CS3.301 Operating Systems and Networks

Virtualization - Process

Karthik Vaidhyanathan

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
H Y D E R A B A D

SERC
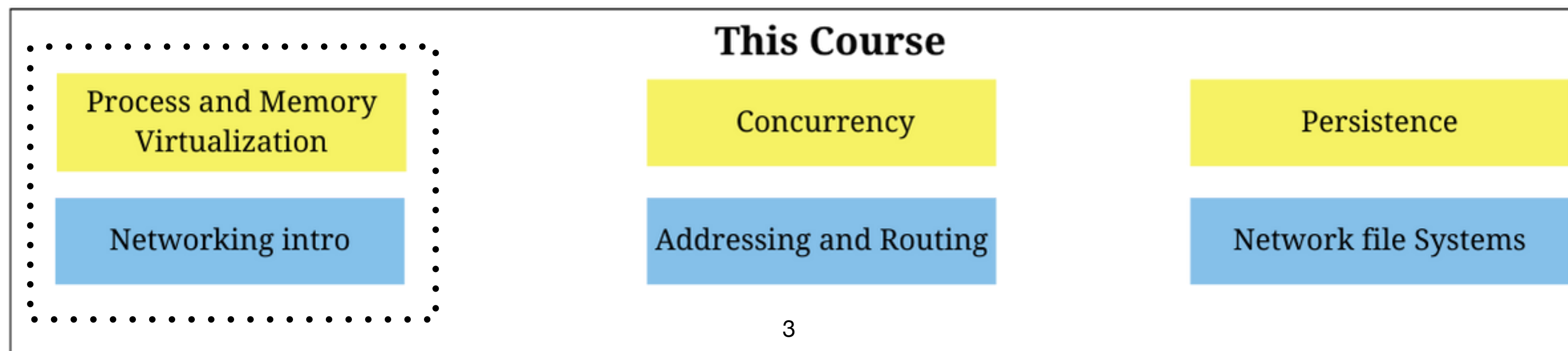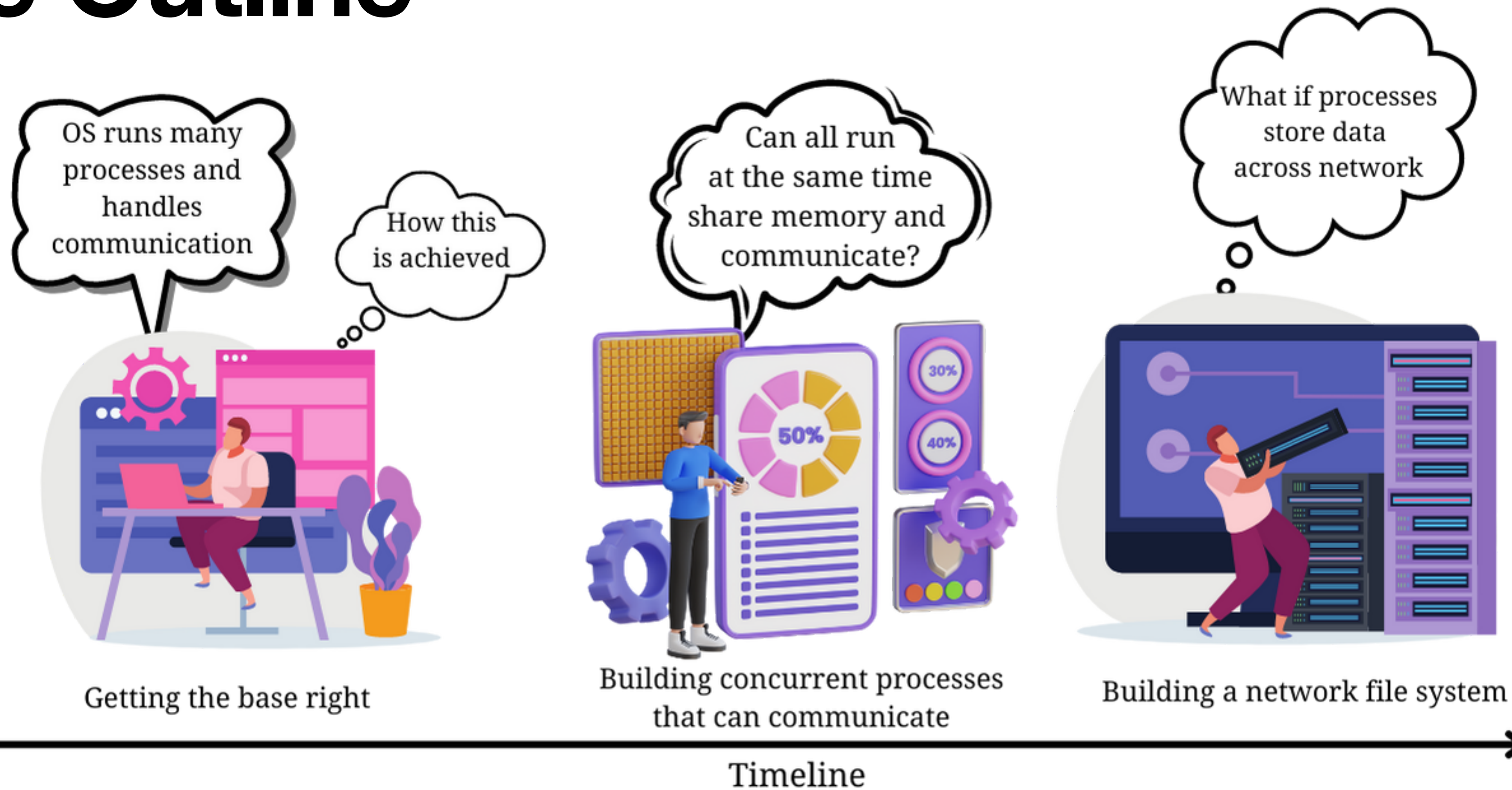Software Engineering Research Centre
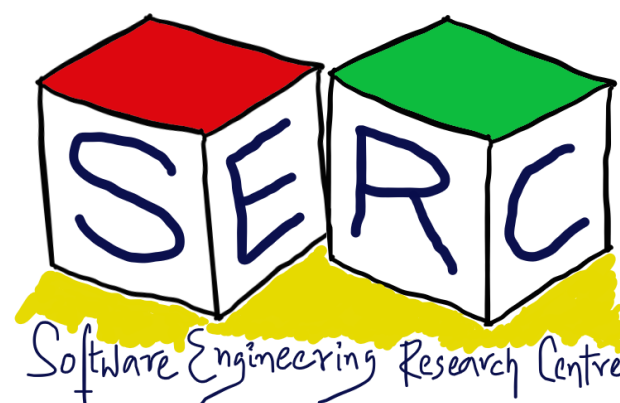
Acknowledgement

