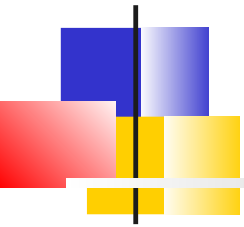
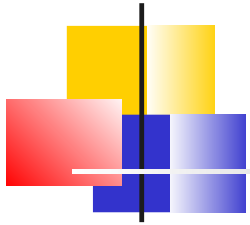


CAS CS 552
Intro to Operating Systems

Richard West

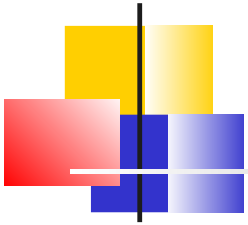
Course Overview and Introduction



Course Summary 1/2

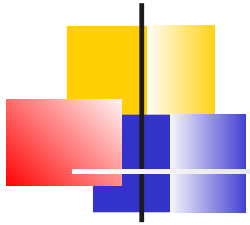
- This course covers the fundamental concepts of operating systems. Topics including OS structure, processes/thread management, synchronization, deadlocks, file systems, I/O and memory management will be discussed

- **A good understanding of C or C++ is required**. A prior understanding of assembly programming will be useful, especially IA32 assembly, but this is not prerequisite knowledge



Course Summary 2/2

- Advanced topics including those based on distributed computing will be discussed, time permitting. Additionally, case studies based on a real-world operating system (e.g., Linux) will be covered, where appropriate, throughout the course
- You will be required to tackle projects that involve kernel-level programming of existing systems such as Linux, and possibly writing your own OS abstractions/features on bare-bones hardware. In the latter case, we will use PC emulation tools such as Bochs, QEMU and VMWare player, time permitting.
  - Our own OS called “Quest” may also be used
- Socket programming and abstractions such as remote procedure calls may be required for some of the projects
- Prerequisites: CS210 (or the consent of the instructor)
  - CS350 is helpful

