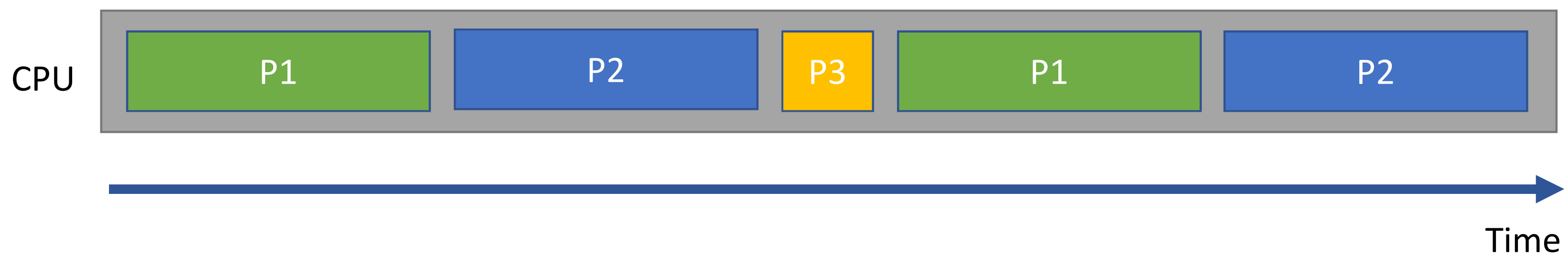
CS 1550

Week 7 – Priority Scheduling with xv6

Teaching Assistant
Xiaoyu(Veronica) Liang

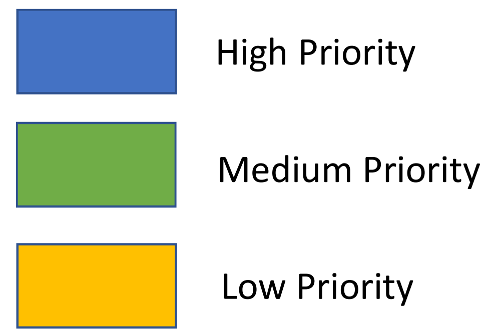
Scheduling of processes

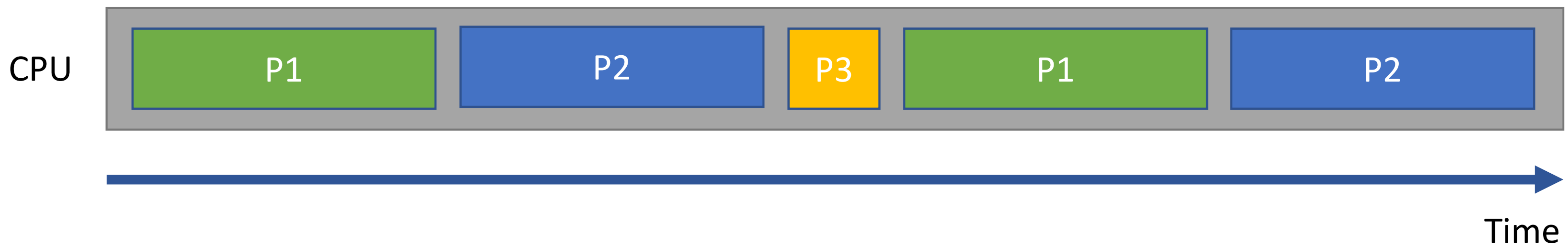
- Switch processes during their execution.
- Currently, processes run in Round Robin inside your XV6



Priority scheduling of processes

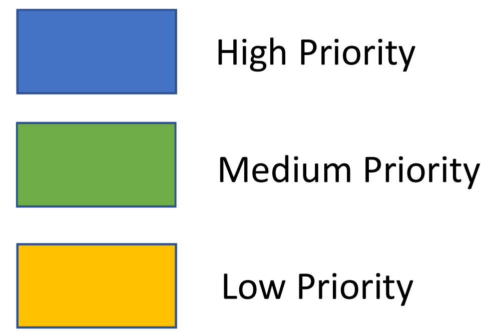
- What if processes have different priorities?
- Is this a better arrangement of processes?

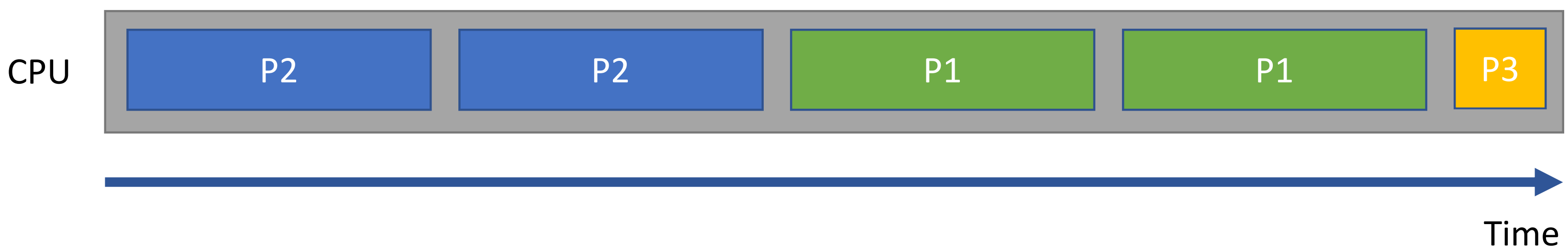




Priority scheduling of processes

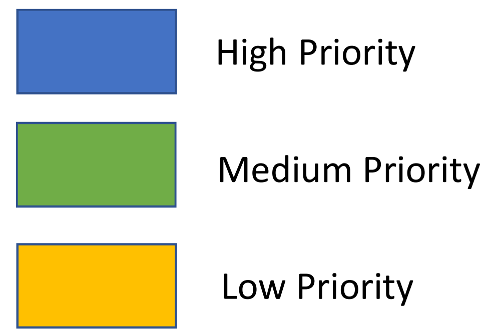
- Let all the higher priority processes finish before moving to lower priority ones

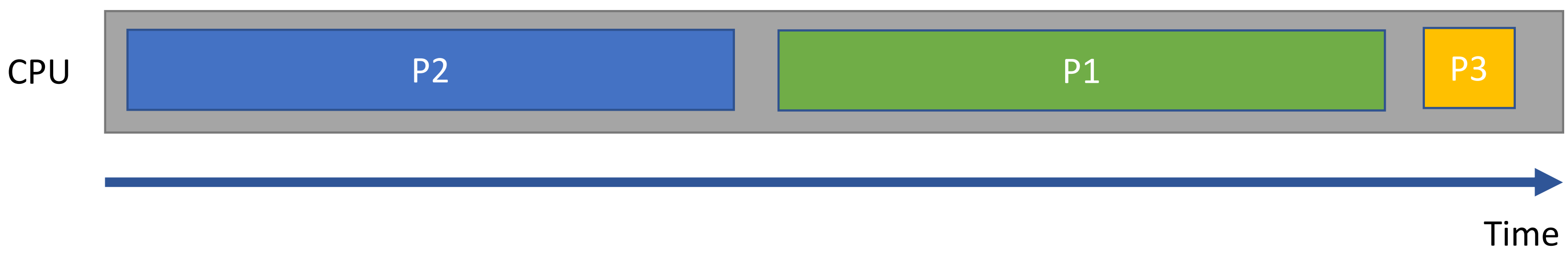




Priority scheduling of processes

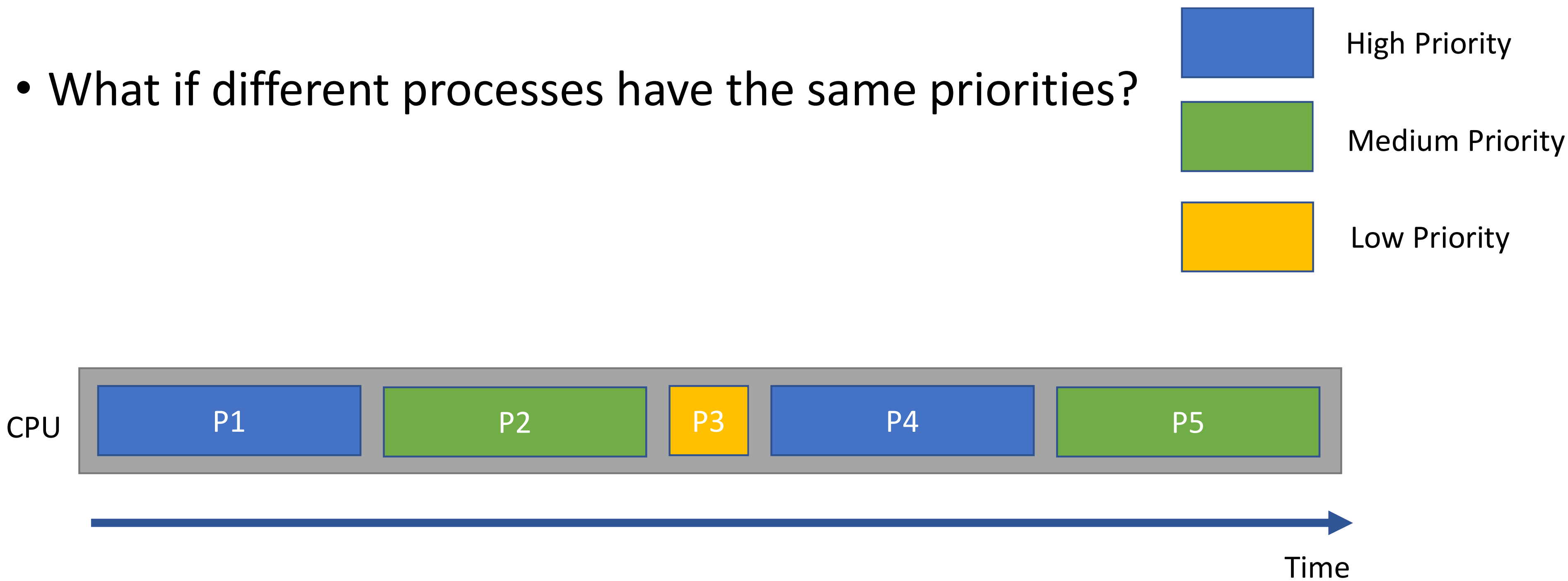
- Even better: Don’t yield if the current process is the **only one** of its priority
- This is the **bonus** part of your lab





