# Chapter 1

# Computer Abstractions and Technology

# The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- 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
  - 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
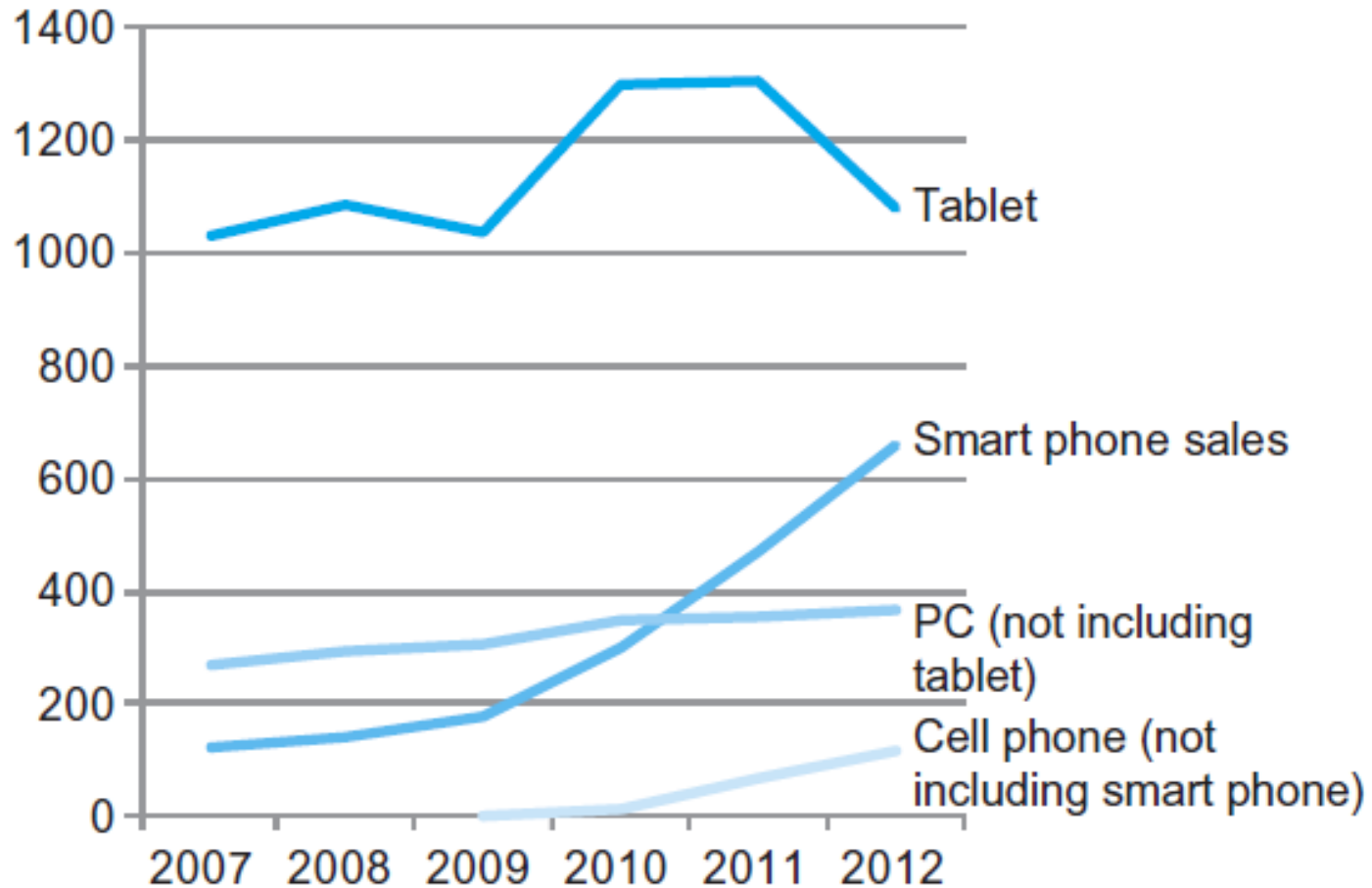
# The PostPC Era

- 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

# Eight Great Ideas

- Design for **Moore's Law**

- 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

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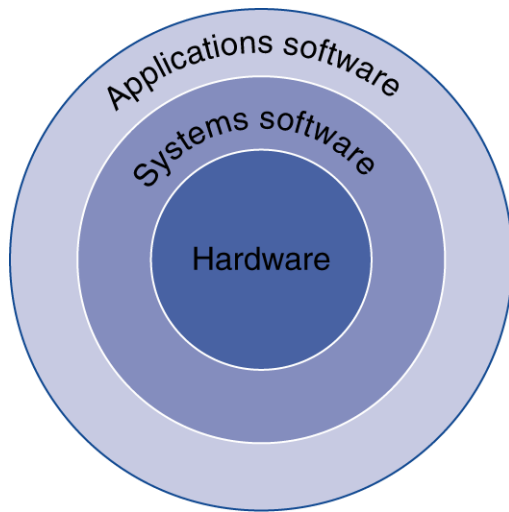
- ## Application software
  - ### Written in high-level language
- ## System software
  - ### Compiler: translates HLL code to machine code
  - ### Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- ## Hardware
  -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

Applications software
Systems software
Hardware

# 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)
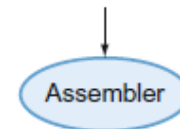  - Encoded instructions and data

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 ARMv8)

```
swap:
    LSL   X10, X1,3
    ADD   X10, X0,X10
    LDUR  X9, [X10,0]
    LDUR  X11,[X10,8]
    STUR  X11,[X10,0]
    STUR  X9, [X10,8]
    BR    X10
```

Assembler

Binary machine language program (for ARMv8)

```
00000000101000100000000100011000
00000000100000100010000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

# Components of a Computer

**The BIG Picture**
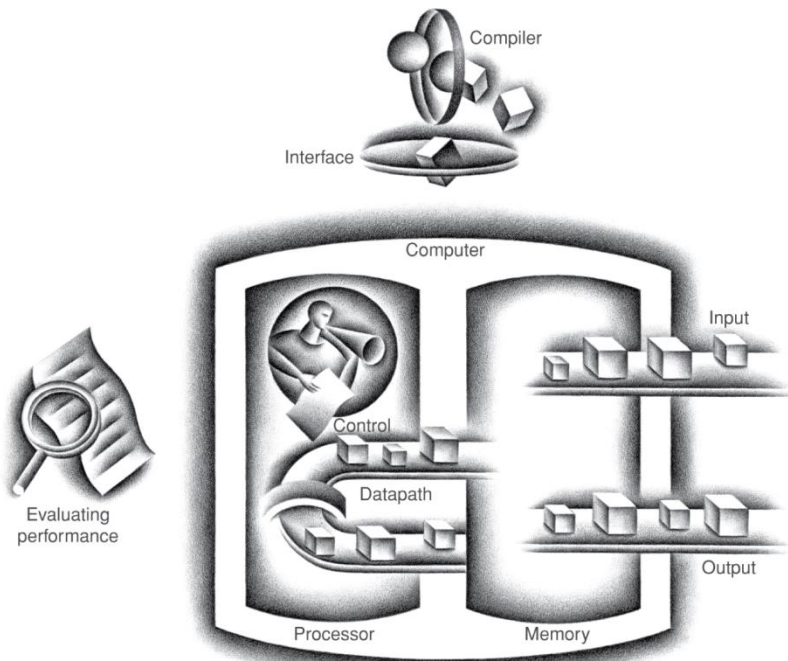


- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

# Touchscreen

- PostPC device

- Supersedes keyboard and mouse

- Resistive and Capacitive types

  - Most tablets, smart phones use capacitive

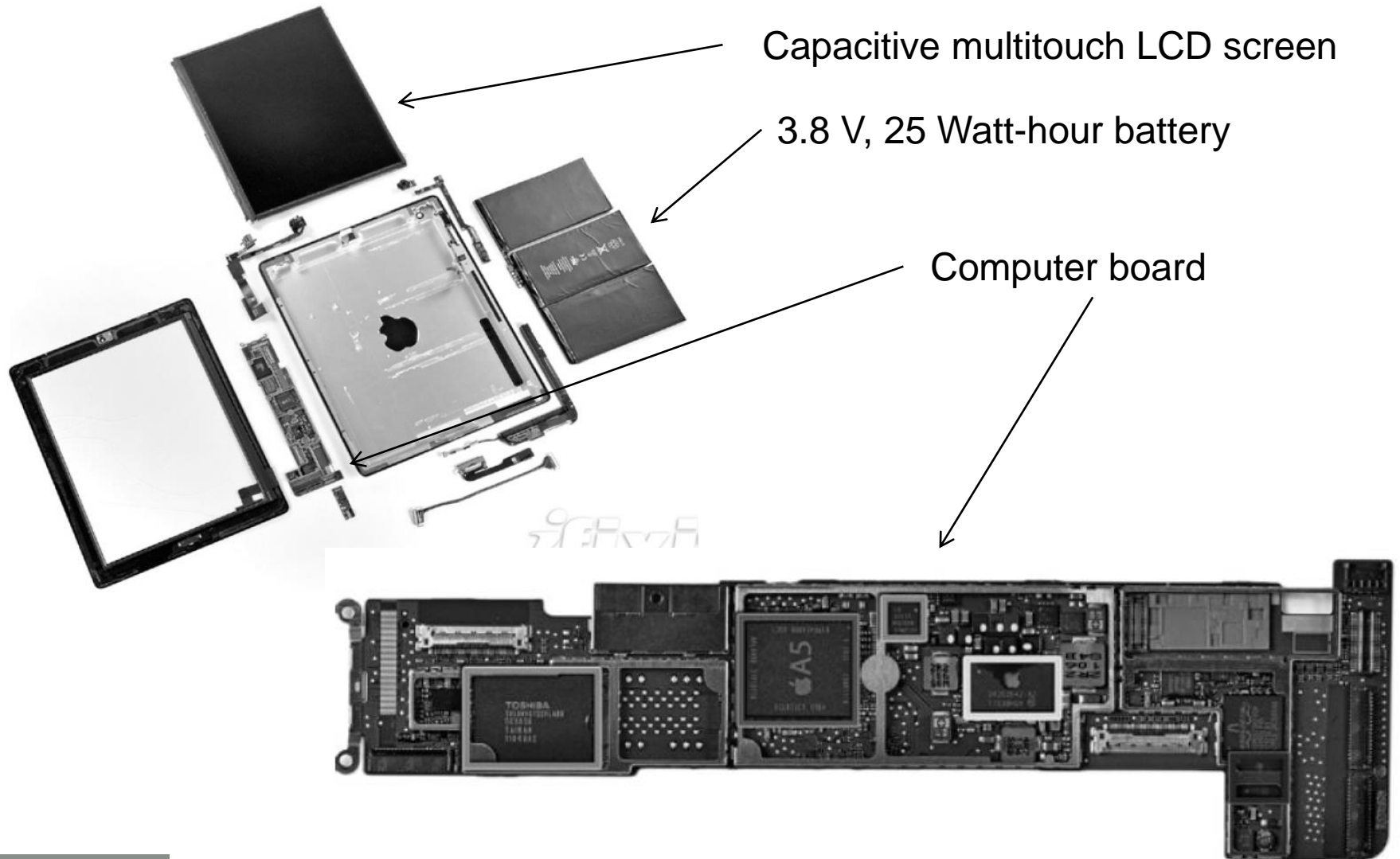  - Capacitive allows multiple touches simultaneously

# Through the Looking Glass

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

# Opening the Box

Capacitive multitouch LCD screen

3.8 V, 25 Watt-hour battery

Computer board
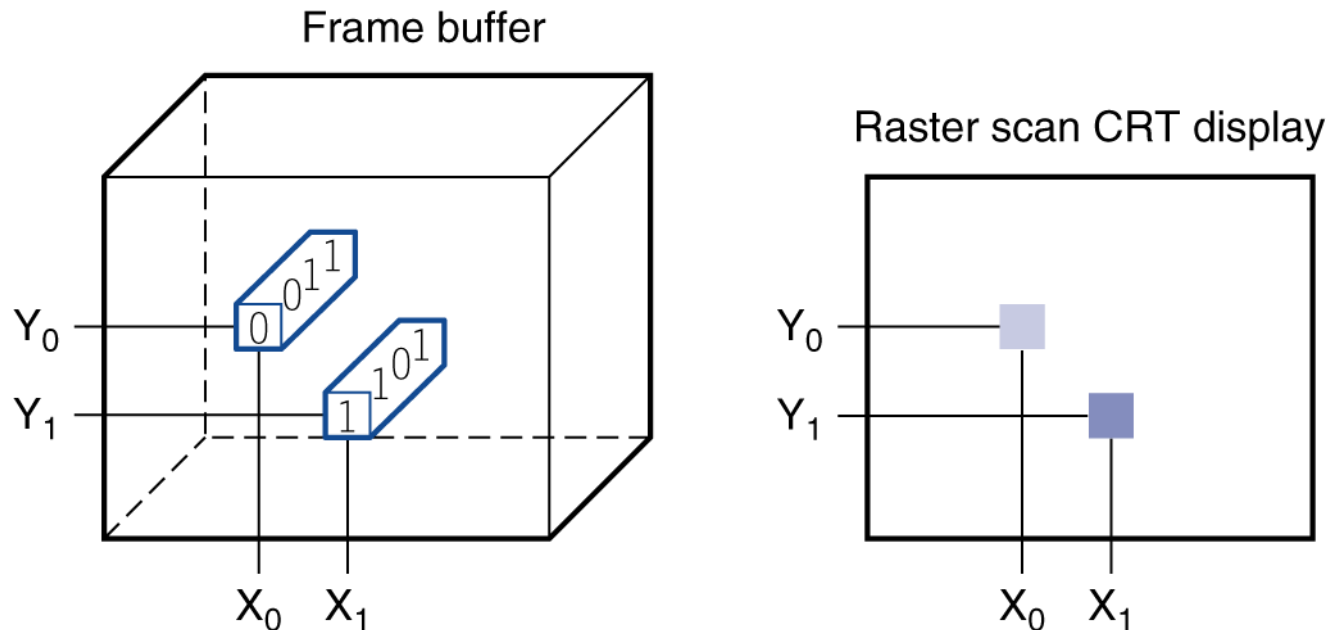
# Inside the Processor (CPU)

- Datapath: performs operations on data

- Control: sequences datapath, memory, ...

- Cache memory

