CS3.301 Operating Systems and Networks

Virtualization - Process

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

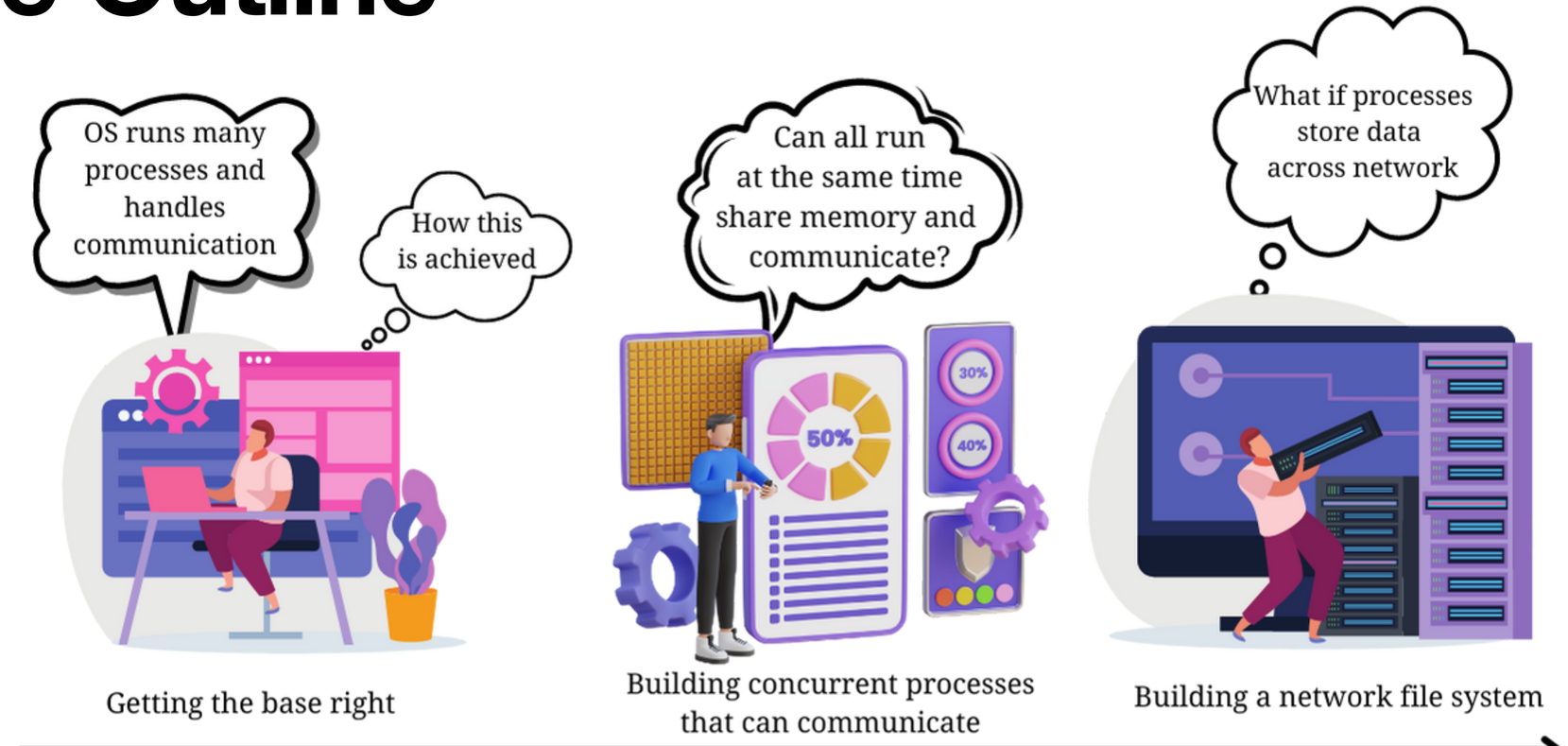
Acknowledgement

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

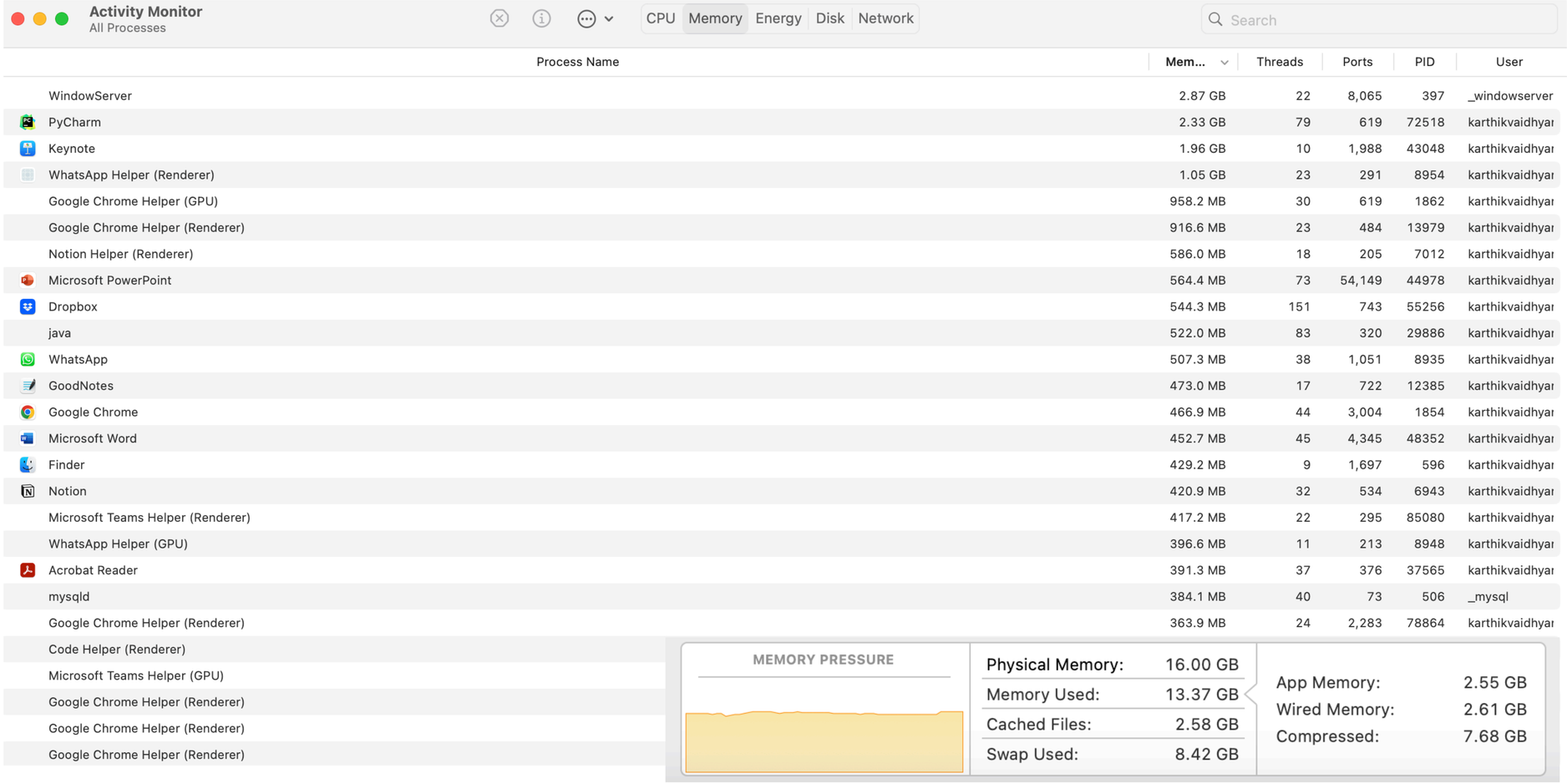


This Course

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

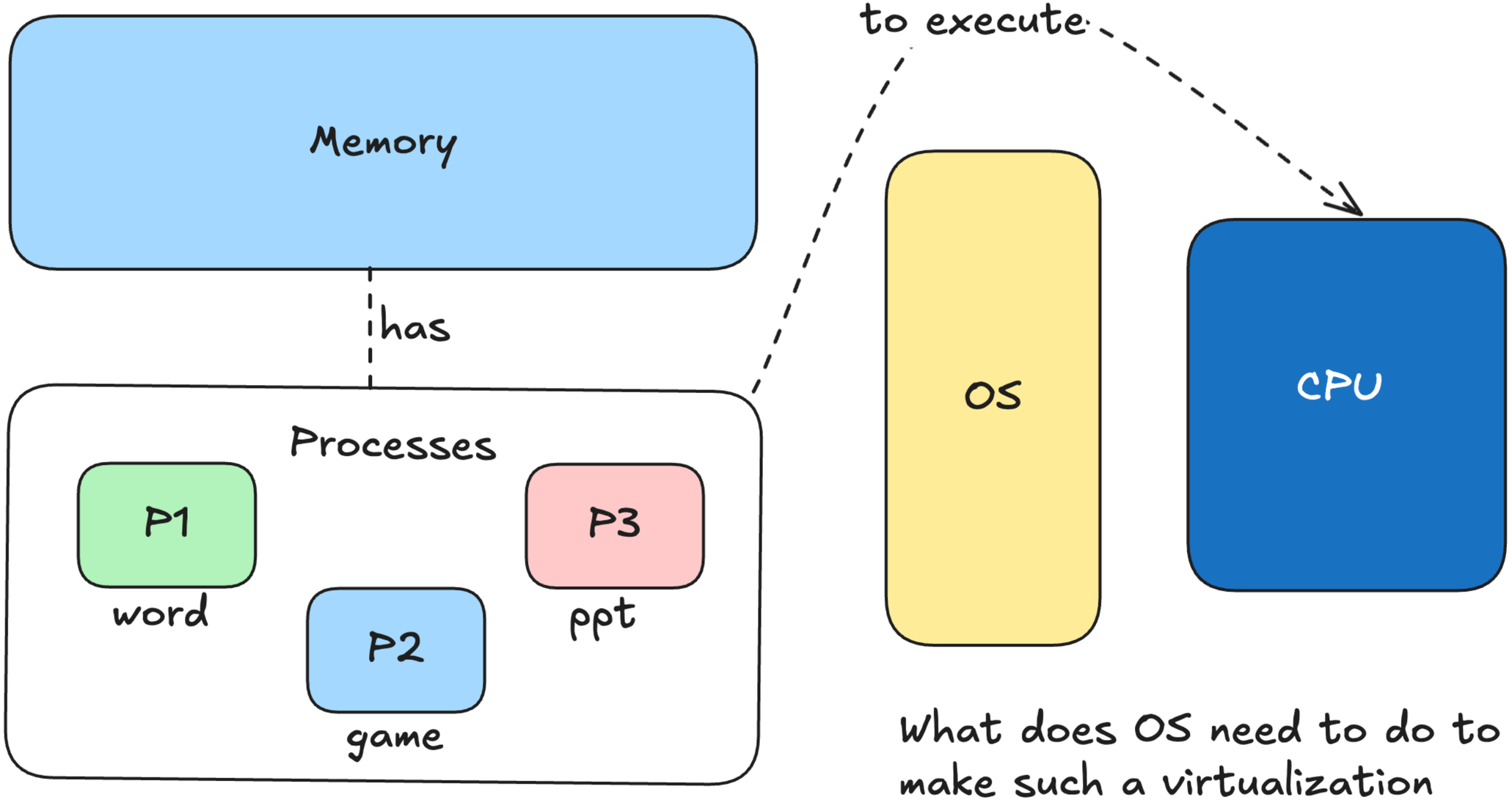
- Each process feels that it has its own CPU
- Even in Single core machines - There can be multiple process that run at the same time
- How is CPU handling this?

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?

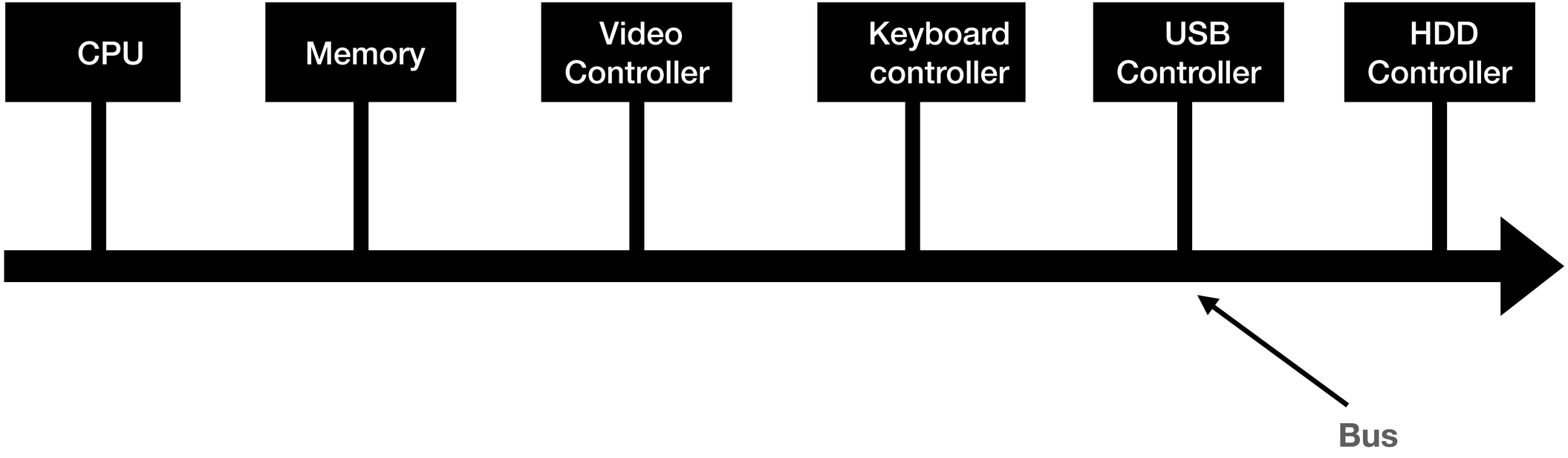
Lets think!



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

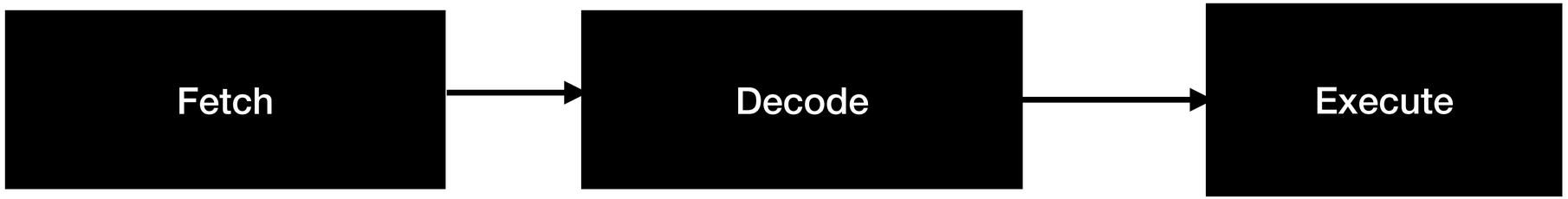
Some Prerequisite - Computer Hardware

- 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

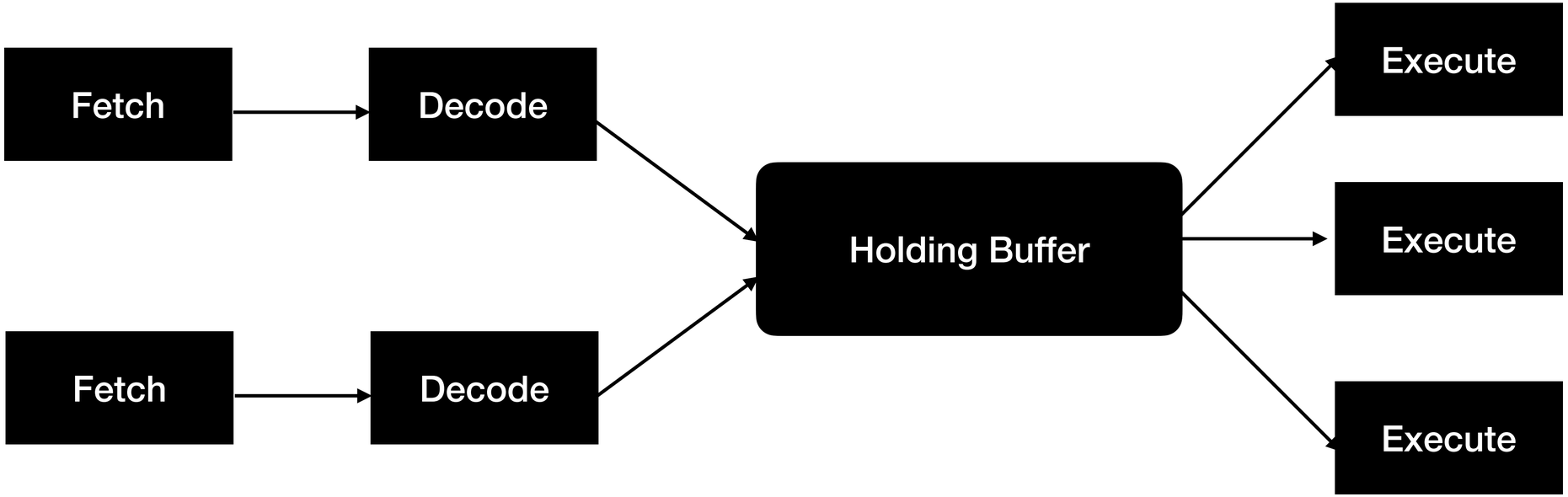
How does CPU execute a program?

- Three stage pipeline



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

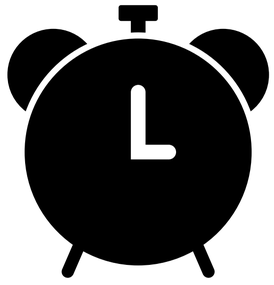
Some Prerequisite - Computer Hardware



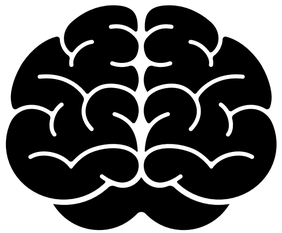
Superscalar CPU

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



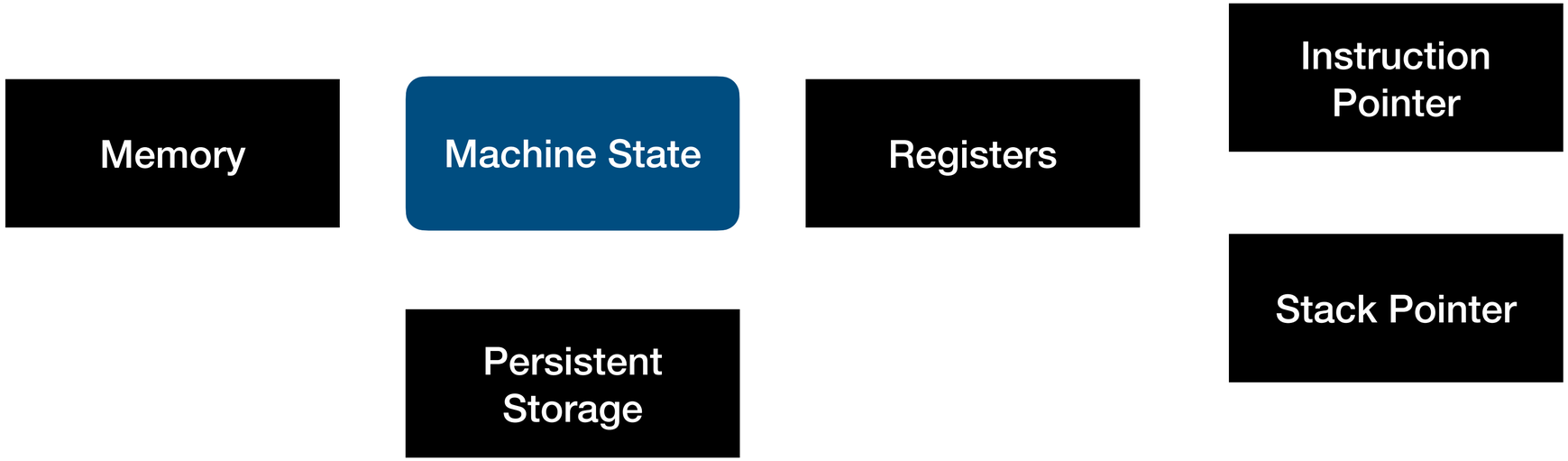
- Basically we need - **low level mechanisms** and **policies** (CPU Scheduler)

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

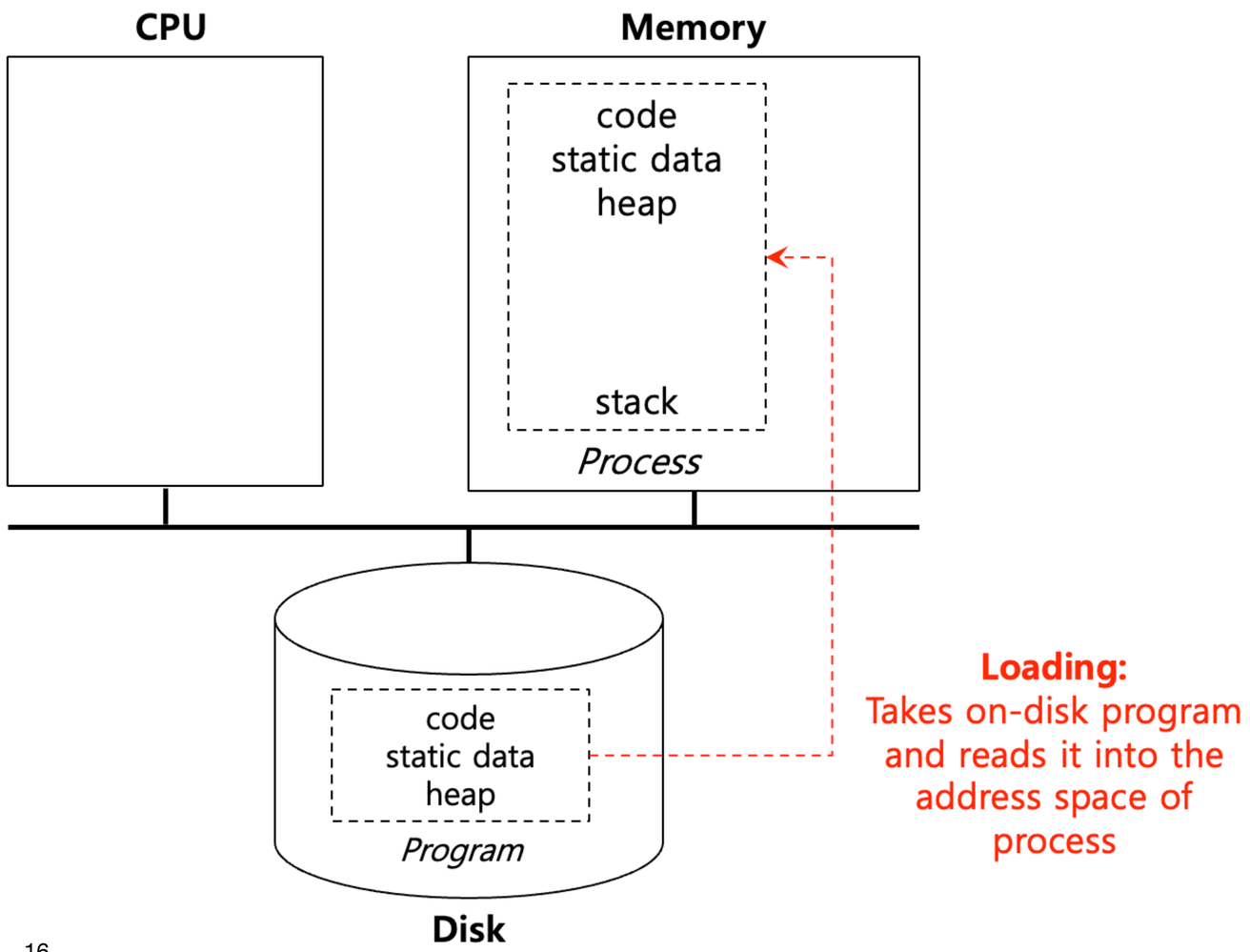


Image from OS lecture by Youjip Won

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**

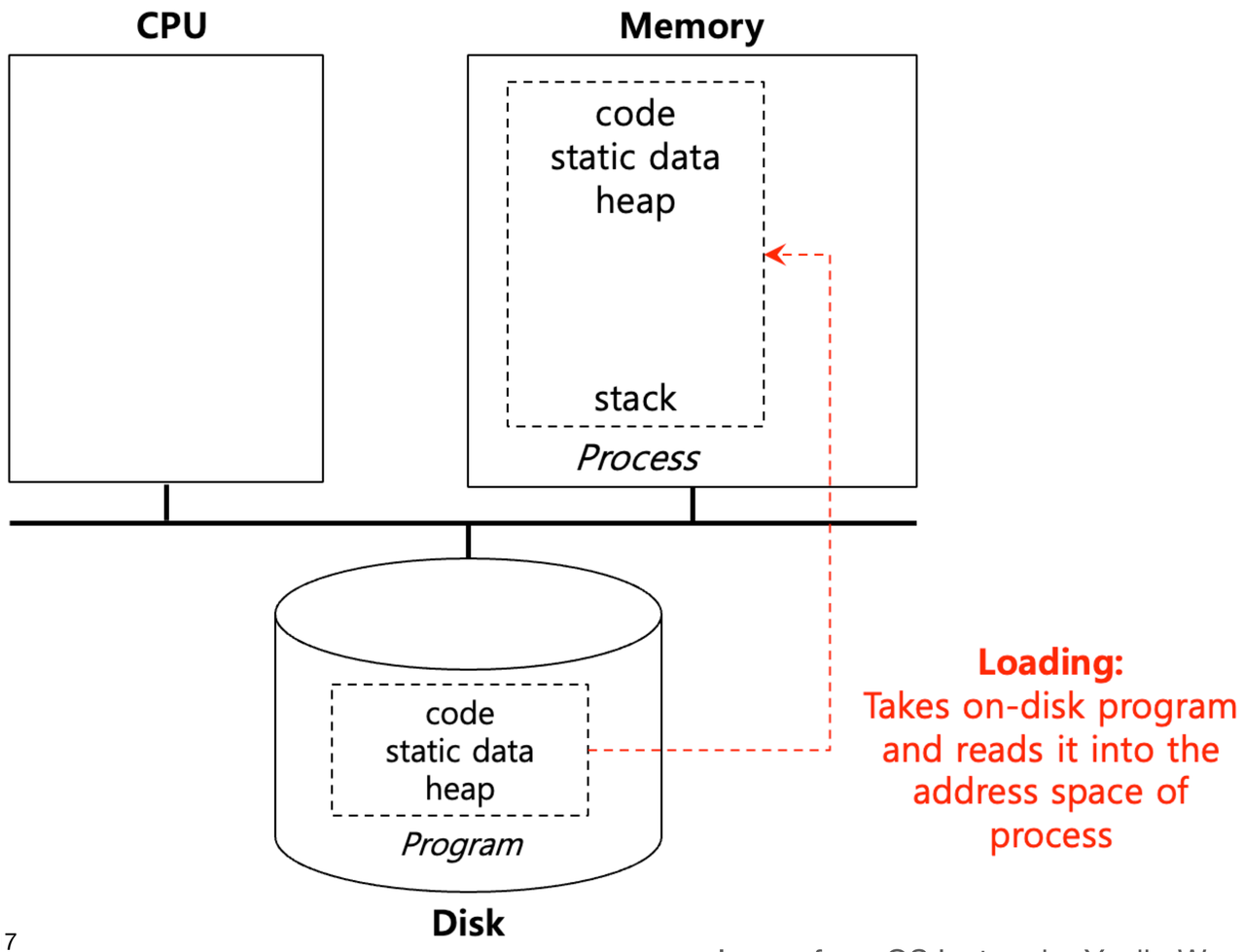


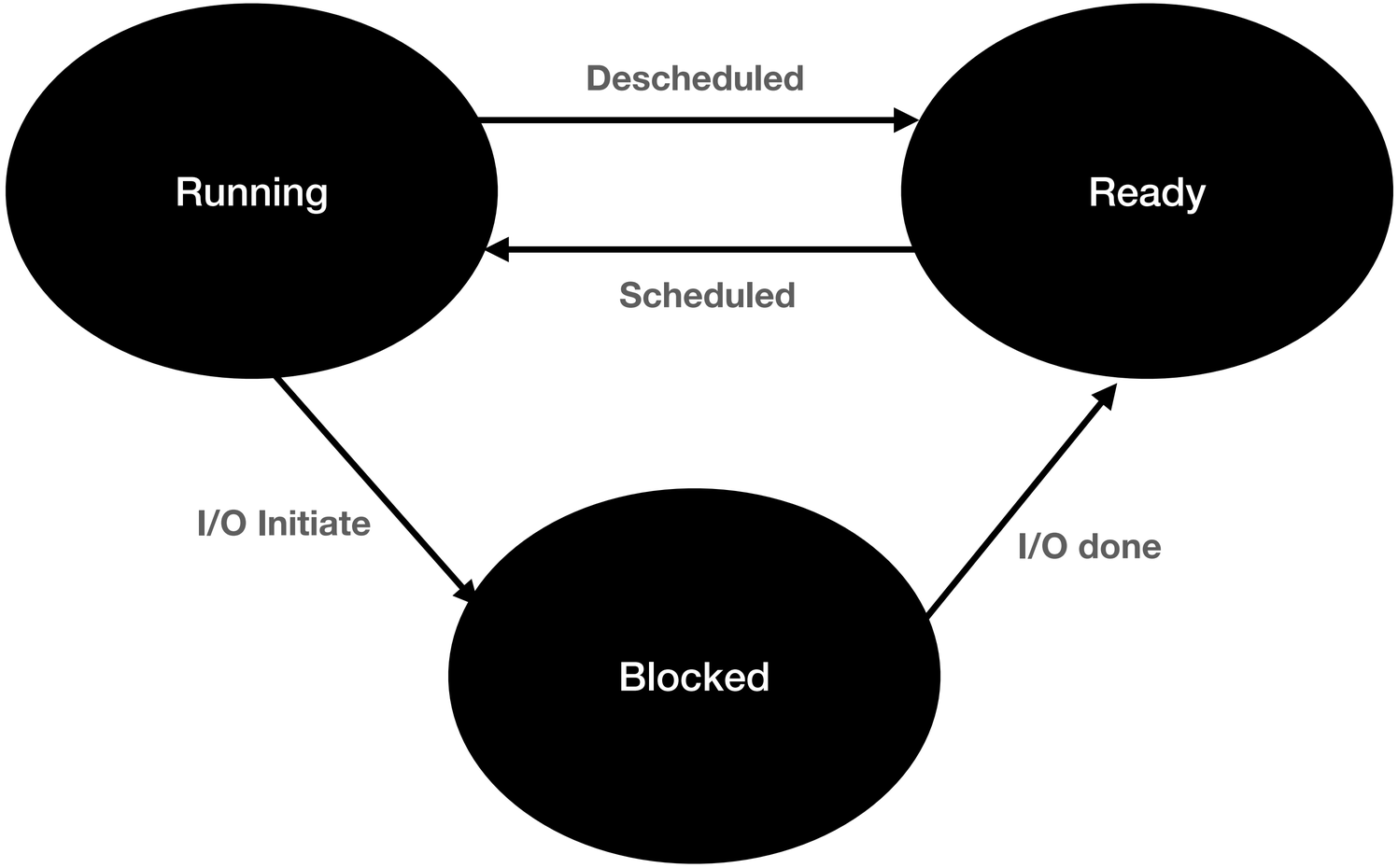
Image from OS lecture by Youjip Won

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

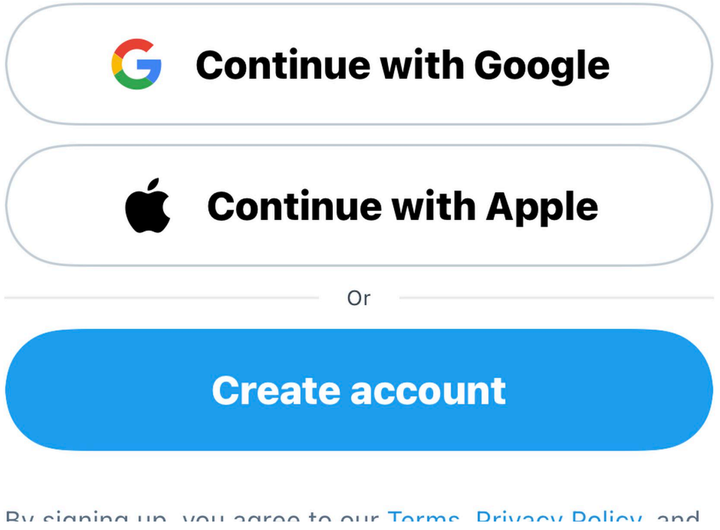
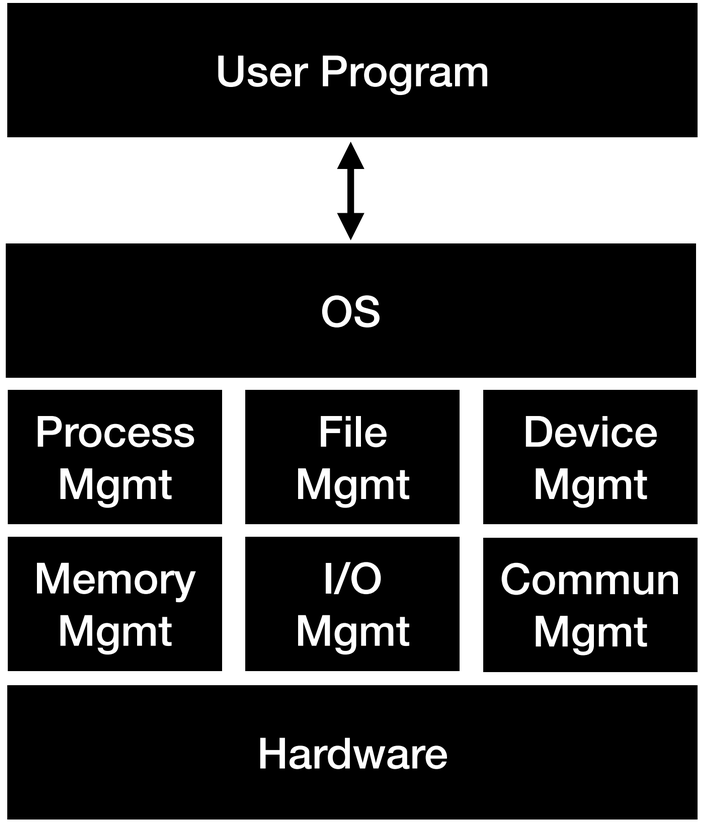


Image source: verge

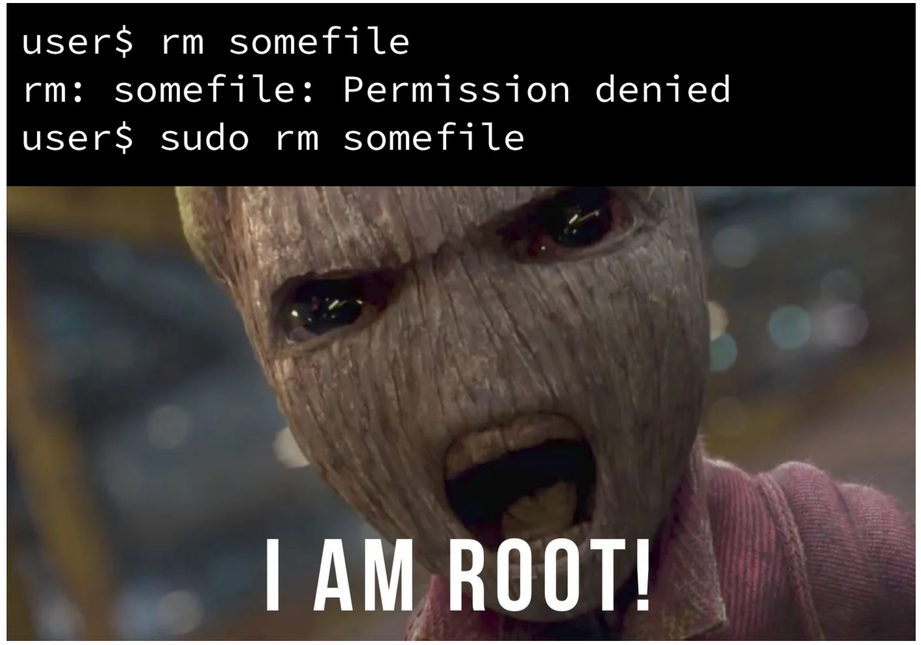
Does OS Provide API? - System Calls!

- 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!

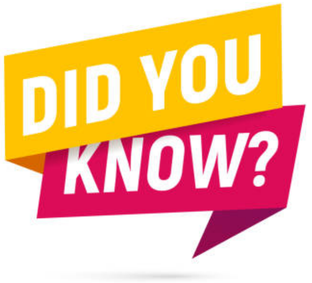
- What if a user gives a instruction to delete all files?
  - Should all the instructions be considered with equal priority?
  - When does the role of OS come in to the main picture?
    - Think about reading a file or writing a file - How to achieve it in C?
    - What if you just wanted to multiply 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!**

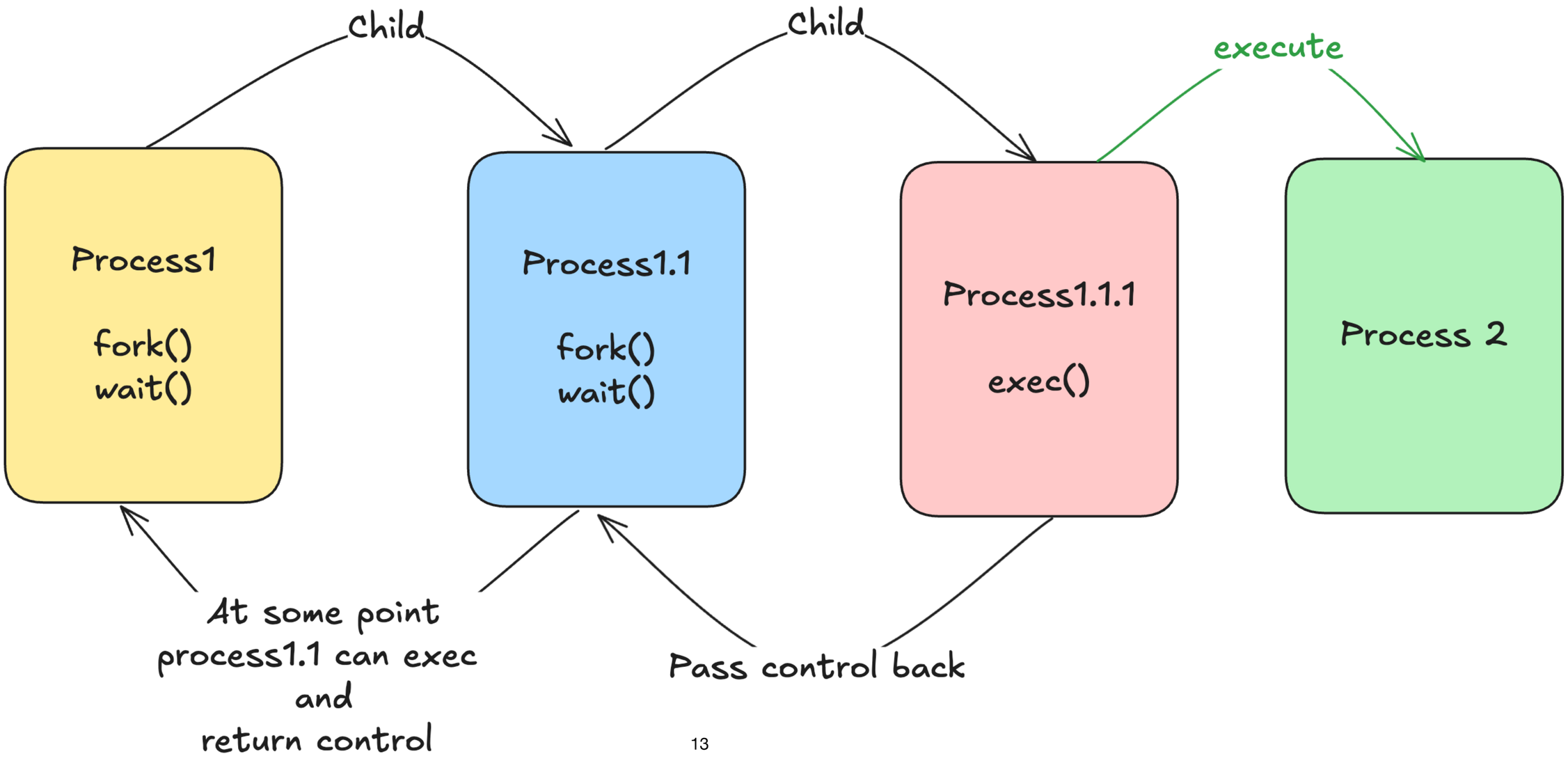
Remember: Init process, adopts orphans and reaps them

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?

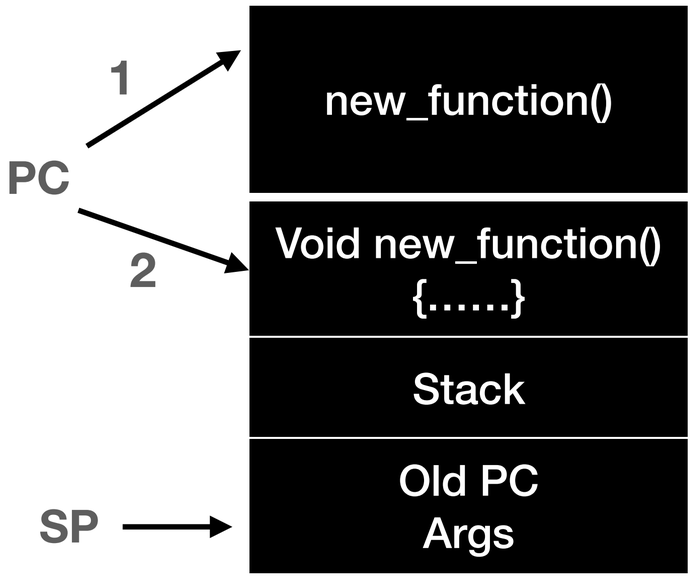
What if we allow process to do whatever it wants?

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?

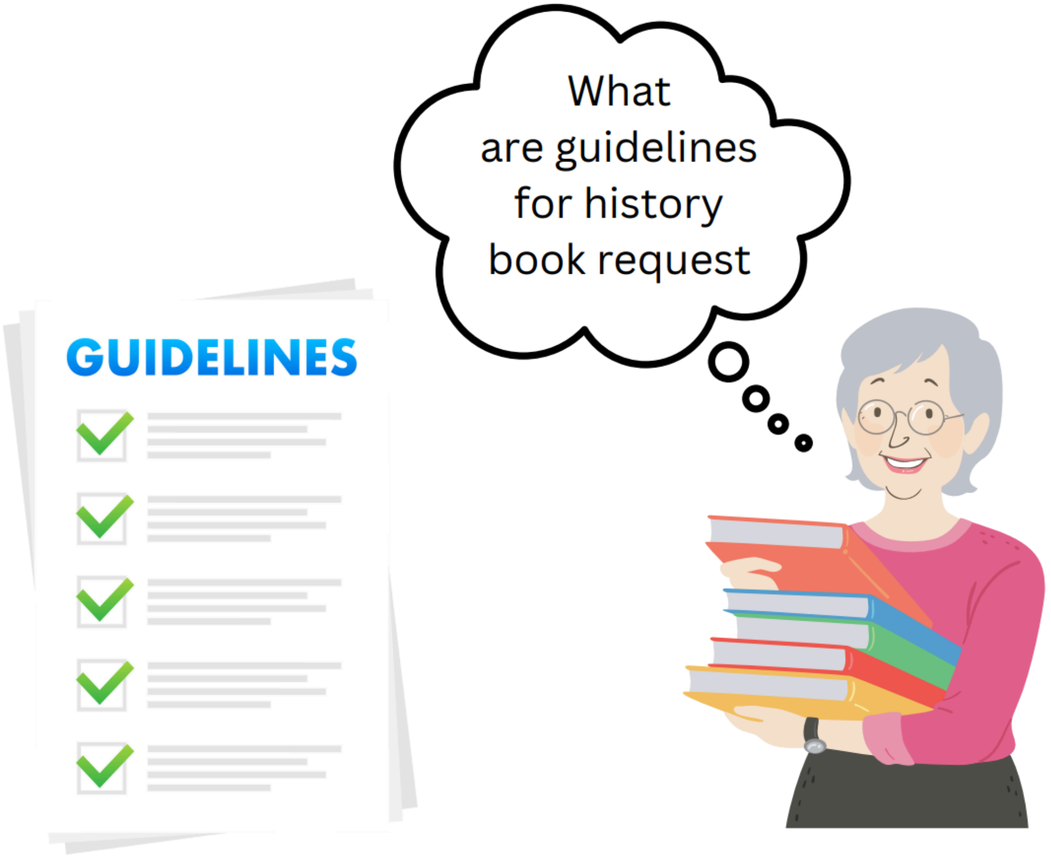
Lets draw some Parallels

Visitor/User

Librarian

Reference Books Section

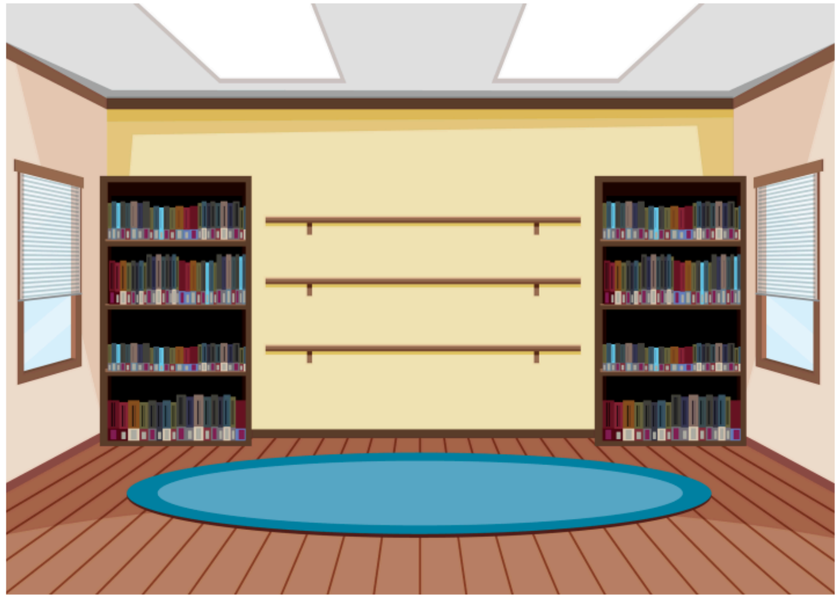
Lets draw some Parallels



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

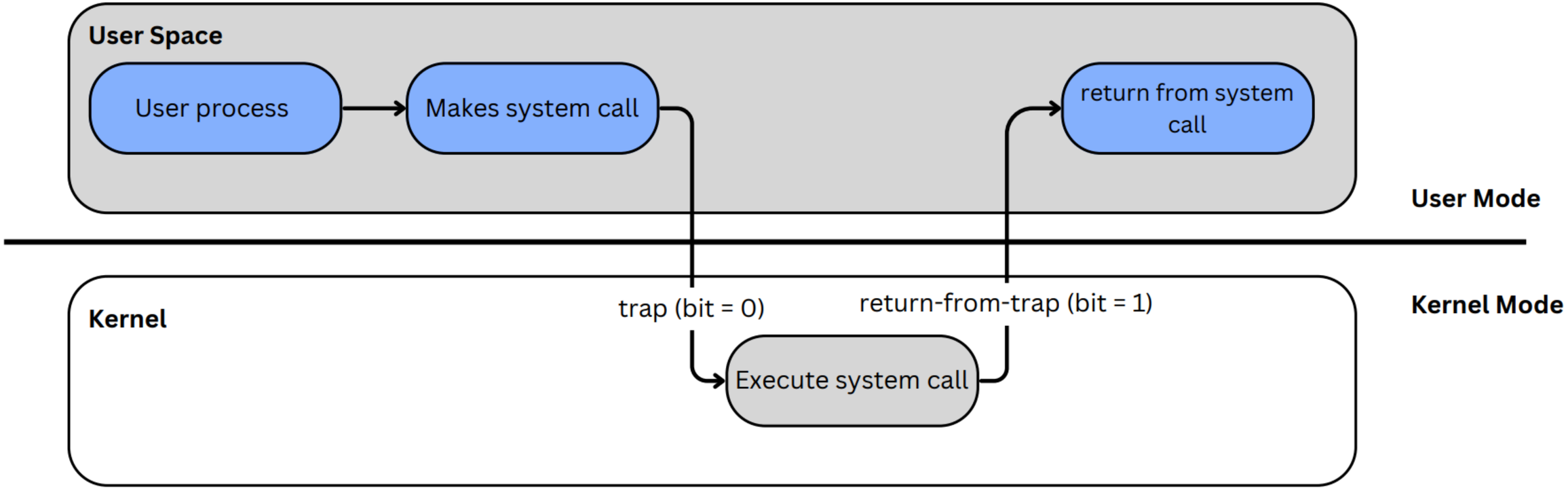
- Special kind of instruction to switch mode from user to kernel
- Allows system to perform what it wants
- When a system call is made, the trap instruction allows to jump into kernel
  - Raise the privilege mode to kernel mode
  - **Return-from-trap** instruction allows switch back to user mode
  -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

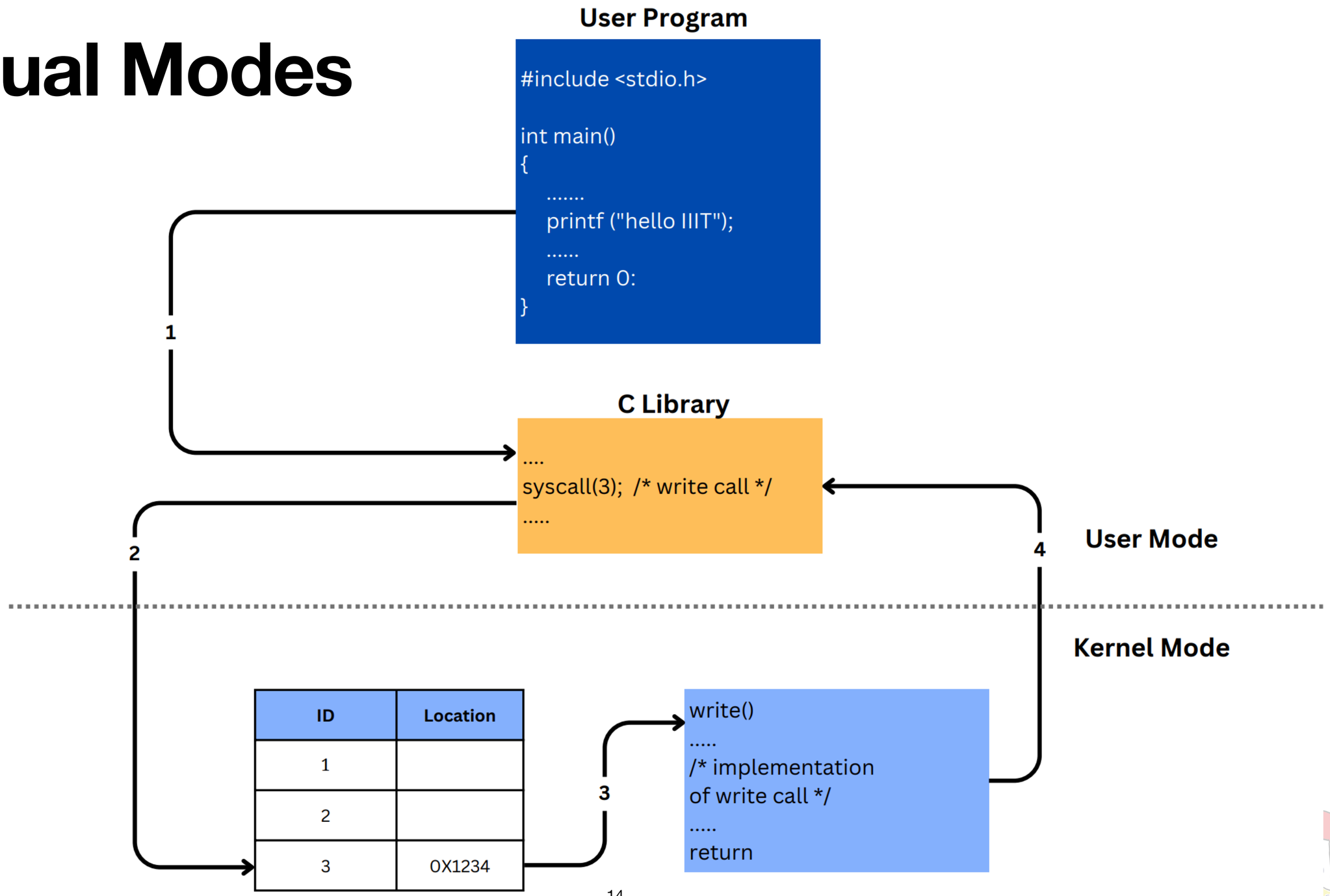
The dual modes

User Mode and Kernel Mode



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

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

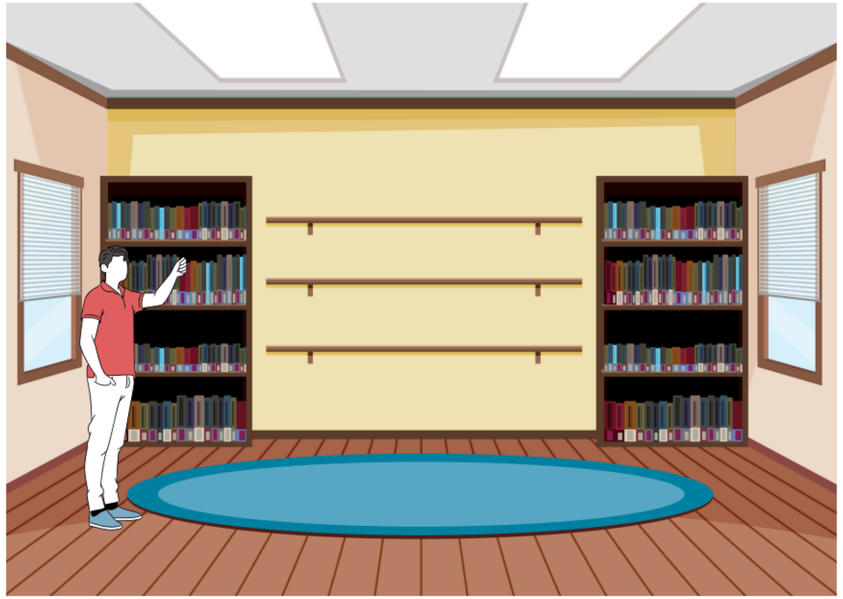
OS @ boot (Kernel mode)	Hardware	
Initialize trap table	Remember address of.. Syscall handler..	
OS @ run (Kernel mode)	**Hardware**	**Program (User mode)**
Create entry for process list Allocate memory for program Load program into memory Setup user stack with arg Fill kernel stack with reg/PC		
	Restore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

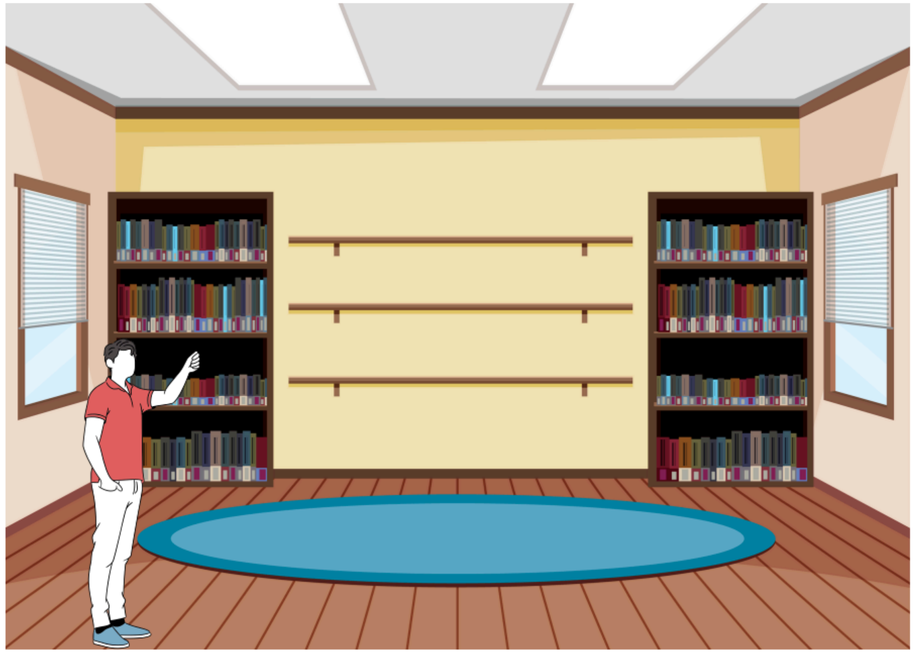
What if?

- 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?

