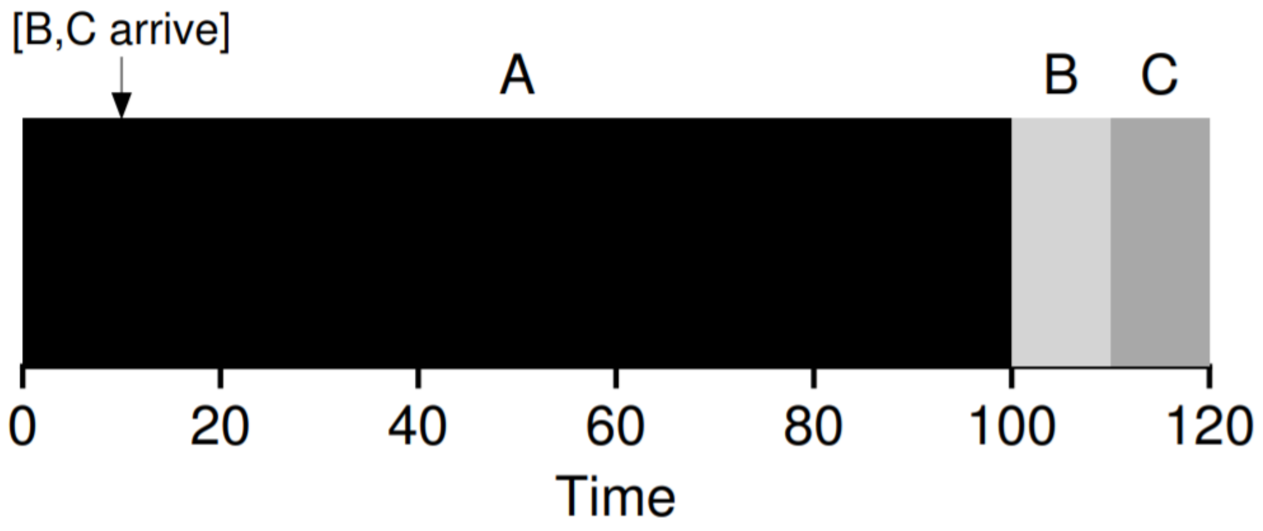
Part 3: Preemptive Policies

SRT, RR

Example #3

Assume 3 jobs (A, B, & C)

- A runs for 100s, B&C run for 10s each.
- A arrives at 0, B&C at 10s.



$$TT_{average} = \frac{100 + 100 + 110}{3} = 103.3$$

Scheduling Assumptions (Rev 3)

All tasks:

1. ~~Run for the same amount of time~~
2. ~~Arrive at the same time (roughly)~~
3. ~~Once started, run to completion~~
4. Only use the CPU
5. Have a known run-time.

only pre-empt when a new task arrives

pre-emp often

Policy #3: Shortest Remaining Time

if a new task arrives which has a shorter execution time than the current task, pre-empt

Assume 3 jobs (A, B, & C)

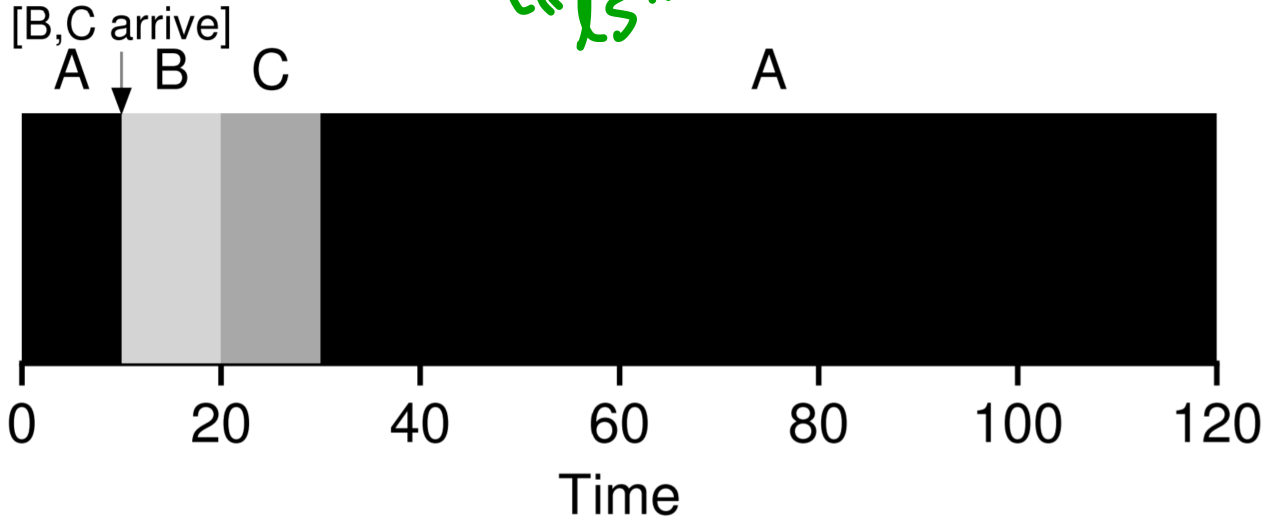
- A runs for 100s, B&C run for 10s each.
- A arrives at 0, B&C at 10s.

