Anthony Ventresque
anthony.ventresque@ucd.ie



# Introduction to COMP30640


**School of Computer Science, UCD**

**Scoil na Ríomheolaíochta, UCD**

# Announcements

- Lab session ***later today*** (1-3 p.m.)
  - E1.17 SCE
  - Basic system commands
  - (connection to your remote server)

# Previously…

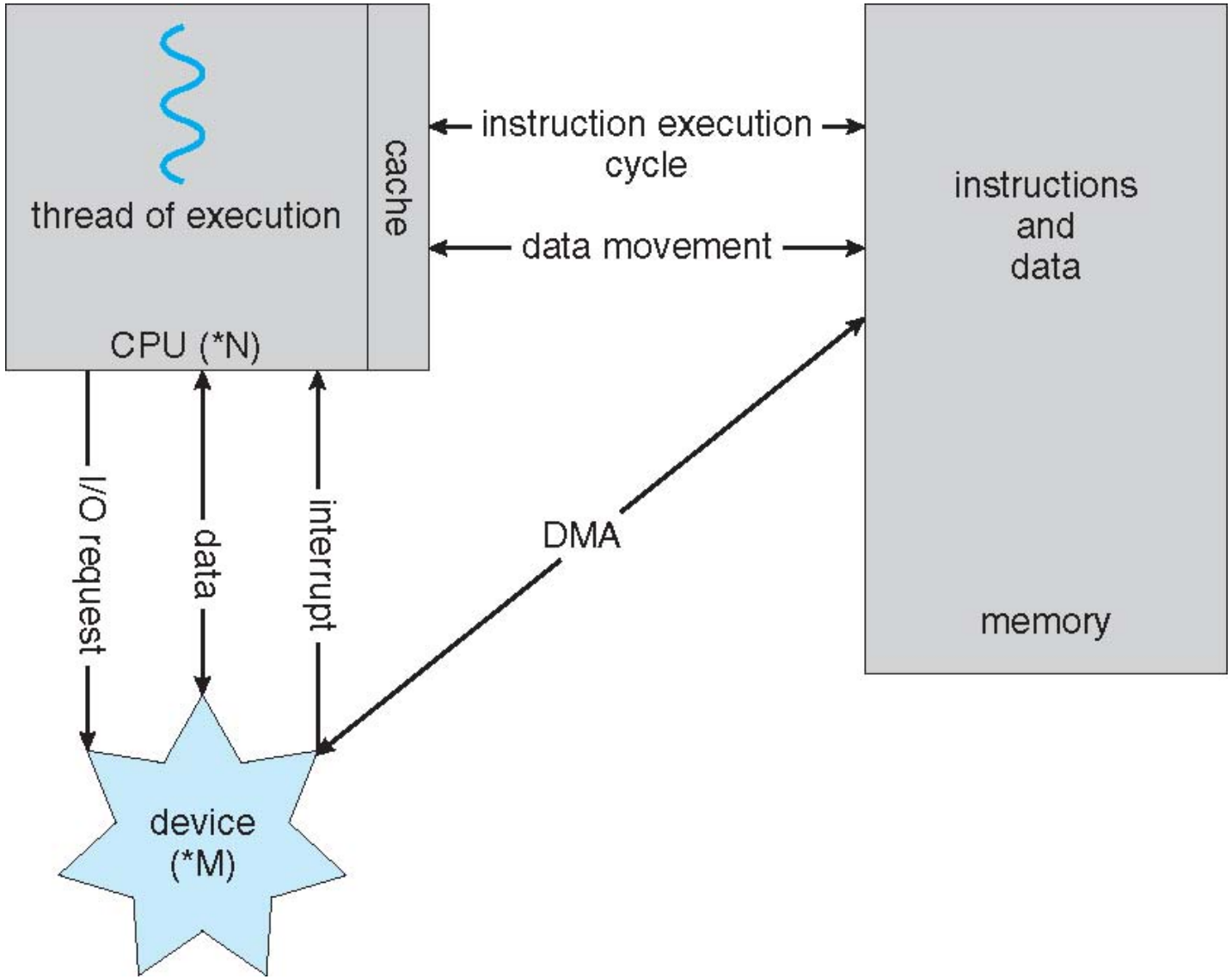
# Outline

- What is an Operating System?
- Operating System Definition
  - Software to manage a computer's resources for its users and applications
- Core Concepts of Modern OS
- OS Challenges
- OS History

Take home message:

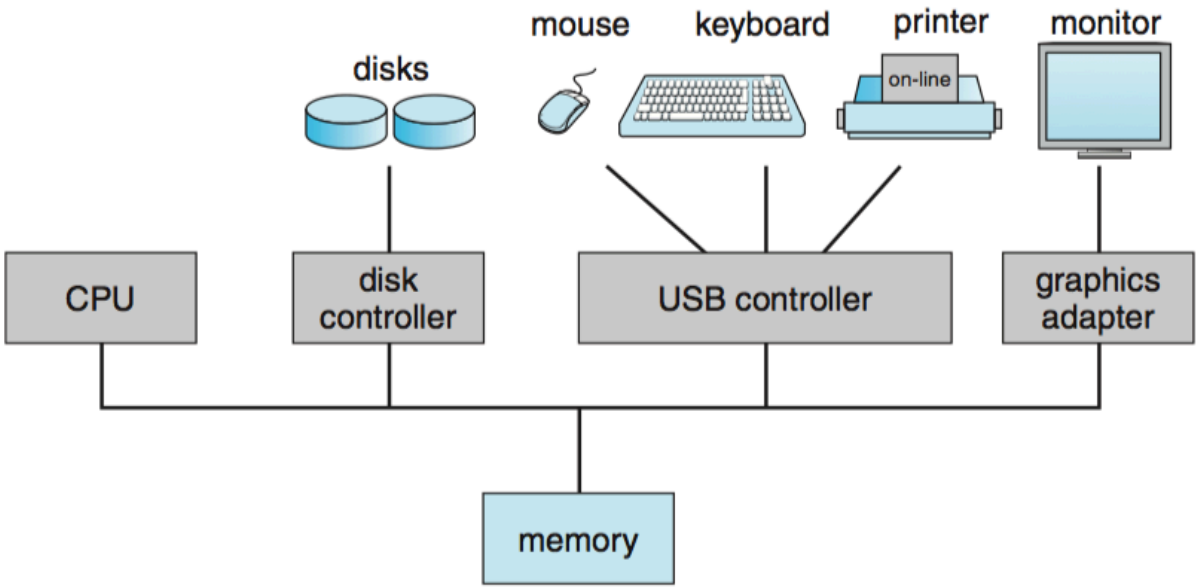
*Operating Systems have to deal with (more and more) complex hardware systems and need to ensure specific properties*

