Introduction to Operating Systems

COS 414/415

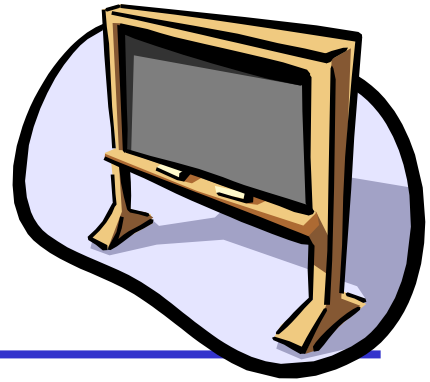
Spring 2004

Prof. E. Gün Sirer

Administrative

- Instructors:
  - Prof. Emin Gün Sirer, egs@cs, 4119A Upson, M 2:15-3:15
- Communication
  - cs414@cs.cornell.edu
- TAs:
  - Krzystof Ostrowski, krzys@cs
  - Bernard Wong, bwong@cs
  - Benjamin Ee
  - Dogan Famer Engin
  - Arjun Rao
- All official information will be on the web site
  - http://www.cs.cornell.edu/courses/cs414/2004SP/

414 Overview



- 414: Introduction to Operating Systems
  - Fundamentals of OS design
  - “Systems” comprise a broad range of topics; we will cover that range
  - Lectures will cover the basic material and supplement the textbook
- Textbook
  - Silberschatz & Galvin, Operating System Concepts, 6th Edition
- Weekly reading and homeworks
  - We will read some research papers in addition to the book chapters

415 Overview

- 415: Practicum in Operating Systems
  - Major programming assignment
  - Designed to complement the course and lectures
  - Also, to expose you to fun, cutting edge systems programming
  - Sections will meet before every assignment to go over the details and expectations

- Must enroll in both 414 and 415
- This year, we'll use tablet PCs for the project
- May work in pairs

Course Goals

- In this class we will learn:
  - What the parts of an OS are
  - How the OS and each sub-part is structured
  - What the important mechanisms are
  - What the important policies are
  - What algorithms and implementation techniques are typically used
- We will do this through reading, lectures, and a project
  - Project will involve some aspect of ubiquitous computing using tablet PCs
  - Reading: Chapters 1 & 2
- You will need to keep up with all three of these

Grading

- 414: Intro to Operating Systems
  - Reading Assignments (~10%)
  - Midterm (~30%)
  - Final (~50%)
  - Subjective criteria (~10%)
- 415: Practicum in Operating Systems
  - Six projects (100%)
- This is a rough guide

Academic Integrity

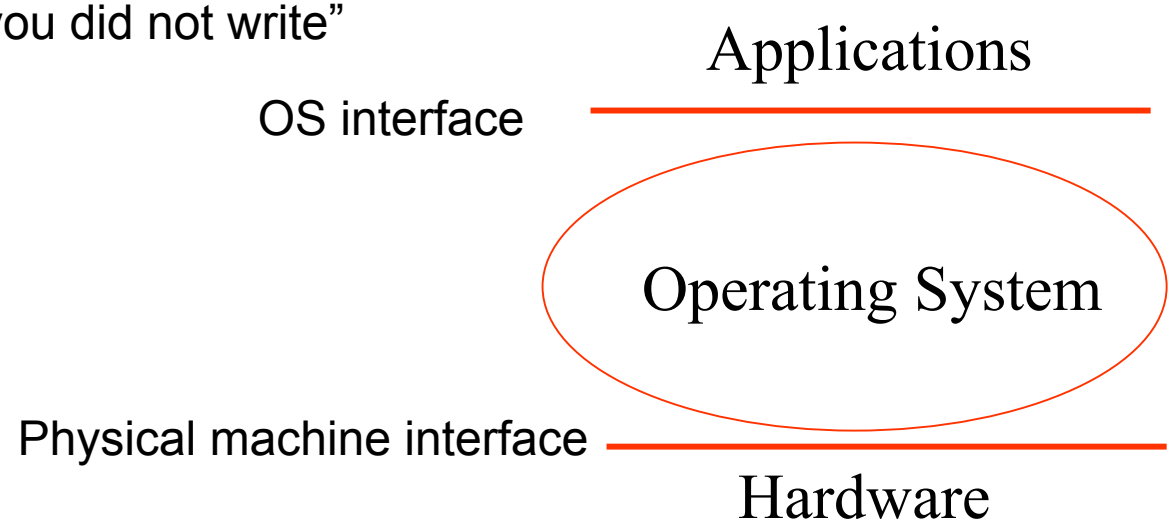
- Everything you turn in must be your own work
- Certain types of collaboration are part of the learning experience
  - May consult with others on C syntax, problem clarification, and debugging strategies.
- Dishonesty has no place in any community
  - May NOT be in possession of someone else's homework or project, may NOT copy, share or collaborate on answers to homework questions, may NOT copy code from another group, etc.
  - The academic integrity guidelines provide the general ground rules
  - The penalty is an immediate F in 414 and 415

