CS3.301 Operating Systems and Networks

Process Virtualisation - Mechanisms and Policies (Part 1)

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

SERC
Software Engineering Research Centre

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 Remzi et al.
- Modern Operating Systems, Tanenbaum et al.
- Other online sources which are duly cited

Lets draw some Parallels

Library

Library Users

Reference Books

- As a visitor/user in the library - check sections, read books, magazines,..
- What about accessing the reference section and get access to some treasured books?

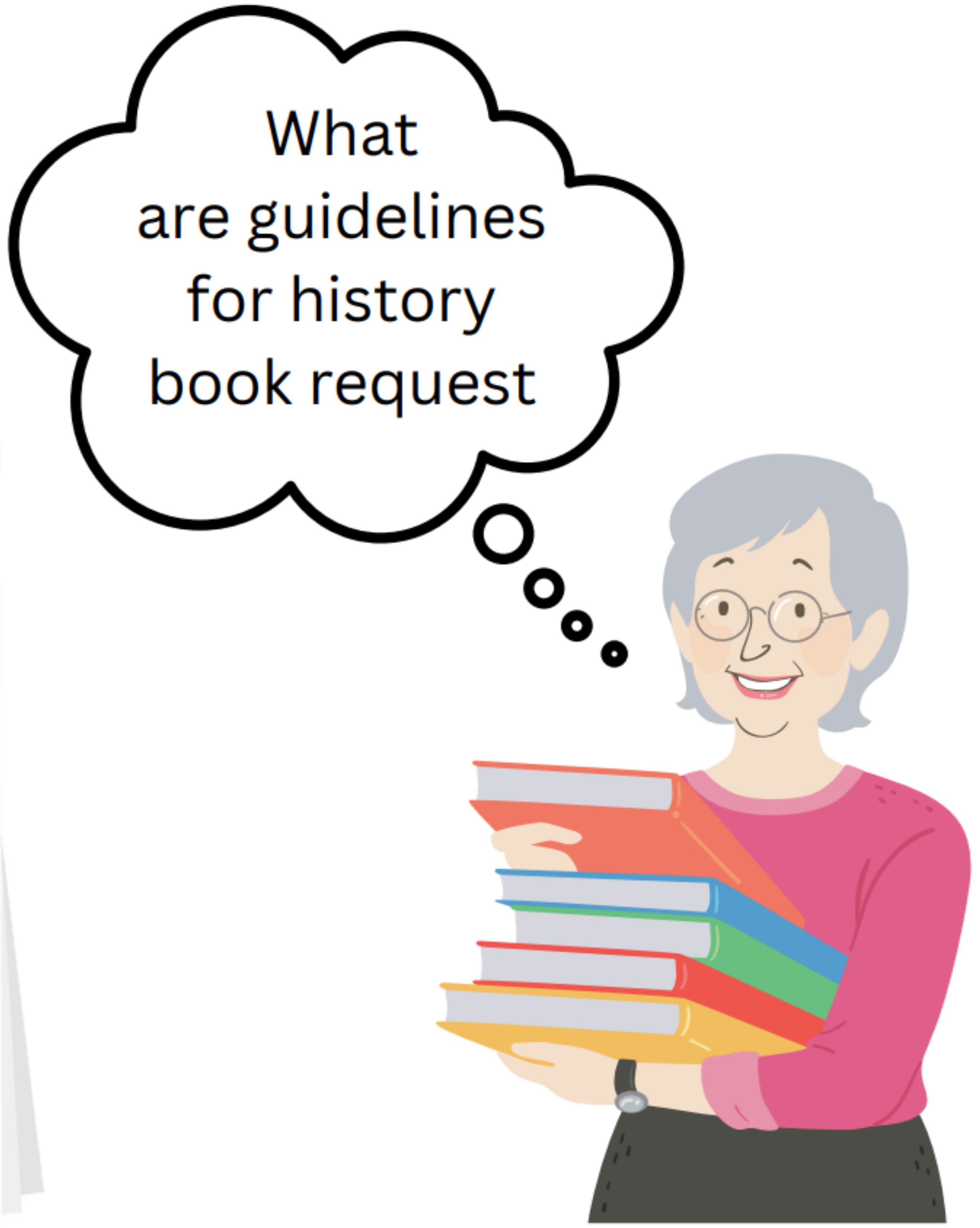
Lets draw some Parallels

Visitor/User

Librarian

Reference Books Section

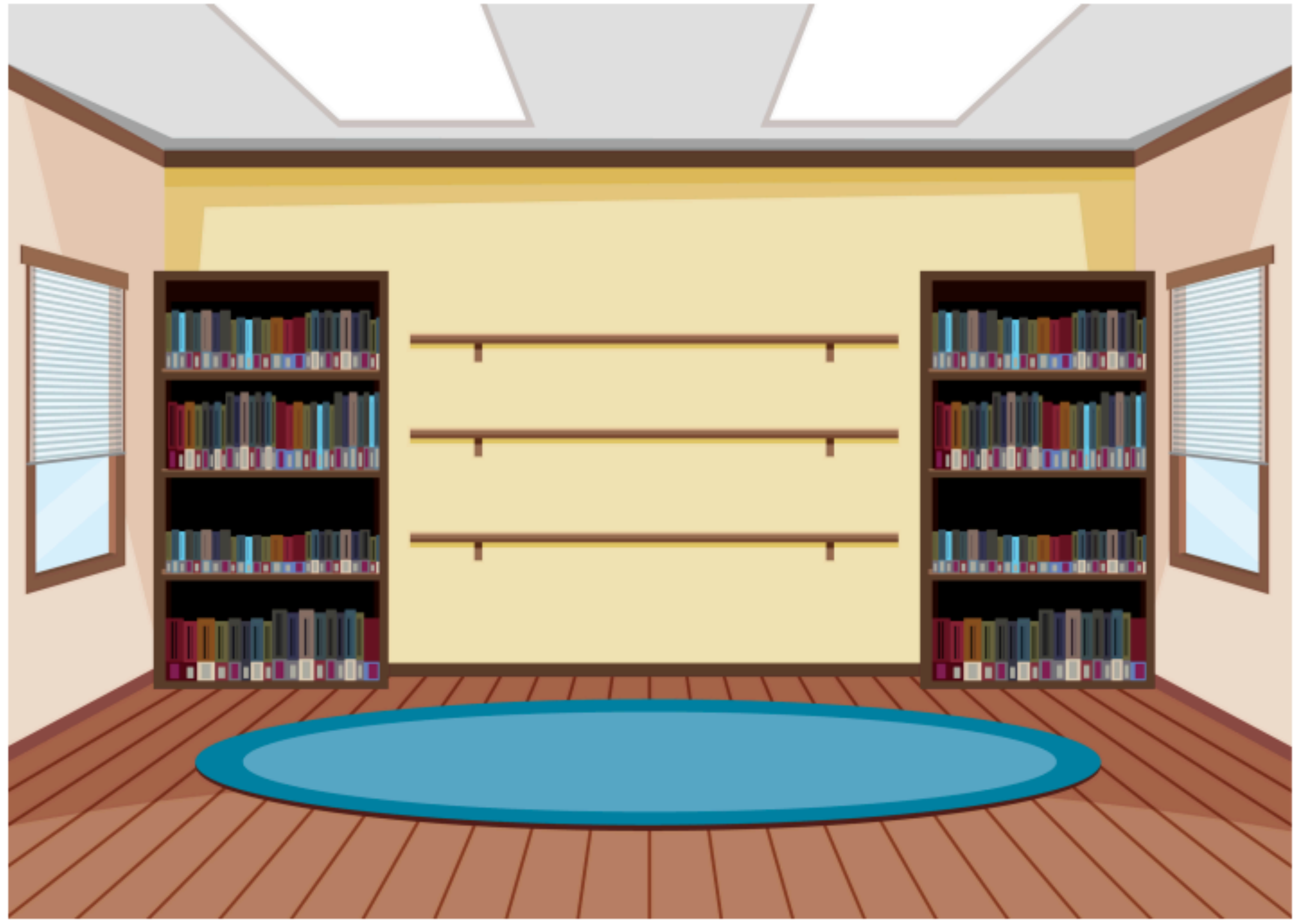
Lets draw some Parallels



Librarian

Provides access to the visitor/User as per guidelines

Lets draw some Parallels



Access completed

User is out of the reference section and continues normal access

Librarian waits for next request

Restricted Operations

- Bring hardware into the picture
    - Introduce a new processor mode
- **User mode**
    - Code is restricted in what it can do
    - Eg: no I/O request, Processor will raise an exception
- **Kernel mode**
    - Code can do whatever it likes to do
    - All privileged operations can be executed

Any challenges that you can think of?

Limited Direct Execution (LDE)

- Low level mechanism that separates the user space from kernel space
