

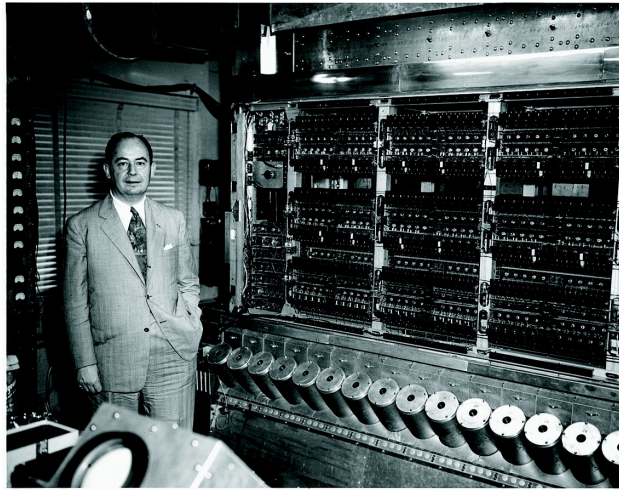


ARQUITECTURA DEL COMPUTADOR

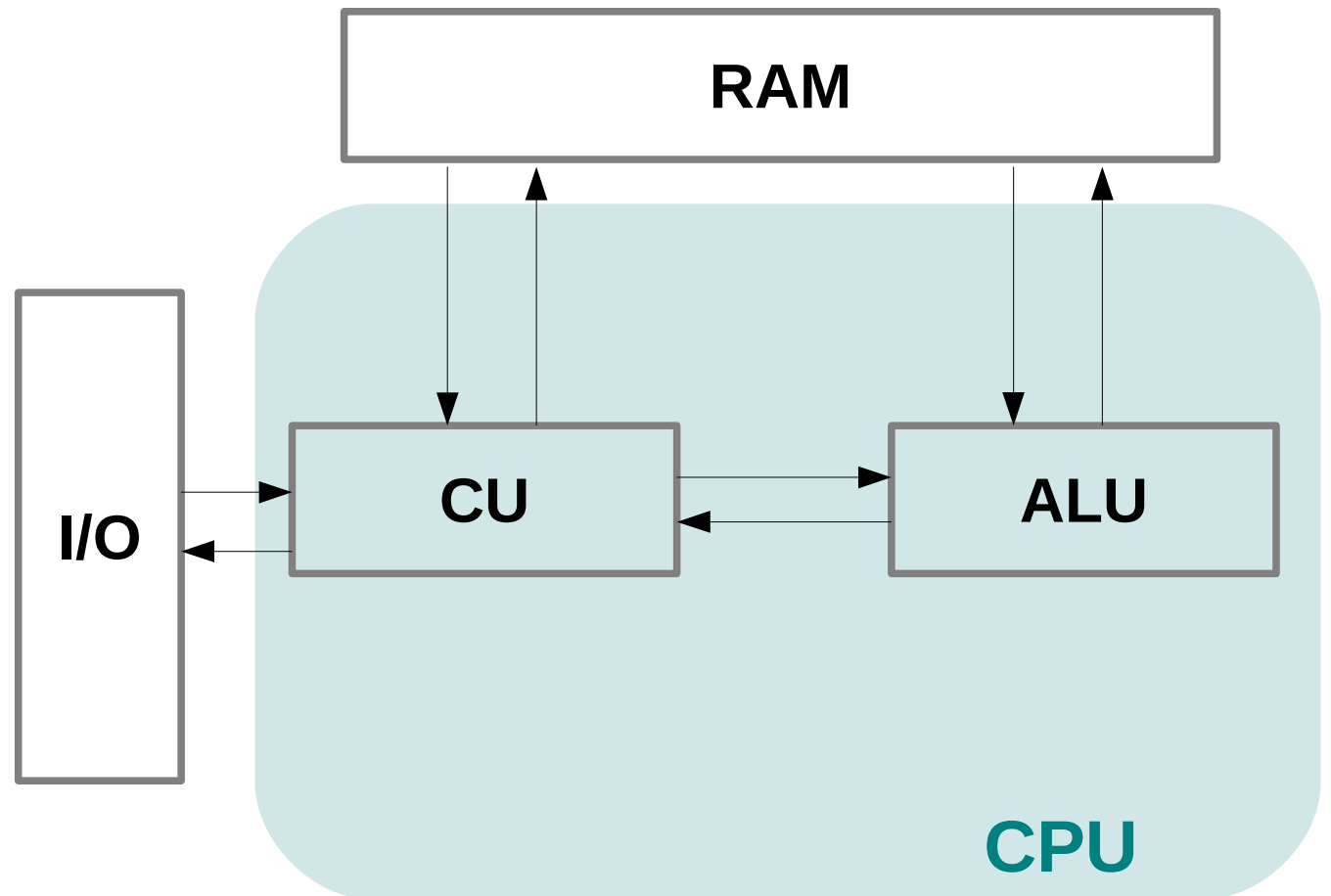
Graciela Molina

`gmolina@herrera.unt.edu.ar`
`m.graciela.molina@gmail.com`

MODELO VON NEUMANN



J. Von Neumann
frente a la
computadora IAS,
1952.

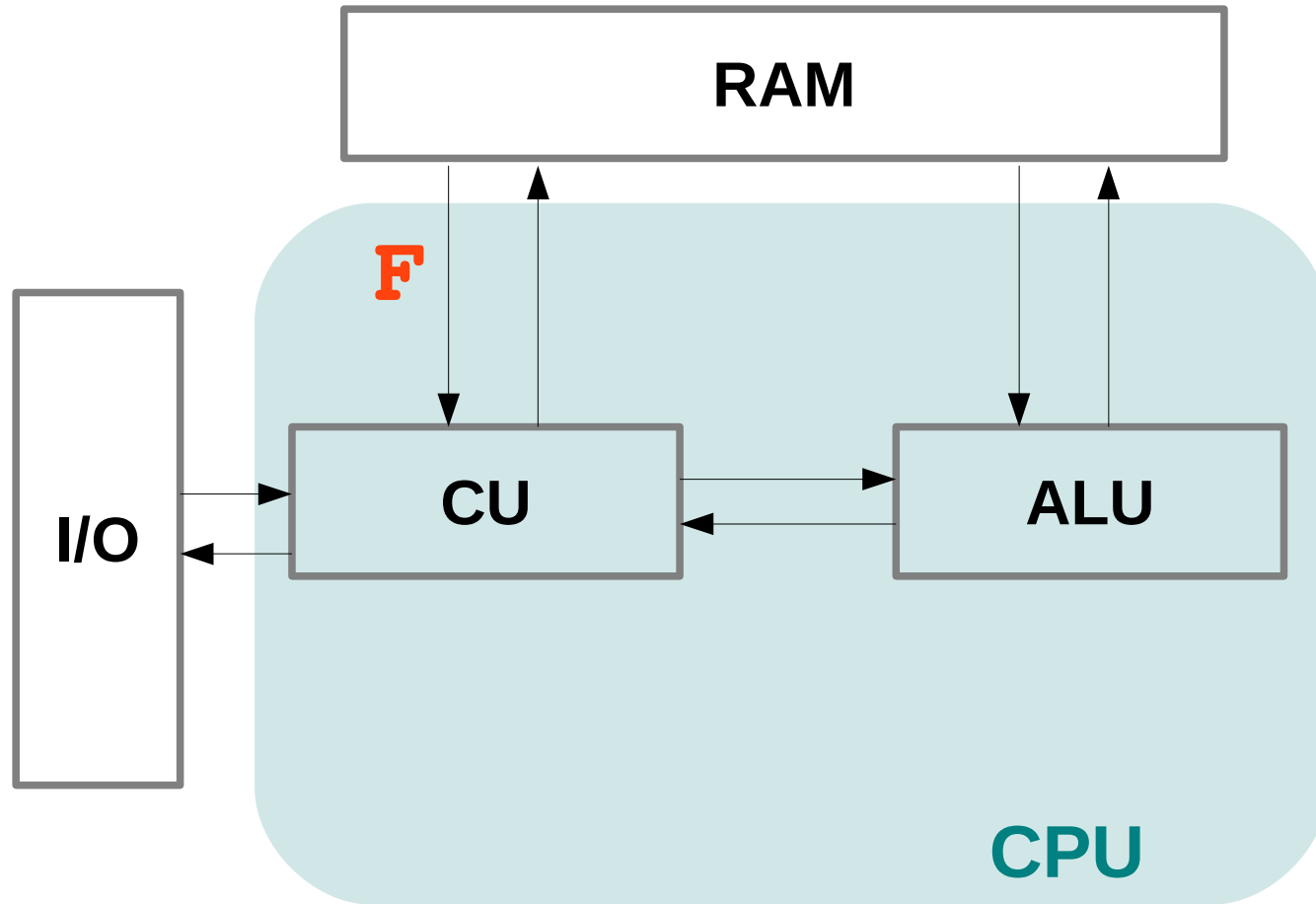


<https://www.ias.edu/people/vonneumann/ecp>

MODELO VON NEUMANN

¿Cómo trabaja?

1. FETCH: lee próxima instrucción

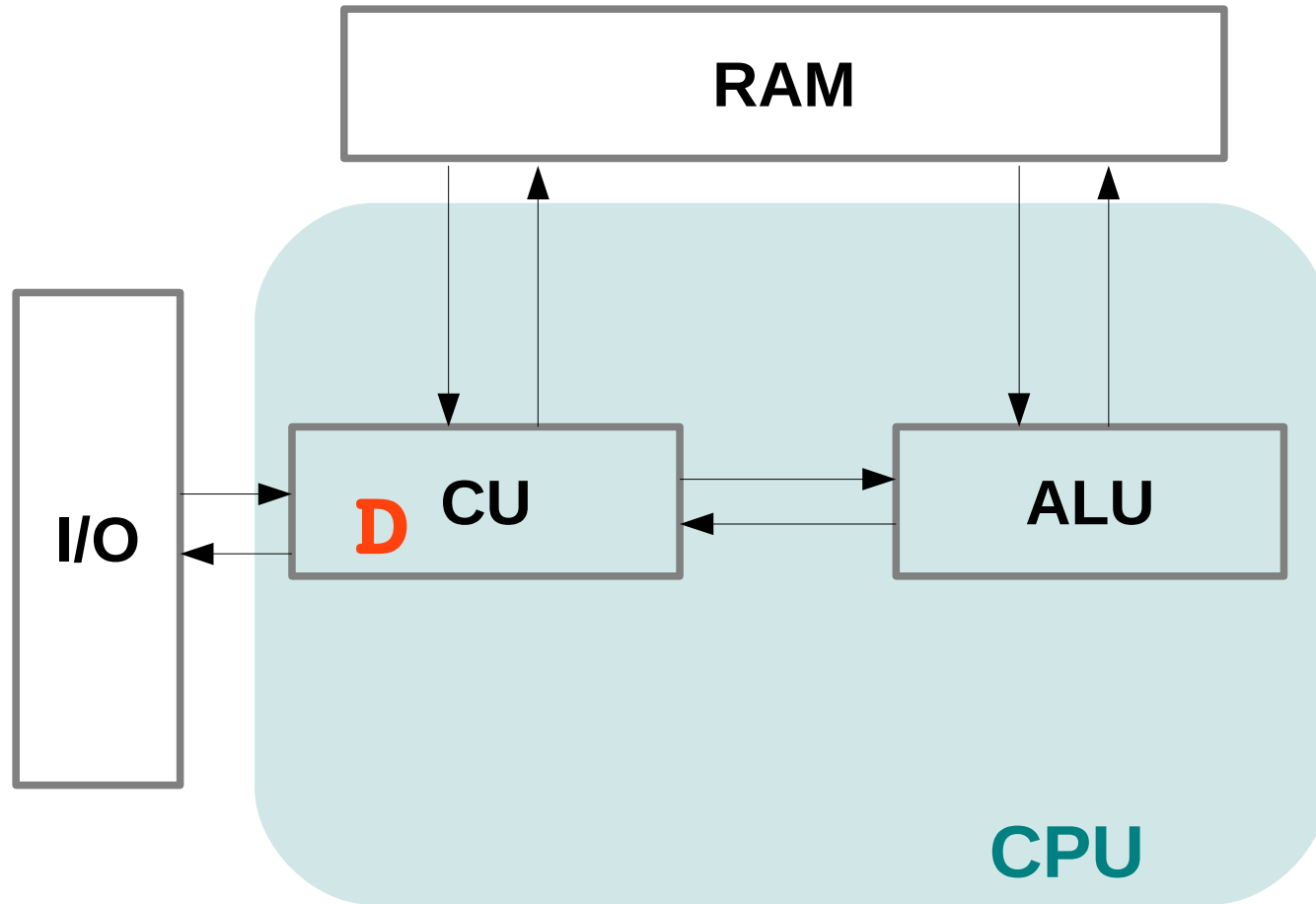


0001 1101 0011 0110 1001 ?

MODELO VON NEUMANN

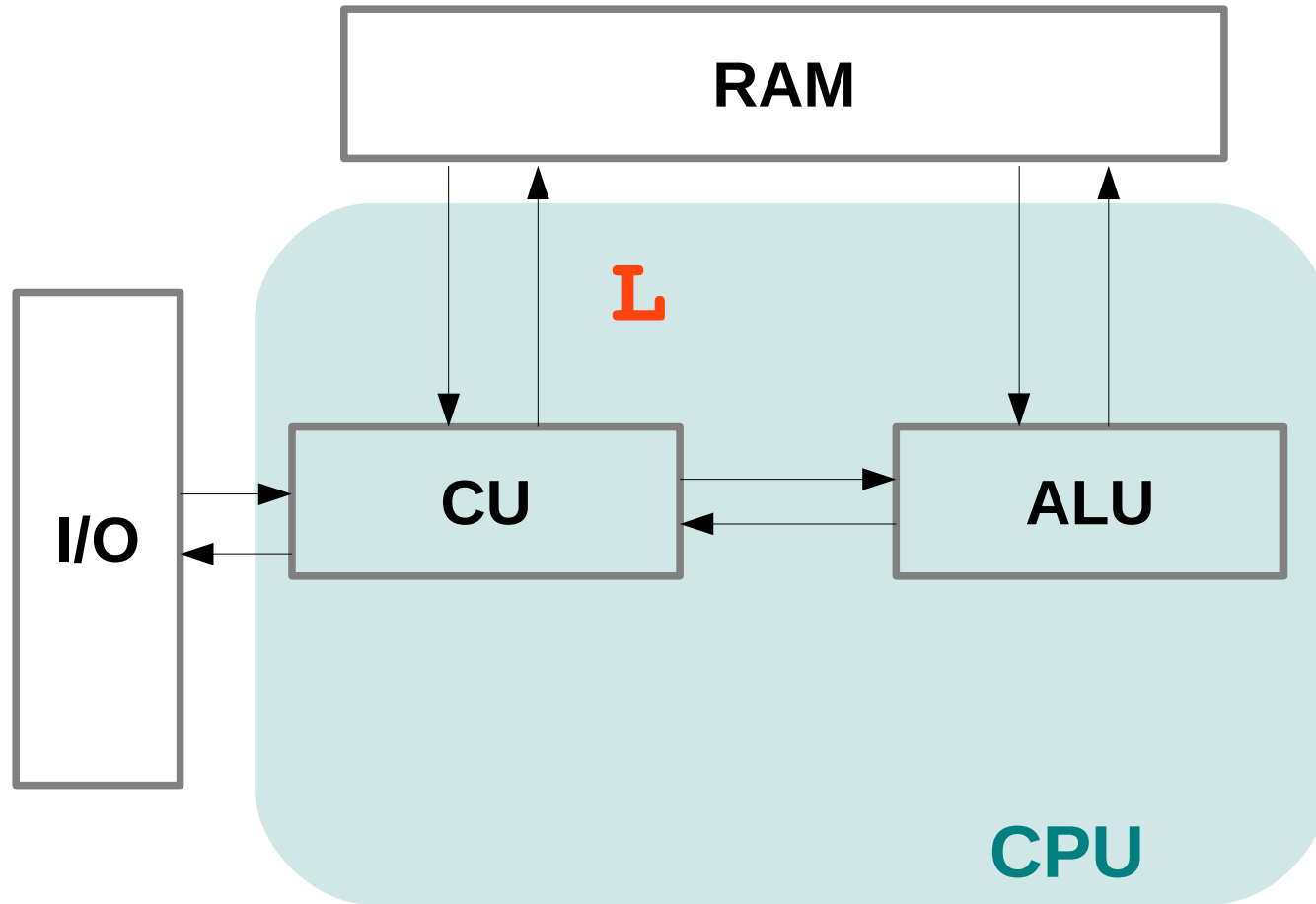
¿Cómo trabaja?

