Lecture 1
Introduction and Performance

School of Computer Science and Engineering
Soongsil University

1. Computer Abstractions and Technology

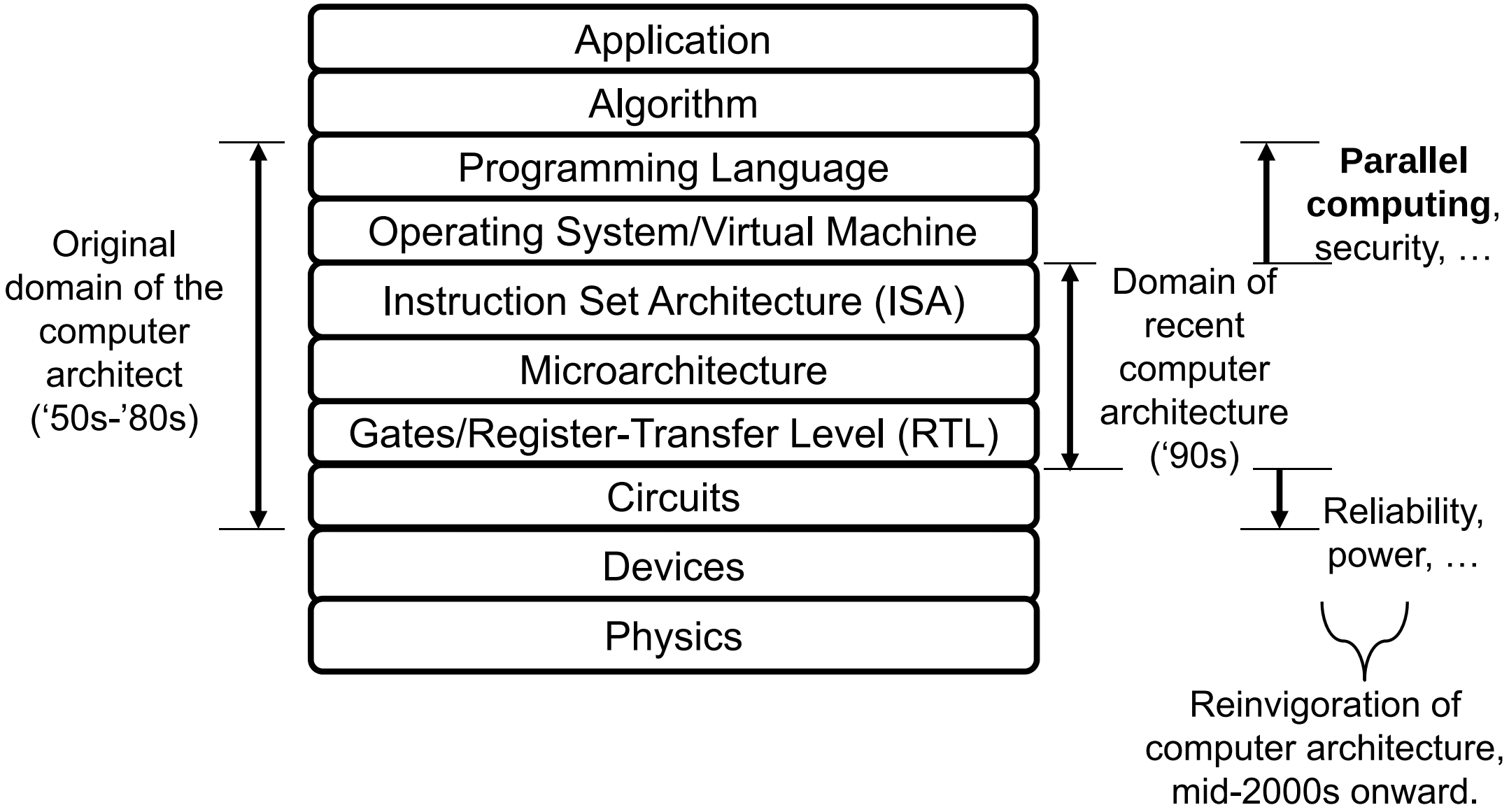
Computer Architecture

- = Instruction Set Architecture (ISA)
- Those aspects of the instruction set available to programmers, independent of the hardware on which the instruction set was implemented
- The term computer architecture was first used in 1964 by Gene Amdahl, G. Anne Blaauw and Frederick Brooks, Jr., the designers of IBM System/360
- IBM System/360 was a family of computers
  - all with the same architecture
  - but with a variety of organization (=implementations).

Architecture vs. Implementation

- **Instruction set architecture (=Architecture)**
  - Interface between hardware and lowest-level software
  - Defines what a computer system does in response to a program and a set of data
  - Includes all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory access, I/O, ...
  - MIPS, IA-32, IA-64, Sparc, PowerPC, IBM 390, etc.
- **Implementation (=Organization=Microarchitecture)**
  - Hardware that obeys the architecture abstraction
  - Defines how a computer does in response to a program and a set of data
  - 8086, 386, 486, Pentium, Pentium II, Pentium 4, etc.

Abstraction Layers in Modern Systems



1.1 Introduction

- **What you can learn in this book**
  1. How are program written in a high-level language translated into the language of the hardware, and how does the hardware execute the resulting program?
  2. What is the interface between the soft ware and the hardware, and how does soft ware instruct the hardware to perform needed functions?
  3. What determines the performance of a program, and how can a programmer improve the performance?
  4. What techniques can be used by hardware designers to improve performance?
  5. What techniques can be used to improve energy efficiency?
  6. What are the reasons for and the consequences of the recent switch from sequential processing to parallel processing?
  7. What great ideas did computer architects come up with that lay the foundation of modern computing?

