Introduction

COL331

Abhilash Jindal

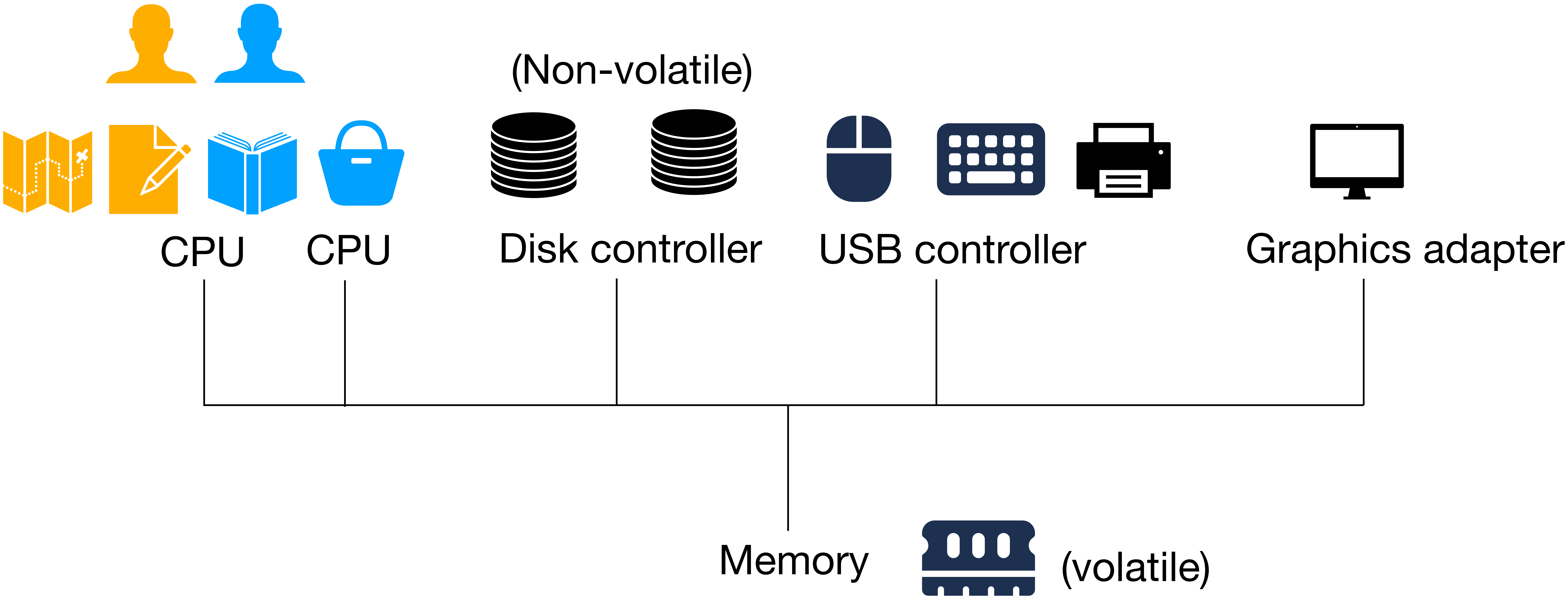
Reference. OSTEP book: Chapter 2

Administrivia

- http://abhilash-jindal.com/teach.html
- Grading criteria, TAs, late policy, audit criteria, quizzes, labs, project, piazza link

Why does OS matter to a computer?

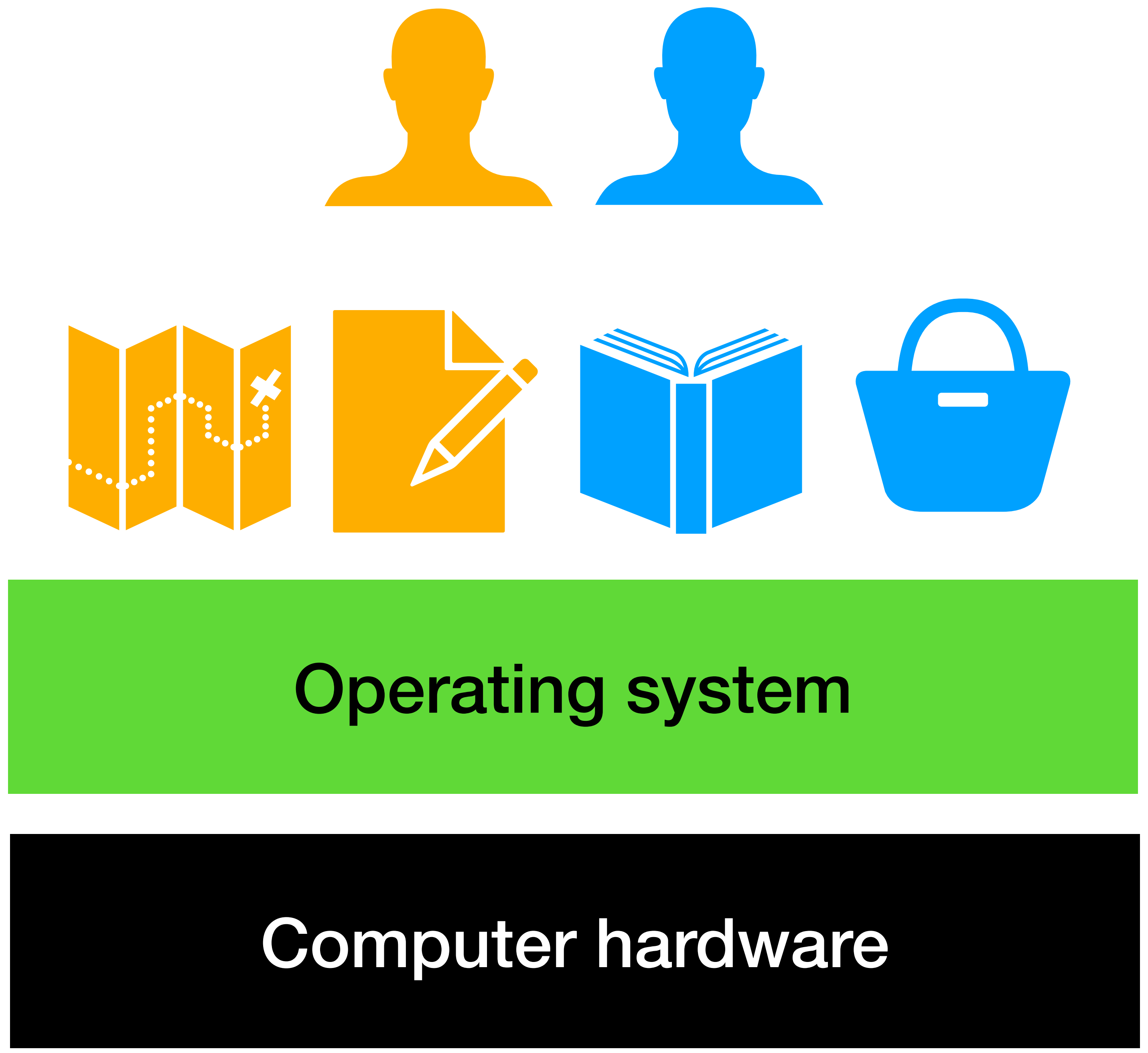
Computer organization



Purpose of an OS

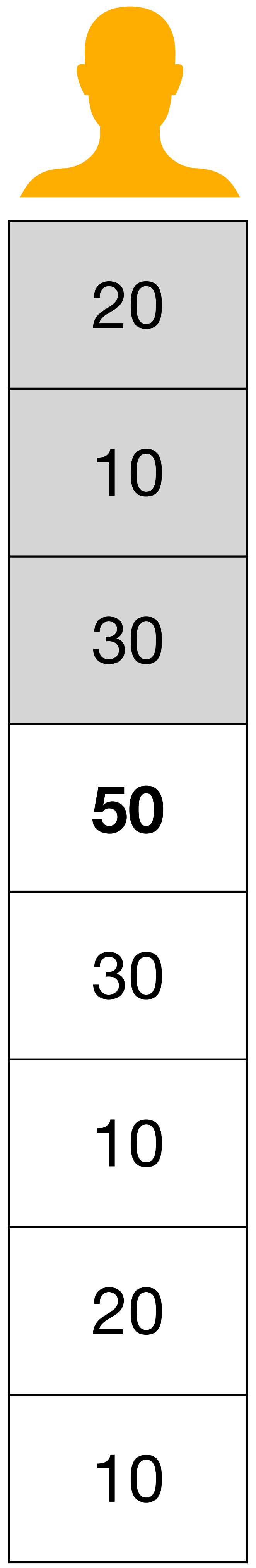
- **Resource management**
- Provide higher-level services
- Protection and isolation

Purpose of OS: Resource management



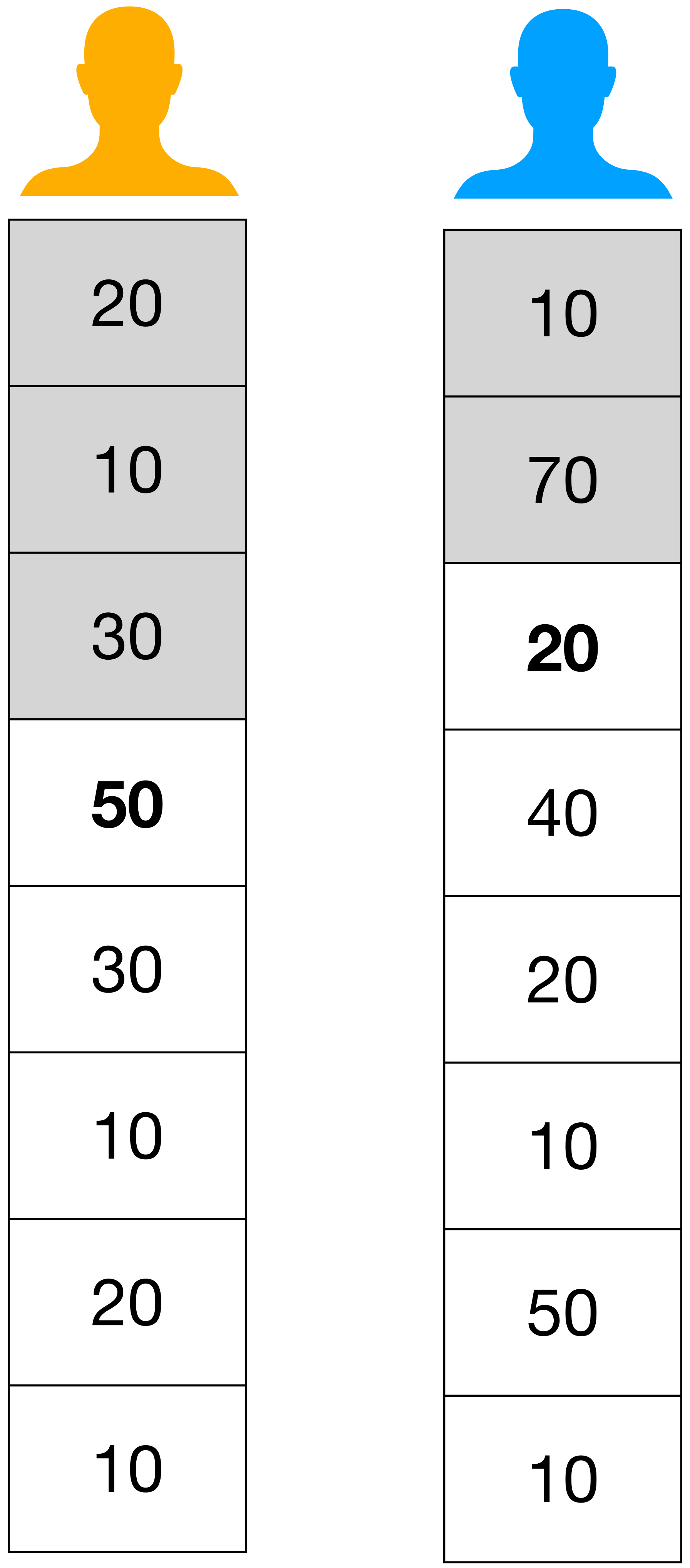
- Example: cpu.c
- Give the illusion of more CPUs than there are
- Multiplex the hardware