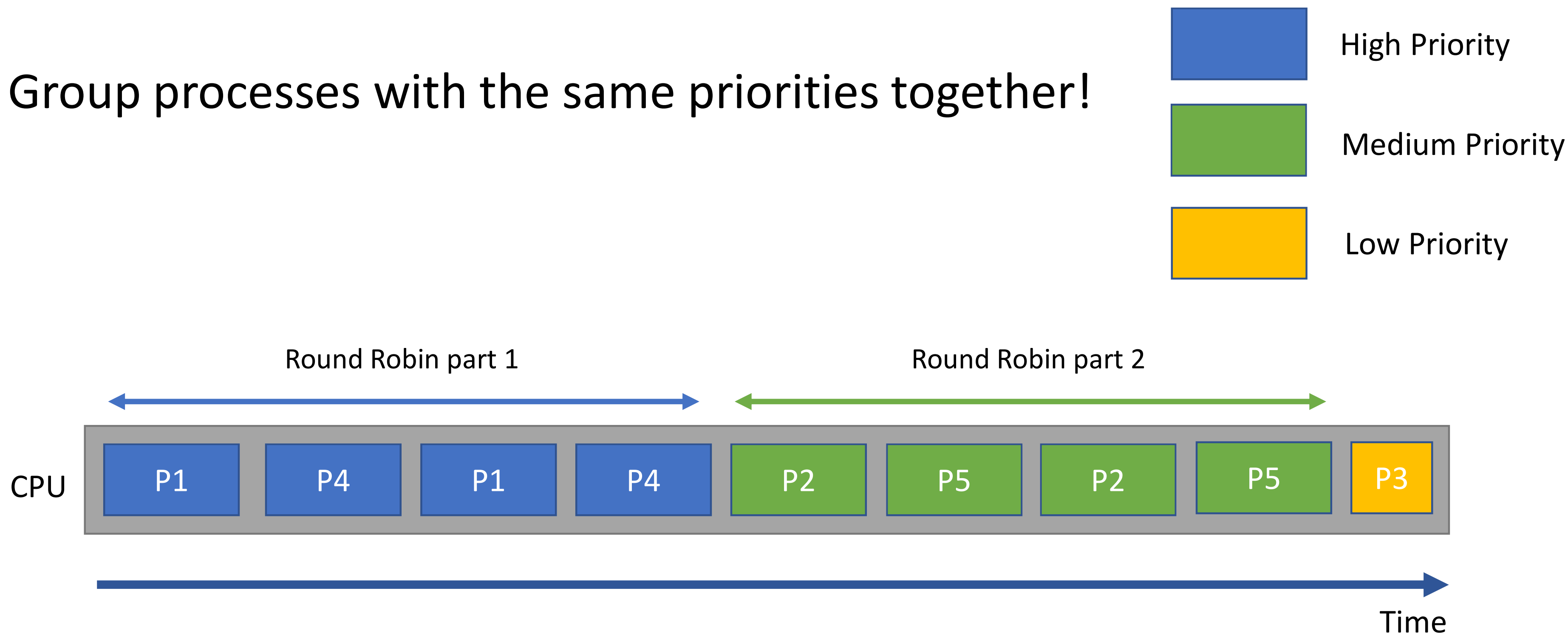
Processes with same priorities

- What if different processes have the same priorities?



Processes with same priorities

- Group processes with the same priorities together!



The scheduler function

- Per-CPU process scheduler.
- Each CPU calls scheduler() after setting itself up.
- Scheduler never returns. It loops, doing:
  - find a process to run
  - Run it until it stops
  - repeat

In proc.c:

```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;        Frans Kaashoek (2 years ago) · Eliminate code

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
      if(p->state != RUNNABLE)
        continue;

      // Switch to chosen process.  It is the process's job
      // to release ptable.lock and then reacquire it
      // before jumping back to us.
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;

      swtch(&(c->scheduler), p->context);
      switchkvm();

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

  }
}
```

Sets the per-CPU current process variable proc

Marks the process as RUNNING

Switches to the process's page table

Calls **swtch** to start running it

Yield in trap

- Yield:
  - Acquire the process table lock ptable.lock
  - Release any other locks it is holding
  - Update its own state (proc->state)
  - Call sched
- Force process to give up CPU on clock tick.
- IRQ stands for Interrupt Requests

In trap.c:

```c
if(myproc() && myproc()->state == RUNNING &&
   tf->trapno == T_IRQ0+IRQ_TIMER)
  yield();
```

In proc.c:

```c
// Give up the CPU for one scheduling round.
void
yield(void)
{
  acquire(&ptable.lock);  //DOC: yieldlock
  myproc()->state = RUNNABLE;
  sched();
  release(&ptable.lock);
}
```

Lab 3 – part 1: priority-based scheduler for XV6

- The valid priority for a process is in the range of 0 to 200.
- The smaller value represents the higher priority.
- Default priority for a process is 50.
- proc.h:
  - Add an **integer** field called ***priority*** to struct proc.
- proc.c:
  - allocproc function:
    - Set the default priority for a process to 50
  - Scheduler function:
    - Replace the scheduler function with your implementation of a priority-based scheduler.

Lab 3 – part 2: add a syscall to set priority

- Add a new syscall, ***setpriority***, for the process to change its priority.
- Changes the current process's priority and returns the old priority.
- Review lab1 to refresh steps to add a new syscall.

Lab 3 is out!

- Due: 11:59 PM Friday (March 8th)