Five Classic Components

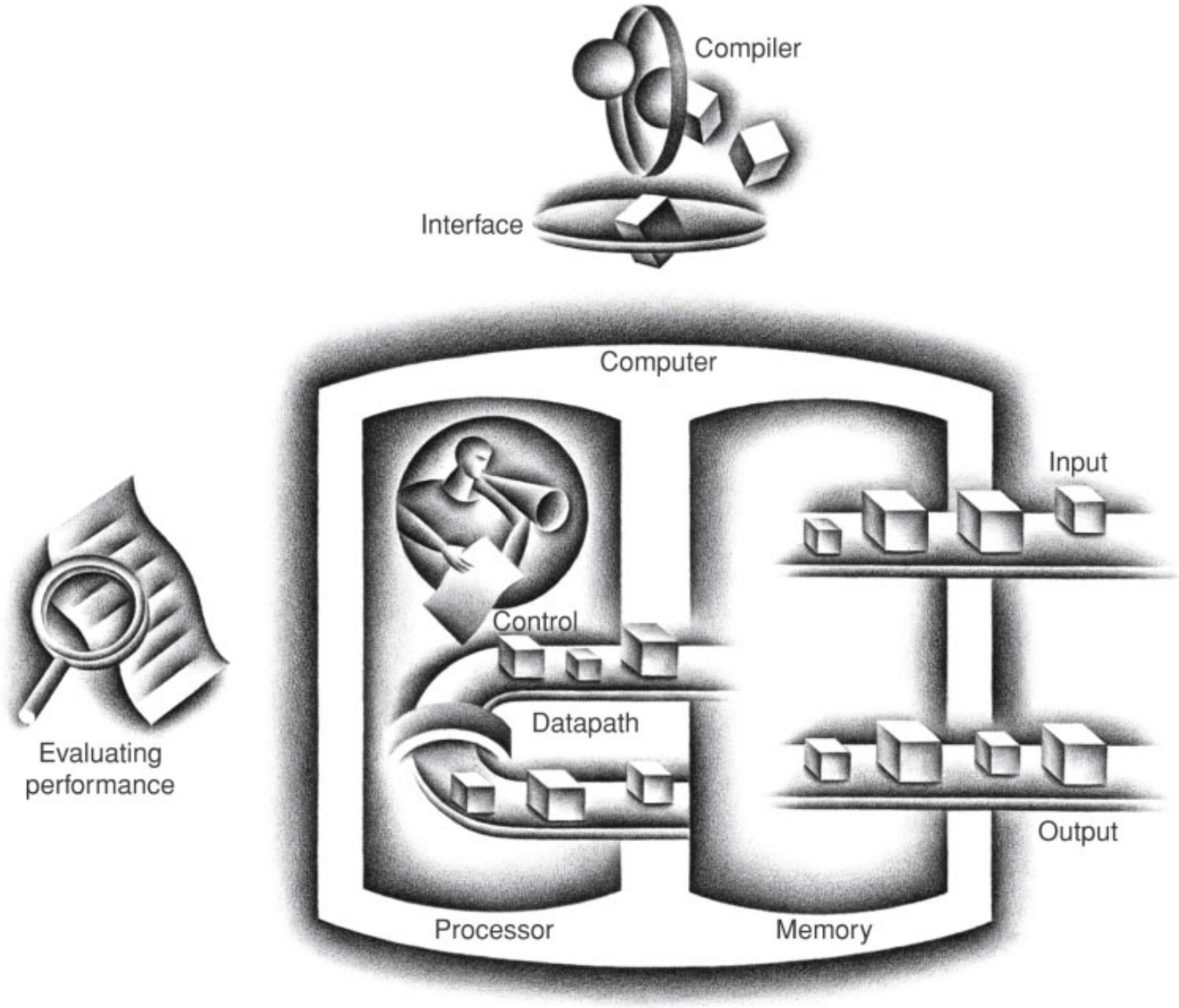
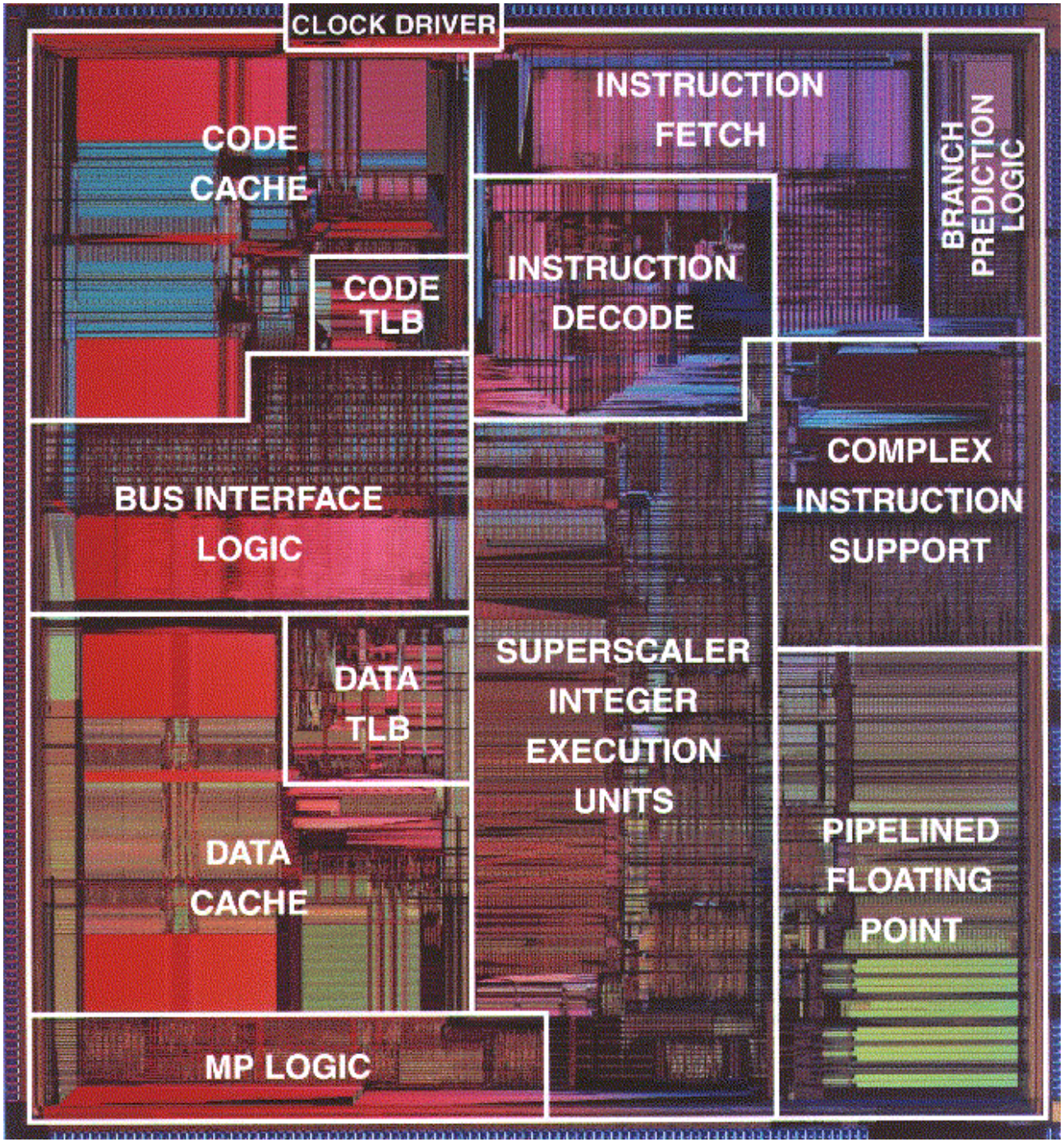


