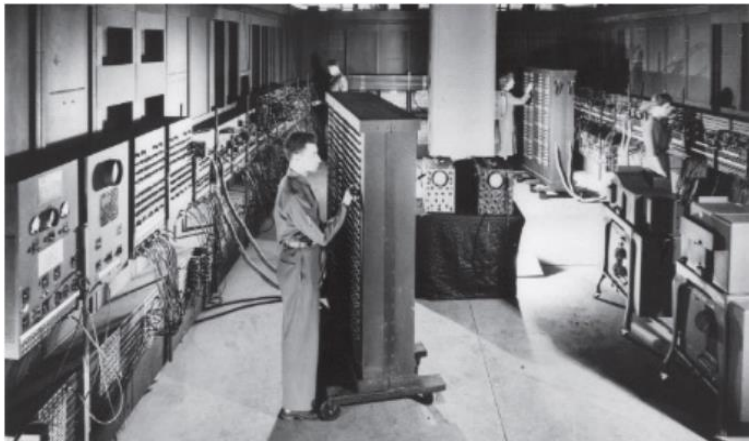
Chapter 1

Computer Abstractions and Technology

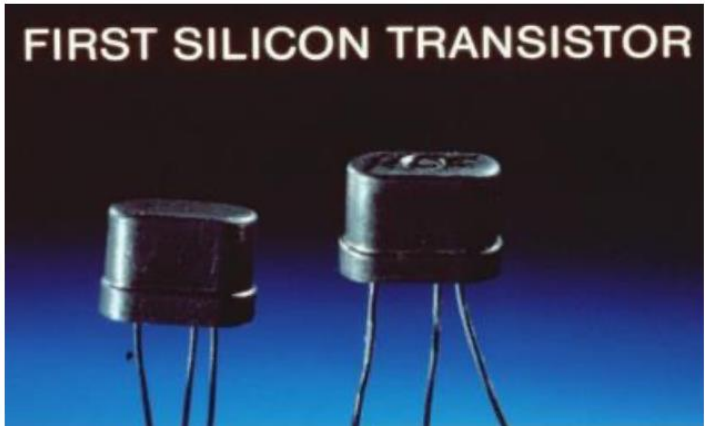
The Computer Revolution



- **First Modern Computer**
  - ENIAC
    (Electronic Numerical Integrator and Calculator)
  - Widely accepted to be the world's first general-purpose computer
  - Work started in 1943, operational during World War II, publicly disclosed after 1946
  - 80 feet long by 8.5 feet high, several feet wide
  - 20 10-digit registers, each 2 feet long
  - Programming: Manually by plugging cables and setting switches
  - Performance: 1900 additions/sec
  - 18,000 Vacuum tubes as electronic switches
- First generation computers: vacuum tubes
  (1940-1956)

The Computer Revolution

- Transistors were Invented almost at the same time with the first computer disclosed
- Bell Lab. (1947) : W. Shockley, J. Bardeen, W. Brattain
  - More reliable, smaller, faster than vacuum tubes
  - Electronic switches in “solids”

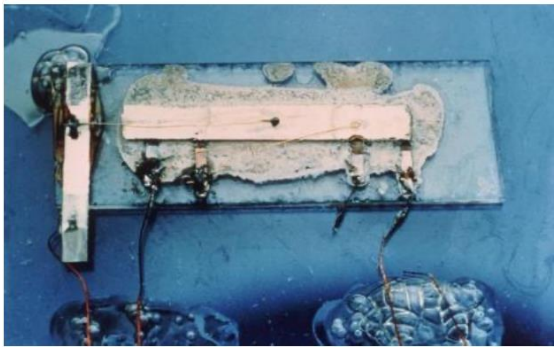


The Computer Revolution

- Second generation computers:
  - Transistors, 1956-1963
- First operational transistor computer
  - *Transistor Computer* by Univ. of Manchester, operational in Nov. 1953
- Commercial transistor computers appear soon after
  - e.g., IBM1401 (1959)

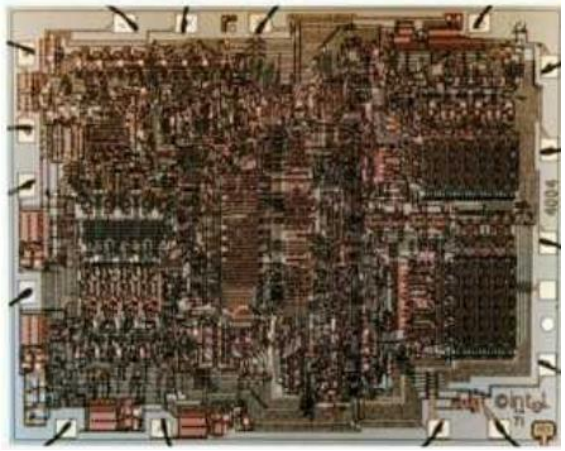
The Computer Revolution

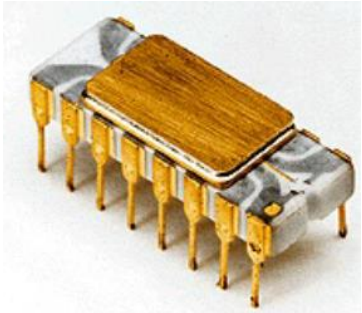
- Another technology breakthrough: IC
  - Jack Kilby of Texas Instrument (1958)
  - Integrated a transistor with resistors and capacitors on a single semiconductor chip, called **IC (Integrated Circuit)**
  - Fast on-chip signal communication, smaller size, more reliable, easier to implement, cheaper
- Third generation computers: IC (1964~1971)
- **Moore's law** comes into play
  - 2x transistors/chip per 1.5~2 years
  - Shrinking transistors



The Computer Revolution

- As IC chips scale …
  - put an entire processor onto a chip
    - Intel 4004 (1971)
    - First microprocessor
    - 2250 transistors
    - 108 KHz, 0.06 MIPS
    - 8-bit, 16-bit, 32-bit microprocessors soon appeared
      - Low-cost, high-volume personal computers
      - CISC (Complex Instruction Set computer) vs. RISC (Reduced Instruction Set computer)
      - Pipelining, cache, ….
  - Fourth generation computers
    - microprocessor, 1971-present

