COS 318: Operating Systems

Introduction

Today

- Course information and logistics
- What is an operating system?
- Evolution of operating systems
- Why study operating systems?

Information and Staff

- Website
  - http://www.cs.princeton.edu/courses/archive/fall20/cos318/
- Textbooks
  - Modern Operating Systems, 4th Edition, Tanenbaum and Bos
- Instructors
  - JP Singh (jps@cs.princeton.edu)
  - Mohammad Shahrad (mshahrad@cs.princeton.edu)
- Teaching assistants (hours and links posted on web site)
  - Ben Burgess (bburgess@cs.p)
  - Samuel Ginzburg (ginzburg@p)
  - Jiannan Li (jiannanl@p)
  - Lucas Salvador (ls24@p)
- Undergraduate Coordinator and Assistants (to be finalized)

Grading

◆ Projects	70%
◆ Exam	20%
◆ Participation	10%

◆ One exam, on Monday, Nov 23

Projects

- Build a small but real OS kernel, bootable on real PCs
- A lot of hacking (in C & x86 assembly) but very rewarding
- Projects
  - Bootloader (150-300 lines)
  - Non-preemptive kernel (200-250 lines)
  - Preemptive kernel (100-150 lines)
  - Inter-process communication and device driver (300-350 lines)
  - Virtual memory (300-450 lines)
  - File system (500+ lines)

Projects

- How
  - Group of three students for projects 1, 2 and 3
  - A different group of three for projects 4, 5 and 6
  - Design review at the end of week one
  - All projects due Sundays at 11:55 pm
- Where to do the projects
  - Develop on courselab machines, via remote login
  - Instructions on how to develop and submit will be on assignment web pages

Project Grading

- Design Review
  - Requirements will be specified for each project
  - Sign up online for making appointments for design review etc
  - 0-5 points for each design review
  - 10% deduction for missing an appointment
- Project completion
  - Assigned project points plus possible extra points
- Late policy for grading projects
  - 1 hour: 98.6%, 6 hours: 92%, 1 day: 71.7%
  - 3 days: 36.8%, 7 days: 9.7%

Logistics

- Precepts
  - Two precept sessions: attend one
    - Mon and Tuesday: Time TBA
- For project 1
  - Tutorial on assembly programming and kernel debugging
    - Mon 9/7 and Tue 9/8: TBA
  - Precept
    - Mon 9/14 and Tue 9/15: TBA
  - Design reviews
    - Two per project: TBA
  - Due: 9/20 (Sunday) 11:55pm

Use Piazza for Discussions

- Piazza is convenient
  - Most of you love it (?)
- Search, ask and answer questions
  - Students are encouraged to answer questions on Piazza
  - Staff will try to answer in a timely manner
- Only use email if your question is personal/private
  - For questions about your specific project grade: send email to the TA in charge

Ethics and Other Issues

- Honor System
  - Ask teaching staff if you are not sure
  - Asking each other questions is okay: best place is on Piazza
  - **Work must be your own (or your team's)**
- If you discover any solutions online, tell staff right away
- Do not put your code or design on the web, in social media, or anywhere public or available to others …
- *Most important thing to do in this course:*

  Do not violate the Honor Code

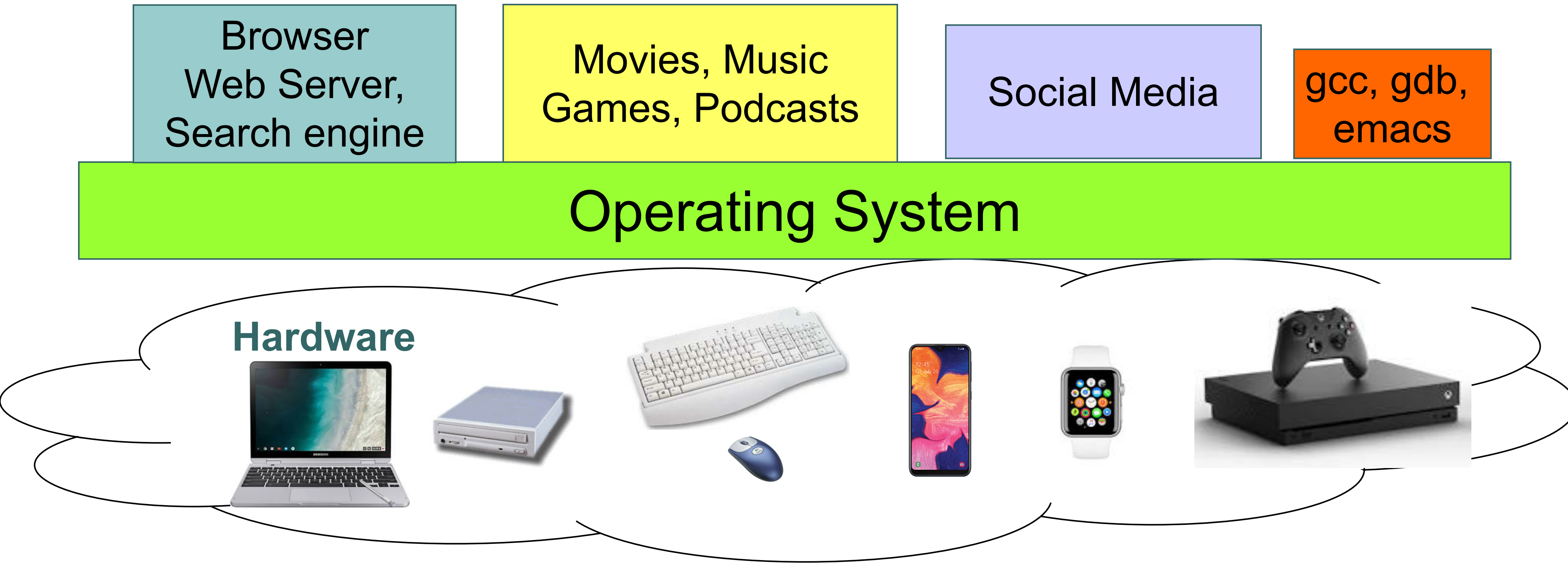
COS318 in Systems Course Sequence

- Prerequisites
  - COS 217: Introduction to Programming Systems
  - COS 226: Algorithms and Data Structures
- 300-400 courses in systems
  - **COS318: Operating Systems**
  - COS320: Compiler Techniques
  - COS333: Advanced Programming Techniques
  - COS432: Information Security
  - COS475: Computer Architecture
- Courses requiring or recommending COS318 as prerequisite
  - COS 418: Distributed Systems
  - COS 461: Computer Networks
  - COS 518: Advanced Operating Systems
  - COS 561: Advanced Computer Networks

