

# 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

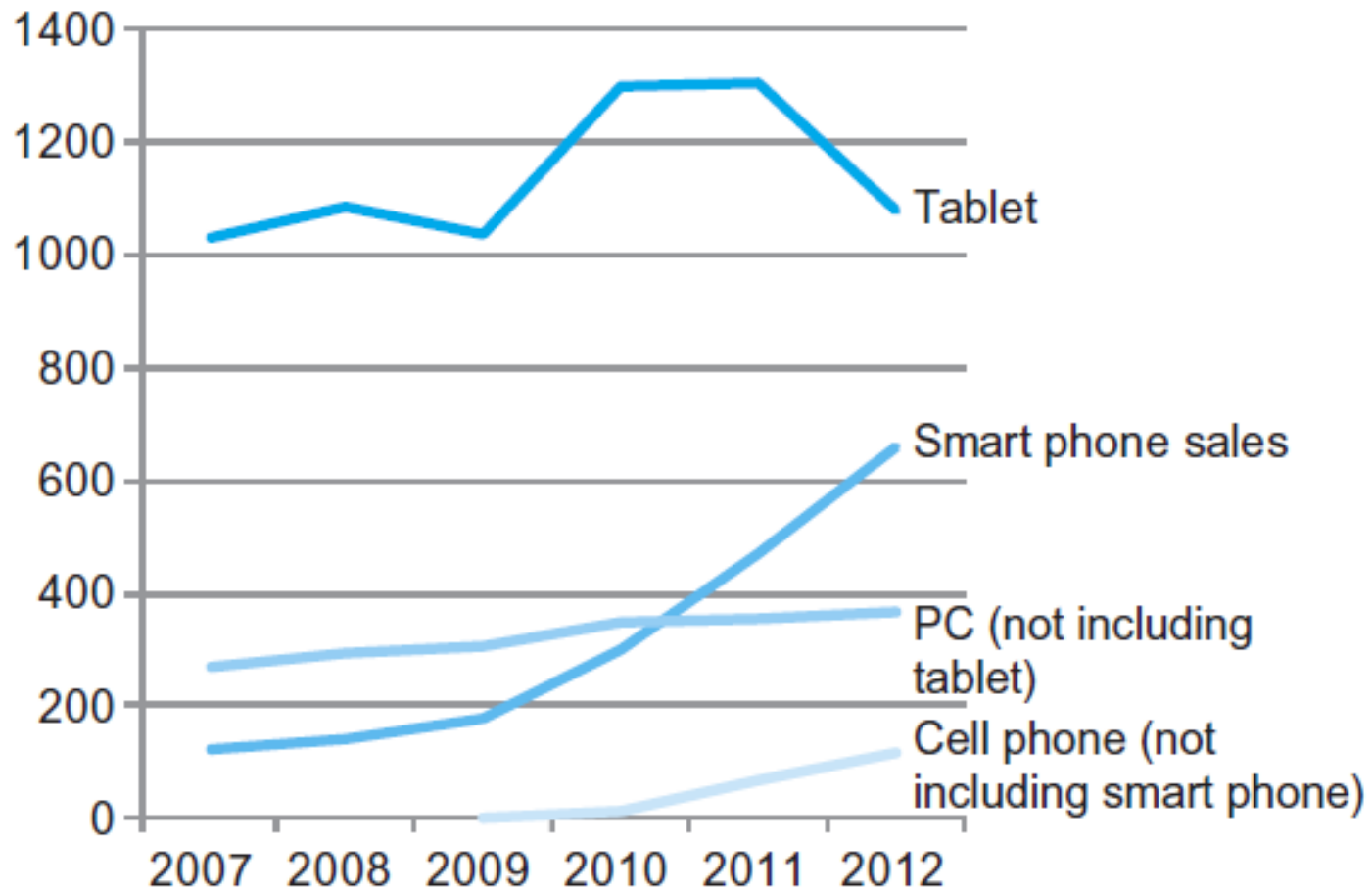
# Classes of Computers

- 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

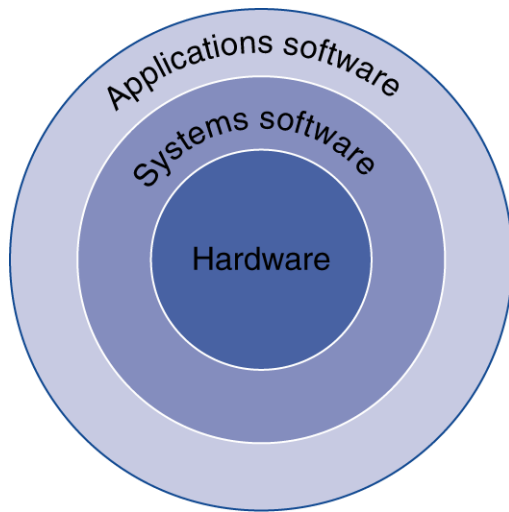


# Eight Great Ideas

- 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

# 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 RISC-V)

```
swap:
  slli x6, x11, 3
  add x6, x10, x6
  ld x5, 0(x6)
  ld x7, 8(x6)
  sd x7, 0(x6)
  sd x5, 8(x6)
  jalr x0, 0(x1)
```

**RISC-V**

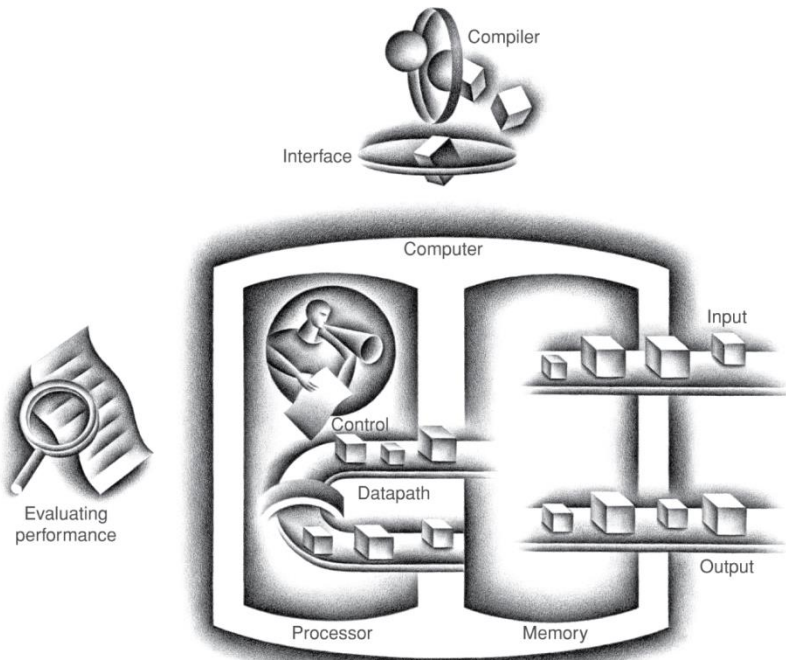
Assembler