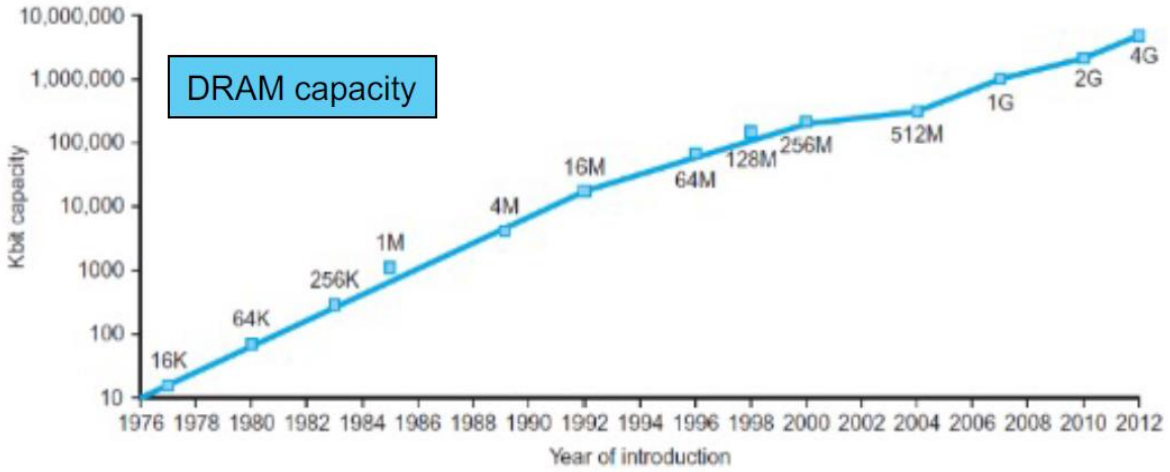




The Computer Revolution

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

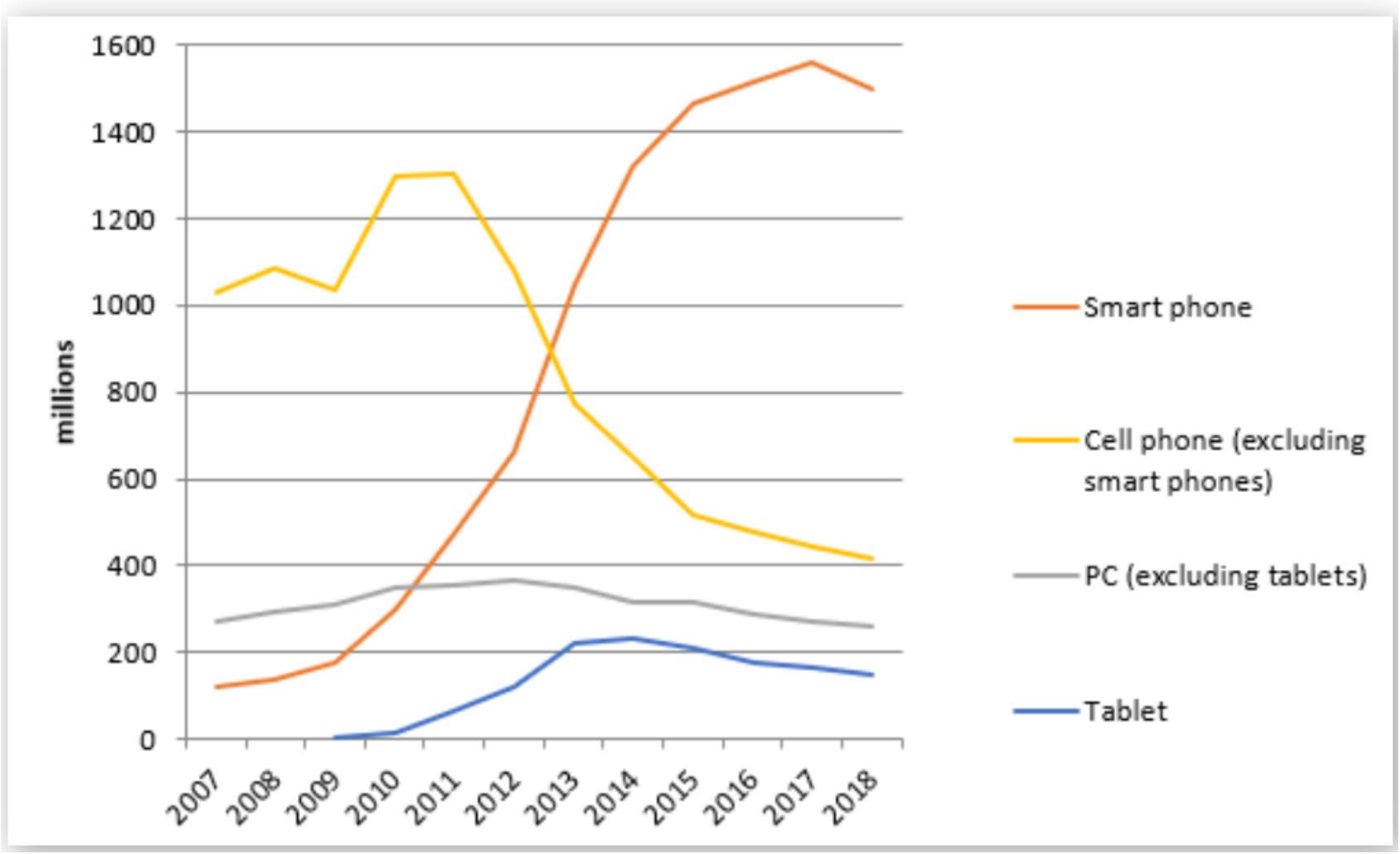
- Technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



Classes of Computers

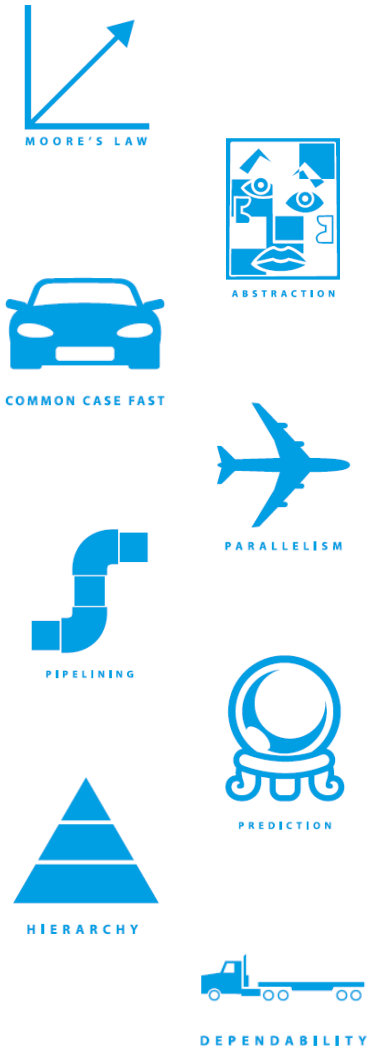
- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Embedded computers
  - Hidden as components of systems
    - Cell phone, drone, self-driving car, VR game
  - Stringent power/performance/cost constraints