  - Small fast SRAM memory for immediate access to data

# Inside the Processor

- Apple A5

# Abstractions

- 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
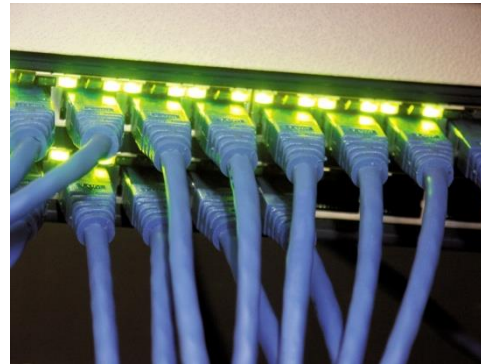
# A Safe Place for Data

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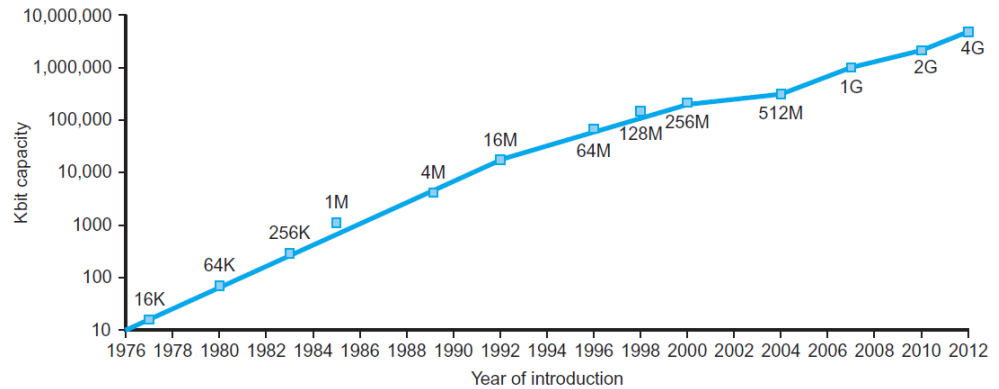    - Optical disk (CDROM, DVD)
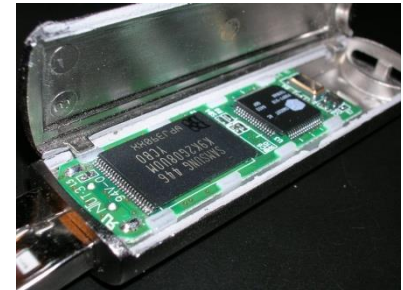
# Networks

- 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

# Manufacturing ICs
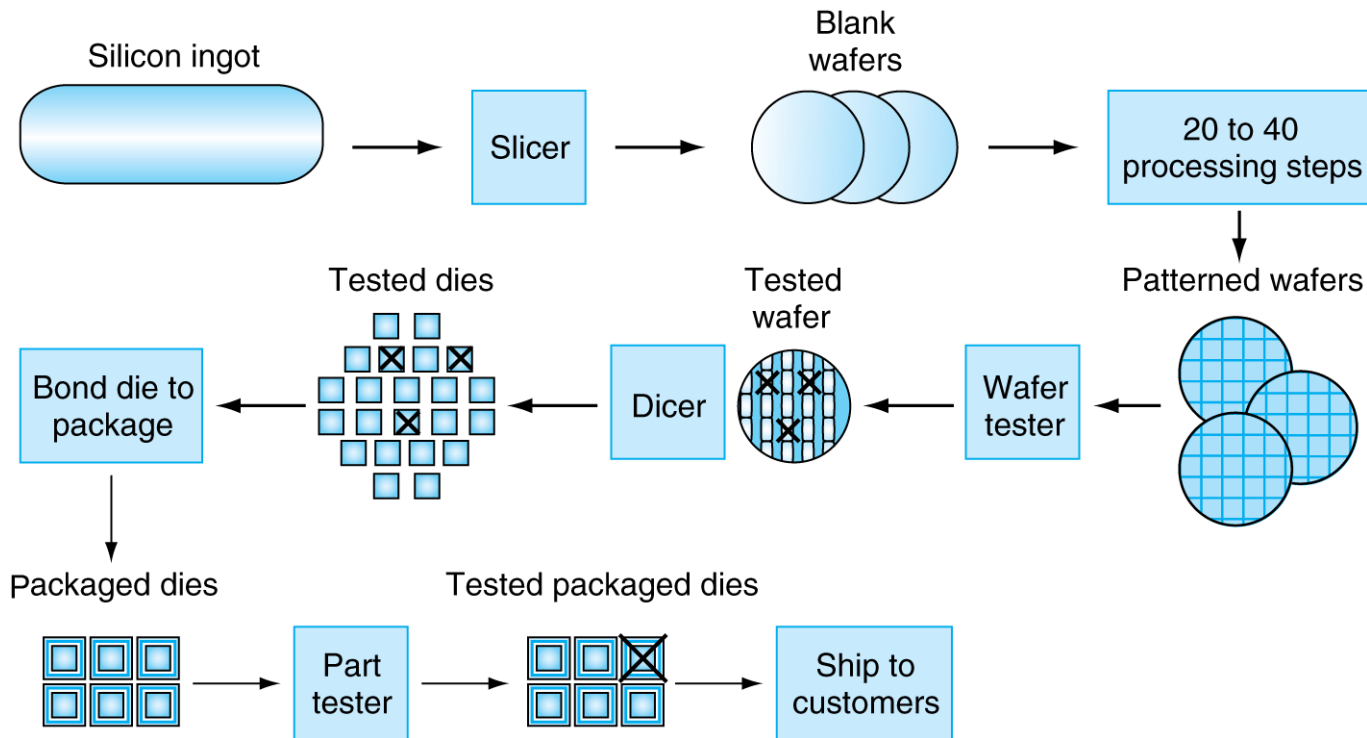


- Yield: proportion of working dies per wafer

# Intel Core i7 Wafer
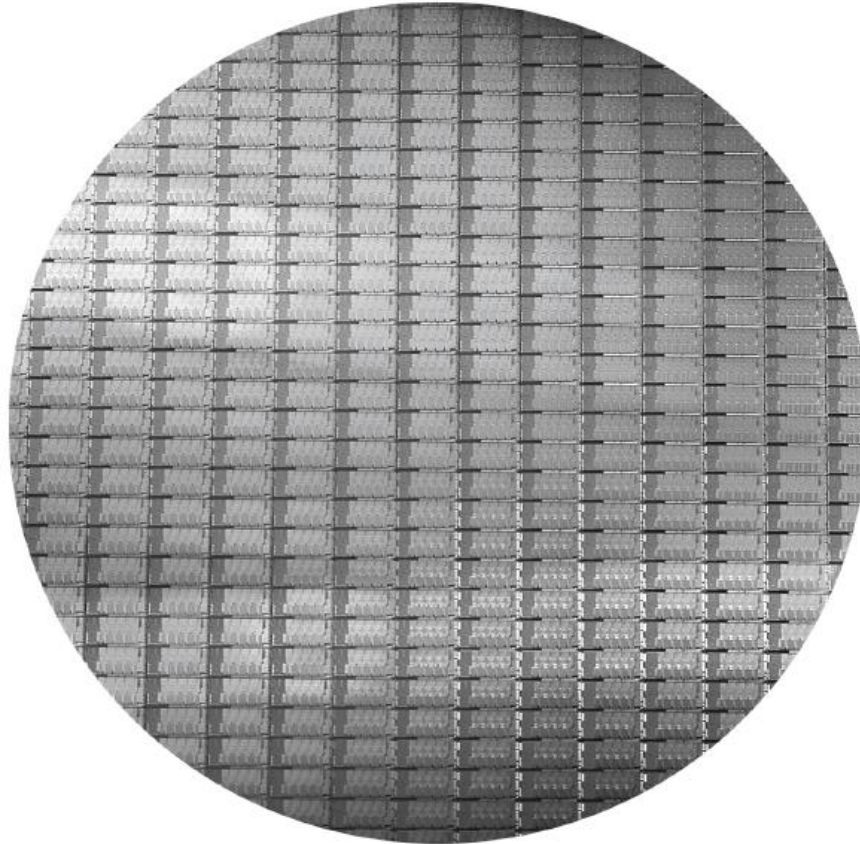


- 300mm wafer, 280 chips, 32nm technology (the smallest feature size on the die 32nm.)
