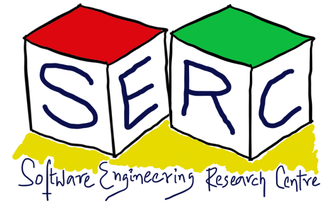
CS3.301 Operating Systems and Networks

Process Virtualisation - Policies (Scheduling) and Process Communication (Intro to networks)

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com



Acknowledgement

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- OSTEP Educator Materials, Remzi et al.
- OSTEP Book by Renzi et al.
- Modern Operating Systems, Tanenbaum et al.
- Networking Fundamentals by Practical Networking (Youtube Channel)
- Other online sources which are duly cited

Can we improve this a bit more?

- What about the user side?
  - What if this is an interactive process?
    - Think about going to Amazon or Working with some desktop application
    - Imagine a user sitting in front of the machine and executing the command
  - The machine identifies the nature of the job and schedules it
  - What about response time?

$$T_{response} = T_{firstrun} - T_{arrival}$$

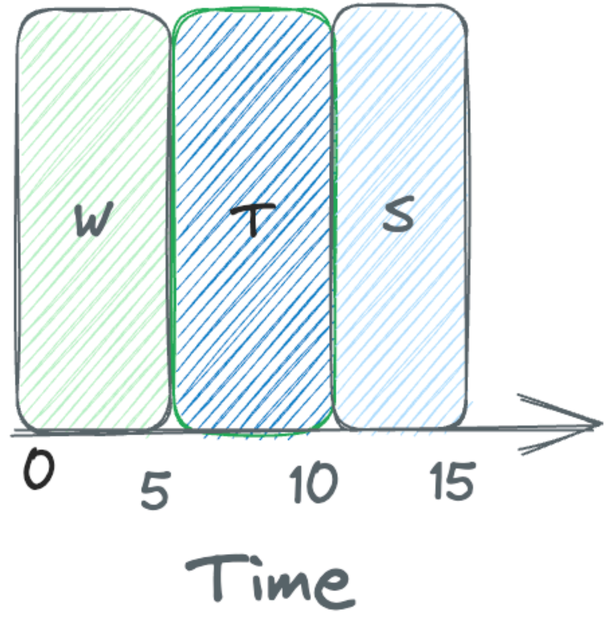
Round Robin Scheduling

- Instead of running jobs for completion
  - Can we run jobs for time intervals?
- **Policy:** Run jobs for a time slice -> switch to next job -> repeat till all are done!
- Key idea: Use the notion of time slice, considering timer interrupts
  - Take into consideration the overhead of Context Switch

Round Robin Scheduling

- What if we used SJF for the below scenario?

Process	Arrival	Time to Complete
W	0	5
S	0	5
T	0	5



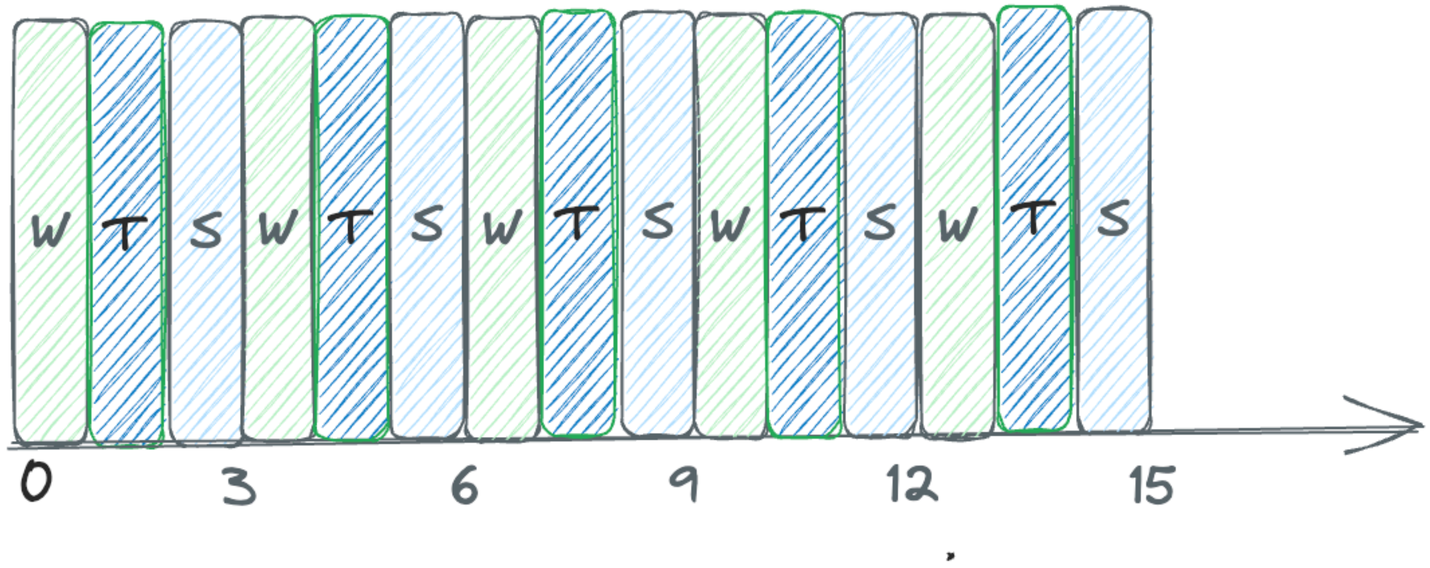
$$Avg(T_{turnaround}) = \frac{5 + 10 + 15}{3} = 10$$

$$Avg(T_{response}) = \frac{0 + 5 + 10}{3} = 5$$

Can we do better?

Round Robin Scheduling

- What if we do round robin with a time slice = 1 sec?



$$Avg(T_{response}) = \frac{0+1+2}{3}$$
$$= 1$$

W is added in the 0th Second
T in the 1st second
S in the second second

Do we foresee some issue?

Round Robin Scheduling

- Time slice plays a critical role in response time part
  - Too small time slice can result in an overhead - Too much Context Switch!
- RR is a good scheduling method
  - Key thing is to find an optimal time slice
  - Response time is the only metric
  - What about turnaround?

$$Avg(T_{turnaround}) = \frac{13 + 14 + 15}{3}$$

$$= 14!$$

Remember: Trade-off

- **Turnaround time only cares about completion**
  - Fairness of scheduling does not come into the picture - Given!
  - Fairness here comes at the cost of turnaround time
- The key aspect is to consider trade-off's
  - Very important in system design
  - Often among quality attributes
  - Eg: security vs performance

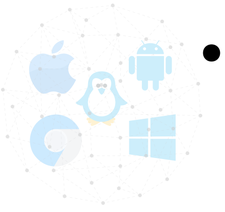
Continuing on the Assumptions

- Jobs don’t perform I/O
- Run-time of each job is known

What can be done to consider I/O? Can we do RR Scheduling by considering I/O?