Example #3, revised

Assume 3 jobs (A, B, & C) (#1) starvation)

- A runs for 100s, B&C run for 10s each.
- A arrives at 0, B&C at 10s.

└"ℓs"



$$TT_{average} = \frac{(120-0) + (20-10) + (30-10)}{3} = 50$$

Scheduling Metric #2

Response Time:

$T_{response} = RT = T_{first\ execution} - T_{arrival}$

$\longrightarrow$ how long until the task responds

$\longrightarrow$ small as possible when considering the user

Policy #4: Round-Robin

Big idea:

- don't pre-empt on arrival
- Instead pre-empt every so often

time slice
(the quantum)

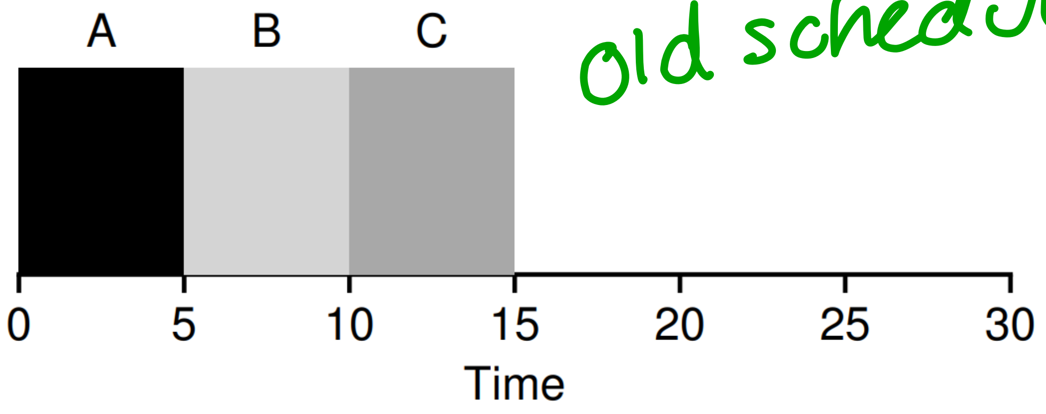
1. move the running task to the back of the ready queue
2. schedule task that is at the front of the queue

