CS162: Operating Systems and Systems Programming

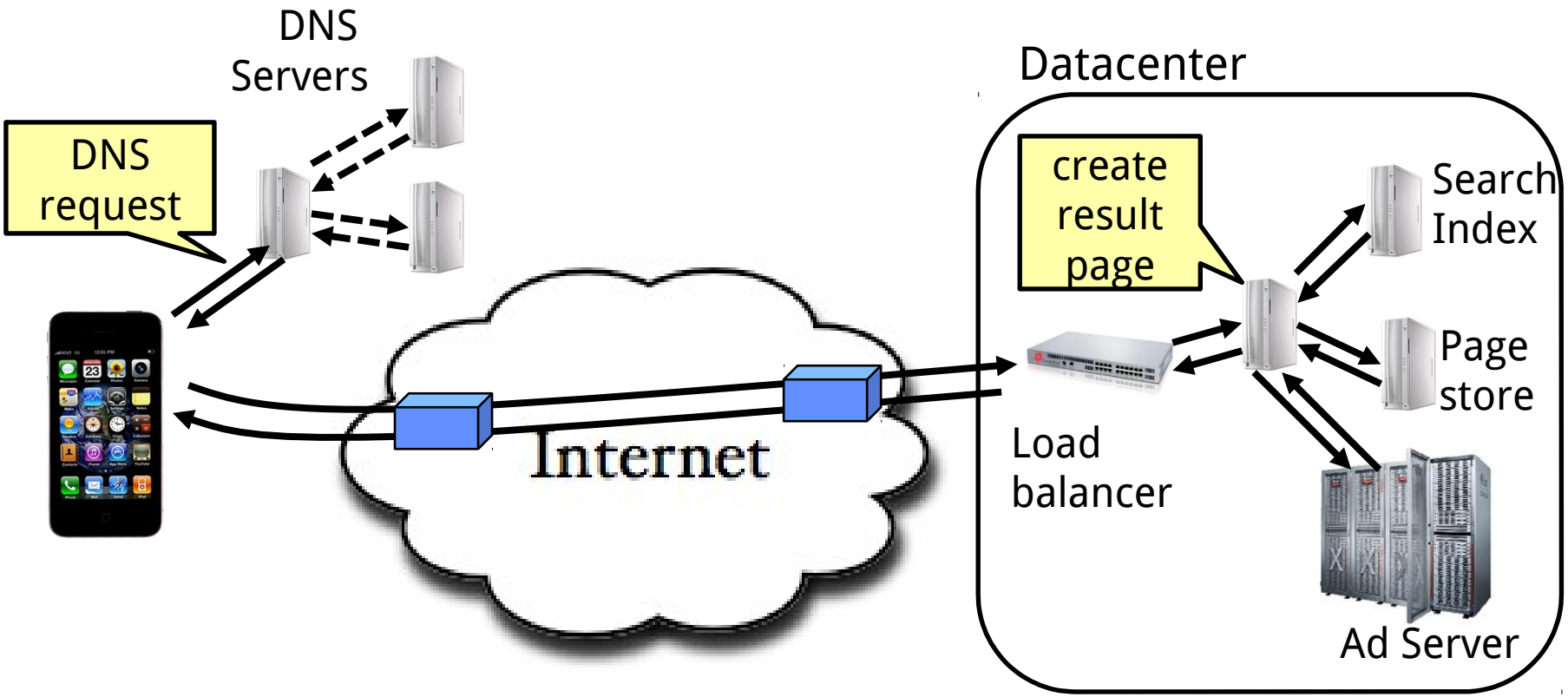
Lecture 1: What is an Operating System?

22 June 2015

Charles Reiss

https://cs162.eecs.berkeley.edu/

Example: Search Query



Components to coordinate

Many shared resources

Why take CS162?

You will build (parts of) operating systems

You will build other software using OS concepts

You will build applications that use OSes

Topics for Today

What is (or is not) an operating system?

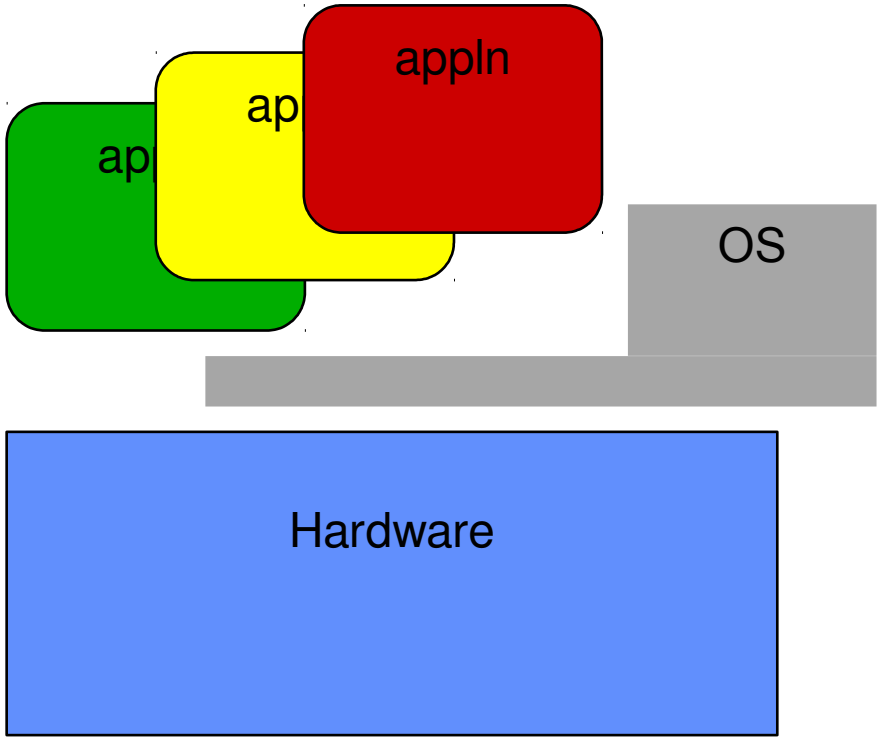
Some common operating system abstractions

... interrupted by Logistics

What is an operating system? (1)

Layer of software that provides applications access to hardware

- Abstraction of complex hardware
- Protected access to shared resources
- Security
- Communication