- Each chip is 20.7 x 10.5 mm

# SRAM Cell Size Scaling



65 nm, 0.570 um²

45 nm, 0.346 um²

32 nm, 0.171 um²

0.5x every 2 years

Transistor density continues to double every 2 years

# Integrated Circuit Cost

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

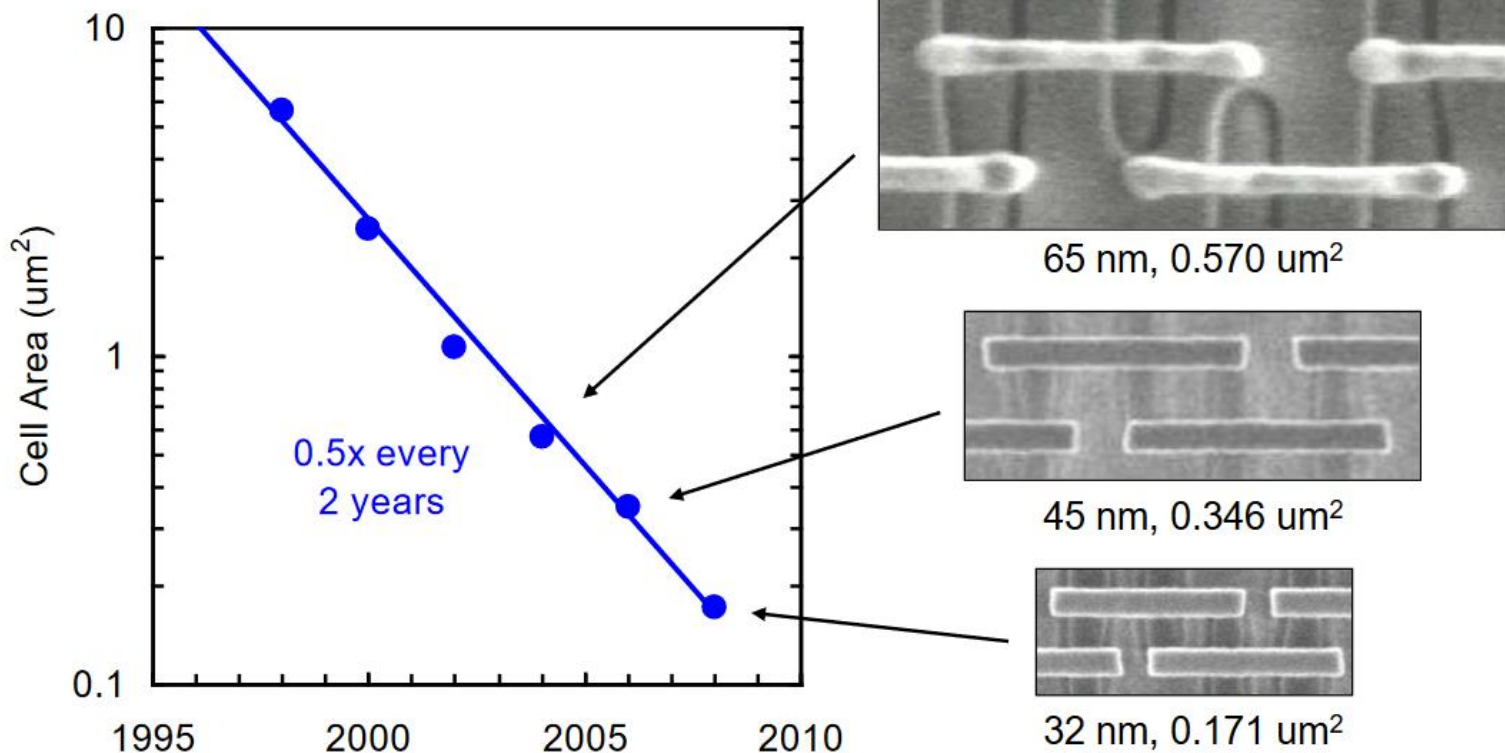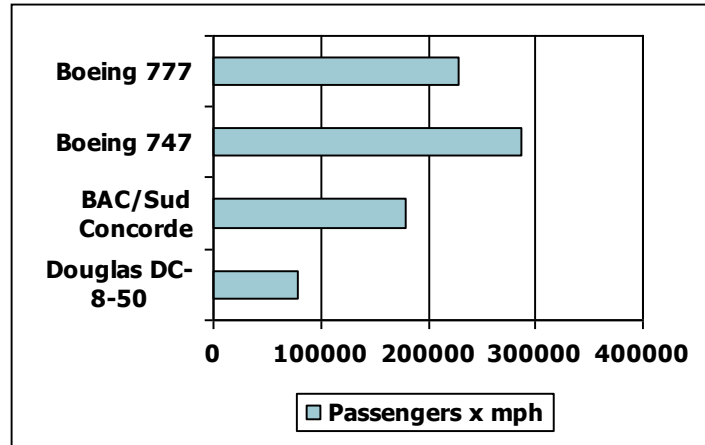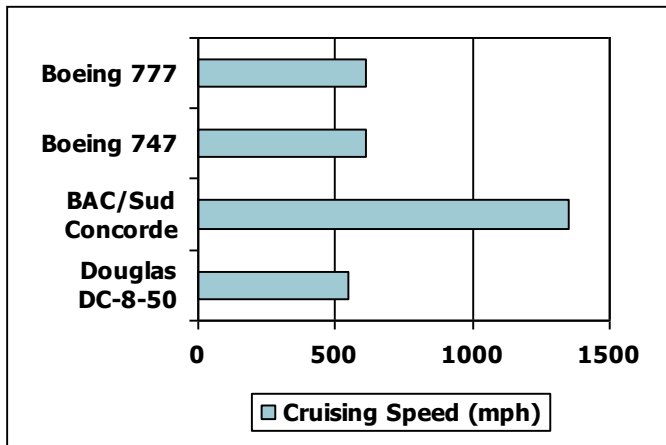$$\text{Dies per wafer} \approx \text{Wafer area/Die area}$$

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

- 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?



Bar chart: Passenger Capacity — Boeing 777 (~375), Boeing 747 (~470), BAC/Sud Concorde (~130), Douglas DC-8-50 (~145). X-axis 0–500.

Bar chart: Cruising Range (miles) — Boeing 777 (~4600), Boeing 747 (~4100), BAC/Sud Concorde (~4000), Douglas DC-8-50 (~8700). X-axis 0–10000.

Bar chart: Cruising Speed (mph) — Boeing 777 (~610), Boeing 747 (~610), BAC/Sud Concorde (~1350), Douglas DC-8-50 (~545). X-axis 0–1500.

Bar chart: Passengers x mph — Boeing 777 (~230000), Boeing 747 (~290000), BAC/Sud Concorde (~180000), Douglas DC-8-50 (~80000). X-axis 0–400000.

