Lecture 8

Jan 18, 2024

Priority Based Scheduling Algorithms

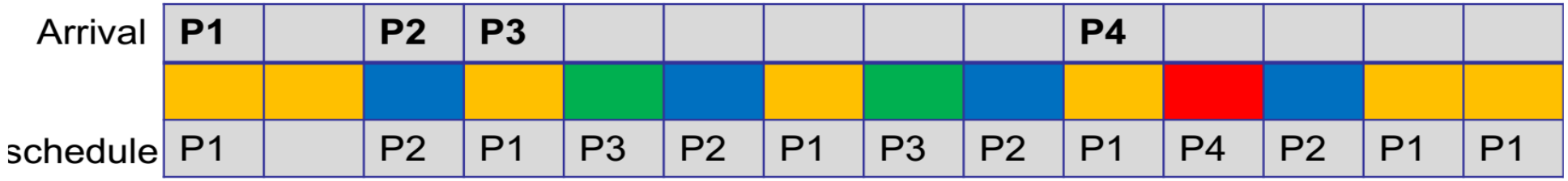
Relook at Round Robin Scheduling

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1

Time slice = 1

Process P2 is a critical process while process P1, P3, and P4 are less critical

Process P2 is delayed considerably



Priorities

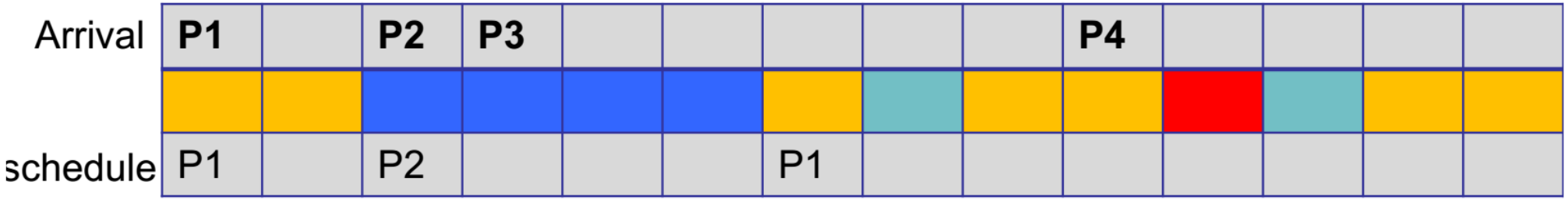
Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	3	2
P4	9	1

Time slice = 1

Process P2 is a critical process while process P1, P3, and P4 are less critical

We need a higher priority for P2, compared to the other processes

This leads to priority based scheduling algorithms



- Priority based Scheduling
  - Each process is assigned a priority
    - A priority is a number in a range (for instance between 0 and 255)
    - A small number would mean high priority while a large number would mean low priority
  - Scheduling policy : pick the process in the ready queue having the highest priority
  - Advantage : mechanism to provide relative importance to processes
  - Disadvantage : could lead to starvation of low priority processes

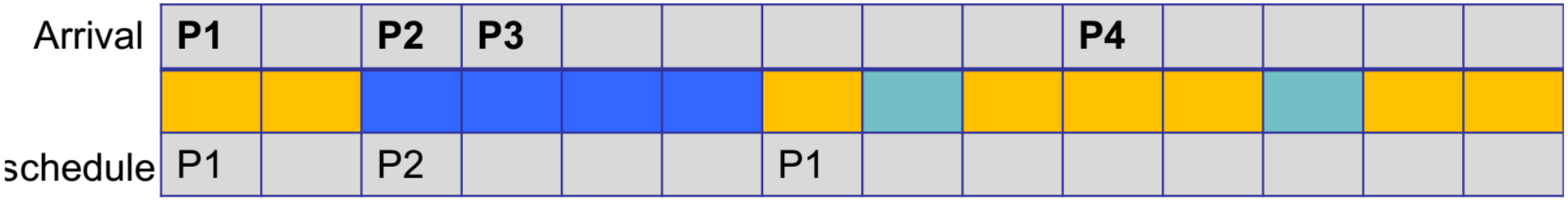
Starvation

Process	Arrival Time	Burst Time
P1	0	8
P2	2	4
P3	3	2
P4	9	1

Time slice = 1

Low priority process may never get a chance to execute.

