Arquitectura de Computadores

Capítulo 1.
Abstracciones, Tecnología y Rendimiento de los Computadores

Chapter 1.
Computer Abstractions and Technology

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Profesores:

G131: Iván González Martínez

G130 y G136: Francisco J. Gómez Arribas

Chapter 1

Computer Abstractions and Technology

(based on the original material of the book: "Computer Organization and Design: The Hardware/Software Interface" 4th edition)

Outline

- Computer: A historical perspective
- Abstractions
- Technology
  - Performance
    - Definition
    - CPU performance
  - Measuring and evaluating performance
  - Power trends: multi-processing

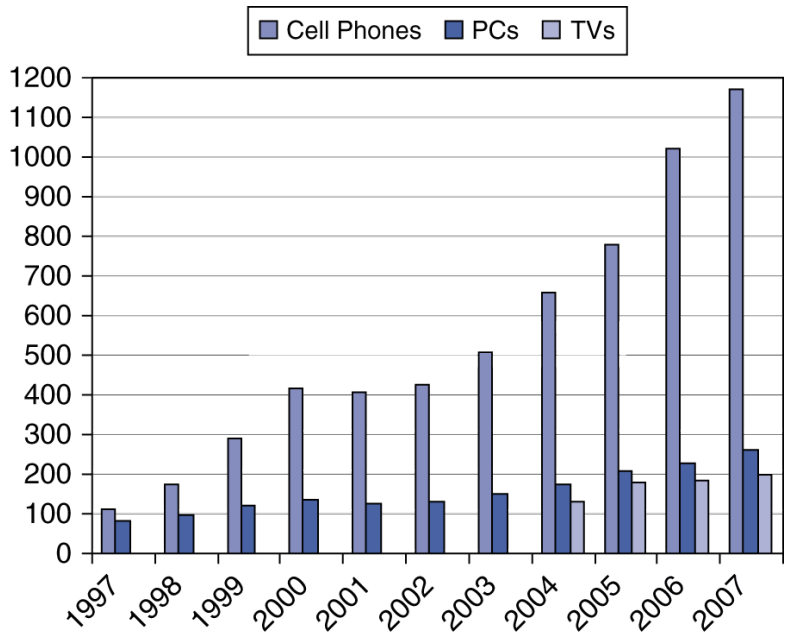
The Computer Revolution

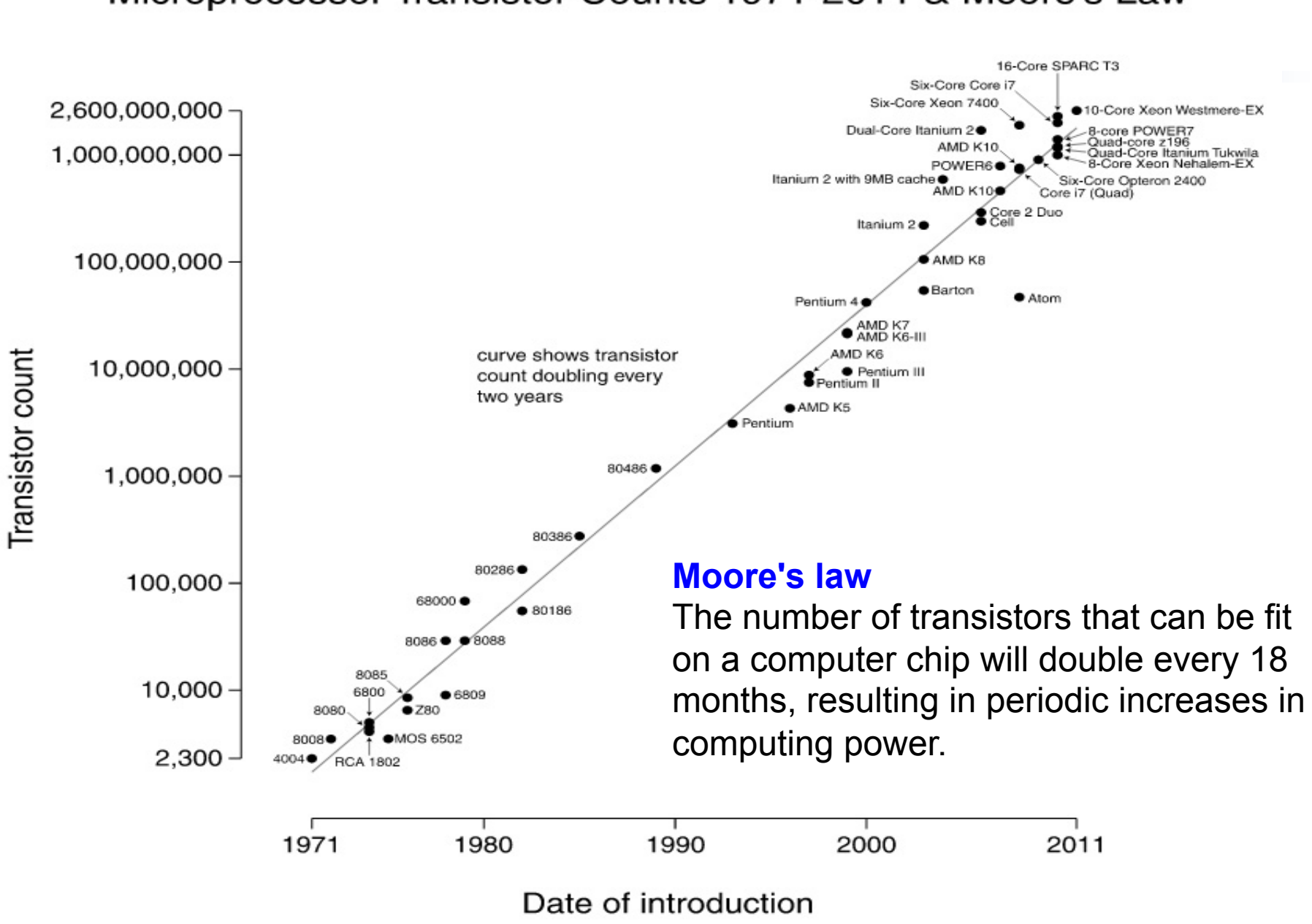
- Progress in computer technology
  - Underpinned by Moore's Law
- Classes of computers :

