

# Chapter 1

## Computer Abstractions and Technology

# The Computer Revolution

- Progress in computer technology
  - Underpinned by domain-specific accelerators
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

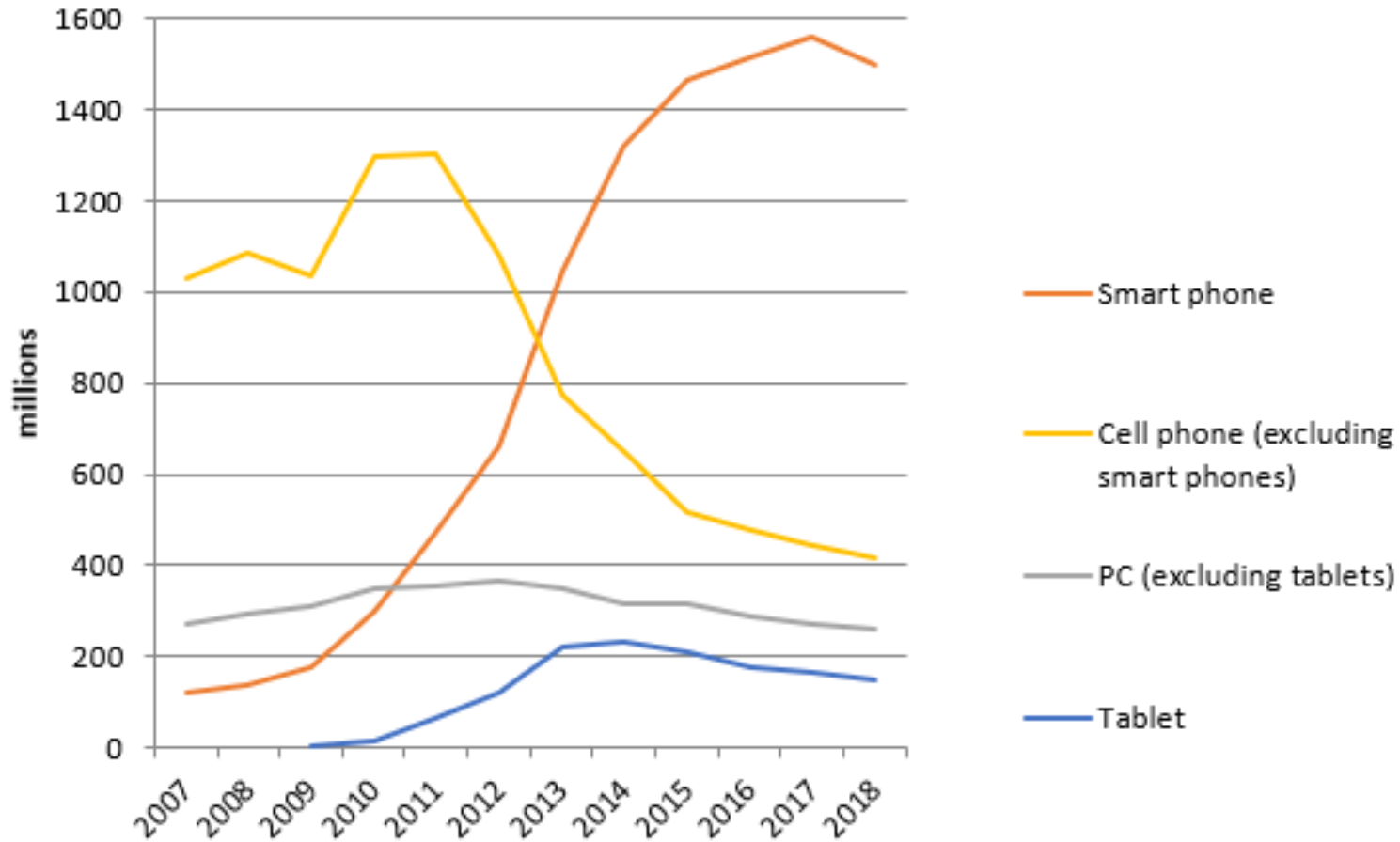
# Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

# Classes of Computers

- Supercomputers
  - Type of server
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