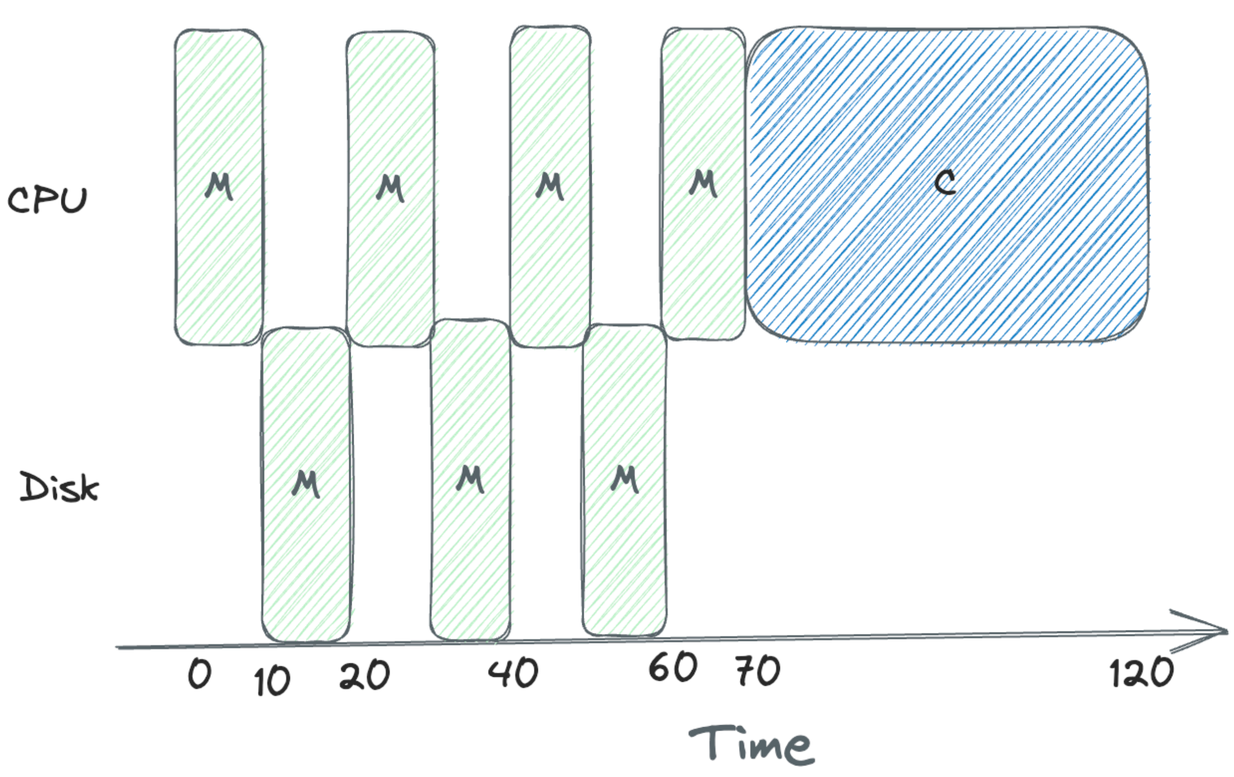
Incorporating I/O

- When there is a job doing I/O, Scheduler needs to be more decisive
  - During I/O what will usually happen?
    - The job will be **blocked** for I/O completion
    - If I/O is hard disk dependant then it may require more time
    - What can be an easy way out?
- When I/O is done - Interrupt is raised
  - OS moves the process (Job) from **blocked** to **ready** state



Lets consider a scenario

- Assume two process: Microsoft Word (autosave), your C program executing some numerical computation (No I/O access)
  - Microsoft Word (**M**): autosaves every 20 seconds => I/O access
  - C program (**C**): No I/O access



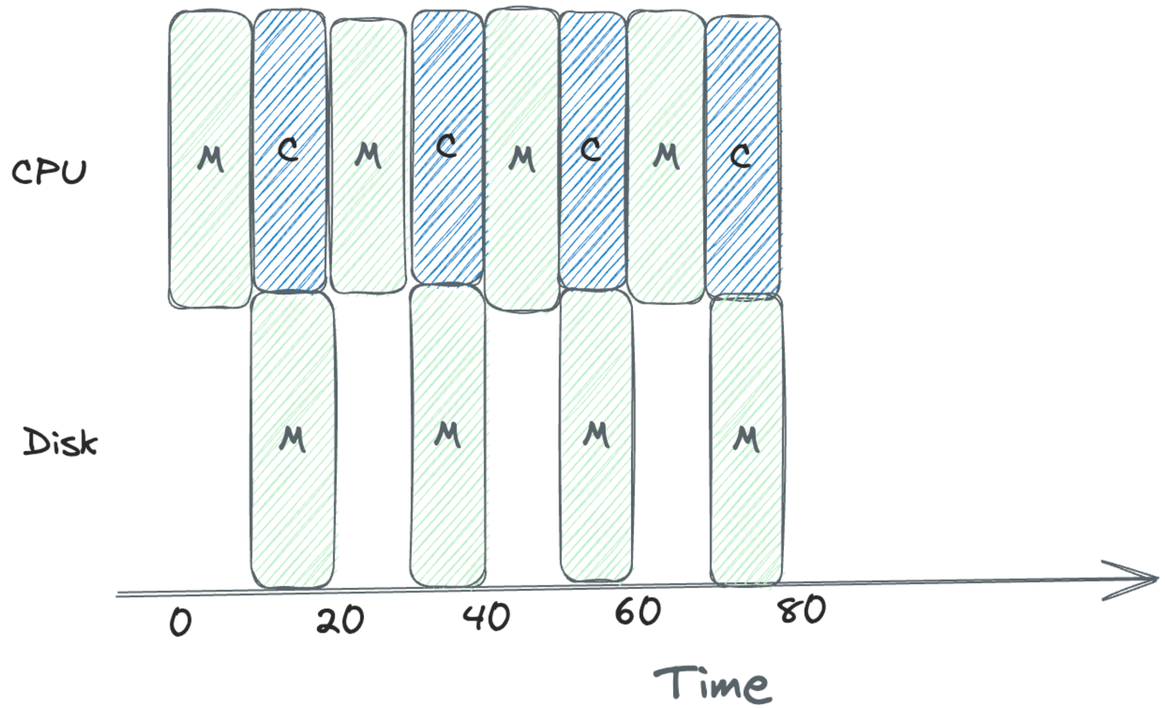
Can we do better?

Shortest Time to Completion First (STCF)

Can we leverage STCF?

- **Policy:** CPU is used by one process while the other one uses the disk
- Each CPU burst can be treated of as separate job
- Better utilization of the processor

Do we miss something?

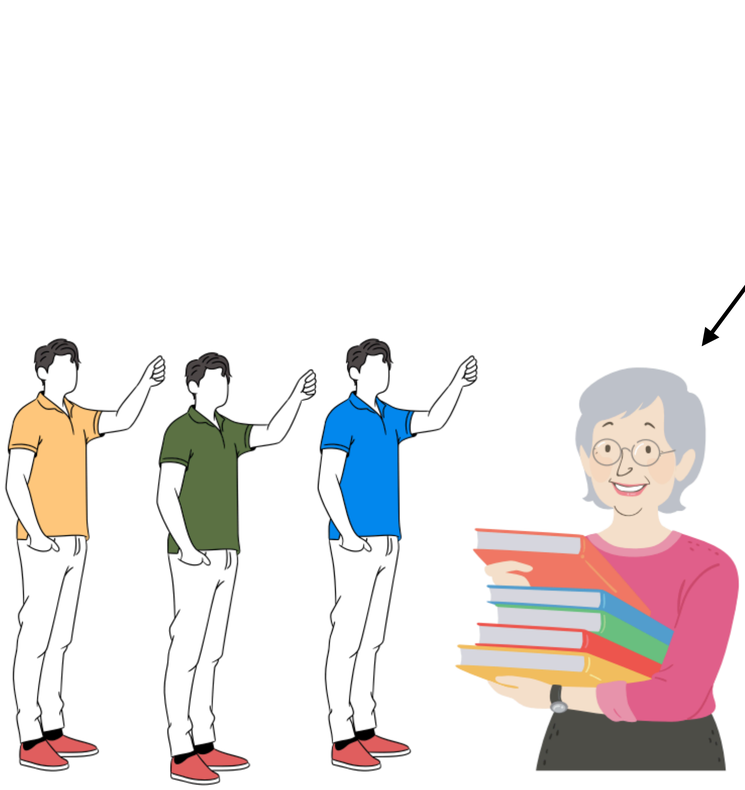


We may not know the length or expected time of completion of a job? — How to handle?

