Course Introduction

Prof. Salvatore Orlando - DAIS, Campus Scientifico – orlando@unive.it

Website

e-learning platform:

http://moodle.unive.it/

The name of the course is: **ET4018**

- You can access the platform by using the credentials of your personal area.

- To enroll in the course, a *keyword* is requested: `introcod`

- The information available on the moodle website are:

  - Lecture notes

  - News, Forum

  - Exam results

  - …

Office hours

Make an appointment by email (orlando@unive.it)

DAIS, Campus Scientifico, Zeta Building

Via Torino 155

Mestre Venezia

Room Zeta-B09

Wed afternoon (if compatible with the professor's schedule)

Textbooks and other materials

- Think Python. How to Think Like a Computer Scientist (2e). Allen Downey. Green Tea Press.
  - http://greenteapress.com/thinkpython2/thinkpython2.pdf
  - The interactive version of the same book: https://runestone.academy/runestone/static/thinkcspy/index.html

- *(interactive) lecture notes.*

Course contents

- A very small introduction on computer fundamentals:
    - How computers store and transmit data
    - How computer hardware and software operate on data
    - How humans use computers to solve problems

- The main topics of the course include
    - notion of computation and problem solving
    - the Python3 language
    - simple algorithms and data structures
    - development of funny software

- If we have time, some topics regarding history of computer science, and social/ethic issues in computing.

Software for coding in Python3

- During the course, we will use several tools
- Install **Anaconda Navigator** and **Anaconda Prompt** on your computer:
  - https://www.anaconda.com/download/
  - **Install Python 3.7 version** (64 bit)
  - **Instructions:**
    - https://docs.anaconda.com/anaconda/install/windows (Windows)
    - https://docs.anaconda.com/anaconda/install/mac-os#macos-graphical-install (Mac-OS)
- Install also a text editor for coding
  - The multi-platform Atom (http://atom.io)
  - Notepad++, only for Windows (http://notepad-plus-plus.org)
- **Repl.it** provides **a cloud coding environment** (to write code stored and run on remote computer systems)
  - https://repl.it/languages/python3

Exam

The exam will be written and organized into two parts:

- The first brief one is concerned with a set of open questions, aiming at testing the proficiency of the student with respect to the various topics of the course, and the specific technical terminology

- The second part of the exam is related to the skill assessment, through the solution of exercises on the course subjects, in particular by solving small problems by coding

A Very Short Introduction to Computer Science

Study Computer Science to Think Computationally

- Computer science is having a revolutionary impact on scientific research and discovery.

- Simply put, it is nearly impossible to do scholarly research in any scientific or engineering discipline without an ability to think computationally.

- The impact of computing extends far beyond science, however, affecting all aspects of our lives. To flourish in today's world, everyone needs computational thinking

Computing systems

A dynamic entity that interacts with its environment

- Hardware The physical elements of a computing system (printer, circuit boards, wires, keyboard...)

- Software The programs that provide the instructions for a computer to execute

- Information/Data that the computing system manages

Understanding a computing system by abstraction

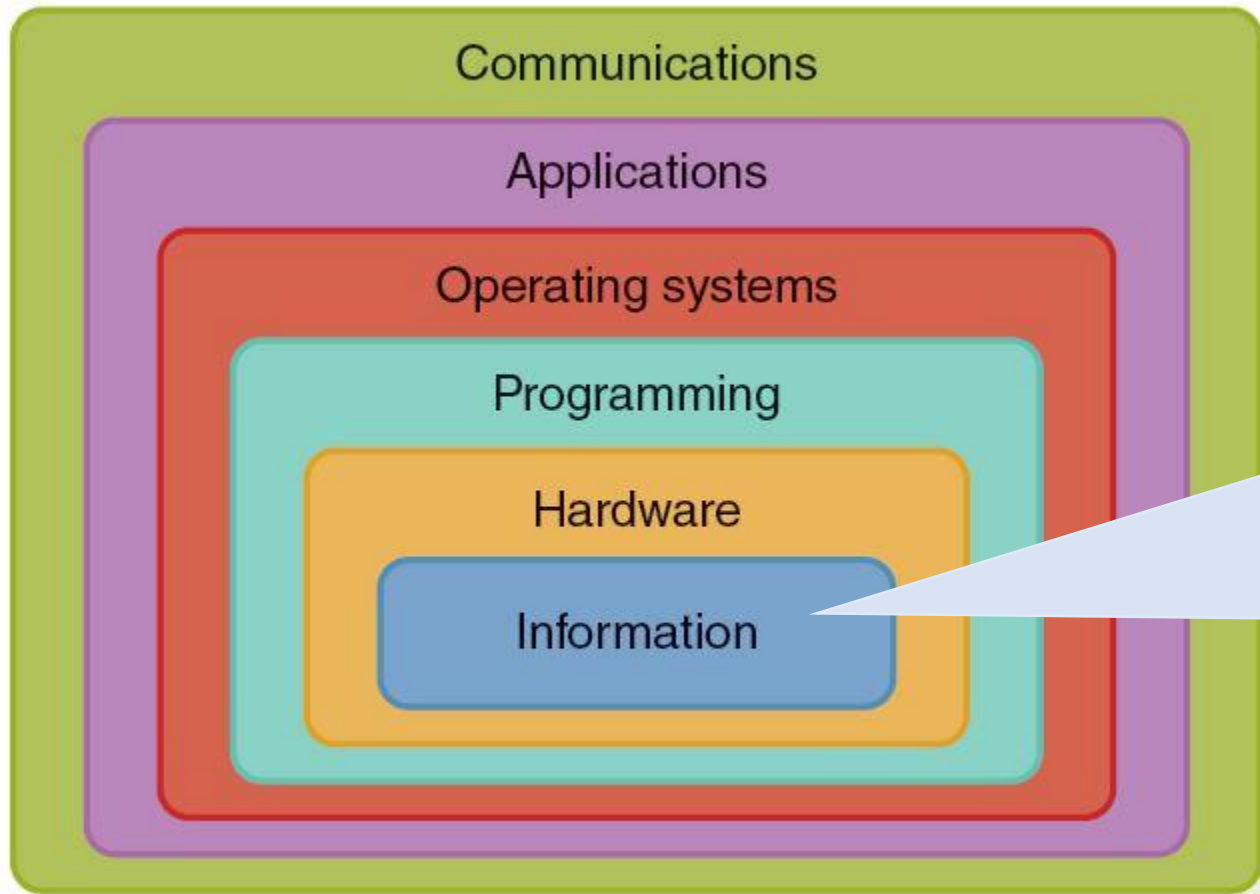- Abstraction A mental model that removes complex details

