CS111, Lecture 26
Virtual Machines and Networking

Extra Topic 1- How can we virtualize the entire computer hardware so that we can run entire other OSes?

Extra Topic 2- How can we write programs that communicate over a network with other programs?

Learning Goals

- Learn about virtual machines and how they build on our understanding of virtualization to virtualize the entire hardware
- Understand the fundamentals of networking and how machines can communicate

Plan For Today

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2:** Networking
- Client-server
- Networking system calls
- Demo: Time Client

Plan For Today

- **Topic 1: Virtual Machines**
- The Hypervisor
- **Topic 2:** Networking
- Client-server
- Networking system calls
- Demo: Time Client

Extra Topic 1- How can we virtualize the entire computer hardware so that we can run entire other OSes?

Virtual Machines

- A Virtual Machine is an abstraction of the entire computer hardware – software enables us to run multiple OSes simultaneously, each thinking it has its own machine!
- Virtual Machines even let us run an OS within an OS!
- A powerful application of the idea of **virtualization** – make one thing look like something else, or many of them.
- Powerful use cases, enabling new features and functionality
- Demo: VMware Fusion

Virtual Machines

Why are Virtual Machines useful?

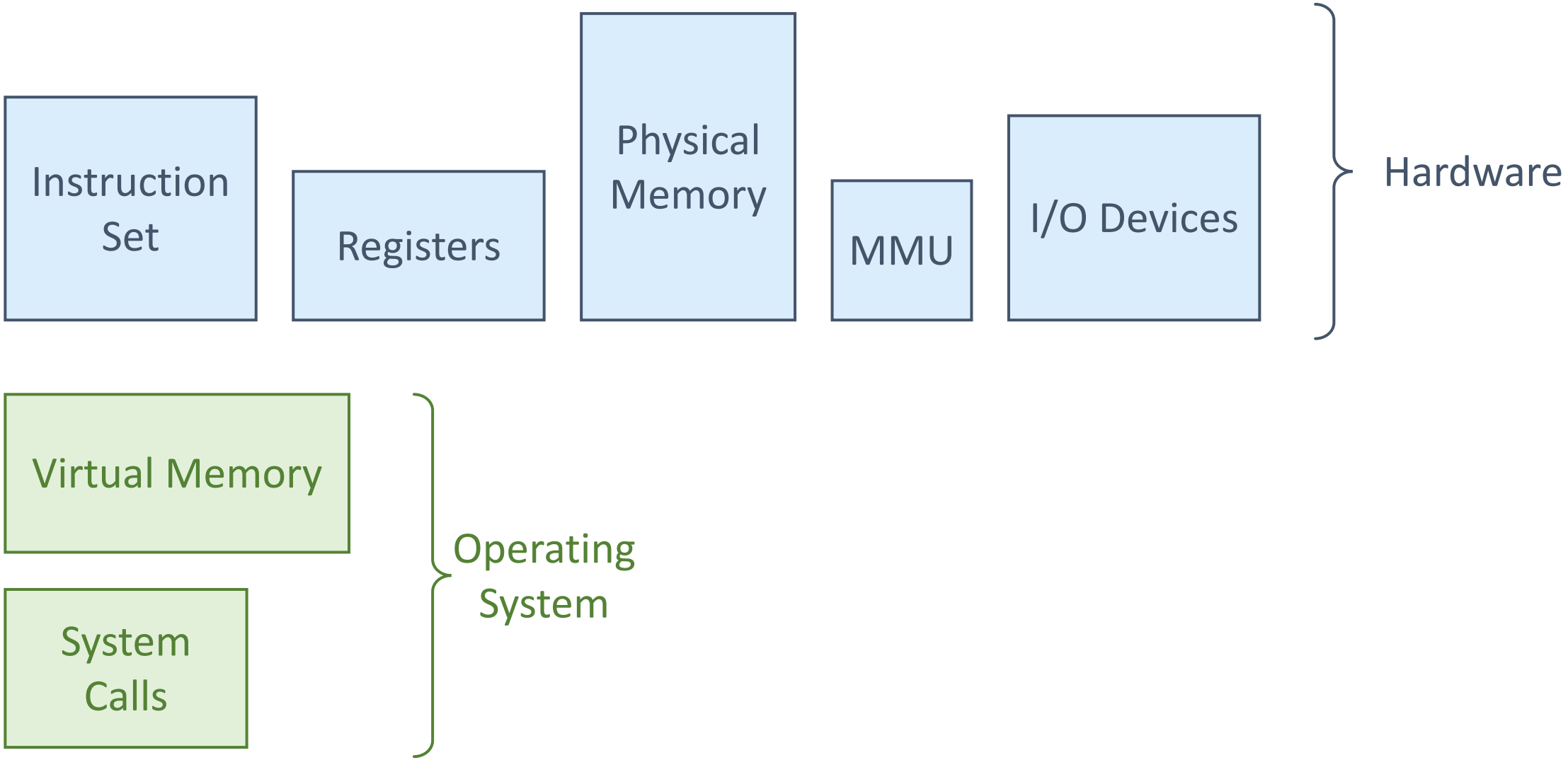
- **Software development** – test software on different OSes / versions on a single machine
- **Datacenters** – rather than one application per machine (for isolation), we can have one VM per application and run several per machine.
- **Snapshots** – we can save / continue / restore VM state!
  - E.g. in a datacenter, migrate VMs between machines to balance load.
  - E.g. in software development, run tests with same saved VM configuration – reproducible tests
- Heavily used in cloud computing (e.g. Amazon Web Services, Google Cloud)
- Variations of this idea – called *containers* (e.g. Docker), like lightweight VMs
- How do these work?

