Chapter 1

Computer Abstractions and Technology

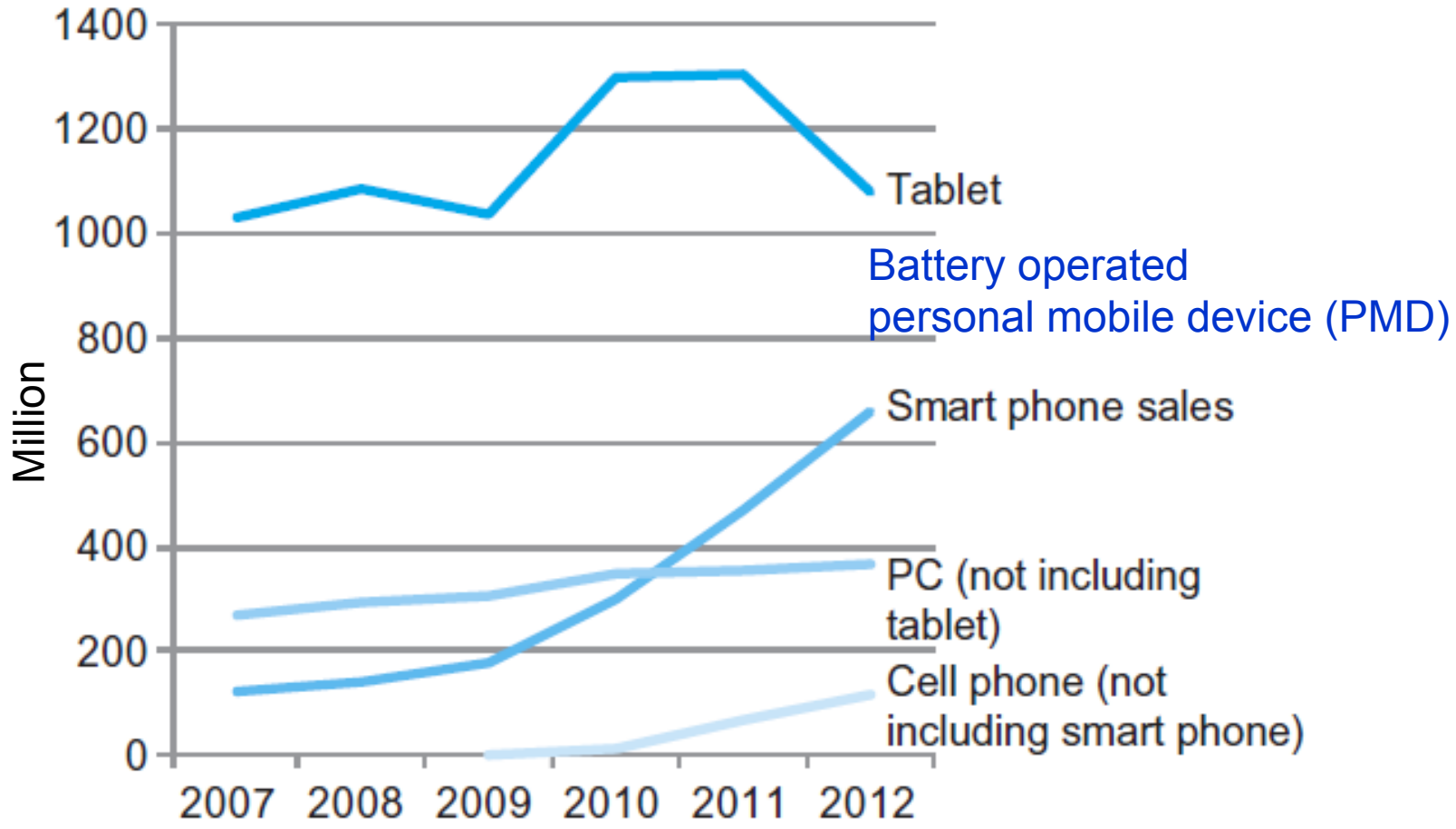
The Computer Revolution

- 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
  - Delivery of good performance to single users at low cost, a general purpose computer
  - Subject to cost/performance tradeoff
- Servers
  - Network based and much larger computers for high capacity, performance, and reliability
  - Widest range in cost and capability
    - Low-end servers for file storage vs. High-end servers (or supercomputers) for scientific and engineering calculations
- Embedded computers
  - Hidden as components of a system
  - Stringent energy/performance/cost constraints and lower tolerance for failure

The PostPC Era



The PostPC Era

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

What You Will Learn

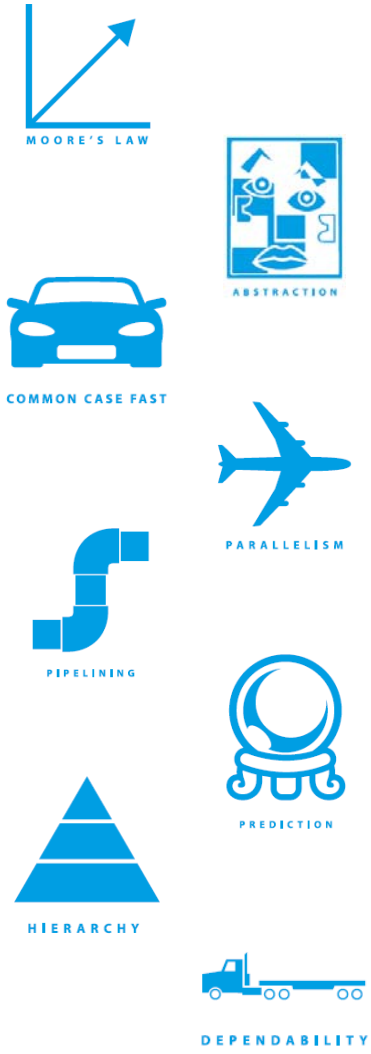
- You can answer the following questions:
  - How high-level C/Java programs are translated into the machine language
    - And how the hardware executes them
  - The hardware/software interface
  - What determines the performance of program
    - And how it can be improved
  - How hardware designers improve performance
  - What is parallel processing

