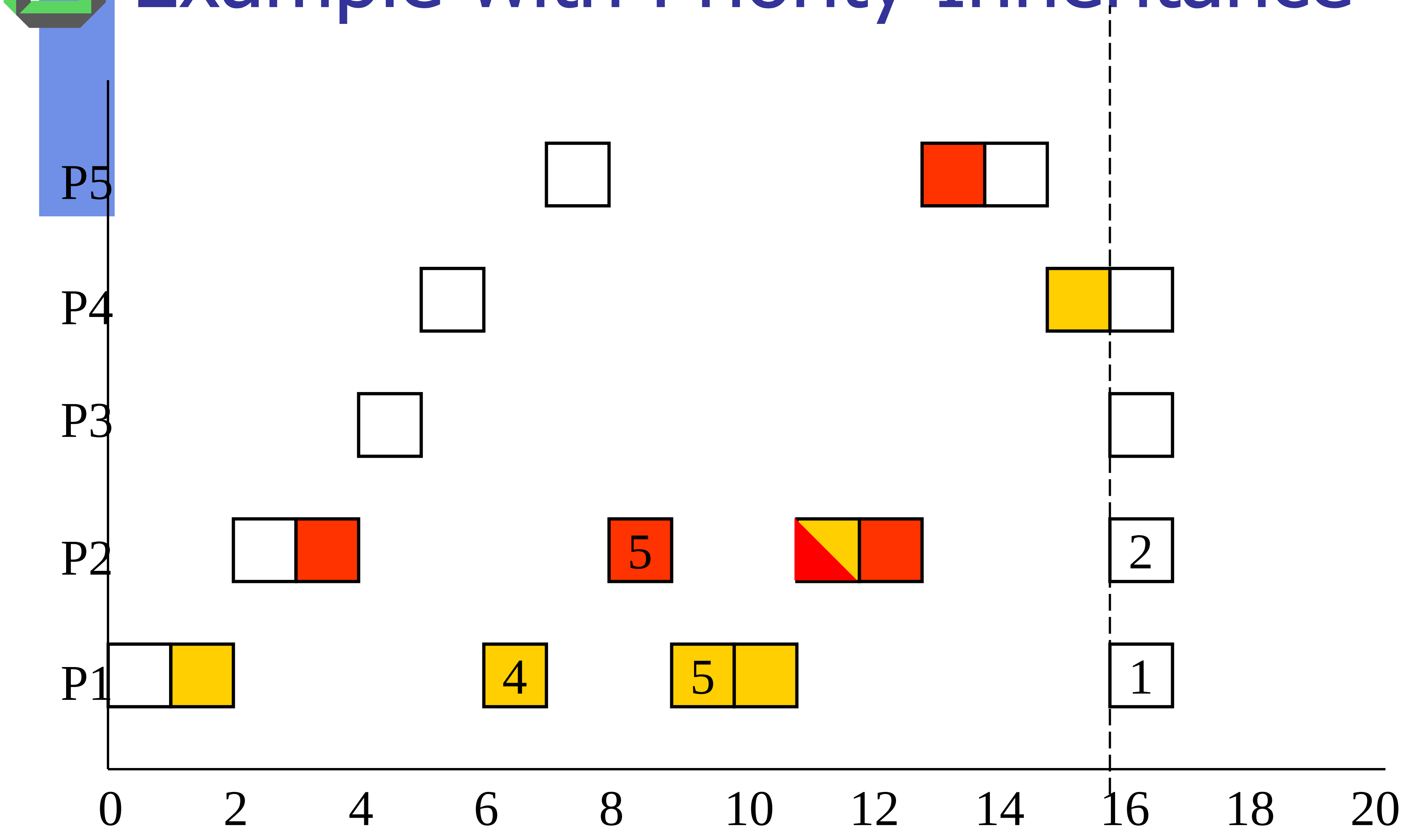
Example with Priority Inheritance



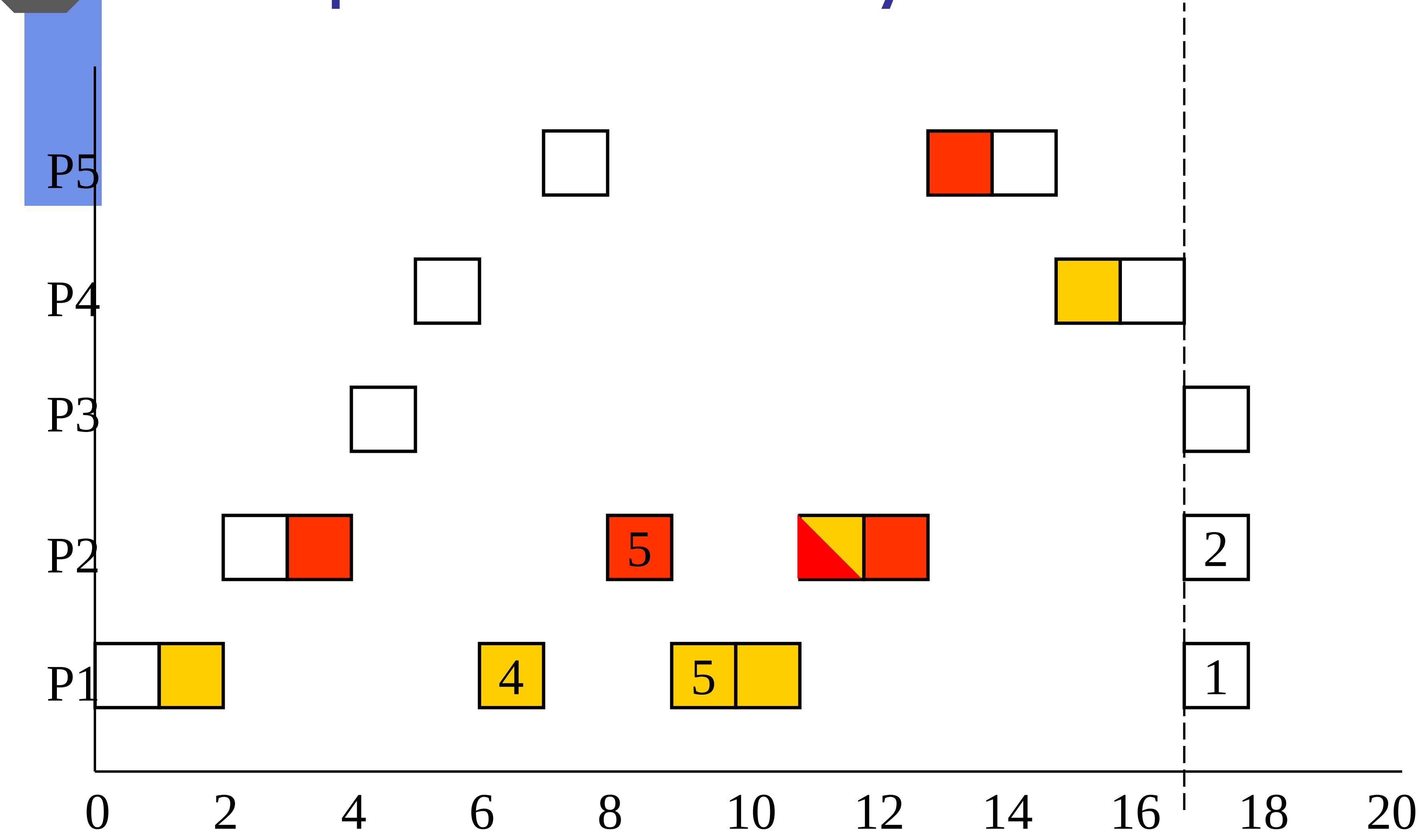
Example with Priority Inheritance



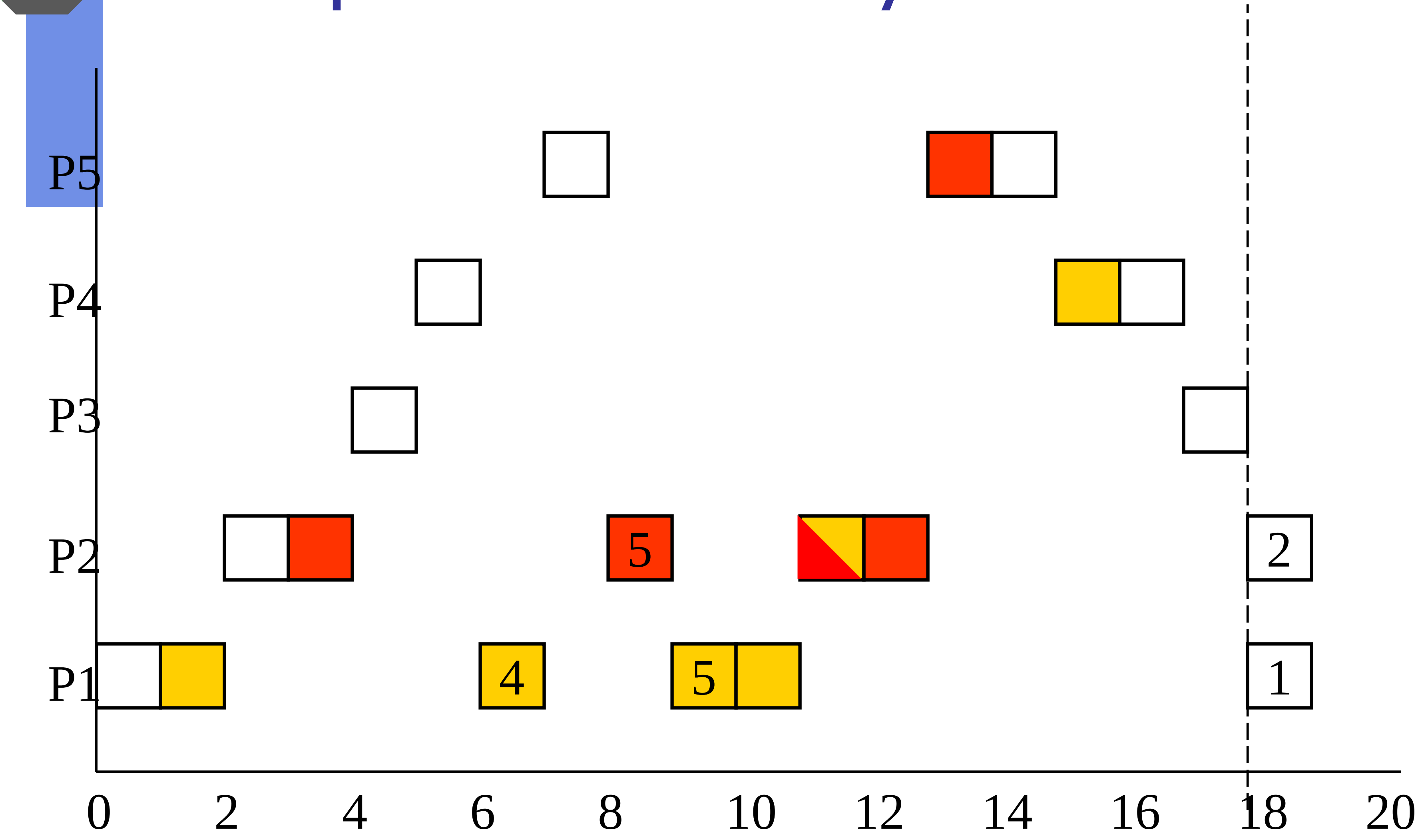
Example with Priority Inheritance



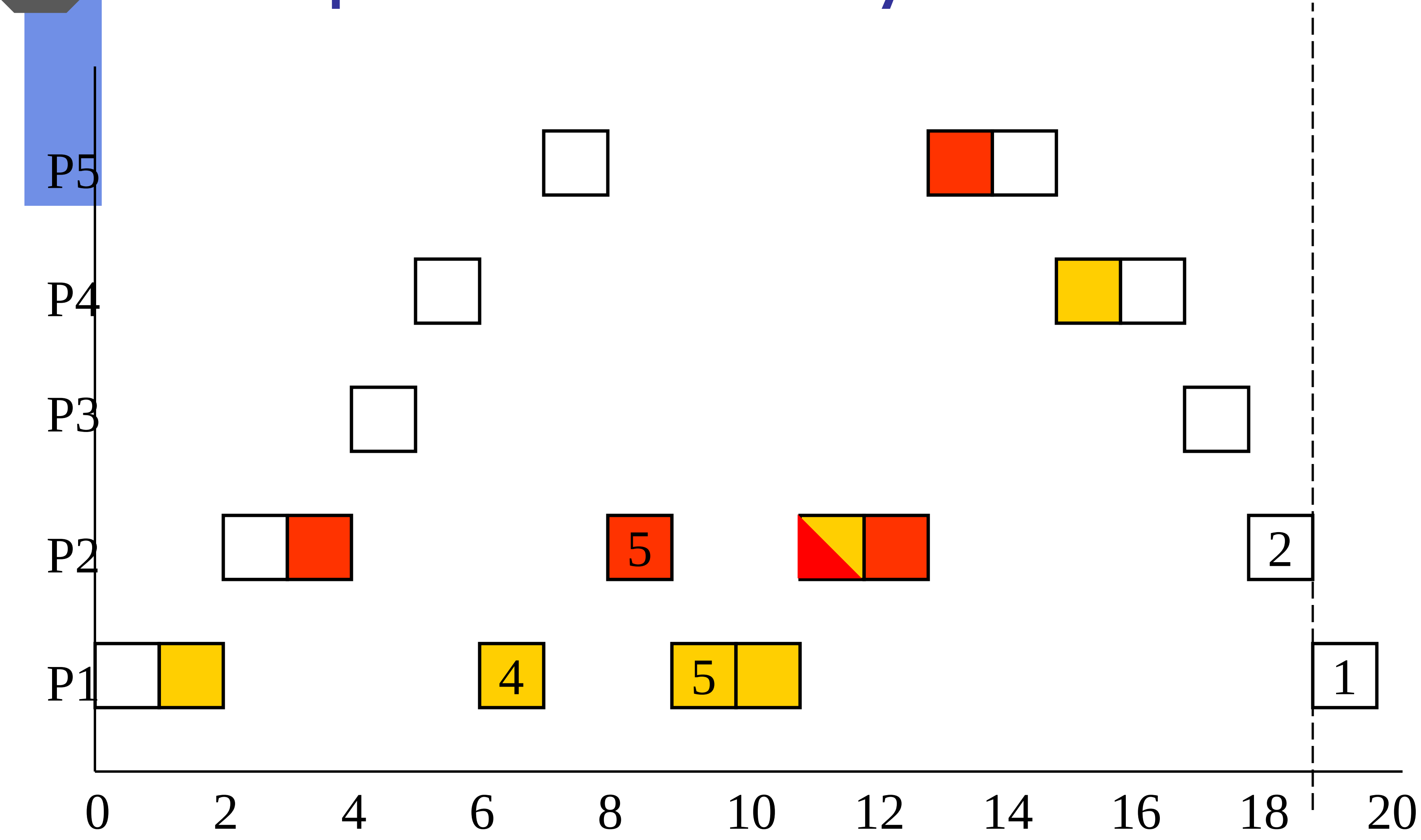
Example with Priority Inheritance



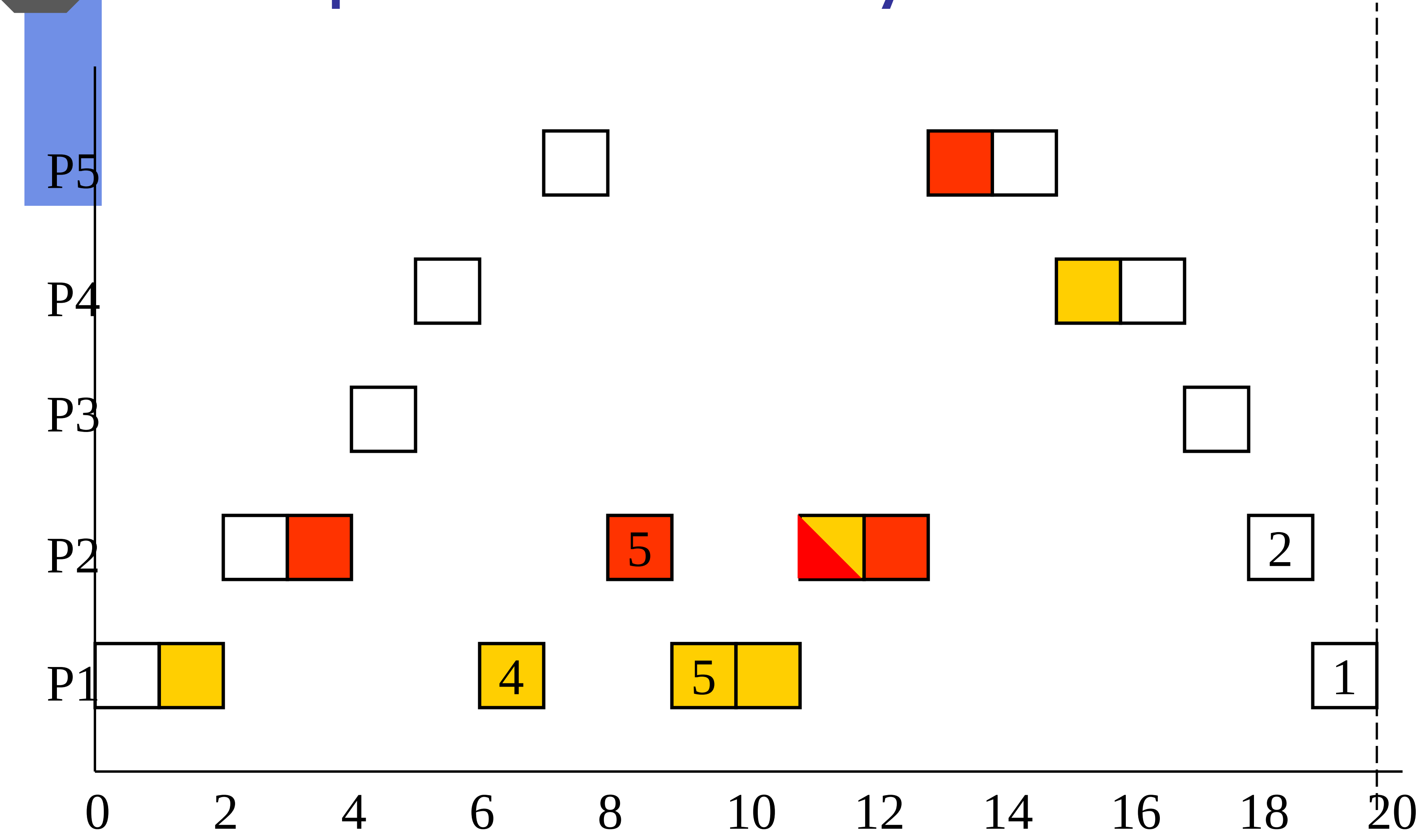
Example with Priority Inheritance



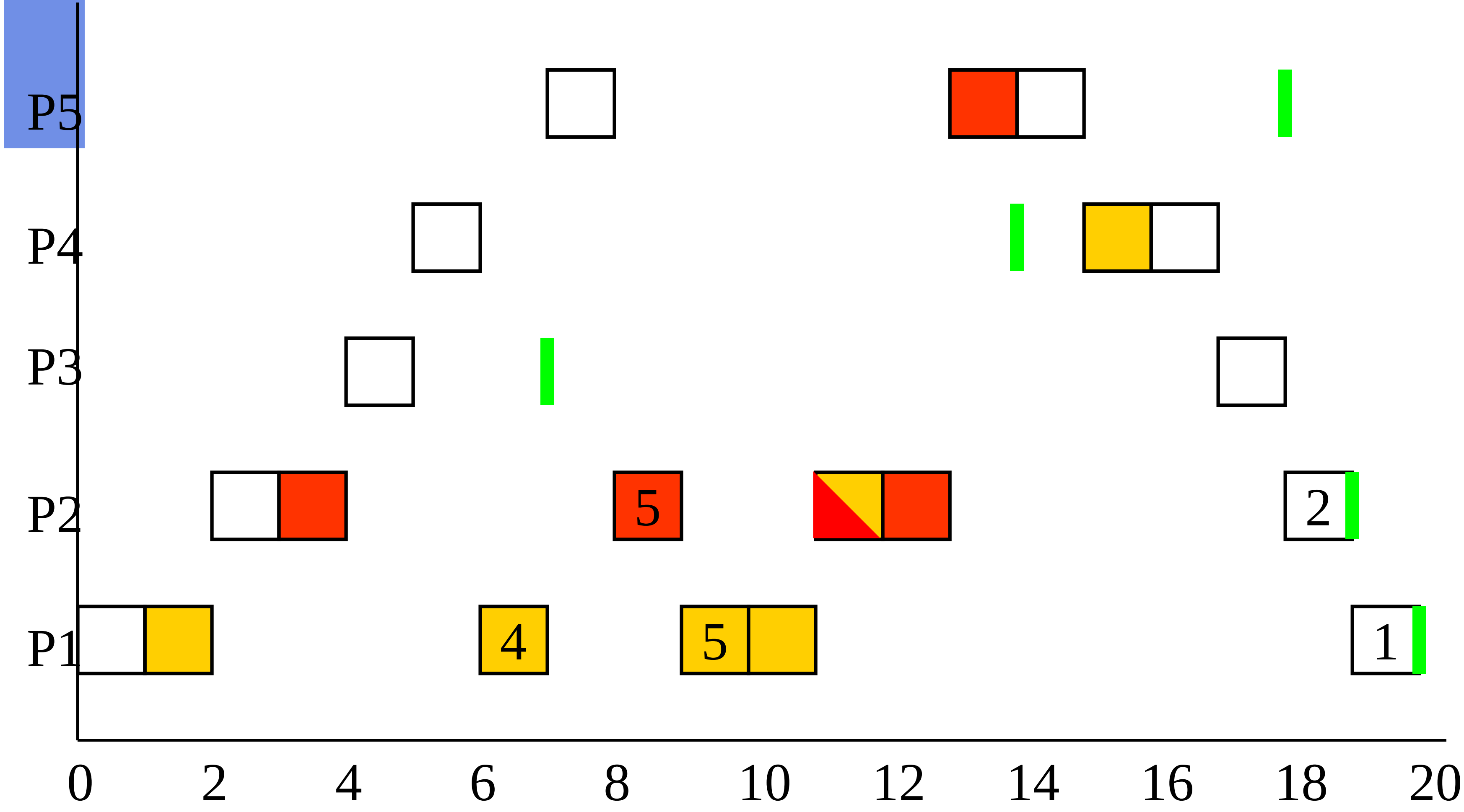
Example with Priority Inheritance



Example with Priority Inheritance



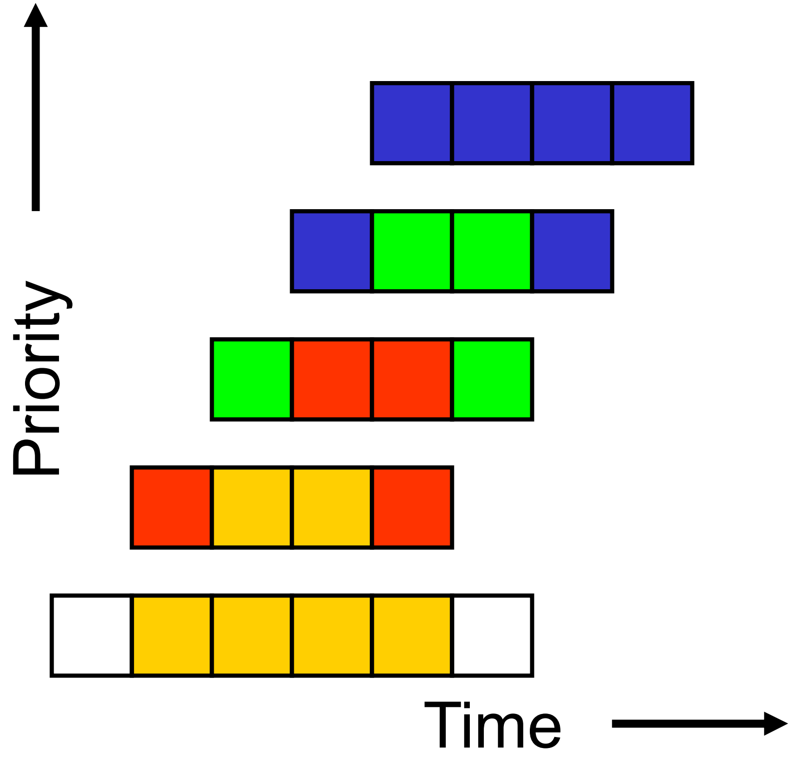
Comparison with SPD Rule



Analysis: Priority Inheritance

- Pros
  - Prevents uncontrolled priority inversion.
  - Needs no knowledge of resource requirements.
- Cons
  - Does *not prevent deadlock*.
  - Does not minimise blocking times.
    - With chained blocking, worst-case blocking time is min(n,m) critical sections
      - n = number of lower priority processes that can block P
      - m = number of resources that can be used to block P
  - Some overhead in a **release** or **acquire** operation

