

Operating Systems 2019

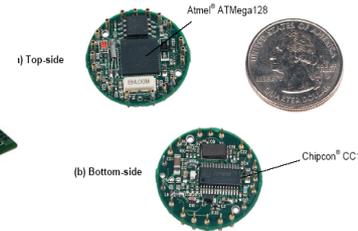
Introduction

Michael O'Boyle
mob@inf.ed.ac.uk

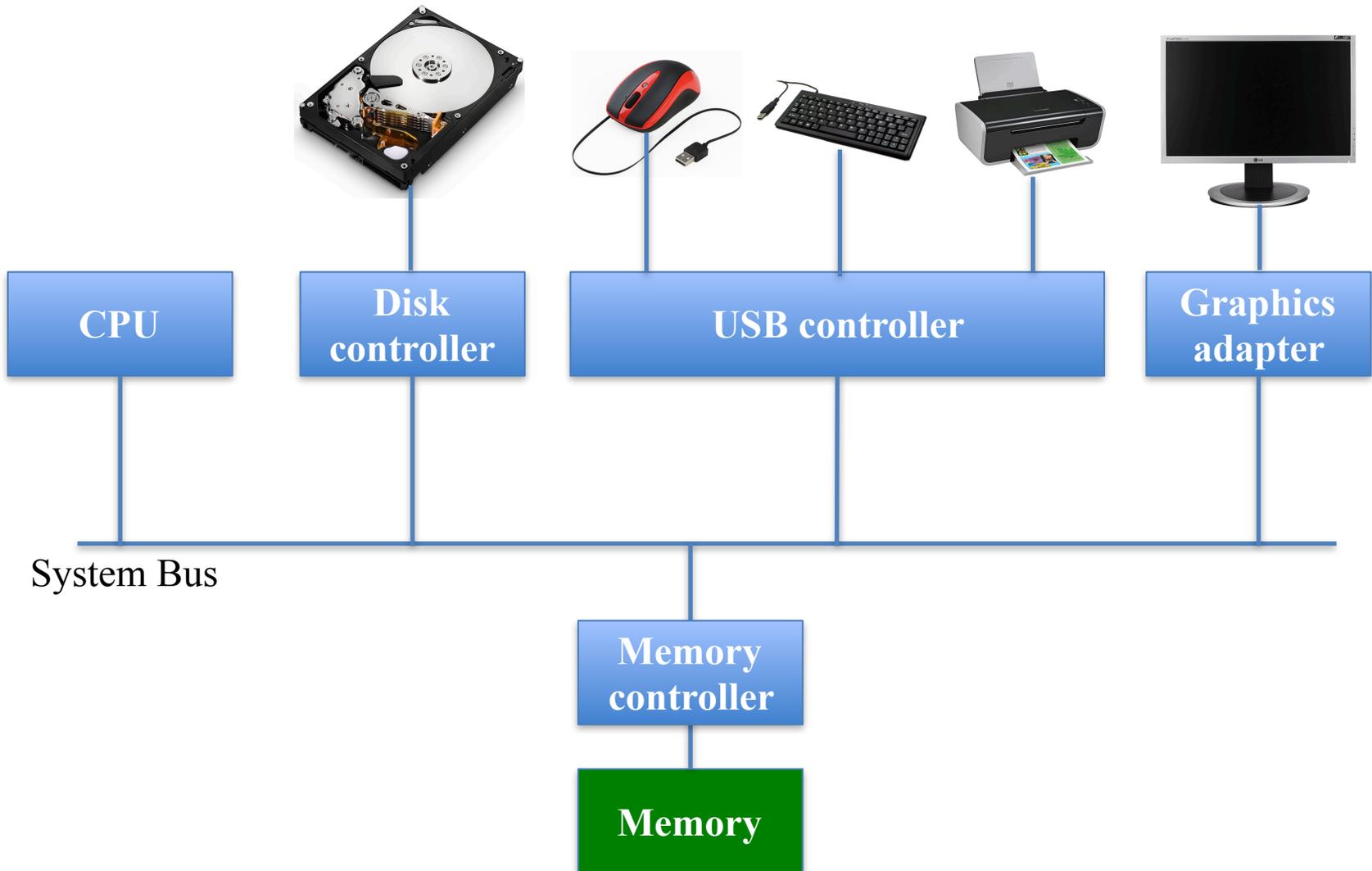
Overview

- Introduction
- Definition of an operating system
 - Hard to pin down
- Historical look
- Key functions
 - Timesharing
 - Multitasking
- Various types of OS
 - Depends on platform and scenario

