Sistemi Operativi

Corso di Laurea in Informatica

a.a. 2019-2020

Gabriele Tolomei

Dipartimento di Informatica

Sapienza Università di Roma

tolomei@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Recap from Last Lecture

- Scheduling allows one process to use the CPU while another is waiting for I/O, thereby maximizing system utilization
- **non-preemptive** vs. **preemptive** scheduler
- Different scheduling policies optimize different metrics
- 2 out of 6 scheduling algorithms:
  - **First-Come-First-Serve** (**FCFS**)
  - **Round Robin** (**RR**)

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)

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

Assuming all jobs arrive at the same time (arrival time = 0)

SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

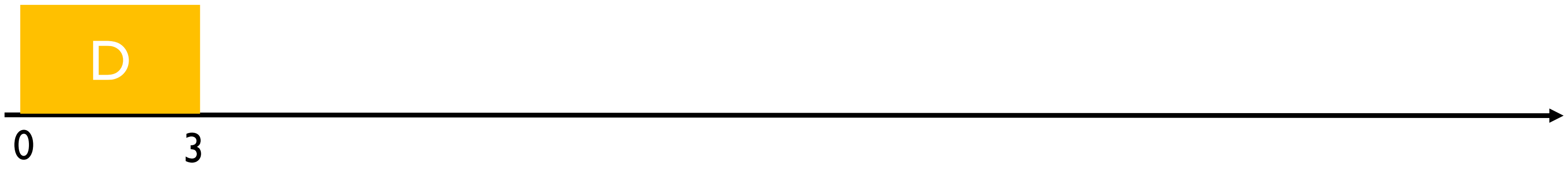
Job	CPU burst (time units)
A	6
B	8
C	7
D	3

0

SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

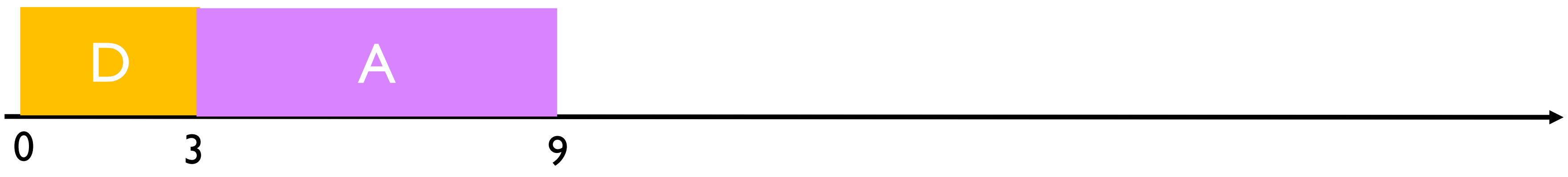
Job	CPU burst (time units)
A	6
B	8
C	7
D	3



SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3



SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

avg. waiting time =

SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

avg. waiting time = (3 + 16 + 9 + 0)/4 = **7**

SJF: PROs and CONs

- **PROs:**
  - Provably optimal when the goal is to minimize the avg. waiting time

SJF: PROs and CONs

- **PROs:**
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers

    (preemptive SJF is called **SRTF** or **S**hortest **R**emaining **T**ime **F**irst)

SJF: PROs and CONs

- **PROs:**
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers

    (preemptive SJF is called **SRTF** or **S**hortest **R**emaining **T**ime **F**irst)
- **CONs:**
  - Almost impossible to predict the amount of CPU time of a job

SJF: PROs and CONs

- **PROs:**
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers

    (preemptive SJF is called **SRTF** or **S**hortest **R**emaining **T**ime **F**irst)
- **CONs:**
  - Almost impossible to predict the amount of CPU time of a job
  - Long running CPU-bound jobs can *starve* (as I/O-bound ones have implicitly higher priority over them)

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

$$x_t = \textit{actual} \text{ length of the } t\text{-th CPU burst}$$

$$s_{t+1} = \textit{predicted} \text{ length of the } (t{+}1)\text{-th CPU burst}$$

$$\alpha \in \mathbb{R},\ 0 \leq \alpha \leq 1$$

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1 - \alpha) s_t$$

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

$$x_t = \textit{actual} \text{ length of the } t\text{-th CPU burst}$$

$$s_{t+1} = \textit{predicted} \text{ length of the } (t{+}1)\text{-th CPU burst}$$

$$\alpha \in \mathbb{R},\ 0 \leq \alpha \leq 1$$

$$\boxed{\begin{aligned} s_1 &= x_0 \\ s_{t+1} &= \alpha \boxed{x_t} + (1 - \alpha) \boxed{s_t} \end{aligned}}$$

weighted average between previous **observation** and previous **prediction**

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \cancel{\alpha x_t} + \boxed{(1-\alpha)s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1 - \alpha)s_t$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \boxed{\alpha x_t} \cancel{+ (1-\alpha)s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

The next burst is assumed to be the same as the last actual CPU burst observed

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \boxed{\alpha x_t} + \cancel{(1-\alpha)s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

The next burst is assumed to be the same as the last actual CPU burst observed

Recent history does not count

Exponential Smoothing

t	0	1	2	3	4	5	6	7	...