# Modern Computer System



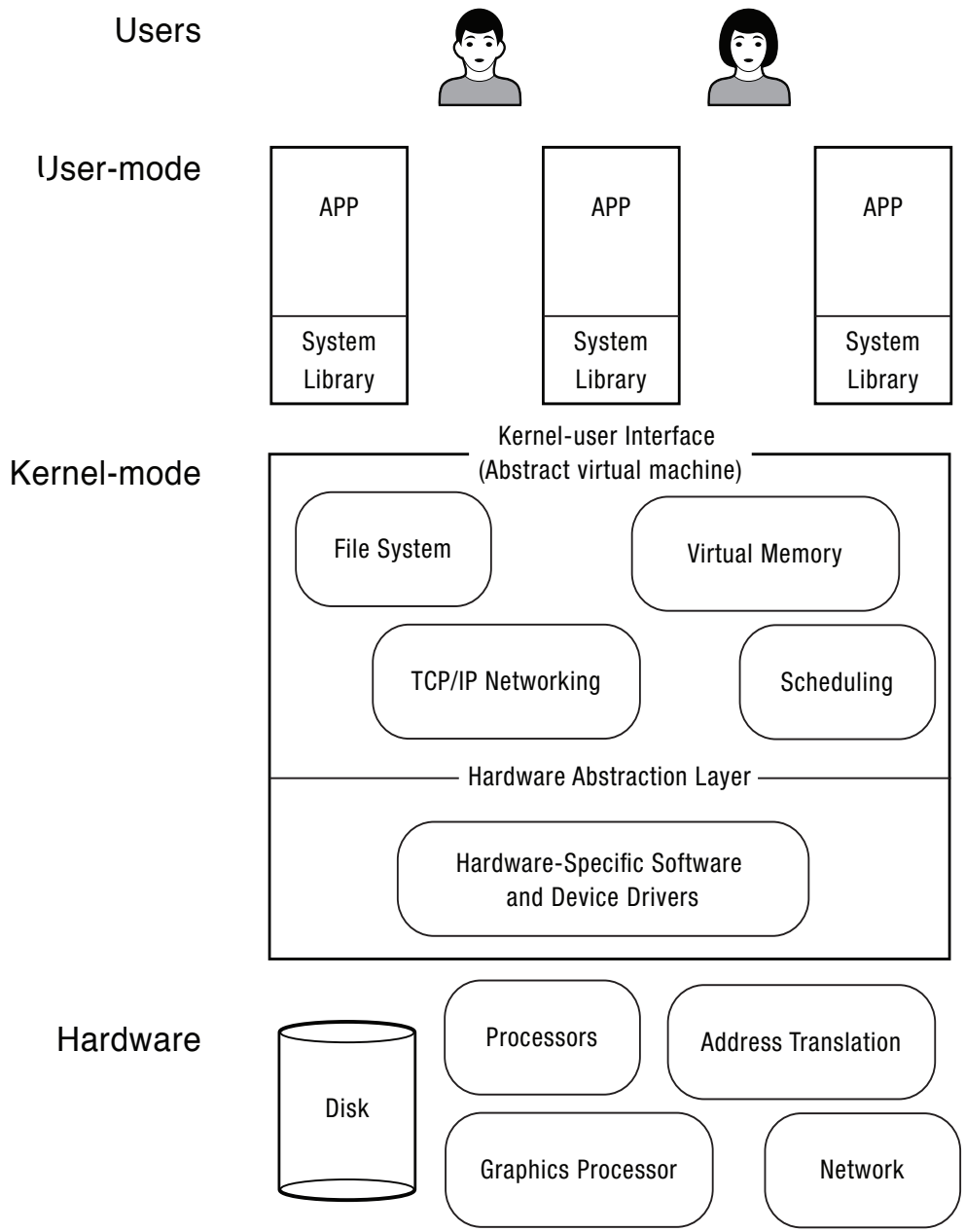
# Computer System Organization

- Computer-system operation
  - One or more CPUs, device controllers connected through common bus providing access to shared memory
  - Concurrent execution of CPUs and devices competing for memory and CPU cycles



# What is an Operating System?

- Software to manage a computer's resources for its users and applications



# What is an Operating System?

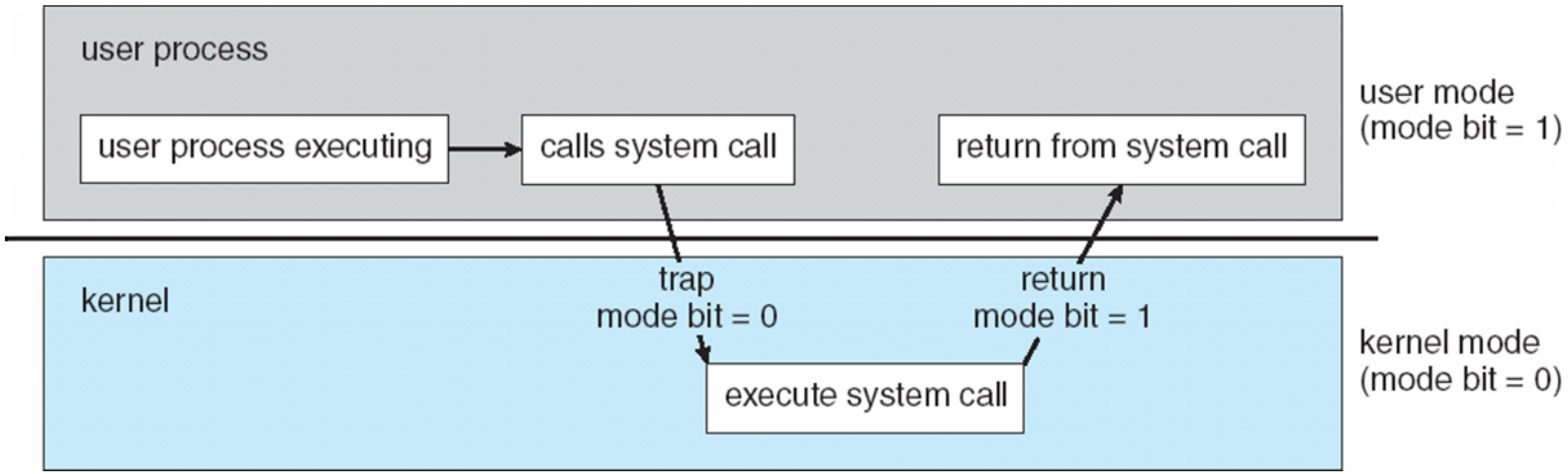
- An OS is a ***program*** that acts as an ***intermediary*** between a user of a computer and the computer hardware
- Goals: Execute user programs, make the computing system easy to use, utilise hardware efficiently
- OS is:
  - ***Resource allocator***: decides between conflicting requests for efficient and fair resource use
  - ***Control programs***: controls execution of programs to prevent errors and improper use of computer

# Operating System Definition

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the ***kernel***.
  - Everything else is either a system program (shipped with the operating system) or an application program

# Dual Mode Operation

- ***Dual-mode operation*** allows OS to protect itself and other system components – User mode and kernel mode
  - Some instructions are only executable in kernel mode, these are privileged
  - Nowadays most CPUs can support multi-mode operations (e.g., VM managers)

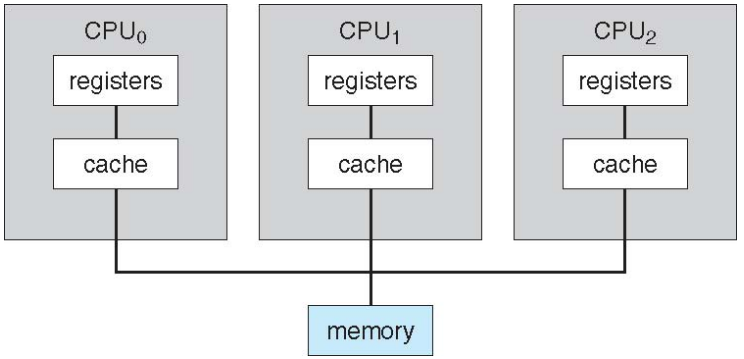


# Bootstrap and Kernel

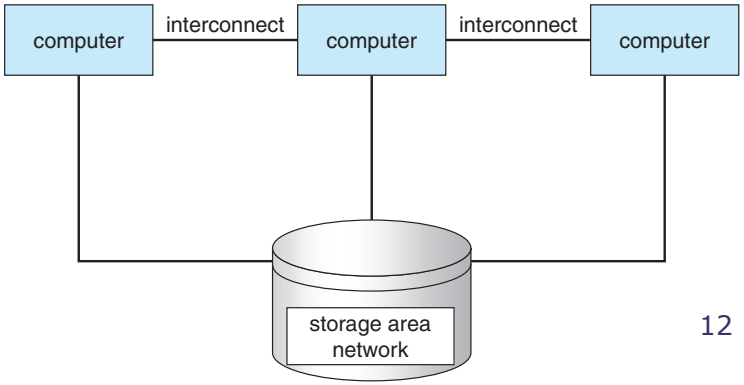
- ***Bootstrap program***: loaded at power-up or reboot
  - Stored in Read-Only Memory (ROM) or Erasable Programmable Read-Only Memory (EPROM), known as firmware
  - Initializes all aspects of system, loads OS kernel and starts execution

# Multiprocessor

- ***Multiprocessor*** Systems: Increased throughput, economy of scale, increased reliability
  - Can be asymmetric (1 processor=1 task) or symmetric (each processor performs all tasks)

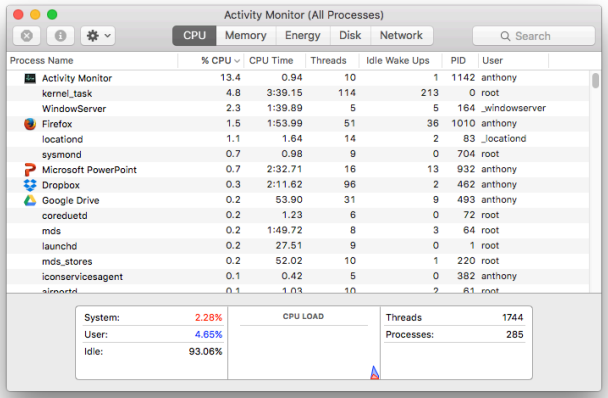



  - Clustered systems – Linked multiprocessor systems



# Multiprogramming and Timesharing

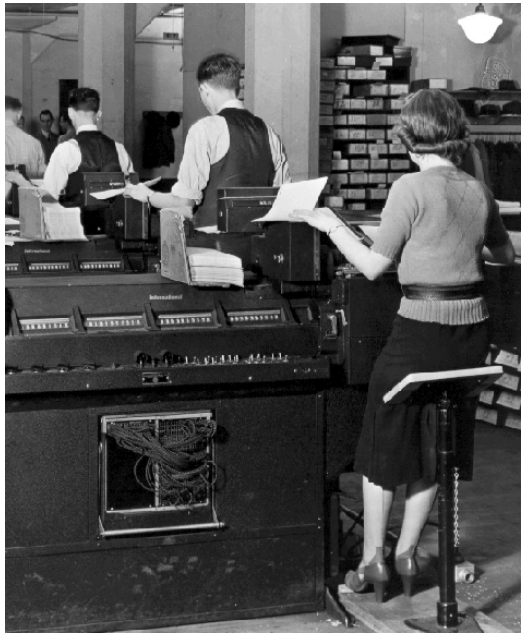
- ***Multiprogramming*** (multitasking) – Provides efficiency via job scheduling
  - When OS has to wait (ex: for I/O), switches to another job
- ***Timesharing*** – CPU switches jobs so frequently that each user can interact with each job while it is running (interactive computing)