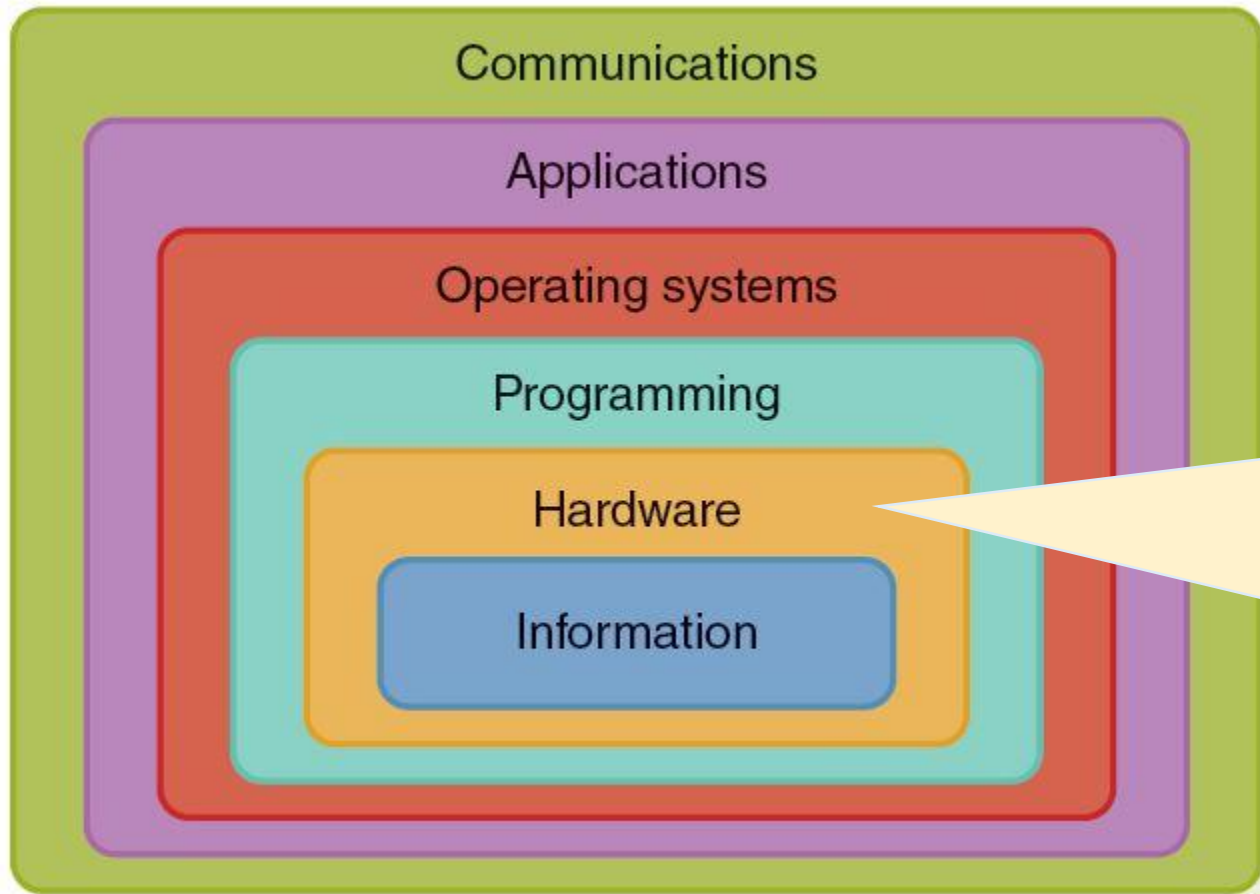FIGURE 1.2 A car engine and the abstraction that allows us to use it

© aospan/Shutterstock, Inc.; © Syda Productions/Shutterstock, Inc.

- Computing system are complex, and some some kind of abstract models are needed to understand and manage them.
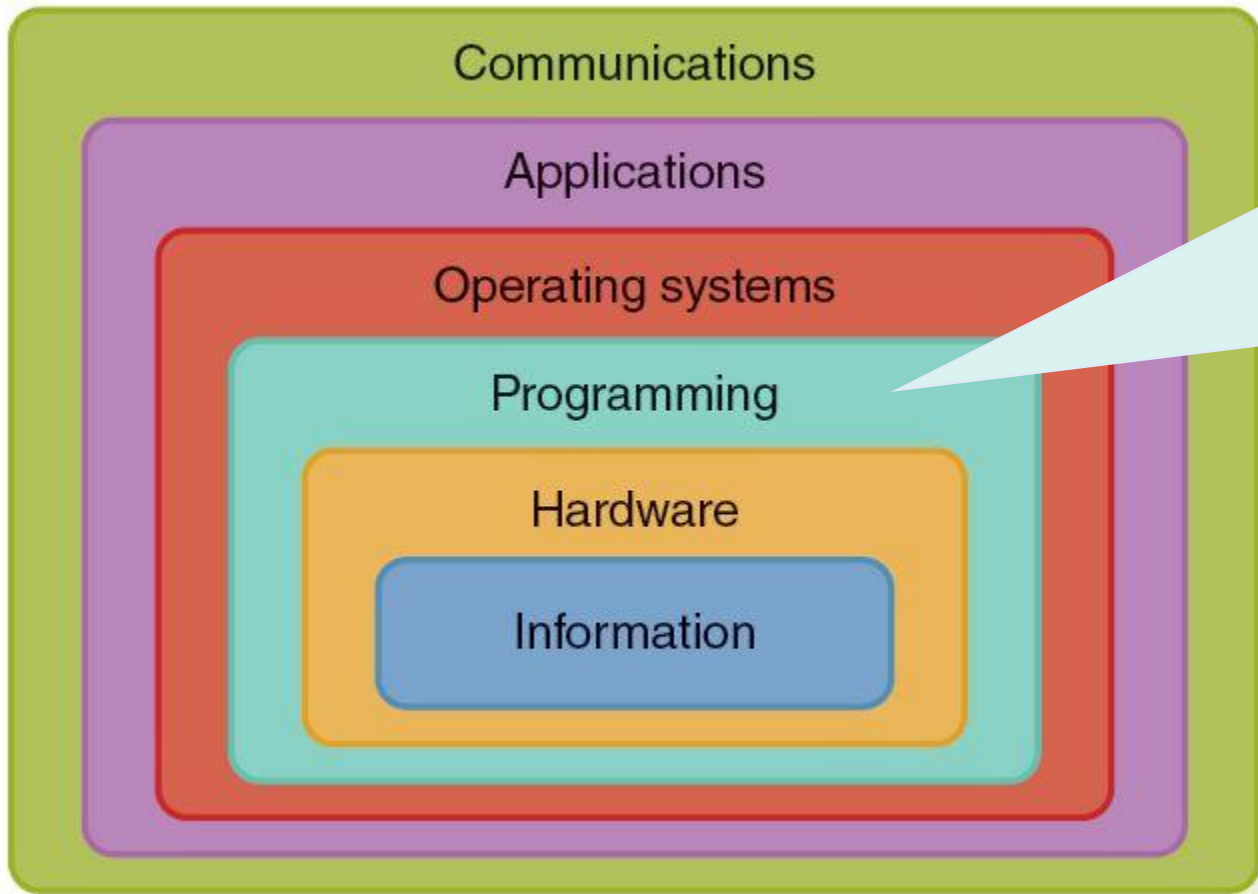- The six layers of computing system is an **abstract model**

Layers of a computing system

Information have to be **<u>represented</u>** and stored in a computer to be managed and elaborated

All the information, numbers, texts, photos, audio, video, Web pages, etc. are all represented using **binary digits 0 and 1**

Layers of a computing system

Communications

Applications

Operating systems

Programming

Hardware

Information

Computer hardware includes electronic circuits and gates, which control the flow of electricity

