Project #3: EDF Scheduler

Kyu-Jin Cho

Systems Software & Architecture Lab.

Seoul National University

2024.04.15

Reminder: Late Submission Policy

You can use up to *3 slip days* during this semester

- You should explicitly declare the number of slip days you want to use on the QnA board right after each submission

- Once slip days have been used, they cannot be canceled later

- 25% of the credit will be deducted for every single day delay

XV6 Process States

- XV6 process states (in proc.h)
  - enum procstate
    {UNUSED, USED, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };

- UNUSED: not used
- USED: initialized for new process
- SLEEPING: wait for I/O, wait() or sleep()
- RUNNABLE: ready to run
- RUNNING: now running
- ZOMBIE: exited and waiting for parent to call wait()

XV6 Scheduler

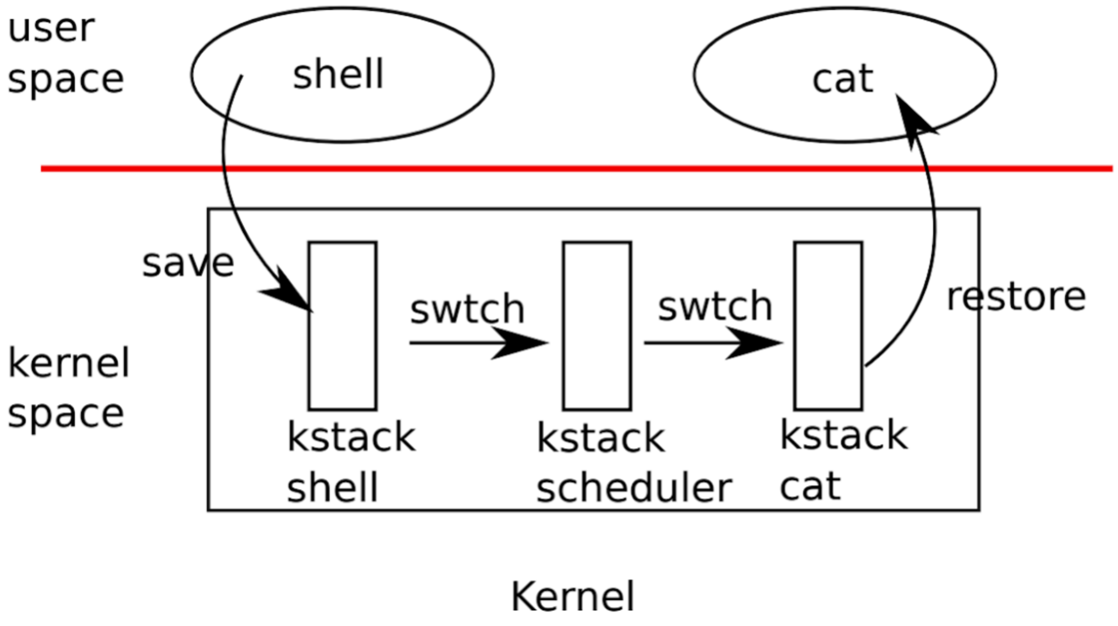
- XV6 multiplexes by switching each CPU from one process to another

- The XV6 scheduler implements a simple scheduling policy
  - Runs each process in turn
  - This is called Round Robin

- Each CPU calls scheduler()
- Scheduler never returns.

XV6 Scheduler

- Steps involved in switching from one user process to another
  - 1. User-kernel transition to the old's process's kernel thread
  - 2. Context switch to the current CPU's scheduler thread
  - 3. Context switch to a new process's kernel thread
  - 4. Trap return to the user-level process



XV6 Code : scheduler()

- In kernel/proc.c
  - void scheduler(void)
- Scheduler loops, doing
  - 1. Choose a RUNNABLE process p to run
  - 2. Mark process p's state to RUNNING
  - 3. Set the per-CPU current process
  - 4. Context switch (start running process p)
  - 5. If process is done running, go to 1.
- Scheduler never returns

```c
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();

  c->proc = 0;
  for(;;){
    // Avoid deadlock by ensuring that devices can interrupt.
    intr_on();

    for(p = proc; p < &proc[NPROC]; p++) {
      acquire(&p->lock);
      if(p->state == RUNNABLE) {
        // Switch to chosen process.  It is the process's job
        // to release its lock and then reacquire it
        // before jumping back to us.
        p->state = RUNNING;
        c->proc = p;
        swtch(&c->context, &p->context);

        // Process is done running for now.
        // It should have changed its p->state before coming back.
        c->proc = 0;
      }
      release(&p->lock);
    }
  }
}
```

XV6 Code : sched()

- In kernel/proc.c
  - void sched(void)

- Called from exit(), sleep(), yield()

- Context switch (return to scheduler)

```
void
sched(void)
{
  int intena;
  struct proc *p = myproc();

  if(!holding(&p->lock))
    panic("sched p->lock");
  if(mycpu()->noff != 1)
    panic("sched locks");
  if(p->state == RUNNING)
    panic("sched running");
  if(intr_get())
    panic("sched interruptible");

  intena = mycpu()->intena;
  swtch(&p->context, &mycpu()->context);
  mycpu()->intena = intena;
}
```

XV6 Code : swtch()

- In kernel/swtch.S
  - void swtch(struct context *old, struct context *new)