Chained Blocking



- 4 lower priority processes
- 4 potentially conflicting resources
- Worst-case blocking time = 16 units[1]

[1] Assume lower priority process allocates its first resource just before higher priority process runs

Priority Ceiling Protocol

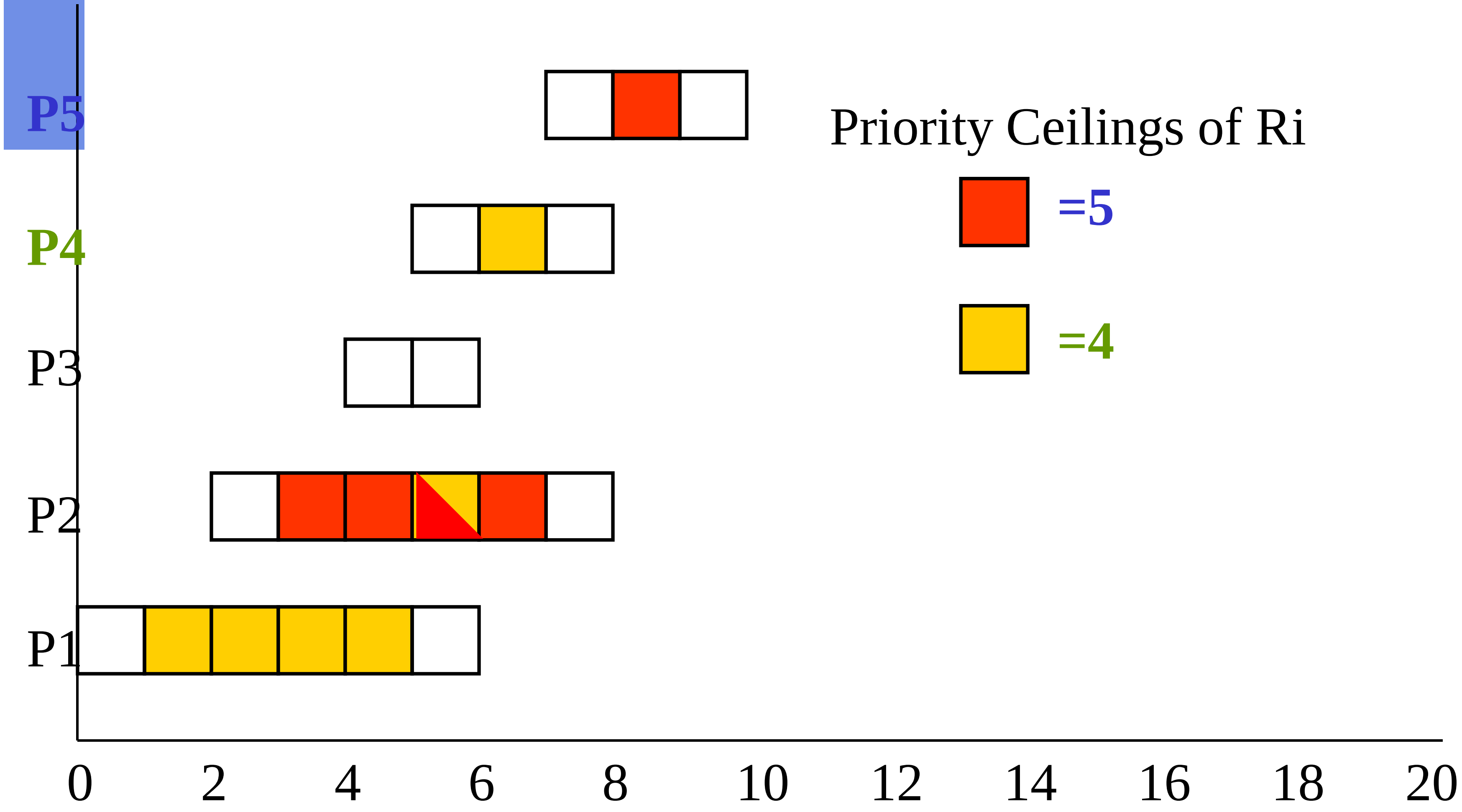
- Avoids deadlock by defining an order of resource acquisition
- Prevents transitive (chained) blocking
  - Worst-case blocking time = single critical section

Description how to implement PCP, see:
http://www.awprofessional.com/articles/article.asp?p=30188&seqNum=5&rl=1

Priority Ceilings

- Resources required by all processes are *known a priori*
  - Similar approach as with deadlock avoidance
- *Priority ceiling* of resource *R_i* is equal to the highest priority of all processes *that use* $\mathrm{R_i}$
- *Priority ceiling of system* is *highest priority ceiling* of all resources *currently in use*

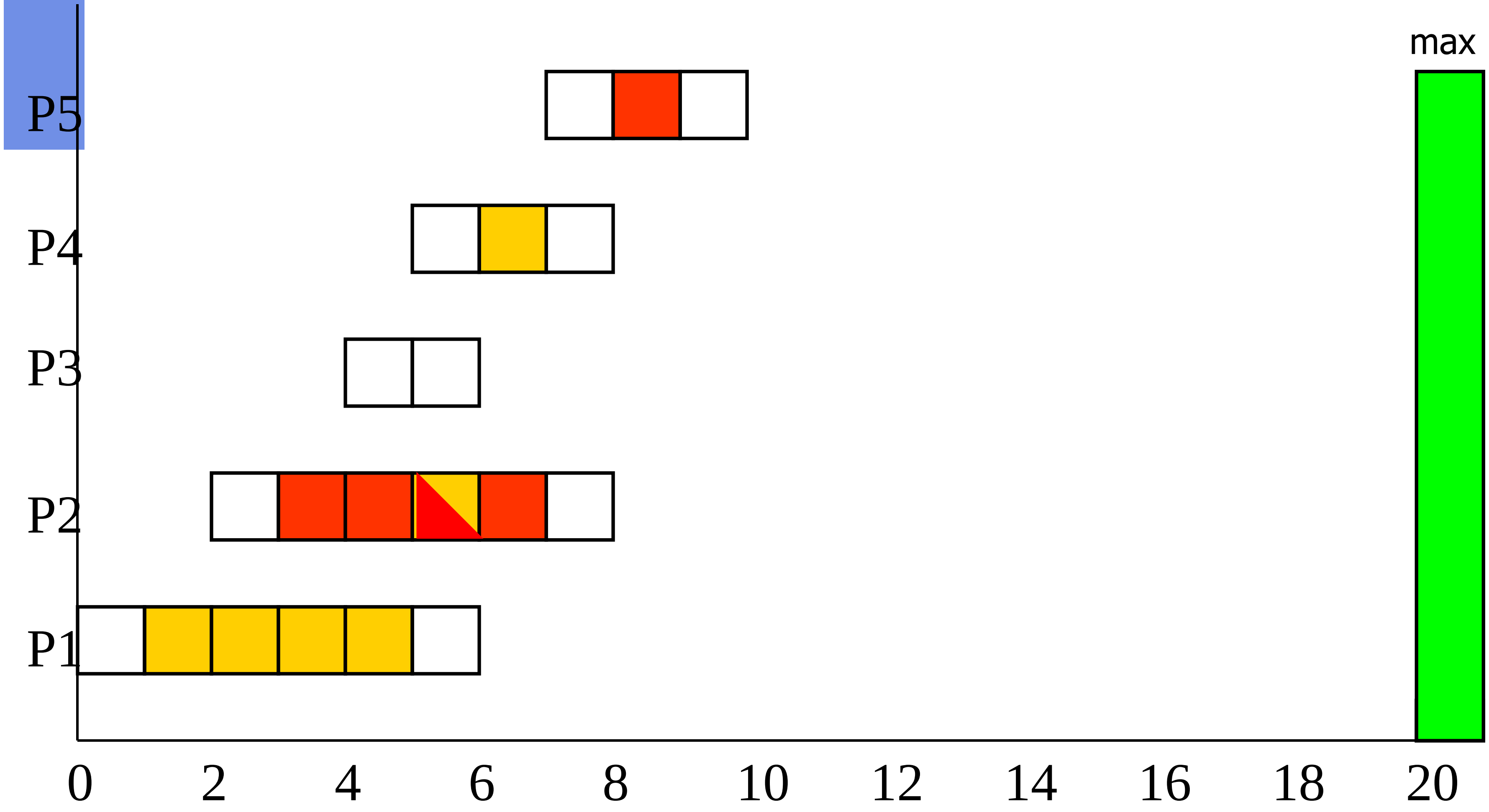
Priority Ceilings of Our Example



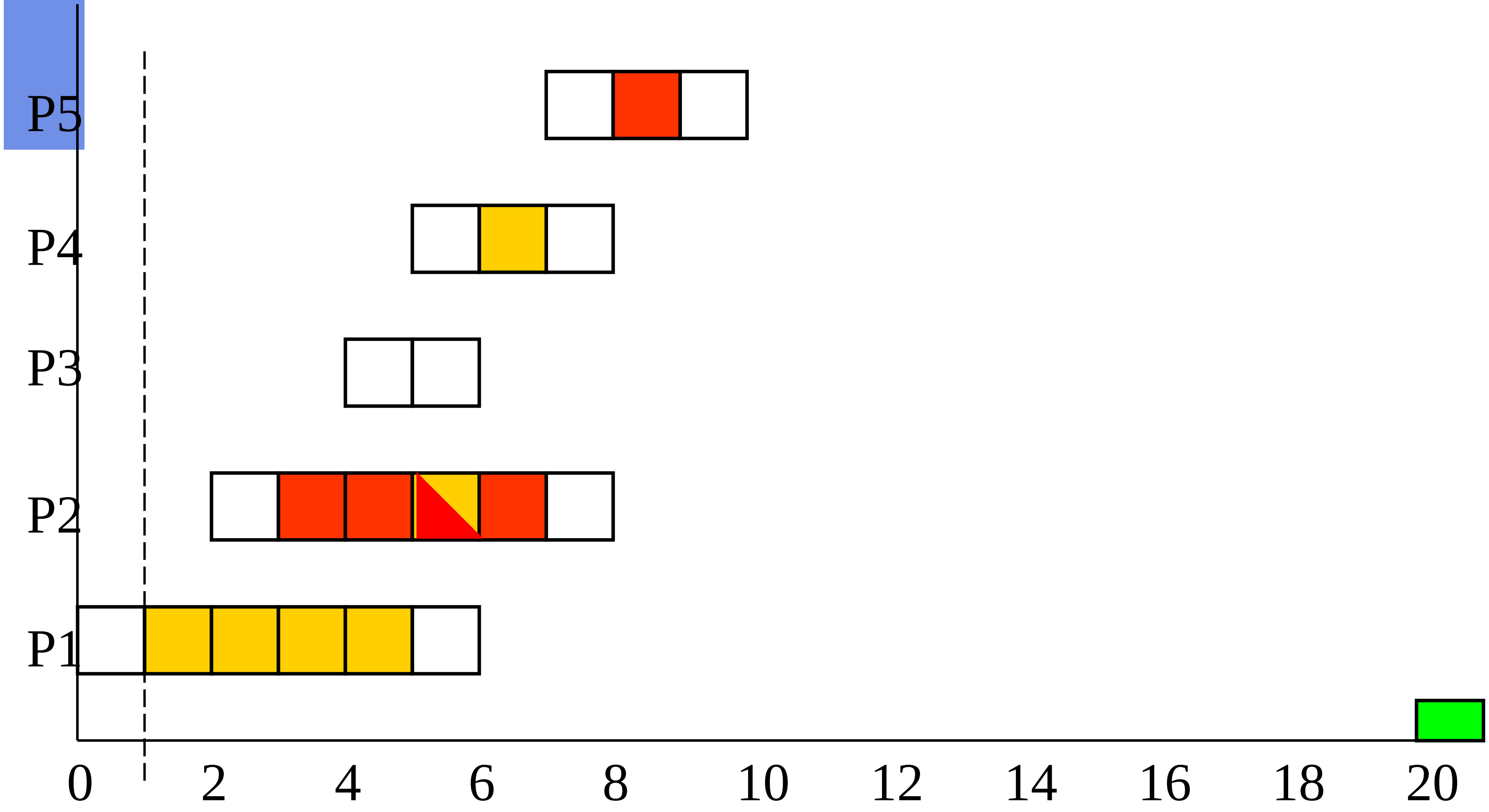
Priority Ceiling Protocol Rules

- Priority inheritance applies as before.
- When a process (P) requests a resource (R) either:
  - If R is *allocated* ⇒ P *blocks (+ priority inheritance)*
  - If R is free,
    - If P's *current priority* > *system's priority ceiling* ⇒ R is allocated to process P
    - If P's current priority ≤ system's priority ceiling ⇒ P *blocks* – except if:
      - P already holds a resource whose priority ceiling is equal to the systems priority ceiling