Understanding Performance

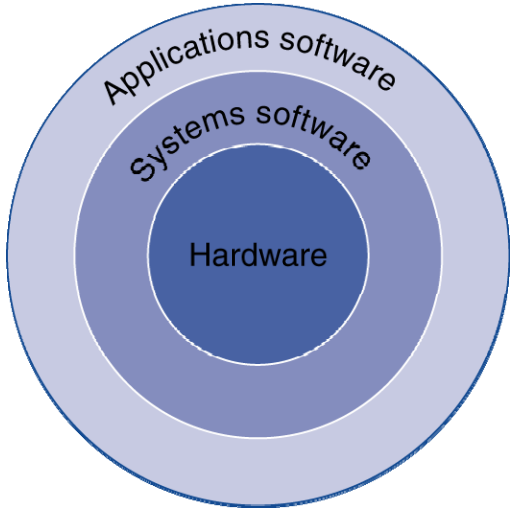
- Algorithm
  - Determine number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per algorithm/operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determine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

Eight Great Ideas in CA

- Design for ***Moore's Law*** (1965)
- 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



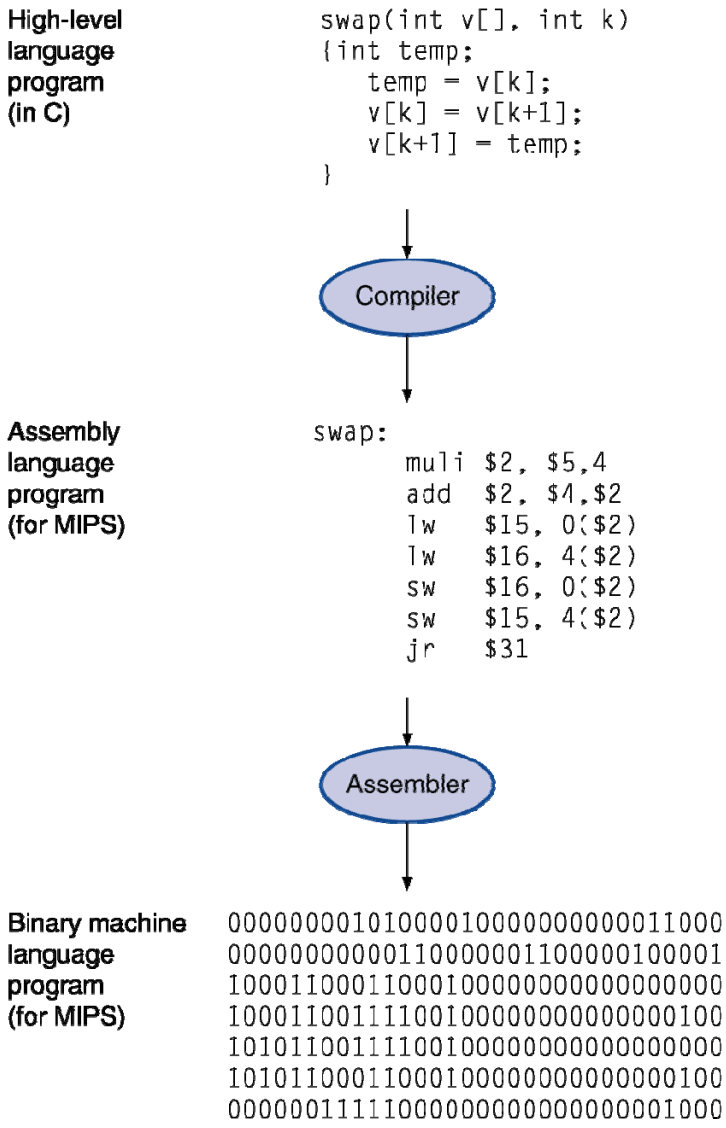
Hierarchy Layers in Programs



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

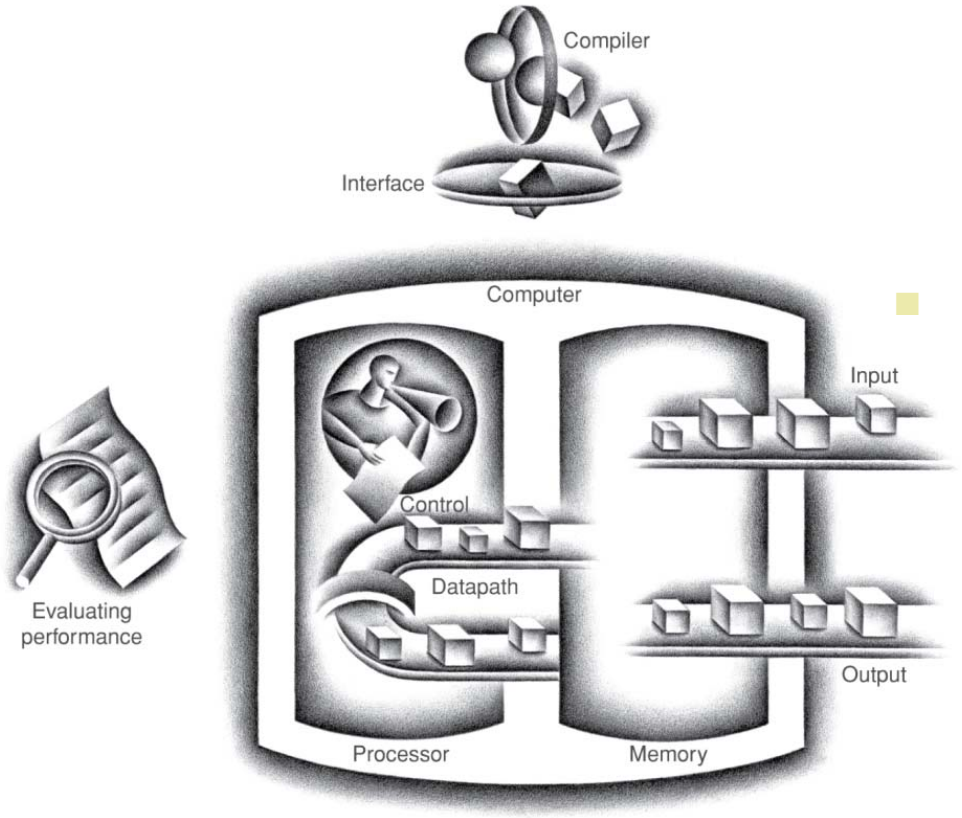
Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provide for productivity and portability
- Assembly language
  - Textual representation of instructions
- Machine language
  - Encoded instructions and data in binary digits (bits)