- Save current registers in old, load from new

```
.globl swtch
swtch:
        sd ra, 0(a0)
        sd sp, 8(a0)
        sd s0, 16(a0)
        sd s1, 24(a0)
        sd s2, 32(a0)
        sd s3, 40(a0)
        sd s4, 48(a0)
        sd s5, 56(a0)
        sd s6, 64(a0)
        sd s7, 72(a0)
        sd s8, 80(a0)
        sd s9, 88(a0)
        sd s10, 96(a0)
        sd s11, 104(a0)

        ld ra, 0(a1)
        ld sp, 8(a1)
        ld s0, 16(a1)
        ld s1, 24(a1)
        ld s2, 32(a1)
        ld s3, 40(a1)
        ld s4, 48(a1)
        ld s5, 56(a1)
        ld s6, 64(a1)
        ld s7, 72(a1)
        ld s8, 80(a1)
        ld s9, 88(a1)
        ld s10, 96(a1)
        ld s11, 104(a1)

        ret
```

Project#3: EDF Scheduler

- In this project, you have to
  1. Implement the sched_setattr() system call (10 points)
  2. Implement the sched_yield() system call (20 points)
  3. Implement the EDF (Earliest Deadline First) scheduler (70 points)

- Due date is 11:59(PM), April 28st (Sunday)

Project#3-1. Implement the sched_setattr()

- int sched_setattr(int pid, int runtime, int period)
  - Sets the EDF scheduler parameters for the process whose id is pid
  - If pid equals 0, the parameters of the calling process will be set
  - The value of parameters should be positive integer where runtime < period
  - For normal processes, these values are initialized to 0

- Return value
  - 0 on success, -1 on error

Project#3-2. Implement the sched_yield()

- void sched_yield()
  - For normal processes, the system call causes the calling process to relinquish the CPU
  - For real-time processes, calling this system call means that the process has completed its execution in the current period

- In the skeleton code,
  - It simply calls the yield() kernel function to schedule the next runnable process
  - You need to modify the sched_yield() for real-time processes, so that the current real-time process gives up the CPU and waits until the start of its next period

Project#3-3. Implement the EDF scheduler

- In EDF scheduling, processes are prioritized based on their deadlines
  - The process with the closest deadline is given the highest priority and is scheduled to run next
  - $P_i = (C_i, T_i)$
    - C_i is the maximum runtime that the process requires to complete under worst-case
    - T_i is the period of the process at which the process repeats

- In the periodic process model, deadlines are equal to periods
  - Once a process is scheduled, the execution of the process should be completed before the start of the next period

Project#3-3. Implement the EDF scheduler

- Add the EDF scheduler on top of the default round-robin scheduler
  - Real-time processes always have a higher priority than normal processes
  - Normal processes are scheduled only when there are no runnable real-time processes

- If more than one real-time process has the same deadline
  1. Select the current process if it is among them
  2. If not, assign the CPU to the process with the smallest pid

EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)

Ticks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1																													
P2																													
P3																													

Release: P0, P1, P2

EDF Example

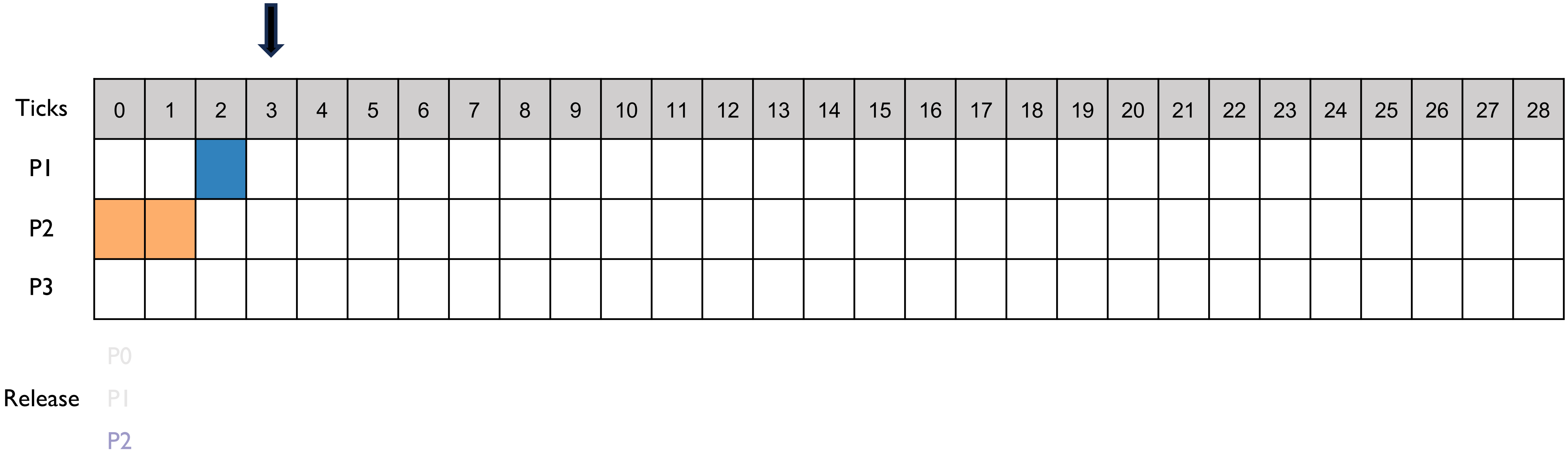
- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)