Computers are pervasive

- 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

The Processor Market

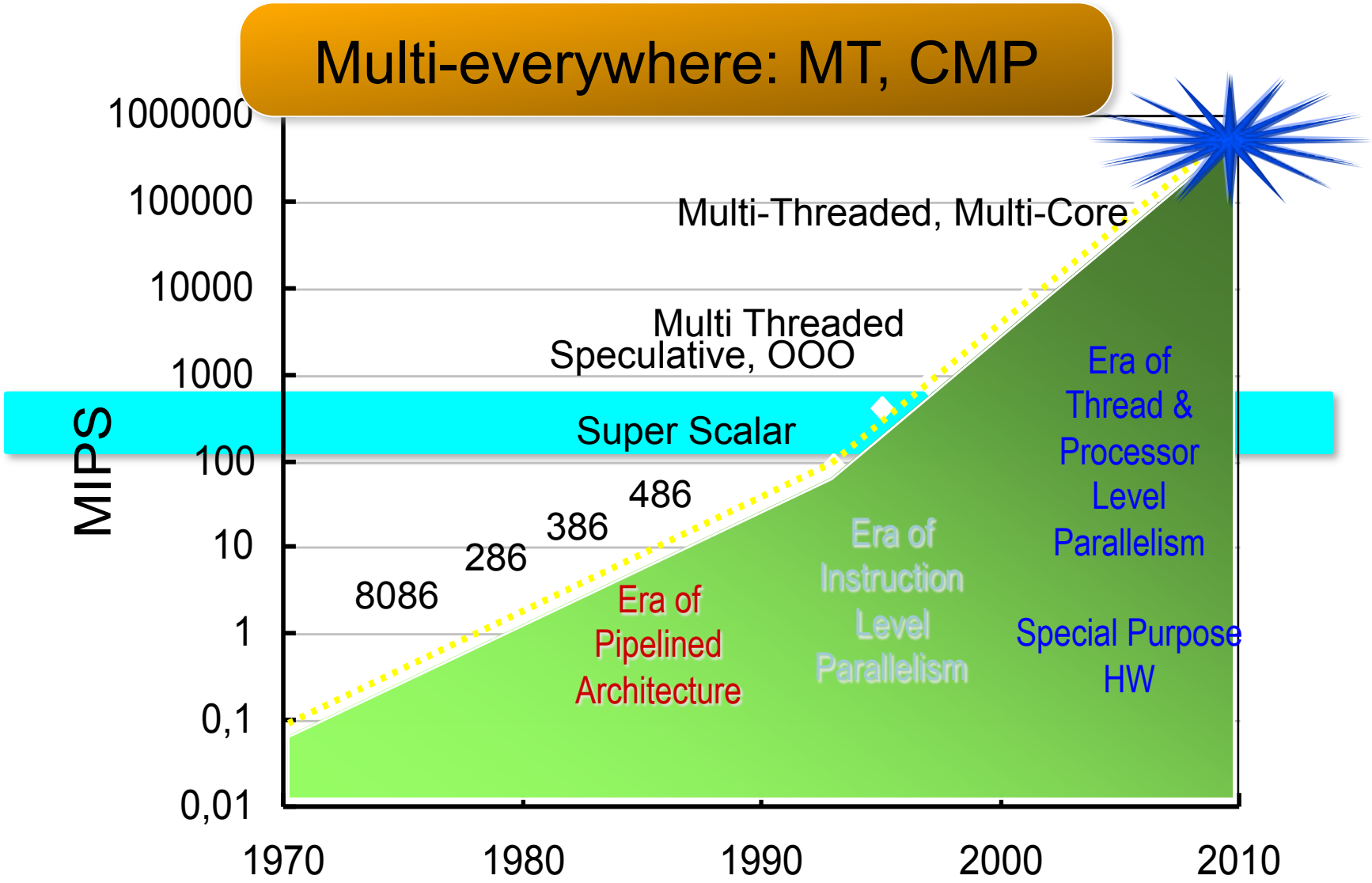




Moore's law

The number of transistors that can be fit on a computer chip will double every 18 months, resulting in periodic increases in computing power.