Components of a Computer

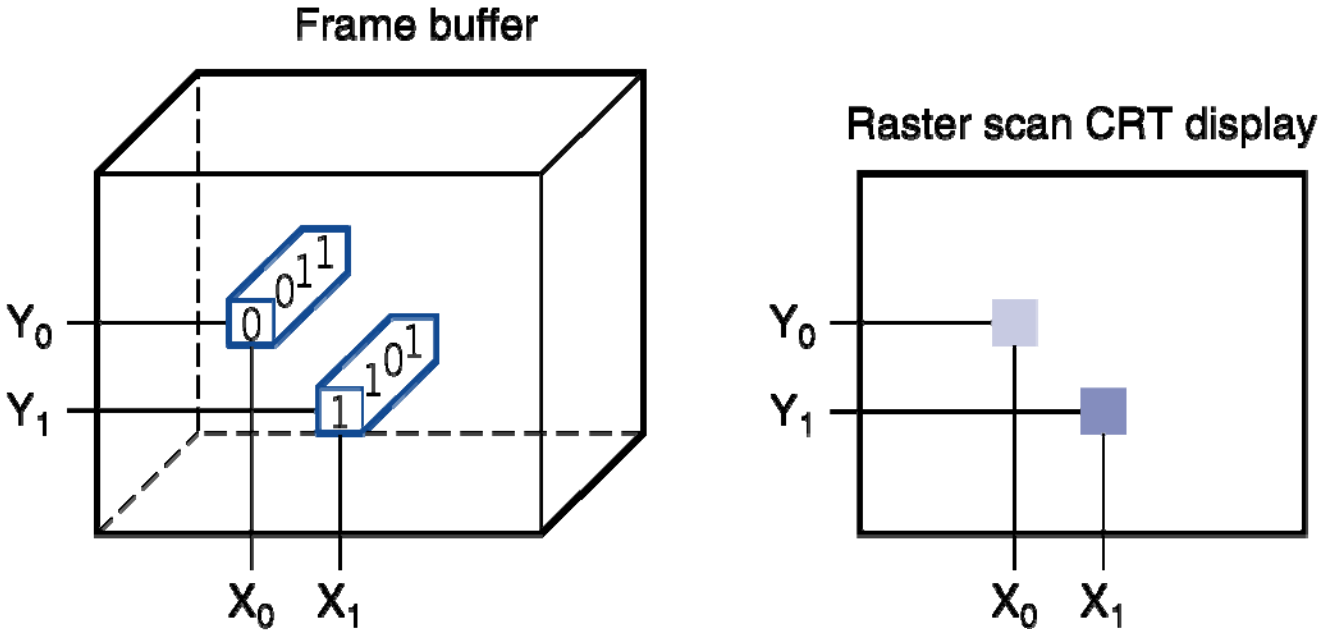
The BIG Picture



- Same components for all kinds of computer
  - Inputting/outputting data, processing data, and storing data
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Through the Looking Glass