Ticks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
P1																													
P2																													
P3																													

Release P0 P1 P2

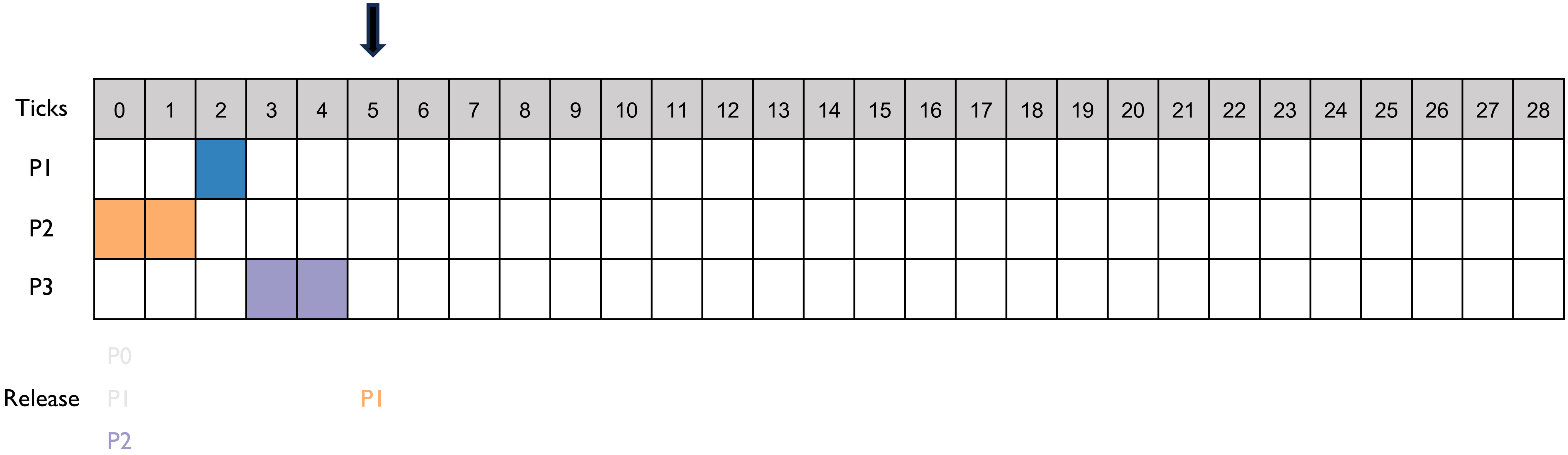
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



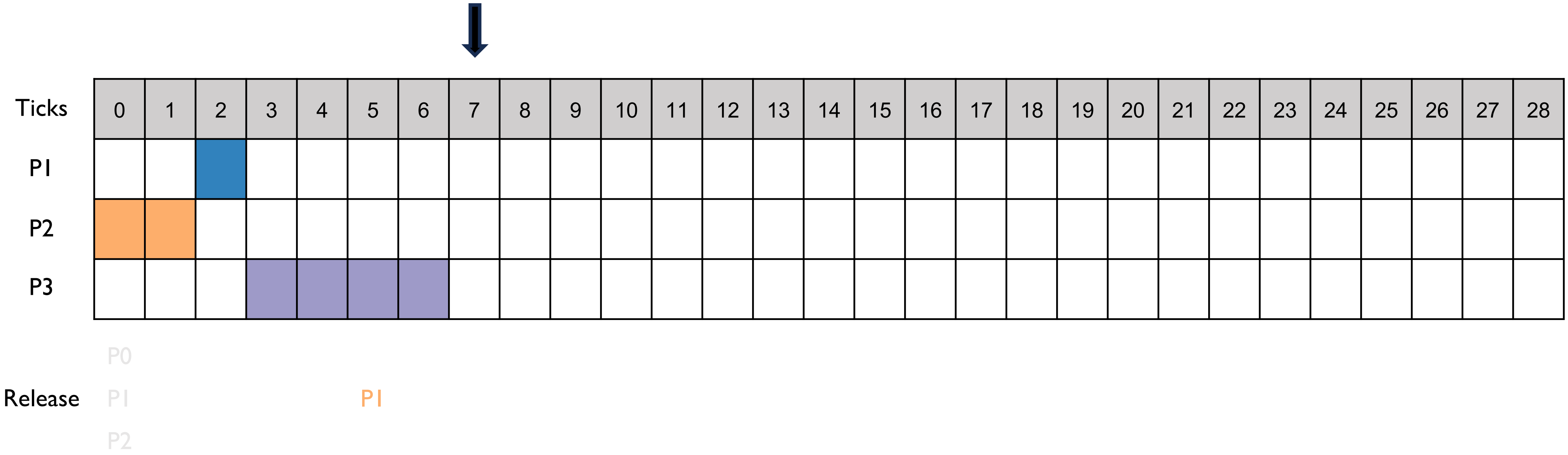
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