- Let the program directly run on the CPU
- Limits what process can do
- Offer privileged operations through well defined channels with the help of OS

At the end we need OS to be more than just a library!

How to move from User to Kernel?

- **System calls** - Kernel performs on behalf of user process
  - Key pieces of functionality exposed by the kernel
    - File system, process management, process communication, memory allocation, etc
  - Most OS provides few 100s of calls
  - Early unix - 20 calls
- Some privileged hardware instruction support is needed - Cannot use normal function call mechanism

System call works little differently

- Kernel does not trust the user stack - You don't want to jump to random addresses
  - Maintains a separate **kernel stack (kernel mode)**
- Kernel cannot rely on user provided address
  - Uses a table - Interrupt Descriptor table (boot time) - **Guidelines in our example**
  - IDT consists of addresses of different kernel functions to run on system calls or other events

TRAP Instruction

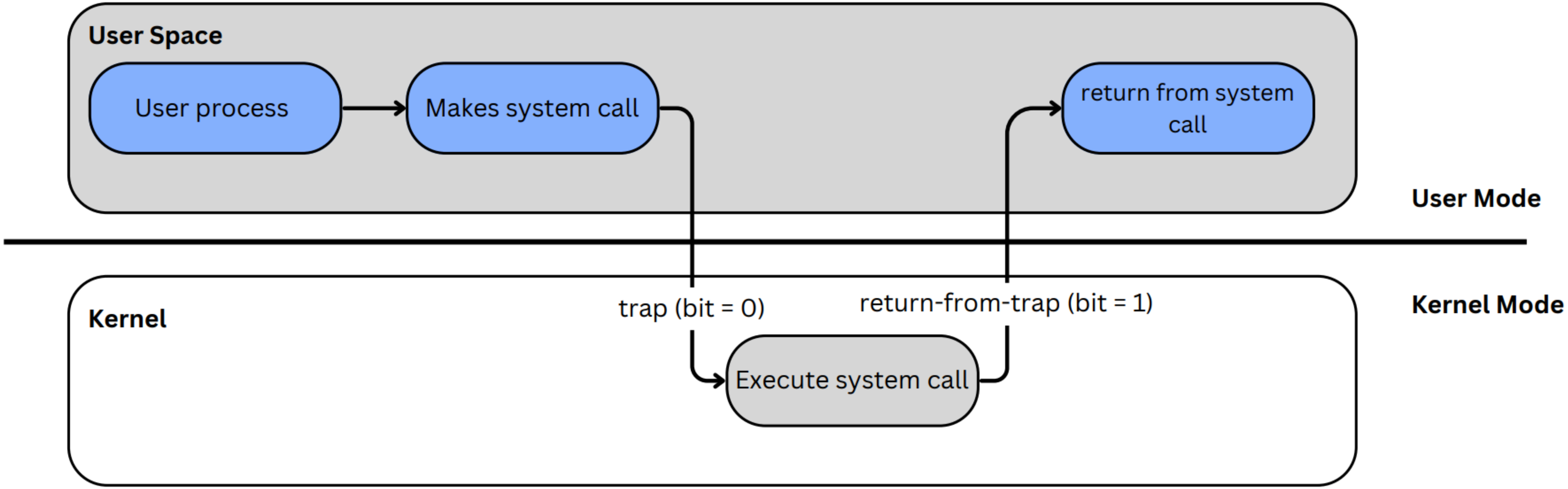
- Special kind of instruction to switch mode from user to kernel
- Allows system to perform what it wants
- When a system call is made, the trap instruction allows to jump into kernel
    - Raise the privilege mode to kernel mode
    - **Return-from-trap** instruction allows switch back to user mode
    - Return into the calling user program
- Normal routine is interrupted

