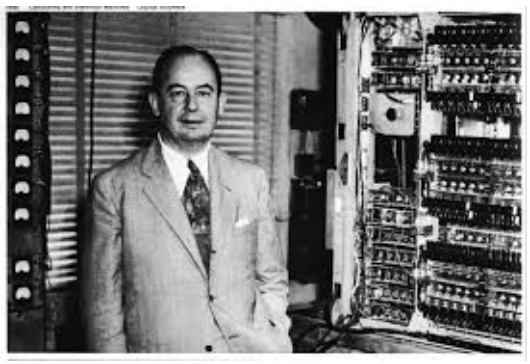
Arquitectura del Computador

M. Graciela Molina

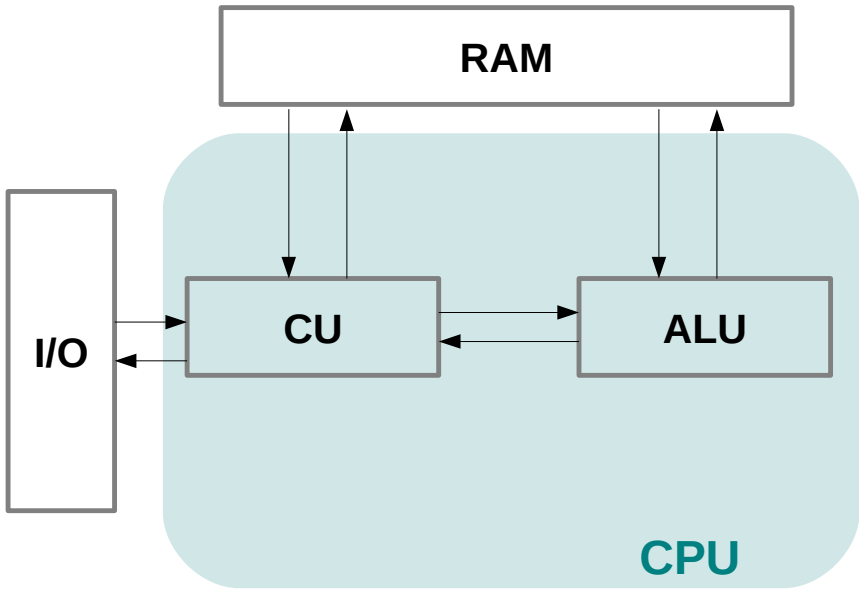
m.graciela.molina@gmail.com

WTPC workshop en técnicas de programación científica

Modelo de von Neumann

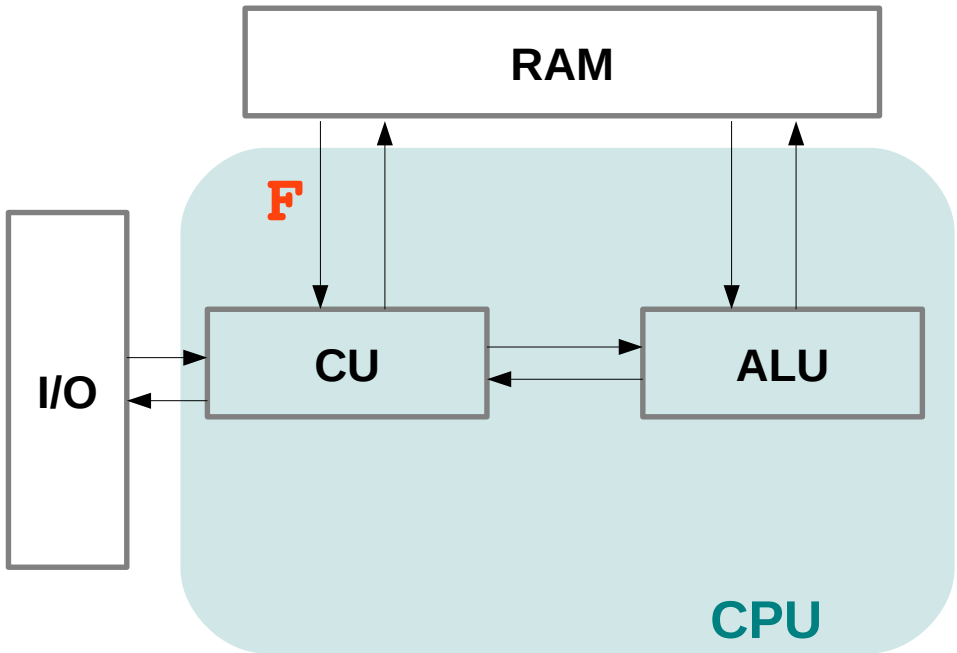


J. Von Neumann frente a la computadora IAS, 1952



Modelo de von Neumann

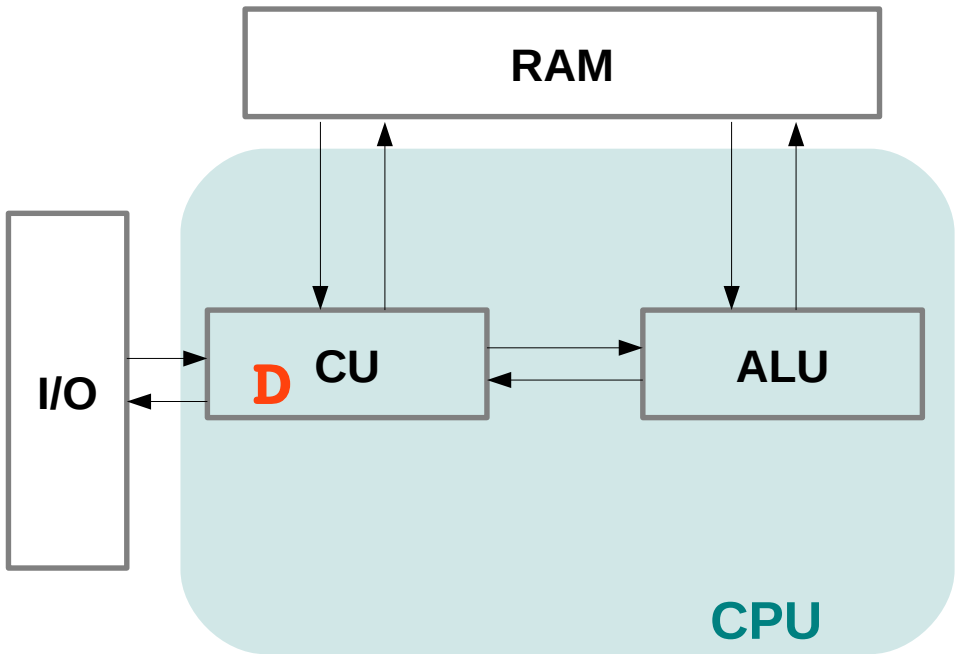
1. FETCH: lee próxima instrucción



0001 1101 0011 0110 1001 ?

Modelo de von Neumann

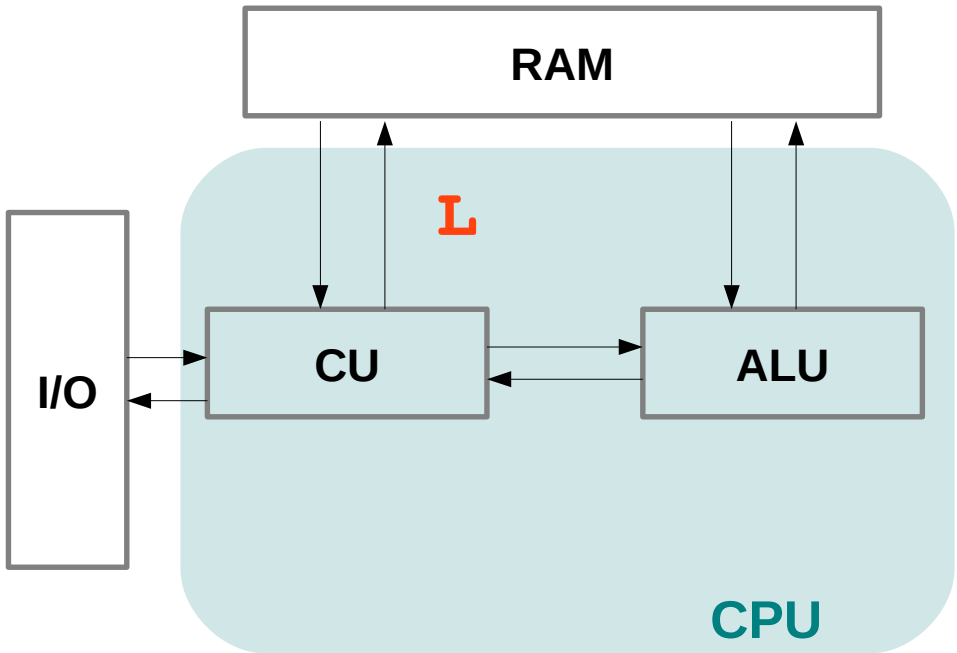
2. DECODE: decodifica la instrucción



add r1, 0xF21E !