Virtual Machines

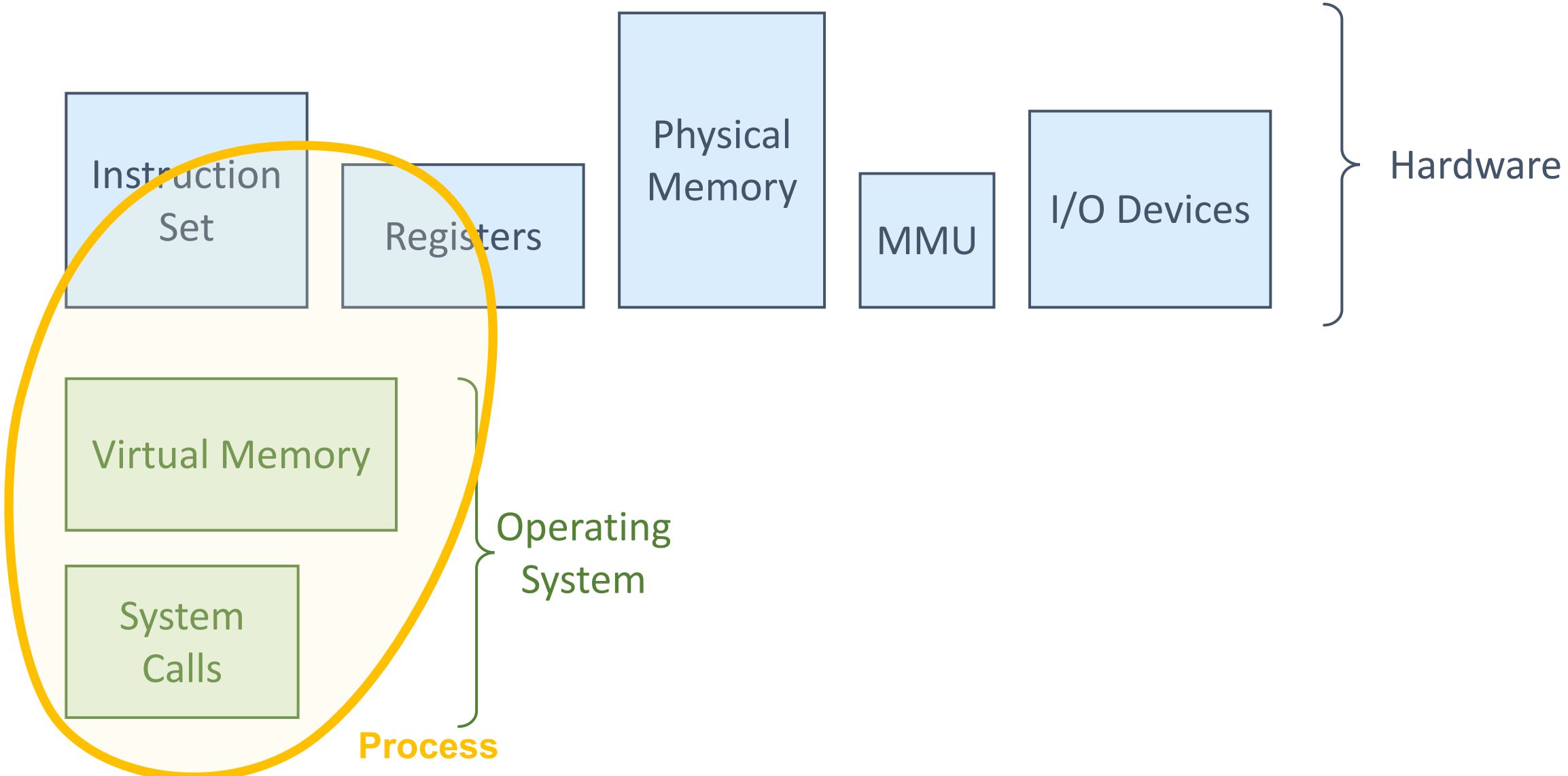
Key idea – we need a more powerful version of a “process” that can run an entire OS.

