#6 : Performance

Computer Architecture 2019/2020

Ricardo Rocha

Computer Science Department, Faculty of Sciences, University of Porto

Slides based on the book

'Computer Organization and Design, The Hardware/Software Interface, 5th Edition

David Patterson and John Hennessy, Morgan Kaufmann'

Section 1.6

Understanding Performance

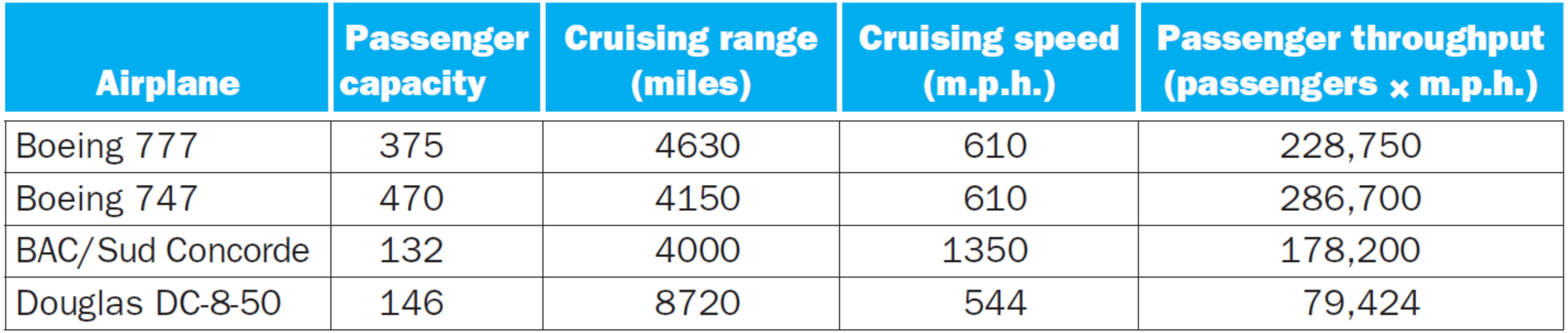
When trying to choose among different computers, **performance is one of the first key attributes** that comes to mind.

Assessing the performance of computers can be **quite challenging** since the scale and intricacy of modern software systems, together with the wide range of performance improvement techniques employed by hardware designers, have made performance assessment much more difficult.

Understanding Performance

When we say one computer system has better performance than another, what do we mean?

An analogy with passenger airplanes shows how subtle the question of performance can be. Which airplane has the best performance?



Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h.)	Passenger throughput (passengers × m.p.h.)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

FIGURE 1.14 The capacity, range, and speed for a number of commercial airplanes. The last column shows the rate at which the airplane transports passengers, which is the capacity times the cruising speed (ignoring range and takeoff and landing times).

Understanding Performance

The **algorithm** determines both the number of source-level statements and the number of I/O operations executed.

The **programming language, compiler and architecture** determines the number of machine instructions executed per each source-level statement.

The **processor and memory system** determines how fast instructions are executed.

The **I/O system (hardware and operating system)** determines how fast I/O operations are executed.

Performance Metrics

There are 2 distinct classes of performance metrics:

- **Performance metrics for processors** – assess the performance of a processing unit, normally done by measuring the speed or the number of operations that it does in a certain period of time
- **Performance metrics for parallel applications** – assess the performance of a parallel application, normally done by comparing the execution time with multiple processing units against the execution time with just one unit

