CSCI 3753
Operating Systems

Introduction

Lecture Notes By
Shivakant Mishra
Computer Science, CU-Boulder
Last Update: 01/14/14

Operating Systems

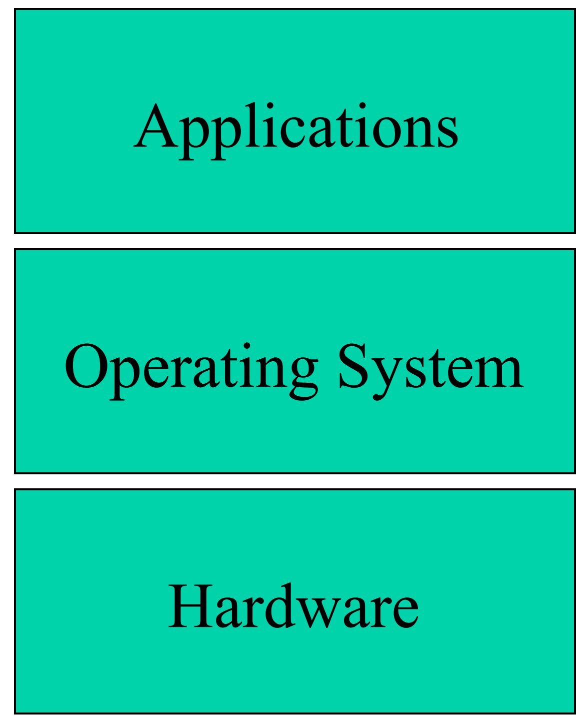
- OS is an essential (and perhaps the most important) part of a computing system.
    - PCs, laptops, supercomputers, cloud, data centers, cell phones, PDAs, tablets, embedded devices, …
    - Controls hardware components.
    - Provides a usable interface.
    - Allows sharing of various computing components.
- One of the largest piece of software.

Windows, Linux, Mac OS X, Google Android, …

> 500 at http://www.operating-system.org/betriebssystem/_english/os-liste.htm

What is an Operating System?

An operating system is a layer of software between applications and hardware that provides useful services to applications

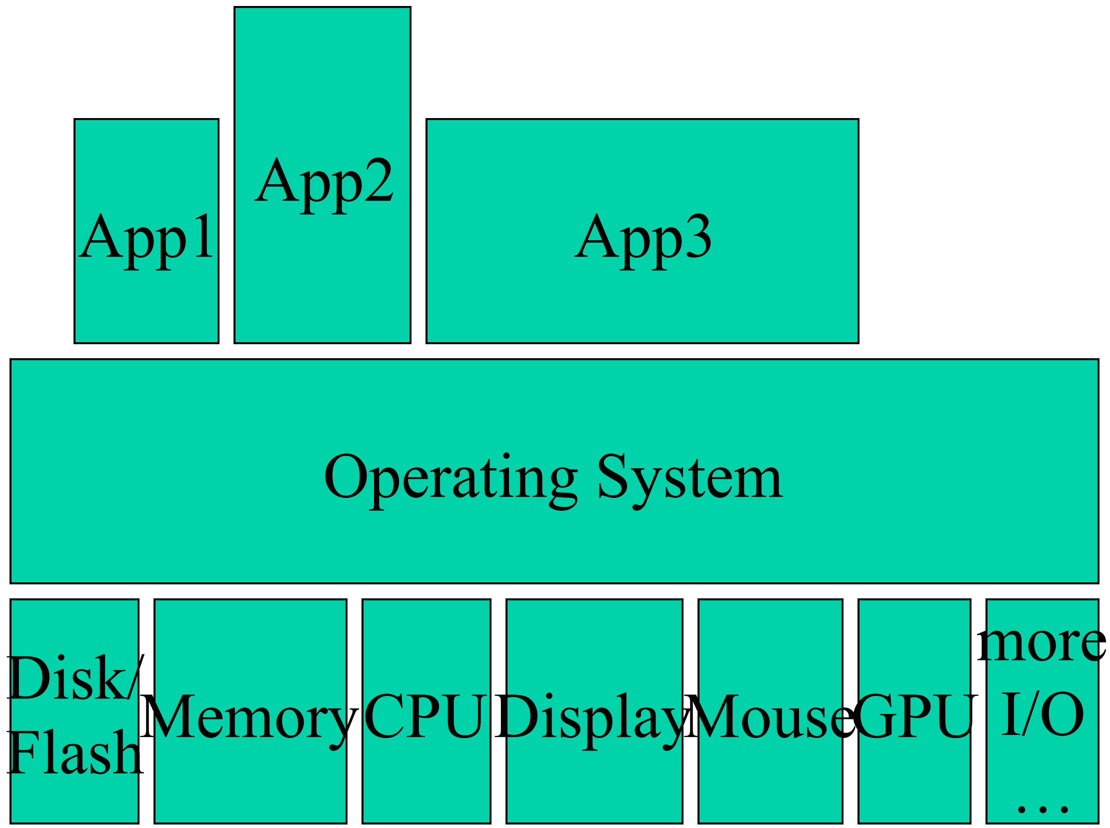


Two views of an OS

- Extended machine
  - Hardware is too complex for most computer users to understand.
  - OS provides users with an equivalent of an extended or virtual machine that is easier to use.
- Resource manager
  - A computing system is comprised of many resources: processors (CPUs), memory, clocks, disks, key board, mouse, monitor, network cards, etc.
  - OS allows sharing and effective utilization of these resources.