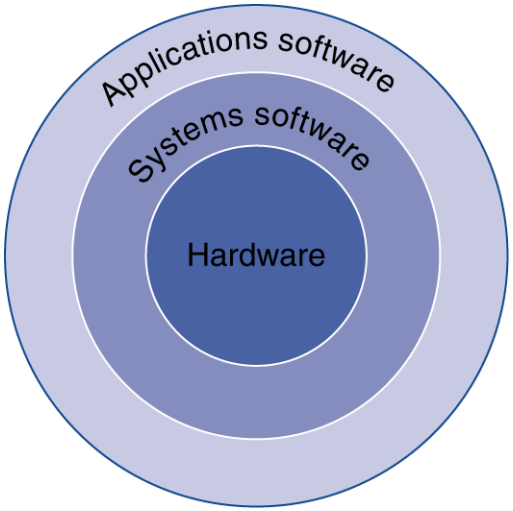
Evolution of Processors in Computers



Understanding Performance

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

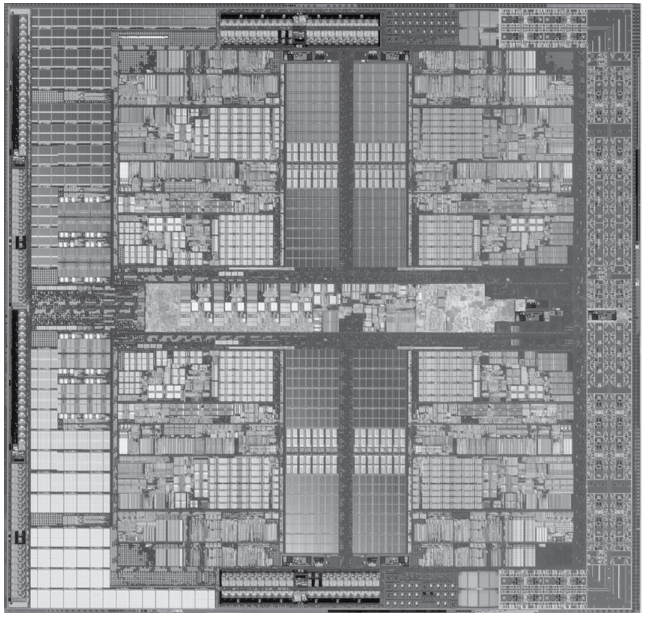
Below Your Program

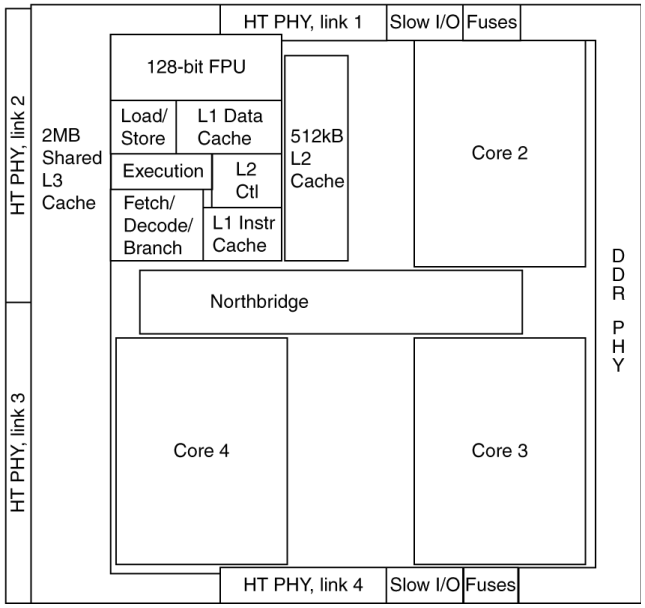


- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O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

Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data





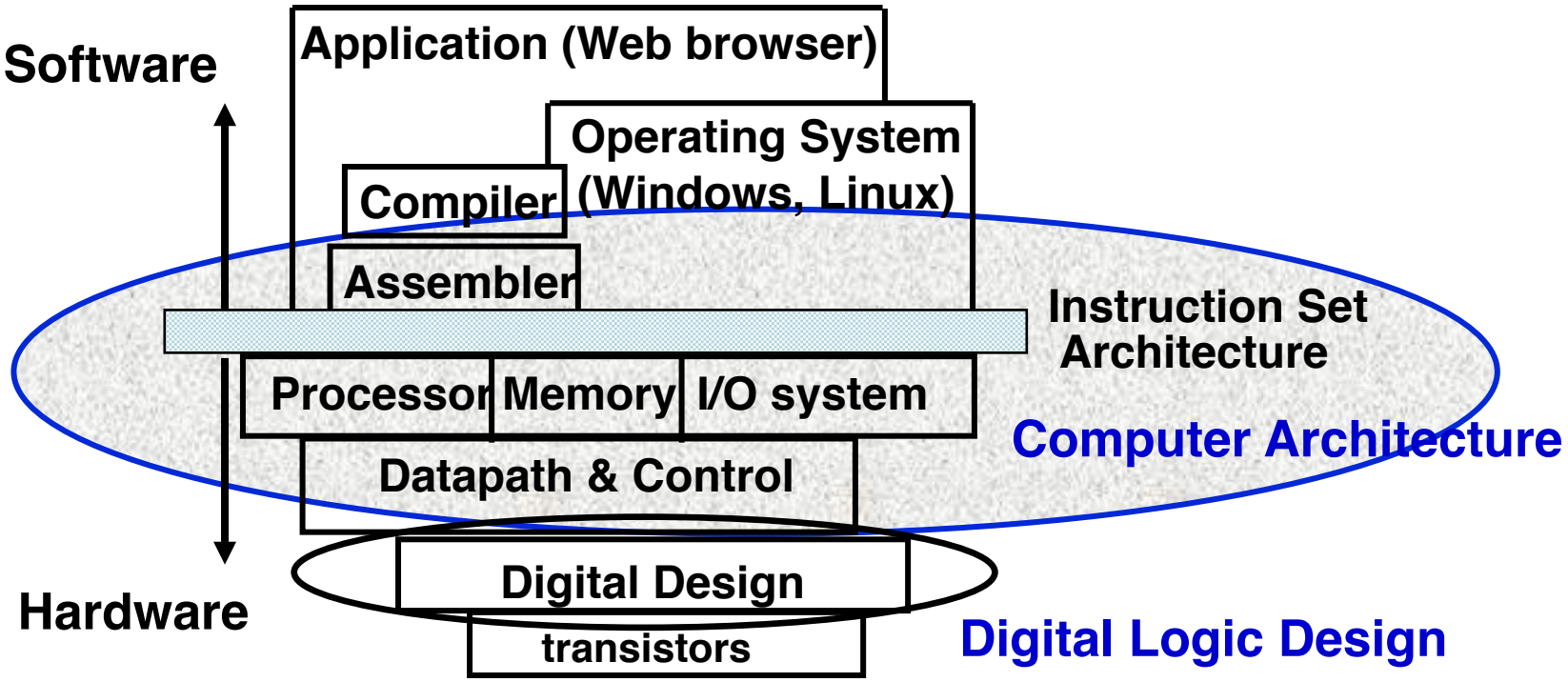
AMD Barcelona: 4 processor cores

Abstractions

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - A set of hardware-implemented instructions, the symbolic name and the binary code format of each instruction.
  - Is the structure of a computer that a machine language programmer (or a compiler) must understand to write a correct (timing independent) program for that machine.
  - The hardware/software interface.