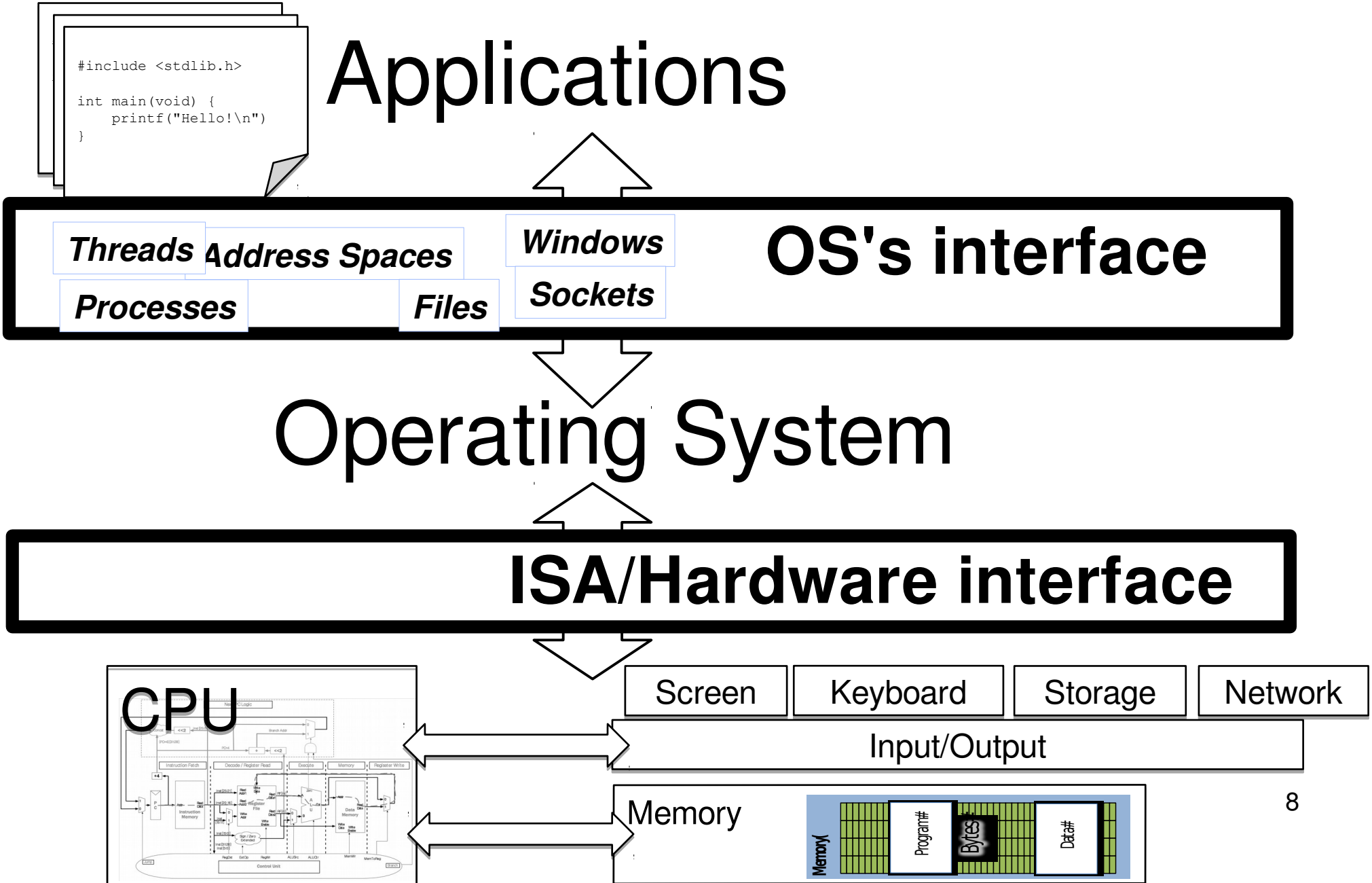


History: The Computer Operator

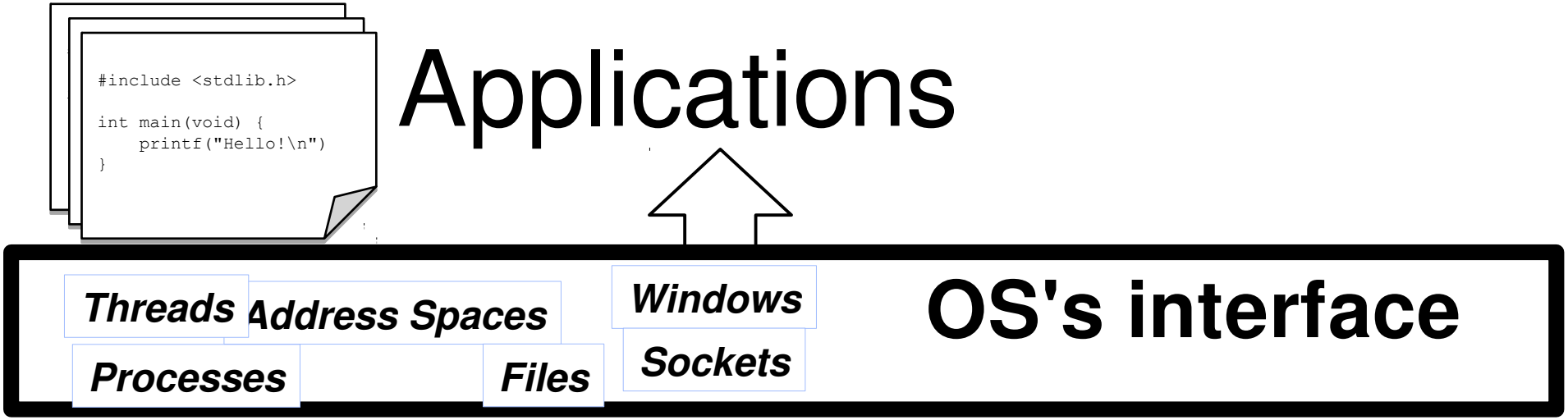
What is an operating system? (2)

Provide programs with a more convenient "abstract" machine instead of the existing machine

The Abstract Virtual Machine (1)