2. DECODE: decodifica la instrucción



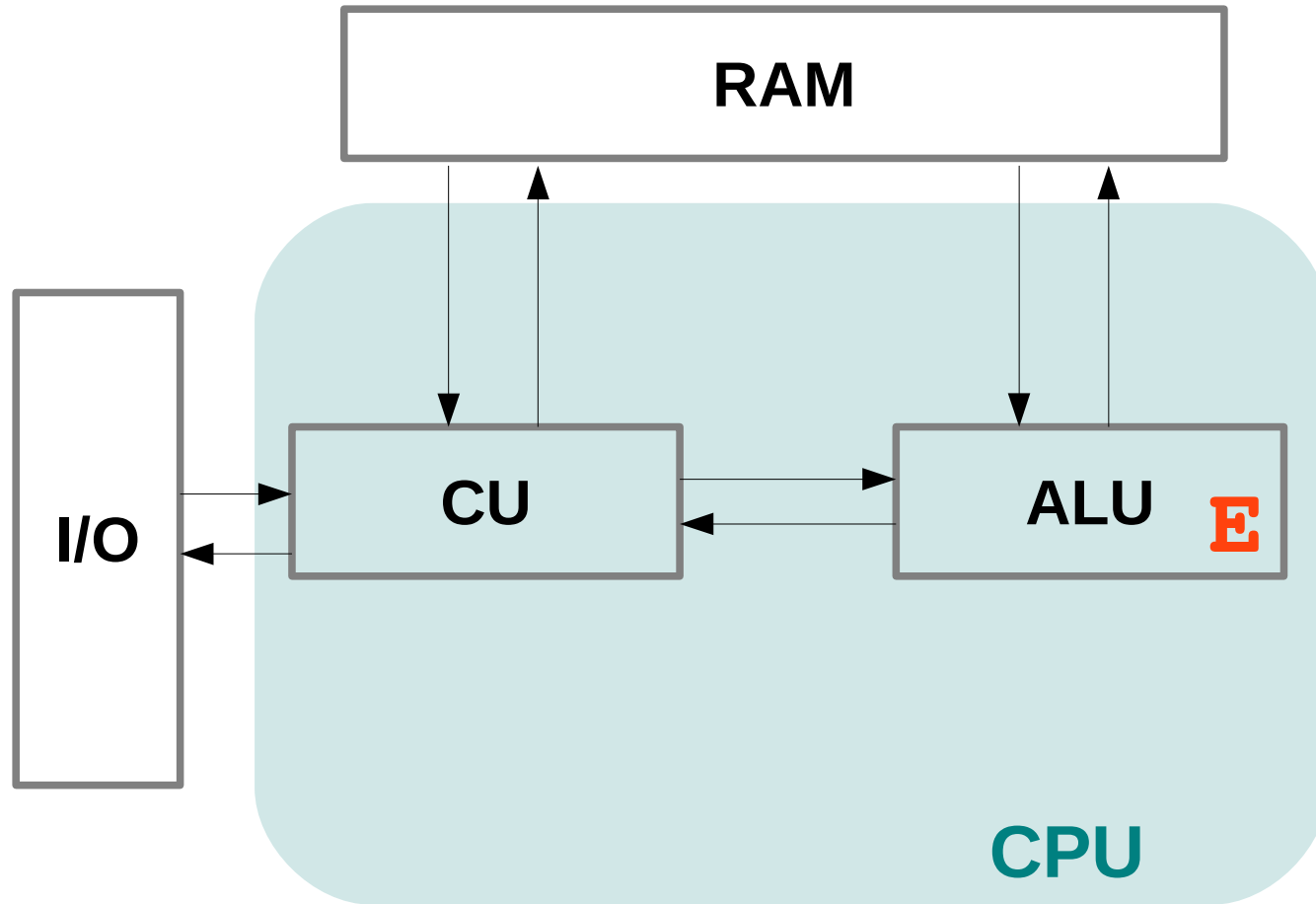
`add r1, 0xF21E` !

3. LOAD: lo que pide la instrucción



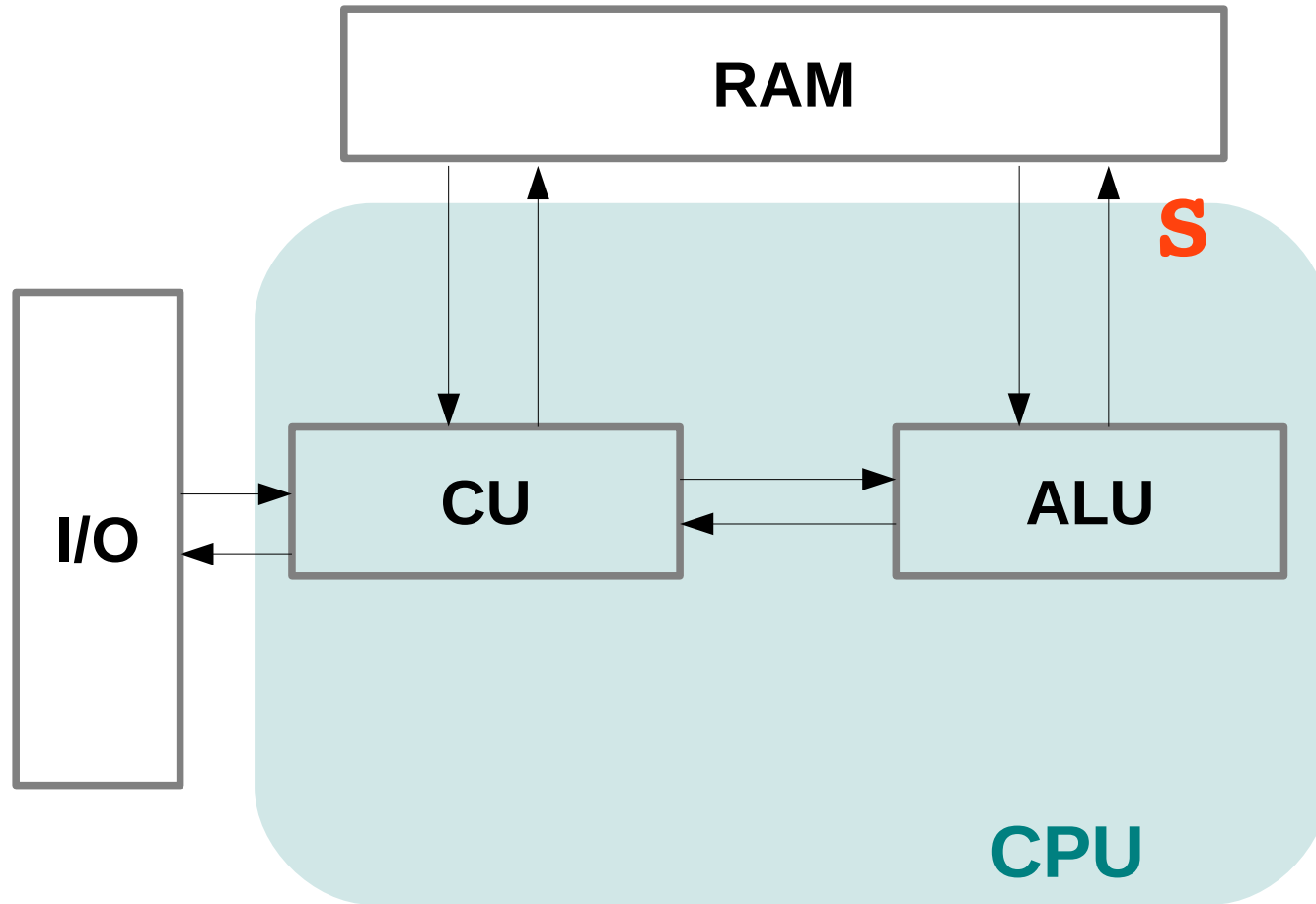
$r2 \leftarrow 0xF21E$

4. EXECUTE: la operación



$r1 \leftarrow r1 + r2$

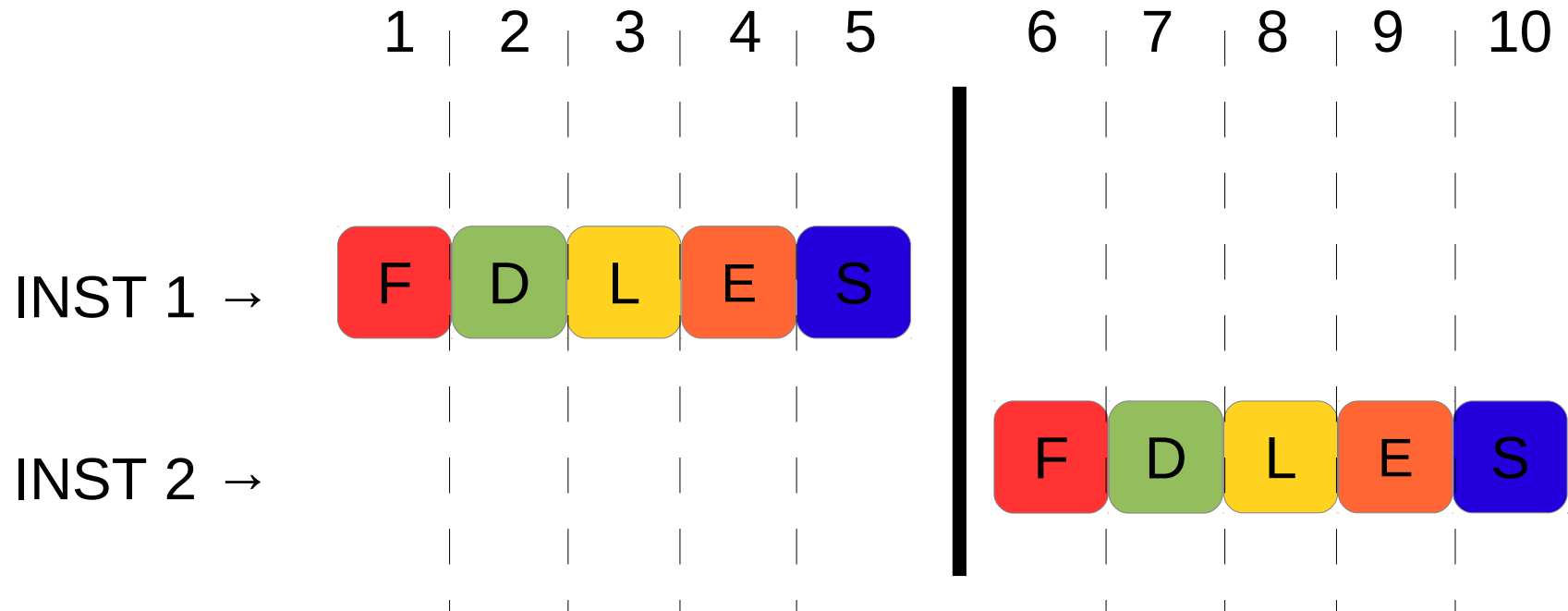
5. STORE: guarda el resultado en memoria



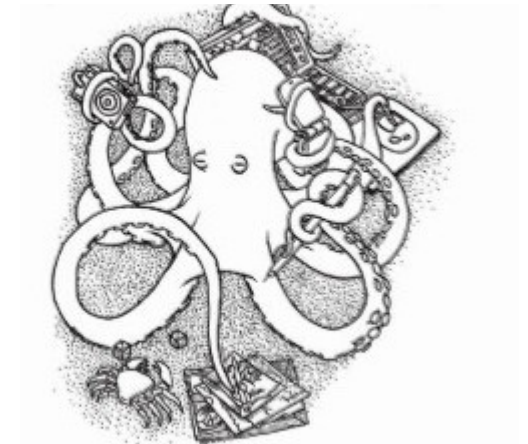
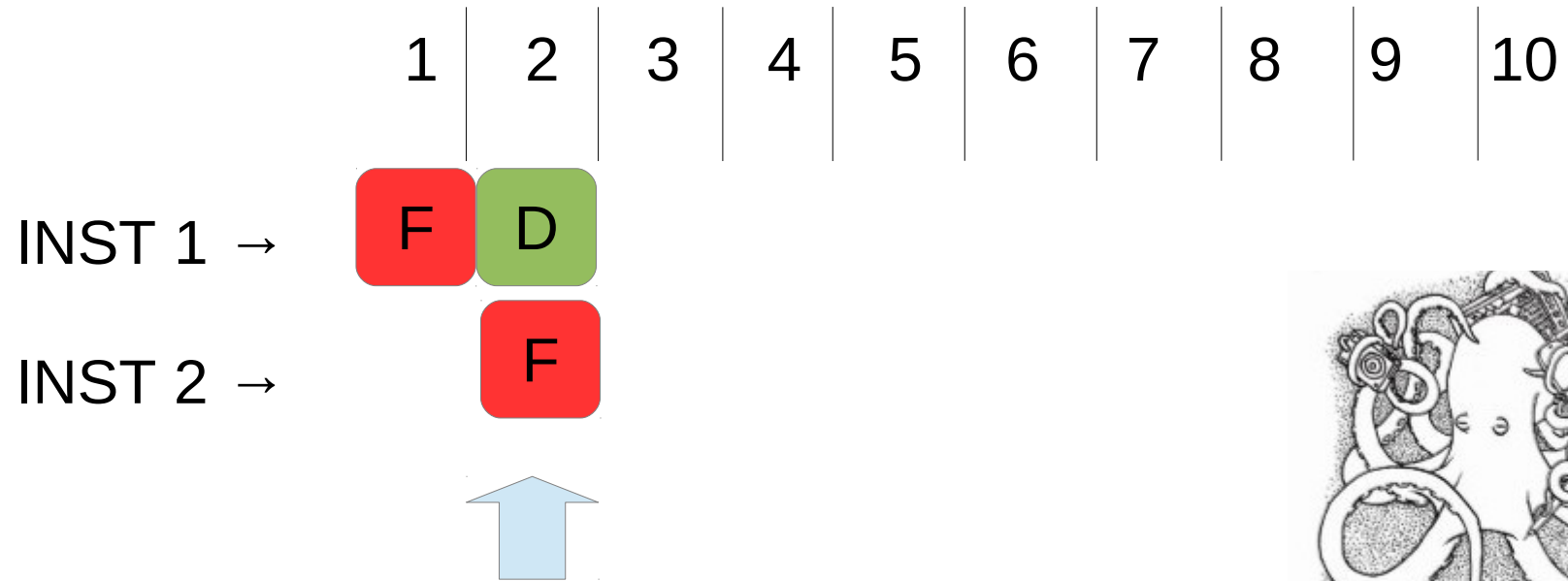
RAM \leftarrow r1

¿Puede hacerse más eficiente el
trabajo?