Modelo de von Neumann

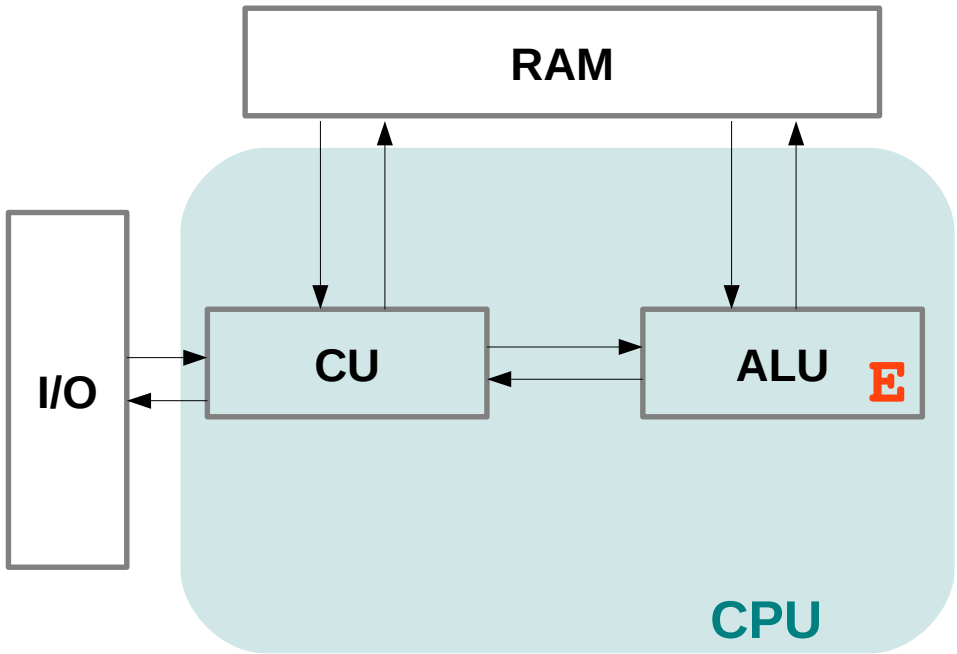
3. LOAD: pide la instrucción



r2 ← 0xF21E

Modelo de von Neumann

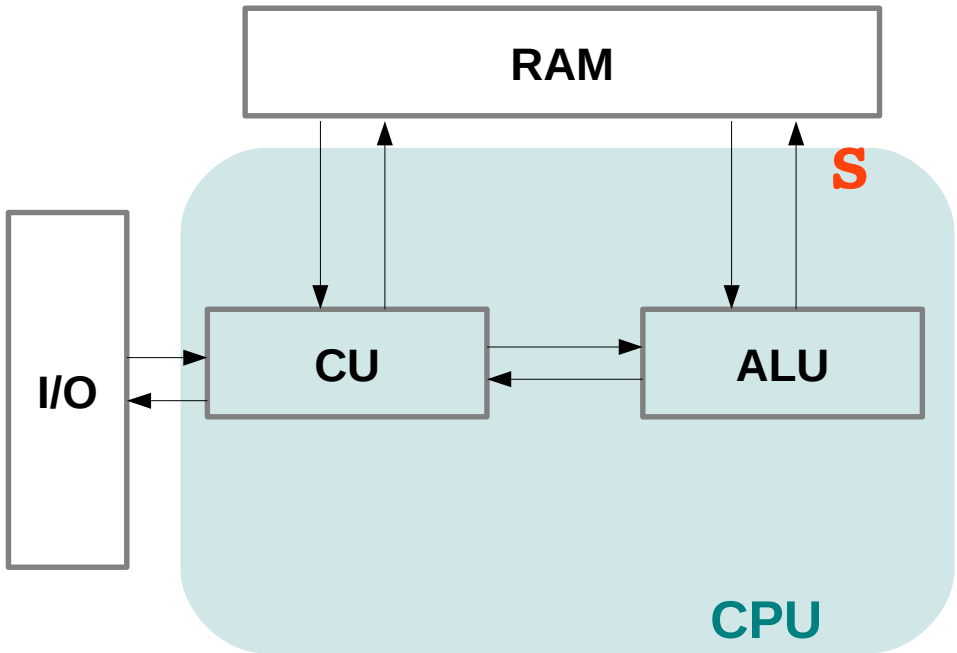
4. EXECUTE: ejecuta la operación



$$r1 \leftarrow r1 + r2$$

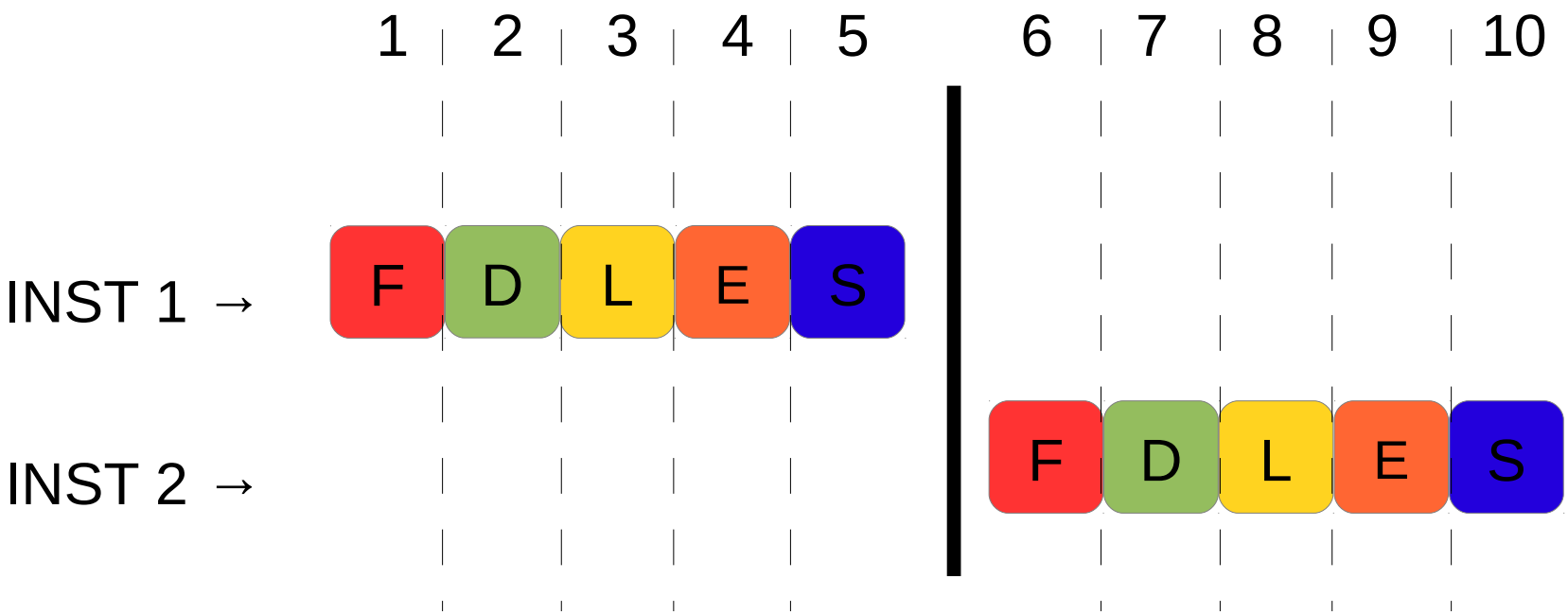
Modelo de von Neumann

5. STORE: guarda el resultado en memoria