Processor Market - The PostPC Era

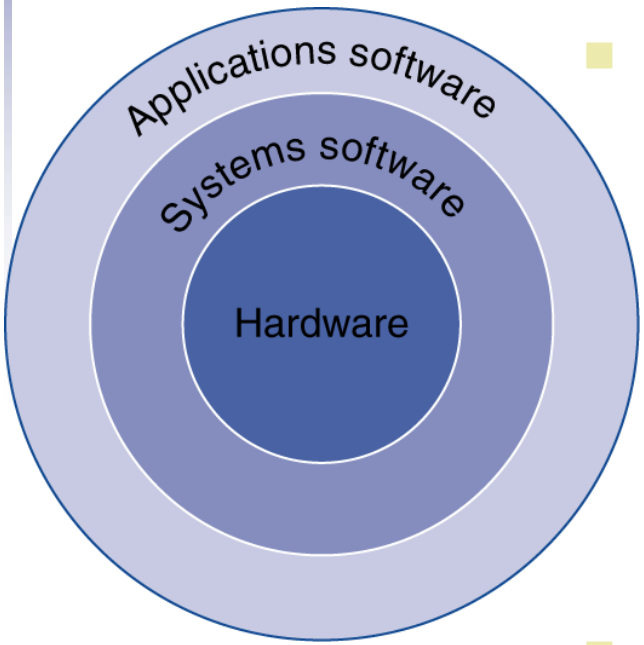


Eight Great Ideas

- 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



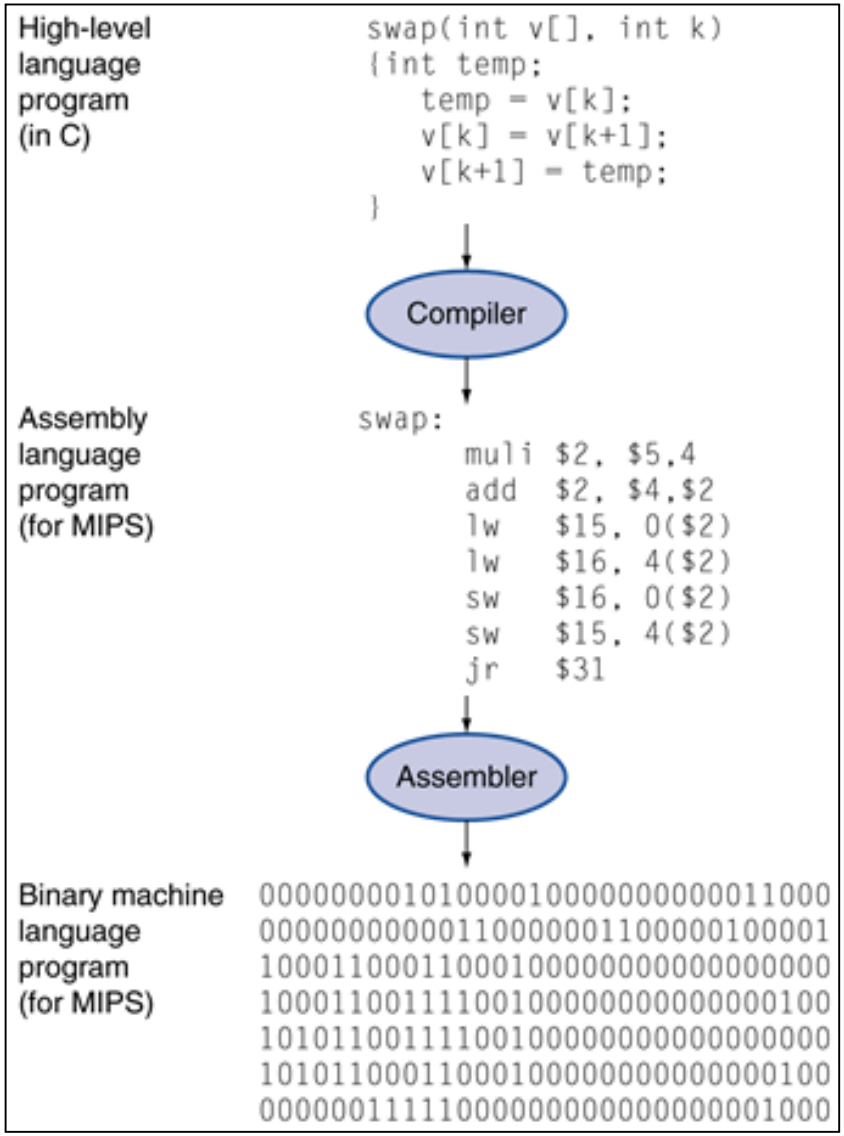
Below Your Program



- 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

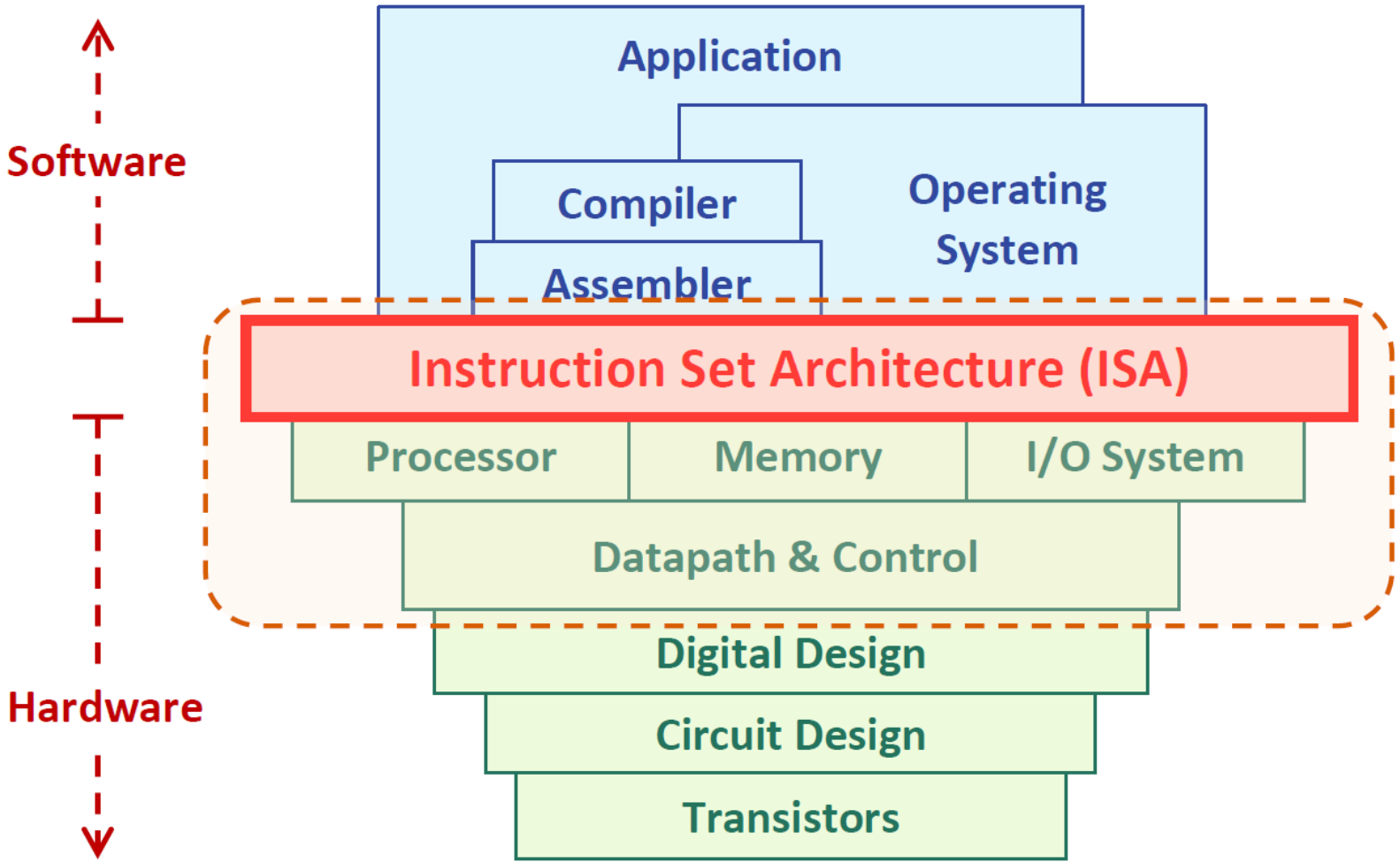


Abstractions

- Abstraction helps us deal with complexity
  - Hide lower-level detail

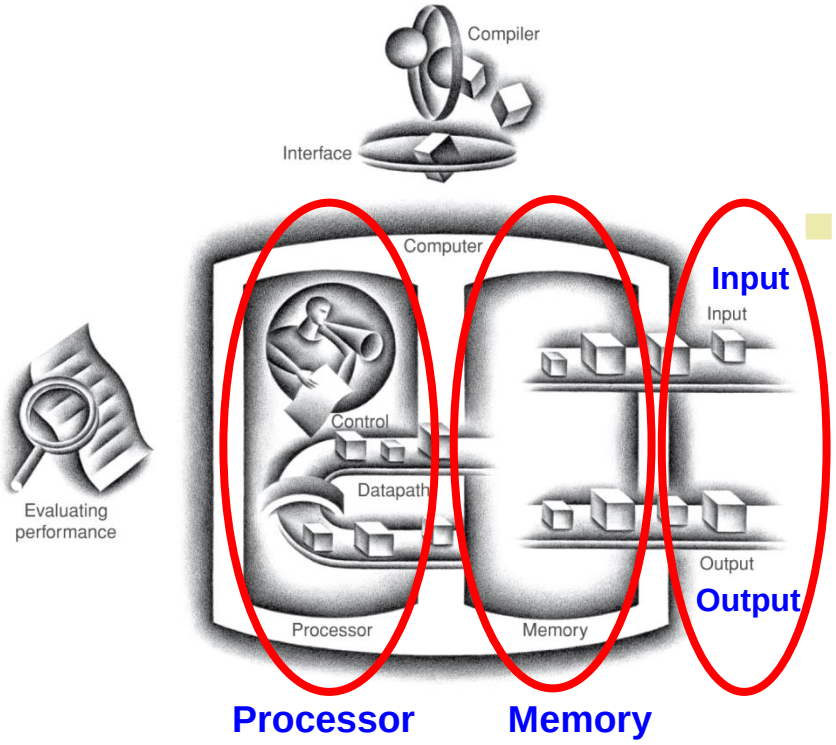
- Instruction set architecture (ISA)
  - The hardware/software interface
  - Instruction set, register set, addressing mode, interrupt, etc.

Instruction Set Architecture (ISA)