⇒ effectively a pre-emptive FIFO

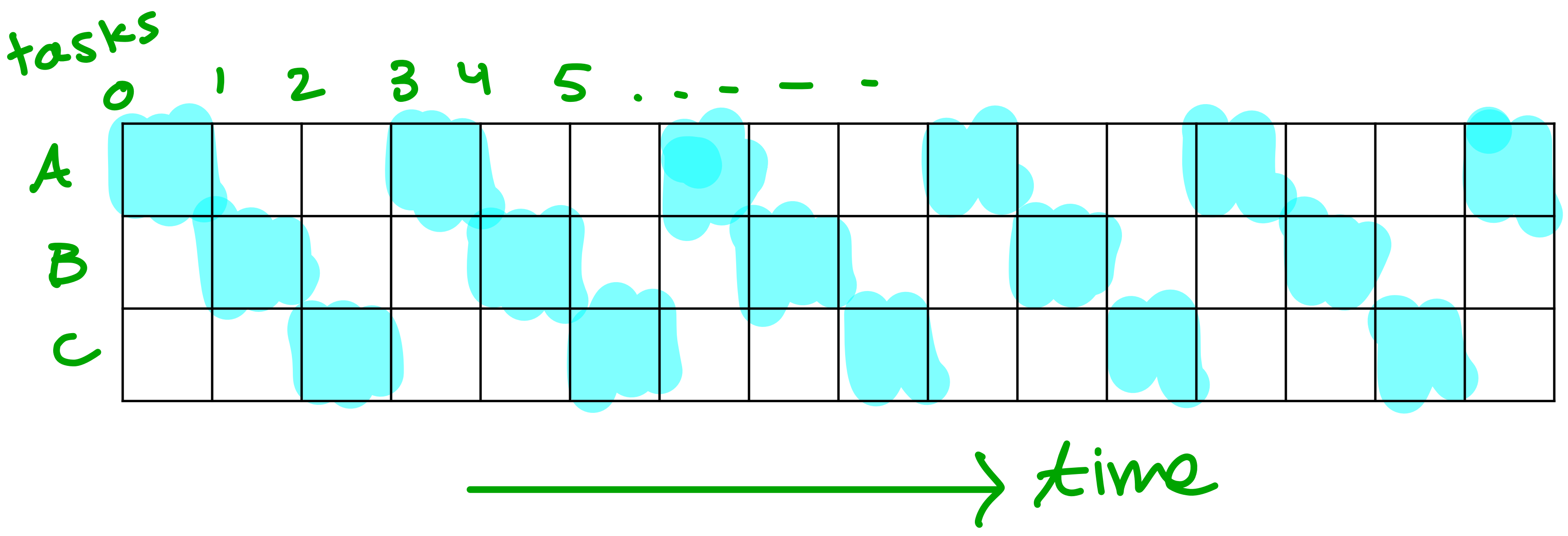
Example #4

3 tasks (A, B, & C):