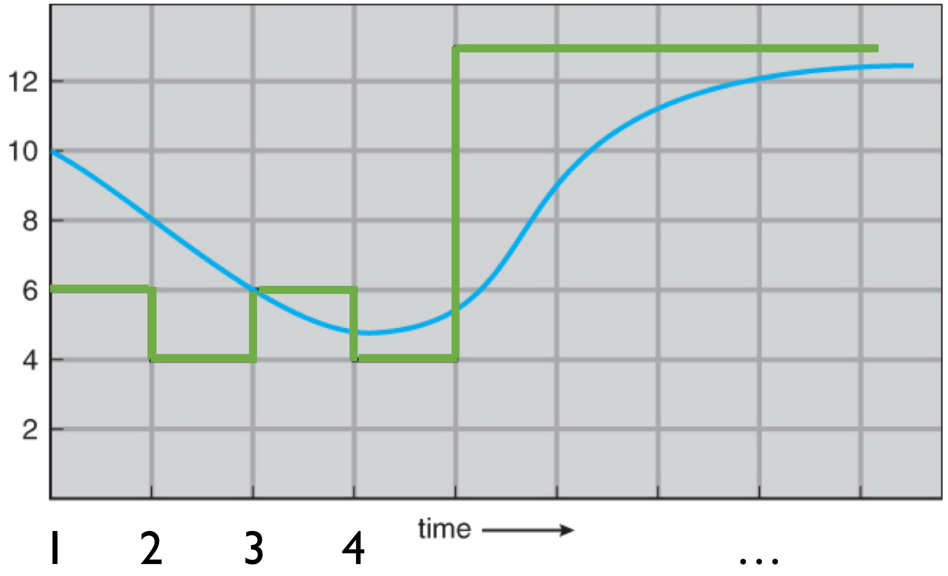
Exponential Smoothing

	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…

Exponential Smoothing

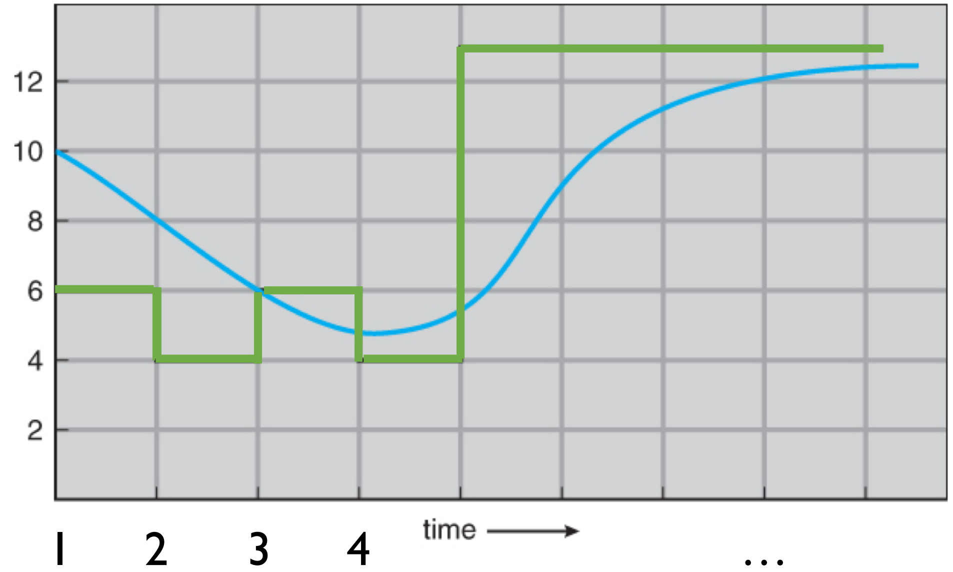
	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