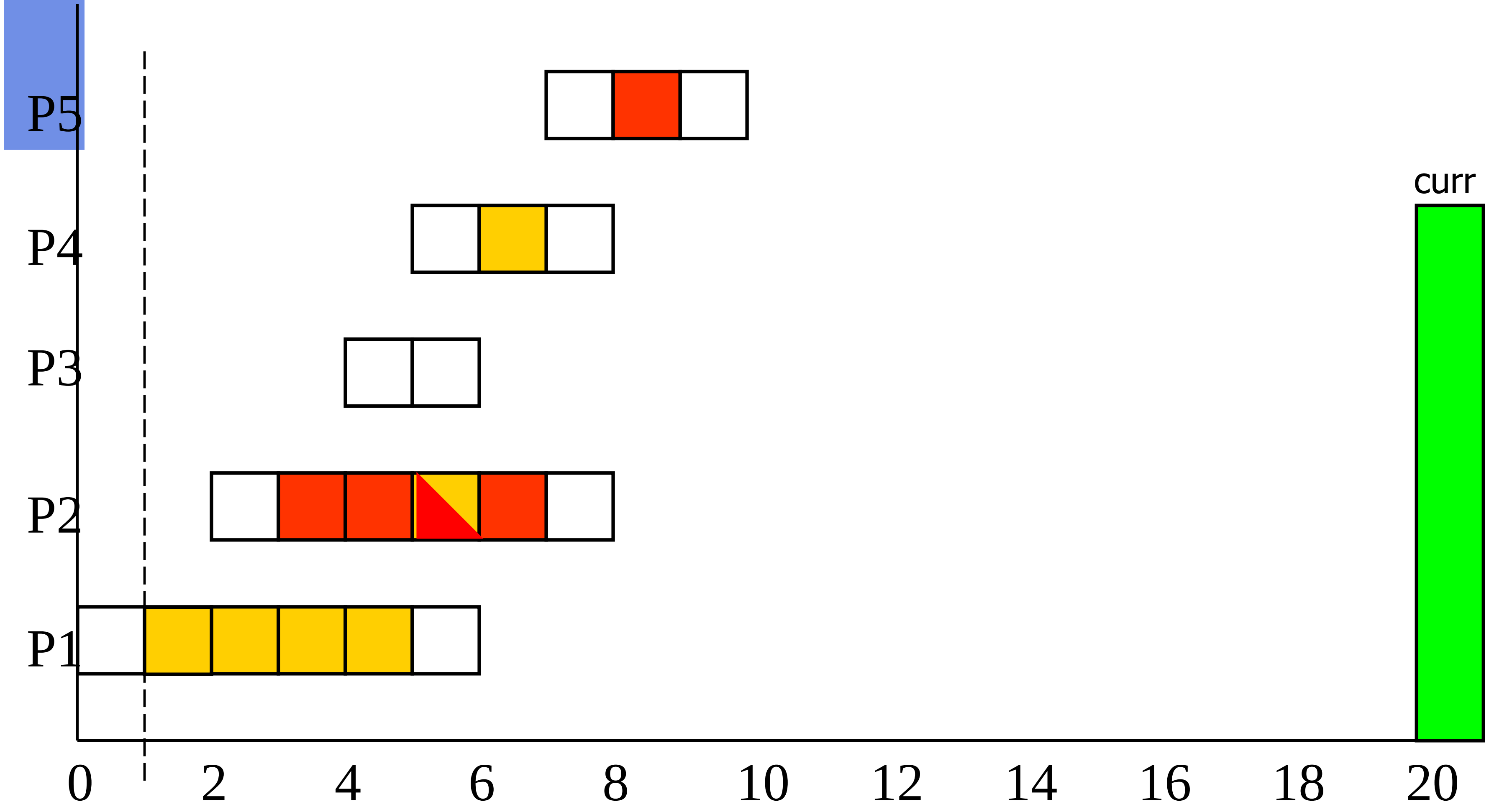
Example



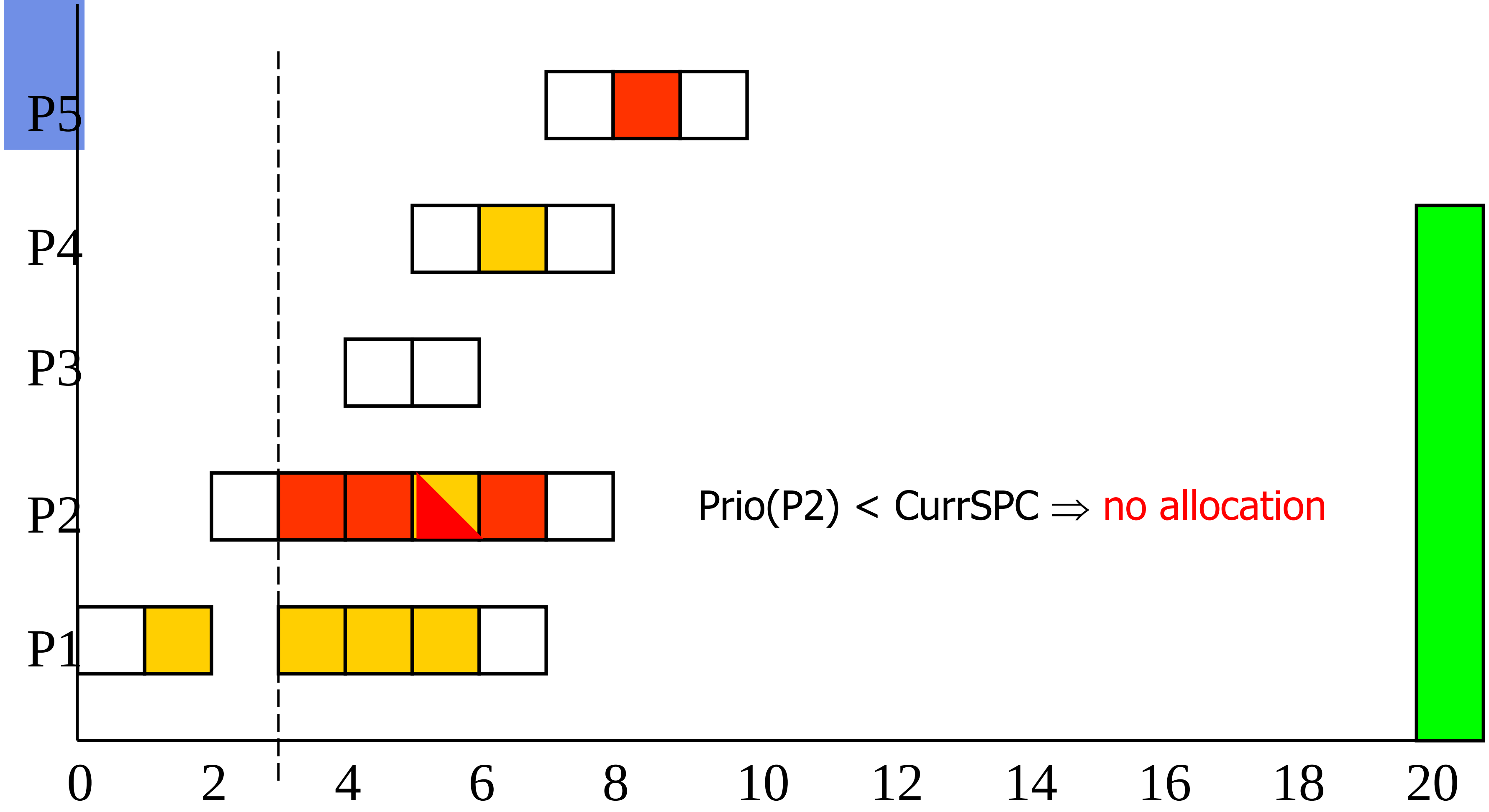
Example



Example

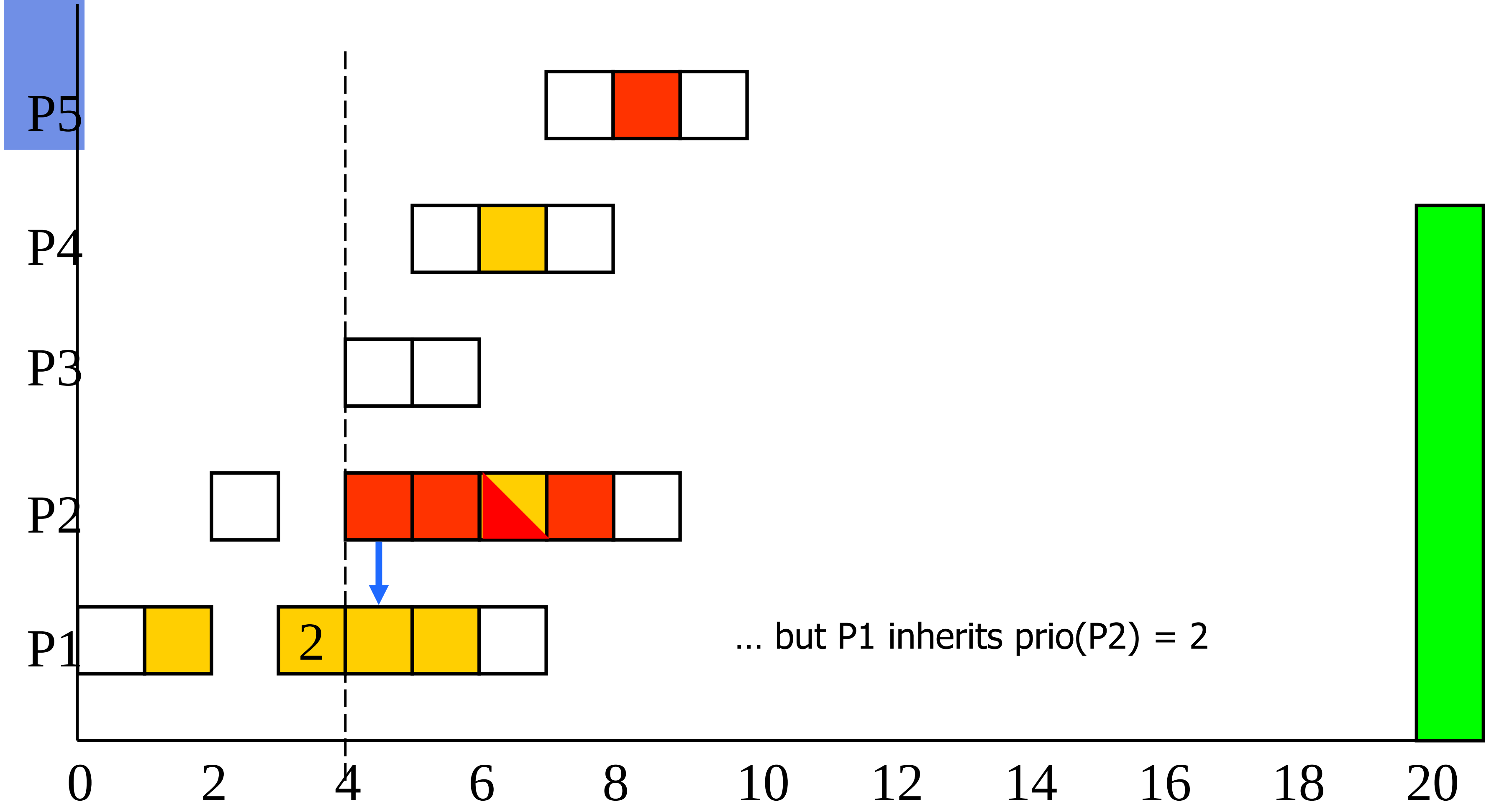


Example



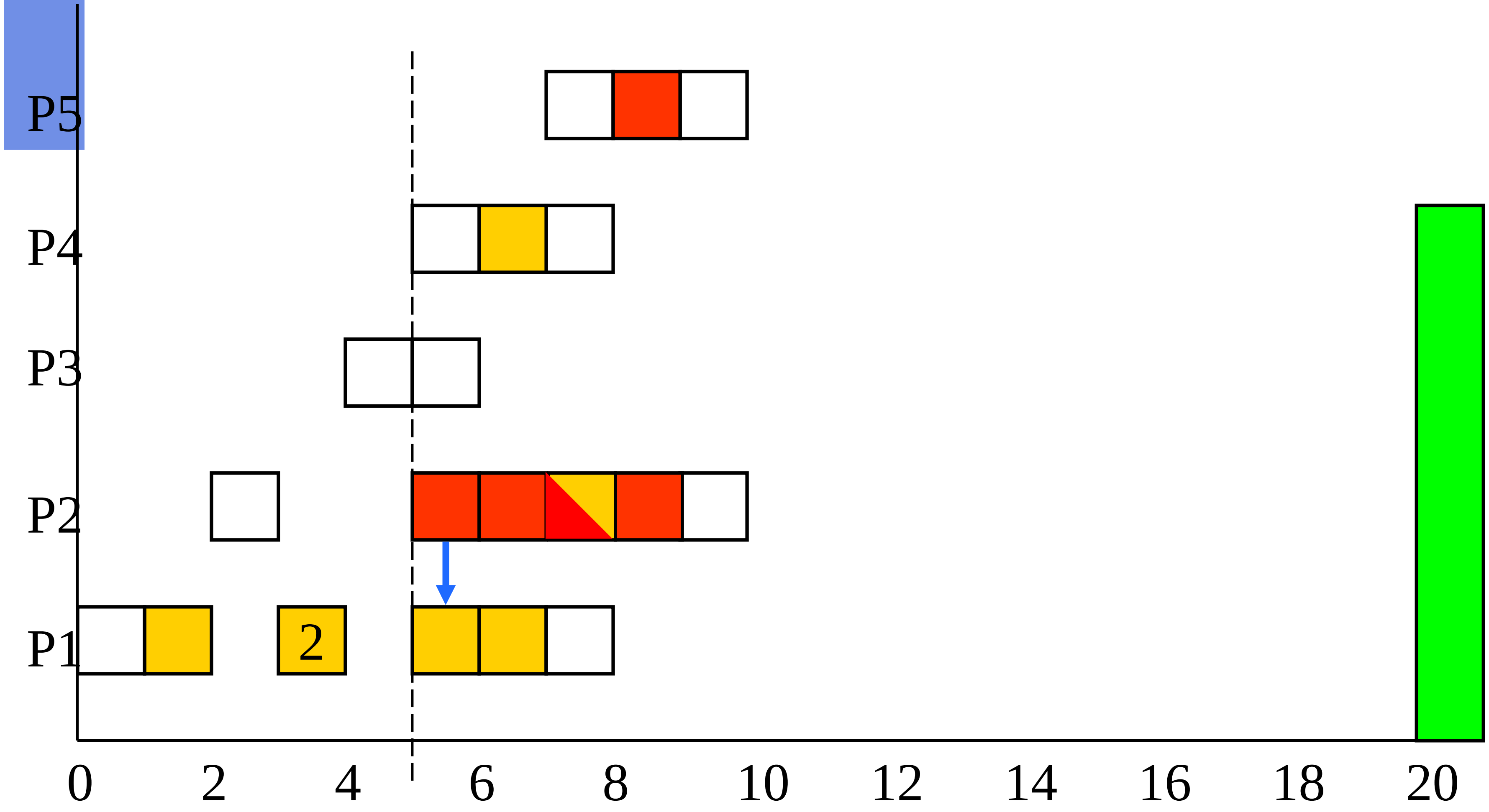
Prio(P2) < CurrSPC ⇒ no allocation

Example

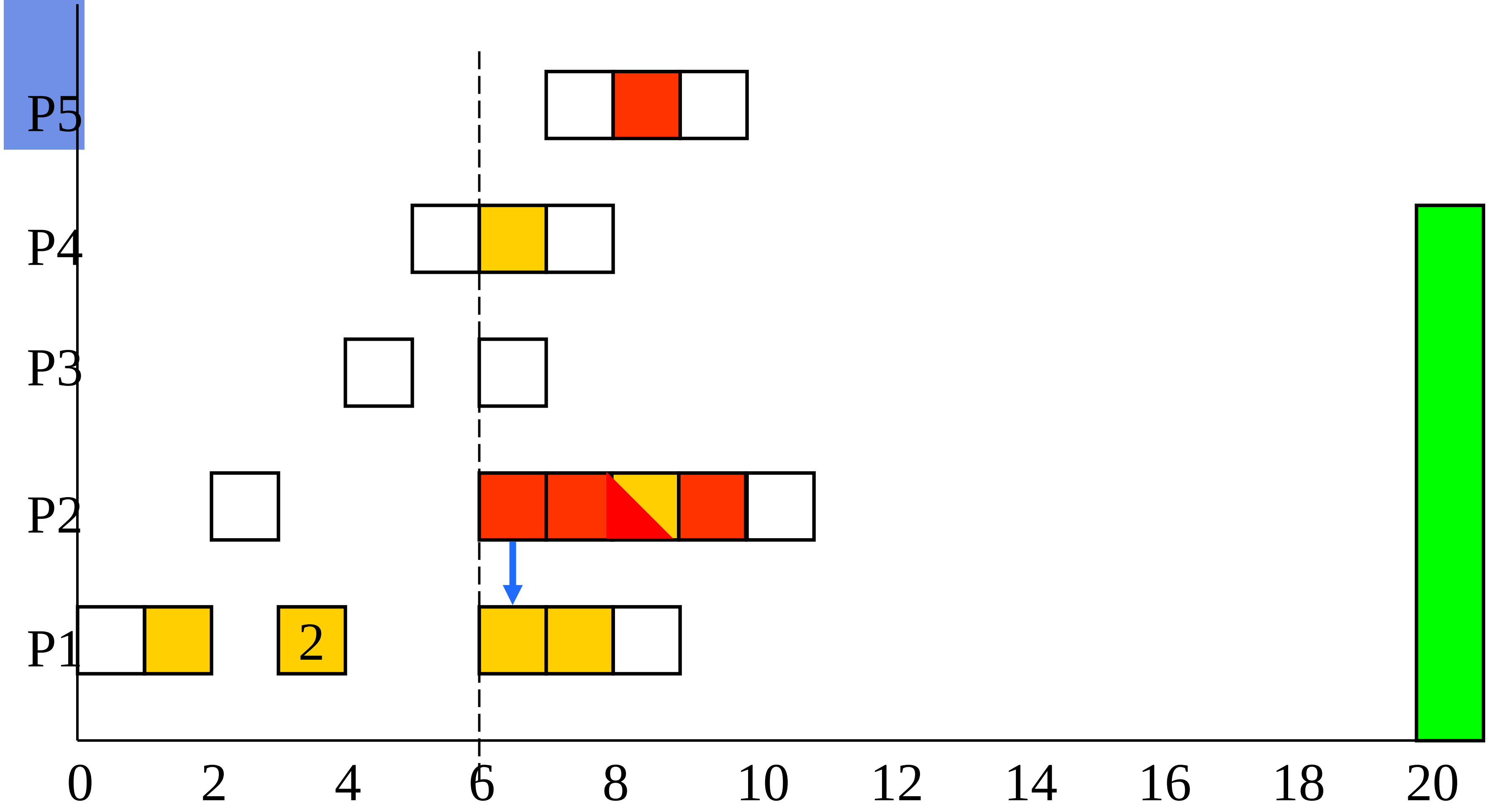


... but P1 inherits prio(P2) = 2

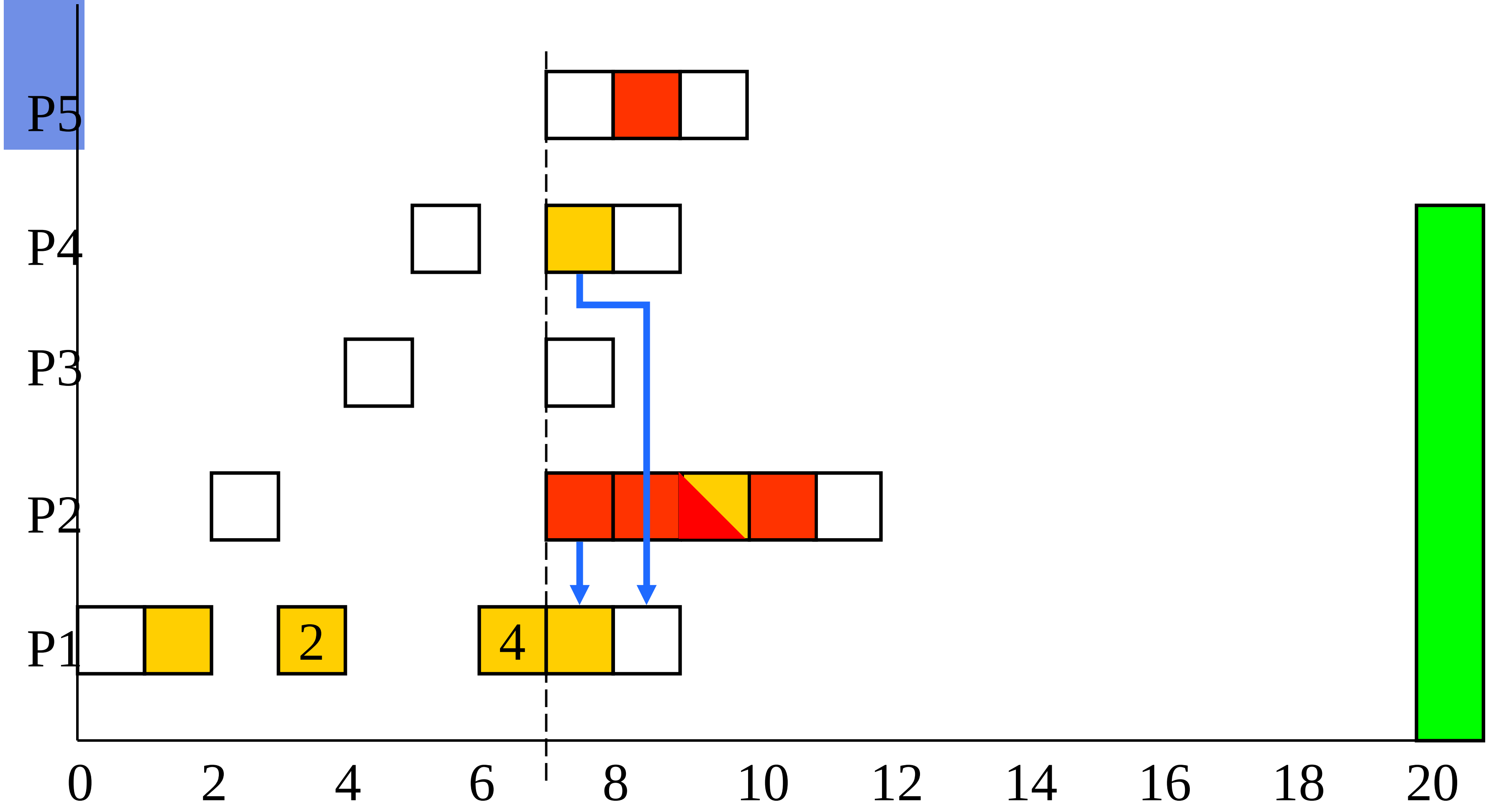
Example



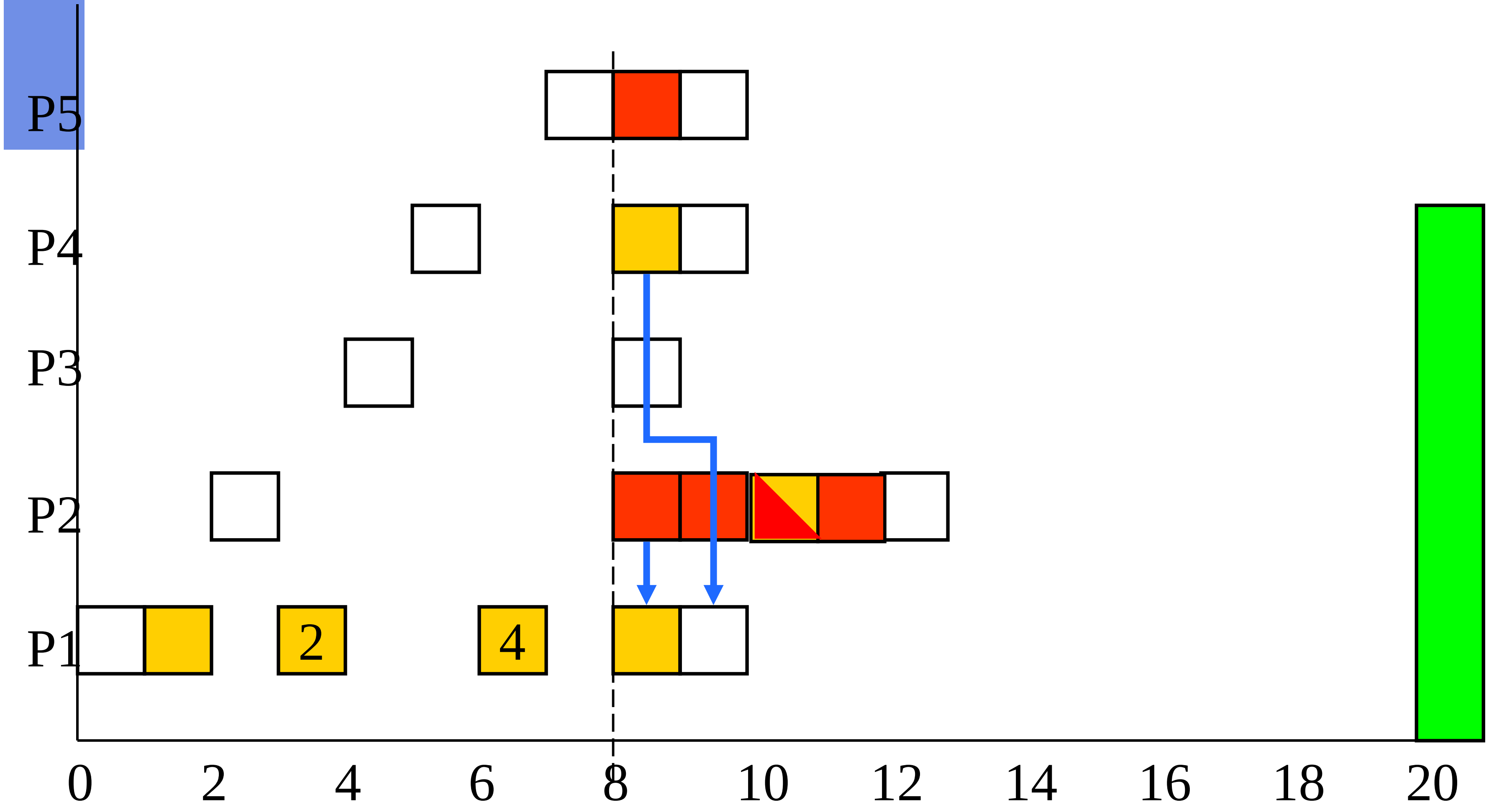
Example



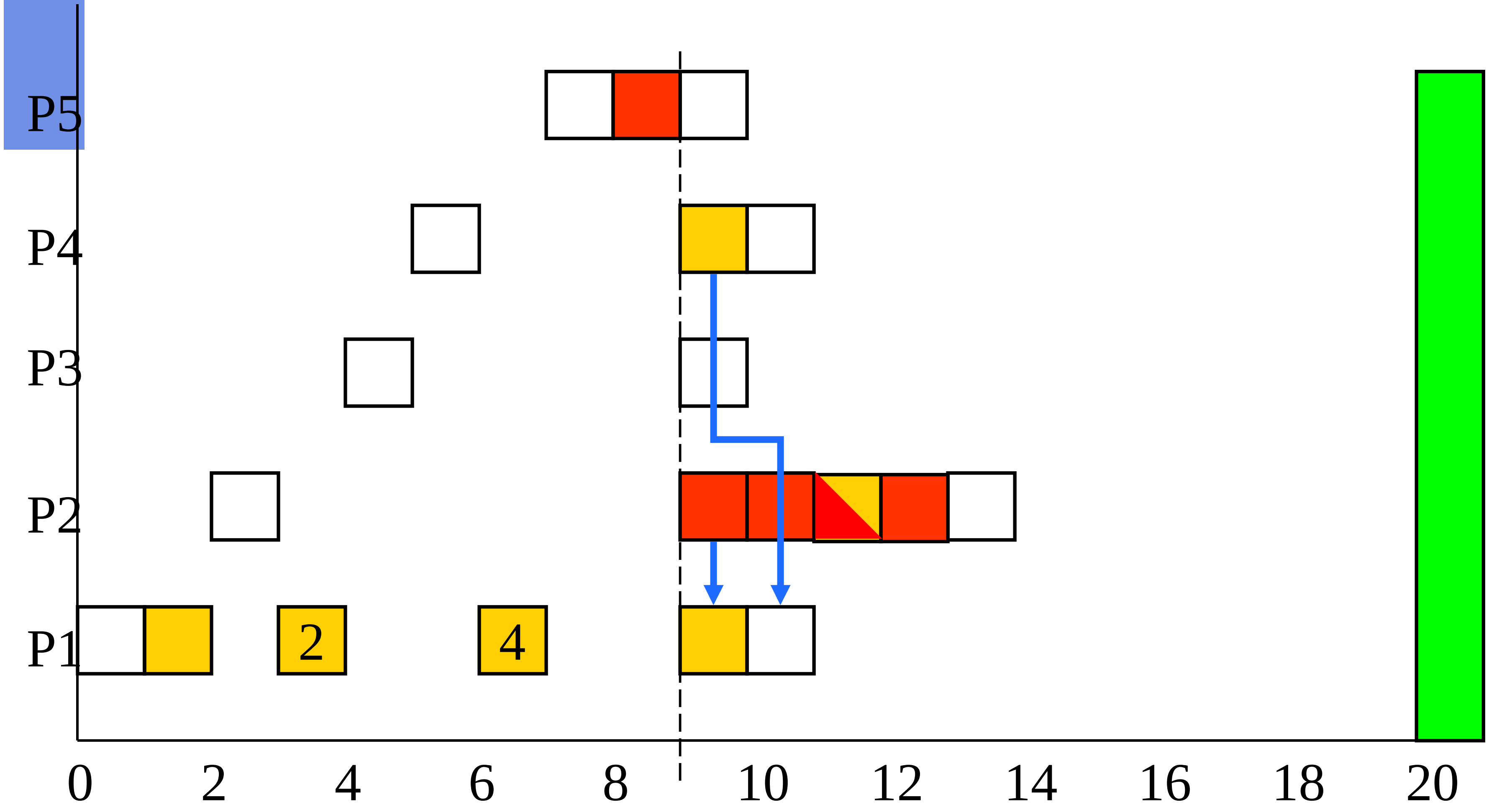
Example



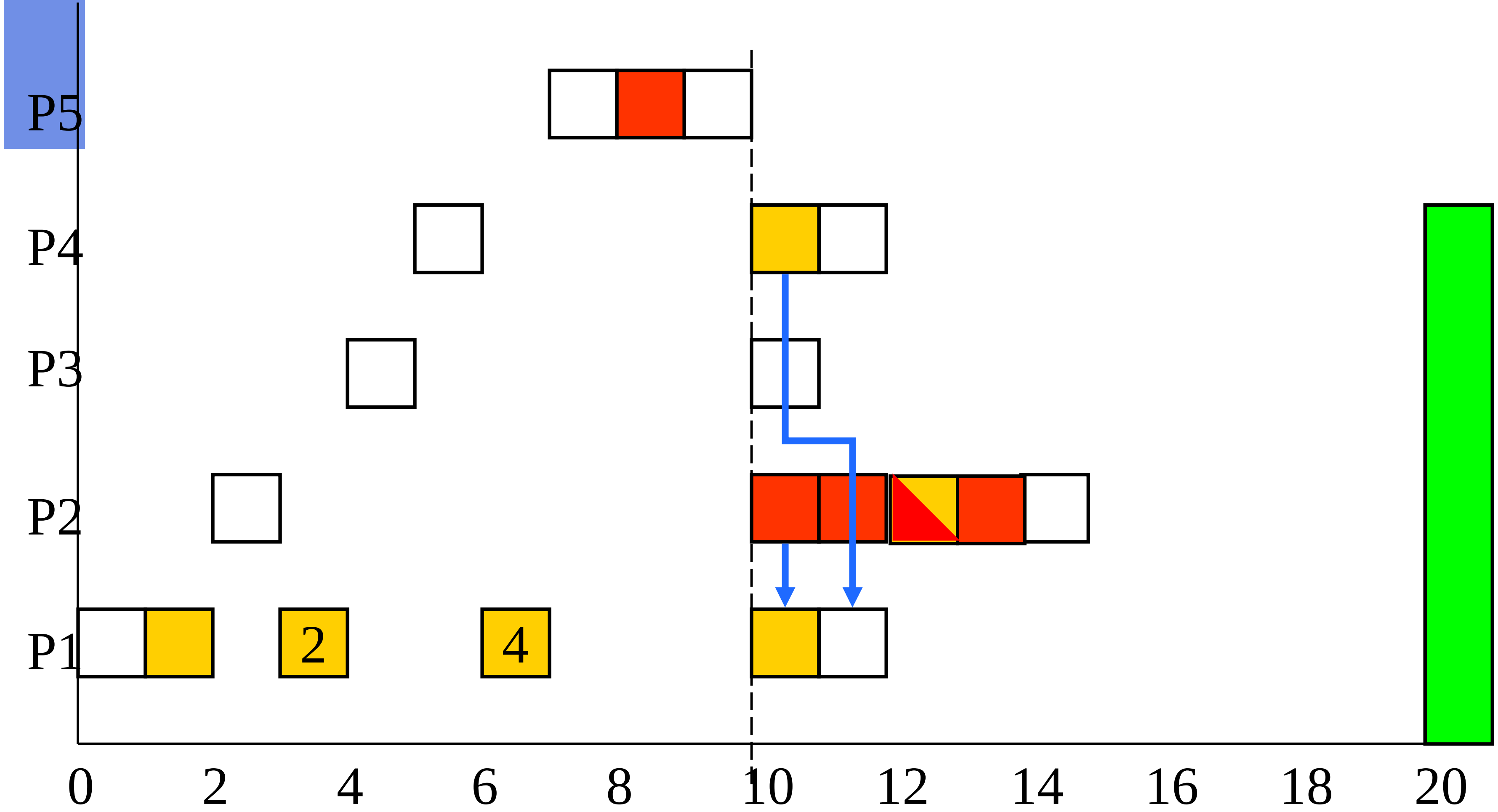
Example



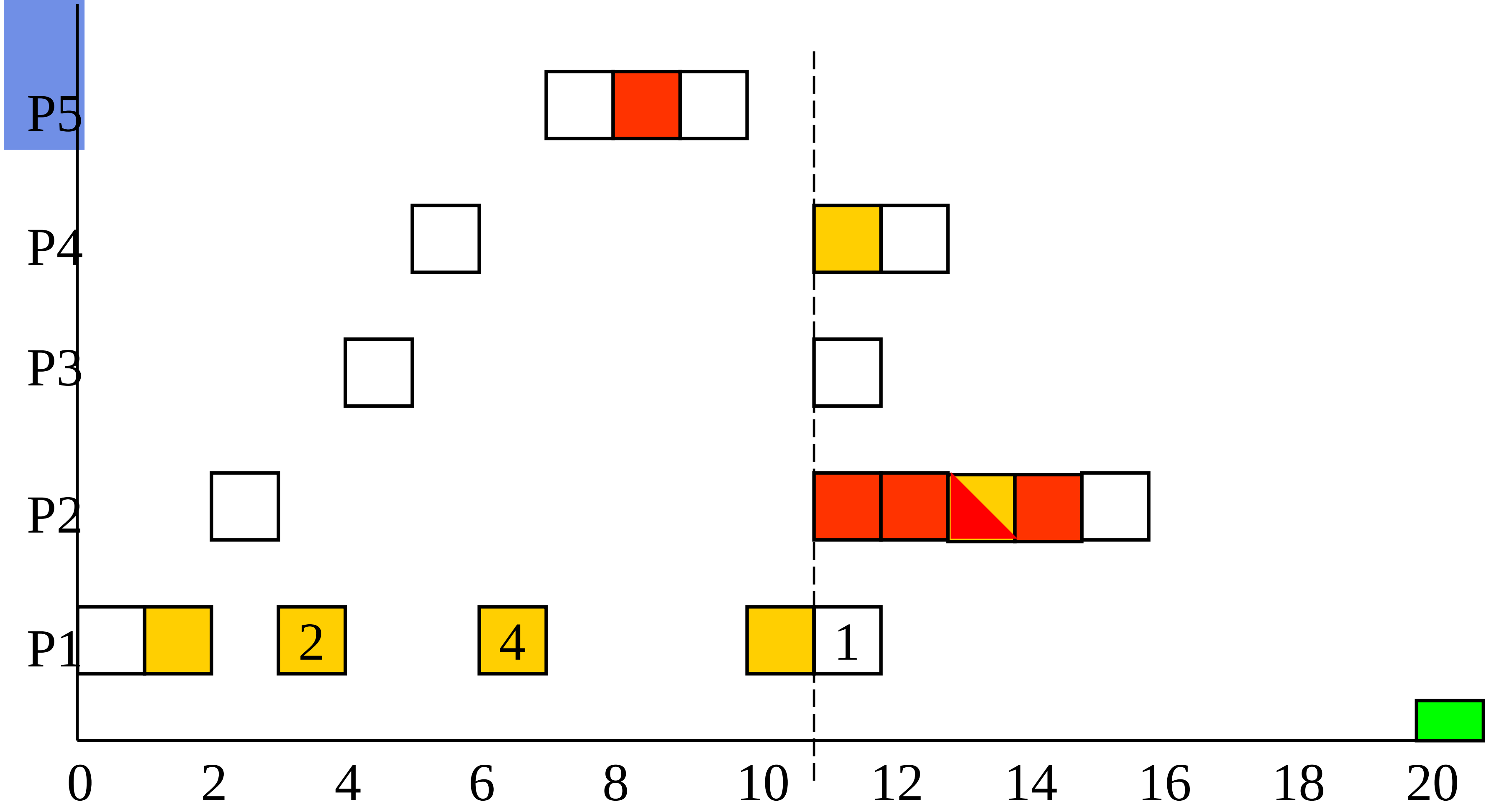
Example



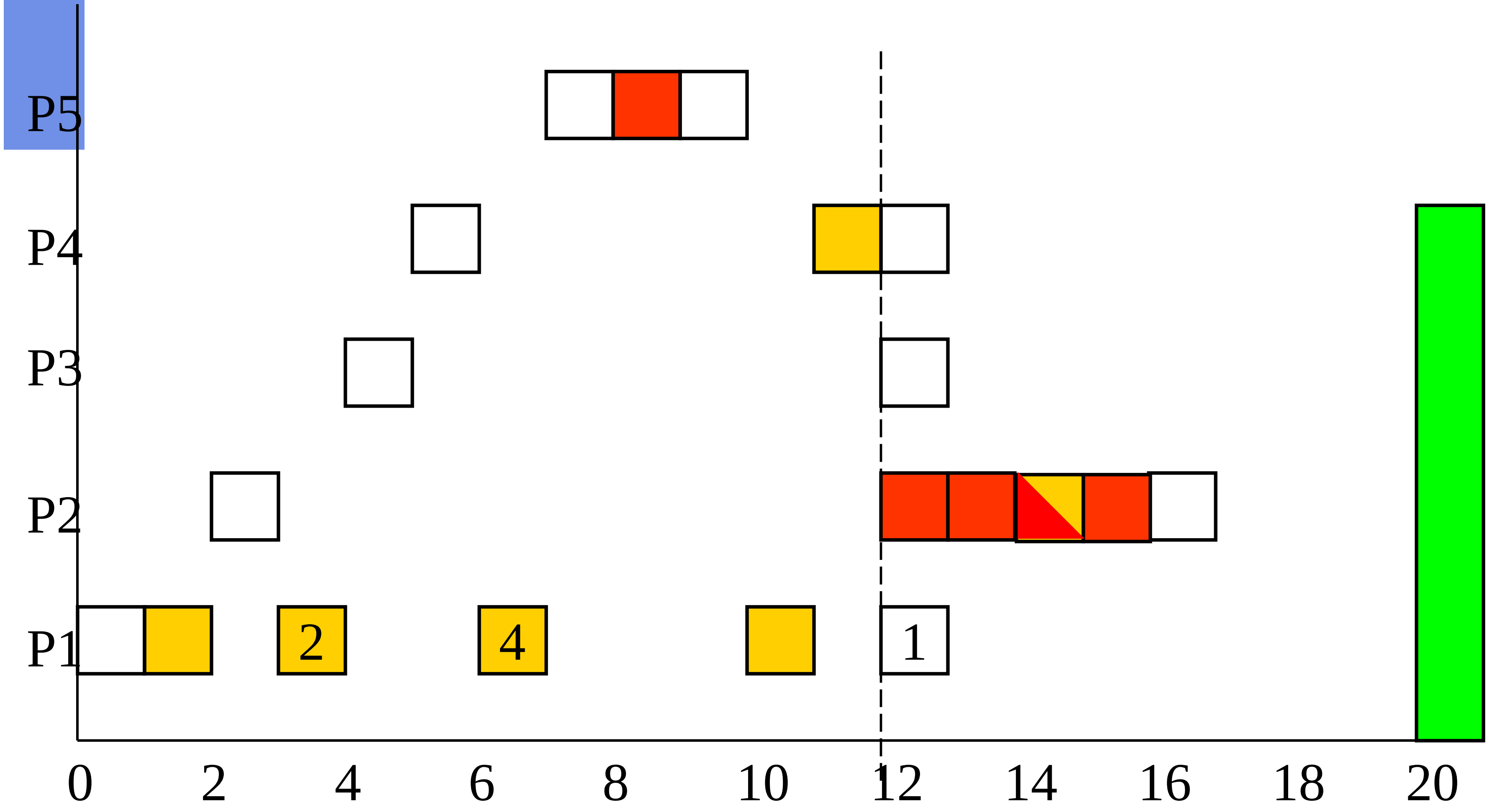
Example



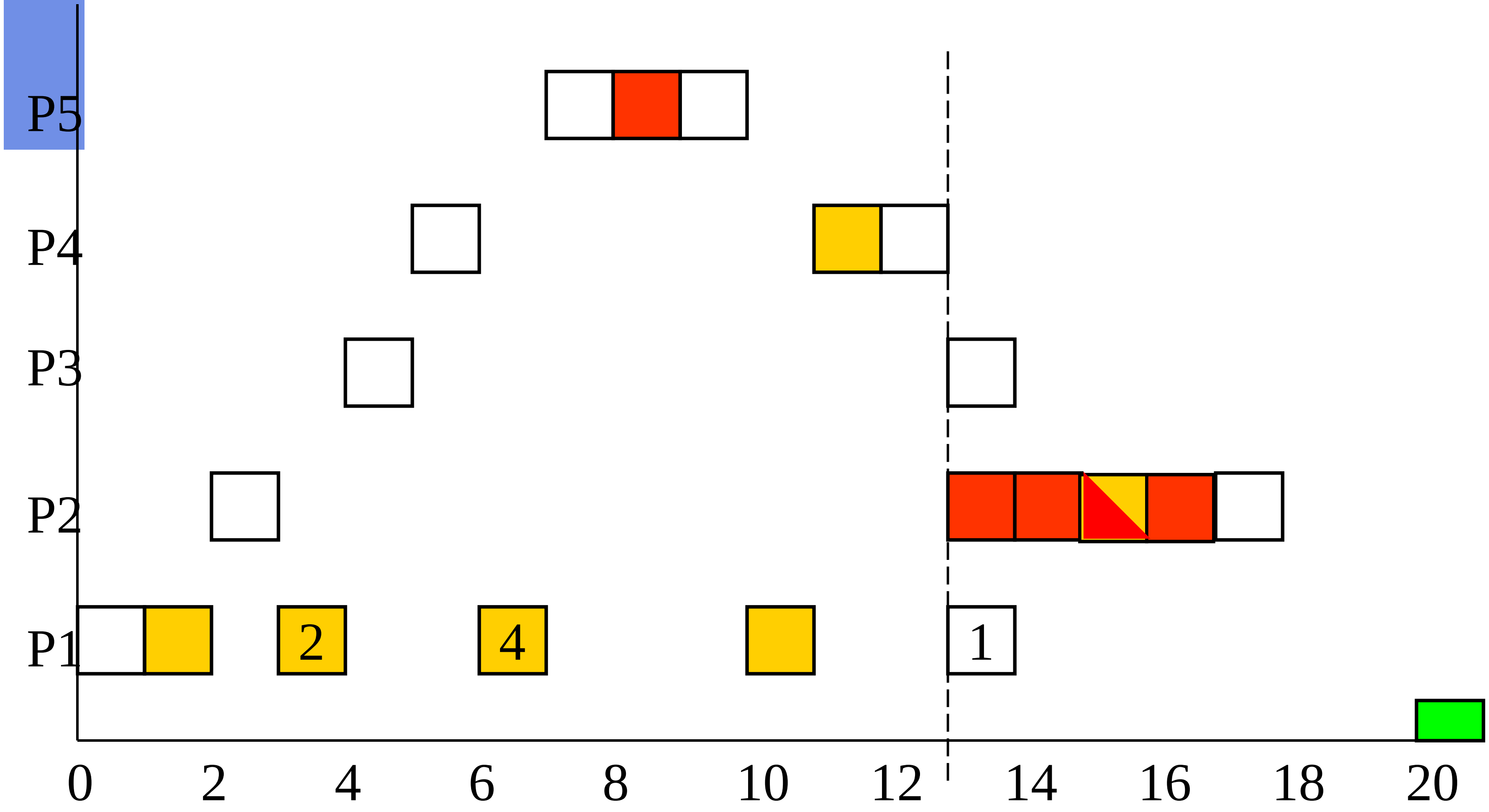
Example



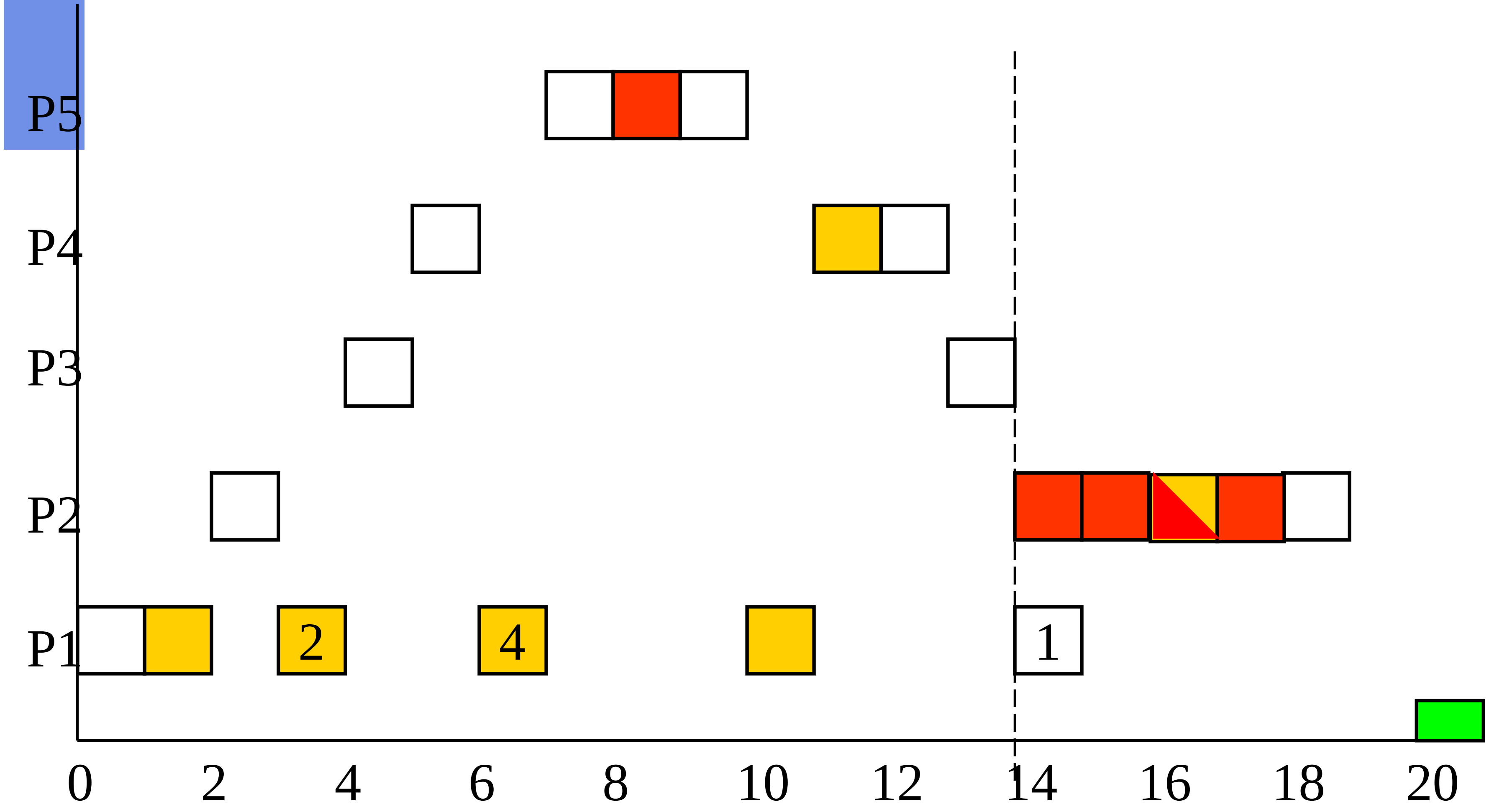
Example