Components of a Computer

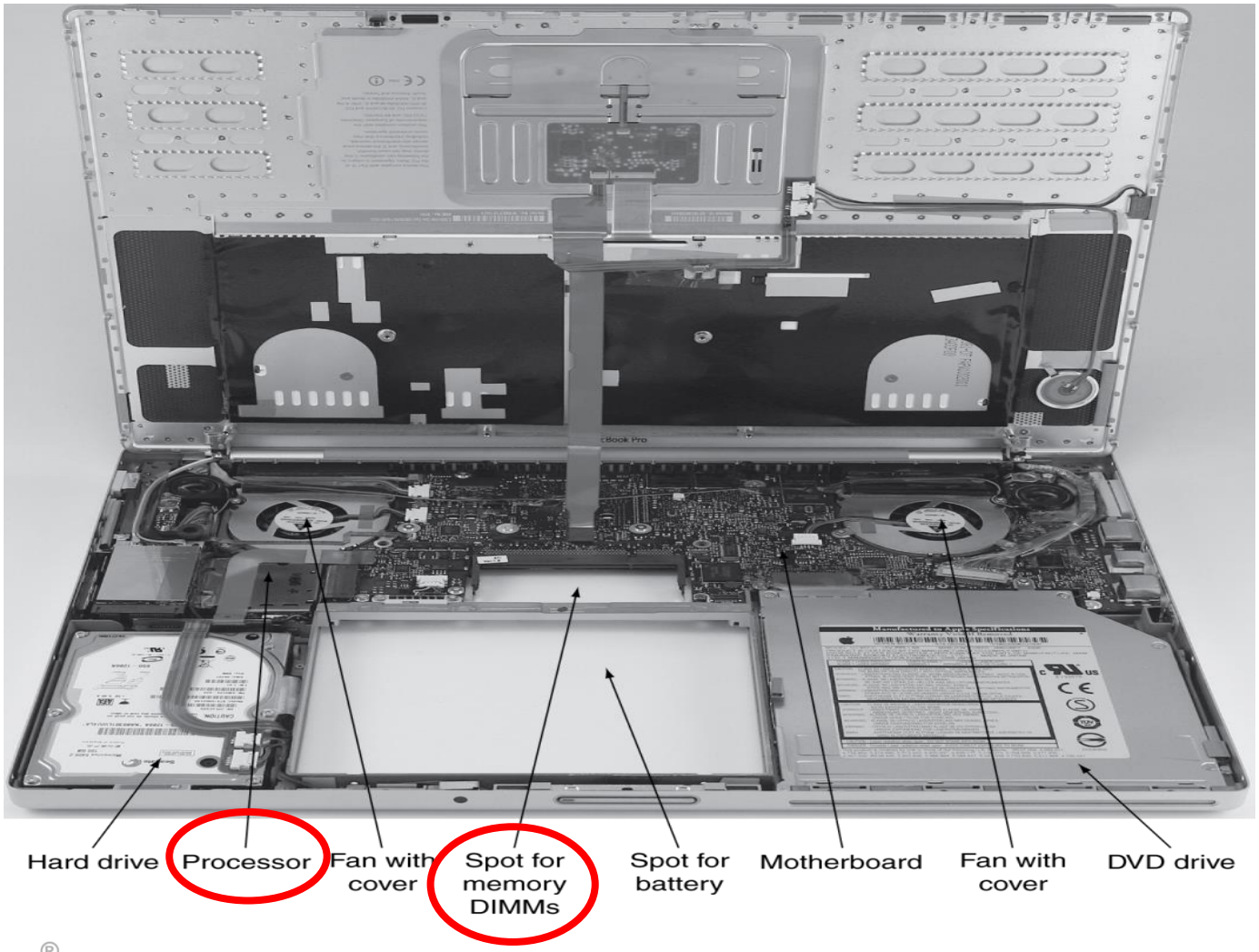
The BIG Picture



- 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

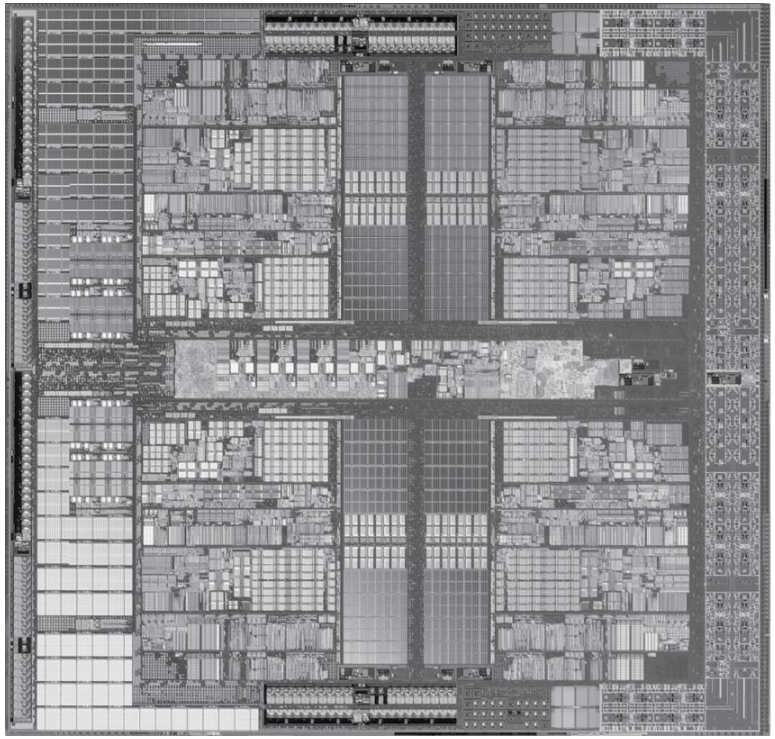
Anatomy of a Computer

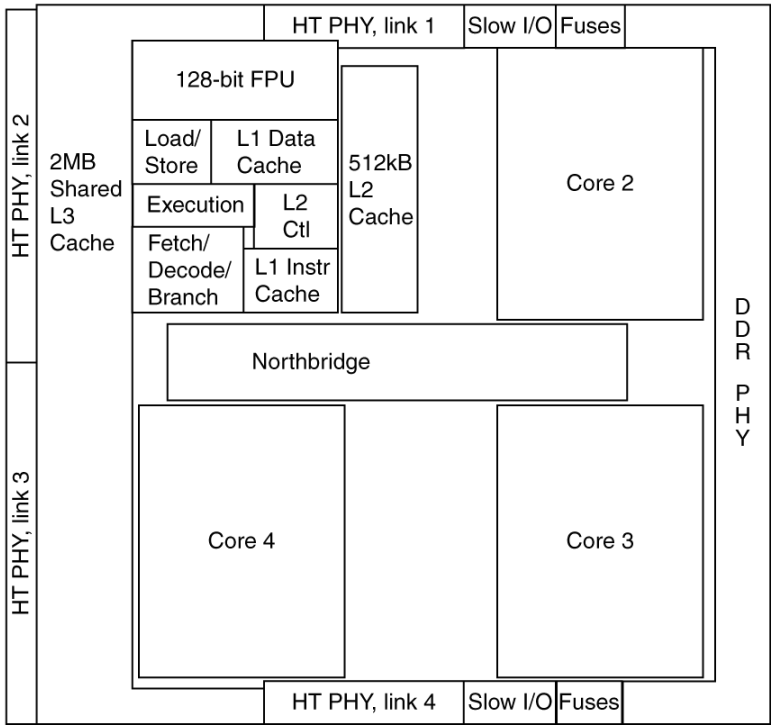
Opening the Box



Inside the Processor (CPU)

