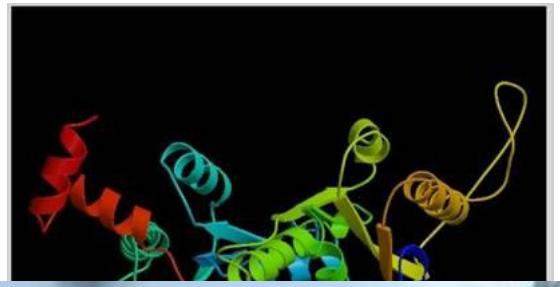
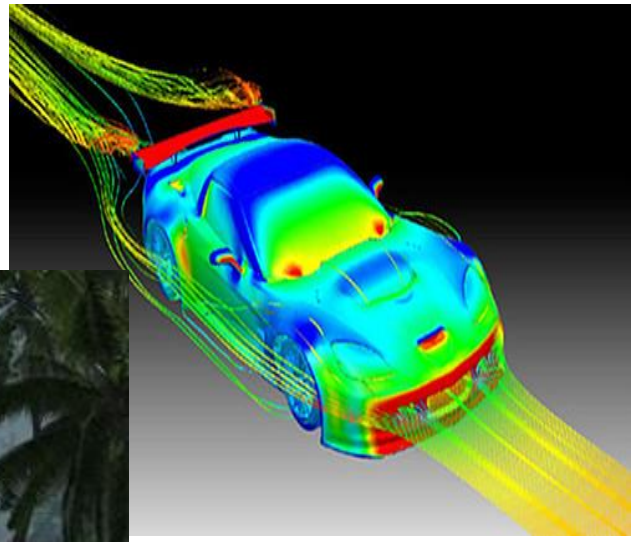
Ingeniería de los Computadores

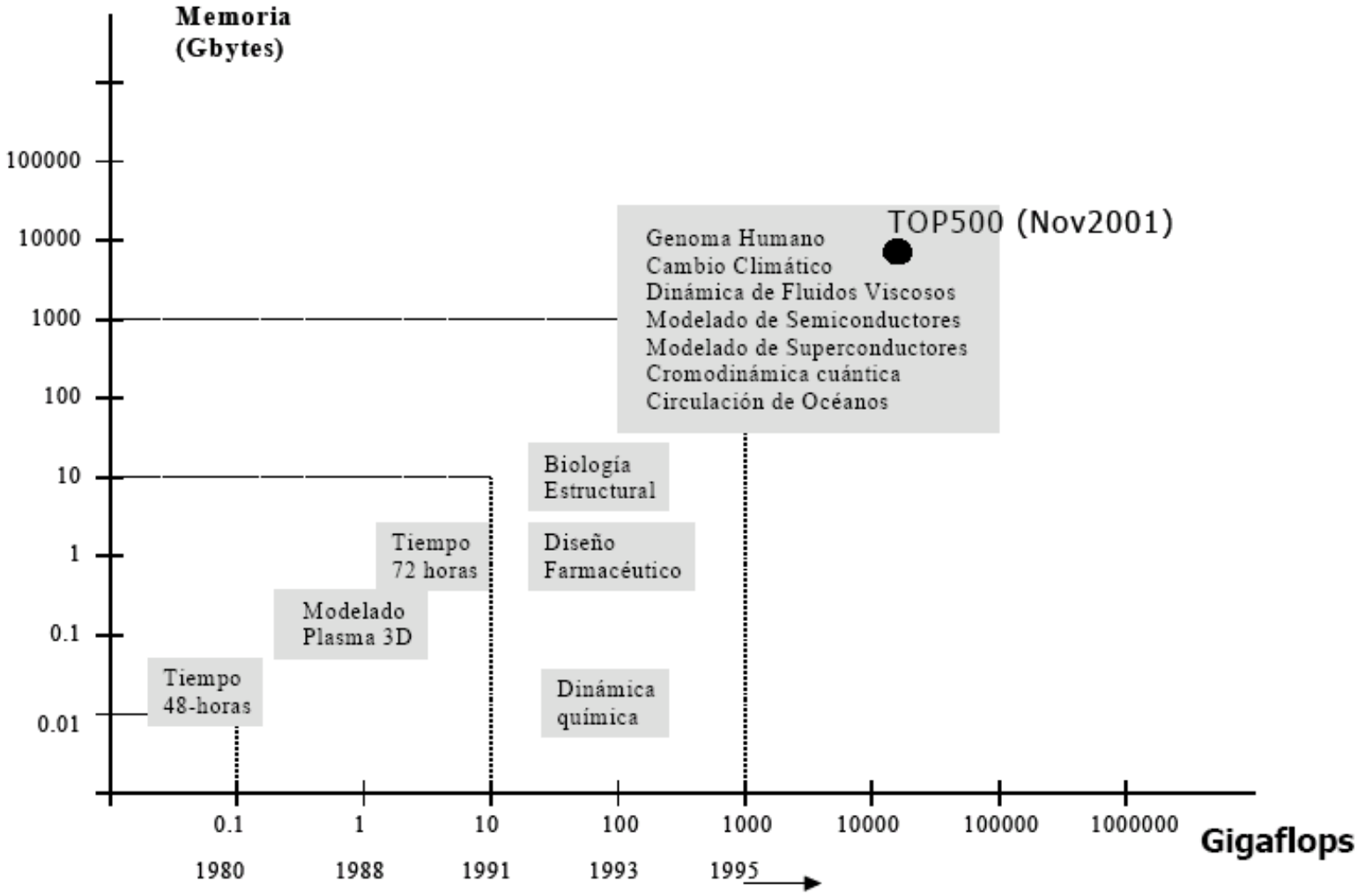
Unidad 1. Introducción al paralelismo

¿Qué?

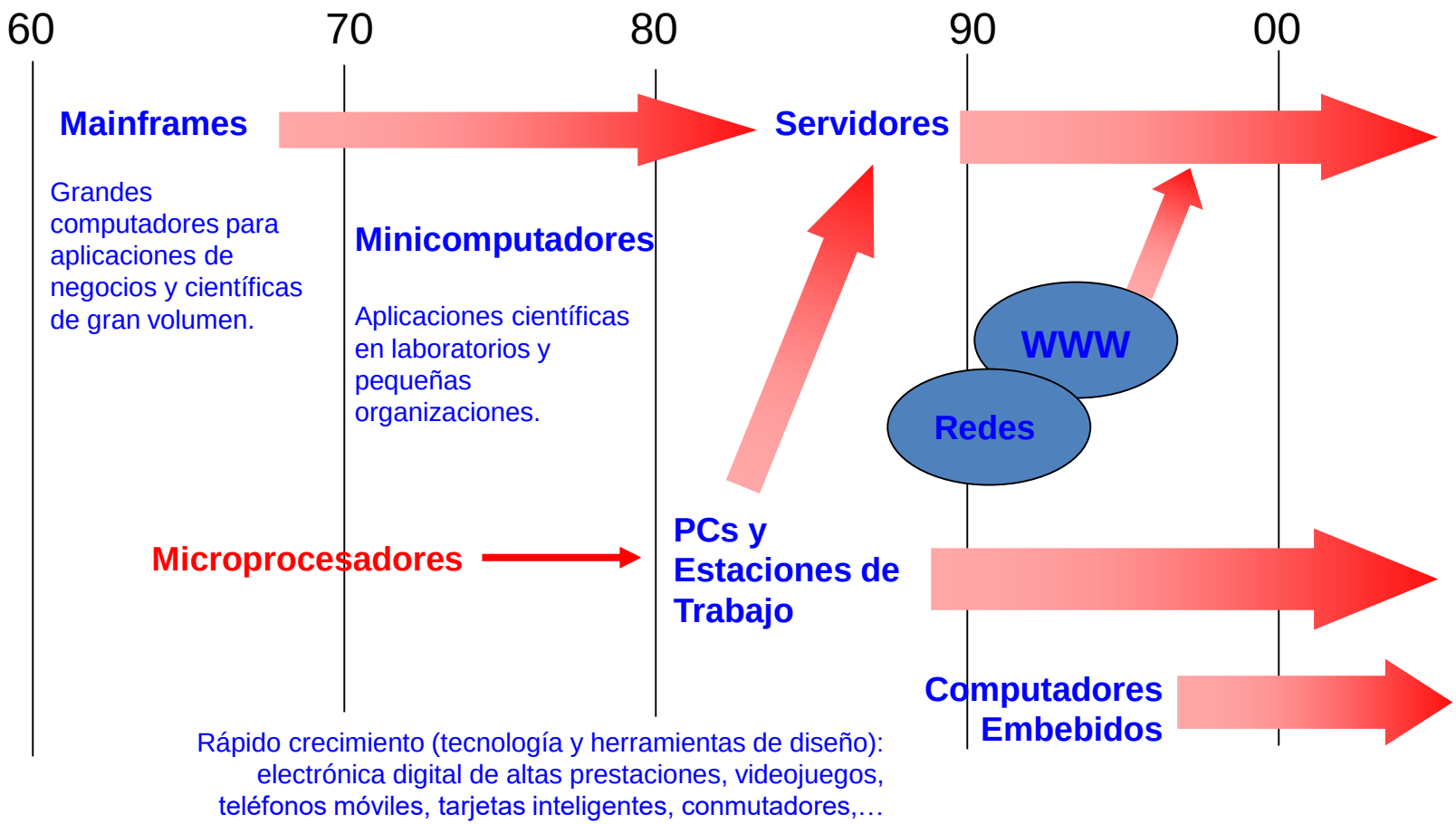
- Evolución de la informática ⟺ Necesidades computacionales


¿Qué?

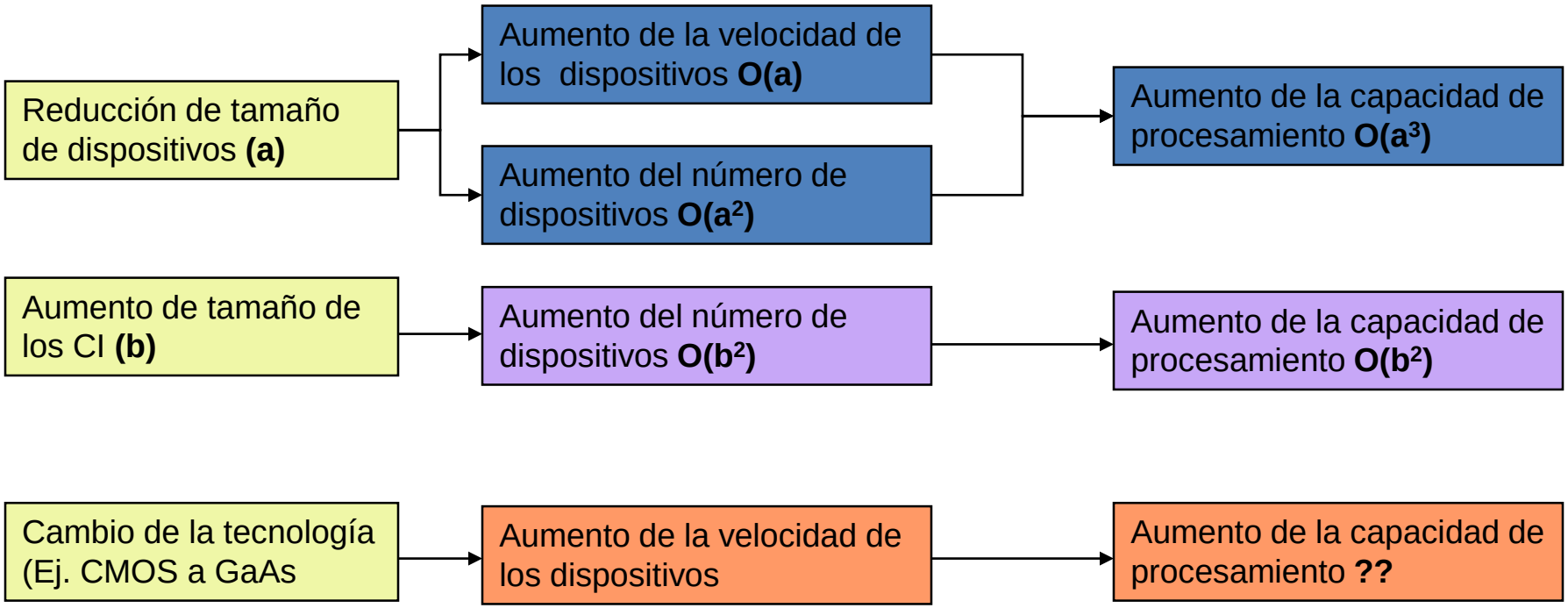

Memoria
(Gbytes)
100000
10000
1000
100
10
1
0.1
0.01
TOP500 (Nov2001)
Genoma Humano
Cambio Climático
Dinámica de Fluidos Viscosos
Modelado de Semiconductores
Modelado de Superconductores
Cromodinámica cuántica
Circulación de Océanos
Biología
Estructural
Diseño
Farmacéutico
Dinámica
química
Tiempo
72 horas
Modelado
Plasma 3D
Tiempo
48-horas
0.1
1
10
100
1000
10000
100000
1000000
Gigaflops
1980
1988
1991
1993
1995

¿Qué?