Course Outline

- Architectural support
- Concurrency, processes, threads
- Synchronization, mutual exclusion, monitors, semaphores, condition variables, deadlocks
- Networking, distributed systems
- Memory Management, virtual memory
- Storage Management, I/O, filesystems
- Security
- Case studies

What is an Operating System?

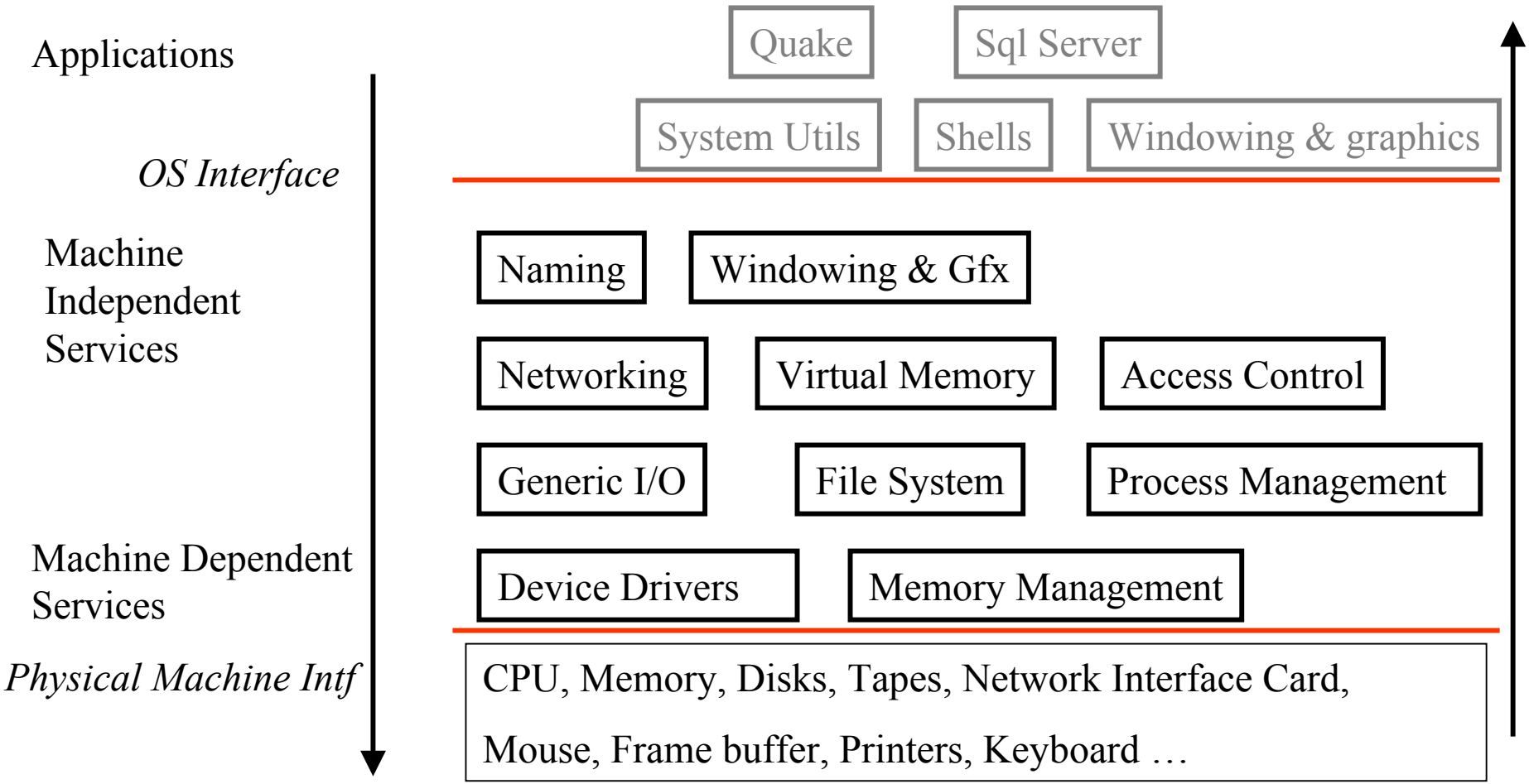
- Definition: An Operating System (OS) comprises all of the code that provides a virtual machine on top of the hardware. Typically, that virtual machine is more convenient (that is, has more desirable properties) than the physical machine
  - "All of the code you did not write"
  - Simpler
  - More reliable
  - More secure
  - More portable
  - More efficient
  - …



