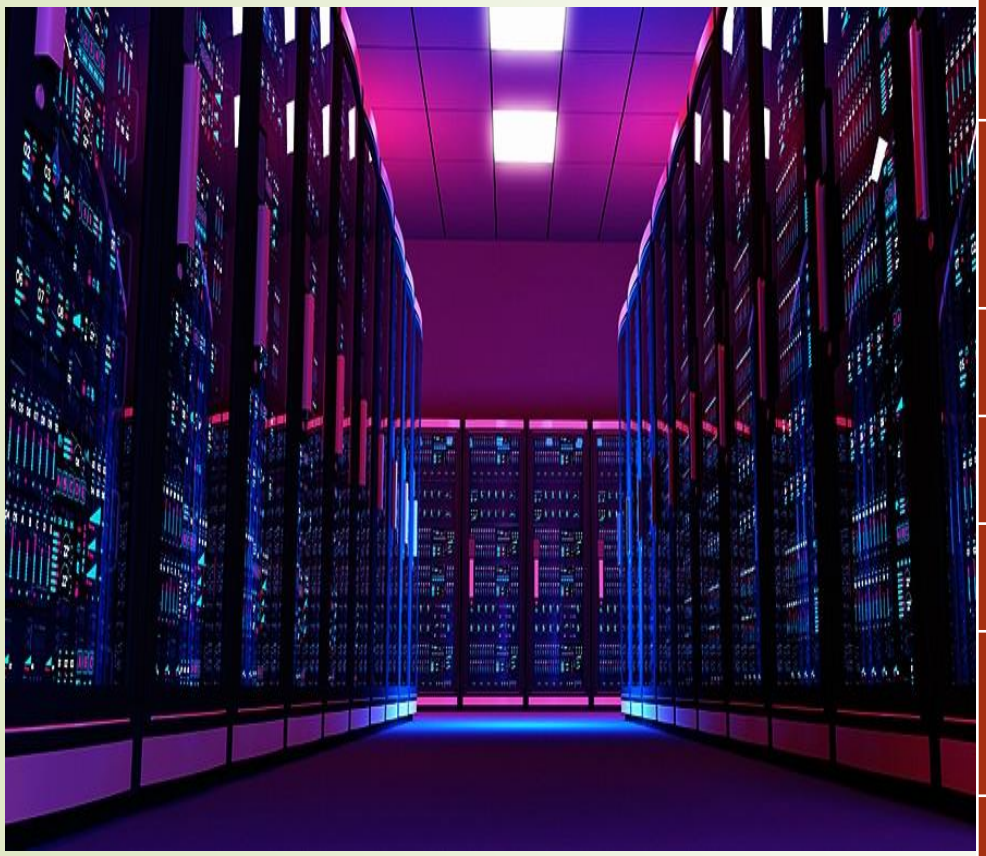
Introduction to High Performance and Data Intensive Computing

Dott. Costantino Zazza, costantino.zazza@unitus.it

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

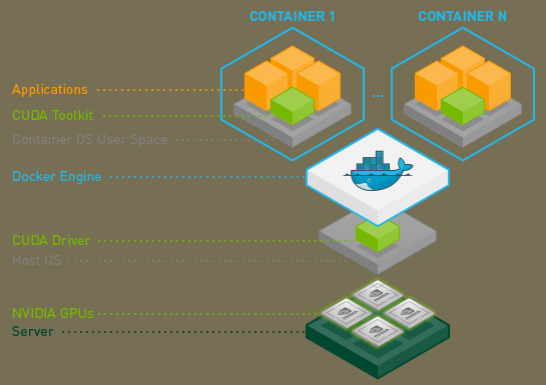
Course overview



	Module
1.	**Computing architectures.** Computer architectures by hardware design. Network topologies. Flynn taxonomy. Memory models and program driven approach. Scalability. Amdhal and Gustafsson laws.
2.	**Software and OS design.** Software design of modern computing systems. Architectures of Operating System. Unix/Linux OS. Software stack and services. Network services. Introduction to storage I/O.
3.	**Lab: Linux hands-on.** Introductory mini-course on linux/unix with hands-on on DIBAF cluster.
4.	**Recent advances.** Cloud driven computing. Docker containers and kubernetics. Quantum Computing.
5.	**Lab: Kubernetes hands-on.** Introductory mini-course on Kubernetes with hands-on on DIBAF Cluster.
6	**Programming.** Introduction to programming technics and methodologies in C and FORTRAN. Compilers. HPC libraries and tools. Programming examples in linear algebra.
7	**Lab: programming hands-on.** Learning by examples in C and FORTRAN programming under Linux OS and tools.

3

Course overview



Lessons' Timetable

Mon 05/05/2025	10:30-12:30 ; 14:30-16:30	Computing Architectures
Wed 07/05/2025	10:30-12:30 ; 14:30-16:30	Software and OS design
Mon 12/05/2025	10:30-12:30 ; 14:30-16:30	Lab: Linux hands on
Wed 14/05/2025	10:30-12:30 ; 14:30-16:30	Recent advances
Mon 19/05/2025	10:30-12:30 ; 14:30-16:30	Lab: docker hands on
Wed 21/05/2025	10:30-12:30 ; 14:30-16:30	Lab: kubernetes hands on
Mon 26/05/2025	10:30-12:30 ; 14:30-16:30	Programming technique
Wed 28/05/2025	10:30-12:30 ; 14:30-16:30	Lab: C and Fortran programs

Wed 04/06/2025 Workshop Area di Ricerca Roma 1, Via Salaria Km 29.300 00015 Monterotondo (RM) – Italy

Follow-up & Examination

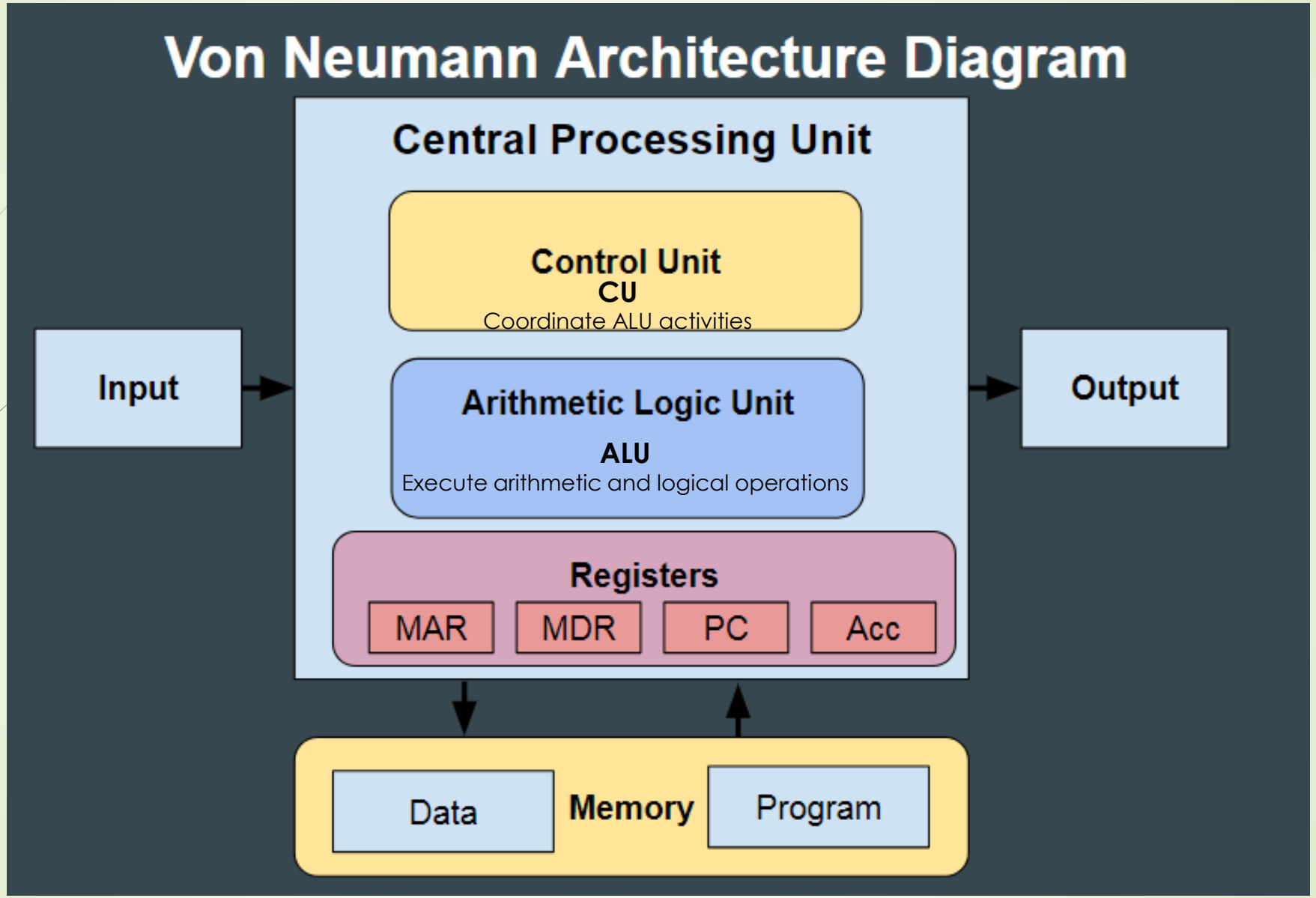
Introduction to Computer Architectures

Sequential Computers

The traditional architecture of a computer, first designed by John von Neumann(*) in early 40's, is the one where the Central Processing Unit (CPU) fetch the instructions from volatile memory where a program reside, then decode and excecute them over some data set present on the same memory type.
Such an architecture is capable of executing ONE instruction (over one data type) at a time and thus computers designed with this architecture are called Sequential (or Serial) computers.