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

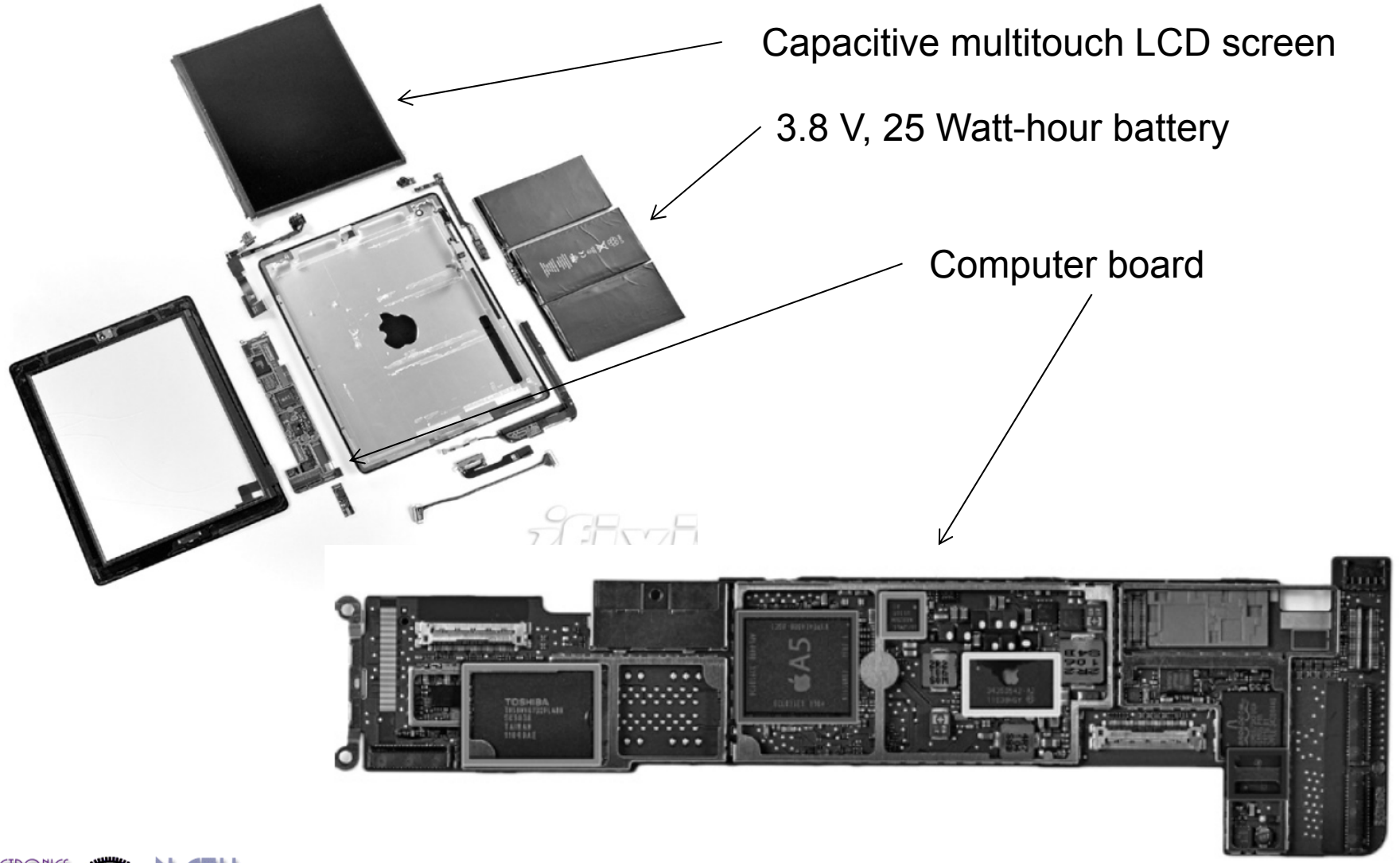


Touchscreen

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

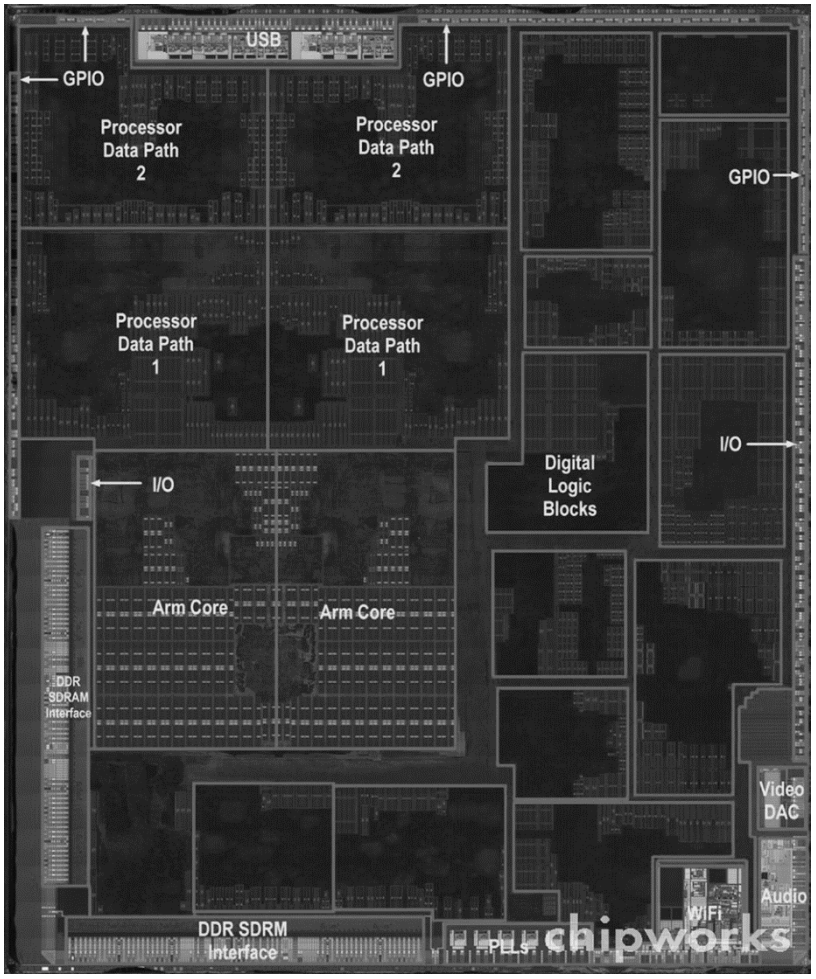
Opening the Box

Apple iPad 2



Inside the Processor

- Apple A5



Inside the Processor (CPU)

- Datapath: performs arithmetic operations on data
- Control: tells datapath, memory, and I/O devices what to do according to the wishes of the instruction
- Memory: data memory and instruction memory
  - DRAM (dynamic random access memory)
- Cache memory
  - Small fast SRAM memory for immediate access to data