What do Operating Systems Do?

- Manage physical resources
  - Drive devices
  - Examples: CPU, memory, disks, tapes, networks, displays, keyboards, mice, cameras, speakers, printers…
- Provide abstractions and a set of well-defined operations on those abstractions
  - Provide synthesized, or virtual, resources and interfaces
  - Examples: files, directories, users, processes, threads, sockets, etc.
  - Simplify programming by hiding complexity through high-level abstractions
  - Improve portability by taking over low-level programming tasks from applications
  - Provide users with a well-behaved environment, mask failures
- Isolate and mediate between entities
  - Act as a trusted intermediary between distrusting applications

What's in an OS?



Logical OS Structure

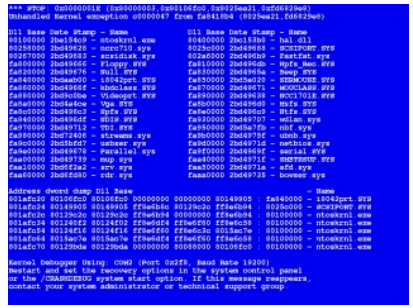
Issues in Operating Systems

- **Structure --** how is an operating system organized ?
- **Concurrency** -- how are parallel activities created and controlled ?
- **Sharing --** how are resources shared among users ?
- **Naming --** how are resources named by users or programs ?
- **Protection --** how is one user/program protected from another ?
- **Security --** how to authenticate, control access, and secure privacy?
- **Performance --** why is it so slow ?
- **Reliability and fault tolerance –** how do we deal with failures ?
- **Extensibility --** how do we add new features ?
- **Communication --** how can we exchange information ?

Issues in OS (2)

- **Scale and growth --** what happens as demands or resources increase ?
- **Persistence --** how to make data outlast the processes that created them
- **Compatibility --** can we ever do anything new ?
- **Distribution --** accessing the world of information
- **Accounting --** who pays the bills, and how do we control resource usage?

Why is this material critical ?



- Concurrency
  - Therac-25, Shuttle livelock
- Communication
  - FAA Air traffic control system
- Persistence
  - Denver airport
- Virtual Memory
  - Blue Screens of Death
- Security
  - IRS