More about TRAP instruction

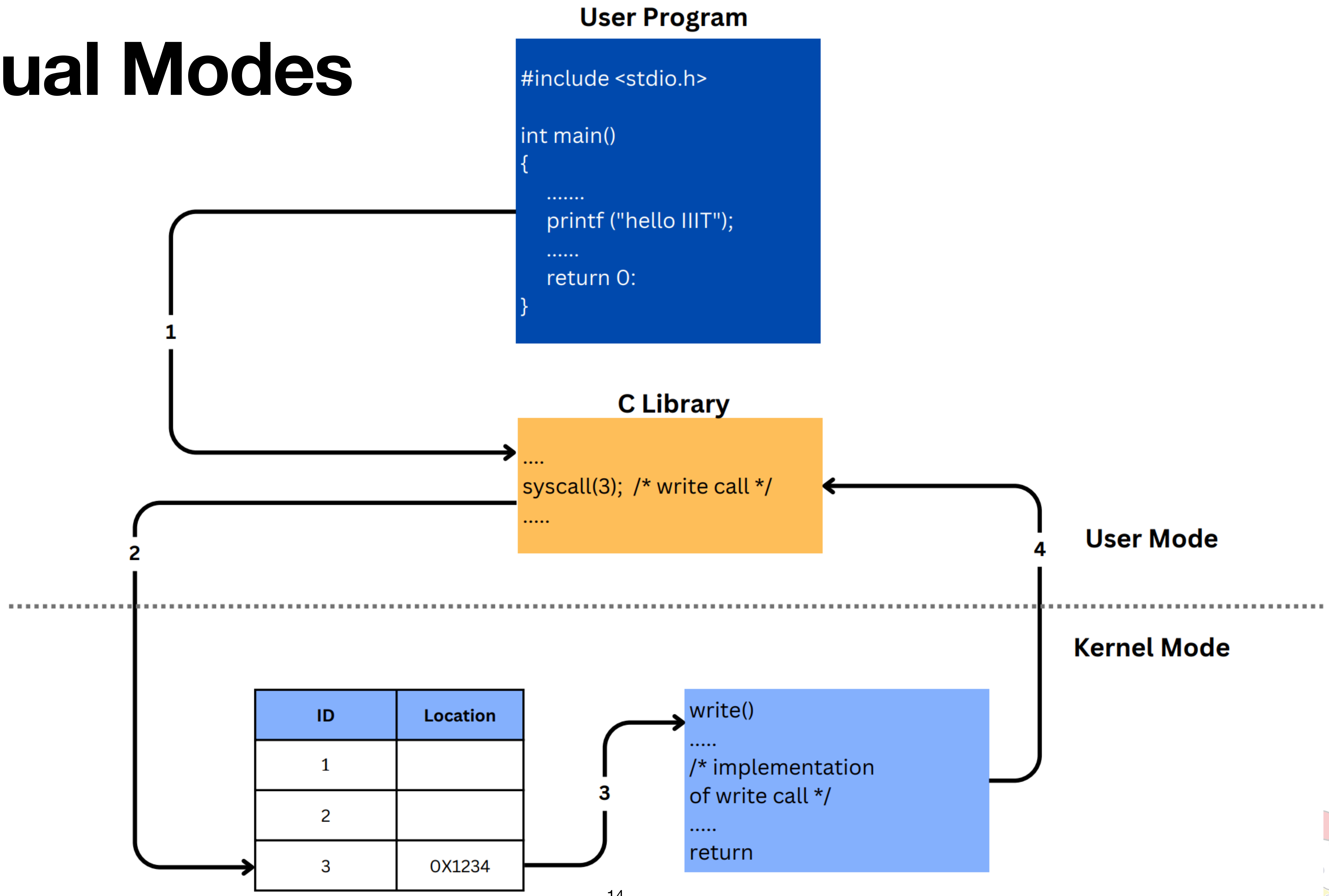
- During TRAP instruction execution
  - CPU to higher privilege level
  - Switch to Kernel Stack
  - Save context (old PC, registers) on Kernel Stack
  - Look up in IDT (Trap Table) and jump to trap handler function in OS code
  - Once in Kernel, privileged instructions can be performed
- Once done, OS calls a special **return-from-trap** instruction
- Returns into calling program, with back to **User mode**

The dual modes

User Mode and Kernel Mode



The Dual Modes



Adapted from: OS02-Limited Direct Execution, Dongkun Shin, SKKU

Interrupt and Trap

- **Interrupt**
  - Signal sent to the CPU due to unexpected event
  - I/O Interrupt, clock Interrupt, Console Interrupt
  - From either Software or Hardware interrupt
    - Hardware may trigger an interrupt by signalling to the CPU
- **Trap**
  - Software generated interrupt caused by
    - Exception: Error from running program (divide by Zero)
    - System call: Invoked by user program

LDE Protocol

OS @ boot (Kernel mode)	**Hardware**	
Initialize trap table	Remember address of.. Syscall handler..	
OS @ run (Kernel mode)	**Hardware**	**Program (User mode)**
Create entry for process list Allocate memory for program Load program into memory Setup user stack with arg Fill kernel stack with reg/PC		
	Restore regs from kernel stack Move to user mode Jump to main	
		Run main() .. System call **trap** into OS

LDE Protocol

OS @ boot (Kernel mode)	Hardware	Program (User mode)
	.... Save regs to kernel stack Move to kernel mode Jump to trap handler	
Handle trap Execute the system call Return-from-trap		
	Restore regs from kernel stack Move to user mode Jump to PC after trap	
		... Return from main() trap (via exit())
Free memory of process Remove process from process list		

Problem 2: How to Switch between Process?

Lets draw some parallels

Librarian does not have a control
when the person is inside the reference section
(only one reference section and a person is already inside)

More users/visitors have requested
to access the reference section

How can this situation be handled? - What can be the possibilities?