Exponential Smoothing



	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

Exponential Smoothing

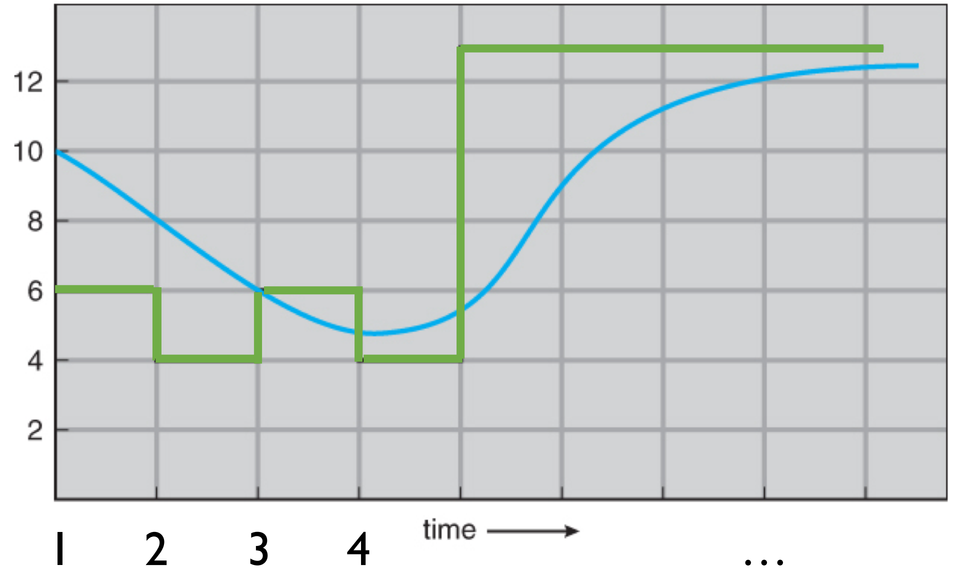


	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

$s_1 = x_0$

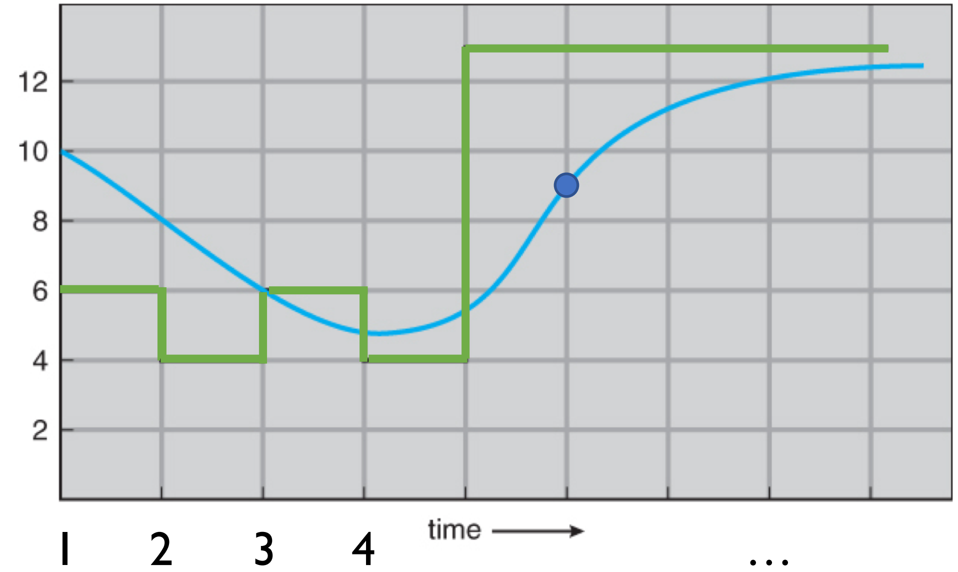
bootstrap

Exponential Smoothing



	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

Exponential Smoothing

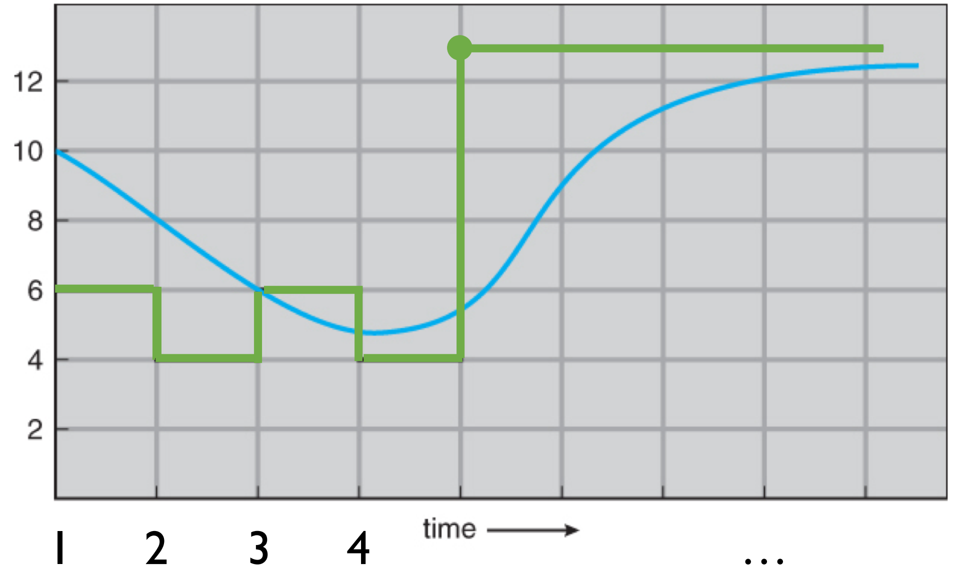


t	0	1	2	3	4	5	6	7	…
observations x_t	10	6	4	6	4	13	13	13	…
predictions s_{t+1}	10	8	6	6	5	9	11	12	…

