CS3.301 Operating Systems and Networks

Process Virtualisation - API and Mechanisms

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

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

What features should the OS Provide?

Consider that we should be able to run multiple processes!

- **Create a process**
  - Double click and something just runs
- **Destroy a process**
  - Force quit, task manager -> end process
- **Wait**
  - Wait before running
- **Suspend**
  - Keep the process in pause and resume (eg: Debugging an application!)
- **Status**
  - Can we get some status of the process (task manager, system monitor, top)

How to make it happen? - Heard of APIs?

- Application Programming Interface - What’s that?
  - How does a travel website get information about different flights and allows booking?
  - What about payment services?
- API allows different programs/applications to communicate with each other
- Provides a software interface for accomplishment
- Comes with detailed documentation

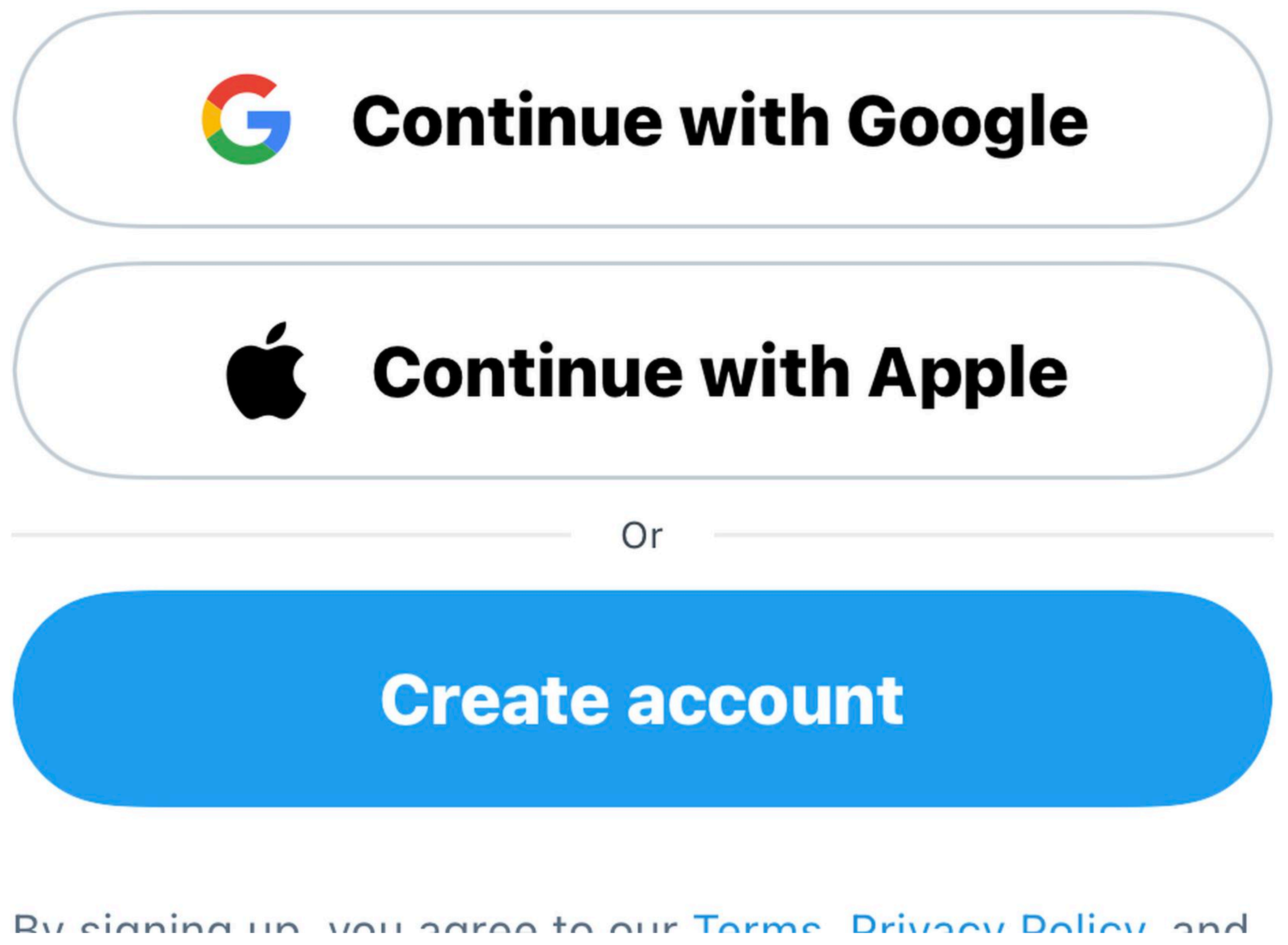
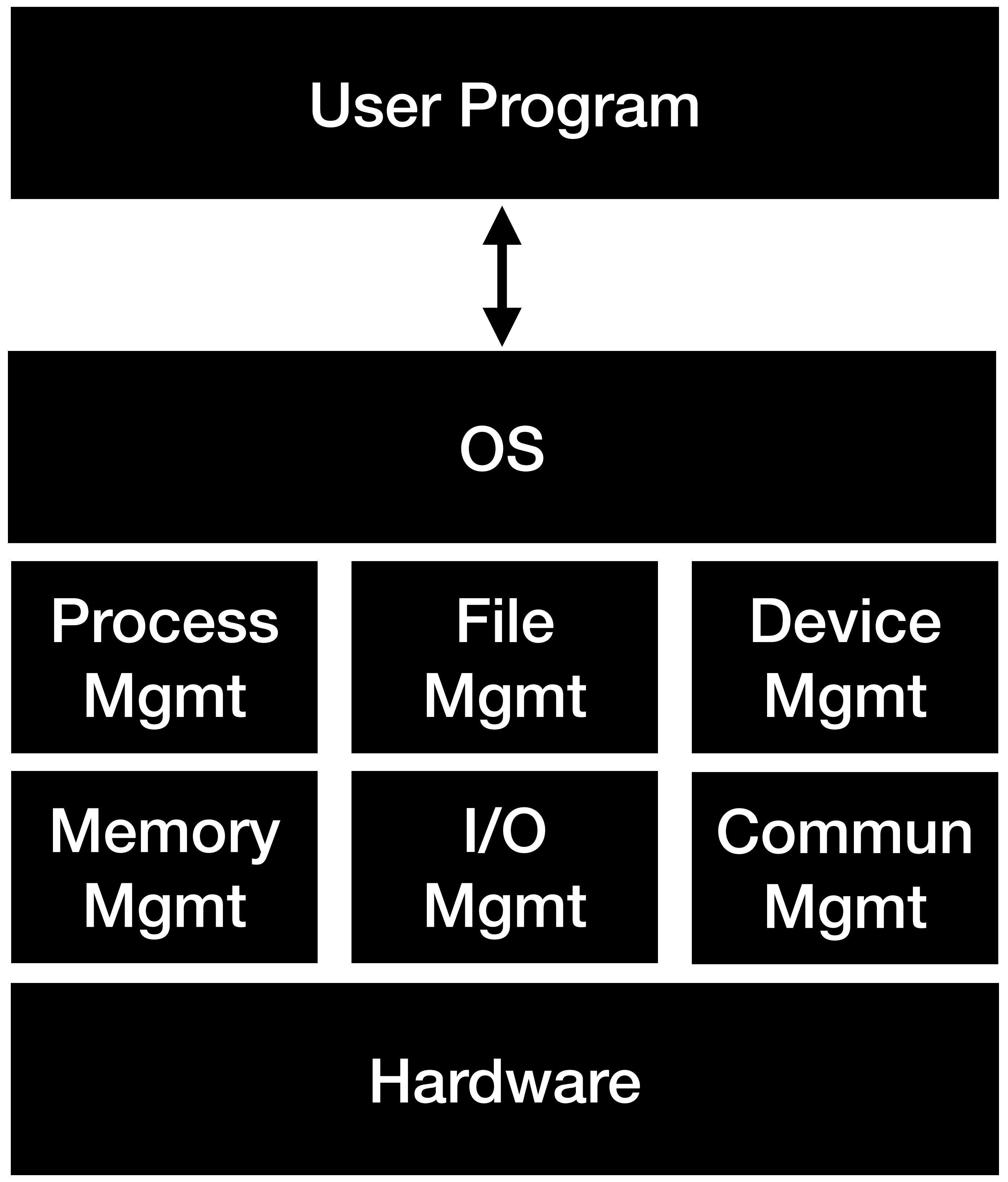