Abstractions: Computer Arch.

- **Organization:** Structures such as datapath, control units, memories, and the busses that interconnect them.
- **Hardware:** The logic, the electronic technology employed, the various physical design aspects of the computer

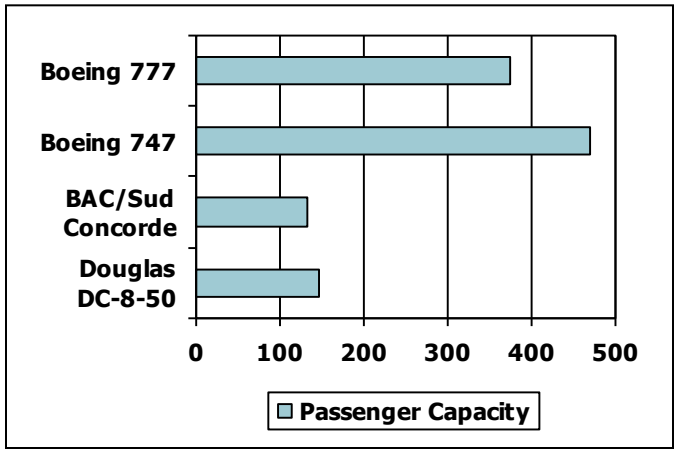


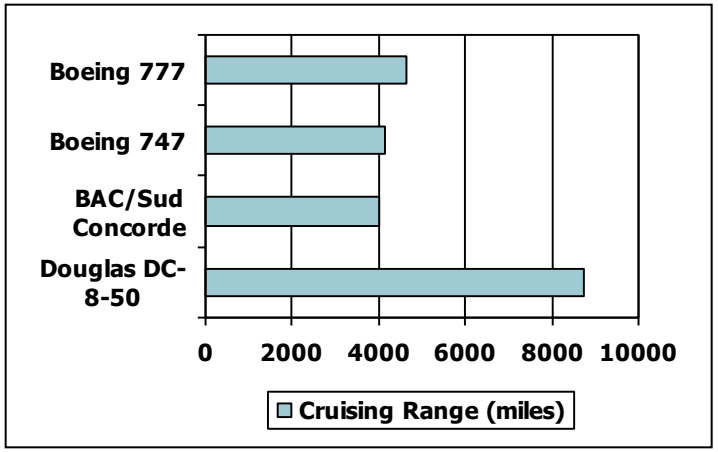
Abstractions: ISA examples

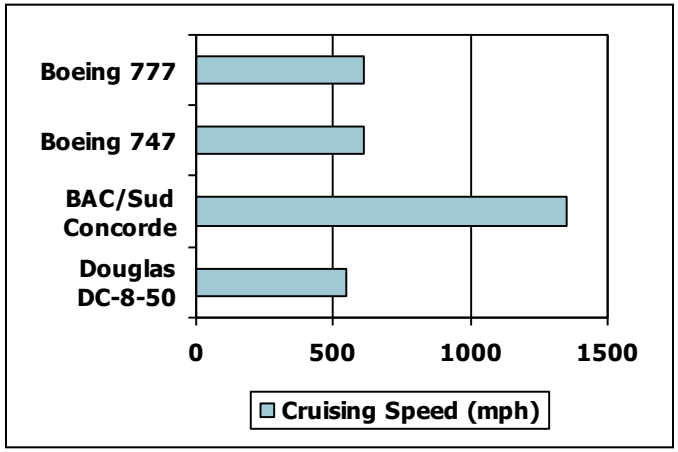
	RISC (reduced instruction set computers)	CISC (complex instruction set computers)
Memory access	restricted to load/store instructions, and data manipulation instructions are register-to-register	is directly available to most types of instructions
Addressing mode	limited in number	substantial in number
Instruction formats	all of the same length	of different lengths
Instructions	perform elementary operations	perform both elementary and complex operations
Control unit	Hardwired, high throughput and fast execution	Microprogrammed, facilitate compact programs and

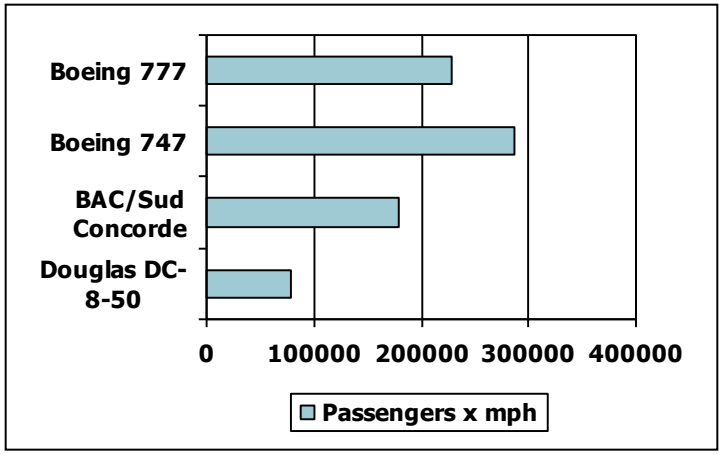
Defining Performance

- 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
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