Cooperative Approach

Non-Preemptive

- Wait for system calls
- OS trusts the processes to behave reasonably (Give control back - **Yield()** call)
- Process transfer the control to the CPU by making a system call
- There can be misbehaving process (They may try to do something they shouldn't)
  - Divide by zero or attempting to access memory it shouldn't
  - Trap to OS -> OS will terminate the process
- Used in initial versions of Mac OS, Old Xerox alto system
- What if there is an infinite loop & process never terminates? - **Reboot**

Non-Cooperative Approach

Preemptive

- **OS takes control**
    - The only way in cooperative approach to take control is reboot
    - Without Hardware support, OS can't do much!
    - How can OS gain control?
- **Simple solution - Use interrupts**
    - Timer interrupt was invented many years ago
    - Every X milliseconds, raise an interrupt -> halt the process -> invoke interrupt handler -> OS regains control

Non-Cooperative Approach

Preemptive - Timer Interrupt

- During boot sequence, OS starts the timer
- The time raises an interrupt every “X” milliseconds
- The timer interrupt gives OS the ability to run again on CPU
- Two decisions are possible - Component called Scheduler comes into picture
    - Continue with current process after handling interrupt
    - Switch to a different process => OS executes **Context Switch**

Context Switch

- A low-level piece of assembly code
- **Save a few register values** from executing process registers to kernel stack
  - General purpose registers
  - Program counter
  - Kernel stack pointer
- Restore values for the next process
  - essentially return-from-trap will go to new process
- Switch to Kernel stack for the next process

LDE Protocol (Timer Interrupt)

OS @ boot (Kernel mode)	Hardware	
Initialise trap table	Remember address of.. Syscall handler.. Timer handler	
Start interrupt timer	Start timer Interrupt CPU every “X” milliseconds	
OS @ run (Kernel mode)	**Hardware**	**Program (User mode)**
		Process A ….
	Timer interrupt Save regs(A) to k-stack(A) Move to kernel mode Jump to trap handler	

LDE Protocol (Timer Interrupt)

OS @ boot (Kernel mode)	Hardware	Program (User mode)
Handle the trap Call switch() routine Save regs(A) to proc-struct(A) Restore regs(B) from proc-struct(B) Switch to k-stack(B) **Return-from-trap (into B)**		
	..... Restore regs(B) from k-stack(B) Move to user mode Jump to B's PC	
		Process B …

What if?

- During handling of one interrupt another interrupt occurs?
  - Disable interrupt during interrupt processing
  - Sophisticated locking mechanism to protect concurrent access to internal data structures

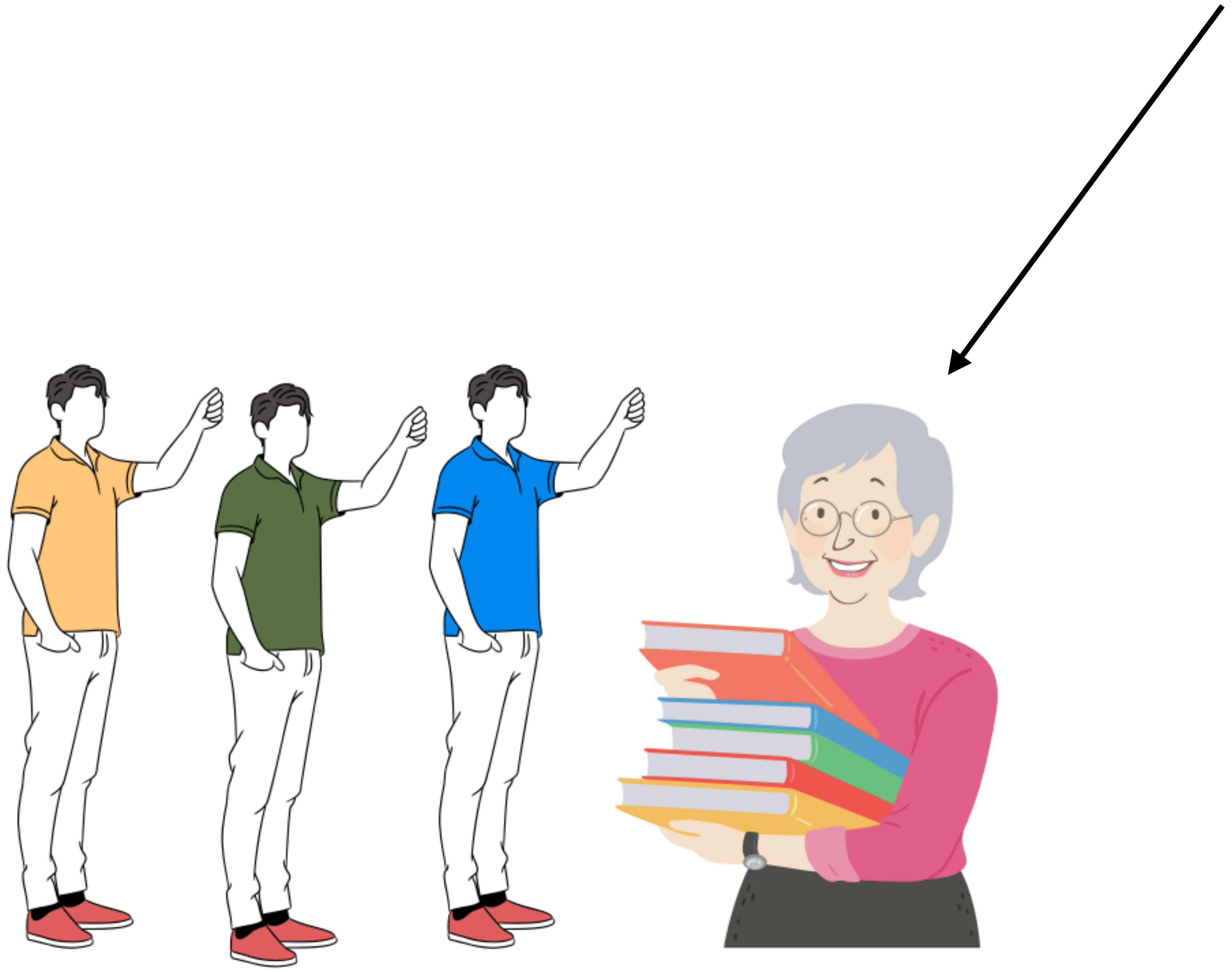
How to decide which process to run next?

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!!)

Scheduler



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 **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?

