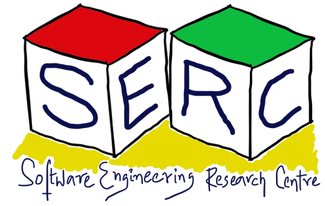
CS3.301 Operating Systems and Networks

Process Virtualisation - Policies (Scheduling)

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.
- Other online sources which are duly cited

Quick Recap

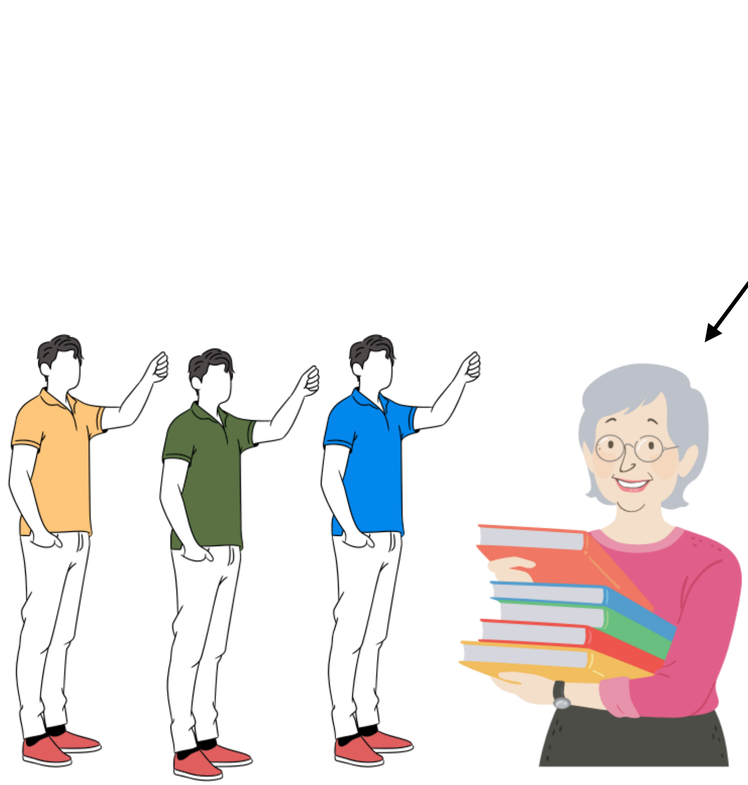
- Process is a running program
- They provide API with clear abstractions (System calls - fork(), exec(),..)
- Have states -> makes use of data structures to save values
- Gives a feeling that each process has its own CPU
- Hardware provides support - LDE
- OS switches between processes -> Context switch

How to decide which process to run on context switch?

Need for Policies (Scheduling)

Which process to schedule next on context switch?

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?

Scheduling in the Library Scenario

What we need to know to ensure good policy?

- How many users want to go to the reference section?
- What's the purpose? - What type of book they want to read?
- How much time are they expected to be in the reference section?
- How frequently are new users coming in?

Essentially it would be good to have these estimates to make a good policy!

What does it mean Concretely?

- For scheduling we need an idea of **workload**
  - Assumptions about processes running in the system
    - Number of processes
    - RAM required
    - CPU utilisation
    - Any Input/Output, if yes what kind?
    - ….

Lets start with some workload assumptions

Each process that is ready/needs to be executed and those executing - **Job!**

Some Assumptions:

1. Each job runs for **same amount of time**
2. All jobs **arrive** at the **same time**
3. All jobs **only use the CPU** (No I/O)
4. The **run time** or execution time of each job is **known**

How good is the policy?
Some Key Scheduling Metrics

- Metric is something we used to measure
- Performance metric: Turnaround time
  - Time difference between job completion time and the arrival time

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- Another metric is fairness - Jains fairness index: How fair is the scheduling?
  - May not go hand in hand with performance

Scenario 1

All Assumptions in tact

- Imagine three jobs - Whatsapp, Skype and Teams update arriving at same time
- Each of them take same time to complete

Process	Arrival	Time to Complete
Whatsapp (w)	~0	20
Skype (S)	~0	20
Teams (T)	~0	20

How to go about this?