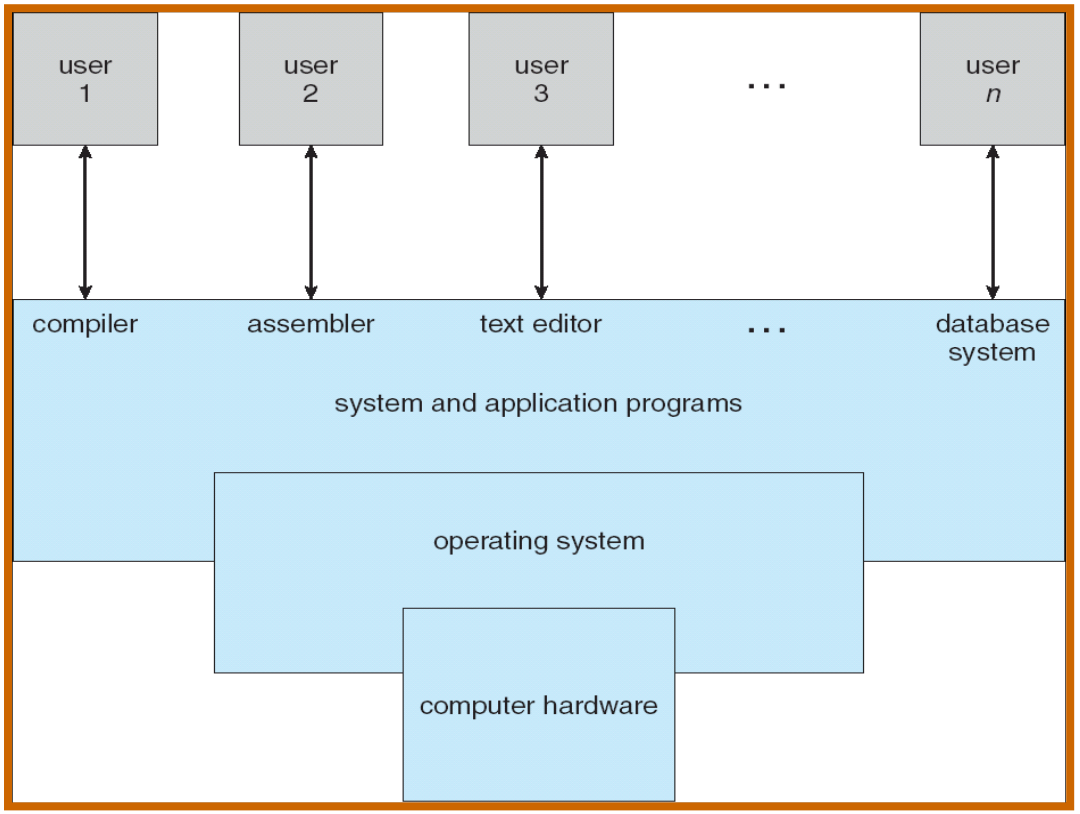


Basic OS Concepts

- Operating system: program between user-level and hardware
  - Allows users to execute progs
  - Aims to provide protection, reliability, efficiency, predictability and convenience
  - “transparency” is a key issue – devil is in the details

Computer Systems

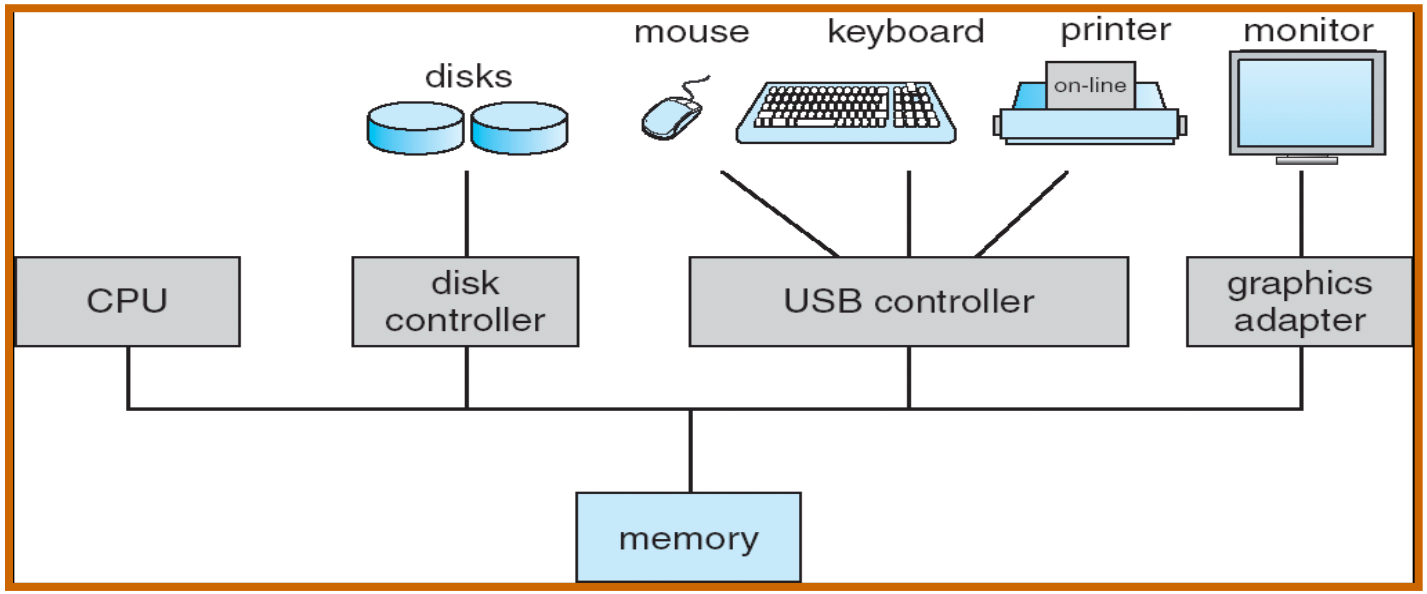
- Computer system: hardware, OS, system/app programs, users