Image source: verge

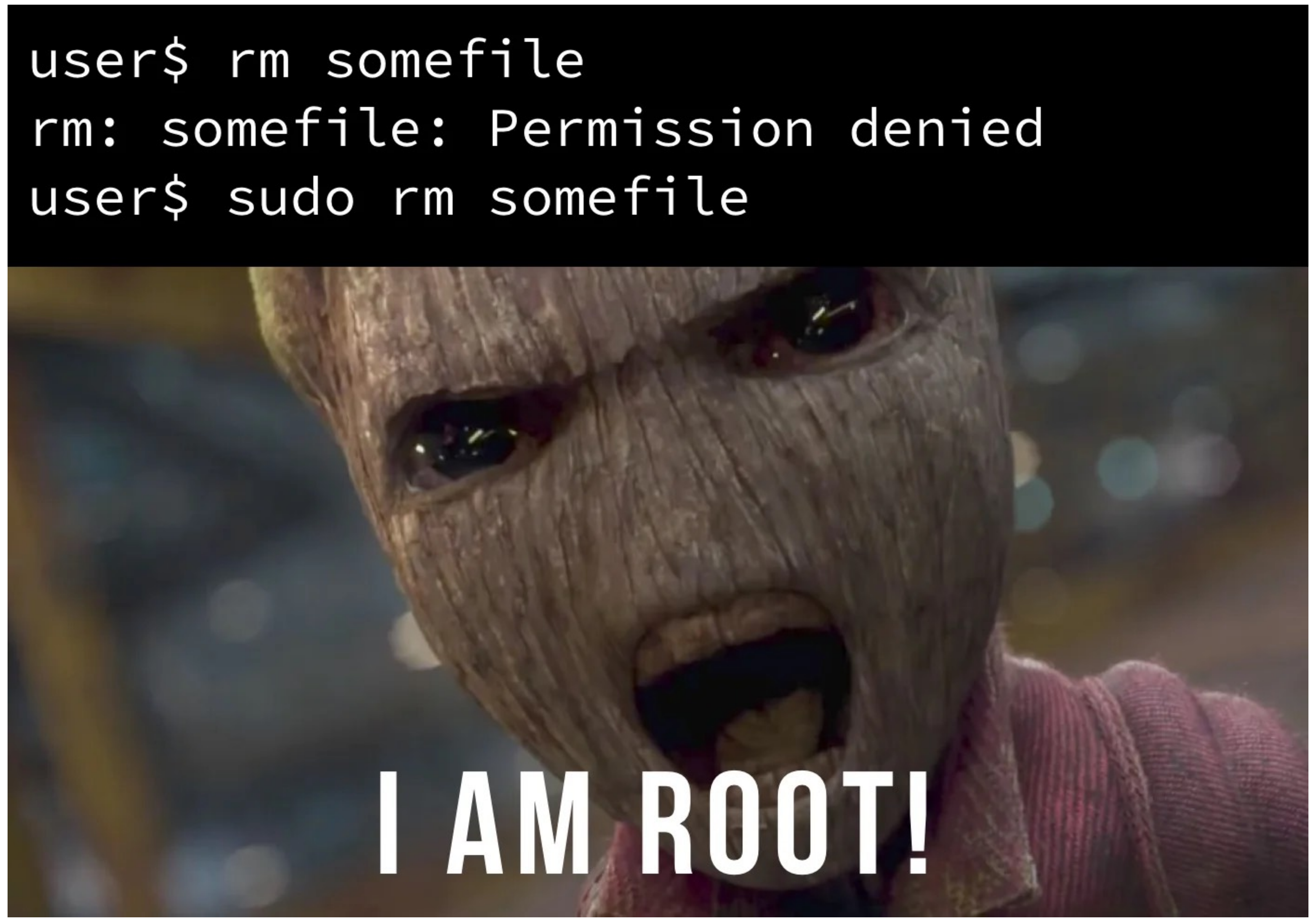
Does OS Provide API? - System Calls!