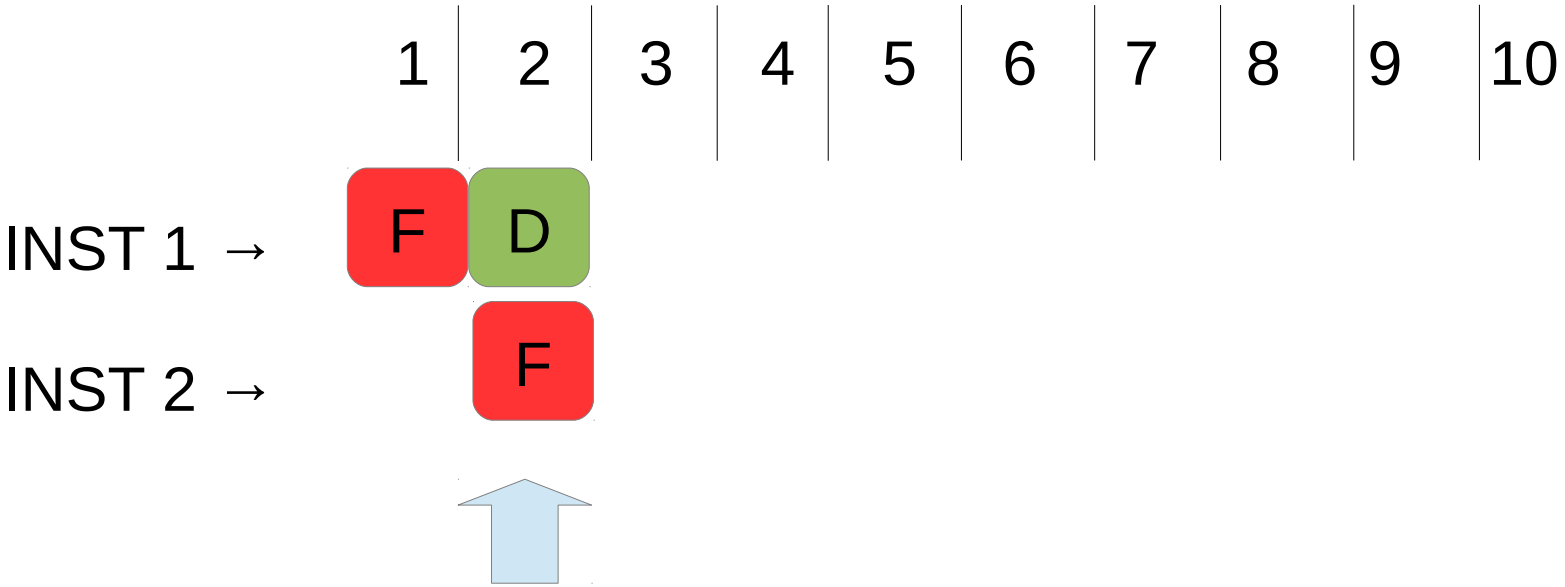


RAM $\leftarrow$ r1

Procesamiento secuencial



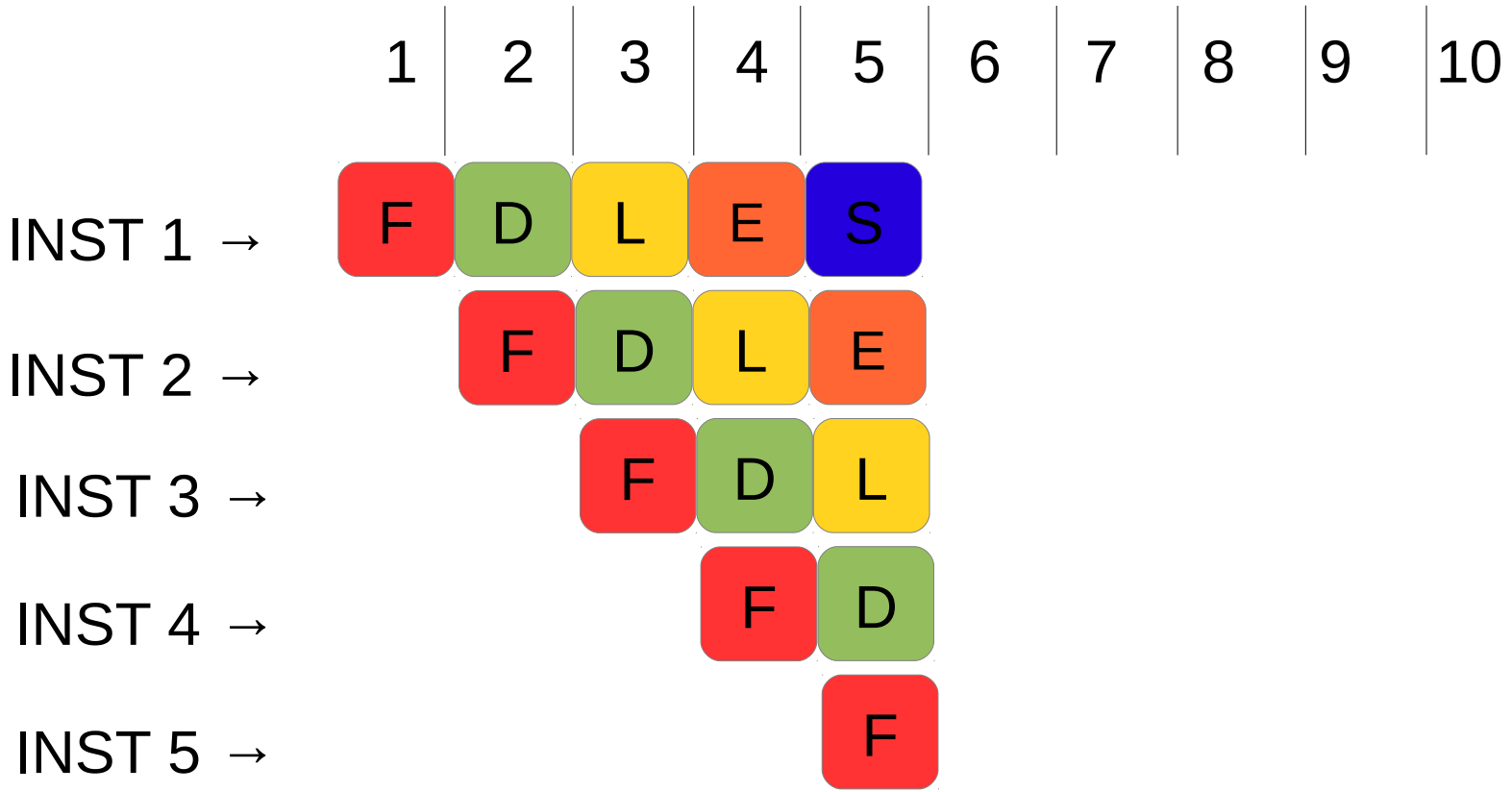
Pipelining



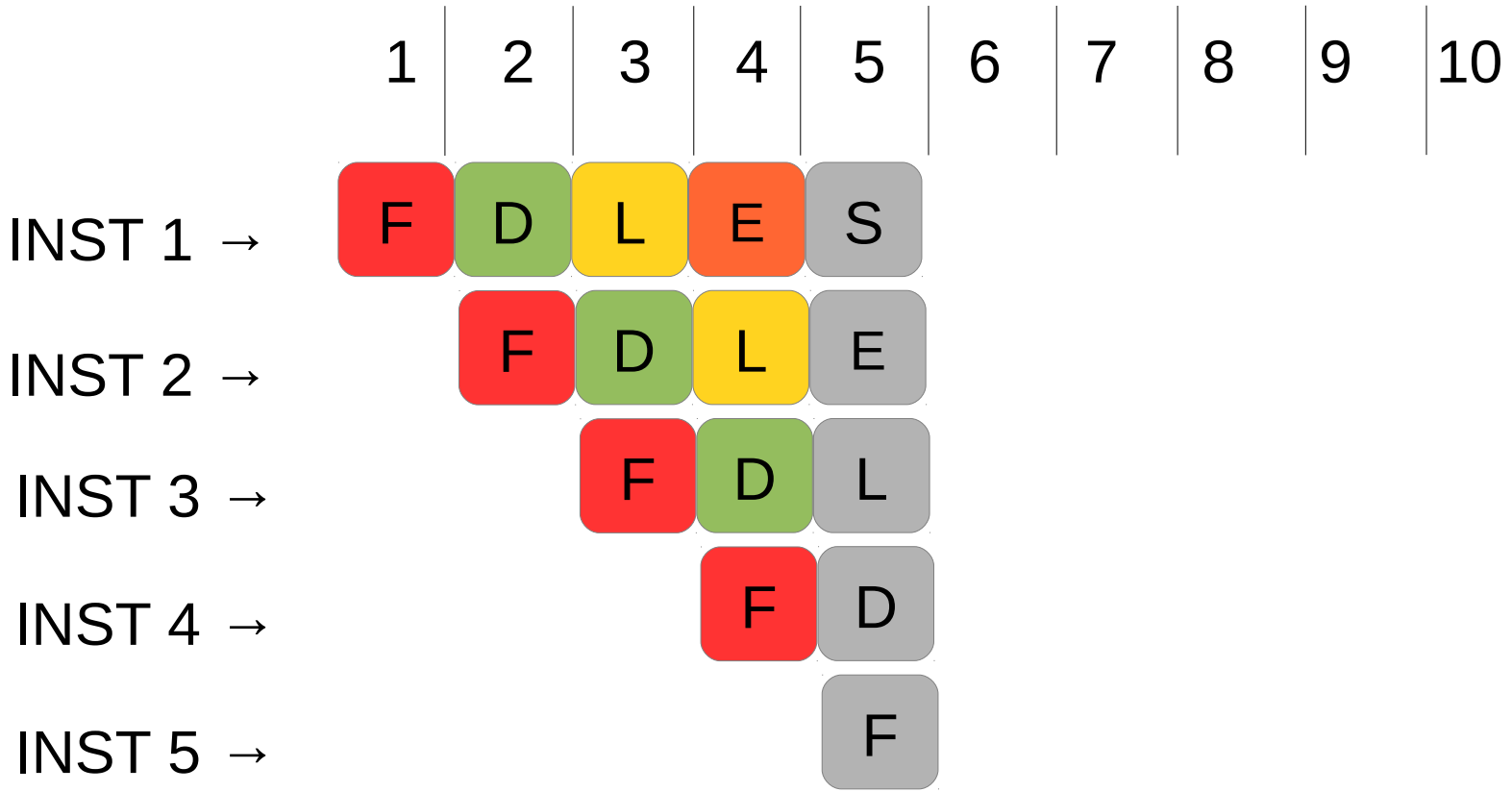
Pipelining

Pipelining

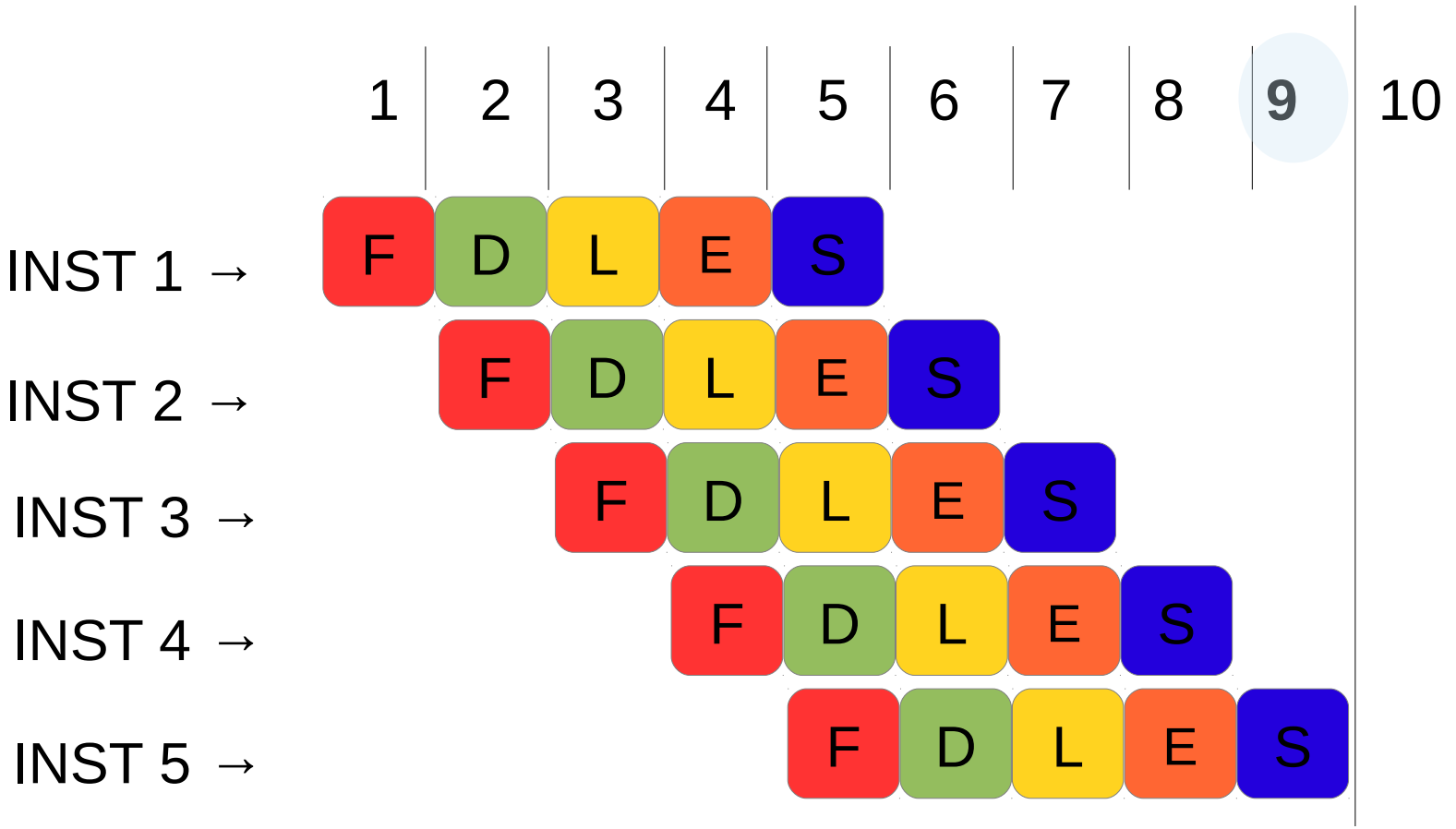
Pipelining


1
2
3
4
5
6
7
8
9
10
INST 1 →
INST 2 →
INST 3 →
INST 4 →
INST 5 →
F
D
L
E
S
F
D
L
E
F
D
L
F
D
F

Pipelining


1
2
3
4
5
6
7
8
9
10
INST 1 →
INST 2 →
INST 3 →
INST 4 →
INST 5 →
F
D
L
E
S
F
D
L
E
F
D
L
F
D
F

Pipelining



Pipelining

Veamos un ejemplo

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

WTPC workshop en técnicas de programación científica

Pipelining

Veamos un ejemplo

z=a*b+c*d

En realidad:

z1=a*b

z2=c*d

z=z1+z2

Solo 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

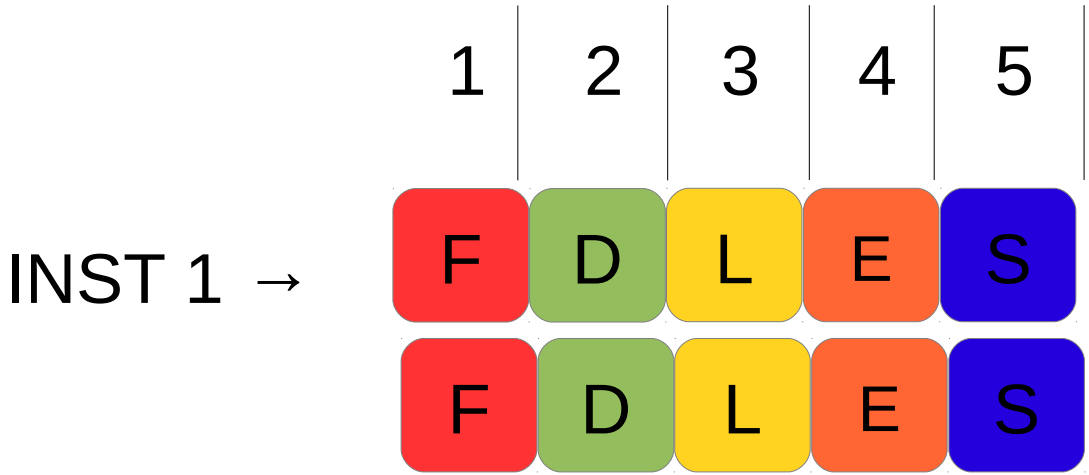
C7. almacenar R6 en z

Inst. prox. operación

Problema: dependencias!

WTPC workshop en técnicas de programación científica

Superscaling



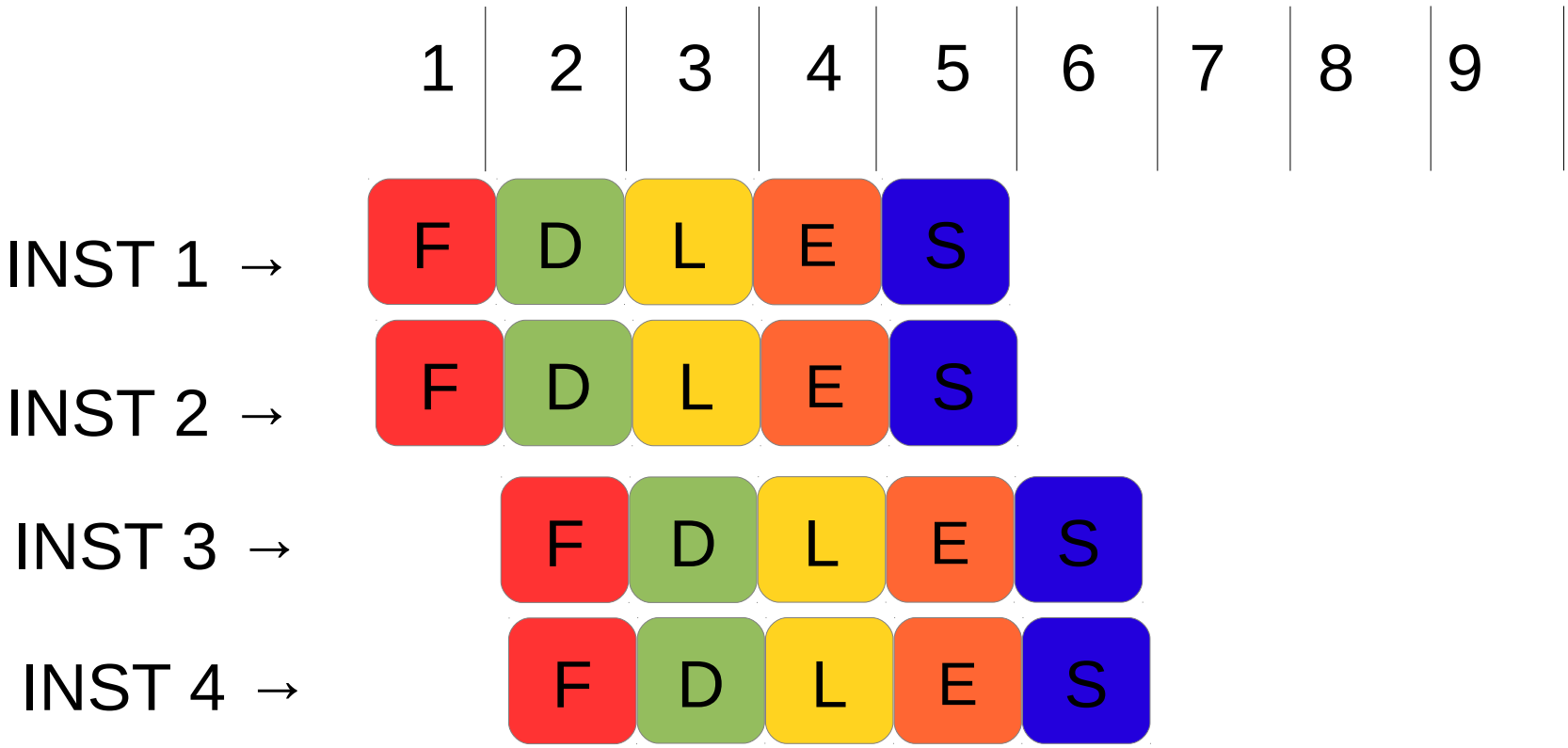
+ de una instrucción por ciclo
inst. independientes

Superscaling

Ojo! Sigue siendo escalar!

- Paralelismo a nivel de instrucción
- Muchas instrucciones se ejecutan simultáneamente, generalmente combinado con pipelining.

Pipelining + Superscaling



Pipelining + Superscaling

Veamos un ejemplo

z=a*b+c*d

En realidad:

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

Inst. prox. operación

C4. R6=R2+R5

Inst. prox. operación

C5. almacenar R6 en z

Inst. prox. operación

WTPC workshop en técnicas de programación científica

Pipelining + Superscaling

Veamos un ejemplo

z=a*b+c*d

En realidad:

z1=a*b

z2=c*d

z=z1+z2

Ganamos 3 paso de 8!

+

3 más si la próxima operación es independiente

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

Inst. prox. operación

C4. R6=R2+R5

Inst. prox. operación

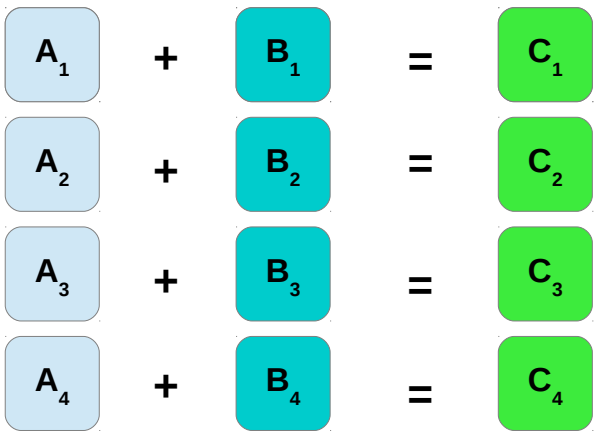
C5. almacenar R6 en z

Inst. prox. operación

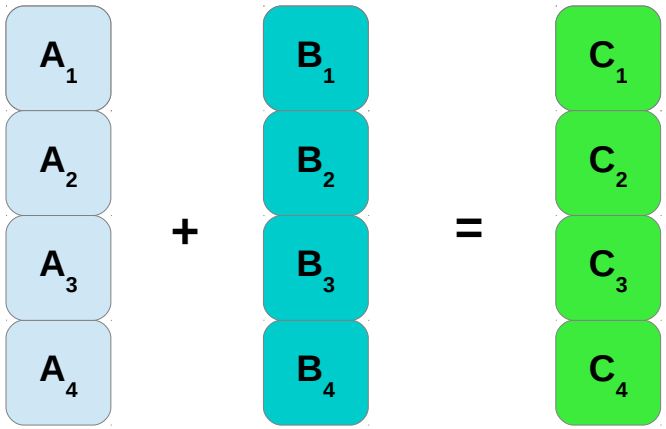
WTPC workshop en técnicas de programación científica

Operaciones escalares vs operaciones vectoriales

Registros escalares



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).

