CS3.301 Operating Systems and Networks

Introduction and Course Overview

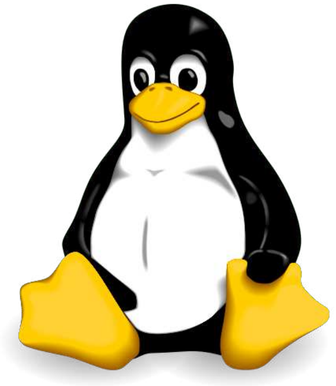
Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

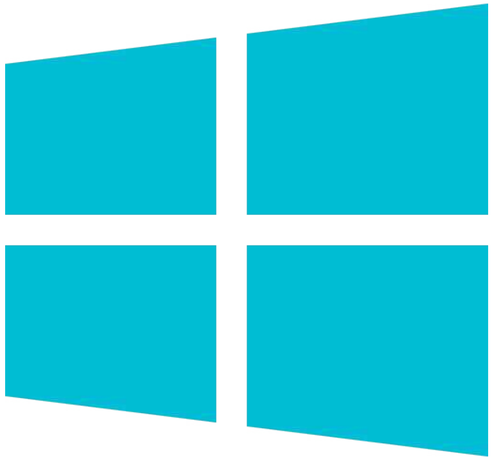
Software Engineering Research Centre

Which OS do you use?



Linux

ChromeOS



Windows

OSX

Wait! what is an OS anyway?

operating system

computing

Print | Cite | Share | Feedback

Also known as: OS

Written by David Hemmendinger
Fact-checked by The Editors of Encyclopaedia Britannica
Last Updated: Article History

operating system (OS), program that manages a computer's resources, especially the allocation of those resources among other programs. Typical resources include

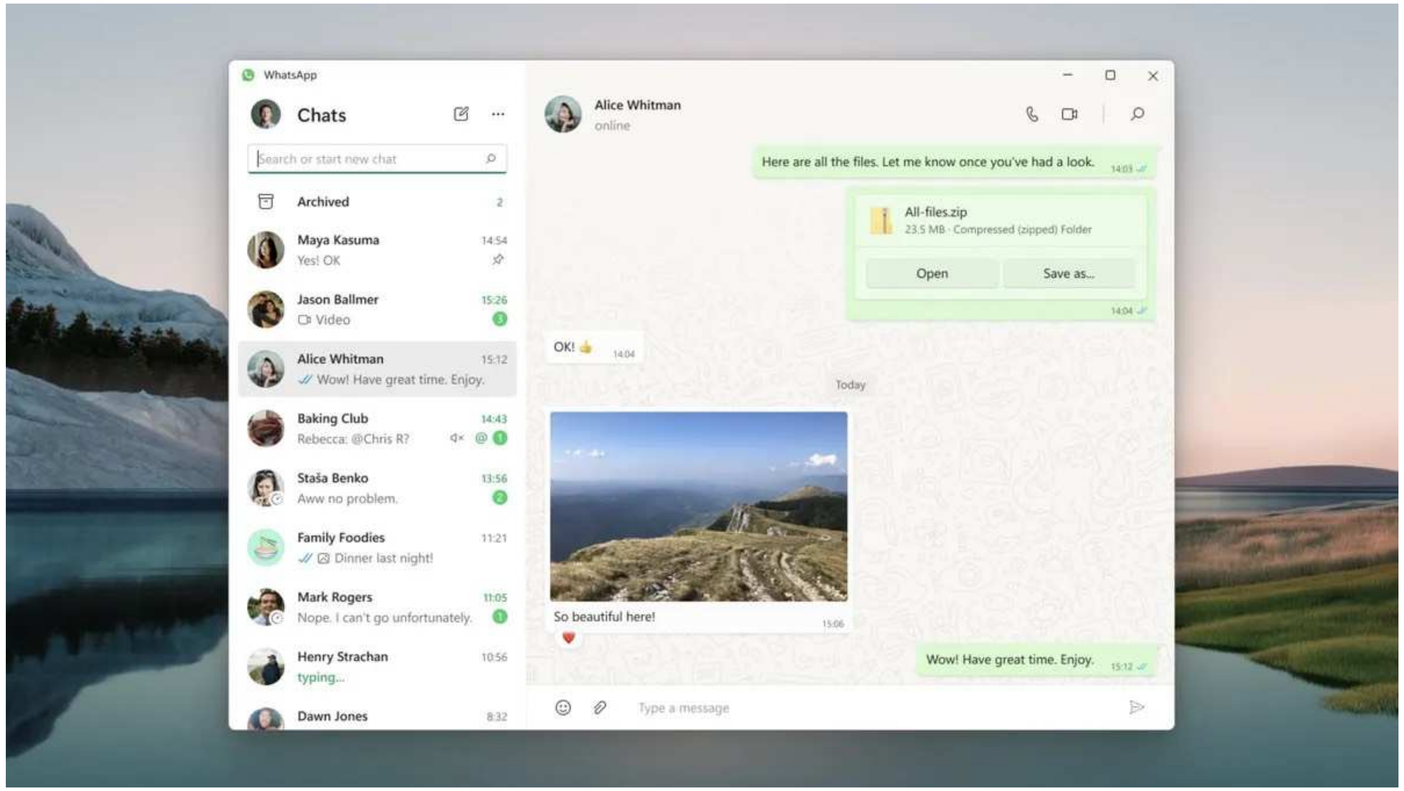
What is an operating system?

An operating system (OS) is a software that acts as an intermediary between computer hardware and the user applications. It manages the computer's hardware resources, such as the CPU, memory, disk storage, and peripheral devices like printers and scanners. The OS enables user programs to execute without needing to worry about the specifics of the hardware.

ChatGPT4 Version!

Caution: Use tools but be aware of the uncertainties they bring in!

Develop Whatsapp like Messaging System



- Users authentication
- Send and receive chats
- Send and receive media contents
- Make audio, video calls
- Record audio, video
- …..

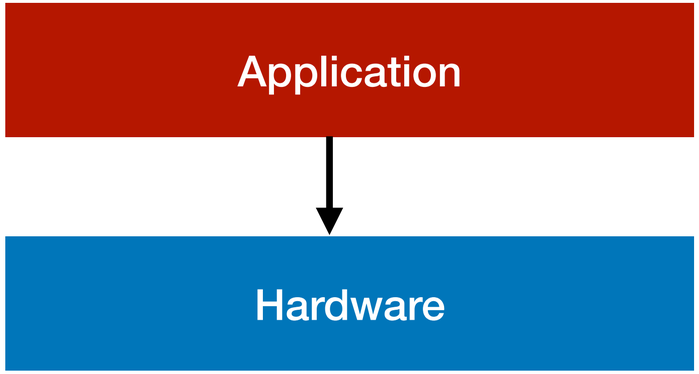
Source: cnet.com

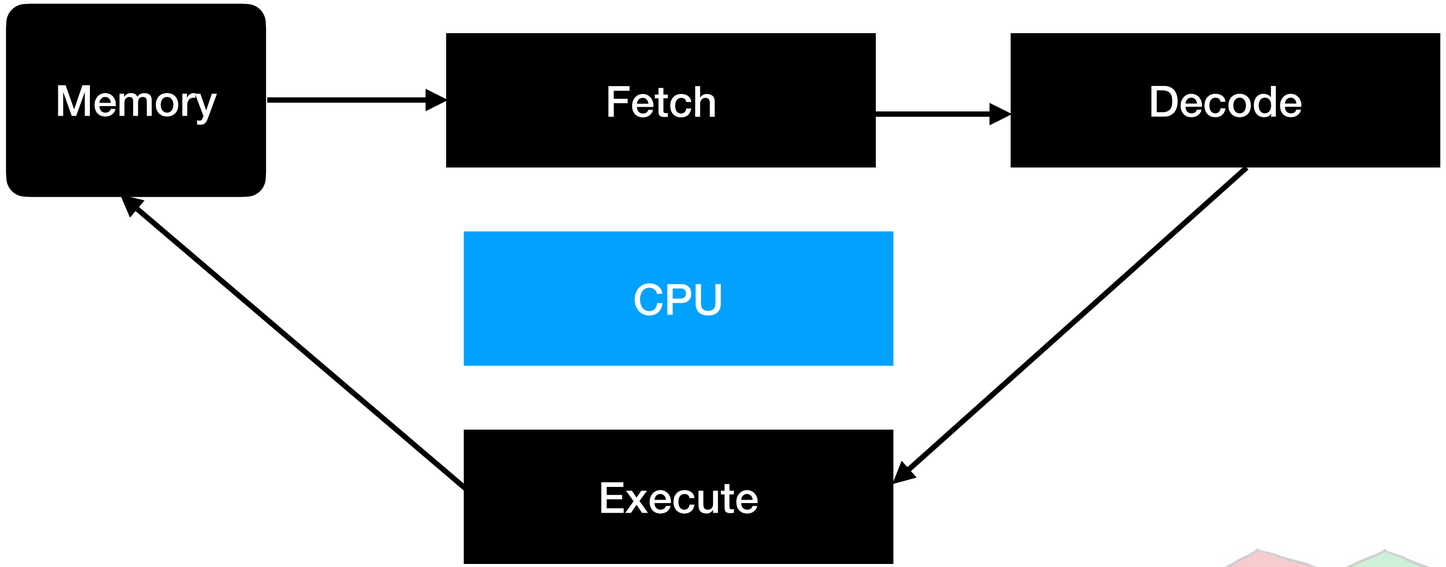
Any Program for that Matter!

- Once your app is build - Compiler comes into action (“c” -> “a.out”) to create the executable (“.dmg”, “.elf”, “.exe”)
- Executable contains: Instructions + data
- Instructions are run by the CPU
- CPU internally consists of registers
  - Program counter or PC
  - Operands of instructions, memory addresses

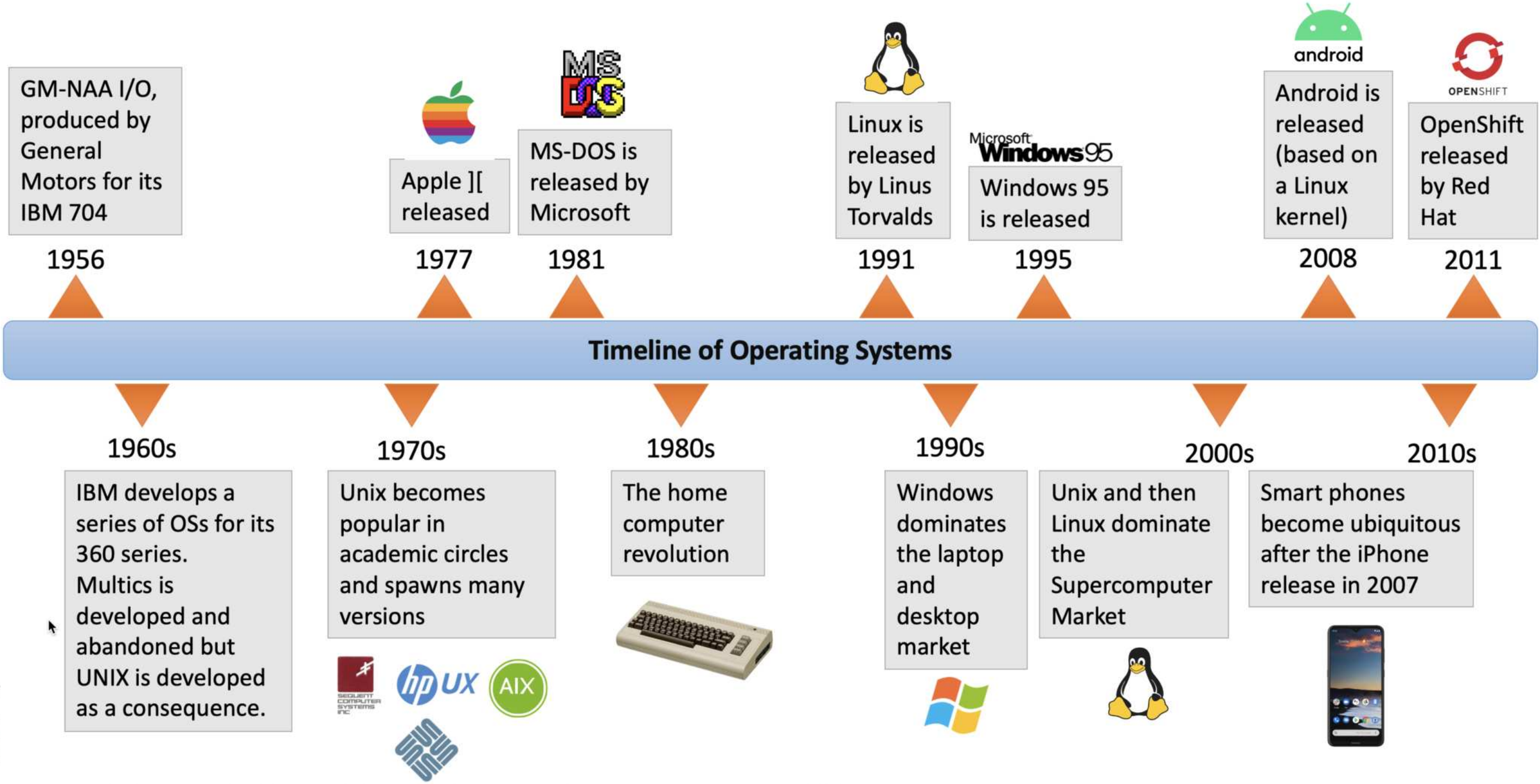
What if there were no OS?

- Application is a program: Processor will Fetch -> Decode -> Execute, Continue
- CPU will have the PC which points to the instructions in the memory



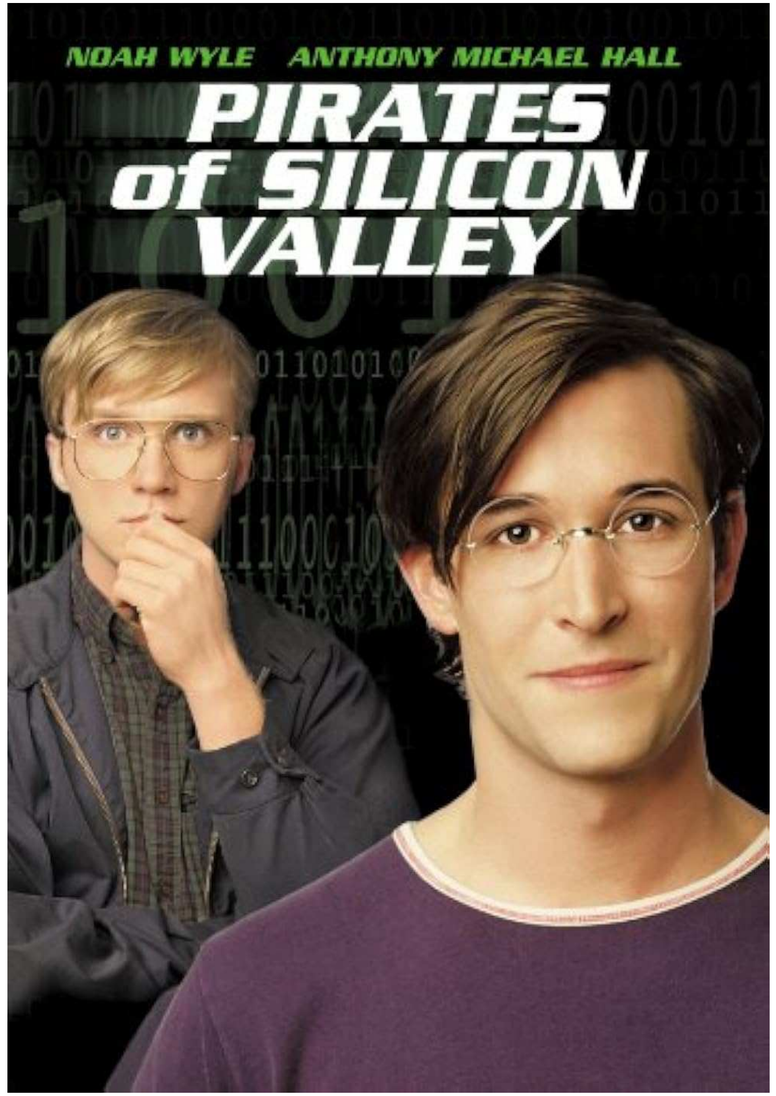


OS: A Brief History



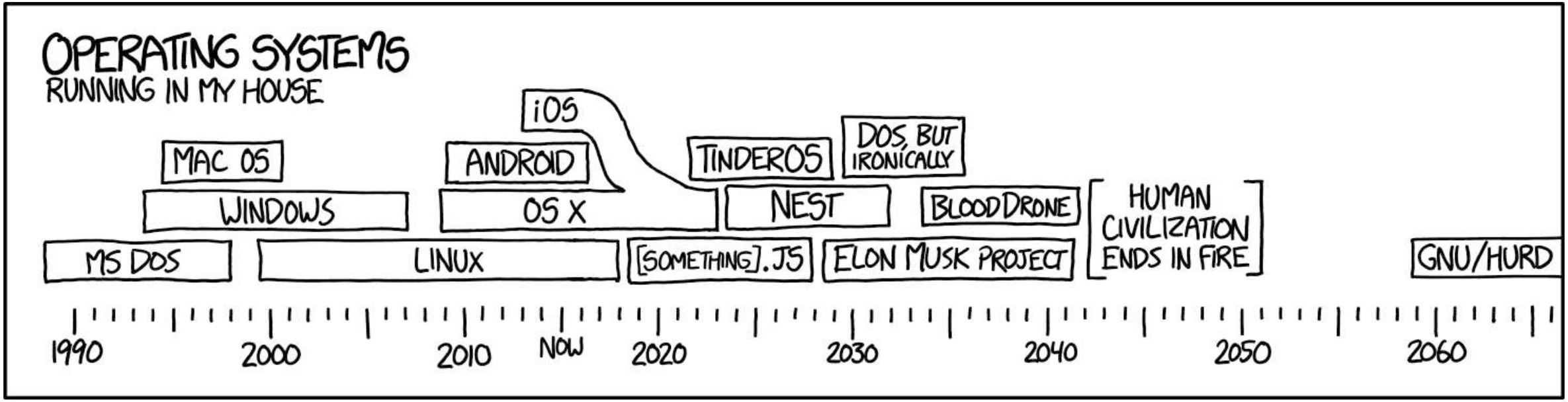
Source: https://www.orb-data.com/the-history-of-the-operating-system-from-paper-tape-to-openshift/

If you are into movies, do give a watch!



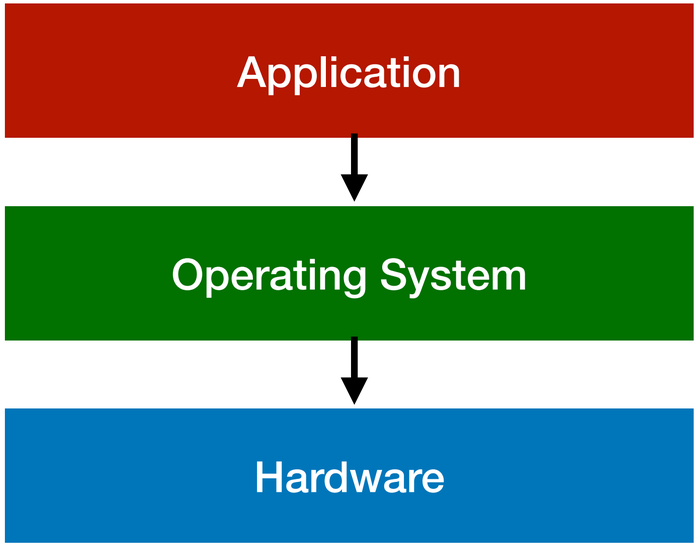
Source: imdb

Forever Lasting!



Source: xkcd.com

Operating Systems: An Overview



- OS basically is a middleware between the hardware and the application/user program
- In charge of making sure that the system operates correctly and efficiently
- Does three key things:
  - Easy to execute programs
  - Manages memory
  - Handles the different I/O devices

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

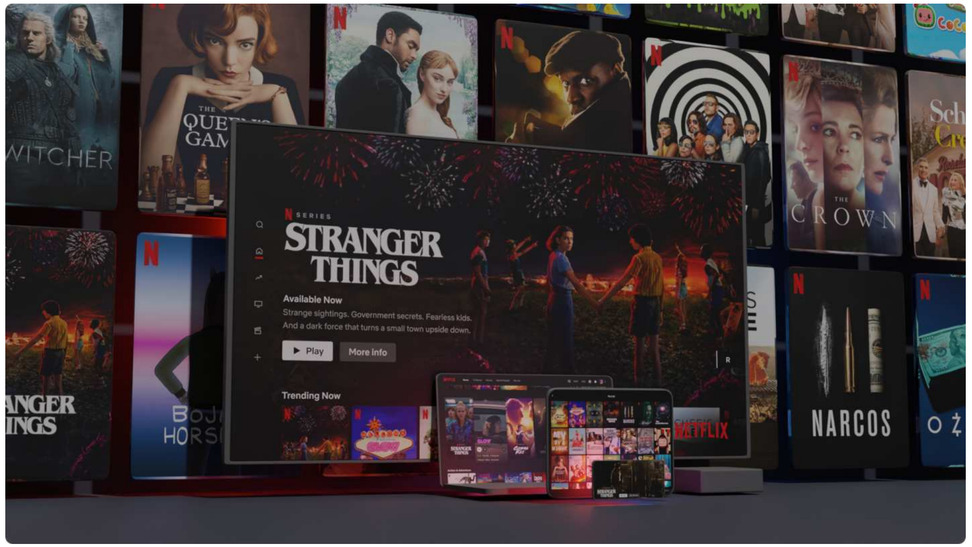
Whats the big deal?

- 8.5 million devices affected! - 1% of the entire windows users
- Airlines - Delta, KLM, Ryanair, Lufthansa,..
  - Many passengers were stranded in airports
- Financial organisations - London stock exchange, Visa, …
- Healthcare - independant pharmacies and GP surgeries
- Media, retail, train operating companies…….

Economic and social impact!!!

Abstraction holds the key

Lets draw some parallels



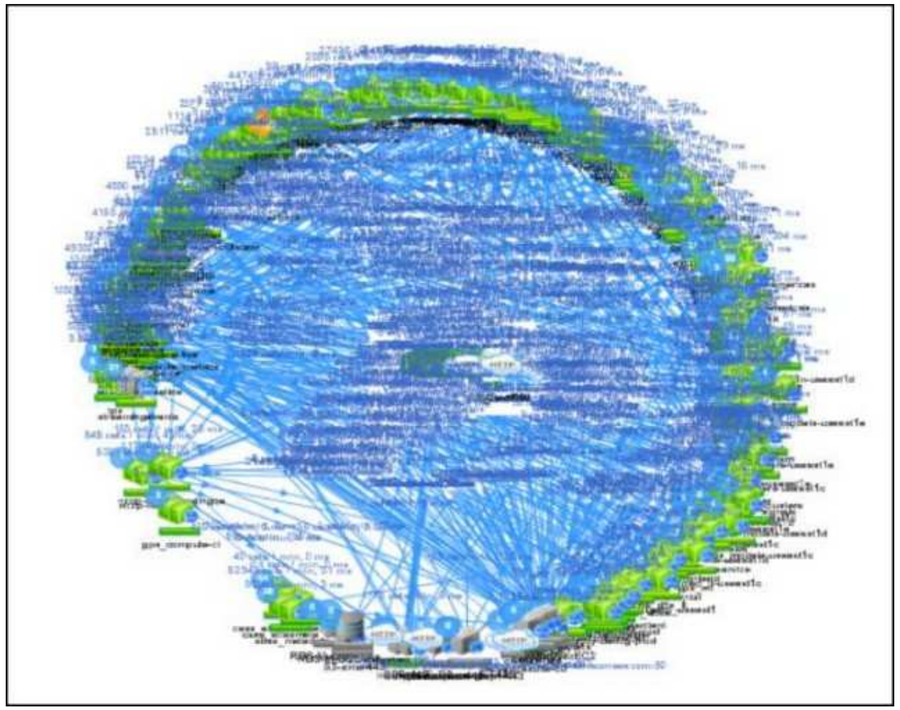
What you see as Netflix!

Source: netflix.com

- One of the key principles in system design - Try to keep things simple
- Abstraction and decomposition - Two main principles in Software Engineering
- Think of libraries in programming languages
  - Provides abstraction in terms of functions
  - All you need to know is functions to be called - Not how they work

Abstraction holds the key

Lets draw some parallels



Behind the scenes!

Source: netflix.com

- One of the key principles in system design - Try to keep things simple
- Abstraction and decomposition - Two main principles in Software Engineering
- Think of libraries in programming languages
  - Provides abstraction in terms of functions
  - All you need to know is functions to be called - Not how they work

OS can be also thought of as a black box!

Keeping things Simple is the Key!

1134 IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 47, NO. 6, JUNE 2021

Evolution of the Unix System Architecture: An Exploratory Case Study

Diomidis Spinellis, *Senior Member, IEEE* and Paris Avgeriou, *Senior Member, IEEE*

Abstract—Unix has evolved for almost five decades, shaping modern operating systems, key software technologies, and development practices. Studying the evolution of this remarkable system from an architectural perspective can provide insights on how to manage the growth of large, complex, and long-lived software systems. Along main Unix releases leading to the FreeBSD lineage we examine core architectural design decisions, the number of features, and code complexity, based on the analysis of source code, reference documentation, and related publications. We report that the growth in size has been uniform, with some notable outliers, while cyclomatic complexity has been religiously safeguarded. A large number of Unix-defining design decisions were implemented right from the very early beginning, with most of them still playing a major role. Unix continues to evolve from an architectural perspective, but the rate of architectural innovation has slowed down over the system's lifetime. Architectural technical debt has accrued in the forms of functionality duplication and unused facilities, but in terms of cyclomatic complexity it is systematically being paid back through what appears to be a self-correcting process. Some unsung architectural forces that shaped Unix are the emphasis on conventions over rigid enforcement, the drive for portability, a sophisticated ecosystem of other operating systems and development organizations, and the emergence of a federated architecture, often through the adoption of third-party subsystems. These findings have led us to form an initial theory on the architecture evolution of large, complex operating system software.

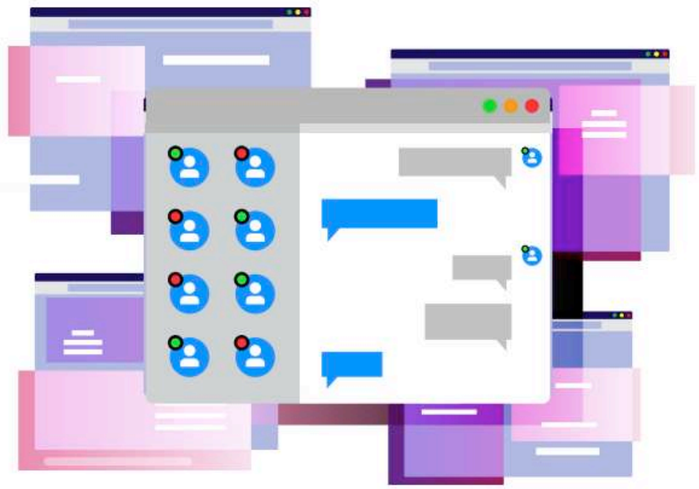
Index Terms—Unix, software architecture, software evolution, architecture design decisions, operating systems

Diomidis Spinellis, Paris Avgeriou, **Evolution of the Unix System Architecture: An Exploratory Study**, IEEE TSE, 2021 https://ieeexplore.ieee.org/abstract/document/8704965/

What does an OS abstract?

Think about the messaging system that needs to be developed

- The messaging application runs on the CPU
- It requires RAM to run but it feels that it has infinite RAM
- There are also other application that needs to run simultaneously
- The application also needs to store data -> images, videos, documents…
- Application also have to interact with devices (Camera, USB,…)



The Three Pillars of Operating Systems

Virtualization (Process and Memory)

1. OS gives every process a feeling that it has own CPU
2. Every process feels that its enough memory

Concurrency

1. Multiple process can run at the same time without resulting in problems
2. OS Provides mechanisms to make them work together

Persistance

1. Disk is an I/O device. It needs to be managed and handled.
2. OS handles interactions with the disk and performs storage management

Process Virtualization

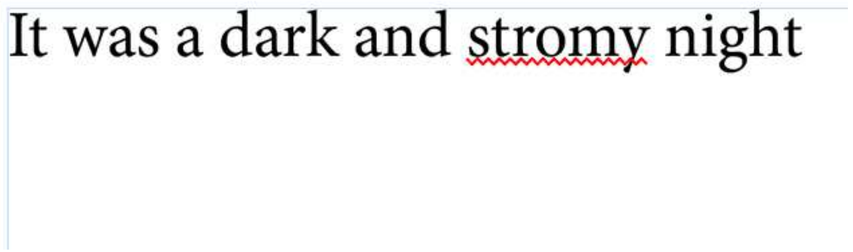
- How many CPU does your computer have?
- Is the number of process equal to the number of CPU your computer have?
- Lets look into a simple example…

Memory Virtualization

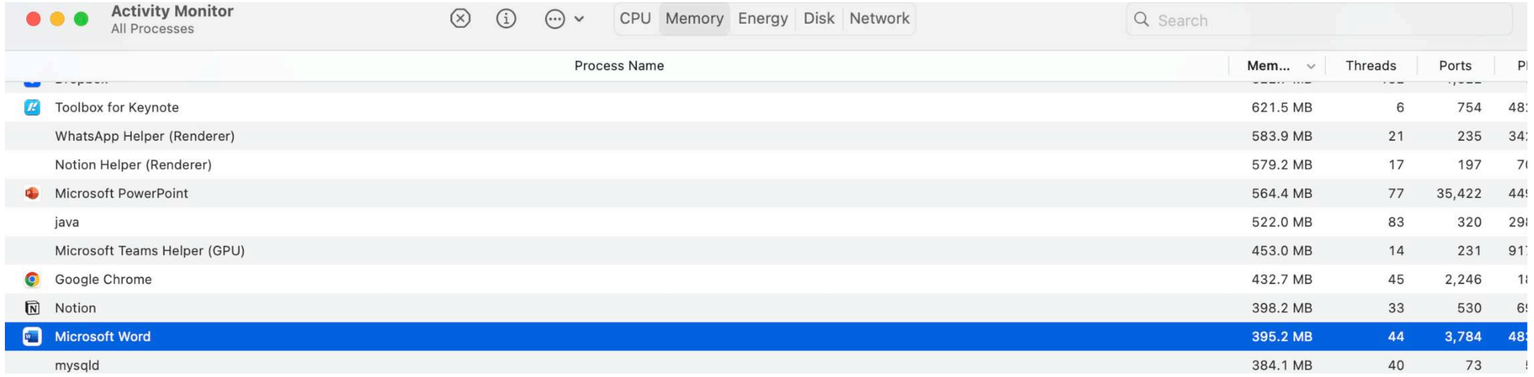
- Every process requires memory to run
- How many processes are active in your system?
- What’s the total memory they require?
- How’s CPU managing this?

Concurrency

- OS has to juggle between multiple processes
- Heard of multi-threaded programs? But wait heard of threads?



Spell check in Microsoft Word



How many threads are active in Microsoft Word?

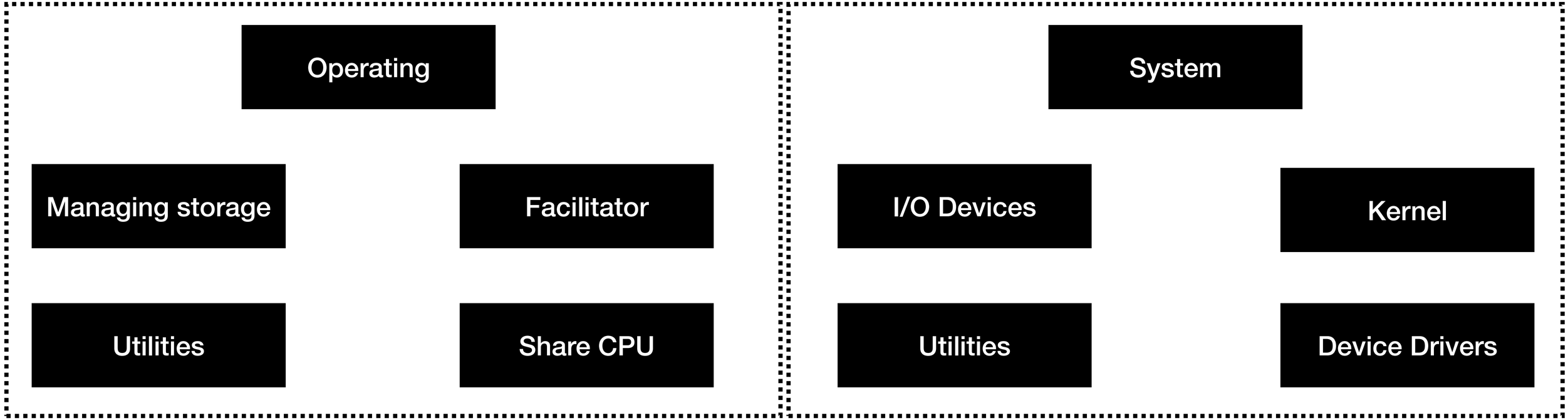
Spell check image source: https://creativepro.com/check-spelling-dynamically/

Persistence

- RAM is Volatile
- Hardware and software are needed to store data persistently
  - Hardware: I/O devices such as hard drive, SSDs, etc.
  - Software:
    - File system manages the disk
    - File system is responsible for any files that the user creates
    - Read, writes are handled by file system which interacts with low level device drivers

“Operating System” - Dual Roles

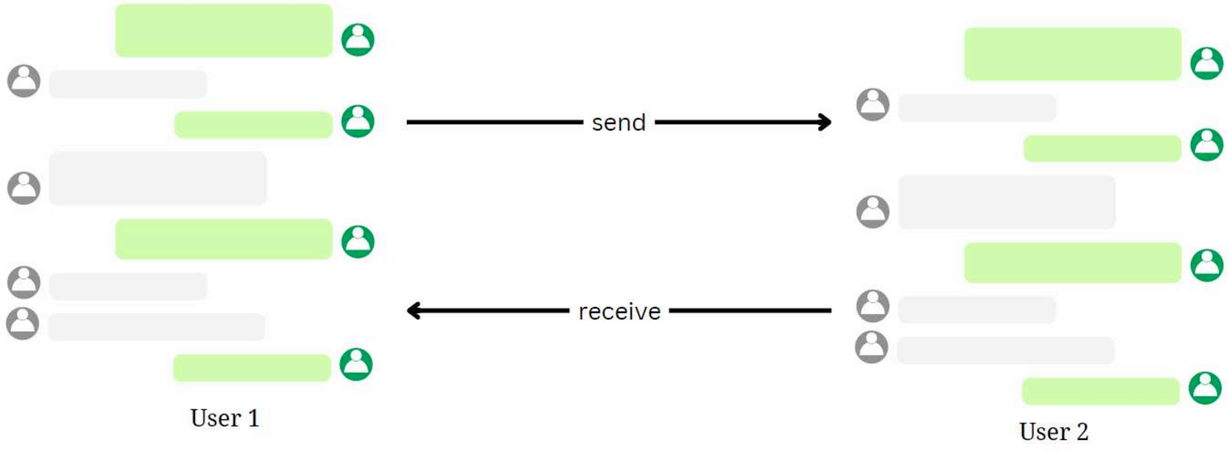
Resource Management and Hardware Abstraction



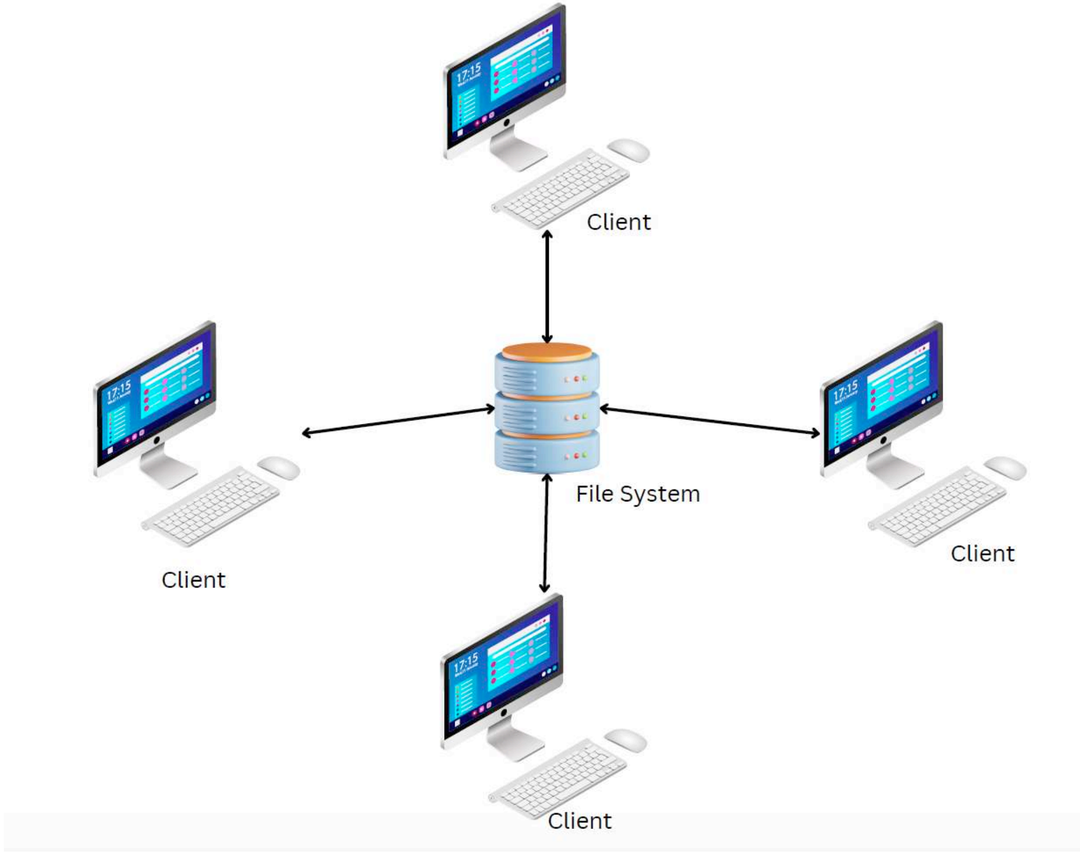
Key Design Goals Of OS

- **Abstraction**: Convenience and easy to use
- **High Performance**: Minimize overhead, Virtualisation should be done but minimise overhead
- **Reliability:** OS must continously run without crashing
- **Other attributes**: Energy efficiency, Security, Mobility
- **Remember:** OS itself is a massively complex software (Softwares rely on OS to function correctly!!)

But how does multiple systems interact?

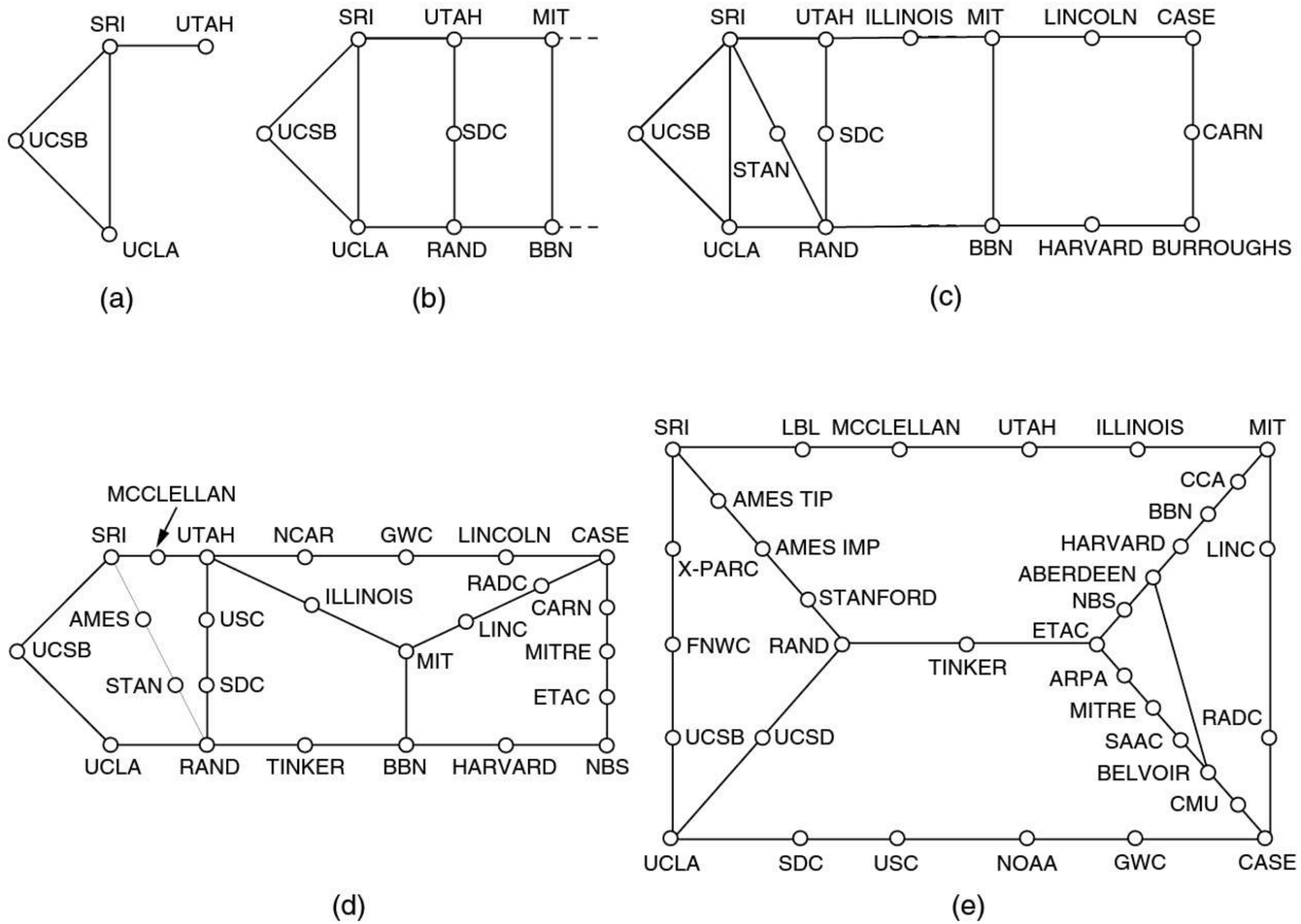


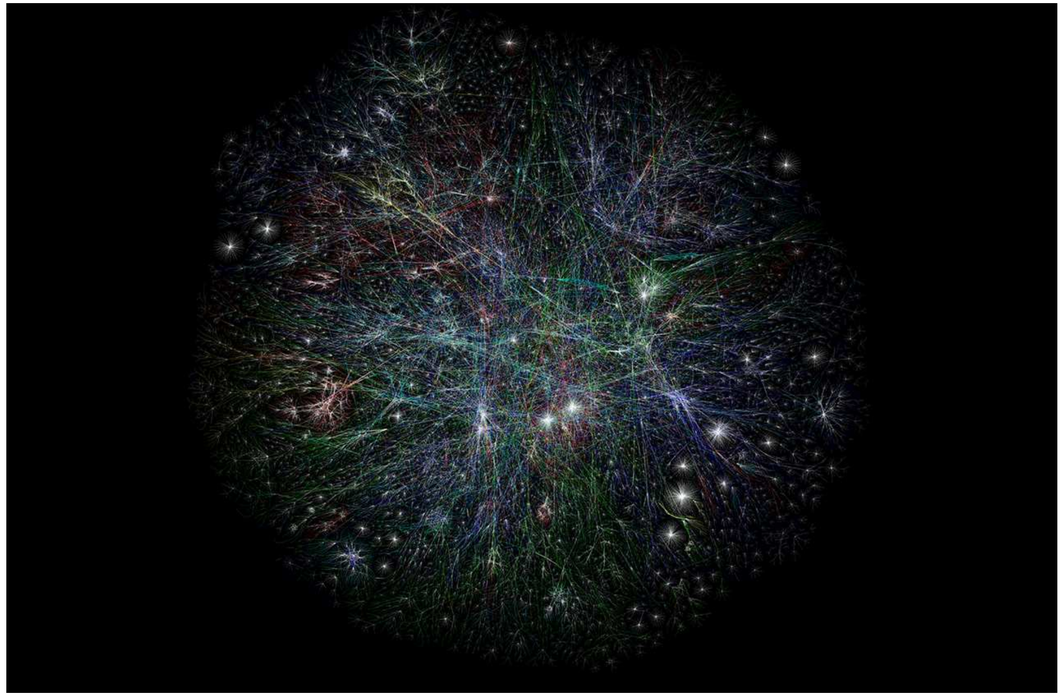
Whatsapp Chat example



Network File System

Any Guess on this?

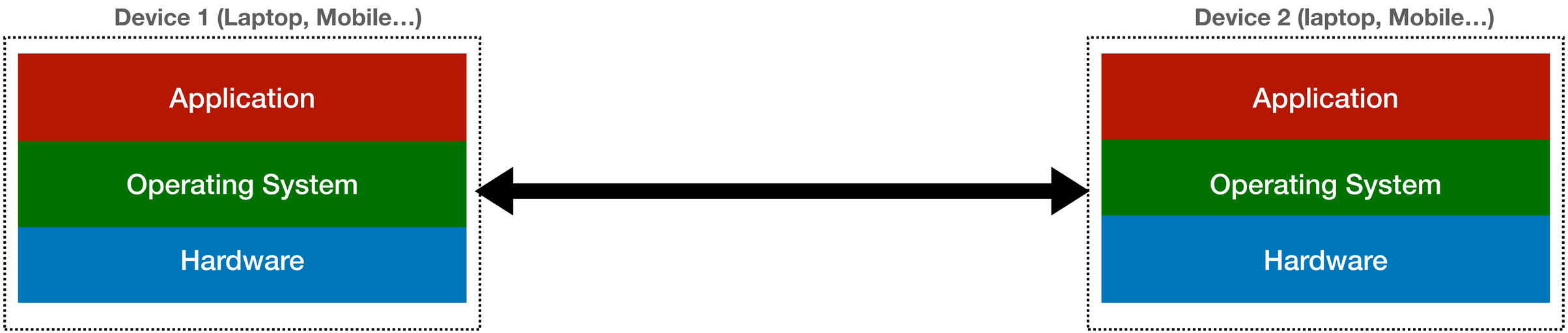




A Map of the internet (as of 2012)

Growth of ARPANET, source: Computer Networks, 5e, Tannenbaum and Weatherall

Computer Networks: The horizontal



- How can application from device 1 and 2 communicate?
- How to ensure the data/information reaches reliably?
- What if multiple applications are running and are communicating?

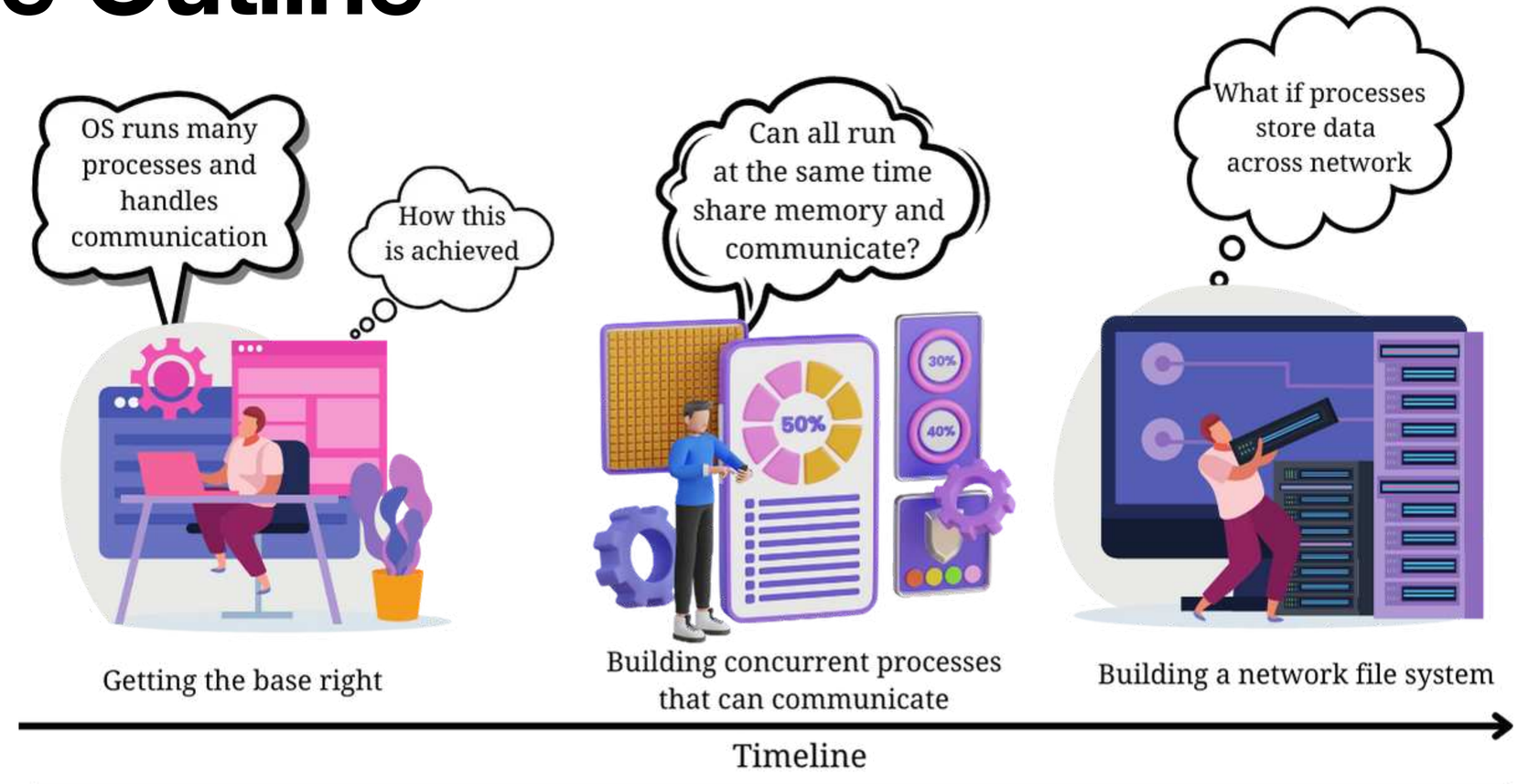
Networks - Different Components

- Different types of network - PAN, LAN, MAN, WAN,..
- Some terminologies - subnets, hosts, routers, switches, transmission lines, interconnected networks (internet) not Internet
- **internet** - formed when distinct networks are interconnected (network of networks)
- Subnets - collection of communication lines and routers owned by network operators
- Protocol - Agreement between communicating parties on how the communication needs to proceed

The Different Pieces of Networks

- As more and more networks started joining ARPANET, need for common protocol arise
  - Invention of TCP/IP model
- Implementing TCP/IP on different platforms was encouraged
  - Sockets - 4.2BSD Release of Berkeley Unix
  - Using networks with sockets proved effective and efficient
- As scale increased, difficult to remember the address of the system
  - Domain Name System (DNS)
  - How to find route from one point to another?

Course Outline



This Course

Process and Memory Virtualization	Concurrency	Persistence
Networking intro	Addressing and Routing	Network file Systems

Grade Distribution

Component	Weightage
Final Exam	30%
Mid-term Exam	15%
Quiz	5%
Mini projects	30%
In-class activities/Bonus	5%
Project	15%

Note: The instructor reserves the right to make any changes based on the course progress

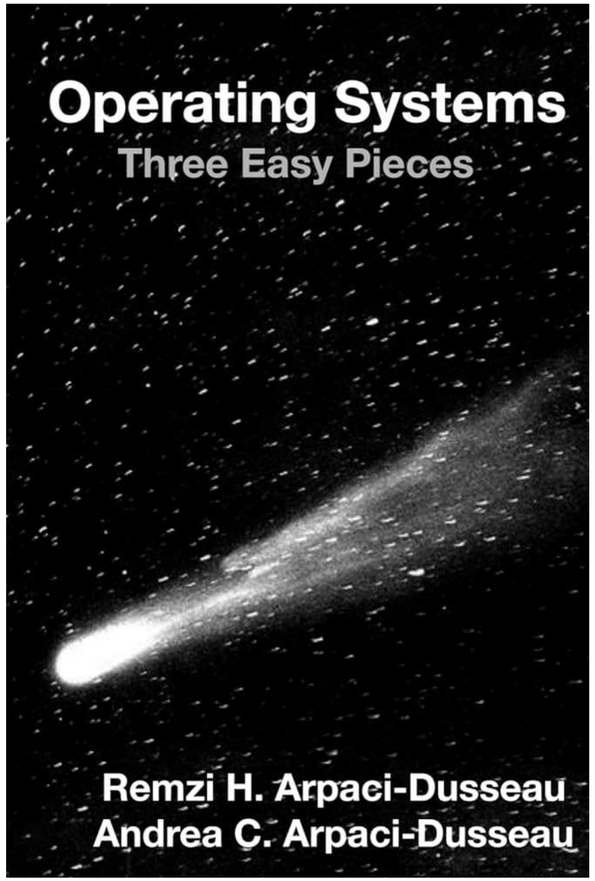
Course Logistics

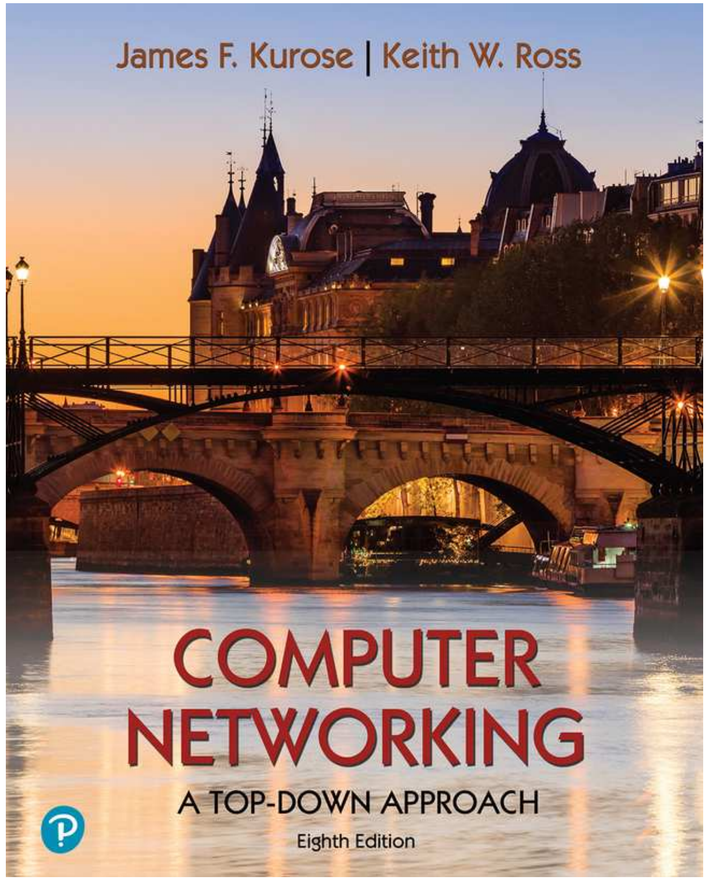
- Course announcements and management - Moodle
- Assignments and projects - Github Classroom
- All resources, information and materials - https://karthikv1392.github.io/cs3301_osn/
- The complete course schedule is already posted (minor changes may happen!)
- Instructor office hours - Thursday 12:00 PM to 1:00 PM
- Lets keep the classes lively and interactive!
- Feedbacks are always welcome!

OSN Continuous Feedback

Course Logistics

Materials





- The OS book is available as e-copy: https://pages.cs.wisc.edu/~remzi/OSTEP/
- Very important to read the **book + the slides + website posts + being attentive!!**

Course Logistics

Some Ground Rules

- **Zero Tolerance on Plagiarism!**
  - Every submission will go through different checks
  - If found, the points will be 0 and may result in an F grade
  - Check the institute policy on Academic misconduct
- You may use AI tools but use them wisely! - Check course policy page
  - Try to understand rather than blindly using generated code!
- Please respect the class timings!

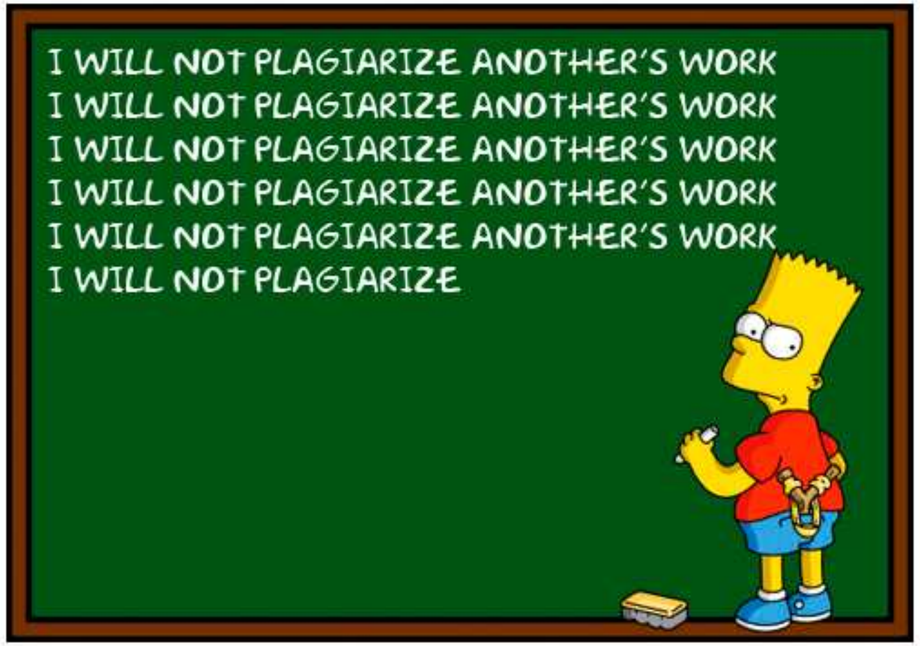


Image Source: Florida Tech News

Course Logistics

Communication and Engagement

- We have planned many ways to make this a very engaging course
- Take part in tutorial sessions - Not just for projects but for more understanding!
- Instructor office hours are available - Walk into office for any queries!
- TA Office hours - Pleaser do use them but kindly respect the timings!
  - Remember TAs are also students and they have their own commitments
- Document will be shared for posting questions
  - Check the questions before you post one!
  - Add questions to the respective category

Course Logistics

Projects

- There will be a mini project 0 (Warmup) and three other mini projects each about a month duration
  - All the mini projects will be graded, there will be viva as well (total 30%)
  - Mini project 3 will be a team project (team of 2)
- Final course project (15%) - Team of 4
- No extensions will be granted unless there is any emergency (get approval):
  - Use extra days and soft deadlines
  - Any submission over the hard deadline will not be considered and will be evaluated with 0

Meet the Team!

Divij D

Ashna Dua

Jhalak Akhilesh Banzal

Aryan Chandramania

Special Mention: Last year TAs

Karthik Vaidhyanathan

Divyansh Pandey

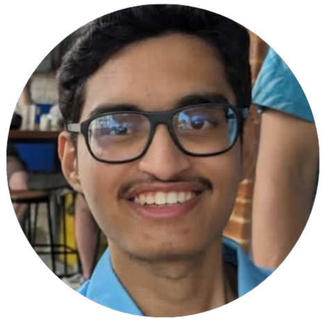
Ishan Gupta

Kritin Maddireddy

Prakhar Singhal

Sanchit Jalan

Miryala Sathvika



Shrikara A

Tanish Gupta

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Virtualization - Process

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

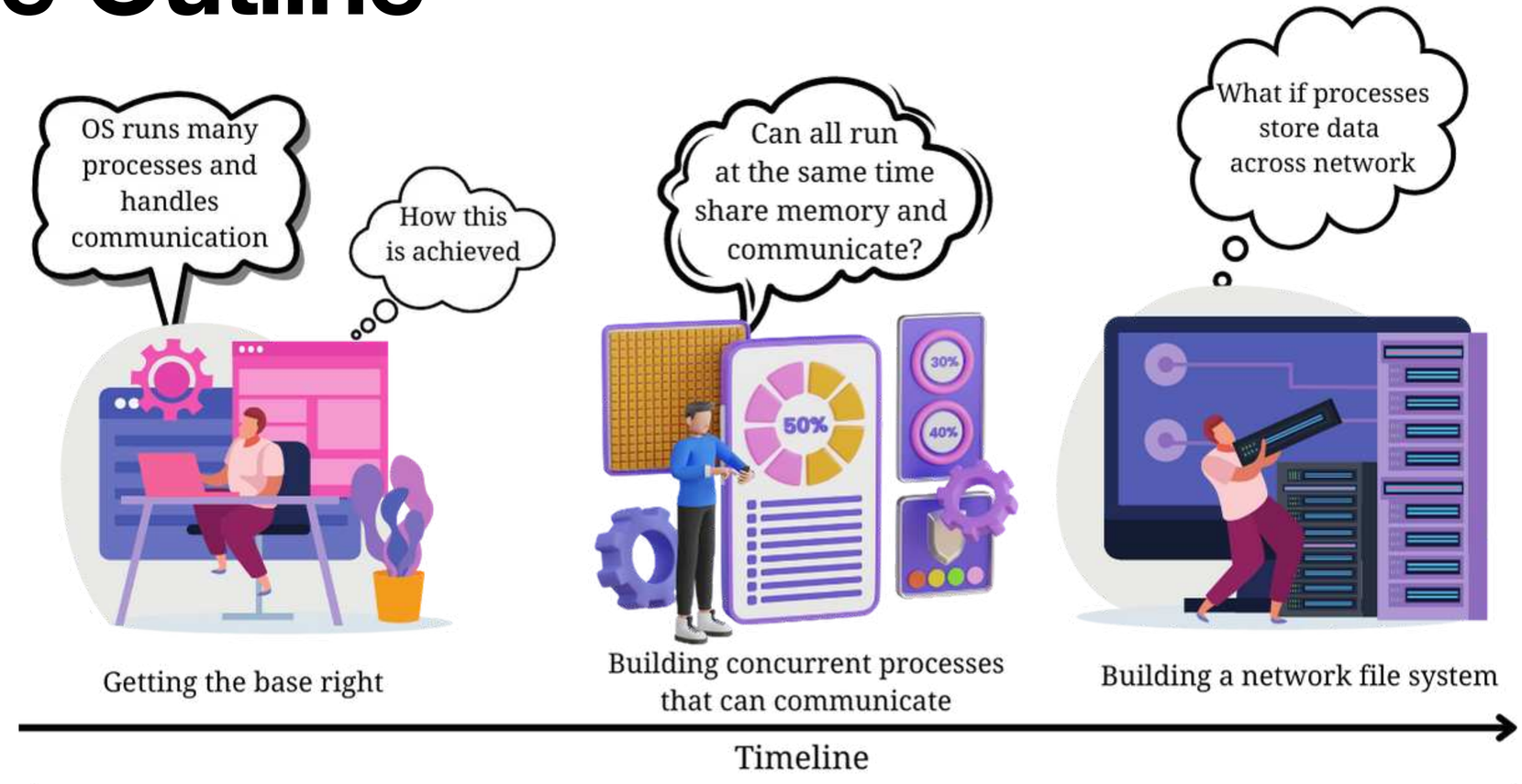
Acknowledgement

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.

Course Outline

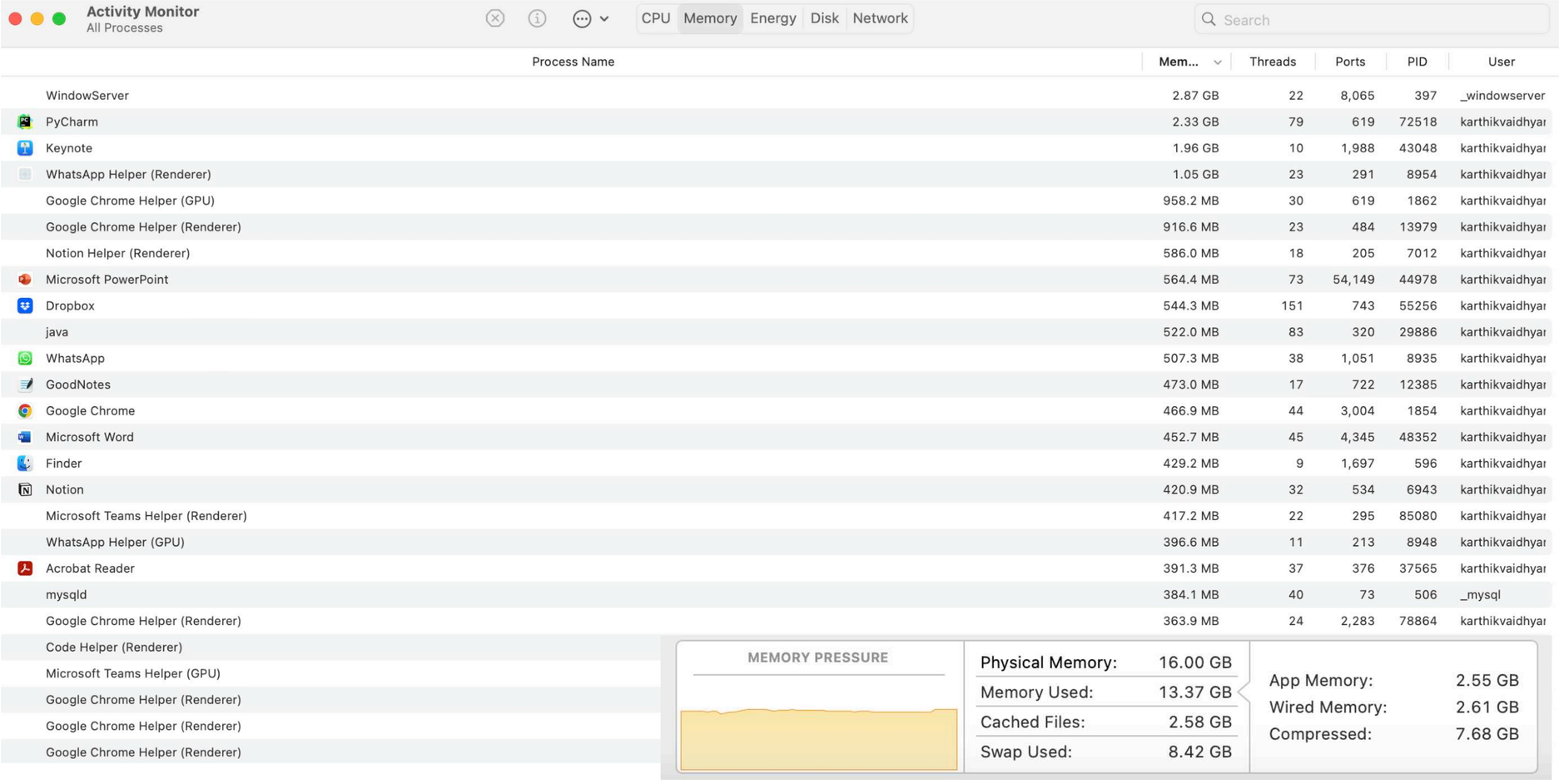


This Course

Process and Memory Virtualization	Concurrency	Persistence
Networking intro	Addressing and Routing	Network file Systems

Many processes run at the same time!

- How many processes are currently running in your machine?



Process Name	Mem...	Threads	Ports	PID	User
WindowServer	2.87 GB	22	8,065	397	_windowserver
PyCharm	2.33 GB	79	619	72518	karthikvaidhyar
Keynote	1.96 GB	10	1,988	43048	karthikvaidhyar
WhatsApp Helper (Renderer)	1.05 GB	23	291	8954	karthikvaidhyar
Google Chrome Helper (GPU)	958.2 MB	30	619	1862	karthikvaidhyar
Google Chrome Helper (Renderer)	916.6 MB	23	484	13979	karthikvaidhyar
Notion Helper (Renderer)	586.0 MB	18	205	7012	karthikvaidhyar
Microsoft PowerPoint	564.4 MB	73	54,149	44978	karthikvaidhyar
Dropbox	544.3 MB	151	743	55256	karthikvaidhyar
java	522.0 MB	83	320	29886	karthikvaidhyar
WhatsApp	507.3 MB	38	1,051	8935	karthikvaidhyar
GoodNotes	473.0 MB	17	722	12385	karthikvaidhyar
Google Chrome	466.9 MB	44	3,004	1854	karthikvaidhyar
Microsoft Word	452.7 MB	45	4,345	48352	karthikvaidhyar
Finder	429.2 MB	9	1,697	596	karthikvaidhyar
Notion	420.9 MB	32	534	6943	karthikvaidhyar
Microsoft Teams Helper (Renderer)	417.2 MB	22	295	85080	karthikvaidhyar
WhatsApp Helper (GPU)	396.6 MB	11	213	8948	karthikvaidhyar
Acrobat Reader	391.3 MB	37	376	37565	karthikvaidhyar
mysqld	384.1 MB	40	73	506	_mysql
Google Chrome Helper (Renderer)	363.9 MB	24	2,283	78864	karthikvaidhyar
Code Helper (Renderer)					
Microsoft Teams Helper (GPU)					
Google Chrome Helper (Renderer)					
Google Chrome Helper (Renderer)					
Google Chrome Helper (Renderer)					

What is a Process?

- A Program is nothing but code
- Processes are **running program**
- There can be more than one process that are created per program

Process Virtualization

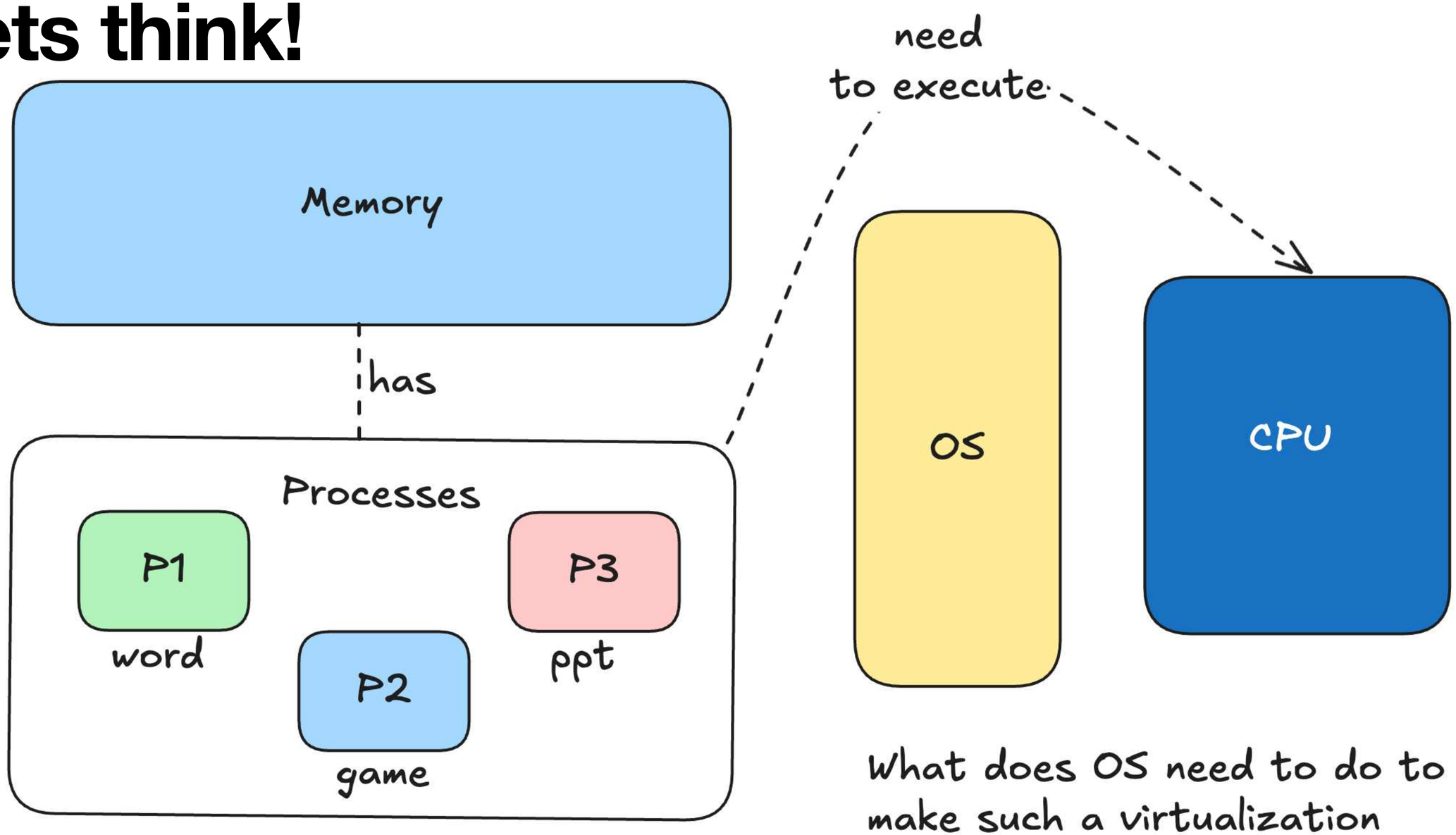
- Each process feels that it has its own CPU
- Even in Single core machines - There can be multiple process that run at the same time
- How is CPU handling this?

With limited CPU can we create an illusion that Endless CPU's are available?

OS achieves this using Virtualization of the CPU

Question: Can you of think of how such thing can be done?

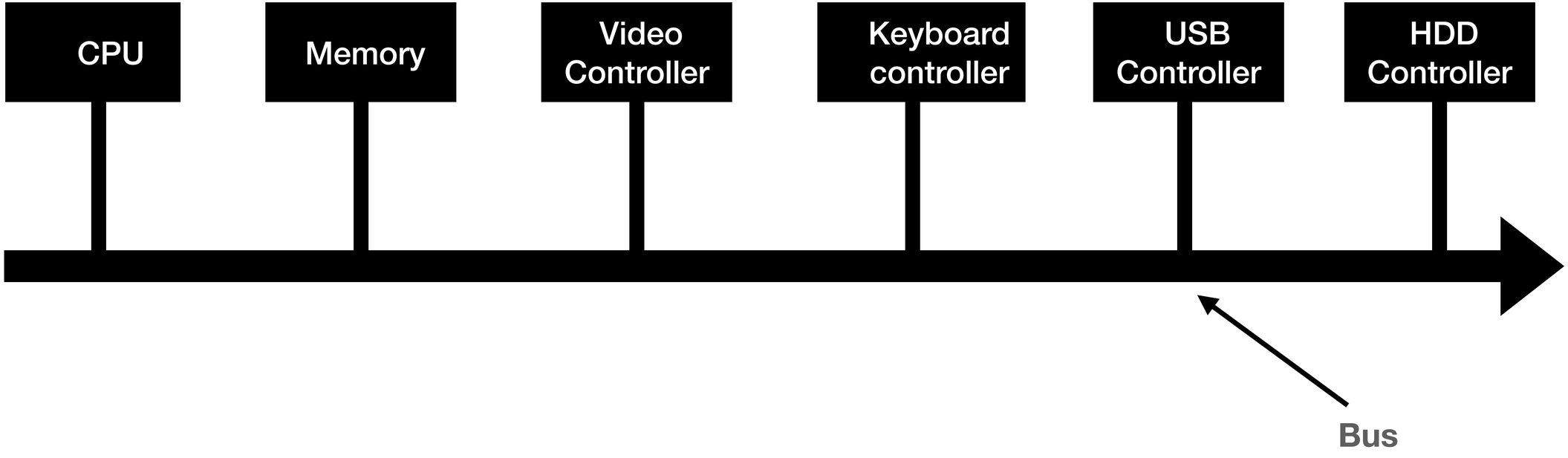
Lets think!



What does OS need to do to make such a virtualization happen?

Each process is made to feel that it has its own CPU

Some Prerequisite



As we go more away from CPU, the more time it takes

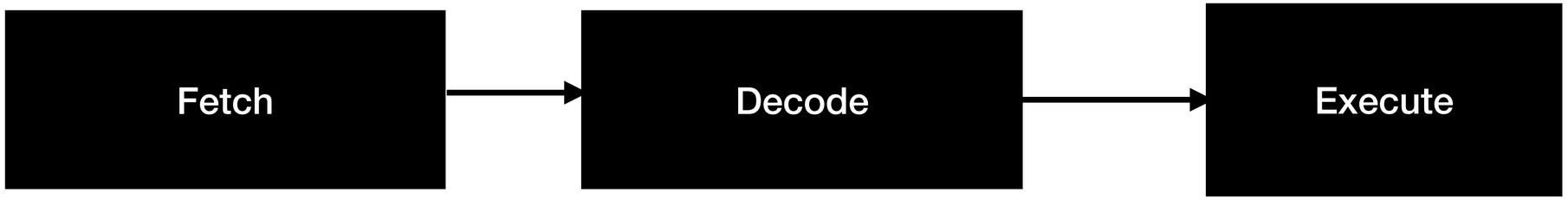
Some Prerequisite - Computer Hardware

- CPU contains some registers
  - Temporary registers
  - Program Counter (PC), Stack pointer (SP), Data register, address register,..
- Some key registers
  - Program counter - Points to the next instruction
  - Stack pointer - Points to top of the stack in the memory
  - Program Status word - Status of current state of CPU and program (condition bits)

Some Prerequisites

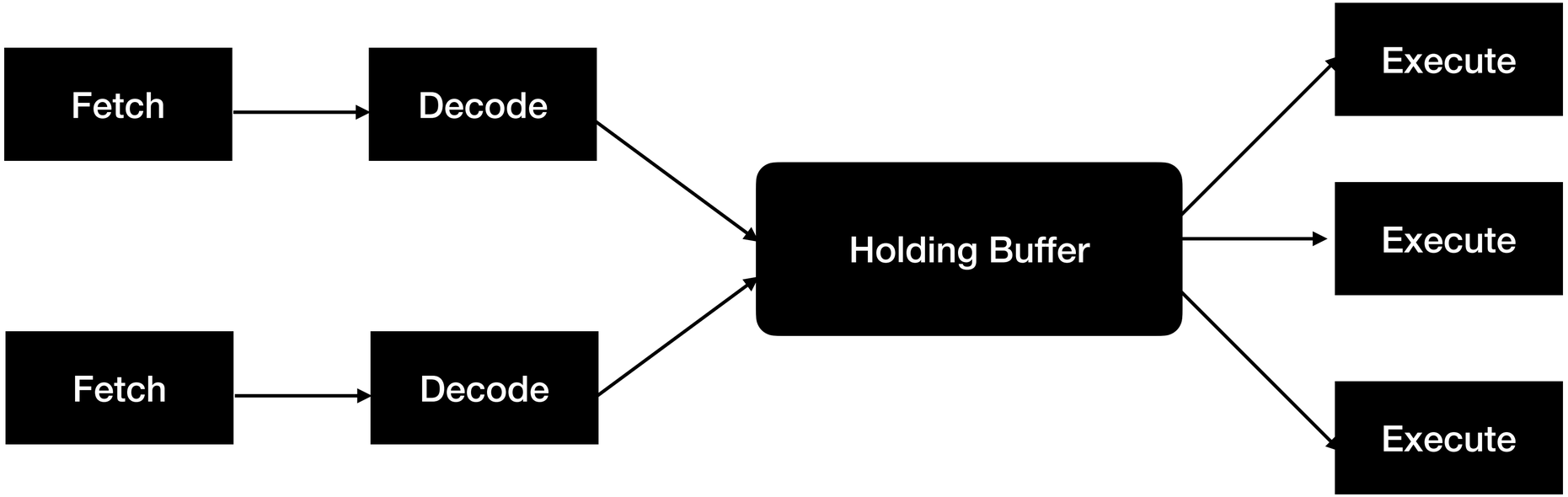
How does CPU execute a program?

- Three stage pipeline



Question: Do you believe that the current hardware structure is similar to this?

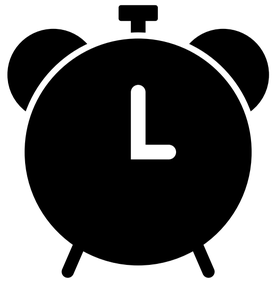
Some Prerequisite - Computer Hardware



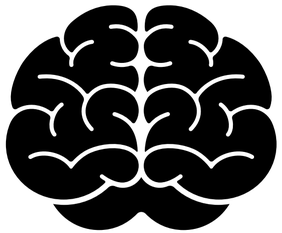
Superscalar CPU

How to make it at software level?

- **We do need support from the hardware**
  - Some mechanism to switch
  - Eg: Each process runs for a particular time and then we switch
  - Low-level mechanism (Context Switch)



- **We also need some intelligence in the software**
  - Some algorithm that can intelligently decide
  - Policies for switching



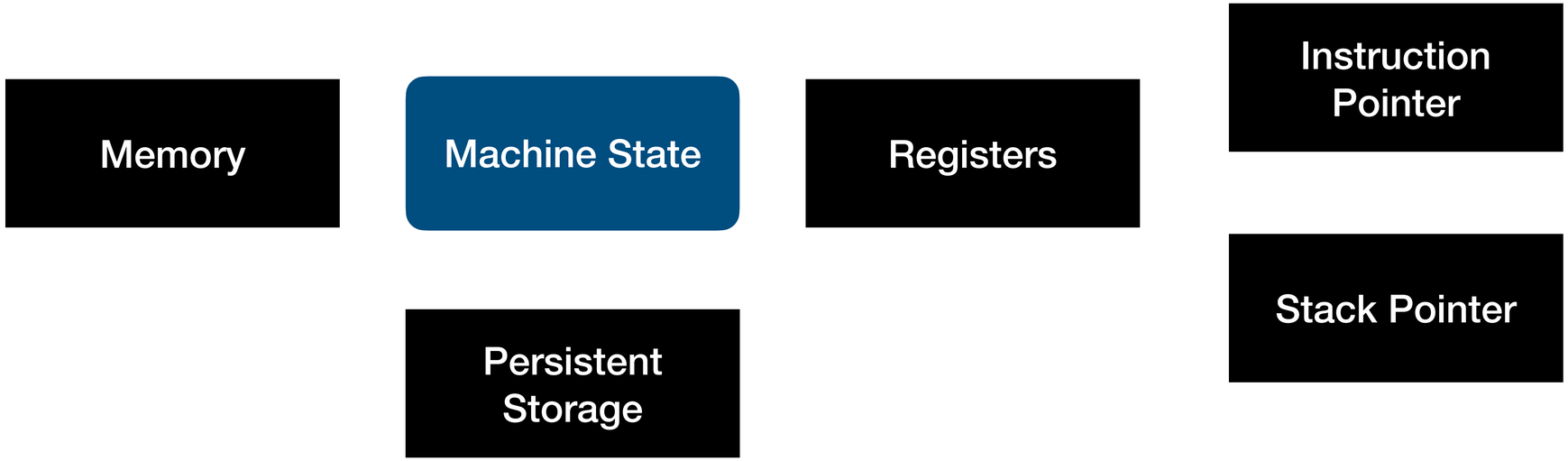
- Basically we need - **low level mechanisms** and **policies** (CPU Scheduler)

What Constitutes a Process?

Lets make it clear - Process is nothing but running program!!

- The Characteristics that make up a process (State)
  - What parts of the machine are important for execution?
- The most obvious component - Memory! Why?
  - Instructions lie in the memory, data (reads and writes) is in the memory
  - **Address space** is part of the process
- What else does a running program need?

What Constitutes a Process?



- **Memory** - address space (Memory that the process can address)
- **Instruction pointer, program counter** - which instruction is executed
- **Stack pointer**- local variables, functions and return addresses
- **Peristent storage** - I/O information

What Constitutes a Process?

- **Unique Identifier (Process ID)**
- **Memory Image**
  - Code and data (static)
  - Stack and Heap (Dynamic)
- **CPU Context: Registers**
  - Program Counter
  - Current Operands
  - Stack Pointer
- **File Descriptors**
  - Pointers to open files and devices

Memory Image of Process

Creation of a Process by OS

- **Load program into memory**
  - Initially program resides on the disk
  - OS does **lazy loading**
- **Allocate runtime stack**
  - Use for local variables
  - Function parameters and return arguments

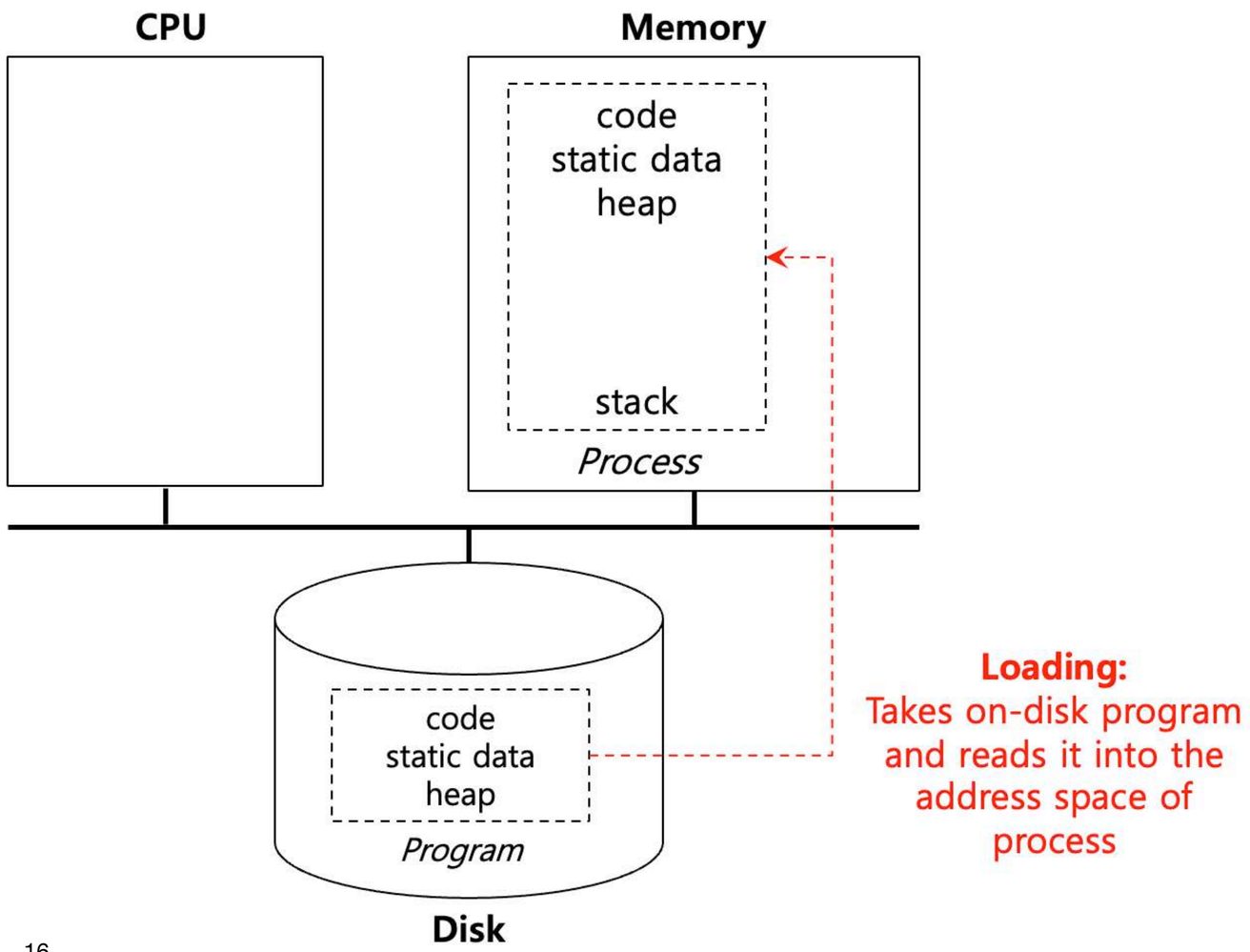


Image from OS lecture by Youjip Won

Creation of a Process by OS

- **Creation of Program heap**
  - Used for dynamically allocated data
  - malloc() and free()
- **Basic file setup**
  - STDIN, OUT, ERR
- **Initialise CPU registers**
  - PC to the first instruction
- **Start the program**

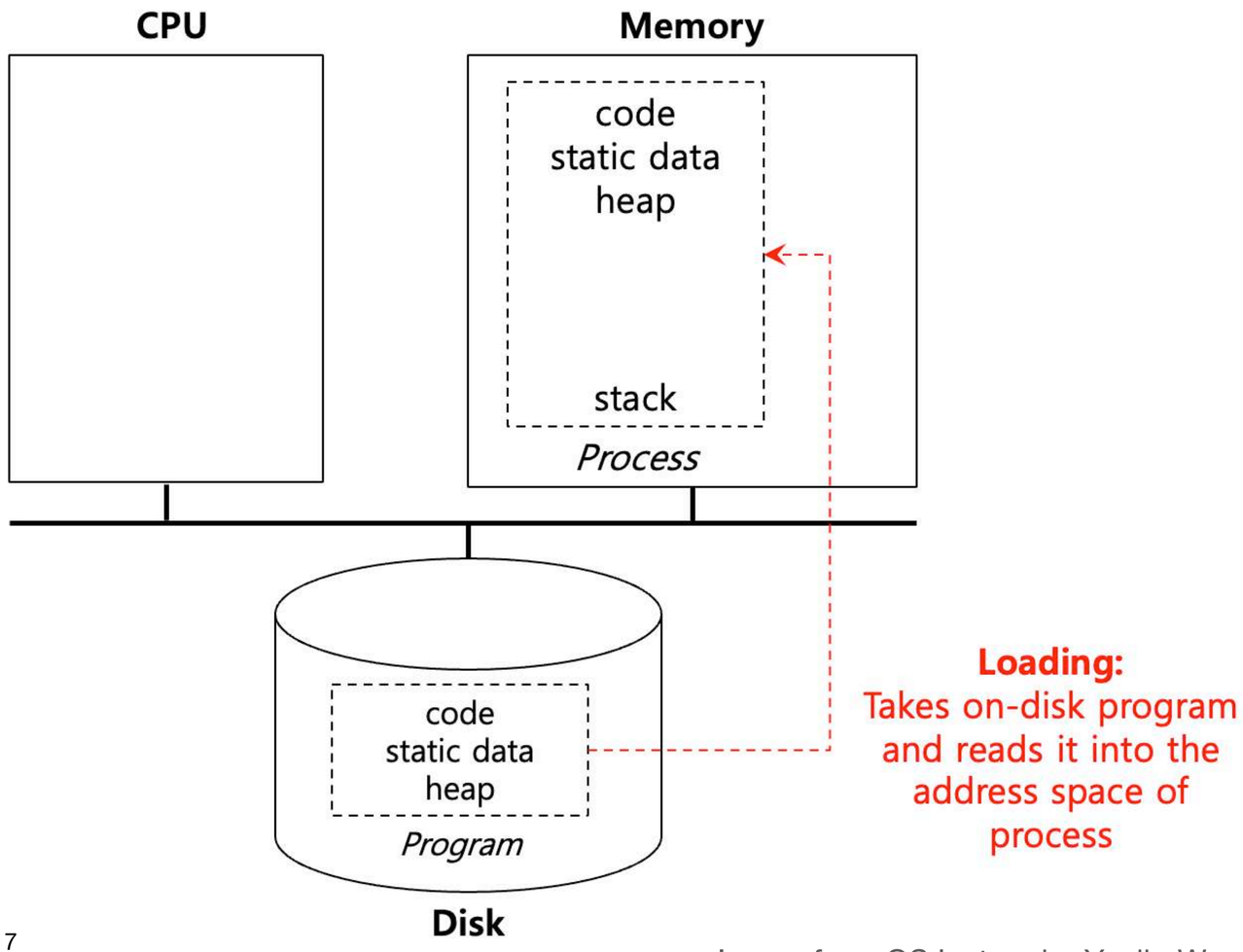


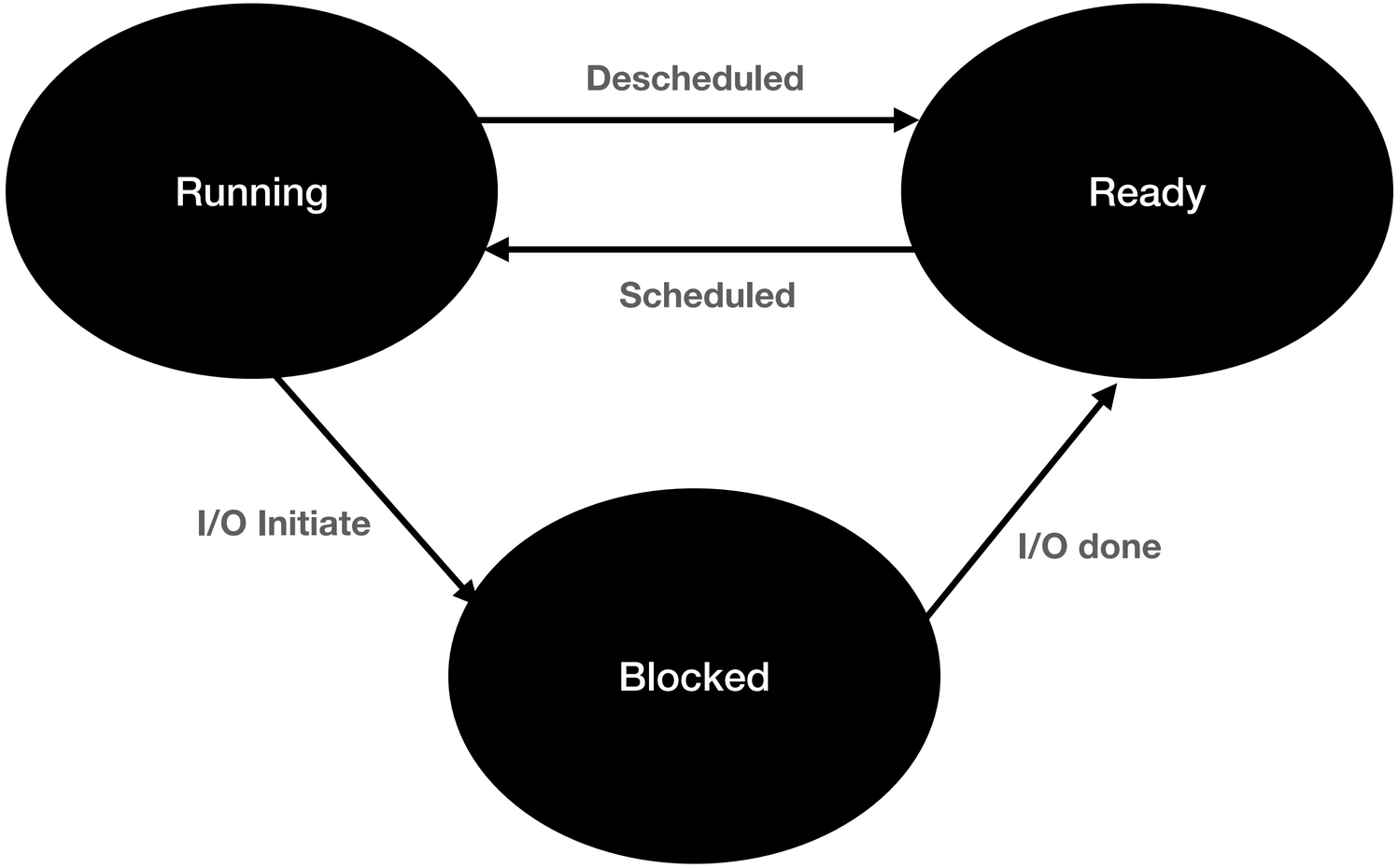
Image from OS lecture by Youjip Won

States of the Process

- At any point process can be in one of the following states
  - **Running** - Its running on the processor
  - **Ready** - Ready to run
  - **Blocked** - Not ready to run, something else is running
    - Any reason that you can think of?
  - **Think of I/O call** - Wait what does that mean?

States of the Process

Process State Transitions



Lets look at an Example

Time	Process 0	Process 1	What's happening
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	Process 0 initiates I/O
4	Blocked	Running	Process 0 is blocked, 1 runs
5	Blocked	Running	
6	Blocked	Running	I/O of process 0 is done
7	Ready	Running	Process 1 is done
8	Running	-	Process 0 is done

How to store Metadata? - Use data structures

- Need for some mechanism to store the state of the process
- **Remember:** OS is a software
  - It leverages data structures to store the information
  - OS makes use of data structure called, **process list**
  - What to store inside each? - Process Control Block (PCB)
    - Process id? - Identification of the process
    - State of the process - ready, running or blocked
    - Address space of the process - the registers

Process List

Xv6 Operating System

Teaching OS developed by MIT - Replicate basic Unix

https://pdos.csail.mit.edu/6.828/2012/xv6.html

Process Structure in Xv6

```
// the information xv6 tracks about each process
// including its register context and state
struct proc {
    char *mem;                  // Start of process memory
    uint sz;                    // Size of process memory
    char *kstack;               // Bottom of kernel stack
                                // for this process
    enum proc_state state;      // Process state
    int pid;                    // Process ID
    struct proc *parent;        // Parent process
    void *chan;                 // If non-zero, sleeping on chan
    int killed;                 // If non-zero, have been killed
    struct file *ofile[NOFILE]; // Open files
    struct inode *cwd;          // Current directory
    struct context context;     // Switch here to run process
    struct trapframe *tf;       // Trap frame for the
                                // current interrupt
};
```

Process Structure in Xv6

```
// the registers xv6 will save and restore
// to stop and subsequently restart a process
struct context {
    int eip;   // Index pointer register
    int esp;   // Stack pointer register
    int ebx;   // Called the base register
    int ecx;   // Called the counter register
    int edx;   // Called the data register
    int esi;   // Source index register
    int edi;   // Destination index register
    int ebp;   // Stack base pointer register
};

// the different states a process can be in
enum proc_state { UNUSED, EMBRYO, SLEEPING,
                  RUNNABLE, RUNNING, ZOMBIE };
```

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Process Virtualisation - API and Mechanisms

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 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

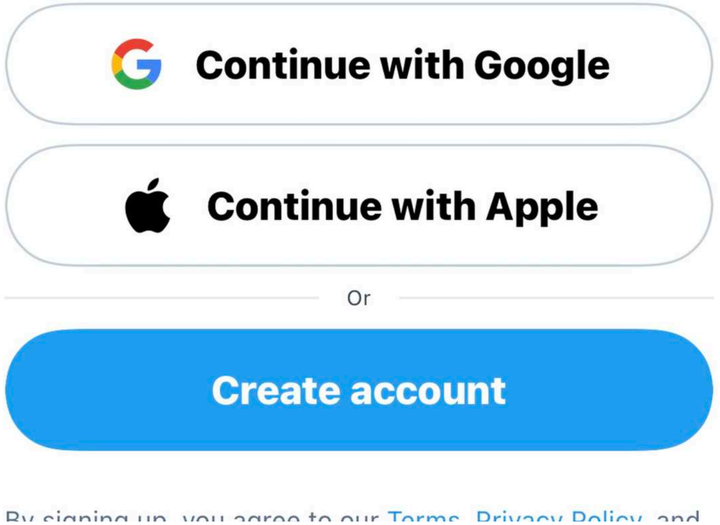
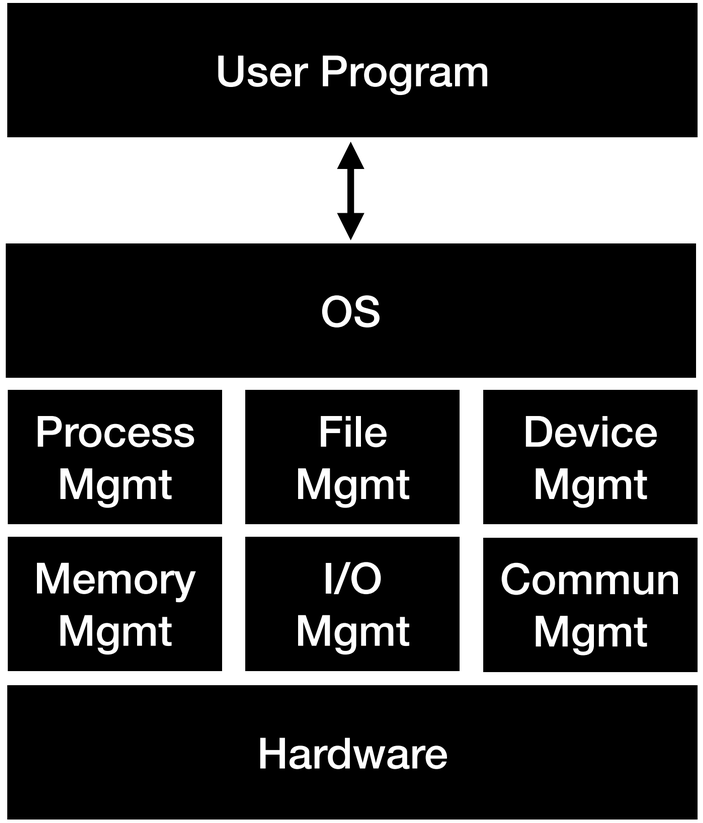


Image source: verge

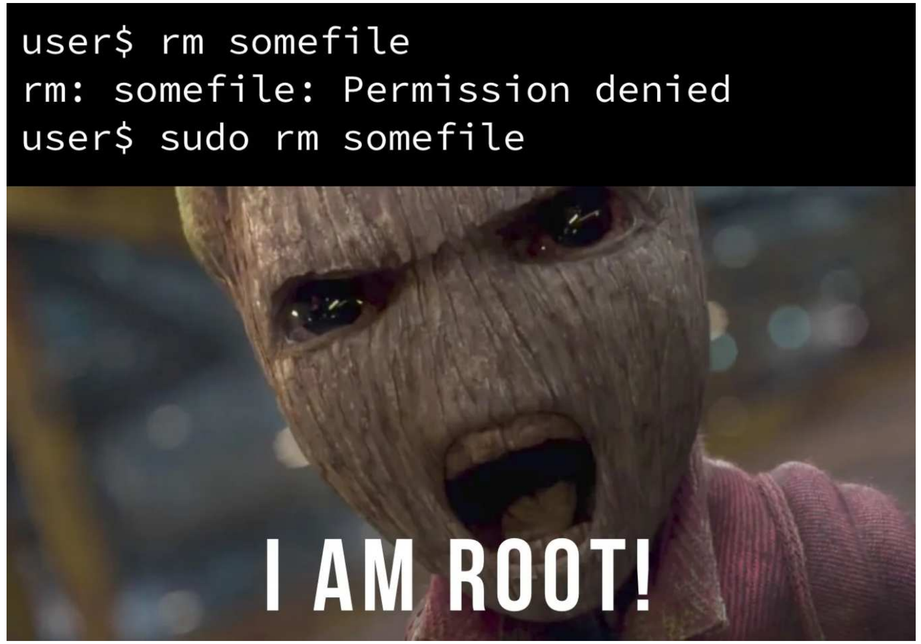
Does OS Provide API? - System Calls!

- 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!

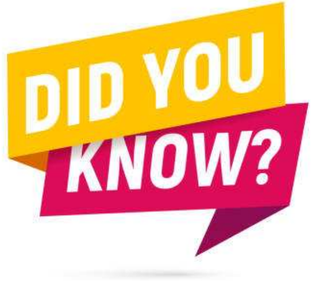
- What if a user gives a instruction to delete all files?
  - Should all the instructions be considered with equal priority?
  - When does the role of OS come in to the main picture?
    - Think about reading a file or writing a file - How to achieve it in C?
    - What if you just wanted to multiply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



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 <u>blocks</u> 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!**

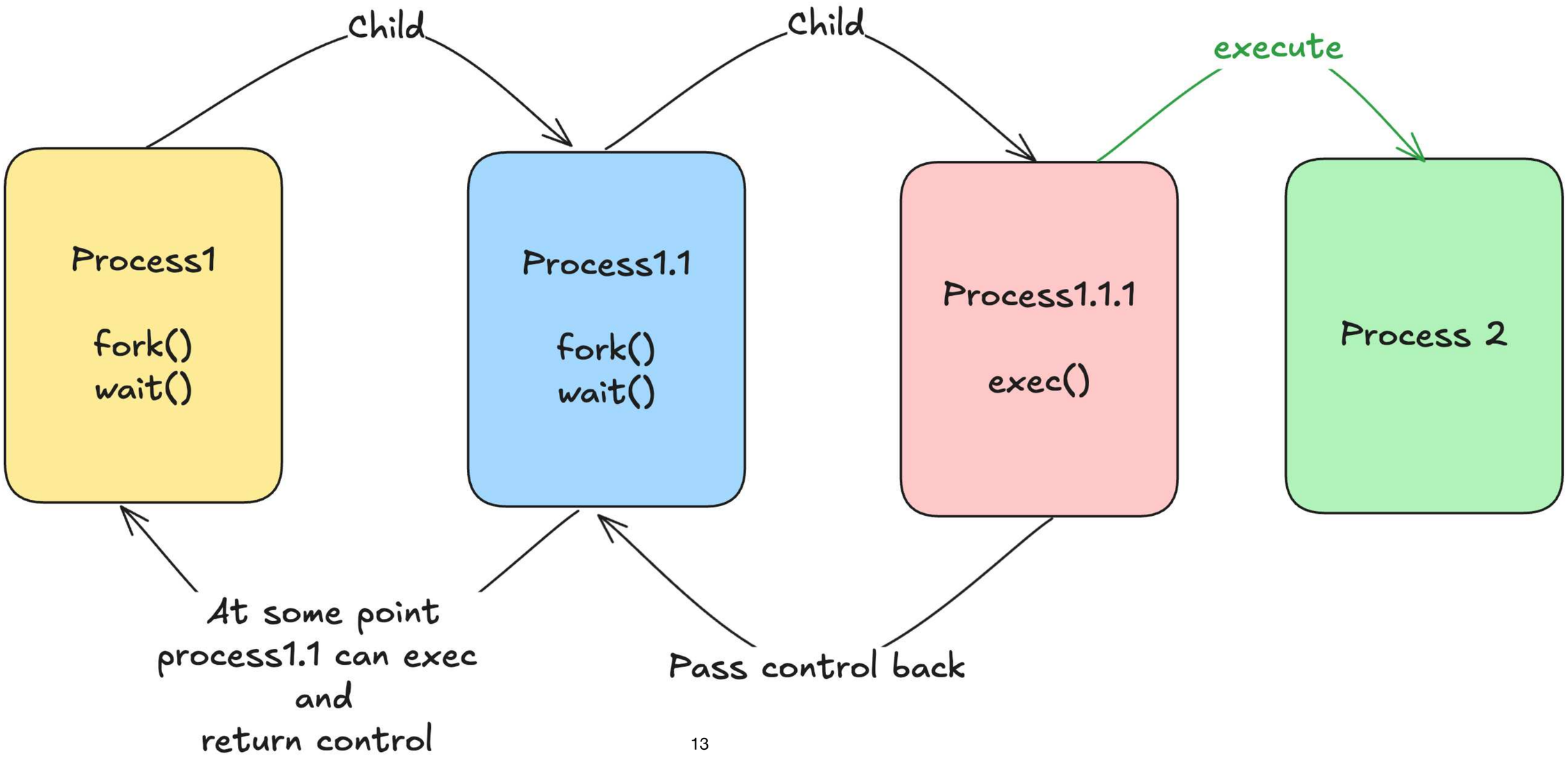
Remember: Init process, adopts orphans and reaps them

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?

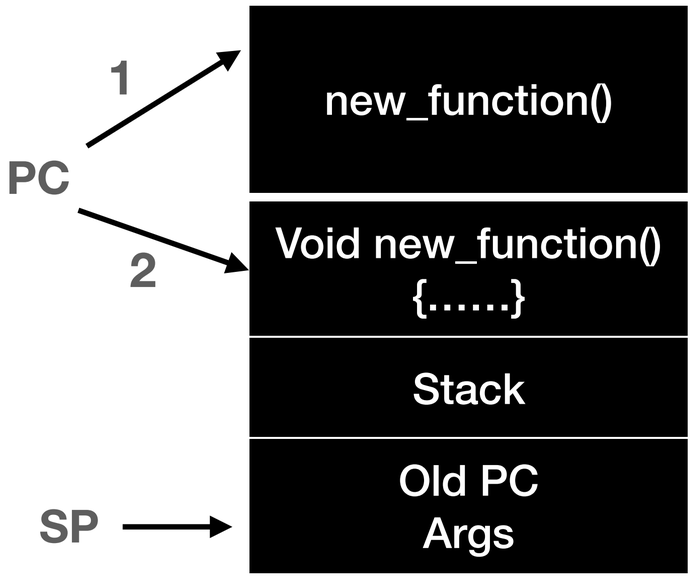
What if we allow process to do whatever it wants?

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

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

CS3.301 Operating Systems and Networks

Process Virtualisation - Mechanisms and Policies (Part 1)

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 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?

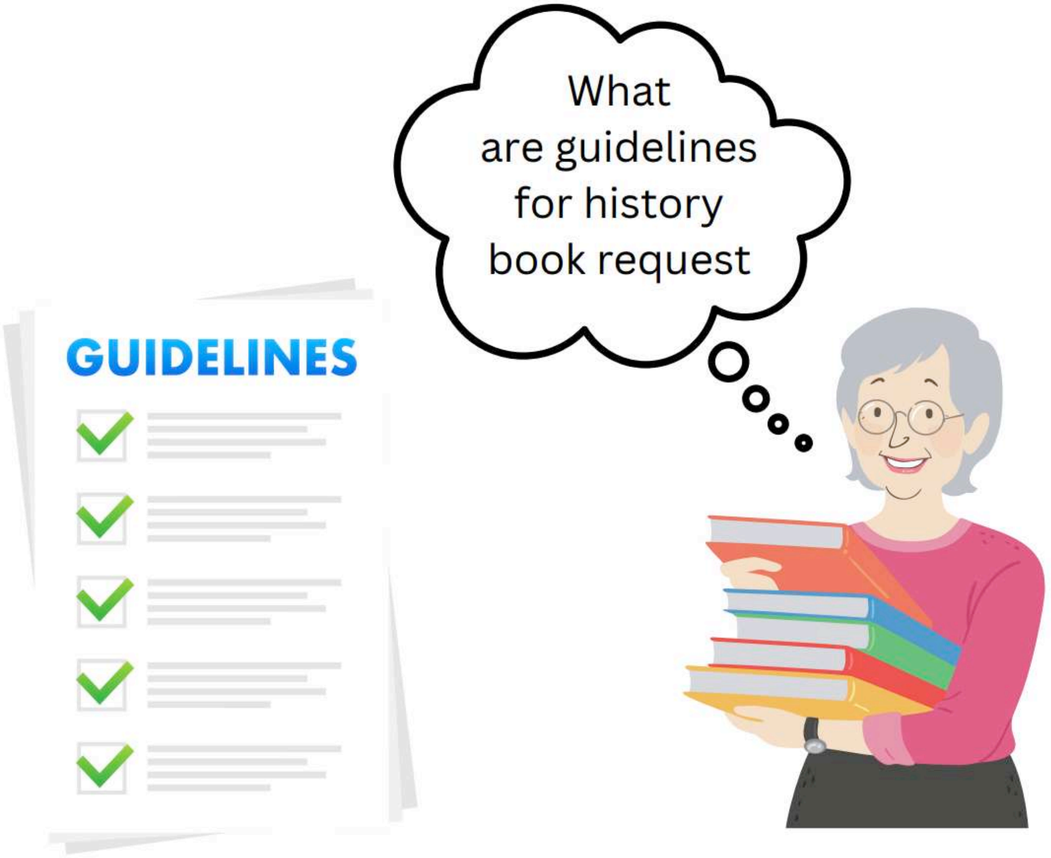
Lets draw some Parallels

Visitor/User

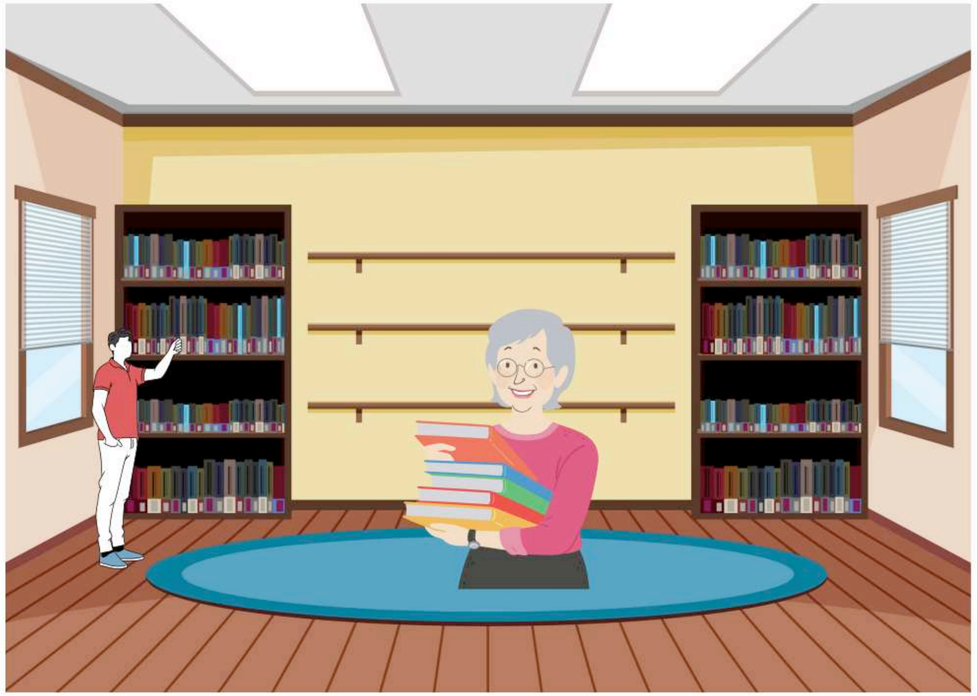
Librarian

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

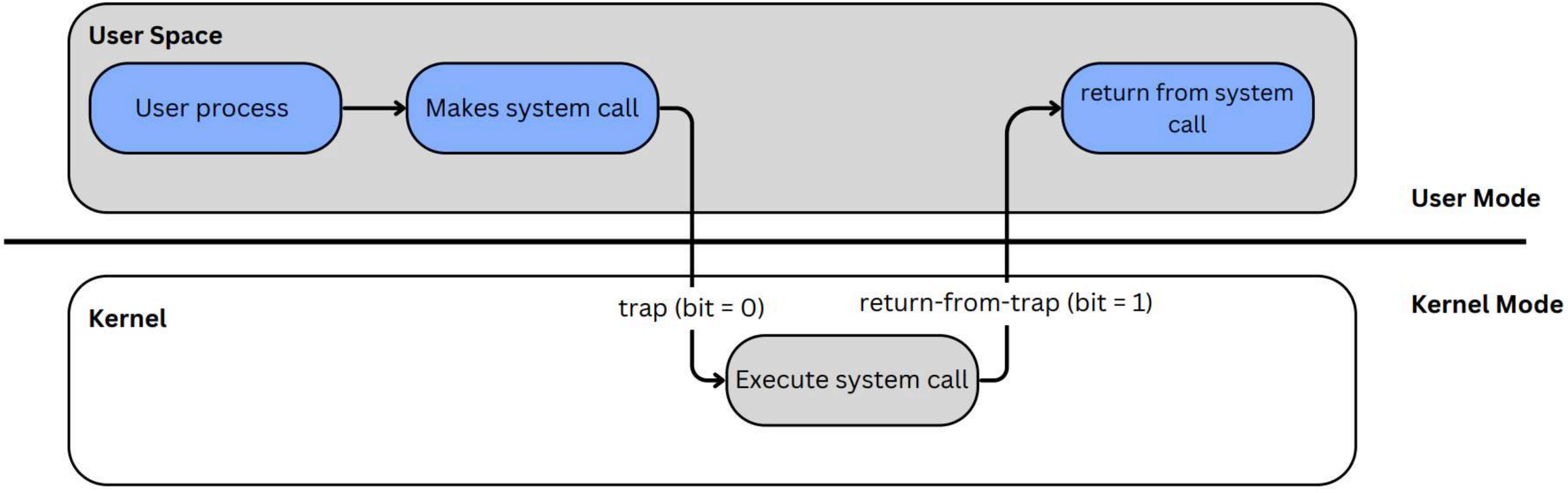
- Special kind of instruction to switch mode from user to kernel
- Allows system to perform what it wants
- When a system call is made, the trap instruction allows to jump into kernel
  - Raise the privilege mode to kernel mode
  - **Return-from-trap** instruction allows switch back to user mode
  -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

More about TRAP instruction

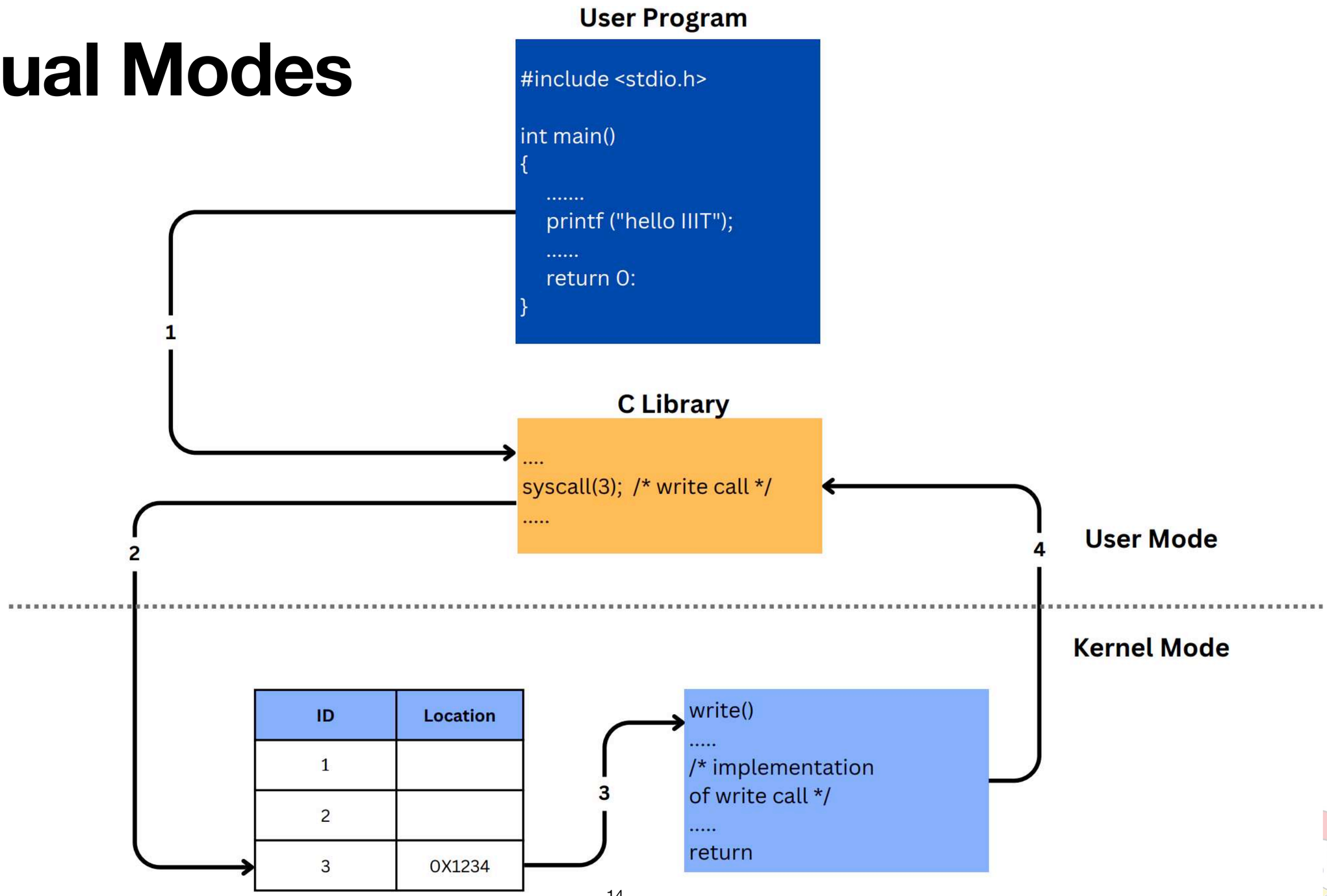
- During TRAP instruction execution
  - CPU to higher privilege level
  - Switch to Kernel Stack
  - Save context (old PC, registers) on Kernel Stack
  - Look up in IDT (Trap Table) and jump to trap handler function in OS code
  - Once in Kernel, privileged instructions can be performed
- Once done, 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

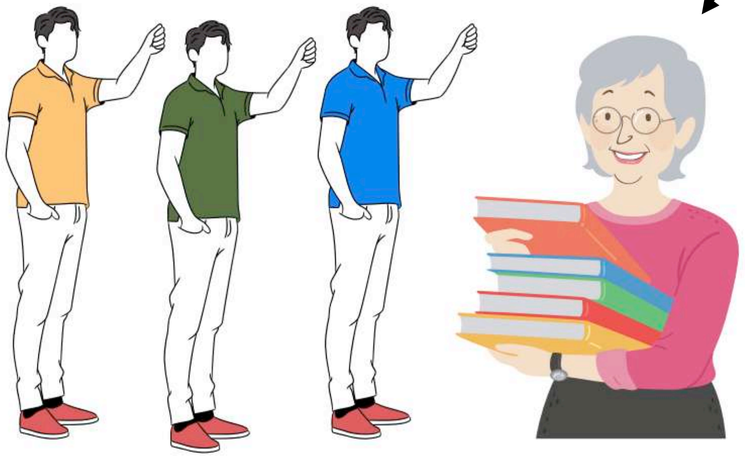
How to decide which process to run next?

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 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?

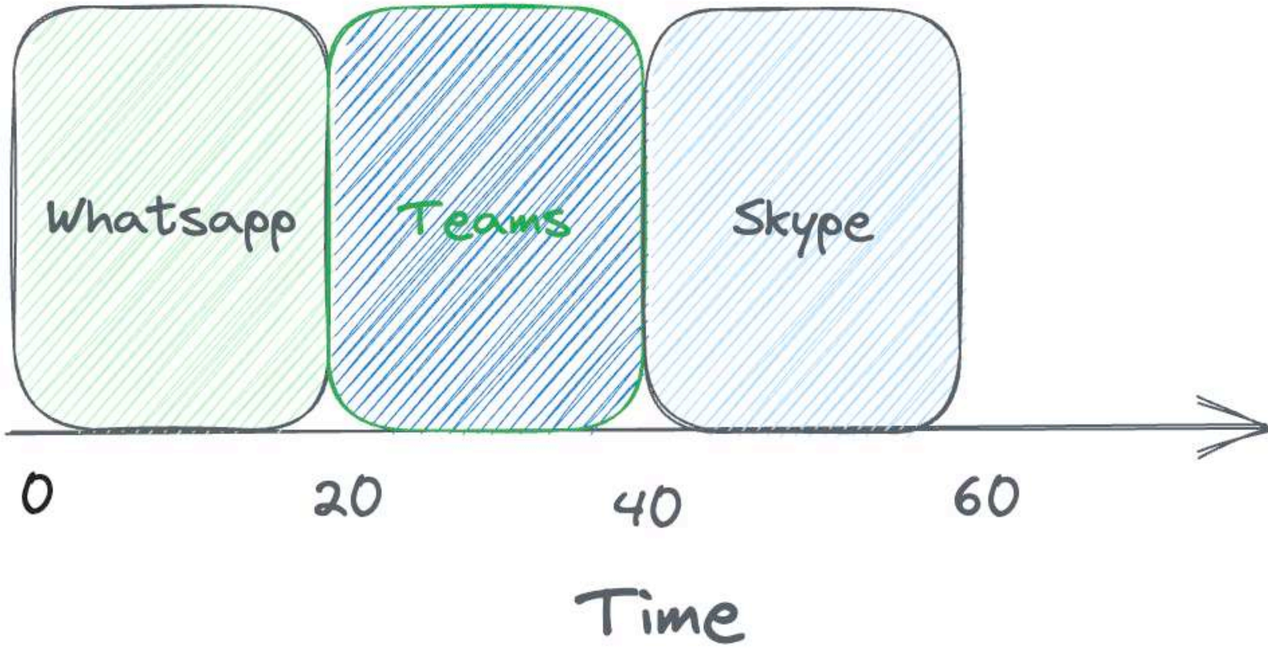
First Come First Serve Policy

- 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

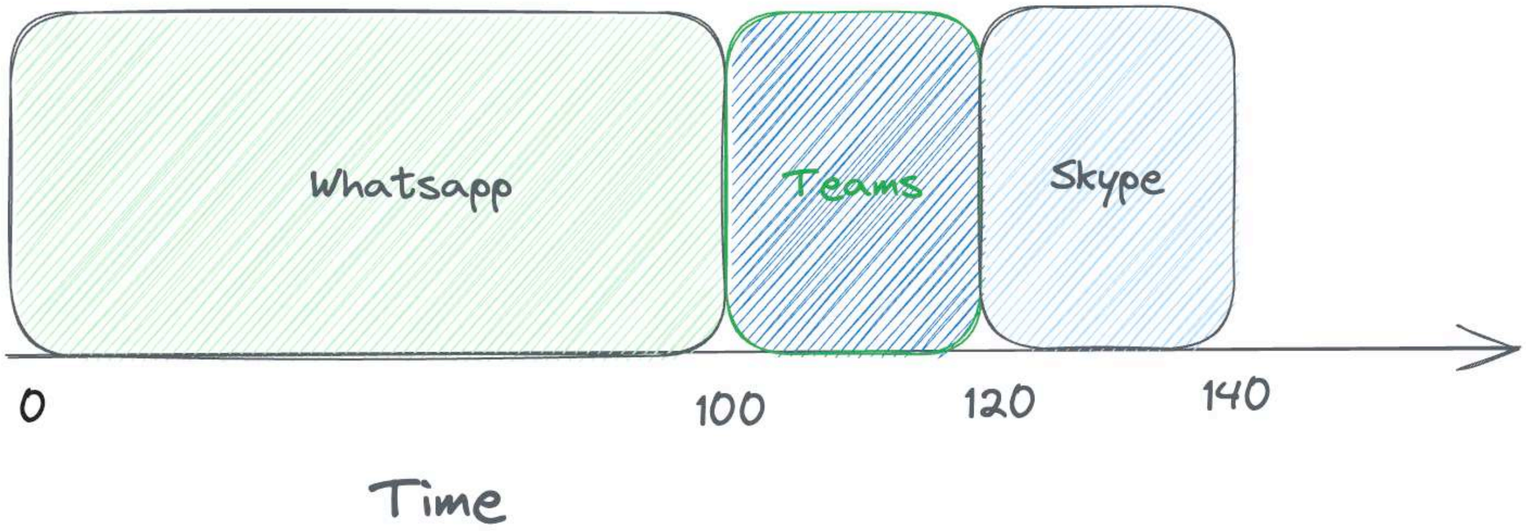


$$Avg(T_{turnaround}) = \frac{20 + 40 + 60}{3}$$

$$= 40$$

What if each job no longer runs for same time?