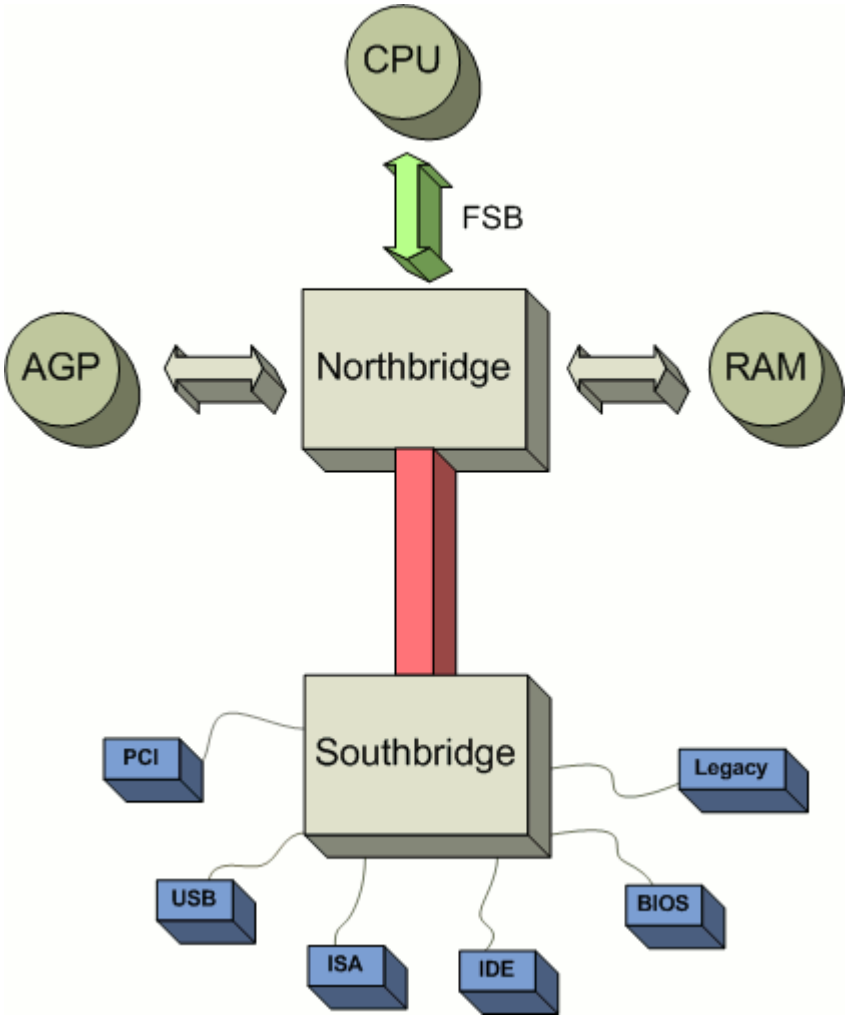
Computer System Organization

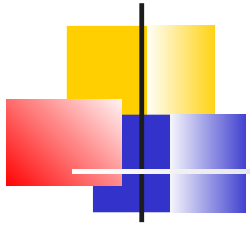
Computer-system operation

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



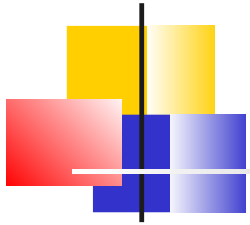
Example – PC Chipset Architecture





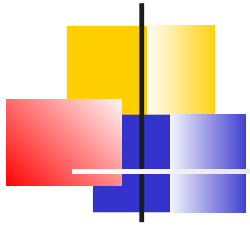
Role of OS 1/2

- Controls and coordinates use of hardware amongst multiple progs for various users
- Provides environment for programs to execute
- Acts as a resource allocator
  - Allocates processors, memory, I/O devices, file storage space etc to progs/users as necessary
  - (Typically) must manage competing requests for resources efficiently and fairly
    - Maybe predictably too (more on this later)



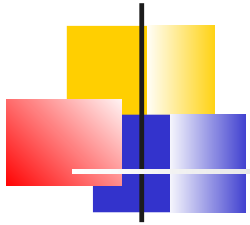
Role of OS 2/2

- OSes can be viewed as control programs
  - Control execution of user progs to prevent “improper” use of computer’s resources
  - Provide convenience
    - Make it easier for users to access common resources without awareness of hardware specifics