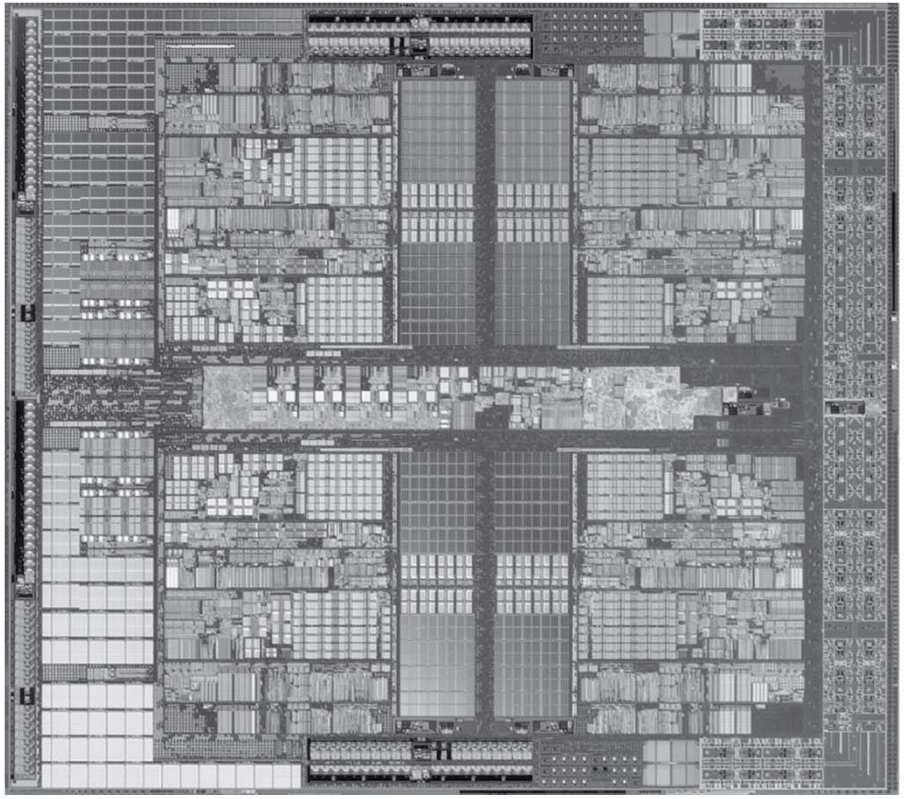
Figure 1.5

Intel Pentium



AMD Barcelona Processor

- **4 processors or "cores"**



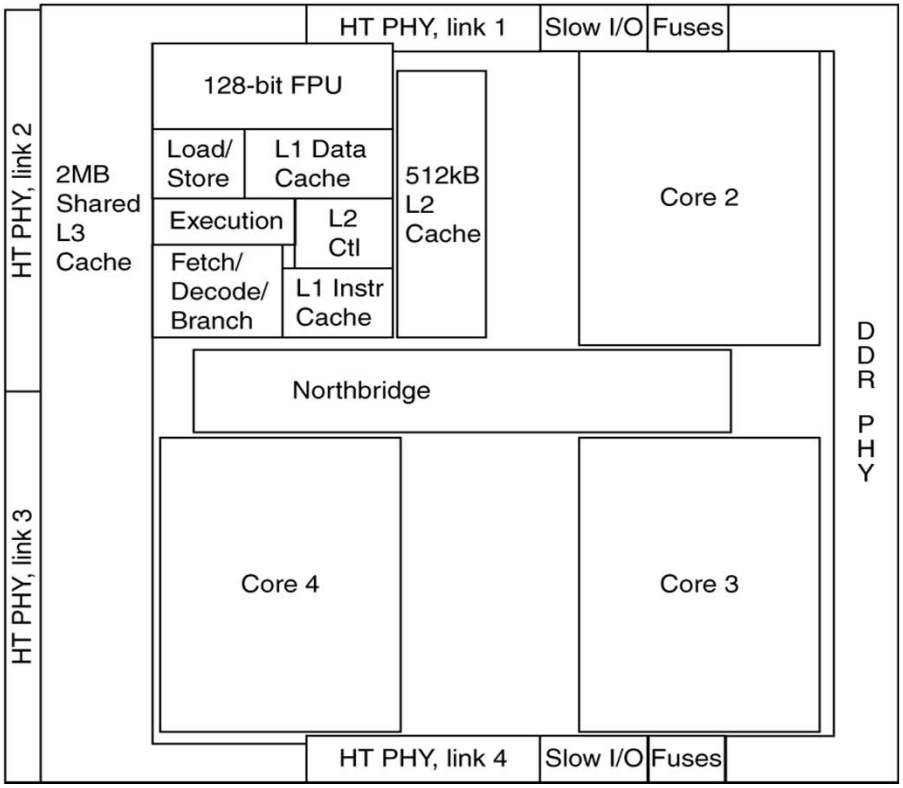


Figure 1.9 in 4ed

Apple A5 Chip in iPad 2

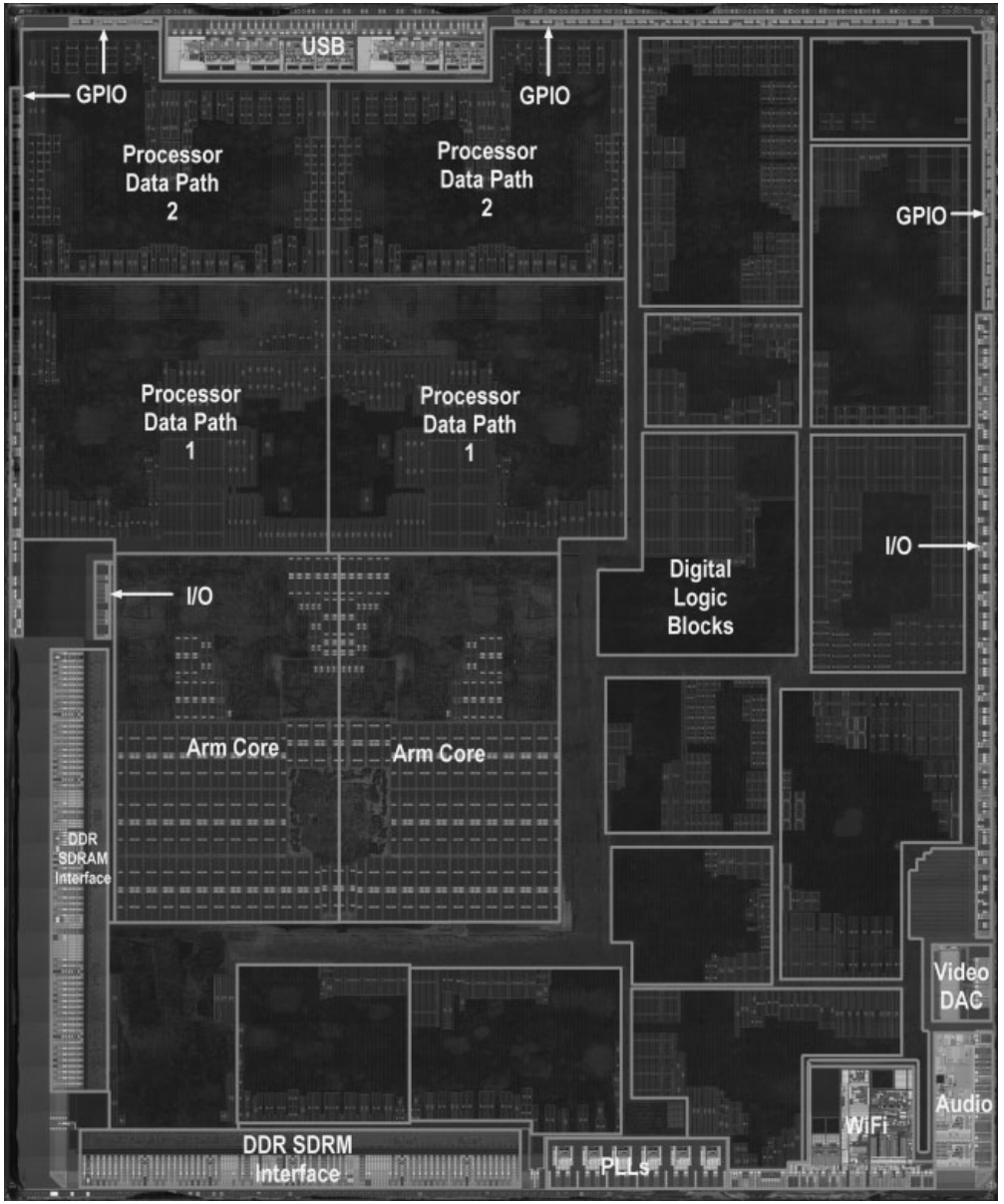


Figure 1.9

FIGURE 1.9 The processor integrated circuit inside the A5 package. The size of chip is 12.1 by 10.1 mm, and it was manufactured originally in a 45-nm process (see Section 1.5). It has two identical ARM processors or cores in the middle left of the chip and a PowerVR graphical processor unit (GPU) with four datapaths in the upper left quadrant. To the left and bottom side of the ARM cores are interfaces to main memory (DRAM). (Courtesy Chipworks, www.chipworks.com)

2^x vs. 10^y

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

Figure 1.1

FIGURE 1.1 The 2^X vs. 10^Y bytes ambiguity was resolved by adding a binary notation for all the common size terms. In the last column we note how much larger the binary term is than its corresponding decimal term, which is compounded as we head down the chart. These prefixes work for bits as well as bytes, so *gigabit* (Gb) is 10^9 bits while *gibibits* (Gib) is 2^{30} bits.