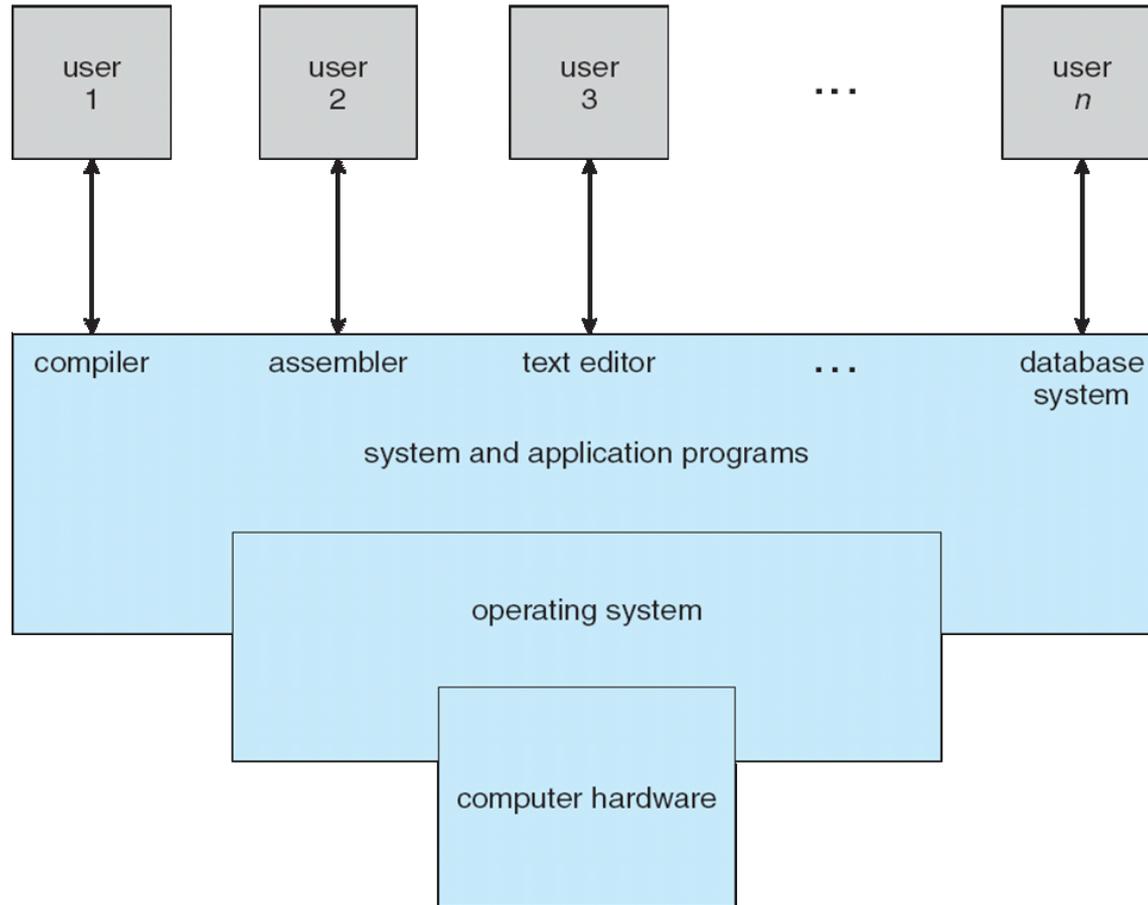
Computing systems are everywhere



Modern computer system



Four Components of a Computer System

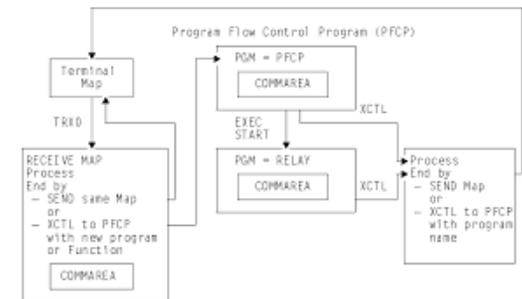


What is an Operating System?

- A big program
 - Linux kernel has 20M lines of code
- A program that
 - **manages** a computer's hardware
- A program that
 - acts an **intermediary** between the user of a computer and computer hardware

Operating System Definition

- OS is a **resource allocator**
 - Manages all resources
 - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
 - Controls execution of programs to prevent errors and improper use of the computer



Operating System Definition (Cont.)

- No universally accepted definition
- “Everything a vendor ships when you order an operating system” is a good approximation
 - But varies wildly
- “The one program running at all times on the computer” is the **kernel**.
 - **Not the case in bare-metal embedded systems**
- Everything else is either
 - a system program (ships with the operating system) , or
 - an application program.

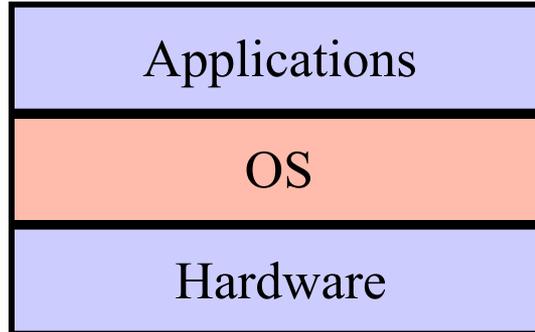


Some goals of operating systems

- Simplify the execution of user programs and make **solving user problems easier**
- Use computer hardware **efficiently**
 - Allow sharing of hardware and software resources
- Make application software **portable and versatile**
- Provide isolation, security and protection among user programs
- Improve overall system reliability
 - error confinement, fault tolerance, reconfiguration



The traditional Picture



- “The OS is everything you don’t need to write in order to run your application”
 - This depiction invites you to think of the OS as a library; we’ll see that
- In some ways, it is:
 - all operations on I/O devices require OS calls (*syscalls*)
- In other ways, it isn't:
 - you use the CPU/memory without OS calls
 - it intervenes without having been explicitly called