Calculator analogy: Computing long sum



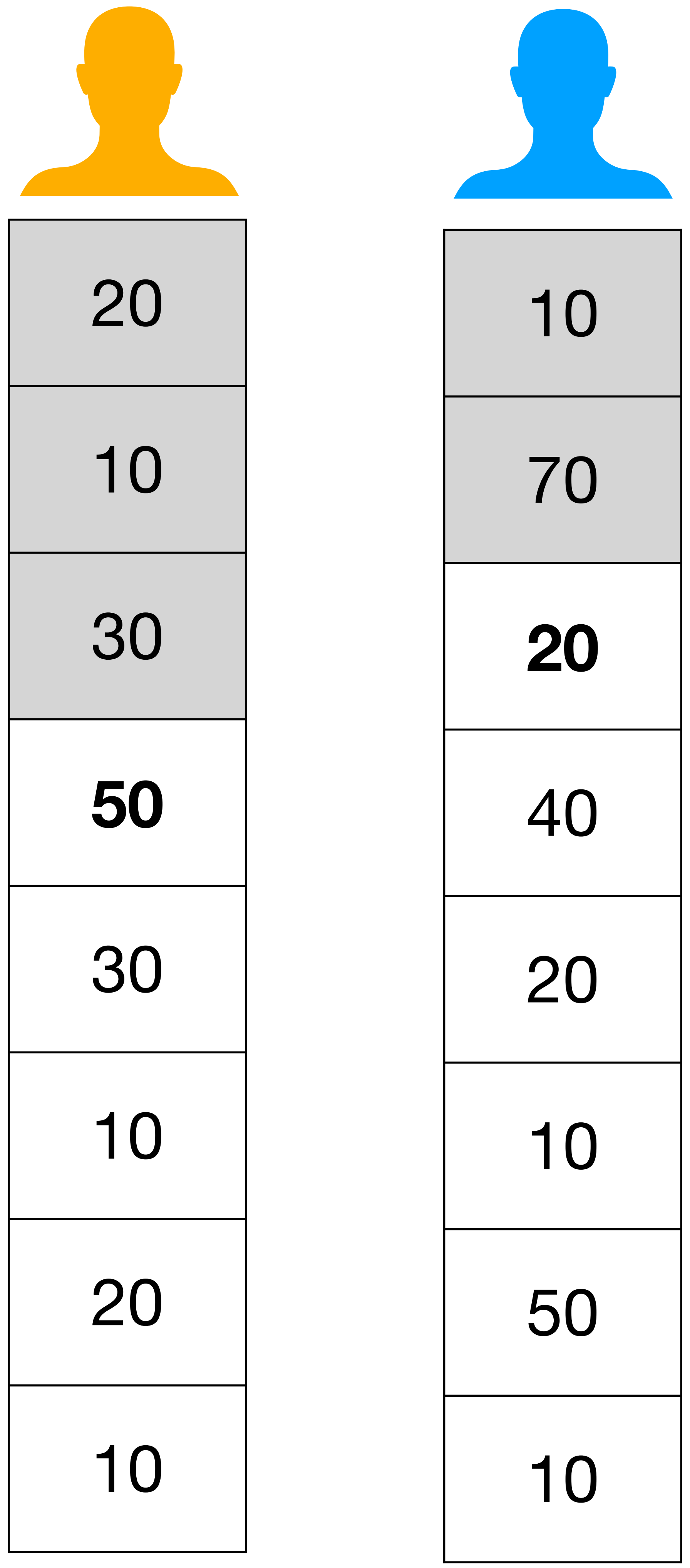
- 2 0 = (move pointer to 10)
- \+ 1 0 = (move pointer to 30)
- \+ 3 0 = (move pointer to 50)
- \+ 5 0 = (move pointer to 30)
- \+ 3 0 = (move pointer to 10)
- \+ 1 0 = (move pointer to 20)
- \+ 2 0 = (move pointer to 10)

Sharing the calculator



- Steps to share the calculator:
  - 20 + 10 = 30 + 30 = 60
  - Write 60 in notebook, remember that we were done till 30, give calculator
  - 10 + 70 = 80
  - Write 80 in notebook, remember that we were done till 70, give the calculator back

Remember whatever is on the screen and give calculator?



- 2 0 = (move pointer to 10)
- + 1 0 = (move pointer to 30)
- + 3 0 = (move pointer to 50)
- + 5 0 = (move pointer to 30)
- + 3 0 = (move pointer to 10)
- + 1 0 = (move pointer to 20)
- + 2 0 = (move pointer to 10)

Can I give calculator here?

“Save 1 and remember pointer to be at 10”

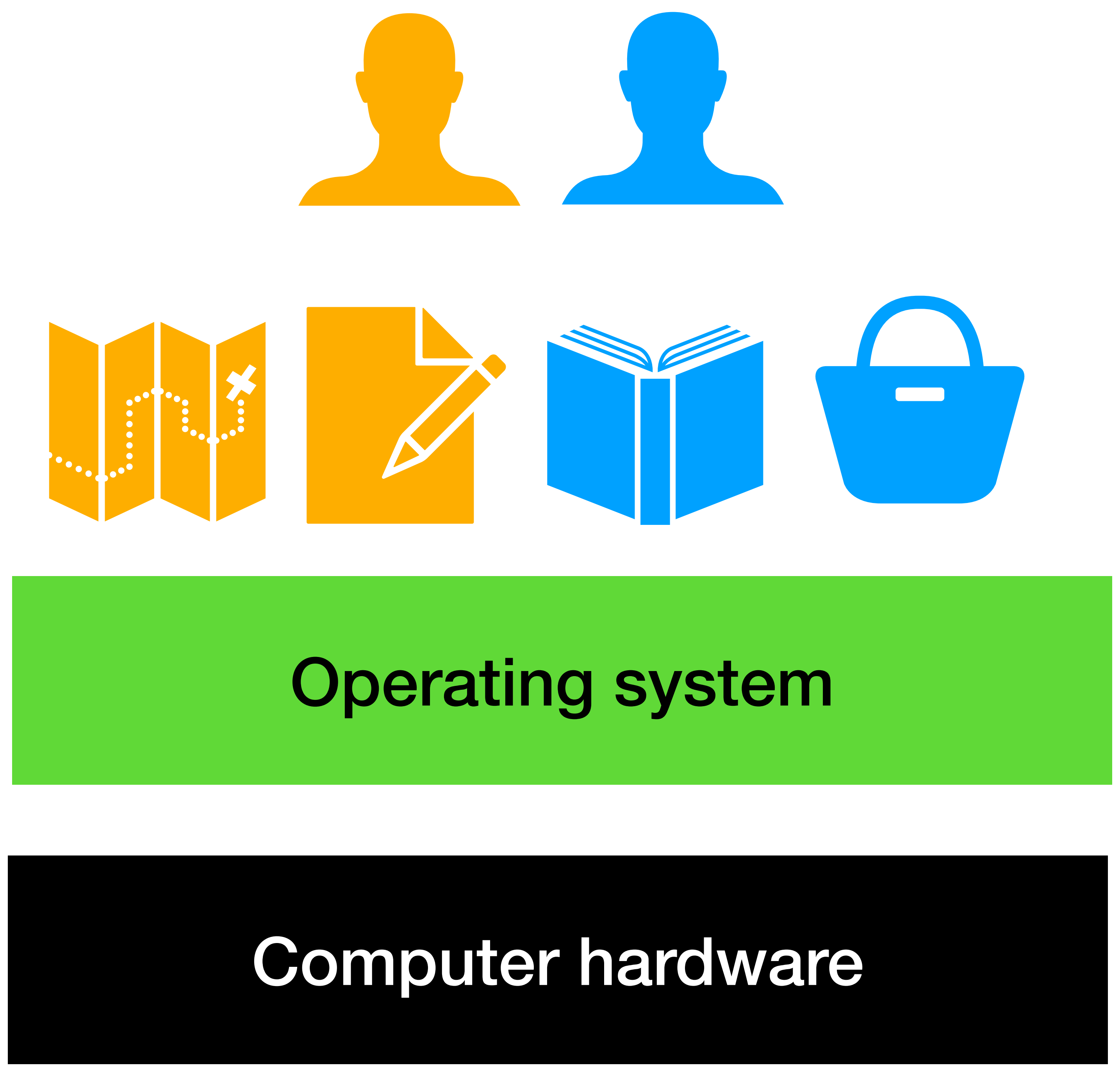
No! Sum would be wrong!

“+ xx = (move pointer)” has to be atomic

Resource manager: multiplexing CPU

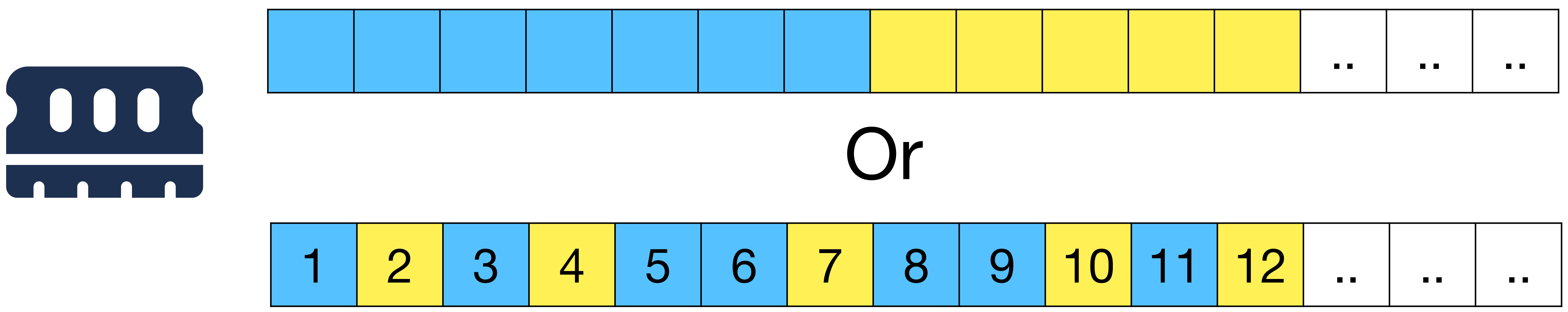
- CPU is also executing one instruction after another and incrementing “instruction pointer” *atomically*
- OS switches CPU between processes in the same manner as our calculator example (mechanism)
- What should happen when multiple processes want to run simultaneously? (policy)
    - Fairness: One banker got more calculator time than others
        - Often need to break away from fairness. Game should get more CPU time than Dropbox to provide good user experience
    - Starvation freedom: When there are multiple bankers, one banker never got the calculator