P4 is a low priority process



Dealing with starvation

- Scheduler adjusts priority of processes to ensure that they all eventually execute
- Several techniques possible. For example,
  - Every process is given a base priority
  - After every time slot increment the priority of all other process
    - This ensures that even a low priority process will eventually execute
  - After a process executes, its priority is reset

Priority Types

- Static priority : typically set at start of execution
  - If not set by user, there is a default value (base priority)
- Dynamic priority : scheduler can change the process priority during execution in order to achieve scheduling goals
  - eg1. decrease priority of a process to give another process a chance to execute
  - eg.2. increase priority for I/O bound processes

Scheduling trade-offs

- FCFS, SJF, STCF optimise turnaround time at the expense of response time
- RR optimises response time at the expense of turnaround time, has more context switches
- FP prioritises jobs based on prior knowledge, but does not avoid starvation of lower priority jobs

- *How can we design a scheduler that minimizes both the response time for interactive jobs and turnaround time without prior knowledge of job length/type*

Homework -3

- **Lottery Scheduling**
- What is it ?
- What metric does it optimize ?
- How would you compare it with other algos ?

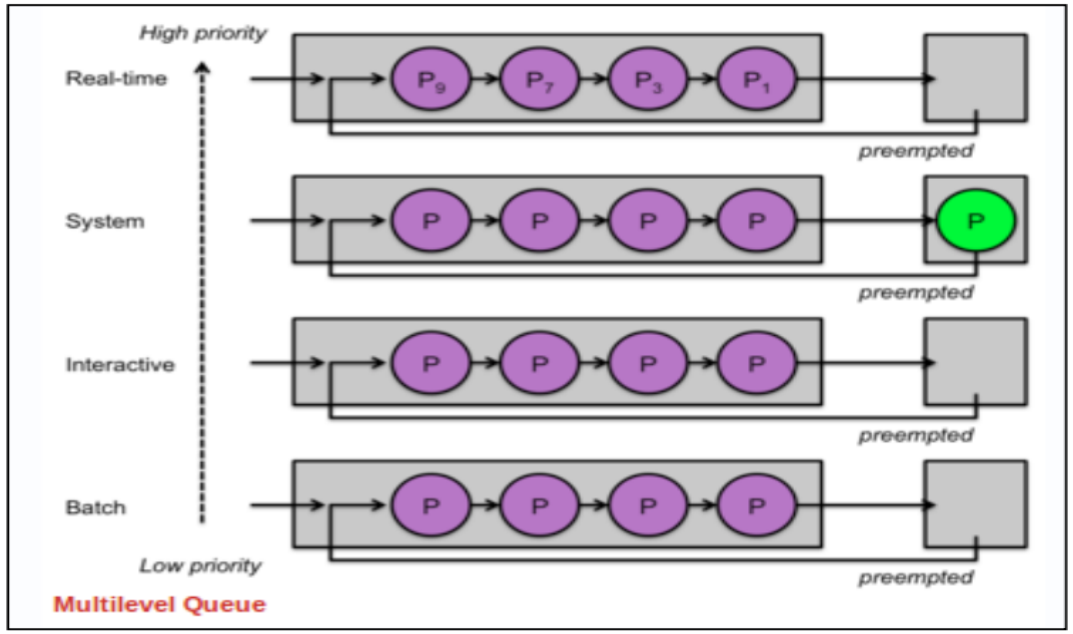
Schedulers in real systems

- Real schedulers are more complex
- •For example, Linux uses a Multi Level FeedbackQueue (MLFQ)
- –Many queues, in order of priority
- –Process from highest priority queue scheduled first
- –Within same priority, any algorithm like RR
- –Priority of process decays with its age

- First ideas proposed by Corbato already in 1962. He received the Turing Award for his contributions.
- Idea 1: Have a hierarchical policy. Top level is FP to ensure prioritisation. Bottom level may be RR to ensure lower response times -- Multilevel
- Idea 2: For each job observe its behaviour and dynamically adjust its scheduling parameters -- Feedback

Multilevel Queues

- Processes assigned to a priority classes
- Each class has its own ready queue
- Scheduler picks the highest priority queue (class) which has at least one ready process
- Selection of a process within the class could have its own policy
  - Typically round robin (but can be changed)
  - High priority classes can implement first come first serve in order to ensure quick response time for critical tasks