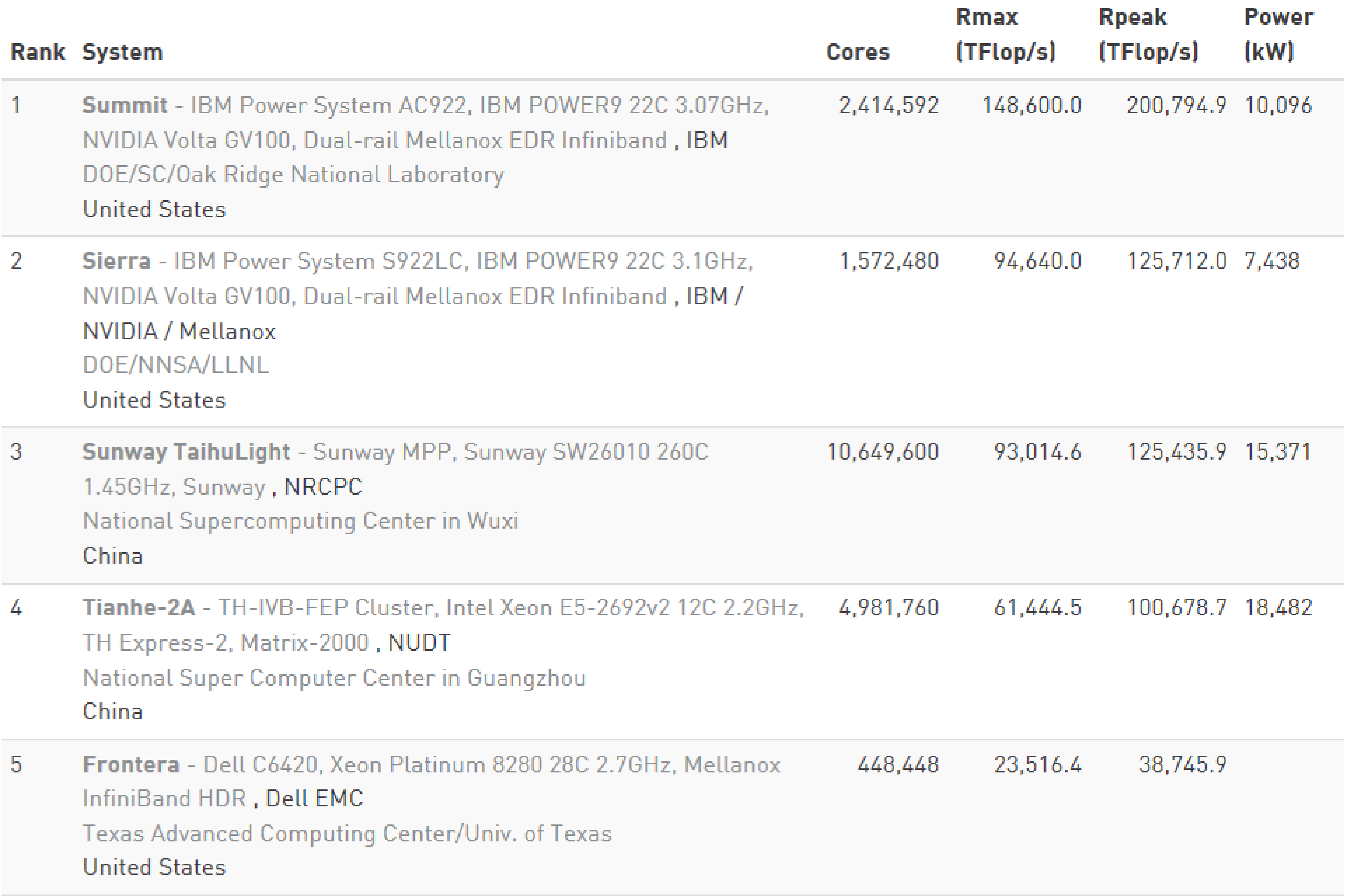
Here, we are mostly interested in metrics that measure the performance of processors.

Performance Metrics for Processors

Some of the best known metrics are:

- **MIPS** – Millions of Instructions Per Second
- **FLOPS** – FLoating point Operations Per Second
- **SPECint** – SPEC (Standard Performance Evaluation Corporation) benchmarks that evaluate processor performance on integer arithmetic (first release in 1992)
- **SPECfp** – SPEC benchmarks that evaluate processor performance on floating point operations (first release in 1989)
- **Whetstone** – synthetic benchmarks to assess processor performance on floating point operations (first release in 1972)
- **Dhrystone** – synthetic benchmarks to assess processor performance on integer arithmetic (first release in 1984)

Top500 Supercomputers – List for June 2019



Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
2	**Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
3	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
4	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482
5	**Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC Texas Advanced Computing Center/Univ. of Texas United States	448,448	23,516.4	38,745.9	

Response Time and Throughput

If running a program on several desktop computers, we can say that the faster one is the one that gets the job done first. The interest is in reducing **response time or execution time**, i.e., how long it takes to do a task.