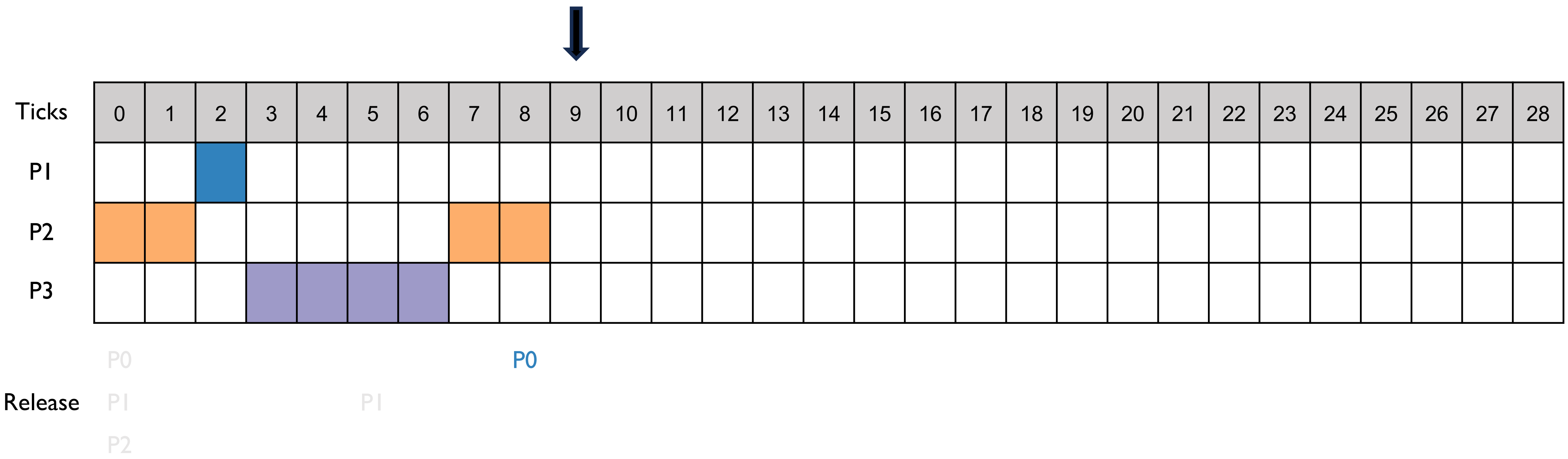
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



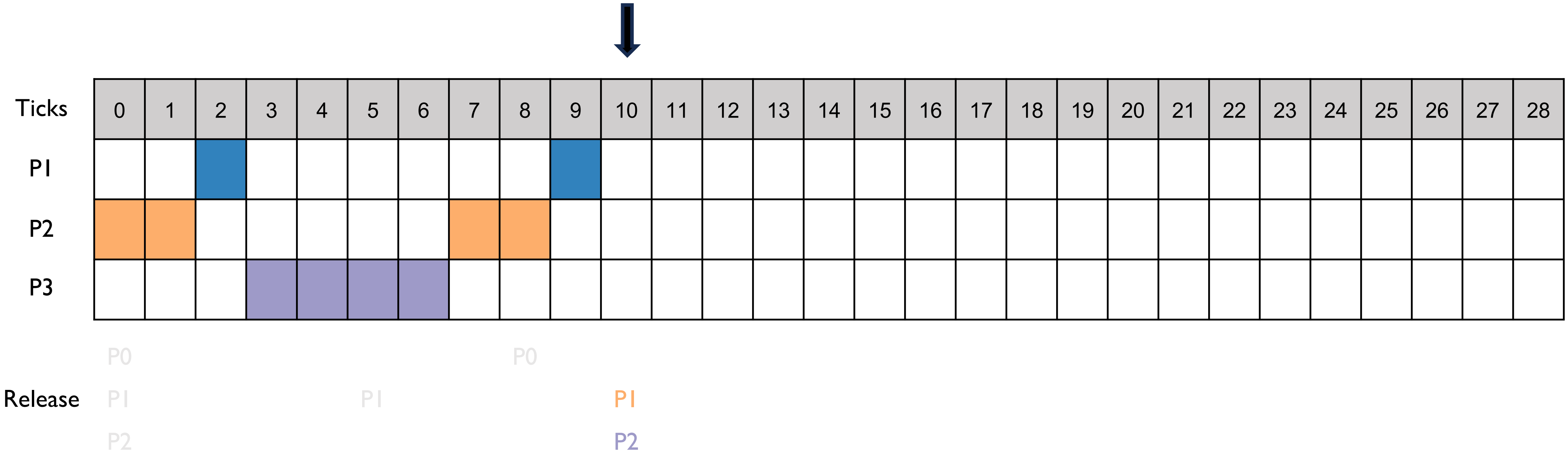
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