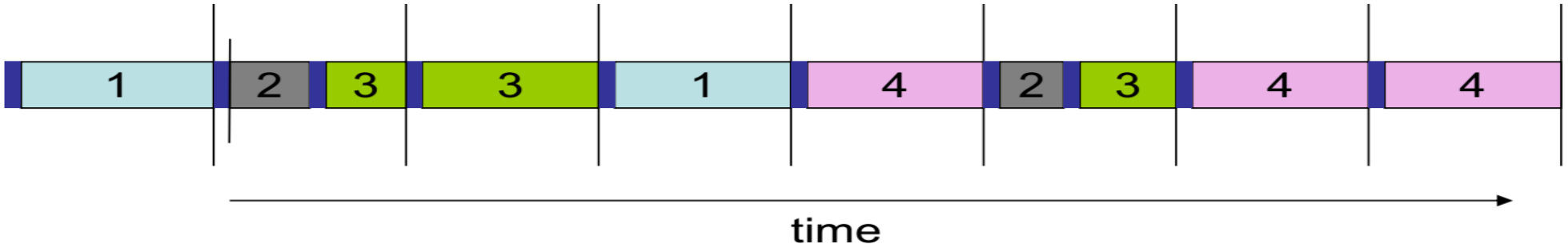
More on MLQs

- Scheduler can adjust time slice based on the queue class picked
  - I/O bound process can be assigned to higher priority classes with longer time slice
  - CPU bound processes can be assigned to lower priority classes with shorter time slices
- Disadvantage :
  - Class of a process must be assigned apriori

    (not the most efficient way to do things!)

Multilevel feedback Queue

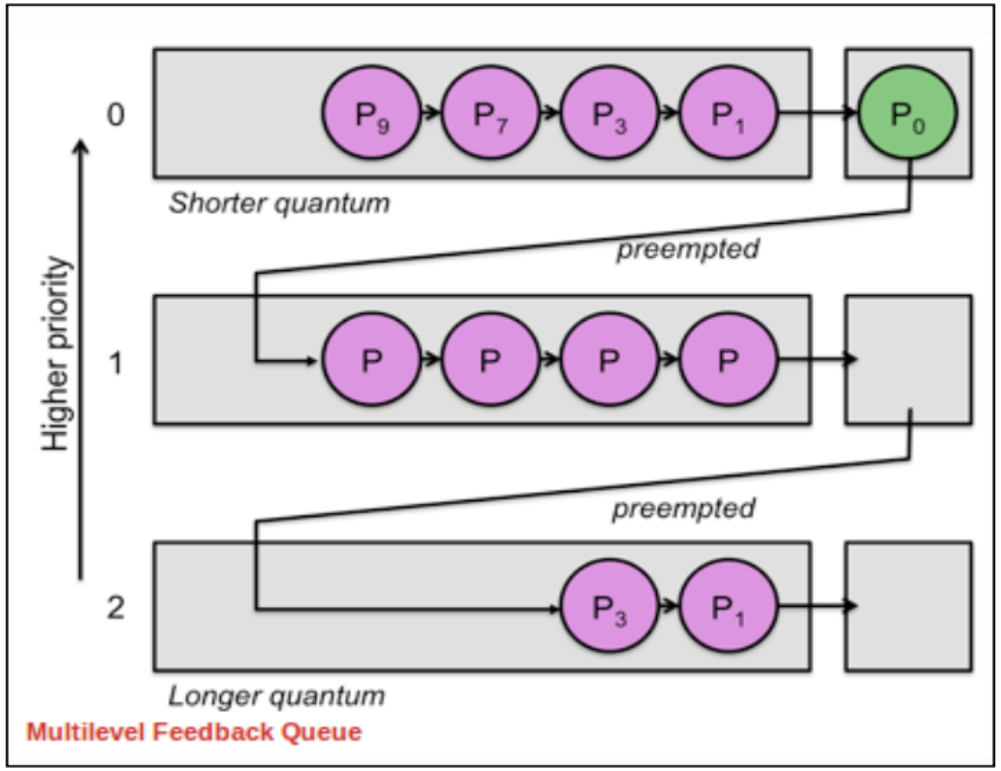
- Process dynamically moves between priority classes based on its CPU/ IO activity
- Basic observation
  - CPU bound process' likely to complete its entire timeslice
  - IO bound process' may not complete the entire time slice



Process 1 and 4 likely CPU bound
Process 2 likely IO bound

Basic Idea

- All processes start in the highest priority class
- If it finishes its time slice (likely CPU bound)
  - Move to the next lower priority class
- If it does not finish its time slice (likely IO bound)
  - Keep it on the same priority class
- As with any other priority based scheduling scheme, starvation needs to be dealt with