# INTERLUDE: USER INTERFACE

# User OS Interfaces

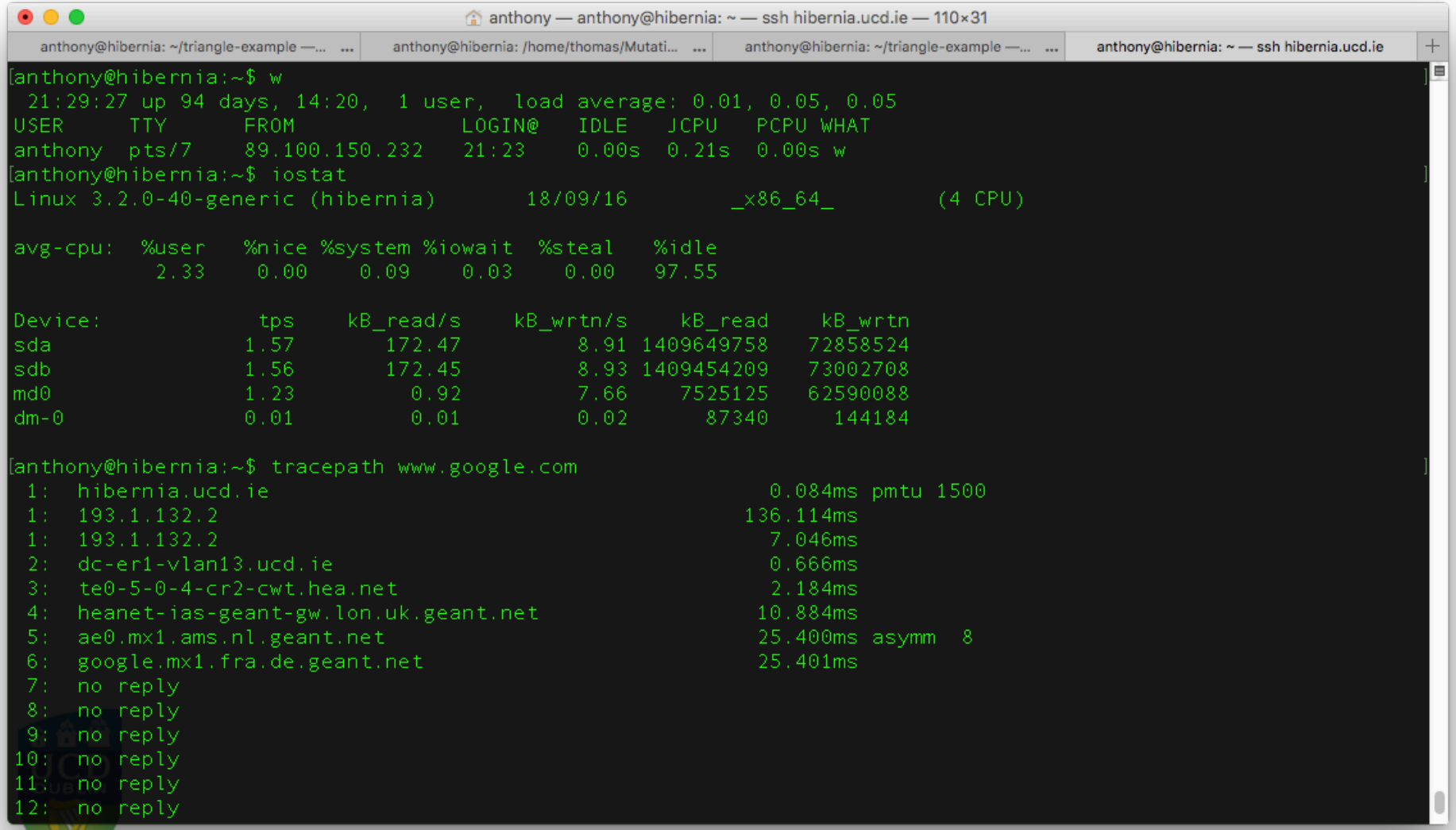
- Almost all operating systems have a user interface (UI).
  - ***Command-Line*** (CLI),
  - ***Graphics User Interface*** (GUI),
  - ***Batch*** (non-interactive, can be organised by job schedulers, workflow managers etc.)



# User OS Interface - CLI

- CLI or ***command interpreter*** allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple flavors implemented – ***shells***
  - Primarily fetches a command from user and executes it
  - Sometimes commands built-in, sometimes just names of programs
    - If the latter, adding new features doesn't require shell modification