Purpose of OS: Resource manager



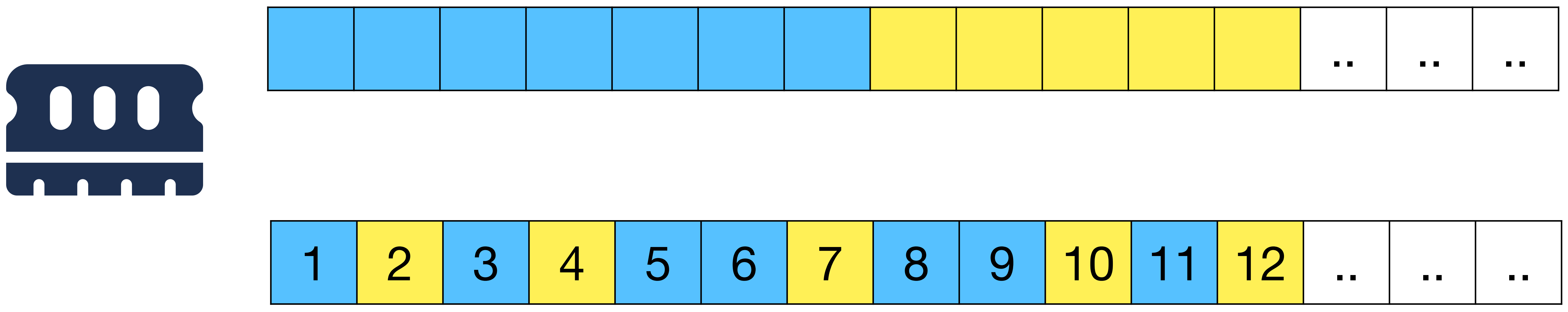
Different approaches to memory management:

- Segment different memory portions to different processes
- Multiplex memory pages across processes



Memory management

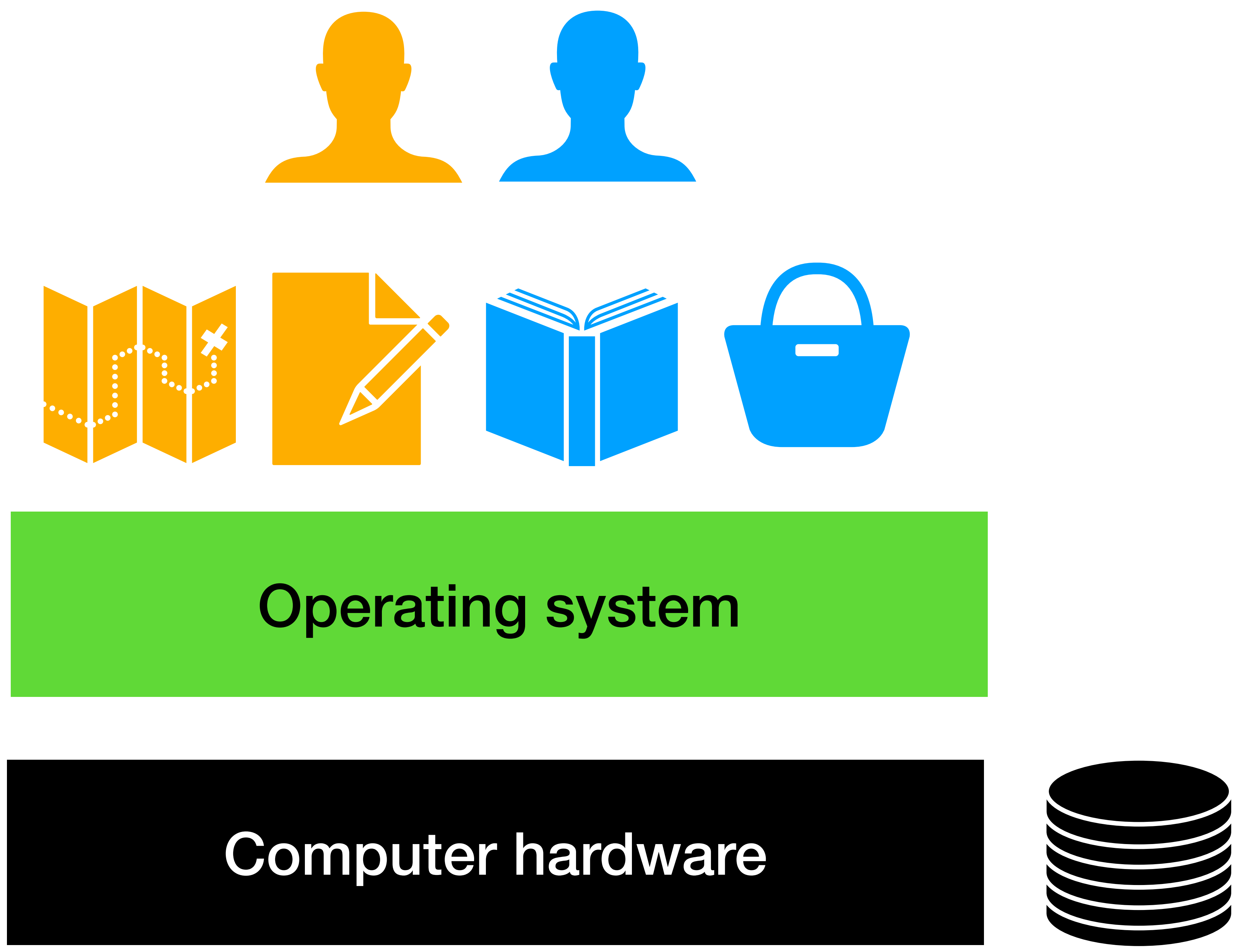
- Segmentation is cheap to implement
- But not flexible. What if a process needs more memory that what OS gave?
- Paging is complicated to implement
- Highly flexible



Purpose of an OS

- Resource management
- **Provide higher-level services**
- Protection and isolation

Purpose of OS: Provide higher-level services

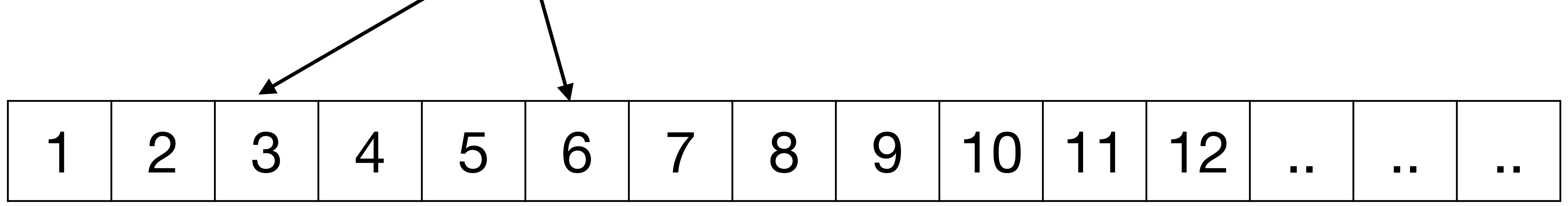


Example: io.c

Disk interface: List of blocks

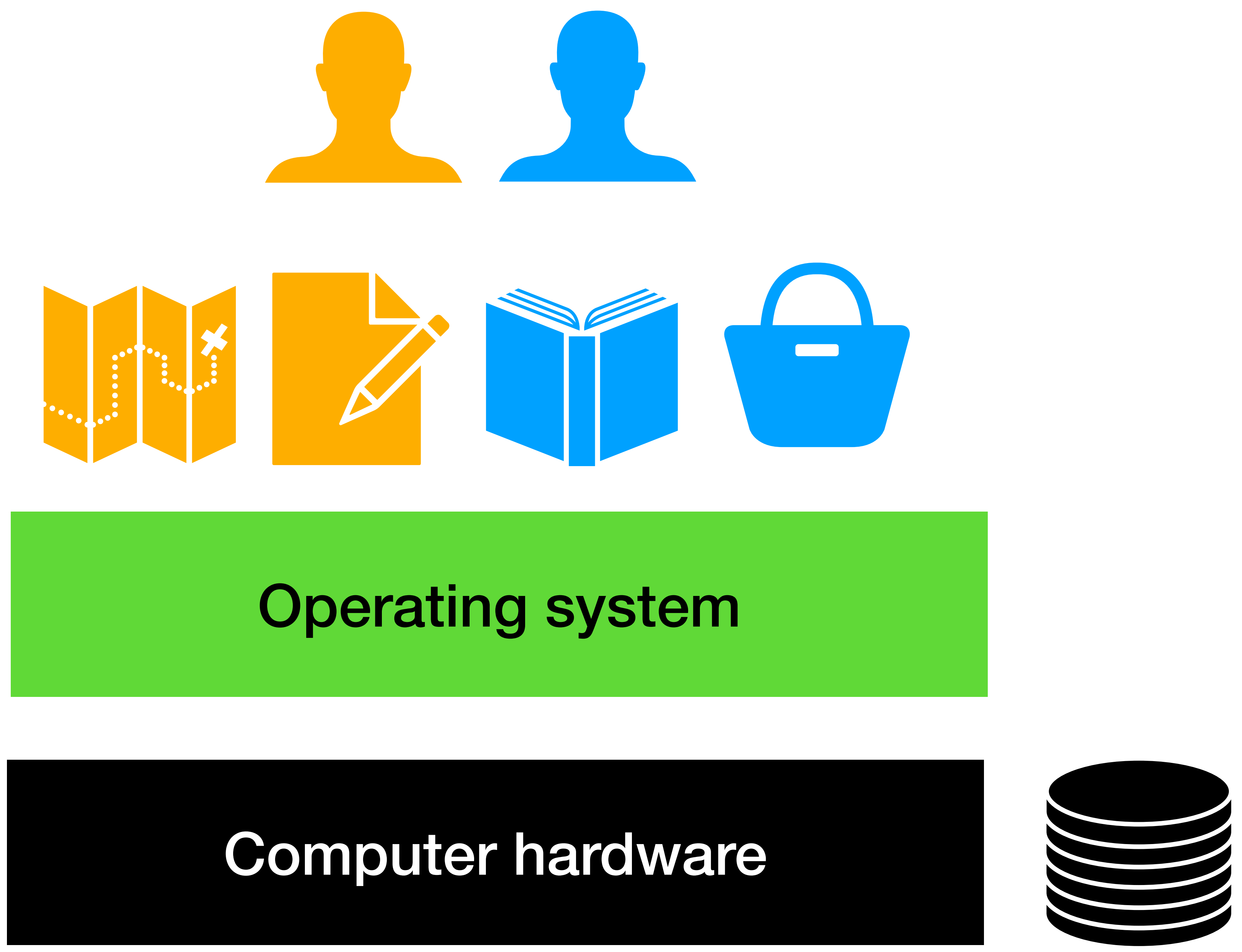
File system OS interface: Folders and files

```
int fd = open("/tmp/file", O_WRONLY | O_CREAT);
int rc = write(fd, "hello world\n", 13);
close(fd);
```

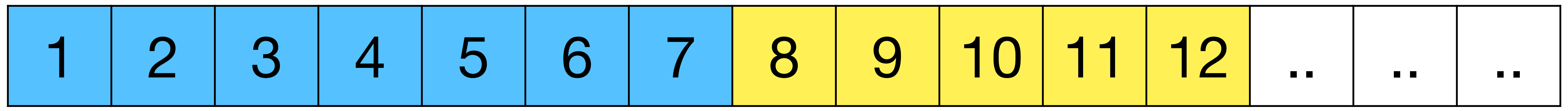


Why file system?