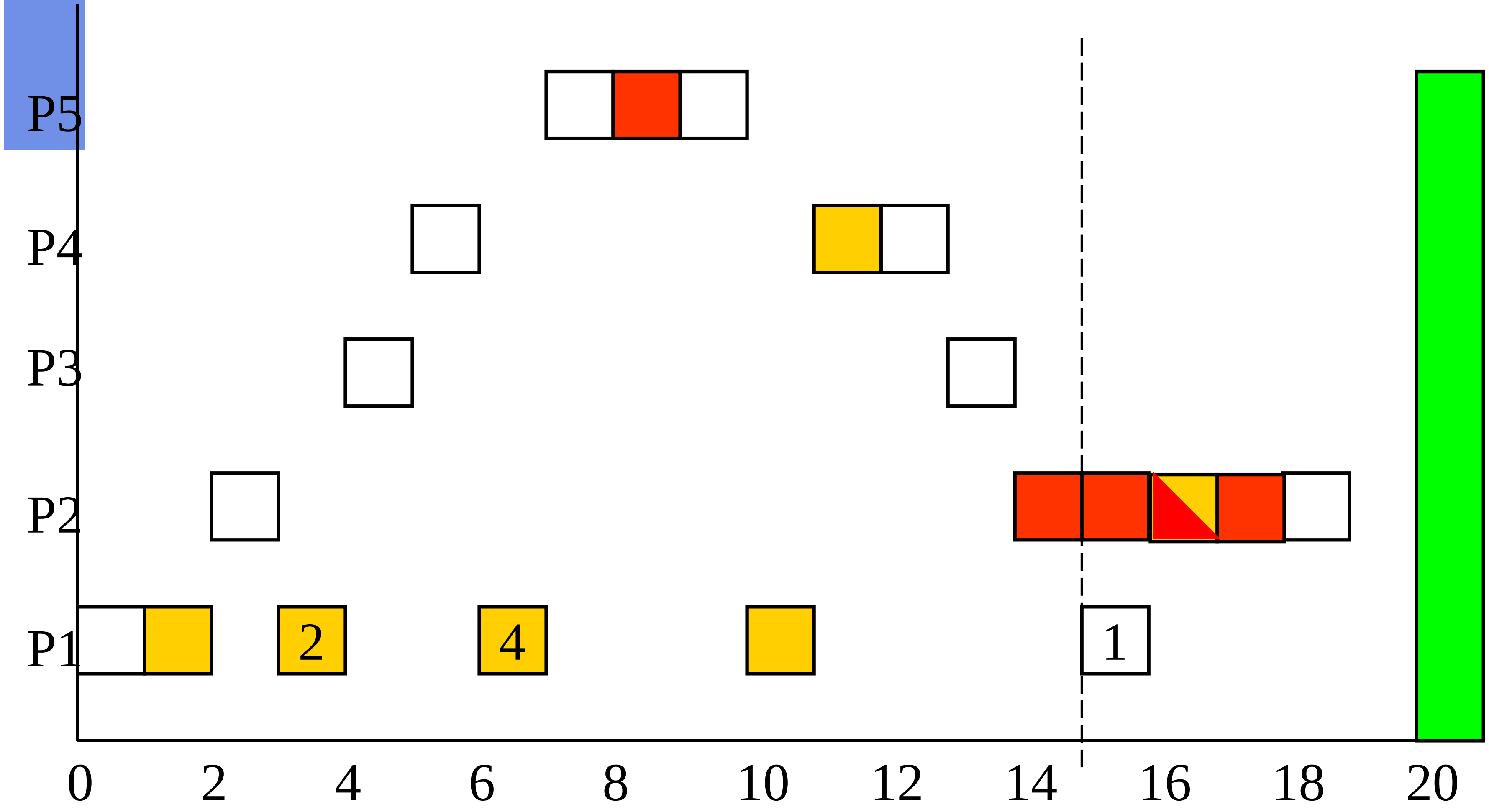
Example



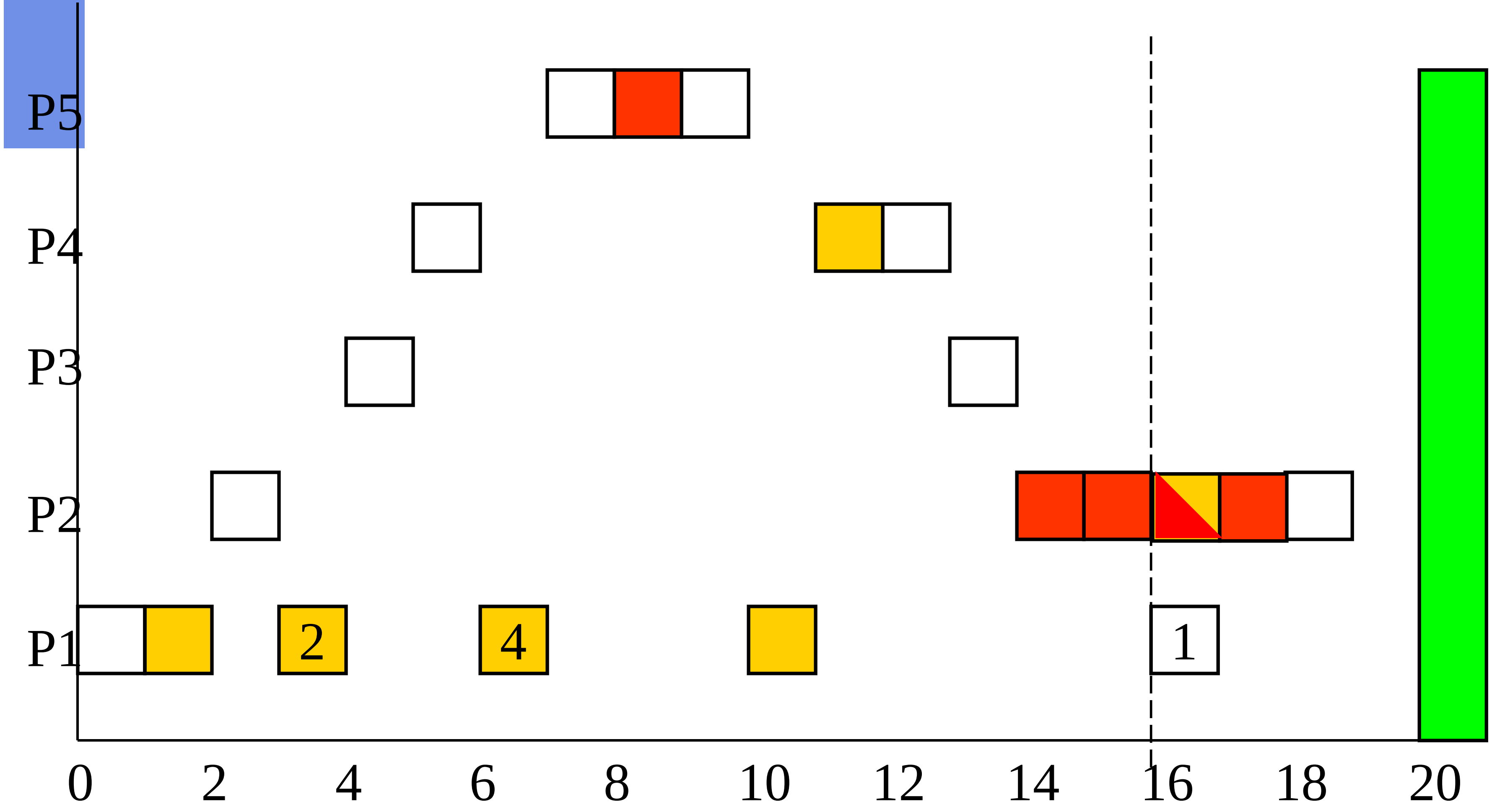
Example



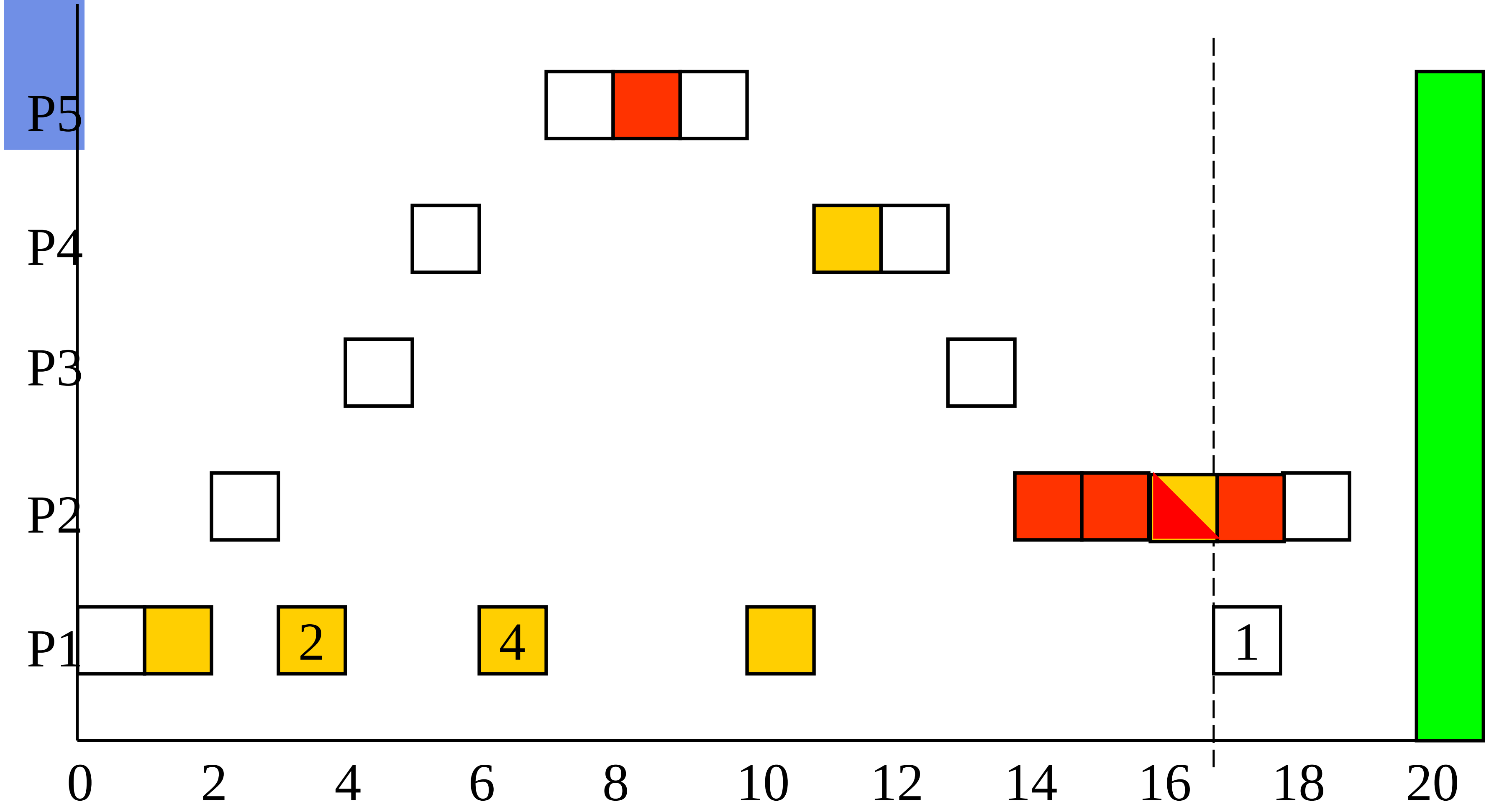
Example



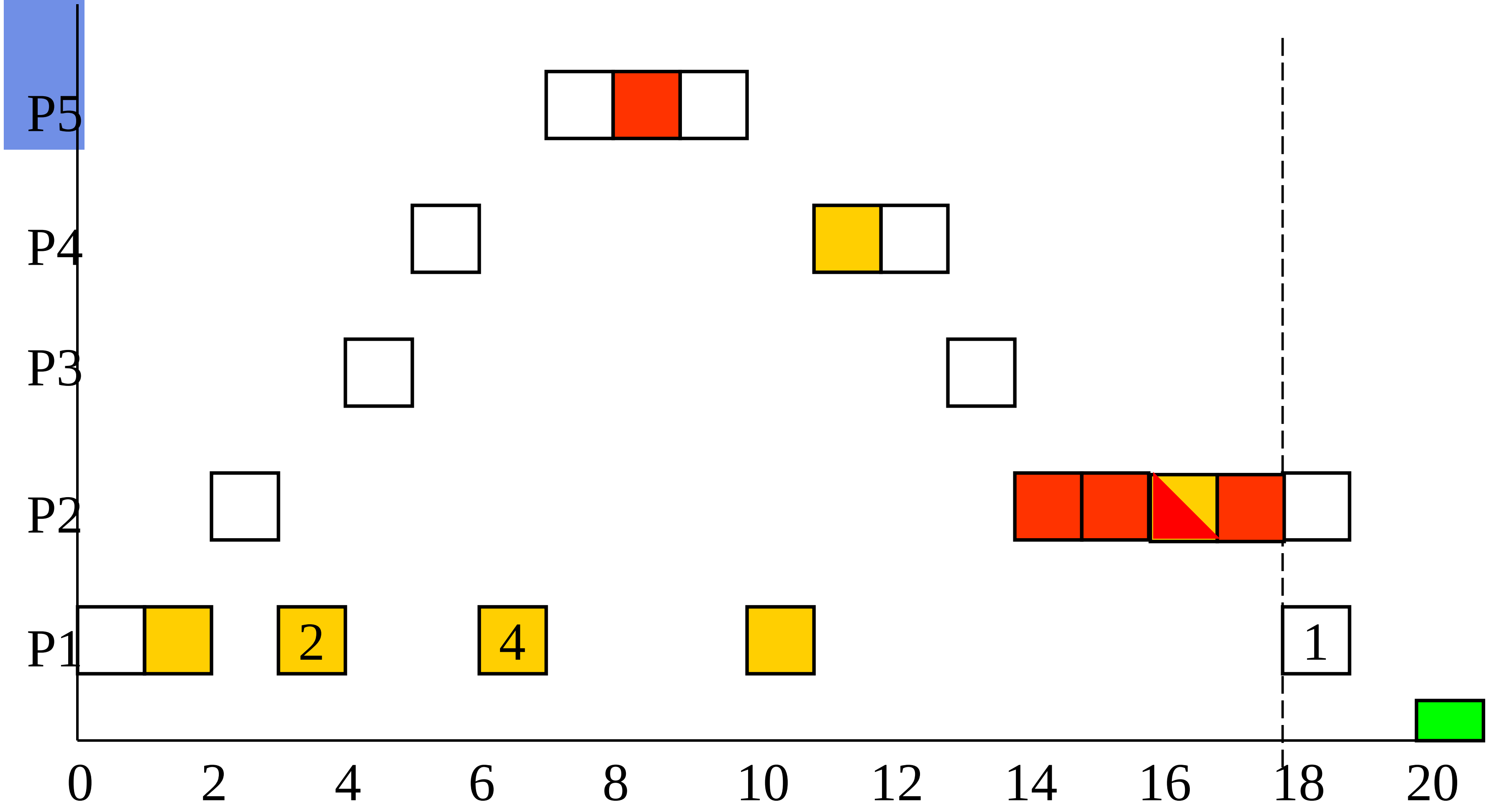
Example



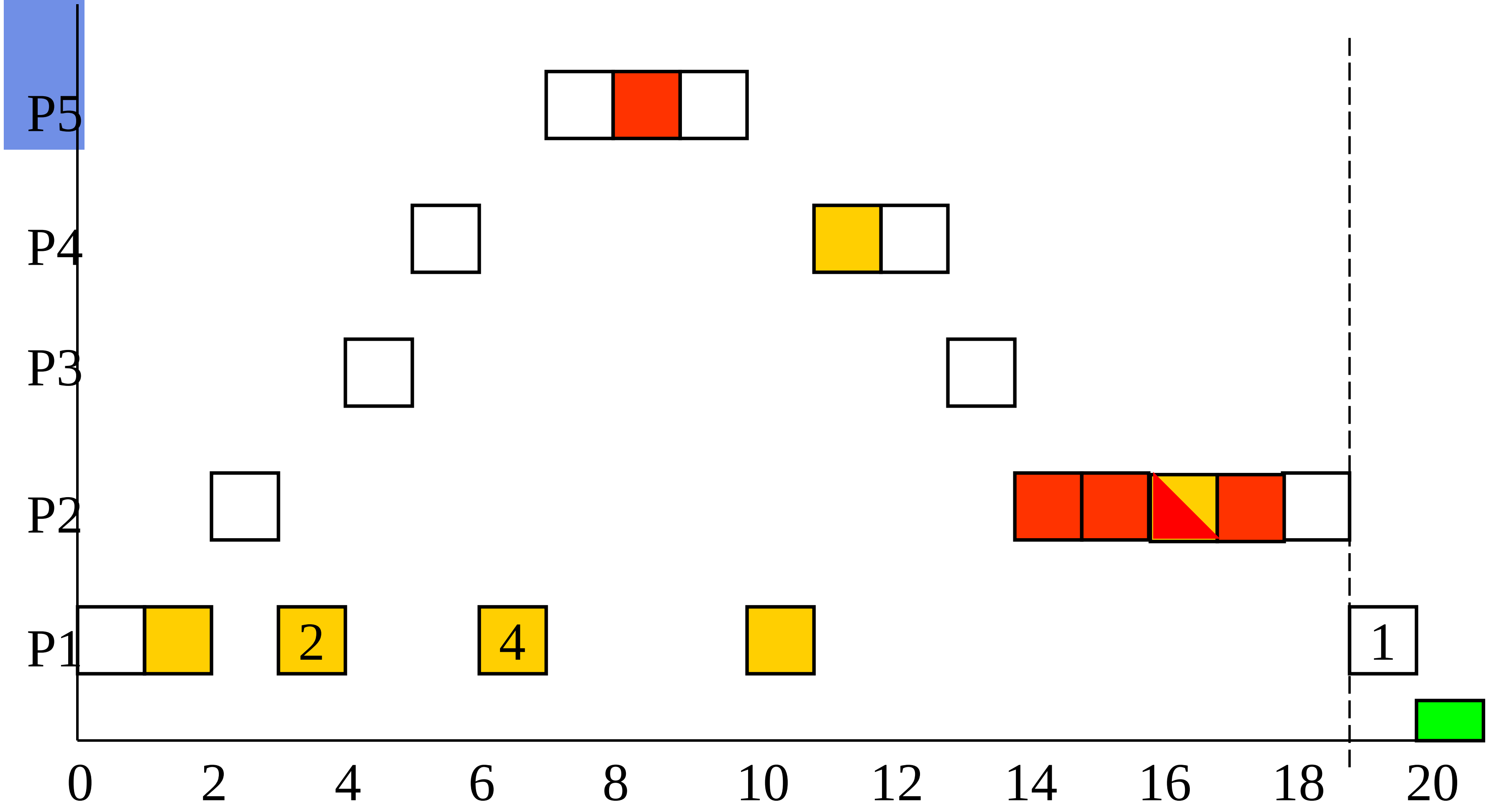
Example



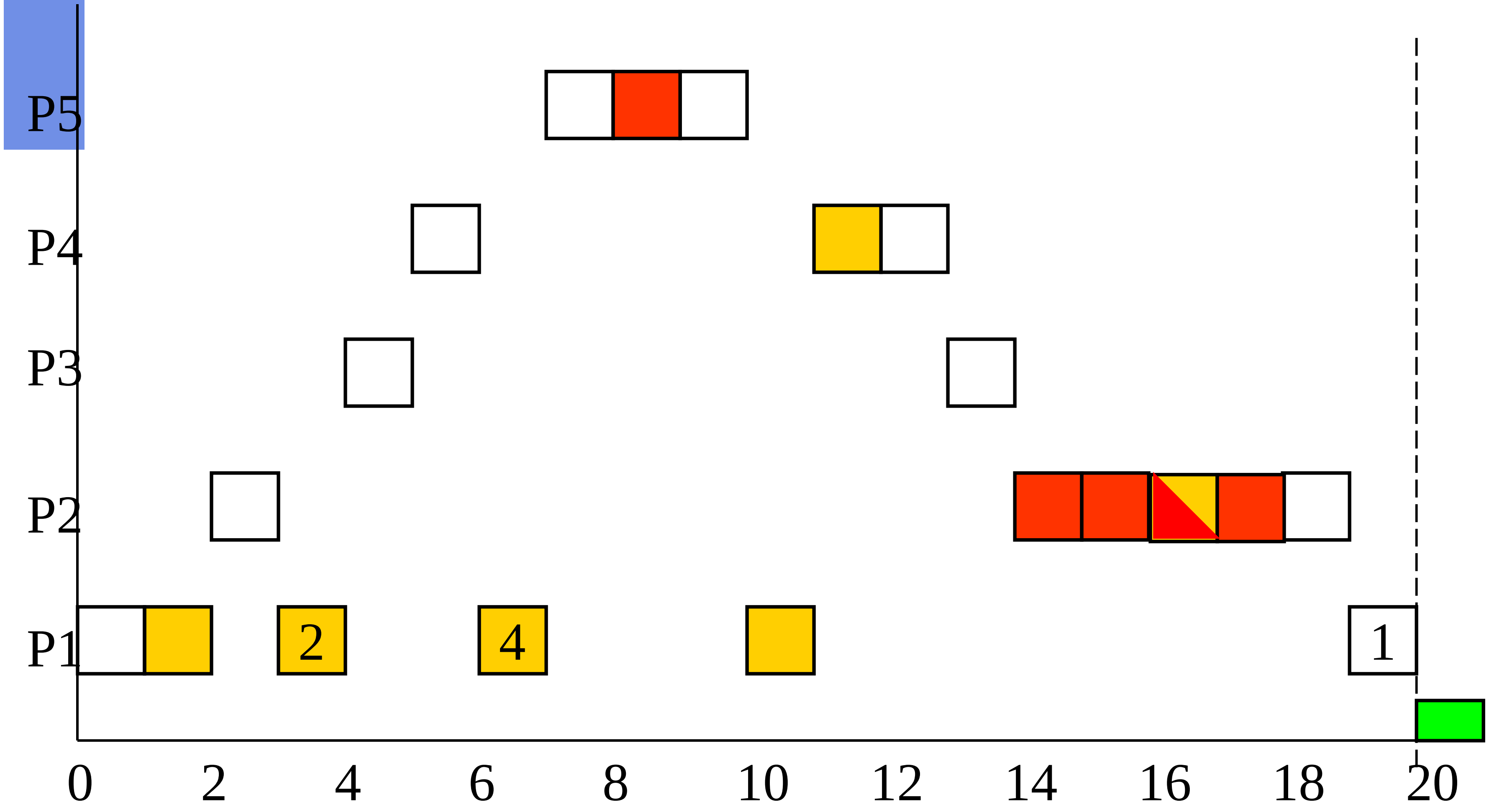
Example



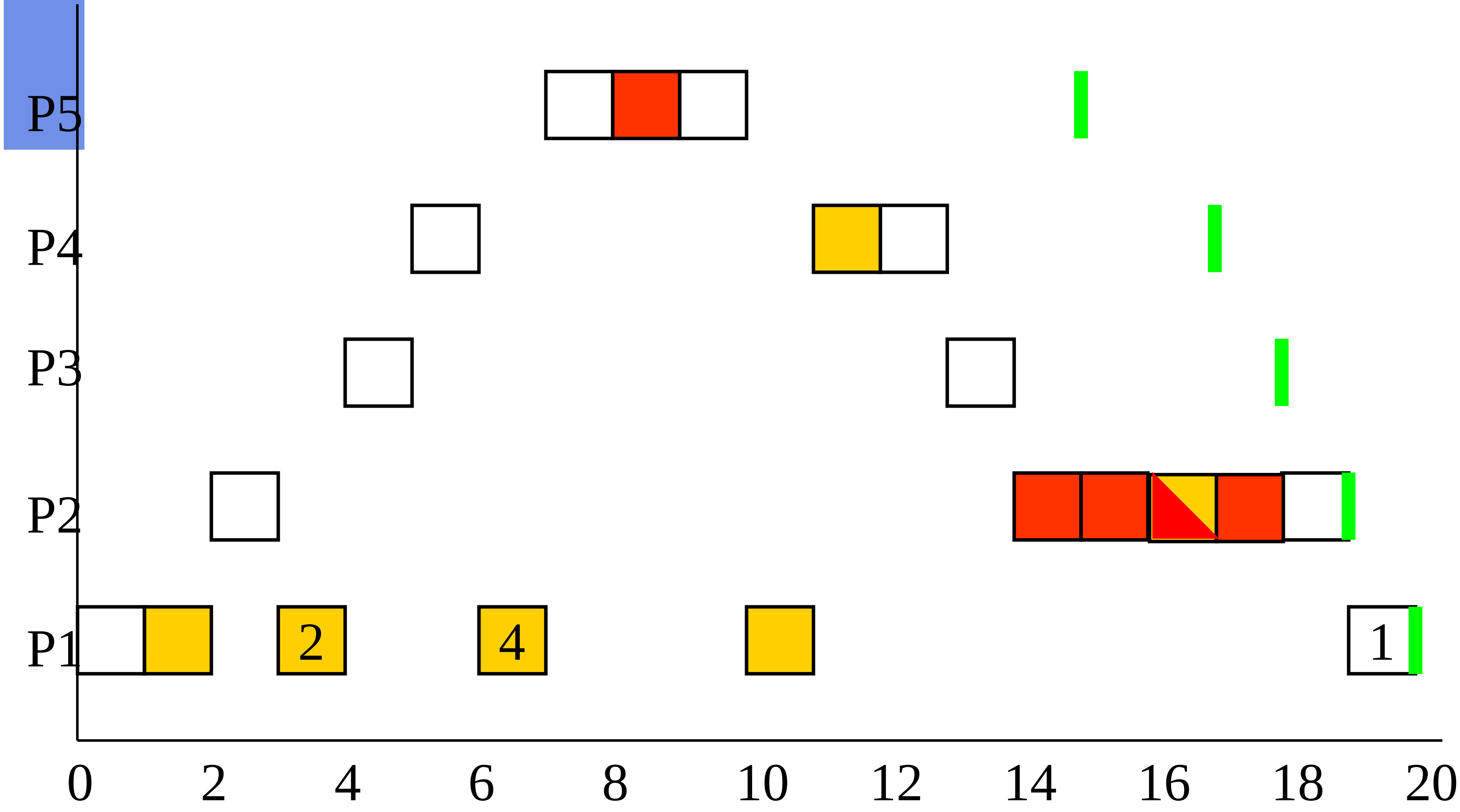
Example



Example



Comparison to Previous Example



Analysis: Priority Ceiling Protocol

- Pros
  - Avoids deadlocks
  - If a process doesn't self suspend, a process is *blocked at most once* during execution
  - Processes cannot be transitively blocked
    - ⇒minimizes blocking time to the longest lower-priority conflicting critical section (+ context switches)