Relaxing assumption 2

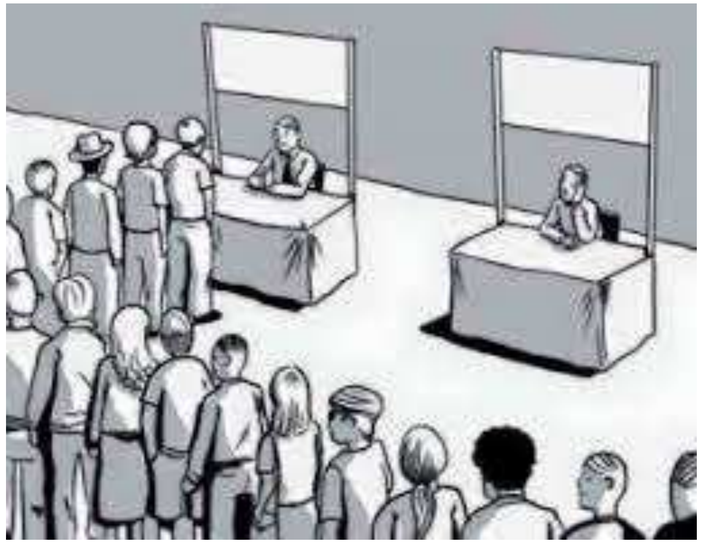


$$Avg(T_{turnaround}) = \frac{100 + 120 + 140}{3}$$

$$= 120$$

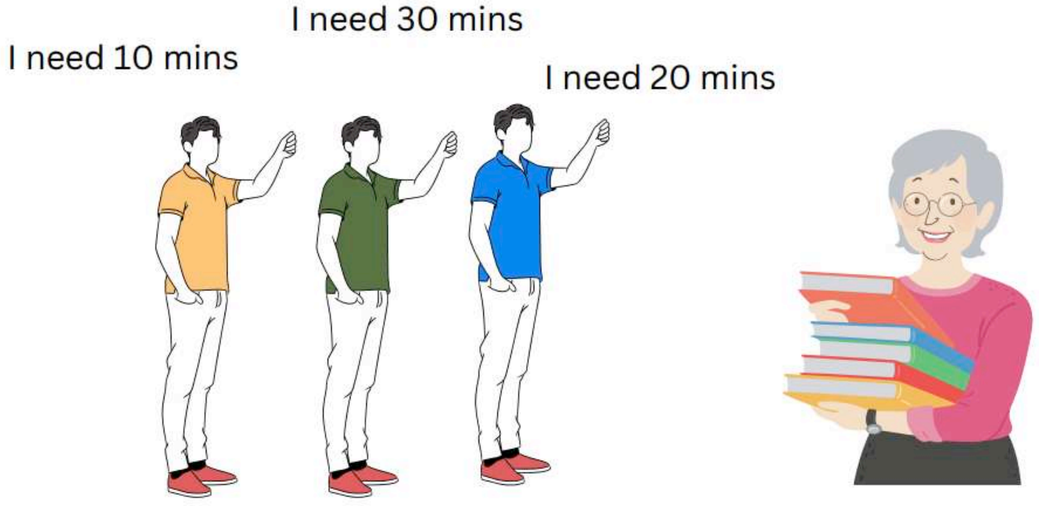
FCFS is not that great

Convoy Effect

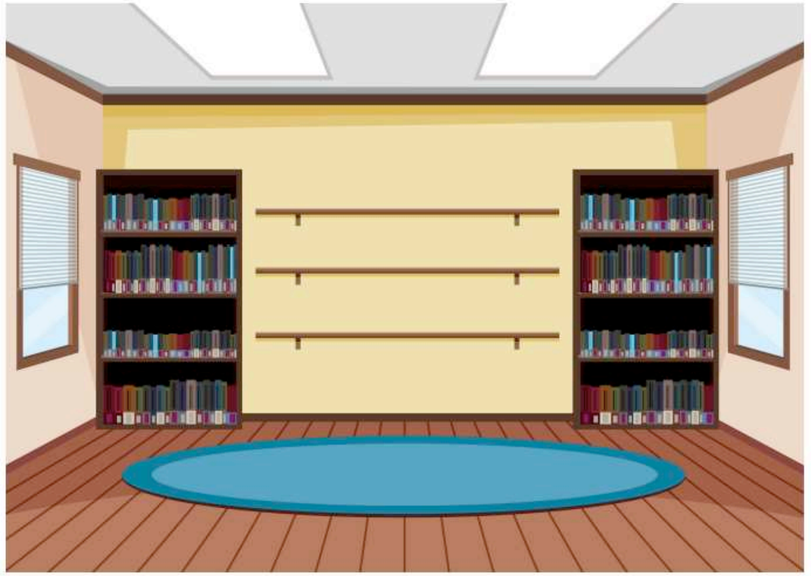


- Waiting time can go very high
  - Convoy effect!
  - Think about waiting in single line in grocery store where you just have one item to purchase

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

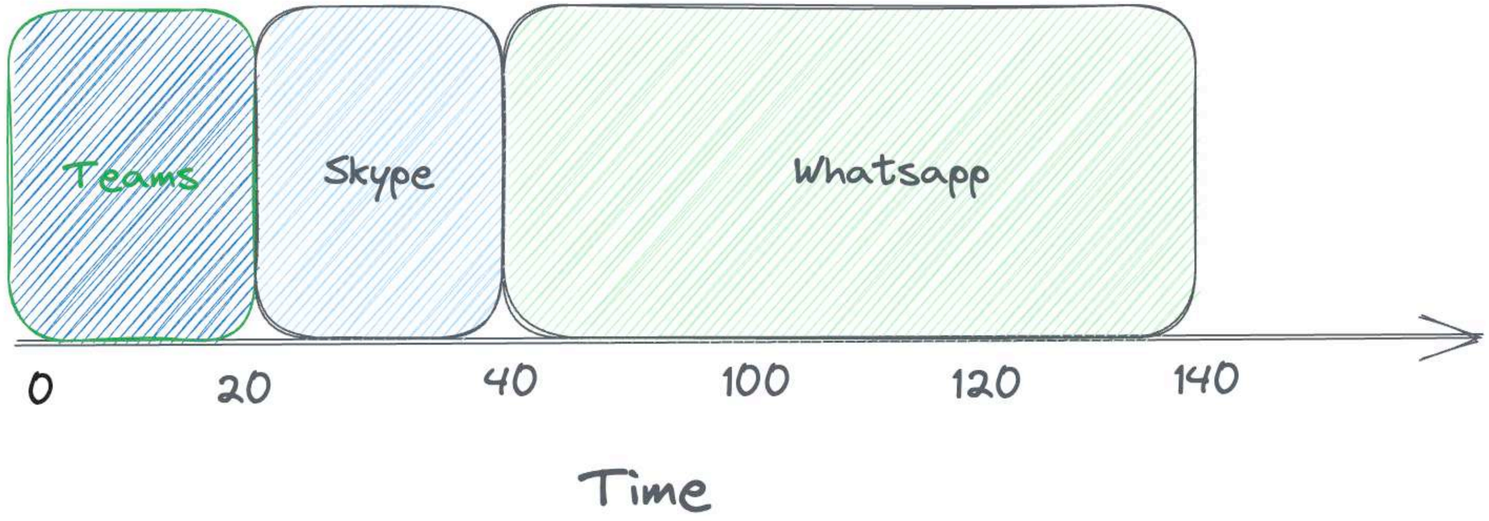
How to go about this?

Shortest Job First (SJF) Policy

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

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

$$= 66.3$$



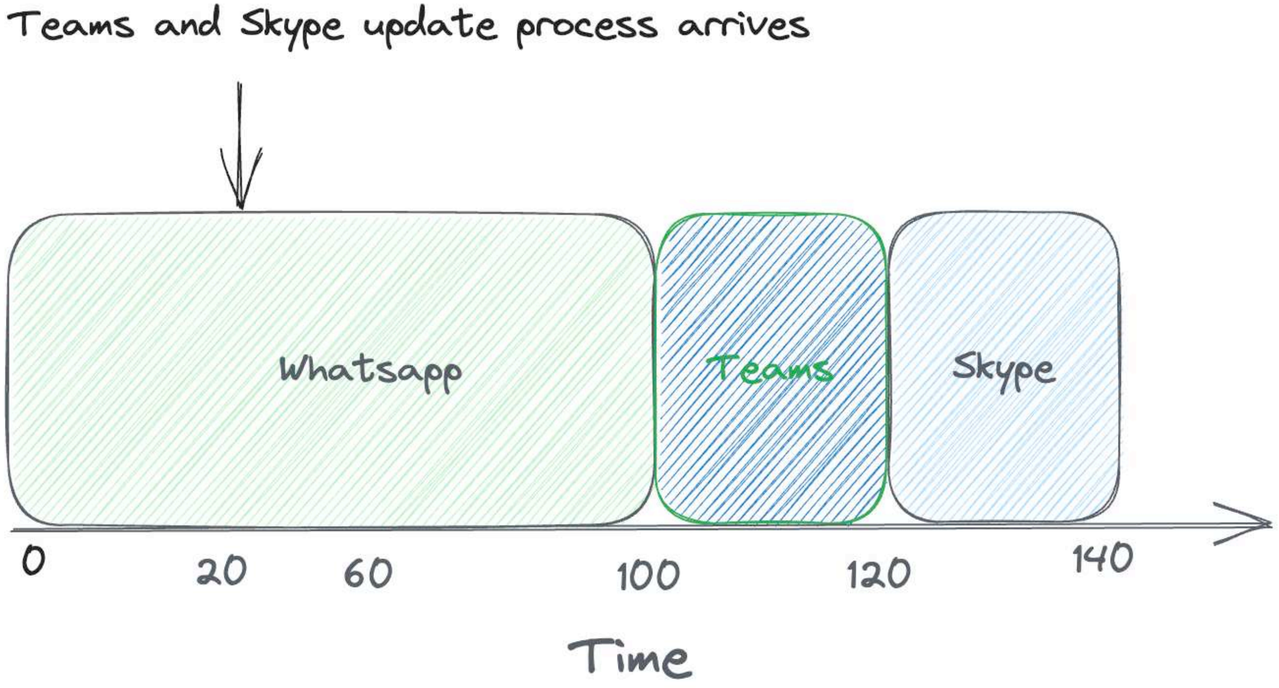
Is that a bit too unrealistic? - In reality jobs can arrive at any time

Shortest Job First (SJF) Policy

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

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

$$= 106.6$$

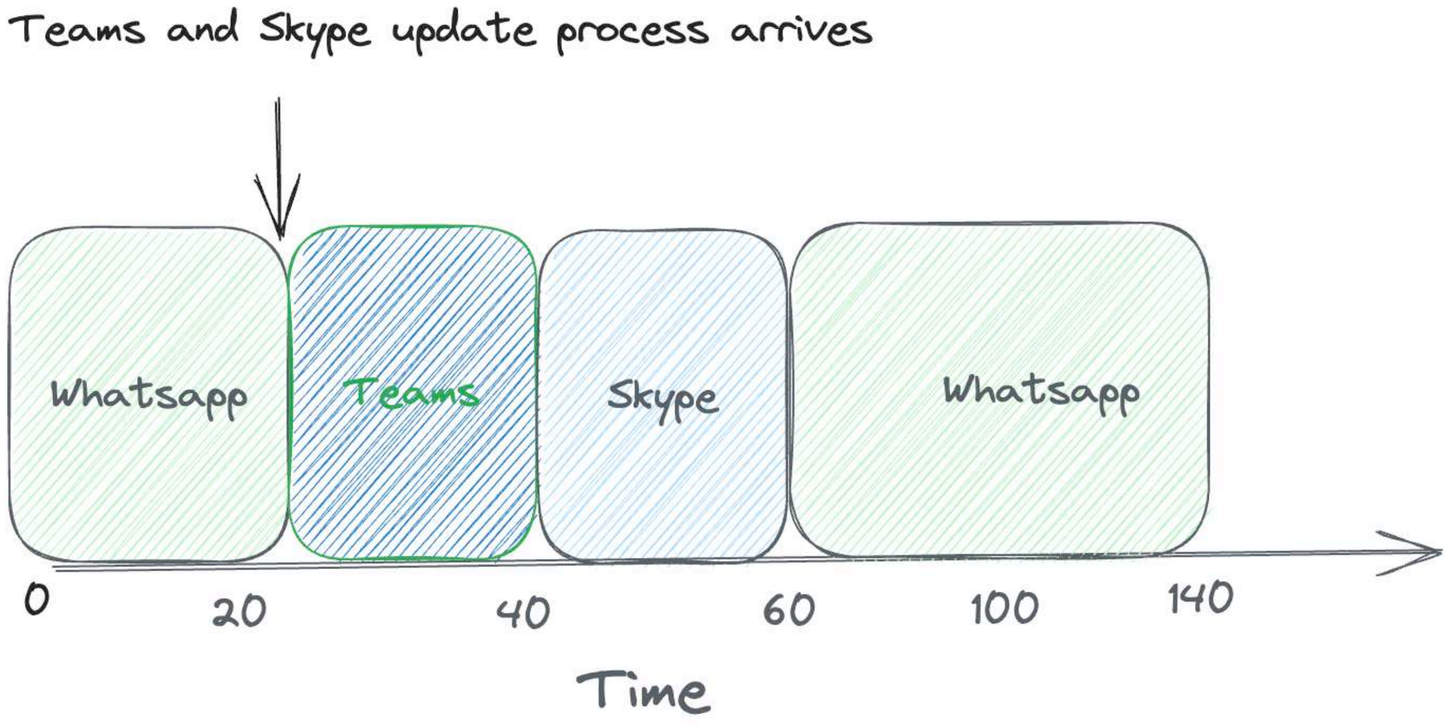


Even worst!! How to improve?

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

CS3.301 Operating Systems and Networks

Process Virtualisation - Policies (Scheduling) and Process Communication (Intro to networks)

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.
- Networking Fundamentals by Practical Networking (Youtube Channel)
- Other online sources which are duly cited

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?

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

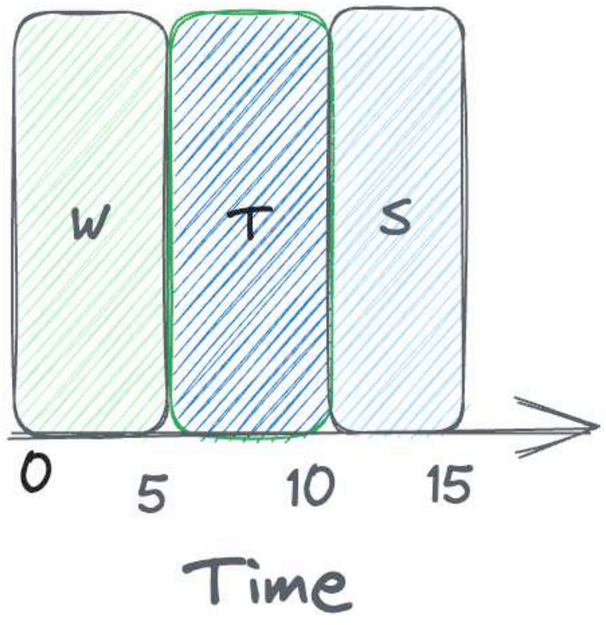
Round Robin Scheduling

- 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



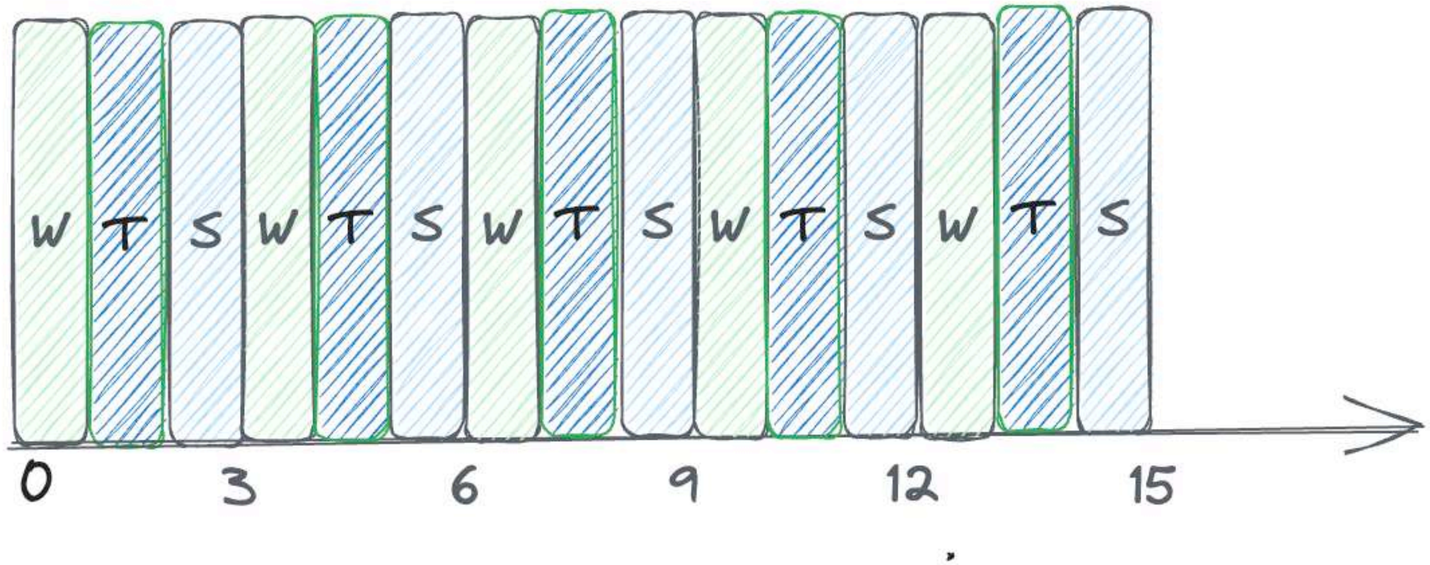
$$Avg(T_{turnaround}) = \frac{5 + 10 + 15}{3} = 10$$

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

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 - Given!
  - Fairness here comes at the cost of turnaround time
- 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

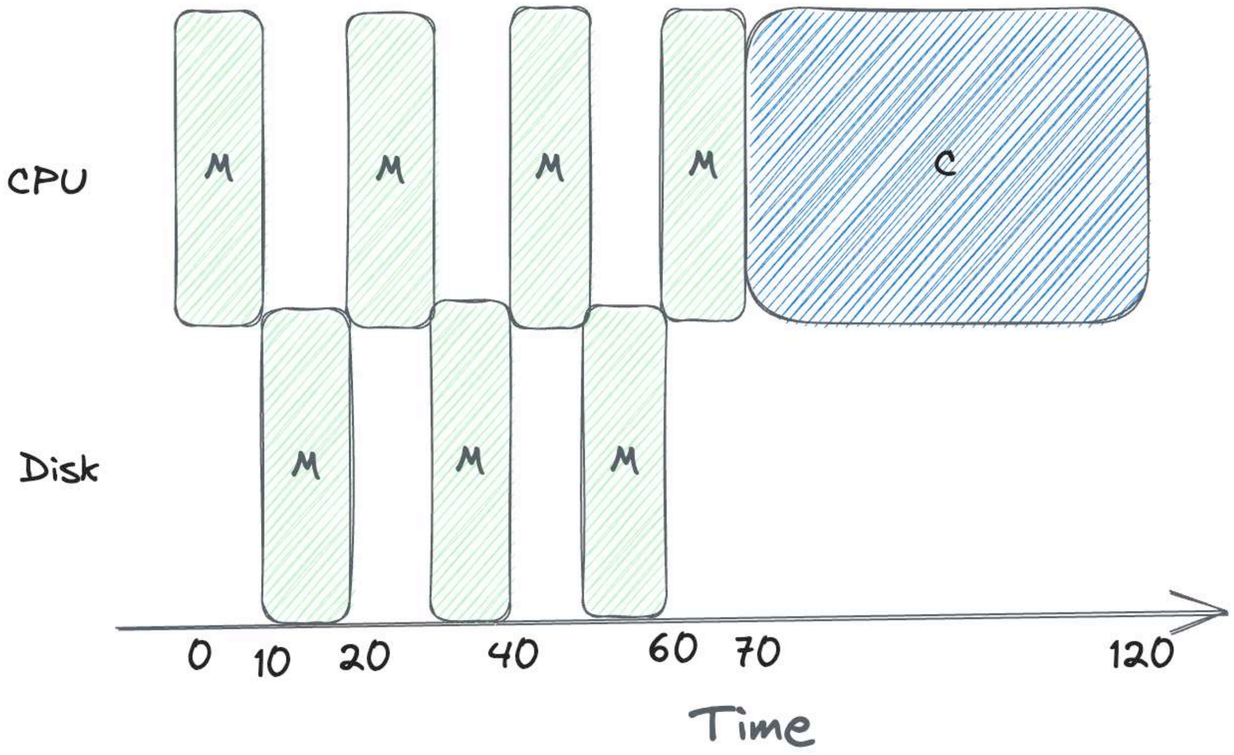
What can be done to consider I/O? Can we do RR Scheduling by considering I/O?

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

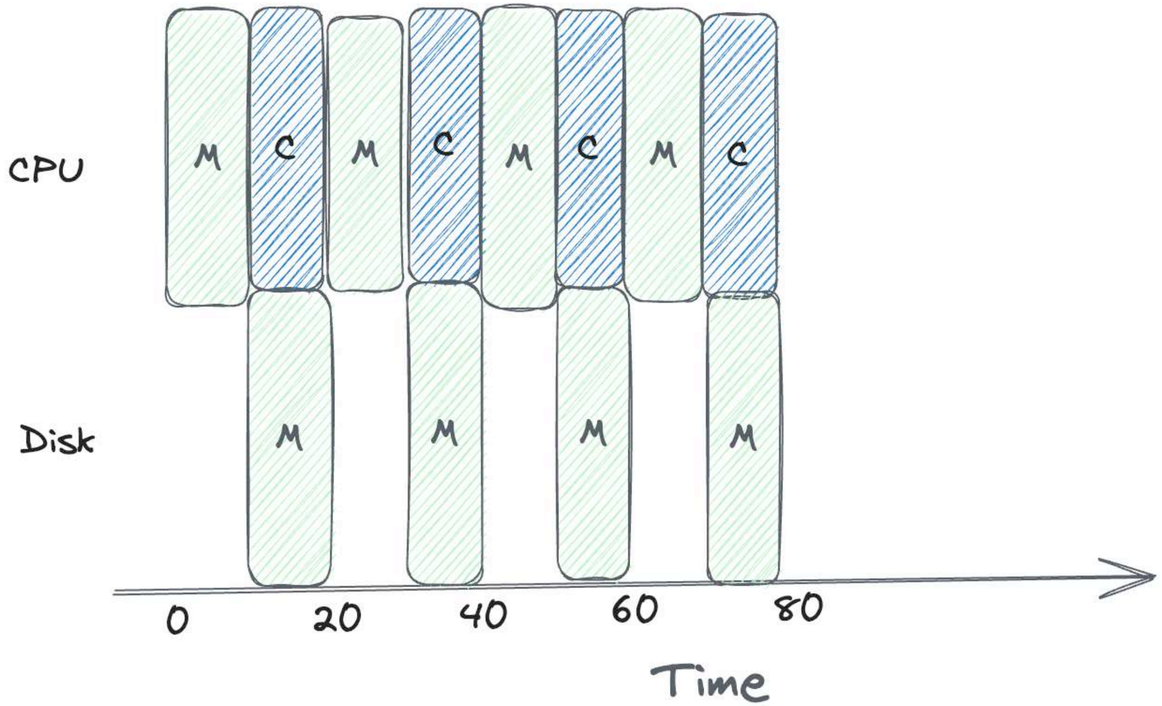


Can we do better?

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?

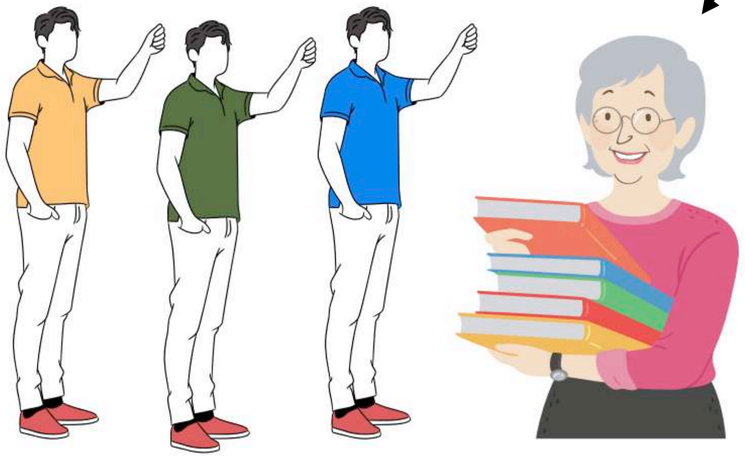
We may not know the length or expected time of completion of a job? — How to handle?

Lets go back to the example

How can Librarian take a guess?

The person is almost done with his reading and might come out soon (or time out!!)

Scheduler



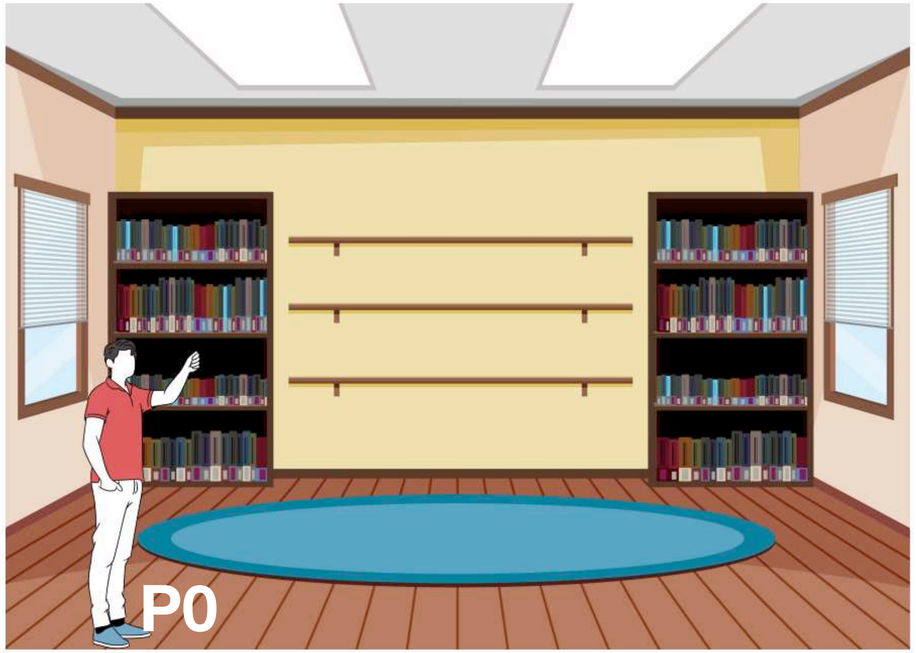
More users/visitors have requested to access the reference section.How to decide whom to sent next?

Why don’t we Prioritise?

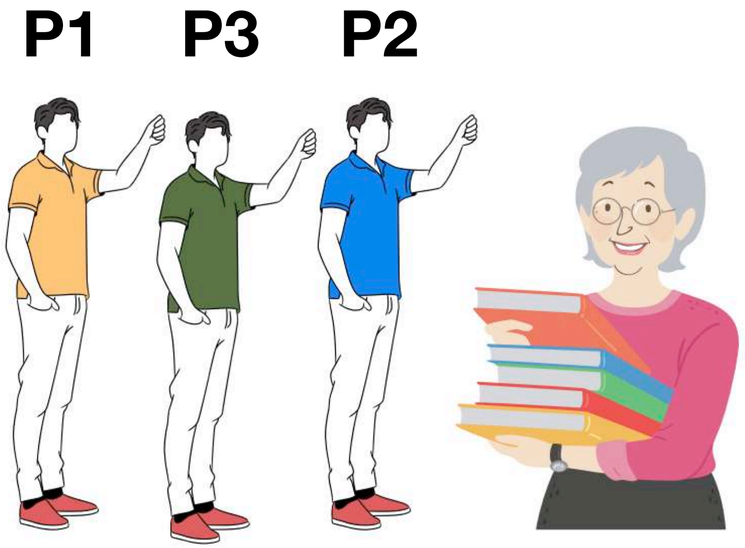
Source: https://imgur.com

Lets go back to the example

Introduce Priority - Give priority, Observe and Improve



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?

Multi Level Feedback Queues (MLFQ)

Two main features

- Reduce turn around times
  - Run shortest jobs first
- Reduce response time

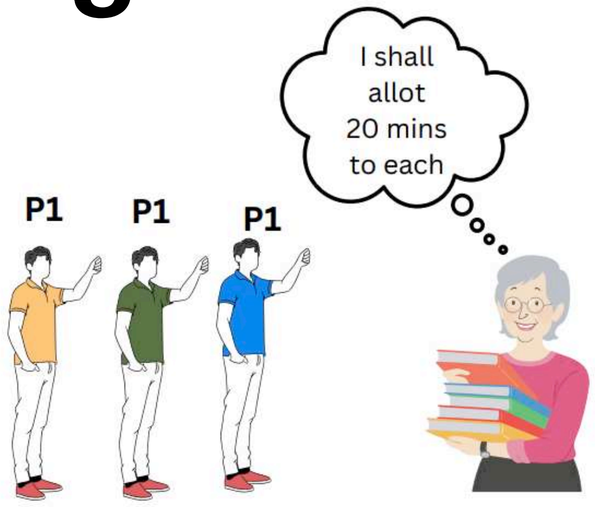
Can the policy learn continously to optimise response time and turnaround time?

Constraint: No apriori knowledge of the job length!

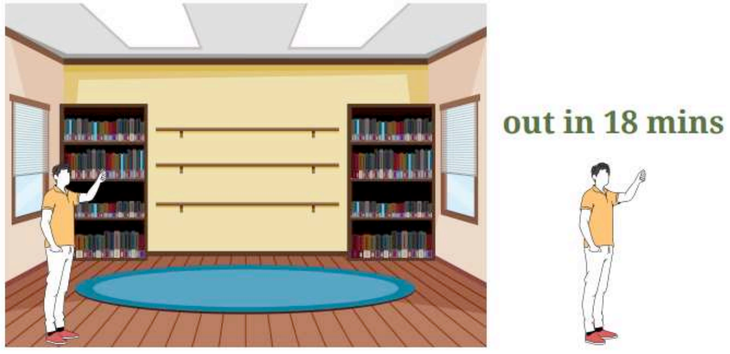
MLFQ: Basic Rules

- Use **n** number of distinct queues
  - Each queue has a different priority level
- Use priority to decide which job should be run at a given time
  - A job with a higher priority => job on a higher queue
- **Key idea:**
  - Scheduler sets priority to different jobs
  - Keep updating the priority based on observed behaviour

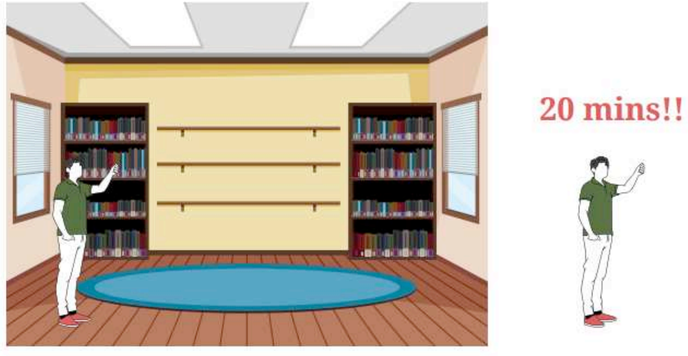
Going back to the example



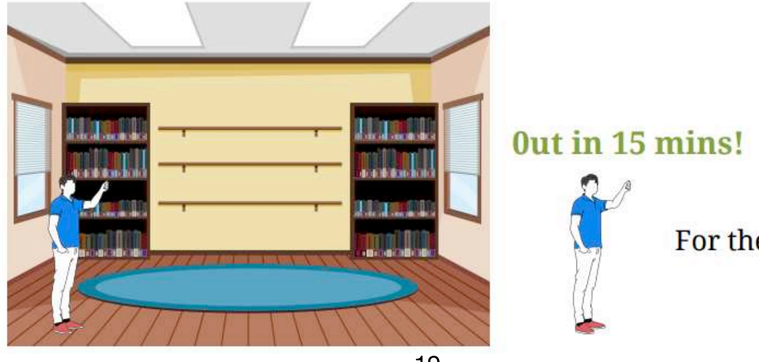
Put all the visitors initially on the same priority
Observe how they behave and then decide!

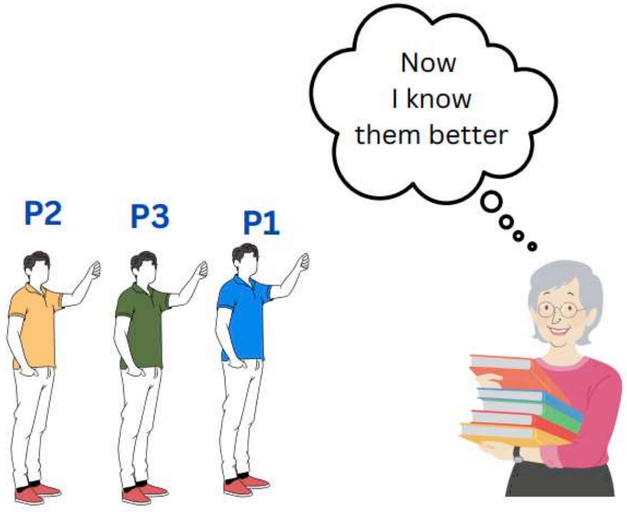


First person gets the chance!



Next person goes in





For the next visit, priorities have been updated

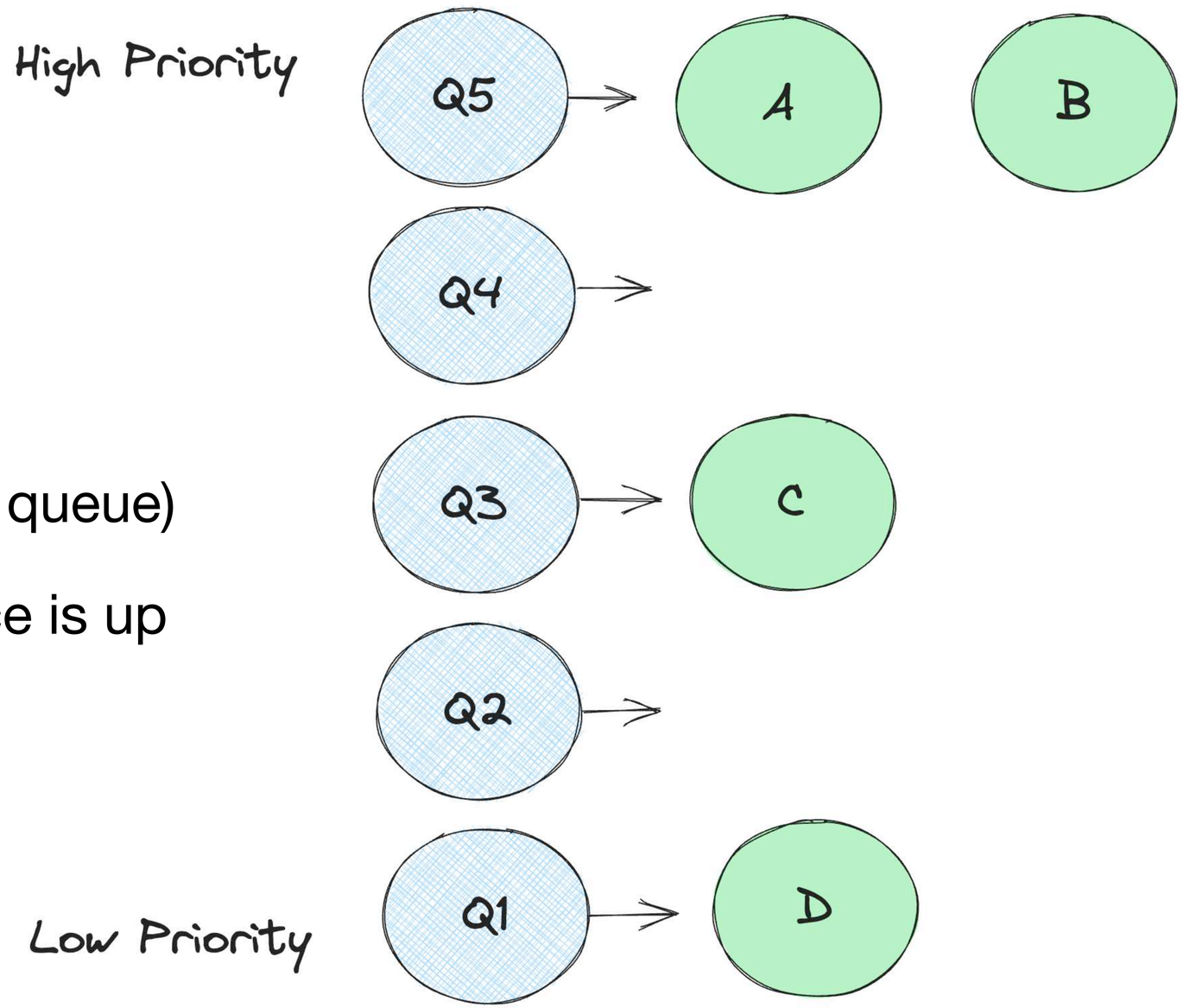
MLFQ

- Jobs that keep giving back the CPU - interactive jobs (higher priority)
- Jobs that uses CPU for more time - Reduce priority
- Learn about the processes as they run and **predict future** (Not using AI)
- **Two basic rules:**
    1. If priority (A) > Priority (B), A runs
    2. If priority (A) = Priority (B), A&B run in Round Robin

MLFQ: Visualisation

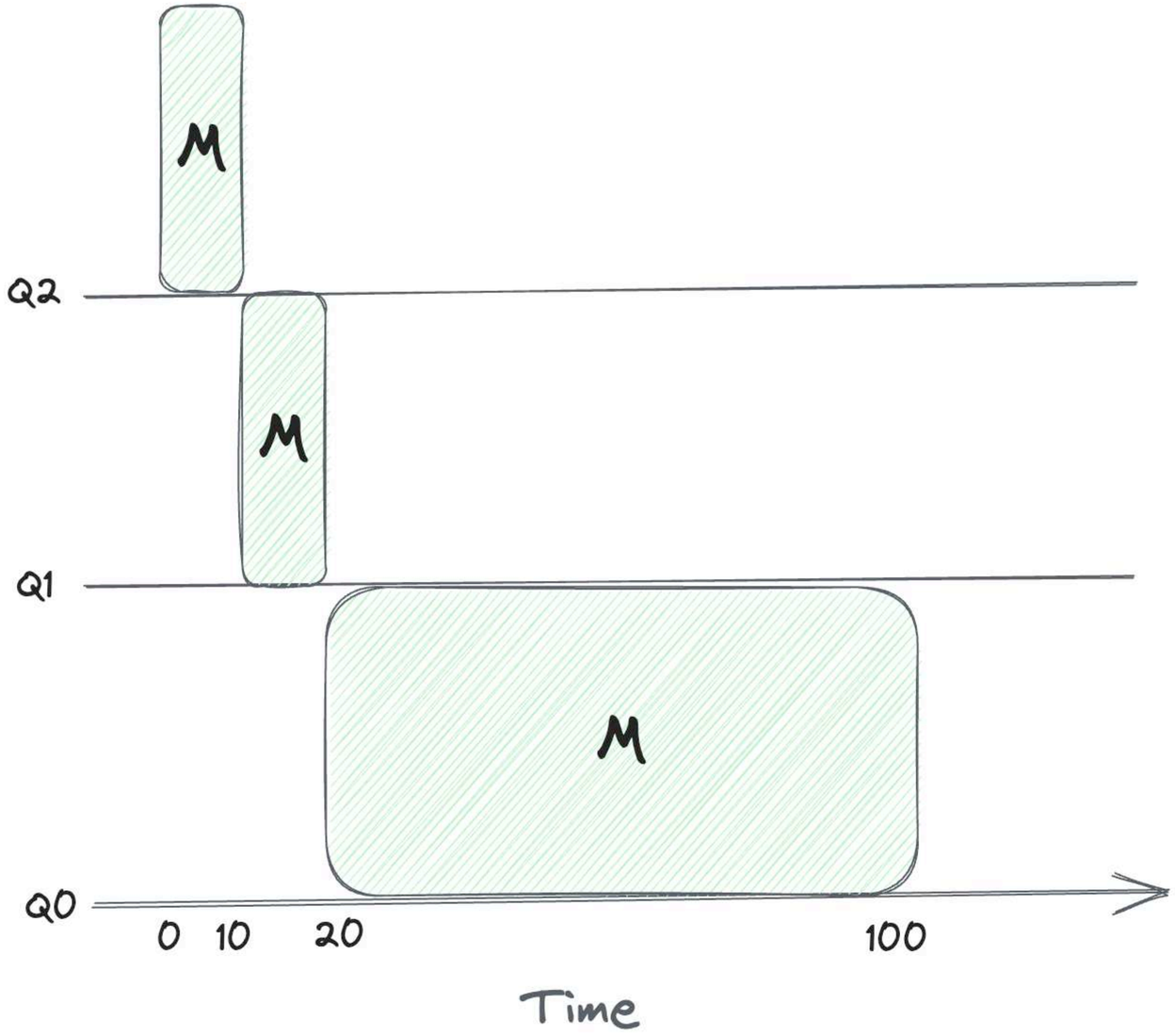
Five queues and 4 jobs

- Use notion of time slices
- When job enters, high priority
- If job uses up the entire time slice
  - Reduce priority (move down one queue)
- If job gives up CPU before time slice is up
  - Keep it at the same priority level



Example: Single Long Running Job

- Three queues
- Priority: Q2 > Q1 > Q0
- Single job M enters at t=0
- Time slice = 10 seconds
- After running for 10 seconds, priority is lowered

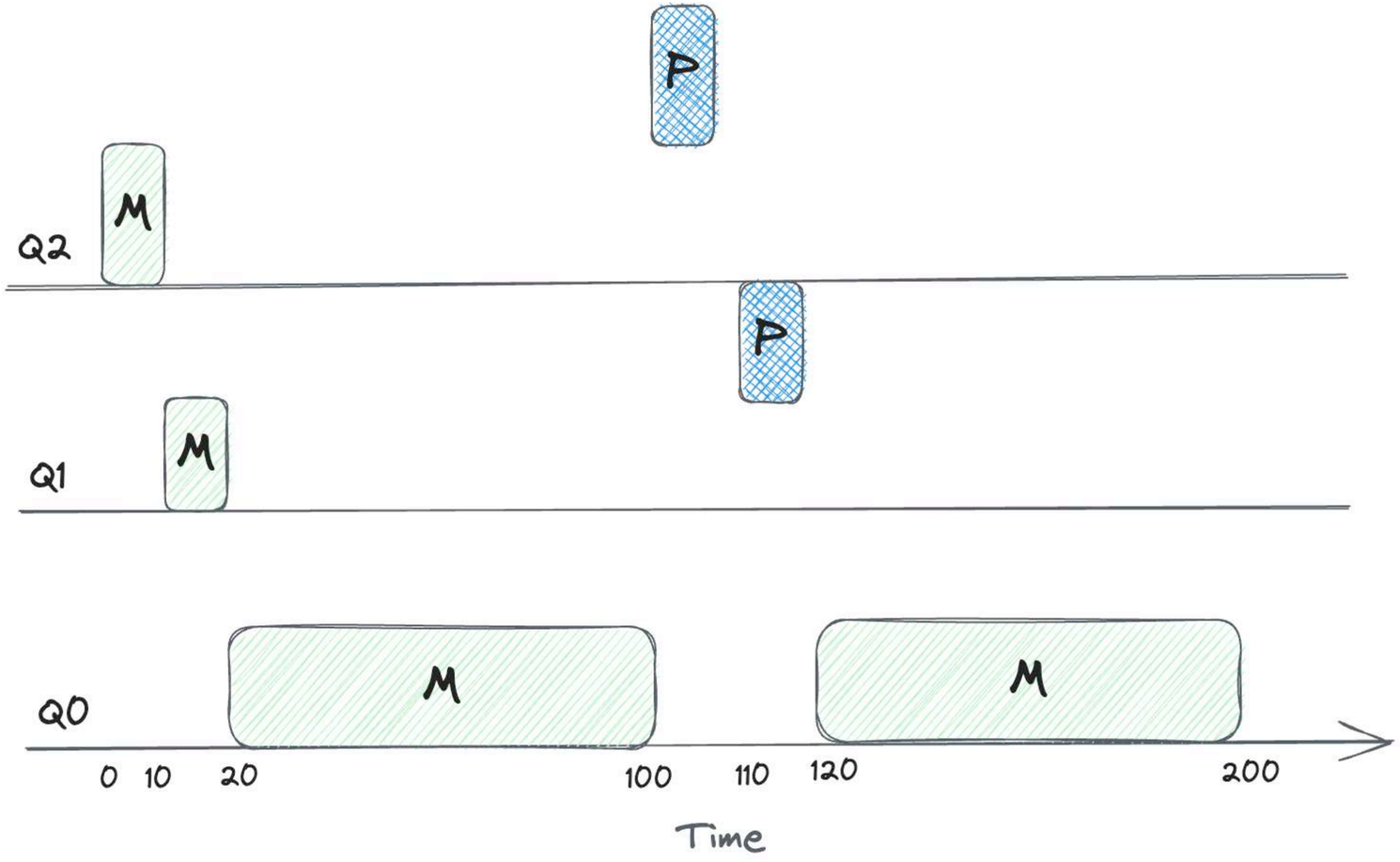


Example: Short job enters

- **P (interactive job)**
  - Enters at t=100
  - Runs for 10 second
  - Goes to the second queue
- Incoming job is considered

short => placed in higher queue

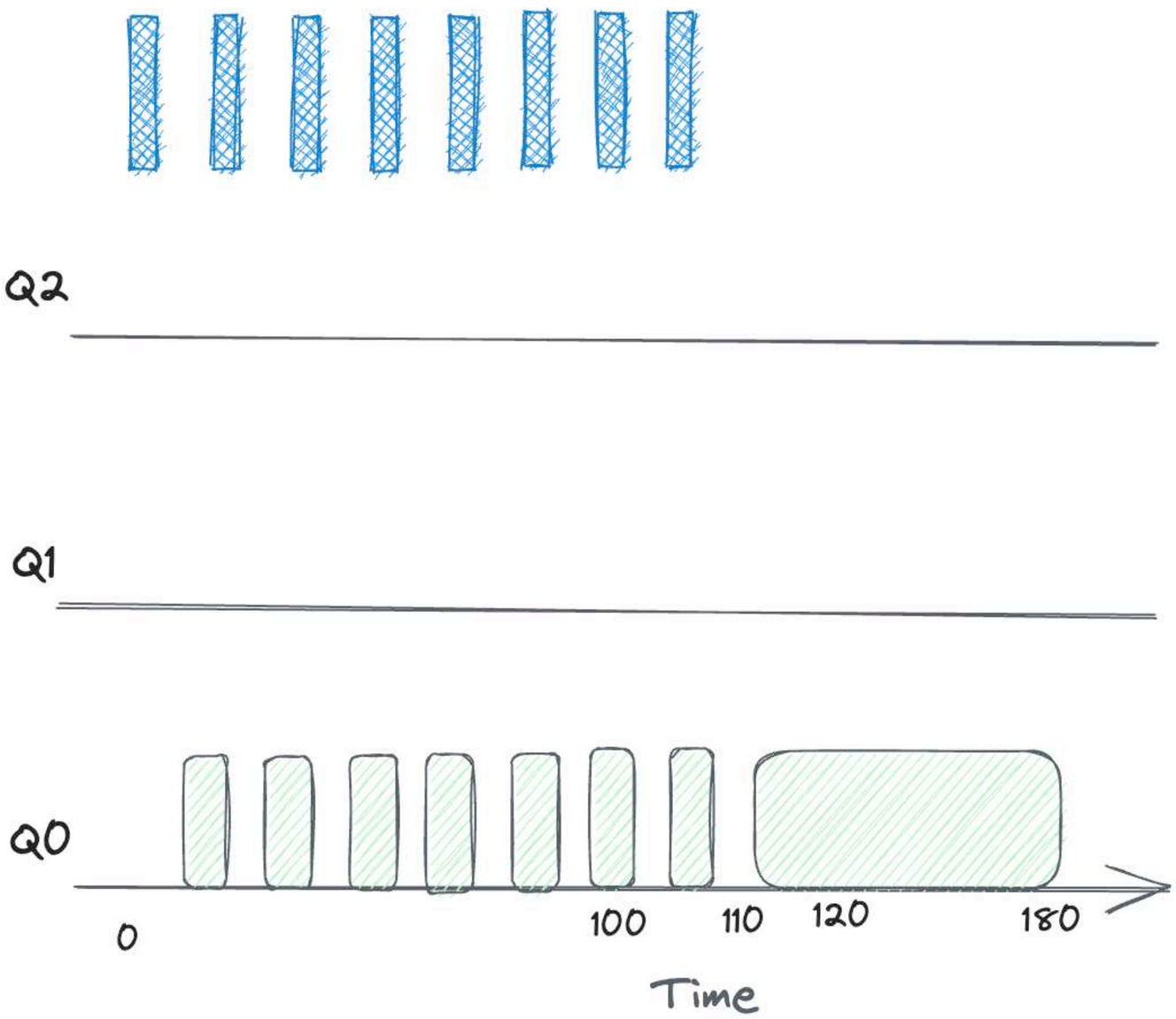
- If its short, it will keep running
  - Else moves down



Incorporating I/O

- Highly interactive job => relinquishes CPU faster
  - Priority is kept at same level
- Long term job can take more time and execute in lower queue

What could be the challenges?



What are the challenges with previous model?

It can lead to two major challenges

- **Starvation**
  - Too many interactive jobs will keep consuming CPU
  - Long running job will never get any CPU - **Starve!!**
- **Gaming of Scheduler**
  - The process can trick the scheduler into giving more than fair share - How?
  - **Idea:** Give an I/O request and relinquish the CPU before time slice is over
    - Priority does not change!! (Monopolise the CPU)

Another Challenge to Consider

- Program may also change behaviour over time
  - Suddenly there may be more CPU intensive phases
  - There may be also phases of interactiveness
  - Eg: Some long numerical computation followed by interactiveness

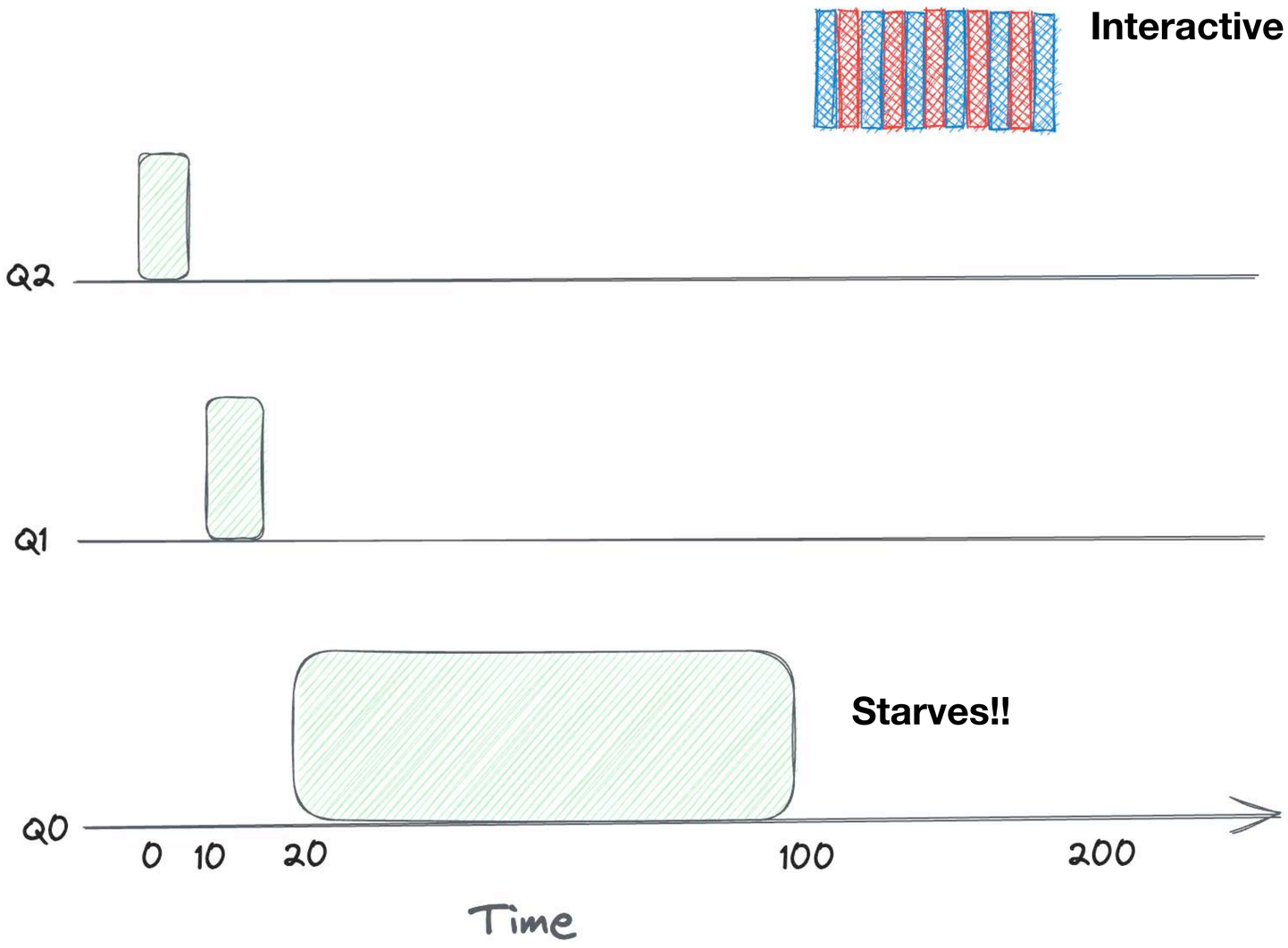
Can you think of some way to handle this?

Priority Boost

- Periodically boost the priority of all the jobs
  - This can **prevent starvation**
- **Rule:** After a time interval ***S***, move all jobs to top most queue
- **Provides two key guarantees**
  - Processes are guaranteed not to starve
  - If CPU bound has become interactive, scheduler will give it chances
    - Thanks to the periodic priority updates!

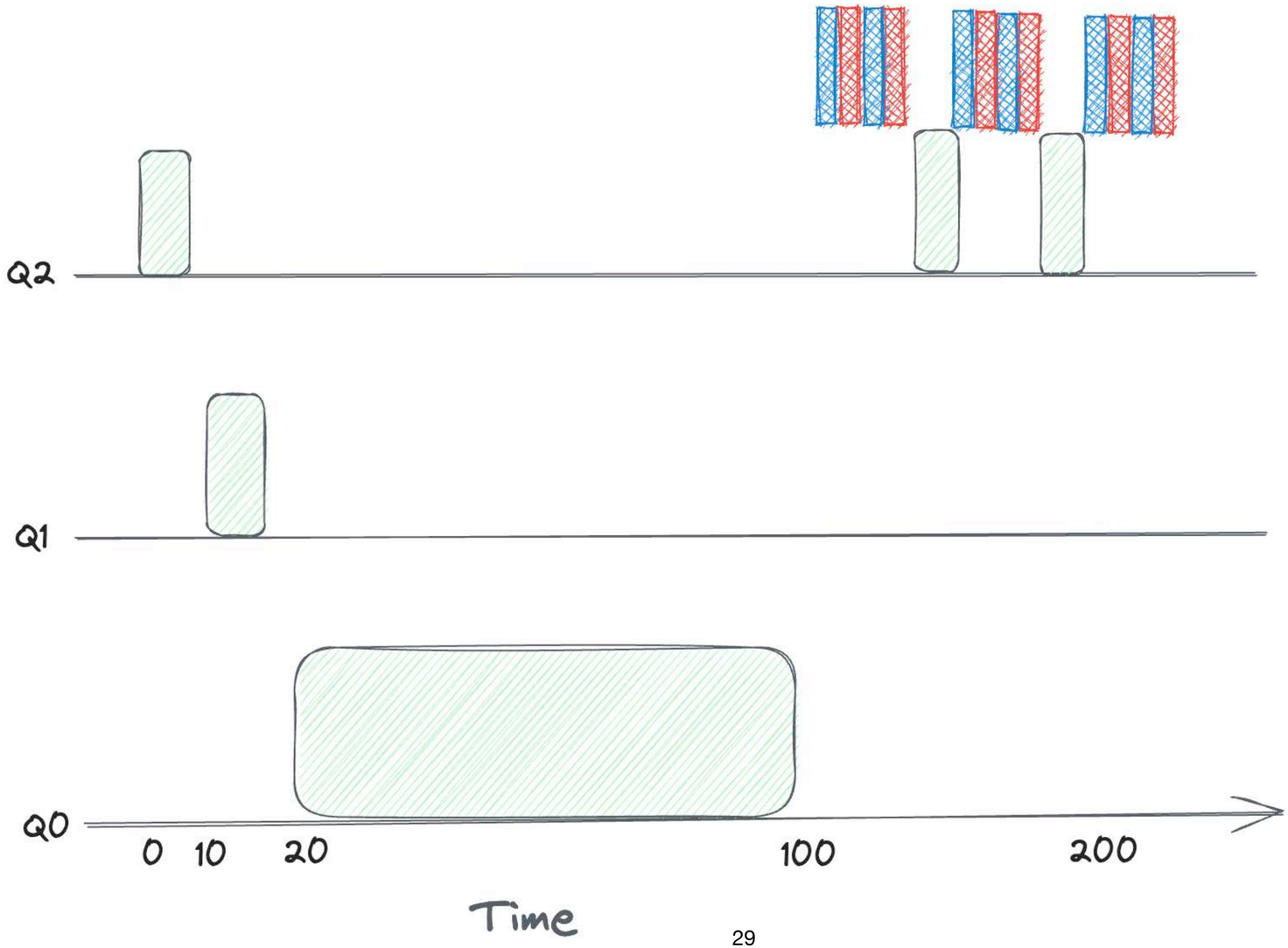
Case 1

Non priority Boost Scenario



Case 2

With Priority Boost (S = 50)



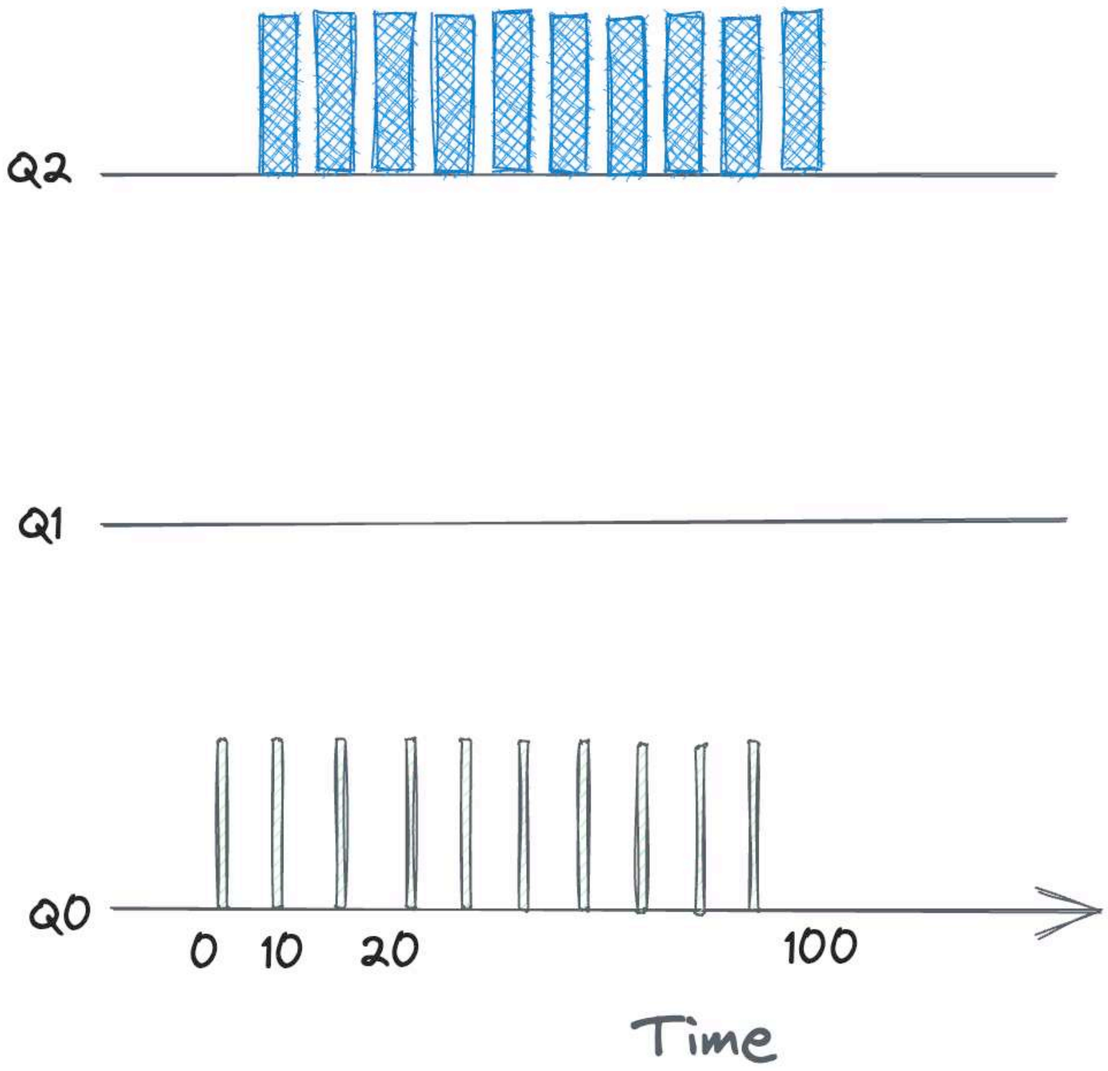
One additional Challenge

- How to determine the value of **S**?
  - If S is very small, interactive jobs may not get proper share of CPU
  - If S is too high, long running jobs could starve
- Tricky part is to come up with a value of S
  - Voo-doo constants - Named by John Ousterhout
- What about gaming the scheduler? - S by itself cannot solve it!

Better Accountability

How to prevent gaming of the scheduler?

- Introduce one more rule
  - Once a job uses its time allotment, it needs to be moved down
  - No consideration if the job has relinquished CPU ahead of time slice
  - Prevent the gaming since no matter what, priority reduces



Tuning MLFQ

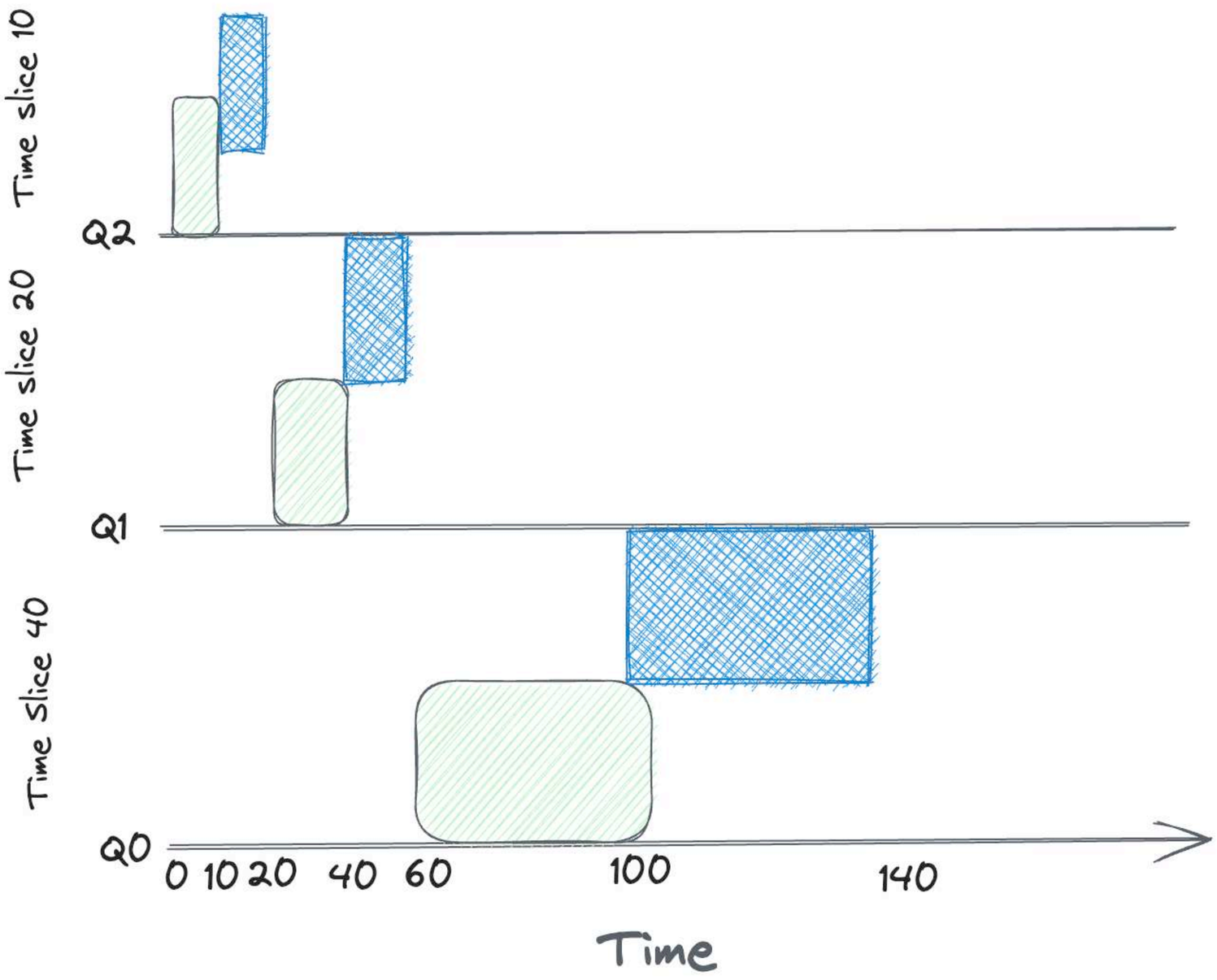
- How to parametrise the scheduler?
  - How many queues?
  - What should be the time slice?
  - How often the priority boost needs to be done?
- **High priority queues:** Interactive jobs with shorter time slices (10 ms)
- **Low priority queues:** CPU bound jobs with longer time slices (100 ms or less)

Example Scenario

- Different queues
- Low priority and high

Priority queues

- Each queue has a different slice interval
- Based on scenario, S changes (boost interval)



Solaris MLFQ

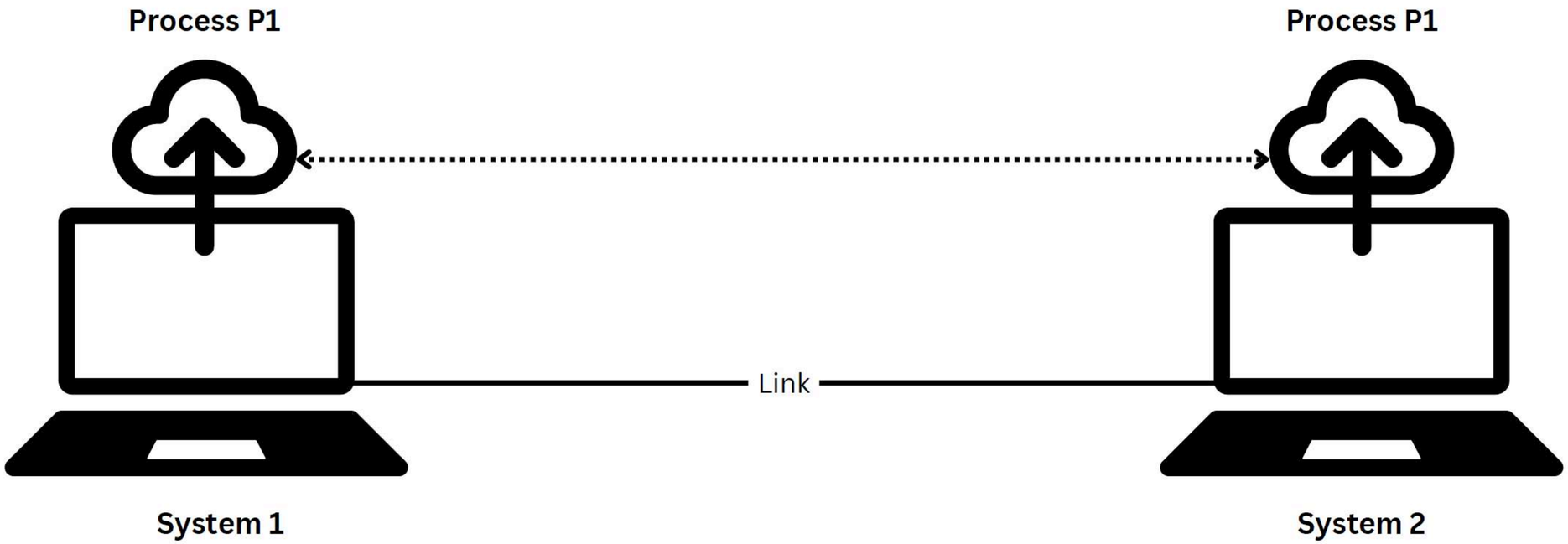
- Tables to configure:
  - How long should be the time slice?
  - How often to boost?
- 60 queues
  - Time slice length from 20ms to few 100 milliseconds
  - Priority boosted every 1 second
- Free BSD scheduler uses formula to calculate priority

MLFQ: Summing Up

- Five key rules are used by MLFQ:
  - If P(A) > P(B), A runs (B not)
  - If P(A) >= P(B), A & B runs in Round Robin using time slice of queue
  - When job enters, its placed in the highest priority
  - Once job uses its time allotment at a given level => priority is reduced
  - After a period, S move all jobs to top most priority queue
- BSD Unix derivatives, Windows NT and Solaris use a form of MLFQ in their basic scheduler

What about processes in different machines?

How can they communicate?



How does message/data from P1 in System 1 reach P1 in System 2?

What is the role of the OS in this and how does it contribute to the effectiveness?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Networking - Introduction (Sockets and Networking Layers)

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

Acknowledgement

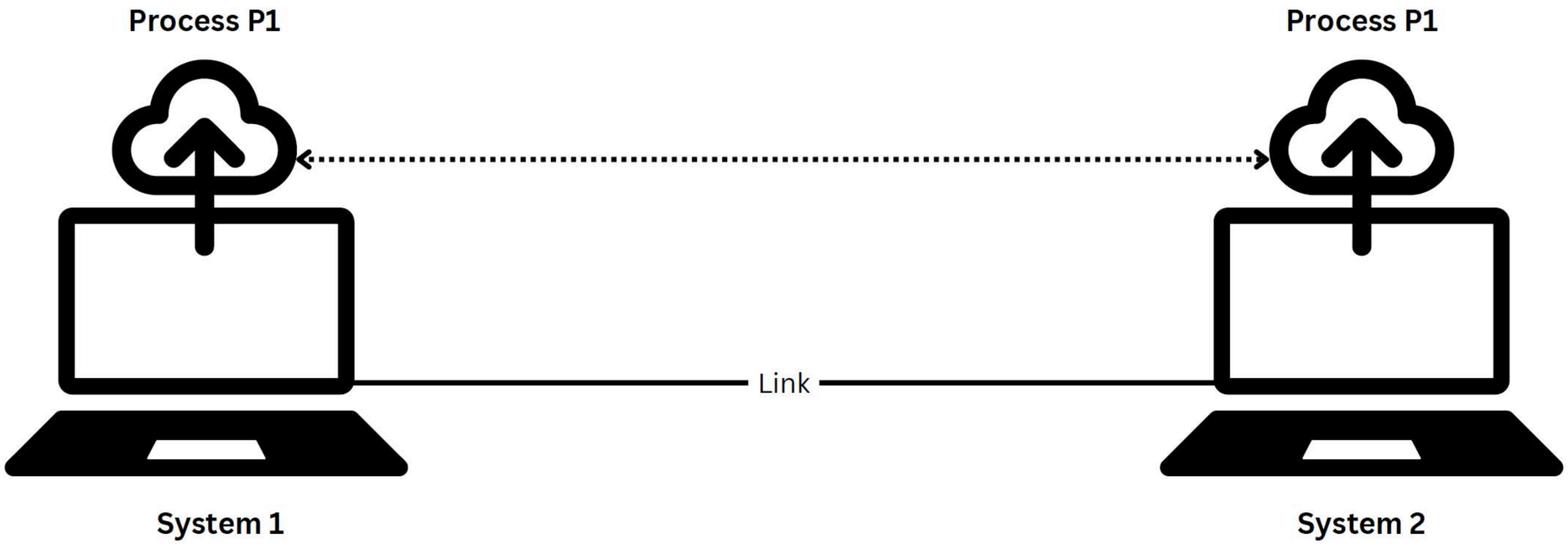
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

What about processes in different machines?

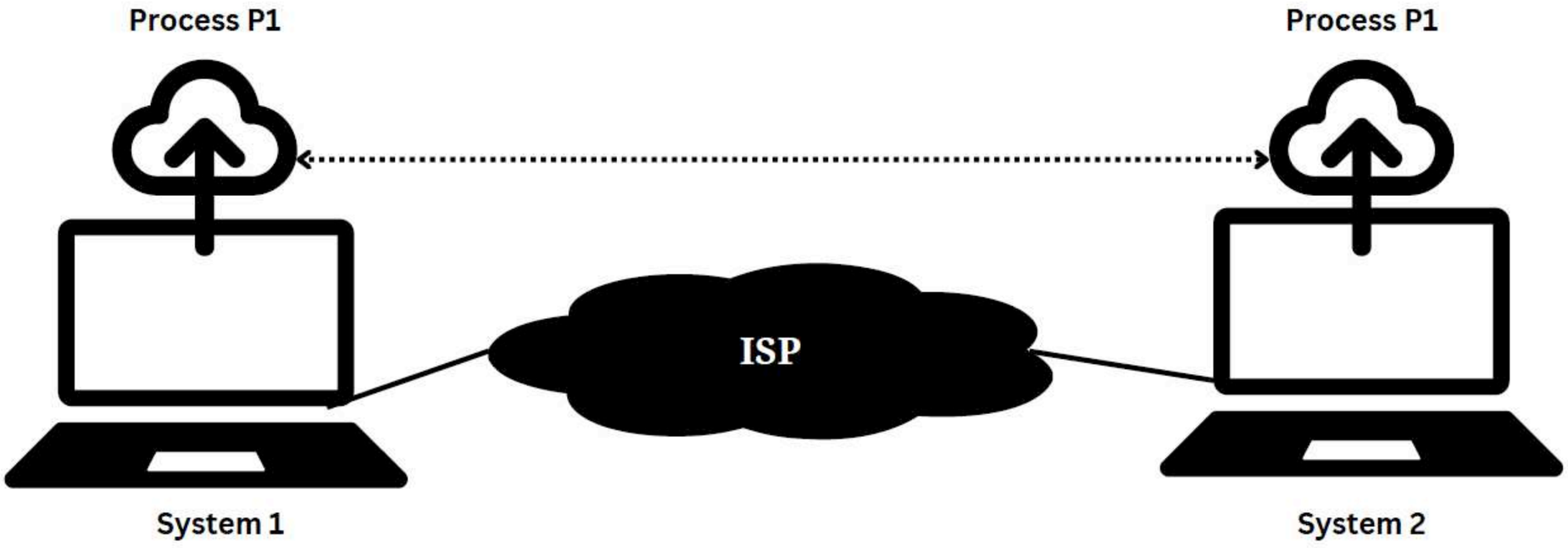
How can they communicate?



How does message/data from P1 in System 1 reach P1 in System 2?

What is the role of the OS in this and how does it contribute to the effectiveness?

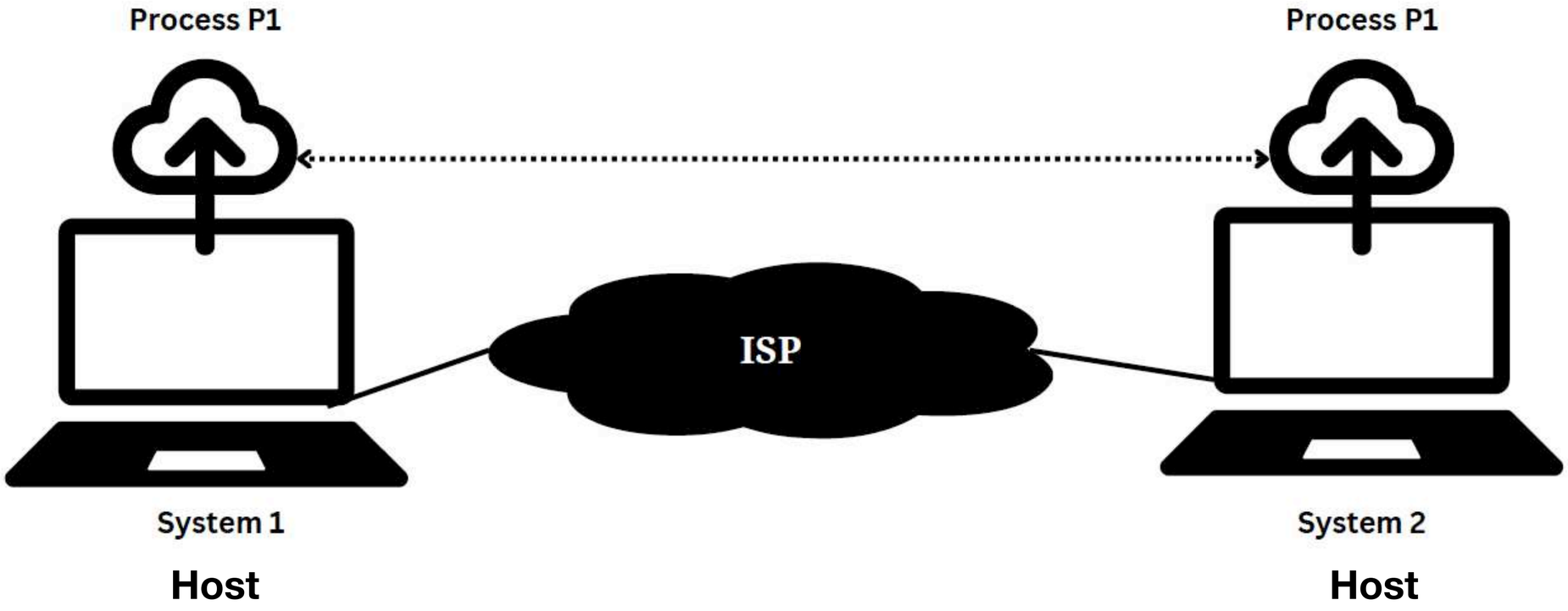
Let us expand it



- Between the process and network there needs to be an interface
- Between the network components there needs to be some interface

Network Components

Host



- Any device that send or receive traffic: Computer, laptop, smartwatch, phone, etc
- Host can be client or server
- Servers can sometime be clients too

How does host send data?

IP Address

- Host needs address to send the data
  - This address is known as **IP address**
  - When client sends data it provides both source IP and destination IP
  - IP addresses are 32 bits. Each bit can be 0 or 1
  - Total of 4 octets. Each octet ranges from **0-255 (8 bits)**
  - IP addresses are usually hierarchically assigned
  - **Eg:** 192.168.1.1

Sometimes we need more signal Strength

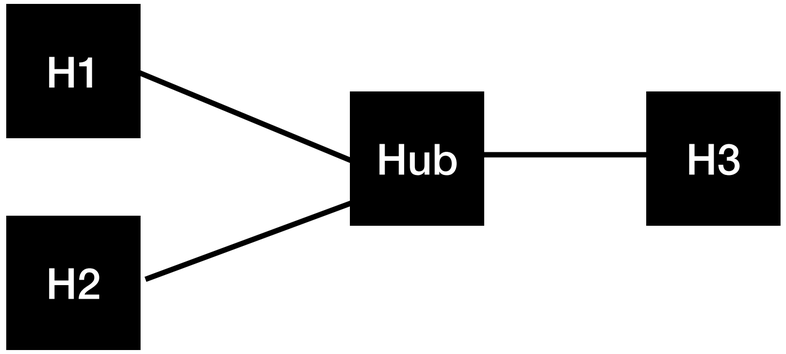
Repeater

- Repeater allows regeneration of signals for long distance communication
- Think of wifi in home
  - Signal strength may not be there
  - Repeater might be needed to transmit to longer ranges
- But we cannot just connect one host to another - **Not Scalable!**

Hubs and Bridges

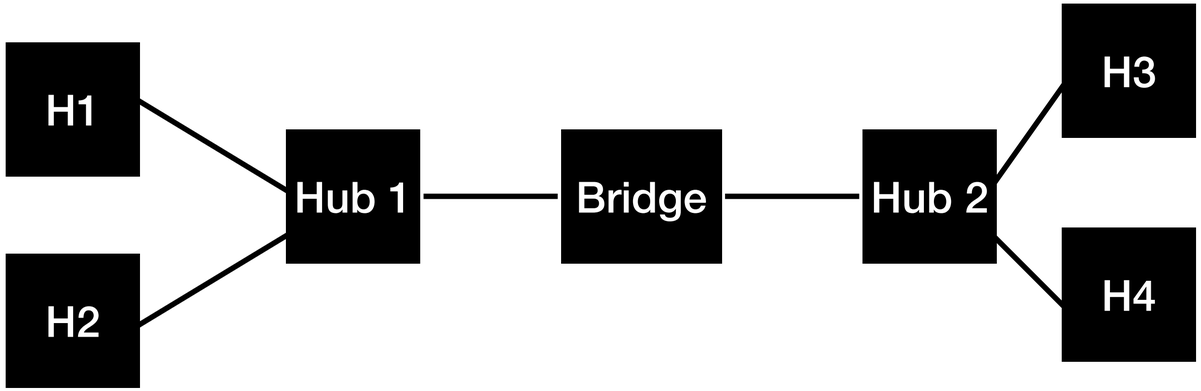
Hubs

- Multi-port repeaters
- Key issue: Everyone receives everyones data



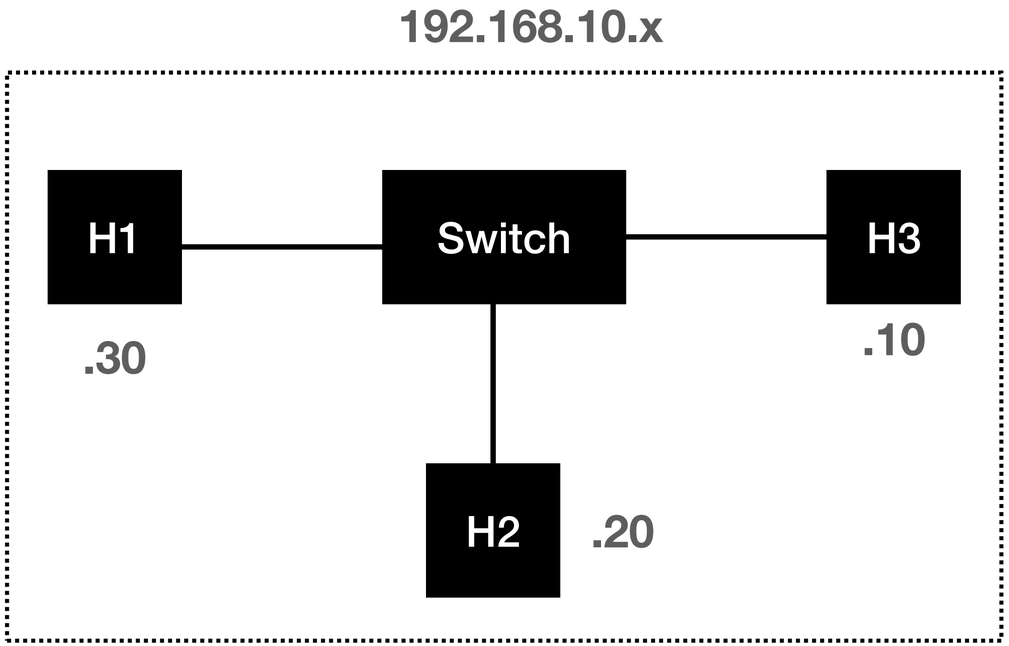
Bridges

- Sit between two hubs
- They have only two ports
- They learn which hosts are on either side (for routing)
- Eg: H1 wants to communicate with H2



Switches

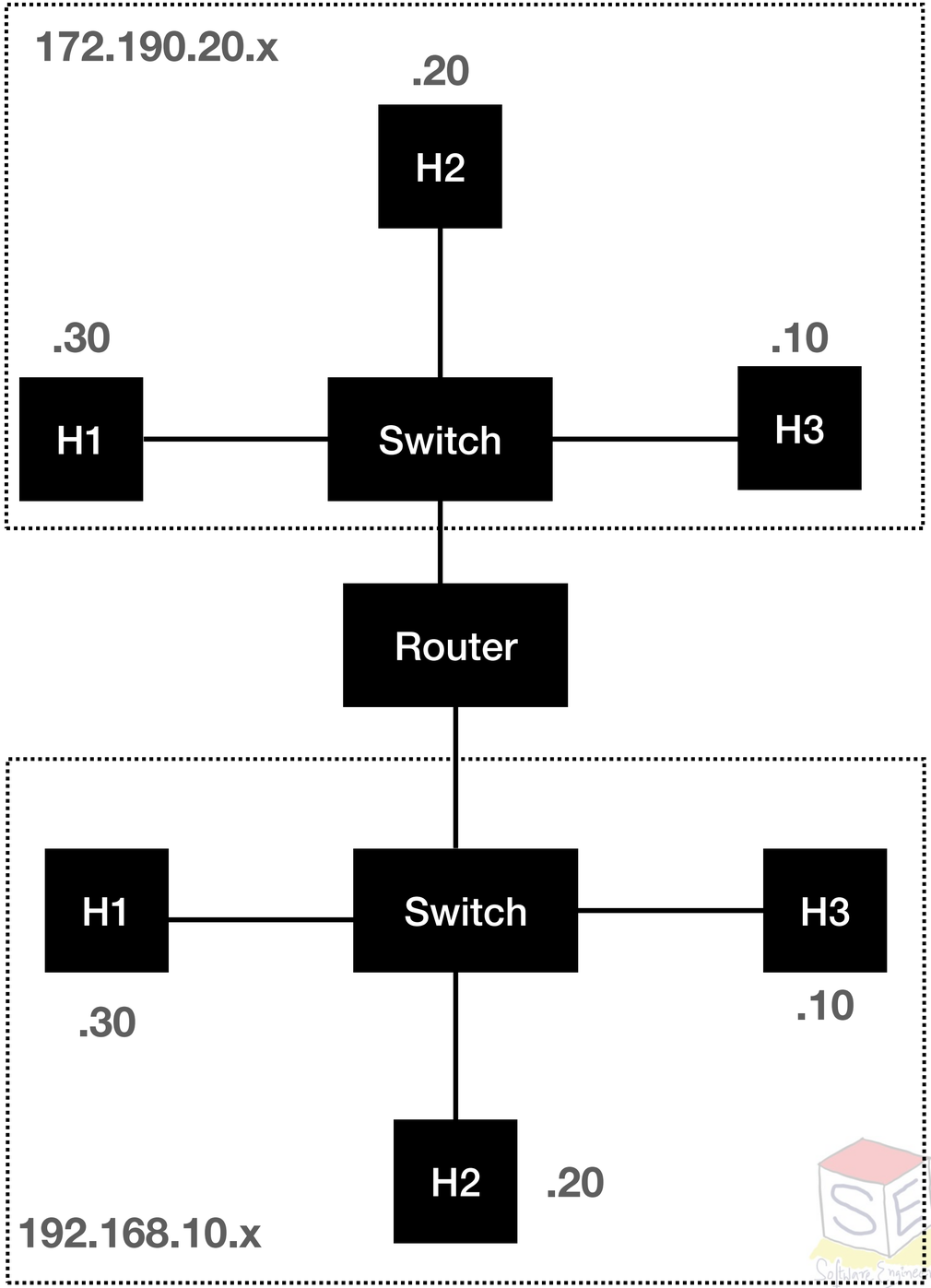
- Devices which facilitate communication within a network
- Combination of hubs and ports
- Knows or infers which hosts are on each port
- They have multiple ports
- Whatever connected to switch becomes part of one network



What if the host 192.168.10.20 wants to communicate with Another host in different network?

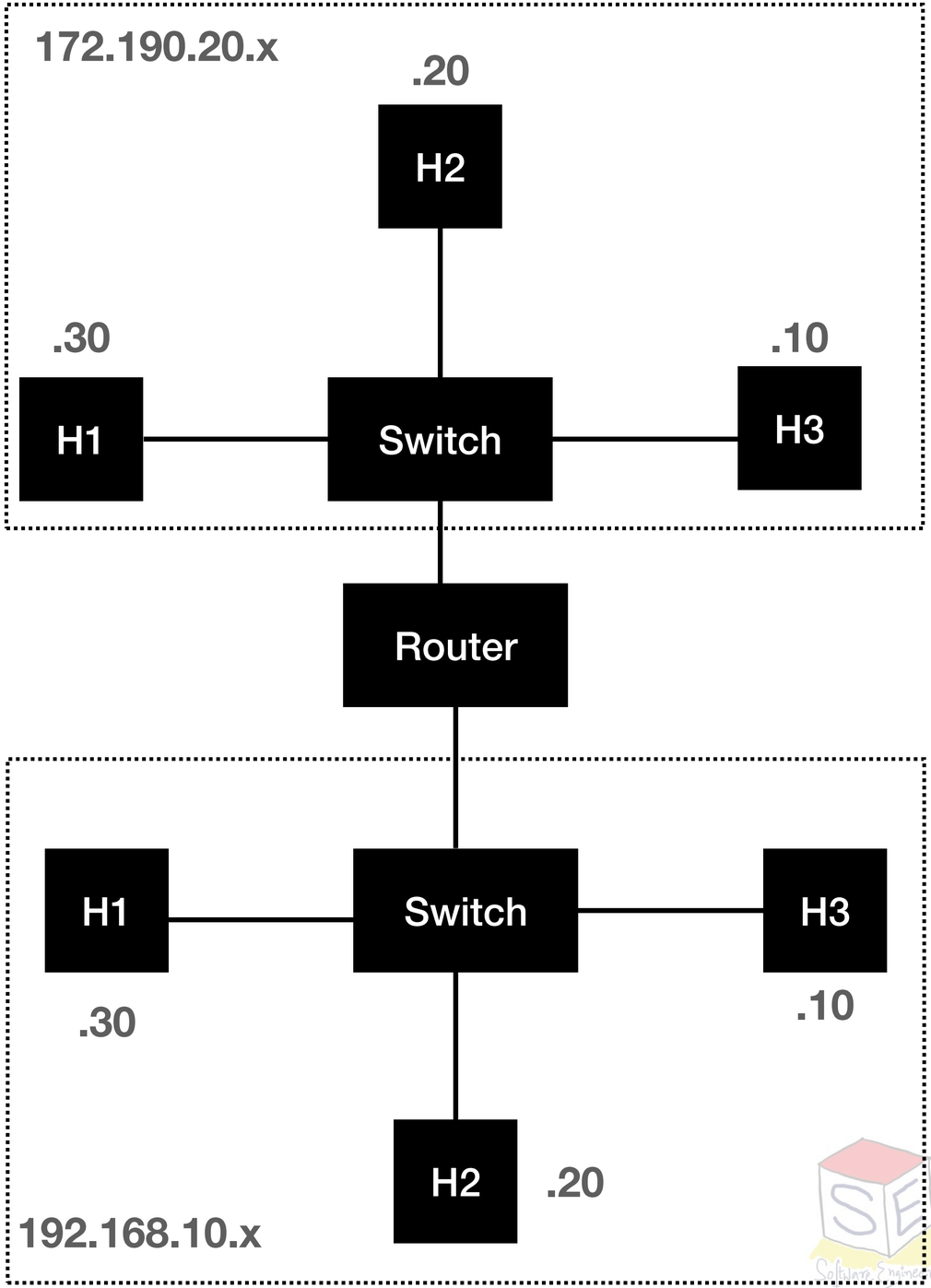
Routers

- Facilitate communication between the networks
- They provide like a traffic control point
  - Security, filtering, redirection
- Routers learn which networks they are attached to
  - Known as routes
  - Stored in a routing table
- Routers have their IP address in the network they are attached to

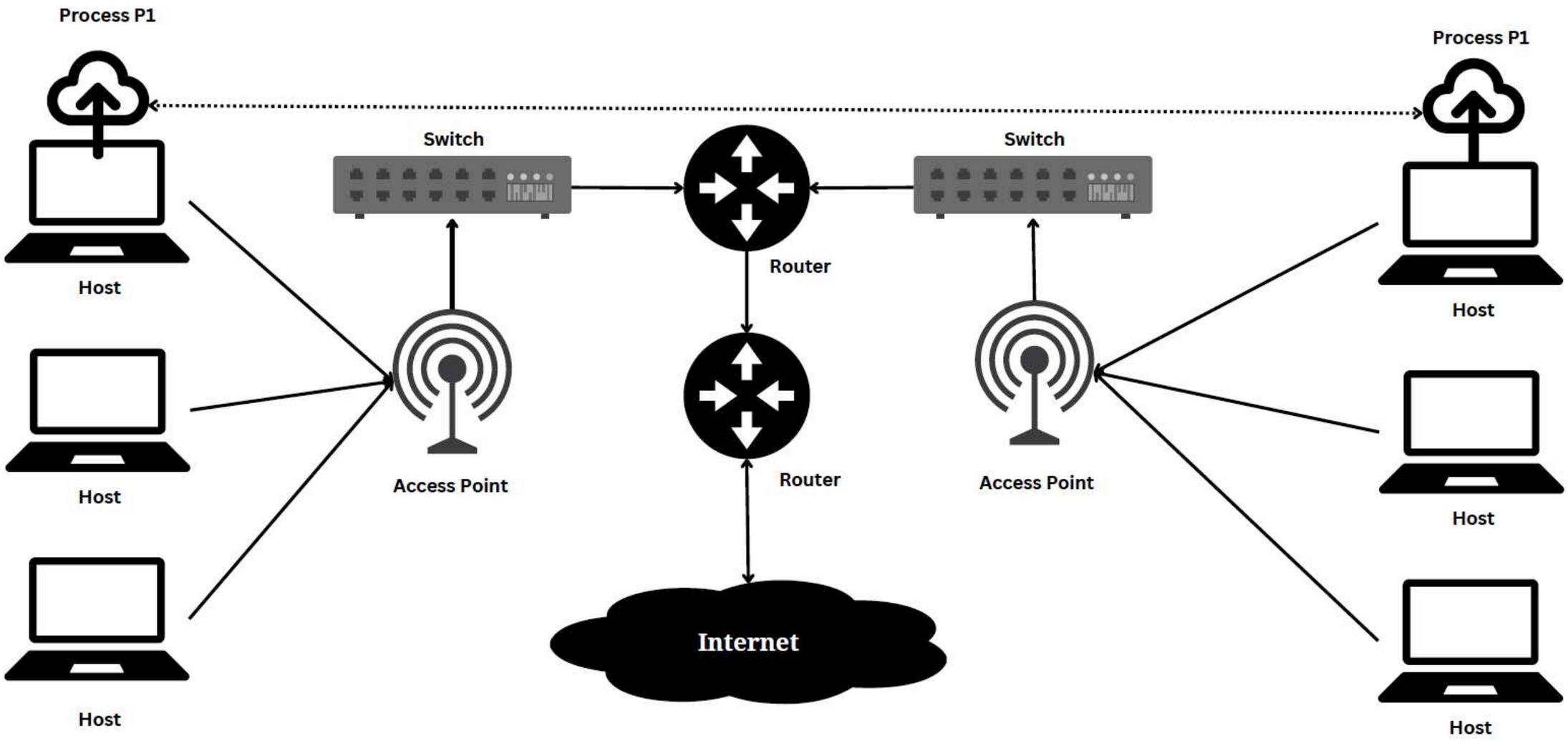


Routers

- Allows creation of hierarchy in the networks
- Every time hosts wants to go out of the network, it goes through router
- Internet is nothing but bunch of routers
- Gateway (IP address of router in the given network)
  - Becomes exit point of hosts outside their network

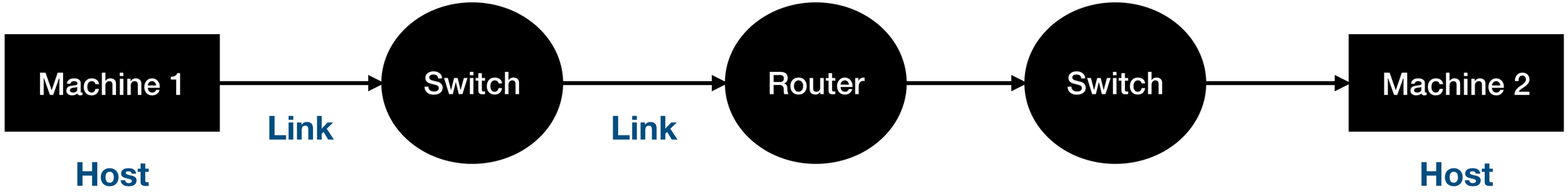


The Bigger Picture



What happens internally when one process wants to share/ receive data from another host?

Components of a network



Process Communicating over network

- Process A (eg: Whatsapp) is executing in Host 1
- Process B (whatsapp) executing in Host 2
- Process A wants to communicate with Process B
  - Some system call is made to access the network
    - Leads to an interrupt
    - Process A should know where process B is
    - Hardware support is needed (some network device with drivers)

Network Types

Multiple types of networks exist

- **Personal Area Network - PAN**
  - Devices communicate over a very short range
  - Eg: Bluetooth (master slave paradigm)
- **Local Area Network - LAN**
  - Private network operating within and nearby a single building (home, office, factory, etc.)
  - Wireless LANs runs speed from 11 Mbps to 7 Gbps (1 Mbps is 1,000,000 bits per sec) - Power in 10 (not in terms of 2)
  - Wired LANs operate at speed ranging from 100 Mbps to 40 Gbps (low latency)
  - Eg: connecting personal computers and consumer electronics (eg: printer)

Network Types

We have come a long way!

- One large Physical LAN can be divided into smaller logical LANs (Virtual LANs)
- Earlier days it was about broadcasting on a single line!
  - At most one computer could transmit successfully at a time
  - Use static allocation techniques
    - Every machine gets some time to transmit or receive
    - Round robin was used for scheduling
  - Packet collision used to happen (Wait for some random time and try again)

Network Types

- **Metropolitan Area Network - MAN**
  - Covers a city (City wide networks)
  - Think of cable TV networks (Earlier Antenna on top of the house and from the top of the hill the communication used to happen)
  - Idea of cable TV was used for providing two services (Internet and TV)
- **Wide Area Network - WAN**
  - Spans a large geographical area (often country, continent, etc.)
  - Eg: Internet is a large WAN (Dedicated WANs also exist)
    - There can be WAN that connects offices of organisation in different locations
  - Higher latency and lower transmission speeds. Cost for dedicated ones can be high

An Illustrative Scenario

Sending Letter/Courier

Community 1

Local Post office

City Post Office

Community 2

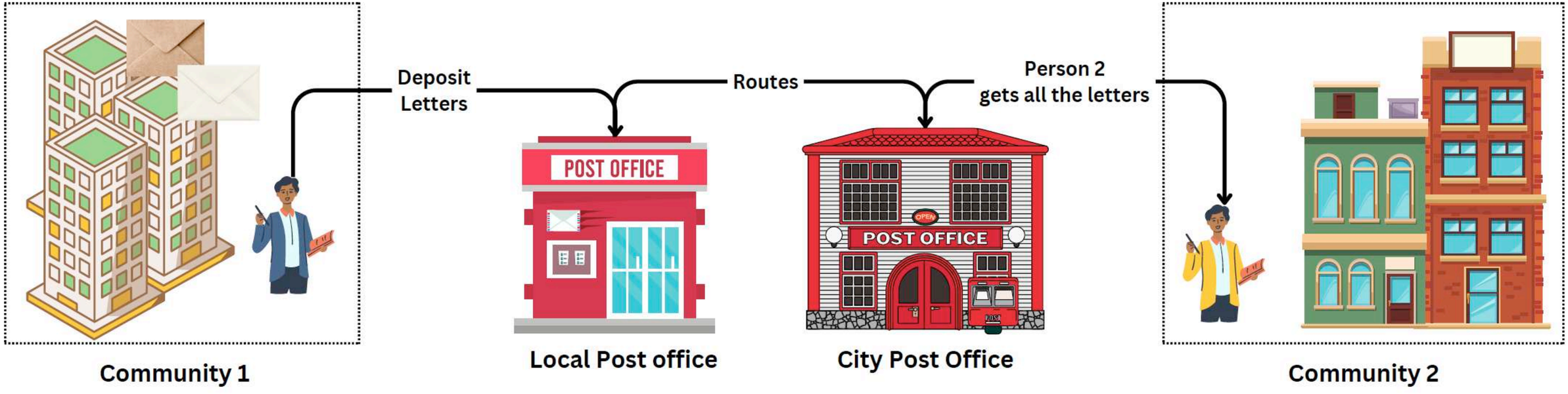
Two large communities in different parts of the city, two people in each community to collect and deliver the letters/couriers

An Illustrative Scenario

People in community send letters/couriers

One person is collecting all the letters/couriers

Person delivers the letter to each person

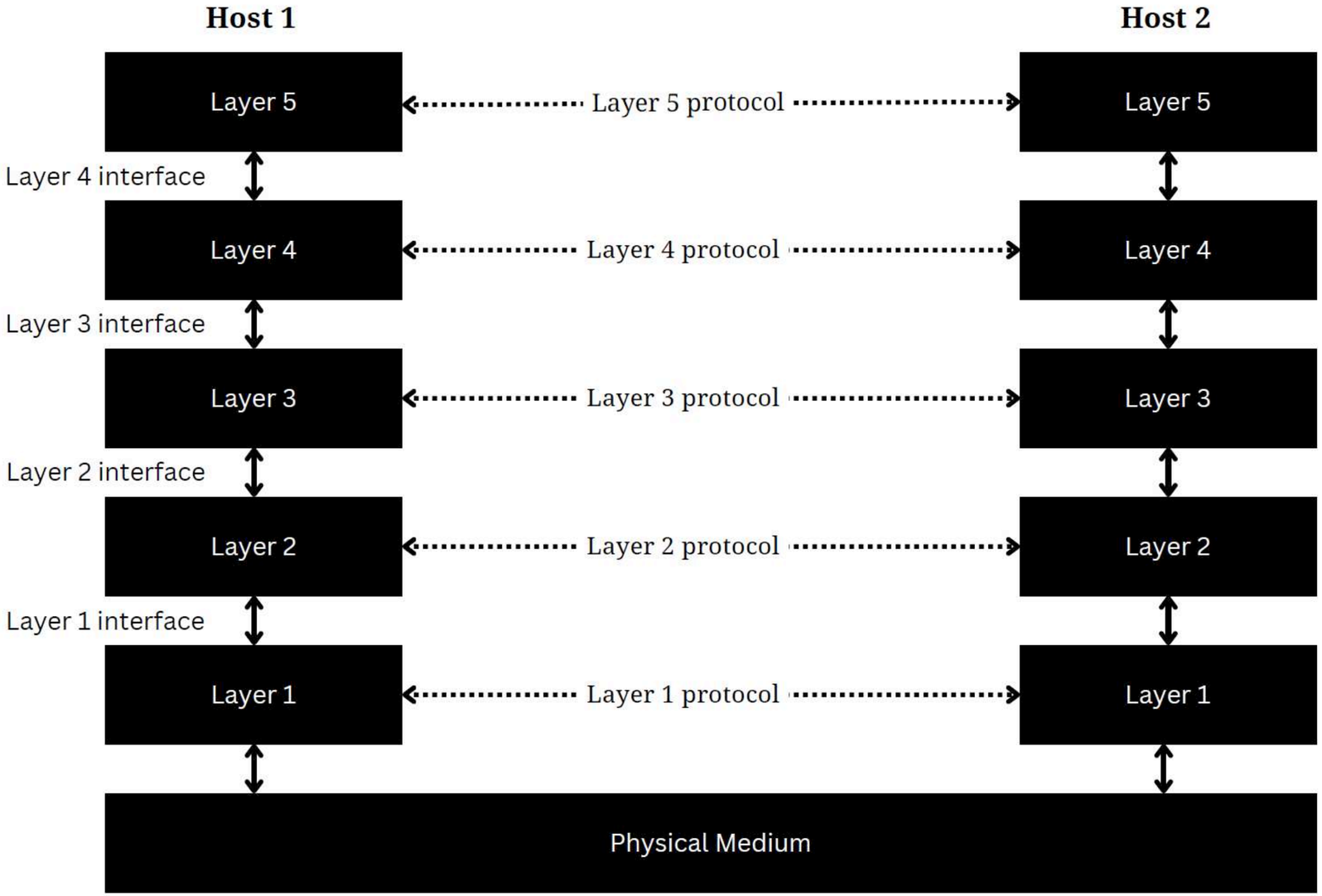


Key points: Door number, building number, Post office has an identifier, etc.

Lets take this to Process communication

- Process A (eg: Whatsapp) is executing in Host 1
  - Process B (Whatsapp) is executing in Host 2
- Host 1 will have an address, same is the case with host 2
- How to ensure the data reaches from Host 1 to Host 2?
  - What all needs to be considered?
  - Remember: There will be multiple processes that are executing in a host

Networking Layers



Networking Layers

- Reduce design complexity, network organised a stack of layers or levels
- The layering provides abstraction in and to other layers
- Communication (in some sense) happens between the corresponding layers
  - Layer n one machine communicates with layer n of another using a **protocol**
  - Protocol is just like agreement between communicating practices
- Set of layers along with protocols => **Network Architecture**
- Between each pair of adjacent layer their is an **interface**
  - Defines the primitive operations and services the lower layer makes available

The OSI Model

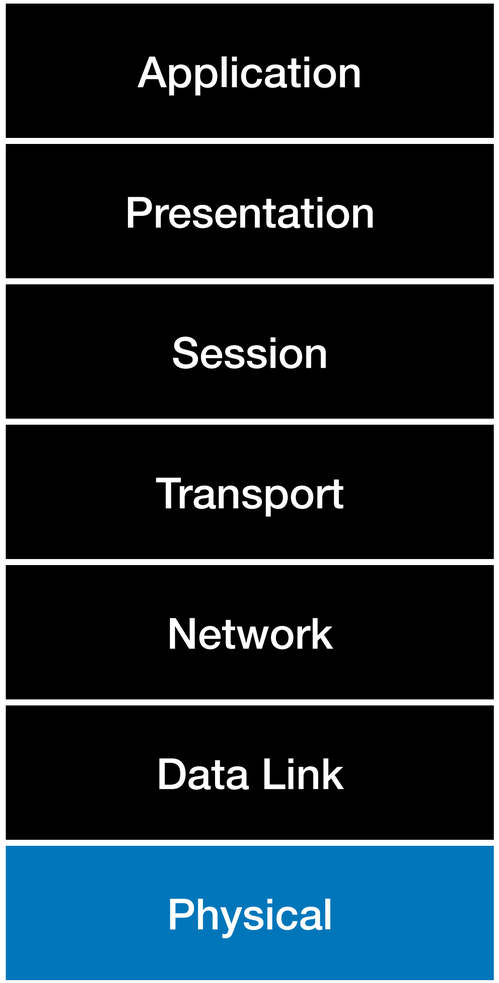
Application
Presentation
Session
Transport
Network
Data Link
Physical

- Open System Interconnection (OSI)
- A Conceptual framework used to understand how communication works through different layers
- Divides the network communication process into seven layers
- Developed to facilitate interoperability between different technologies
- Each layer has a specific function. If they all do what they are supposed to do => sharing of data

Physical Layer (L1)

Ultimately everything is 0's and 1's

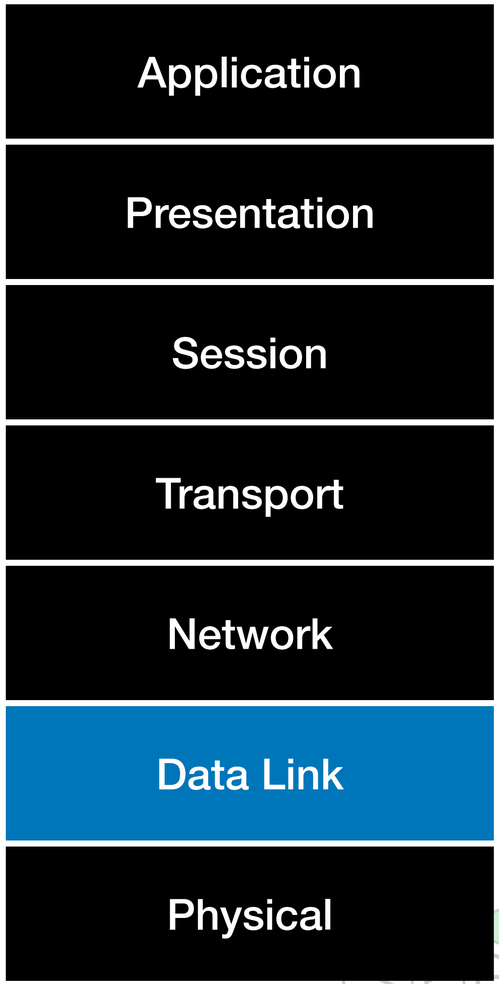
- Data is in the form of bits - 0s and 1s
- Something has to transport the bits from one machine to another - Physical layer
- Concerned with transmission of raw bits over physical medium, like a cable
- L1 technologies: Ethernet cables, Optical fiber, Coaxial cable, etc.
    - Even WiFi is L1 technology, hub, repeater, etc.



Data Link Layer (L2)

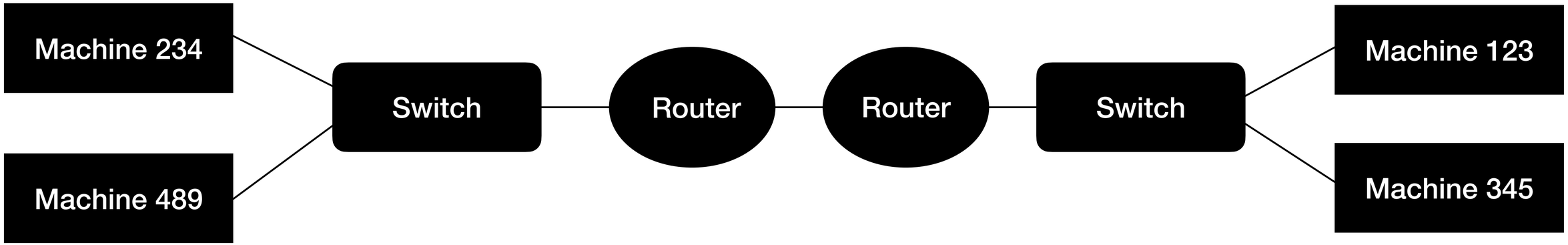
Hop to Hop connectivity

- Interacts with the physical medium
- Adds/takes data to/from L1 technology
- Responsible for creating a reliable link between two directly connected nodes, error correction, etc.
- L2's responsibility is mainly taking data from one hop to another
  - Uses an addressing scheme - MAC addressing
  - 48 bits represented as 12 hex digits
- L2 Technologies : NIC, WiFi access cards
- Switches are also L2 technology (help in moving data)



Communication is just more than Hop to Hop

What about communication from 234 to 345?

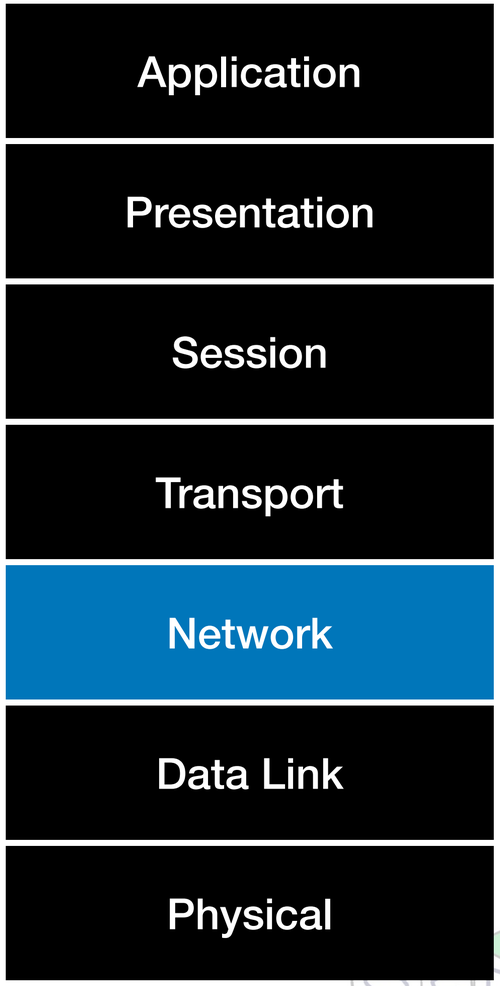


- Hop to hop is guaranteed by L2
- What about end-to-end delivery?

Network Layer (L3)

End-to-end Communication

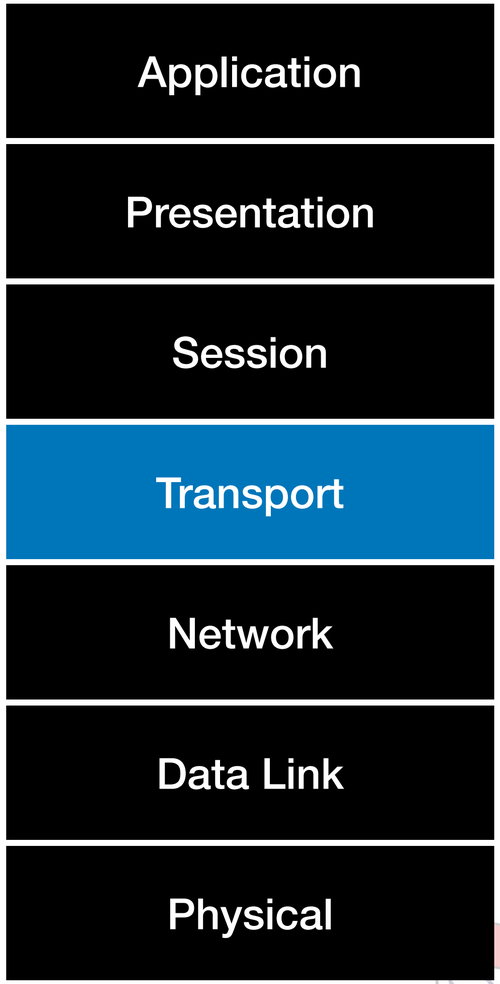
- Manages routing through different routes in a large network
- Uses an addressing scheme - IP addressing
  - 32 bits represented as 4 octets (IPv4)
- Performs functionalities such as Logical addressing (IP), Path selection and packet forwarding
- L3 technologies: routers, even hosts are L3, L3 switches



Transport Layer (L4)

Service to Service

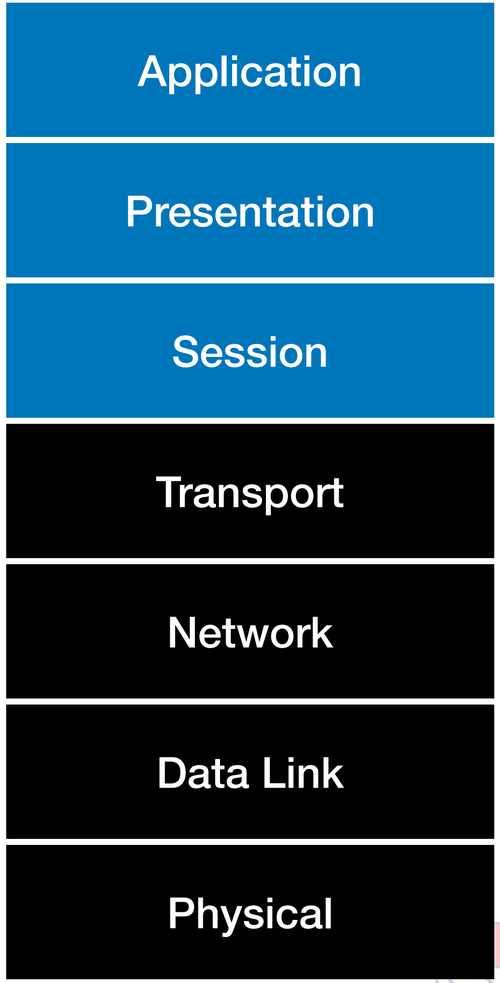
- How to ensure that the right process receives the data?
  - Many process will be executing or are waiting to execute?
- Ensures data transfer is reliable, sequential and free from others
- Manages flow control and error correction
- Layer 4 has an addressing scheme to guarantee message delivery
  - Ports! (0 - 65535), Privileged: 0-1023, Registered: 1024 - 49151
  - Every process will have a port through which sending or receiving data
- **L4 Technologies:** TCP, UDP



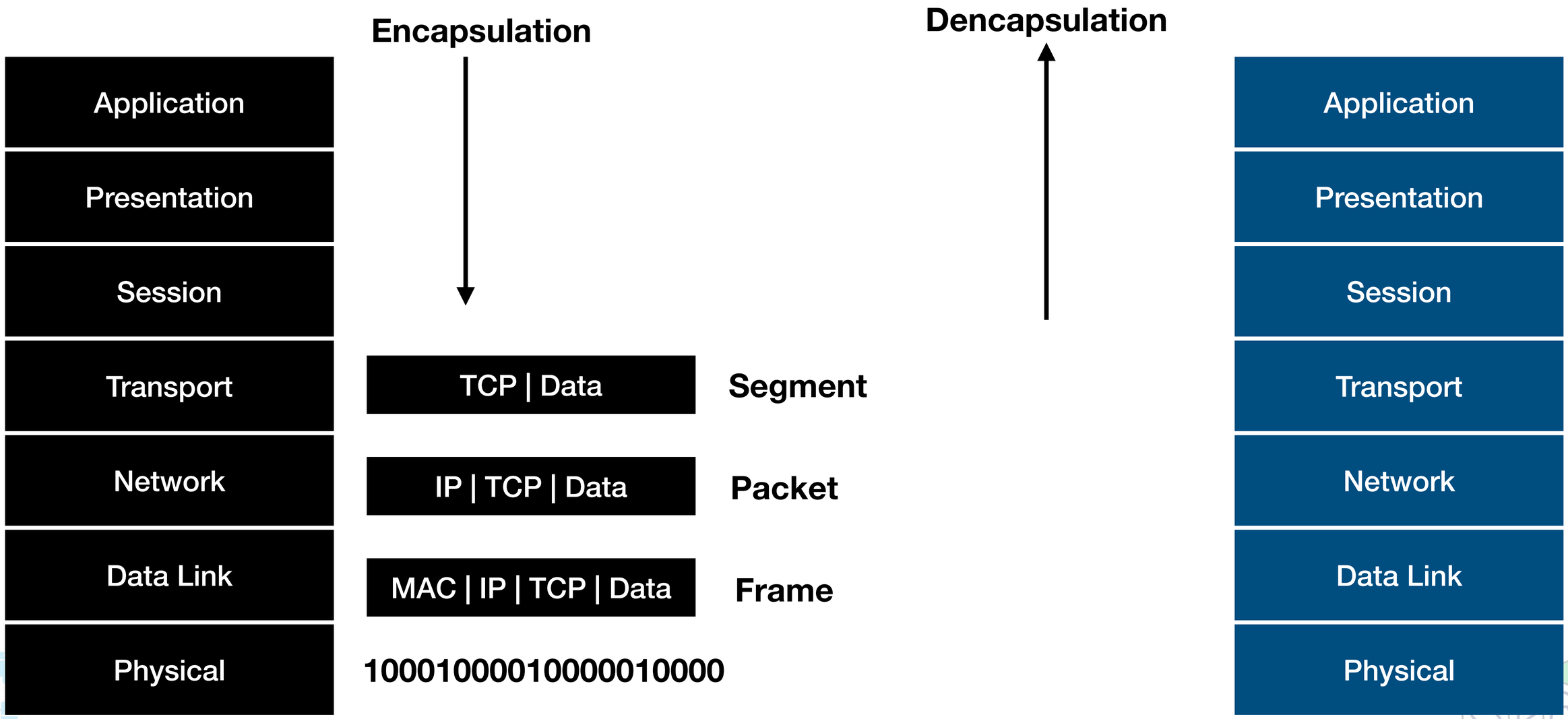
Session, Presentation and Application

Application to Application

- **Session Layer (L5)**
  - Manages connection between different devices
  - Establishing, maintaining and terminating connections
- **Presentation Layer (L6)**
  - Ensures that data is in format that sender and receiver can understand
  - Manages data encryption, compression
- **Application Layer (L7)**
  - Provides network services to the application processes
  - Eg: web browser, email clients, other softwares/apps



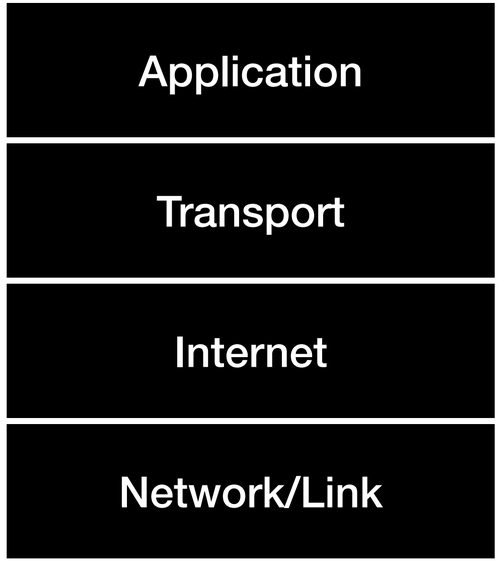
Putting It Together



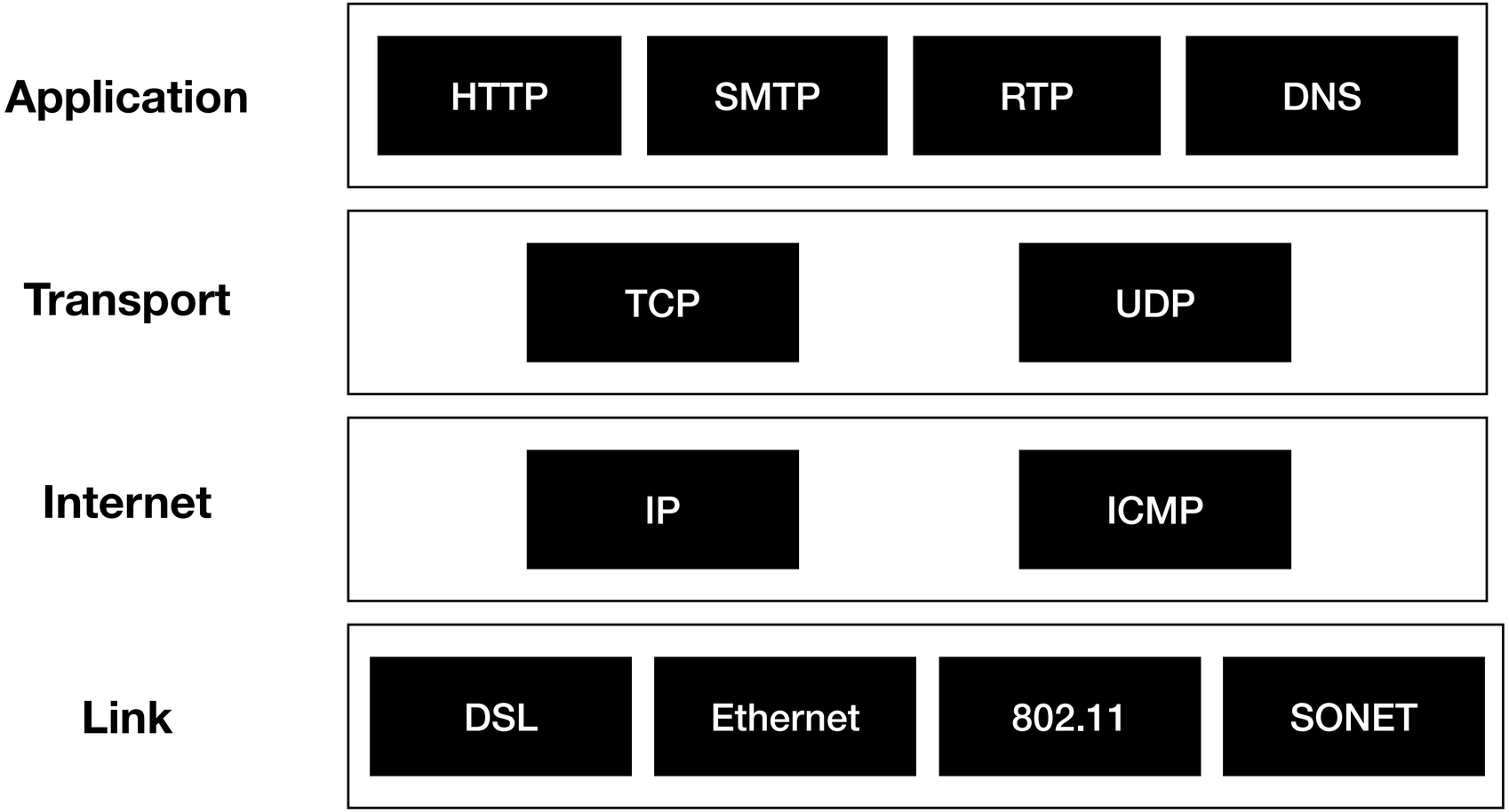
The 4 Layer Model

Internet Model or TCP/IP model,

- OSI model is more educational purpose
- 4 layer model more used in reality
- Application layer - Corresponds to application, presentation and session
- Transport layer - Transport layer of OSI
- Internet layer - Network layer of OSI
- Network - Physical and data link layers of OSI



Network Protocol Stack



SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Transport Layer and how it works!

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

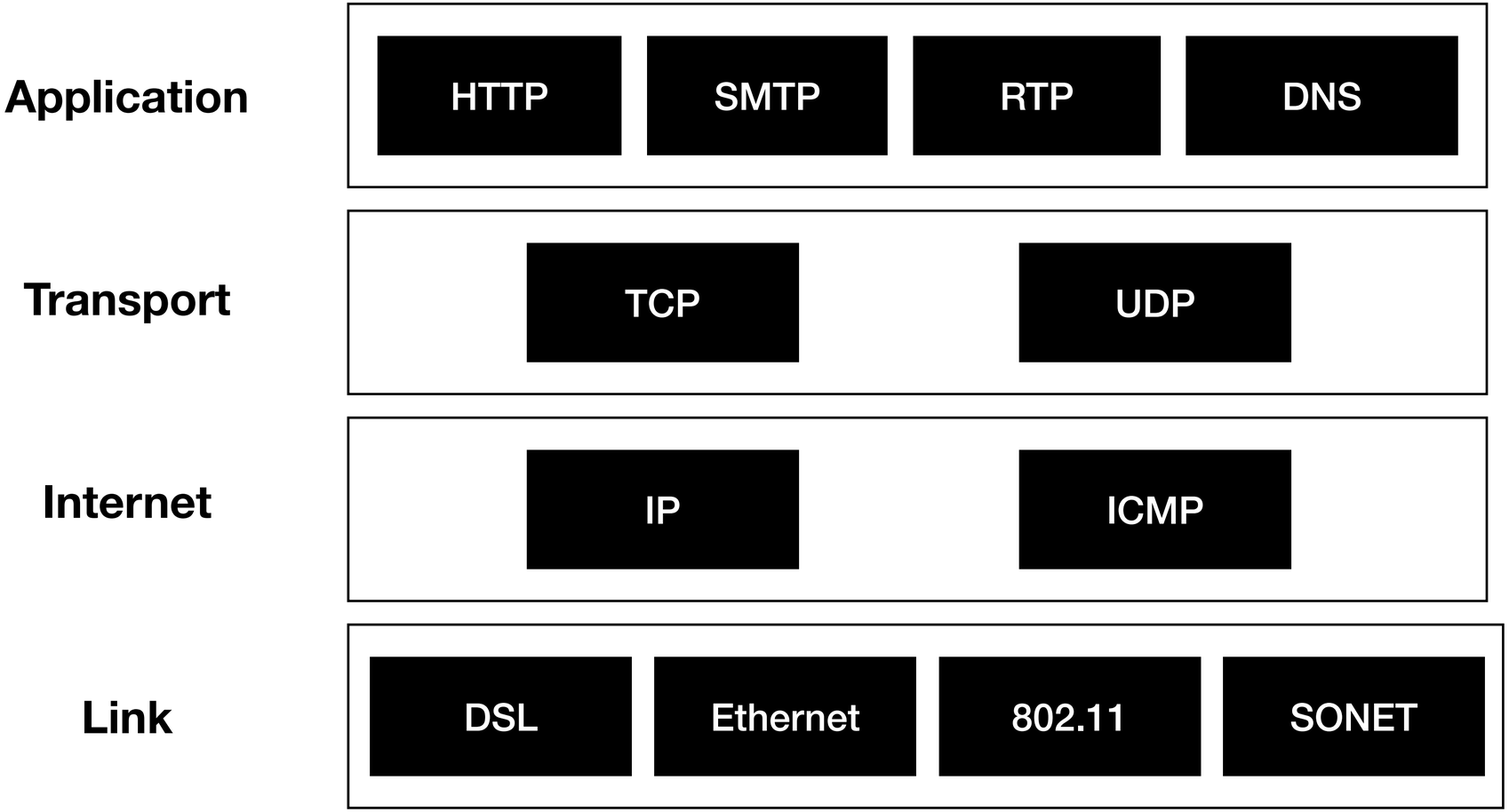
Acknowledgement

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:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

Network Protocol Stack



Onto Transport Layer

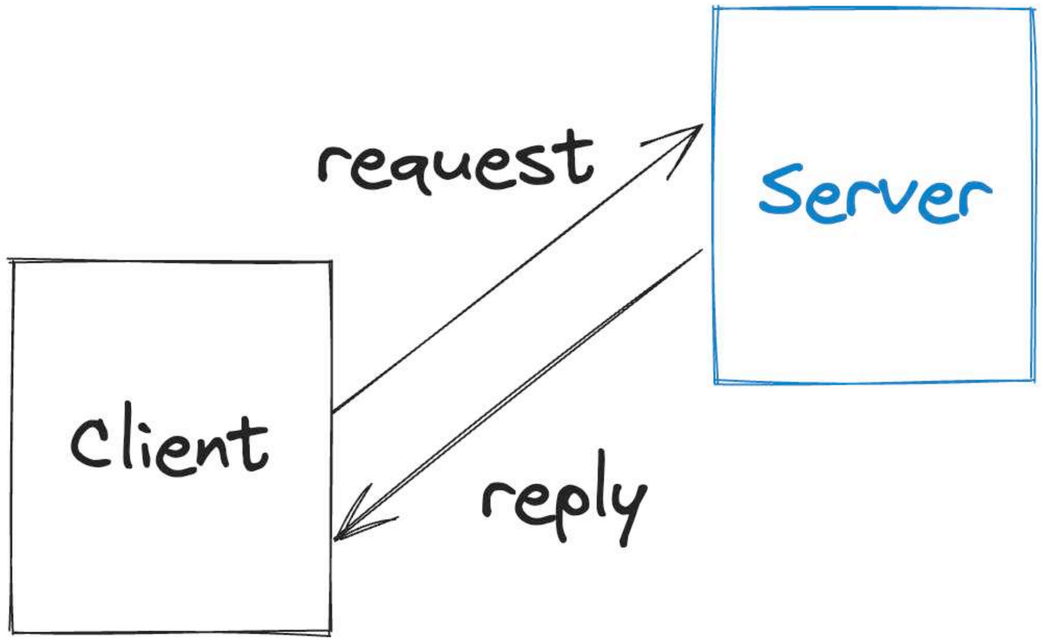
Service to Service Delivery

- **Multiplexing** and **Demultiplexing**
- Addressing scheme: **Ports**
- Two strategies/protocols that allows this
  - Transmission Control Protocol (TCP) - favours reliability
  - User Datagram Protocol (UDP) - favours efficiency

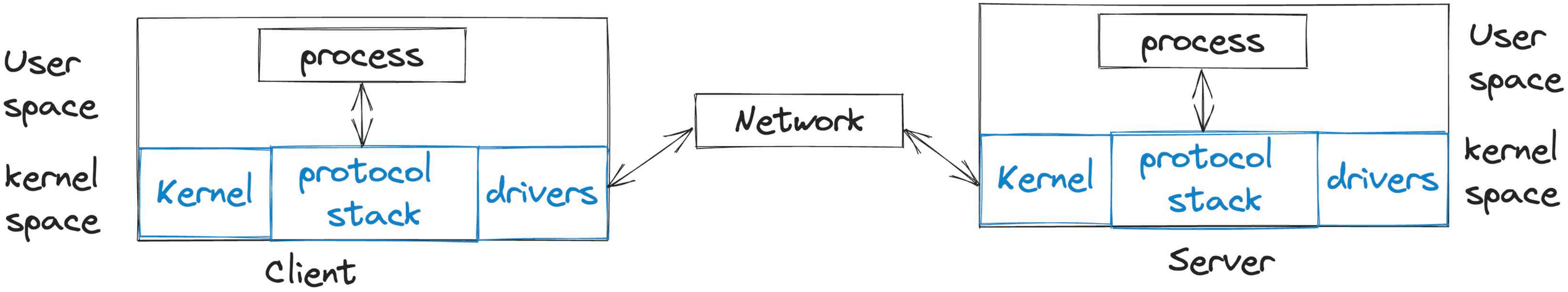
Lets go back

Two process wants to communicate

- Client sends a request to the server
- The server provides a reply based on the request made by client
- Eg: File transfer to get files
- Eg: web browsing, sent a url and get page
- How to transport data from process to process?



