CCSYA

Fundamentals of Computer Architectures

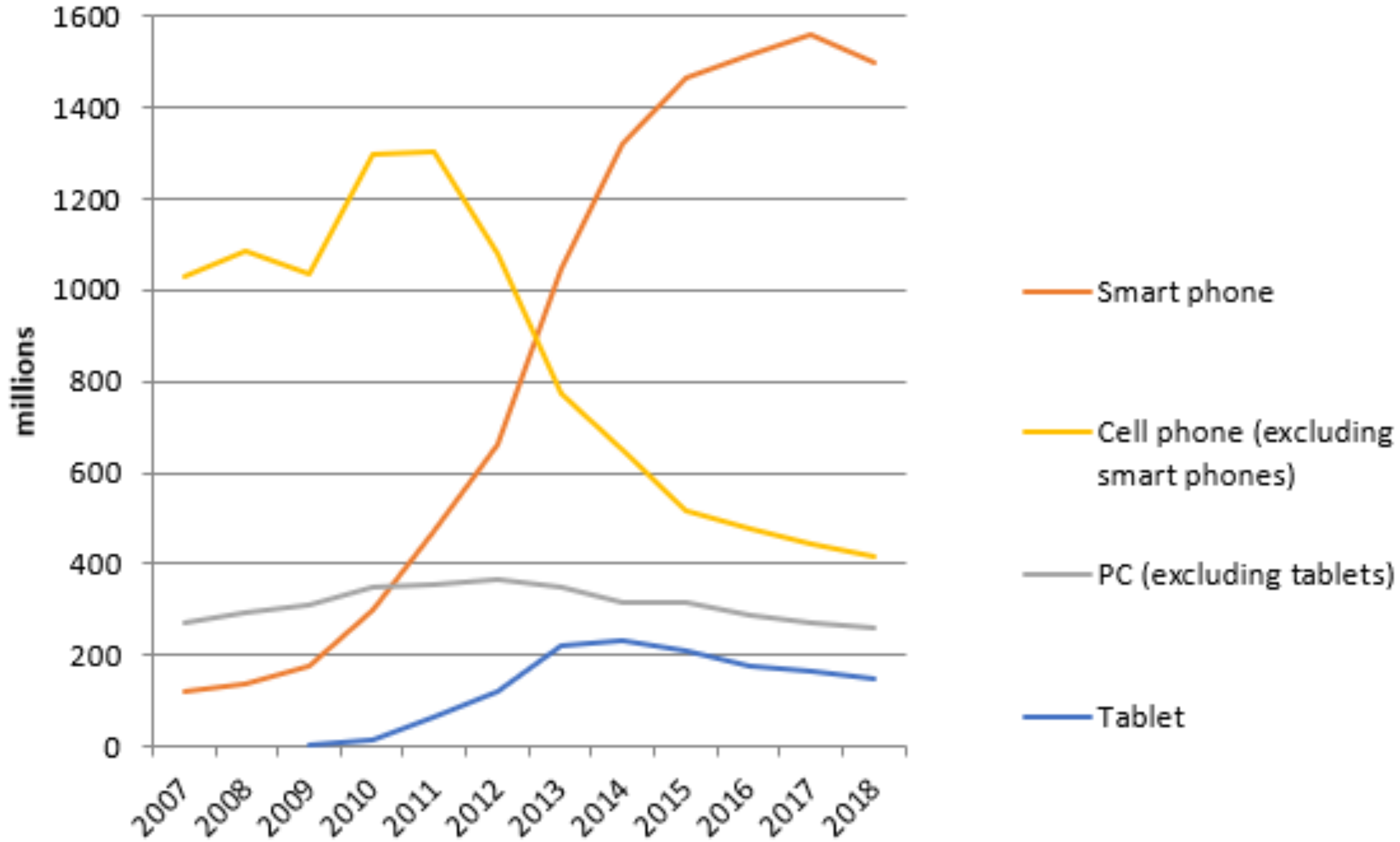
Departamento de Engenharia Informática
Instituto Superior de Engenharia do Porto

Luís Nogueira (lmn@isep.ipp.pt)