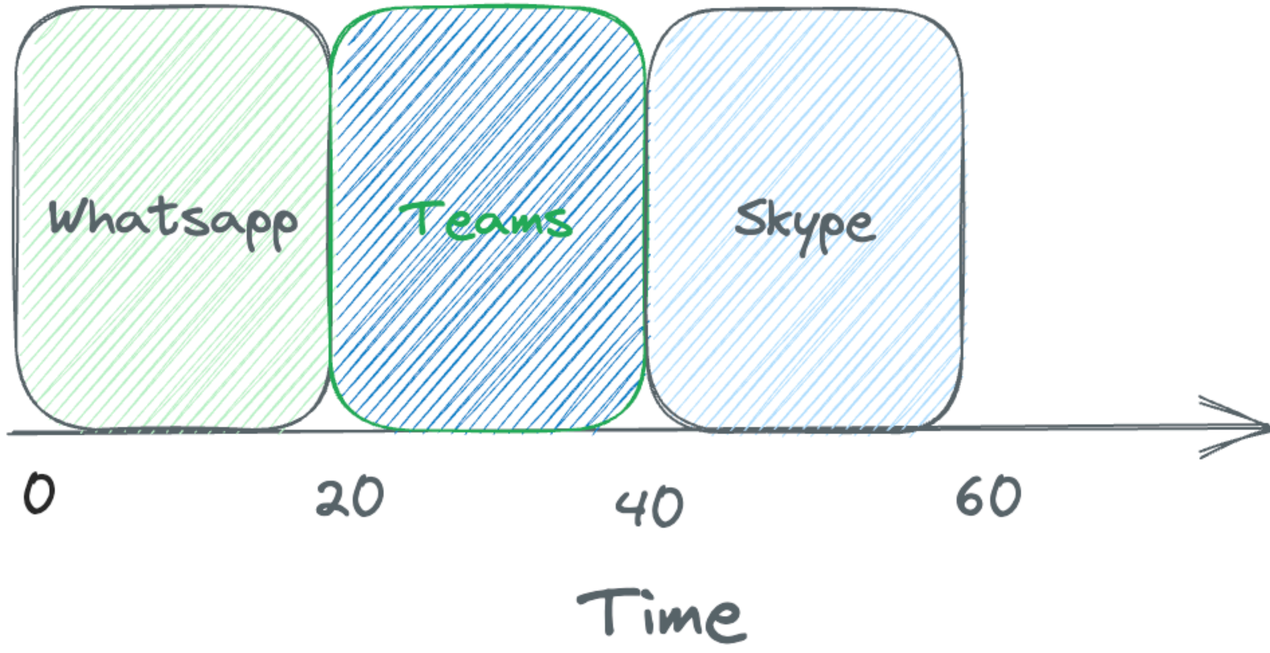
First Come First Serve Policy

- The most basic algorithm a scheduler can implement
  - Whoever comes first, give them the access

- Assume that they arrive at the same time - At time = 0
  - For sake of simplicity W just arrived before T which just arrived before S

First Come First Serve (FCFS) Policy

- **Policy:** Schedule the job came first
- As soon as it is done, schedule the job that came next, continue
- There is an assumption here that each job runs for the same time
  - What if that's not the case?
  - Let us relax this assumption