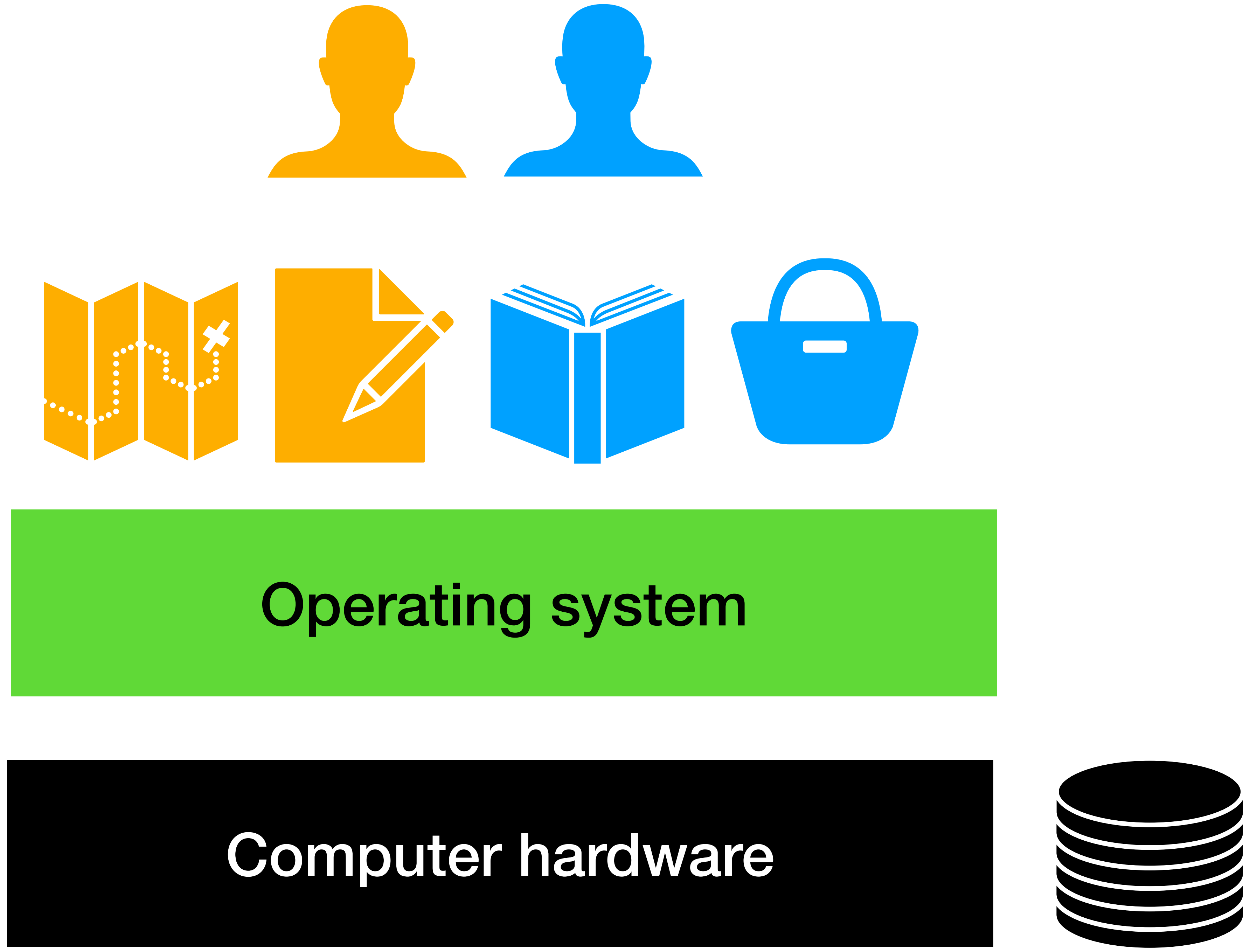
Why not just multiplex disk blocks like memory?



- Disk blocks live after programs exits, computer restarts
- Different programs read / write same file
  - vim writes io.c
  - gcc reads io.c, write io
  - We finally run io



Higher level services provide portability



- Abstract away hardware details
- Programs need not be rewritten when moving from hard-disk drive to solid state drive

```
int fd = open("/tmp/file", O_WRONLY | O_CREAT);
int rc = write(fd, "hello world\n", 13);
close(fd);
```

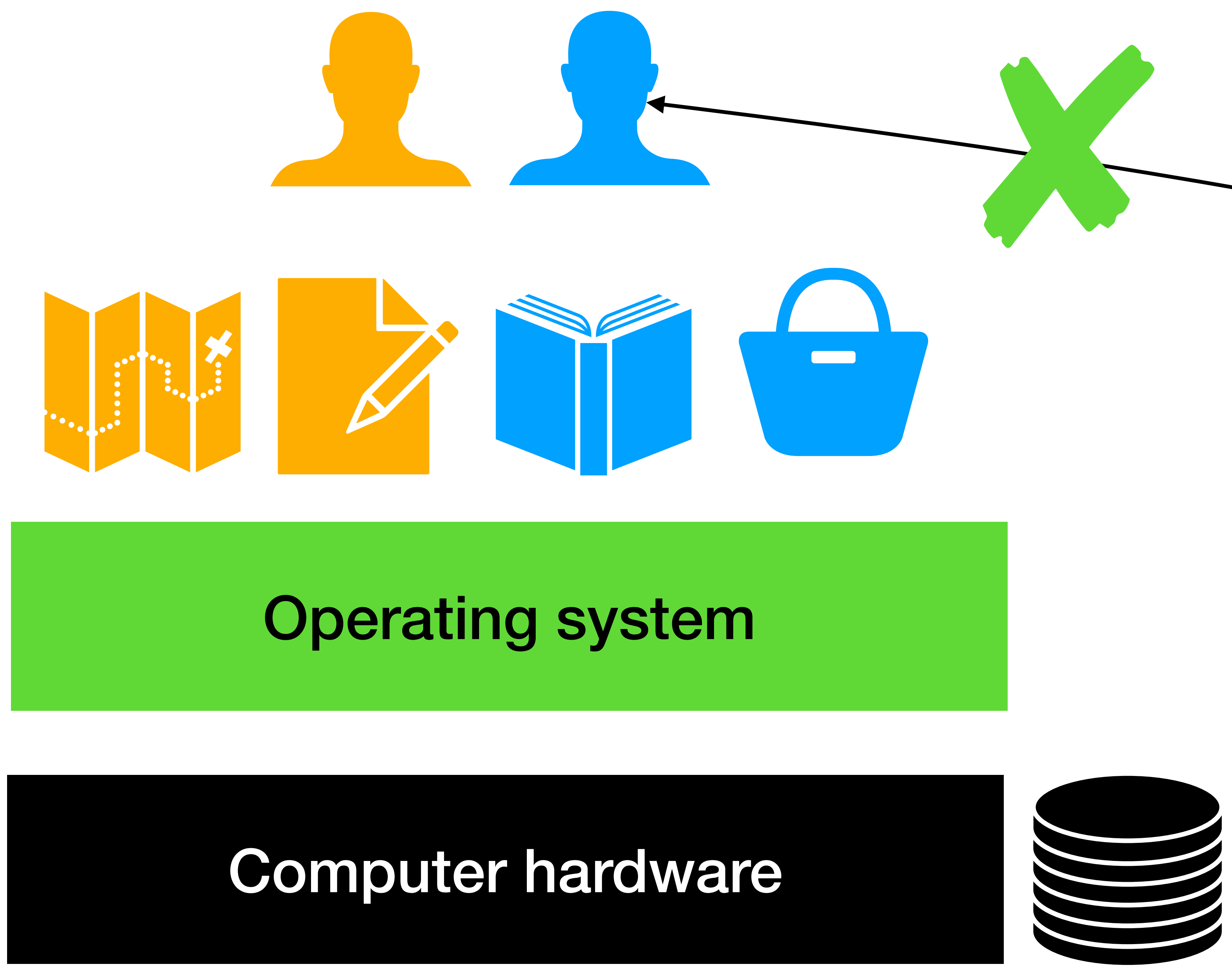
1	2	3	4	5	6	7	8	9	10	11	12	..	..	..

Purpose of an OS

- Resource management
- Provide higher-level services
- **Protection and isolation**

Purpose of OS: Protection

- S_IRUSR | S_IWUSR: File can only be rw by user
- Disallow inappropriate accesses. Example: users reading each other's files

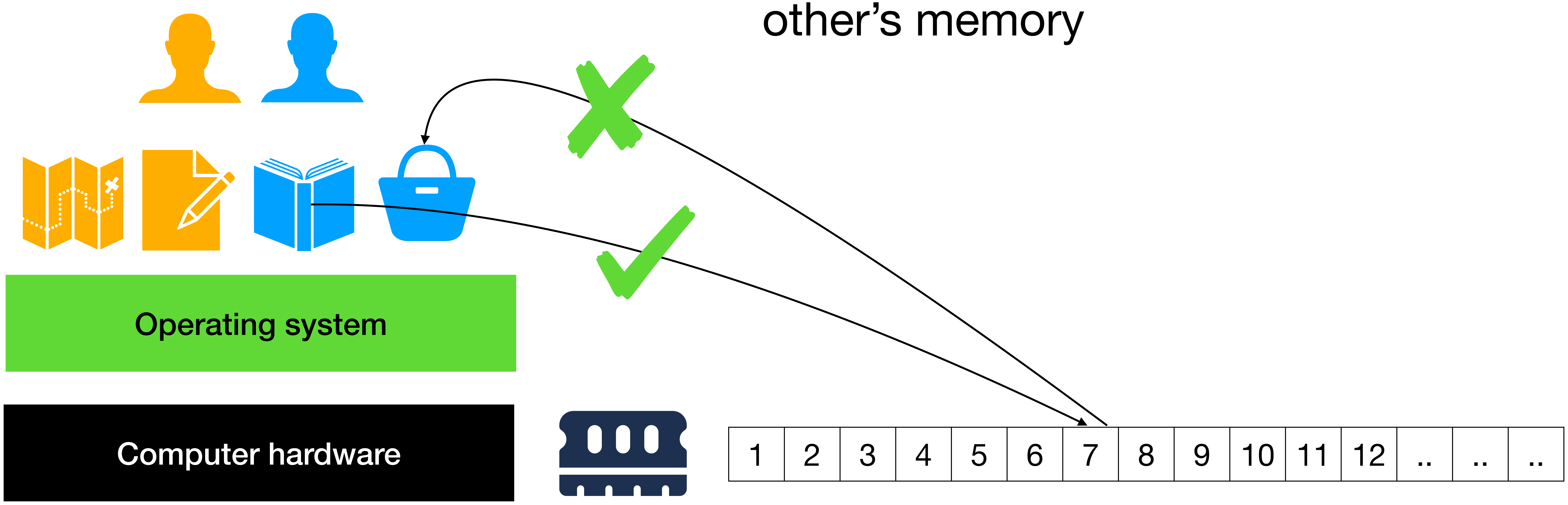



```
int fd = open("/home/abhilash/exam.txt", S_IRUSR | S_IWUSR);
int rc = write(fd, "Q1: \n", 4);
close(fd);
```