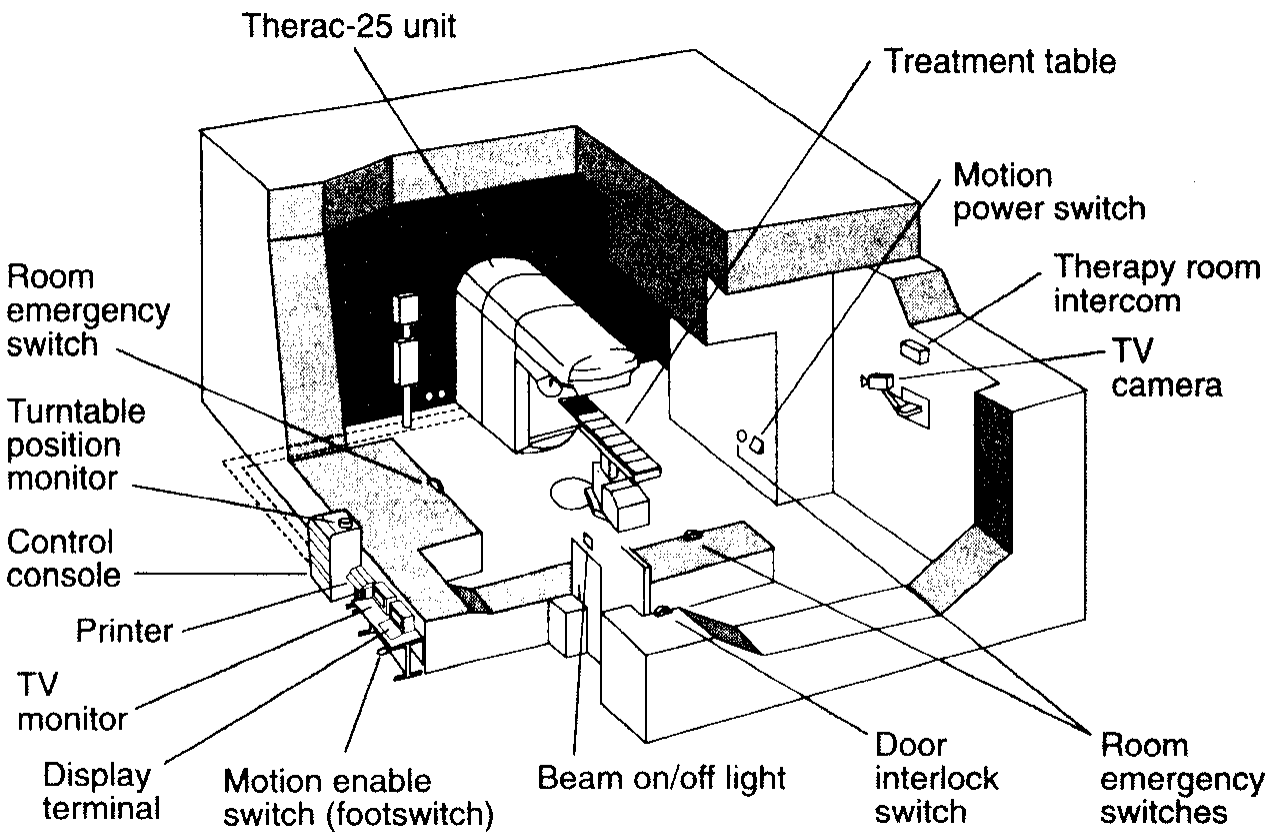
BSOD


Delta Air Lines
FIRST CLASS
154
153
Delta
quakecon 2002 - mesquite, texas, USA - august 2002 - photo: yossarian holmberg (yossman@yossman.net)

BSOD

Photo courtesy of zem.

Therac-25



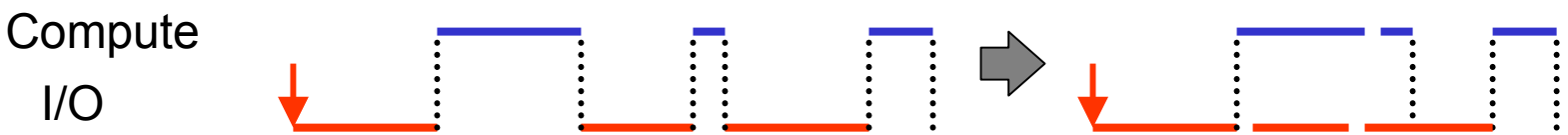
- A safety-critical system with software interlocks
- Beam controlled entirely through a custom OS