Operaciones escalares: Pipelining + Superscaling

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

Repetir pasos 4-5 según índice

WTPC workshop en técnicas de programación científica

Operaciones vectoriales: Pipelining + Superscaling

```
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){
        z[i]=a[i]*b[i]+c[i]*d[i];
        z[i+1]=a[i+1]*b[i+1]+c[i+1]*d[i+1];}
```

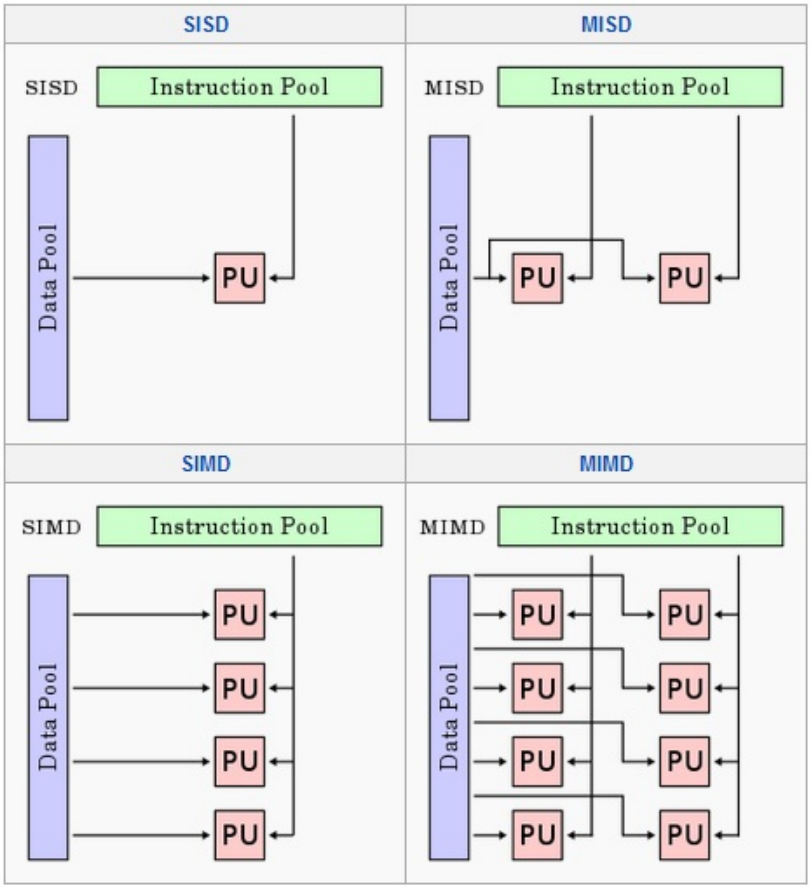
WTPC workshop en técnicas de programación científica

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



Como medir la performance

Unidad de medida $\rightarrow$FLOP/s

Peak Performance (teórico):
Estimación del desempeño de la CPU cuando trabaja a máxima velocidad

Benchmark Performance:
Se utilizan herramientas específicas para medir el pico de performance "real".

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

WTPC workshop en técnicas de programación científica

Como medir la performance: Benchmark

Linpack: 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/)

OS	Linux 6.2 RedHat (Kernel 2.2.14)
C compiler	gcc (egcs-2.91.66 egcs-1.1.2 release)
C flags	-fomit-frame-pointer -O3 -funroll-loops
MPI	MPIch 1.2.1
BLAS	ATLAS (Version 3.0 beta)
Comments	09 / 00

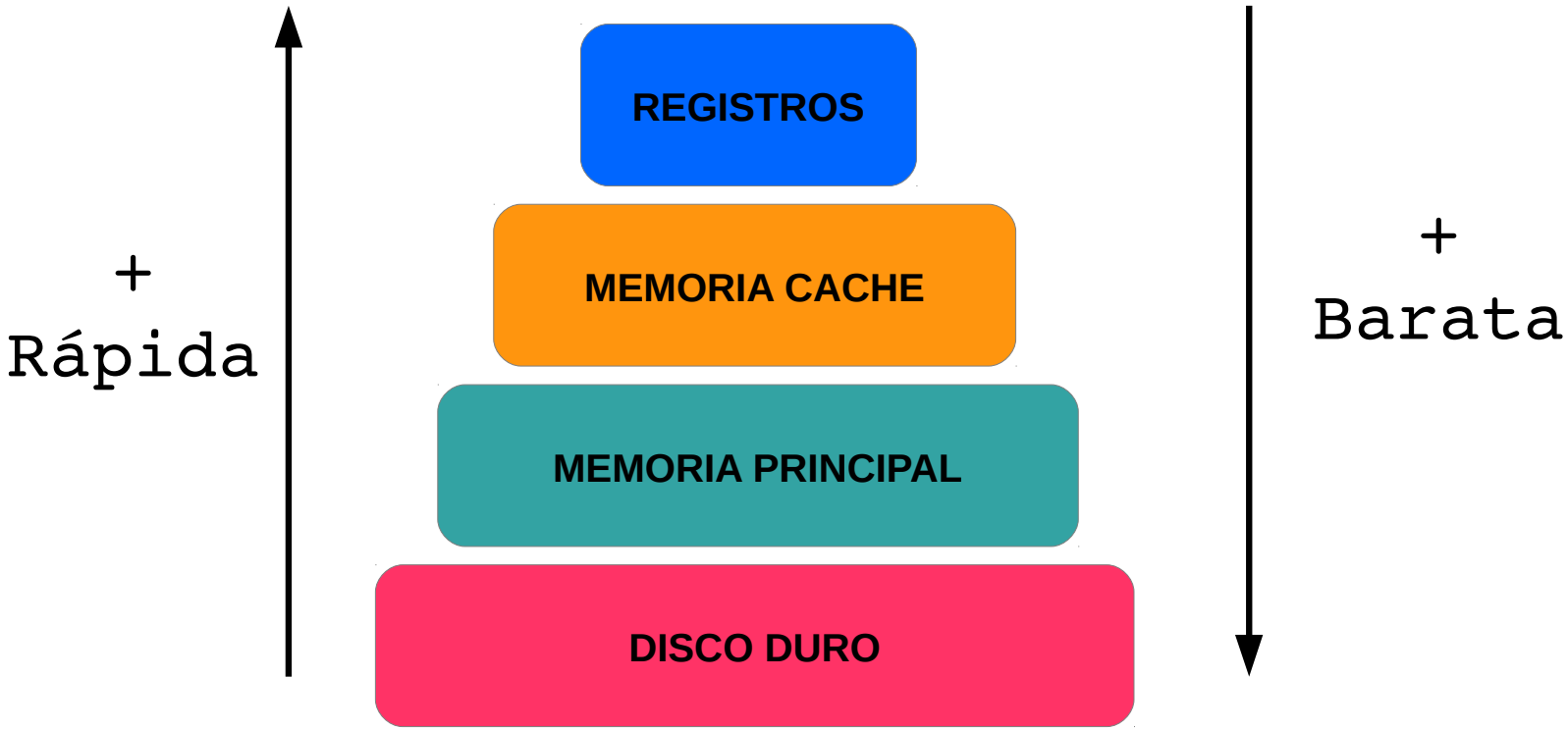
Performance (Gflops) w.r.t Problem size on 4 nodes.