The Computer Revolution

- **Progress in computer technology**
  - Advancing technology makes more complex and powerful chips feasible to manufacture
  - Now affects almost every aspect of our society
- **Makes novel applications feasible**
  - Computers in automobiles, cell phones
  - Human genome project, World Wide Web
  - Search engines, …
- **Computers are pervasive**
  - Personal computers, server computers, supercomputers
  - Personal mobile devices, embedded devices

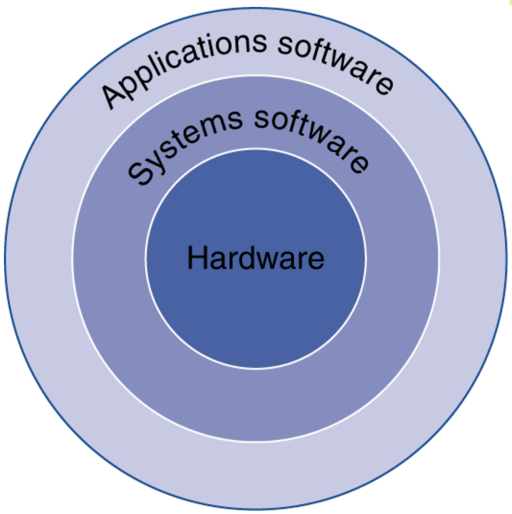
The PostPC Era



The Computer Revolution

- **Difficult to predict exactly what level of cost/performance computers will have in the future**
  - It's a safe bet that they will be much better than they are today
- **To participate in these advances, computer designers and programmers must understand a wider variety of issues**
  - The parallel nature of processors
  - The hierarchical nature of memories
  - Energy efficiency of their programs
  - …

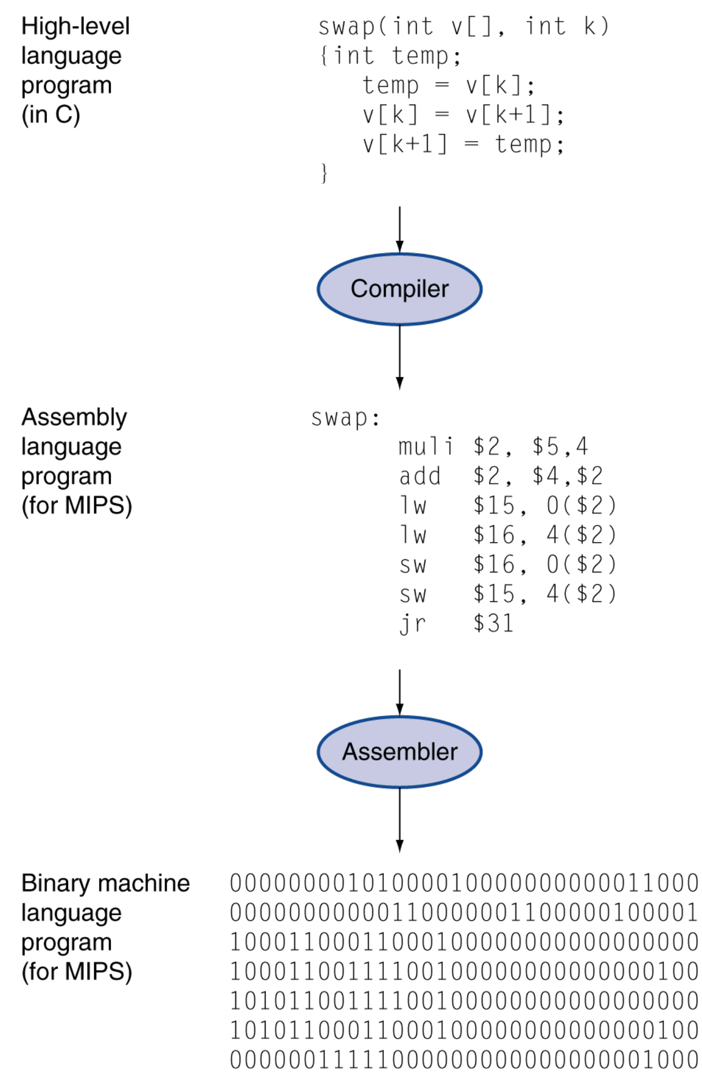
Below Your Program



- **Application software**
  - Usually 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