If running a datacenter with several servers running jobs submitted by many users, we can say that the faster one is the one that completes the most jobs during a day. The interest is in increasing **throughput or bandwidth**, i.e., the total amount of work done per time unit.

In most cases, we will need different performance metrics as well as different applications to benchmark response time versus throughput.

Execution Time Performance

We can relate performance and execution time as follows:

$$Performance = \frac{1}{ExecutionTime}$$

We say that a computer A is N times faster than a computer B (or that computer B is N times slower than computer A) if:

$$\frac{Performance_A}{Performance_B} = \frac{ExecutionTime_B}{ExecutionTime_A} = N$$

Measuring Execution Time

Elapsed time – total response time to complete a task

- Includes everything – disk accesses, memory accesses, input/output (I/O) activities, operating system overhead, idle time

CPU time – CPU time spent processing a given task

- Excludes time spent waiting for I/O or running other programs

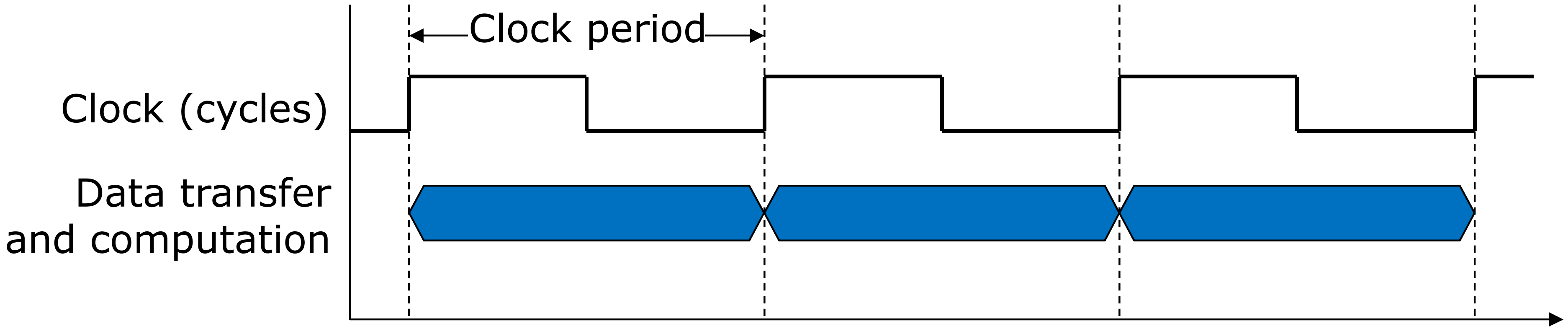
CPU time can be further divided in:

- **User CPU time** – time spent in the program itself (user's code)
- **System CPU time** – time spent in the operating system performing tasks on behalf of the program (system calls)

CPU Clocking

Sometimes, it might be convenient to think about performance in other metrics. For example, by using a measure that relates to **how fast the hardware can perform basic functions**.

Almost all computers use a **clock** that determines **when events take place in the hardware**. These discrete time intervals are called **clock cycles (or ticks, clock ticks, clocks, cycles)**.



CPU Clocking

The duration of a complete clock cycle is the **clock period** and the number of cycles per second is the **clock rate (or clock frequency)**, which is the inverse of the clock period.

$$ClockRate\,(Hz) = \frac{1}{ClockPeriod\,(s)}$$

Clock period (duration of a clock cycle)

- e.g., 250ps (picoseconds) = 0.25ns (nanoseconds) = 250 × 10^{-12}s (seconds)

Clock rate (cycles per second)

- e.g., 4.0GHz = 4000MHz = 4.0 × 10^{9}Hz = 1 / (250 × 10^{-12}s)

CPU Clocking

A simple formula relates clock cycles to CPU time for a given program P:

$$CPUTime_P = CPUClockCycles_P \times ClockPeriod$$

Or alternatively, because clock rate and clock period are inverses:

$$CPUTime_P = \frac{CPUClockCycles_P}{ClockRate}$$

Instruction Count, CPI and IPC

Another way to think about CPU time is that it equals the number of instructions executed multiplied by the average time per instruction.