- Way for user program to interact with the OS
- OS provides some functions that can be leveraged by user programs
- Available in the form of “System calls”
  - Function call into OS code that runs at a higher privilege level
  - Think about access to hardware
- What if user wants to execute a process?



But you need Privileges!

- What if a user gives a instruction to delete all files?
  - Should all the instructions be considered with equal priority?
  - When does the role of OS come in to the main picture?
    - Think about reading a file or writing a file - How to achieve it in C?
    - What if you just wanted to multiply two numbers?
    - What about the command to get list of available directories?
- Two modes of execution - **User mode** and **Kernel mode**

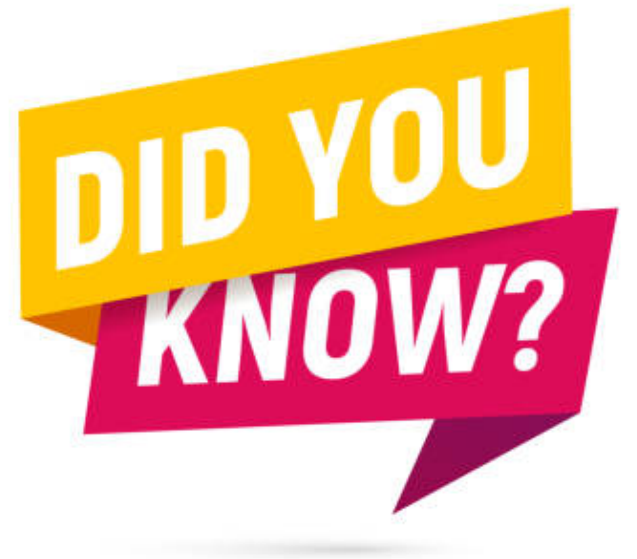


```
user$ rm somefile
rm: somefile: Permission denied
user$ sudo rm somefile
```

Source: reddit

For Each OS = Rewrite Programs?

- POSIX API (Portable Operating Systems Interface)
  - Standard set of System calls that an OS must implement
  - Most modern OS's are POSIX compliant
  - Ensures portability
- Programming language libraries abstract systems calls
  - printf() in C internally invokes write system call
  - User programs usually do not worry about system calls



Some System Calls

File Management	Process Management	Communication	Protection
fd =open(file,..)	fork()	Pipe()	chmod()
close(fd)	wait()	Shmget()	Unmask()
write(fd, …)	exec()	Mmap()	chown()
…	…	…	…

System Calls for Process (Unix)

System Call	Supports
fork()	Creates a new child process
exec()	Makes a process execute (runs an executable)
wait()	Causes a parent to block until child terminates
exit()	Terminates a process

- Many variants of the above calls exist
- **init** process is the ancestor of all processes

The Fork System Call

- A new process is created
    - Parent process image copy is made
- The new process is added to the list of processes and scheduled
- Parent and child start execution just after fork (with different return values)
- Parent and child execute and modify memory independently

The Wait API

- ***Wait()*** call blocks in parent until child terminates (options like ***waitpid()*** exists)
- Wait() also collects exit status of the terminated child process
  - Provides some visibility to the parent process
- Without wait, if process terminates - **Zombie process**
  - Exit status not collected by the parent
- Wait allows OS to reclaim the resources of the child - Prevent zombies
- What if Parent terminates before the child? - **Think!**