The role of Operating System



- Software component in the OS that supports network calls - **Protocol stack**
- Provides Service primitives which are nothing but system calls - **Some API?**

Any idea on what should be some functionalities that should be made available?

Hint: Think of process API

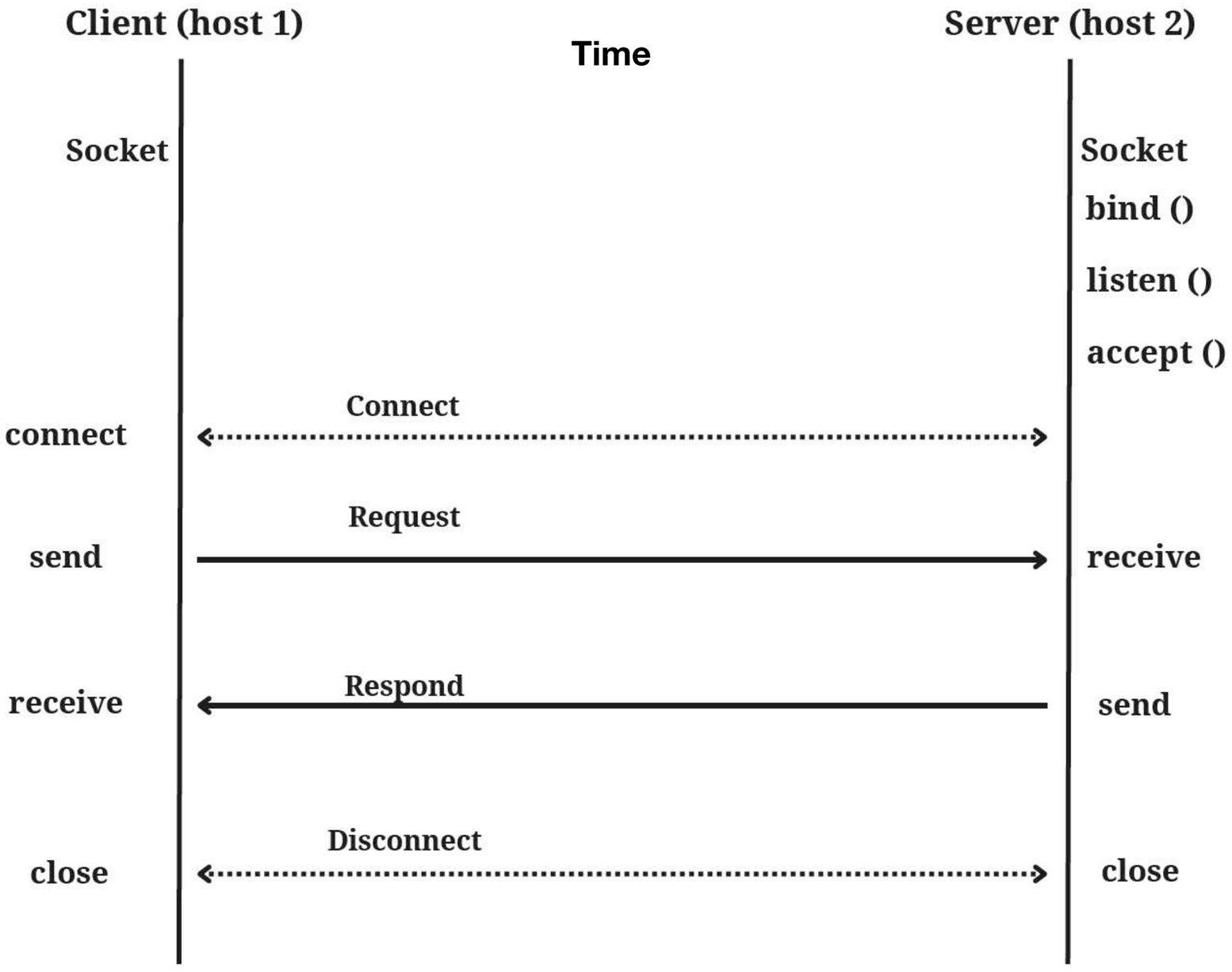
The Socket API

- Simple abstraction to use the network
- The network service API used to write all network applications
- Part of all OS and all language [Berkley Unix, 1983]
  - Allows user space applications to interact with networking subsystem
- Two services:
  - **Streams:** Reliable (Connection-oriented)
  - **Datagram:** Unreliable (Connection-less)
- Allows applications to attach to the network at different ports

The Socket API

Function	Description
socket()	Creates a new socket of a certain type (depending on TCP or UDP) and returns file descriptor
bind()	Associates the socket with a specific IP and port
listen()	For server sockets, it allows sockets to listen for incoming connections
accept()	For server sockets, it waits for client to connect and then return a new file descriptor
connect()	For client sockets, it initiates a connection to a server.
send() / receive()	Transmit data or receive data
close()	Terminate the connection

Using Sockets



More about Ports

- Application process is identified by tuple (IP address, Protocol, Port)
    - Port are 16-bit integers representing “mailboxes” that process leases
- Servers are often bind to “well-known-ports”
- Clients are assigned ephemeral ports
    - Chosen by the OS temporarily

Some well Known Ports

Port	Protocol	Use
20, 21	FTP	File Transfer
22	SSH	Remote login
25	SMTP	Email
80	HTTP	World wide web
443	HTTPS	Secured web
543	RTSP	Media Player Control

An Opportunity for a Context Switch?

- The calls of establishing socket are blocking calls
  - connect(), accept(), receive()
  - Once the call is made, OS halts the program to wait to receive some response
  - They are essentially **System calls**
  - Trap instruction is called and there is an opportunity for a context switch

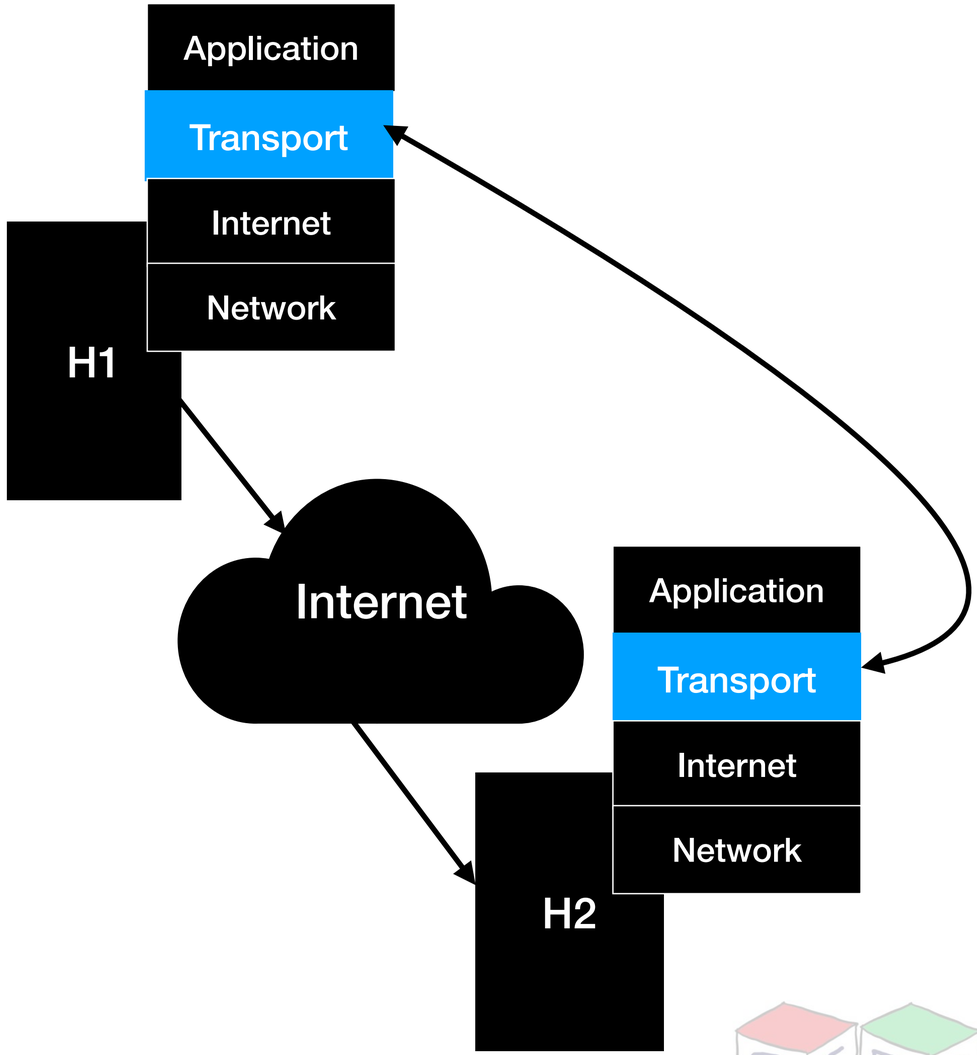
Let us take a step back

Types of Links

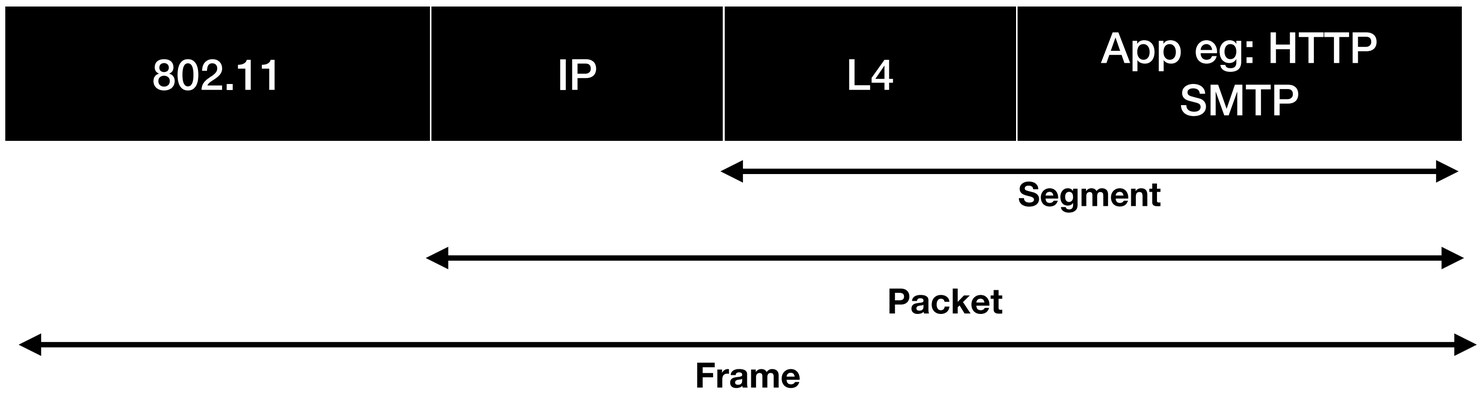
- **Full Duplex**
  - Bidirectional
  - Both sides at the same time
- **Half-duplex**
  - Bidirectional
  - Both the sides but only one direction at a time (eg: walkie talkies)
- **Simplex**
  - Unidirectional

Transport Services and Protocols

- Provides logical communication between application processes running on different hosts
- Transport protocols actions in the end systems:
  - Sender: breaks application messages into segments, passes to network layer
  - Receiver: reassembles messages into messages, passes to application layer
- Protocols: TCP, UDP



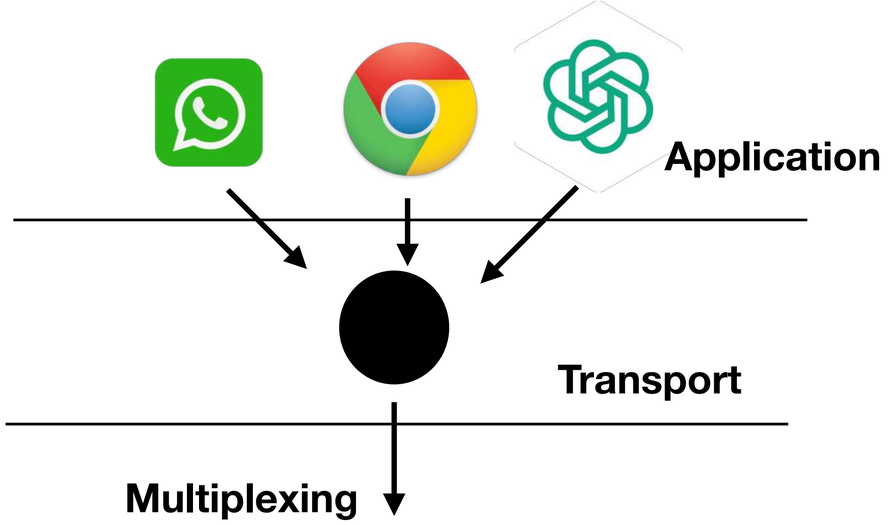
Quick Recap



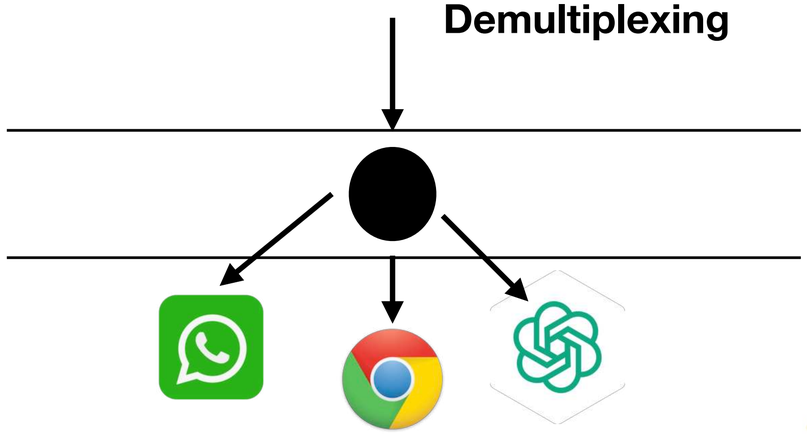
- Segments carry data across the network
- **Segments** are carried within the packets, within frames
- Each layer adds a **header** (Above L4 will be replaced by its header)

Multiplexing and Demultiplexing

- **Multiplexing as sender**: Handle data from multiple sockets, add transport header

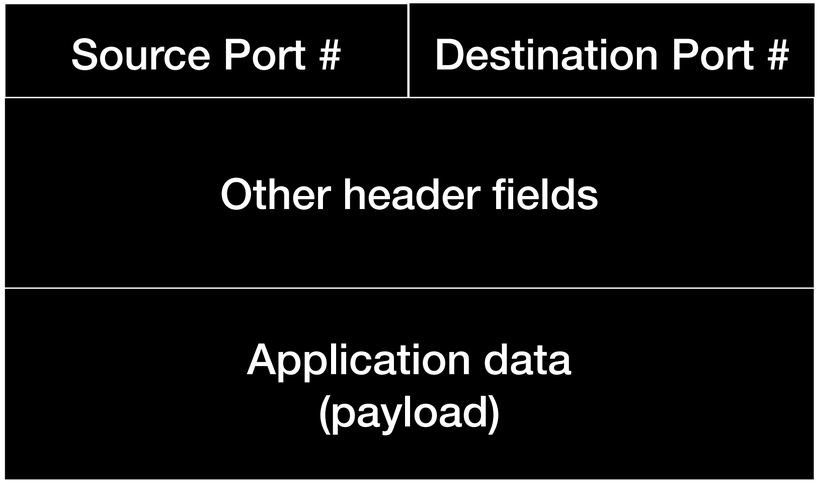


- **Demultiplexing as receiver**: Use header info to deliver received segments to correct socket



Working of Demultiplexing

- Host receives IP datagrams
  - Each datagram has source IP address, destination IP address
  - Each datagram carries one transport layer segment
  - Each segment has source and destination port number
- IP addresses and ports are used to direct segment to appropriate socket



TCP/UDP Segment format

Connection Oriented vs Connectionless

Demultiplexing Scenarios

- **Connection oriented (TCP)**
  - TCP socket identified by 4 tuple
    - Source IP, destination IP, source port and destination port
  - Receiver uses all 4 to direct segment to appropriate socket
  - Server may support many TCP sockets
    - **Each socket has it own client**

- **Connectionless (UDP)**
  - UDP socket identified by 2 tuple
    - Destination IP and port
  - Receiver uses the port to redirect to the corresponding socket
  - UDP segments with same destination port but different IP or source port
    - **Redirected to same socket**

TCP vs UDP

TCP	UDP
Connection Oriented	Not Connection Oriented
Reliability (order is maintained and retransmission)	Unreliable
Higher overhead - reliability, error checking, etc	Low overhead
Flow control (based on network)	No implicit flow control
Error detection - retransmit erroneous packets	Has some error checking - Erroneous packets are discarded without notification
Congestion Control	No Congestion Control
Use cases: HTTP/HTTPS, File transfer, Mail	Use cases: Streaming data, VoIP, DNS queries, ..

Connection Oriented and Reliability

- **Connection Oriented**
  - In TCP, the connection is first established before the data is transmitted
  - In UDP there is no notion of connection starting and ending (use timeout)
- **Reliability**
  - Confirmation of data delivery (Acknowledgement is there) in TCP
    - Order is preserved or maintained
    - Error can be handled (Awareness). TCP can handle it.
  - In UDP there is no confirmation, the client trusts that there is someone to receive the data (Fire and Forget)
    - No error awareness (at L4). Protocol does not handle it

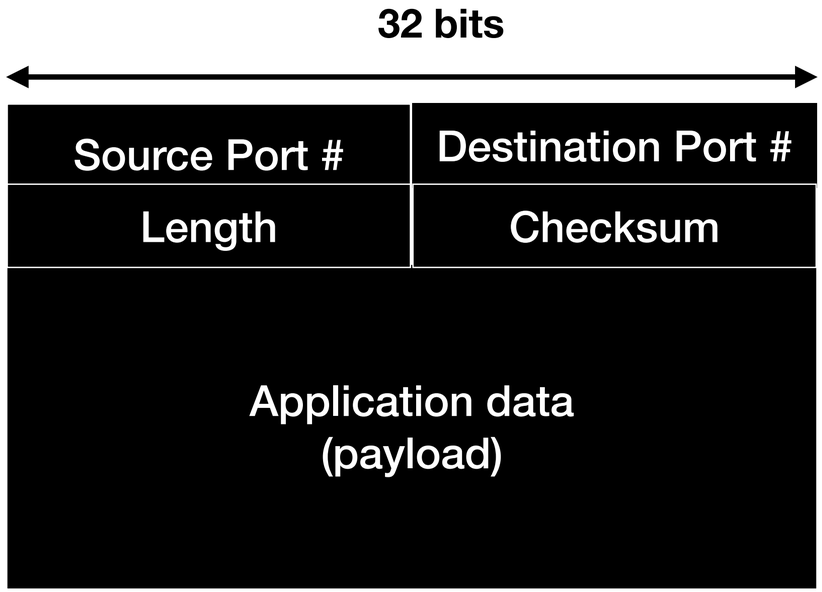
Flow Control and Overhead

- Flow Control
  - TCP can adjust the transmission rate to use maximum available bandwidth
  - Check how much the receiver can receive and adjust accordingly

- Overhead
  - TCP Adds a larger header to the data ~ 20 bytes or even more
  - TCP has more features that does not exist in UDP
  - In UDP the header length is ~ 8 bytes

UDP Segment Header

- Length: In bytes of the UDP segment including the header
- Checksum: For error detection (16 bit value which represents the sum of UDP header, payload and Pseudo header from IP layer)
  - Supports Error detection
  - Makes use of 1's compliment arithmetic to find the sum



UDP Segment Format

Checksum Process

- **Sender**
  - All contents of the header including IP addresses are treated as sequence of 16 bit integers
  - Checksum: addition (one's complement) of segment content
- **Receiver**
  - Compute checksum of received content
  - Check if received and header checksum are equal - No error
  - Else, Error detected

Example

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Add it back	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
Sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
Checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

TCP is the most used protocol on the internet. How does TCP work?

What all you need to provide some features that TCP provides?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

TCP Explained and Introduction to Memory Virtualization

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

TCP is the most used protocol on the internet. How does TCP work?

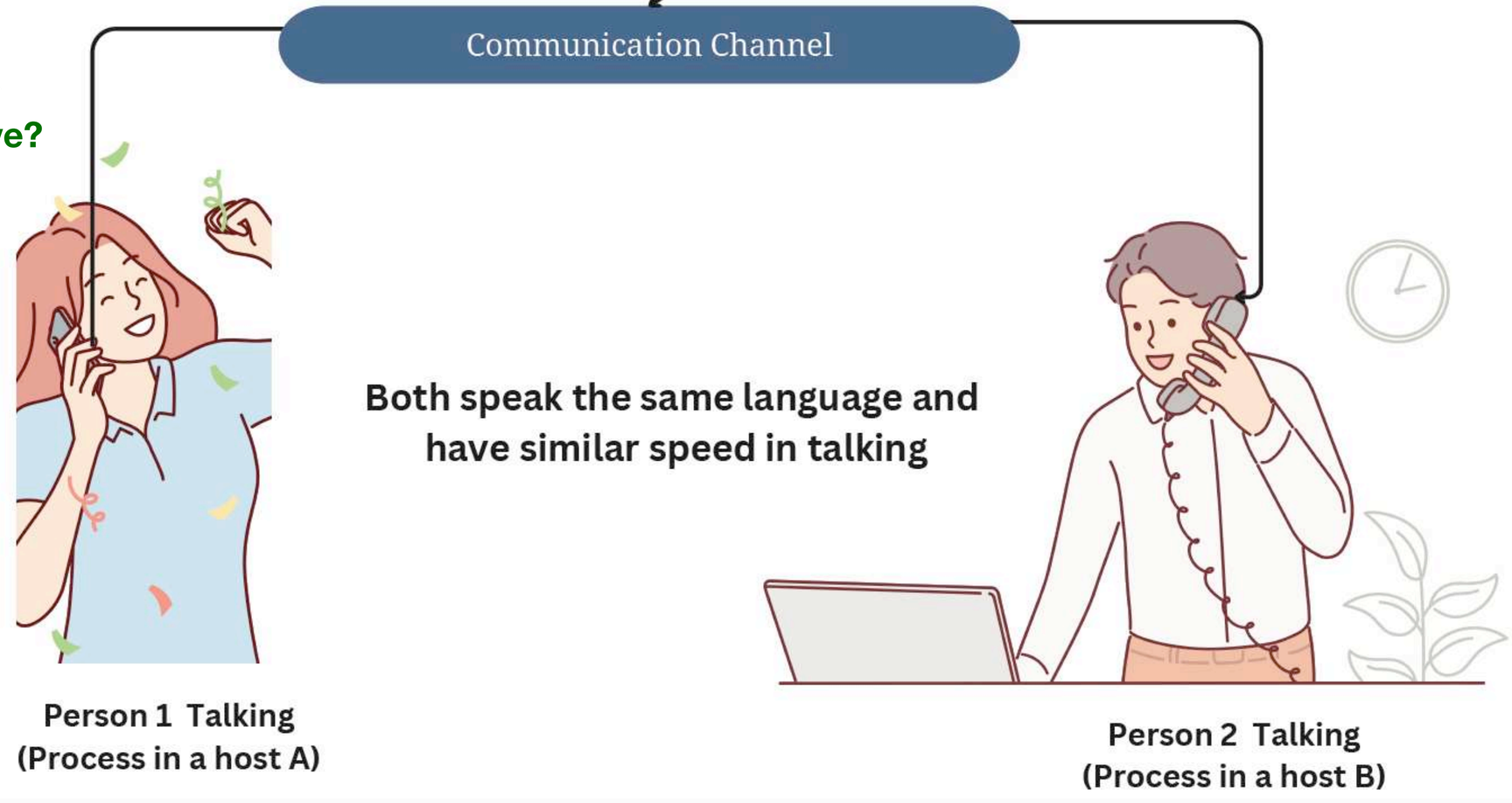
What all you need to provide some features that TCP provides?

A Small Analogy

Network and Link Layer

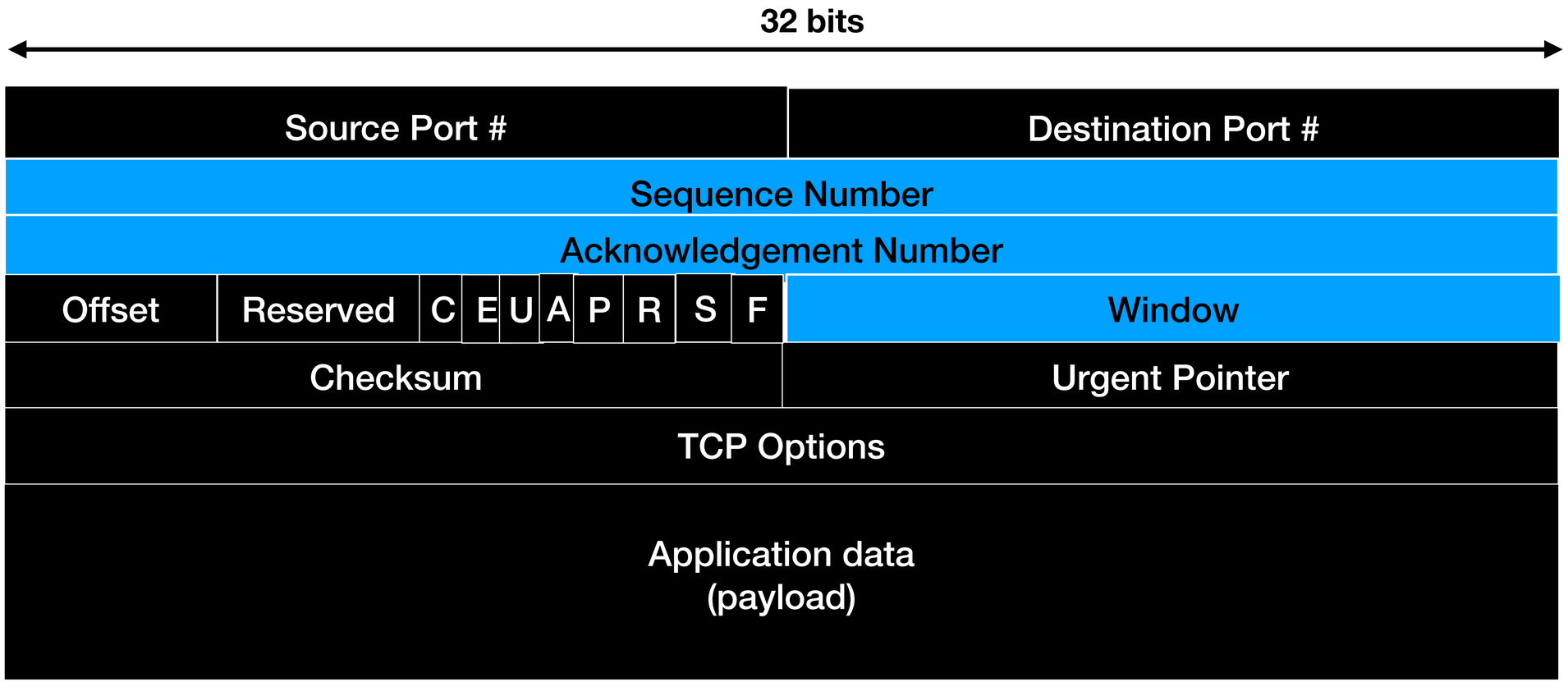
Communication Channel cannot be always reliable!!

What can we do from the protocol perspective?



Do we foresee some challenges?

Lets go into TCP - Header



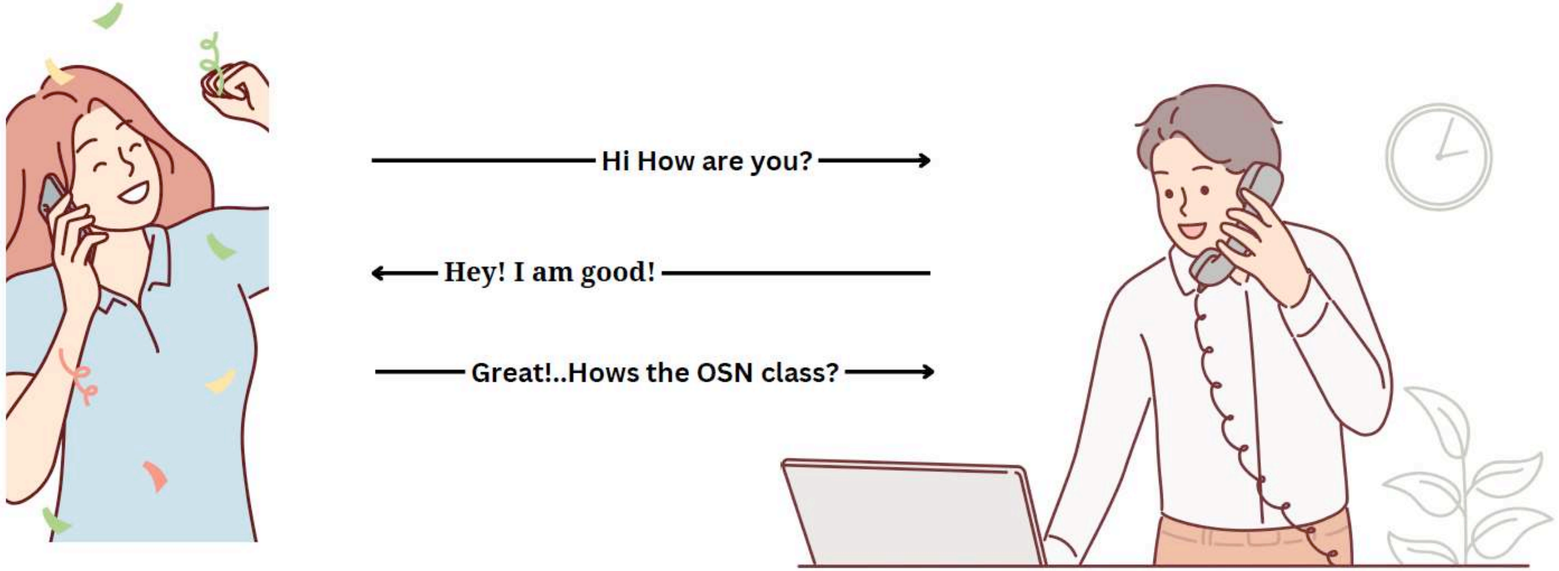
TCP Segment Header

Header Elements

- **Sequence number:** Tracks bytes that are sent (# of bytes that are sent)
- **Acknowledgement number:** Tracks bytes that are received (Sequence number of the next expected byte)
- **Window/Receive Window:** Number of bytes the receiver can accept (Flow control)
- **A:** Acknowledgement bit
- **R, S, F:** Connection management
- **C, E:** Congestion notification
- **Offset:** Length of the TCP header

What do ACK and Sequence Number do?

Reliability!!



Person 1 Talking
(Process in a host A)

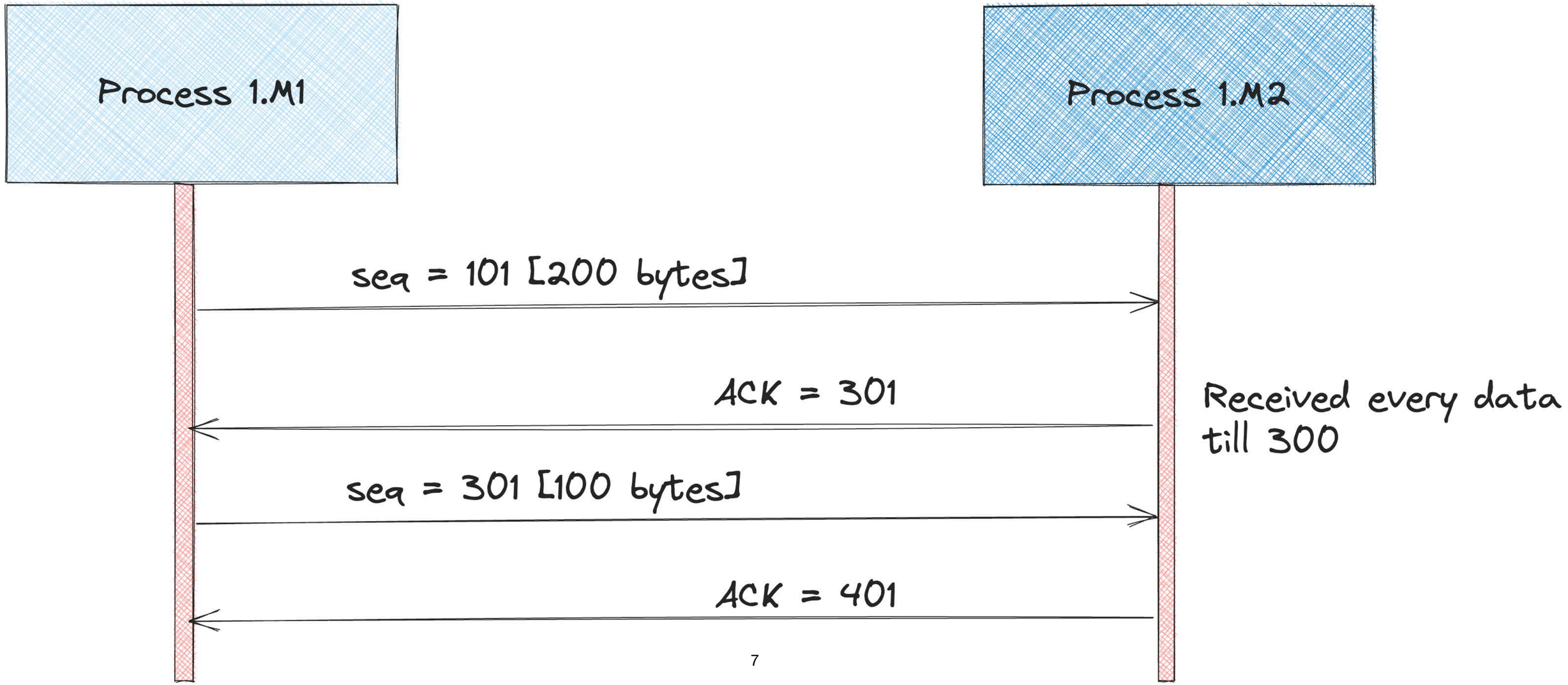
Person 2 Talking
(Process in a host B)

Each word the Person 1 says reaches person 2 in the same order

Whatever Person 1 Says, Person 2 acknowledges before adding new points to the conversation

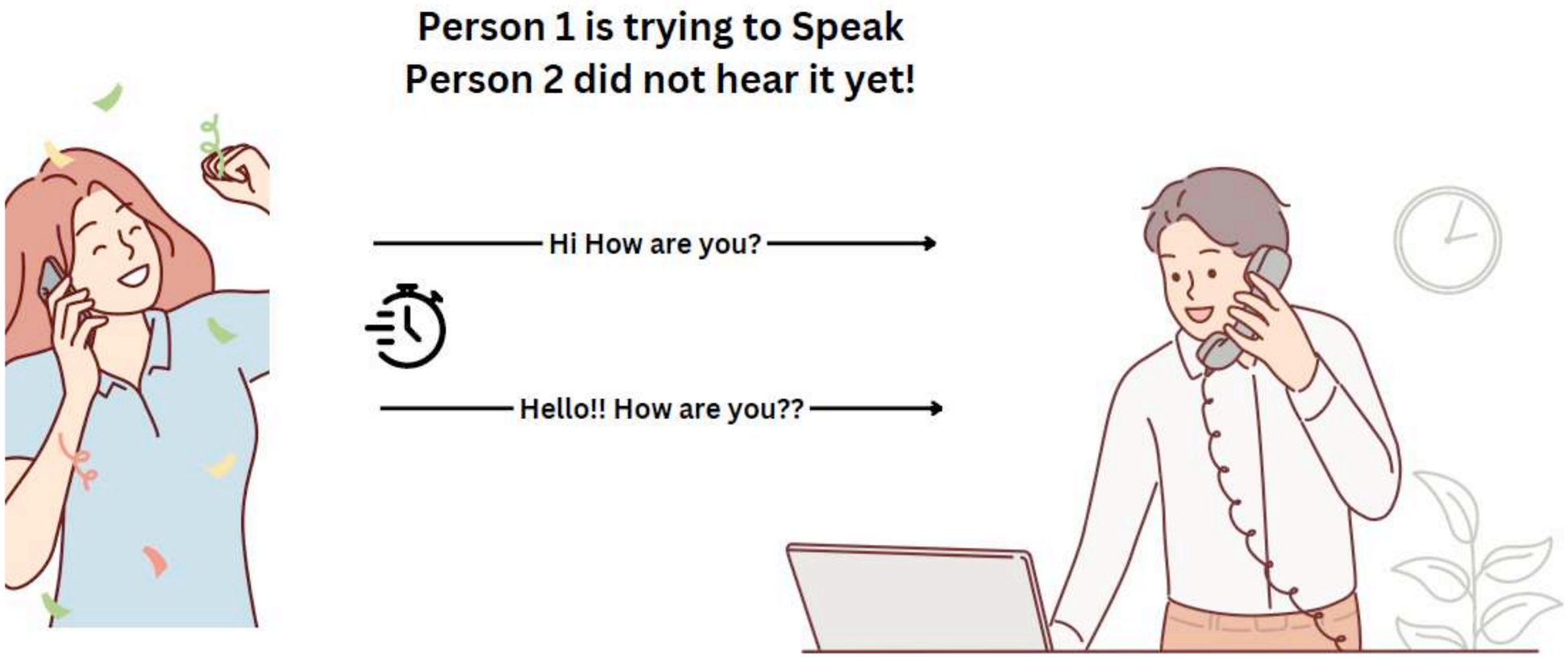
What do ACK and Sequence Number do?

Reliability!!



How to handle if data is lost?

Can we retransmit?



Person 1 Talking
(Process in a host A)

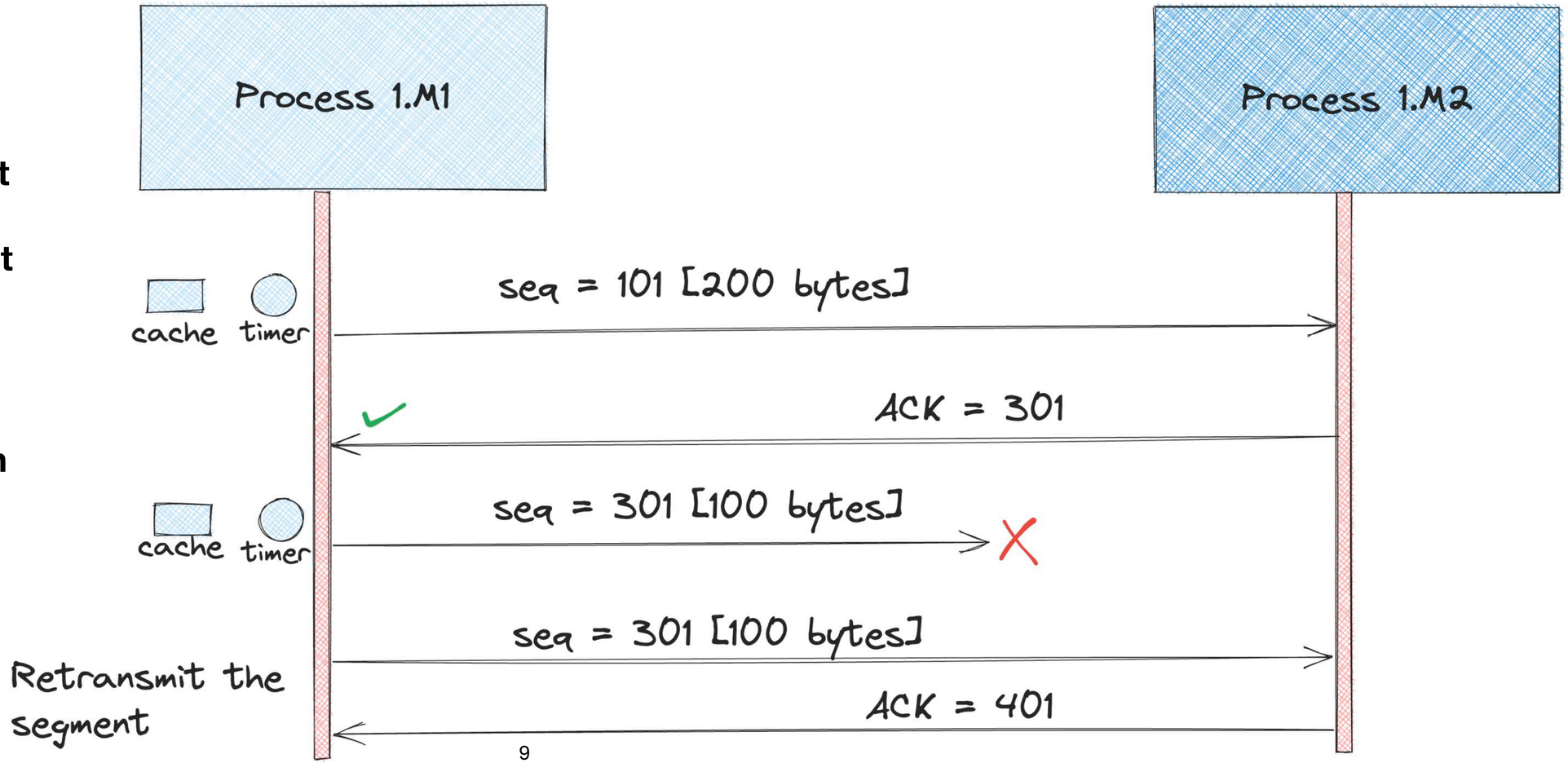
Person 2 Talking
(Process in a host B)

How to handle if data is lost?

Retransmission timeout also known as Round Trip Timeout (RTT)

TCP caches every data sent in a buffer (OS supports) Until retransmission timeout

What if ACK does not reach Back Process 1.M1?



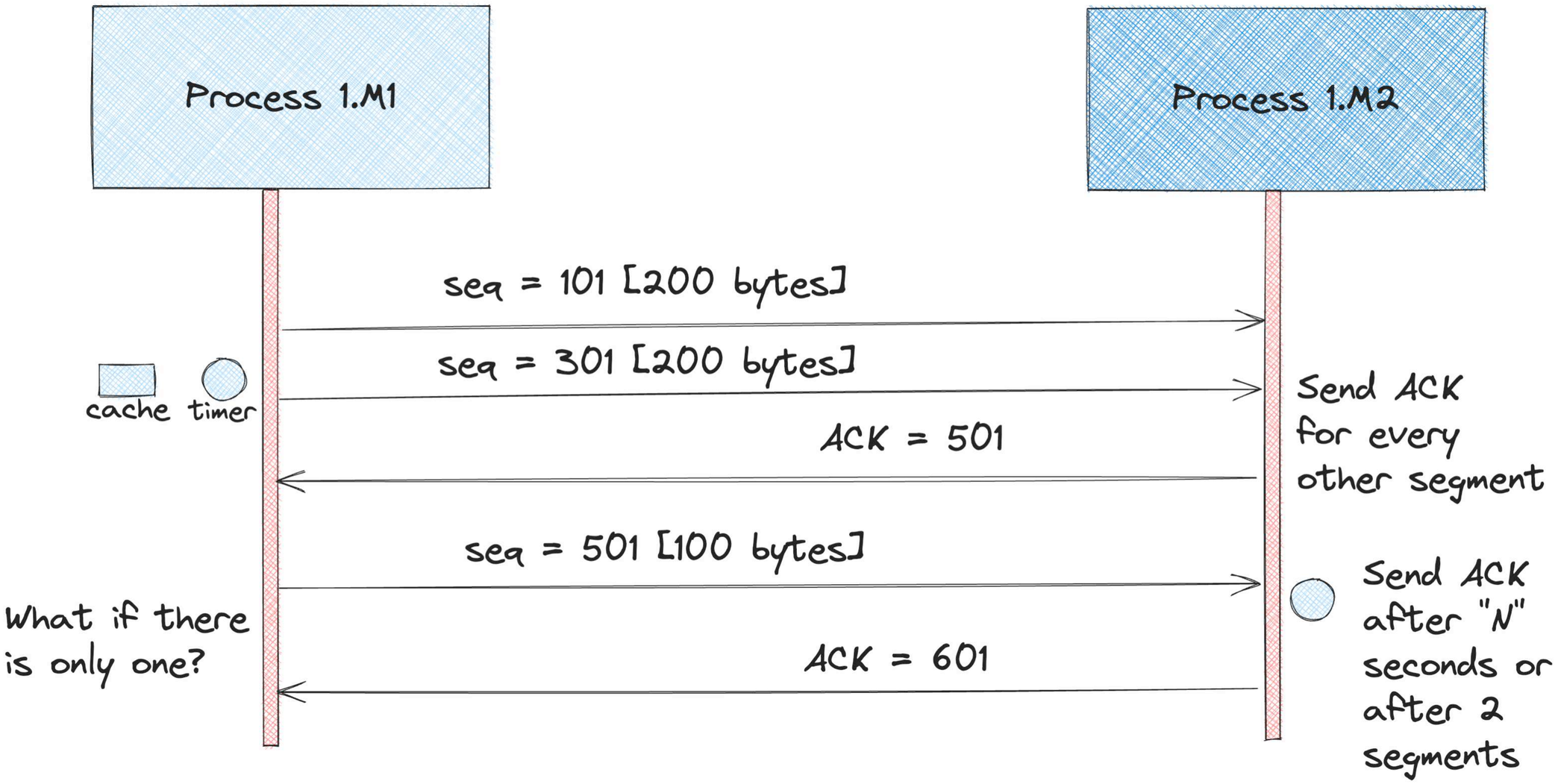
How to calculate RTT?

```
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

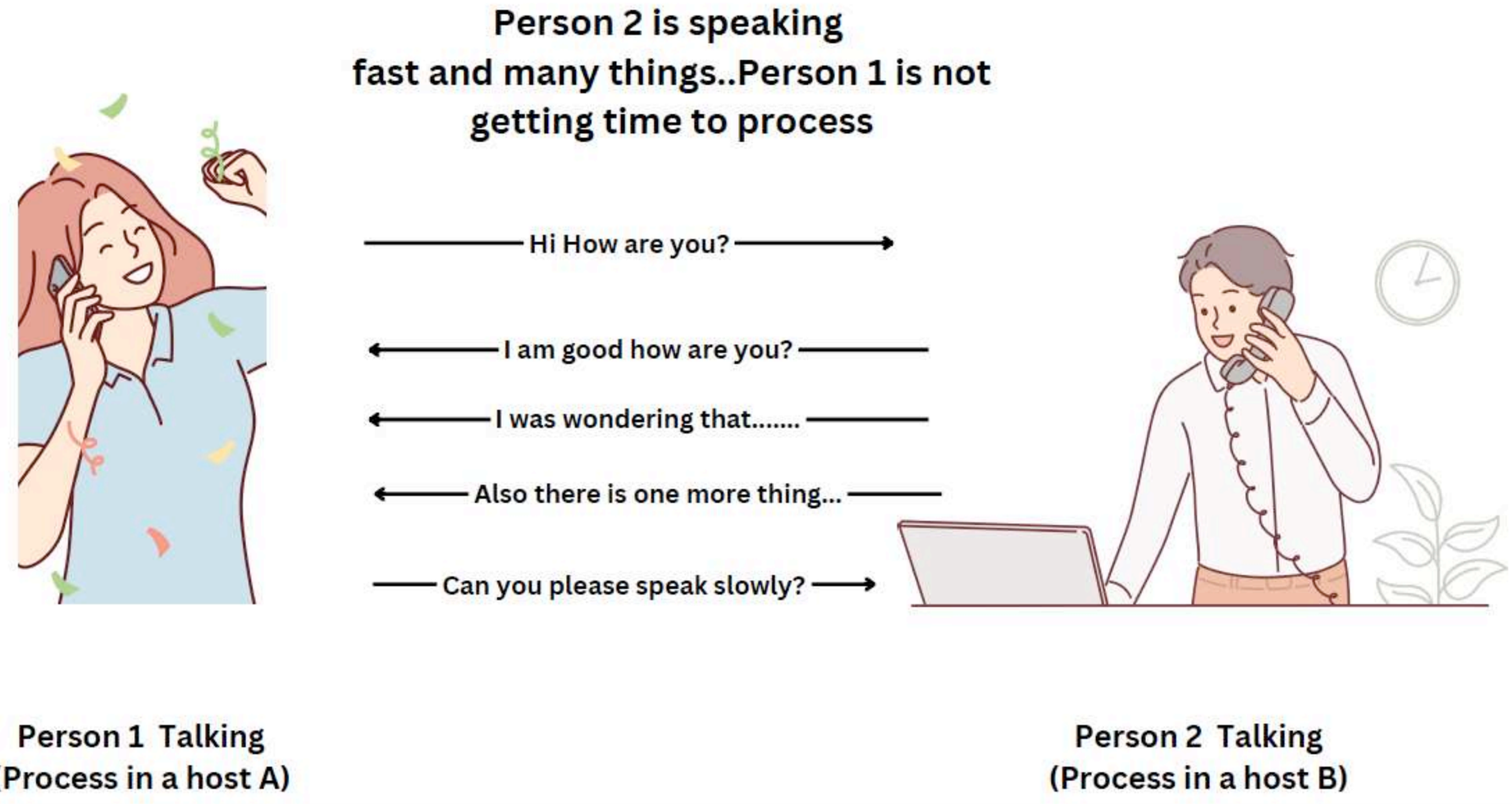
- **SampleRTT:** Time measured from segment transmission until ACK receipt
- **EstimatedRTT:** Estimated weighted moving average (EWMA) $\alpha = 0.25$
- **DevRTT:** EWMA of sampleRTT deviation from EstimatedRTT $\beta = 0.75$
- **TimeoutInterval:** Estimated Time plus some kind of safety margin

Do We need to Send ACK for each segment?

Use delayed acknowledgements

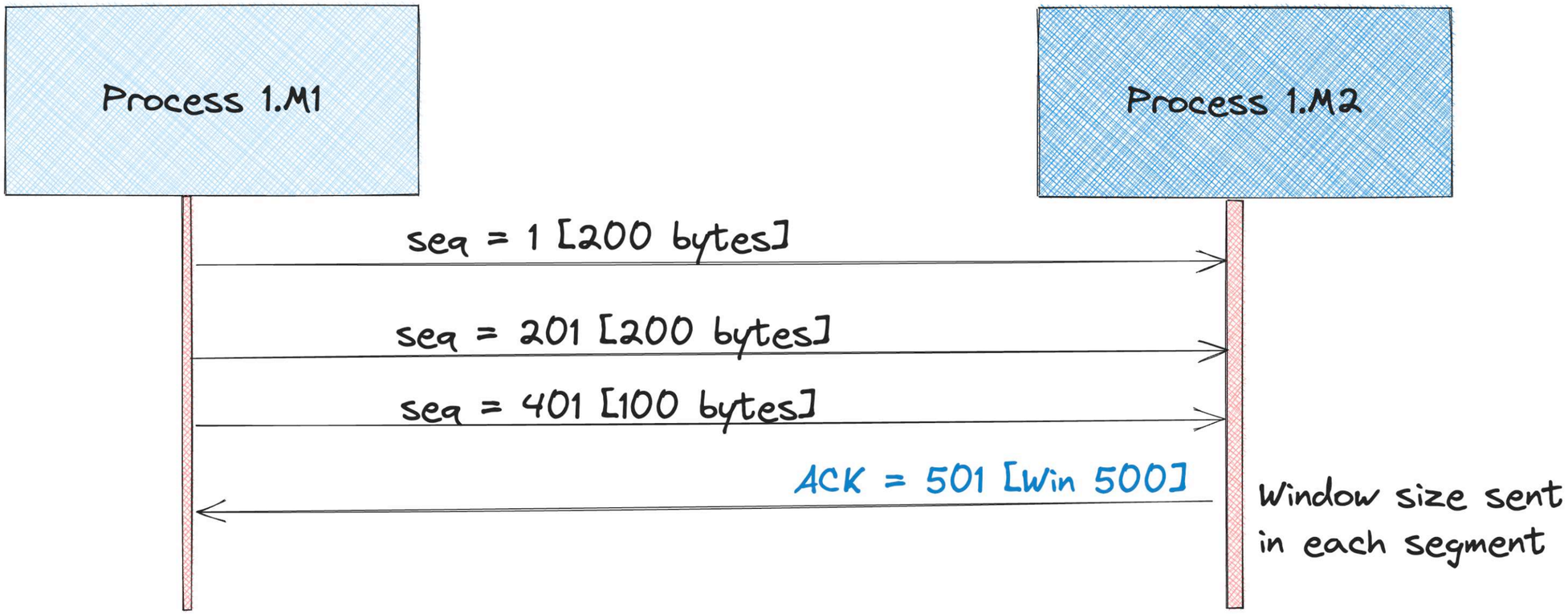


What if the speed is high?



Sending too much data is also problem

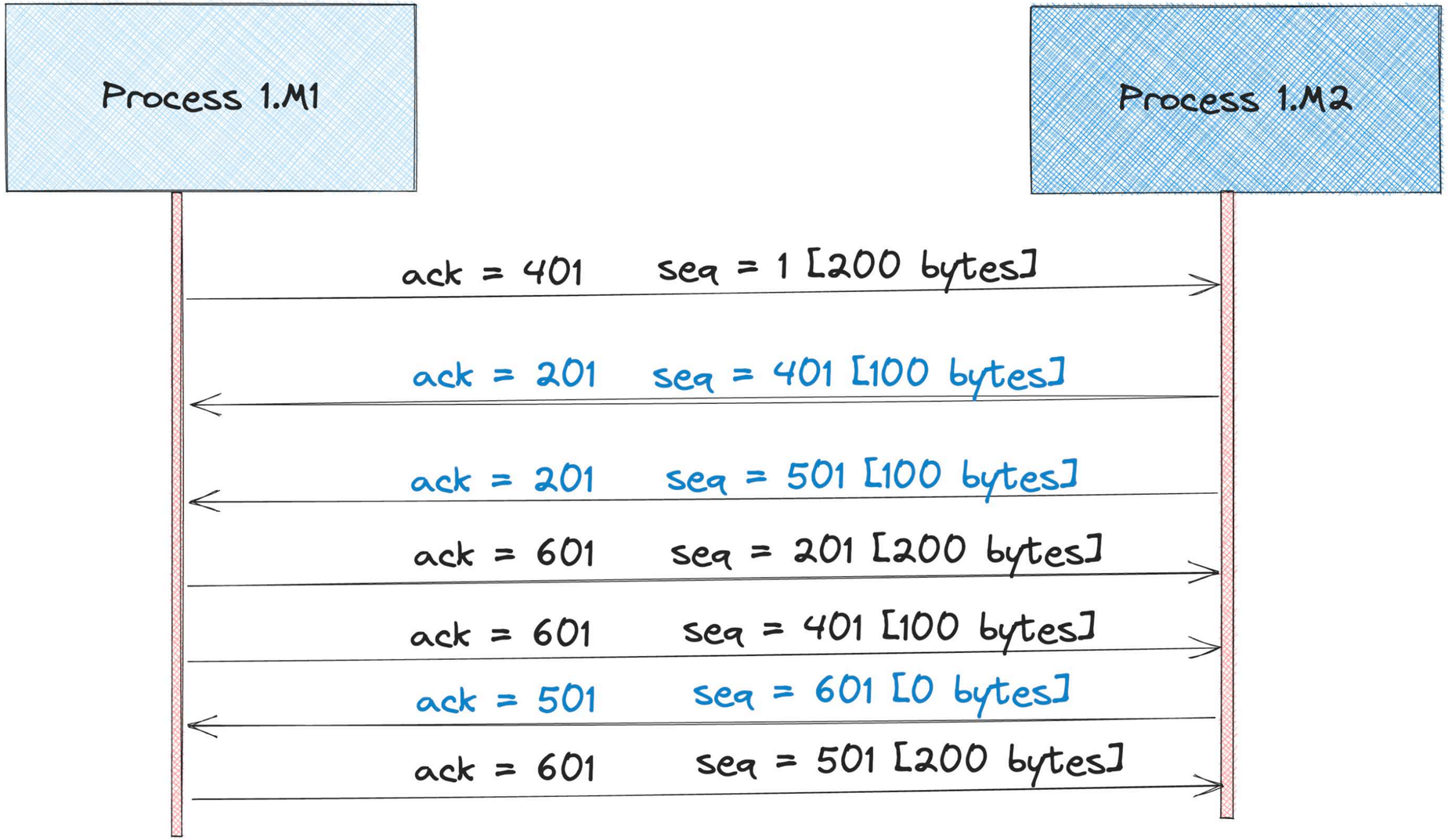
Window Size - Flow Control



- Dynamic update of Window size will enable flow control
- What if Process 1.M2 sends a windows size of 0?

TCP is bidirectional

Both Senders can send data

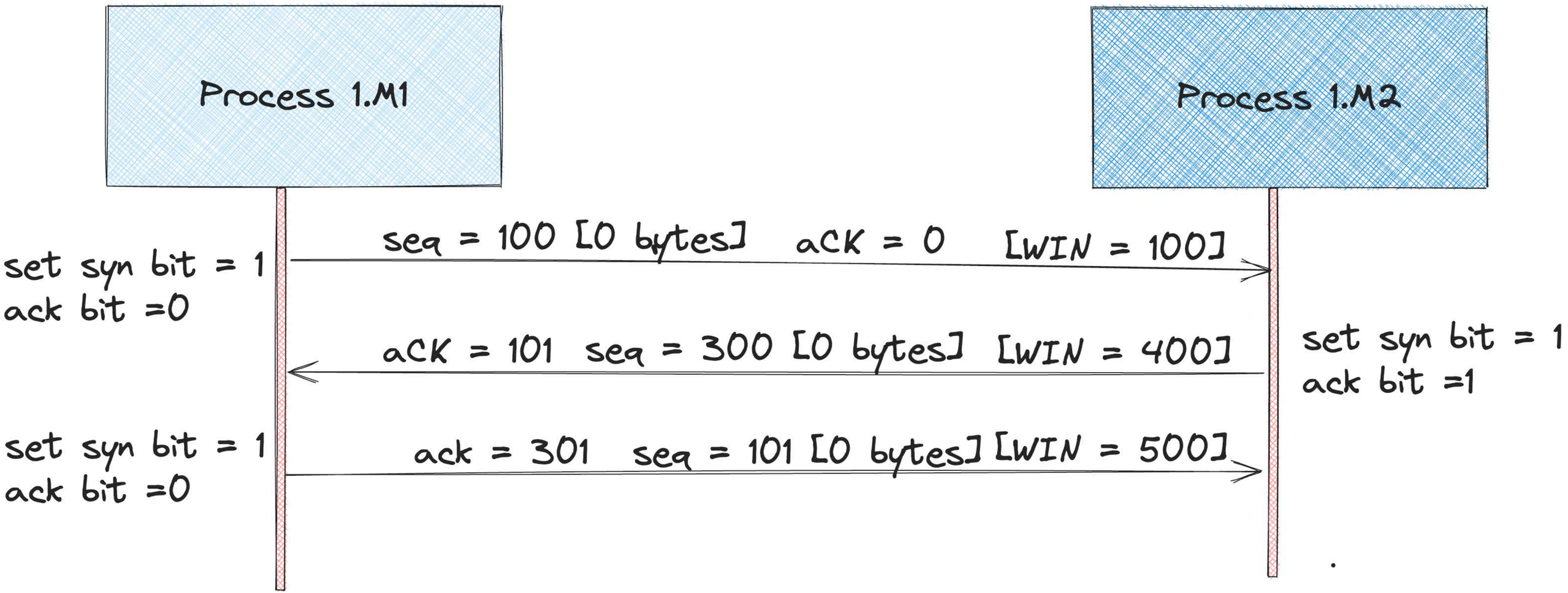


How to choose sequential numbers?

- Initial sequence numbers are randomly chosen by the senders
- Each can select a sequence number during the connection establishment
- Connection establishment in TCP happens through 3-way handshake
- The 3-way handshake consist of 4 events:
  - Process 1.M1 sends a connection request with SYN bit set and sequence number of X
  - Process 1.M2 acknowledges the connection request and sends back an ACK with X+1
  - Process 1.M2 also sends a request with the SYN bit set and sequence number [Y]
  - Process 1. M1 acknowledges the receipt by sending ACK [Y+1]

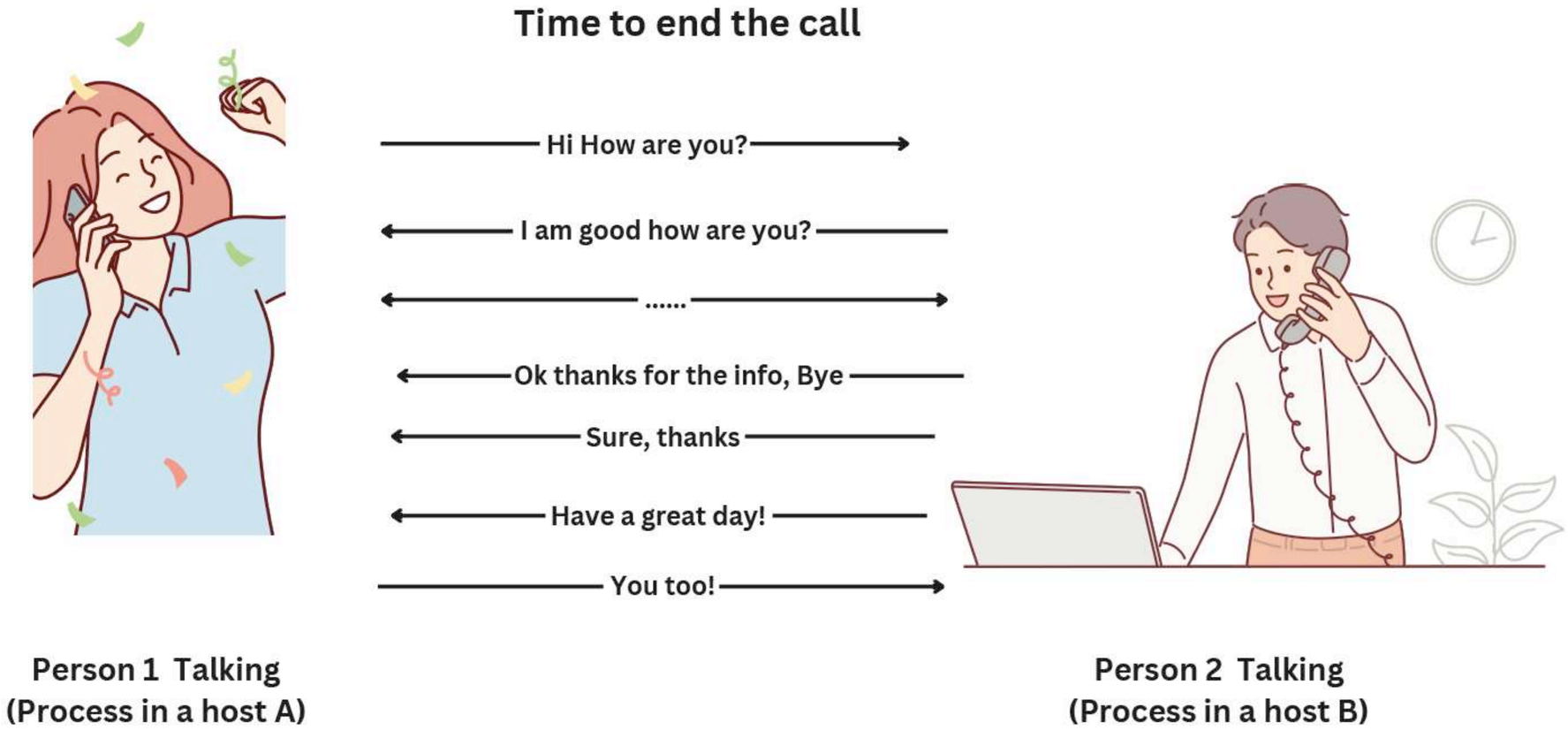
Three Way Handshake

Establishing Connection



Software Engineering Research Centre

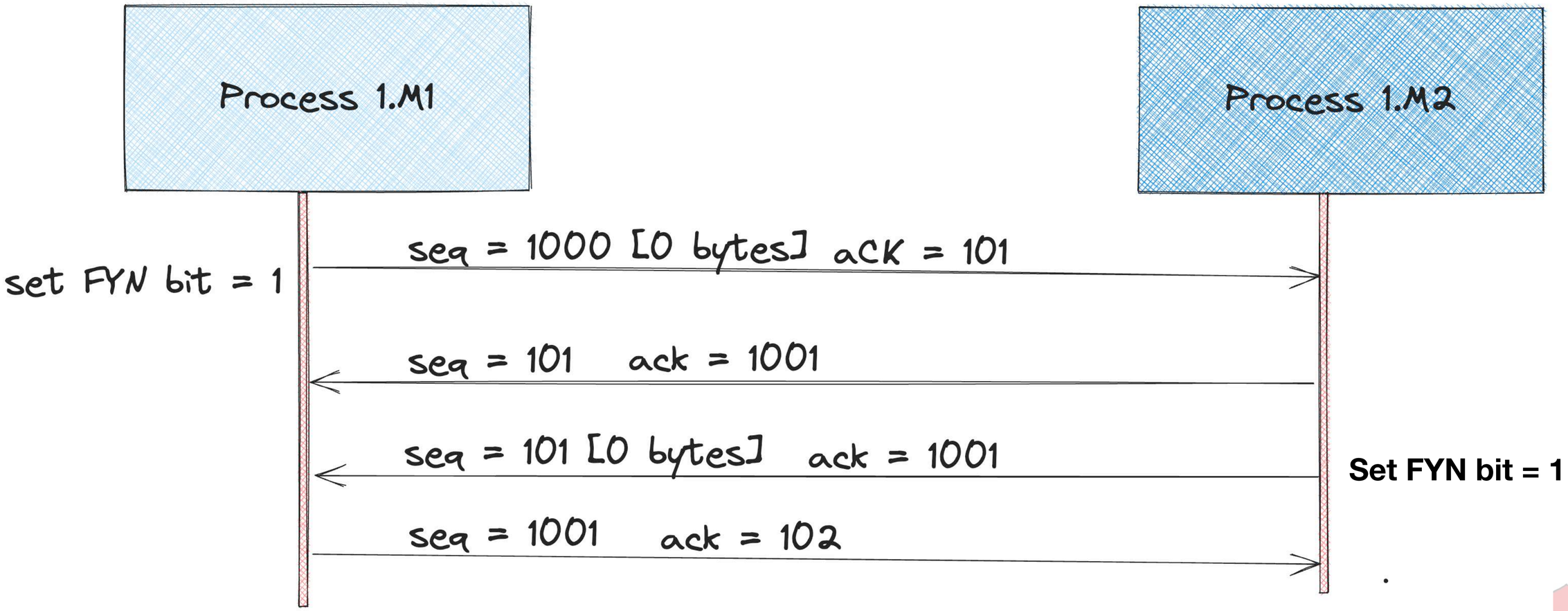
Closing Connection



- TCP has two ways to close connection: **FIN** and **RST** flags

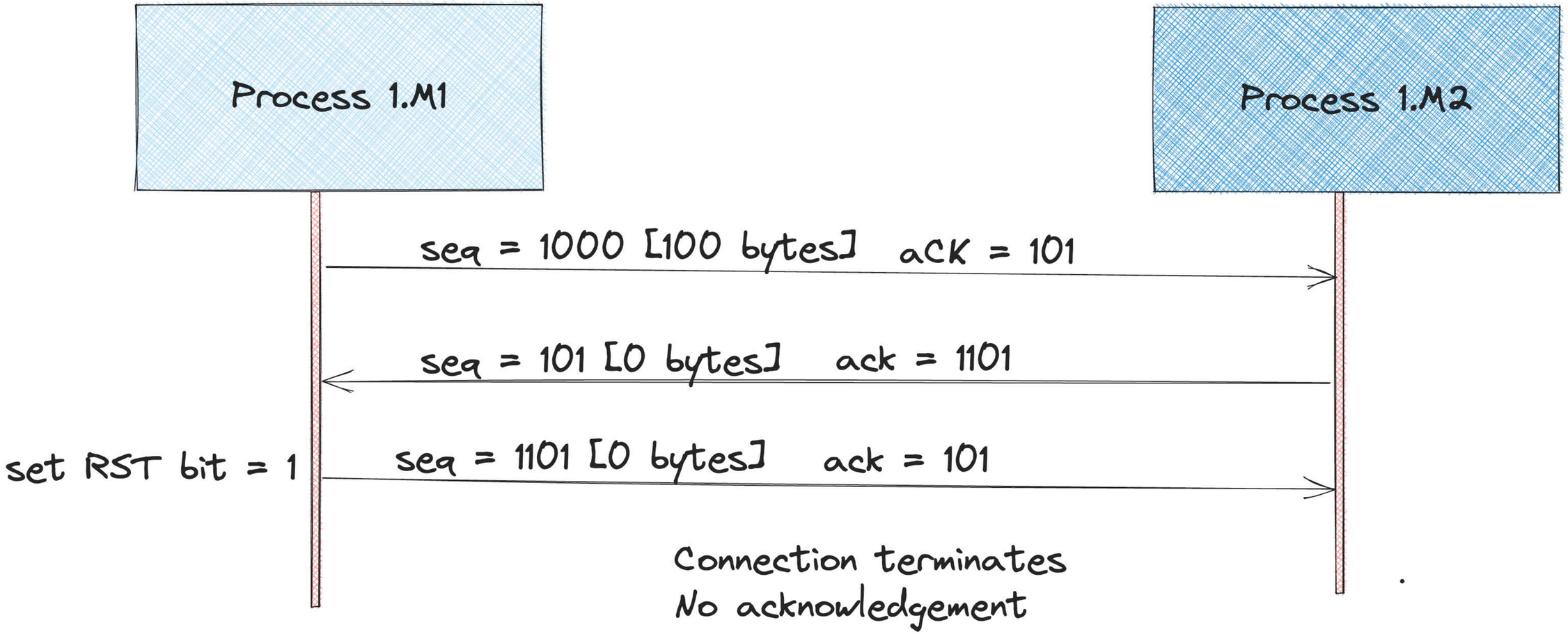
Using FYN bit

Graceful termination (Four-way closure)



Using RST Flags

Ungraceful closing



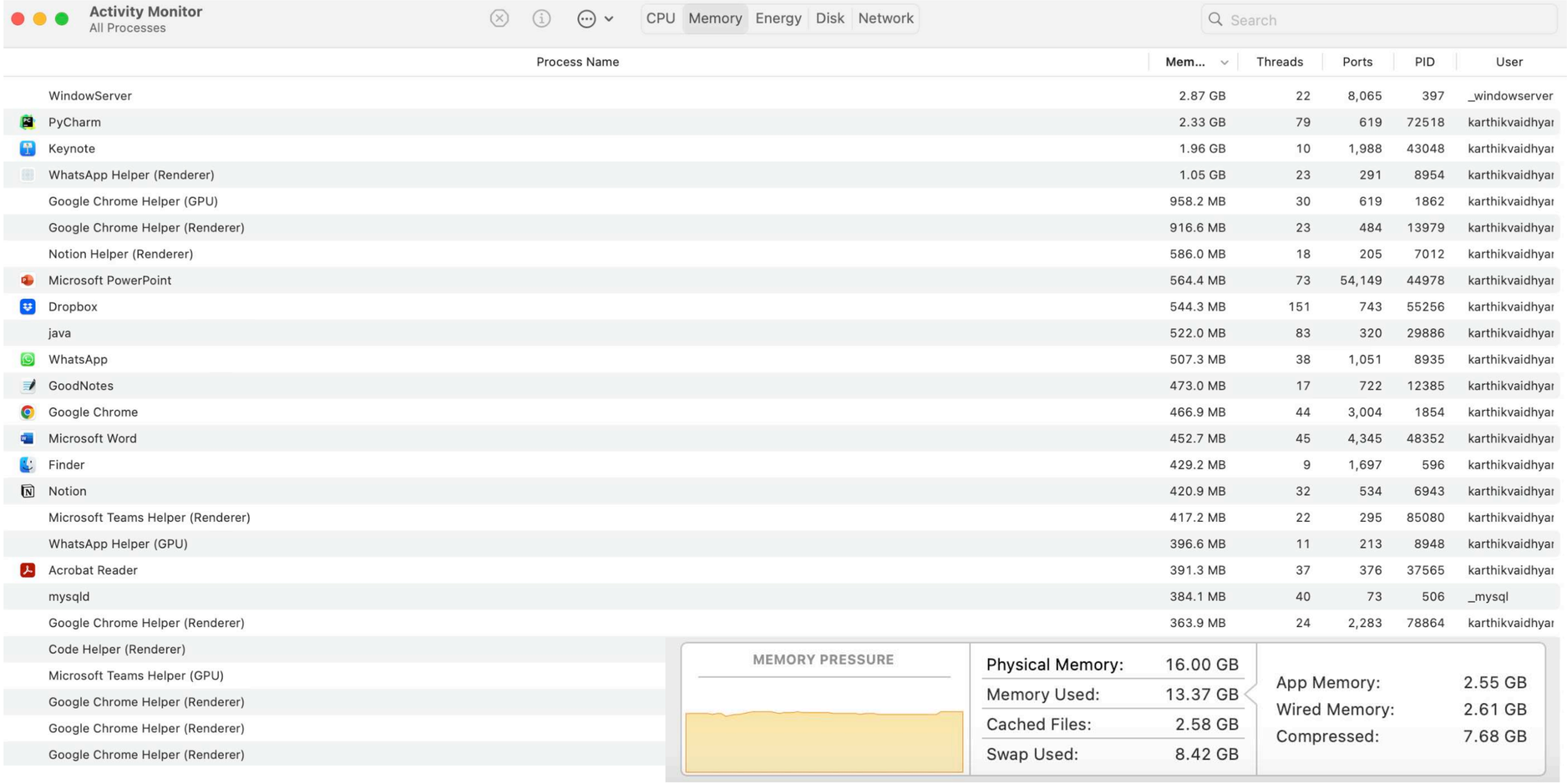
But we need Memory!

How does OS handle the memory requirements of all these?

Where is the process stored? What about network buffer?

Many processes run at the same time!

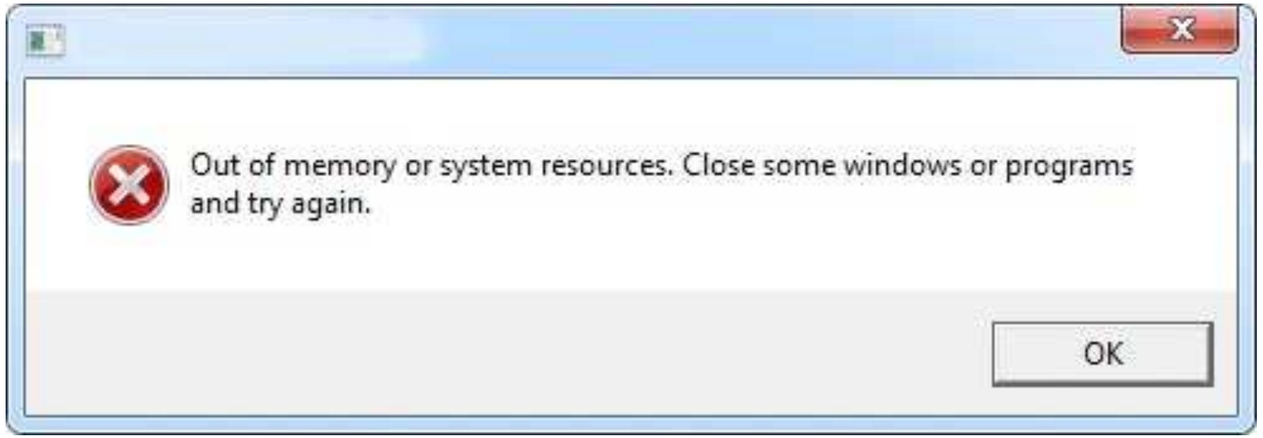
- What about Memory? Do we have enough Memory?



Process Name	Mem...	Threads	Ports	PID	User
WindowServer	2.87 GB	22	8,065	397	_windowserver
PyCharm	2.33 GB	79	619	72518	karthikvaidhyar
Keynote	1.96 GB	10	1,988	43048	karthikvaidhyar
WhatsApp Helper (Renderer)	1.05 GB	23	291	8954	karthikvaidhyar
Google Chrome Helper (GPU)	958.2 MB	30	619	1862	karthikvaidhyar
Google Chrome Helper (Renderer)	916.6 MB	23	484	13979	karthikvaidhyar
Notion Helper (Renderer)	586.0 MB	18	205	7012	karthikvaidhyar
Microsoft PowerPoint	564.4 MB	73	54,149	44978	karthikvaidhyar
Dropbox	544.3 MB	151	743	55256	karthikvaidhyar
java	522.0 MB	83	320	29886	karthikvaidhyar
WhatsApp	507.3 MB	38	1,051	8935	karthikvaidhyar
GoodNotes	473.0 MB	17	722	12385	karthikvaidhyar
Google Chrome	466.9 MB	44	3,004	1854	karthikvaidhyar
Microsoft Word	452.7 MB	45	4,345	48352	karthikvaidhyar
Finder	429.2 MB	9	1,697	596	karthikvaidhyar
Notion	420.9 MB	32	534	6943	karthikvaidhyar
Microsoft Teams Helper (Renderer)	417.2 MB	22	295	85080	karthikvaidhyar
WhatsApp Helper (GPU)	396.6 MB	11	213	8948	karthikvaidhyar
Acrobat Reader	391.3 MB	37	376	37565	karthikvaidhyar
mysqld	384.1 MB	40	73	506	_mysql
Google Chrome Helper (Renderer)	363.9 MB	24	2,283	78864	karthikvaidhyar
Code Helper (Renderer)					
Microsoft Teams Helper (GPU)					
Google Chrome Helper (Renderer)					
Google Chrome Helper (Renderer)					
Google Chrome Helper (Renderer)					

Real View of Memory can be Messy!

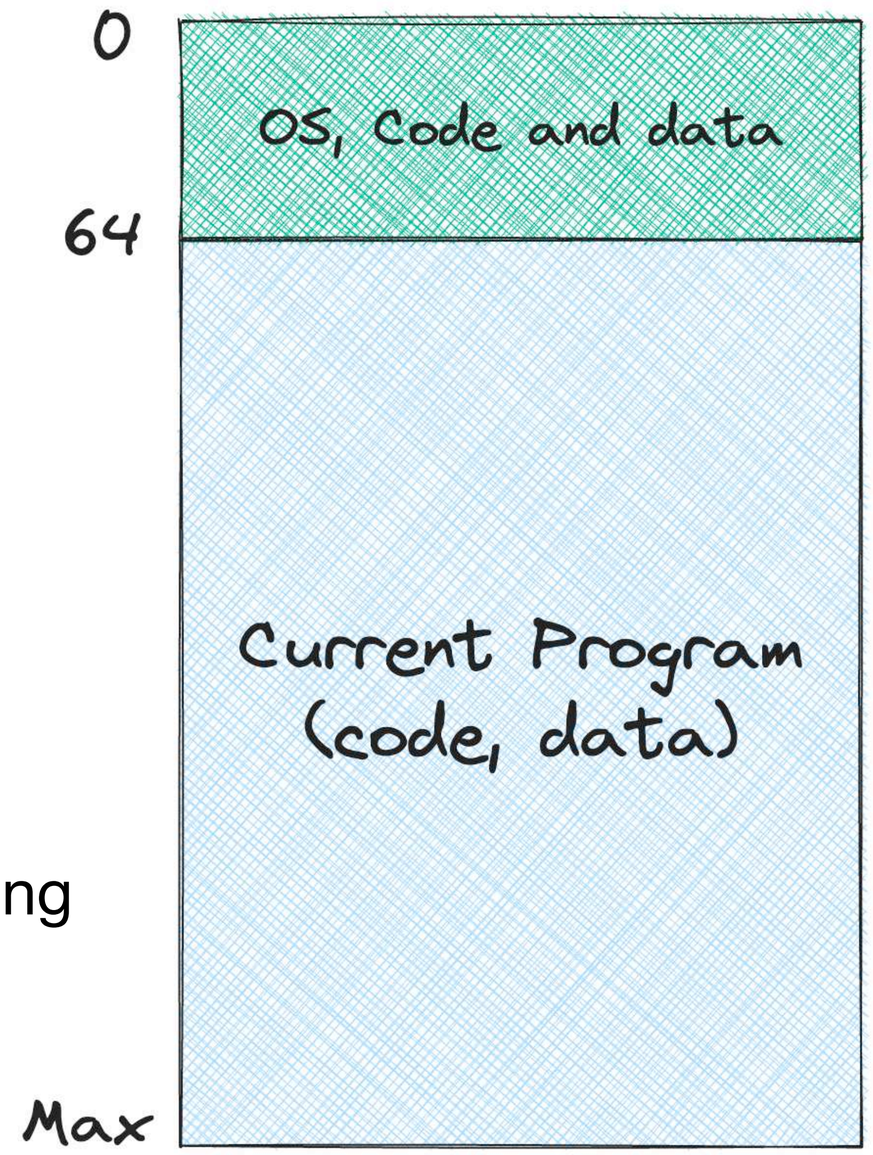
Managing it can be even further difficult



Source: Google Images

Memory Virtualization

- Early days OS had just one program
- OS, its code and data resides in one part
- The running program, its code and data resides in one part
- Does it work today?
  - Today its about multiple processes
  - Run process for sometime save everything to disk, run next - **Problems?**
  - OS provides process virtualisation



Only One running Program

Memory Virtualization: Why?

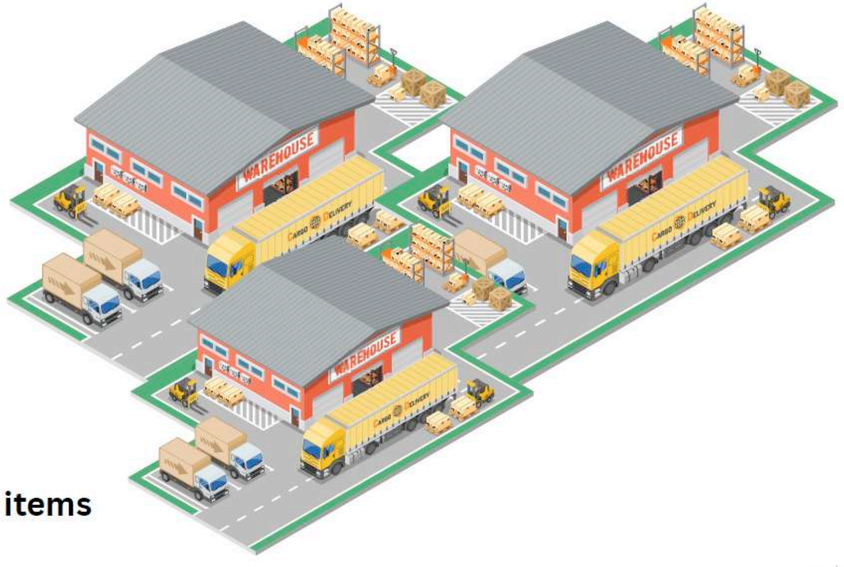
- We need to think about multiple processes
- Need to increase utilisation and efficiency
- Particularly useful in olden times when it costed millions of dollars for machines
- Soon came era of time sharing
- Batch computing was not anymore appreciated
- Instead of saving in the disk, can we keep the process on disk itself?

An Analogy

Onsite Shopping

Every users have access to different items but to a limited set

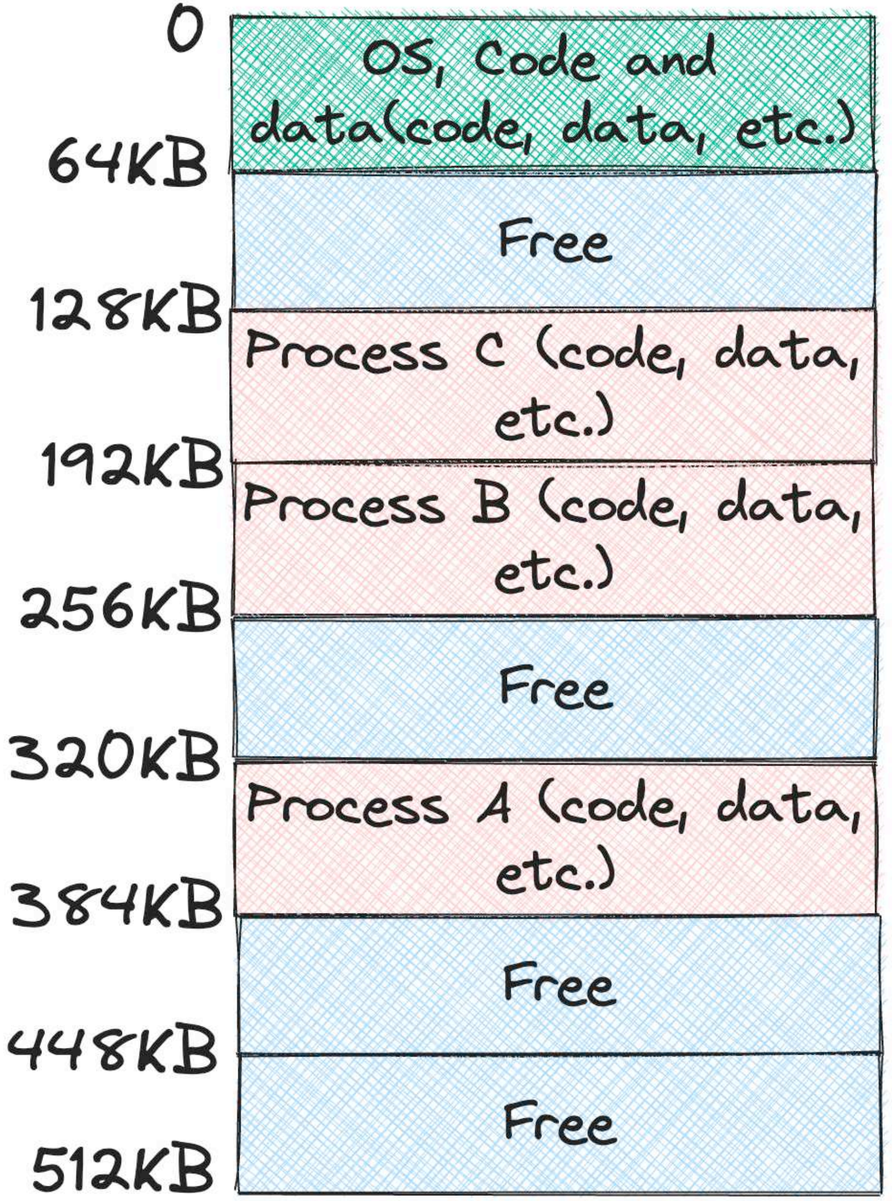
Online Shopping



Every Users feel that they have access to infinite items

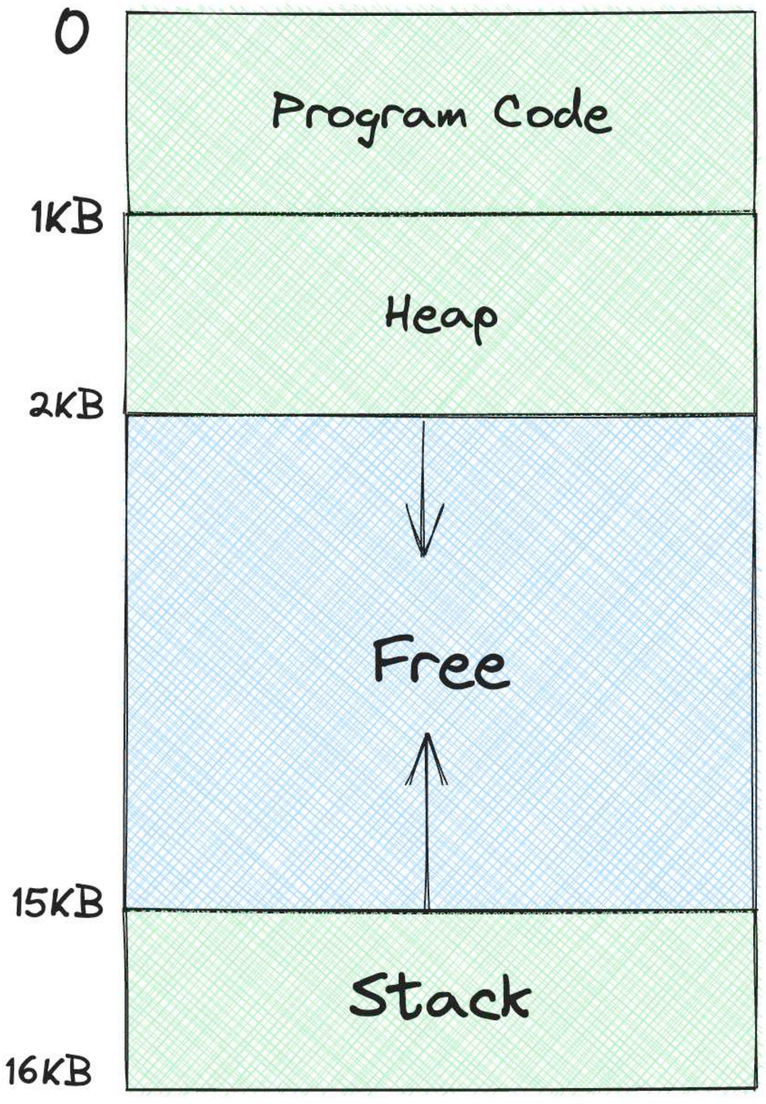
Keep Process in the Memory

- Each process is given a dedicated location
- There are multiple free spaces where process can be added
  - Main challenge: We don't want any process to read any other process data
  - Real life OS has 100s of process that will be running
  - Giving control to user may make it hard



Abstraction: Virtual Address Space

- OS creates easy to use abstraction of the physical space
- Address space (Memory image of process)
  - Program Code (and static data)
  - Heap - Dynamic memory allocations (malloc)
  - Stack - Function calls during runtime
  - The stack and heap grow during runtime
- Every process assumes that it has access to large block of memory from 0 to MAX
- CPU issues loads and stores to virtual addresses

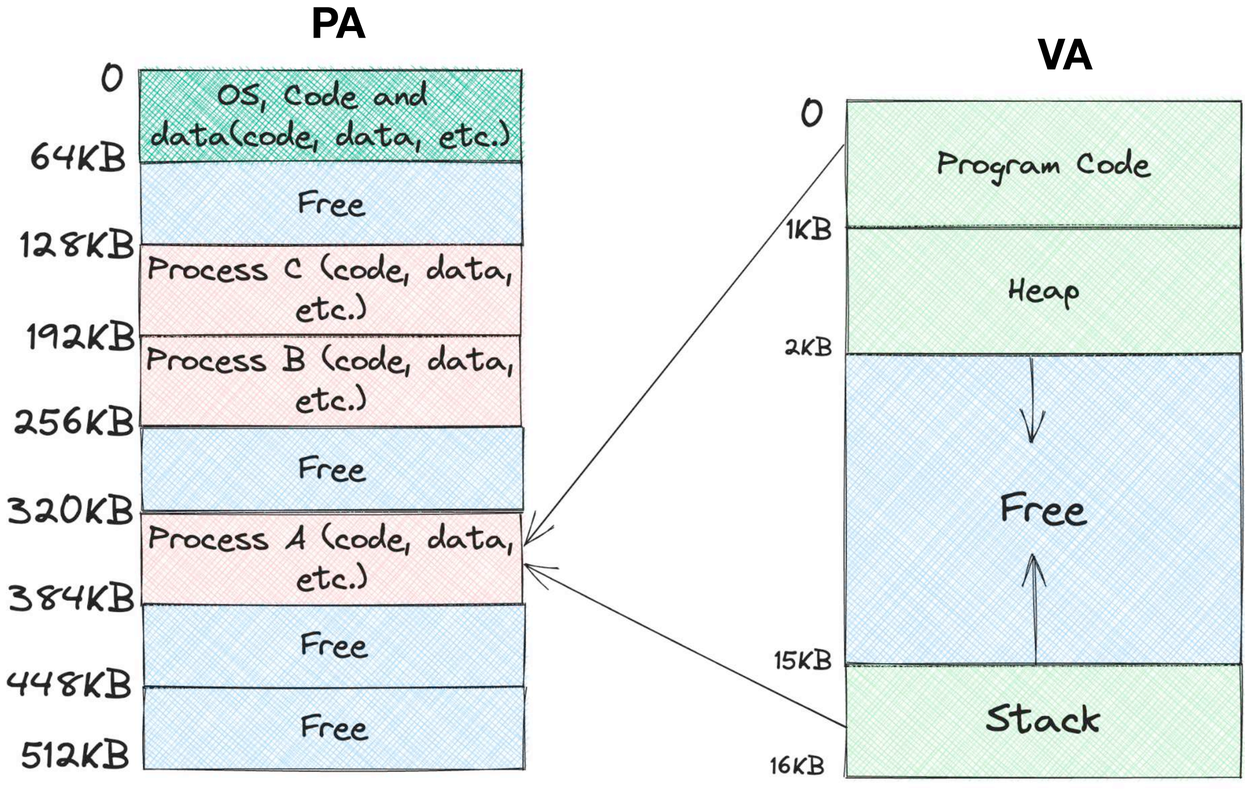


There is only one physical memory

- How can OS build the abstraction of a private large address space on top of single physical memory?
  - There is only one physical memory, process feels has it has its own starting at 0
  - When a process tries to load from a particular location, **K** (0)
    - OS with some hardware support ensures that the load doesn't go to actual location
    - Rather to the physical address **Z** (320) - **Virtualization**

How actual memory is reached?

- Address translation from virtual address (VA) to physical address (PA)
  - CPU loads/stores to VA but memory hardware access PA
- OS allocates memory and tracks the location of the process
- Translation is done by Memory Management Unit (MMU)
  - OS makes necessary information available



SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Memory Virtualization - Dynamic relocation and Segmentation

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:

- Operating Systems: In three easy pieces, by Remzi et al.

Goals of Virtualization

- **Transparency**
  - Illusion that physical memory is not visible to any processes
  - Take away worry from the user program about what happens behind scenes
- **Efficiency**
  - Minimize overhead in terms of space and access time
- **Protection**
  - Protect process from one another even OS itself
  - Each process must be running its own isolated cocoon safe from malicious process

Memory API

- For process virtualization, we learned about APIs to create, destroy, duplicate processes
  - What about memory?
  - Can we think of some ways to do it?
  - What are the interfaces for it?
  - What are some common pitfalls that needs to be avoided?

Memory Allocations and Deallocations

First Type of Memory Allocation

- In C program, two types of memory allocation happens
  - Stack Memory
    - Allocations and deallocations are managed implicitly by compilers
    - Called **Automatic memory**
    - Once execution is done, compiler deallocates memory
- Static/global variables are allocated in executable

C Program Snapshot

```c
void functionName()
{
  int x; // declares an integer on the stack
  ...
}
```

Memory Allocations and Deallocations

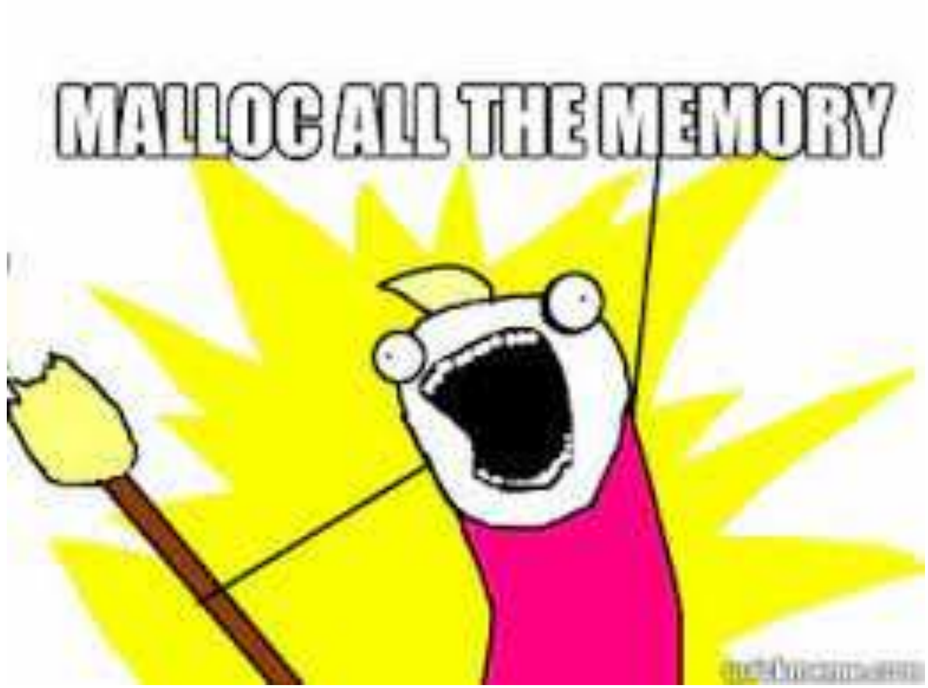
Second type of Memory Allocation

- Heap memory
  - Allocations and deallocations are handled explicitly by the programmer
  - **malloc()** requests for space of integer on the heap
  - The routine returns the address of the integer
  - Heap memory is more challenging to play with

C Program Snapshot

```
void functionName()
{
  int *x = (int *)malloc(sizeof(int));
  ...
}
```

The malloc() call



The malloc() call

- Quite a simple call
- Just pass as parameter, the size required in the heap (**size_t**) - Number of bytes
- The call will return pointer to new space
  - Returns NULL on failure
  - Under library stdlib.h
  - For allocating double precision value:

```
double *d  = (double *)malloc(sizeof (double));
```

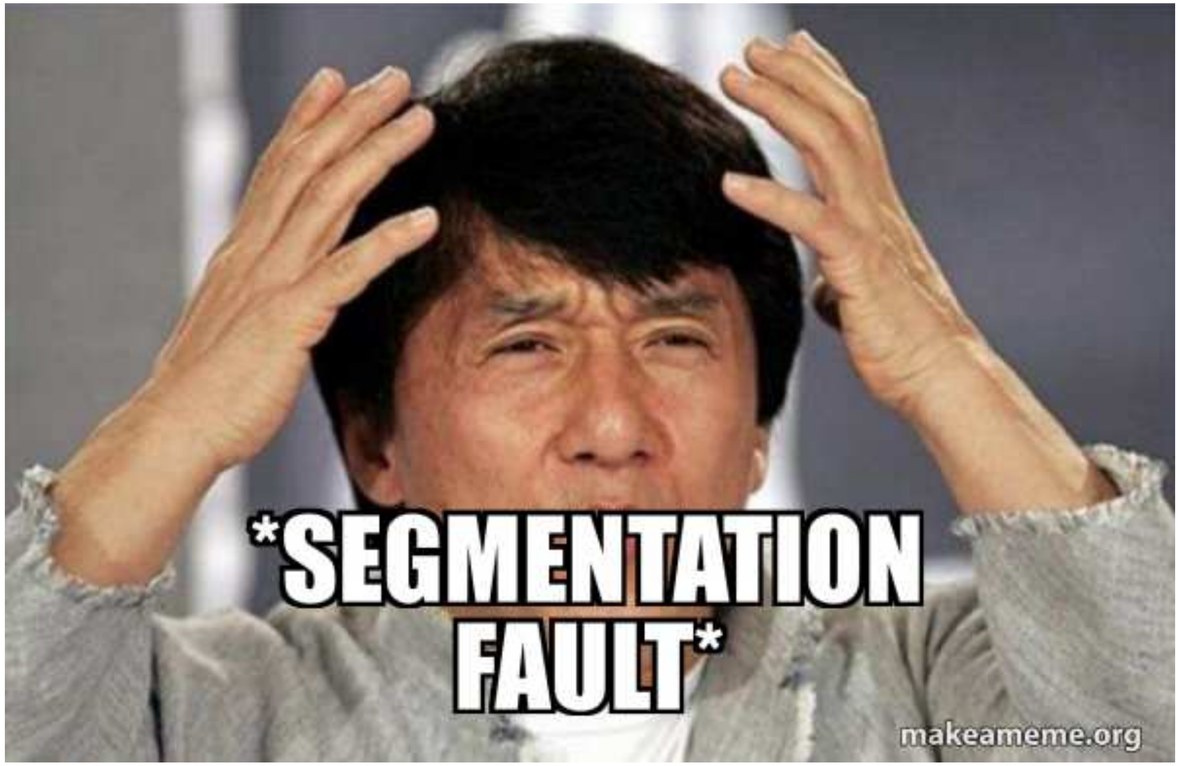
Free() call

- Free the heap memory
- Takes as argument the pointer returned by malloc.
  - The size of allocated region is not passed by user
  - Tracked by the memory allocation library itself
- Not enough we do malloc
  - Its very important to free it - why?

Common Errors made by Programmers

- Lot of errors arise in the usage of malloc() and free()
- Error free memory management has always been a problem
  - Modern programming languages support it implicitly
  - Most of the times we may call something similar to malloc()
  - Free is not called in most languages by programmers
    - Garbage collectors in Java

Ever Come Across This?



Error 1: Forgetting to allocate memory

- Many routines expect memory to be allocated before invoked

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *str = "hello";
  char *dst;
  strcpy(dst, str);
  return 0;
}
```

Is there some issue?

Segmentation Fault!!

Not allocating enough Memory

Yes, this can also be a problem

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *str = "hello";
  char *dst = (char*)(malloc(strlen(src)));
  strcpy(dst, str);
  return 0;
}
```

- Depending on how malloc is implemented, this may work more often
- **strcpy** may write one byte past the allocated space
- This may result in **overflow - It ran correctly doesn't mean its correct!**

Forgetting to Initialize Allotted Memory

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *src = "hello";
  char *dst = (char*)(malloc(strlen(src)));
  printf("%s\n", dst);
  return 0;
}
```

- malloc() is called properly but no value assigned
- May result in an error -> **Uninitialized read**
- It may read some data of unknown value from the heap => program will be affected!

Forgetting to Free Memory

- Results in **Memory Leak**
- Occurs when one forgets to free memory after use
- Slowly leaking memory => system runs out of memory => **System restart!!**
- When done with chunk of memory - free it off
- Best solution: Ensure program exits! OS will clean up everything

Freeing Memory before the completing the use

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *src = "hello";
  char *dst = (char*)(malloc(strlen(src)+1));
  free(dst);
  strcpy(dst,src);
  printf("%s\n", dst);
  return 0;
}
```

- Calling free before using it
  - Subsequent call of the pointer can crash the program or overload memory
- Results in a **potential error** due to **Dangling Pointer**

Freeing More than once

Too much of anything is dangerous!!

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *src = "hello";
  char *dst = (char*)(malloc(strlen(src)+1));
  strcpy(dst,src);
  printf("%s\n", dst);
  free(dst);
  free(dst);
  return 0;
}
```

- Free memory more than once
  - **Double free** error
- May result in undefined issues - Memory allocation library may get confused

Calling free incorrectly

Strcpy on two strings

```
int main (int argc, char *argv[])
{
  char *src = "hello";
  char *dst = (char*)(malloc(strlen(src)+1));
  strcpy(dst,src);
  printf("%s\n", dst);
  free(src); //Passing src instead of dst
  return 0;
}
```

- free() expects to get the pointer returned from malloc() as input
- When another value is passed, bad things happen
- **Invalid free** needs to be avoided

Common Issues with Memory

- Lots of issues with memory exist and abusing of memory happens
  - Lots of tools exist to solve issues - valgrind, purify, etc.
- malloc() and free() are not system calls rather just library calls
  - stdlib.h - library in C that provides functions malloc and free
  - Built on top of system calls - brk or sbrk
  - Brk or sbrk increases or decreases the size of heap based on value
  - Not advised to call them directly

More on Memory related APIs

- Another system call that can be used is mmap()
  - Creates anonymous memory region within the program
- Variations of malloc() exist
  - calloc() -> allocates memory and initialises with 0's.
  - realloc() -> add something more to the existing space allocated with malloc()

The Big Question: How to Virtualise

- Each process requires memory
- OS performs context switch between processes
- Process should not overwrite each others memory
- Users should not worry about memory allocations and where to store
- OS needs to virtualise memory
  - Can we do something similar to process virtualisation?
  - What are the two key aspects that enabled process virtualisation?

Memory Virtualisation: Key Requirements

- Bring hardware into the picture (similar to LDE)
  - Use some hardware support for memory management - **efficiency**
- OS can play its role when it comes to controlling
  - Ensuring that no application has direct access to memory by its own
  - Keep track of which locations are free and which are in use - **control**
- There should also be flexibility
  - Allow programs to use address space in the ways they like

The Overall Goal

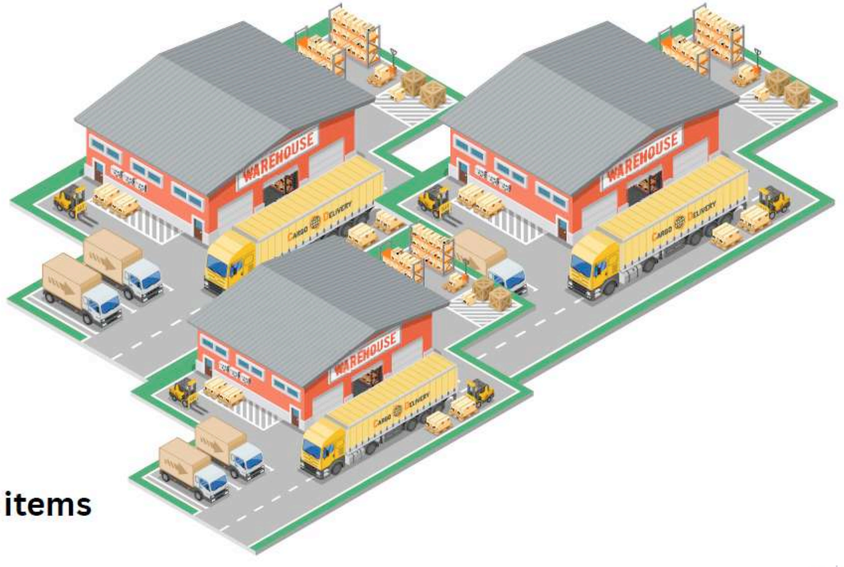
- **Goal:** Create an illusion that each process has its own private memory where the code and data reside
  - Reality: Many processes are actually sharing memory at the same time!
- How to make this happen? - Three Key assumptions:
  - User address space must be placed contiguously in physical memory
  - Size of address space is not too big; less than size of physical memory
  - Each address space is of exactly the same size

Memory Virtualization: An Analogy

Onsite Shopping

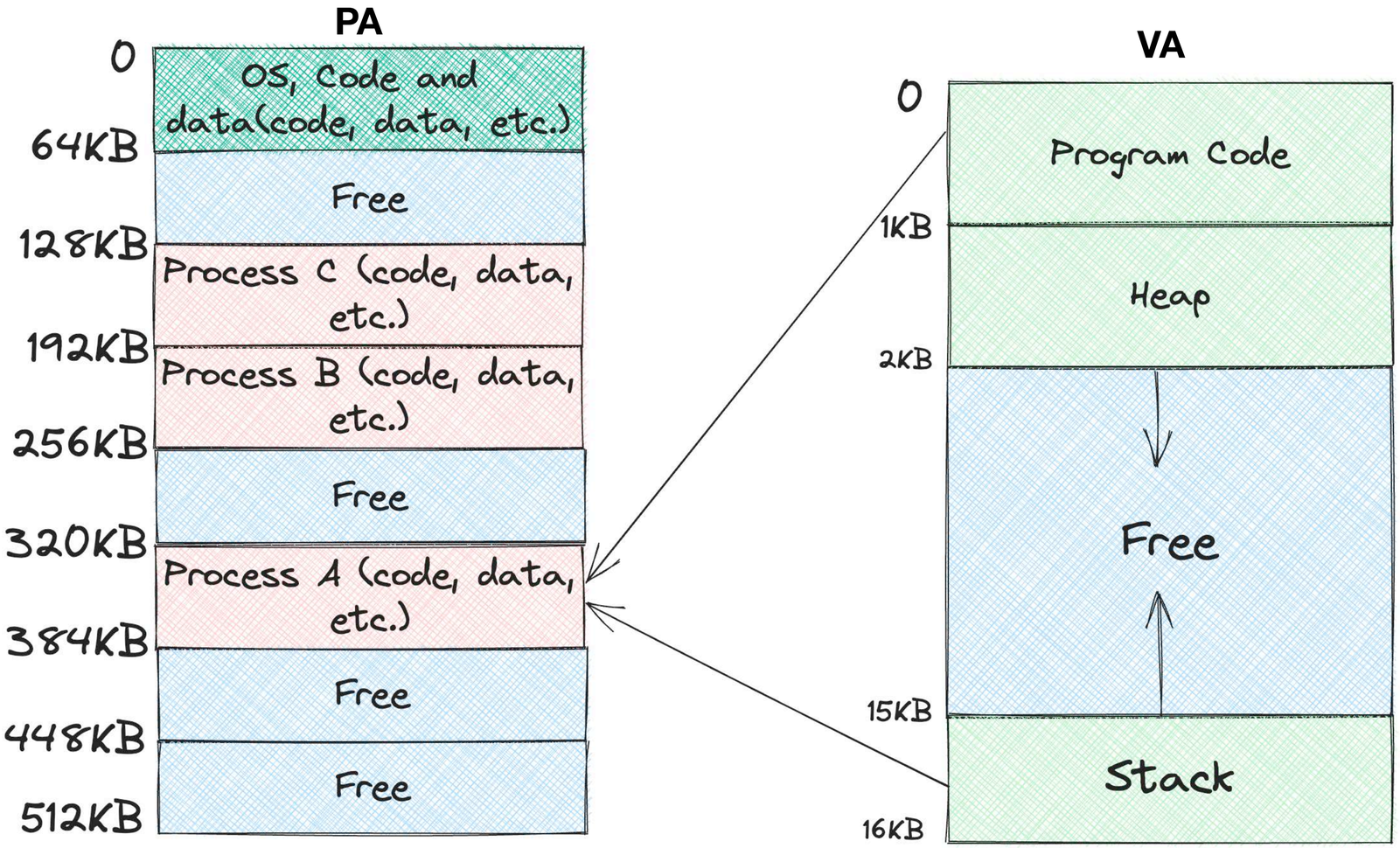
Every users have access to different items but to a limited set

Online Shopping

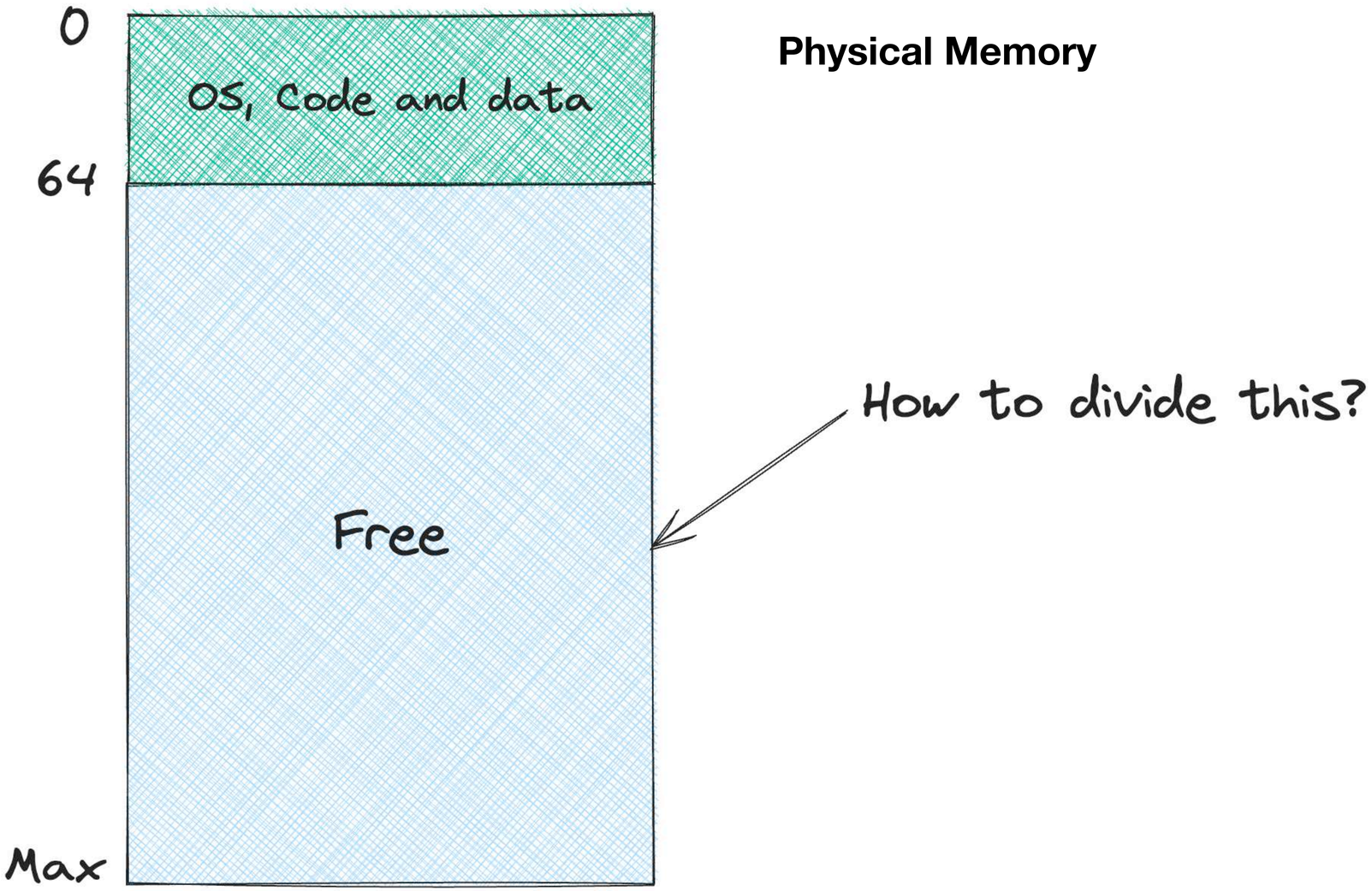


Every Users feel that they have access to infinite items

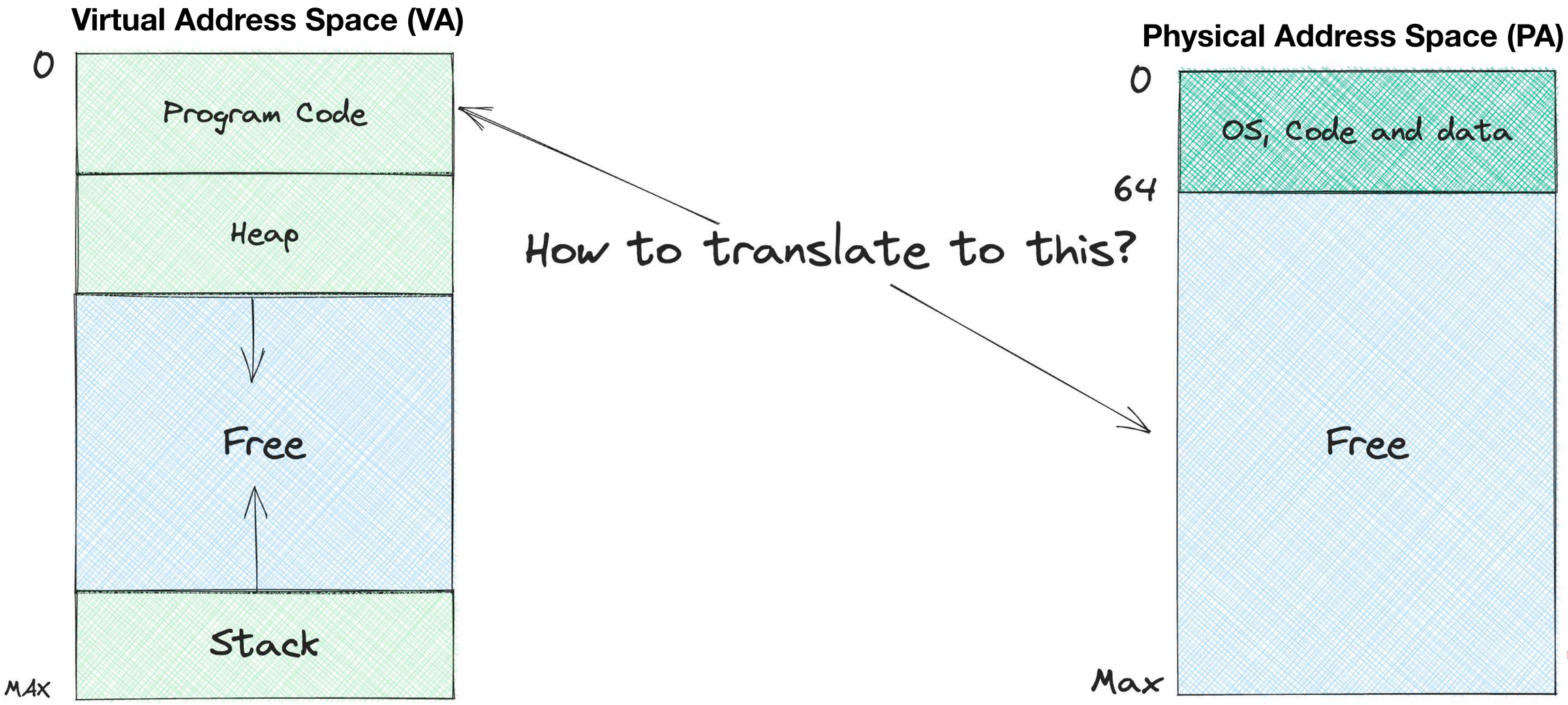
Address Translation — Recap



Essentially its about two things!



Essentially its about two things!



Simple Program

C Program to Assembly

Sample Program

```c
void func()
{
  int x;
  ...
  x = x + 3;
}
```

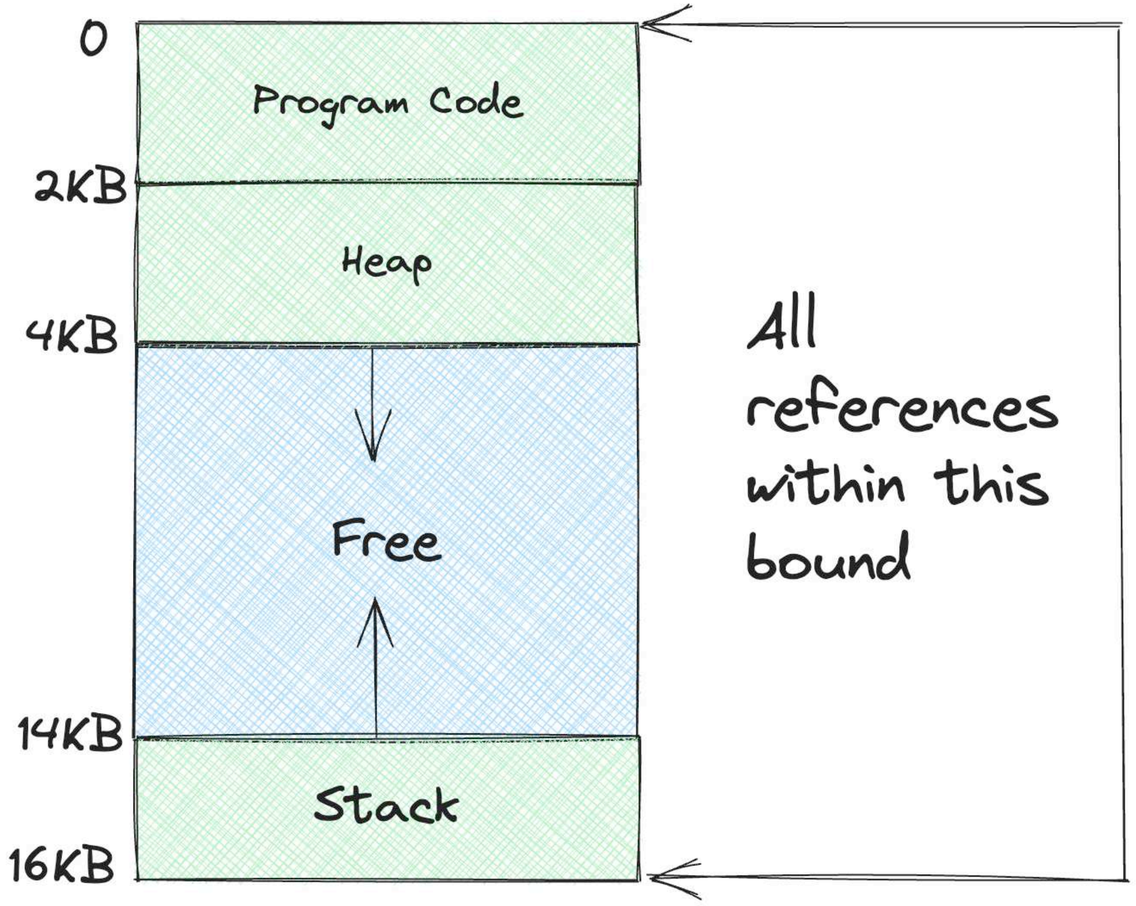
Assembly Code

```
128: movl 0x0, %eax ;load 0+ebx into eax
132: addl 0x3, %eax ;add 3 to eax register
135: movl %eax, 0x0 (%ebx) ;store eax back to mem
```

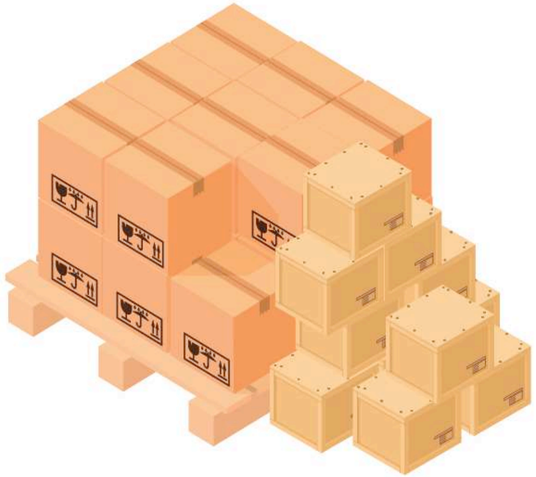
int x = 3000;

Following Process happens

1. Fetch instruction at 128
2. Execute the instruction (load address)
3. Fetch instruction at 132
4. Execute the instruction (No memory reference)
5. Fetch instruction at 135
6. Execute the instruction (Store to 15 KB)



Warehouse Scenario



Warehouse with lots of new packages/shipments

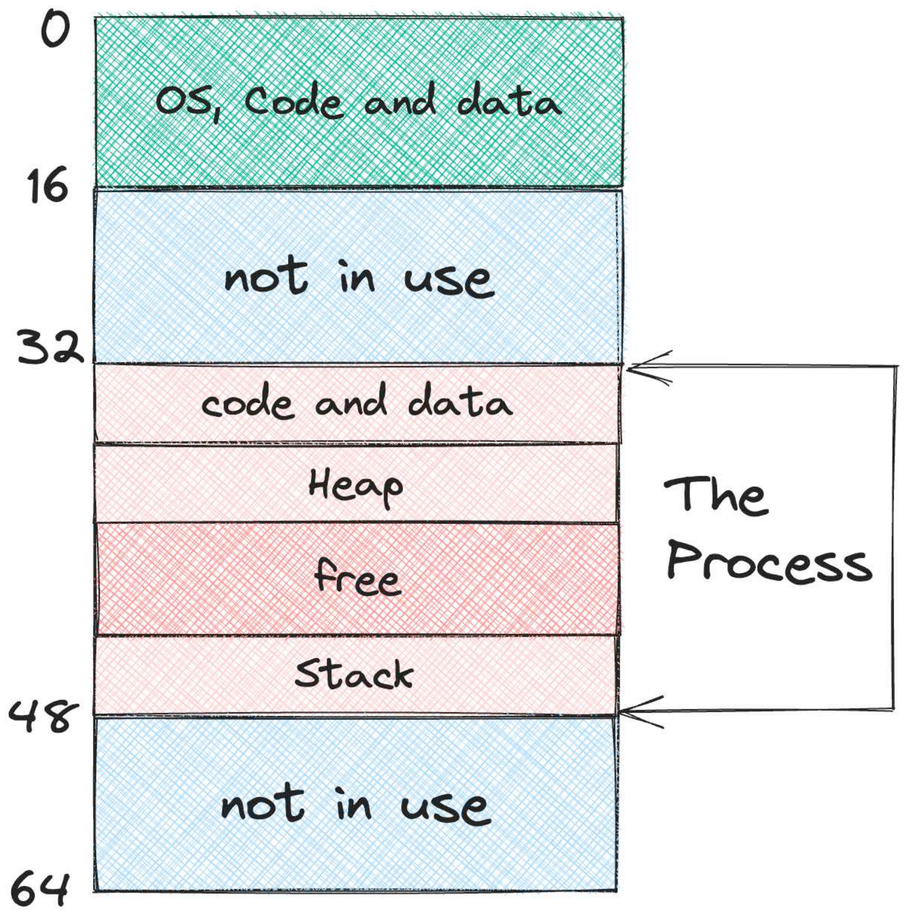
Based on a Category: Range can be decided
Category like Electronics, Clothing, etc

They can be grouped - Each type of shipment can be grouped in a range of locations (0 - 200: Electronics)

Manager/other staff: Simply go to the corresponding range to find the product - There is a starting and ending value

Can we not do this at Physical Memory Level?

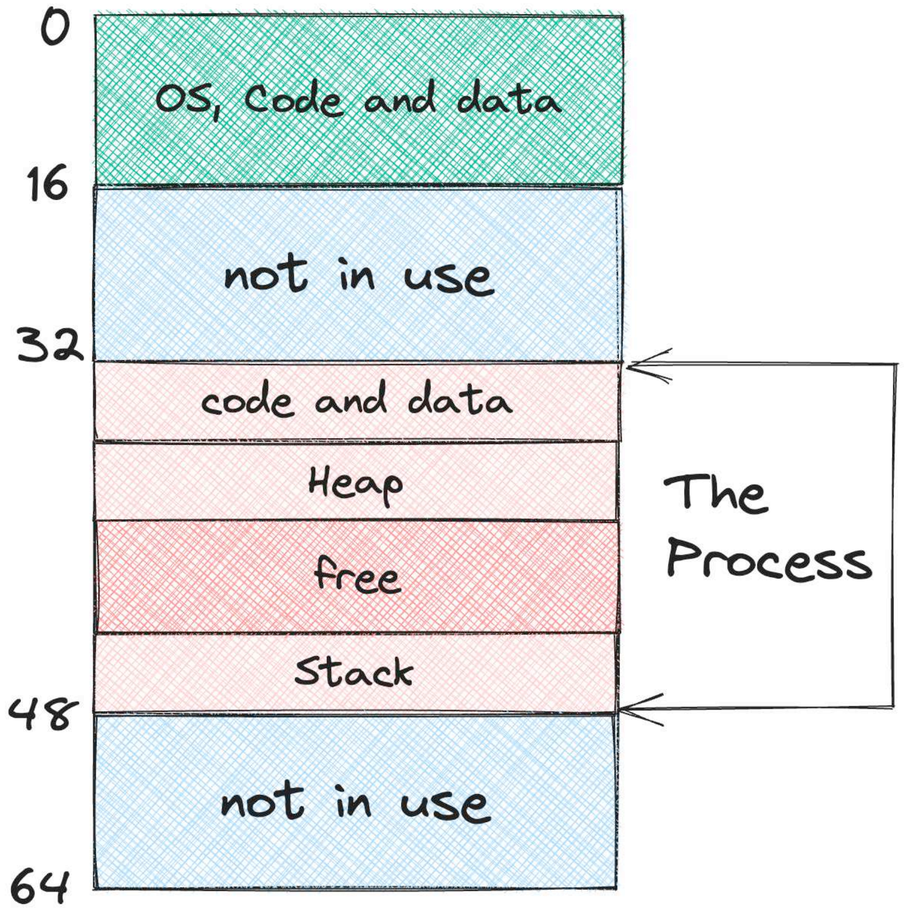
- To virtualize, OS cannot place the process starting from 0.
- The process requires same amount of space as in Virtual address space but somewhere else.
- The reality of physical memory is different from what the process sees!
- The process of translation just needs to map the two
- Can you think of a simple approach?