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

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

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

  - Weighted average CPI

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

Performance Summary

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

- 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

Amdahl's Law

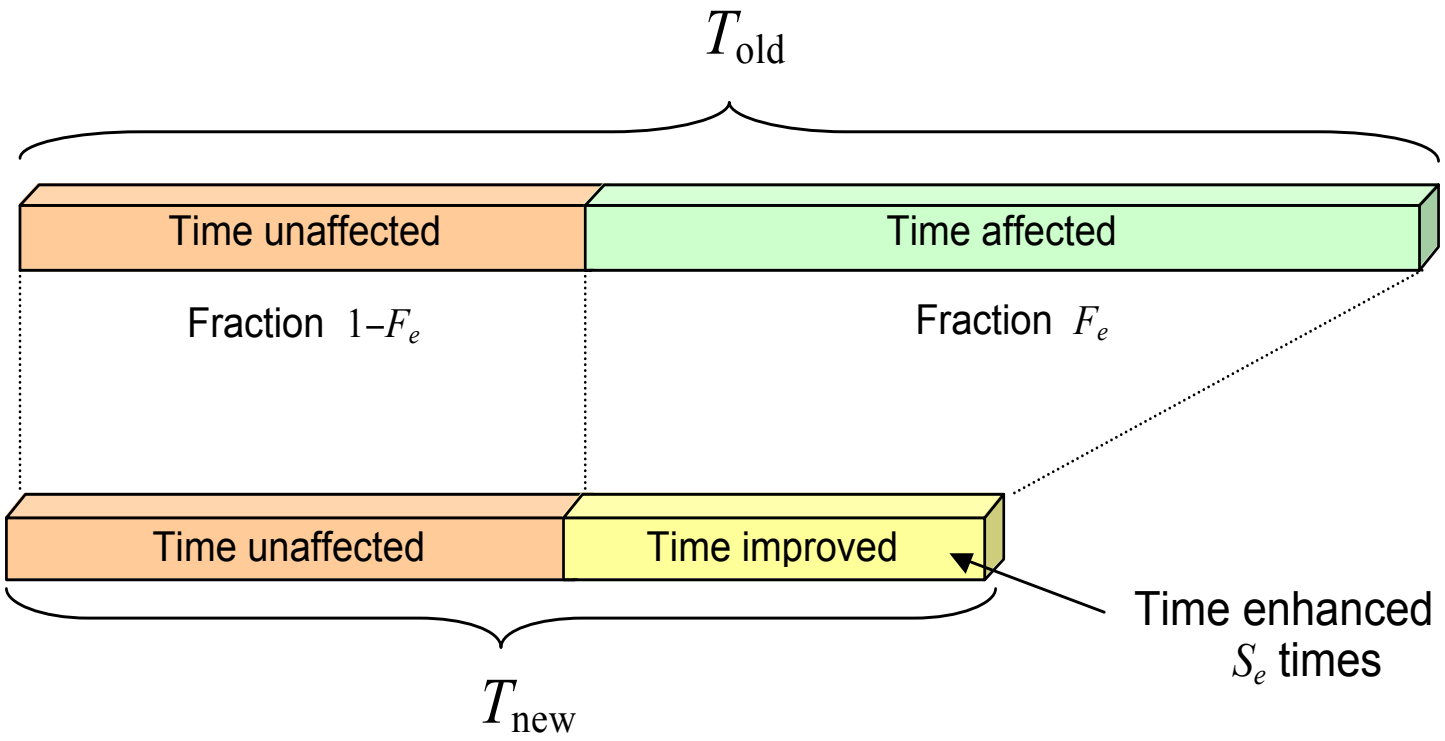
- Speedup is the improvement achieved in execution time.

$$Speedup_{Overall} = \frac{Execution\ time_{old}}{Execution\ time_{new}}$$

- Amdahl´s Law is used to find the speedup to an overall system when only part of the system is improved.

$$Speedup_{Overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Amdahl´s law: 1 improvement



$$Speedup_{Overall} = \frac{1}{(1-F_e) + \frac{F_e}{S_e}}$$

F_e: Fraction enhanced

S_e: S*peedup* enhanced. Amount of improvement

Amdahl´s law

- Performance is limited to the non-speedup portion of the program.

Execution time after improvement = Execution time of unaffected part + (Execution time of affected part / Amount of Improvement)

- Corollary of Amdahl's law: *Make the common case fast*.

Amdahl´s law: example 1

- 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 **12 seconds**, what will the speedup be if **half of the 12 seconds** is spent executing floating-point instructions?

Time = 6 (other instr) + 6 (FP instr) / 5 = 7.2 seconds.

Speedup = 12/7.2 = 1.67

$$Speedup_{Overall} = \frac{1}{(1-F_e)+\frac{F_e}{S_e}} = \frac{1}{(1-0.5)+\frac{0.5}{5}} = \frac{1}{0.6} = 1.67$$