9 =

$s_6 =$

Exponential Smoothing



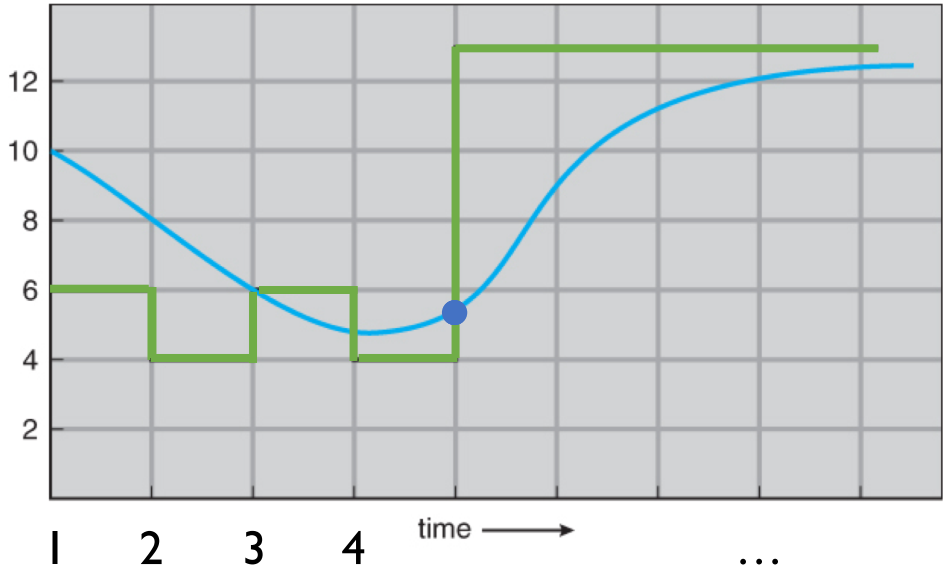
Usually, alpha is set to 0.5

	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

9 = 0.5 * 13

$$s_6 = \quad \alpha x_5$$

Exponential Smoothing



Usually, alpha is set to 0.5

	t	0	1	2	3	4	5	6	7	…
observations	x_t	10	6	4	6	4	13	13	13	…
predictions	s_{t+1}	10	8	6	6	5	9	11	12	…

$$9 = 0.5 * 13 + 0.5 * 5$$

$$s_6 = \alpha x_5 + (1-\alpha) s_5$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$
$$s_t = \alpha x_{t-1} + (1-\alpha)s_{t-1}$$
$$\ldots$$
$$s_2 = \alpha x_1 + (1-\alpha)s_1$$
$$s_1 = x_0$$

Exponential Smoothing

$$\boxed{s_{t+1}} = \alpha x_t + (1-\alpha)s_t$$
$$\boxed{s_t} = \alpha x_{t-1} + (1-\alpha)s_{t-1}$$
$$\cdots$$
$$\boxed{s_2} = \alpha x_1 + (1-\alpha)s_1$$
$$\boxed{s_1} = x_0$$

predictions/forecasts

Exponential Smoothing

$$
\begin{aligned}
s_{t+1} &= \alpha \boxed{x_t} + (1-\alpha)s_t \\
s_t &= \alpha \boxed{x_{t-1}} + (1-\alpha)s_{t-1} \\
&\dots \\
s_2 &= \alpha \boxed{x_1} + (1-\alpha)s_1 \\
s_1 &= \boxed{x_0}
\end{aligned}
$$

actual observations

Exponential Smoothing

$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

$$= \alpha x_t + (1-\alpha)[\underbrace{\alpha x_{t-1} + (1-\alpha)s_{t-1}}_{s_t}]$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

$$= \alpha x_t + (1-\alpha)[\underbrace{\alpha x_{t-1} + (1-\alpha)s_{t-1}}_{s_t}]$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2 s_{t-1}$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

$$= \alpha x_t + (1-\alpha)[\underbrace{\alpha x_{t-1} + (1-\alpha)s_{t-1}}_{s_t}]$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2 s_{t-1}$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2[\underbrace{\alpha x_{t-2} + (1-\alpha)s_{t-2}}_{s_{t-1}}]$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + (1-\alpha)^3 s_{t-2}$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

$$= \alpha x_t + (1-\alpha)[\underbrace{\alpha x_{t-1} + (1-\alpha)s_{t-1}}_{s_t}]$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2 s_{t-1}$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + (1-\alpha)^2[\underbrace{\alpha x_{t-2} + (1-\alpha)s_{t-2}}_{s_{t-1}}]$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + (1-\alpha)^3 s_{t-2}$$

$$\cdots$$

$$= \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \alpha(1-\alpha)^3 x_{t-3} + \ldots + (1-\alpha)^{t-1} s_2$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + \alpha(1-\alpha)^{t-1} x_1 + (1-\alpha)^t x_0$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + \alpha(1-\alpha)^{t-1} x_1 + (1-\alpha)^t x_0$$

$$= \alpha[x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} x_1] + (1-\alpha)^t x_0$$

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + \alpha(1-\alpha)^{t-1} x_1 + (1-\alpha)^t x_0$$

$$= \alpha \boxed{[x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} x_1]} + (1-\alpha)^t x_0$$

Past *t* **observations**

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + \alpha(1-\alpha)^{t-1} x_1 + (1-\alpha)^t x_0$$

$$= \alpha \boxed{[x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} x_1]} + (1-\alpha)^t \boxed{x_0}$$

Past *t* **observations** **bootstrap**

Exponential Smoothing

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} s_2$$

$$s_2 = \alpha x_1 + (1-\alpha)s_1 = \alpha x_1 + (1-\alpha)x_0 \text{ because } s_1 = x_0$$

$$s_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots + \alpha(1-\alpha)^{t-1} x_1 + (1-\alpha)^t x_0$$

$$= \boxed{\alpha}\boxed{[x_t + (1-\alpha)x_{t-1} + (1-\alpha)^2 x_{t-2} + \ldots + (1-\alpha)^{t-1} x_1]} + \boxed{(1-\alpha)^t}\boxed{x_0}$$