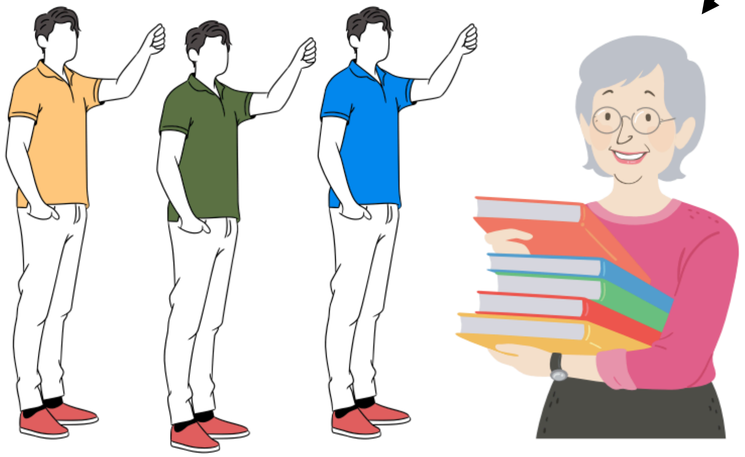
Need for Policies (Scheduling)

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?

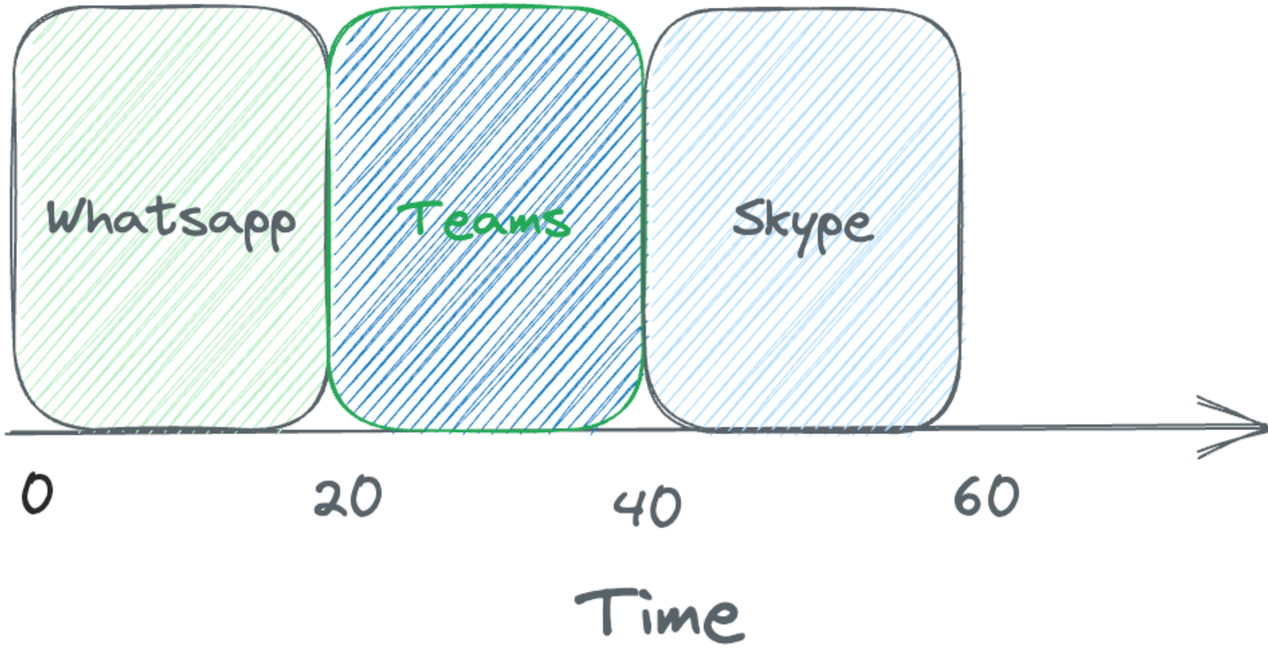
First Come First Serve Policy

- 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

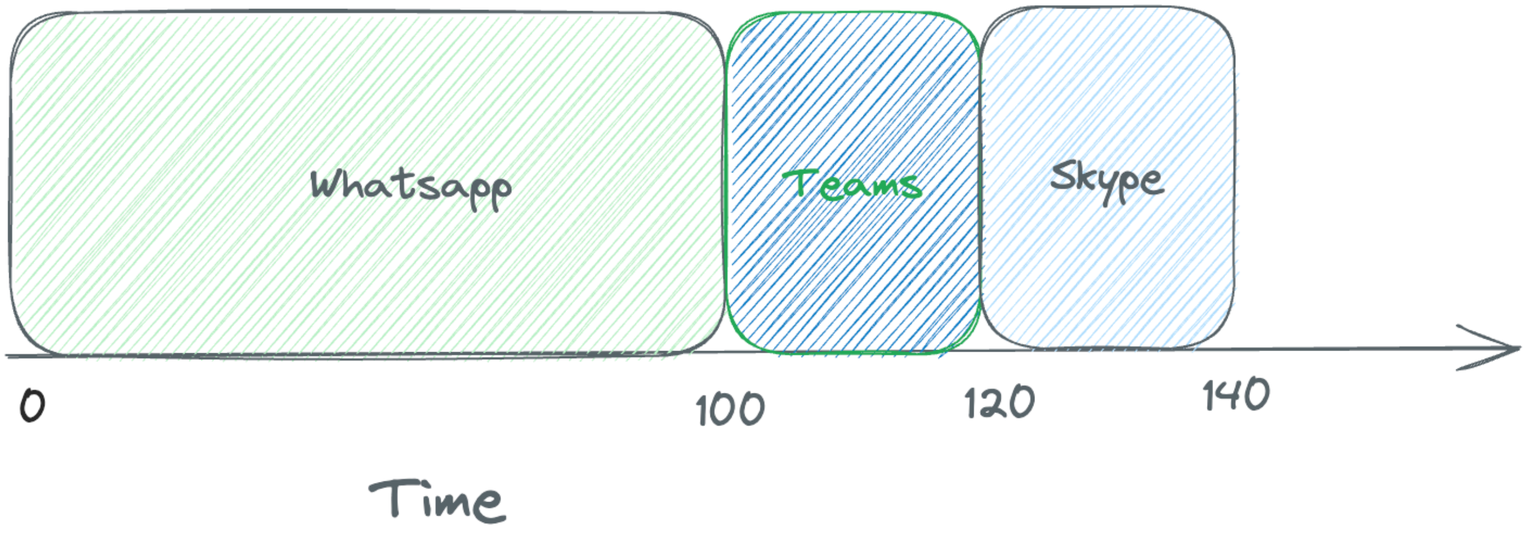


$$Avg(T_{turnaround}) = \frac{20 + 40 + 60}{3}$$

$$= 40$$

What if each job no longer runs for same time?

Relaxing assumption 2

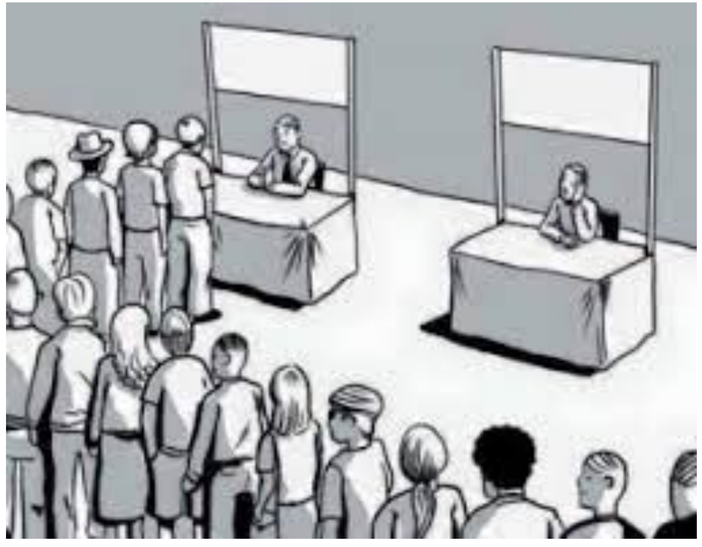


$$Avg(T_{turnaround}) = \frac{100 + 120 + 140}{3}$$

$$= 120$$

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

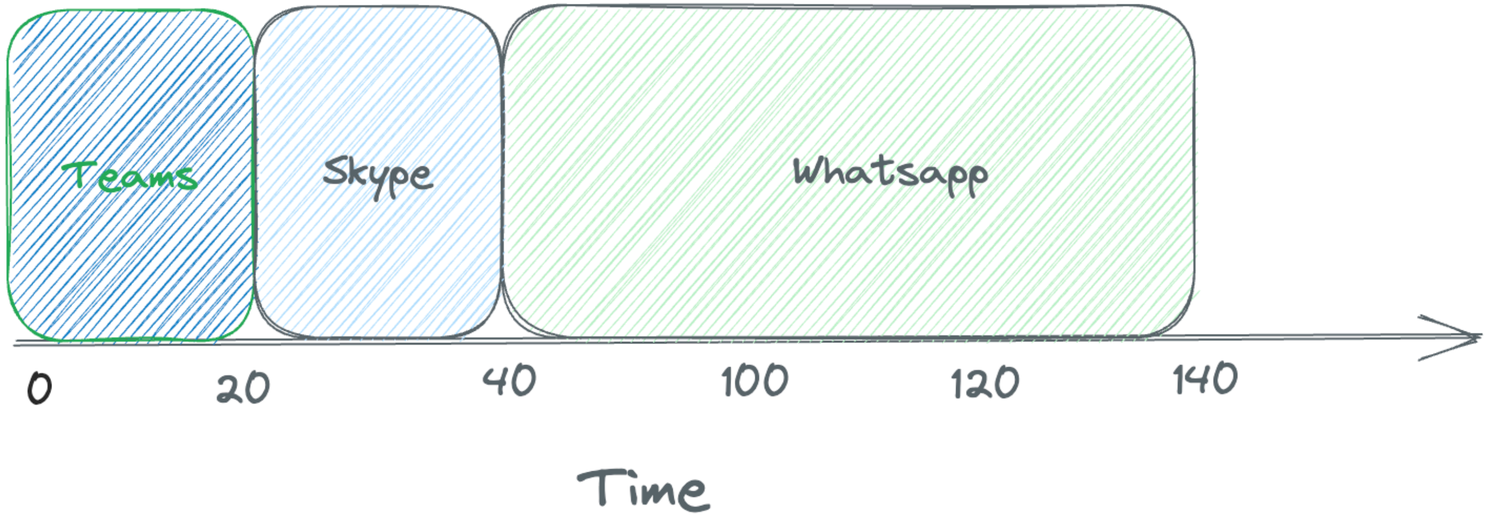
How to go about this?

Shortest Job First (SJF) Policy

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

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

$$= 66.3$$



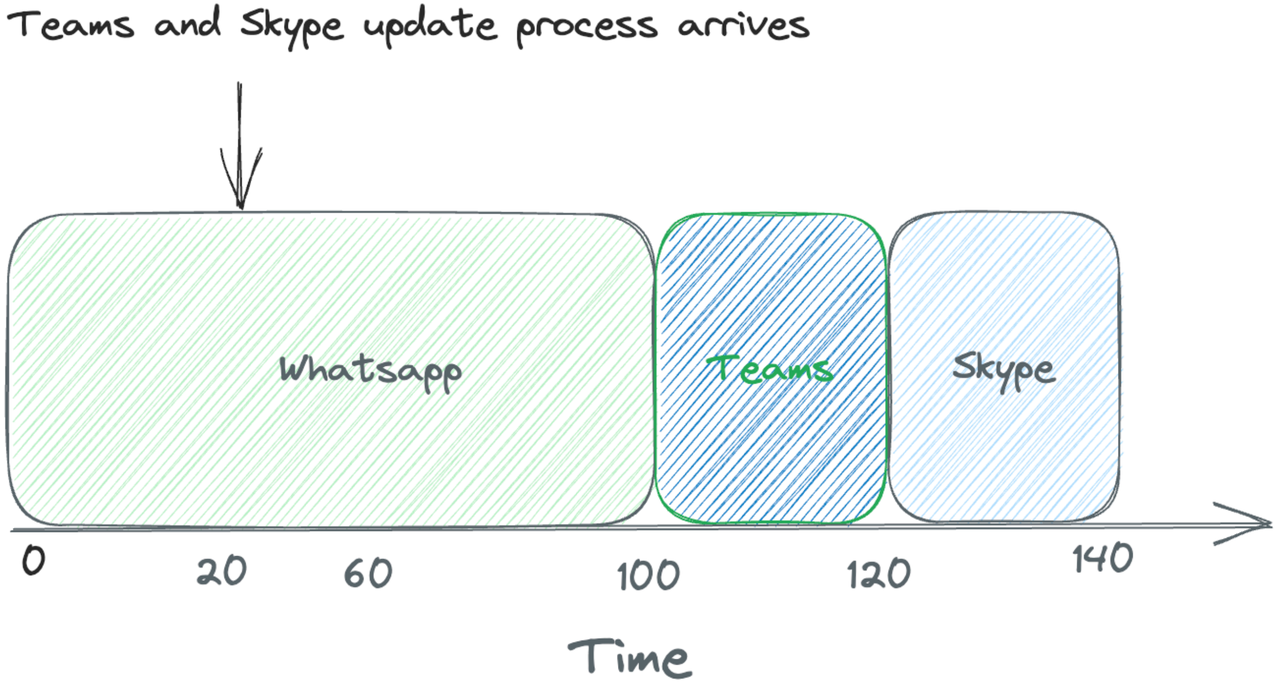
Is that a bit too unrealistic? - In reality jobs can arrive at any time

Shortest Job First (SJF) Policy

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

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

$$= 106.6$$

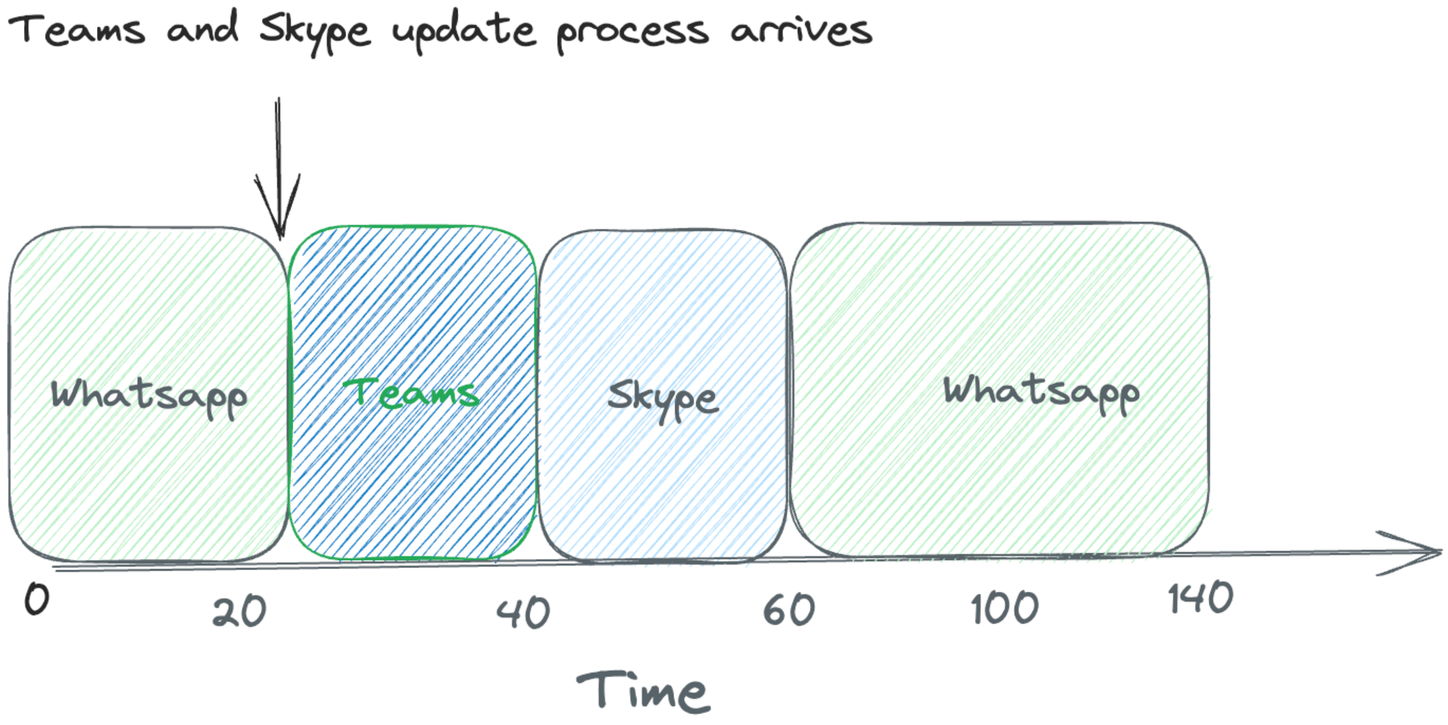


Even worst!! How to improve?

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?

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

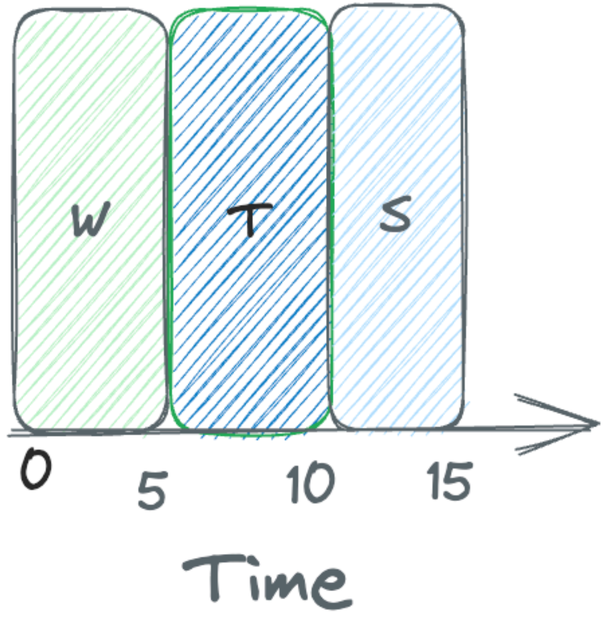
Round Robin Scheduling

- 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



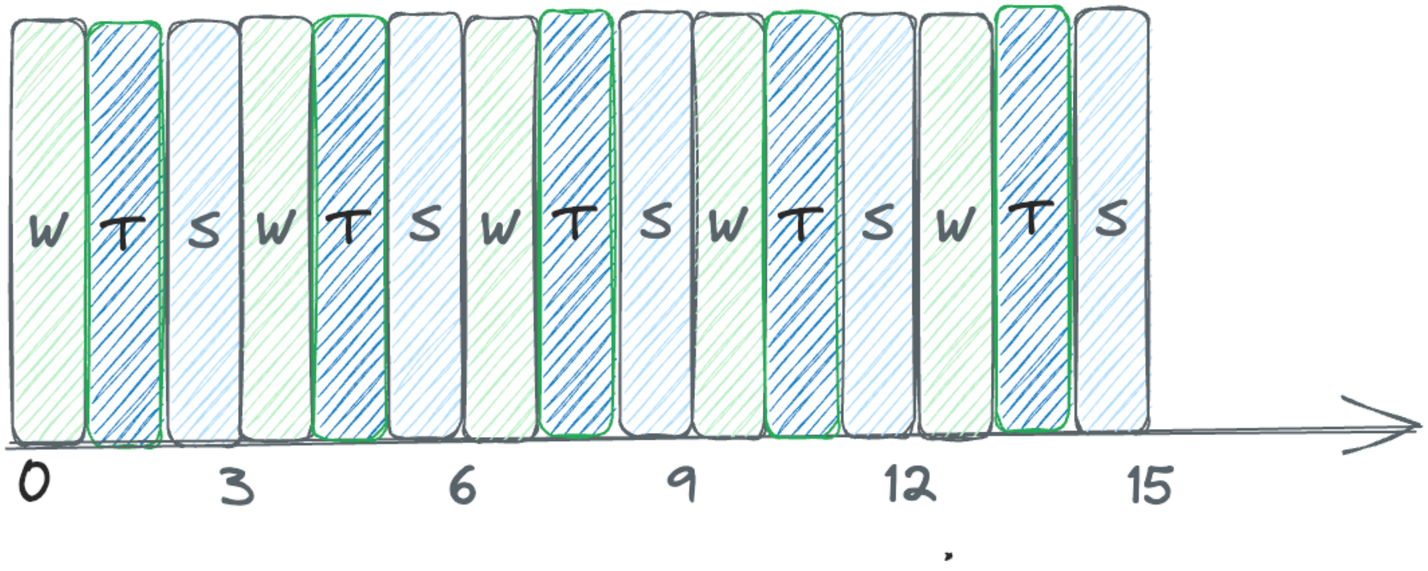
$$Avg(T_{turnaround}) = \frac{5 + 10 + 15}{3} = 10$$

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

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

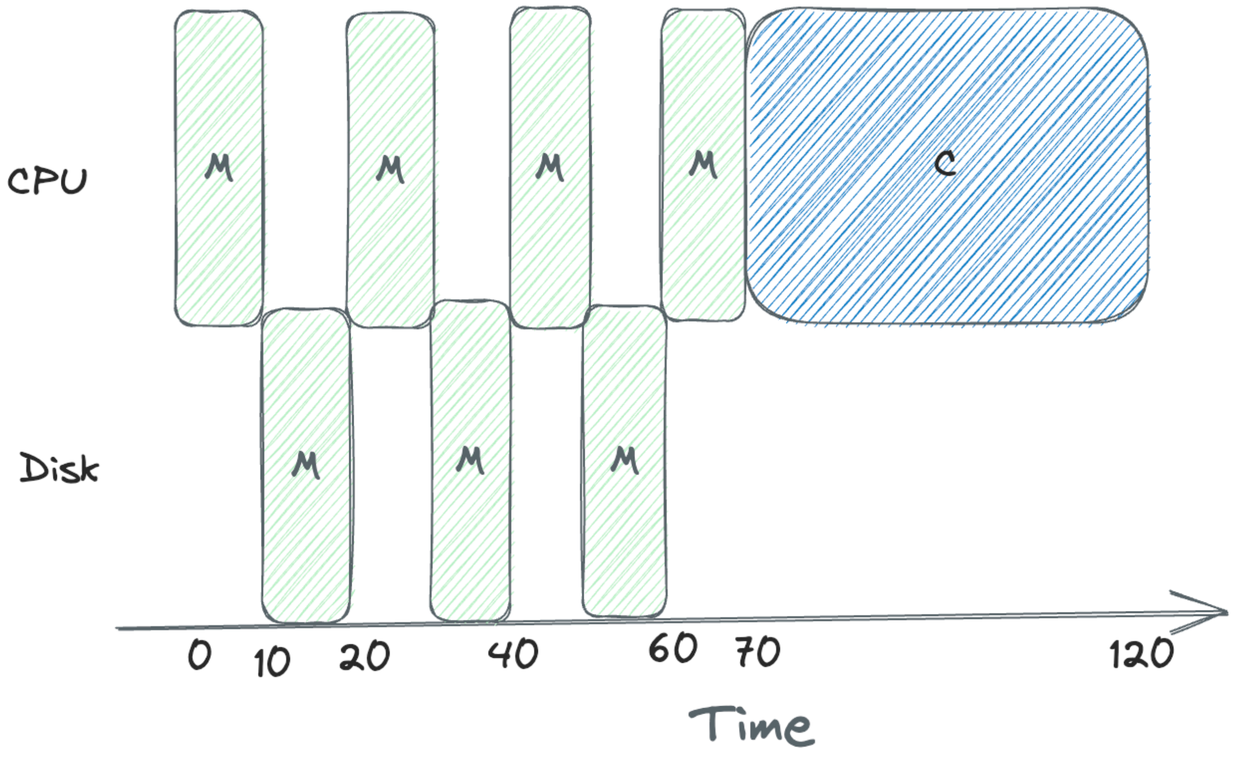
What can be done to consider I/O? Can we do RR Scheduling by considering I/O?

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

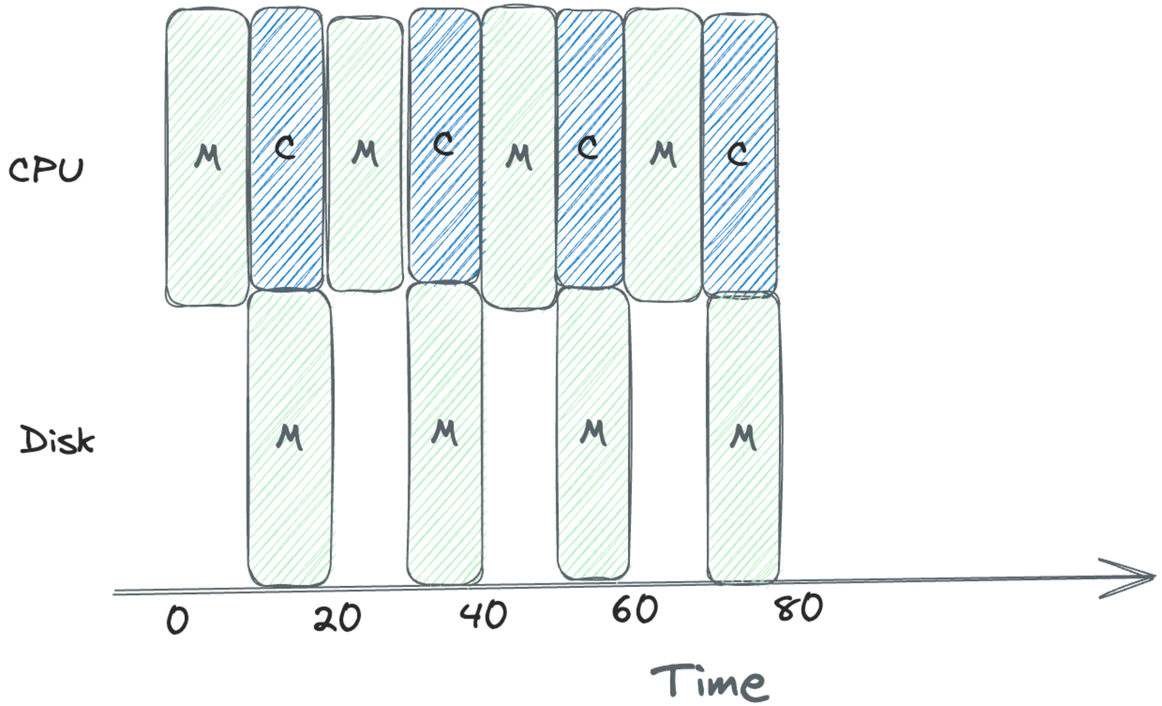


Can we do better?

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

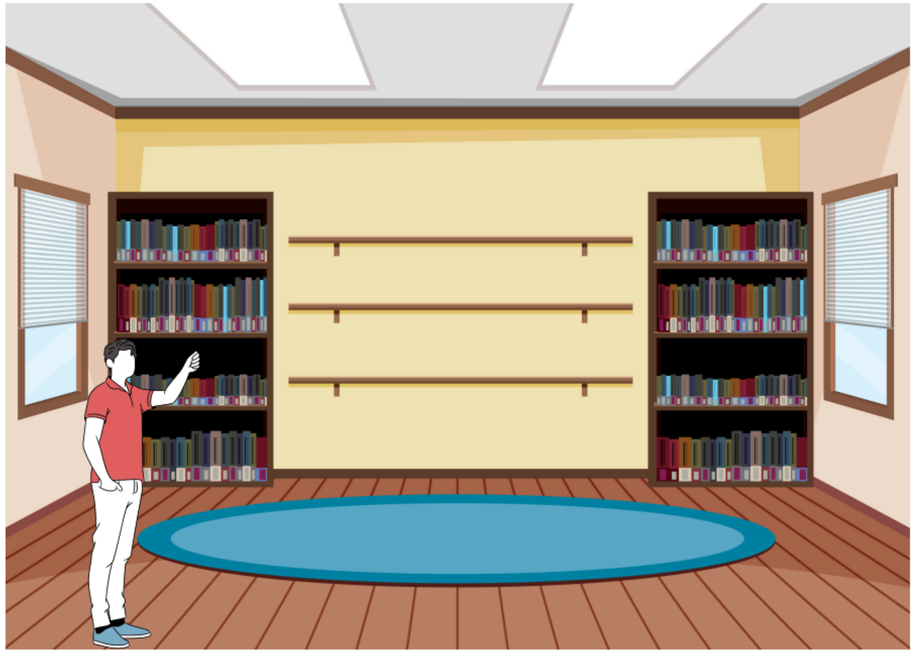


Do we miss something?

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

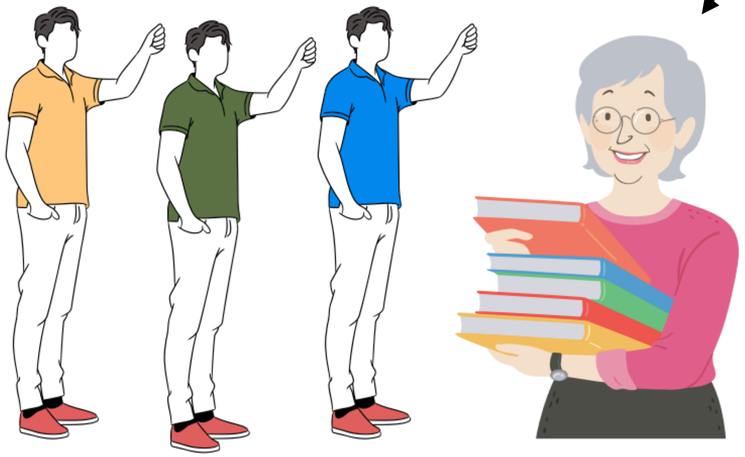
Lets go back to the example

How can Librarian take a guess?



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

Scheduler



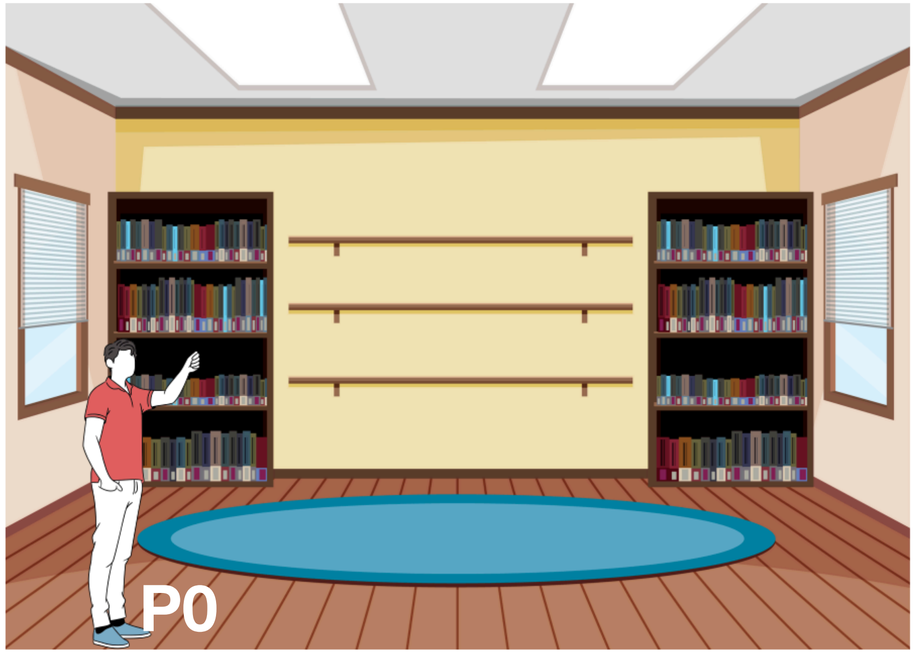
More users/visitors have requested to access the reference section.How to decide whom to sent next?

Why don’t we Prioritise?

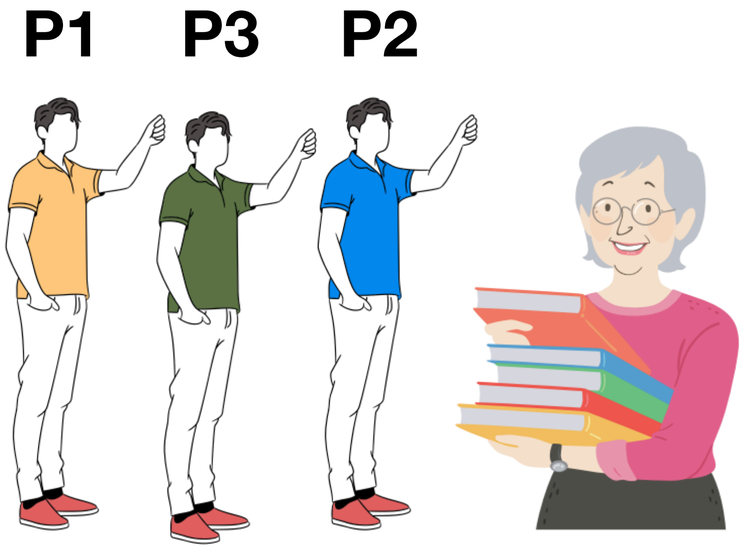
Source: https://imgur.com

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

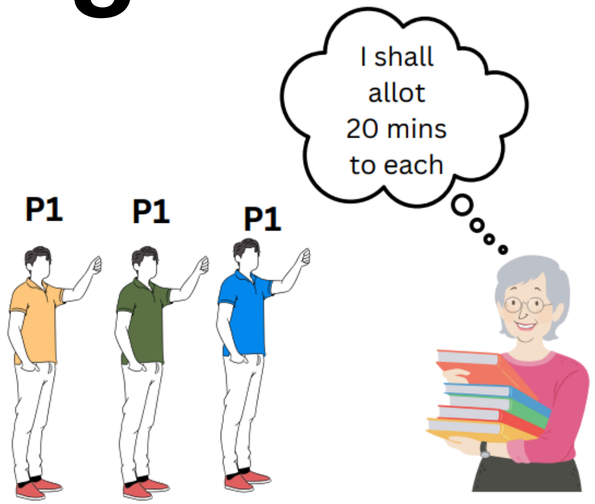
Can the policy learn continously to optimise response time and turnaround time?

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!

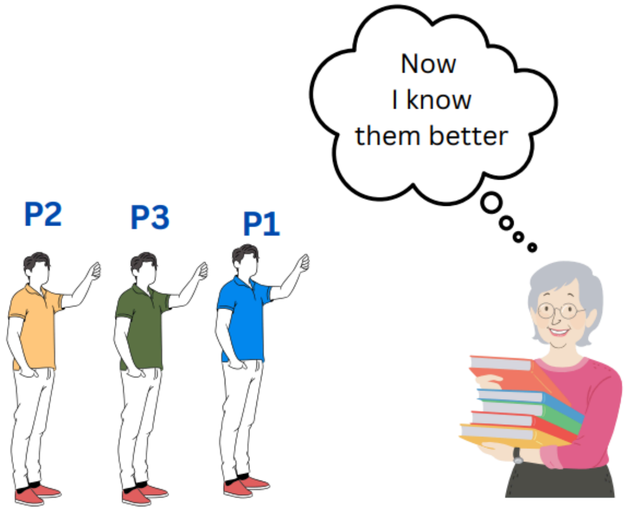
First person gets the chance!

out in 18 mins

Next person goes in

20 mins!!

0ut in 15 mins!



For the next visit, priorities have been updated

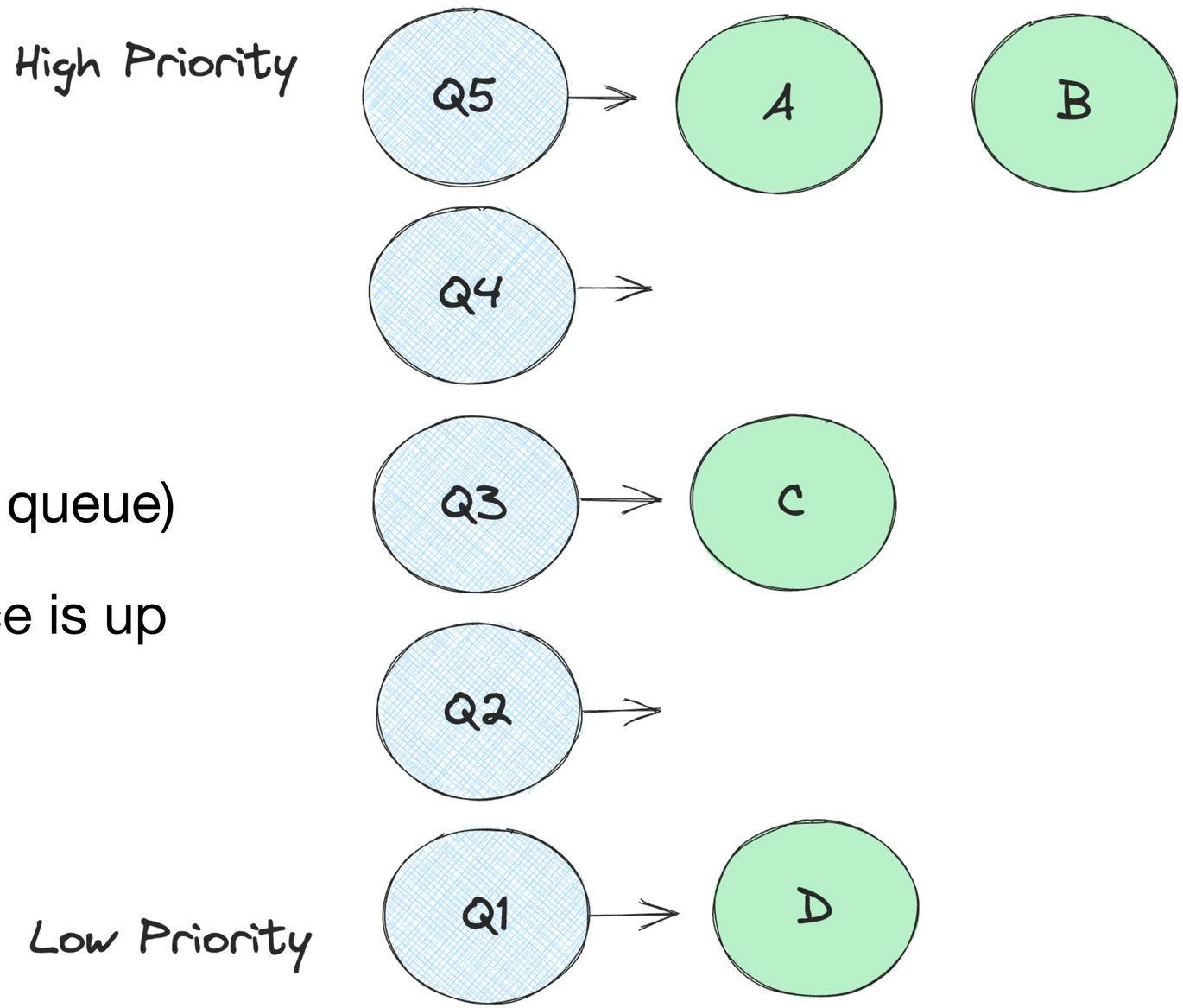
MLFQ

- 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

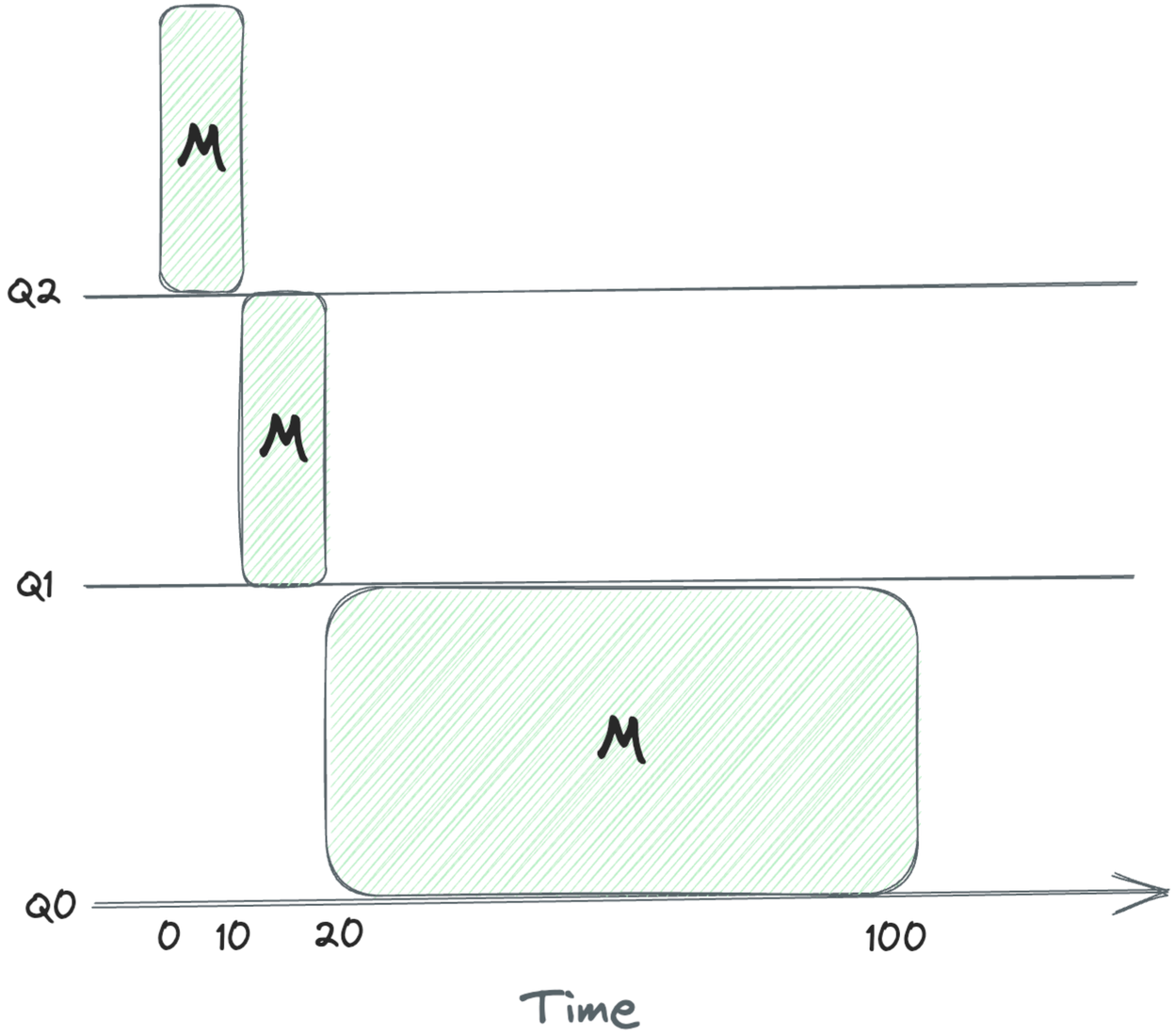
Five queues and 4 jobs

- 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

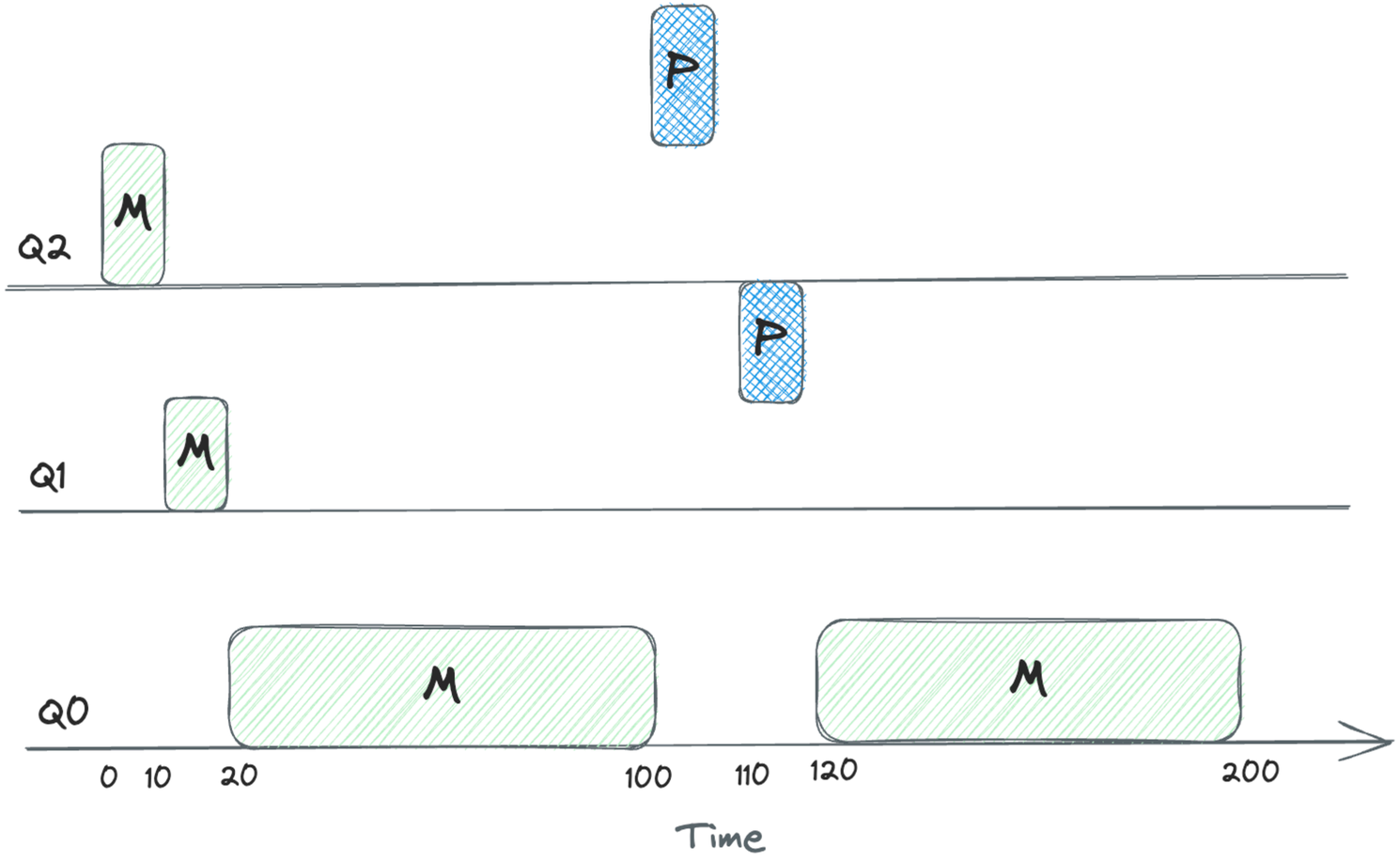


Example: Short job enters

- **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

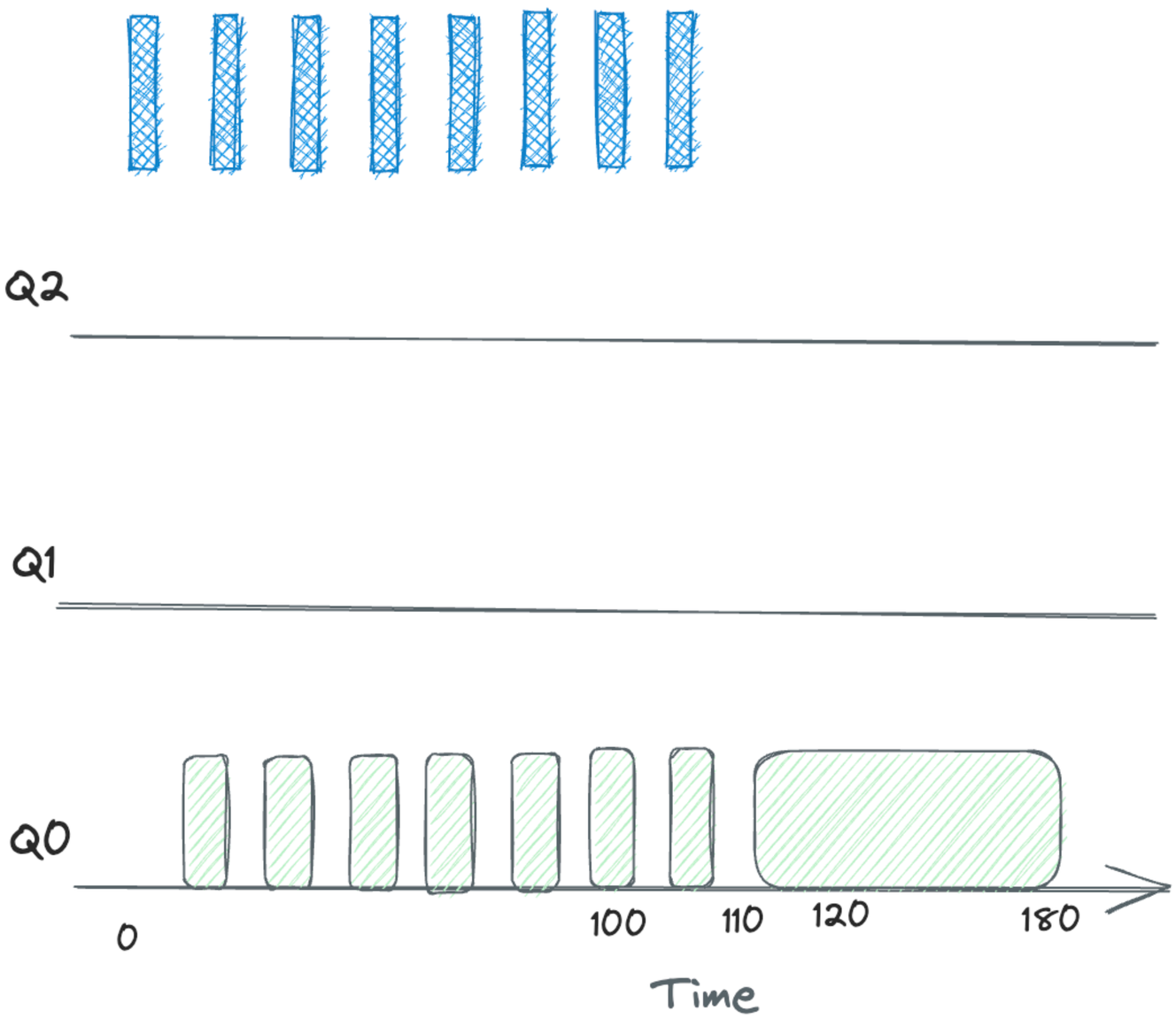
- If its short, it will keep running
  - Else moves down



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

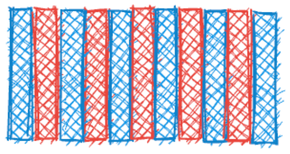
Can you think of some way to handle this?

Priority Boost

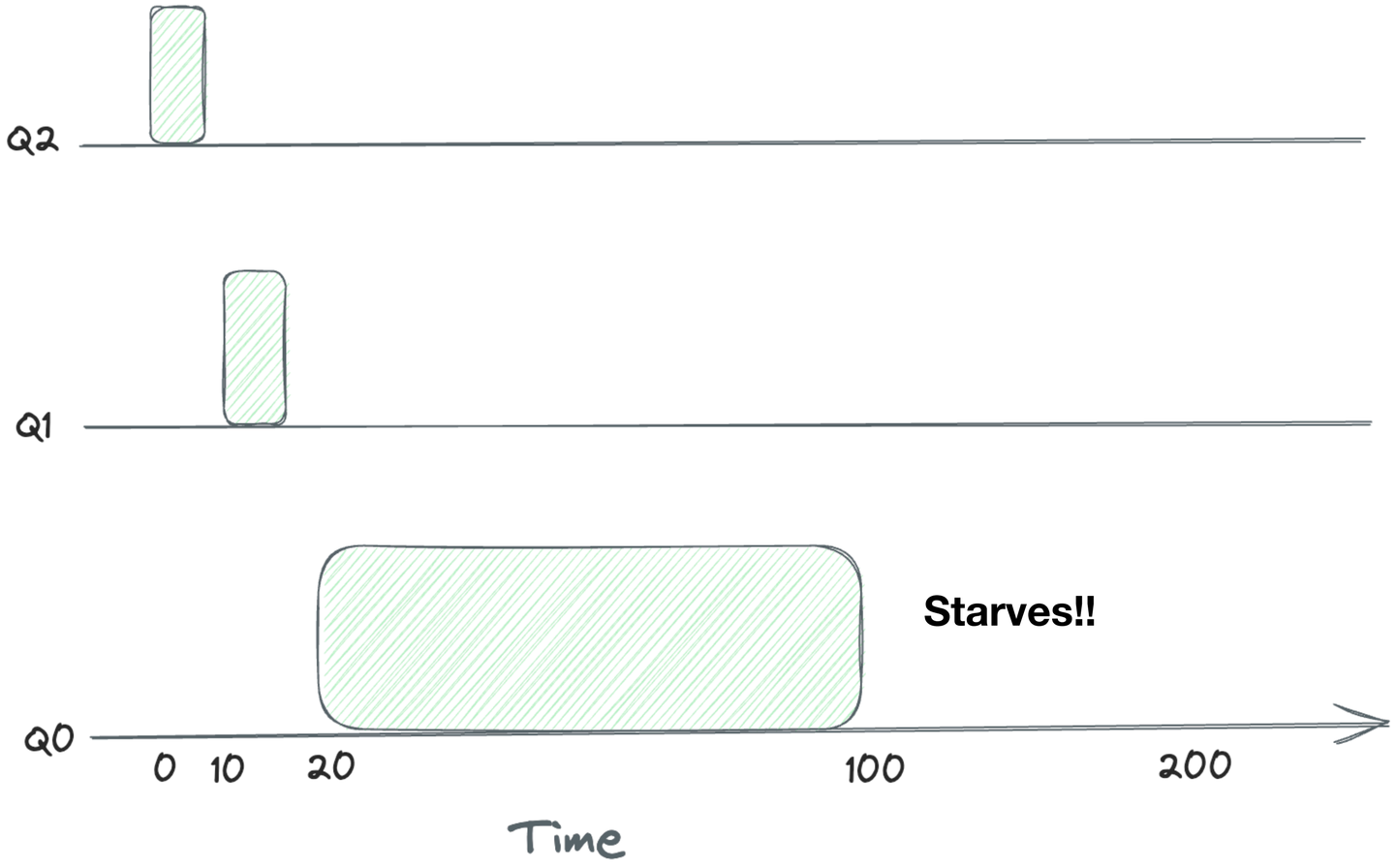
- Periodically boost the priority of all the jobs
  - This can **prevent starvation**
- **Rule:** After a time interval ***S***, move all jobs to top most queue
- **Provides two key guarantees**
  - Processes are guaranteed not to starve
  - If CPU bound has become interactive, scheduler will give it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

Case 1

Non priority Boost Scenario



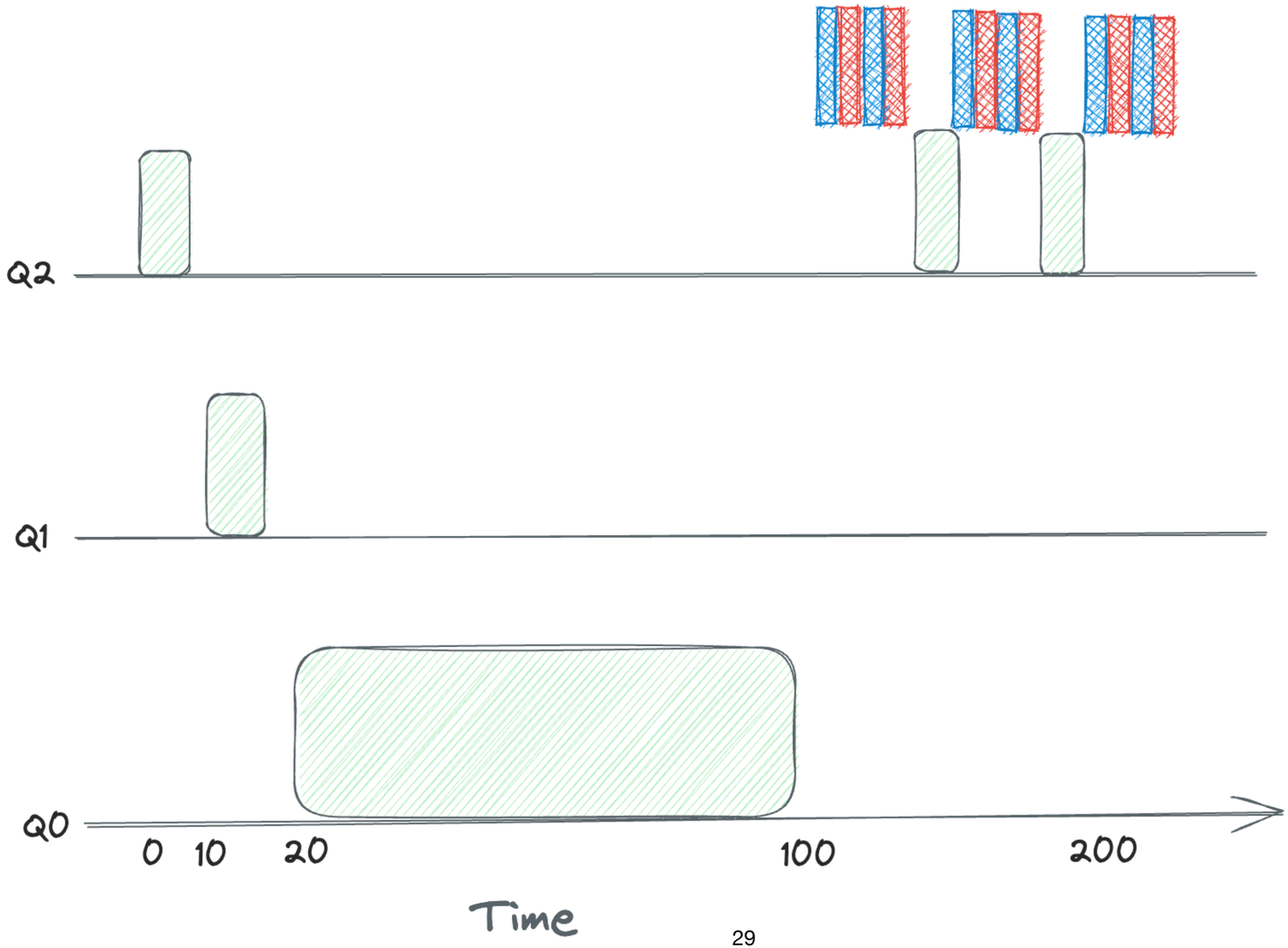
Interactive jobs keep getting the CPU



Starves!!