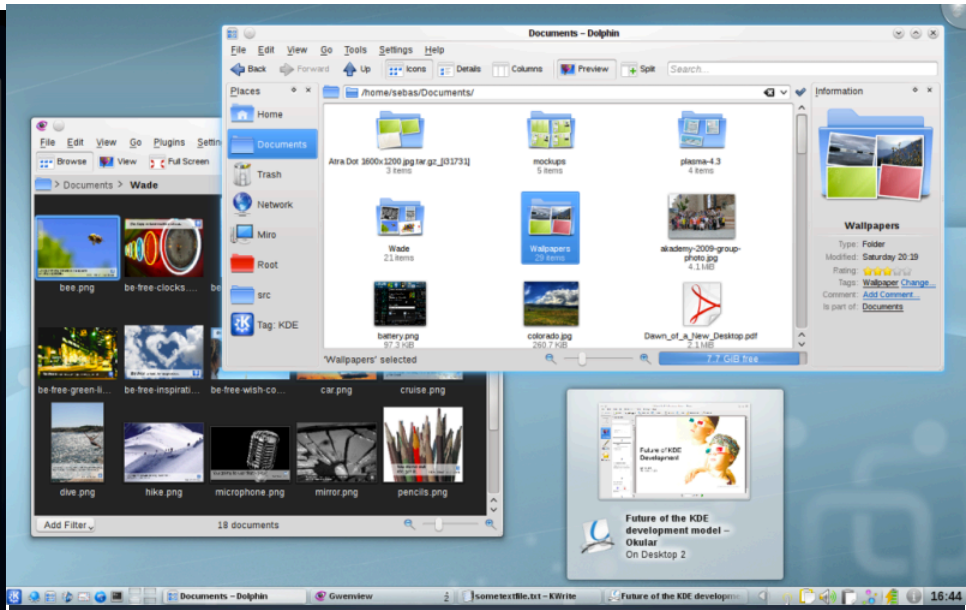
# Bash - Bourne-again shell

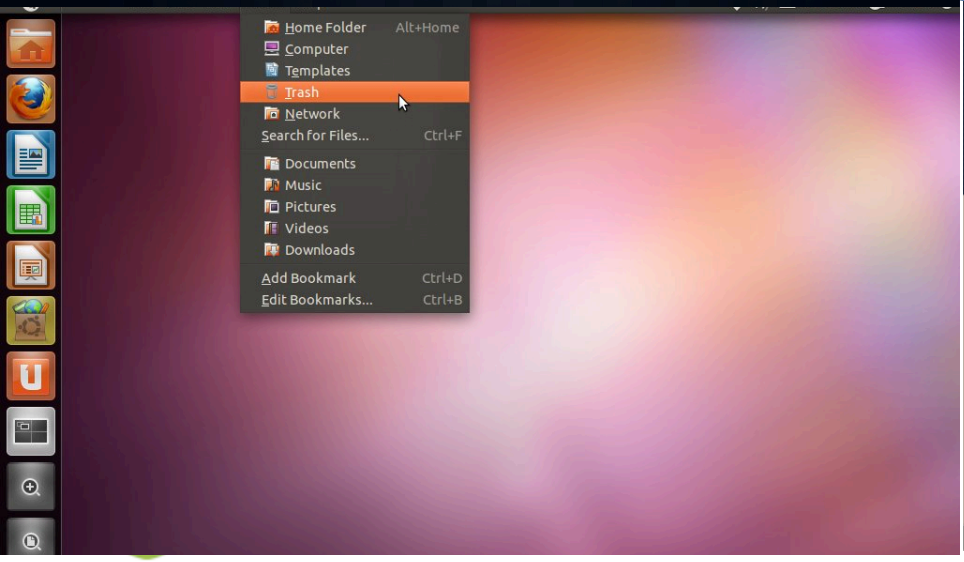


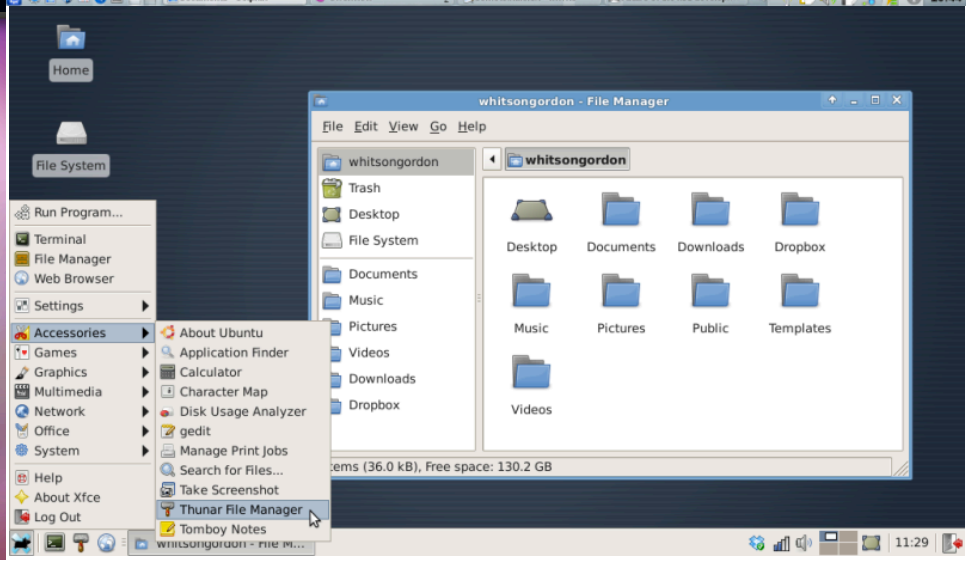
# User OS Interface - GUI

- User-friendly ***desktop*** metaphor interface
  - Usually mouse, keyboard, and monitor
  - ***Icons*** represent files, programs, actions, etc.
  - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a ***folder***)
  - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces

# Gnome vs KDE vs Unity vs XFCE vs etc.

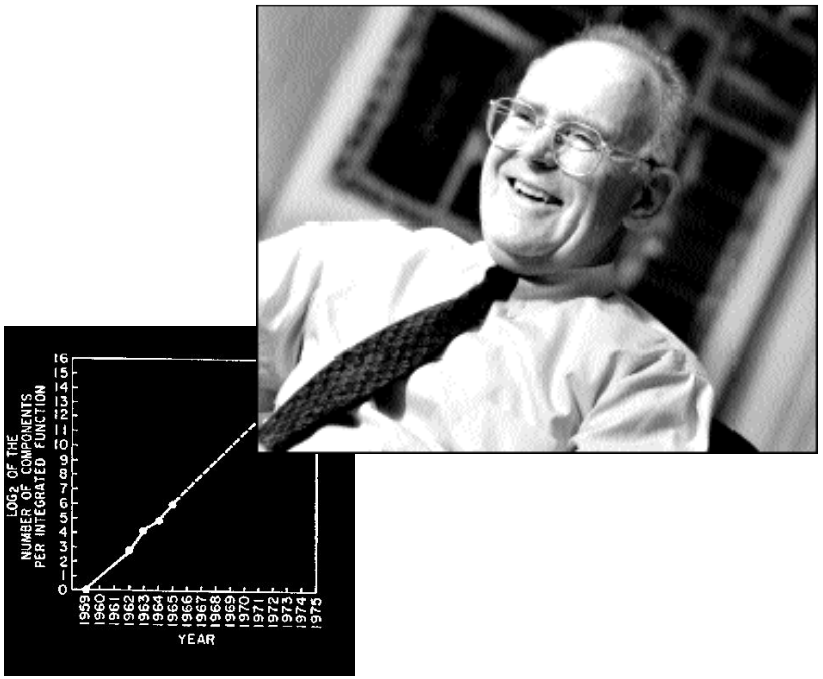



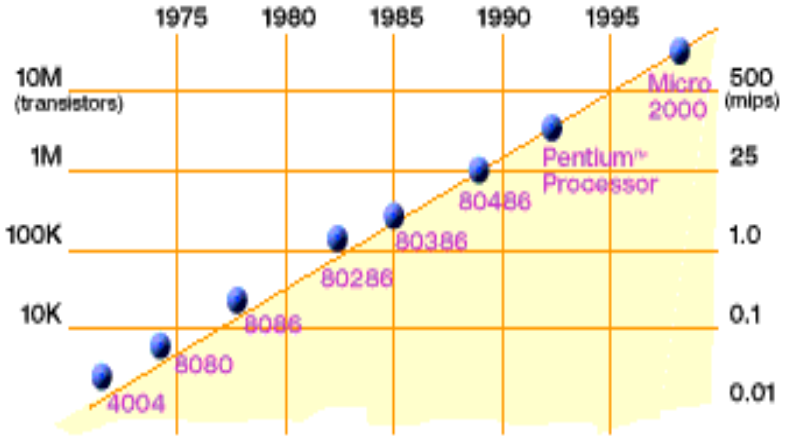
# Touchscreen Interfaces

- Touchscreen devices require new interfaces
  - Mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry
- Voice commands