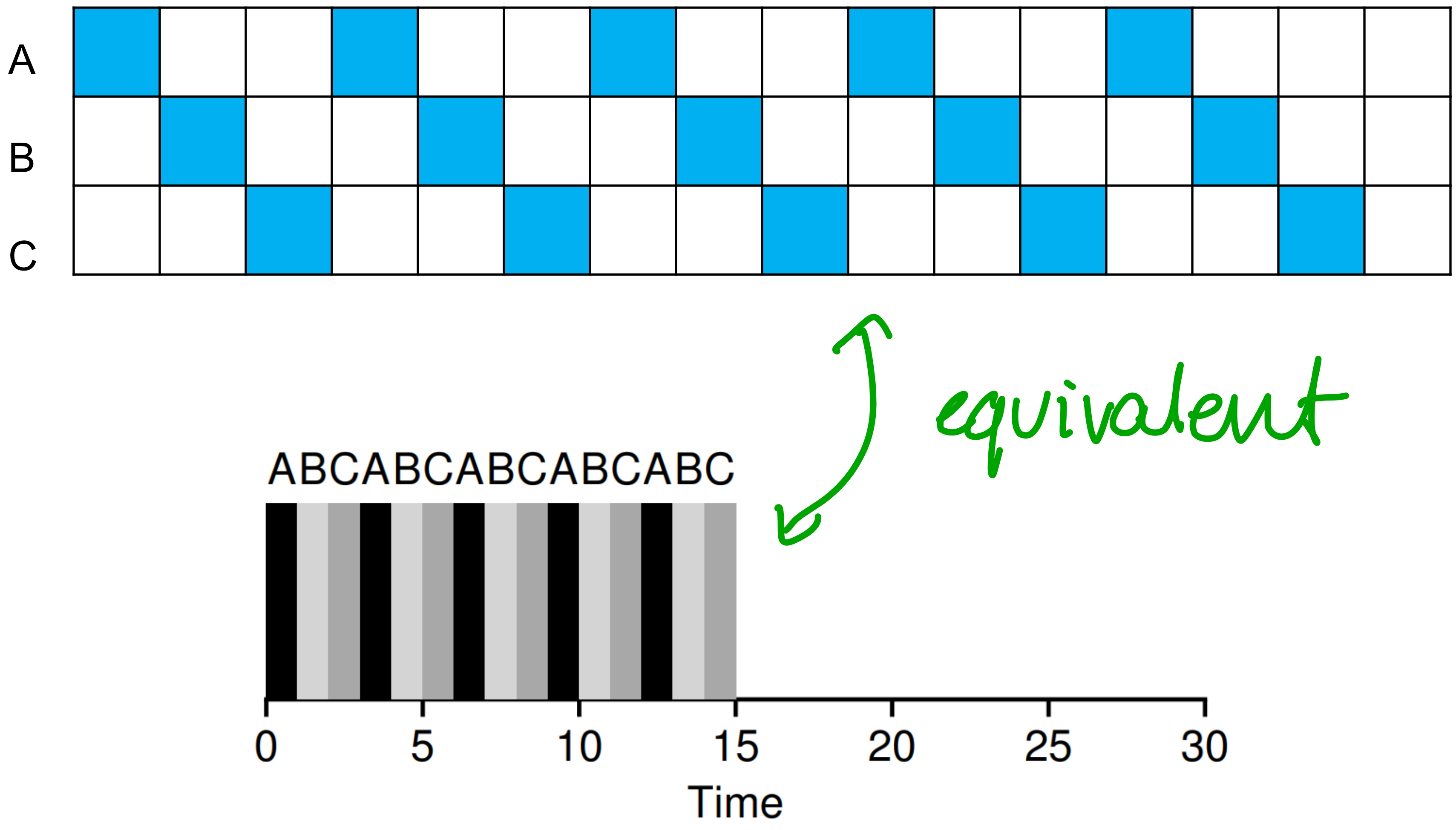
- Arrive at T = 0
- Each run for 5s



old schedule

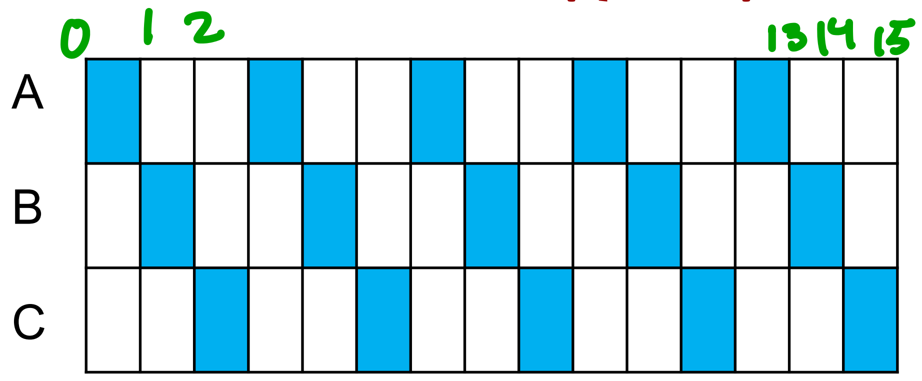


Example 4



Example 4: Comparison

Round Robin (q = 1)



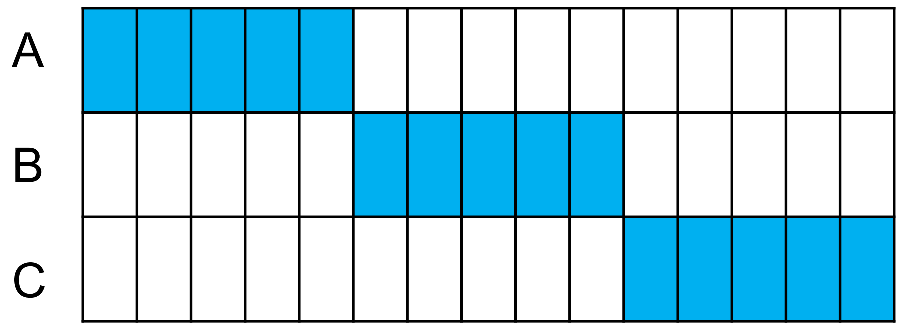
$$RT_{average} = \frac{(0-0)+(1-0)+(2-0)}{3}$$