The main hardware components are the Central Processing Units (CPU), the memory (volatile and stable), the peripherals for input/output (keyboard, screen, network)
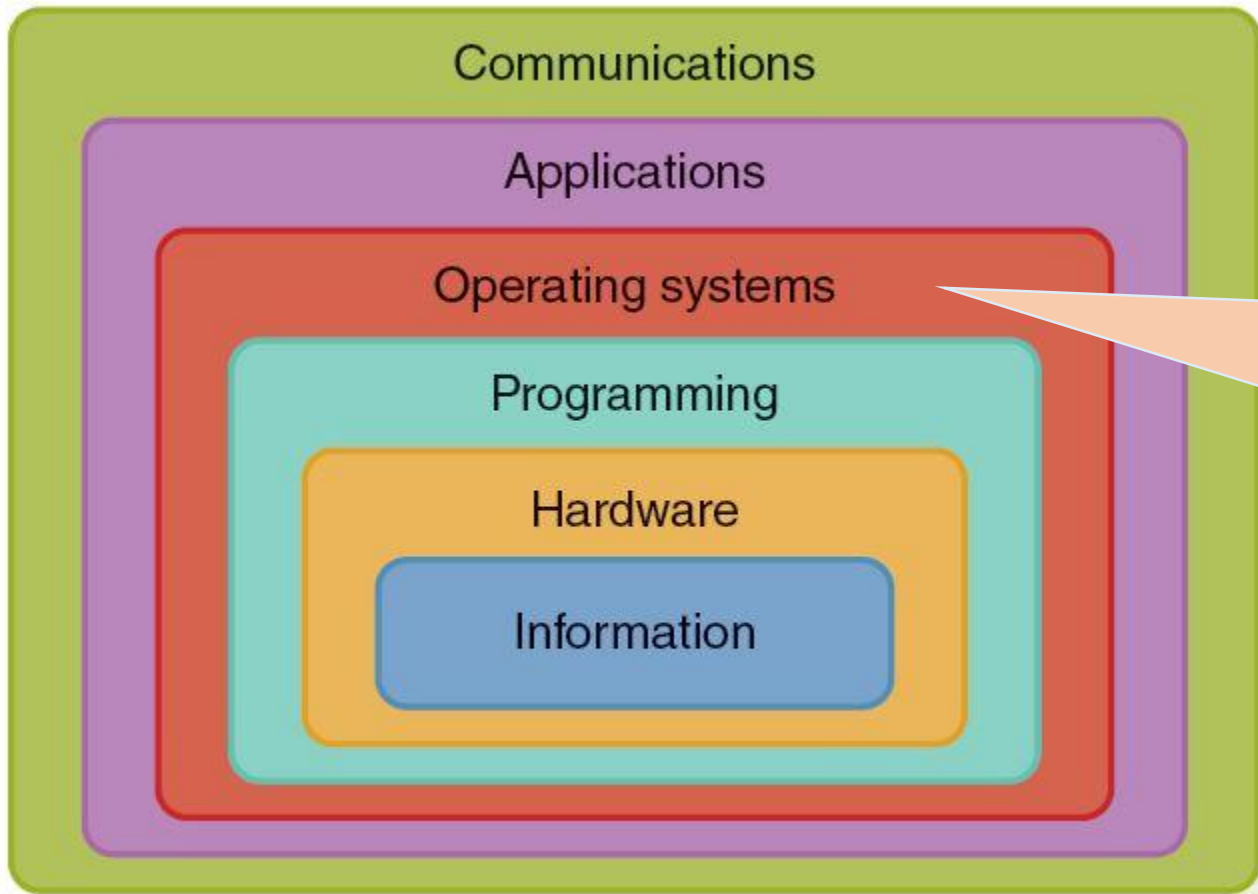
Layers of a computing system

This layer deal with software, the instructions that the system executes to accomplish computations and manage data.

Learning how to program or code is the main goal of this course.

Coding can be done at various levels, from a low level very close to the hardware (assembly), to higher levels, much easier for humans (Python, Java, C, etc.)
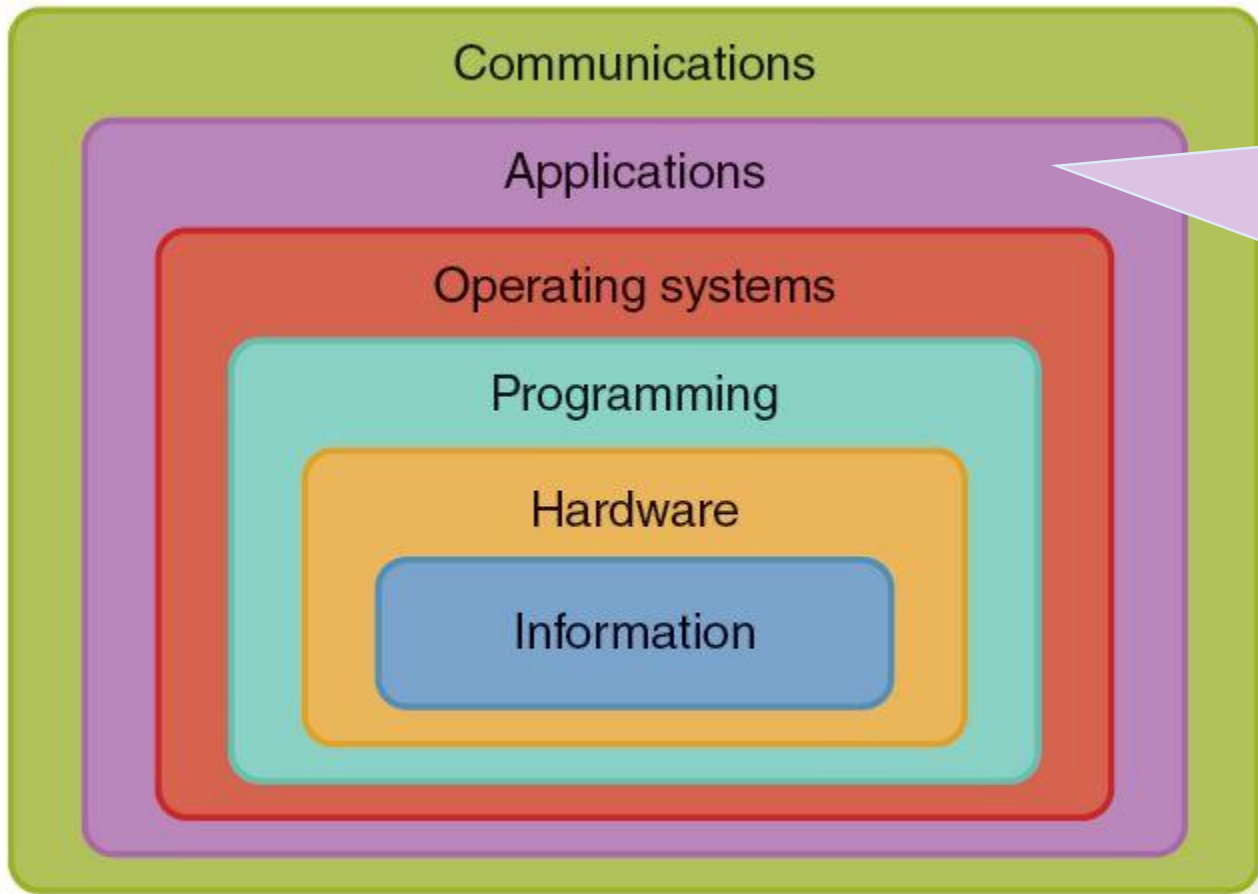
Layers of a computing system

Every computer has an operating system (OS) to help manage the computing system's resources

Microsoft Windows 10, Mac OS, Linus, Android, etc.

Layers of a computing system

Computing systems users run application programs to perform real-world problems, for example - write a text document, draw a picture or play a game.

Every program you run on your computer's operating system is an application program that help you to reach you goals or solve a certain problem.