Case 2

With Priority Boost (S = 50)



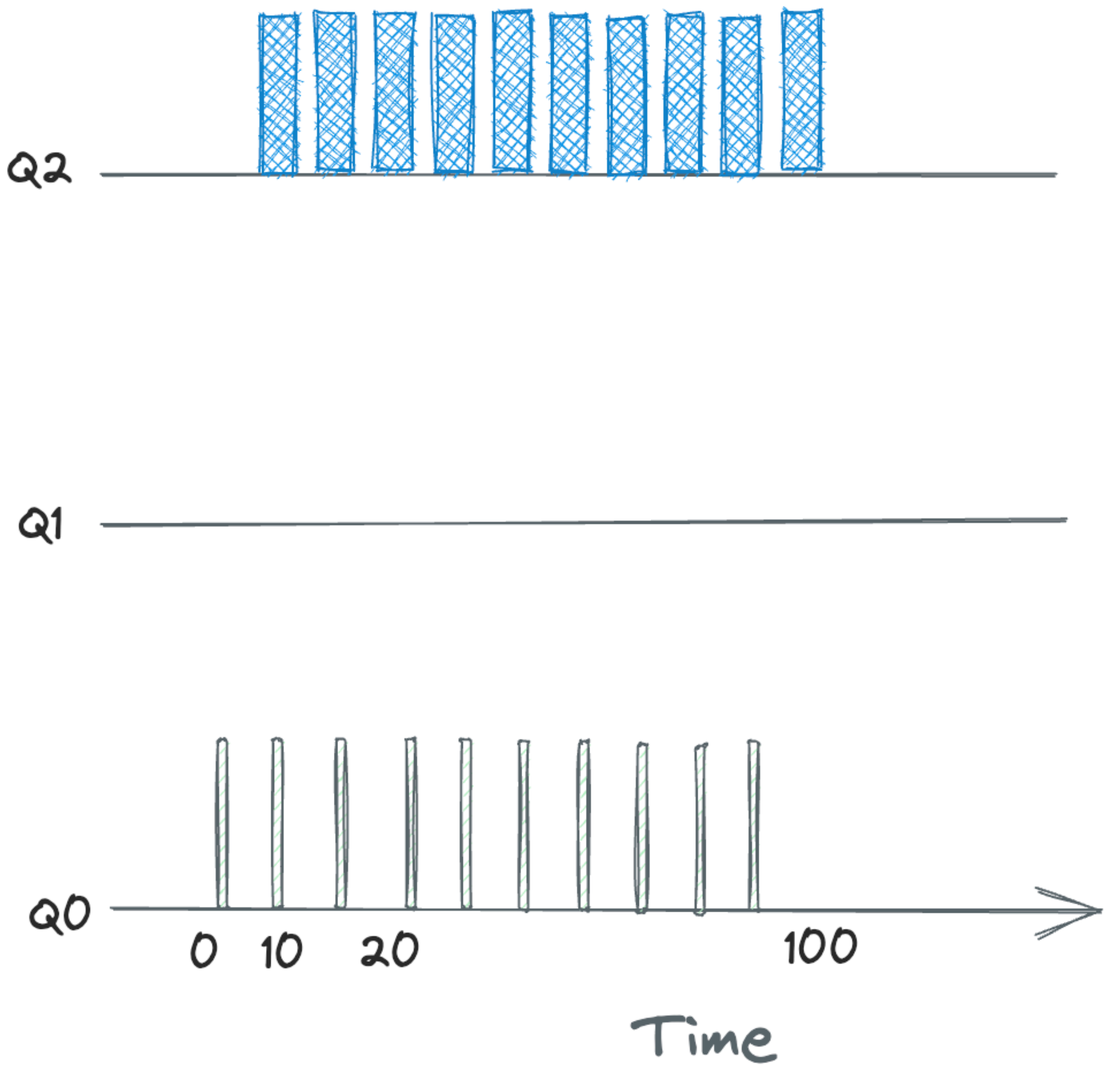
One additional Challenge

- 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

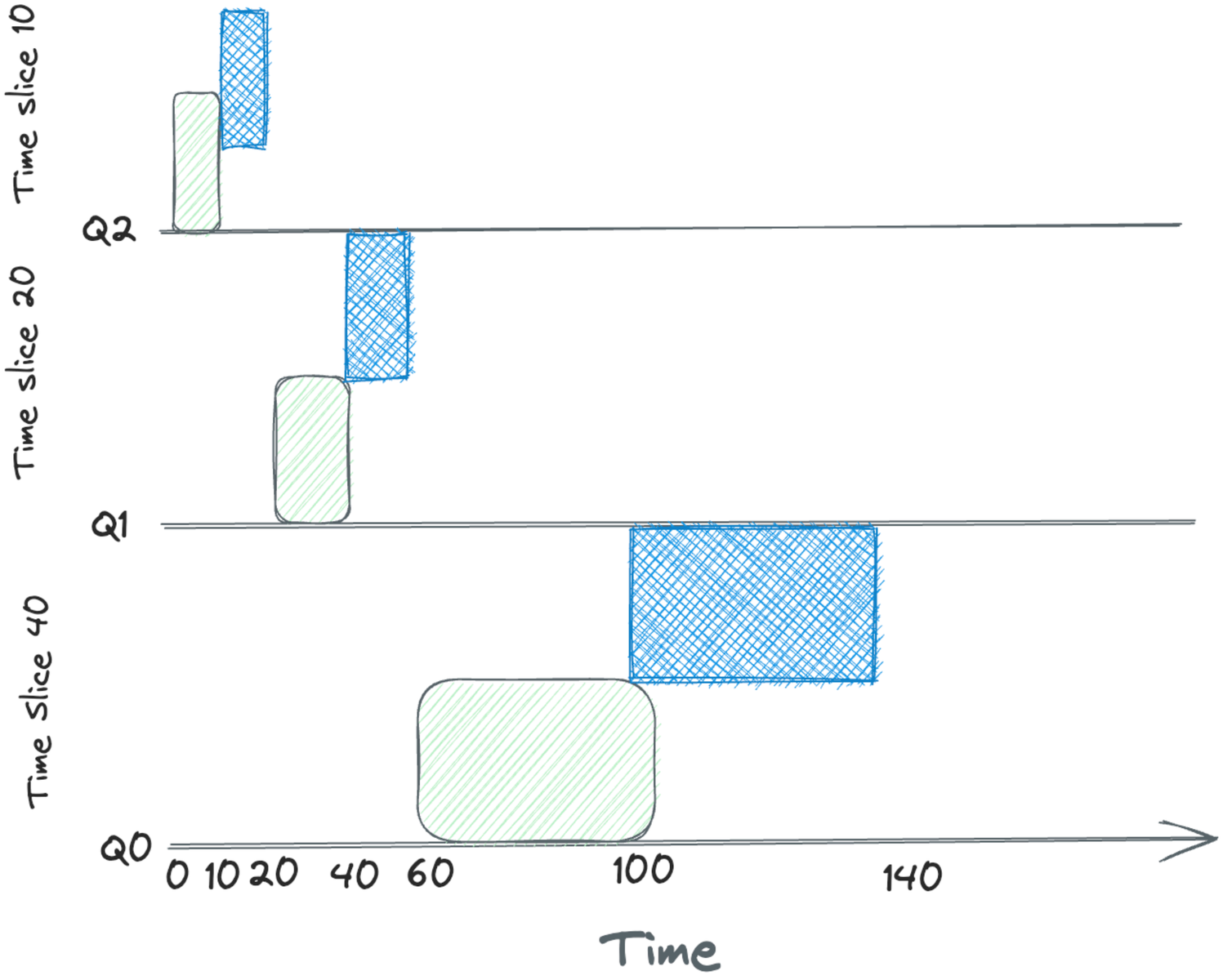
- How to parametrise the scheduler?
  - How many queues?
  - What should be the time slice?
  - How often the priority boost needs to be done?
- **High priority queues:** Interactive jobs with shorter time slices (10 ms)
- **Low priority queues:** CPU bound jobs with longer time slices (100 ms or less)

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

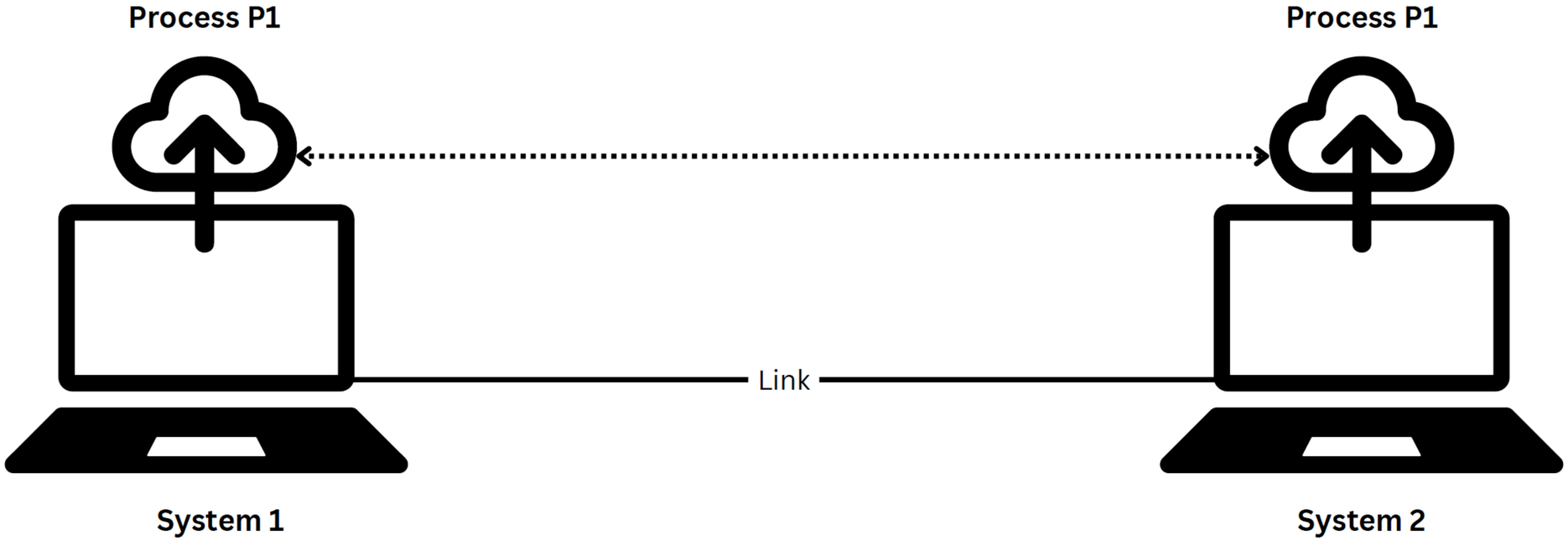
- Tables to configure:
  - How long should be the time slice?
  - How often to boost?
- 60 queues
  - Time slice length from 20ms to few 100 milliseconds
  - Priority boosted every 1 second
- Free BSD scheduler uses formula to calculate priority

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

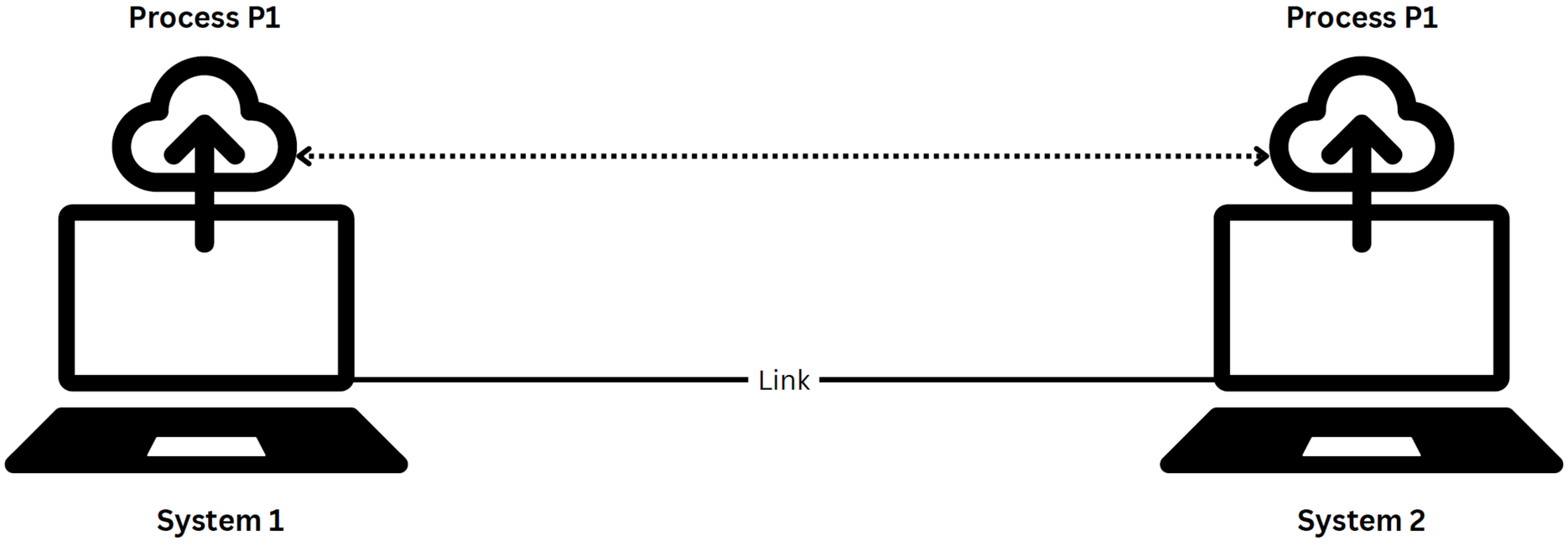
The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge -- Karthik Vaidhyanathan

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?

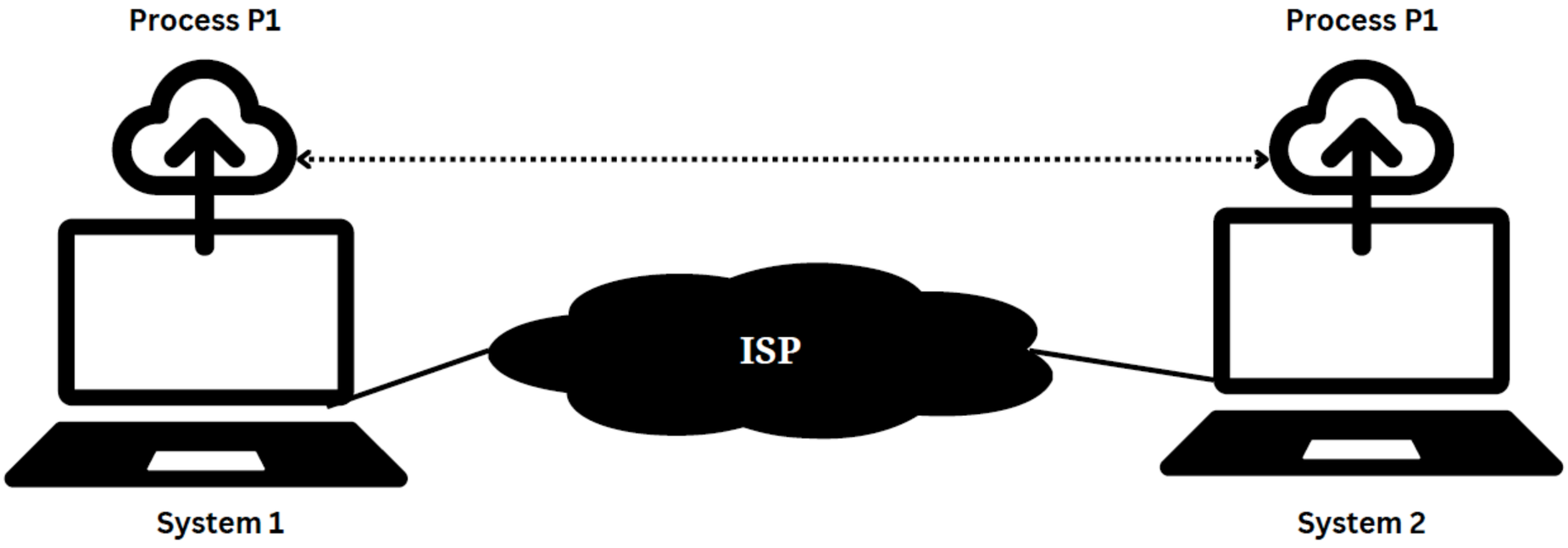
How can they communicate?



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?

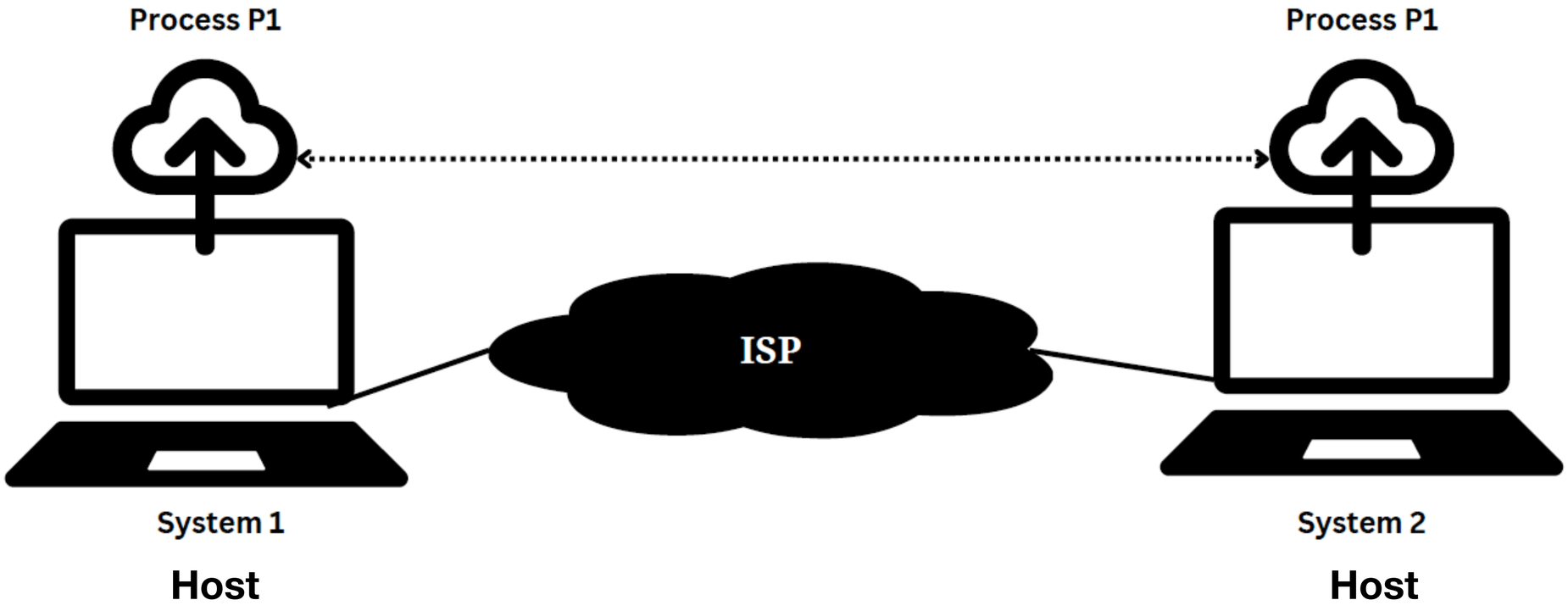
Let us expand it



- 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

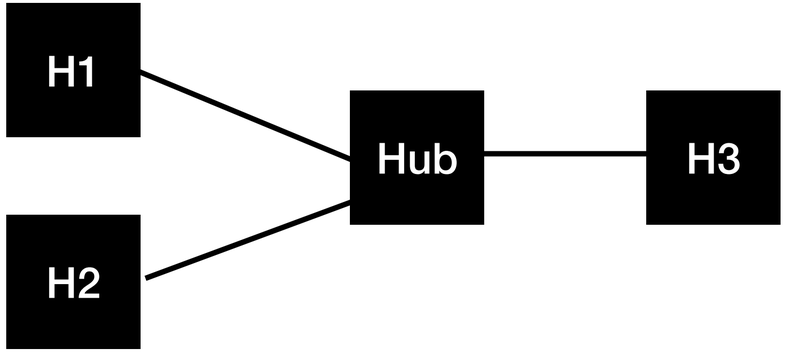
Repeater

- 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

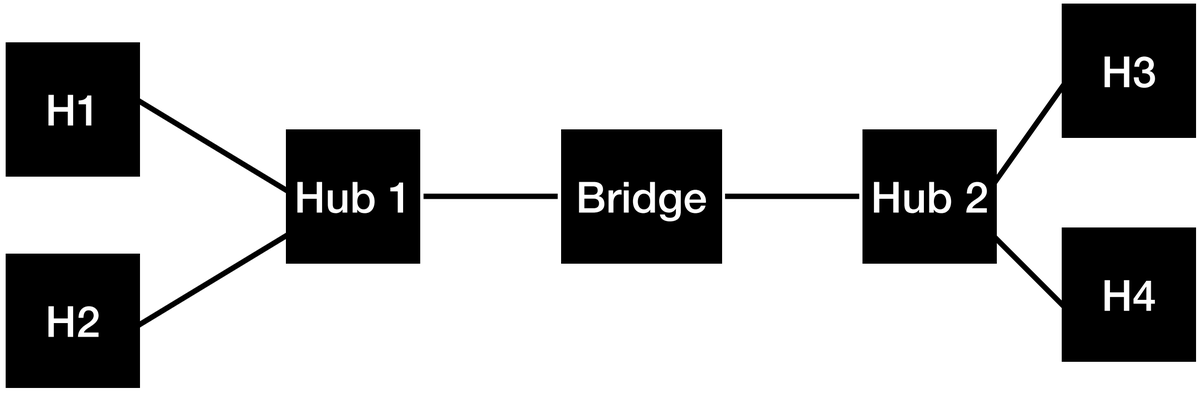
Hubs

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



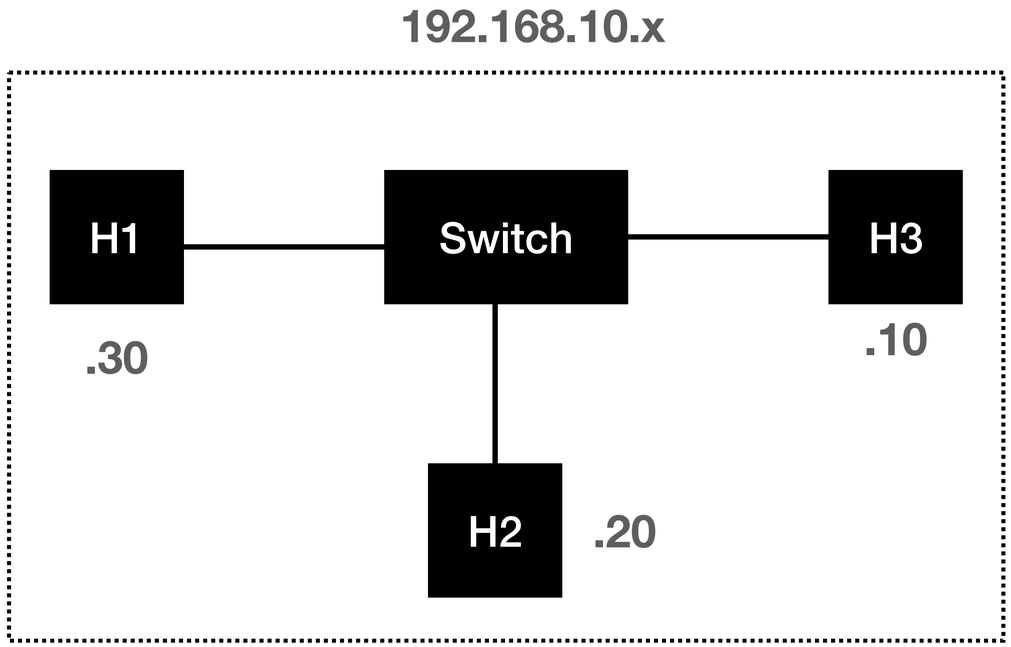
Bridges

- 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

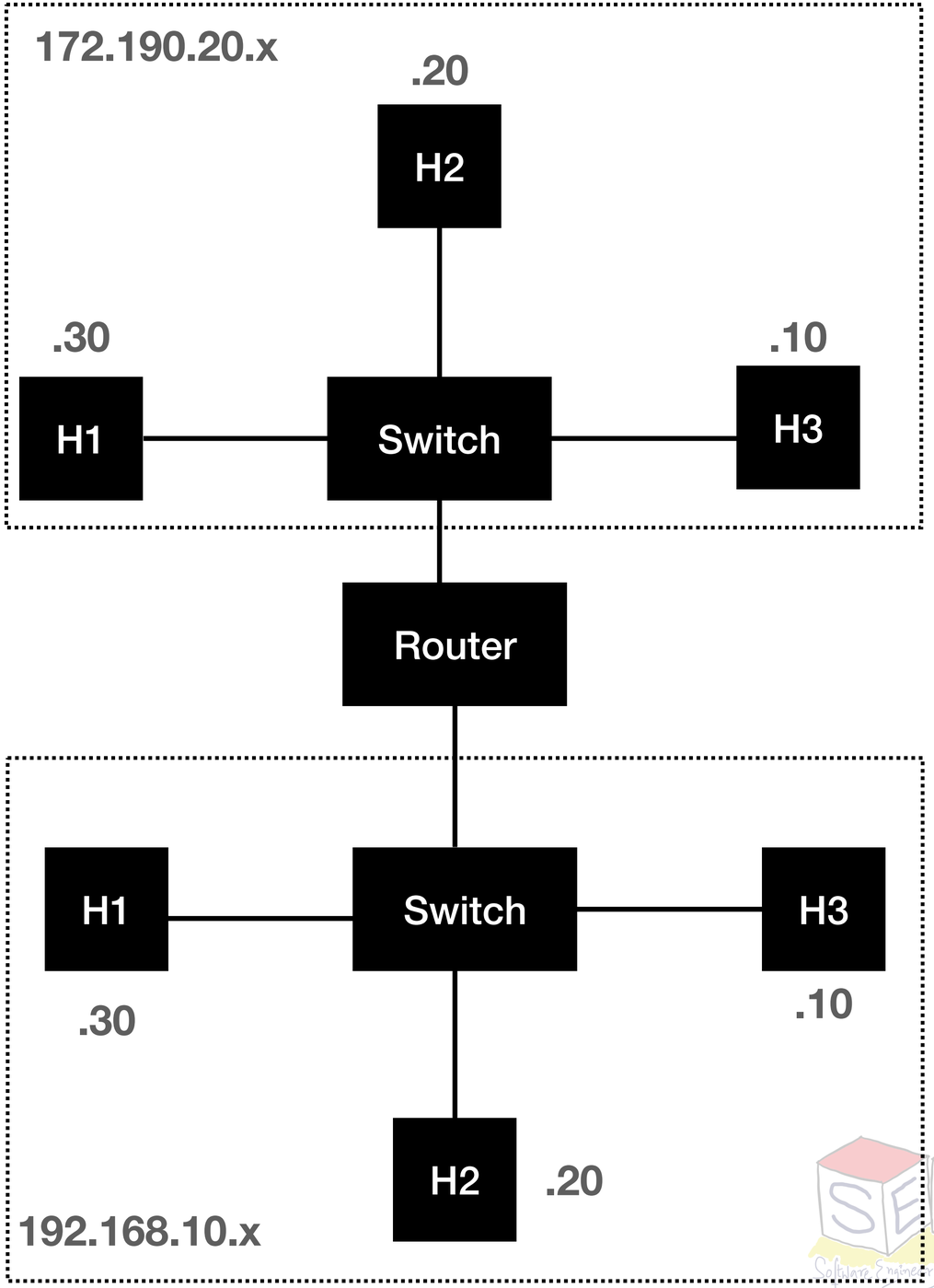
- Devices which facilitate communication within a network
- Combination of hubs and ports
- Knows or infers which hosts are on each port
- They have multiple ports
- Whatever connected to switch becomes part of one network



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

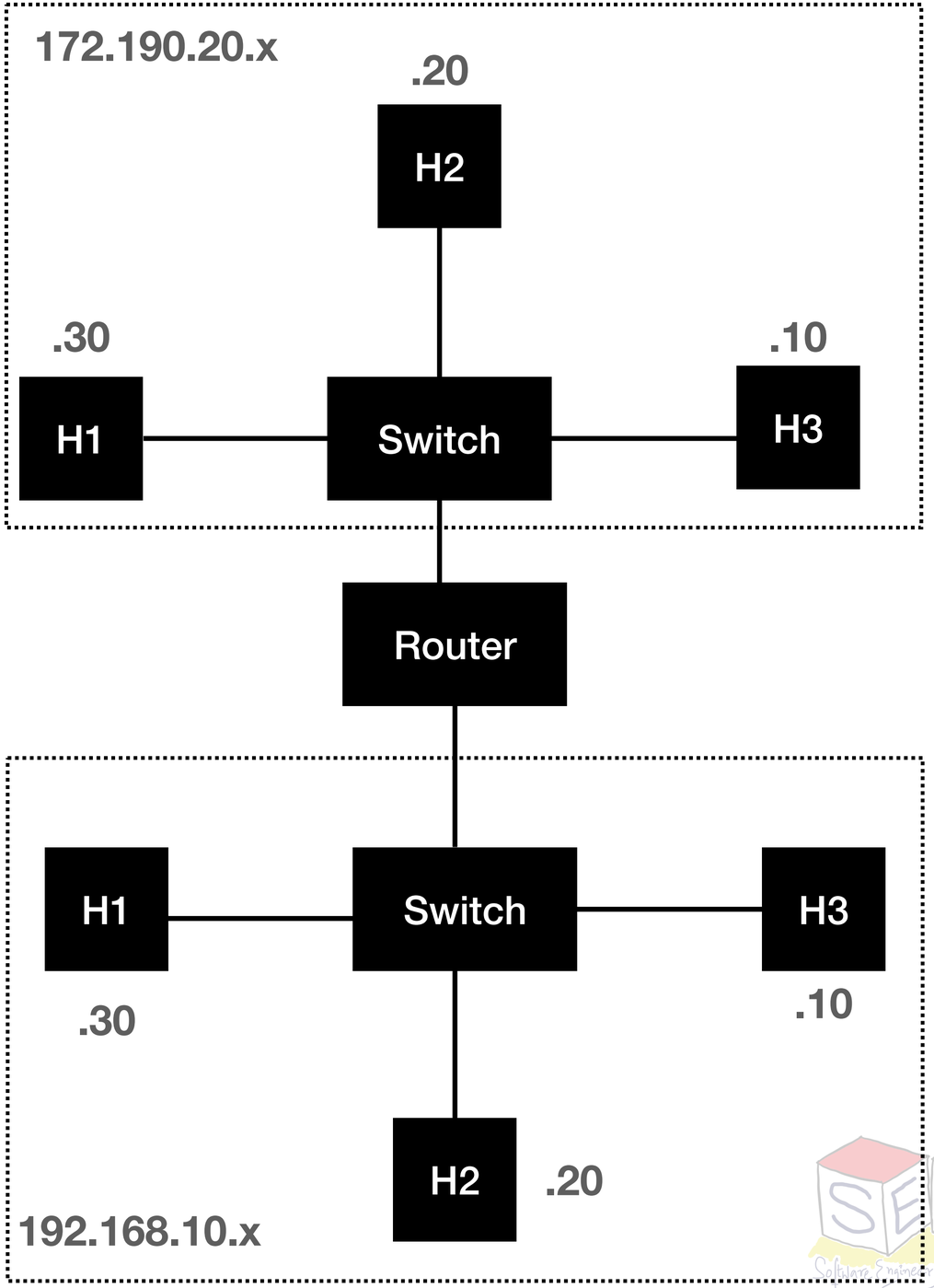
Routers

- 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

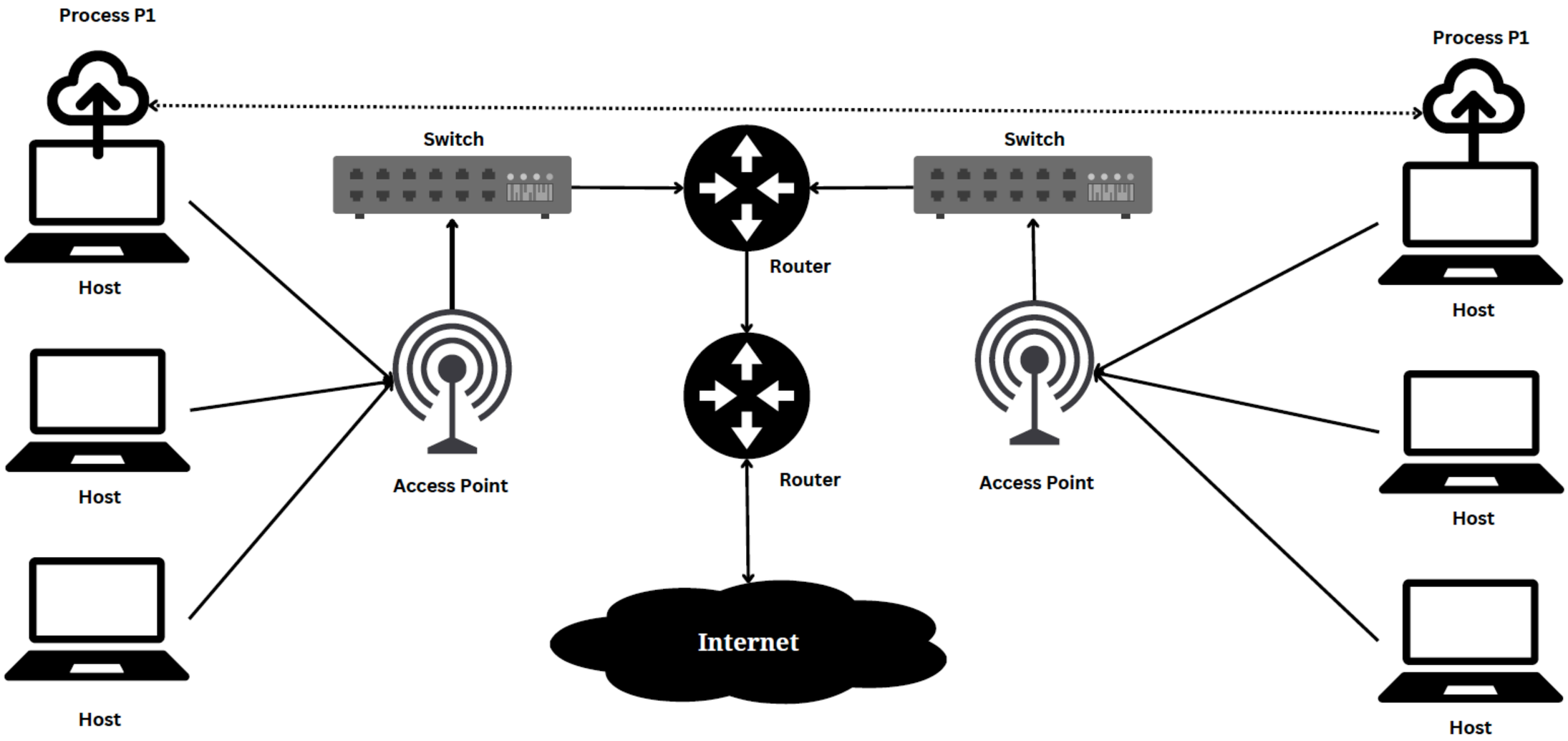


Routers

- 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

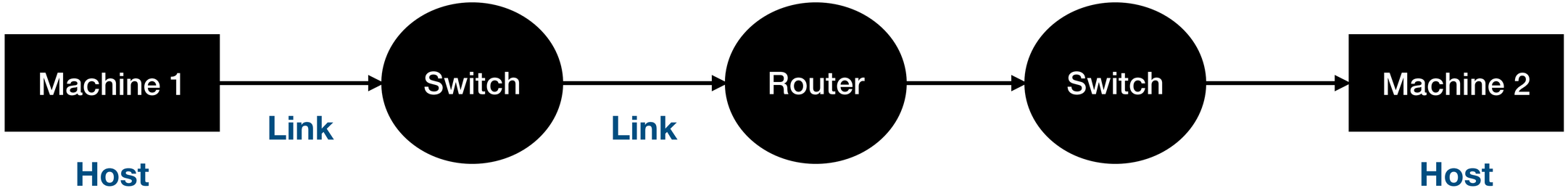


The Bigger Picture



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

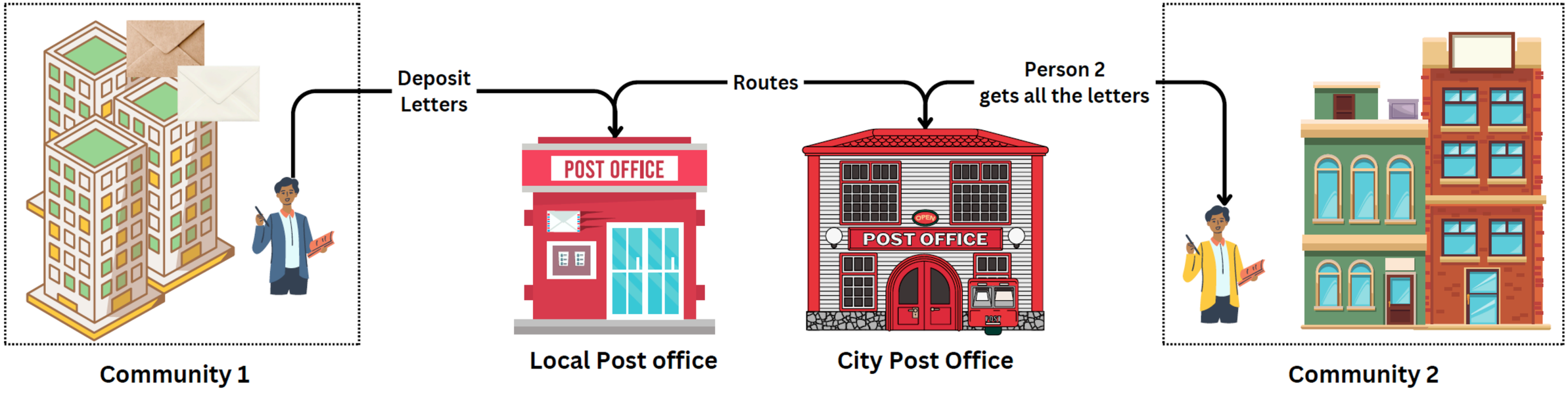
Two large communities in different parts of the city, two people in each community to collect and deliver the letters/couriers

An Illustrative Scenario

People in community send letters/couriers

One person is collecting all the letters/couriers

Person delivers the letter to each person

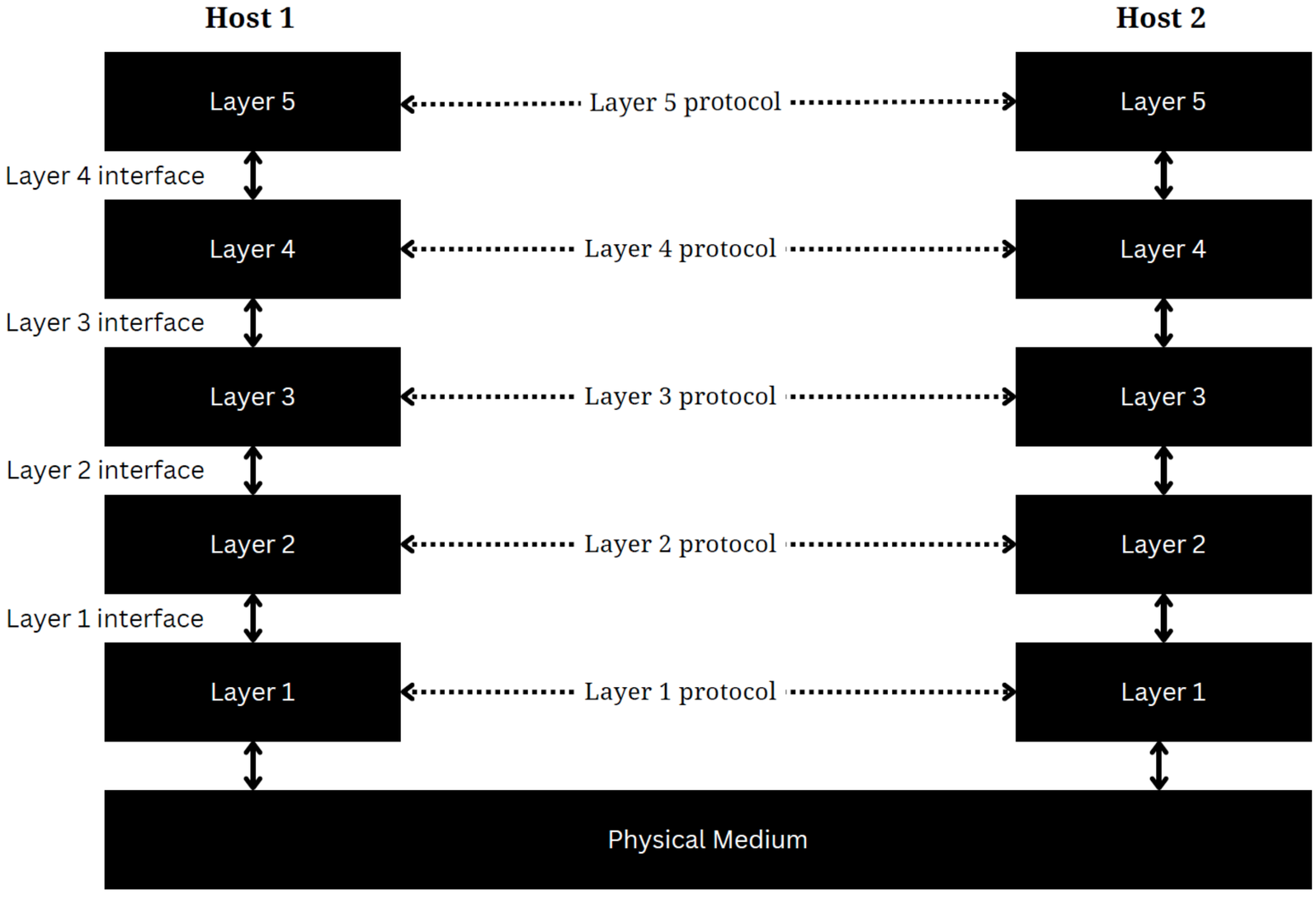


Key points: Door number, building number, Post office has an identifier, etc.

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

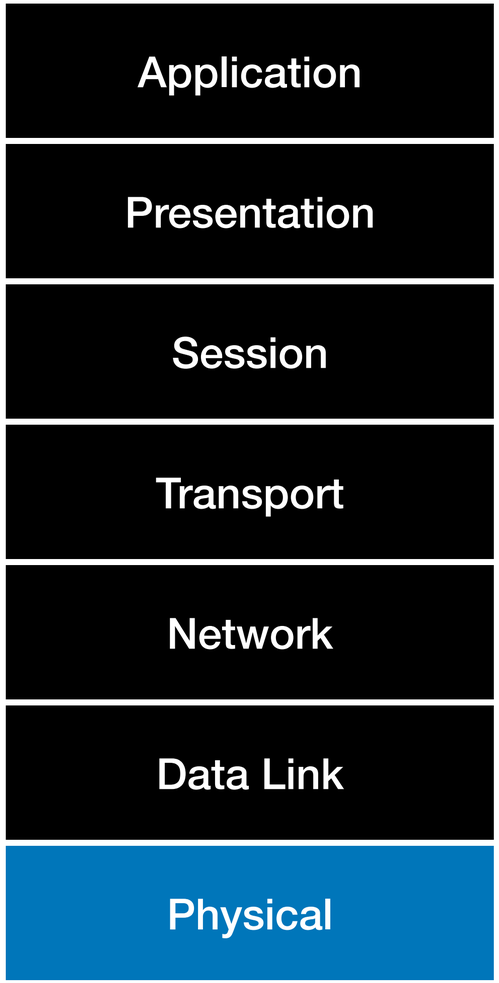
Application
Presentation
Session
Transport
Network
Data Link
Physical

- 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

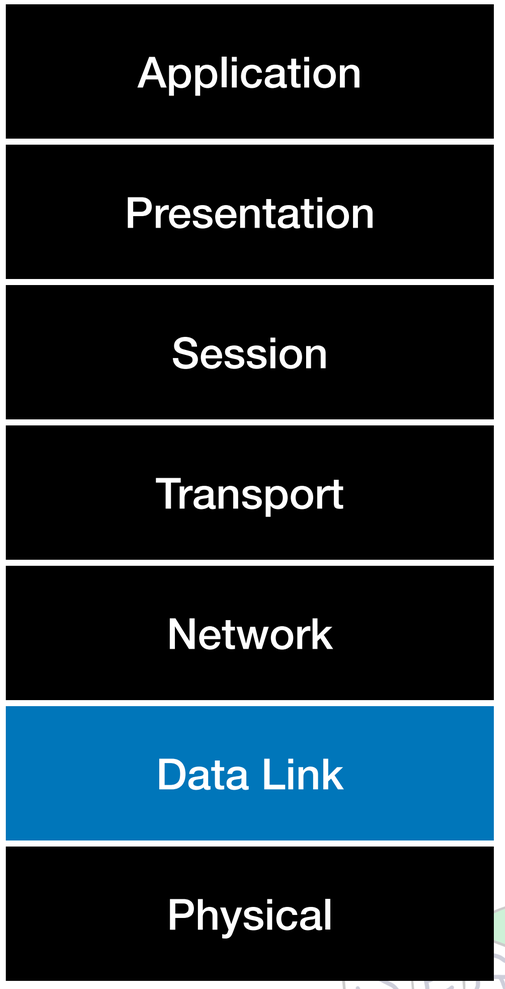
- Data is in the form of bits - 0s and 1s
- Something has to transport the bits from one machine to another - Physical layer
- Concerned with transmission of raw bits over physical medium, like a cable
- L1 technologies: Ethernet cables, Optical fiber, Coaxial cable, etc.
    - Even WiFi is L1 technology,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



Data Link Layer (L2)

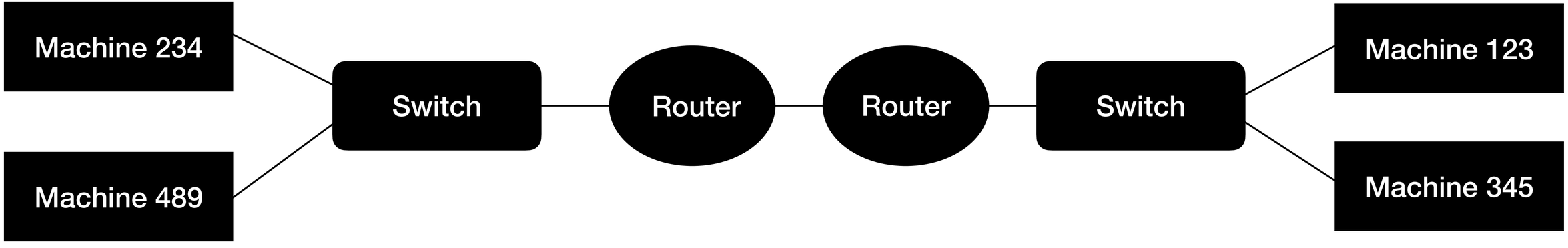
Hop to Hop connectivity

- 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?

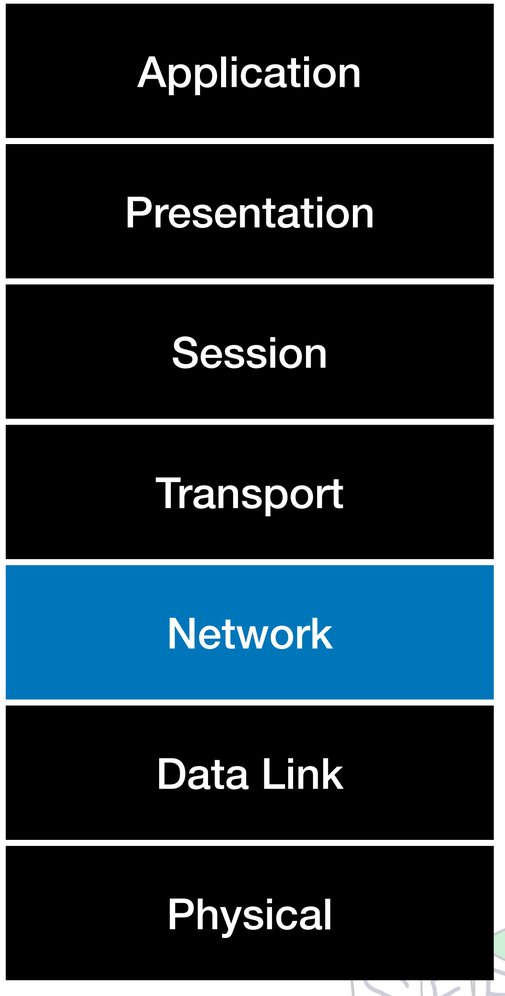


- Hop to hop is guaranteed by L2
- What about end-to-end delivery?

