Systems and Networking I

Applied Computer Science and Artificial Intelligence

2025-2026

Gabriele Tolomei

Computer Science Department

Sapienza Università di Roma

tolomei@di.uniroma1.it

Sapienza Università di Roma

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling

LAST TIME

- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

TODAY

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- **Multilevel Queue (MLQ)**
- Multilevel Feedback-Queue (MLFQ)

MLQ: Idea

- Use multiple separate queues, one for each job **category**

MLQ: Idea

- Use multiple separate queues, one for each job **category**
- Each queue implements whatever scheduling algorithm is most appropriate for that type of jobs

MLQ: Idea

- Use multiple separate queues, one for each job **category**
- Each queue implements whatever scheduling algorithm is most appropriate for that type of jobs
- Scheduling must be done between queues!

MLQ: Idea

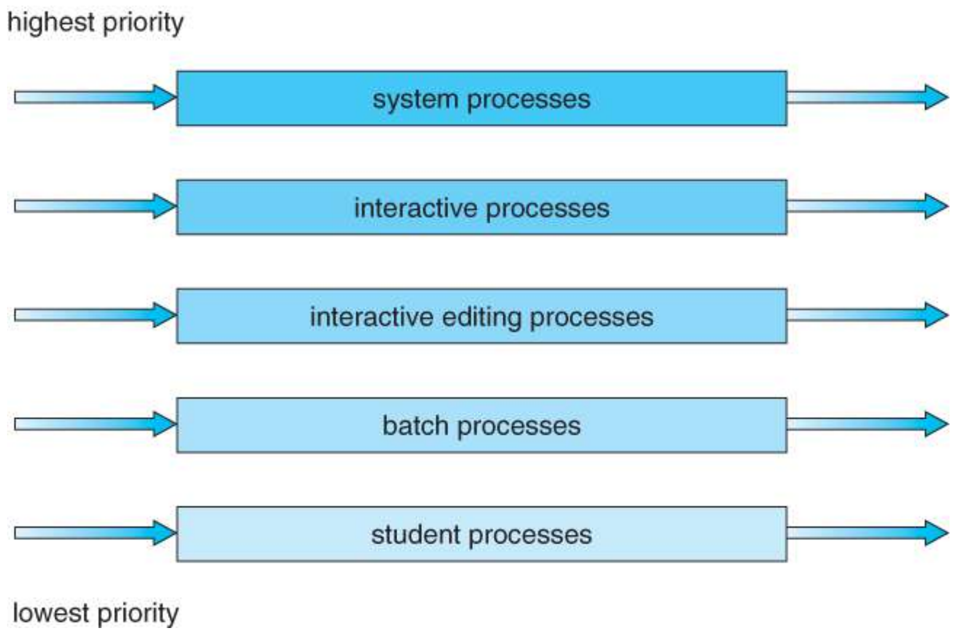
- Use multiple separate queues, one for each job **category**
- Each queue implements whatever scheduling algorithm is most appropriate for that type of jobs
- Scheduling must be done between queues!
- Two common options are:
  - **strict priority** → no job in a lower priority queue runs until all higher priority queues are empty
  - **round-robin** → each queue gets a time slice in turn, possibly of different sizes