Today

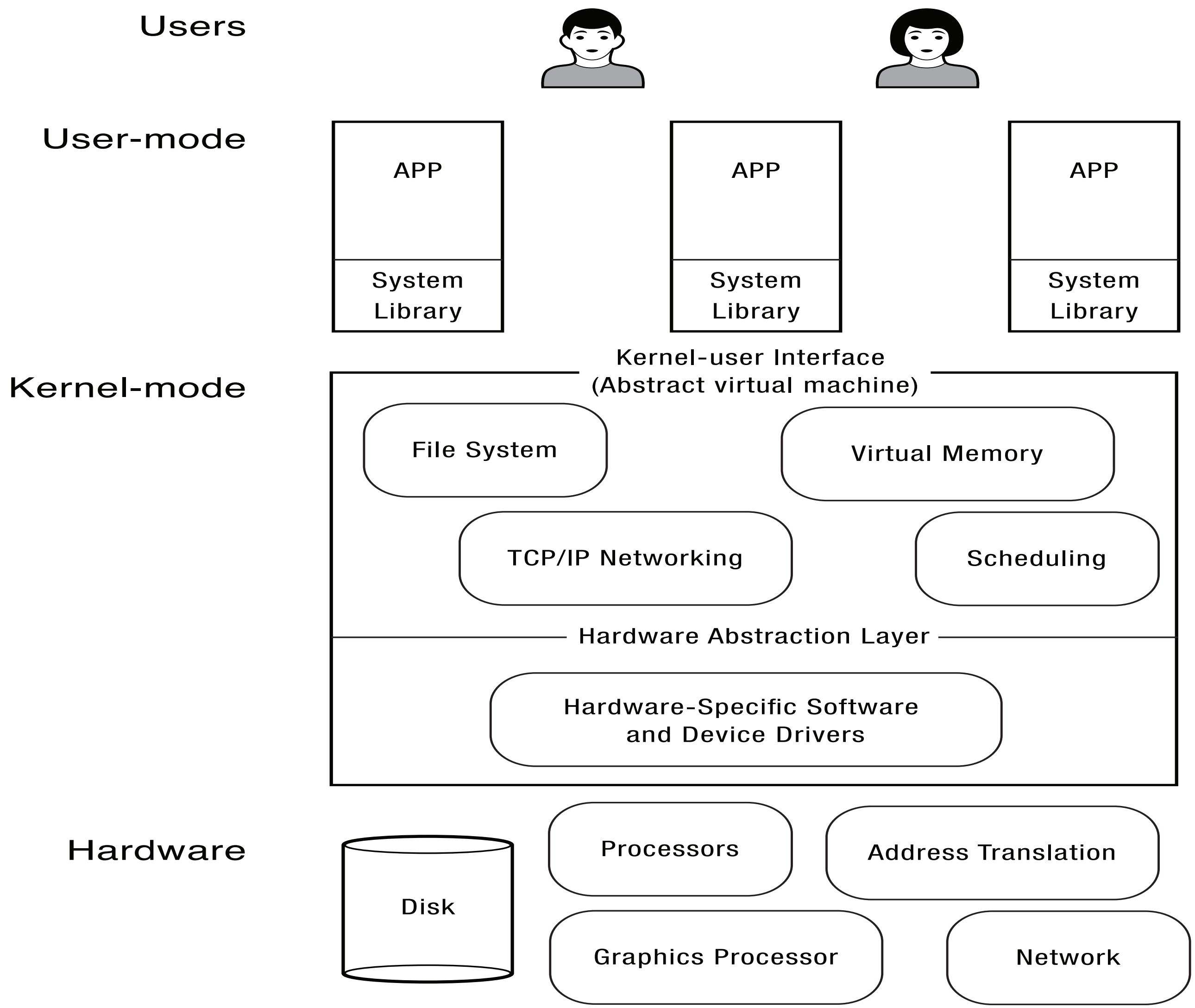
- Course information and logistics
- What is an operating system?
- Evolution of operating systems
- Why study operating systems?

What Is an Operating System?



- Software between applications and hardware
- Provides abstractions to layers above
- Implements abstractions for and manages resources below

In a Little More Depth: The Software



What Does an Operating System Do?

- Provides abstractions to user-level software above
- Implements the abstractions: manages resources

Providing abstractions to the software above

- Allows user programs to deal with **simpler, high-level concepts**
  - files instead of disk blocks
  - virtual memory instead of physical
- **Hides** complex and unreliable hardware
  - and variety of hardware
- **Provides illusions** like “sole application running” or “infinite memory”
- For each area, we can ask:
  - what is the HW interface?
  - What nicer interface does the OS provide?
  - what even nicer interface does the library provide?

Implementing the abstractions

- **Map** from virtual abstractions to physical resources
- **Manage** application interaction with hardware resources
- **Provide standard services**: program execution, I/O operations, file system manipulation, communication, accounting
- **Allow multiple** applications and multiple users to share resources effectively without hurting one another
- **Protect** applications from one another and from crashing the system

Some Examples

- What if a user tries to access disk blocks directly?
- What if a user program can access all RAM memory?
- What if programs run infinite loops?

```
while (1);
```

- What if a user runs the following code:

```
int main() {
    while(1) fork();
}
```

Operating System Roles

- Illusionist
  - Every application appears to have the entire machine to itself
    - Processor/processors
    - All of memory (and in fact vastly more than all of physical memory)
    - Reliable storage
    - Reliable network transport
- Referee
  - Resource allocation among users, applications
  - Protection/isolation of users, applications from one another
- Glue
  - Communication between users, applications
  - Libraries, user interface widgets, …

Example: Storage

- Different types of disks, with very different structures
  - Floppy, various kinds of hard drives, Flash, IDE, …
- Different hardware mechanisms to read, different layouts of data on disk, different mechanics