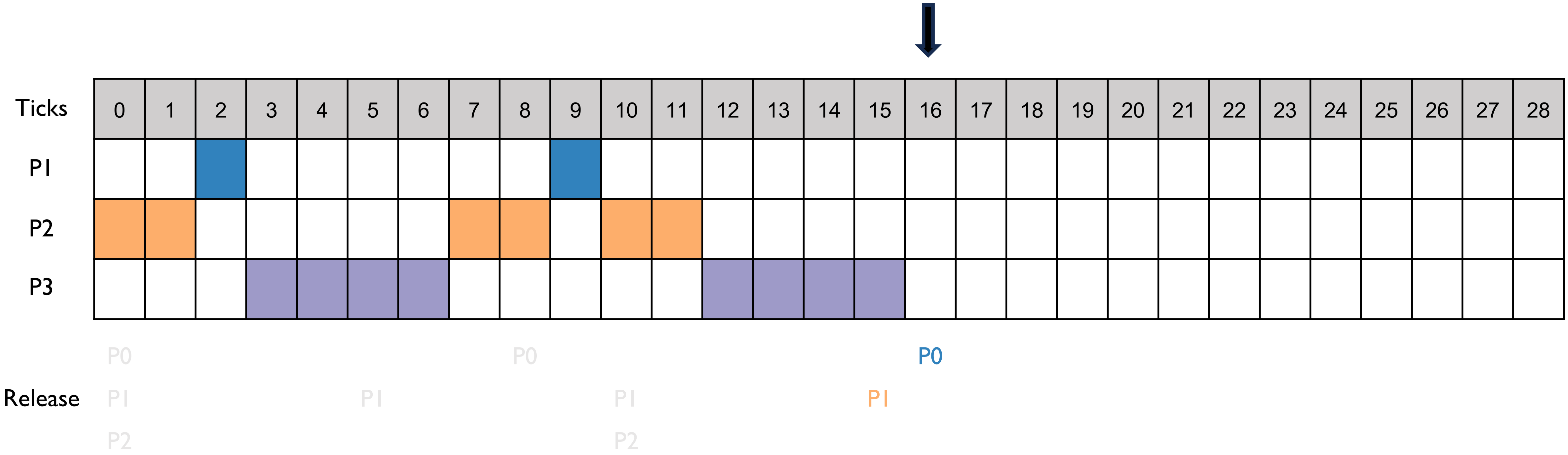
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



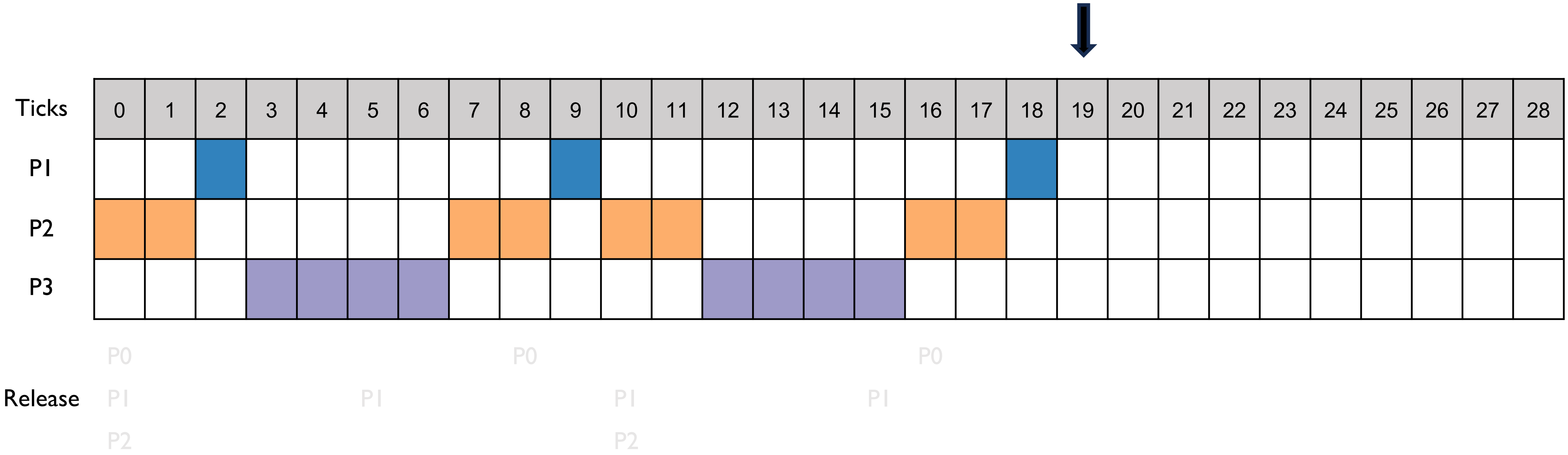
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



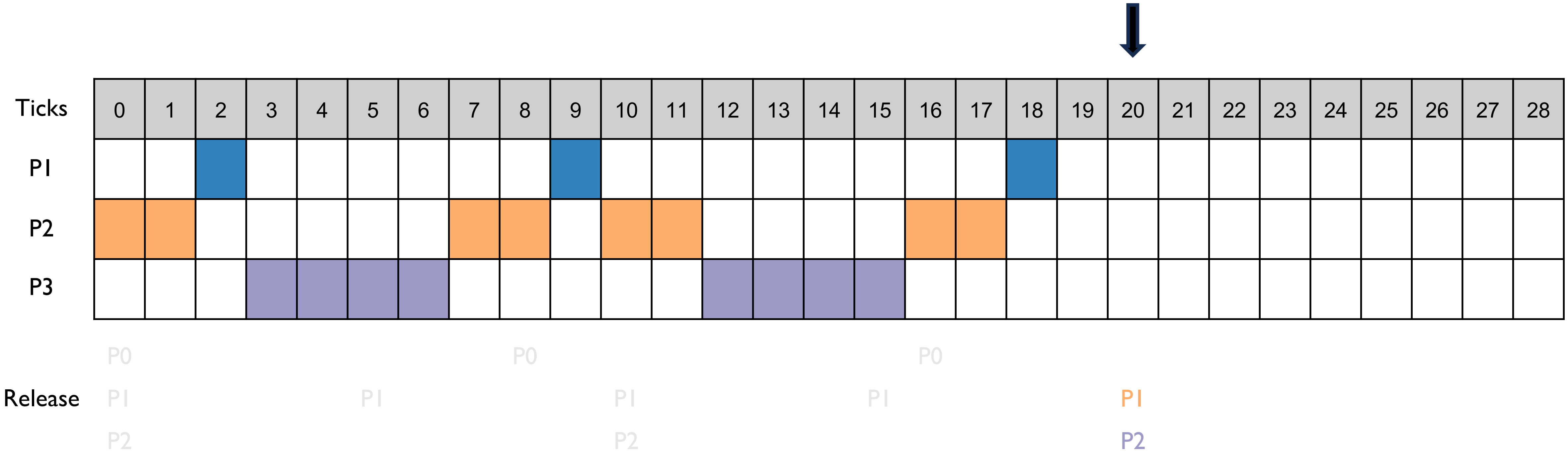
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