Layers of a computing system

Communications

Applications

Operating systems

Programming

Hardware
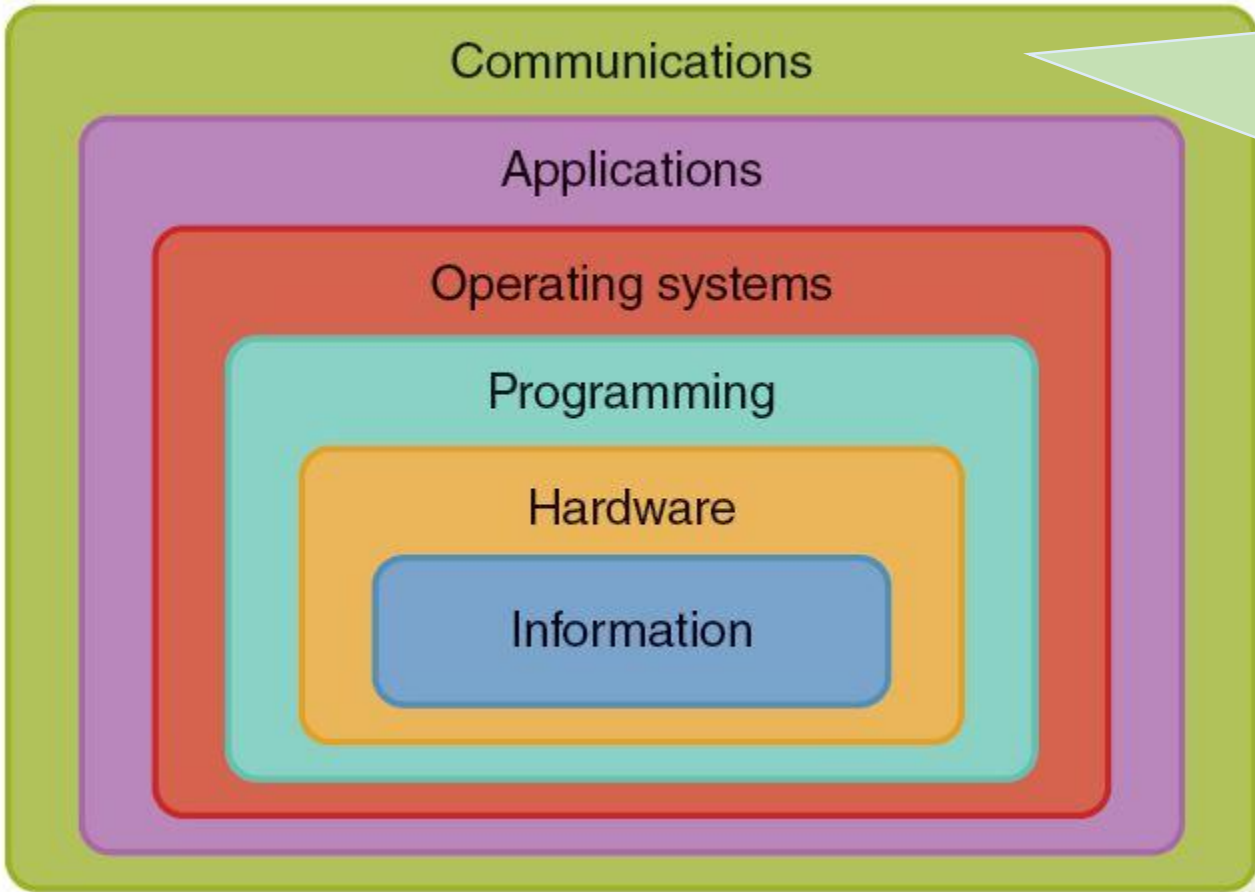
Information

Communication layer is a layer where two computer system can operate and exchange date.

Computers are connected into a networks to share information and resources.

We use internet to exchange data, access the Web, or simply chat.

Information layer

Data rapresentation

- Computers are **multimedia** devices, dealing with a vast array of information categories

- Computers store, present, and help us modify
  - Numbers
  - Text
  - Audio
  - Images and graphics
  - Video

  All information is stored as patterns of **binary digits** (**bits**): 0 or 1

- Why a binary alphabet?
  - Will be clear in a while

Analog vs. Digital Information

Computers are finite!

How do we represent an infinite world?

- We represent enough of the world to satisfy our computational needs and our senses of sight and sound

- Two possible way to represents data

    Analog: A **continuous** representation, analogous to the actual information it represents

    Digital data: A **discrete** representation, breaking the information up into separate elements

Analog vs. Digital Information

A mercury thermometer is an analog device

Actually, we have a continuous quantity (temperature) which is measured continuously along the time (another continuous value)
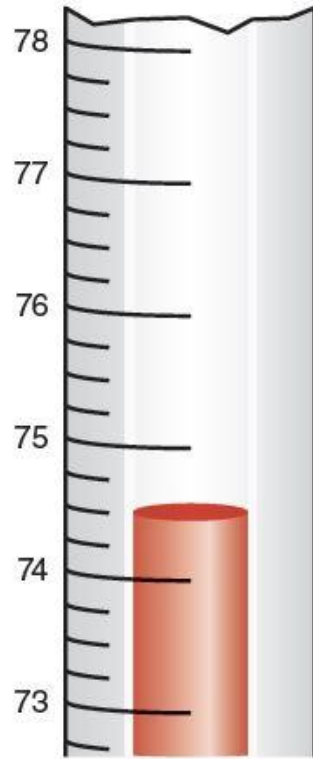
FIGURE 3.1 A mercury thermometer continually rises in direct proportion to the temperature

Computers cannot work well with analog continuous data, so we **digitize** the data

> **Discretization:** Read an analog signal at regular time intervals (frequency), sampling the value of the signal at the point.

> **Quantization:** Sampled values *(samples)* are rounded to a fixed set of numbers

Computers store (an transmit) units called binary digits or bits

Low Voltage = 0
High Voltage = 1

A digital signal thus becomes

- a sequence of **discrete numbers**, in turn represented as ***binary patterns of fixed length***

Electronic signals used to store (discrete) values

Important facts about electronic signals

- An analog signal continually fluctuates in voltage up and down

- A digital signal, if we use **binary numbers**, has only a **high state** or a **low state**, corresponding to the two binary digits 0/1
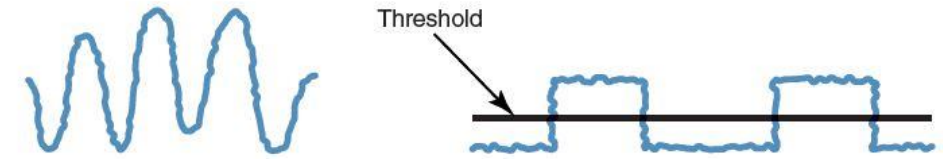
FIGURE 3.2 An analog signal and a digital signal

Threshold

FIGURE 3.3 Degradation of analog and digital signals

All the electronic signals (both analog and digital) degrade as they move down a wire. The voltage of the signal fluctuates due to environmental effects.
**Digital signals can be easily reshaped, and are less error-prone than analog signals.**

Binary numbers and computers

- Computers have storage units called binary digits or bits

Low Voltage = 0
High Voltage = 1

- Computer memory stores digital information as multiple of bytes, where a byte is composed of **8 bits = 2^3 bits**

- A volatile memory to store data with *byte* and *word addresses*:

bit 31.........................bit 0			
Address 0	byte 3	byte 2	byte 1	byte 0
Address 4	byte 7	byte 6	byte 5	byte 4
Address 8	byte 11	byte 10	byte 9	byte 8
Address 12	byte 15	byte 14	byte 13	byte 12
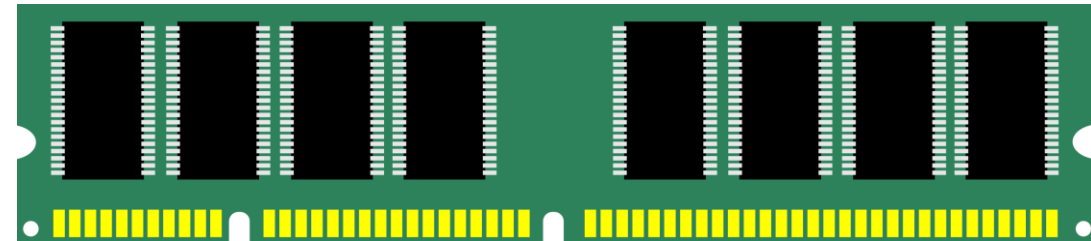
Binary representation

- *How many things can* n bits *represent?*

- With 1 bits: 2^1 things

- By doubling the bits, that is 2 bits: 2^2 things

- In general, with n bits we thus represent: 2^n things

| 1 Bit	2 Bits	3 Bits	4 Bits	5 Bits
0	00	000	0000	00000
1	01	001	0001	00001
	10	010	0010	00010
	11	011	0011	00011
		100	0100	00100
		101	0101	00101
		110	0110	00110
		111	0111	00111
			1000	01000
			1001	01001
			1010	01010
			1011	01011
			1100	01100
			1101	01101
			1110	01110
			1111	01111
				10000
				10001
				10010
				10011
				10100
				10101
				10110
				10111
				11000
				11001
				11010
				11011
				11100
				11101
				11110
				11111

FIGURE 3.4 Bit combinations

Example to represent 7 things (days of a week)