Dynamic Relocation

The Base and Bounds approach

- Each process allocated **contiguous memory (Segment)**
- Two hardware registers in the CPU (MMU)
  - Base register
  - Bounds register (limits register)
- Each program is written and compiled as if it is loaded at 0
  - However, when the program needs to be run, OS decides the location in physical memory
  - Sets base register to that value
  - Here 32 KB becomes the value in base register



Dynamic Relocation

The Base and Bounds Approach

Physical address = Virtual address + base

- Every memory reference generate by process is virtual address
- Hardware just adds the base value to generate the actual physical address
- This process of transforming VA to PA => (hardware-based) **Address translation**
- Since this happens at runtime => **Dynamic relocation**
- There is only one pair of base and bounds register in the MMU
- OS can make use of simple data structure to keep track of available memory (free list)

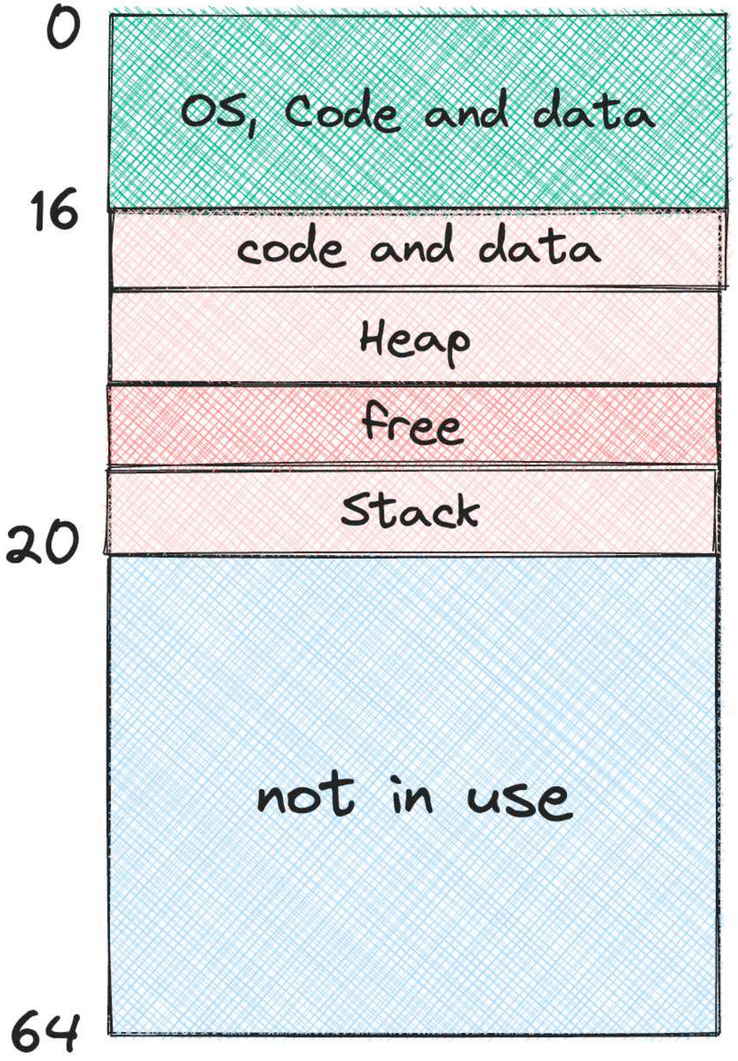
Dynamic Relocation

The Base and Bounds Approach

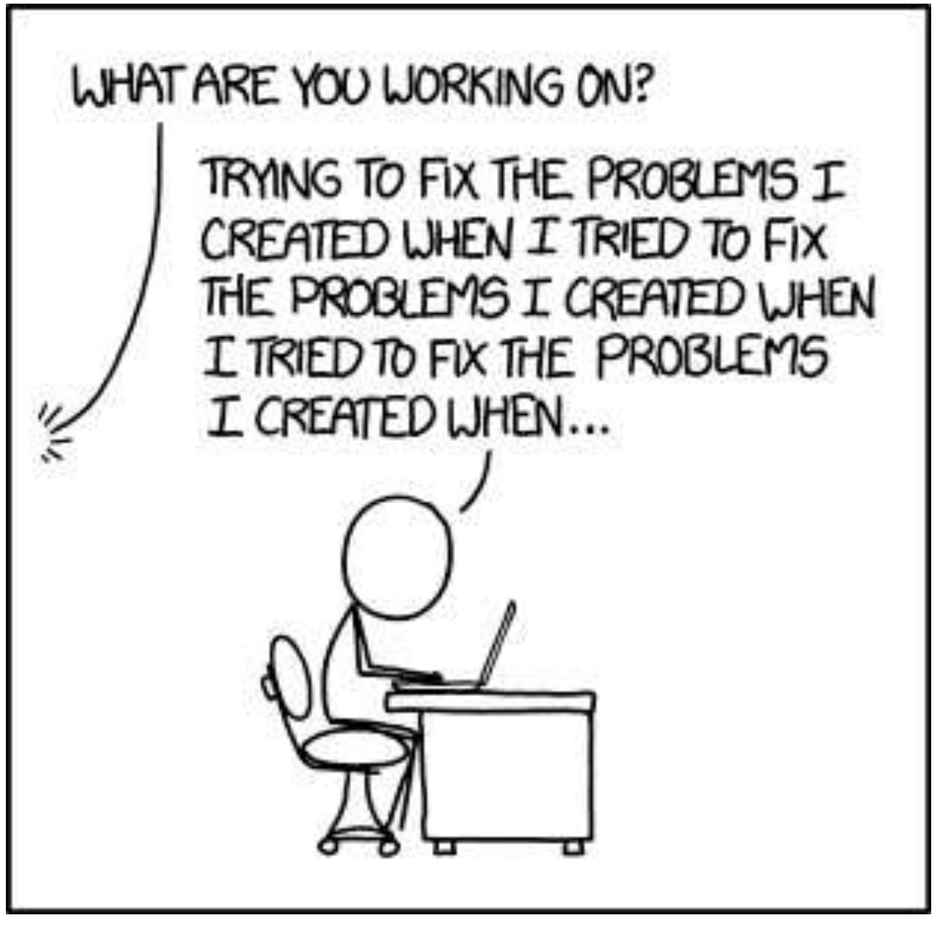
- **Bounds** register ensures that any memory reference is within bounds
  - Everything has to be a legal access
  - If process generates address > bounds (Either relative to VA or PA)
    - CPU raises an exception (Interrupt raised)
    - Process is terminated
- The base and bounds are registers part of hardware (Kept on chip)
- These registers will be inside **Memory Management Unit (MMU)**

Illustration of Base and Bounds Approach

- Process A has an address space of 4 KB, assume that the base is 16 KB
  - Lets say there is an access to VA 0 - PA?
    - PA: 16KB
  - Access to VA 3000 - PA?
    - PA: 16384 + 3000 = 19384
  - Access to VA 4400 - PA?
    - PA: 16384 + 4400 = 20784! **Fault! Why?**



There are some issues!



Source: xkcd

Some Possible Issues

- Simple base and bounds approach is very limiting
  - Memory is contiguous
  - One base and bounds pair per process in the MMU
  - How to support large address space?
- Lot of free space between stack and heap may go unused
  - A typical program would use only certain amount of memory
  - But may demand more! - How to address this?

Segmentation

Generalized Base and Bounds approach

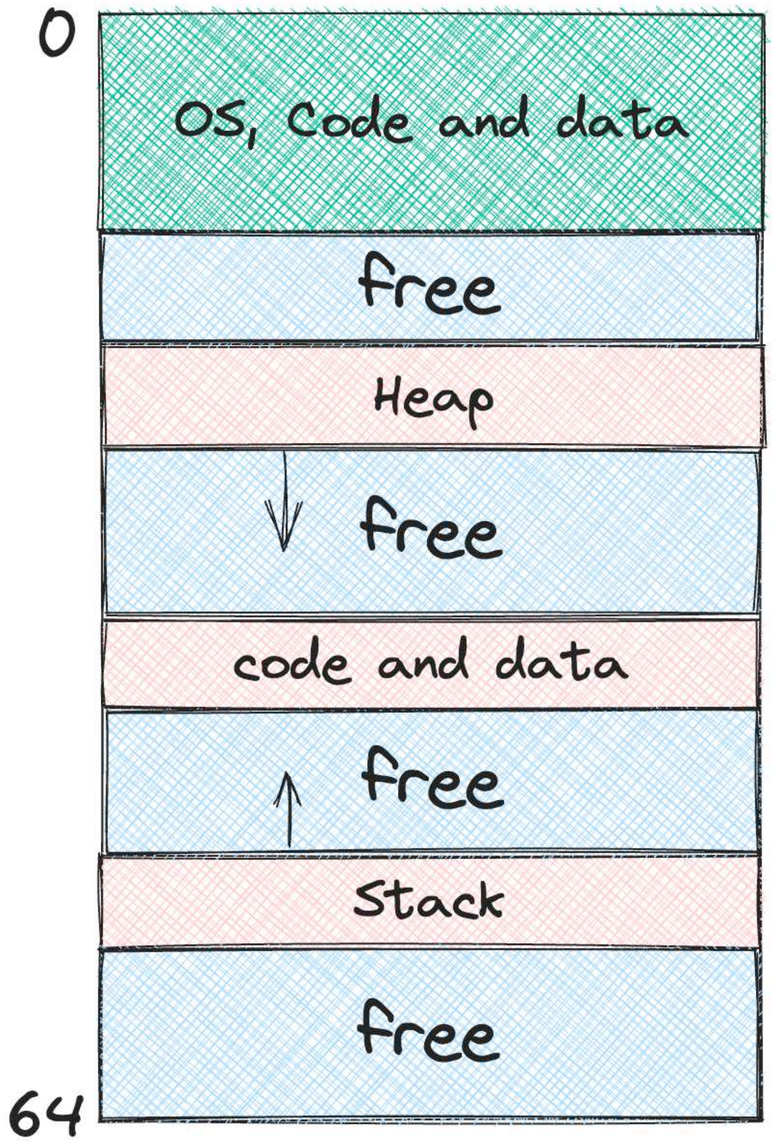
- Instead of having one base and bounds per process
  - Why not have it per logical segment of the address space?
- Segment: Contiguous portion of the address space of a particular length
  - In canonical address space - Three segments
    - Code, Stack and Heap
  - Segmentation basically allows each segment to placed in different parts of memory

Segmentation

Generalized Base and Bounds

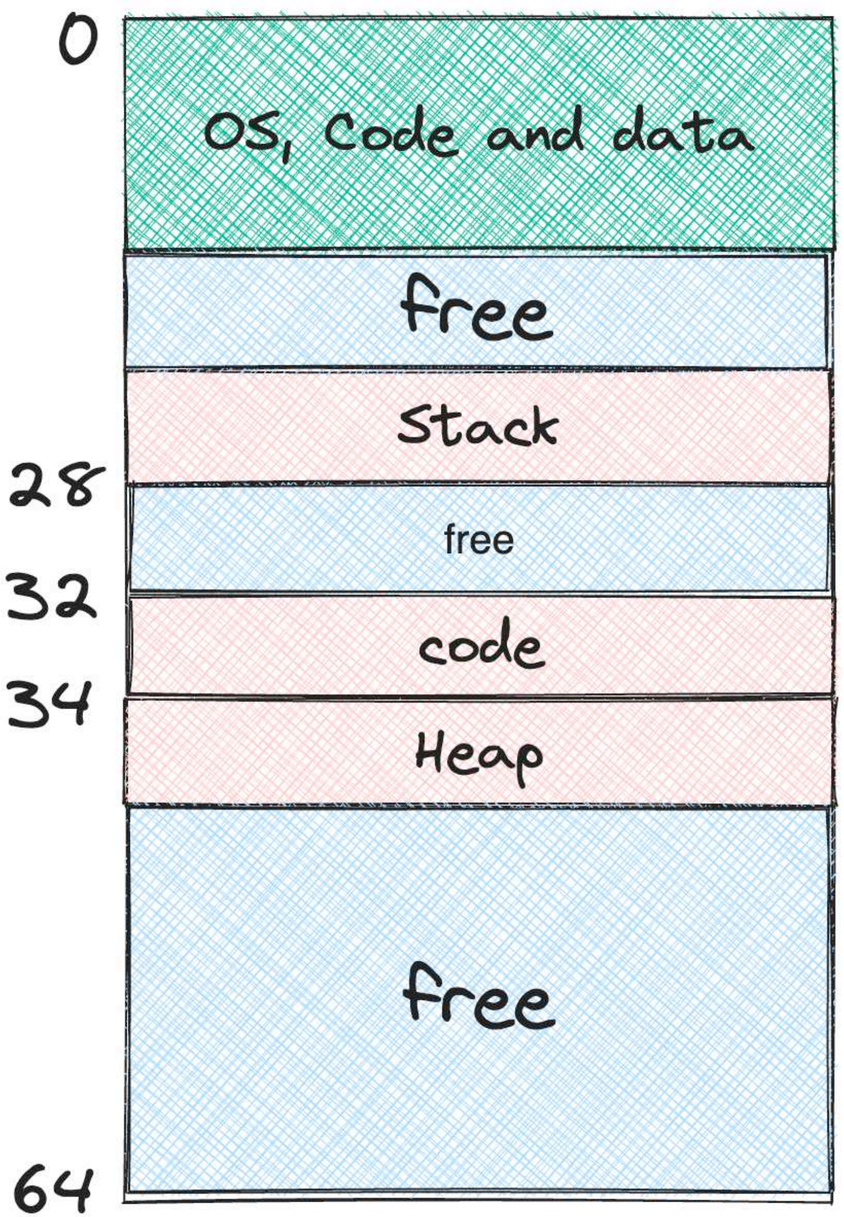
- Only used memory is allocated in physical memory
  - Allows allocating large address space
  - Sparse address space

- Note: Different segments can be placed in different parts of the memory - **How does mapping work?**



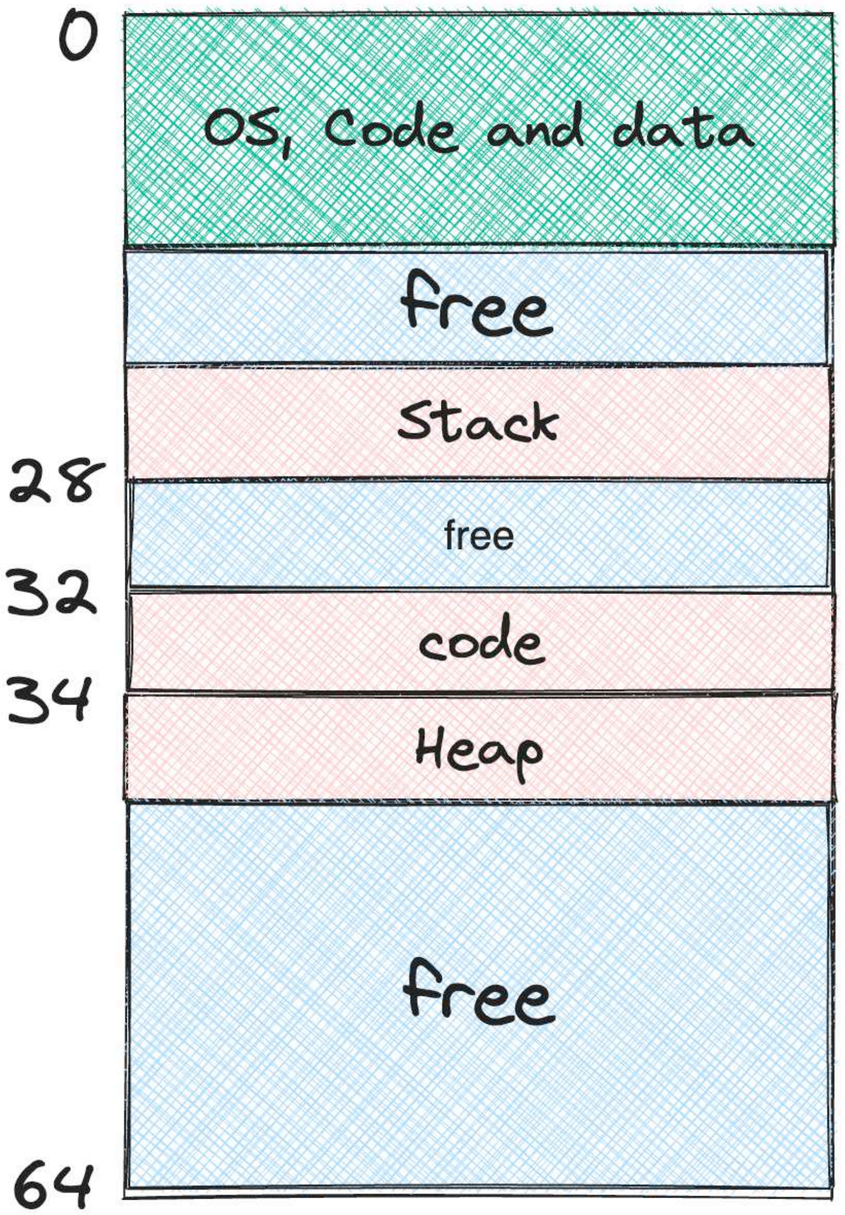
Hardware support (Registers)

Segment	Base	Size
Code (00)	32K	2K
Heap (01)	34K	2K
Stack (11)	28K	2K



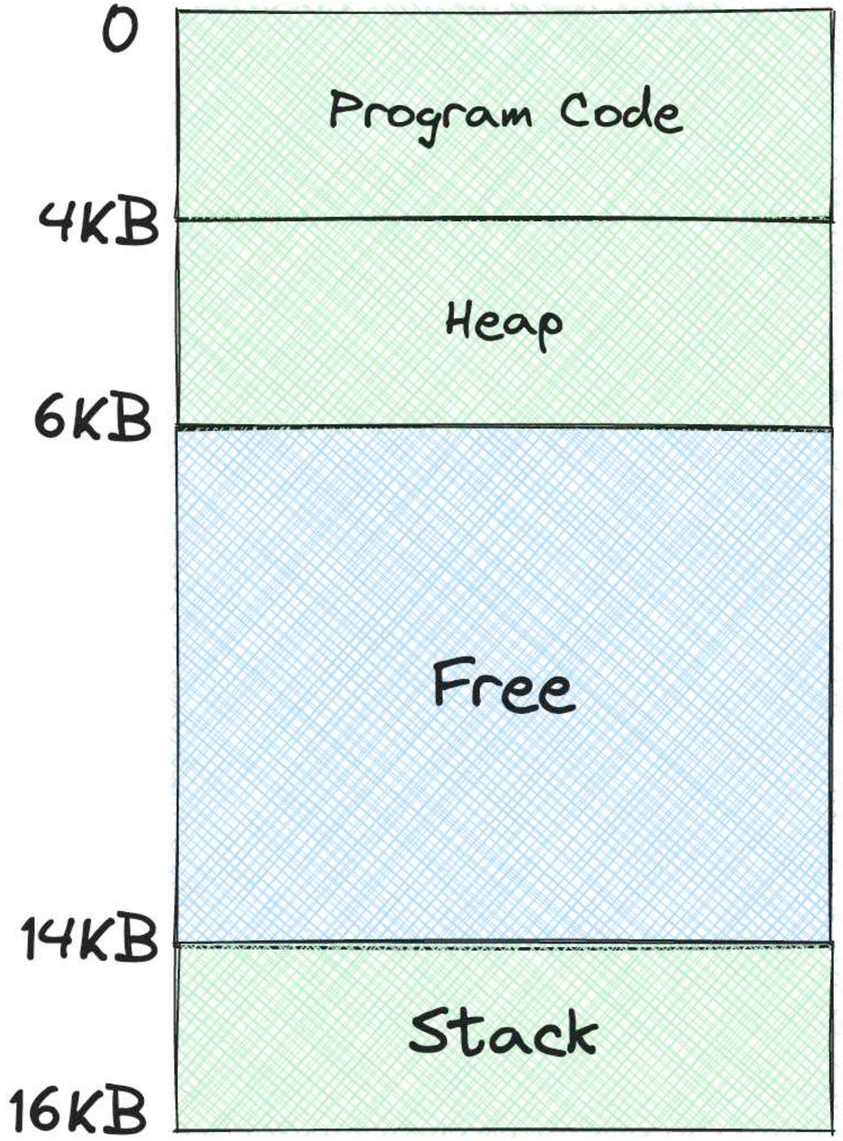
Simple Address Translation

- Reference is made to VA: 100 and code segment
  - MMU: Code starts at 32K
  - PA: 100 + 32868 (32 KB) (100 < 2K)
- Reference made to 4200 to heap segment
  - Can we just add 4200 to base of heap - 34816?
  - Code starts at 0 in virtual address space
  - Heap starts at different location - Get offset?



Simple Address Translation

- Heap starts at 4KB in VA:
  - Offset is 4200
  - Actual base value: 4200 - 4096 = 104: PA?
    - PA: 104 + 34816 (34 KB) = 34920
- How about VA of 7KB (beyond heap address)?
  - Address out of bounds - process termination
  - **Segmentation fault or violation!**



Wait! How to Identify the segments?

- Different segments per process - Code, stack and heap
- Two different approaches - Explicit and implicit
- Explicit approach
  - VA: 14 bit address
  - Use first two bits to identify segment and rest offset

Bits	Segment
00	Code
01	Heap
11	Stack
10	-

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	1	0	0	0

Segment (12-13)

Offset (0-11)

Wait! How to Identify the segments?

- With two bits - Code, heap and stack can be referred
  - Still pair of bits go unused
  - Some systems puts code and heap in segment and uses only one bit
- Implicit Approach
  - Based on how address was formed
  - If it was generated by programming counter during fetch => **Code**
  - Based on stack pointer => **Stack**; else -> **Heap!**

What about Stack?

- Stack grows backwards!
- Some support from hardware to understand which direction to go
  - It is not just about addition to base
  - One bit can be used to indicate direction
  - Each bit implies extra bit to represent the address

Segment	Base	Size	Grows Positive?
Code	32K	2K	1
Heap	34K	2K (max 4K)	1
Stack	28K	2K (max 4K)	0

Example of translation involving stack

- Reference VA: 15 KB - Physical?
  - Try to put 15 KB in binary

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Segment: Stack (13 -12)

Offset: 3 KB (0 - 11)

  - Grows positive 0 (Going negative)
  - Maximum segment size in address space: 4 KB
  - Absolute value = 3 - 4 = -1 KB
  - PA: -1 + 28 (base) = **27 KB**

Bounds Check and Beyond

- For bounds check, ensure that absolute negative value of offset is less than segment size
- The different registers for storing these values are called **segment registers**
- **Can we make this more memory efficient?**
  - **Can we share some segments of the memory?**
  - Code sharing is still in use in many systems
  - Hardware introduce support in the form of protection bits
  - Code segment can be set to read only (Hardware can check if address is within bounds and permissible)

Coarse-grained vs Fine-grained

- **Coarse-grained**: Memory management which takes only few segments into consideration
  - Chops memory into large sized segments
- **Fine-grained:** Address space consisted of large number of smaller sized segments
  - This requires further hardware support
  - **Segment table** stored in-memory

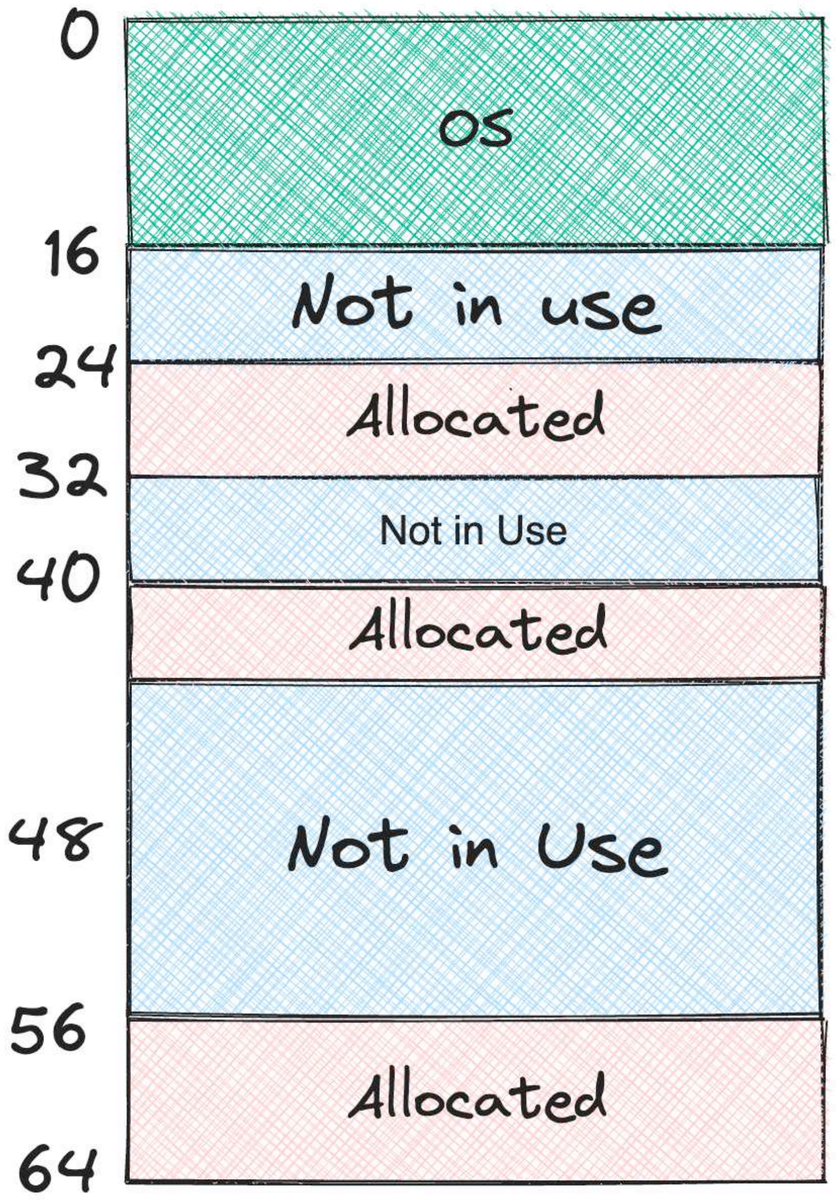
Some Challenges/Issues

- Context-switch:
  - OS must save segment registers and restore them - Each process has own VA
- Free space management:
  - OS should be able find physical memory for its segments
  - Each process has number of segments and each segment could be different size
  - Results in **External Fragmentation!**

Source: imageflip.com

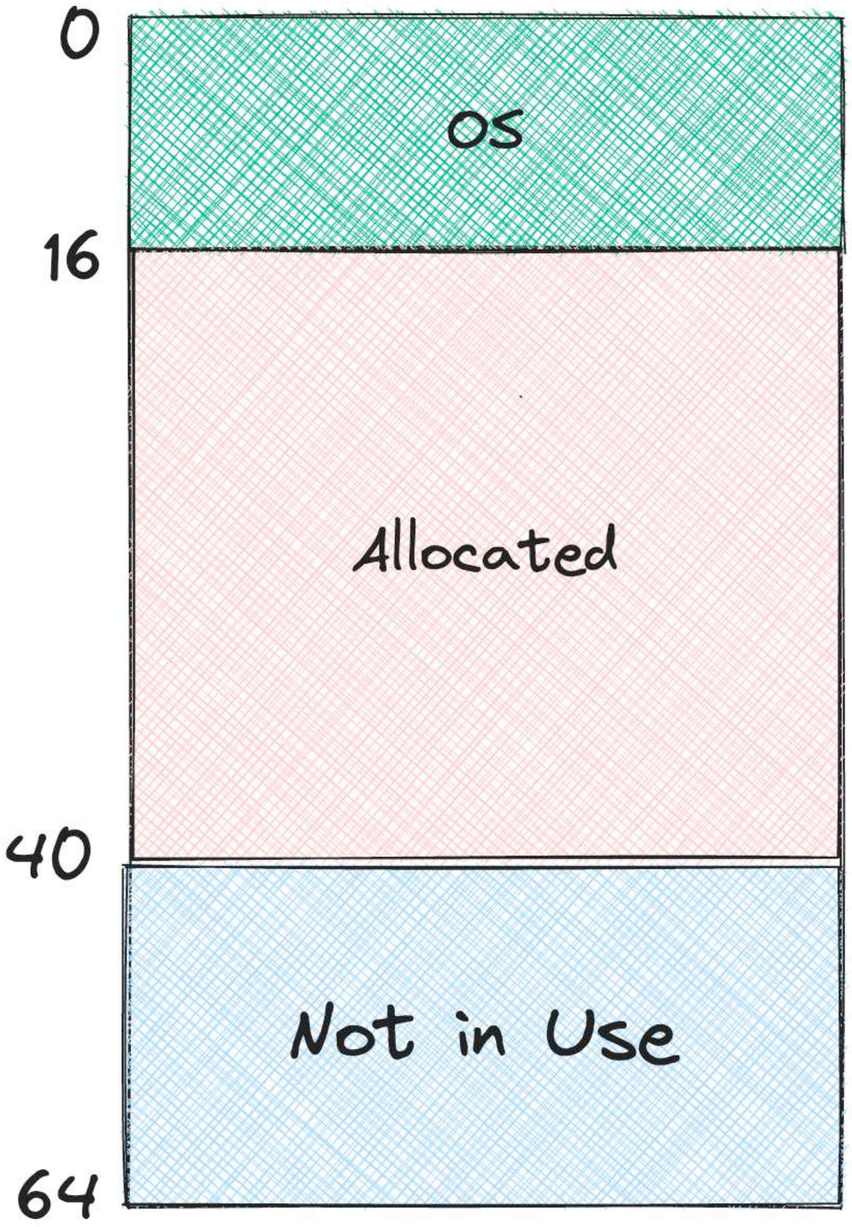
External Fragmentation

- Physical memory quickly becomes full of little holes
- Hard to allocate new segments
- Consider process wishes to allocate a 20 KB segment - 24 KB is free but not in a contiguous space!!
  - Can we come up with a compact version of this?



Compacted Version

- Seems like a more easy solution - OS could stop the running process
  - Copy data into a contiguous region
  - Change segment values to point to new region
  - Now there is larger memory
- Process is very **memory intensive!**



SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Memory Virtualization - Introduction to Paging

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:

- Operating Systems: In three easy pieces, by Remzi et al.
- Lectures on Operating Systems by Youjip Won, Hanyang University

Example of translation involving stack

- Reference VA: 15 KB - Physical?
  - Try to put 15 KB in binary

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Segment: Stack (13 -12)

Offset: 3 KB (0 - 11)

  - Grows positive 0 (Going negative)
  - Maximum segment size in address space: 4 KB
  - Absolute value = 3 - 4 = -1 KB
  - PA: -1 + 28 (base) = **27 KB**

Bounds Check and Beyond

- For bounds check, ensure that absolute negative value of offset is less than segment size
- The different registers for storing these values are called **segment registers**
- **Can we make this more memory efficient?**
  - **Can we share some segments of the memory?**
  - Code sharing is still in use in many systems
  - Hardware introduce support in the form of protection bits
  - Code segment can be set to read only (Hardware can check if address is within bounds and permissible)

Coarse-grained vs Fine-grained

- **Coarse-grained**: Memory management which takes only few segments into consideration
  - Chops memory into large sized segments
- **Fine-grained:** Address space consisted of large number of smaller sized segments
  - This requires further hardware support
  - **Segment table** stored in-memory

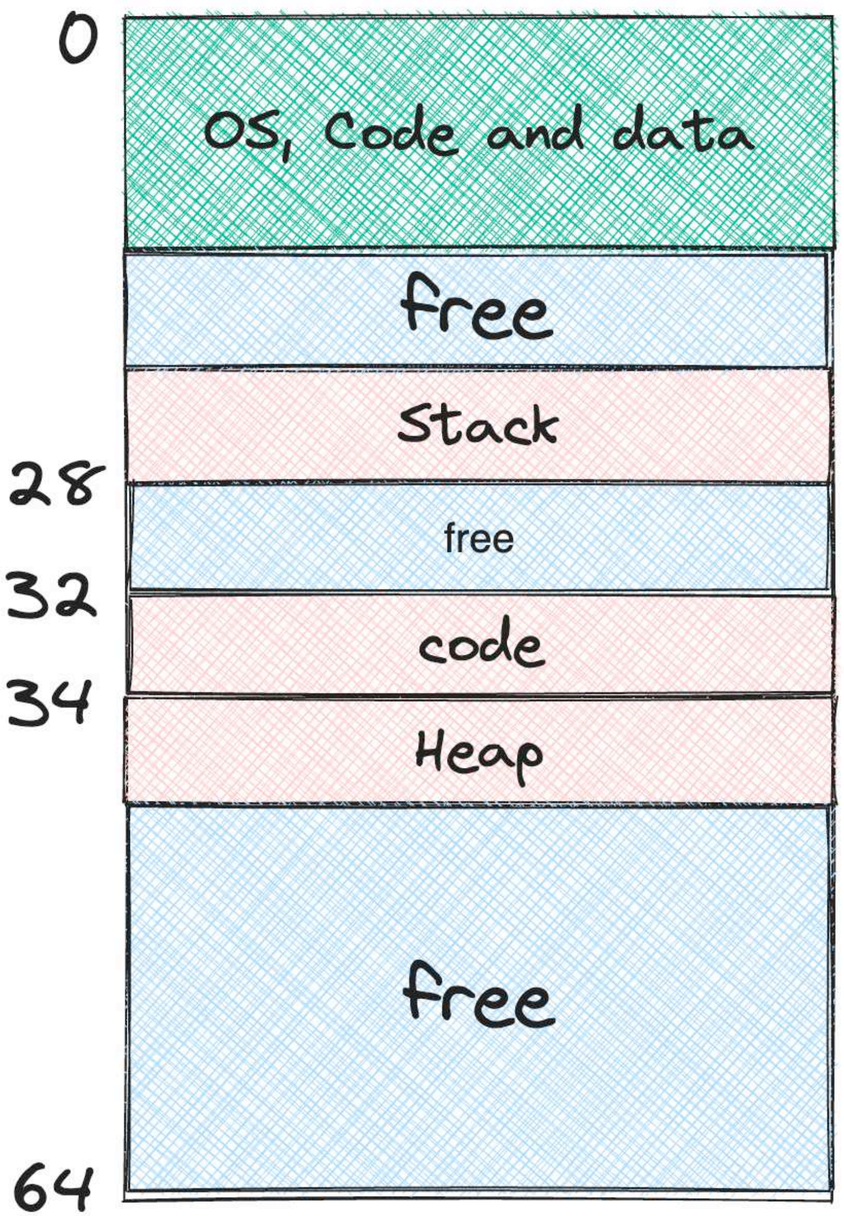
Some Challenges/Issues

- Context-switch:
  - OS must save segment registers and restore them - Each process has own VA
- Free space management:
  - OS should be able find physical memory for its segments
  - Each process has number of segments and each segment could be different size
  - Results in **External Fragmentation!**

Source: imageflip.com

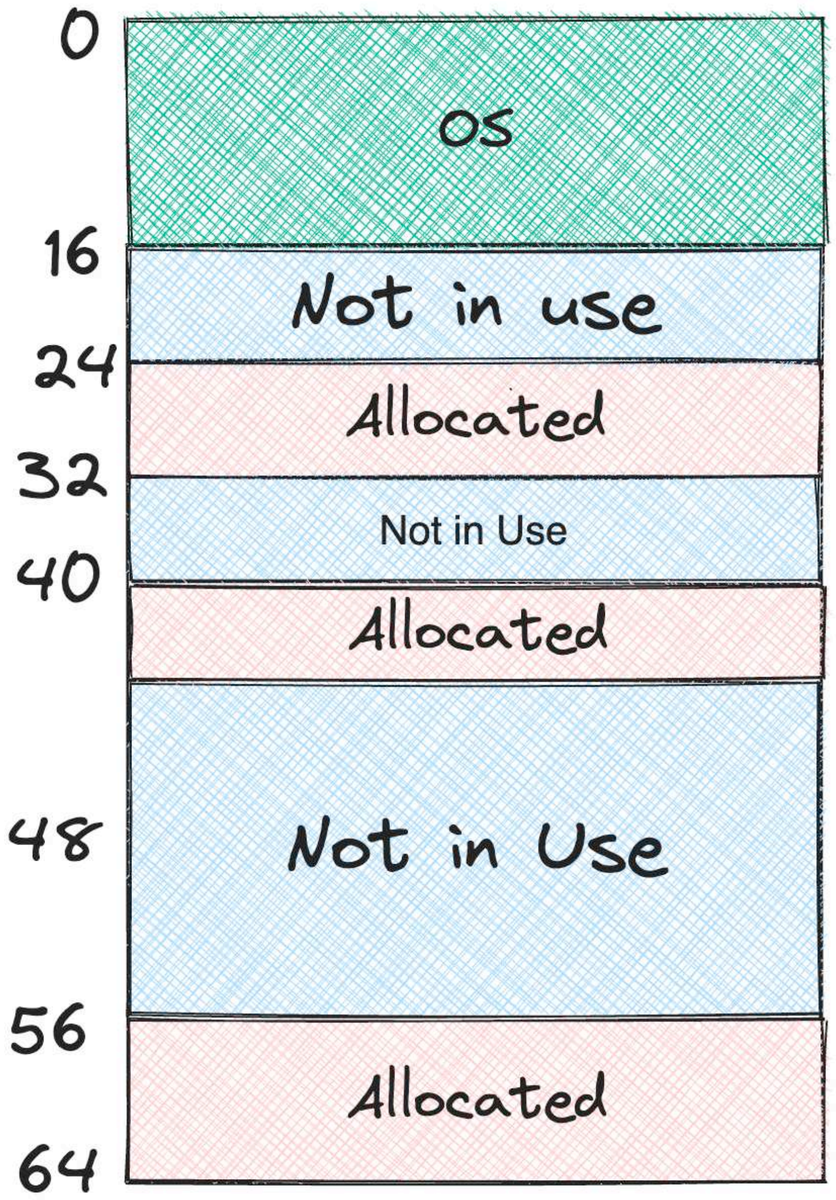
Hardware support (Registers)

Segment	Base	Size (Max 4K)
Code (00)	32K	2K
Heap (01)	34K	2K
Stack (11)	28K	2K



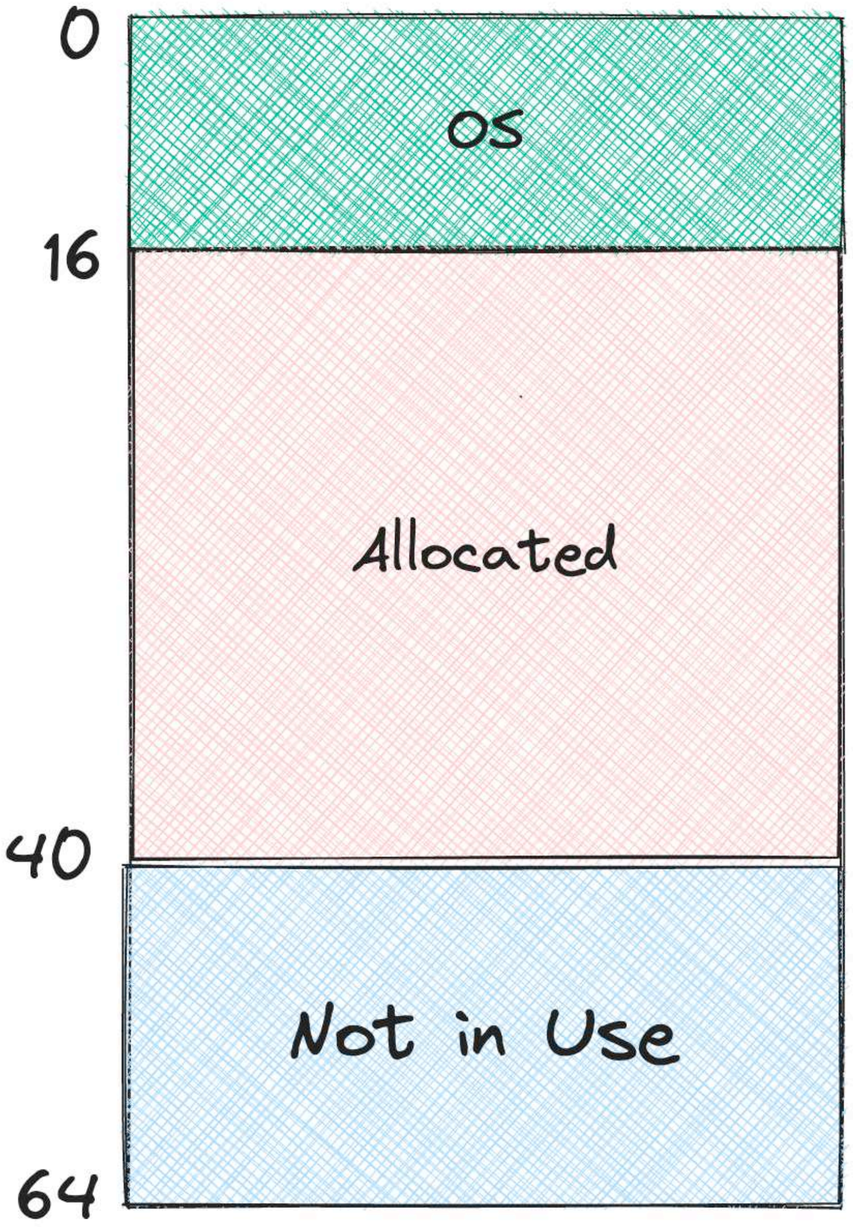
External Fragmentation

- Physical memory quickly becomes full of little holes
- Hard to allocate new segments
- Consider process wishes to allocate a 20 KB segment - 24 KB is free but not in a contiguous space!!
  - Can we come up with a compact version of this?



Compacted Version

- Seems like a more easy solution - OS could stop the running process
  - Copy data into a contiguous region
  - Change segment values to point to new region
  - Now there is larger memory
- Process is very **memory intensive!**



Algorithmic Approaches can be used

- Free-list management algorithm
  - OS can keep track of available memory
  - Different algorithms can be used to intelligently allocate
    - **Best-fit:** returns one closest in size that satisfies the request
    - Then other algorithms like first-fit, worst-fit, buddy algorithm, etc.
  - Good algorithm can only minimise external fragmentation but not avoid it
  - The only possible solution - **Why to allocate variable sized segments?**

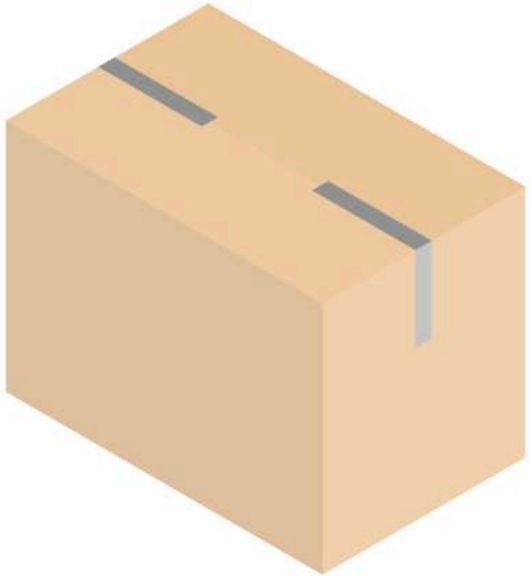
What can be done differently?

- Can we divide the physical memory into fixed sized lengths?
  - External fragmentation can be avoided
  - It will still result in internal fragmentation - This may be better!
  - This is what **Paging** does!

Going back to the Analogy

Over the idea of categories

Use fixed sized shelves

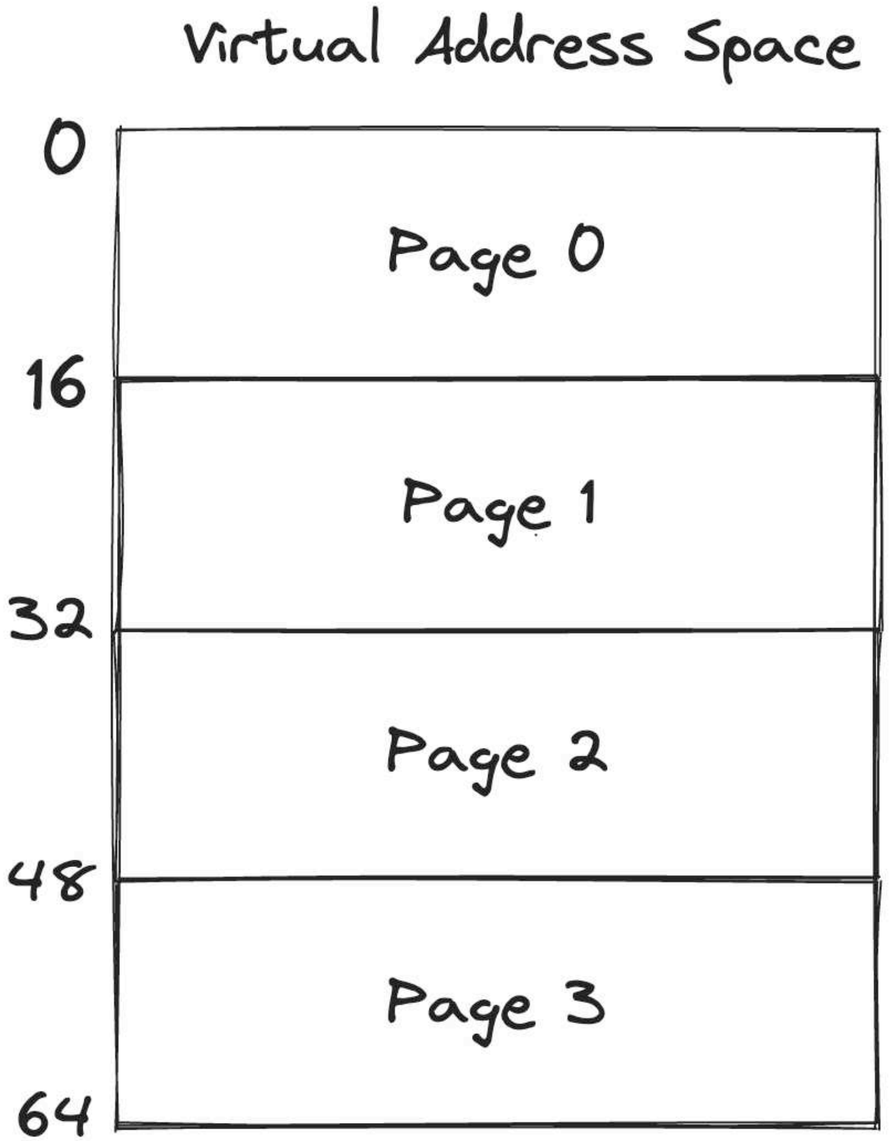


Each box can contain some items
The boxes are placed inside fixed sized shelves

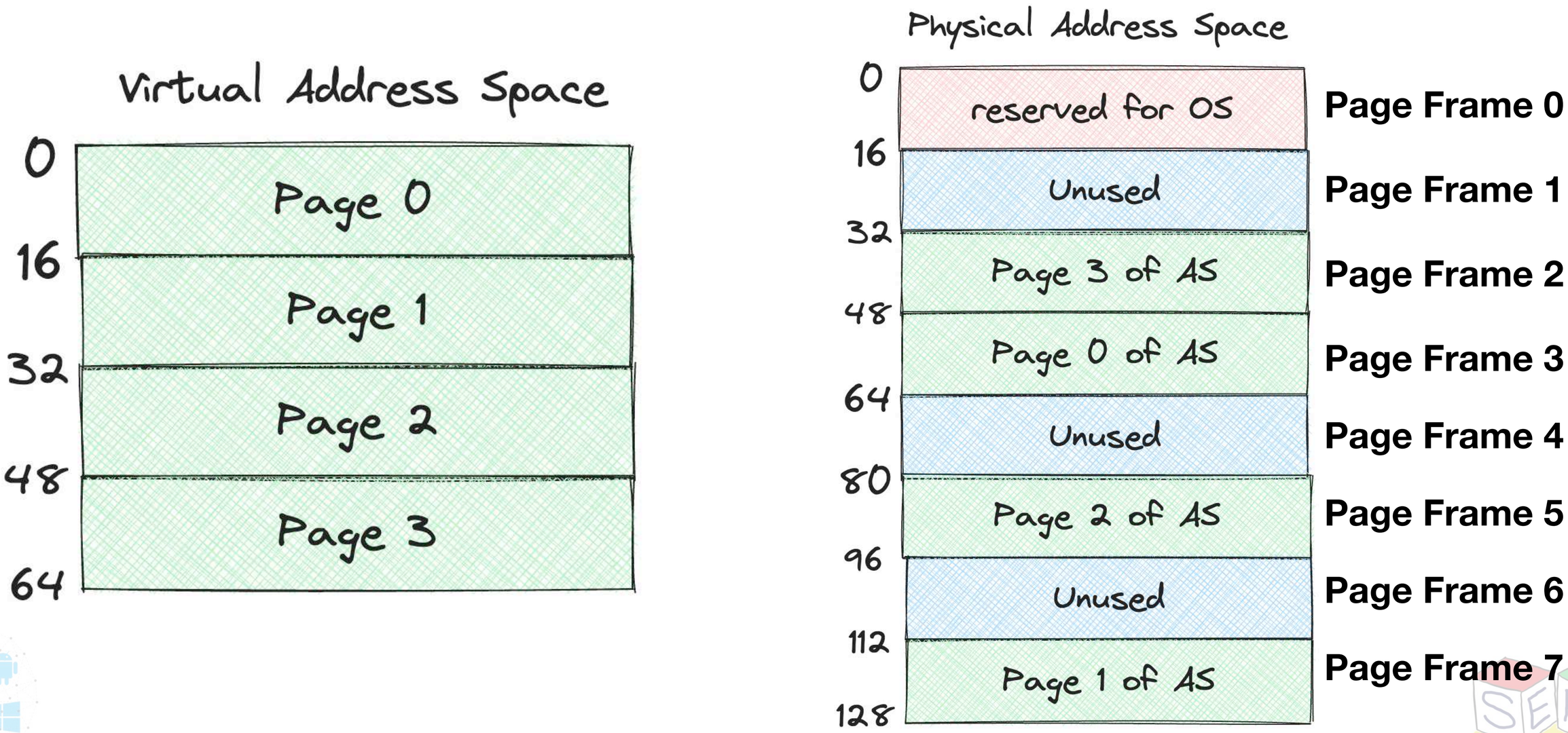
Some mapping between product, box and shelves

Introduction to Paging

- Idea goes back to the earlier systems - Altas
- Instead of splitting address space into three logical segments (variable size)
  - Split the address space into fixed sized units - **Pages**
  - Virtual address space and Physical address space are split into fixed sized pages
  - Its all about mapping **page in VA** to **page frame in PA**
- Eg: Virtual Address -> 64 bytes



Virtual Address and Physical Address



Advantages of Paging

- **Flexibility**
  - Easy support for abstraction of physical space
  - No need to make assumption on the heap and stack space
- **Simplicity**
  - All that the OS needs to do is to find some free pages and load the process
- The OS makes use of per-process data structure - **Page table**
  - To record the mapping between page number and page frame

Page Table

- Role of page table is for address translation
- Page table is per-process data structure
- If another process runs, it will have its own page table
- The VPN maps to different PFNs as long as they don't share!!
- How can this be implemented?

Virtual Address Space	Physical Address Space
VPN 0	PFN 3
VPN 1	PFN 7
VPN 2	PFN 5
VPN 3	PFN 2

Process of Translation

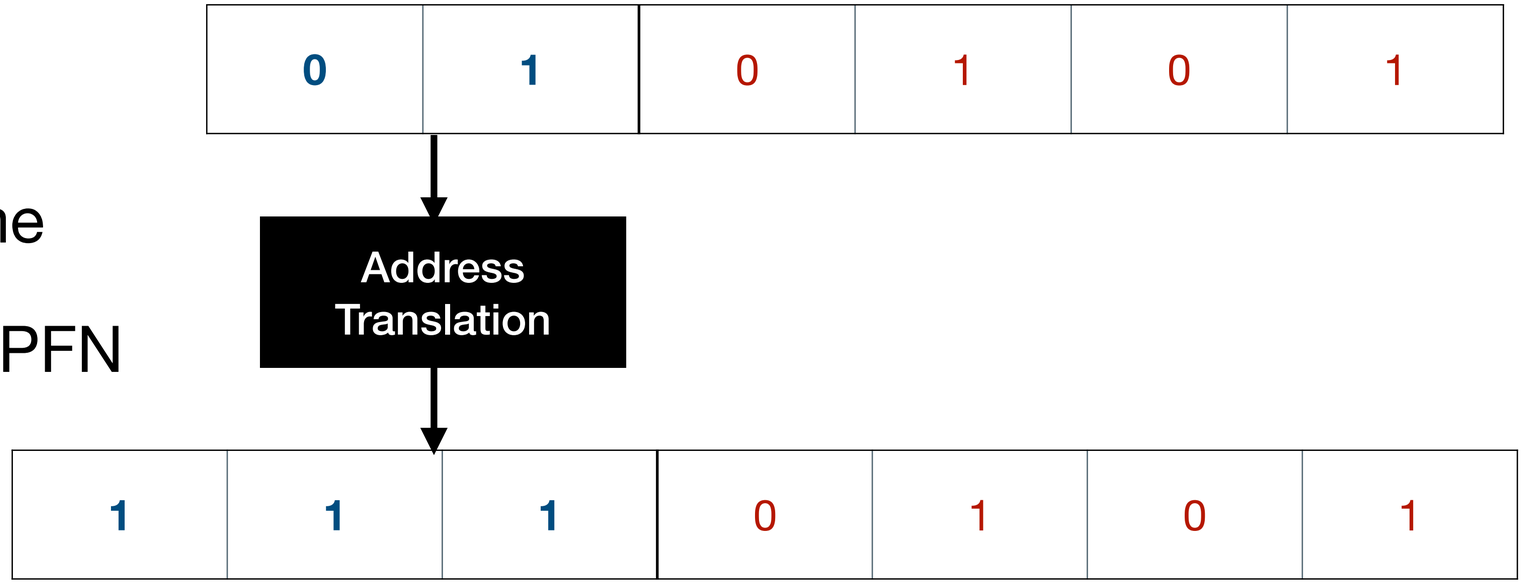
- Eg: Translation **movl <virtual address> %eax**
  - Split them into two components
    - Virtual Page Number (VPN)
    - Offset
    - VPN can be determined by number of pages
    - 64 bytes = 2^6 => 6 bits
    - Each page, 16 bytes = 2^4 => **4 bits (offset)**
    - **Remaining 2 bits for VPN**

VPN (2 bits)		**Offset (2 bits)**			
VA5	**VA4**	VA3	VA2	VA1	VA0

Simple Example

Example instruction: mov 21 %eax

- Convert 21 into binary:



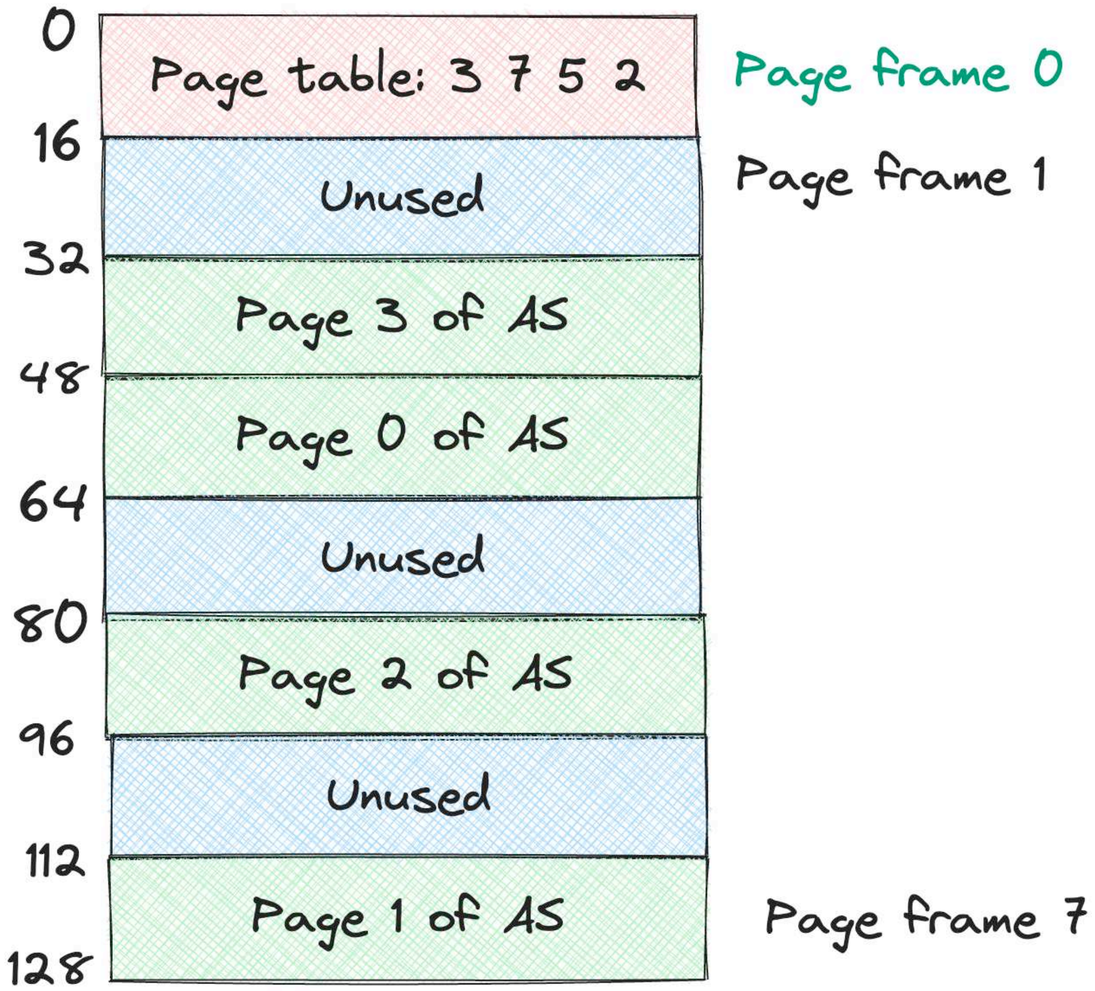
- Offset stays the same
- Its all about VPN -> PFN
- Page table helps!

Location of Page Tables

- Let us take a guess about the size of a page table
  - Consider a 32 bit address space with 4 KB pages. What will be the number of bits for offset?
    - 12 (4 KB = 2^{12})
    - VPN: 20 bits, think about possible translations?
    - 2^{20} translations ~ **a million mappings**
    - If each mapping is 4 bytes => total 2^{20} X 4 = 4 MB per process per page table
    - What if there are 100 processes => **400 MB just for address translations!**

Storing Page Table

- Page tables can be huge
- Storing in MMU or any hardware register may be hard
- Instead, stored in-memory
- Assume that page table lives in Physical memory that OS handles



Contents of Page Table

- Page Table: data structure to map VA to PA
  - Each entry in a page table: **Page Table Entry (PTE)**
  - Simplest form is a linear page table: Just an array
  - Array indexed with VPN
  - Identify PTE at a given index to get to the PFN
  - Each PTE consists of a number of bits

Page Table Entry

- Valid bit: Whether page is valid (unused pages will be marked invalid)
  - Any access to invalid will result in trap -> termination
- Protection bit: Whether page can be read from, written to, executed from
  - Wrong access traps into the OS
- Present bit: Indicates whether page is present in physical memory
- Dirty bit: Whether page has been modified since brought into memory
- Reference bit (accessed bit): Whether page has been accessed (recently used!)

VPN	**PFN**	**V**	P	Present	Dirty	Ref

What about Efficiency of Paging

Page Table Base Register (PTBR) —contains→ Starting location of page table

- Hardware must know where the page table is for a process
- To reach correct location => Identify VPN and offset
  1. Index into the array of PTE as pointed out by PTBR
  2. Get PTE from memory, extract PFN
  3. Add offset to PFN to get to final address
  4. Finally hardware fetches data

What about Efficiency of Paging

- Page tables in memory can be too big!
  - They can also slow down things
  - mov 21 %eax
  - Translate 21 into VA into PA (117 for example)
  - Before loading, system must fetch PTE from page table
  - Perform translation and get the translated address and get desired data
  - Any issues so far?

What’s the problem?

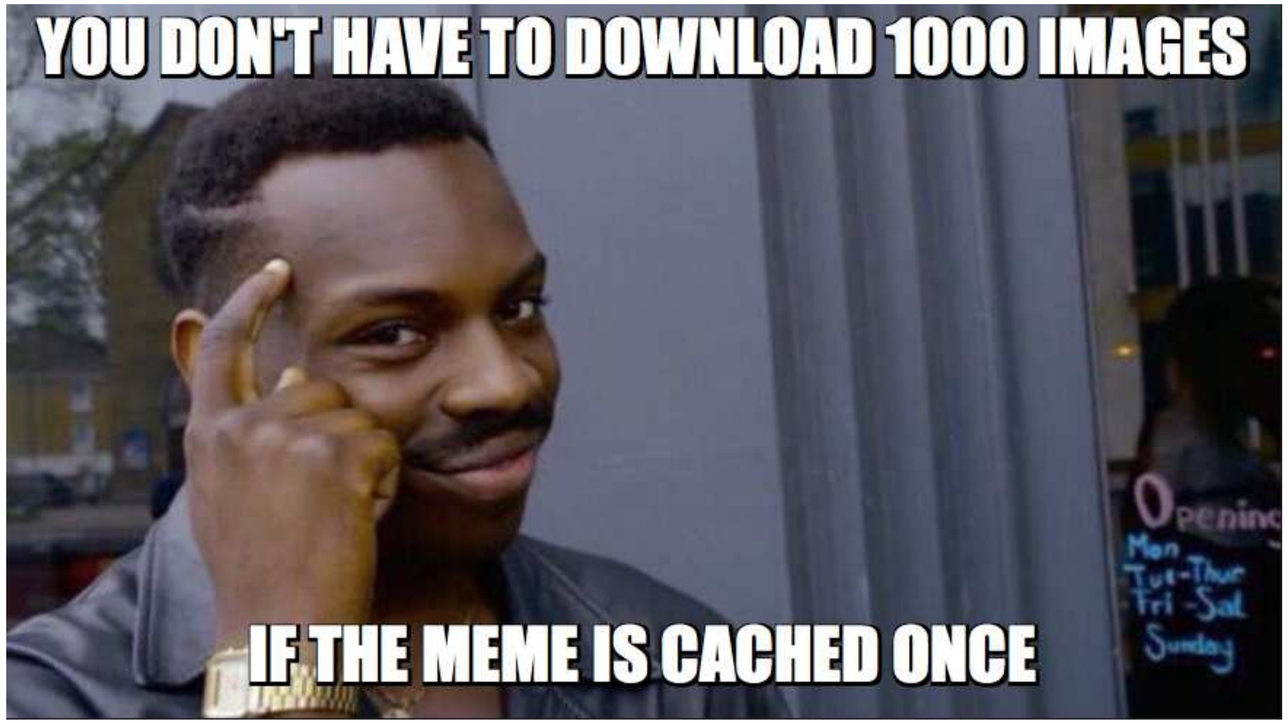
- There is one extra memory reference here - How?
- This can lead to a big slowdown
- Two main issues needs to be addressed:
  - Size of the page table is very big (Memory Overhead)
  - They can also be too slow to access (Time Overhead)
- Paging does help in solving **external fragmentation!**

Improving the speed of translation

- How to speed up translation and avoid the extra reference that paging requires?
  - What hardware support is required?
  - What OS involvement is needed?

- Let us think for a brief moment. Imagine that you are a designer! How to about this?

Why don’t we cache it?



Source: medium.com

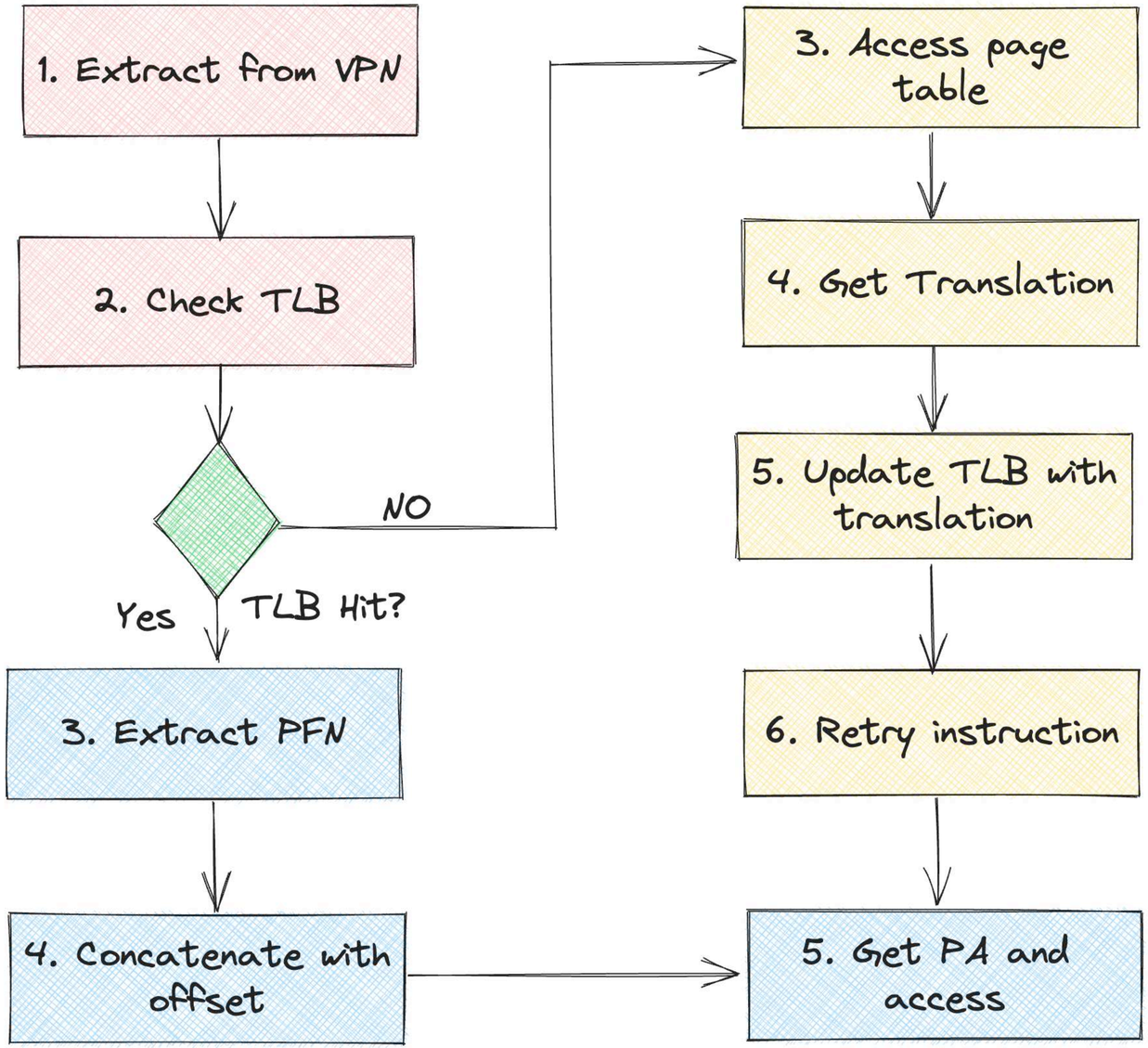
Put Hardware in action

Comes Translation Lookaside Buffer (TLB)

- Introduce translation Lookaside buffer to MMU
- To speed up address translations
  - Add TLB
  - This is similar to a hardware cache of popular VA to PA transitions
    - Address Translation Cache
  - For each translation
    - Check TLB if the page has already been accessed
    - If no, get it. If yes, use the address

TLB Control Flow Algorithm

- TLB is like all caches
- If translation found
  - Little overhead
- When there is miss
  - Cost of memory access!
- **Goal:** reduce TLB miss!



Illustrative Example: Array Access

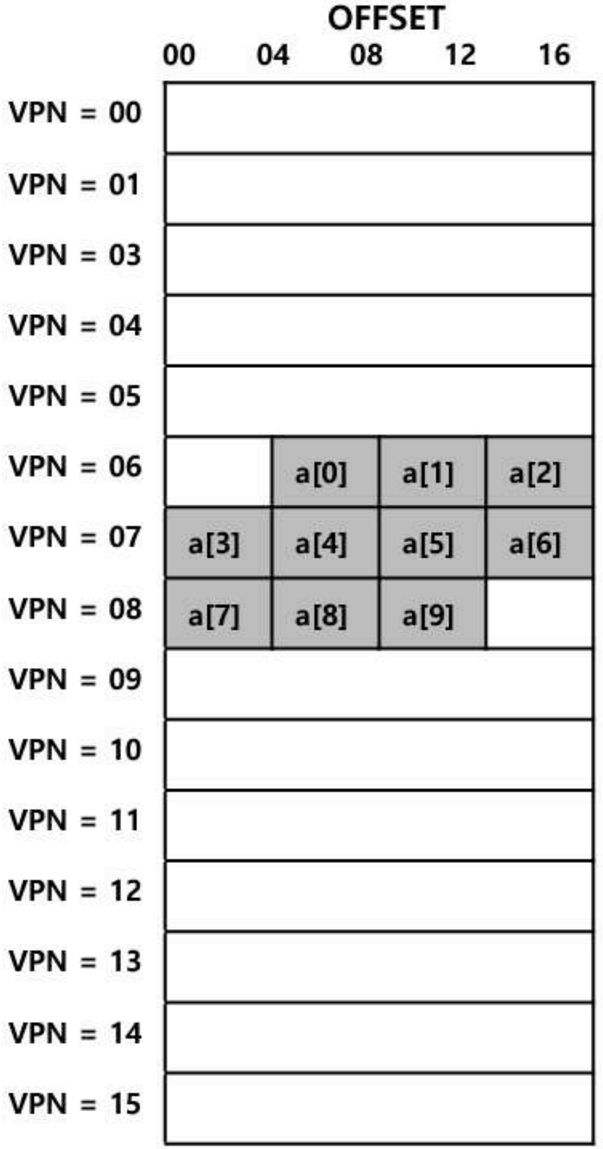
- Assume an array of 10 4-byte integers starting at VA 100. Assume an 8-bit VA space with 16 byte pages
  - This implies 4 bits for offset and 4 bits for VPN
  - Each page is 2^4 = 16 byte size
- How shall the virtual address space be organized?

Array access program in C

```c
int main(int argc, char* argv[])
{
  int sum = 0;
  for (int i = 0; i < 10; i++)
  {
    sum = sum + a[i];
  }
  return 0;
}
```

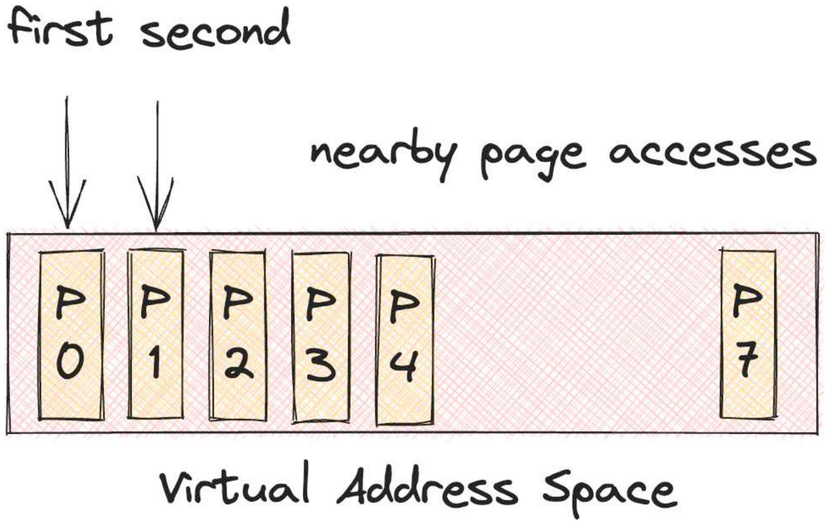
Illustrative Example: Array Access

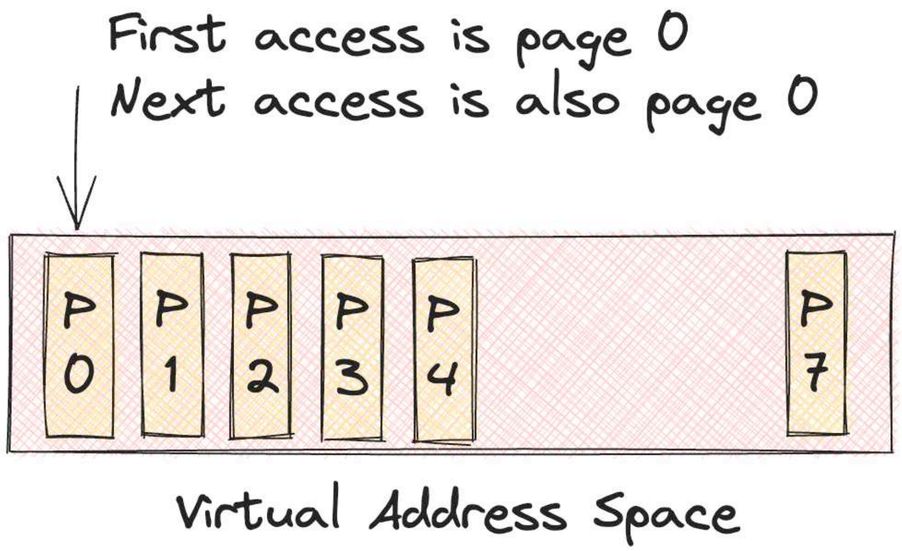
- Access to a[0] with VA 100 - How?
  - It will be a TLB miss!
- Access to a[1] will be a TLB hit!
- Access to a[3] will be a TLB miss!
- Access to a[4] to a[6] will be a hit!
- Access to a[7] will be a miss then a[8] and a[9] will be hit
- miss, hit, hit, miss, hit, hit, hit, miss, hit, hit
- Hit rate: 70% - Not bad!



Locality as a factor

- **Spatial Locality**
  - If a program accesses memory at x, it will likely soon access memory near x.
  - What if page size was 32 bytes in previous example?
- **Temporal Locality**
  - An instruction or data that has been recently accessed will likely be re-accessed soon in the future





Handling TLB Miss?

Hardware Handler

- Hardware handles the TLB miss entirely on CISC
  - Older systems based on Intel x86
  - Hardware has to know where the page tables are stored
  - The base address is stored in Page Table Base Register
  - Hardware would walk the entire page table in parallel to find the correct PTE and **extract** the translation, **update** and **retry** instruction

Handling TLB Miss?

Software/OS Handler

- Software handles the TLB miss entirely on RISC
  - Hardware simply raises an exception on a miss (trap handler)
  - Trap handler for handling TLB miss is invoked
  - The code will look up the page table for translation, updates TLB with privileged instruction and return from trap
  - At this point hardware retries the instruction
    - There is a subtlety to context switch here - Why?

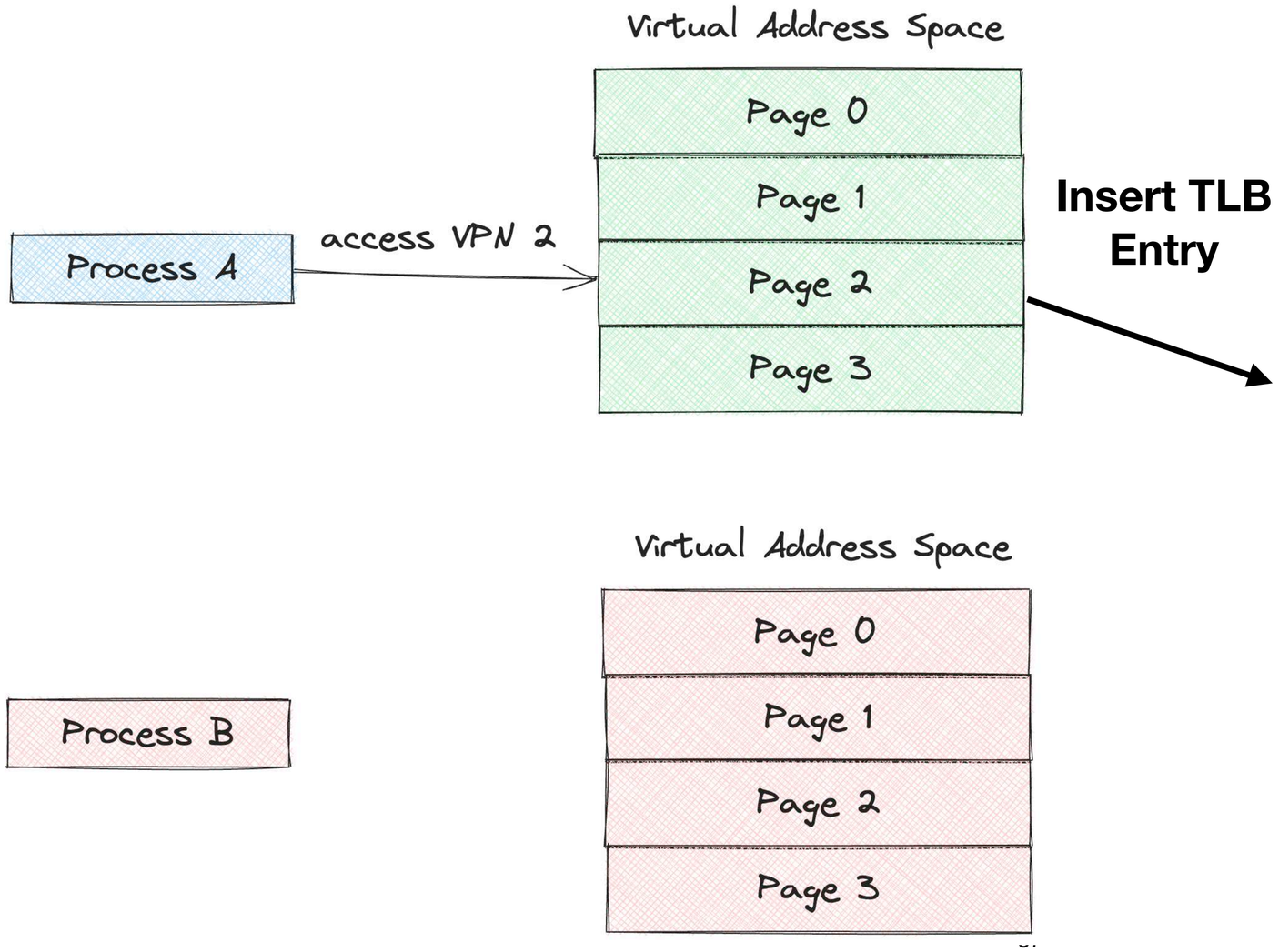
Some Caution!!

- When returning from TLB miss
  - Hardware must save the PC of current instruction and again execute the same instruction for **retry** resulting in hit
  - In usual scenario the return-from-trap goes to next instruction
- OS should not go into infinite chain of TLB misses
  - TLB miss handlers are also code that requires address translation
  - Keep TLB miss handlers in physical memory
  - Use some permanent translation slots for handler code

Inside TLB

- TLB is managed by fully associative method
  - A typical TLB have 32, 64 or 128 entries
  - Hardware searches the entire TLB in parallel to find the translation
  - Other bits
    - **Valid:** Enty has a valid translation
    - **Protection**: how a page can be accessed (read only, read, etc)
    - Other bits: **address space identifier**, **dirty bit**, etc.

What about Context Switch?

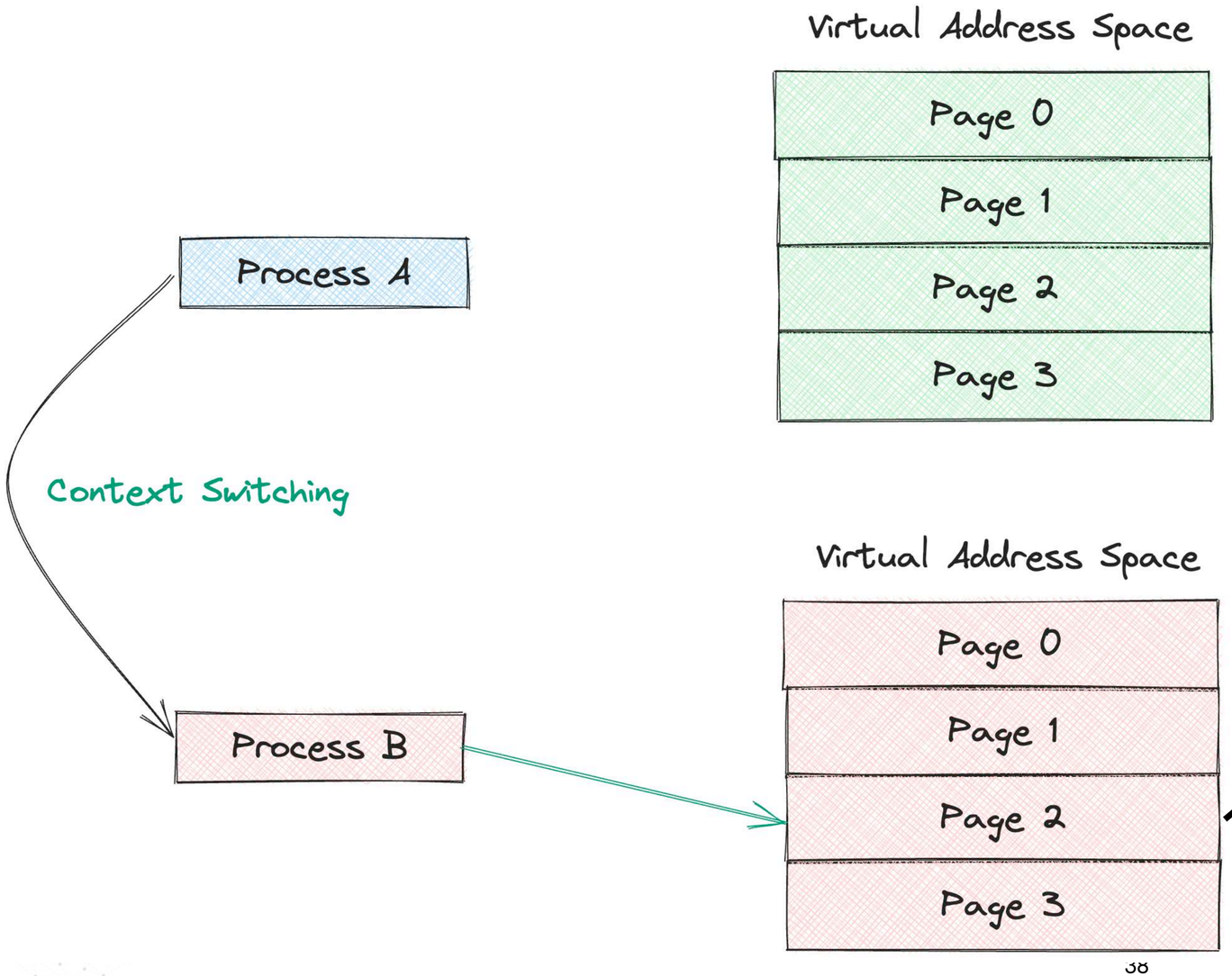


Insert TLB Entry

TLB Table

VPN	PFN	Valid	Prot
2	100	1	rwx
...	...	...	...

Context Switching and TLB



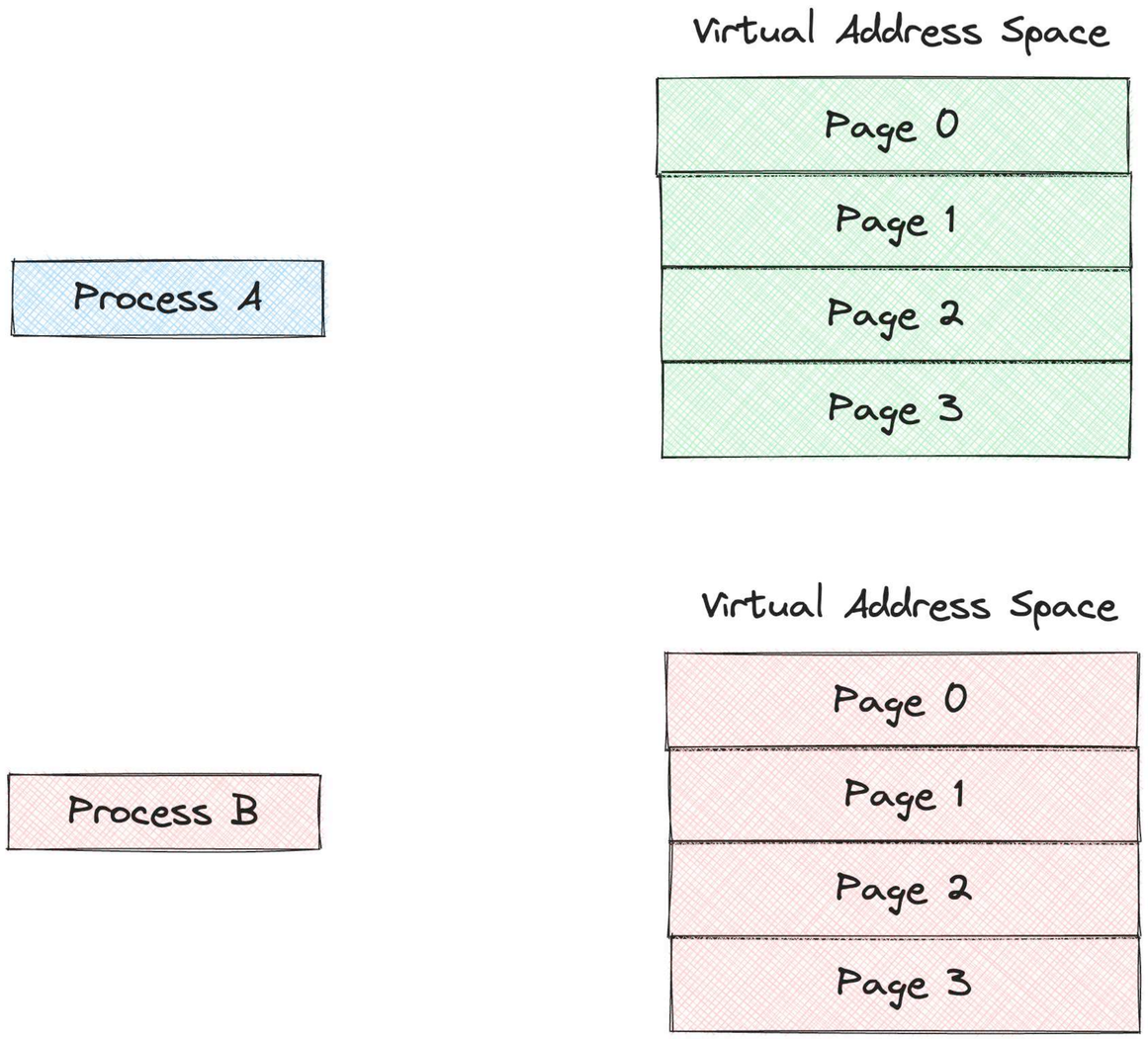
Flushing can be an option but… TLB miss?

TLB Table

VPN	PFN	Valid	Prot
2	100	1	rwx
…	…	…	…
2	180	1	rwx

Insert TLB Entry

Address Space Identifier



VPN	PFN	Valid	Prot	ASID
2	100	1	rwx	1
...	...	...	...	
2	180	1	rwx	2

Address Space Identifier (ASID) field to distinguish (8 bits)

Processes can Share a Page

VPN	PFN	Valid	Prot	ASID
2	100	1	r-x	1
...	...	...	...	
10	100	1	r-x	2

- Sharing can be useful, it reduces the number of physical frames

Still some issues!

- TLB has limited size
  - Which entry to replace when adding new entries?
- What about the size of the page table?
  - Are there ways to minimise them?
  - Where will they be stored?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Memory Virtualization - Paging: Smaller Page Tables

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:

- Operating Systems: In three easy pieces, by Remzi et al.
- Lectures on Operating Systems by Youjip Won, Hanyang University

Illustrative Example: Array Access

- Assume an array of 10 4-byte integers starting at VA 100. Assume an 8-bit VA space with 16 byte pages
  - This implies 4 bits for offset and 4 bits for VPN
  - Each page is 2^4 = 16 byte size
- How shall the virtual address space be organized?

Array access program in C

```c
int main(int argc, char* argv[])
{
  int sum = 0;
  for (int i = 0; i < 10; i++)
  {
    sum = sum + a[i];
  }
  return 0;
}
```

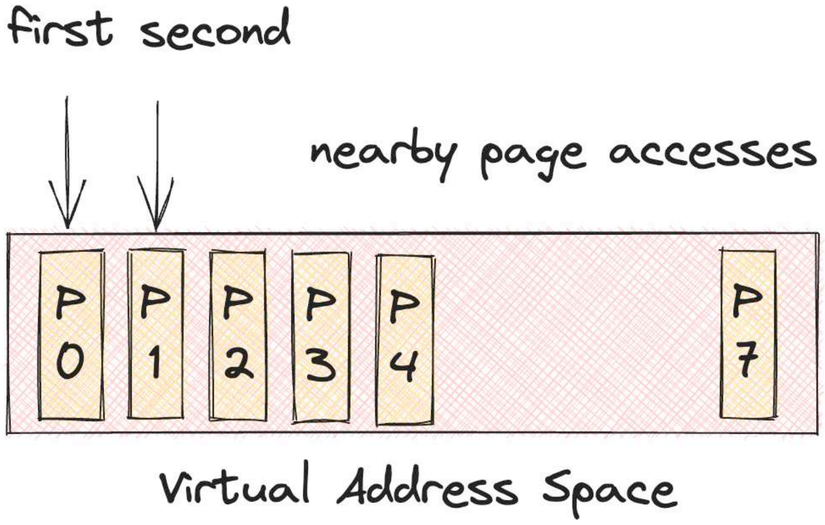
Illustrative Example: Array Access

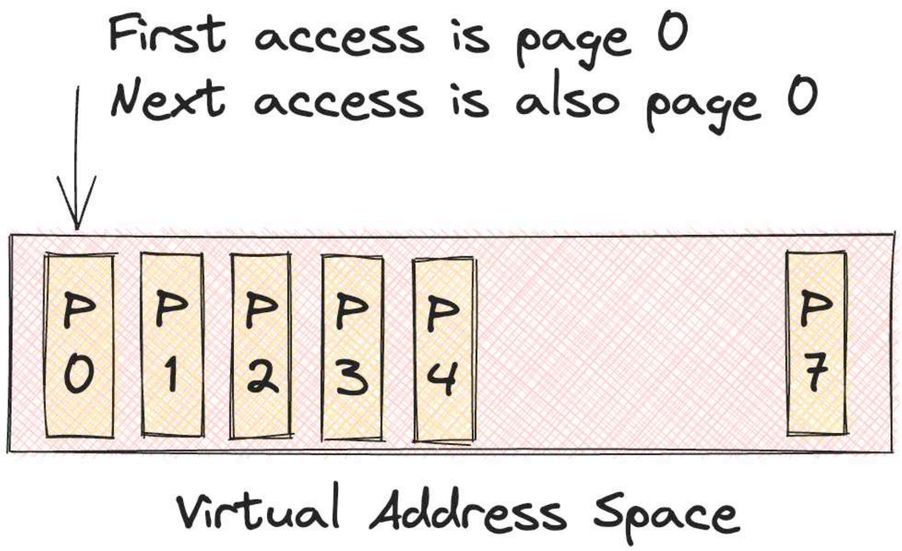
- Access to a[0] with VA 100 - How?
  - It will be a TLB miss!
- Access to a[1] will be a TLB hit!
- Access to a[3] will be a TLB miss!
- Access to a[4] to a[6] will be a hit!
- Access to a[7] will be a miss then a[8] and a[9] will be hit
- miss, hit, hit, miss, hit, hit, hit, miss, hit, hit
- Hit rate: 70% - Not bad!

	OFFSET 00	04	08	12	16
VPN = 00					
VPN = 01					
VPN = 03					
VPN = 04					
VPN = 05					
VPN = 06		a[0]	a[1]	a[2]	
VPN = 07	a[3]	a[4]	a[5]	a[6]	
VPN = 08	a[7]	a[8]	a[9]		
VPN = 09					
VPN = 10					
VPN = 11					
VPN = 12					
VPN = 13					
VPN = 14					
VPN = 15					

Locality as a factor

- **Spatial Locality**
  - If a program accesses memory at x, it will likely soon access memory near x.
  - What if page size was 32 bytes in previous example?
- **Temporal Locality**
  - An instruction or data that has been recently accessed will likely be re-accessed soon in the future





Handling TLB Miss?

Hardware Handler

- Hardware handles the TLB miss entirely on CISC
  - Older systems based on Intel x86
  - Hardware has to know where the page tables are stored
  - The base address is stored in Page Table Base Register
  - Hardware would walk the entire page table in parallel to find the correct PTE and **extract** the translation, **update** and **retry** instruction

Handling TLB Miss?

Software/OS Handler

- Software handles the TLB miss entirely on RISC
  - Hardware simply raises an exception on a miss (trap handler)
  - Trap handler for handling TLB miss is invoked
  - The code will look up the page table for translation, updates TLB with privileged instruction and return from trap
  - At this point hardware retries the instruction
    - **There is a subtlety to context switch here - Why?**

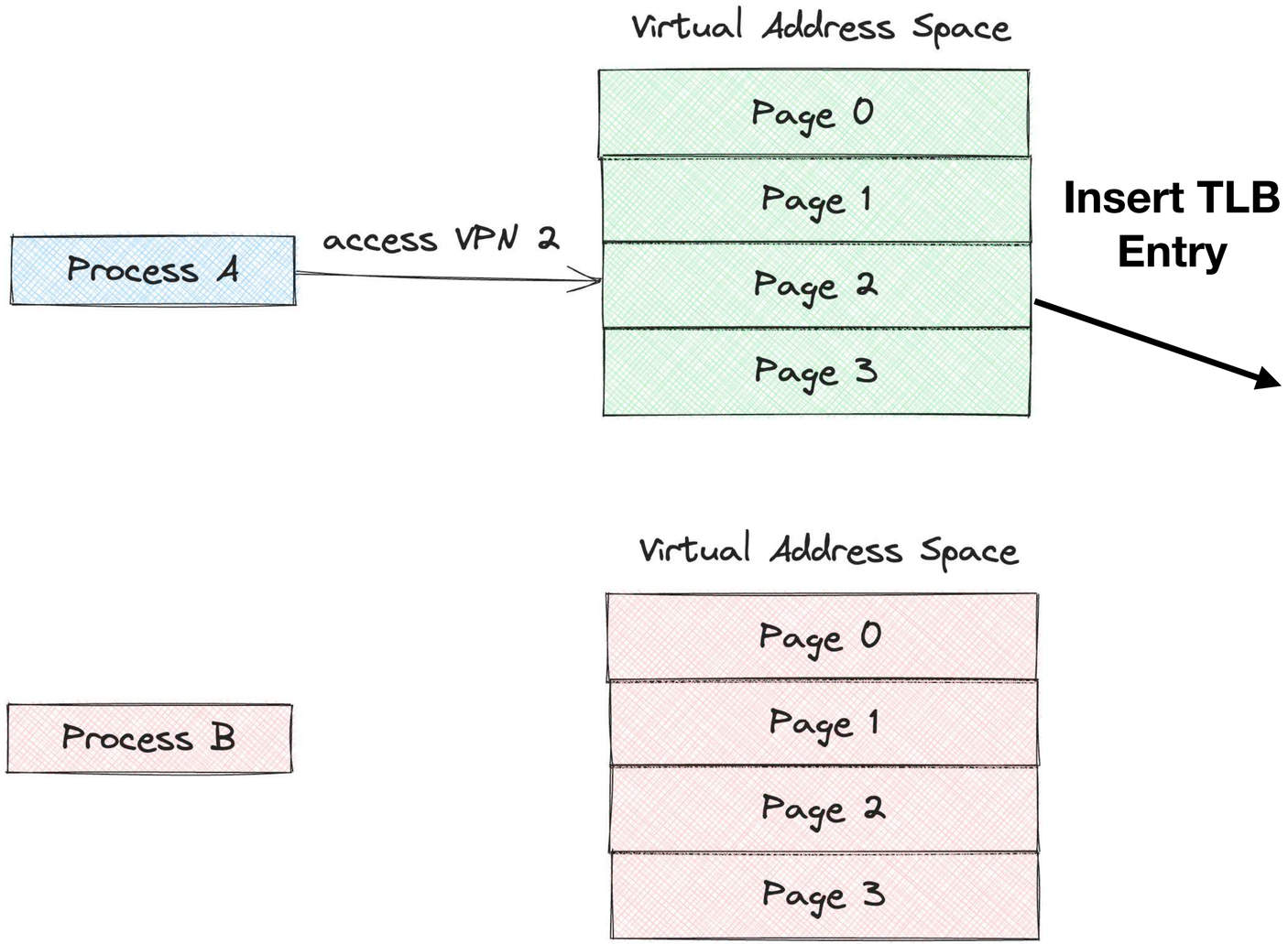
Some Caution!!

- When returning from TLB miss
  - Hardware must save the PC of current instruction and again execute the same instruction for **retry** resulting in hit
  - In usual scenario the return-from-trap goes to next instruction
- OS should not go into infinite chain of TLB misses
  - TLB miss handlers are also code that requires address translation
  - Keep TLB miss handlers in physical memory
  - Use some permanent translation slots for handler code

Inside TLB

- TLB is managed by fully associative method
  - A typical TLB have 32, 64 or 128 entries
  - Hardware searches the entire TLB in parallel to find the translation
  - Other bits
    - **Valid:** Enty has a valid translation
    - **Protection**: how a page can be accessed (read only, read, etc)
    - Other bits: **address space identifier**, **dirty bit**, etc.

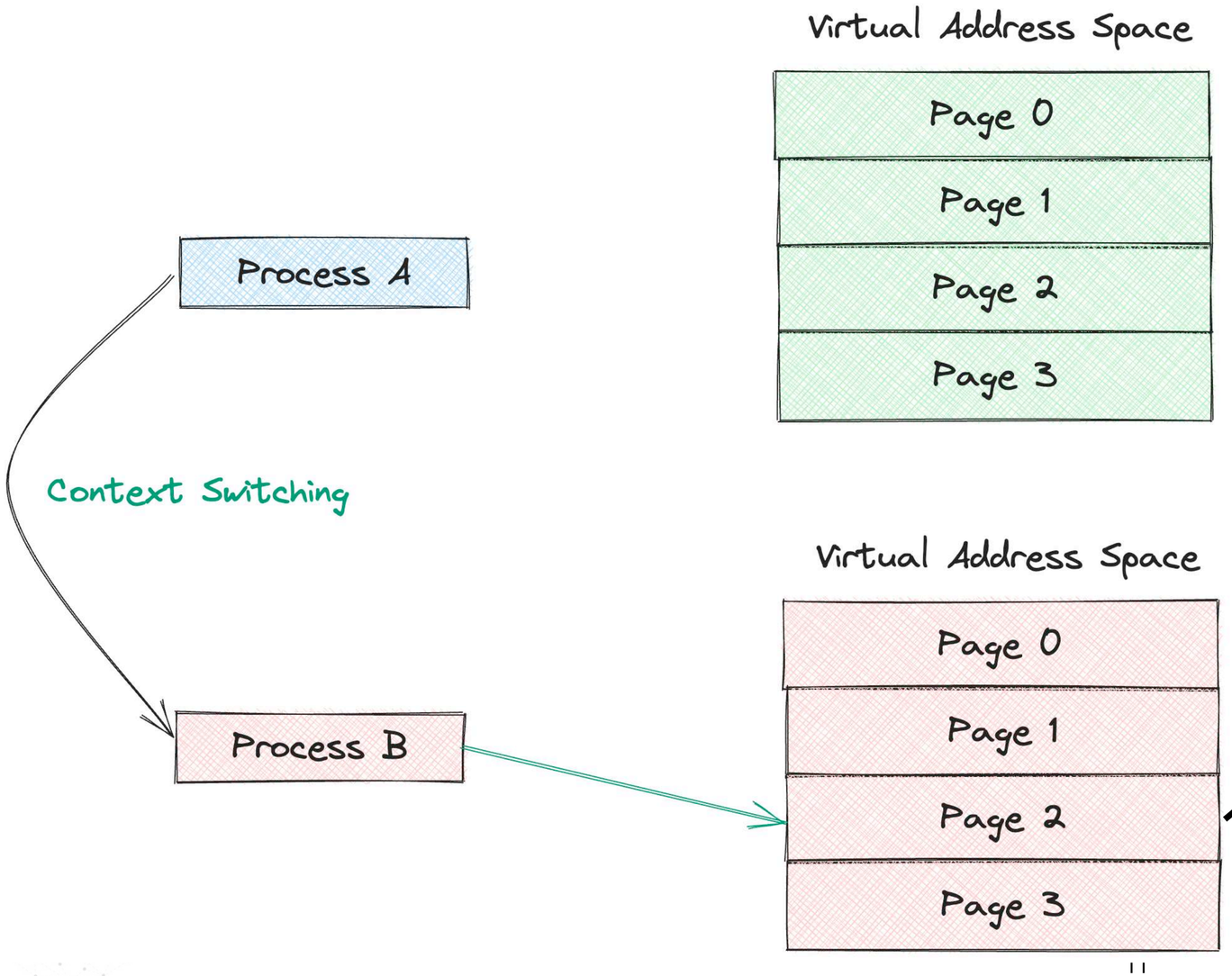
What about Context Switch?



TLB Table

VPN	PFN	Valid	Prot
2	100	1	rwx
...	...	...	...

Context Switching and TLB



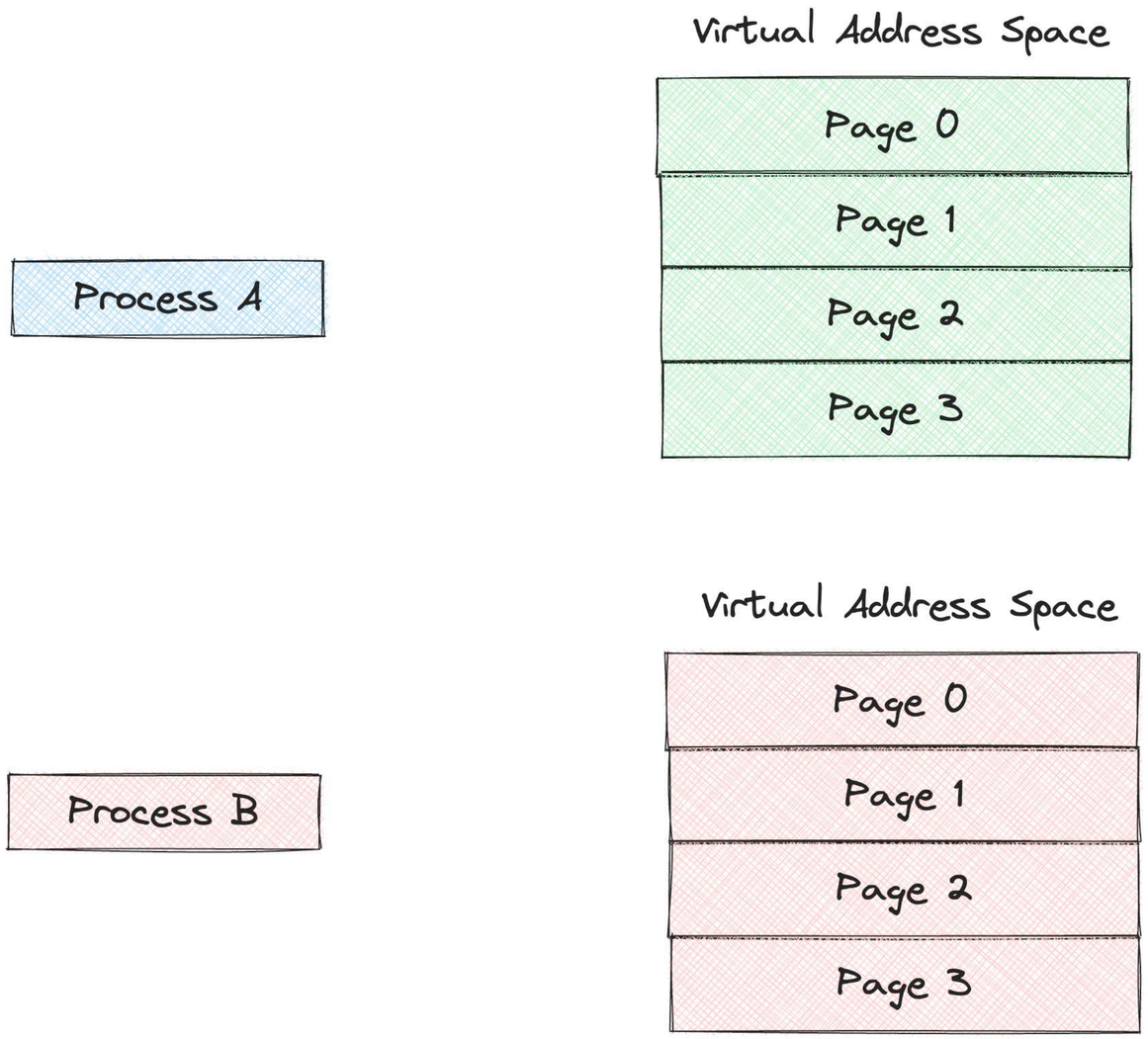
Flushing can be an option but… TLB miss?

TLB Table

VPN	PFN	Valid	Prot
2	100	1	rwx
…	…	…	…
2	180	1	rwx

Insert TLB Entry

Address Space Identifier



VPN	PFN	Valid	Prot	ASID
2	100	1	rwx	1
...	...	...	...	
2	180	1	rwx	2

Address Space Identifier (ASID) field to distinguish (8 bits)

Processes can Share a Page

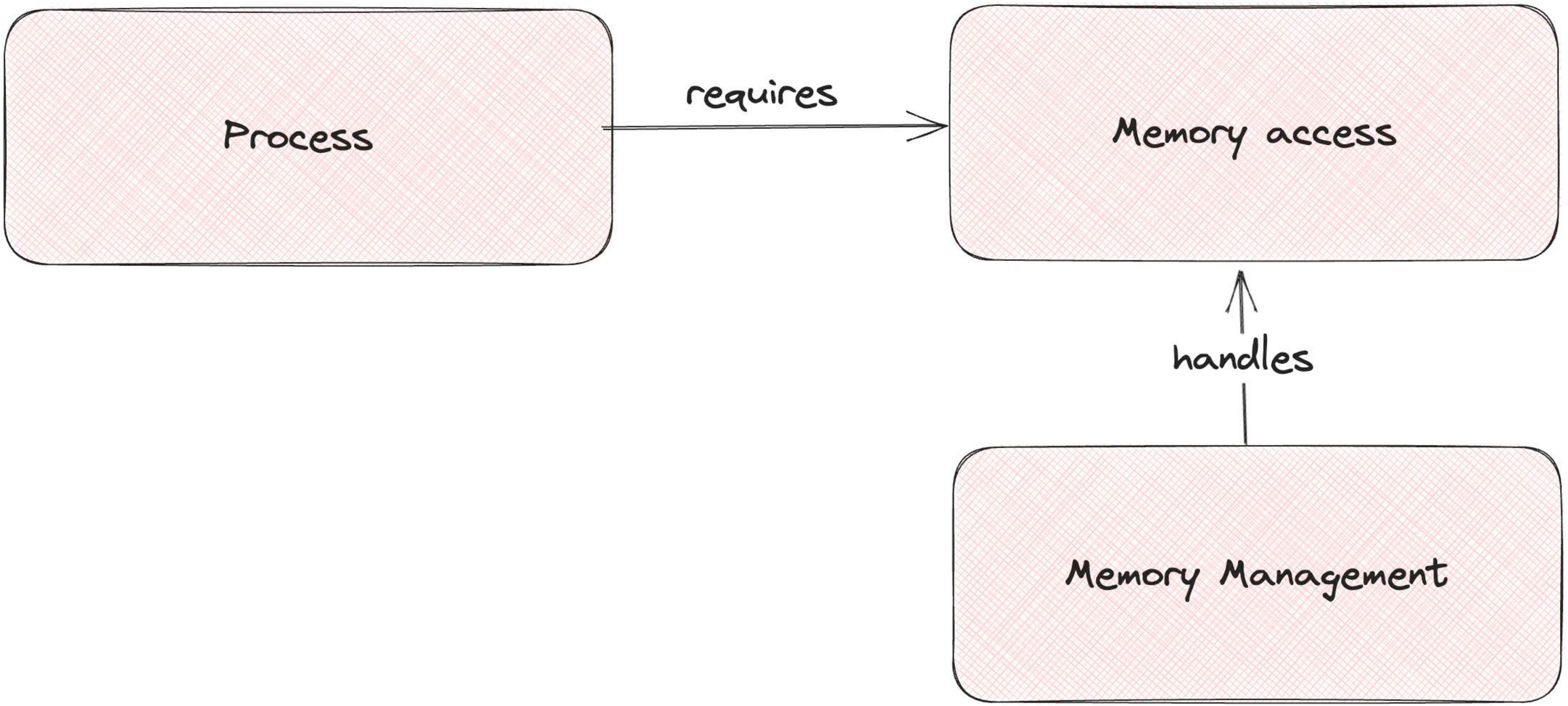
VPN	PFN	Valid	Prot	ASID
2	100	1	r-x	1
...	...	...	...	
10	100	1	r-x	2

- Sharing can be useful, it reduces the number of physical frames

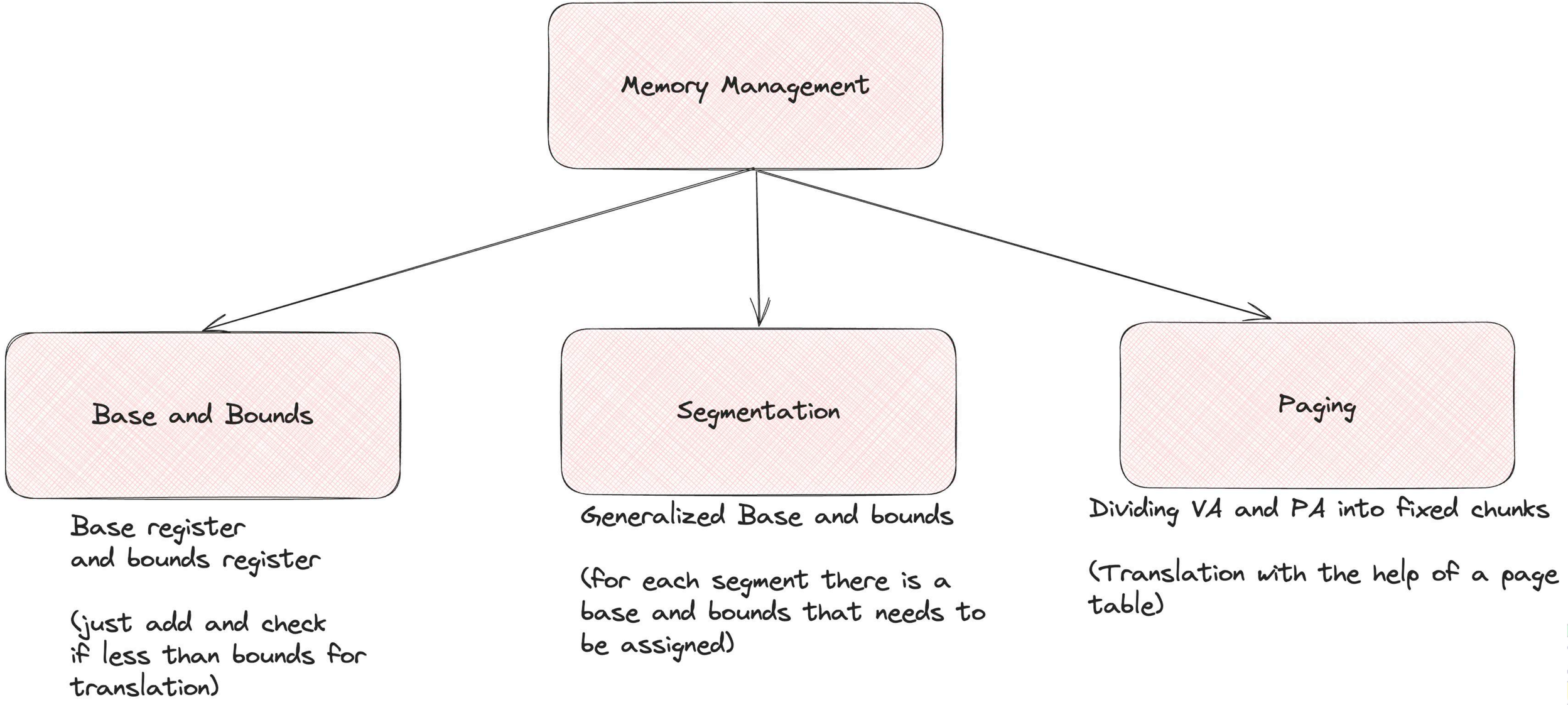
Still some issues!

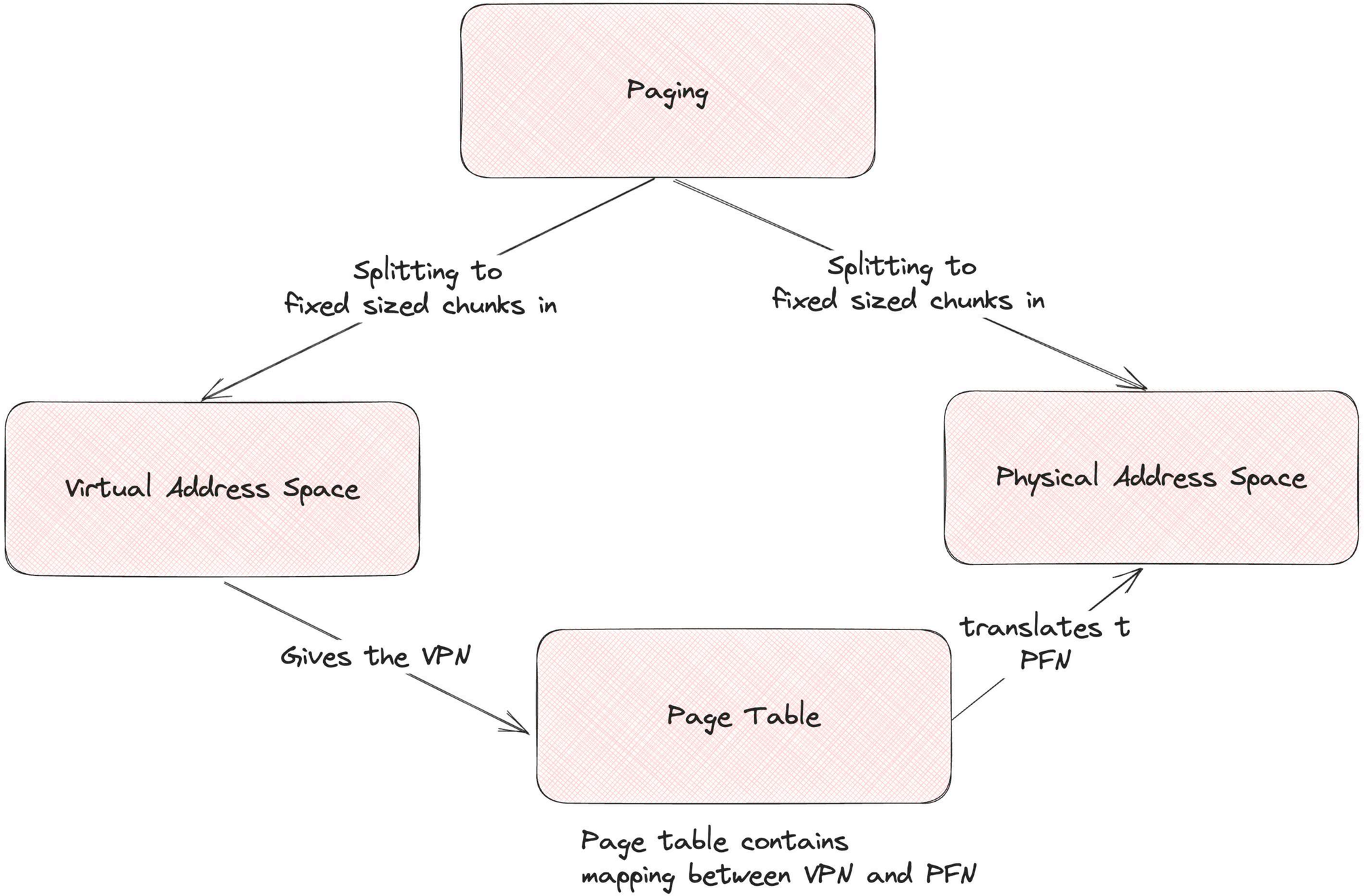
- TLB has limited size
  - Which entry to replace when adding new entries?
- What about the size of the page table?
  - Are there ways to minimise them?
  - Where will they be stored?

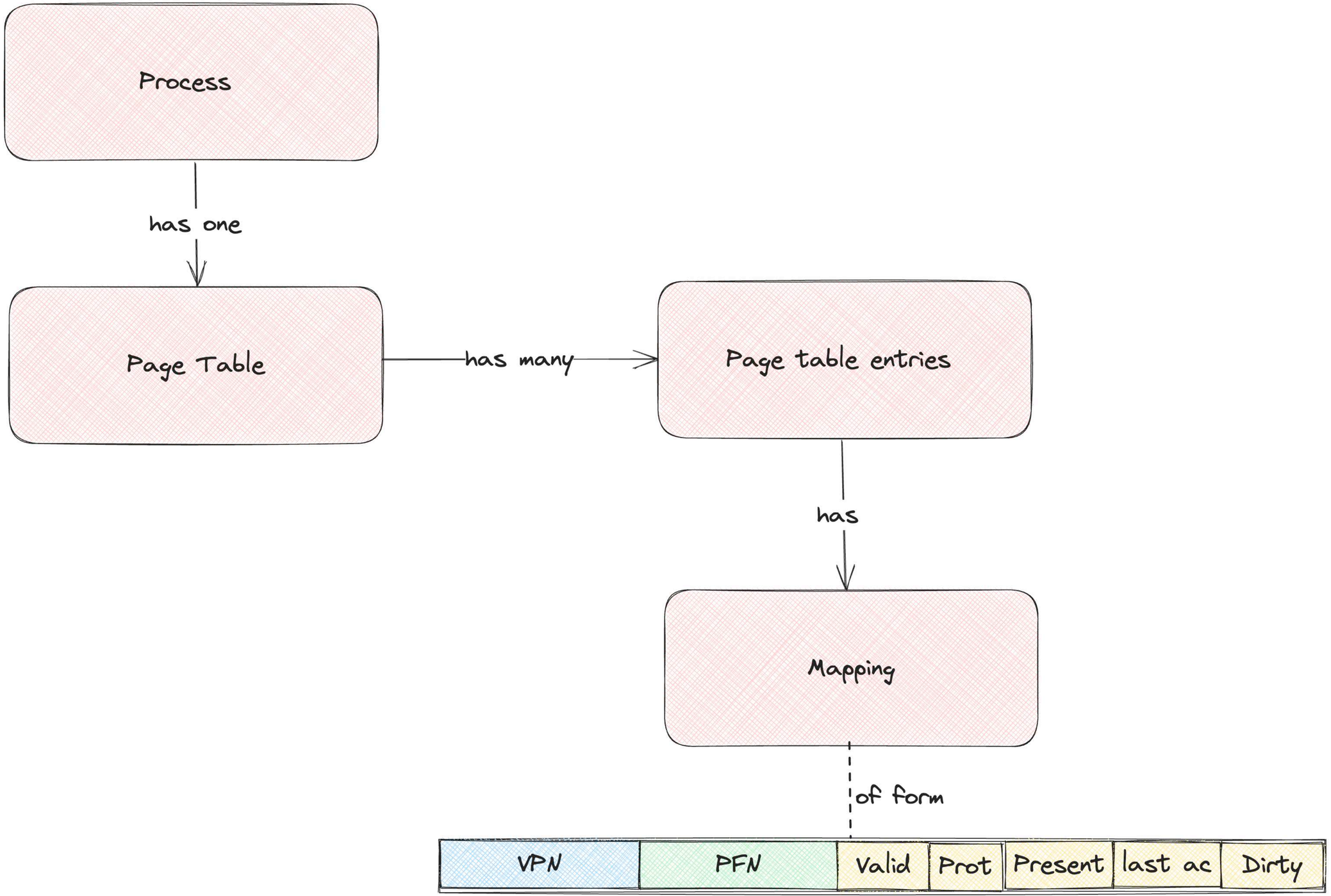
Memory Virtualization: High level view

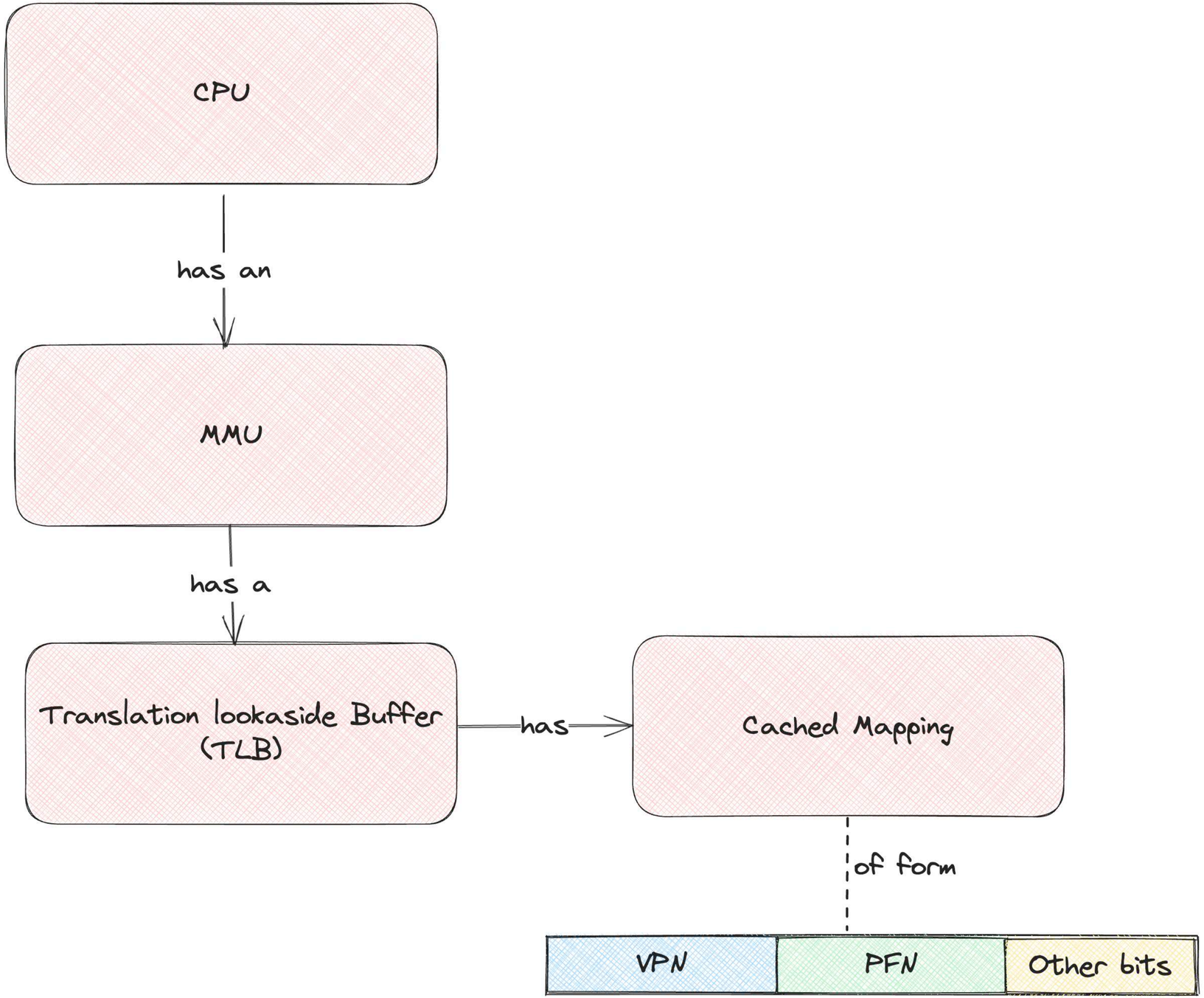


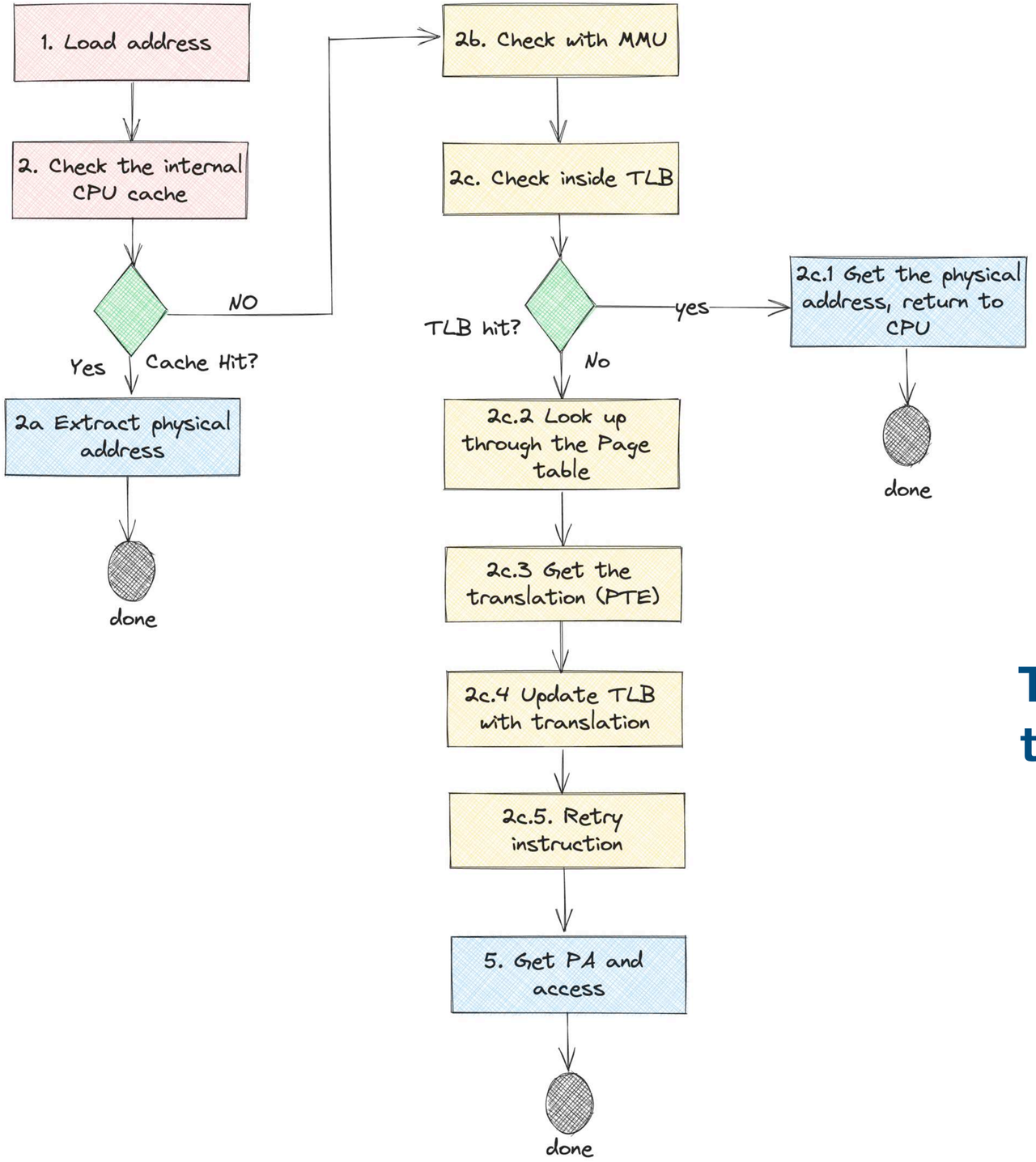
Memory Management: A Quick Recap




Paging
Splitting to
fixed sized chunks in
Splitting to
fixed sized chunks in
Virtual Address Space
Physical Address Space
Gives the VPN
translates t
PFN
Page Table
Page table contains
mapping between VPN and PFN


Process
has one
Page Table
has many
Page table entries
has
Mapping
of form
VPN
PFN
Valid
Prot
Present
last ac
Dirty


CPU
has an
MMU
has a
Translation lookaside Buffer (TLB)
has
Cached Mapping
of form
VPN
PFN
Other bits



The overall address translation Process

Size of Page Tables

- Let us take a guess about the size of a page table
  - Consider a 32 bit address space with 4 KB pages. What will be the number of bits for offset?
    - 12 (4 KB = 2^{12})
    - VPN: 20 bits, think about possible translations?
    - 2^{20} translations ~ **a million mappings**
    - If each mapping is 4 bytes => total 2^{20} X 4 = 4 MB per process per page table
    - What if there are 100 processes => **400 MB just for address translations!**

What can be done to manage size?

