Operating Systems Principles

CPU Management

CPU Scheduling (2), Process Synchronization

Slides are adapted from Prof. Bhuvan Urgaonkar

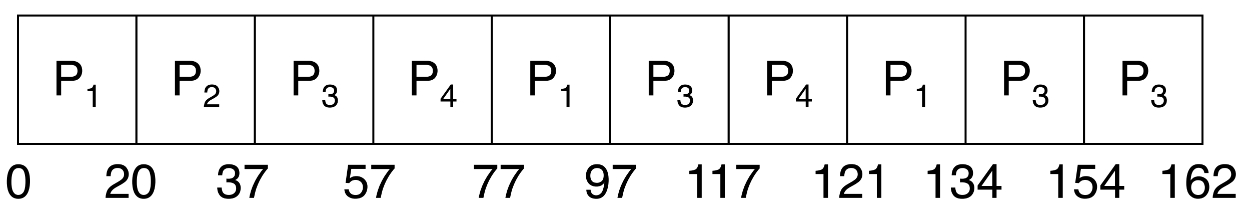
Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large => FCFS
  - q small => q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 20

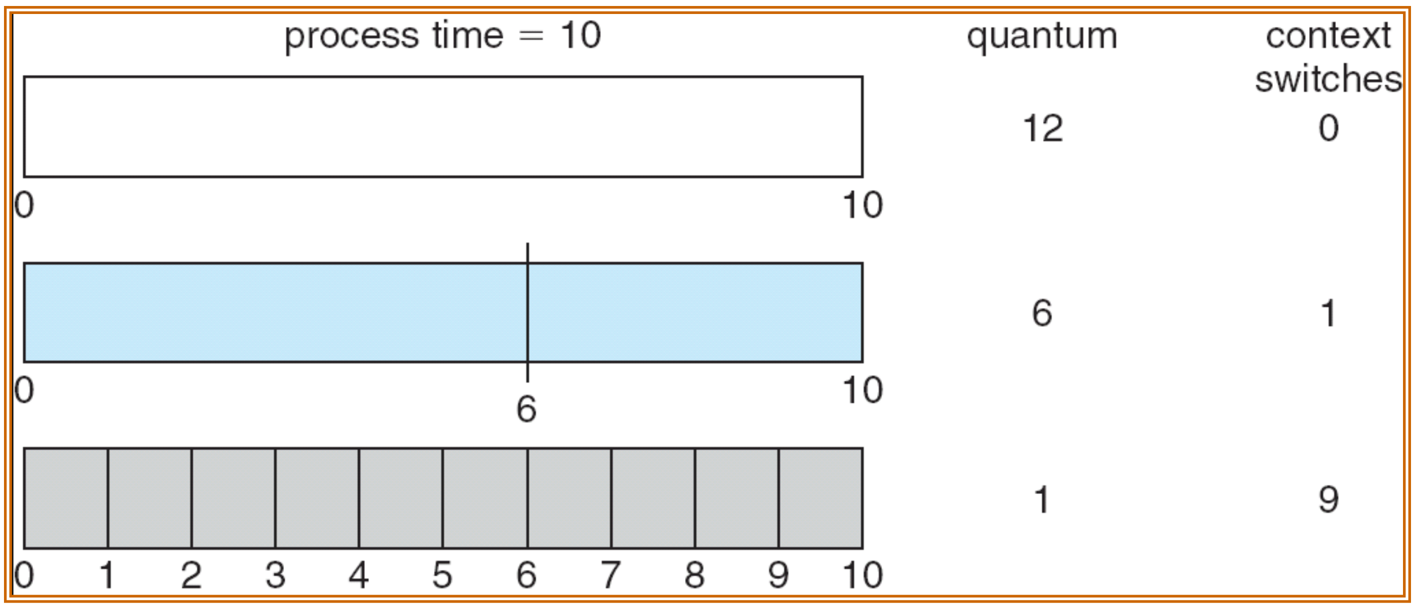
Process	CPU Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:

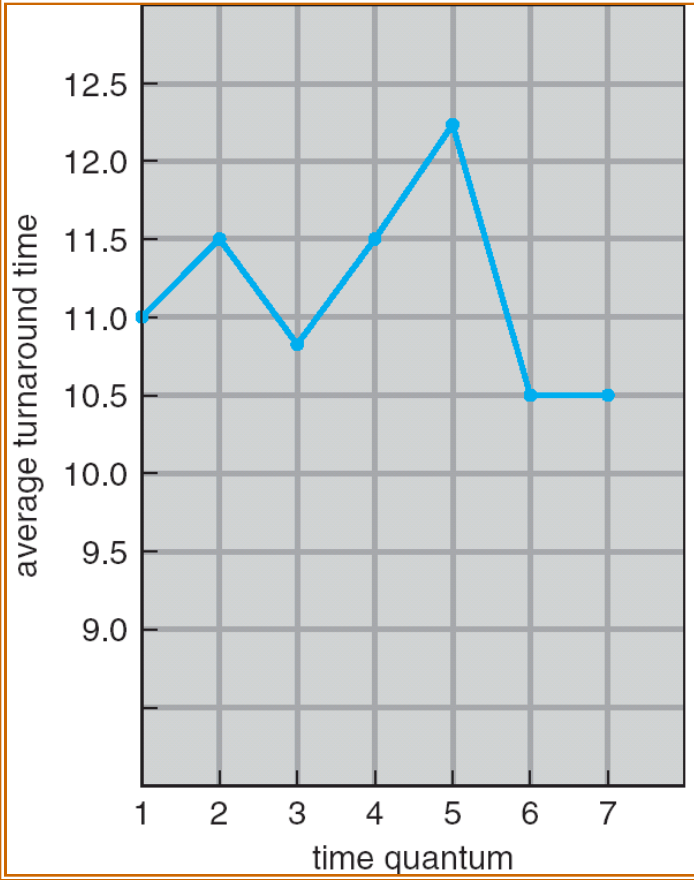


- Typically, higher average turnaround than SJF, but better *response*

Time Quantum and Context Switch Time



Turnaround Time Varies With Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

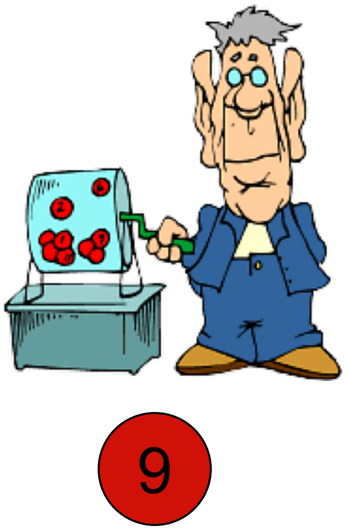
Proportional-Share Schedulers

- A generalization of round robin
- Process P_i given a CPU weight $w_i > 0$
- The scheduler needs to ensure the following
  - *For all* i, j, $|T_i(t_1, t_2)/T_j(t_1,t_2) - w_i/w_j| \leq e$
  - Given P_i and P_j were backlogged during $[t_1,t_2]$
- Que: Who chooses the weights and how?

Lottery Scheduling

- Perhaps the simplest proportional-share scheduler
- Create lottery tickets equal to the sum of the weights of all processes
- Draw a lottery ticket and schedule the process that owns that ticket