Network Layer (L3)

End-to-end Communication

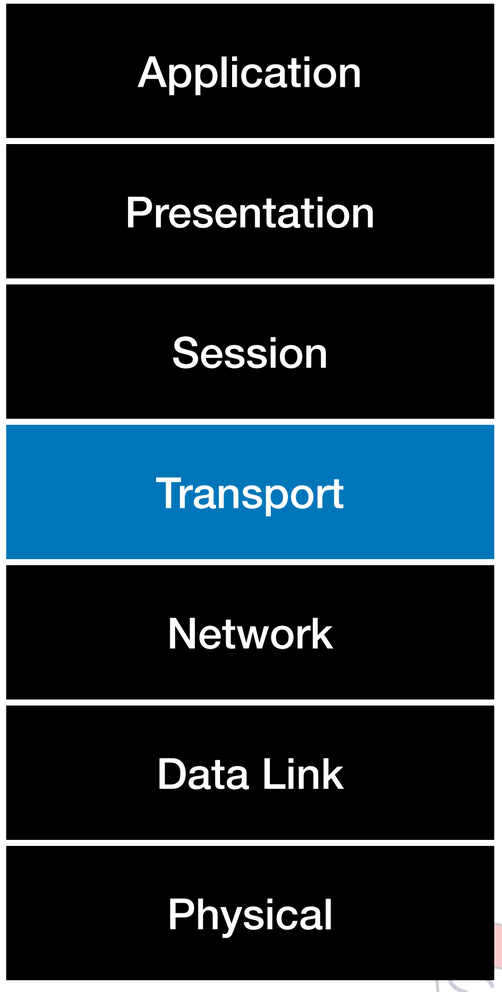
- Manages routing through different routes in a large network
- Uses an addressing scheme - IP addressing
  - 32 bits represented as 4 octets (IPv4)
- Performs functionalities such as Logical addressing (IP), Path selection and packet forwarding
- L3 technologies: routers, even hosts are L3, L3 switches



Transport Layer (L4)

Service to Service

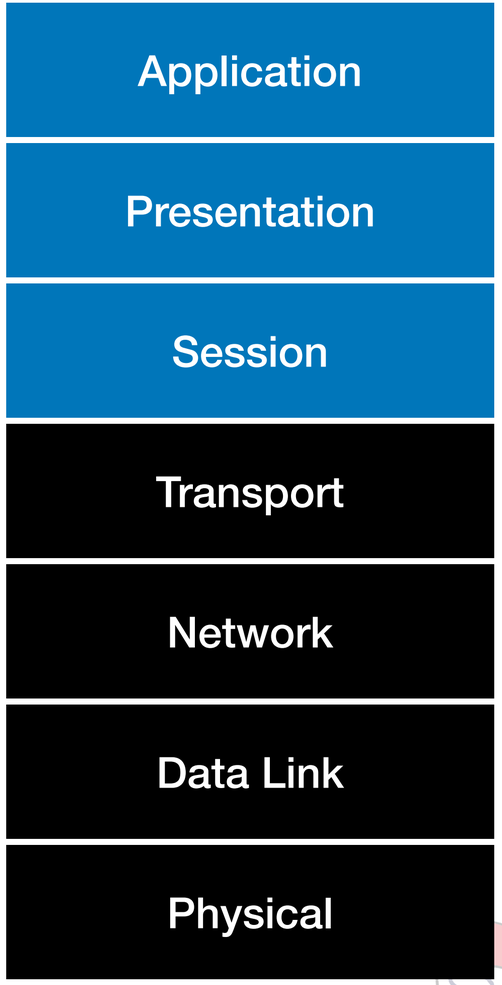
- How to ensure that the right process receives the data?
  - Many process will be executing or are waiting to execute?
- Ensures data transfer is reliable, sequential and free from others
- Manages flow control and error correction
- Layer 4 has an addressing scheme to guarantee message delivery
  - 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



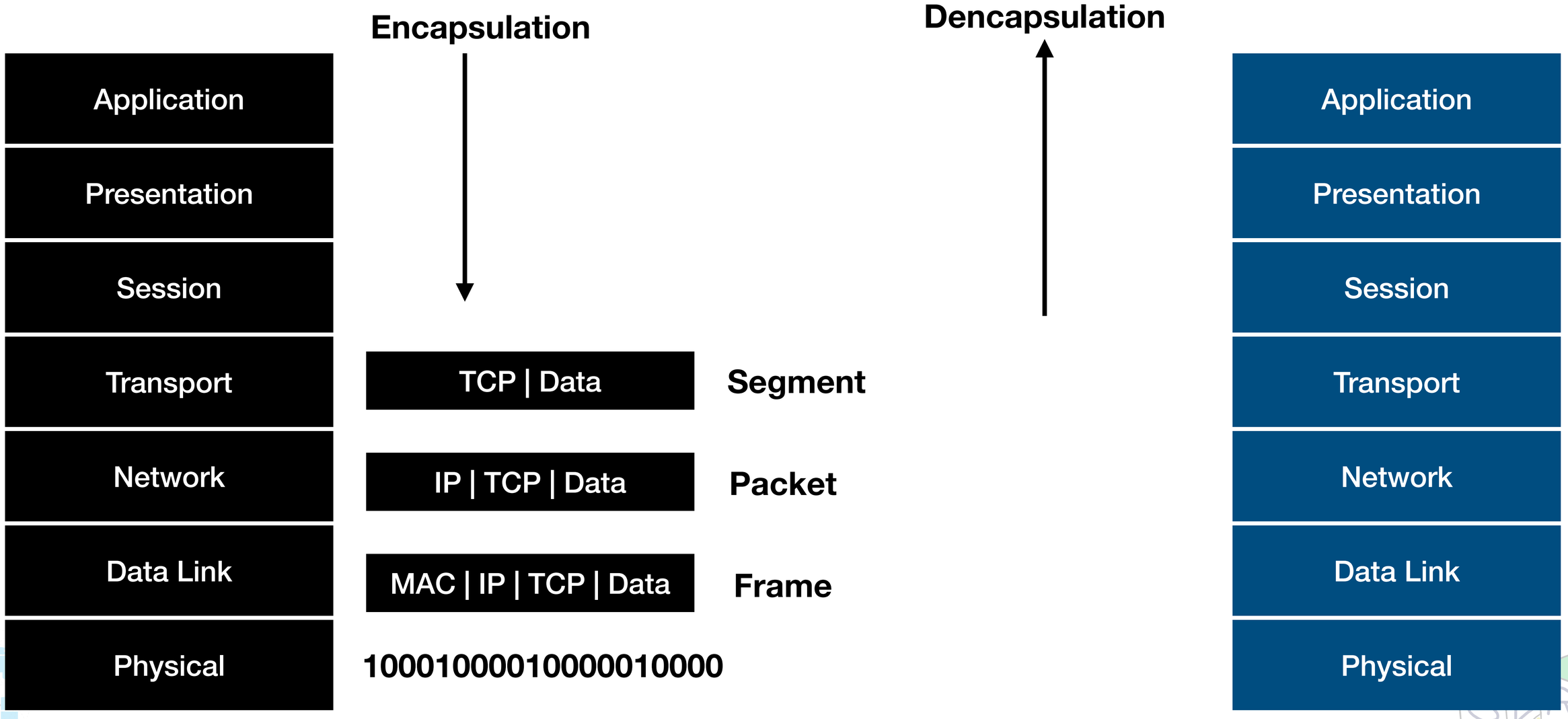
Session, Presentation and Application

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



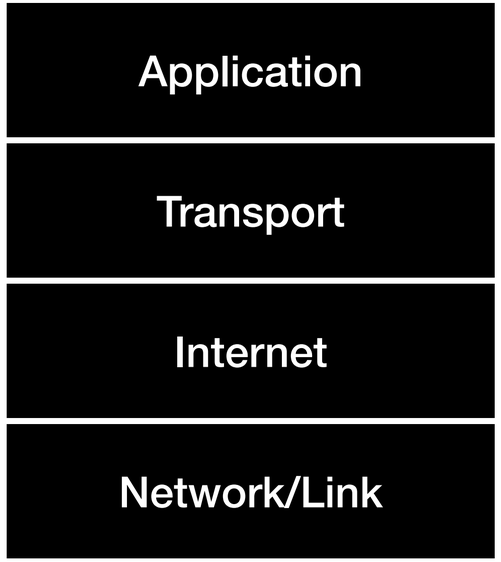
Putting It Together



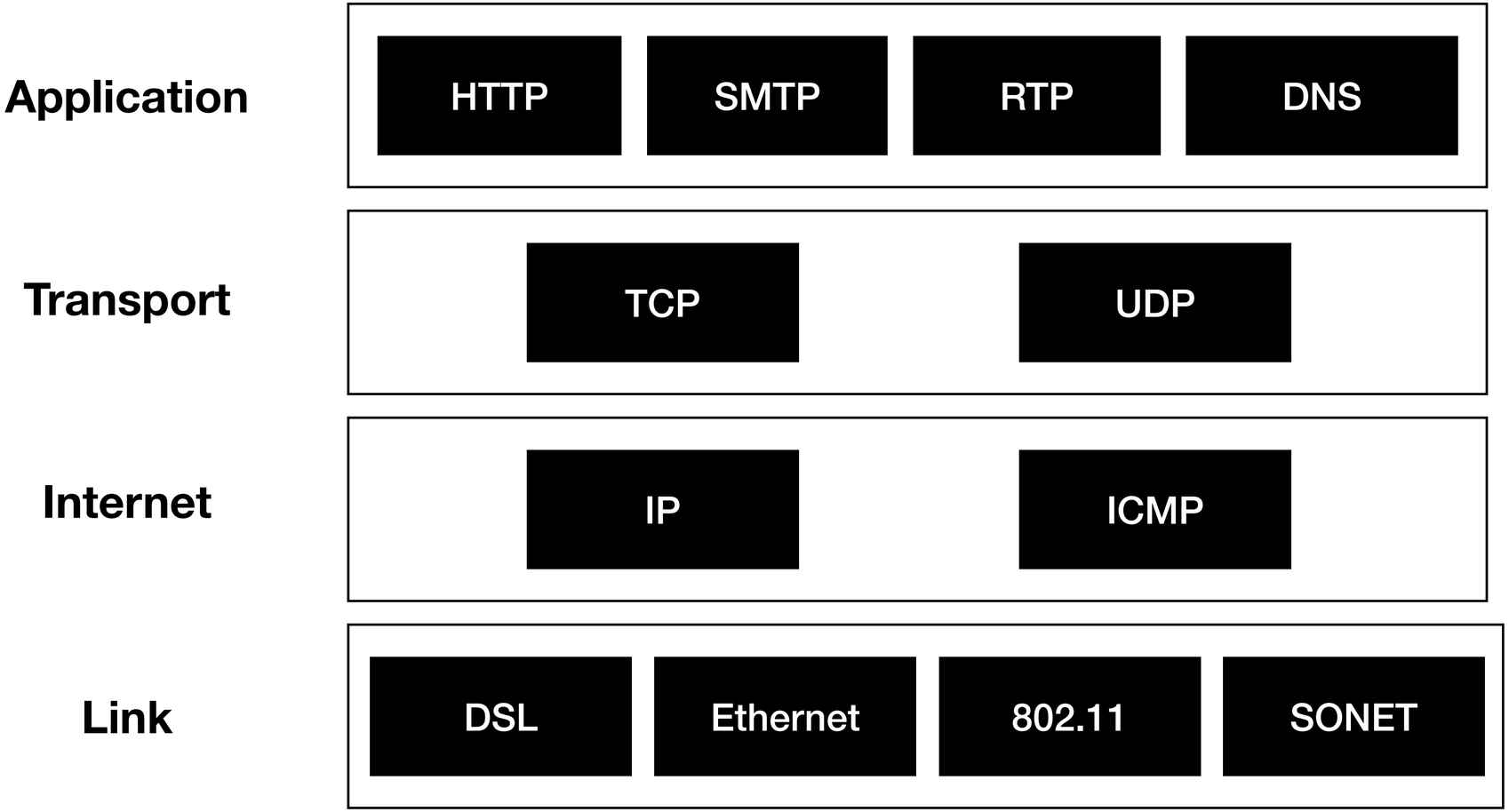
The 4 Layer Model

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

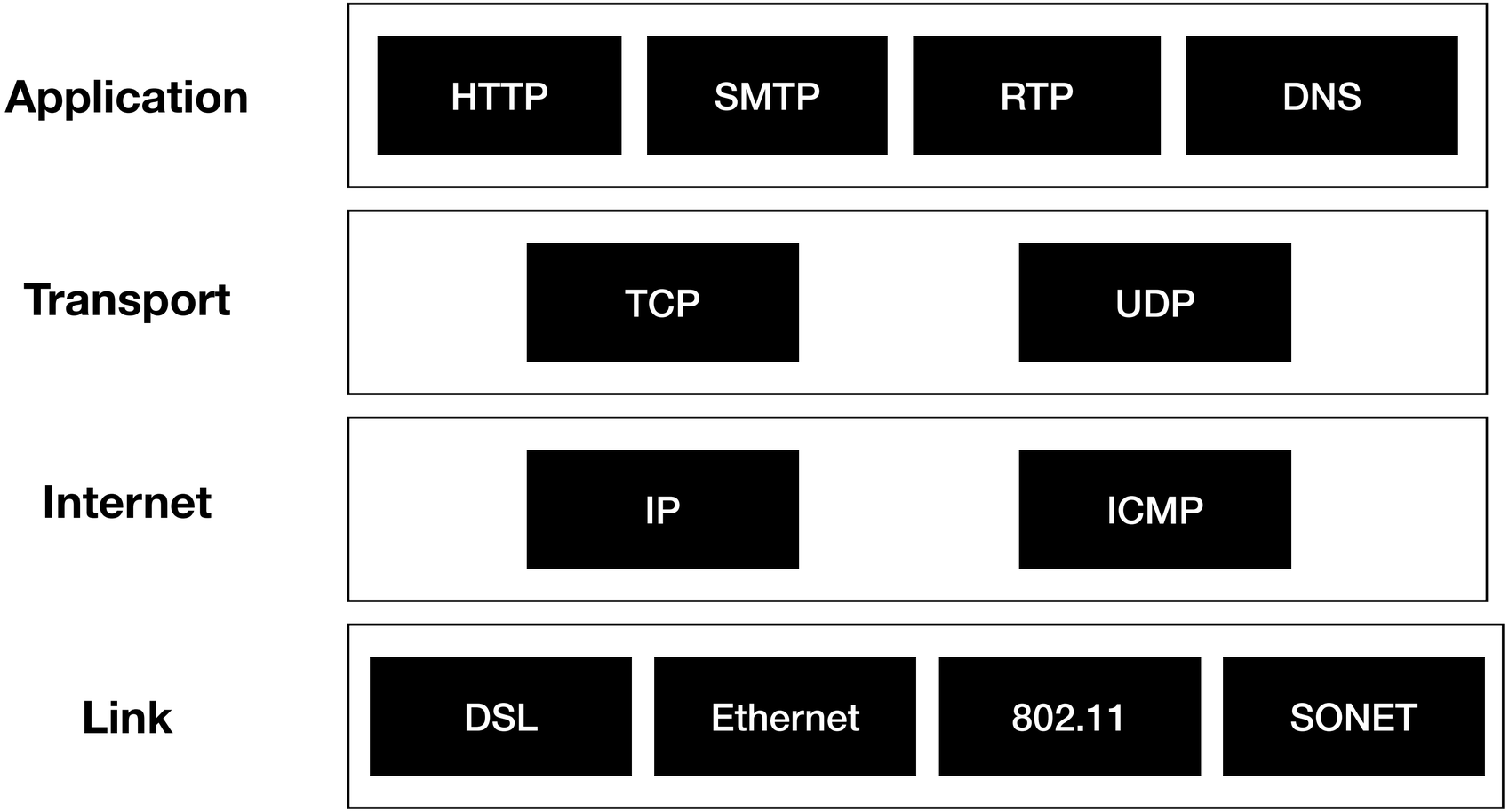
Acknowledgement

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

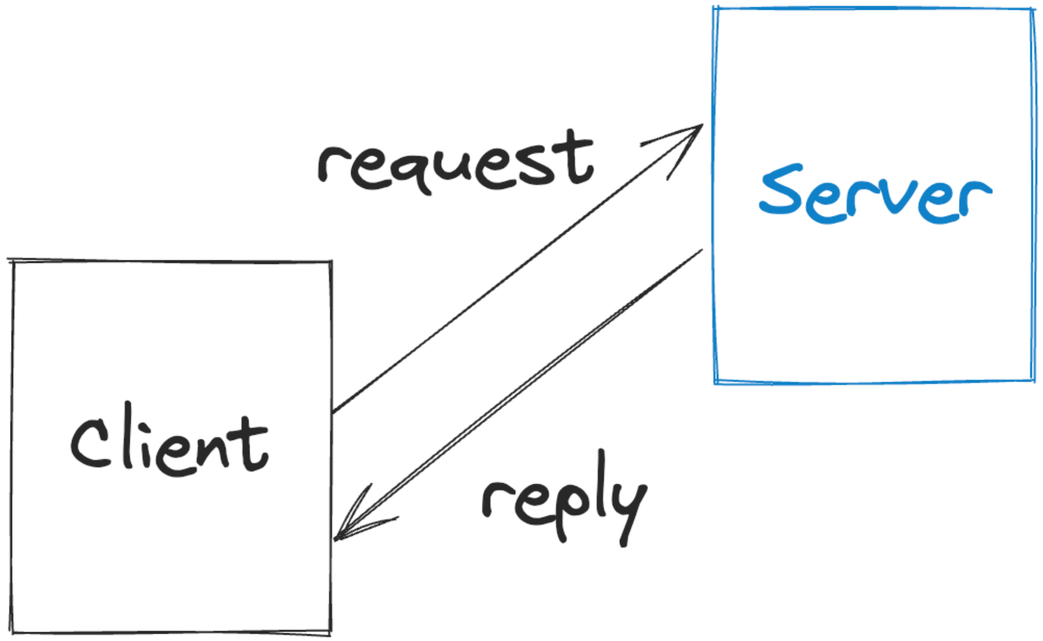
Service to Service Delivery

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

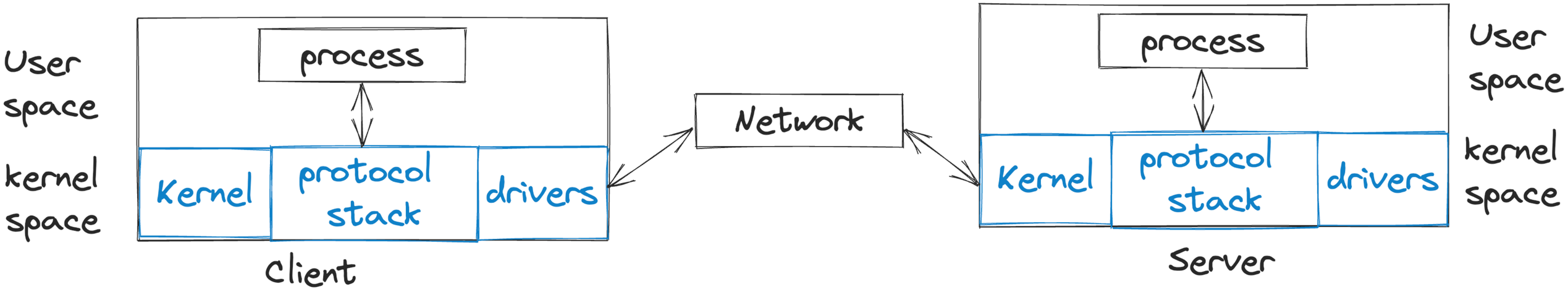
Lets go back

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

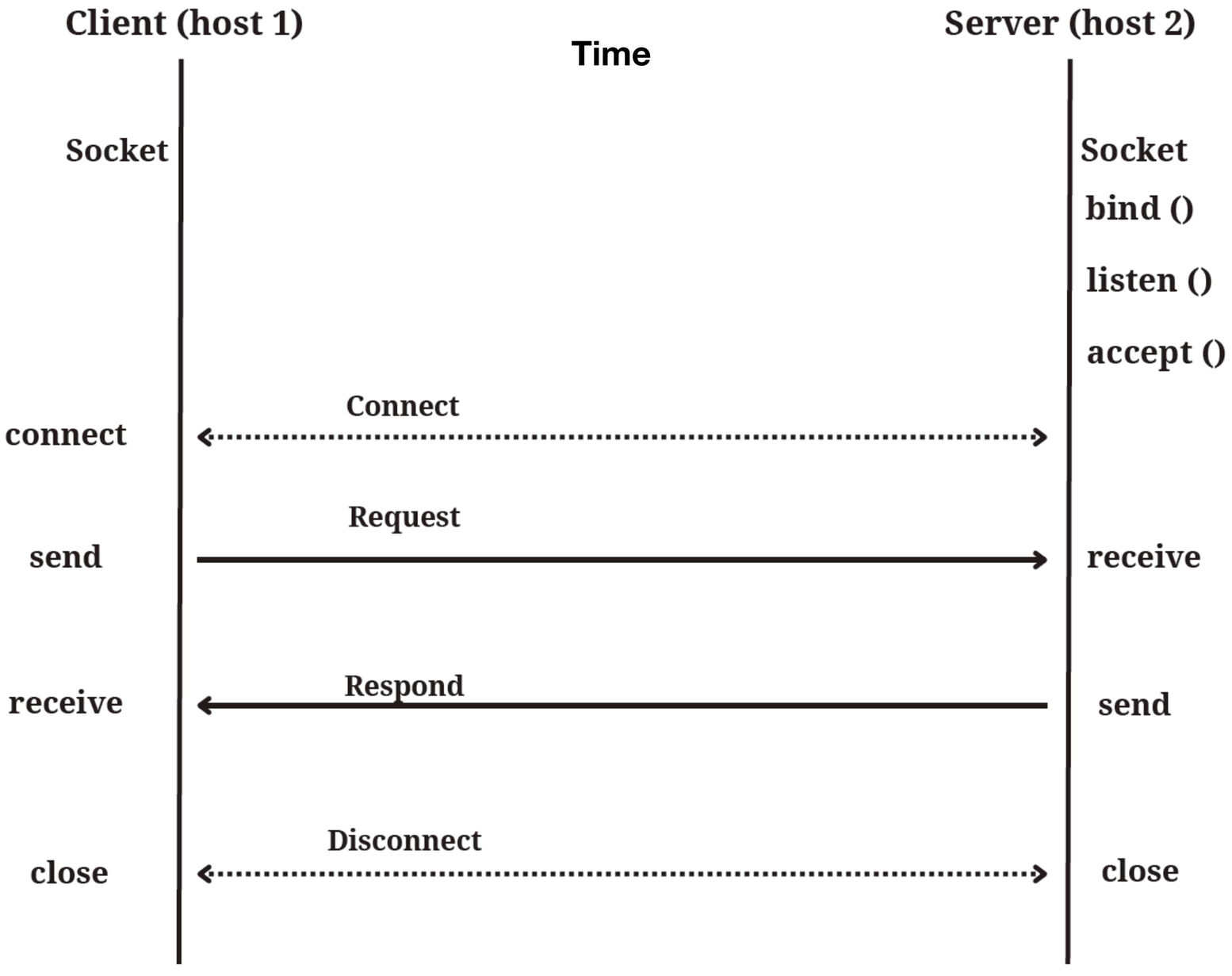
The Socket API

- 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

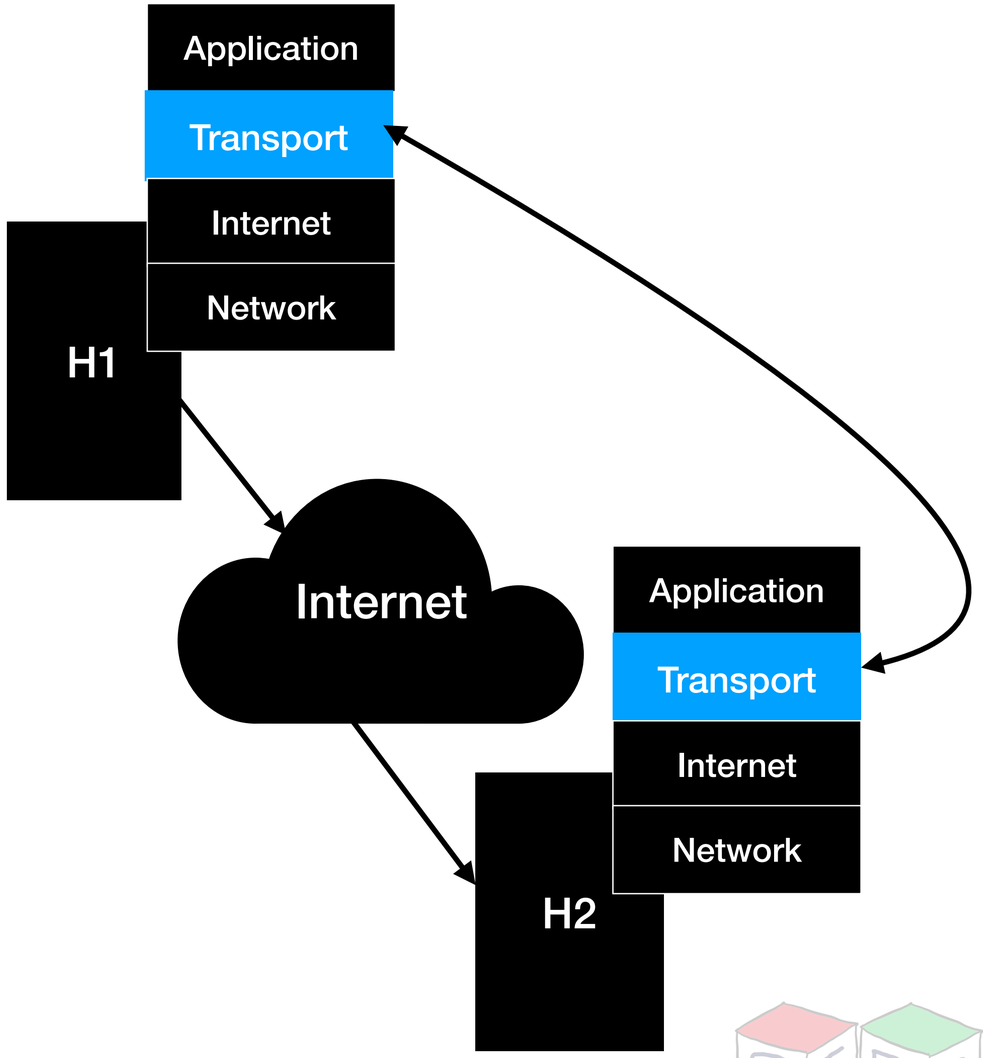
Let us take a step back

Types of Links

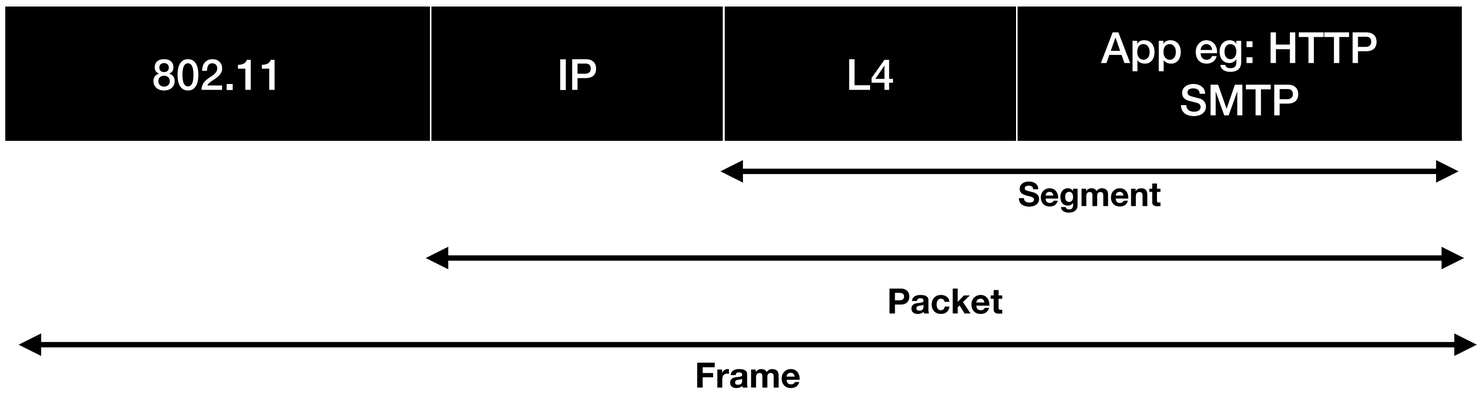
- **Full Duplex**
  - Bidirectional
  - Both sides at the same time
- **Half-duplex**
  - Bidirectional
  - Both the sides but only one direction at a time (eg: walkie talkies)
- **Simplex**
  - Unidirectional

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



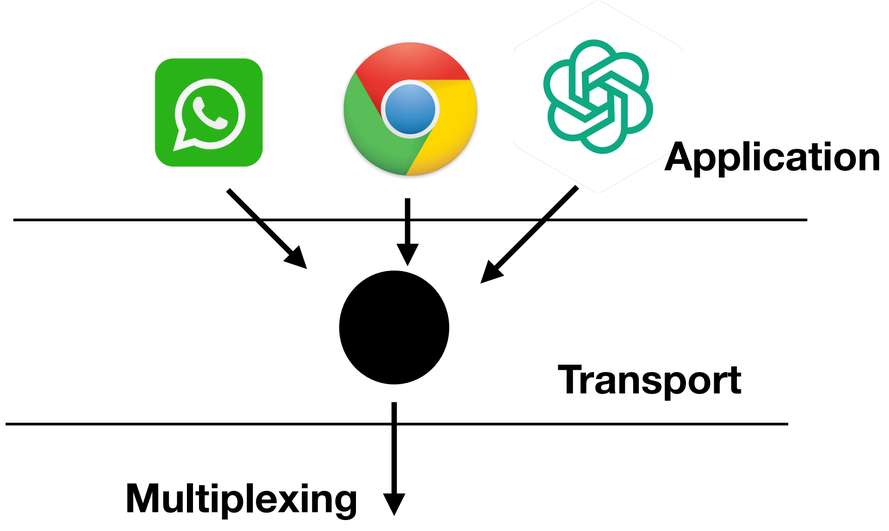
Quick Recap



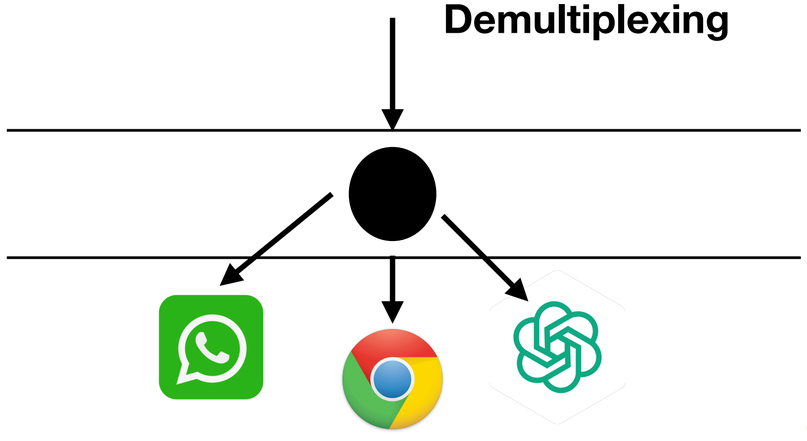
- Segments carry data across the network
- **Segments** are carried within the packets, within frames
- Each layer adds a **header** (Above L4 will be replaced by its header)

Multiplexing and Demultiplexing

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

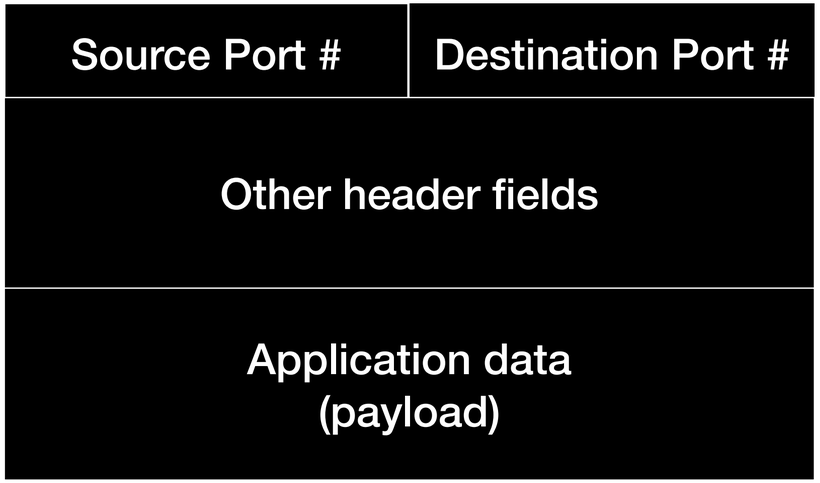


- **Demultiplexing as receiver**: Use header info to deliver received segments to correct socket



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

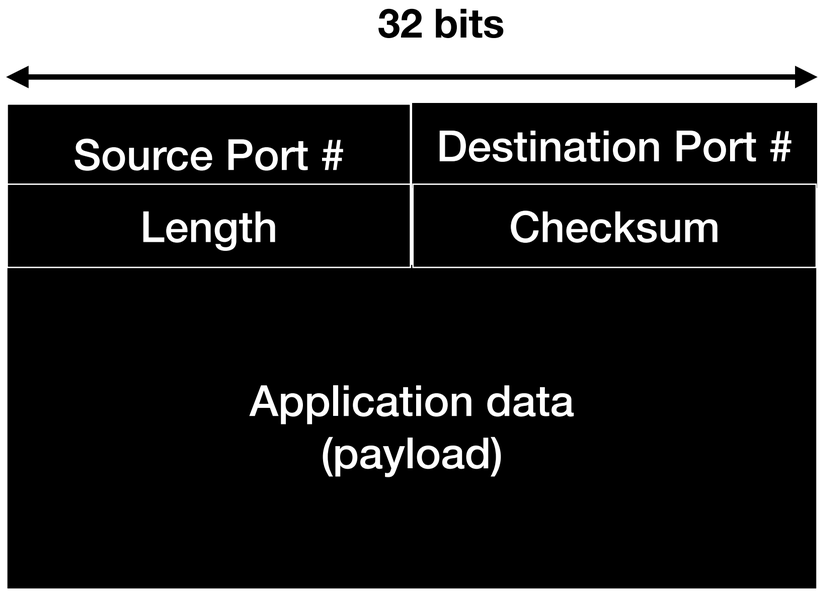
Flow Control and Overhead

- 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?

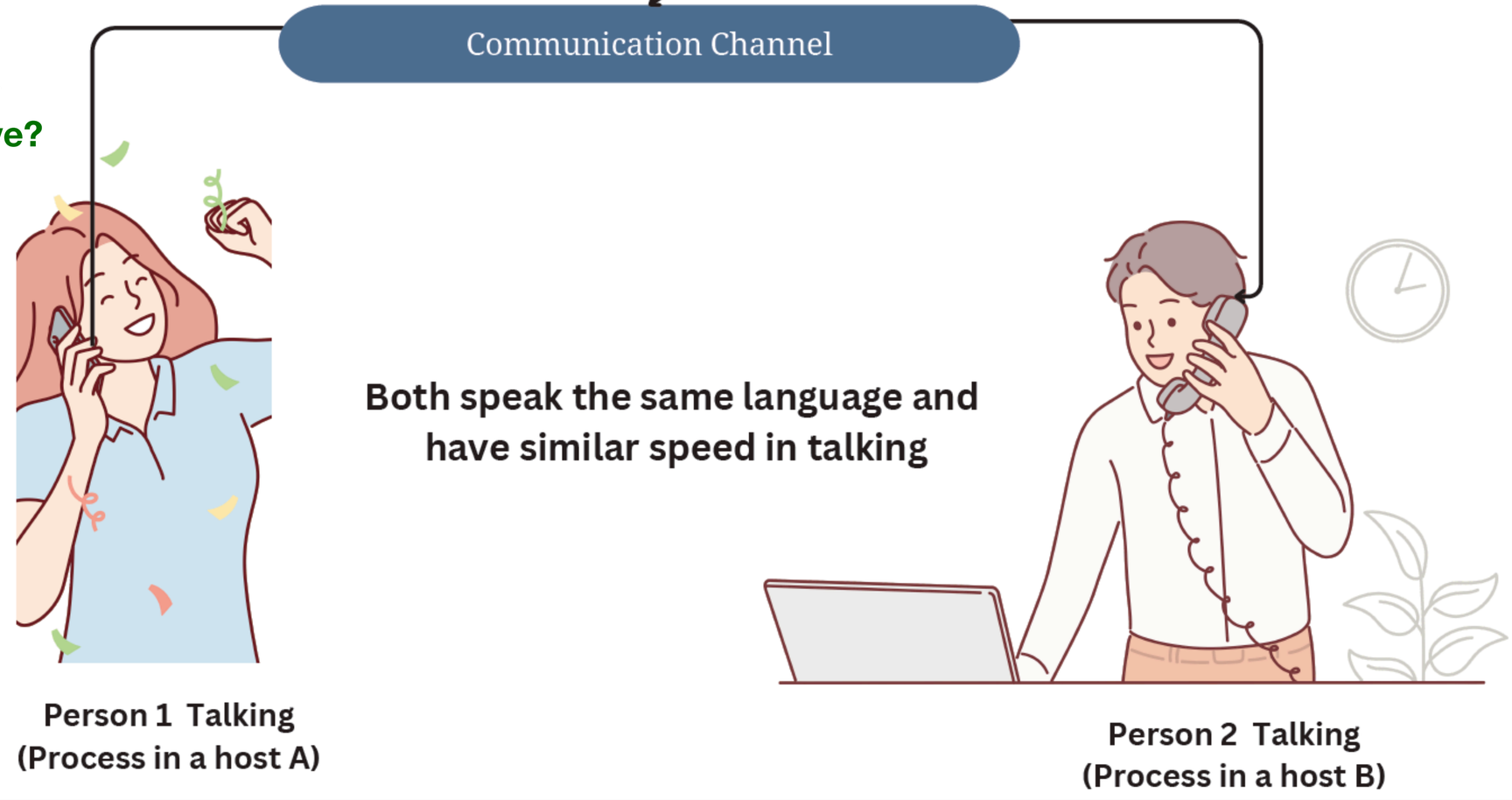
What all you need to provide some features that TCP provides?

A Small Analogy

Network and Link Layer

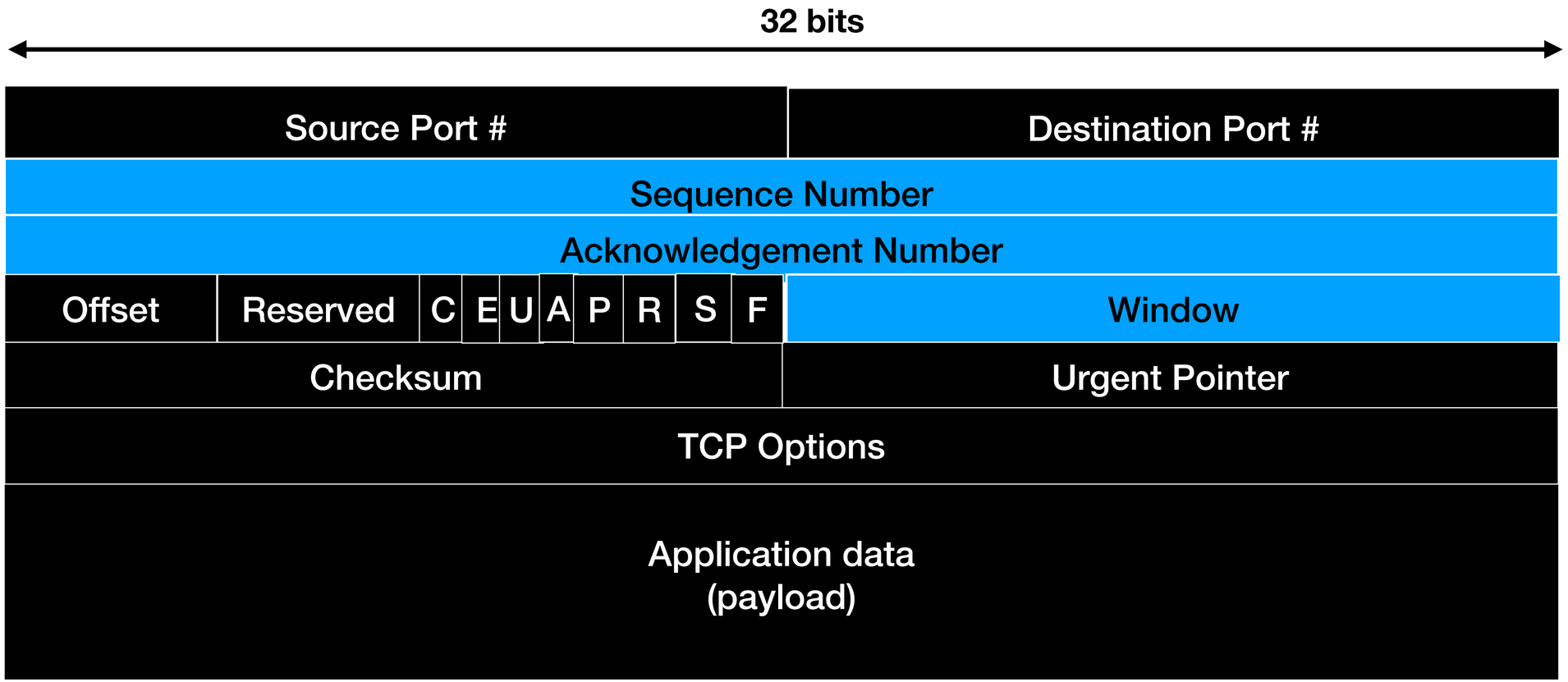
Communication Channel cannot be always reliable!!

What can we do from the protocol perspective?



Do we foresee some challenges?

Lets go into TCP - Header



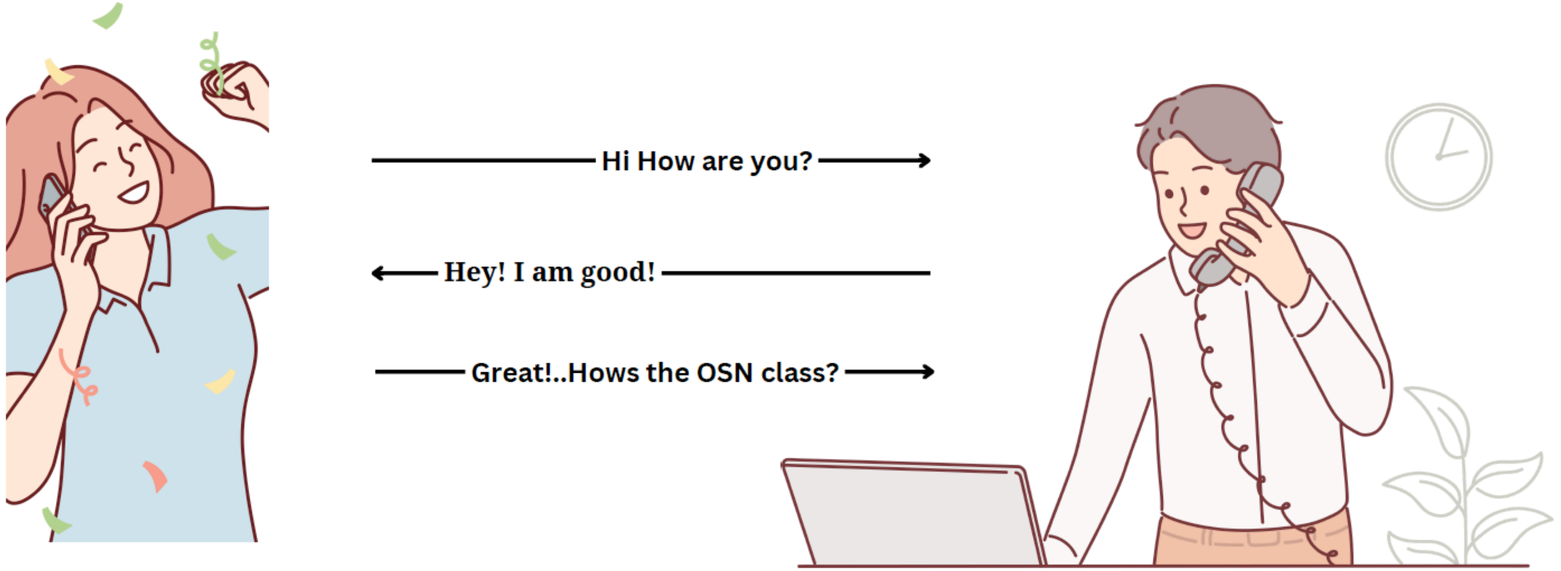
TCP Segment Header

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)

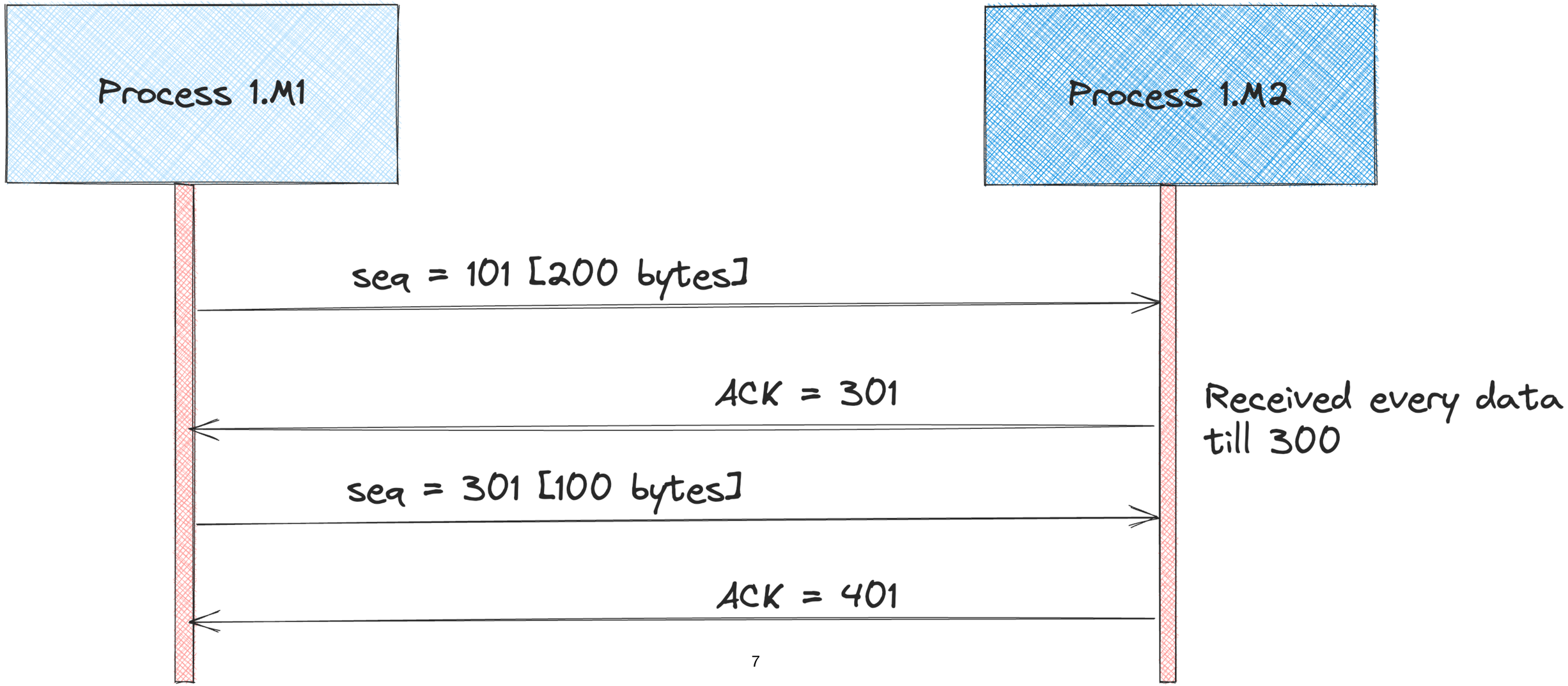
Person 2 Talking
(Process in a host B)

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

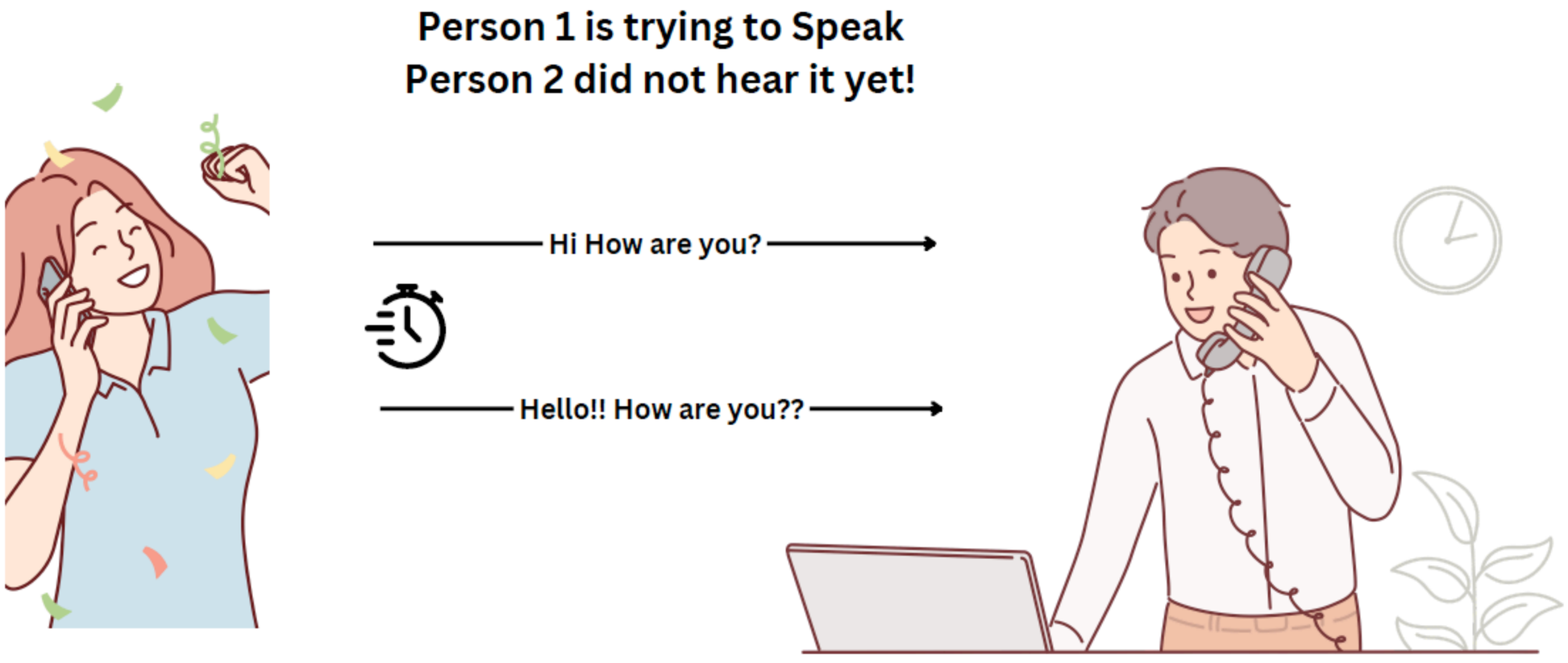
What do ACK and Sequence Number do?

Reliability!!



How to handle if data is lost?

Can we retransmit?



Person 1 Talking
(Process in a host A)

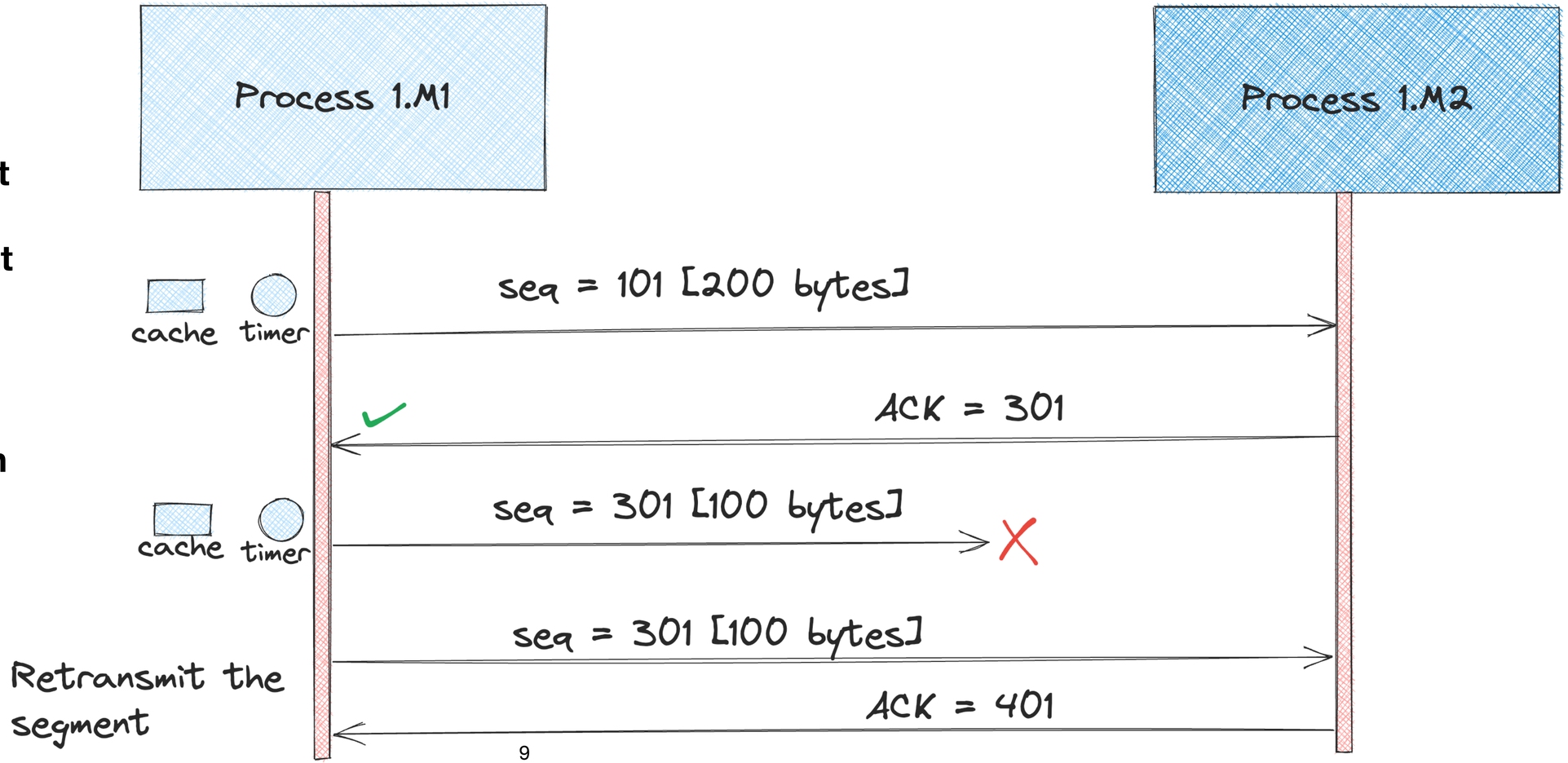
Person 2 Talking
(Process in a host B)

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?



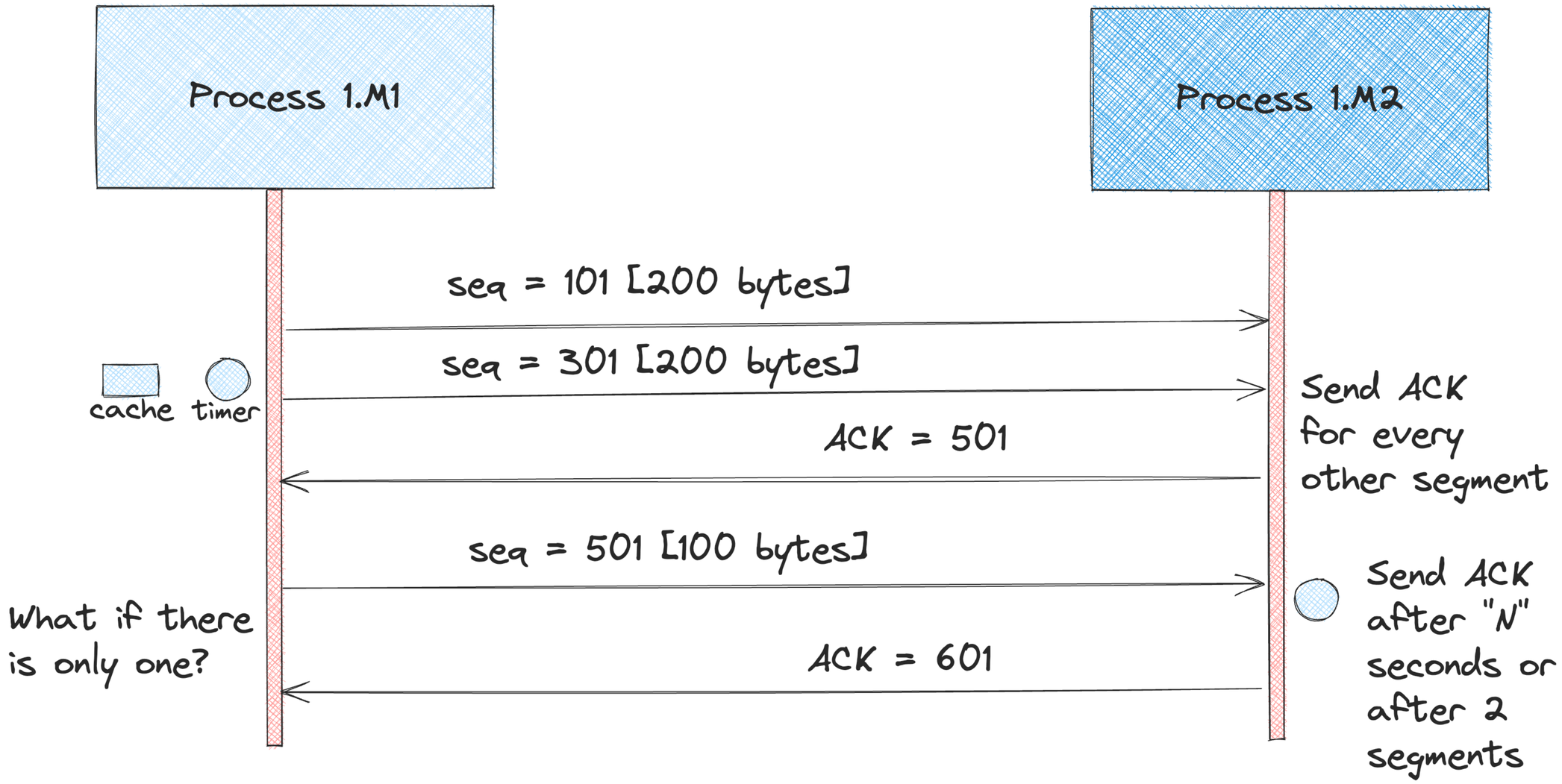
How to calculate RTT?

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

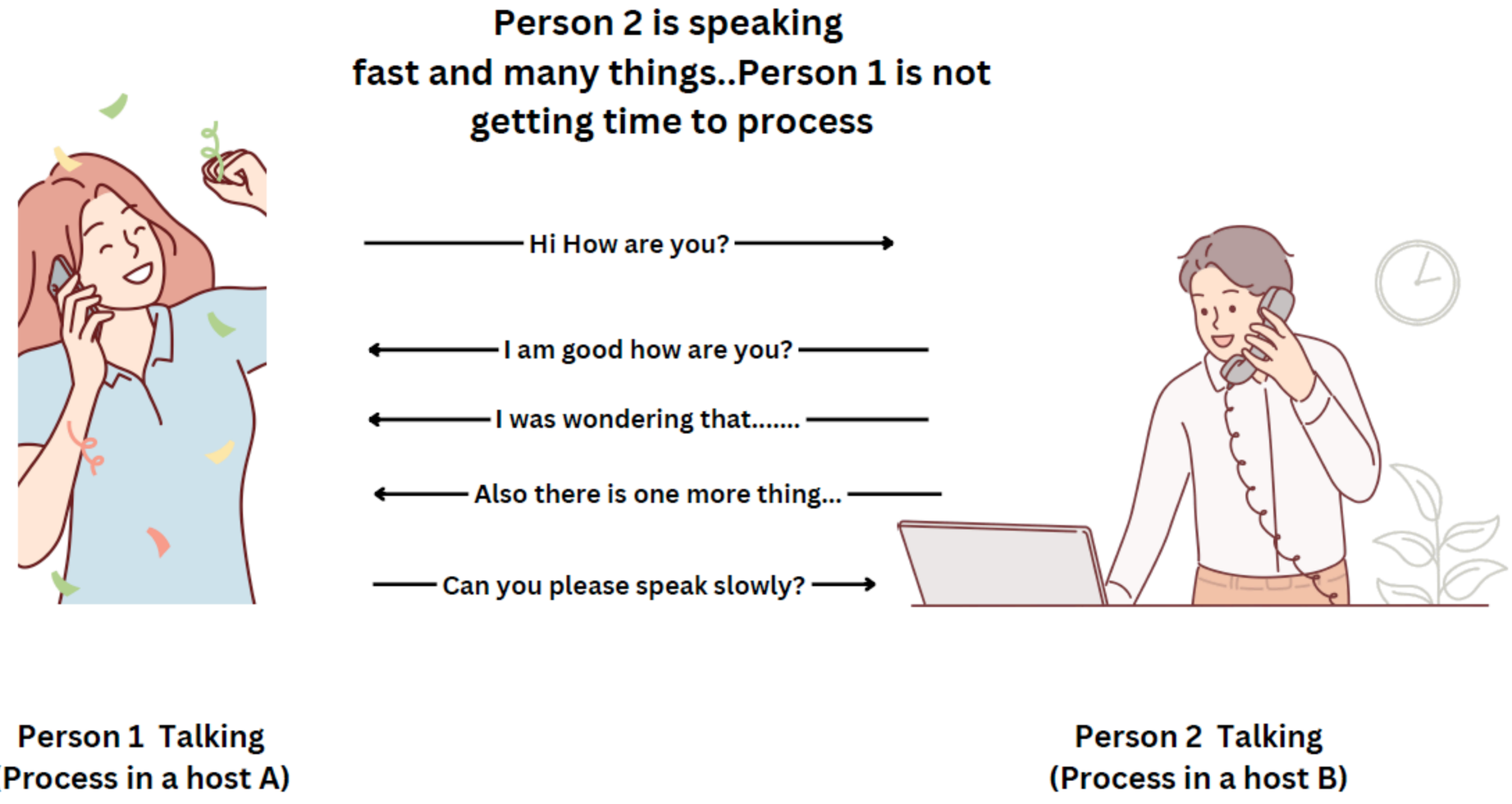
- **SampleRTT:** Time measured from segment transmission until ACK receipt
- **EstimatedRTT:** Estimated weighted moving average (EWMA) $\alpha = 0.25$
- **DevRTT:** EWMA of sampleRTT deviation from EstimatedRTT $\beta = 0.75$
- **TimeoutInterval:** Estimated Time plus some kind of safety margin

Do We need to Send ACK for each segment?

Use delayed acknowledgements

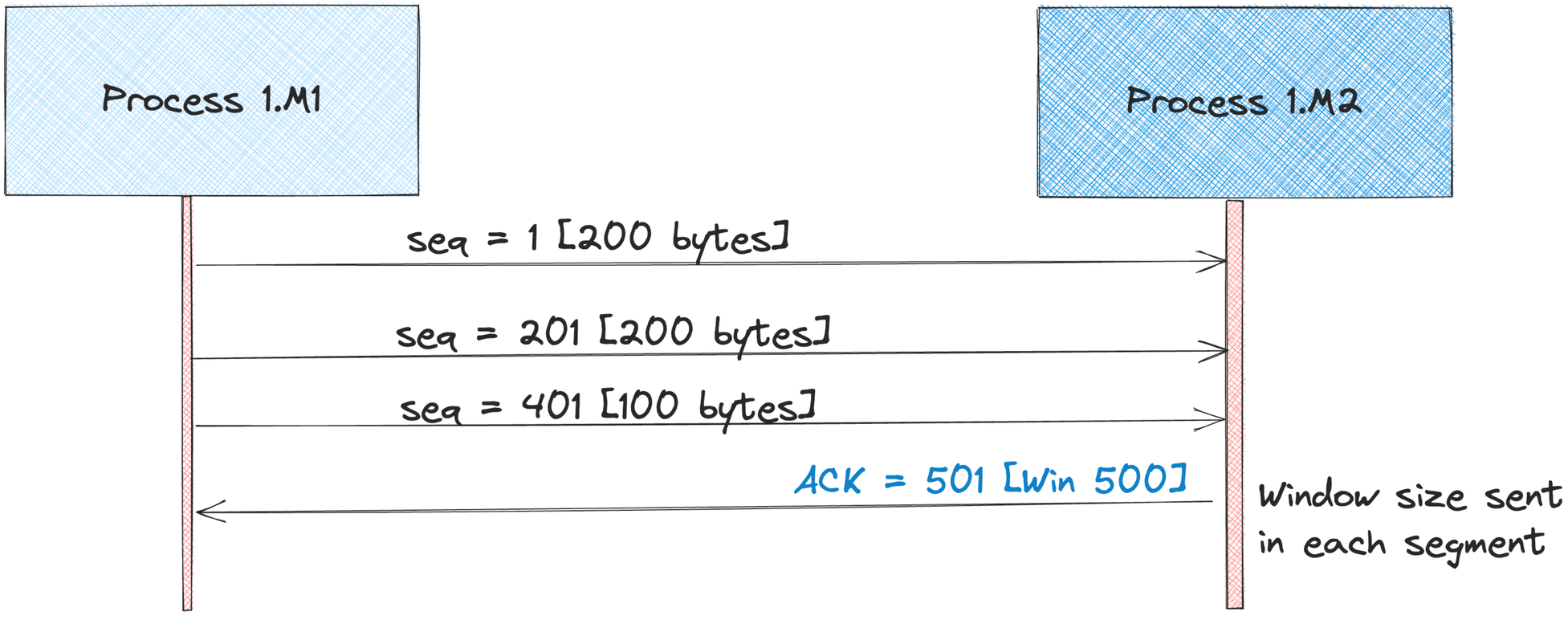


What if the speed is high?