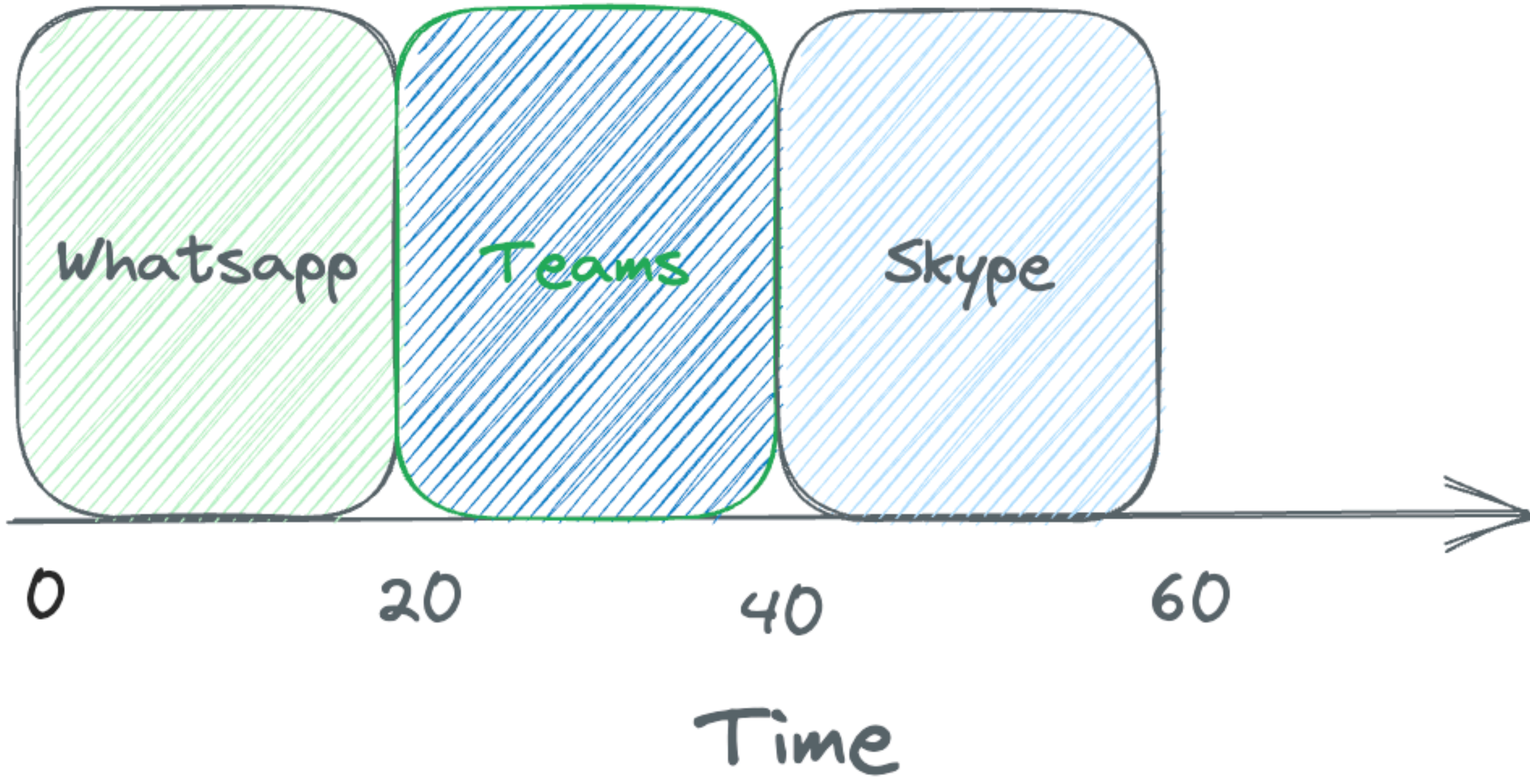
First Come First Serve Policy

- 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

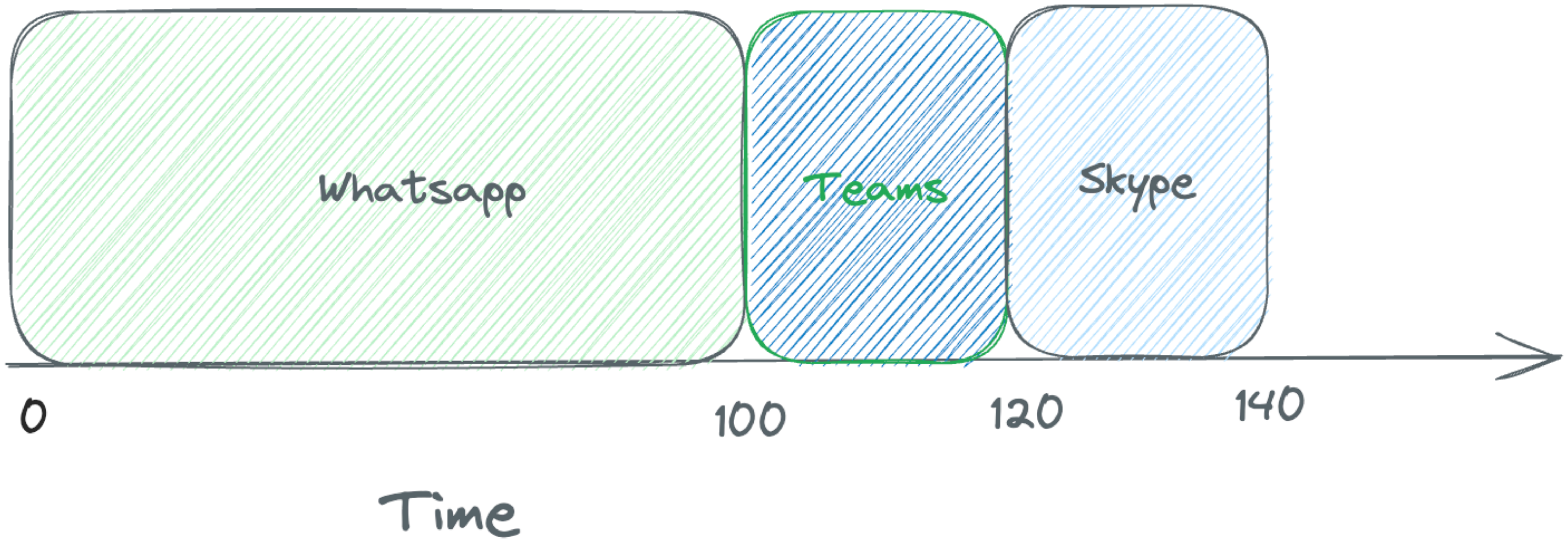


$$Avg(T_{turnaround}) = \frac{20 + 40 + 60}{3}$$

$$= 40$$

What if each job no longer runs for same time?