    - Processes only receive their first resource when all required resources are not held by lower priority processes
- Cons
  - *A priori knowledge* of resource needs is required

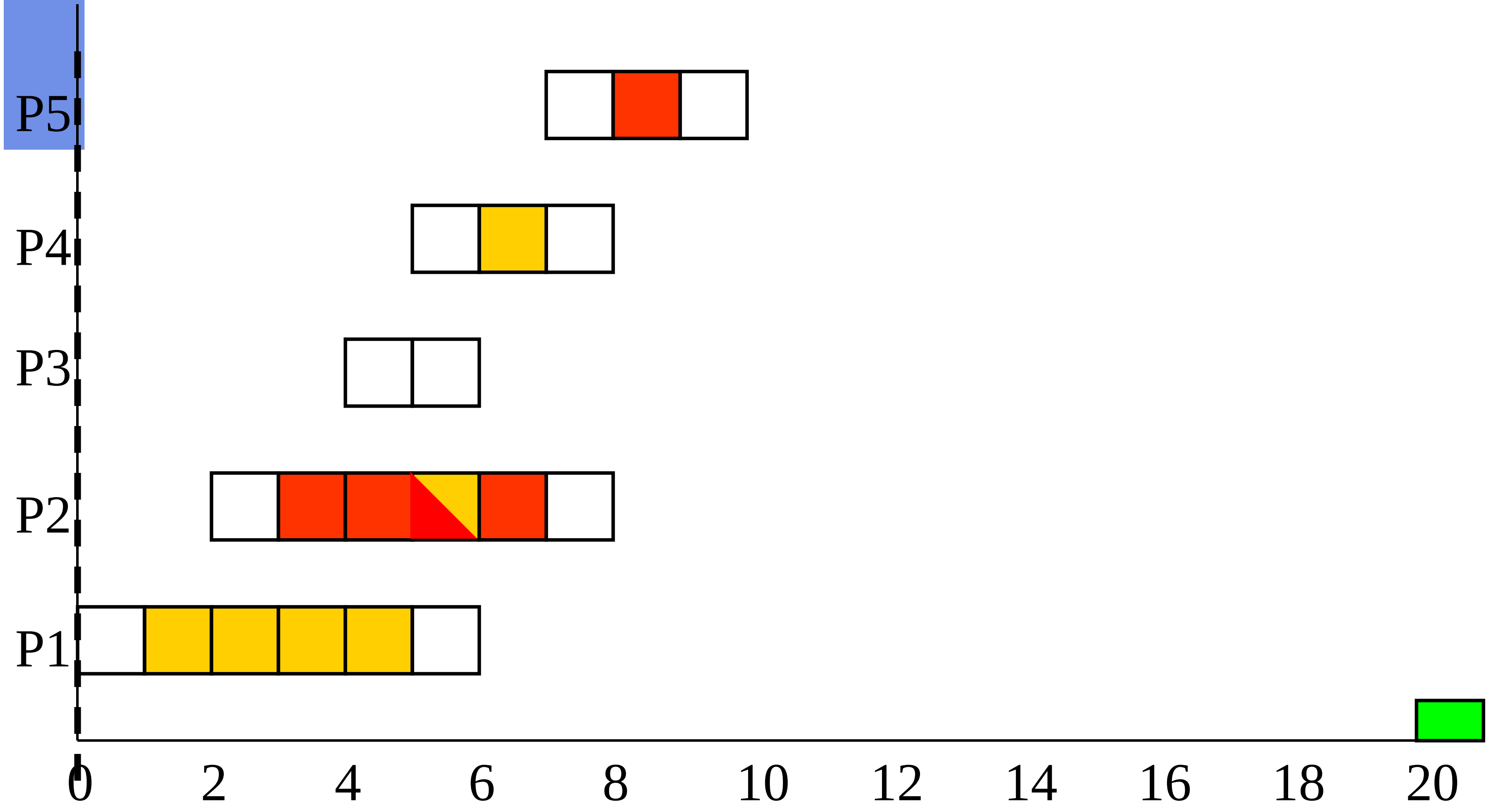
Stack-Based Priority Ceiling Protocol

- The motivation is to share a single stack for all processes
  - Saves stack space.
- Restriction: processes cannot self-suspend.

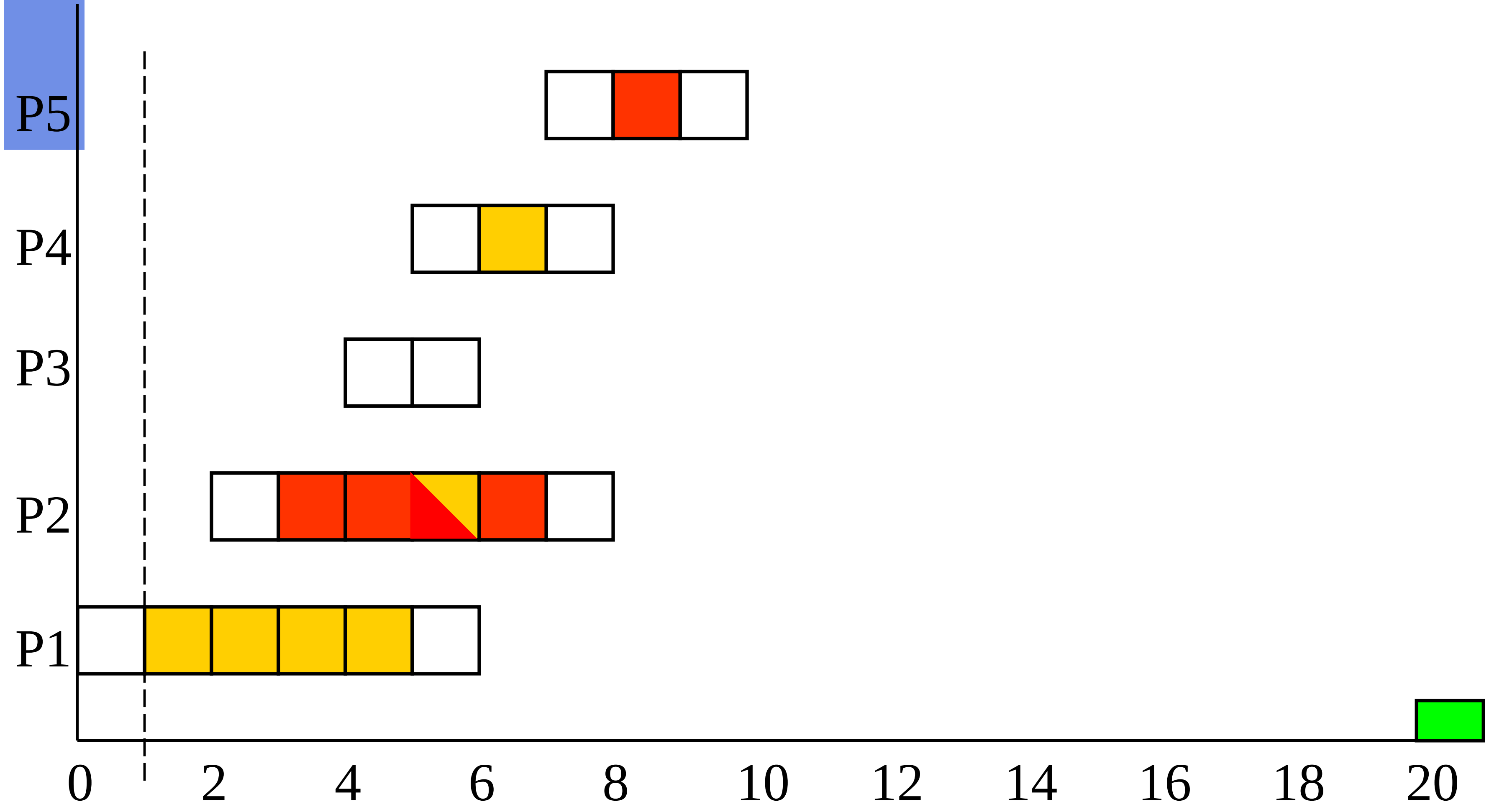
Rules

- Scheduling:
  - After a process is released, it is blocked from starting until its assigned priority is higher than the current system priority ceiling.
  - Unblocked processes are preemptively priority scheduled according to their assigned priority.
- Resource allocation:
  - Whenever a process requests a resource it receives the resource.