Binary machine  
language  
program  
(for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000011001111
```

# Components of a Computer

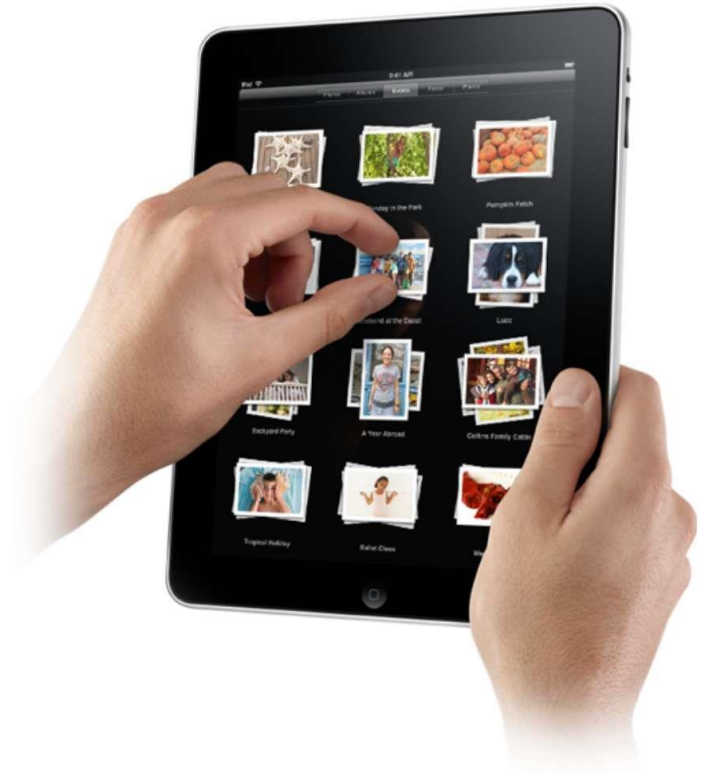
## The BIG Picture



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

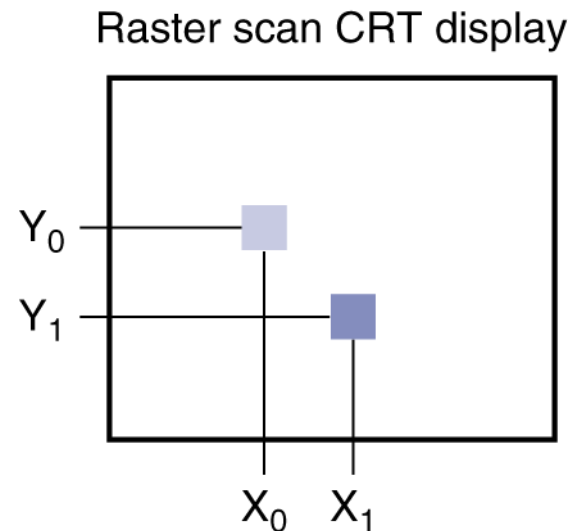
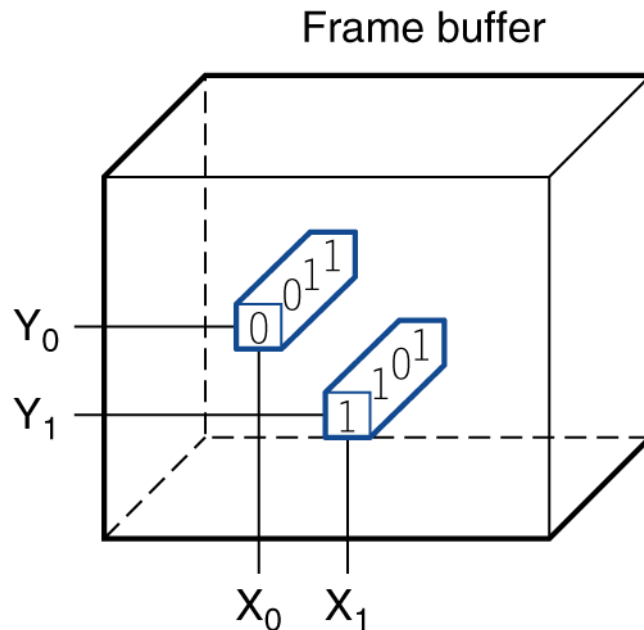
# Touchscreen

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

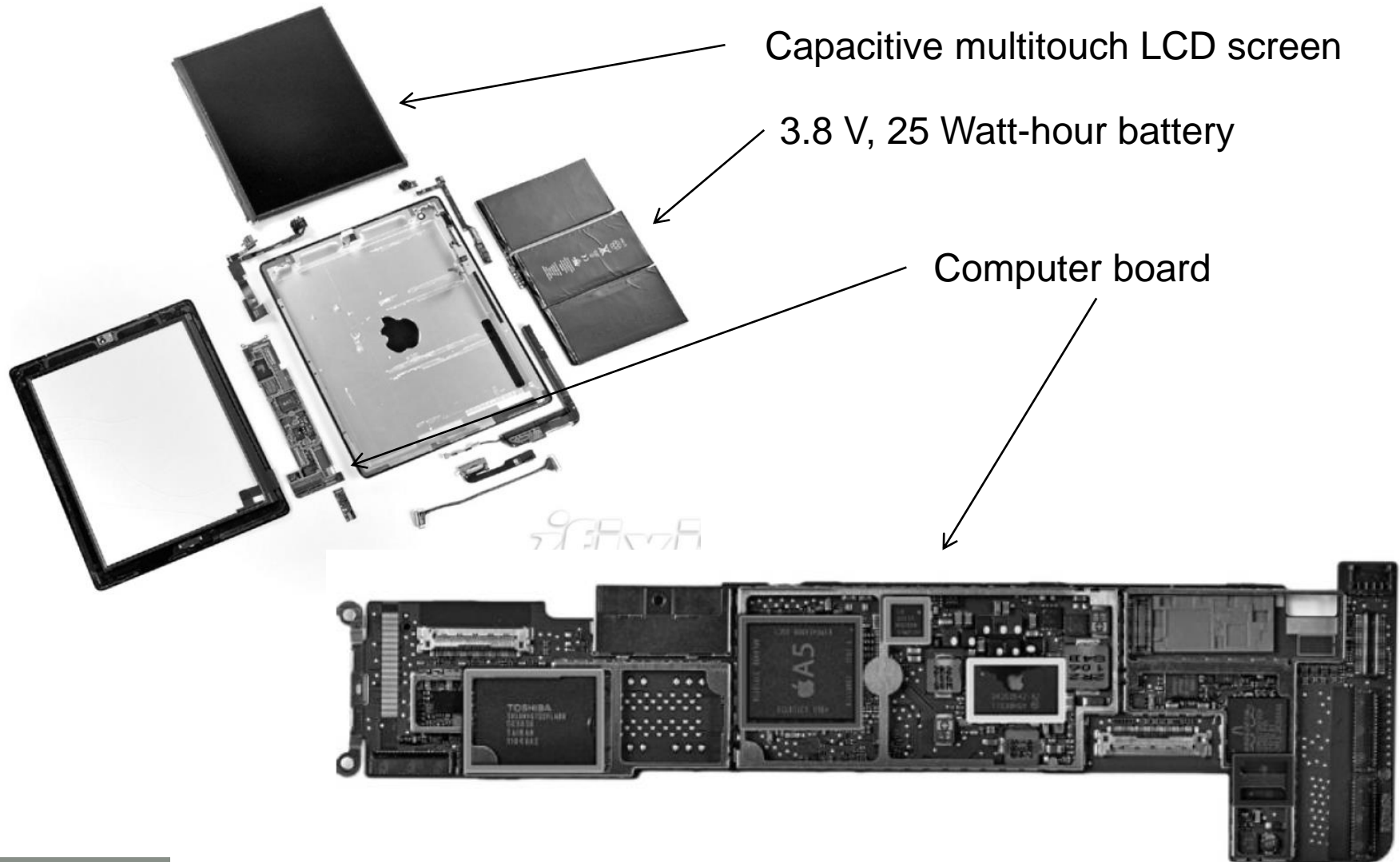


# Through the Looking Glass

- 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

- Apple A5