# 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/Execution Time

- "X is $n$ time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 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
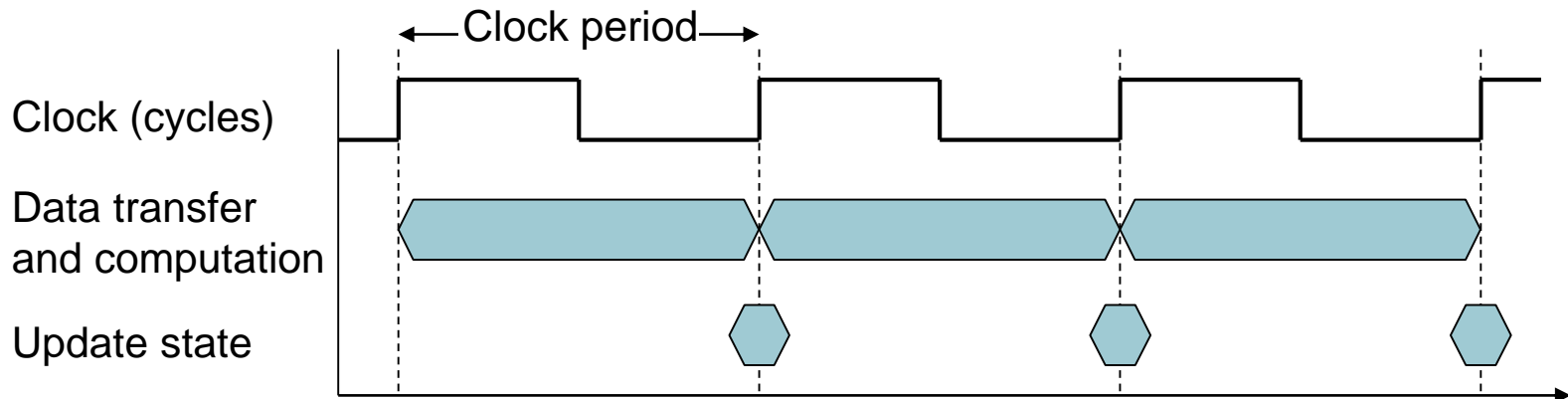


■ Clock period: duration of a clock cycle
  ■ e.g., 250ps = 0.25ns = $250 \times 10^{-12}$s
■ Clock frequency (rate): cycles per second
  ■ e.g., 4.0GHz = 4000MHz = $4.0 \times 10^9$Hz

# CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- 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 × clock cycles
- How fast must Computer B clock be? (4GHz)

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

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10s \times 2\text{GHz} = 20 \times 10^9$$

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

# Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

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

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

- 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

# 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?

$$CPU\,Time_A = Instruction\,Count \times CPI_A \times Cycle\,Time_A$$

$$= I \times 2.0 \times 250ps = I \times 500ps \quad \leftarrow \boxed{A\ is\ faster\ldots}$$

$$CPU\,Time_B = Instruction\,Count \times CPI_B \times Cycle\,Time_B$$

$$= I \times 1.2 \times 500ps = I \times 600ps$$

$$\frac{CPU\,Time_B}{CPU\,Time_A} = \frac{I \times 600ps}{I \times 500ps} = 1.2 \quad \leftarrow \boxed{\ldots by\ this\ much}$$

# 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
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

# Performance Summary

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

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

# Understanding Program Performance

| Hardware or software component | Affects what? | How? |
|---|---|---|
| Algorithm | Instruction count, possibly CPI | The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI. |
| Programming language | Instruction count, CPI | The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions. |
| Compiler | Instruction count, CPI | The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in varied ways. |
| Instruction set architecture | Instruction count, clock rate, CPI | The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor. |

# Power Trends

- Figure 1.16 shows the increase in clock rate and power of eight generations of Intel microprocessors over 30 years.

- Both clock rate and power increased rapidly for decades and then flattened off recently.

- The reason they grew together is that they are correlated, and

- the reason for their recent slowing is that we have run into the practical power limit for cooling commodity microprocessors.
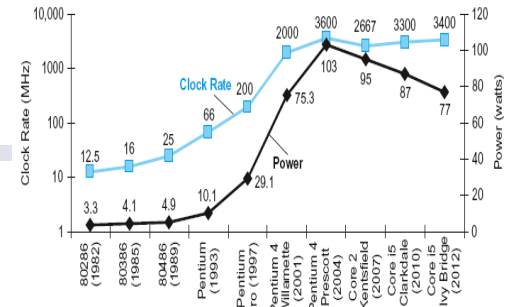
# Power Trends
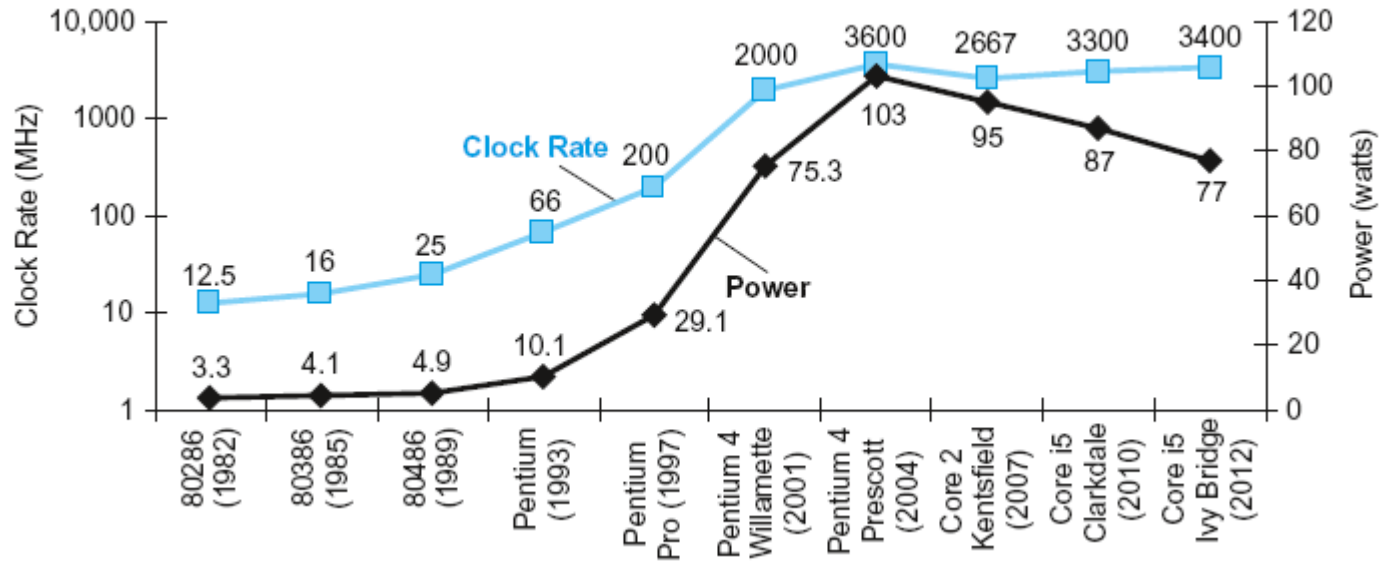
- In CMOS IC technology

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

×30

5V → 1V

×1000

# Power Trends

- For CMOS, the primary source of energy consumption is so-called dynamic energy—that is, energy that is consumed when transistors switch states from 0 to 1 and vice versa. The dynamic energy depends on the capacitive loading of each transistor and the voltage applied.
