Chapter 1

Computer Abstractions and Technology

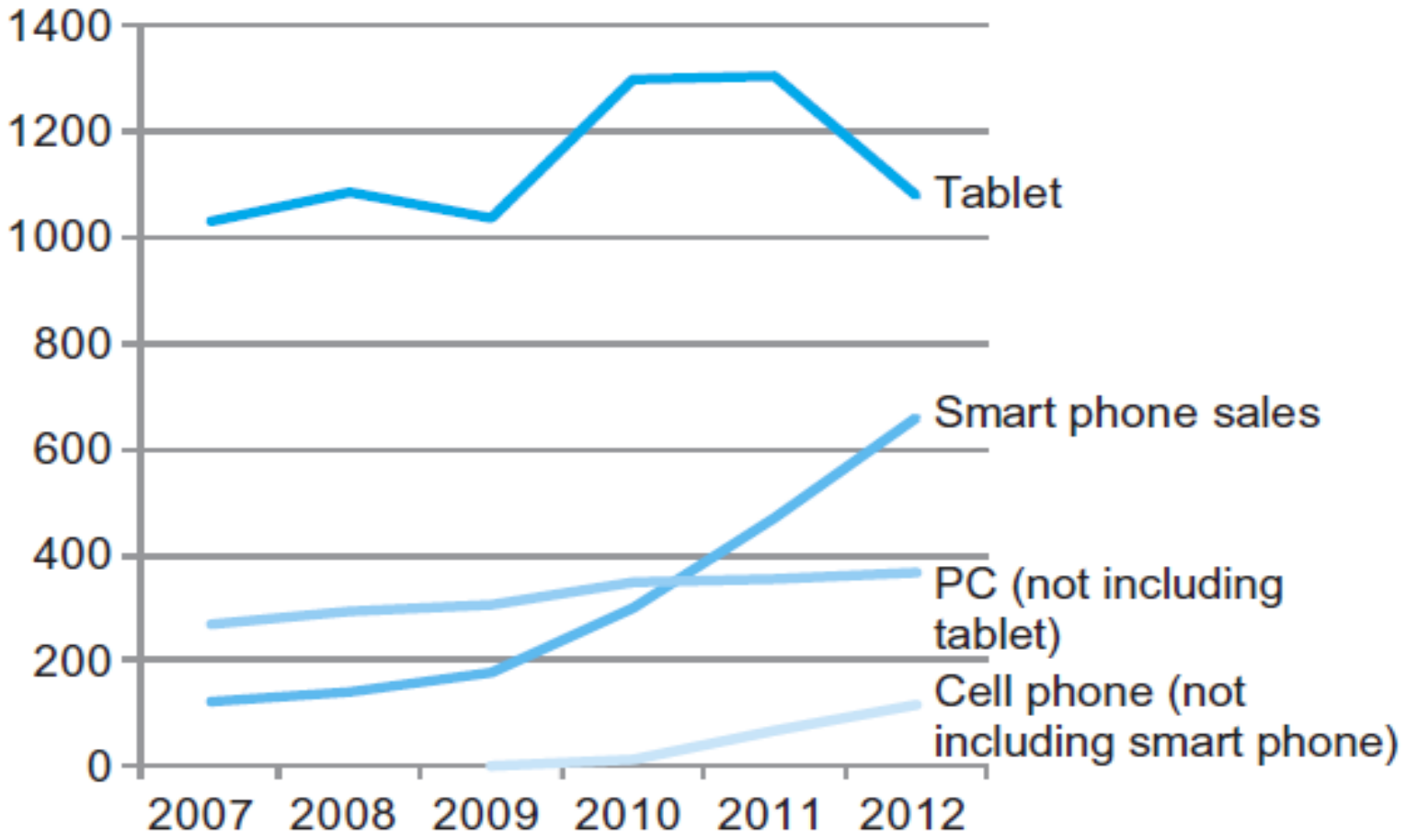
The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
    - The number of transistors in a IC doubles every 18-24 months
- Makes novel applications feasible
  - Cell phones
  - World Wide Web
  - Search Engines
  - Self-driving car, drone
  - VR games
- Computers are pervasive

Classes of Computers

- Desktop 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
  - Stringent power/performance/cost constraints