- Floppy disk has ~20 commands to interact with it
- Read/write have 13 parameters; controller returns 23 codes
- Motor may be on or off, don't read when motor off, etc.
- And this is only one simple disk type

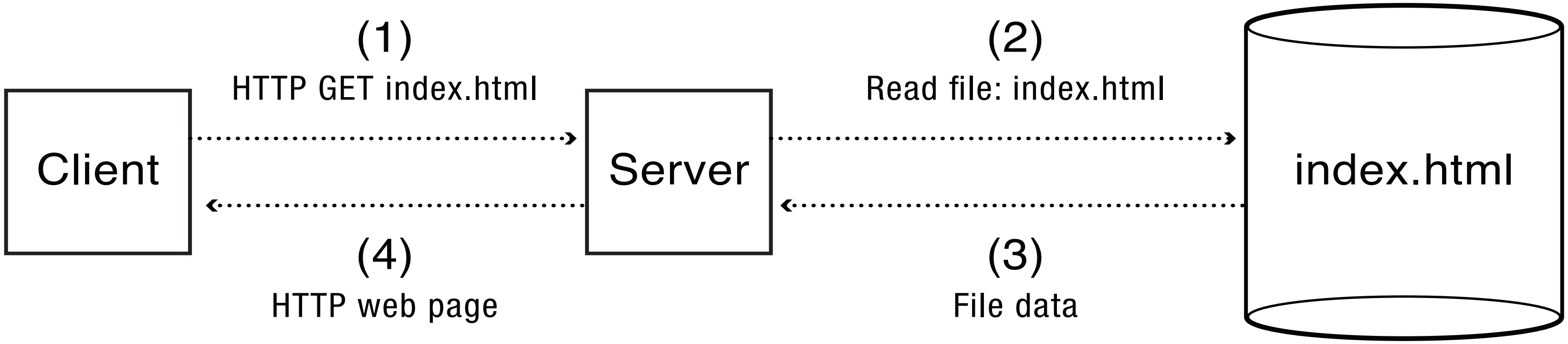
Example: Illusionist Role in Storage

- Allows user programs to deal with **simpler, high-level concepts**
  - Really, data on disk are stored in blocks, which is all that disk knows about
    - No protection of blocks
  - User can think in terms of named files in a file system
- **Hides** complex and unreliable hardware
  - User program needn't know where file data are or structure of the disks
- **Provides illusions**
  - Files are sequential
  - Files can be (nearly) arbitrarily large
  - Files persist even if machine crashes in the middle of a save

Example: Referee Role in Storage

- Enables performance, scalability, fairness and other desirable properties in the face of concurrency
- Prevents users from accessing others' files without permission
- Prevents programs from crashing other programs or the OS

Example: Web Application



- How does the server manage many simultaneous client requests and share CPU and other resources among them?
- How do we keep the client safe from spyware embedded in scripts on a web site?

Today

- Course information and logistics
- What is an operating system?
- Evolution of operating systems
- Why study operating systems?

Evolution of Operating Systems

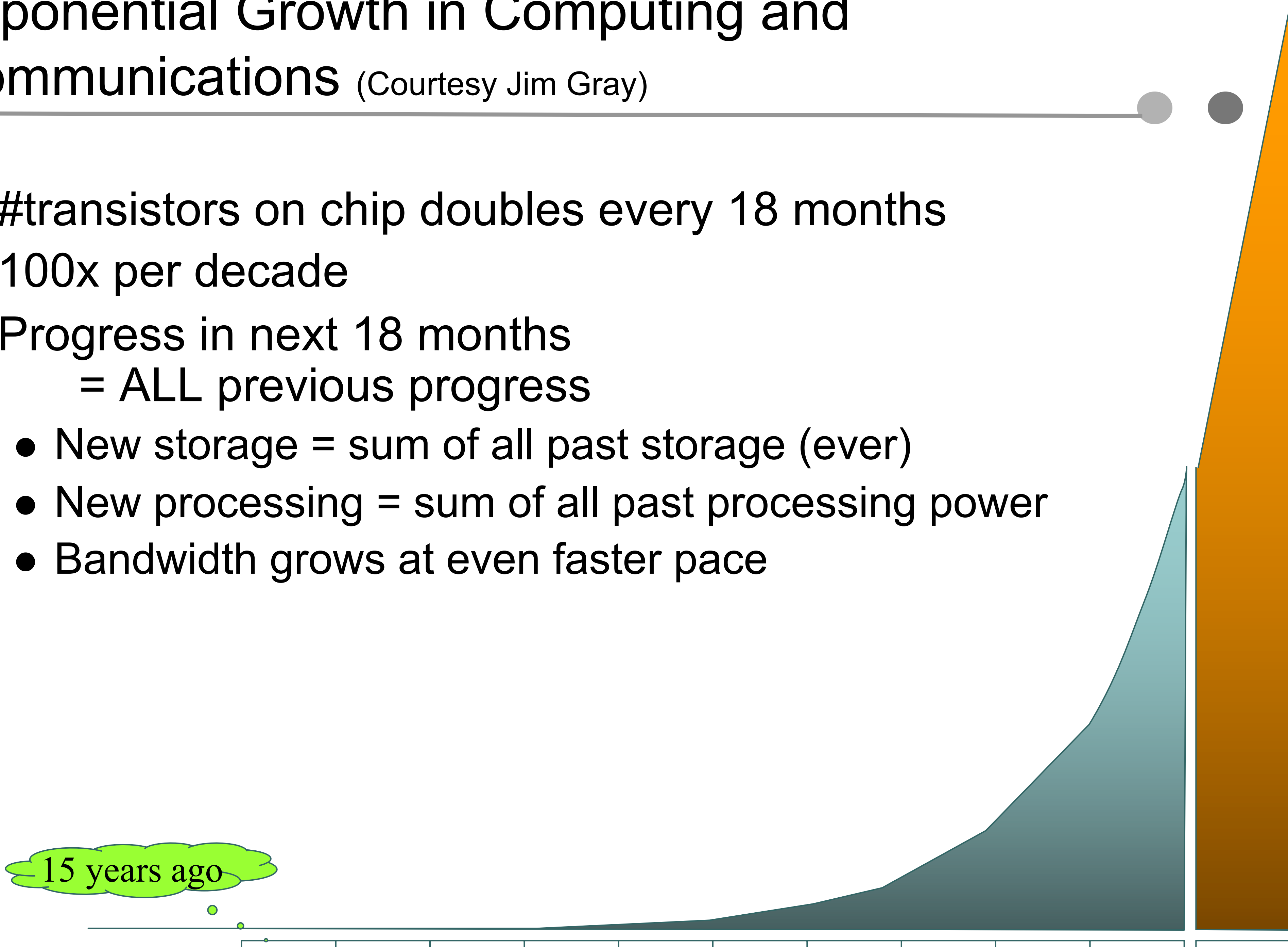
Operating system capabilities and needs depend on:

- Needs of applications and usage contexts (from above)
- Capabilities and constraints of technology (from below)

- For example:
  - compute-heavy versus I/O or networking heavy applications
  - huge or very constrained memory
  - multi-user multi-node data center, multi-user mainframe, smartphone, watch, programmable sensor

Exponential Growth in Computing and Communications (Courtesy Jim Gray)

- #transistors on chip doubles every 18 months
- 100x per decade
- Progress in next 18 months
  = ALL previous progress
  - New storage = sum of all past storage (ever)
  - New processing = sum of all past processing power
  - Bandwidth grows at even faster pace



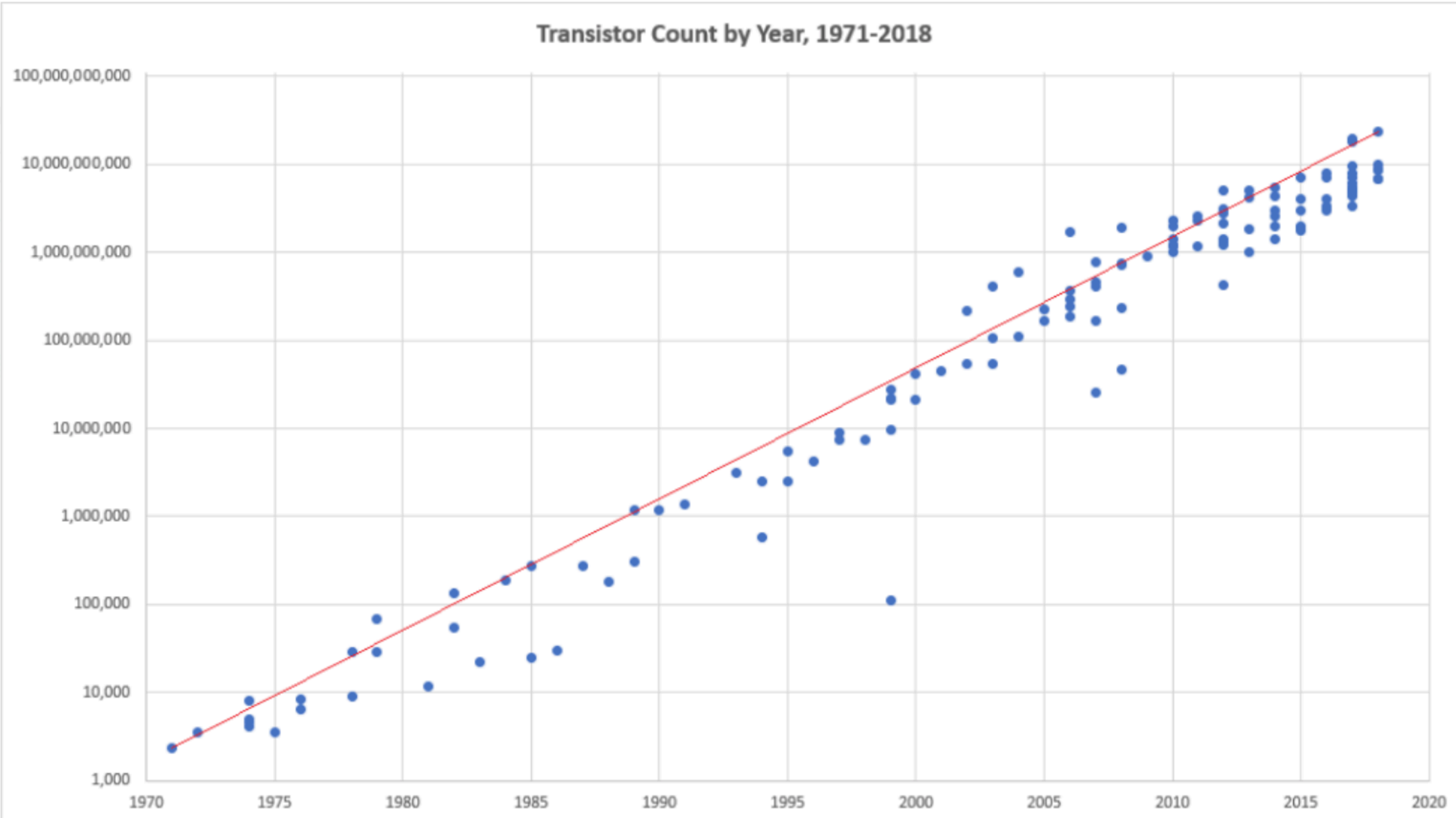
Personal Computers Then and Now

- Osborne Executive PC (1982) vs Apple iPhone
  - 100x weight, 500x volume, 10x cost (adjusted), 1/100 clock frequency