Relaxing assumption 2

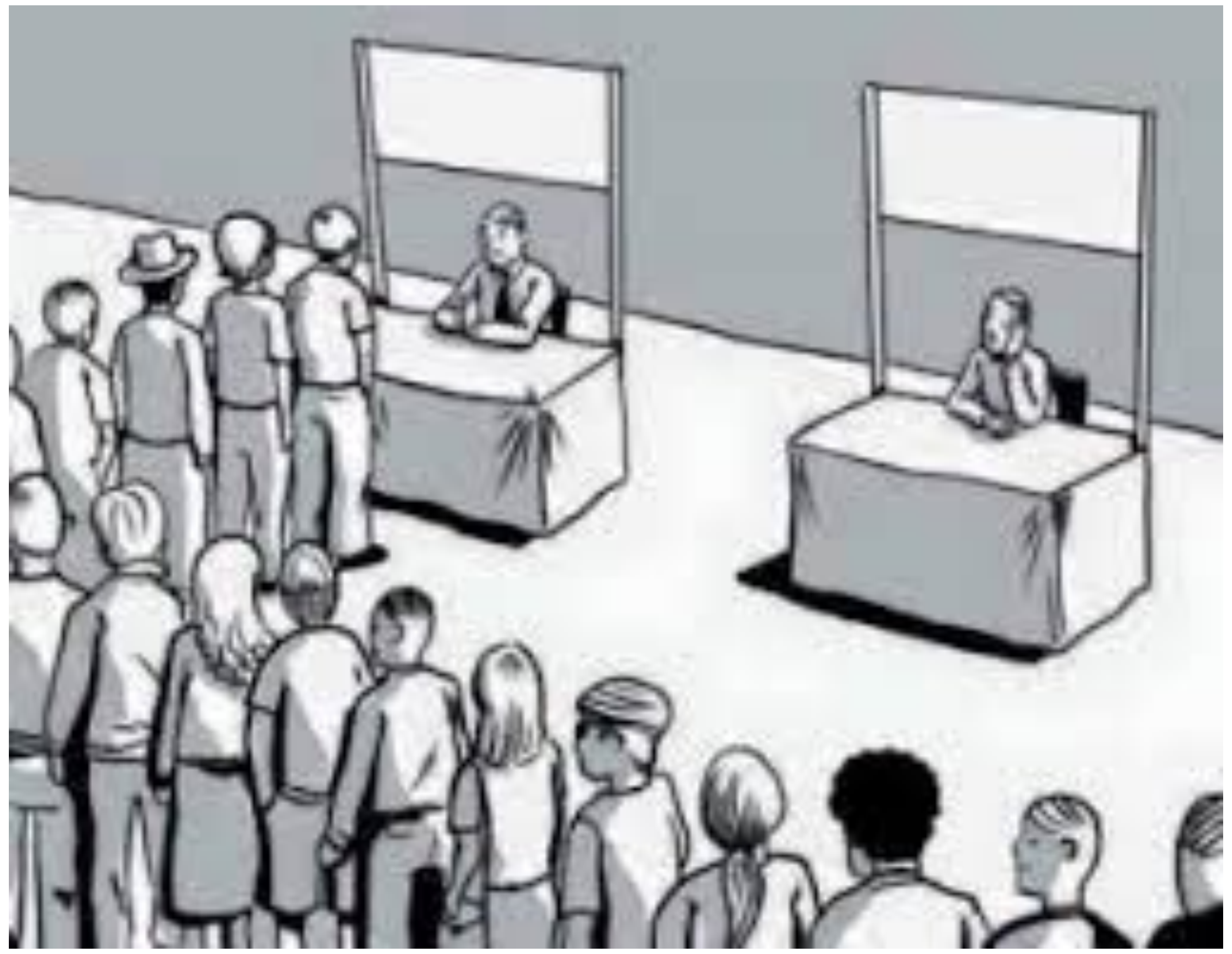


$$Avg(T_{turnaround}) = \frac{100 + 120 + 140}{3}$$

$$= 120$$

FCFS is not that great