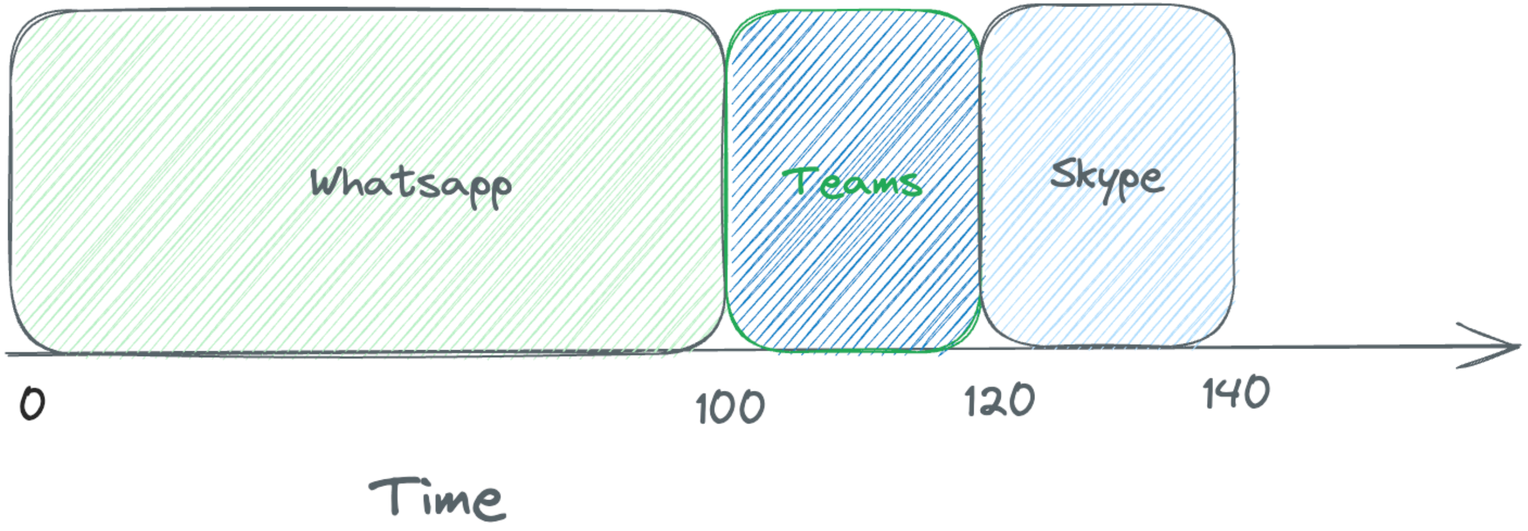


$$Avg(T_{turnaround}) = \frac{20 + 40 + 60}{3}$$

$$= 40$$

What if each job no longer runs for same time?

Relaxing assumption 2



$$Avg(T_{turnaround}) = \frac{100 + 120 + 140}{3}$$

$$= 120$$

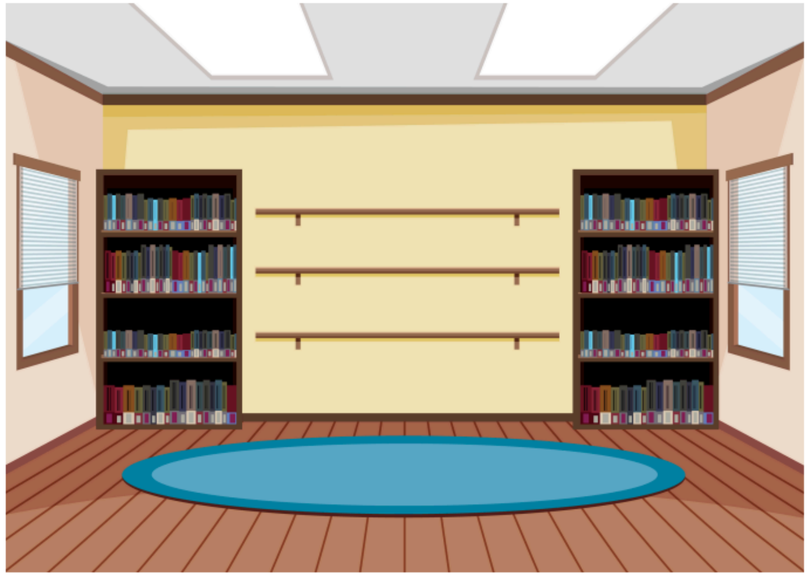
FCFS is not that great

Convoy Effect

- Waiting time can go very high
  - Convoy effect!
  - Think about waiting in single line in grocery store where you just have one item to purchase

What if?

Visitors/Users need to use the reference room, but who to give access to now? How to determine whom to give access to?