Regular processes can’t do privileged OS tasks nor access all hardware functionality that an OS needs; it’s running in user mode!

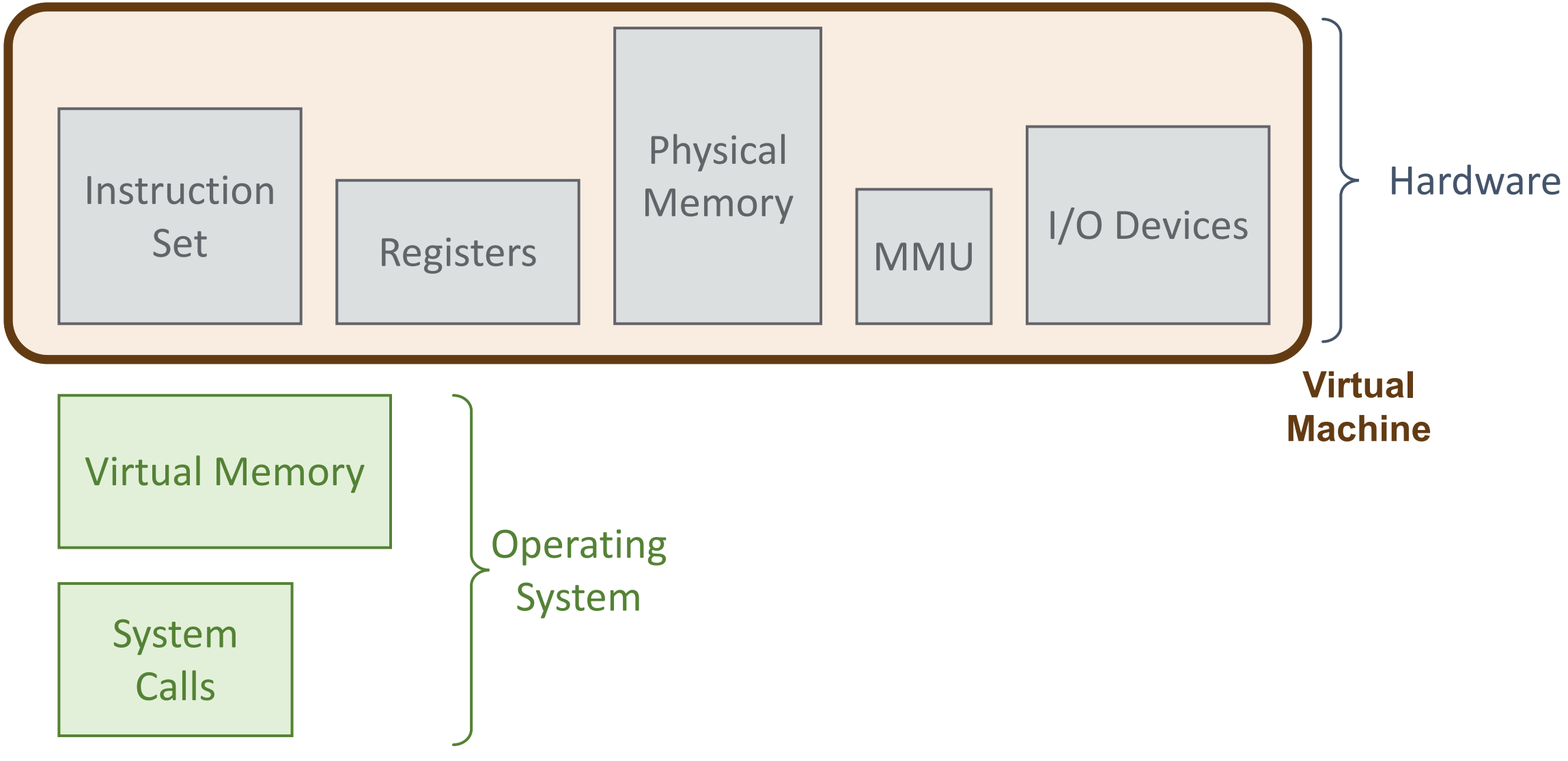
Process Abstraction



Process Abstraction



Process Abstraction



Plan For Today

- **Topic 1:** Virtual Machines
- **The Hypervisor**
- **Topic 2:** Networking
- Client-server
- Networking system calls
- Demo: Time Client

Virtual Machines

A hypervisor is the software that enables this - interface for OSes to run.

- Can run just hypervisor on machine, and then 1 or more OSes on top of it
- Can run hypervisor on top of OS, running an OS within an OS

Hypervisor

- **One possible approach** – simulate *everything* in software (even instruction execution). But too slow….
- We want to give the virtual machine access to the real hardware as much as possible to improve performance.
- **Idea:** if the OS does something that a normal process can do, just do as normal on real hardware. For other things, have hypervisor step in and simulate.

Privileged Instructions

Example #1: what if the guest OS executes an instruction only OSes can run?