PROCESAMIENTO SECUENCIAL



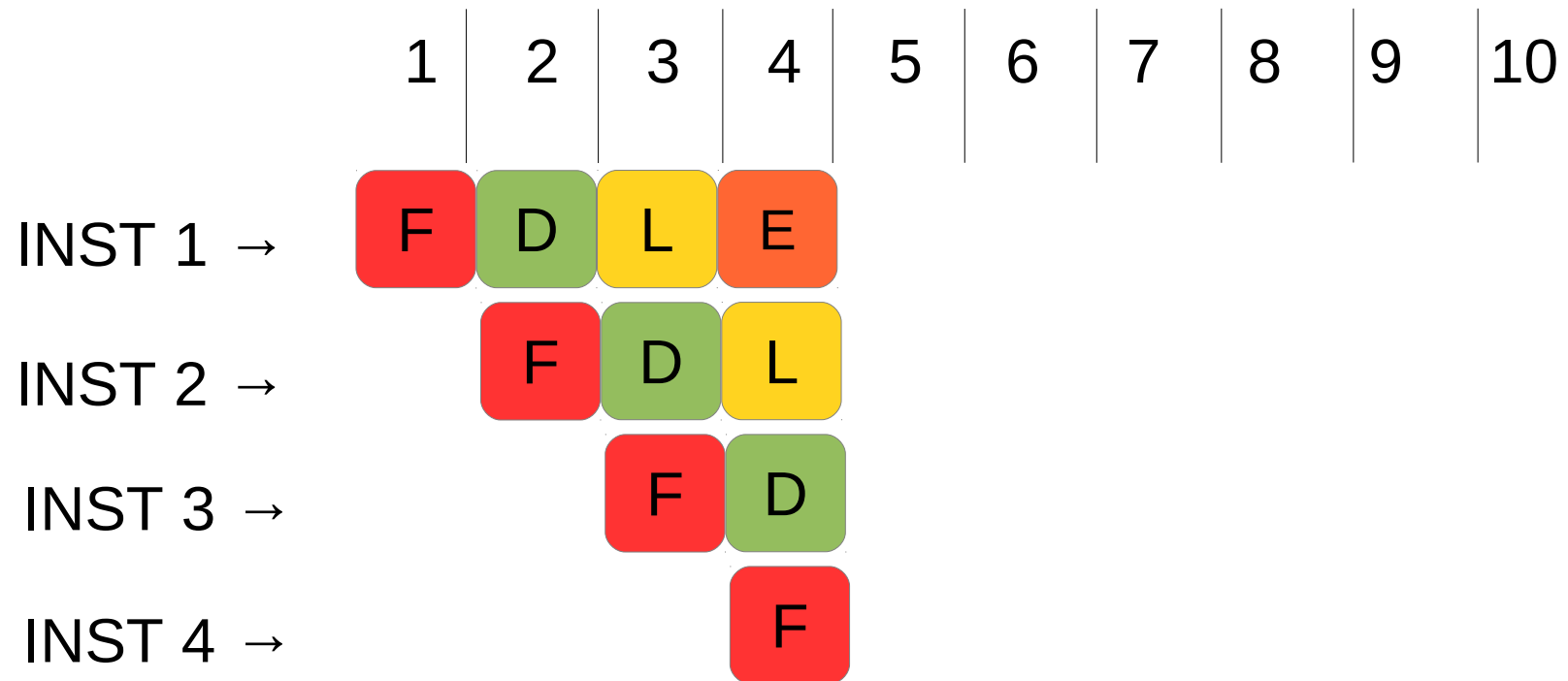
PIPELINING



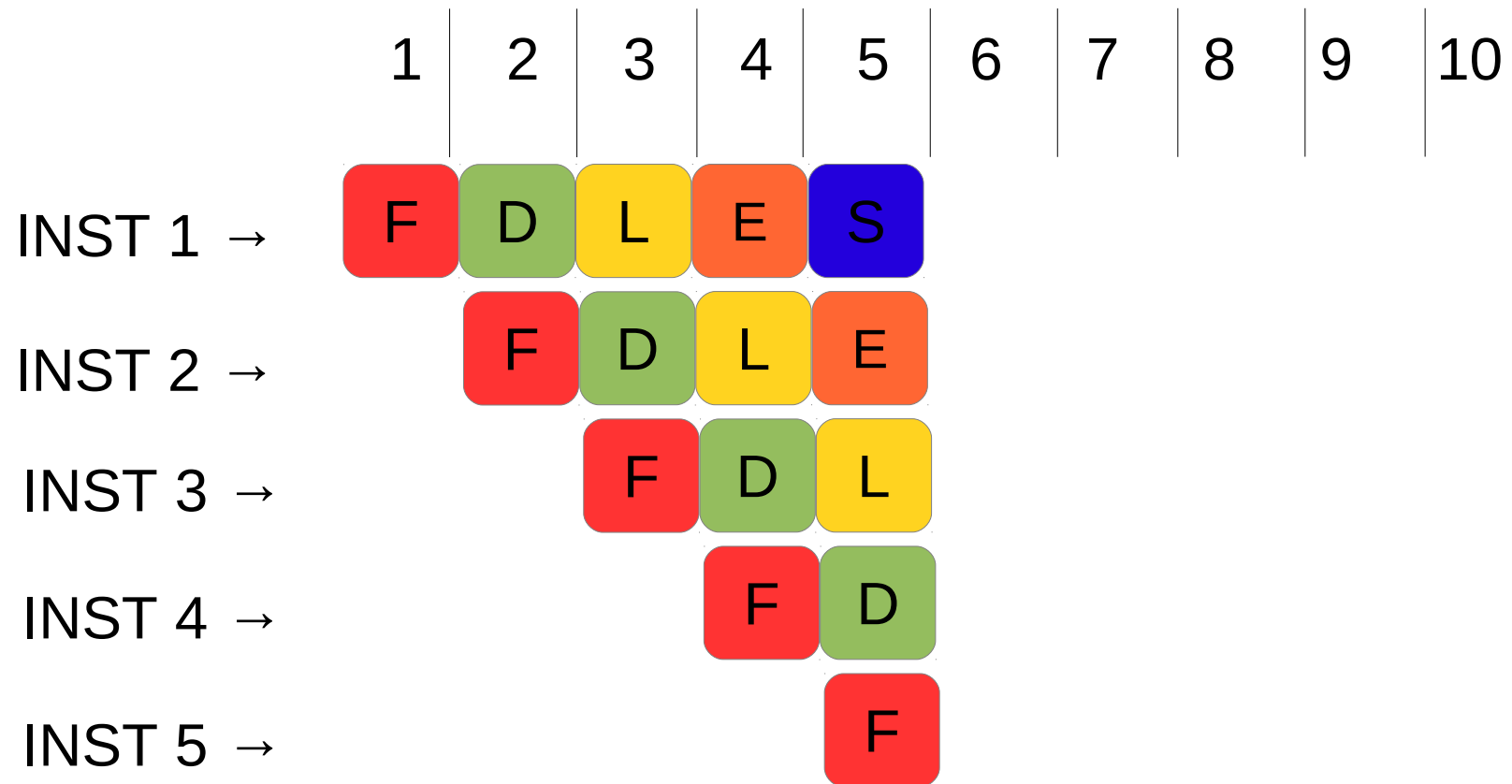
PIPELINING



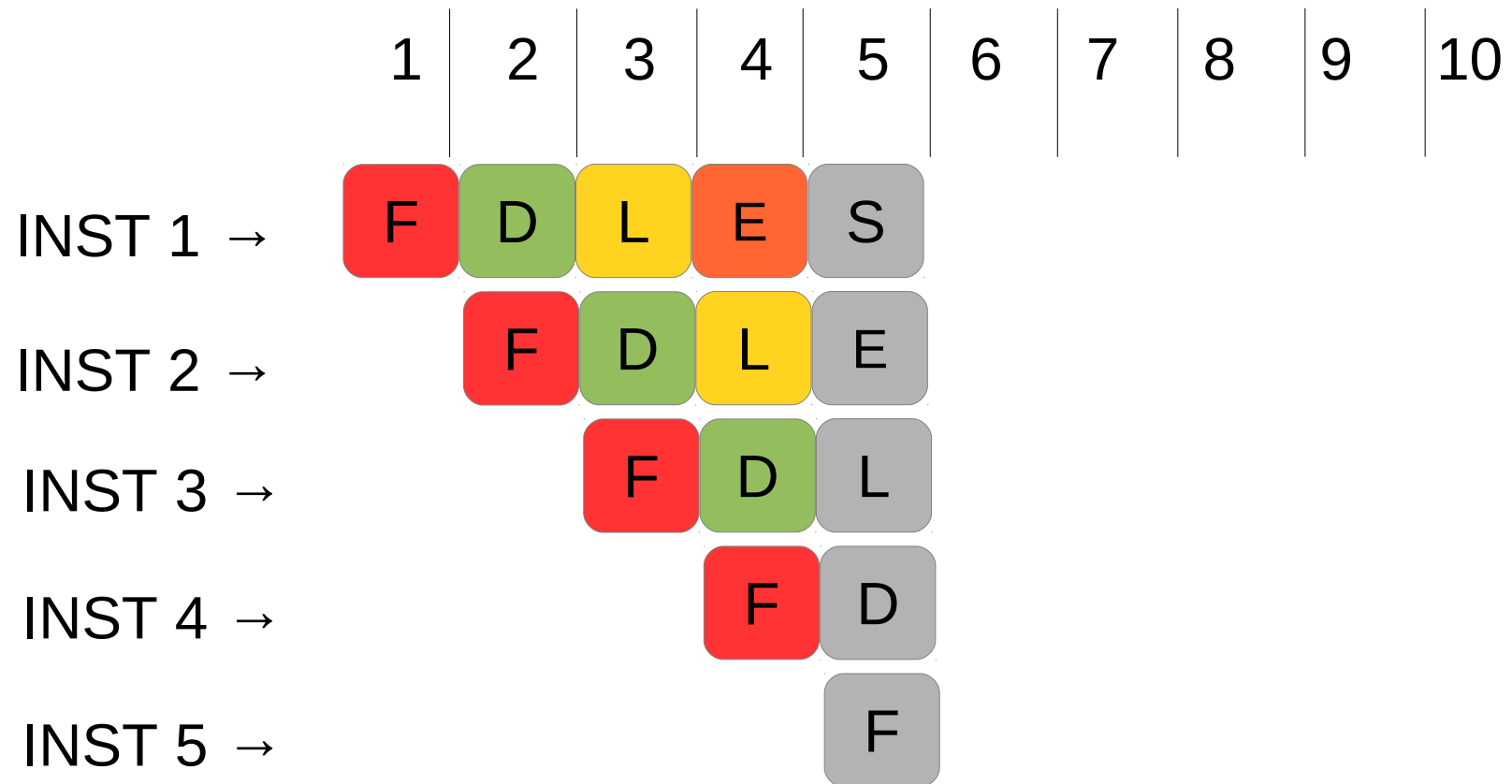
PIPELINING



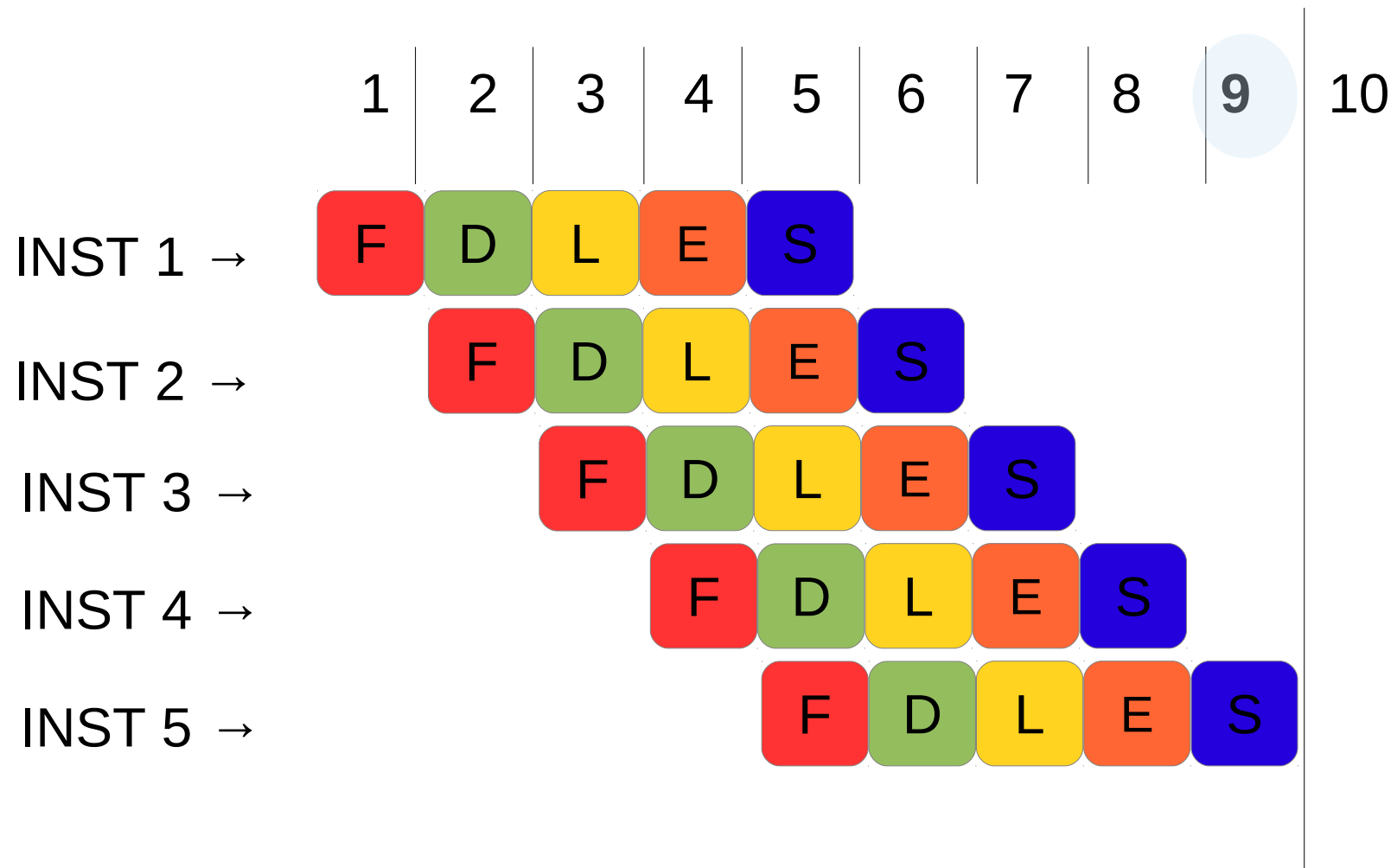
PIPELINING



PIPELINING



PIPELINING



PIPELINING

$z = a * b + c * d$

En realidad:

$z1 = a * b$

$z2 = c * d$

$z = z1 + z2$

C1. cargar a en R0

C2. cargar b en R1

C3. $R2 = R0 * R1$

cargar c en R3

C4. cargar d en R4

C5. $R5 = R3 * R4$

Inst. prox. operación

C6. $R6 = R2 + R5$

Inst. prox. operación

C7. almacenar R6 en z

Inst. prox. operación

PIPELINING

$z = a * b + c * d$

En realidad:

$z1 = a * b$

$z2 = c * d$

$z = z1 + z2$

C1. cargar a en R0

C2. cargar b en R1

C3. $R2 = R0 * R1$

~~cargar c en R3~~

C4. cargar d en R4

C5. $R5 = R3 * R4$

Inst. prox. operación

C6. $R6 = R2 + R5$

Inst. prox. operación

C7. almacenar R6 en z

Inst. prox. operación

Dependencia



PIPELINING

$z = a * b + c * d$

En realidad:

$z1 = a * b$

$z2 = c * d$

$z = z1 + z2$

Cuanto ganamos?

1 paso de 8!
+
3 más si la
próxima operación
es independiente

C1. cargar a en R0

C2. cargar b en R1

C3. $R2 = R0 * R1$

cargar c en R3

C4. cargar d en R4

C5. $R5 = R3 * R4$

Inst. prox. operación

C6. $R6 = R2 + R5$

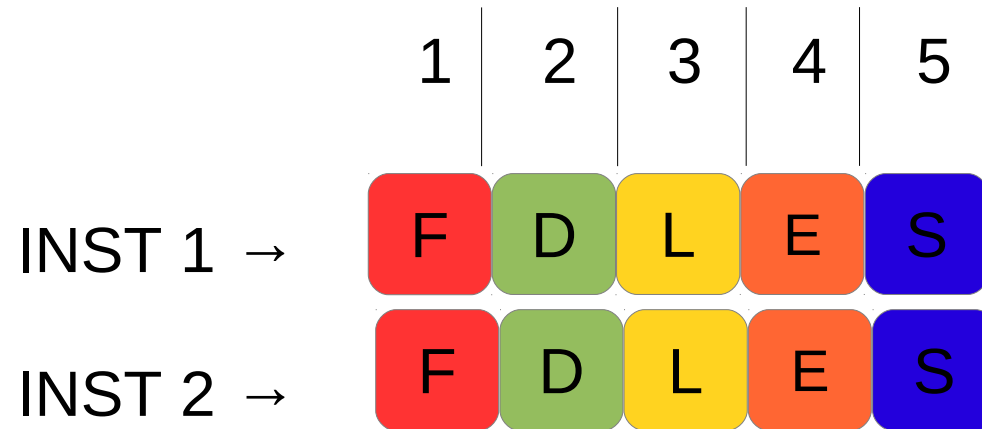
Inst. prox. operación

C7. almacenar R6 en z

Inst. prox. operación

¿Cómo mejorar aún más?