Past *t* **observations** **bootstrap**

weighted average

Assuming alpha > 0, the weight of each past term decreases as we move backward in history proportionally to the terms of a geometric progression $\{1, (1-\alpha), (1-\alpha)^2, (1-\alpha)^3, \ldots\}$

Exponential Smoothing

In general, for any given *T* it holds the following

$$s_T = \alpha \cdot \left[\sum_{i=0}^{T-2} (1-\alpha)^i x_{T-1-i} \right] + (1-\alpha)^{T-1} x_0$$

SJF vs. SRTF: Non-preemptive vs. Preemptive

- **SJF** (non-preemptive) → Once the CPU is given to a process this will execute until it completes its CPU burst

SJF vs. SRTF: Non-preemptive vs. Preemptive

- **SJF** (non-preemptive) → Once the CPU is given to a process this will execute until it completes its CPU burst
- **SRTF** (preemptive) → Preemption occurs whenever a new process arrives in the ready queue and its predicted CPU burst is shorter than the one remaining of the current executing process

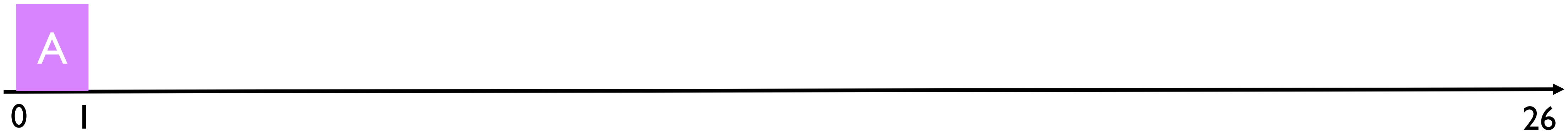
SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

0

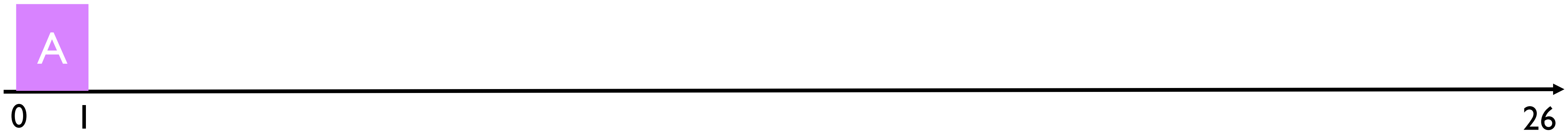
SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



SRTF: Example

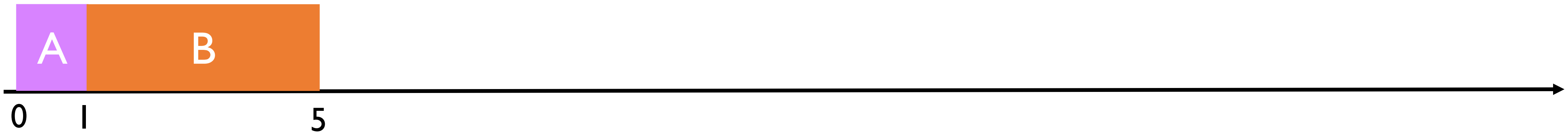
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



At time t=1 **B** arrives and its CPU burst (**4**) is less than the remaining CPU burst of **A** (**8-1=7**)

SRTF: Example

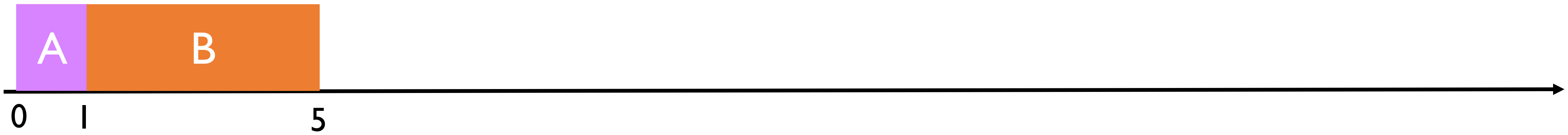
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



B is scheduled and will execute until it finishes its **4** CPU burst units

SRTF: Example

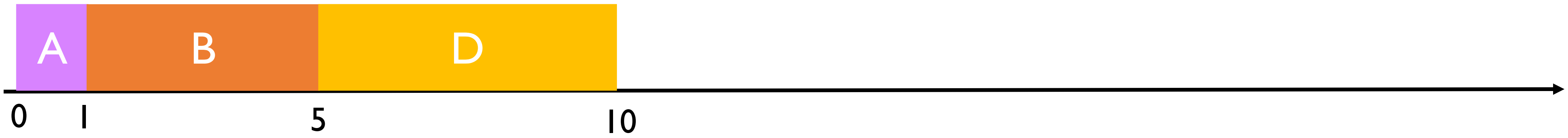
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



Both **C** and **D** are arrived with **9** and **5** CPU burst units, respectively
A has still **7** CPU burst units left…

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



D is scheduled and will execute until it finishes its **5** CPU burst units (no more jobs arrived in the meantime)