60
70
80
90
00
Mainframes
Grandes computadores para aplicaciones de negocios y científicas de gran volumen.
Minicomputadores
Aplicaciones científicas en laboratorios y pequeñas organizaciones.
Microprocesadores
Servidores
WWW
Redes
PCs y Estaciones de Trabajo
Computadores Embebidos
Rápido crecimiento (tecnología y herramientas de diseño): electrónica digital de altas prestaciones, videojuegos, teléfonos móviles, tarjetas inteligentes, conmutadores,…

¿Qué?

- Mejora de prestaciones
  - ➢ Avances en tecnologías (límites físicos: calor, ruido, etc.)



¿Qué?

- Mejora de prestaciones
  - Avances en arquitecturas
    - **Paralelismo**:
      - Segmentación de cauces.
      - Repetición de elementos: Utilizar varias unidades funcionales, procesadores, módulos de memoria, etc. para distribuir el trabajo.
    - **Localidad:** Acercar datos e instrucciones al lugar donde se necesitan para que el acceso a los mismos sea lo más rápido posible (jerarquía de memoria).

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso software

- Repertorio de instrucciones (RISC, CISC, …)
- Arquitecturas VLIW
- Extensiones SIMD (MMX, SSE, 3DNOW, …)

¿Qué?

- Desarrollo de las arquitecturas de computadores - Objetivo

MAYOR CAPACIDAD COMPUTACIONAL

Iniciativas mayor peso hardware

- Arquitecturas segmentadas
- Arquitecturas vectoriales
- Arquitecturas superescalares
- Arquitecturas paralelas o de alto rendimiento

¿Qué?

- Arquitectura de Computadores

"Conjunto de instrucciones, recursos y características del procesador que son visibles al software que se ejecuta en el mismo. Por tanto, la arquitectura determina el software que el procesador puede ejecutar directamente, y esencialmente define las especificaciones a las que debe ajustarse la microarquitectura" [Ortega, 2005]

- En Ingeniería de Computadores veremos:
  - Arquitecturas superescalares
  - Arquitecturas paralelas: multicomputadores y multiprocesadores

¿Qué?

- Ámbito de la arquitectura de computadores
  - El lenguaje máquina del computador, la microarquitectura del procesador y la interfaz para los programas en lenguaje máquina (lenguaje máquina y arquitectura concreta del procesador).
  - Los elementos del computador y como interactúan (es decir la arquitectura concreta del computador, la estructura y organización).
  - La interfaz que se ofrece a los programas de alto nivel y los módulos que permiten controlar el funcionamiento del computador (sistema operativo y la arquitectura abstracta del computador).
  - Los procedimientos cuantitativos para evaluar los sistemas (benchmarking).
  - Las alternativas posibles y las tendencias en su evolución

¿Qué?

- Niveles estructurales de Bell y Newell
  - Descripción del computador mediante una aproximación por capas.
  - Cada capa utiliza los servicios que proporciona la del nivel inferior.
  - Propone 5 niveles:
    - De componente
    - Electrónico
    - Digital
    - Transferencia entre registros (RT)
    - Procesador-Memoria-Interconexión (PMS)

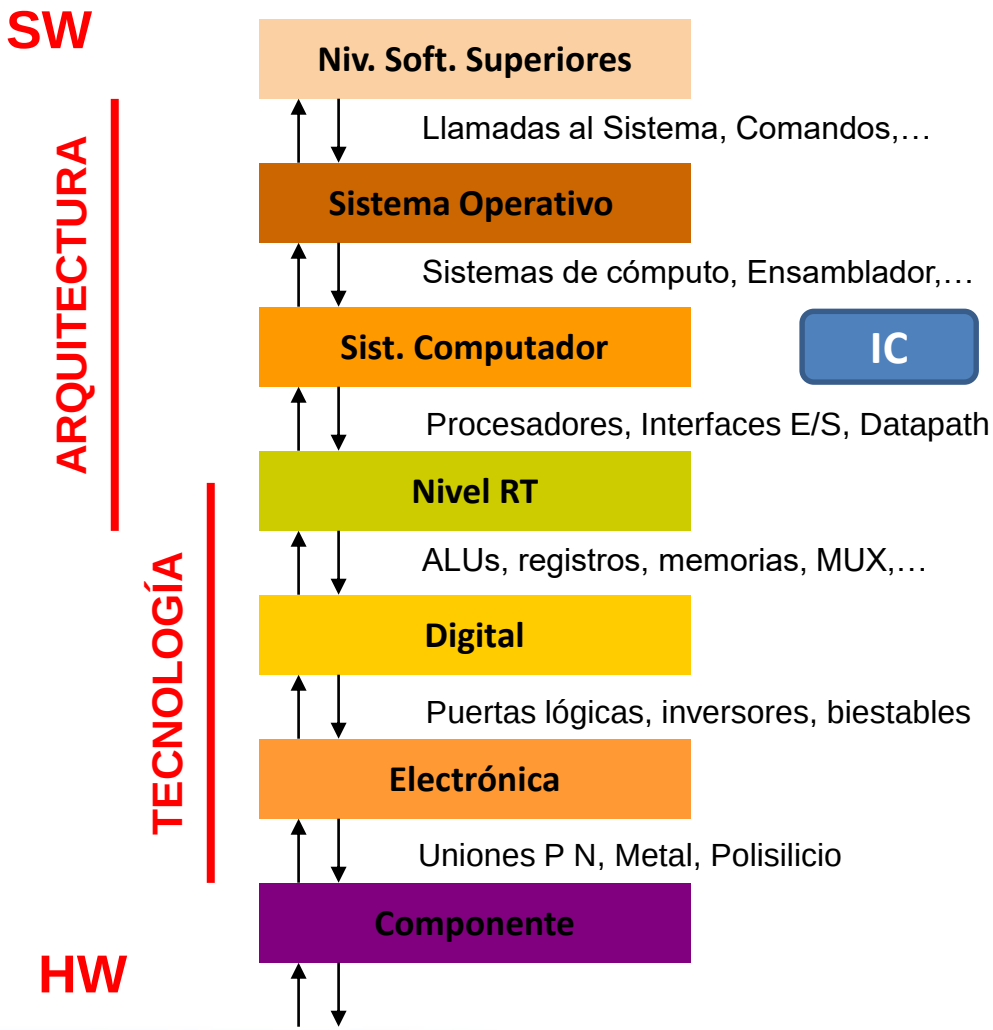
¿Qué?

- Niveles de interpretación de Levy
  - Contemplan al computador desde un punto de vista funcional.
  - Constituido por una serie de máquinas virtuales superpuestas.
  - Cada máquina interpreta las instrucciones de su nivel, proporcionando servicios a la máquina de nivel superior y aprovechando los de la máquina de nivel inferior.
  - Se distinguen 5 niveles:
    - Aplicaciones
    - Lenguajes de alto nivel
    - Sistema Operativo
    - Instrucciones máquina
    - Microinstrucciones
  - Estos niveles son similares a los niveles funcionales de Tanenbaum

¿Qué?

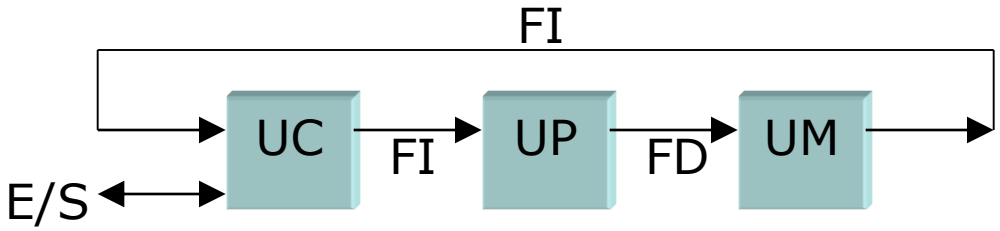
- Niveles de abstracción de un computador

Integra la orientación **estructural** de los niveles de Bell y Newell y el punto de vista **funcional** de los niveles de Levy y Tanenbaum.

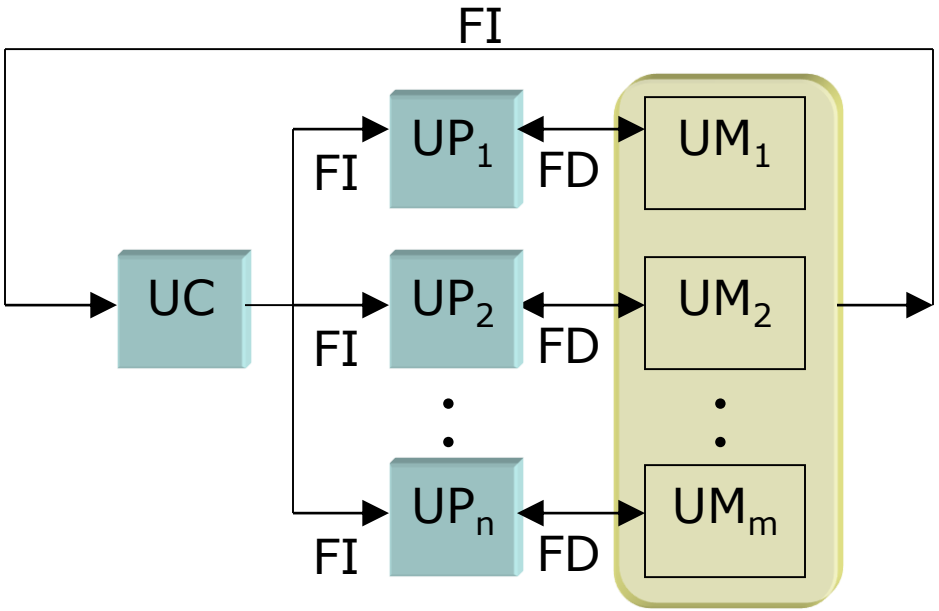


¿Qué?