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

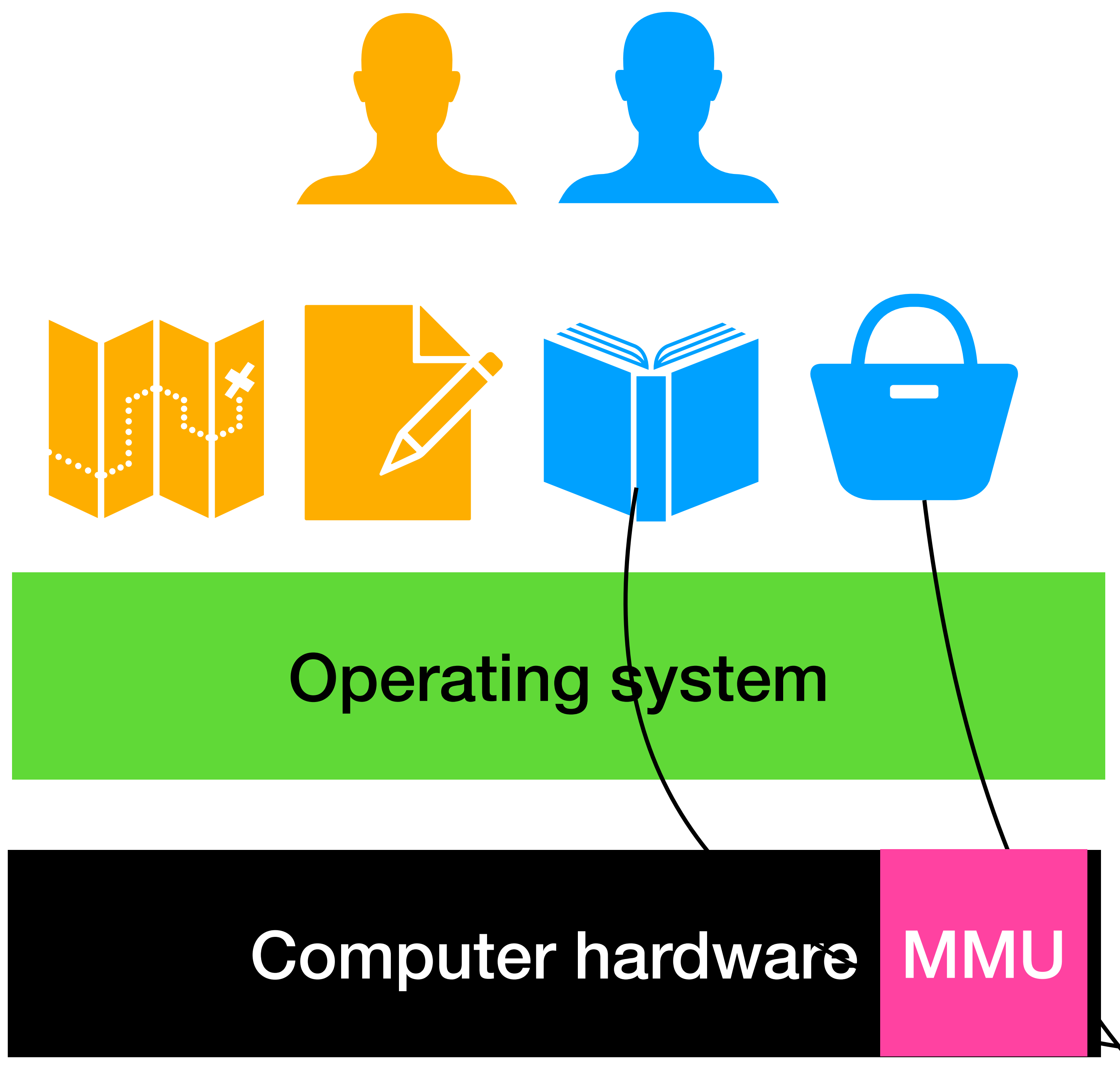
Purpose of OS: Protection

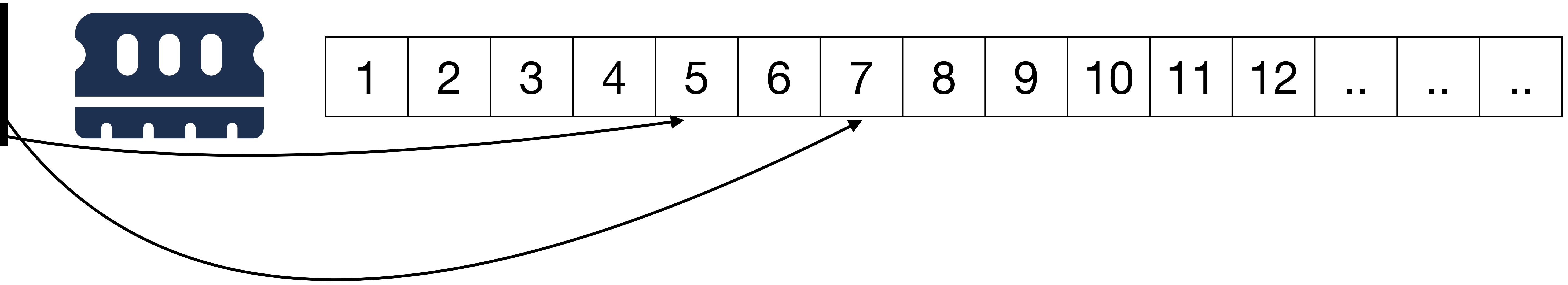
- Disallow inappropriate accesses. Example: processes reading each other's memory



Purpose of OS: Isolation

- Example: mem.c
- Multiple processes access the same pointer location but the OS isolated them with the help of hardware
- Each process thinks they own the memory





Course structure: OS in action

- We will build an OS (xv6) from scratch
  - Booting: Bootloader, ELF format
  - Input-output: Programmable interrupt controllers, traps, interrupt descriptor table
  - File system: FS layout, buffer cache layer, name layer, crash consistency, devices as files
  - Processes: memory segmentation, rings, process table, context switching, scheduling, system calls, exec system call
  - Concurrency: data races, different types of locks
  - Memory virtualization: memory hierarchy, address translation mechanism, demand paging, thrashing, fork system call
  - Shell: Pipes, IO redirection
  - Parallelism: Enable more CPUs, revisit locks

Data races due to concurrency

./threads 100000

Thread 1	Thread 2
Read counter = 0	
Write counter = 1	
	Read counter = 1
	Writer counter = 2
Read counter = 2	
	Read counter = 2
	Writer counter = 3
Writer counter = 3	

Why should I learn OS in 2026?
Isn't it a solved problem?

We have indeed made good progress ..

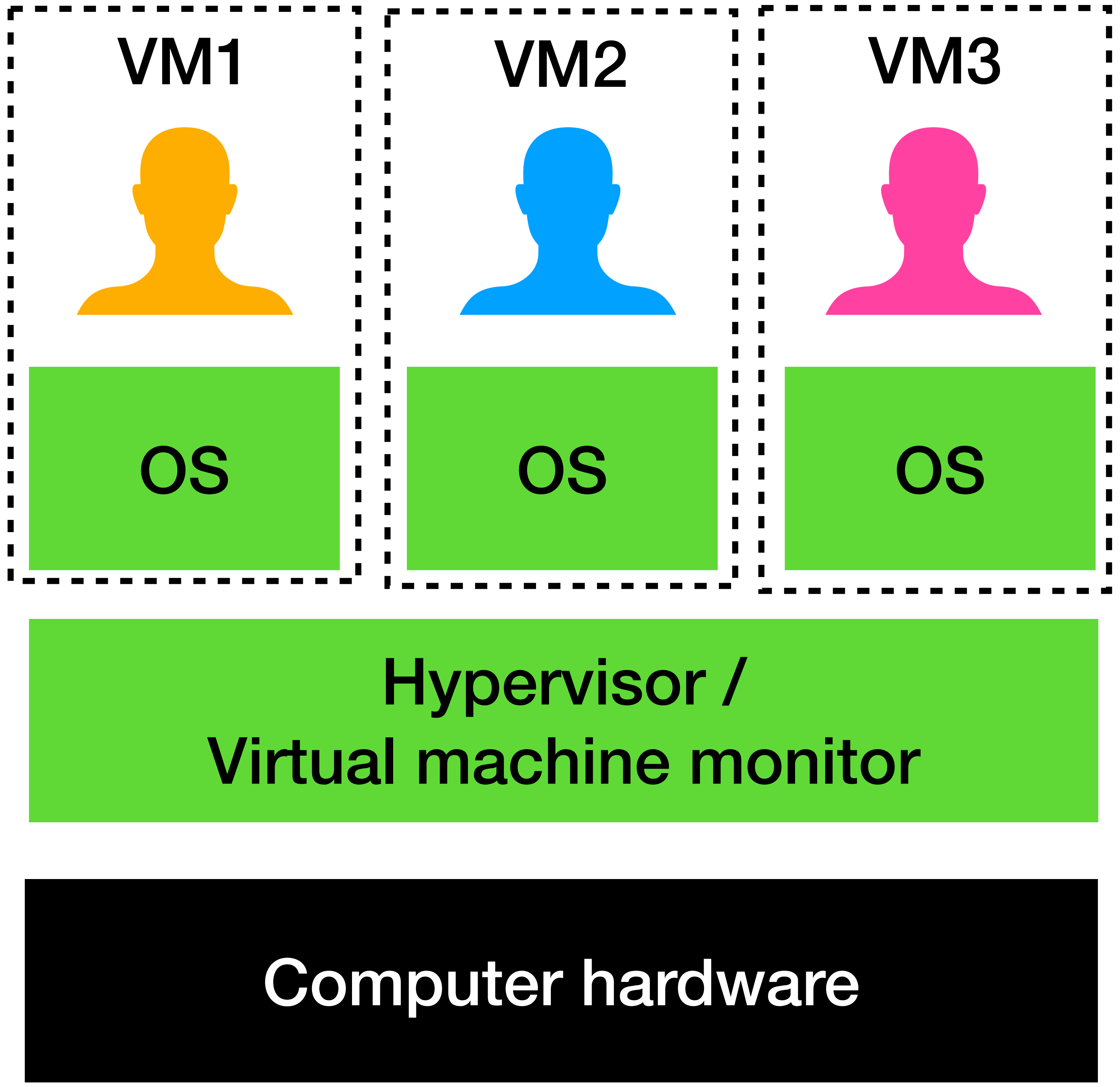
Computer operators

1960s: Batch jobs

Personal computers

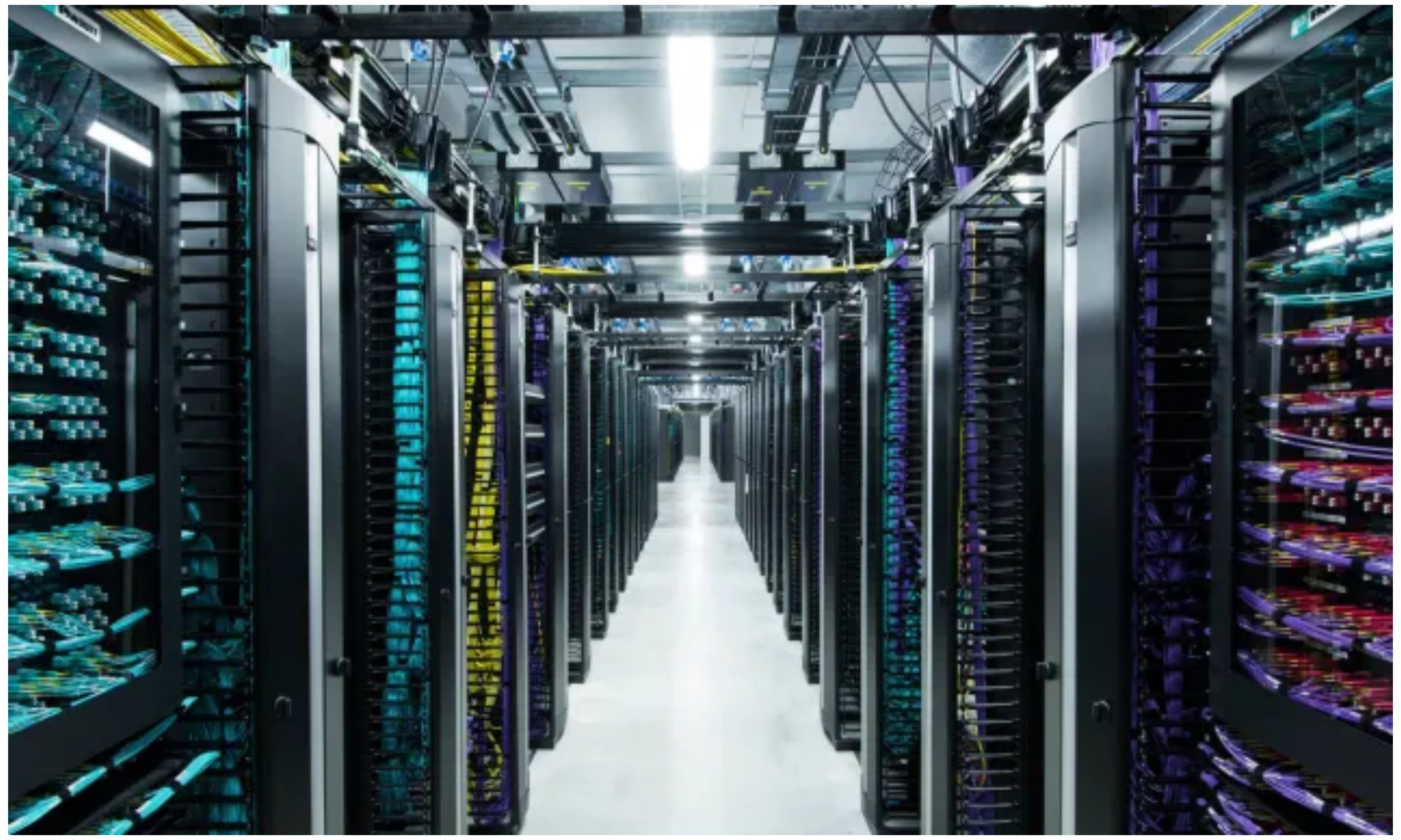
1980s: Interactive jobs!