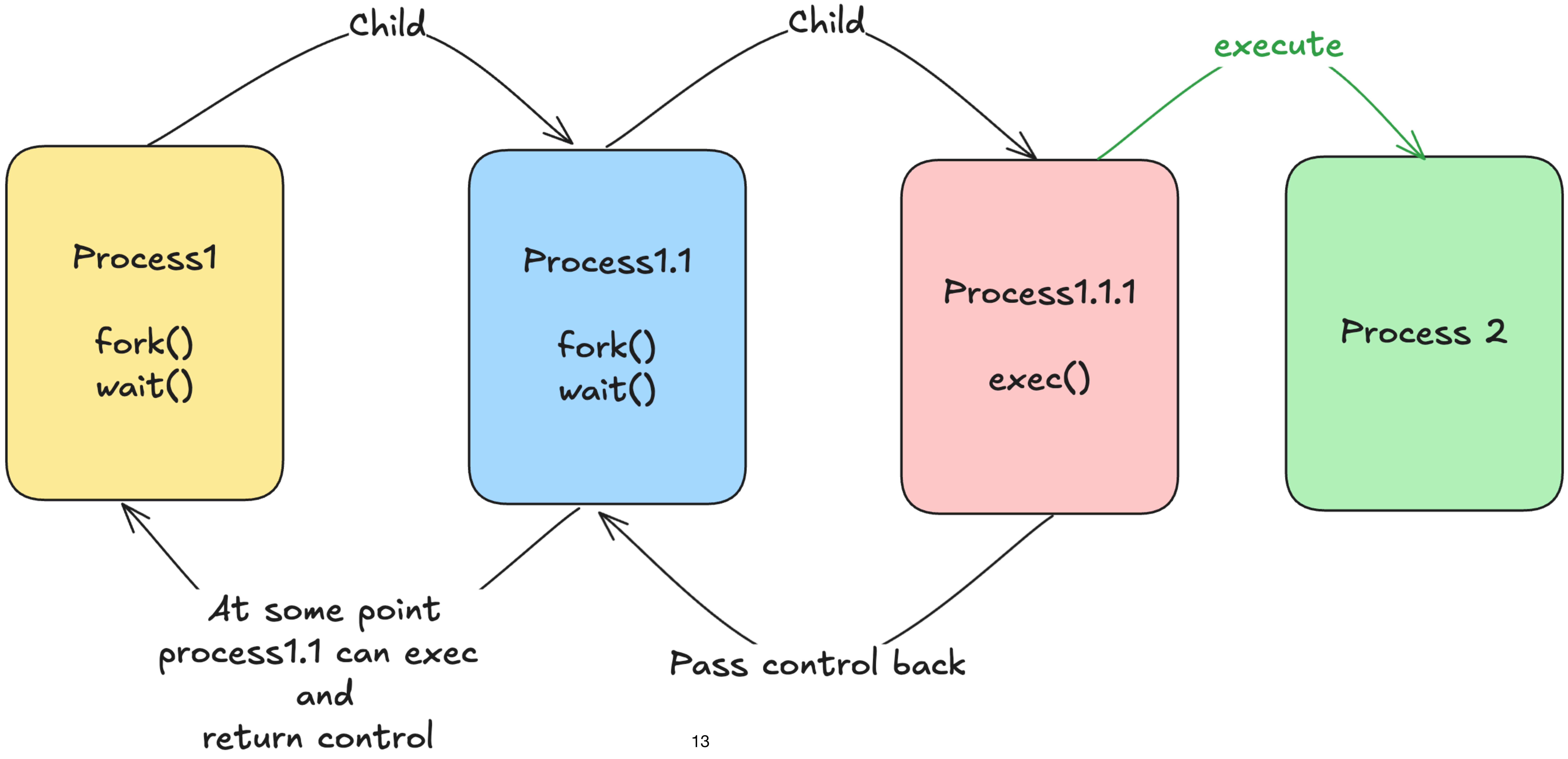
Remember: Init process, adopts orphans and reaps them

The Exec API

- When we perform a fork(), the parent and child execute the same code
    - Do you see some problem there?

- exec() comes to the rescue
    - Load a different executable to the memory
    - **Essence:** Child can run a different program from parent
    - The process ID of the process will remain the same
- In some variants of exec(), command lines to the executables can be passed!

Illustrative Flow



How does the Shell work? - Ever thought?