1.2 8 Great Ideas in Computer Architecture

1. Design for Moore's Law
2. Use Abstraction to Simplify Design
3. Make the Common Case Fast
4. Performance via Parallelism
5. Performance via Pipelining
6. Performance via Prediction
7. Hierarchy of Memories
8. Dependability via Redundancy

1.6 Performance

Year	Technology	Relative performance/unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	IC	900
1995	VLSI	2,400,000
2013	ULSI	250,000,000,000

Figure 1.10

- **Moore's law**

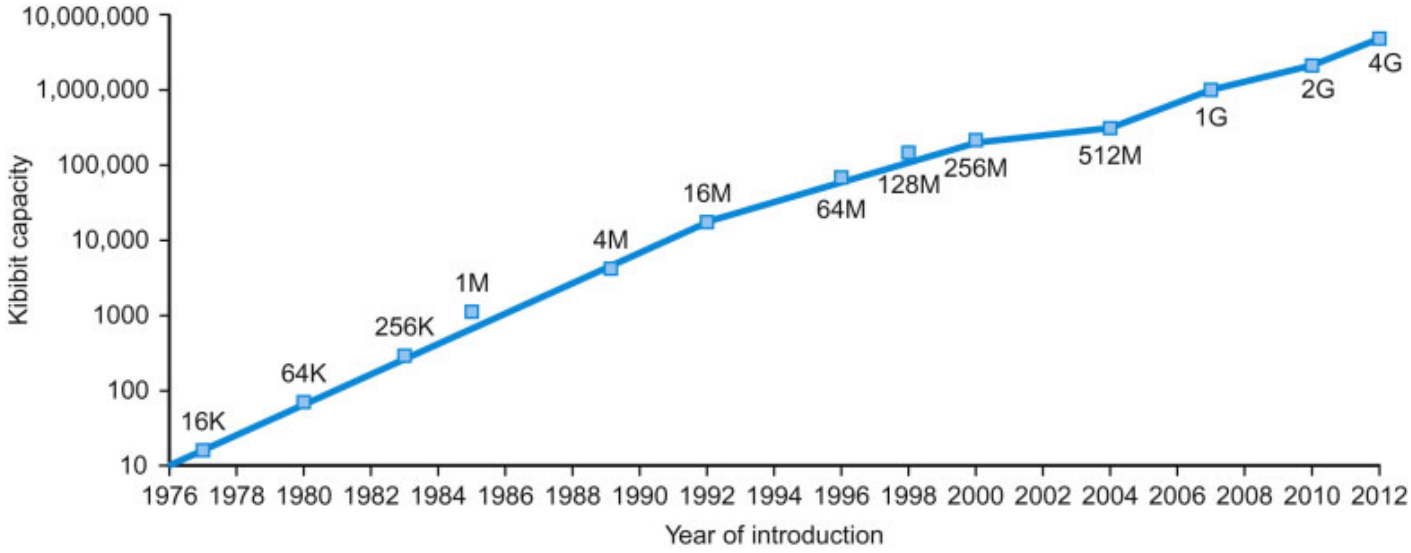
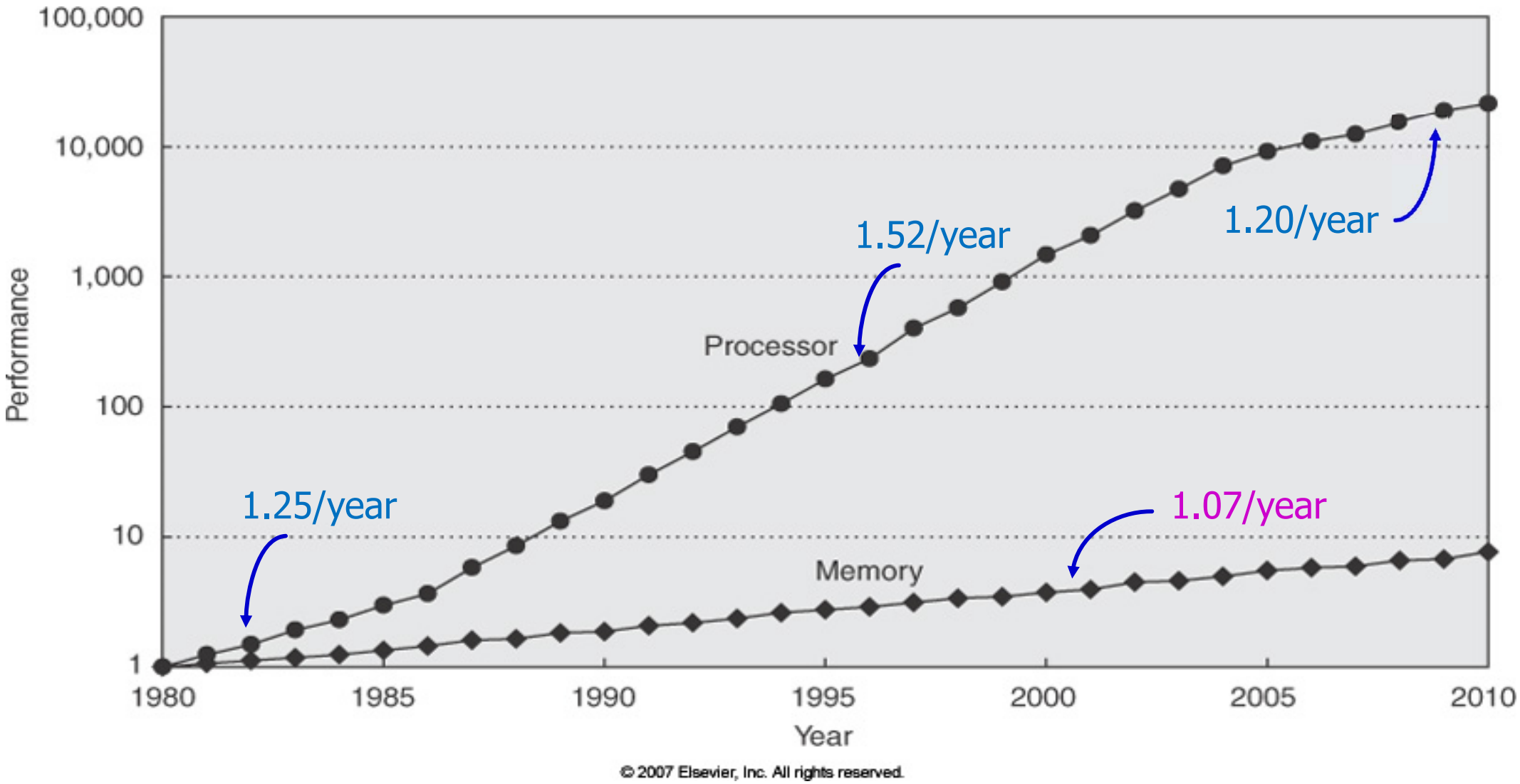