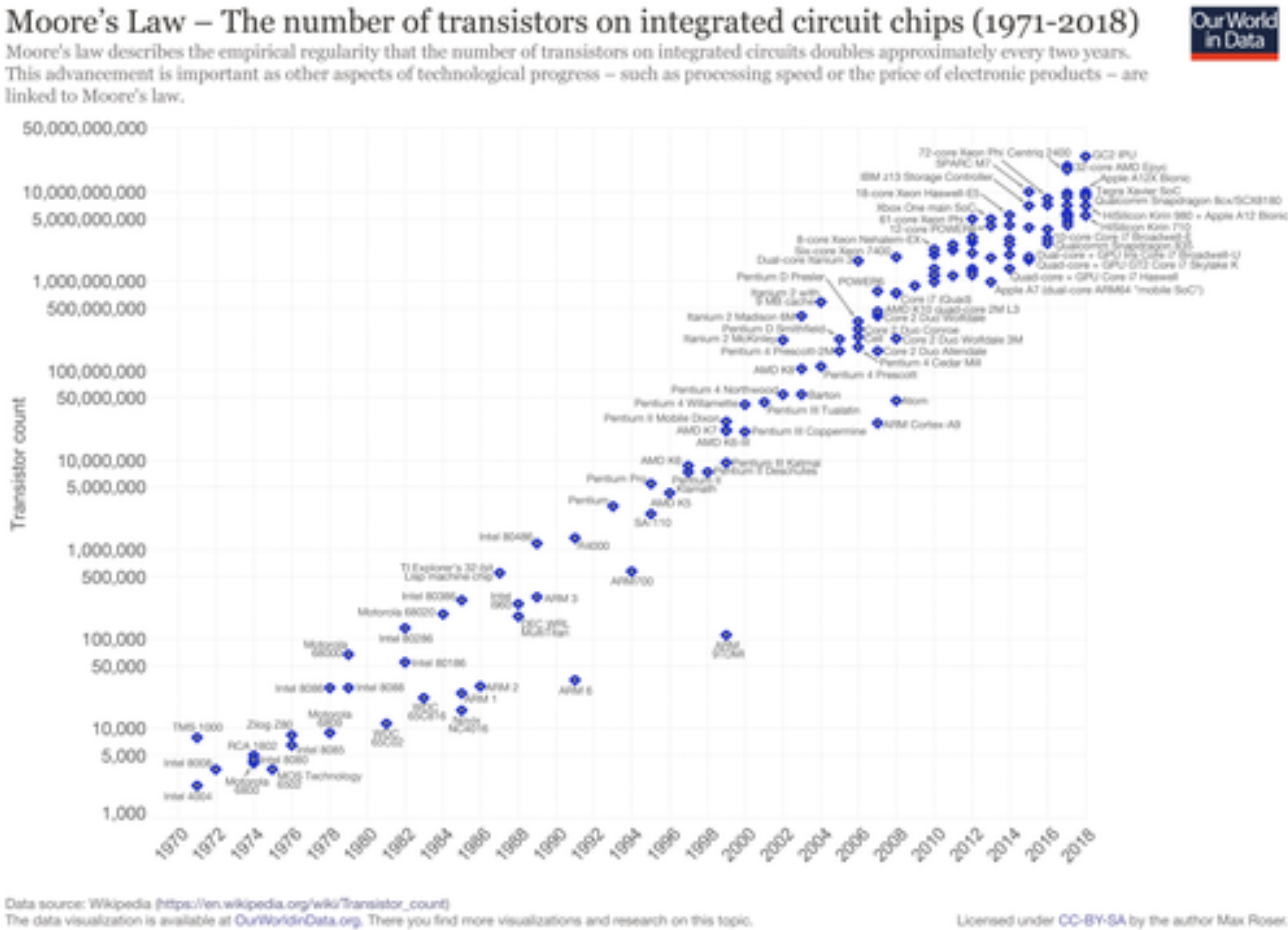
A Typical Academic Computer (1980 vs. 2020)

	1980	2020	Ratio
Intel CPU transistors	0.1M	2B	~20000x
Intel CPU core x clock	10Mhz	8×2.5-5Ghz	~3,000x
DRAM	1MB	64GB	64,000x
Disk	5MB	1TB	200,000x
Network BW	10Mbits/sec	10GBits/sec	1000x
Address bits	32	64	2x
Users/machine	10s	< 1	>10x
$/machine	$30K	$1.5K	1/20x
$/Mhz	$3,000	$0.5	1/6,000x

Transistor Count on Processor Chips over Time



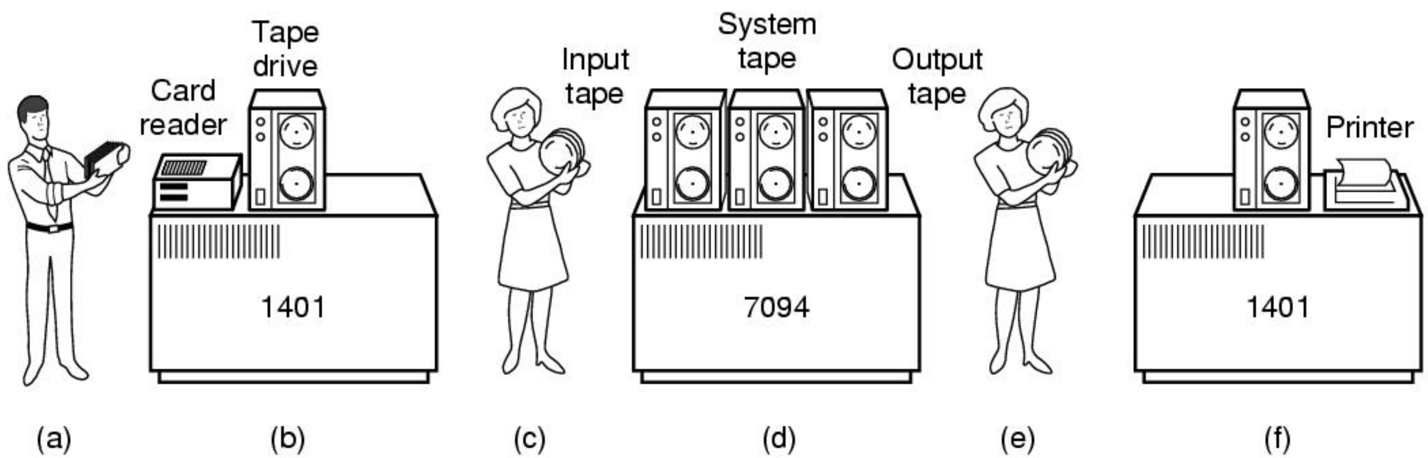
Transistor Count on Processor Chips over Time