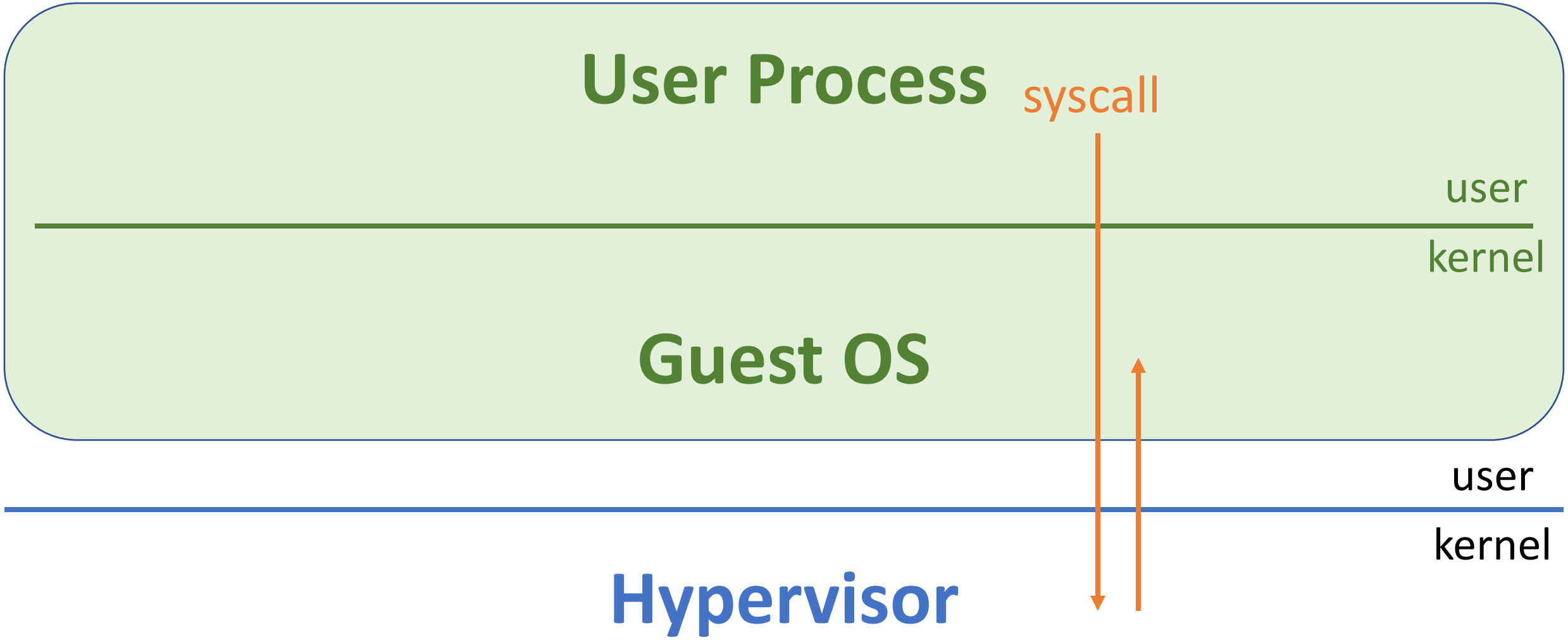
- Since virtual machine runs in user mode, these cause “illegal instruction” traps into hypervisor
- Hypervisor catches these traps, simulates appropriate behavior

System Calls

Example #2: what if a program running in the guest OS makes a system call?

- By default, goes to host OS, not guest OS!
- Hypervisor traps, analyzes trapping instruction, simulates system call back to guest OS

System Calls



I/O Devices

Example #3: what if the guest OS interacts with I/O devices?