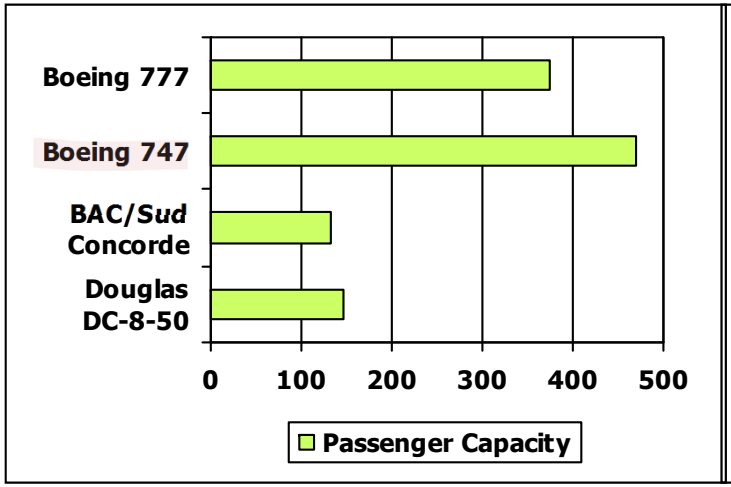
Figure 1.11

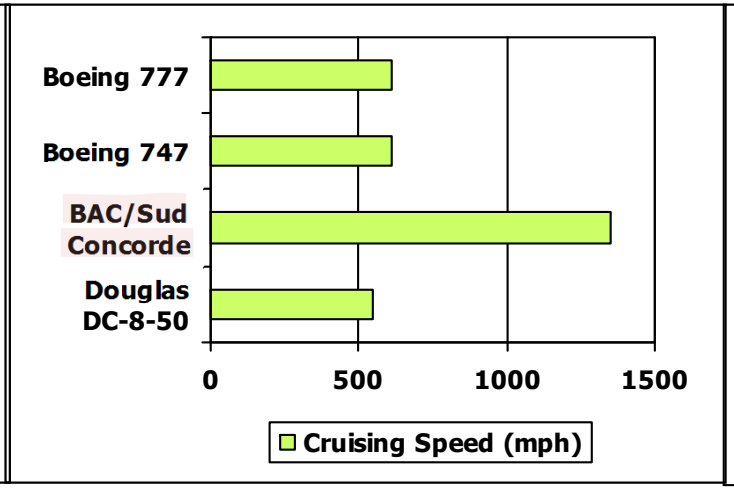
Processor-Memory Performance Gap

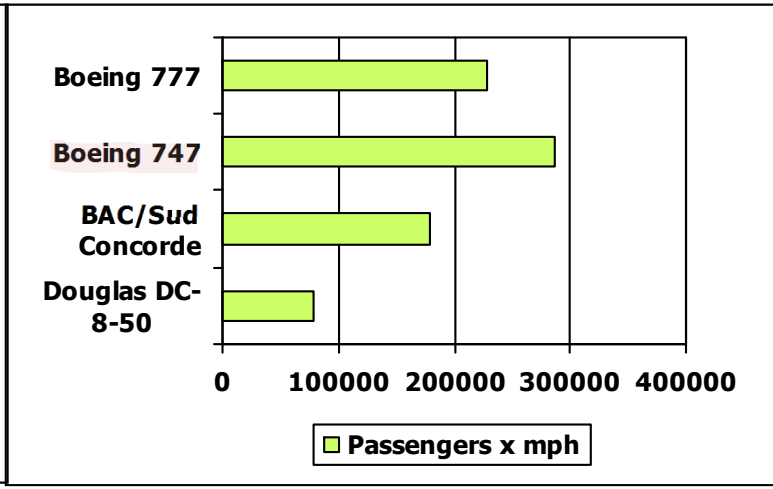


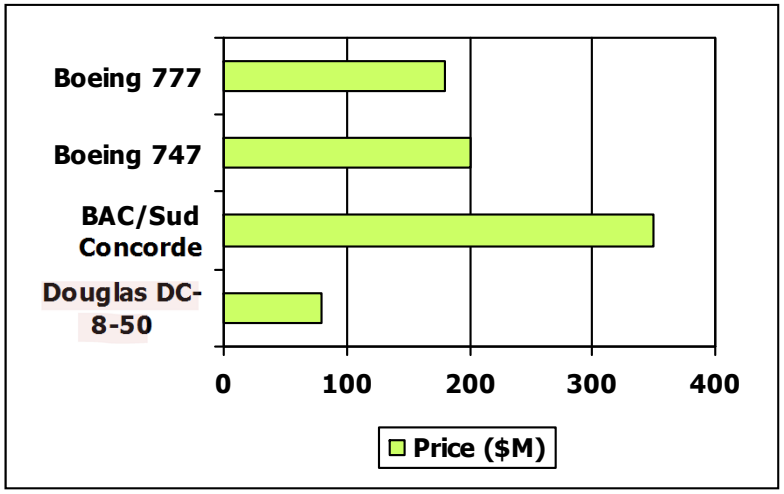
Defining Performance

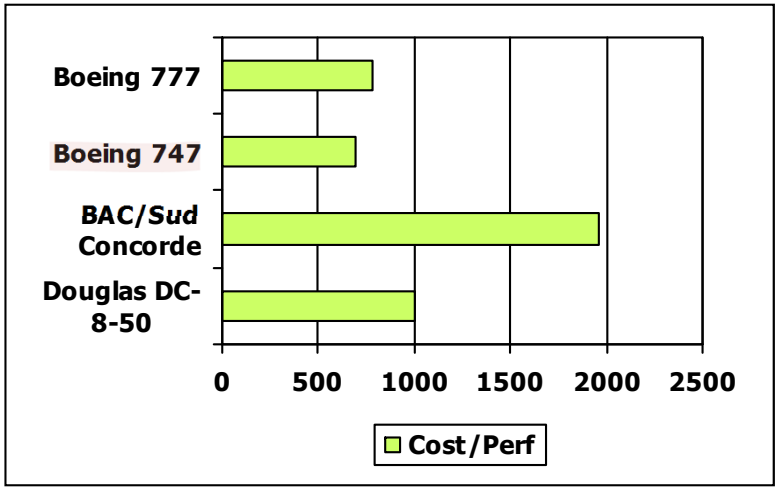
- **Which airplane is the best ?**











Performance of a Computer

- **Response time (= execution time)**
  - The time between the start and completion of a task
- **Throughput (= bandwidth)**
  - The number of tasks completed per unit time
- **Performance and execution time**
  - $\text{Performance}_x = 1 / \text{Execution time}_x$
- **X is n times faster than Y**