Amdahl´s law: example 2

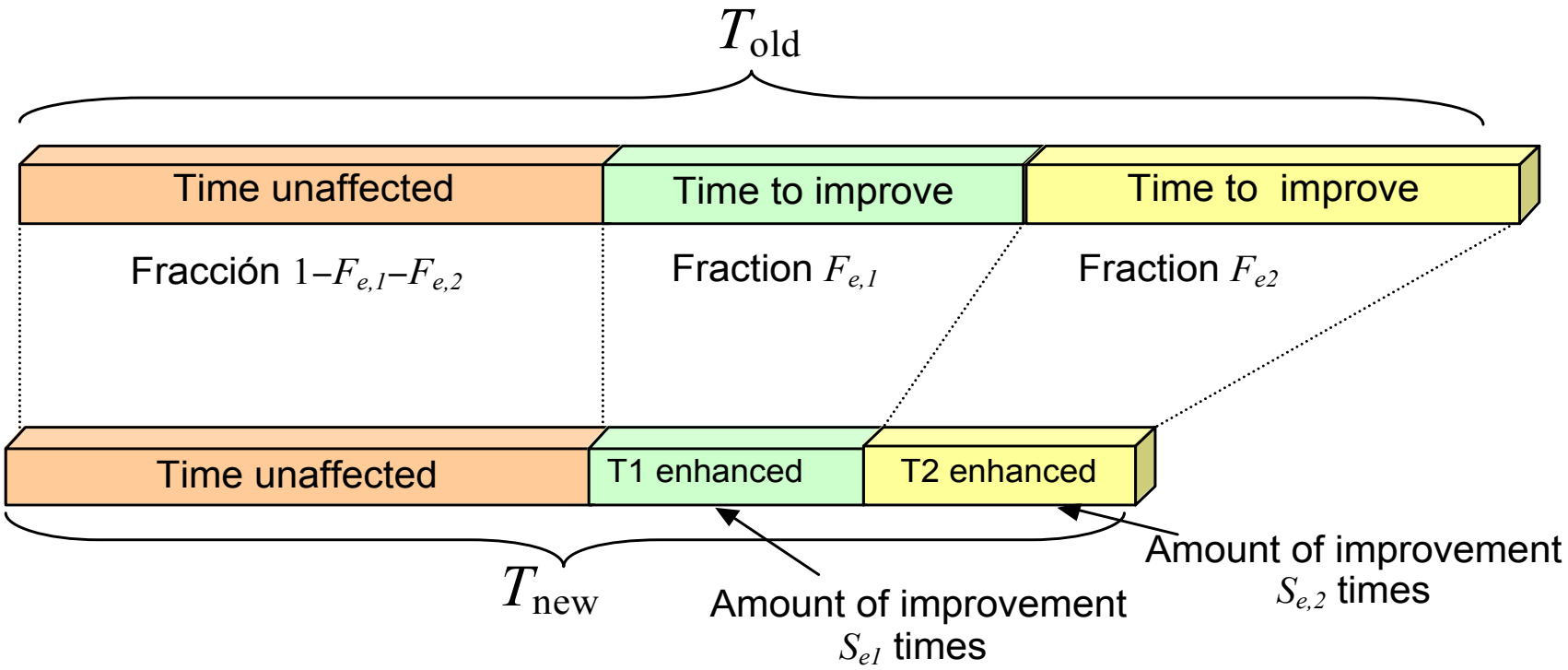
- We are looking for a benchmark to show off the **new floating-point unit** described in the previous example, and we want the overall benchmark to show a **speedup of 3**. One benchmark we are considering **runs for 100 seconds** with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Speedup = 3 = 100 / (Time_FP / 5 + 100 – Time_FP)

Time_FP= 83.33 seconds

$$Speedup_{Overall} = \frac{1}{(1-F_e)+\frac{F_e}{S_e}}; \quad 3 = \frac{1}{(1-F_e)+\frac{F_e}{5}}; \quad 3 = \frac{5}{5-4F_e}; \quad F_e = 0{,}83$$

Amdahl´s law generalization



$$Speedup_{overall} = \frac{1}{1 - \sum_{i=1}^{n} F_{e,i} + \sum_{i=1}^{n} \frac{F_{e,i}}{S_{e,i}}}$$

Amdahl´s law: An exercise

Example: Using the Amdahl's law. Calculate the speed up of the system in both cases:

Case 1: x10 in FP Operation	ARITHMETIC LOGIC UNIT (ALU)			
	INT	**FP SUM**	**FP MUL**	**FP DIV**
Instruction (%)	**70**	**15**	**10**	**5**
Old CPI	**2**	**40**	**60**	**100**
New CPI improved	**2**	**4**	**6**	**10**

Case 2: FP with different improvements	ARITHMETIC LOGIC UNIT (ALU)			
	INT	**FP SUM**	**FP MUL**	**FP DIV**
Instruction (%)	**70**	**15**	**10**	**5**
Old CPI	**2**	**40**	**60**	**100**
New CPI improved	**2**	**2**	**15**	**10**

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

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

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

  - CPI varies between programs on a given CPU

MFLOPs as a Performance Metric

- MFLOPS: Millions of Floating-point Operations Per Second

$$\text{MFLOPS} = \frac{\text{Floating point instruction count}}{\text{Execution time} \times 10^6}$$

The MFLOPS measure, as described by [ref1] , is widely employed because of its efficiency and effectiveness in terms of simplicity and its indicative abilities.

See www.top500.org

[1] Evaluating the Mflops Measure Ran Giladi , Journal IEEE Micro archive Vol 16 Issue 4, 1996, pp 69-75

MIPS vs MFLOPs: An exercise

Example: Compare the performance @ 100 MHz of M1-CISC and M2-RISC. (Also using MIPS and MFLOPs metrics) :

	Alu INT	Mem	FP
IC M1-cisc	$5x10^6$	10^6	10^6
IC M2-risc	$10x10^6$	10^6	10^6
CPI	1	2	3

- The performance of M1-CISC =1,5 x M2-RISC (50% better)
- M1 MIPS < M2 MIPS
- M1 MFLOPs > M2 MFLOPs

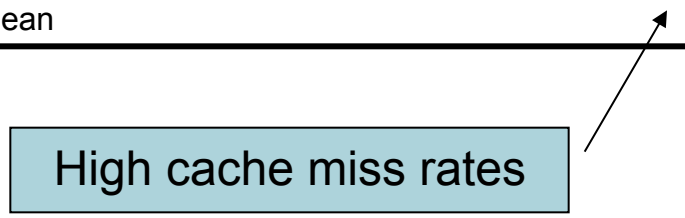
SPEC CPU Benchmark

- 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)