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

Sources:
- OSTEP Educator Materials, Remzi et al.
- OSTEP Book by Renzi et al.
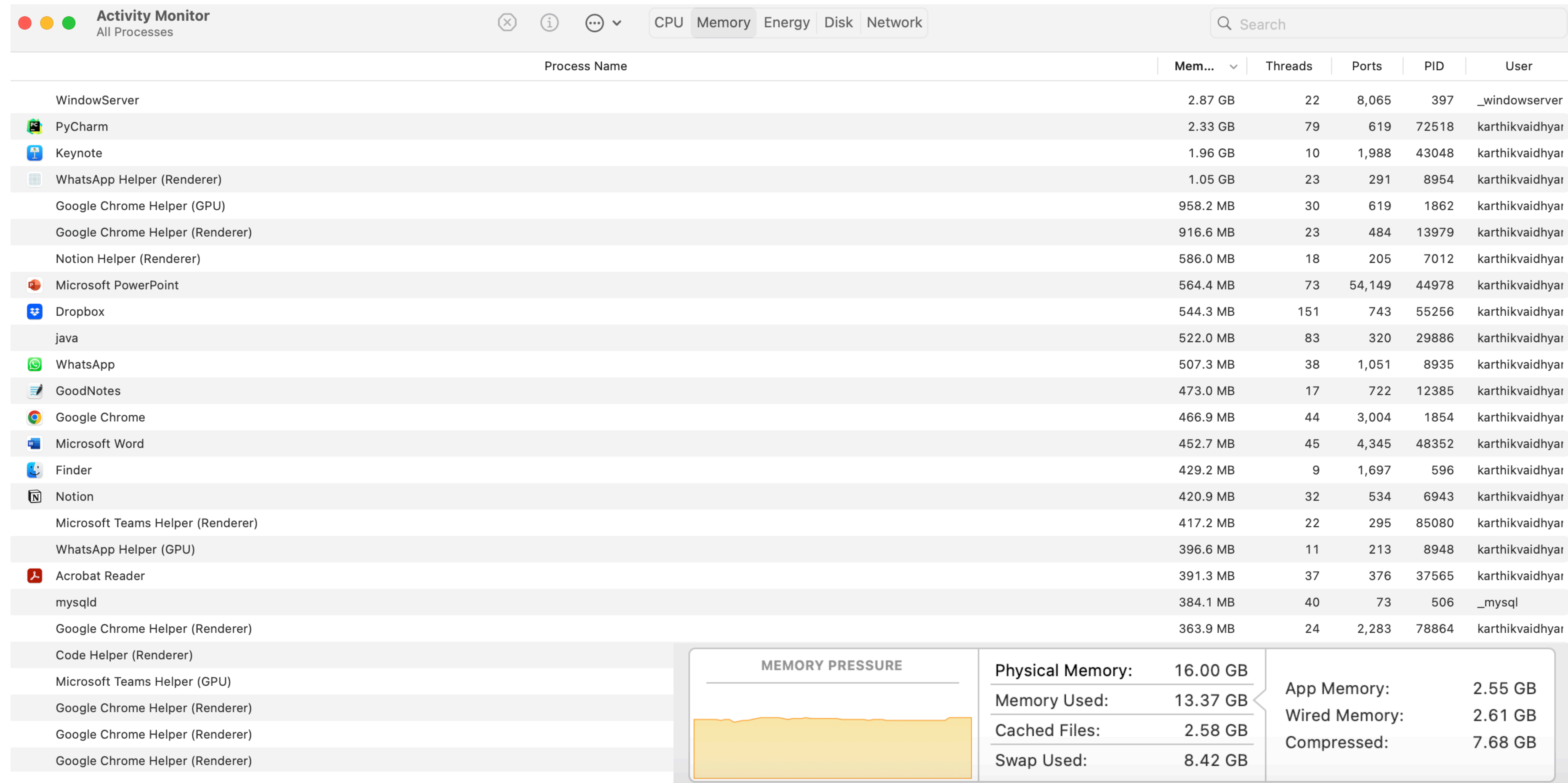- Modern Operating Systems, Tanenbaum et al.