The OS and Hardware

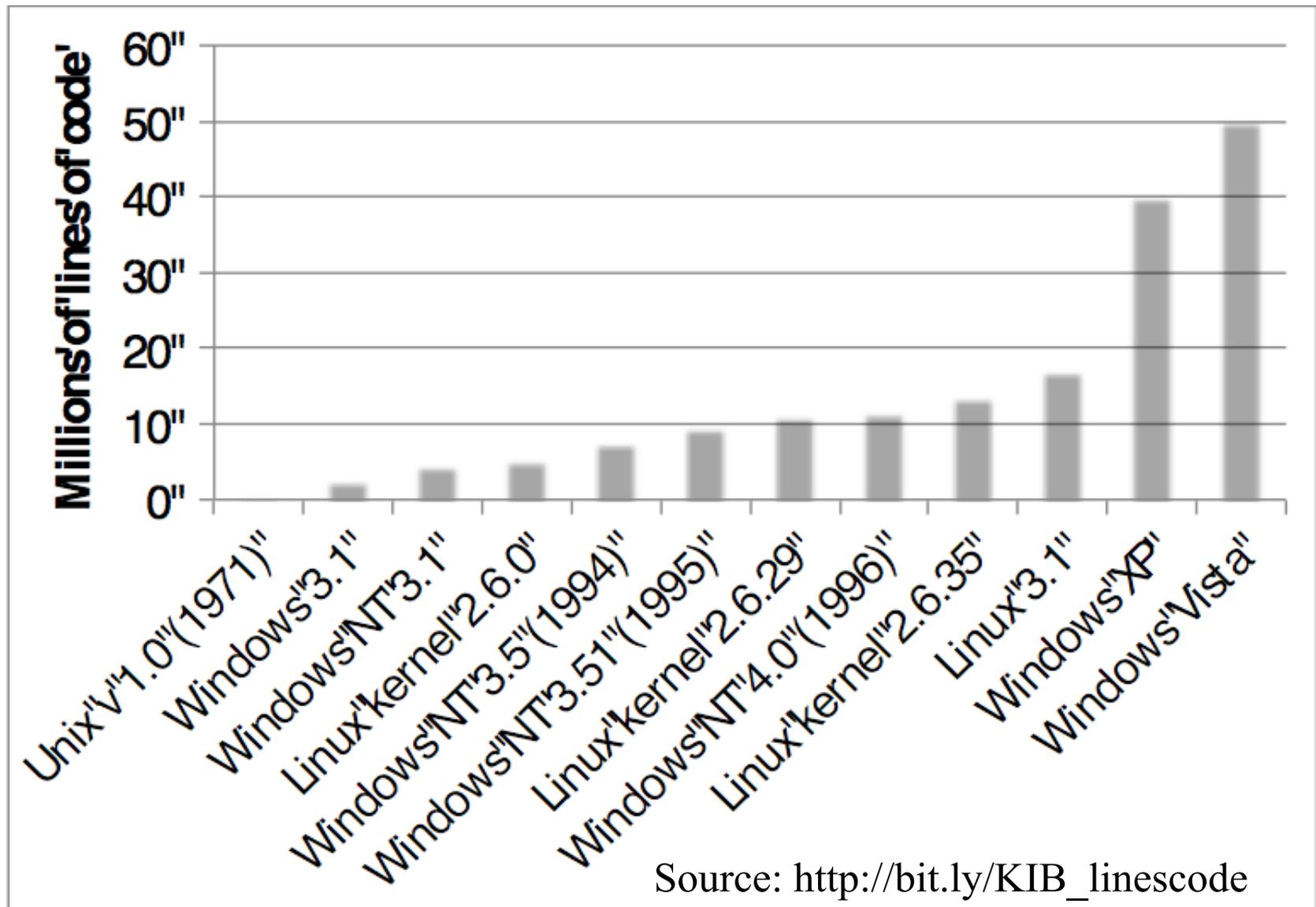
- An OS **mediates** programs' access to hardware resources (*sharing and protection*)
 - computation (CPU)
 - volatile storage (memory) and persistent storage (disk, etc.)
 - network communications (TCP/IP stacks, Ethernet cards, etc.)
 - input/output devices (keyboard, display, sound card, etc.)
- The OS **abstracts** hardware into **logical resources** and well-defined **interfaces** to those resources (*ease of use*)
 - processes (CPU, memory)
 - files (disk)
 - programs (sequences of instructions)
 - sockets (network)

Why Bother with an OS?

- Application benefits
 - programming **simplicity**
 - see high-level abstractions (files) instead of low-level hardware details (device registers)
 - abstractions are **reusable** across many programs
 - **portability** (across machine configurations or architectures)
 - device independence: 3com card or Intel card?
- User benefits
 - **safety**
 - program “sees” its own virtual machine, thinks it “owns” the computer
 - OS **protects** programs from each other
 - OS **fairly multiplexes** resources across programs
 - **efficiency** (cost and speed)
 - **share** one computer across many users
 - **concurrent** execution of multiple programs