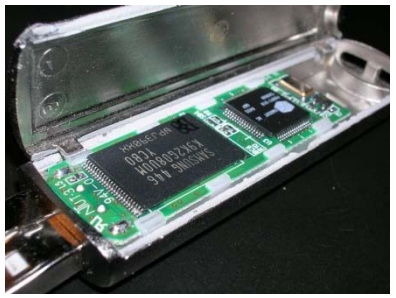
Abstractions

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

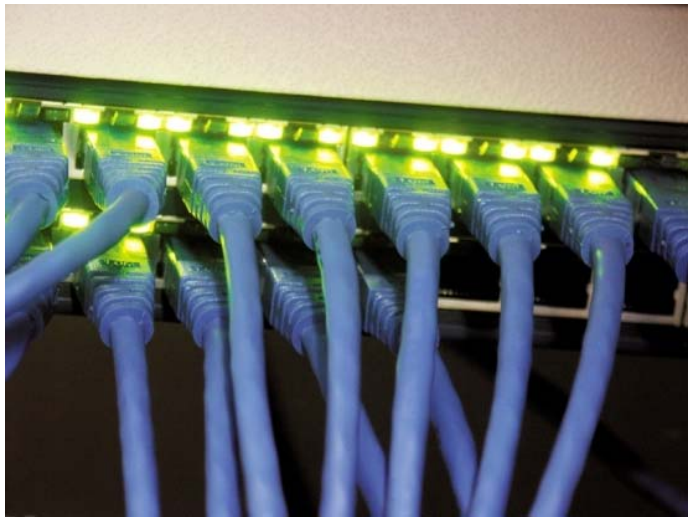
A Safe Place for Data

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



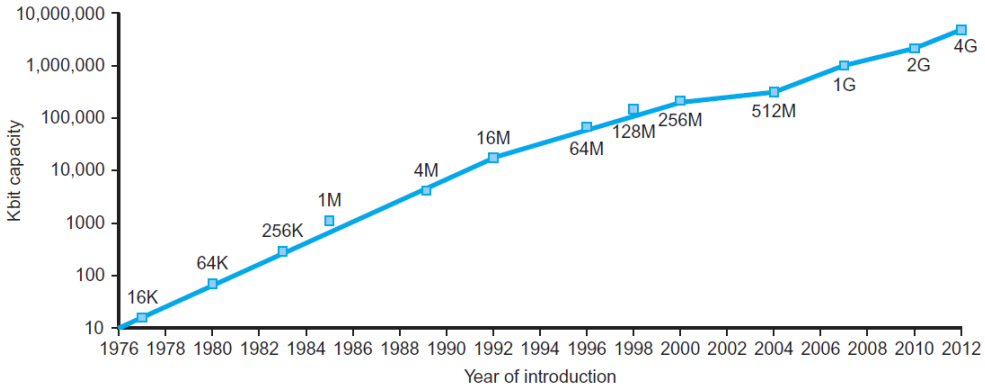
Networks (Communicating with Others)

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



Technology Trends

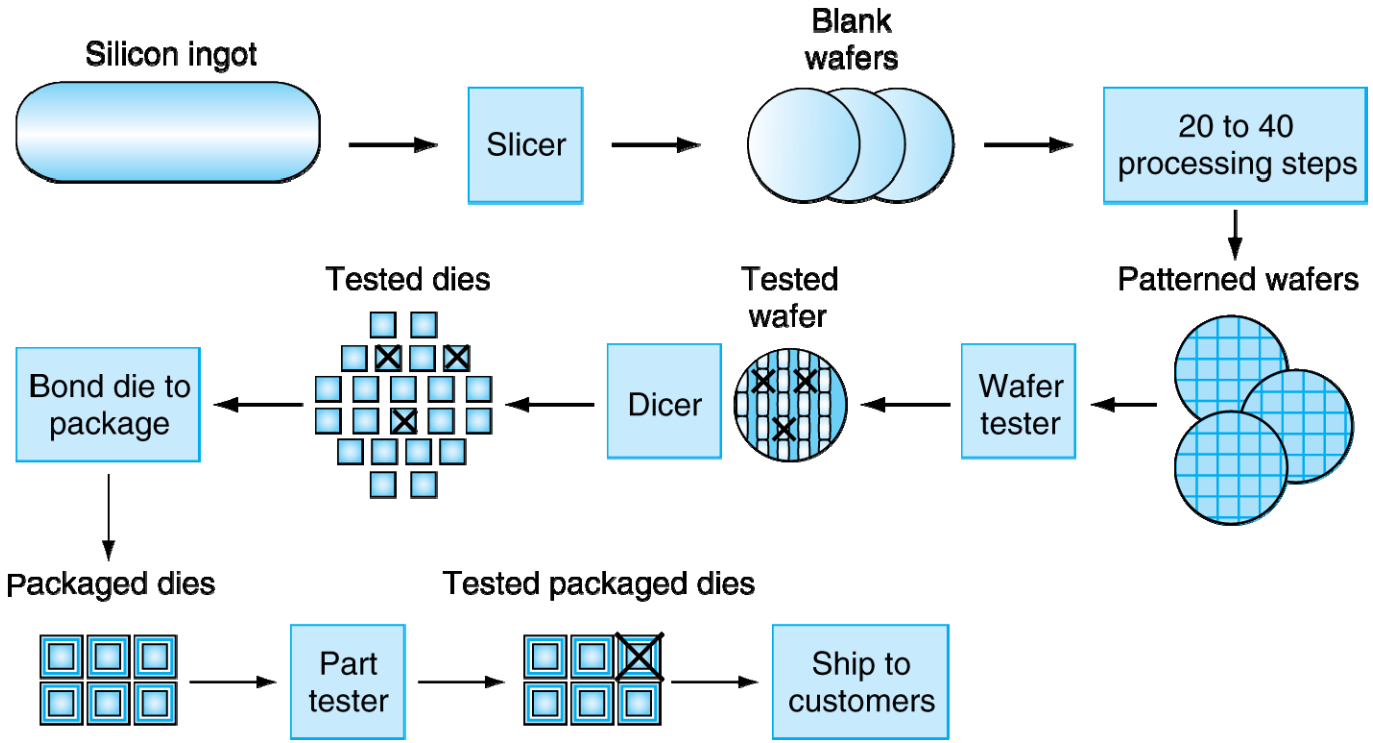
- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity

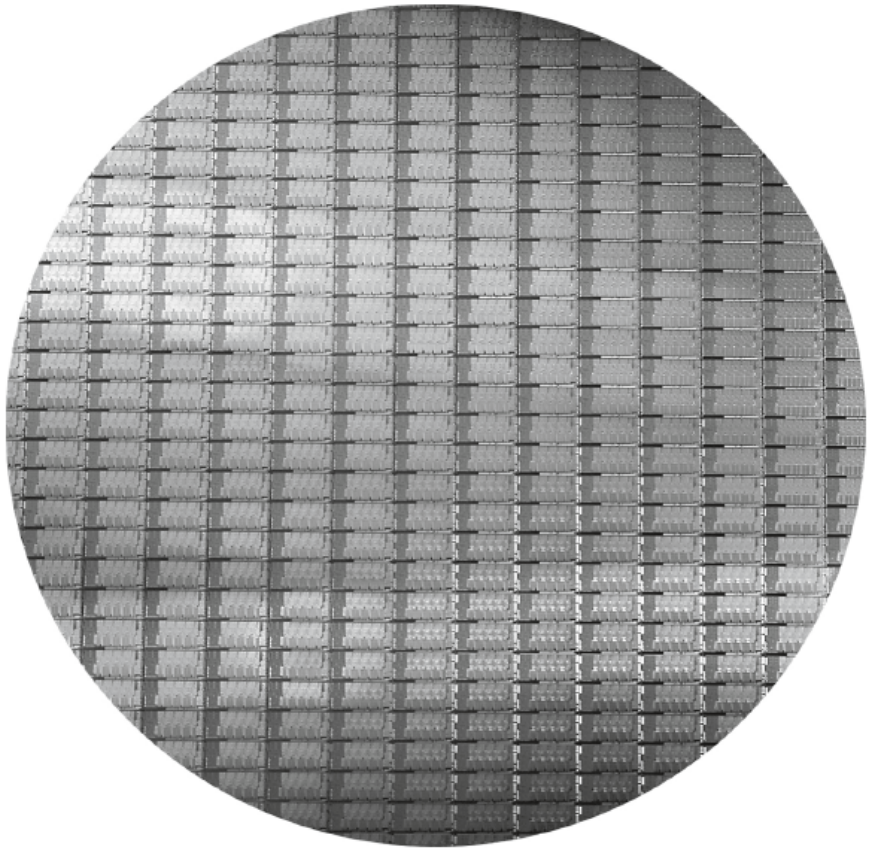
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Chip Manufacturing Process