$$\boxed{= 1s}$$

$$TT_{average} = \frac{13+14+15}{3}$$

$$\boxed{= 14s}$$

FCFS / SPN / SRT



$$RT_{average} = \frac{(0-0)+5+10}{3}$$

$$\boxed{= 5s}$$

$$TT_{average} = \frac{5+10+15}{3}$$

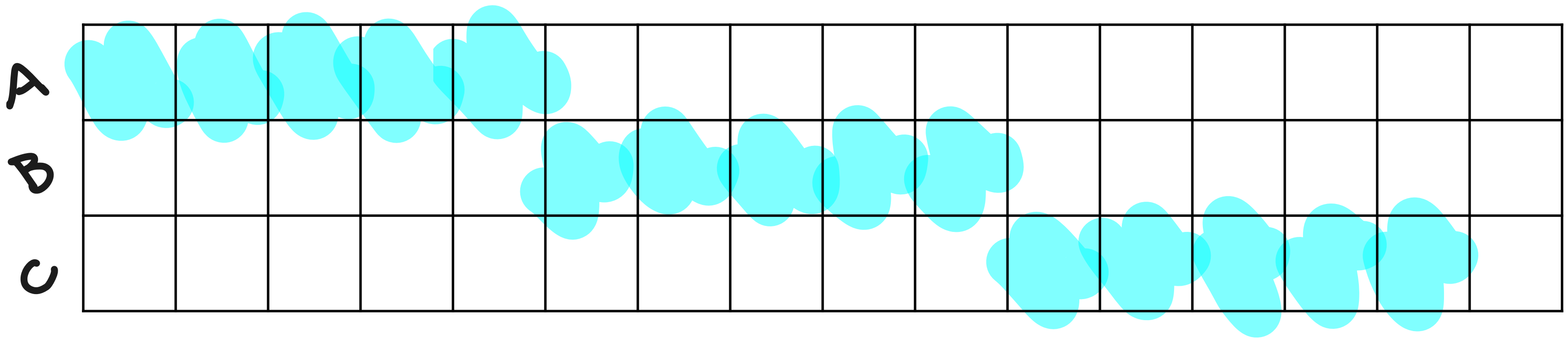
$$\boxed{= 10}$$

What Happens if We Set q = 5?

3 tasks (A, B, & C):

- Arrive at T = 0
- Each run for 5s

The length of the time slice is critical

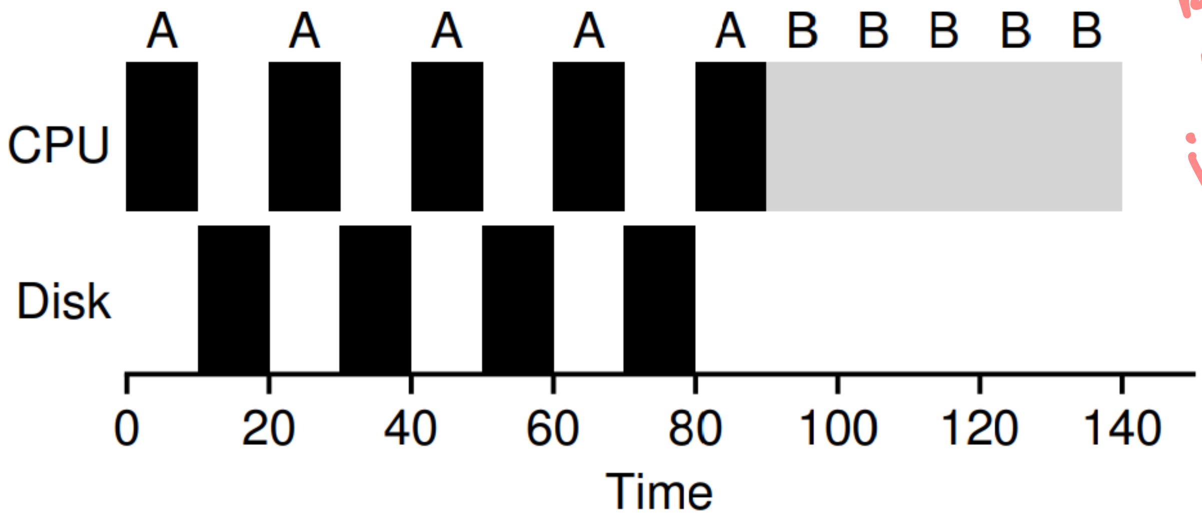


Scheduling Assumptions (Rev 4)

All tasks:

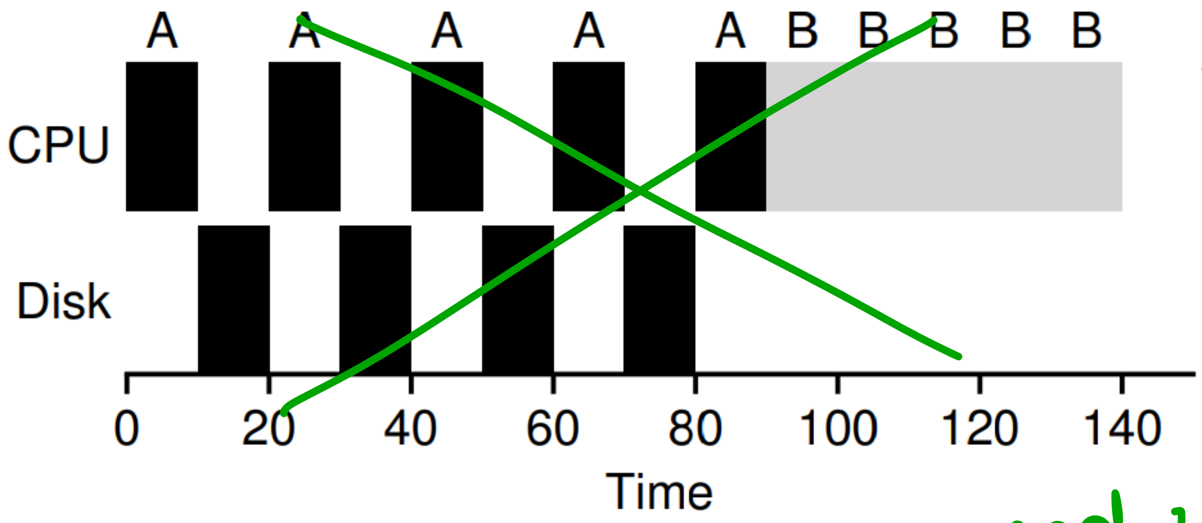
1. ~~Run for the same amount of time~~
2. ~~Arrive at the same time (roughly)~~
3. ~~Once started, run to completion~~
4. ~~Only use the CPU~~ add I/O
5. Have a known run-time.