Early Digital Computers did not Have OSes

- Charles Babbage's Analytical Engine (1800s)
  - Mechanical device, no operating system
  - Didn't get built, since precise enough parts weren't available
  - But Babbage realize he'd need software for it
    - Hired Ada Lovelace, daughter of Lord Byron
    - First programmer Ada programming language named after her
- No activity till mid 1900s (World War II)
  - Computers made from vacuum tubes or relays
  - Some programmable, some not
  - If so, in machine language or by rewiring circuits
    - Stop complaining about having to program in assembly language ☺
  - No difference between designer, builder, operator, user

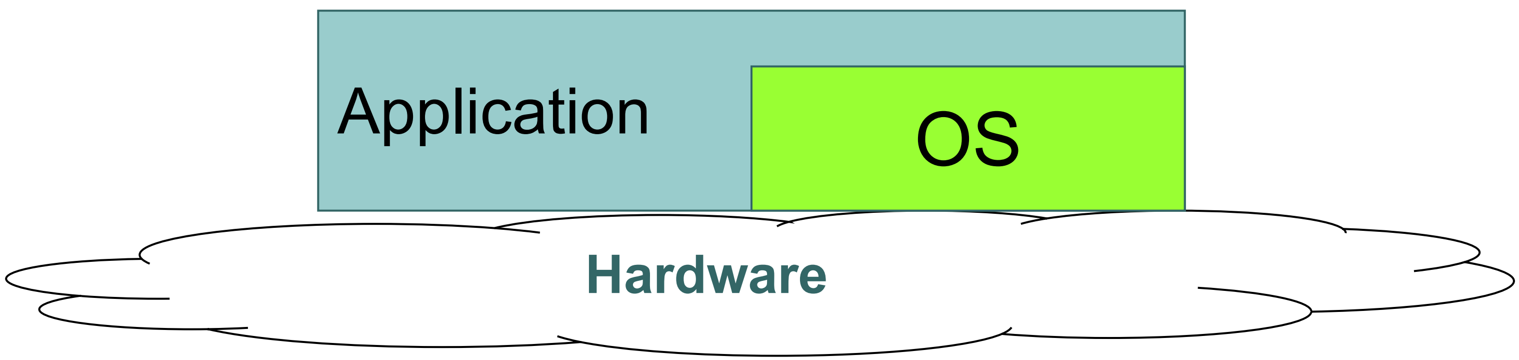
How it Worked in Early Batch Systems



(a) Programmers bring cards to 1401
(b) 1401 reads batch of jobs onto tape
(c) Operator carries input tape with batch of jobs to 7094
(d) 7094 does computing, and puts outputs on output tape
(e) When batch is done operator carries output tape to 1401
(f) 1401 prints output; operator loads next input tape and output tape

OS Phase 0: User at Console

- Machine is expensive relative to human
- Scientific and business applications
  - Q: What programming languages were used for these?
- One program at a time, OS as subroutine library
- User has complete control of machine (no referee role for OS)
- Assumption: No bad people. No bad programs. Minimum interactions
- Problem: A lot of the (expensive) hardware sits idle a lot. Q: Why?

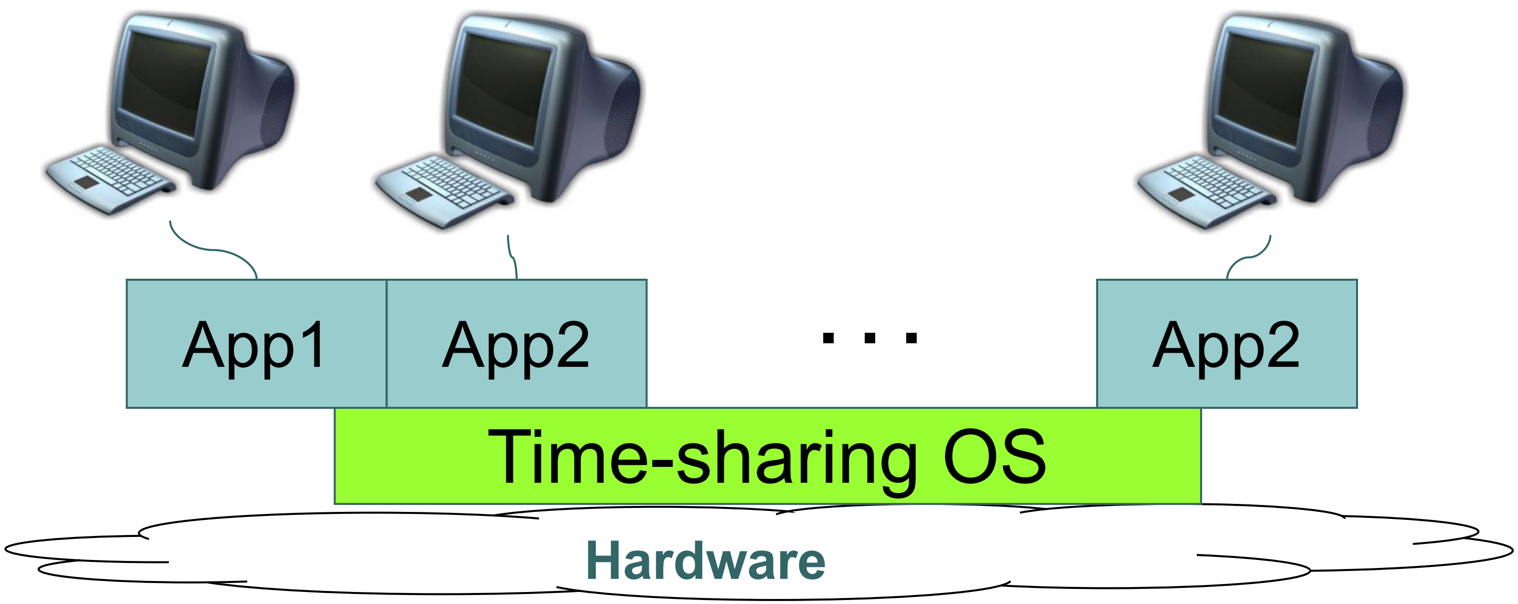


Phase 1: Batch Systems and Multiprogramming

- HW still expensive, human cheap: similar applications
- Goal: Better utilization of expensive hardware
- Batch together programs, run a batch at a time
  - Batch monitor (no protection): load, run, print
- Problems
  - No interactivity. Bad use of (relatively cheap) human time
  - Programs can hurt one another within batch (need protection)
- Developments: Multiprogramming
  - Interrupts; overlap I/O and CPU
  - Direct Memory Access (DMA)
  - Memory protection: keep bugs to individual programs
  - Multics: designed in 1963 and run in 1969; multiprogramming