Sending too much data is also problem

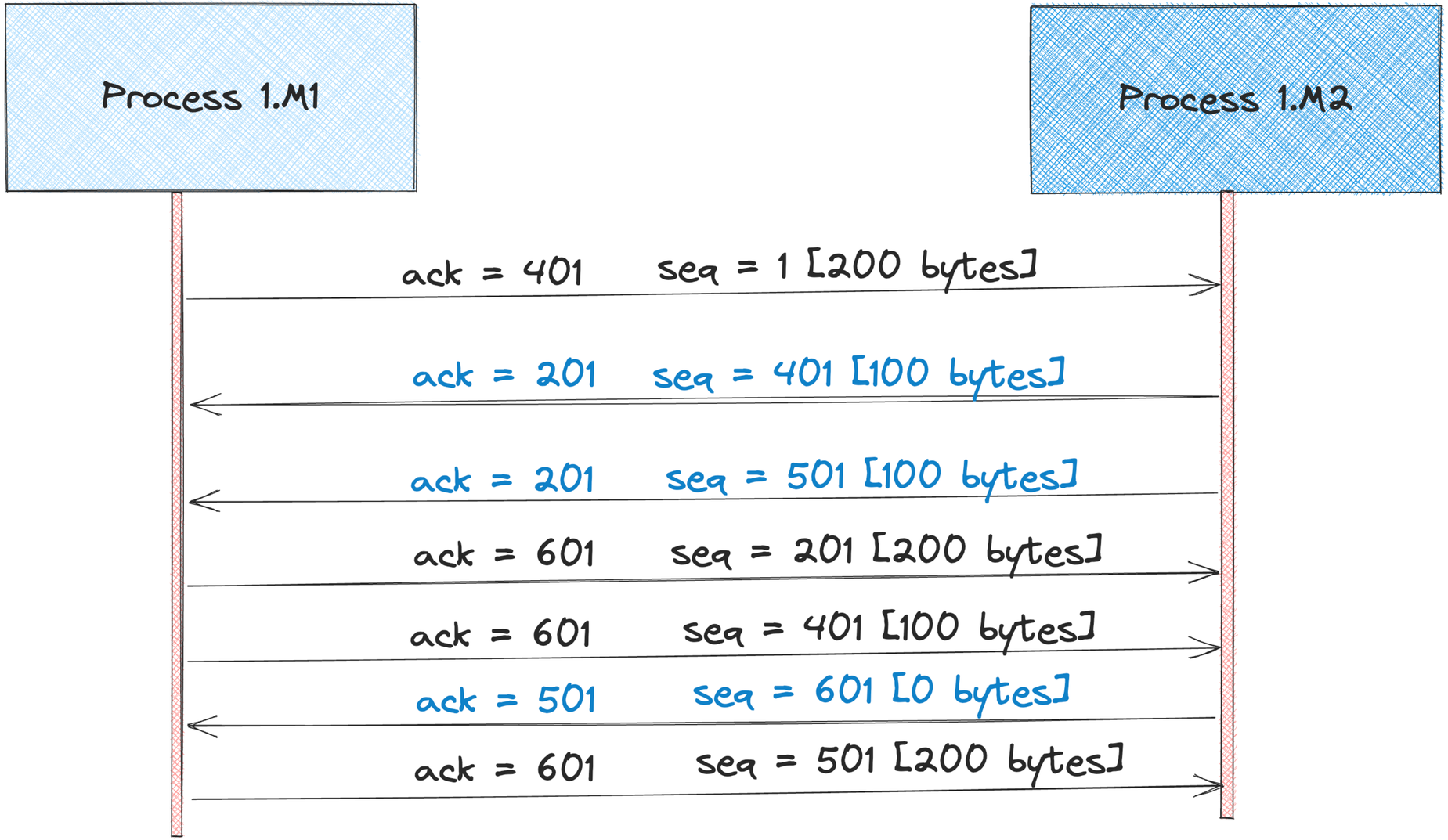
Window Size - Flow Control



- 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

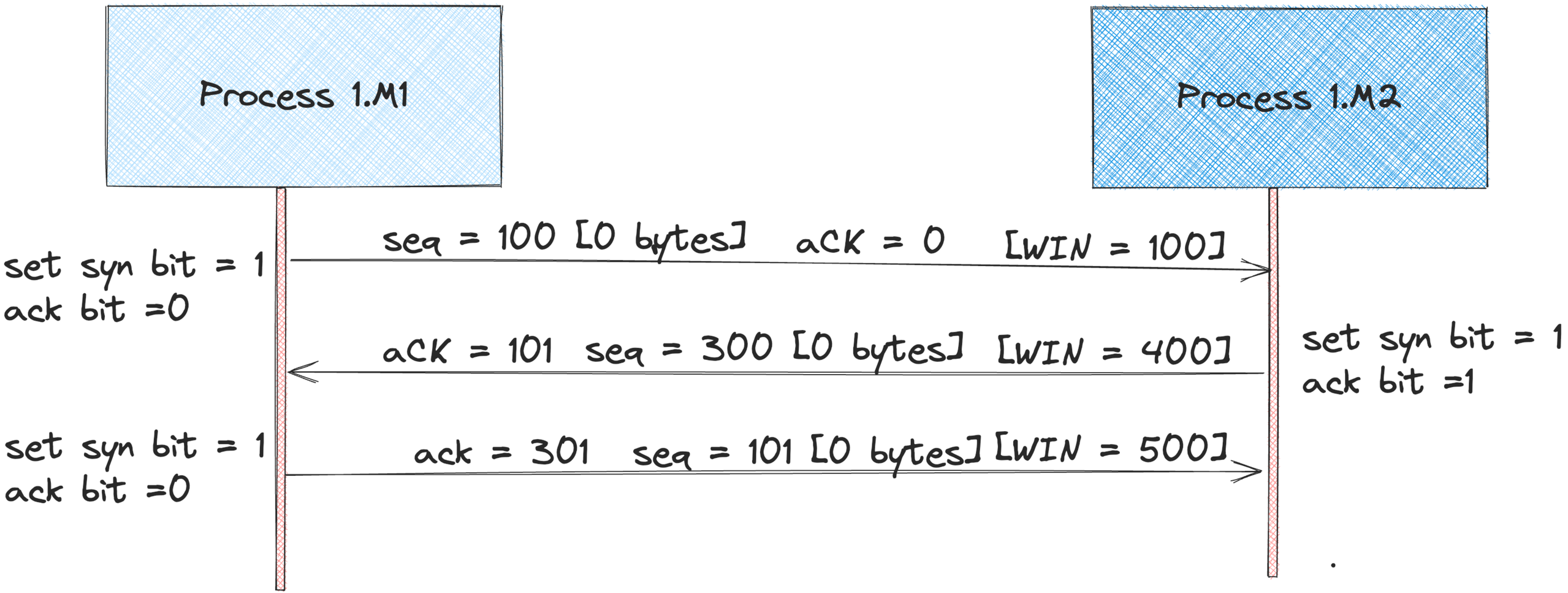


How to choose sequential numbers?

- 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]

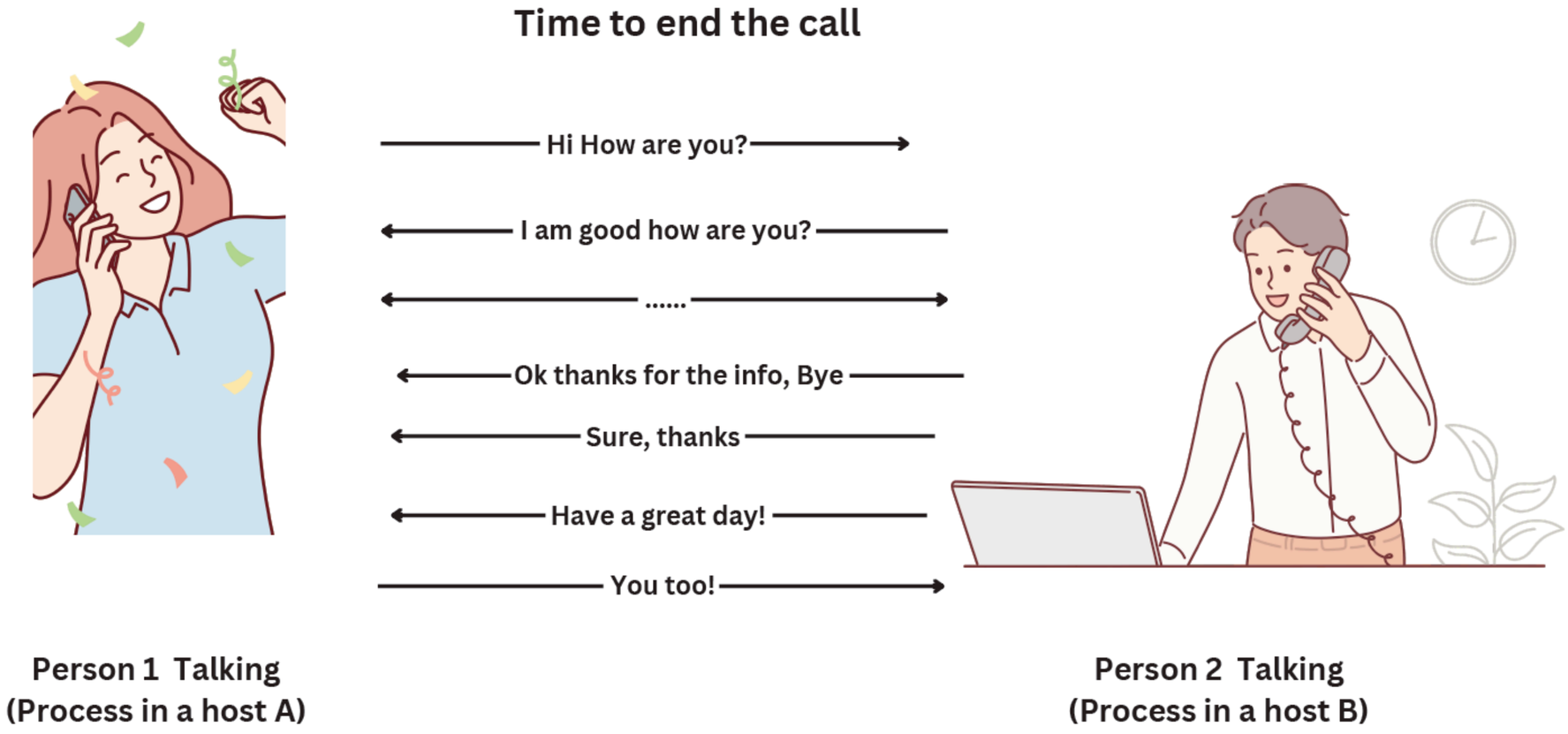
Three Way Handshake

Establishing Connection



Software Engineering Research Centre

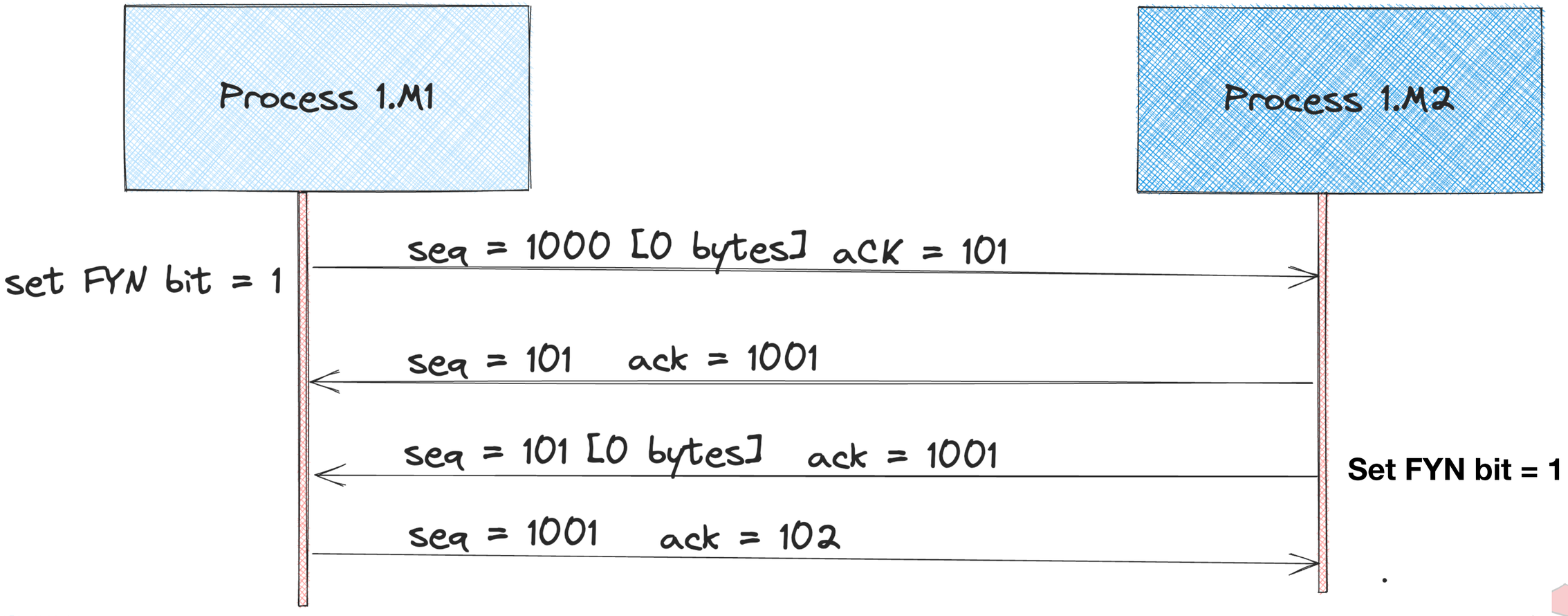
Closing Connection



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

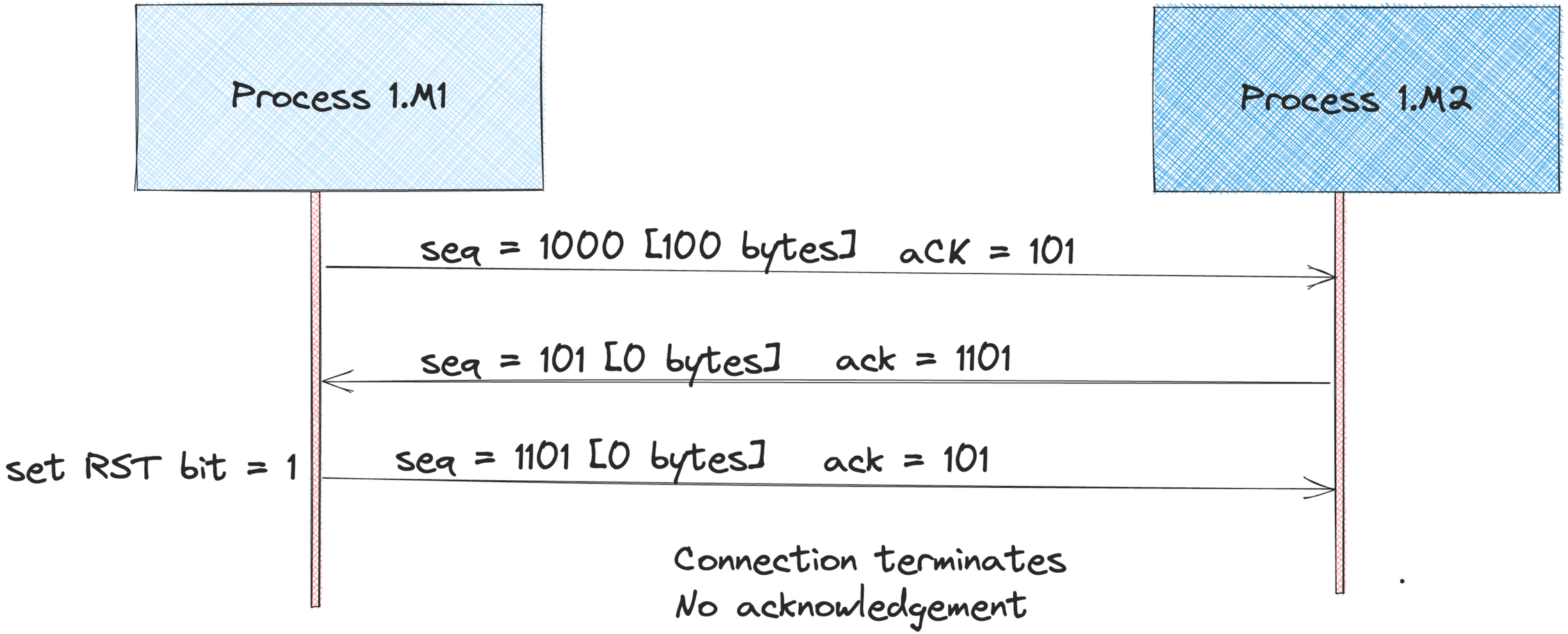
Using FYN bit

Graceful termination (Four-way closure)



Using RST Flags

Ungraceful closing



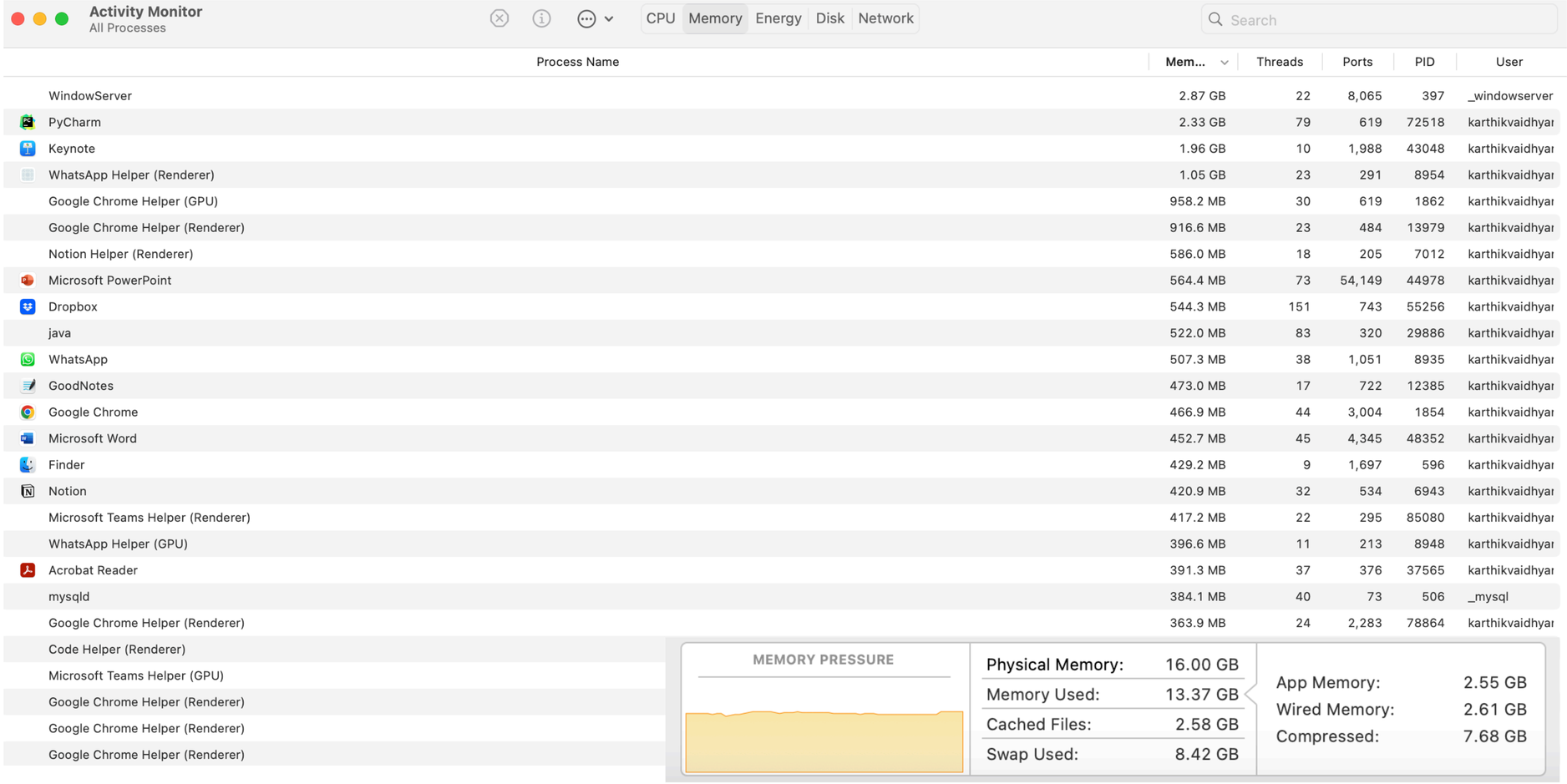
But we need Memory!

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!

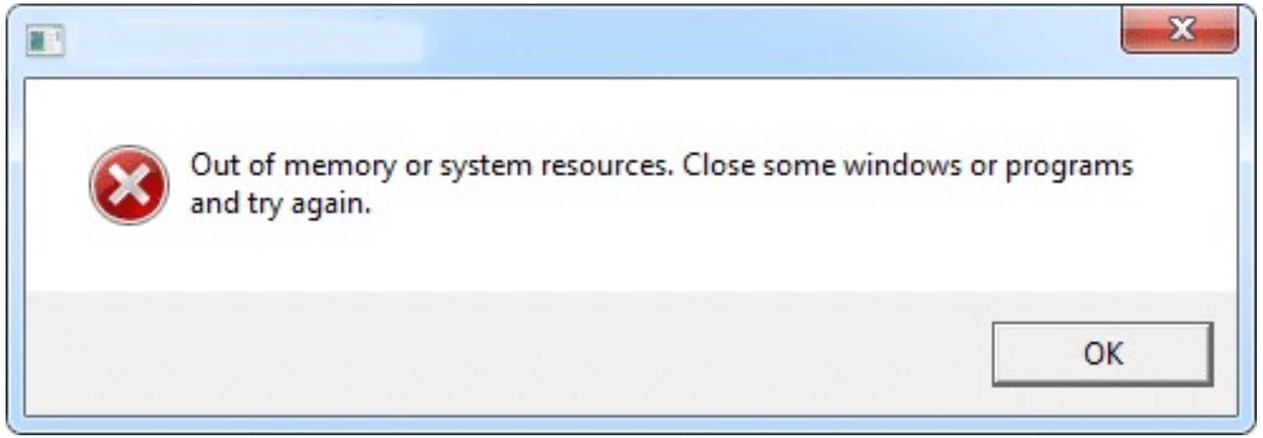
- What about Memory? Do we have enough Memory?

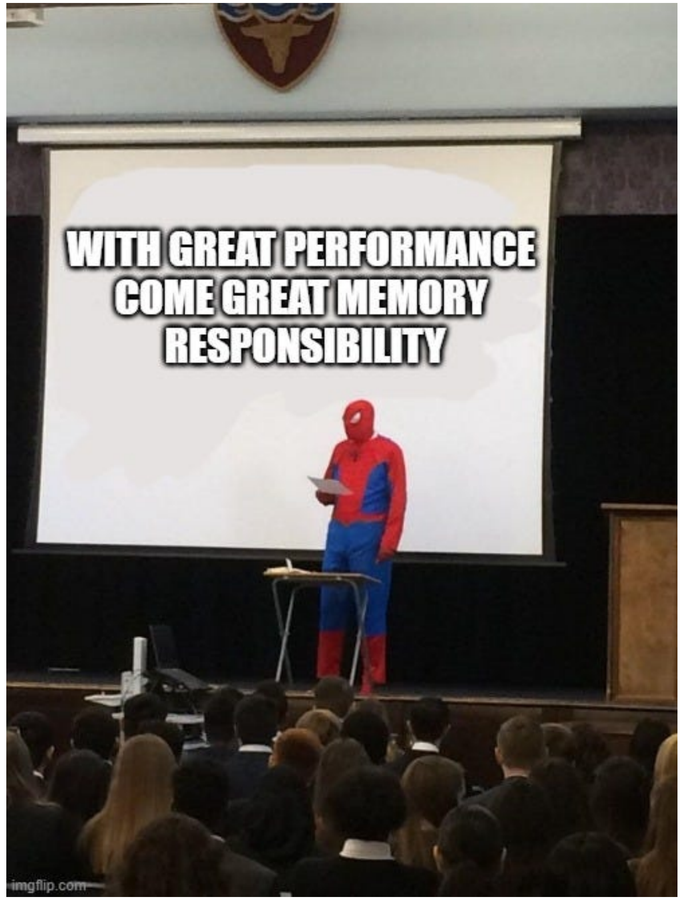


Process Name	Mem...	Threads	Ports	PID	User
WindowServer	2.87 GB	22	8,065	397	_windowserver
PyCharm	2.33 GB	79	619	72518	karthikvaidhyar
Keynote	1.96 GB	10	1,988	43048	karthikvaidhyar
WhatsApp Helper (Renderer)	1.05 GB	23	291	8954	karthikvaidhyar
Google Chrome Helper (GPU)	958.2 MB	30	619	1862	karthikvaidhyar
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

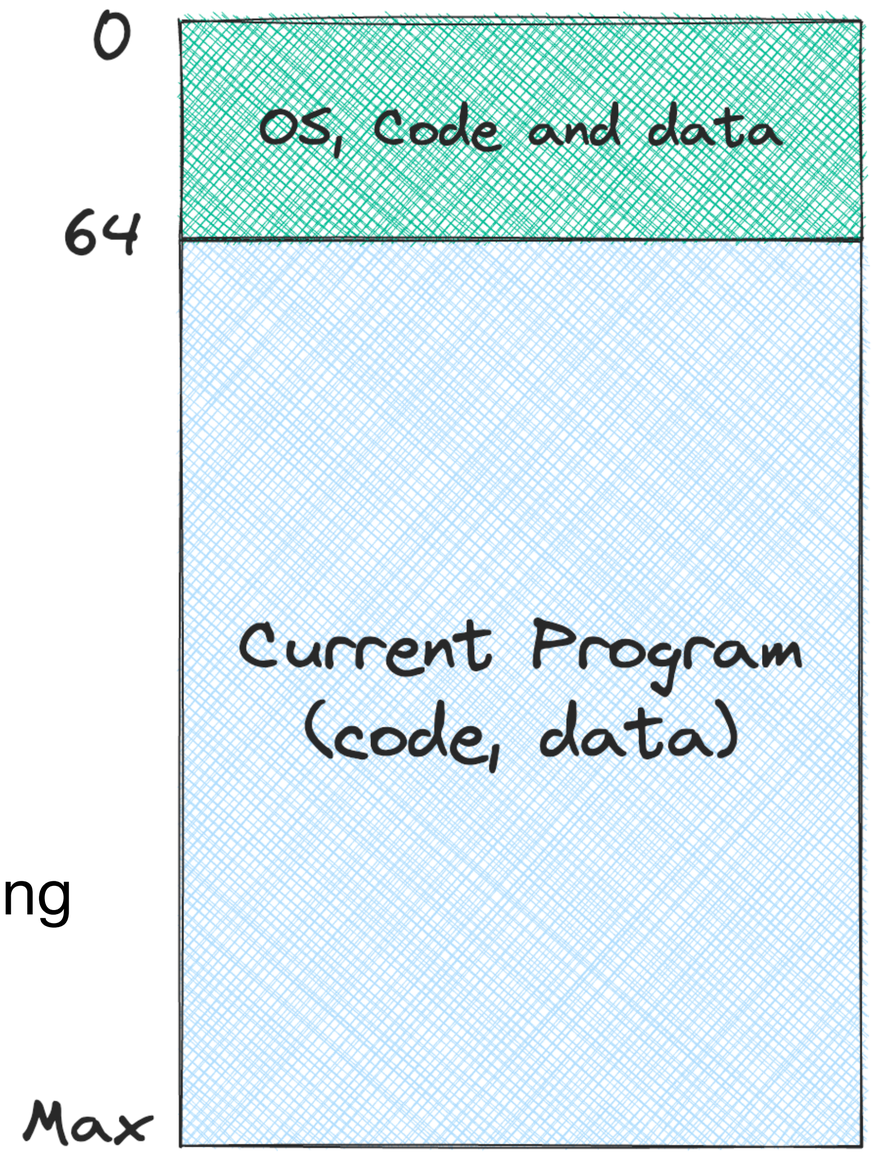




Source: Google Images

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?

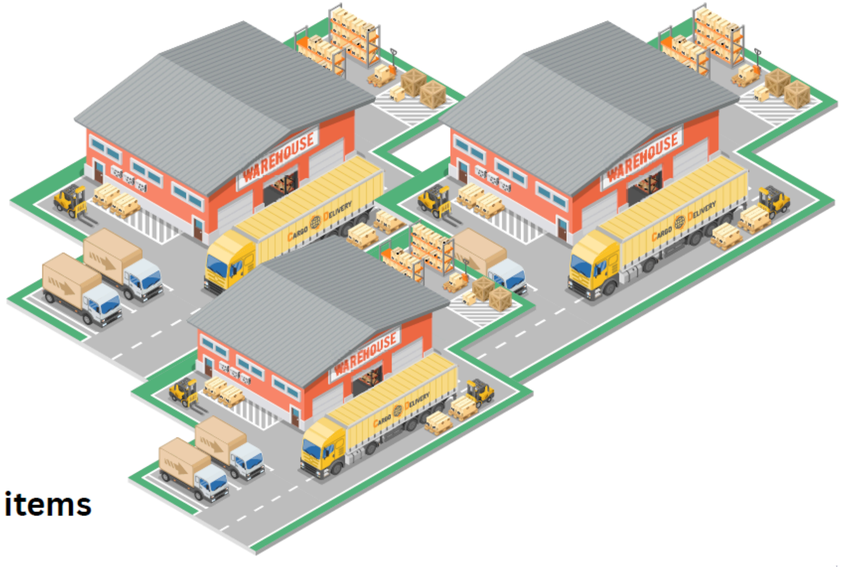
- We need to think about multiple processes
- Need to increase utilisation and efficiency
- Particularly useful in olden times when it costed millions of dollars for machines
- Soon came era of time sharing
- Batch computing was not anymore appreciated
- Instead of saving in the disk, can we keep the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

An Analogy

Onsite Shopping

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

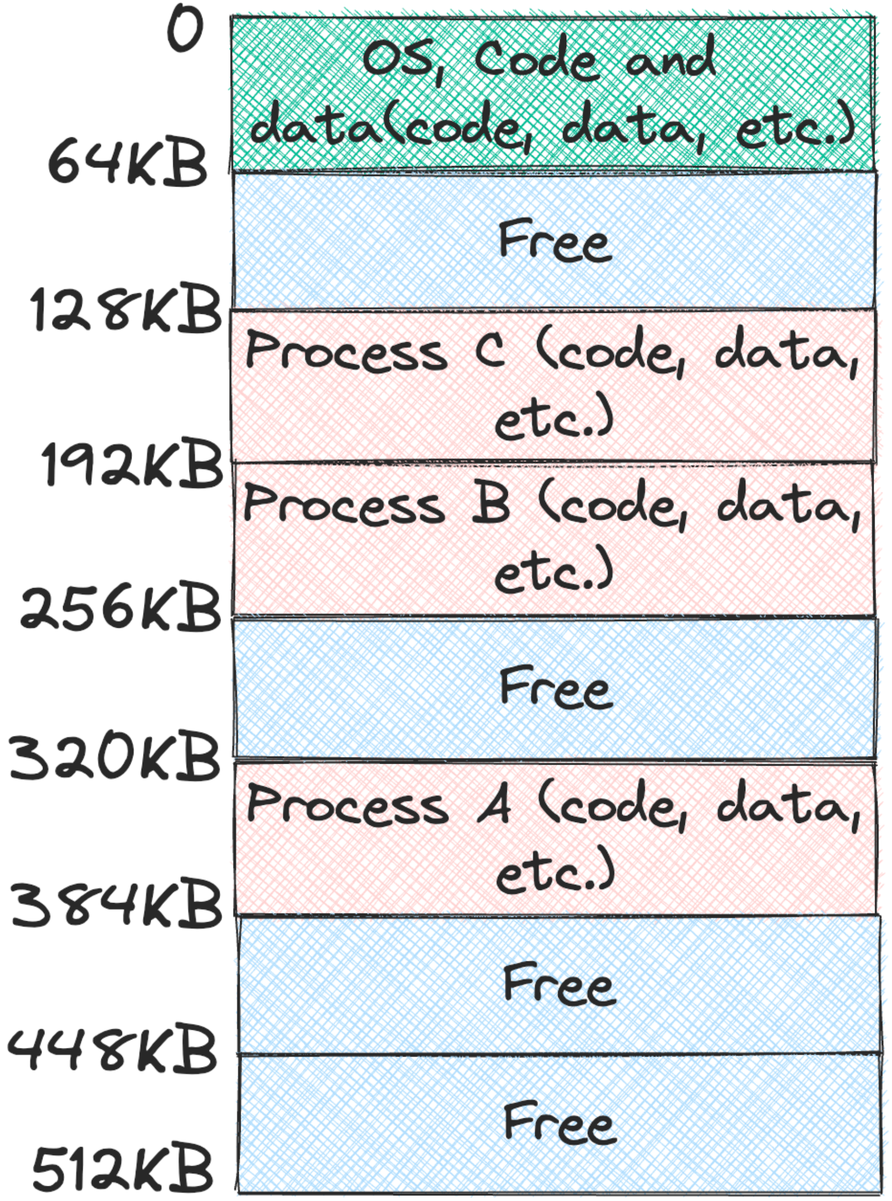
Online Shopping



Every Users feel that they have access to infinite items

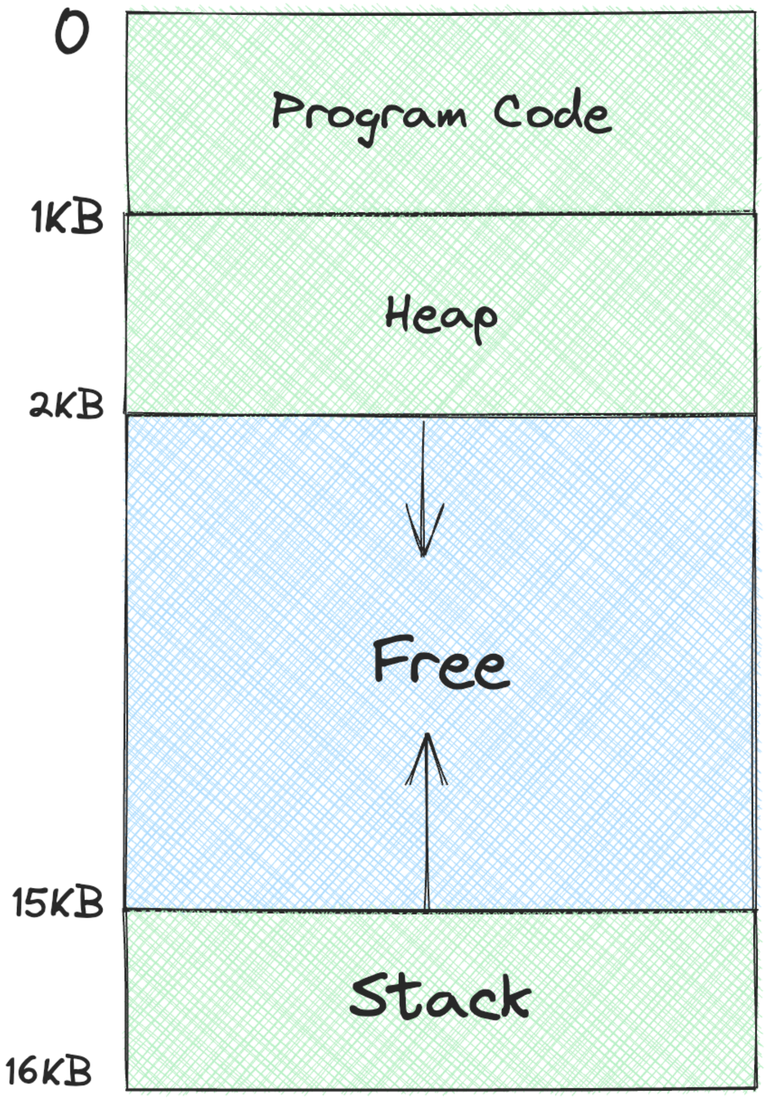
Keep Process in the Memory

- 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

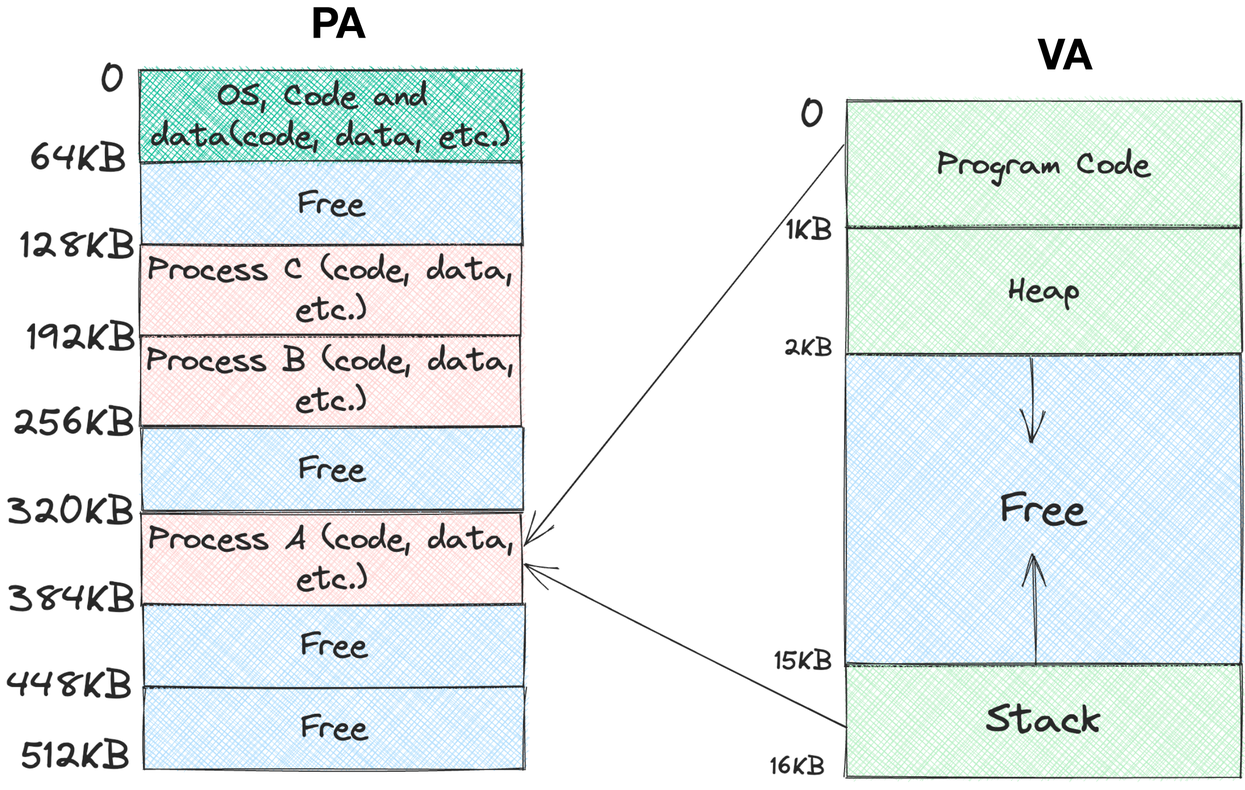


There is only one physical memory

- 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

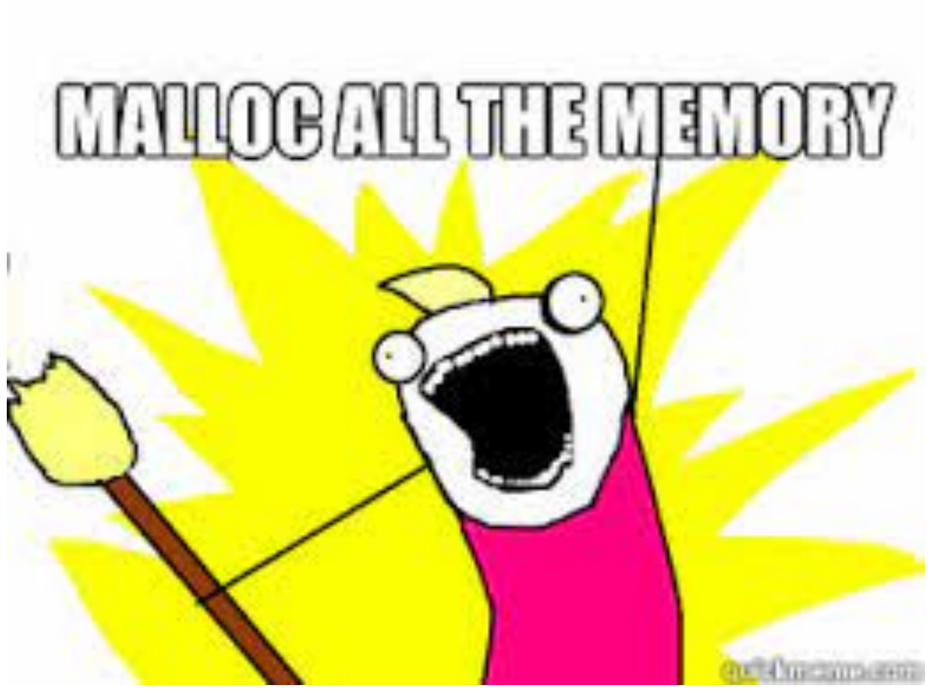
Second type of Memory Allocation

- 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

```c
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));
```

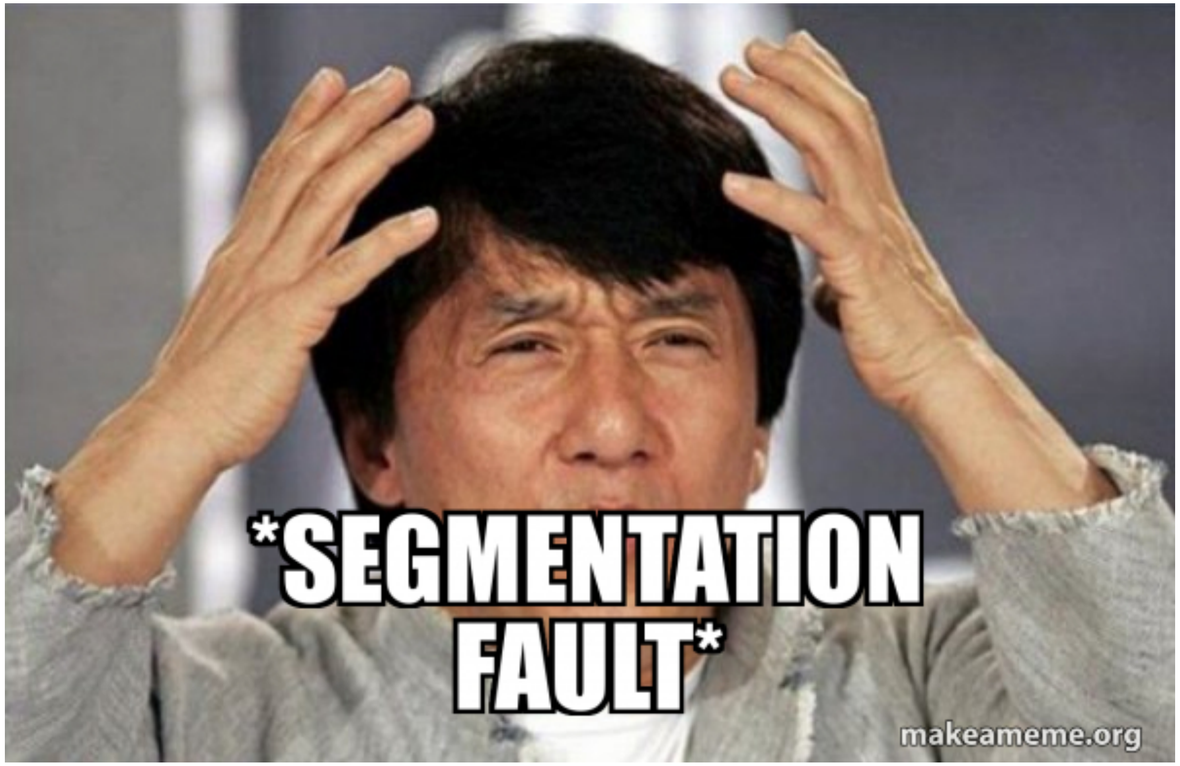
Free() call

- 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

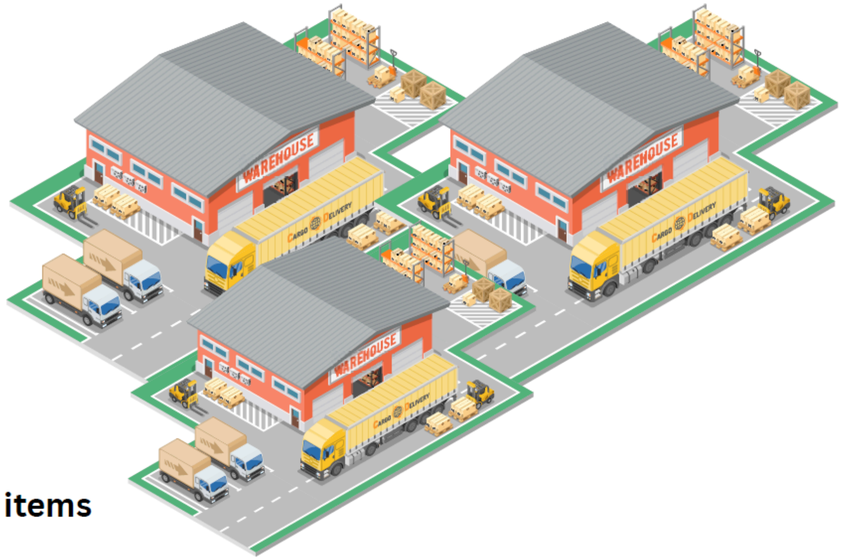
- **Goal:** Create an illusion that each process has its own private memory where the code and data reside
  - Reality: Many processes are actually sharing memory at the same time!
- How to make this happen? - Three Key assumptions:
  - User address space must be placed contiguously in physical memory
  - 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

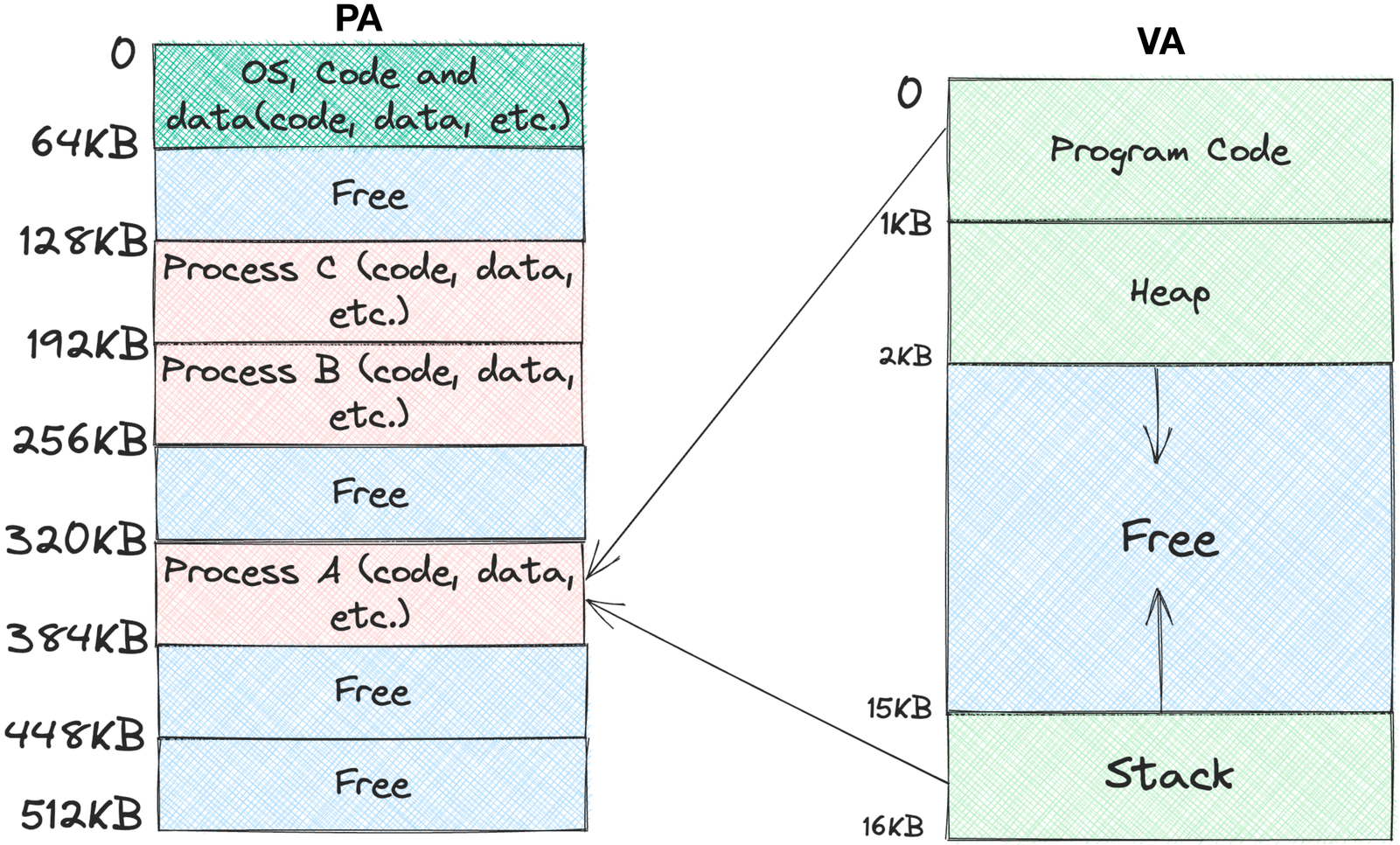
Every users have access to different items but to a limited set

Online Shopping

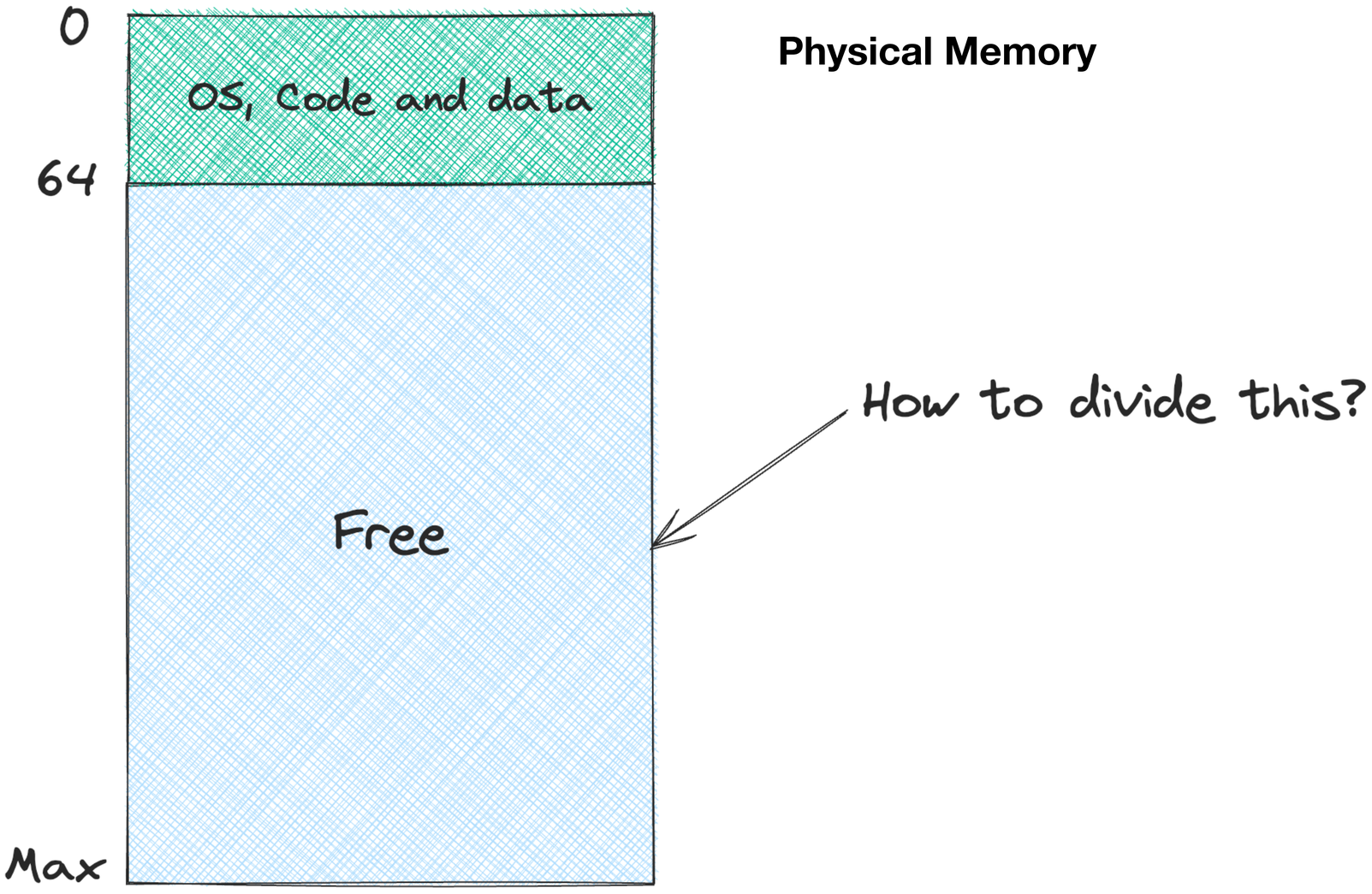


Every Users feel that they have access to infinite items

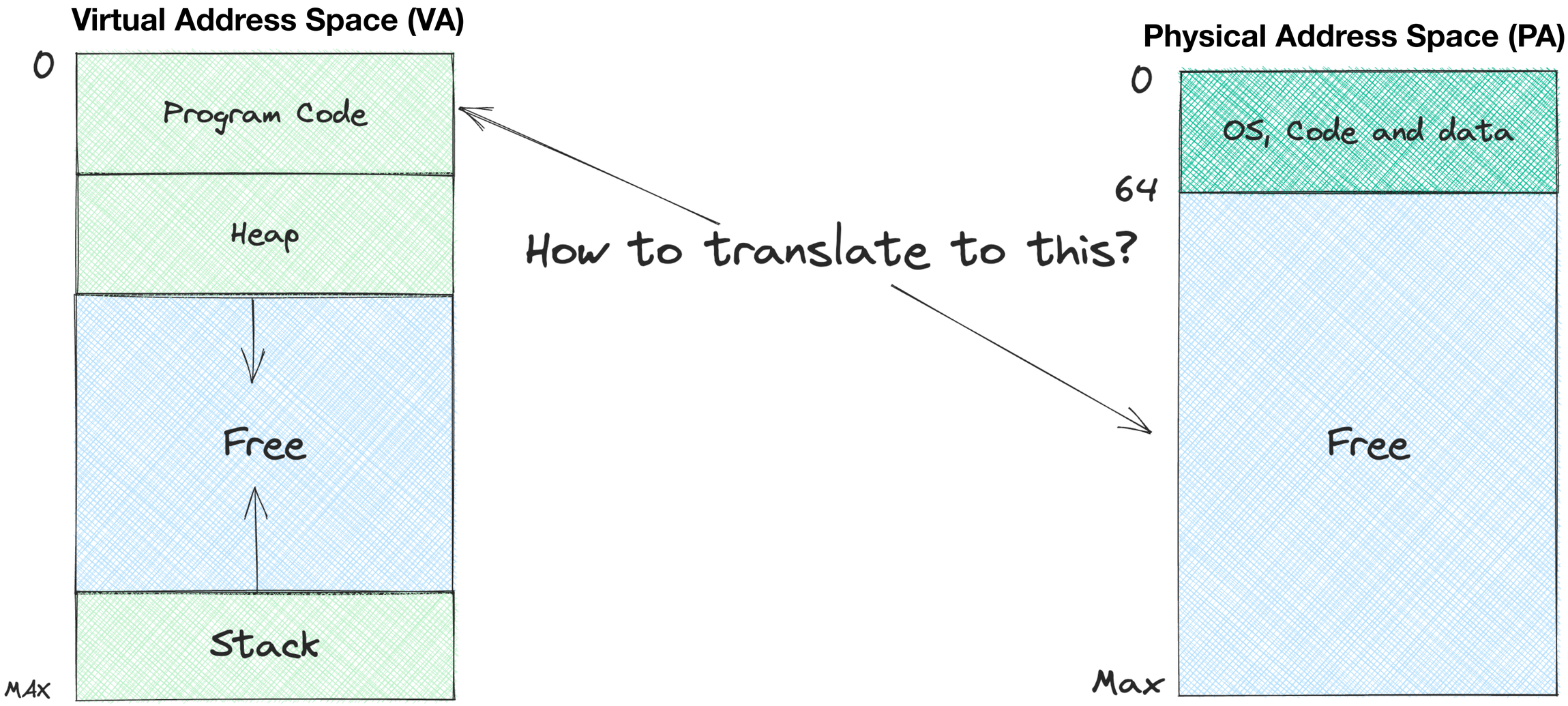
Address Translation — Recap



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;
}
```

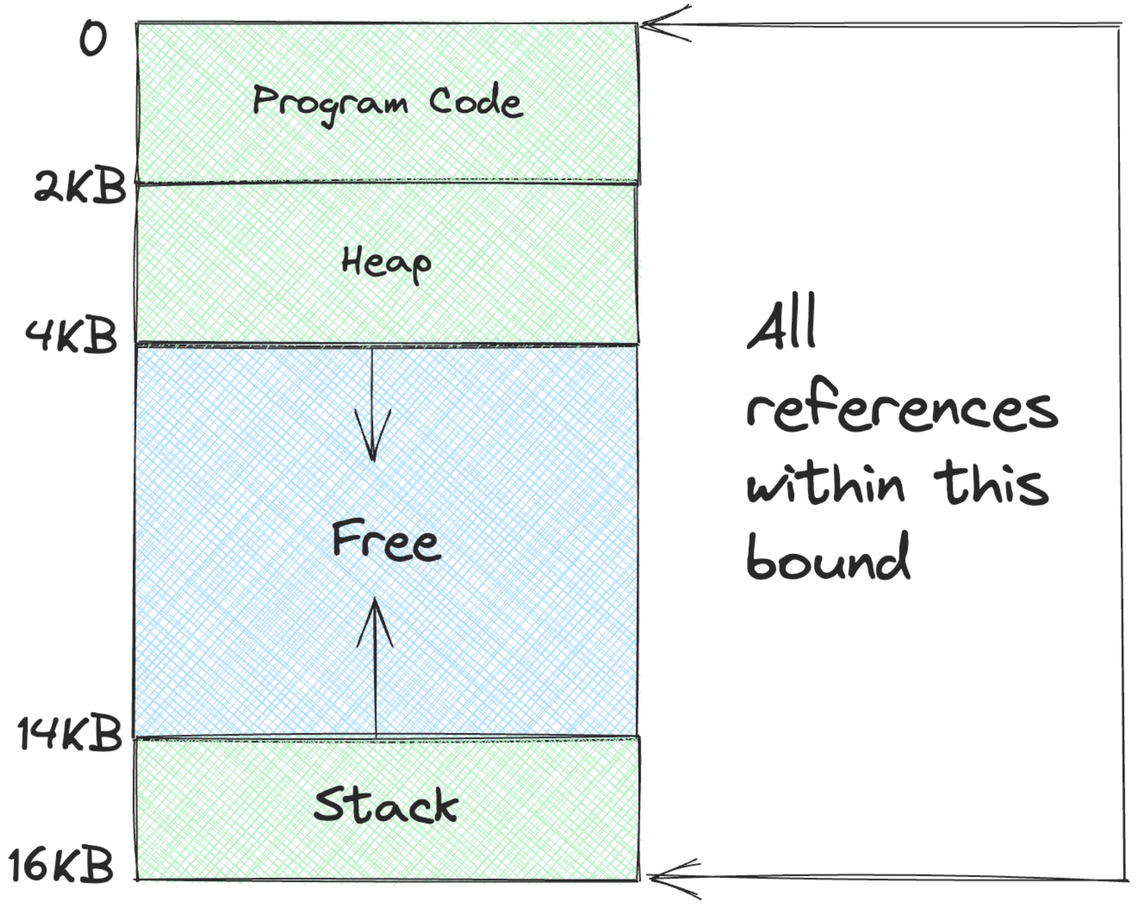
Assembly Code