MLQ: Idea

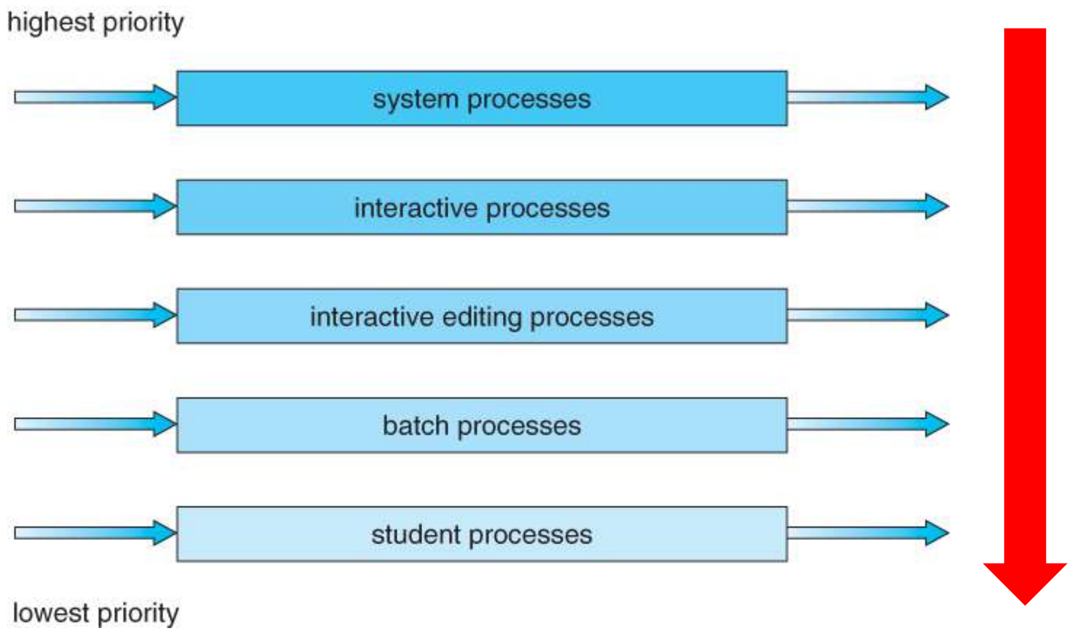
- Use multiple separate queues, one for each job **category**
- Each queue implements whatever scheduling algorithm is most appropriate for that type of jobs
- Scheduling must be done between queues!
- Two common options are:
  - **strict priority** → no job in a lower priority queue runs until all higher priority queues are empty
  - **round-robin** → each queue gets a time slice in turn, possibly of different sizes

Note: Jobs cannot switch from queue to queue

MLQ: Overview



MLQ: Overview



Time slice usually increases (exponentially) as priority gets lower

Scheduling Algorithms: An Overview

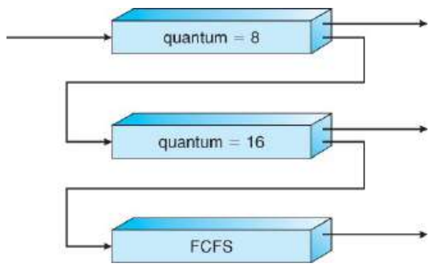
- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MLQ)
- **Multilevel Feedback-Queue (MLFQ)**

MLFQ: Idea

- Similar to the ordinary MLQ scheduling, except jobs may be moved from one queue to another

MLFQ: Idea

- Similar to the ordinary MLQ scheduling, except jobs may be moved from one queue to another
- Moving jobs may be required when:
  - The characteristics of a job change between CPU-intensive and I/O-intensive
  - A job that has waited for a long time can get bumped up into a higher priority queue for a while (to compensate the aging problem)



MLFQ: Adjusting Job Priority

- Job starts in the highest priority queue (by default)

MLFQ: Adjusting Job Priority

- Job starts in the highest priority queue (by default)
- If job's time slice expires → drop its priority level by one unit

MLFQ: Adjusting Job Priority

- Job starts in the highest priority queue (by default)
- If job's time slice expires → drop its priority level by one unit
- If job's time slice does not expire (i.e., the context switch occurs due to an I/O request, instead) → increase its priority level by one unit (up to the top) or stay the same

MLFQ: Adjusting Job Priority

- Job starts in the highest priority queue (by default)
- If job's time slice expires → drop its priority level by one unit
- If job's time slice does not expire (i.e., the context switch occurs due to an I/O request, instead) → increase its priority level by one unit (up to the top) or stay the same
- CPU-bound jobs will quickly drop their priority