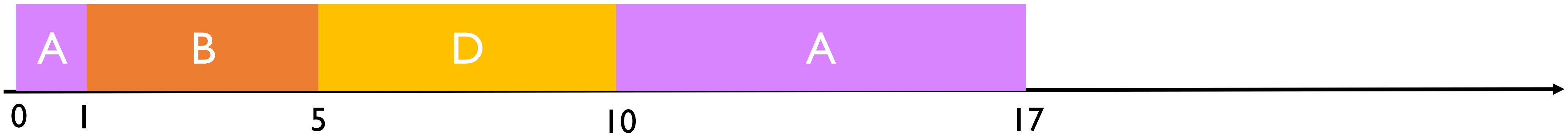
SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

A has still **7** CPU burst units left and C has **9**…

SRTF: Example

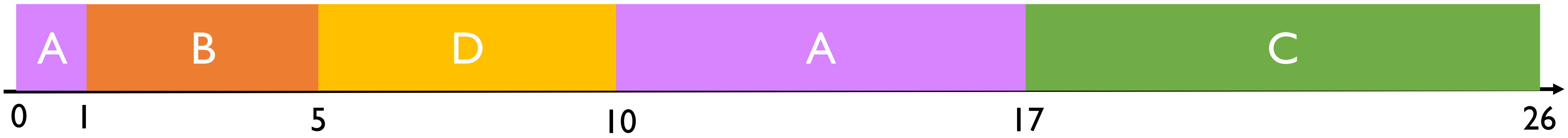
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



A is scheduled again until it finishes

SRTF: Example

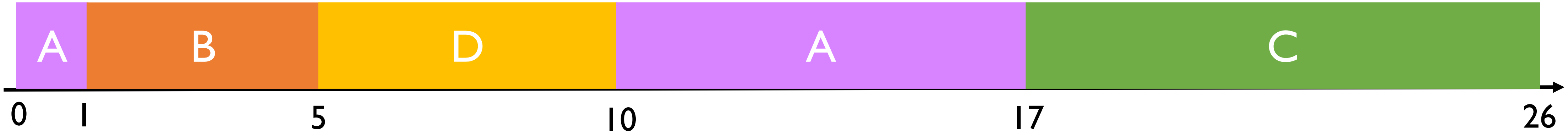
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



Eventually, C is scheduled as well

SRTF: Example

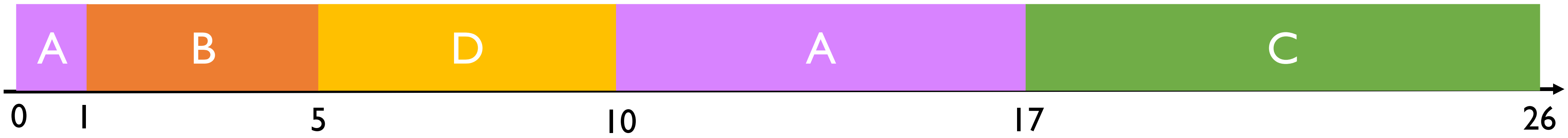
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



avg. waiting time =

SRTF: Example

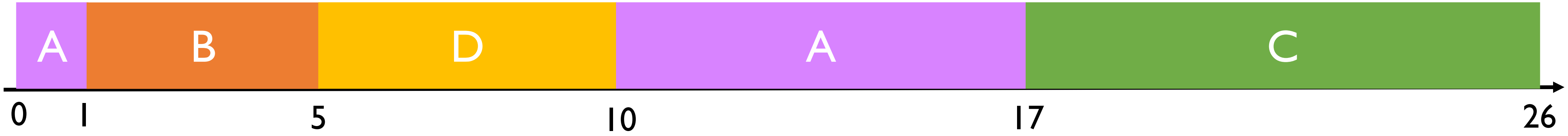
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



avg. waiting time = [(17-0-8)]/4

SRTF: Example

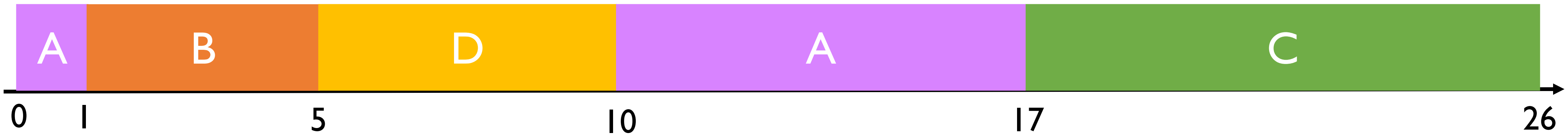
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



avg. waiting time = [(17-0-8) + (5-1-4)]/4

SRTF: Example

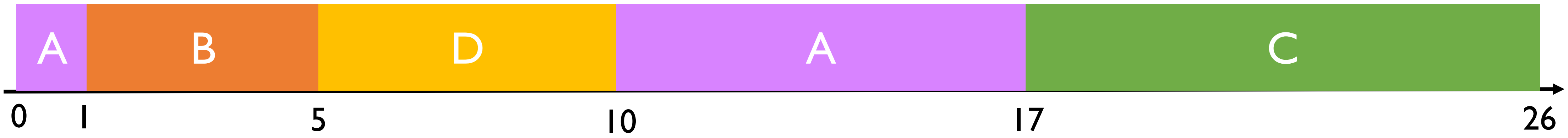
Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



avg. waiting time = [(17-0-8) + (5-1-4) + (26-2-9)]/4

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



avg. waiting time $= [(17-0-8) + (5-1-4) + (26-2-9) + (10-3-5)]\,/4 = 26/4 \sim \mathbf{6.5}$