- To represent 7 distinct items we need *at least* 3 bits
  - 3 bits, by which we can represent $2^3 = 8$ distinct objects, is the minimum number of bits we can use
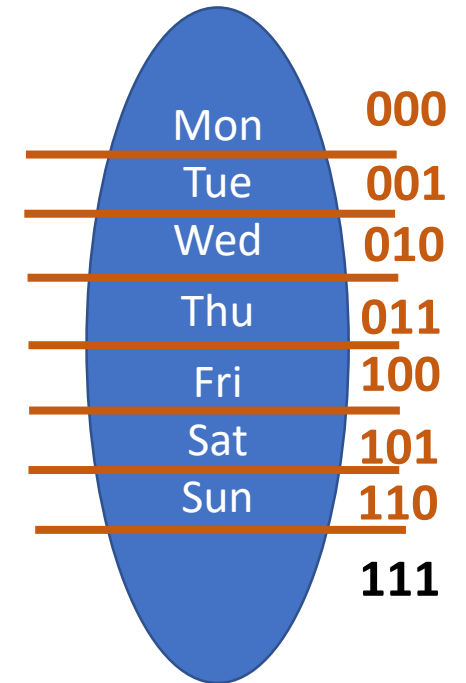
000 Mon
001 Tue
010 Wed
011 Thu
100 Fri
101 Sat
110 Sun
111 *Non used*
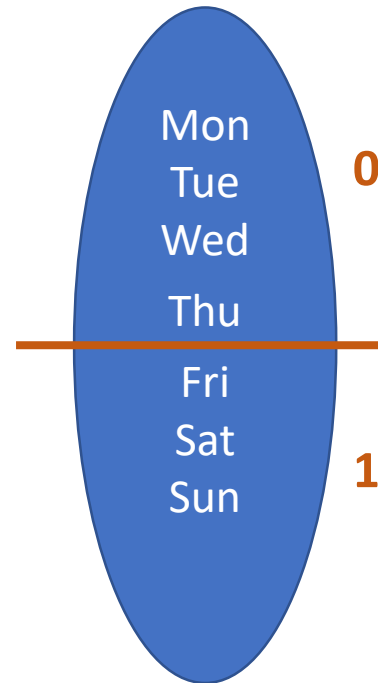
**1-BIT code
Only 2 groups**
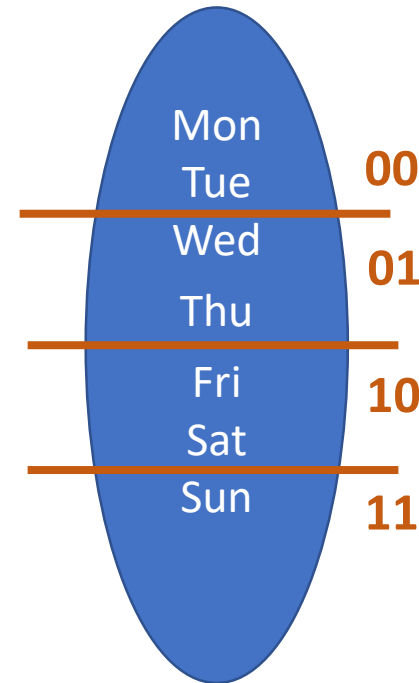
**2-BIT code
Only 4 groups**

**3-BIT code
8 groups**

Representing text

Fortunately the number of characters to represent is finite (whew!), so list them all and assign each a binary string

Character set

A list of characters and the codes used to represent each one.
Computer manufacturers agreed to standardize

ASCII character set

American Standard Code for Information Interchange
7 bits version allows 128 unique characters

ASCII Character Set Mapping

Left Digit(s)	Right Digit	0	1	2	3	4	5	6	7	8	9
						ASCII					
0		NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1		LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2		DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3		RS	US	□	!	"	#	$	%	&	'
4		(	)	*	+	,	–	.	/	0	1
5		2	3	4	5	6	7	8	9	:	;
6		<	=	>	?	@	A	B	C	D	E
7		F	G	H	I	J	K	L	M	N	O
8		P	Q	R	S	T	U	V	W	X	Y
9		Z	[	\	]	^	_	`	a	b	c
10		d	e	f	g	h	i	j	k	l	m
11		n	o	p	q	r	s	t	u	v	w
12		x	y	z	{	|	}	~	DEL		

FIGURE 3.5 The ASCII character set

Digital images

The images is **sampled**:

- subdivided in points called pixel (pixel element)

Each pixel is **quantized** by encoding as a discrete number corresponding to:

- a specific color
- a particolar grey tone for b/w images

Image represented as a grid/matrix of numbers

Usually numbers associated with pixels have a fixed length (a multiple of a byte)

For color images, a common method is to represent each pixel with 3 numbers, each corresponding to a fundamental color (RGB – Red,Green,Blue)

Along with the pixel values we have to store the size of the image (no. of rows and columns) and the resolution (dpi, "dot per inch")

Images are finally stored in files as sequences of numbers, one row of pixels after the other.
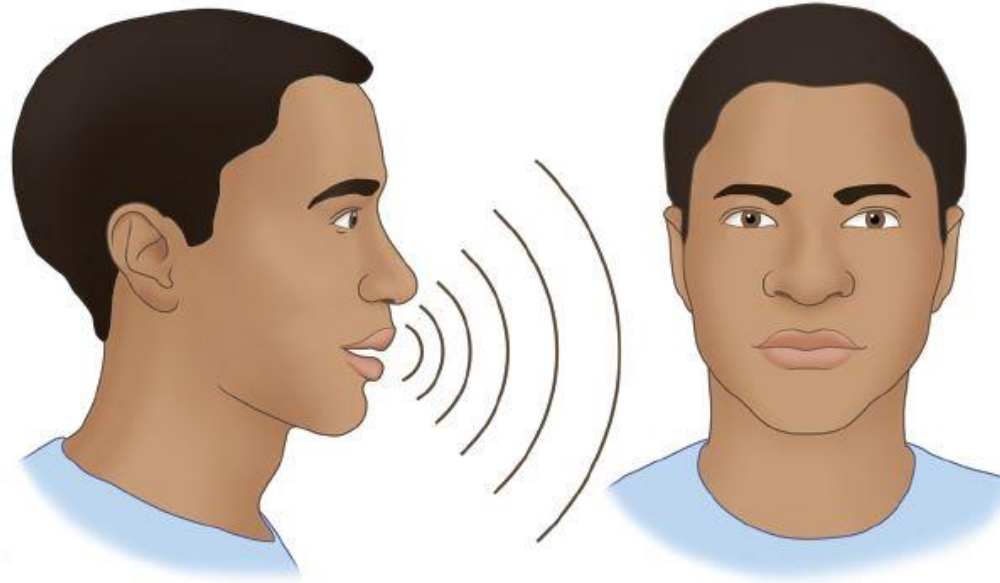
Representing Audio Information

FIGURE 3.7 A sound wave vibrates our eardrums

We perceive sound when a series of air pressure waves vibrate a membrane in our ear, which sends signals to our brain

Representing Audio Information

- Digital audio systems includes microphones converting **sound** to an **analog electrical signal**
  - Then an analog-to-digital converter converts the analog signal into a digital signal.

- Digitize the analog electric signal by
  - **Sampling**: periodically measure the voltage
  - **Quantization**: represent the voltage as a number using a finite number of bits

How often should we sample?