SUPERSCALING



**+ de una instrucción,
inst. independientes**

SUPERSCALING

Ojo! Sigue siendo escalar

Paralelismo a nivel de
instrucción

Muchas instrucciones se
ejecutan simultáneamente,
generalmente combinado con
pipelining.

PIPELINING + SUPERSCALING



PIPELINING + SUPERSCALING

$z = a * b + c * d$

$z1 = a * b$

$z2 = c * d$

$z = z1 + z2$

C1. cargar a en R0

cargar b en R1

C2. $R2 = R0 * R1$

cargar c en R3

cargar d en R4

C3. $R5 = R3 * R4$

Próxima operación

C4. $R6 = R2 + R5$

Próxima operación

C5. almacenar R6 en z

PIPELINING + SUPERSCALING

$z = a * b + c * d$

$z1 = a * b$
 $z2 = c * d$
 $z = z1 + z2$

Pipelining

C1. cargar a en R0

cargar b en R1

C2. $R2 = R0 * R1$

cargar c en R3

cargar d en R4

C3. $R5 = R3 * R4$

Próxima operación

C4. $R6 = R2 + R5$

Próxima operación

C5. almacenar R6 en z

PIPELINING + SUPERSCALING

$z = a * b + c * d$

$z1 = a * b$
 $z2 = c * d$
 $z = z1 + z2$

superscaling



C1. cargar a en R0
cargar b en R1

C2. $R2 = R0 * R1$
cargar c en R3
cargar d en R4

C3. $R5 = R3 * R4$
Próxima operación

C4. $R6 = R2 + R5$
Próxima operación

C5. almacenar R6 en z

PIPELINING + SUPERSCALING

$z = a * b + c * d$

$z1 = a * b$

$z2 = c * d$

$z = z1 + z2$

C1. cargar a en R0

cargar b en R1

C2. $R2 = R0 * R1$

cargar c en R3

cargar d en R4

C3. $R5 = R3 * R4$

Próxima operación

C4. $R6 = R2 + R5$

Próxima operación

C5. almacenar R6 en z

Cuanto ganamos?

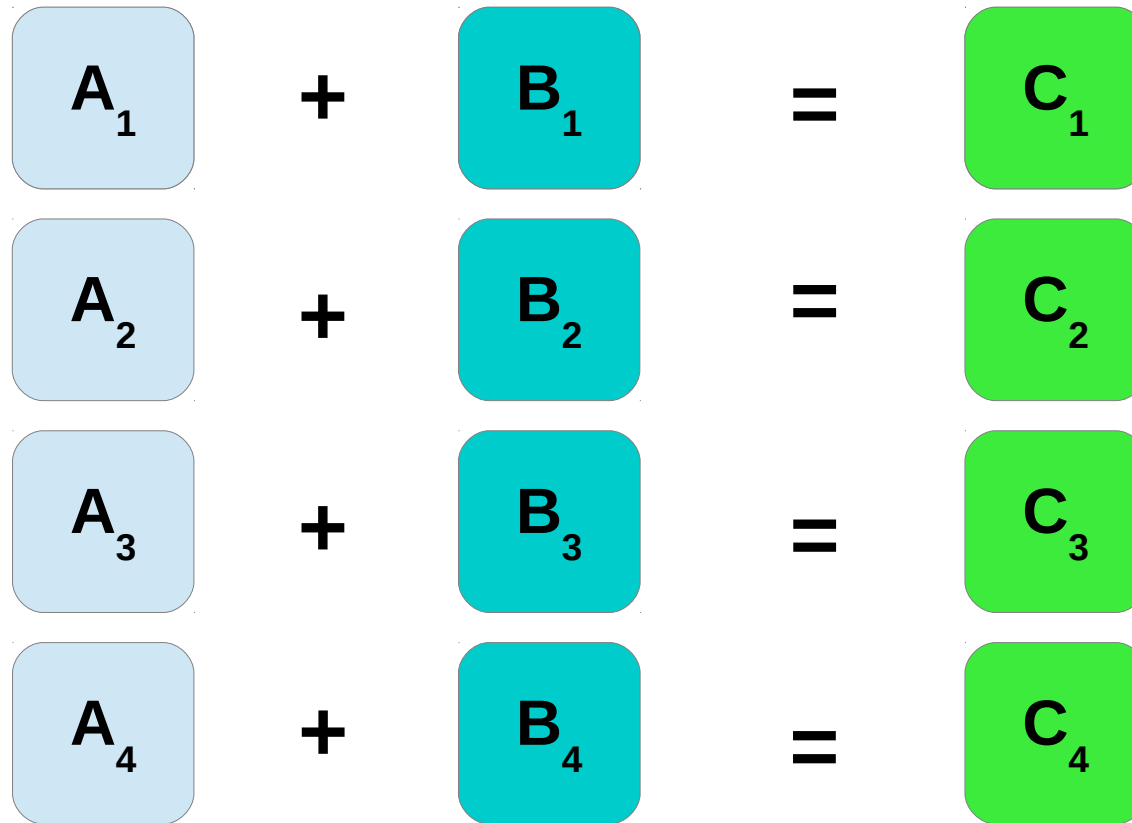
3 pasos de 8!

+

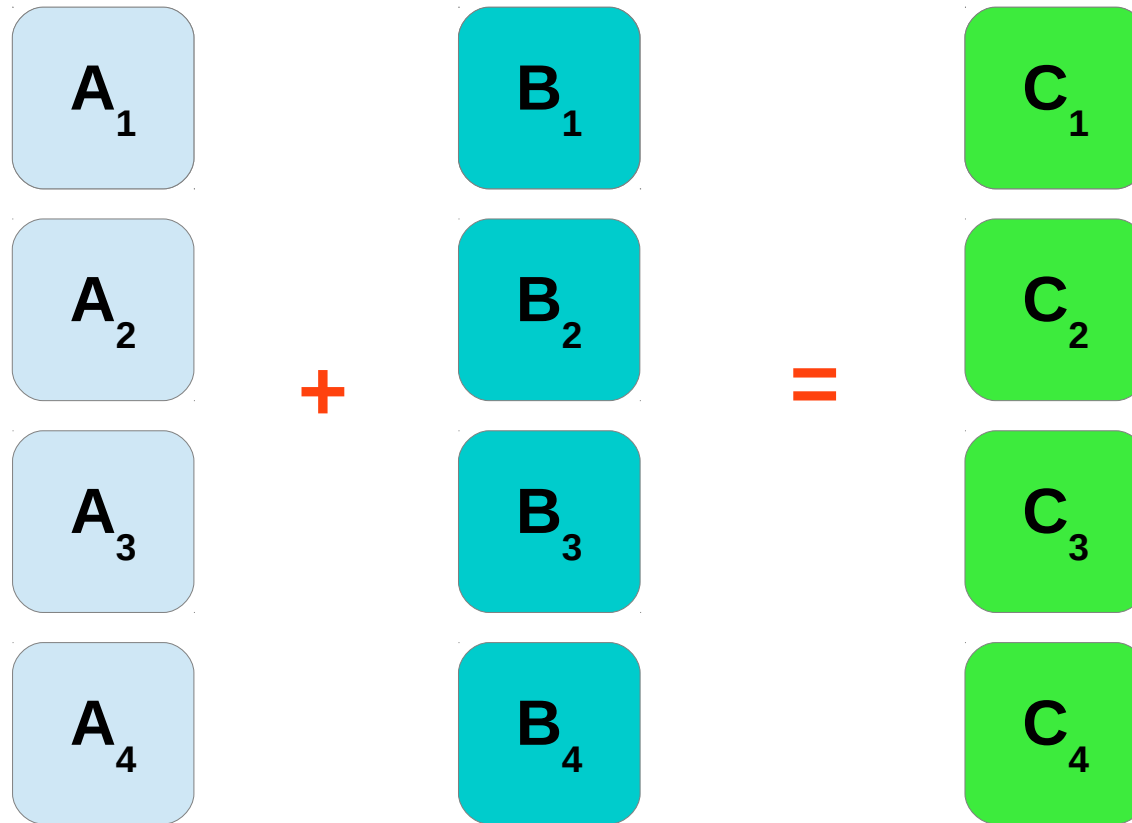
3 si la próxima
operación es
independiente

Hasta aquí todas las operaciones
son escalares !

OPERACIONES ESCALARES



OPERACIONES VECTORIALES



registros vectoriales!

CPU VECTORIAL

- Capaz de ejecutar operaciones matemáticas sobre múltiples datos de forma simultánea. (registros vectoriales)
- En general es adicional al pipelining superescalar.
- Instrucciones vectoriales especiales (SSE,AVX,etc)

Escalar

vs

Vectorial

```
For (i=0;i<length;i++){  
    z[i]=a[i]+b[i];  
}
```

Super escalar + pipeline

```
For (i=0;i<length;i++){  
    z[i]=a[i]*b[i]+c[i]*d[i]; }
```

- | | |
|---|---|
| 1. a[0] en R0
b[0] en R1 | 4. R6=R2+R5
c[1] en R3
d[1] en R4 |
| 2. R2=R0*R1
c[0] en R3
D[0] en R4 | 5. R6 en z[0]
R2=R0*R1
R5=R3*R4 |
| 3. R5=R3*R4
a[1] en R0
b[1] en R1 | a[2] en R0
b[2] en R1 |

Repetir pasos 4-5 según índice

Vectorial

```
For (i=0;i<length;i++){  
    z[i]=a[i]*b[i]+c[i]*d[i]; }
```



Los registros
vectoriales pueden
almacenar y operar
en paralelo

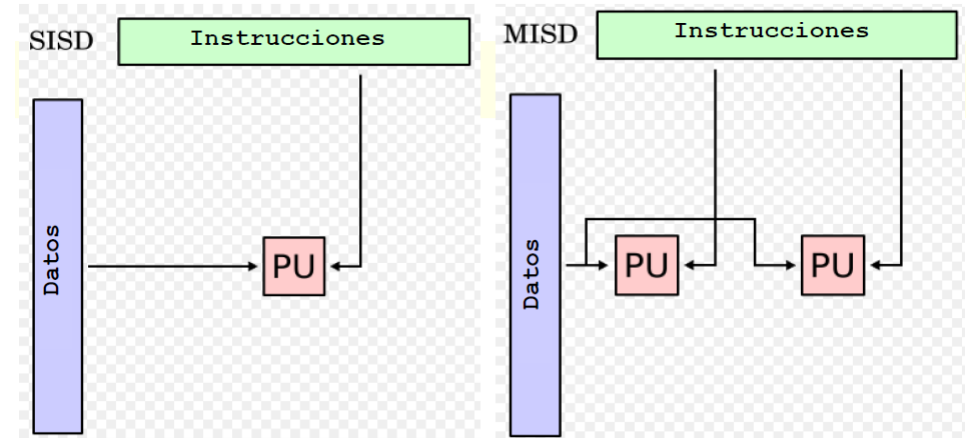
```
For (i=0;i<length;i+=2){  
    en { z[i]=a[i]*b[i]+c[i]*d[i];  
simultaneo { z[i+1]=a[i+1]*b[i+1]+c[i+1]*d[i+1];  
            }  
}
```

CLASIFICACIÓN DE ARQUITECTURAS

		Instrucciones	
		SI	MI
Datos	SD	SISD	(MISD)
	MD	SIMD	MIMD

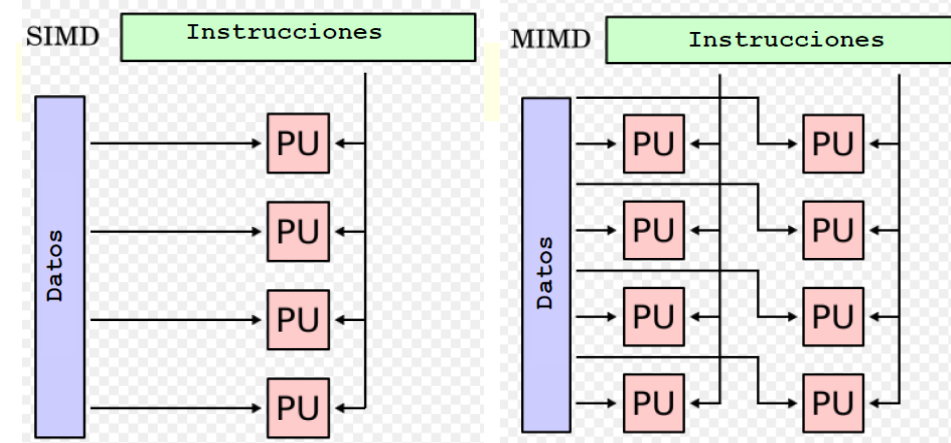
Taxonomía de Flynn (1966)