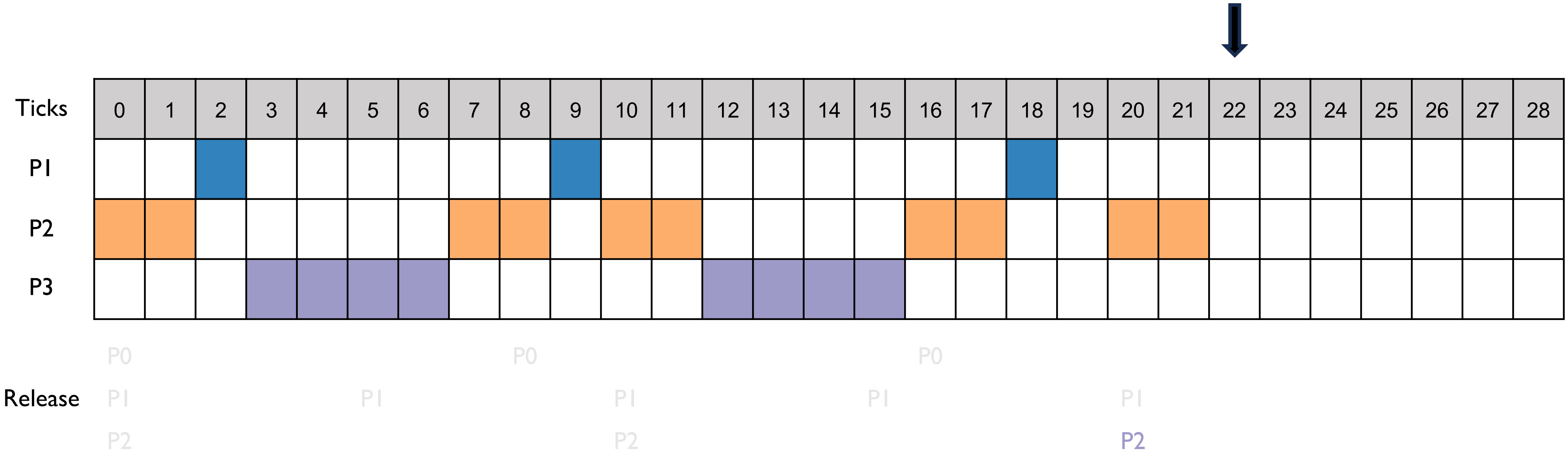
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



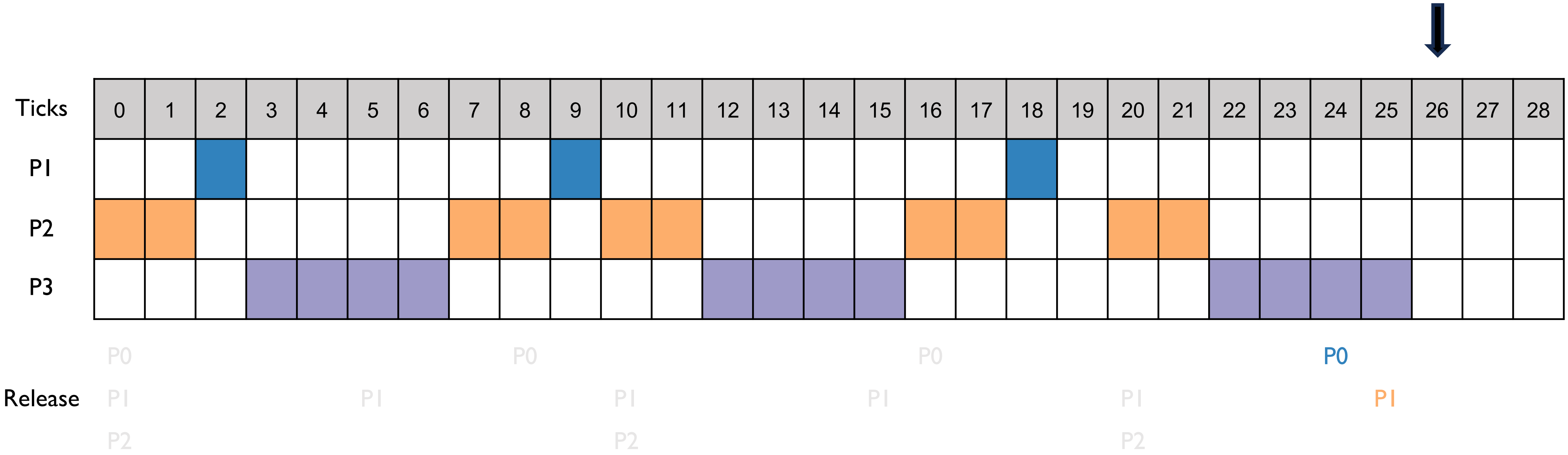
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