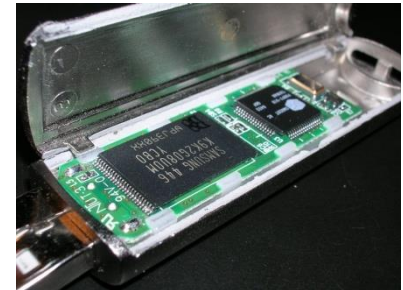
# Abstractions

## The BIG Picture

- Abstraction helps us deal with **complexity**
  - Hide lower-level detail
- **Instruction set architecture (ISA)**
  - The hardware/software interface
- **Application binary interface (ABI)**
  - 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

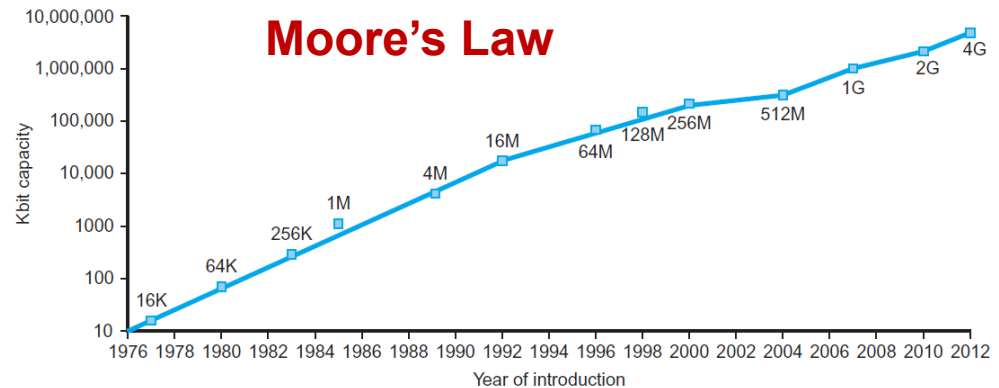
- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



# Technology Trends

## ■ Electronics technology **continues to evolve**

- Increased capacity and performance
- Reduced cost