S=single, M=multi, I=Instrucción, D=Datos



Single Instruction Single Data

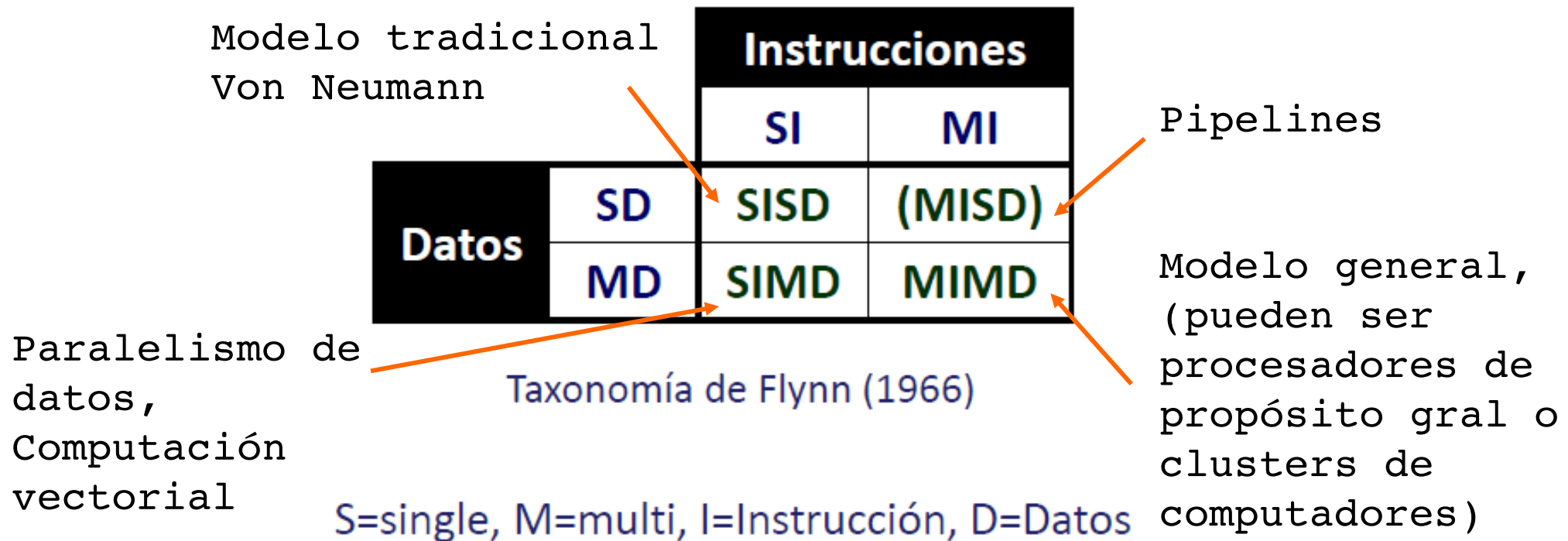
Multiple Instruction Single Data



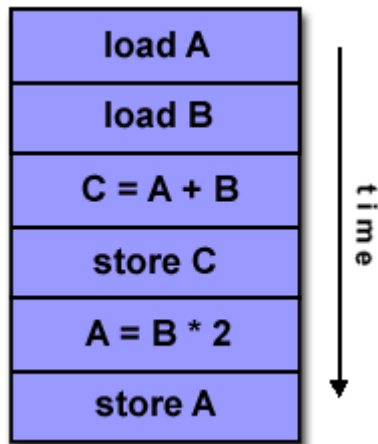
Single Instruction Multiple Data

Multiple Instruction Multiple Data

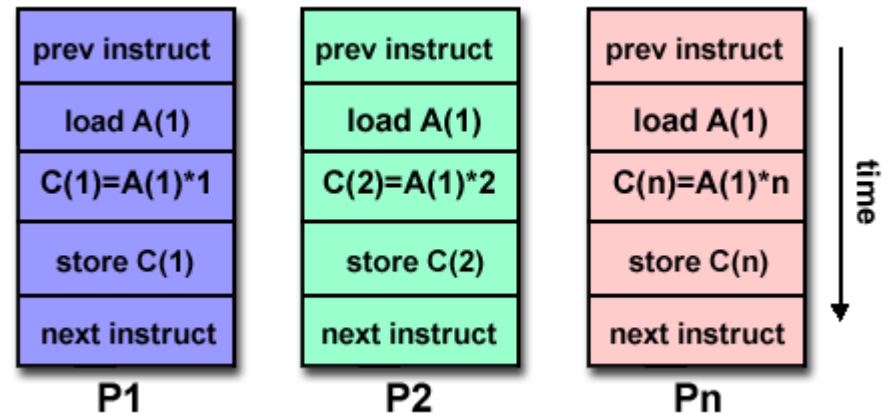
CLASIFICACIÓN DE ARQUITECTURAS



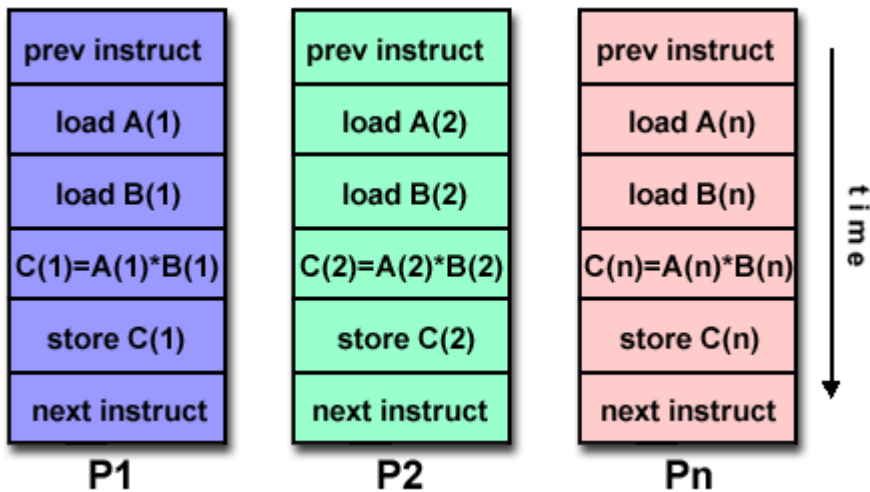
CLASIFICACIÓN DE ARQUITECTURAS



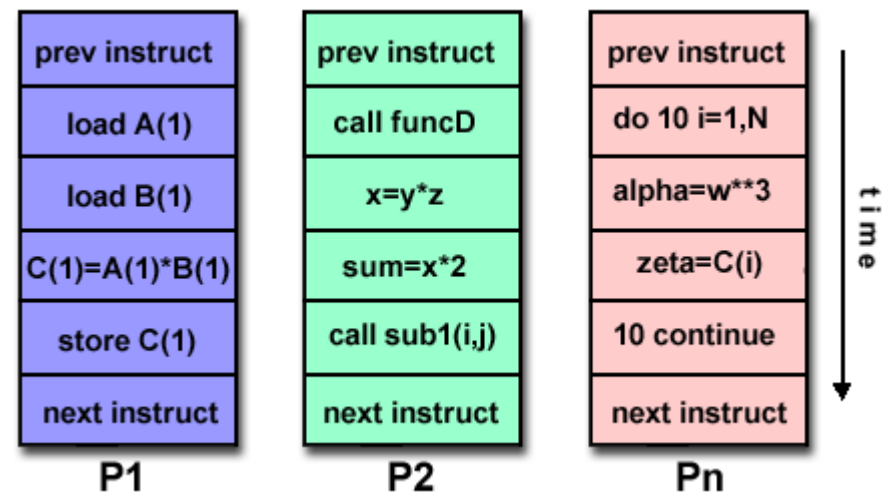
SISD



MISD



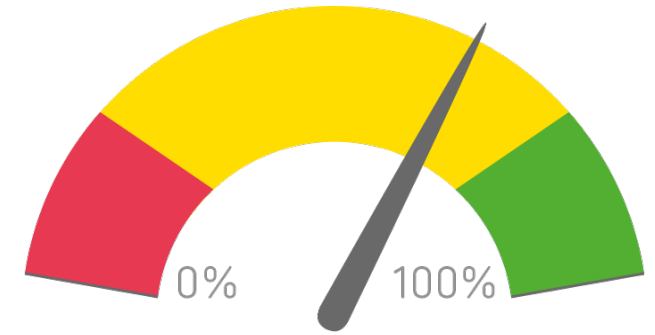
SIMD



MIMD

PERFORMANCE

¿Cómo medir?



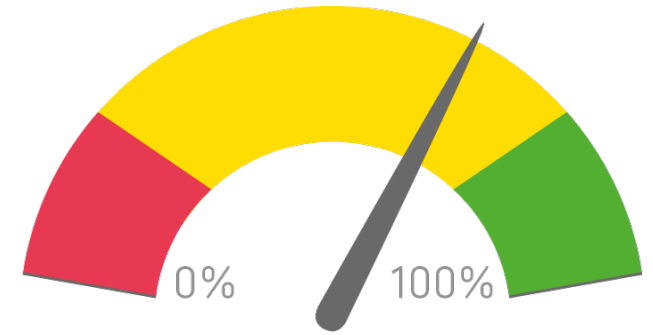
Unidad de medida \rightarrow FLOP/s

Peak Performance (teórico):

Estimación del desempeño de la CPU cuando trabaja a máxima velocidad

PERFORMANCE

¿Cómo medir?



Unidad de medida \rightarrow FLOP/s

Benchmark Performance:

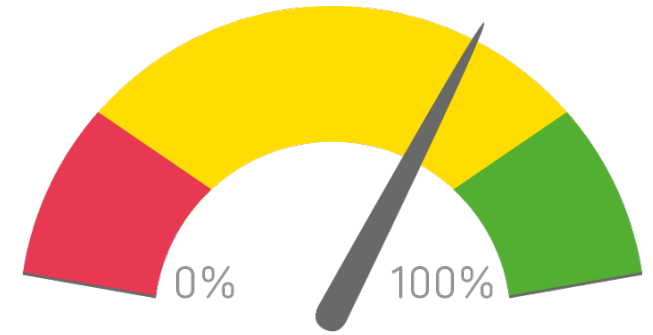
Se utilizan herramientas específicas para medir el pico de performance “real”.

Un ejemplo es Linpack, que es un paquete que resuelve un sistema de matrices densas (con valores aleatorios) de doble precisión (64 bits) y mide el rendimiento del sistema.

<http://www.netlib.org/benchmark/hpl/>

PERFORMANCE

¿Cómo medir?



Linpack

8 Duals Intel PIII 550 Mhz (512 Mb) - Myrinet

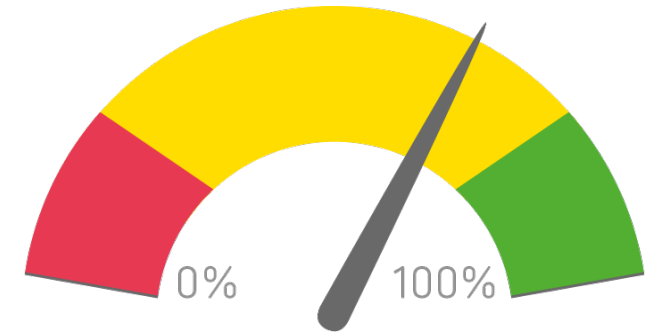
OS	Linux 6.1 RedHat (Kernel 2.2.15)
C compiler	gcc (egcs-2.91.66 egcs-1.1.2 release)
C flags	-fomit-frame-pointer -O3 -funroll-loops
MPI	MPI GM (Version 1.2.3)
BLAS	ATLAS (Version 3.0 beta)
Comments	UTK / ICL - Torc cluster - 09 / 00

Performance (Gflops) w.r.t Problem size on 8- and 16-processors grids.

GRID	2000	5000	8000	10000	15000	20000
2 x 4	1.76	2.32	2.51	2.58	2.72	2.73
4 x 4	2.27	3.94	4.46	4.68	5.00	5.16

PERFORMANCE

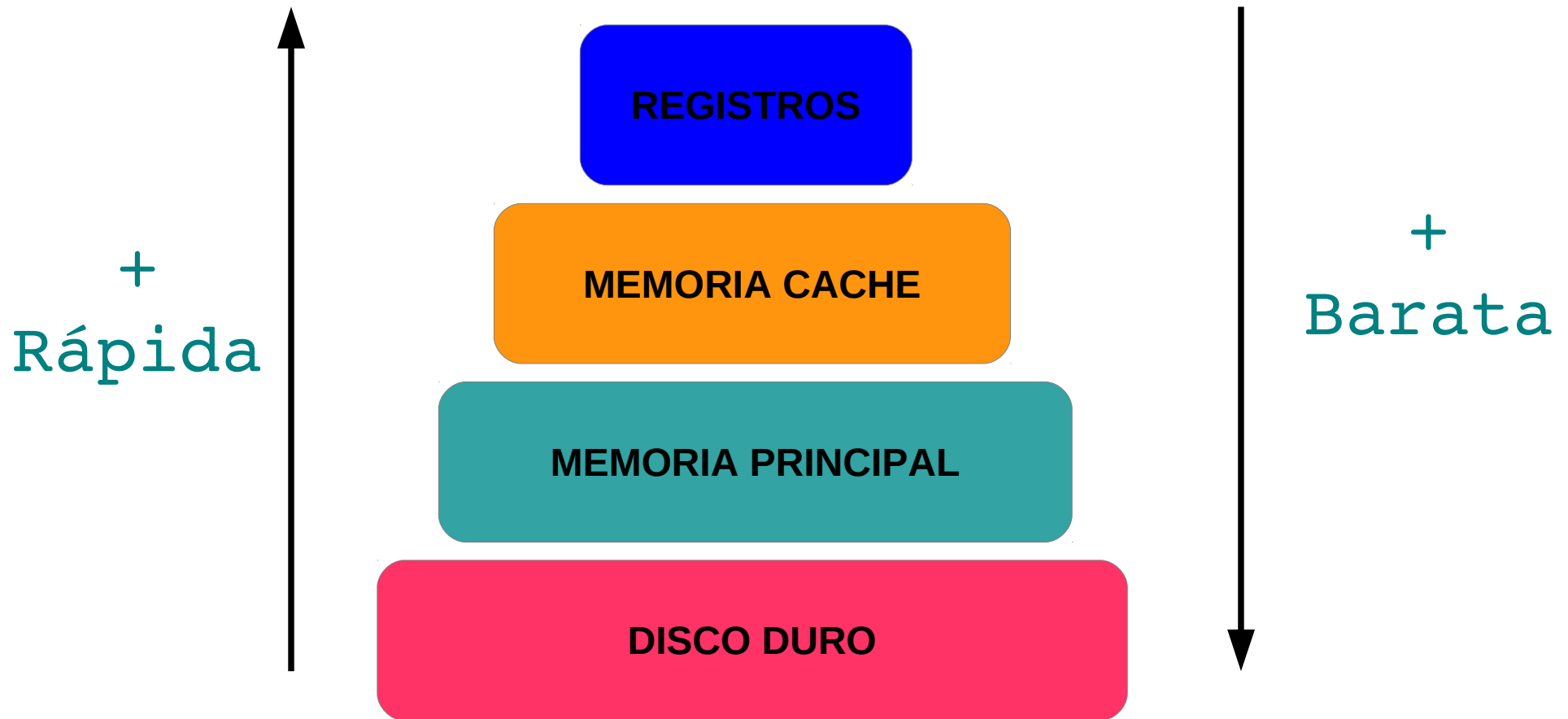
¿Cómo medir?



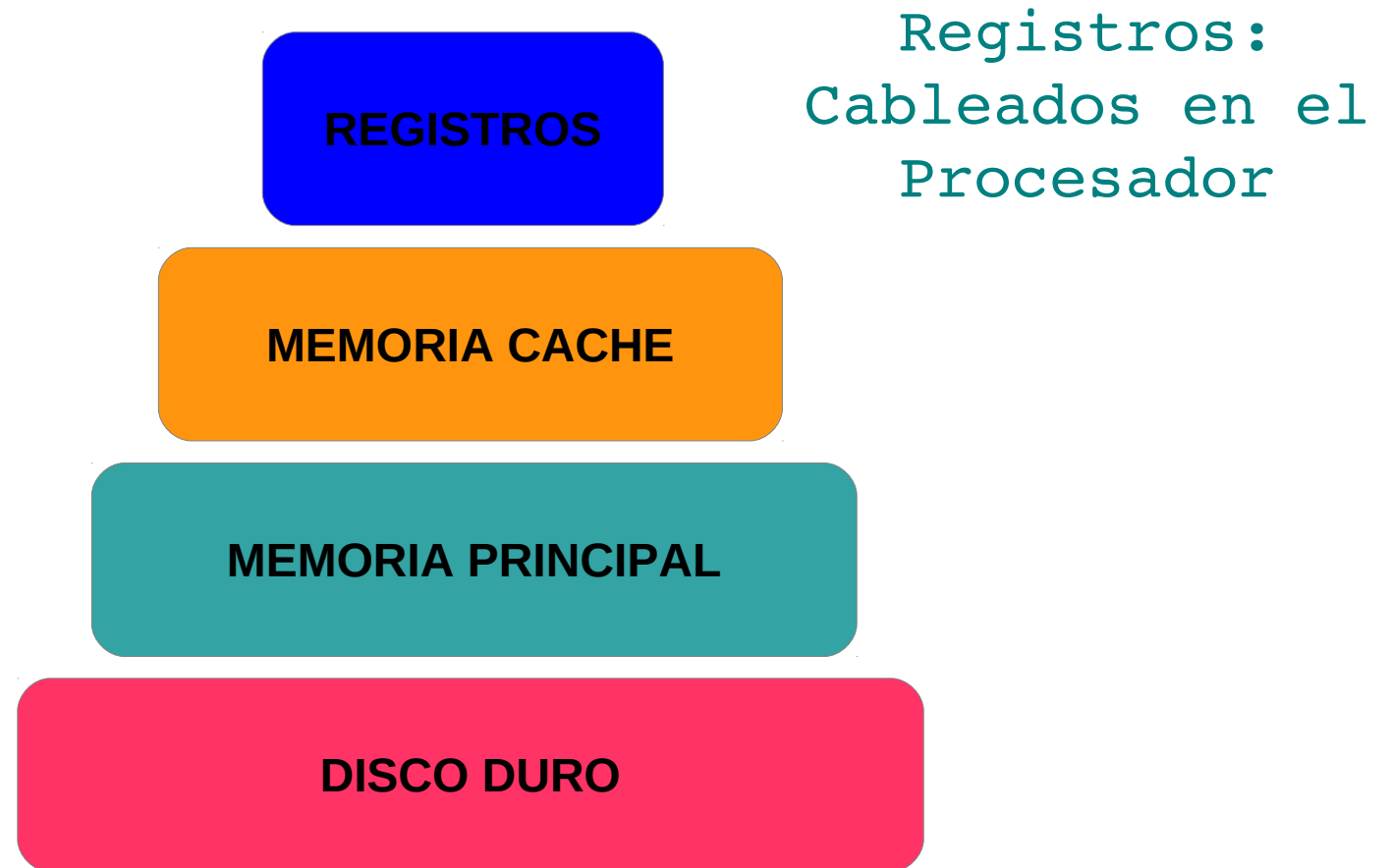
Unidad de medida \rightarrow FLOP/s

Real Performance: medición realizada con el programa que quiero correr.