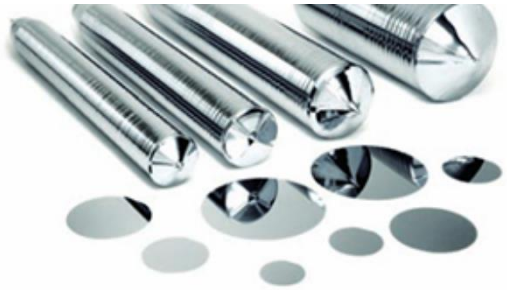
- AMD Barcelona: 4 processor cores

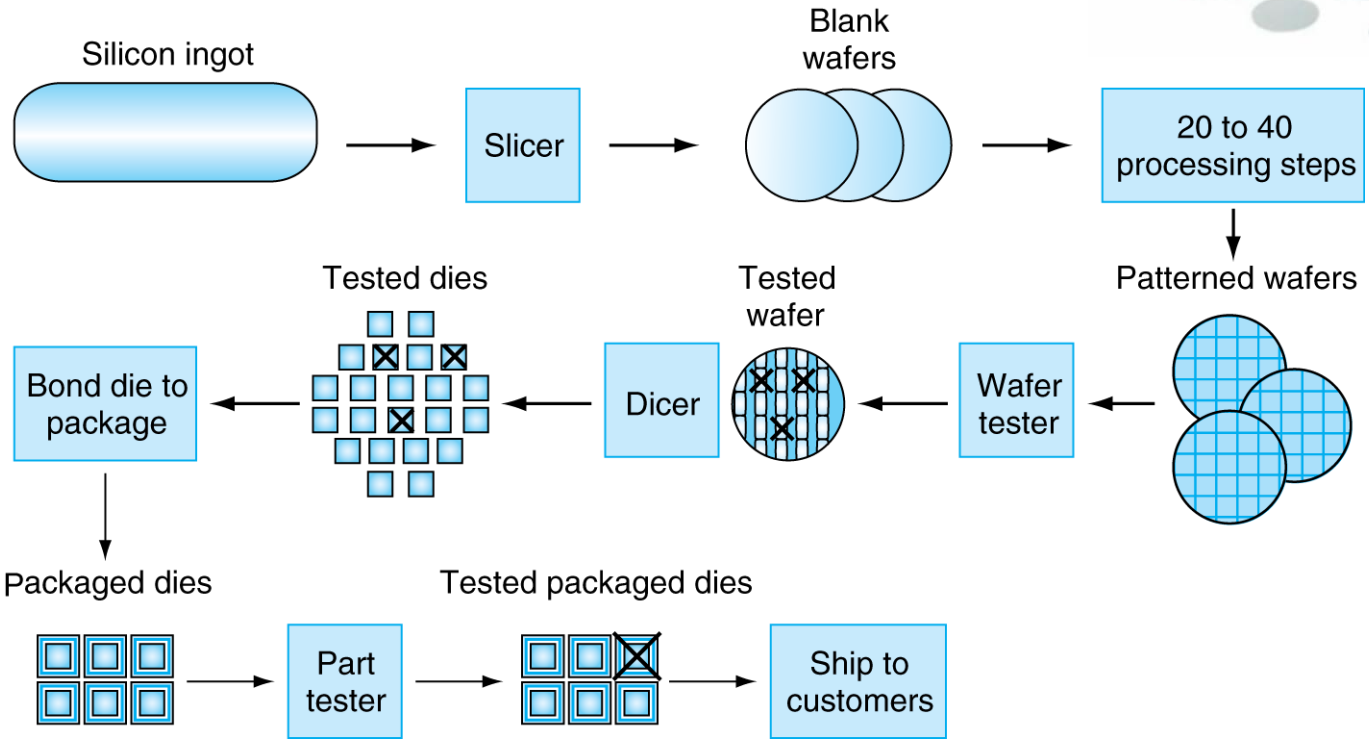




Cache memory : Small fast SRAM memory for immediate access to data

Manufacturing ICs



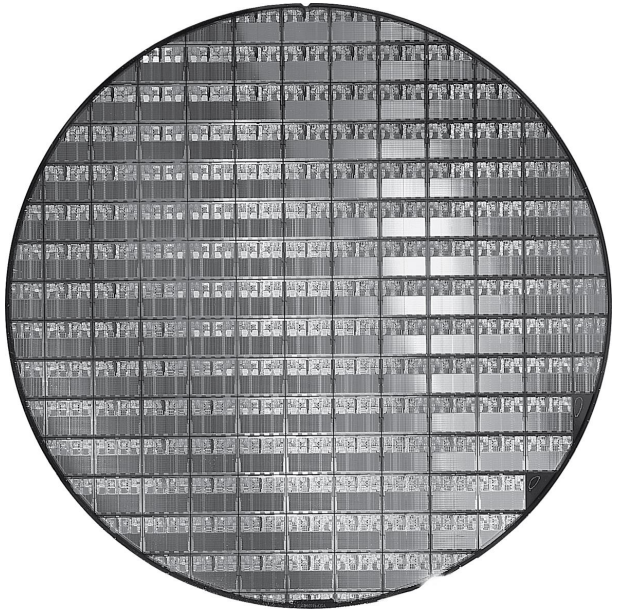


- Yield: proportion of working dies per wafer

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

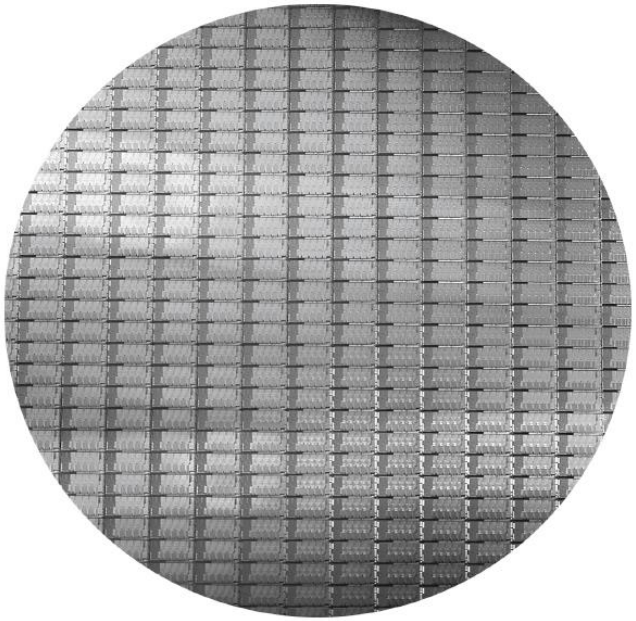
AMD Opteron X2 Wafer

AMD Opteron X2 Wafer



- X2: 12-inch wafer, 117 chips, 90nm technology

Intel Core i7 Wafer



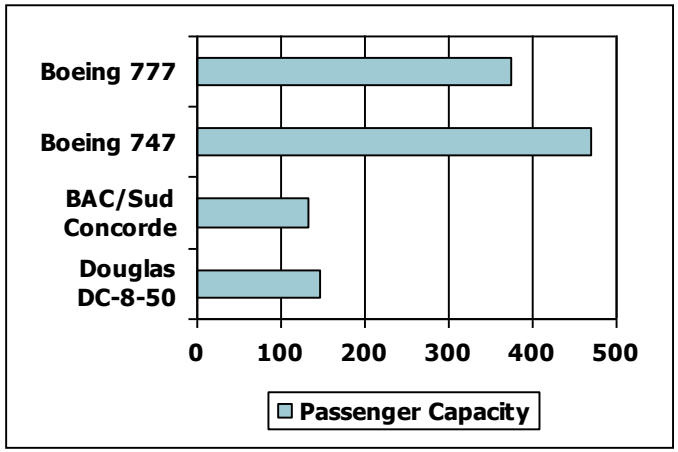
- 12-inch wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

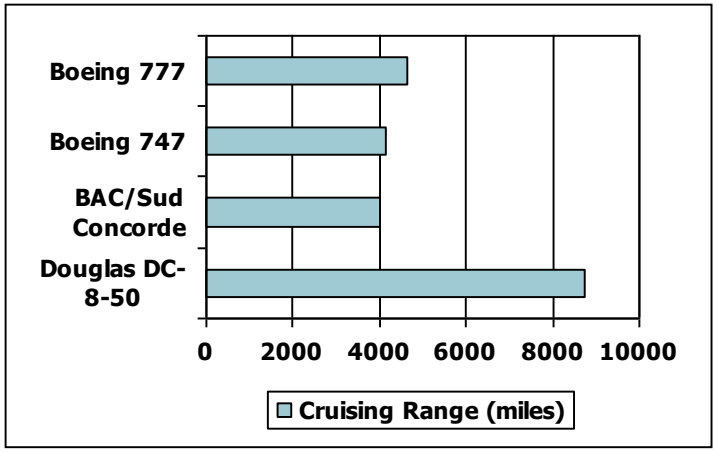
Understanding Performance

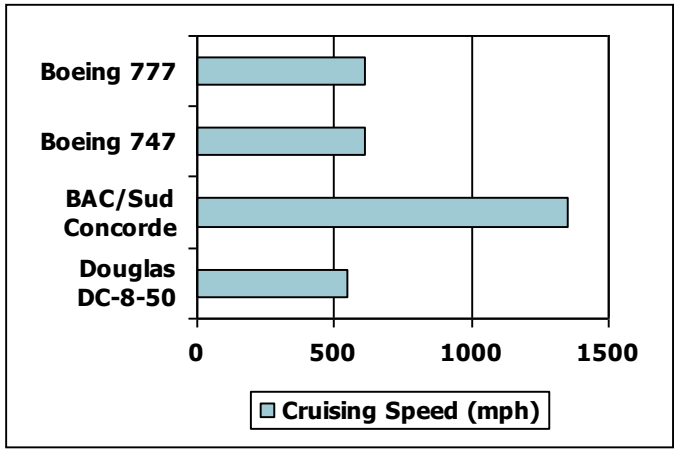
- Algorithm
  - Determines *number of operations* executed
- Programming language, compiler