# OS CHALLENGES

# Technology Trends: Moore's Law



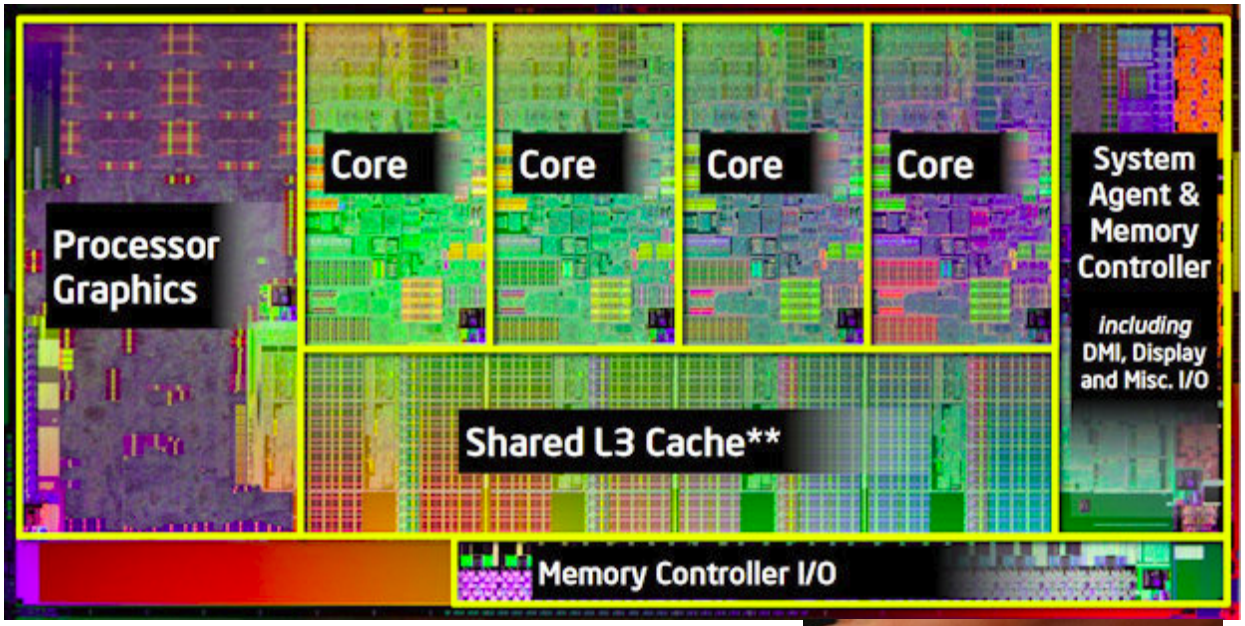

2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months

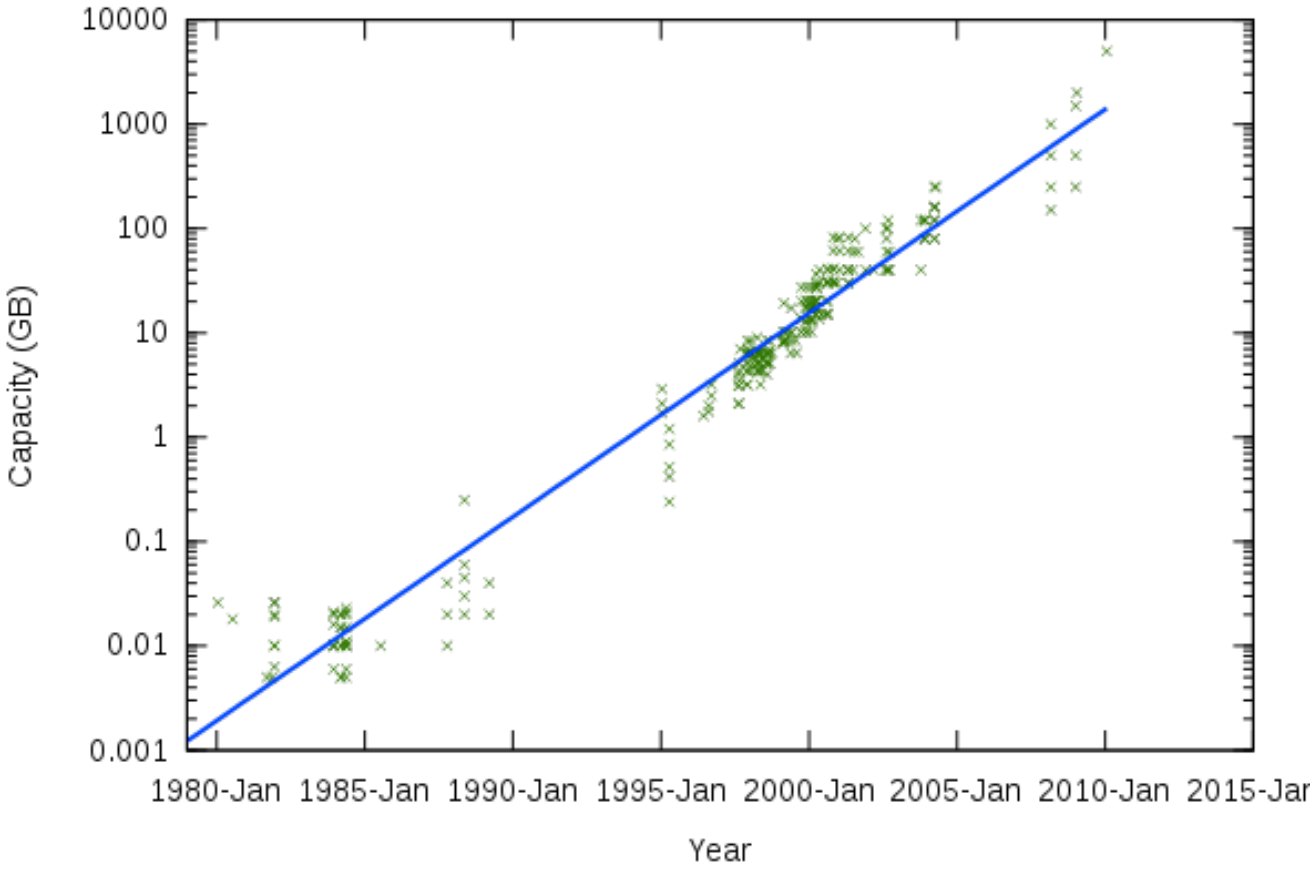
Microprocessors have become smaller, denser, and more powerful

# A Modern Processor: SandyBridge



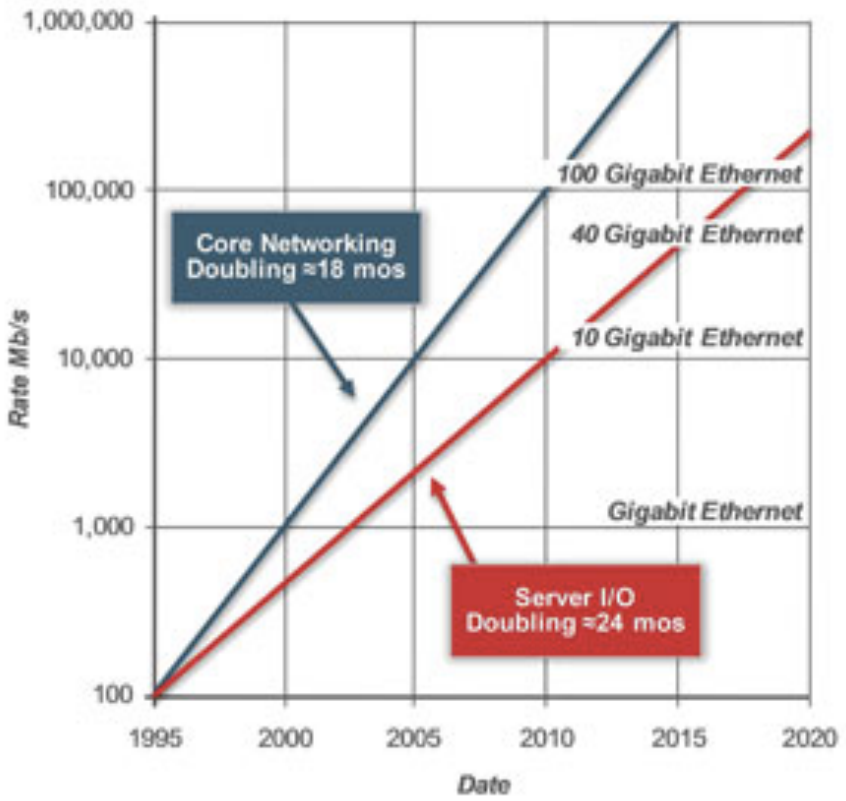

- Package: LGA 1155
  - 1155 pins, 95W design envelope
- Cache:
  - L1: 32K Inst, 32K Data (3 clock access)
  - L2: 256K (8 clock access)
  - Shared L3: 3MB – 20MB
- Transistor count:
  - 504 Million (2 cores, 3MB L3)
  - 2.27 Billion (8 cores, 20MB L3)

# Storage Capacity



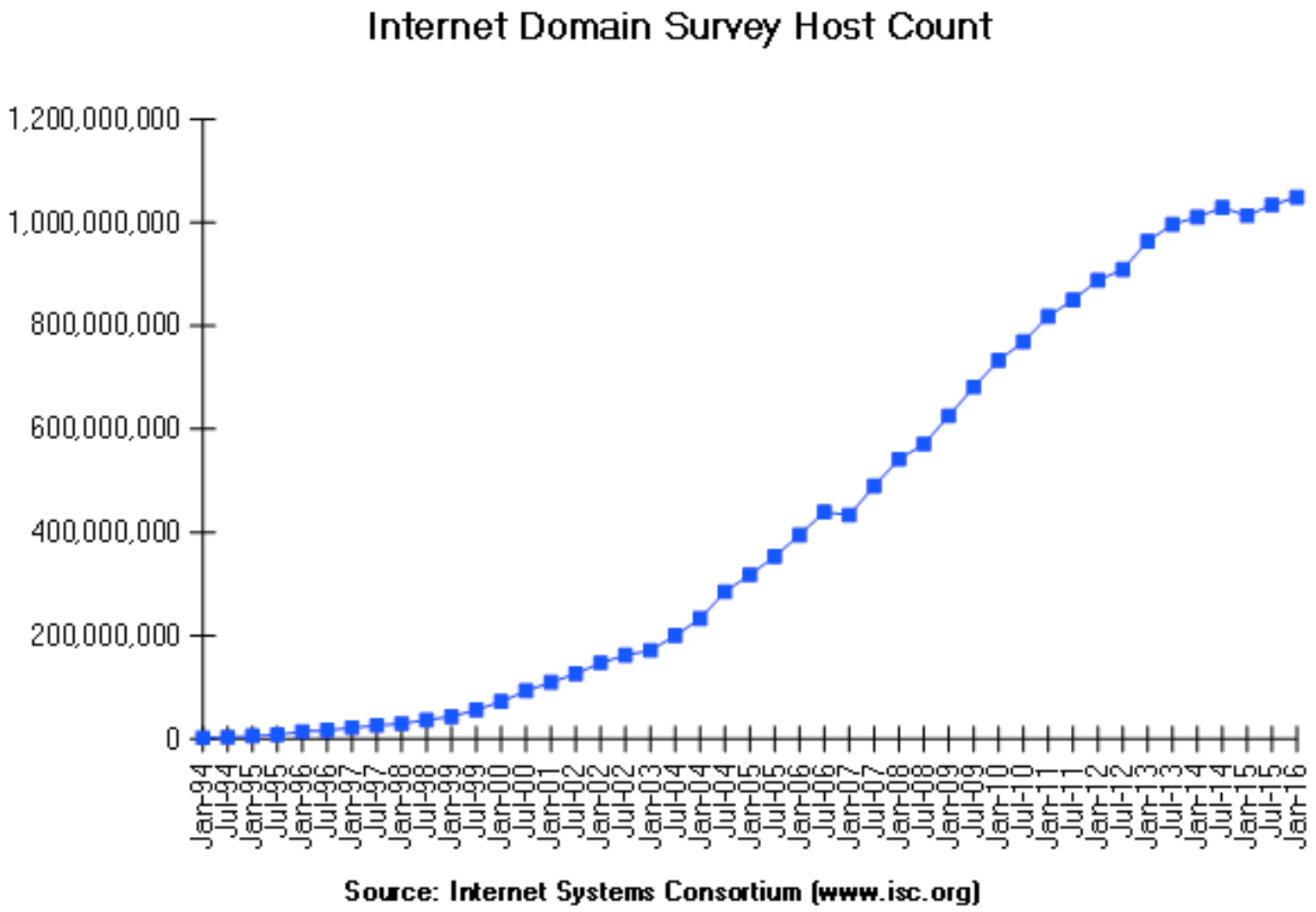

- Retail Hard Disk capacity in GB
  (source: http://www.digitaltonto.com/2011/our-emergent-digital-future)