Processor Market - The PostPC Era

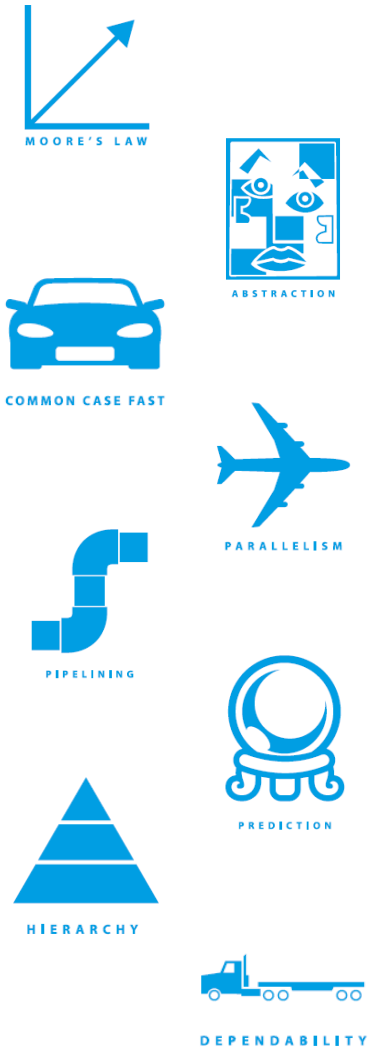


Understanding Performance

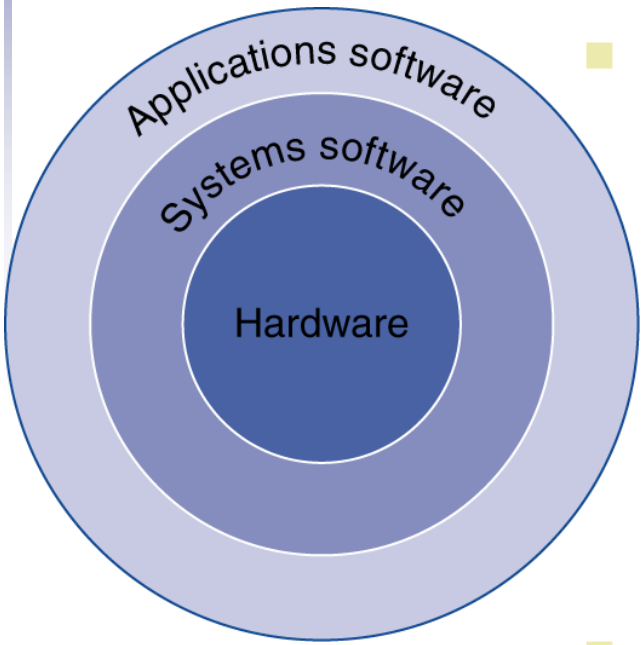
- Algorithm
  - Determines *number of operations* executed
- Programming language, compiler
  - Determine *number of machine instructions* for each source-level statement
- Processor and memory system
  - Determine *how fast machine instructions are executed*
- I/O system (including OS)
  - Determines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