Security and protection

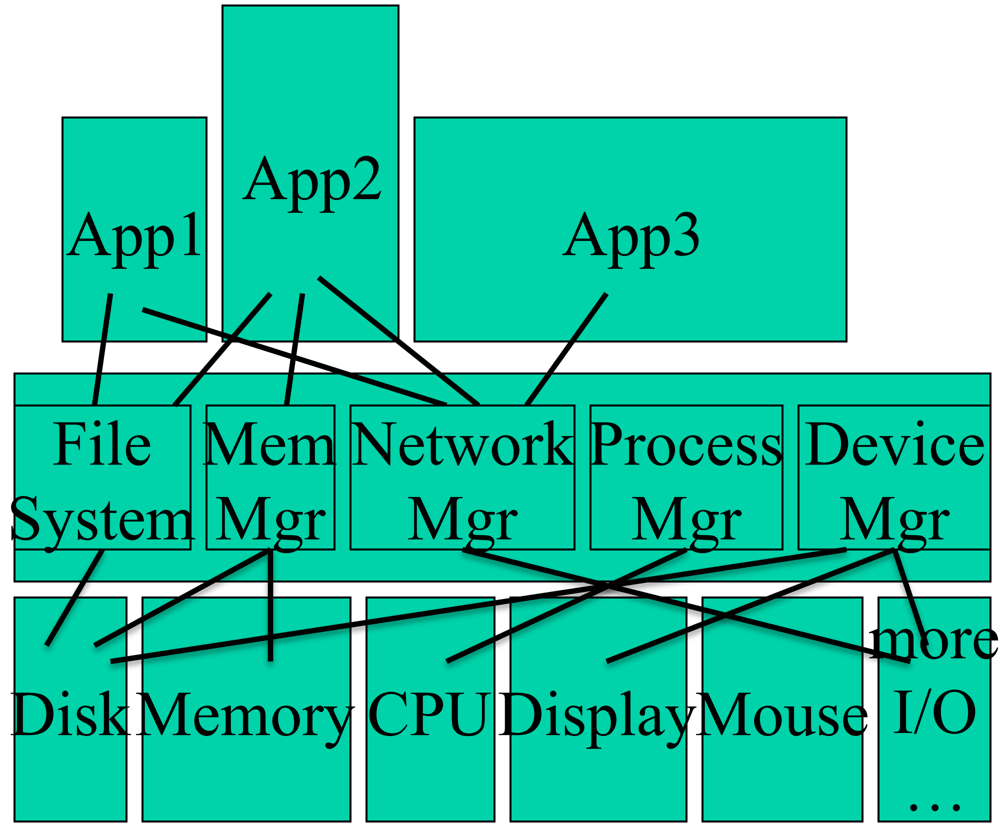
What is an Operating System?



Other devices include: wired network card, WiFi, camera, microphone, audio output, keyboard, DVD/CD, USB, etc.

OS as Resource Manager

- Process management
- Memory management
- File system
- Device management
- Network management