- 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^2

Cost of a Chip Includes ...

- Die cost
  - affected by wafer cost, number of dies per wafer, and die yield (#good dies/#total dies)
  - goes roughly with the cube of the die area
- Testing cost
- Packaging cost
  - depends on pins, heat dissipation, ...

Cost of an IC

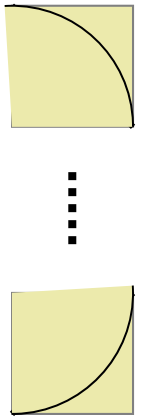
- A wafer is tested and chopped into dies

$$C_{\text{die}} = \frac{C_{\text{wafer}}}{\text{Die per wafer} \times \text{Die yield}}$$

$$\text{Die per wafer} \approx \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2} \times \sqrt{\text{Die area}}}$$

- The die is still tested and packaged into IC

$$C_{\text{IC}} = \frac{C_{\text{die}} + C_{\text{testing die}} + C_{\text{packaging and final test}}}{\text{Final test yield}}$$



Three Equations for IC Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$ Exactly derived eq.

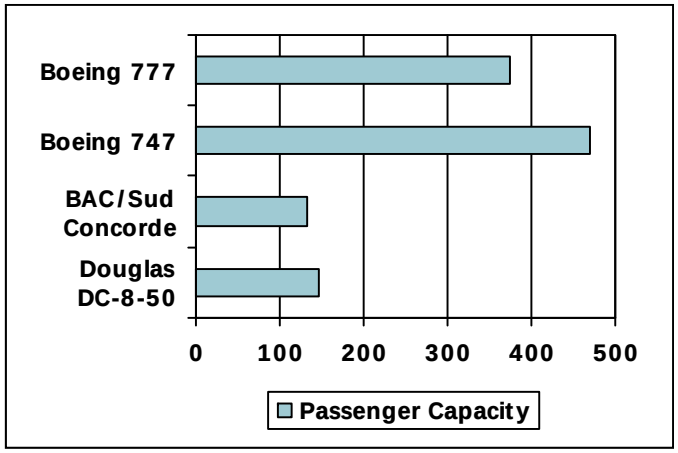
$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$ Approximation eq.

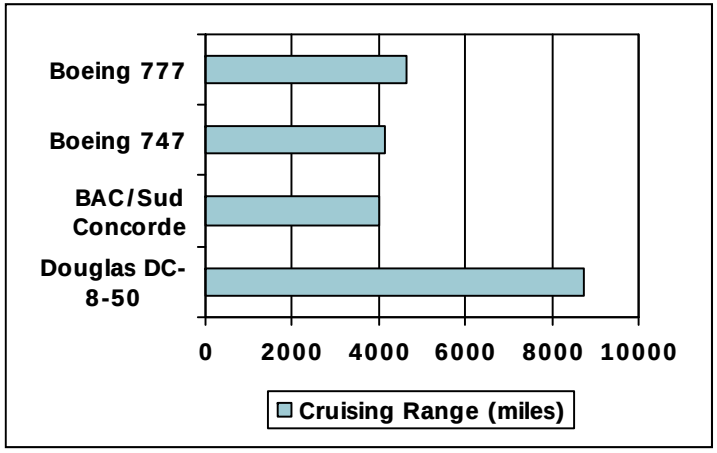
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$ Statistical eq.

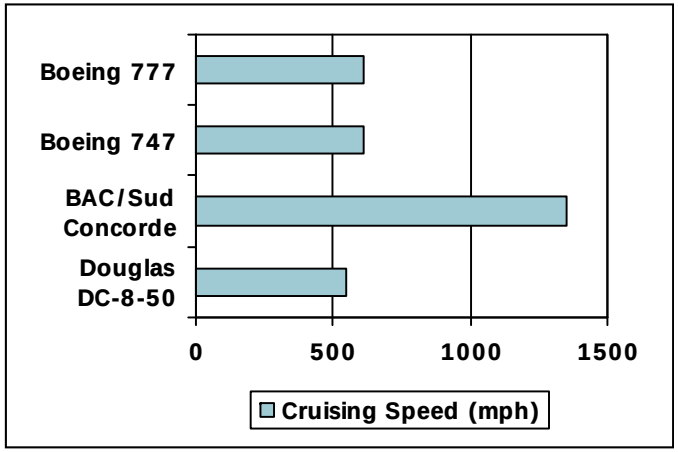
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

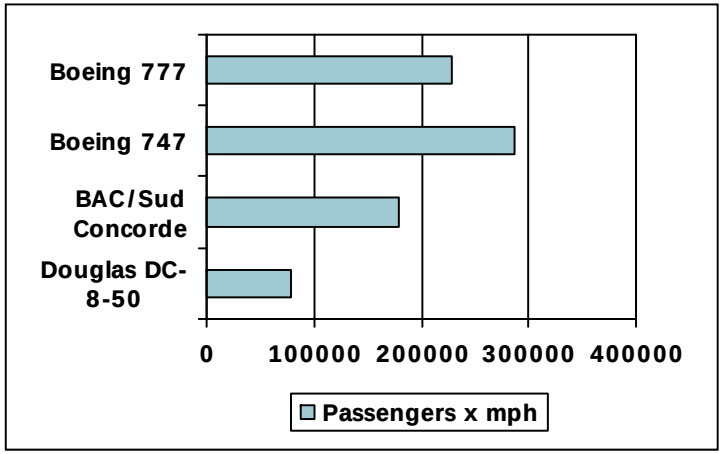
Defining Performance

- Which airplane has the best performance?