Therac-25

- A synchronization failure was triggered when competent nurses used the back arrow to change the data on the screen “too quickly”
- Beam killed one person directly, burned others, and may have given inadequate treatment to cancer patients
- Problem was very difficult to diagnose; initial fix involved removal of the back arrow key from the keyboard
- People died because a programmer could (or did) not implement correct synchronization primitives
- This class will ensure that you will understand the science and engineering behind building complex systems

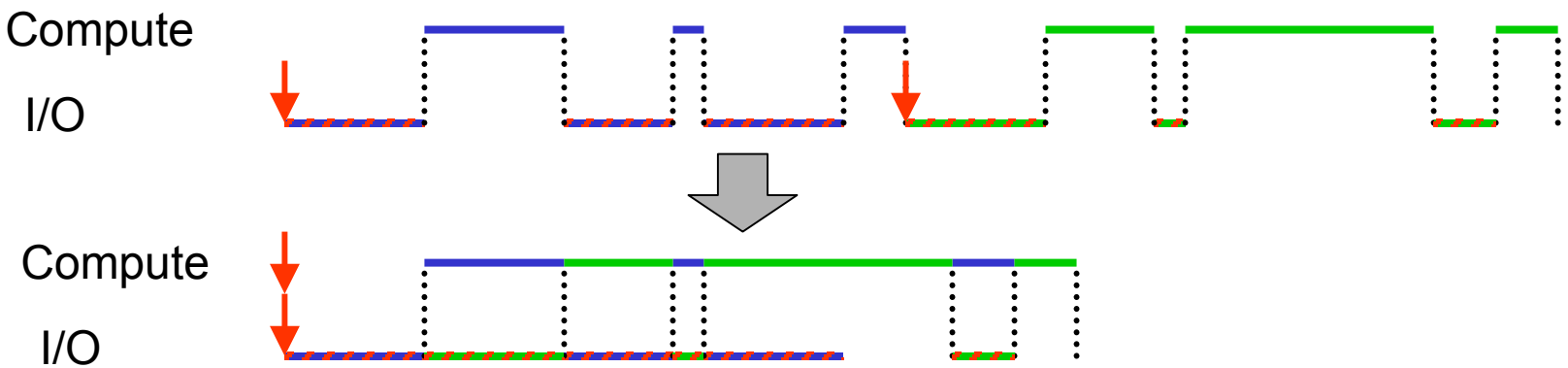
A Brief History of Operating Systems

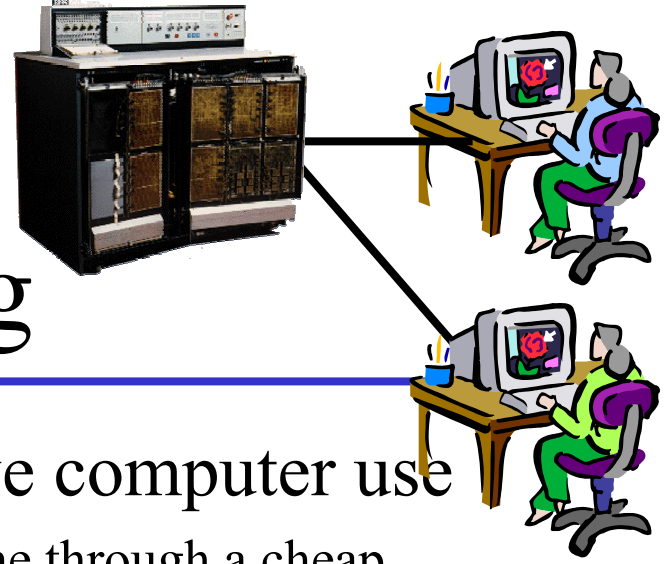
- Initially, the OS was just a run-time library
  - You linked your application with the OS, loaded the whole program into memory, and ran it
  - How do you get it into the computer ? Through the control panel!
- Simple batch systems
  - Permanently resident OS in primary memory
  - It loaded a single job from card reader, ran it, and loaded the next job...
  - *Control cards* in the input file told the OS what to do
  - *Spooling* allowed jobs to be read ahead of time onto tape/disk or into memory