Extended Machine View

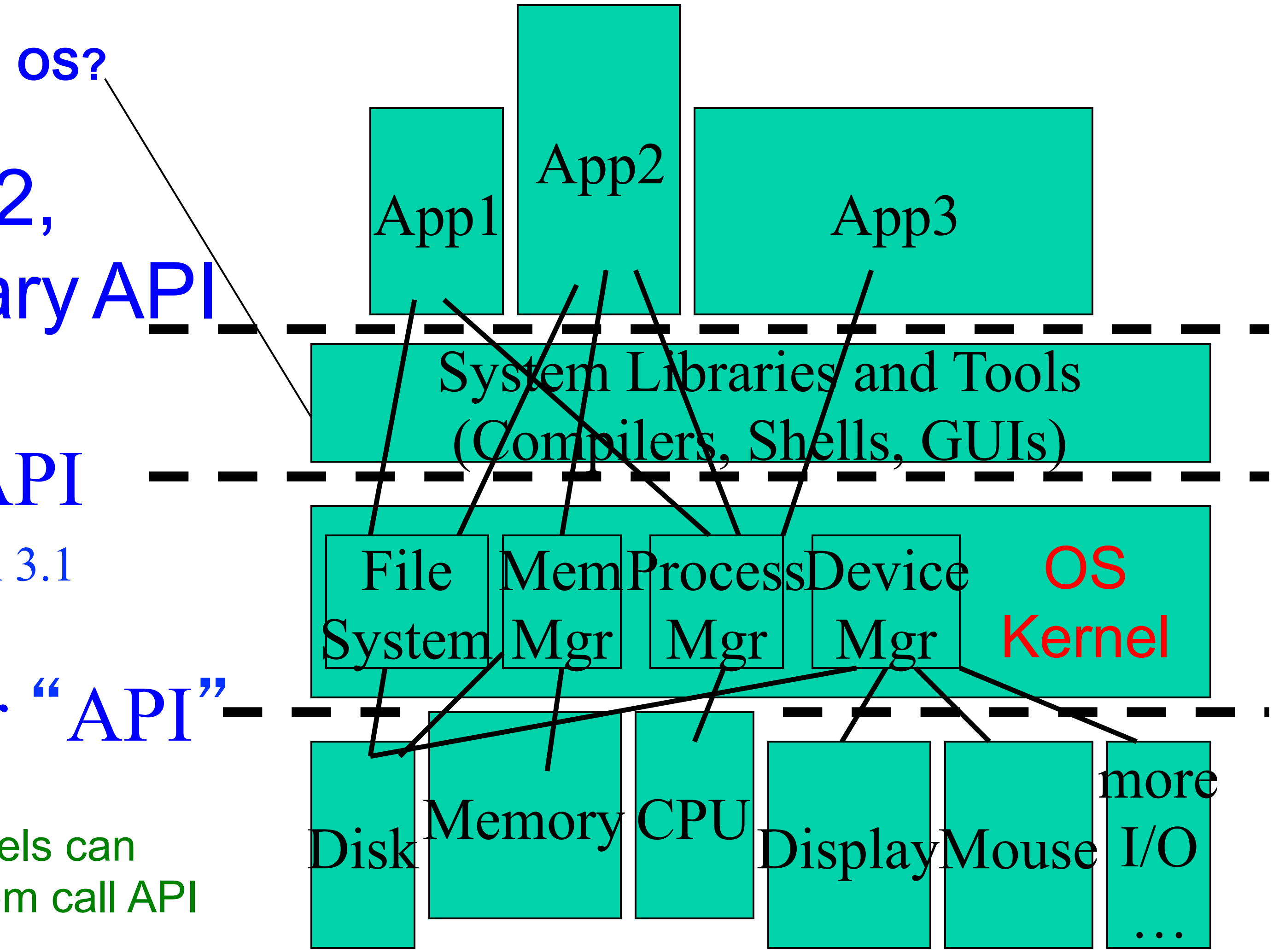
Posix, Win32, Java, C library API

System call API

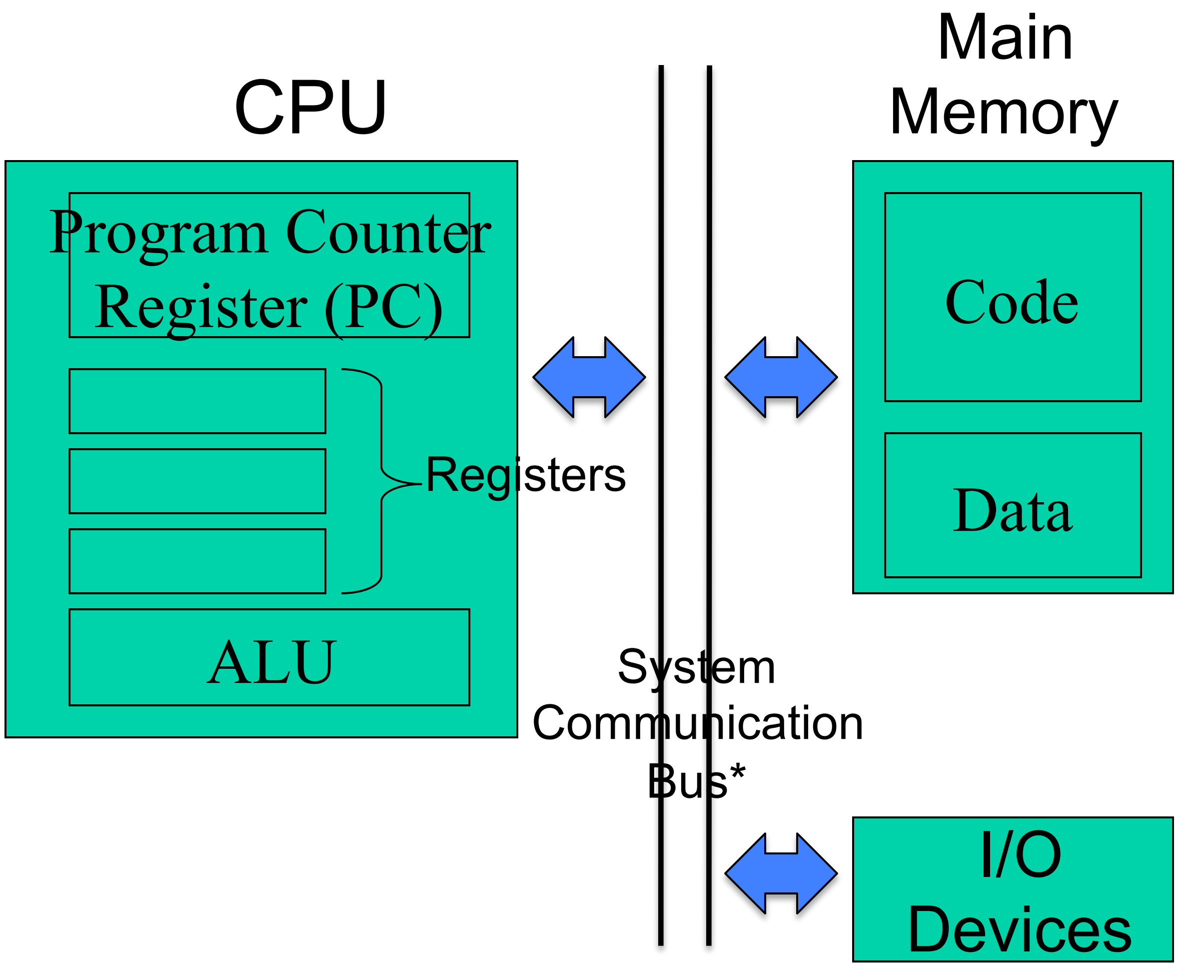
- >300 in Linux 3.1

Device driver "API"