```
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
```

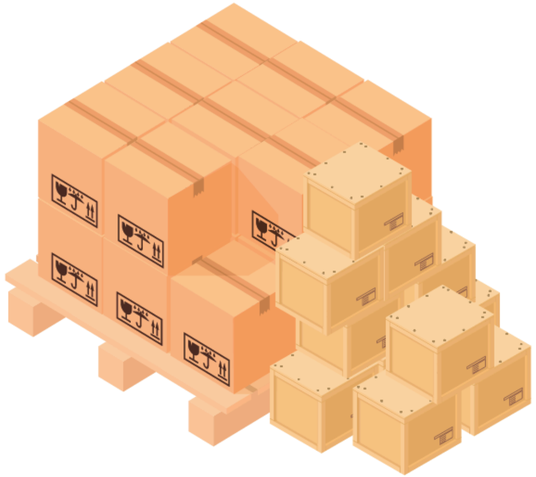
int x = 3000;

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

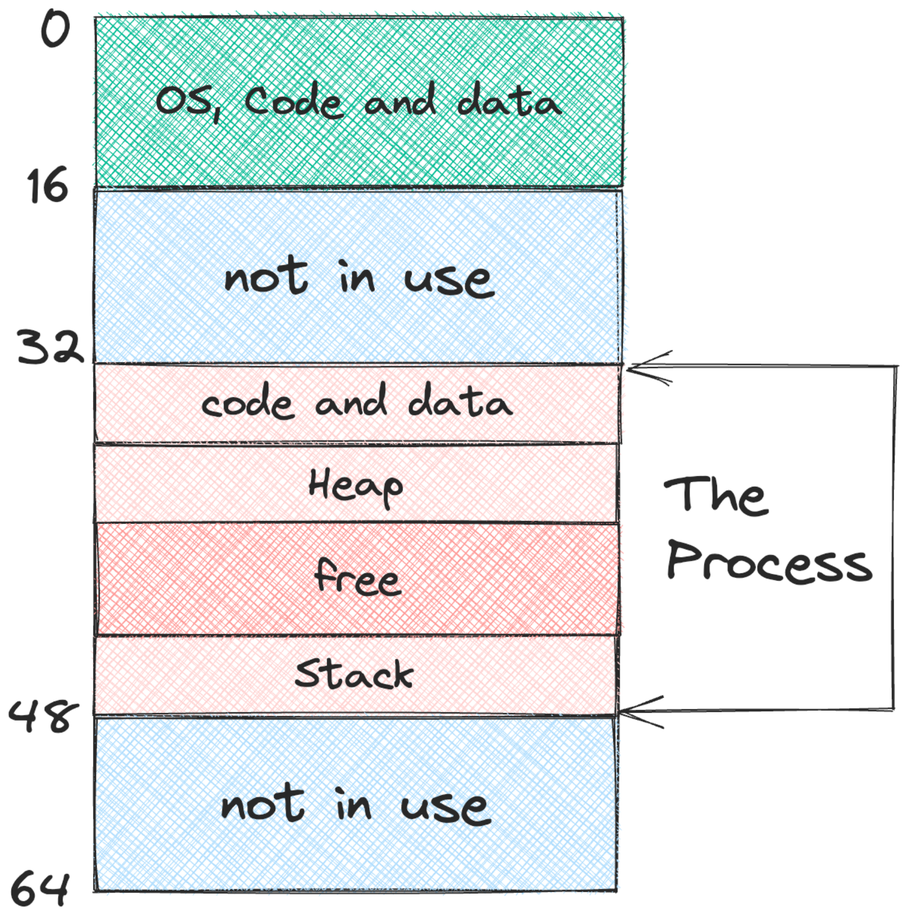
Based on a Category: Range can be decided
Category like Electronics, Clothing, etc

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?

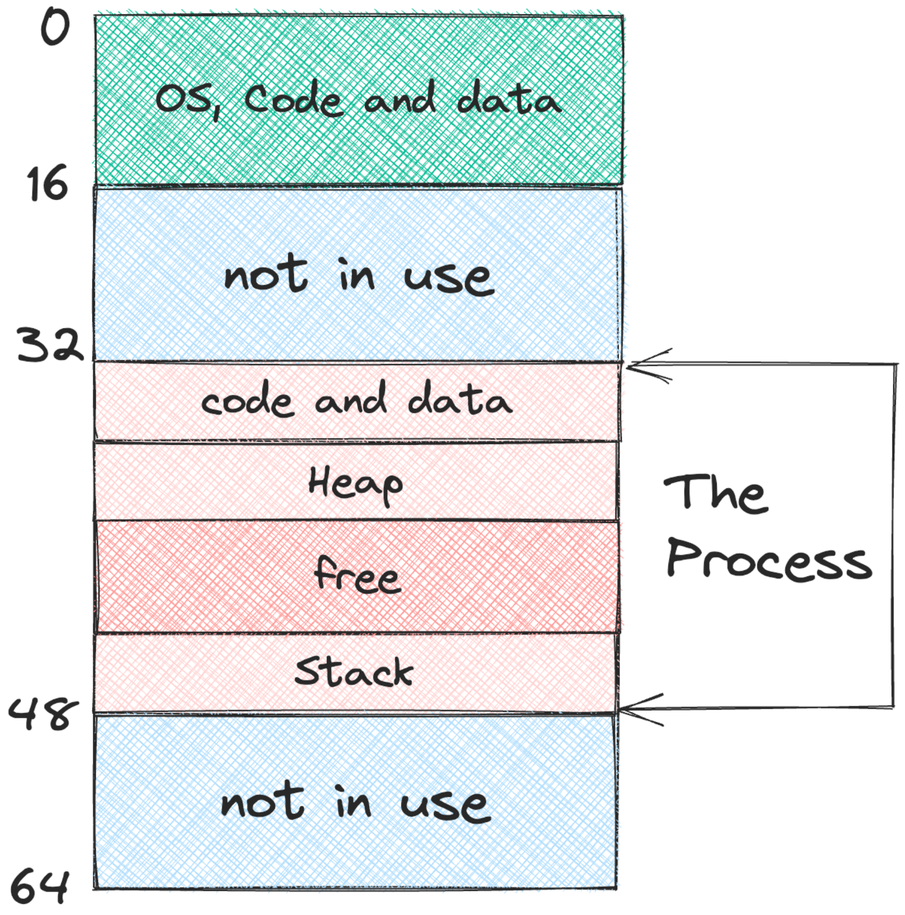
- To virtualize, OS cannot place the process starting from 0.
- The process requires same amount of space as in Virtual address space but somewhere else.
- The reality of physical memory is different from what the process sees!
- The process of translation just needs to map the two
- Can you think of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

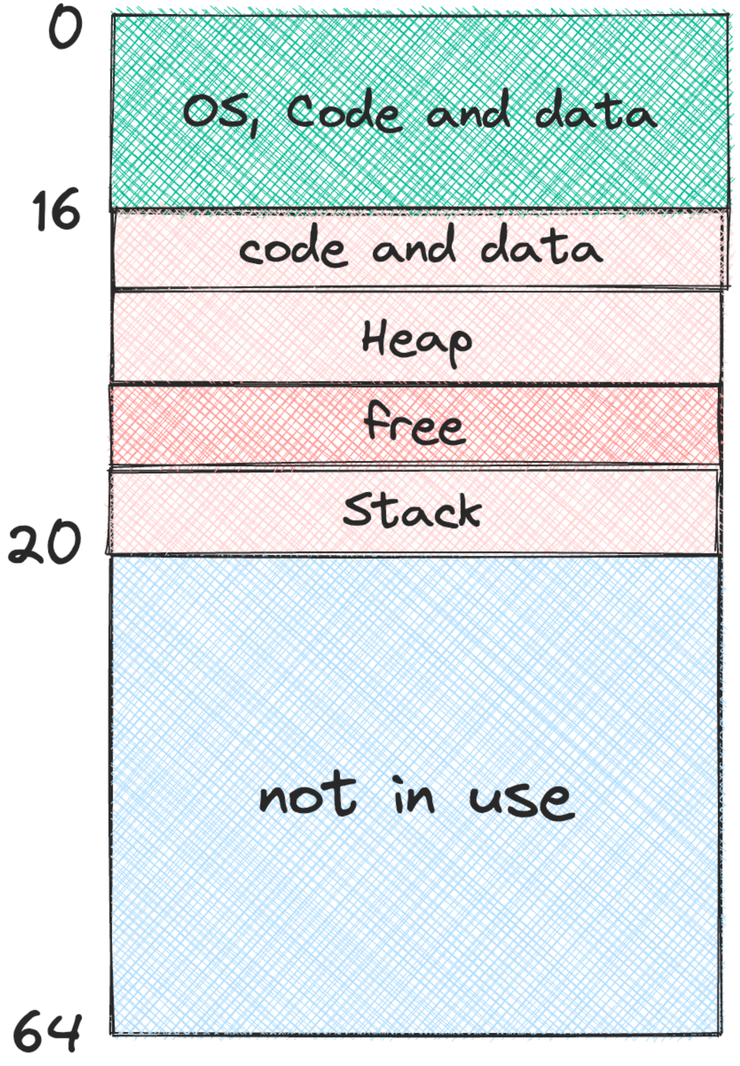
Dynamic Relocation

The Base and Bounds Approach

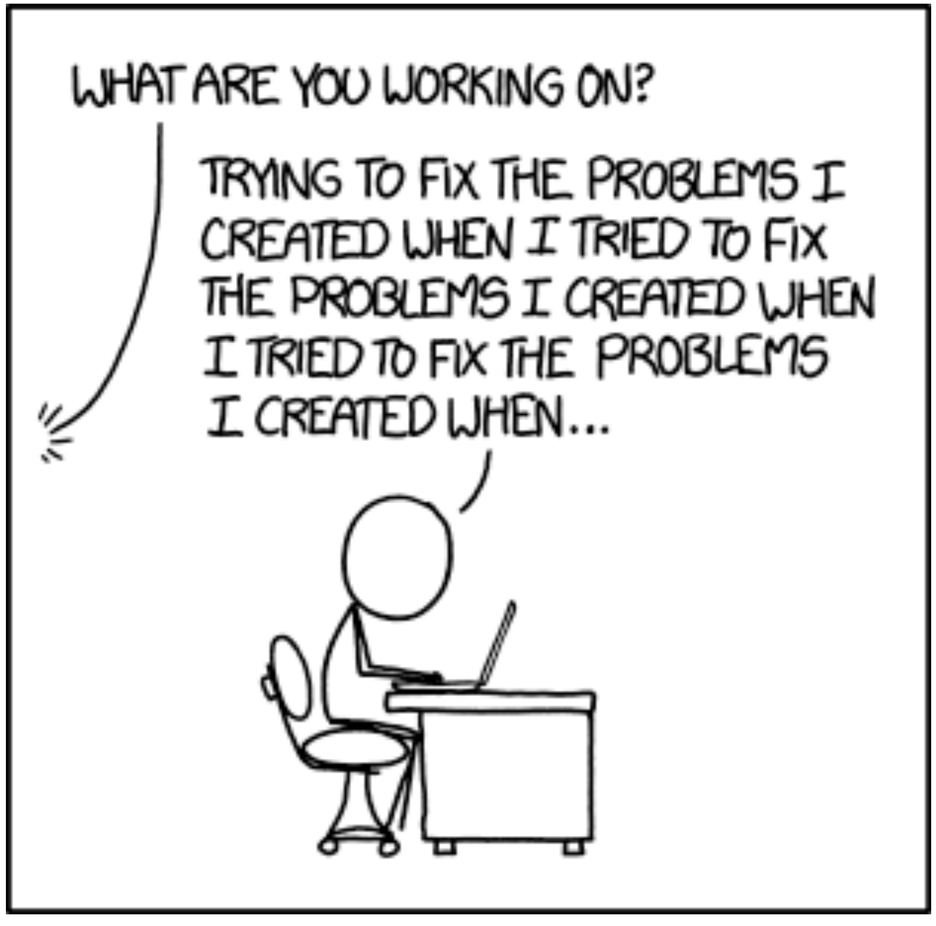
- **Bounds** register ensures that any memory reference is within bounds
  - Everything has to be a legal access
  - If process generates address > bounds (Either relative to VA or PA)
    - CPU raises an exception (Interrupt raised)
    - Process is terminated
- The base and bounds are registers part 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

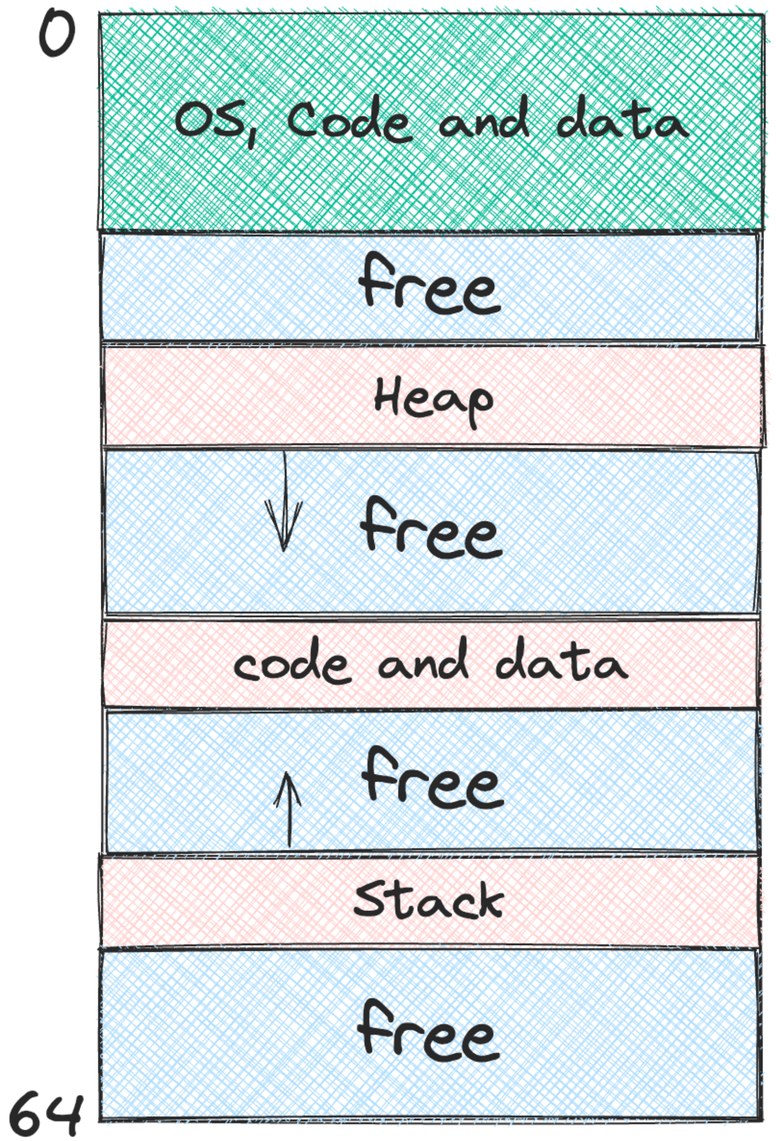
- Instead of having one base and bounds per process
  - Why not have it per logical segment of the address space?
- Segment: Contiguous portion of the address space of a particular length
  - In canonical address space - Three segments
    - Code, Stack and Heap
  - Segmentation basically allows each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

Segmentation

Generalized Base and Bounds

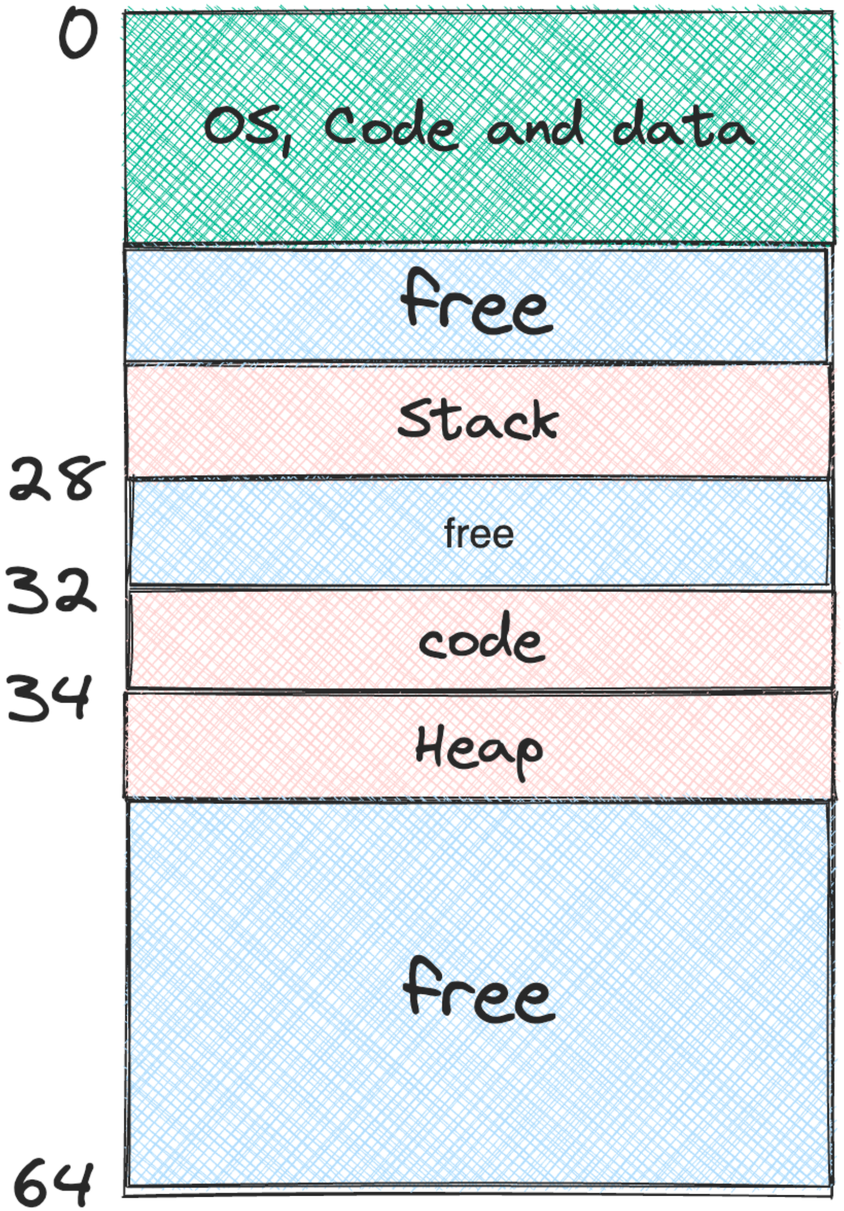
- Only used memory is allocated in physical memory
  - Allows allocating large address space
  - Sparse address space

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



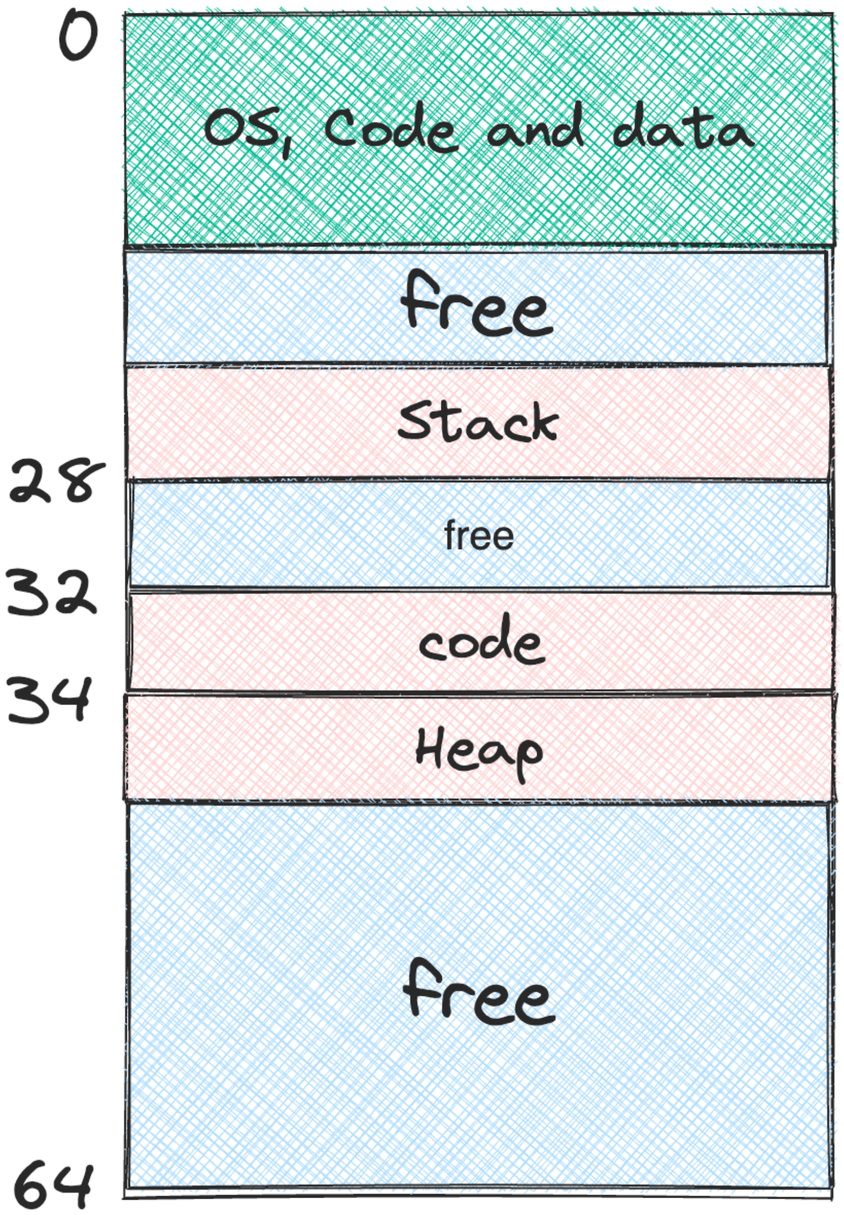
Hardware support (Registers)

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



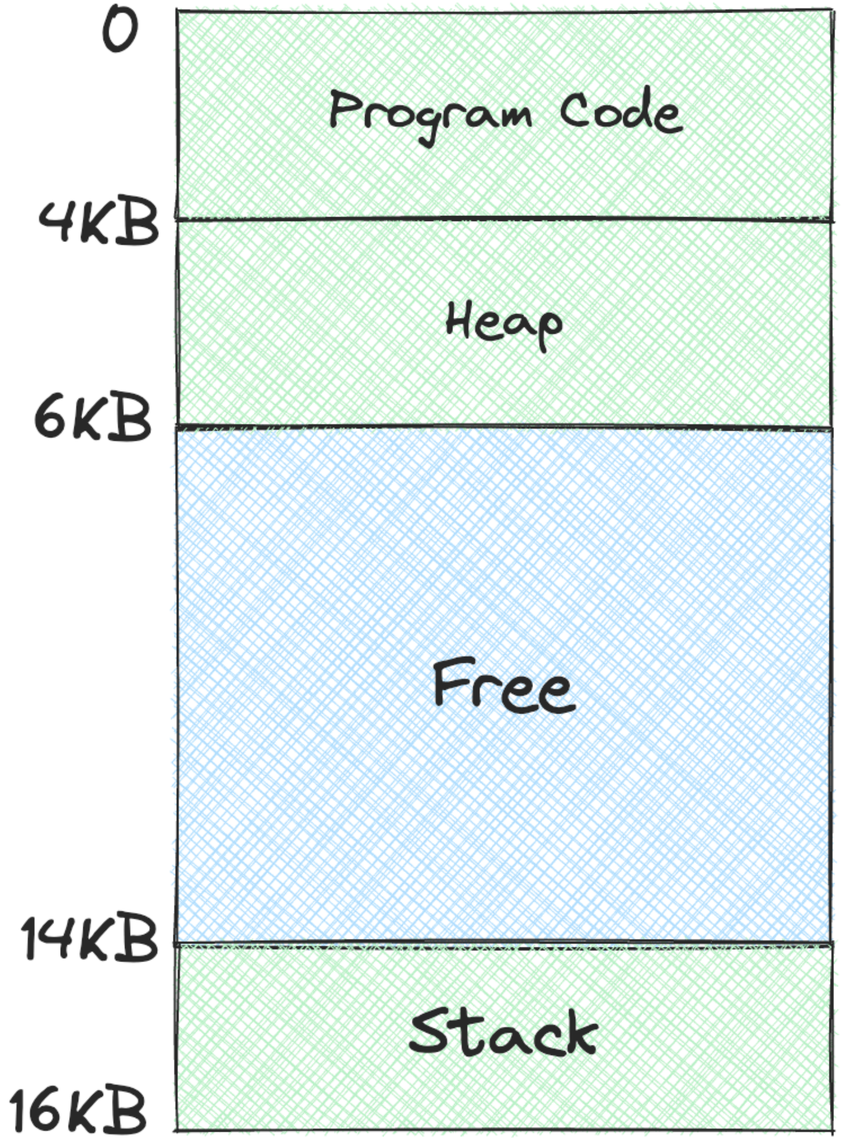
Simple Address Translation

- 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

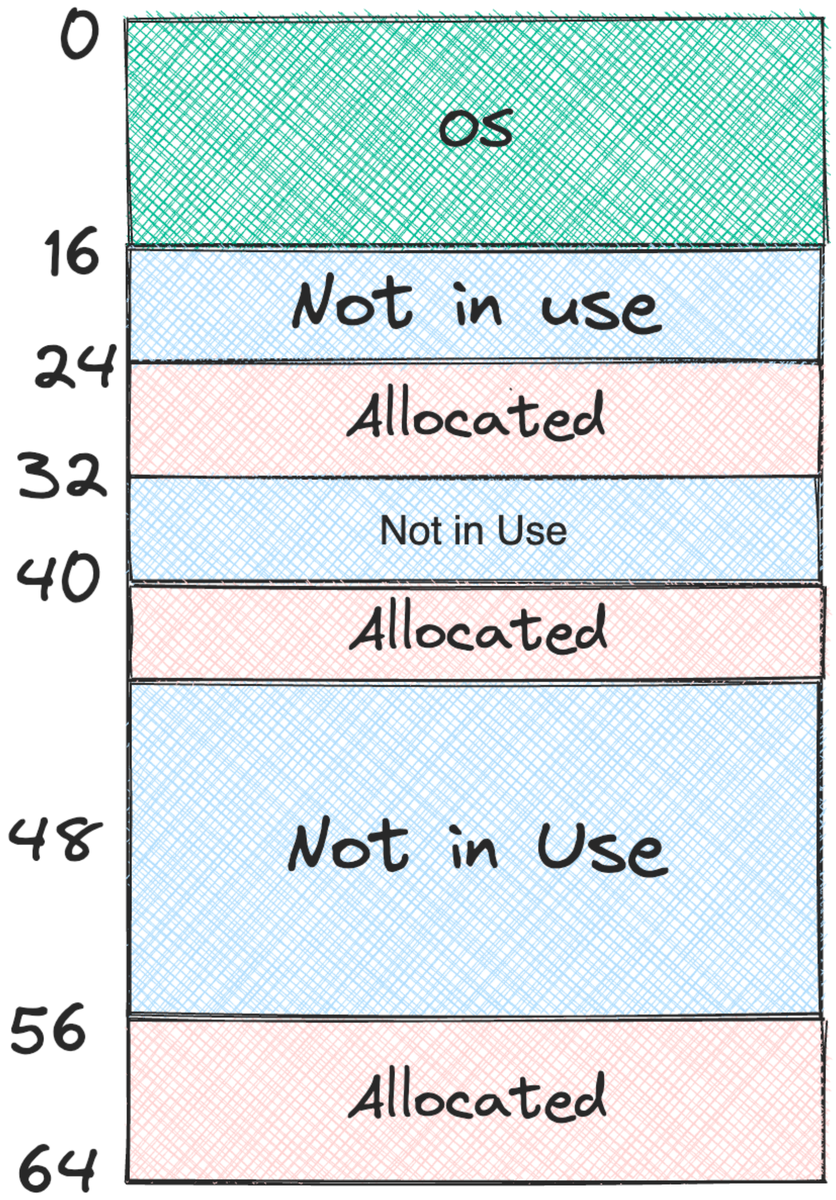
Some Challenges/Issues

- 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

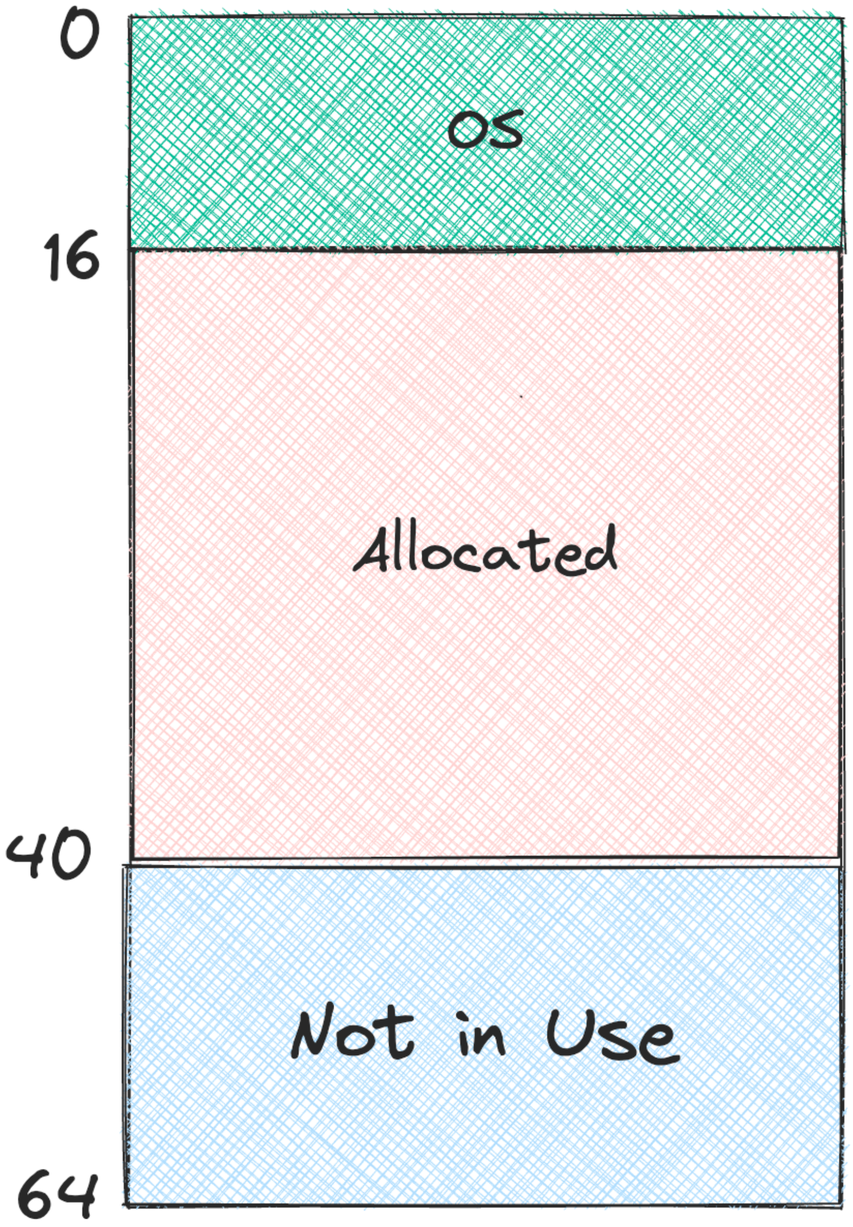
External Fragmentation

- 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

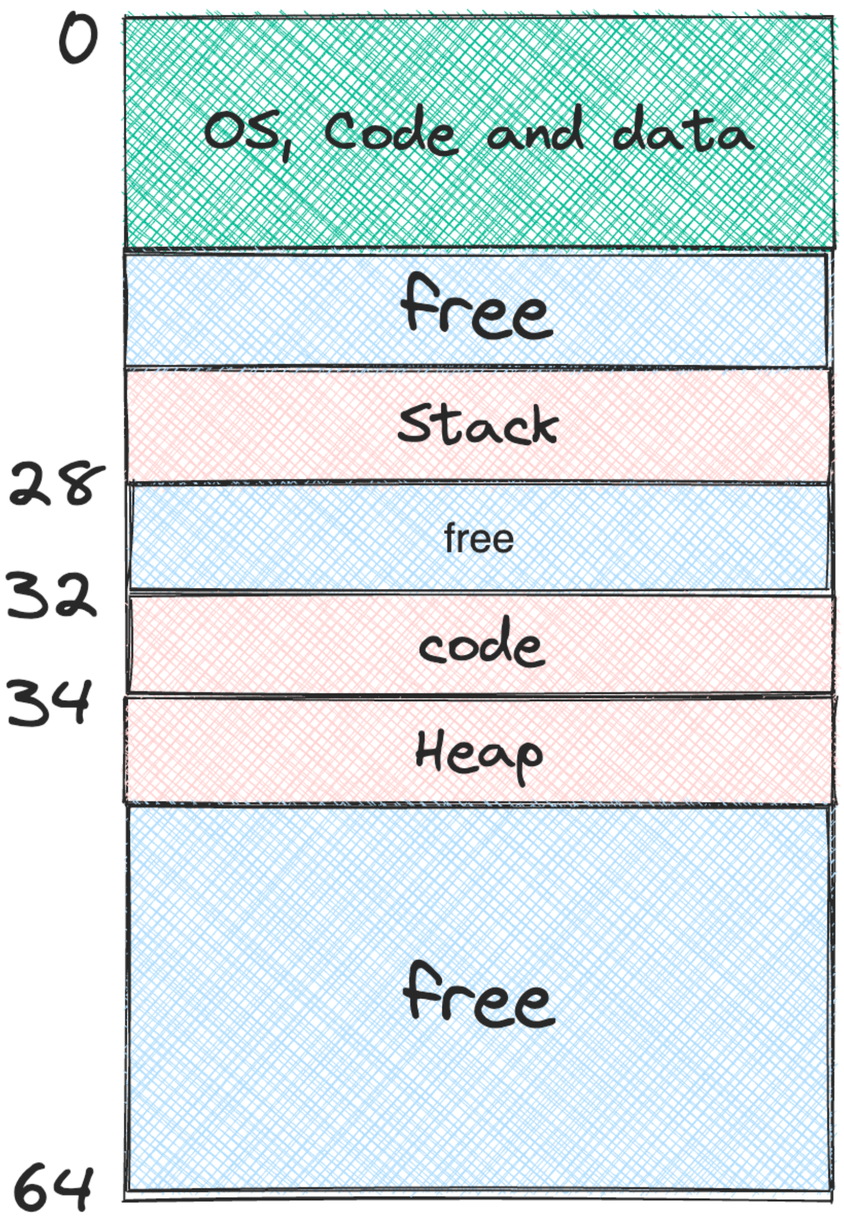
Some Challenges/Issues

- 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

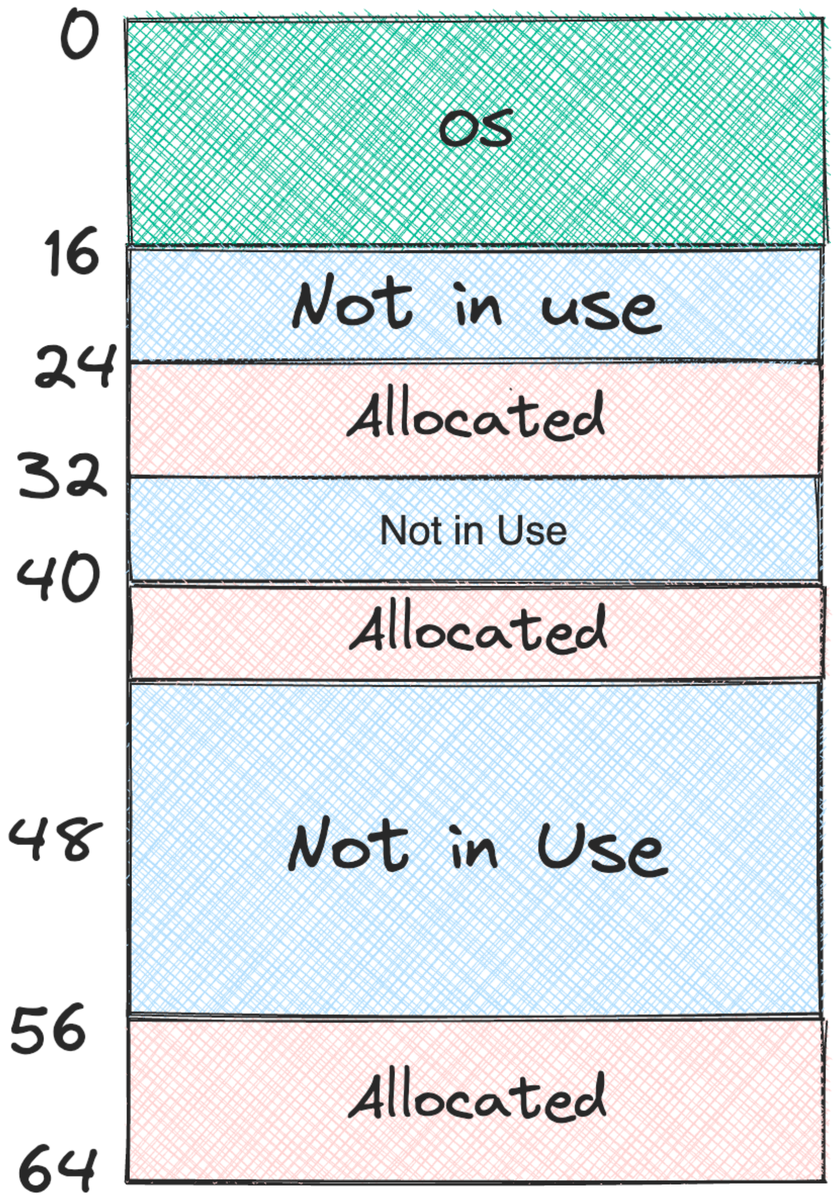
Hardware support (Registers)

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



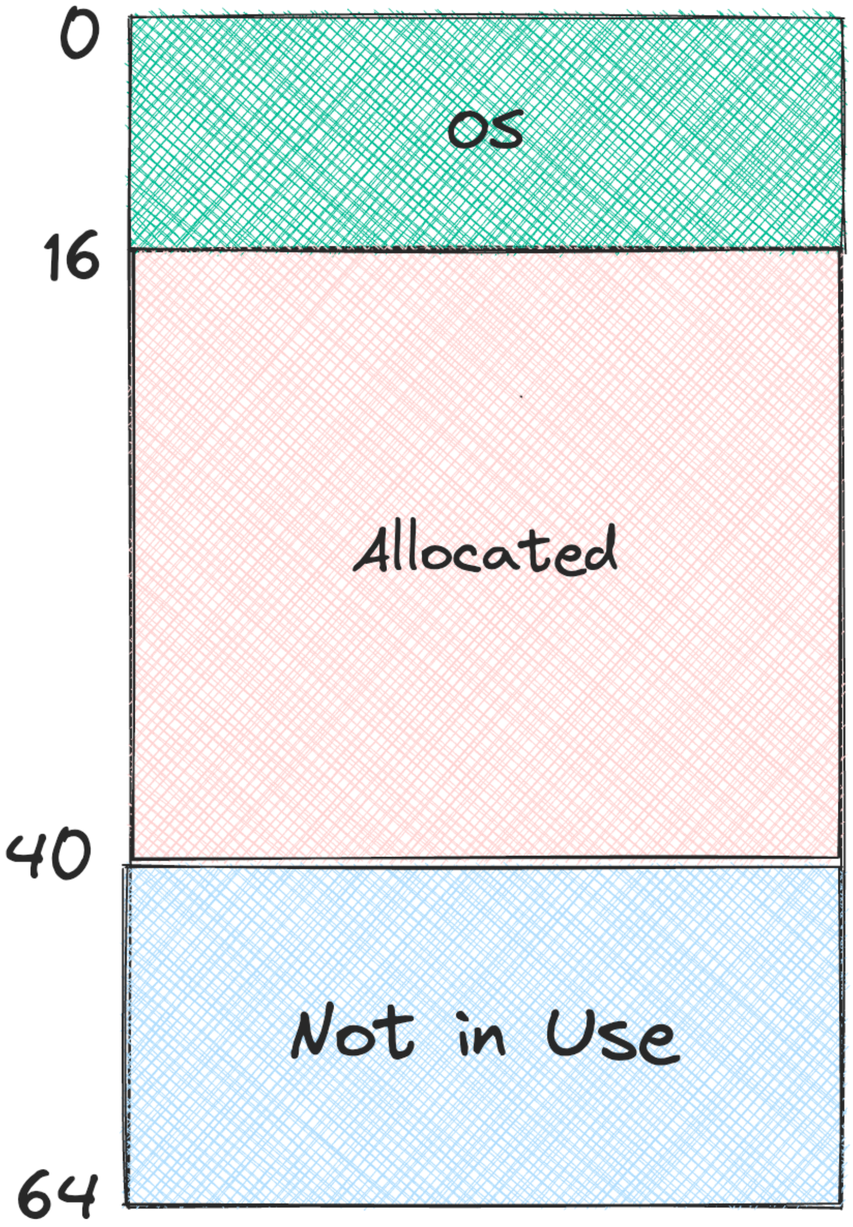
External Fragmentation

- 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?**

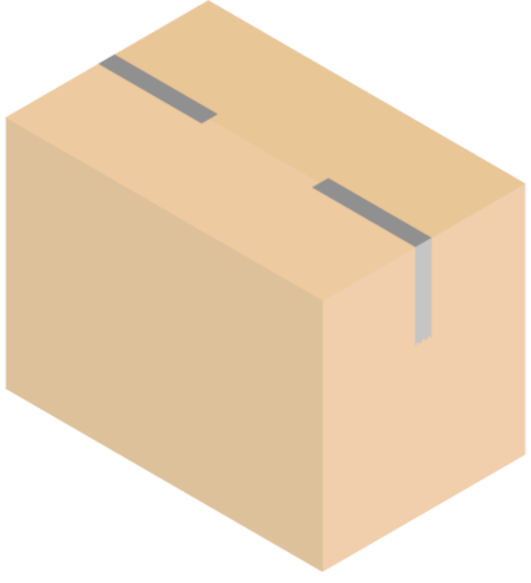
What can be done differently?

- 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

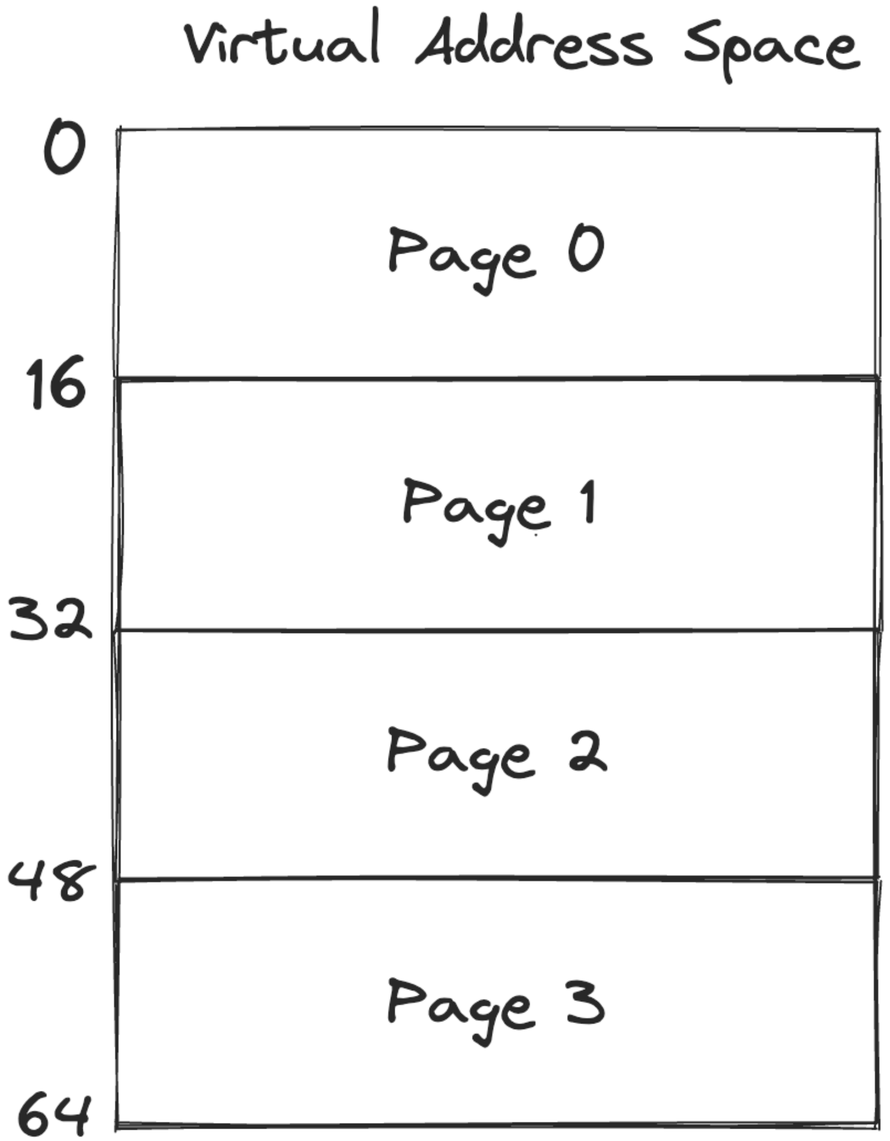


Each box can contain some items
The boxes are placed inside fixed sized shelves

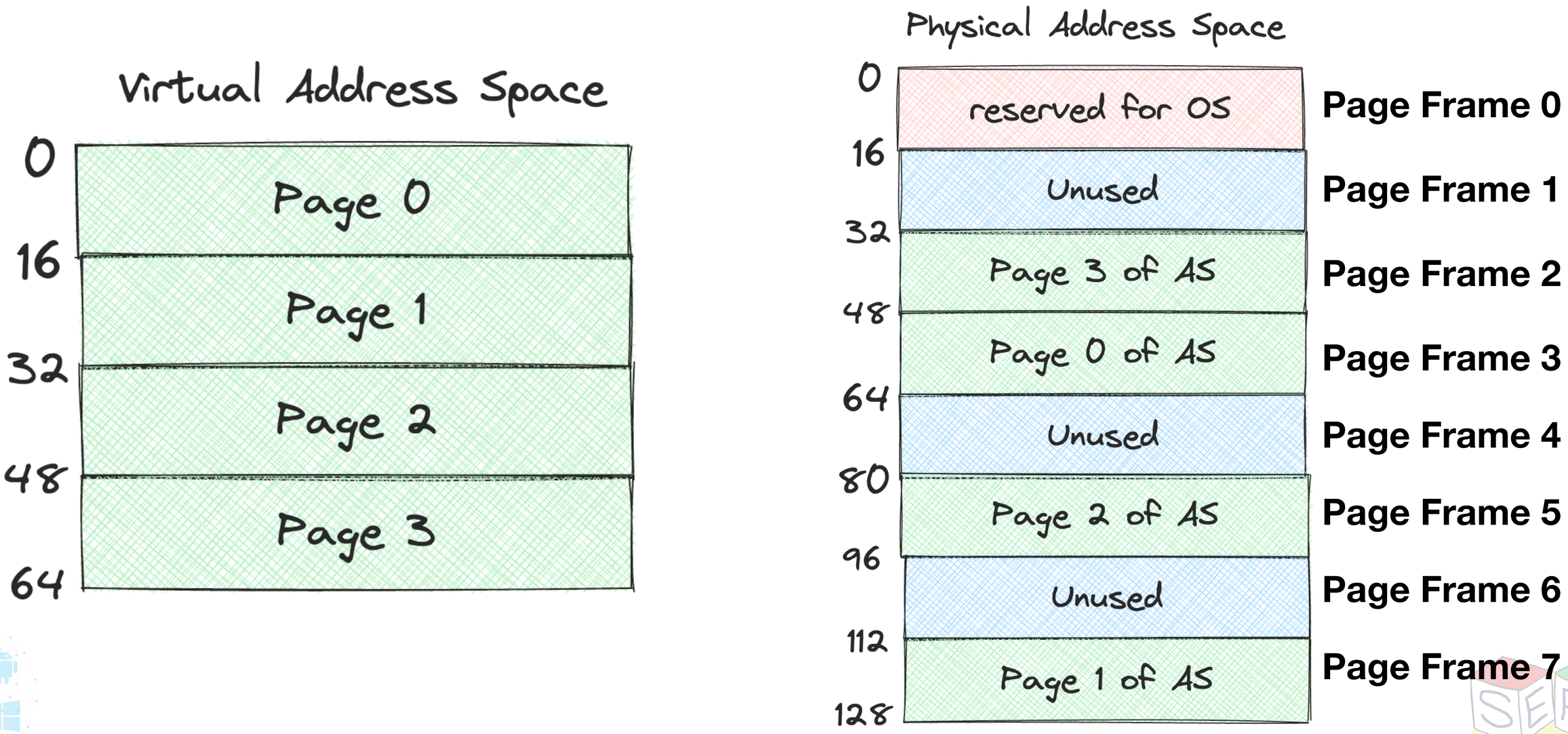
Some mapping between product, box and shelves

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

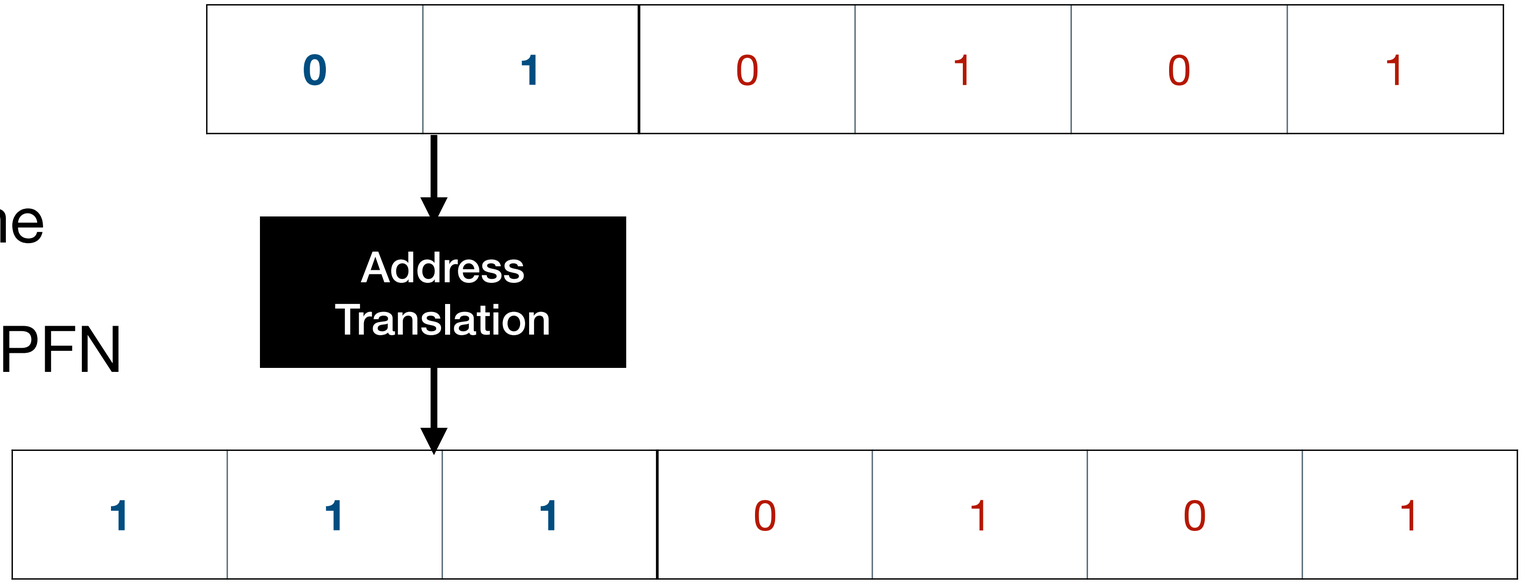
- Eg: Translation **movl <virtual address> %eax**
  - Split them into two components
    - Virtual Page Number (VPN)
    - Offset
    - VPN can be determined by number of pages
    - 64 bytes = 2^6 => 6 bits
    - Each page, 16 bytes = 2^4 => **4 bits (offset)**
    - **Remaining 2 bits for VPN**

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

Simple Example

Example instruction: mov 21 %eax

- Convert 21 into binary:



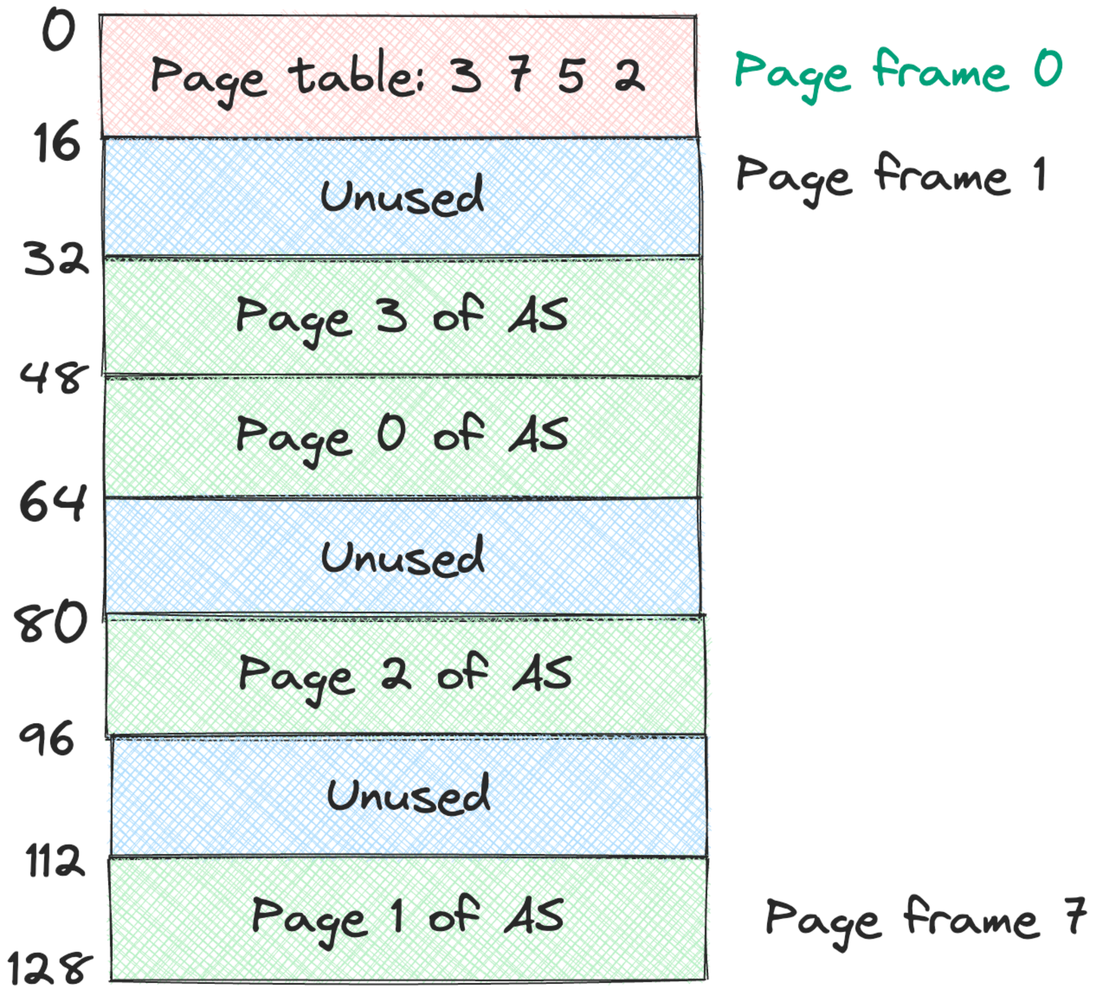
- Offset stays the same
- Its all about VPN -> PFN
- Page table helps!

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

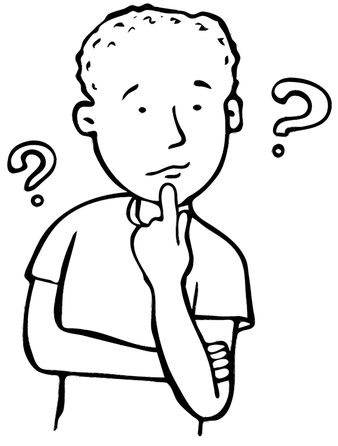
- Page tables in memory can be too big!
  - They can also slow down things
  - mov 21 %eax
  - Translate 21 into VA into PA (117 for example)
  - Before loading, system must fetch PTE from page table
  - Perform translation and get the translated address and get desired data
  - Any issues so far?

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!**

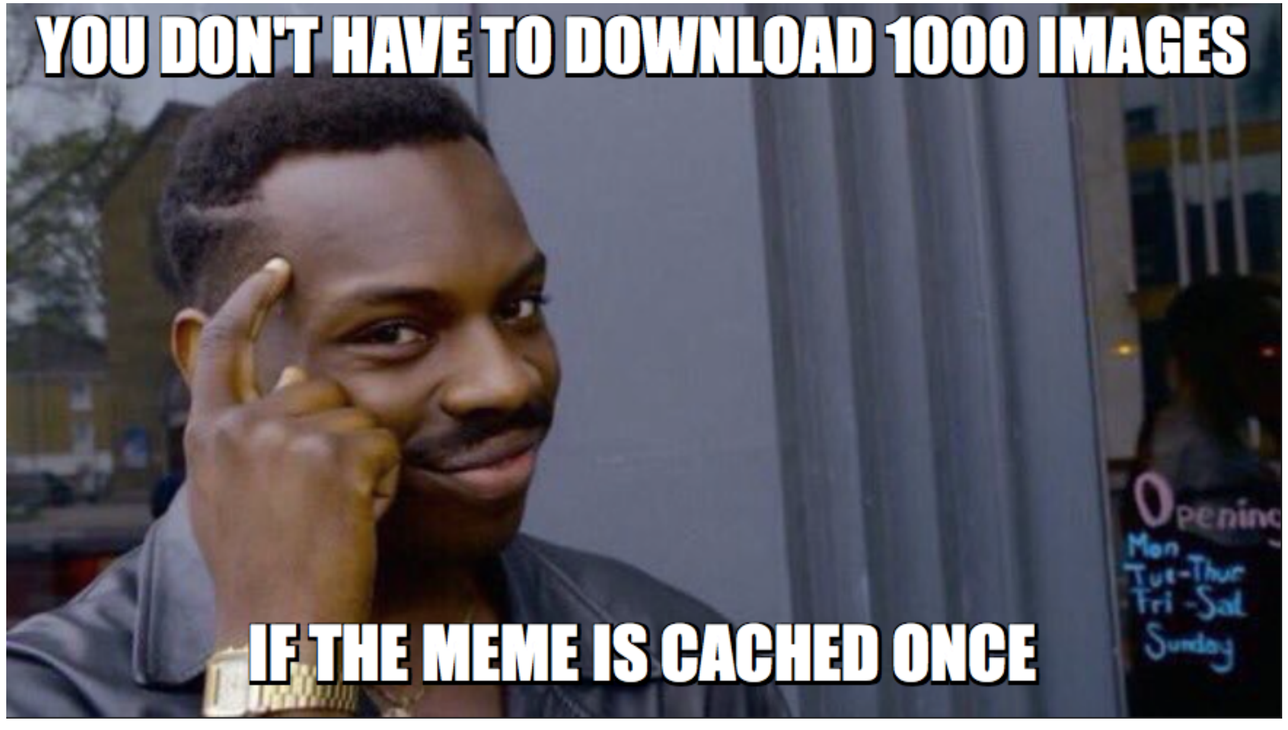
Improving the speed of translation

- 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

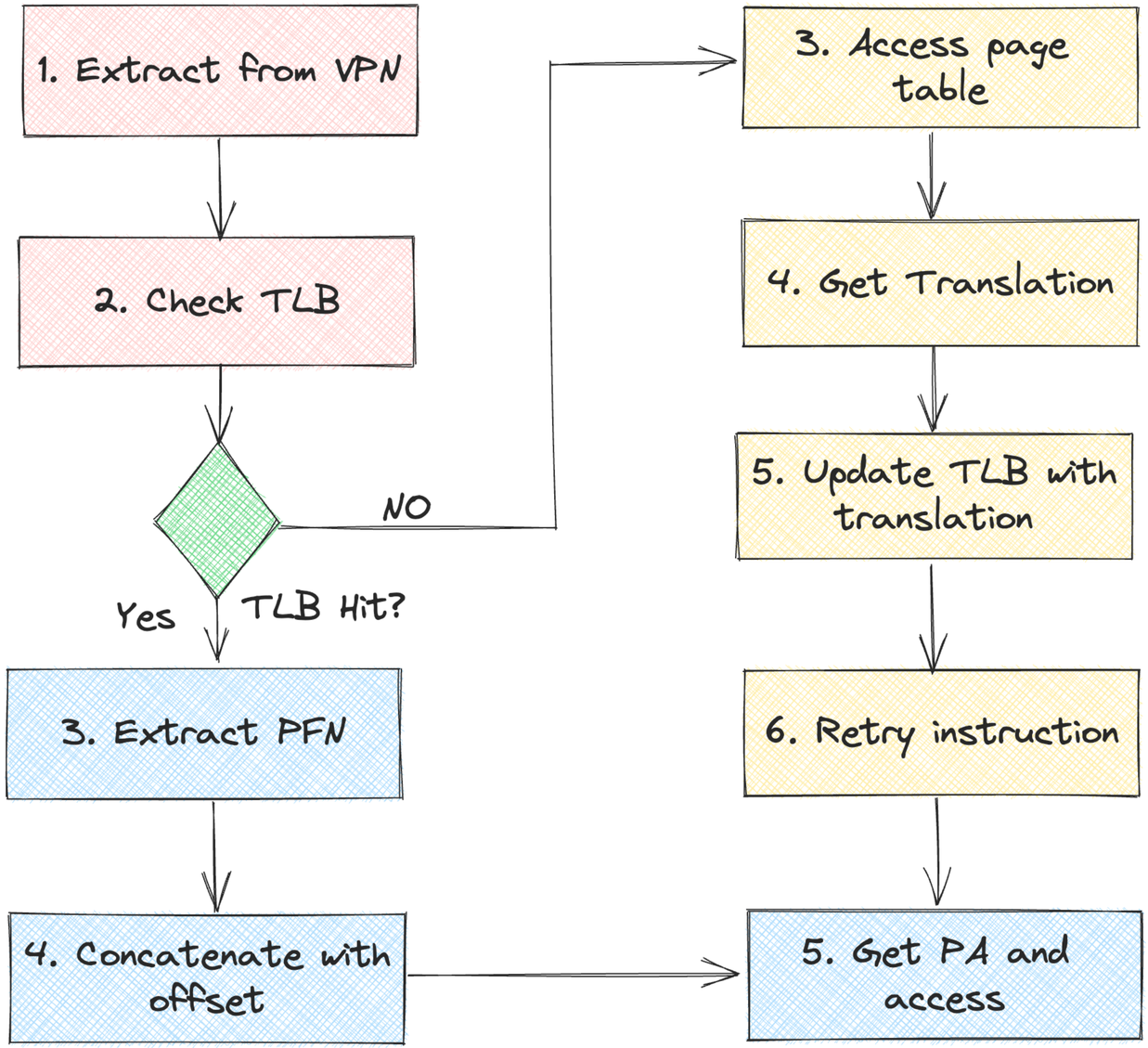
Put Hardware in action

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

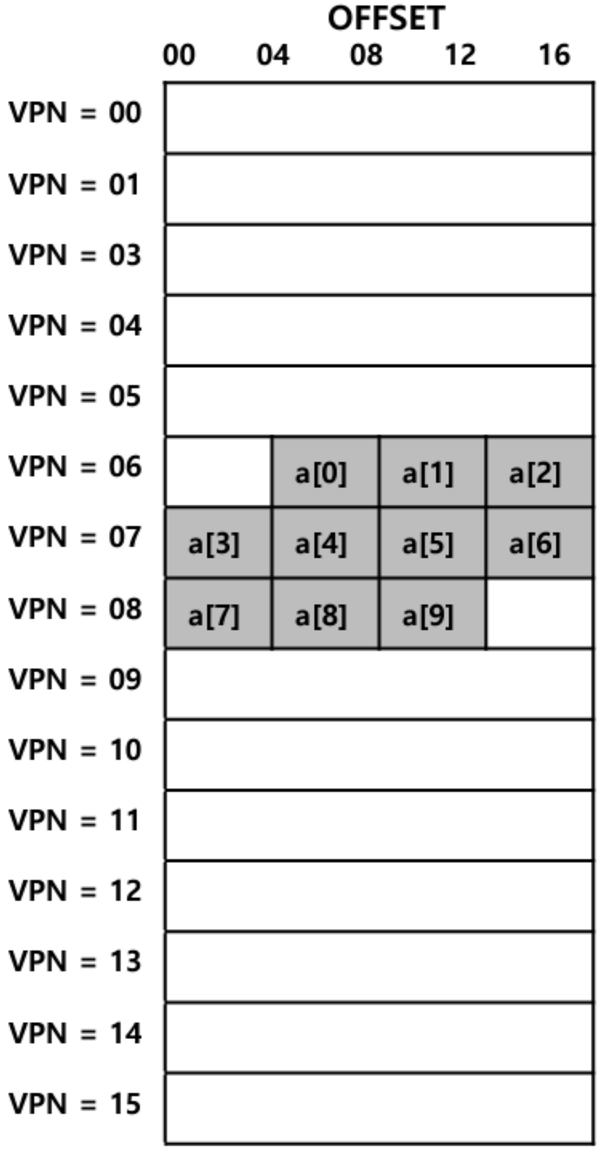
- Assume an array of 10 4-byte integers starting at VA 100. Assume an 8-bit VA space with 16 byte pages
  - This implies 4 bits for offset and 4 bits for VPN
  - Each page is 2^4 = 16 byte size
- How shall the virtual address space be organized?

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;
}
```