Course Outline

Getting the base right

Building concurrent processes that can communicate

Building a network file system

Timeline

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?

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

| Program | Compile and Build → | Executable | Execute → | Running Process |

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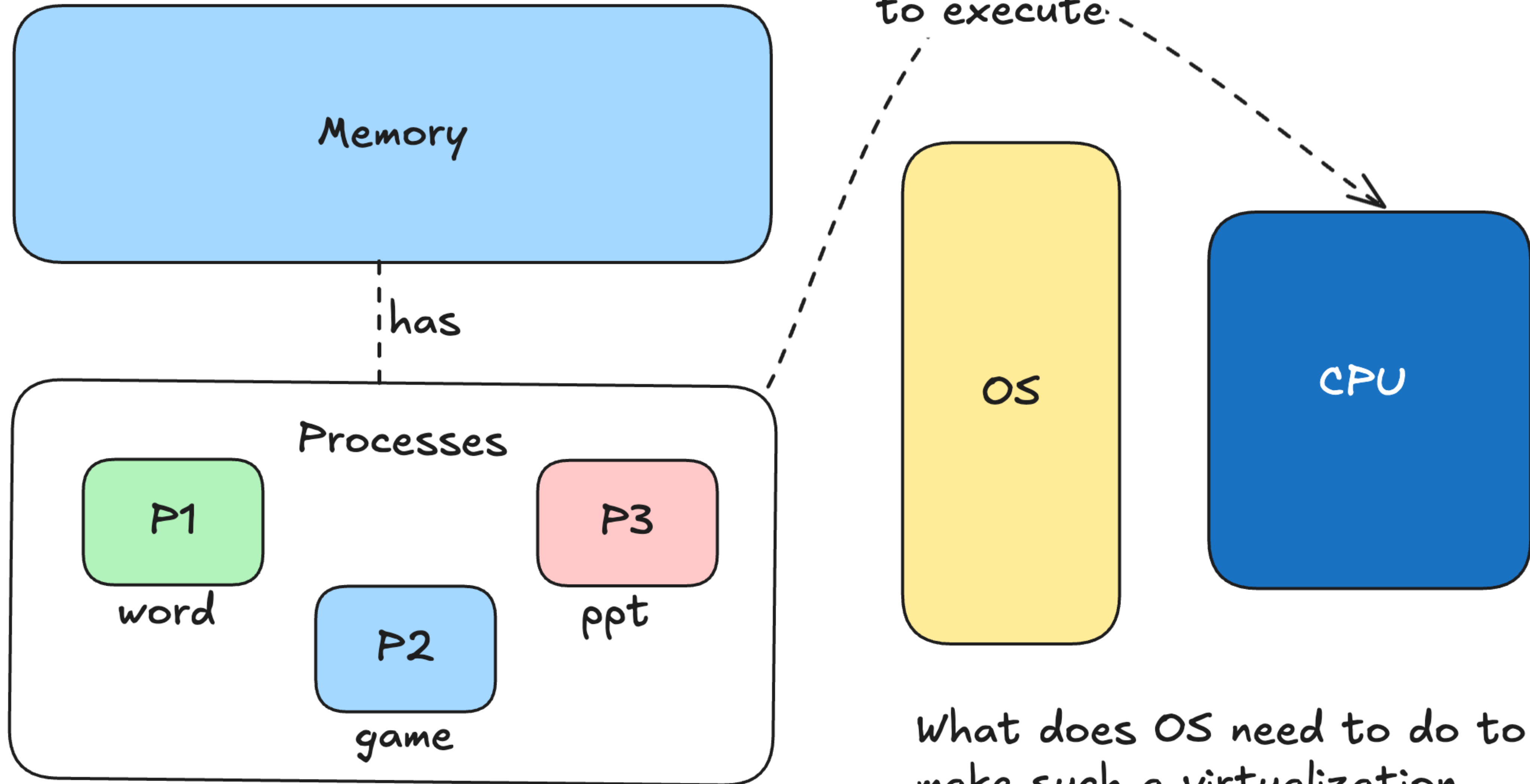
Each process is made to feel that it has its own CPU

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

Some Prerequisite

| CPU | Memory | Video Controller | Keyboard controller | USB Controller | HDD Controller |