All jobs have c=4, a=0

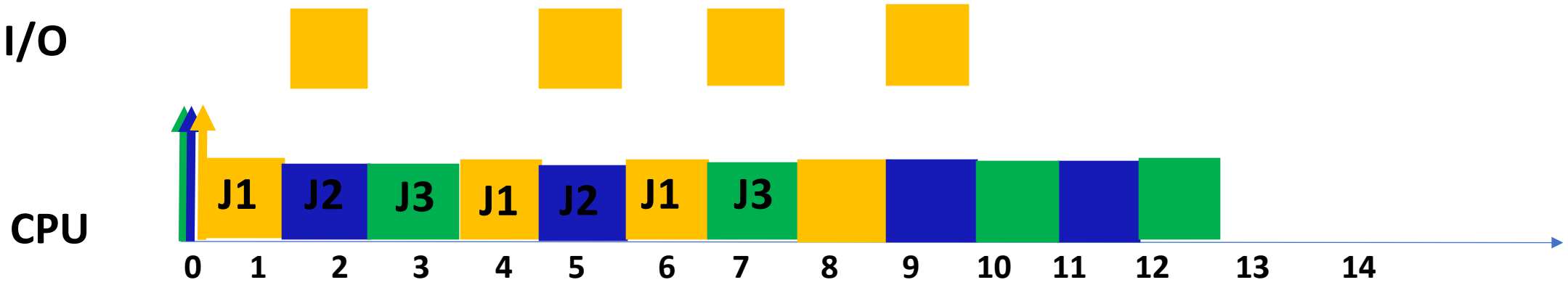
J1 accesses IO for 1 unit every 1 unit

J1, J2, J3 are in the Q1 and run under RR for time slice = 1

J1 gives up CPU; hence remains in Q1

J2, J3 don't give up CPU and move to Q2

MLFQ



J1 runs with the highest priority

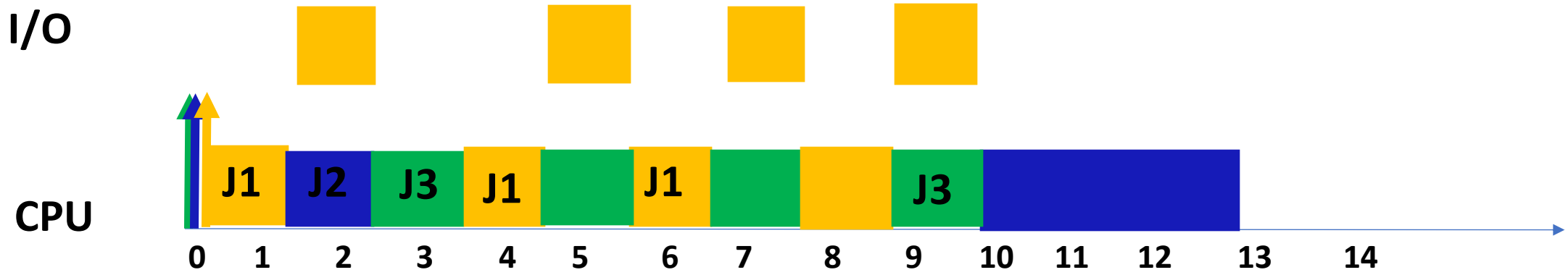
In the background, J2 and J3 run under RR

J2 and J3 will be further downgraded in priority, together

Gaming the System

J3 decides to act clever and voluntarily gives up access after executing for 0.999

How would the schedule look ?



J1, J3 remain in top priority in Q1

J2 moves down to Q2 and thus executes only after J1 and J3 finish

Solution

- Don’t just look at the binary state of whether a process has used up the entire time slice or not
- Use the amount of time executed instead
- if the total amount of time executed in a certain number of slots is exceeded, demote the priority

One last Problem

- **Starvation**
- Long CPU intensive jobs will fall down to the lowest priority and may starve
- Solutions:
- Periodically boost priority of all jobs
- Allow user to provide 'suggestions' to the OS on how important a job is