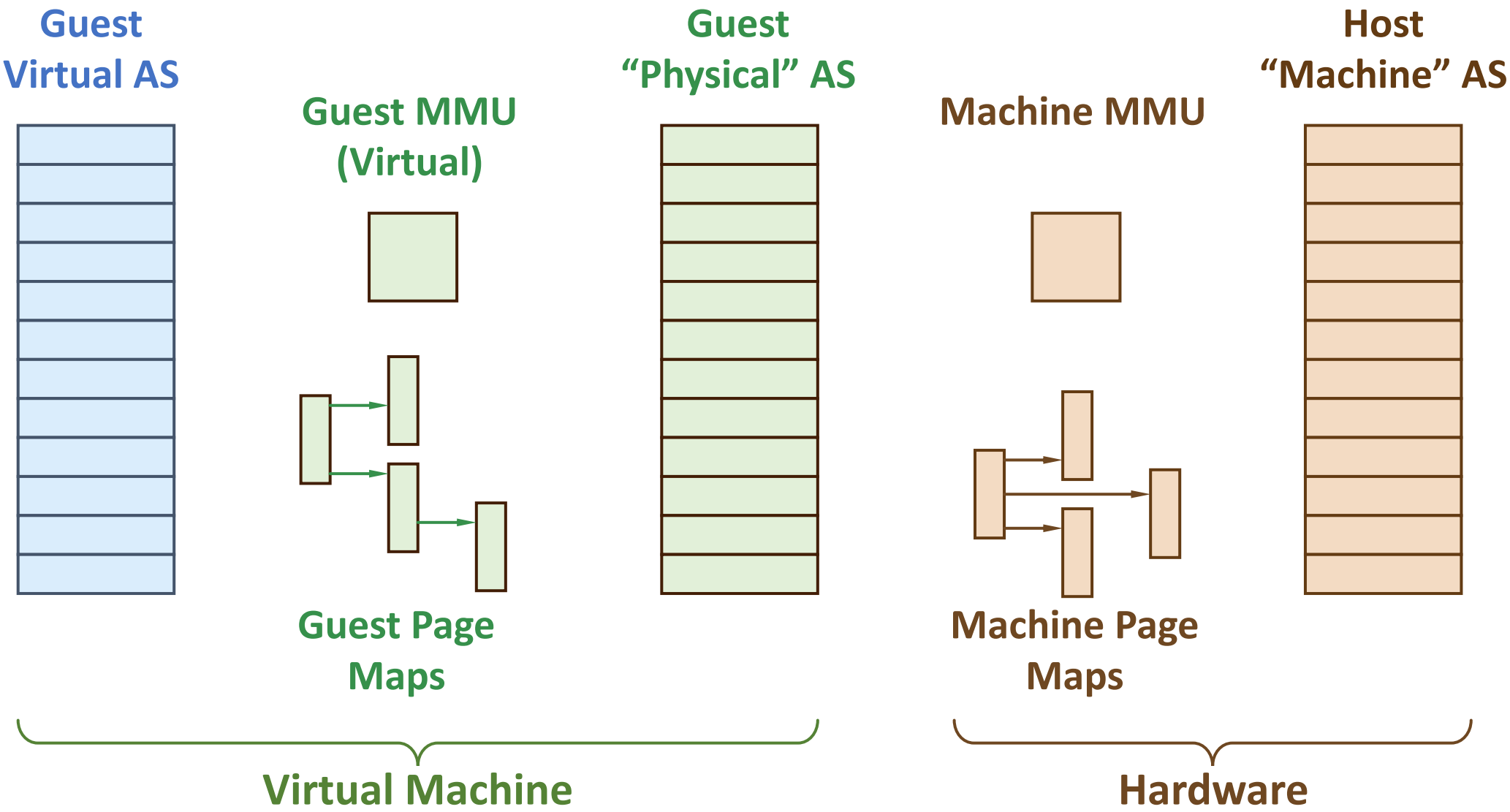
- Hypervisor configures guest OS devices such that it can intercept communication
- Hypervisor can then handle it – e.g. when actual I/O operation completes, hypervisor simulates interrupt into the guest OS
- This can be slow – one solution is to provide hypervisor-specific functions that the guest OS can call (breaks abstraction!) for devices

Virtual Memory

What about memory access and management?

- The hypervisor gives guest OSes memory like processes get memory – virtual addresses mapped to physical addresses behind the scenes.
- **Mind-bending:** the guest OS uses this virtual address space as its *physical memory*, and it parcels that out to virtual address spaces in its own processes!

Virtualizing Virtual Memory



Virtualizing Virtual Memory

- **Three** levels of memory!
- Implementation today – *extended page maps* (Intel support in recent years)

Virtual Machines

- A powerful application of the idea of **virtualization** – make one thing look like something else, or many of them.
- Powerful use cases, enabling new features and functionality

Plan For Today

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2: Networking**
- Client-server
- Networking system calls
- Demo: Time Client

Extra Topic 2- How can we write programs that communicate over a network with other programs?