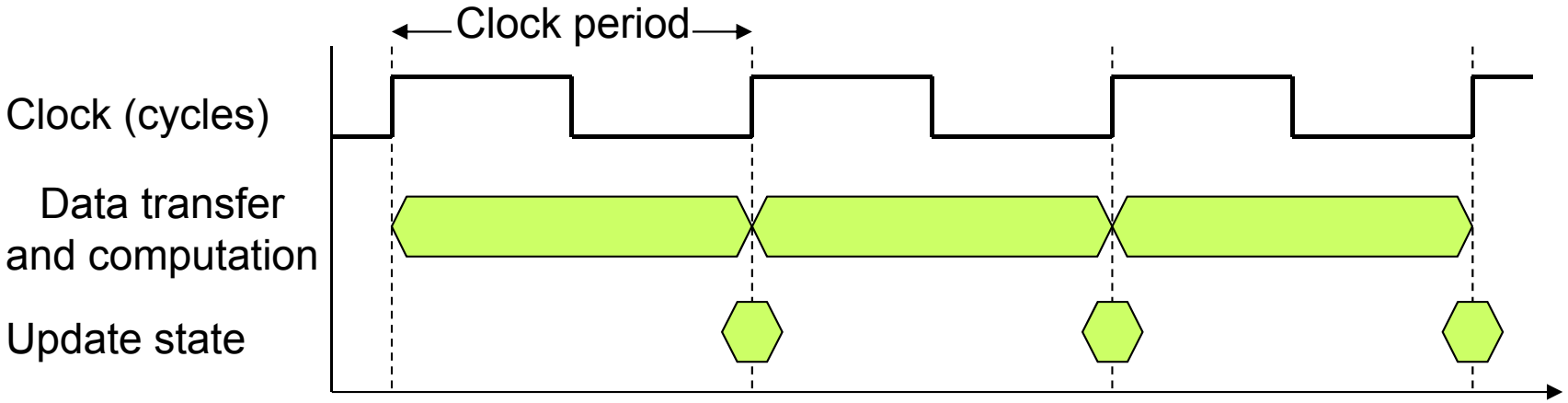
$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Exceution Time}_X} = n$$

Measuring Performance

- **Definitions of time**
  - Wall-clock time = Response time = Elapsed time
    - Total time to complete a task
    - Including disk accesses, memory accesses, I/O activities, OS overhead and etc.
    - Determines system performance
  - CPU execution time = CPU time
    - The time CPU spends computing for this task
    - Not including time spent waiting for I/O or running other programs
    - CPU time = User CPU time + System CPU time
    - Determines CPU performance
- **Definitions of performance**
  - System performance: based on elapsed time
  - CPU performance: based on user CPU time
- **We will focus on CPU performance for now!**

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock
- **Clock period=clock cycle time: duration of a clock cycle**
  - e.g., 250ps = 250×10^{-12}s= 0.25ns = 0.25×10^{-9}s
- **Clock frequency (rate): cycles per second**
  - e.g., 1/ (0.25×10^{-9}s) = 4.0×10^{9}Hz = 4000MHz = 4.0GHz



CPU Performance and Its Factors

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

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

- **Performance improved by**
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

Example: Improving Performance

- 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 Performance

Clock Cycles = IC x CPI

- **Instruction Count (IC) for a program**
  - Determined by program, ISA and compiler
- **Average Cycles Per Instruction (CPI)**
  - Determined by CPU hardware
  - If different instructions have different CPI,
    - average CPI affected by instruction mix

Example: Using the Performance Equation

- Same instruction set architecture, same program
- Clock cycle time$_A$ = 250ps, CPI$_A$ = 2.0
- Clock cycle time$_B$ = 500ps, CPI$_B$ = 1.2
- **Which is faster, and by how much ?**

[Answer]

- Let I = instruction count for the program.
- CPU time$_A$ = IC$_A$ x CPI$_A$ x clock cycle time$_A$
  = I x 2.0 x 250 ps = 500 x I ps
- CPU time$_B$ = I x 1.2 x 500 ps = 600 x I ps
- Then

$$\frac{\text{CPU Performanc e}_A}{\text{CPU Performanc e}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \times \text{I ps}}{500 \times \text{I ps}} = 1.2$$

- Thus, A is 1.2 times faster than B for this program.

The Classic CPU Performance Equation

CPU Time = IC x CPI x clock cycle time

= (IC x CPI) / clock rate

- **Example: Comparing Code Segments**
  - **Which code sequence executes the most instructions?**
  - **Which will be faster ?**
  - **What is the CPI for each sequence ?**

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

[Answer]

- instruction count$_1$ = 2 + 1 + 2 = 5 and
  instruction count$_2$ = 4 + 1 + 1 = 6
  Thus (1) executes fewer instructions.
- CPU clock cycles$_1$ = 2x1 + 1x2 + 2x3 = 10 and
  CPU clock cycles$_2$ = 4x1 + 1x2 + 1x3 = 9
  Thus (2) is faster.
- CPI$_1$ = CPU clock cycles$_1$ / instruction count$_1$
  = 10 / 5 =2.0
  CPI$_2$ = 9 / 6 = 1.5
  (2) has lower CPI.

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

Supplement

1.1 Introduction

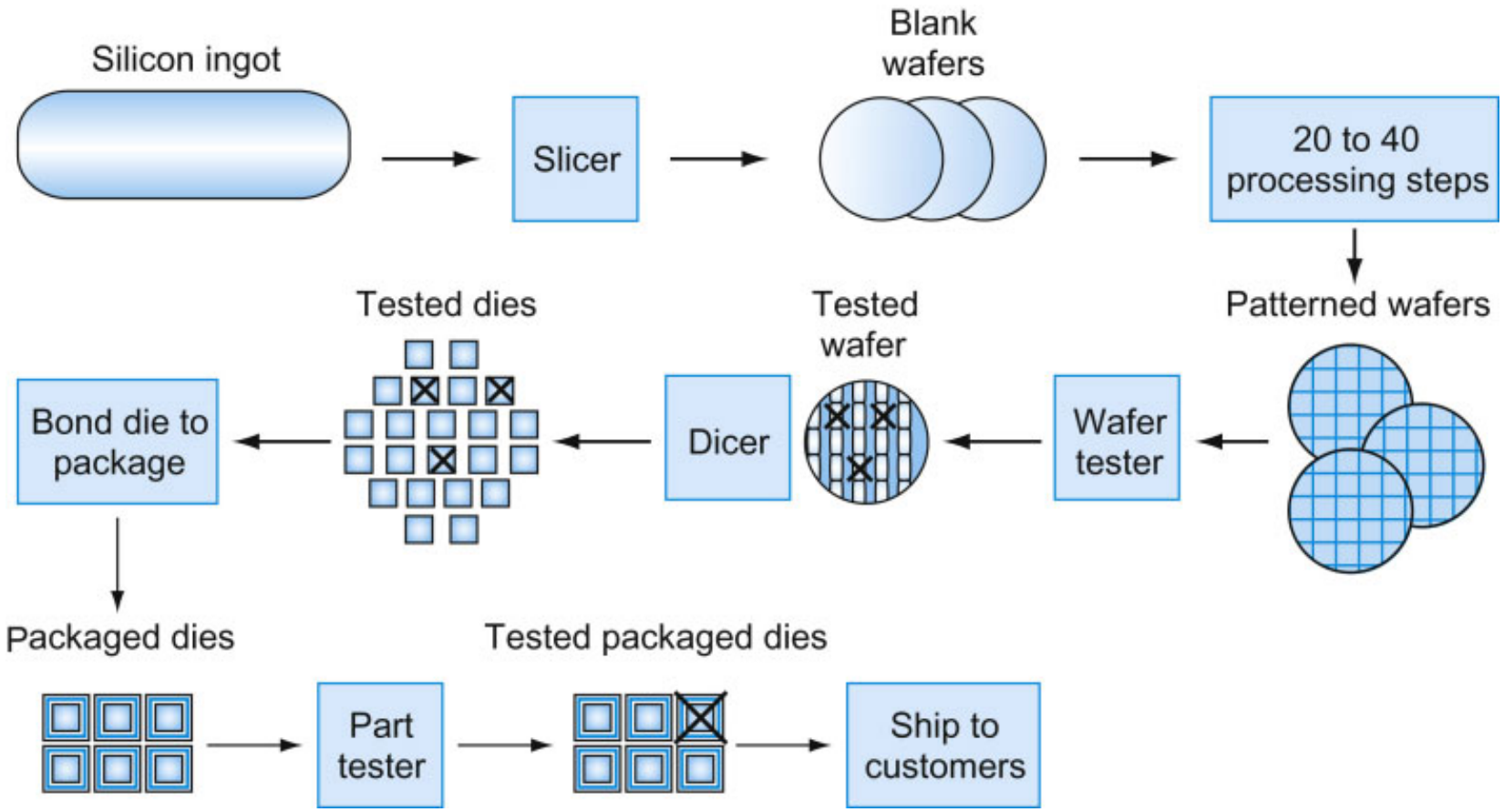
- **Progress in computer technology**
  - Moore's Law
  - Doubling every 18 months
- **Makes novel applications feasible**
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- **Computers are pervasive**
  - Ubiquitous computing