GRID	**2000**	**5000**	**8000**	**10000**
1 x 4	1.28	1.73	1.89	1.95
2 x 2	1.17	1.68	1.88	1.93
4 x 1	0.81	1.43	1.70	1.80

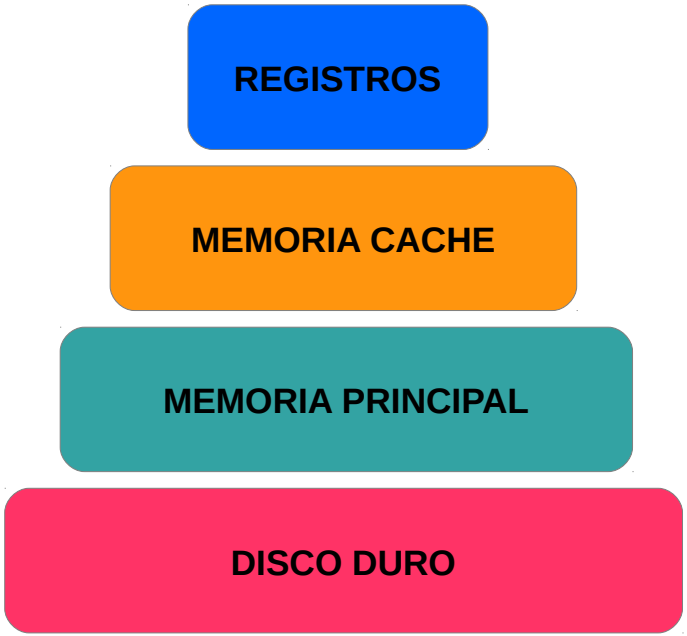
4 AMD Athlon K7 500 Mhz (256 Mb) - (2x) 100 Mbs Switched
2 NICs per node (channel bonding)

WTPC workshop en técnicas de programación científica

Jerarquía de memoria



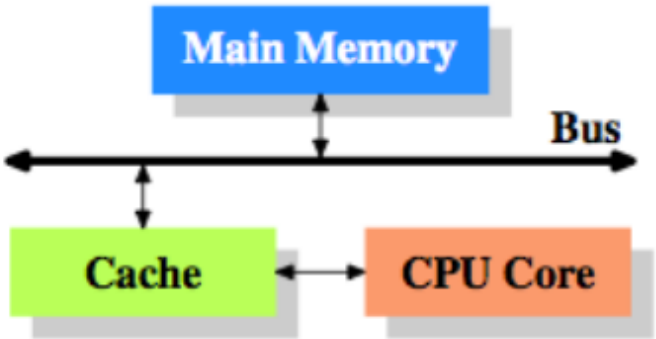
Jerarquía de memoria

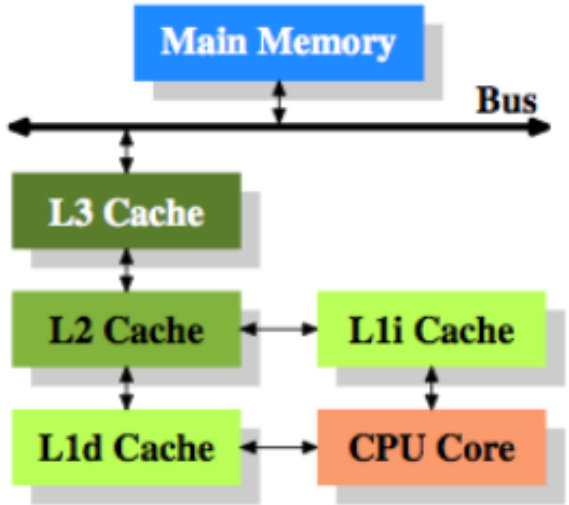


Registros: Cableados en el Procesador
Cache: Memoria rápida, cercana al procesador y cara
Memoria principal: RAM, lenta y barata
Disco duro: lentísimo y baratísimo

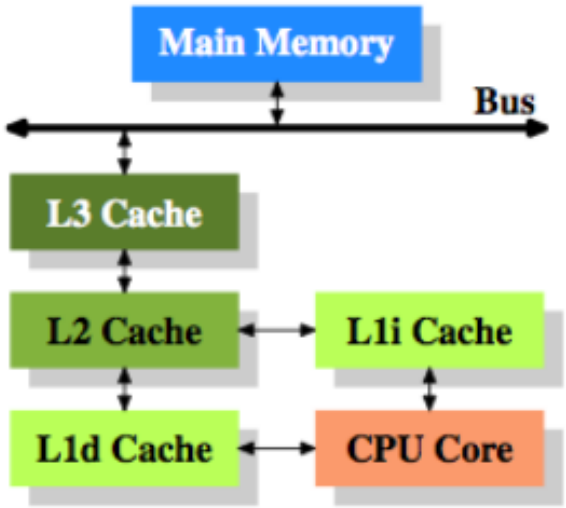
Memoria cache

Mayor costo
Mayor velocidad
Menor capacidad
Los datos se transfieren a cache en bloques de un determinado tamaño
$\rightarrow$**cache lines**





Memoria cache




```
maria@maria-UX21E: ~
maria@maria-UX21E:~$ lscpu
Arquitectura:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Orden de bytes:          Little Endian
CPU(s):                  4
On-line CPU(s) list:     0-3
Hilo(s) por núcleo:      2
Núcleo(s) por zócalo:    2
Socket(s):               1
Nodo(s) NUMA:            1
ID del vendedor:         GenuineIntel
Familia de CPU:          6
Modelo:                  42
Stepping:                7
CPU MHz:                 800.000
BogoMIPS:                3190.05
Virtualización:          VT-x
caché L1d:               32K
caché L1i:               32K
caché L2:                256K
caché L3:                3072K
NUMA node0 CPU(s):       0-3
```

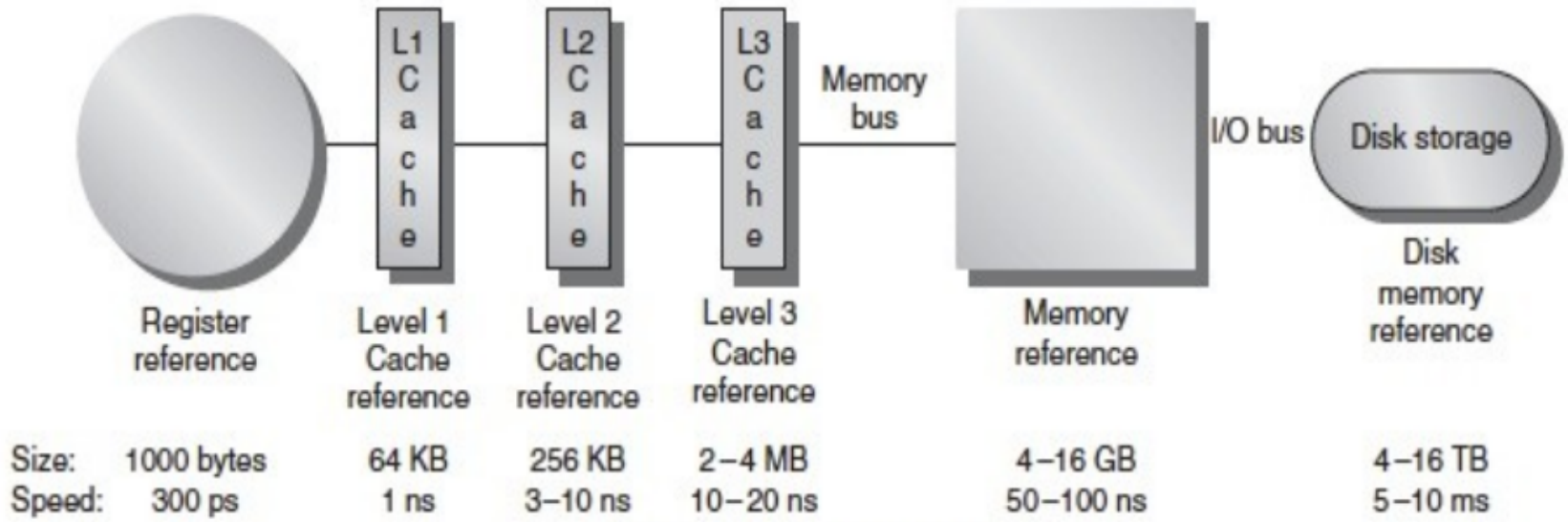
Memoria cache

Principio de localidad:

- **Espacial**
  Por ejemplo, recorrer un vector con un loop
- **Temporal**
  Por ejemplo, un llamado a una función dentro de un loop

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

Memoria cache



Jerarquía de memoria

Como comparar de manera mas legible las velocidades?