- The capacitive load per transistor is a function of both the number of transistors connected to an output (called the fanout) and the technology.

- With regard to Figure 1.16, how could clock rates grow by a factor of 1000 while power increased by only a factor of 30? Energy and thus power can be reduced by lowering the voltage, which occurred with each new generation of technology, and power is a function of the voltage squared.
- Typically, the voltage was reduced about 15% per generation. In 20 years, voltages have gone from 5 V to 1 V, which is why the increase in power is only 30 times.

■ In CMOS IC technology, power is as follows:

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

×30      5V → 1V      ×1000

# Power density
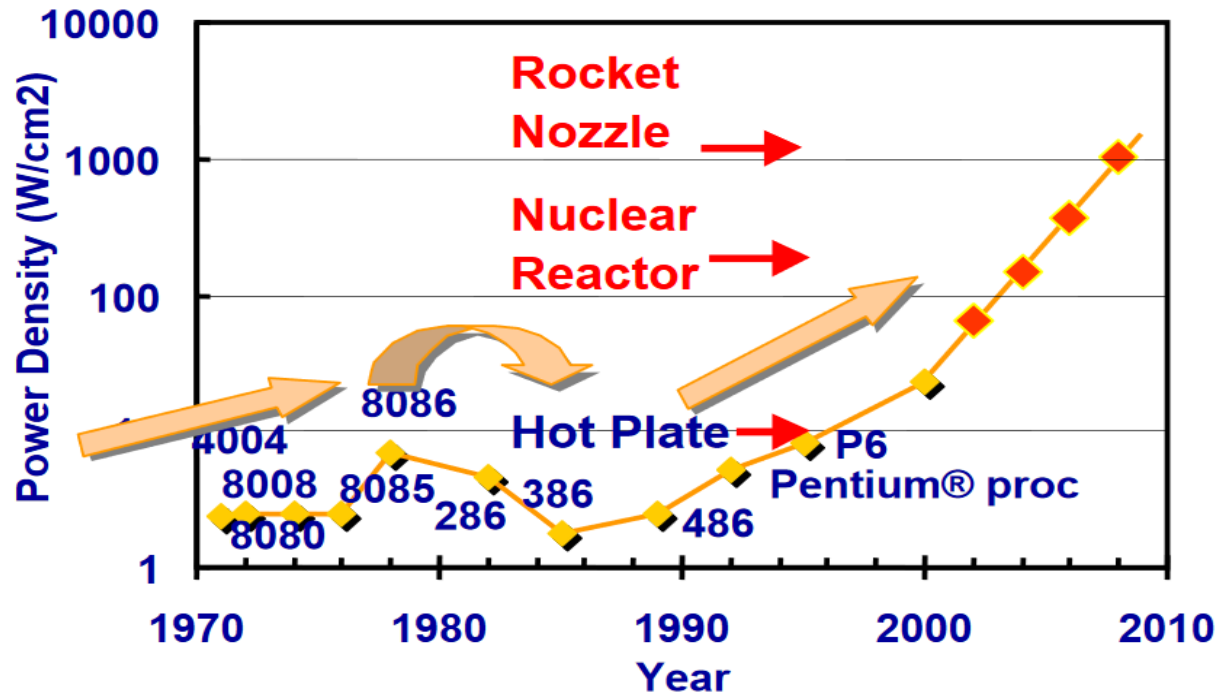


**Power Density**

Power Density increase

# Power density

# Power consumption

- Although power provides a limit to what we can cool, in the post-PC era the really valuable resource is energy. Battery life can trump performance in the personal mobile device, and

- the architects of warehouse scale computers try to reduce the costs of powering and cooling 100,000 servers as the costs are high at this scale.

# Reducing Power

- Suppose a new CPU has
    - 85% of capacitive load of old CPU
    - 15% less voltage and 15% frequency reduction
- What is the impact on dynamic power?

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

- Hence, the new processor uses about half the power of the old processor.
- The power wall
    - We can't reduce voltage further
    - We can't remove more heat
- How else can we improve performance?