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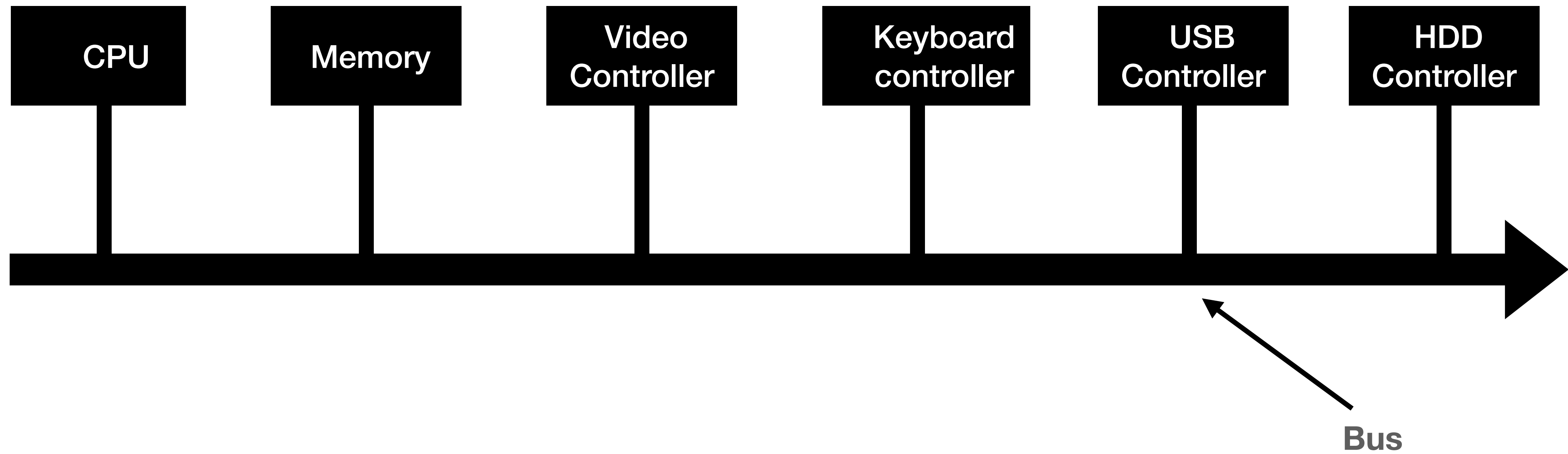
As we go more away from CPU, the more time it takes

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

```
Fetch  →  Decode  →  Execute
```
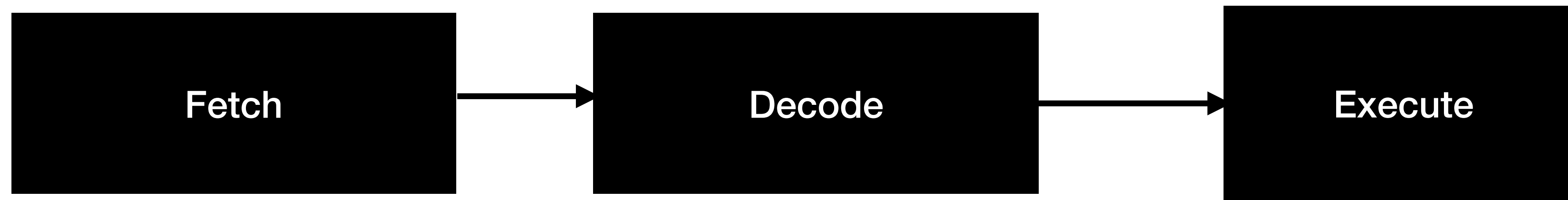
Question: Do you believe that the current hardware structure is similar to this?

Some Prerequisite - Computer Hardware
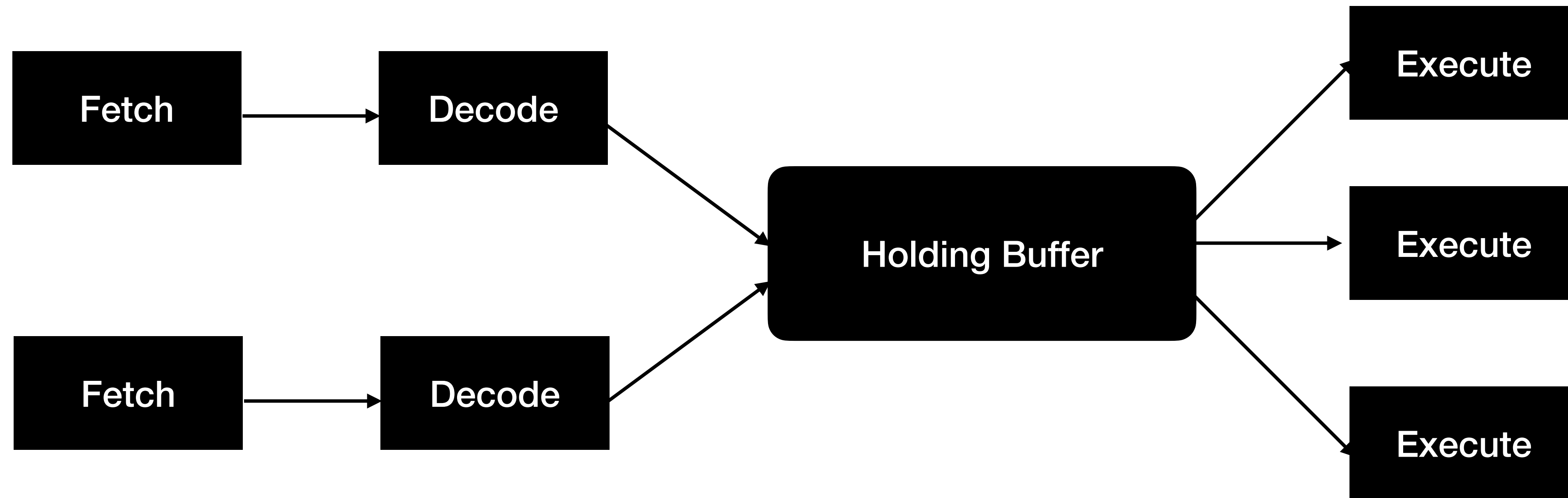
Superscalar CPU