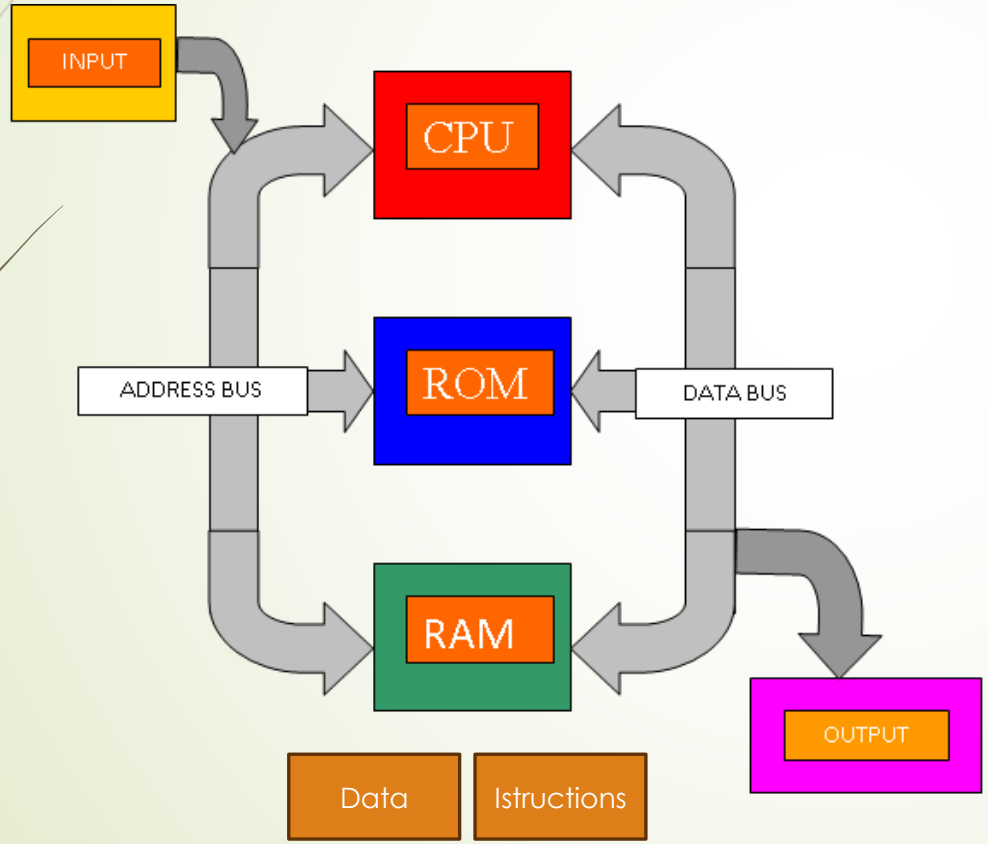
(*) John von Neumann (Budapest 28/12/1903 - Washington 8/2/57) was one of the most brilliant matematicians of last century. Far beyond his contribution to computing architecture, his work has given major contributions to a number of fields from mathematics to physics, economy, computers and statistics. https://en.wikipedia.org/wiki/John_von_Neumann

RAM:
Randon Access Memory



The Harvard Architecture

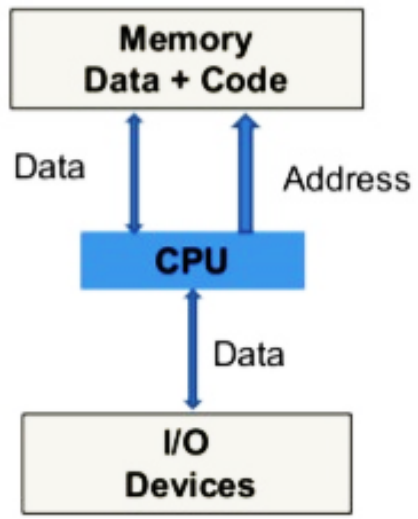
Two separated buses for instructions and data



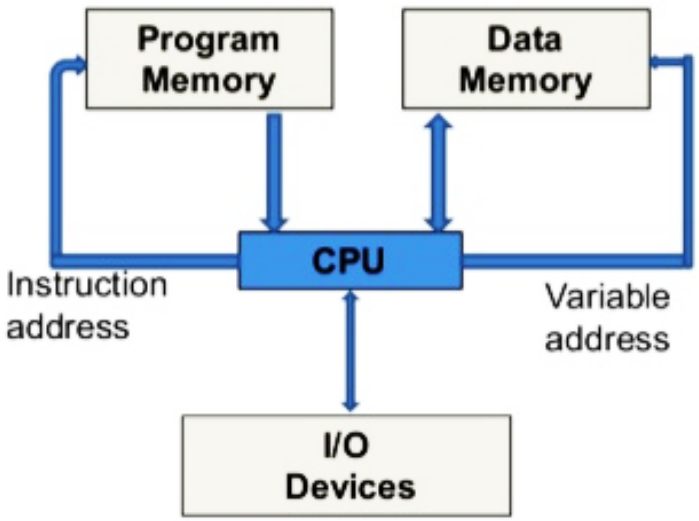
The Harvard architecture is implemented on all modern RISC/CISC CPUs.

Multi-core PC
Clusters
GPU (Many SP)

Von Neumann vs. Harvard Architecture



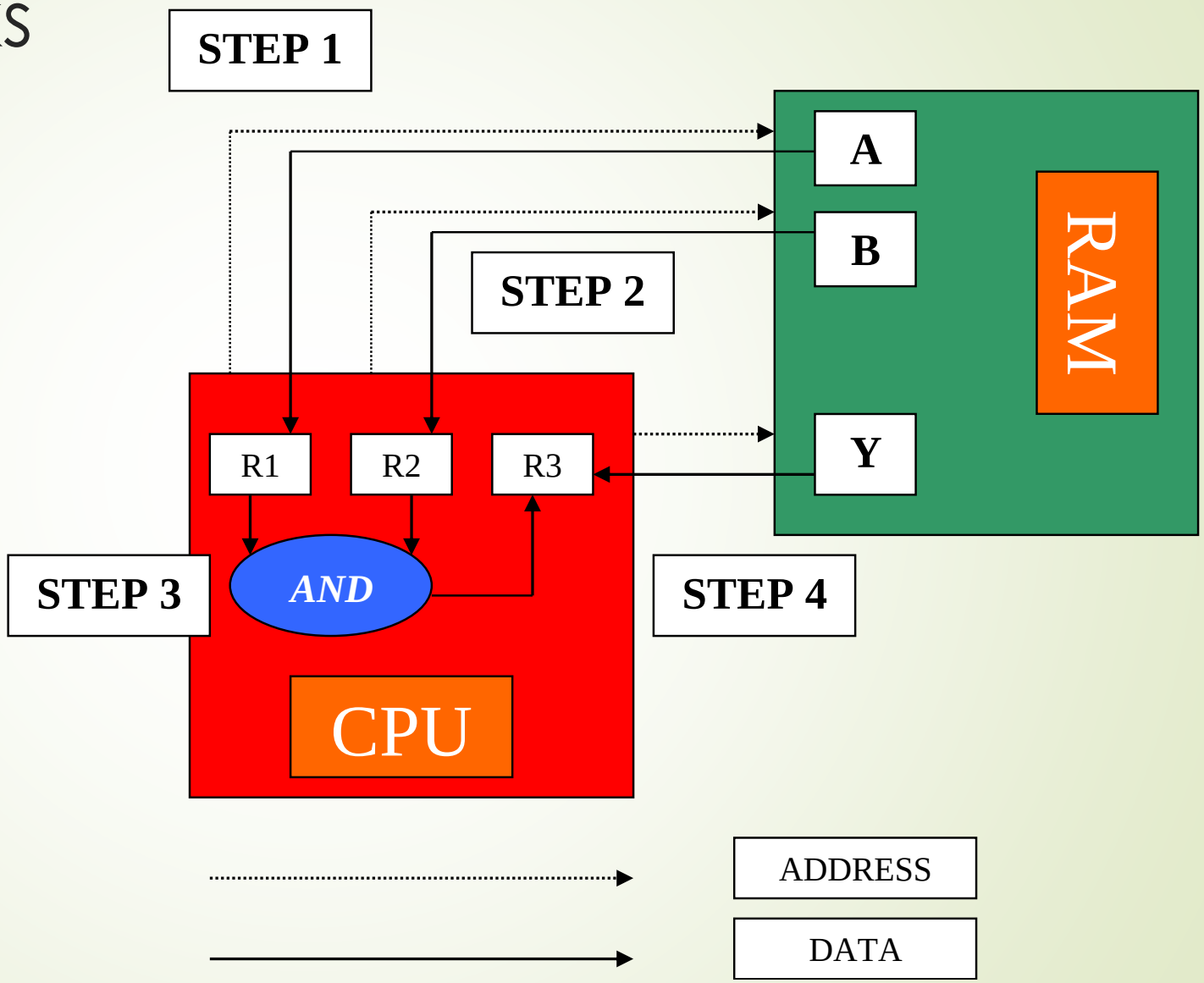
Von Neumann Machine



Harvard Machine

How it works

$$Y = A \cdot B = A \text{ AND } B$$



But, wait please …

Memory, RAM, ROM
CPU, registers, ALU

…

…

…

A AND B, what the hell those formulas mean?