A sampling rate of about 40,000 times per second is enough to create a reasonable sound reproduction.
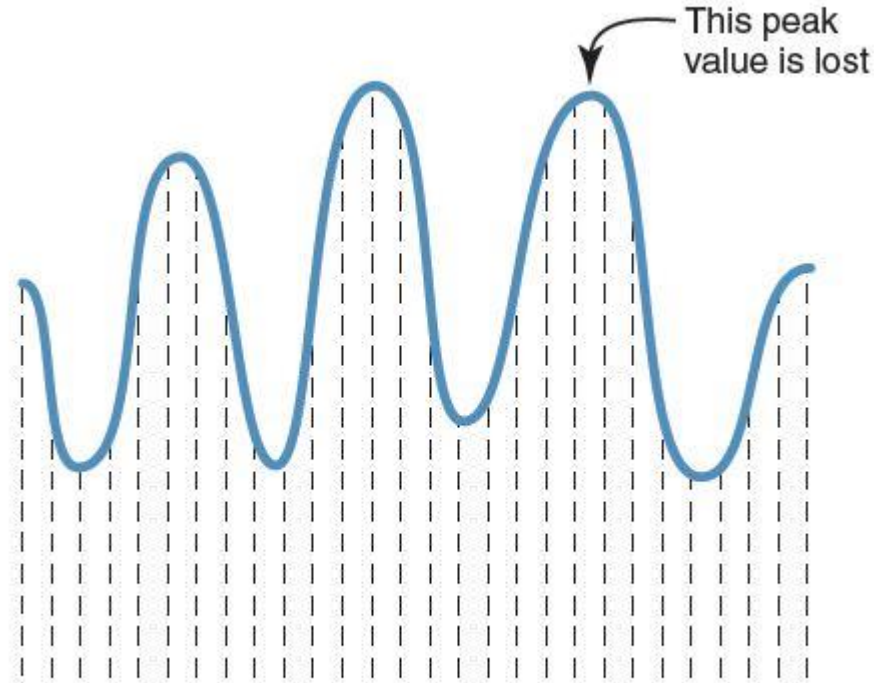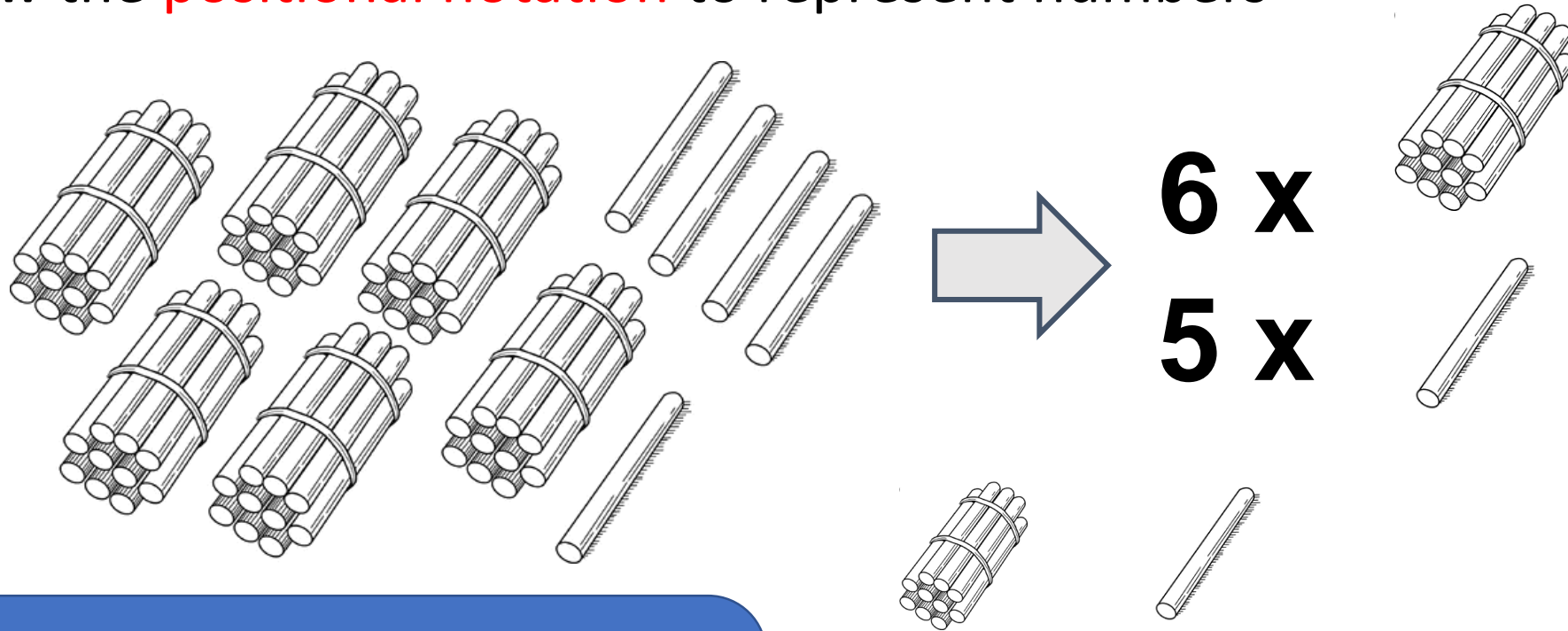
Representing Audio Information

This peak value is lost

Some data is lost, but a reasonable sound is reproduced

FIGURE 3.8 Sampling an audio signal

Integer numbers

Review the positional notation to represent numbers

6 x

5 x

10 is the base of this representation

We need 10 simbols, whose value depends on the position

| 6 | 5 |

Positional notation

- Value of a number represented in a **generic base B** with **symbols used for digits representing quantities: 0,1, …., B-1**

$$d_n * B^{n-1} + d_{n-1} * B^{n-2} + ... + d_2 * B^1 + d_1 * B^0$$

B is the base of the number

n is the number of digits in the number

d_i is the digit in the ith position in the number

642 (**$d_3 d_2 d_1$ expressed in base B=10**) is equal to:

6 * 10^2 + 4 * 10^1 + 2 * 10^0 = 600 + 40 + 2

Binary integer numbers

- Decimal numbers have base 10 and need 10 digit symbols:
  0,1,2,3,4,5,6,7,8,9

- Exadecimal numbers have base 16 and need 16 digit symbols:
  0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Binary numbers have base 2 and only need 2 digit symbols:
  0,1

For a number to exist in a given base B, it can only contain the digits in that base, which range from 0 up to B-1

Converting from binary to decimal

- *What is the decimal equivalent N_{10} of the binary number $N_2 = 1101110$?*

$$1 \times 2^6 = 1 \times 64 = 64$$
$$+ 1 \times 2^5 = 1 \times 32 = 32$$
$$+ 0 \times 2^4 = 0 \times 16 = 0$$
$$+ 1 \times 2^3 = 1 \times 8 = 8$$
$$+ 1 \times 2^2 = 1 \times 4 = 4$$
$$+ 1 \times 2^1 = 1 \times 2 = 2$$
$$+ 0 \times 2^0 = 0 \times 1 = 0$$

Sum = **110** in base 10

Converting from decimal to binary

Given a natural number N_{10}, where its representation in base 2 is:

$$N_2 = d_{n-1} \dots d_1 \, d_0$$

we have:

$$N_{10} = d_{n-1} * 2^{n-1} + \dots + d_1 * 2^1 + d_0 * 2^0$$

We determine the various $d_i \in \{0,1\}$ with the division-by-2 method :

- N is divided by two.
- The remainder is the least-significant bit d_0.
- The quotient is again divided by 2
- The remainder is the next least significant bit d_0.
- And so on, until a quotient of zero is reached.

- NOTE: each remainder must be either 0 or 1 when dividing by 2

Converting from decimal to binary

Binary equivalent N_2 *of the binary number* $N_{10} = 110$:

N_{10}	Reminder
110	$0 = d_0$
55	$1 = d_1$
27	$1 = d_2$
13	$1 = d_3$
6	$0 = d_4$
3	$1 = d_5$
1	$1 = d_6$

quotient

quotient

$$N_2 = 1101110$$

Converting from decimal to binary (alternative way)

How to store the number 57 with 8 bits?

We need to find the *0/1* coefficients of $2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7$ so that their sum is 57.

Subtractions method:

Input	Powers of 2	Is larger than or equal to 2^i?	Coefficient
57	$2^7 = 128$	$57 - 128 < 0$	0
57	$2^6 = 64$	$57 - 64 < 0$	0
57	$2^5 = 32$	$57 - 32 = 25 \geq 0$	1
25	$2^4 = 16$	$25 - 16 = 9 \geq 0$	1
9	$2^3 = 8$	$9 - 8 = 1 \geq 0$	1
1	$2^2 = 4$	$1 - 4 < 0$	0
1	$2^1 = 2$	$1 - 2 < 0$	0
1	$2^0 = 1$	$1 - 1 = 0 \geq 0$	1

The binary representation of 57 is **00111001**.