MLFQ: Adjusting Job Priority

- Job starts in the highest priority queue (by default)
- If job's time slice expires → drop its priority level by one unit
- If job's time slice does not expire (i.e., the context switch occurs due to an I/O request, instead) → increase its priority level by one unit (up to the top) or stay the same
- CPU-bound jobs will quickly drop their priority
- I/O-bound jobs will stay at higher priority levels

MLFQ: Idea

- MLFQ is the most flexible but it is also the most complex to implement

MLFQ: Idea

- MLFQ is the most flexible but it is also the most complex to implement
- Some of the (many) parameters which define MLFQ systems include:
  - The number of queues
  - The scheduling algorithm for each queue
  - The methods used to upgrade or demote processes from one queue to another
  - The method used to determine which queue a process enters initially

Multilevel Feedback Queue (MLFQ): Example I

New A B C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Multilevel Feedback Queue (MLFQ): Example I

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

No I/O burst

Initial time quantum = 1

Context switch = 0

3 queues

Multilevel Feedback Queue (MLFQ): Example I

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

No I/O burst

Initial time quantum = 1

Context switch = 0

3 queues

strict priority between queues

Multilevel Feedback Queue (MLFQ): Example I

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

$$\text{JOB\_ID}_{total\_elapsed\_time}^{job\_exec\_time} = \text{The job JOB\_ID has executed } job\_exec\_time \text{ time units after } total\_elapsed\_time \text{ time units}$$

$$A_7^2 = \text{The job } A \text{ has executed } 2 \text{ time units after } 7 \text{ time units overall}$$

Multilevel Feedback Queue (MLFQ): Example I

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Queue	Time Slice (time units)	Jobs
1	1	
2	2	
3	4	

Multilevel Feedback Queue (MLFQ): Example I

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	
3	4	

Multilevel Feedback Queue (MLFQ): Example I

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	A^3_5, B^3_7, C^3_9
3	4	

Multilevel Feedback Queue (MLFQ): Example I

New | A | B | C |

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	A^3_5, B^3_7, C^3_9
3	4	A^7_{13}, B^7_{17}, C^7_{21}

Multilevel Feedback Queue (MLFQ): Example I

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	A^3_5, B^3_7, C^3_9
3	4	A^7_{13}, B^7_{17}, C^7_{21} A^{11}_{25}, B^{11}_{29}, C^{10}_{32}

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	
2	2	

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	

Multilevel Feedback Queue (MLFQ): Example II

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3
2	2	A^3_5

Multilevel Feedback Queue (MLFQ): Example II

New	A	B	C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3, C^2_6
2	2	A^3_5

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3, C^2_6
2	2	A^3_5, B^3_8

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	$A^1{}_1$, $B^1{}_2$, $C^1{}_3$, $C^2{}_6$, $C^3{}_9$
2	2	$A^3{}_5$, $B^3{}_8$

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^1_1, B^1_2, C^1_3, C^2_6, C^3_9
2	2	A^3_5, B^3_8, A^5_{11}

Multilevel Feedback Queue (MLFQ): Example II

New | A | B | C |

Order	Job	CPU burst (time units)
1	A	30
2	B	20
3	C	10

2 queues and C now alternates 1 time unit of CPU with 1 time unit of I/O

Queue	Time Slice (time units)	Jobs
1	1	A^{1}_{1}, B^{1}_{2}, C^{1}_{3}, C^{2}_{6}, C^{3}_{9}, C^{4}_{12}, …, C^{10}_{30}
2	2	A^{3}_{5}, B^{3}_{8}, A^{5}_{11}, B^{5}_{14}, …, B^{12}_{32}, A^{14}_{34}, …

MLFQ: Fairness Issue

- MLFQ tries to mimic the optimal behavior of SJF in terms of average waiting time