# The PostPC Era



# The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

# What You Will Learn

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

# Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed



# Seven Great Ideas

- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



ABSTRACTION



COMMON CASE FAST



PARALLELISM



PIPELINING



PREDICTION

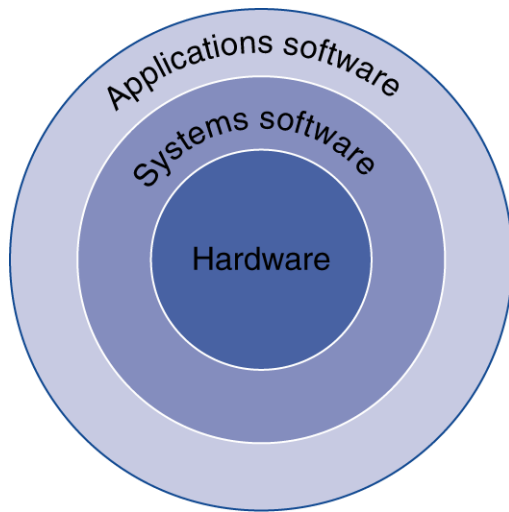


HIERARCHY



DEPENDABILITY

# Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

# Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly  
language  
program  
(for MIPS)

```
swap:
  muli $2, $5, 4
  add $2, $4, $2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```

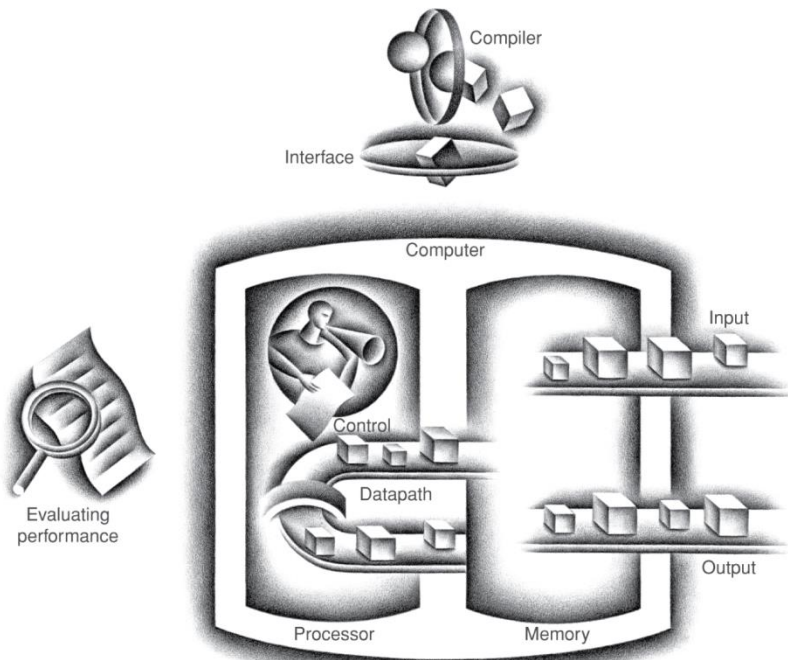
Assembler

Binary machine  
language  
program  
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

# Components of a Computer

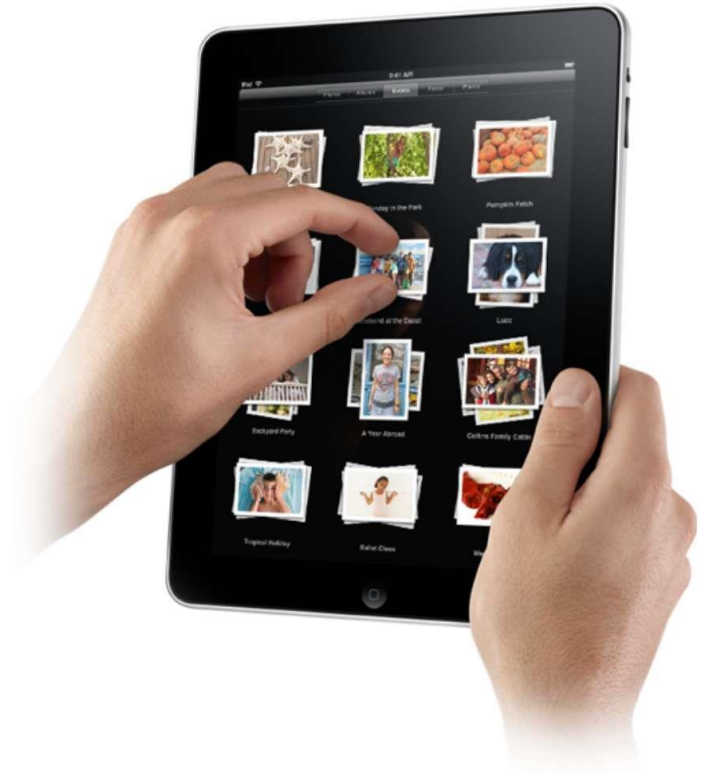
## The BIG Picture



- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

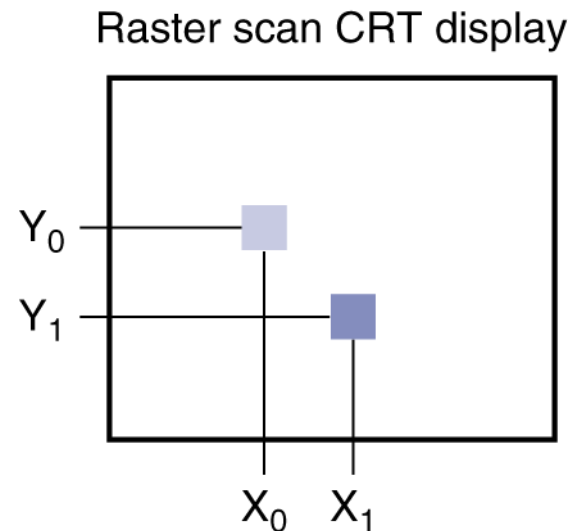
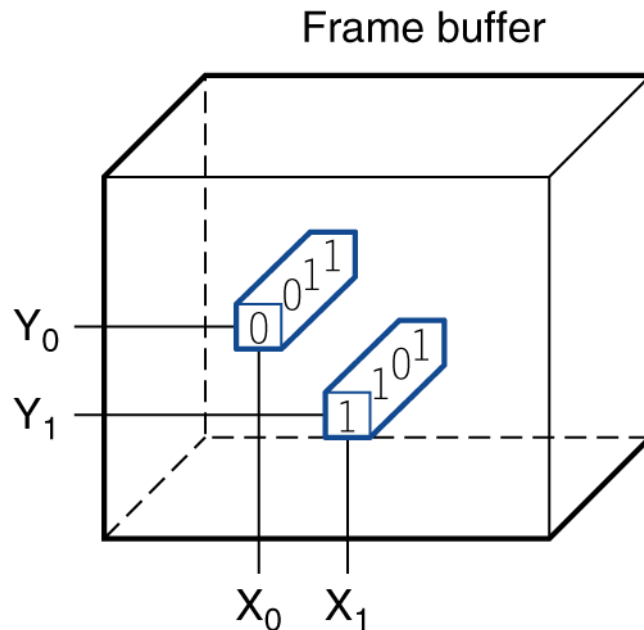
# Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

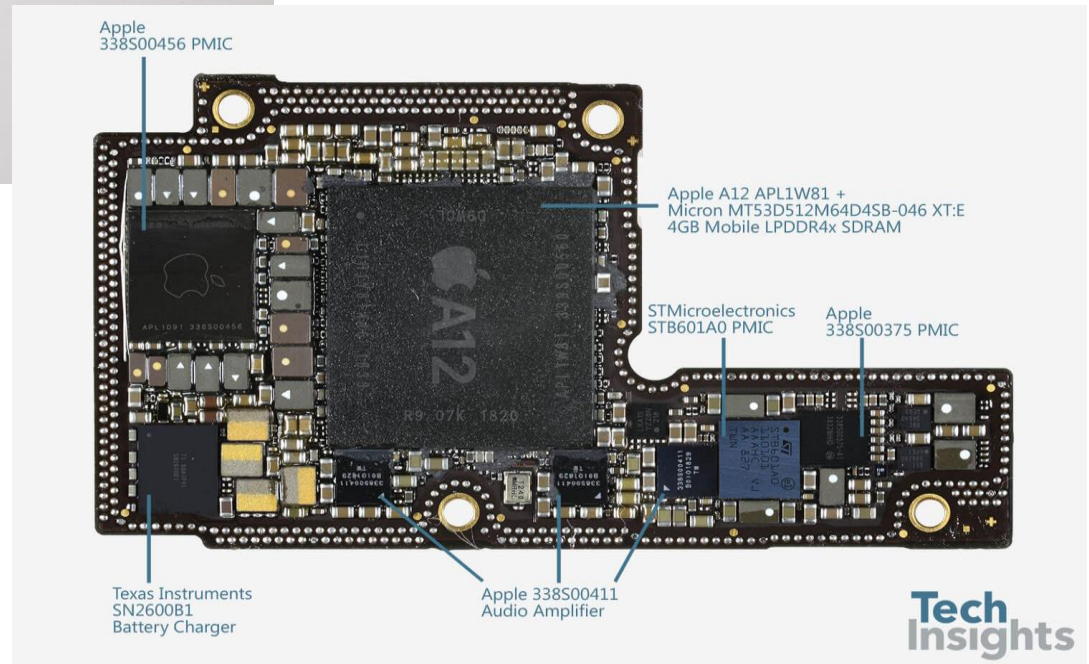
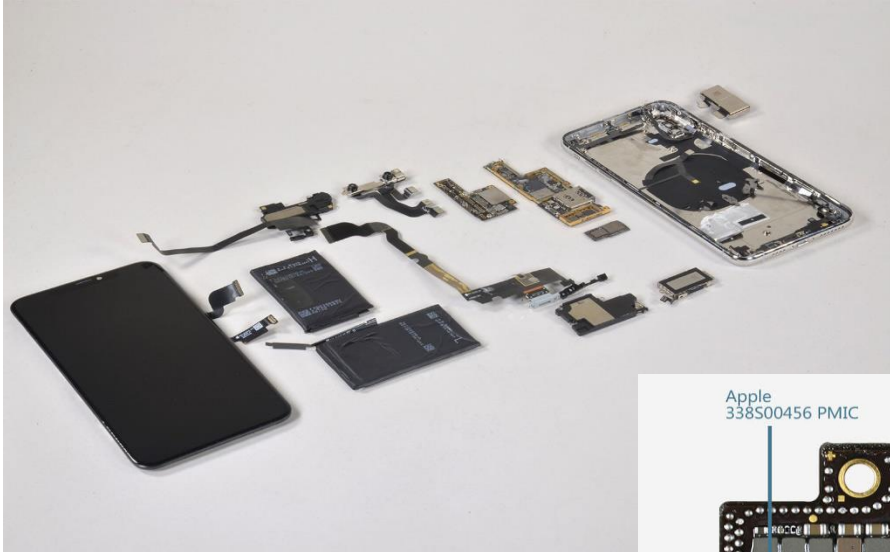


# Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



# Opening the Box



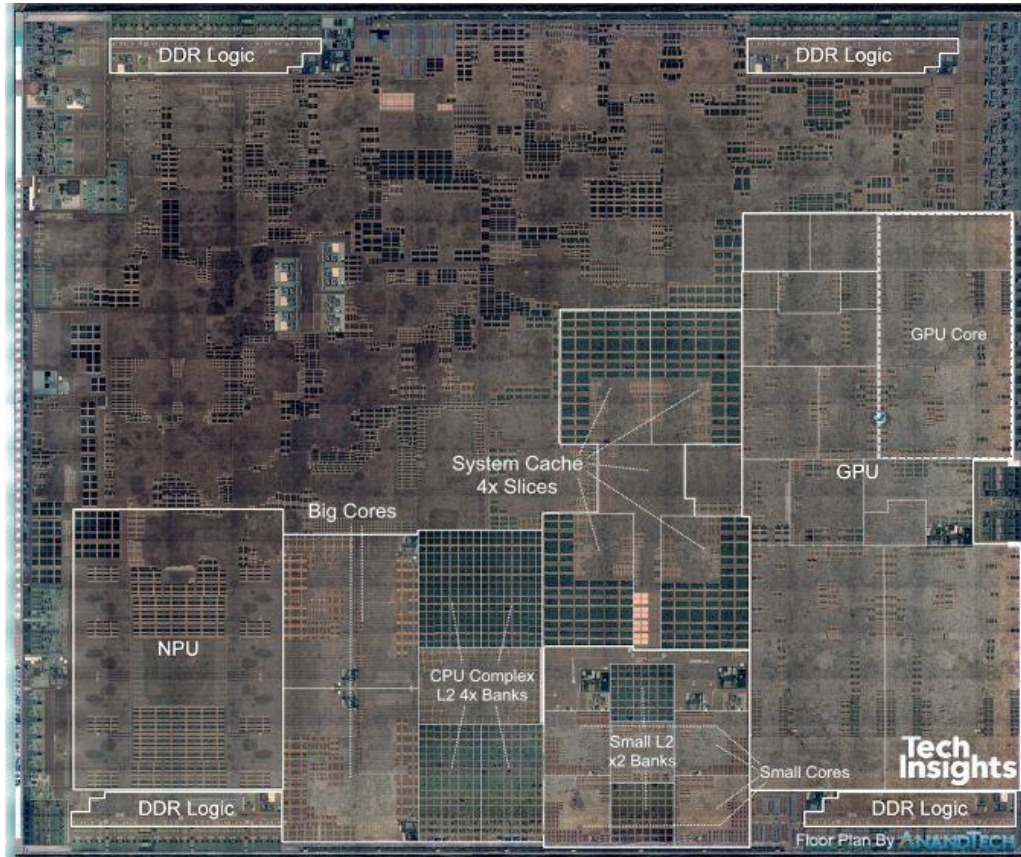
# Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data



# Inside the Processor

- A12 processor



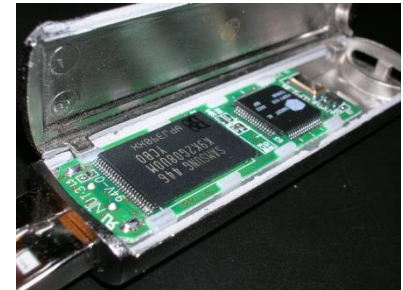
# Abstractions

## The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

# A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



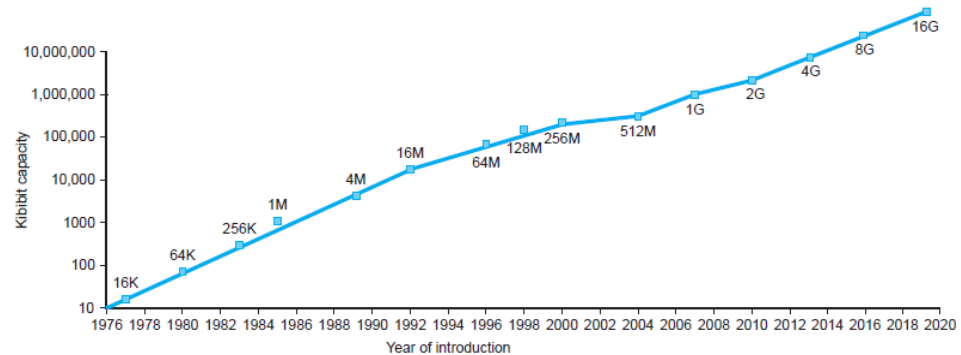
# Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



# Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