Abstract VM: Application View



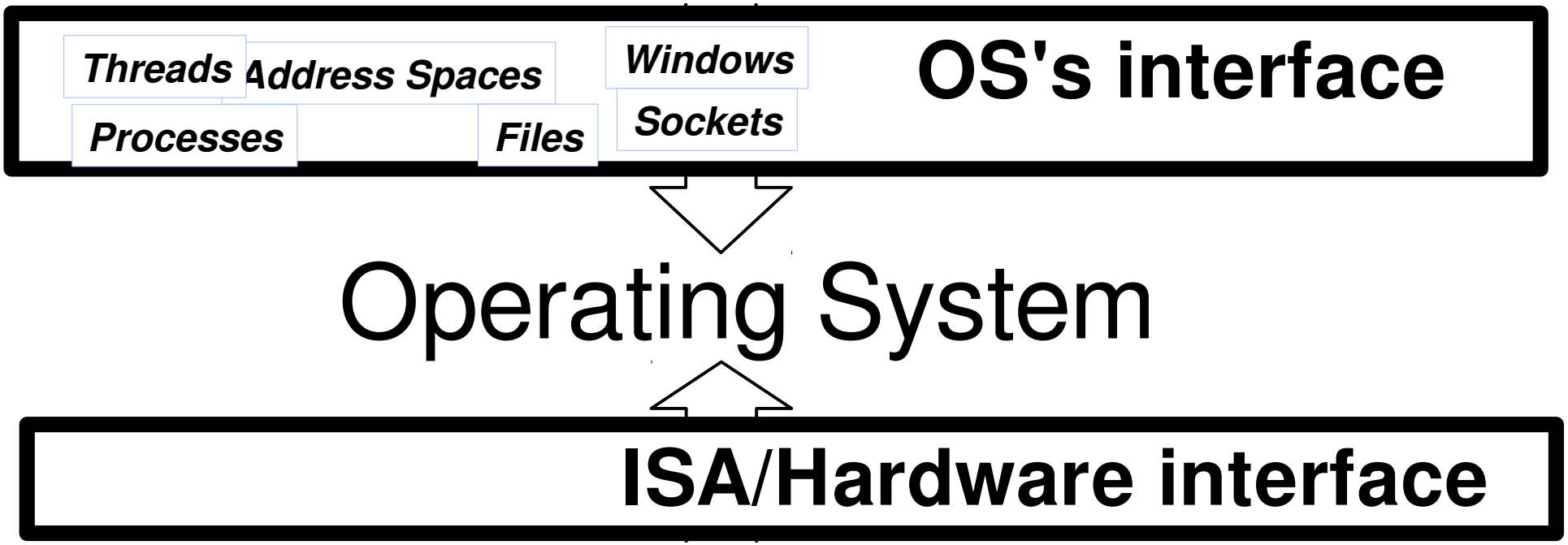
Application's "machine" *is* the OS

No hardware I/O details

- Future-proof

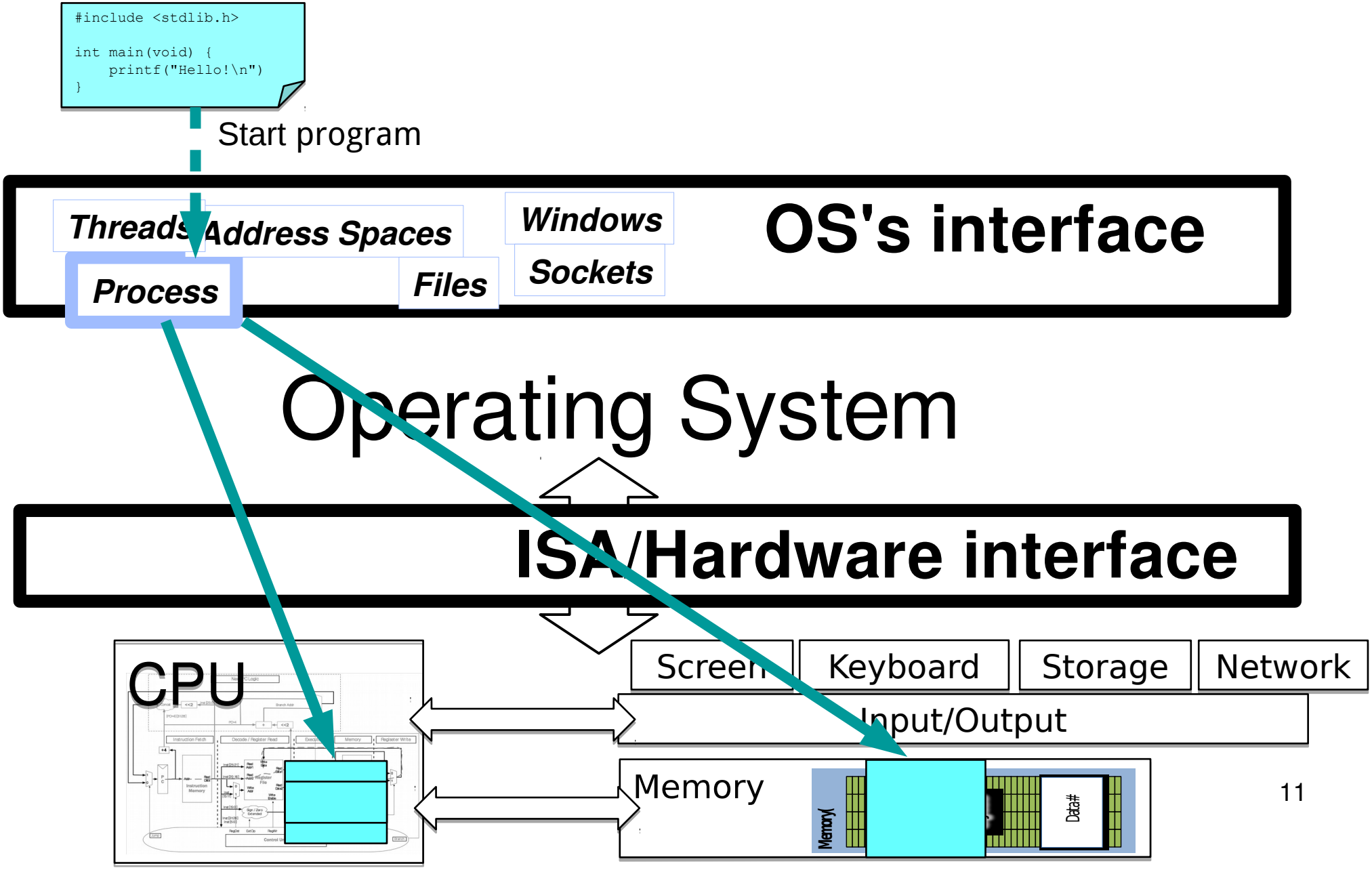
More featureful interfaces than HW

Abstract VM: OS View

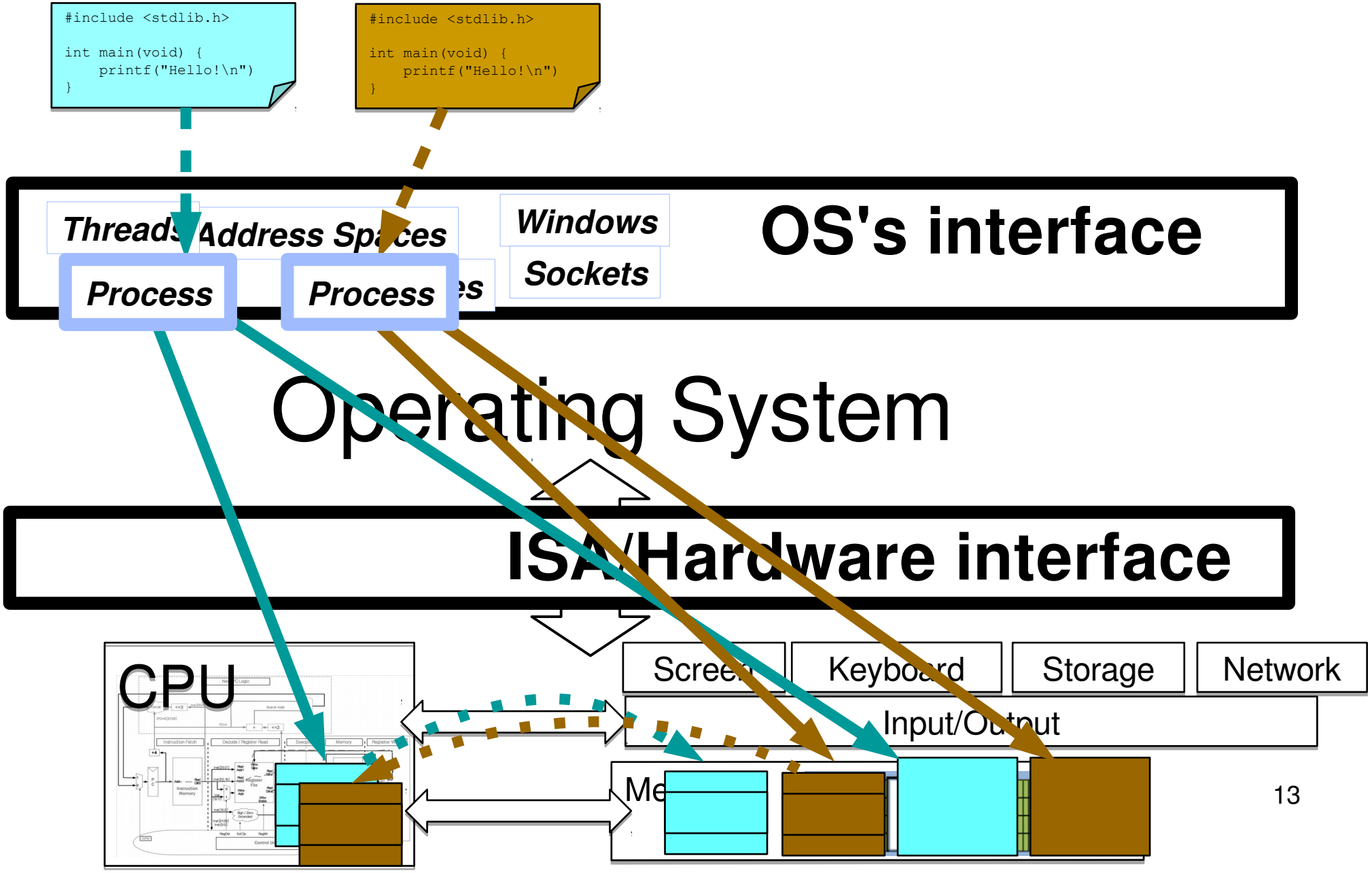


Operating system ***translates*** from one interface to the other

Abstraction: Program → Process



Multiple Processes: Context Switch



Abstraction: Input/Output

```
#include <stdlib.h>

int main(void) {
    printf("Hello!\n")
}
```