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



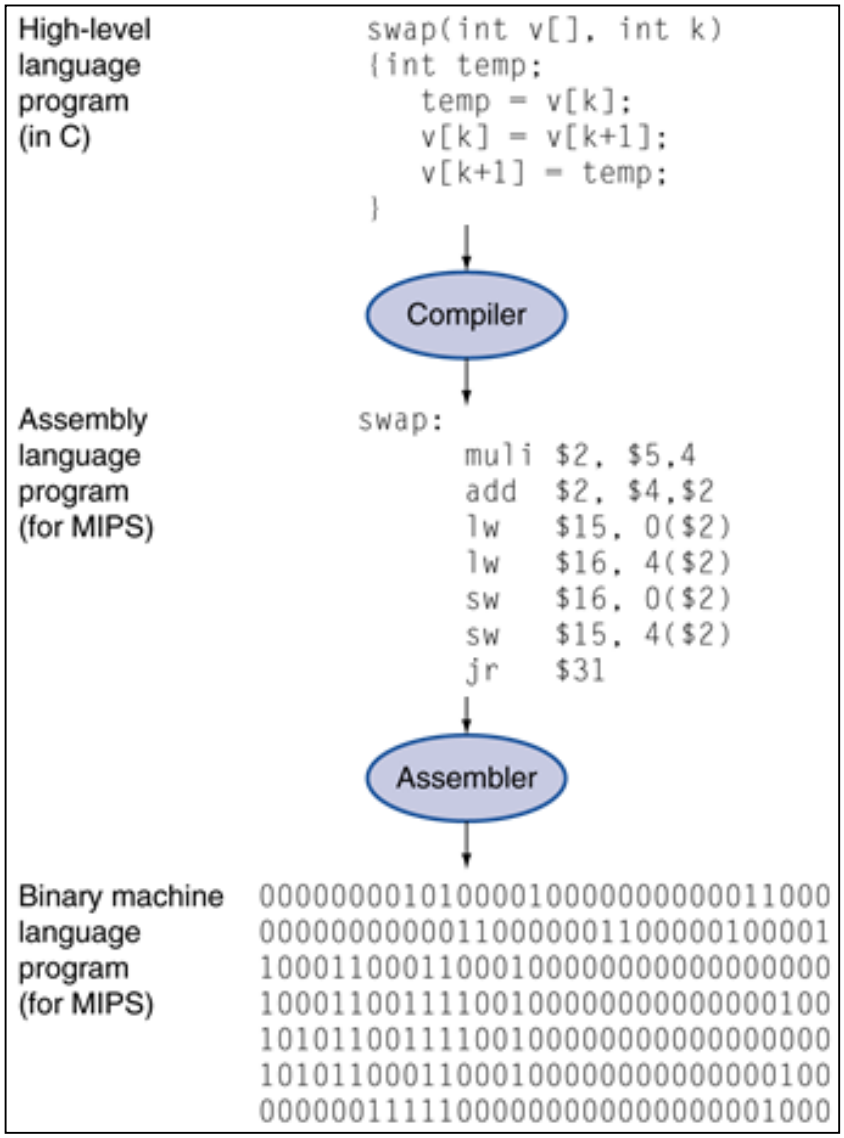
Below Your Program



- 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

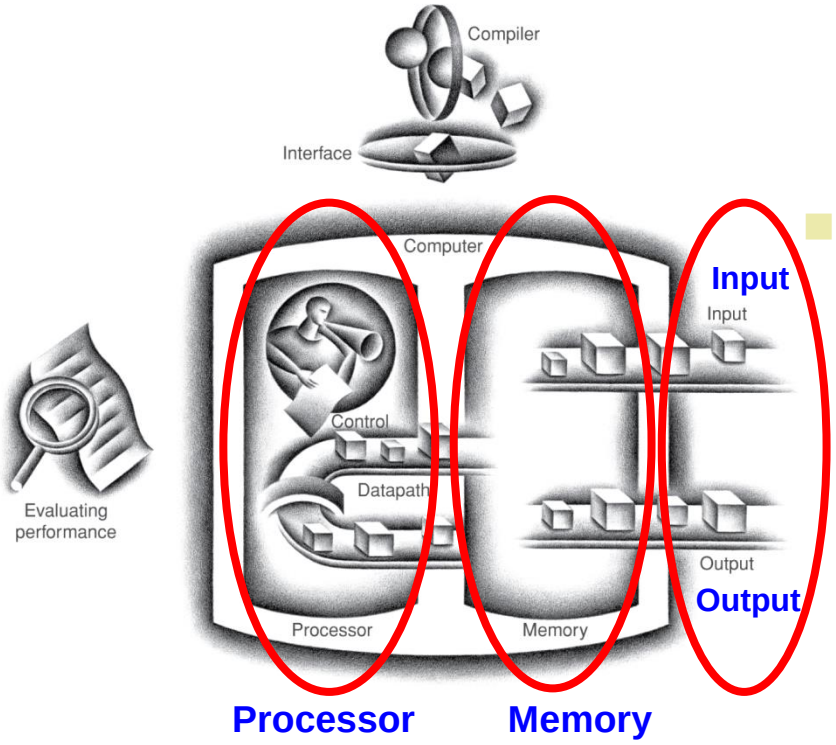
Levels of Program Code

- 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



Components of a Computer

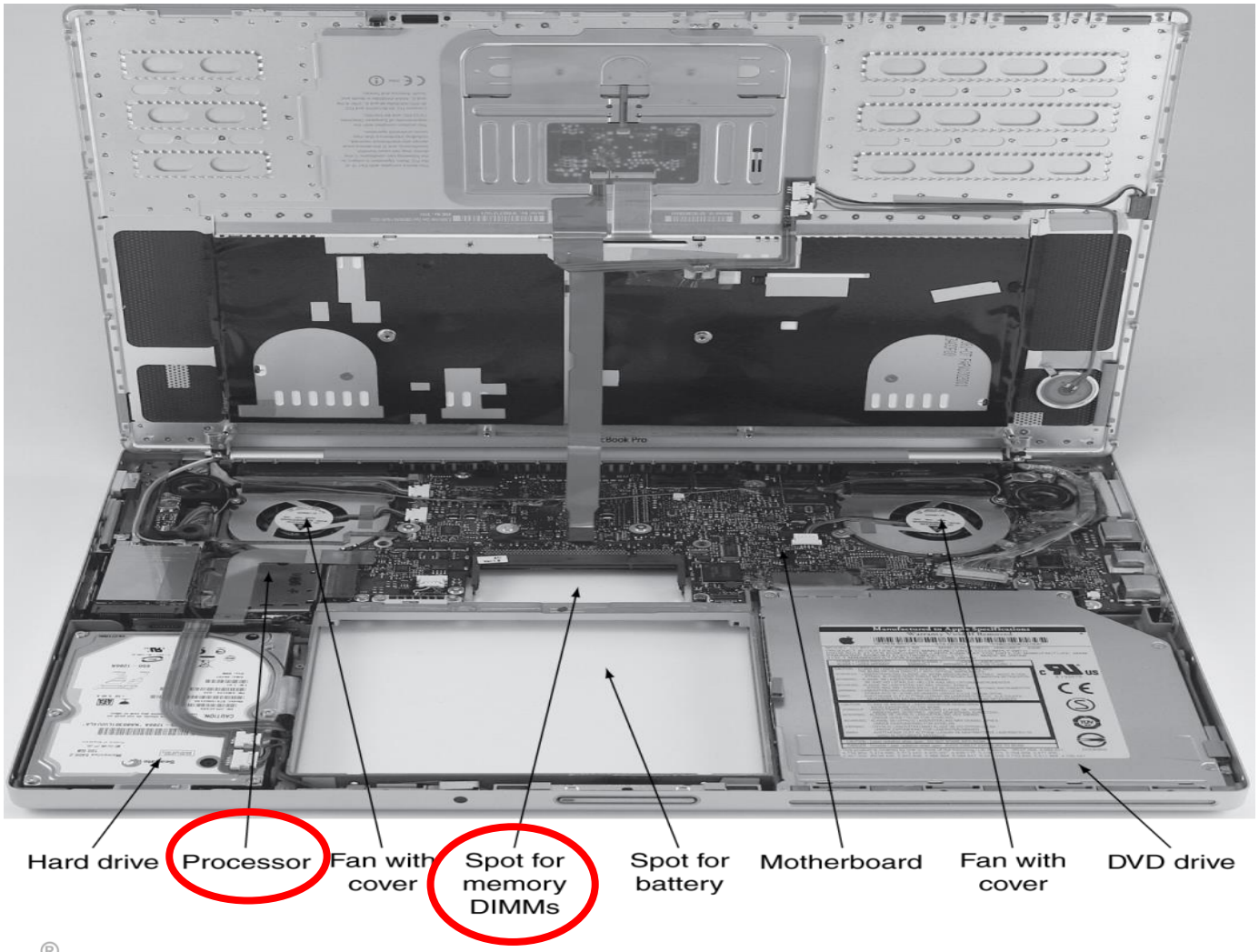
The BIG Picture



- 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