JERARQUIA DE MEMORIA

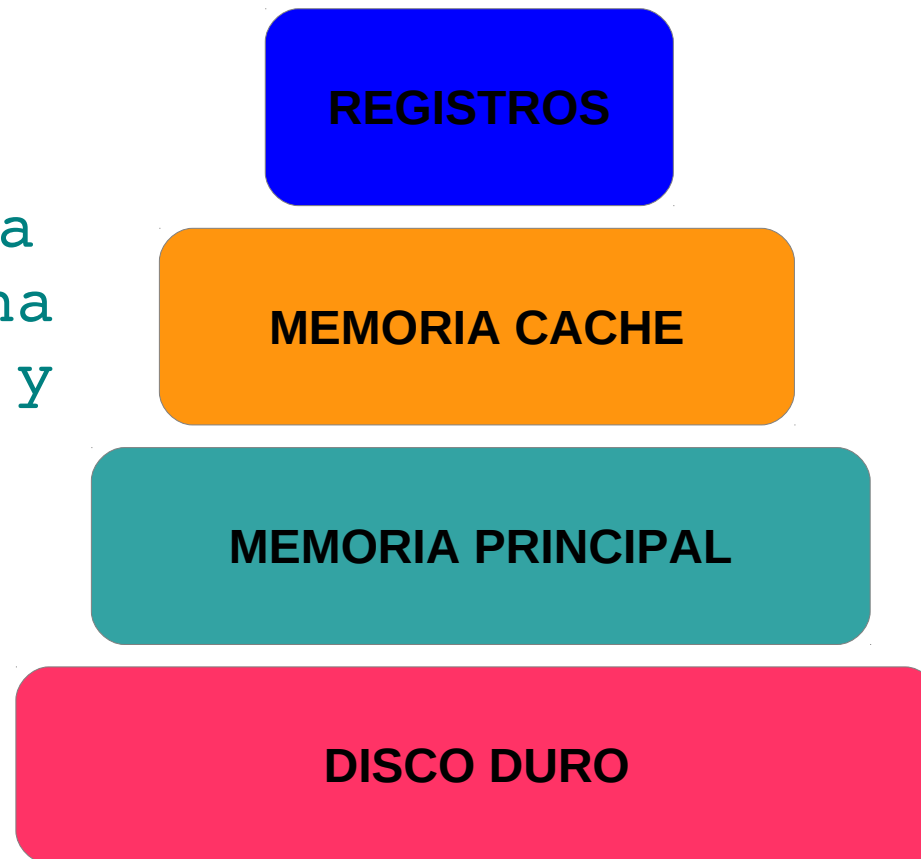


JERARQUIA DE MEMORIA

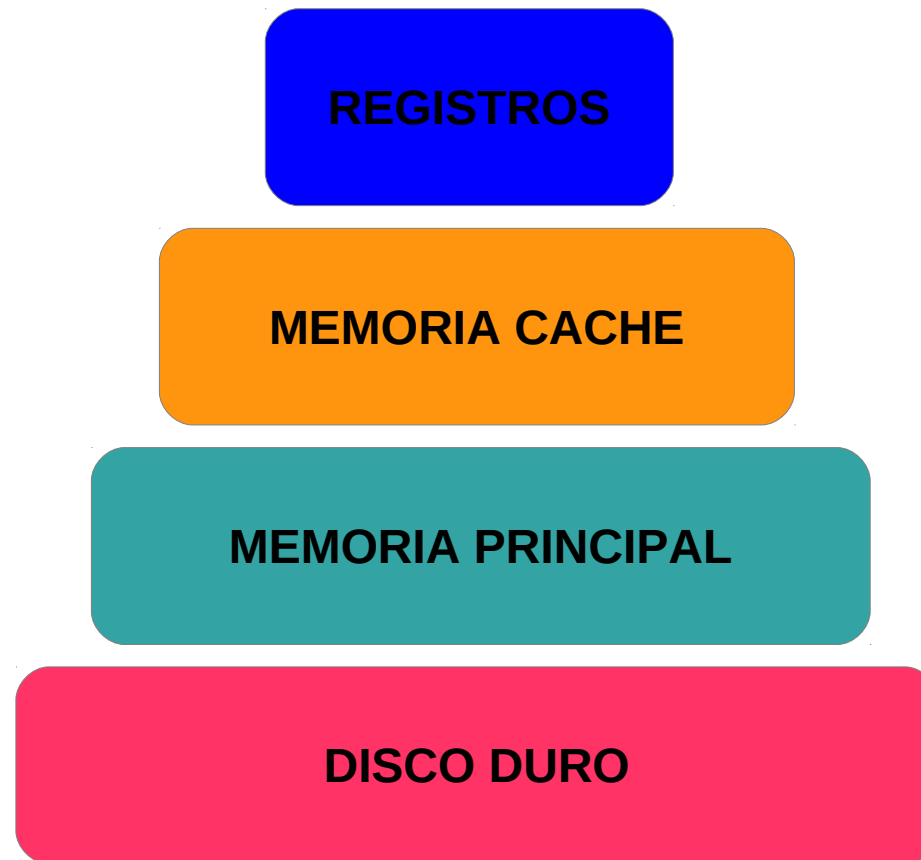


JERARQUIA DE MEMORIA

Cache: Memoria
rápida, cercana
al procesador y
cara

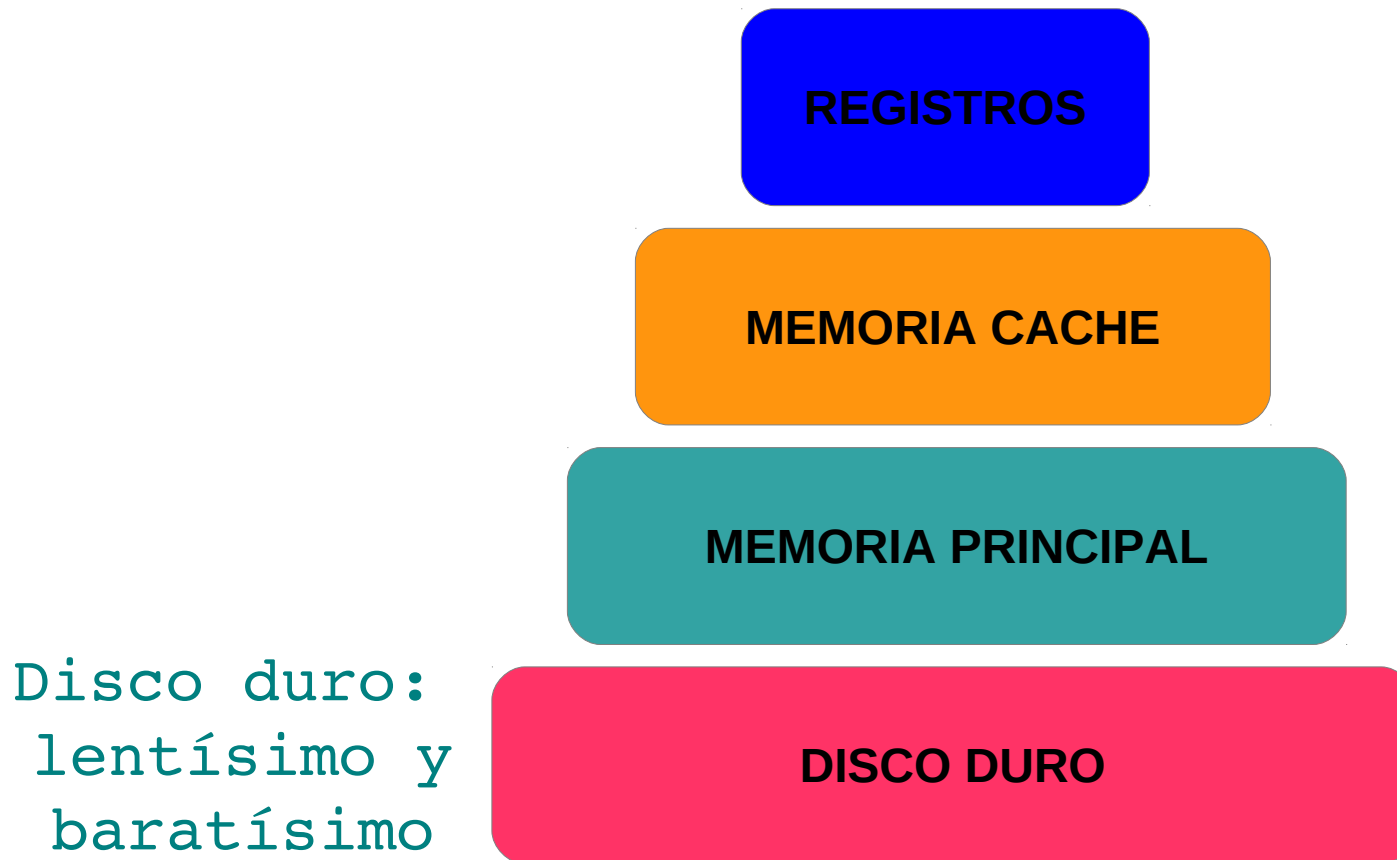


JERARQUIA DE MEMORIA



Memoria ppl:
RAM, lenta y
barata

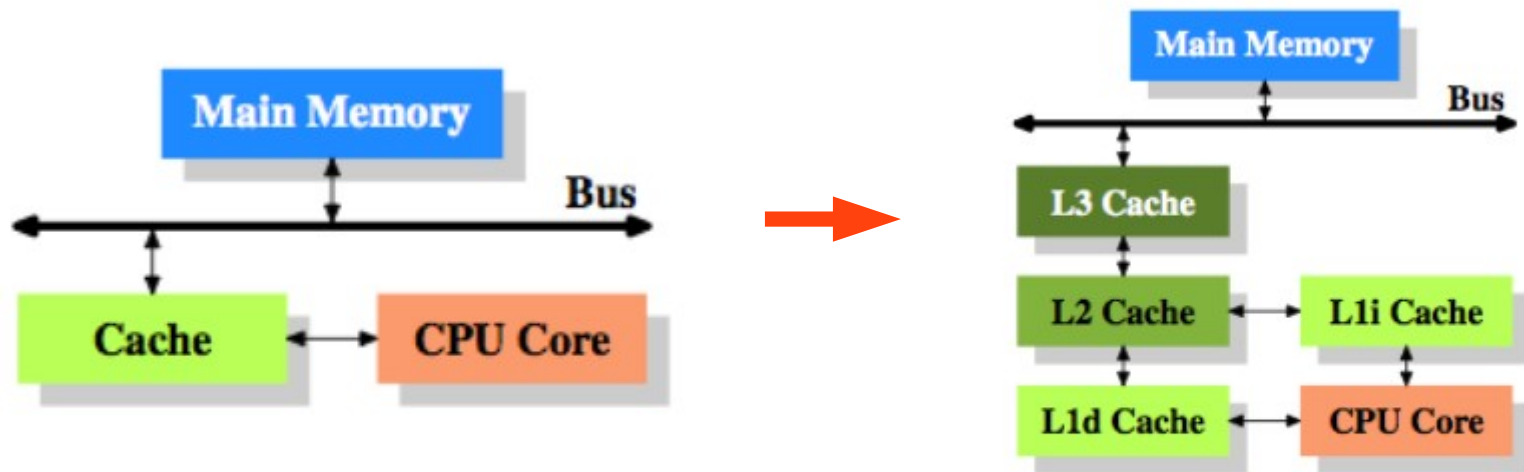
JERARQUIA DE MEMORIA



MEMORIA CACHE


- > costo
- > velocidad
- < capacidad

Los datos se transfieren a cache en bloques de un determinado tamaño → **cache lines**

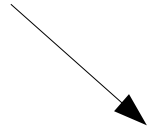


MEMORIA CACHE

Diseñada de acuerdo al principio de localidad → **espacial y temporal**



Por ejemplo,
recorrer un
vector con un
loop



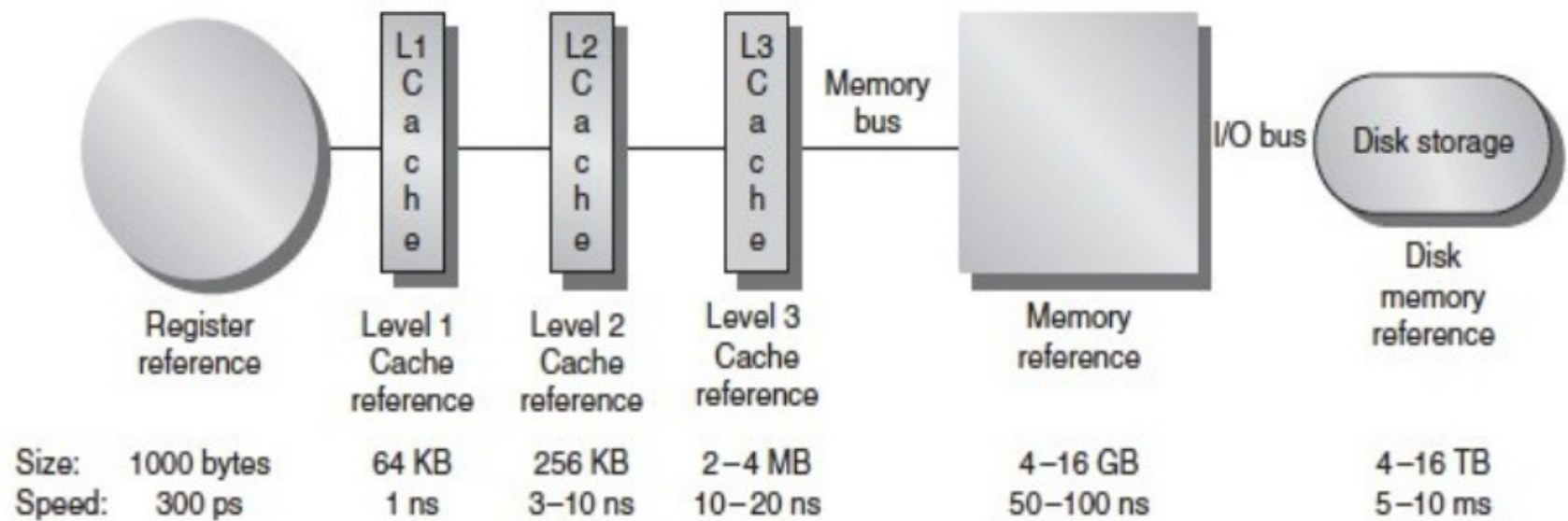
Por ejemplo, un
llamado a una
función dentro
de un loop

MEMORIA CACHE

Diseñada de acuerdo al principio de localidad → **espacial y temporal**

Cada vez que se realiza una operación LOAD/STORE puede ocurrir:
cache miss o cache hit

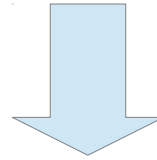
MEMORIA CACHE



De manera más legible

1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 μ s	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

Todo esto para un único procesador!