Read/Write/Open/Close

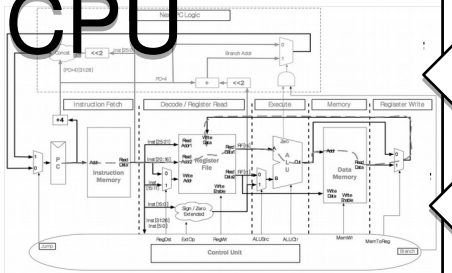
OS's interface

Threads *Address Spaces* *Windows* *Process* *Files* *Sockets*

Operating System

ISA/Hardware interface

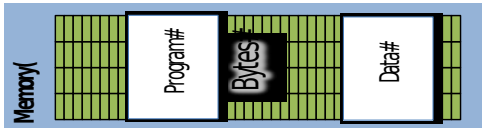
CPU



Screen | Keyboard | Storage | Network

Input/Output

Memory



Security and Protection

```
#include <stdio.h>
int main(void) {
  FILE* fh = open("/etc/shadow");
}
```

Permission Denied

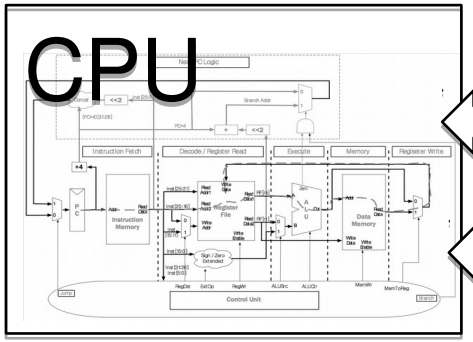
Threads ***Address Spaces*** ***Windows***

Processes ***Files*** ***Sockets***

OS's interface

Operating System

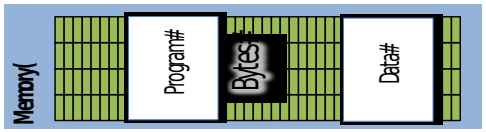
ISA/Hardware interface



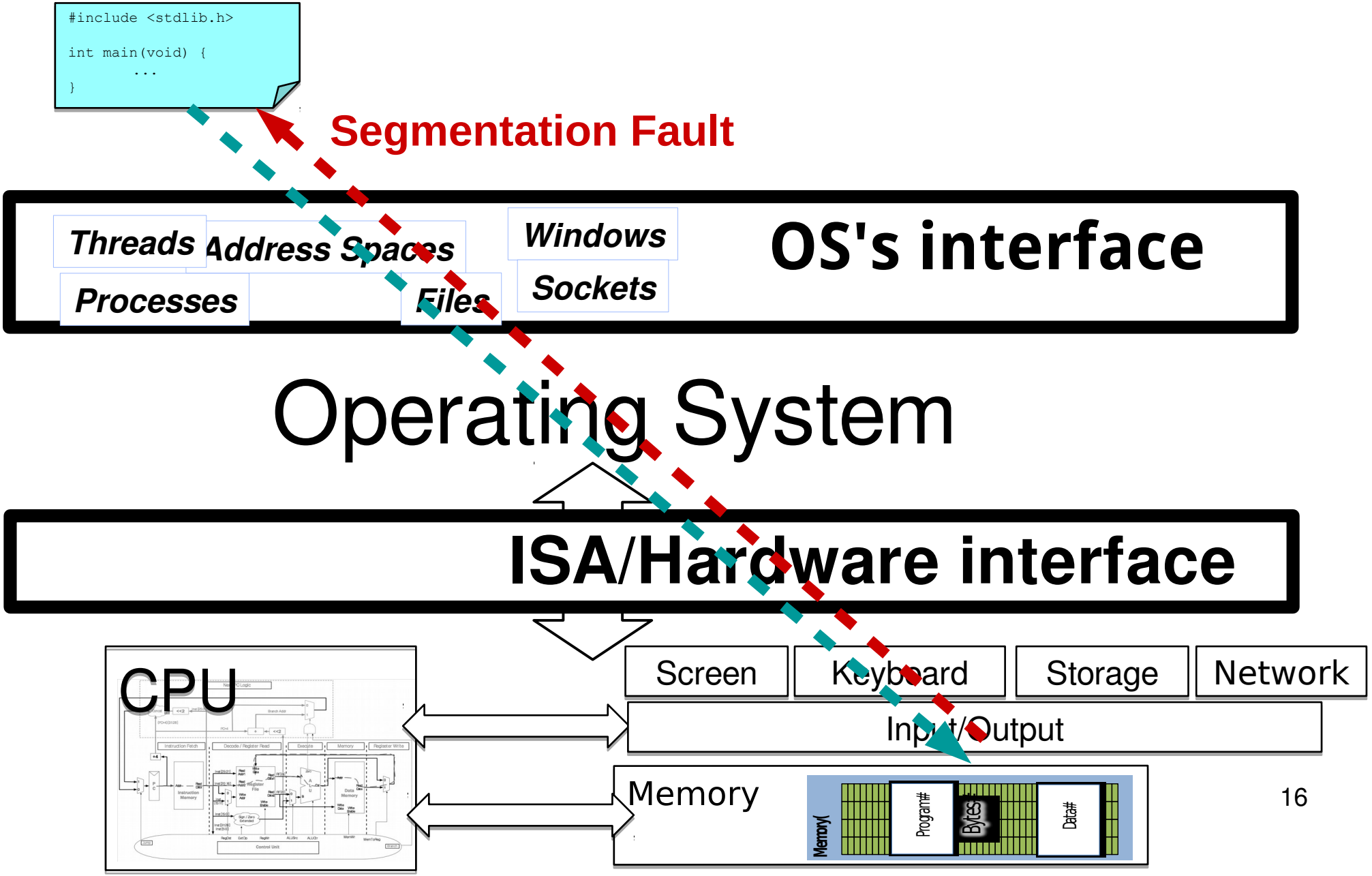
Screen | Keyboard | Storage | Network

Input/Output

Memory



Security and Protection



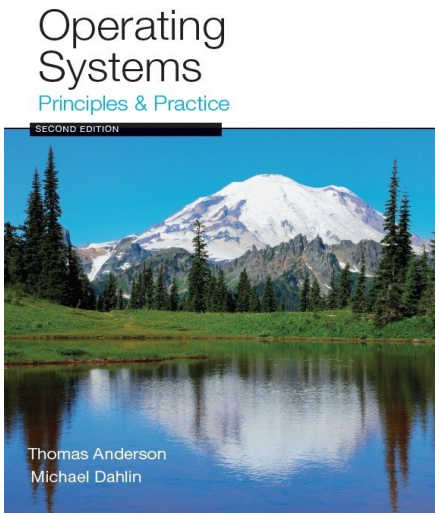
Interlude: Logistics

Infrastructure

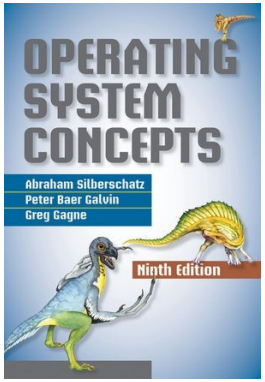
Website:
https://cs162.eecs.berkeley.edu/