Lets go back to the example

How can Librarian take a guess?

The person is almost done with his reading and might come out soon (or time out!!)



More users/visitors have requested to access the reference section.How to decide whom to sent next?

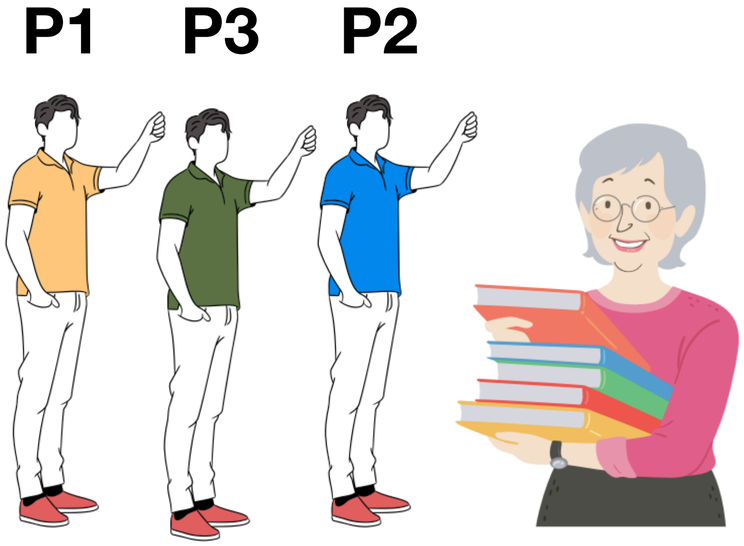
Why don’t we Prioritise?

Source: https://imgur.com

Lets go back to the example

Introduce Priority - Give priority, Observe and Improve

The person is almost done with his reading and might come out soon (or time out!!)



More users/visitors have requested to access the reference section.How to decide whom to sent next?

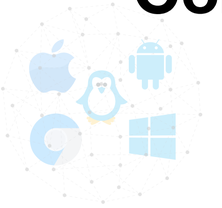
Multi Level Feedback Queues (MLFQ)

Two main features

- Reduce turn around times
  - Run shortest jobs first
- Reduce response time

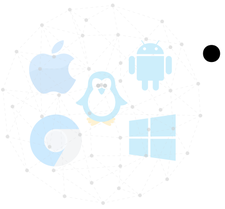
Can the policy learn continously to optimise response time and turnaround time?

Constraint: No apriori knowledge of the job length!

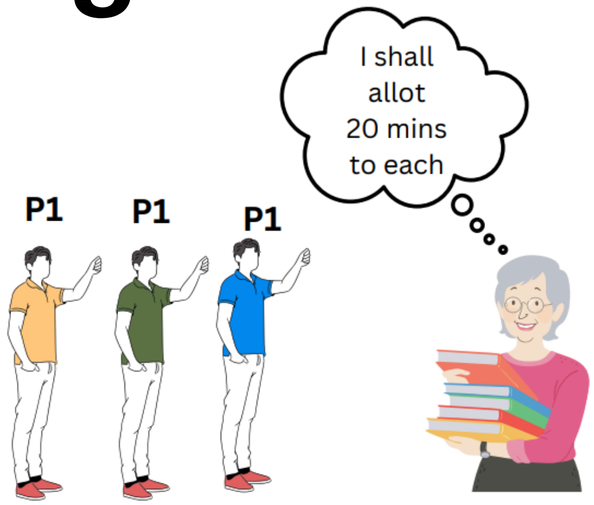


MLFQ: Basic Rules

- Use **n** number of distinct queues
  - Each queue has a different priority level
- Use priority to decide which job should be run at a given time
  - A job with a higher priority => job on a higher queue
- **Key idea:**
  - Scheduler sets priority to different jobs
  - Keep updating the priority based on observed behaviour



Going back to the example



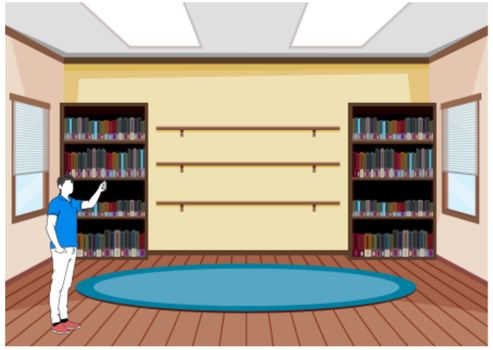
Put all the visitors initially on the same priority
Observe how they behave and then decide!

First person gets the chance!

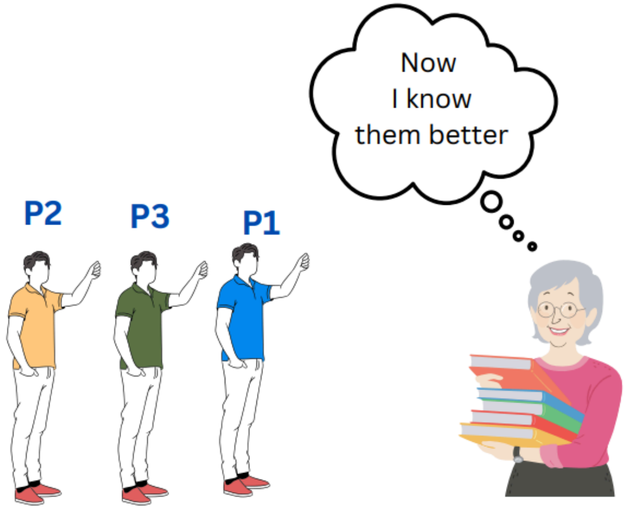
out in 18 mins

Next person goes in

20 mins!!



0ut in 15 mins!