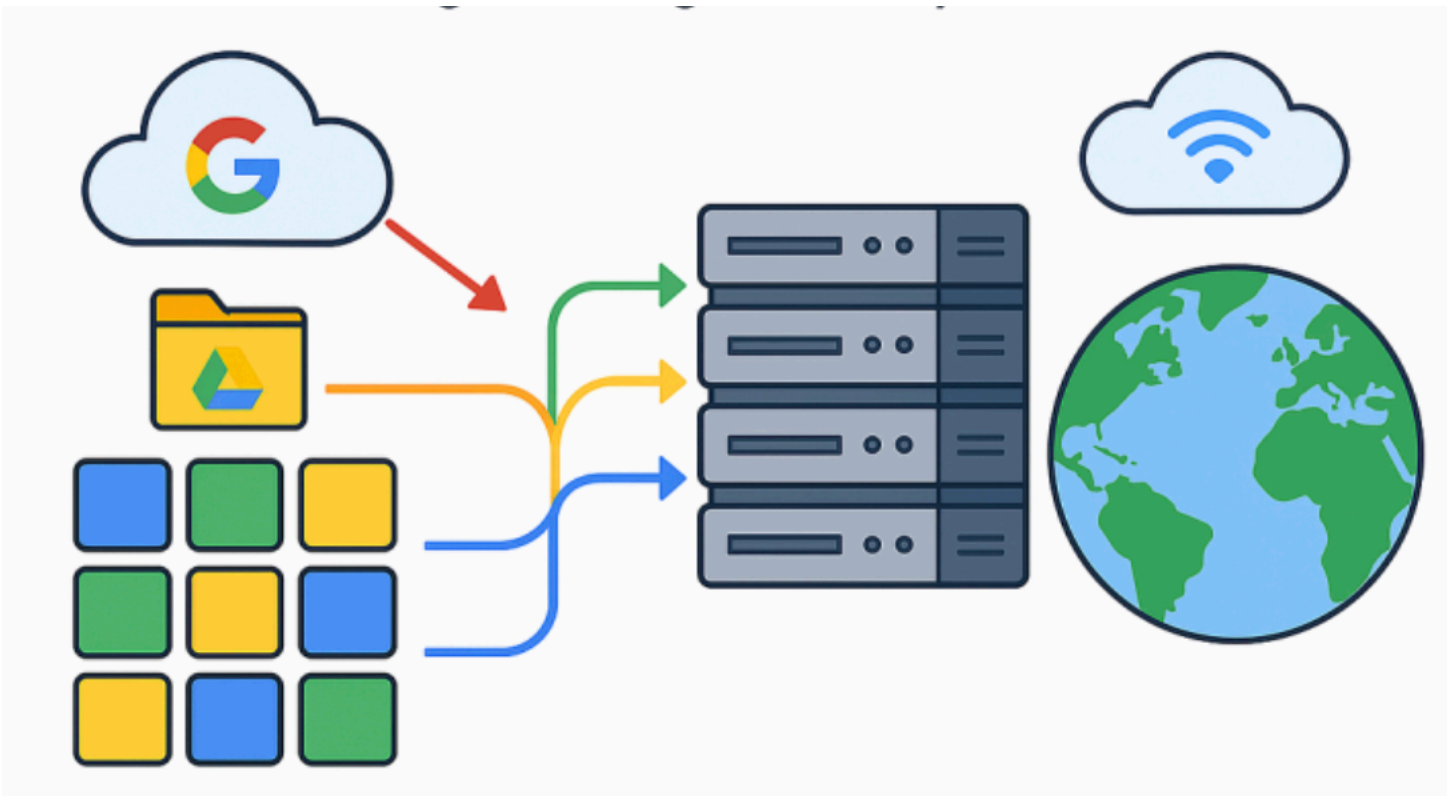
2000s: Cloud



- Cloud: I can rent *virtual machines* so I don't have to buy and manage servers.
- Hypervisor provides facilities to operating systems that OS provides to processes
  - multiplexes hardware among OS
  - protects and isolates OS from each other
- Hypervisors fundamentally enabled cloud computing

2000s: Distributed systems

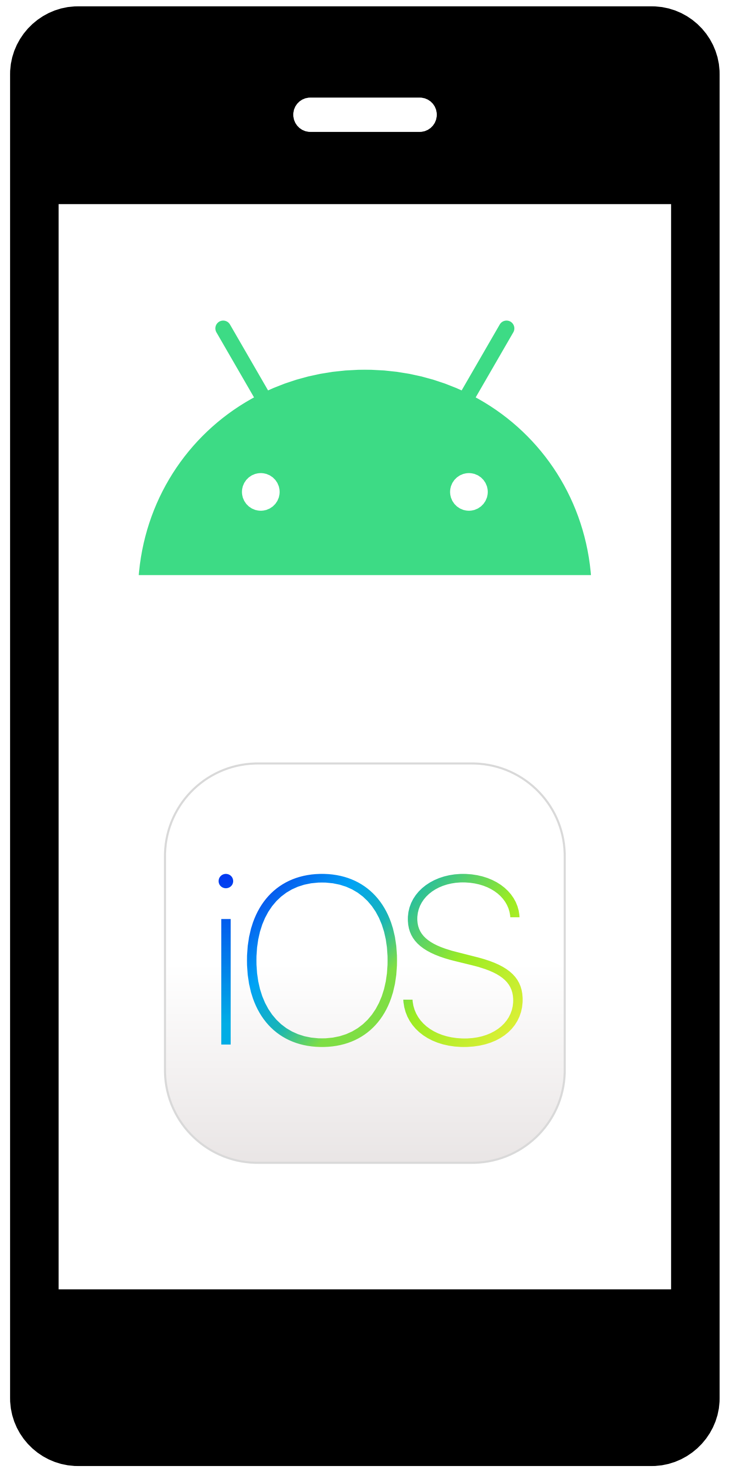




- My data does not fit in a single machine. Distributed file systems; e.g., crawled web data is stored in Google File System
- My “tasks” span 100s-1000s of machines; e.g., computing page rank, training large language models over the entire internet

2010s: Smartphones

- New kinds of higher-level services: localisation, cellular, accelerometer, touch interface, etc.
- Resource constraints: Power management, UI system, etc.
- Even higher-level services: voice recognition, augmented reality, etc.
- Increased security concerns because of increase in sensitive data with rise of mobile banking, UPI etc. and because of moving devices