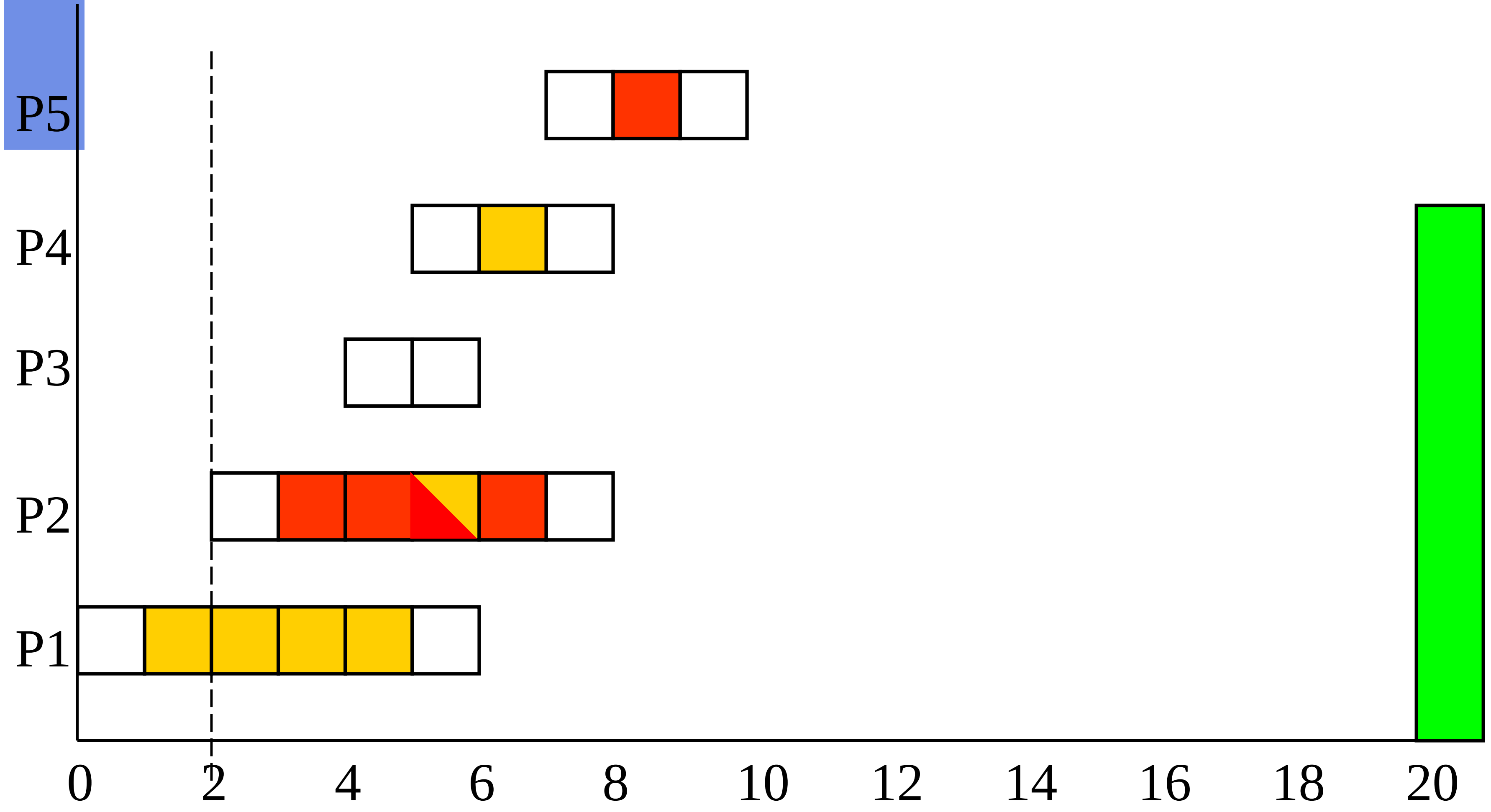
Example



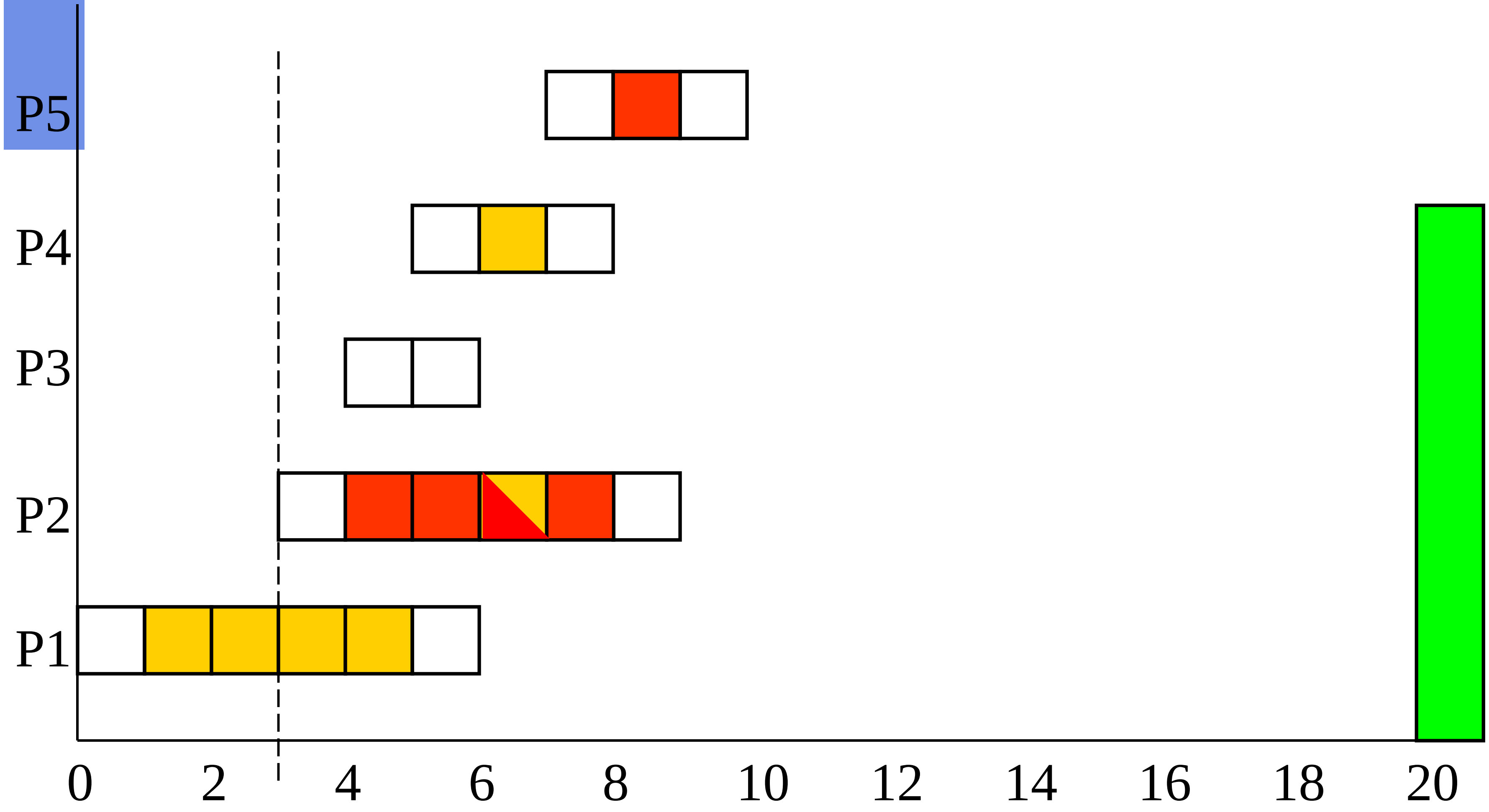
Example



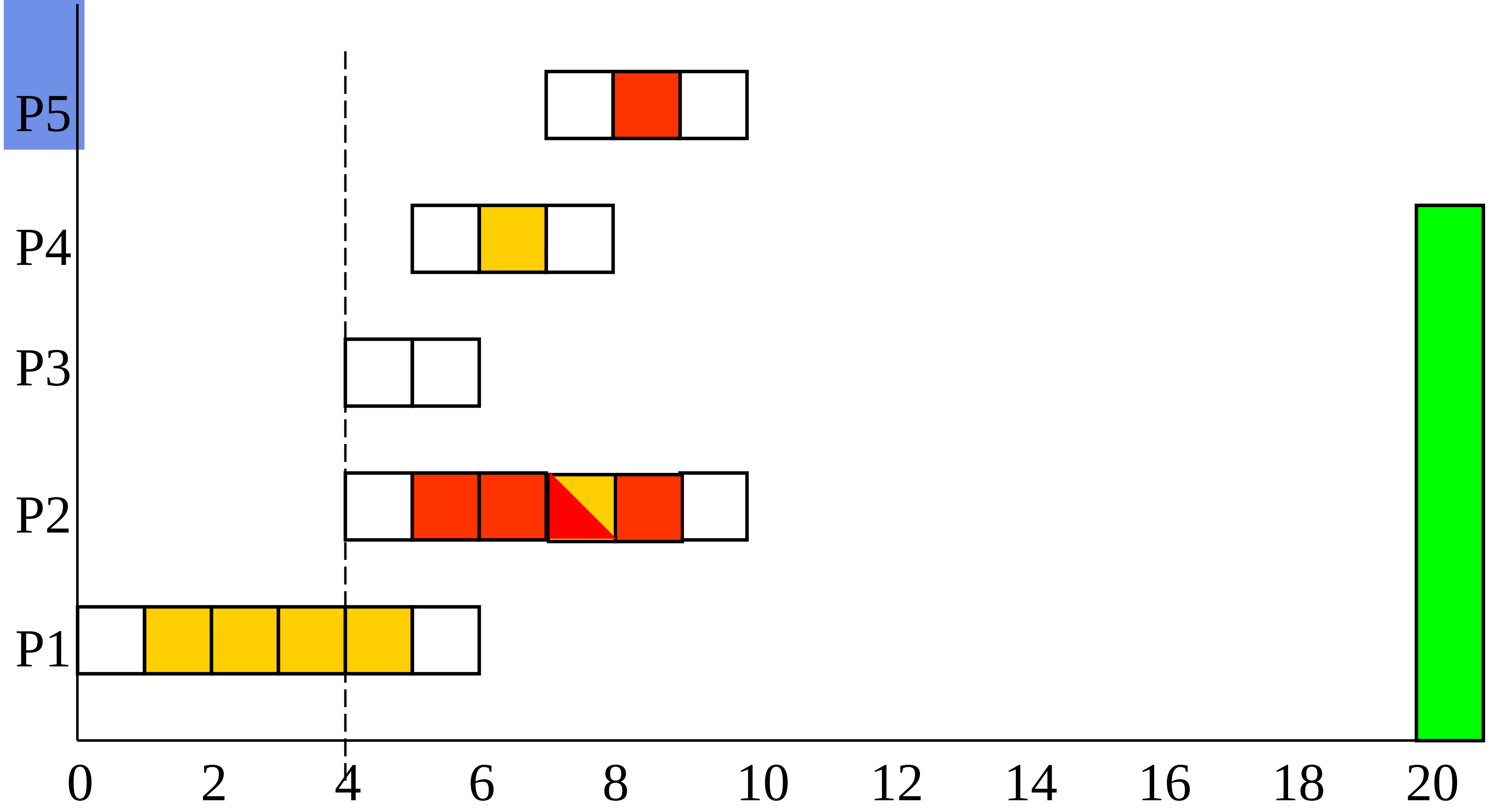
Example



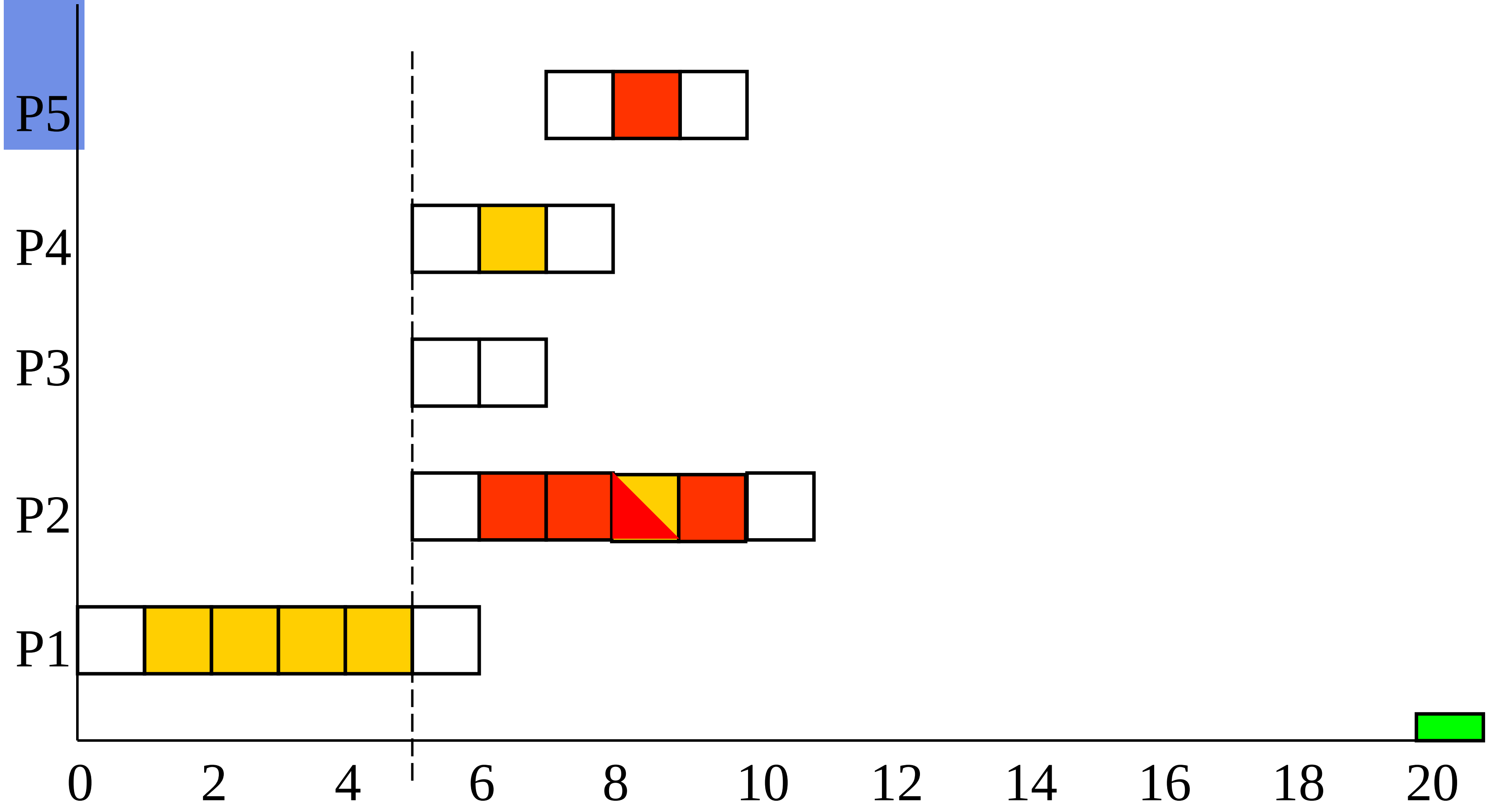
Example



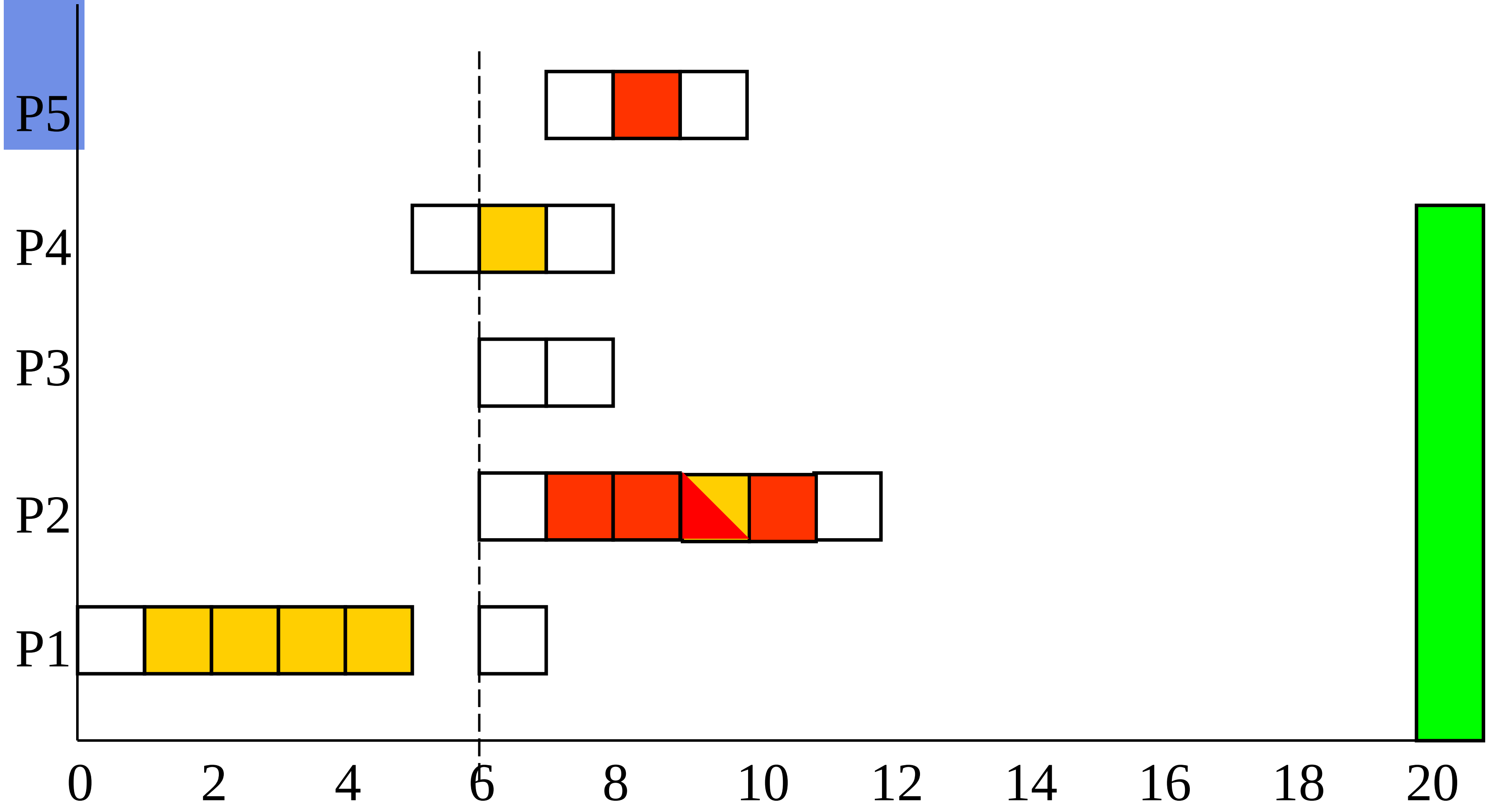
Example



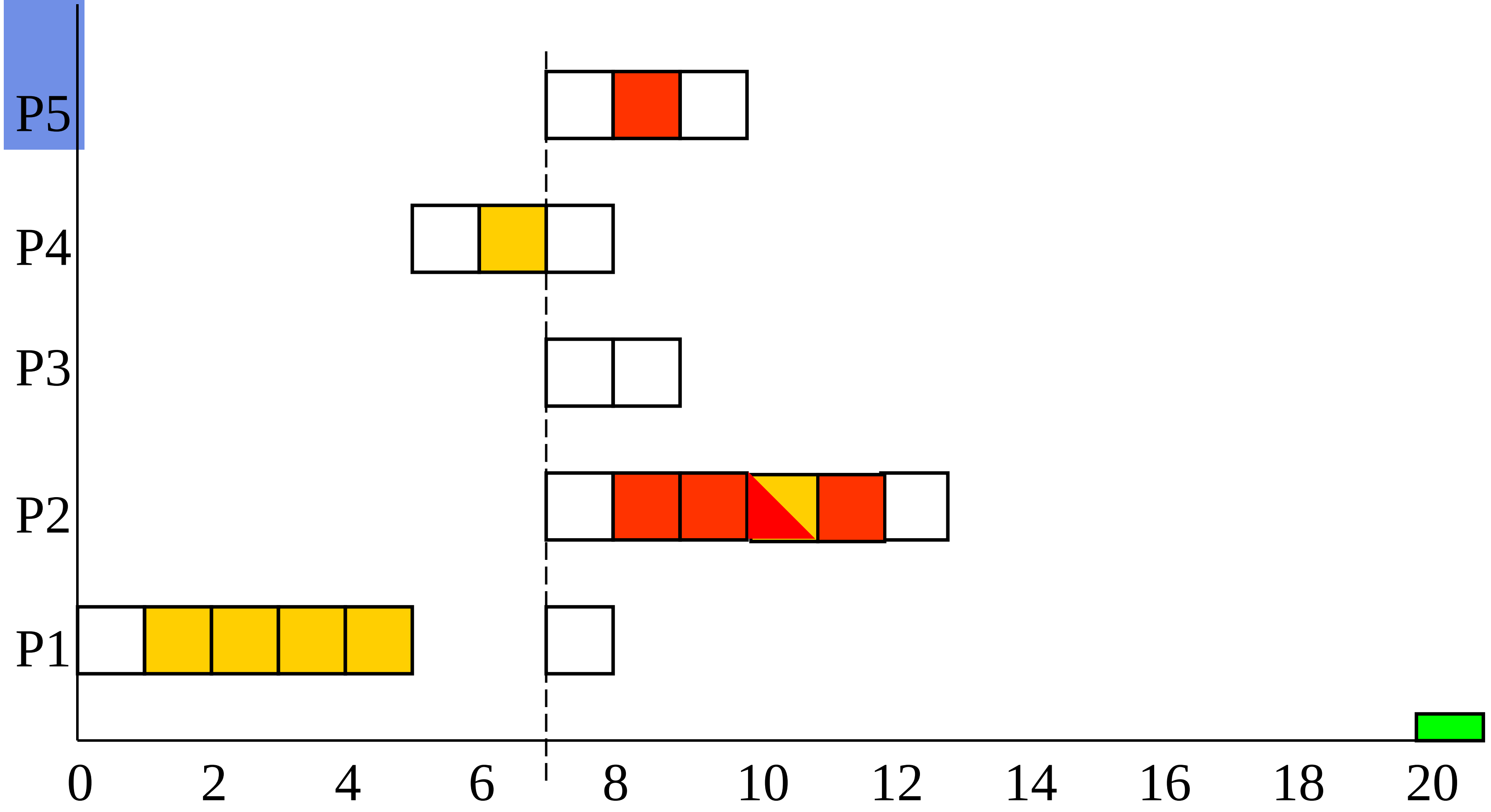
Example



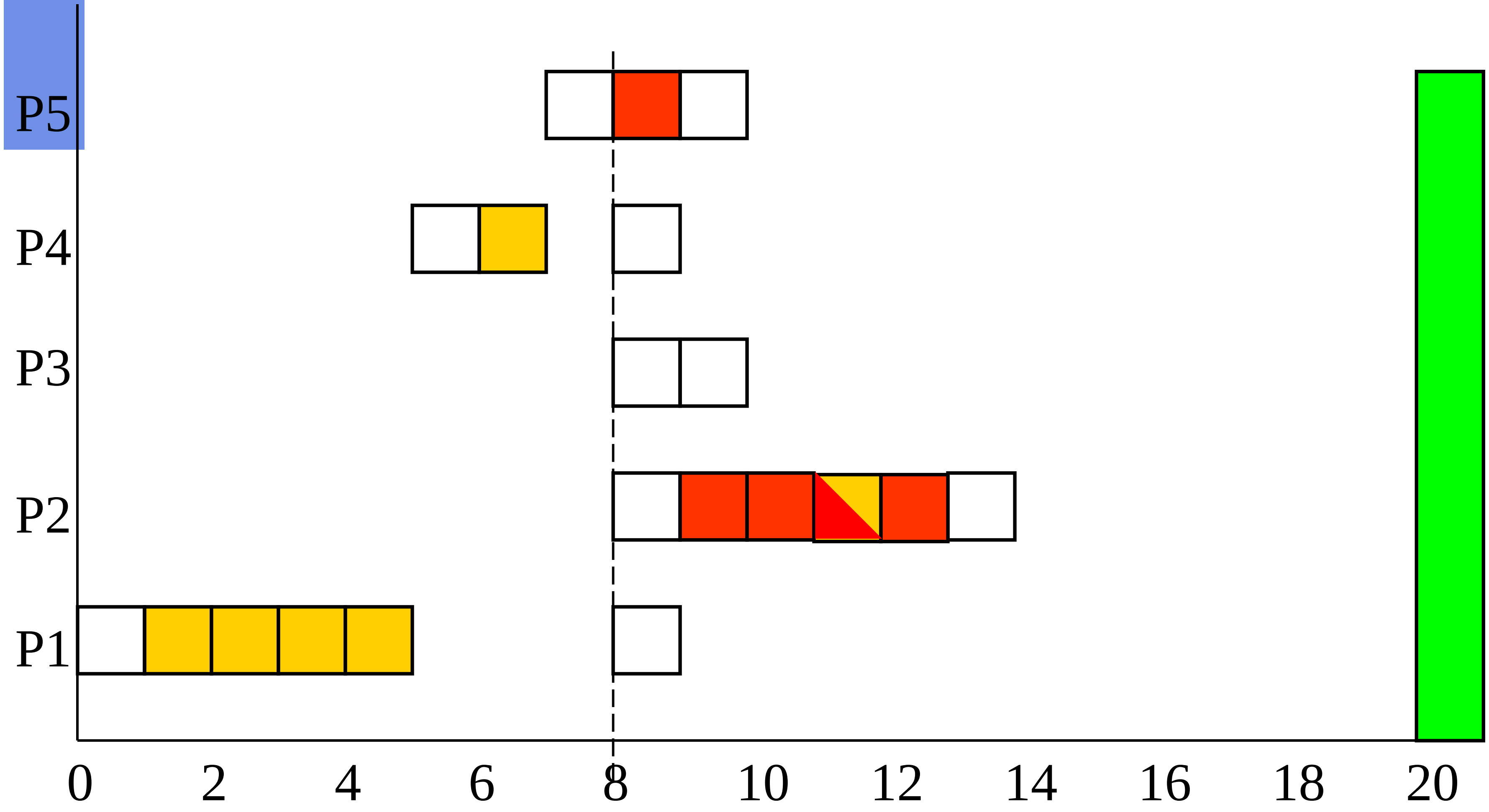
Example



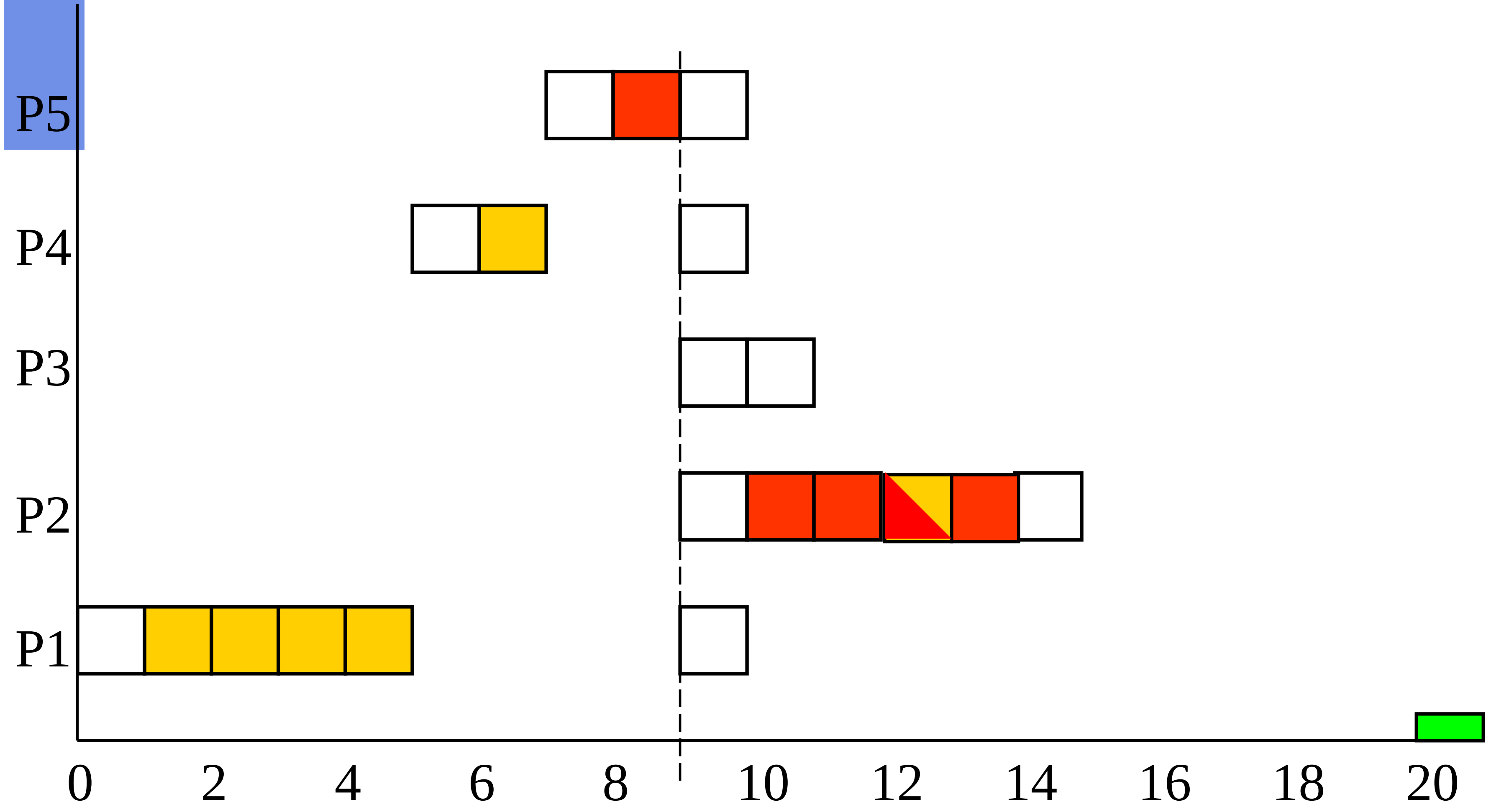
Example



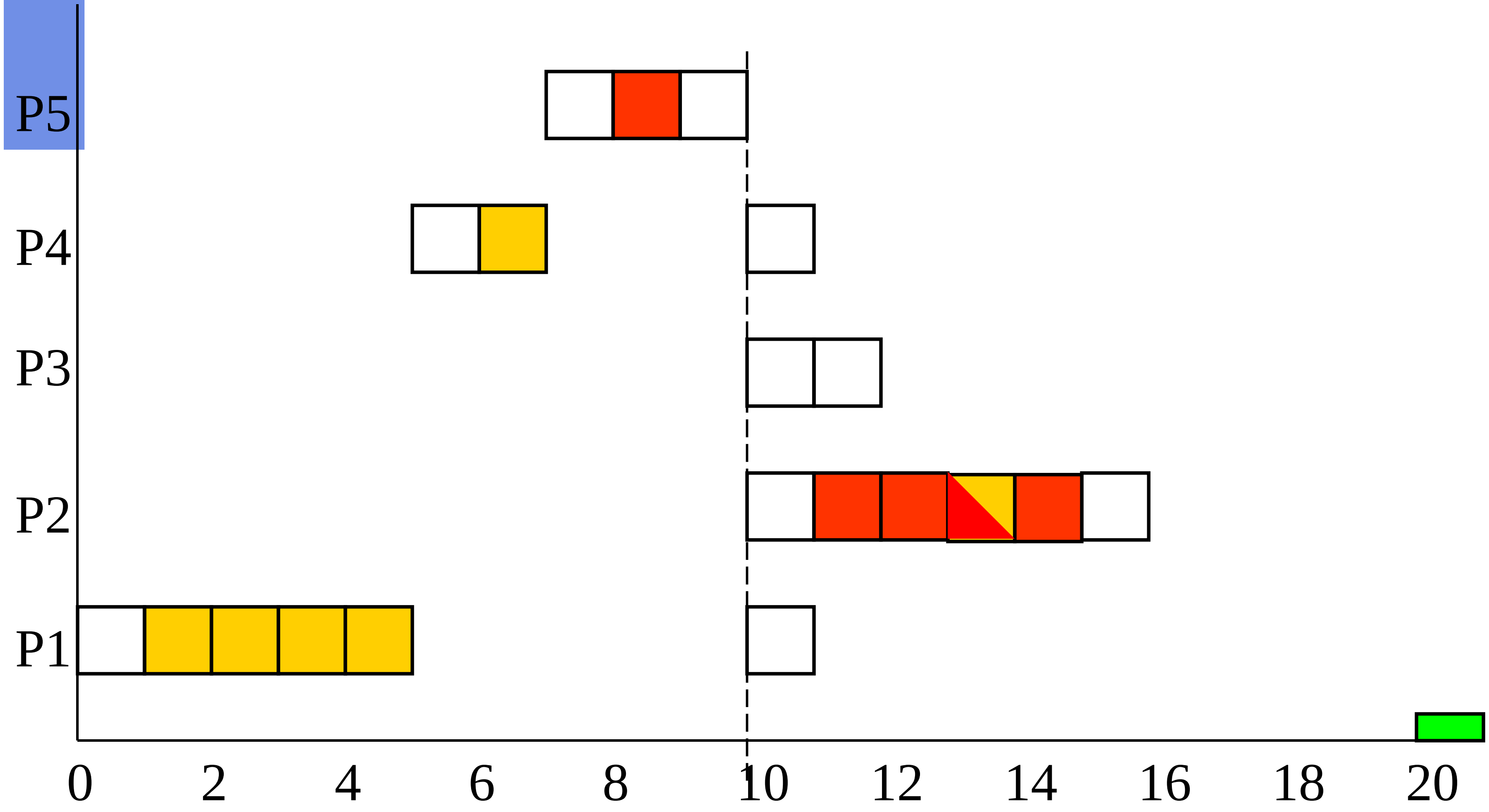
Example



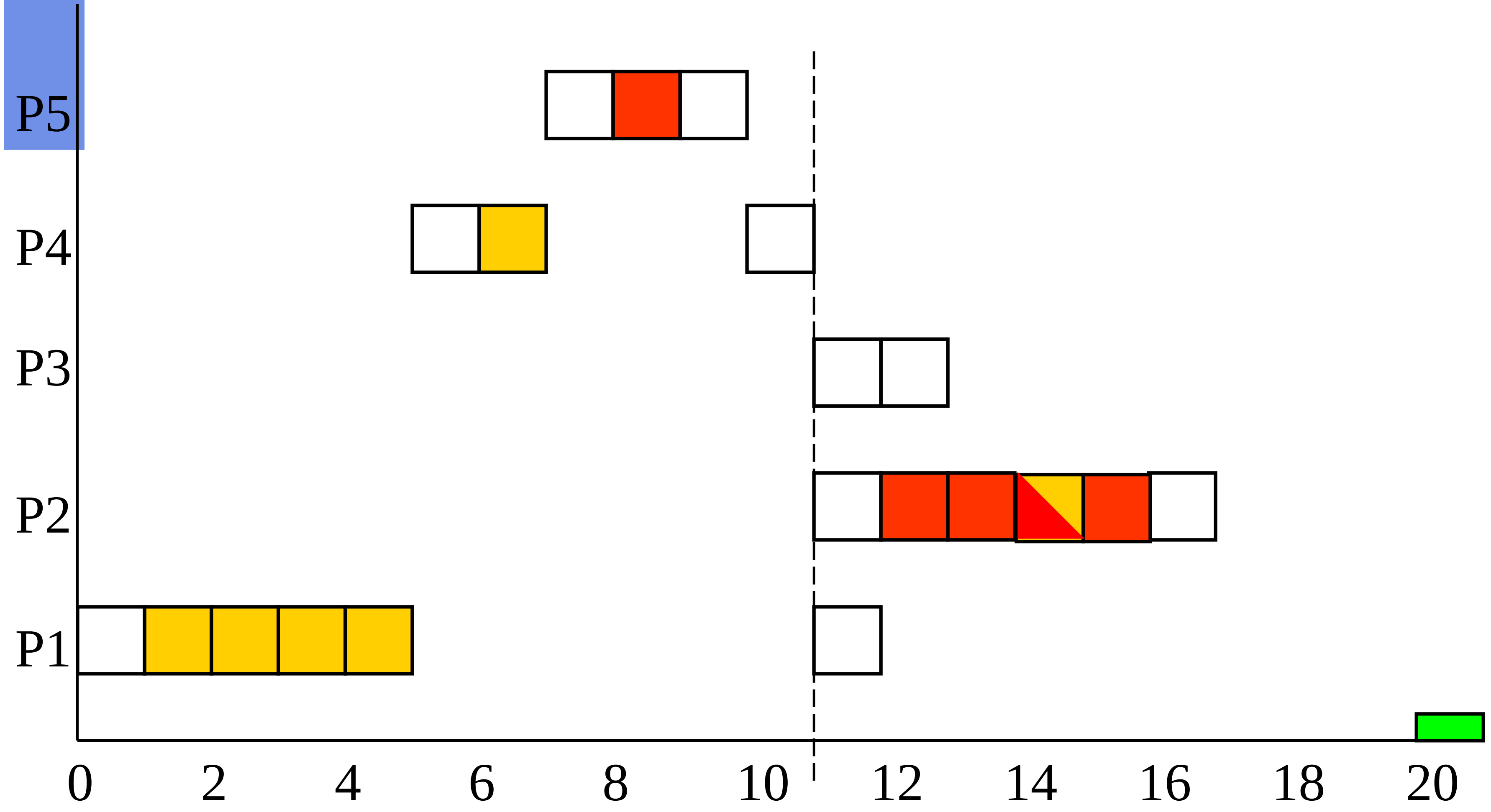
Example



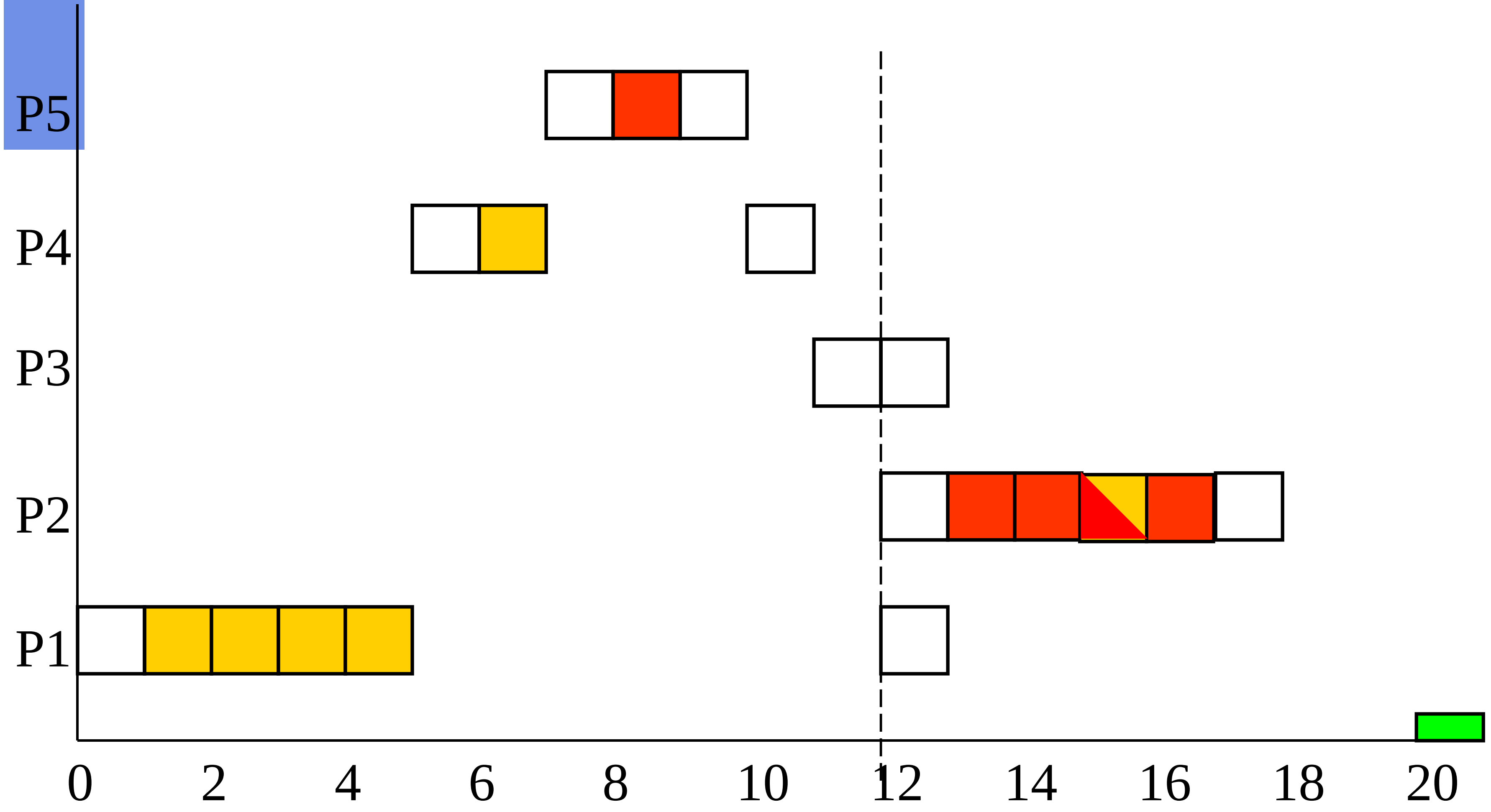