No problem at all !!!

Let's review together some basics of computer science.

Boolean and binary representation

Digital computers use BINARY numbers

The lowest element of binary algebra (that is a number in base 2) is the BIT (from Binary digiT).

The bit may assume only two values: 0 or 1.

In boolean (or binary) algebra any positive integer number is made out of a linear combination of power of 2:

$$N = \sum_i k \cdot 2^j \qquad \forall i, j, k \in I \qquad j \mid \max(2^j, N) = 2^J$$

In example:

$$35 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + 1 \cdot 2^5$$
$$35 = 1 + 2 + 0 + 0 + 0 + 32$$

In the electronic circuits of a compunting machine the status 0 of a bit correspond to a LOW voltage (0V, GND) while the status 1 has a HIGH voltage (+5V).
We are typically used to think at numbers in base 10:

$$35 = 5 \cdot 10^0 + 3 \cdot 10^1$$
$$35 = \quad 5 \quad + \quad 30$$

that is

$$N = \sum_i k \cdot 10^j \qquad \forall i, j, k \in I$$
$$j \mid \max(10^j, N) = 10^J$$

So, we need a minimum of abstraction to understand the operations in binary even if in every day life a pletora of activities are directly representable via a binary algebra:

The light switch at home
The booking service of travel seats
…
…

So, given the powers of 2

K	7	6	5	4	3	2	1	0
2^k	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$2_{10}{}^k$	128	64	32	16	8	4	2	1

we may express any integer N in base 2 switching ON (1) or OFF(0) the bit of its figures.

In example, if we got N=182:

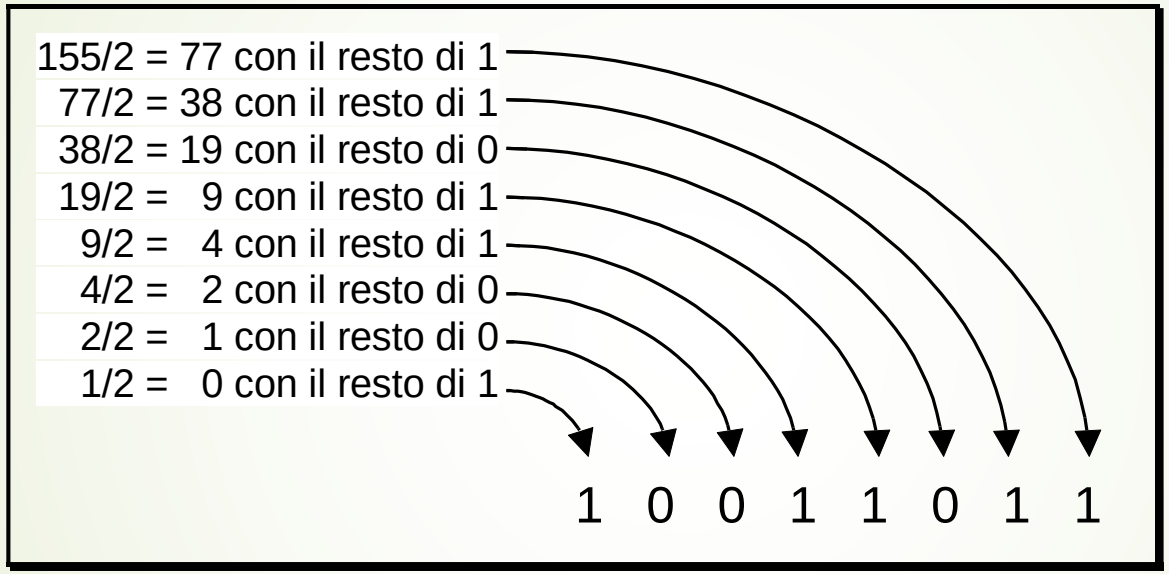
2^k	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
$2_{10}{}^k$	128	64	32	16	8	4	2	1
Binario	1	0	1	1	0	1	1	0
Decimale	128	-	32	16	-	4	2	-

and then, we may write

$$182_{10} = 10110110_2$$

Decimal to binary conversion

In example, 155_{10}



So that,

$$155_{10} = 10011011_2$$

Binary-to-Decimal Conversion

Example 1: 11011010

Example 2: 10011101

	Column 8	Column 7	Column 6	Column 5	Column 4	Column 3	Column 2	Column 1
Base^{exp}	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Weight	128	64	32	16	8	4	2	1
Example1	1	1	0	1	1	0	1	0
Example2	1	0	0	1	1	1	0	1

Example1:

11011010 = (1 * 128) + (1 * 64) + (0 * 32) + (1 * 16) + (1 * 8) + (0 * 4) + (1 * 2) + (0 * 1) =

= 128 + 64 + 16 + 8 + 2 = **218**

Example2:

10011101 = (1 * 128) + (0 * 64) + (0 * 32) + (1 * 16) + (1 * 8) + (1 * 4) + (0 * 2) + (1 * 1) =

= 128 + 16 + 8 + 4 + 1 = **157**

Multiples of the bit (b) 1Byte = 8 bit (0;1) !

Memory unit	Description
Kilo Byte	1 KB = 1024 Bytes
Mega Byte	1 MB = 1024 KB
Giga Byte	1 GB = 1024 MB
Tera Byte	1 TB = 1024 GB
Peta Byte	1 PB = 1024 TB
Hexa Byte	1 EB = 1024 PB
Zetta Byte	1 ZB = 1024 EB
Yotta Byte	1 YB =1024 ZB
Bronto Byte	1 Bronto Byte = 1024 YB
Geop Byte	1 Geo Byte = 1024 Bronto Bytes

Anyway, binary representation of integer numbers may results almost unreadble above a dozen of digits and so even difficult to remember.

Try to write **3 TB** in binary: you'd need **40 figures** made out of 0 and 1 !

For those reasons there are many other representations for binary (or base 2) numbers:

OCTAL representation (base 8)

$$N = \sum_i k \cdot 8^j \qquad \forall i, j, k \in I \qquad j | \max(8^j, N) = 8^J$$

In examples

$$2357_8 = 1263_{10}$$

EXADECIMAL representation (base 16)

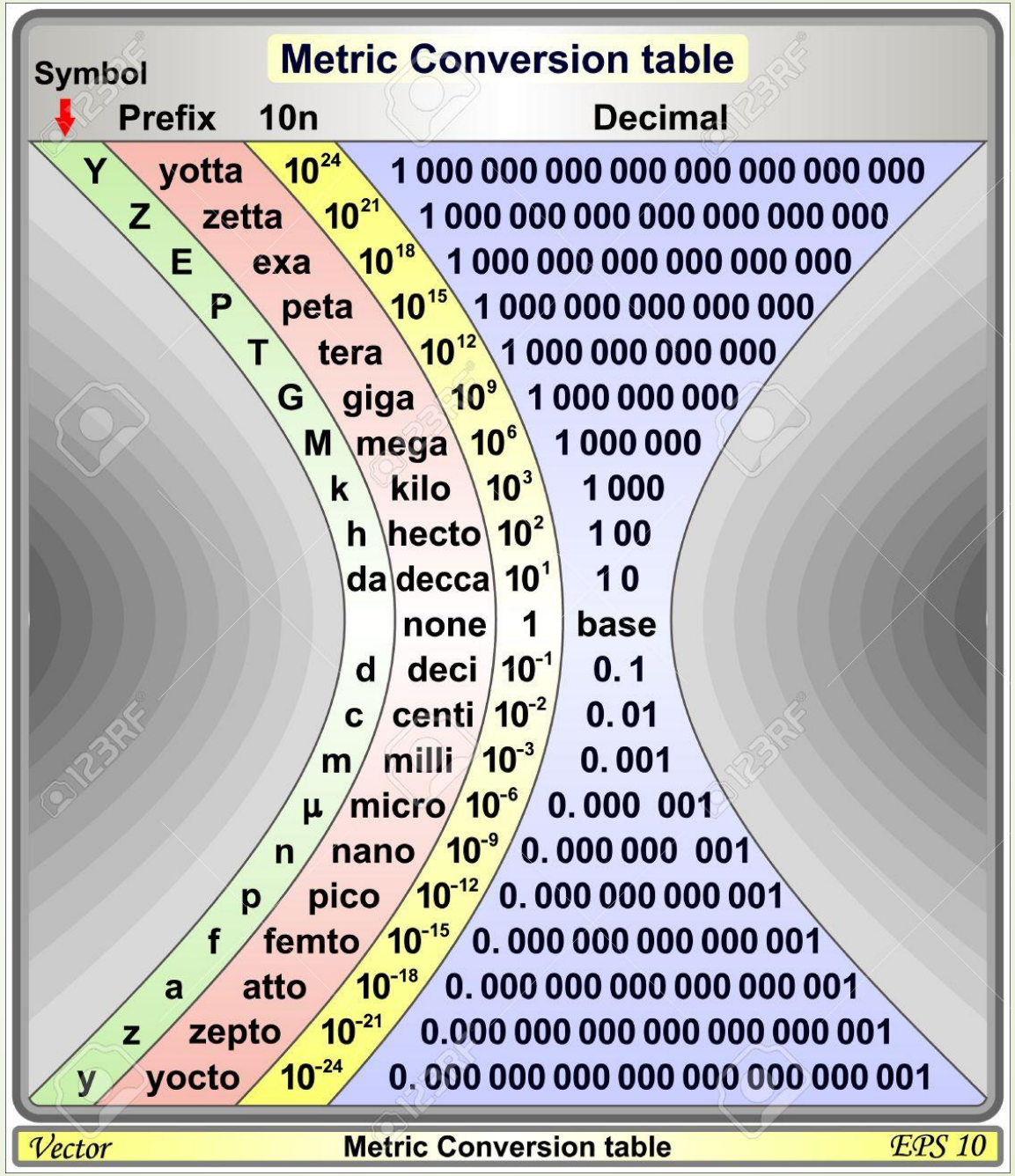
$$N = \sum_i k \cdot 16^j \qquad \forall i, j, k \in I \qquad j | \max(16^j, N) = 16^J$$