For the next visit, priorities have been updated

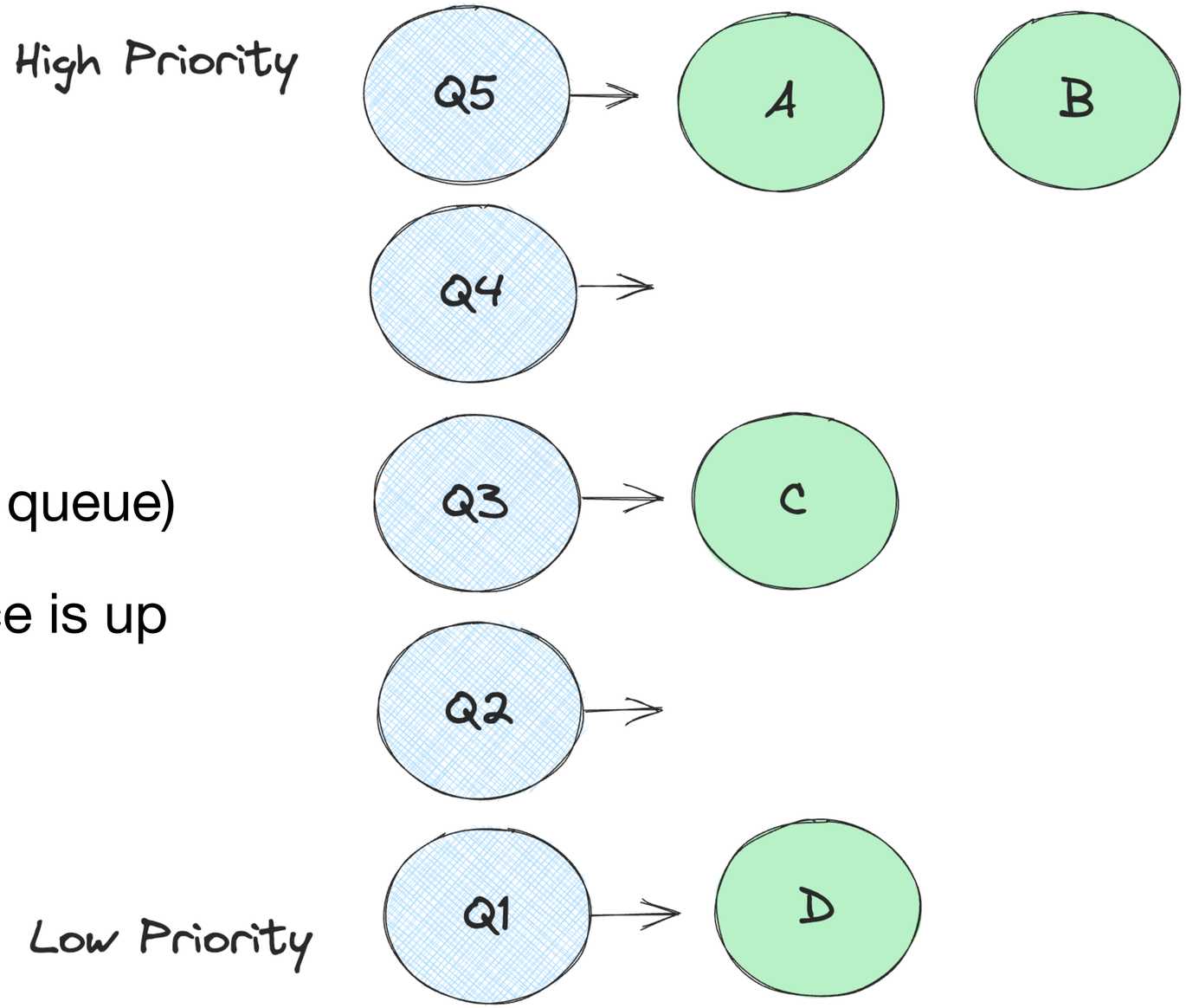
MLFQ

- Jobs that keep giving back the CPU - interactive jobs (higher priority)
- Jobs that uses CPU for more time - Reduce priority
- Learn about the processes as they run and **predict future** (Not using AI)
- **Two basic rules:**
    1. If priority (A) > Priority (B), A runs
    2. If priority (A) = Priority (B), A&B run in Round Robin

MLFQ: Visualisation

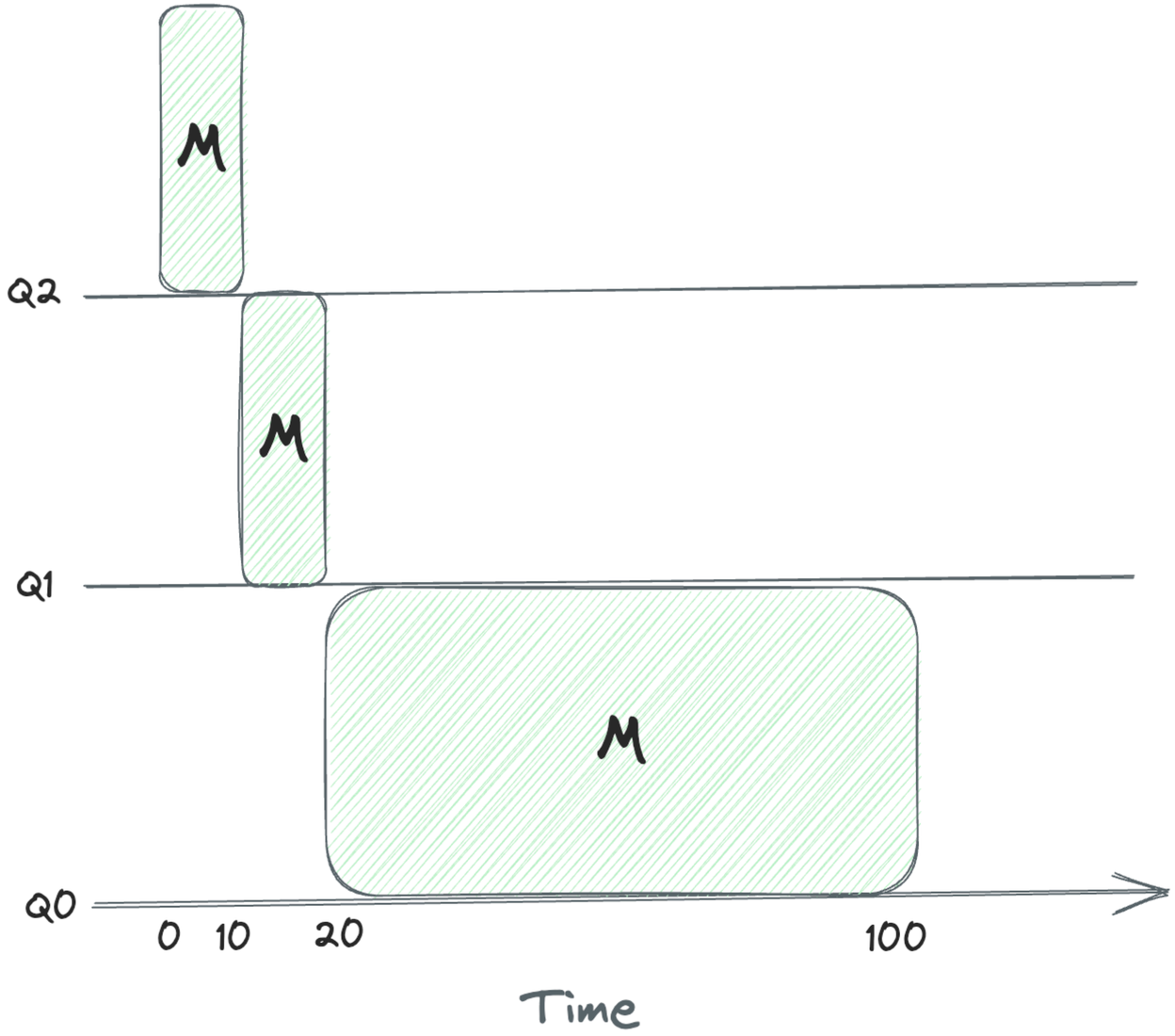
Five queues and 4 jobs

- Use notion of time slices
- When job enters, high priority
- If job uses up the entire time slice
  - Reduce priority (move down one queue)
- If job gives up CPU before time slice is up
  - Keep it at the same priority level



Example: Single Long Running Job

- Three queues
- Priority: Q2 > Q1 > Q0
- Single job M enters at t=0
- Time slice = 10 seconds
- After running for 10 seconds, priority is lowered