Lottery Scheduling Example



P1=6

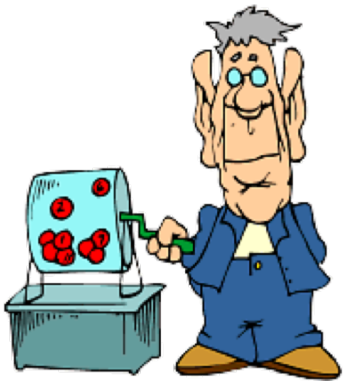
1	4
2	5
3	6

P2=9

7	10	13
8	11	14
9	12	15

Schedule P2

Lottery Scheduling Example



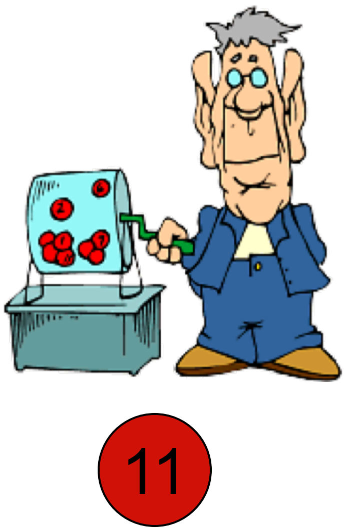
3

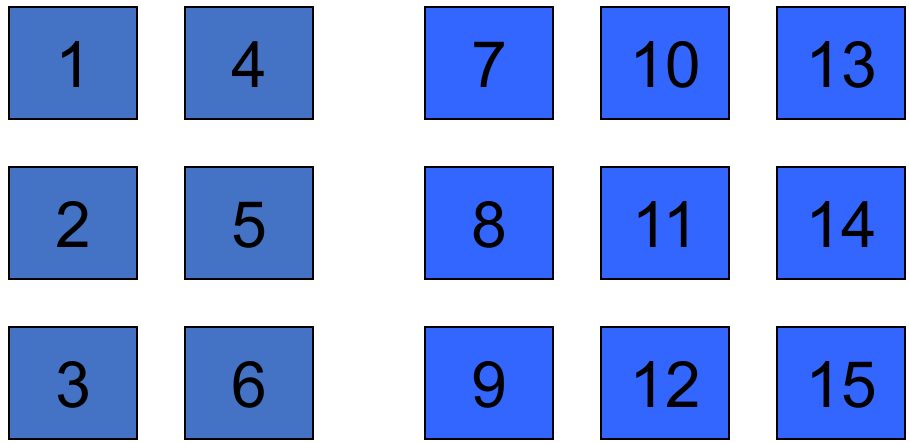
P1=6		*P2=9*		
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

Schedule P1

Lottery Scheduling Example

P1=6 *P2=9*





- As t $\longrightarrow \infty$, processes will get their share (unless they were blocked a lot)
- Problem with Lottery scheduling: Only probabilistic guarantee
- What does the scheduler have to do
  - When a new process arrives?
  - When a process terminates?

Schedule P2

Lottery Scheduling

- Exercise: Calculate the time complexity of the operations Lottery scheduling will involve