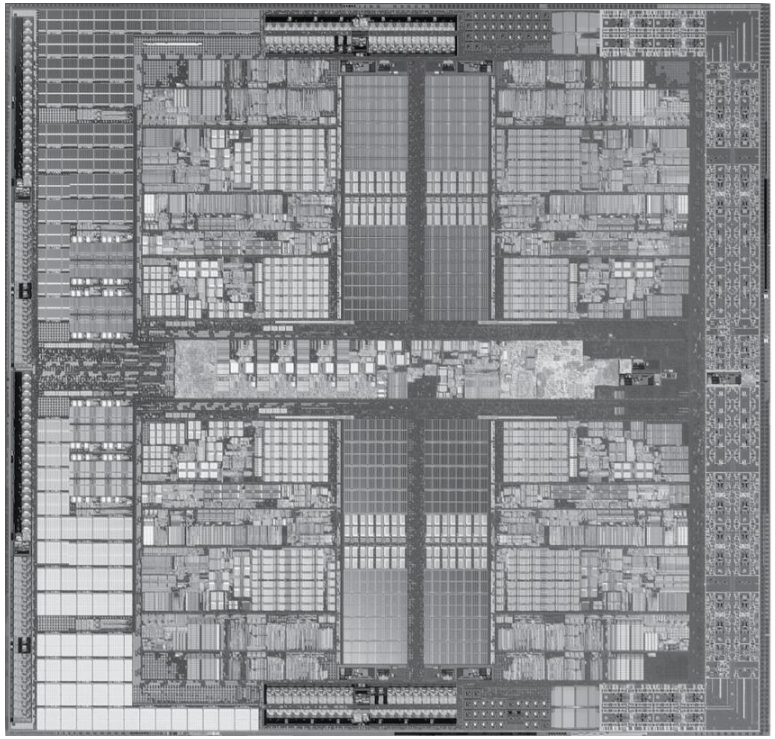
Anatomy of a Computer

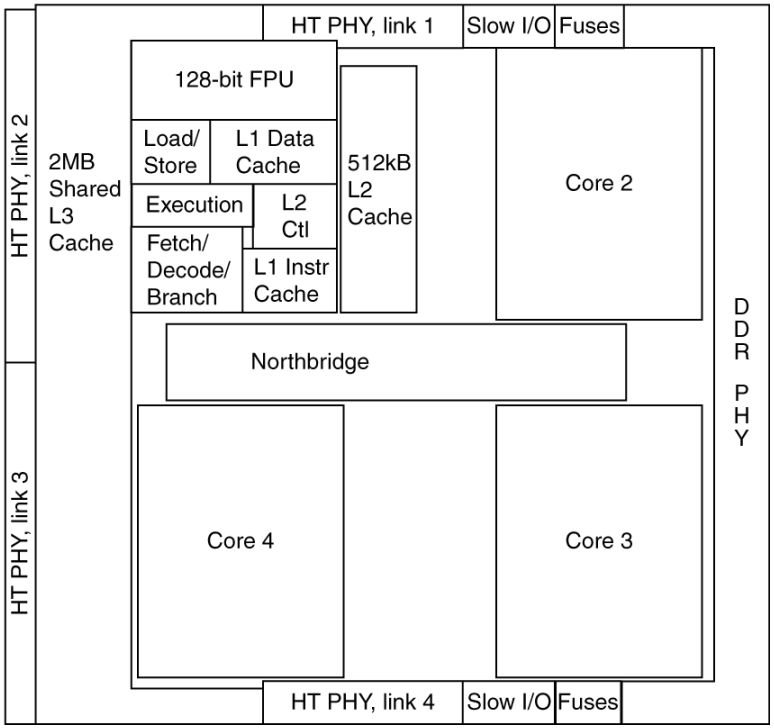
Opening the Box



Inside the Processor (CPU)

- AMD Barcelona: 4 processor cores





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

Abstractions

The BIG Picture

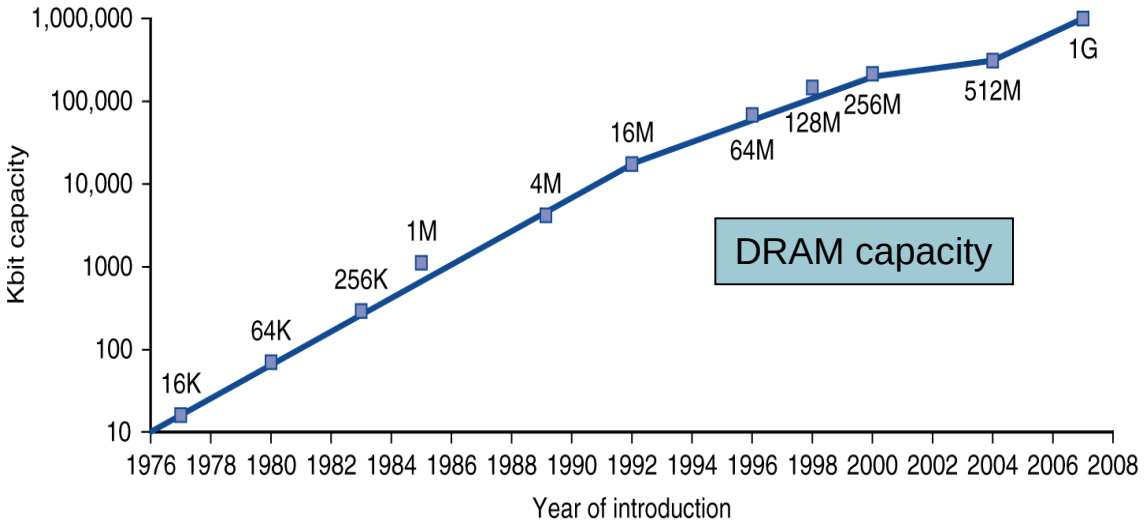
- Abstraction helps us deal with complexity
  - Hide lower-level detail