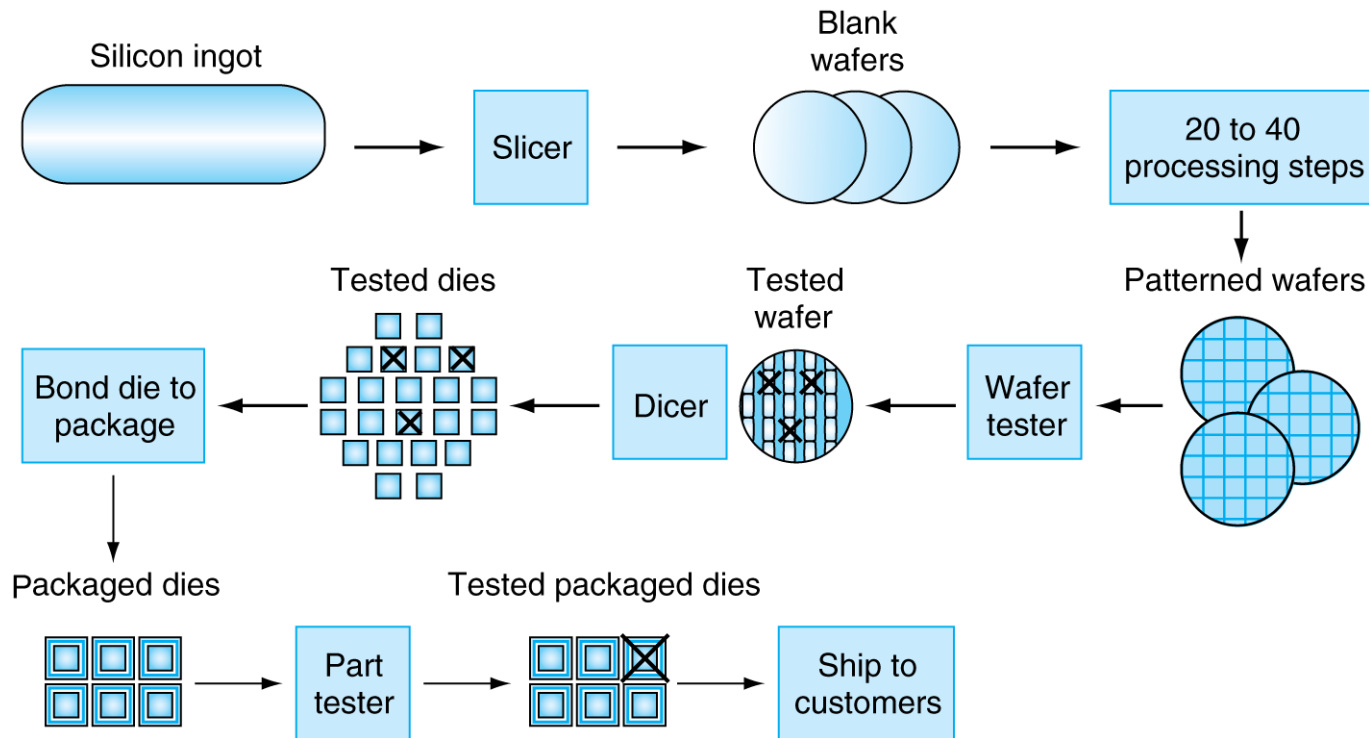
DRAM capacity

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

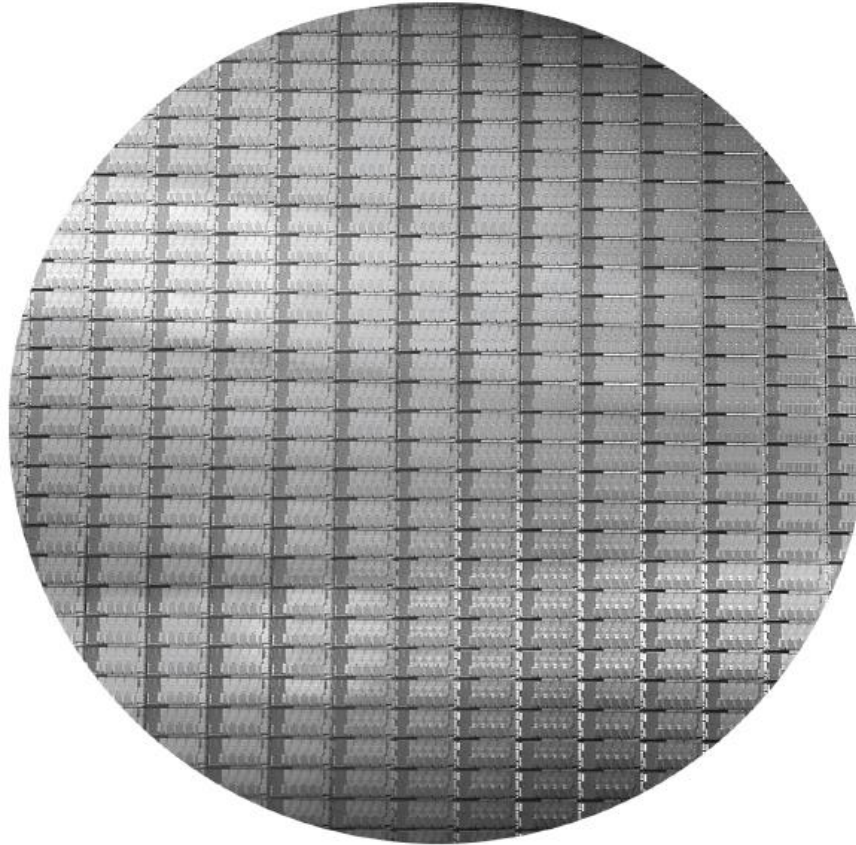
# Manufacturing ICs



- Yield: proportion of working dies per wafer



# Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm



# Integrated Circuit Cost

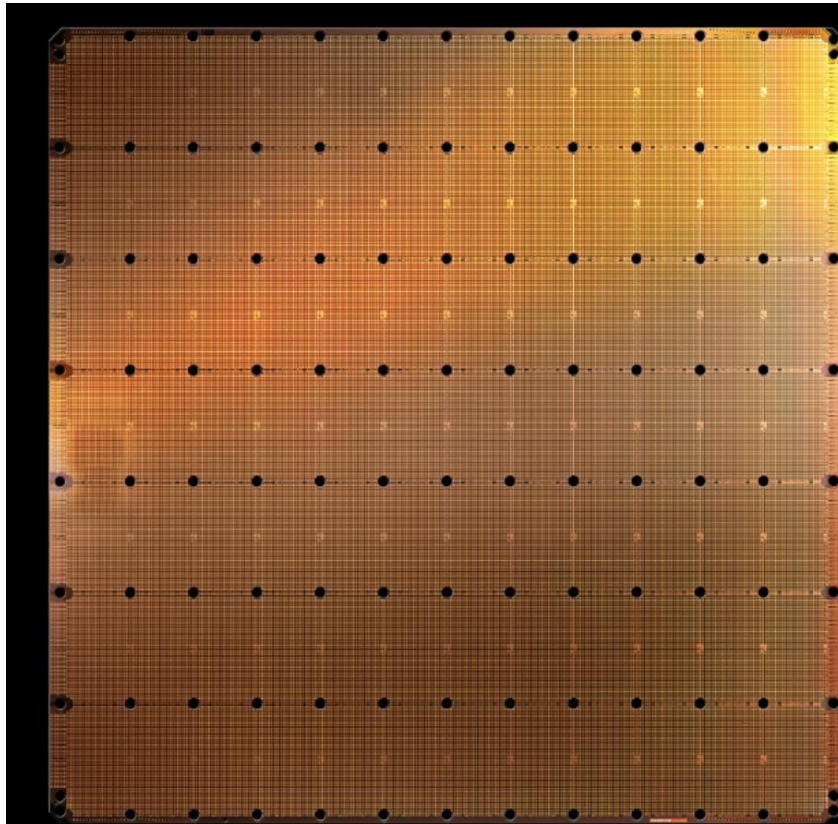
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

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

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

- 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