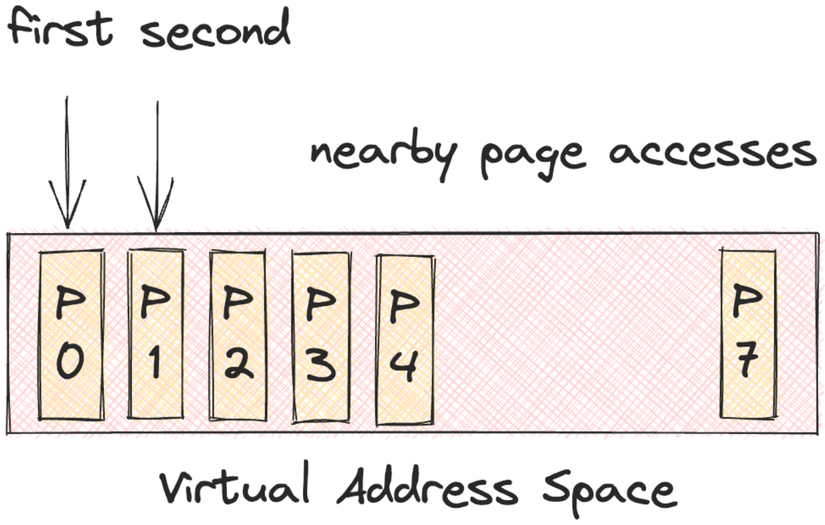
Illustrative Example: Array Access

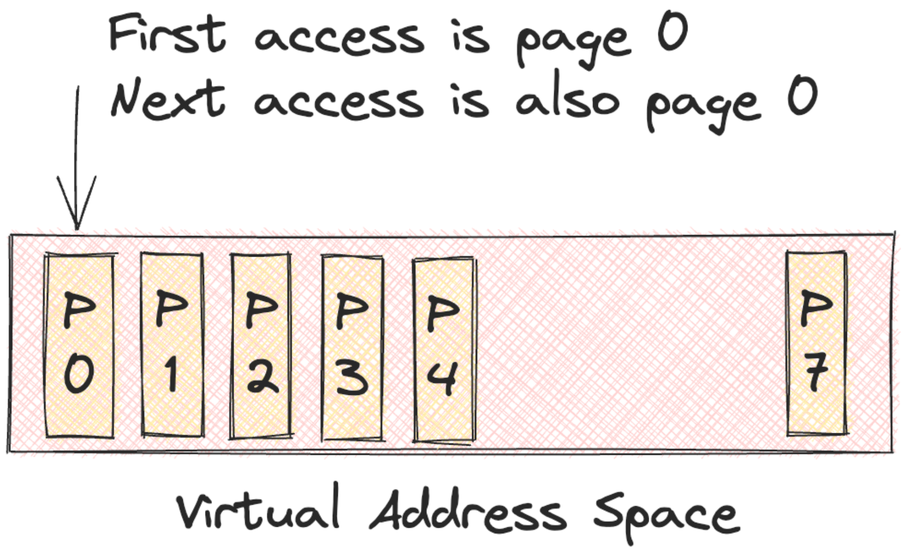
- Access to a[0] with VA 100 - How?
  - It will be a TLB miss!
- Access to a[1] will be a TLB hit!
- Access to a[3] will be a TLB miss!
- Access to a[4] to a[6] will be a hit!
- Access to a[7] will be a miss then a[8] and a[9] will be hit
- miss, hit, hit, miss, hit, hit, hit, miss, hit, hit
- Hit rate: 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?

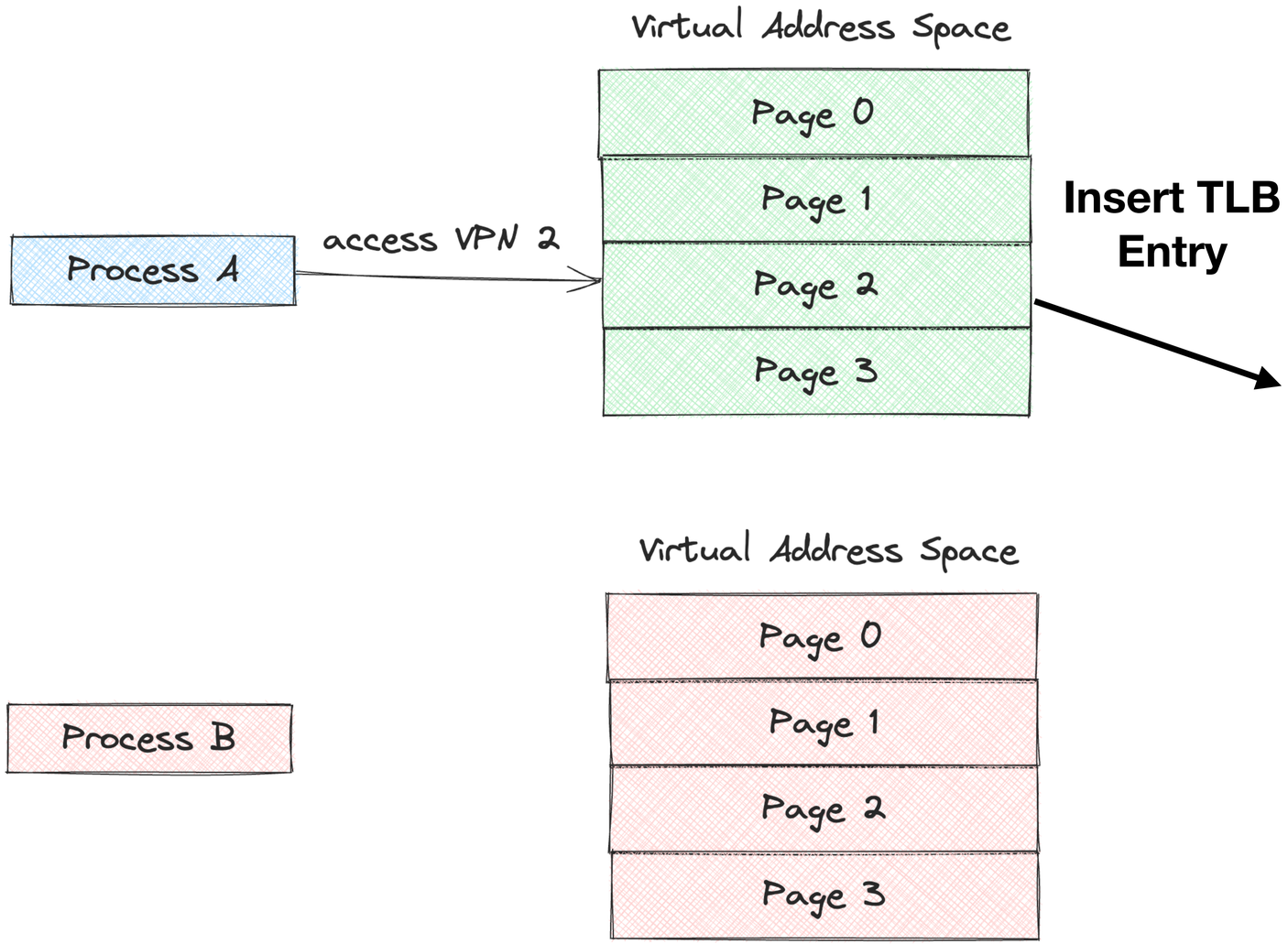
Some Caution!!

- 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?

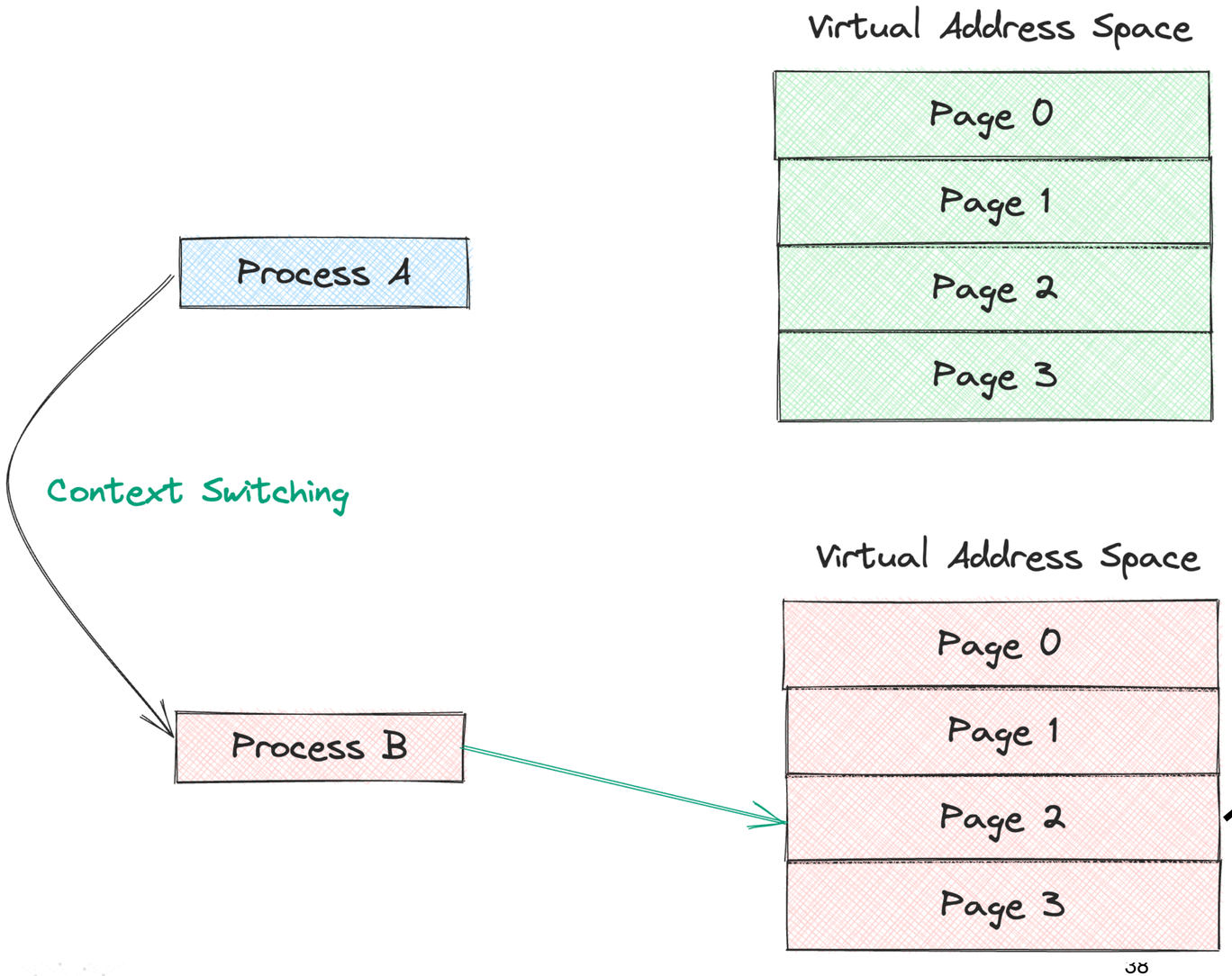


Insert TLB Entry

TLB Table

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

Context Switching and TLB



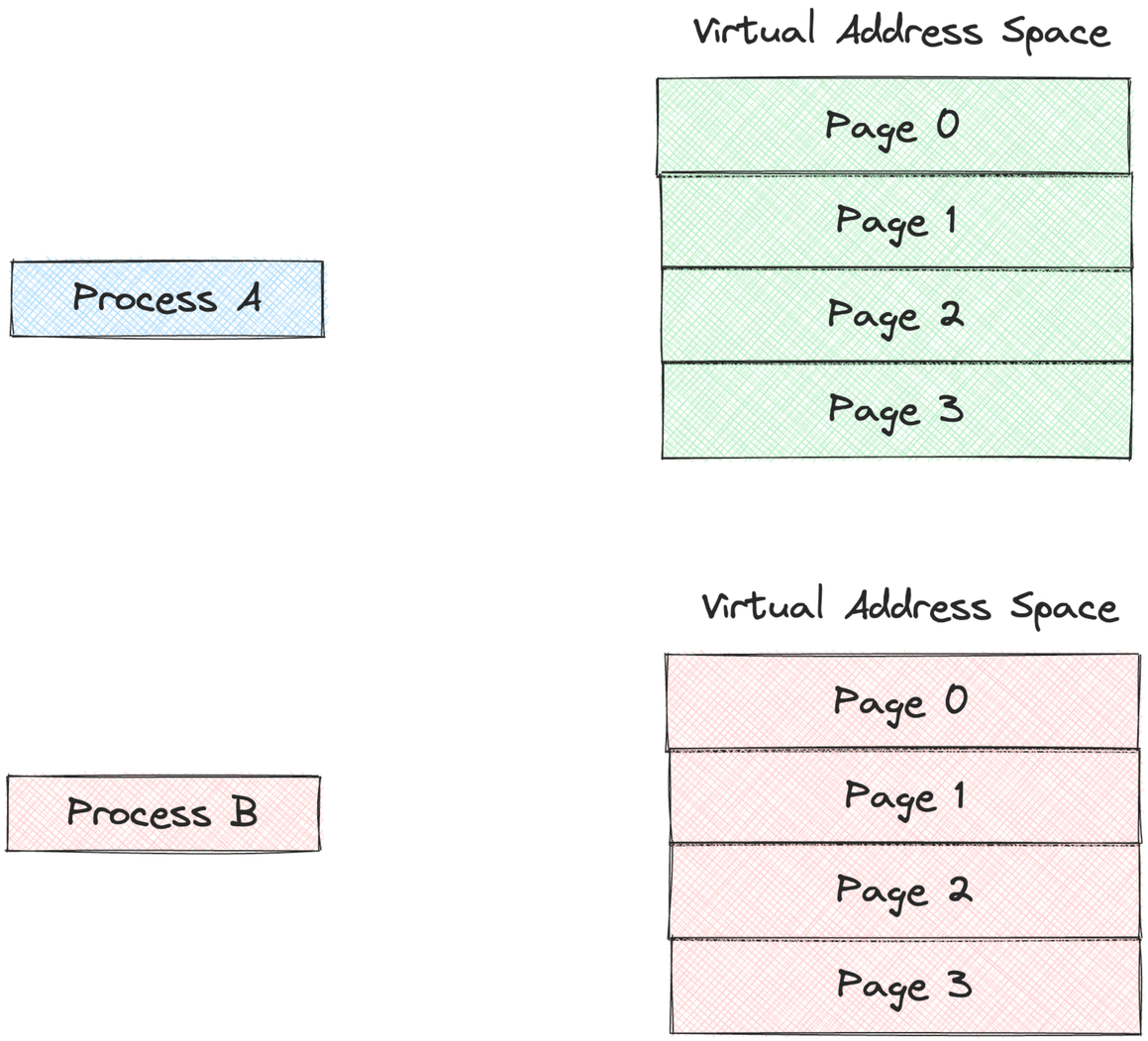
Flushing can be an option but… TLB miss?