- Taxonomía de Flynn
  - SISD

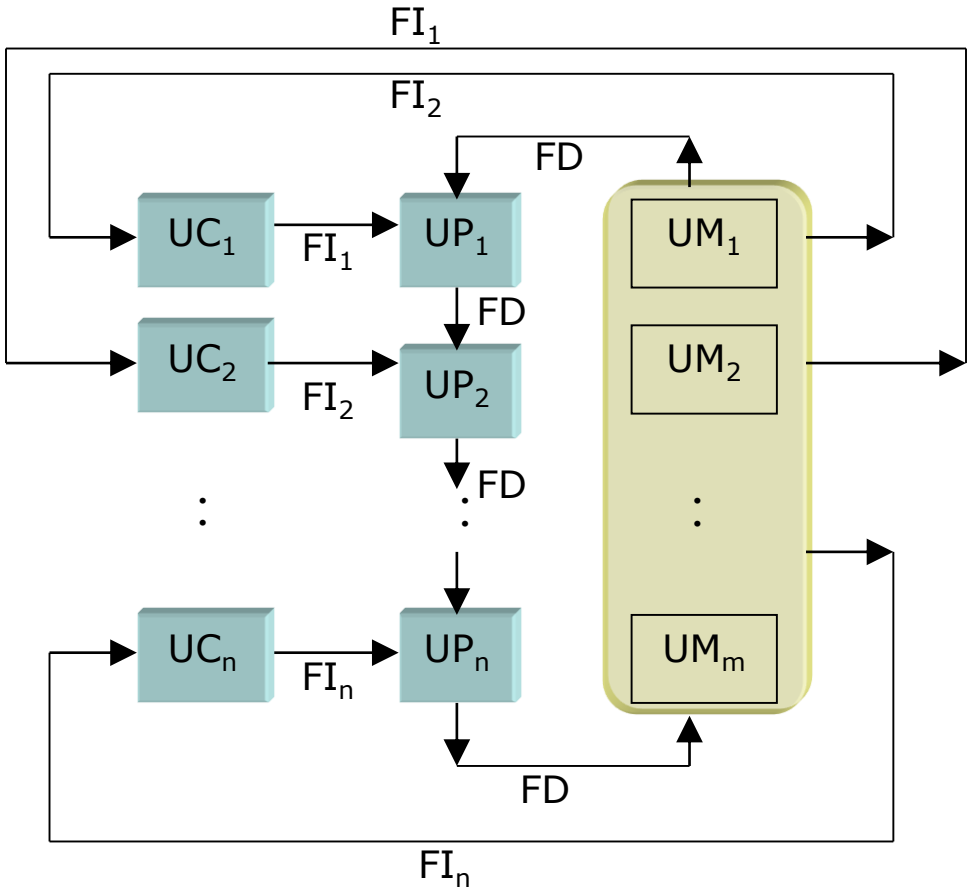


  - SIMD



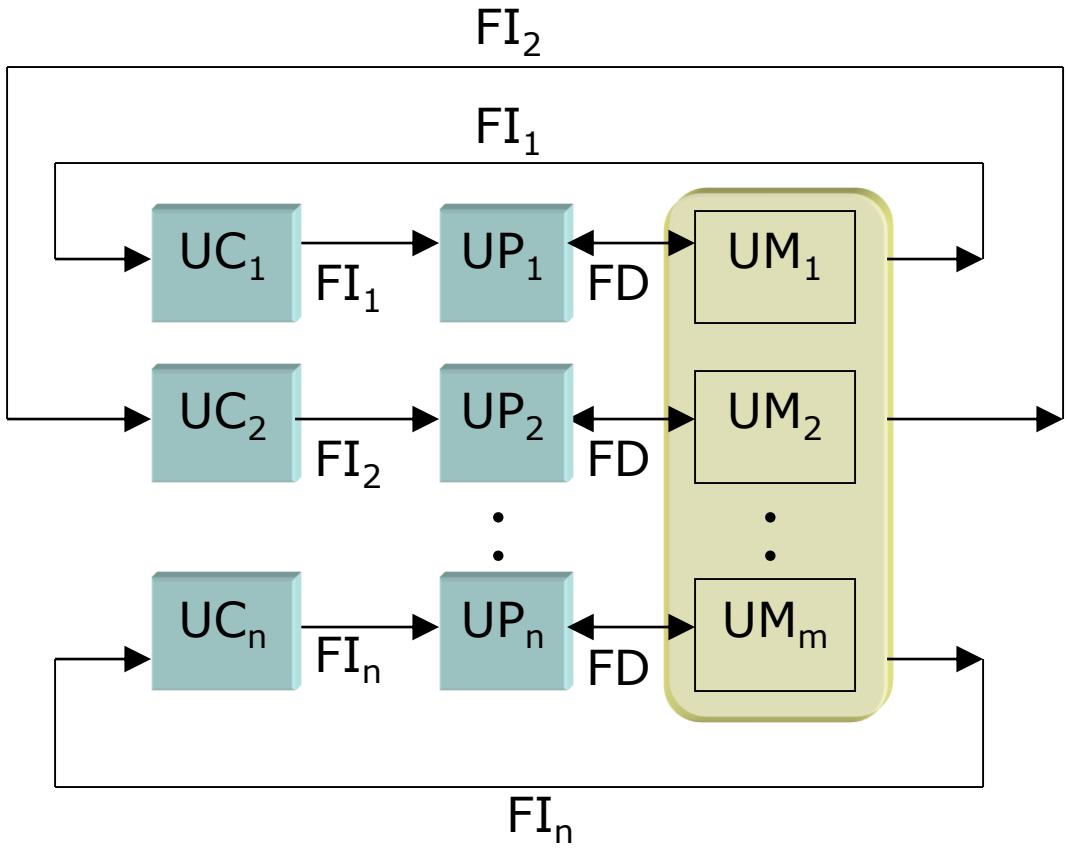
¿Qué?

- Taxonomía de Flynn
  - MISD



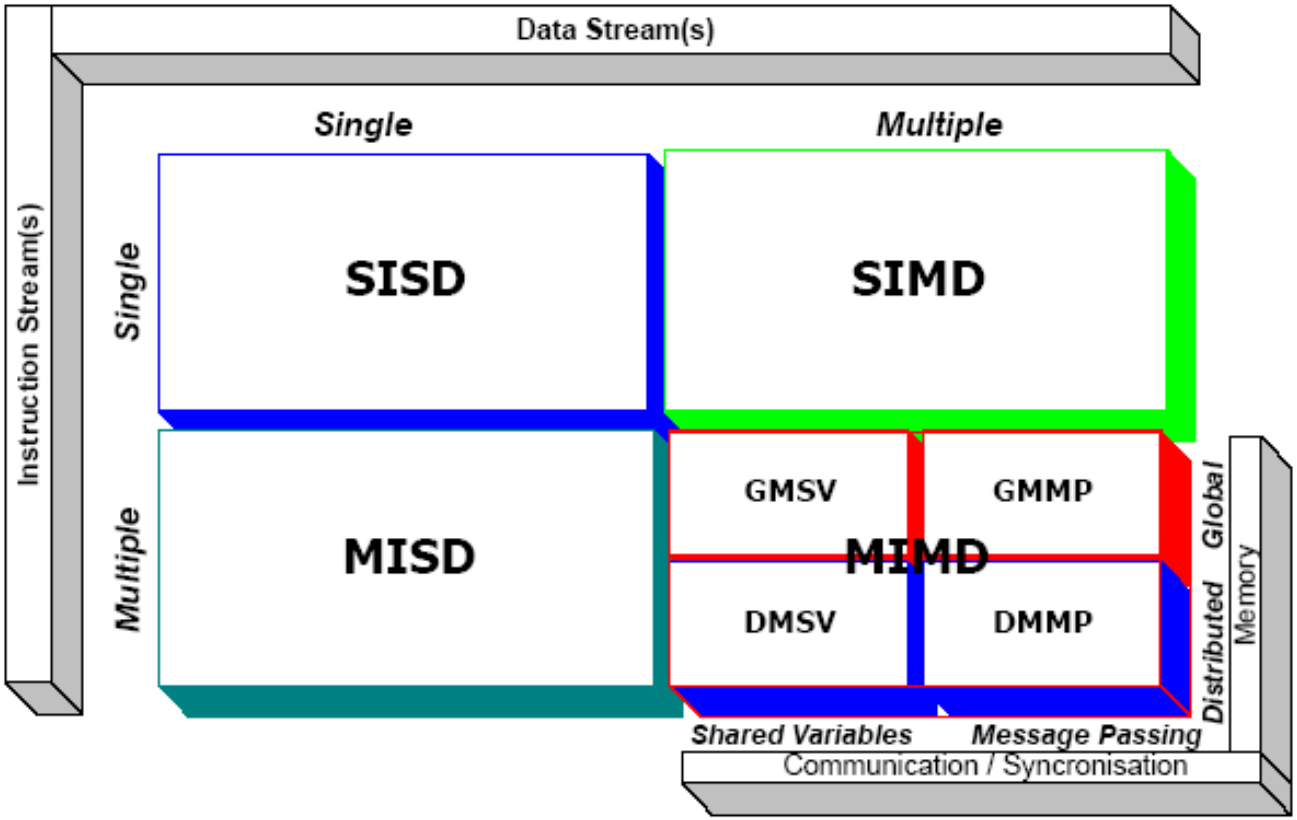
¿Qué?

- Taxonomía de Flynn
  - MIMD



¿Qué?

- Taxonomía de Flynn-Johnson

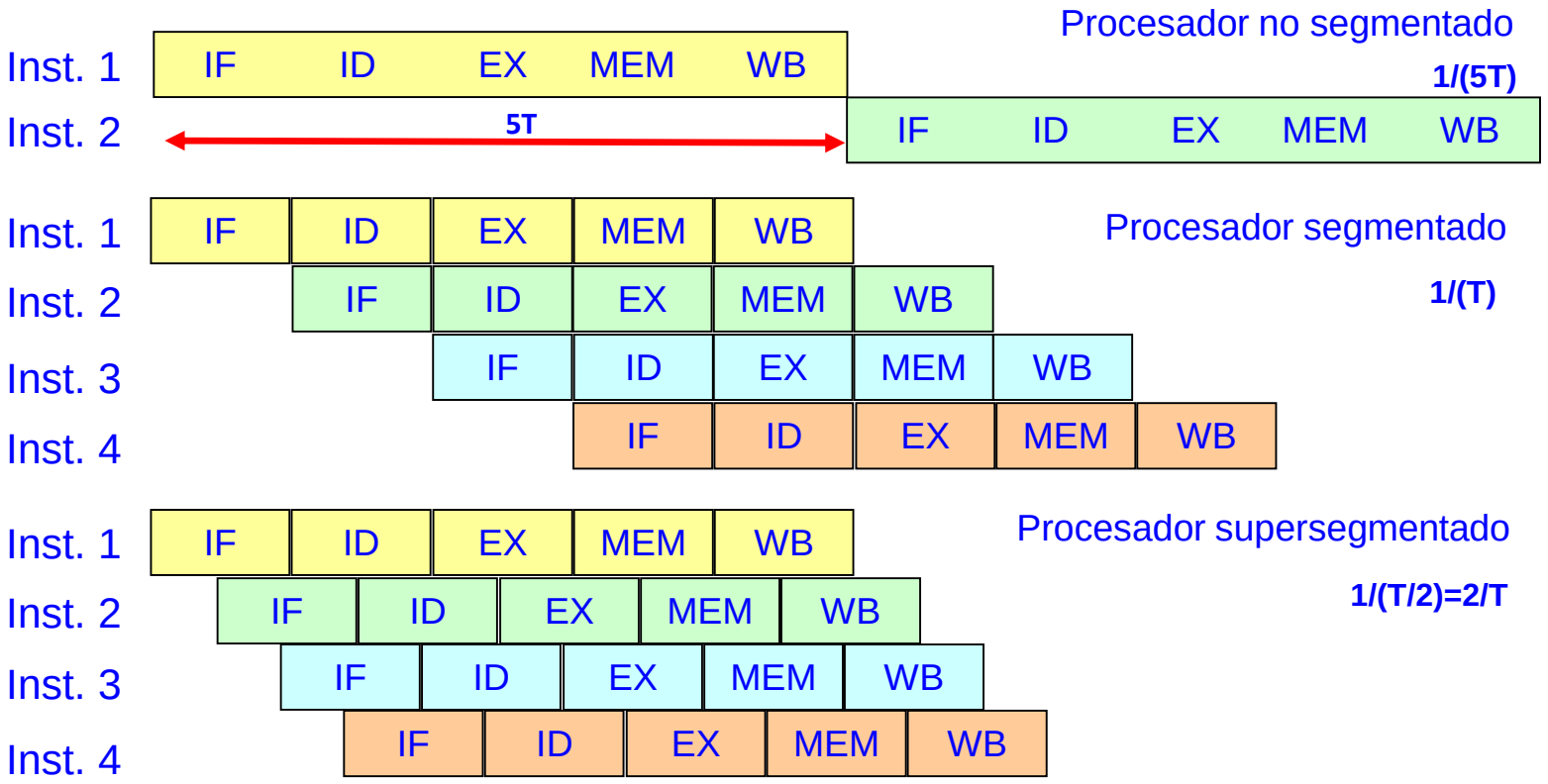


Tipos de paralelismo

- Tipos de paralelismo
  - **Paralelismo de datos:** La misma función, instrucción, etc. se ejecuta en paralelo pero en cada una de esas ejecuciones se aplica sobre un conjunto de datos distinto.
  - **Paralelismo funcional:** Varias funciones, tareas, instrucciones, etc. (iguales o distintas) se ejecutan en paralelo.
    - Nivel de instrucción (ILP) – se ejecutan en paralelo las instrucciones de un programa. Granularidad fina.
    - Nivel de bucle o hebra (Thread) – se ejecutan en paralelo distintas iteraciones de un bucle o secuencias de instrucciones de un programa. Granularidad fina/media.
    - Nivel de procedimiento (Proceso) –distintos procedimientos que constituyen un programa se ejecutan simultáneamente. Grano medio.
    - Nivel de programa – la plataforma ejecuta en paralelo programas diferentes que pueden corresponder, o no, a una misma aplicación. Granularidad gruesa.