Summary – Multi level Queues

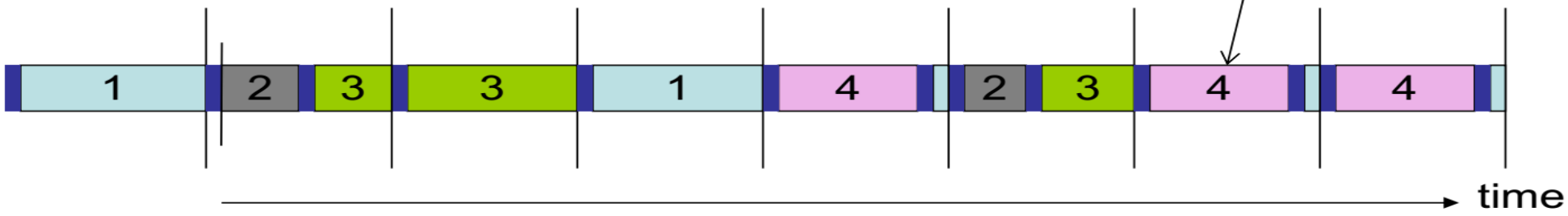
- • Multiple Queues at various levels
- • Static Priority, base priority
- • Dynamic priority set based on some heuristics
- – IO bound processes should have a higher priority than CPU bound processes
- • Timeslice changed dynamically based on heuristics
- – IO bound processes should get a longer timeslice than CPU bound processes
- • Starvation dealt with
- – Every process, even the lowest priority process should execute.

- A compute intensive process can trick the scheduler and remain in the high priority queue (class)

```
while(1){
    do some work for most of the time slice
    sleep(till the end of the time slice)
}
```

Sleep will force a context switch

Process 4 is gaming the system



Scheduling in Linux

- Real time
  - Deadlines that have to be met
  - Should never be blocked by a low priority task
- Normal Processes
  - Interactive
    - Constantly interact with their users, therefore spend a lot of time waiting for key presses and mouse operations.
    - When input is received, the process must wake up quickly (delay must be between 50 to 150 ms)
  - Batch
    - Does not require any user interaction, often runs in the background.

Once a process is specified real time, it is always considered a real time process

A process may act as an interactive process for some time and then become a batch process.

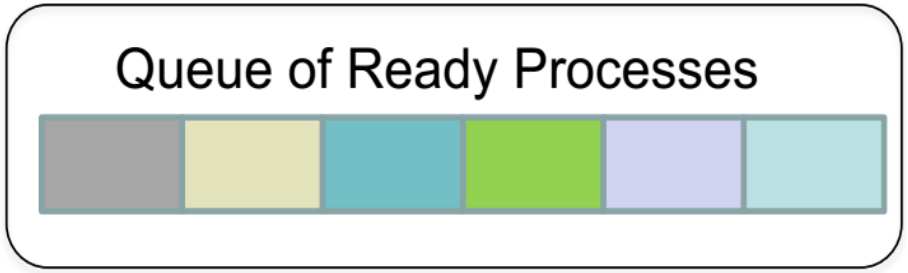
Linux uses sophisticated heuristics based on past behavior of the process to decide whether a given process should be considered interactive or batch

Normal Process Schedulers

- O(n) scheduler
  - Linux 2.4 to 2.6
- O(1) scheduler
  - Linux 2.6 to 2.6.22
- CFS scheduler
  - Linux 2.6.23 onwards

O(n) Scheduler

- At every context switch
  - Scan the list of runnable processes
  - Compute priorities
  - Select the best process to run



- O(n), when n is the number of runnable processes … not scalable!!
  - Scalability issues observed when Java was introduced (JVM spawns many tasks)
- Used a global run-queue in SMP systems
  - Again, not scalable!!

O(1) Scheduler

- Constant time required to pick the next process to execute
  - easily scales to large number of processes
- Processes divided into 2 types
  - Real time
    - Priorities from 0 to 99
  - Normal processes
    - Interactive
    - Batch
    - Priorities from 100 to 139 (100 highest, 139 lowest priority)