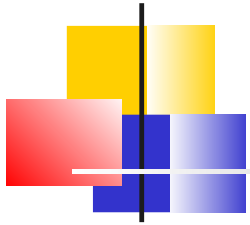
Multiprogrammed Systems

- Multiprogramming – use of CPU by more than one job (task/process/thread)
  - e.g., while one job is blocked waiting for an I/O device another may use the CPU
  - Aim to keep CPU busy – efficiency issue
  - Multiple jobs are resident in main memory at any given time and CPU(s) is(are) multiplexed between them



Time Sharing Systems

- Timesharing (or multitasking) implies many users simultaneously share computer
  - Like multiprogramming
  - CPU switches between multiple jobs
  - Job switches are frequent, allowing users to interact with program during execution
  - User-computer interaction –
    - E.g. input via keyboard, output via display device
    - Typically requires a file system to store data + code
      - Files store a collection of related data e.g., text, programs etc
  - Interactive systems require short response times to user requests (typically a few seconds at most)



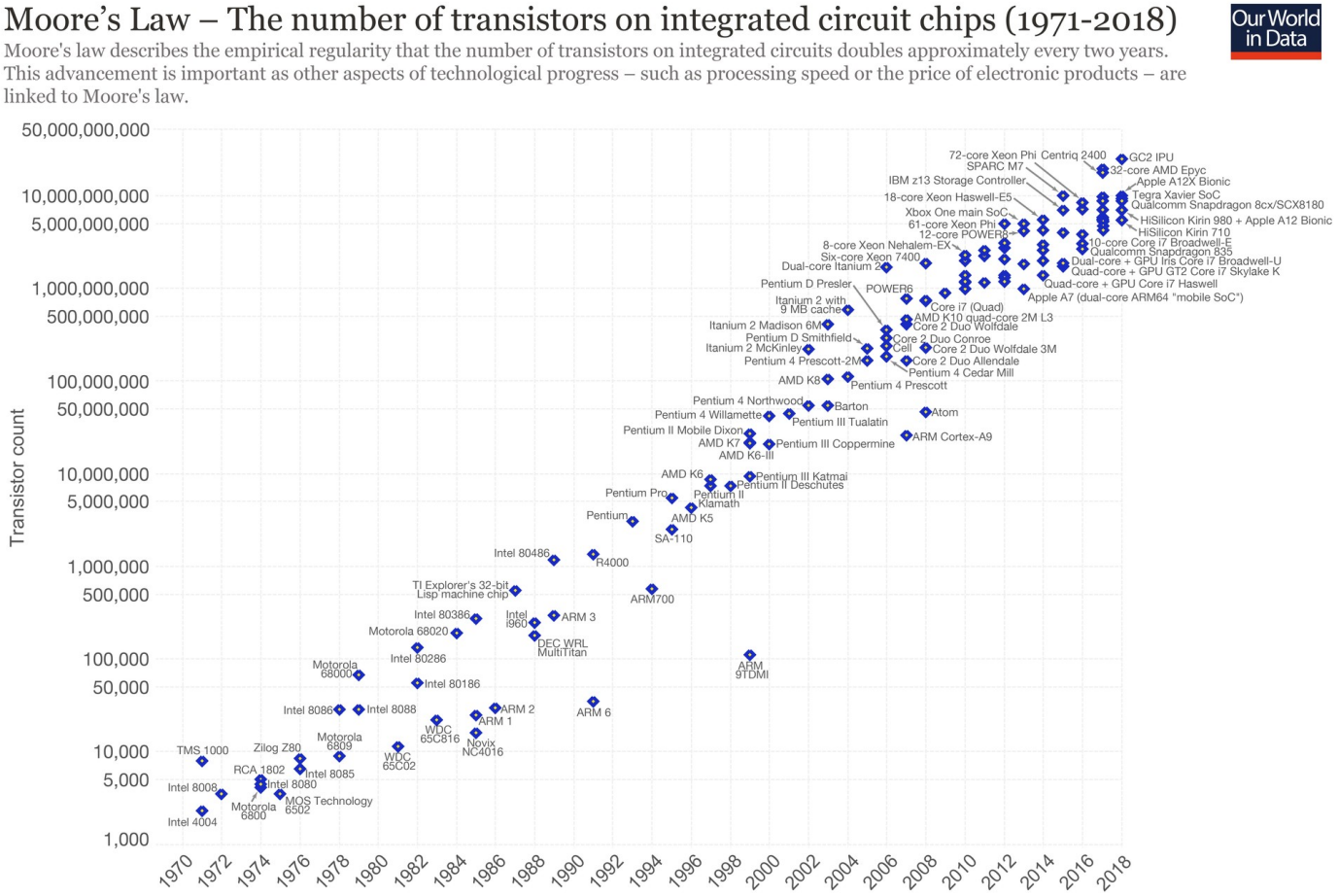
Personal Computer Systems