Note: different OS kernels can support the same system call API

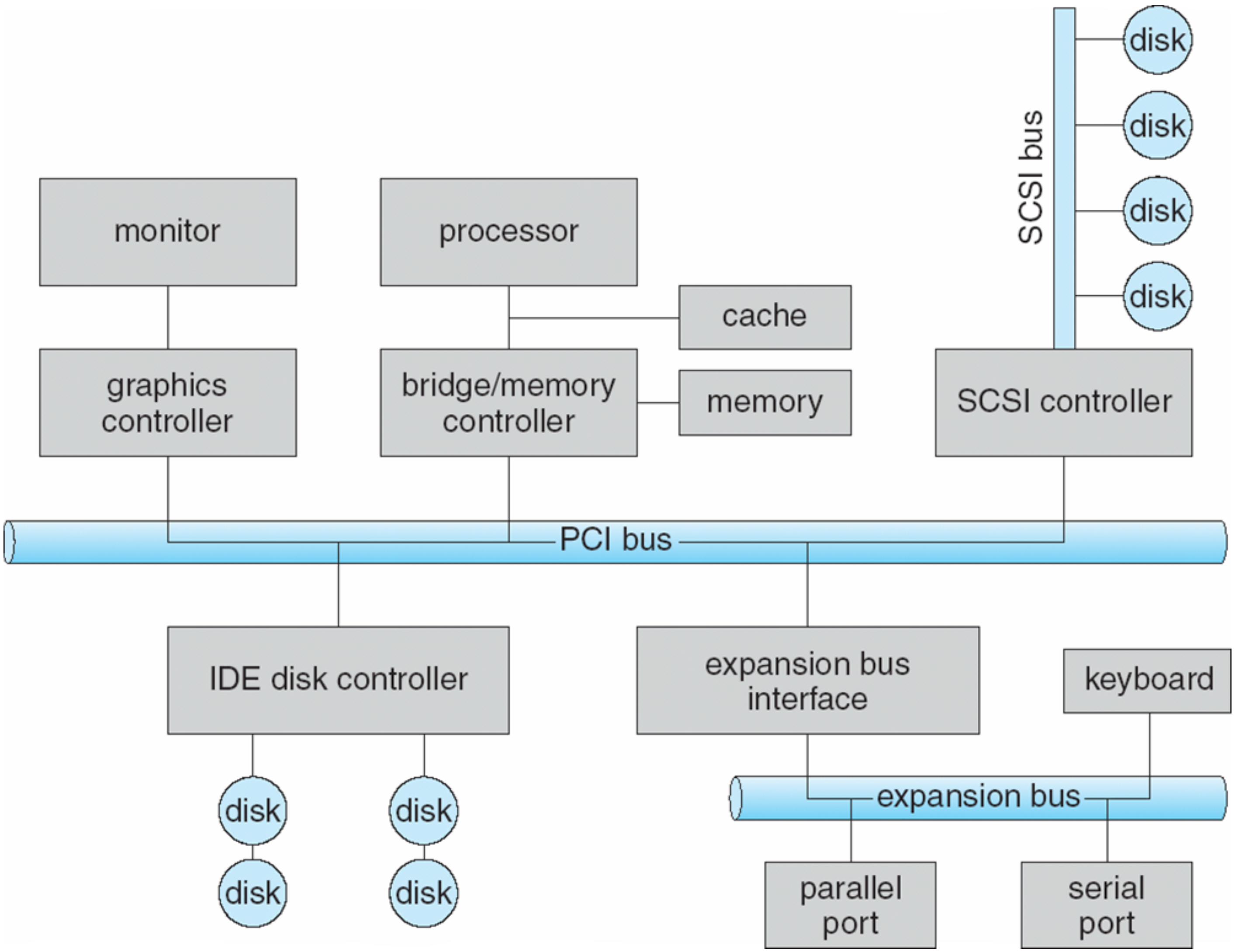


The von Neumann Computer's Hardware Architecture and I/O



* Includes control, address, and data buses

A Typical PC Bus Structure



Evolution of OS

- 1940's and 50's
  - Vacuum tubes
  - Single user
  - No programming language: machine language
  - Used for straight-forward numerical calculations
  - Single program
  - No OS
- mid 50's and early 60's
  - Transistors (more reliable, smaller in size, cheaper)
  - Punch cards
  - **Batch processing**
  - Used for scientific and engineering calculations.
  - FORTRAN

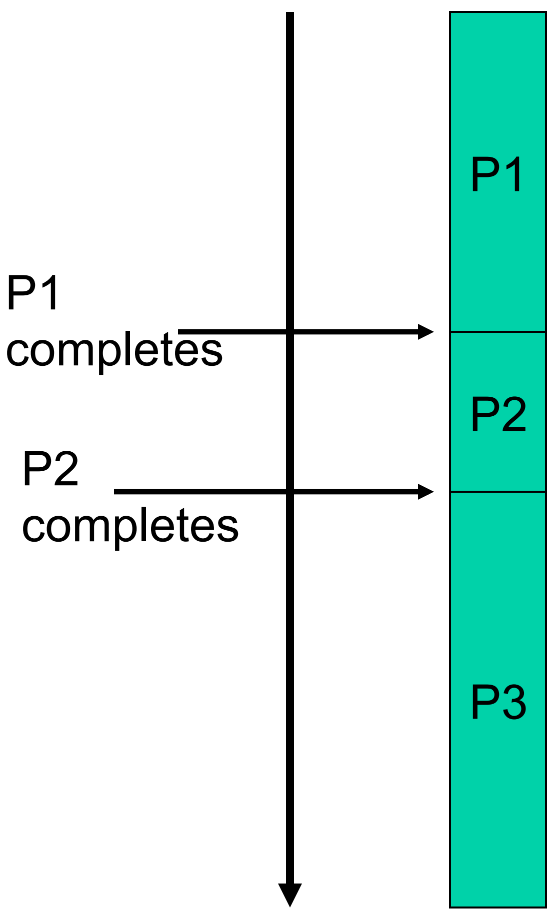
Batch Processing

- Execute a pre-defined collection of programs (jobs) called a batch.
- No human interaction