Binary Arithmetic

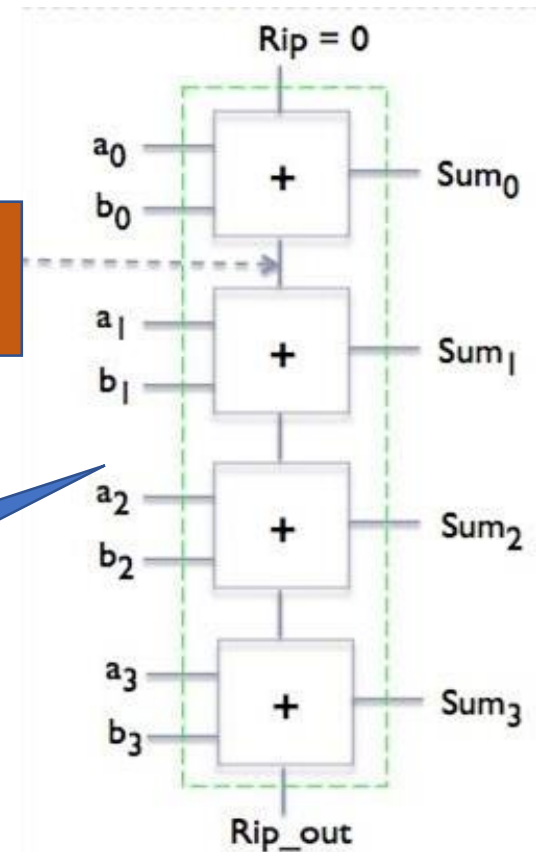- Remember that there are only 2 digit symbols in binary, 0 and 1. Therefore 1 + 1 = 0 with a *carry*
- *This is analogous to decimal arithmetic: 5 + 5 = 0 with a carry*

Example of 7 (111) + 6 (110) = 13 (1101):
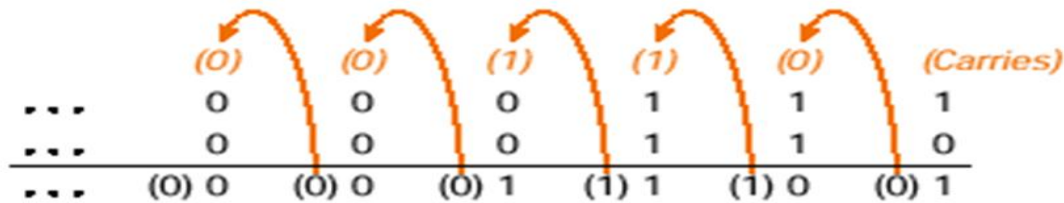
Carry propagation

Arithmetic Logic Unit (ALU) made of Single-bit Adders

Real numbers to binary

- Using a Byte (8 b), we can use a simple approach (**Fixed-point representation**):
  - **4 bits** for the **integer part**, and **4 bits** to represent the **fractional part**

- **Warning !!** The smallest number we can represent is $2^{-4}=1/16=0.0625$

- **Warning !! Even worse!!** We can represent only 2^4 numbers between 0 and 1.

- Not only are we limited to a **smallest** and **largest** representable number, but since real numbers are infinite while bit configurations are finite, we can only represent a limited set of number in between the **smallest** and **largest** values.

Real numbers to binary

- Since any binary number can be represented as *±1.????...?? × 2^N*, computers most commonly use:

- **Single precision** (32 bits) **Floating-point representation**:
  - 1 bit for the sign (negative vs. positive)
  - 8 bits for the exponent *N* of 2^N
  - 23 bits for the significand *????..??*

- **Double precision** uses 64 bits, and it is therefore more accurate (but more expensive)

Different Binary Units

Note that
$1024 = 2^{10} \approx 10^3 = 1000$

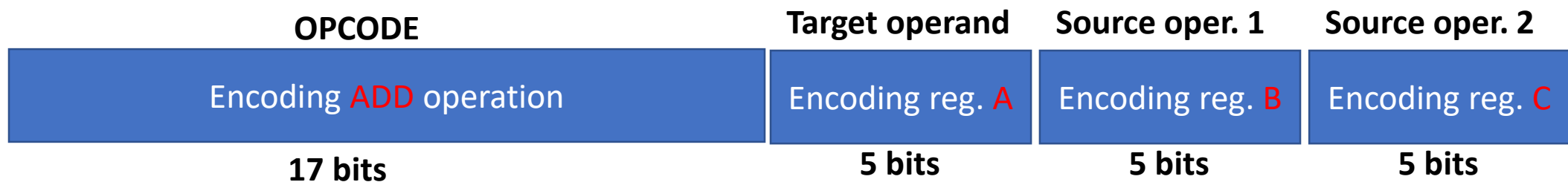| UNIT	VALUE	STORAGE
Bit	1 or 0	
Byte (B)	8 Bits	Character
Kilobyte (KB)	1024 Bytes = 2^{10} Bytes	Half page of text
Megabyte (MB)	1024 Kilobytes = 2^{20} Bytes	About 2 mins MP3 file
Gigabyte (GB)	1024 Megabytes = 2^{30} Bytes	About one hour Movie
Terabyte (TB)	1024 Gigabytes = 2^{40} Bytes	128 DVD Movies
Petabyte (PB)	1024 Terabyte = 2^{50} Bytes	7 billion Facebook photos
Exabyte (EB)	1024 Petabyte = 2^{60} Bytes	50,000 years of DVD
Zettabyte (ZB)	1024 Exabyte = 2^{70} Bytes	Global internet traffic per year (2016)

Hardware layer

Von Neumann architecture of a computing system

- Architecture composed of
  - CPU: Central processing unit
    - ALU to perform arithmetic/logic operations
  - Memory to store
    - Programs (stored-program)
    - Data accessed by program
  - Input/Output (I/O) devices
    - including disks and SSD
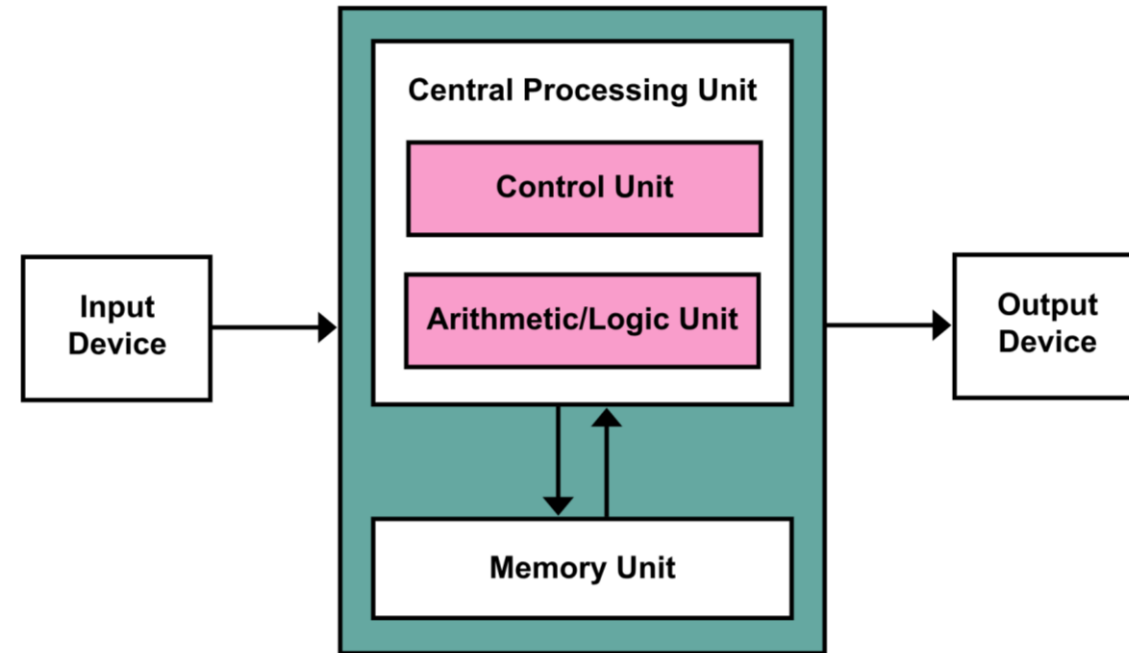
Machine languages

- CPU are designed to recognize **machine instructions** stored in memory as sequences of bits

- The (short) list of instructions with their binary encoding is called **machine languages**

- Example of machine language
  - ADD A,B,C   where A,B, and C are operands, and the semantics is: $A \leftarrow B + C$
  - Encoding on 4 Bytes, where A, B and C are fast memory registers in the CPU

OPCODE	Target operand	Source oper. 1	Source oper. 2
Encoding ADD operation	Encoding reg. A	Encoding reg. B	Encoding reg. C
17 bits	**5 bits**	**5 bits**	**5 bits**

Program execution

- The order in which instructions are fetched/executed (**control flow**) corresponds to the <u>sequential order</u> in which instruction are stored in memory

- So the flow of control of the CPU is guided by an *address in memory*
  - The Program Counter (**PC**) maintain for a running program the address of the next machine instruction to execute

Program execution

- The CPU continuously **repeat** the following 3 steps:
  1. Fetch from memory the next instructions pointed by the PC
  2. Decode the instruction
  3. Execute the instruction, and update the PC to the next instruction to fetch

- The sequential **control flow can be modified** by the program, executing special instruction of JUMP/BRANCH.