- As computer costs decreased, became feasible to have a machine for one user again i.e., personal computers

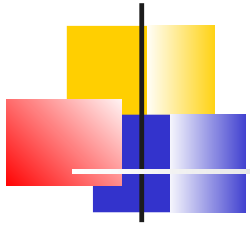
- PC OSes aim to provide convenience in preference to resource efficiency (or high resource utilization)
  - Possible due to reduced costs of e.g., memory and CPUs
    - Moore’s Law
    - PC OSes include Windows (NT,95,98,Me,Vista,7-10), Mac OS X, Linux

Moore’s Law

- Double transistor density every 18 mo – 2 yrs



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.



Example Past OSes

- 1951 – LEO 1 (Lyon's UK) for EDSAC
- 1955 – General Motors OS for IBM 701
- 1961 – Burroughs MCP
- 1962 – Atlas Supervisor (for Atlas Supercomputer)
  - One-level store (a.k.a. paged virtual memory)
- 1964-66 – OS/360 IBM, TOPS-10 DEC, Berkeley Timesharing System
- 1965 – T.H.E. Dijkstra et al
- 1965-1970 Multics (MIT)
  - Bell Labs – UNIX
- Circa 1970 UNIX for PDP-11 minicomputer

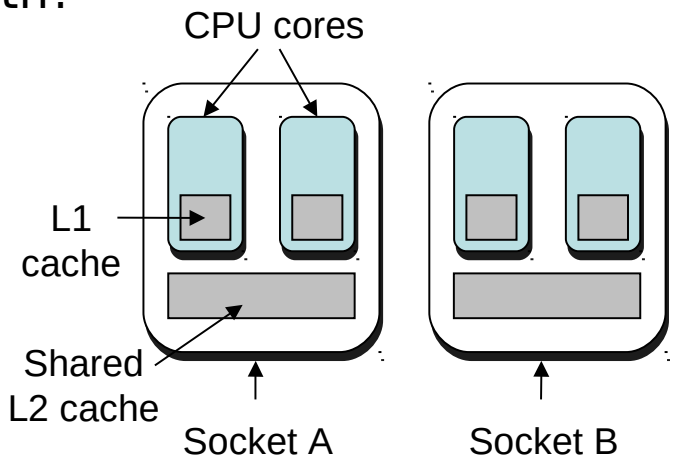
- (More history later…)
- Today, UNIX-like features in PC OSes

Parallel Systems (1)