Translating and Starting a Program

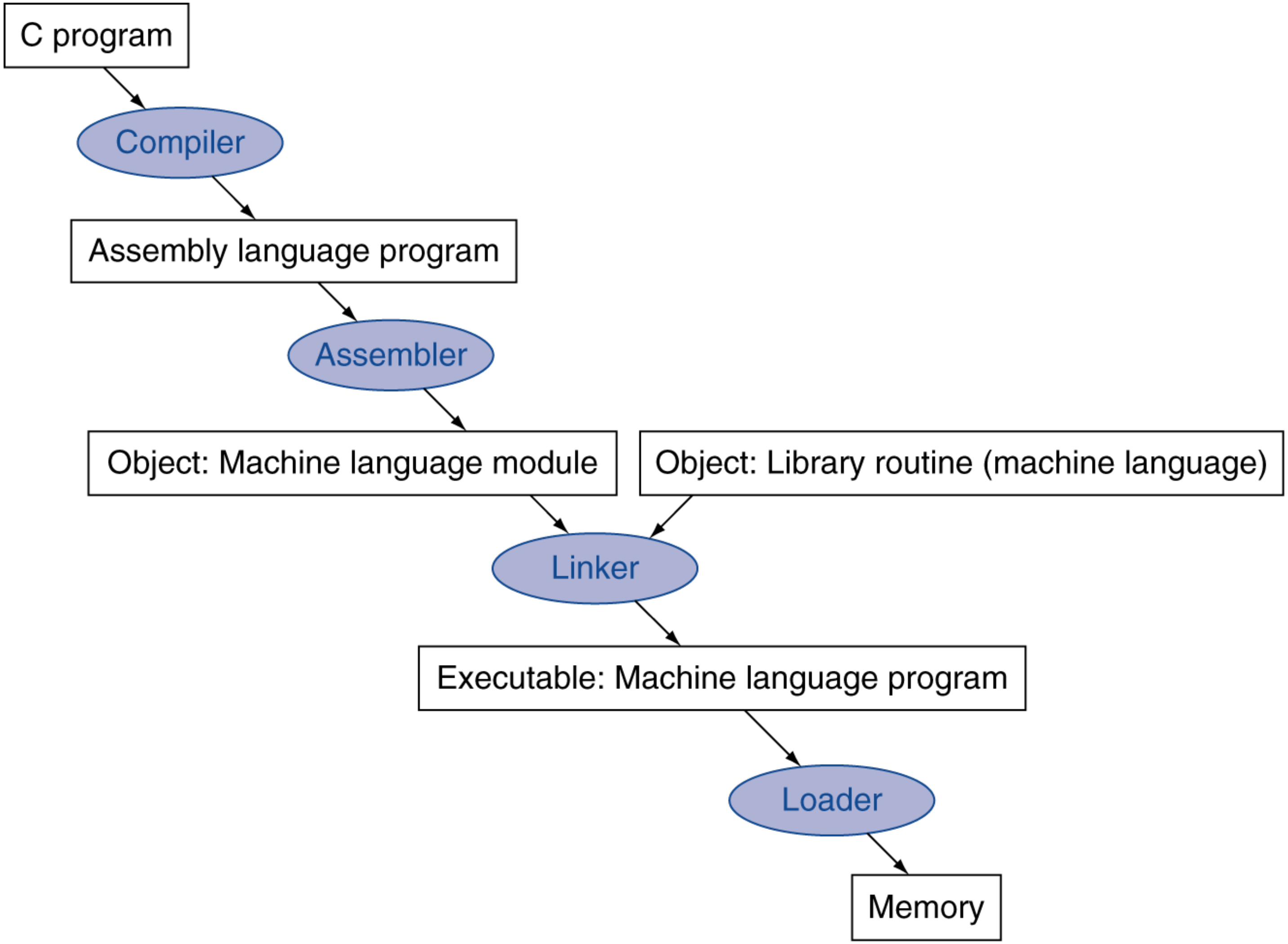
- **Four hierarchical steps when transforming a HLL program in a file on disk and then into a process running on a computer:**
  1. Compiler step
  2. Assembler step
  3. Linker step
  4. Loader step