MLFQ: Fairness Issue

- MLFQ tries to mimic the optimal behavior of SJF in terms of average waiting time
- It explicitly promotes short jobs (i.e., I/O-bound ones) by design

MLFQ: Fairness Issue

- MLFQ tries to mimic the optimal behavior of SJF in terms of average waiting time
- It explicitly promotes short jobs (i.e., I/O-bound ones) by design
- **Problem:** SJF (and MLFQ) might be unfair (as opposed to RR)

Any increase in fairness by giving long jobs a fraction of the CPU when shorter jobs could be instead selected will increase waiting time

MLFQ: Improving Fairness

- Give each queue a fraction of the CPU time
  - This is fair only if jobs are evenly distributed (i.e., uniformly) across queues

MLFQ: Improving Fairness

- Give each queue a fraction of the CPU time
  - This is fair only if jobs are evenly distributed (i.e., uniformly) across queues
- Adjust dinamically the priority of jobs as they don't get scheduled
  - This avoids starvation but average waiting time might increase when the system is overloaded (all jobs get to the highest priority queue, eventually)

Advanced Topics: Lottery Scheduling

- Give every job a certain number of lottery tickets

Advanced Topics: Lottery Scheduling

- Give every job a certain number of lottery tickets
- On each time slice, pick a winning ticket **uniformly at random**

Advanced Topics: Lottery Scheduling

- Give every job a certain number of lottery tickets
- On each time slice, pick a winning ticket **uniformly at random**
- CPU time will be assigned to jobs according to the original distribution of tickets

Advanced Topics: Lottery Scheduling

- Give every job a certain number of lottery tickets
- On each time slice, pick a winning ticket **uniformly at random**
- CPU time will be assigned to jobs according to the original distribution of tickets

As the number of time slices
(i.e., the number of random picks) goes to infinity

Law of Large Numbers

Lottery Scheduling: Policy

- Assign tickets to jobs as follows:
  - Give more tickets to short running jobs
  - Give few tickets to long running jobs

Lottery Scheduling: Policy

- Assign tickets to jobs as follows:
  - Give more tickets to short running jobs
  - Give few tickets to long running jobs

simulating SJF

Lottery Scheduling: Policy

- Assign tickets to jobs as follows:
  - Give more tickets to short running jobs
  - Give few tickets to long running jobs

simulating SJF

- To avoid starvation, each job gets at least one ticket

Lottery Scheduling: Policy

- Assign tickets to jobs as follows:
  - Give more tickets to short running jobs
  - Give few tickets to long running jobs

simulating SJF

- To avoid starvation, each job gets at least one ticket
- Degrades gracefully as system load changes
  - Adding/deleting a job affects all the other jobs proportionally

Lottery Scheduling vs. All

Question:

What is the main difference between lottery scheduling and any other algorithgm we have seen so far?

Lottery Scheduling vs. All

Question:

What is the main difference between lottery scheduling and any other algorithgm we have seen so far?

Answer:

This is the only example of randomized scheduler

(rather than deterministic one)

Lottery Scheduling: Example

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1		

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2		

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2	-	

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2	-	50% (1/2)

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2	-	50% (1/2)
2/0	50% (10/20)	-

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2	-	50% (1/2)
2/0	50% (10/20)	-
10/1	~9.9% (10/101)	~0.99% (1/101)

Lottery Scheduling: Example

short jobs get 10 tickets each | long jobs get 1 ticket each

#short jobs / #long jobs	% of CPU for each short job	% of CPU for each long job
1/1	~91% (10/11)	~9% (1/11)
0/2	-	50% (1/2)
2/0	50% (10/20)	-
10/1	~9.9% (10/101)	~0.99% (1/101)
1/10	50% (10/20)	5% (1/20)