  - Determine *number of machine instructions* for each source-level statement
- Processor and memory system
  - Determine *how fast machine instructions are executed*
- I/O system (including OS)
  - Determines *how fast I/O operations* are executed

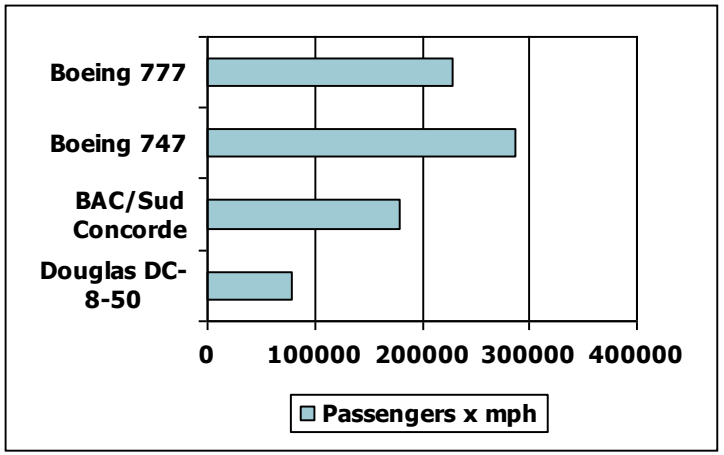
Defining Performance

- Which airplane has the best performance?









CPU performance

- 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

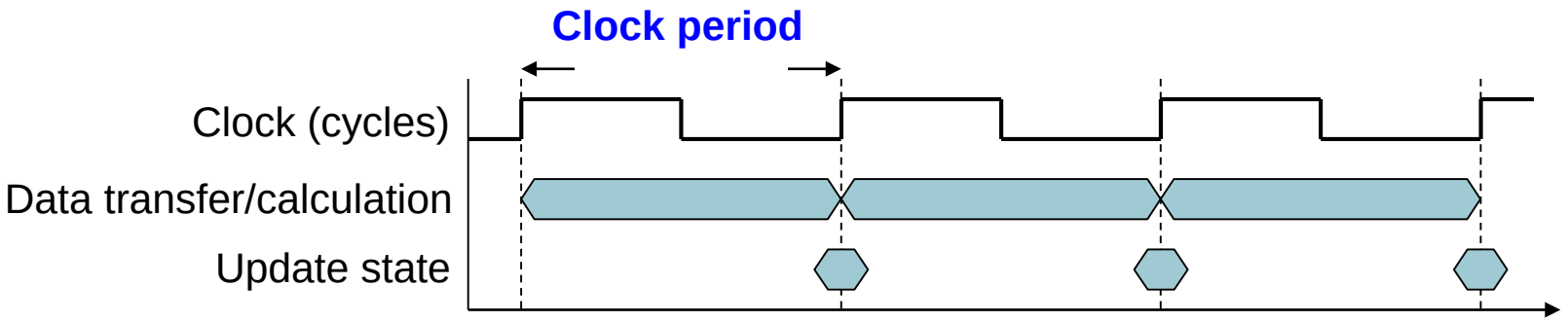
Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - CPU processing, I/O, OS overhead, idle time
  - Determines system performance

- CPU time ← The time we focus
  - Comprises *user CPU time* and *system CPU time*
  - We focus on user CPU time which means the CPU time spent in the code of our application.