| Year | Technology                 | Relative performance/cost |
|------|----------------------------|---------------------------|
| 1951 | Vacuum tube                | 1                         |
| 1965 | Transistor                 | 35                        |
| 1975 | Integrated circuit (IC)    | 900                       |
| 1995 | Very large scale IC (VLSI) | 2,400,000                 |
| 2013 | Ultra large scale IC       | 250,000,000,000           |

# Semiconductor Technology

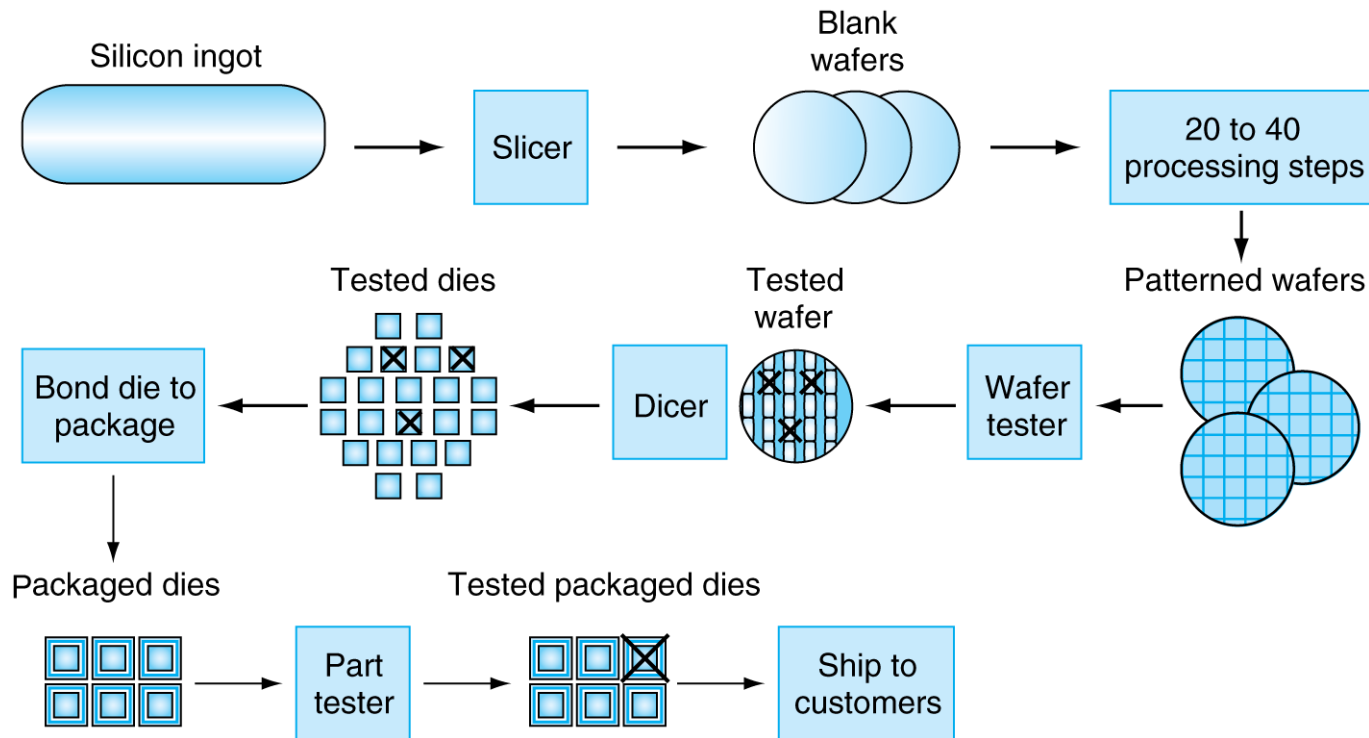
- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch



# Pop Quiz

- In 1980 64 KiB of RAM cost around \$400-\$500. By year 1994, you could get a 4 MiB RAM module for \$130 - \$160. What would be your best estimate for a price of 1 GiB of RAM in 2005?
  - ⌘ A: \$10
  - ⌘ B: \$40
  - ⌘ C: \$200
  - ⌘ D: \$600

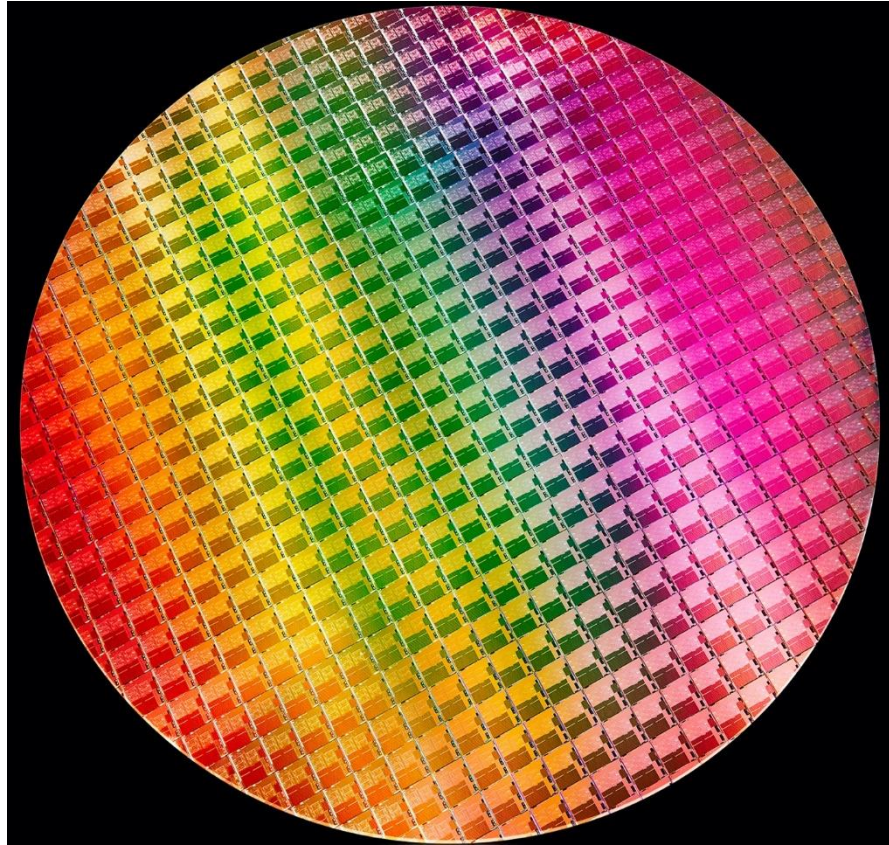
# Manufacturing ICs



- **Yield:** proportion of working dies per wafer



# Intel® Core 10<sup>th</sup> Gen



- 300mm wafer, 506 chips, 10nm technology
- Each chip is 11.4 x 10.7 mm

# Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

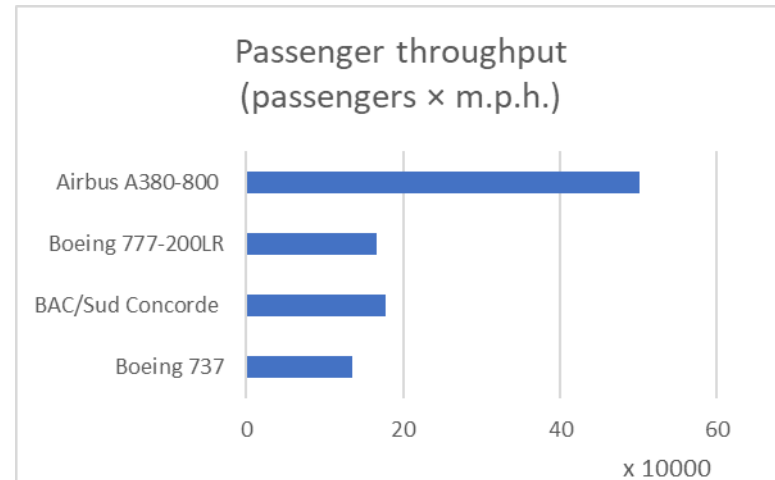
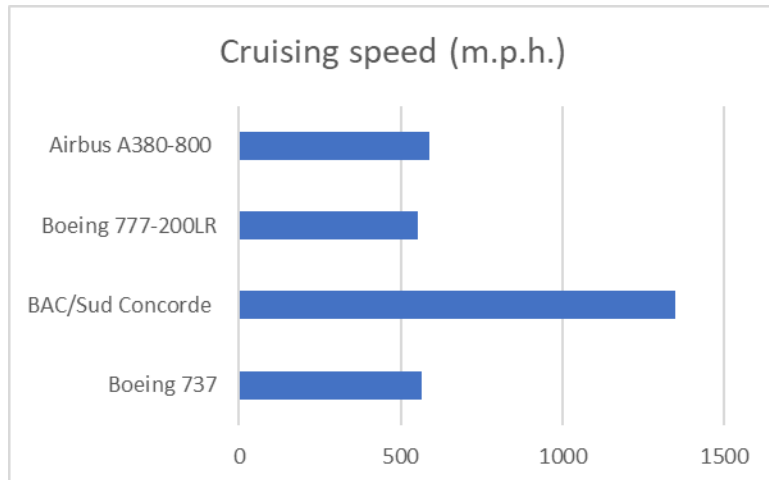
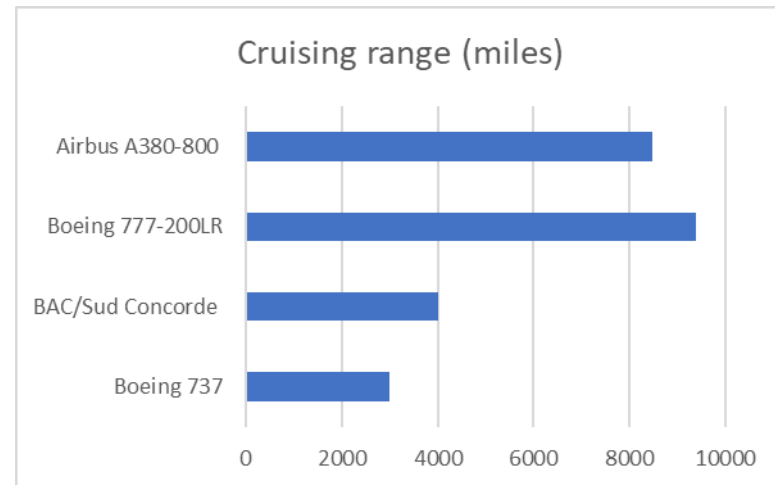
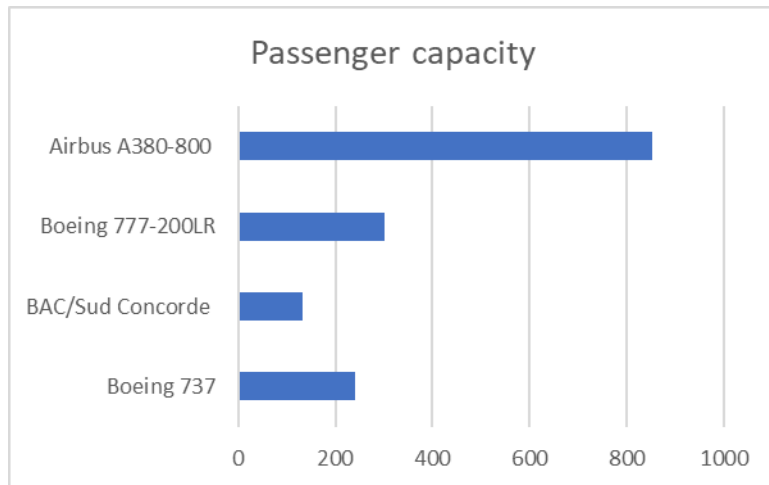
$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

# Defining Performance

- Which airplane has the best performance?



# Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now...

# Relative Performance

- Define Performance =  $1/\text{Execution Time}$
- “X is  $n$  time faster than Y”

$$\begin{aligned} & \text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n \end{aligned}$$

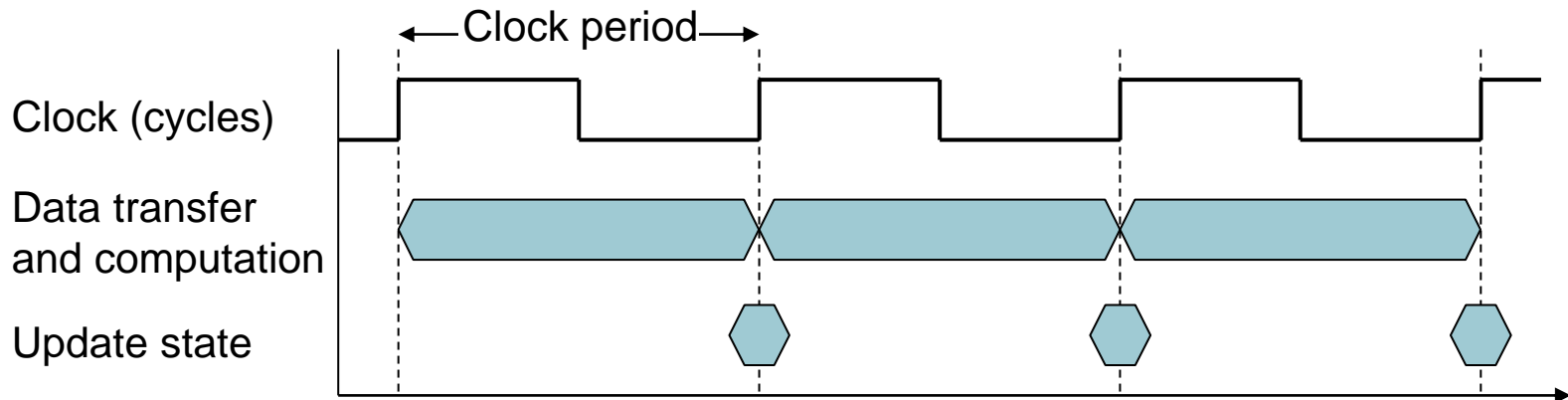