- Every one said that they will need this much time for accessing the reference section
- Librarian schedules based on the time they say

Shortest Job First (SJF) Policy

- Idea originating from operations research
- **Policy:** Run the shortest job first

Process	Arrival	Time to Complete
W	0	100
S	0	20
T	0	20

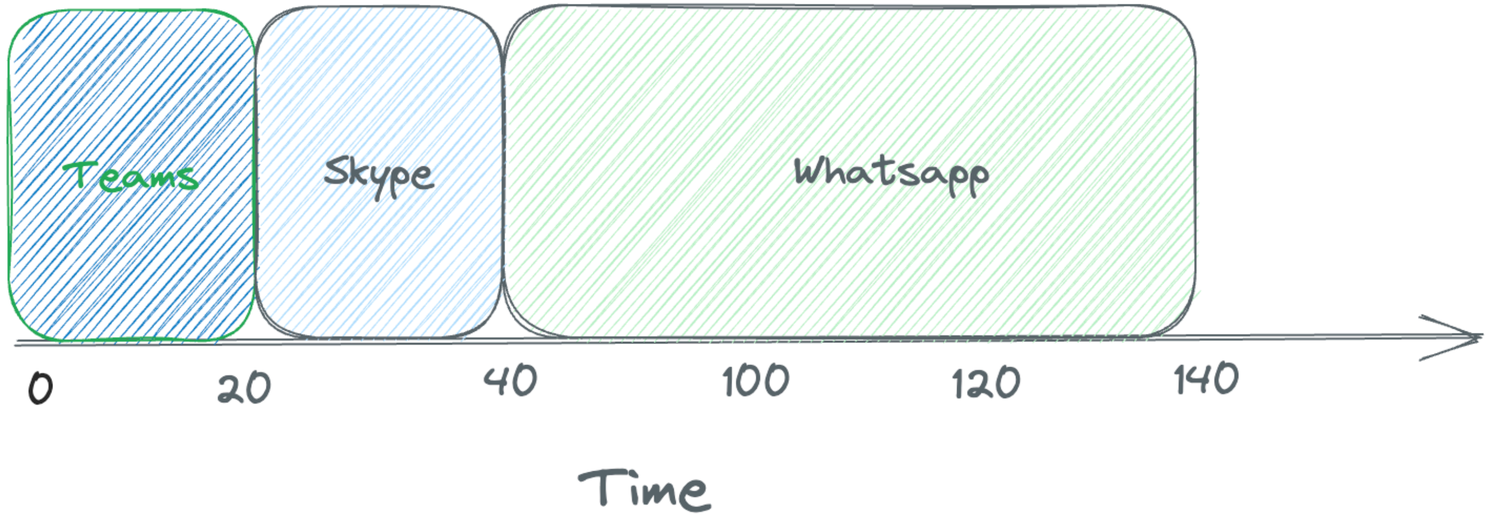
How to go about this?

Shortest Job First (SJF) Policy

- Assume that all jobs came at the same time
- Clearly whatsapp takes most amount of time

$$Avg(T_{turnaround}) = \frac{20 + 40 + 140}{3}$$

$$= 66.3$$



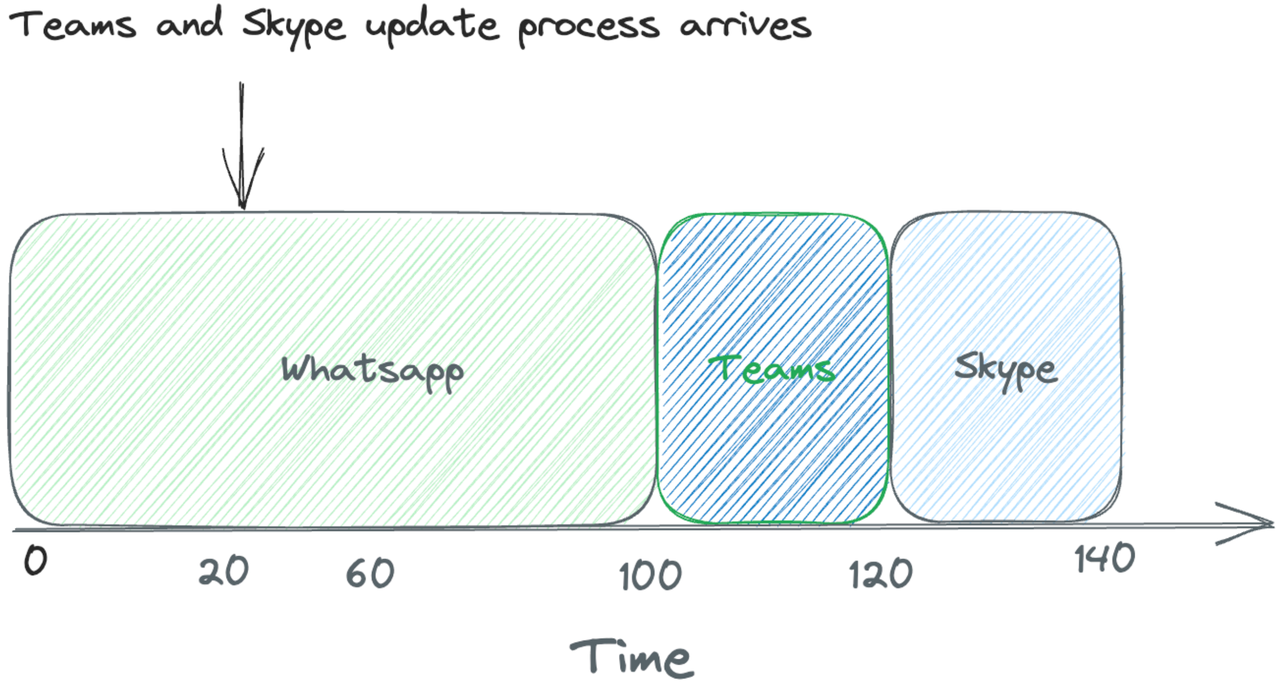
Is that a bit too unrealistic? - In reality jobs can arrive at any time