- **Some systems combine these steps to reduce translation time, but these are the logical four steps that programs go through**

Translating and Starting a Program

- **The compiler transforms the high-level language program to an assembly language program**
  - A symbolic form of what the machine understands
- **The assembler turns the assembly language program into an object file**
  - Which is a combination of machine language instructions, data, and information needed to place instructions properly in memory
- **The linker combines independently assembled object files and resolves all undefined labels into an executable file**
- **The loader places an executable file in main memory so that it is ready to execute**

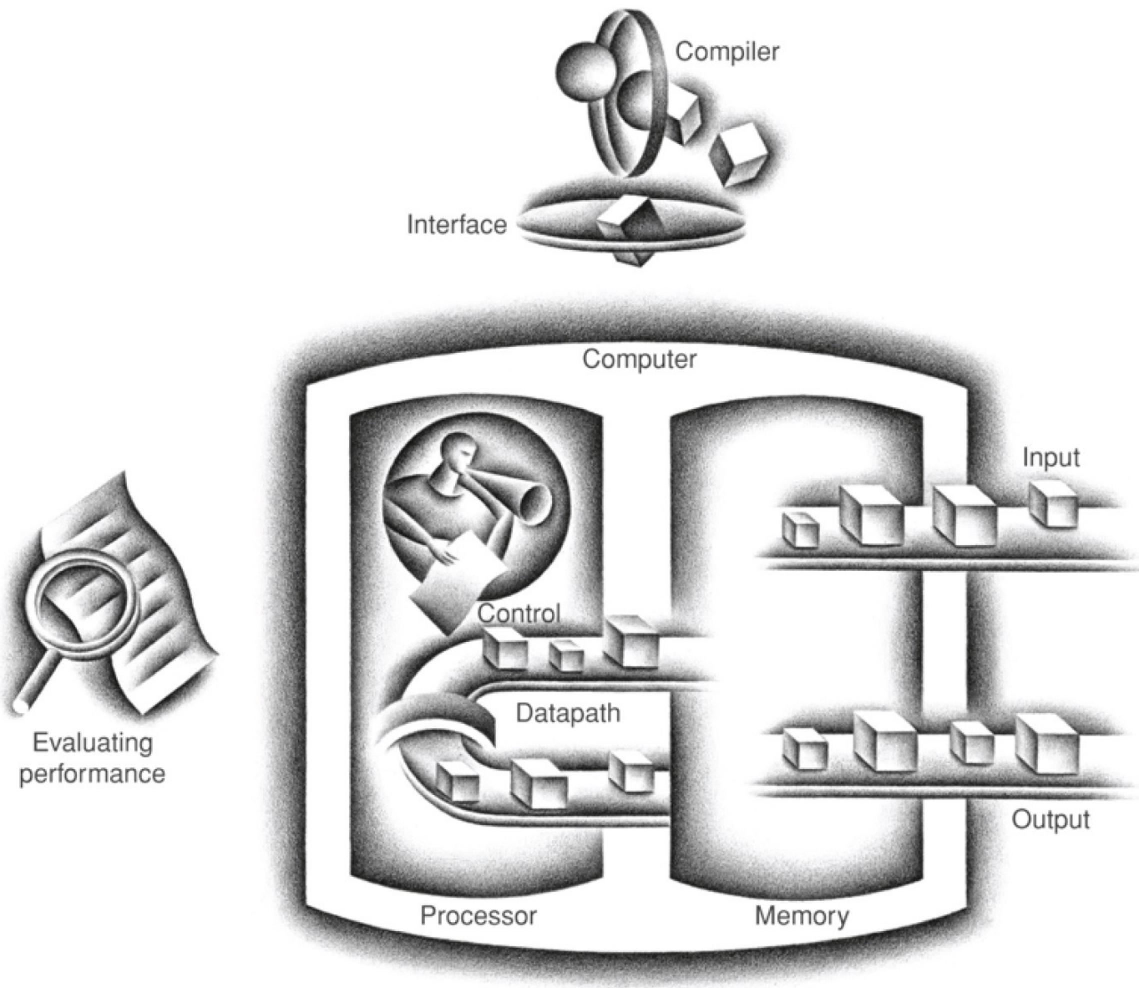
Translating and Starting a Program



Abstractions

- **Abstraction helps us deal with complexity**
  - Hide lower-level detail
- **Instruction set architecture (ISA)**
  - The hardware/software interface