# Network Capacity

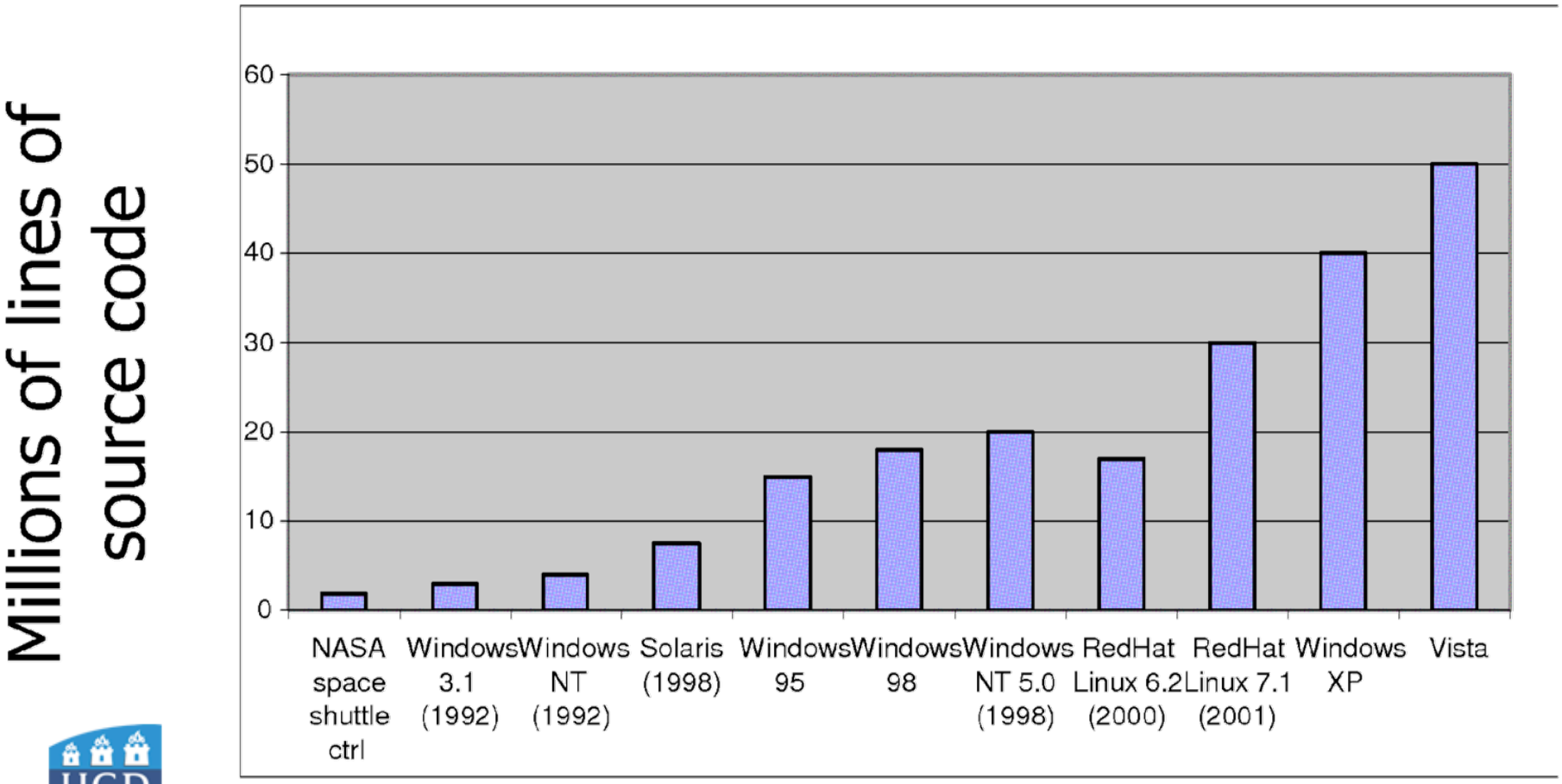



(source: http://www.ospmag.com/issue/article/time-is-not-always-on-our-side)