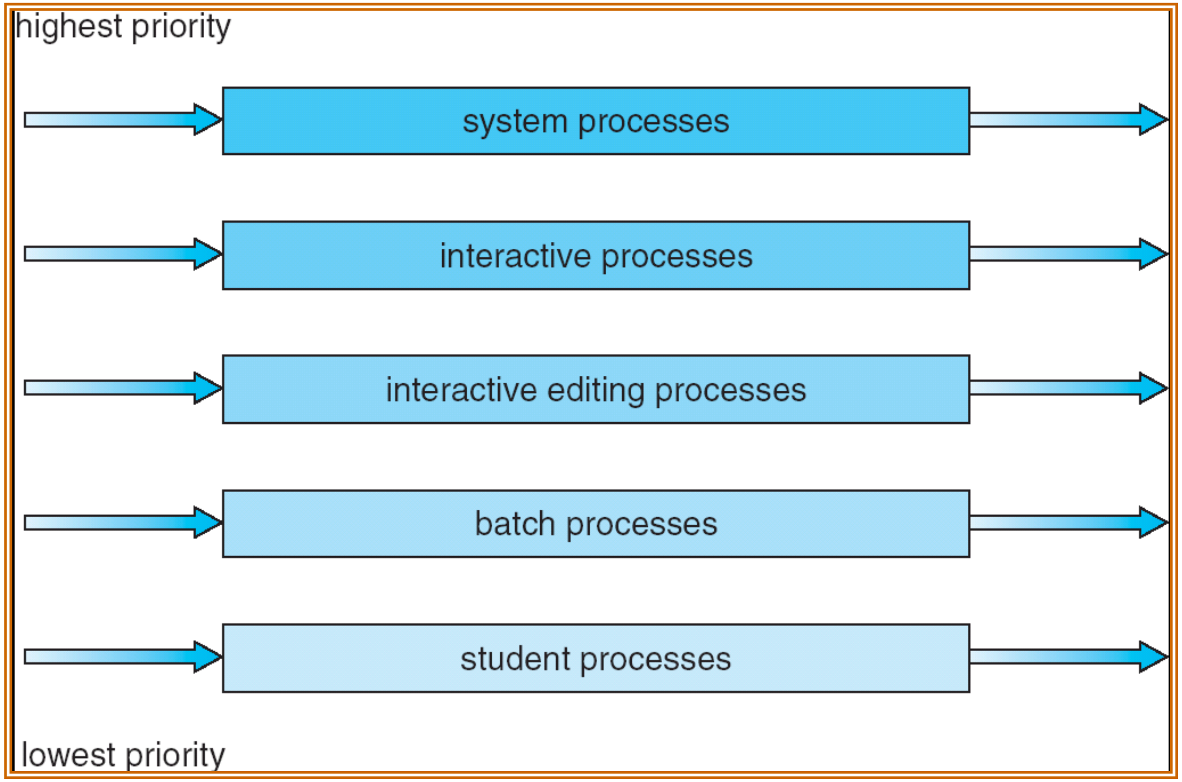
Priority-based Scheduling

- Associate with each process a quantity called its *CPU priority*
- At each scheduling instant
  - Pick the ready process with the highest CPU priority
  - Update (usually decrement) the priority of the process last running
    - Priority = Time since arrival => FCFS
    - Priority = 1/Size => SJF
    - Priority = 1/Remaining Time => SRPT
    - Priority = Time since last run => Round-robin (RR)
- UNIX variants
  - Priority values are positive integers with upper bounds
  - Decreased every quantum
    - Fairness, avoid starvation
  - Increased if the process was waiting, more wait => larger increase
    - To make interactive processes more responsive
  - Problems
    - Hard to analyze theoretically, so hard to give any guarantees
    - May unfairly reward blocking processes

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- Each queue has its own scheduling algorithm
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues
  - Fixed priority scheduling; (i.e., serve all from foreground then from background).
    - Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes;
    - i.e., 80% to foreground in RR, 20% to background in FCFS

Multilevel Queue Scheduling



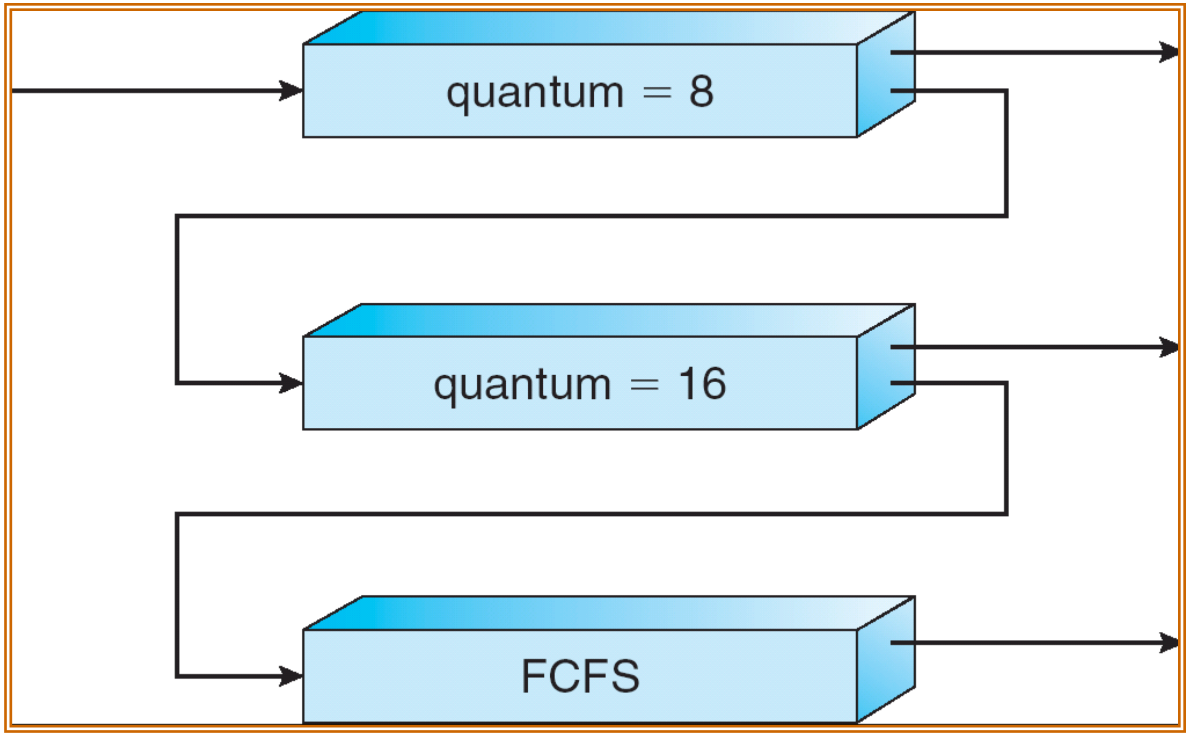
Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