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

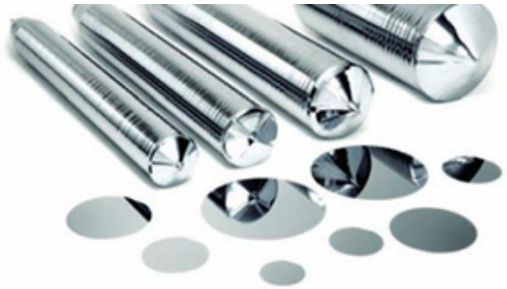
Technology Trend

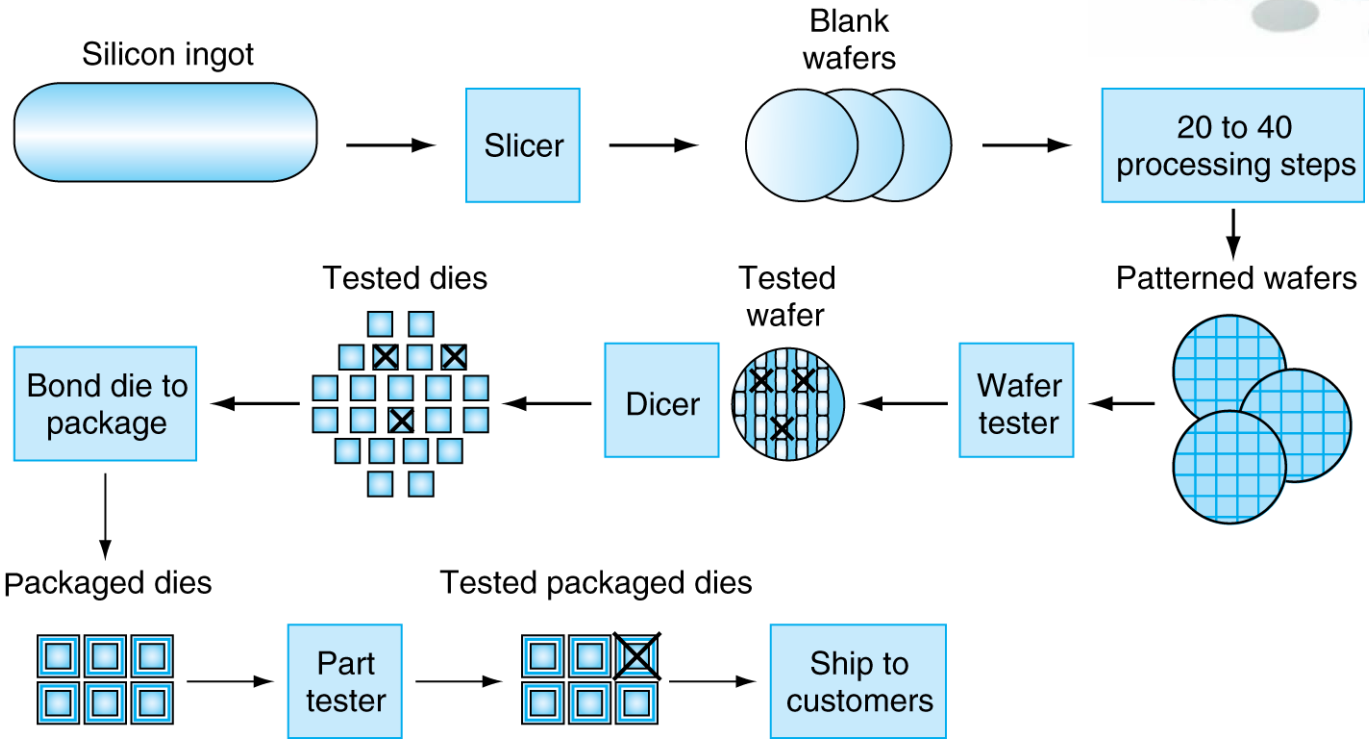
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