# Internet Scale



# Increasing Software Complexity



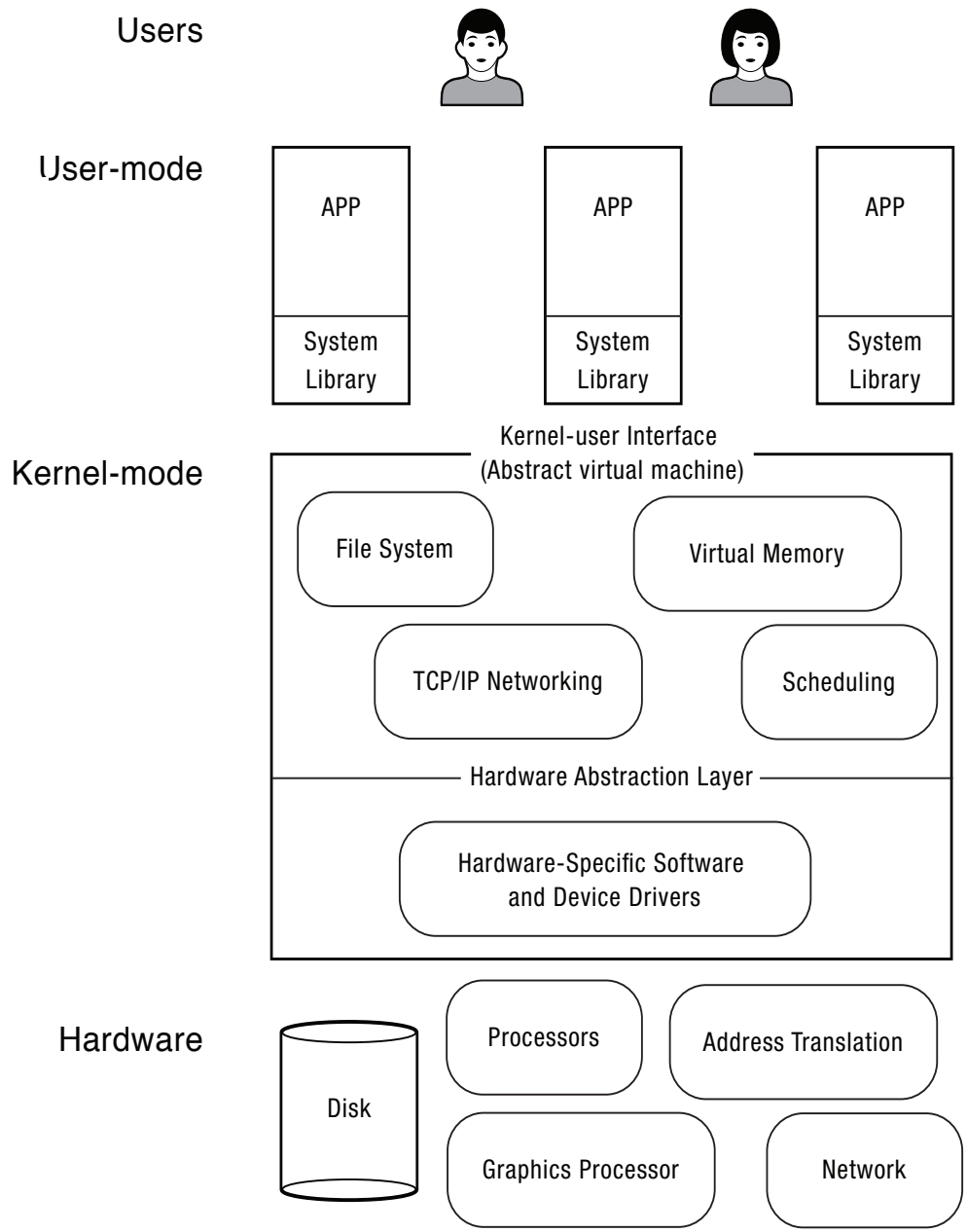

From MIT's 6.033 course

# How Do we Tame Complexity?

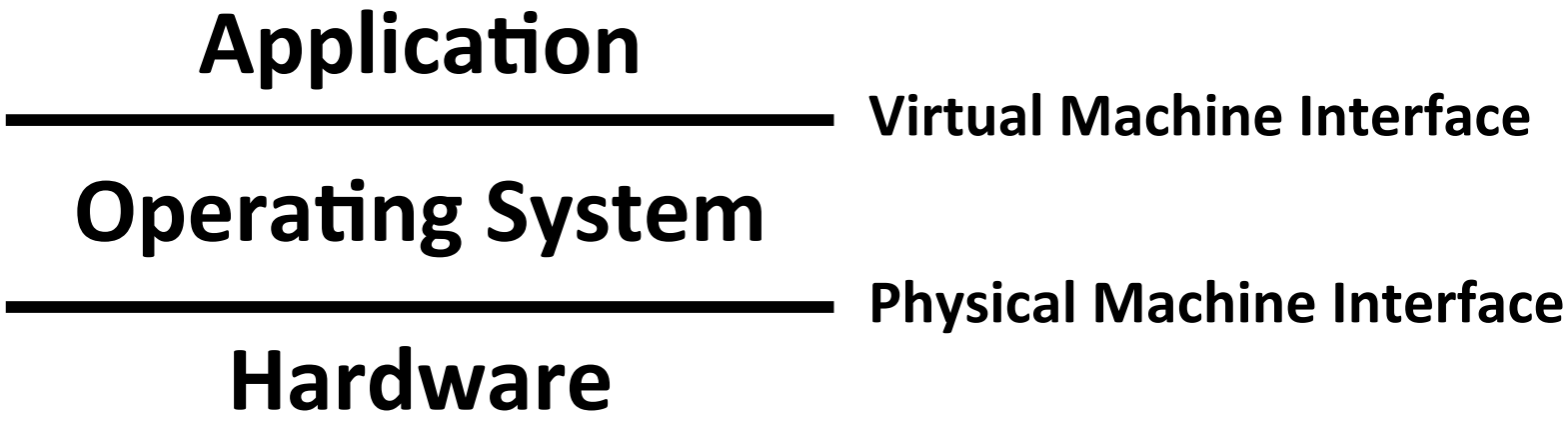
- Every piece of computer hardware different
  - Different CPU
    - Pentium, PowerPC, ColdFire, ARM, MIPS
  - Different amounts of memory, disk, ...
  - Different types of devices
- Mice, Keyboards, Sensors, Cameras, Fingerprint readers
  - Different networking environment
    - Cable, DSL, Wireless, Firewalls,...
- Questions:
  - Does the programmer need to write a single program that performs many independent activities?
  - Does every program have to be altered for every piece of hardware?
  - Does a faulty program crash everything?
  - Does every program have access to all hardware?

# OS Challenges

- Portability
  - For programs
    - Application programming interface (API)
    - Abstract Virtual Machine (AVM)
  - For the Operating System
    - Hardware abstraction layer



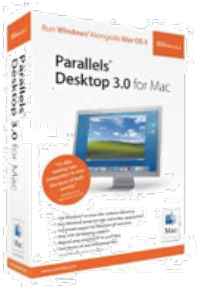
# OS Tool: Virtual Machine Abstraction




- Software Engineering Principle of Abstraction
- For Any OS area (e.g. file systems, virtual memory, networking, scheduling):
  - What's the hardware interface? (physical reality)
  - What's the application interface? (nicer abstraction)

# Virtual Machines

- Software emulation of an abstract machine
  - Give programs illusion they own the machine
  - Make it look like hardware has features you want
- Two types of "Virtual Machines"
  - Process VM: supports the execution of a single program; this functionality typically provided by OS
  - System VM: supports the execution of an entire OS and its applications (e.g., VMWare Fusion, Virtual box, Parallels Desktop, Xen)

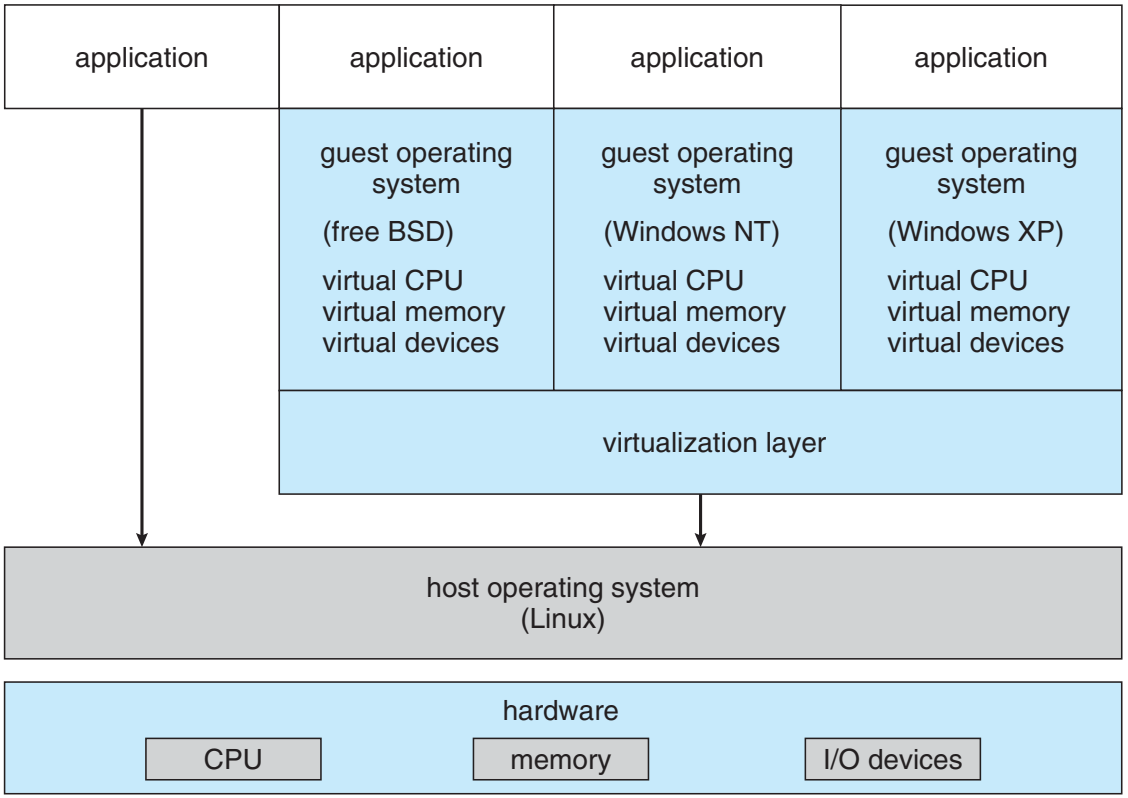


# Process VMs

- Programming simplicity
  - Each process thinks it has all memory/CPU time
  - Each process thinks it owns all devices
  - Different devices appear to have same high level interface
  - Device interfaces more powerful than raw hardware
    - Ethernet card ⇒ reliable, ordered, networking (TCP/IP)
- Fault Isolation
  - Processes unable to directly impact other processes
  - Bugs cannot crash whole machine
- Protection and Portability
  - Java interface safe and stable across many platforms

# System Virtual Machines: Layers of OSs

- Useful for OS development
  - When OS crashes, restricted to one VM
  - Can aid testing programs on other OSs



# OS Challenges

- Reliability
  - Does the system do what it was designed to do?
- Naming
  - How are resources named (by users or programs)?
- Protection & security
  - How do we keep users/programs from interfering with each other, and how do we protect the system from malicious parties?

# OS Challenges

- Data persistence
  - How is data kept beyond the lifetime of programs?
- Performance, reliability & fault tolerance
  - How to handle the system in an efficient manner, and how to cope when something goes wrong?
- Extensibility, scalability & portability
  - How to design a system that can grow, and how do we run it under different hardware configurations?

# OS Challenges

- Performance
  - Latency/response time
    - How long does an operation take to complete?
  - Throughput
    - How many operations can be done per unit of time?
  - Overhead
    - How much extra work is done by the OS?
  - Fairness
    - How equal is the performance received by different users?
  - Predictability
    - How consistent is the performance over time?