Piazza:
https://piazza.com/berkeley/summer2015/cs162

Textbook



Anderson and Dahlin,
Operating Systems: Principles and Practice (2nd ed)



Recommended Text:
Silberchartz, Galvin, and Gagne
Operating Systems Concepts (9th ed)

Online supplements

- Linked off "readings"

Syllabus Summary

OS Interfaces: processes, I/O, sockets, …

Concurrency: threads, scheduling, locks, …

Address spaces: virtual memory, protection, …

Distributed systems: RPC, consistency, NFS, …

File systems: I/O devices, naming, caching, …

Reliability & Security: fault tolerance, security, …

Syllabus Summary

Project 1
Project 2
Project 3

OS Interfaces: processes, I/O, sockets, …

Concurrency: threads, scheduling, locks, …

Address spaces: virtual memory, protection, …

Distributed systems: RPC, consistency, NFS, …

File systems: I/O devices, naming, caching, …

Reliability & Security: fault tolerance, security, …

Learn by Doing

Homeworks: **Individual** assignments

- HW0: Tools, Autograding, C, Executables
- HW1: Simple Shell
- HW2: Malloc
- HW3: Trivial Filesystem

Projects: **Group**

- 1: Threads & Scheduling
- 2: User programs
- 3: Key-value store

Start HW0 ***Immediately***

Due **Friday** at **11:59 PM**

Gets your course account

Gets you a class repo repository on github

Setup VM for the course

Group Projects

4 person teams

- Never 5

Find teams soon (or we will for you)

Communicate!

- Everyone should do work – clear responsibility!
- Each group assigned a TA

Design documents

Project Grading

[20 pts] Initial Design + Design Review

[50 pts] Code (2 checkpoints)

[30 pts] Final Design

[0 pts] Peer Evaluation

Submission: **git push** to release branch

Grading

20% Midterm

20% Final

- Both during class time, see schedule on web

45% Projects (15% each)

15% Homeworks (~4% each)

Collaboration Policy

OK:

- Explaining a concept to someone in another group
- Discussing algorithms/testing strategies with other groups
- Helping debug someone else's code (in another group)
- Searching online for generic algorithms (e.g., hash table)

NOT OKAY:

- Sharing (pseudo-)code or test cases with another group
- Copying OR reading another group's code or test cases
- Copying OR reading online code or test cases from prior years
- Sharing homework code with other members of your project group

Confused? *Ask.* We can't help you after you cheat.

Late Policy

Assignments are due 11:59PM Pacific Time (except initial design documents).

- There is a "grace period"; please don't try to test it.

There are no slip days.

Submit up to 48 hours late: 20% deduction

- Not for initial design documents for projects

Submit more than 48 hours late: no credit

- We'll still tell you how you did, if you want...

Staff

Instructor Charles Reiss

- OH: MWF 1-2p 651 Soda

GSI Frank Nothaft

- Main discussion TA (both discussions)
- Starting Wednesday/Thursday
- Possible project TA
- OH: TuTh 1-2p 651 Soda

GSI Alex Yang

- Lead project/homework TA
- Possible project TA
- OH: MW 2-3p, Tu 5-6p 651 Soda

Readers: Nick Xu and Leon Wang

- Grade your final design docs; manual parts of homeworks; etc.

Topics for Today

What is (or is not) an operating system?

Some common operating system abstractions

... interrupted by Logistics

What is an operating system? (3)

Referee

- Resource sharing, protection, isolation

Illusionist

- Clean, easy abstractions

Glue: Common services

- Storage
- Window systems
- Authorization
- Networking

Unify theme: Complexity

All the definitions: operating systems are about ***hiding complexity***

Challenge: Complexity

Competing applications with

- Unexpected *failures*
- Even *malicious* applications

Varying hardware with

- Diverse, evolving interfaces
- Unexpected failures

→ **Not feasible to test all configurations!**

Complexity Example: Mars Pathfinder

Lots of types of hardware:

- radios, scientific instruments, batteries, solar panels, and locomotion equipment

Reliability requirements:

- No reset button: must reboot self automatically
- Must **always** be able to receive commands from Earth

Remote debugging?

- Software will **crash** (e.g. instrument breaks – is that handled?)
- Need to debug from earth