Phase 2: Time Sharing

- Humans get more expensive too, productivity apps as well
- Use cheap terminals to share a computer (time-sharing OS)
- Better resource usage
- Unix enters mainstream as hardware gets cheaper: minis
- Problems: thrashing as users increase; unpredictable response times



Phase 3: HW Cheaper, Human More Expensive

- More GUI applications, communication on network
  - Pop-menu window interface, email, publishing SW, spreadsheet, FTP, Telnet
- Personal computer
  - Altos OS, Ethernet, Bitmap display,
    laser printer (79)
  - Became >200M units per year
- PC operating system
  - Memory protection
  - Multiprogramming
  - Networking

First PC at Xerox PARC

Now: > 1 Machines per User

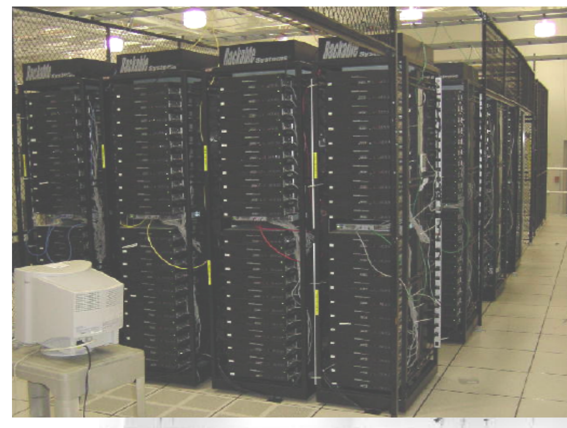
- Pervasive computers
  - Wearable computers
  - Communication devices
  - Entertainment equipment
  - Computerized vehicle
  - Phones: billions units /year
- OS are specialized
  - Embedded OS
  - Specialty general-purpose OS (e.g. iOS, Android)






Now: Multiple Processors per "Machine"

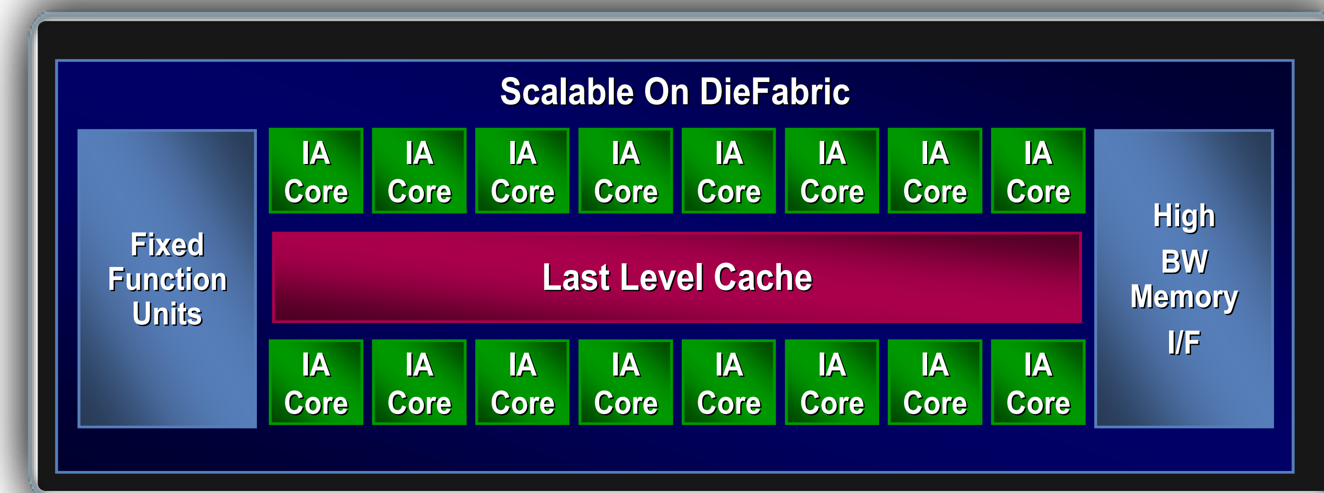
- Multiprocessors
  - SMP: Symmetric MultiProcessor
  - ccNUMA: Cache-Coherent Non-Uniform Memory Access
  - General-purpose, single-image OS with multiproccesor support



- Multicomputers
  - Supercomputer with many CPUs and high-speed communication
  - Specialized OS with special message-passing support
- Clusters
  - A network of PCs
  - Server OS w/ cluster abstraction (e.g. MapReduce)