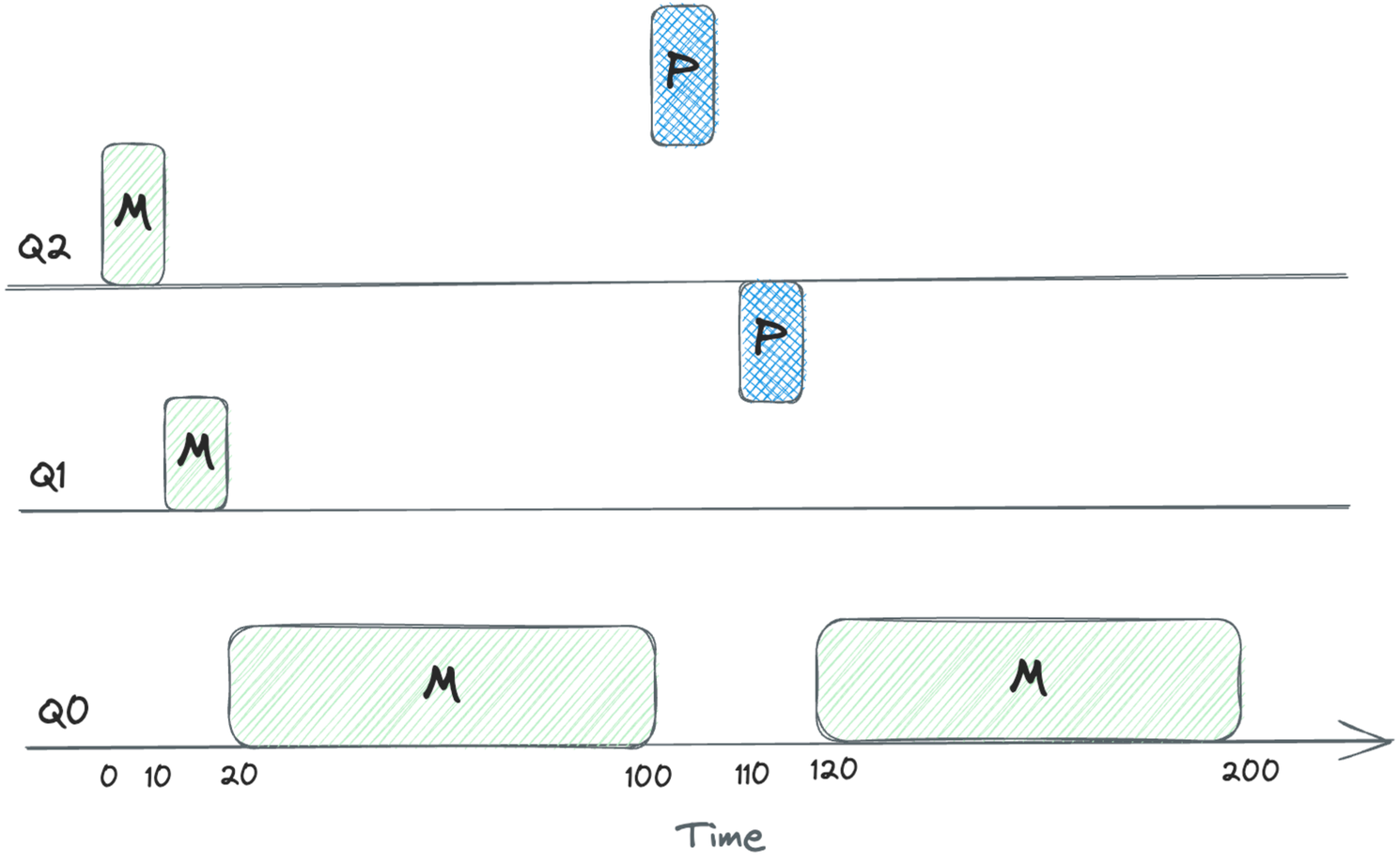


Example: Short job enters

- **P (interactive job)**
  - Enters at t=100
  - Runs for 10 second
  - Goes to the second queue
- Incoming job is considered

short => placed in higher queue

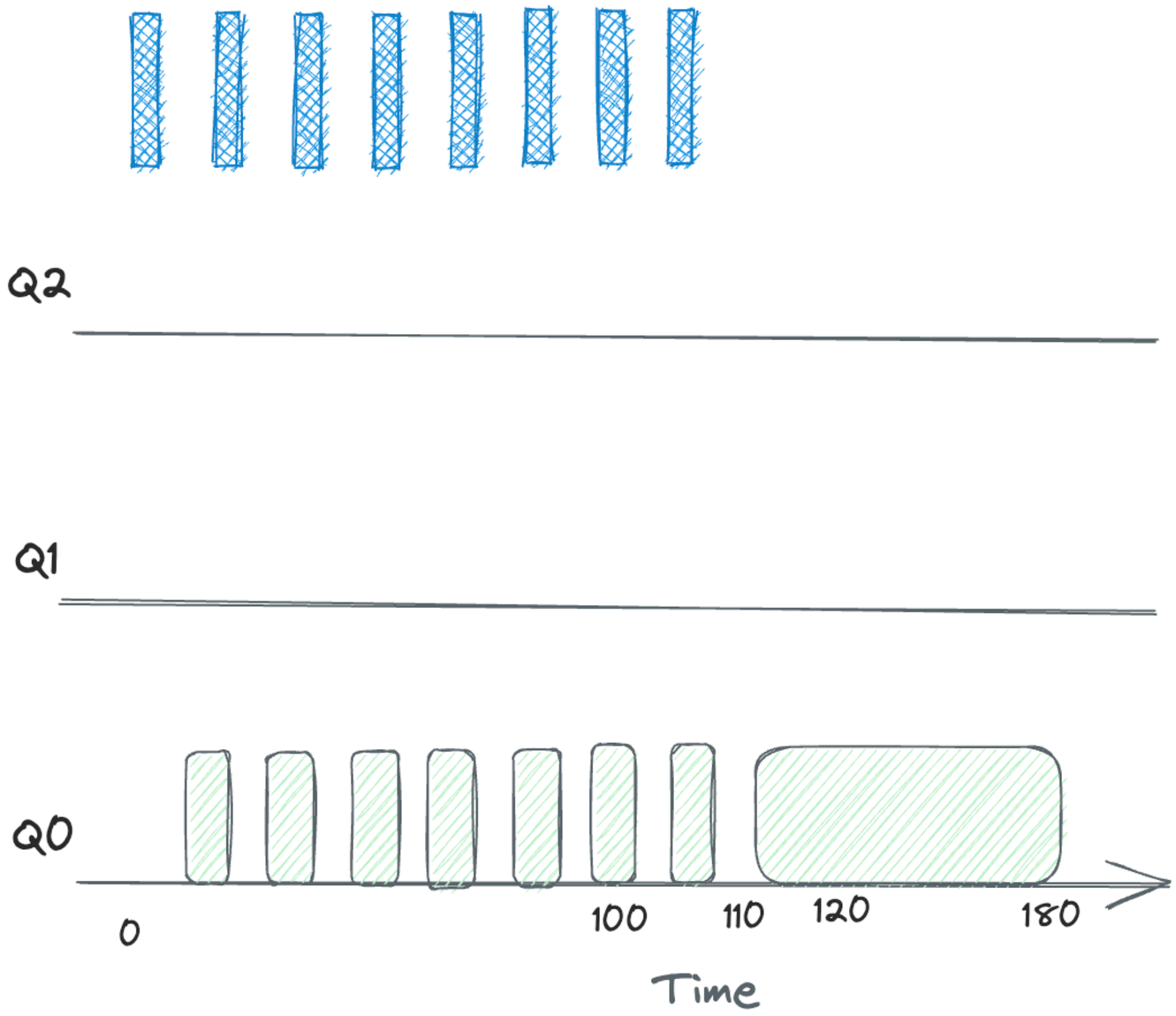
- If its short, it will keep running
  - Else moves down



Incorporating I/O

- Highly interactive job => relinquishes CPU faster
  - Priority is kept at same level
- Long term job can take more time and execute in lower queue

What could be the challenges?