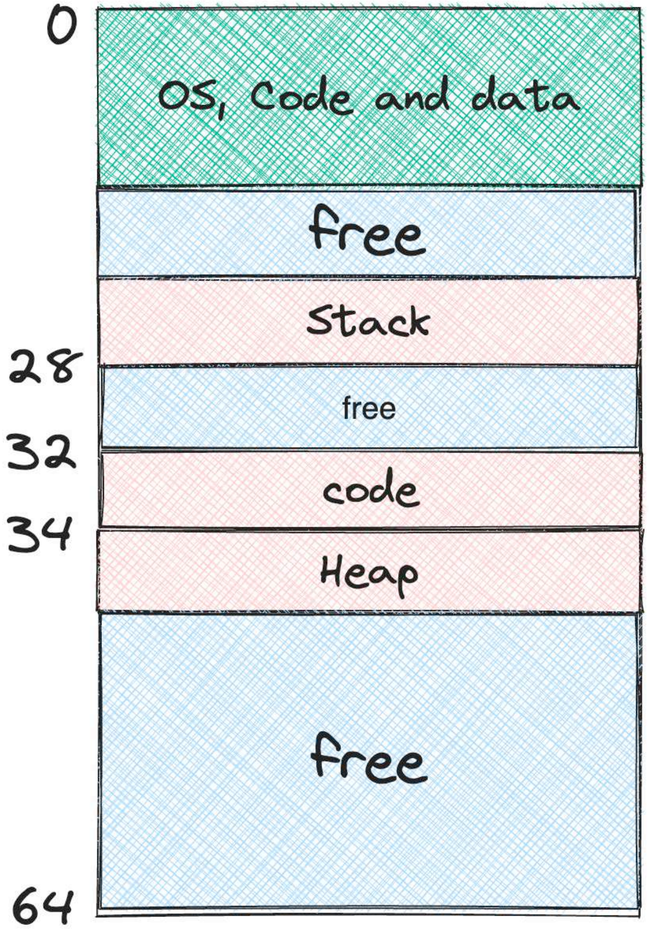
- Can we come up with smaller page tables?
  - Linear page tables did help but they can be really big!!
- Simple solution: Why not larger pages?
  - Assume a 32 bit address space but instead of 4 KB pages how about 16 KB pages
  - 16 KB = 2^{14} (Offset -> 14 bits, VPN -> 18 bits)
  - 2^{18} entries in the page table ~ **1 MB per page table** (assuming 4 bytes per entry) - Almost 1/4 reduction!!

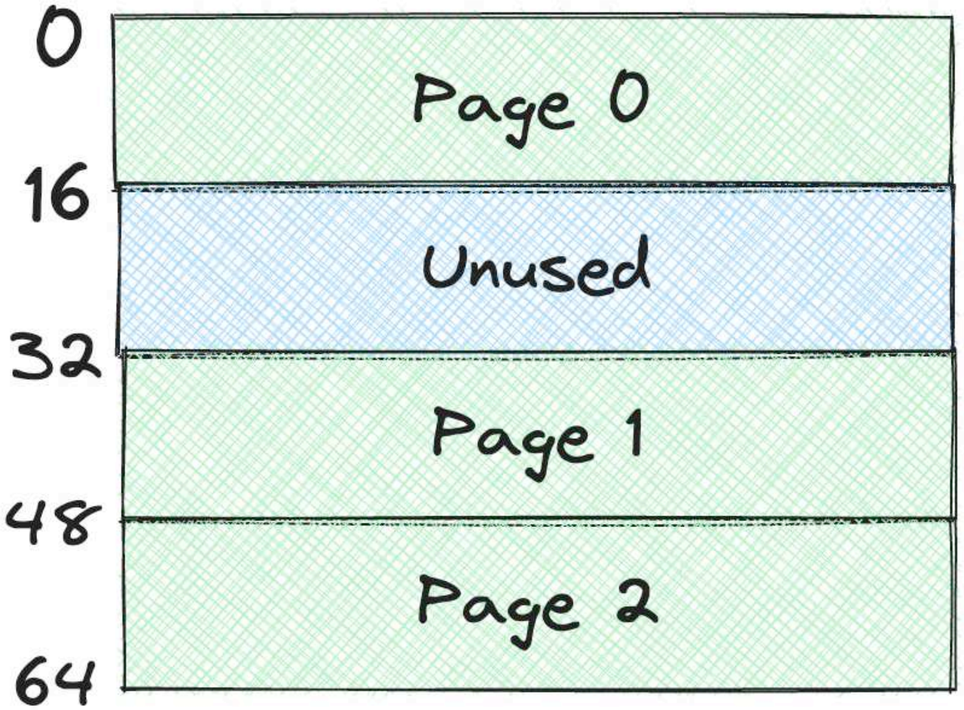
Increasing Page Size: How good is it?

- Larger page sizes can suffer more issues with internal fragmentation
- Applications overly gets filled up with large pages and there will be empty sizes
- Due to this issue, most system relatively small page size
  - Common size: 4 KB or 8 KB
- Can we think of something else?
  - How about thinking in the lines of segmentation and paging?

Hybrid approach: Paging and Segmentation

Can we get the best of two worlds?



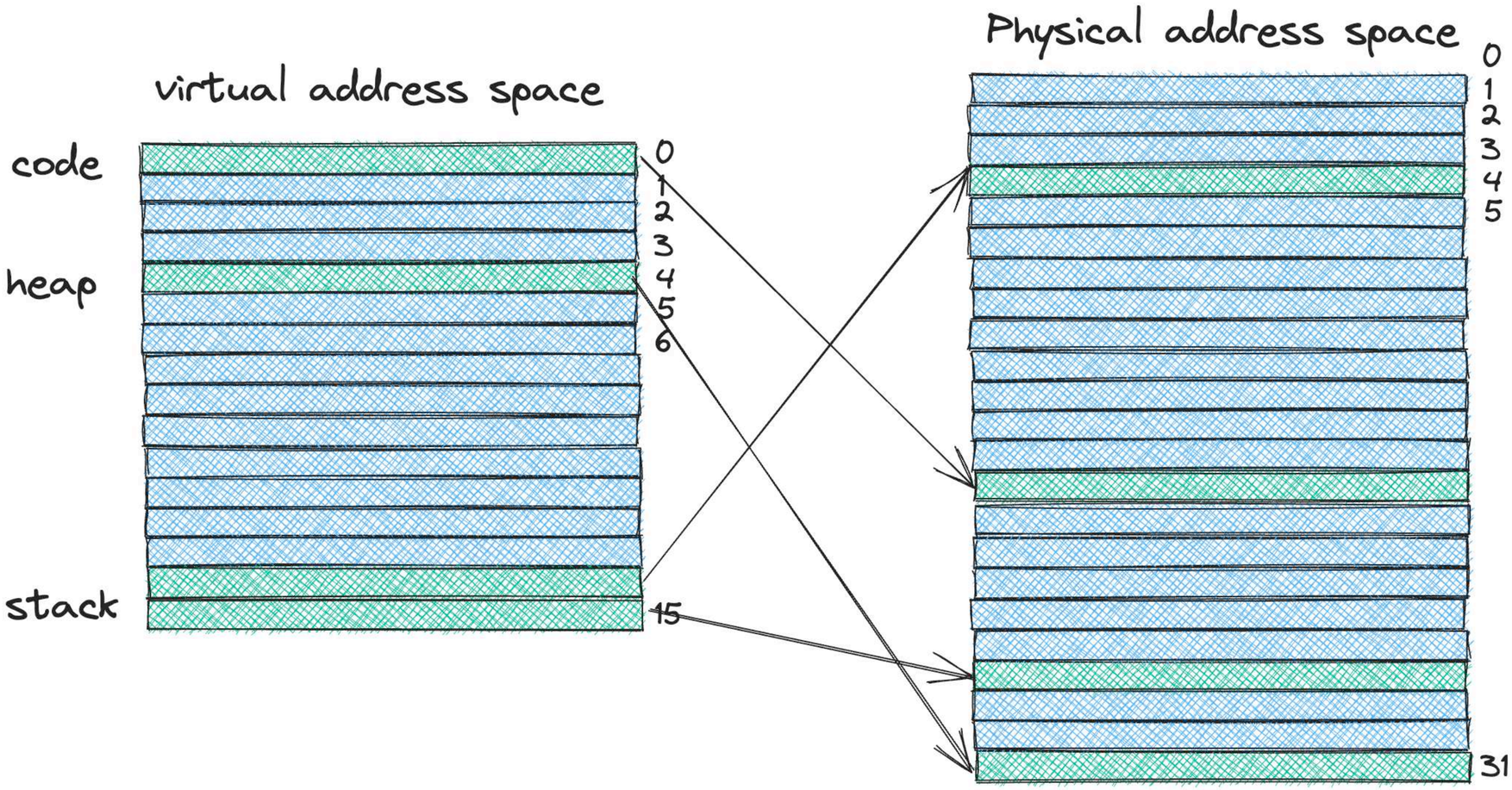


Hybrid Approach: Paging and Segmentation

- Jack Dennis, the creator of Multics had such an idea
  - Construction of Multics virtual memory system
  - Combine Paging and Segmentation was the key aspect
- This can have a good reduction in the size of page table
  - What are the key segments in a process?
  - What was the approach used in Paging?
- Let us revisit linear page table!

Revisiting Linear Page table

- Consider a tiny 16 KB address space with 1 KB pages



Most of the entries will be invalid

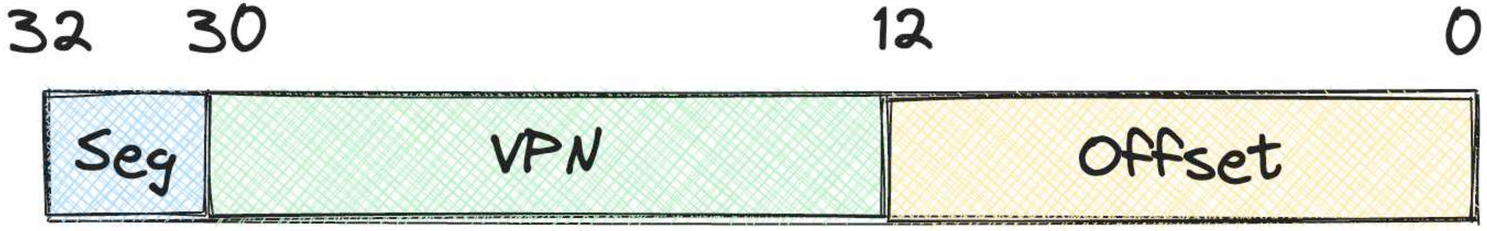
Lots of space Will be wasted!!

Hybrid Approach: Paging and Segmentation

- Instead of having one page table per process?
  - Can we have it for every segment?
  - There will be total of 3 pages
  - With segmentation
    - It was one base register
    - One bound register
  - In hybrid, there will be those registers in the MMU - What would be this?

What about registers?

- Base register and bounds register
- Base register is about base of the page table that correspond to the segment
- Bound indicates the end of the page table
- Assume 32 bit VA space with 4 KB pages and address space split into three segments
- How to do addressing?



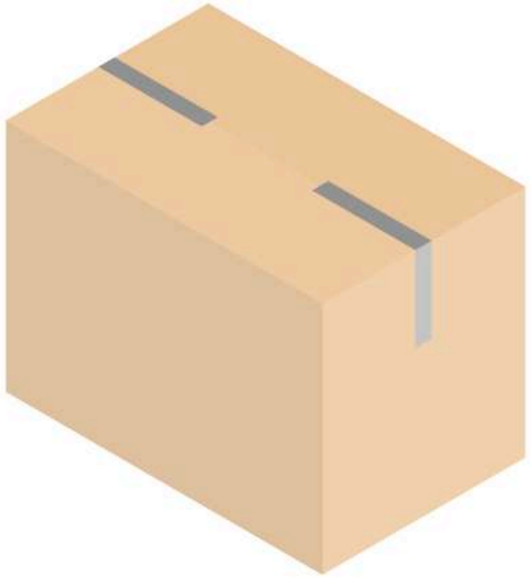
How good is the Hybrid Approach

- The good parts
  - Ensures significant memory savings compared to linear page table
    - Any reason?
    - Unallocated size between stack and heap no longer takes up space in page table?
- Some issues
  - Segmentation is used - Not quite flexible as paging
  - The problem of external fragmentation comes back - **Page tables** can be of arbitrary size. Finding memory for them can be difficult

Going back to the Analogy

Over the idea of categories

Use fixed sized shelves

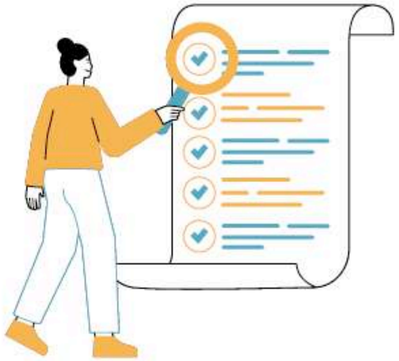


Each box can contain some items
The boxes are placed inside fixed sized shelves

Some mapping between product, box and shelves

Going back to the Analogy

Warehouse



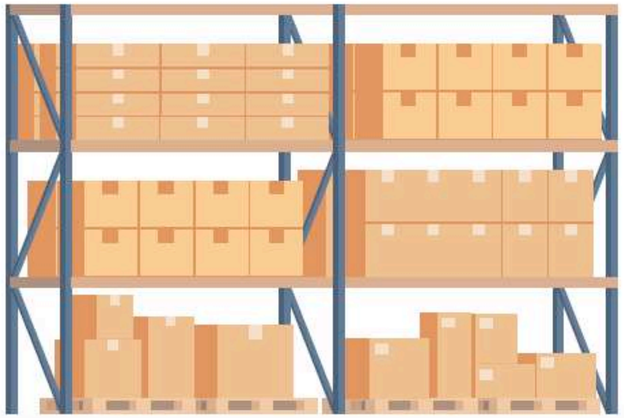
Location List

Index

TV

Index

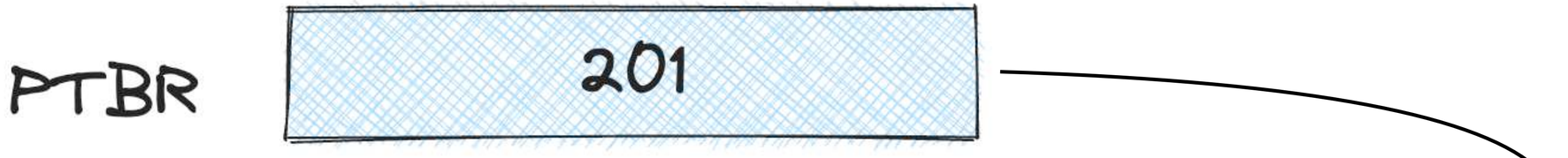
Living Room Furniture



Introducing Multi-level Page Tables

- In simple terms - Turn the page table into a tree like structure
- Chop up the page table into page sized chunks
  - If an entire page of page table is full of invalid entries, don't allocate that page of the page table at all.
  - To track, use a simple data structure, **Page Directory**
- Page directory can specify where the Page of page table is located
  - It allows to ensure that a part of the page table contains no valid pages

Lets revisit Linear Page Table in Action



Valid	Prot	PFN
1	rw	15
1	rwx	16
0	-	-
0	-	-
0	-	-
1	rx	100
1	rwx	25

Multiple levels of Mappings are Required

PDBR

200

Valid	PFN
1	201
0	-
0	-
1	204

The Page Directory (PFN: 200)

Page 1 and 2 are not allocated - PDBR?

Page 0 of PT (PFN 201)

Valid	Prot	PFN
1	rw	15
0	rwx	16
0	-	-
0	-	-

Page 3 of PT (PFN 204)

Valid	Prot	PFN
1	rx	100
0	rwx	25
0	-	-
0	-	-

Multi-Level Page Tables has Advantages

- Allocate page table space promotional to the address space being used
- Each portion of the page table fits neatly within a page
  - Easier to manage memory
  - OS can simply grab the next free page when it needs to grow the page table
- Compare with linear page table indexed by VPN requires contiguous memory
  - If page table is large, finding such block of memory may be hard!
- Multi-level page table also comes with a cost - Any guess?

Multi-Level Page Table comes with Overhead

- TLB Miss Scenario
  - There will be two loads from the memory
  - One for the Page directory and one for PTE itself
  - **Trade-off** between time and space
  - Smaller page tables, yes but not free! - Lookups are costly
- Another issue is complexity
  - Building the multi-level tables and complicated lookups

An Illustrative Example

- Consider an address space of 16 KB with 64-byte pages
  - 16 KB: 2^{14} as bits
  - 14 bit virtual address space
  - 64 byte implies - 6 bits for offset
  - 8 bits remaining for VPN
  - Assume that 0 and 1 are for code, 4 and 5 for stack, 254 and 255 for heap
- If it was full table then 256 entries - Assume each entry 4 bytes ~ 1 KB table

An Illustrative Example

- Can be divided into 16, 64 byte pages
- Each page can hold 16 PTE (16 *4 = 64)
- 256 entries spread over 16 pages
- Number of entries in the Page Directory: 16 How?
- Number of bits required to index into directory: 4 How?
- Number of bits required to index into PTE: 4 How?

Representation

Page Directory		Page of PT (@PFN:100)			Page of PT (@PFN:101)		
PFN	valid?	PFN	valid	prot	PFN	valid	prot
100	1	10	1	r-x	—	0	—
—	0	23	1	r-x	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	80	1	rw-	—	0	—
—	0	59	1	rw-	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	—	0	—
—	0	—	0	—	55	1	rw-
101	1	—	0	—	45	1	rw-

An Illustrative Example

- Eg: Address translation
  - 11 1111 1000 0000
  - 8 bits VPN: 4 bits for PDR, 4 bits for getting to PTE
  - Last 6 bits offset
  - Try to work it out!

Solution

- 1111 points to 101
  - PT corresponding to PFN 101
- Next step is to get the PTE
  - VPN (1110) corresponds to 14th entry
- PFN of the 14th entry is 55!
  - Add offset to 55 PFN to get the exact physical address!

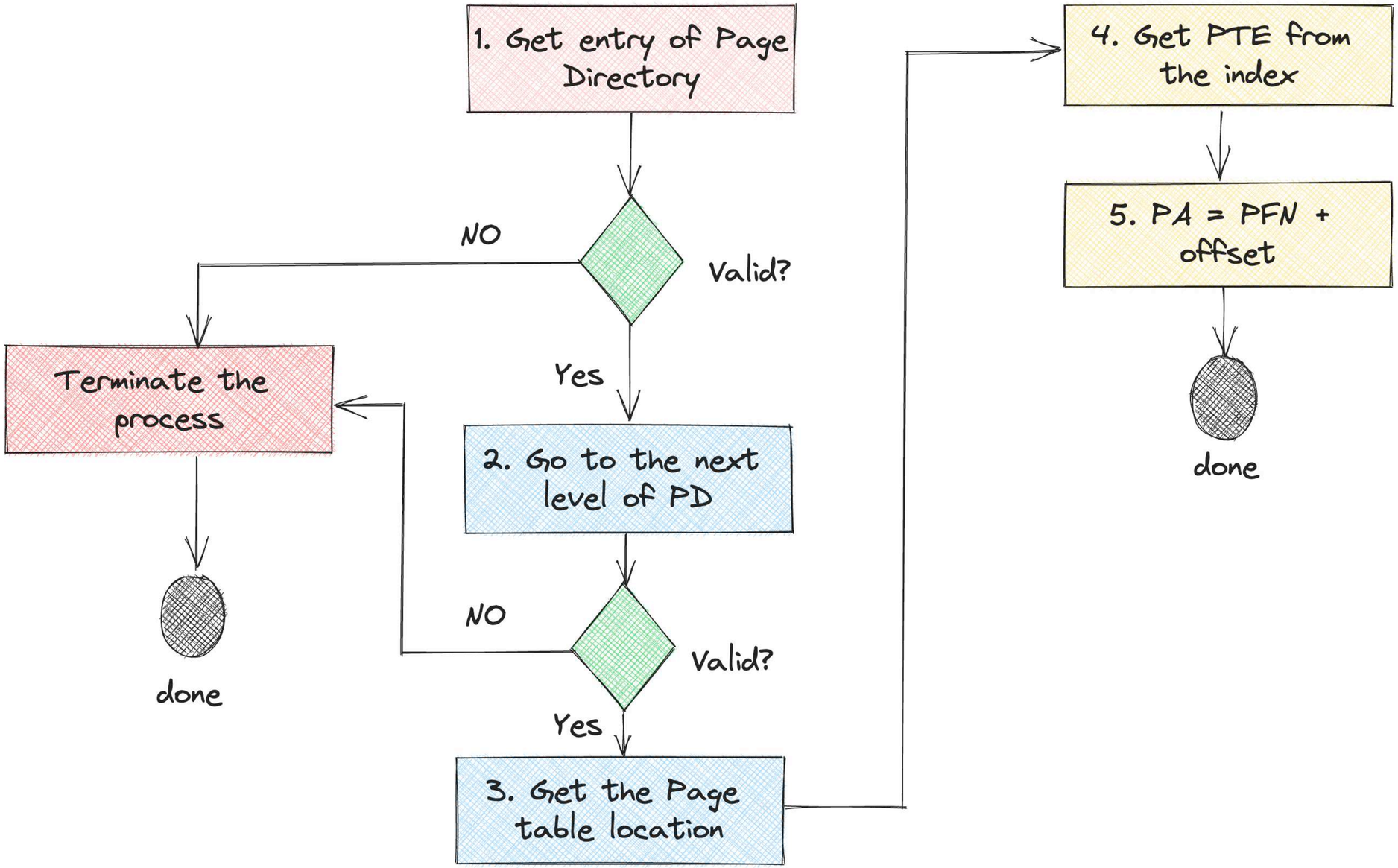
More than Two Levels

- Its not about just two levels in **Multi-level page tables**
  - Deeper tree is possible
  - Why is it required?
- Assume a 30 bit Virtual address space and 512 byte pages
  - Virtual address has 21 bits for VPN and 9 bits for offset
  - Assuming PTE is 4 bytes, 128 PTE can fit on single page
  - 7 bits required to index into PT to get the PTE
  - Remaining 14 bits for mapping to get PT from PDR!!

Multi-level page tables

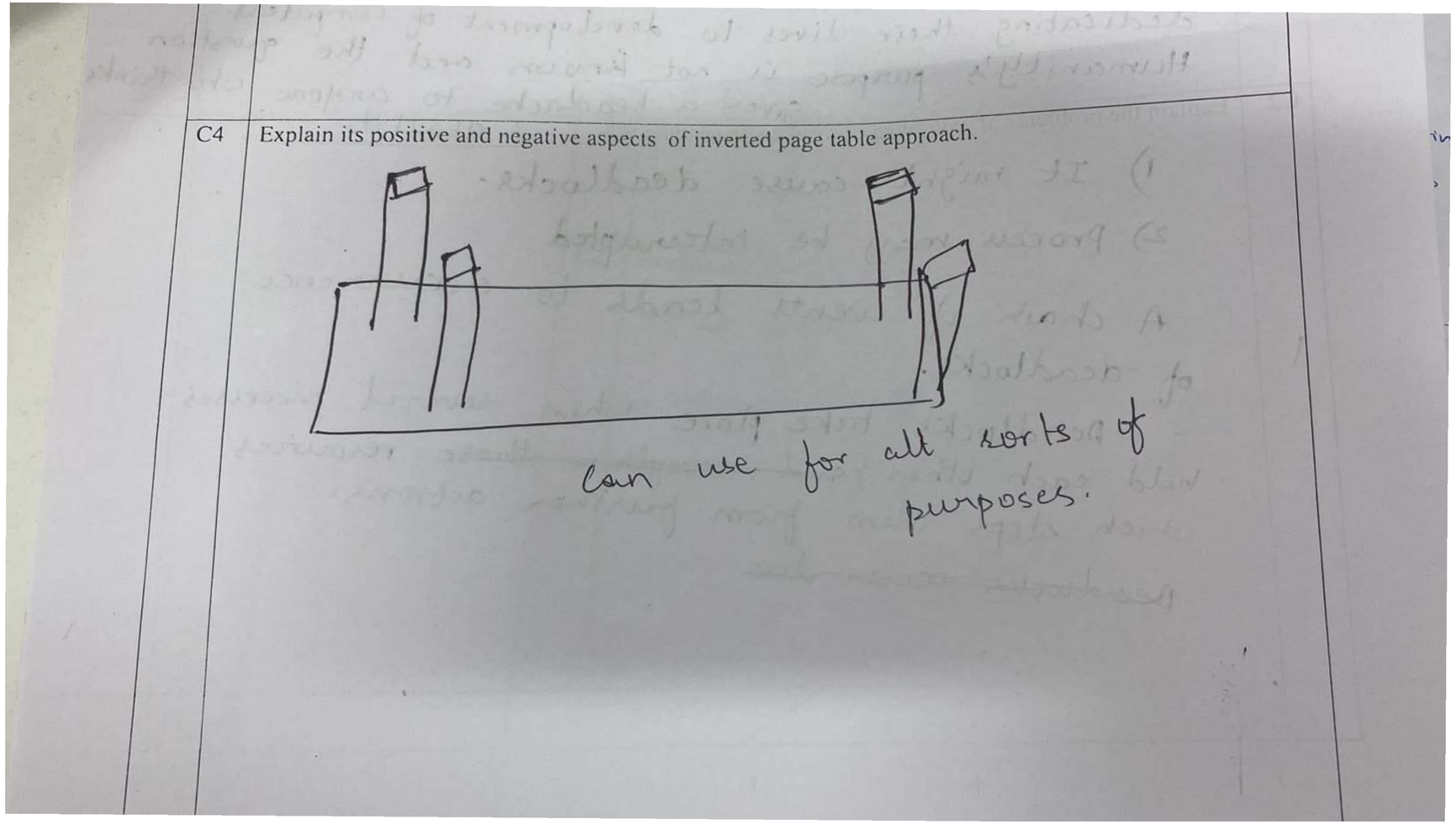
- Page directory itself is divided into multiple pages
  - One additional meta directory on top of all directories
  - The top most one points to the right directory
  - The directory further points to the correct Page table to fetch PTE
  - The PTE and offset is combined to generate physical address
- In case of TLB miss, the memory look up will be multi access
  - In two level itself it can be two additional access

The overall flow



Inverted Page Tables

Not this one!



Inverted Page Tables

- Instead of having one page table per process
  - Have one single page table for all the processes
  - Searching for an entry is like searching through entire table
  - Linear scan can be very expensive!!
  - **Remember:** Page tables are just data structures
    - All crazy combinations can be though off! (Hash tables)

Some more questions needs answers!

- What if the size of the address space is larger than the physical space?

- Are all pages of all active processes always in main memory?
  - All of them may not fit in the main memory - so how does it work?

- OS uses a part of the disk (swap space) to store pages that are not active in use - How does that work?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Memory Virtualization - Paging: Mechanisms and Policies

Karthik Vaidhyanathan

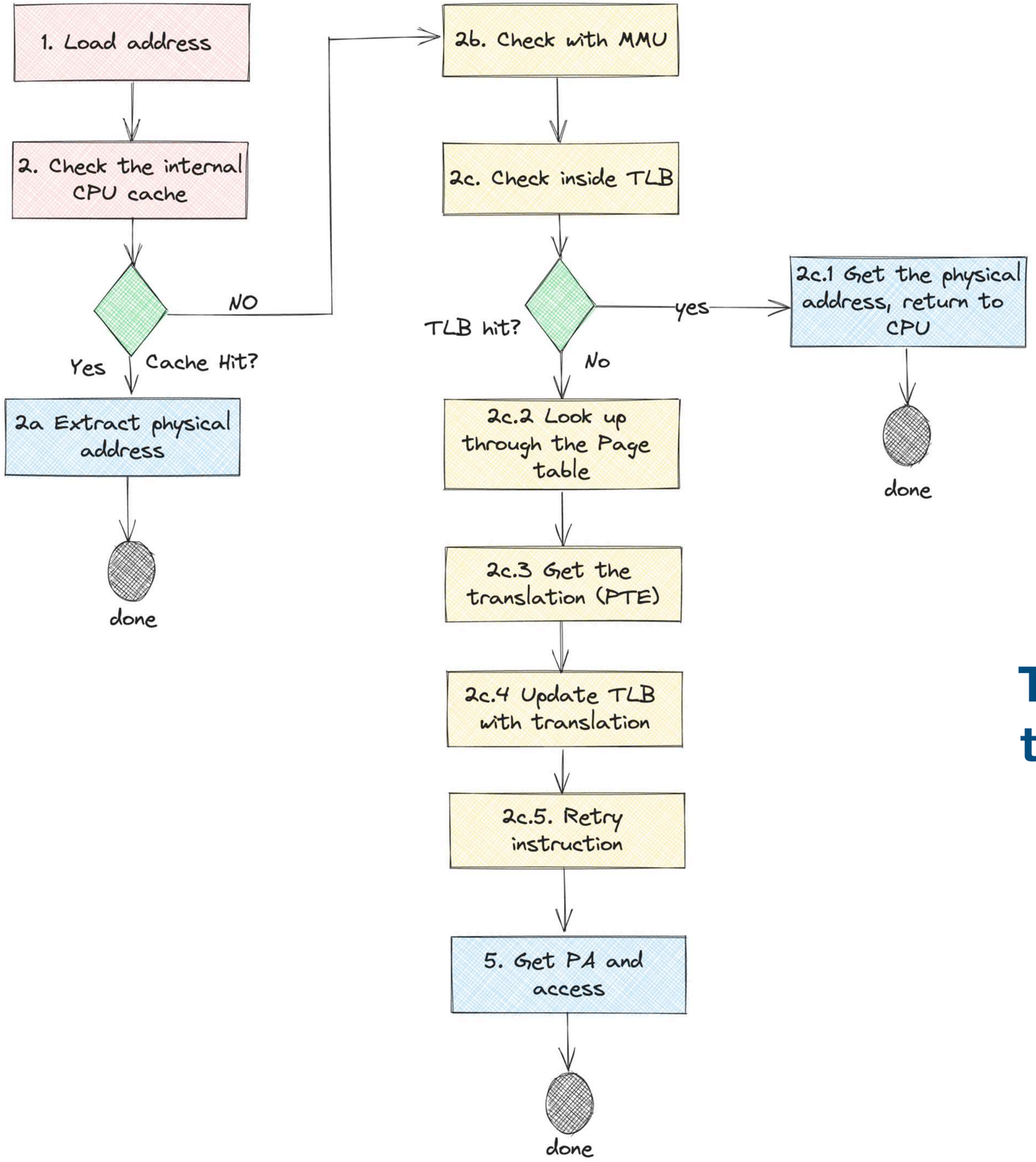
https://karthikvaidhyanathan.com

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

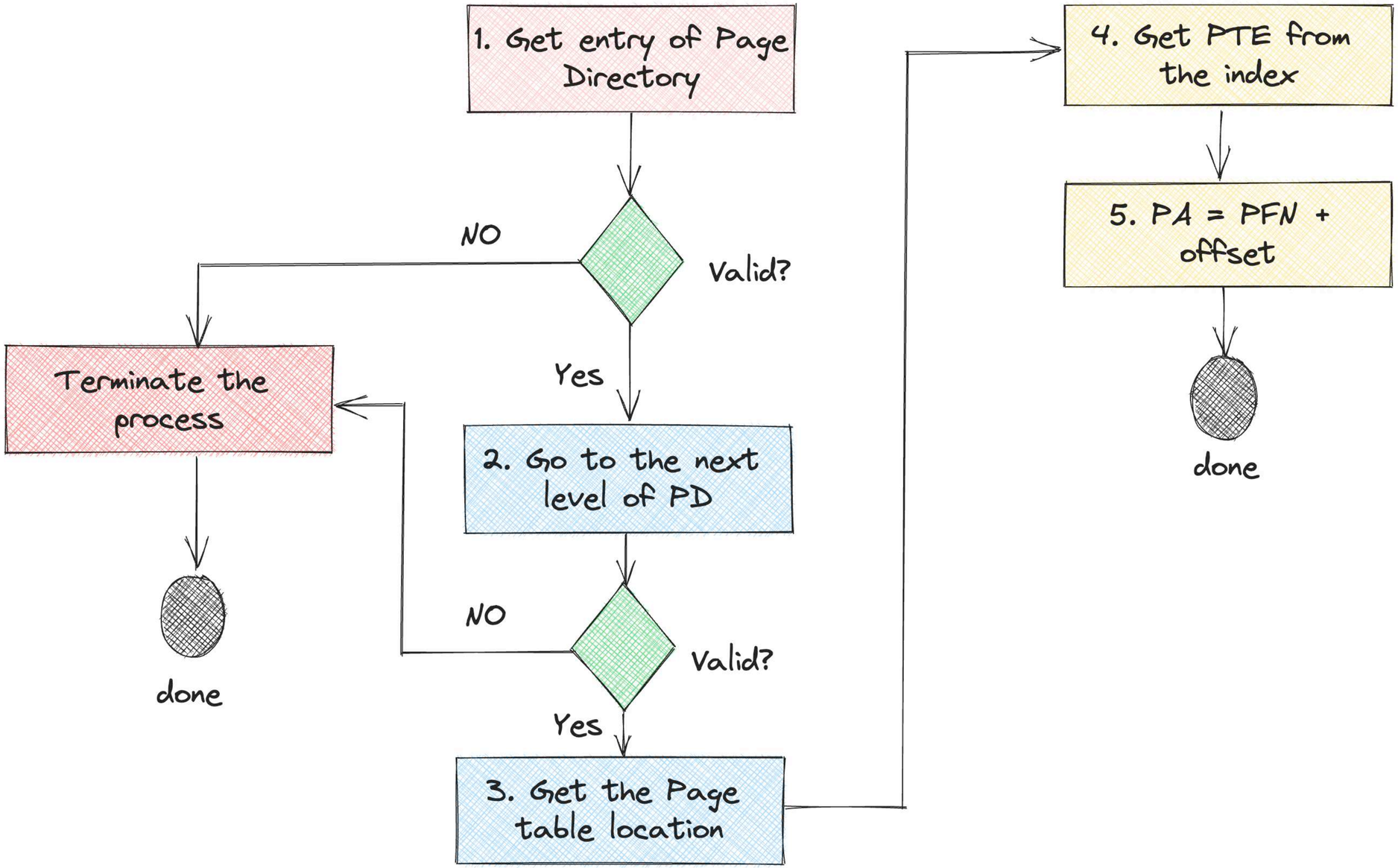
Sources:

- Operating Systems: In three easy pieces, by Remzi et al.
- Lectures on Operating Systems by Youjip Won, Hanyang University



The overall address translation Process

The overall Flow of Multi-level page tables



Some more questions needs answers!

- What if the size of the address space is larger than the physical space?

- Are all pages of all active processes always in main memory?
  - All of them may not fit in the main memory - so how does it work?

- OS uses a part of the disk (swap space) to store pages that are not active in use - How does that work?

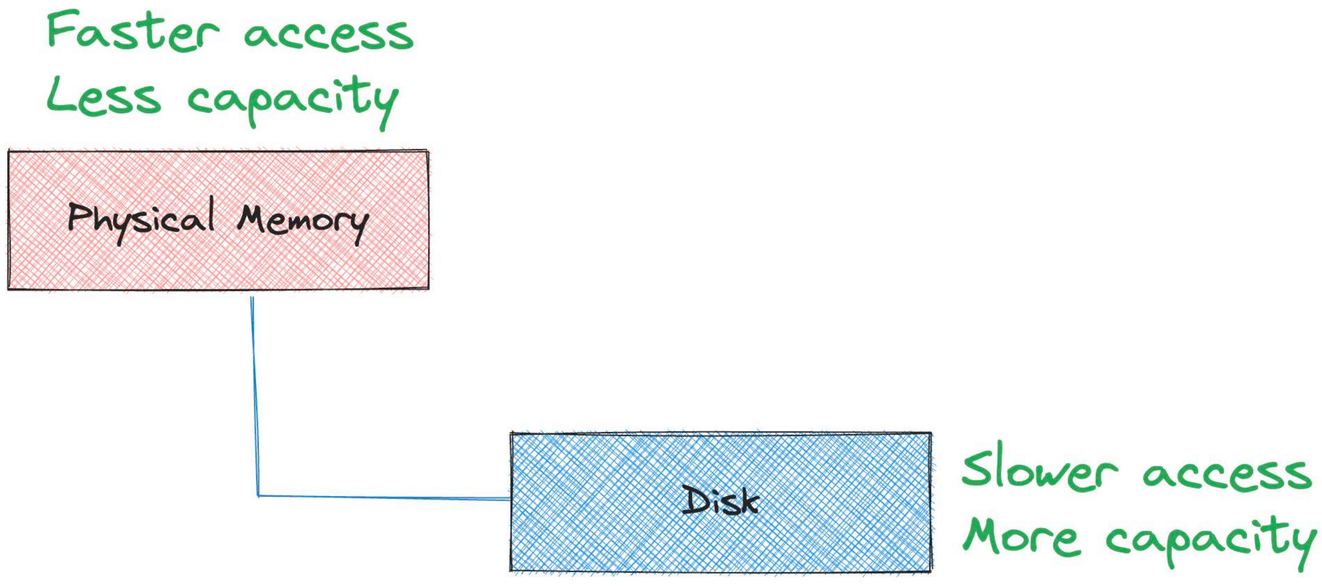
What about Memory Hierarchy?

- Not every page needs to be available in physical memory
  - Use additional level of memory
  - OS can stash away portions of address spaces that are currently not in demand
  - What is something that has more memory than physical memory?
    - What about hard disk? But how?

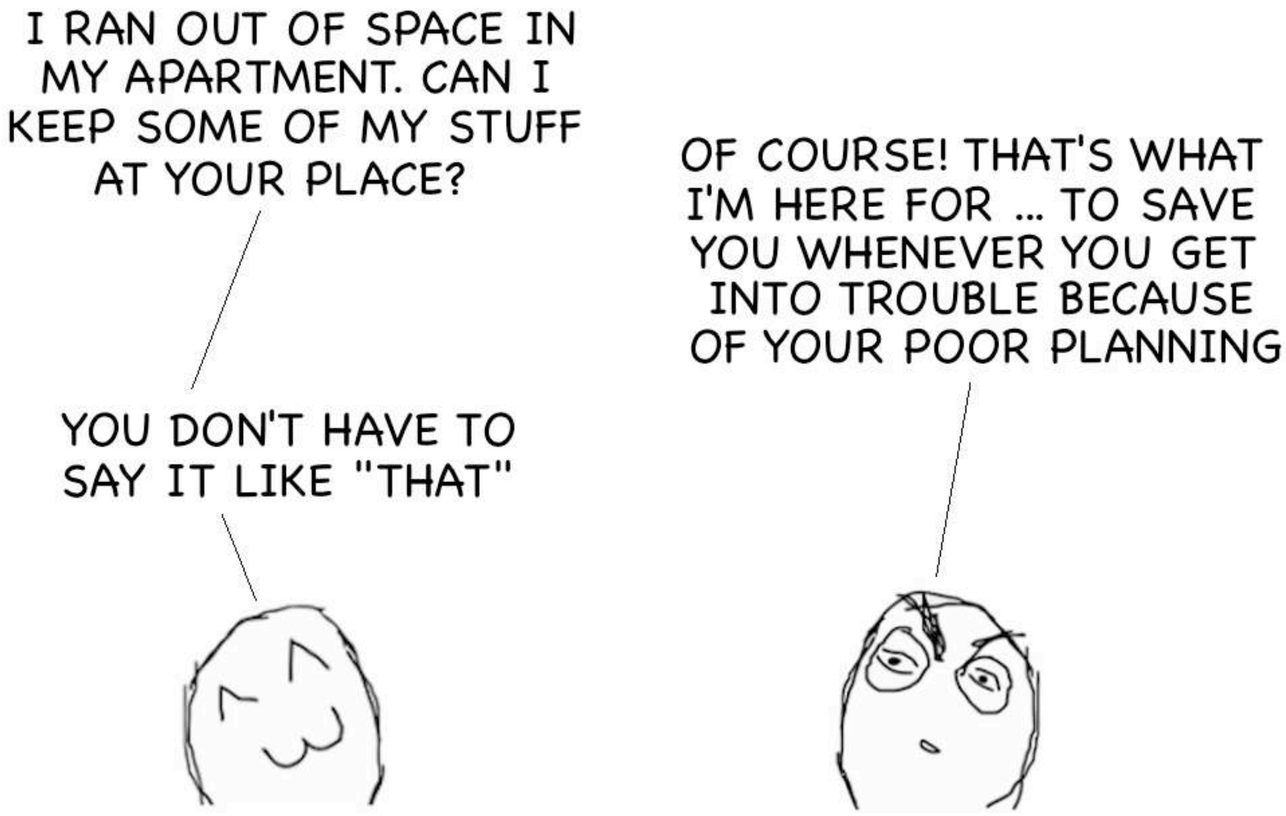
Moving between Physical Memory and Disk

Demand Paging

- How can OS make use of the slower, larger and faster, smaller device more effectively?
- This allows to provide the illusion of large virtual address space
- Think about manually moving pieces of code and data in and out of memory as needed



Leverage Swap Space!!

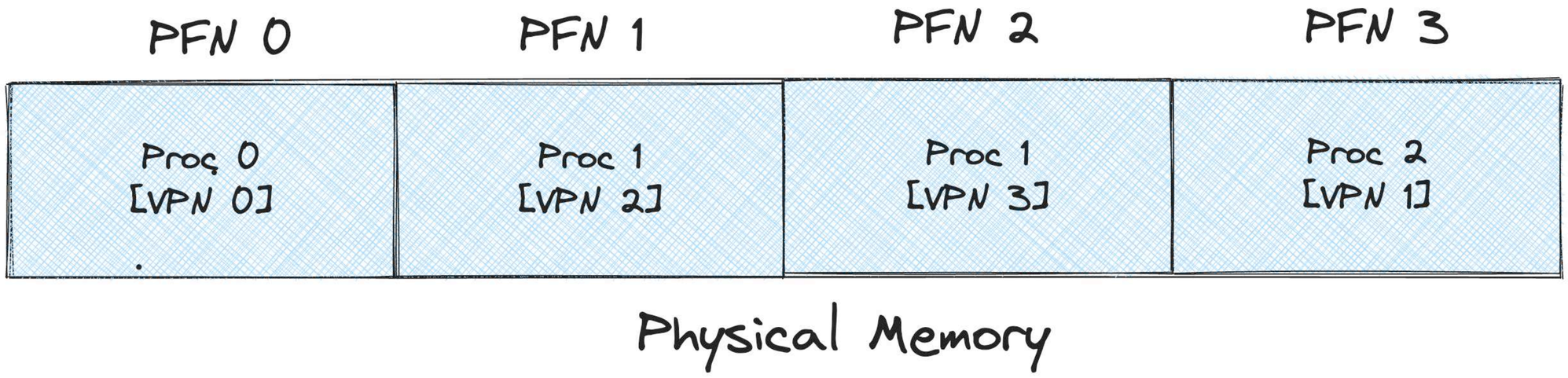


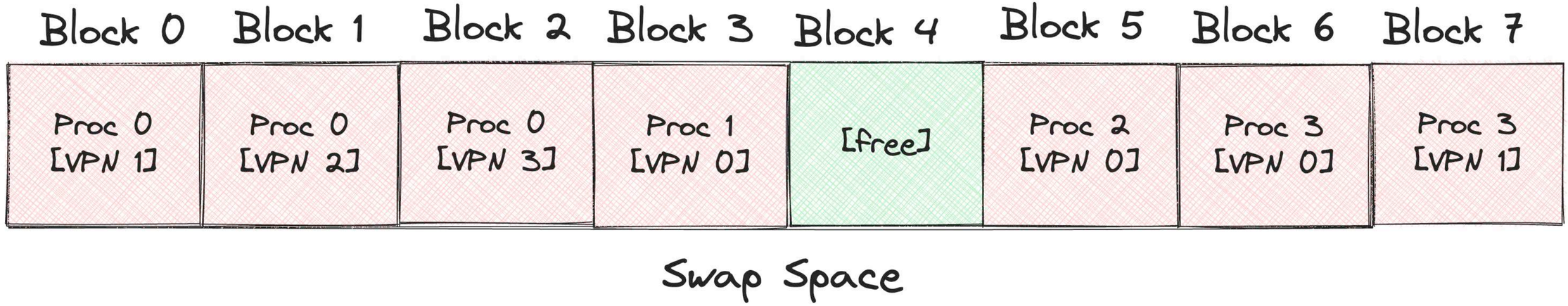
https://prateekvjoshi.com/2015/03/21/how-to-add-swap-space-on-ubuntu/

Leverage Swap Space

- Enables OS to swap out some pages to **swap space (in disk)**
- This is much needed in multi-programming
- Physical memory cannot hold all the pages - Always limited
- **Swap Space:** Dedicated space in the memory which can be used by OS to swap in and out pages
- OS should be able to read and write page sized units from/to swap space
- How does OS remember the location of page in swap space?

Swap Spaces


PFN 0
PFN 1
PFN 2
PFN 3
Proc 0
[VPN 0]
Proc 1
[VPN 2]
Proc 1
[VPN 3]
Proc 2
[VPN 1]
Physical Memory


Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Proc 0
[VPN 1]
Proc 0
[VPN 2]
Proc 0
[VPN 3]
Proc 1
[VPN 0]
[free]
Proc 2
[VPN 0]
Proc 3
[VPN 0]
Proc 3
[VPN 1]
Swap Space

Swap Spaces

- The use of swap space allows OS to give perception that process has abundant memory
- OS can take from memory and add it to swap space
  - Swapping provides a big support to OS for memory management
  - How to make use of swap?
    - When to use swap?
    - How to make memory management work?

Can we not leverage present bit?

- In usual scenario, if there is TLB miss
  - OS gets the PTE from the page table
  - But if we need to use swap
    - The OS needs to know if the page is in the physical memory
    - Leverages the use of **present bit**
    - If present bit is 1, page is in the page table
    - If present bit is 0, page is in the swap space

Page Fault

- The act of accessing page that is not there in the physical memory
- When a page is not in the physical memory
  - Hardware does not know how to handle it, raises exception
  - The OS has to service the page fault
  - Piece of code to achieve this - **Page Fault Handler**
  - This needs to be done both in the case of hardware or software-managed TLB

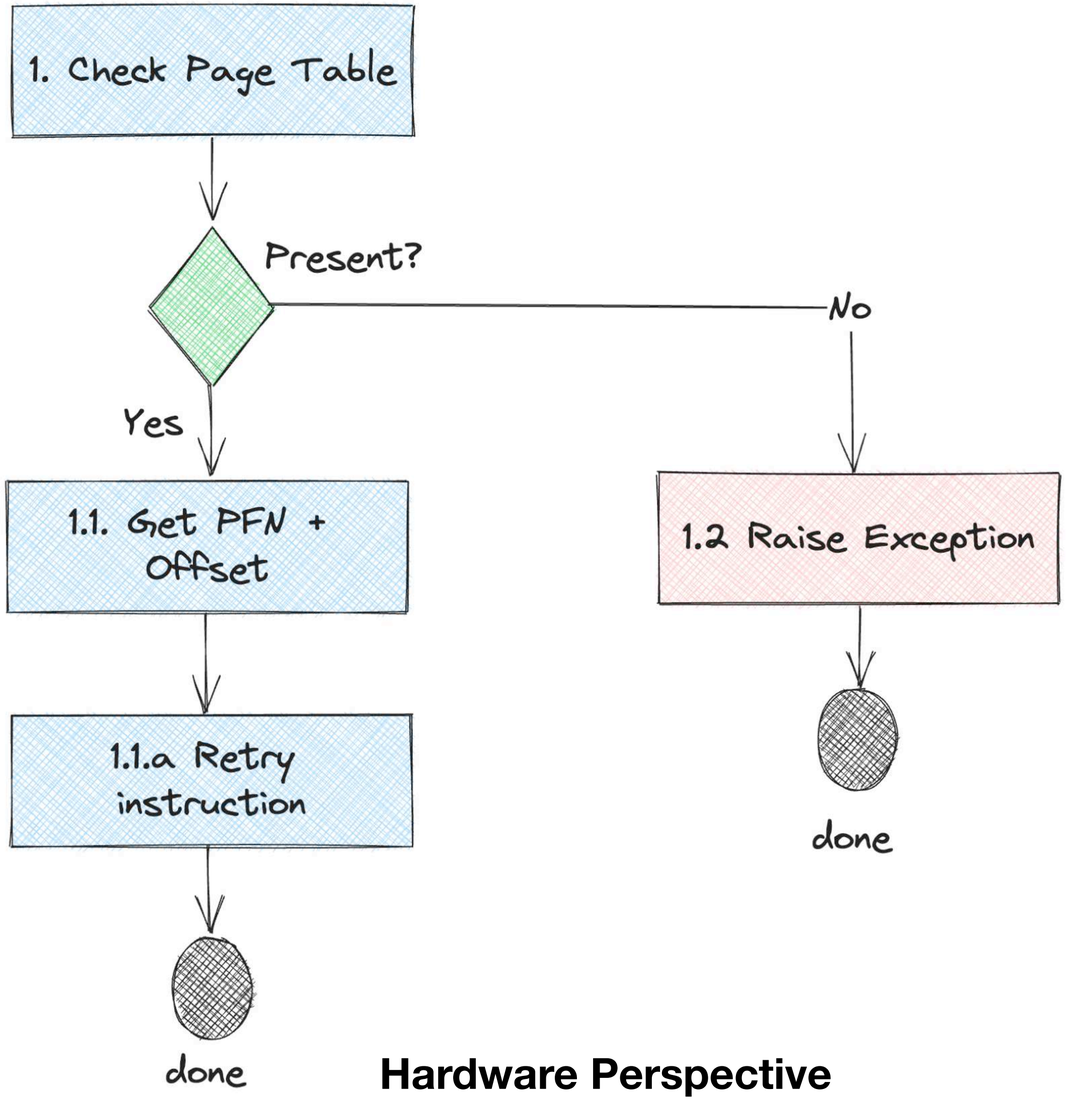
Page Fault

- How does OS know where to find a page during page fault?
  - OS can make use of bits in PTE (PFN) to store such information
  - On page fault
    - OS searches through PTEs
    - Gets the address from the PFN
    - Page fault handling involves Disk I/O
    - There is a possibility for **Context-switch** - How?

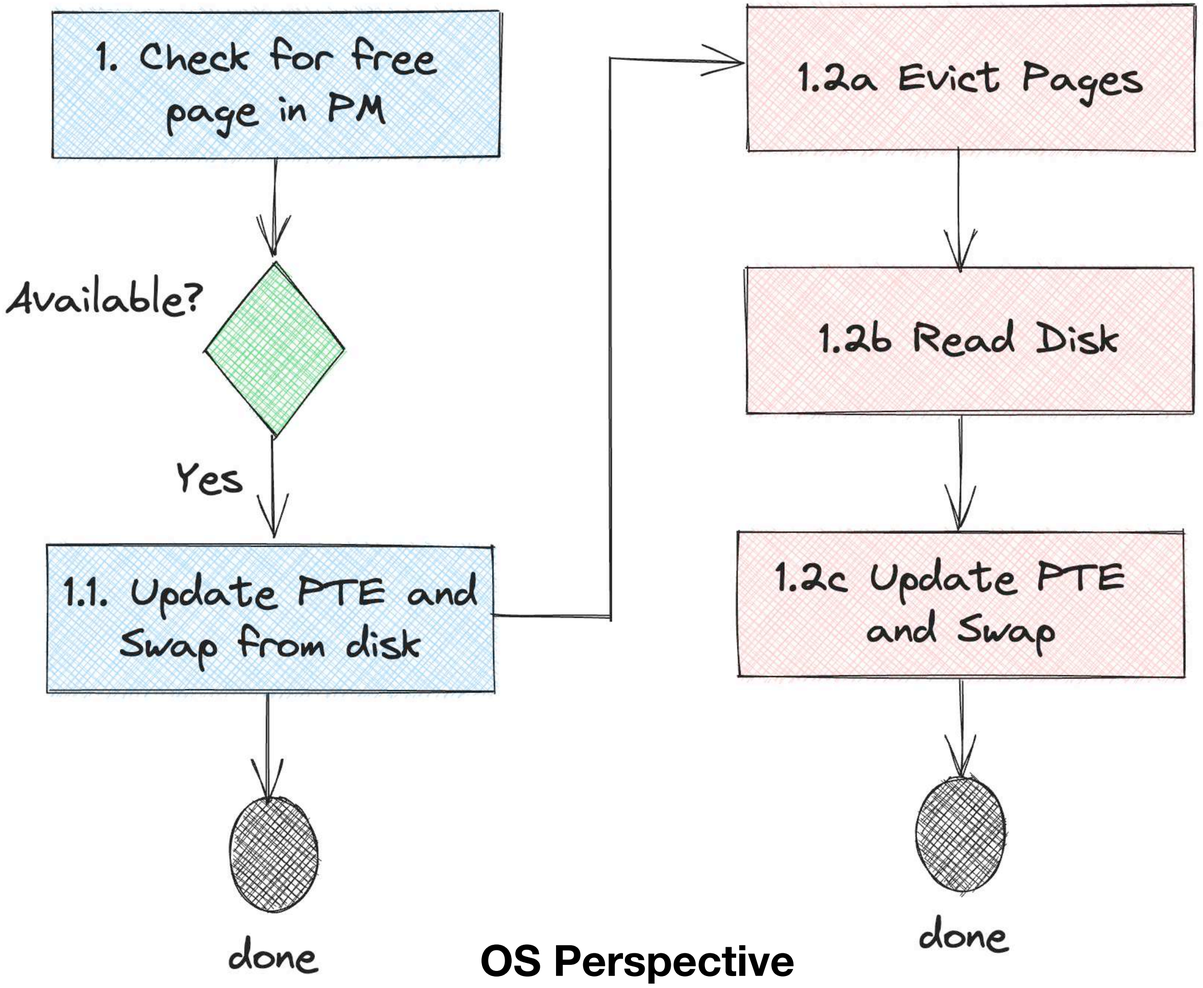
Page Fault and Context Switches

- On page fault
  - OS has to handle the fault
  - The process will be moved to a **blocked** state
  - OS will be free to run other process
  - On the servicing of page fault -> Another process can be executed
- What if the memory is full?
  - Some pages has to be moved out of physical memory
  - The process of swapping pages in/out from/of memory - **Page Replacement!**

Page Fault Control Flow



Hardware Perspective



OS Perspective

Software Engineering Research Centre

Page Replacement Policy

- How can the OS decide which **pages to evict** form the memory?
  - Makes use of a replacement policy
- Goal is to **maximize page access** from memory
  - If we treat physical (main) memory as cache
  - Goal is to **minimize cache misses**
  - In other words, goal is to maximize number of times page is read from/ written to in main memory

Defining a metric

- % of hits to cache = Cache Hit
- % of misses on cache = Cache Miss
- Average Memory Access Time (AMAT)
  - $AMAT = (Hit\% * T_M) + (Miss\% * T_D)$
  - T_M - Cost of accessing Memory
  - T_D - Cost of accessing Disk

A Small Illustration

- Consider a tiny address space of 4 KB with 16 pages
  - Page size: 256 bytes; VPN: 4 bits and offset: 8 bits
  - Assume that every page except Page 3 are in virtual memory
  - Sequence: hit, hit, hit, miss, hit, hit, hit, hit, hit, hit
  - Hit% = 90, 10% mis rate
  - AMAT = 0.9 * 100 (nano seconds) + 0.1*10 (microseconds) ~ **1 ms**
  - **What if hit rate is 99.9%?**
  - **The cost of disk access can be so high - A single miss is very costly!!**

Optimal Replacement Policy

- Developed by Belady, many years ago, also known as MIN
- Simple approach - replace the page that will be accessed farthest in future
- If some page needs to be evicted
  - Evict the page that is needed farthest from now
  - Pages in the cache are more important now than the pages that will be access farthest in future
- Known as the optimal policy - **not practical!**

Optimal Replacement Policy

- Consider a stream of Virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1 cache size: 3

Access	Hit/Miss?	Evict	Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	**2**	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	**3 (0 or 3)**	0, 1, 2
1	Hit		0, 1, 2

Hits: 6/11

Optimal Replacement Policy

- 2 is replaced first in the example as it is required least in future compared to others
- The first three access are misses - Cache is in empty state
  - This is called cold-start miss or compulsory miss
- Optimal is like an ideal policy for any set of access
- Very difficult to implement as in reality future is not known apriori!
- What can be a good alternative?

FIFO Policy

First in First Out

- Earlier systems avoided the complexity to implement the optimal policy
- Simple first in first out approach
- Pages are placed in a queue
  - When need for eviction: Evict the pages on the top
- Advantage: Very easy to implement
- At any point, evict the one that came first!

FIFO Policy

- Consider access to a stream of Virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1, cache size: 3 pages

Access	Hit/Miss?	Evict	Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0, 1, 2
0	Hit		-> 0, 1, 2
1	Hit		-> 0, 1, 2
3	Miss	**0**	-> 1, 2, 3
0	Miss	**1**	-> 2, 3, 0
3	Hit		-> 2, 3, 0
1	Miss	**2**	-> 3, 0, 1
2	Miss	**3**	-> 0, 1, 2
1	Hit		-> 0, 1, 2

Hits: 4/11

Needs Improvement !!

Belady's Anomaly

- Consider a stream 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, cache size: 3

Access	Hit/Miss?	Evict	Cache State
1	Miss		1
2	Miss		1, 2
3	Miss		1, 2, 3
4	Miss	1	2, 3, 4
1	Miss	2	3, 4, 1
2	Miss	**3**	4, 1, 2
5	Miss	4	1, 2, 5
1	Hit		1, 2, 5
2	Hit		1, 2, 5
3	Miss	**1**	2, 5,3
4	Miss	2	5, 3, 4
5	Hit		5, 3, 4

Hits: 3/12

What if Cache size is 4?

Belady's Anomaly

- Consider a stream 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5, cache size: 4

Access	Hit/Miss?	Evict		Cache State
1	Miss			1
2	Miss			1, 2
3	Miss			1, 2, 3
4	Miss			1, 2, 3, 4
1	**Hit**			1, 2, 3, 4
2	**Hit**			1, 2, 3, 4
5	Miss	1		2, 3, 4, 5
1	Miss	2		3, 4, 5, 1
2	Miss	3		4, 5, 1, 2
3	Miss	**4**		5, 1, 2, 3
4	Miss	5		1, 2, 3, 4
5	Miss	1		2, 3, 4, 5

Hits: 2/12

Something strange?

Random Policy

- Picks a random page to replace under memory pressure
- Has properties similar to FIFO - very easy to implement
- Random totally depends on luck to get it right
- Need to run multiple times to get good approximation
- May perform a touch better than FIFO but little less than optimal
- How does that work?

Random Policy

- Consider access to a stream of Virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1, cache size: 3 pages

Access	Hit/Miss?	Evict	Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	**1**	0, 2, 3
0	Hit		0, 2, 3
3	Hit		0, 2, 3
1	Miss	**2**	0, 3, 1
2	Miss	**0**	3, 1, 2
1	Hit		3, 1, 2

Hits: 5/11

Least Recently Used (LRU)

- As done in scheduling can we use history as a guide?
- **Idea:** If a page was referenced recently, it may be likely to be referenced again
- The historical information that page replacement can use: **Frequency**
- The more recently a page has been accessed, it should not be replaced soon
- **LRU:** replaces the least recently used page
- Works well due to locality of references (Temporal locality here)

Least Recently Used

- Consider access to a stream of Virtual pages: 0, 1, 2, 0, 1, 3, 0, 3, 1, 2, 1, cache size: 3 pages

Access	Hit/Miss?	Evict	Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		1, 2, 0
1	Hit		LRU -> 2, 0, 1
3	Miss	**2**	LRU -> 0, 1, 3
0	Hit		LRU -> 1, 3, 0
3	Hit		LRU -> 1, 0, 3
1	Hit		LRU -> 0, 3, 1
2	Miss	**0**	LRU -> 3, 1, 2
1	Hit		LRU -> 3, 2, 1

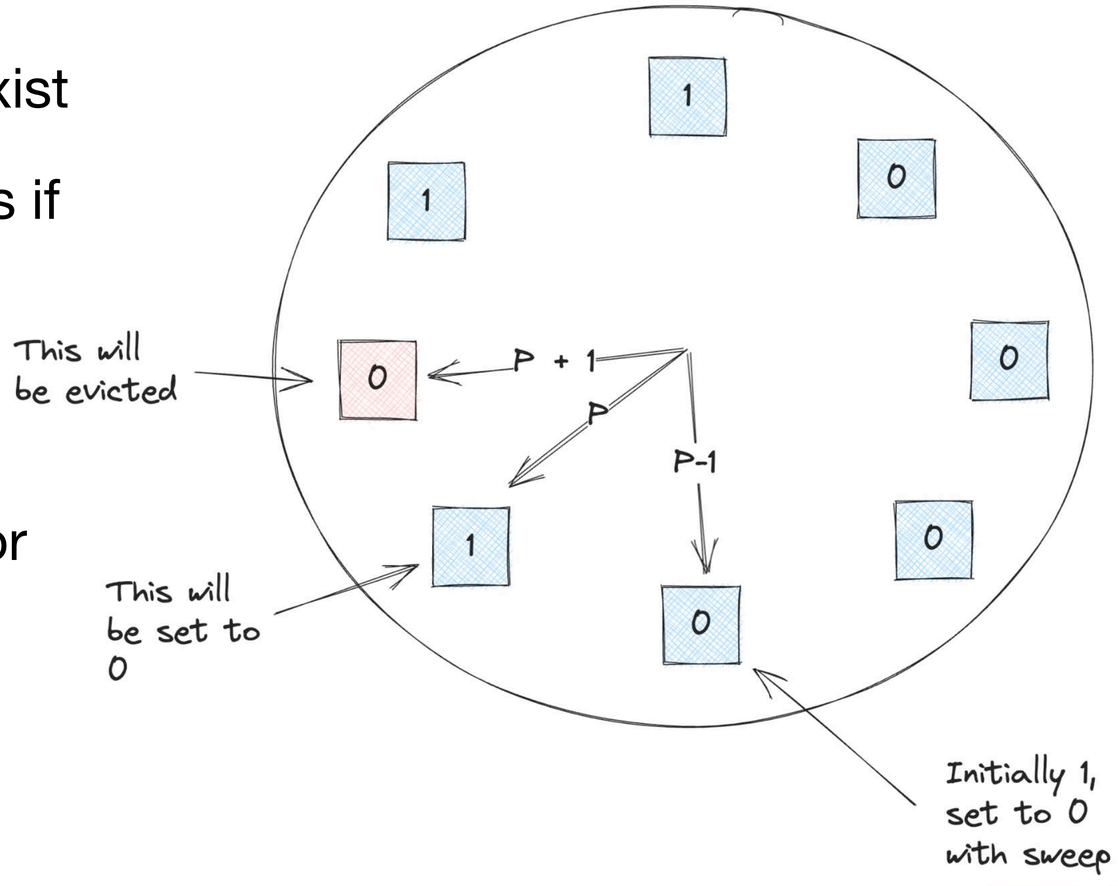
Hits: 6/11

Implementing LRU

- How to implement policies like LRU?
- OS is not involved in every memory access - How does it know which page is the least recently used?
- Hardware can help along with some approximations
- When a page is accessed, MMU can set the “accessed” bit to 1 in PTE
  - It is the responsibility of OS to clear the bit
- How can the “accessed” bit be used by the OS to implement LRU?

Implementing LRU

- Simple and early approach, many approaches exist
- When replacement needs to be done, OS checks if a page, P pointed to has use bit 1 or 0
  - If use bit is 1, set that to 0 and go to the next page
  - If use bit is 0, that page becomes the victim for eviction
  - Avoid repetitive scanning, not the optimal, still works better
- Improved version makes use of **dirty bit**

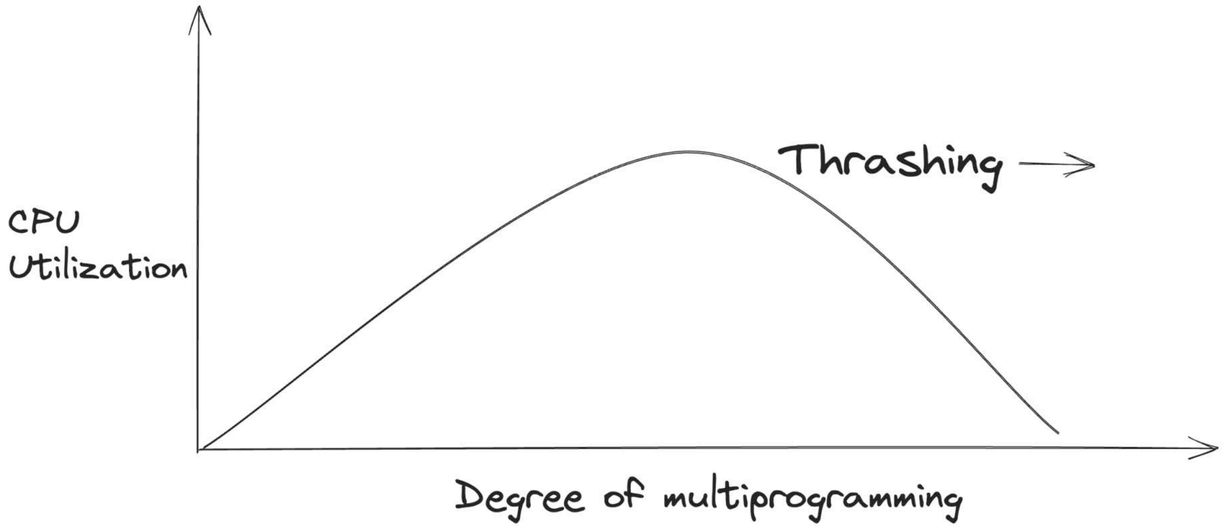


Implementing LRU

- If a page has been modified
  - It has to be pushed to disk and write to disk needs to be done
  - If not, if page is clean, it can simply be replaced - overhead is less!
- Make use of a new bit - modified or dirty bit in the PTE
  - If a page has been recently modified -> dirty bit is set to 1
  - If the page is clean and not modified -> dirty bit is set to 0
- Clock algorithm first scans for pages that are both unused and clean
- If no page with both unused and clean status, evict unused and dirty pages

Thrashing

- What to do if the memory is oversubscribed?
  - Memory demands of running processes exceeds available physical memory
  - System will be constantly paging - Thrashing
  - One approach is admission control
  - Try to run only a subset of processes instead of trying to accommodate everything
  - More aggressive approach: Kill memory intensive processes



SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Networking - Application Layer

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:

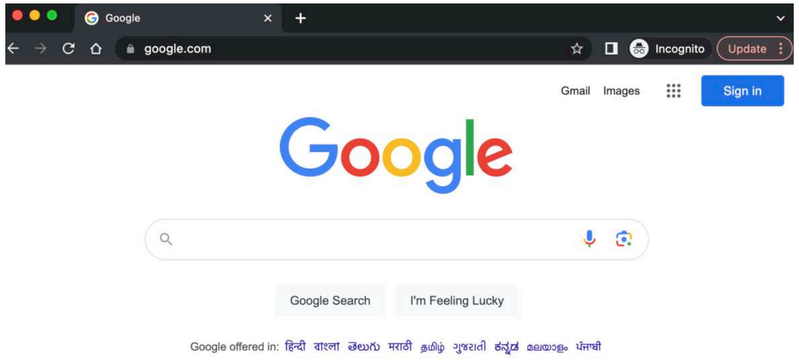
- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Instructor Materials on Computer Networks: A Top Down Approach, Kurose and Ross

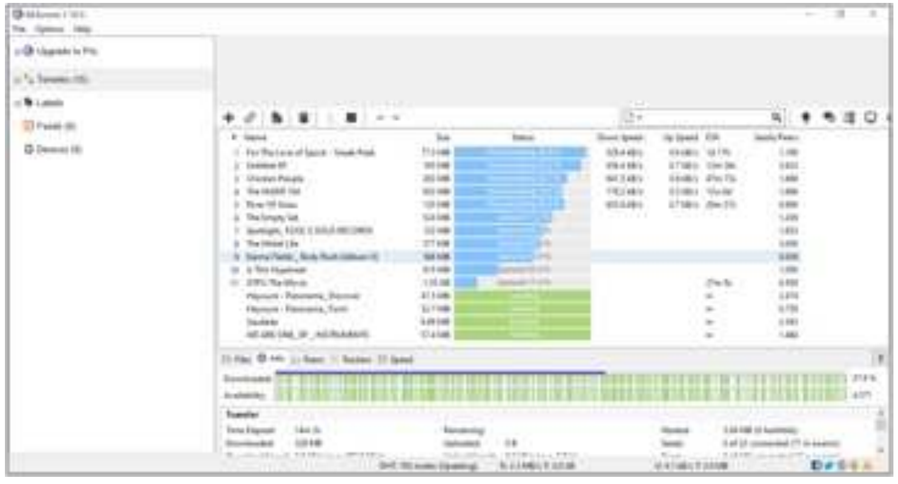
What are the different Network Applications that you come across?

- Streaming stored video (eg: Netflix, Youtube)
- VOIP apps (eg: Skype)
- Web browser (eg: Chrome, Firefox)
- Social Networking (eg: Facebook, Instagram)
- Instant Messaging (eg: Whatsapp)
- P2P File Sharing (eg: Bit Torrent)
- Real-time Video Conferencing (eg: Zoom)
- ….

Many Processes run on the OS

Rather applications!





Browser - Type any URL and you get the page - How?

The process on the other side needs to provide the page

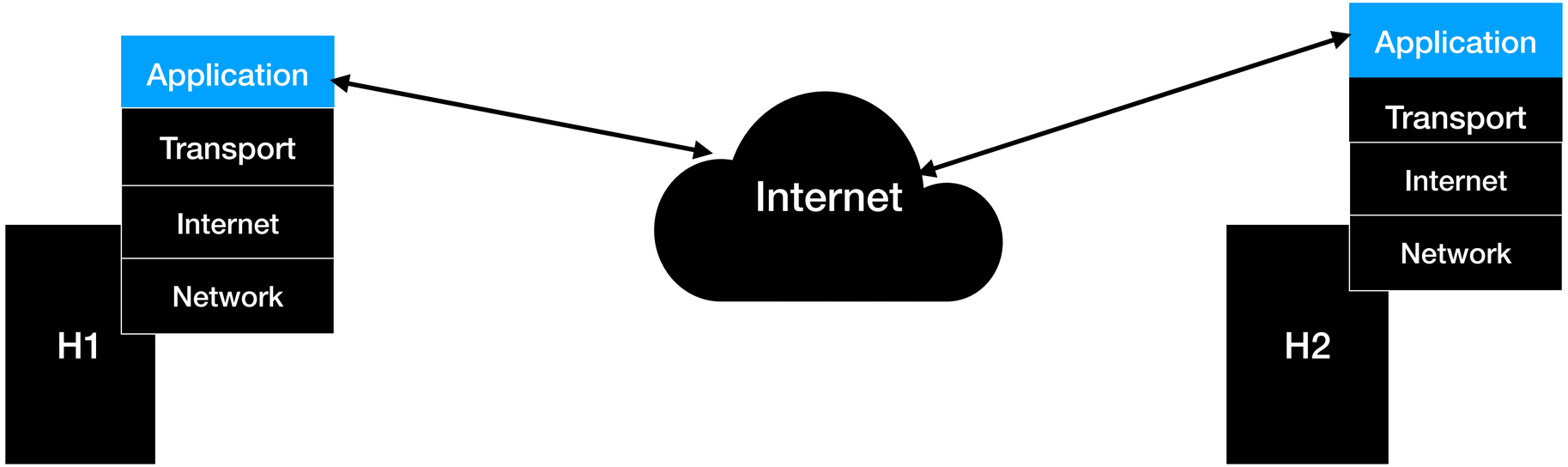
Large videos may have to be served to different users

There are also programs like torrent -

How does that work?

Is transport layer enough?

- Process communicates over the network
- The real communication happens through the application layer



Building a Network Application

- **Write programs:**
  - That can run on different systems
  - Communicate over the network
  - Eg: Web server communicates with browser (Apache)
- **Core-network device software:**
  - Network devices do not run user applications

Network Application Architectures

- From a developer perspective, network architecture is fixed
  - Application architecture is something that can be controlled
- Two main types are available: **Client-server** and **P2P**
- **Client-Server**
  - Host that is always on, serving the clients - Server
  - Host that requests for services - Clients
  - Eg: Web client and Web server

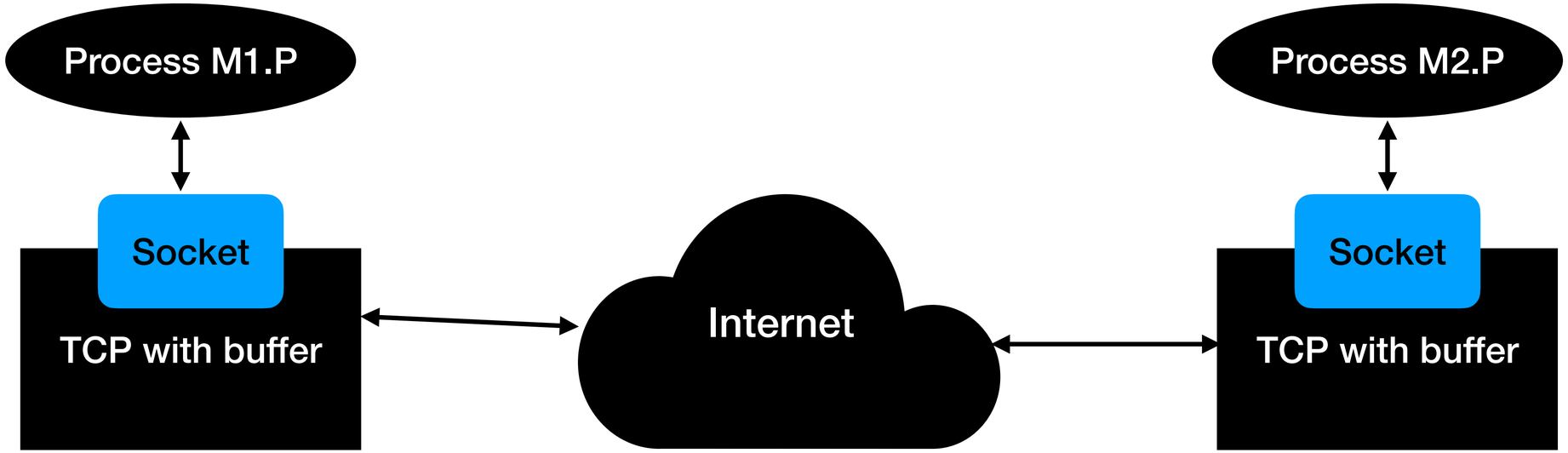
Network Application Architecture

- Server has mostly a fixed IP address (or rather domain)
- Clients can always connect by sending packet to server IP address
  - Eg: Web browser, FTP, e-mail
- Often a single server may not be enough - leverage data centres
- Data centres have hundreds or thousands of servers that must be processed and maintained - **Energy!!**
  - **Spare a thought for the carbon foot print - Can we do something?**

Network Application Architecture

- **Peer-to-Peer (P2P) Architecture**
  - There is minimal or no reliance on dedicated servers (No always on Servers)
  - Peers communicate among each other
  - Peer technically acts as a client and a server
  - Not owned by any service provider and does not pass through dedicated server
  - Advantage: Self-scalability, cost effective
  - Challenges: Security, Performance, reliability, etc.
  - Eg: Bit Torrent, Skype (earlier versions where fully using p2p)

Lets take a step back



- Process sends to and receives messages from network using socket interface
- The developer has more control on the application side than socket
- Application just sends message with IP address and port - Rest other layers

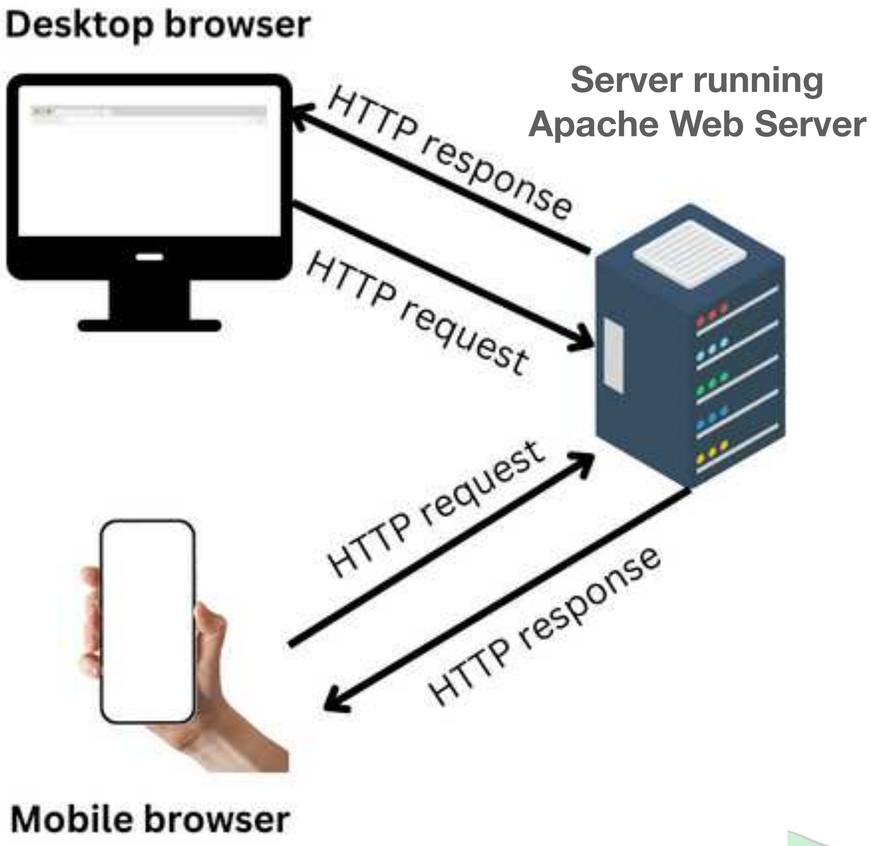
Application Layer Protocol

What does it mean?

- Application layer protocol defines the following:
  - Types of message exchanges (request/response)
  - Syntax of various message types
  - Semantics of the fields
  - When and how the process sends and responds to messages
- Some protocols: HTTP, SMTP, DNS, etc.

HTTP: Hyper Text Transfer Protocol

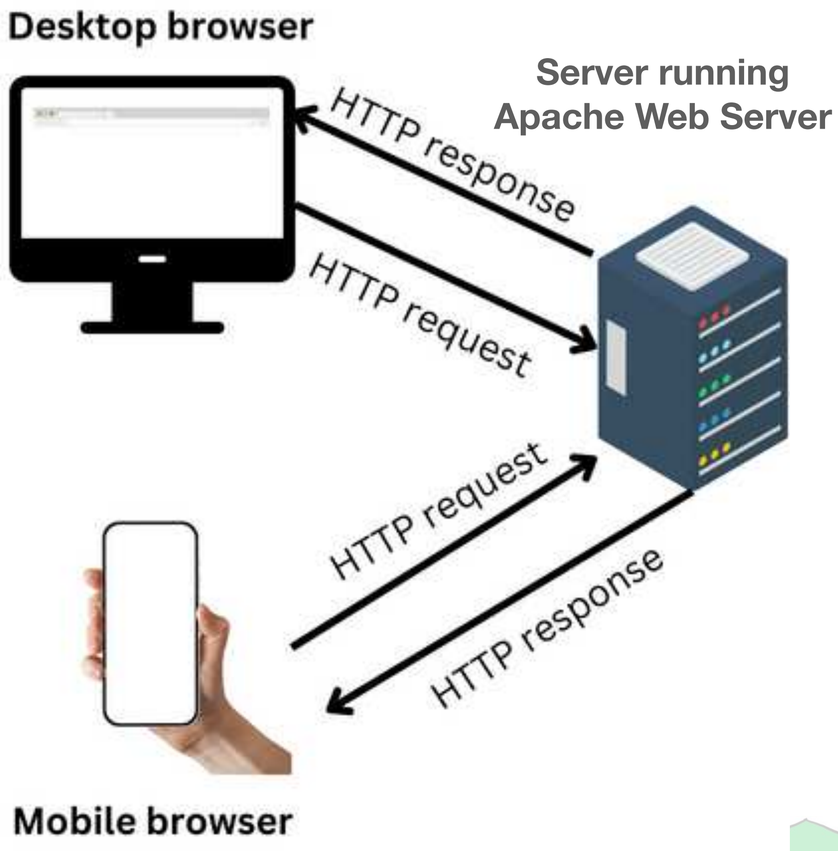
- Application layer protocol of the web
- Implemented in two programs: Client and Server
- HTTP protocol defines structure of messages
- **Client:** browser that sends requests, receives and displays web objects (using HTTP protocol)
- **Server:** Web server that sends objects in response to requests (using HTTP protocol)



https://iiit.ac.in/samplePage.html
Host name: iiii.ac.in
Object: samplePage.html

HTTP: Hyper Text Transfer Protocol

- HTTP uses **TCP at transport layer**
  - Client initiates TCP to server (Port 80)
  - Server accepts TCP connection from client
  - HTTP messages are exchanges
  - Connection is closed
  - HTTP is reliable - why?
- HTTP is **stateless**
  - Server maintains no information about client



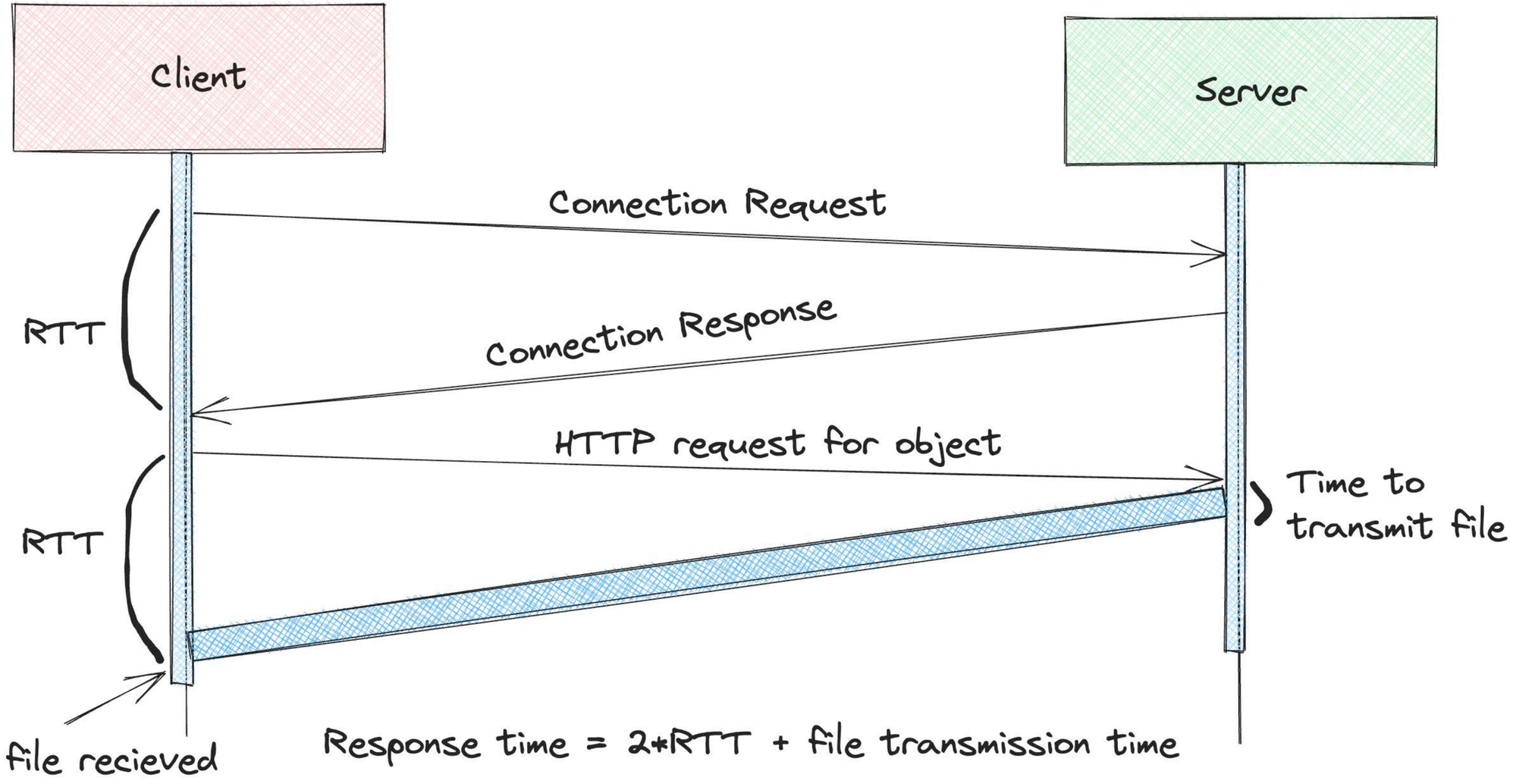
Types of HTTP Connection

Persistent and Non-Persistent Connections

- **Non-persistent HTTP Connection (HTTP/1.0)**
  - For every connection, the client has to create a request (one page may require multiple objects)
  - Downloading multiple objects requires multiple connections
  - Opened connection is closed after each request response
- **Persistent HTTP Connection (HTTP/1.1)**
  - One connection for all the objects
  - The opened connection is maintained

Non Persistent Connection: Response time

HTTP 1.0



Response time = 2*RTT + file transmission time

HTTP Request Message

- Two types of HTTP messages: **request** and **response**
- HTTP request message: ASCII (human-readable format)
- HTTP supports different methods - GET, POST, PUT, HEAD, DELETE
- Request line: Method, URL and Version

Request line

Header lines

```
GET /index.html HTTP/1.1\r\n
Host: www.iiit.ac.in/centers \r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
   10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Response Message

Status line

Header lines

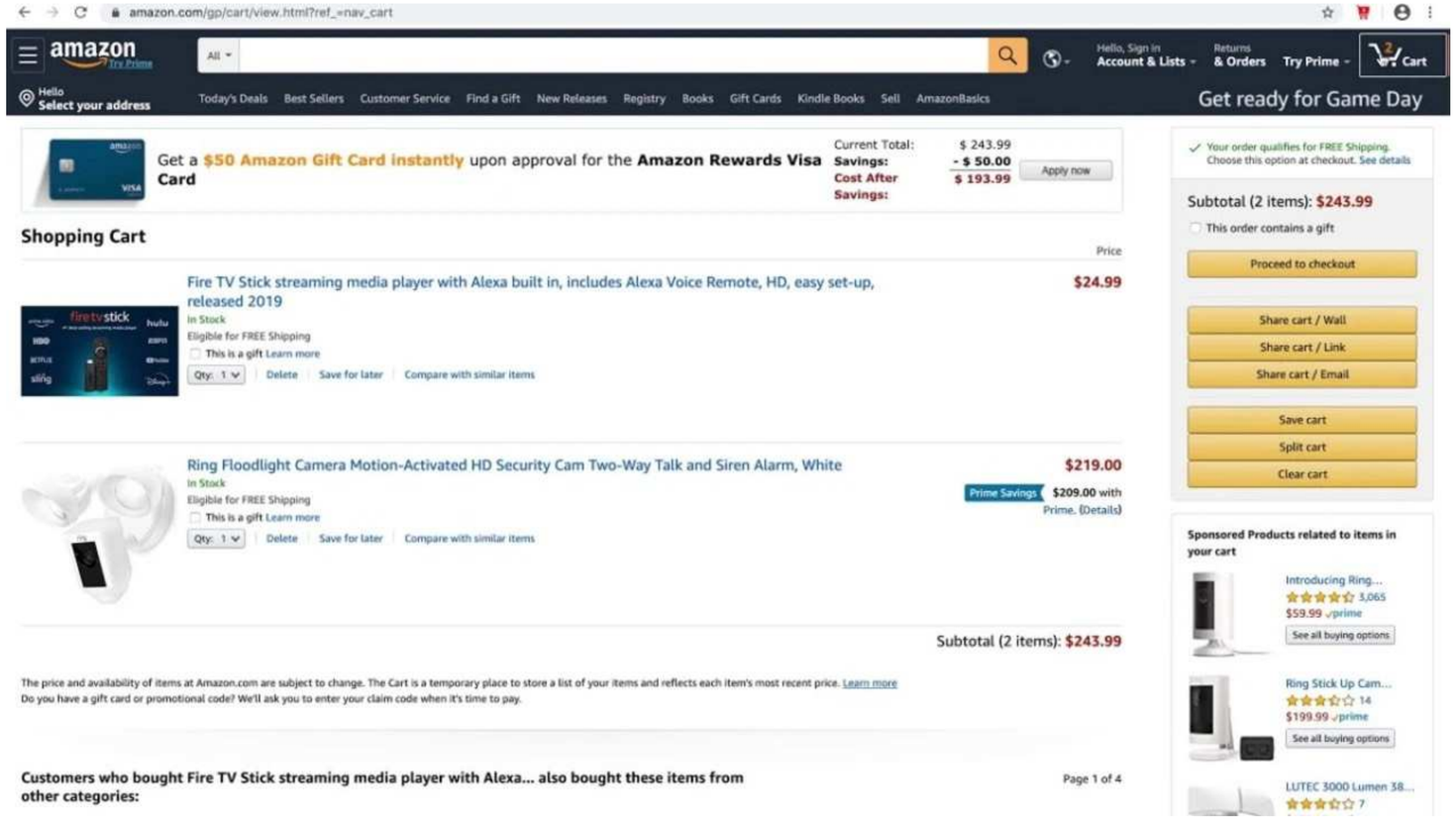
```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2020 00:53:20 GMT
Server: Apache/2.4.6 (CentOS)
   OpenSSL/1.0.2k-fips PHP/7.4.9
   mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Tue, 01 Mar 2016 18:57:50 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 2651
Content-Type: text/html; charset=UTF-8
\r\n
data data data data data ...
```

- Status line: version and status code
- Last-modified in header can help - **How?**

HTTP Status Codes

Status Code	Description
200	Request succeeded, status ok
301	Moved permanently, requested object moved, new location in the location: field (header)
400	Bad request, request message not understood by server
404	Document does not exist
505	HTTP version not supported

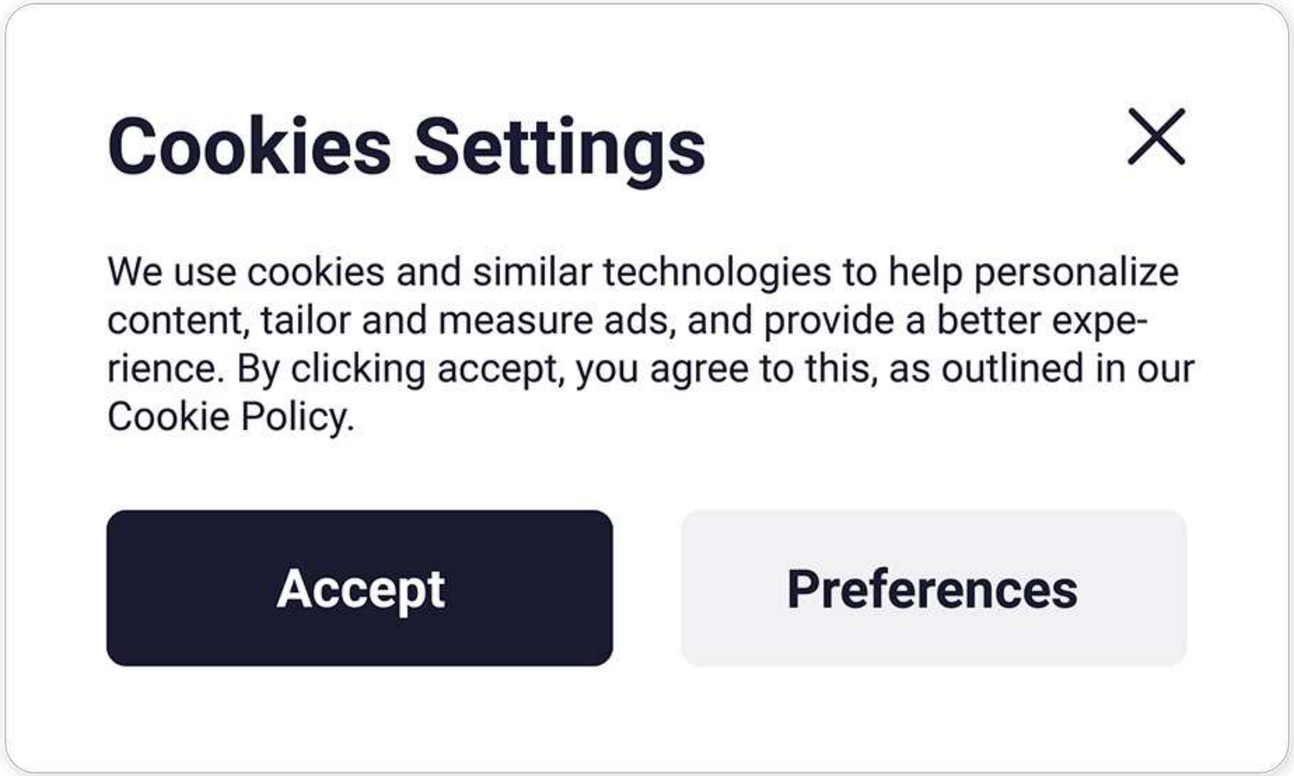
Can this happen in stateless protocol



- Even after closing and opening the website, the cart has the items
- How's website like Amazon, ebay, etc able to do this?

Image source: https://en.softonic.com/articles/how-to-use-share-a-cart-for-amazon

Why not cookies!



Source: https://boom visibility.com

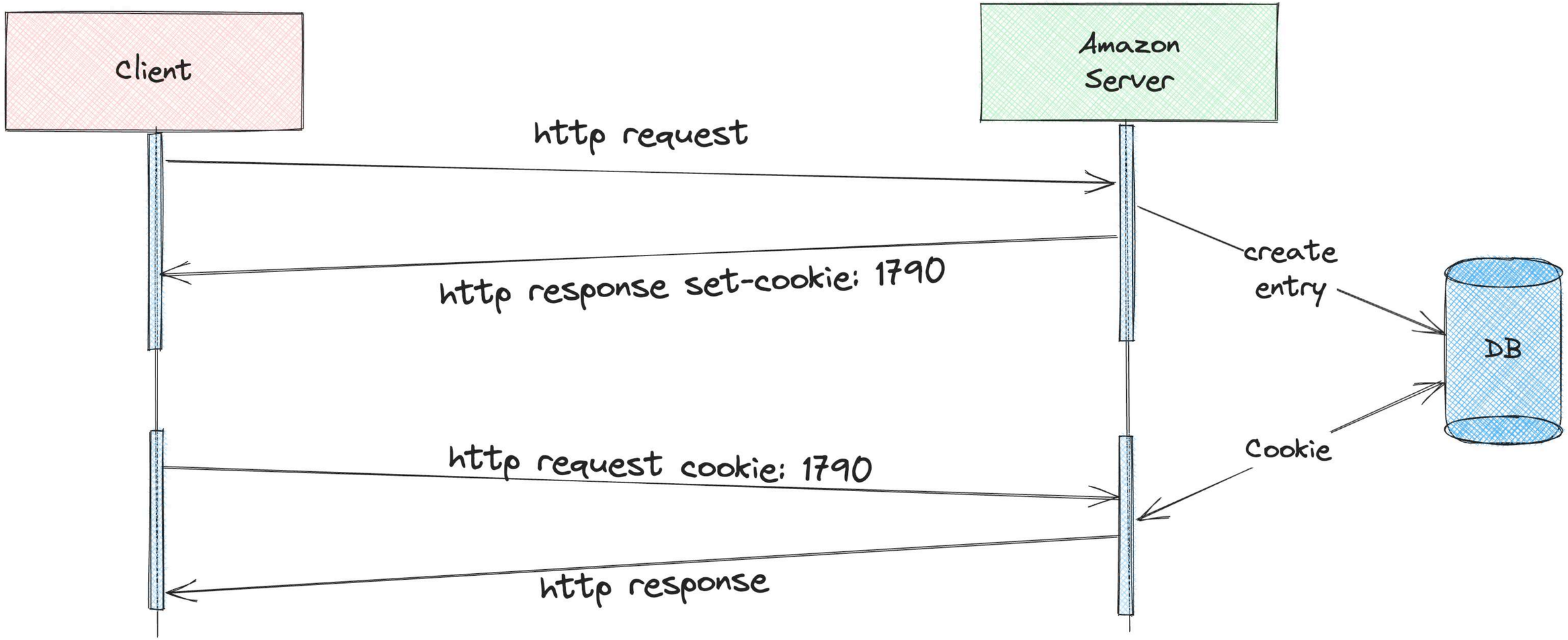
Maintaining User/Server State: Cookies

- HTTP server is **stateless**
- Helps in supporting thousands of simultaneous connections
- Each connection is treated separately
- Website may want to identify users for various reasons
  - Keep session information
  - Recommend similar products

Cookies

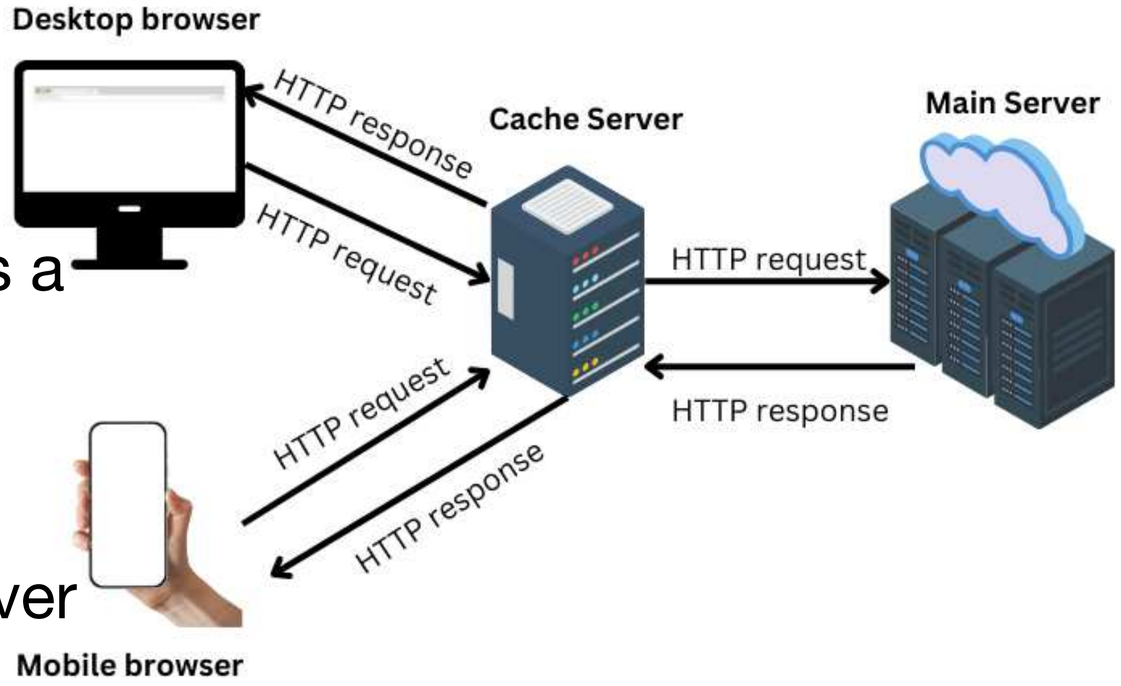
- HTTP header consists of information for cookies
- Consists of four components:
  - Cookie header line in HTTP response
  - Cookie header line in HTTP request message
  - Cookie file kept in clients system
  - Backend database on the server/website
  - Cookies can be used to create user session on top of HTTP - **Can be invasion of Privacy!**

Cookie Illustration



Web Caches

- Not every time we need to access the main (original) web server
- We can have proxy server that satisfies request on behalf of main server
- Browser can be controlled to point towards a cache (mentioned in response header)
  - If cache hit: return object from cache
  - Else cache request object from main server and returns it

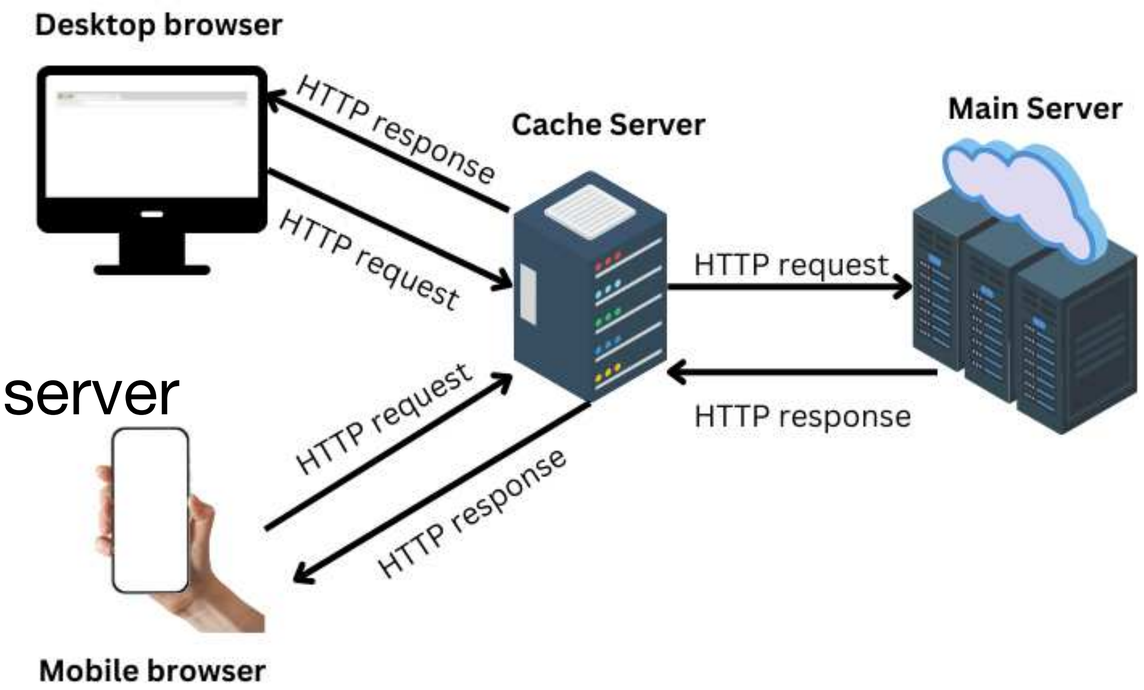



```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```

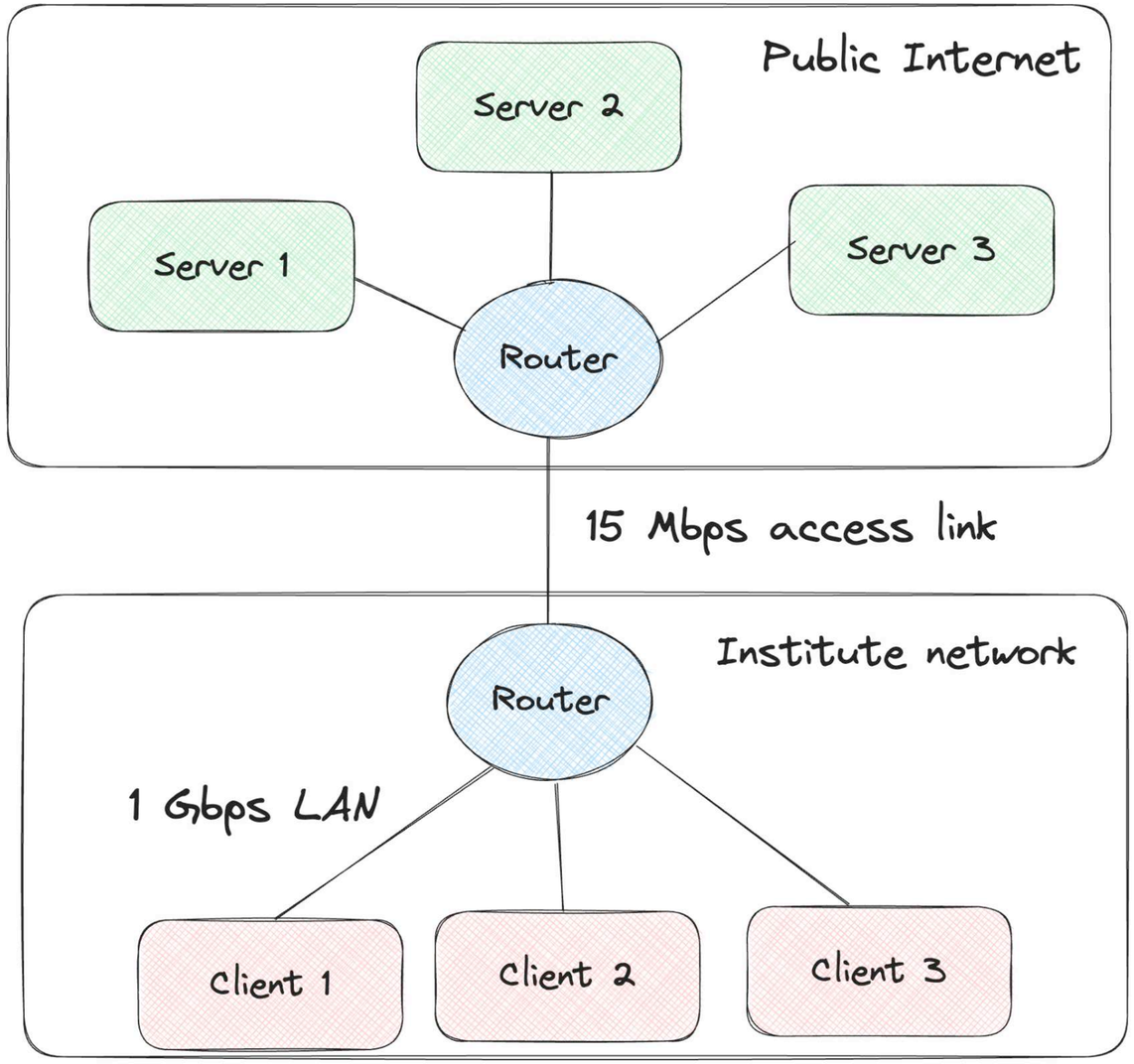
Caching Benefits

- Reduce response time for a client request
  - Cache is closer to client
  - There may be bandwidth constraints
- Reduce traffic on the internet and to main server
  - Significantly improve performance
- Internet is dense with caches
  - Effective delivery of content



University Scenario

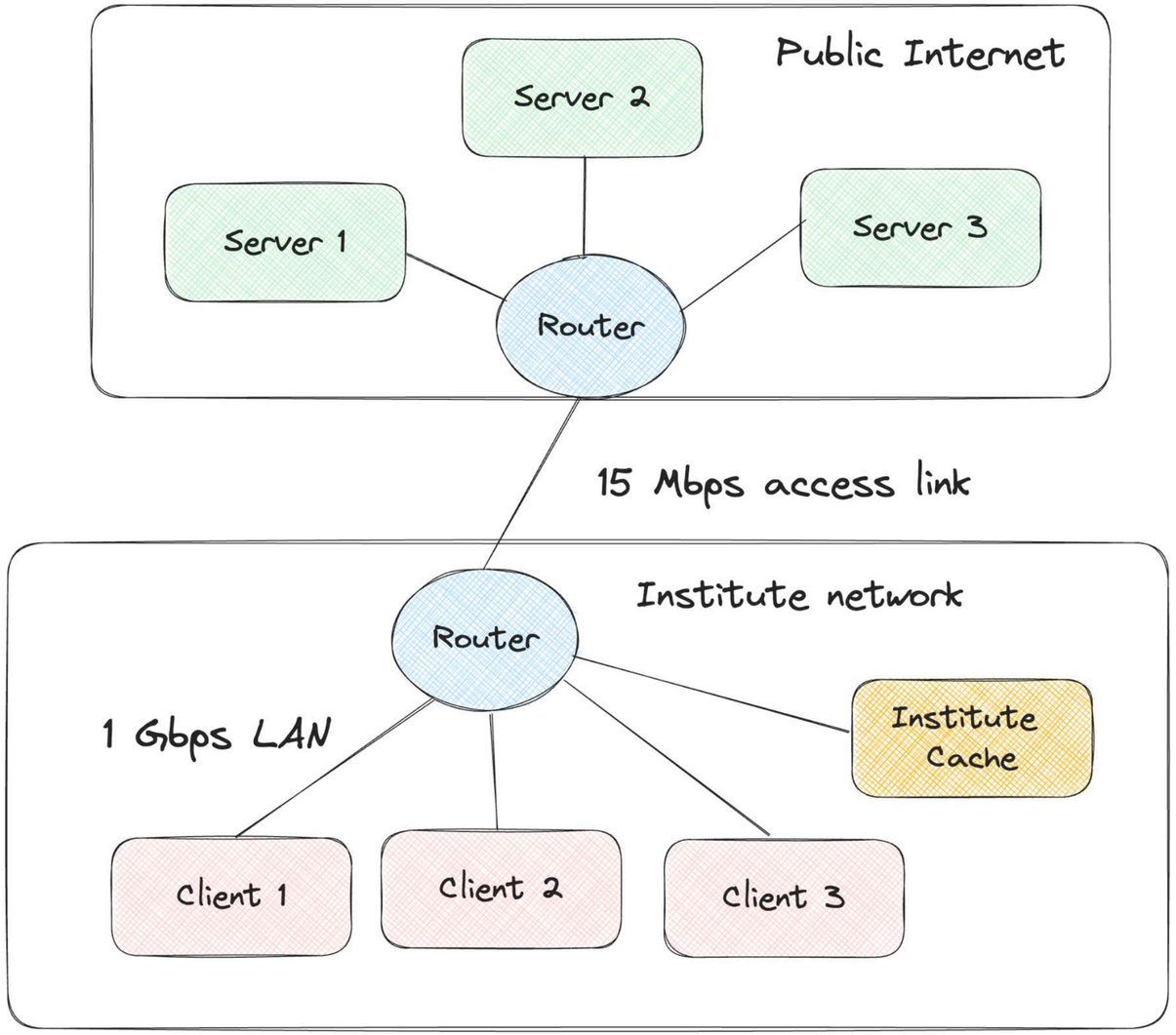
- Assume that:
  - servers to router in public internet incurs ~ 2 sec delay
  - Data is 1Mbits
  - Request interval from client ~ 15 requests/ second
- End-to-end delay = Internet delay + access link delay + LAN delay
  - ~ 2 sec + ~ mins + ~ u sec (High!!)



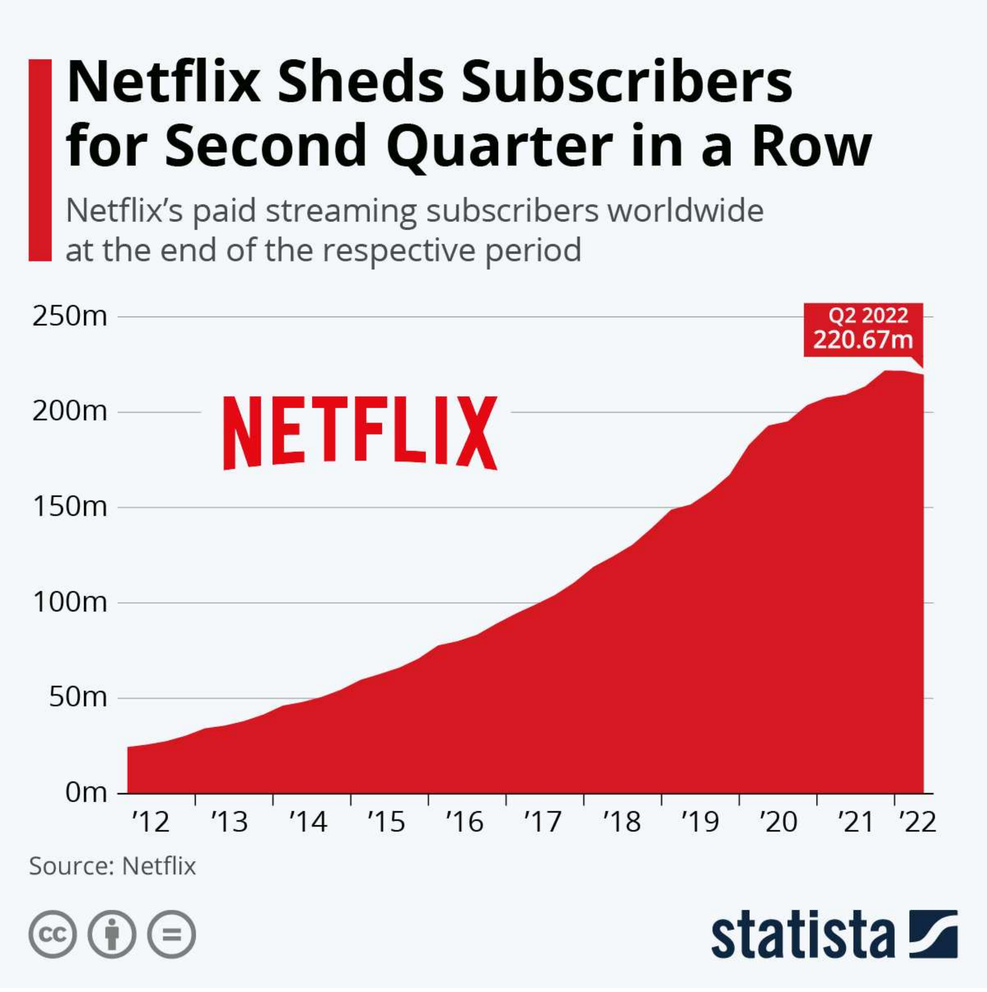
University Scenario

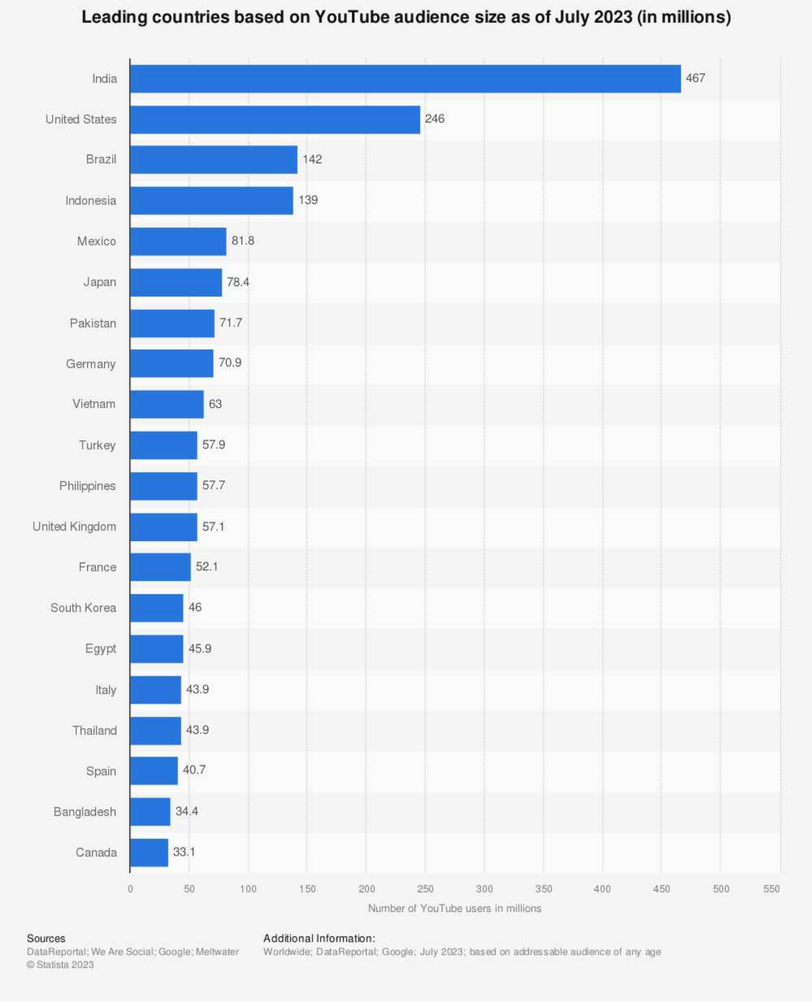
- Easy way out is to increase bandwidth
  - Costly solution, request rate may increase
- Introduce Cache server at Institute level
  - Even if there is 40% hits => significant improvement
- Delay = 0.6*(2) + (0.4)*(0.01)

  = ~1.2 secs
- Cheaper solution, lower end-to-end delay



Web Cache has broader Applications





Content Distribution Networks (CDN)

- Many internet companies are distributing on-demand video streams to millions of users on daily basis
  - Youtube (~2.7 Billion users), Netflix (~238 Million subscribers), ..
  - Distributed across the world - Having one large data centre may not work - Why?
- To support growing demand, scale - distribute over different CDNs
  - Servers are distributed among different geographical locations
  - Requests are redirected to CDNs (private or third party)

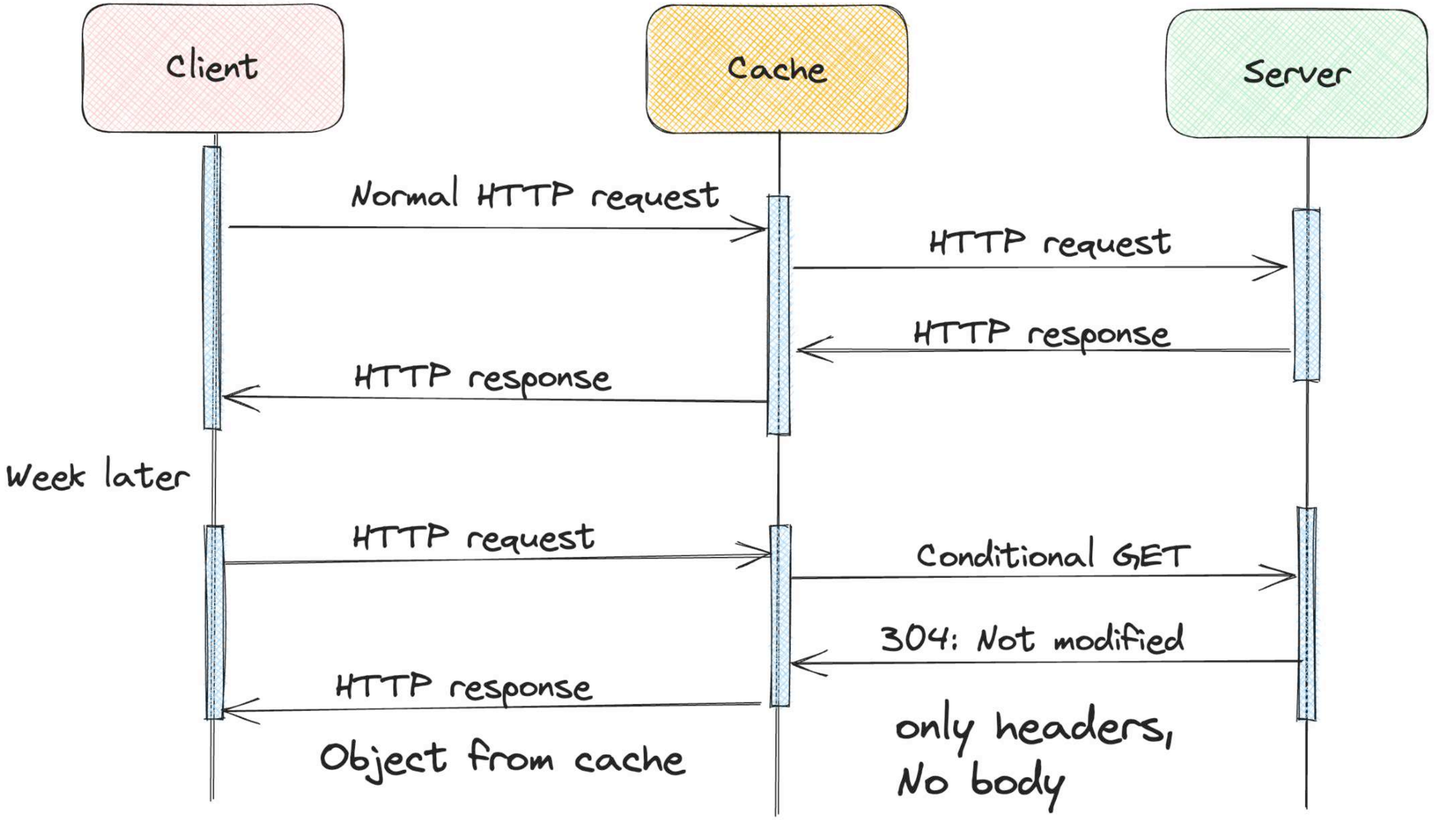
Content Distribution Networks (CDN)

- CDNs adopt two different server placement strategies
  - **Enter Deep:** Deploy server clusters in all access ISPs
    - High maintenance, higher throughput and lower delays
  - **Bring Home:** Building larger cluster at smaller number of sites (eg: 10s)
    - Lower maintenance, lower throughput and higher delays
- Client needs to pick a cluster - Geographically closest or using real-time information (dynamic choices based on heuristics)

Conditional GET

Cache does not contain originals

- If things are kept in cache, the copy of original may become stale!
- HTTP provides mechanism to verify if object is up-to-date (conditional GET)
- HTTP request header - use field “If-modified-since: “
- Server provides “Last-Modified” in the response header



HTTP 1.1 and Beyond

- **HTTP 2.0**
  - Standardised in 2015
  - As of 2020, 40% of top 10 million websites support HTTP/2
  - Primary goals
    - Enable request response multiplexing over single TCP
    - Request prioritisation
    - Server push
    - Compression of HTTP header fields
- **HTTP 3.0** underway (drafts as of 2020)

Domain Name System (DNS)

- Students have roll numbers but addressed by names
  - Similar is the case with citizens - Aadhar, SSN
- What about the internet?
  - Google.com, Facebook.com, YouTube.com - Are they enough?
  - How to locate them? Eg: iiit.ac.in - Provides info that it is in India but exactly where?
  - IP address can help - They have a hierarchical representation

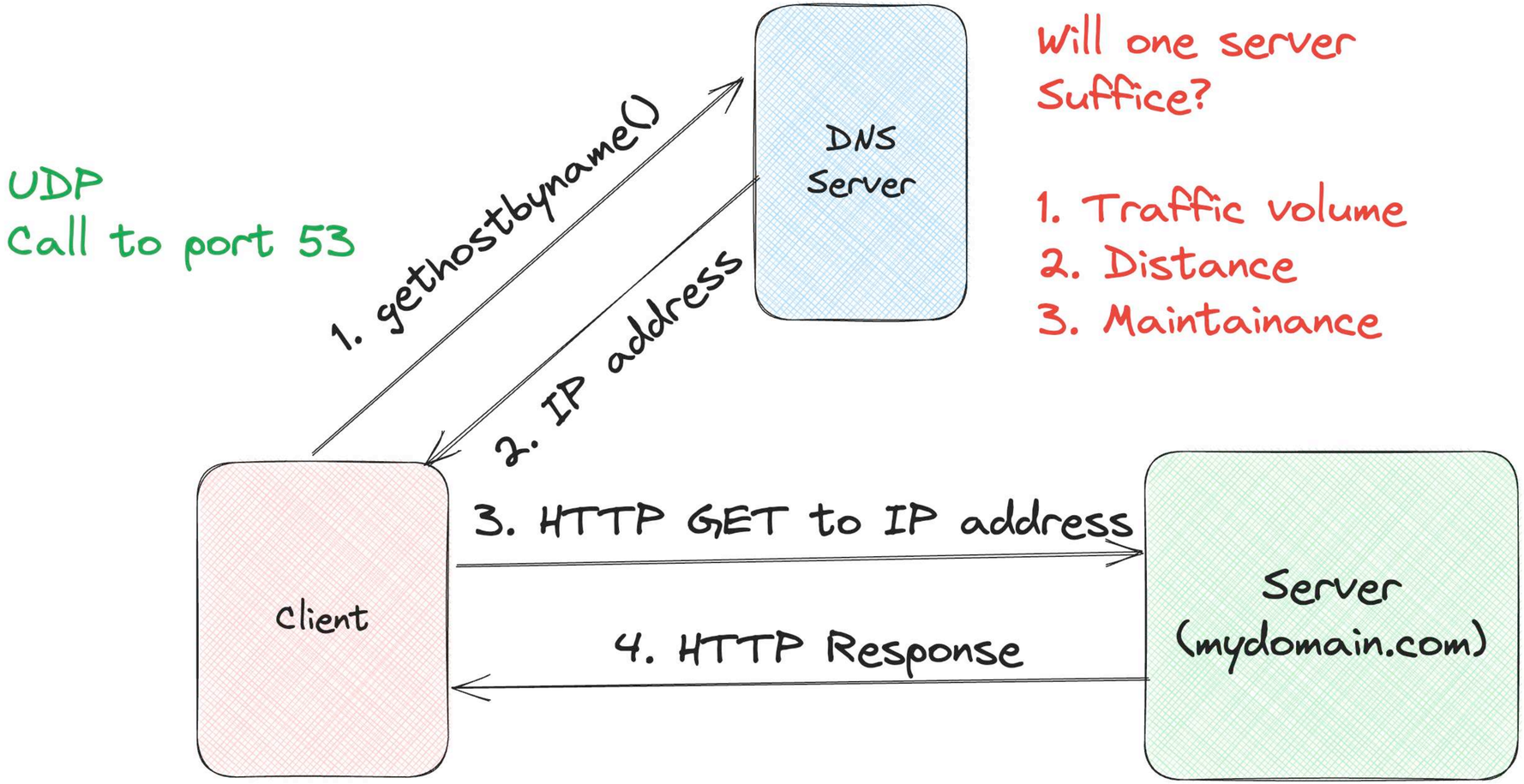
Domain Name System (DNS)

- Directory service of the internet that translates hostnames to IP addresses
- People prefer mnemonic names, routers prefer IP addresses
- Application layer protocol, allows hosts to query a distributed database
- DNS servers are UNIX machines running **Berkley Internet Name Domain** (BIND) software
- DNS runs over **UDP** and uses port **53**
- Leveraged by protocols like HTTP and SMTP to translate hostnames to IP addresses

Domain Name System (DNS) Services

- **Host aliasing**
  - Same hosts can have multiple aliases, resolve the names (get canonical names of host)
  - Eg: 1231242s-us-west.aws.com: mydomain.com, e-markt.com
- **Mail Server Aliasing**
  - Mail servers may also have aliases
  - DNS can provide canonical names of mail server to mail clients
- **Load distribution**
  - Perform distribution among replicated servers
  - Eg: Amazon may be replicated to multiple servers (IP address), keep giving back IP in different order

DNS: How does it work?



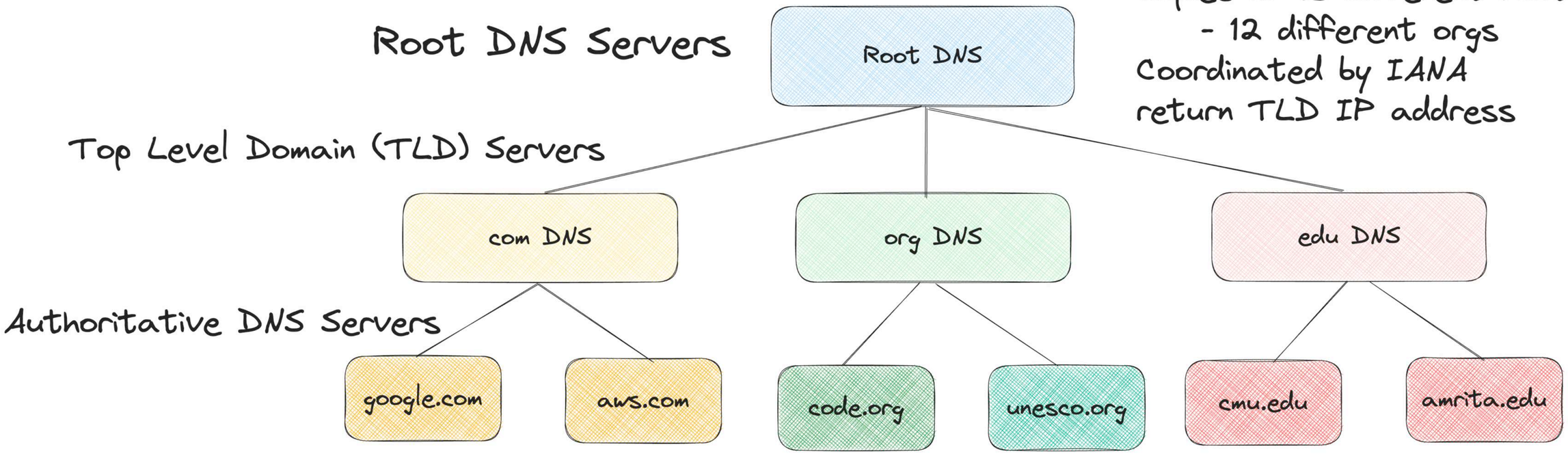
Will one server Suffice?