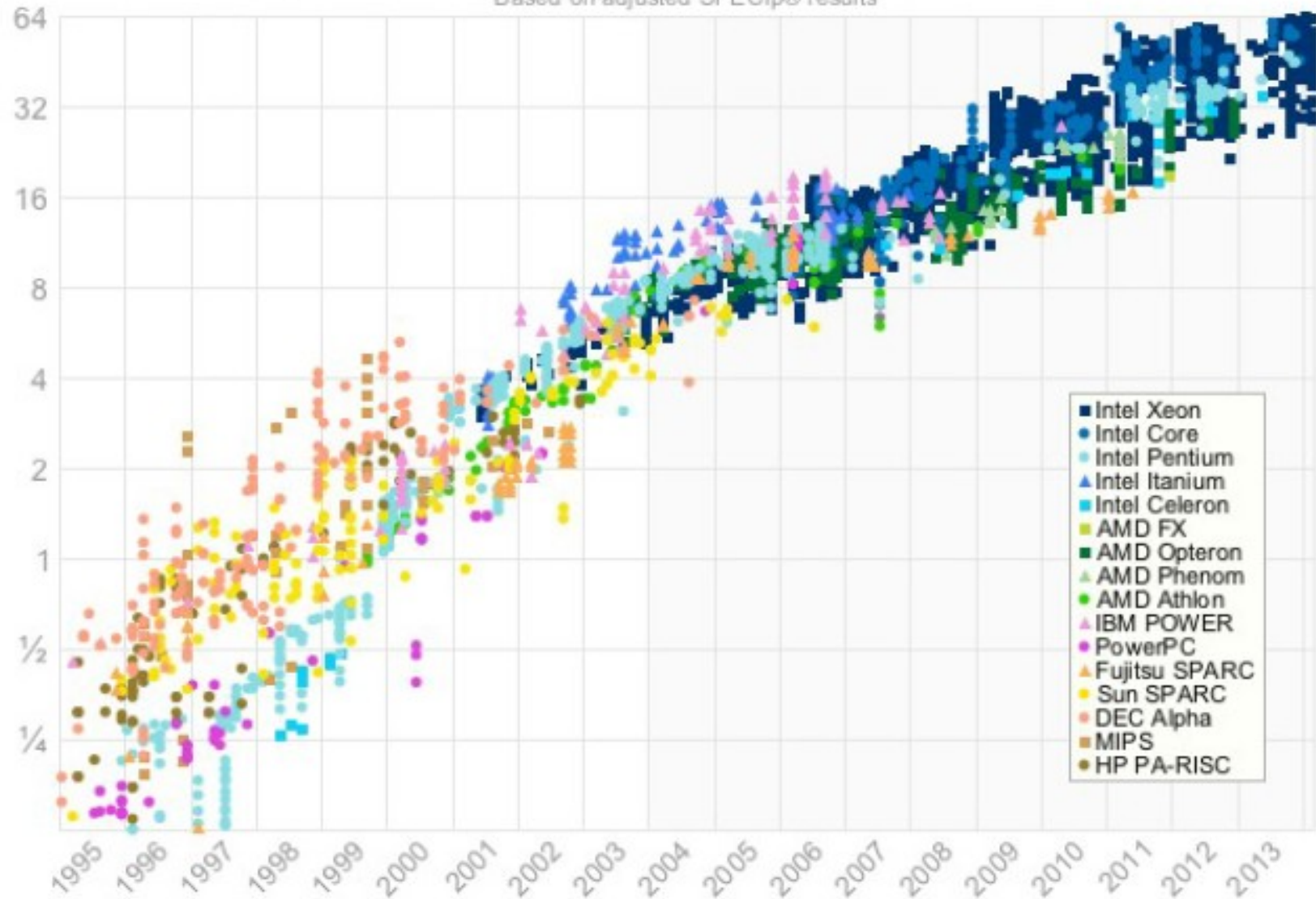


Actualmente → más de un procesador

MULTIPROCESADORES

Single-Threaded Floating-Point Performance

Based on adjusted SPECfp® results

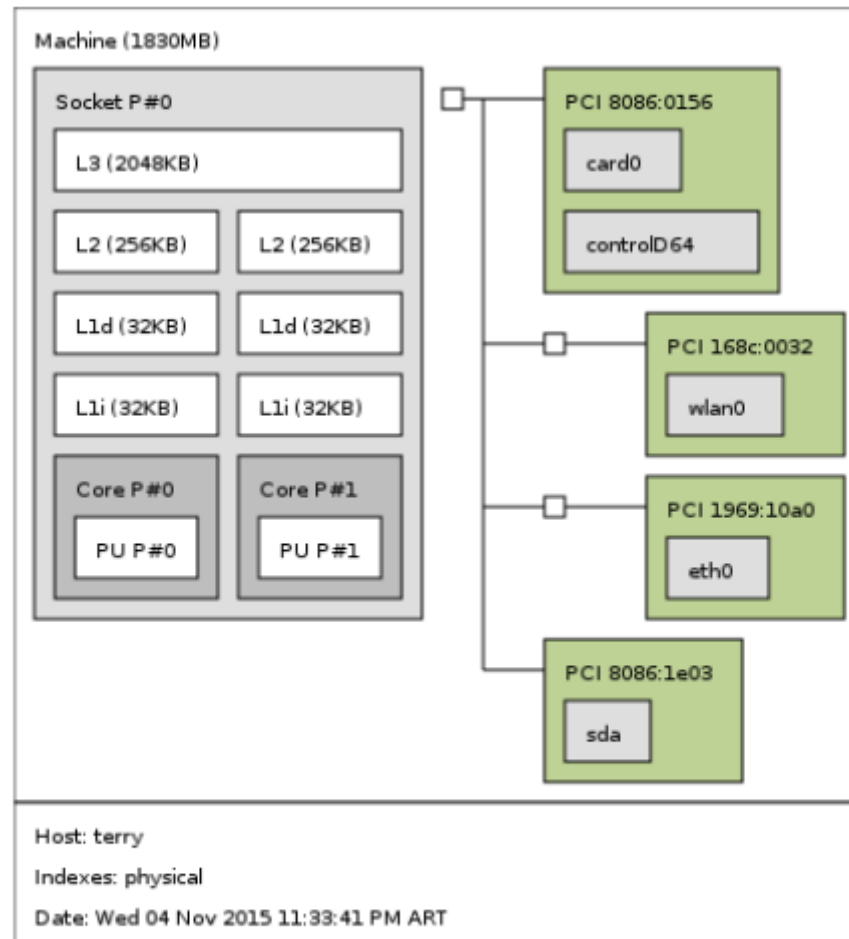


<http://www.spec.org/>

Ley de Moore ????

COMO ES EN REALIDAD

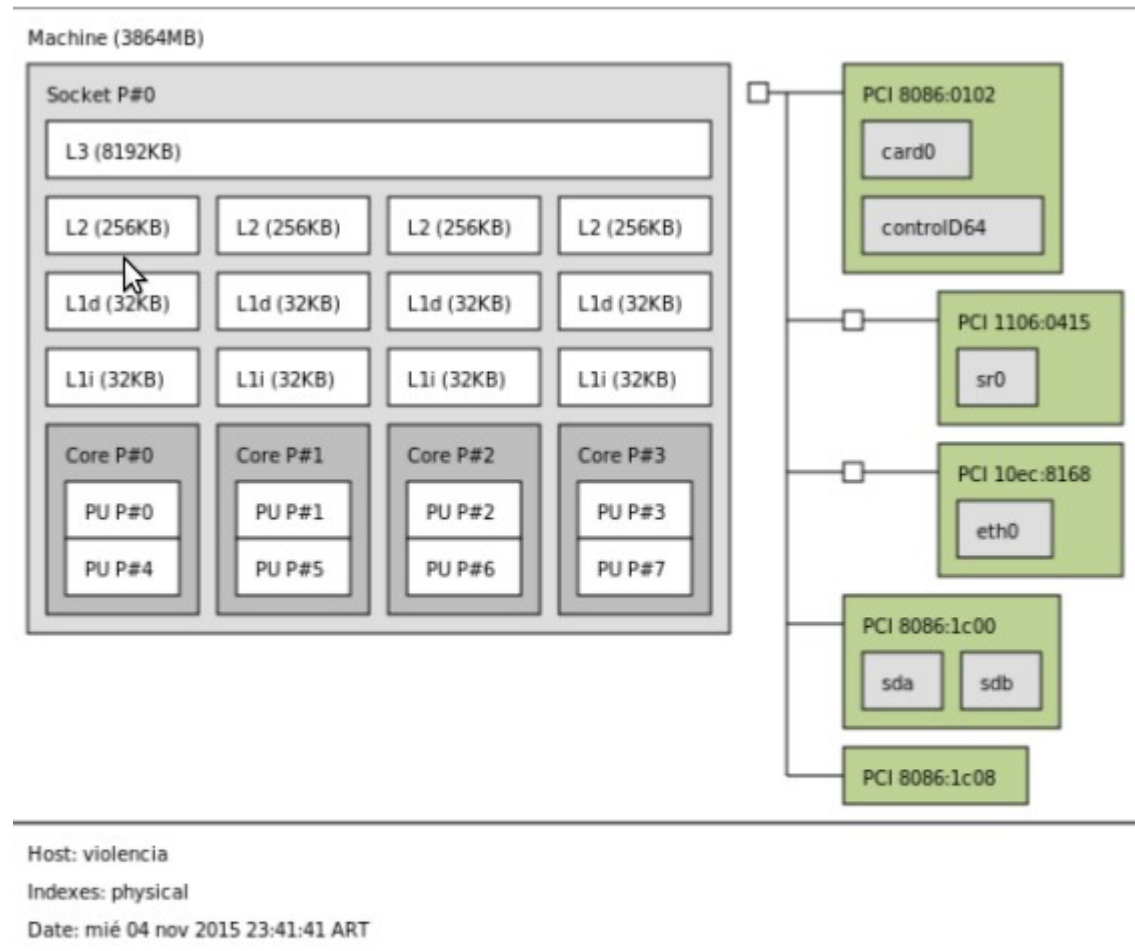
Básicamente
von Neumann



Intel(R) Celeron(R) CPU 1007U

COMO ES EN REALIDAD

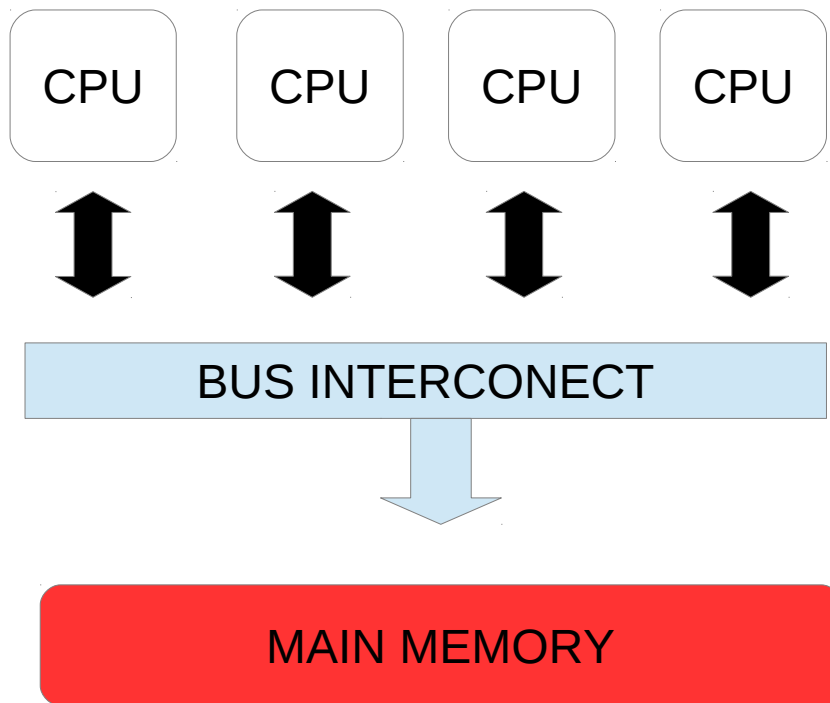
Básicamente
von Neumann



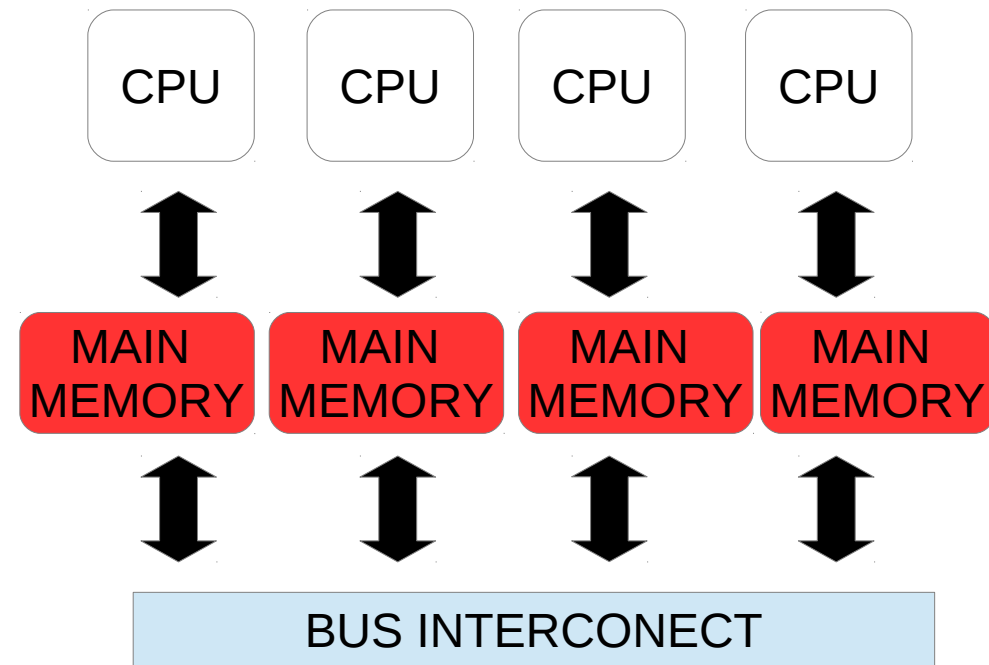
Intel(R) Core(TM) i7-2600 CPU

ARQUITECTURA DE UN CLUSTER

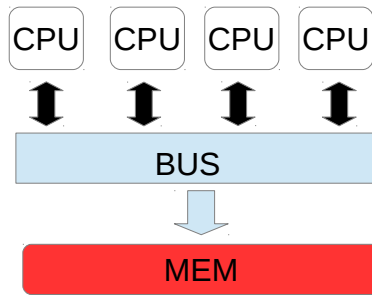
Symmetric MultiProcessors



NonUniform Memory Access



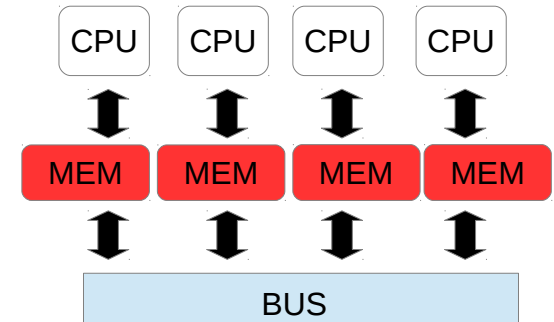
ARQUITECTURA DE UN CLUSTER



MEMORIA COMPARTIDA

Ventajas

- ❖ + fácil al programador.
- ❖ Compartir datos +rápido y directo.

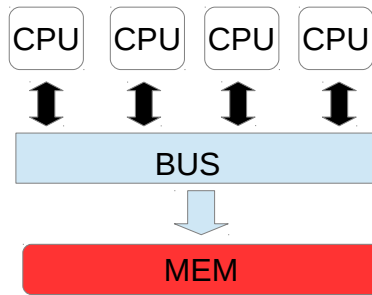


MEMORIA DISTRIBUIDA

Ventajas

- ❖ Escalabilidad de memoria con el num de procesadores.
- ❖ Cada procesador puede **acceder rápidamente a su propia memoria local** sin interferencias y sin overhead.
- ❖ Relación costo-beneficio: se puede usar hardware off-the-shelf con una performance muy razonable.

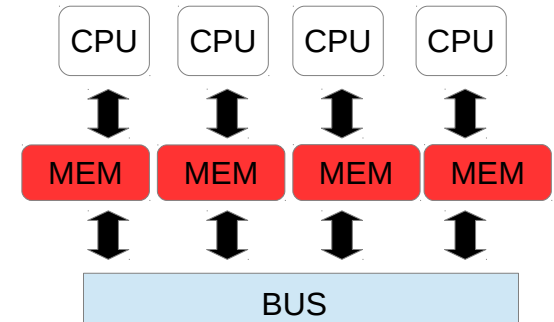
ARQUITECTURA DE UN CLUSTER



MEMORIA COMPARTIDA

Desventajas

- ❖ Escalabilidad pobre: + procesadores, > trafico memoria-procesador.
- ❖ Programador implementar la **sincronización**
- ❖ Costo: + difícil y caro diseñar y producir maquinas con memoria compartida a medida que se aumenta la cantidad de procesadores.