In examples

$$2C6E_{16} = 11374_{10}$$

Decimal	Exadecimal	Octal	Binary
0	0	00	0000
1	1	01	0001
2	2	02	0010
3	3	03	0011
4	4	04	0100
5	5	05	0101
6	6	06	0110
7	7	07	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Among the various representations, the exadecimal one is by far the most used. Very often an hexadecimal number has the symbol «0x» in front of the number itself.


Metric Conversion table
Symbol
Prefix
10n
Decimal
Y yotta 10^{24} 1 000 000 000 000 000 000 000 000
Z zetta 10^{21} 1 000 000 000 000 000 000 000
E exa 10^{18} 1 000 000 000 000 000 000
P peta 10^{15} 1 000 000 000 000 000
T tera 10^{12} 1 000 000 000 000
G giga 10^{9} 1 000 000 000
M mega 10^{6} 1 000 000
k kilo 10^{3} 1 000
h hecto 10^{2} 1 00
da decca 10^{1} 1 0
none 1 base
d deci 10^{-1} 0. 1
c centi 10^{-2} 0. 01
m milli 10^{-3} 0. 001
µ micro 10^{-6} 0. 000 001
n nano 10^{-9} 0. 000 000 001
p pico 10^{-12} 0. 000 000 000 001
f femto 10^{-15} 0. 000 000 000 000 001
a atto 10^{-18} 0. 000 000 000 000 000 001
z zepto 10^{-21} 0.000 000 000 000 000 000 001
y yocto 10^{-24} 0. 000 000 000 000 000 000 000 001
Vector
Metric Conversion table
EPS 10

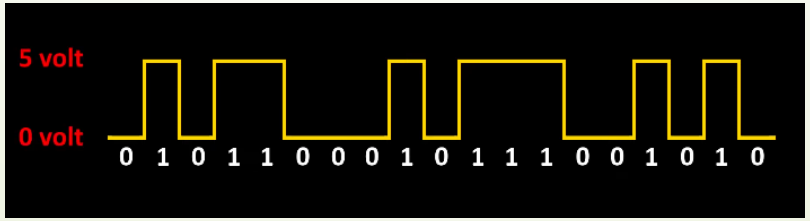
Bool algebra and electronic devices

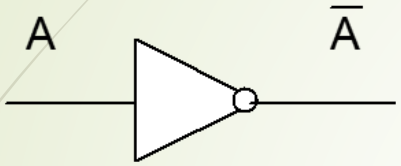
Electronic computers are build with device capable of managing discrete voltages of 0V and 5V.

Thus, it is expected that some sort of correlation should be there to connect the intrinsic logic of those devices and the binary algebra.

In fact, as for algebraic operations among decimal numbers we may write a set of equivalent operations in binary.

Let's have a look at the most basic algebraic binary operations by taking into account the digital devices implementing them at hardware level: the **LOGIC GATES** (it: porte logiche):

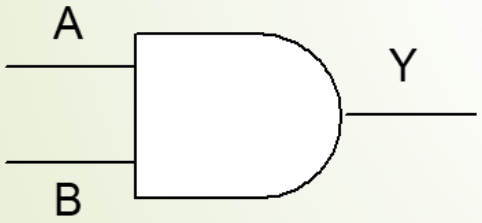




0	1
1	0

NOT
(INVERTER)

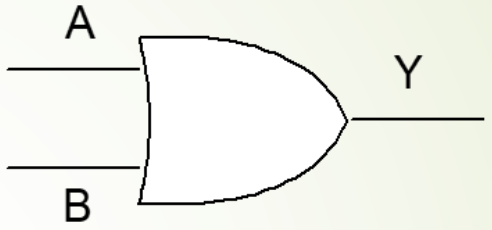
$A = \overline{A}$



0	0	0
0	1	0
1	0	0
1	1	1

AND

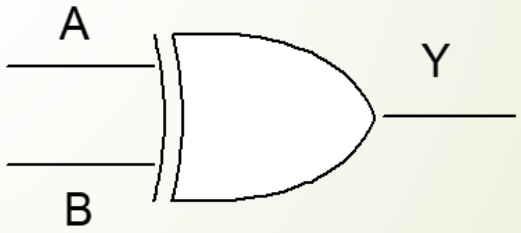
$A \cdot B = Y$



0	0	0
0	1	1
1	0	1
1	1	1

OR

A+B=Y



XOR
(OR exclusivo)

A+B=Y

Logic gates Truth Tables

NOT

Input	output
A	A^-
0	1
1	0

OR

input		output	
A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

AND

input		output	
A	B	AND	NAND
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

XOR

input		output	
A	B	XOR	XNOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

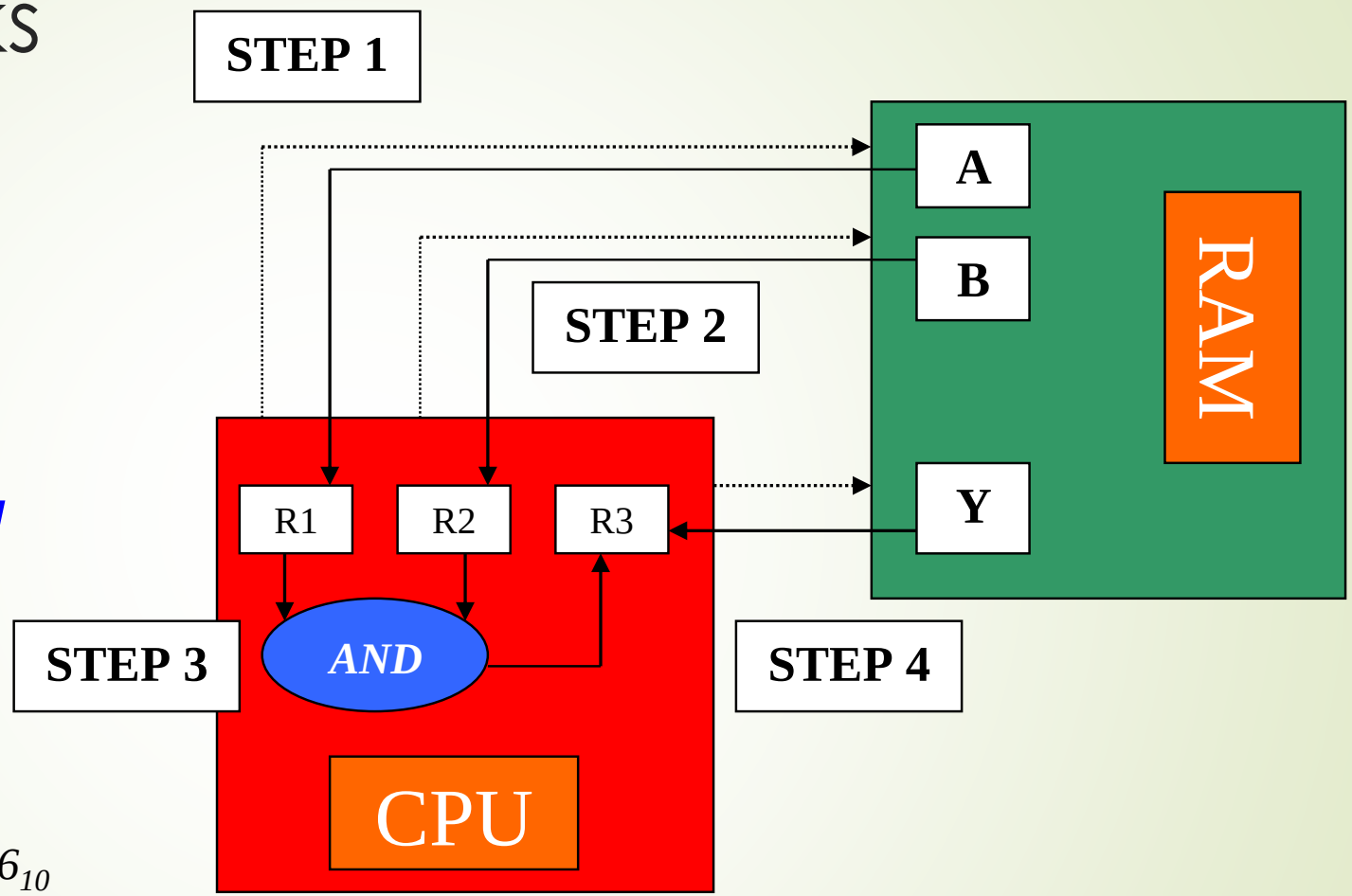
How it works

$Y = A \cdot B = A \text{ AND } B$

```
1. LoadR1[loc(A)]
2. LoadR2[loc(B)]

2a.LoadInstr[loc(AND)]

3. R3 = R1 AND R2
4. STOR3[loc(Y)]
5. END
```