# Wafer-Scale Computer!!



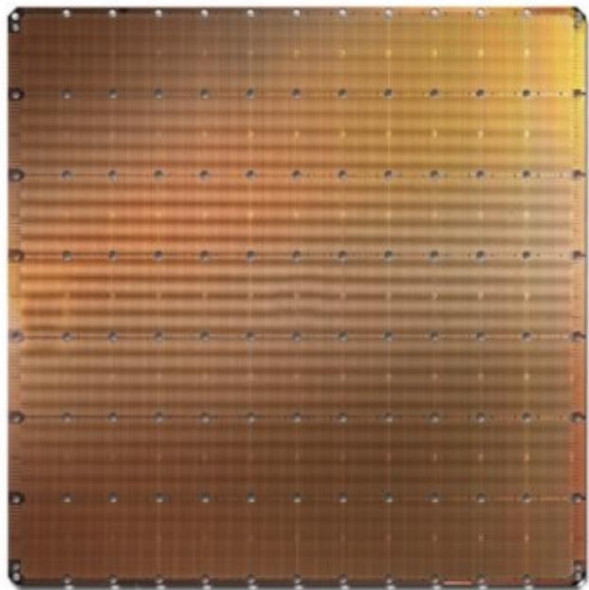
**Cerebras WSE**

1.2 Trillion transistors  
46,225 mm<sup>2</sup> silicon



**Largest GPU**

21.1 Billion transistors  
815 mm<sup>2</sup> silicon



**The WSE is the largest chip ever built**

**56x the size of the largest GPU**

The Cerebras Wafer Scale Engine 46,225 mm<sup>2</sup> with 1.2 Trillion transistors and 400,000 AI-optimized cores.

By comparison, the largest Graphics Processing Unit is 815 mm<sup>2</sup> and has 21.1 Billion transistors.