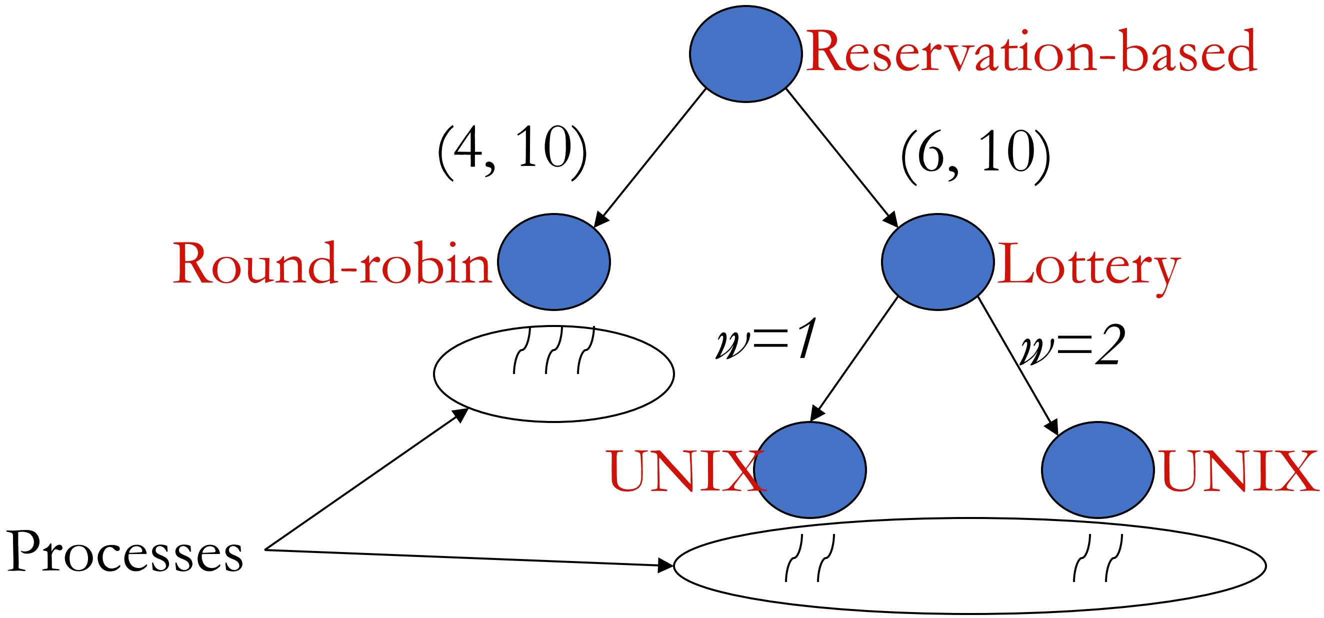
Work Conservation

- Examples of work-conserving schedulers: All schedulers we have studied so far
  - Work conservation: The resource (e.g., CPU) can not be idle if there is some work to be done (e.g., at least one ready process)
- Example of a non-work-conserving scheduler:
  - Reservation-based (coming up)