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;
}
```

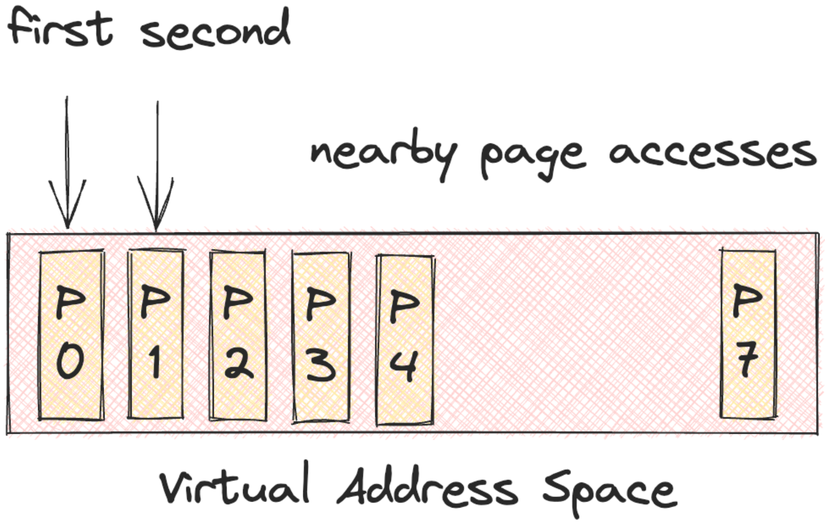
Illustrative Example: Array Access

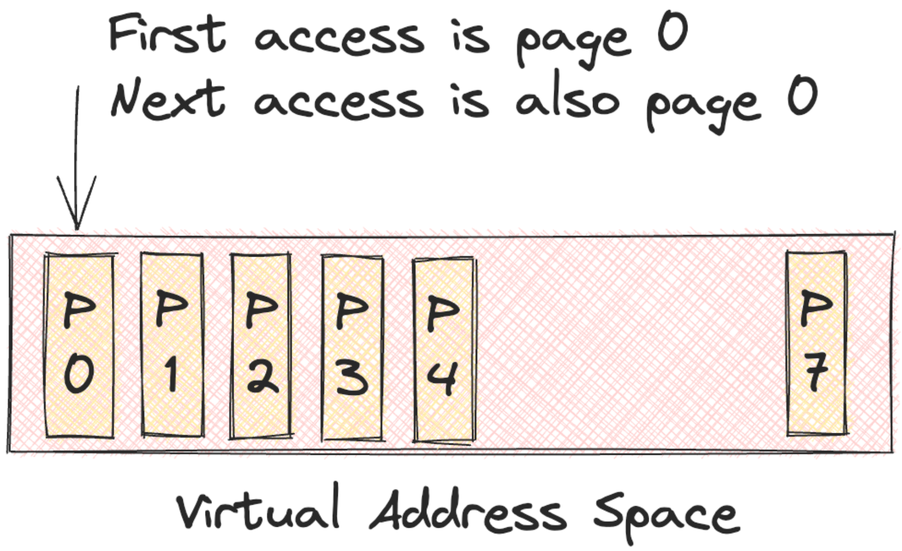
- Access to a[0] with VA 100 - How?
  - It will be a TLB miss!
- Access to a[1] will be a TLB hit!
- Access to a[3] will be a TLB miss!
- Access to a[4] to a[6] will be a hit!
- Access to a[7] will be a miss then a[8] and a[9] will be hit
- miss, hit, hit, miss, hit, hit, hit, miss, hit, hit
- Hit rate: 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?**

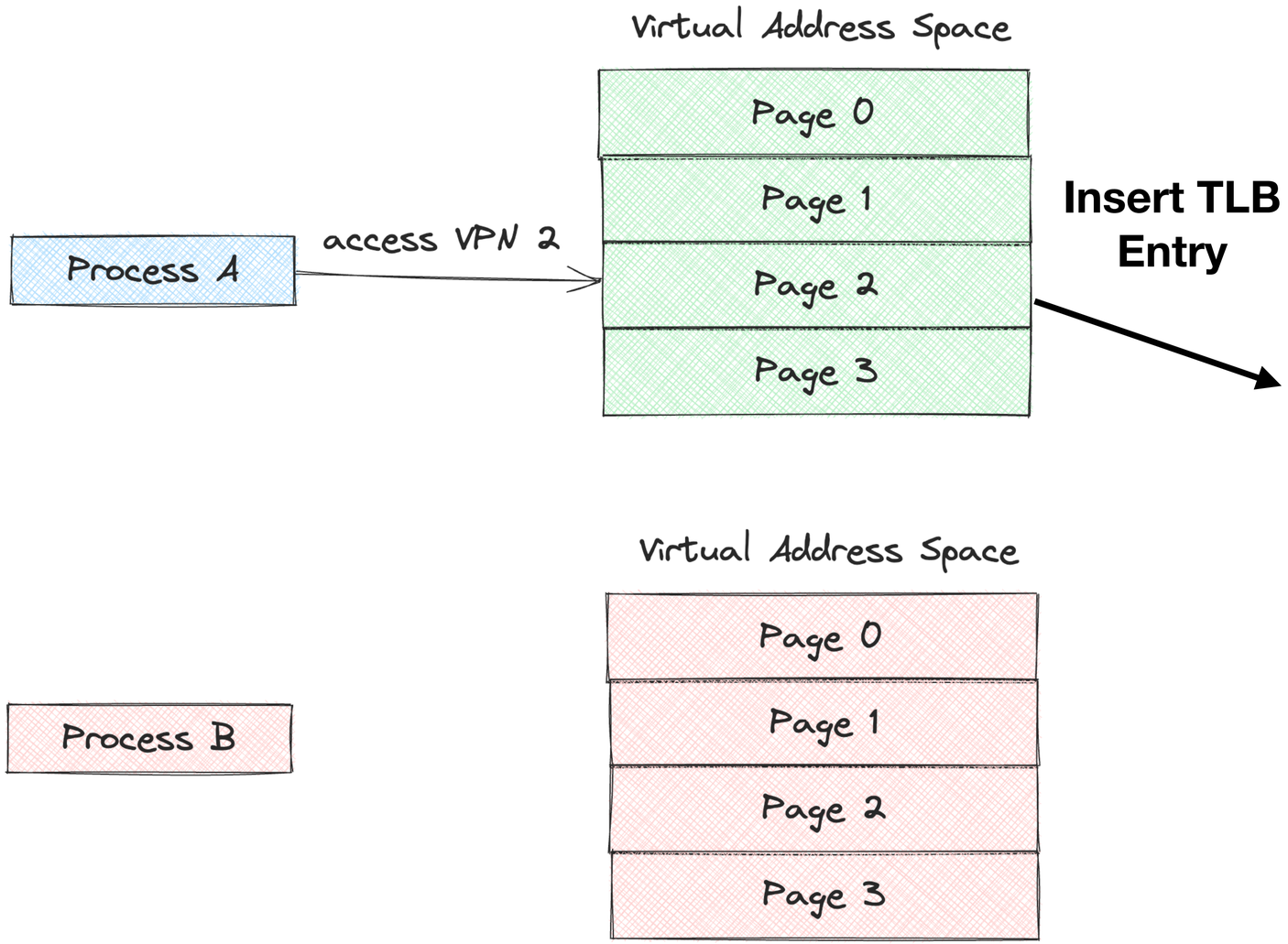
Some Caution!!

- 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.

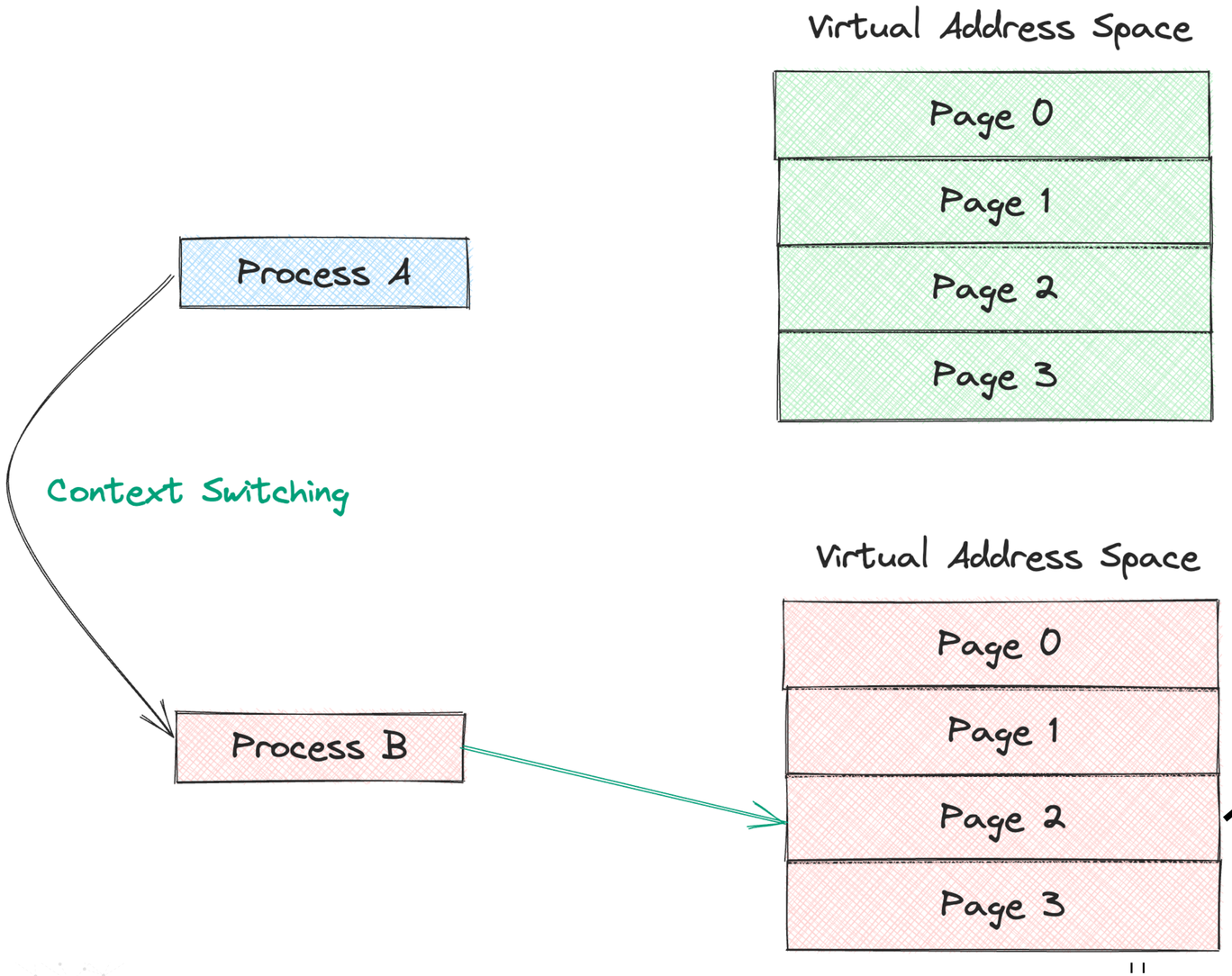
What about Context Switch?



TLB Table

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

Context Switching and TLB



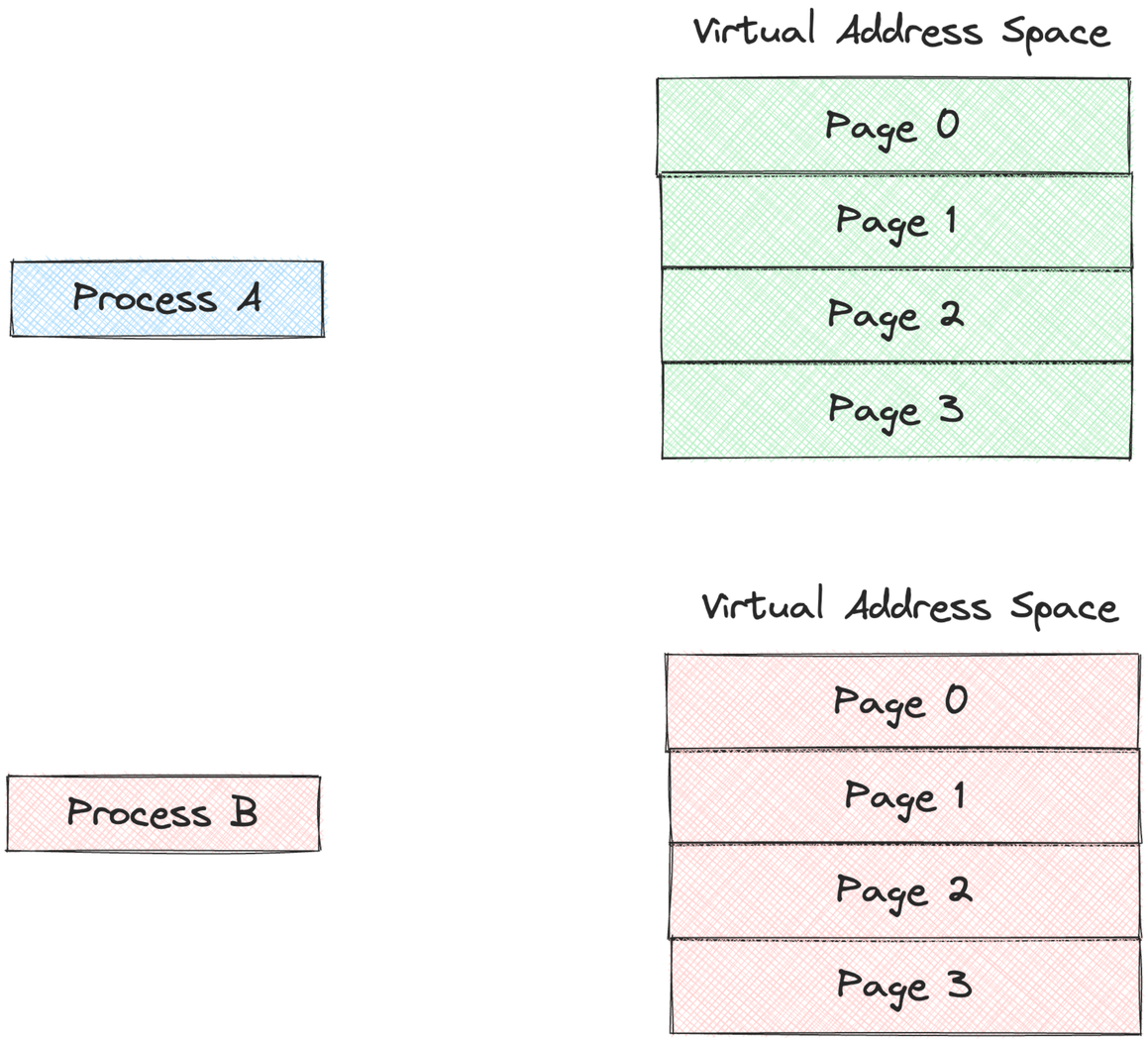
Flushing can be an option but… TLB miss?

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

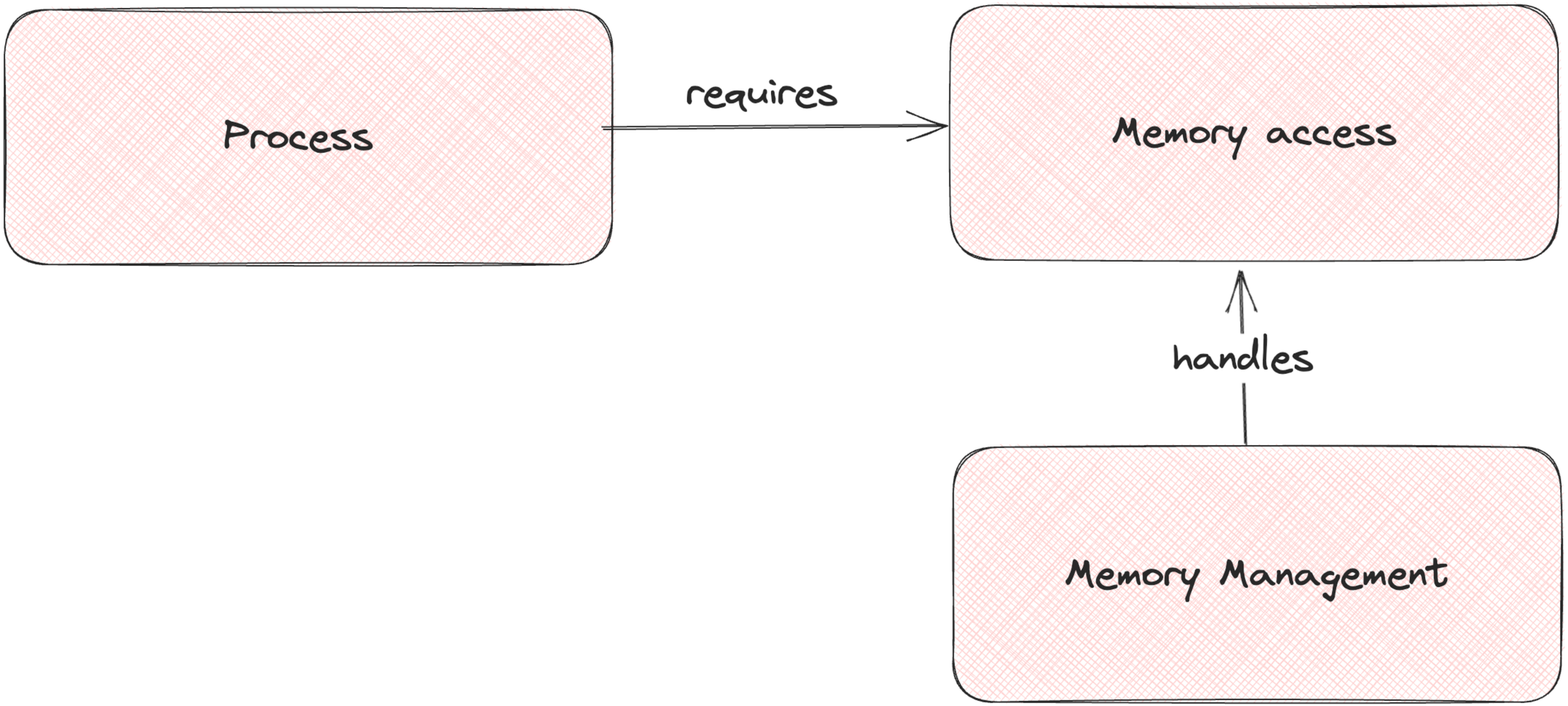
VPN	PFN	Valid	Prot	ASID
2	100	1	r-x	1
...	...	...	...	
10	100	1	r-x	2

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

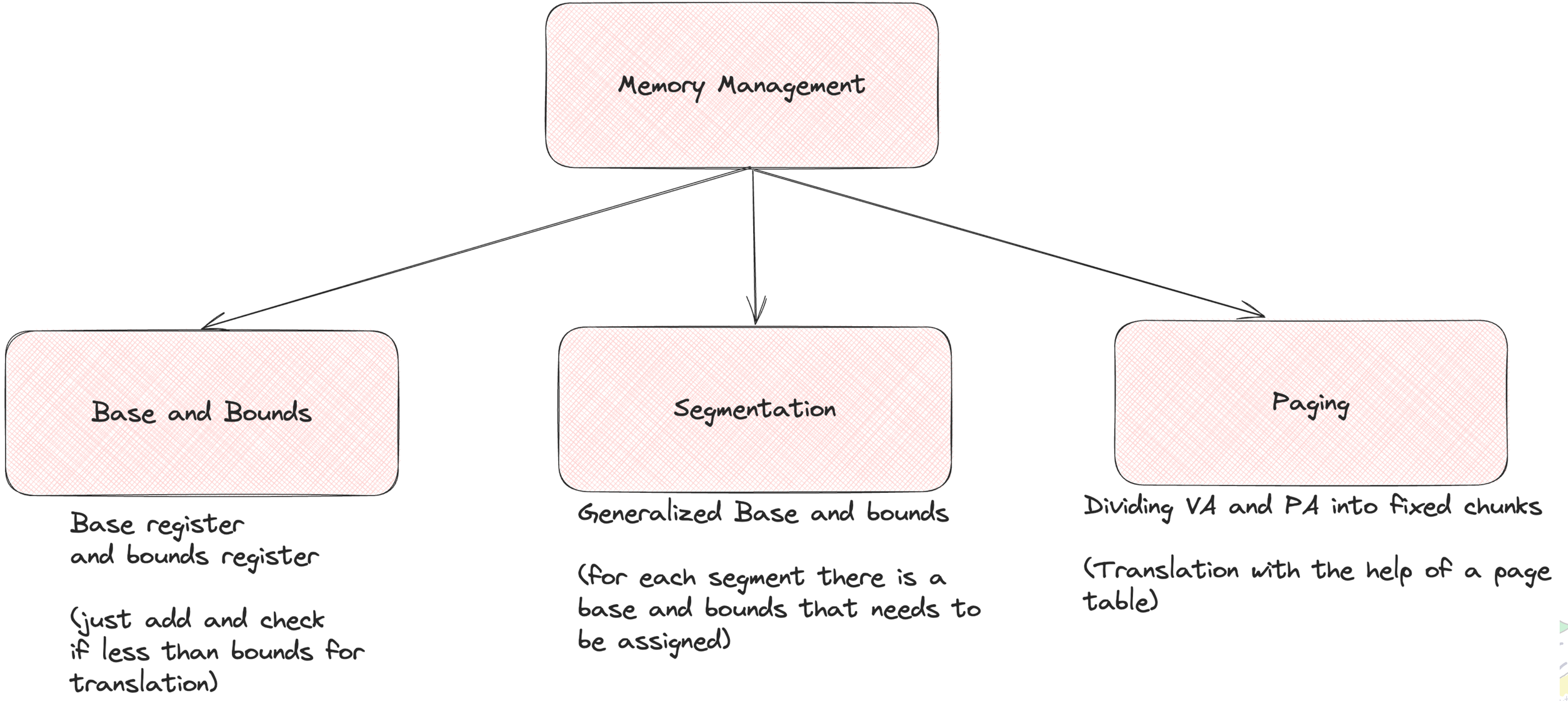
Still some issues!

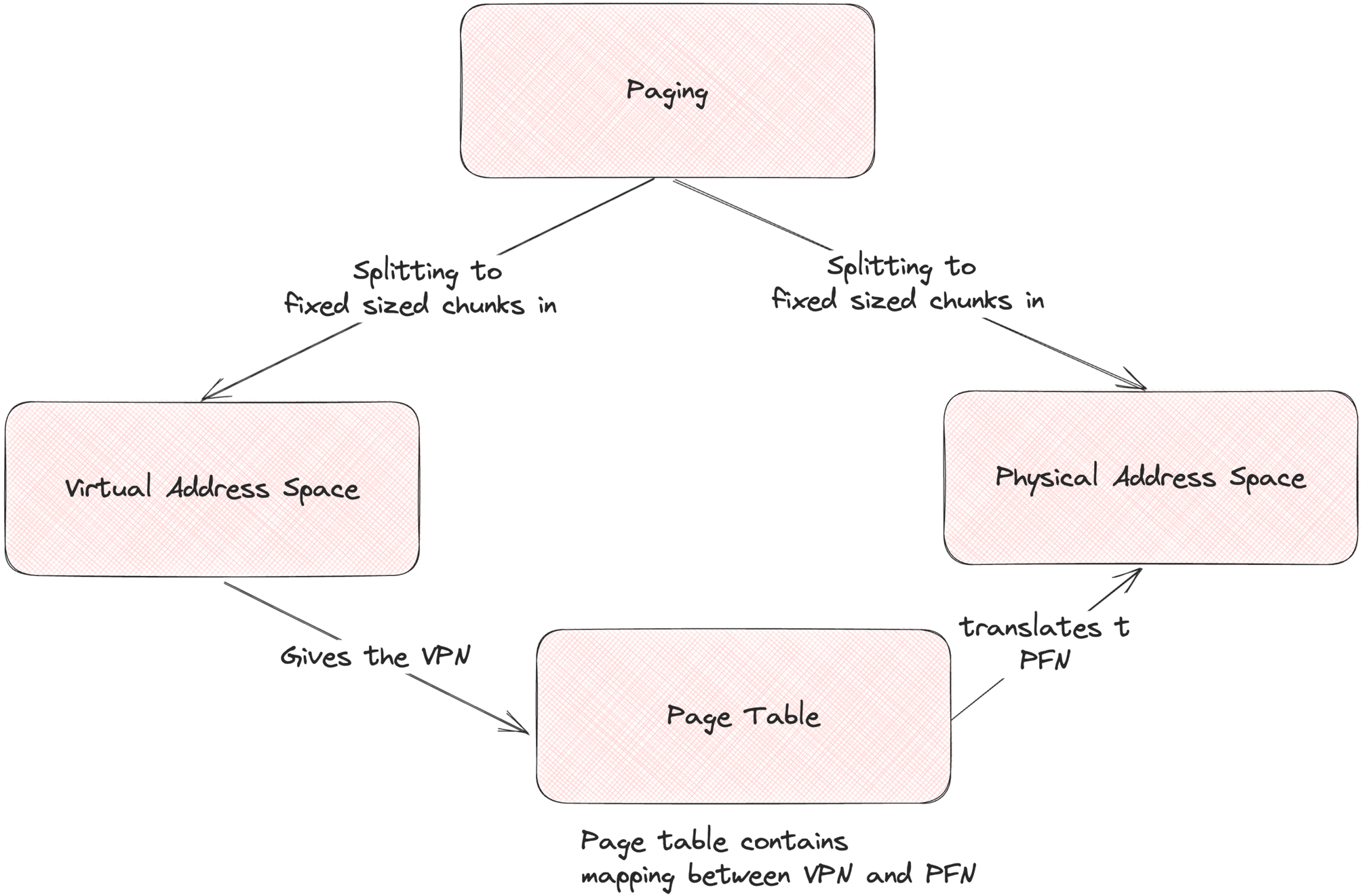
- TLB has limited size
  - Which entry to replace when adding new entries?
- What about the size of the page table?
  - Are there ways to minimise them?
  - Where will they be stored?

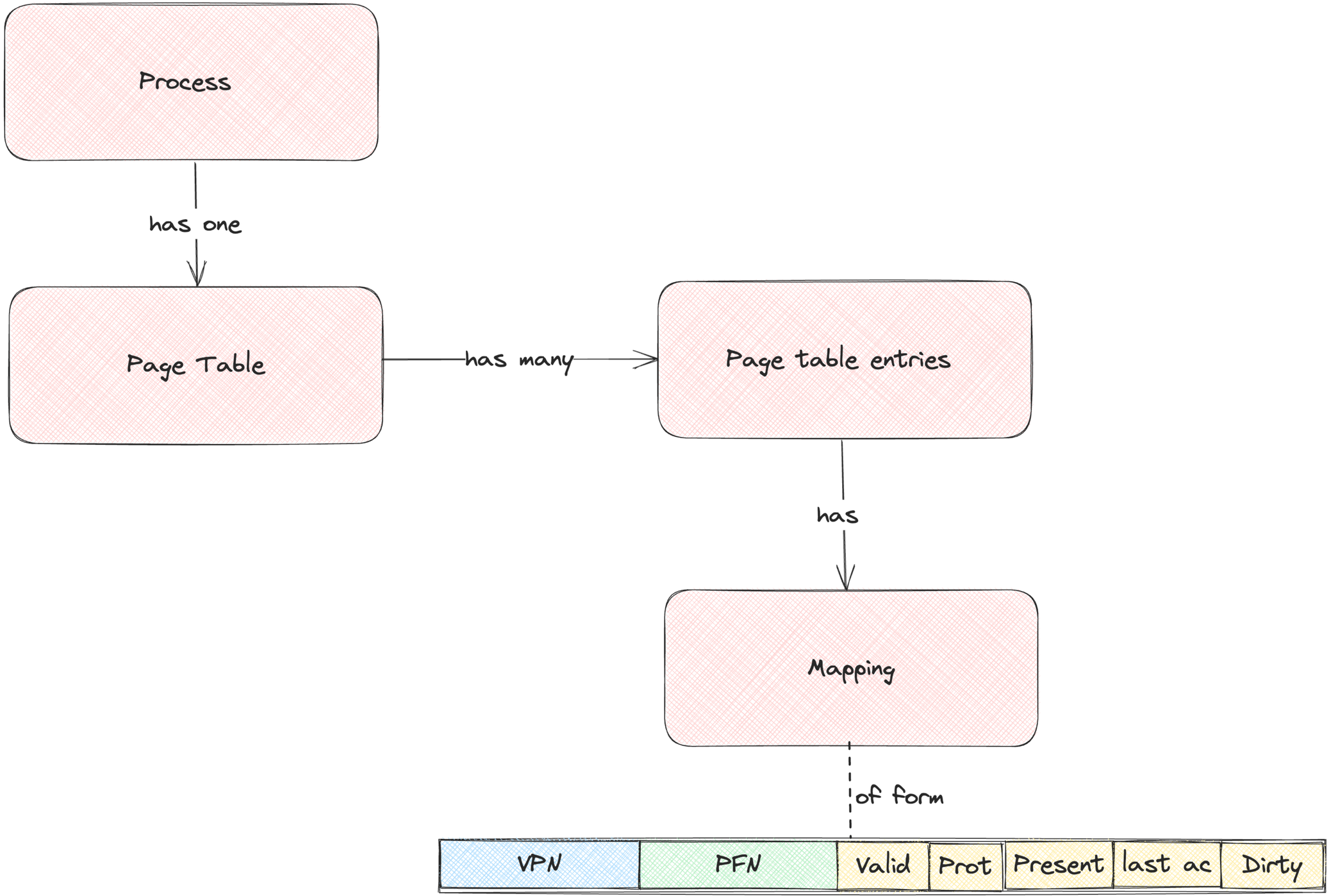
Memory Virtualization: High level view

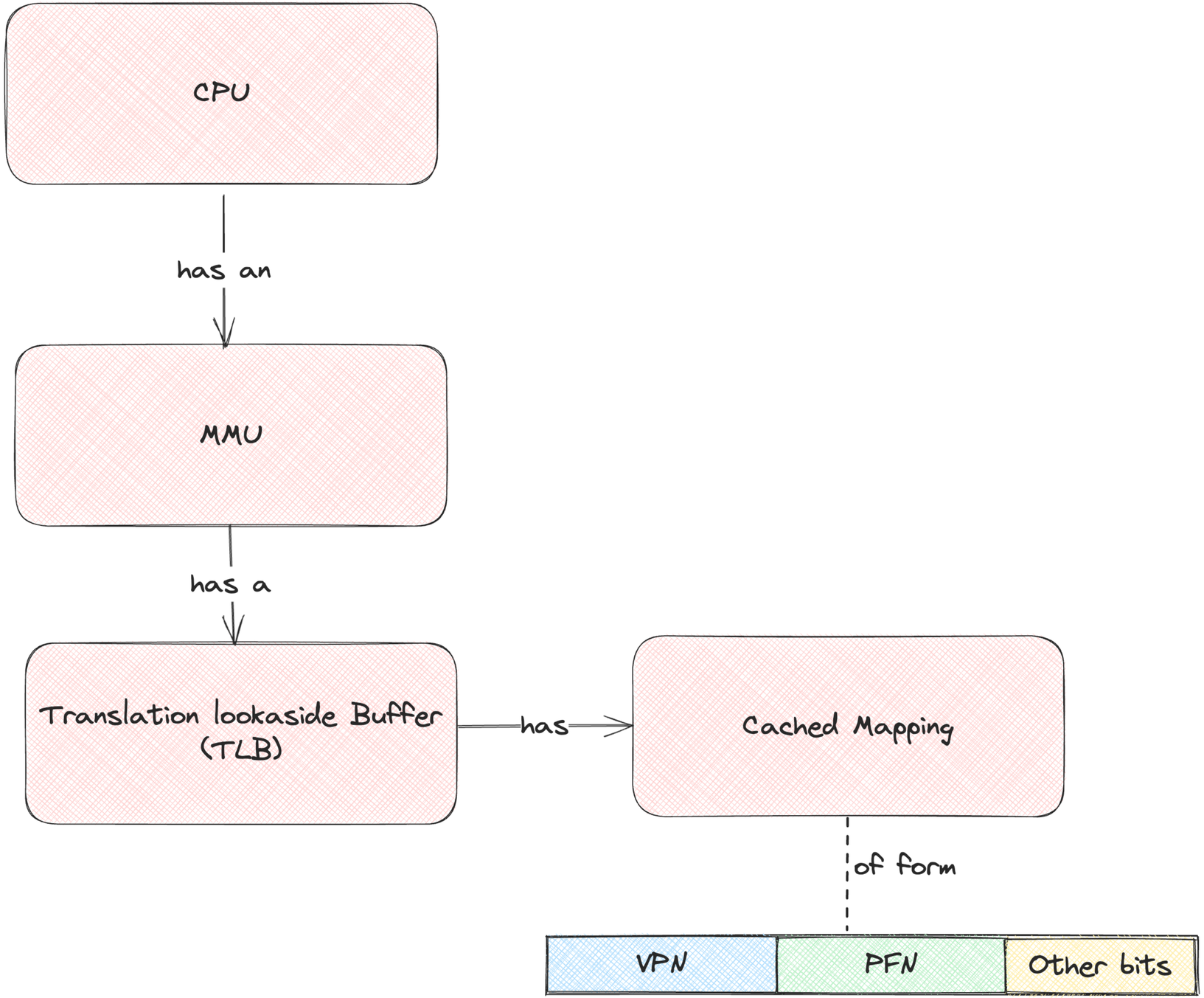


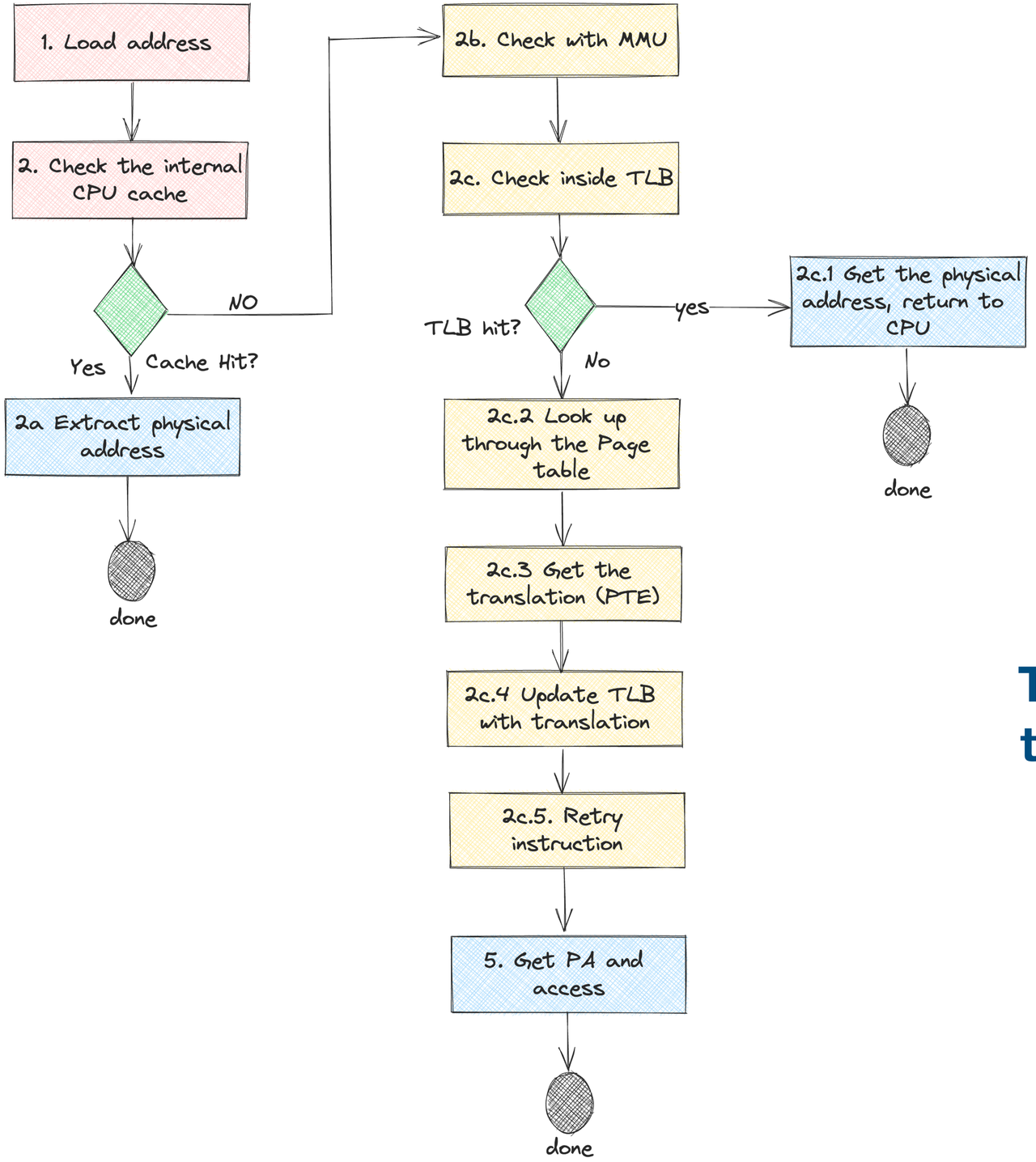
Memory Management: A Quick Recap




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?

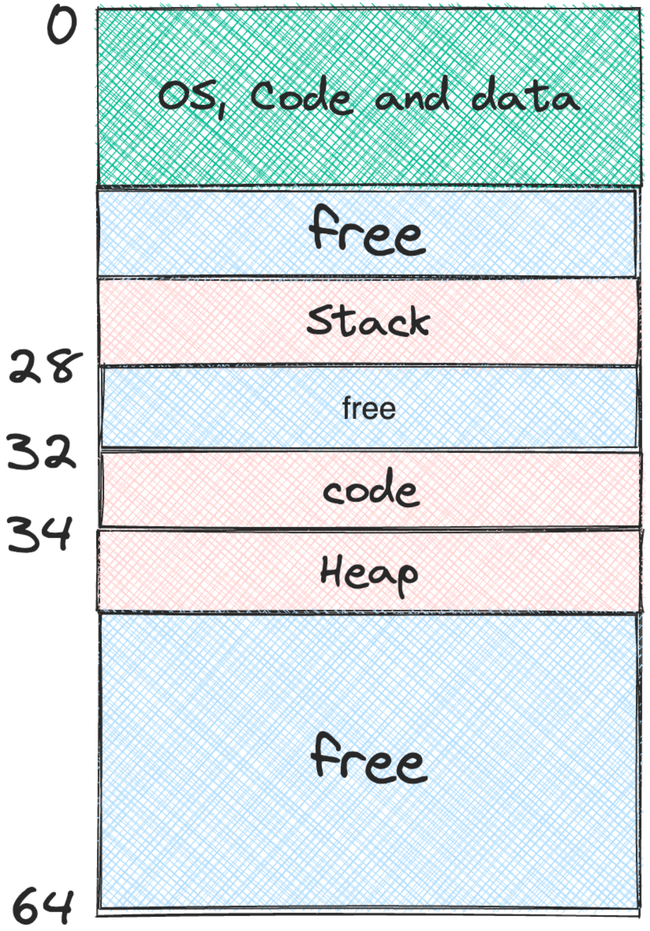
- Can we come up with smaller page tables?
  - Linear page tables did help but they can be really big!!
- Simple solution: Why not larger pages?
  - Assume a 32 bit address space but instead of 4 KB pages how about 16 KB pages
  - 16 KB = 2^{14} (Offset -> 14 bits, VPN -> 18 bits)
  - 2^{18} entries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

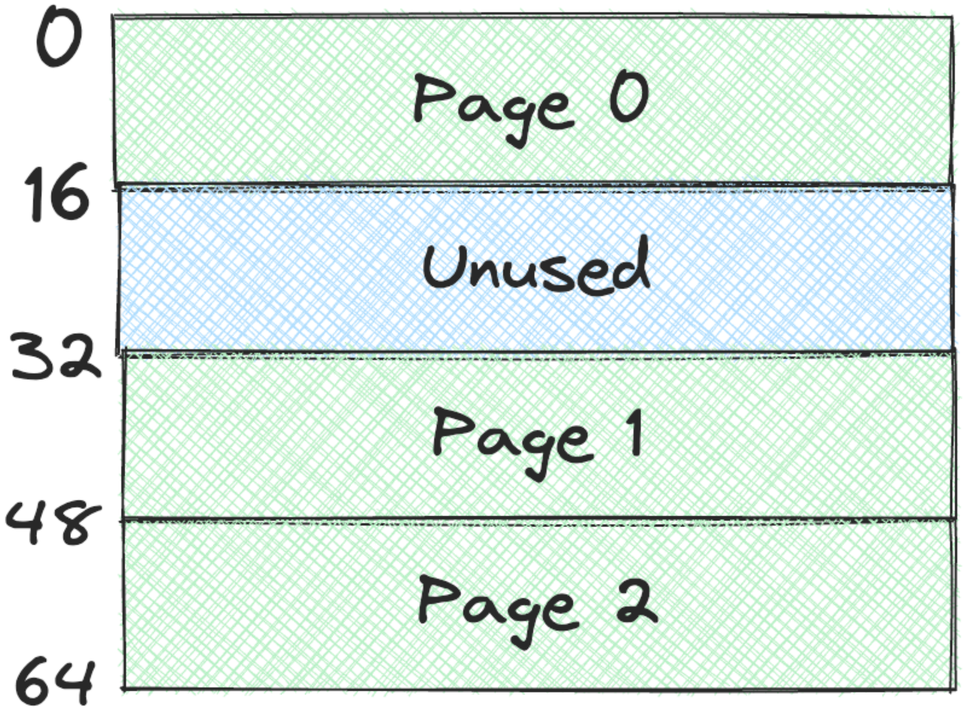
Increasing Page Size: How good is it?

- 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?



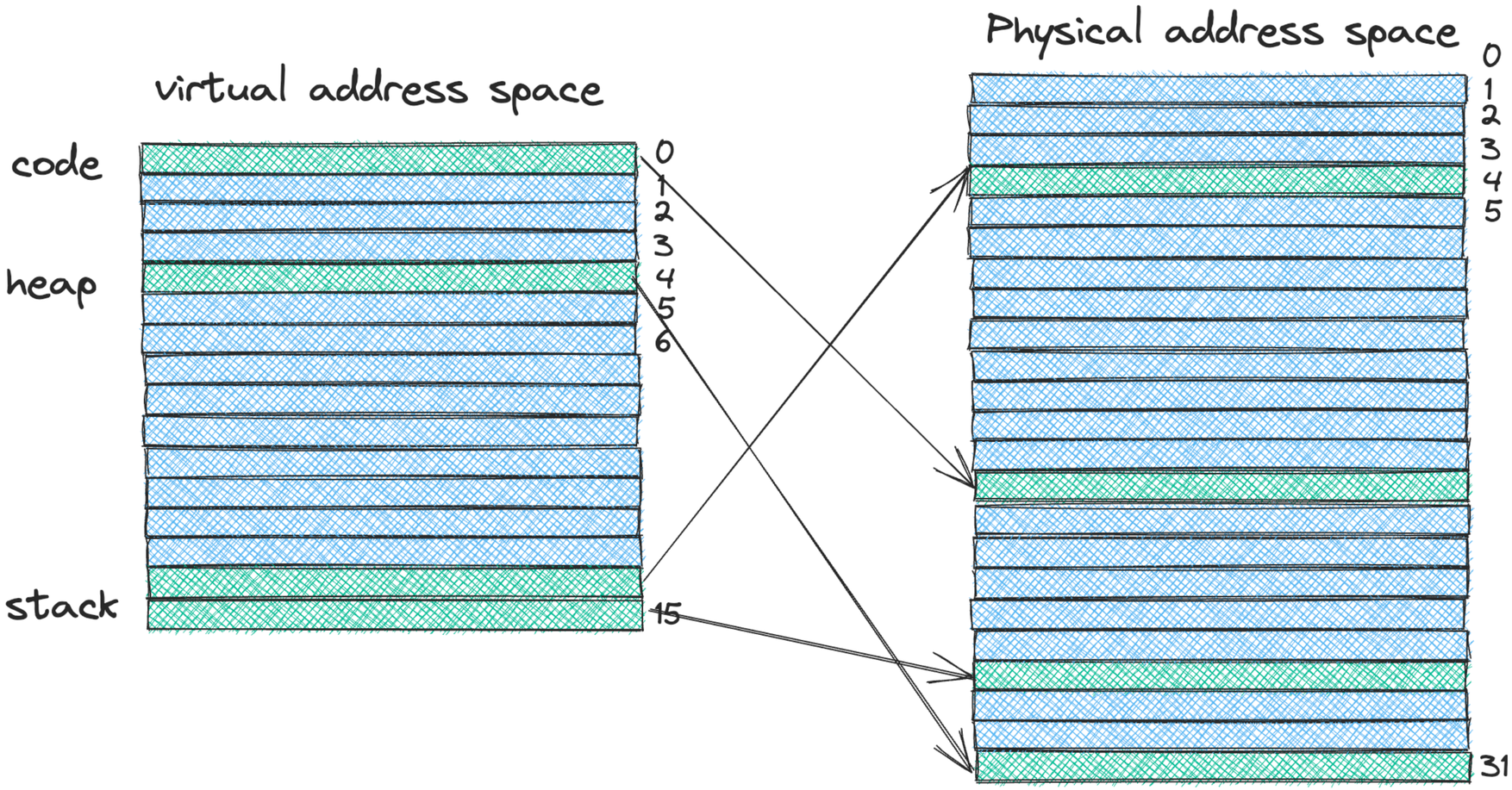


Hybrid Approach: Paging and Segmentation

- 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

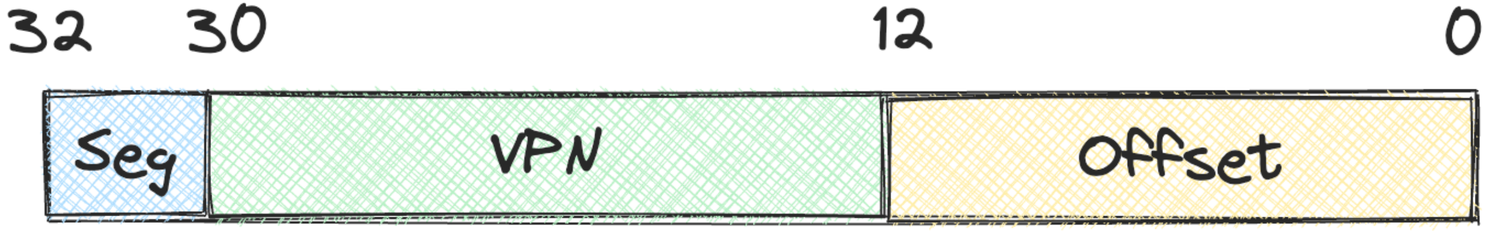
Lots of space Will be wasted!!

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?



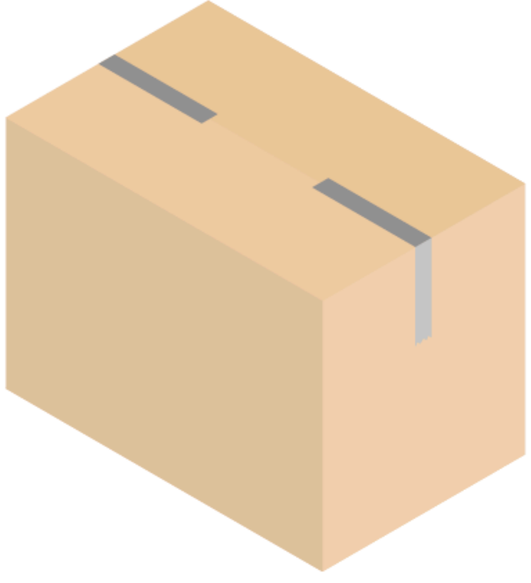
How good is the Hybrid Approach

- 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

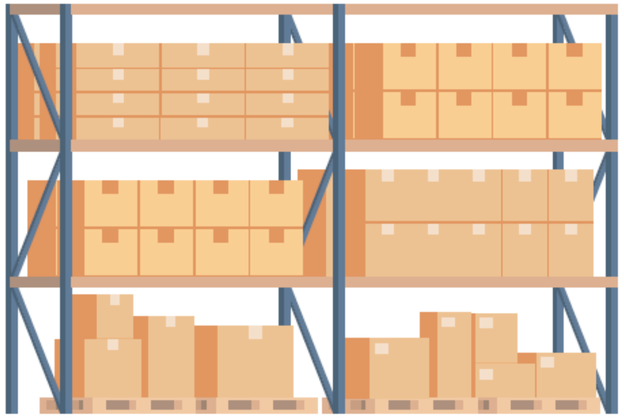
Going back to the Analogy

Warehouse

Location List

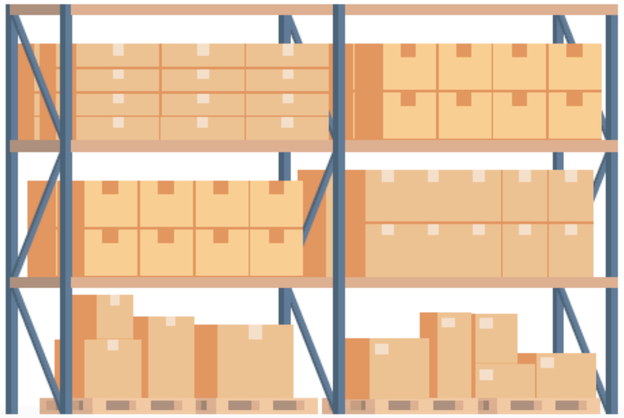
Index

TV



Index

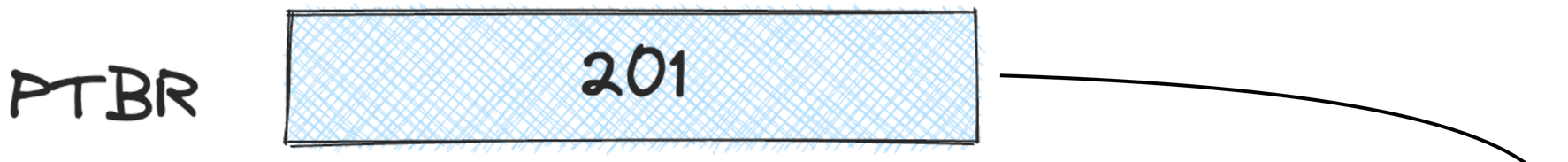
Living Room Furniture



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!

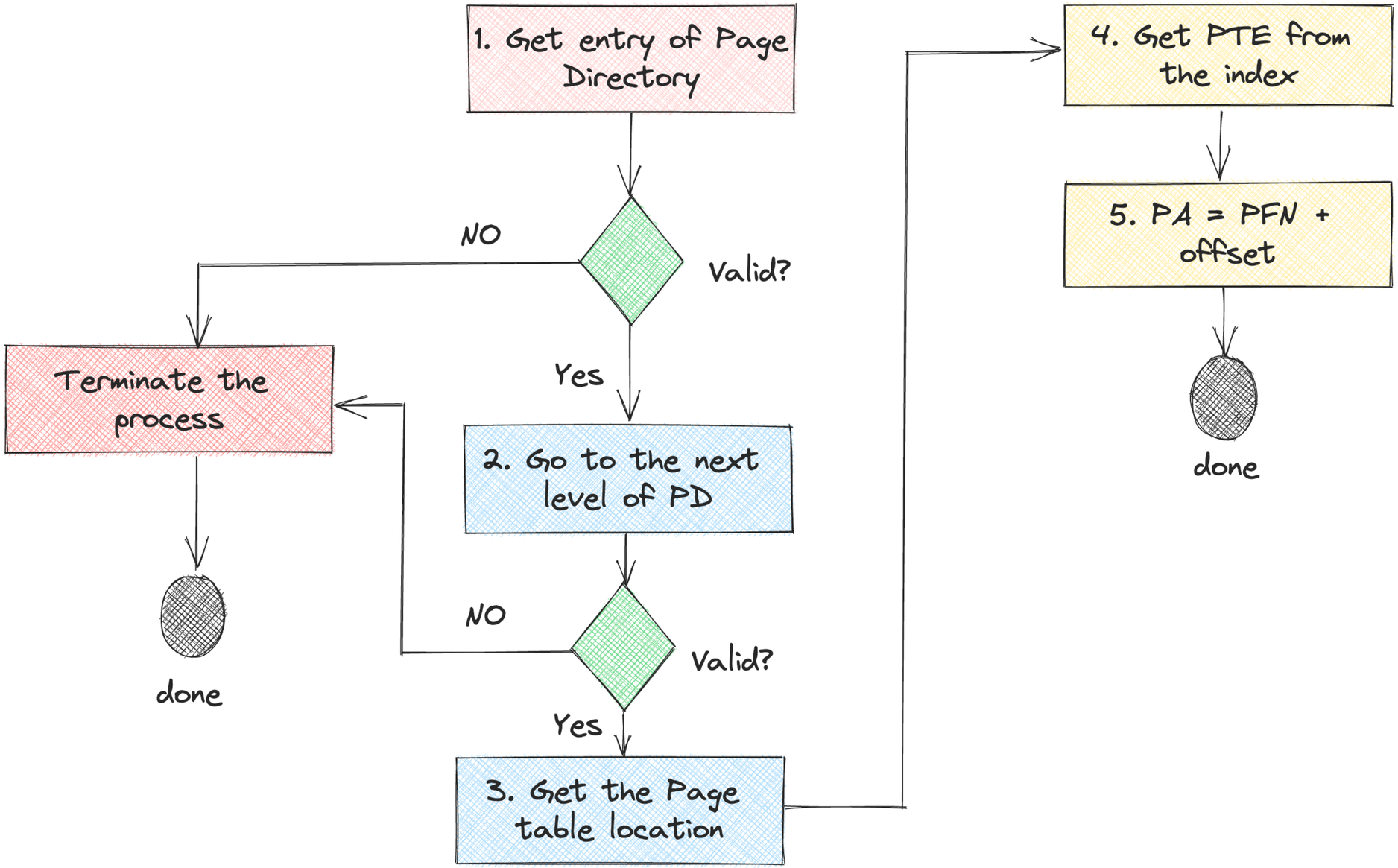
More than Two Levels

- 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

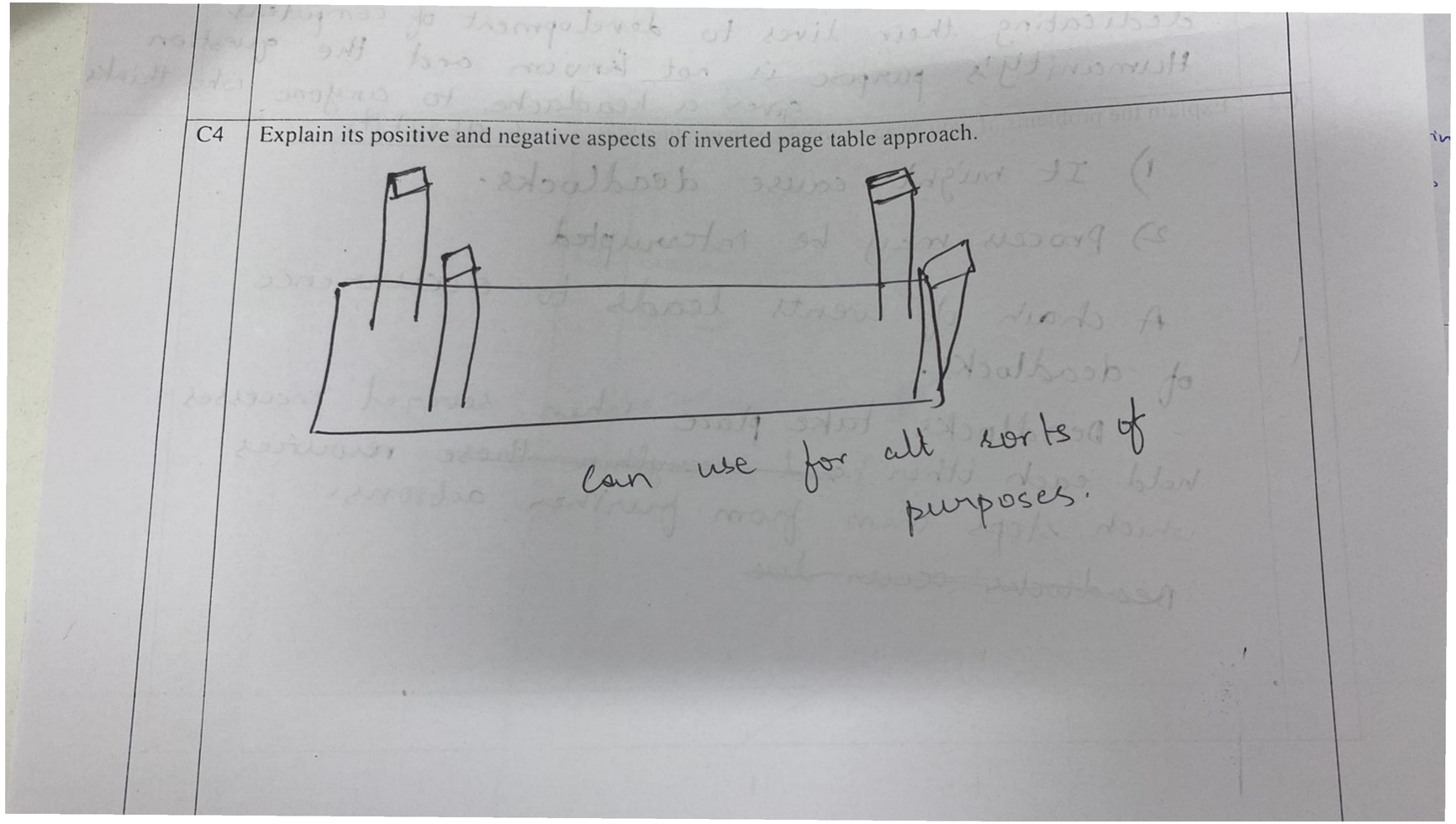
- Page directory itself is divided into multiple pages
  - One additional meta directory on top of all directories
  - The top most one points to the right directory
  - The directory further points to the correct Page table to fetch PTE
  - The PTE and offset is combined to generate physical address
- 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

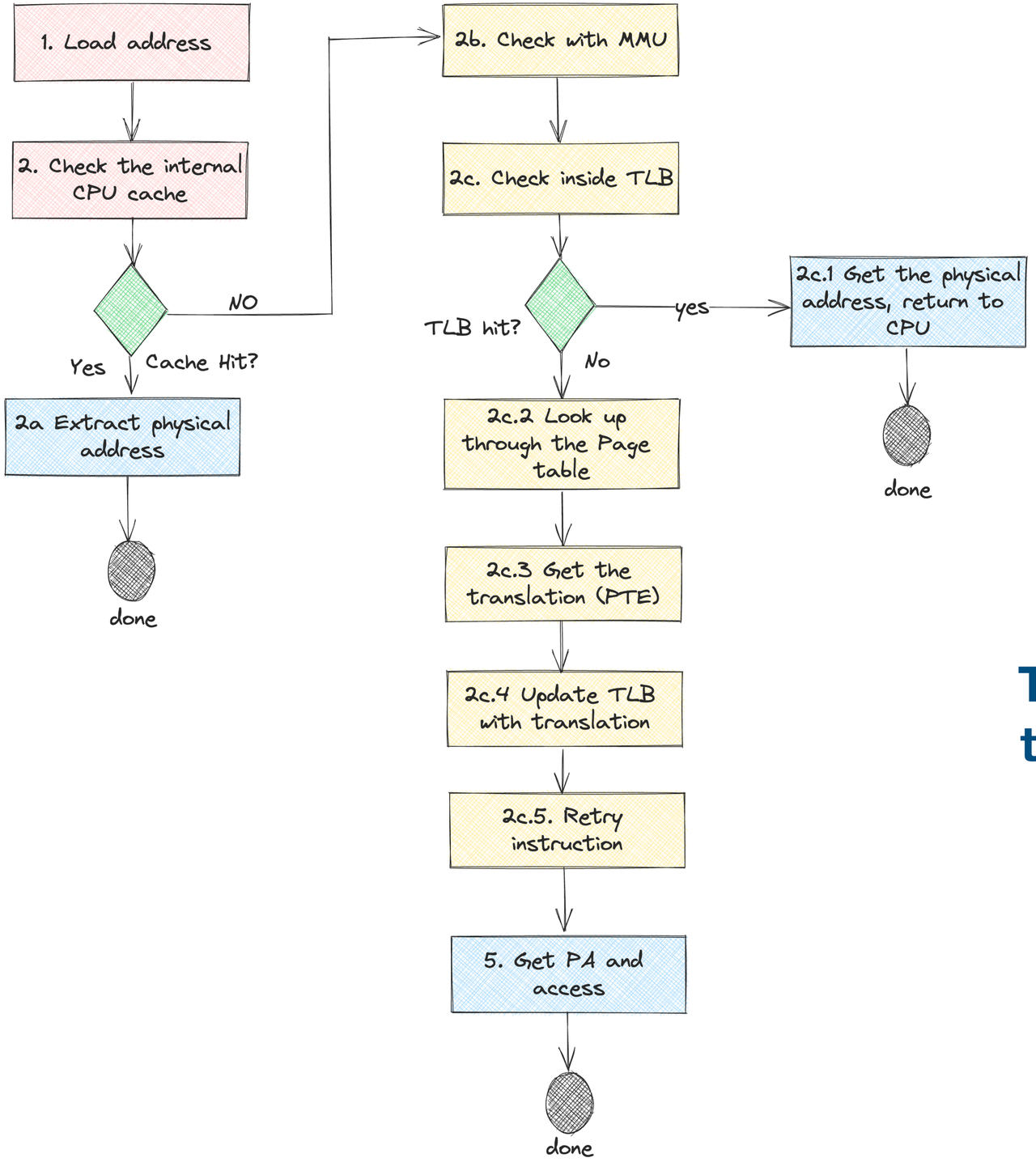
https://karthikvaidhyanathan.com

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

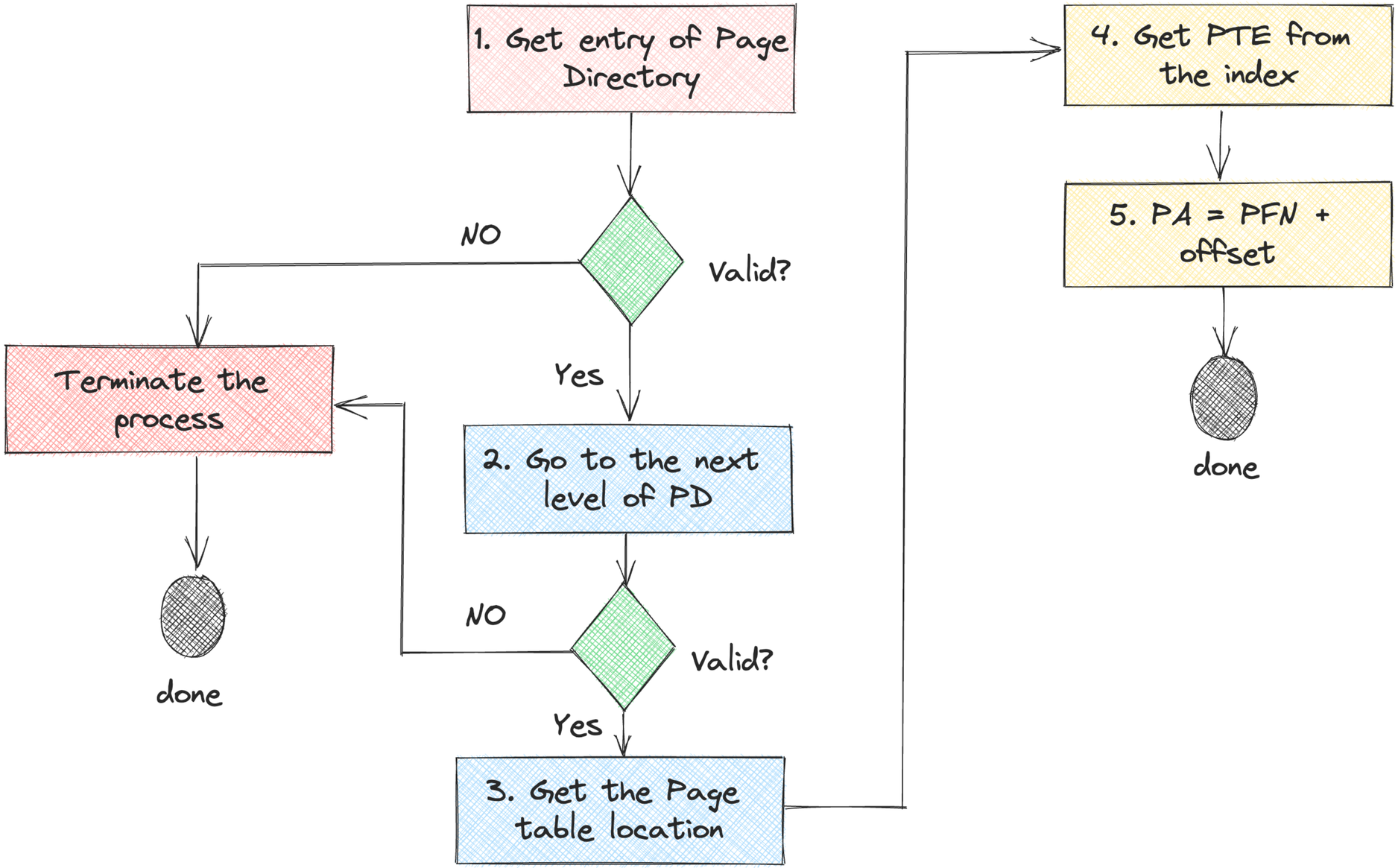
Sources:

- 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?

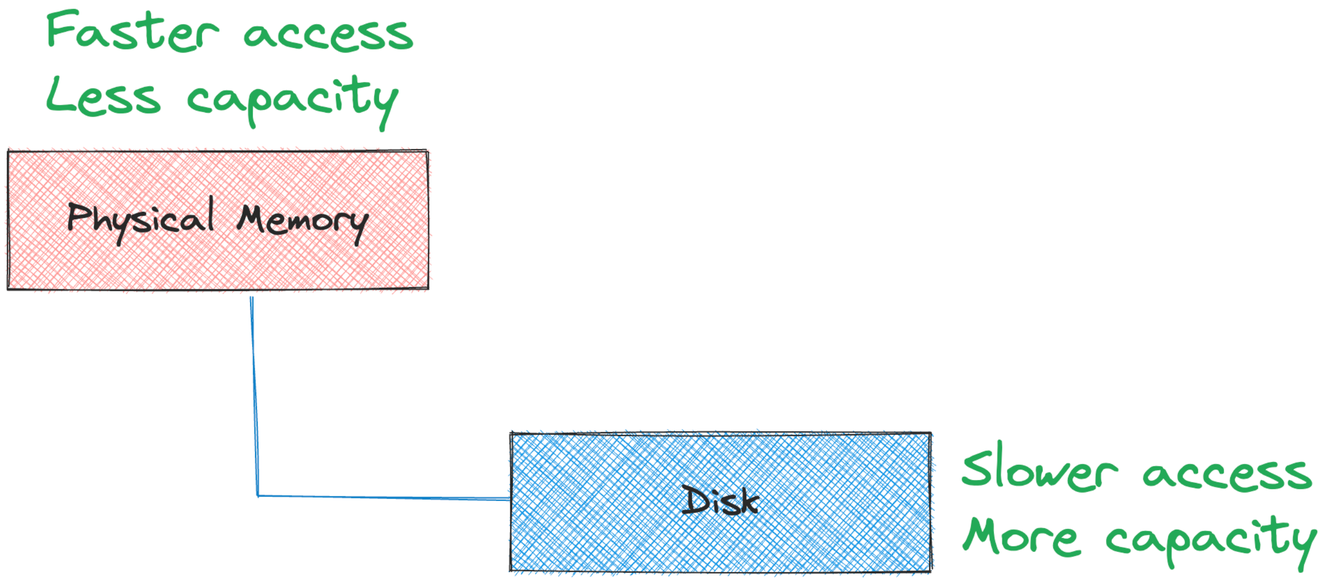
What about Memory Hierarchy?

- 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?

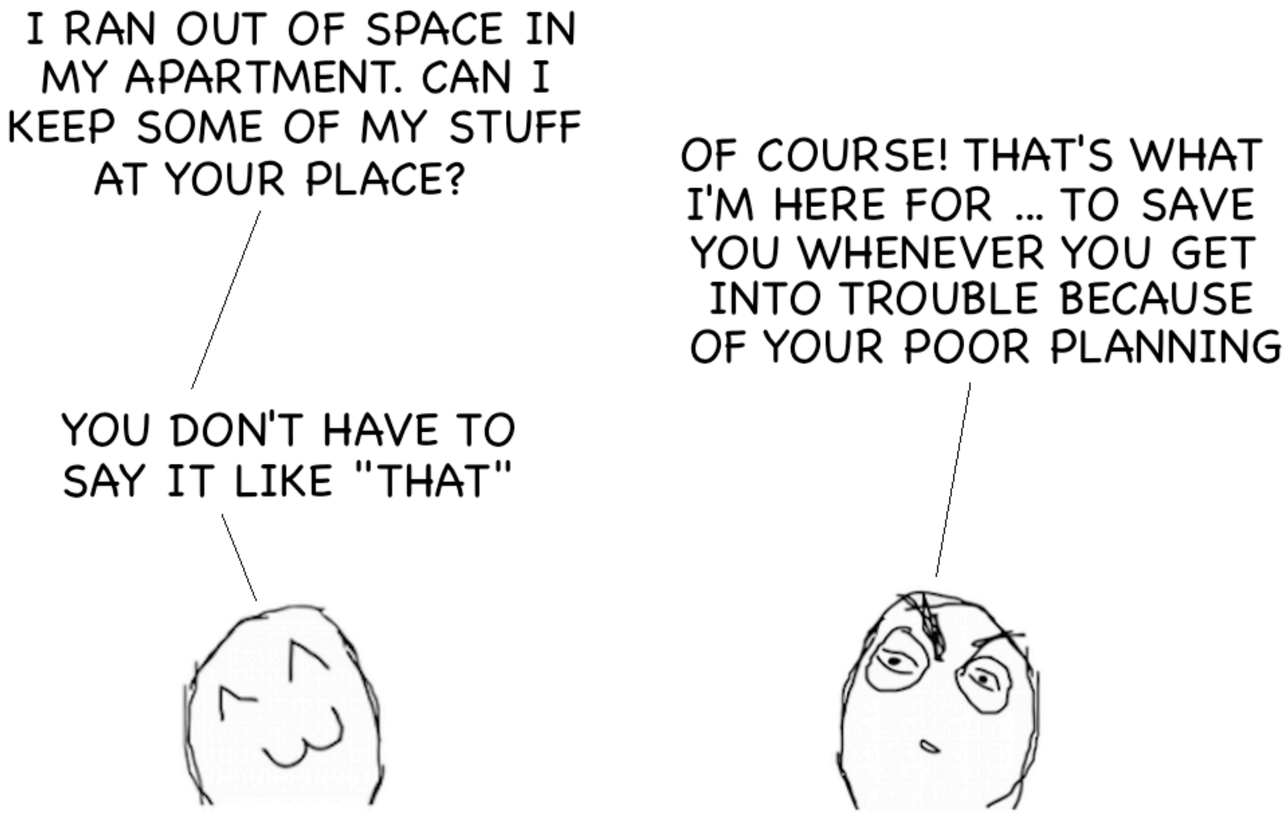
Moving between Physical Memory and Disk

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!!

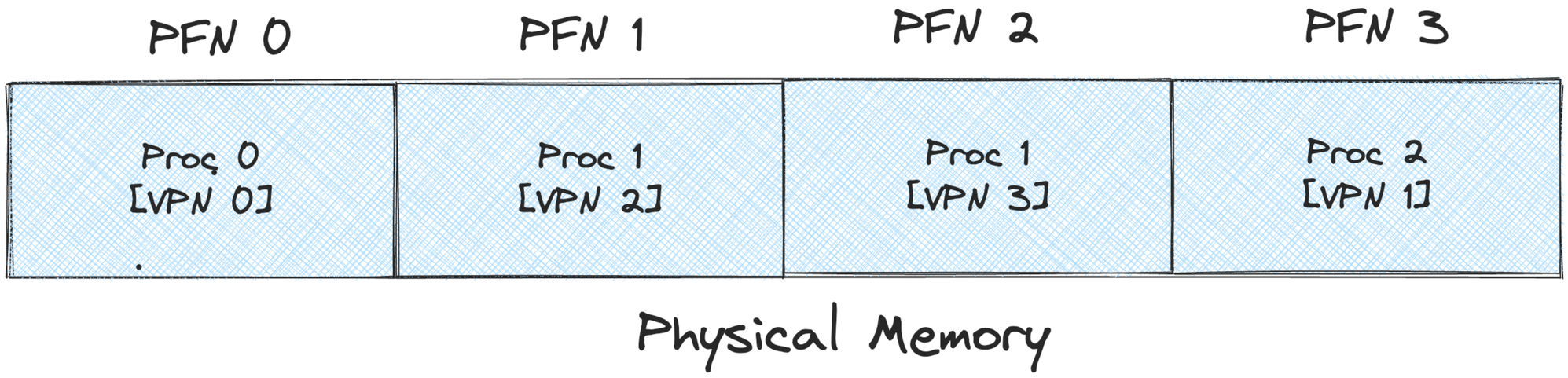


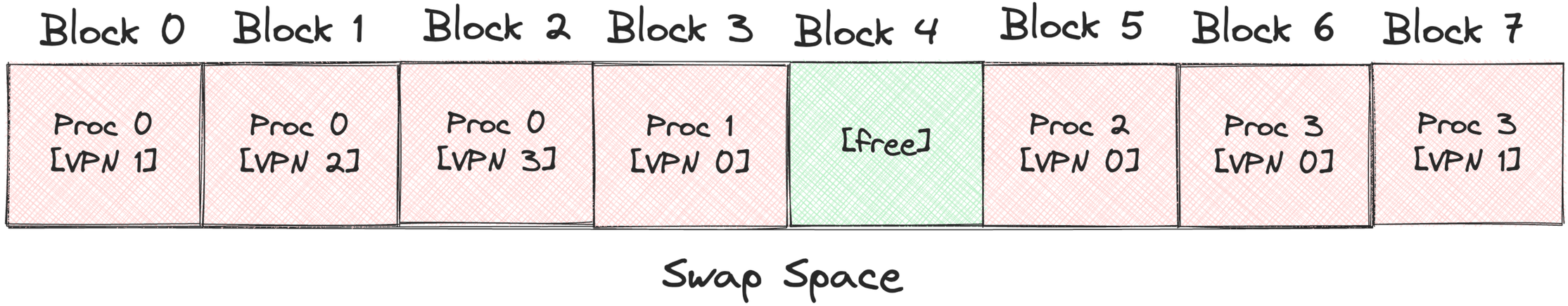
https://prateekvjoshi.com/2015/03/21/how-to-add-swap-space-on-ubuntu/

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

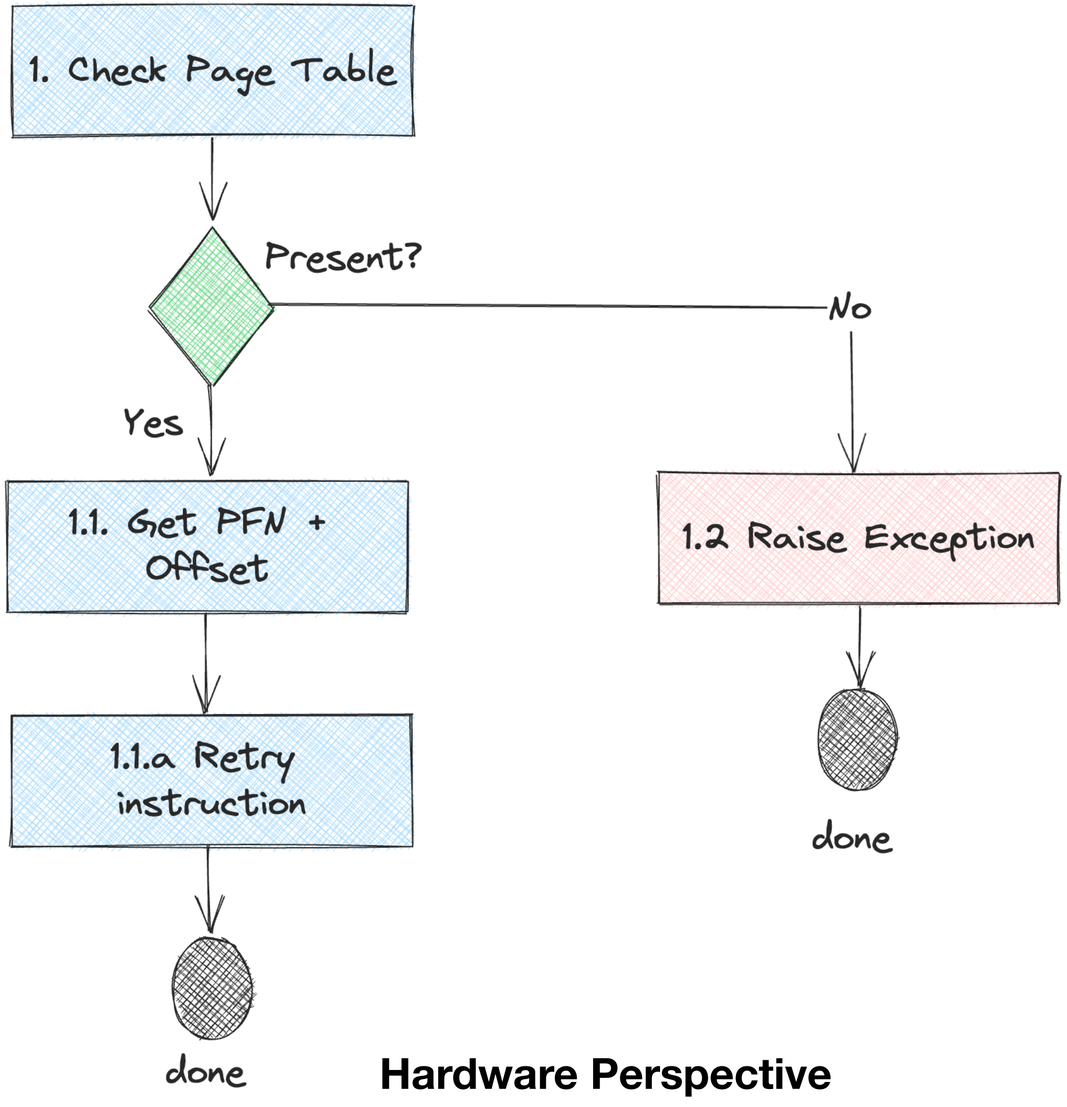
Page Fault

- 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?

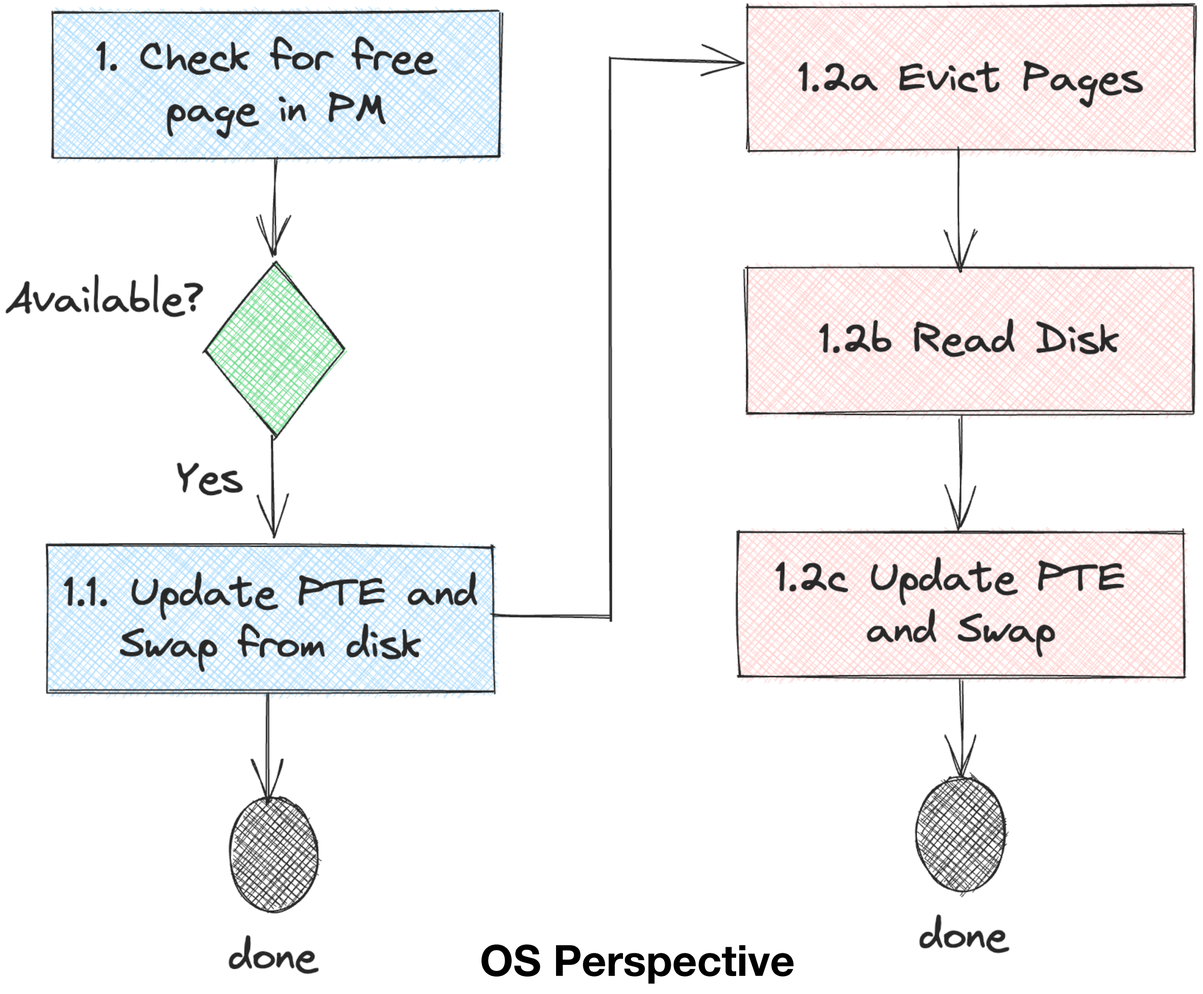
Page Fault and Context Switches

- 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

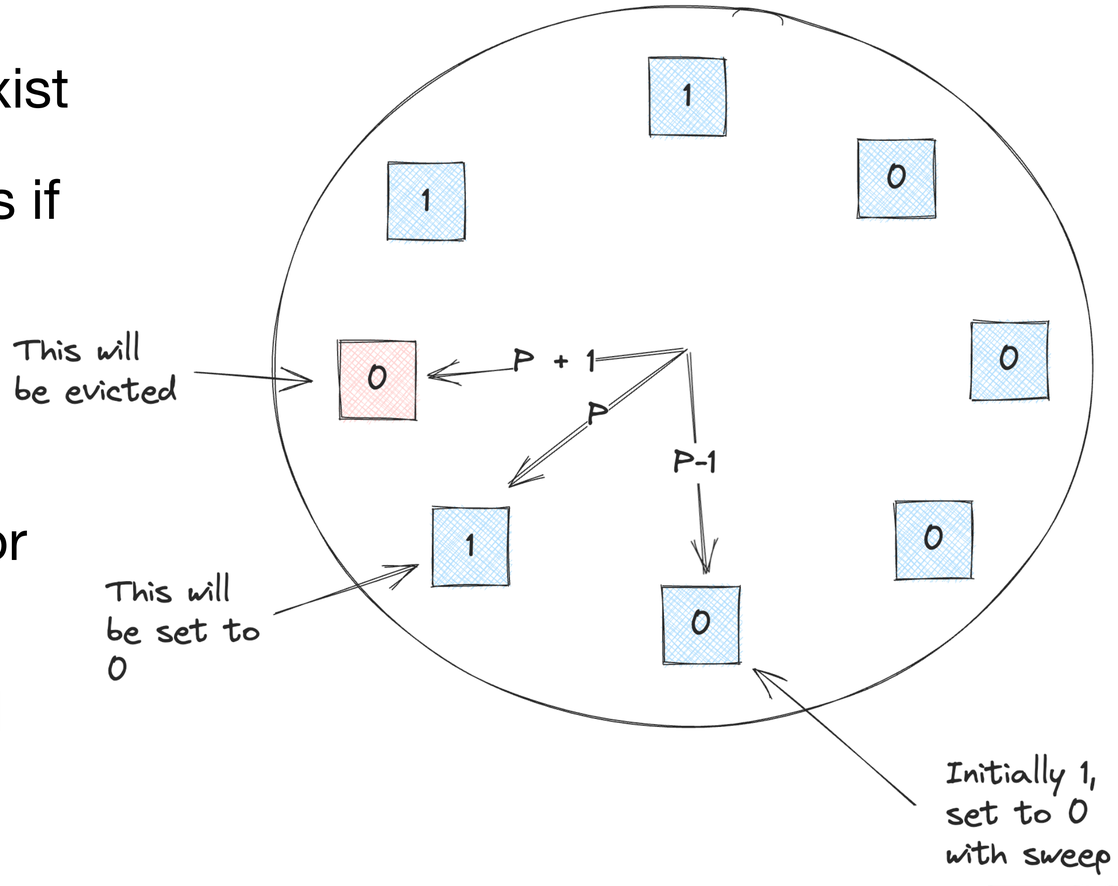
Hits: 6/11

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**

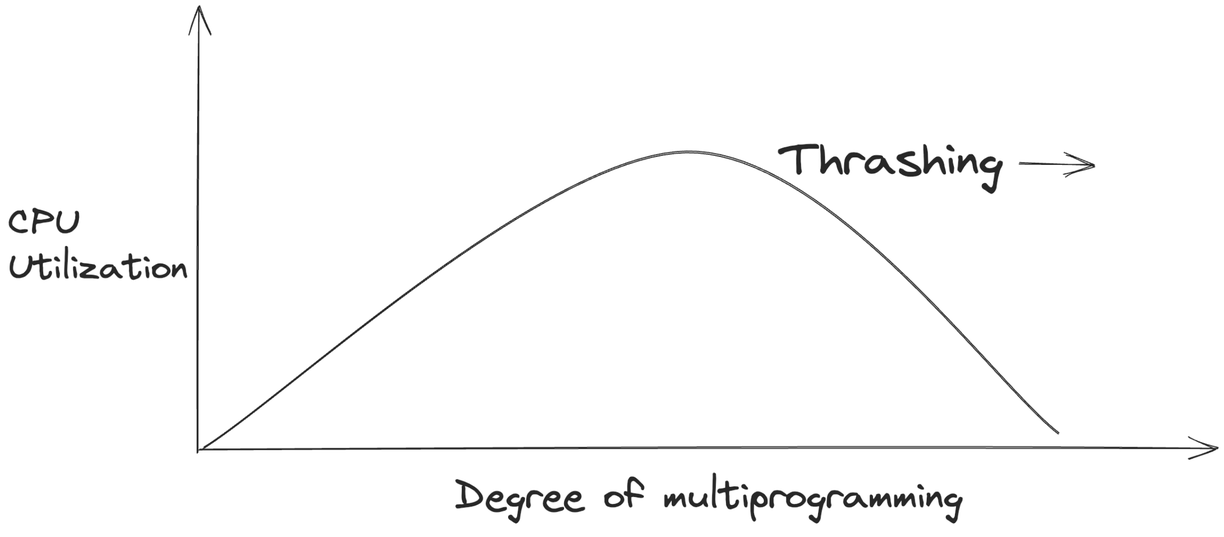


Implementing LRU

- 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