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



Manufacturing ICs



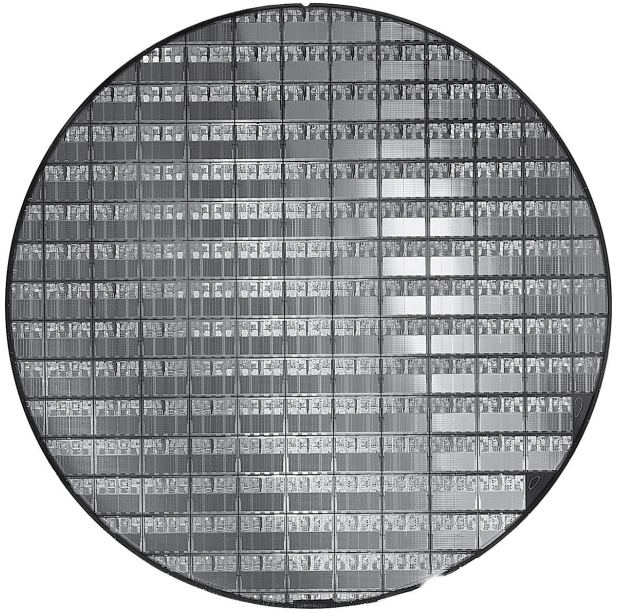


- Yield: proportion of working dies per wafer

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

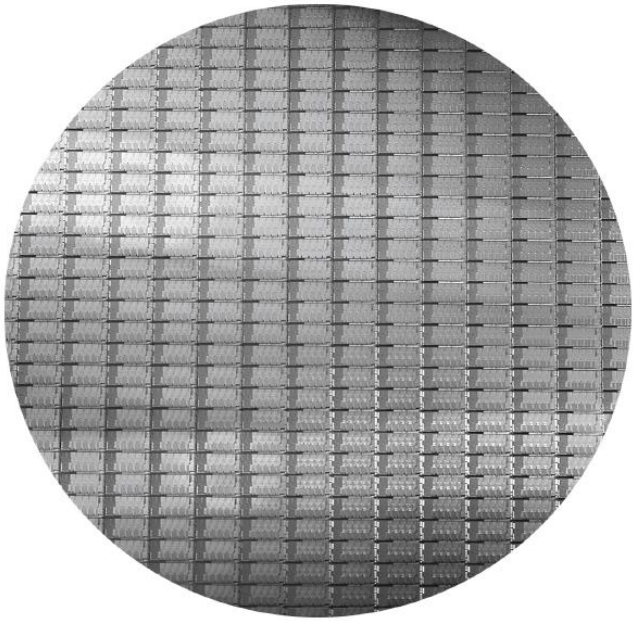
AMD Opteron X2 Wafer

AMD Opteron X2 Wafer



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

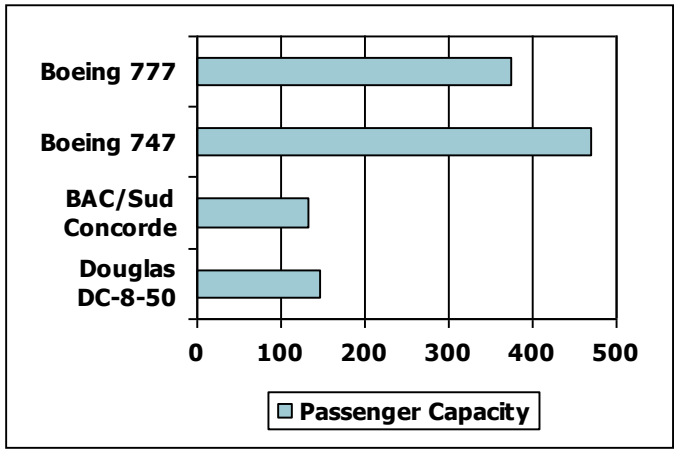
Intel Core i7 Wafer

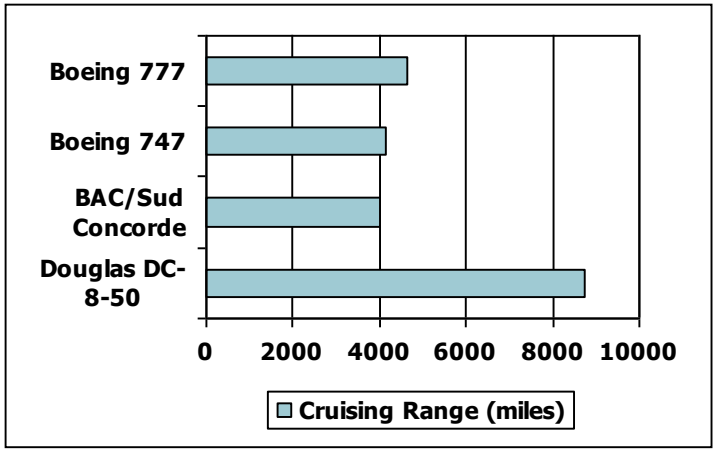


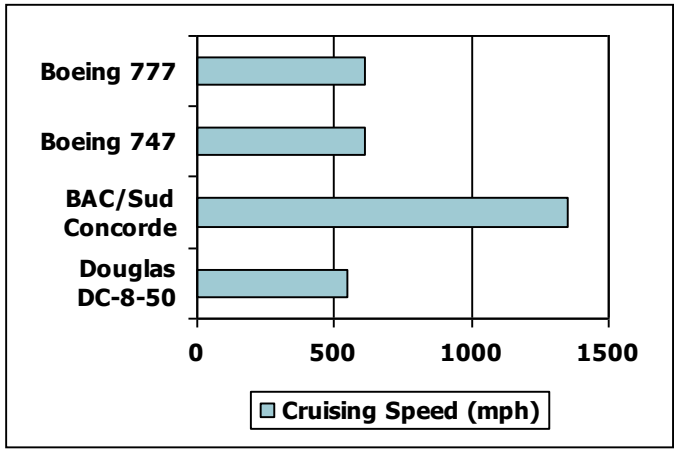
- 12-inch wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