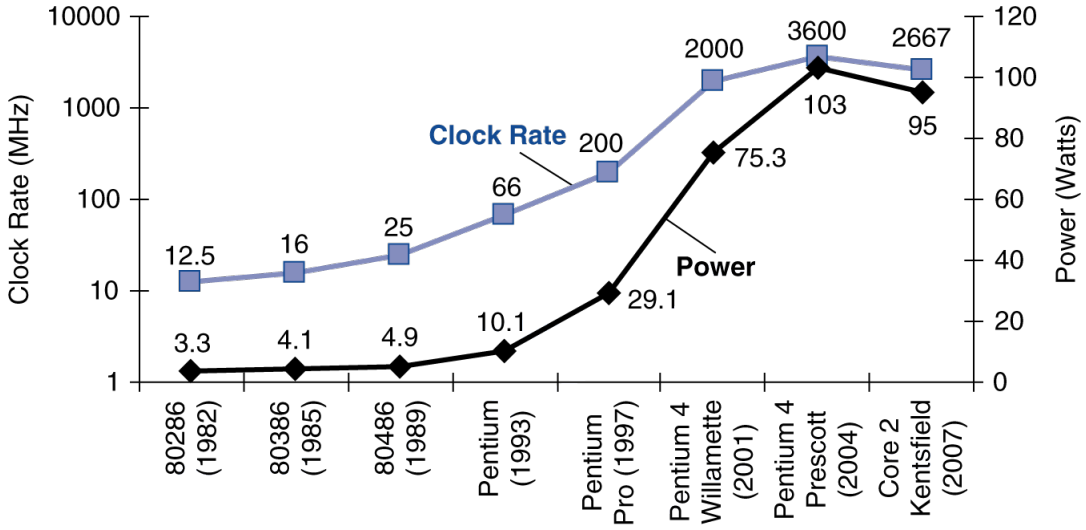
$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

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	0.40	724	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	0.40	837	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