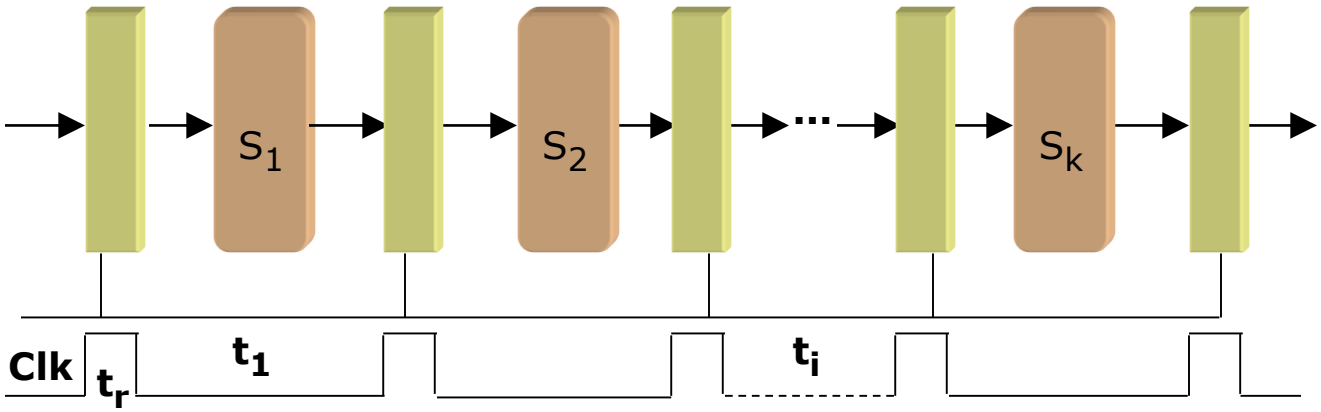
ILP

- Segmentación: ILP



ILP

- Segmentación
  - Identificación de fases en el procesamiento de una tarea.
  - Rediseño para implementar cada fase de forma independiente al resto.
  - Paralelismo por etapas (el sistema procesa varias tareas al mismo tiempo aunque sea en etapas distintas).
  - Se aumenta el número de tareas que se completan por unidad de tiempo.



- Segmentación

  Ganancia. Suponemos que TLI (tiempo de latencia de inicio) = T, tiempo que tarda en ejecutarse una operación en una unidad sin segmentar.

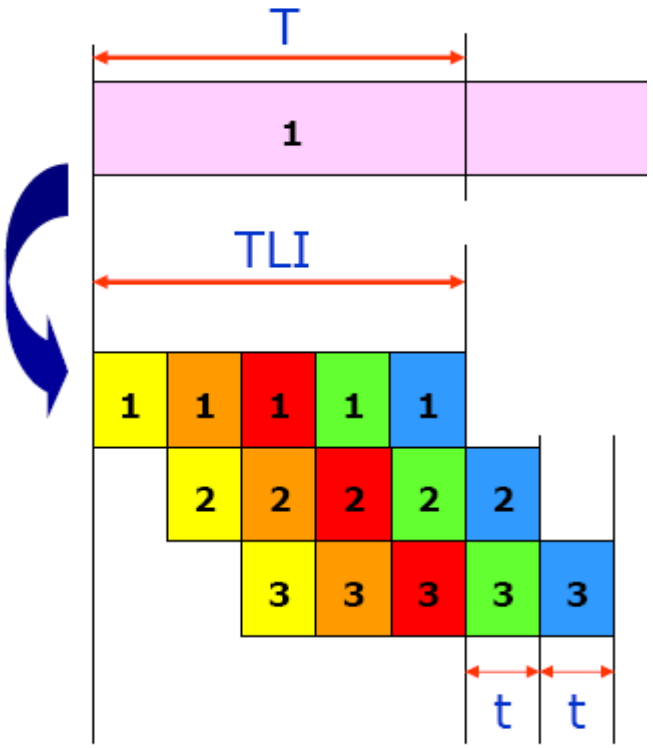
  TLI = k·t, siendo **k** el nº de etapas del cauce, y **t** la duración de cada etapa

$$G_k = \frac{T_1}{T_k} = \frac{k \cdot n \cdot t}{k \cdot t + (n-1) \cdot t} =$$

$$= \frac{k \cdot n}{k + n\text{-}1}$$

$$\lim_{n \to \infty} G_k = k$$

Normalmente, ***¿TLI>T ó TLI<T?***



- Ganancia real

$$G_k = \frac{T_{sin\_segmentar}}{T_{segmentado}} = \frac{n \times T}{TLI + (n-1) \times t}$$

$$G_{max} = \lim_{n \to \infty} \frac{n \times T}{(k \times t) + (n-1) \times t} = \frac{T}{t} < k$$

$TLI = kt$

$T < TLI = kt$

- Tipos de riesgos (detección del cauce)
  - **Riesgos de datos**. Se producen por dependencias entre operandos y resultados de instrucciones distintas.
  - **Riesgos de control**. Se originan a partir de instrucciones de salto condicional que, según su resultado, determinan la secuencia de instrucciones que hay que procesar tras ellas.
  - **Riesgos estructurales o colisiones**. Se producen cuando instrucciones diferentes necesitan el mismo recurso al mismo tiempo

ILP

- Riesgos de datos

RAW (Read After Write)

R2 := R1 + R2
R1 := R2 + R3

WAR (Write After Read)

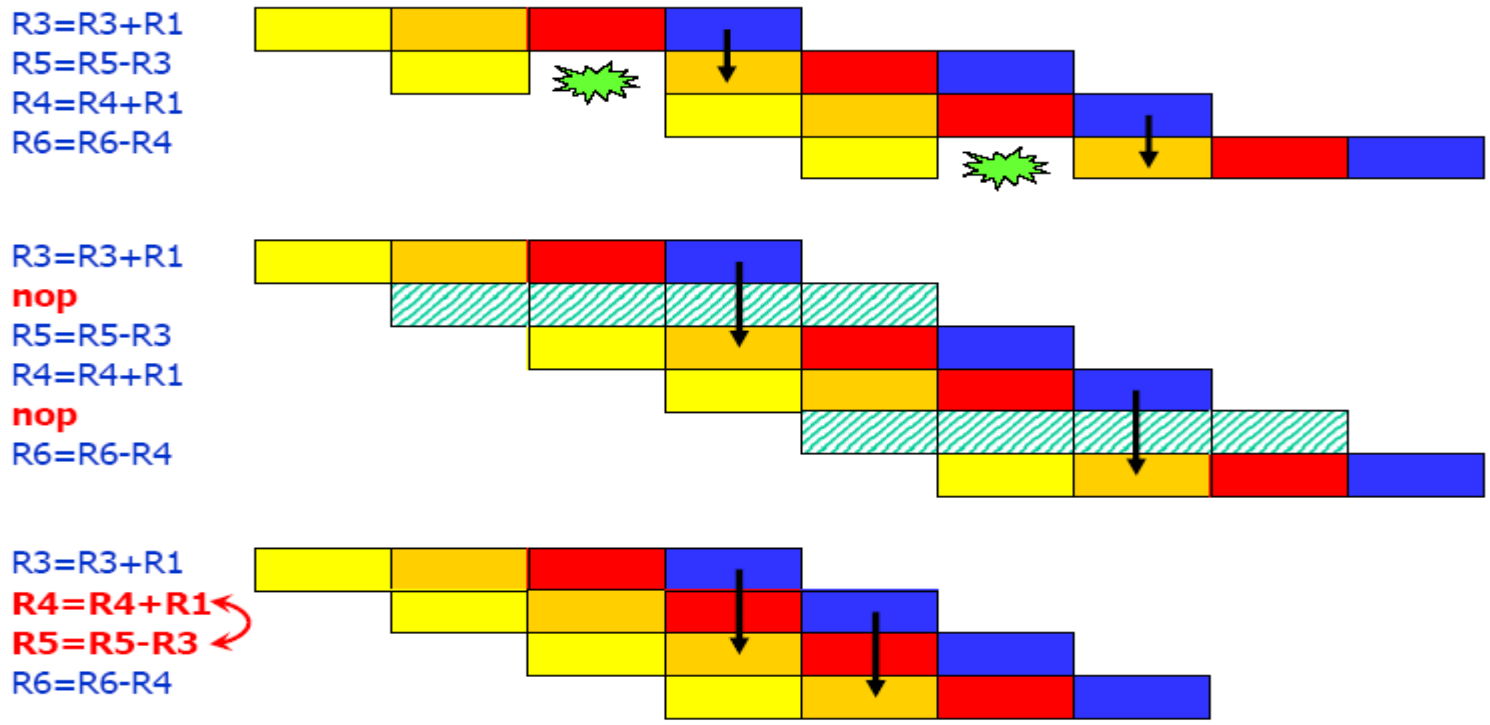
R2 := R1 + R2
R1 := R2 + R3

WAW (Write After Write)