CPU Clocking

- Digital hardware governed by a constant-rate clock



reciprocal

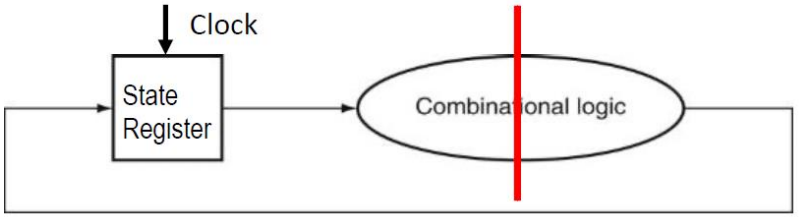
- **Clock period** (**cycle time**): duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Note: **s** **ms** **(10^{-3})** **us(10^{-6})** **ns(10^{-9})** **ps(10^{-12})**
- **Clock rate** (**frequency**) : cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz
  - Note: **Hz** **KHz** **(10^{3})** **MHz(10^{6})** **GHz(10^{9})** **THz (10^{12})**

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

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

- 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? (i.e., clock rate?)

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

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

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

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 (IC) for a program
  - Determined by program (language/algorithm), compiler, ISA
- Average cycles per instruction (CPI)
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix
    - A “multiply” takes more clocks than an “add”

```
swap:
        multi  $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
```

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- A, B : Same ISA ⟹ **they mean "same instr. count"**
- Same program running on A and B, 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
  - E.g., add/sub, lw/sw, multi, jump
    class A, class B, class C, class D

```
swap:
        multi  $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
```

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

- Weighted average CPI

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

Relative frequency of i^{th} class of instr. in the program

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	2M	1M	2M
IC in sequence 2	4M	1M	1M

- Sequence 1: IC = 5M
  - Clock Cycles
    = 2M×1+1M×2+2M×3
    = 10M
  - Avg. CPI = 10M/5M = 2
- Sequence 2: IC = 6M
  - Clock Cycles
    = 4M×1+1M×2+1M×3
    = 9M
  - Avg. CPI = 9M/6M = 1.5

Performance Summary

- Computer performance of a program is

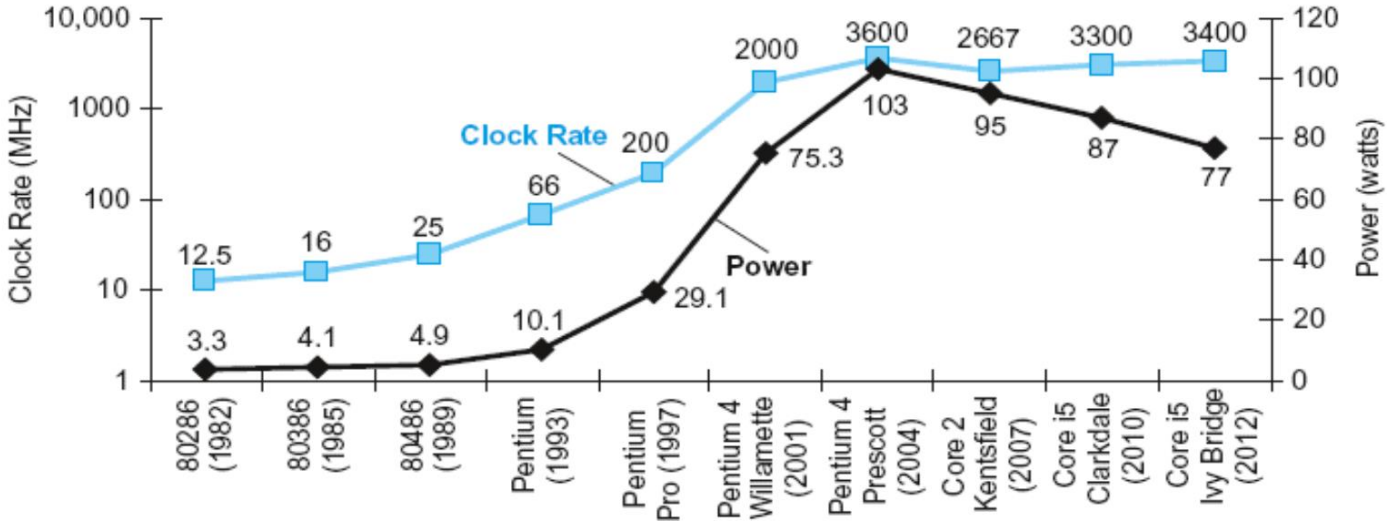
IC **CPI** **T**cycle **(=1/CR)**

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

- Performance depends on

	Instr. Count	CPI	Clock Rate
Algorithm	√	√	
Program Language	√	√	
Compiler	√	√	
ISA	√	√	√

Power Trends



- Power consumption goes with clock rate
- Slowing down after 2004 because of “power wall”

- In CMOS IC technology

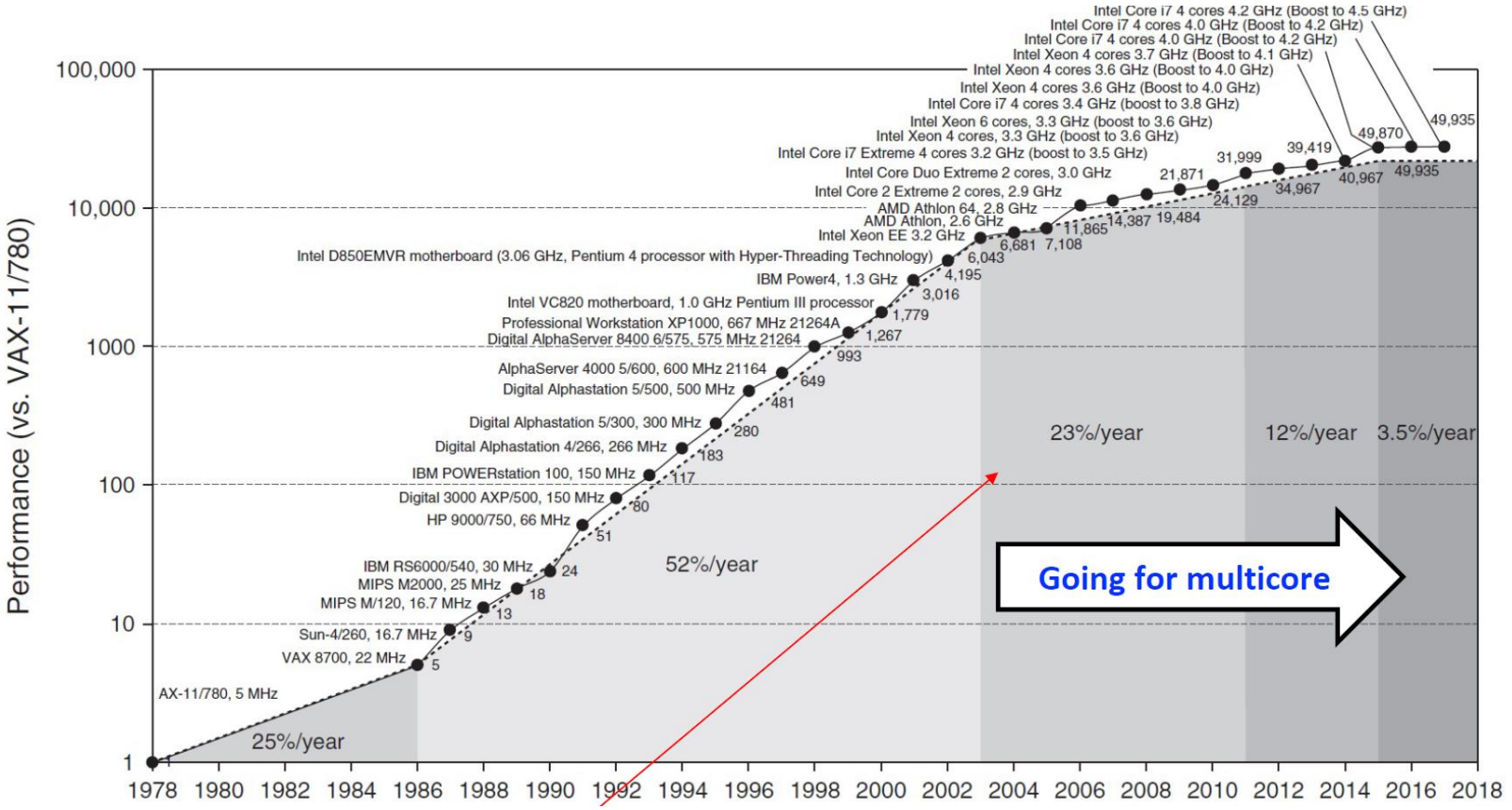
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