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
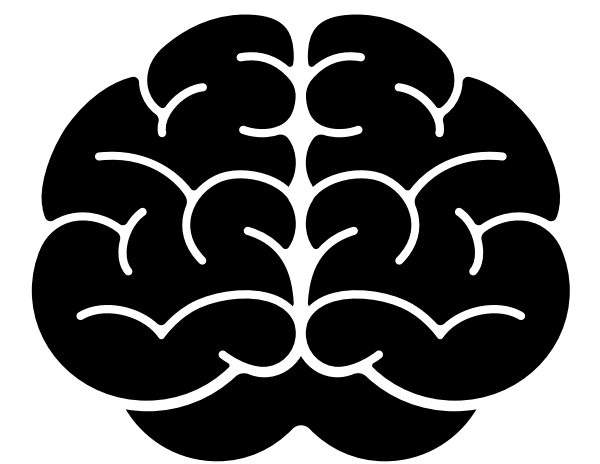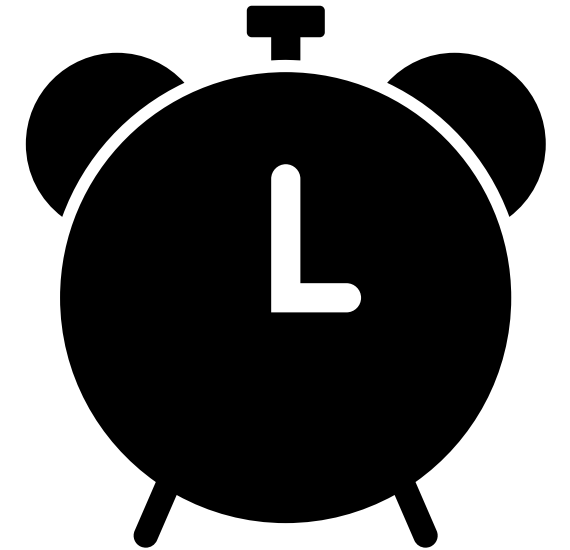
- Basically we need - **low level mechanisms** and **policies** (CPU Scheduler)
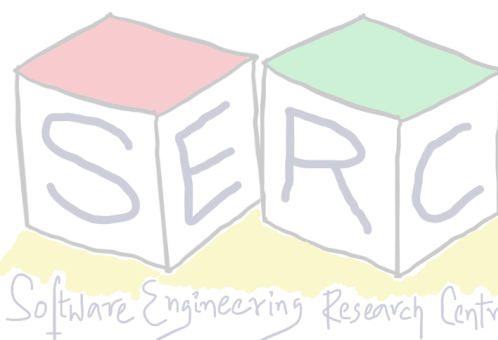
What Constitutes a Process?

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

Machine State

Registers

Instruction Pointer

Stack Pointer

Persistent Storage
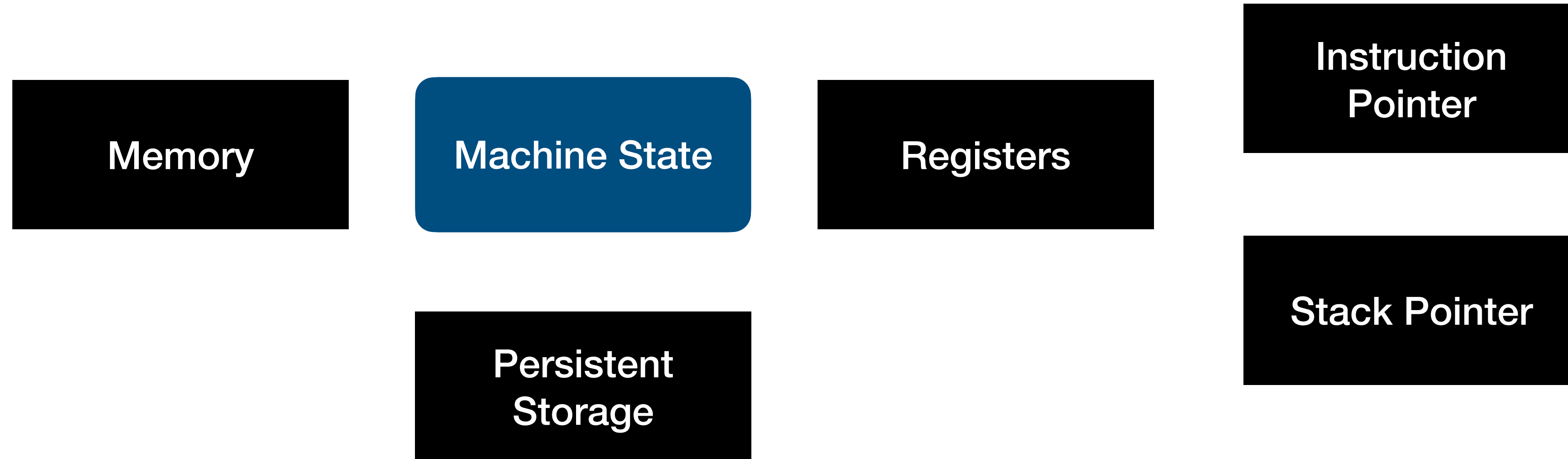
- **Memory** - address space (Memory that the process can address)