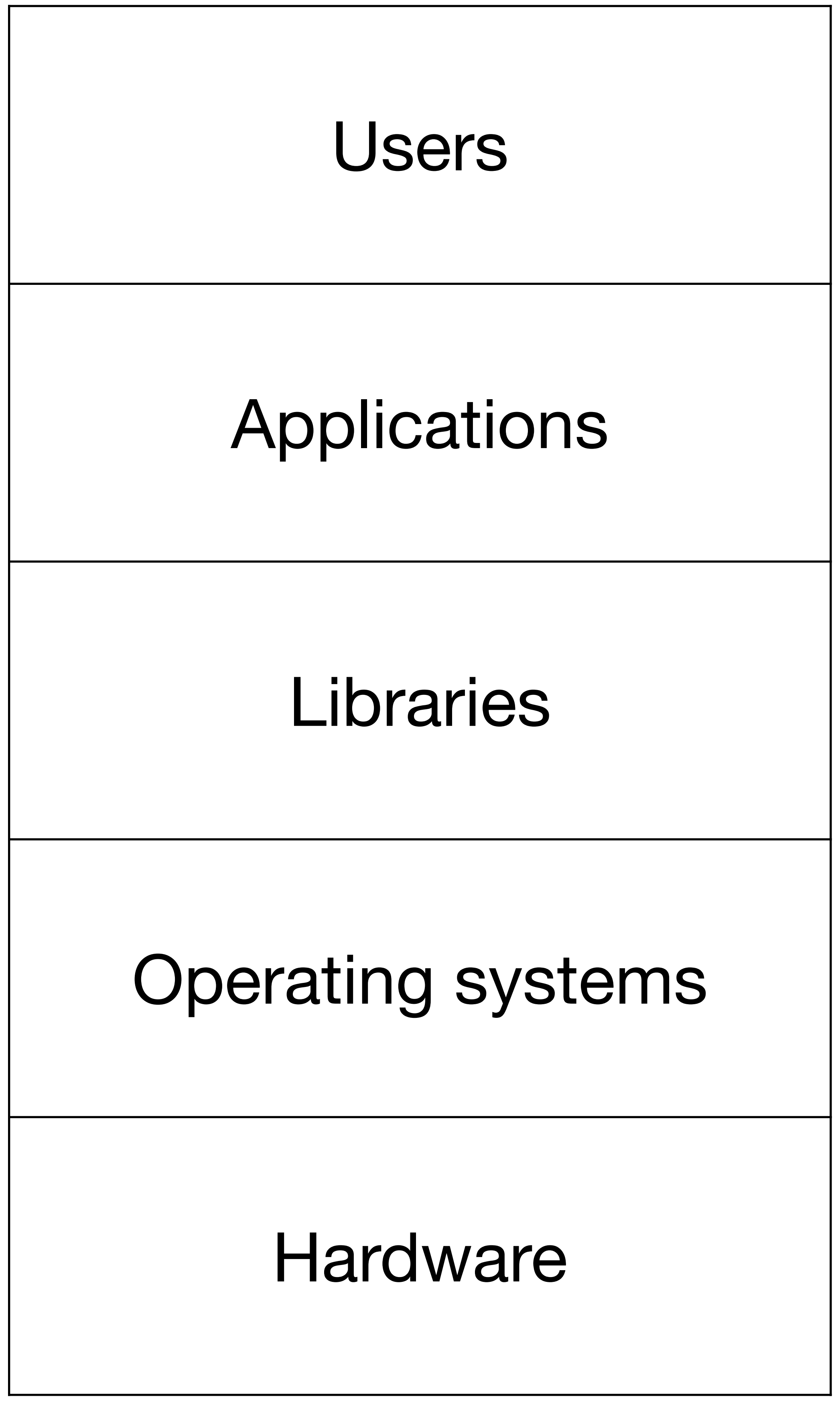


2010s: Cyber-physical systems

OS must not crash! Formally verified OS. Example: seL4

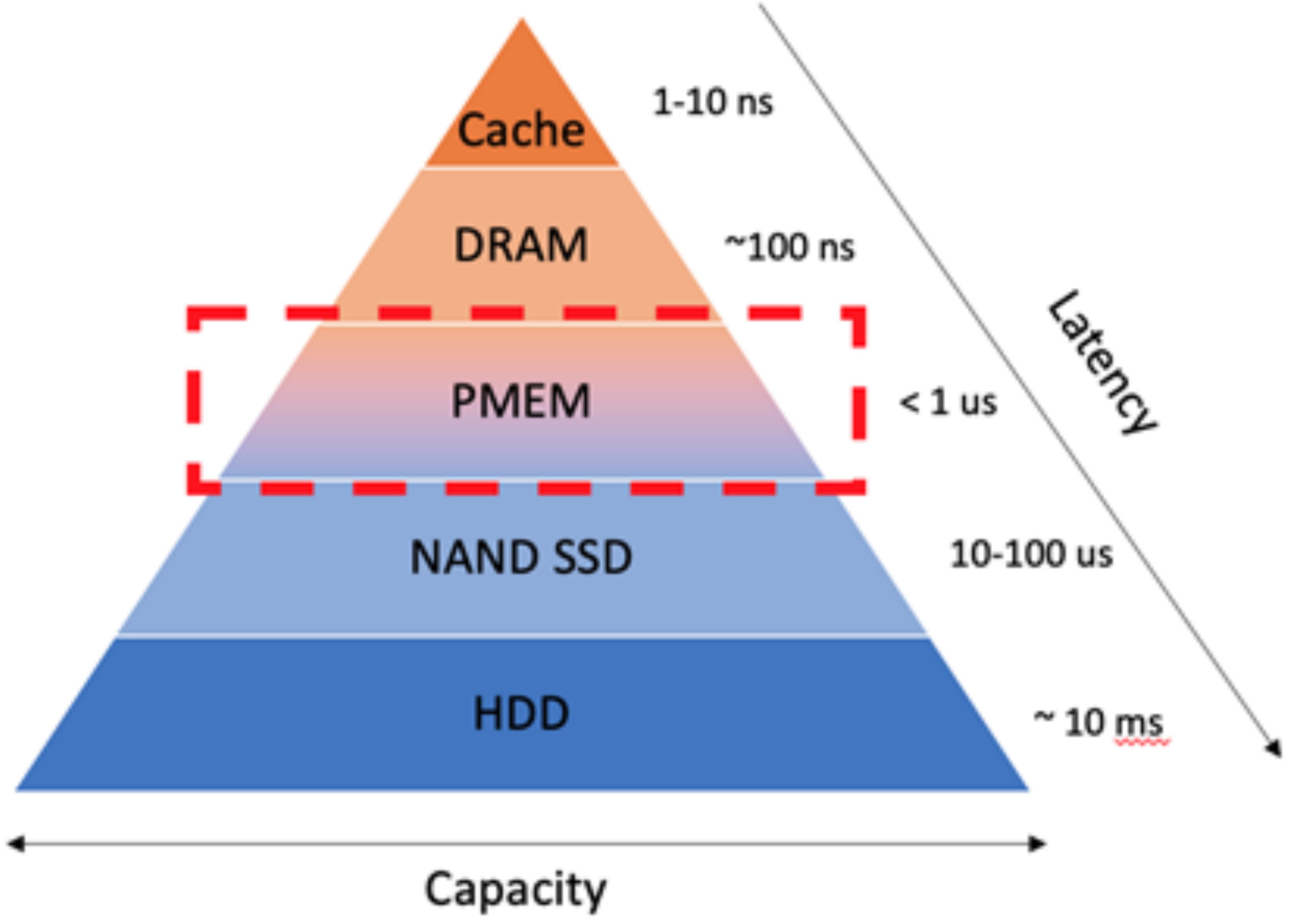
Typical progression of systems research

- Systems optimise for the “common case”
    - If common case changes, we need to rethink OS design
- Macro examples
    - Personal computers: batch jobs to interactive jobs
    - Smartphones: resource constraints, new sensors
    - Cyber physical systems: risk of human life



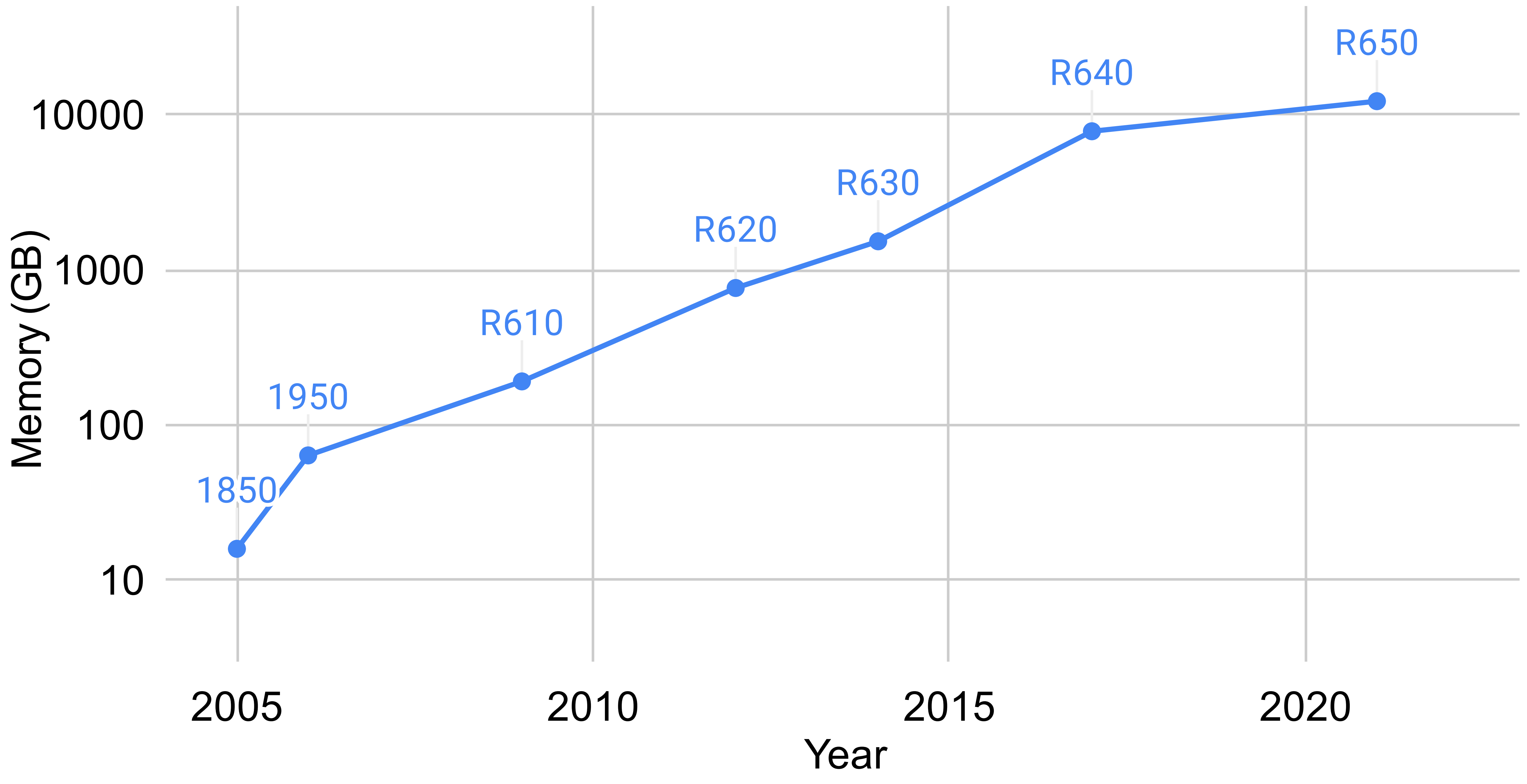
More trends: persistent memory

- Memory: volatile but fast
- Disk: persistent but large
- PMEM: persistent and fast!
  - PMEM aware file systems



More trends: big memory

- OS were designed when memory was scarce: few KBs
- You can now buy a server with 12TB of DRAM!
- Transparent huge pages
- Caching: Redis/memcached, etc.
- In memory compute: Spark, etc