- Init process is started upon hardware initialisation
- The init process spawns a shell like bash
- Shell does the following
  - Read user command
  - Forks a child and exec the command
  - Wait for it to finish -> next command

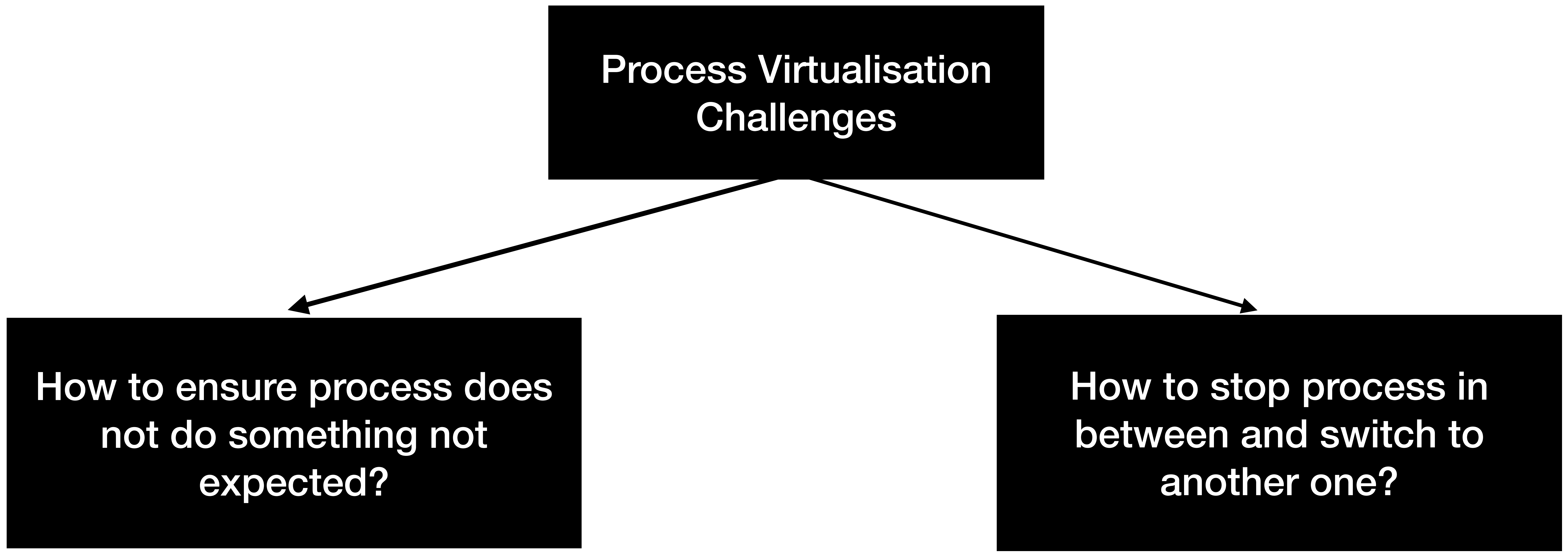
Can you think how this works?

- **> wc process_sample3.c > output.txt**
- Shell will fork a child
  - Rewires its standard output to text file (output.txt)
  - Calls exec on the child (wc process_sample.c)
  - The output will be redirected to output.txt
- Have you seen Unix pipes "|"
  - Output of one goes as input to the other

Note: fork(), exec() and wait() are required

The Big Question - How to run multiple Processes?

Two Major Problems to be Solved


Process Virtualisation Challenges
How to ensure process does not do something not expected?
How to stop process in between and switch to another one?