- **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

Code
Data
Stack
Heap
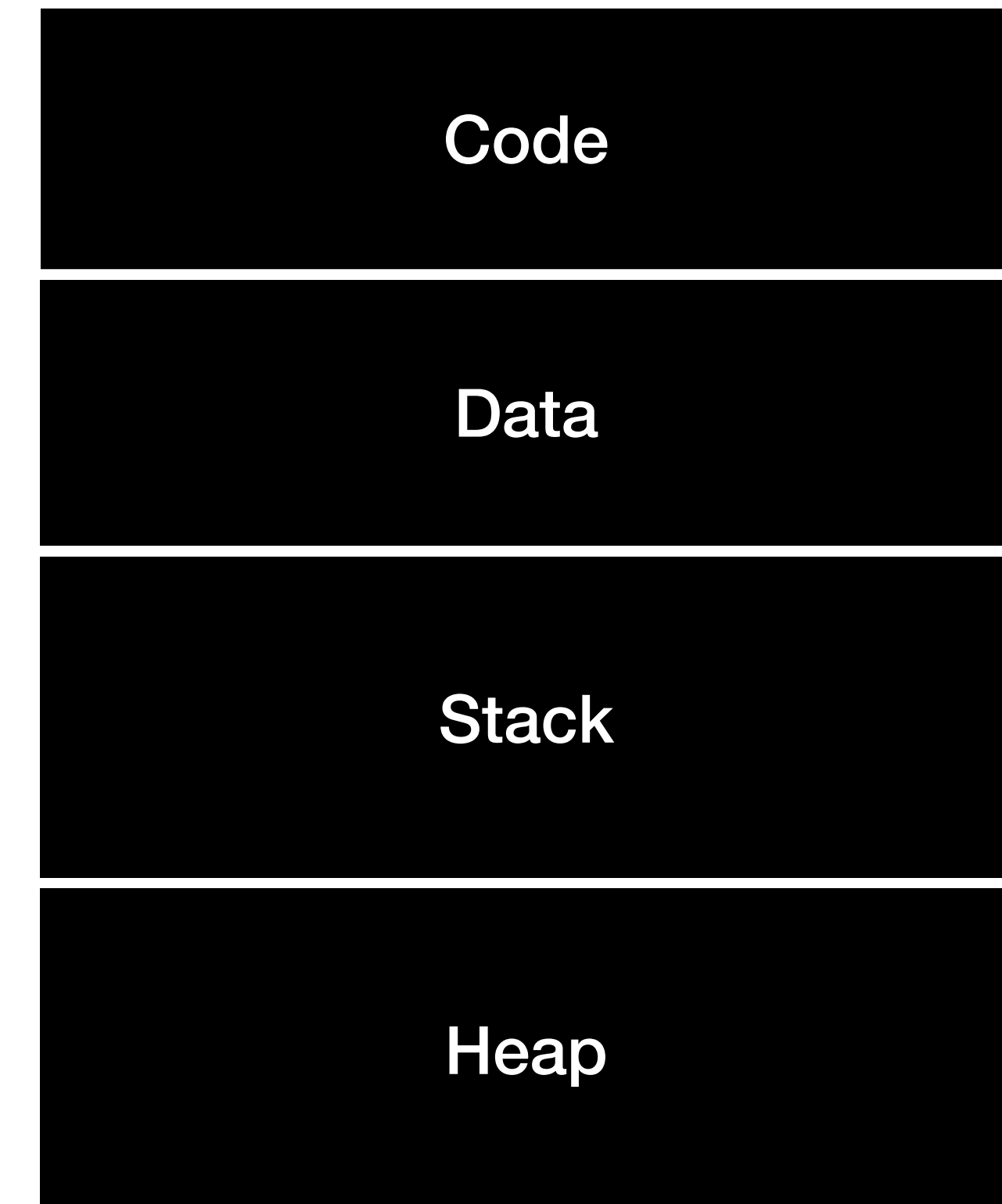
Creation of a Process by OS