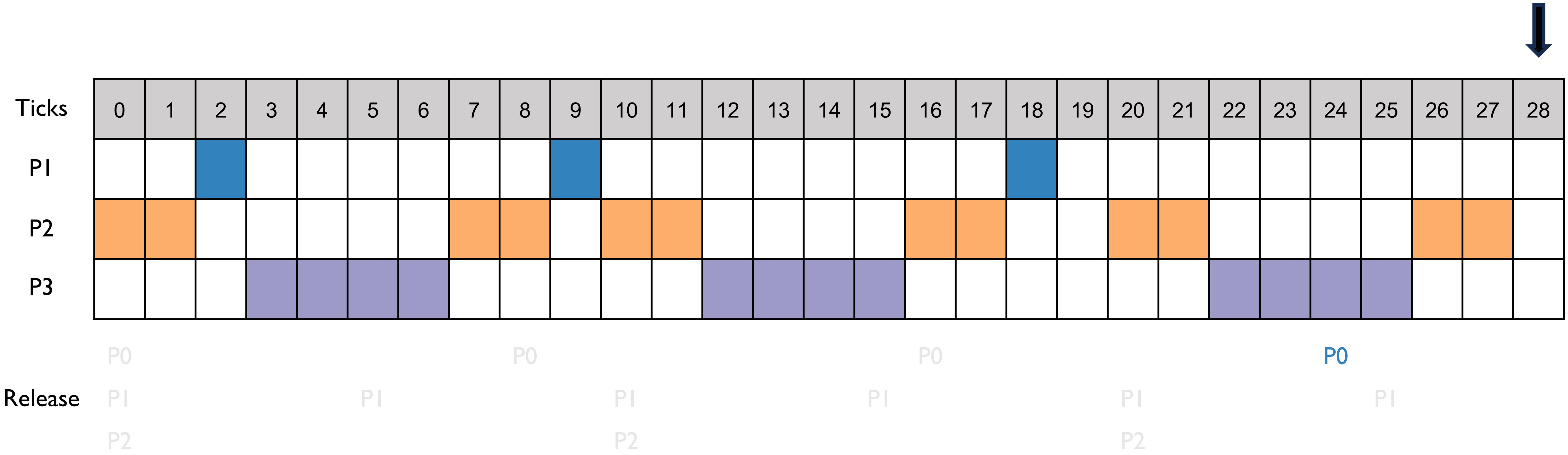
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



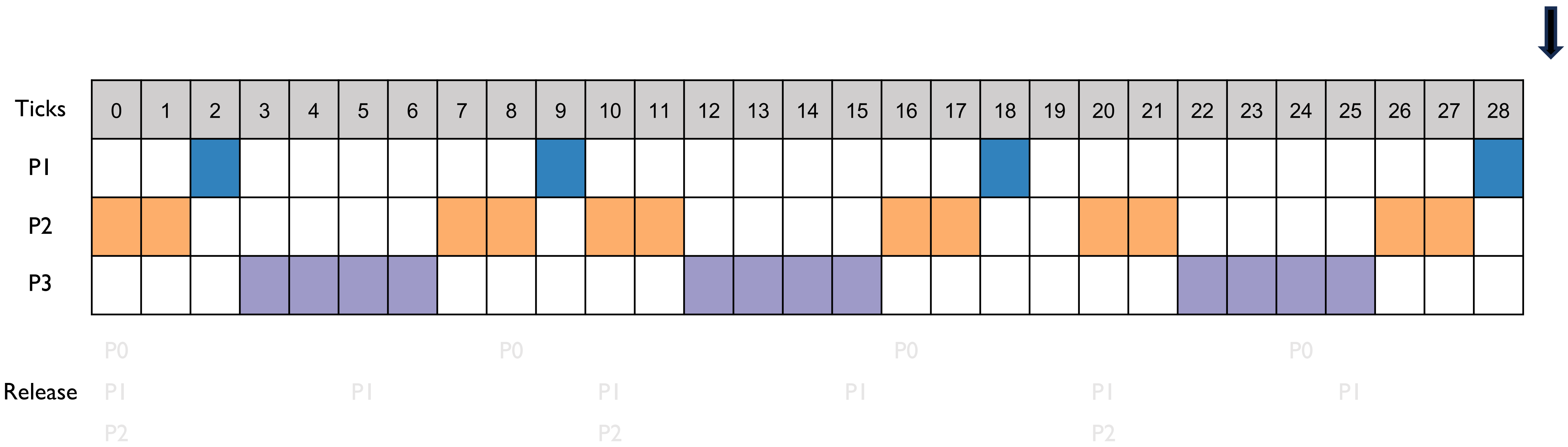
EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



EDF Example

- Scheduling 3 processes, P0 = (1, 8), P1 = (2, 5) and P2 = (4, 10)