Time critical functions

- Stop before hitting something
- Earth communication

Complexity: Variety

Different CPUs

- Pentium, PowerPC, ColdFire, ARM, MIPS

Different amounts of memory, disk, …

Different types of devices

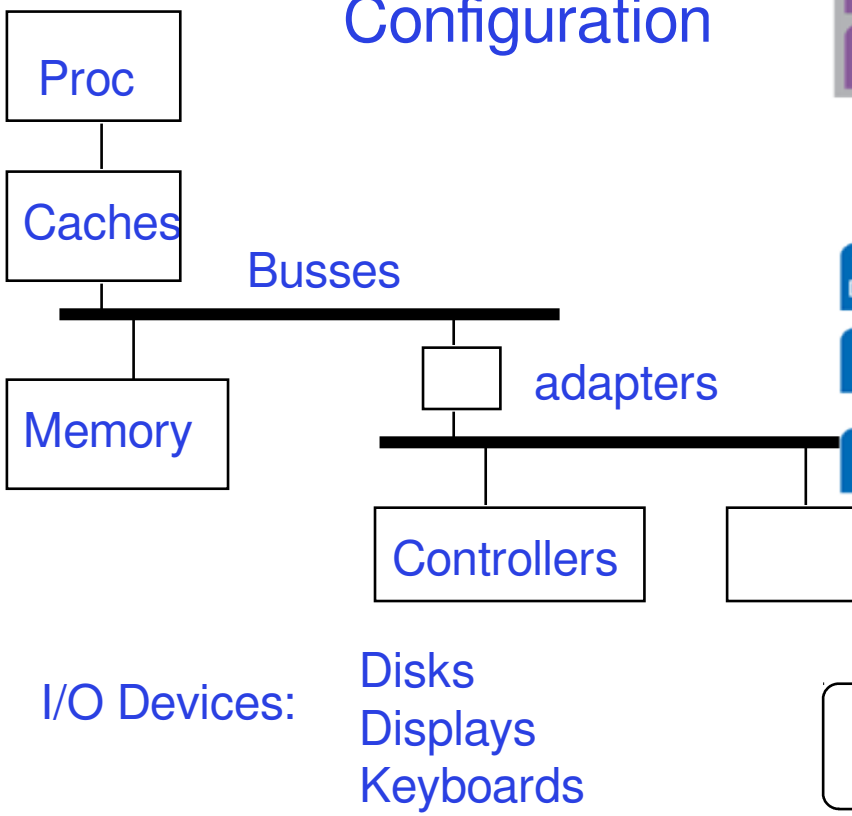
- Mice, Keyboards, Sensors, Cameras, …

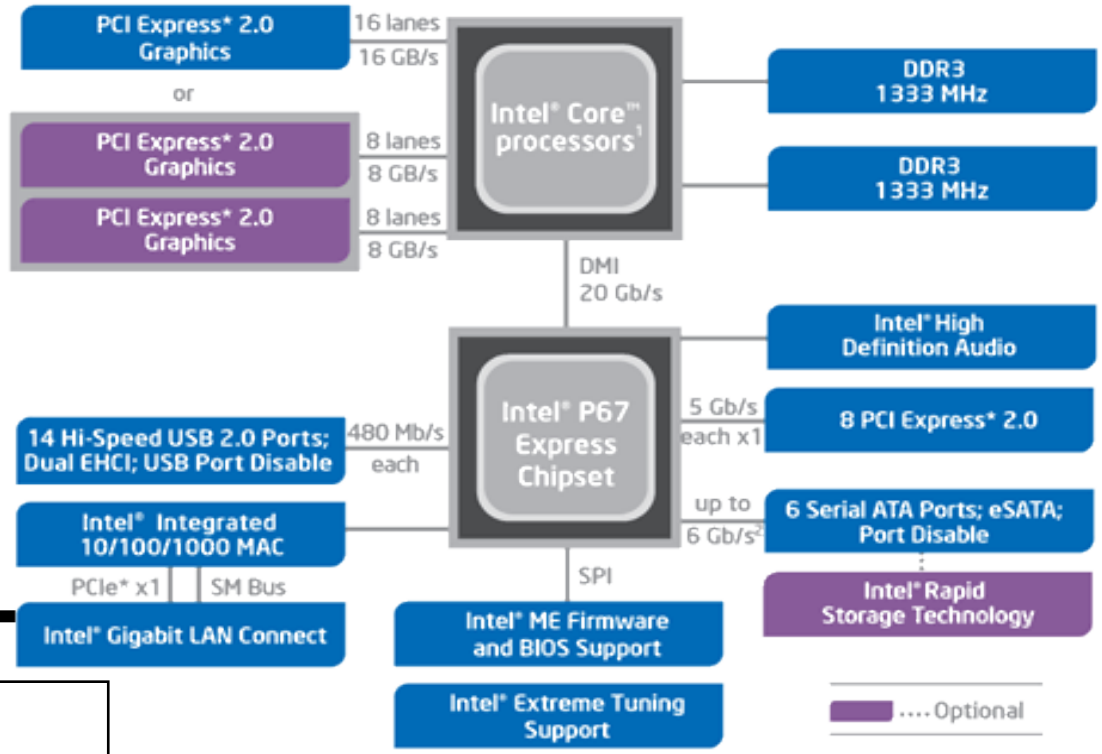
Different networking equipment

- Cable, DSL, Wireless, Ethernet, …

Complexity: Modern Hardware

SandyBridge I/O Configuration





I/O Devices:

Disks
Displays
Keyboards

Networks

Unifying theme: Helping the application programmer

Write once for lots of hardware

Without reimplementing common functionality

Run alongside other programs

One program crashing doesn't crash everything

Break

Abstract VM Goals

Application

Virtual Machine Interface

Operating System

Physical Machine Interface

Hardware

OS Goals:

- **Remove** hardware/software **quirks** *(fight complexity)*
- **Optimize** for convenience, utilization, reliability, ... *(help programmer)*

Big questions for any area of OS (file systems, scheduling, ...):

- What hardware interface to handle? (**physical reality**)
- What software interface to provide? (**nicer abstraction**)

Side Note: Virtual Machines

Definition: Software emulation of *abstract machine*

- Programs believe *they own the machine*
- Simulated "hardware" has the *features we want*

Two types:

- *Process VM*: run a single program (typical OS)
- *System VM:* run entire OS + its applications (Virtualbox, VMWare, Parallels, Xen, ...)

Process VM

Each process thinks …

- It has all the memory, CPU time, devices, …
- All devices have the same interface
- Device interfaces more powerful than raw hardware:
  - Reliable TCP/IP instead of Ethernet card
  - Windowing system instead of bitmapped display

Fault isolation:

- Can't directly impact other processes
- Bugs won't crash everything

Portability: Write application for OS, not hardware

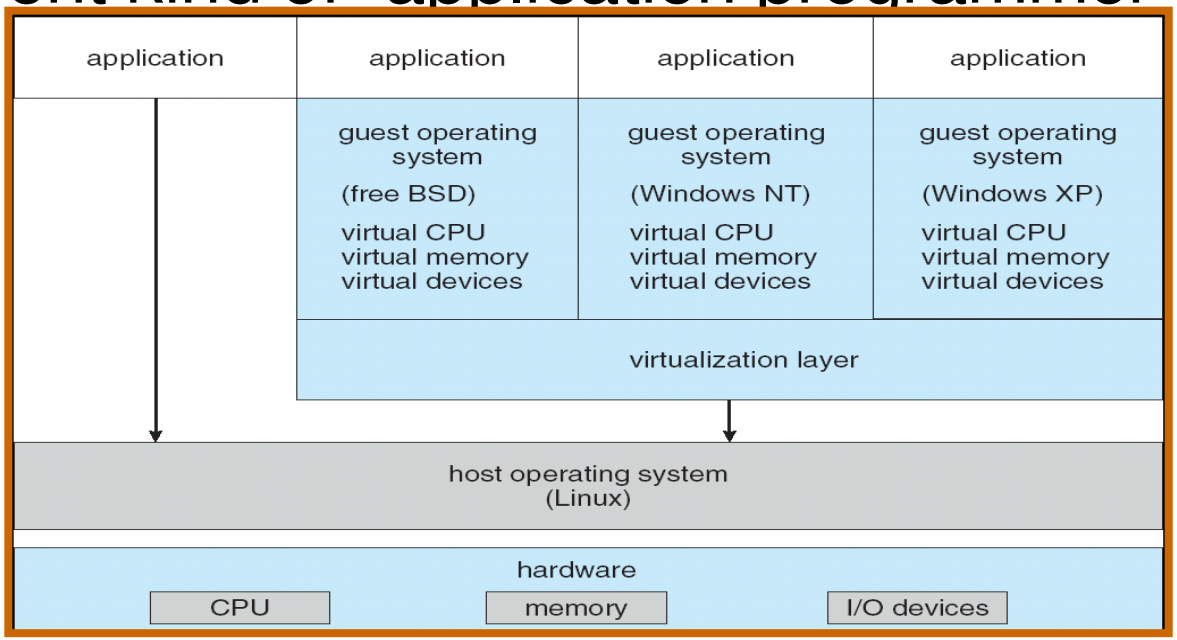
System VMs: Layers of OSs

Run entire OSs

- Good for OS development, portability

Interface closer to raw HW

- to aid porting/running OSs built to use raw HW
- different kind of "application programmer" to design for



What is an operating system? (4)

Always:

- Memory management
- I/O management
- CPU scheduling
- Communication?

Sometimes:

- Filesystems
- ? Multimedia support
- ? User interface
- ? Internet browser

What is an operating system? (5)

No universal definition

One approximation: "Everything a vendor ships when you order an OS"

- But varies a lot!

"The one program running at all times" is the **kernel**. The rest are

- **System programs** (ship with operating system) and
- **Application programs**

Topics for Today

What is (or is not) an operating system?

Some common operating system abstractions

... interrupted by Logistics

OS Goal: Protecting Processes

Most important task of OSs?

Run multiple applications, and...

Keep them from crashing the OS

Keep them from crashing each other

[Keep parts of OS from crashing other parts?]