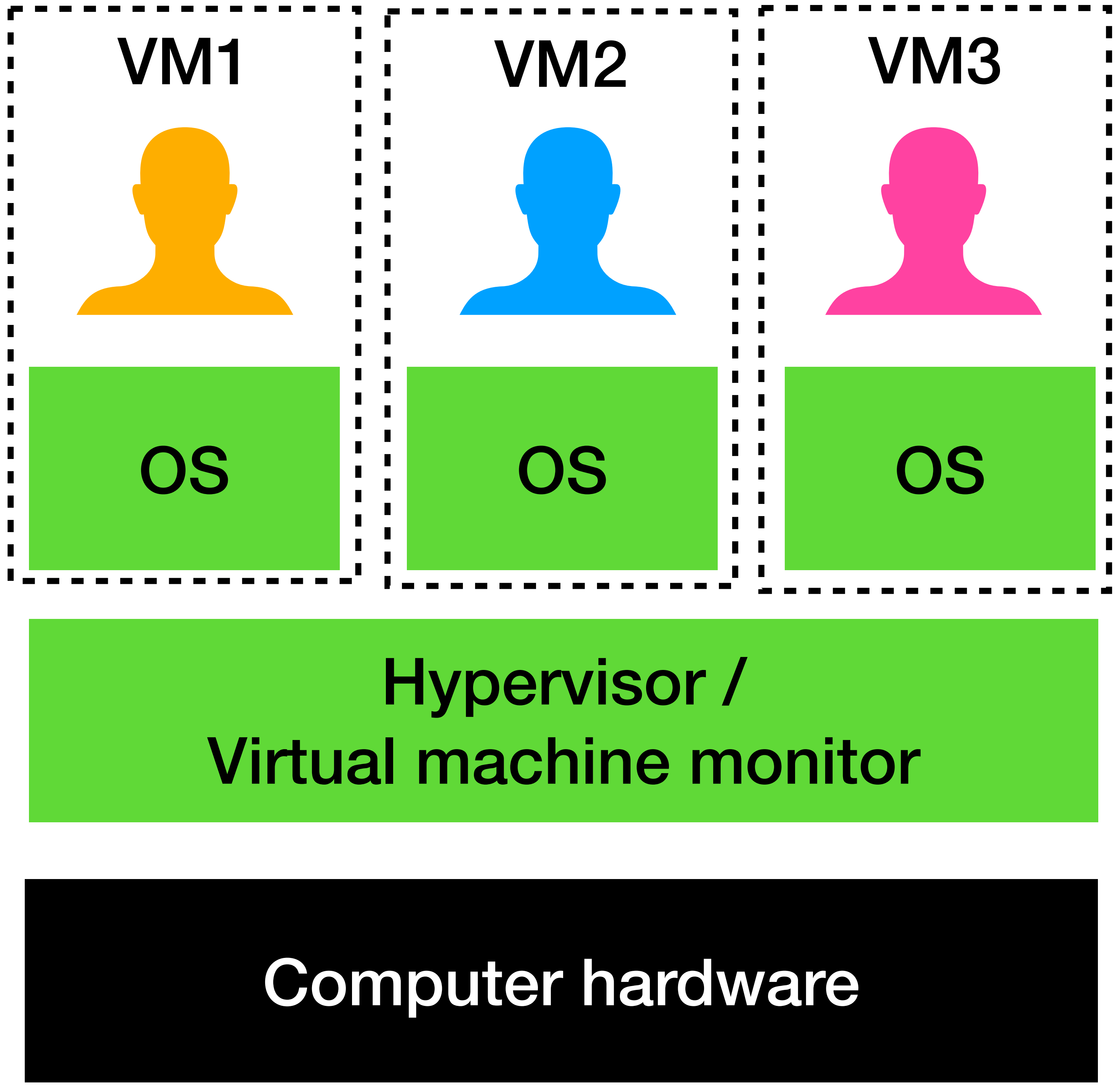


More trends: fast networks

- OS typically assumed network is *much slower* than DRAM
  - Far memory

2020s	Latency	Bandwidth
DRAM	15ns	400 GBps
Ethernet	500ns	50 GBps

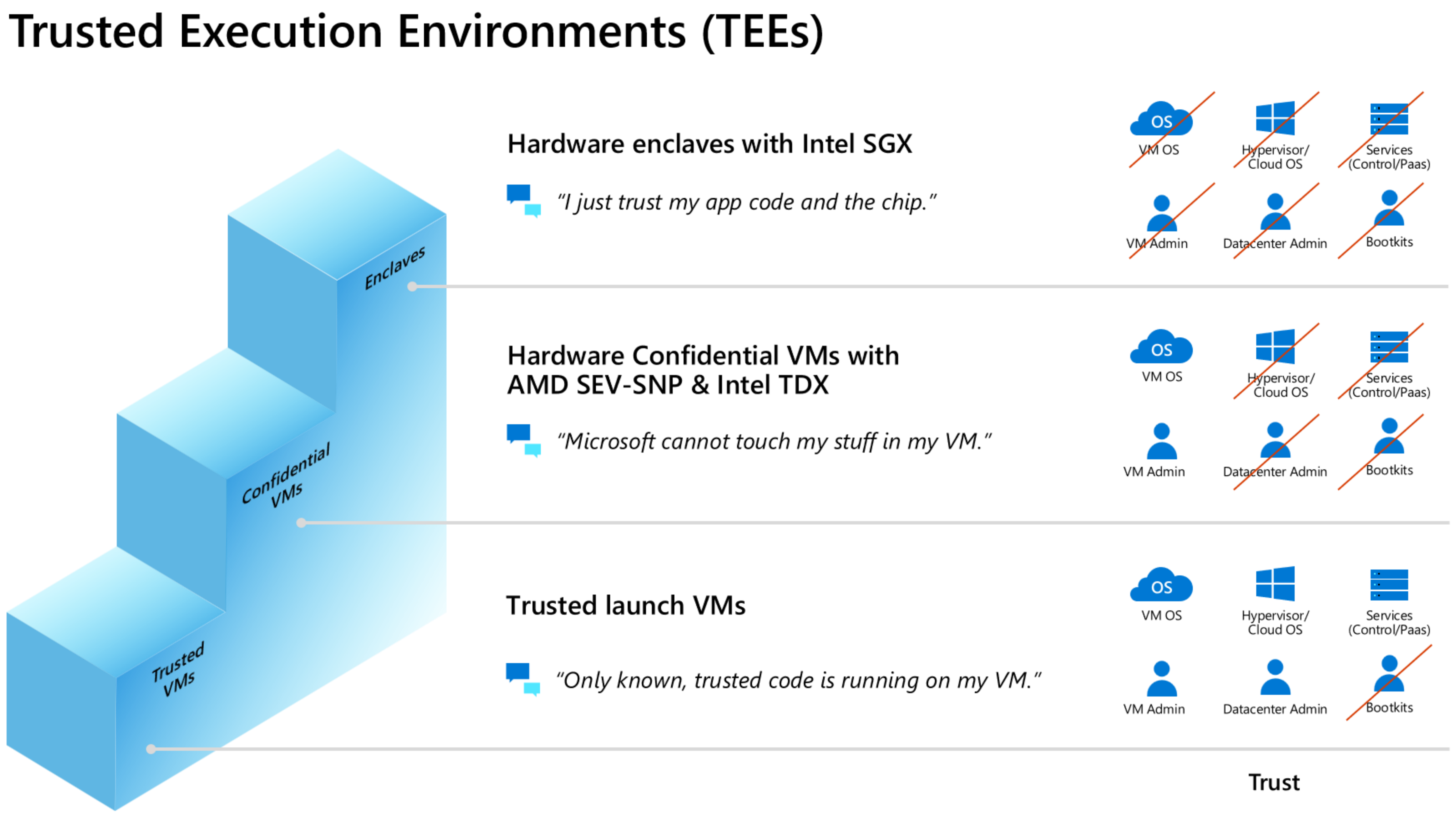
More trends: Rise of Unikernels



- OS optimises for common behaviours across all applications
- Each OS is now running only single application
- Unikernels optimise only for a single application

Confidential computing

- Developers want to run their proprietary code/models on sensitive data on someone else's machine (cloud) where all software including (host) OS is untrusted



More trends: ML/LLM Systems

- ML training at scale
  - Pathways/TensorFlow (Google), Ray (Anyscale), …
- LLM inference at scale
  - VLLM, SGLang, etc.
- Agentic programs

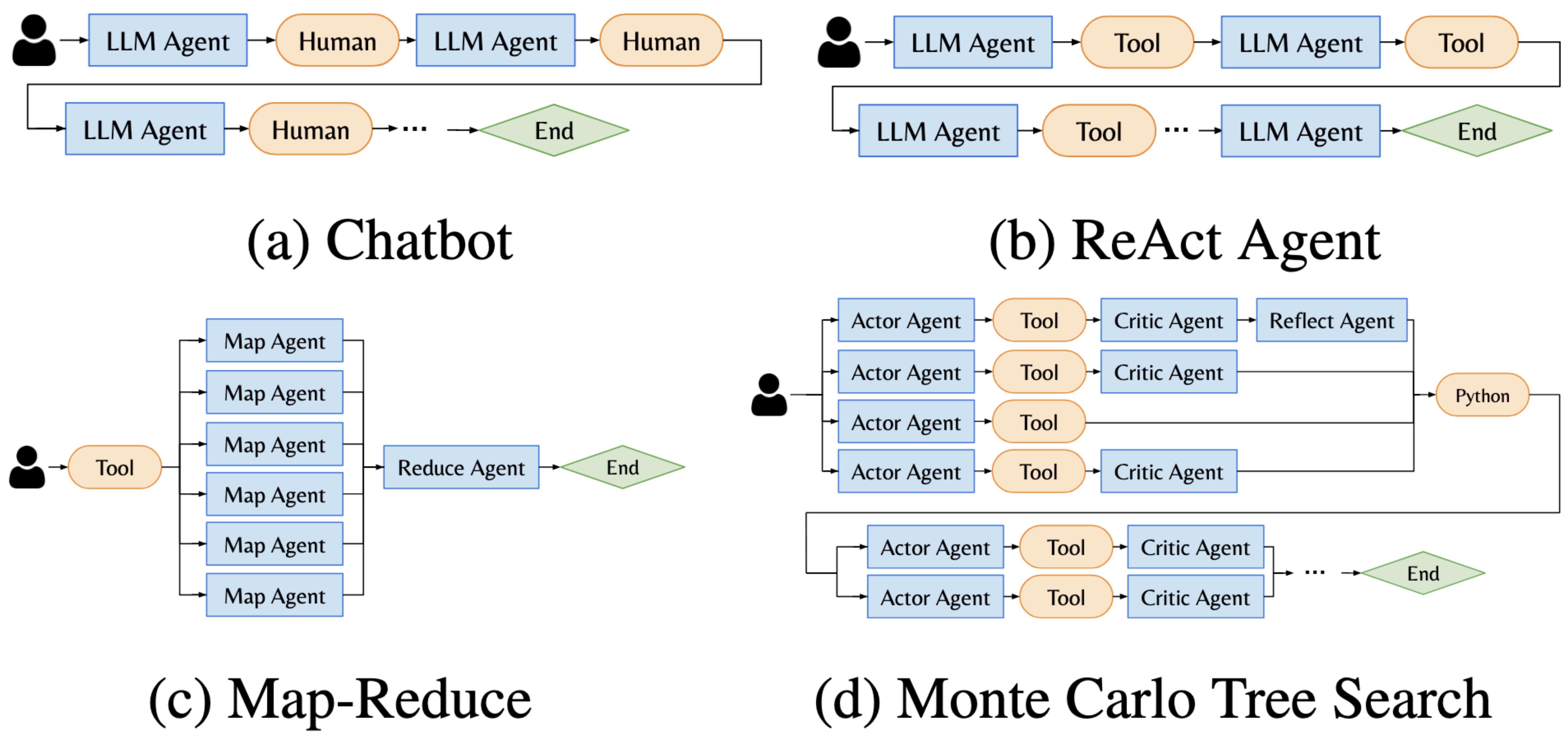


Figure 1: **Execution workflows for *Agentic Programs*.** Agentic programs are highly dynamic execution workflows that follow a directed acyclic graph (DAG). It consists of LLM calls from one or more LLM agents and external interrupts (i.e. tool calls, humans).

Image from Autellix paper

Why should I care about learning OS?

If I don’t want to do systems research

- OS is a study of abstraction. Absorb all the complexity away from the developer / the user.
- OS has to provide high performance. Manage resources, provide isolation and protection with minimal overheads.
- Principles are useful when designing any practical system