5V → 1V

Reducing Power

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

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

➔ **Turn to design multiple processors (cores)**

Moving to Multiprocessors

- Free lunch for software in uniprocessor
  - IC technology (e.g.,clock rate) automatically improve performance
  - Instruction-level parallelism (pipeline)
    - Hardware executes multiple instructions at once
    - Hidden from the programmer (In the past, programmers could rely on hardware to double the performance every 18 months without having to change a line of code)

- Multicore microprocessors
  - More than one processor per chip
  - Requires explicitly parallel programming
    - Rewrite programs for parallelism
    - Load balancing
    - Optimizing communication and synchronization

No more free lunch for software in multiprocessor

Performance with Benchmarks

- Recall that "X is *n* time faster than Y" means

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

  - It is easy to compare two computers using one program, but what about multiple programs?

- ***Benchmarking***
  - A standard set of programs specifically chosen to measure and compare computer performance
  - Benchmark represents a workload that will predict the performance of the actual workload

How to summarize performance

	Machine A	Machine B
Program 1	2 s	4 s
Program 2	12 s	8 s

- Ratio of total execution time
  - Find total execution time, then find performance ratio
    - Total execution time of machine A: (2+12)
    - Total execution time of machine B: (4+8)
    - Performance of machine B relative to A
      - (2+12) / (4+8) = 1.17
    - → B is 1.17 times faster than A
  - But, machine A runs program 1 twice faster than B
  - Programs are not equally important

→ Not good!

How to summarize performance

	Machine A	Machine B
Program 1	2 s	4 s
Program 2	12 s	8 s

- Arithmetic Mean of execution time ratio
  - Find performance ratio first, then average them
    - Performance of machine A relative to B
      - (4/2 + 8/12) /2 = 4/3 → A is 1.33 times faster than B
    - Performance of machine B relative to A
      - (2/4 + 12/8) /2 = 1 → A is the same as B
  - Different conclusion if different references are used!
  - **Not good!**

How to summarize performance

	Machine A	Machine B
Program 1	2 s	4 s
Program 2	12 s	8 s

- **Geometric Mean** of execution time ratio

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

  - Consistent results regardless of reference → **Good!**

	Machine A (B as reference)	Machine B (A as reference)
Program 1	4/2 = 2.0	2/4 = 0.5
Program 2	8/12 = 0.667	12/8 = 1.5
Geometric Mean	$(2*0.667)^{1/2} = 1.155$	$(0.5*1.5)^{1/2} = 0.866$
	A : B = 1.155: 1	A : B = 1 : 0.866

SPEC CPU Benchmark

- Standard Performance Evaluation Corp (SPEC)
  - Began in 1989 on benchmarking workstation and server
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2017 (43 benchmarks)
  - Elapsed time to execute a selection of programs
  - Normalize relative to a reference machine
  - Summarize as **geometric mean** of performance ratios

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

SPEC CPU2017 Integer Benchmarks

SPECrate 2017	SPECspeed 2017	Lang.	KLOC	Application Area
500.perlbench_r	600.perlbench_s	C	362	Perl interpreter
502.gcc_r	602.gcc_s	C	1,304	GNU C compiler
505.mcf_r	605.mcf_s	C	3	Route planning
520.omnetpp_r	620.omnetpp_s	C++	134	Discrete Event simulation - computer network
523.xalancbmk_r	623.xalancbmk_s	C++	520	XML to HTML conversion via XSLT
525.x264_r	625.x264_s	C	96	Video compression
531.deepsjeng_r	631.deepsjeng_s	C++	10	Artificial Intelligence: alpha-beta tree search (Chess)
541.leela_r	641.leela_s	C++	21	Artificial Intelligence: Monte Carlo tree search (Go)
548.exchange2_r	648.exchange2_s	Fortran	1	Artificial Intelligence: recursive solution generator (Sudoku)
557.xz_r	657.xz_s	C	33	General data compression

KLOC: line count for source files used in a build / 1000

SPEC CPU2017 Floating-point Benchmarks

SPECrate 2017	SPECspeed 2017	Lang.	KLOC	Application Area
503.bwaves_r	603.bwaves_s	Fortran	1	Explosion modeling
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	Physics: relativity
508.namd_r		C++	8	Molecular dynamics
510.parest_r		C++	427	Biomedical imaging
511.povray_r		C++, C	170	Ray tracing
519.lbm_r	619.lbm_s	C	1	Fluid dynamics
521.wrf_r	621.wrf_s	Fortran, C	991	Weather forecasting
526.blender_r		C++, C	1,577	3D rendering & animation
527.cam4_r	627.cam4_s	Fortran, C	407	Atmosphere modeling
	628.pop2_s	Fortran, C	338	Wide-scale ocean modeling
538.imagick_r	638.imagick_s	C	259	Image manipulation
544.nab_r	644.nab_s	C	24	Molecular dynamics
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	Computational Electromagnetics
554.roms_r	654.roms_s	Fortran	210	Regional ocean modeling

SPECspeed2017 Integer Benchmarks on Intel Xeon E5-2650L

Description	*Name*	*Instruction Count x 10^9*	*CPI*	*Clock cycle time (seconds x 10^–9)*	*Execution Time (seconds)*	*Reference Time (seconds)*	*SPECratio*
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

SPECratio: Ref-time/Exec-time (The bigger the SPECratio, the faster the CPU is).

Amdahl's Law

- **Pitfall**: Improve an aspect of a computer and expect a proportional improvement in overall performance

  ➔ Incorrect

- **Amdahl's Law:**

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

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 4× overall performance?

$$\frac{100}{4} = \frac{80}{n} + 20$$

  - n =16
  - How about 5× ?

$$\frac{100}{5} = \frac{80}{n} + 20$$

  - Can't be done!

Example

- Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 sec, what will the speedup be if half of the 10 sec is spent executing floating-point instructions?

- We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 2. One benchmark we are considering runs for 100 sec with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield desired speedup on this benchmark?

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
      - even with the same ISA, different programs on the same CPU have different MIPS,

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

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

      - 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