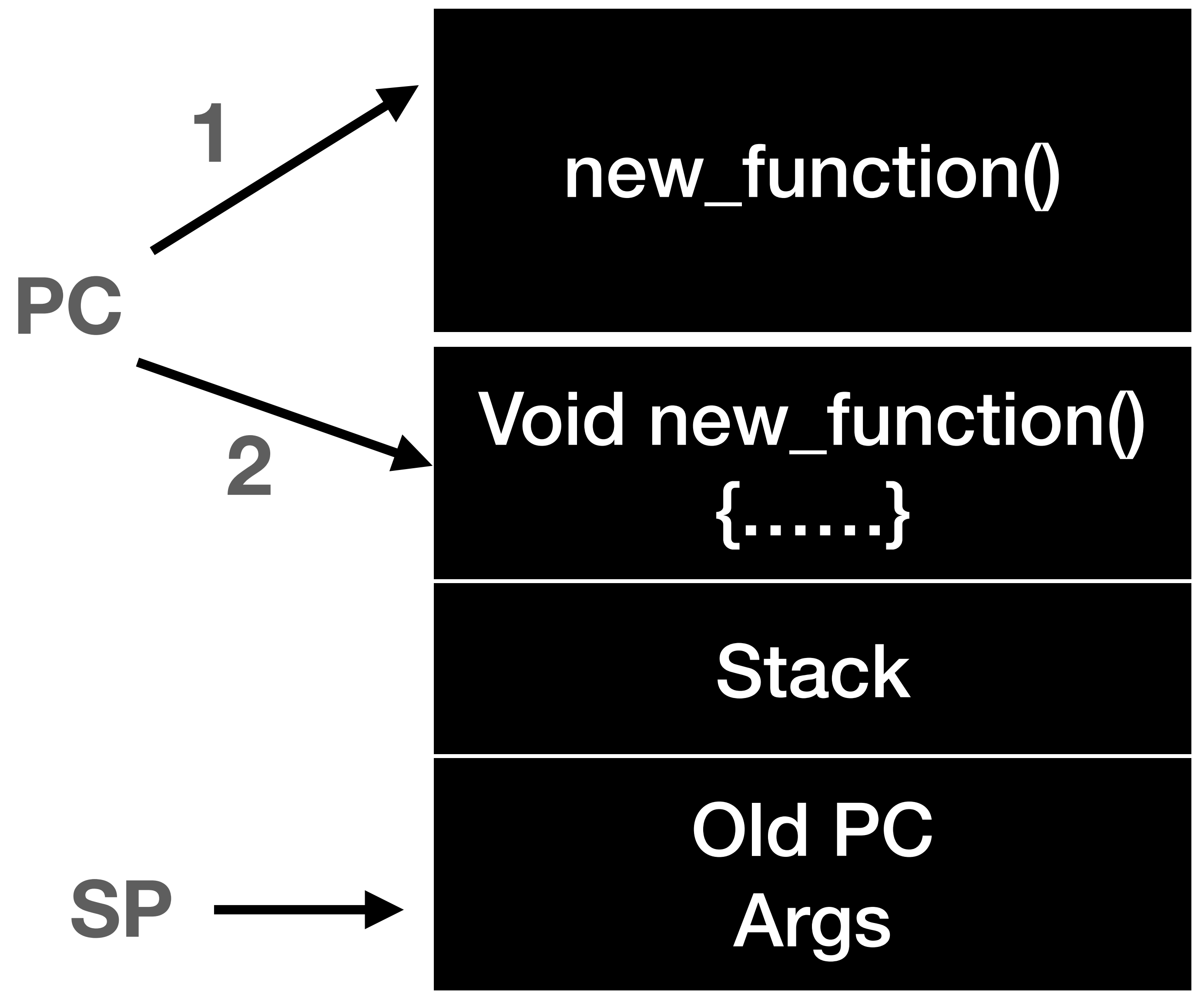
What if we allow process to do whatever it wants?

How can multiple processes run?

- **Hardware Support**
  - Have some low level mechanisms to switch process
  - What are the challenges?
    - Performance Overhead?
- **Software support**
  - Have some policies which decides what needs to be executed
  - What are some of the challenges?
    - Control overhead?

Normal Function call

- Function call translates to a jump instruction
  - One instruction to another instruction
- A new stack frame is pushed to the stack, Stack pointer is updated
- Old value of program counter (return value) pushed to stack and PC is updated
- Stack frame contains return value, function arguments, etc,



Is this enough?

OS	Program
1. Create an entry in process list 2. Allocate memory for the program 3. Load program into memory 4. Setup stack with argc/argv 5. Clear registers 6. Execute call main()	
	7. Run main () 8. Execute return from main()
9. Free memory of process 10. Remove process from process list	

What if?

- The process wants to perform operations such as:
  - Issuing I/O request to disk
  - Access to memory or other system resources
- Can we let the process do whatever it wants?

Idea: Can we think of limiting the access of a process?

Challenge 1: Prevent Unintentional behaviour

Limit Direct Execution

Only Kernel has access

THIS PROPERTY IS CLOSED TO THE PUBLIC
NO ENTRY WITHOUT PERMISSION
NO PHOTOGRAPHY
NO VIDEO RECORDING
RESTRICTED AREA
AUTHORISED EMPLOYEES ONLY
DELHI P

https://tribuneindia.com

User program can go until this point

SCAN ME

Thank you

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