  - One of the most important abstractions in computing
  - Permits multiple **implementations** that may vary in performance, physical size, cost, among other things
- **Application binary interface (ABI)**
  - The ISA plus system software interface

Components of a Computer



Components of a Computer

- The processor gets instructions and data from memory
- Input writes data to memory, and output reads data from memory
- Control sends the signals that determine the operations of the datapath, memory, input, and output
- How these functions are performed is the primary topic of this course

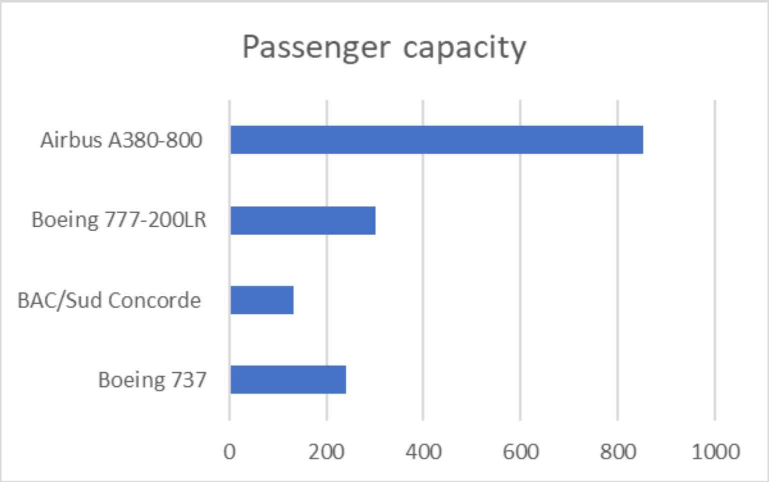
Understanding performance

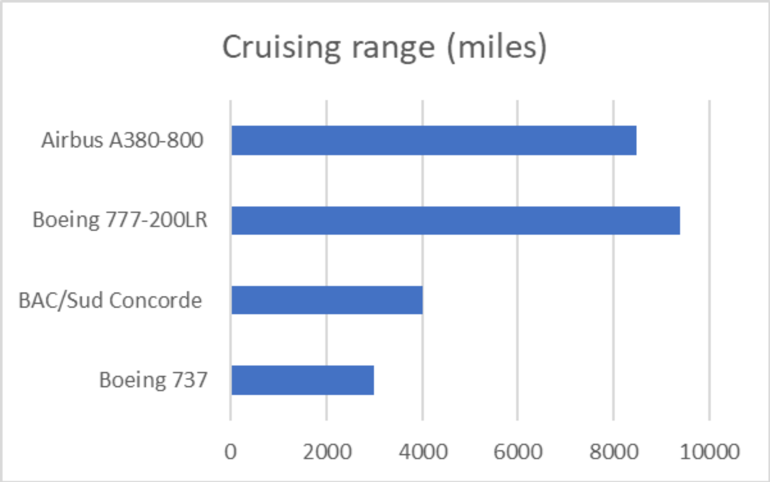
- **When trying to choose among different computers, performance is one of the first key attributes that comes to mind**
  - Accurately measuring and comparing different computers is critical to purchasers and therefore to designers