Hardware/Software Changes with Time

Software Complexity Increases

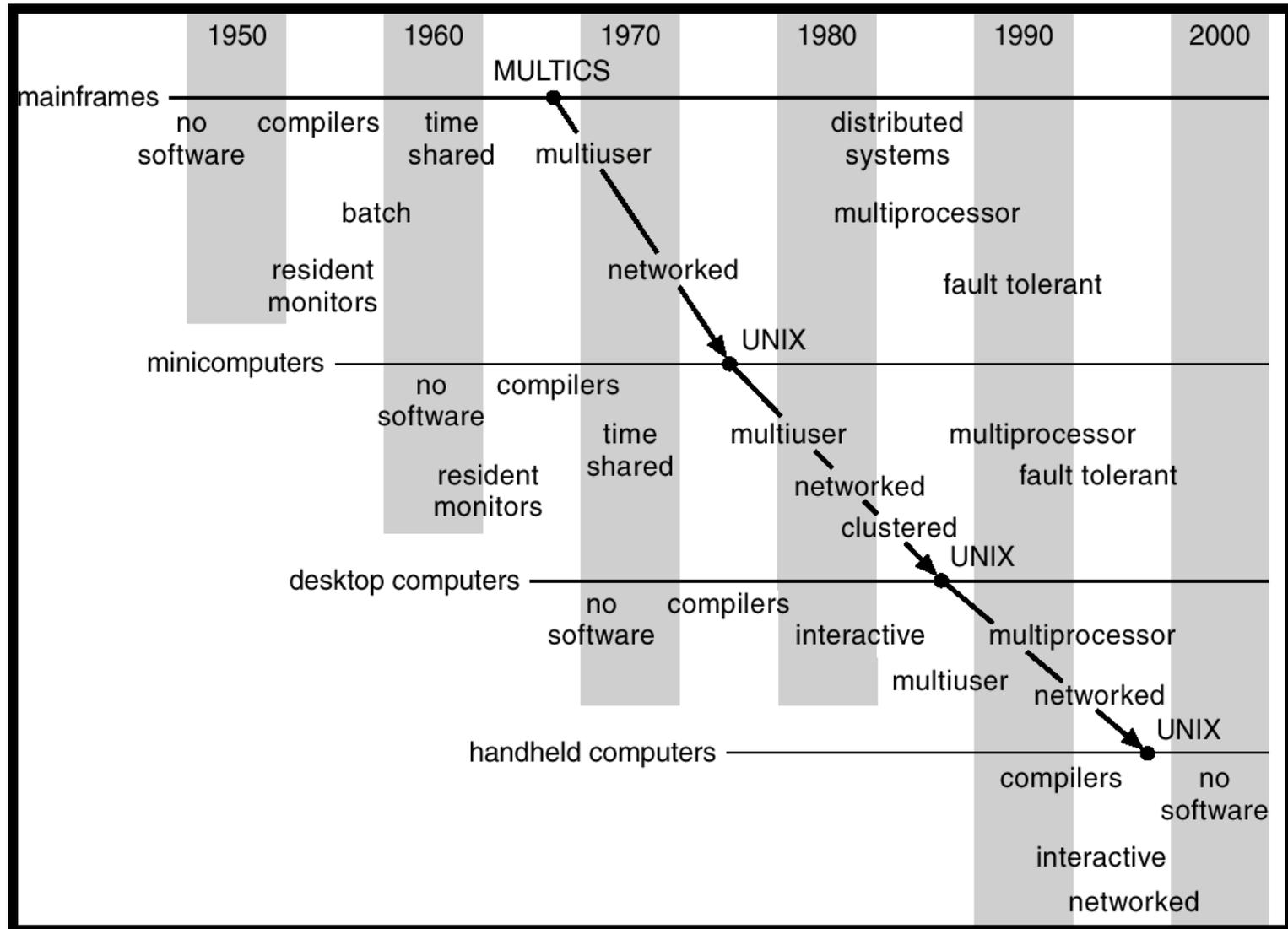


Source: http://bit.ly/KIB_linescode

Hardware/Software Changes with Time

- 1960s: mainframe computers (IBM)
- 1970s: minicomputers (DEC)
- 1980s: microprocessors and workstations (SUN), local-area networking, the Internet
- 1990s: PCs (rise of Microsoft, Intel, Dell), the Web
- 2000s:
 - Internet Services / Clusters (Amazon)
 - General Cloud Computing (Google, Amazon, Microsoft)
 - Mobile/ubiquitous/embedded computing (iPod, iPhone, iPad, Android)
- 2010s: sensor networks, “data-intensive computing,” computers and the physical world
- 2020: exascale, IoT ??

Progression of Concepts and Form Factors



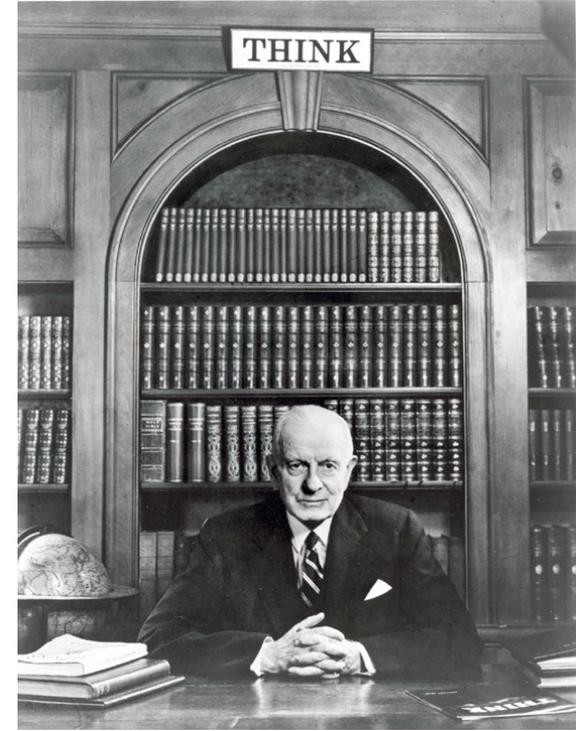
An OS History Lesson

- Operating systems are the result of a 60 year long evolutionary process
 - They were born out of need
- Examine their evolution
- Explains what some of their functions are, and why

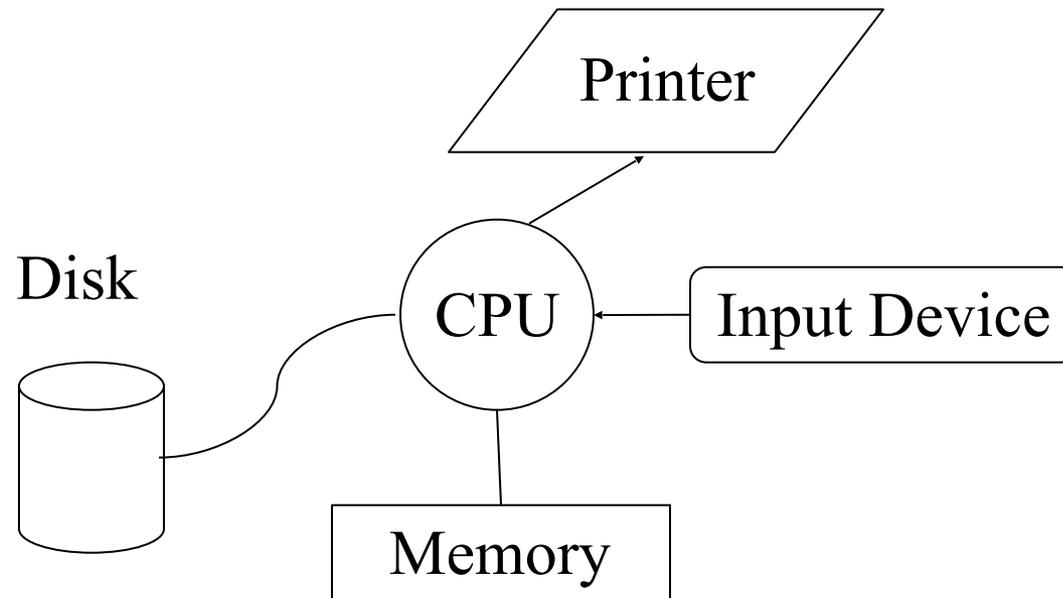


Early days

- 1943
 - T.J. Watson (created IBM):
“I think there is a world market for maybe five computers.”
- Fast forward ... 1950
 - There are maybe 20 computers in the world
 - They were unbelievably expensive
 - Machine time is considerably more valuable than person time!
 - Ergo: efficient use of the hardware is paramount
 - Operating systems are born
 - They carry with them the vestiges of these economic assumptions