Shortest Job First (SJF) Policy

- Whatsapp job arrives first
- Teams and Skype jobs arrives around t = 20

$$Avg(T_{turnaround}) = \frac{100 + 100 + 120}{3}$$

$$= 106.6$$

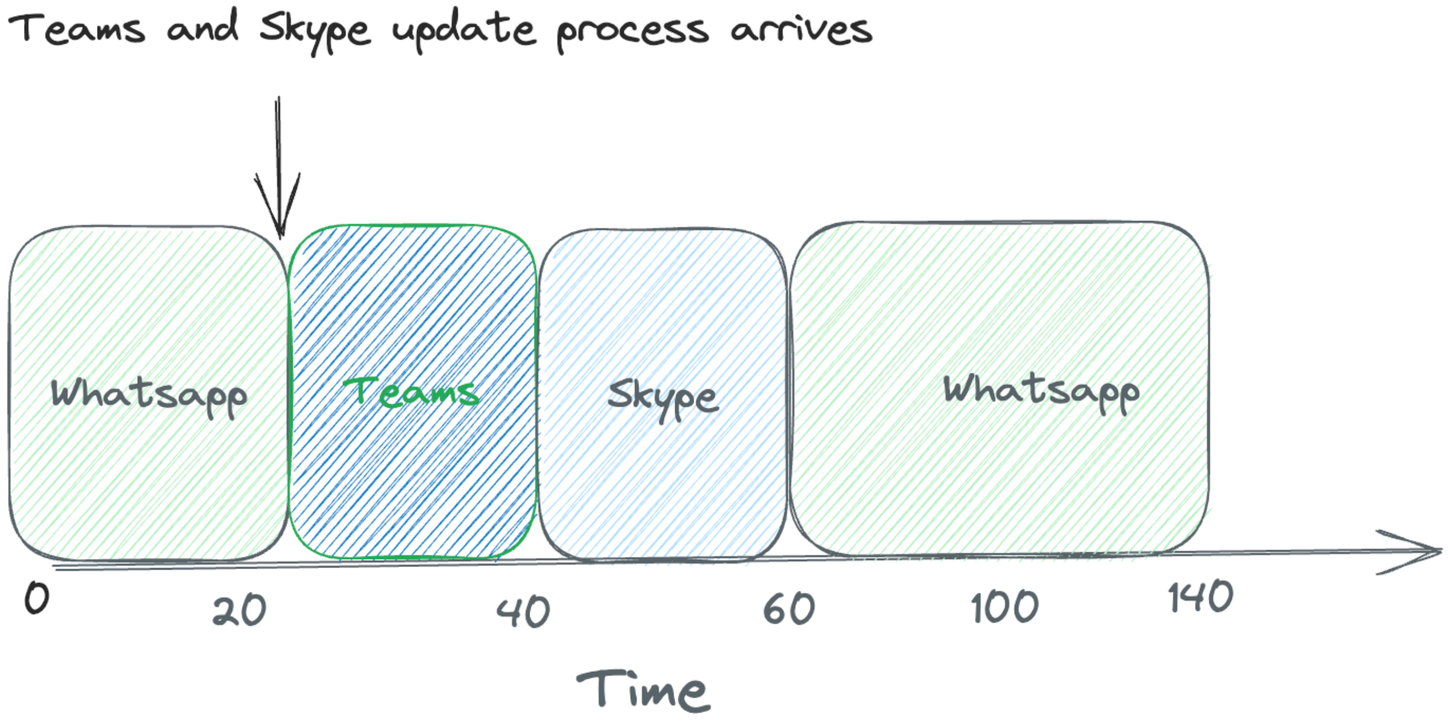


Even worst!! How to improve?

Shortest Time to Completion First (STCF)

- Adding preemption to Shortest Job First (SJF) Policy
  - More like preemptive SJF
- **Policy:** Any time a new job enters the system,
  - Check how much time is remaining for existing jobs
  - Check the time that is required for the new one