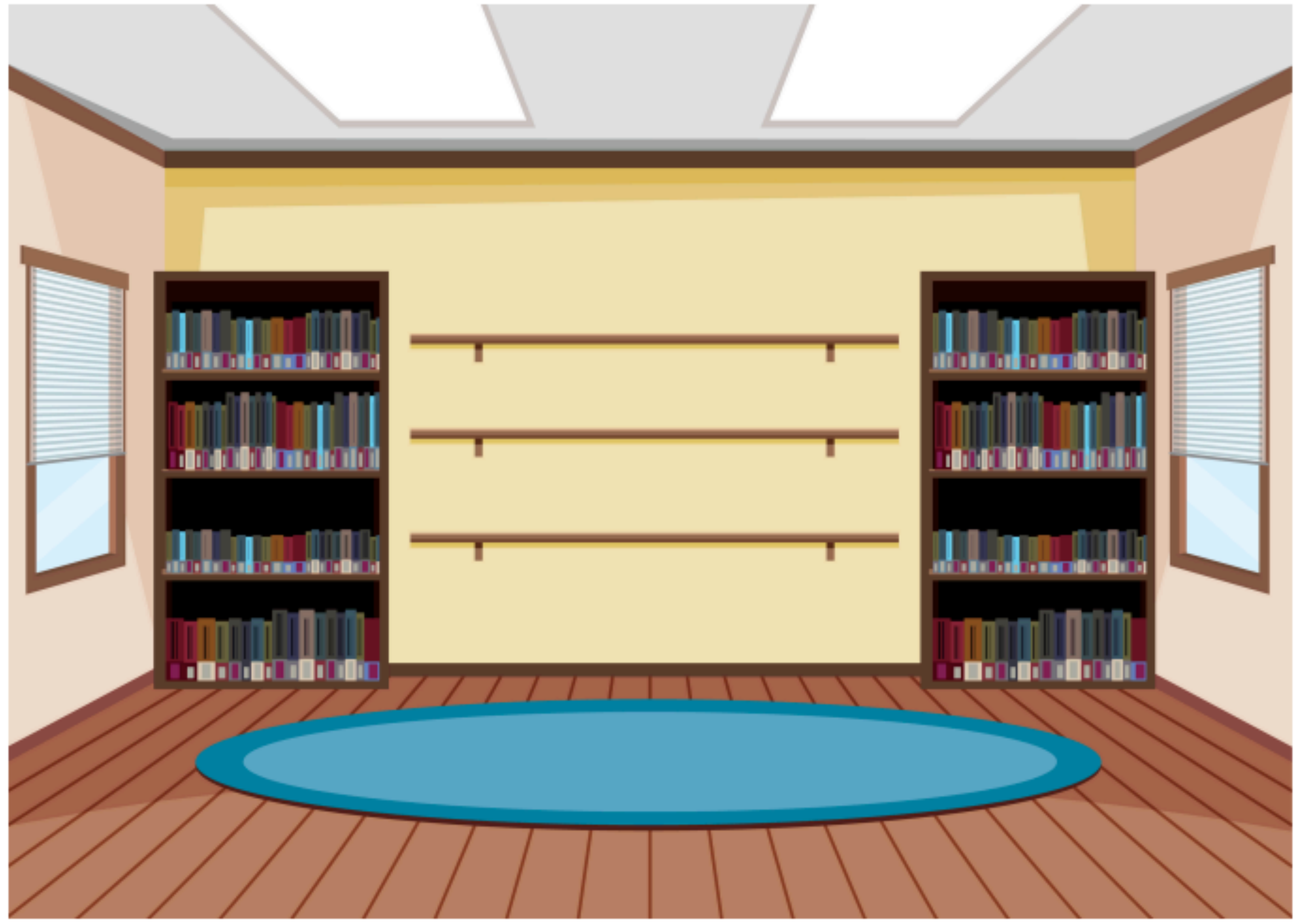
Convoy Effect



- 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

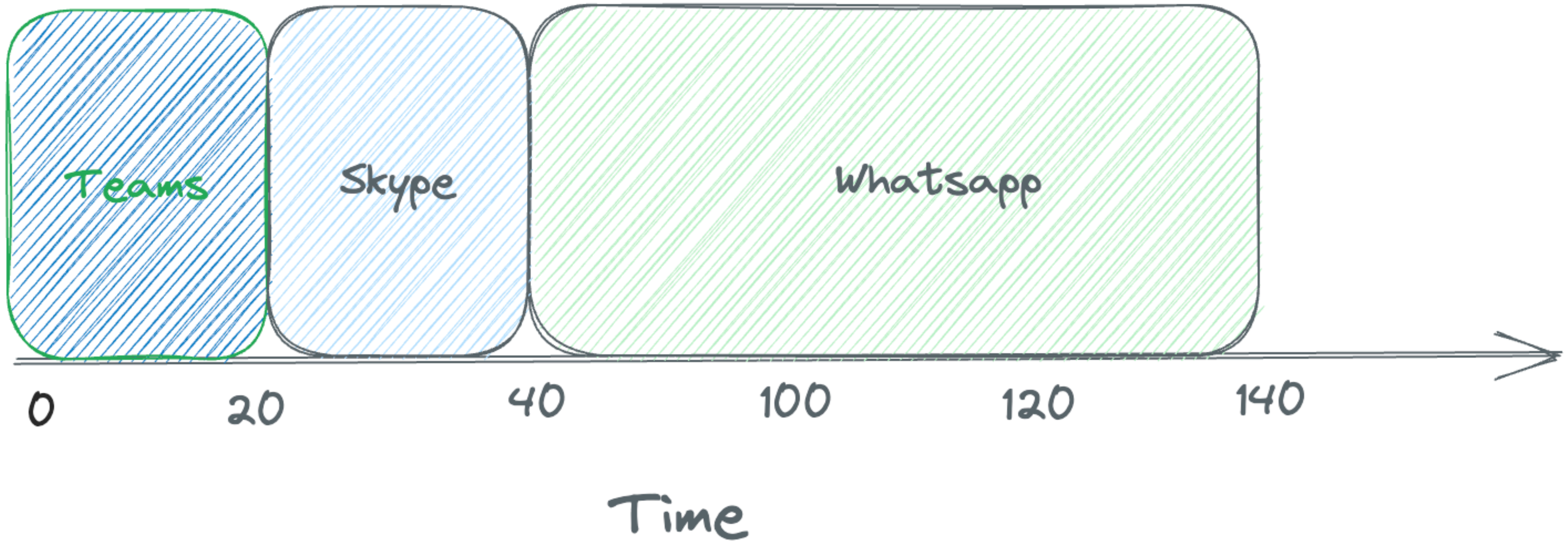
How to go about this?