Classes of Computing Applications and Their Characteristics

1. **Personal computers (PCs)**
   - Good performance to a single user at low cost
   - Third-party software
1. **Servers**
   - Modern form of mainframes, mini- and supercomputers
   - Usually accessed via a network
   - Expandability and dependability
   - Low-end servers, supercomputers, computer clusters
3. **Embedded computers**
   - A computer inside another device used for running one predetermined application or collection of software
   - Minimum performance with stringent limitations on cost or power

1.5 Technologies for Building Processors and Memory

- **The chip manufacturing process (Figure 1.12)**



12-inch (300-nm) Wafer of Intel Core i7

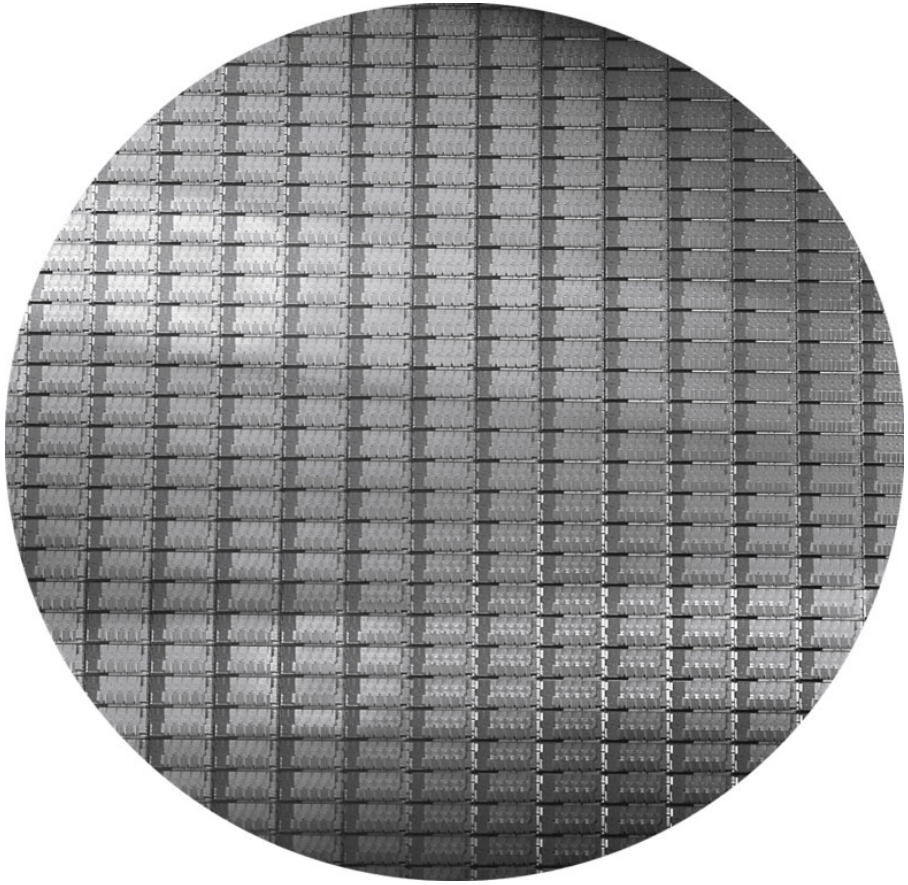


Figure 1.13

- **280 chips, each 20.7 x 10.5 nm**
- **32-nm technology**

Elaboration

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

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

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

- **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

1.7 The Power Wall

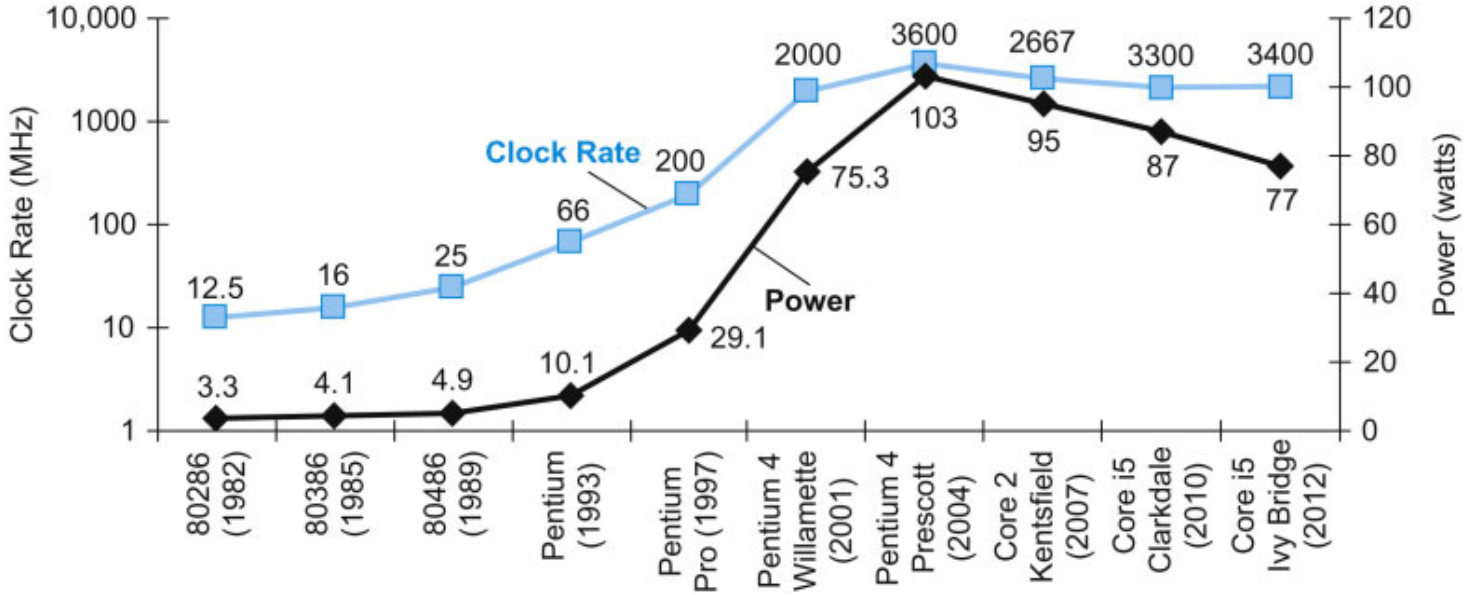


Figure 1.16

- **In CMOS IC technology**

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

$\times 30$ $\quad$ $5\text{V} \rightarrow 1\text{V}$ $\quad$ $\times 1000$

Example: Relative Power

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

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

- **The power wall**
  - We can't reduce voltage further
  - We can't remove more heat
- **How else can we improve performance?**