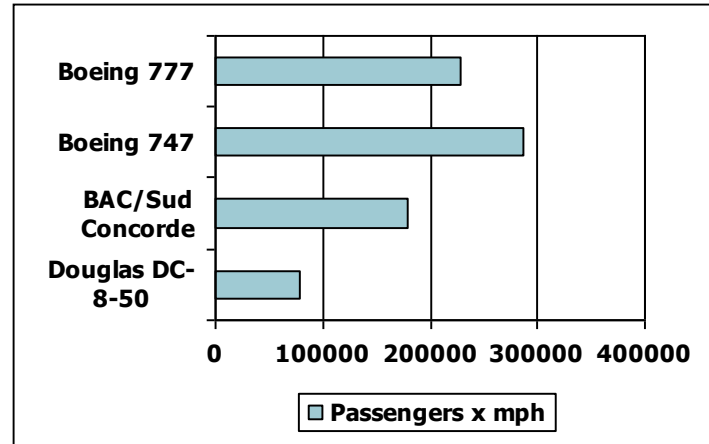
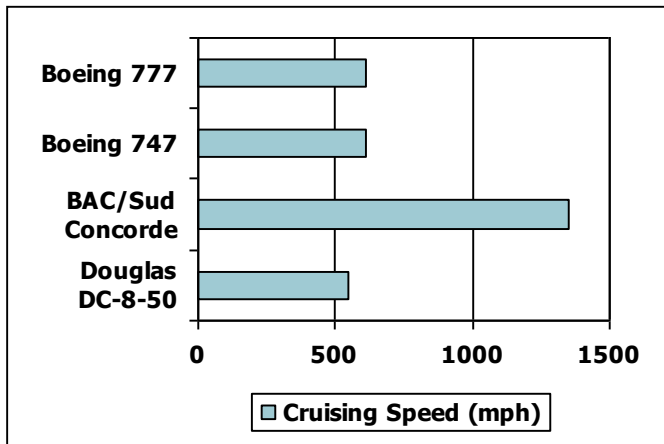
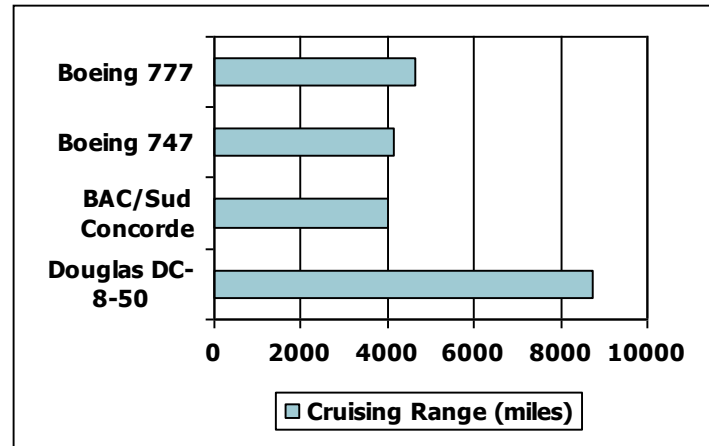
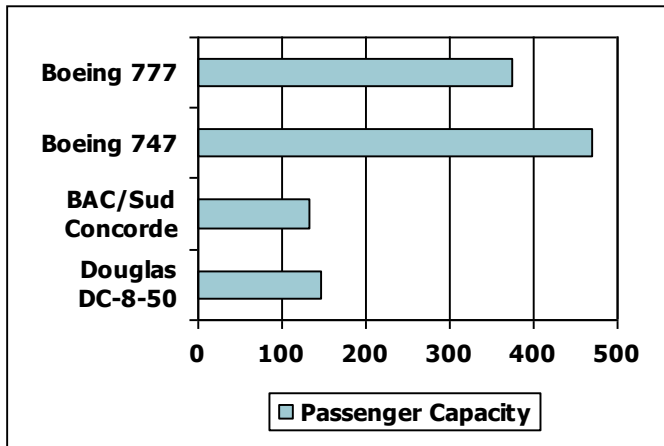
電腦系統公司Cerebras Systems發表了一款有史以來最大的半導體運算晶片Cerebras Wafer Scale Engine (WSE)，其尺寸超過一個iPad Air大小，上面擁有超過1.2兆個電晶體，以及40萬個人工智慧最佳化的核心，要為深度學習應用提供更強大的運算能力。

Cerebras Systems是一間由電腦架構師、電腦科學家以及深度學習研究人員組成的新公司，目的是要建立新型態的電腦，以加速深度學習的應用，而晶片WSE則是Cerebras Systems所發展的解決方案，其中一個重要的組成元素。這個巨型晶片的邊長接近22公分，面積比目前最大的GPU，還要大上56倍，而上面所擁有的運算資源，包括核心、記憶體甚至是記憶體頻寬，都比單一GPU晶片多上許多。

<https://www.ithome.com.tw/news/132516>

# 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/Execution Time
- “X is  $n$  time faster than Y”

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

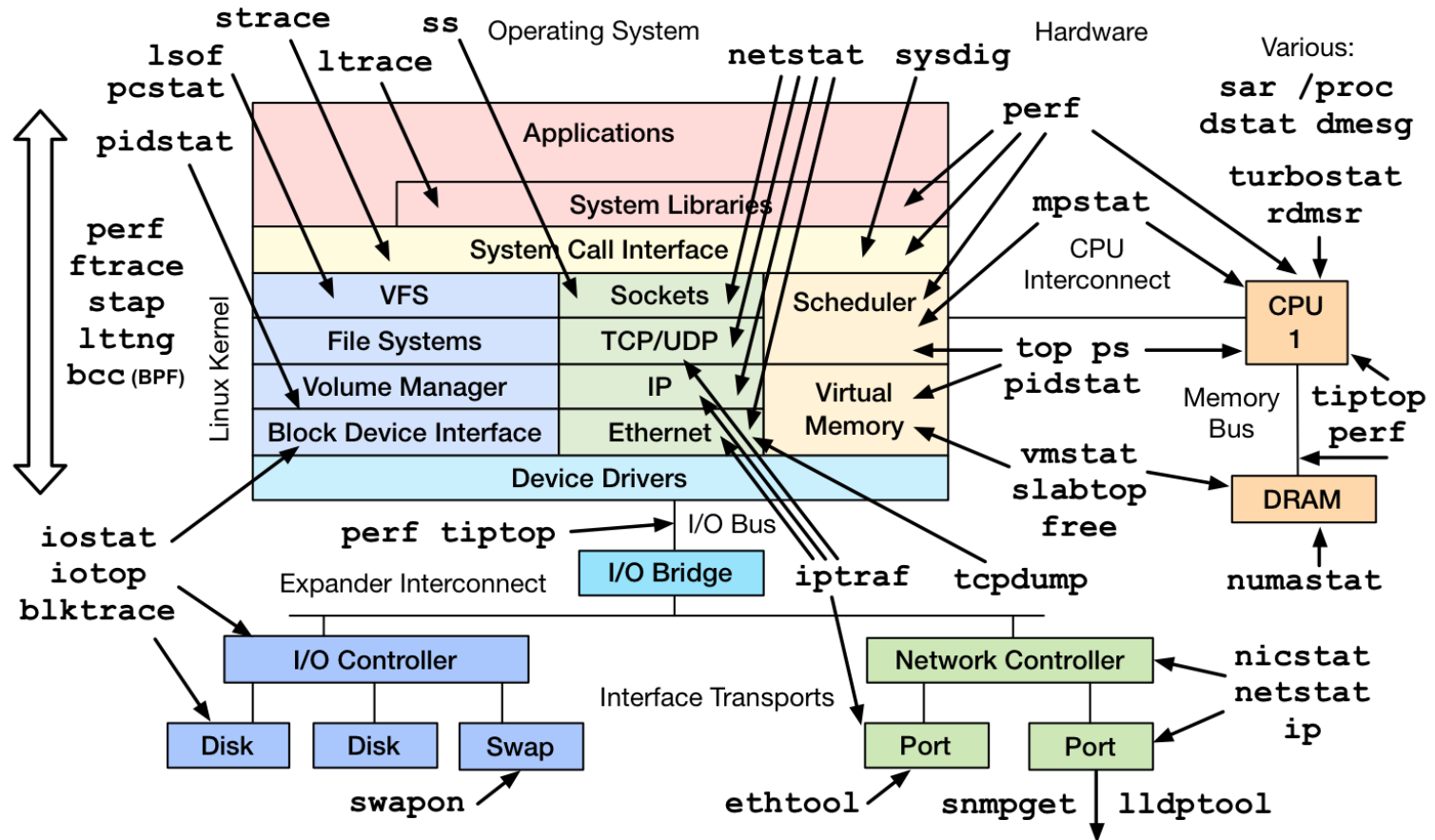
- Example: time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - So A is 1.5 times faster than B

# 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

# Performance Monitoring Tools

## Linux Performance Observability Tools



<http://www.brendangregg.com/linuxperf.html> 2017



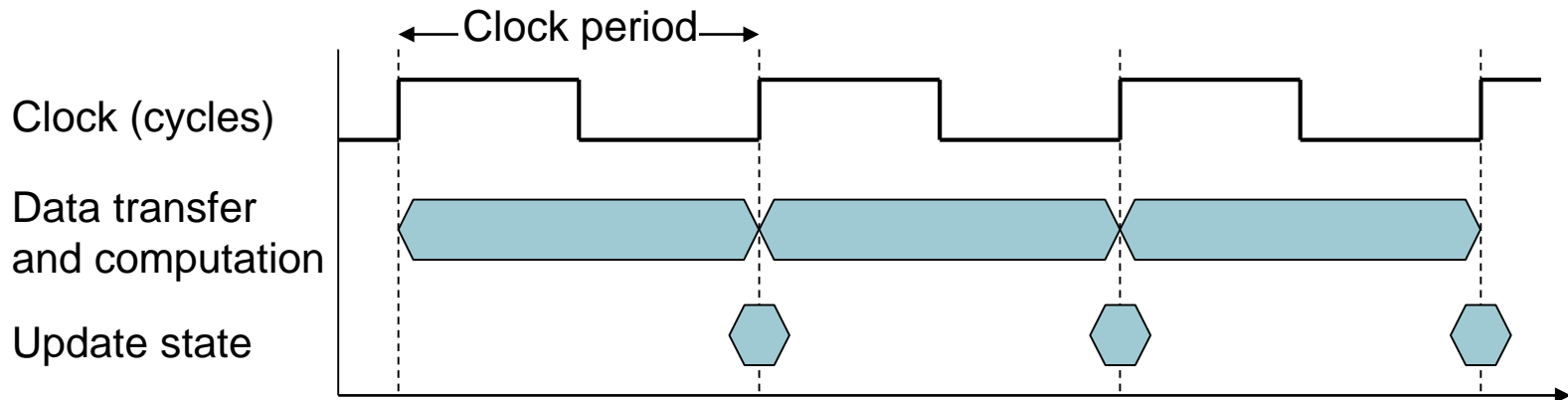