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 µs	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

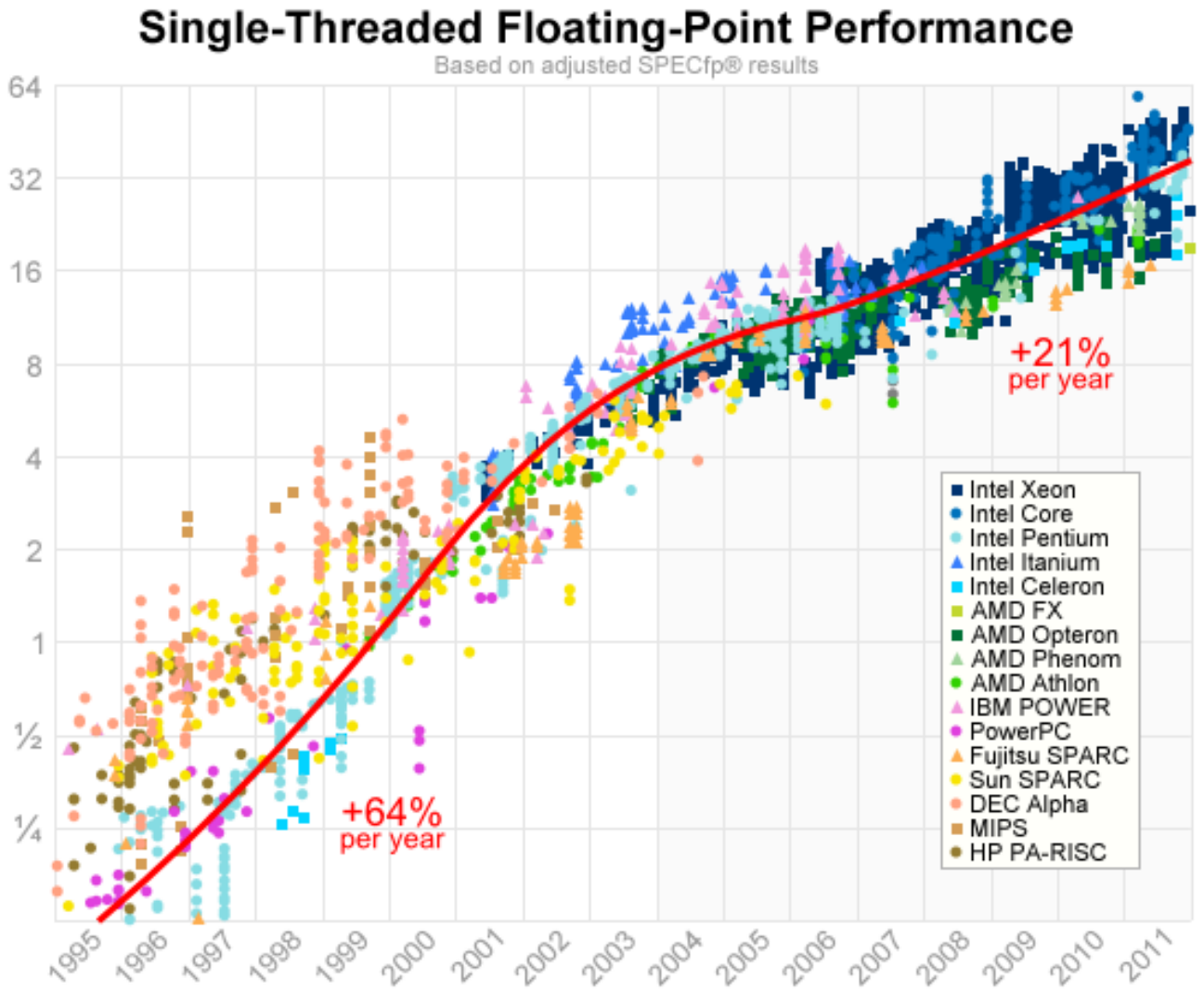
Jerarquía de memoria

Todo esto para un solo procesador!

↓

Actualmente todas las CPUs poseen más de un procesador

Multiprocesadores

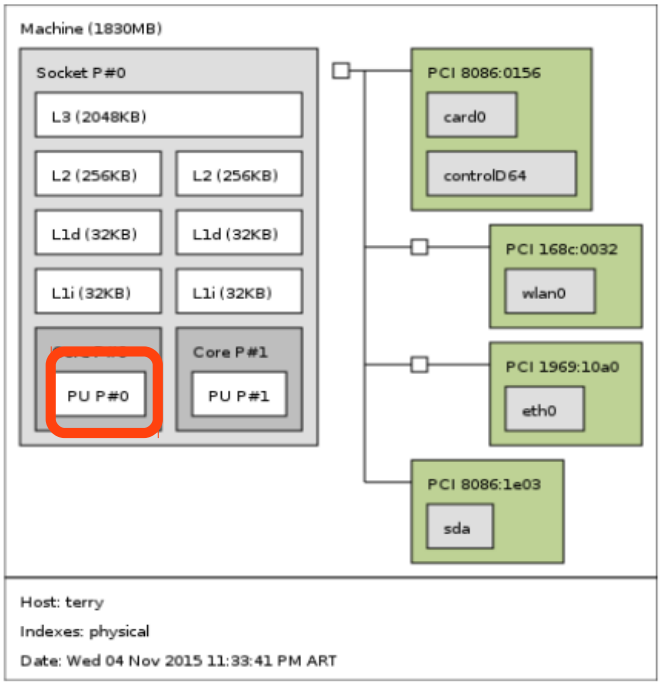


Ley de Moore ?

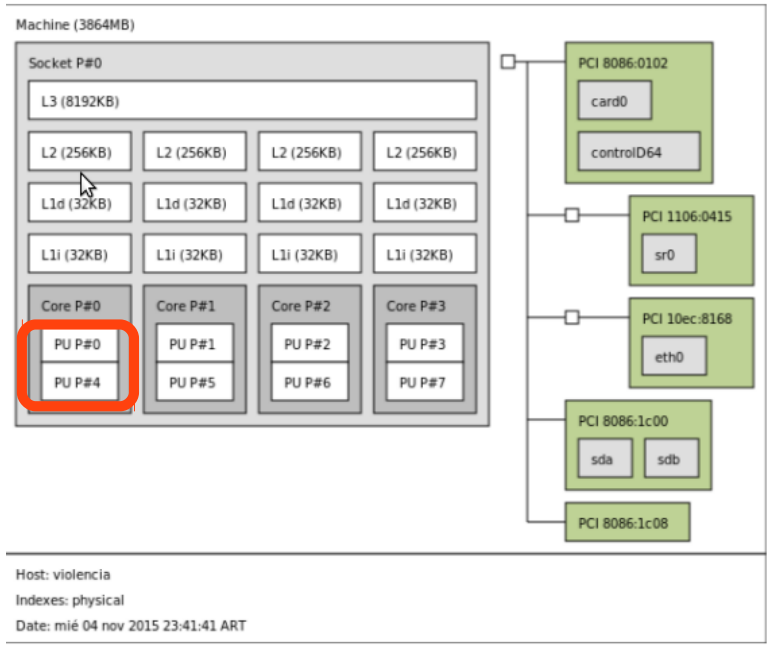
WTPC workshop en técnicas de programación científica

Multiprocesadores

Como es en realidad



Intel(R) Celeron(R) CPU 1007U



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

Básicamente von Neumann

WTPC workshop en técnicas de programación científica

Arquitectura del Computador

M. Graciela Molina

m.graciela.molina@gmail.com

WTPC
workshop en técnicas de programación científica