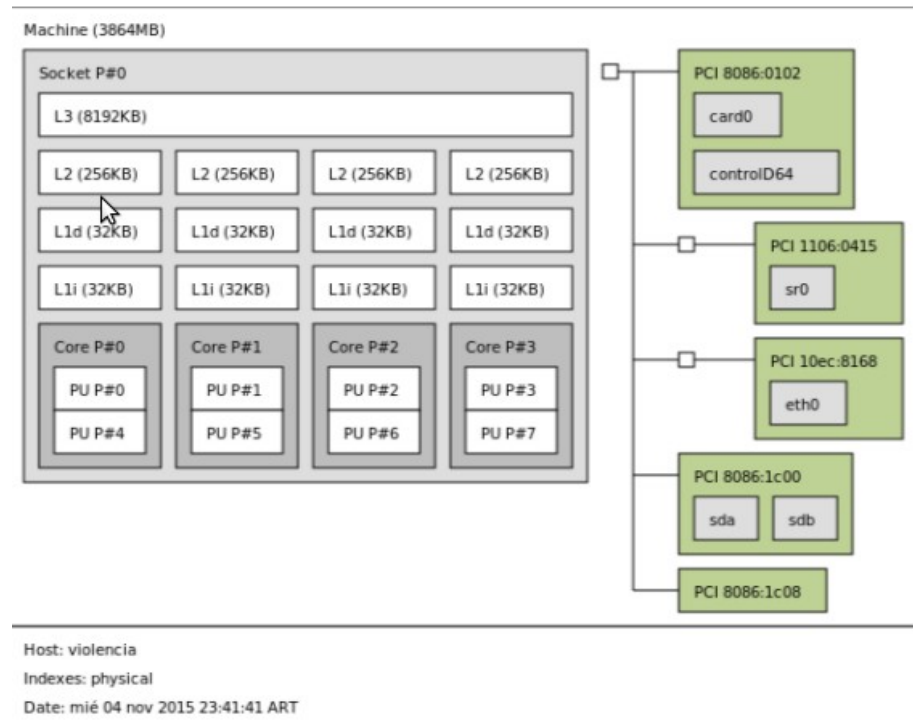
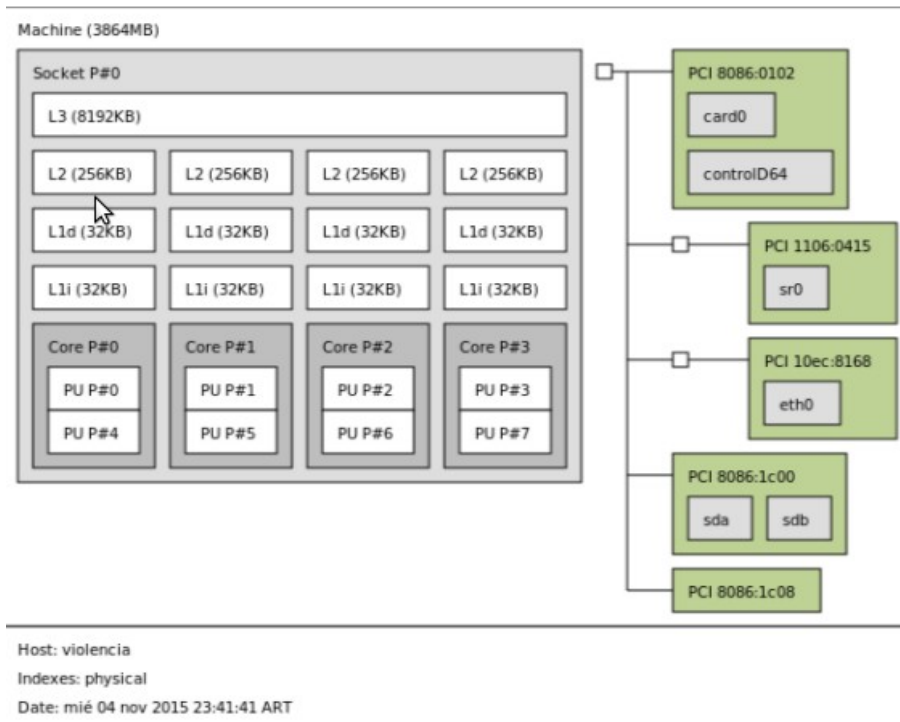


MEMORIA DISTRIBUIDA

Desventajas

- ❖ El programador es responsable de detalles en la comunicación de datos entre procesos.
- ❖ Complicado adaptar código existente basado en memoria compartida.
- ❖ El tiempo de acceso a los datos no es uniformes (y varia mucho!)

ARQUITECTURA DE UN CLUSTER



...

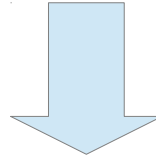
Symmetric Multiprocessors
NonUniform Memory Access

¿COMO IMPACTA EN EL DISEÑO DEL SOFTWARE?

- * Paralelismo Masivo
- * Complejidad Creciente
- * Menos eficiencia para software viejo
- * Poca previsibilidad

HAY QUE PENSAR EN EL HW AL
MOMENTO DE CODIFICAR !

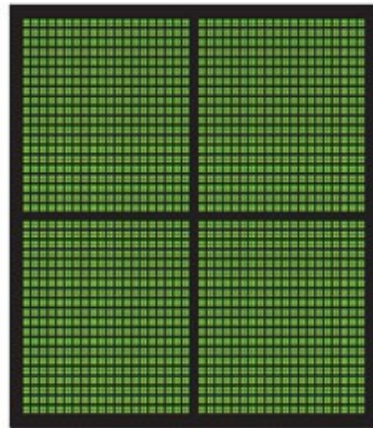
Y ESO NO ES TODO ...



Aceleradores



CPU
MULTIPLE CORES



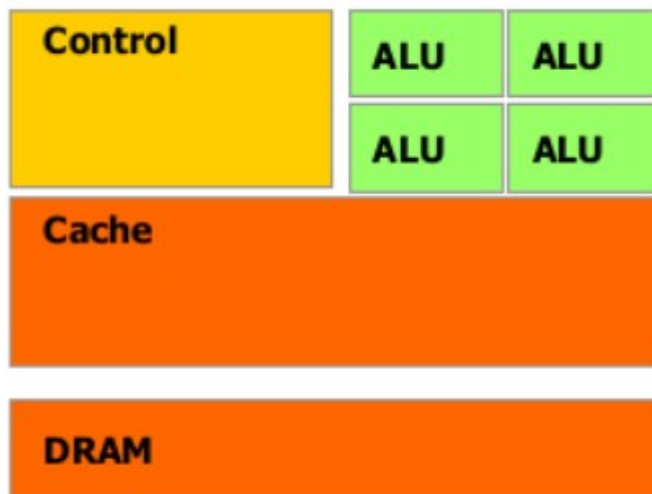
GPU
THOUSANDS OF CORES

unidad de
procesamiento
de gráficos

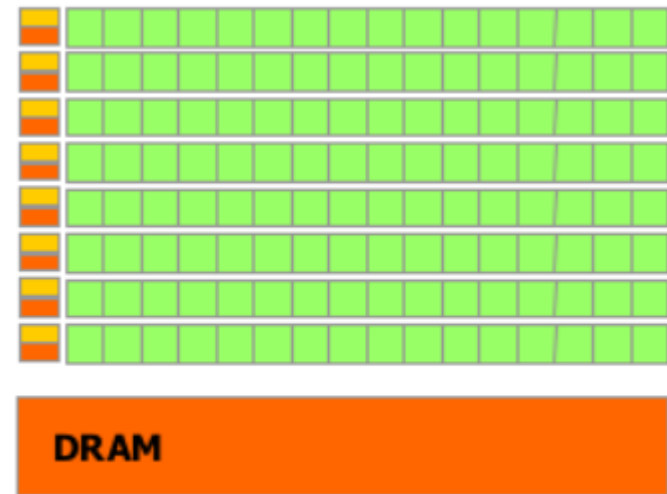


GPU : IDEA GENERAL

Esquemáticamente la diferencia es:



CPU



GPU

GPU : IDEA GENERAL

<ctrl

<cache

>ALUs



Paralelismo masivo

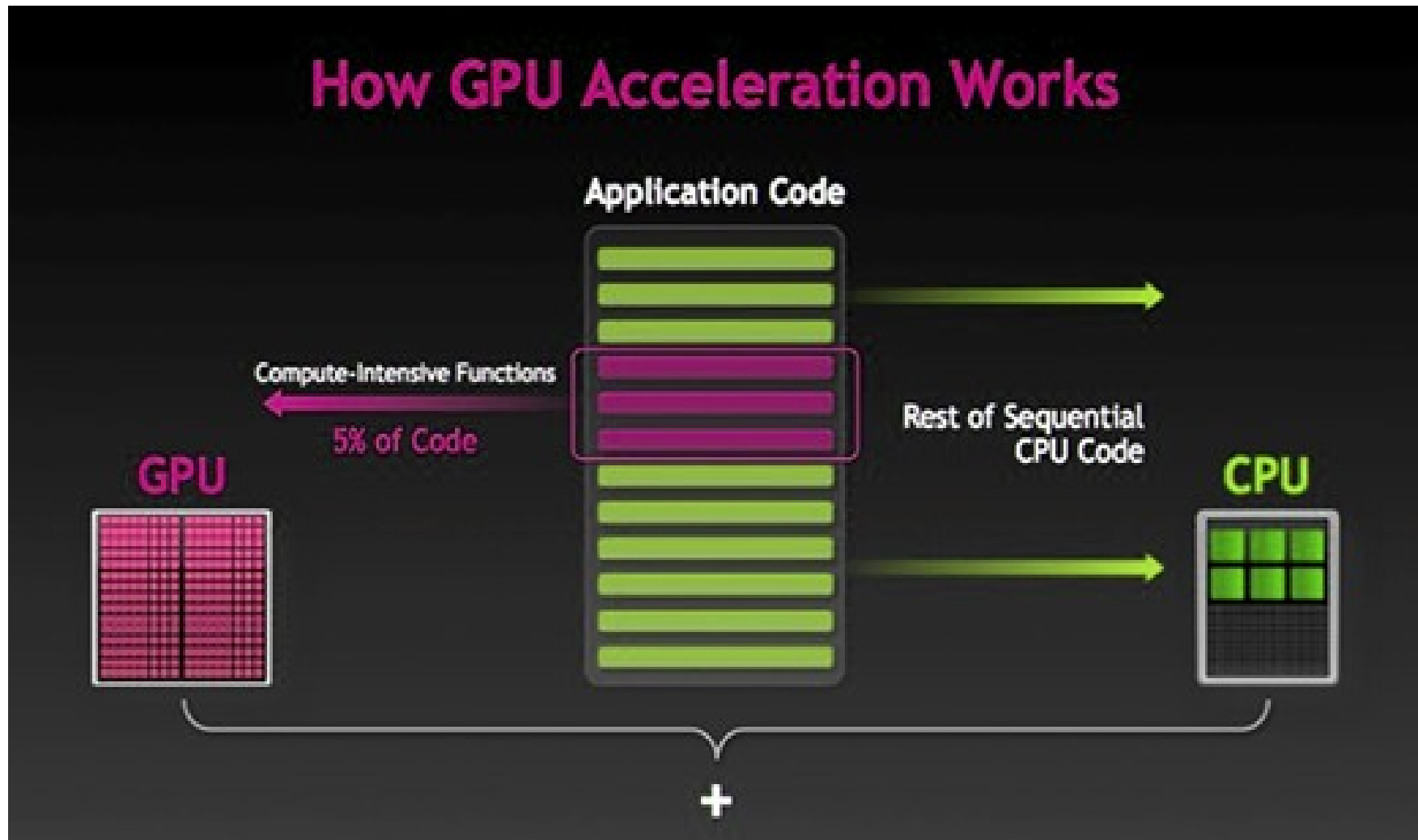


Para alimentar
tantas ALUs

Paralelismo de datos

Suficiente como
para ocultar la
latencia

GPU : IDEA GENERAL



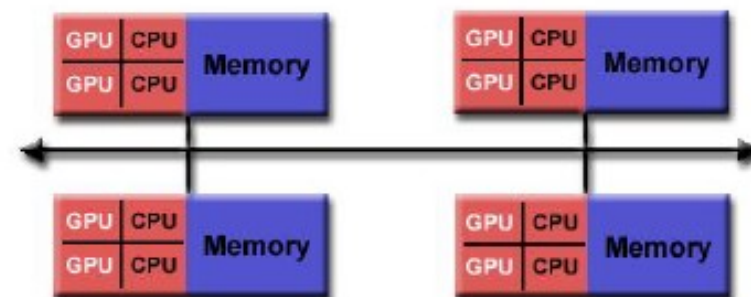
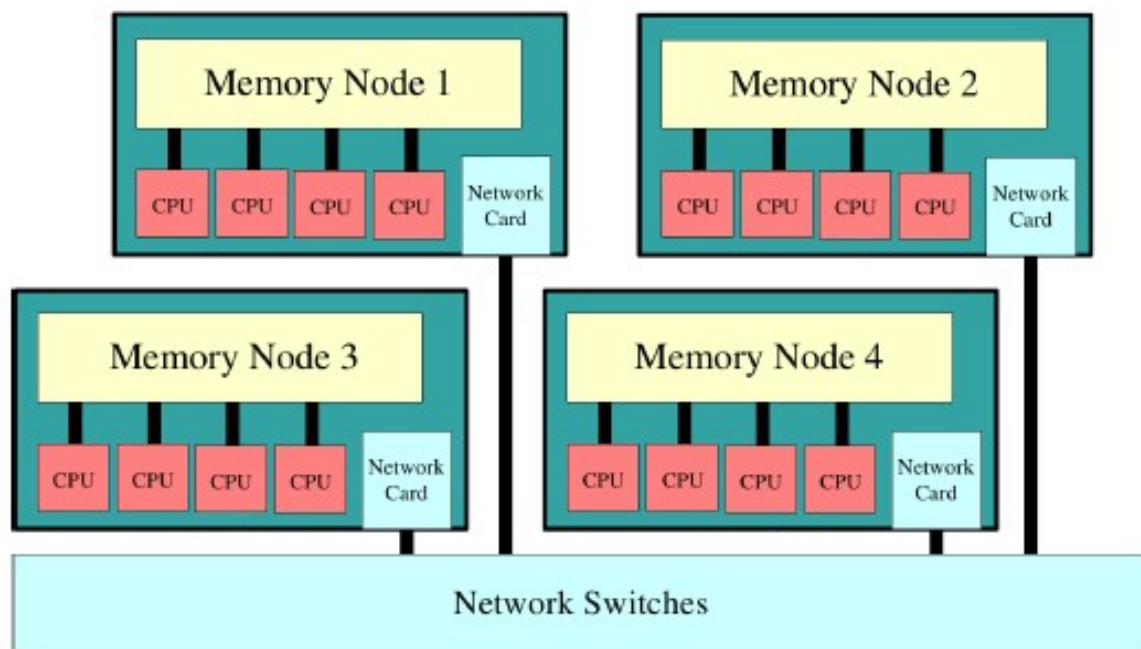
POR QUE ACELERA

- * Diseño muy escalable
- * Mucho ancho de banda
- * Muchos procesadores de baja frecuencia
- * Ideal para el procesamiento de data masiva

NO SIEMPRE ACELERA ...

- * Hay que pasarle la información a la placa
- * Difícil sincronizar los procesadores
- * Ejecución en serie MUY lenta
- * Modelo de memoria demasiado complejo

ARQUITECTURAS HIBRIDAS





ARQUITECTURA DEL COMPUTADOR

Graciela Molina

`gmolina@herrera.unt.edu.ar`
`m.graciela.molina@gmail.com`