1. Traffic volume
2. Distance
3. Maintainance

wants to
send HTTP request
to mydomain.com

DNS: Distributed Hierarchical Database

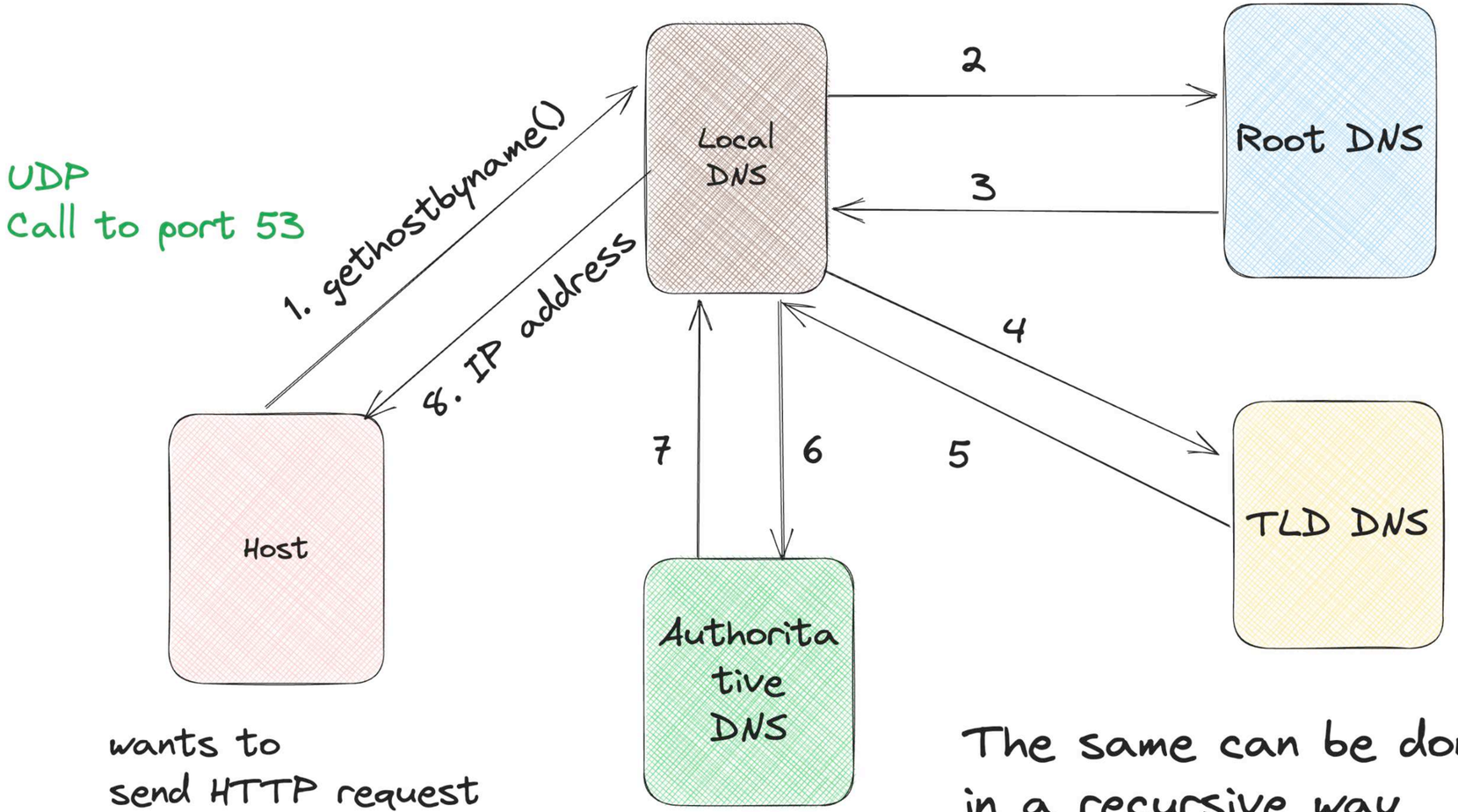
More than 1000 root servers
Copies of 13 different root servers
- 12 different orgs
Coordinated by IANA
return TLD IP address



TLD Servers - can be maintained by orgs, provide IP of authoritative DNS Servers
Authoritative DNS servers - Orgs can choose to implement their own or go for third party
All DNS records have to be made public - that maps hosts to IP address

Local DNS

Each ISP can have DNS and clients can connect to that



wants to
send HTTP request
to mydomain.com

The same can be done
in a recursive way
putting overhead on contacted
server

Caching can be done at the ISP Local DNS

DNS Records

DNS Servers stores Resource Records (RR)

- Each RR is a tuple: **(name, value, type, ttl)**
- **type = A**
  - name is hostname
  - value is IP address
  - (abc.com, 122.x.x.x, A, 3600)
- **type = NS**
  - name is domain
  - value is hostname of authoritative
  - (abc.com, ns.host.com, NS, 86400)
- **type = CNAME**
  - name is alias of canonical name
  - value is canonical name
  - (abc.com, west-abc.com, CNAME, 86400)
- **type = MX**
  - name is domain
  - value is name of SMTP mail server
  - (abc.com, mail.abc.com, MX, 3600)

More questions needs to be answered

How does Network layer route the traffic?

Before that: Its not that only one process will be running at one time?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Concurrency - Introduction

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

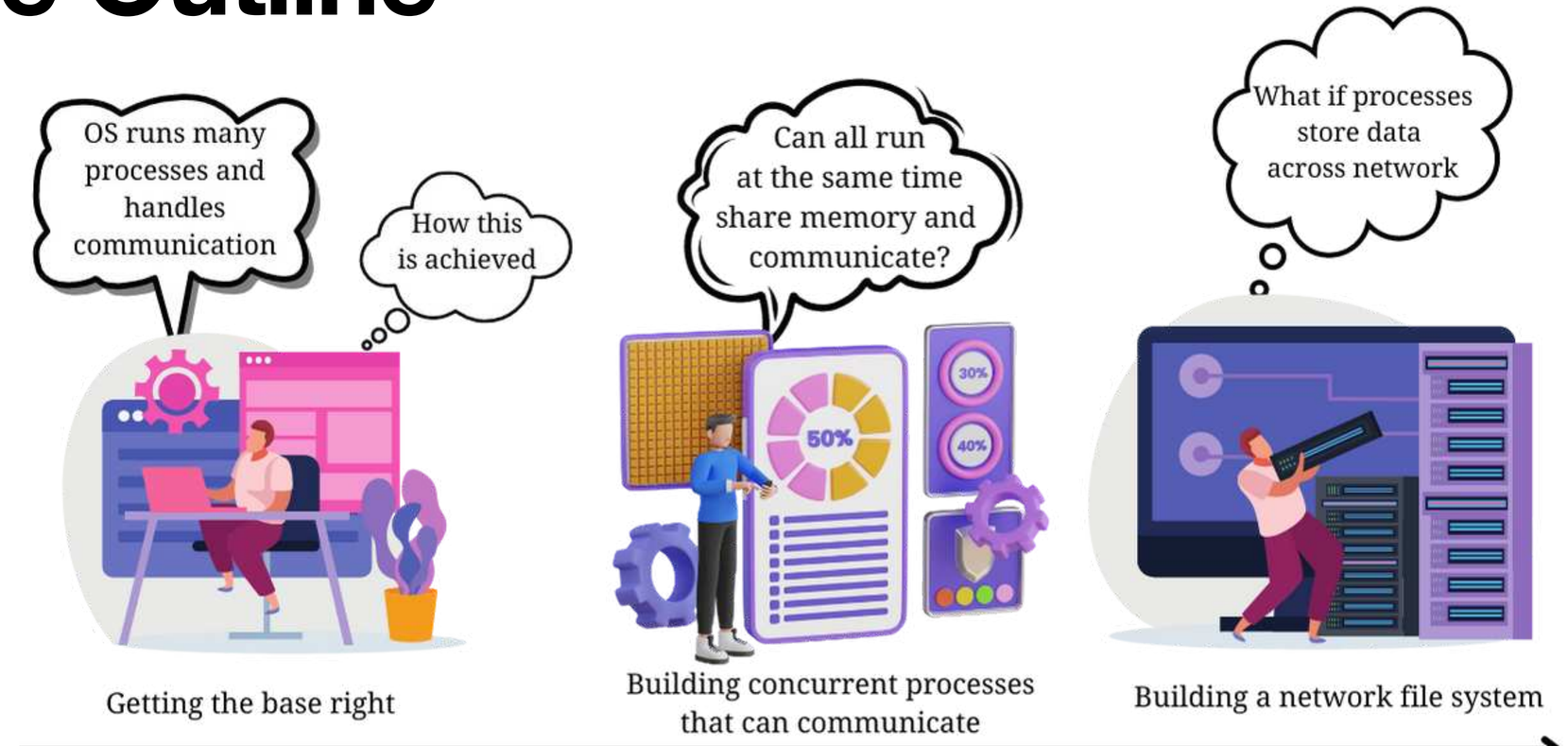
Acknowledgement

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:

- Operating Systems in three easy pieces by Remzi et al.

Course Outline



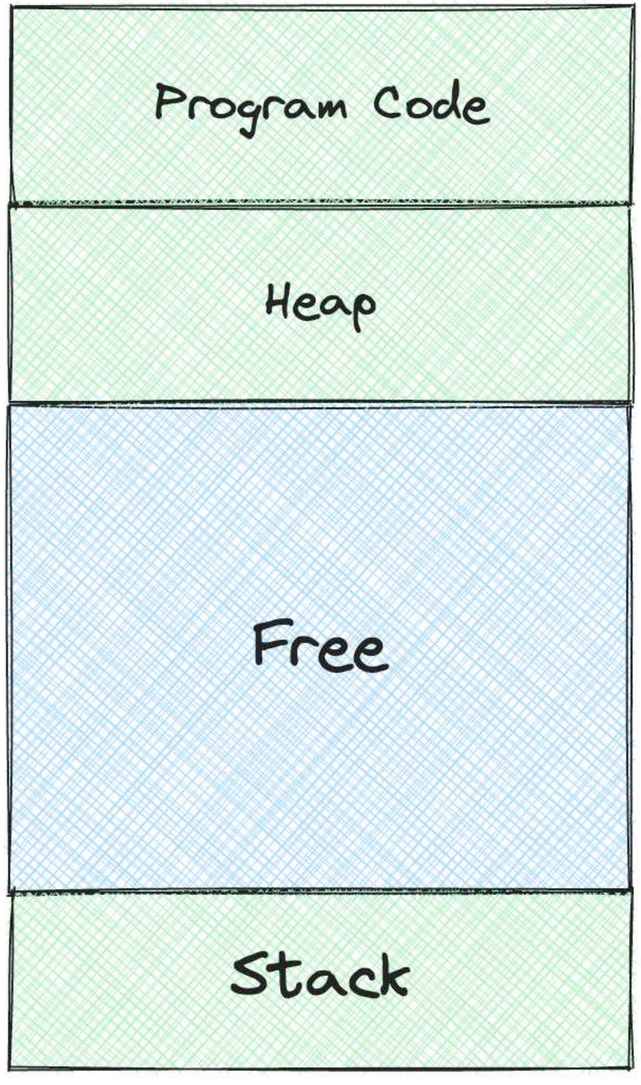
This Course

Process and Memory Virtualization	Concurrency	Persistence
Networking intro	Addressing and Routing	Network file Systems

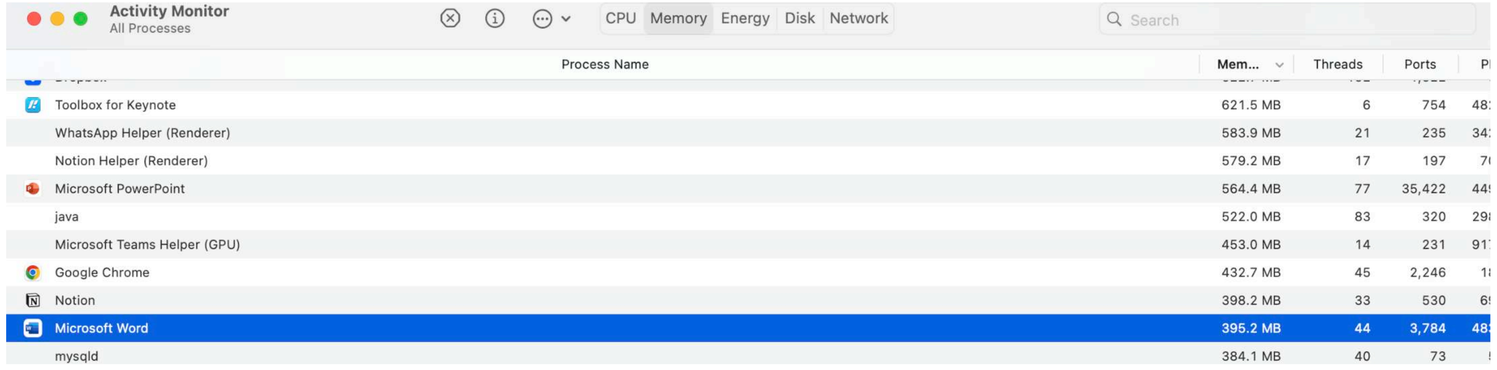
The Type of Process we have seen so far!

Some Recap

- Process during execution
  - Program Counter (PC): Points to the current instruction that is being run
  - Stack Pointer (SP): Points to the current frame of the function call
- What about the memory? - Paging!
- This is a single thread execution
- But in reality process is more than a single thread of execution



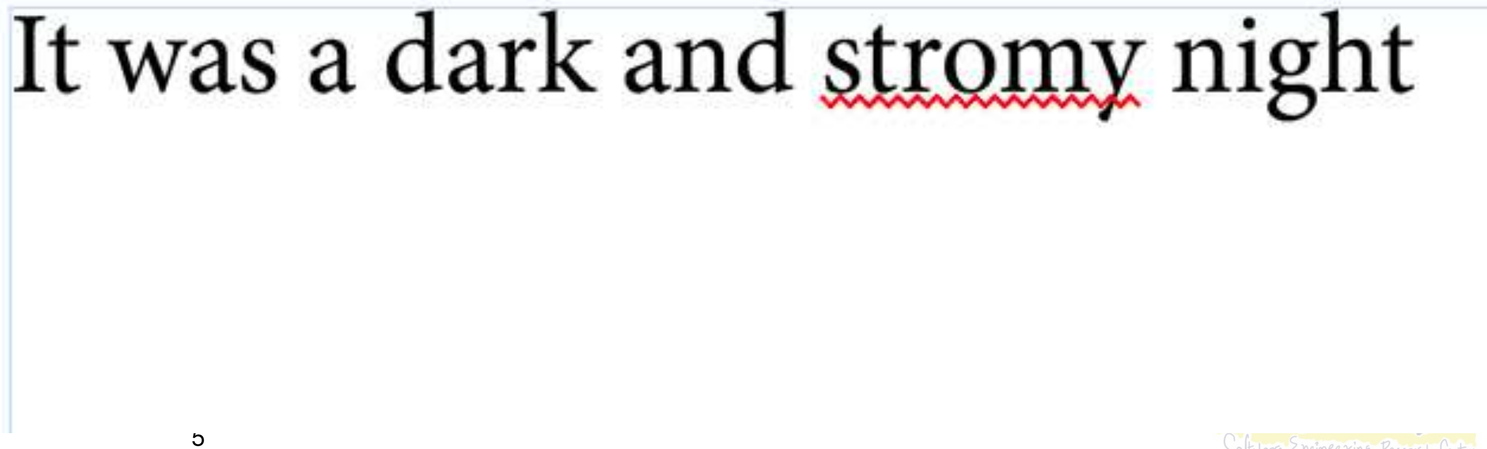
In reality a process does more things!



Check the processes running in your OS

Microsoft Word is a process, while using it:

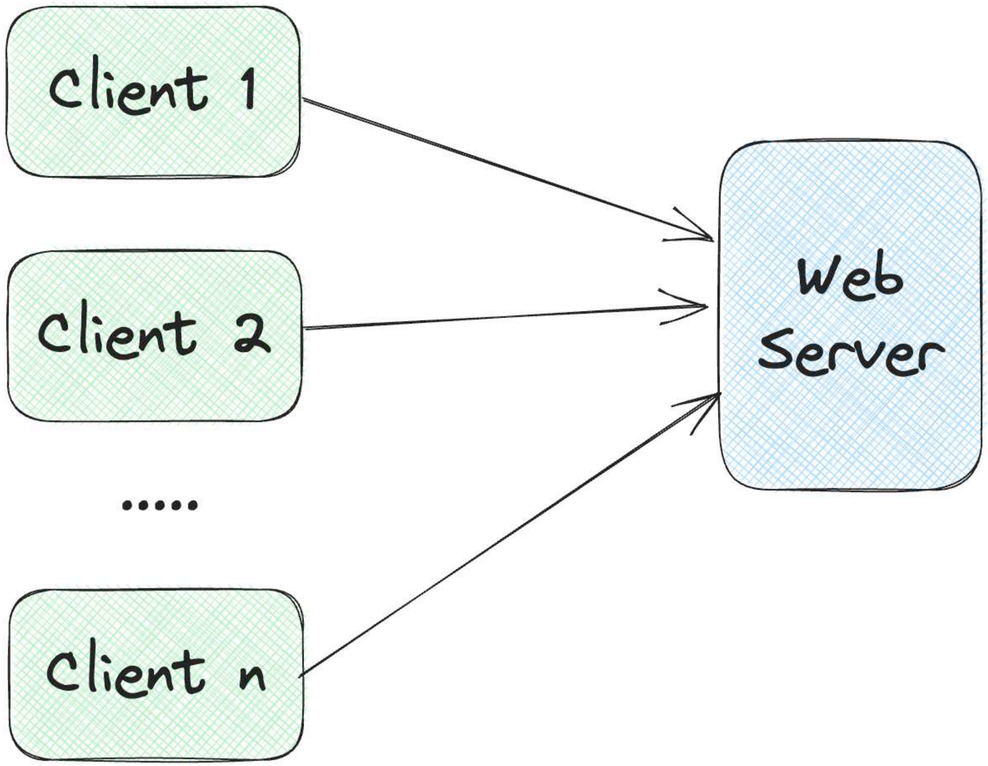
1. Spell checker works
2. Auto save happens
3. Auto formatting happens
4. ….



Software Engineering Research Centre

Think about a web server

- Web server runs a process to serve the clients
- Multiple clients may sent request to web server at the same time
- If the process handles each client sequentially - What can be an issue?
- How to make it more faster and better performing?
- What mechanism do we need? - Does multiple processes work?



An Analogy: Classrooms and Courses

Two Classrooms, two faculties teaching two different courses

Classroom 1: CS3.315 OS

Classroom 2: CS3.390 Networks

This is very similar to two separate processes

An Analogy: What if two faculties teach one course?

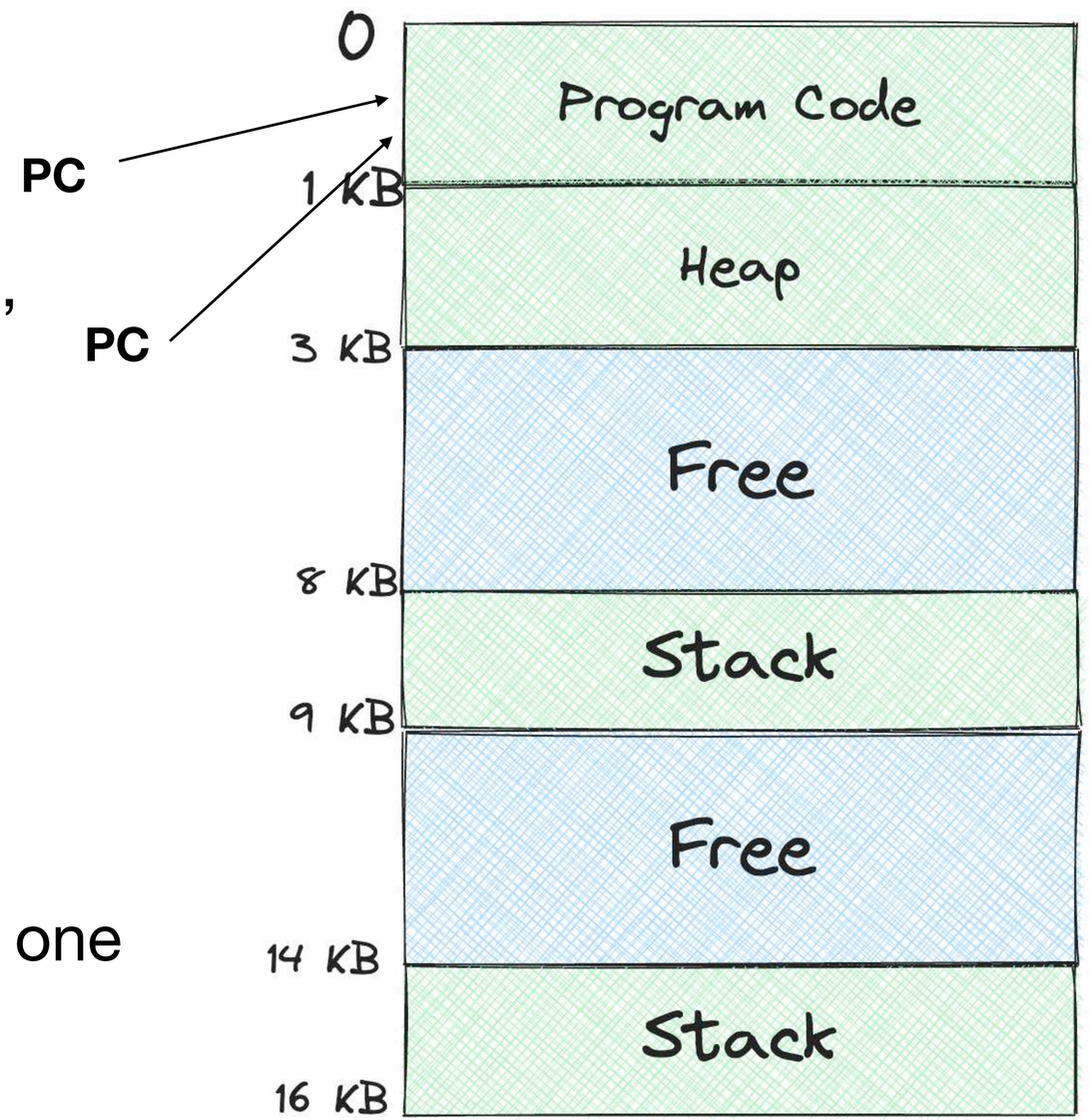
Two faculties teaching one course

Classroom 1: CS3.398 OS and Networks

- Can they teach at the same time?
  - Imagine such a scenario :-D
- Each teacher may take turns
- They may be at the class at the same time as well!
- There is only one attendance sheet, one course ID, one mark sheet
  - **Each faculty teaches in their style**
  - When question paper is set, they may take turns
  - The respective course content may be different
  - Somethings are shared!!

Process can have Threads!

- **Thread:** Another copy of the process that executes independently (lightweight process)
- Threads share the same address space (code, heap)
- Each thread:
  - Has separate Program counter
  - Separate stack for managing independent function calls
- In single thread, it was just about one PC and one stack



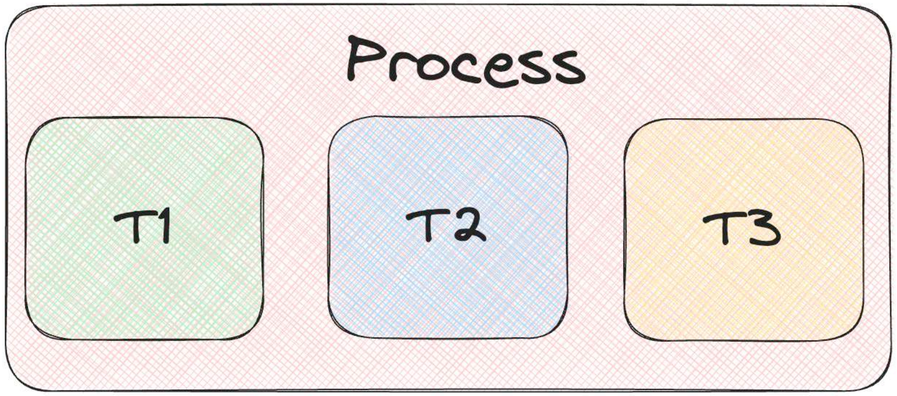
Wait, what about Process vs Threads?

Lets revisit parent and child - forks!

- What happens in a fork?
  - Parent and Child **do not share any memory**
    - Page tables are not shared, shared until changes - **Copy on Write (CoW)**
    - Subtle variations exist to improve efficiency but essentially parent and child are two different process
    - What about exec? - **Think!**
  - If they have to communicate, complicated inter process communication needs to be done (sockets, pipes, etc)
  - Extra copies of data, code, etc needs to be done

Threads

- Threads are another copy of process that executes independently
- Any process (parent process) can have multiple threads
  - Eg: Two threads T1 and T2
  - Both share the same address space - No separate page table, same code and same variables
  - Communication happens through shared variables (global)
  - Smaller memory footprint
- Threads are like separate process but share same address space

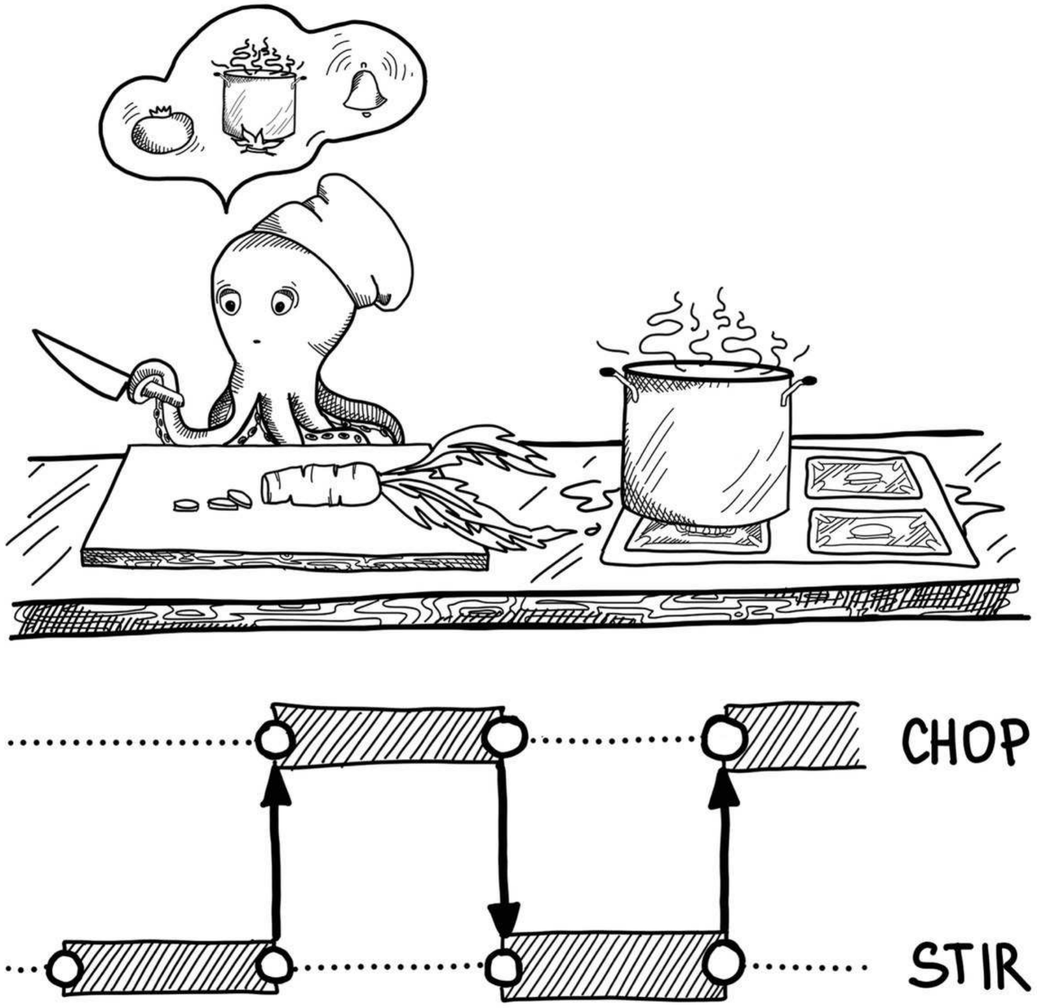


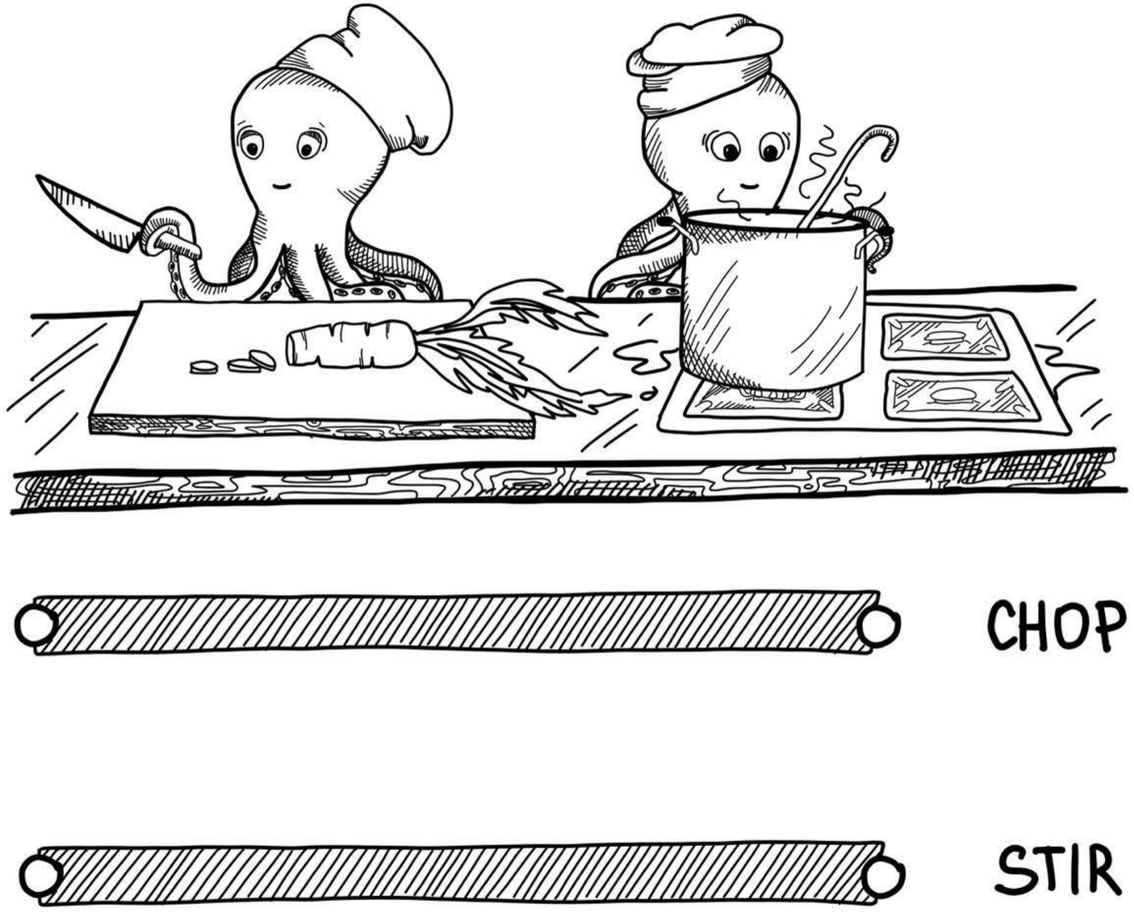
Why to do all these? Why Threads?

- Machine can be single core or multi-core:
  - Single process can effectively use multiple or even single CPU cores
  - Each thread can run independently and call different routines
  - Multi-threaded program has more than one point of execution
  - Within a process: one thread can perform I/O, one can perform computation, etc.
  - Scheduling happens between the threads - **Parallelism?**

Concurrency and Parallelism

What is what?





Source: https://freecontent.manning.com/concurrency-vs-parallelism/

Software Engineering Research Centre

Concurrency Vs Parallelism

Concurrency is about dealing with lot of things at once while parallelism is doing lot of things at once

- **Concurrency:** Running multiple threads/processes at the same time, even on a single CPU by interleaving their executions
- **Parallelism:** Running multiple threads/processes in parallel over different CPU cores
- Concurrent computations can be parallelized without changing correctness of result
- Concurrency by itself does not imply parallelism and vice versa
- Parallelism can be thought of as subclass of concurrency

Scheduling Threads

- OS schedules threads that are **ready**, similar to scheduling processes
- The context of thread (PC, registers) is saved into/restored from **Thread Control Block (TCB)**
  - Every PCB can have one or more linked TCBs corresponding to threads
- OS also has kernel level processes, each has threads - **Kernel threads**
  - **Kernel threads** can perform various tasks - system calls, handling interrupts, background tasks, etc. Execute in kernel mode. Eg: Linux pthreads
- **User threads** - managed by user level libraries. Execute in user mode
  - Eg: POSIX threads, anything that need not be managed by kernel

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Concurrency - Locks

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:

- Operating Systems in three easy pieces by Remzi et al.

Creating a Thread

- POSIX provides interface for management of threads - pthread.h

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine)(void *), void *arg)
```

- ***thread:** Pointer to pthread_t variable
- ***attr:** holds the attributes for new thread, stack size, scheduling policy, etc. NULL points to default
- ***start_routine:** Pointer to the function that will be executed by the thread upon execution
  - Takes a single void parameters and returns void value
- ***arg:** Argument that will be passed to the start_routine function
- Returns 0 if thread successfully created

Some Interesting things to be considered

```c
void *worker_thread(void *arg)
{
    printf("%s\n", (char *) arg);
}
```

```
Starting the threading demo
thread 1
thread 2
end
```

```
Starting the threading demo
thread 2
thread 1
end
```

- Order of execution can be non deterministic
- Its hard to predict which thread executes first
- Two executions have two different sequence here!!
- So what could have happened?

An Ideal Trace

main	Thread 1	Thread 2
Running prints “Starting the threading demo” Creates T1 Creates T2 Waits for T1		
	Runs Prints “thread 1” Returns	
Waits for T2		Runs Prints “thread 2” Returns
print “end”		

This can also happen!

main	Thread 1	Thread 2
Running prints “Starting the threading demo” Creates T1		
	Runs Prints “thread 1” Returns	
Creates T2		Runs Prints “thread 2” Returns
Waits for T1 Waits for T2 prints “end”		

Shared Data - More Tricky

```
void *worker_thread(void *arg)
{
    int index;
    for (index =0; index<max_index; index++)
    {
        counter++;
    }
}
```

```
Inital value of the counter 0
Final value of the counter 4000
```

```
Inital value of the counter 0
Final value of the counter 3790
```

- Max size: 2000, assume global variable counter initialised to 0
- Desired final result: **4000!, even on a single processor system there is no guarantee**
- **Why does this happen?**

Lets break the code down in assembly

counter = counter + 1

```
mov 0x8049a1c, %eax
add &0x1, %eax
mov %eax, 0x8049a1
```

1. Load memory value to register eax
2. Increment the value in the register by 1
3. Move the value from register back to memory

What can happen?

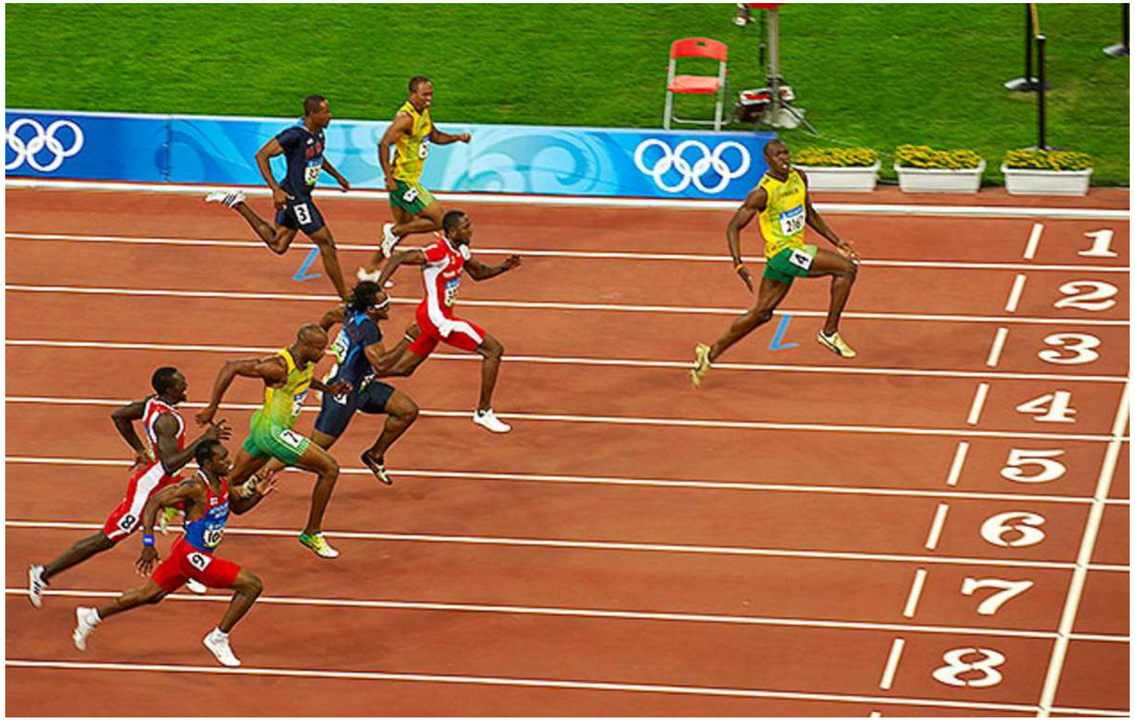
OS	Thread 1	Thread 2	Counter and Register (Initial value of counter = 50
	mov 0x8049a1c, %eax add $0x1, %eax		eax = 51 **counter = 50**
interrupt **Save T1' state** **Restore T2's state**			

		mov 0x8049a1c, %eax
		add $0x1, %eax
		Mov %eax, 0x8049a1c
			eax = 51
			counter = 51

interrupt **Save T2' state** **Restore T1's state**			

Race Condition and Critical Section

- **Race Condition: Condition where**
  - Multiple threads executing concurrently and
  - results depend on order of execution (time)
  - Scheduler can swap threads, also interrupts
  - Non-deterministic results
- **Critical Section:**
  - The section of code that is shared between the threads (leads to race conditions)
  - Shared variables or data



Source: https://www.si.com/olympics/2016/07/14/usain-bolt-2016-rio-olympics

Concurrency is tricky!

Race conditions can result in fatal issues

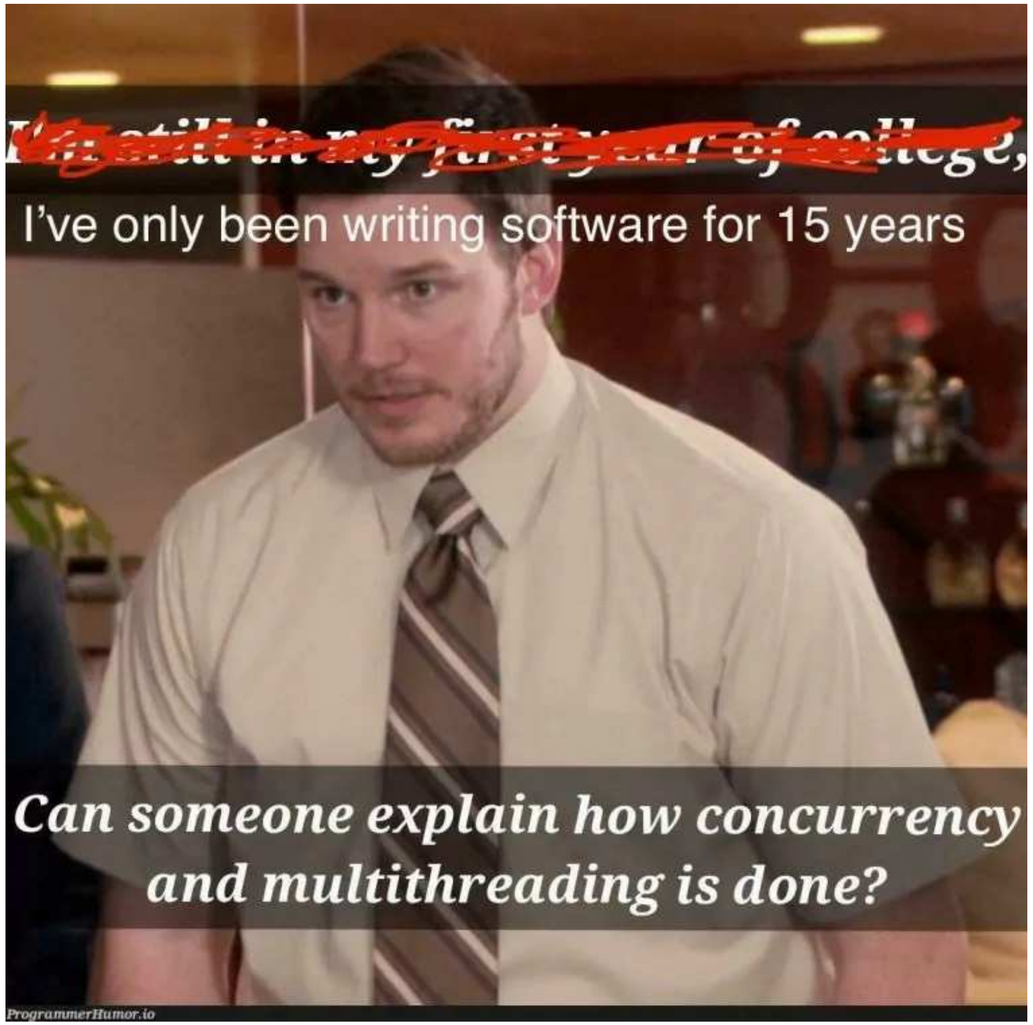
Therac 25

Northeast Blackout of 2003, US

https://www.everydayshouldbesaturday.com/2018/8/14/17687734/flashback-the-blackout-of-2003

Source: https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/

Concurrency can be tricky!



Source: programmerhumor.io

What can be done?

Bring in Atomicity

- What we want here is mutual exclusion!
  - When one thread is accessing critical section, others should wait
  - No two threads should access critical section at the same time
- In other words, **atomicity** needs to be provided
  - What if there was one instruction in assembly:
    - memory-add 0x8049a1c, &0x1 - Reality is not this!!

```
mov 0x8049a1c, %eax
add &0x1, %eax
mov %eax, 0x8049a1
```

This should execute atomically!!

What we need?

- Need to build synchronization primitives
  - Hardware + software support
  - Ensure that critical section is accessed in synchronised and controlled manner
- **One part:** Build some primitives for synchronisation
  - This will ensure atomicity (avoid race conditions)
- **Second part:** Ensure every thread gets access!
  - No one should starve

Some Issues needs to be addressed

- What support do we need from hardware?
- What support is needed from software?
- How to build primitives correctly and effectively?
- How can programs use these primitives?

Locks: A Basic Idea

- Lets go back to the shared variable code in **Critical Section (CS):**

$$counter = counter + 1$$

- What if we can have a lock surrounding the statement

```
lock_t mutex; //a global lock
....
lock (&mutex);   // lock the crticial section
counter = counter + 1;
unlock (&mutex); // release the lock
```

What are locks?

- Lock here is just a variable
- The lock variable holds the state of lock at any instant of time
- It is either available (free) or acquired:
  - **Available:** No threads hold the lock
  - **Acquired:** Lock not available, one thread is holding it and in CS
- We can also enrich with more information - which thread holds the lock, create a queue for threads to get locks, etc.

Lock and Unlock

- The thread that holds the lock - **Owner**
- Owner needs to call unlock to free the lock
  - The lock becomes free
  - There may be threads waiting for the lock, one of them will acquire it
  - The next thread with lock enters CS
  - When no thread is waiting for the lock, the lock stays as free
- **How to go about building a lock?**

How to go about building a lock?

- Think about classrooms and locks
- In the physical locks itself there are many options
- Many hardware primitives have been added to instruction set architecture to support locks
- The way in which the primitives are used + OS support forms the key

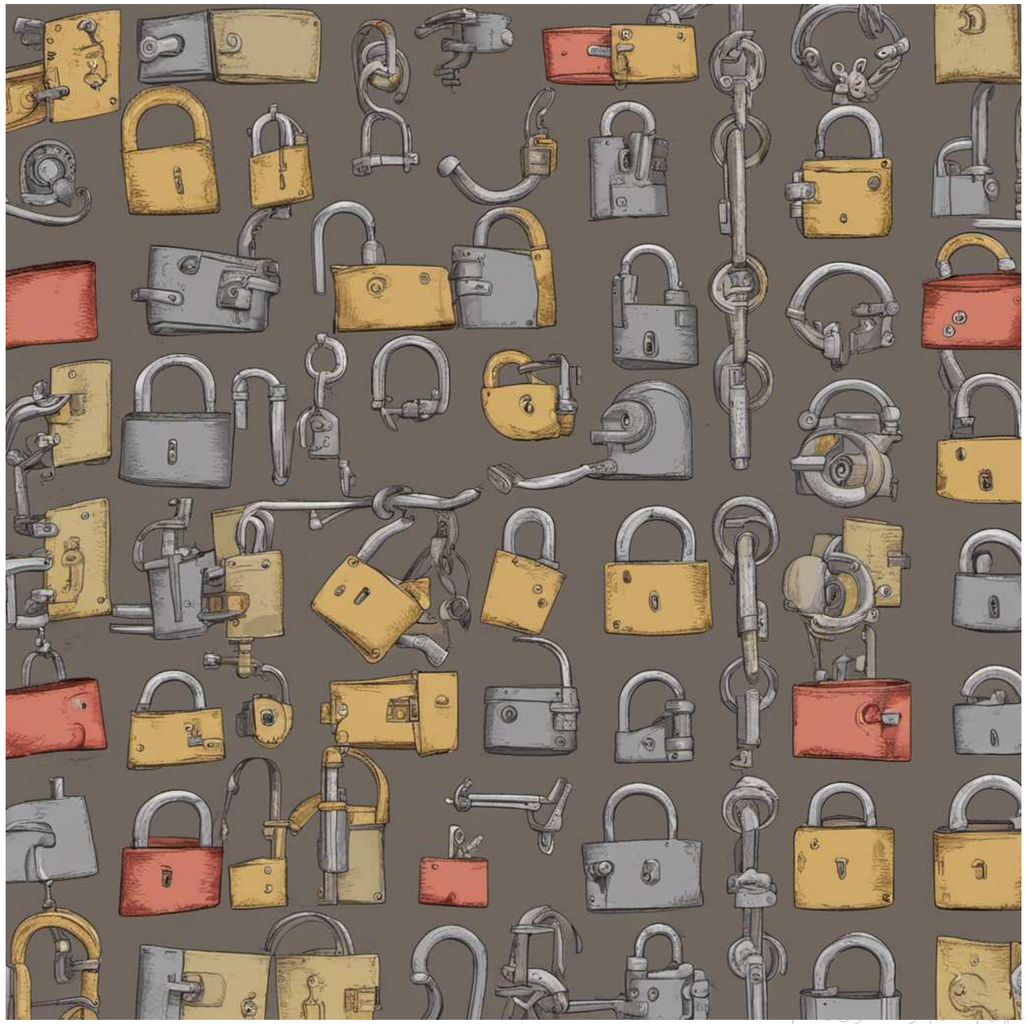


Image generated using stable diffusion

Criteria to evaluate

- **Mutual Exclusion**
  - Does the lock prevent multiple threads from entering CS at same time?
- **Fairness**
  - Does each thread get a fair chance to enter into the CS? - Avoid starvation!!
- **Performance**
  - When there is only one thread what is the overhead?
  - Multiple threads on single CPU - What about overhead?
  - Multiple threads on multiple CPU - What's are overhead?

Why threading is challenging?

- Two threads running at the same time
  - Interrupts - Can we disable interrupt inside a lock?

```
void lock()
{
  DisableInterrupts();
  //disables interrupts
}

void unlock()
{
  EnableInterrupt();
}
```

Main positive approach is simplicity

Are there any issues?

There are Negatives

- Thread gets a **very high privilege**
  - Thread can switch on and off the interrupts
  - Arbitrary thread can monopolize the processor
  - Errant program could call lock() and get into endless loop
- The approach does not work on multiple processor systems
  - Even if interrupts are disabled, other threads can run on different processor
- **Inefficiency:** Code that masks or unmasks interrupts are executed slowly
- Interrupt control is used in limited context as mutual exclusion primitive **(inside OS)**

Can we try with a software lock?

Software based lock

```
typedef struct __lock_t
{
    int flag;
} lock_t;

void init (lock_t *mutex)
{
  mutex -> flag = 0;
}

void lock(lock_t *mutex)
{
  while(mutex->flag==1)
    //do nothing
  mutex -> flag = 1;

}

void unlock (lock_t *mutex)
{
  mutex->flag = 0;
}
```

- Use software based locking mechanism
- Create a lock which has a flag value
- Every time a thread wants to enter CS
  - Invoke lock function, if flag = 1, wait for the lock to be available
  - Else acquire the lock and enter CS
- **Do you foresee some problem here?**

Simple Trace

Thread 1	Thread 2
Call lock() while (flag== 1) Interrupt to Thread 2	

	Call lock()
	While (flag==1)
	flag = 1;
	Interrupt to thread 1

flag = 1	

- No mutual exclusion - Both threads have flag set to 1
- Performance overhead due to spin-waiting

Working Spin Lock with Test-And-Set

Test and Set Instruction

Test and Set Lock

```
int TestAndSet(int *ptr, int new)
{
  int old = *ptr;
  *ptr = new;
  return old;
}
```

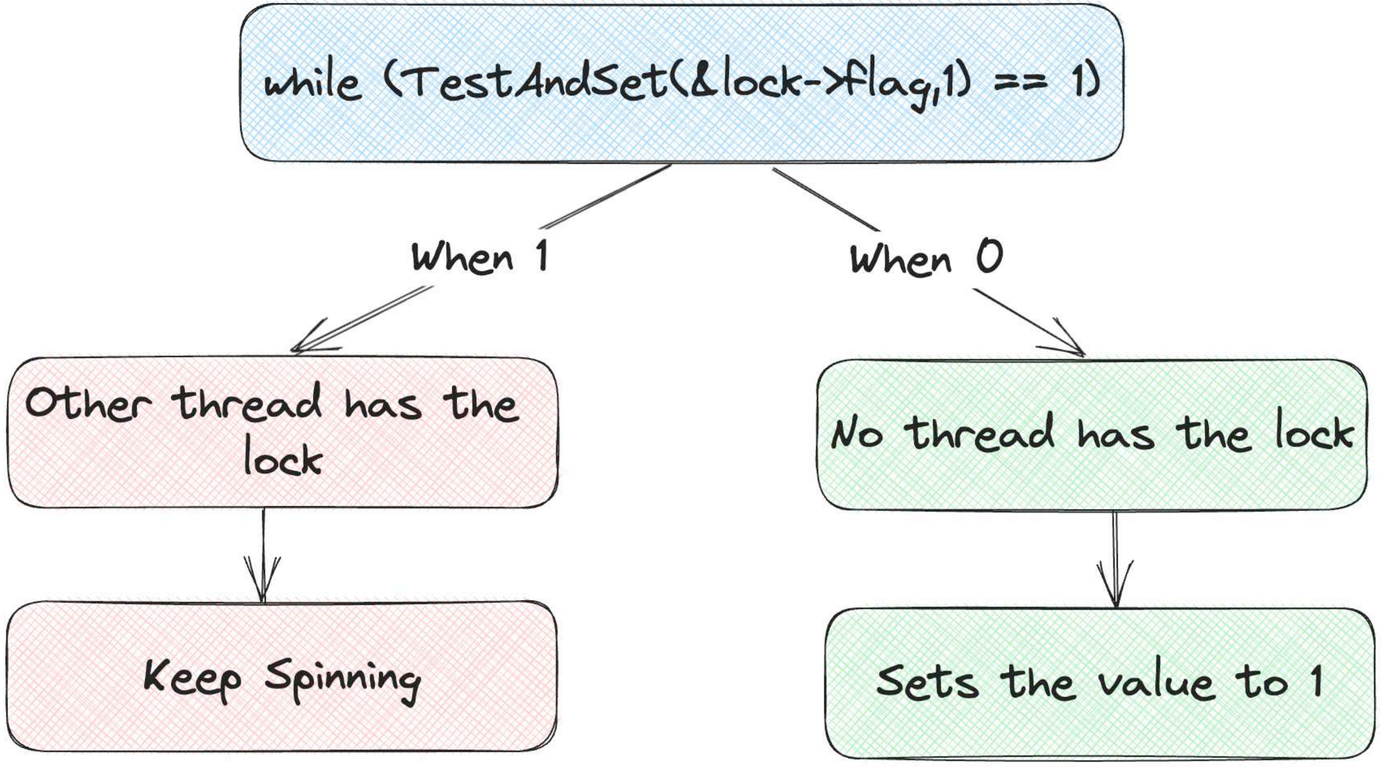
C Pseudocode

- Simplest hardware primitive - test-and-set or atomic exchange instruction
- Sequence of instructions executed atomically
- Enables testing of old values while setting the main to a new value
- Think about implementing a CS code with this lock!

Implementing using Test and Set Lock

Using TestAndSet Lock

```
typedef struct __lock_t
{
    int flag
} lock_t;


void init (lock_t *lock)
{
  lock -> flag = 0;
}

void lock (lock_t *lock)
{
  while (TestAndSet(&lock-> flag, 1)==1)
    // Keep spinning
}

void unlock (lock_t *lock)
{
  lock -> flag = 0;
}
```



Only one thread can acquire the lock

Evaluating Spin Locks

- **Correctness perspective** Provides mutual exclusion, so correct!
- **Fairness perspective**
  - They don’t provide fairness guarantee
  - Threads can keep spinning forever leading to starvation
- **Performance perspective**
  - In single CPU - significant overhead, What if thread holding the lock is preempted in critical section? - Run all N-1 threads to waste CPU
  - On multiple CPUs these locks work reasonably well (esp if n. Threads ~ n. CPUs)

Compare-And-Swap

Another Hardware Primitive

C Pseudocode

Compare-And-Swap

```
int CompareAndSwap(int *ptr, int expected, int new)
{
  int actual = *ptr;
  if (actual == expected)
  {
    *ptr = new;
  }
  return actual;
}
```

Inside the lock function the call will be:

```
while(CompareAndSwap(&lock->flag, 0,1) ==1)
```

- **Basic idea:** Test whether the value at address specified by ptr is equal to expected
  - If yes, update the memory location pointed to ptr by new value
  - If not, do nothing!
- In either case, return actual value at the memory location

Load-Linked and Store-Conditional (LL SC)

- In MIPS architecture, they can be used in tandem to build locks and concurrent structures

Load Linked

```
int LoadLinked (int *ptr)
{
  return *ptr;
}
```

Store Conditional

```
int StoreConditional(int *ptr, int value)
{
  if (no one has updated ptr since load linked)
  {
    *ptr = value;
     return 1;
  }
  else
  {
    return 0;
  }
}
```

C Pseudocode

LL/SC for building locks

- Load linked is like a typical load operation
  - Simply fetches a value from memory and places it in a register
- Store conditional succeeds if no intermittent store to address has taken place
  - In case of success, it updates ptr to value and returns 1 else returns 0

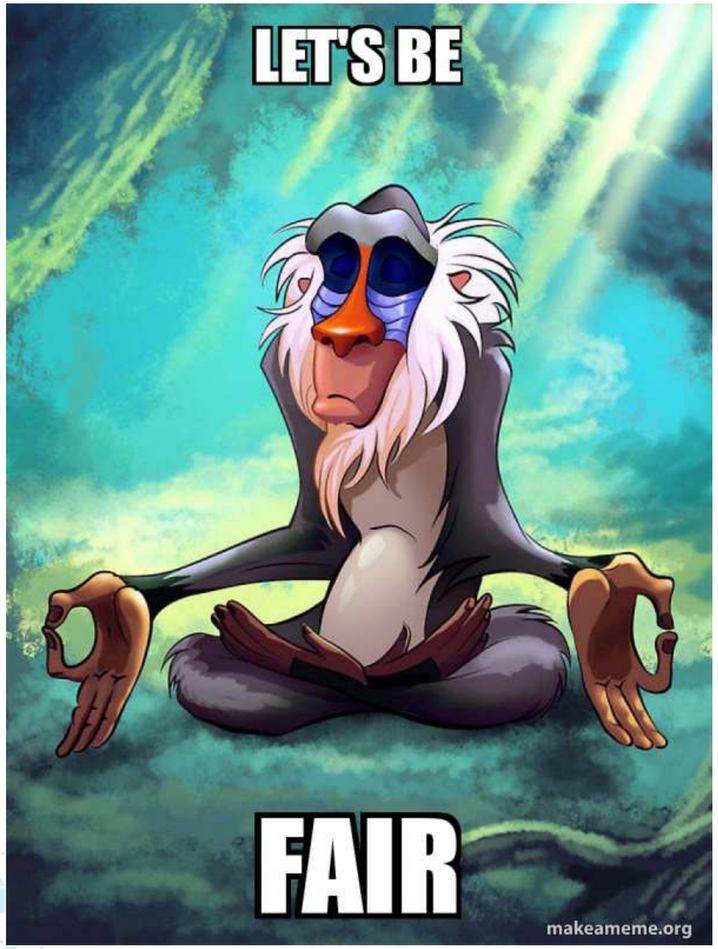
When does failure condition of store conditional arise?

LL/SC for building locks

```
void lock(lock_t *lock)
{
  while (1)
  {
    while (LoadLinked(&lock->flag)==1)
    ; //Keep spinning

    if (StoreConditional(&lock->flag,1)==1)
    {
     //store is successful, retrun else repeat all
     again
      return;
    }
  }
}

void unlock(lock_t *lock)
{
  lock->flag = 0;
}
```

What about Fairness?

Can we incorporate fairness through locks?



- Lock only has a flag variable
- Every time a thread acquires, it checks for flag
- However, which threads are checking is not recorded
- The threads that are looking for locks may have to be stored somewhere
- Can we store more information within each lock?

What if we leverage Turns and Tickets

Think about going to some crowded office space

Current Number and Window

Source: juvale.com

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Concurrency - Condition Variables

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:

- Operating Systems in three easy pieces by Remzi et al.

What if we leverage Turns and Tickets

Think about going to some crowded office space

Current Number and Window

Source: juvale.com

Fetch and Add

Yet another hardware primitive but very powerful

Fetch And Add

```
int FetchAndAdd(int *ptr)
{
  int old = *ptr;
  *ptr = old + 1;
  return old;    //returns old value
}
```

- Atomically increment a value while returning the old value at a particular address
- Used to build interesting type of lock - **The ticket lock**
- Instead of single variable a combination of ticket and turn variable is used
- Not just flag: ticket and turn

Ticket Lock

Ticket Lock

```
typedef struct __lock_t
{
    int ticket;
    int turn;
} lock_t;

void lock_init (lock_t *lock)
{
  lock_t -> ticket = 0;
  lock_t -> turn = 0;
}
```

Ticket Lock

```
void lock (lock_t *lock)
{
  int myturn = FetchAndAdd(&lock->ticket);
  //when ticket value = my turn, thread goes into CS
  while (lock->turn != myturn)
  {
    // keep spinning
  }
}

void unlock (lock_t *lock)
{
  // next waiting thread can enter CS
  FetchAndAdd(&lock->turn);
}
```

An Illustration of Ticket Lock

Four Processor Ticket Lock Example

Row	Action	next_ticket	now_serving	P1 my_ticket	P2 my_ticket	P3 my_ticket	P4 my_ticket
1	Initialized to 0	0	0	-	-	-	-
2	P1 tries to acquire lock (succeed)	1	0	0	-	-	-
3	P3 tries to acquire lock (fail + wait)	2	0	0	-	1	-
4	P2 tries to acquire lock (fail + wait)	3	0	0	2	1	-
5	P1 releases lock, P3 acquires lock	3	1	0	2	1	-
6	P3 releases lock, P2 acquires lock	3	2	0	2	1	-
7	P4 tries to acquire lock (fail + wait)	4	2	0	2	1	3
8	P2 releases lock, P4 acquires lock	4	3	0	2	1	3
9	P4 releases lock	4	4	0	2	1	3
10	...	4	4	0	2	1	3

Source: wikipedia article on ticket lock

How good is the spin based locks?

- Simple hardware based locks are simple to implement and powerful
- They are also quite inefficient especially when it comes to performance
  - Consider that there are two threads and one thread has the lock
  - When thread has lock, it may get interrupted, the other thread spins for a time slice, waste CPU cycle
  - Think about N threads, N-1 threads might waste CPU cycles in spinning (especially if round robin)
- **Can we come up with something better instead of wasting cycles with spinning?**

OS support can help

The yield call

- If the thread is aware that it is going to spin - Why not give up the CPU to some other thread?
  - Simple OS primitive system call: **yield()**
  - Moves the thread from running state to ready state => another thread can run
  - Does this solution work efficiently?
    - What if there are 100 threads?
    - Still costly! - 99 threads runs to yield
    - Possibility of infinite yields as well - **Starvation! - Why?**

```
while (TestAndSet(&lock->flag,1) == 1)
{
    yield ();
}
```

Can we make thread sleep rather than spinning?

- Why don’t we make use of some queue based structures?
- Keep a queue to track which thread needs access to CS
- Syscalls by Solaris: **park()** and **unpark()**
  - park(): puts a thread to sleep
  - unpark(tid): wakeup that particular thread
- If a thread wants to acquire a lock
  - Check if others have the lock, if yes put thread to sleep
  - If lock is free, wake up the thread and give the lock

Locks do help in access to CS! But more challenges

- Locks ensures that thread can get access to CS
  - With help of HW and SW mechanisms efficient locks can be built
- But, thread while executing may want to check for some conditions
  - A parent thread may want to check if the child thread has completed before proceeding
    - Remember join() operation? - How to make it work?
    - **Why don't we use shared variable?**

Checks using shared variable

```
int done = 0;

void *child (void *arg)
{
  printf (" child\n");
  done = 1;
  return NULL;
}

int main (int argc, char *argv[])
{
  printf ("parent\n");
  pthread_t c;
  pthread_create(&c, NULL, child, NULL);
  while (done == 0)
  {
    ; //Keep spinning
  }
  printf (" done \n");
  return 0;
}
```

Condition Variables

- **Condition variable:** Explicit queues that the threads can put themselves on when a state of condition is not as desired
  - **Eg:** lock is not available (flag might be 0)
- When condition is met, thread can be woken up to continue

pthread_cond_t c;

- c is a condition variable with two operations - wait() and signal()
- **wait():** when thread wants to put itself to sleep
- **signal():** there is some change and thread wants to wake up thread waiting on condition

Condition Variables in Action

```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c  = PTHREAD_COND_INITIALIZER;
void thread_exit()
{
    pthread_mutex_lock(&m);
    done = 1;
    pthread_cond_signal(&c);
    pthread_mutex_unlock(&m);
}

void *worker_thread(void *arg)
{
    printf("child \n");
    thread_exit();
    return NULL;
}
```

```c
void thread_join()
{
    pthread_mutex_lock(&m);
    while (done == 0)
    {
        pthread_cond_wait(&c,&m);
    }
    pthread_mutex_unlock(&m);
}

int main (int argc, char *argv[])
{
    pthread_t thread_p1;
    printf("Starting parent thread \n");
    pthread_create(&thread_p1, NULL, worker_thread, NULL);
    thread_join();
    printf("Parent: end\n");
    return 0;
}
```

Two cases to consider as it works

- **Parent creates the Child and continues running**
  - Goes into the join call
  - Checks the state variable since child is not done, puts itself to sleep
  - Child runs and invokes exit -> updates state variable and wakes up parent thread
  - Parent will run returning from wait and prints done
- **Child runs immidiately upon creation**
  - Sets done to 1, wakes up sleeping thread (none available) so returns
  - Parent runs join, the done variable is 1 so returns
- **Do we need while loop for checking state and do we need locks?**

State variable and Locks

- **What if we don't have the state variable done**
  - Exit and join functions simply calls wait and join
  - What if child runs first and calls exit, child will signal but no parent thread
  - When parent runs, it will simply wait and never come out of it
- **What if there are no locks around statements in exit and join?**
  - Parent calls join, checks that done is 0, sleeps
  - Just before sleep call, interrupt, child runs and sets done to 1 and signals
  - No thread is waiting, parent runs goes into sleep and forever sleeps - **Race condition**

An Analogy

chef prepares food

Prepared items

orders food via app

orders form counter

One cannot get food items that are not yet ready!

The Producer/Consumer Problem

AKA Bounded Buffer Problem

- Think about web servers
  - **Producer:** Produces HTTP requests into a queue
  - **Consumer:** The threads that process the HTTP requests from the queue
  - **Bounded buffer:** The work queue
- Piped calls in unix: grep linux os.txt | wc -l
  - **Producer:** grep gets text that contains “linux” from os.txt and puts them to standard output
  - **Consumer:** Shell redirects them to pipe call, where wc as another process counts and prints the number of lines
  - **Buffer:** Shared resource

Wait There is a challenge

- Bounded buffer is a shared resource
- Producer puts data to empty buffer
- Consumer can only consume from full buffer
- We need synchronisation mechanisms to access it
  - Else it may result in **race conditions**
- How to solve the problem?
- What kind of synchronisation mechanisms can be developed?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Concurrency - Semaphores and Classical Concurrency Problems

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:

- Operating Systems in three easy pieces by Remzi et al.

Producer-Consumer Problem

The Bounded Buffer Problem

chef prepares food

Prepared items

orders food via app

orders form counter

One cannot get food items that are not yet ready!

Lets start simple

- Consider buffer can hold only one item, a single integer - **How to solve?**

Producer-Consumer-GetAndPut

```
int buffer = 0
int count  = 0

int get()
{
  assert(count==1);
  count = 0;
  return buffer;
}

void put (int value)
{
  assert (count==0);
  buffer = value;
  count = 1;
}
```

Producer-Consumer

```
void *producer (void *arg)
{
  int i;
  int maxLoops = (int) arg;
  for (i=0; i<maxLoops; i++)
  {
    put(i);
  }
}

int *consumer(void *arg)
{
  int value;
  while (1)
  {
    value = get();
    printf("%d\n", value);
  }
}
```


Software Engineering Research Centre

Surround with Locks and Condition Variables

Only one producer and one consumer

Producer

```
cond_t cond;
mutex_t mutex;

void *producer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0; i<maxLoops; i++)
  {
    pthread_mutex_lock(&mutex); //get the lock into CS
    if (count==1) // check if something exist
    {
      pthread_cond_wait(&cond,&mutex);
    }
    put (i);
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&unlock);
  }
}
```

Consumer

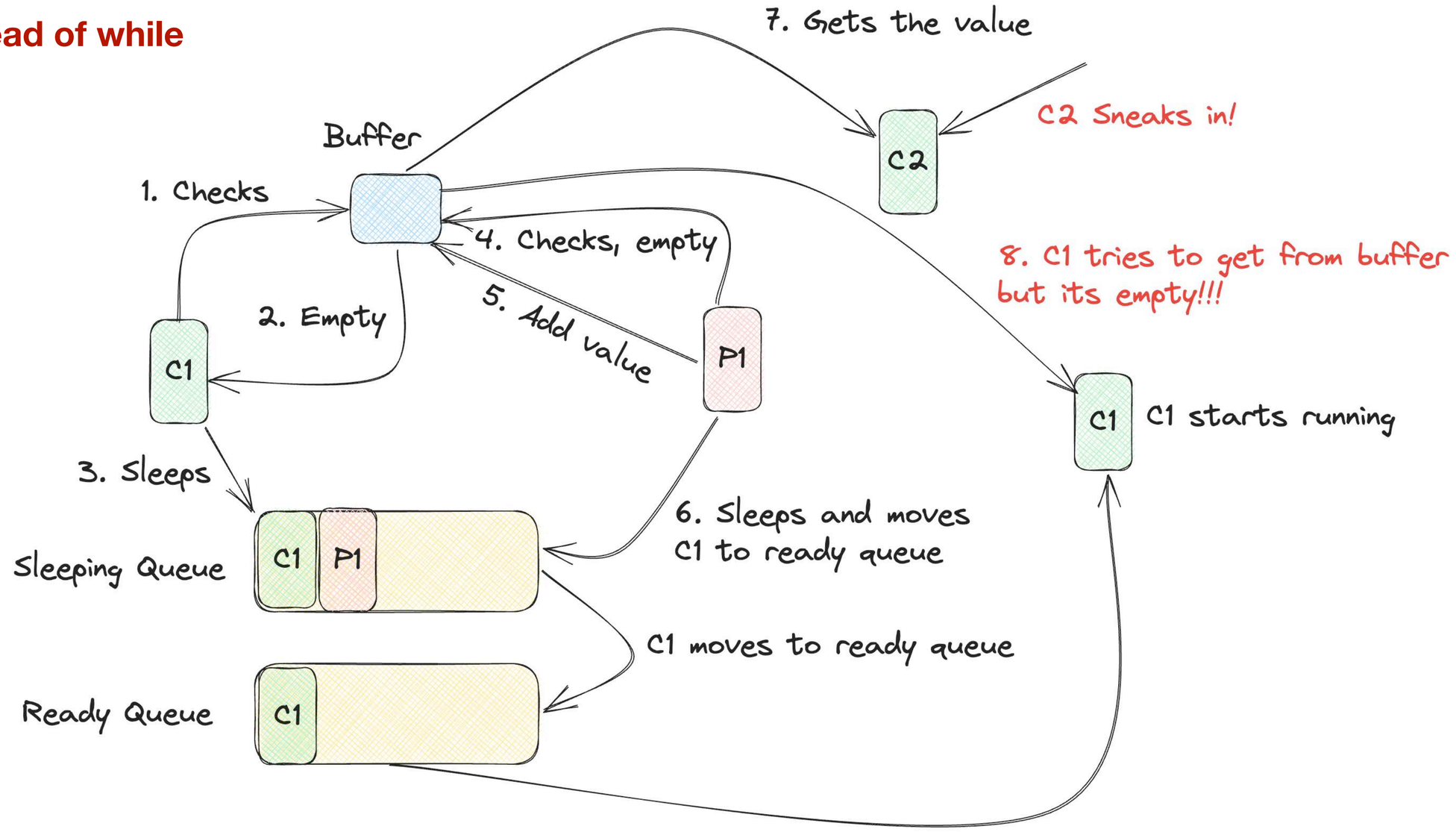
```
cond_t cond;
mutex_t mutex;

void *consumer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0; i<maxLoops; i++)
  {
    pthread_mutex_lock(&mutex); //get the lock into CS
    if (count==0) // check if there is nothing
    {
      pthread_cond_wait(&cond,&mutex);
    }
    int temp = get();
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&unlock);
    printf ("%d\n", temp);
  }
}
```


What if there are multiple consumers?

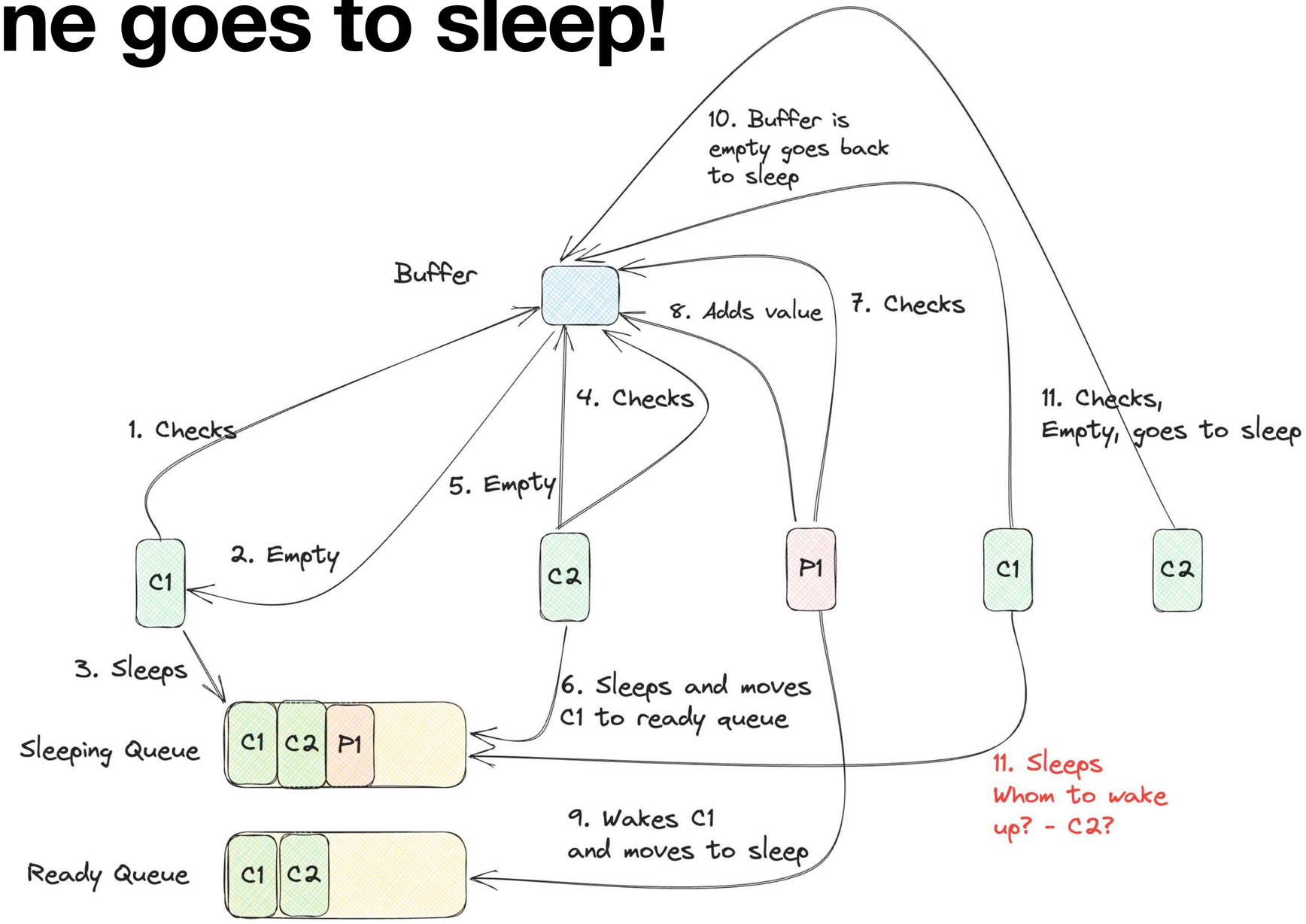
What if there are more producers and consumers

Two Key Challenges

If instead of while



Everyone goes to sleep!



Use Two Condition Variables

```
cond_t fill;
cond_t empty;
```

- **Producer waits on empty** condition => waits for consumer to empty the buffer
  - Signals on fill => signals consumer that buffer is filled!
- **Consumer waits on fill** condition = > waits for producer to fill buffer
  - Signals on empty => signals producer that buffer is empty!
- Producer cannot awaken producer and consumer cannot awaken consumer
- What about more than one in the buffer? - **Buffer can be an array of integers**

Producer Consumer Problem Solution

Get and Put for large sized buffer

```
int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;

void put (int value)
{
  buffer[fill] = value;
  fill = (fill + 1)%MAX;
  count ++;
}

int get()
{
  int tmp = buffer[use];
  use = (use + 1)%MAX;
  count --;
  return tmp;
}
```

- Buffer now can hold an array of integers
- Fill and use are used to manage indexing
- Producers can keep pushing data to the buffer
- Consumers can keep reading data from the buffer
- How to implement producer and consumer?

Producer Consumer Problem Solution

Producer with two condition variables

```
cond_t empty;  //two condition variables
cond_t fill;
mutex_t mutex;

void *producer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0; i<maxLoops; i++)
  {
    pthread_mutex_lock(&mutex); //get the lock into CS
    while (count==MAX) // check if its already full
    {
      pthread_cond_wait(&empty,&mutex);
    }
    put (i);
    pthread_cond_signal(&fill);
    pthread_mutex_unlock(&unlock);
  }
}
```

Consumer with two condition variables

```
cond_t empty;  //two condition variables
cond_t fill;
mutex_t mutex;

void *consumer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0; i<maxLoops; i++)
  {
    pthread_mutex_lock(&mutex); //get the lock into CS
    while (count==0) // check if there is nothing
    {
      pthread_cond_wait(&fill,&mutex);
    }
    int temp = get();
    pthread_cond_signal(&empty);
    pthread_mutex_unlock(&unlock);
    printf ("%d\n", temp);
  }
}
```


Software Engineering Research Centre

Is there a better way to do this?

- **Locks:** Provide atomic access to critical section

- **Condition Variables:** Allows signalling between threads or passing some information on condition between threads

- What if both can be done using a single mechanism?
  - Edsger W. Dijkstra did that through the concept of **Semaphores**

Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better

Semaphore: One structure which can act as both condition Variable and lock

Edsger W. Dijkstra

An Analogy

May be a waiter can help better?

chef prepares food

Prepared items

orders food via app

orders form counter

Waiter

Semaphore

- An object with an integer value that we can manipulate with two routines: wait and post. As per original naming:
  - **P(): proberen** - Decrease the value, Check
  - **V(): Verhogen** - Increase the value
- In POSIX, there are two routines:
    - **sem_wait():** decrease the semaphore, if negative block
    - **sem_post():** increase the semaphore value

Semaphore

Semaphore

```
#include <semaphore.h>
sem_t s;
sem_init (&s, 0, 1);
```

Semaphore shared between threads in same process

Initial value of semaphore

Semaphore - Wait

```
int sem_wait(sem_t *s)
{
  // decrement s by 1
  // wait if value of s is negative
}
```

Semaphore - Post

```
int sem_post(sem_t *s)
{
  // increment value of s by 1
  // if there are threads waiting,
  // wake one of them
}
```

Semaphore

- **sem_wait():**
  - Either, it will either return right away after decrementing the value
  - Or, it will cause the caller to suspend execution waiting for a subsequent post
  - When there are multiple threads, they can call wait and get queued
- **sem_post():**
  - Simply increments the value
  - If the thread is waiting, wakes one of them up
- Value of semaphore, when negative equals to number of waiting threads

Semaphores as Locks

Binary Semaphores - How to use Semaphores as locks?

- Always think about what should be the initial value of semaphore, **here it is 1**
- Assume there are two threads
  - Thread 0 calls sem_wait()
  - Decrements the value to 0
  - Thread 0 can enter CS
  - At this time if Thread 1 wants to enter CS -> calls sem_wait() -> -1, sleeps
  - Once thread 0 is done, calls sem_post
  - Increments value by 1, wakes thread 1

Semaphore - Locks

```
sem_t sem_var;
sem_init(&sem_var, 0, 1);
sem_wait (&sem_var);
//Critical section code here
sem_post (&sem_var);
```

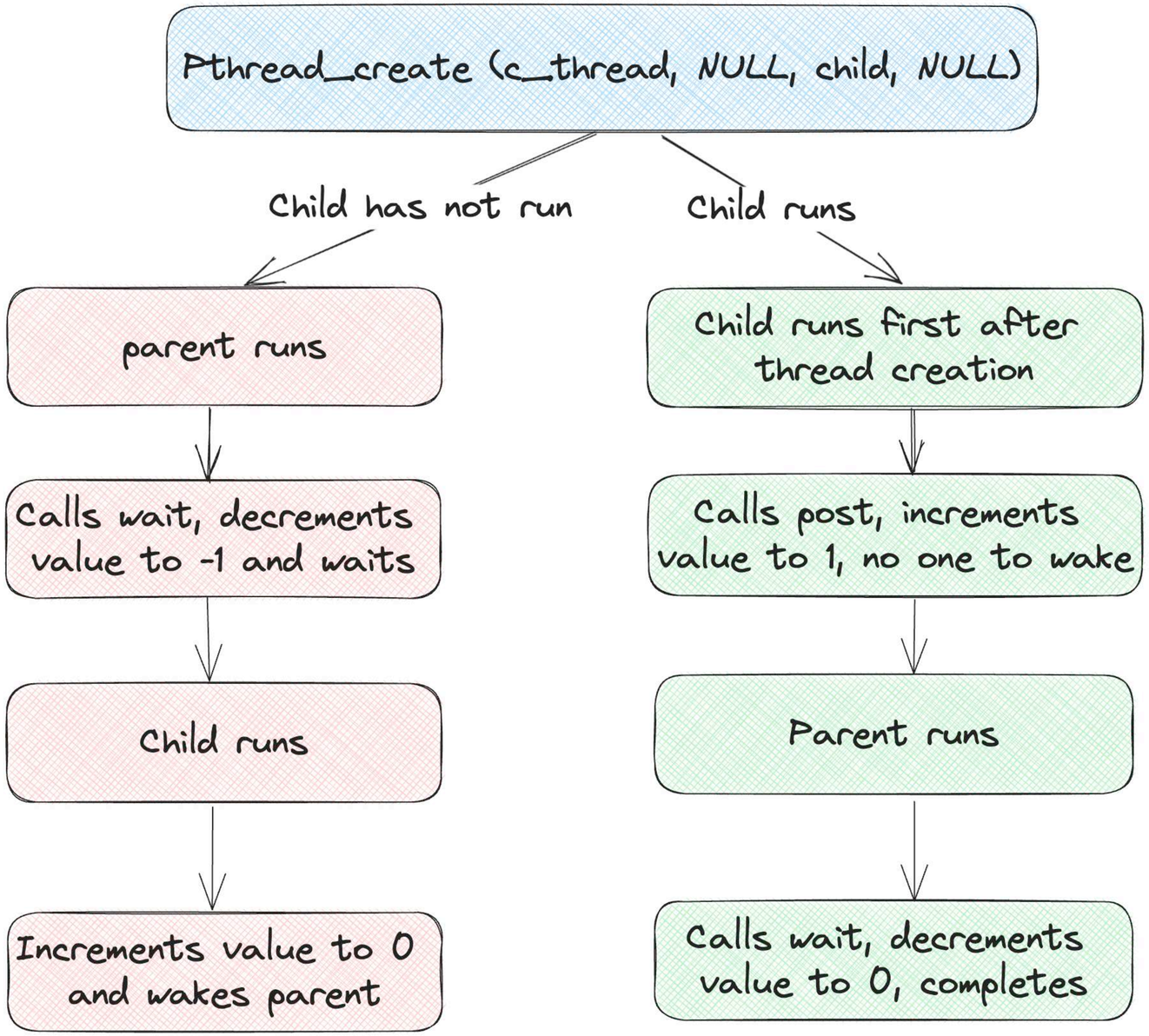
Semaphores can also function as condition Variables

Semaphore - Condition variables

```
sem_t sem_var;

void *child(void *arg)
{
  printf("child\n");
  sem_post(&sem_var);
  return NULL;
}

int main (int argc, char *argv[])
{
  sem_init(&s, 0, 1);
  pthread_t c_thread;
  pthread_create(c_thread, NULL, child, NULL);
  sem_wait(&sem_var);
  printf("parent\n");
}
```

- There are two main possible execution
- Parent runs, create the child and the child has not run yet
- Parent runs, creates the child and the child immidiately runs
- How does the semaphore help with both the above condition?
  - **What should be value of sem_var?**

Semaphores as condition variables



Producer Consumer Problem Using Semaphores

- Let us start with 2 semaphores: empty and wait, Buffer with MAX = 1

Get and Put for large sized buffer

```
int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;

void put (int value)
{
  buffer[fill] = value;
  fill = (fill + 1)%MAX;
  count ++;
}

int get()
{
  int tmp = buffer[use];
  use = (use + 1)%MAX;
  count --;
  return tmp;
}
```

Producer-Consumer with buffer

```
sem_t empty;
sem_t full;

void *producer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0;i<maxLoops;i++)
  {
    sem_wait(&empty);
    put (i);
    sem_post(&full);
  }
}

void *consumer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0;i<maxLoops;i++)
  {
    sem_wait(&full);
    int tmp = get();
    sem_post(&empty);
    printf("%d\n", tmp);
  }
}
```

Is our solution fine?

- Consider two threads (producer and consumer) on single thread
- Assume consume runs first sem_wait(&full)
  - Decrements **full (0) to -1** and waits for the thread to call post
  - Moves to a blocked state
- Producer runs, calls sem_wait (&empty)
  - **Empty (1) is decremented to 0** and proceeds to add value
  - Once done, calls post and moves consumer to ready
  - If producer runs again, it will keep looping, consumer when runs, can get the lock
- This can work for multiple producers and consumers but what **if MAX>1**

Is our solution fine?

- Consider two threads (producer and consumer) on single thread
- Assume consume runs first sem_wait(&full)
  - Decrements **full (0) to -1** and waits for the thread to call post
  - Moves to a blocked state
- Producer runs, calls sem_wait (&empty)
  - **Empty (1) is decremented to 0** and proceeds to add value
  - Once done, calls post and moves consumer to ready
  - If producer runs again, it will keep looping, consumer when runs, can get the lock
- This can work for multiple producers and consumers but what **if MAX>1**

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Classical Concurrency Problems and Concurrency Bugs

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:

- Operating Systems in three easy pieces by Remzi et al.

Producer Consumer Problem Using Semaphores

- Let us start with 2 semaphores: empty and wait, Buffer with MAX = 1

Get and Put for large sized buffer

```
int buffer[MAX];
int fill = 0;
int use = 0;
int count = 0;

void put (int value)
{
  buffer[fill] = value;
  fill = (fill + 1)%MAX;
  count ++;
}

int get()
{
  int tmp = buffer[use];
  use = (use + 1)%MAX;
  count --;
  return tmp;
}
```

Producer-Consumer with buffer

```
sem_t empty;
sem_t full;

void *producer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0;i<maxLoops;i++)
  {
    sem_wait(&empty);
    put (i);
    sem_post(&full);
  }
}

void *consumer(void *arg)
{
  int i;
  int maxLoops = (int)arg;
  for (i=0;i<maxLoops;i++)
  {
    sem_wait(&full);
    int tmp = get();
    sem_post(&empty);
    printf("%d\n", tmp);
  }
}
```

Is our solution fine?

- Consider two threads (producer and consumer) on single thread
- Assume consume runs first sem_wait(&full)
  - Decrements **full (0) to -1** and waits for the thread to call post
  - Moves to a blocked state
- Producer runs, calls sem_wait (&empty)
  - **Empty (1) is decremented to 0** and proceeds to add value
  - Once done, calls post and moves consumer to ready
  - If producer runs again, it will keep looping, consumer when runs, can get the lock
- This can work for multiple producers and consumers but what **if MAX>1**

What about buffer with MAX>1

- Assume two producers, **P1 and P2**
- P1 runs first, fills the buffer entry, before updated, interrupt happens
- P2 starts to run and overwrites the value written by P1
- The reason:
  - Two producers **calling put() at the same time!!**
  - Race condition is triggered!
- Remember we have not locked get and put here. What can be done?

Add mutex to solve Producer-Consumer

Producer

```
sem_wait (&mutex);
sem_wait (&empty);
put(i);
sem_post (&full);
sem_post (&mutex);
```

Consumer

```
sem_wait (&mutex);
sem_wait (&full);
get();
sem_post (&empty);
sem_post (&mutex);
```

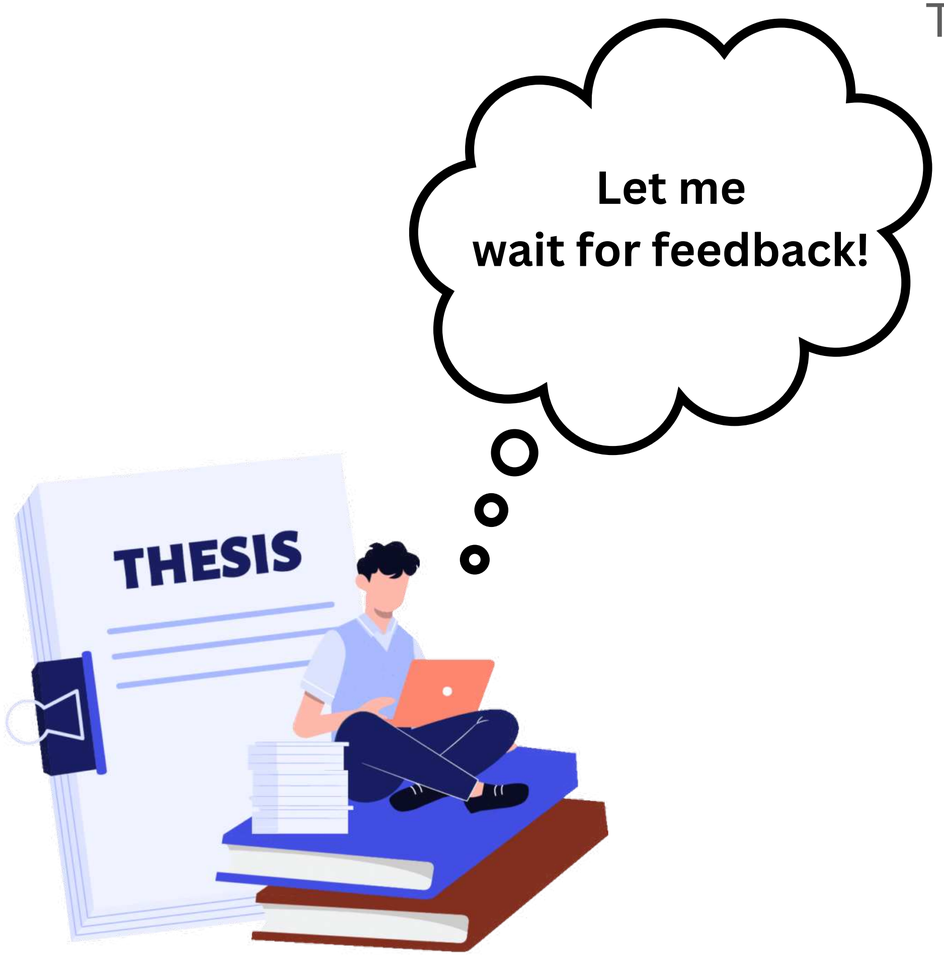
- Is there any issue with above code?
- C1 runs first gets mutex but waits on empty, P1 runs but waits for mutex - **Deadlock!!**

Deadlocks?

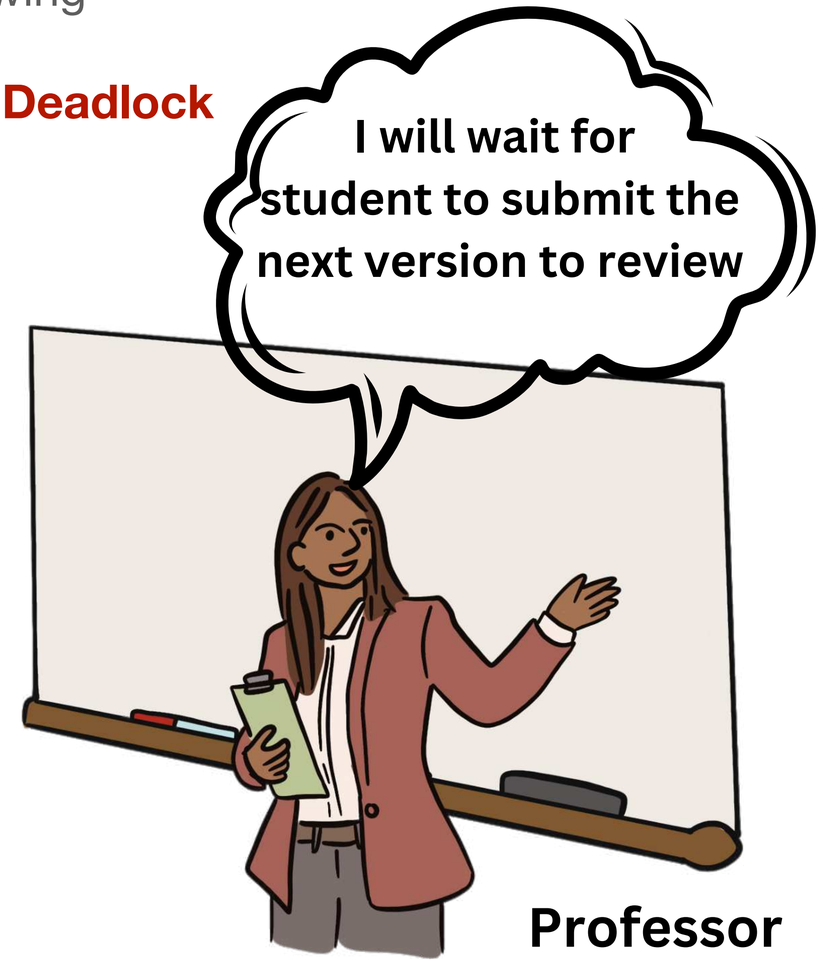
Student has submitted a draft and
Waits for review

Professor is expecting student
To submit better version to start
Reviewing

Both wait - Deadlock



Student



Professor

Producer Consumer Problem Using Semaphores

The Solution

Producer

```
sem_wait (&empty);
sem_wait (&mutex);
put(i);
sem_post (&mutex);
sem_post (&full);
```

Consumer

```
sem_wait (&full);
sem_wait (&mutex);
get();
sem_post (&mutex);
sem_post (&empty);
```

- Add mutex lock around put and get
- Let producer and consumer get the signal and then lock when entering CS

An Analogy

One Person writing

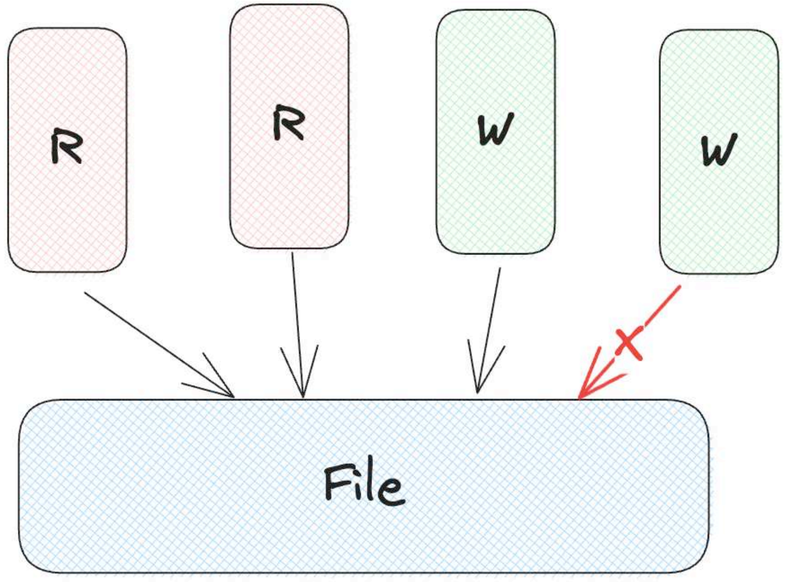
Many people reading at the same time

Online word processors

Databases

Readers/Writers Problem

- **Reader:** Process or thread that reads from memory
- **Writer:** Process or thread that writes on the memory
- Two readers can work on the same file at the same time
- Multiple writers cannot work on the same file at the same time



Readers/Writers Problem

Intuition

- **Only one writer can write at any point of time!**
- Reader thread can come in:
  - More readers come in, they can be allowed access
  - The moment writers come, it can be blocked
  - Once readers are done with reading, writers can start writing
- Can you think about writing a solution to this?
  - Do you foresee any challenges here?

Readers/Writers Problem

Readers/Writers Problem Solution

```
typedef struct _rwlock_t
{
    int readers;
    sem_t lock;
    sem_t writelock;
}rwlock_t;

void rwlock_init(rwlock_t *rw)
{
  rw -> readers = 0;
  sem_init(&rw->lock,0,1);
  sem_init(&rw->writelock,0,1);
}
```

Readers/writers Problem Solution

Readers/Writers Problem Solution

```
void acquire_readlock(rwlock_t *rw)
{
  sem_wait(&rw->lock);
  rw->readers++;
  if(rw->readers == 1)
    //disable writers to enter
    sem_wait(&rw->writelock);
  sem_post(&rw->lock);
}

void release_readlock(rwlock_t *rw)
{
  sem_wait(&rw->lock);
  rw->readers --;
  if(rw->readers == 0)
    //free the write lock
    sem_post(&rw->writelock);
  sem_post(&rw->lock);
}

void acquire_writelock(rwlock_t *rw)
{
  sem_wait(&rw->writelock)
}

void release_writelock(rwlock_t *rw)
{
  sem_post(&rw->writelock)
}
```

Writers Starve!!!

Readers/Writers Problem Solution

Add a lock that can act as priority common to both

Readers Writers - Better solution

```
sem_t serviceQueue;

//writer code code

sem_wait(&serviceQueue);
sem_wait(&rw->writelock)

  ......


sem_post(&rw-> writelock)
sem_post (&serviceQueue);
```

Writer

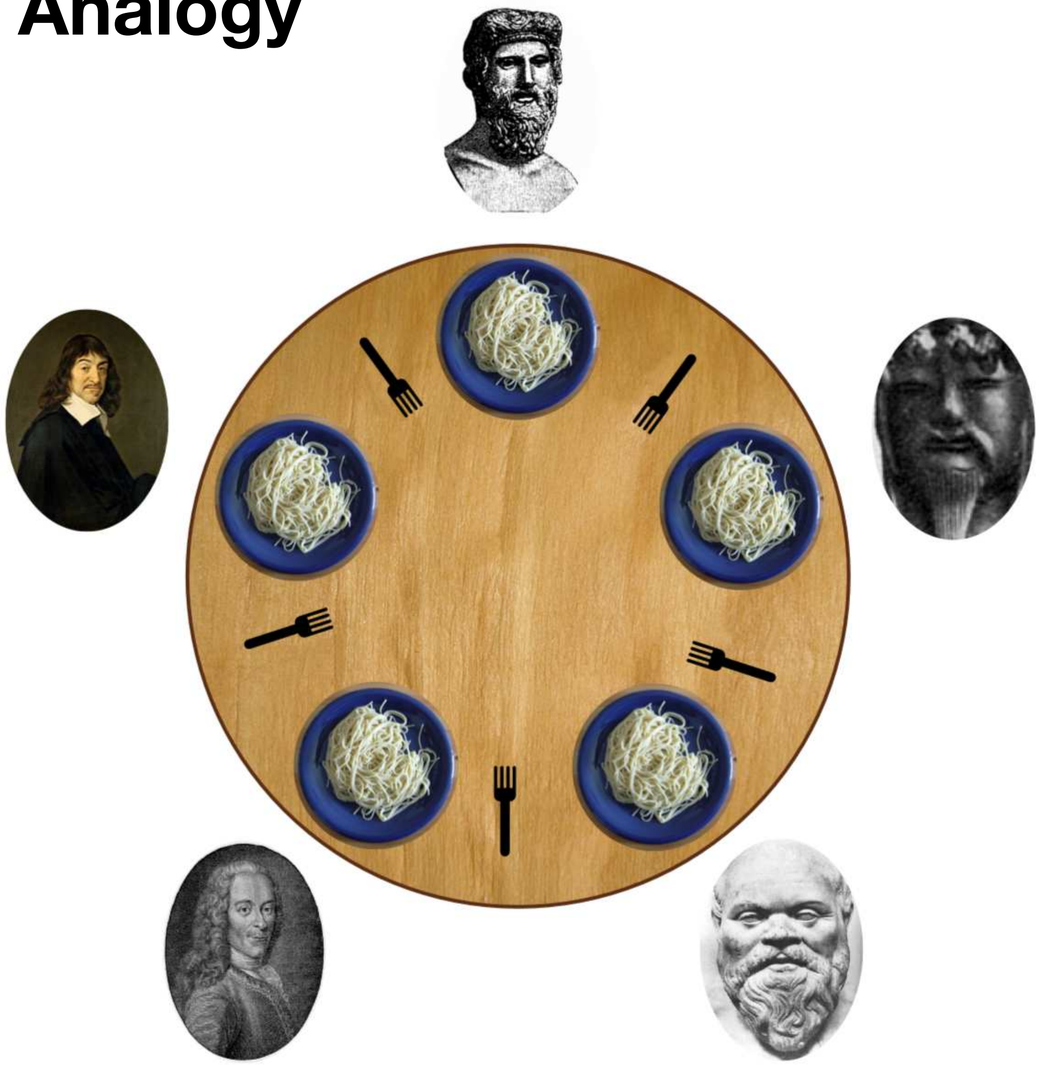
Readers Writers - Better solution

```
sem_t serviceQueue;

//reader code

sem_wait(&serviceQueue);
sem_wait(&rw->lock)

  ......


sem_post(&rw-> lock)
sem_post (&serviceQueue);
```

Reader

The Dinning Philosophers

An Analogy



- Five philosophers sit around a dinning table
- They think for sometime and eat spaghetti for sometime!
- There is one fork on the left and one on the right of each
- If they get two forks, then they can start eating, once done, they can keep it down
- How to solve it?

Image source: Wikipedia

Classic Problem: Dining Philosophers

```
while (1)
{
    think();
    get_forks(p);
    eat();
    put_forks(p);
}
```

- Each philosopher is a unique thread with an id (p = 0 to 4);
- Get forks and put forks needs to be written by ensuring there is no deadlock
  - is there a **possibility of deadlock?** Why?
  - Also **no philosopher should starve!**
- Can you think of implementing get_forks(p) and put_forks(p)?

Possible Solution

All semaphores initiated to 1

Dinning Philosophers Problem

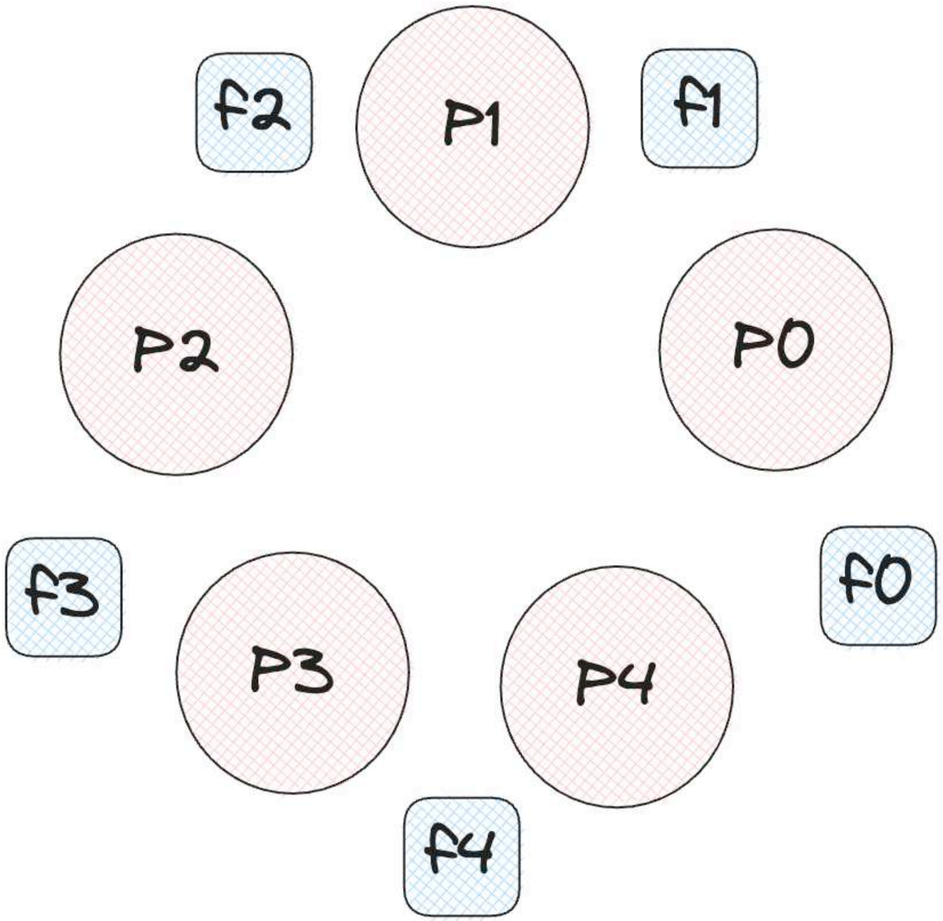
```
int left(int p)
{
  return p;
}

int right (int p)
{
  return (p+1)%5; //consider the
person on right
}
```

Dinning Philosophers Problem

```
sem_t forks[5]; //array of
semaphores, one for each fork

void get_forks(int p)
{
  sem_wait(&forks[left(p)])
  sem_wait(&forks[right(p)])
}

void put_forks(int p)
{
  sem_post(&forks[left(p)])
  sem_post(&forks[right(p])
}
```

Any issues here? **Deadlock!!, How?**

How deadlock happened?



- Each philosopher is one thread and they start running
- The first philosopher (0) has wait on 1, gets it (since initial semaphore is 1)
- Immidiately second philosopher (1) runs, wait on 0, but gets on right
- Third will run, waits on 2nd fork but gets the 4th one
- Fourth will run, waits on third but waits on 0
- **All philosophers wait for their left fork and we have a deadlock**

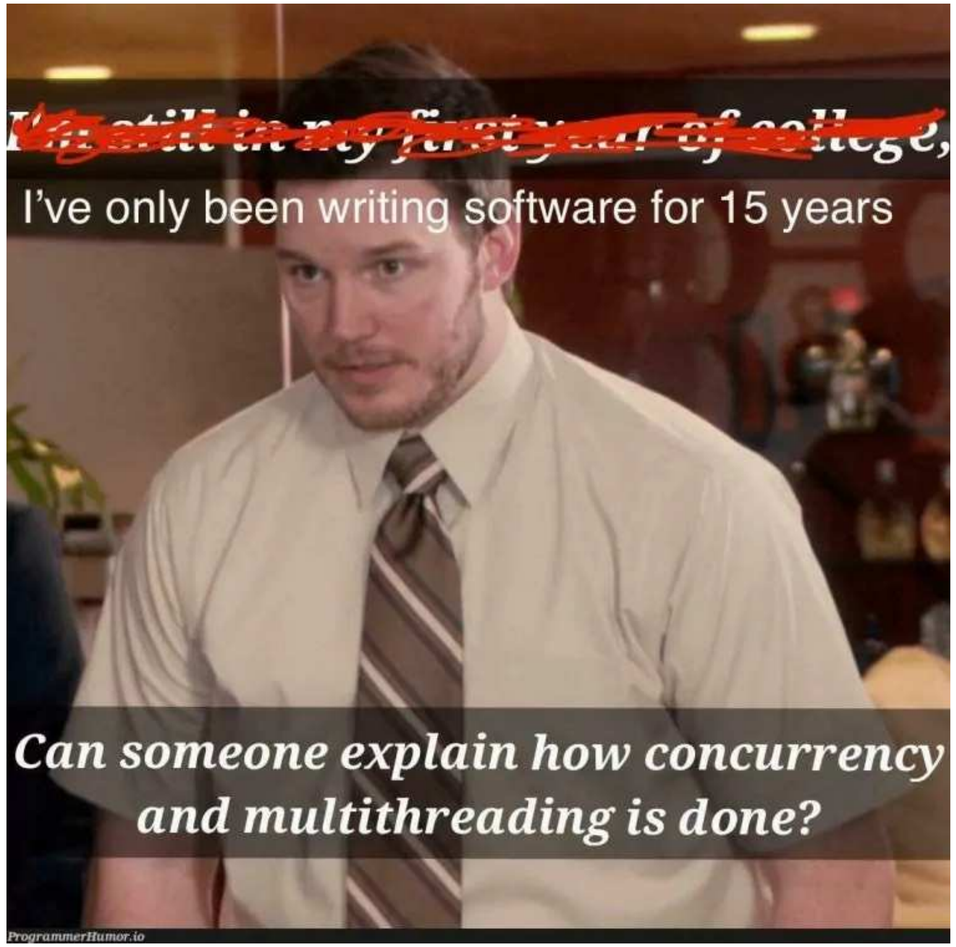
Possible Solution

Dinning Philosophers Problem

```
sem_t forks[5]; //array of
semaphores, one for each fork

void get_forks(int p)
{
  if (p==4)
  {
    sem_wait(&forks[right(p)])
    sem_wait(&forks[left(p)])
  }
  sem_wait(&forks[left(p)])
  sem_wait(&forks[right(p)])
}

void put_forks(int p)
{
  sem_post(&forks[left(p)])
  sem_post(&forks[right(p])
}
```

- Change the order in which they eat
- Philosopher 4 acquires the fork in a different order
- There won't be a situation in which one philosopher grabs one and has to wait for other
- **The cycle of waiting is broken**
- More solutions exist!

Concurrency Can be tricky!



There are some common **Concurrency Bugs** that can help identify some common bugs

Types of Bugs

- Bugs are very non-deterministic - Occurrence order cannot be fixed
- Two types of bugs
  - **Non-deadlock bugs**: Incorrect results when threads execute
  - **Deadlock bugs**: Threads keep waiting for each other

Application	Description	# of Bug Samples	
		Non-Deadlock	Deadlock
MySQL	Database Server	14	9
Apache	Web Server	13	4
Mozilla	Browser Suite	41	16
OpenOffice	Office Suite	6	2
Total		74	31

Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. 2008. **Learning from mistakes: a comprehensive study on real world concurrency bug characteristics,** ASPLOS, 2008

Source: https://pages.cs.wisc.edu/~shanlu/paper/asplos122-lu.pdf

Non-deadlock Bugs

Findings on Bug Patterns (Section 3)	**Implications**
(1) Almost all (97%) of the examined non-deadlock bugs belong to one of the *two simple bug patterns*: atomicity-violation or order-violation*.	Concurrency bug detection can focus on these two bug patterns to detect most concurrency bugs.
(2) About one third (32%) of the examined non-deadlock bugs are *order-violation bugs*, which are *not* well addressed in previous work.	*New* concurrency bug detection tools are needed to detect order-violation bugs, which are not addressed by existing atomicity violation or race detectors.

Non-deadlock bugs make the majority of the bugs among concurrency bugs

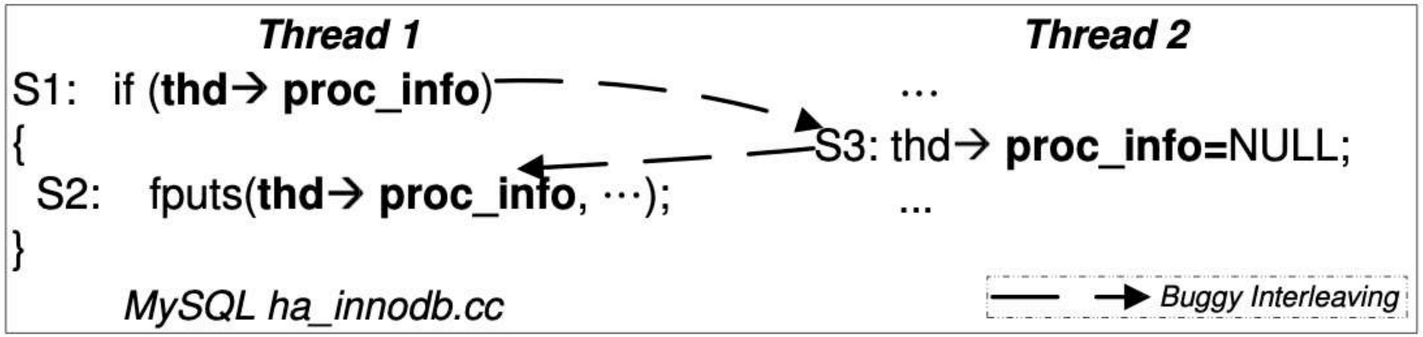
- Two types of non-deadlock bugs
  - Atomicity violation bugs
  - Order violation bugs

Atomicity Bugs

- Atomicity assumptions made during development are violated during execution of threads
- **Example:** From MySQL where one thread reads and modifies a shared variable while other tries to modify it

Atomicity Bug

```
Thread 1::

if (thd->proc_info)
  fputs(thd->proc_info,..);
....

Thread 2::
thd->proc_info = NULL;
```



How to go about solving it?

Atomicity Bugs

Use locks when accessing shared data

Atomicity Bug

```
pthread_mutex_lock_t thd_proc_info = PTHREAD_MUTEX_INITIALIZER;
Thread 1::

pthread_mutex_lock(&thd_proc_info);
if (thd->proc_info)
{
  ...
  fputs(thd->proc_info,..);
  ...
}
pthread_mutex_unlock(&thd_proc_info);
....