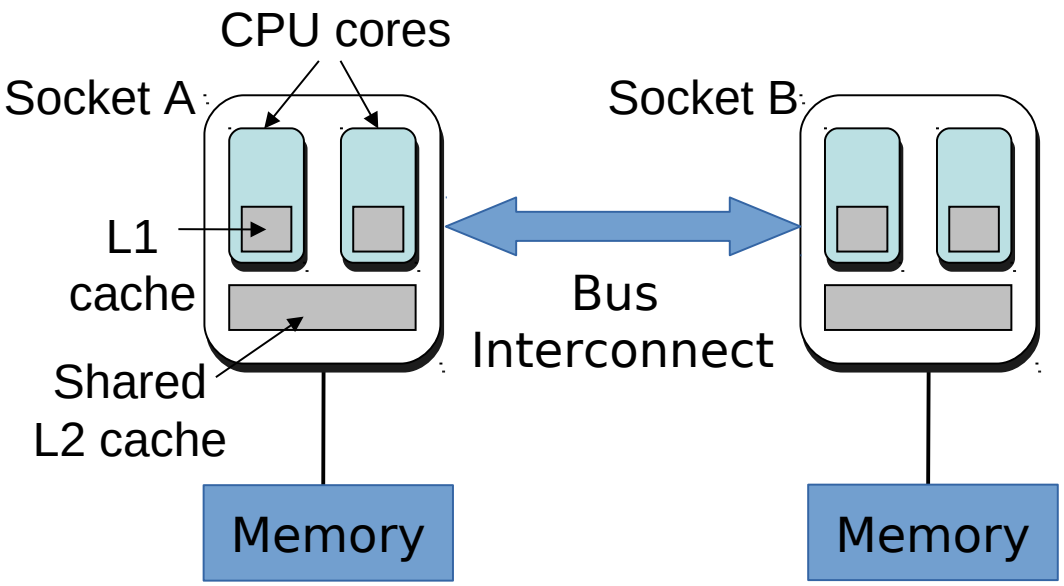
- Multiprocessor systems have more than one CPU
- Modern chip multi-processors (CMPs) have multiple cores in the same chip package

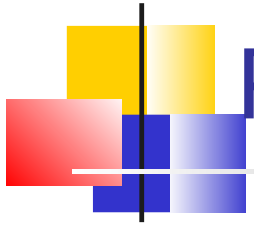
  - Parallelism goal: to provide computational speedup (of program execution)
  - Multiple processors can share prog data, stored on same disks and kept in physically shared memory, rather than spread across multiple machines

  - But what about shared caches? NUMA effects? Memory bus bandwidth contention and so forth?



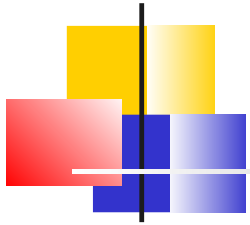
Non-Uniform Memory Access





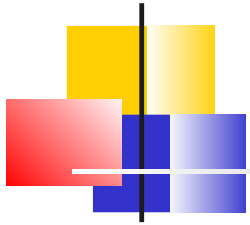
Parallel Systems (2)

- Provide fault-tolerance/reliability – failure of one CPU only slows down execution rather than resulting in system failure
- Symmetric multiprocessing (SMP) – each processor runs an identical copy of OS
  - CPUs communicate as necessary to coordinate resource management
  - Most modern OSes (Linux, Windows, OS X, Solaris) are SMP-compliant these days
- Asymmetric multiprocessing – each processor is assigned a specific task
  - Master processor controls system and slave processors to perform various tasks/services



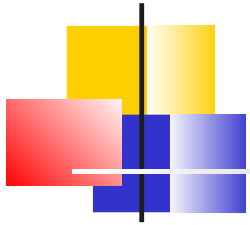
Distributed Systems

- Loosely coupled processors on different machines communicate across networks and collectively perform a distributed computation
  - No physically shared memory or common clock
- Enable sharing of remote resources
- Provide computational speedup
- Reliability through replication/redundancy



Real-Time Systems

- Timeliness constraints on operation of a processor or execution of a job/task
  - Program correctness depends not only on the result of execution but when response is generated
  - Hard real-time systems – missing any single time constraint (deadline) can result in failure of the application/system e.g., flight navigation/control, factory automation, nuclear power plant
  - Soft real-time systems – can tolerate certain fraction of deadline misses at cost of reduction in perceived quality/utility e.g., multimedia systems



Other Systems?

- Embedded
  - Often real-time, typically supporting only one application or dedicated function (not general purpose)
    - Found in avionics, automotive and manufacturing environments, medical equipment, and now PDAs, mobile phones…
- Peer-to-peer (P2P)
  - No traditional backend server for a number of clients
  - Each node in distributed system considered a “peer”
  - Popular for information storage, lookup and retrieval
    - e.g., Gnutella, Kazaa, Chord, CAN, Pastry, Bittorrent
    - Made popular as a result of file sharing (especially mp3 audio files)
- Mobile Oses: Android, iOS, WebOS, Symbian, Windows Phone 7 to 8.1, RIM, Maemo, …