What are the challenges with previous model?

It can lead to two major challenges

- **Starvation**
  - Too many interactive jobs will keep consuming CPU
  - Long running job will never get any CPU - **Starve!!**
- **Gaming of Scheduler**
  - The process can trick the scheduler into giving more than fair share - How?
  - **Idea:** Give an I/O request and relinquish the CPU before time slice is over
    - Priority does not change!! (Monopolise the CPU)

Another Challenge to Consider

- Program may also change behaviour over time
  - Suddenly there may be more CPU intensive phases
  - There may be also phases of interactiveness
  - Eg: Some long numerical computation followed by interactiveness

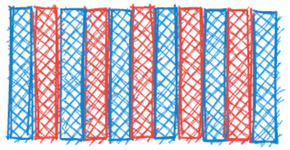
Can you think of some way to handle this?

Priority Boost

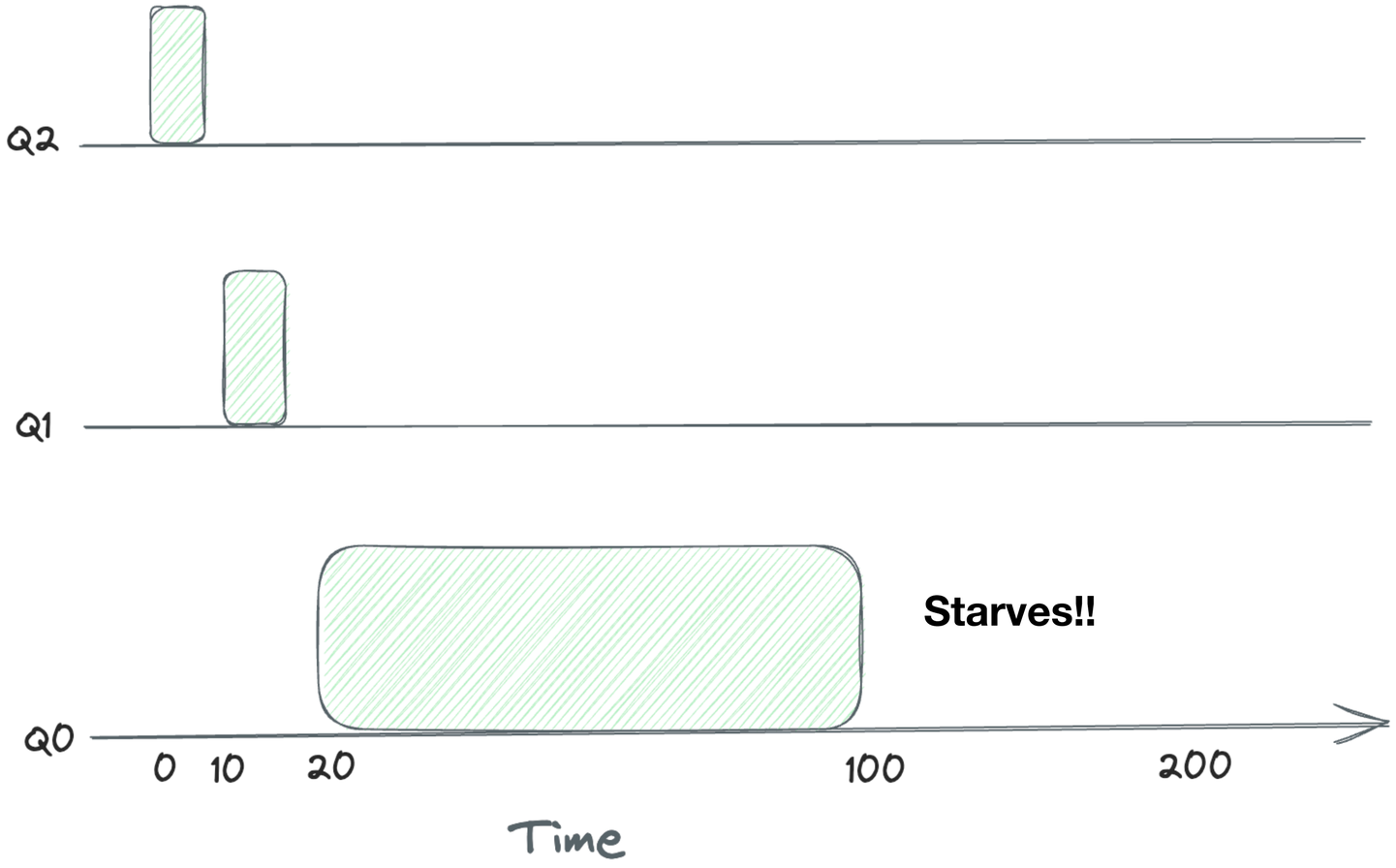
- Periodically boost the priority of all the jobs
  - This can **prevent starvation**
- **Rule:** After a time interval ***S***, move all jobs to top most queue
- **Provides two key guarantees**
  - Processes are guaranteed not to starve
  - If CPU bound has become interactive, scheduler will give it chances
    - Thanks to the periodic priority updates!