Thread 2::
pthread_mutex_lock(&thd_proc_info);
thd->proc_info = NULL;
pthread_mutex_unlock(&thd_proc_info);
```

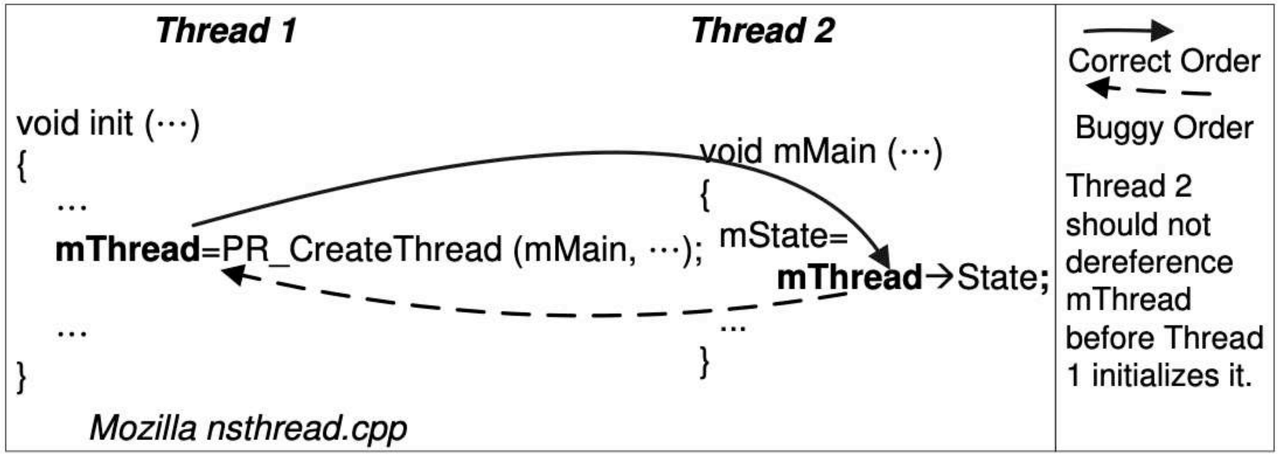
Order Violation Bugs

- Desired/assumed order of execution of memory access is violated during concurrent execution of threads

- **Example:** Assume thread 1 and thread 2. Thread 2 may assume that thread 1 has already run

Order Violation Bug

```
Thread 1::

void init(..)
{
  ...
  mThread = PR_CreateThread(mMain,...);
  ...
}


Thread 2::

void mMain(..)
{
  mState = mTrhead->state;
}
```



How to go about solving it?

Order Violation Bugs

Use condition variables or semaphores

Order Violation Bug - Solution

```
pthread_mutex_t mLock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  mCond  = PTHREAD_COND_INITIALIZER;
int mInit = 0;

Thread 1::

void init(..)
{
  ...
  mThread = PR_CreateThread(mMain,...);
  pthread_mutex_lock(&mLock);
  mInit = 1
  pthread_cond_signal(&mCond);
  pthread_mutex_unlock(&mLock);
  ...
}


Thread 2::

void mMain(..)
{
  pthread_mutex_lock(&mLock);
  while(mInit == 0)
    pthread_cond_wait(&mCond,&mLock);
  pthread_mutex_unlock(&mLock);
  mState = mTrhead->state;
  ....
}
```

- Use condition variables
  - Dependant thread can wait for dependency operation to be completed
  - Use combination of wait and signal
  - Semaphores can also be used here!
- **Remember:** Locks are still needed to handle the shared variable operation

Deadlock Bugs

(7) Almost all (97%) of the examined deadlock bugs involve two threads circularly waiting for at most *two resources*.	Pairwise testing on the acquisition/release sequences to two resources can expose most deadlock concurrency bugs, and reduce testing complexity.

Thread 1

```
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);
```

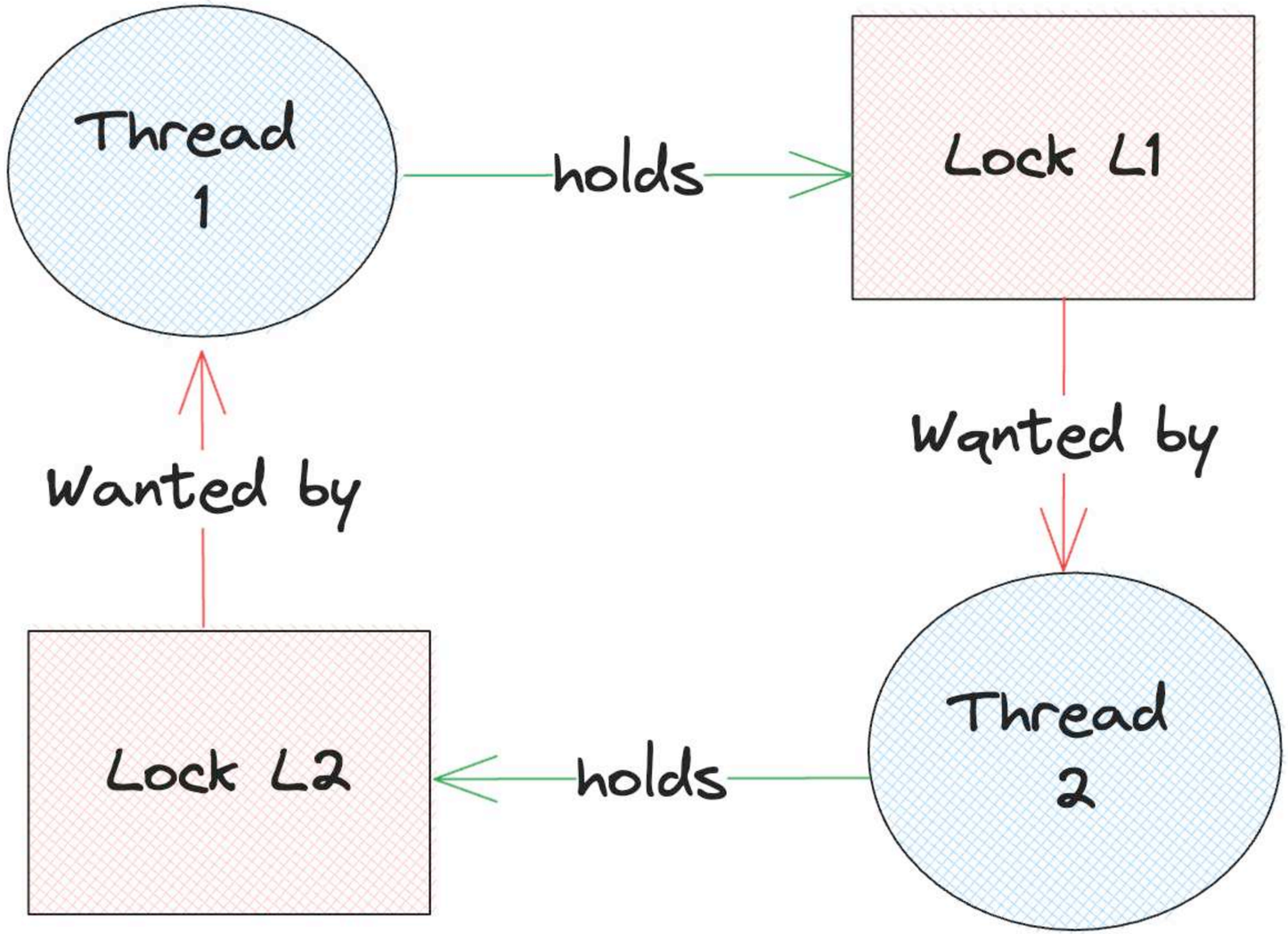
Thread 2

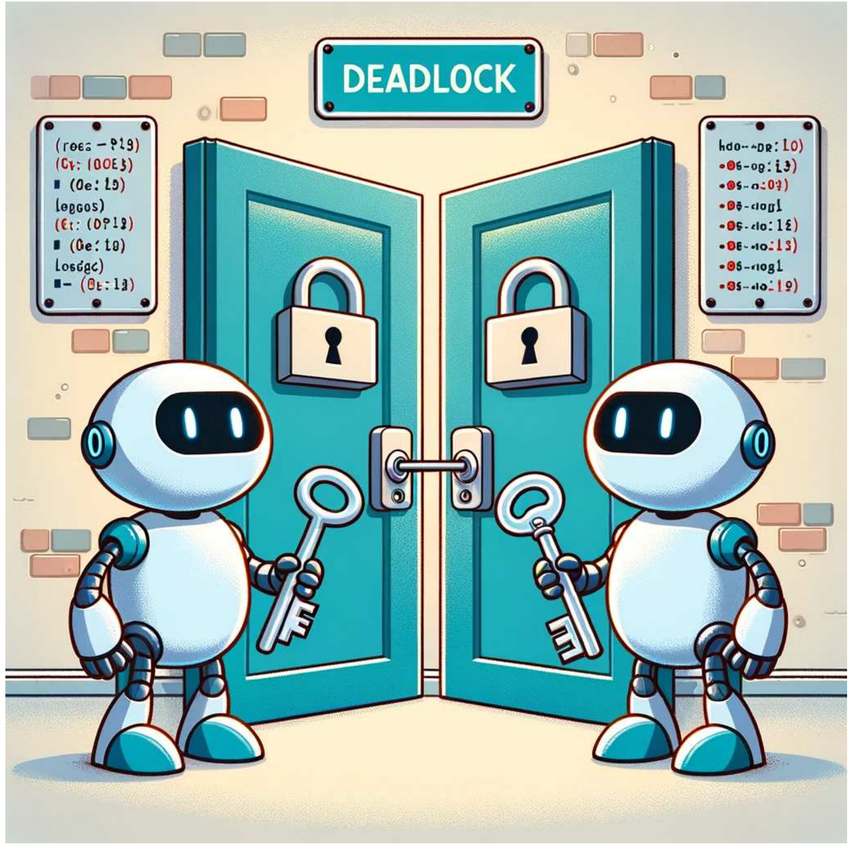
```
pthread_mutex_lock(L2);
pthread_mutex_lock(L1);
```

- Its not always the case that deadlock occurs
- If executions overlap and context switches from thread after acquiring one lock

Deadlock: A Visual Representation

Cycle in a dependency graph





Generated by DALLE-3

Conditions for deadlock

Four conditions should together hold for deadlock

- **Mutual Exclusion:** Thread claims exclusive control of a resource (eg: lock)
- **Hold-and-wait:** Thread holds a resource and is waiting for another
- **No Preemption:** Thread cannot be made to give up its resource (eg: cannot take back a lock)
- **Circular Wait:** There exists a cycle in the resource dependency graph

Prevention of Circular Wait

- Acquire locks in a particular order
  - Eg: Thread 1 and thread 2 acquires lock in the same order
- Provide a **total ordering** for lock acquisition
  - If there are only two locks, L1 and L2 => always acquire L1 before L2
- In more complex systems, more than two locks exist => **partial ordering**
  - Some locks can be given higher ordering than other locks
- Lock ordering can also by done using the address of the lock

```
if (m1 > m2)
pthread_mutex_lock(m1);
pthread_mutex_lock(m2);
} else {
pthread_mutex_lock(m2);
pthread_mutex_lock(m1); }
```

Preventing Hold-and-Wait

- Hold all the locks at once, atomically by acquiring a master lock first

```
pthread_mutex_lock(master);
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);
....
 ...
pthread_mutex_unlock(master);
```

- This may have an impact on concurrent execution and performance gains

“Trying” to get some Preemption Done

```
top:
    pthread_mutex_lock(L1);

    if (pthread_mutex_trylock(L2)≠ 0) {
        pthread_mutex_unlock(L1);
        goto top;
    }
```

- Thread can try for a lock before getting it - **pthread_mutex_trylock**
- Function returns 0 on successfully acquiring the lock
- If other thread also does in same order => **possibility of livelock**
- Periodic delay can be added to avoid live locking

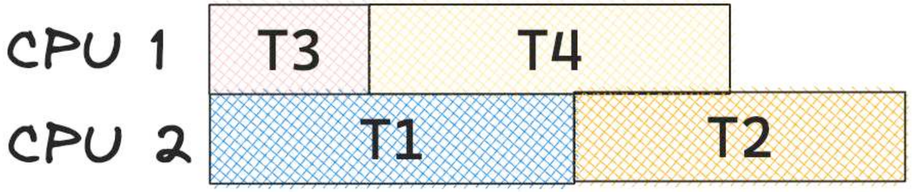
What about avoiding need for mutual exclusion?

- Not using any locks like pthread_locks or condition variables
- Using powerful hardware instructions
  - No need to do explicit locking
  - Hardware primitives like Compare-and-swap can be used
  - For instance, atomic incremental of shared value can be done using 1 line of compare and swap
- No lock, no deadlock but livelock is still a possibility

Deadlock Avoidance

- In some scenarios avoidance is preferable instead of prevention
- Deadlock avoidance via **Scheduling**
  - If OS knows which threads requires locks at which point of times, it can schedule them accordingly

	T1	T2	T3	T4
L1	yes	yes	no	no
L2	yes	yes	yes	no



- T1 and T2 are not run at the same time
- T1 and T3 do not share a lock
- Methods like **Bankers algorithm** by Dijkstra have been suggested but practically not applicable

Deadlock Avoidance

Detect and Recover

- Allow deadlocks to occur occasionally and take some action
  - If OS freezes, reboot the system
- Some systems like databases employ deadlock detection and recovery technique
  - Deadlock detector **runs periodically**
  - **Resource graph** is created to detect cycles
  - In the event of cycles, **restart the system**

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Networking - Link Layer and MAC Addressing

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

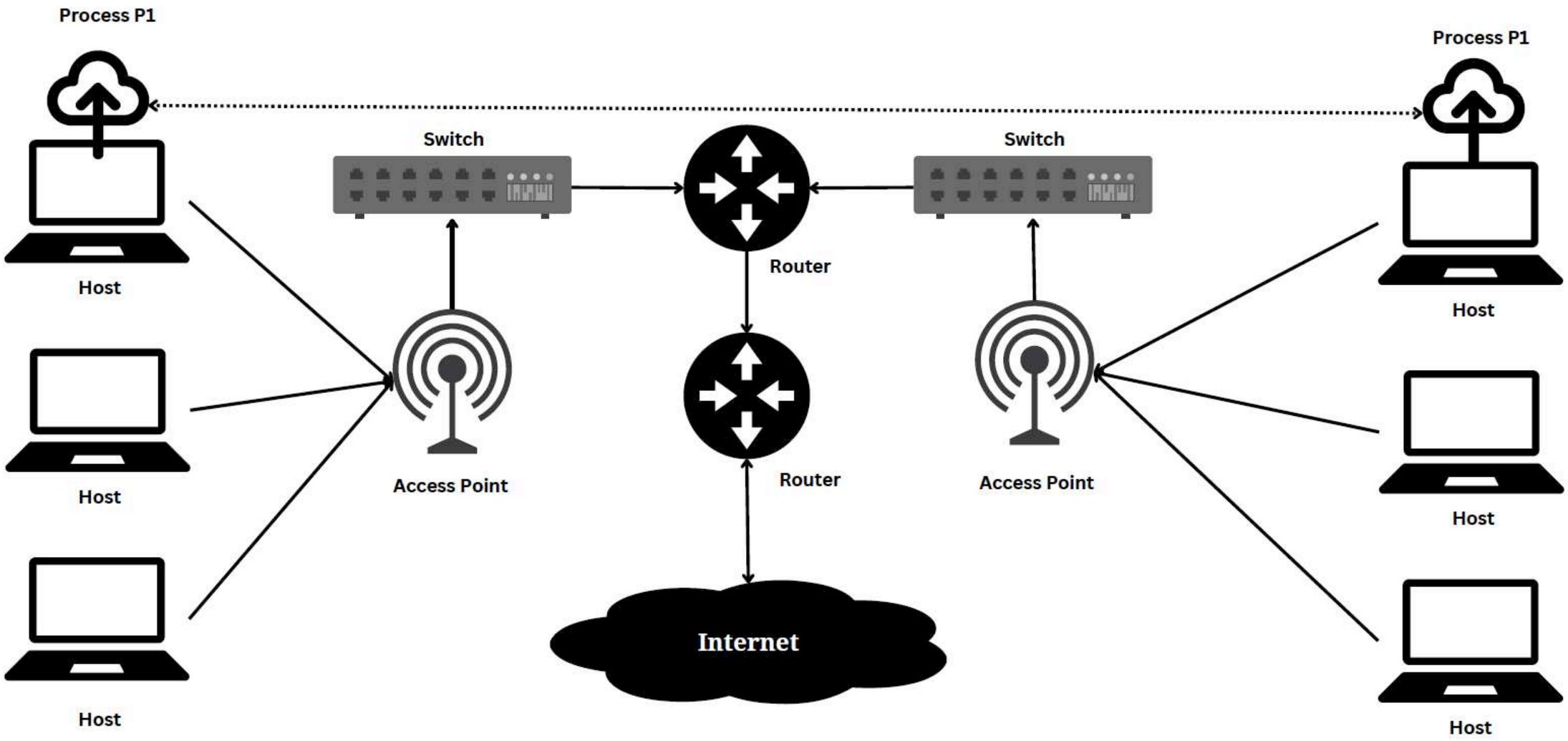
Acknowledgement

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:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

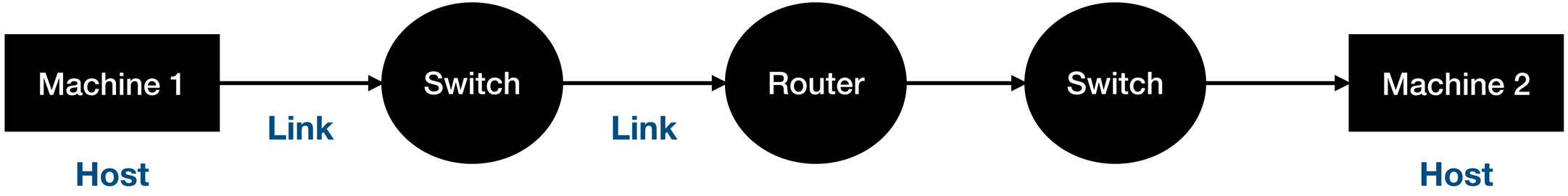
The Bigger Picture



How does host to host communication work?

What’s the contribution of the link layer?

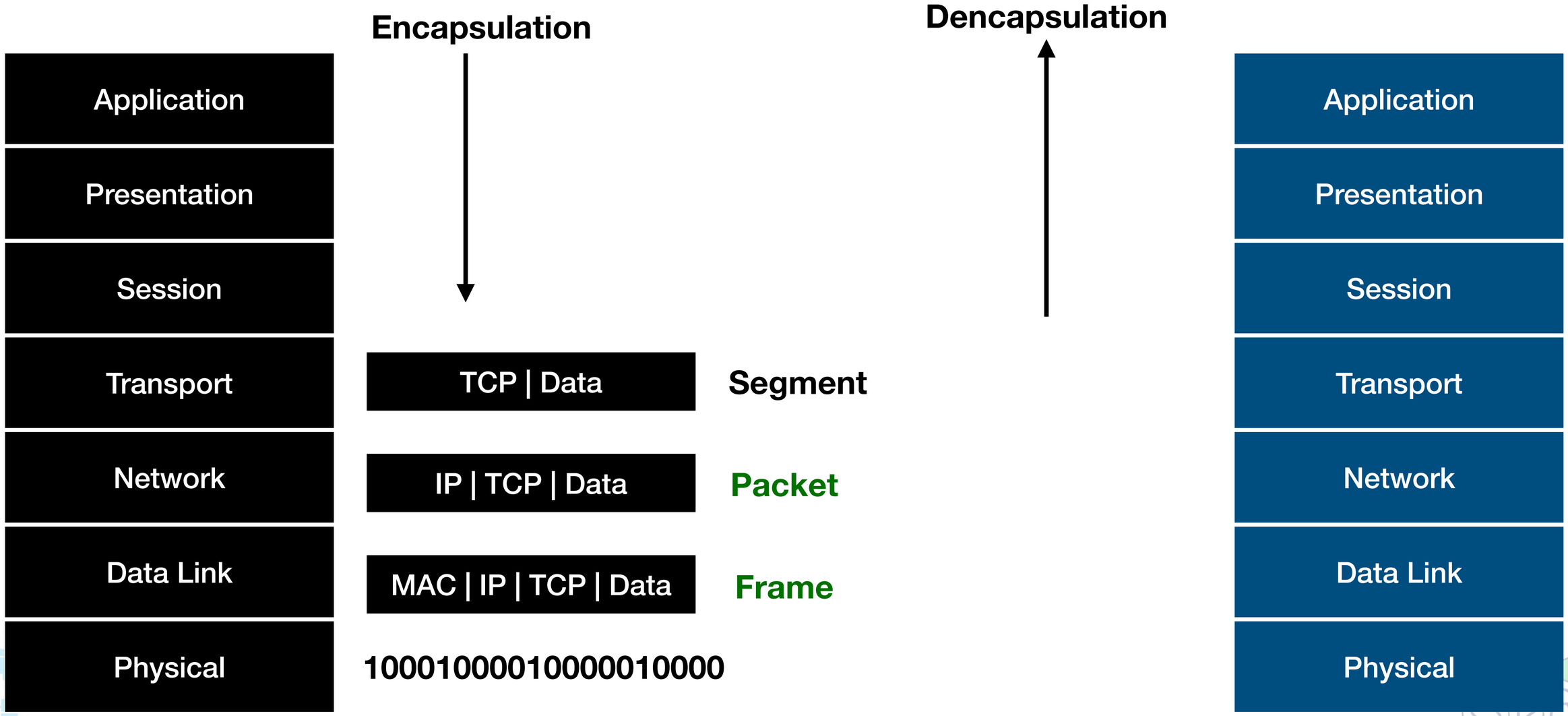
Remember the Components?



What we have seen so far

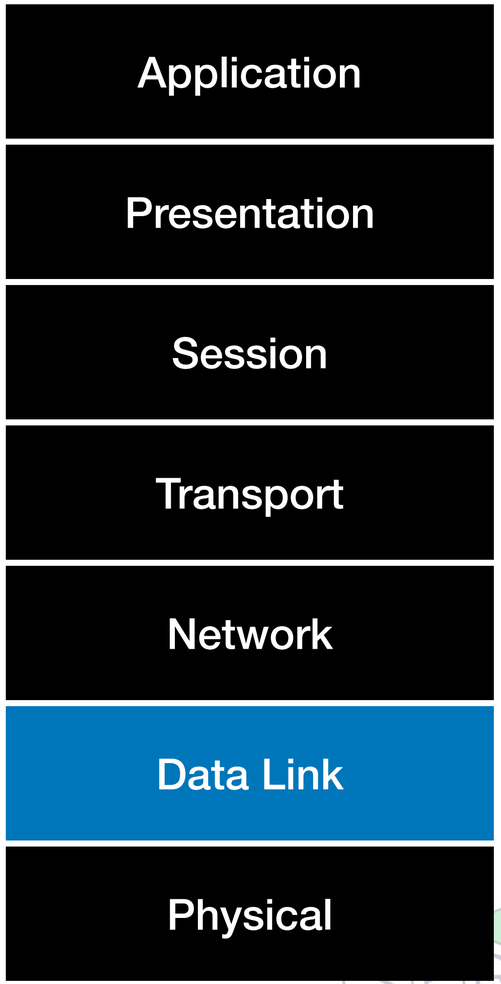
- **Application layer**
  - Provides support for end applications to format and manage data
  - HTTP, DNS, SMTP, etc.
  - In turn they make use of transport layer protocols
- **Transport layer**
  - Provides support for communication between services
  - TCP, UDP
  - Ports helps in identifying the right services/process
  - But transport layer by itself is not enough! - **Requires underlying support - Why?**

Putting It Together



Link Layer

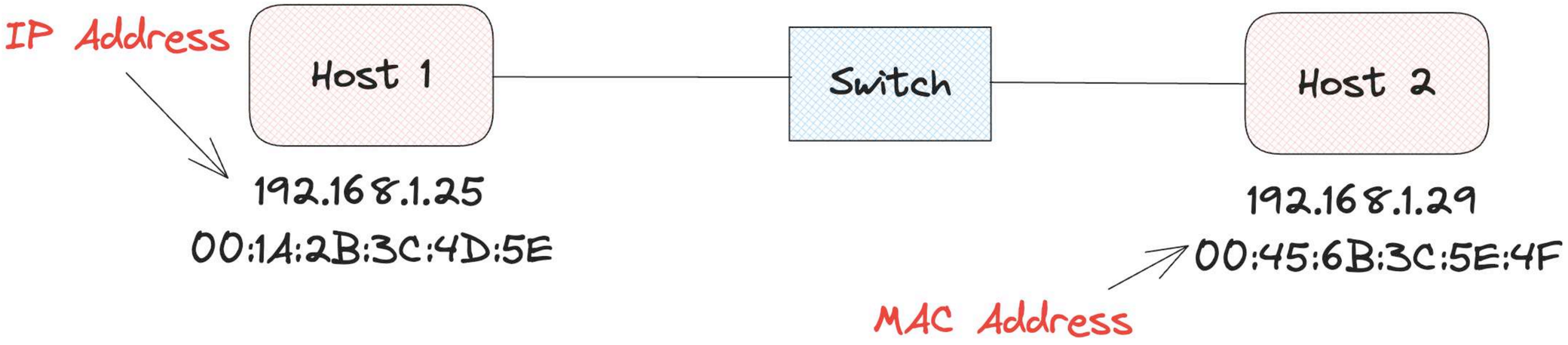
- Responsibility of transferring datagram from one node to a **physically adjacent node** over a link (no intermediate L3 routers)
- Supports **hop-to-hop** communication
- Ensures reliable connection link between two directly connected nodes (flow control, error correction and detection, etc.)
- Supported by **Media Access Control (MAC)** addressing
- Addressing scheme: MAC addressing (48 bit address, 12 hex digits, 6 bytes)
    - Eg: **00:1A:2B:3C:4D:5E**
    - First three identify manufacturer (IEEE)
    - Next three are assigned by manufacturer and should be unique



Two process want to communicate with each other

- Two main possibilities
  - Two processes are running in machines in the same network
  - Two processes are running in machines in two different networks
- Same network
  - Two machines are connected through switches
  - It can be a large intranet but the machines are still in same network

When machines are in same network

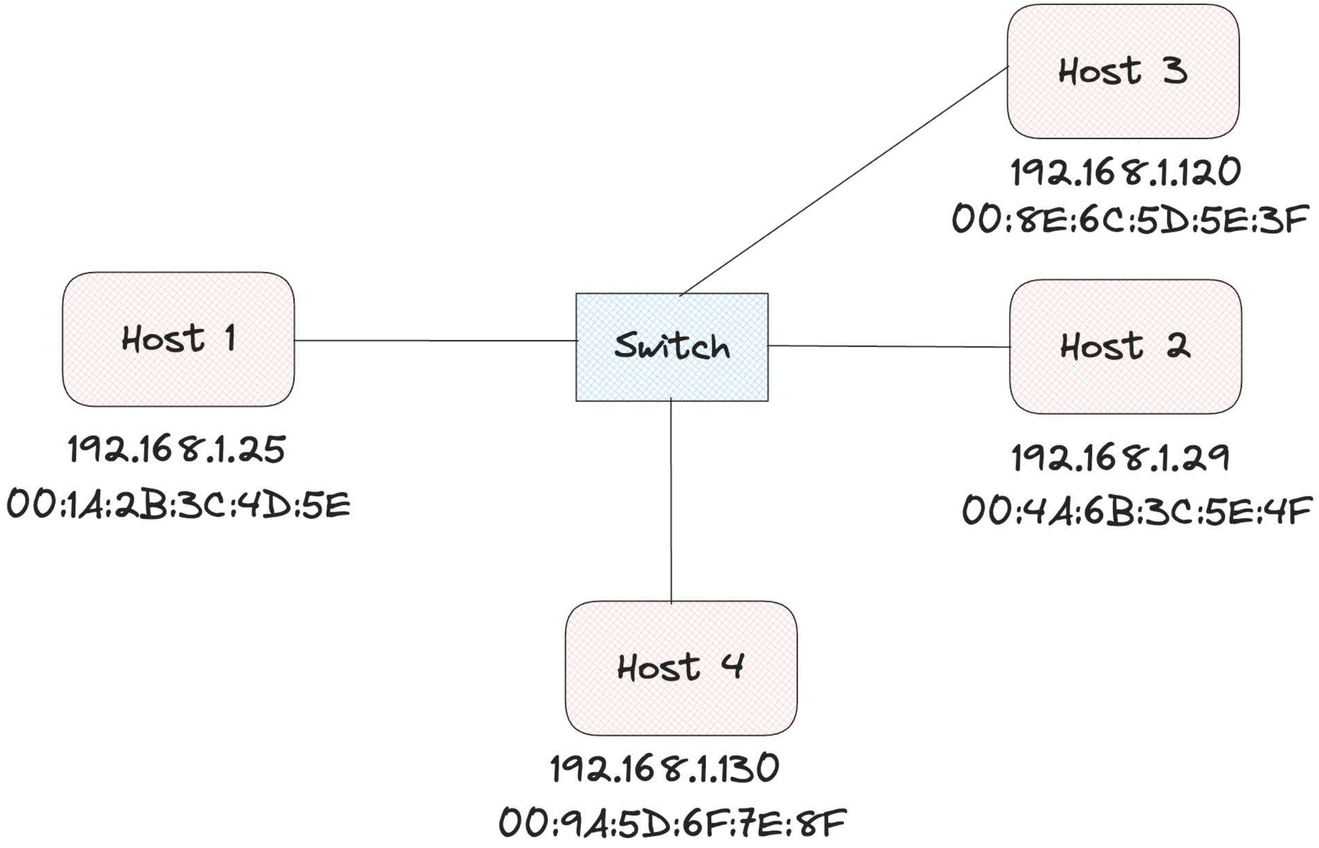


- Assume Host 1 wants to communicate to Host 2 known by domain host2.com
- Host 1 will know the IP address of Host 2 - **How?**
- IP address will help with L3 communication
- Will just the IP address suffice? What about L2? How to get the MAC address?

Why MAC and not just IP

- **Analogy:**
  - MAC address is like Aadhar card number
  - IP address is like Postal address
- **MAC address is unique to each interface**
  - Each interface on a LAN has globally unique MAC address
  - Each interface has a **locally unique IP address**
- How to determine interface's MAC address knowing its IP address?

ARP - Address Resolution Protocol

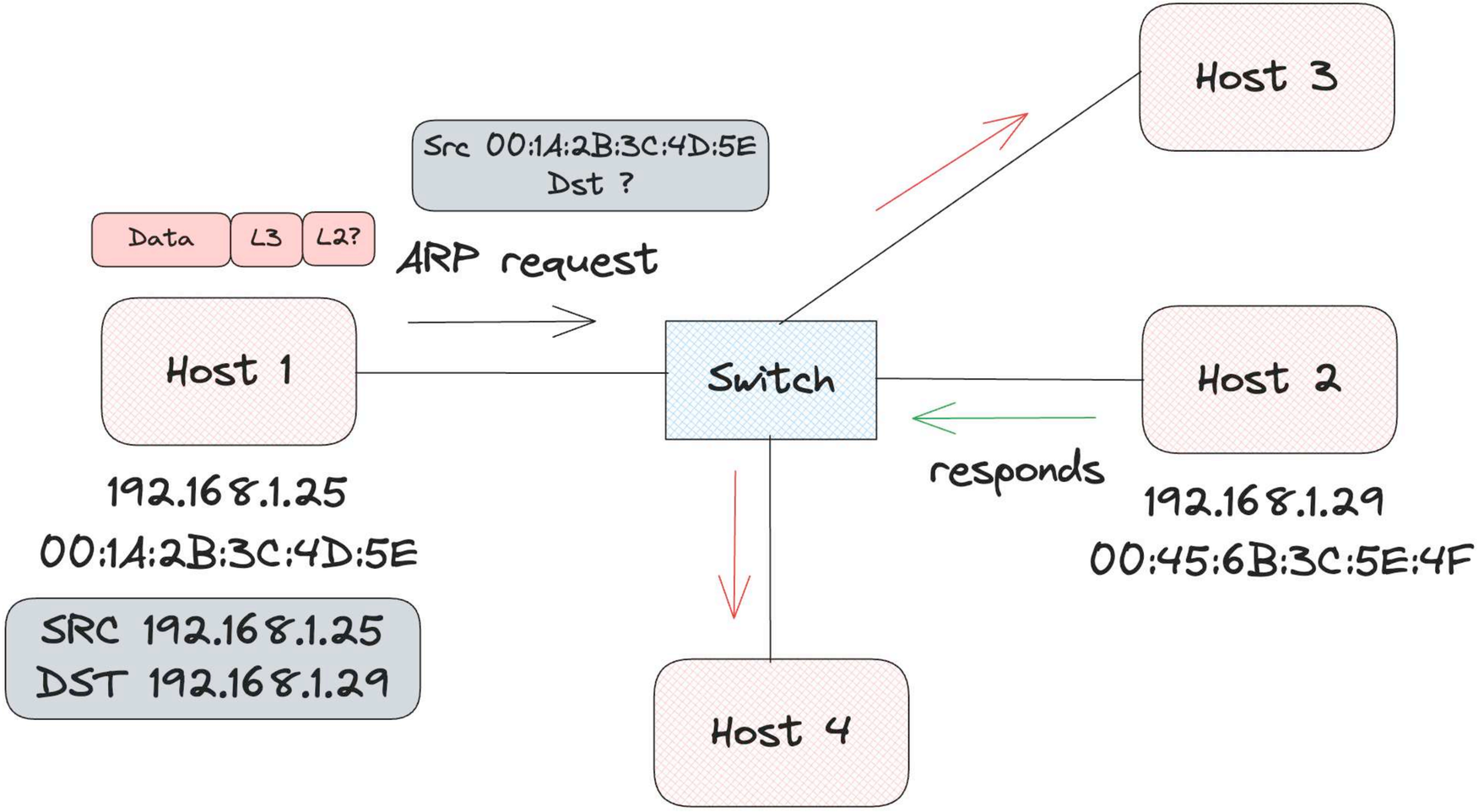


- Each IP node (router, host) on the LAN has a table - **ARP Table**
- IP/MAC address mappings for some LAN nodes
  - **<ip address, MAC address, TTL>**
- **TTL:** Time to live, time after which the mapping will be forgotten (20 mins)

ARP - Address Resolution Protocol

ARP Table in Host 1

IP	MAC	TTL



- Host 1 wants to send data to Host 2
- Sends ARP request to all the hosts in the network and updates table

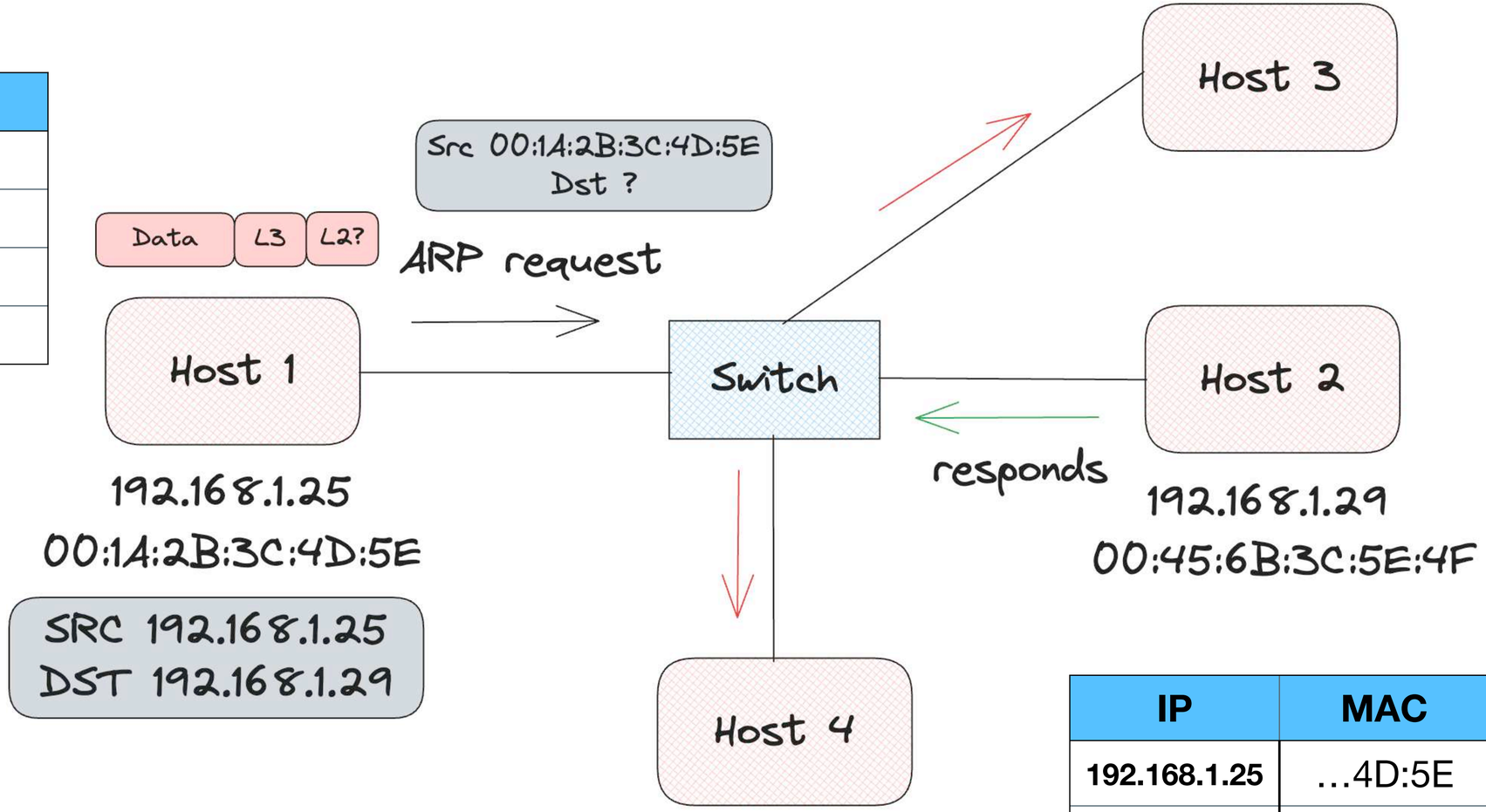
ARP Query

- When ARP query is sent initially, it is broadcast to all the nodes in the network
- The request includes senders IP address and MAC address
- It also includes the target IP address
  - Destination MAC is set as **FF:FF:FF:FF:FF:FF** (Reserved to send packet to all in the network)
- All the nodes will have an ARP cache or ARP table
  - It stores the mapping, when the initial request is send from one host, all other hosts stores the incoming mapping as well

ARP Table

IP	MAC	TTL
192.168.1.29	…:5E:4F	20

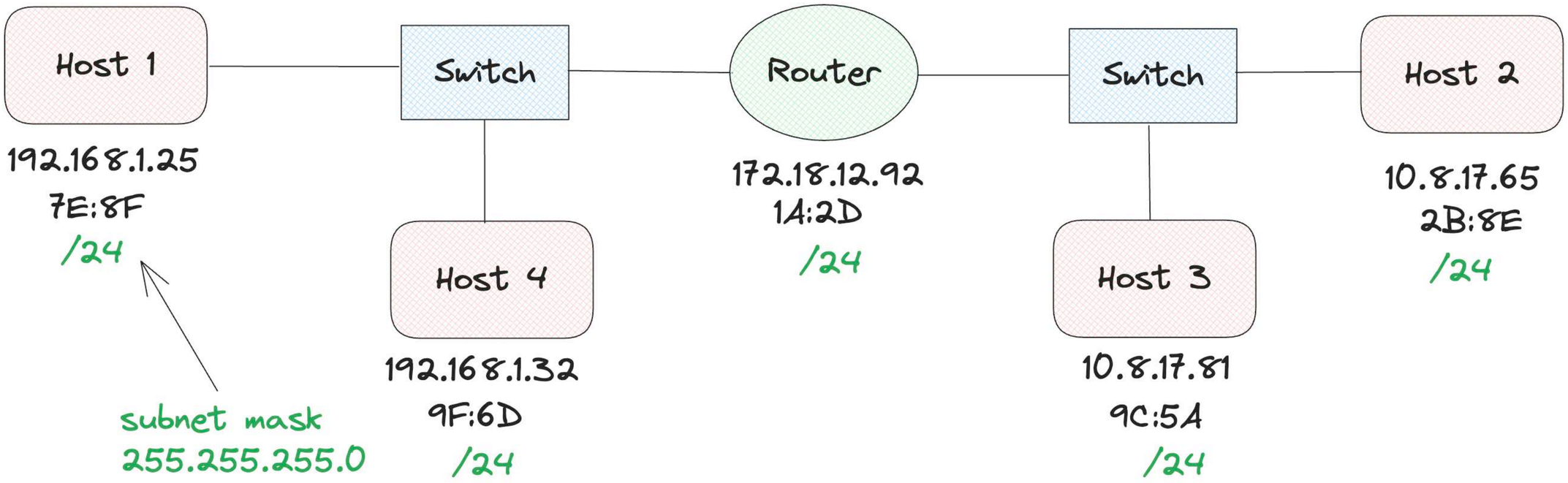
ARP Table at Host 1



IP	MAC	TTL
192.168.1.25	…4D:5E	20

Host 2 is saving the MAC of Host 1 request in its ARP table

When Machines are in Different Network

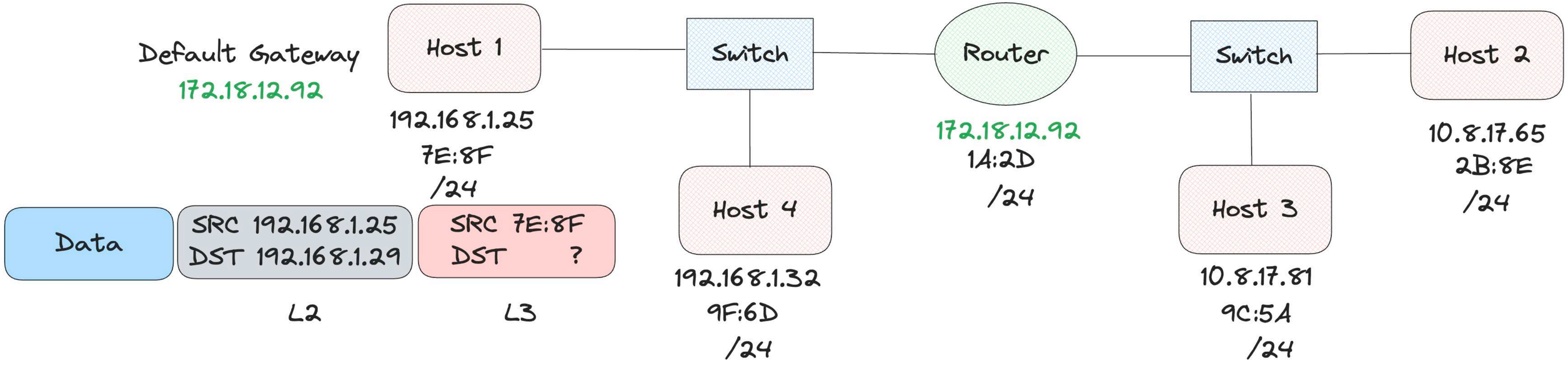


- Host 1 wants to communicate to Host 2 - Knows the IP address
- Host 1 also knows that Host 2 is not in the same network? - **How?**
  - /24 - Subnet mask, what do we mean by that?
  - Router **bridges** the two different networks

Subnets

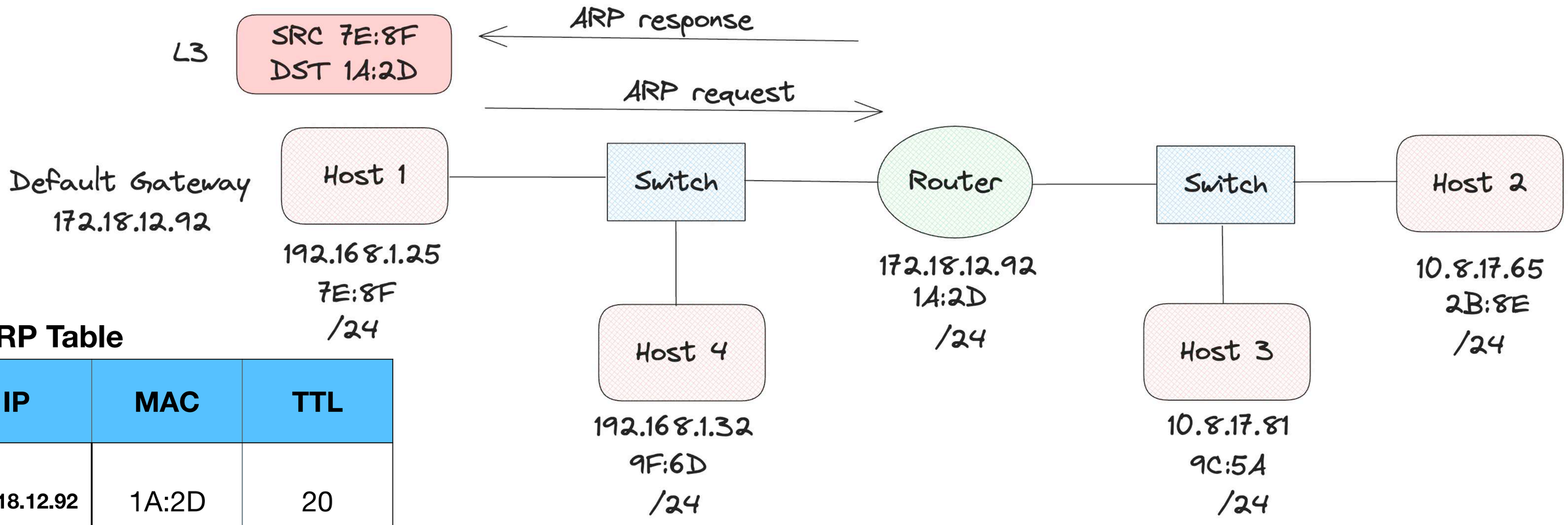
- Practice of dividing a network into one or more networks is subnetting
- Allows to create hierarchy within an organisation
  - Think about 172.18.21.x (country.organization.department.machine)
  - Another set of IP within organisations 172.18.y.x
- Consider an IP address 172.18.21.0 with a subnet mask of 255.255.255.0 or /24
  - Implies one network that can contain 254 host addresses (only the last one can change)
  - /24 - CIDR Notation (Classless Inter domain routing) - Number of 1s in the address
  - **What about subnet mask 255.255.0.0? Or /16?**

When Machines are in Different Network



- Host 1 does not know the destination MAC address
- It need not broadcast to all MAC in the same network - **Why?**
- It can make use of the default gateway - IP address of the router

When Machines are in Different Network



ARP Table

IP	MAC	TTL
172.18.12.92	1A:2D	20

- The ARP process needs to happen only once, since router is the gateway
- First step - Check if the IP of the receiver is in the same or different network
- If different network => Send ARP to gateway else, send ARP to all nodes in the network (FF:FF……:FF)

How does end-to-end communication work?

What is the role of the network layer?

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Networking - Network Layer

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

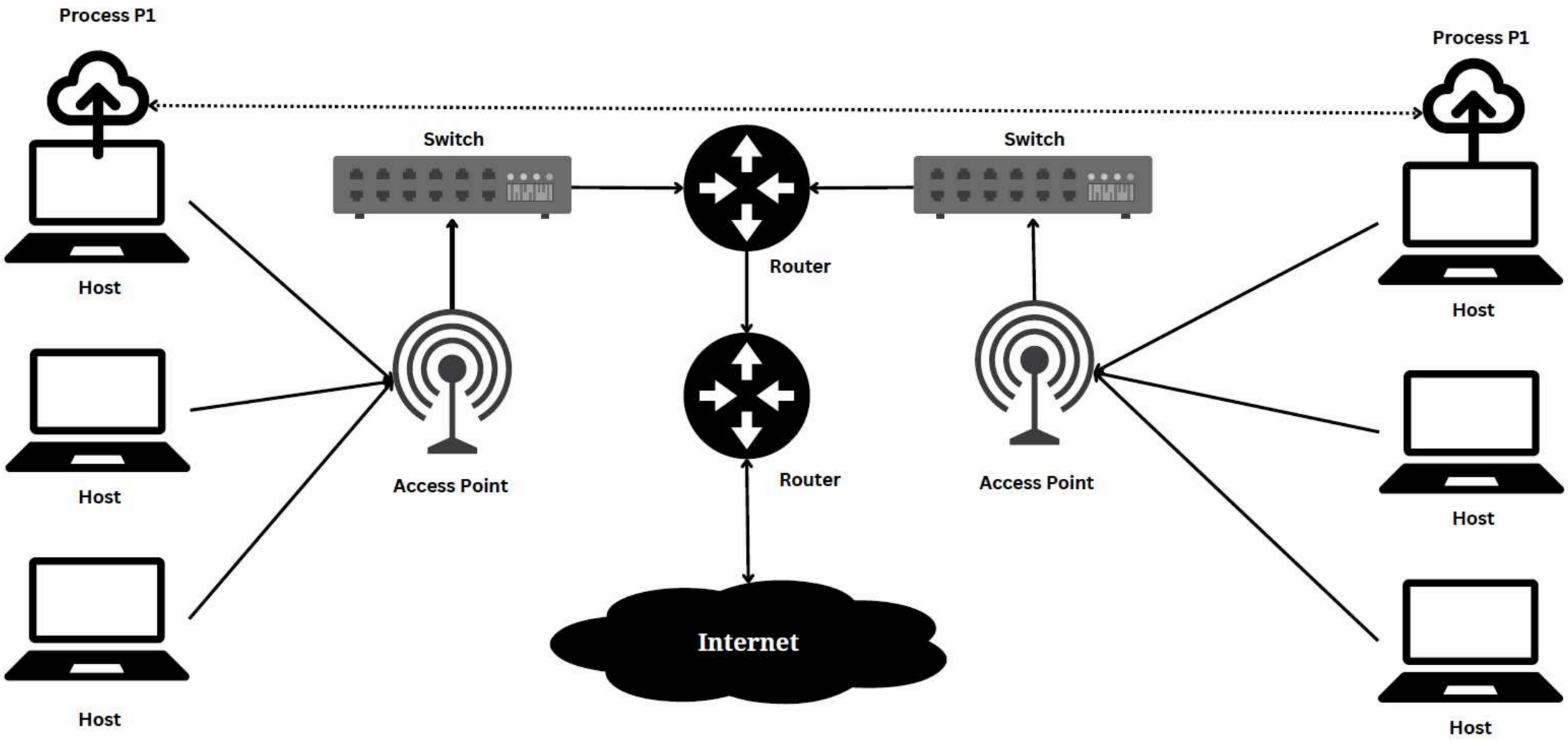
Acknowledgement

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

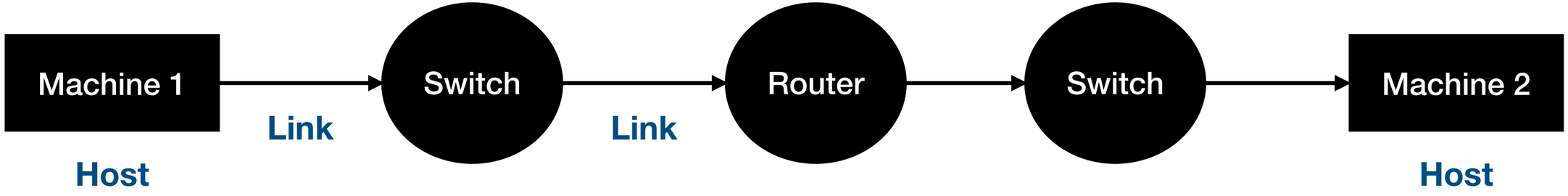
Sources:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

The Bigger Picture



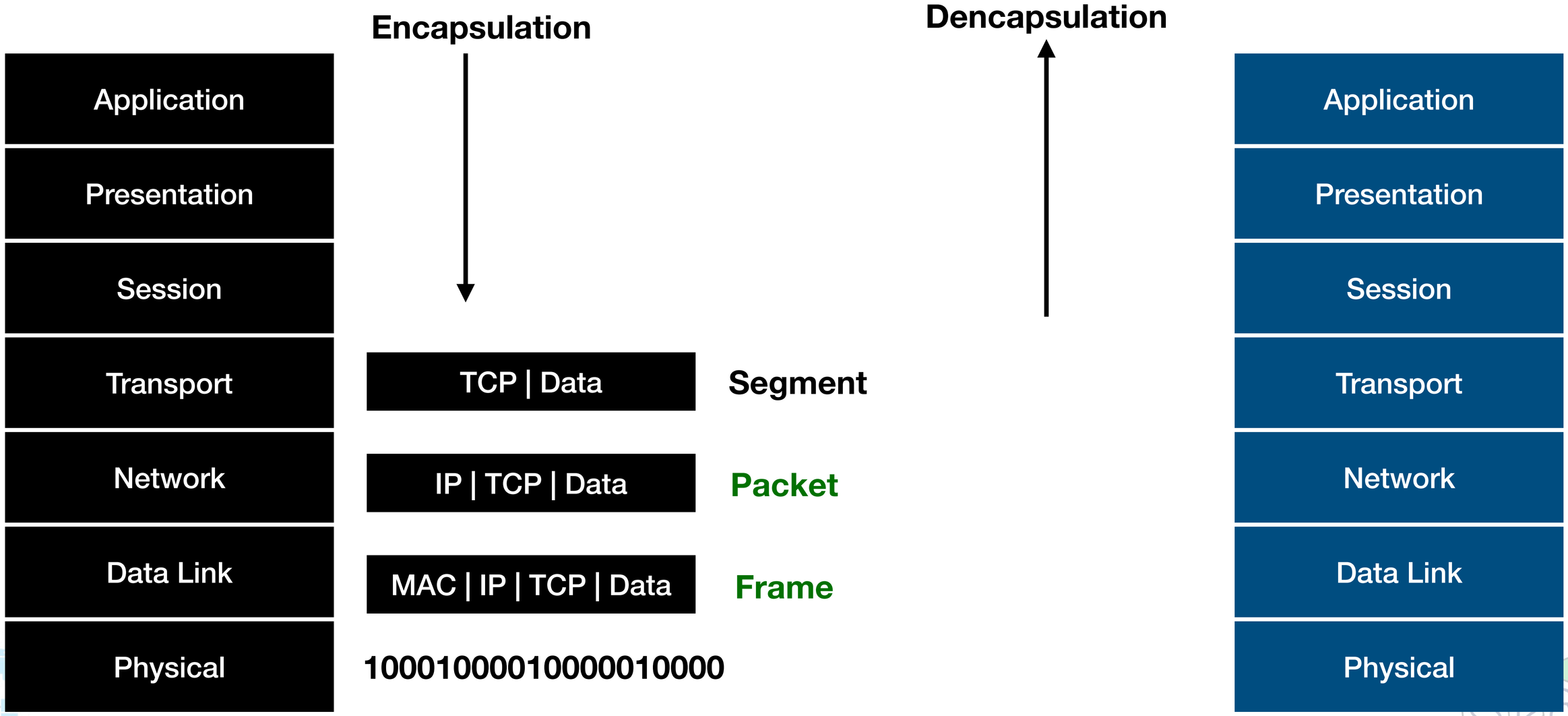
Remember the Components?



What we have seen so far

- **Application layer**
  - Provides support for end applications to format and manage data
  - HTTP, DNS, SMTP, etc.
  - In turn they make use of transport layer protocols
- **Transport layer**
  - Provides support for communication between services
  - TCP, UDP
  - Ports helps in identifying the right services/process
  - But transport layer by itself is not enough! - **Requires underlying support - Why?**

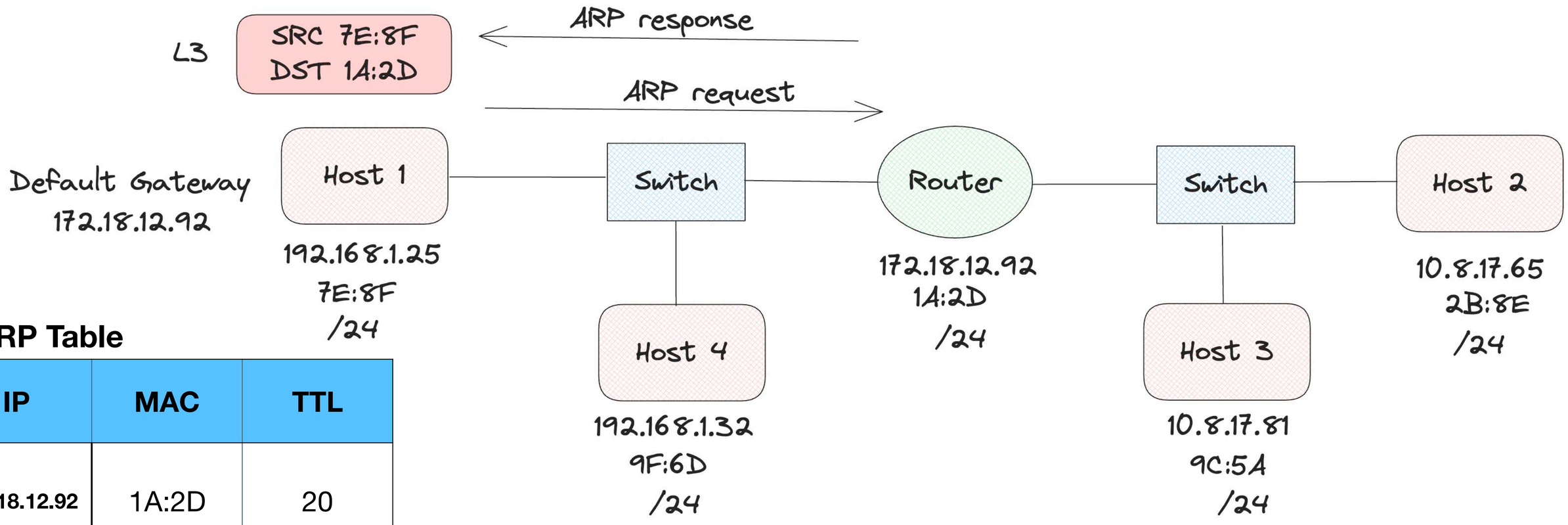
Putting It Together



Subnets

- Practice of dividing a network into one or more networks is subnetting
- Allows to create hierarchy within an organisation
  - Think about 172.18.21.x (country.organization.department.machine)
  - Another set of IP within organisations 172.18.y.x
- Consider an IP address 172.18.21.0 with a subnet mask of 255.255.255.0 or /24
  - Implies one network that can contain 254 host addresses (only the last one can change)
  - /24 - CIDR Notation (Classless Inter domain routing) - Number of 1s in the address
  - **What about subnet mask 255.255.0.0? Or /16?**

Link Layer Working



ARP Table

IP	MAC	TTL
172.18.12.92	1A:2D	20

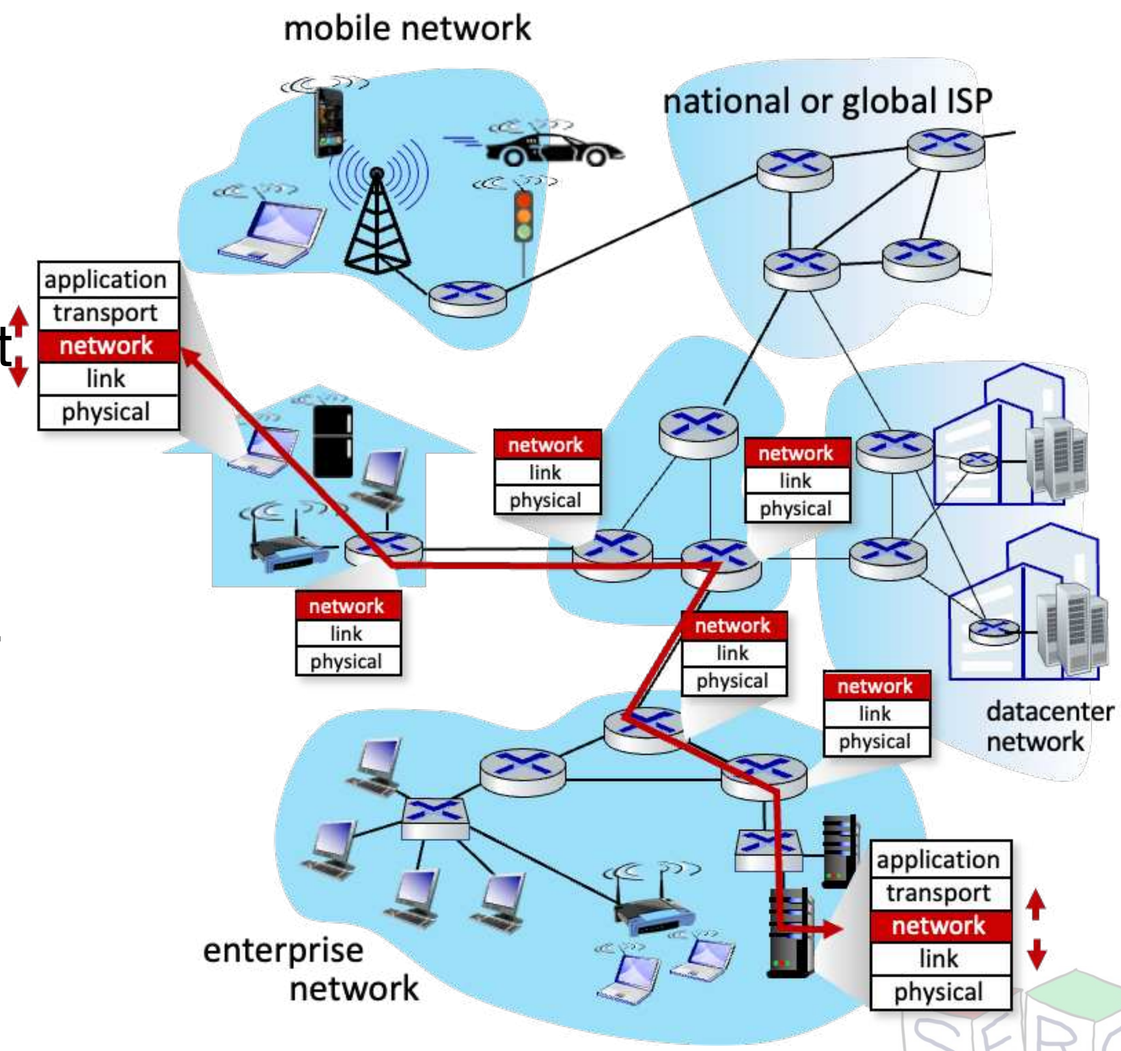
- The ARP process needs to happen only once, since router is the gateway
- First step - Check if the IP of the receiver is in the same or different network
- If different network => Send ARP to gateway else, send ARP to all nodes in the network (FF:FF……:FF)

How does end-to-end communication work?

What is the role of the network layer?

Network Layer - Functionalities

- Plays key role in end-to-end communication
  - Link layer is concerned about just hop to hop
- Transport segment from sending to receiving host
  - **Sender:** Encapsulates segments into datagrams, passes to Link layer
  - **Receiver:** Delivers segments to transport layer protocol
- Network layer protocols in every internet device
  - Hosts and routers



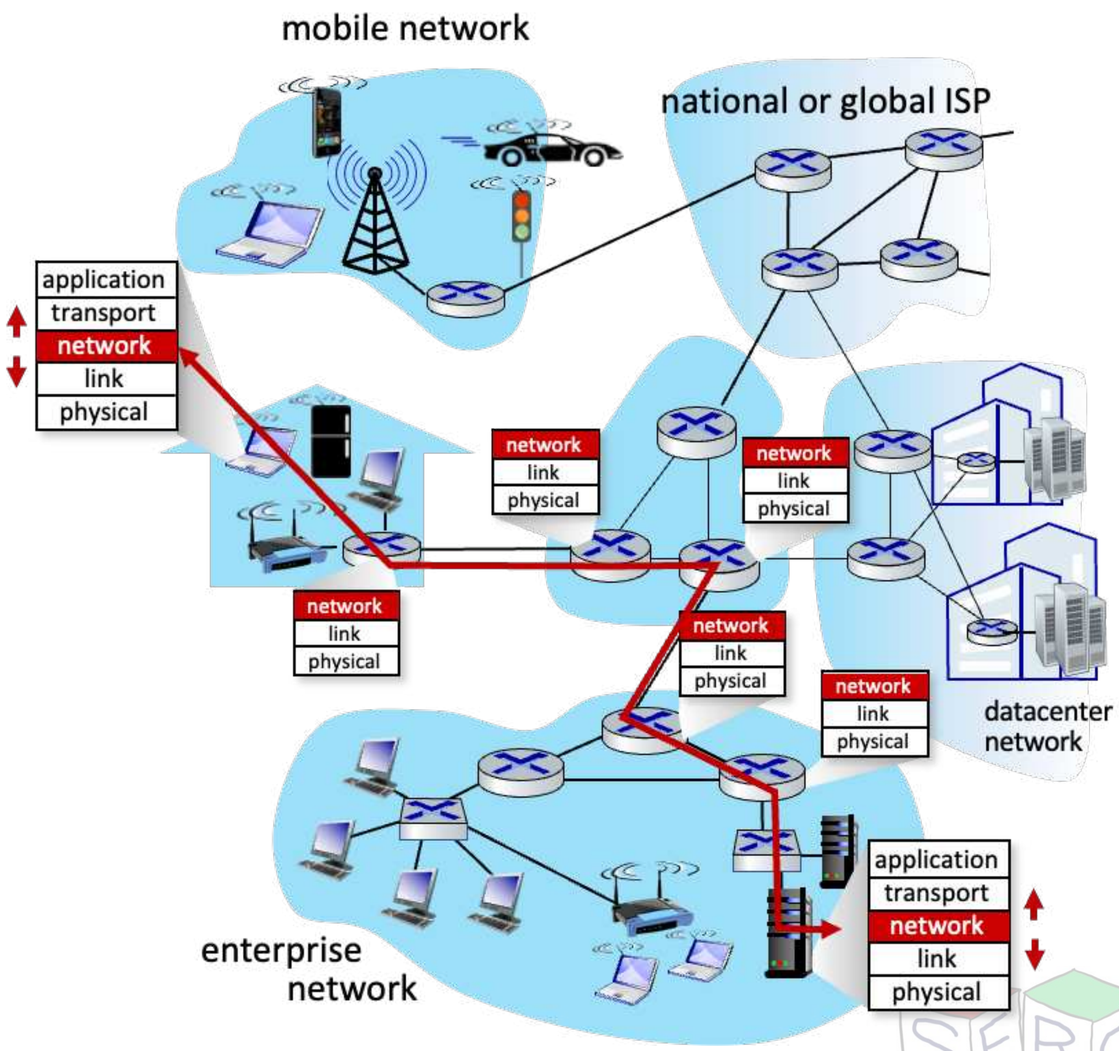
Network Layer - Functionalities

Addressing

- Devices in network are assigned logical address for unique identification - IP
- Network layer uses IP to forward packets to the intended destinations

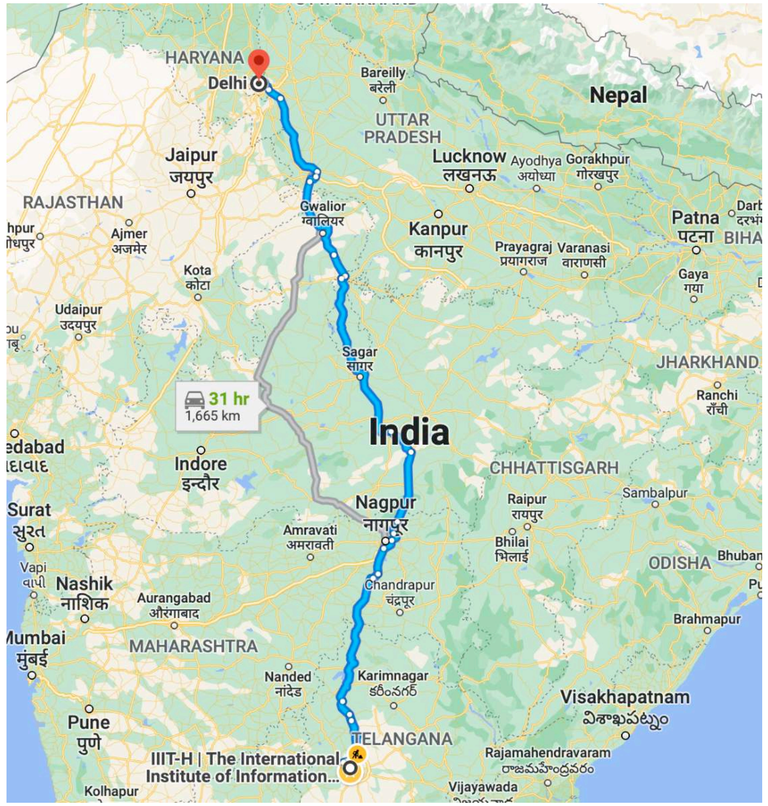
Route Determination

- Identifies best path for packets to reach to destination
- This process is dynamic and changes based on network conditions



Two Key Network Layer Functions

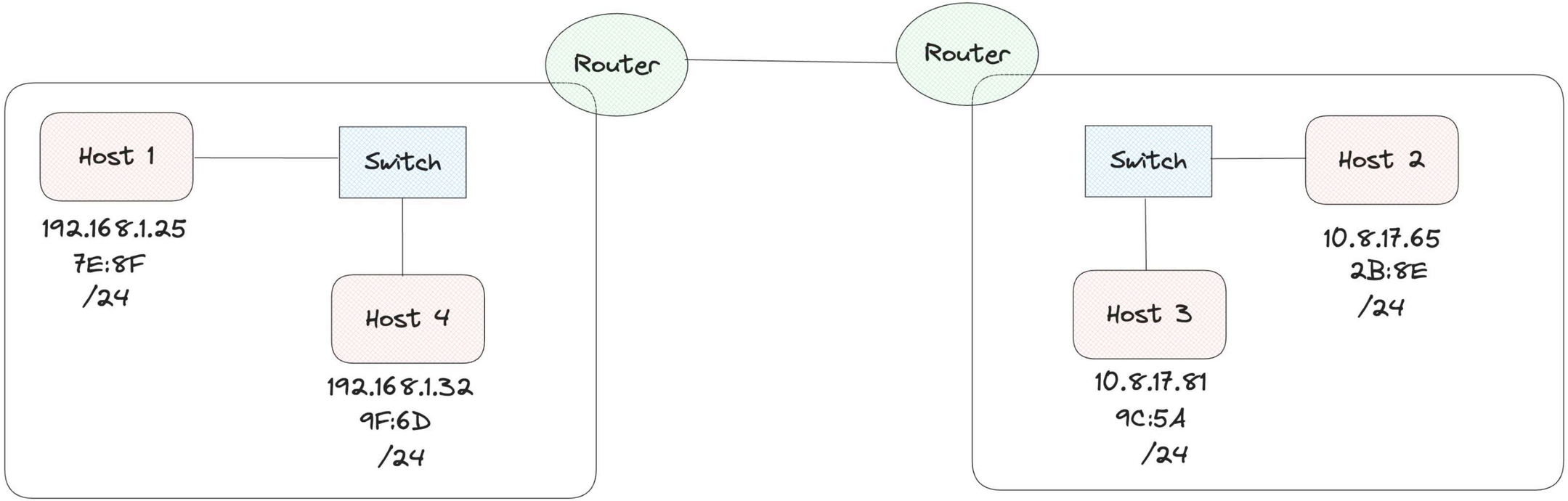
Forwarding (Interchanges)



Routing (Source to destination route)

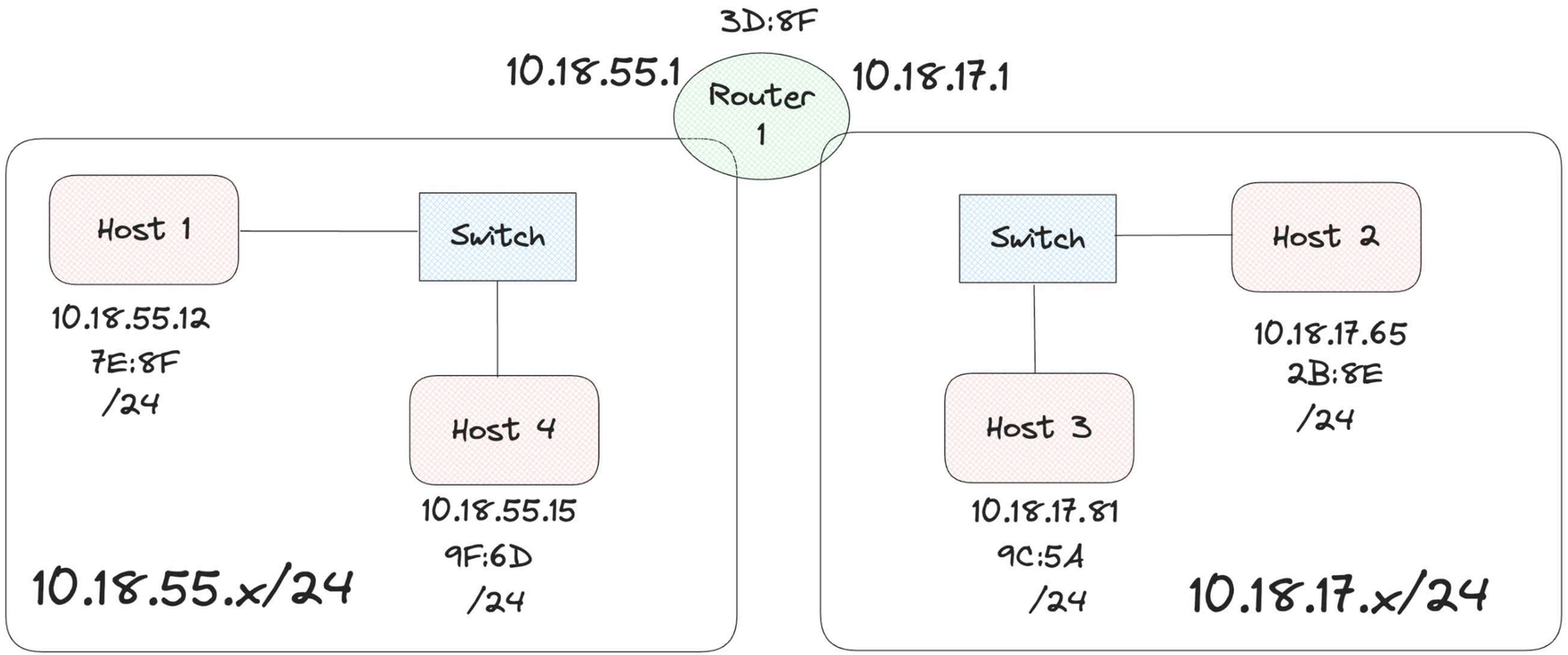
- **Forwarding**
  - Move packets from routers input link to output link
- **Routing**
  - Determine route to be taken by packets from source to destination

Routers - Devices in L3 that makes things happen



- Routers are connected to a network (have IP and MAC)
- Routers are node that **forwards packets** not explicitly addressed to itself
- Hosts are any nodes that are not a router (RFC 2460, IPV6) - They can discard packets

More about Routers

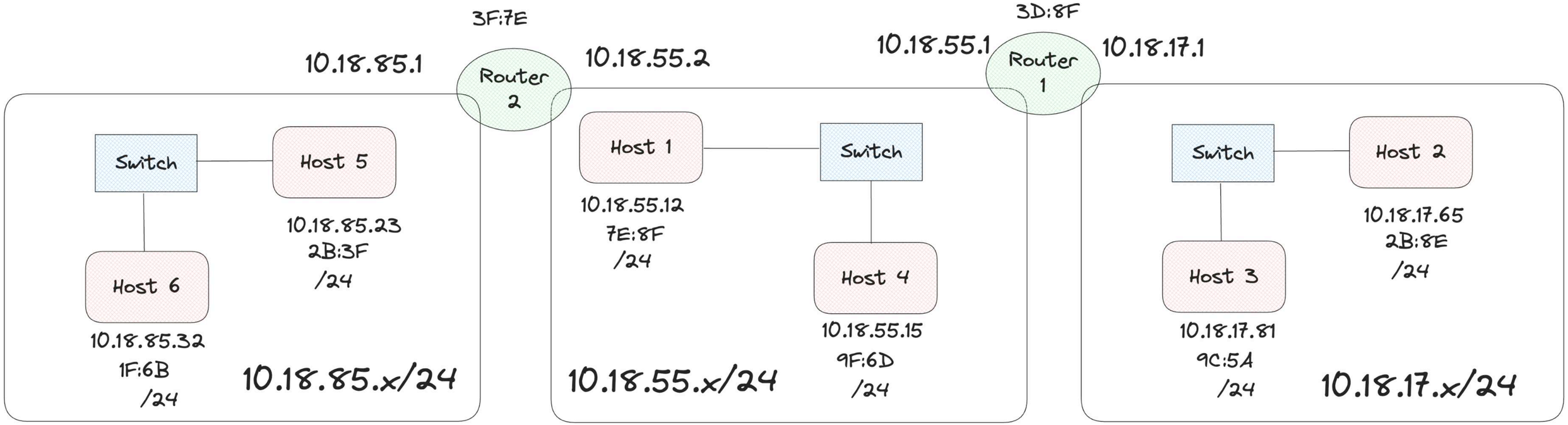


Router 1 routing table

Destination	Interface
10.18.55.x/24	Left
10.18.17.x/24	Right

- Routers maintain a map of all networks they know about
  - **Routing Table**: Used by routers as a map to connect to the networks they know about given the destination IP
  - **Note** the table is just a sample, in reality instead of left and right it can be eth/0, eth/1, etc.

Simple Example



Destination	Interface
10.18.85.x/24	Left
10.18.55.x/24	Right

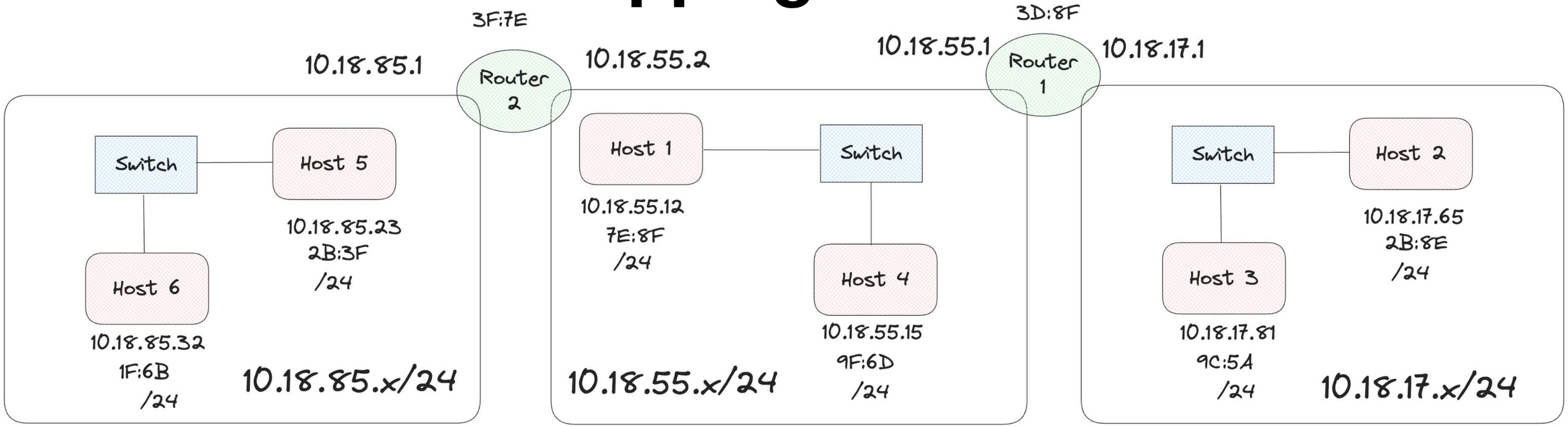
Router 2 routing table

Destination	Interface
10.18.55.x/24	Left
10.18.17.x/24	Right

Router 1 routing table

How can Host 6 communicate with Host 2?

Admin can add mappings to table!



Type	Destination	Interface
DC	**10.18.85.x/24**	Left
DC	**10.18.55.x/24**	Right
Static	**10.18.17.x/24**	10.18.55.1

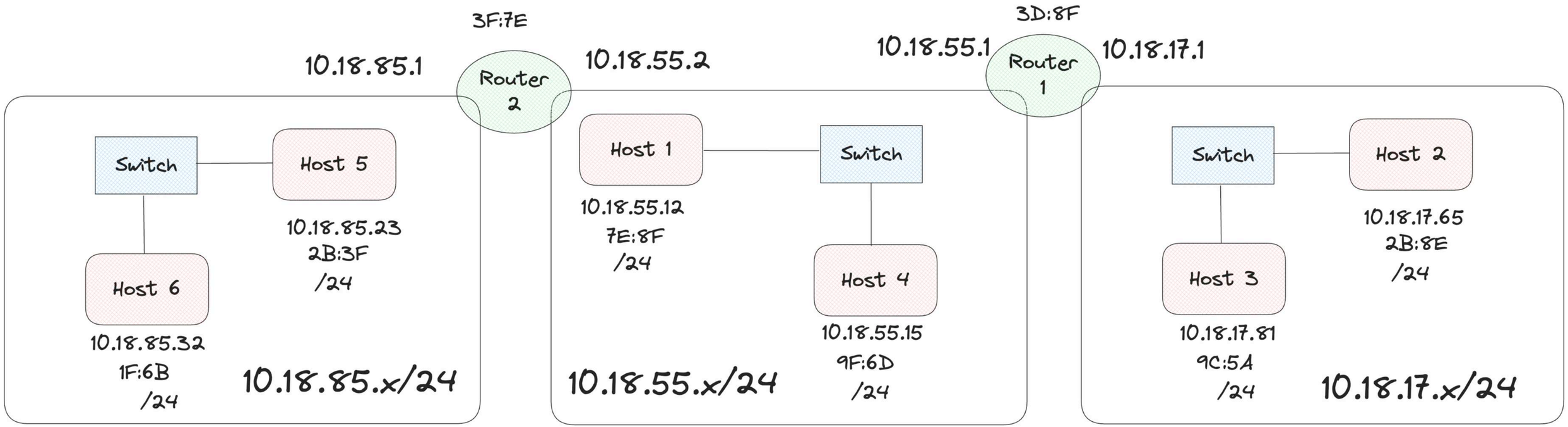
Router 2 routing table

Type	Destination	Interface
DC	**10.18.55.x/24**	Left
DC	**10.18.17.x/24**	Right
Static	**10.18.85.x/24**	10.18.55.2

Router 1 routing table

Admin can add it!!

What if Routers can learn by themselves?



Type	Destination	Interface
DC	**10.18.85.x/24**	Left
DC	**10.18.55.x/24**	Right
Dyn	**10.18.17.x/24**	10.18.55.1

Router 2 routing table

Type	Destination	Interface
DC	**10.18.55.x/24**	Left
DC	**10.18.17.x/24**	Right
Dyn	**10.18.85.x/24**	10.18.55.2

Router 1 routing table

Dynamic discovery And addition

But how to send data to the host

- Routers have an IP and MAC
- Routers have routing tables - Map to every network
- Routers also have ARP tables
  - Mapping of L3 address to L2 address
  - Anything in network with IP will have an ARP table
  - ARP table is populated on the fly - Why?
  - Routing tables needs to be ready apriori - Routers may drop packets if IP is not known

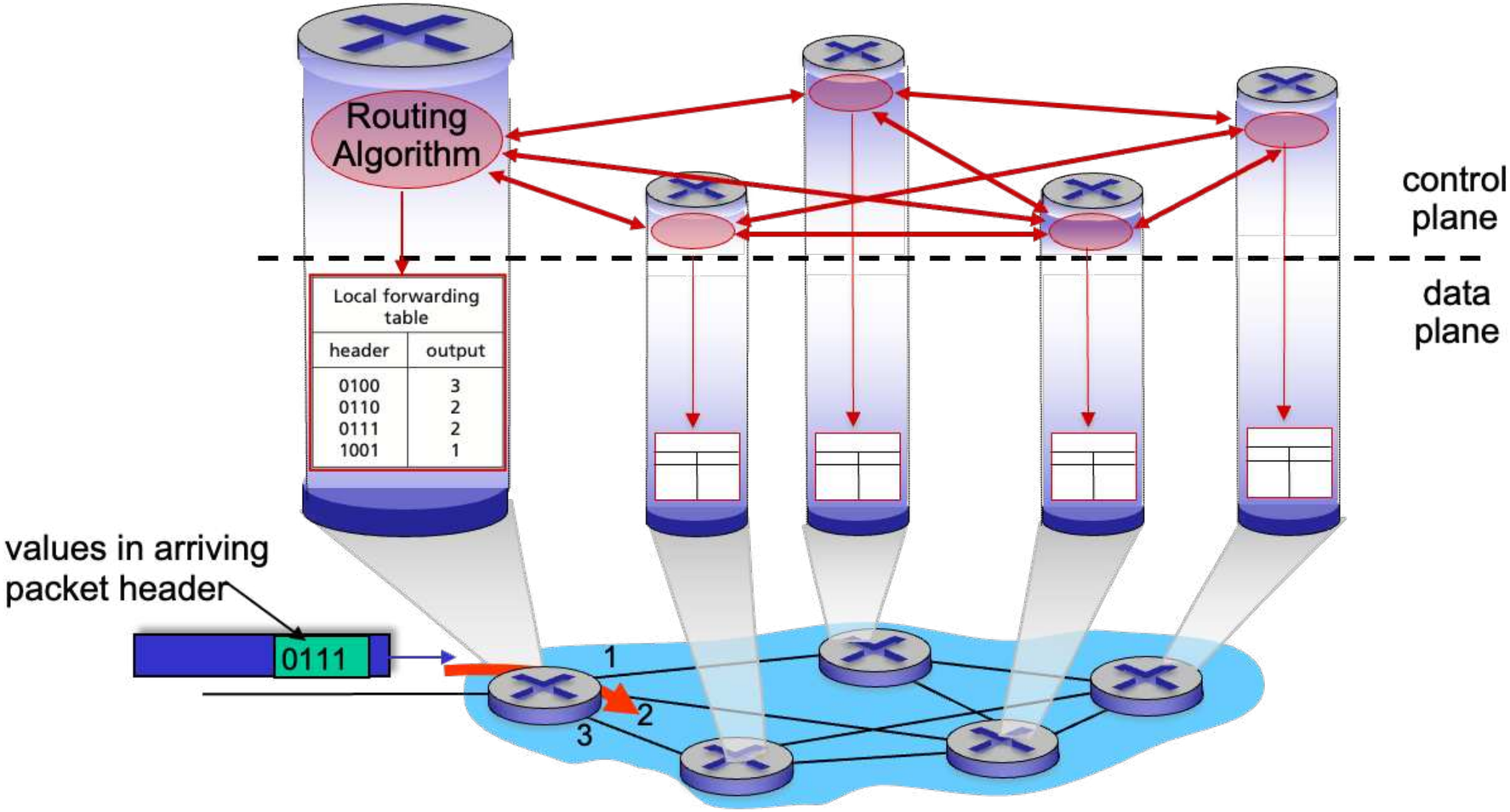
Routing Table

- Three methods to populate routing table
  - **Directly connected:** Networks to which the router is directly attached to
  - **Static routes:** Routes manually provided by an administrator
  - **Dynamic routes:** Routes automatically learned from other routers
    - Routers communicate with each other to know about different networks
    - Different protocols: OSPF, BGP, EIGRP, IS-IS
    - Used by routers to inform about the different networks they are connected to

Network Layer: Data Plane and Control Plane

- One can divide network layer functions into two planes: Data and control
- **Data plane:** Local per router function
  - Determines how datagram arriving on router input port is forwarded to router output port
- **Control plane:** Network wide logic
  - Determine how datagram is routed along end to end path form source to destination
  - Two approaches: Traditional routing algorithms, or Software defined networking (SDN)

Traditional Control Plane Approach



How does one router know whom to send to?

- Routers have forwarding table consisting of routes
- But there are **billions of destinations** - Not everything can be stored in each router!!!
- Sending so many links with each other can itself bring down the network
- There are two parts to it:
  - Internet: network of networks
  - Each network admin may want to control routing in its own network

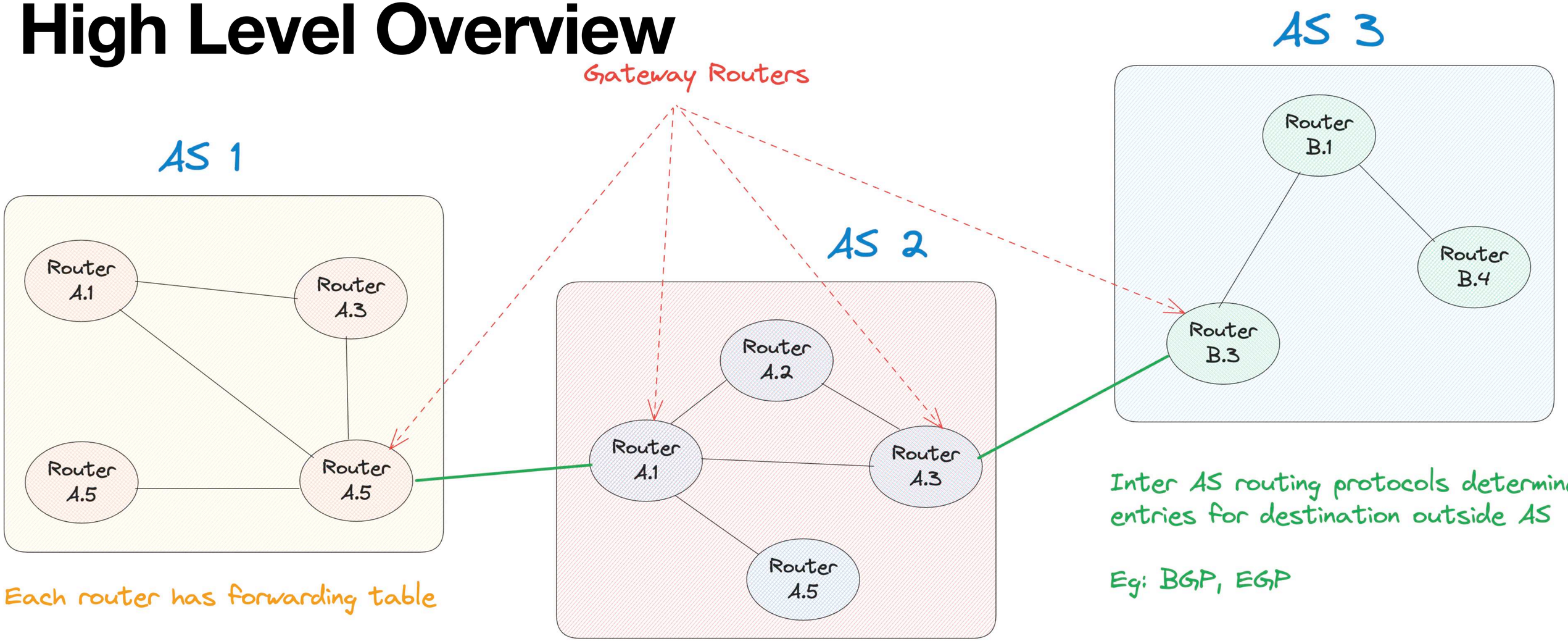
Protocols used in routing

- Intra-AS routing protocols:
  - **OSPF (Open Shortest Path First) Protocol**
    - Classic link state routing (Dijkstra's algorithm)
  - Others include: RIP, EIGRP (RIP: Routing Information Protocol)
- Inter-AS routing protocols:
  - **BGP (Broader Gateway protocol)**
    - Path vector protocol
  - Considered as "glue that holds internet together"

Internet approach to scalable routing

- Aggregate routers into regions known as **“Autonomous Systems” (AS)** a.k.a “domains”
  - Total of around 70,000 AS’s have been assigned not all are active
- There are mechanisms for handling routing within the domain and across AS
- **Intra-AS or Intra-domain**
  - All routers in AS must run the same intra-domain protocol
  - There is a **gateway router** at the edge of each AS which connects with router in another AS
- **Inter-AS or Inter-domain**
  - Routing among AS’s
  - Gateways perform inter-domain as well as intra-domain within their network

High Level Overview



Each router has forwarding table

Configured by both intra and inter AS routing algorithms

Intra AS routing protocols determine entries for destination within AS

Eg: OSPF, EIGP, RIP

Inter AS routing protocols determine entries for destination outside AS

Eg: BGP, EGP

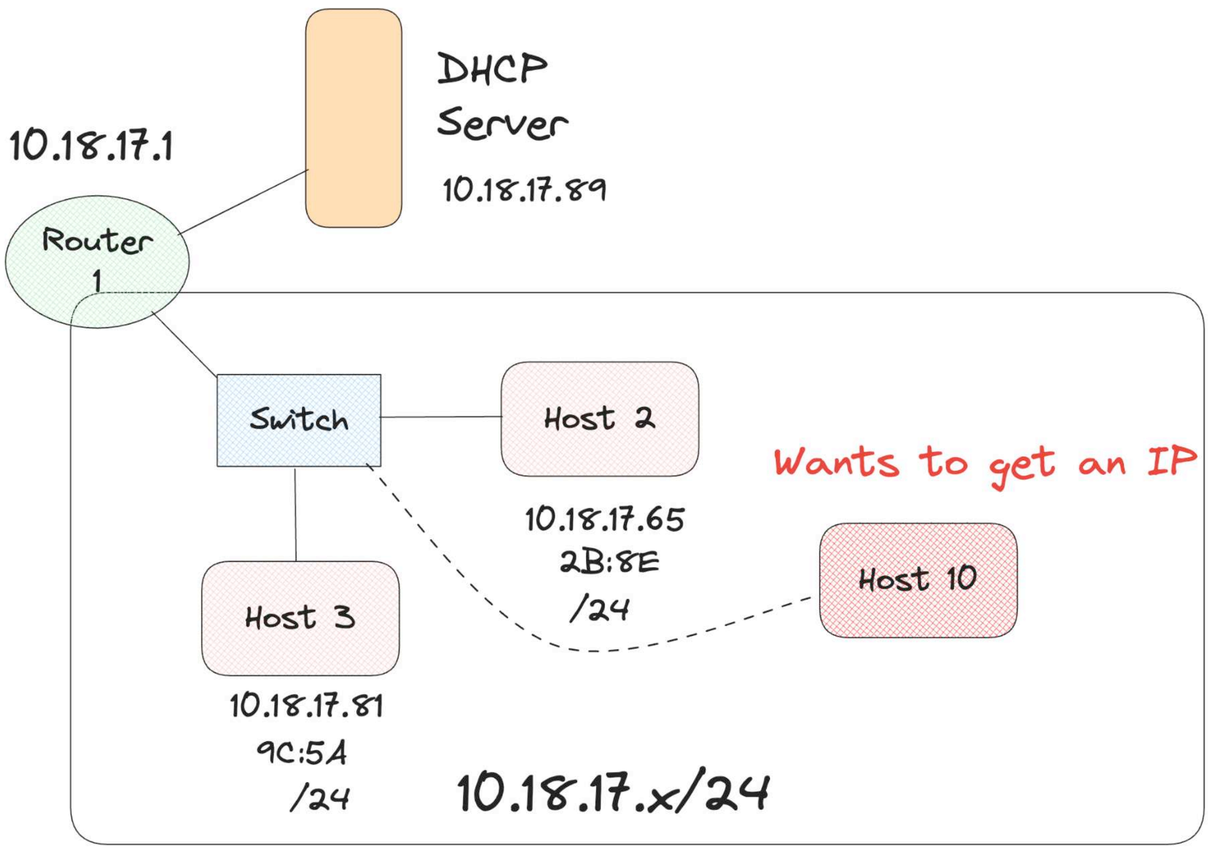
Note: BGP is the used (External Gateway Protocol)

Software Engineering Research Centre

Taking a step back: How to get IP address?

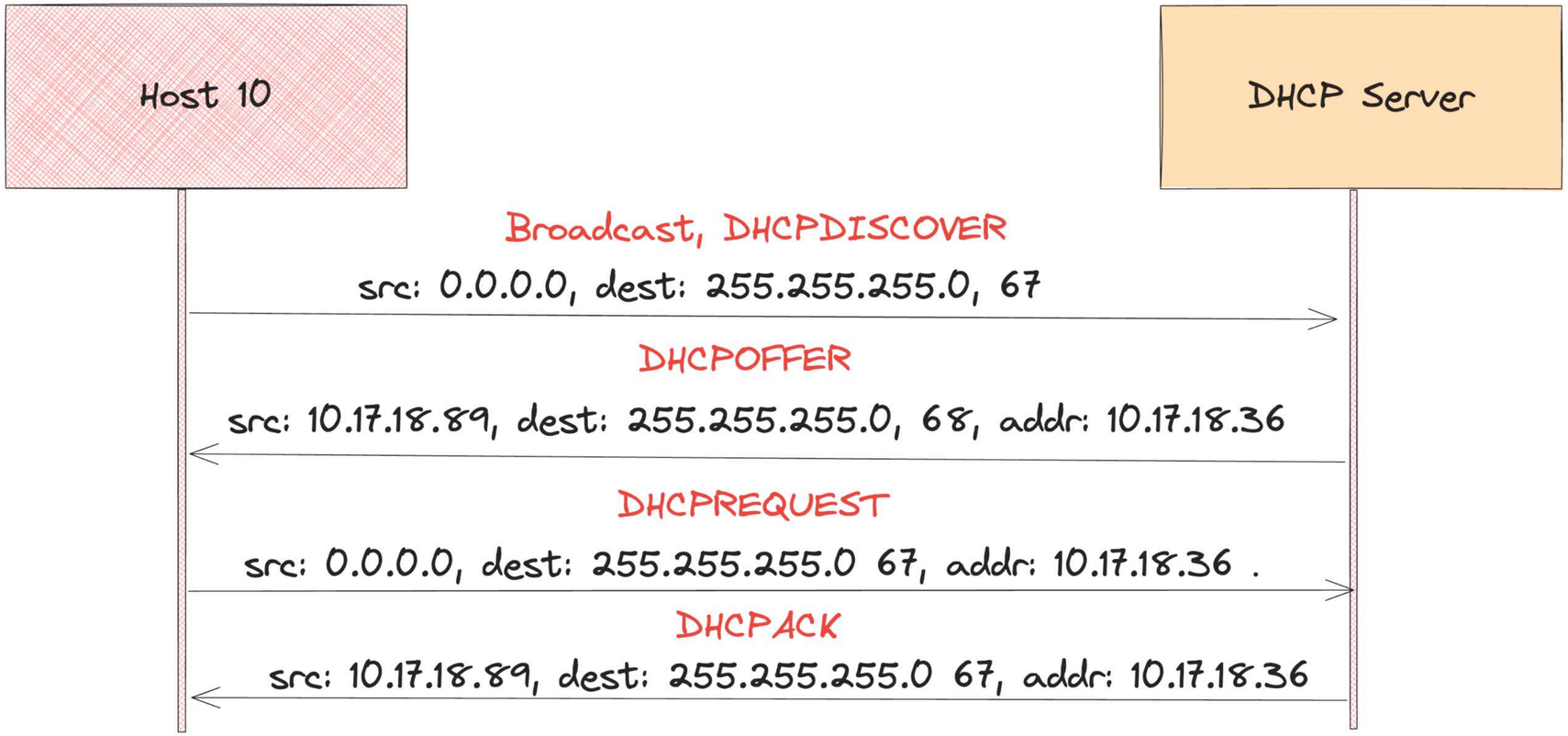
- Two questions needs to be answered:
  - How does host get IP address within its network?
  - What about the network address?
- How does host get an IP address?
  - Hard-coded by sysadmin in config file (e.g., /etc/rc.config in UNIX)
  - **DHCP:** Dynamic Host Configuration Protocol - Dynamically get IP address when joining from a server

Working of DHCP



- Host 10 is the client here
- It sends out a broadcast DHCP request to every node in the network to get DHCP server
- Every device in the network will get the request
- DHCP runs over UDP
- Client uses port 68 and server port (listens on port 67)

Working of DHCP

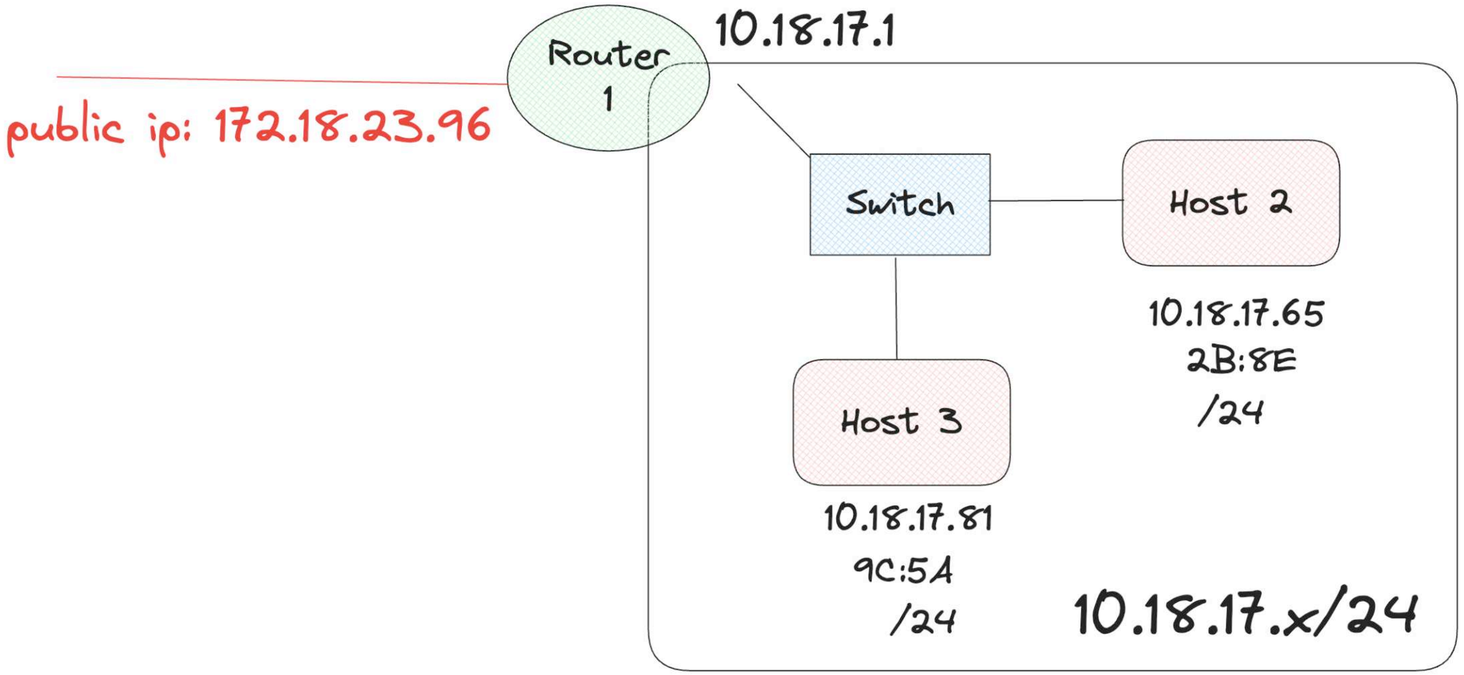


- DHCPDISCOVER will be broadcasted to all the DHCP servers
- The IP address offer will be given by multiple DHCP servers, client chooses one (first response) and broadcasts the acceptance
- DHCP server can also give details like address of DNS server, address of first hop router, network mask, etc.

How to get IP Address

- ISP gets IP address block from ICANN (Internet Corporation for Assigned Names and Numbers) - http://ican.org
  - Allocates IP addresses through 5 regional registries (RRs)
  - There are not enough IPV4 addresses - Last chunk was allocated to RRs in 2011
  - IPV6 - 128 bit address space
- We are still able to function with IPV4 due to NAT (Network Address Translation)

Network Address Translation (NAT)

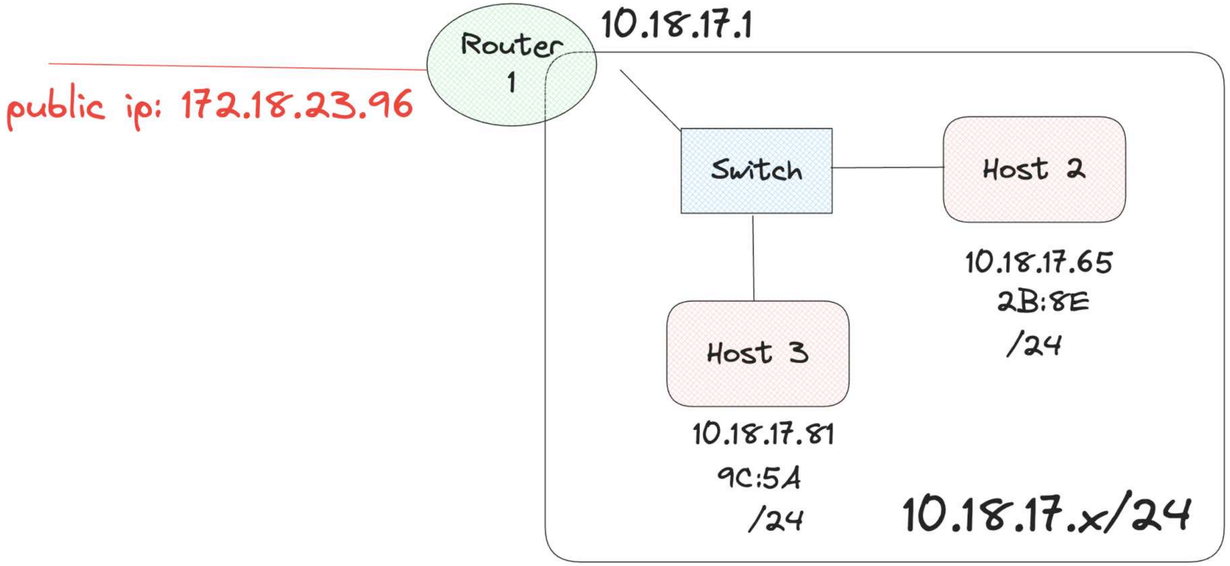


- All devices in the network share just one IPV4 address as far as the outside world is concerned
  - They can still communicate with different hosts outside the network with one public IP
  - How is that possible and how to make this work?

Network Address Translation (NAT)

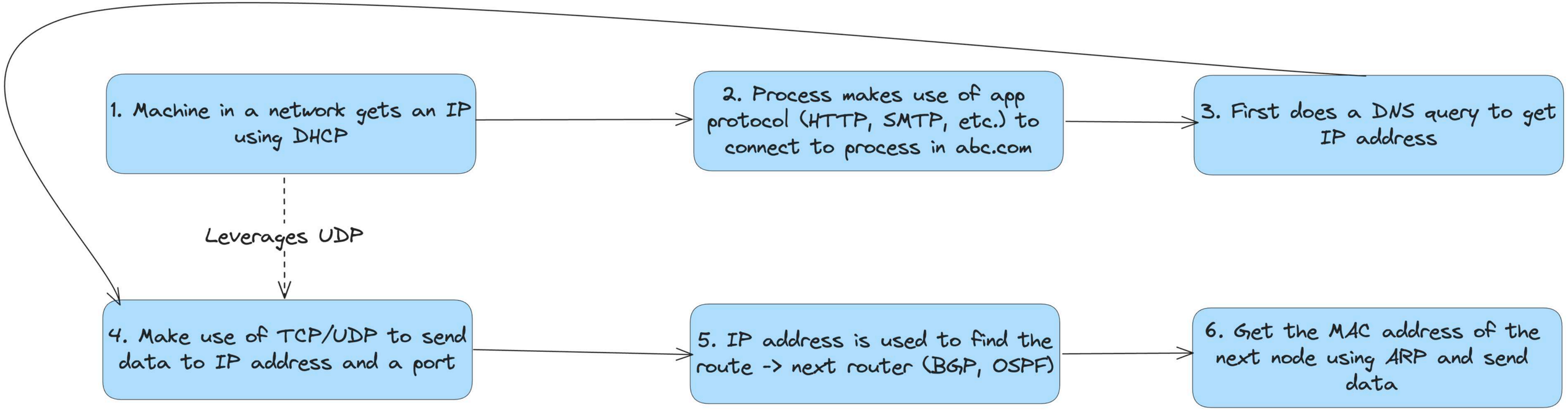
NAT Translation Table

WAN side address	LAN side address
172.18.23.96 5501	10.18.17.81 3801
....	.....



- NAT allows a router (similar device) to translate private IP addresses to its own public IP address
- When devices from network wants to communicate with outside network:
  - NAT modifies the source IP to make it appear that communication is from the larger public IP
  - A translation table is used for managing the translations
- **Multiple types:** Static NAT, Dynamic NAT, Port Address Translation or NAT Overload

Putting it together



SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Persistence: Hard Disks

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

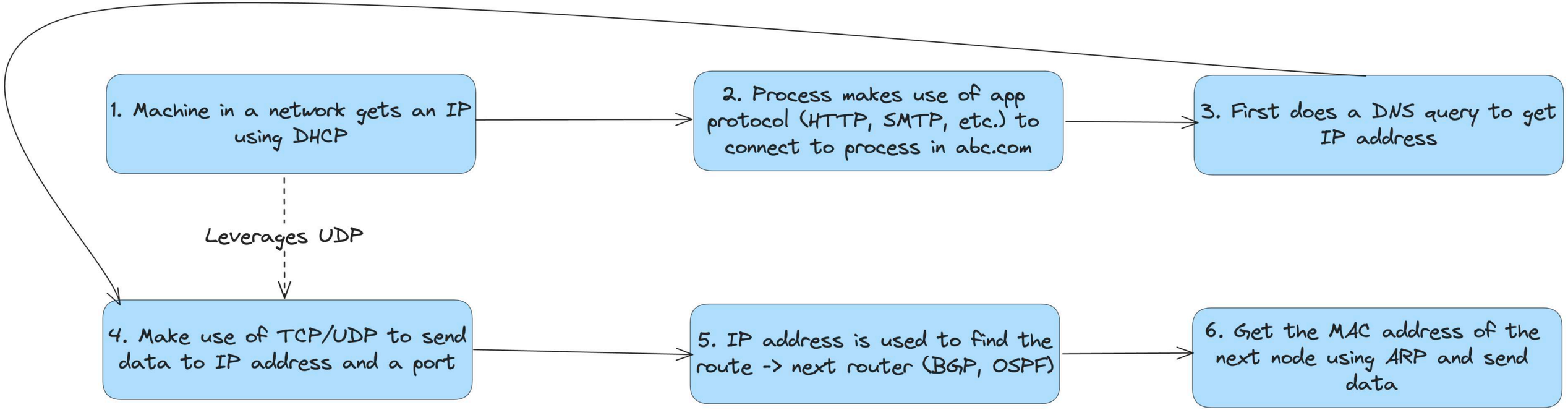
Acknowledgement

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:

- Computer Networks, 6e by Tanebaum, Teamster and Wetherall
- Computer Networks: A Top Down Approach by Kurose and Ross
- Computer Networking essentials, Youtube Channel
- Other online sources which are duly cited

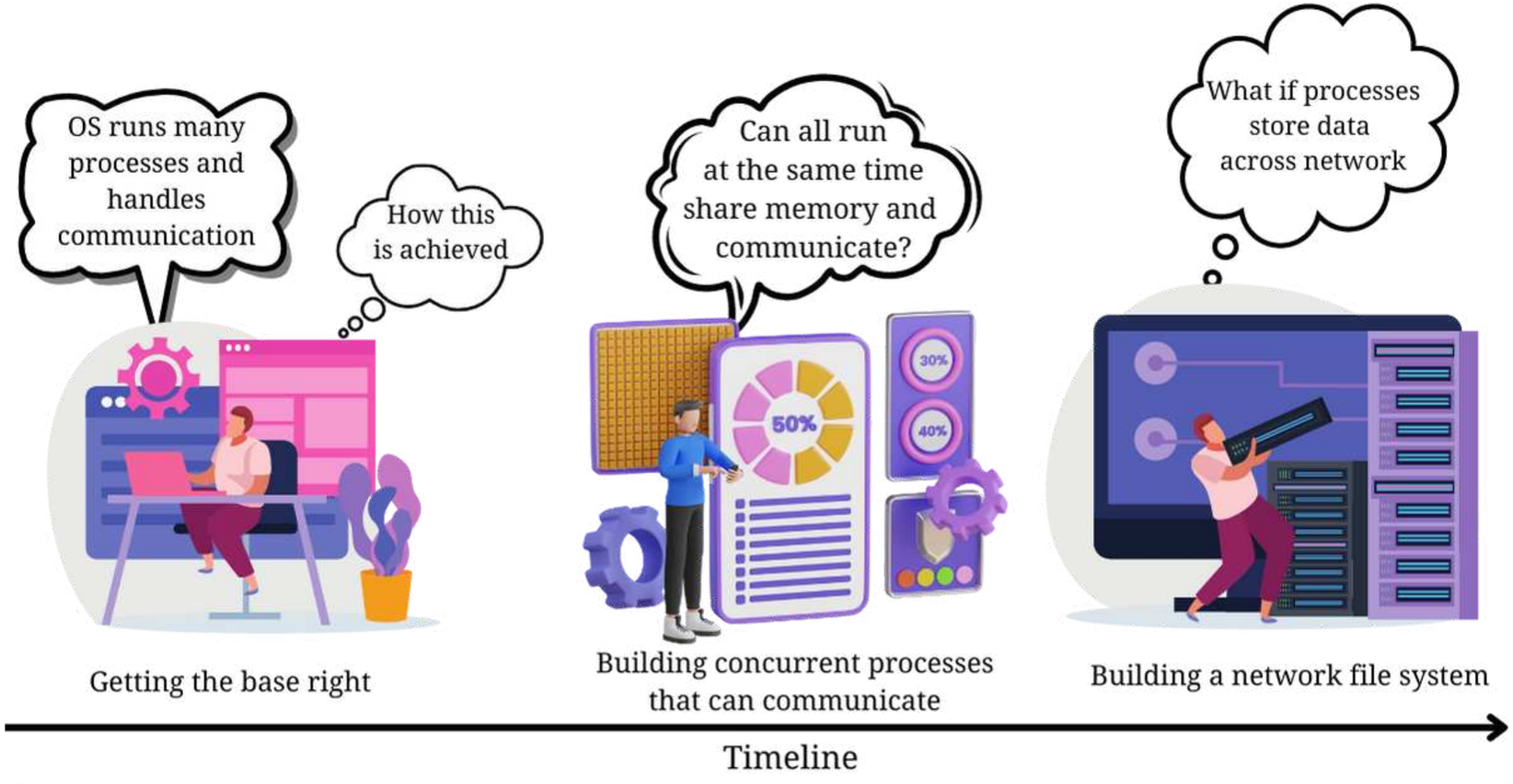
Putting it together for Networks



Wait we still have one main part

How does OS handles the data and how is it stored?

Lets go to the final part



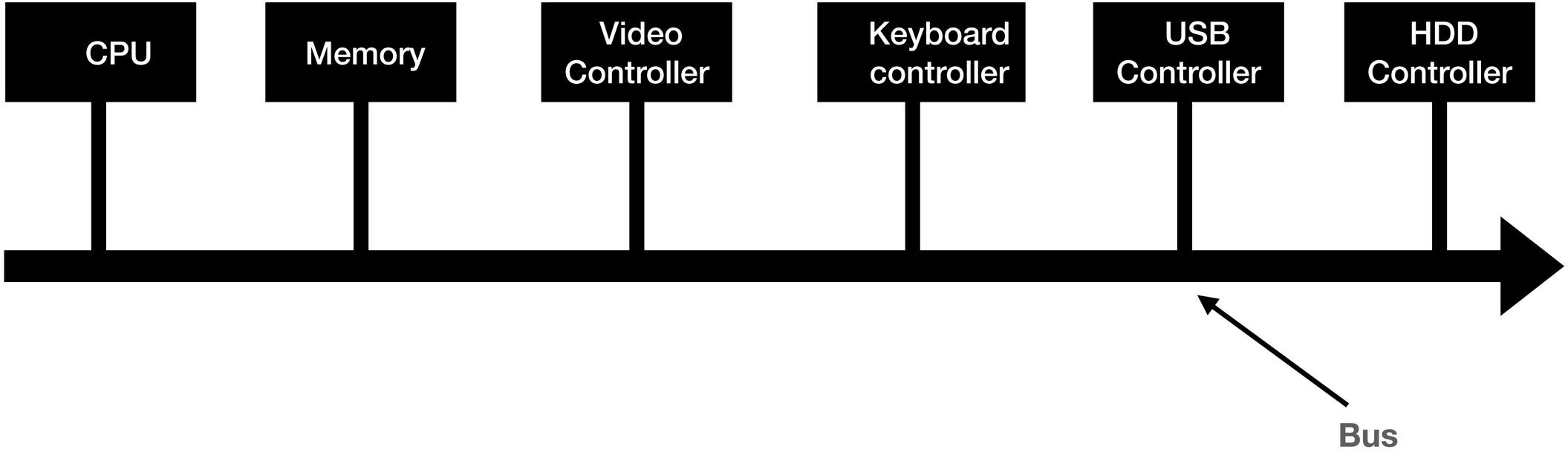
This Course

Process and Memory Virtualization	Concurrency	Persistence
Networking intro	Addressing and Routing	Network file Systems

Persistence

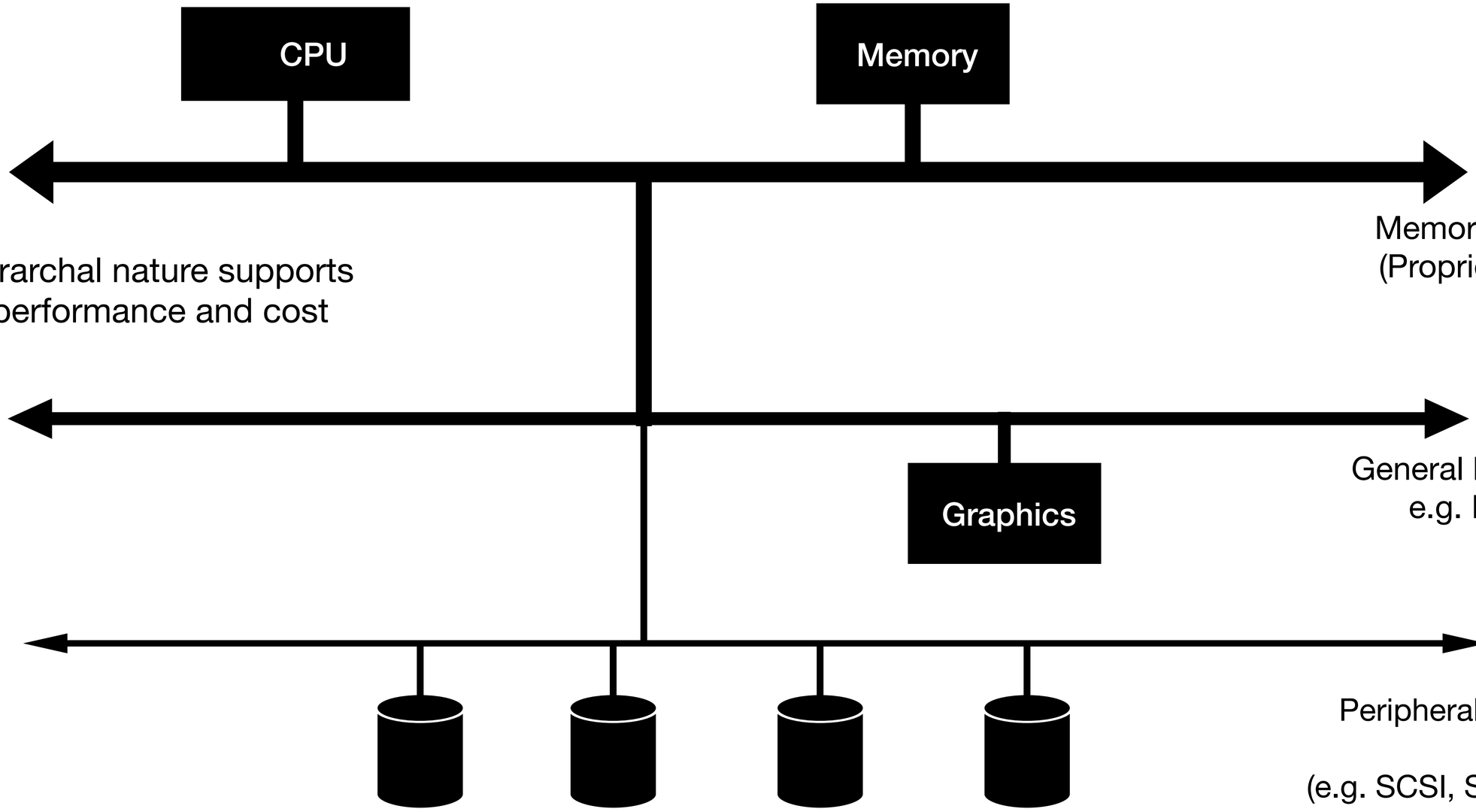
- RAM is Volatile - Again we need two parts here
- **Hardware** and **software** are needed to store data persistently
  - Hardware: I/O devices such as hard drive, SSDs, etc.
  - Software:
    - File system manages the disk
    - File system is responsible for any files that the user creates
    - Read, writes are handled by file system which interacts with low level device drivers

Some Prerequisite



As we go more away from CPU, the more time it takes

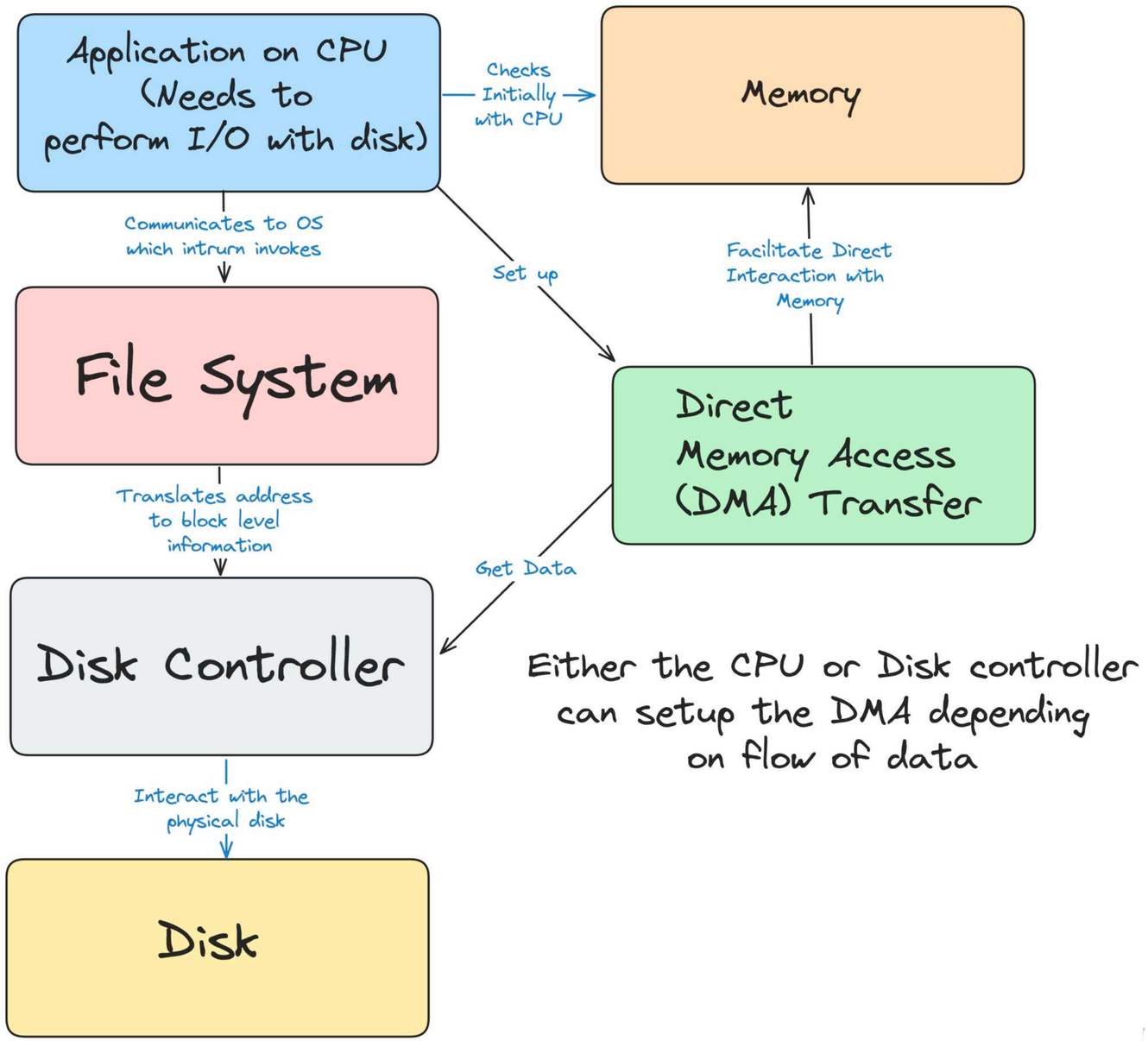
Prototypical System Architecture



The hierarchal nature supports both performance and cost

The flow of access

- Application performs read or write to a file
- CPU communicates to OS which invokes the File System (FS)
- The OS may check in its cache if its already there
- FS prepares block level information to disk controller
- A Direct Memory Access (DMA) is set up
- Disk controller performs the physical read or write based on commands from DMA and file system
- If its read, Disk -> DMA, for writes, DMA -> Disk



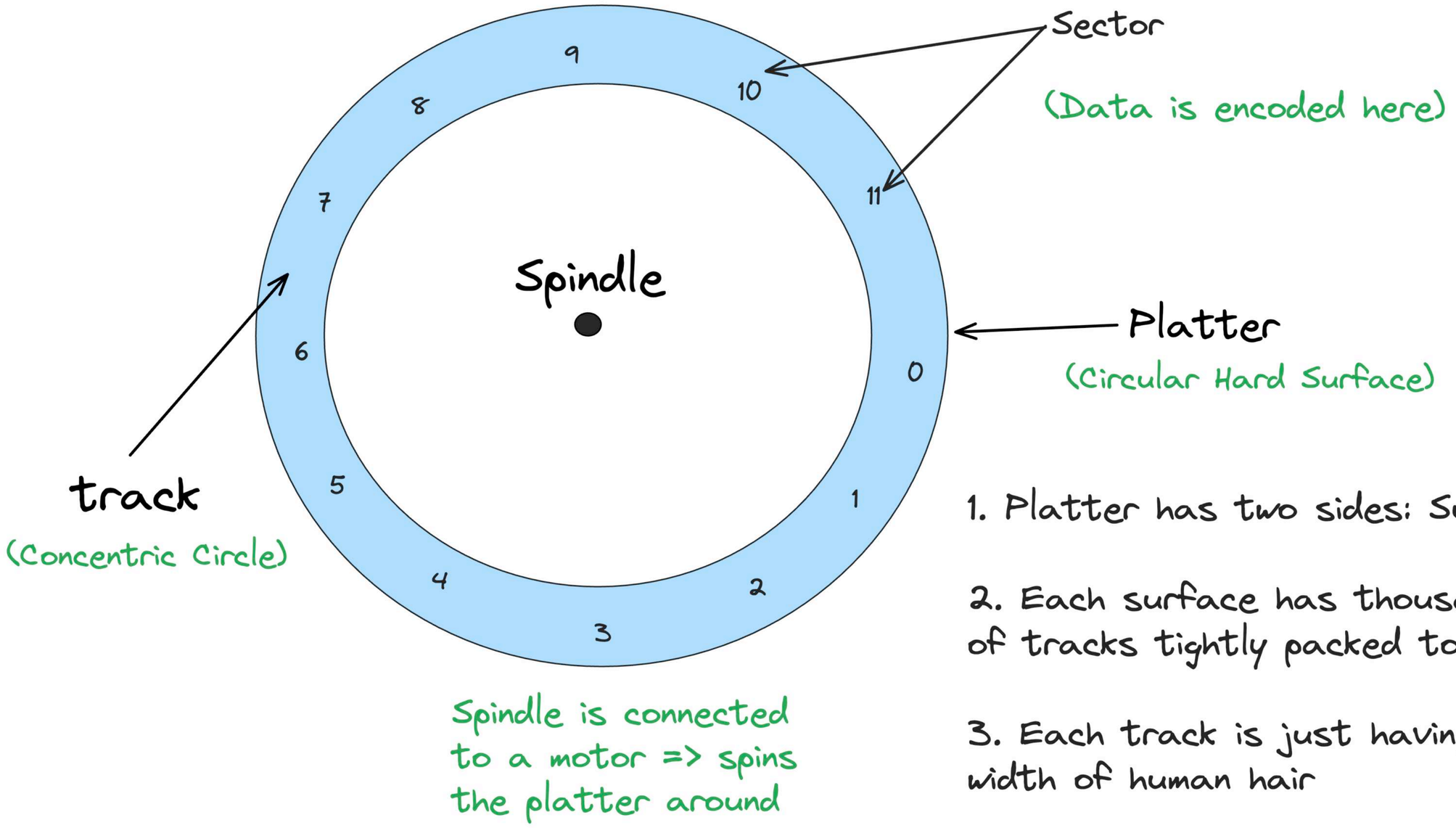
Hard Disk and How it works!

- Main form of storage in computer systems over decades
- How to store and access data?
  - How do modern hard disks store data?
    - What is the interface?
  - How is the **data laid out** and **accessed?**
  - How does **disk scheduling** improve performance?

Interface to Hard disk

- Drive consists of a number of 512 byte blocks each of which can be read or written
- Sectors are numbered from *0* to *n-1* on disk with n sectors - **Address space**
- Many file systems read or write **4 KB** at a time (or more)
- The main guarantee that is provided is that a single 512 byte write is **atomic**
  - Either it happens completely or not at all
- Power loss in between can result in a portion of write becoming incomplete **(torn write)**

Basic Geometry

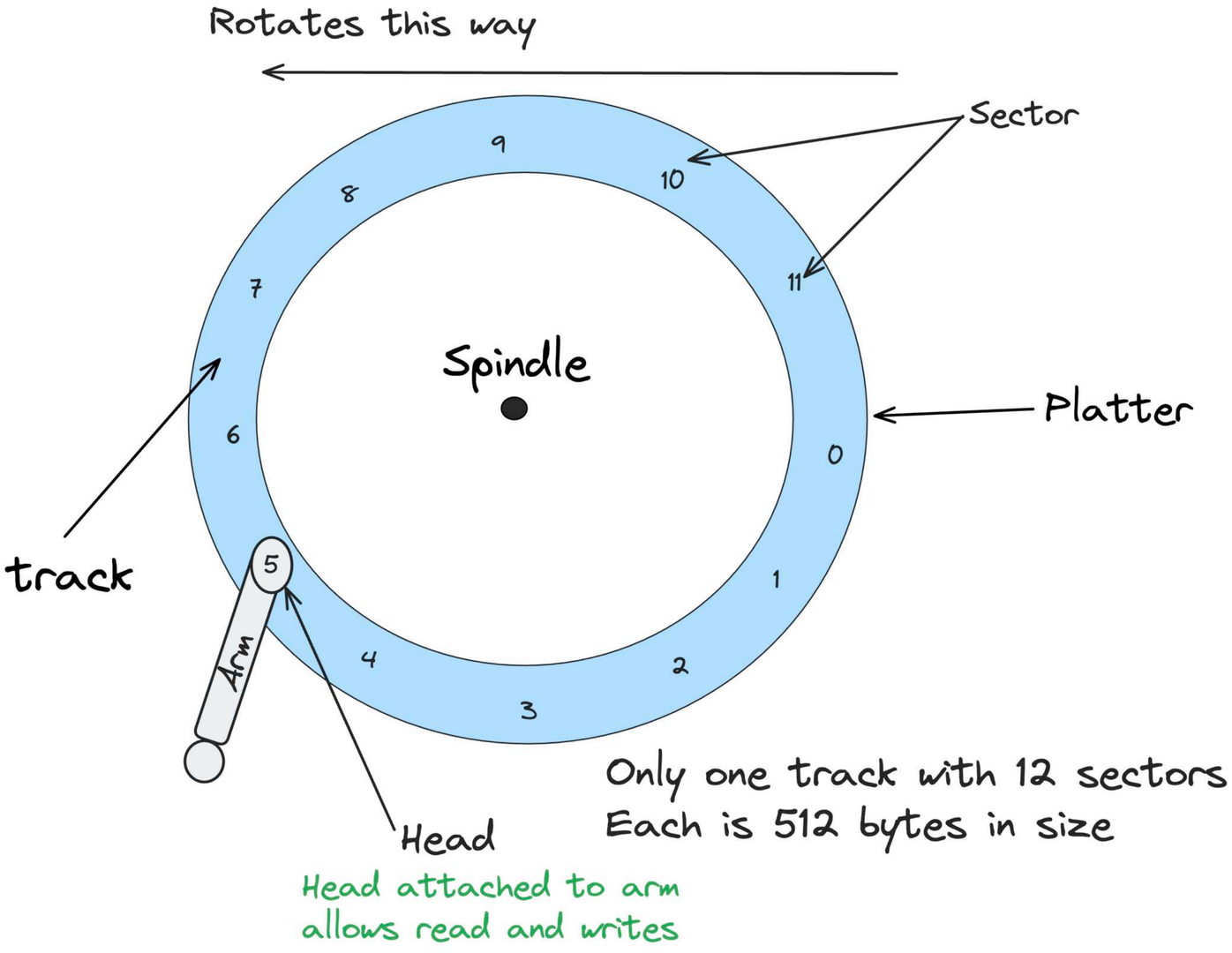


1. Platter has two sides: Surface

2. Each surface has thousands of tracks tightly packed together

3. Each track is just having width of human hair

Simple Disk Drive



- Disk arm moves across to support reads and writes
- Spindle connected to a motor, spins the platter around at a constant fixed rate
- The rate is measured in rotations per minute (RPM)
- Modern values in range **7200 to 15000 RPM**
- For drive that rotates at 10,000 RPM, single rotations takes at **6ms**

Single-Track Latency: Rotational Delay

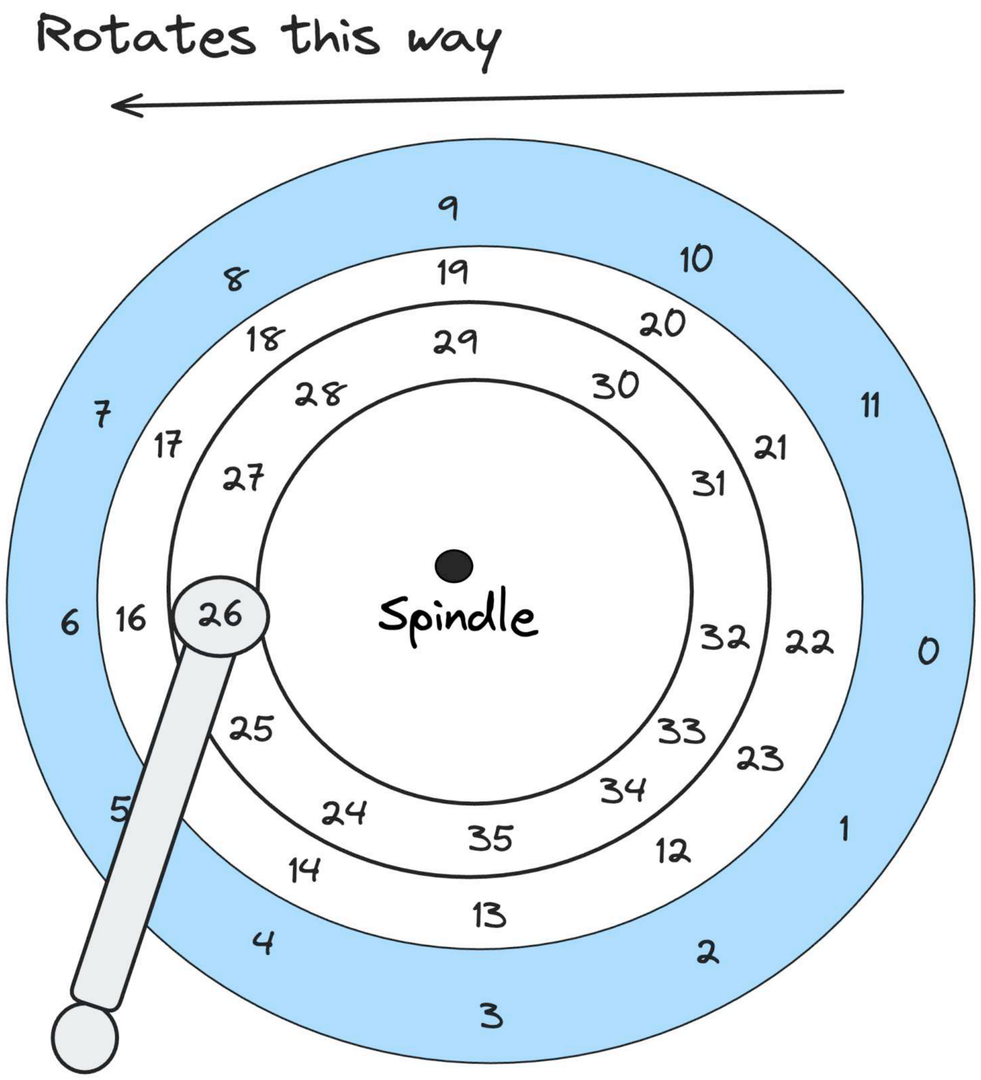
- Consider in the previous case that the access has to be done in block 0
  - Remember that the disk rotates and as soon as the head reaches the desired sector (block), it can read
  - The position of head was at sector 5
  - It has to wait for the disk to rotate and the head to reach at sector 0
  - This is known as **rotational delay** or **rotation delay**
  - What if the read access request arrived for sector 4?

Almost full rotation!!

But Disk is not just about Single Tracks

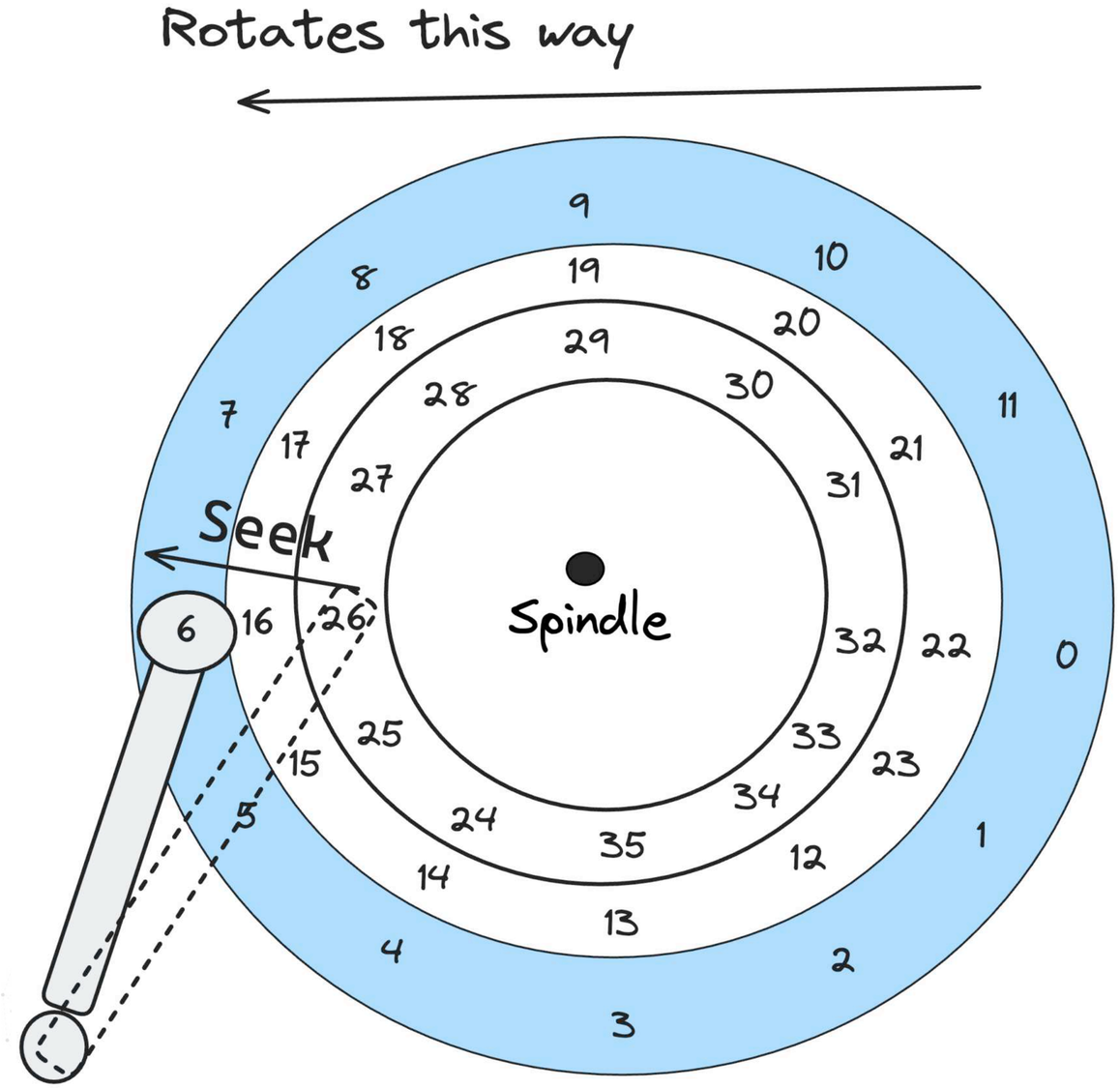
Source: Times of India

Multiple Tracks: Seek Time



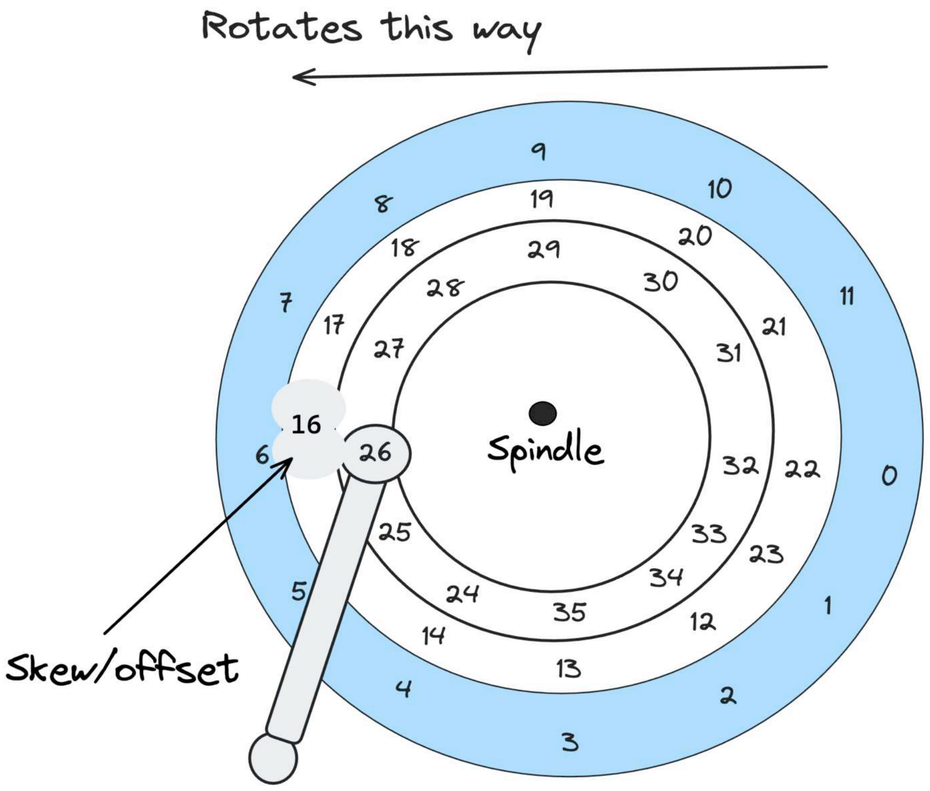
- Its not just about rotational delay
- In a real setting, a disk surface has 100s of tracks
- Read or write may happen at any block located in different tracks
- Rotation will only help in movement within a track
- Across tracks, the operation performed is **Seek**

Multiple Tracks: Seek Time



- Seek is a costly operation in terms of time
- Seek has multiple phases
  - **Acceleration:** Arm starts moving
  - **Coasting:** Arm moving at full speed
  - **Settling:** Head over correct track
- Settling time is 0.5 - 2ms High!
- Final phase of I/O is **transfer** - Read or write from surface
- **Seek, rotate and transfer** - three key phases

We may also have to consider skew



- Many drives some kind of **track skew** to make sure that sequential reads can be properly serviced
- When head moves from one track to another:
  - By the time the head moves, the desired block in the track would have got rotated
  - Head would now have to wait for a longer rotational delay
- To avoid this beginning of next track is slightly offset or **skewed**
- This is done to optimize performance

Modern Disk Drives also have Cache!

- The cache is often referred to as **Track buffer**
  - Allow the drive to quickly respond to requests
  - Small amount of memory (usually around 8 to 16 MB)
  - Can be used by drive to read from/write to the disk
- **Reads:** When reading from one sector, read all sectors in that tack and cache
  - Subsequent reads can be very fast
- **Writes:** Two choices: **Write through** and **Write back**

Write on Cache

- **Writeback** (Immediate reporting)
  - Acknowledge a write has completed as soon as the data reaches cache memory
  - It makes drives appear very fast but it can be dangerous
  - Especially if order needs to be preserved - **This can lead to problems!**
- **Writethrough**
  - Acknowledge when the write has been written to the disk
  - Here data written to cache is also written to cache and disk simultaneously
  - Overall performance here might be an issue

Some Analysis

- Disk rotates at **10,000 RPM** and has transfer rate of **100 MB/sec**
  - **How much milliseconds does a single rotation take?**
    - 1 minute = 60 seconds = 60,000 ms
    - 10,000 RPM in 60,000 ms => **6 ms for 1 rotation**
  - **How much time to transfer 512 KB blocks of data?**
    - 0.5 MB of data
    - 100 MB/sec => 0.1 MB/ms => **5 ms for 0.5 MB of data**

I/O Time of Disk

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

Rate of I/O $$R_{I/O} = \frac{Size_{transfer}}{T_{I/O}}$$

- We can perform different analysis given some characteristics
- Assume that there are two different workloads
  - Random workload
  - Sequential workload

I/O Time of Disks

- **Random Workload**
  - Issues small (4 KB) reads to random locations on the disk
  - Very common in applications like Database management systems
- **Sequential Workload**
  - Reads large number of sectors consecutively from disk
  - These are also quite common!
- Given workload, can we perform some comparison on the disk performance
  - We would also need some disk characteristics

Disk Performance Analysis

Characterestic	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max. Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects Via	SCSI	SATA

- What are some observations about the disks here?

Some Observations

- One is about performance - Cheetah
  - Drives are engineered to spin as fast as possible
  - Delivers low seek time and fast transfer rate
- Another is about capacity of the storage - Barracuda
  - Cost per byte is important
  - Drives are slow but packs as many bits into given space

Some Analysis

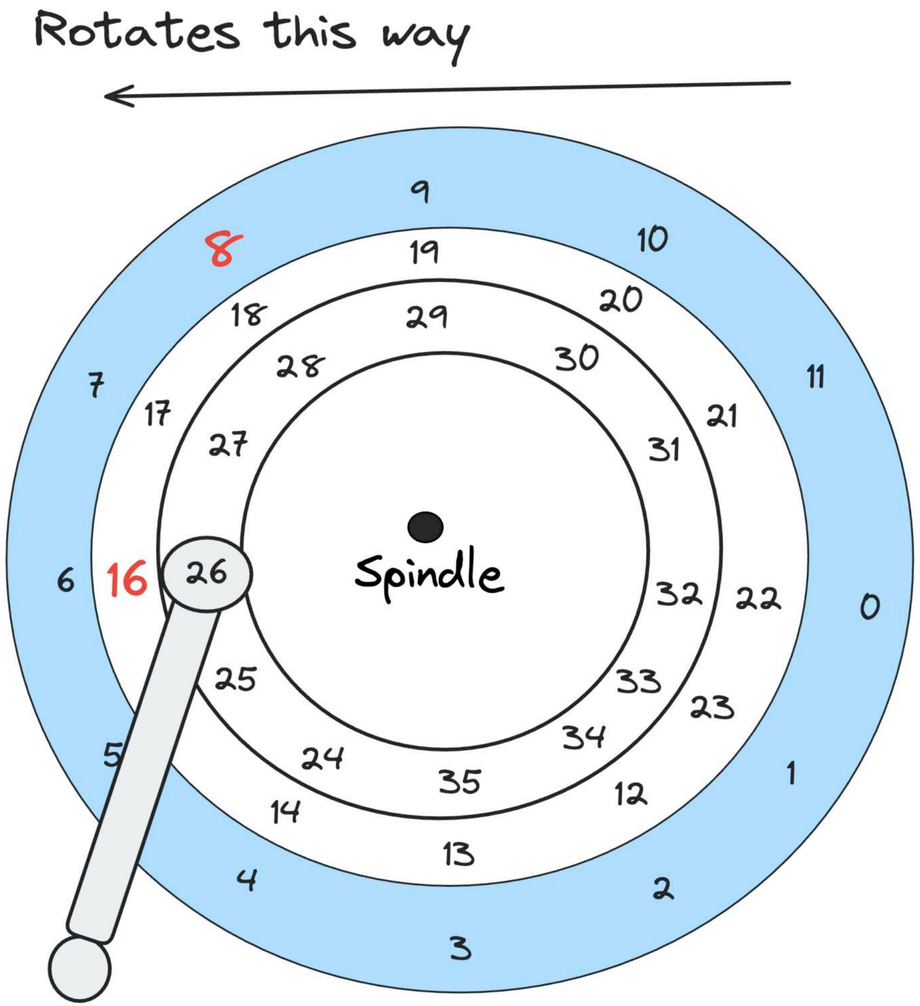
Workload	Metric	Cheetah	Barracuda
	Tseek	4 ms	9 ms
	Trotation	2 ms	4.2 ms
Random (4 KB reads)	Ttransfer	30 micros	38 micros
	TI/O	6 ms	13.2 ms
	RI/O	0.66 MB/s	0.31 MB/s
Sequential (100 MB reads)	Ttransfer	800 ms	950 ms
	TI/O	806 ms	936.2 ms
	RI/O	125 MB/s	105 MB/s

There is large difference in performance between **high-end performance** drives and **low-end capacity** drives

Disk Scheduling

- OS plays a role in the order of I/O issued to disk
- Given a set of I/O requests, is it possible to schedule them in an order
  - Disk scheduler decides which I/O request to schedule next!
  - Decides which request to schedule to improve performance
- How can disk scheduler make a guess on what request shall be better?
  - Remember: Disk scheduler can estimate how much time each request can take
  - Parameters like rotational delay, transfer, seek are known/easily estimated

SSTF: Shortest Seek Time First



- Orders the queue of I/O requests by track
  - Pick the block in the queue from nearest track to complete first
- In this case
  - The closest one is 16
  - Followed by 8
  - Schedule: 16 -> 8

Scheduling Requests: 8, 16

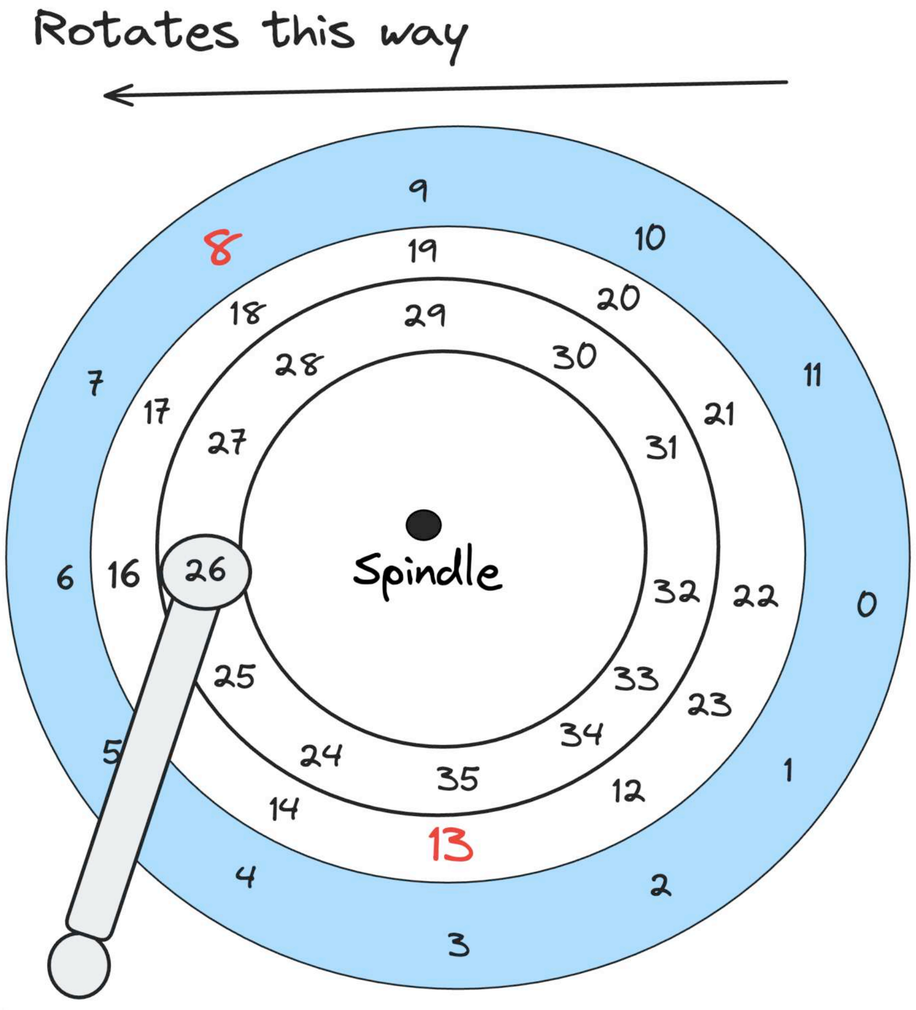
Two main issues

- Drive geometry is not available to the OS
  - It sees everything as blocks
  - One way out: Implement something like **Nearest Block First (NBF)**
- There is another main issue: **Starvation!**
  - If there is a steady stream of requests to inner tracks
  - Request to outer tracks may be ignored completely leading to starvation
- Can we do something better to avoid starvation?

Elevator (a.k.a SCAN or C-SCAN)

- Simply move back and forth, servicing the requests in order
  - **Sweep:** A Single Pass across the disk
  - If a request comes for a block on a track that has already been serviced in this sweep, it has to wait in a queue till the next sweep
- **F-SCAN**
  - Freeze the queue to be serviced when doing a sweep
  - Avoid starvation of far-away requests
- **C-SCAN (Circular scan)**
  - Sweep from inner-to-outer and outer-to-inner, etc.

Shortest Positioning Time First



- Rotation and seek needs to be considered
- Which to give preference - **It depends!**
- Assume I/O requests to 8 and 13
  - Seek and rotation both are time consuming
  - Assume that seek is faster than rotation
    - 8 -> 13 makes more sense
  - If rotation is faster than seek
    - 13 -> 8 makes more sense

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Persistence: RAIDs

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

Acknowledgement

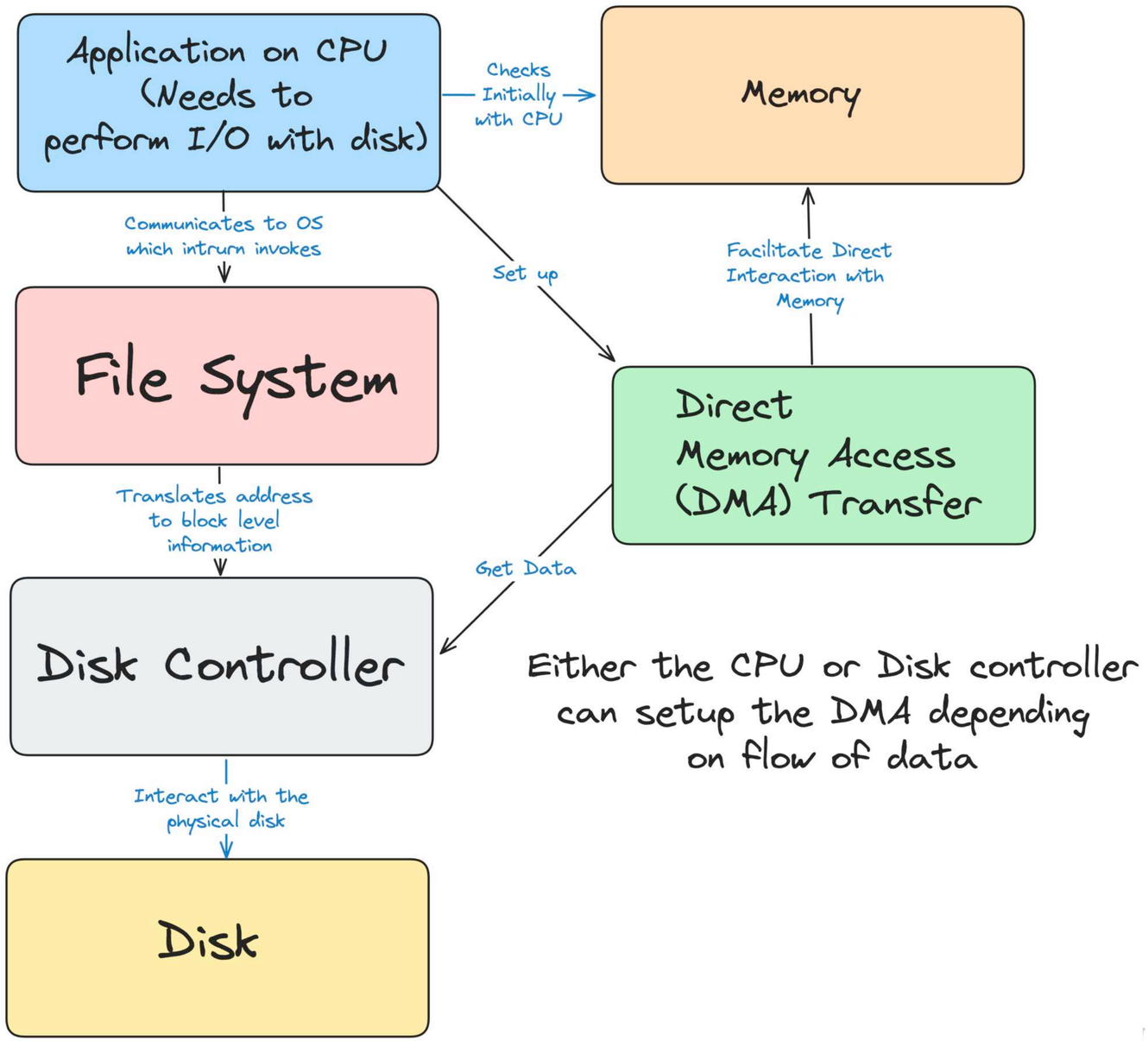
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Operating Systems in Three Easy Pieces by Remzi et al.

The flow of access

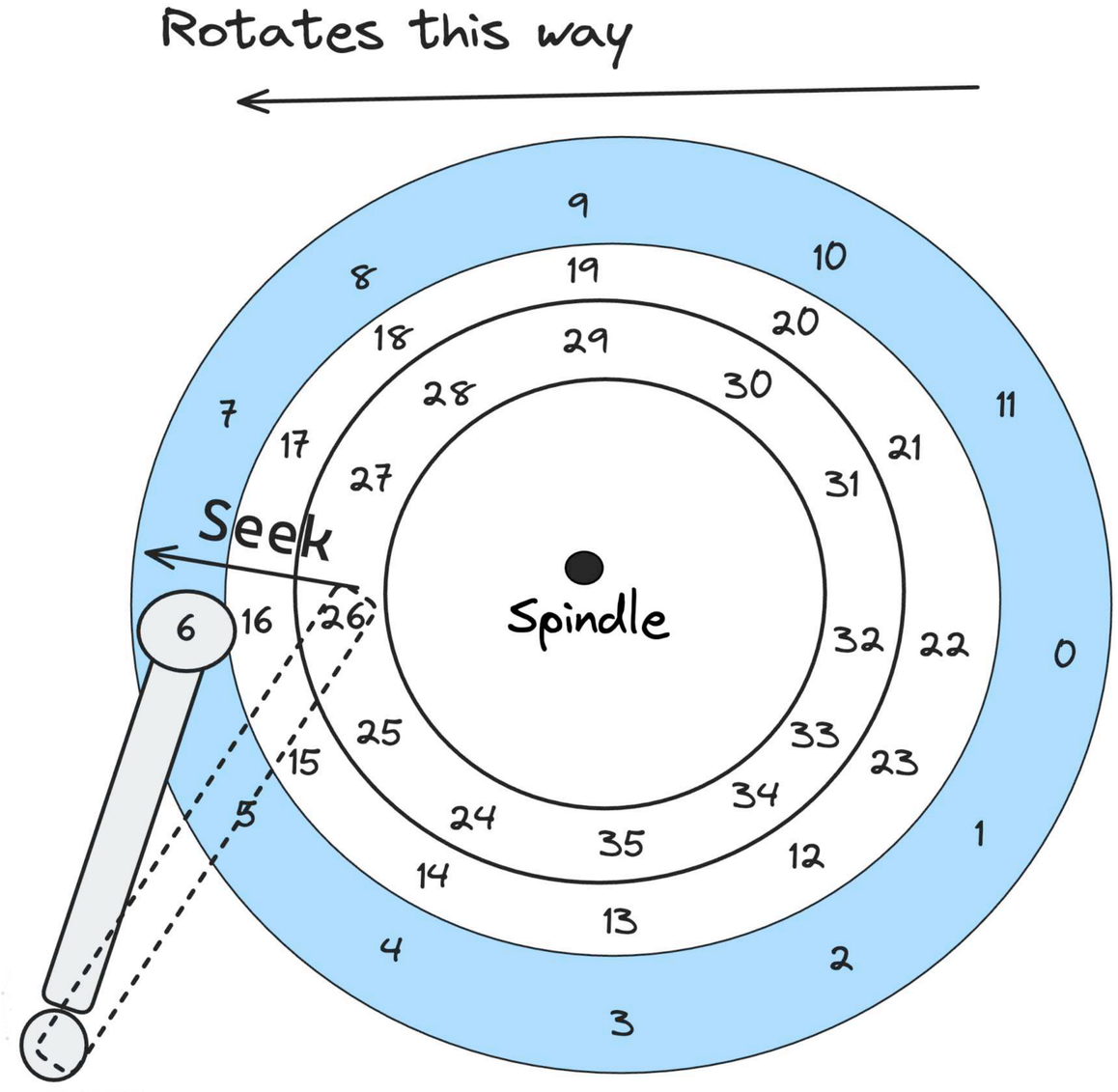
- Application performs read or write to a file
- CPU communicates to OS which invokes the File System (FS)
- The OS may check in its cache if its already there
- FS prepares block level information to disk controller
- A Direct Memory Access (DMA) is set up
- Disk controller performs the physical read or write based on commands from DMA and file system
- If its read, Disk -> DMA, for writes, DMA -> Disk



Modern Hard Disks

Source: Times of India

Quick Overview

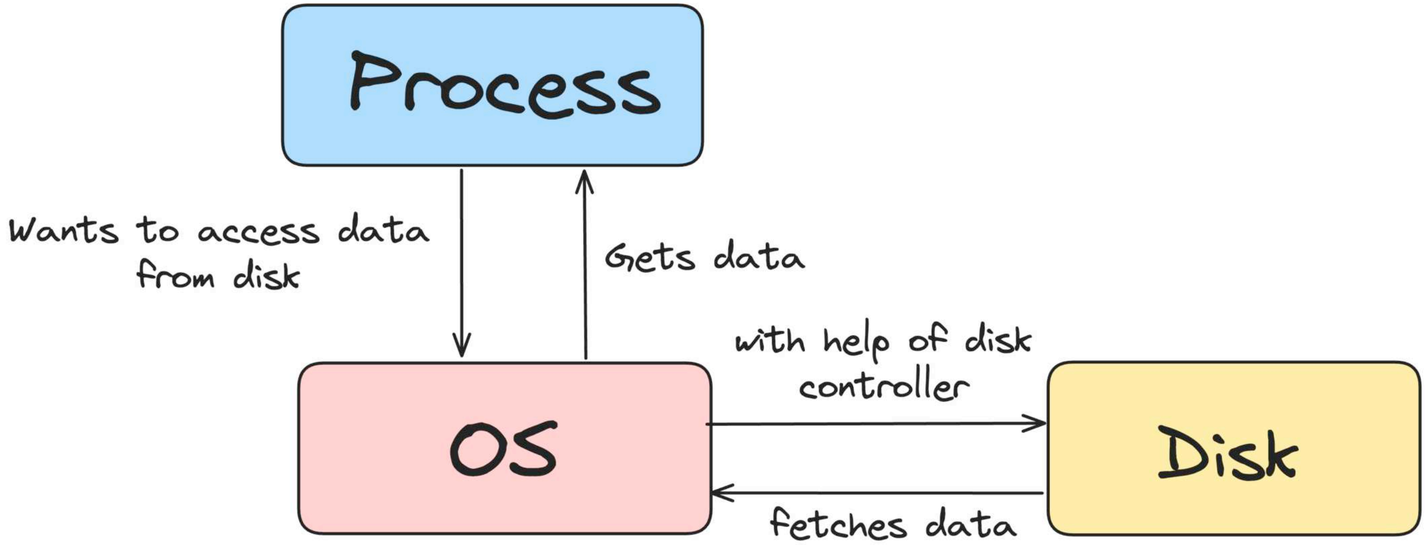


- Disk rotates on a spindle
  - The arm can move across (seek) or stay as the disk rotates
  - The head is used to read/write
- Data is arranged in tracks as blocks/ sectors
- There are 100s of tracks on a single disk
- **Seek, rotate and transfer** - three key phases

I/O Time of Disks

- **Random Workload**
  - Issues small (4 KB) reads to random locations on the disk
  - Very common in applications like Database management systems
- **Sequential Workload**
  - Reads large number of sectors consecutively from disk
  - These are also quite common!
- Given workload, we can perform some comparison on the disk performance
  - We would also need some disk characteristics

So far its about one disk!

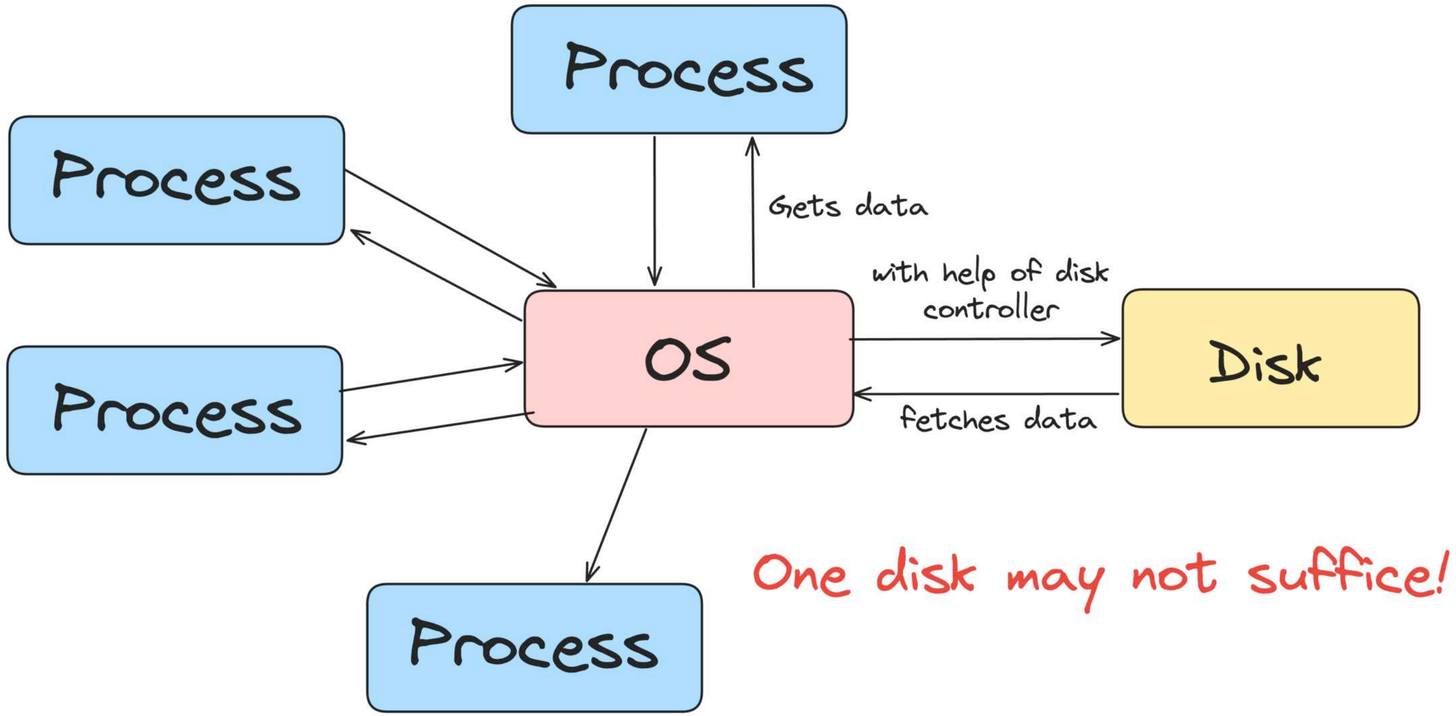


Consumer laptops

Data centers

Will the idea of one disk be enough?

We may need more!



- Disks are slower!
- I/O is slower - Bottleneck!
- Disks may get fuller
- Disk can also fail
- Multiple facets needs to be considered
- What can be a better mechanism?

Redundant Arrays of Inexpensive Disks (RAID)

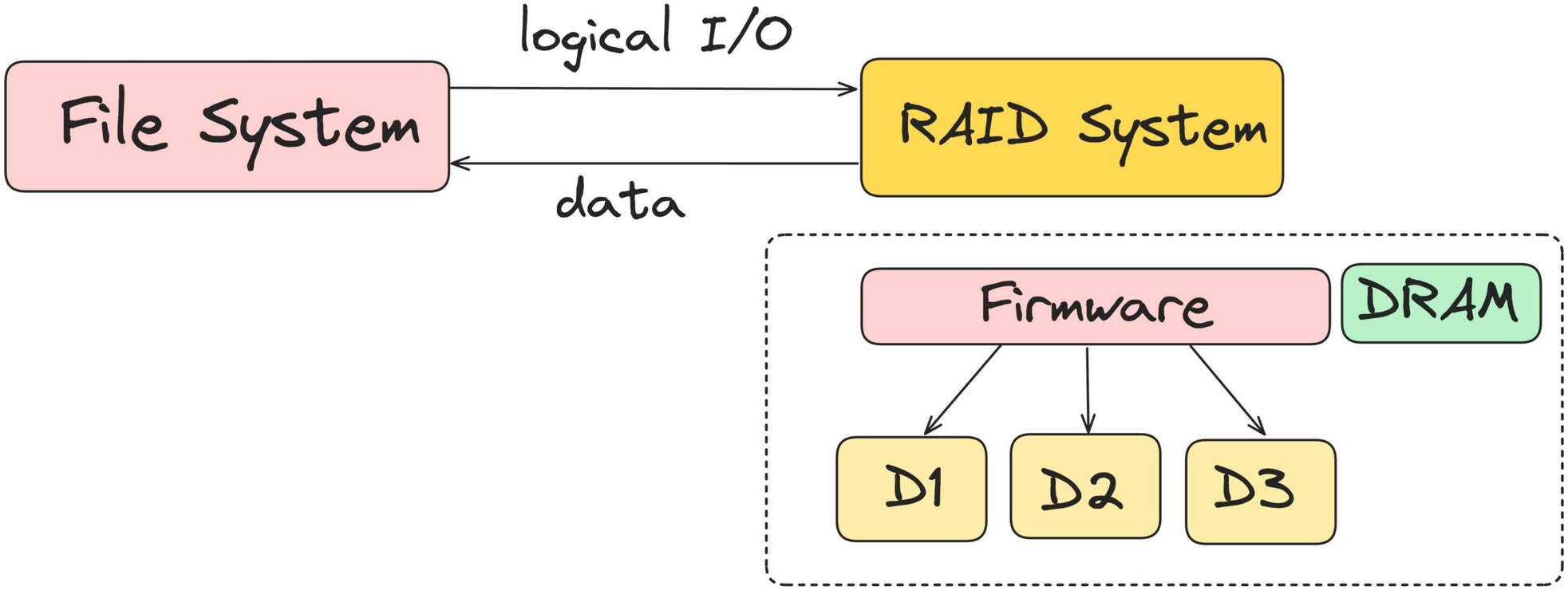
Redundant Arrays of Independent Disks!

- Techniques to use multiple disks in concert to build **faster, bigger and more reliable** disk system
- Term introduced in late 90's by a group of researchers in UC Berkley
- Externally RAIDs look just like group of blocks one can read or write
  - Internally RAID is very complex
    - Consisting of **multiple disks**
    - **Its own memory** - DRAM
    - **One or more processor** to manage the system

RAIDs vs Traditional Disks

- One advantage is **performance**
- Multiple disks in parallel can greatly enhance speed
- More disks => More **capacity** as well
- RAIDS can also enhance **reliability**
  - Without RAID techniques, the disk is vulnerable to loose data
  - RAIDs can tolerate loss of data and keep operating as if nothing went wrong - Redundant disks
- RAID provides advantages **transparently** to the system
  - OS feels that its just interacting with a single disk

RAIDs: Simple Illustration



- As far as **File System (the subcomponent inside OS) is concerned**
  - RAID is just like a disk
  - Linear array of blocks each of which can be read or written

RAID in Action

ThinkSystem ST250 V2 Tower Server

Specs Features Models

Starting at: **₹125,990**

Shop

	Slot 3: PCIe x4 slot with PCIe Gen3 x4 lanes Slot 4: PCIe x8 slot with PCIe Gen3 x4 lanes
Network Interface	2x GbE on-board ports (Broadcom BCM5720); 1x GbE port dedicated for XCC management
Ports	Front: 1x USB 3.2 G2 (10Gb) port, 1x USB 2.0 port for local management using the XCC Mobile app Rear: 4x USB 3.2 G2 (10Gb) ports, 2x RJ45 Gigabit Ethernet ports, 1x 1GbE dedicated XCC port for remote management, 1x Serial port and 1x VGA port
HBA/RAID Support	Intel® VROC Software RAID support with both simple-swap and hot-swap configuration; multiple hardware RAID configurations supported

https://www.lenovo.com/in/en/data-center/servers/towers/

RAIDs

- At a high level, RAIDs are like a computer system
    - RAID is like a box with standard (SCSI or SATA) to a host
    - Provides a consistent interface to the OS
- Internally RAIDs are very complex
    - Consists of a **microcontroller** that runs a firmware
    - **Volatile memory** such as DRAM to buffer data blocks as they are read and written
    - **Non-volatile memory** to buffer writes safely and for parity calculation as well
- Instead of running application RAID, runs specialised software designed to operate RAID

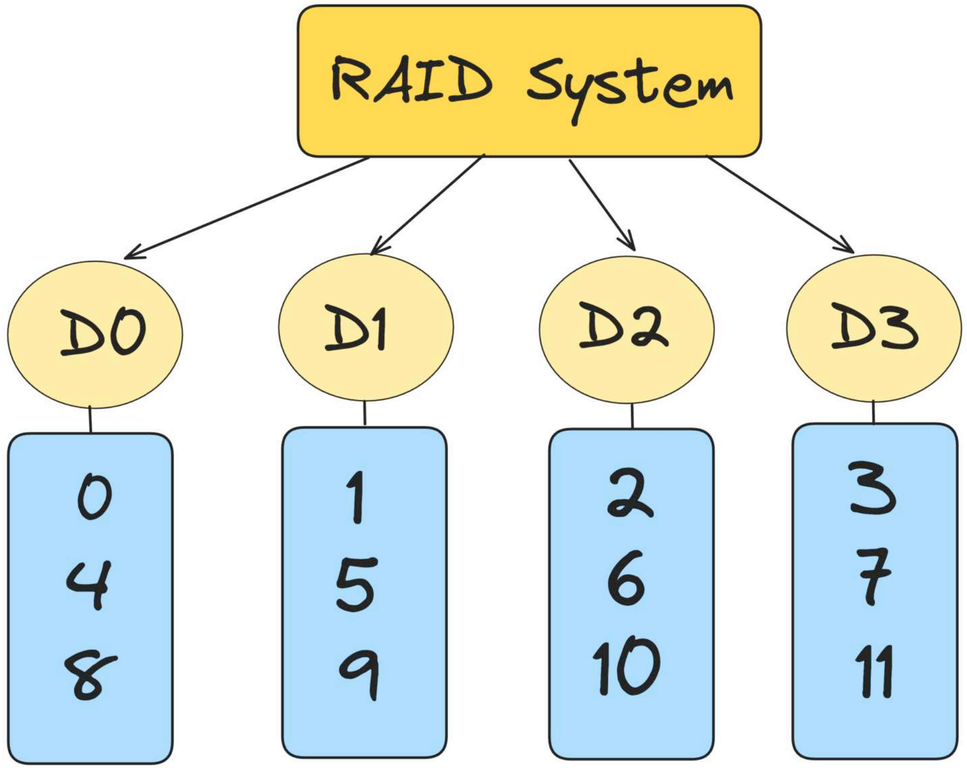
Evaluating RAIDs

- Many approaches are there to build a RAID system
  - Each has different characteristics
- Three axes can be used for evaluation
  - **Capacity**
  - **Reliability**
  - **Performance**

Evaluating RAIDs

- **Capacity**
  - Given a set of N disks each of size B blocks. How much capacity is available for usage?
    - Some redundancy may be required => N/2 when each is replicated
- **Performance**
  - What's the impact of different workload on the latency of I/O?
  - What's the throughput? Rate of transfer -Transfers/second!
- **Reliability**
  - How many failures/faults can the RAID system tolerate?
  - The fault model considered: A fault => total disk has failed!

RAID level 0: Striping

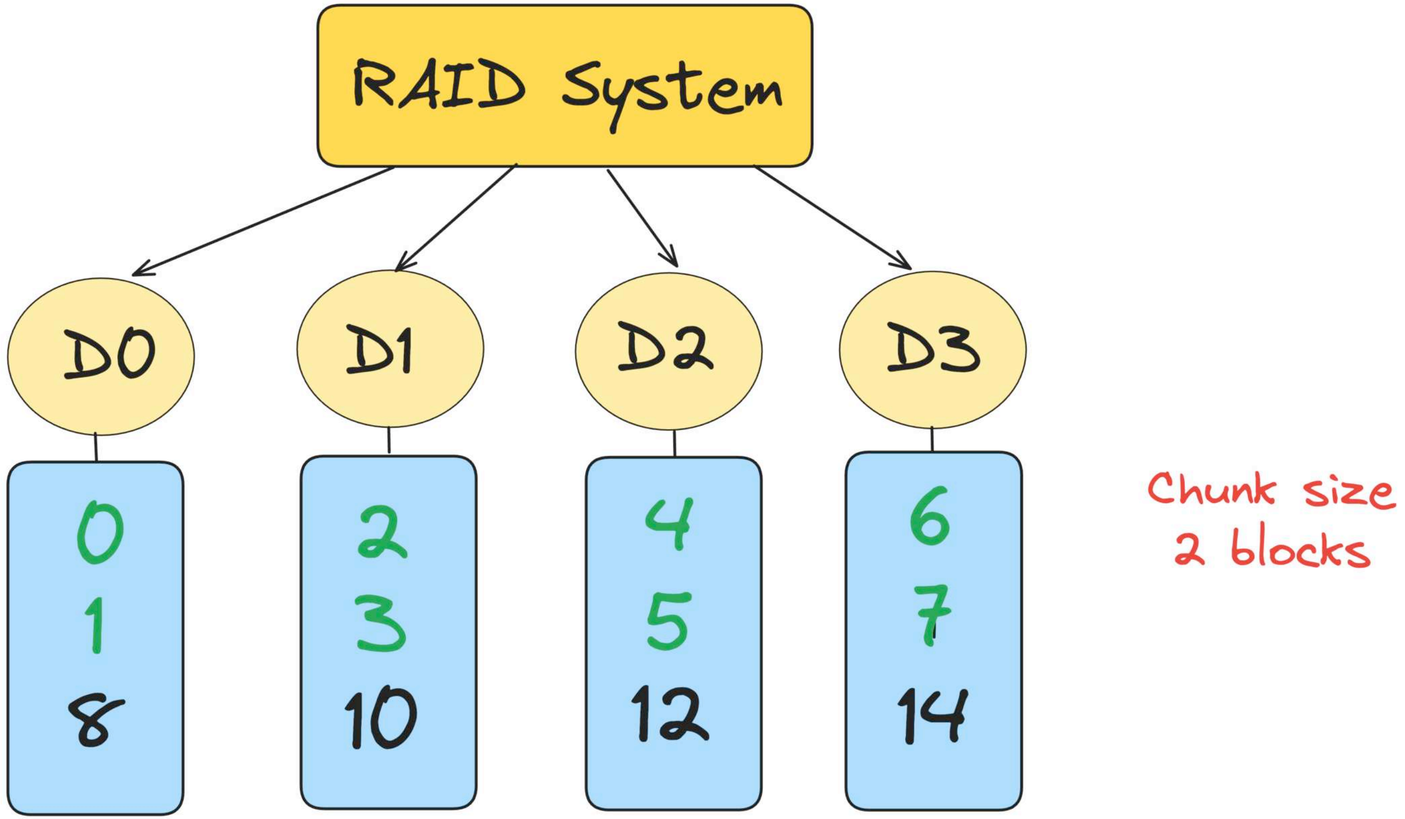


Here blocks 0, 1, 2 and 3 are in same stripe

Each block is of size 4 KB

- **Simple form:** Spread the blocks across the disks in a round robin fashion
- Blocks in the same row - **Stripe**
- No redundancy

RAID level 0: Striping



- Two 4 KB blocks are placed in one disk before moving to another
- Chunk size is 8 KB and a stripe consists of 4 chunks -> 32 KB of data
- Chunk size do have an impact on the performance! - **How?**

RAID Level 0: Impact of Chunk Size

- **Small chunk size**
  - Many files will get stripped across disks
  - Increases parallelisms of reads and writes
  - Positioning time to access blocks across disks increases
- **Big chunk size**
  - Reduces intra-file parallelism, relies on multiple concurrent request to achieve high throughput
  - Large chunk size reduces positioning time (One file in one disk) same as using one disk
- **Best chunk size is hard to get** - Depends on the workload!

RAID Level 0: Performance Analysis

- Two main things to evaluate:
  - **Single-request latency:** latency of single I/O request to RAID
  - **Steady-state throughput:** Total bandwidth of concurrent requests
- Two main workloads:
  - **Sequential:** Request to disk arrive in large contiguous chunks
  - **Random:** Each request is small to a random location on disk
- Assume disk transfers at **S MB/s** under sequential and **R MB/s** under random

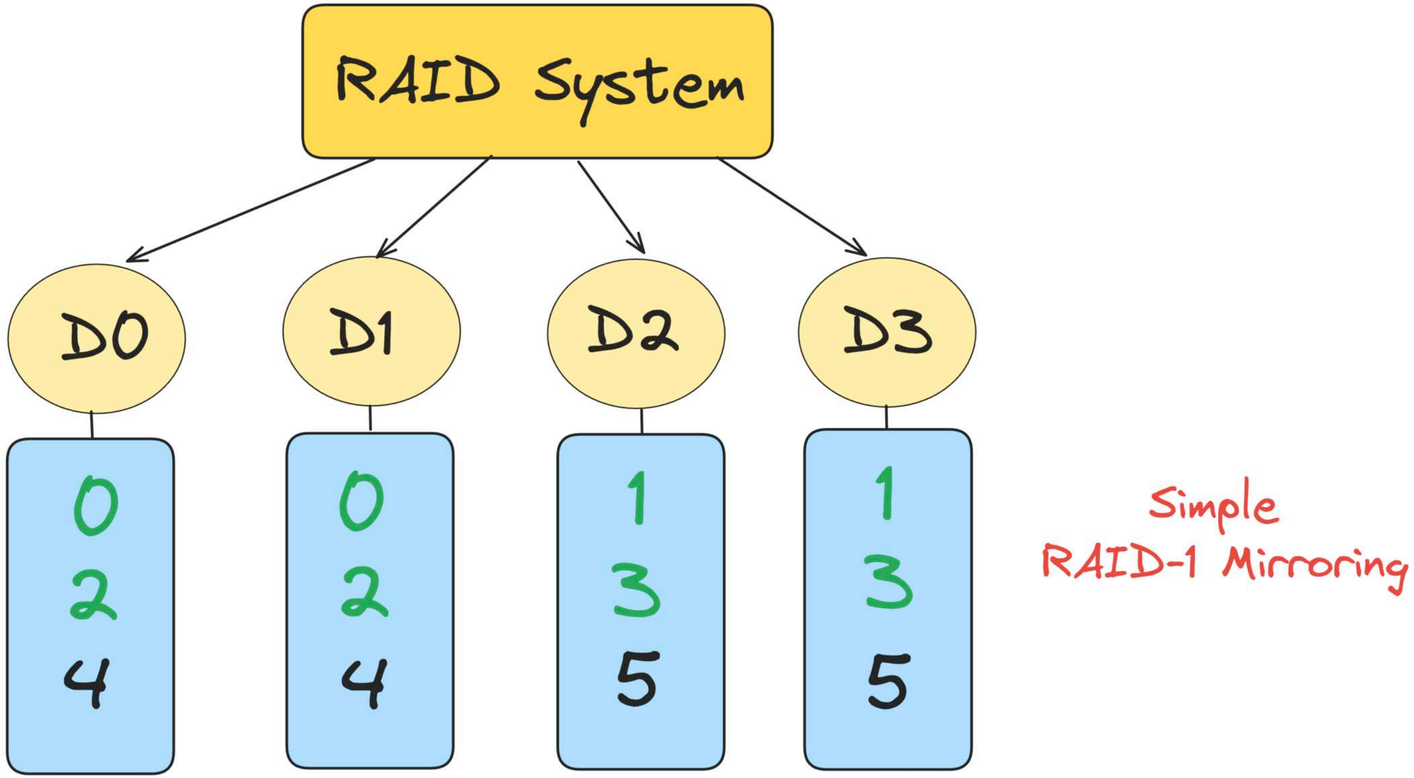
RAID Level 0: Performance Analysis

- Consider the following disk characteristics
  - Sequential transfer of size 10 MB on average
  - Random transfer of size 10 KB on average
  - Average seek time 7 ms
  - Average rotational delay 3 ms
  - Transfer rate of disk 50 MB/s
- How to calculate **S** and **R**?

RAID Level 0: Analysis

- 7 ms spend seeking and 3 ms spend in rotation => total: 10 ms
- 10 MB @ 50MB/s => 200 ms for transfer => total: 200 + 10 = 210 ms
- S = 10 MB / 210ms = 47.62 MB/s
- For R, 10 KB @ 50 MB/s => 0.195 ms => total: 10 +. 0.195 = 10.195 ms
- R = 10 KB / 10.195 ms = 0.981 MB/s
- Steady-state throughput equals N*S MB/s or N*R MB/s depending on workload
- RAID 0 is more like an upper bound

RAID level 1: Mirroring



- Copies are made, each copy is placed in a different disk - Handle failures!
- Data is stripped across mirrored pairs

RAID Level 1: Mirroring

- **Read**
  - When reading from a block, RAID has a choice!
  - Assume a read comes to 0, the system can either use Disk 0 or 1
- **Write**
  - No choice exists, the write needs to happen in both copies of data
  - This promotes reliability, writes can happen in parallel

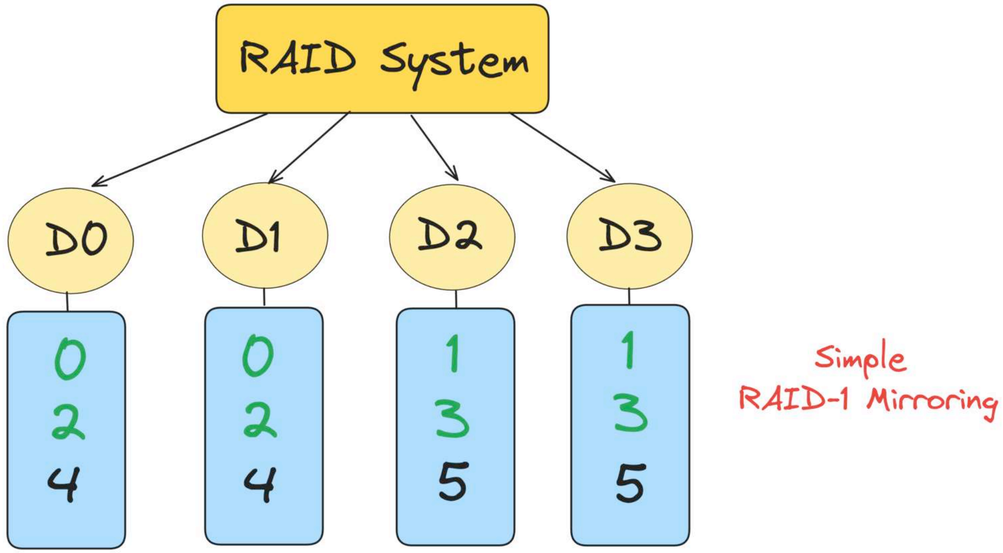
RAID 1: Analysis

- **Capacity**, with all replicated, achieved capacity: N/2
- **Reliability**, RAID 1 can tolerate failure of 1 disk
- **Performance**
  - For single read request, RAID-1 just needs to redirect to one of the copies
  - Write is little different: Two writes needs to happen and it will happen in parallel => time will be almost equal to single write
  - But, due to worst case rotational of two requests, it will be higher than write to a single disk

RAID 1: Analysis

- Steady state throughput
  - Bandwidth during sequential write is (N/2) * S MB/s or half the peak
    - Each write involves writing in two different locations
  - Sequential reads also has a similar bandwidth:
    - Consider reads that needs to be done on blocks: 0,1, 2, 3, 4, 5, 6, 7
    - What will be the bandwidth or steady state throughput in this case?

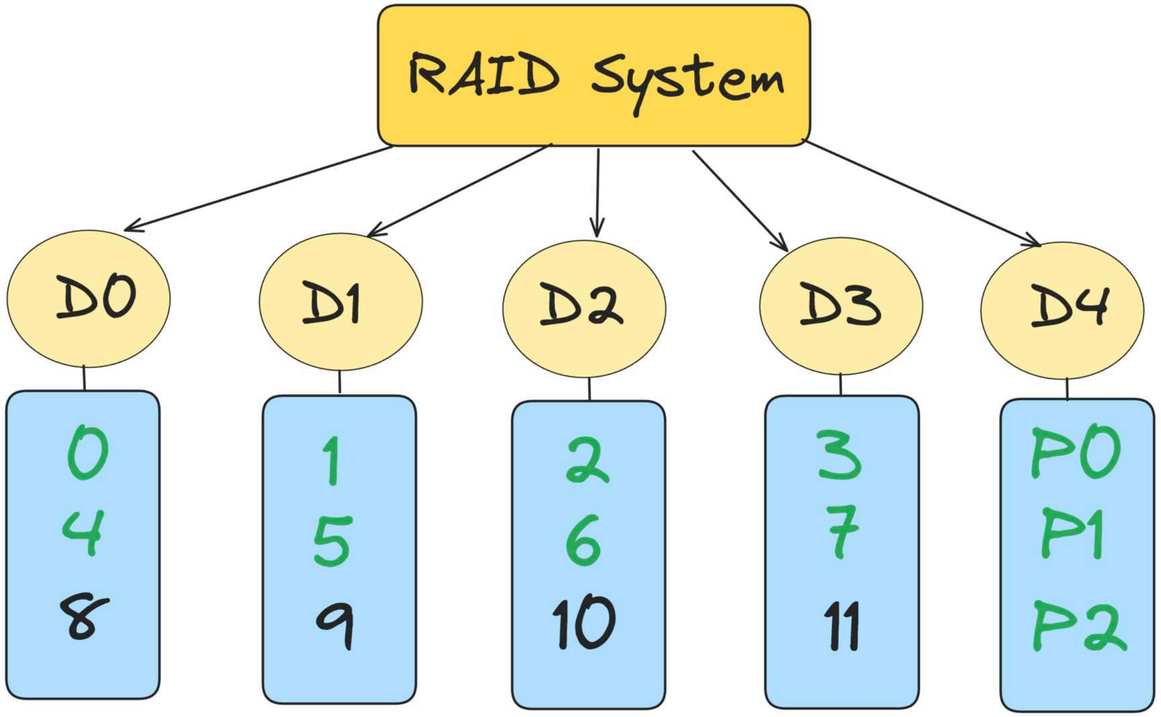
RAID 1: Analysis



Redundancy is good but can we do better?

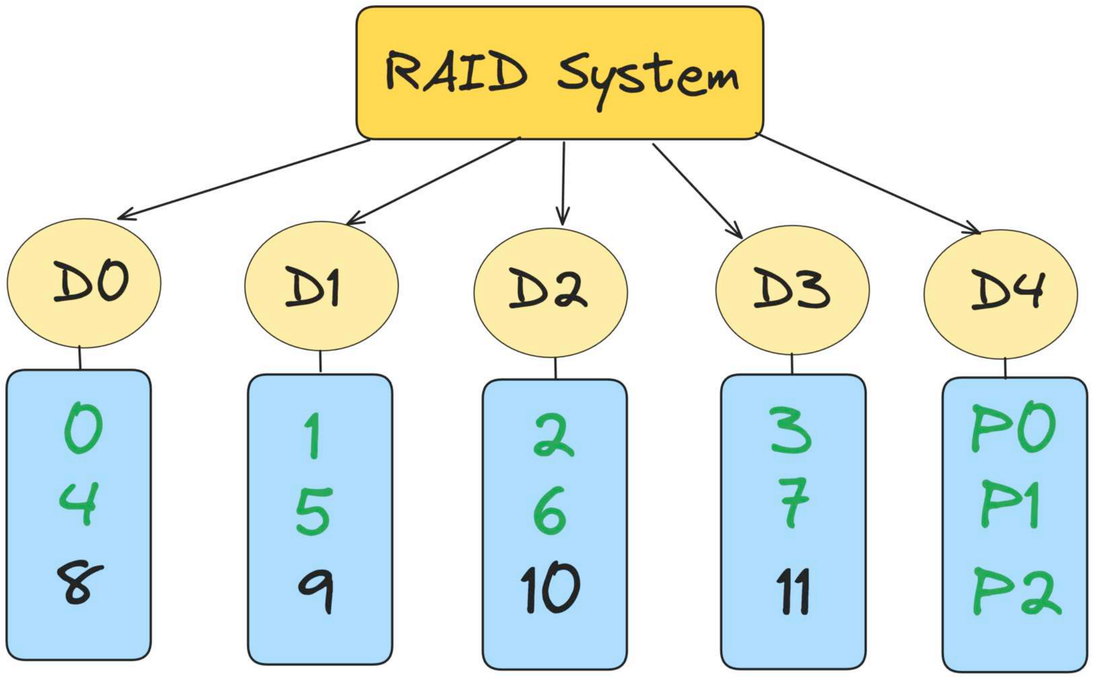
- 0 is send to D0, 1 to D2, 2 to D1, 3 to D3….
- 0 comes to D0 then next one is 4, 2 is skipped (since it goes to D1)
- Simply keeps rotating without doing useful transfer (as D1 is taken care)
- Each disk will only deliver half the peak bandwidth, (N/2) * S MB/s for **Sequential reads**
- **Random reads** N*R and write (N/2) * R MB/s

RAID Level 4: Introducing Parity



- Another method for better managing redundancy: **Parity**
- They aim to use less capacity and overcome space issues at cost of performance
- For each stripe of data above, a parity block is added that stores the redundant information for that block

RAID Level 4



- P1 has redundant information that it is calculated from blocks 4, 5, 6 and 7
- To compute parity **XOR** function is used
  - XOR returns 1 if there are odd no of 1's
  - XOR returns 0 if there are even no of 1's
- This allows to identify if there were some faults in any of the disks - **how?**

RAID Level 4

C0	C1	C2	C3	P
0	0	1	1	XOR (0,0,1,1) = 0
0	1	0	0	XOR (0,1,0,0) = 1

- The parity information can be used to recover from failure
- Assume **data in first row of C2 is lost (it was 1)**
  - Read all the other values in the row and reconstruct the answer
  - Without value of C2 (1), XOR (0,0,0,1) = 0; Hence we can find that C2 needs to be 1

RAID Level 4

Block 0	Block 1	Block 2	Block 3	Parity
00	10	11	10	11
10	01	00	1	10

- In the larger context perform bitwise XOR of all the bits
- Perform Bitwise XOR across each bit of data blocks
  - Put the result of each bit in the corresponding bit slot in parity block
  - Assume that Block 2 fails
  - Block 2 = (00) XOR (10) XOR (10) XOR (11) = 11

RAID Level 4: Analysis

- **Capacity:** 1 disk is for parity hence (N-1)*B
- **Reliability:** Tolerates 1 disk failure, if more than 1 is lost, no way to recover
- **Performance,** Steady-state-throughput:
  - **Sequential reads:** (N-1)*S MB/s
  - **Sequential writes:** (N-1)*S MB/s (write also parity in parallel, full-stripe write)
    - **Note:** writing to parity at same time is not performance gain for client! Hence N-1
  - **Random read:** (N-1)*R MB/s
  - **Random writes?**

RAID Level 4: Analysis

- Main operations involved in write, especially random write:
  - Update a block + update of parity
- Method 1: **Additive Parity**
  - Read in all of the other blocks in that stripe
  - XOR those blocks with the new block
  - **Problem:** As number of blocks increase, this can be challenging, reading of all blocks to perform XOR

RAID Level 4: Analysis

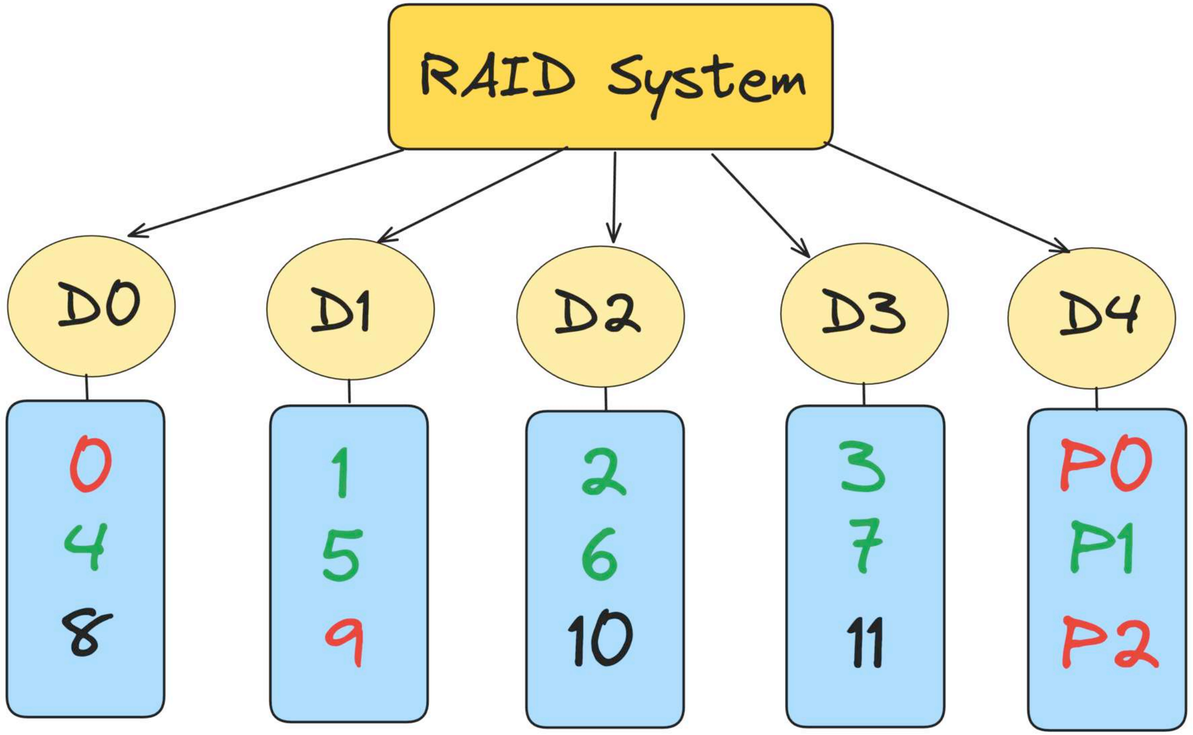
- Method 2: **Subtractive Parity**

C0	C1	C2	C3	P
0	0	1	1	XOR (0,0,1,1) = 0

  - Update C2(old) -> C2 (new)
  - Read old data in C2 (C2(old)=1) and old data in parity (P(old) = 0)
  - Calculate *P(new) = (C2(old) XOR C2(new)) XOR P(old)*
    - If C2(new) == C2 (old) -> P(new) = P(old)
    - If C2(new)!=C2 (old) -> Flip the old parity bit

Small-write Problem

- The parity disk can be a bottleneck
  - Example: Writes to 0 and 9
  - Disk 0 and Disk 1 can be accessed in parallel
  - Disk 4 prevents any parallelism

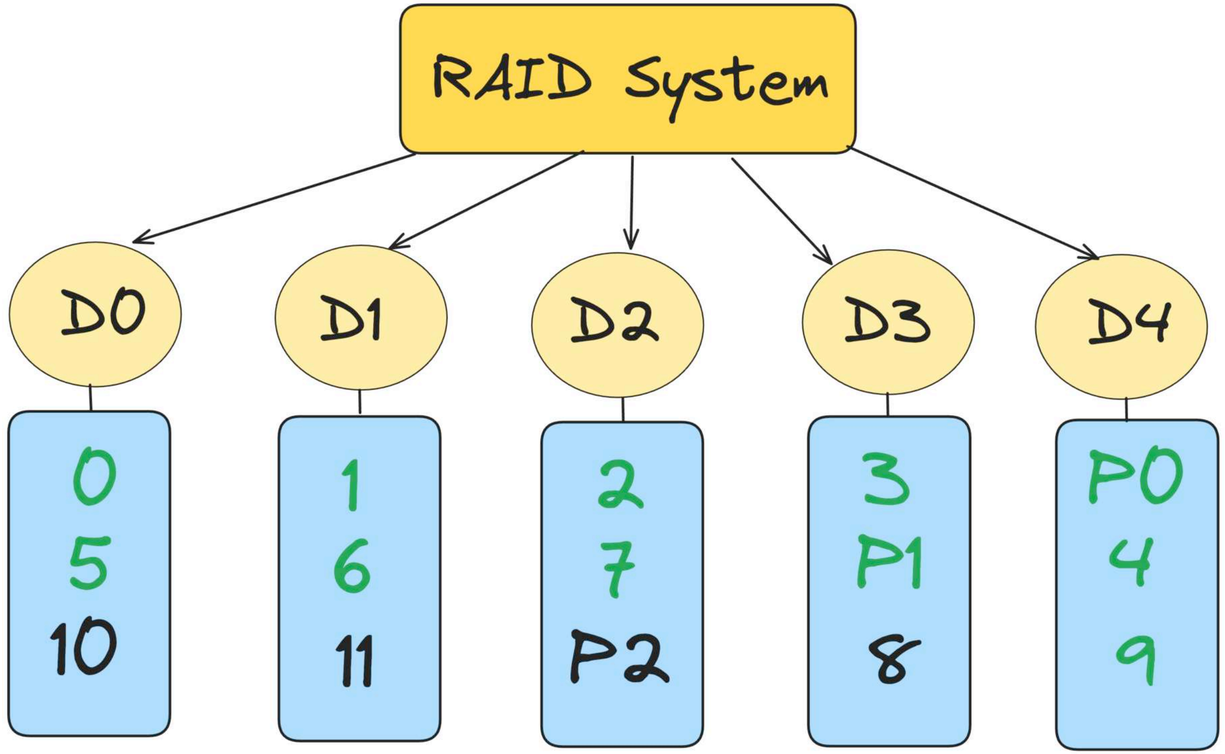


  - RAID-4 under random workload, small writes is (R/2) MB/s - **terrible!**
  - How to improve further?

I/O Latency in RAID-4

- A single read
  - Equivalent to latency of single disk request
- A single write
  - Two reads + Two writes
  - Data block + parity block
  - The reads and writes can happen in parallel
  - Total latency is twice that of single disk

RAID Level 5: Rotating Parity



- Addresses the small-write problem
- Similar to RAID-4 except that keeps rotating the parity block
- Removes the parity-disk bottleneck for RAID-4

RAID-5 Analysis

- **Capacity** and **reliability** identical to RAID-4
- Sequential read and write performance similar to RAID-4
- Random read performance is little better (utilize all disks)
- Random write performance
  - Here the write requests can be parallelized as parity is not bottleneck
  - Given large number of random write requests, all disks can be evenly kept busy, total bandwidth = (N/4)*R MB/s. Still 4 I/O operations (as parity is there)

Summarizing RAIDS

- **Performance** and do not care about reliability -> RAID-0 (Striping)
- **Random** I/O performance and **reliability** -> RAID-1 (Mirroring)
- **Capacity** and **Reliability** -> RAID-5
- **Sequential** I/O and Maximise **Capacity** -> RAID-5

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Persistence: Files and Directories

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

Acknowledgement

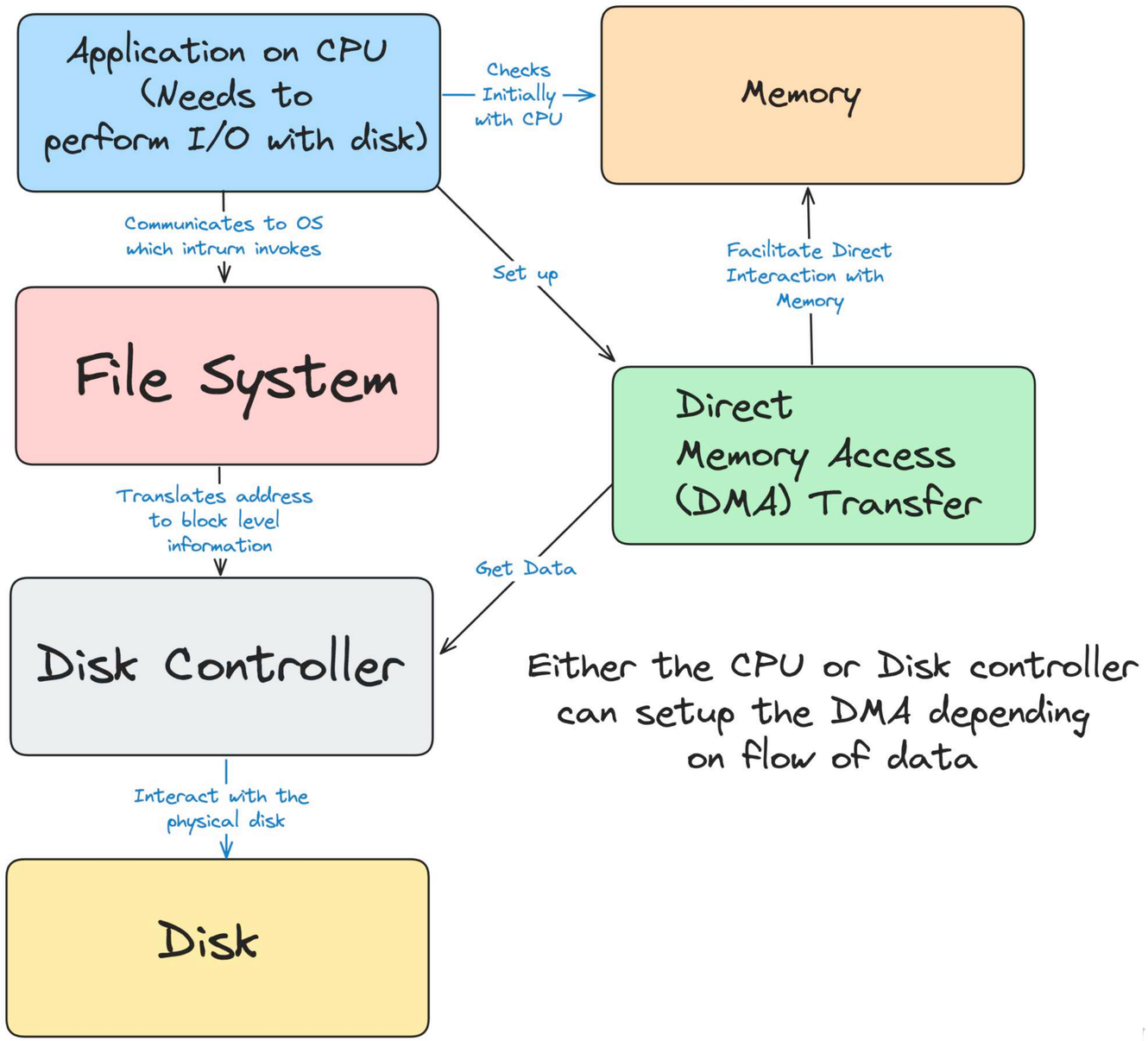
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Operating Systems in Three Easy Pieces by Remzi et al.

The flow of access

- Application performs read or write to a file
- CPU communicates to OS which invokes the File System (FS)
- The OS may check in its cache if its already there
- FS prepares block level information to disk controller
- A Direct Memory Access (DMA) is set up
- Disk controller performs the physical read or write based on commands from DMA and file system
- If its read, Disk -> DMA, for writes, DMA -> Disk



So far!

- **Devices for Persistence**
  - Hard disk - Simple interface, store data in magnetic disks
  - RAIDs provide support for improved capacity, performance and reliability
- **What we still need!**
  - How to manage a persistence device?
  - What about the APIs?
  - What are some key implementation aspects!

Virtualization of Storage

Image source: Dalle-3

Files

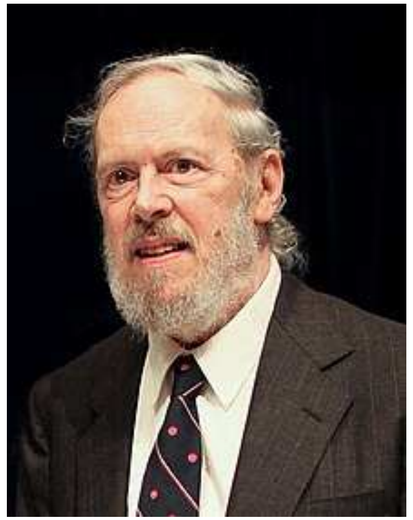
- Linear array of bytes each of which can be read or written
- Each file has a human-readable name - **“sample.pdf”**
- Each file has a unique low-level name (not user given, OS given) - **inode number (i-number)**
- Type of the file is not the concern of the OS (image, code, etc)
  - File system should ensure that data is stored persistently
  - Also ensures that data is retrieved when requested
- Applications can worry about extensions and reading file in the way needed

Directories

- A directory is just like a file
- It also has a low-level name: inode number
- Contains a list of pairs **(user readable file name, i-node number)**
  - Eg: consider a directory name **OSN**
    - (Lectures, 123) -> Directory
    - (OSN_L23.pdf, 326) -> File
- Basically directory is a special type of files with contents: files, directories and corresponding i-node numbers

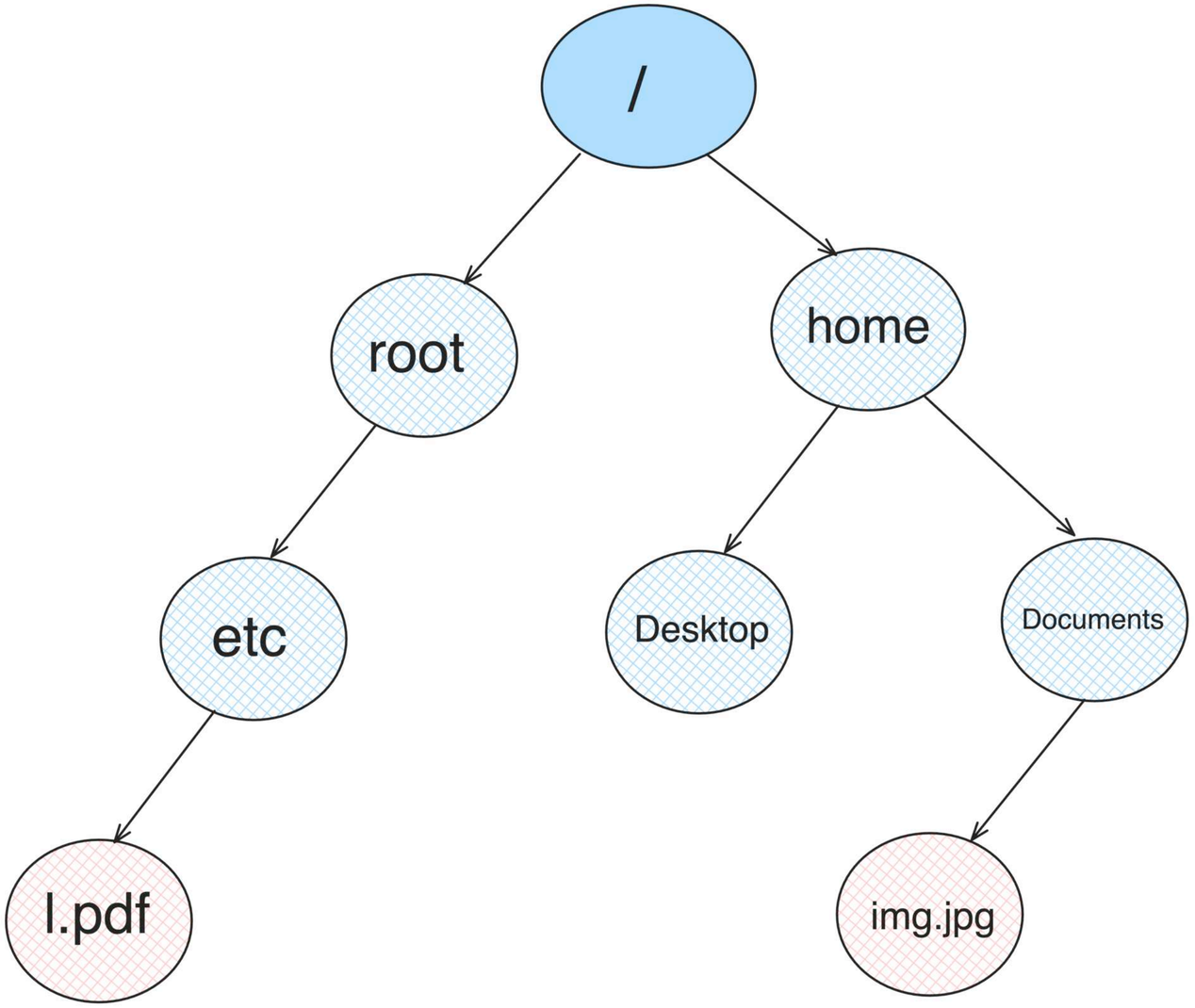
Inode Number - Truth!

“In truth, I don't know either. It was just a term that we started to use. ***‘Index’ is my best guess,*** *because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk…*



Dennis Ritchie

The Unix Directory Tree



- Files and directories arranged in a tree
- Directory hierarchy starts at root directory - referred to as /
- Uses a separator to name subsequent directories
- **Absolute pathname** can be used:
  - /home/Documents/img.jpg
- File has two parts:
  - Arbitrary name - “img”
  - Type - “.jpg”
- Everything is an abstraction by OS

File System Interface

- Everything in Unix is virtually a file
- Mainly the file system has to provide three interfaces
  - **Creation of files** - Support creating files, allocate space
  - **Accessing files** - Reading and writing files
  - **Deletion of files** - Delete files and clear space
- Internally everything is 1s and 0s in the disk so File system has a big responsibility!

Creation Interface

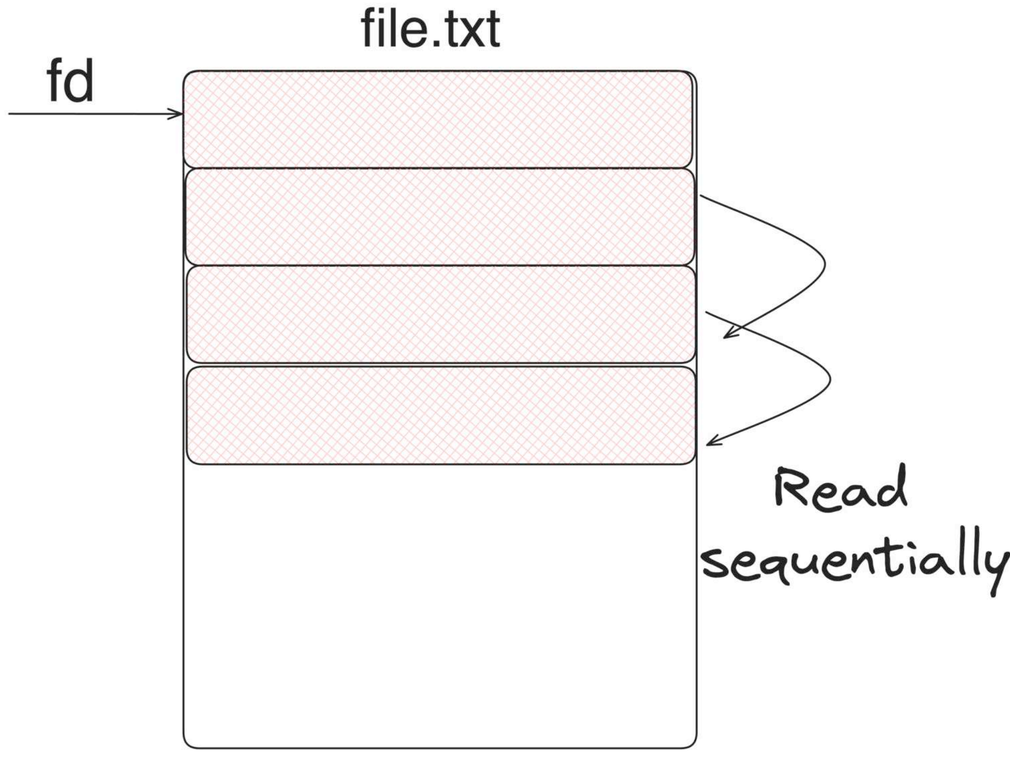
- **open()** system call with flag to create file

int fd = open("sample", O_CREAT | O_WRONLY, O_TRUNC, S_IRUSR | S_IWUSR);

  - O_CREAT: creates a file if it does not exist
  - O_WRONLY: file is write only
  - O_TRUNC: truncates file to zero bytes if it already exists
  - S_IRUSR or S_IWUSR: permissions - make file readable or writeable
- The call returns a number, **file descriptor:** operations on file uses the file descriptor
- Existing files must be opened before they can be read or written
- **close():** closes the file

Access Interface

- *read () / write ()* system calls: Reading/writing files
  - Three arguments: file descriptor, buffer with data, size
  - Buffer - where data will be placed and size - size of buffer
  - Reading and writing happens sequentially by default
  - Successive read/write calls fetches from the offset that is being used
- Every process has three files opened - stdin, stdout, stderr with fd 0, 1 and 2



Random Reading and Writing

- In general file is accessed sequentially
  - Read/write from beginning to end
- What if it needs to be randomly accessed for read/write?
  - *lseek()* system call - seek to random offset
  - Start reading and writing from random offset
  - ***off_t lseek(int flides, off_t offset, int whence);***
    - flides - file descriptor
    - off_t - moves pointer to a given offset,
    - Whence - determines how seek is performed (from an offset, from given + some offset or size of file + offset

lseek has nothing to do with disk seek!

A Simple Example - Normal Read

System calls	Return Code	Current Offset
fd = open("file.txt", O_RDONLY);	3	0
read (fd, buffer, 100);	100	100
read (fd, buffer, 100);	100	200
read (fd, buffer, 100);	100	300
read (fd, buffer, 100);	0	300
close(fd);	0	-

- Offset is initialised to 0 when opened
- For each read call, the offset is incremented fixed value - sequentially
- At the end, 0 denotes the read has been completed

A Simple Example - Seeking

System calls	Return Code	Current Offset
fd = open("file.txt", O_RDONLY);	3	0
lseek(fd, 200, SEEK_SET);	200	200
read (fd, buffer, 50);	50	250
close(fd);	0	-

- Offset is initialised to 0 when opened
- lseek sets the offset to 200
- Read call, reads the next 50 bytes and updates offset

There is a buffer - How to write immidiately?

- Regular writes, write() puts the data to buffer => some point it will be written to persistent storage
- This is done for performance enhancement (keep in buffer for 5 to 30 seconds)
- Some applications require more real-time guarantees
- System call: *fsync(int fd)*: returns 0, once write is complete
- Sometimes fsync has to be called on directory itself that contains the file
  - This ensures that file is on disk

fsync example