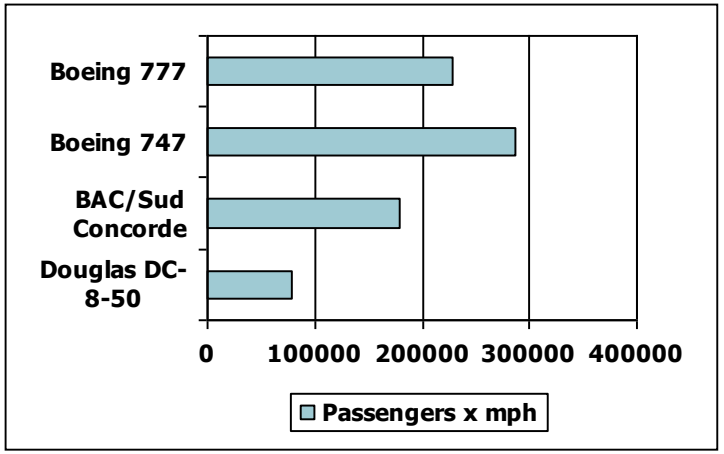
Defining Performance

- 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"

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

- 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

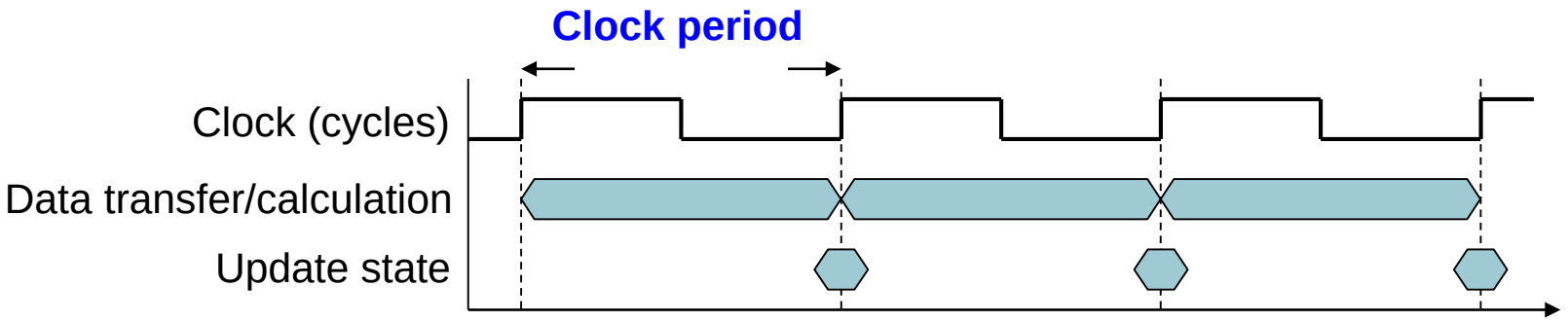
Measuring Execution Time

- 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 for a program
  - Determined by program (language/algorithm), compiler, ISA
- 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
- 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

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

  - 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
    = 2x1 + 1x2 + 2x3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4x1 + 1x2 + 1x3
    = 9
  - Avg. CPI = 9/6 = 1.5