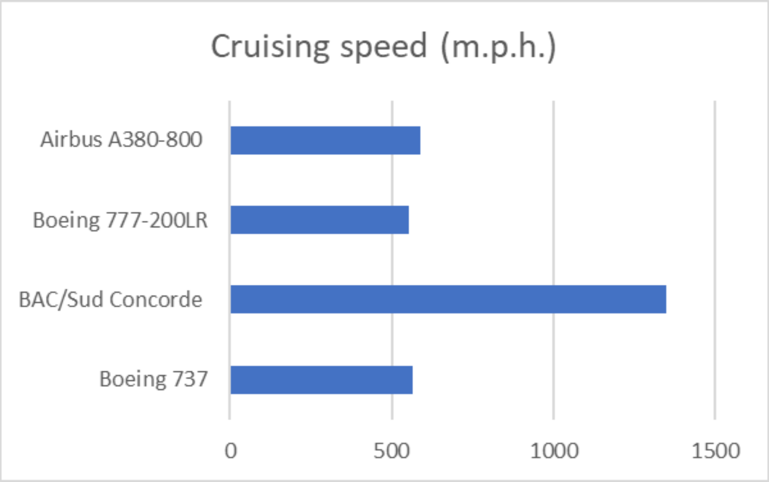
- **Assessing the performance of computers can be quite challenging**
  - The wide range of performance improvement techniques employed by hardware designers have made performance assessment much more difficult

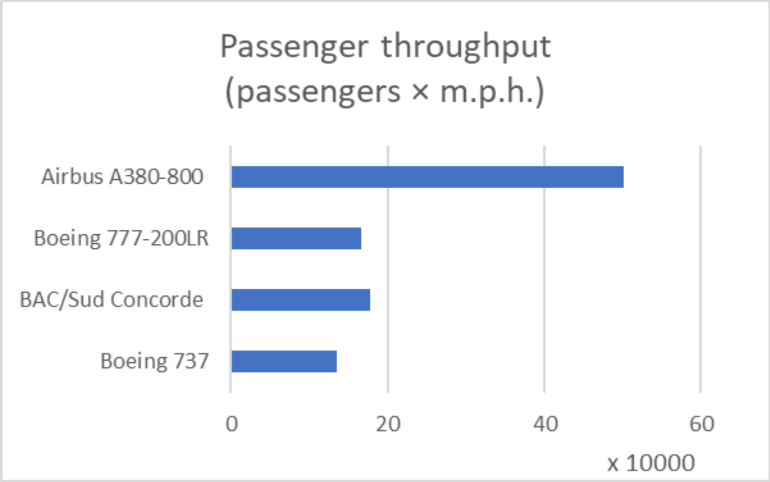
Defining Performance

- **Which airplane has the best performance?**









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

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

- **We can relate performance and execution time as** **Performance = 1/Execution Time**
  - To maximize performance, we want to minimize response time or execution time for some task

- **"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

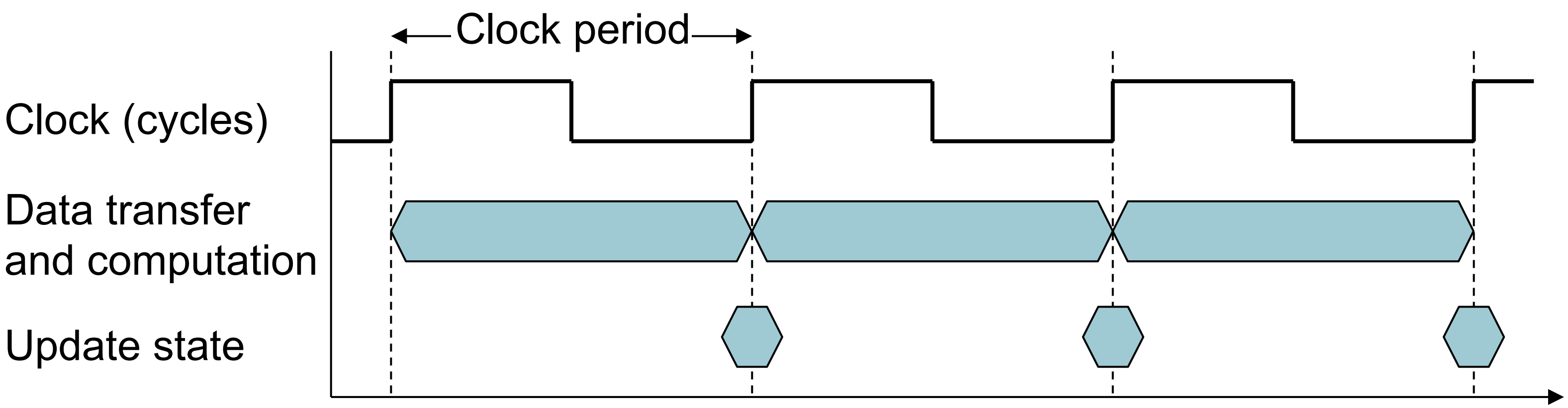
- **Time can be defined in different ways, depending on what we count**