1.8 The Sea Change: The Switch from Uniprocessors to Multiprocessors

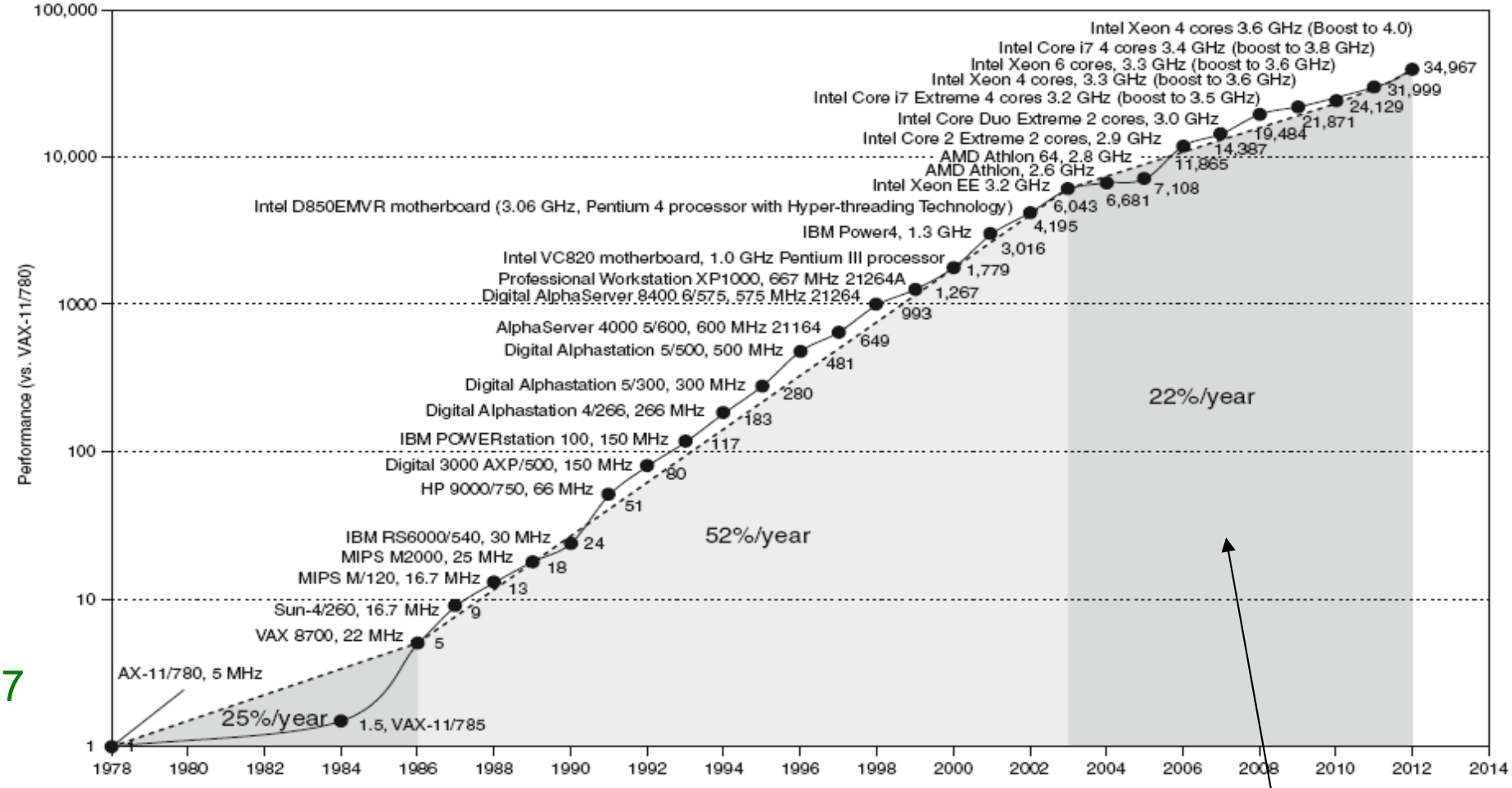


Figure 1.17

Constrained by power, instruction-level parallelism, memory latency

1.8 Real Stuff: Benchmarking the Intel Core i7

- **Benchmark**
  - A program selected for use in comparing computer performance
  - Forming a workload that the user hopes will predict the performance of the actual workload
- **Standard Performance Evaluation Corporation (SPEC)**
  - Founded in 1988
  - Members
- **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)

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

Figure 1.18

SPEC Power Benchmark

- **SPECpower**
  - Power consumption of server at different workload levels
  - Divided into 10% increments, over a period of time
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

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

- **Stochastic Simulation in Java (SSJ)**
  - A Java library for stochastic simulation

SPECpower_ssj2008 for Intel Xeon X5650

- Running on a dual socket 2.66 GHz Intel Xeon X5650 with 16 GB of DRAM and one 100 GB SSD disk.

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	1922
∑ssj_ops / ∑power =		2490

Figure 1.19

1.8 Fallacies and Pitfalls

- **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 4× overall?**

$$\frac{100}{4} = \frac{80}{n} + 20 \quad \Rightarrow \quad 5 = \frac{80}{n} \quad \Rightarrow \quad n = 16$$

- **Corollary:** Make the common case fast

Fallacy: Computers at low utilization use little power

- **Utilization of servers in Google's warehouse scale computer**
  - Mostly operates at 10% ~ 50% load
  - At 100% load less than 1% of the time
  - Even with the special configuration, the best results in 2012 still uses 33% of the peak power at 10% of the load
- **Luiz Barroso and Urs Hὸlzle [2007]**
  - We should redesign hardware to achieve "energy-proportional computing."

Server Manufacturer	Microprocessor	Total Cores/Sockets	Clock Rate	Peak Performance (ssj_ops)	100% Load Power	50% Load Power	50% Load/100% Power	10% Load Power	10% Load/100% Power	Active Idle Power	Active Idle/100% Power
HP	Xeon E5440	8/2	3.0 GHz	308, 022	269 W	227 W	84%	174 W	65%	160 W	59%
Dell	Xeon E5440	8/2	2.8 GHz	305, 413	276 W	230 W	83%	173 W	63%	157 W	57%
Fujitsu Siemens	Xeon X3220	4/1	2.4 GHz	143, 742	132 W	110 W	83%	85 W	65%	80 W	60%

Pitfall: Using a subset of the performance equation as a performance metric

- Must predict performance based on clock rate, IC and CPI
- **Millions of Instructions Per Second (MIPS)**

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

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

  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions
  - CPI varies between programs on a given CPU

1.11 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**

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

  - The best performance measure
- **Power is a limiting factor**
  - Use parallelism to improve performance