Shortest Job First (SJF) Policy

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

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

$$= 66.3$$



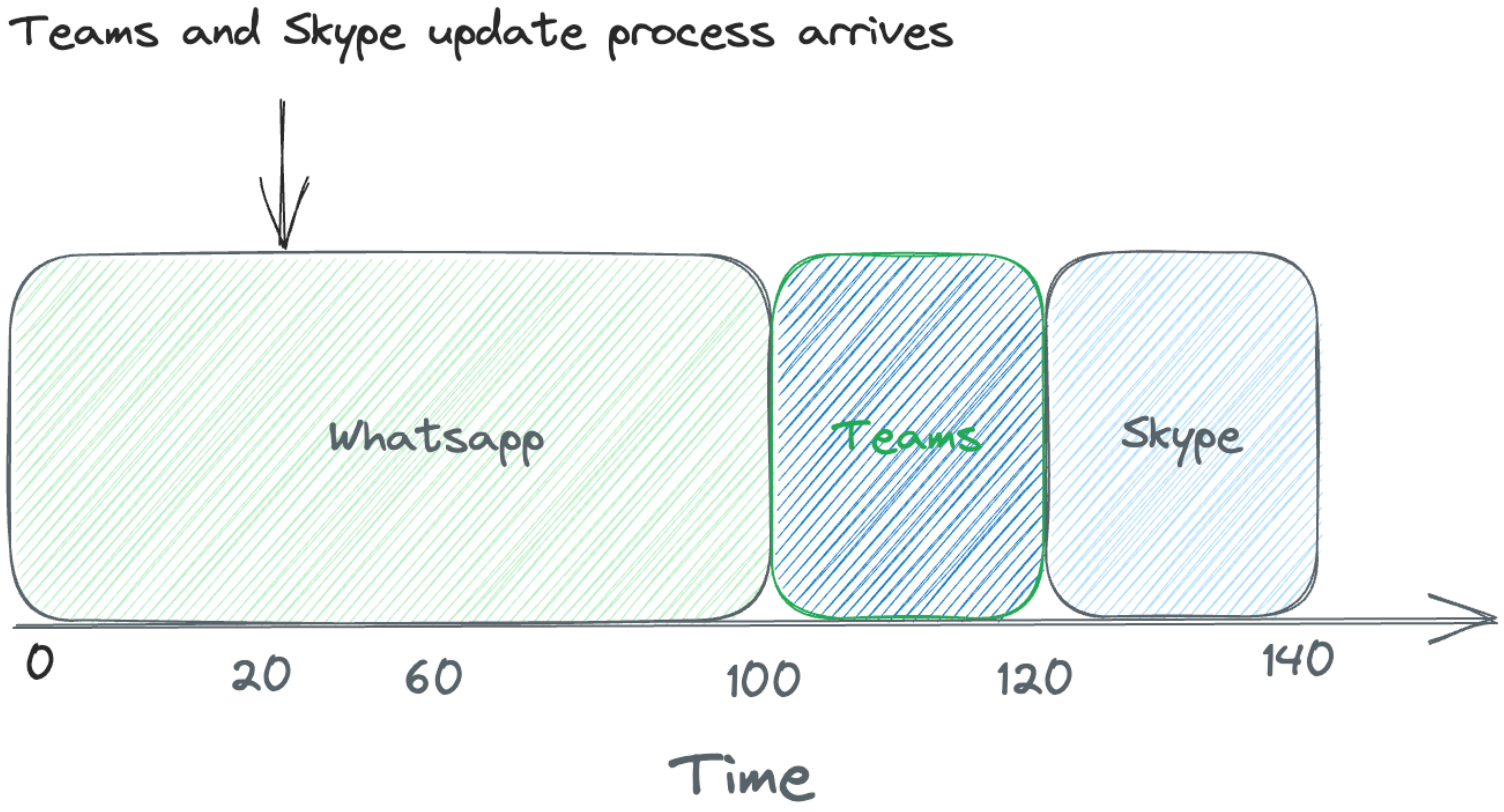
Is that a bit too unrealistic? - In reality jobs can arrive at any time

Shortest Job First (SJF) Policy

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

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

$$= 106.6$$

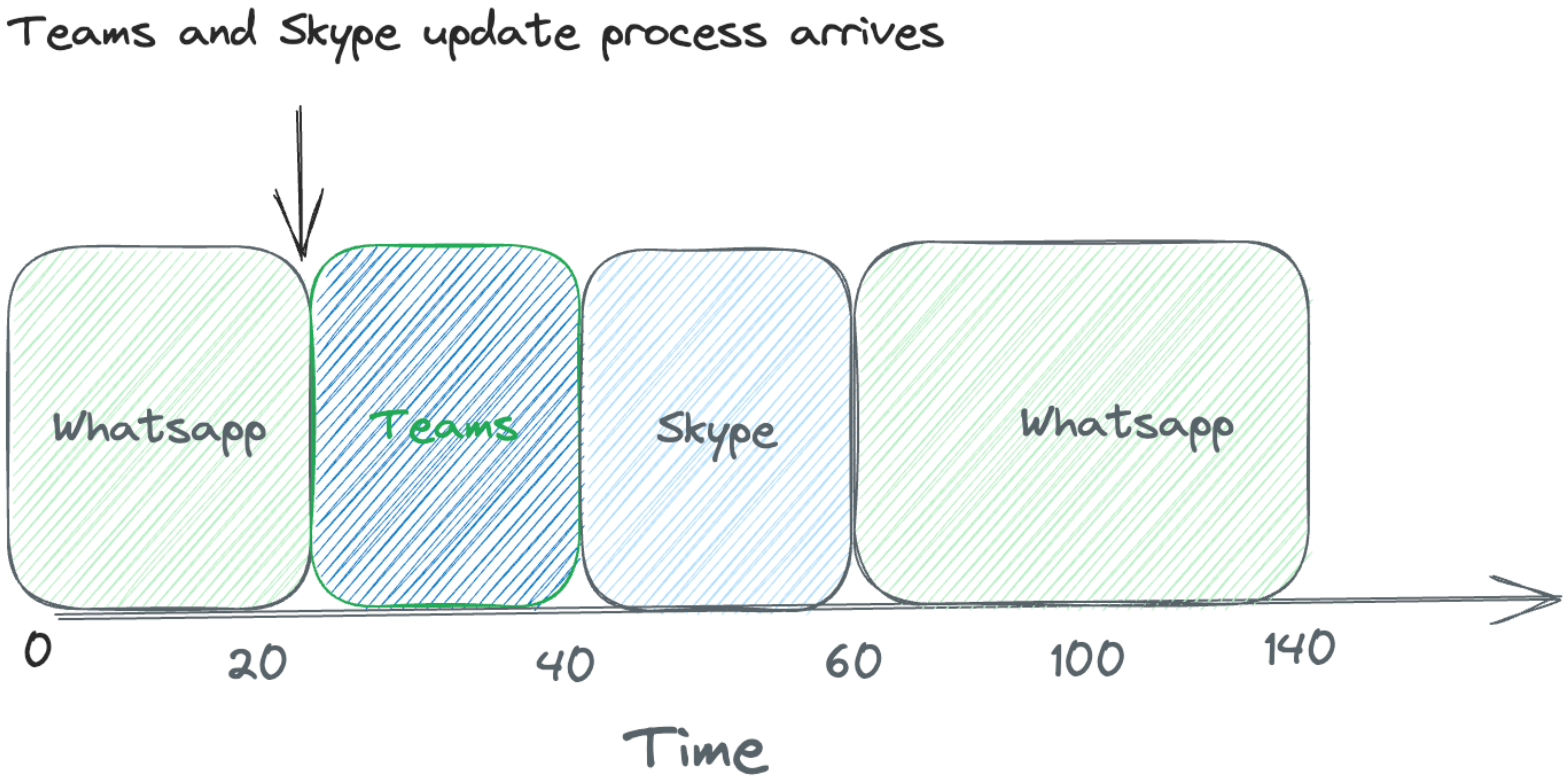


Even worst!! How to improve?

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?

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

SCAN ME

Thank you

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