- **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 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
  - Increasing clock rate

- **A hardware designer must often trade off clock rate against cycle count**
  - Many techniques that decrease the number of clock cycles may also increase the clock cycle time

Instruction Performance

- **Previous performance equations did not include any reference to the number of instructions needed for the program**

- **Clearly, the execution time must depend on the number of instructions in a program**

- **The number of clock cycles required for a program can be written as**

Clock Cycles = Instruction Count x Average Clock Cycles per Instruction

Clock Cycles Per Instruction

- **CPI is an average of all the instructions executed in the program**
  - Different instructions may take different amounts of time depending on what they do

- **CPI provides one way of comparing two different implementations of the same ISA**
  - Since the number of instructions executed for a program will, of course, be the same

Instruction Count and CPI

- **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

The CPU Performance Equation

- **The performance equation in terms of:**
  - Instruction count
  - CPI
  - Clock cycle time

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- **Particularly useful because it separates the three key factors that affect performance**

Pitfall: MIPS as a Performance Metric

- **MIPS: Millions of Instructions Per Second**
  - Doesn't account for differences in:
    - ISAs between computers
    - Complexity between instructions
  - CPI varies between programs on a given CPU

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

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

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, Clock rate

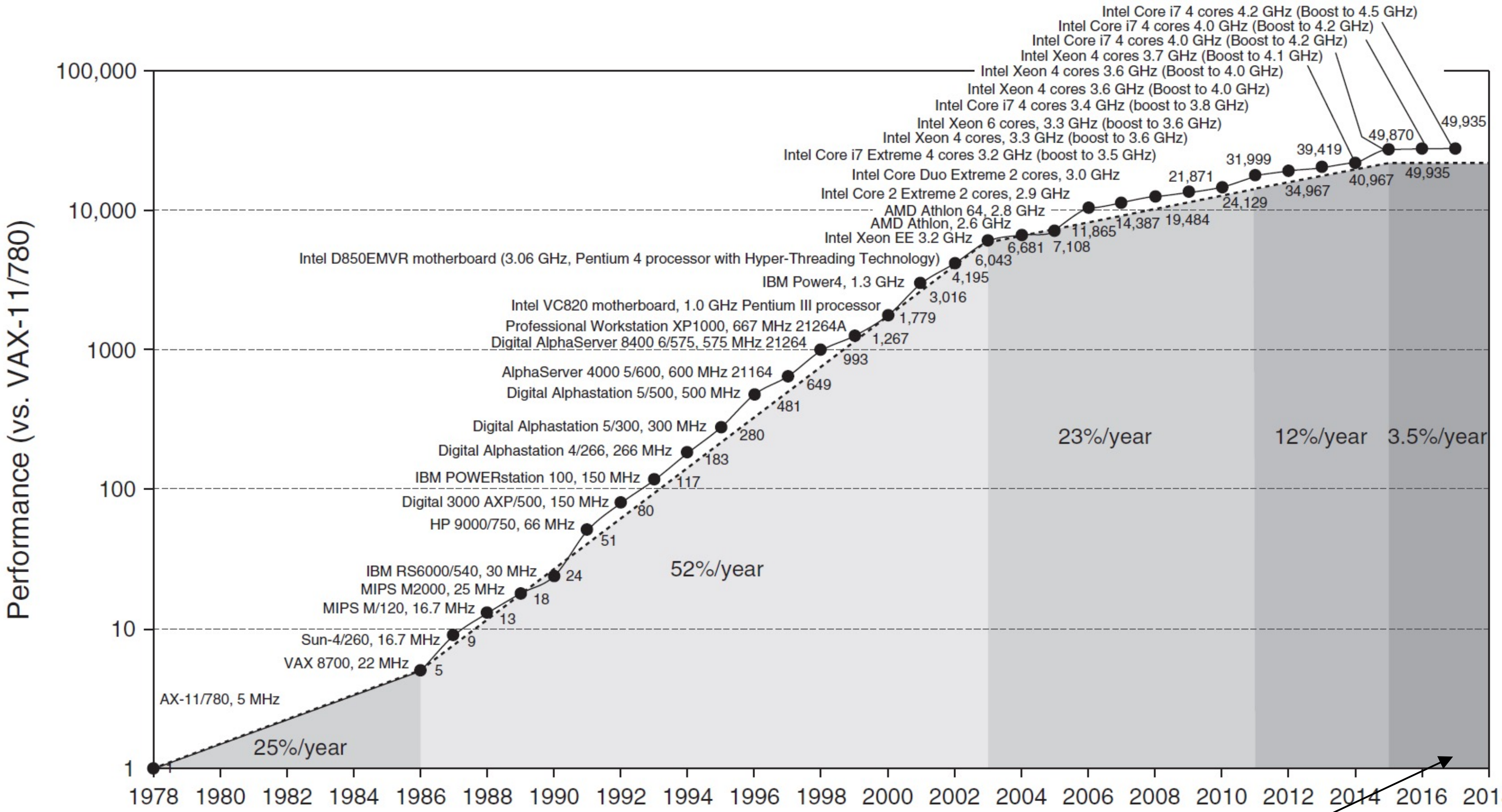
Growth in Processor Performance

- **Prior to the mid-1980s, processor performance growth was largely technology-driven and averaged about 25% per year**
  - i.e., doubling performance every 3.5 years
- **Starting in 1986, we saw an increase in growth to about 52% (doubling every 2 years)**
  - Attributable to more advanced architectural and organizational ideas typified in RISC architectures
  - e.g., use of caches and the exploitation of instruction-level parallelism

Growth in Processor Performance

- **Since 2003, growth of uniprocessor performance slowed to about 23% per year until 2011**
  - Limits of power, available instruction-level parallelism, and long memory latency