$Y = A \cdot B = 182_{10} \, AND \, 100_{10} = 36_{10}$

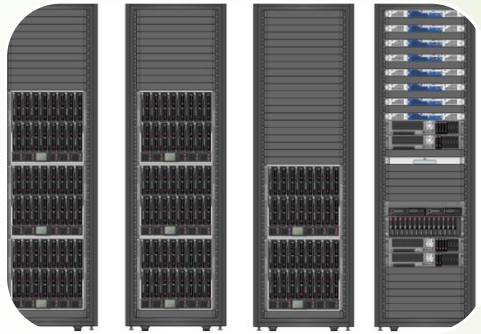
2^k	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
2_{10}^k	128	64	32	16	8	4	2	1
A	1	0	1	1	0	1	1	0
B	0	1	1	0	0	1	0	0
Y	0	0	1	0	0	1	0	0

Introduction to Computer Architectures

… back to our study by defining a basic classification of computer architectures.

The Flynn Taxonomy

SIMD



MIMD

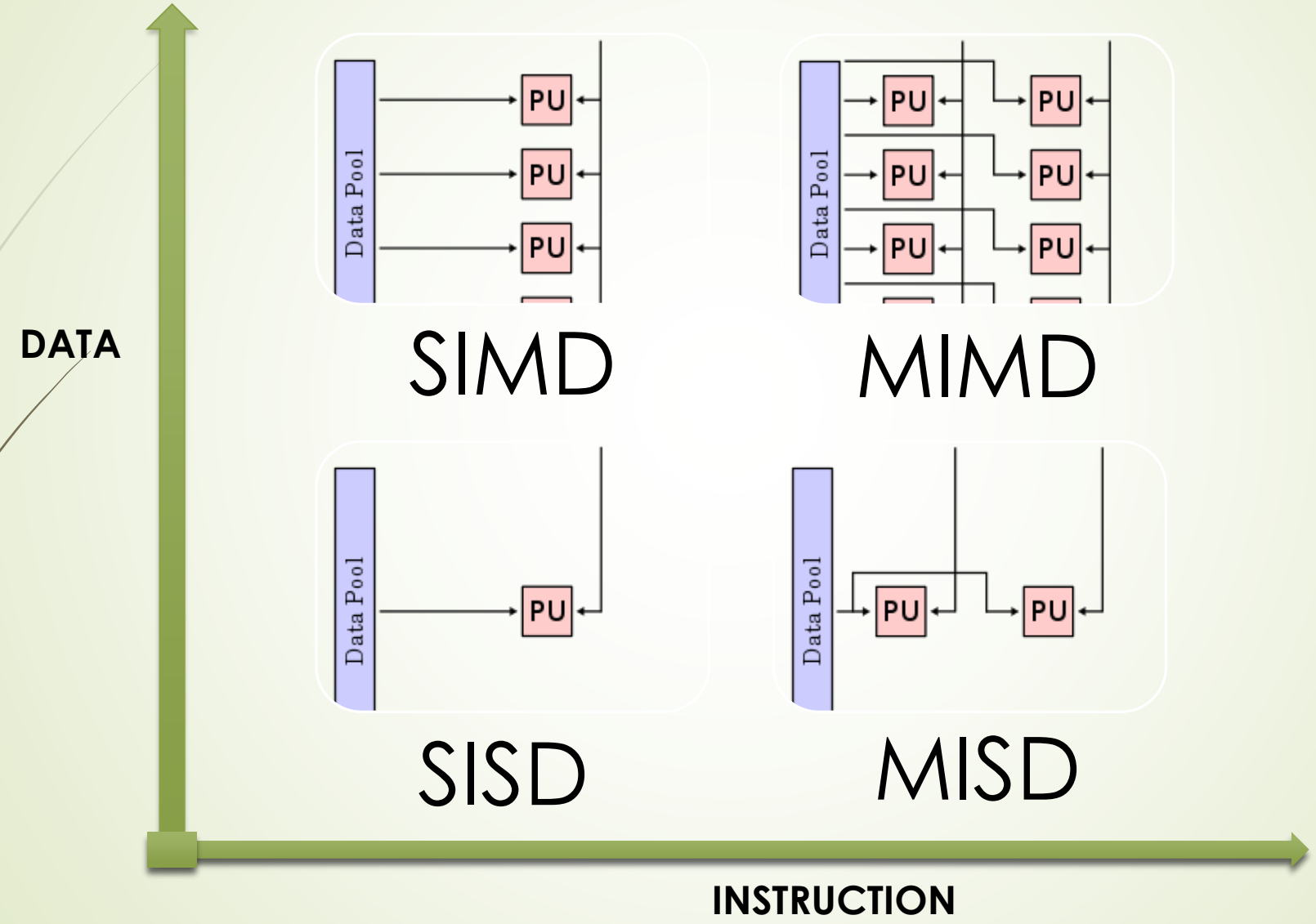
SISD

MISD

DATA

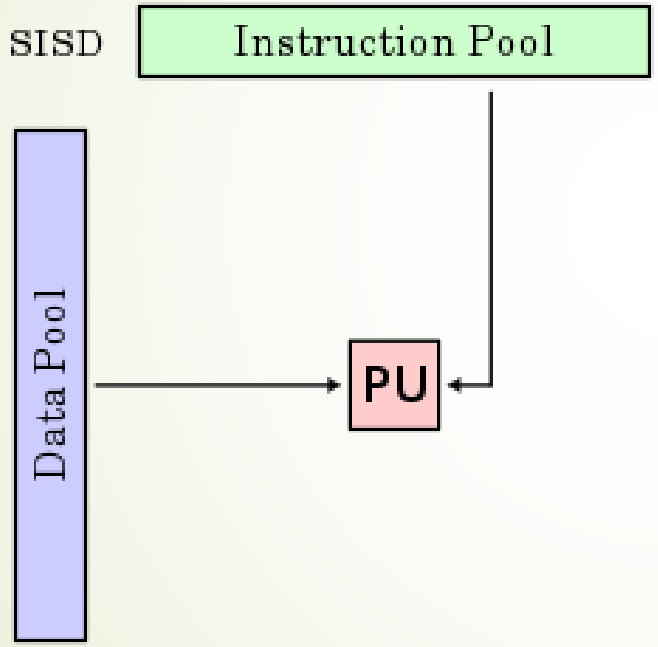
INSTRUCTION

The Flynn Taxonomy



The Flynn Taxonomy

Single **I**nstruction Stream - - **S**ingle **D**ata Stream

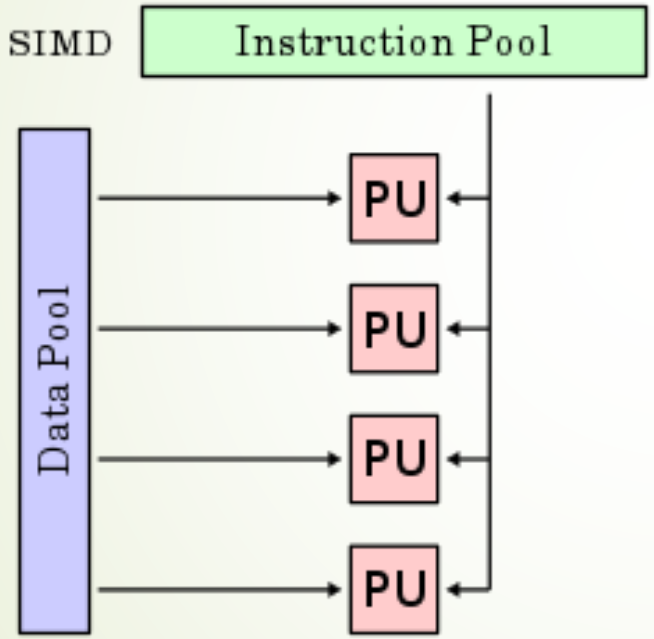


Von Neumann Architecture

Single Core PC

The Flynn Taxonomy

Single **I**nstruction Stream - - **M**ultiple **D**ata Stream

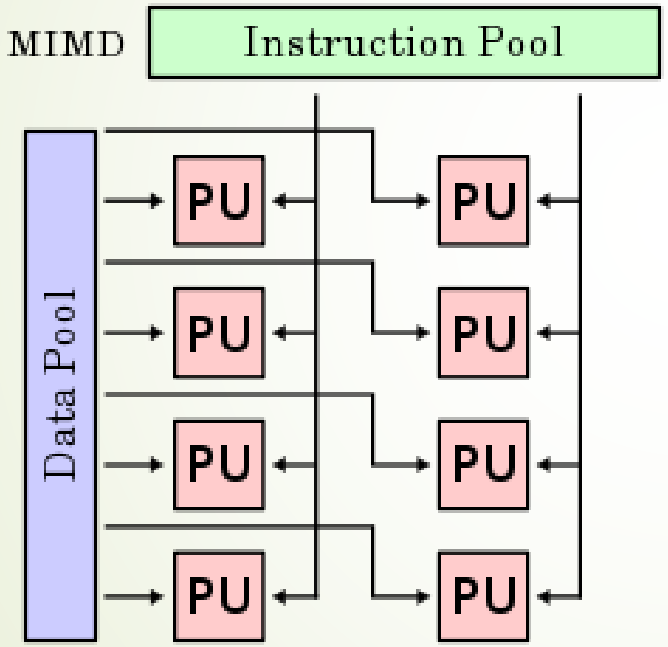