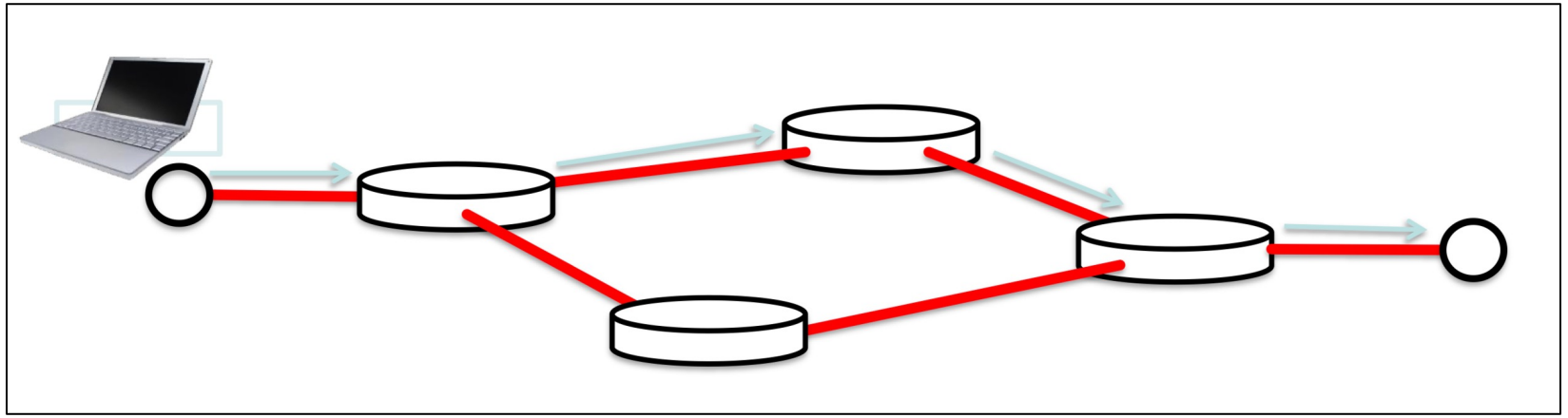
Networking

- We have learned how to write programs that can communicate with other programs via mechanisms like pipes.
- However, the communicating programs must both be running on the same machine.
- Networking allows us to write code to send and receive data to/from a program running on another machine.
- Many new questions, such as:
  - how does the data get there?
  - what functions do we use to send/receive data? (new system calls!)

Networking and CS144

Take CS144 if you're interested in learning more about how networks work – how data gets from one place to another. Questions addressed include:

- How is data packaged up to be sent over the network? (packets)
- How does my data make it to the destination in one piece? (packet loss, TCP)
- How do packets get routed across the network from one machine to another?



(diagram from CS144 slides)

Networking and CS142

Take CS142 if you're interested in learning more about how to write web-based programs that leverage networking functionality.

Plan For Today

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2:** Networking
- **Client-server**
- Networking system calls
- Demo: Time Client

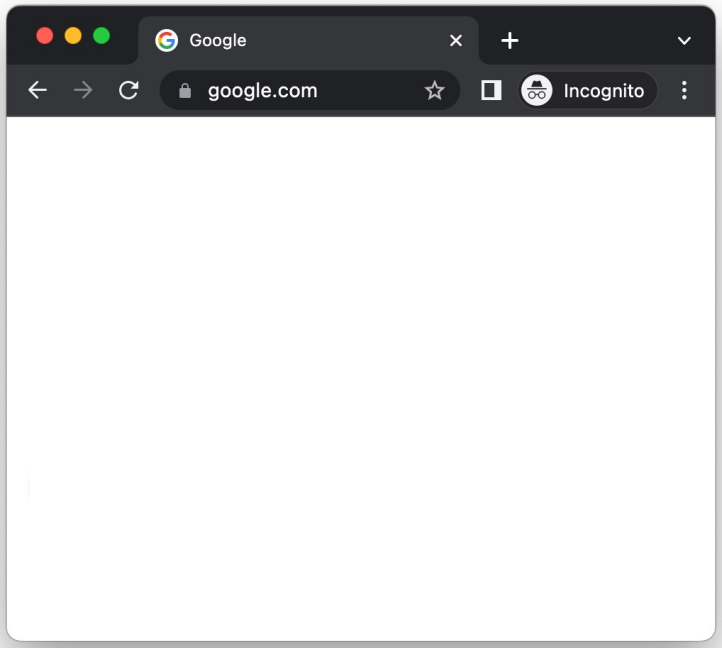
Networking Patterns

Most networked programs rely on a pattern called the "client-server model”

- This refers to two program "roles": **clients** and **servers**
- **clients** send requests to **servers**, who respond to those requests
  - e.g. YouTube app (client) sends requests to the YouTube servers for what content to display
  - e.g. Web browser (client) sends requests to the server at a URL for what content to display
- A **server** continually listens for incoming requests and sends responses back ("running a phone call center")
- A **client** sends a request to a server and does something with the response ("making a call")