- **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

CPU

Memory

code
static data
heap

stack

Process

code
static data
heap

Program

Disk

Loading:
Takes on-disk program and reads it into the address space of process

16
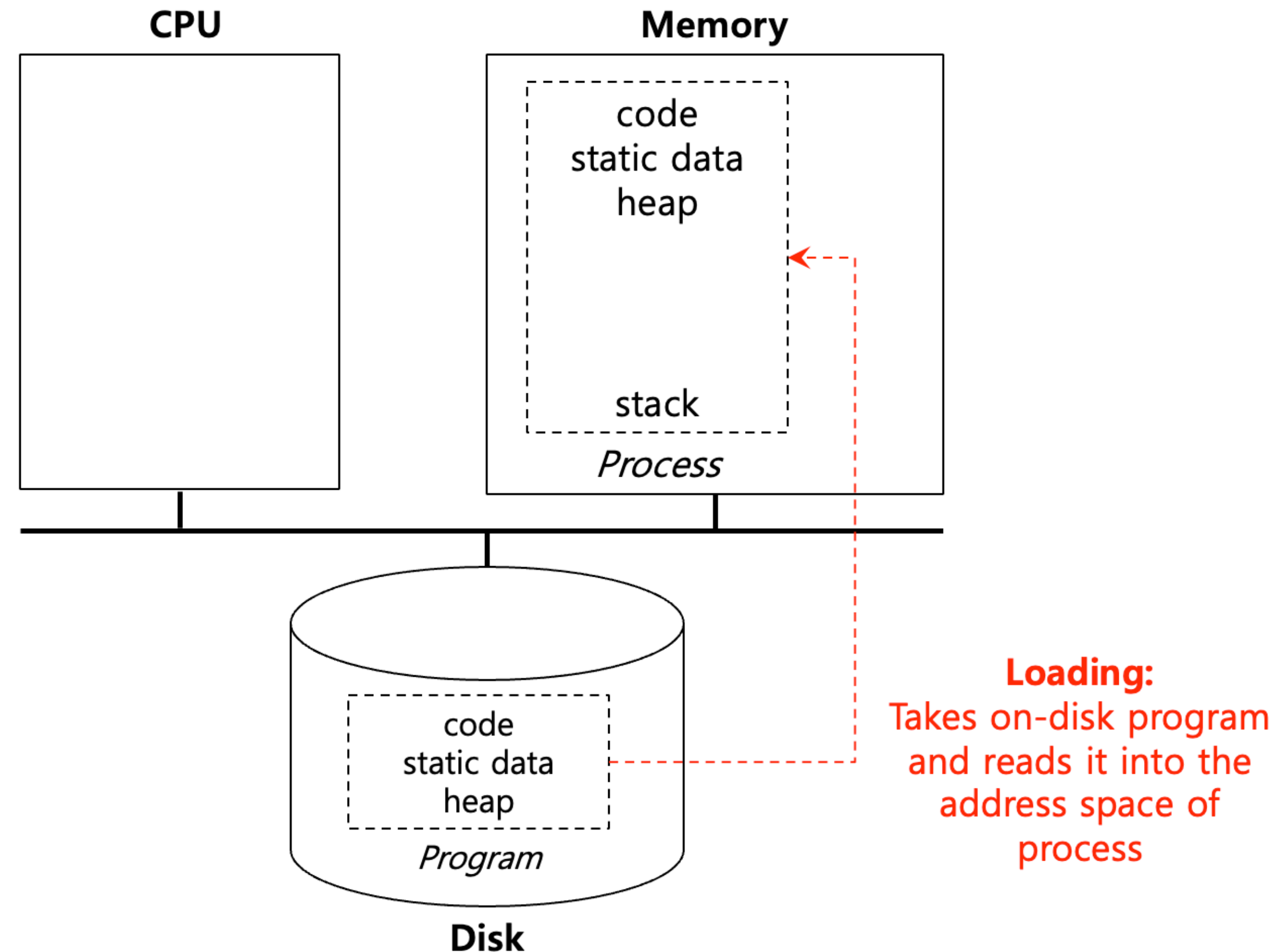
Creation of a Process by OS

- **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**

Loading:
Takes on-disk program and reads it into the address space of process

Image from OS lecture by Youjip Won
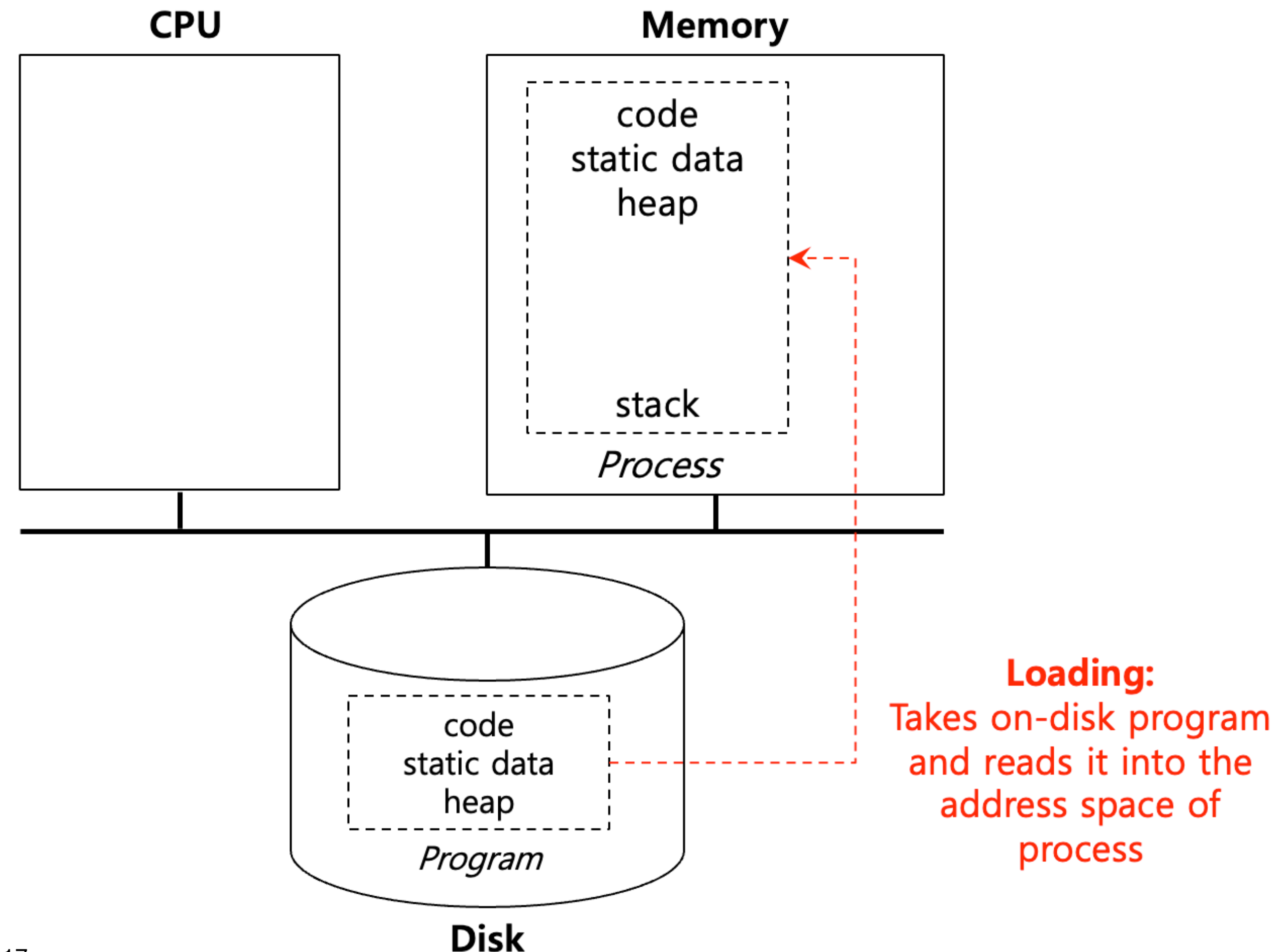
States of the Process