One Instruction repeated over many different data

Intel SSE/AVX
GPU (Streaming Processor - SP)

The Flynn Taxonomy

Multiple **I**nstruction Stream - - **M**ultiple **D**ata Stream

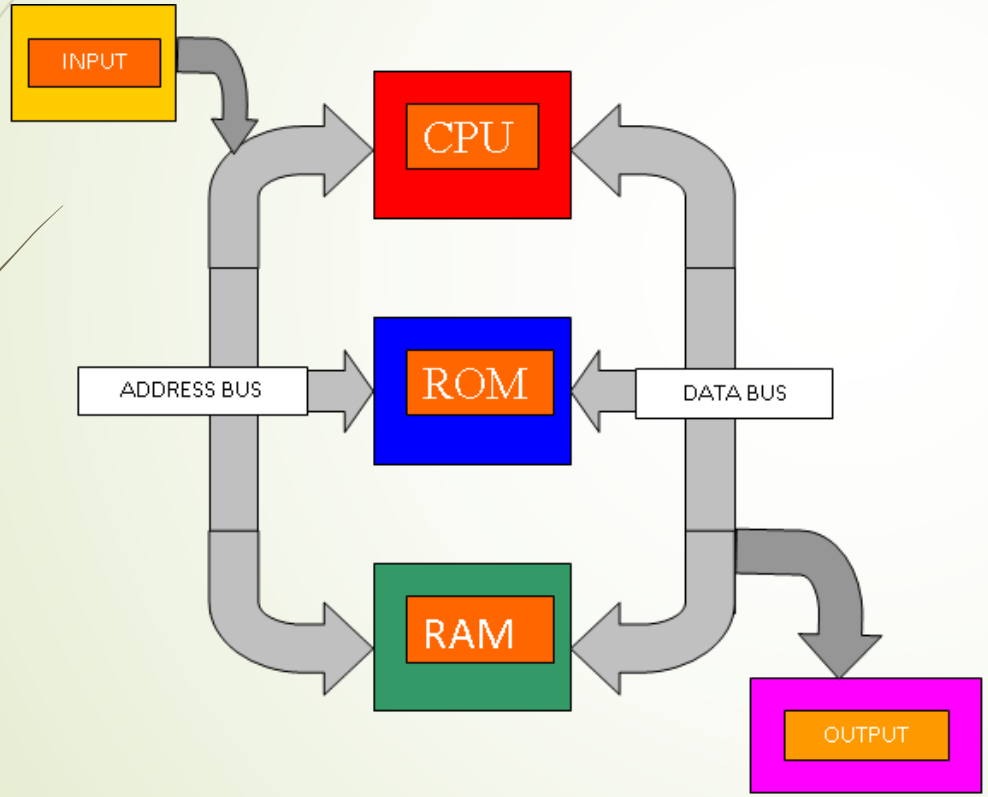


Many different Instructions applied over many different data

Multi-core PC
Clusters
GPU (Many SP)

The Harvard Architecture

Two separated buses for instructions and data

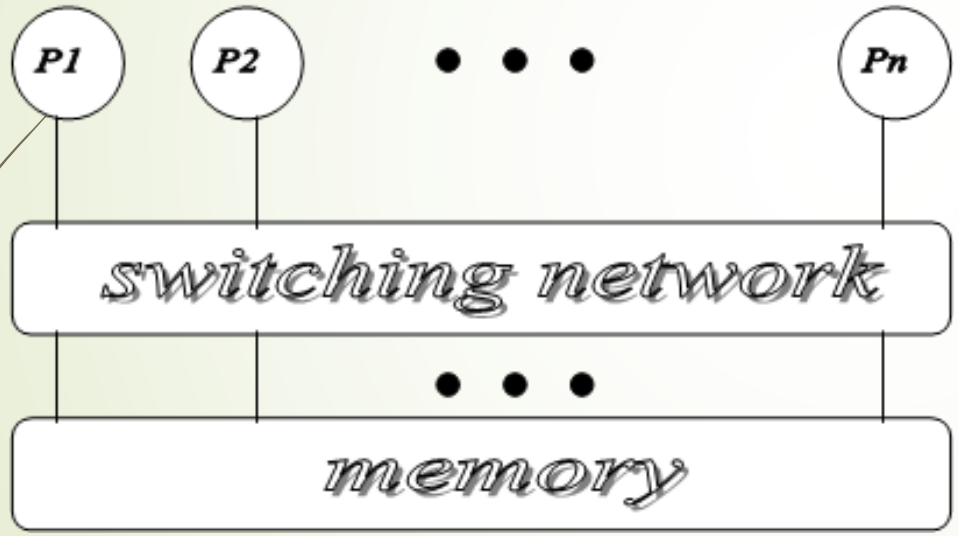


The Harvard architecture is implemented on all modern RISC/CISC CPUs.

Multi-core PC
Clusters
GPU (Many SP)

Shared Memory Architectures

A large memory bank shared by all processors

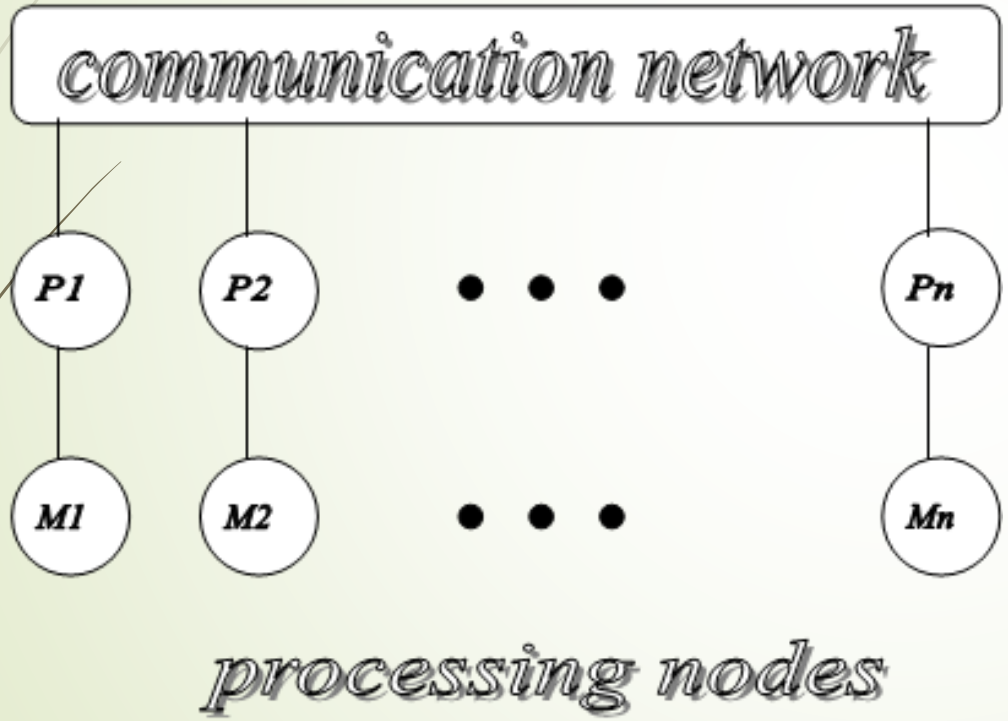


This architecture give access to the memory data through a switched bus, in a way where all processors potentially should have the same latency (delay).