Example



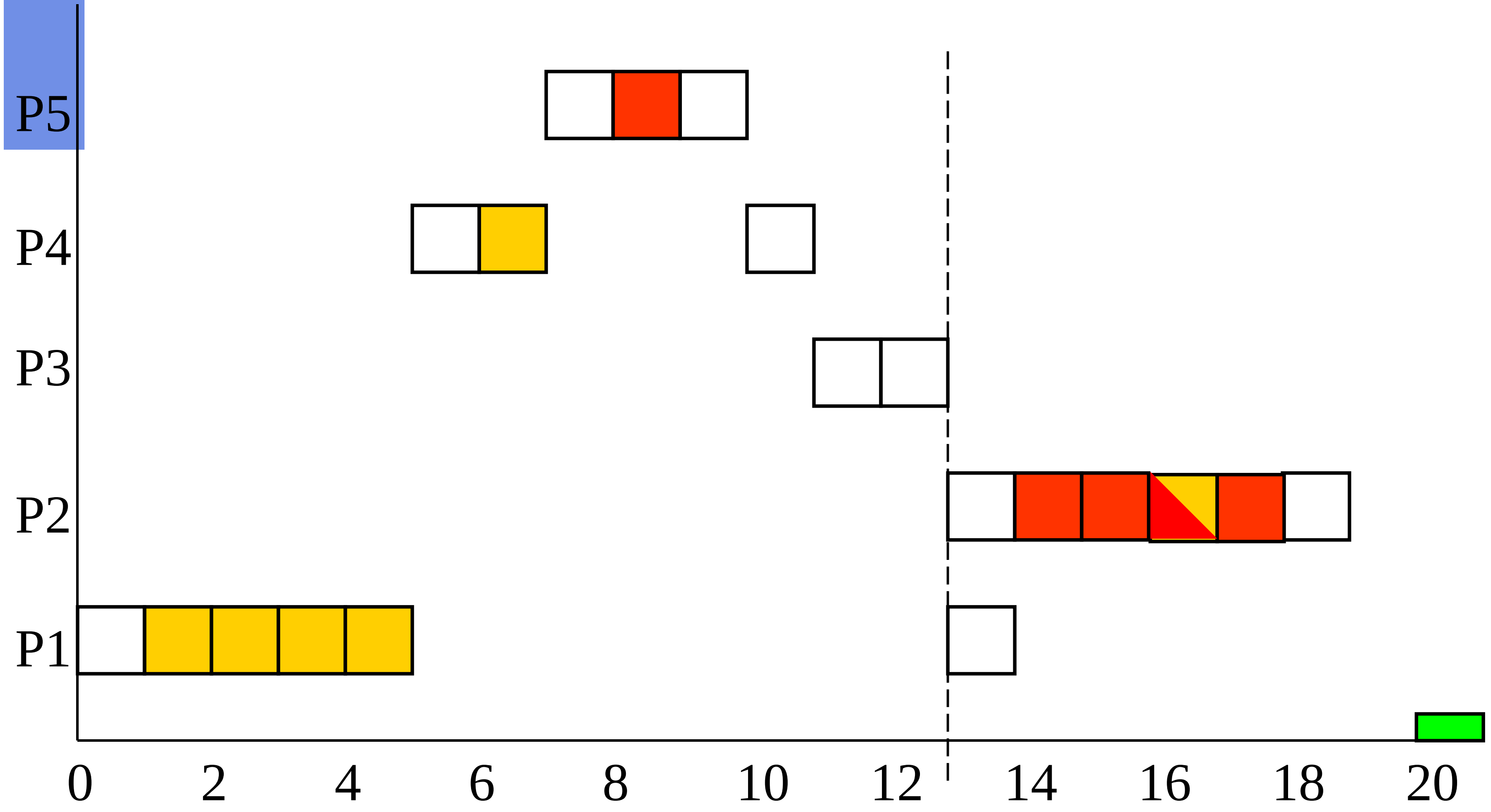
Example



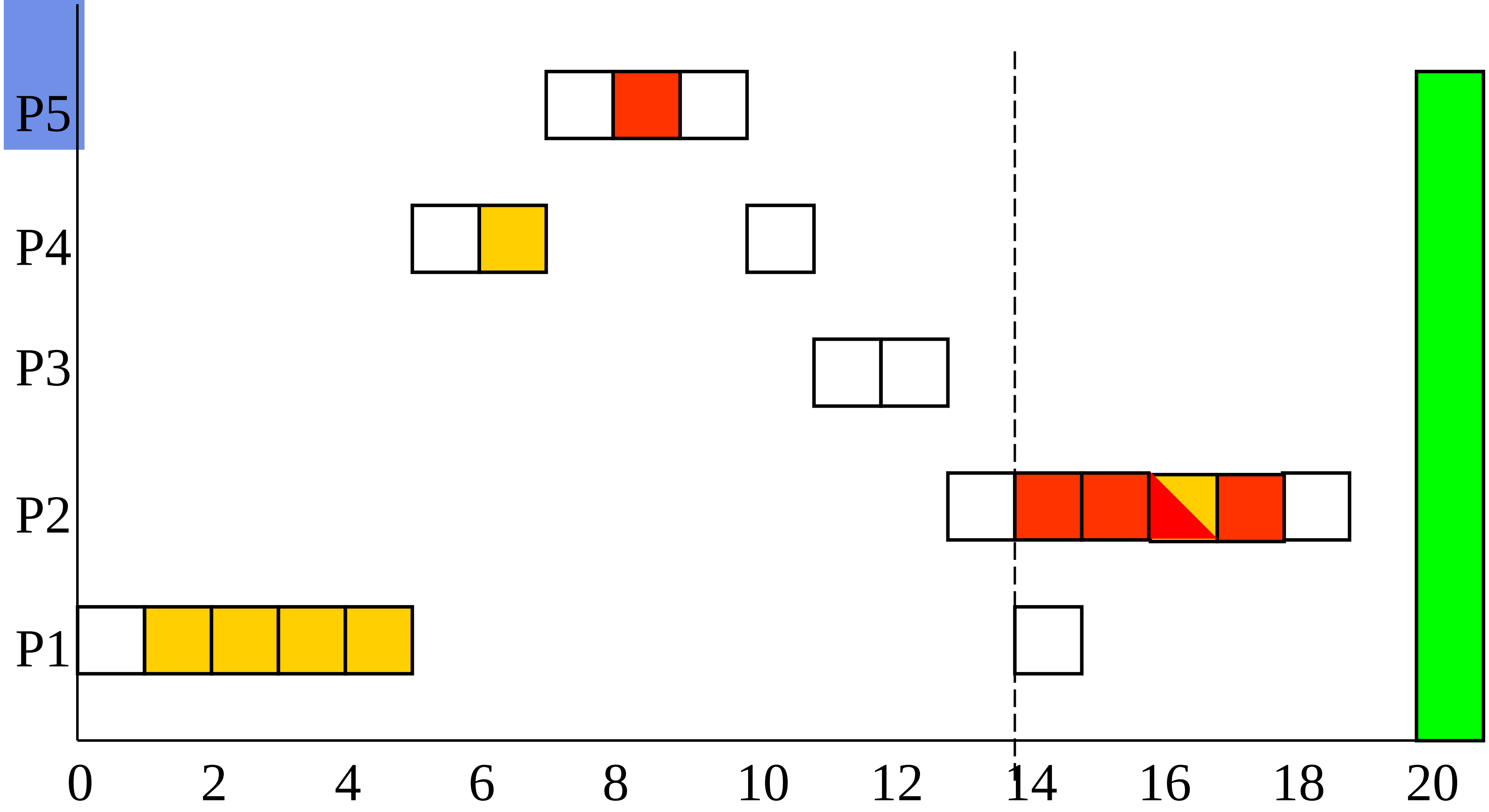
Example



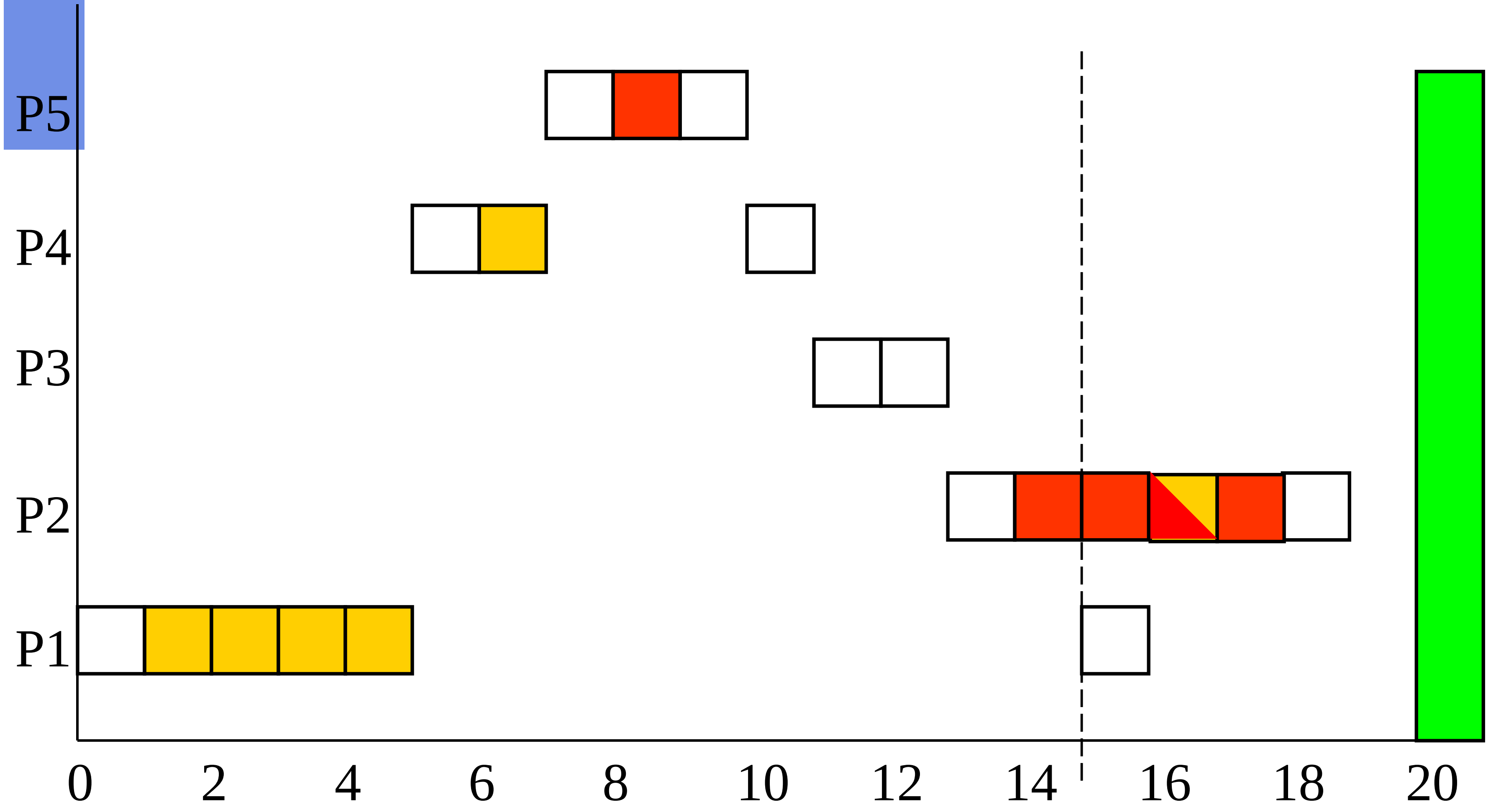
Example



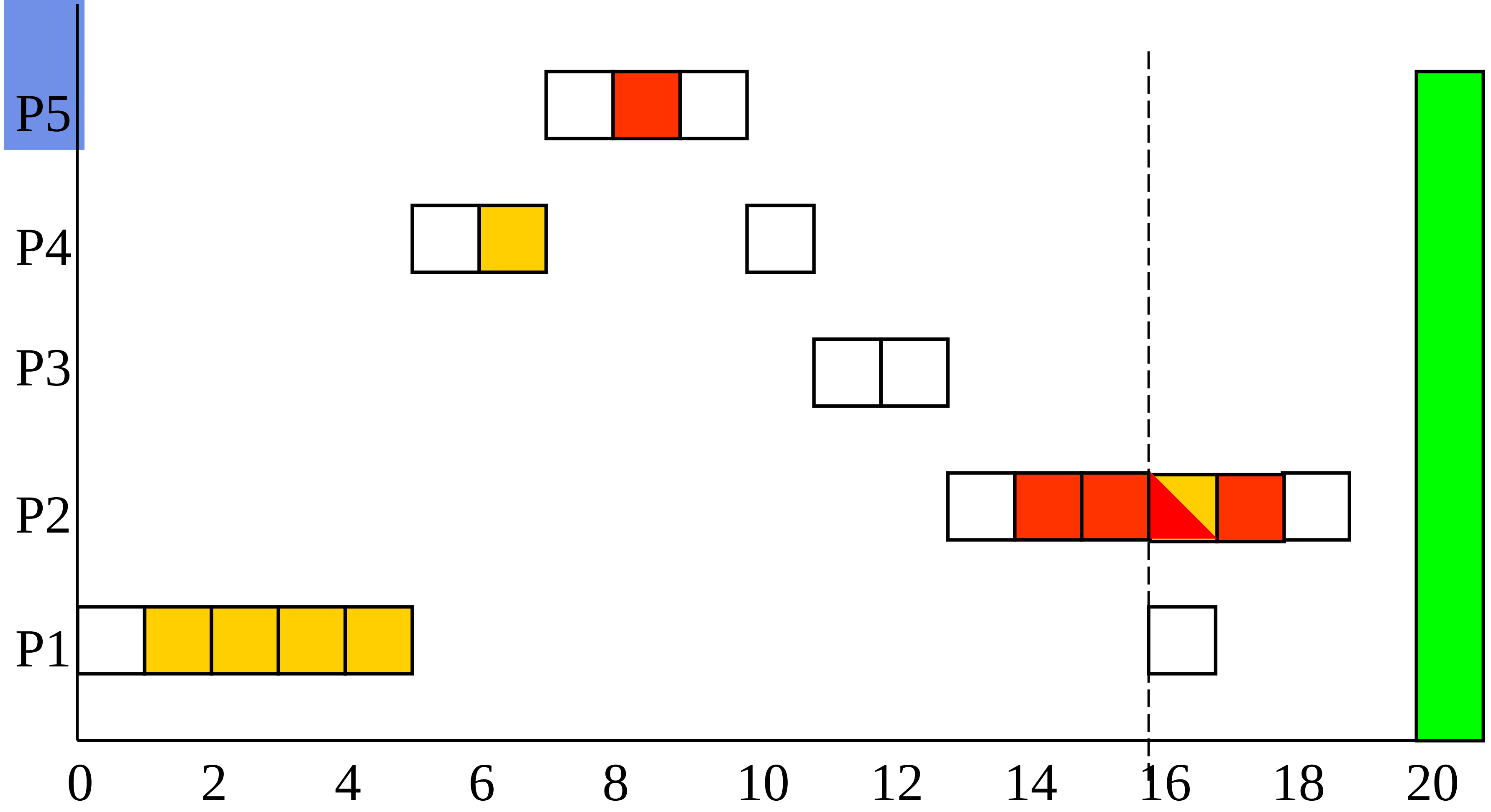
Example



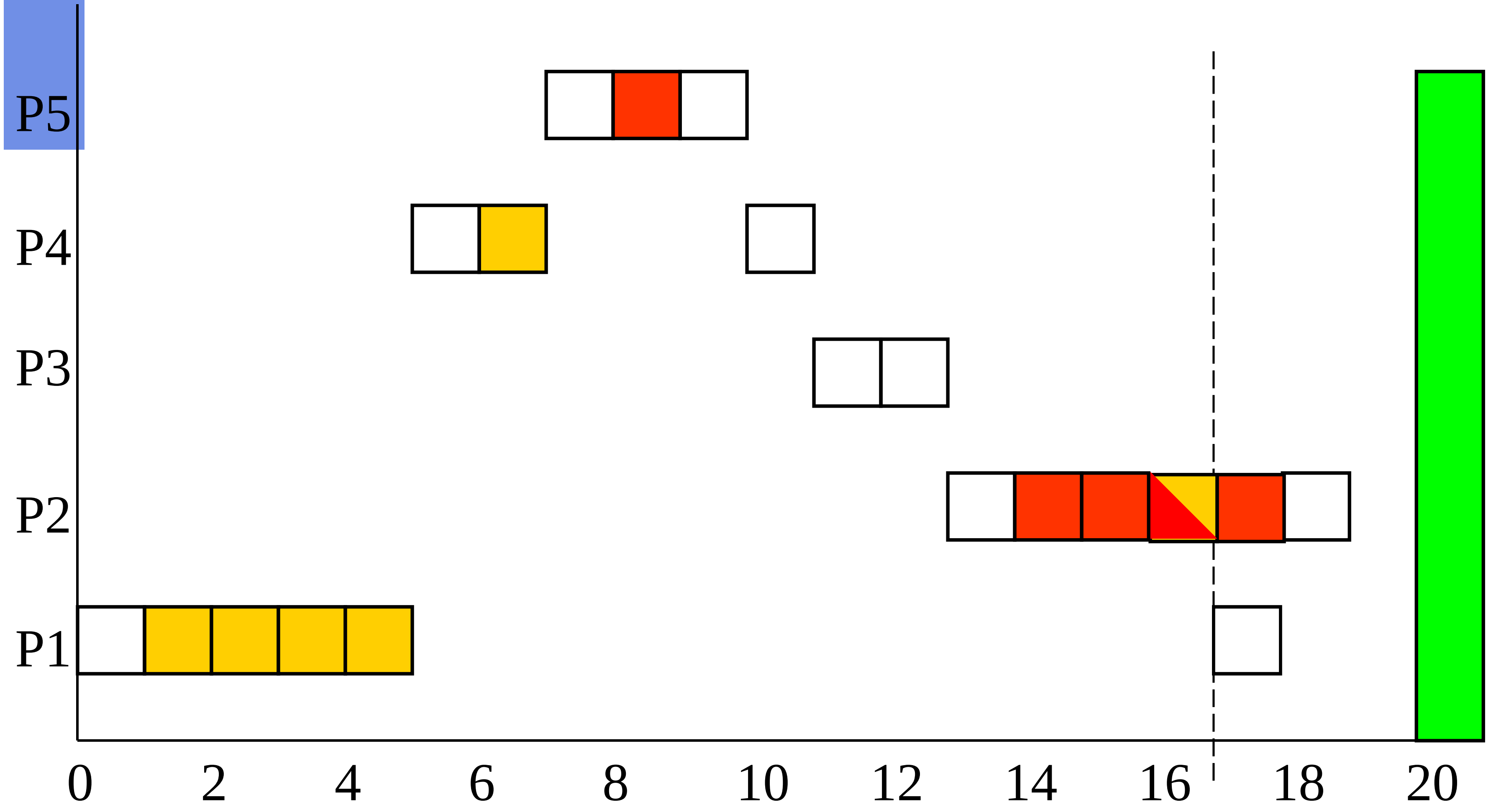
Example



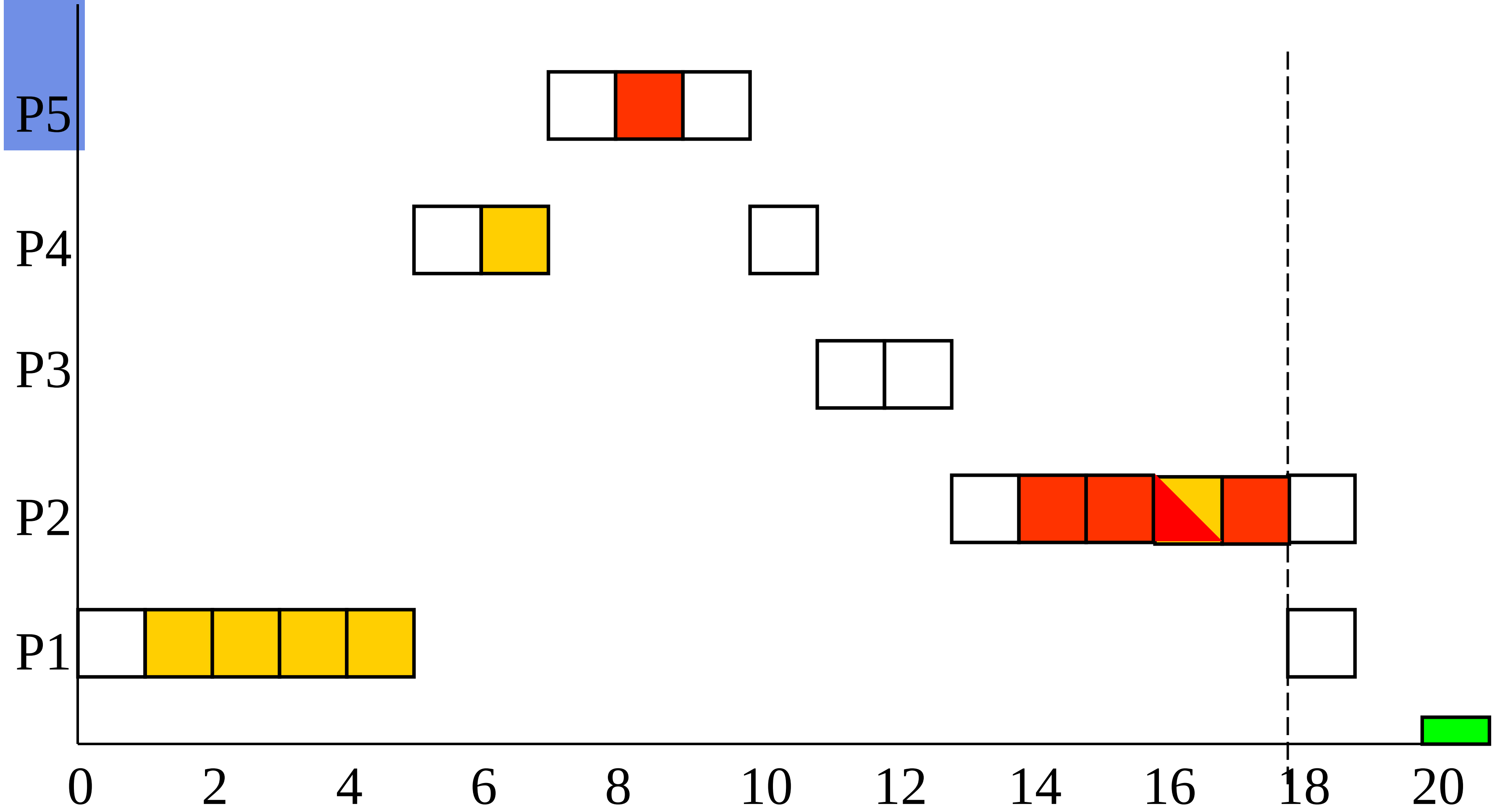
Example



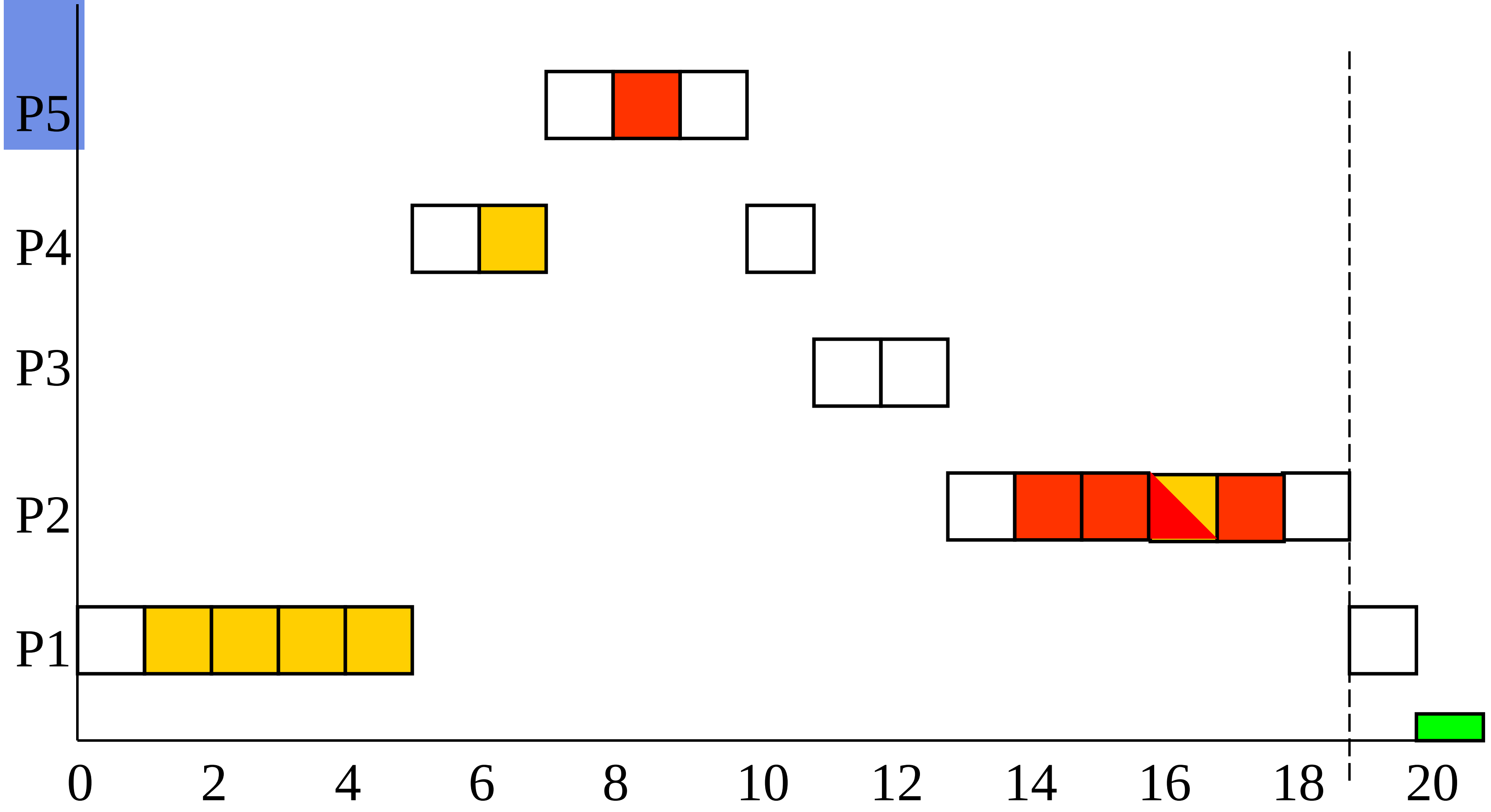
Example



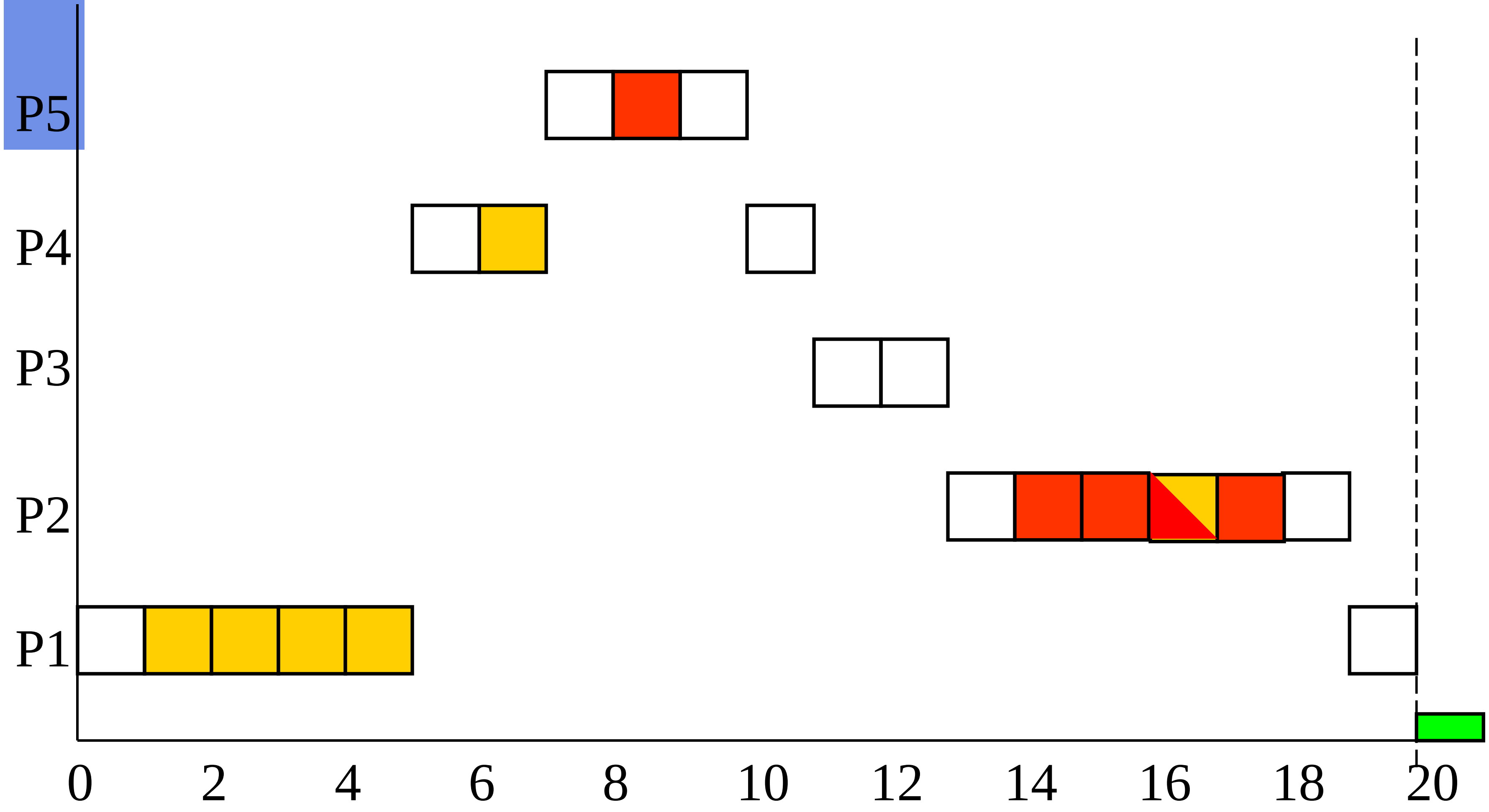
Example



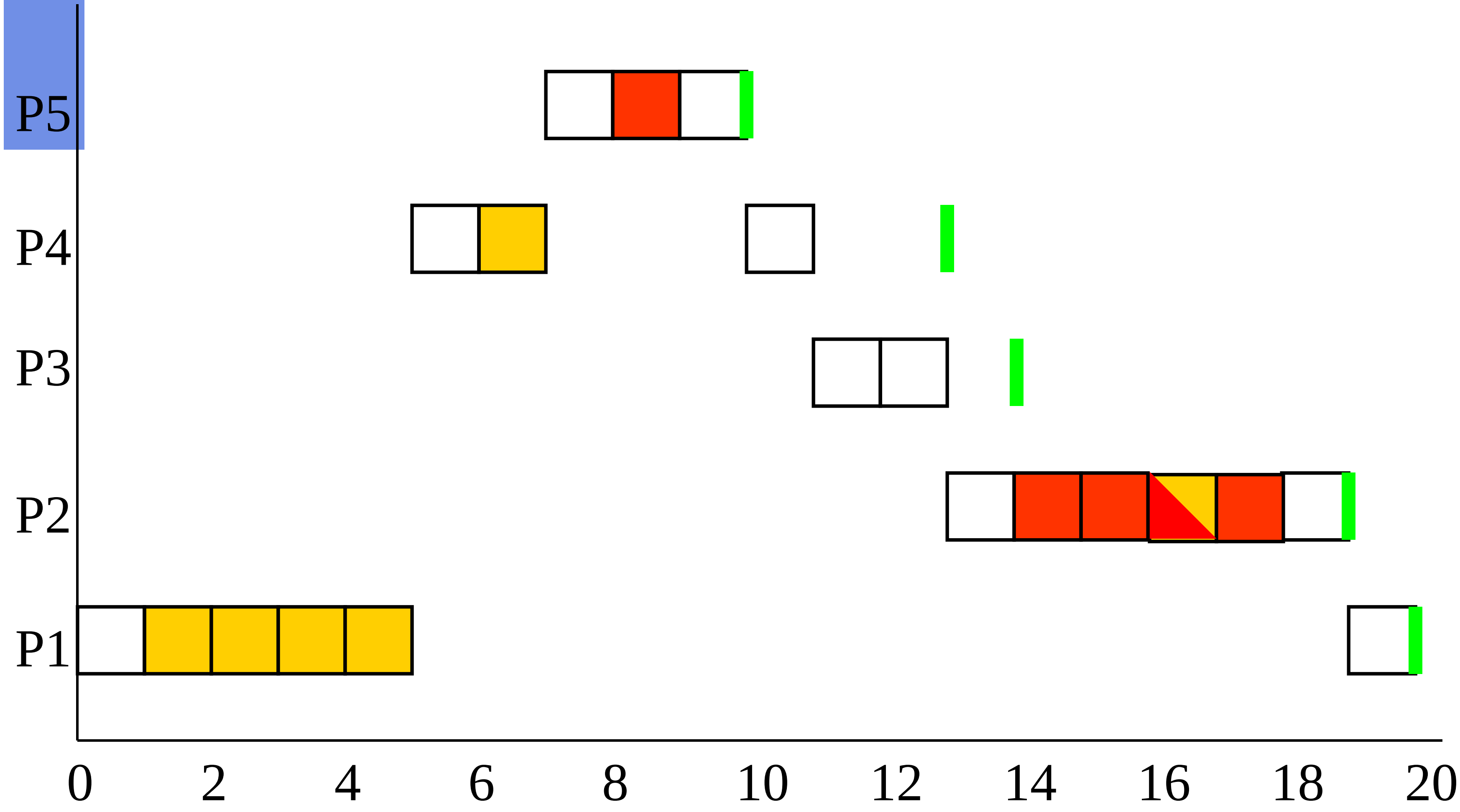
Example



Example



Comparison with Priority Ceiling Protocol



Analysis: Stack-Based Priority Ceiling

- Pros
  - Simple to implement.
  - Slightly better worst-case when compared to normal PCP – two less context switches.
  - No priority inheritance needed.
- Cons
  - Threads cannot self suspend.

Summary

- 4 protocols controlling resource access in priority driven preemptive systems
  - NPCS
  - PI
  - PCP
  - SPCP

Summary

- NPCS and PI do not require a priori knowledge of resource requirements
- PI neither prevents deadlocks nor avoids deadlocks
- All protocols -except PI- ensure that processes are blocked *at most once**