- **Instruction count (IC)** is determined by the program, the compiler and the instruction set architecture
- **Clock cycles per instruction (CPI)** is determined by CPU hardware and is the average of all instructions executed in the program
- **Instructions per clock cycle (IPC)** is an alternative designation, which is the inverse of the CPI

$$CPUClockCycles_P = InstructionCount_P \times CPI_P$$
$$= \frac{InstructionCount_P}{IPC_P}$$

Performance Equation

$$CPUTime_P = InstructionCount_P \times CPI_P \times ClockPeriod$$

$$= \frac{InstructionCount_P \times CPI_P}{ClockRate}$$

$$= \frac{InstructionCount_P}{IPC_P \times ClockRate}$$

Performance can be improved by reducing CPU time, i.e., by:

- **Decreasing instruction count**
- **Decreasing CPI (or increasing IPC)**
- **Decreasing clock period (or increasing clock rate)**

Example I

A compiler is trying to decide between two different code sequences for a computer requiring the following instruction counts.

Class of Instruction	A	B	C
CPI for class	1	2	3
IC in Sequence 1	2	1	2
IC in Sequence 2	4	1	1

Which sequence has more instructions?

- Sequence 1 (S1) has 2+1+2 = 5 instructions
- Sequence 2 (S2) has 4+1+1 = 6 instructions

Which will be faster?

- S1 takes (2x1)+(1x2)+(2x3) = 10 cycles
- S2 takes (4x1)+(1x2)+(1x3) = 9 cycles

What is the CPI for each sequence?

- CPI(S1) = 10/5 = 2.0
- CPI(S2) = 9/6 = 1.5

Example II

Consider two computers with the same instruction set architecture. Computer A has a clock period of 250ps and a CPI of 2.0 for a program P. Computer B has a clock period of 500ps and a CPI of 1.2 for the same program P. Which computer is faster for this program and by how much?

$$\begin{aligned} CPUTime_{PA} &= IC_P \times CPI_{PA} \times ClockPeriod_A \\ &= IC_P \times 2.0 \times 250ps \\ &= IC_P \times 500ps \\ CPUTime_{PB} &= IC_P \times CPI_{PB} \times ClockPeriod_B \\ &= IC_P \times 600ps \end{aligned}$$

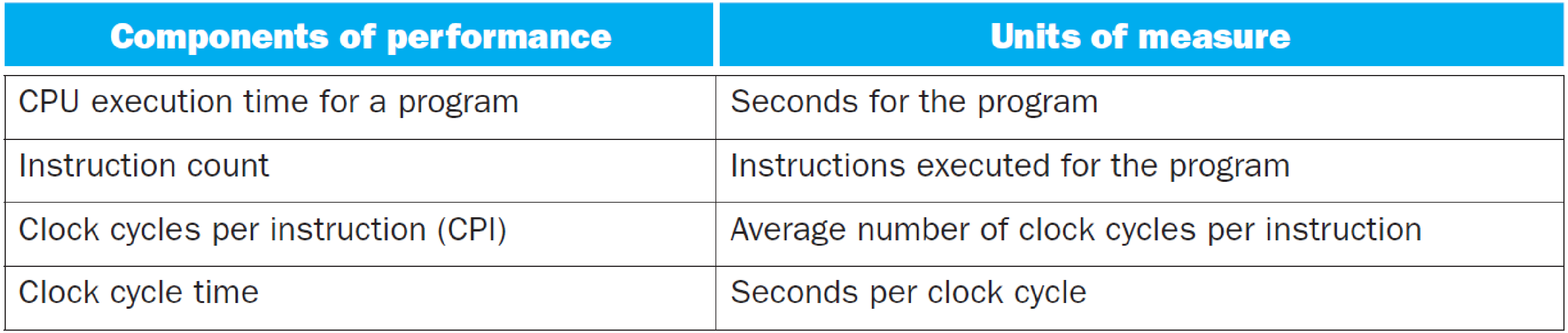
Computer A is 1.2 (600/500) times faster than computer B for program P.

Why RISC is Good?

$$CPUTime_P = InstructionCount_P \times CPI_P \times ClockPeriod$$

While RISC binaries files are larger than those of non-RISC architectures (instruction count increases), the CPI (clock cycles per instruction) and the clock period (set to the length of the longest pipeline stage) decrease further. As a result, performance is greatly increased.