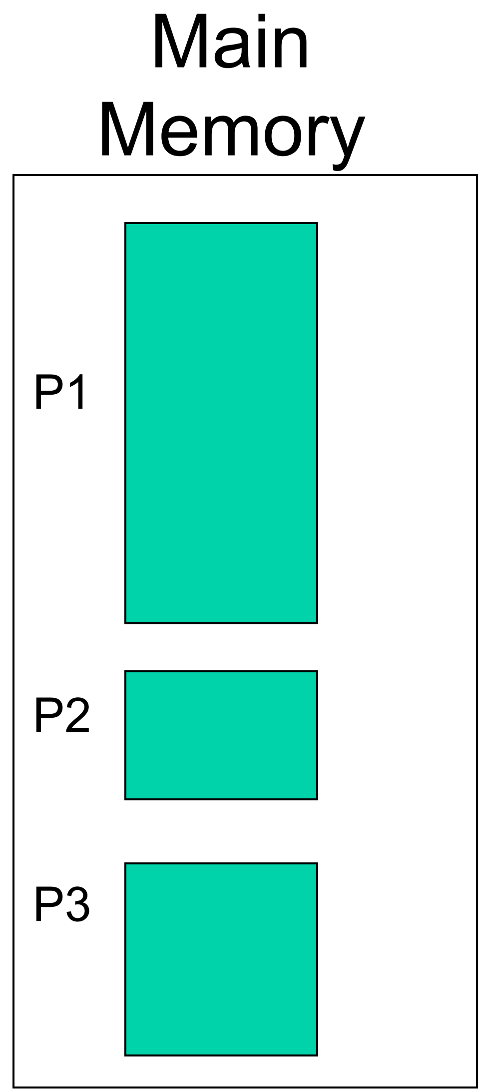
Batch Processing

Time

Programs Executing on CPU



Let programs execute *sequentially*. When P1 is done, P2 takes over, and so on. In this case, P1, P2, and P3 are all *compute-bound*, i.e. executing continuously on the CPU until they’ re done

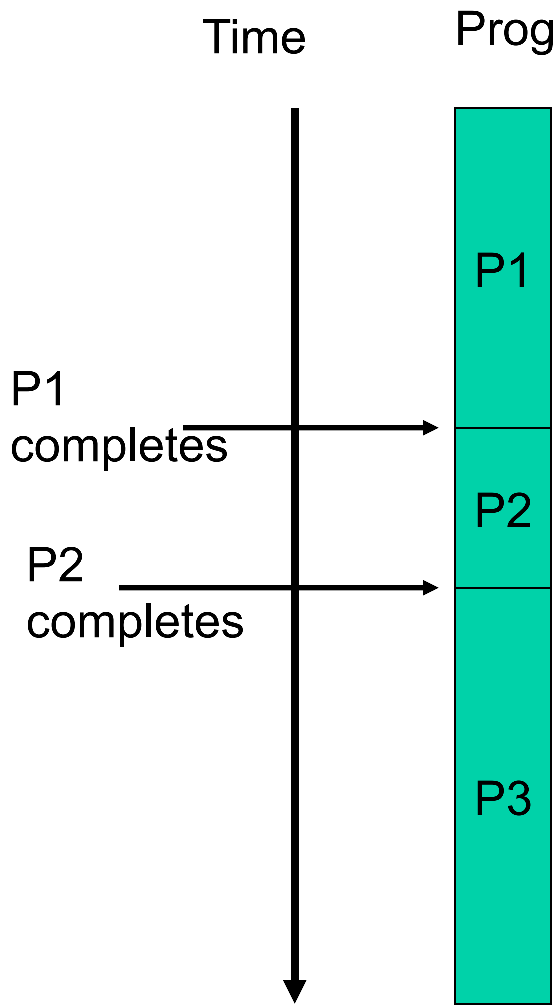


Limitations of Batch Processing

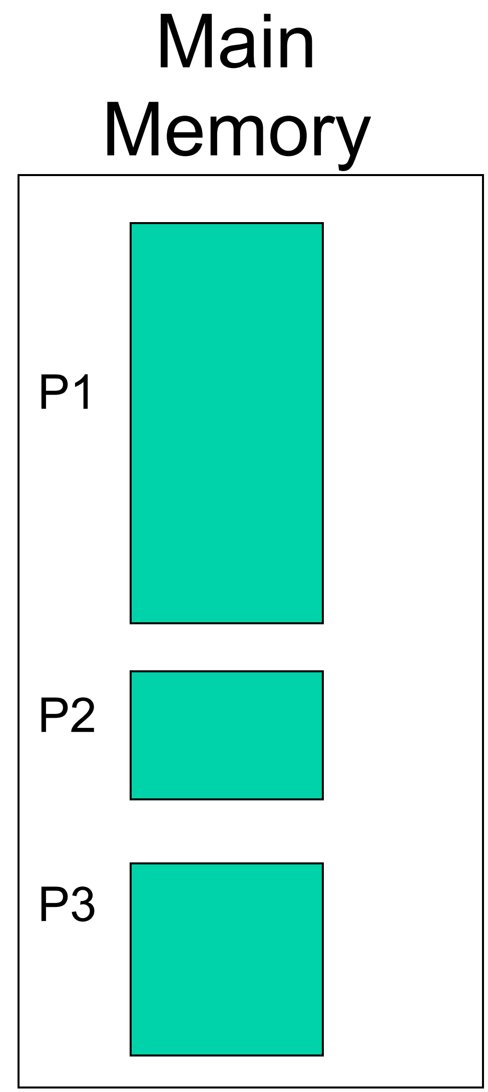
- Batch jobs are very non-interactive
  - Don't support a shell application for example
  - design jobs to yield much sooner than an I/O block, to give the impression of interactivity
  - If one of the programs was a shell, then it appears to the human user as if the computer is instantly responsive
  - In the small time segment a shell is given, it can draw a character on the screen that you've just typed => appearance of real-time interactivity

- mid 60’s to mid 70’s
  - Integrated circuits.
  - **Multiprogramming:** When CPU is idle, e.g. when the running program is blocked for an I/O, start another program.
  - **Multitasking:** Switch CPU between different programs irrespective of whether the running program is blocked.
  - Examples: CTSS, IBM 360, Multics, Unix