Incorporating I/O

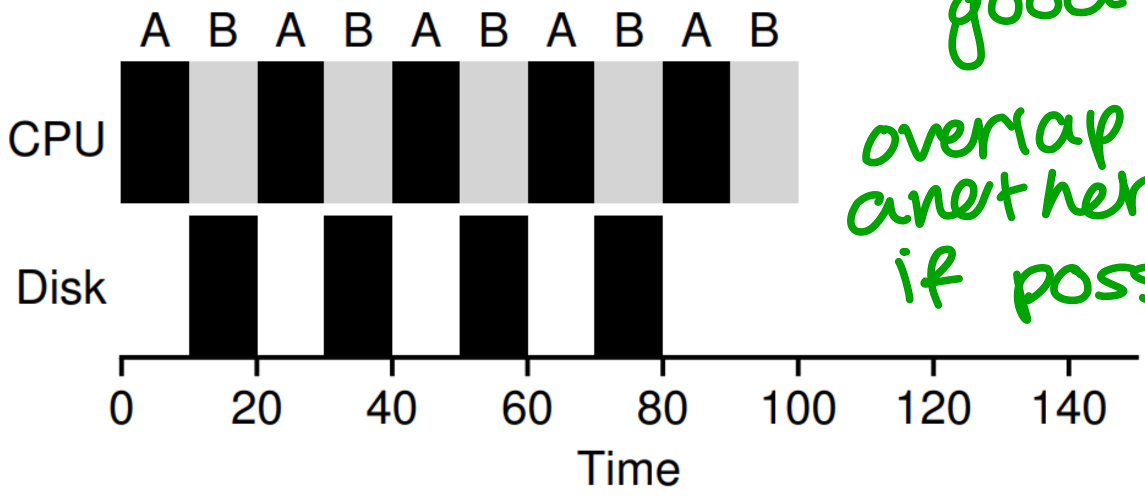


bad way to incorporate I/O

Incorporating I/O



Bad



good ✓

overlap I/O w/ another task if possible

Figure 7.9: **Overlap Allows Better Use Of Resources**

Interlude: Project 4

Part 4: The RealWorld™

We finally break all the bad assumptions

Scheduling Assumptions (Rev 5)

All tasks:

1. ~~Run for the same amount of time~~
2. ~~Arrive at the same time (roughly)~~
3. ~~Once started, run to completion~~
4. ~~Only use the CPU~~
5. ~~Have a known run time.~~

Predicting Run Time: Simple Average (a)

1. Track historical Runtimes
2. average history to predict the future

$$\underset{\text{predicted execution time}}{S_{n+1}} = \underset{\text{\# of bursts}}{\frac{1}{n}} \Big(\overbrace{T_0 + T_1 + \ldots + T_n}^{\sum_{i=0}^{n} T_i} \Big)$$

historical run time

Predicting Run Time: Simple Average (b)

Key Idea: No need to recalculate the entire sum

$$S_{n+1} = \frac{1}{n}\sum_{i=1}^{n} T_i$$

$$S_{n+1} = \frac{1}{n} T_n + \underbrace{\left(1 - \frac{1}{n}\right)}_{\frac{n-1}{n}} S_n$$