Simplified early computer

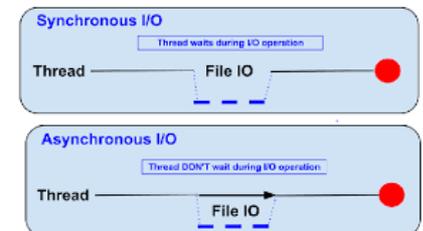


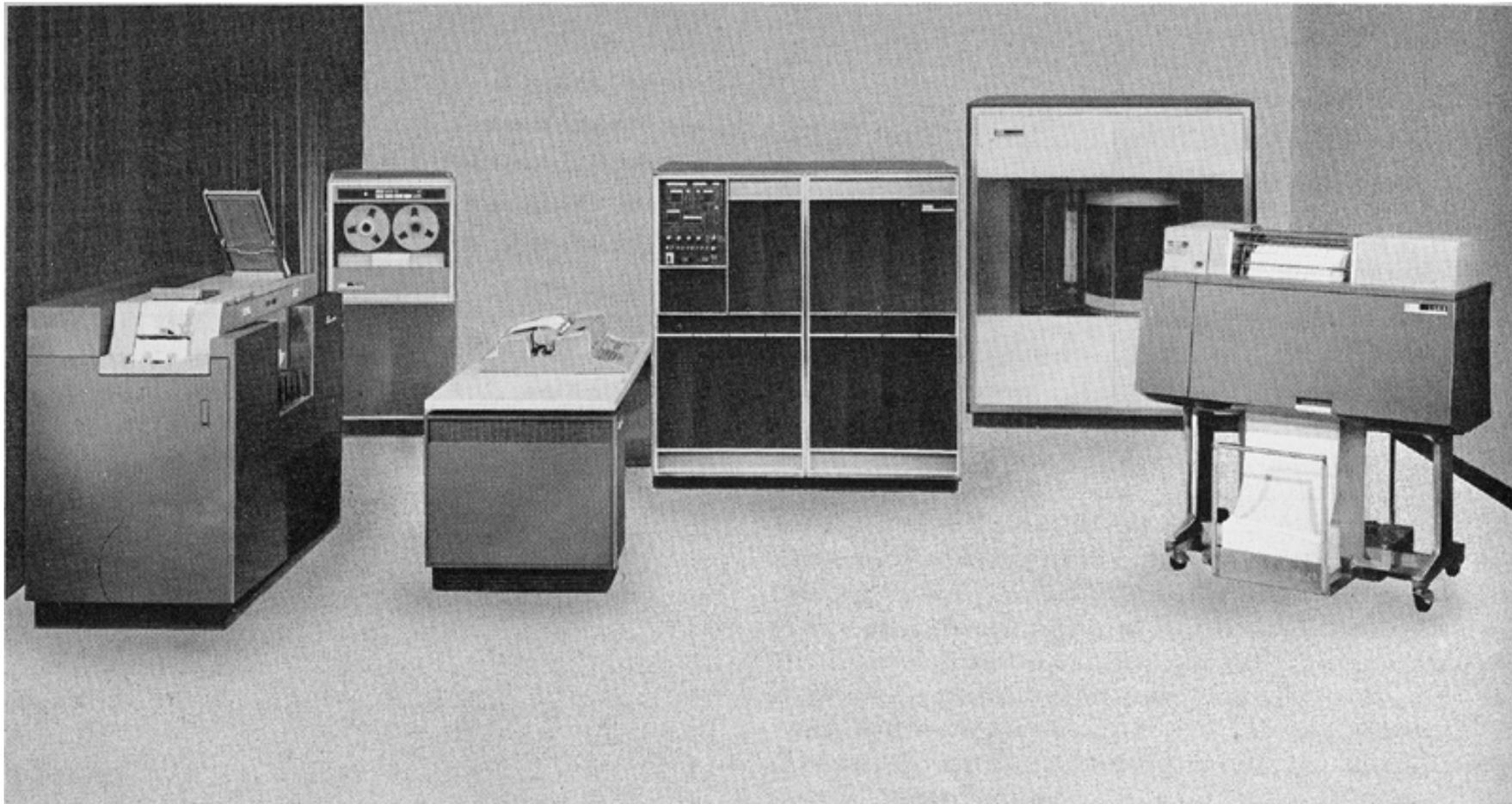
The OS as a linked library

- In the very beginning...
 - OS was just a library of code that you linked into your program; programs were loaded in their entirety into memory, and executed
 - “OS” had an “API” that let you control the disk, control the printer, etc.
 - Interfaces were literally switches and blinking lights
 - When you were done running your program, you’d leave and turn the computer over to the next person
- Not so very different from some embedded devices today

Asynchronous I/O

- The disk was really slow
- Add hardware so that the disk could operate without tying up the CPU
 - Disk controller
- Programmers could now write code that:
 - Starts an I/O
 - Goes off and does some computing
 - Checks if the I/O is done at some later time
- Upside
 - Helps increase (expensive) CPU utilization
- Downsides
 - It's hard to get right
 - The benefits are job specific