Multiprogrammed BatchSystems

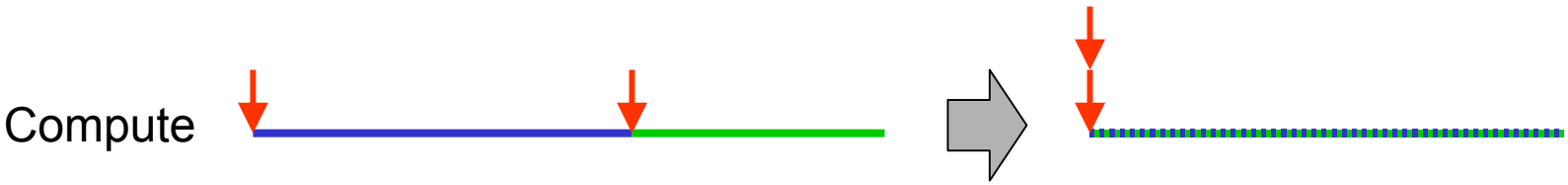
- *Multiprogramming* systems provided increased utilization
  - Keeps multiple runnable jobs loaded in memory
  - Overlaps I/O processing of a job with computation of another
  - Benefits from I/O devices that can operate asynchronously
  - Requires the use of interrupts and DMA
  - Optimizes for *throughput* at the cost of *response time*





Timesharing

- *Timesharing* supported interactive computer use
  - Each user connects to a central machine through a cheap terminal, feels as if she has the entire machine
  - Based on time-slicing -- dividing CPU equally among the users
  - Permitted active viewing, editing, debugging, participation of users in the execution process
  - Security mechanisms required to isolate users from each other
  - Requires memory protection hardware for isolation
  - Optimizes for *response time* at the cost of *throughput*



Personal Computing

- Computers are cheap, so give everyone a dedicated computer
- Initially, the OS became a library again due to hardware constraints
- Multiprogramming, memory protection, and other advances were added back
    - For entirely different reasons

Parallel Operating Systems

- Support parallel applications wishing to get speedup of computationally complex tasks
- Needs basic primitives for dividing one task into multiple parallel activities
- Supports efficient communication between those activities
- Supports synchronization of activities to coordinate sharing of information
- It's common now to use networks of high-performance PCs/workstations as a parallel computer

Distributed Operating Systems

- Distributed systems facilitate use of geographically distributed resources
    - Machines connected by wires, no shared memory or clock
- Supports communication between parts of a job or different jobs
    - Interprocess communication
- Sharing of distributed resources, hardware and software
    - Resource utilization and access
- Permits some parallelism, but speedup is not the issue

Real-time Operating Systems

- Goal: To cope with rigid time constraints
- Hard real-time
  - OS guarantees that applications will meet their deadlines
  - Examples: TCAS, health monitors, factory control, etc.
- Soft real-time
  - OS provides prioritization, on a best-effort basis
  - No deadline guarantees, but bounded delays
  - Examples: most electronic appliances
- Real-time means “predictable”
  - NOT fast

Ubiquitous Computing

- The decreased cost of processing makes it possible to embed computers everywhere. Each "embedded" application needs its own control software:
  - PDAs, cell phones, intelligent appliances, etc.
- In the near future, you will have 100s of these devices
  - If not already
- Poses lots of problems for current systems
  - Structure, naming, scaling, security, etc.
- We will tackle some of them in this class

Lessons from History

- The point is not that batch systems were ridiculous
  - They were exactly right for the tradeoffs at the time
- The tradeoffs change

	1981	2001	Factor
MIPS	1	1000	1000
$/MIPS	$100000	$5000	20000
DRAM	128KB	256MB	2000
Disk	10MB	80GB	8000
Net Bandwidth	9600 b/s	100 Mb/s	10000
# Users	>> 10	<= 1	0.1

- Need to understand the fundamentals
  - So you can design better systems for tomorrow's tradeoffs

Future

- …