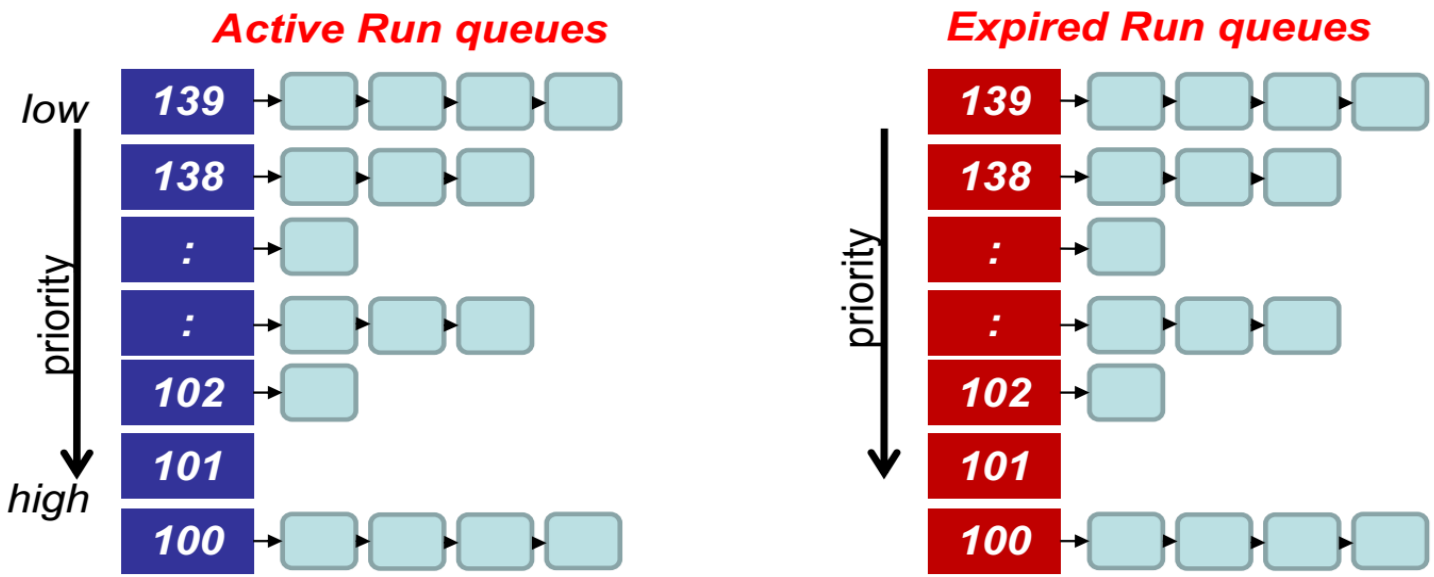
Follows the MLFQ method:

- Pick the highest priority (or level) with a non-empty queue of runnable process
- Pick first process from that priority level
- Is this O(1) - ??

Two Copies

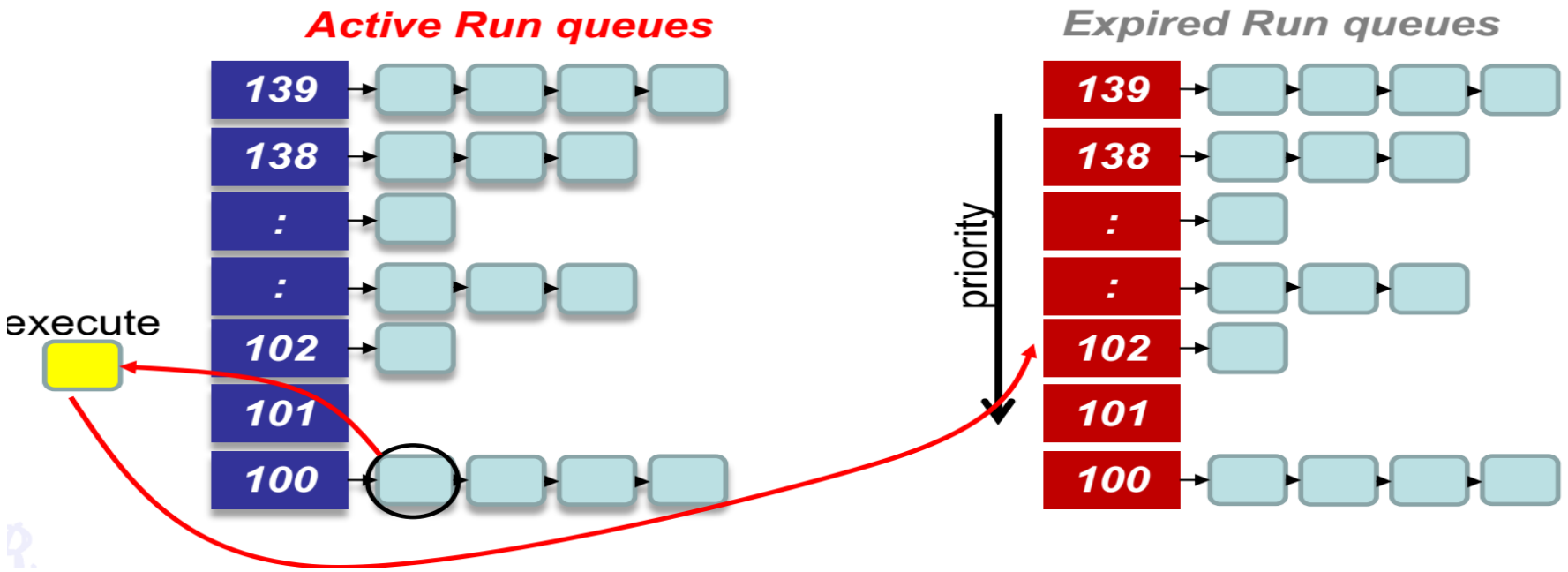
- O(1) scheduler maintains two copies of the MLFQ: **Active** and **Expired**
- Processes are run from **active queues** upon expiry of time slice they are moved to expired queues (not for interactive processes)
- After **all** processes in active queue are done, active and expired queues are swapped
- Aim : Do not touch each process in every context switch