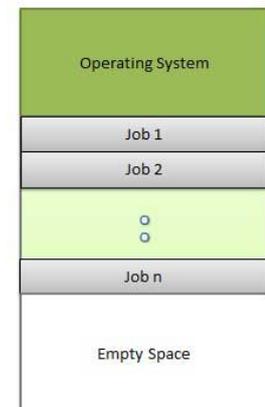


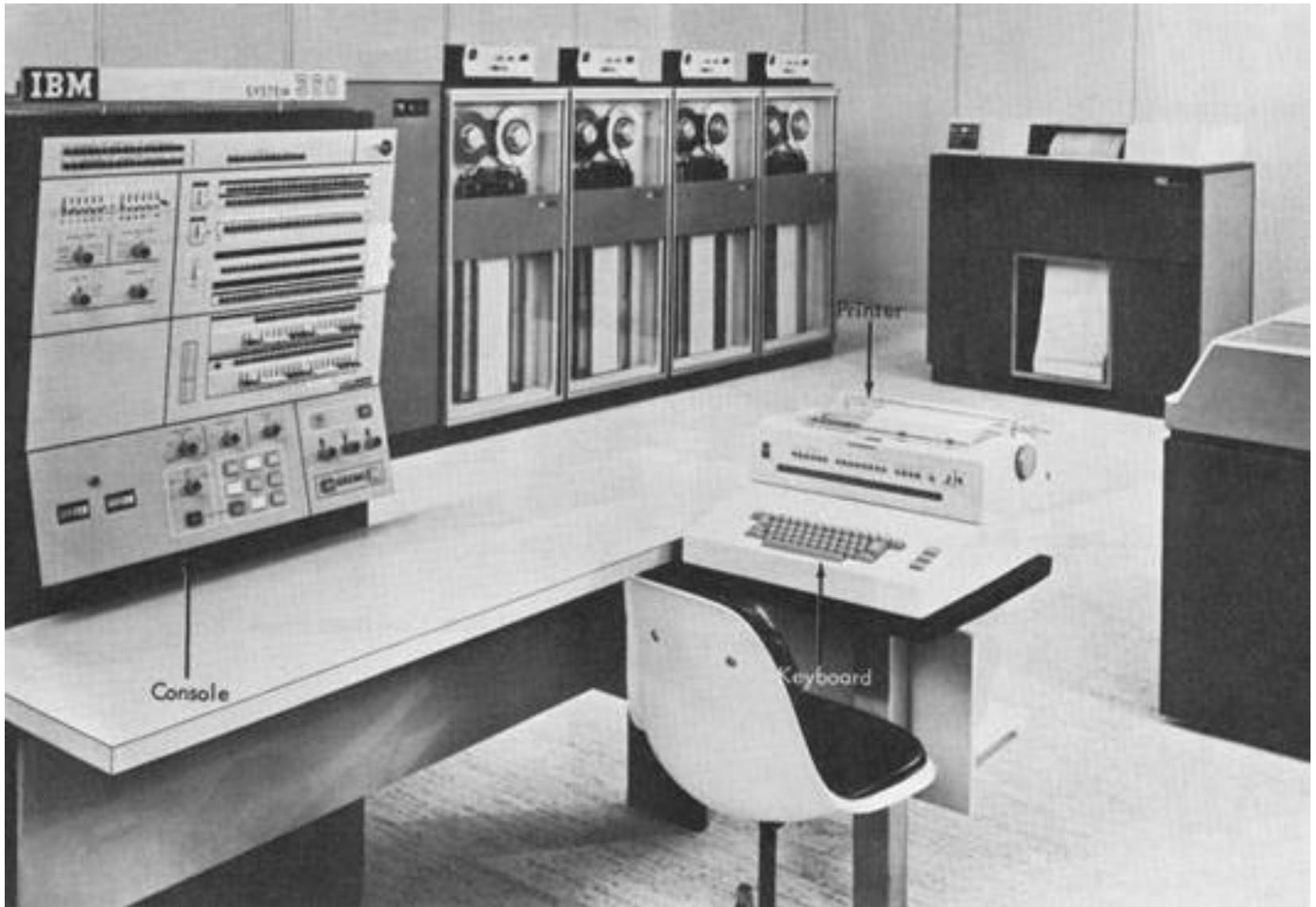


IBM 1401

Multiprogramming

- To further increase system utilization, **multiprogramming** OSs were invented
 - keeps multiple runnable jobs loaded in memory at once
 - overlaps I/O of one job with computing of another
 - while one job waits for I/O completion, another job uses the CPU
- Can get rid of asynchronous I/O within individual jobs
 - Life of application programmer becomes simpler; only the OS programmer needs to deal with asynchronous events
- How do we tell when devices are done?
 - Interrupts
 - Polling
- What new requirements does this impose?





IBM System 360

Timesharing



- To support interactive use, create a **timesharing OS**:
 - multiple terminals into one machine
 - each user has illusion of entire machine to him/herself
 - optimize response time, perhaps at the cost of throughput
- Timeslicing
 - divide CPU equally among the users
 - if job is truly interactive (e.g., editor), then can jump between programs and users faster than users can generate work
 - permits users to interactively view, edit, debug running programs
- Multics system (operational 1968) was the first large timeshared system
 - nearly all OS concepts can be traced back to Multics

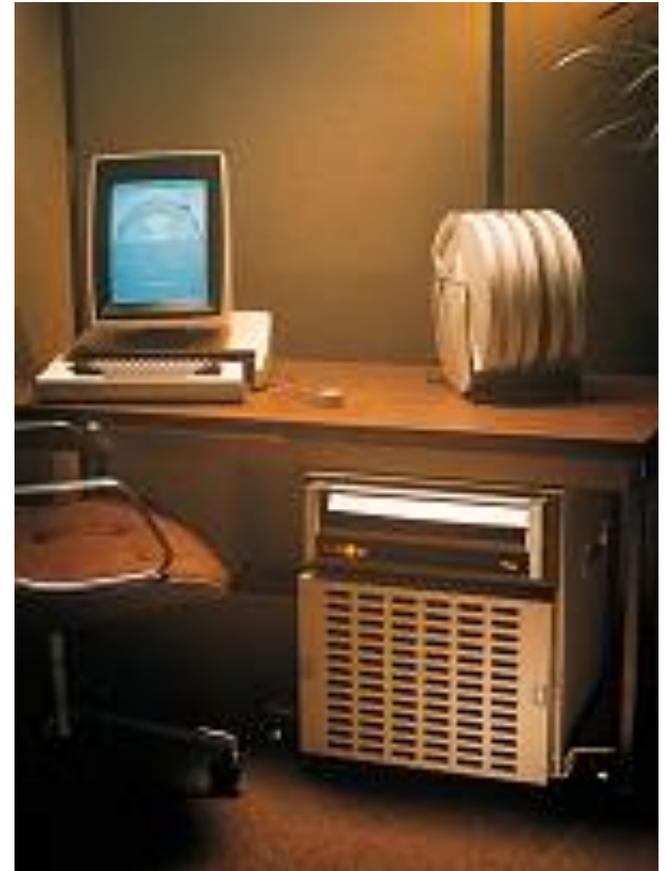
Parallel Systems



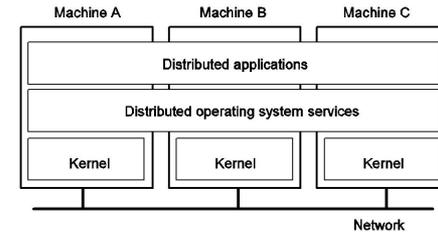
- Some applications can be written as multiple parallel **threads** or **processes**
 - can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs [Burroughs D825, 1962]
 - need OS and language primitives for dividing program into multiple parallel activities
 - need OS primitives for fast communication among activities
 - degree of speedup dictated by communication/computation ratio
- Many flavors of parallel computers today
 - Multi-cores – all(ish) processors are parallel
 - SMPs (symmetric multi-processors)
 - MPPs (massively parallel processors)
 - NOWs (networks of workstations) –less common
 - Massive clusters (Google, Amazon.com, Microsoft)
 - Heterogeneous accelerators eg GPUs

Personal Computing

- Primary goal was to enable new kinds of applications
- Bit mapped display [Xerox Alto, 1973]
 - new classes of applications
 - new input device (the mouse)
- Move computing near the display
 - why?
- Window systems
 - the display as a managed resource
- Local area networks [Ethernet]
 - why?
- Effect on OS?