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	**CPU burst**	**FCFS**	**RR**	**SJF**	**FCFS**	**RR**	**SJF**
A	50	50	150		0	100	
B	40	90	140		50	100	
C	30	120	120		90	90	
D	20	140	90		120	70	
E	10	150	50		140	40	
Avg.		**110**	**110**		**80**	**80**	

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	**CPU burst**	**FCFS**	**RR**	**SJF**	**FCFS**	**RR**	**SJF**
A	50	50	150	150	0	100	
B	40	90	140	100	50	100	
C	30	120	120	60	90	90	
D	20	140	90	30	120	70	
E	10	150	50	10	140	40	
Avg.		**110**	**110**	**70**	**80**	**80**	

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	**CPU burst**	**FCFS**	**RR**	**SJF**	**FCFS**	**RR**	**SJF**
A	50	50	150	150	0	100	100
B	40	90	140	100	50	100	60
C	30	120	120	60	90	90	30
D	20	140	90	30	120	70	10
E	10	150	50	10	140	40	0
Avg.		**110**	**110**	**70**	**80**	**80**	**40**

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)

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first
- SJF is a priority scheduling where priority is the predicted next CPU burst time

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- In practice, priorities are implemented using integers within a fixed range
  - No convention on whether "high" priorities use large or small numbers
  - Usually, low numbers for high priorities (0 = the highest possible priority)

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.
- **External** priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.
- **External** priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.
- Priority scheduling can be either **preemptive** or **non-preemptive**

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority
- Stuck jobs may eventually run when the system load is lighter or after a shutdown/crash and a reboot

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority
- Stuck jobs may eventually run when the system load is lighter or after a shutdown/crash and a reboot
- **Aging** → solves starvation by increasing the priority of jobs proportionally to the time they wait, until they are eventually scheduled

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!

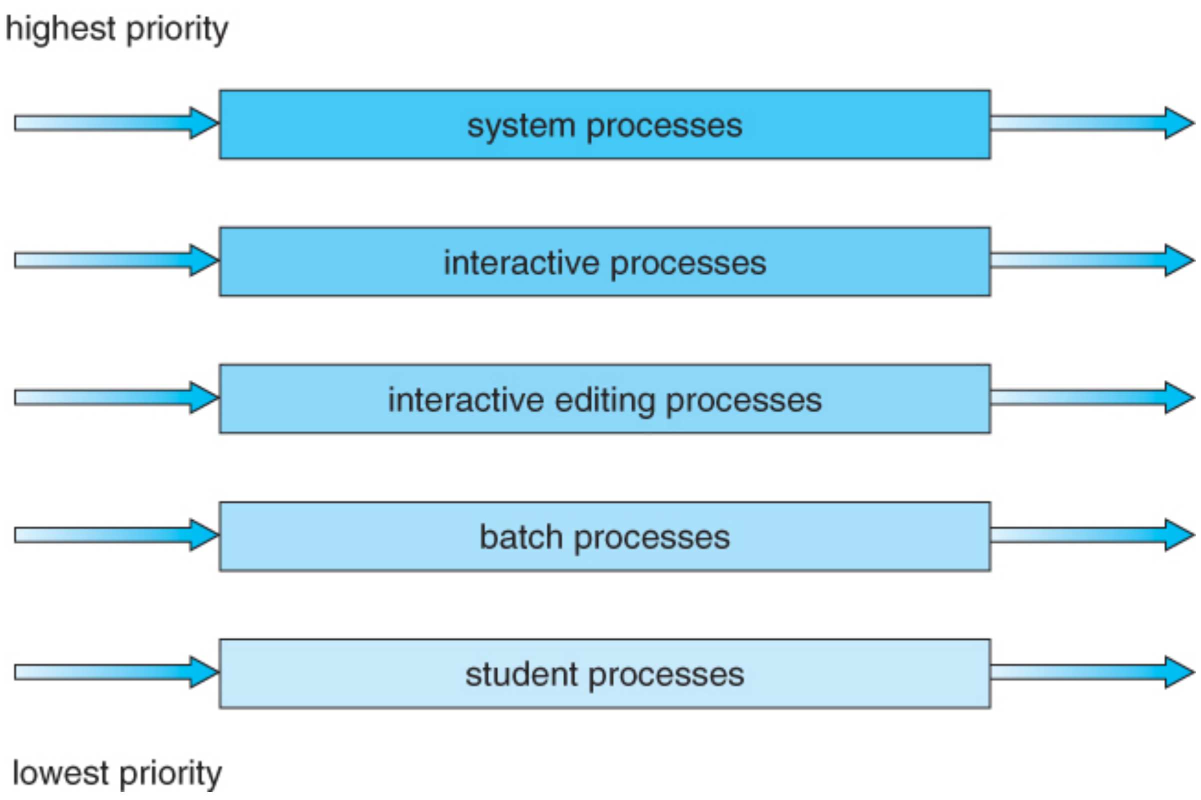
MLQ: Idea

- 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

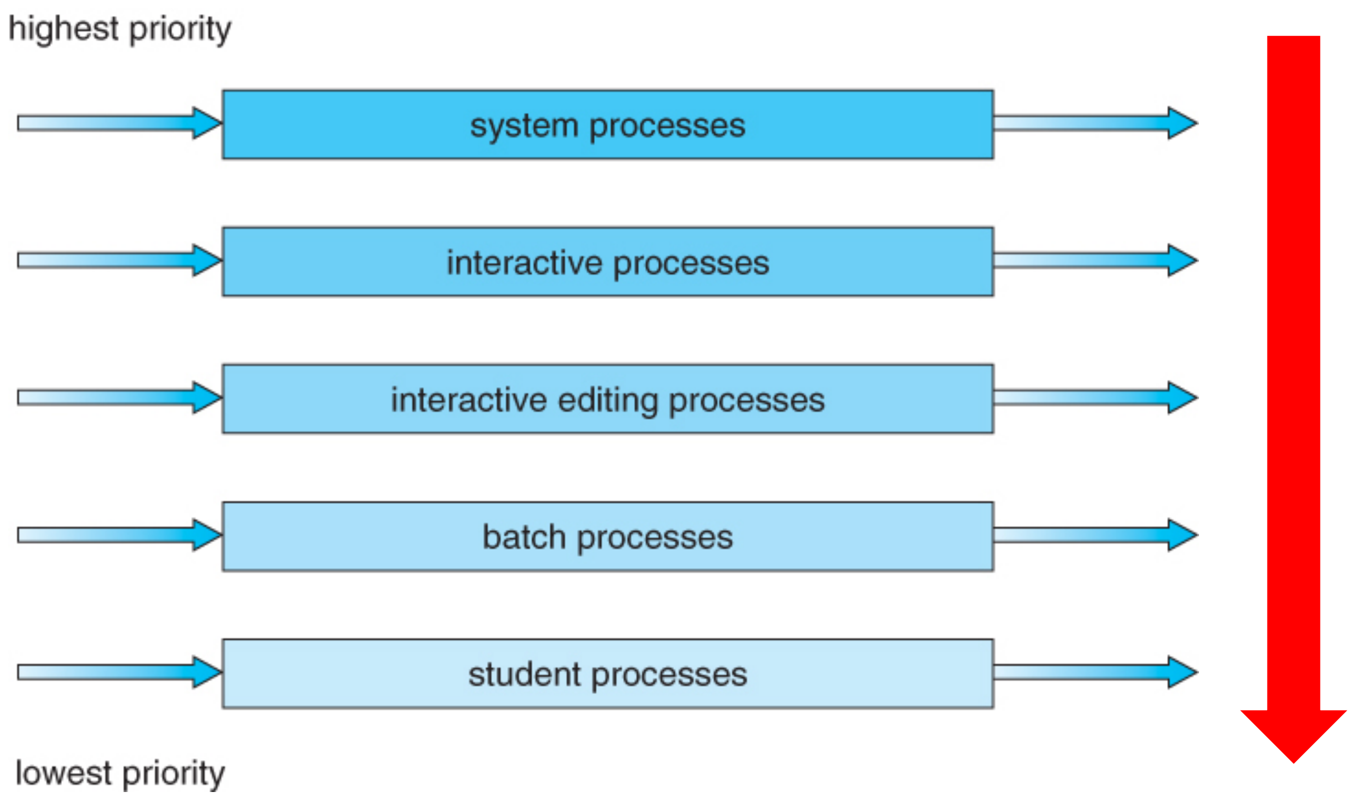
MLQ: Idea

- 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

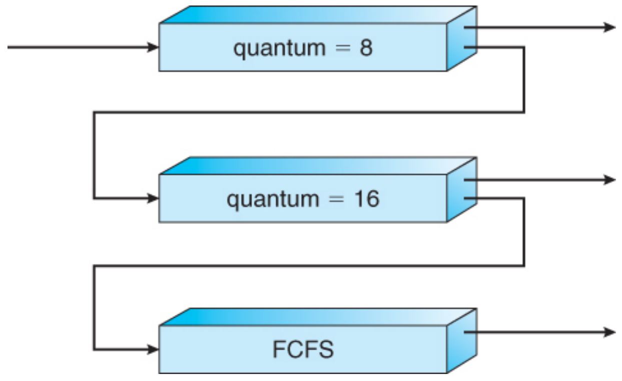
- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MLQ)
- **Multilevel Feedback-Queue (MLFQ)**

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)

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)
- 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
- 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 1

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

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

$$A^2_7 = \text{The job } A \text{ has executed } 2 \text{ time units after } 7 \text{ time units overall}$$

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 1

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 1

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 1

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

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

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} = total number of *short* jobs
n_{long} = total number of *long* jobs
$N = n_{short} + n_{long}$ = total number of jobs

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

$$\boxed{\begin{gathered}\mathrm{CPU}_{short} = \frac{m_{short}}{M} \\ \mathrm{CPU}_{long} = \frac{m_{long}}{M}\end{gathered}}$$

Lottery Scheduling: CPU Assignment Probability

m_i = number of tickets assigned to job i

N = 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