every term is weighted equally

Predicting Run Time: Exponential Average

Key Idea: Give more weight to recent history

$$S_{n+1} = \frac{1}{n}T_n + \left(1 - \frac{1}{n}\right)S_n$$

$$S_{n+1} = \alpha \cdot T_n + (1-\alpha) S_n$$

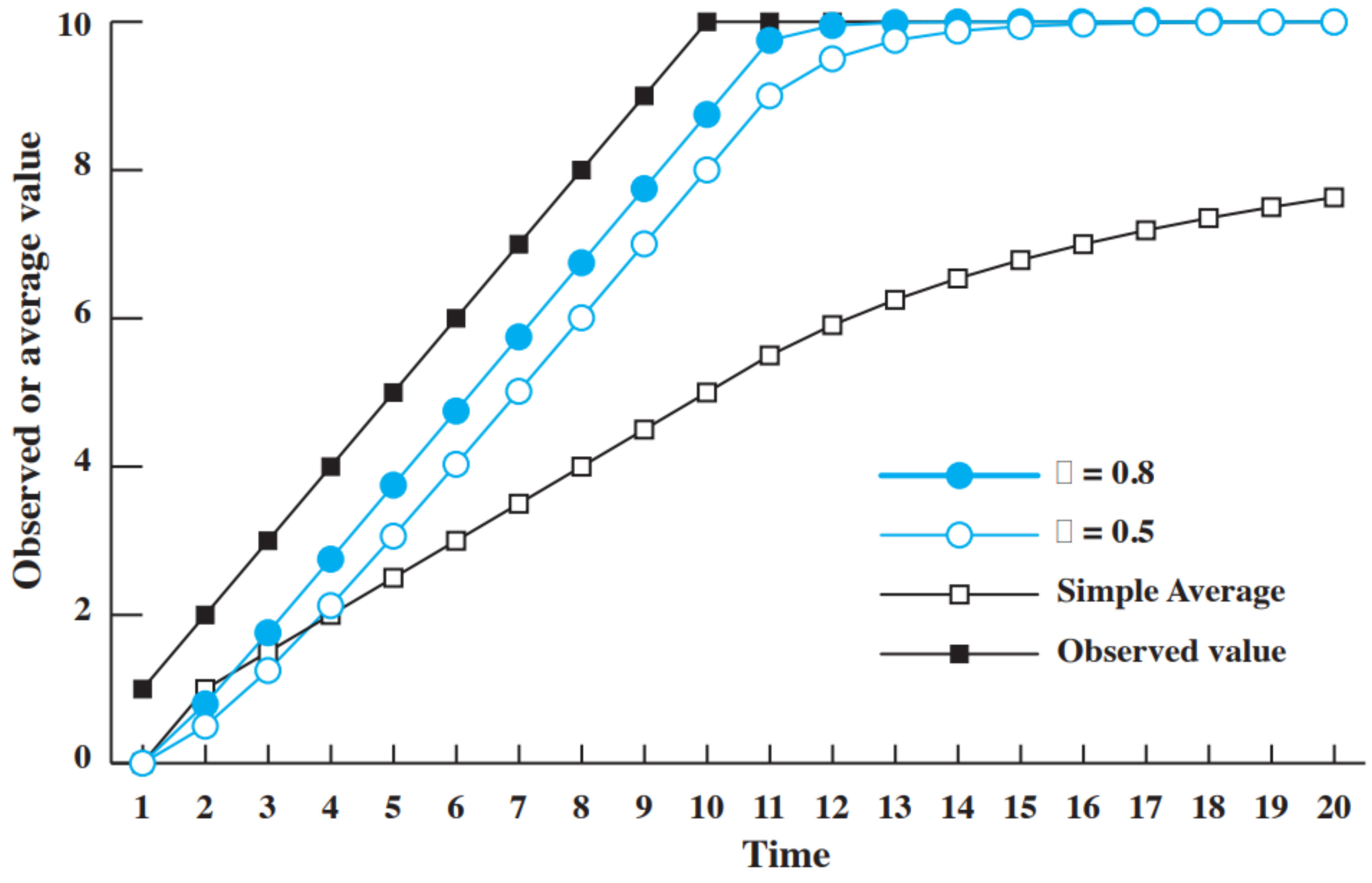
↳ constant
ex. 0.80
"80% weight on most recent term