Symmetric/Shared Multi Processor (SMP)
Multi-core PC

Distributed Memory Architectures

A distributed set of private memory banks shared via network by all processors

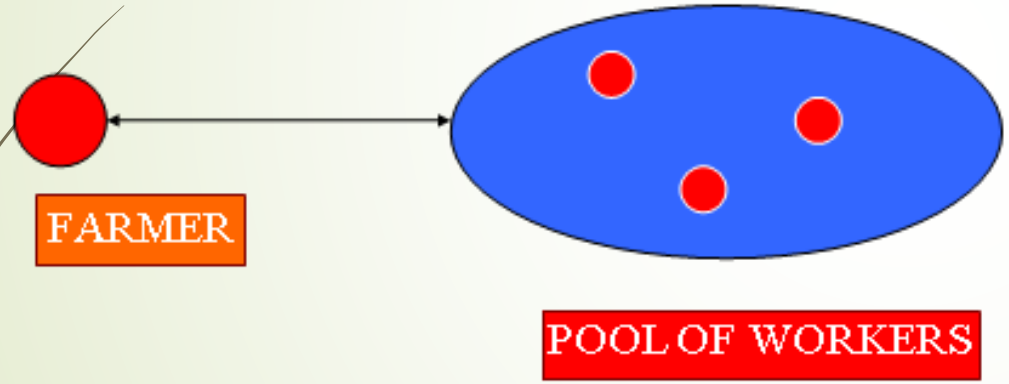


In this architecture the access to the memory data is given through a switched network, where all processors share their memories with the same latency (delay) depending on the network speed.

Clusters
Clusters of GPUs

Parallel Computing Models

A conceptual view – Master/Slaves – Farmer/Workers Model

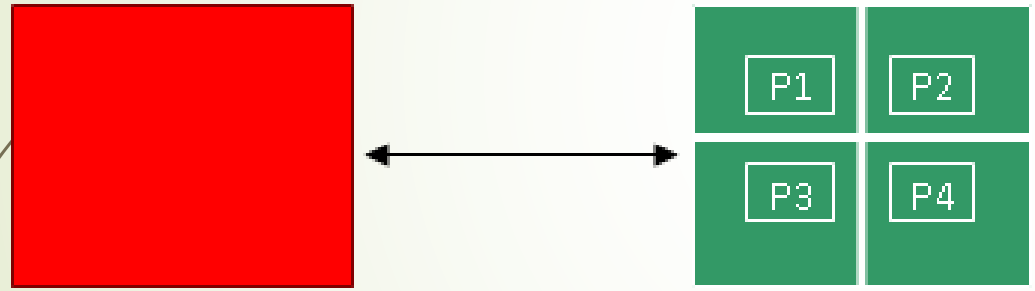


In this model of programming a main process acts as a front-end to the computing processes/threads by coordinating their activities.

Clusters
Clusters of GPUs

Parallel Computing Models

A conceptual view – Data Parallel Model



In this model of programming a dataset is subdivided into chunks private to all processors.

Clusters (OpenMP)
Clusters of GPUs

Parallel Computing Models

A system view – Imagine program and data separately

SD Single Data Space (no distribution)
MD Multiple Data Space (distribution)

SP Single Program
MP Multiple (different) Programs
* Replication Operator (multiple copies)

In this model of programming one has to think to his own program and data separately, thus abstracting the underlying hardware.

Clusters (MPI)
Clusters of GPUs

Parallel Computing Models

A system view – P/D/* qualifiers in action

SPSD Single Program Single Data Space (no distribution)

SPMD Single Program Multiple Data Space (distribution)

MP*MD Multiple (different) Programs Multiple Data w/ Replication Operator (multiple copies)

SPSD describes a serial program acting on a unique dataset.
SPMD refers to a single program acting on a distributed dataset.
MPMD is the most flexible model where any program/dataset will be distrubute all over the available processors and memories.

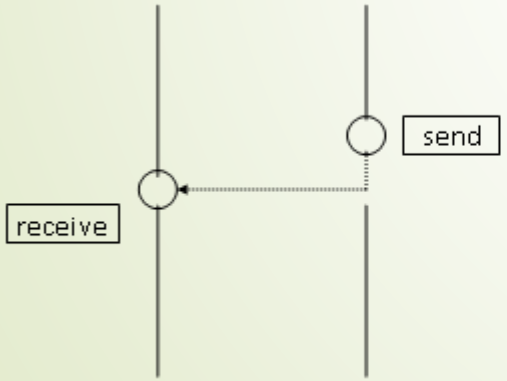
Clusters (OpenMP+MPI)
Clusters of GPUs

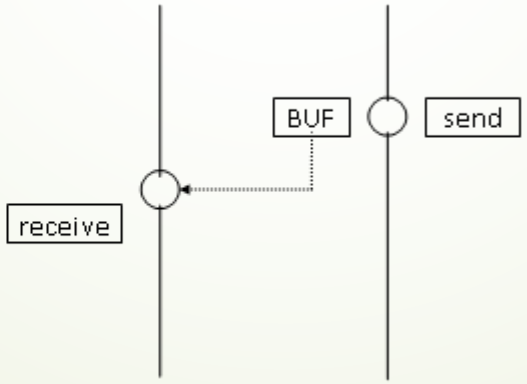
Parallel Computing Archs&Models

Message Passing Interface - MPI

```
send (<destinazione>, <nome_variabile>)

receive (<sorgente>, <nome_variabile>)
```





The Message Passing Interface is based on a programming model which refers to processes capable to send/receive distributed variables. Communications could be **synchronous** (blocking) or **asynchronous** (non-blocking).

Clusters (OpenMP+MPI)
Clusters of GPUs

Parallel Computing Archs&Models

Today, Parallel Computers are Distribuited Shared Memory MultiProcessors

Modern computing architectures are based on replicated building blocks of Shared Memory Processors (SMP) interconnected with high bandwidth / low latency communication subsystem.
Those systems are referreto as «**MULTICORE CLUSTER**».

When multicore cluster nodes are specialized with additional devices to support the CPUs for computing intensive labor (i.e., FPGA, GPU), we refer to «**MANYCORE CLUSTER**».

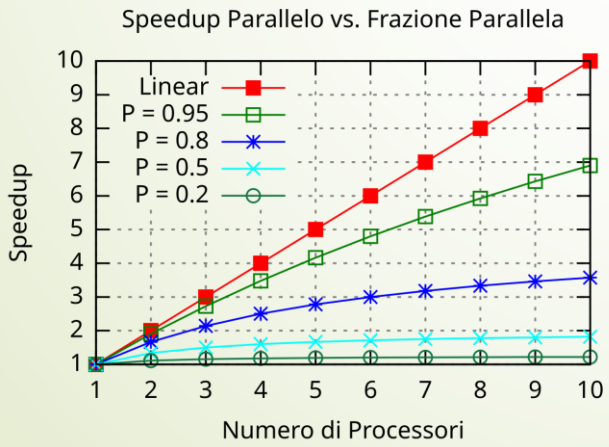
The majority of supercomputers are either multicore or more recently, manycore (w/ GPU). The computing architecture is referred as DSM(M)P and the programming models of preference are **SPMD** (via OpenMP+MPI) and Data Parallel into the GPU via CUDA.

Clusters (OpenMP+MPI)
Clusters of GPUs

Parallel Computing Performance

The Amdhal Law "Make the common case fast" (rendi veloce il caso più frequente)