R2 := R1 + R2
R2 := R4 + R3

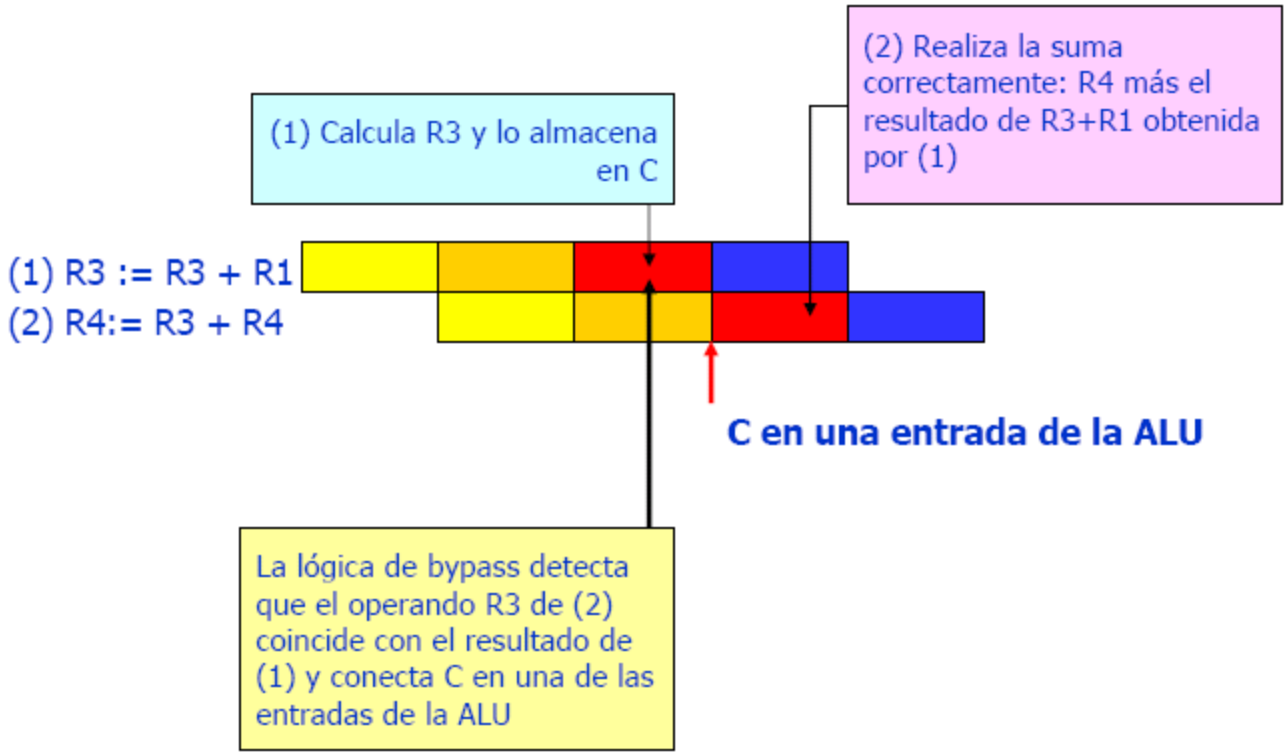
ILP

- Soluciones a los riesgos de datos
  - Reorganización de código (intercambio de instrucciones e inserción de NOP)



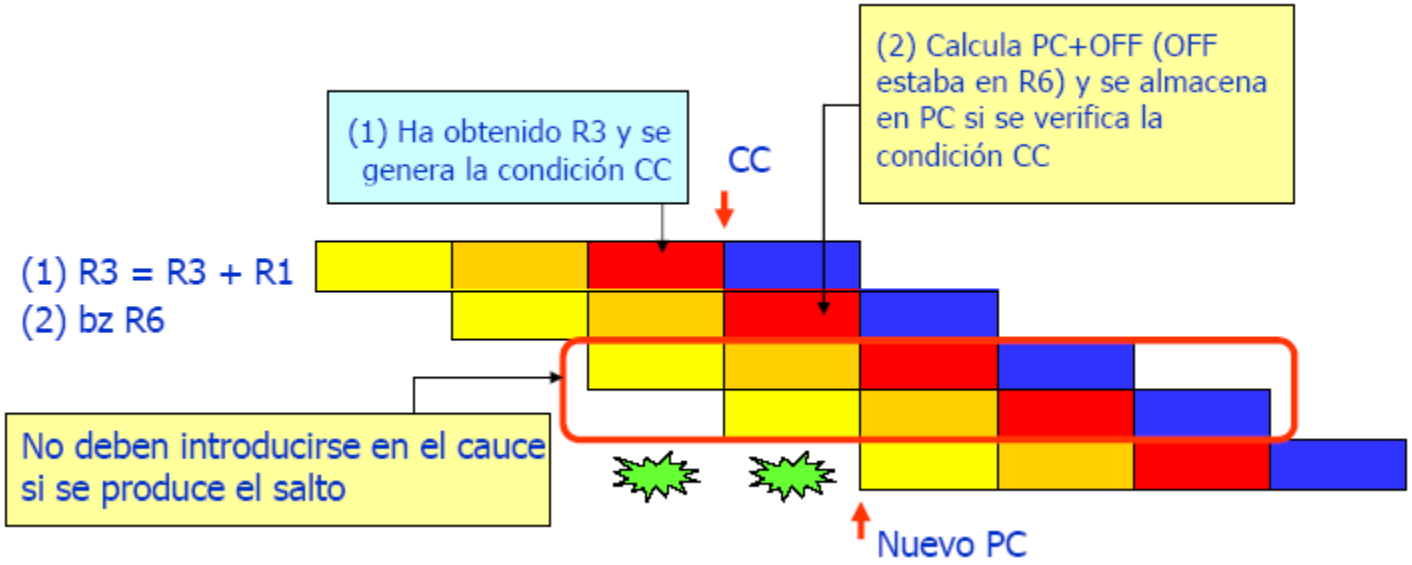
ILP

- Soluciones a los riesgos de datos
  - Forwardings



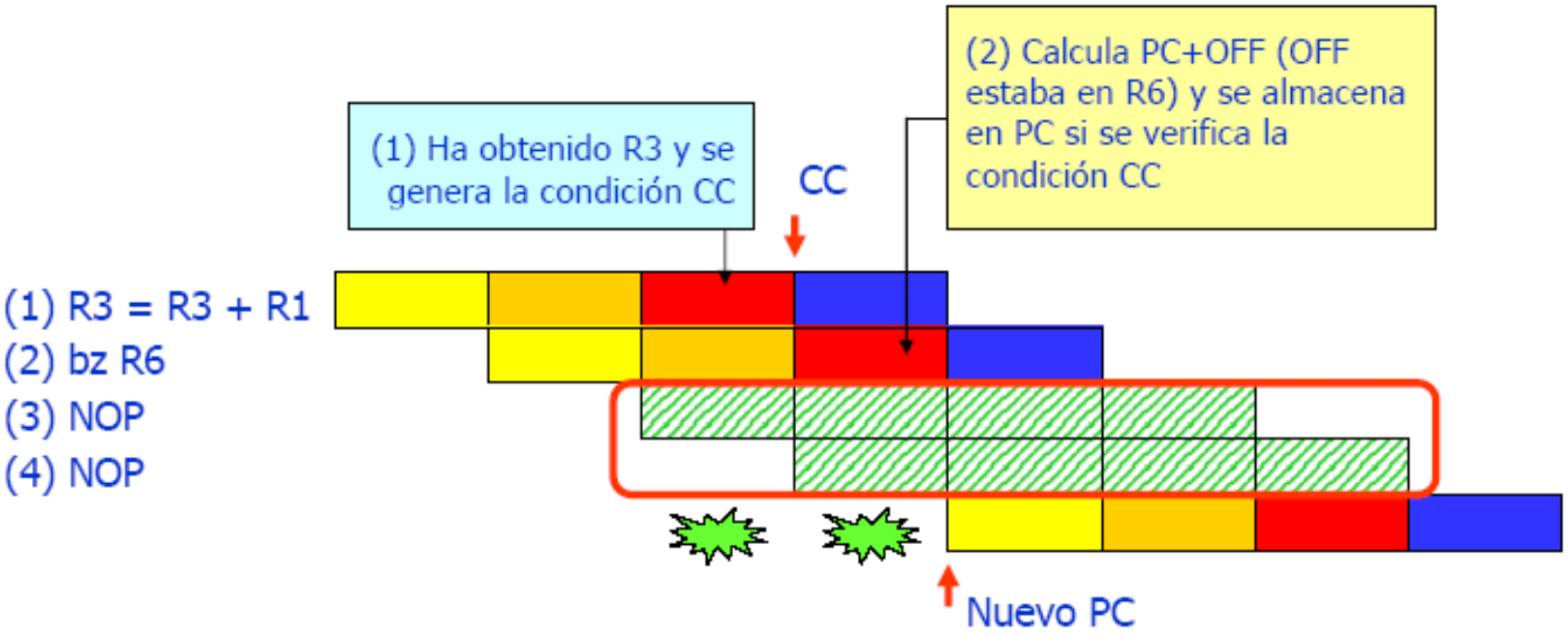
ILP

- Soluciones a los riesgos de control
  - Abortar operaciones



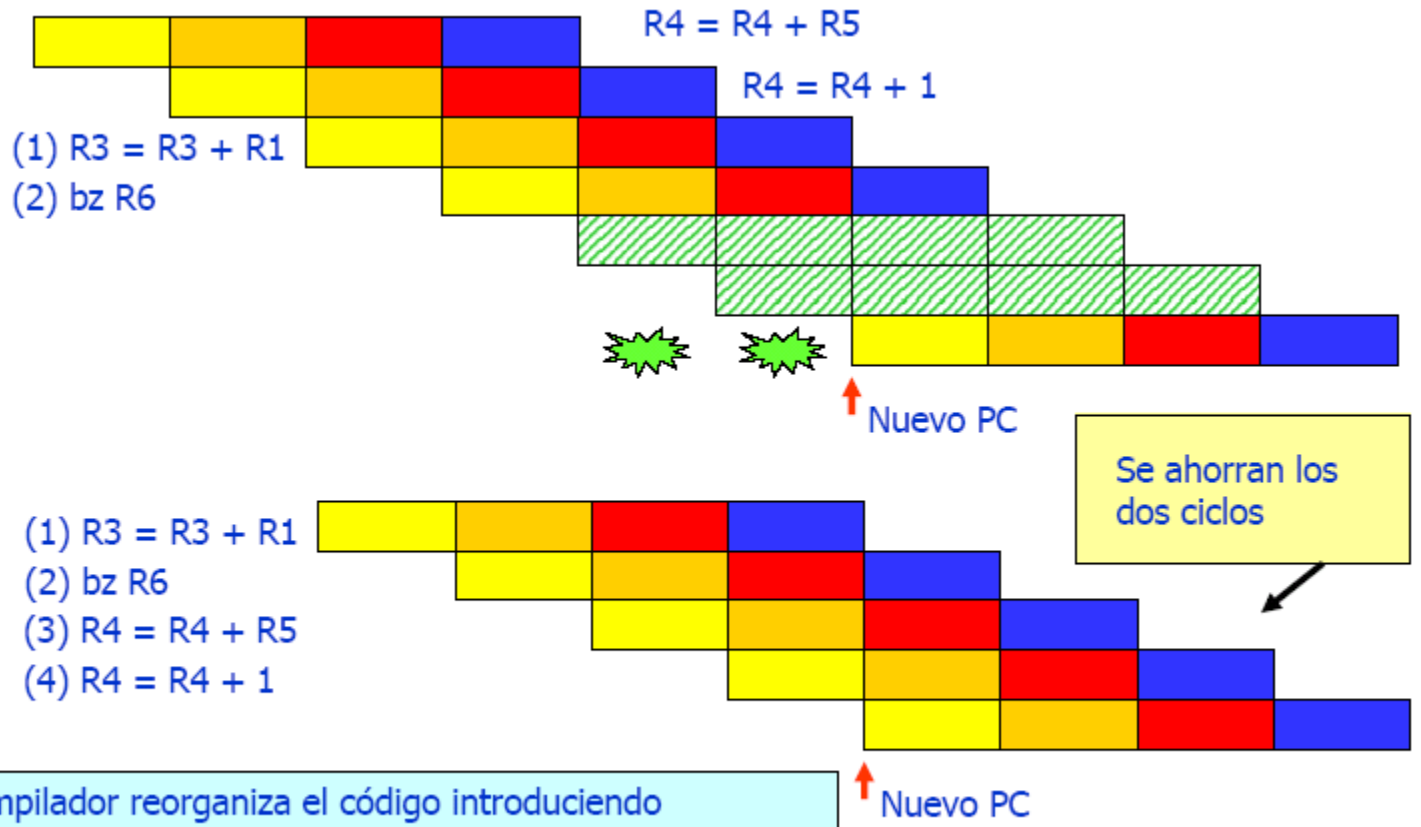
ILP

- Soluciones a los riesgos de control
  - Bloqueos o uso de NOP



ILP

- Soluciones a los riesgos de control
  - Delayed branch



El compilador reorganiza el código introduciendo instrucciones anteriores a la de salto después de ella (debe comprobarse que el programa se ejecuta correctamente)

Rendimiento

- Tiempo de ejecución de un programa
  - Tiempo de CPU (usuario y sistema)
  - Tiempo de E/S (comunicaciones, acceso a memoria, visualización, etc.)