```
#include <stdio.h>


int main ()
{
  int fd = open("sample", ...);
  assert (fd > -1);
  int rc = write (fd, buffer, size);
  assert (rc == size);
  rc = fsync(fd);
  assert(rc == 0);
  return 0;
}
```

Metadata of files

- File system stores fair amount of data about files
- Information include: file size, last access, last modified, user id of the owner, links count, pointers to data blocks, etc.
- This metadata is stored by file systems in a structure called **inode**
- **Inode** - persistent data structure used by the file system
  - They store all the metadata information for a file
  - They are stored in the disks but copies are cached to main memory when needed!

Interface for Directories

- Directories can also be accessed like files
  - Operations like create, open, read, close
- Create directory - **mkdir()** system call, when created its empty. It has two entries
  - “.” And “..” Itself and the parent directory respectively
- Listing all the directories - **ls** command (internally - opendir(), readdir() and closedir())
- What about **rm * ** and **rm -rf * ** ? - Powerful double-edged sword!
- Directory entry contains information such as name, I-node number,
- Deleting directory - **rmdir ()** - System call and command have same name

Hard Links

- Hard linking creates another file that points to the **same i-node number** (hence same underlying data)
- Assume a file, “file1” which just contains a string “test” - What if we need file2 linked to this?
- Another file that links to this can be created using **link()** call - **ln** command
- Essentially both files have same underlying data - just two different user-given names
- I-node maintains a **link count,** file deleted only when no further links to it
- One can only unlink file, OS decides when to delete

Hard links

```
prompt> echo hello > file1
prompt> cat file1
hello OSN Students
prompt> ln file1 file2
prompt> cat file2
hello OSN Students
```

Hard links

```
prompt> rm file1
removed "file1"
prompt> cat file2
hello OSN Students
```

Symbolic Links or Soft Links

- Another way to create link - This time in much simpler
  - Hard links are limited - link to directory not possible
  - Hard link to files in other disk not possible
  - I-node is unique within a file system
- **Symbolic link or soft link** creates a file by itself
  - The name can be different
  - **i-node number will be different**
  - If the main file is deleted, link points to an invalid entry: **dangling reference**

Soft Links

```
prompt> echo "Hello OSN" > file1
prompt> cat file1
Hello OSN
prompt> ln -s file1 file2
prompt> cat file2
Hello OSN
prompt> rm file1
prompt> cat file2
cat: file2: No such file or directory
```

Beyond Files and Directories

- Mounting a file system connects the files to specific point in the directory tree

mount -t ext3 /dev/sda1 /home/users

ls /home/users

- Assembling directory tree from underlying file system
  - Accomplished by mounting the file system
  - Two tasks: **making the file system** and **mounting**
- Several devices and file systems are mounted on a typical machine
  - Can be accessed with mount command

How can we build a simple File System?

What structures are needed in disk and how to access?

File System

- Organization of files and directories on disk
- OS has one more file systems
- File system is **pure software,** features:
  - Provide support for the sys calls
  - Manage the storage of data
  - No additional hardware support
- Great deal of **flexibility** when building FS
- Details vary with various file systems

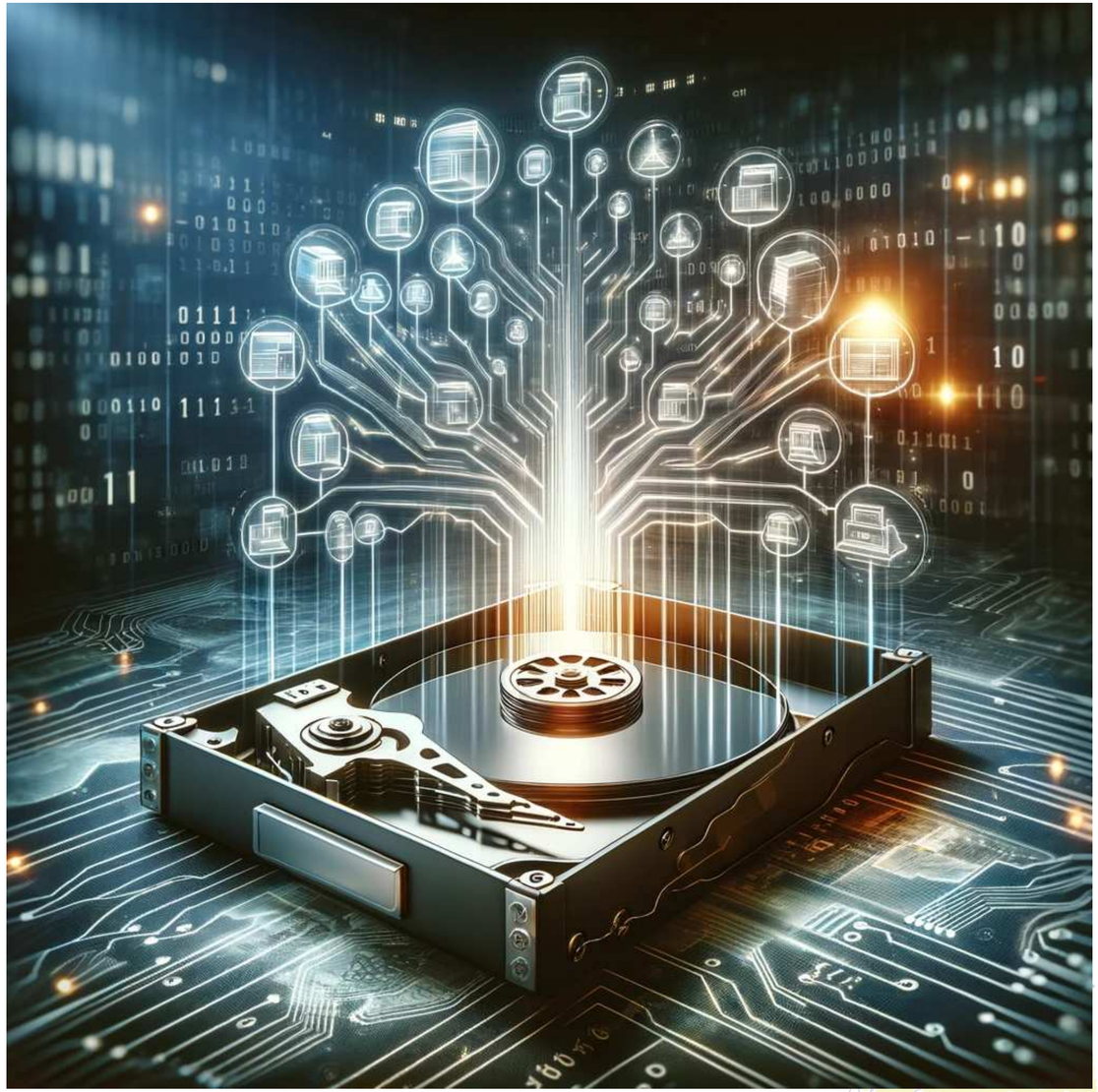


Image credits: Dalle-3

Software Engineering Research Centre

Breaking down into two main aspects

- Lets try building a simple file system - **Very Simple File System (VSFS)**
- In any FS, two key things make the difference

Data Structures

- What types of on-disk data structures are utilized by the file system to organise its data and metadata?
- VSFS can make use of simple structures like array of blocks (complex ones: trees)

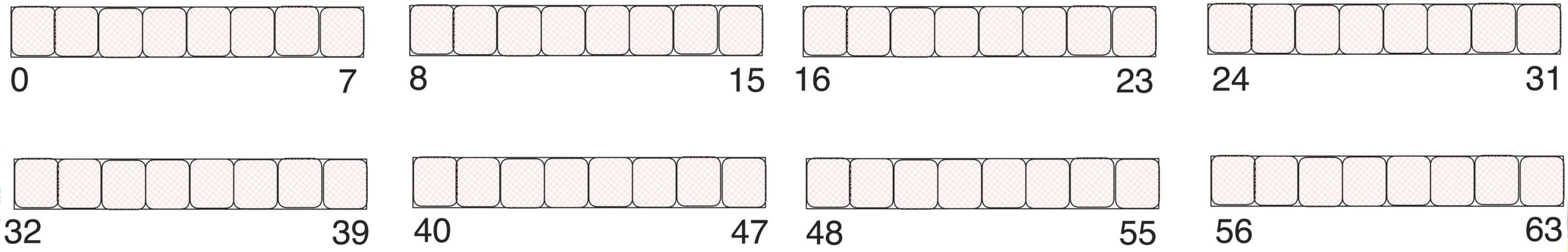
Access Methods

- How can the calls like open(), read(), write(), etc made by process be mapped?
- Which structures are read during the execution of a system call?
- What about the efficiency?

Data structures

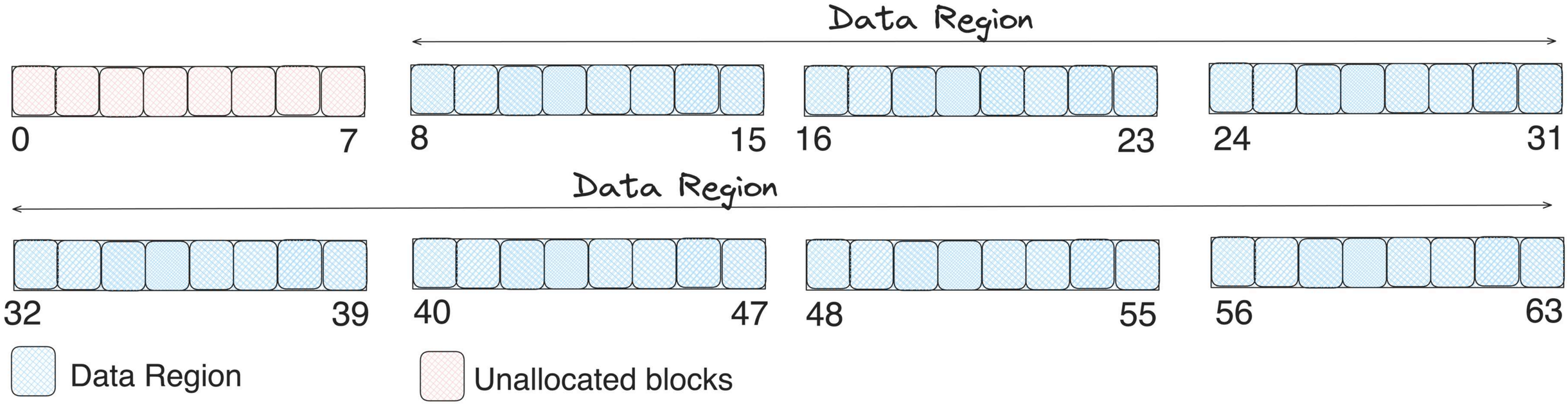
On-disk organisation of VSFS

- Remember: Disk exposes a set of **blocks**
- File system has to organise the files into blocks - **Data**
- The information about the files also have to be stored - **metadata**
- Consider a disk with 64 blocks, each of size 4 KB (same sized blocks)
  - 0 to 63 in general **0 to N-1**
  - **What needs to be stored in these blocks?**



Data Region in the File System

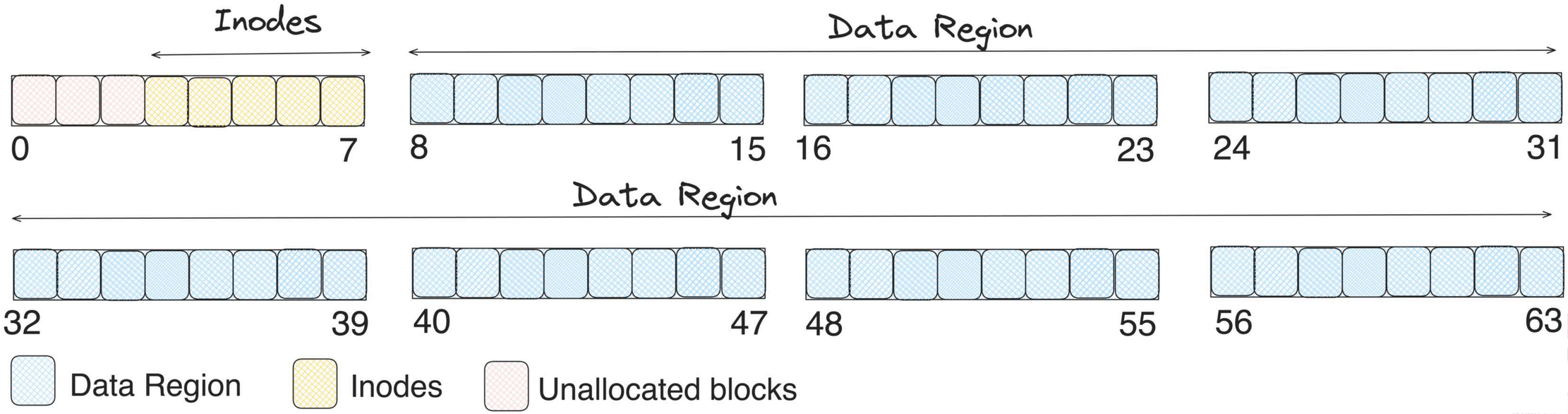
Some blocks needs to be reserved for storing data - **data region**



- More information needs to be stored about where the data blocks are located, type of file, etc
- The inodes need to be stored

Some Space for Inodes!

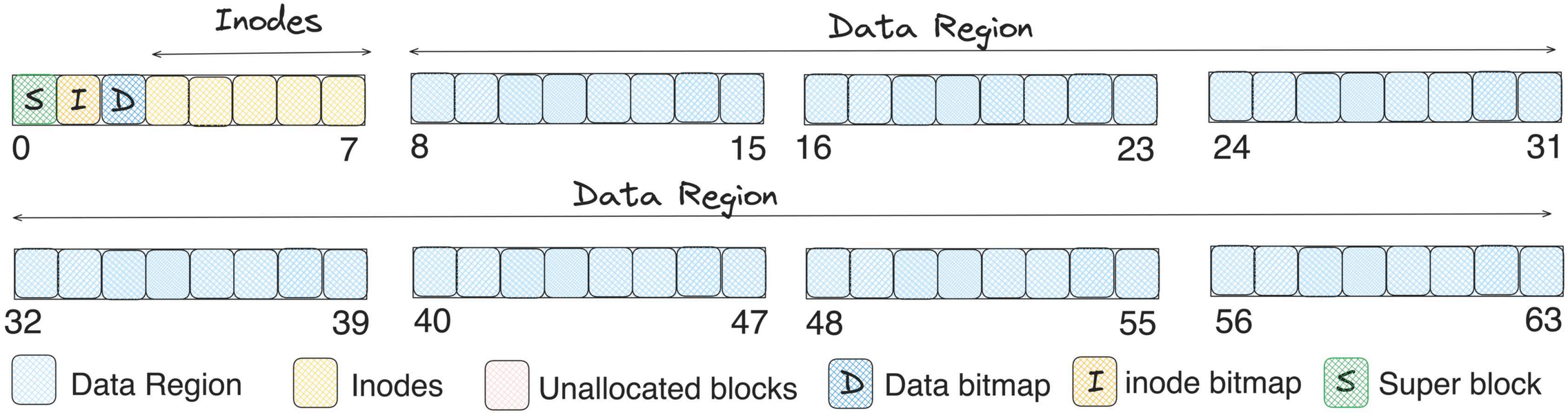
- Dedicate some space for inode table
  - This can hold an array of on-disk inodes
  - Consider each inode takes 256 bytes and 5 blocks are dedicated
  - Each block can hold 16 inodes => file system can hold 80 files



We still miss something!

- FS needs some mechanism to track which inodes are free and which data blocks are free
- How can such information be tracked? Which are free and which are available?
  - Use **bitmaps**, each bit can be used to denote if corresponding block is free or not
    - 0 if the corresponding block is free
    - 1 if the corresponding block is allocated
  - In our vsfs - 80 inodes and 56 blocks for data
  - Assume that we dedicate **two blocks for bitmaps** for **inode** and **data**

A more complete representation



- **Super block** holds the entire organisation of all other blocks
  - Which blocks are inodes, which are data blocks, where does data block start, where Inode begins, type of file system, etc
  - During the mount, OS reads super block to initialise various parameters

SCAN ME

Thank you

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

CS3.301 Operating Systems and Networks

Persistence: File System Implementation

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

Acknowledgement

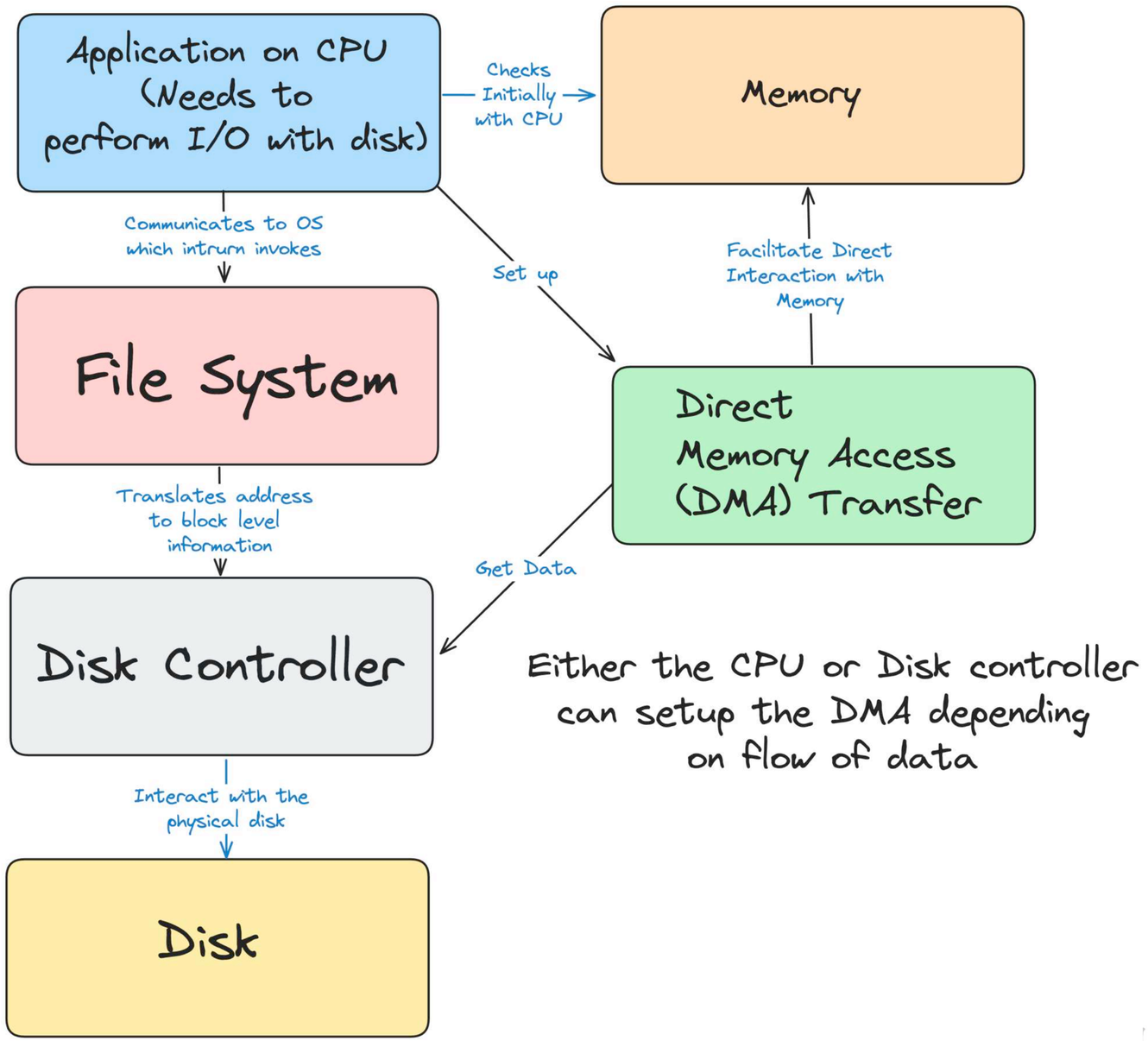
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

Sources:

- Operating Systems in Three Easy Pieces by Remzi et al.
- File System implementation by Youjip Won, Hanyang University

The flow of access

- Application performs read or write to a file
- CPU communicates to OS which invokes the File System (FS)
- The OS may check in its cache if its already there
- FS prepares block level information to disk controller
- A Direct Memory Access (DMA) is set up
- Disk controller performs the physical read or write based on commands from DMA and file system
- If its read, Disk -> DMA, for writes, DMA -> Disk



Virtualization of Storage

Image source: Dalle-3

Metadata of files

- File system stores fair amount of data about files
- Information include: file size, last access, last modified, user id of the owner, links count, pointers to data blocks, etc.
- This metadata is stored by file systems in a structure called **inode**
- **Inode** - persistent data structure used by the file system
  - They store all the metadata information for a file
  - They are stored in the disks but copies are cached to main memory when needed!

How can we build a simple File System?

What structures are needed in disk and how to access?

File System

- Organization of files and directories on disk
- OS has one more file systems
- File system is **pure software,** features:
  - Provide support for the sys calls
  - Manage the storage of data
  - No additional hardware support
- Great deal of **flexibility** when building FS
- Details vary with various file systems

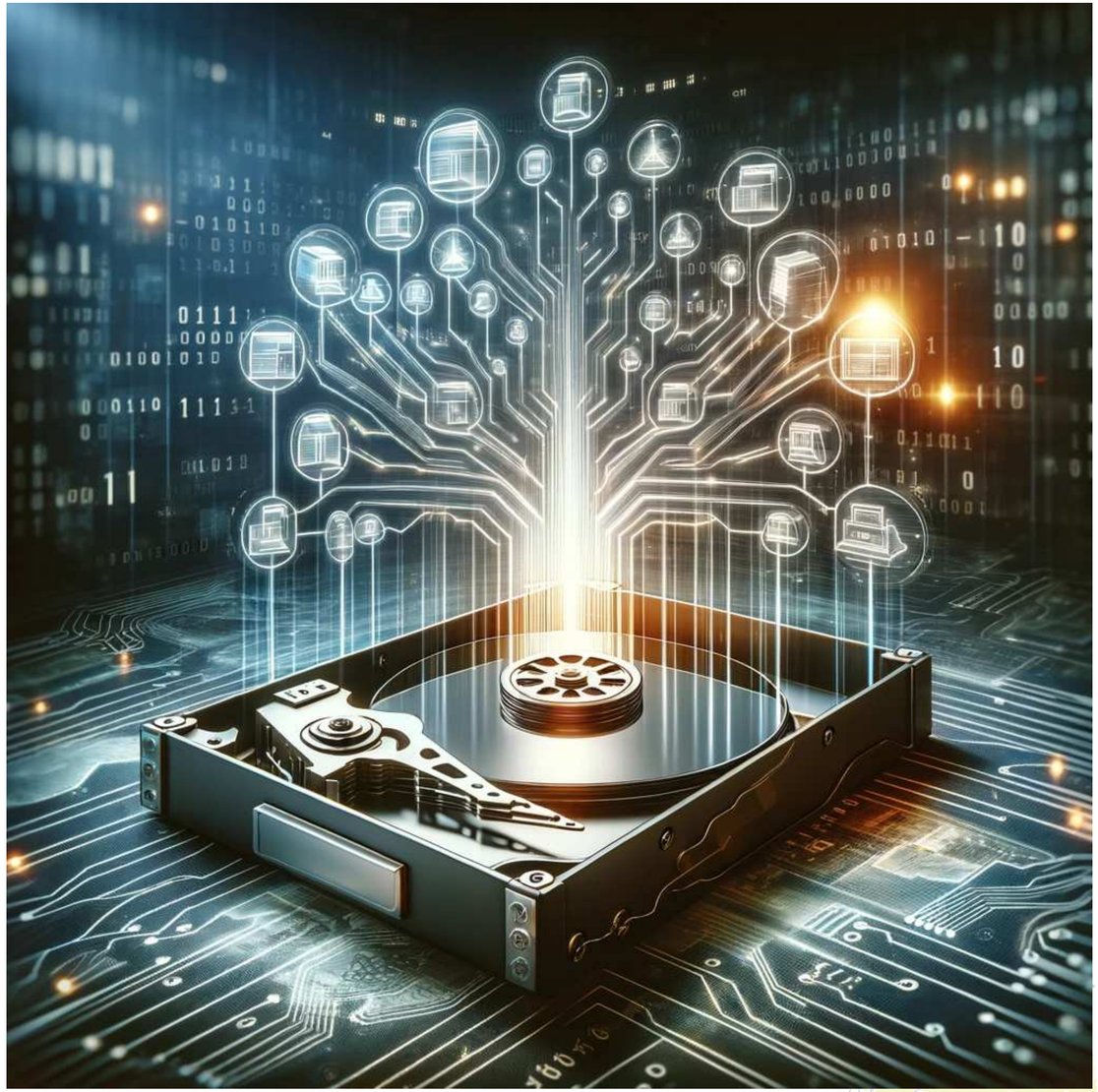


Image credits: Dalle-3

Software Engineering Research Centre

Breaking down into two main aspects

- Lets try building a simple file system - **Very Simple File System (VSFS)**
- In any FS, two key things make the difference

Data Structures

- What types of on-disk data structures are utilized by the file system to organise its data and metadata?
- VSFS can make use of simple structures like array of blocks (complex ones: trees)

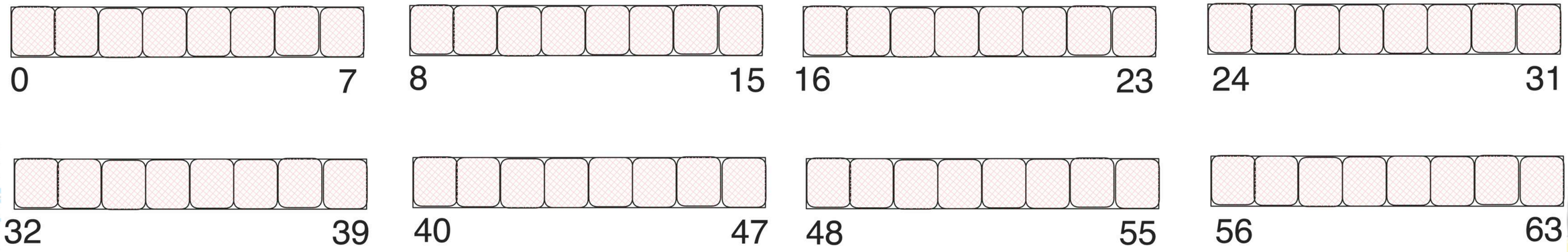
Access Methods

- How can the calls like open(), read(), write(), etc made by process be mapped?
- Which structures are read during the execution of a system call?
- What about the efficiency?

Data structures

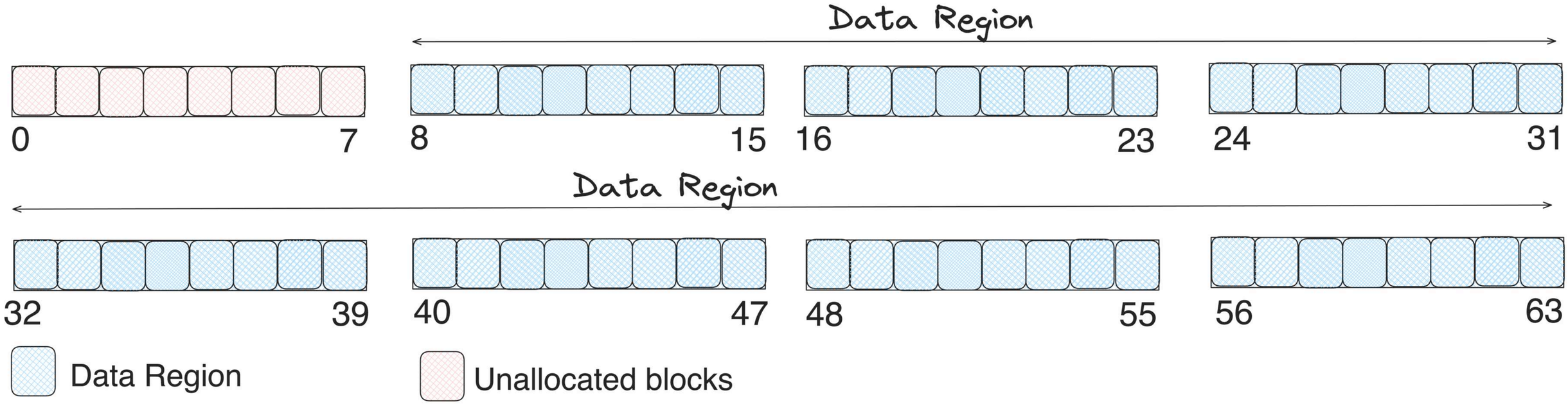
On-disk organisation of VSFS

- Remember: Disk exposes a set of **blocks**
- File system has to organise the files into blocks - **Data**
- The information about the files also have to be stored - **metadata**
- Consider a disk with 64 blocks, each of size 4 KB (same sized blocks)
  - 0 to 63 in general **0 to N-1**
  - **What needs to be stored in these blocks?**



Data Region in the File System

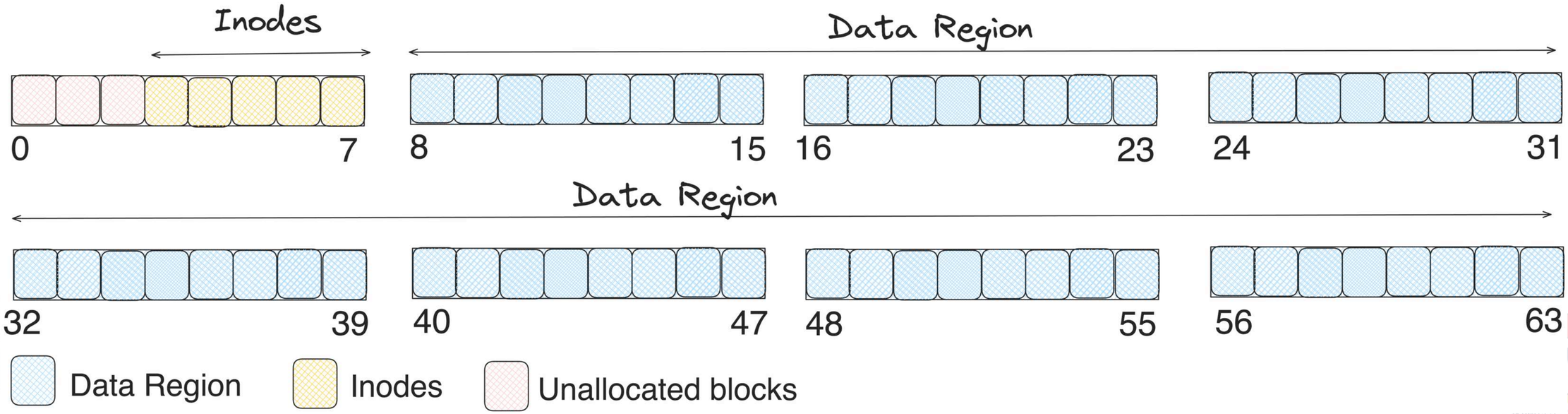
Some blocks needs to be reserved for storing data - **data region**



- More information needs to be stored about where the data blocks are located, type of file, etc
- The inodes need to be stored

Some Space for Inodes!

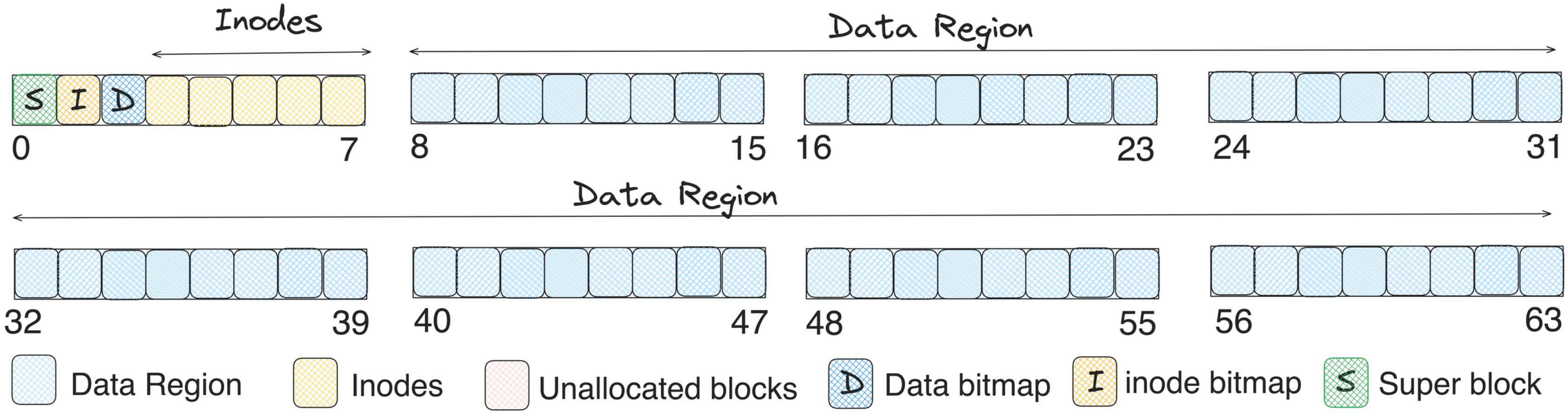
- Dedicate some space for inode table
  - This can hold an array of on-disk inodes
  - Consider each inode takes 256 bytes and 5 blocks are dedicated
  - Each block can hold 16 inodes => file system can hold 80 files



We still miss something!

- FS needs some mechanism to track which inodes are free and which data blocks are free
- How can such information be tracked? Which are free and which are available?
  - Use **bitmaps**, each bit can be used to denote if corresponding block is free or not
    - 0 if the corresponding block is free
    - 1 if the corresponding block is allocated
  - In our vsfs - 80 inodes and 56 blocks for data
  - Assume that we dedicate **two blocks for bitmaps** for **inode** and **data**

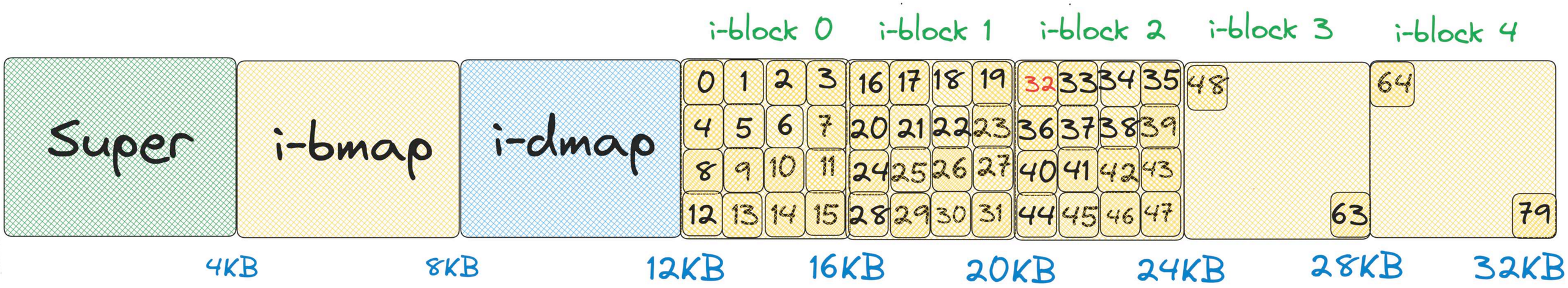
A more complete representation



- **Super block** holds the entire organisation of all other blocks
  - Which blocks are inodes, which are data blocks, where does data block start, where Inode begins, type of file system, etc
  - During the mount, OS reads super block to initialise various parameters

File Organization: The inode

- Each inode is referred to by the inode number
  - Using inode number, FS can locate inode, eg: inode number: 32
  - Calculate offset into inode: 32 X (sizeof(inode)) = 32 * 256 = **8192 => 8 KB**
  - Add offset with start address of inode = 12KB + 8KB = **20KB**



Software Engineering Research Centre

What does inode contain?

- inode contains all the information about a file - The metadata
  - File type (regular file, directory, etc.)
  - Size, number of blocks allocated to it
  - Protection information (who can access, what access, etc.)
  - Time information (modified time, access time, etc)
  - Many more

Simplified EXT2 inode

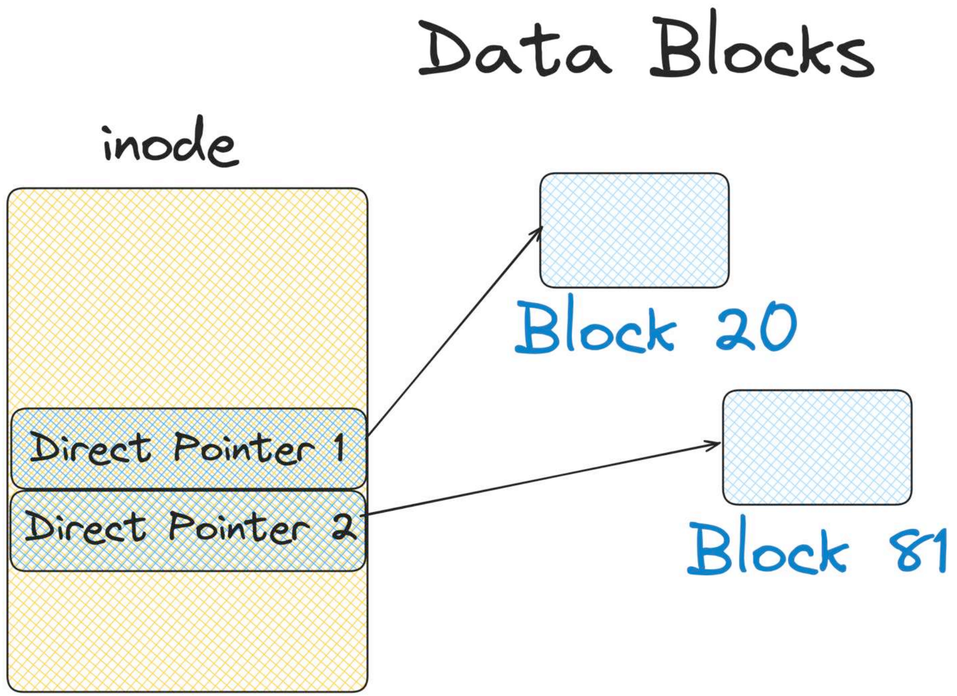
Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
4	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
2	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists
4	faddr	an unsupported field
12	i_osd2	another OS-dependent field

Total 128 bytes

How can inode get to data blocks?

More about inodes

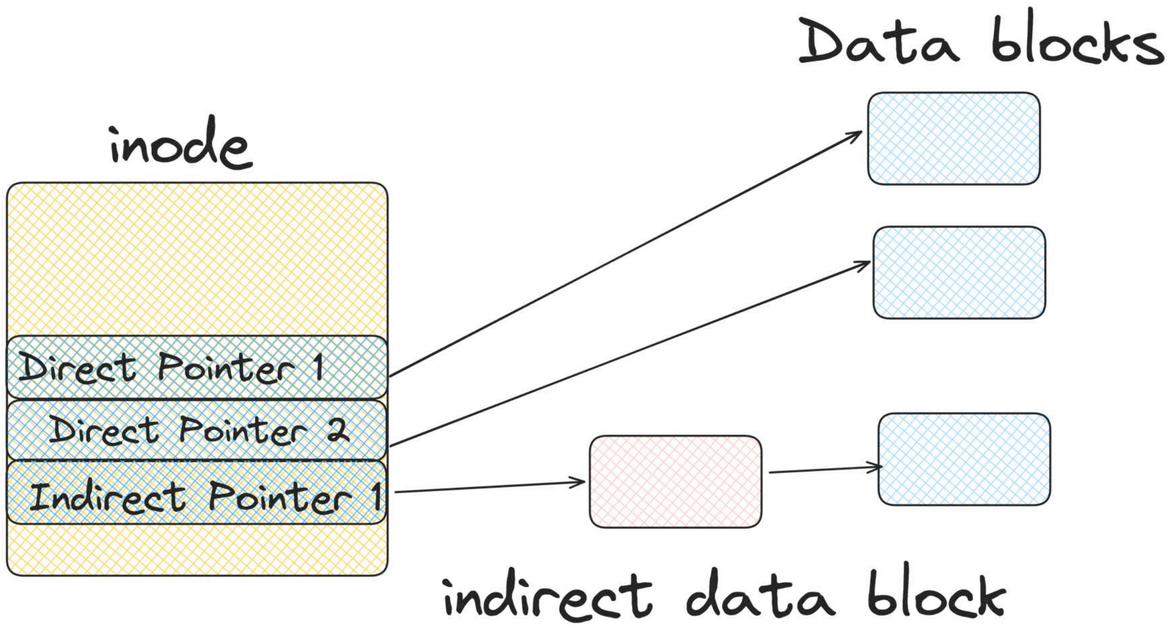
- Each inode needs to track disk block numbers of a file
- File data is not stored contiguously on disk
  - How to track multiple block numbers of a file?
  - Store pointer to the block inside the inode
  - Numbers of first few blocks are stored in inode itself
  - Each pointer can point to the location in the disk block - **direct pointers**
  - **What if the file size is large?** - How many block numbers can i-node store?
    - Need for better mechanism



Size of one block is 4 KB here!

Indirect Pointers

- To support large files, few direct pointers may not suffice!
- Use a special pointer - **indirect pointer**
  - Point to a block that contains more pointers - **indirect data block**
  - Each of the pointer can further point to data blocks
  - The indirect block is allocated from the data region
  - Inode array may have 12 direct pointers and one indirect pointer

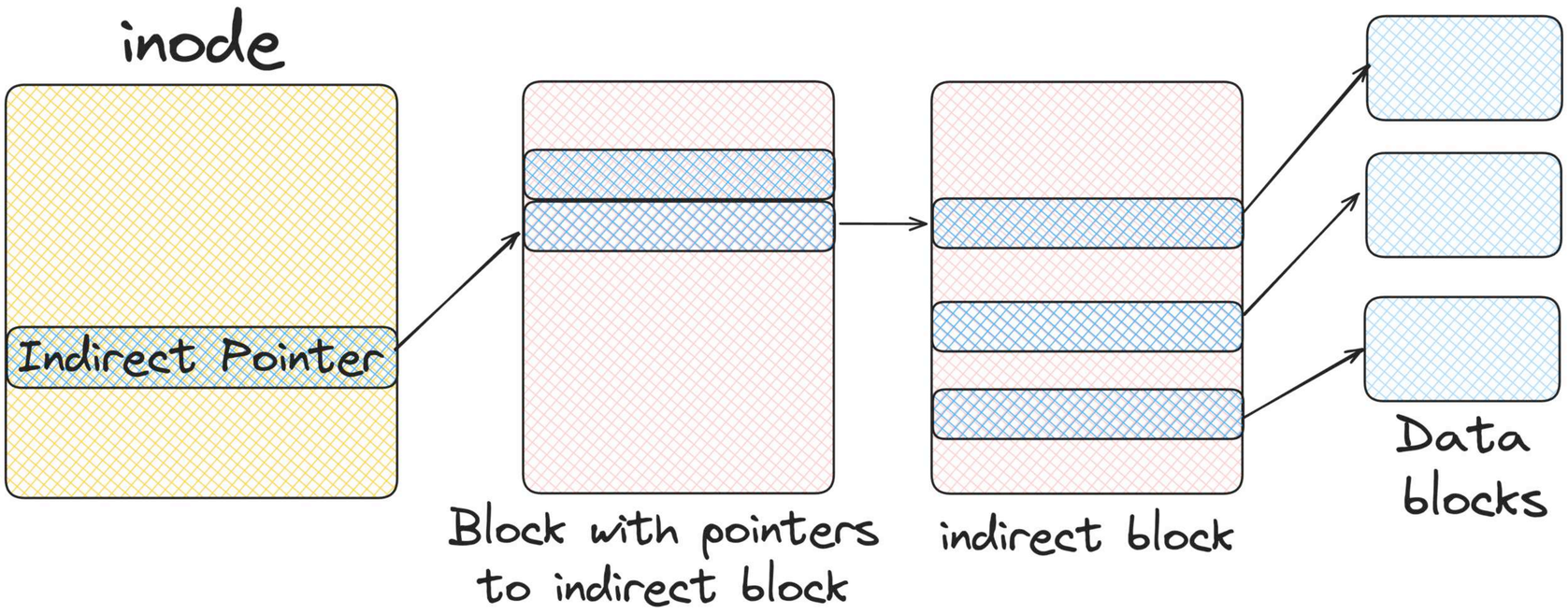


How much files can be supported?

Having one indirect pointer

- Each block is 4 KB
- Each inode can contain 12 direct pointers => 12*4 = 48 KB of file can be addressed
- 1 indirect pointer points to a block of size 4 KB
  - Each address takes around 4 bytes
  - Indirect blocks can have around 1024 pointers (4 KB / 4)
- Total size of file that can be addressed = (12 + 1024) * 4K = 4144 KB
- What if the file is even larger? How can the inode capture all the blocks?

The Multi-Level Index



- **Double indirect pointer**: Points to a block with pointers to indirect block
  - Each of the pointers in indirect block points to data blocks
  - Size now that can be supported is 1024*1024*4 ~ 4GB
- For more even **triple indirect pointers** can be sought of

Why this direct and indirect pointers?

- One finding over many years of research: most of files are small
- Thus with small number of direct pointers, inode can point to 48 KB of data
- All that is needed is one or few indirect blocks

Most files are small	~2K is the most common size
Average file size is growing	Almost 200K is the average
Most bytes are stored in large files	A few big files use most of space
File systems contain lots of files	Almost 100K on average
File systems are roughly half full	Even as disks grow, file systems remain ~50% full
Directories are typically small	Many have few entries; most have 20 or fewer

"A Five-Year Study of File-System Metadata" by Nitin Agrawal, William J. Bolosky, John R. Douceur, Jacob R. Lorch. FAST '07, San Jose, California, February 2007.

What about Directories?

- Directory stores the mapping of file names and their inode numbers
- Each directory has two extra files
  - “.” for current directory and “..” for parent directory
  - Assume that a directory “OSN” has three files (l01, l02, lect03)
- Directory is a special type of file and has inode and data blocks (stores file records)

inum	reclen	strlen	name
5	12	2	.
2	12	3	..
12	12	4	l01
13	12	4	l02
24	36	7	lect03

inum - inode number
reclen - total bytes for name
strlen - length of the name
name - actual name

Free Space Management

- FS has to keep track of which inodes and data blocks are free
- Multiple methods can be used and many design choices exist. Eg:
  - Use **bitmaps** for inodes and data blocks, store one bit per block to indicate free or not
  - **Free list:** Super block can store pointer to first free block which can then point to next free block and so on.
- Eg: Linux FS such as ext2 and ext3 checks for sequence of blocks on new file creation
  - Sequence of data blocks are allocated contiguously for performance
  - Pre-allocation policy is commonly used heuristic when allocating data blocks

Access: Reading File From Disks

- FS also needs better ways of managing access to file (apart from data structure)
- Eg: FS has been mounted and read issued to */OSN/l01* - open, read, close
- Assume that file size is 12 KB (3 blocks in size)
    - sys call open("/OSN/l01", O_RDONLY)
- Intuitively: FS must traverse the pathname and locate the file
    - What will be the process to achieve this?

Opening Files

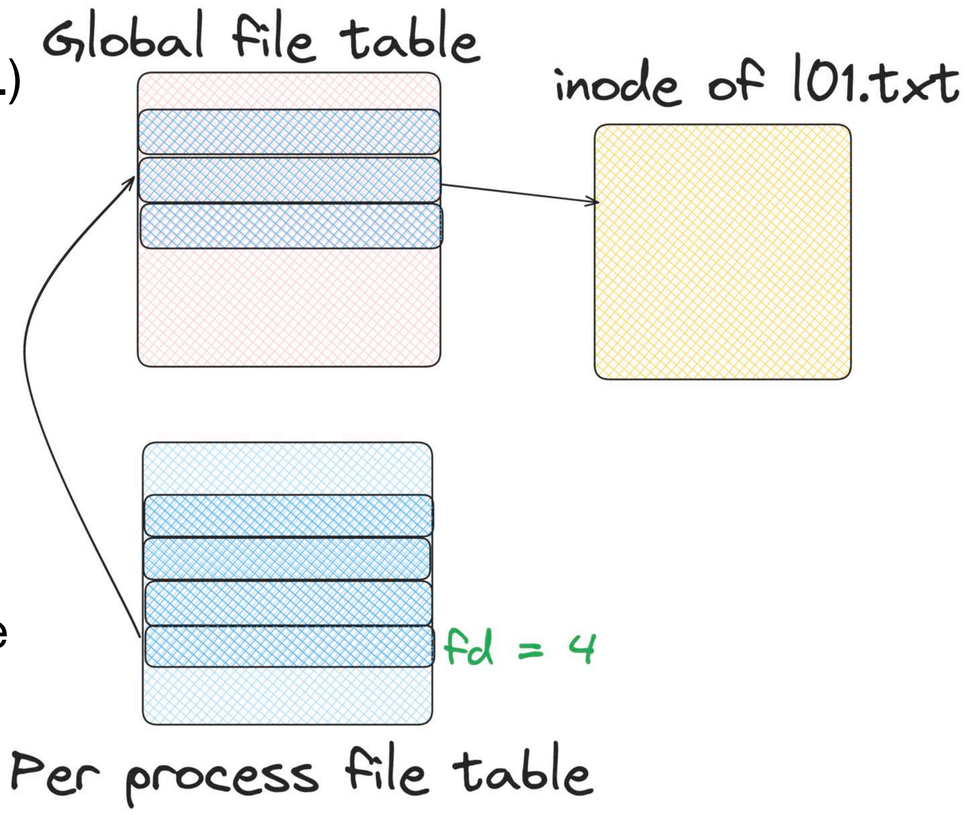
- First part of read is always open sys call - Why?
  - Take the inode and load it in the memory for future operations
  - Open returns file descriptor which points to in-memory I-node
  - Reads and writes can access file data from I-node
- Assume a sys call *open("/OSN/lectures/l01.txt", O_RDONLY)*
  - Traverse the path name and then locate desired inode
  - Begin at the root of the FS (/), root inode number is 2 in Unix FS (mostly)
  - FS reads the block that contains inode number 2

Opening Files

- Recursively: Read the data blocks of root directory, find the name “lectures” and get its inode number
  - Get inode of lectures -> get inode number of “l01.txt” -> get inode
  - Keep repeating the process until the end of the path
- Read inode of “l01.txt” into memory, make final permission check
- Allocate file descriptor for this process and return file descriptor to user
  - Allocation will be done in the in-memory **open file table.** It will be updated for each read - offset
- In the case of new file, new inode and data blocks will be allocated using bitmap and update directory entry

Open File Table

- Kernel uses a set of data structures to track all open files
- **Global open file table**
  - One entry for every open file (stores also sockets, pipes, etc.)
  - Entry points to the in-memory inode of the file (remember opening of file)
- **Per-process open file table**
  - Array of all the files that the process has opened
  - File descriptor is index into the array
  - Per process file entry -> global file table entry -> inode of file
  - Every process has three files (stdin, stdout, err) open by default
- Open system call creates entries in both table and returns file descriptor number



Reading a File

- Make a call read() to read from file
  - Read in the first data block of the file with help of inode
  - Update the inode with last accessed time
  - Update in-memory open file table for file descriptor, file offset
  - Repeat the process for reading each block of data
- Once file is closed
  - Just the file descriptor should be deallocated - No disk I/O

Reading a File From Disk

	data bitmap	inode bitmap	root inode	lectures inode	101 inode	root data	lecture data	101 data [0]	101 data [1]
Timeline	open ()		read			read			
				read			read		
					read				
	read ()				read				
					write			read	
	read ()				read				
					write			read	

SERC
Software Engineering Research Centre

Writes to a File

- Make a call write() to write into the file on the disk
- Data block may have to be allocated (if not overwriting)
  - Need to update data bitmap and data block
  - Total of five I/O:
    - One to read data bitmap
    - Write to data bitmap
    - Two more to read and write the inode
    - Write to the actual block itself
- In case of creation of new file, number of I/Os can go really high!

Writing a File To Disk

Timeline	data bitmap	inode bitmap	root inode	lectures inode	101 inode	root data	lecture data	101 data [0]	101 data [1]
create () /lecture/101			read						
						read			
				read					
							read		
		read write							
							write		
					read write				
				write					
write ()					read				
	read write								
								write	
					write				

Can we do something about performance?

- Reading and writing files are expensive
- Imagine opening and reading a file by providing a long path
  - Each inode needs to be fetched, corresponding data then read of files
  - Can go upto 100s of I/Os
- Use the concept of caching and buffering
  - Use system memory to cache important blocks - **Minimise overheads!**
  - Early FS, used **fixed-size cache** -> store popular blocks (10% at boot time)
  - Use strategies like LRU to evict blocks

Caching and Buffering

- Static partitioning of memory is not always useful - **Wastages!**
- Modern systems employ dynamic partitioning approach
  - Integrate virtual memory pages and FS pages into unified page cache
  - First open may generate lot of I/O but subsequent will be in cache!
- Writes is little tricky as at some point the disk has to be accessed to store
  - **Write buffering** - Delay writes to disk, perform batch I/O
  - Schedule I/Os in a particular order for performance gain
  - Writes can be avoided totally - file is created and deleted in few seconds! **(Don't write)**

Caching and Buffering

- Applications like DB avoids caching altogether - direct I/O
  - System calls like fsync() allows writes to be pushed immidiately
  - Unexpected data loss may happen since data is in memory
  - Has impact on overall system performance
- At the end its all about trade-off's
  - Durability vs Performance tradeoff
  - Has big dependance on the application
    - Browser vs Transactional database!

SCAN ME

Thank you

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