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - So A is 1.5 times faster than B

# Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

# CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock period: duration of a clock cycle
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate): cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

# CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count



# CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes  $1.2 \times$  clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10s \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}$$

# Instruction Count and CPI

Clock Cycles = Instruction Count  $\times$  Cycles per Instruction

CPU Time = Instruction Count  $\times$  CPI  $\times$  Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# Pop Quiz

- The choice of which of the following hardware and software components affects the CPI of a program?
  - ⌘ A: Algorithm
  - ⌘ B: Programming language
  - ⌘ C: Compiler
  - ⌘ D: Instruction set architecture (ISA)

# CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \leftarrow \text{A is faster...}$$

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2 \leftarrow$$

...by this much

# CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^n \left( \text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

| Class            | A | B | C |
|------------------|---|---|---|
| CPI for class    | 1 | 2 | 3 |
| IC in sequence 1 | 2 | 1 | 2 |
| IC in sequence 2 | 4 | 1 | 1 |

- Sequence 1: IC = 5

- Clock Cycles  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
- Avg. CPI =  $10/5 = 2.0$

- Sequence 2: IC = 6

- Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
- Avg. CPI =  $9/6 = 1.5$

# Performance Summary

## The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

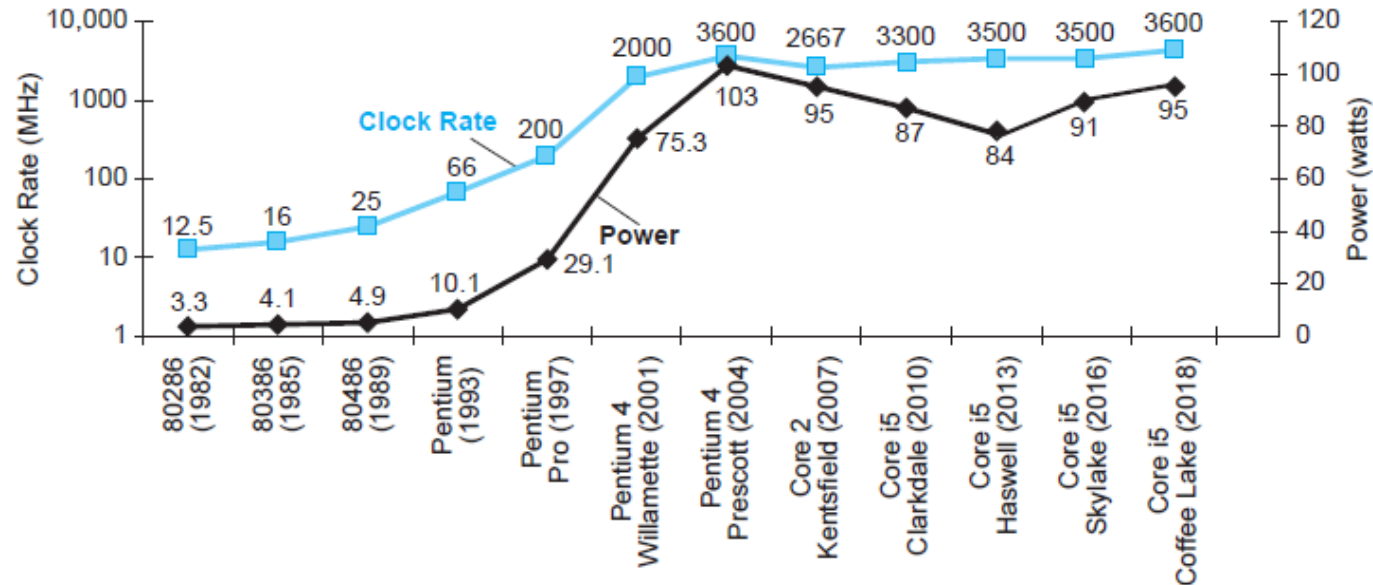
- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI,  $T_c$

# Pop Quiz

- A Java program runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.8 as many instructions as the old compiler. Unfortunately, it increases the CPI by 20%. How fast can we expect the program to run using this new compiler?
  - ⌘ A: 10
  - ⌘ B: 14.4
  - ⌘ C: 22.5
  - ⌘ D: None of the above



# Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

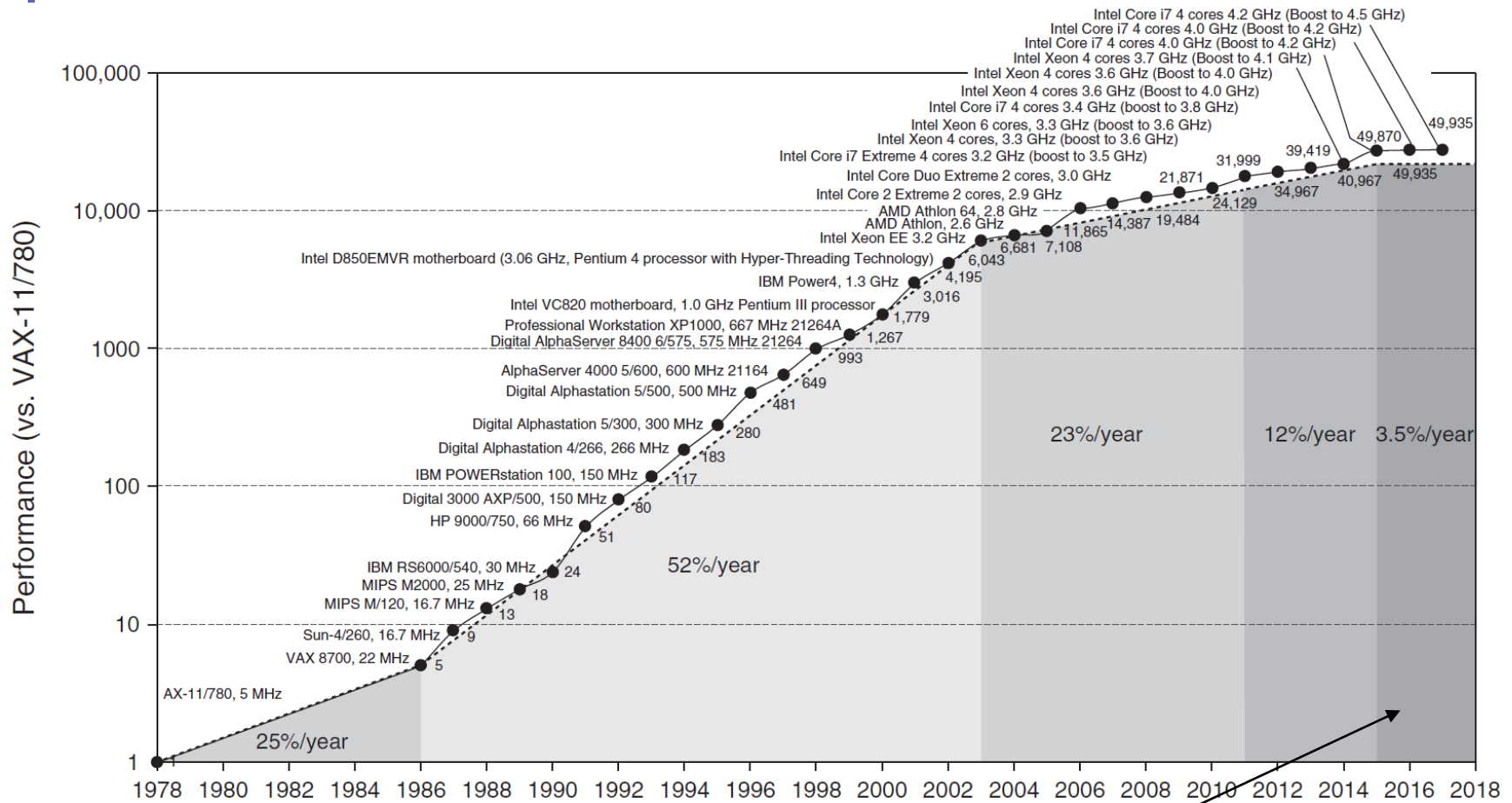
# Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

# Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

# Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}_i}$$

# SPECspeed 2017 Integer benchmarks on a 1.8 GHz Intel Xeon E5-2650L

| <i>Description</i>                                             | <i>Name</i> | <i>Instruction Count x 10<sup>9</sup></i> | <i>CPI</i> | <i>Clock cycle time (seconds x 10<sup>-9</sup>)</i> | <i>Execution Time (seconds)</i> | <i>Reference Time (seconds)</i> | <i>SPECratio</i> |
|----------------------------------------------------------------|-------------|-------------------------------------------|------------|-----------------------------------------------------|---------------------------------|---------------------------------|------------------|
| Perl interpreter                                               | perlbench   | 2684                                      | 0.42       | 0.556                                               | 627                             | 1774                            | 2.83             |