$$\text{Tiempo de CPU } (T_{CPU}) = \text{Ciclos\_del\_Programa} \times T_{CICLO} = \frac{\text{Ciclos\_del\_Programa}}{\text{Frecuencia\_de\_Reloj}}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos\_del\_Programa}}{\text{Número\_de\_Instrucciones (NI)}}$$

$$T_{CPU} = NI \times CPI \times T_{CICLO}$$

$T_{CICLO}=1/F$

F=frecuencia

Rendimiento

- Tiempo para arquitecturas capaces de emitir a ejecución varias instrucciones por unidad de tiempo

$$T_{CPU} = NI \times (CPE / IPE) \times T_{CICLO}$$

CPI

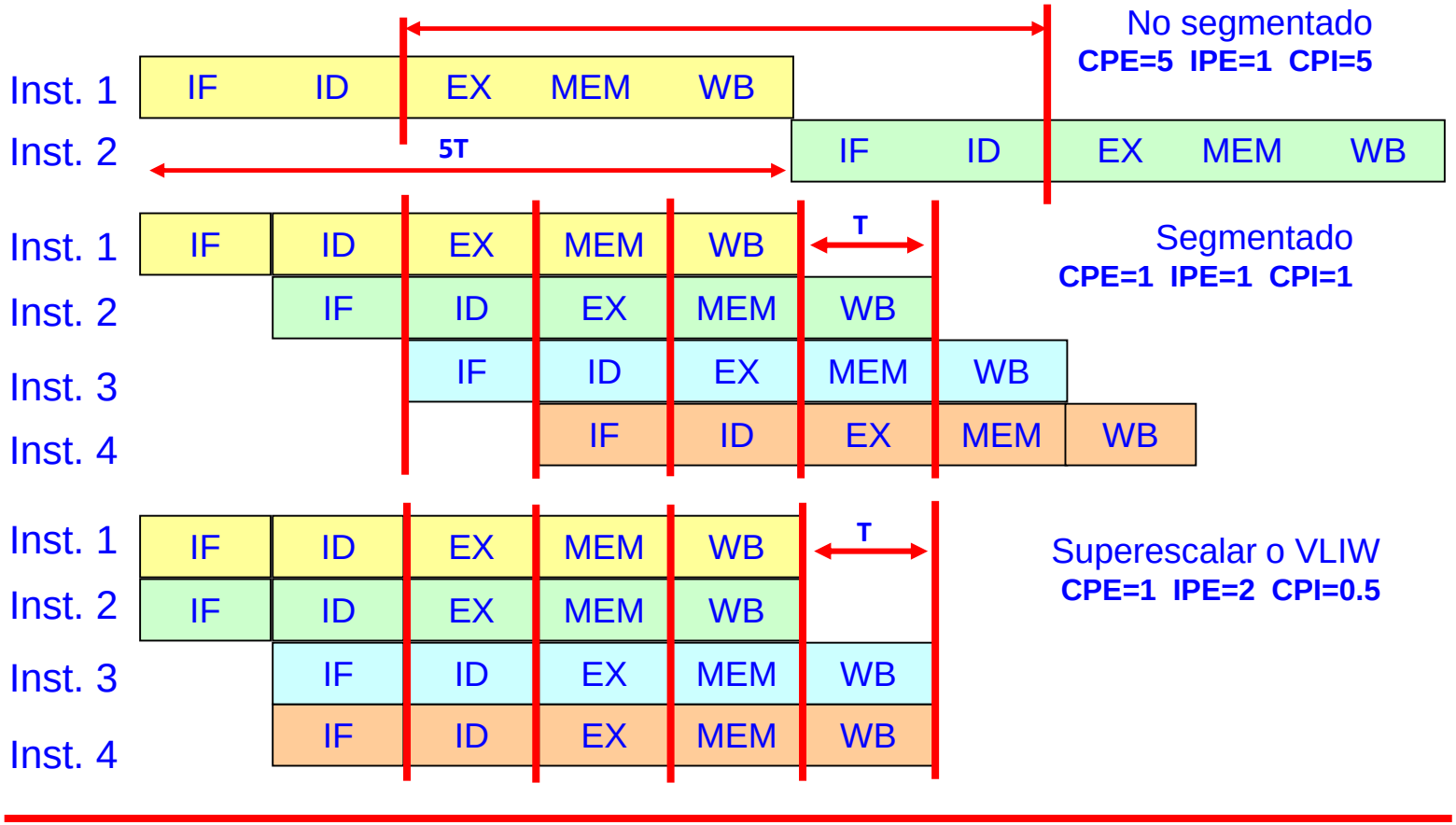
CPE = ciclos entre inicio de emisión de instrucciones.

IPE = instrucciones que pueden emitirse (empezar la ejecución) cada vez que se produce ésta.

Rendimiento

Ejemplo:

CPI = CPE/IPE



Rendimiento

- Procesadores que codifican varias operaciones en una instrucción (VLIW)

$$T_{CPU} = \underbrace{(Noper / Op\_instr)}_{NI} \times CPI \times T_{CICLO}$$

Noper = número de operaciones que realiza el programa.

Op_instr = número de operaciones que puede codificar una instrucción.

Rendimiento

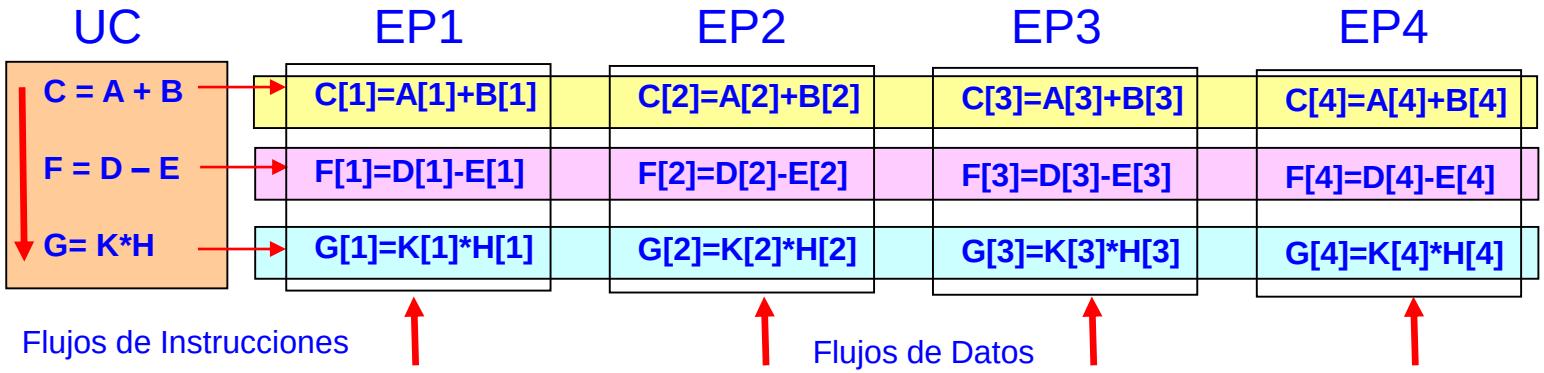
Ejemplo:

NI = Noper/Op_instr

Ejemplo paralelismo datos

Procesador Matricial

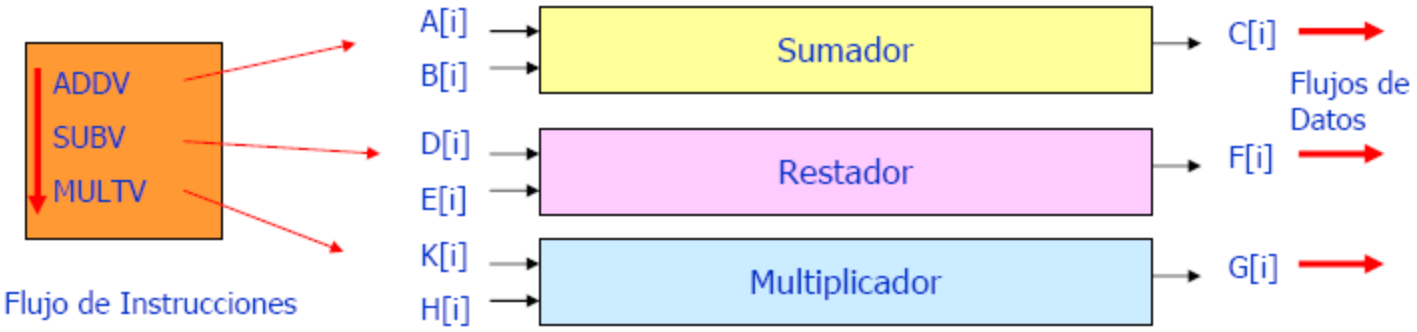
Noper=12 Op_instr=4 NI=3



Ejemplo paralelismo instrucciones

Procesador Vectorial

Noper=12 Op_instr=4 NI=3



Rendimiento

- Medidas de rendimiento:
  - Ganancia

$$G_P = \frac{T_1}{T_P}; \qquad G_P \leq P$$

  - Eficiencia

$$E_P = \frac{G_p}{P}; \qquad E_P \leq 1$$

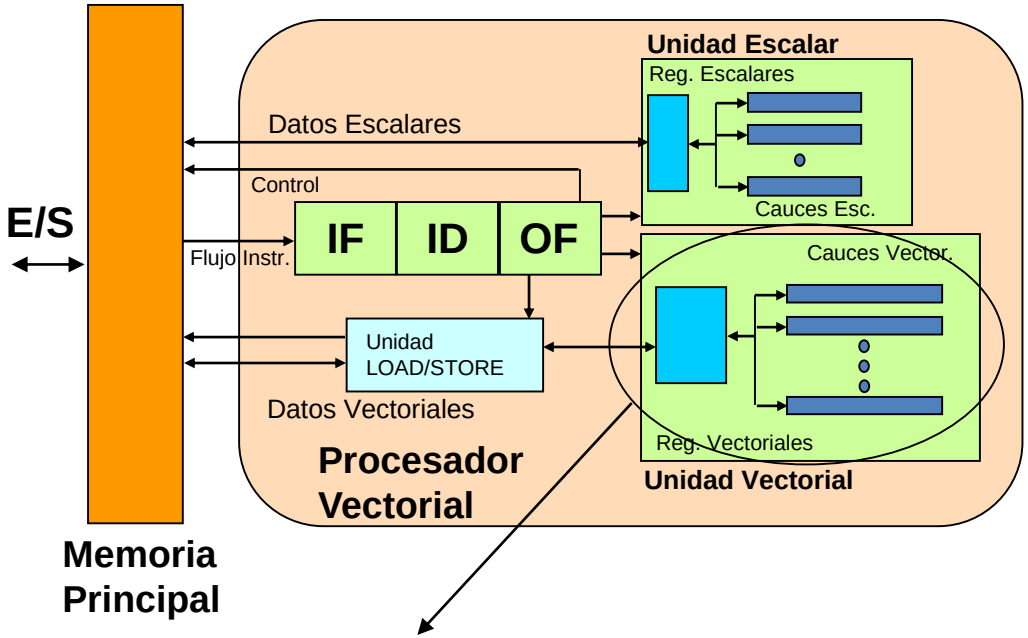
  - Productividad

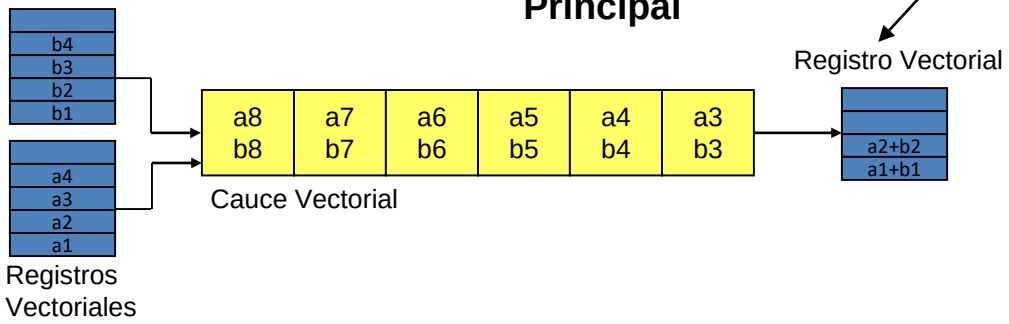
Vectoriales

- Arquitecturas vectoriales: ILP y paralelismo de datos

El procesamiento de instrucciones está segmentado y se utilizan múltiples unidades funcionales.