Skeleton Code

- You should work on the pa3 branch as follows:

```
$ git clone https://github.com/snu-csl/xv6-riscv-snu
$ cd xv6-riscv-snu
$ git checkout pa3
```

- Then, you have to set your STUDENTID in the Makefile

- Also, you should install python matplotlib packages.

```
$ sudo apt install python3-matplotlib
```

Skeleton Code

- The pa3 branch includes two user-level programs, "task1” and “task2”
  - The first column shows the current time as measured by the time() system call
    - Time is displayed in units of 0.1 tick (E.g. a value of 110 corresponds to 11.0 ticks)
  - The second column represents the PID of the process

```
xv6 kernel is booting

init: starting sh
$ task1
110 4 starts
118 4 ends
118 5 starts
120 4 starts
…
```

  - Note that the initial tick number may vary depending on when you execute the program

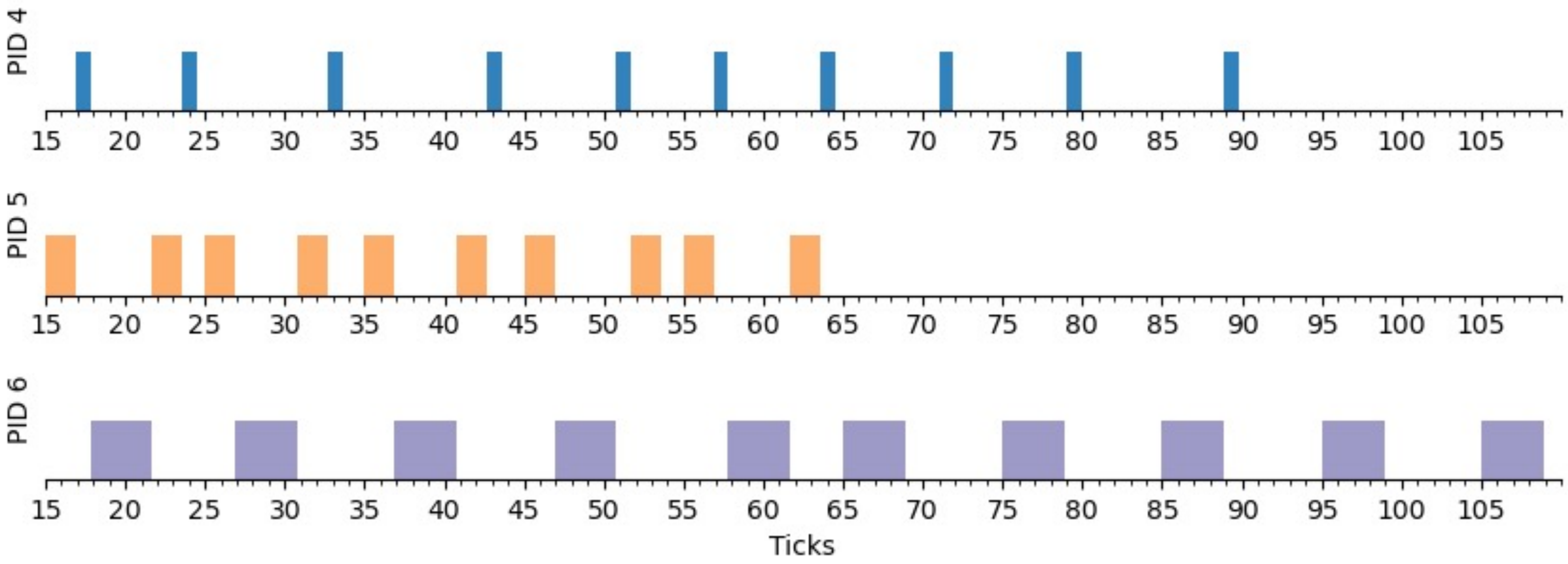
Drawing Scheduling Graph

- We provide you with a Python script called graph.py
- You can use this Python script to convert the above xv6 output into a graph image

```
$ make qemu-log
qemu-system-riscv64 -machine virt … | tee xv6.log

xv6 kernel is booting

init: starting sh

$ task2                    <--- Run the task2 program
110 4 starts
118 4 ends
...
QEMU: Terminated        <--- Quit qemu using ^a-x
*** The output of xv6 is logged in the 'xv6.log' file.

$ make png                 <--- Generate the graph. (this should be done on Ubuntu, not on xv6)
./graph.py xv6.log graph.png
```

Drawing Scheduling Graph

- If everything goes fine, you will get the following graph:



Restrictions

- The number of CPUs is already set to 1 in the Makefile
- You can assume that
  - The actual execution time of a real-time process is always less than its worst-case runtime
  - Real-time processes perform no I/O operations
- The implementation of the EDF scheduler should not affect the functionality of the default round-robin scheduler
- You only need to modify those files in the ./kernel directory
  - Changes to other source code will be ignored during grading.
- Please remove all the debugging outputs before you submit

Tips

- You may want to consult:
  - kernel/proc.{c, h}
    - Process related function handling
  - kernel/sysproc.c
    - Several system call implementations
  - kernel/trap.c
    - Trap handling

  - And other files if necessary

Submission

- Perform the make submit command to generate a compressed tar file
- Upload this tar file to the submission server
- The total number of submissions will be limited to 30
- Only the version marked FINAL will be considered
- It takes a long time to grading, so please wait for a few minutes

Thank you!