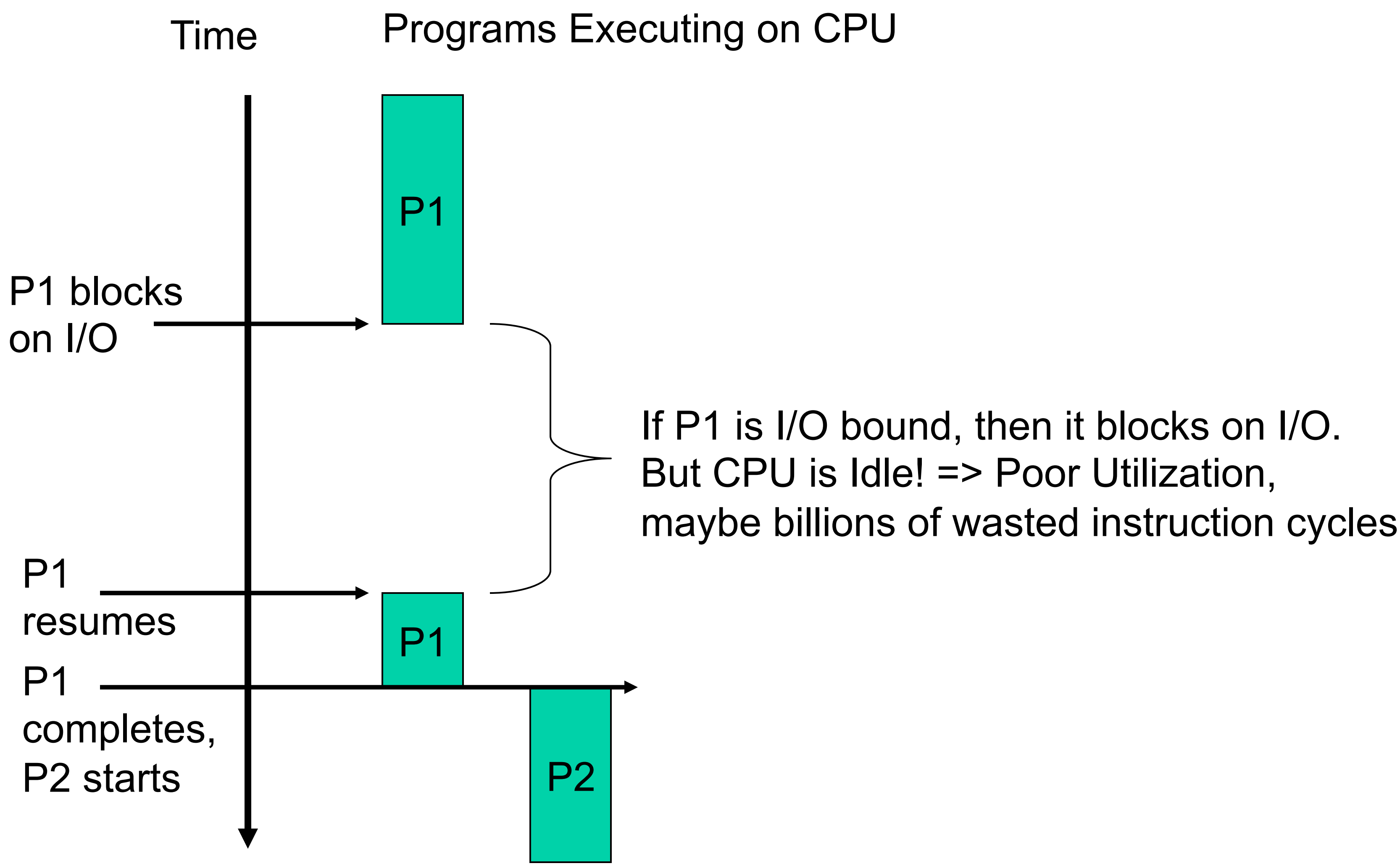
Multiprogramming: sequential execution



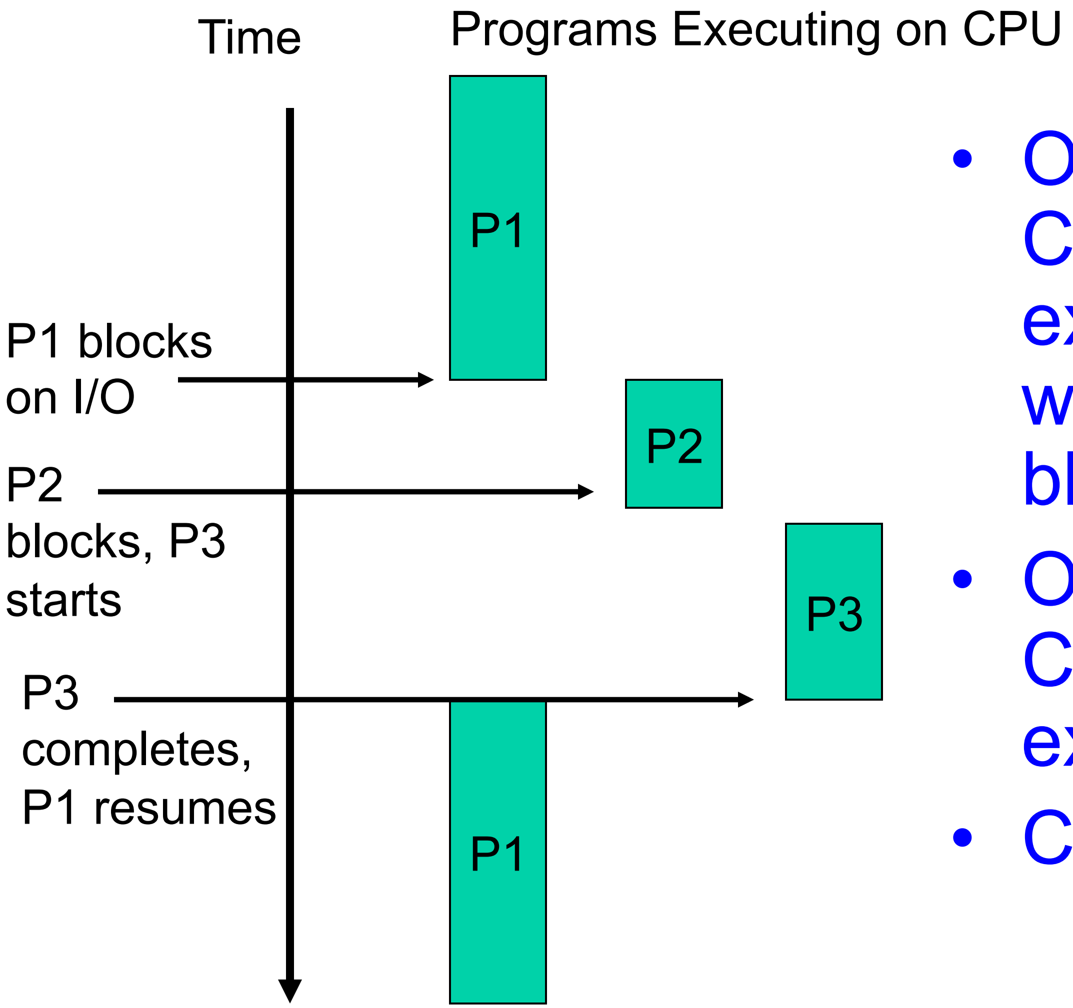
Let programs execute *sequentially*. When P1 is done, P2 takes over, and so on. In this case, P1, P2, and P3 are all *compute-bound*, i.e. executing continuously on the CPU until they' re done



Multiprogramming: Problem with sequential execution



Multiprogramming



- OS Scheduler switches CPU between multiple executing programs when a program is blocked, e.g. for I/O
- OS *time-multiplexes* CPU between executable programs
- Context switch

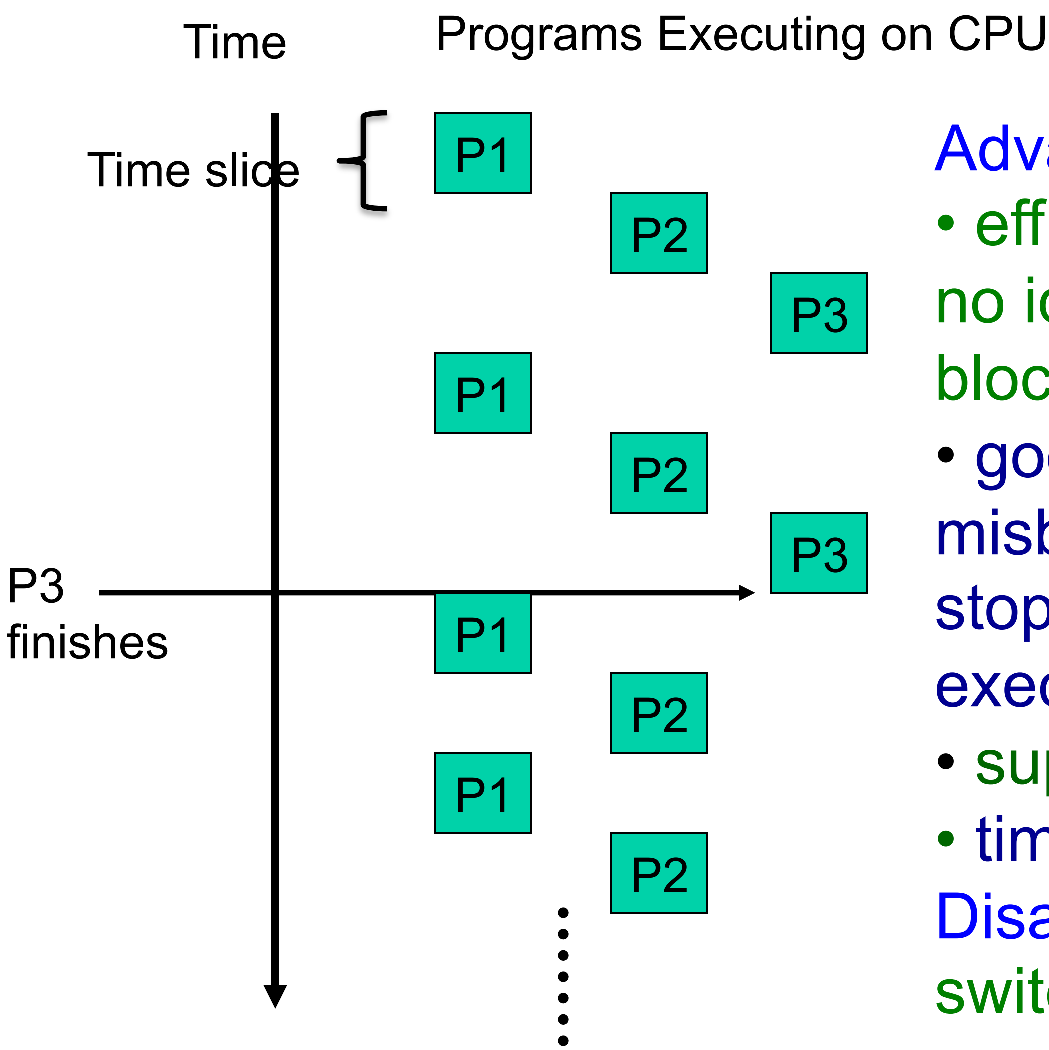
Context Switch

- Switching CPU from one running program to another is called a *context switch*
  - Save the current state (PC, IR, data registers, stack pointers, etc.) of the running application
  - Load the state of the new application
  - there is overhead due to this context switching
- No useful progress occurs for any applications while the OS is context-switching the CPU.

- Problem with pure multiprogramming: A CPU-bound program may delay the execution of other programs
  - A program with an infinite loop
  - A program to calculate the value of pi to the one-billionth decimal place

Multitasking

- CPU rapidly switches between programs



Advantages:

- efficient CPU usage, i.e. no idle time if one program blocks
- good isolation – a misbehaving program can't stop other programs from executing
- supports interactivity
- timesharing

Disadvantage: context switch overhead

Cooperative vs Preemptive Multitasking

- In cooperative multitasking, programs quickly and voluntarily yield CPU before they're done
  - Early OSs did this (Windows 3.1, Mac OS 9.*)
  - Poor fault isolation due to misbehaving programs
- In preemptive multitasking, OS forces programs to give up CPU
  - Fault isolation, interactivity, efficient CPU utilization
  - Time slice: time interval for which a program is allowed to run at any time
- All modern OSs are preemptively multitasked
  - Linux, BSD Unix, Windows NT/XP/Vista/7, Mac OS X 10.*

Preemptive Multitasking

- How does the OS force *rapid* switching?
  - Timer interrupt fires periodically
  - This suspends execution of the currently executing program and returns control to the OS scheduler
  - The scheduler decides the next program to execute and loads it, then passes control to it
- Context switch overhead: If you choose your default time slice too small, then you'll incur a lot of context-switching overhead as a % of your overall CPU utilization.

- mid 70's to 90's
  - LSI/VLSI; LANs
  - PCs and workstations
  - Process control and real-time systems
  - User friendly software
  - Distributed operating systems
  - MS DOS, 4.2 BSD Unix with TCP/IP, Mac OS with GUI, Linux, …

OS Design

- OS design is guided by two factors:
  - Technology: CPU, storage, communication, new I/O devices, …
  - Application needs

Operating System: Current Trends

- Hardware has evolved quickly - OS must adjust
  - Moore's Law roughly applies to CPU speed and/or memory size: doubles every 18 months => exponential!
    - Enables complex modern operating systems like Linux, Windows, UNIX, OS X

Storage Evolution:

Punch card
stores code
1950s-80s

Tape Drive
1950s-80s

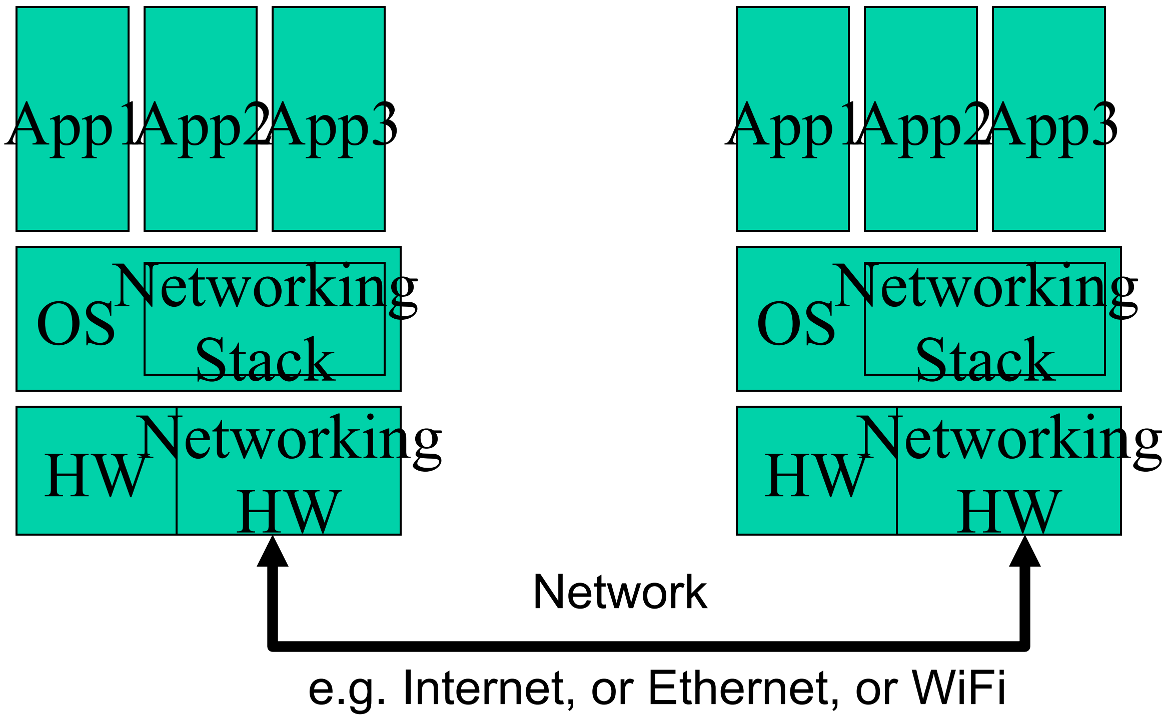
Disk Drive
1960s-2000s

??

Flash Drive
already at 16 GB @
$30 => OS disk
scheduling obsolete??

But Moore's Law doesn't apply to disk access speed or to battery life

Distributed operating systems



- Examples:
  - App1 is a distributed client server app, e.g. App1 on left is Web browser, App1 on right is Web server

- Networked File System
- OS adds TCP/IP Network Stack
- Device driver support for Networking cards

Operating System: Current Trends

- Diversification of OS's to many different target environments
  - Energy-efficient cell phone OSs - scaling down
    - iPhone's iOS, Google's Android, …
  - Multi-processor OSs - scaling up
    - Adapting Linux and Windows to multiple cores. Massively parallel supercomputers.
  - Real-Time OS for Embedded and Multimedia Systems
    - VXWorks, robotic OSs, …

Operating System: Current Trends

- Virtualization – Virtual Machines (VMs)
  - Running a Windows VM inside a Linux OS, and vice versa.
  - More layers of abstraction
- Cloud computing rents VMs on racks of PCs at a massive scale

Google Data Center in The Dalles, Oregon

Size of football field