Protecting Processes

Available mechanisms (hardware interfaces):

- Address translation
- Dual mode operation

Simple policy: Programs are not allowed to read or write memory of other programs or OS

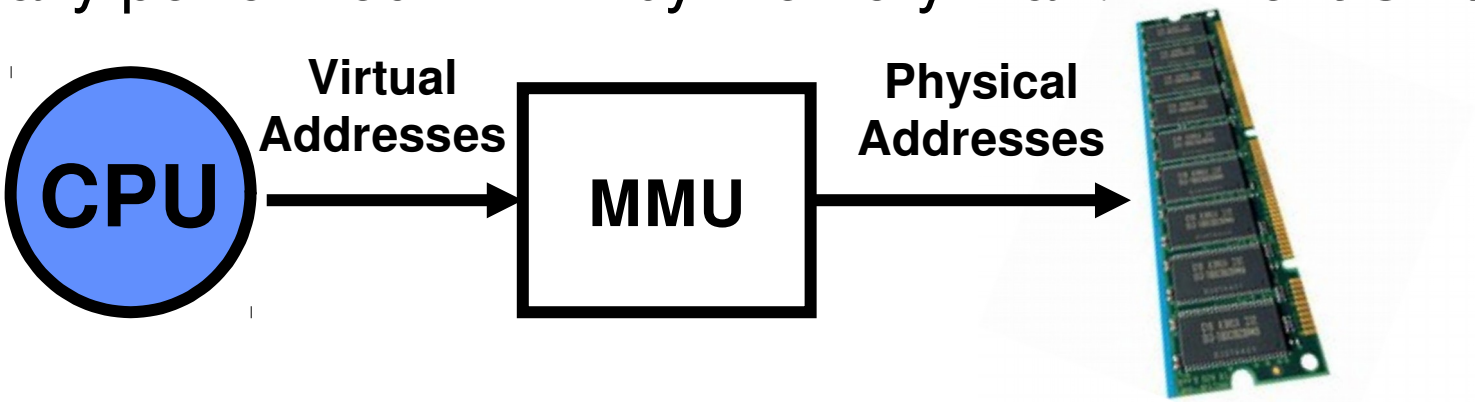
Address Translation

Abstraction: Address space

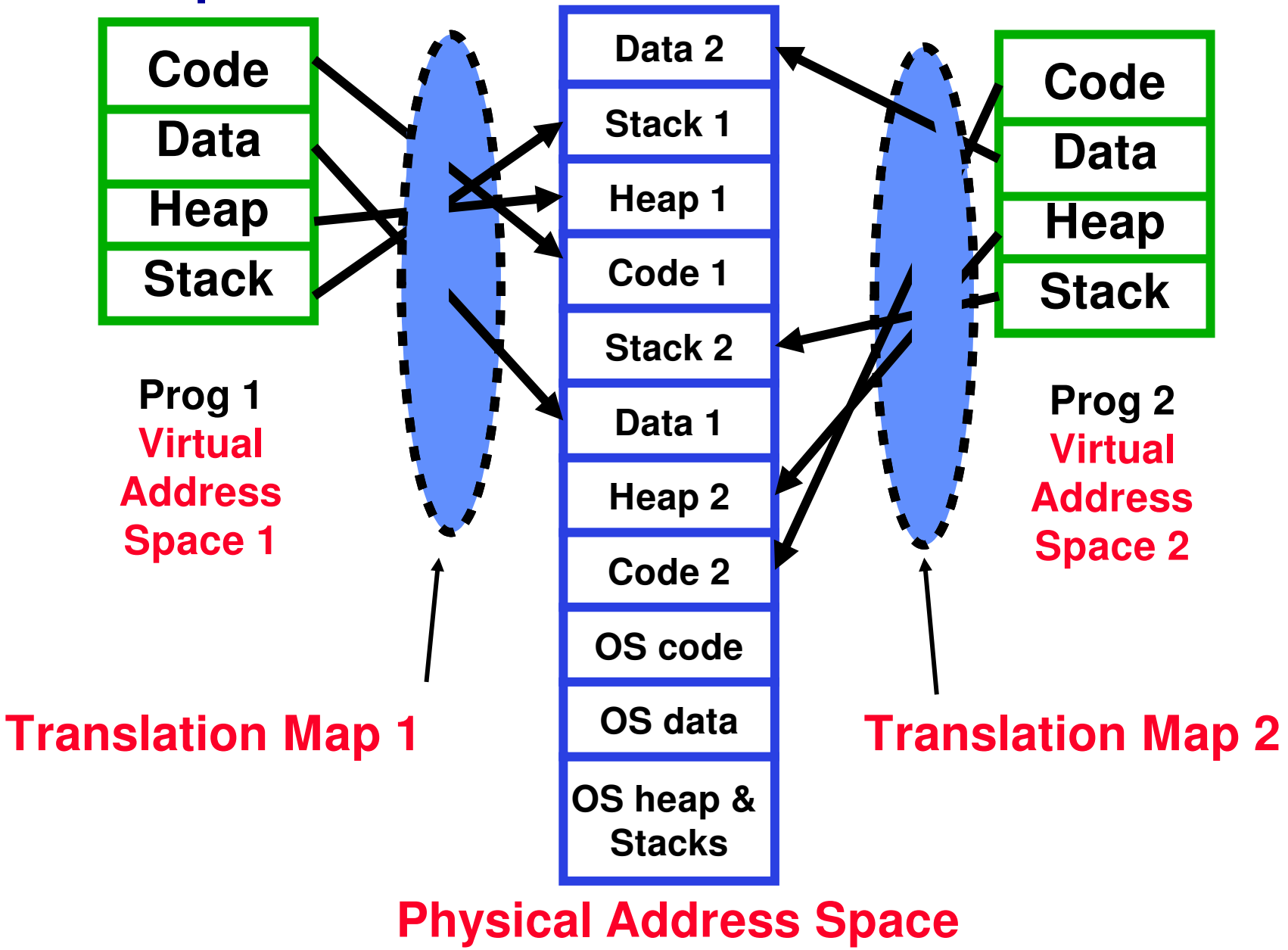
- Group of memory addresses usable by something
- Each process, kernel have *different address spaces*
- Illusion of owning all memory

Mechanism: Address Translation

- Translate virtual addresses (emitted by CPU) to physical addresses
- Usually performed in HW by Memory Management Unit (MMU)

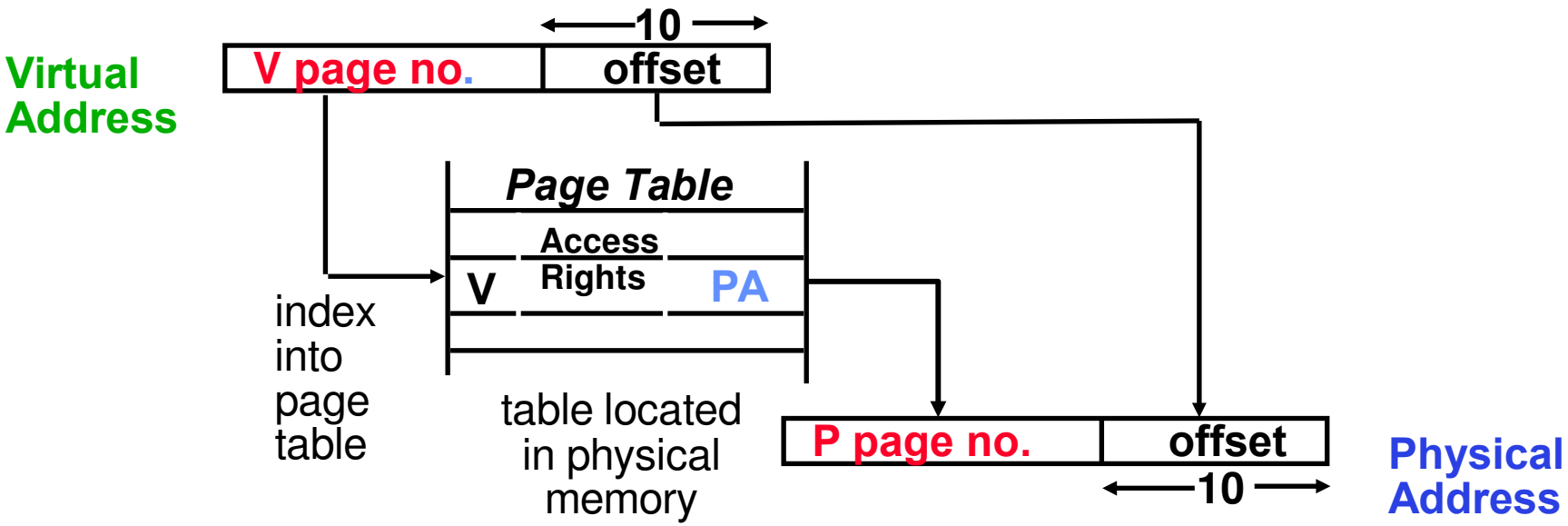


Example of Address Translation



Address Translation Details

(Covered in 61C?)