Paralelismo de datos: cada instrucción vectorial codifica una operación sobre todos los componentes del vector.





Unidades funcionales segmentadas

Vectoriales

- Arquitectura orientada al procesamiento de vectores (suma de vectores, productos escalares, etc.)
- Repertorio de instrucciones especializado
- Características
  - Cálculo de los componentes del vector de forma independiente (buenos rendimientos)
  - Cada operación vectorial codifica gran cantidad de cálculos (se reduce el número de instrucciones y se evitan riesgos de control)
  - Se optimiza el uso de memoria (entrelazado de memoria y organizaciones S y C)

Vectoriales

Ejemplo: Sumar dos vectores de 100 elementos.

Pseudo-código escalar

```
for i:= 1 to 100 do c(i)=b(i)+a(i)
```

Ensamblador escalar (con bucle de 100 iteraciones)

```
LOADI R5, BASEa
LOADI R6, BASEb
LOADI R7, BASEc
LOADI R1, 0
INI ADDRI R5, R5, 1
    ADDRI R6, R6, 1
    ADDRI R7, R7, 1
    ADDMR R8, R5, R6
    STORE R7, R8
    INC R1
   COMP R1, 100
   JUMP NOT.EQUAL INI
```

Pseudo-código vectorial

```
c(1:100:1) = a(1:100:1) + b(1:100:1)
```

Ensamblador vectorial

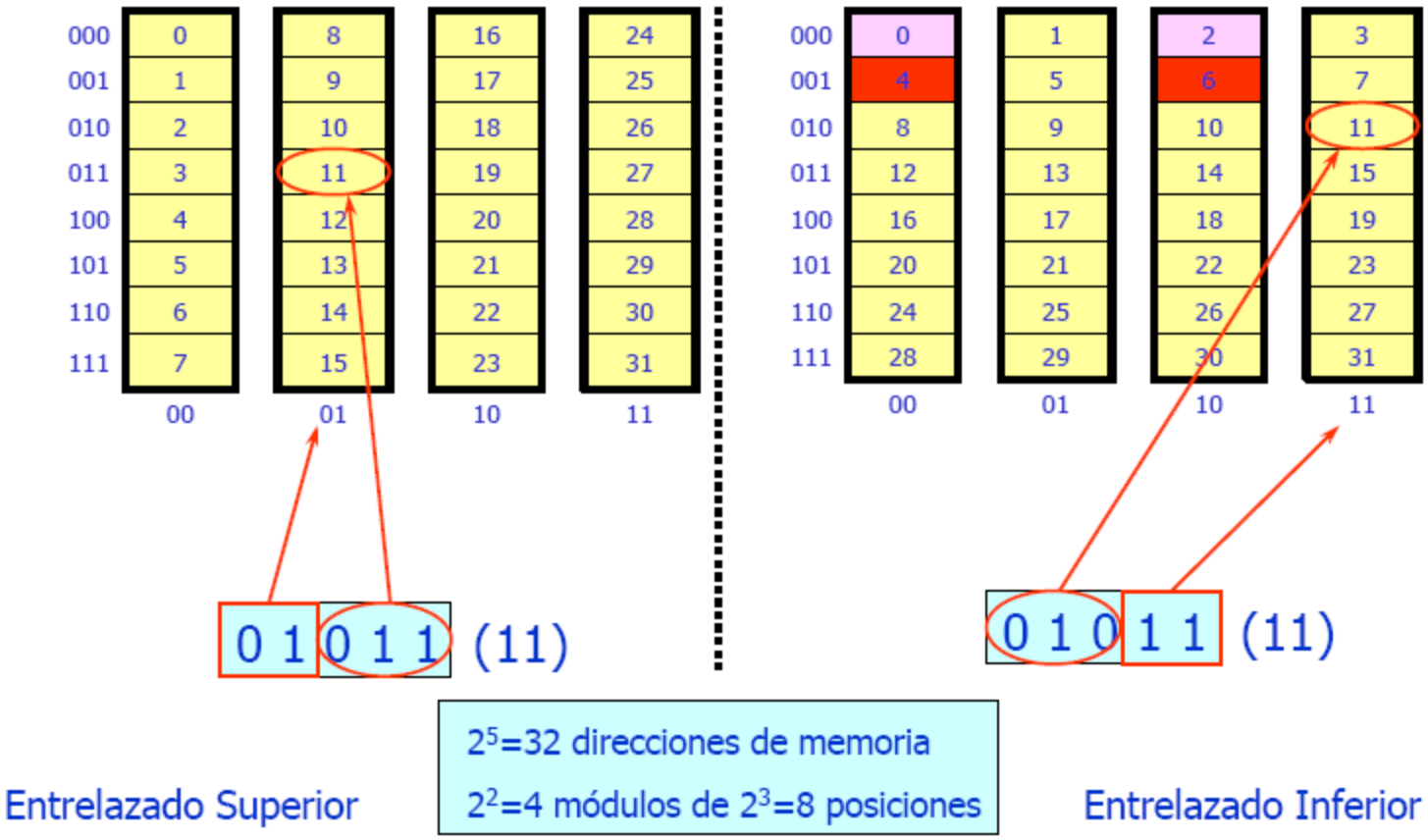
```
ADDV c, a, b, 1, 100
```

Vectoriales

Vector instruction	Operands	Function
ADDV	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDSV	V1,F0,V2	Add F0 to each element of V2, then put each result in V1.
SUBV	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULTV	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULTSV	V1,F0,V2	Multiply F0 by each element of V2, then put each result in V1.
DIVV	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i*R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i*R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.
SVI	(R1+V2),V1	Store V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0,1*R1,2*R1,...,63*R1 into V1.
S_V S_SV	V1,V2 F0,V1	Compare (EQ, NE, GT, LT, GE, LE) the elements in V1 and V2. If condition is true put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S_SV performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MOVI2S	VLR,R1	Move contents of R1 to the vector-length register.
MOVS2I	R1,VLR	Move the contents of the vector-length register to R1.
MOVF2S	VM,F0	Move contents of F0 to the vector-mask register.
MOVS2F	F0,VM	Move contents of vector-mask register to F0.