Case 1

Non priority Boost Scenario

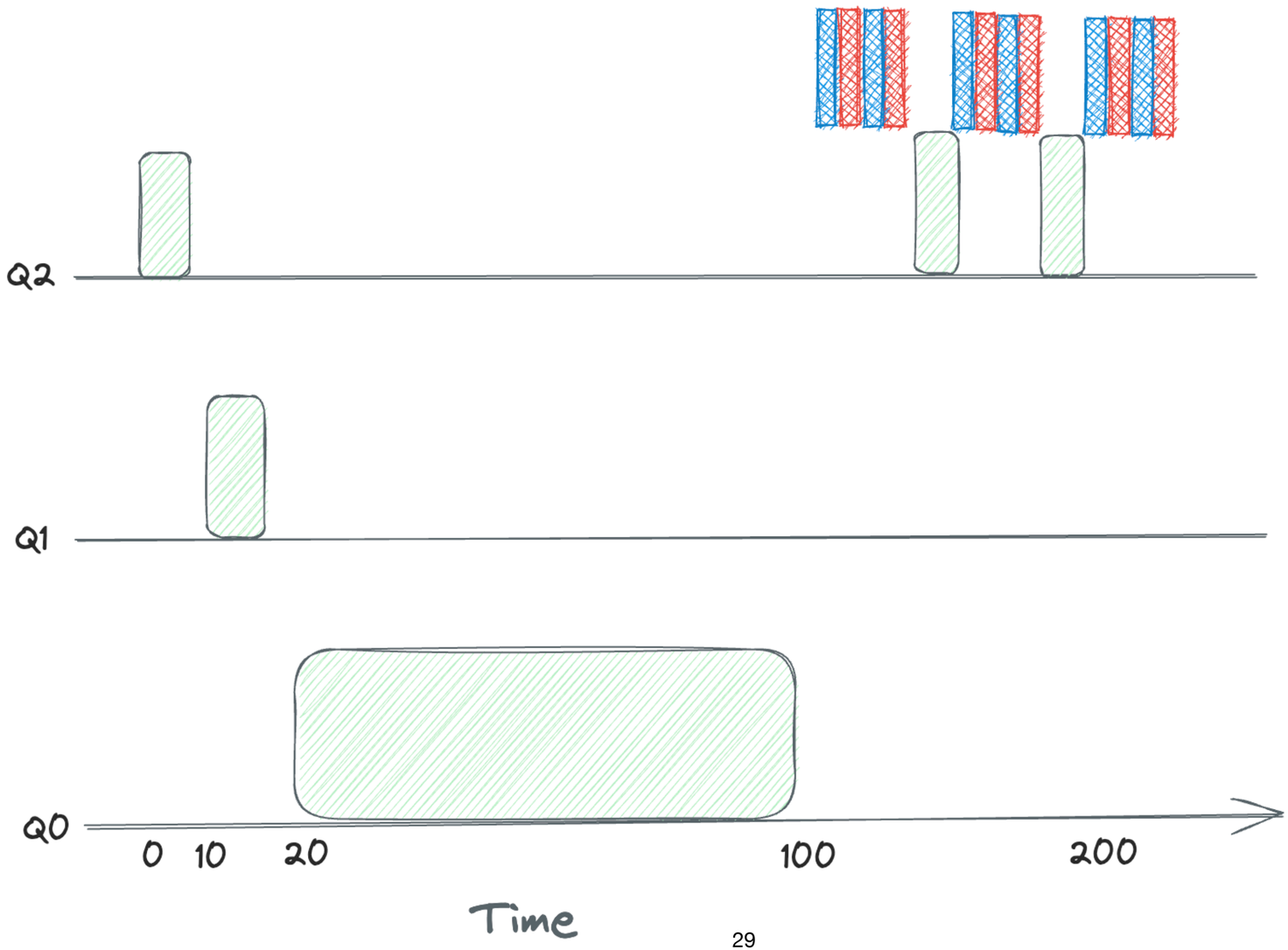


Interactive jobs keep getting the CPU



Case 2

With Priority Boost (S = 50)



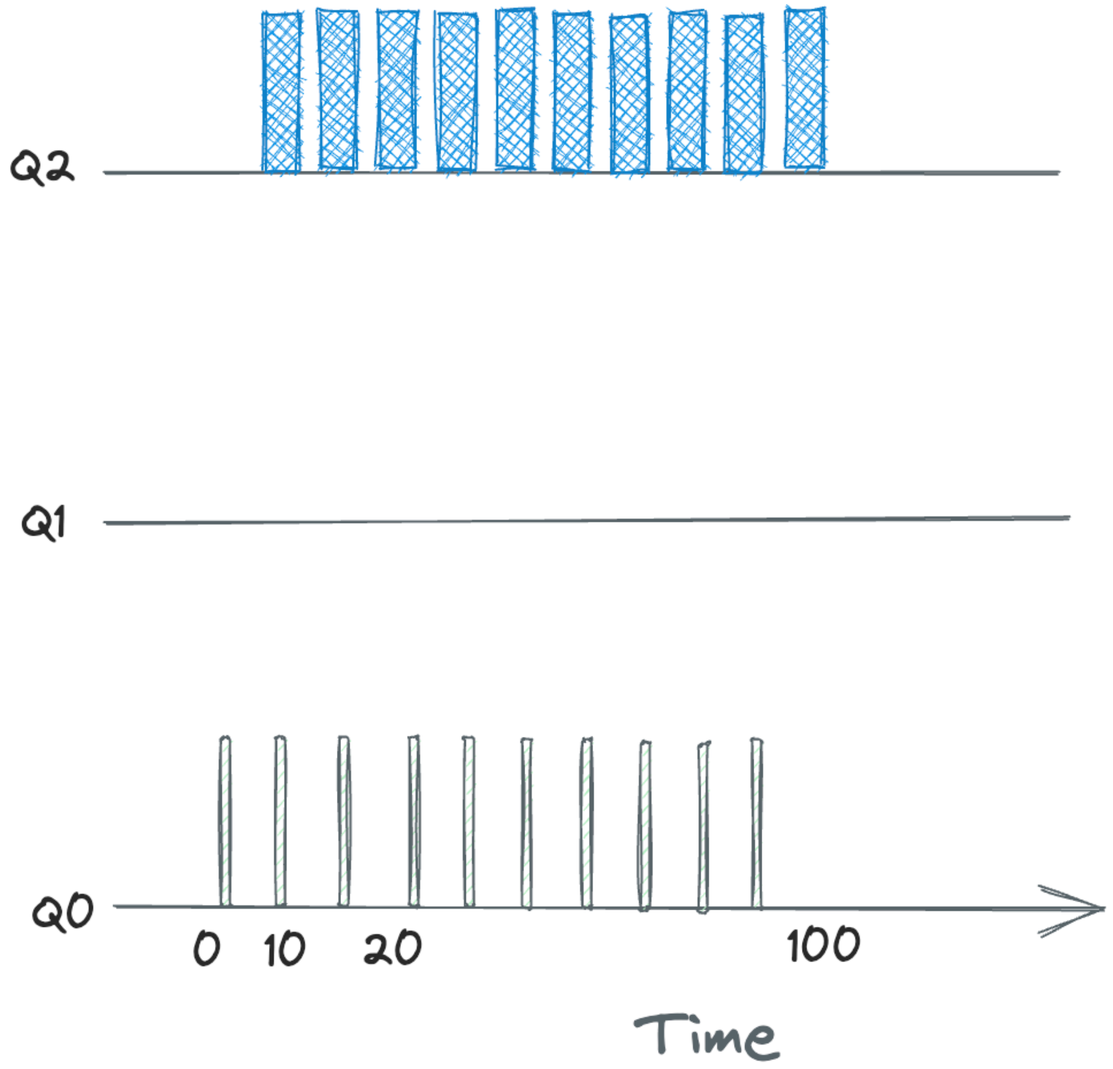
One additional Challenge

- How to determine the value of **S**?
  - If S is very small, interactive jobs may not get proper share of CPU
  - If S is too high, long running jobs could starve
- Tricky part is to come up with a value of S
  - Voo-doo constants - Named by John Ousterhout
- What about gaming the scheduler? - S by itself cannot solve it!

Better Accountability

How to prevent gaming of the scheduler?