Response Time and Throughput

- Response time (aka execution time)
  - How long it takes to do a task
- Throughput (aka bandwidth)
  - Total work done per unit time
- PMDs are more focused on response time, while servers are more focused on throughput.
- Example: 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"

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

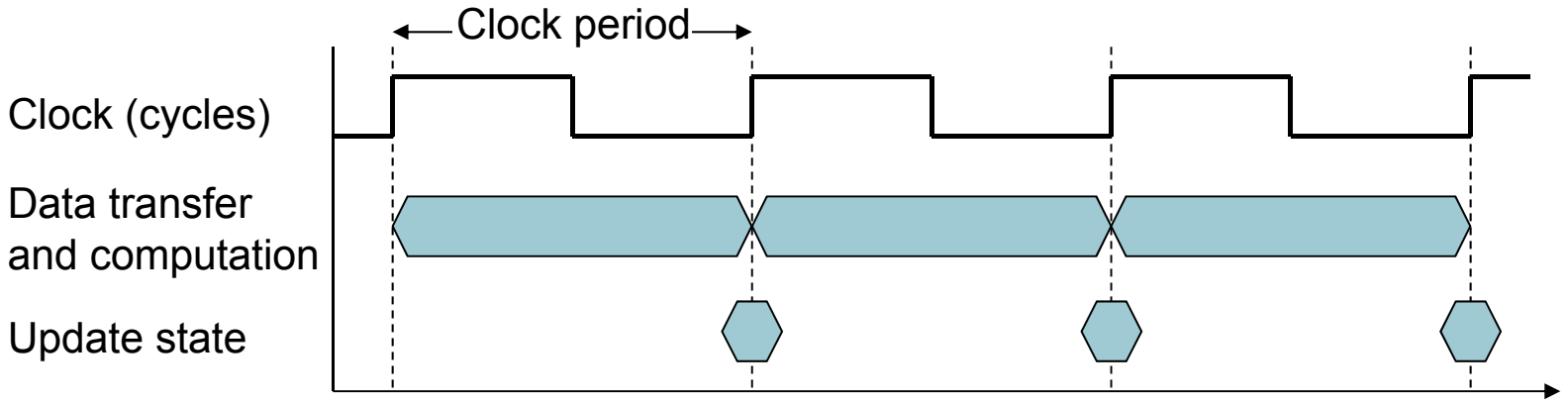
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

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 by CPU for processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
    - User CPU time: the CPU time spent in a program itself
    - System CPU time: the CPU time spent in OS performing tasks on behalf of the program

Execution Time and CPU Clocking

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



  - Clock period: duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Clock frequency (rate): cycles per second
    - e.g., 4.0GHz = 4000MHz = 4.0×10^{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 (or cycle count)
  - 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?

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

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

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

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

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?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

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

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles
  - Average CPI affected by instruction mix