Now: Multiple "Cores" per Processor

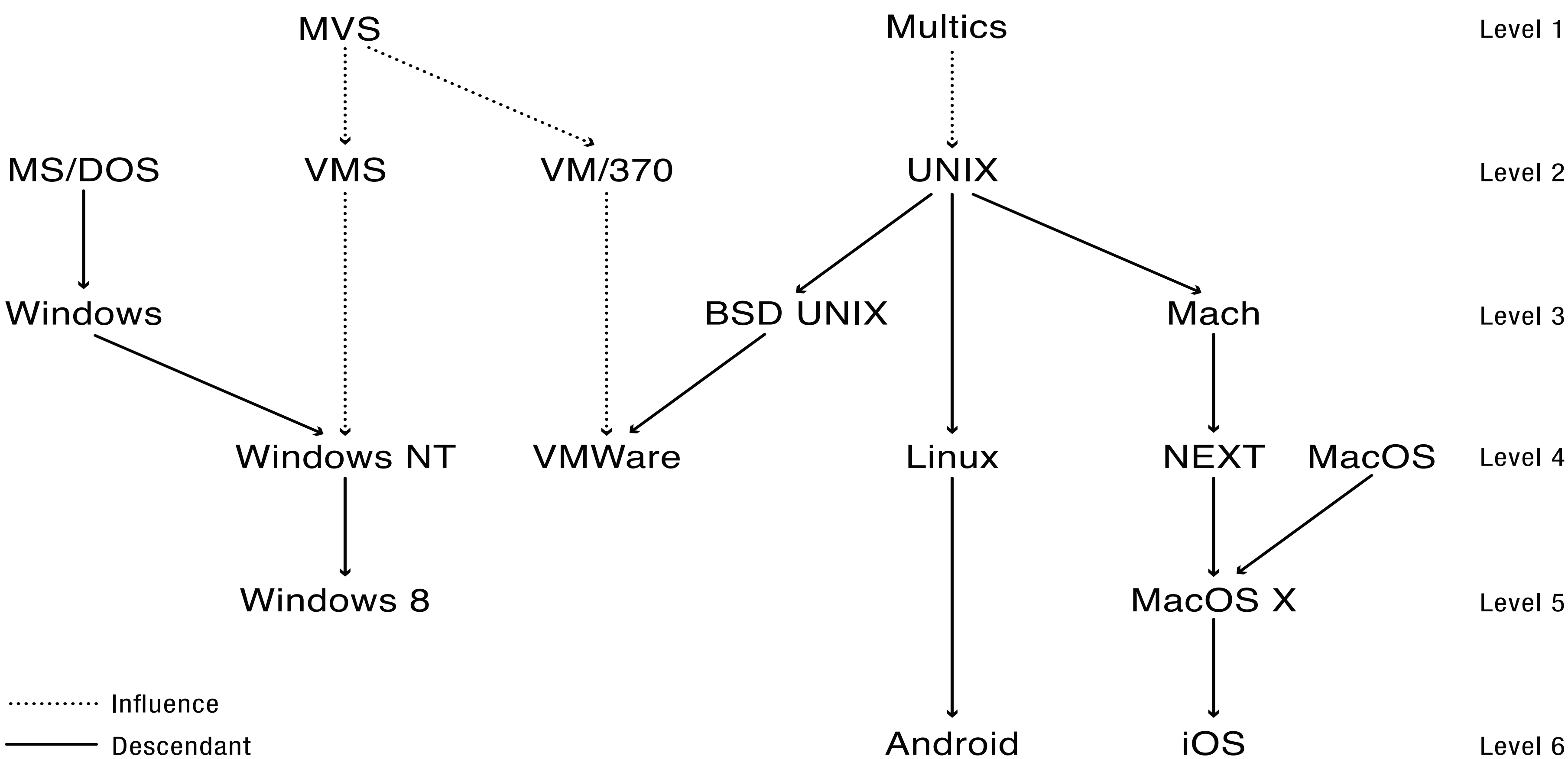
- Multicore or Manycore transition
  - Intel Xeon processor has 10 cores / 20 threads
  - Intel Xeon Phi has 72 cores, Core X goes up to 18 cores
  - nVidia GPUs has thousands of FPUs
- Accelerated need for software support
  - OS support for many cores
  - Parallel programming of applications



Now: Datacenter as A Computer

- Cloud computing
  - Hosting data in the cloud
  - Software as services
  - Examples:
    - Sales/HR/Payment apps, VoIP telephony …
- Utility computing
  - Pay as you go for computing resources
  - Outsourced warehouse-scale hardware and software
  - Examples:
    - Amazon, Google, Microsoft

OS history


MVS
Multics
Level 1
MS/DOS
VMS
VM/370
UNIX
Level 2
Windows
BSD UNIX
Mach
Level 3
Windows NT
VMWare
Linux
NEXT
MacOS
Level 4
Windows 8
MacOS X
Level 5
Android
iOS
Level 6
Influence
Descendant

Today

- Course information and logistics
- What is an operating system?
- Evolution of operating systems
- Why study operating systems?
  - Q: What are the two main reasons in your mind?

Why Study OS?

- OS is a key part of a computer system
  - It makes our life better (or worse)
  - It is “magic” to realize what we want
  - It gives us “power” (reduce fear factor)
- Learn how computer systems really work, who does what, how
- Learn key CS concepts: abstraction, layering, virtualization, indirection
- Learn about concurrency
  - Parallel programs run on OS
  - OS runs on parallel hardware
  - Great way to learn concurrent programming
- Understand how a system works
  - How many procedures does a key stroke invoke?
  - What happens when your application references 0 as a pointer?

Why Study OS?

- Basic knowledge for many areas
  - Networking, distributed systems, security, …
- Build an OS
  - Real OS is huge, but building a small OS will go a long way
- More employable
  - Become someone who “understands systems”
  - Join the top group of “athletes”
  - Ability to build things from ground up
  - Deeply understand abstractions, concurrency, virtualization

Does COS318 Require A Lot of Time?

- Yes
- But less than a few years ago
- But yes
- Q: Why are the two main reasons writing an OS is difficult?

Why is Writing an OS Hard?

- Concurrent programming is hard
- Difficult to use high-level programming languages for OS
  - device drivers are inherently low-level
  - lack of debugging support (use simulation)
  - real-time requirements
- Tension between functionality and performance
- Different contexts (mobile devices, data centers, embedded)
- Portability and backward compatibility
  - many APIs are already fixed (e.g., GUI, networking)
  - OS design tradeoffs change as hardware changes

Why is Writing an OS Hard

- Needs to be reliable
  - Does the system do what it was designed to do?
- Needs to keep the system available
  - What portion of the time is the system working?
  - Mean Time To Failure (MTTF), Mean Time to Repair
- Needs to keep the system secure
  - Can the system be compromised by an attacker?
- Needs to provide privacy
  - Data is accessible only to authorized users

Main Techniques and Design Principles

- Keep things simple
- Use abstraction
  - hide implementation complexity behind simple interface
- Use modularity
  - decompose system into isolated pieces
- Virtualize (for the magic)
- Keep things concurrent (for utilization and efficiency)
- What about performance?
  - find bottlenecks --- the 80-20 rule
  - use prediction and exploits locality (cache)
- What about security and reliability?
  - Continuing research, especially in light of new contexts

Things to Do

- Read the sections for next lecture (see web site)
- ~~Make “tent” with your name~~
  - ~~Use from now on till the end of the semester~~
- Use Piazza to find two partners for your group of threee
  - Find groups before end of next lecture for projects 1, 2, 3