  - Execute the one that shall complete first

Shortest Time to Completion First (STCF)



$$Avg(T_{turnaround}) = \frac{(140 - 0) + (40 - 20) + (60 - 20)}{3}$$

$$= 66.3$$

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?

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

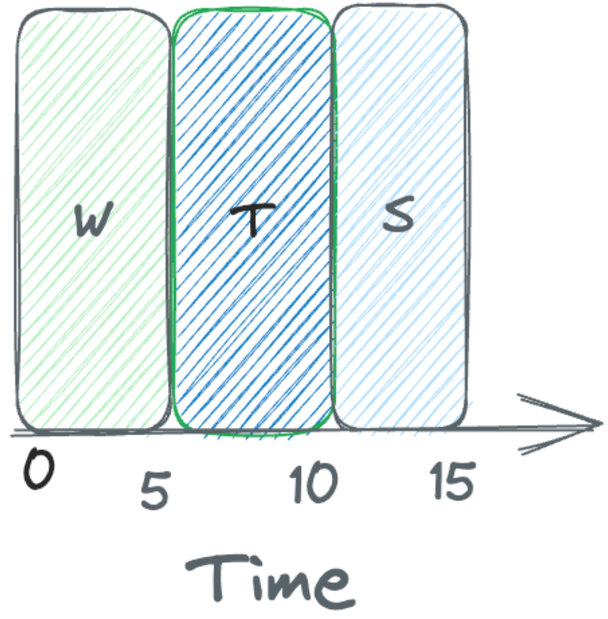
Round Robin Scheduling

- 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



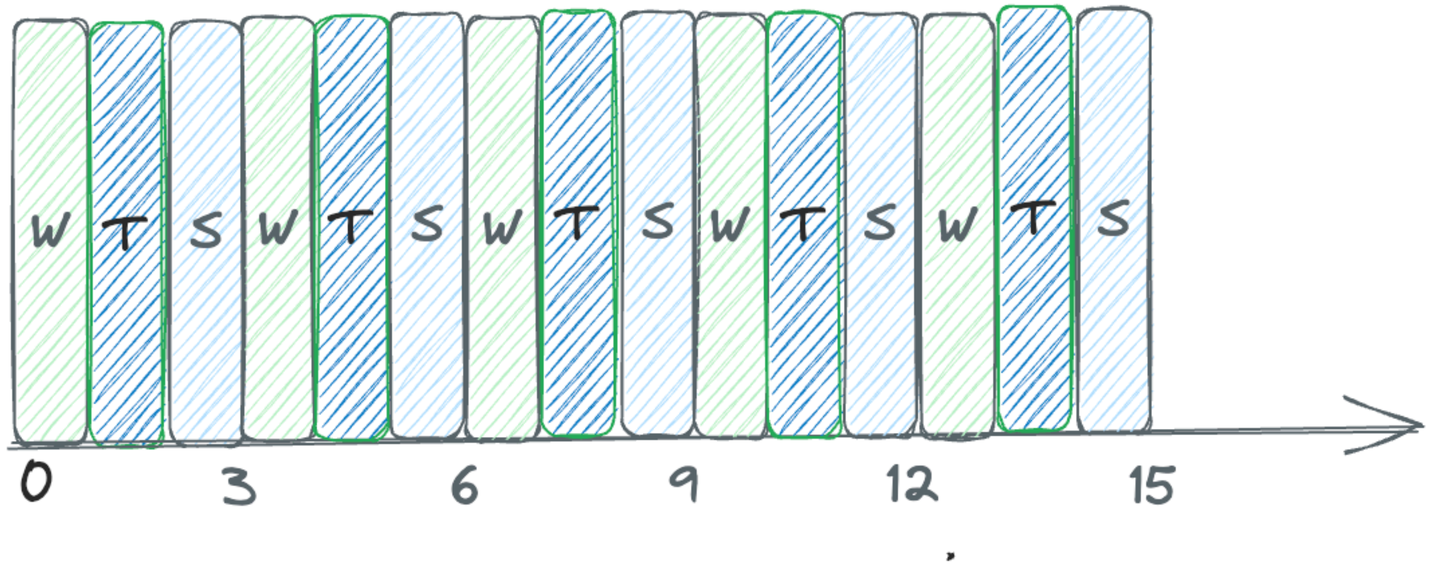
$$Avg(T_{turnaround}) = \frac{5 + 10 + 15}{3} = 10$$