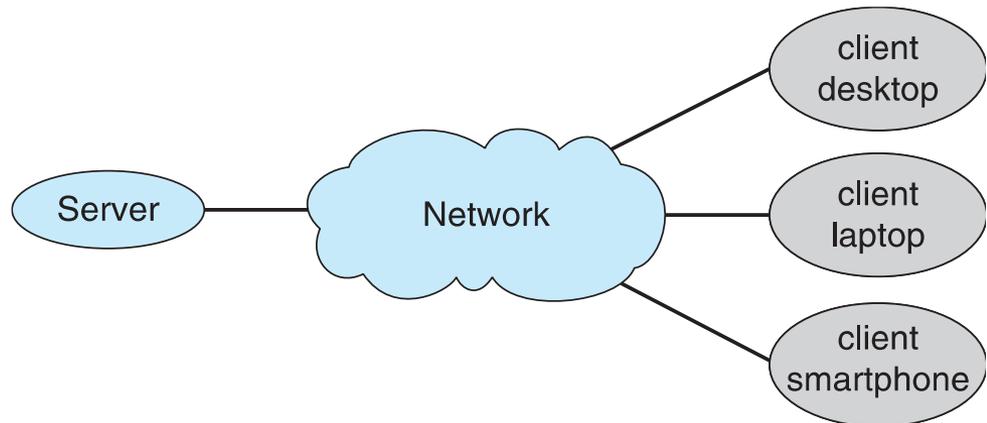
Distributed OS



- Distributed systems to facilitate use of geographically distributed resources
 - workstations on a LAN
 - servers across the Internet
- Supports communications between programs
 - interprocess communication
 - message passing, shared memory
 - networking stacks
- Sharing of distributed resources (hardware, software)
 - load balancing, authentication and access control, ...
- Speedup isn't the issue
 - access to diversity of resources is goal

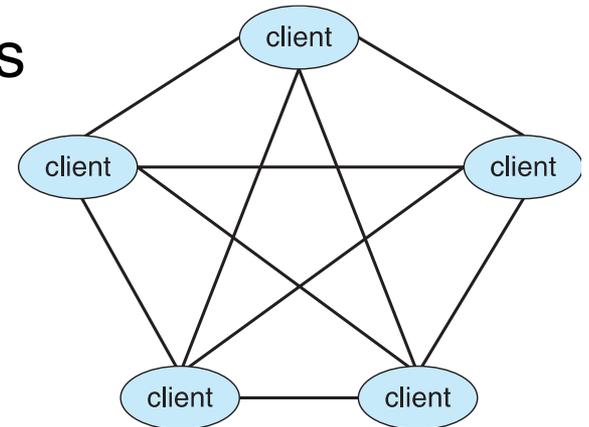
Client/Server Computing

- Dumb terminals supplanted by smart PCs
 - Many systems now servers, responding to requests generated by clients
- Compute-server system
 - provides an interface to client to request services (i.e., database)
- File-server system
 - provides interface for clients to store and retrieve files
- Mail server/service
- Print server/service
- Game server/service
- Music server/service
- Web server/service
- etc.