- 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
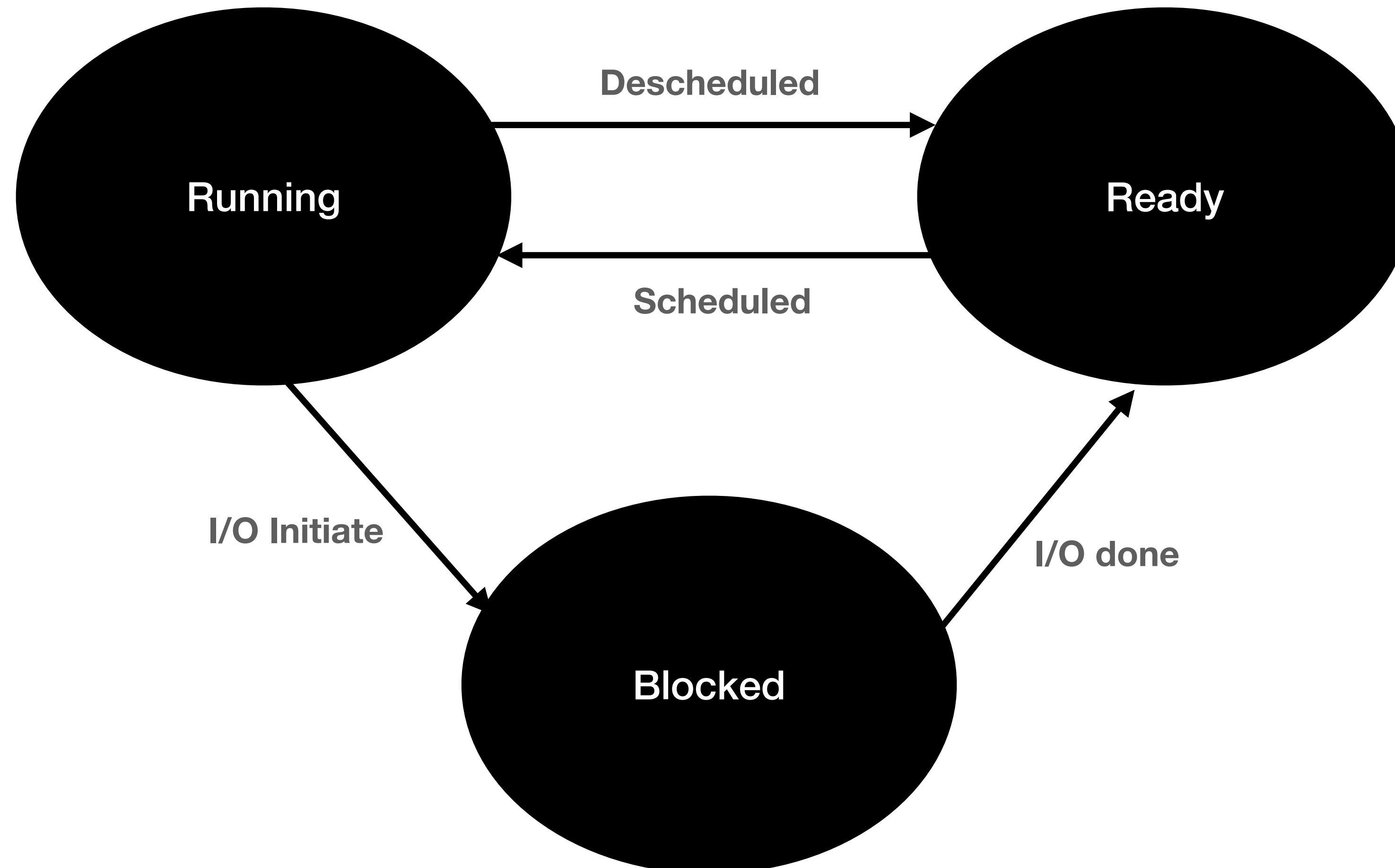
Lets look at an Example

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

P1	P2	P3	...

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

```c
// 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

```c
// 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 };
```

Thank you

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