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

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

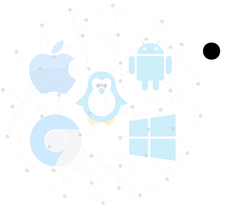
- Time slice plays a critical role in response time part
  - Too small time slice can result in an overhead - Too much Context Switch!
- RR is a good scheduling method
  - Key thing is to find an optimal time slice
  - Response time is the only metric
  - What about turnaround?

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

$$= 14!$$

Remember: Trade-off

- Turnaround time only cares about completion
  - Fairness of scheduling does not come into the picture
  - Processes may starve
- The key aspect is to consider trade-off's
  - Very important in system design
  - Often among quality attributes
  - Eg: security vs performance



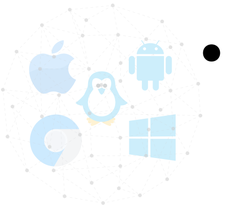
Continuing on the Assumptions

- 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?

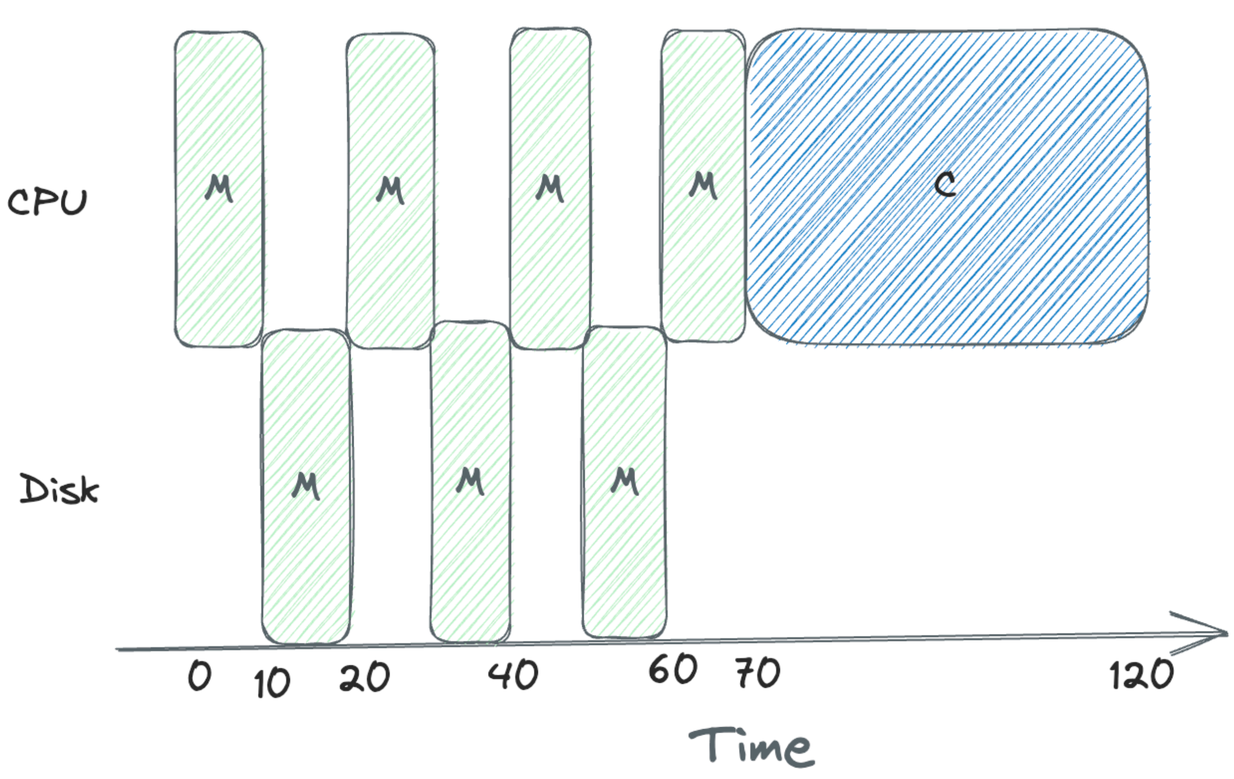
Incorporating I/O

- 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



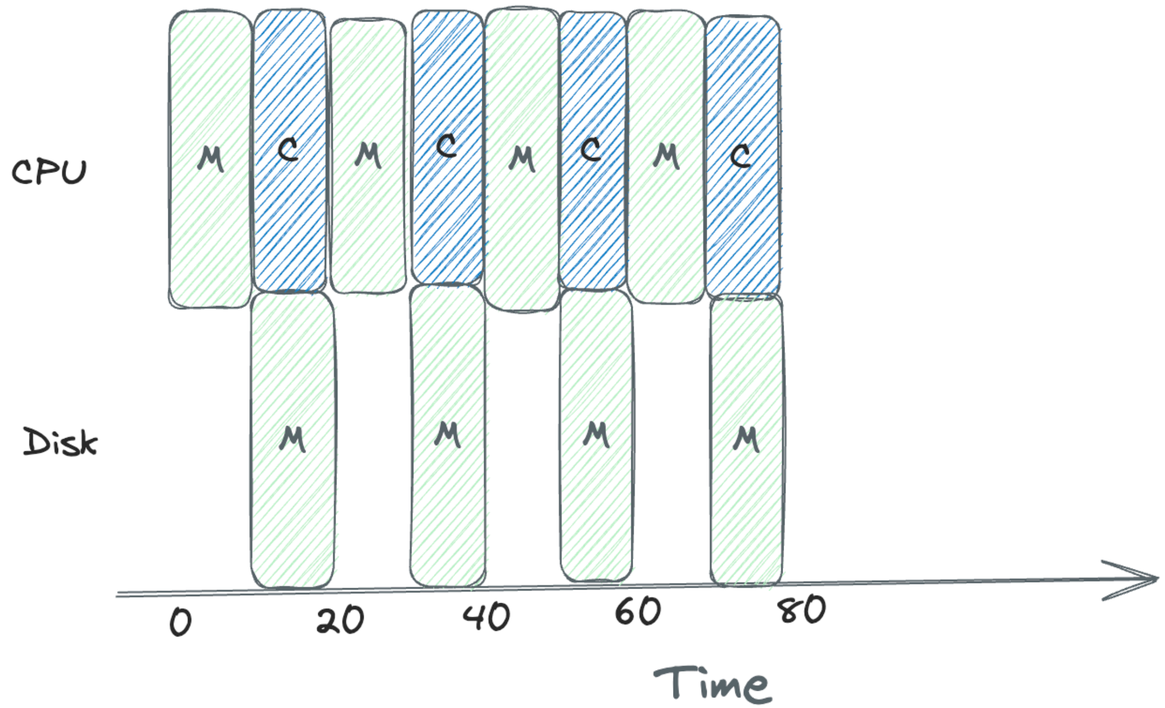
Can we do better?

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?



One more assumption to consider

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

Thank you

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