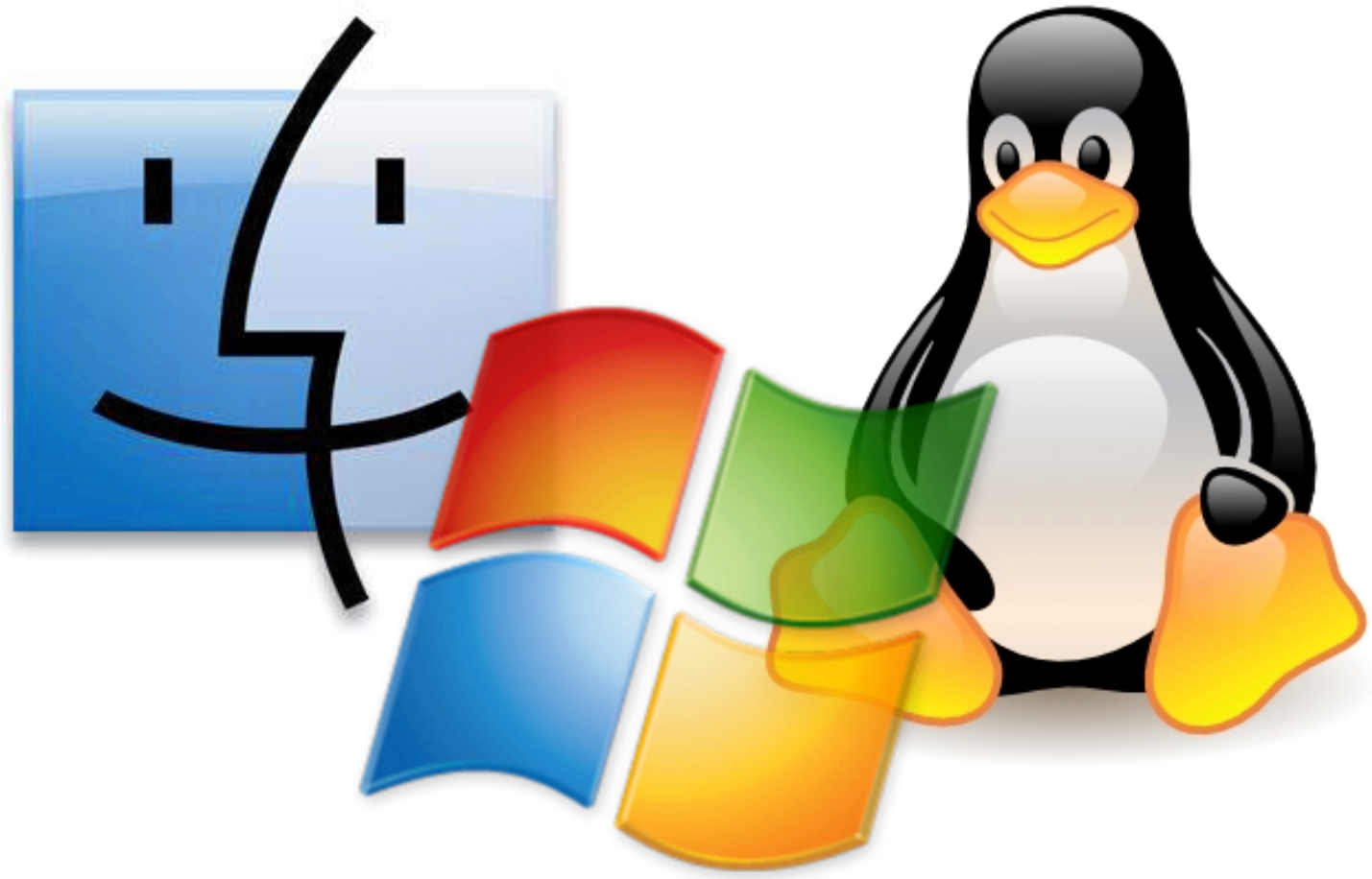
# HISTORY OF OS

# Cost of Computers ↘

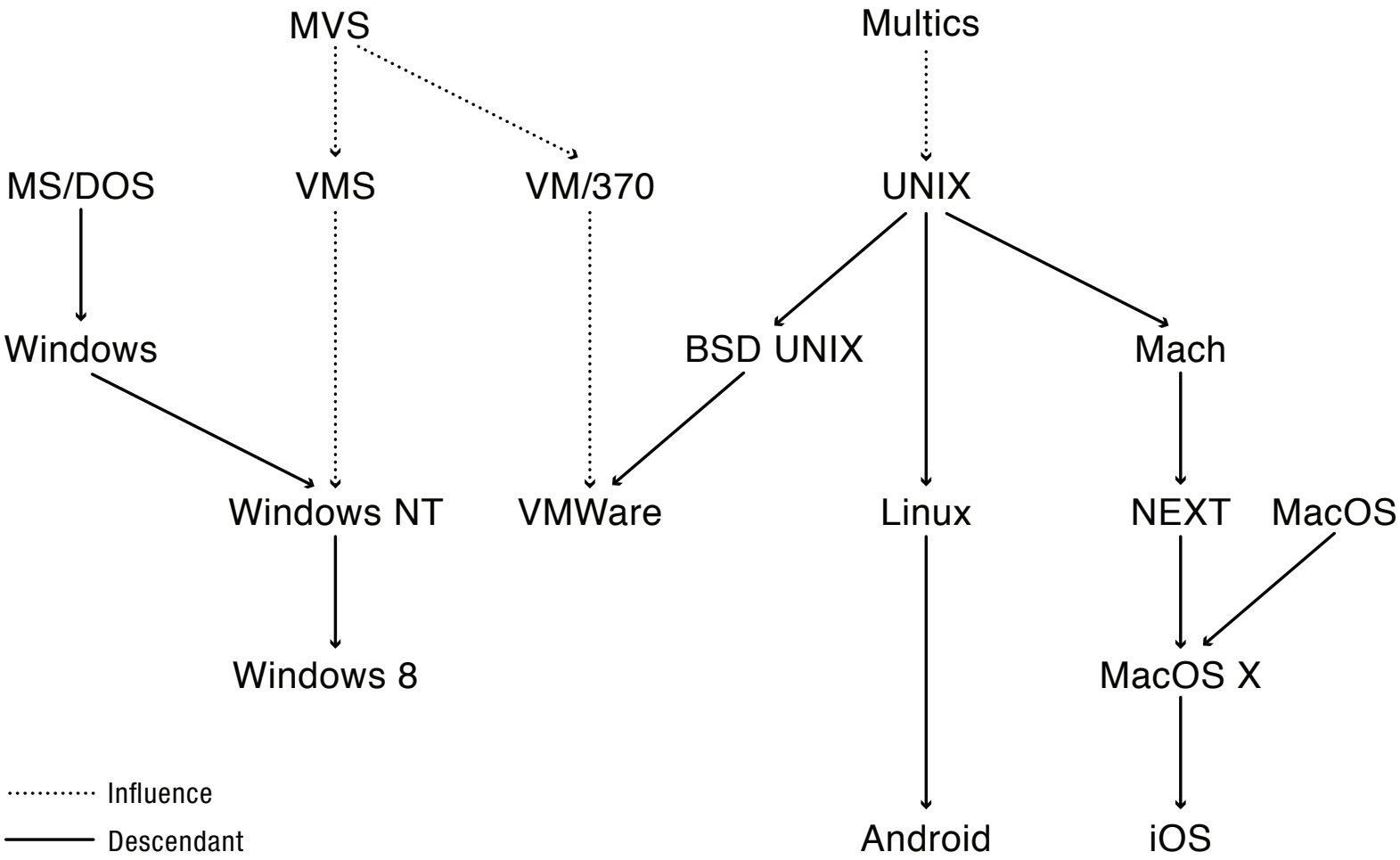
- Several Distinct Phases:
  - Hardware Expensive, Humans Cheap
    - Eniac, … Multics
  - Hardware Cheaper, Humans Expensive
    - PCs, Workstations, Rise of GUIs
  - Hardware Really Cheap, Humans Really Expensive
    - Ubiquitous devices, Widespread networking

"I think there is a world market for maybe five computers." – *Thomas Watson, chairman of IBM, 1943*

# Desktop OS



# Desktop OS History



# Real-time OS

# Real-time OS

- *Guarantee a response to physical events in a fixed interval of time*
  - used for specialised applications: car electronics, subway systems, flight control, factories, power stations, etc. scheduling of activities is critical in order to meet requirements within predetermined timeframes
- most embedded systems run real-time OSs
- Soft real-time is implemented by all OSs in modern PCs
  - example: multimedia applications

# Mobile Device OS

# Mobile Device OS

- Optimisation of portable computing devices
- Initially: personal digital assistants (PDAs) and pocket-PCs
- Currently: mobile phones, tablets

- Limitations
  - Hardware: display, I/O devices, physical space special GUIs
  - Energy constraints: slower processors

# Conclusion

- Operating Systems create a Virtual Machine to hide the complexity of hardware systems
- Operating Systems have to deal with (more and more) complex hardware systems and need to ensure specific properties