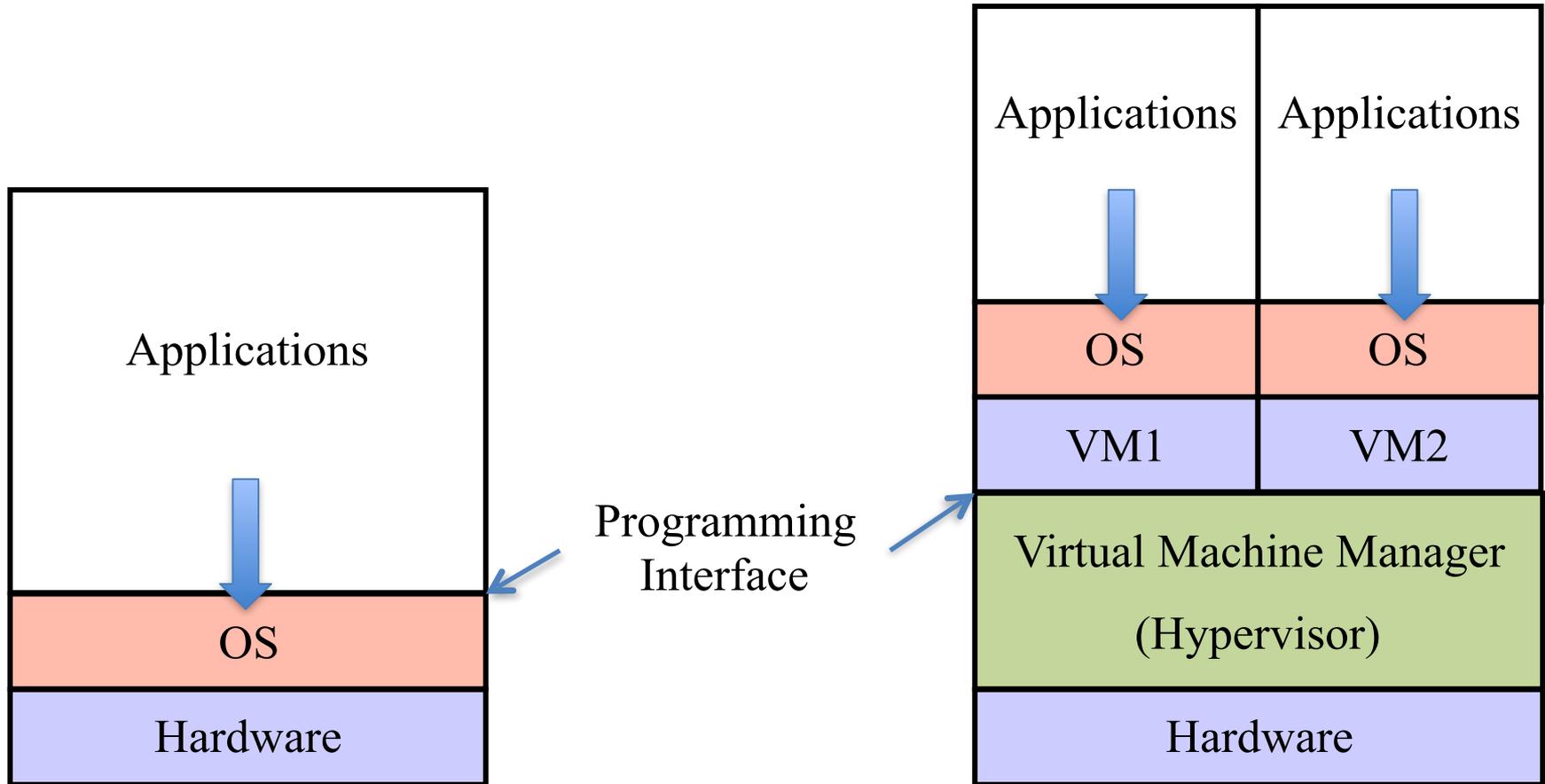


Peer-to-Peer (p2p) Systems

- Another model of distributed system
- Does not distinguish clients and servers
 - All nodes are considered peers
- Each may act as client or server
- Node must join P2P network
 - Registers its service with central lookup service on network, or
 - Broadcast request for service and respond to requests for service via ***discovery protocol***
- Examples include Napster and Gnutella, **Voice over IP (VoIP)** such as Skype

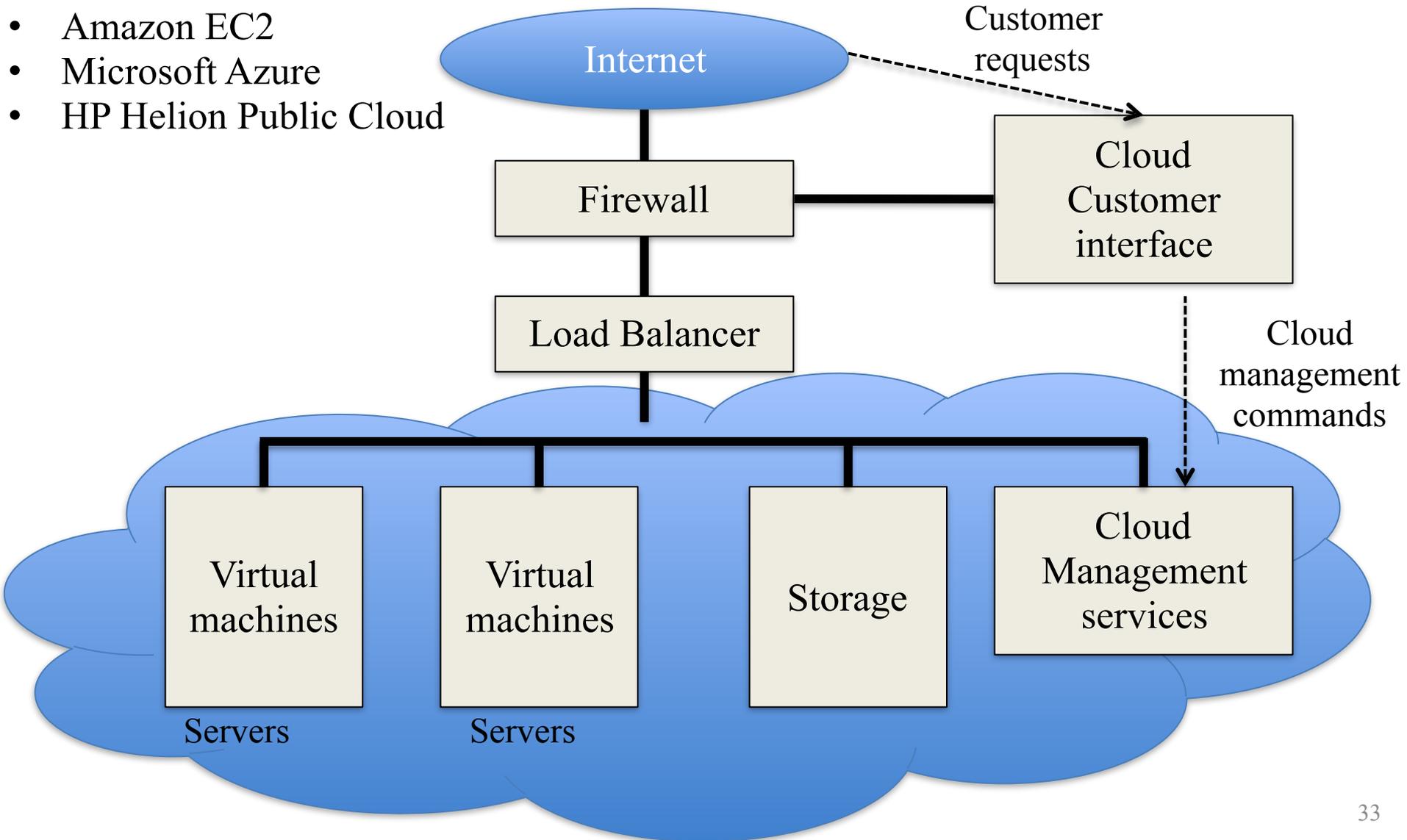


Virtualization



Cloud Computing

- Amazon EC2
- Microsoft Azure
- HP Helion Public Cloud



The major OS issues

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
- **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **communication**: how do programs exchange information, including across a network?

More OS issues...

- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?

There are tradeoffs, solution depends on scenario

Summary

- Introduction
- Definition of an operating system
 - Hard to pin down
- Historical look
- Key functions
 - Timesharing
 - Multitasking
- Various types of OS
 - Depends on platform and scenario
- Next lecture: structure and organisation