Lottery Scheduling: CPU Assignment

$$n_{short} = \text{total number of } \textit{short} \text{ jobs}$$
$$n_{long} = \text{total number of } \textit{long} \text{ jobs}$$
$$N = n_{short} + n_{long} = \text{total number of jobs}$$

$$m_{short} = \text{number of tickets assigned to each } \textit{short} \text{ job}$$
$$m_{long} = \text{number of tickets assigned to each } \textit{long} \text{ job}$$
$$M = m_{short} * n_{short} + m_{long} * n_{long} = \text{total number of tickets}$$

Lottery Scheduling: CPU Assignment

$$n_{short} = \text{total number of } \textit{short} \text{ jobs}$$
$$n_{long} = \text{total number of } \textit{long} \text{ jobs}$$
$$N = n_{short} + n_{long} = \text{total number of jobs}$$

$$m_{short} = \text{number of tickets assigned to each } \textit{short} \text{ job}$$
$$m_{long} = \text{number of tickets assigned to each } \textit{long} \text{ job}$$
$$M = m_{short} * n_{short} + m_{long} * n_{long} = \text{total number of tickets}$$

$$\boxed{\begin{aligned} \text{CPU}_{short} &= \frac{m_{short}}{M} \\ \text{CPU}_{long} &= \frac{m_{long}}{M} \end{aligned}}$$

Lottery Scheduling: CPU Assignment Probability

$$m_i = \text{number of tickets assigned to job } i$$

$$N = \text{total number of jobs}$$

$$M = \sum_{i=1}^{N} m_i = \text{total number of tickets}$$

$$P(i) = \frac{m_i}{M} = \text{probability of job } i \text{ being scheduled}$$

Summary of CPU Scheduling Algorithms

- First-Come-First-Serve (FCFS): Very simple, non-preemptive, generally unfair, and highly variant waiting time

Summary of CPU Scheduling Algorithms

- **First-Come-First-Serve (FCFS)**: Very simple, non-preemptive, generally unfair, and highly variant waiting time
- **Round Robin (RR)**: Fair but average waiting time may be long

Summary of CPU Scheduling Algorithms

- First-Come-First-Serve (FCFS): Very simple, non-preemptive, generally unfair, and highly variant waiting time
- Round Robin (RR): Fair but average waiting time may be long
- Shortest Job First (SJF): Generally unfair and possible starvation, but provably optimal in terms of average waiting time (assuming we are able to correctly predict the length of the next CPU burst)

Summary of CPU Scheduling Algorithms

- **First-Come-First-Serve** (FCFS): Very simple, non-preemptive, generally unfair, and highly variant waiting time
- **Round Robin** (RR): Fair but average waiting time may be long
- **Shortest Job First** (SJF): Generally unfair and possible starvation, but provably optimal in terms of average waiting time (assuming we are able to correctly predict the length of the next CPU burst)
- **Multilevel Queuing** (MLQ/MLFQ): An approximation of SJF

Summary of CPU Scheduling Algorithms

- **First-Come-First-Serve** (FCFS): Very simple, non-preemptive, generally unfair, and highly variant waiting time
- **Round Robin** (RR): Fair but average waiting time may be long
- **Shortest Job First** (SJF): Generally unfair and possible starvation, but provably optimal in terms of average waiting time (assuming we are able to correctly predict the length of the next CPU burst)
- **Multilevel Queuing** (MLQ/MLFQ): An approximation of SJF
- **Lottery**: Fairer with a low average waiting time yet less predictable due to randomization

Summary of CPU Scheduling Algorithms

- **First-Come-First-Serve (FCFS)**: Very simple, non-preemptive, generally unfair, and highly variant waiting time
- **Round Robin (RR)**: Fair but average waiting time may be long
- **Shortest Job First (SJF)**: Generally unfair and possible starvation, but provably optimal in terms of average waiting time (assuming we are able to correctly predict the length of the next CPU burst)
- **Multilevel Queuing (MLQ/MLFQ)**: An approximation of SJF
- **Lottery**: Fairer with a low average waiting time yet less predictable due to randomization