Implements policy? Can't read/write if no translation

Problem:

- **Can user change page table pointer?**

Dual Mode Operation

Hardware provides at least two modes:

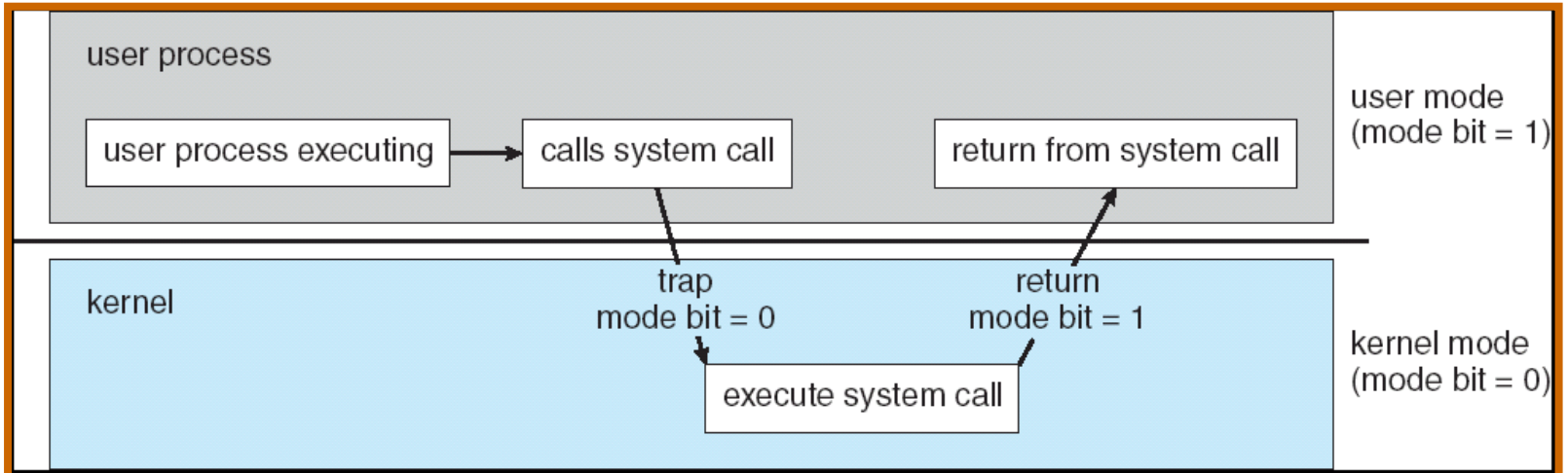
- "**Kernel**" (or "supervisor" or "protected")
- "**User**" mode

Some operations ***prohibited*** in user mode

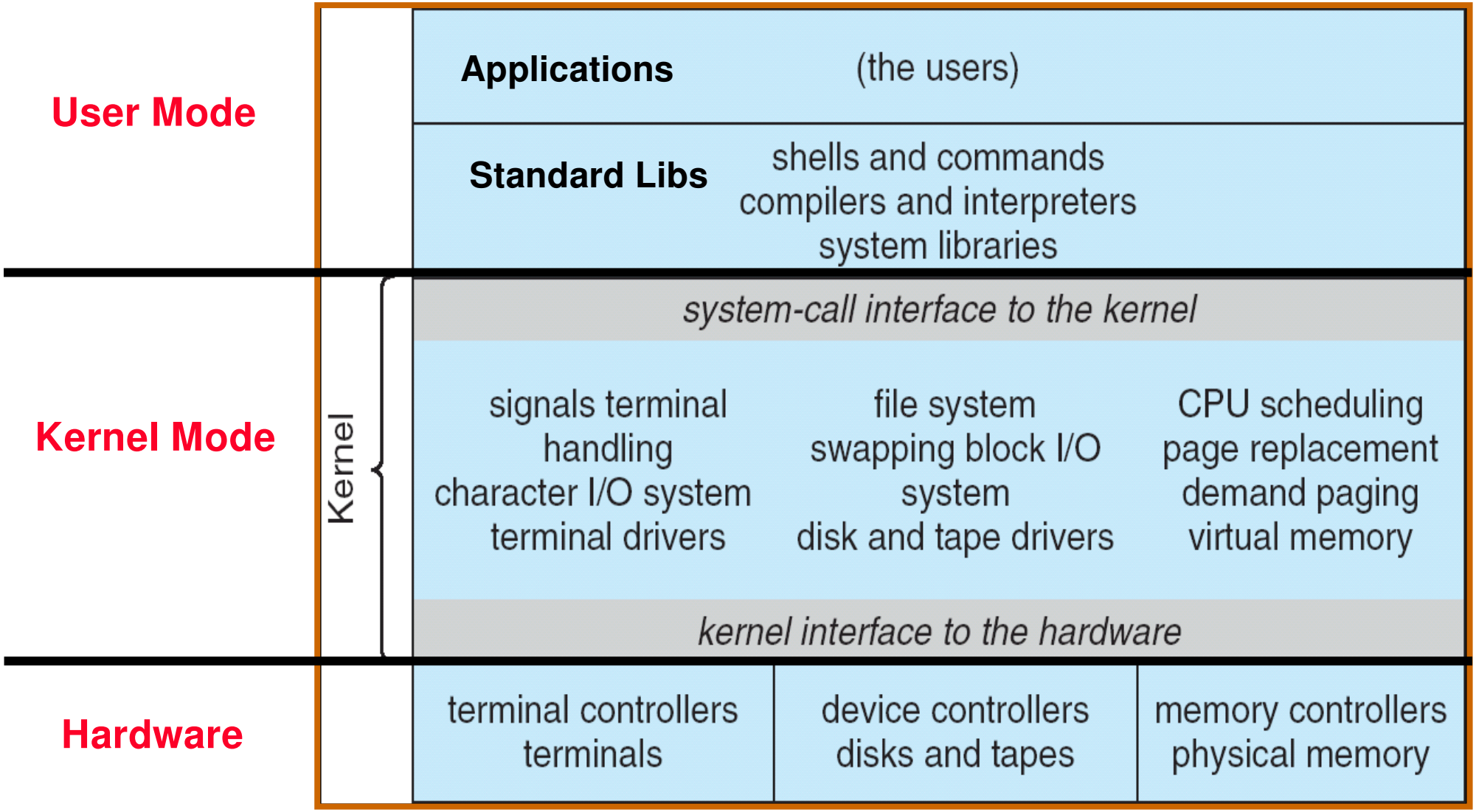
- e.g. changing the page table pointer

Controlled transitions from user to kernel mode

- Systems calls, interrupts, exceptions, ...



UNIX System Structure



Conclusion

Operating systems

- provide a virtual machine abstraction to handle diverse hardware
- coordinate resources and protect users from each other
- simplify application development by providing standard services
- can provide an array of fault containment, fault tolerance, and fault recovery