- Two ready queues in each CPU
  - Each queue has 40 priority classes (100 – 139)
  - 100 has highest priority, 139 has lowest priority



Scheduling Policy

- Pick the first task from the lowest numbered run queue
- When done put task in the appropriate queue in the expired run queue



How it is constant time?

- There are 2 steps in the scheduling
  1. Find the lowest numbered queue with at least 1 task
  2. Choose the first task from that queue

- step 2 is obviously constant time
- Is step 1 contant time?
  - Store bitmap of run queues with non-zero entries
  - Use special instruction '*find-first-bit-set*'
    - *bsfl* on intel

Steps of the scheduler

- Set static priority at start
- Update dynamic priority at each context switch
- Decide qualification on "interactive" process
- Calculate timeslice at each context switch

Priority

- 0 to 99 meant for real time processes
- 100 is the highest priority for a normal process
- 139 is the lowest priority
- Static Priorities
  - 120 is the base priority (default)
  - **nice :** command line to change default priority of a process

    $nice –n N ./a.out
  - N is a value from +19 to -20;
    - most selfish ‘-20’; (I want to go first)
    - most generous ‘+19’; (I will go last)

Dynamic Priority

- To distinguish between batch and interactive processes
- Uses a 'bonus', which changes based on a heuristic

dynamic priority = MAX(100, MIN(static priority – bonus + 5), 139))

Has a value between 0 and 10

Click to add text

If bonus < 5, implies less interaction with the user
thus more of a CPU bound process.
The dynamic priority is therefore decreased (toward 139)

If bonus > 5, implies more interaction with the user
thus more of an interactive process.
The dynamic priority is increased (toward 100).

Recall from MLFQ why we need dynamic priority

Bonus is a number in [0,10] which measures average sleep time
Eg. > 1s ➡10
< 100 ms ➡0

More sleeping ➡
More Bonus ➡Lower DP ➡Higher Priority

Dynamic Priority

- To distinguish between batch and interactive processes
- Based on average sleep time
  - An I/O bound process will sleep more therefore should get a higher priority
  - A CPU bound process will sleep less, therefore should get lower priority

dynamic priority = MAX(100, MIN(static priority – bonus + 5), 139))

heuristic

Average sleep time	Bonus
Greater than or equal to 0 but smaller than 100 ms	0
Greater than or equal to 100 ms but smaller than 200 ms	1
Greater than or equal to 200 ms but smaller than 300 ms	2
Greater than or equal to 300 ms but smaller than 400 ms	3
Greater than or equal to 400 ms but smaller than 500 ms	4
Greater than or equal to 500 ms but smaller than 600 ms	5
Greater than or equal to 600 ms but smaller than 700 ms	6
Greater than or equal to 700 ms but smaller than 800 ms	7
Greater than or equal to 800 ms but smaller than 900 ms	8
Greater than or equal to 900 ms but smaller than 1000 ms	9
1 second	10