Reservation-based Schedulers

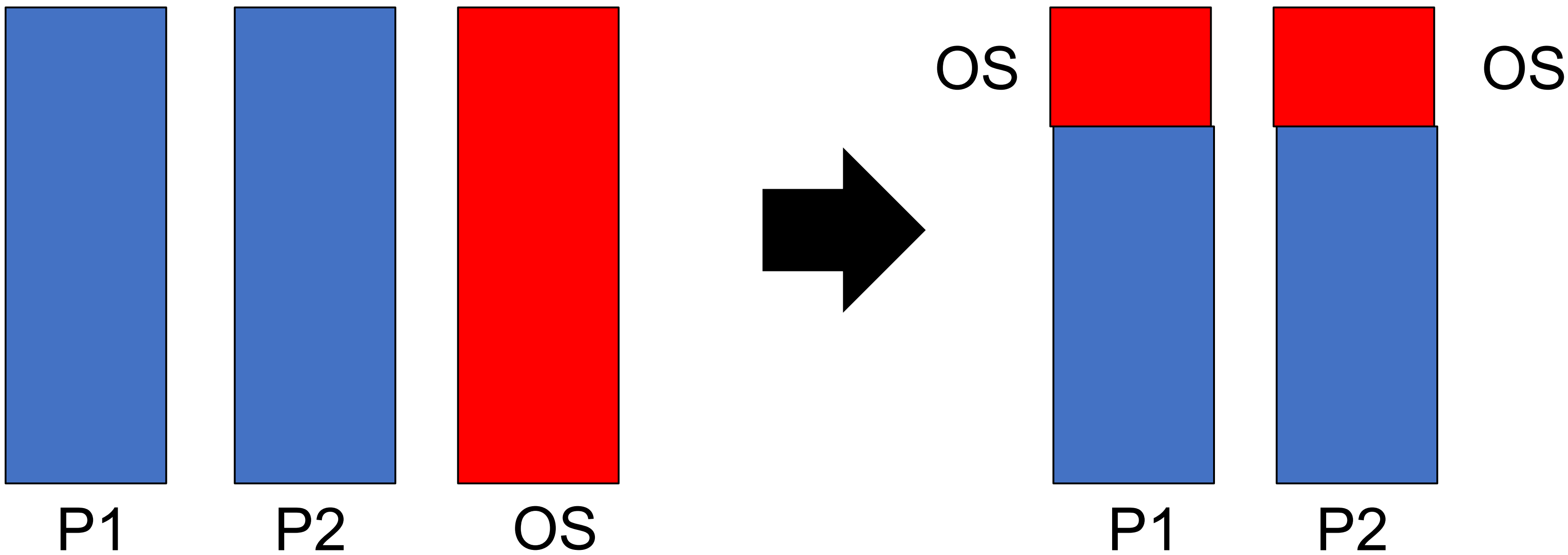
- Each process has a pair (x, y)
  - Divide time into periods of length y each
  - Guaranteed to get x time units every period
- Why is this NWC?
- Why design a NWC scheduler?

Hierarchical Schedulers



- A way to compose a variety of schedulers
- Sets of processes with different scheduling needs
- Exercise: Think of conditions under which a tree of schedulers becomes NWC

Address Spaces: A Refinement

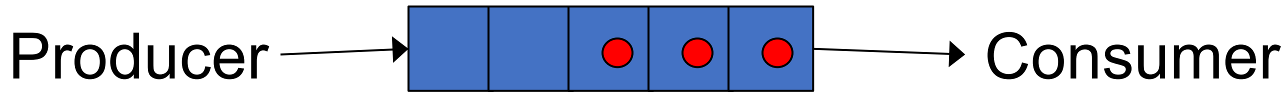


- The OS is made part of the address space of every process
- Why?
  - User/kernel transitions are not context switches anymore!
  - Need to disallow virtual address range of OS from being accessed by processes -> MMU helps with this

Process Synchronization

- We will study it in the context of the shared memory model

Motivating Example: Producer and Consumer




```
while (true) {
   /* produce an item and put in nextProduced  */
        while (count == BUFFER_SIZE); // do nothing

       buffer [in] = nextProduced;
       in = (in + 1) % BUFFER_SIZE;
       count++;
}
```

```
while (true)  {
         while (count == 0);  // do nothing

          nextConsumed =  buffer[out];
          out = (out + 1) % BUFFER_SIZE;
          count--;
      /*  consume the item in nextConsumed
   }
```

- Shared variables: buffer and count
- Local variables: in, out, nextProduced, nextConsumed