$$\frac{1}{S} = F + \frac{(1-F)}{P}$$

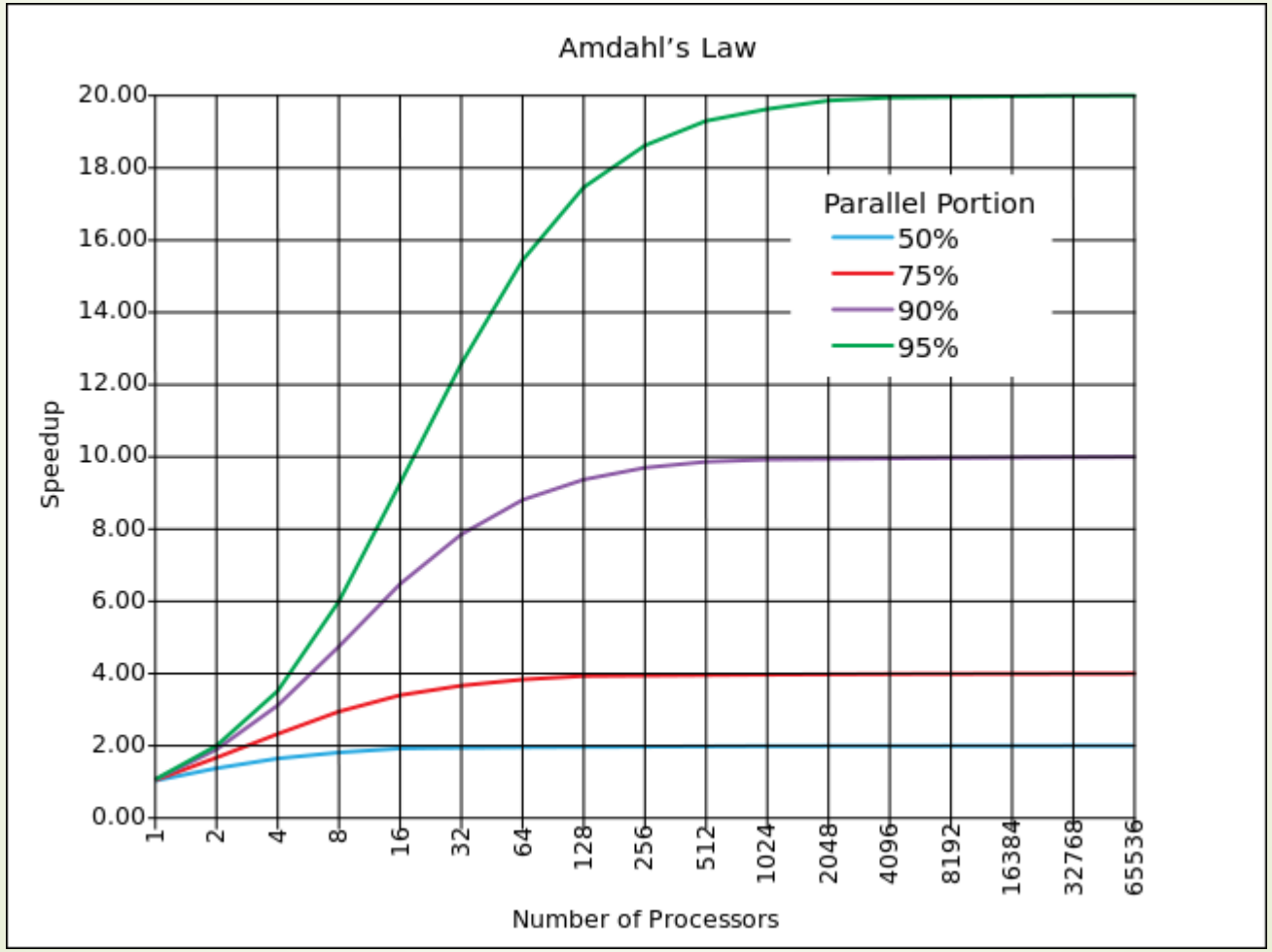


S is the SpeedUP (SU) of your parallel code when running over **P** processors where a fraction **F** of the program will run in sequential (NOT parallel) mode.

DSMP Clusters (OpenMP+MPI)
DSMP Clusters of GPUs

Parallel Computing Performance

The Amdhal Law



Parallel Computing Performance

The Amdhal Law with overheads

$$S = \frac{1}{F + \frac{(1-F)}{P} + K_1 + K_2 P^2}$$

In real cases, when you run your parallel code you encounter overheads, i.e., delay in the execution time due to architectural constrains or other limiting factor to the computing performance. Here we take into account in the modified Amdahl law of an overhead independent by the number of processors **P** and a second term which has a quadratic dependency vs **P**.

DSMP Clusters (OpenMPI)
DSMP Clusters of GPUs

Parallel Computing Performance

The Gustafson Law – taking into account dataset size

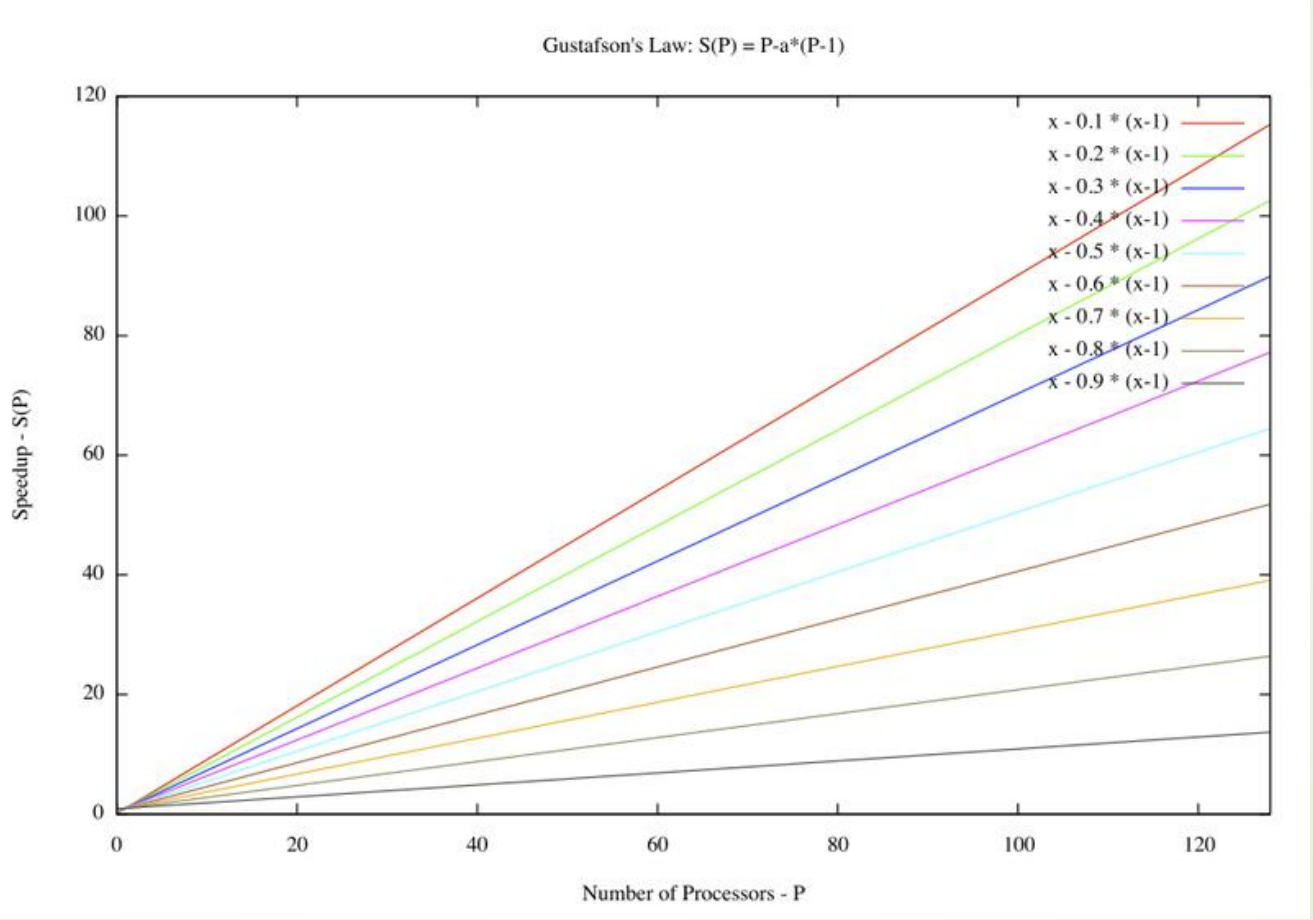
$$S(P) = P - F\,(P-1)$$

Gustafson's Law (also known as **Gustafson-Barsis' law**) is a law in computer science which says that computations involving arbitrarily large data sets can be efficiently parallelized. Gustafson's Law provides a counterpoint to Amdahl's law, which describes a limit on the speed-up that parallelization can provide, given a fixed data set size.

S speedup
P # processors
F sequential fraction

Parallel Computing Performance

The Gustafson Law – SCALED SPEEDUP



Computing Performance Measurements

In computing, floating point operations per second (**FLOPS**, flops or flop/s) is a measure of computer performance, useful in fields of scientific computations that require floating-point calculations. For such cases it is a more accurate measure than measuring instructions per second.
The similar term FLOP is often used for floating-point operation, for example as a unit of counting floating-point operations carried out by an algorithm or computer hardware.

Floating-point arithmetic
Floating-point arithmetic is needed for very large or very small real numbers, or computations that require a large dynamic range. Floating-point representation is similar to scientific notation, except everything is carried out in base two, rather than base ten. The encoding scheme stores the sign, the **exponent** and the **significand** or **mantissa** (number after the radix point). While several similar formats are in use, the most common is ANSI/IEEE Std. 754-1985. This standard defines the format for **32-bit** numbers called **single precision**, as well as **64-bit** numbers called **double precision** and longer numbers called extended precision (used for intermediate results).

© Wikipedia

Computing Performance Measurements

Computational performance

FLOPS and MIPS are units of measure for the numerical computing performance of a computer. Floating-point operations are typically used in fields such as scientific computational research. The unit MIPS measures integer performance of a computer. Examples of integer operation include data movement (A to B) or value testing (If A = B, then C). MIPS as a performance benchmark is adequate when a computer is used in database queries, word processing, spreadsheets, or to run multiple virtual operating systems. Frank H. McMahon, of the Lawrence Livermore National Laboratory, invented the terms FLOPS and MFLOPS (megaFLOPS) so that he could compare the supercomputers of the day by the number of floating-point calculations they performed per second. This was much better than using the prevalent MIPS to compare computers as this statistic usually had little bearing on the arithmetic capability of the machine.

Computer performance

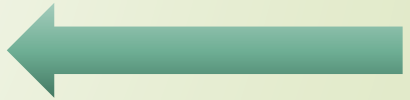
Name	Unit	Value
kiloFLOPS	kFLOPS	10^3
megaFLOPS	MFLOPS	10^6
gigaFLOPS	GFLOPS	10^9
teraFLOPS	TFLOPS	10^{12}
petaFLOPS	PFLOPS	10^{15}
exaFLOPS	EFLOPS	10^{18}
zettaFLOPS	ZFLOPS	10^{21}
yottaFLOPS	YFLOPS	10^{24}

© *Wikipedia*

Computing Performance Measurements

Top500

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
5	**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	148.60	200.79	10,096



© top500.org