Performance Summary



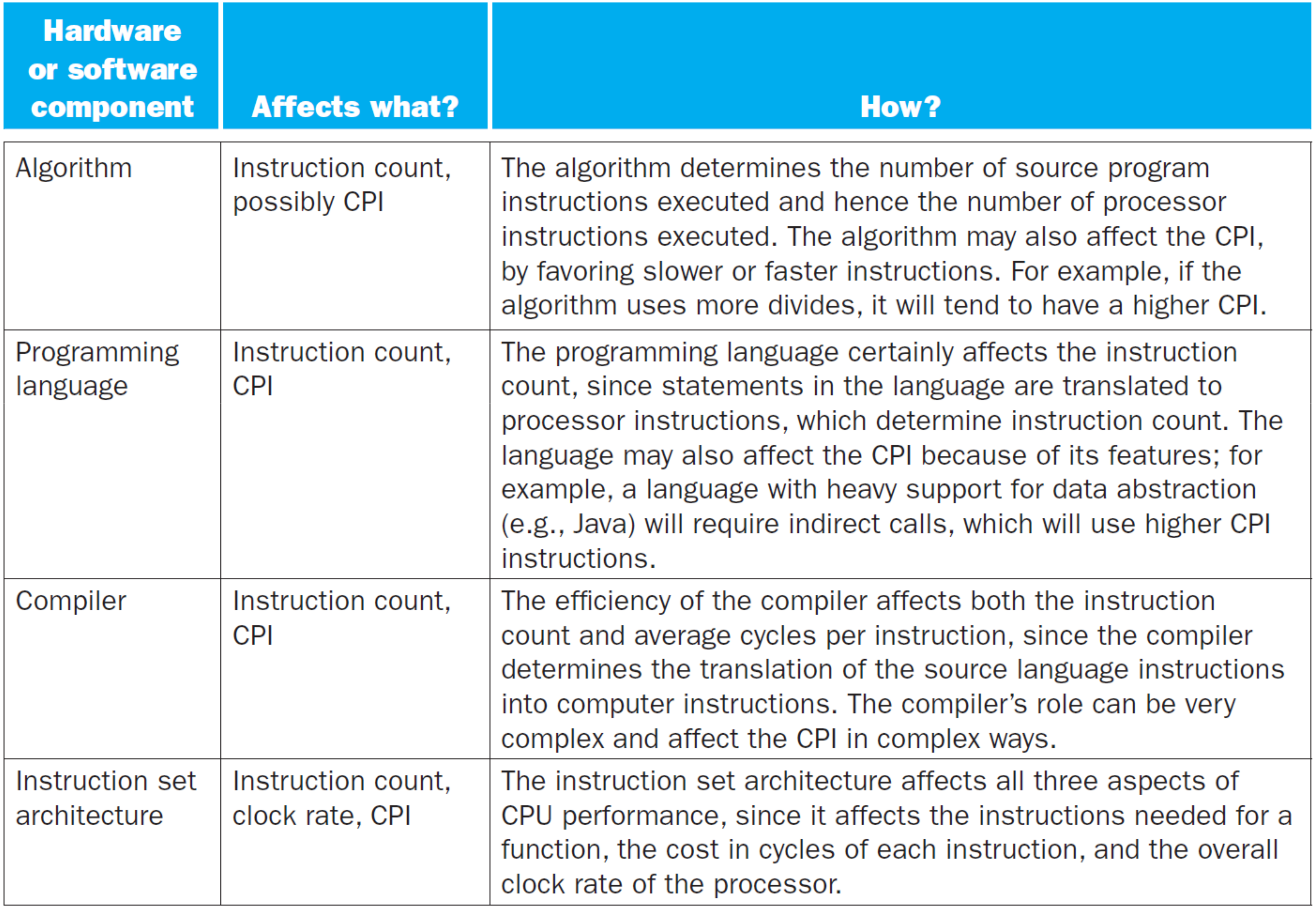
Components of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instruction (CPI)	Average number of clock cycles per instruction
Clock cycle time	Seconds per clock cycle

FIGURE 1.15 The basic components of performance and how each is measured.

$$CPUTime = InstructionCount \times CPI \times ClockPeriod$$

$$= \frac{Instructions}{Program} \times \frac{ClockCycles}{Instruction} \times \frac{Seconds}{ClockCycle}$$

Performance Summary



Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.