- Introduce one more rule
  - Once a job uses its time allotment, it needs to be moved down
  - No consideration if the job has relinquished CPU ahead of time slice
  - Prevent the gaming since no matter what, priority reduces



Tuning MLFQ

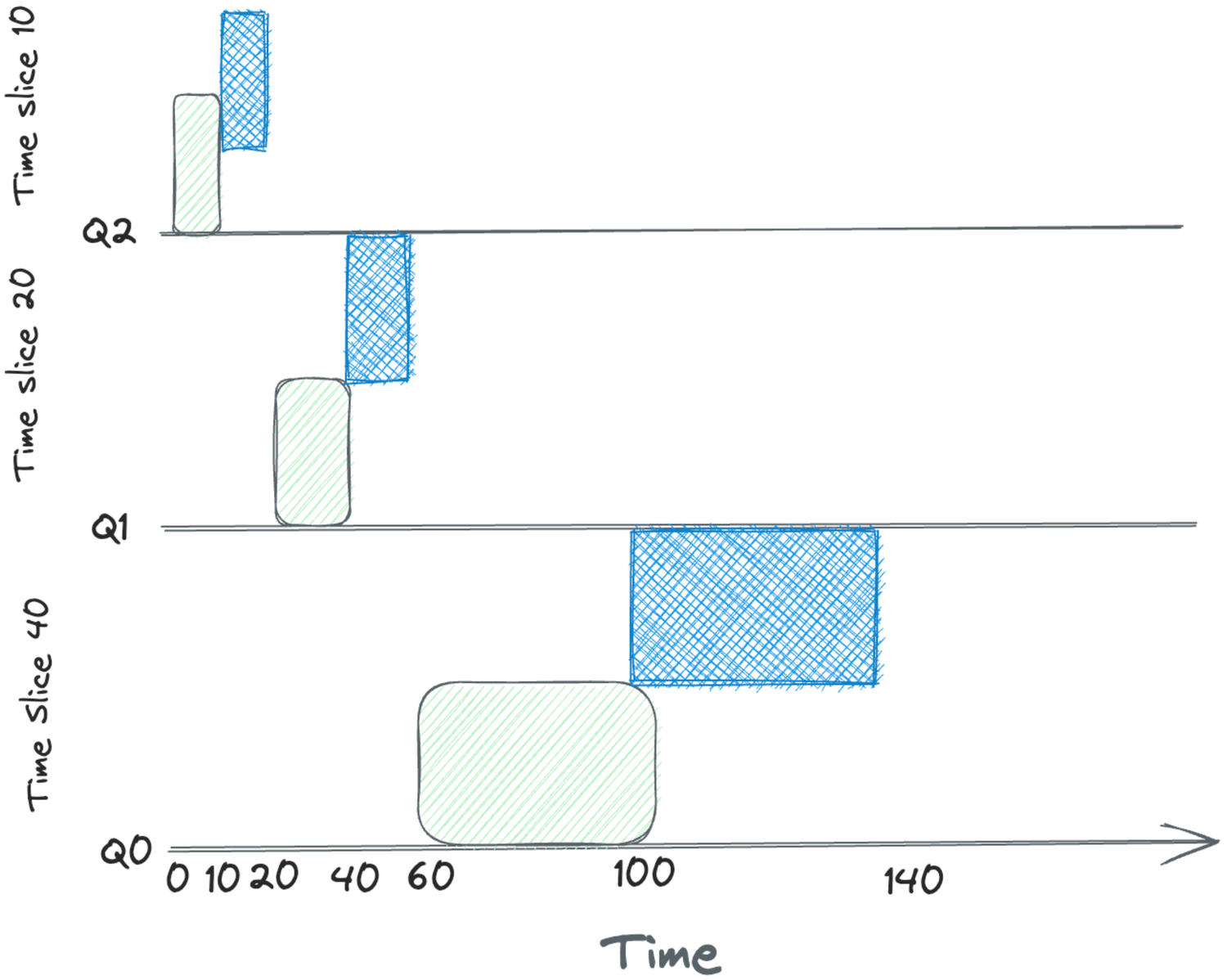
- How to parametrise the scheduler?
  - How many queues?
  - What should be the time slice?
  - How often the priority boost needs to be done?
- **High priority queues:** Interactive jobs with shorter time slices (10 ms)
- **Low priority queues:** CPU bound jobs with longer time slices (100 ms or less)

Example Scenario

- Different queues
- Low priority and high

Priority queues

- Each queue has a different slice interval
- Based on scenario, S changes (boost interval)



Solaris MLFQ

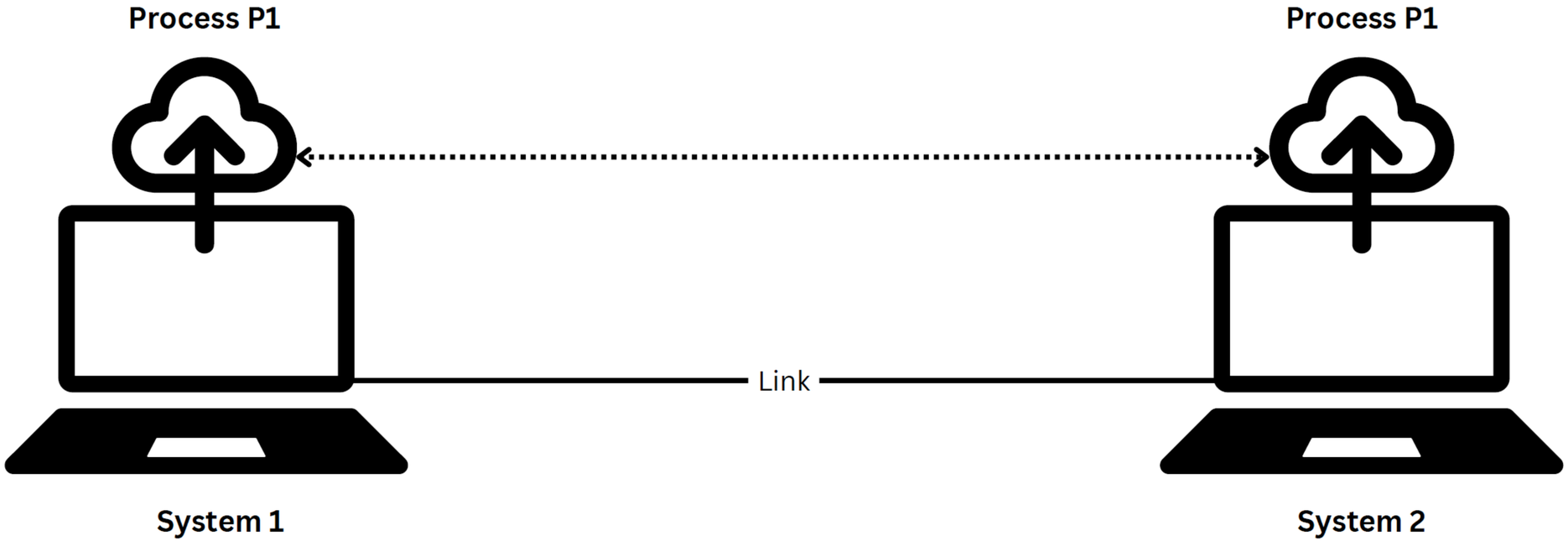
- Tables to configure:
  - How long should be the time slice?
  - How often to boost?
- 60 queues
  - Time slice length from 20ms to few 100 milliseconds
  - Priority boosted every 1 second
- Free BSD scheduler uses formula to calculate priority

MLFQ: Summing Up

- Five key rules are used by MLFQ:
  - If P(A) > P(B), A runs (B not)
  - If P(A) >= P(B), A & B runs in Round Robin using time slice of queue
  - When job enters, its placed in the highest priority
  - Once job uses its time allotment at a given level => priority is reduced
  - After a period, S move all jobs to top most priority queue
- BSD Unix derivatives, Windows NT and Solaris use a form of MLFQ in their basic scheduler

What about processes in different machines?

How can they communicate?



How does message/data from P1 in System 1 reach P1 in System 2?

What is the role of the OS in this and how does it contribute to the effectiveness?

Thank you

Course site: karthikv1392.github.io/cs3301_osn
Email: karthik.vaidhyanathan@iiit.ac.in
Twitter: @karthi_ishere