$= \alpha T_n + (1-\alpha)[\alpha T_{n-1} + (1-\alpha)S_{n-1}]$

$= \alpha T_n + \alpha(1-\alpha)T_{n-1} + (1-\alpha)^2 S_{n-1}$

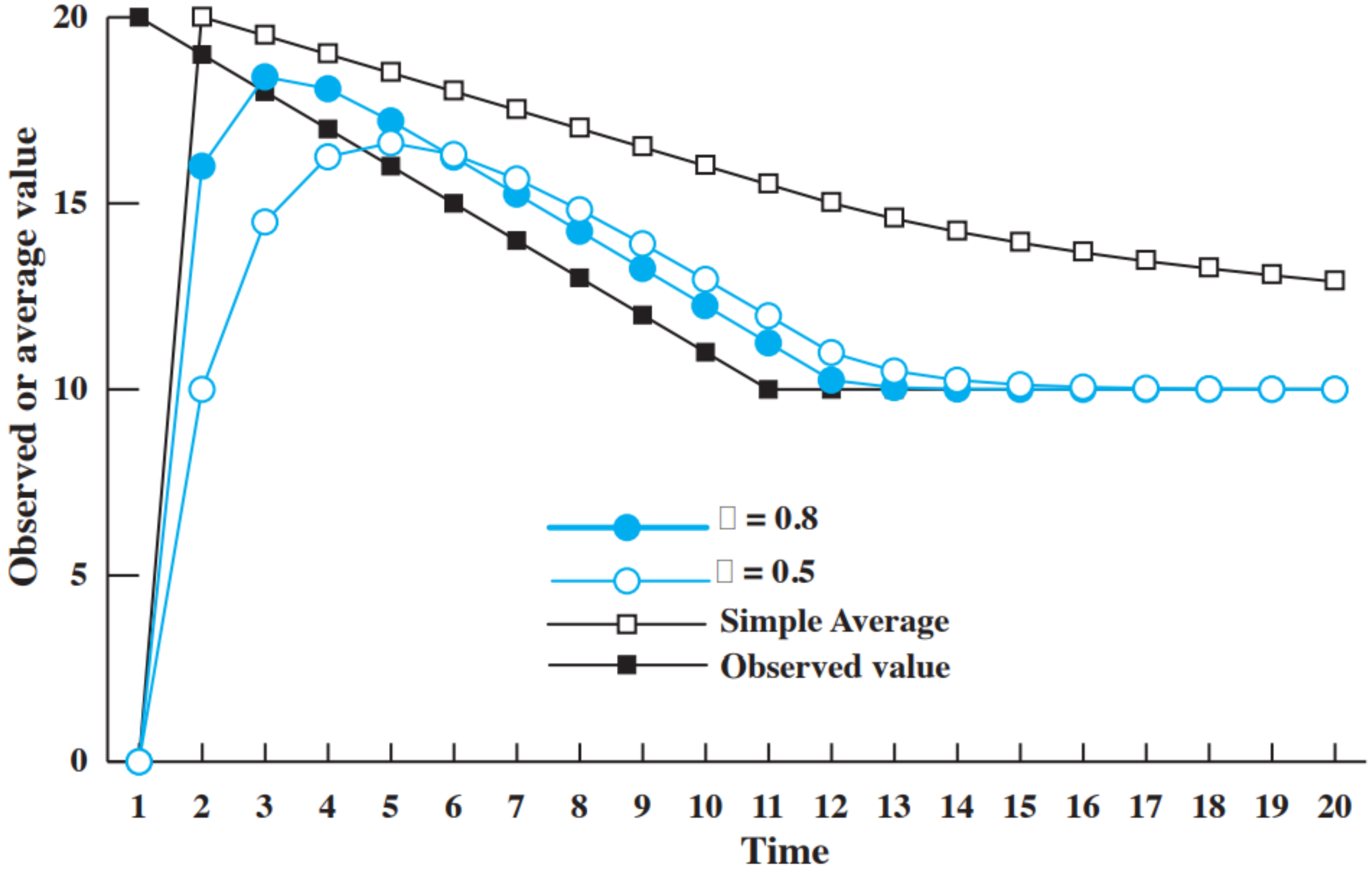
$= \alpha T_n + \alpha(1-\alpha)T_{n-1} + \ldots + \alpha(1-\alpha)^{n-1}T_1 + (1-\alpha)^n S_1$

How Well Does Averaging Work?



(a) Increasing function

How Well Does Averaging Work?



(b) Decreasing function

When Might We Need This?

…that was a lot of math. When is it useful?

1. SPN (shortest process next)
2. SRT (shortest remaing time)
3. HRRN

Policy #5: Highest Response Ratio Next

Response Ratio:

$$R = \frac{W + S}{S}$$

how long a task has been waiting

pred exec time

account for the age of a task and how long it will execute

$\Rightarrow$ a balance b/w FCFS and SPN/SRT + fixes starvation

Wouldn't it Be Nice

If we…

1. Didn't need to know (/predict) service time
2. Could *balance* turnaround and response time?

multi level feedback queues

MLFQ