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

  - Weighted average CPI

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

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

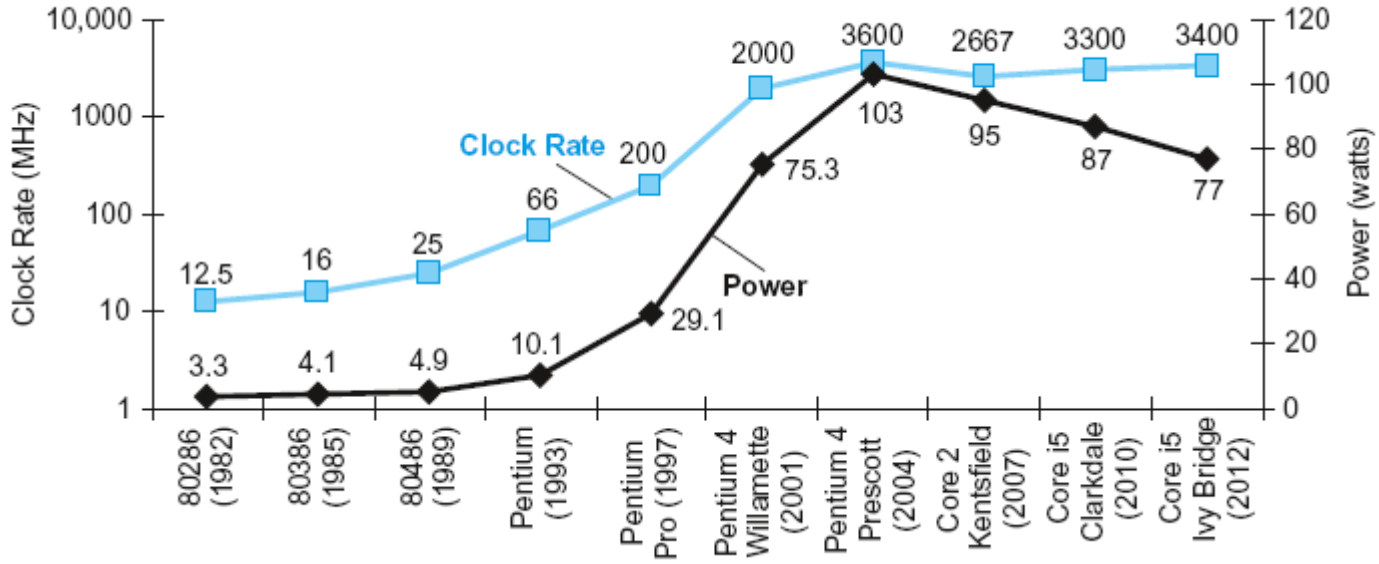
Performance Summary

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

Clock Rate and Power Trends



- In CMOS IC technology

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

×30

5V → 1V

×1000

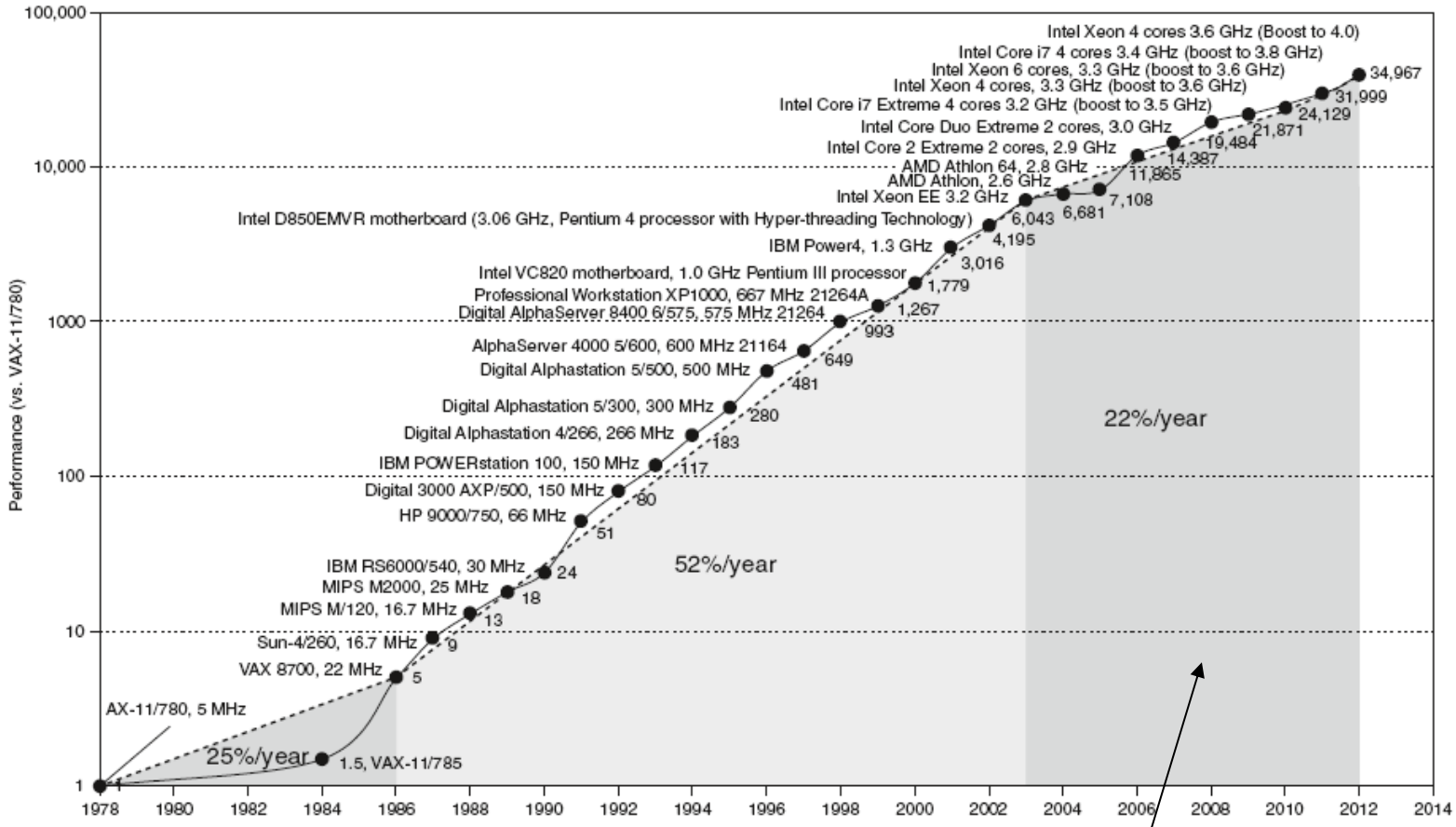
Reducing Power

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

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

- 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

- 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)

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

Remark

$$\text{e.g. } 1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

- SPECRatio is just a ratio rather than an absolute execution time
- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

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

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

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

server side Java operations per second per watt

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
Σssj_ops/Σpower =		2,490

有關效能的另一個公式

從台北到高雄要多久？

0.5小時

4小時

0.5小時

如果改坐飛機，
台北到高雄只要1小時
全程可以加快多少？

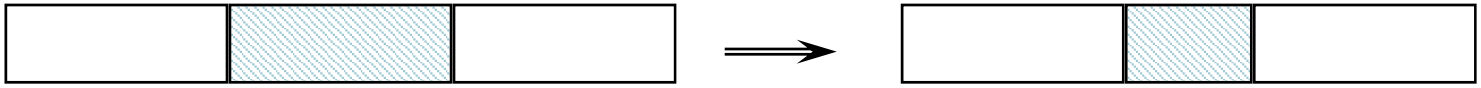
由台北到高雄

- 不能enhance的部份為在市區的時間: 0.5 + 0.5 = 1小時
- 可以enhance的部份為在高速公路上的4小時
  => 佔總時間的 4/(4+1) = 0.8 = F
- 現在改用飛機, 可以enhance的部份縮短為1小時
  => **S = 4/1 = 4**
- $$\text{speedup} = \frac{\text{走高速公路所需時間}}{\text{坐飛機所需時間}} = \frac{4+1}{1+1} = 2.5$$
- 另一種算法 (Amdahl's Law):
  $$\text{speedup} = \frac{1}{((1-0.8)+0.8/4)} = \frac{1}{(1-0.8)+0.8/4}$$
- **When S -> ∞, speedup -> 5**

Amdahl's Law

- Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{Execution Time without E}}{\text{Execution Time with E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



- Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected then,

$$\text{Execution Time(w/ E)} = \left((1-F) + \frac{F}{S}\right) \times \text{Execution Time(w/o E)}$$

$$\text{Speedup(w/ E)} = \frac{1}{(1-F) + \frac{F}{S}} \overset{S \to \infty}{\approx} \frac{1}{1-F}$$

Pitfall: Amdahl's Law

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

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

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

$$20 = \frac{80}{n} + 20$$

  - 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

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

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

  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions
  - 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