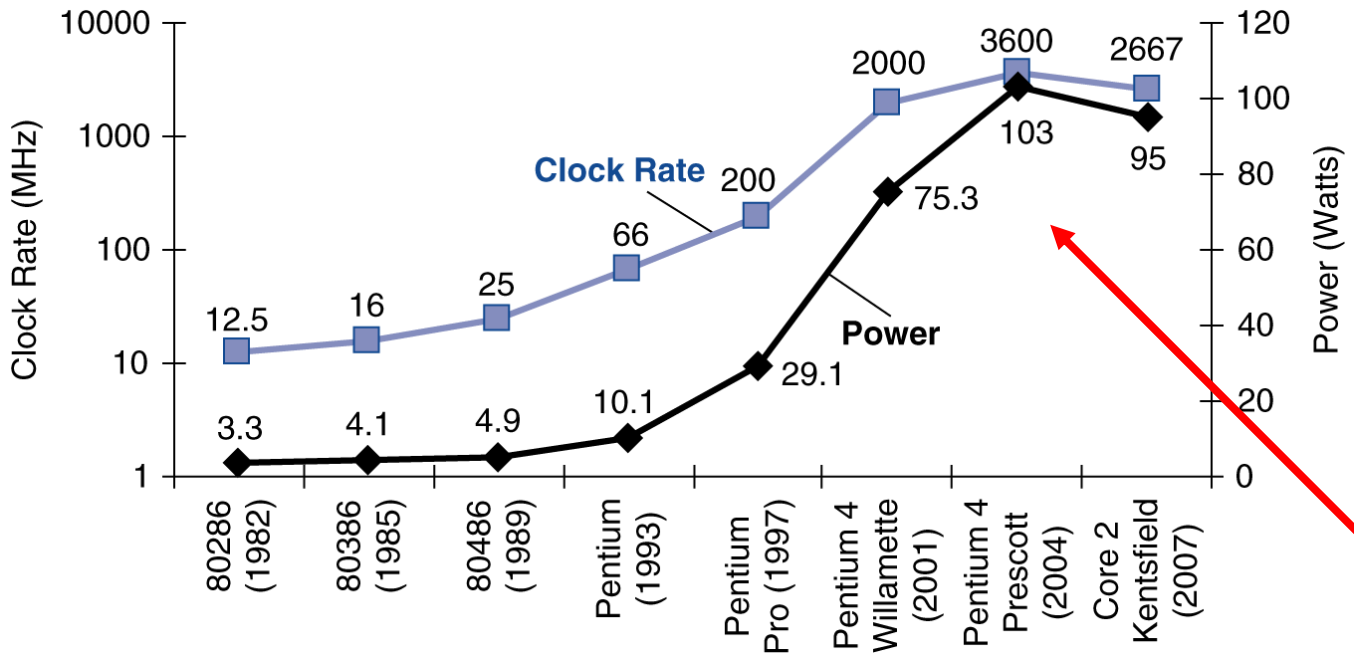
Performance Summary

The BIG Picture

$$\text{CPU Time} = \underbrace{\frac{\text{Instructions}}{\text{Program}}}_{\text{IC}} \times \underbrace{\frac{\text{Clock cycles}}{\text{Instruction}}}_{\text{CPI}} \times \underbrace{\frac{\text{Seconds}}{\text{Clock cycle}}}_{T_{cycle}\ (=1/CR)}$$

- 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

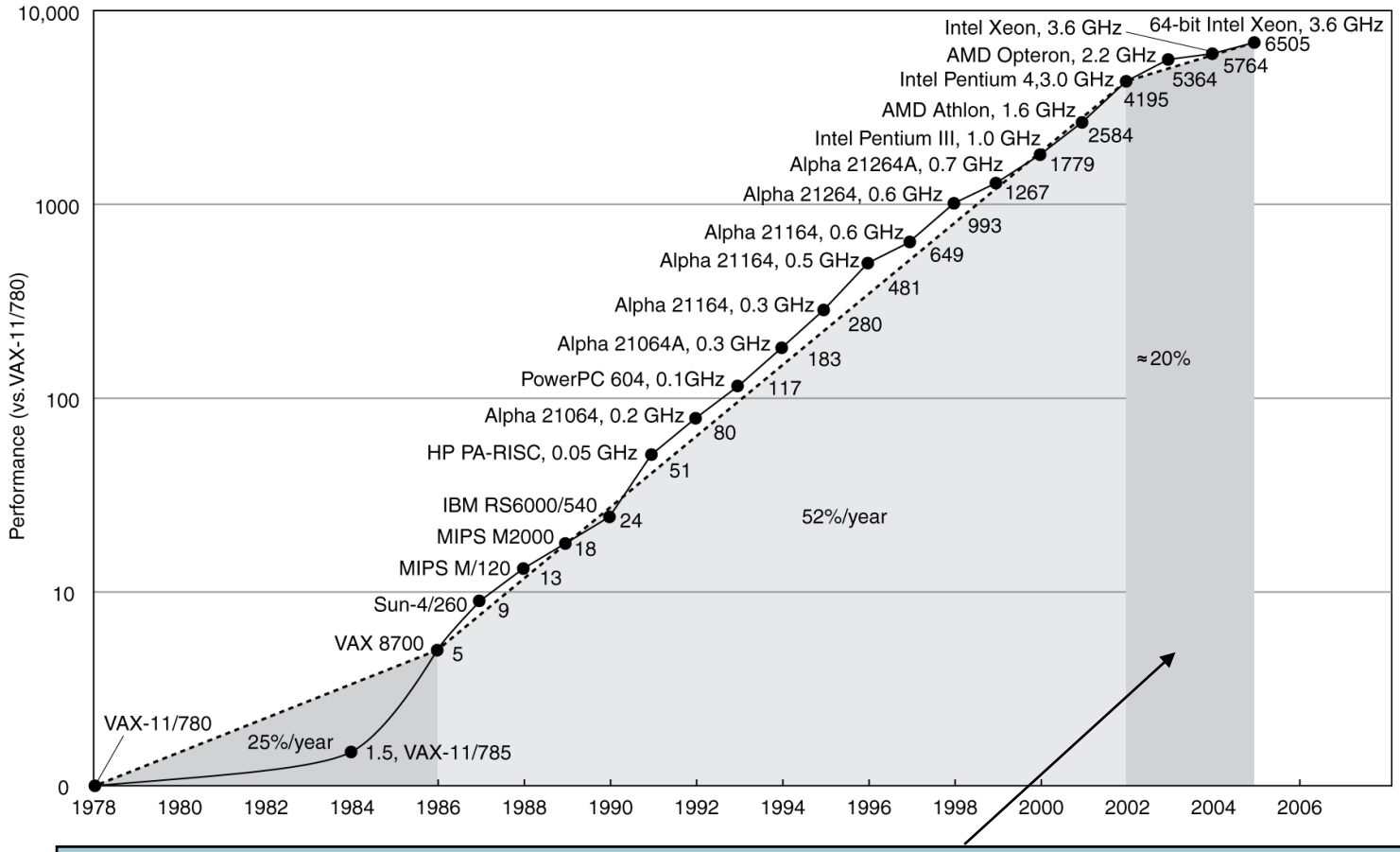
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×22 | 5V → 1V | ×300

Reducing Power

- 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

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

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Rewrite programs for parallelism
  - Load balancing
  - Optimizing communication and synchronization
- Comparison
  - With instruction-level parallelism (pipeline)
    - Hardware executes multiple instructions at once
    - Hidden from the programmer (Note: In the past, programmers could rely on innovations in hardware to double the performance every 18 months without having to change a line of code)

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 Opteron X4 2356

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
						Geometric mean ➔	11.7

Ref time: provided by SPEC
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