- **From 2011 to 2015, the improvement was less than 12% (doubling every 8 years)**
  - In part due to the limits of parallelism of Amdahl’s Law
  - Since 2015, improvement has been just 3.5% (doubling every 20 years!)

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

The Power Wall

- **Both clock rate and power increased rapidly for decades, and then flattened off recently**

- **We have run into the practical power limit for cooling commodity microprocessors**

- **Energy efficiency has replaced die area as the most critical resource of microprocessor design**
  - Power delivery and dissipation limits have emerged as a key constraint in the design of microprocessors

Power Trends

- **In CMOS IC technology the dynamic energy depends on the capacitive loading of each transistor and the voltage applied**

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

×30

$5\text{V} \rightarrow 1\text{V}$

×1000

- **Energy, and thus power, can be reduced by lowering the voltage**
  - In 20 years, voltages have gone from 5 V to 1 V, which is why the increase in power is only 30 times while clock rates grown by a factor of 1000

Reducing Power

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

Half the power

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

- **Further lowering of the voltage appears to make the transistors too leaky**
  - We can't reduce voltage further
  - We can't remove more heat

- **How else can we improve performance?**

Multiprocessors

- **Multicore microprocessors**
  - More than one processor per chip
- **Requires explicitly parallel programming**
  - Harder to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization
- **Compare with instruction level parallelism**
  - Hardware executes multiple instructions at once
  - Hidden from the programmer

Pitfall: Amdahl's Law

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

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

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

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

- Can't be done!

- **Corollary: make the common case fast**
  - In everyday life this concept also yields what we call the law of diminishing returns

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

External References

- **Wikipedia: History of Computing Hardware**
  - https://en.wikipedia.org/wiki/History_of_computing_hardware

- **Computer History Museum: Timeline of Computer History**
  - https://www.computerhistory.org/timeline/computers

- **Singularity Prosperity: The History of Computing**
  - https://www.youtube.com/watch?v=-M6lANfzFsM