Client-Server Model

Client



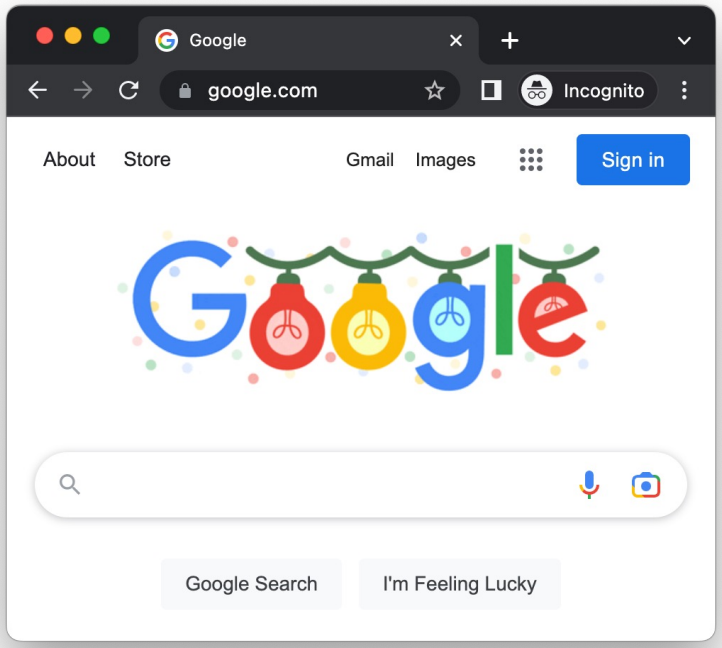
Server

google.com, please!

Sure, here's the content for that page.

Client-Server Model

Client



Server

google.com, please!

Sure, here's the content for that page.

Plan For Today

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2:** Networking
- Client-server
- **Networking system calls**
- Demo: Time Client

Networking System Calls

If we're a client program, how do we communicate with a server?

- Linux uses the same **descriptor** abstraction for network connections as it does for files!
- You can open a connection to a program on another machine and you'll get back a **socket descriptor** number (using your file descriptor table)
- A **socket** is an endpoint of a single connection. It is represented as a descriptor we can read from/write to, and we close it when we're done.
- **Like a pipe, but with only *one* descriptor, not two**.
- **Key idea:** networking is remote function call and return

Networking System Calls

How do you specify who you want to talk to?

- IP address – the address of the machine you want to connect to
- Port number – the program on that machine you want to connect to
  - Every listening program (e.g. server program) on a machine has a unique port number
  - Like “virtual process IDs”
- A **server** program will run on a machine and be assigned a port number (there are established port numbers for some common types of programs, e.g. HTTP (internet traffic) is always port 80)
- A **client** program wishing to connect to that server must communicate with that port number at that IP address.
- We commonly map names to IP addresses (e.g. www.google.com) to make it easier to specify who we want to connect to.

Plan For Today

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2:** Networking
- Client-server
- Networking system calls
- **Demo: Time Client**

Demo: Client Program

Let's write our first program that sends a request to a server!

- **Example:** I am running a server on **myth64.stanford.edu**, port **12345** that can tell you the current time. Whenever you connect to it, it will send back the current time as text.
- Let's write a client program that connects and prints out what the server says.
- Demo: **time-client-descriptor.cc**

Helper function to connect to a server (implemented via system calls:

```
// Opens a connection to a server
int createClientSocket(const string& host, unsigned short port);
```

Networking

Key Idea: there is no code in the client that is itself calculating the current time. All that logic is in the server that the client connects to! Essentially "remote function call and return".

How do servers work?

- Constantly running, listening for incoming requests
- Use system calls to listen for requests and respond to them
- Multithreading is a powerful tool for helping servers respond to many incoming requests in parallel!

Recap

- **Topic 1:** Virtual Machines
- The Hypervisor
- **Topic 2:** Networking
- Client-server
- Networking system calls
- Demo: Time Client

Next time: wrap-up / life after CS111

Lecture 26 takeaway: Virtual machines are an abstraction of the entire machine that lets us run multiple OSes. Networking allows programs on separate machines to communicate.