Power Trends



- In CMOS IC technology

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

×30 | 5V → 1V | ×1000

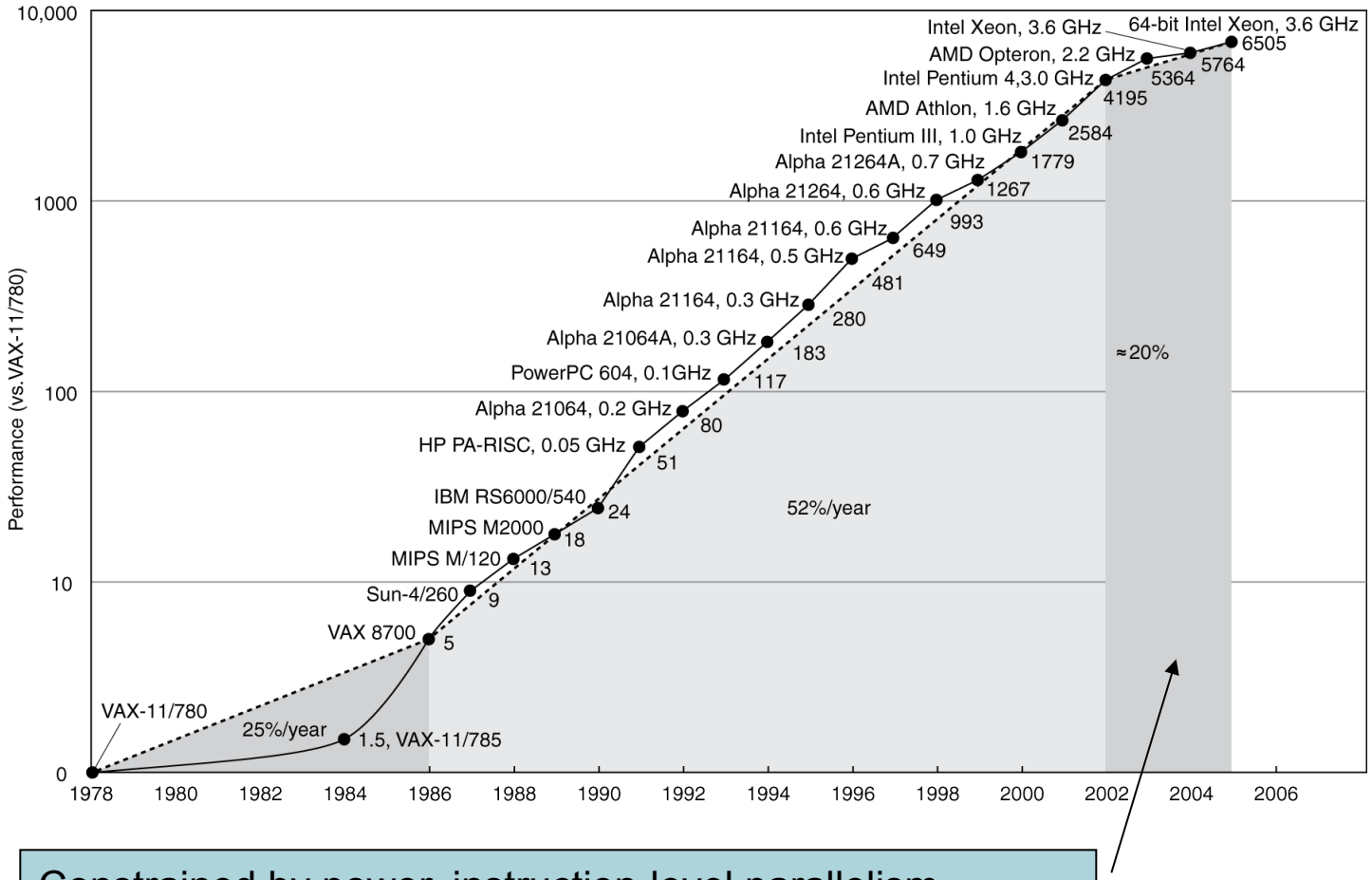
Reducing Power

- 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

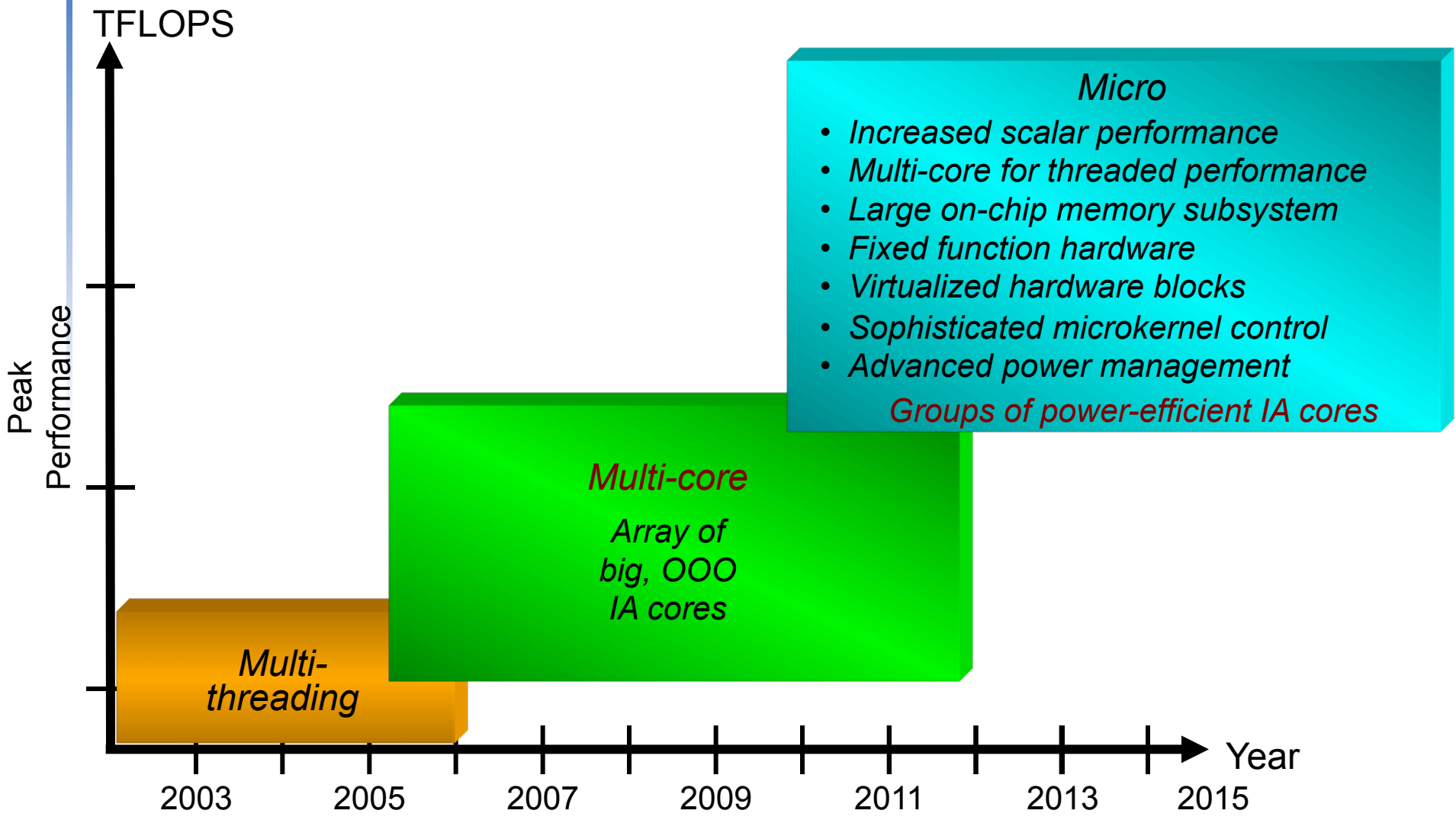


Constrained by power, instruction-level parallelism, memory latency

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

New Era of Computing



SPEC Power Benchmark

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

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

(*) ssj_ops/sec : server side Java operations per second

Fallacy: Low Power at Idle

SPECpower_ssj2008 for X4

Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	920,35	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
∑ssj_ops/ ∑power		493

- Look at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- 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

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

Info Adicional: Fórmulas y relaciones útiles para los ejercicios

T_{CPU} = NI x CPI x T_{CICLO} =

= NroCiclosReloj / FreqReloj

$$Rendimiento(X) = 1 / T_{EJEC}(X)$$

X es n veces más rápido que Y significa:

$$n = \frac{Rendimiento(X)}{Rendimiento(Y)} = \frac{T_{EJEC}(Y)}{T_{EJEC}(X)}$$

Una mejora en el sistema se mide con la:

$$\textit{Aceleración} = \frac{T_{EJ}(\text{sin mejora})}{T_{EJ}(\text{con mejora})} = \frac{R(\text{con m.})}{R(\text{sin m.})}$$

La ***ley de Amdahl*** mide la mejora del rendimiento global de un sistema. Donde:

F_m: Fracción de tiempo mejorada y Acc_m: Aceleración mejorada

$$Acc_{global} = \frac{1}{(1 - F_m) + \frac{F_m}{Acc_m}}$$

La ***ley de Amdahl*** para varias mejoras:

$$Acc_{global} = \frac{1}{1 - \sum_{i=1}^{n} F_{m,i} + \sum_{i=1}^{n} \frac{F_{m,i}}{Acc_{m,i}}}$$

$$MIPS = NI / (T_{CPU} \times 10^6) = NI / (NI \times CPI \times T_{Ciclo} \times 10^6) = Frec / (CPI \times 10^6)$$