Programming layer

What is a program?

- A **program** is a **sequence of instructions** that specifies how to perform a computation.
  - Examples: finding the roots of a polynomial (math), searching and replacing text in a document (symbolic), or processing an image (graphical)

Programming as the process of breaking a large, complex task into smaller and smaller subtasks until the subtasks are simple enough to be performed by combining basic instructions.

- The program is **run/executed** by the computer to realize a given application, following the **control flow**
  - Luckily, we can avoid to write programs in **machine language !!**

What is a program?

- We can program in different languages, and generally we have ways to do:
  - **input:** Get data from the keyboard, a file, the network, or some other device.
  - **output:** Display data on the screen, save it in a file, send it over the network, etc.
  - **math:** Perform basic mathematical operations like addition and multiplication.
  - **conditional execution:** Check for certain conditions and run the appropriate code.
  - **repetition:** Perform some action repeatedly, usually with some variation.

Languages for programming?

- **Natural languages**: languages that people speak, such as English, Italian, or French, and they evolved naturally.

- **Formal languages:** particular languages designed for specific applications. Example: *math notation*.
  - **Strict syntax** rules concerning **tokens** and **structure**
  - No **ambiguity**

- **Programming languages** are **formal languages** that have been designed to express computations.

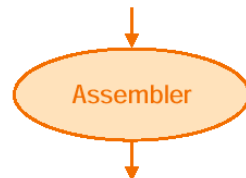# Abstracting the machine to ease the programming job

High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

C compiler
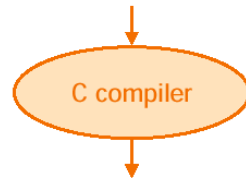
Assembly
language
program
(for MIPS)

```
swap:
        muli $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

- On the left, an example of language hierarchies

- High-level language
  - Productivity
  - Code portability because of translators, such as compiler or assembler
- Assembly language
  - Mnemonic representation of machine instructions
- Machine language
  - Binary digits (bits)
  - Encoding of machine instructions and data

Learning to code with Python

From Wikipedia, the free encyclopedia

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.   Guido was a fan of the TV series *Monty Python's Flying Circus :-)*

An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in other high-level languages such as C++ or Java.

An interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, <u>without previously compiling them into a machine language program</u>
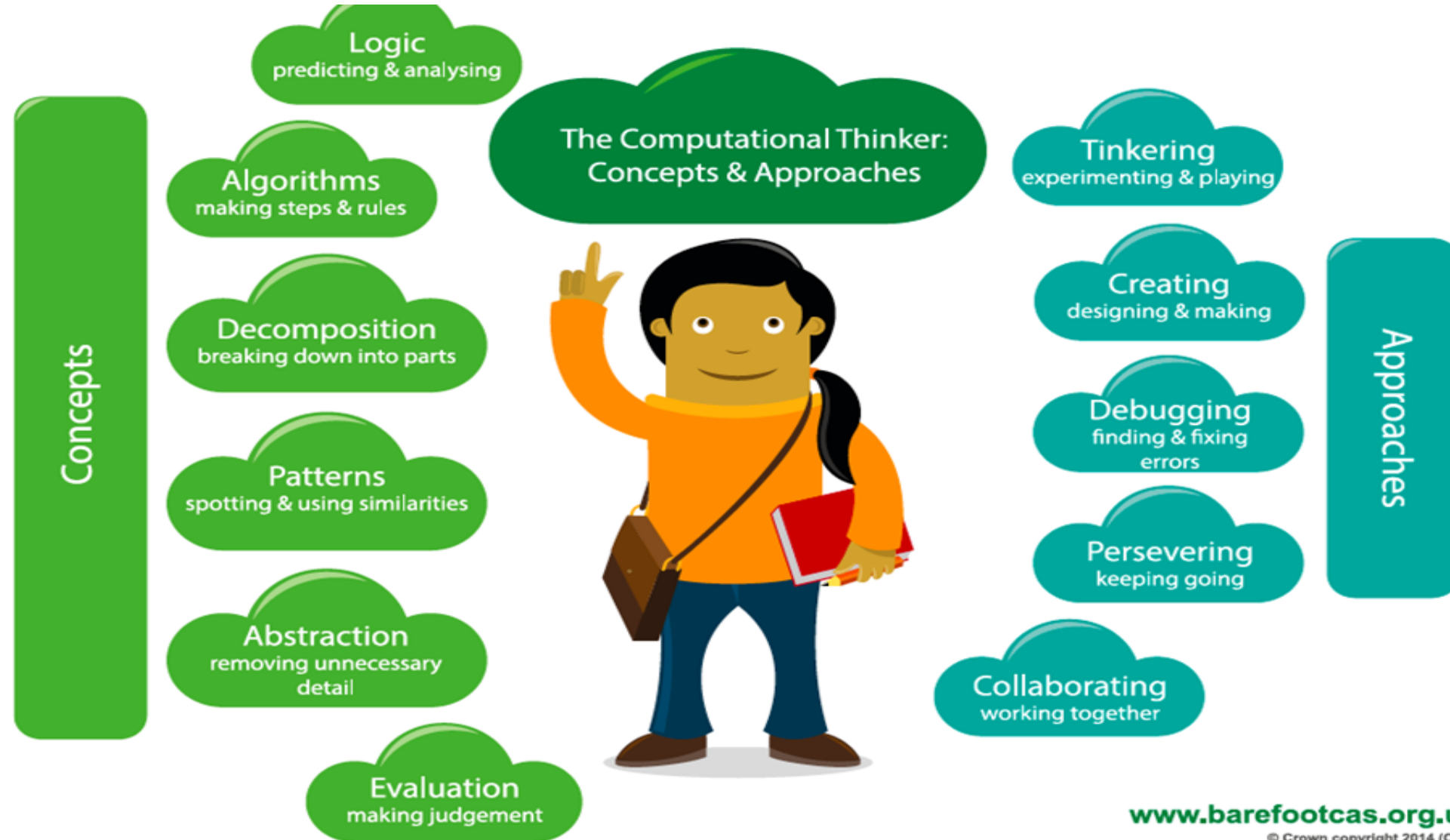
Computational thinking

What is Computational Thinking

Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can effectively be carried out by an information-processing agent.

Cuny, Snyder, Wing

Important elements of computational thinking

The Computational Thinker: Concepts & Approaches

Concepts
- Logic — predicting & analysing
- Algorithms — making steps & rules
- Decomposition — breaking down into parts
- Patterns — spotting & using similarities
- Abstraction — removing unnecessary detail
- Evaluation — making judgement

Approaches
- Tinkering — experimenting & playing
- Creating — designing & making
- Debugging — finding & fixing errors
- Persevering — keeping going
- Collaborating — working together

www.barefootcas.org.uk
© Crown copyright 2014 (OGL)

Problem solving and programming

- The single most important skill for a computer scientist is thus **problem solving**
  - the ability to formulate problems,
  - think creatively about solutions, and
  - express a solution clearly and accurately.

  with solutions that can be carried out by an information-processing agent

- Learning how to **code/program** is an excellent opportunity to practice problem-solving skills

Programming for problem solving is funny and creative