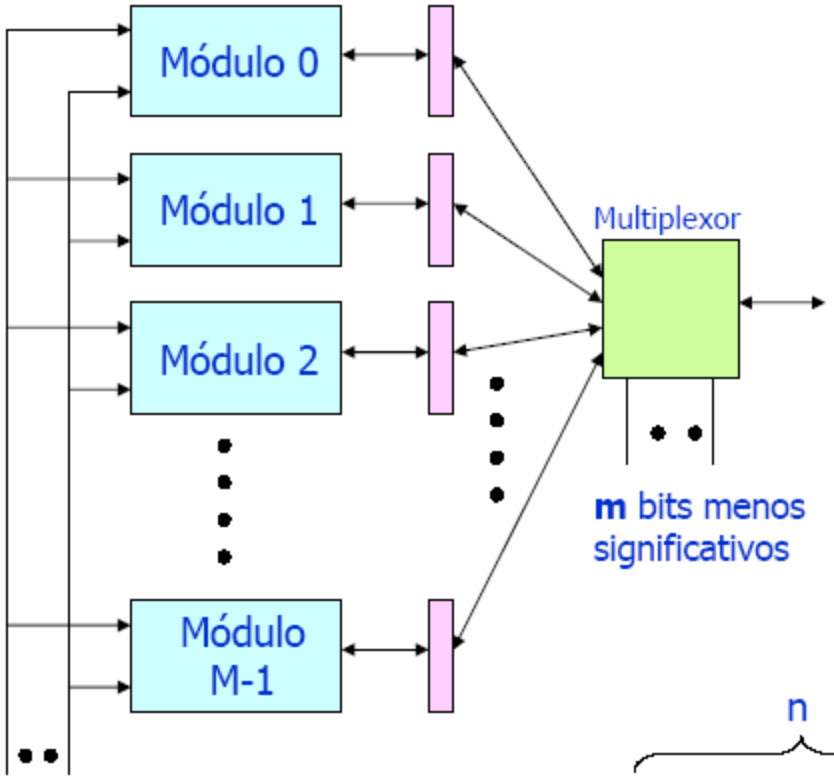
Vectoriales

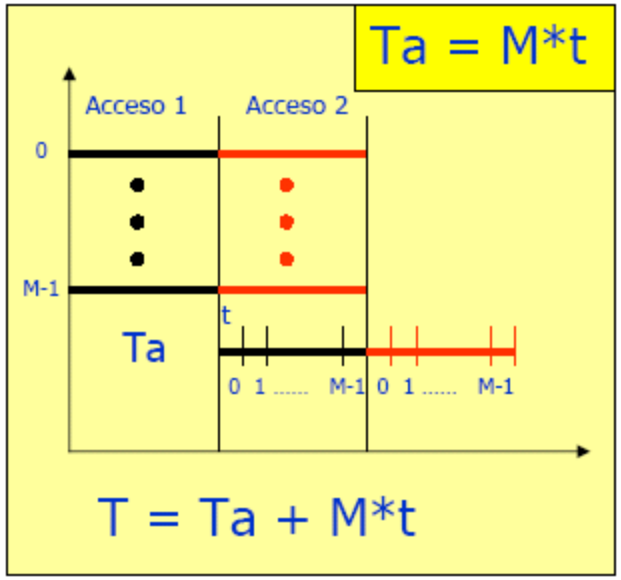
- Entrelazado de memoria

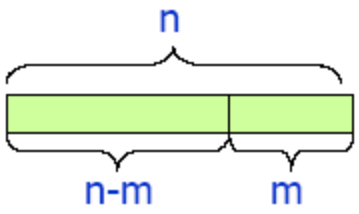


Vectoriales

- Acceso a memoria simultáneo o tipo S







Con Entrelazado Inferior

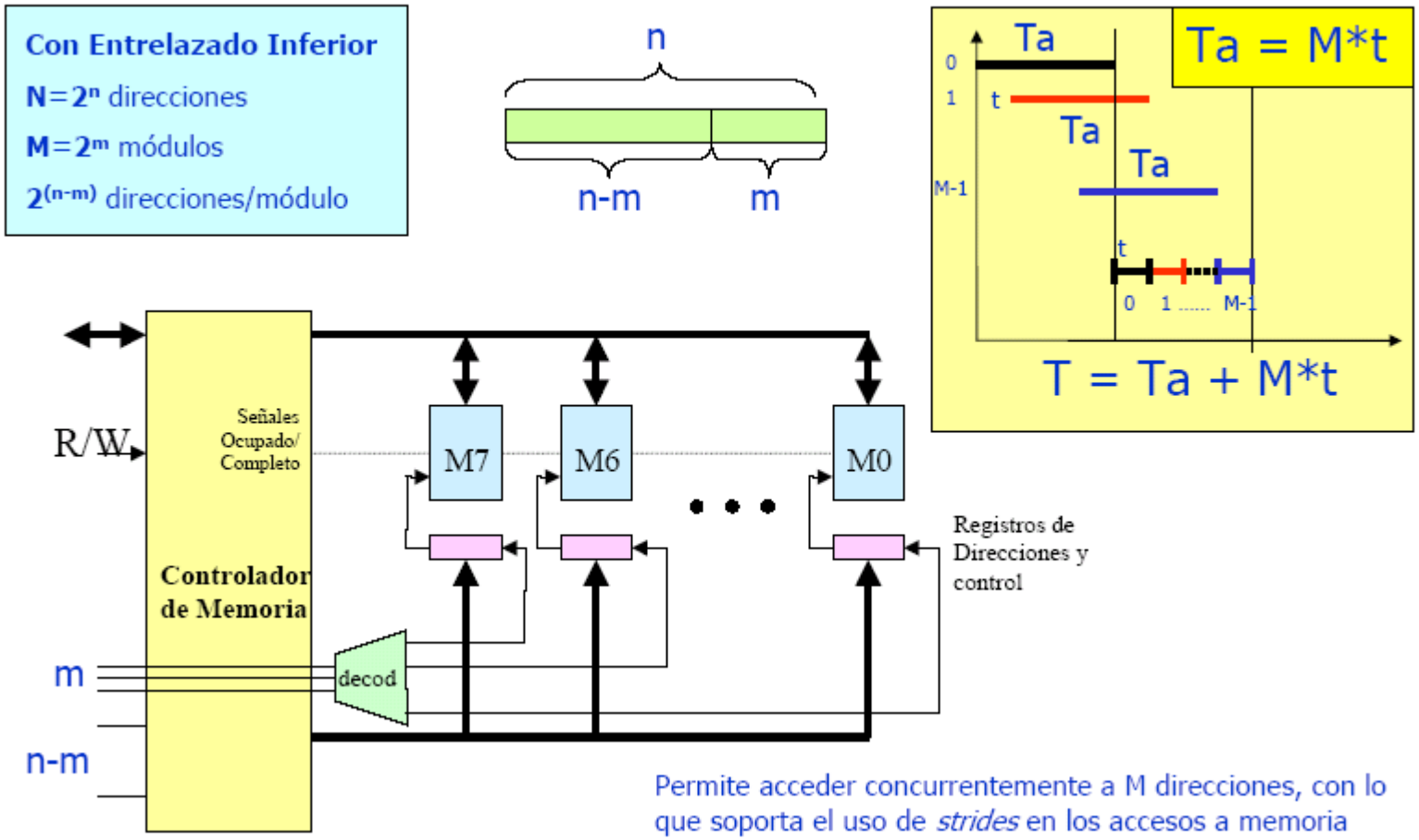
$N = 2^n$ direcciones

$M = 2^m$ módulos

$2^{(n-m)}$ direcciones/módulo

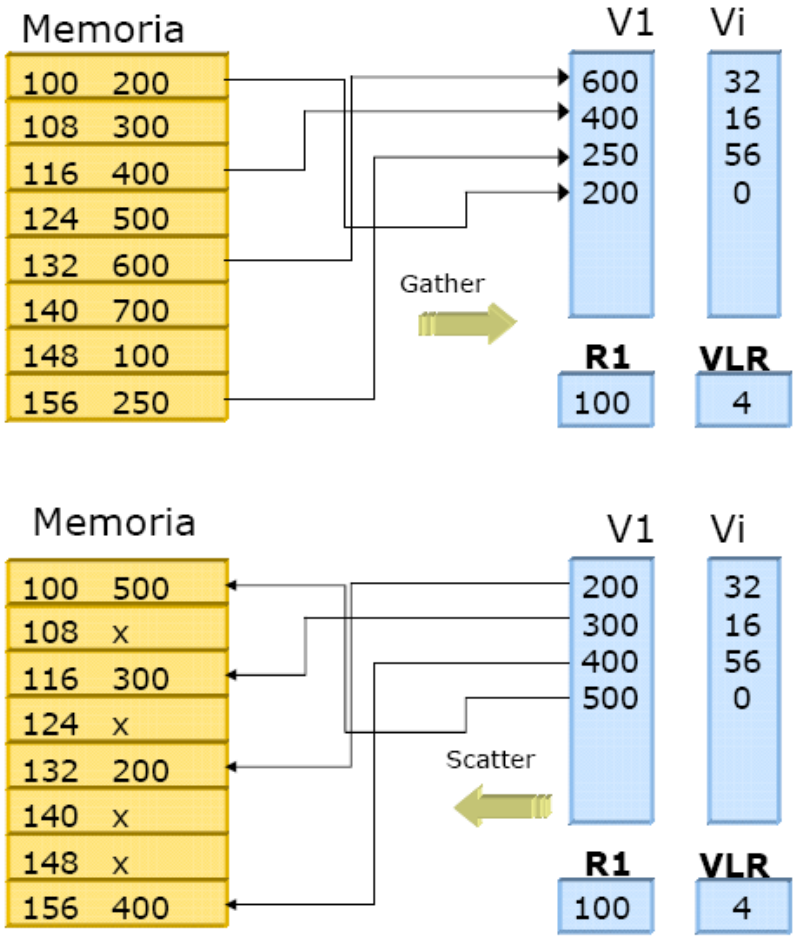
Vectoriales

- Acceso a memoria concurrente o tipo C



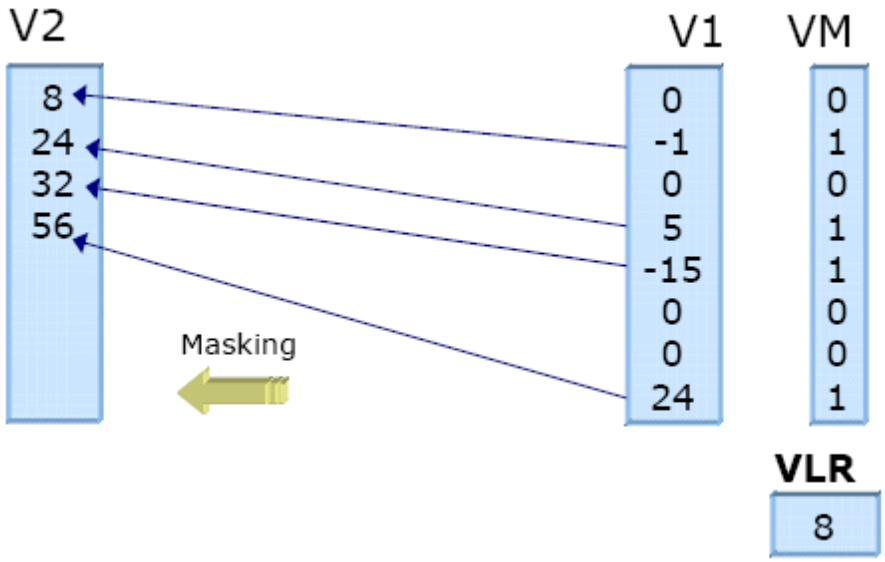
Vectoriales

- Operaciones gather-scatter



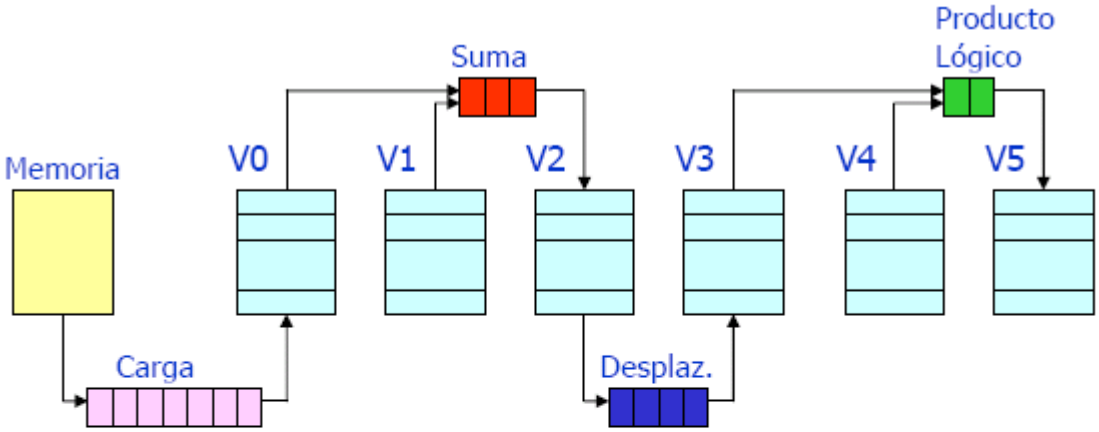
Vectoriales

- Enmascaramiento (gestión de matrices dispersas)



Vectoriales

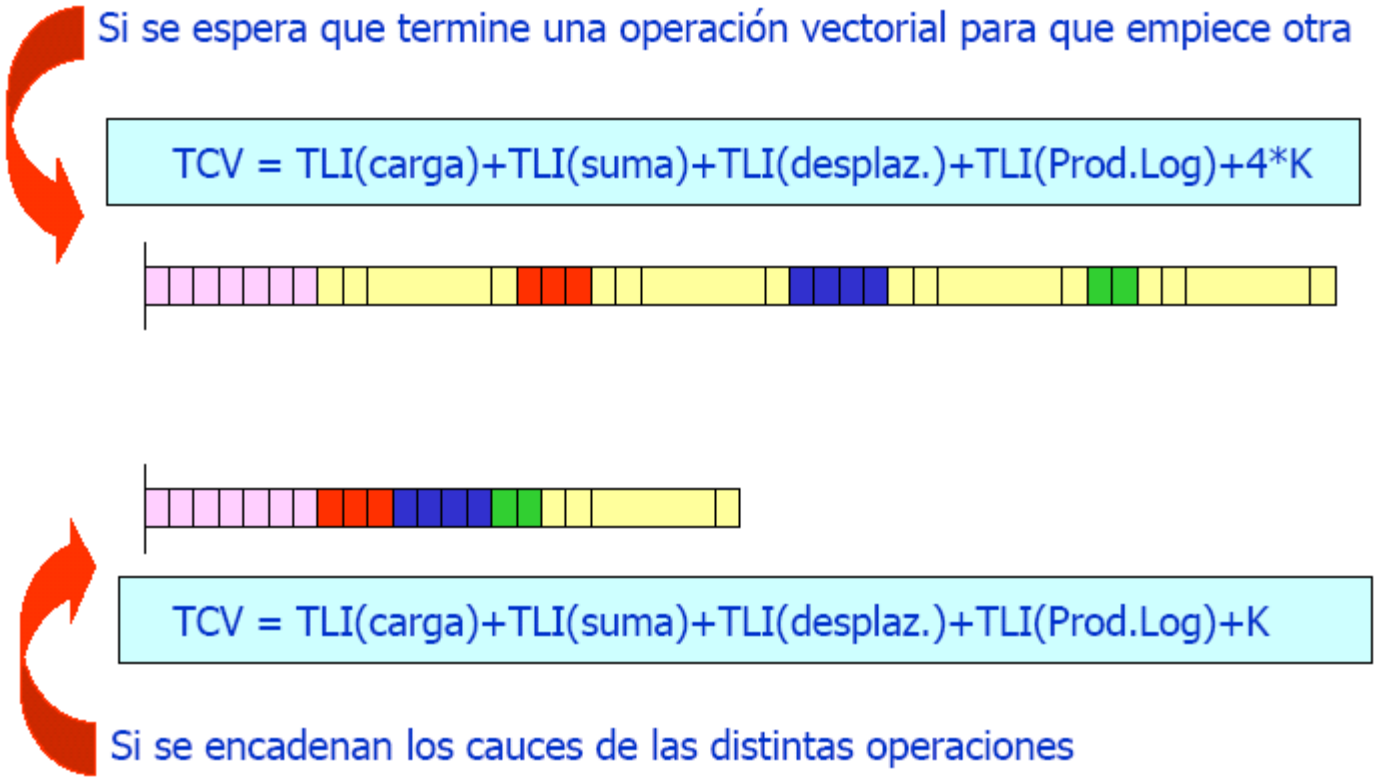
- Rendimiento: encadenamiento de cauce



V0 = Load(Memoria)	(TLI=7)
V2 = V0 + V1	(TLI=3)
V3 = V1 < A3	(TLI=4)
V5 = V3 ^ V4	(TLI=2)

Vectoriales

- Rendimiento: encadenamiento de cauce



Vectoriales