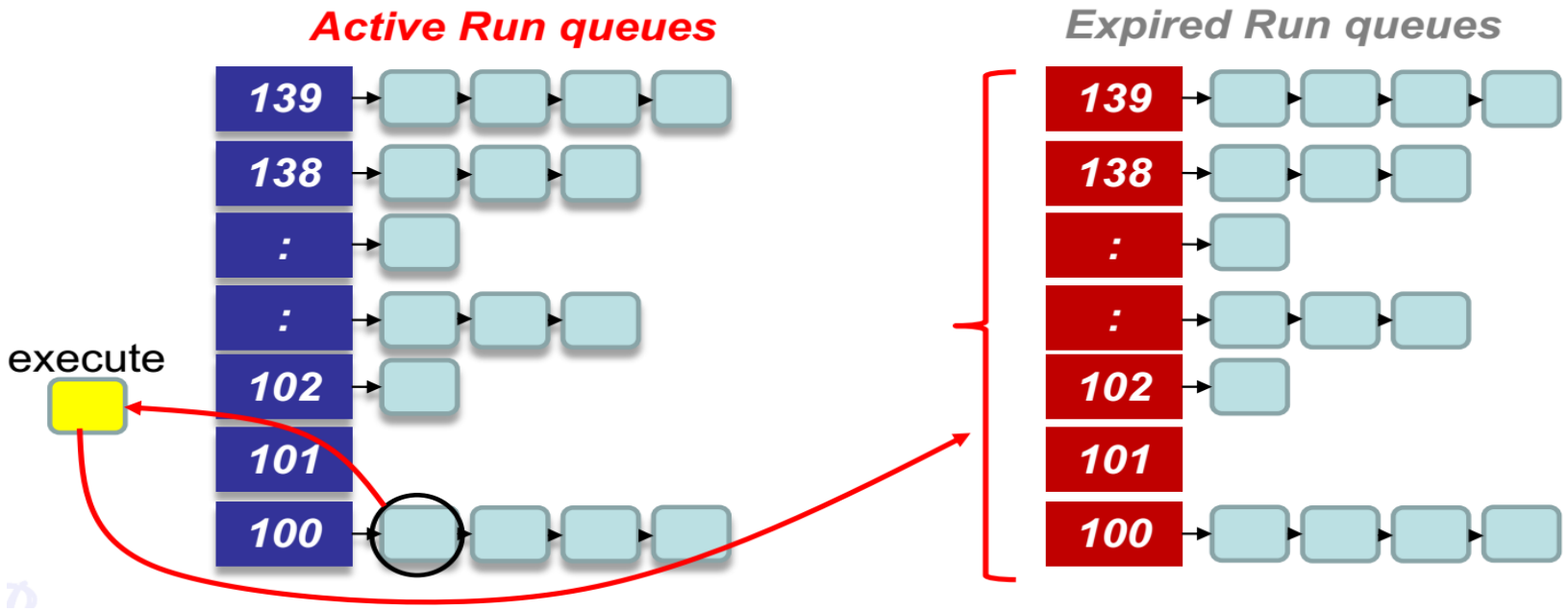
A process is **"interactive"** if **bonus –5 >= SP/4 -28**

So for a default process with SP= 120, if average wait time is ~ 700 ms then it qualifies as an interactive process

Implication: Upon completion of slot, an interactive process is not pushed to expired

Run Queues and Priority

- Dynamic priority used to determine which run queue to put the task
- No matter how ‘nice’ you are, you still need to wait on run queues --- prevents starvation



Setting the Timeslice

- Interactive processes have high priorities.
  - But likely to not complete their timeslice
  - Give it the largest timeslice to ensure that it completes its burst without being preempted. More heuristics

```
If priority < 120
        time slice = (140 – priority) * 20    milliseconds
else
        time slice = (140 – priority) * 5    milliseconds
```

Higher priority gets larger time slot
Large Variation:
SP = 100 => Time slice = 800ms
SP =139 => Time slice = 5 ms

Priority:	Static Pri	Niceness	Quantum
Highest	100	-20	800 ms
High	110	-10	600 ms
Normal	120	0	100 ms
Low	130	10	50 ms
Lowest	139	19	5 ms

Overall what happens

- We set the nice value and execute
- If it is I/O intensive, then average sleep time is large => Bonus is large => Priority is large => timeslice is large
- If large timeslot is fully utilized, then a process is likely to be severely demoted
- Again O(1), because we are touching a proc only during the entry and exit of a context switch

O(1) in a nutshell

- **Queues:** Multi level feed back queues with 40 priority classes
- **Base Priority:** Base priority set to 120 by default; modifiable by users using nice.
- **Dynamic Priority:** Dynamic priority set by heuristics based on process' sleep time
- **Dynamic timeslices:** Time slice interval for each process is set based on the dynamic priority
- **Starvation:** is dealt with by the two queues

- Too complex heuristics to distinguish between interactive and non-interactive processes
- Dependence between timeslice and priority
- Priority and timeslice values not uniform