# Intel Vtune Performance Analyzer

- <https://software.intel.com/en-us/videos/hpc-applications-need-high-performance-analysis>



# 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

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

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

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

- 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?

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

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

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

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

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

...by this much

# CPI in More Details

- 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 =  $9/6 = 1.5$



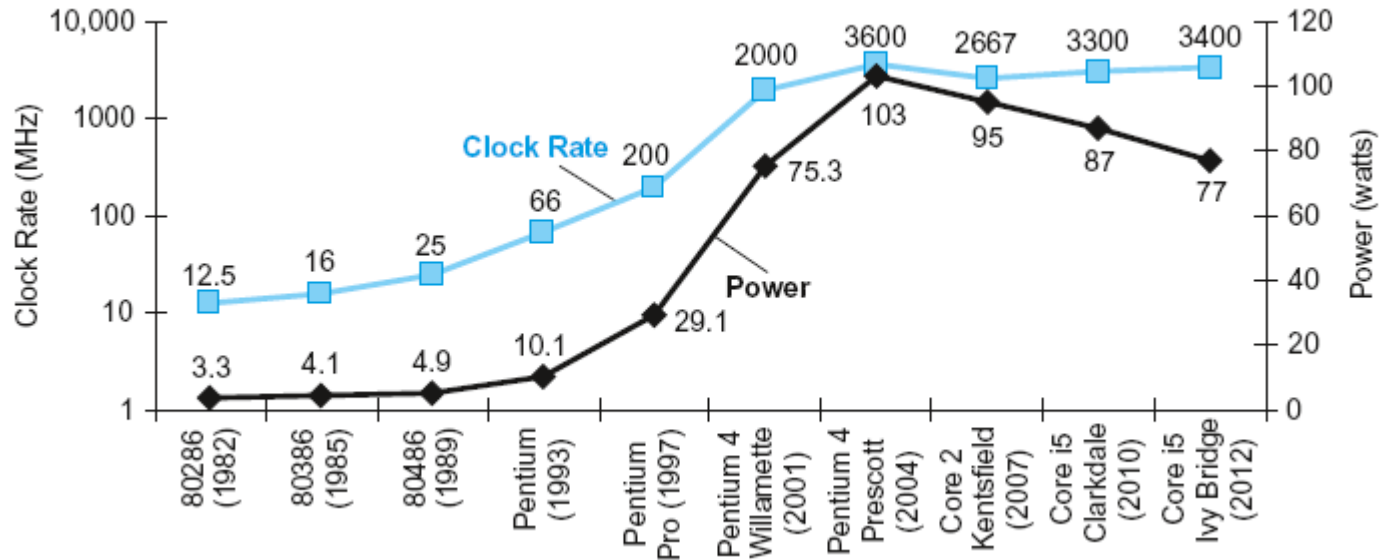
# Performance Summary

## 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,  $T_c$

# Power Trends



- In CMOS IC technology

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

×30

5V → 1V

×1000

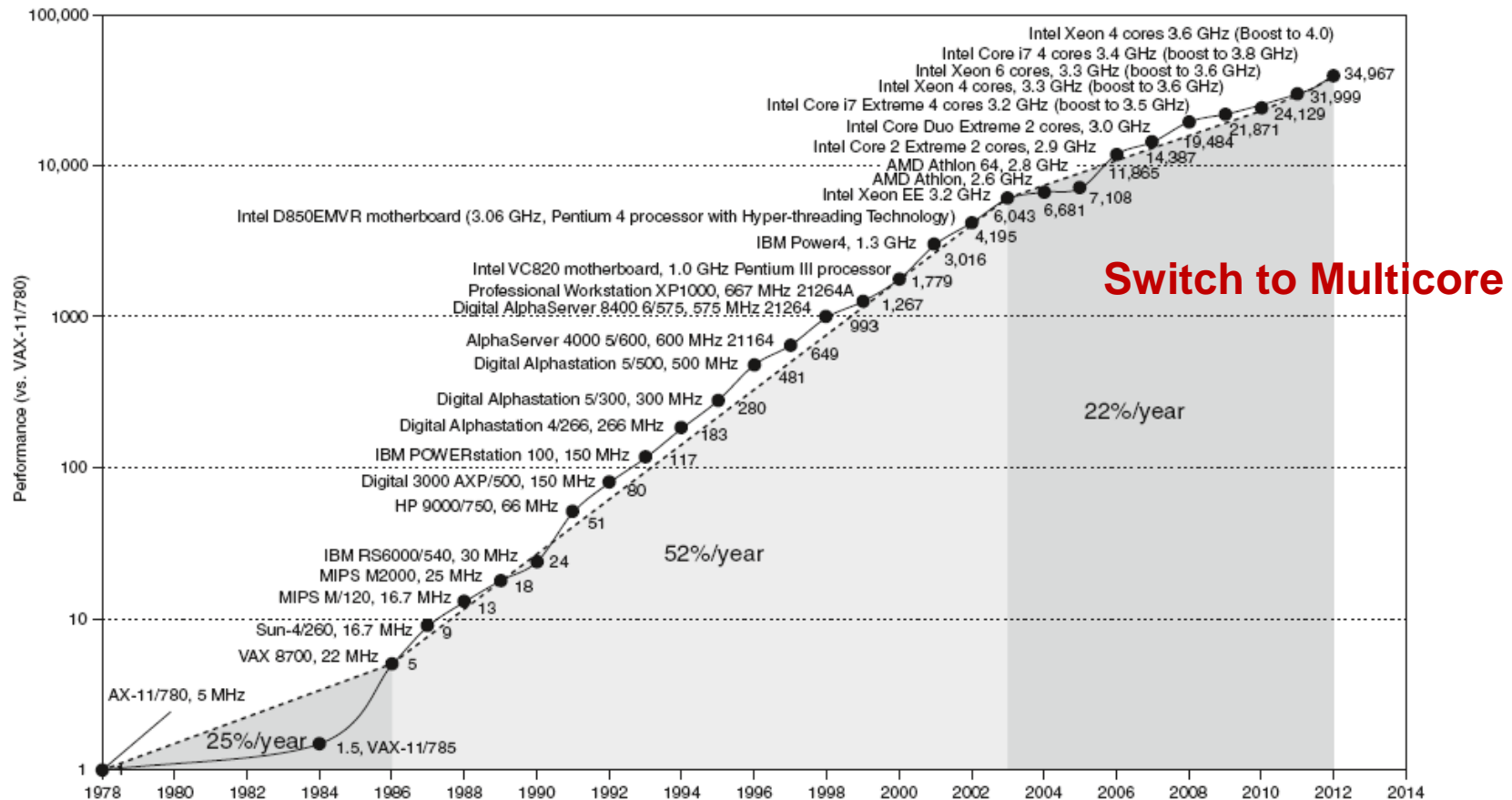
# Reducing Power

- 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



# 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)

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

# CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 <sup>9</sup>	CPI	Clock cycle time (seconds x 10 <sup>-9</sup> )	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

- 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 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
$\Sigma \text{ssj\_ops} / \Sigma \text{power} =$		2,490

# 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

# 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

# Ongoing Innovations

- Exascale computing
  - Accelerators for deep learning
  - Reconfigurable computing
  - Neuromorphic computing
  - Quantum computing
- 
- **Apple, Google, Facebook, Amazon, Microsoft – Which does not pay attention to computer architecture?**

# This Is an Engineering Course

- What does engineering mean?
- Engineering principles, and know-hows
- Research methodologies in science and engineering
- Improve core technologies to enable new applications

*We don't make AI; we make AI better!*

*~ AI System Architect*