# Uniprocessor Performance
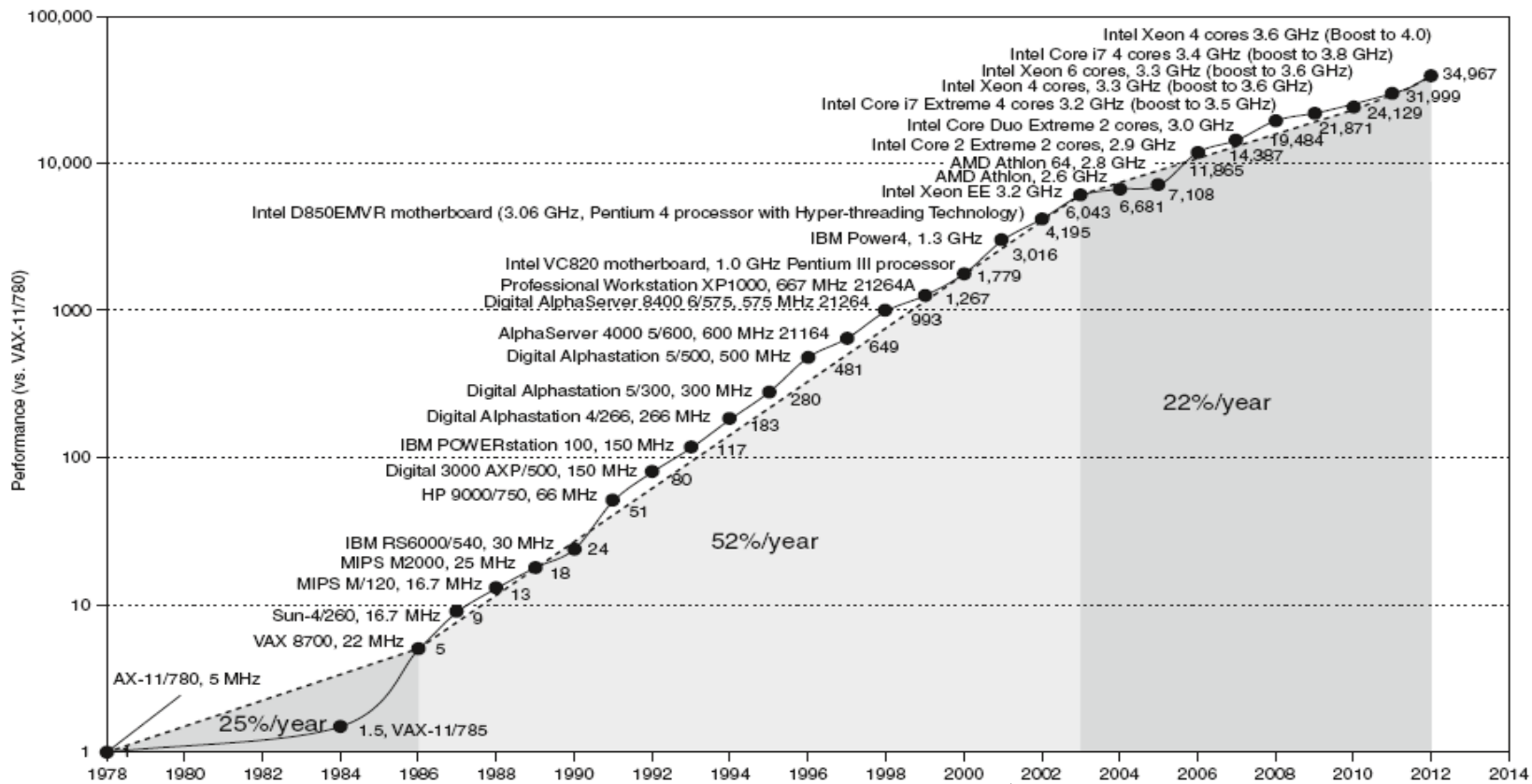
Since 2002, the limits of power, available instruction-level parallelism, and

long memory latency have slowed uniprocessor performance recently, to about 22% per year.

# Multiprocessors

- Rather than continuing to decrease the response time of one program running on the single processor, as of 2006 all desktop and server companies are shipping microprocessors with multiple processors per chip, where the benefit is often more on throughput than on response time.

- 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)

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

# CINT2006 for Intel Core i7 920

| Description | Name | Instruction Count x $10^9$ | CPI | Clock cycle time (seconds x $10^{-9}$) | Execution Time (seconds) | Reference Time (seconds) | SPECratio |
|---|---|---|---|---|---|---|---|
| Interpreted string processing | perl | 2252 | 0.60 | 0.376 | 508 | 9770 | 19.2 |
| Block-sorting compression | bzip2 | 2390 | 0.70 | 0.376 | 629 | 9650 | 15.4 |
| GNU C compiler | gcc | 794 | 1.20 | 0.376 | 358 | 8050 | 22.5 |
| Combinatorial optimization | mcf | 221 | 2.66 | 0.376 | 221 | 9120 | 41.2 |
| Go game (AI) | go | 1274 | 1.10 | 0.376 | 527 | 10490 | 19.9 |
| Search gene sequence | hmmer | 2616 | 0.60 | 0.376 | 590 | 9330 | 15.8 |
| Chess game (AI) | sjeng | 1948 | 0.80 | 0.376 | 586 | 12100 | 20.7 |
| Quantum computer simulation | libquantum | 659 | 0.44 | 0.376 | 109 | 20720 | 190.0 |
| Video compression | h264avc | 3793 | 0.50 | 0.376 | 713 | 22130 | 31.0 |
| Discrete event simulation library | omnetpp | 367 | 2.10 | 0.376 | 290 | 6250 | 21.5 |
| Games/path finding | astar | 1250 | 1.00 | 0.376 | 470 | 7020 | 14.9 |
| XML parsing | xalancbmk | 1045 | 0.70 | 0.376 | 275 | 6900 | 25.1 |
| Geometric mean | – | – | – | – | – | – | 25.7 |

# SPEC Power Benchmark

- Given the increasing importance of energy and power, SPEC added a benchmark to measure power. It reports power consumption of servers at different workload levels, divided into 10% increments, over a period of time.
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

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

- ssj_ops$i$ is performance at each 10% increment and
- power$i$ is power consumed at each performance level.

# SPECpower_ssj2008 for Xeon X5650

| Target Load % | Performance (ssj_ops) | Average Power (Watts) |
|---|---|---|
| 100% | 865,618 | 258 |
| 90% | 786,688 | 242 |
| 80% | 698,051 | 224 |
| 70% | 607,826 | 204 |
| 60% | 521,391 | 185 |
| 50% | 436,757 | 170 |
| 40% | 345,919 | 157 |
| 30% | 262,071 | 146 |
| 20% | 176,061 | 135 |
| 10% | 86,784 | 121 |
| 0% | 0 | 80 |
| Overall Sum | 4,787,166 | 1,922 |
| $\Sigma$ssj_ops/$\Sigma$power = | | 2,490 |

# Pitfall: Amdahl's Law

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

$$T_{improved} = \frac{T_{affected}}{improvemen\ t\ factor} + T_{unaffected}$$

■ Example: multiply accounts for 80s/100s

  ■ Suppose a program runs in 100 seconds on a computer, with multiply operations responsible for 80 seconds of this time. How much do I have to improve the speed of multiplication if I want my program to run five times faster?

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

■ Corollary: make the common case fast

# 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

$$MIPS = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - 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