| GNU C compiler                                                 | gcc         | 2322                                      | 0.67       | 0.556                                               | 863                             | 3976                            | 4.61             |
| Route planning                                                 | mcf         | 1786                                      | 1.22       | 0.556                                               | 1215                            | 4721                            | 3.89             |
| Discrete Event simulation - computer network                   | omnetpp     | 1107                                      | 0.82       | 0.556                                               | 507                             | 1630                            | 3.21             |
| XML to HTML conversion via XSLT                                | xalancbmk   | 1314                                      | 0.75       | 0.556                                               | 549                             | 1417                            | 2.58             |
| Video compression                                              | x264        | 4488                                      | 0.32       | 0.556                                               | 813                             | 1763                            | 2.17             |
| Artificial Intelligence: alpha-beta tree search (Chess)        | deepsjeng   | 2216                                      | 0.57       | 0.556                                               | 698                             | 1432                            | 2.05             |
| Artificial Intelligence: Monte Carlo tree search (Go)          | leela       | 2236                                      | 0.79       | 0.556                                               | 987                             | 1703                            | 1.73             |
| Artificial Intelligence: recursive solution generator (Sudoku) | exchange2   | 6683                                      | 0.46       | 0.556                                               | 1718                            | 2939                            | 1.71             |
| General data compression                                       | xz          | 8533                                      | 1.32       | 0.556                                               | 6290                            | 6182                            | 0.98             |
| Geometric mean                                                 |             |                                           |            |                                                     |                                 |                                 | 2.36             |

# SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj\_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left( \sum_{i=0}^{10} \text{ssj\_ops}_i \right) / \left( \sum_{i=0}^{10} \text{power}_i \right)$$

# SPECpower\_ssj2008 for Xeon E5-2650L

| Target Load %                                    | Performance (ssj_ops) | Average Power (watts) |
|--------------------------------------------------|-----------------------|-----------------------|
| 100%                                             | 4,864,136             | 347                   |
| 90%                                              | 4,389,196             | 312                   |
| 80%                                              | 3,905,724             | 278                   |
| 70%                                              | 3,418,737             | 241                   |
| 60%                                              | 2,925,811             | 212                   |
| 50%                                              | 2,439,017             | 183                   |
| 40%                                              | 1,951,394             | 160                   |
| 30%                                              | 1,461,411             | 141                   |
| 20%                                              | 974,045               | 128                   |
| 10%                                              | 485,973               | 115                   |
| 0%                                               | 0                     | 48                    |
| Overall Sum                                      | 26,815,444            | 2,165                 |
| $\Sigma \text{ssj\_ops} / \Sigma \text{power} =$ |                       | 12,385                |



# Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20 \quad \quad \quad \blacksquare \text{ Can't be done!}$$

- Corollary: make the common case fast

# Pop Quiz

- You run your sequential code and observe that it spends 60 seconds multiplying and 30 seconds on all other tasks (I/O, memory management, etc.) You change it so that it now uses four cores to perform multiplications concurrently. What kind of total runtime would you expect from the parallel version of your code?
  - ⌘ A: 22.5
  - ⌘ B: 30
  - ⌘ C: 45
  - ⌘ D: 60

# Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

# Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

- CPI varies between programs on a given CPU

# Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance