

# **Ingeniería de los Computadores**

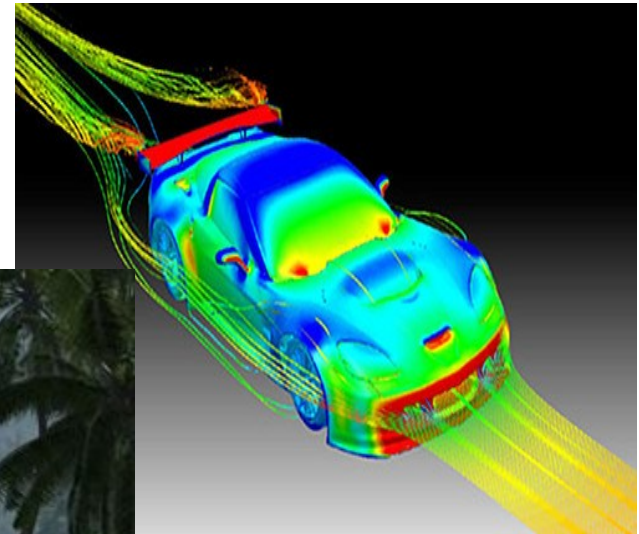
## Sesión 1. Introducción

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

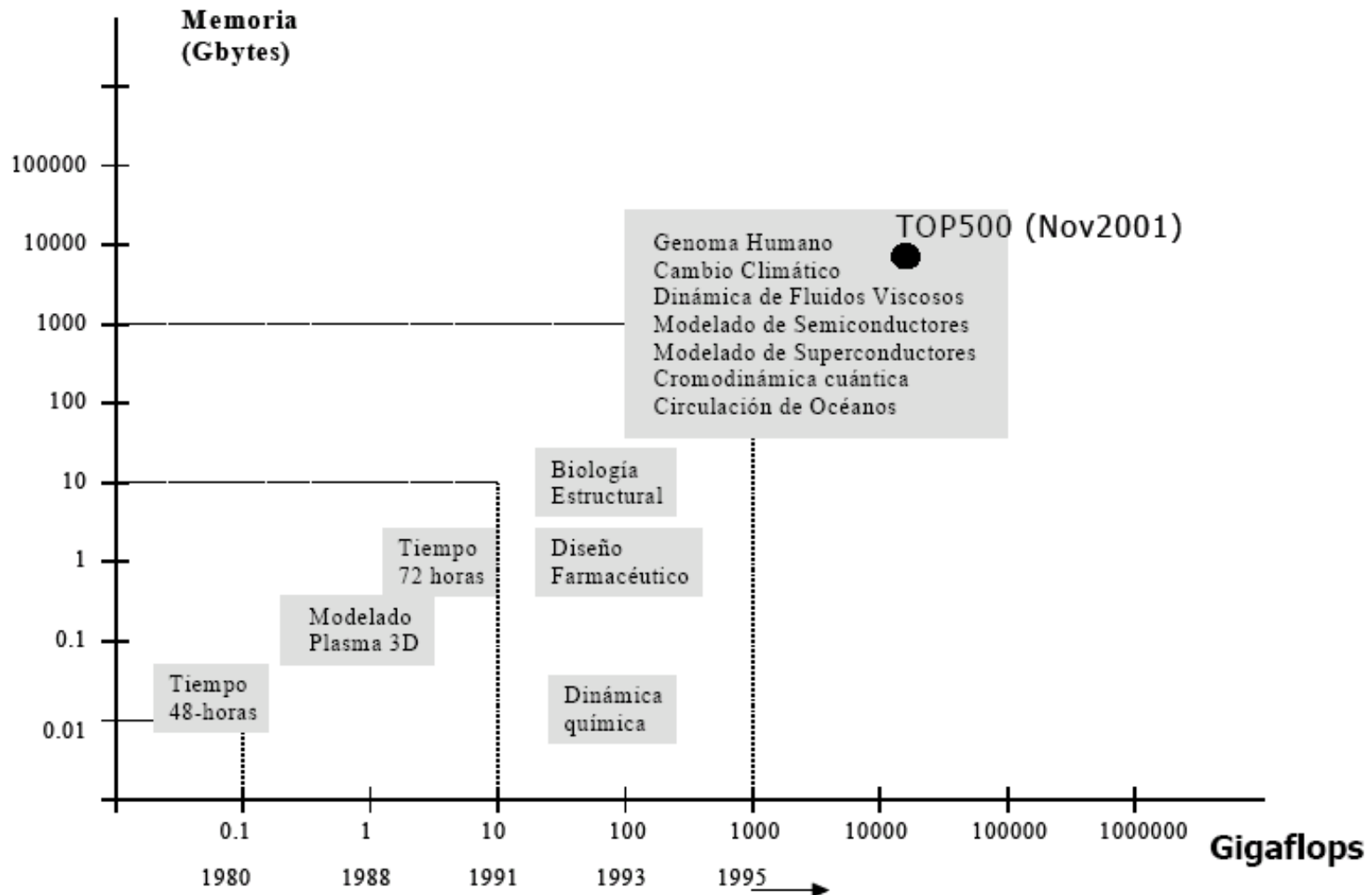
- Evolución de la informática ↔ Necesidades computacionales



# Ingeniería de los Computadores

## Sesión 1. Introducción

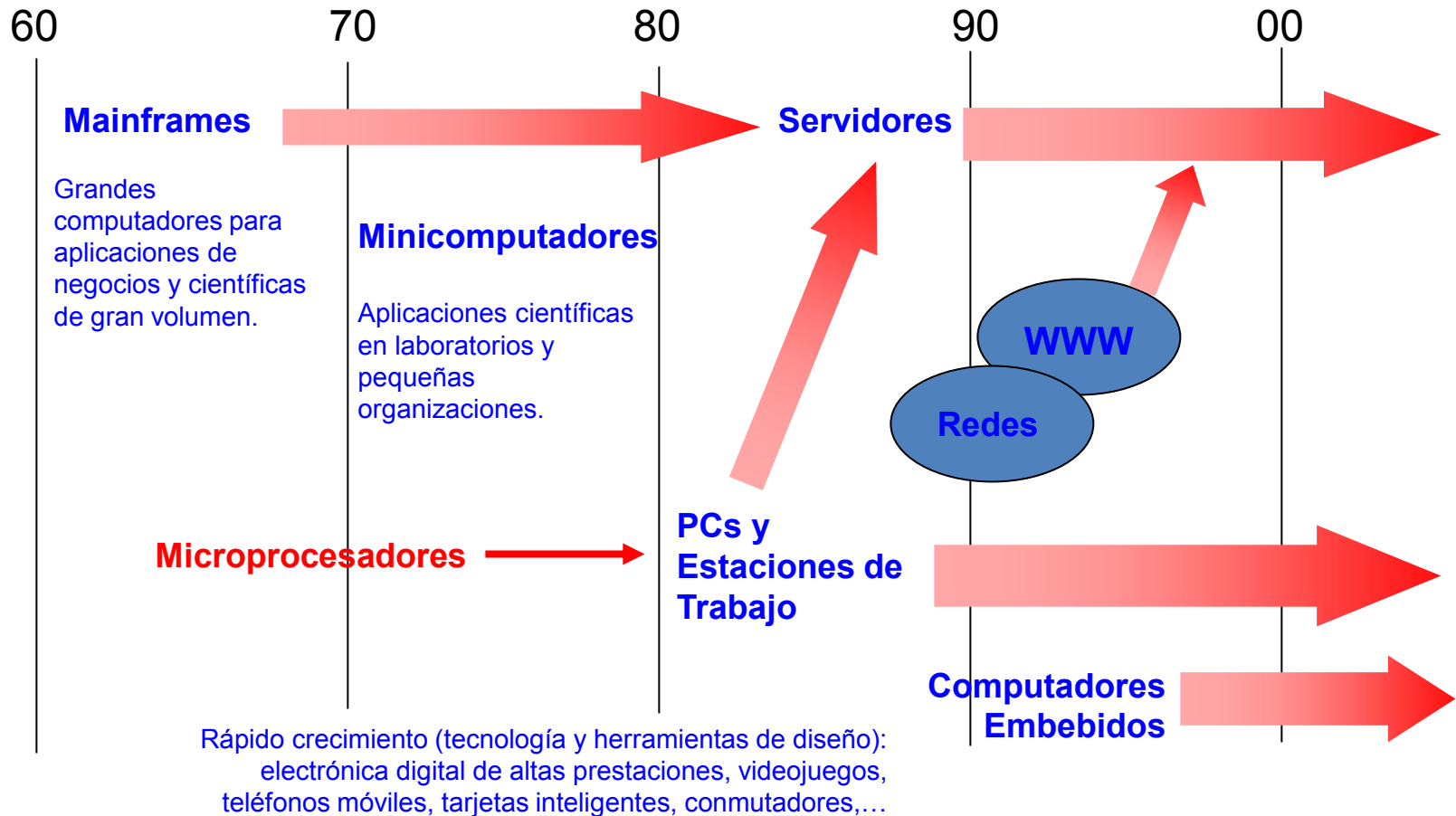
¿Qué?



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

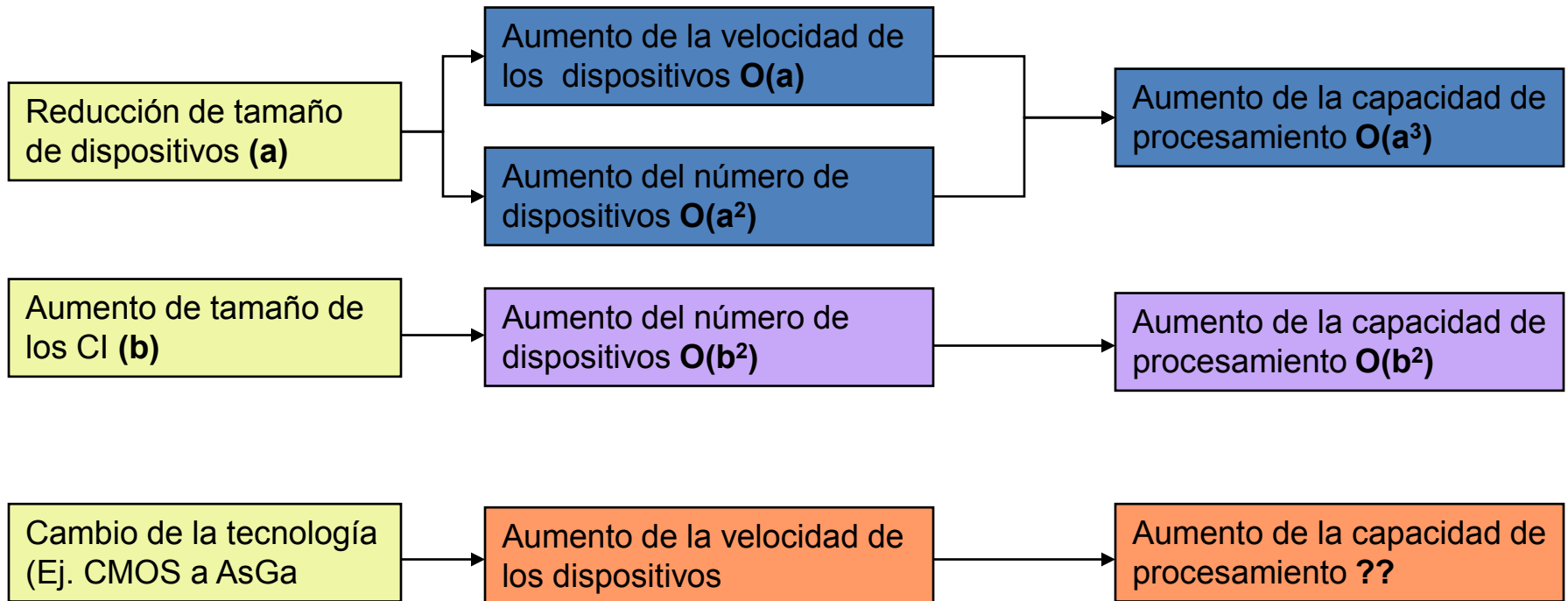


# Ingeniería de los Computadores

## Sesión 1. Introducción

¿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

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

- Niveles de abstracción de un computador

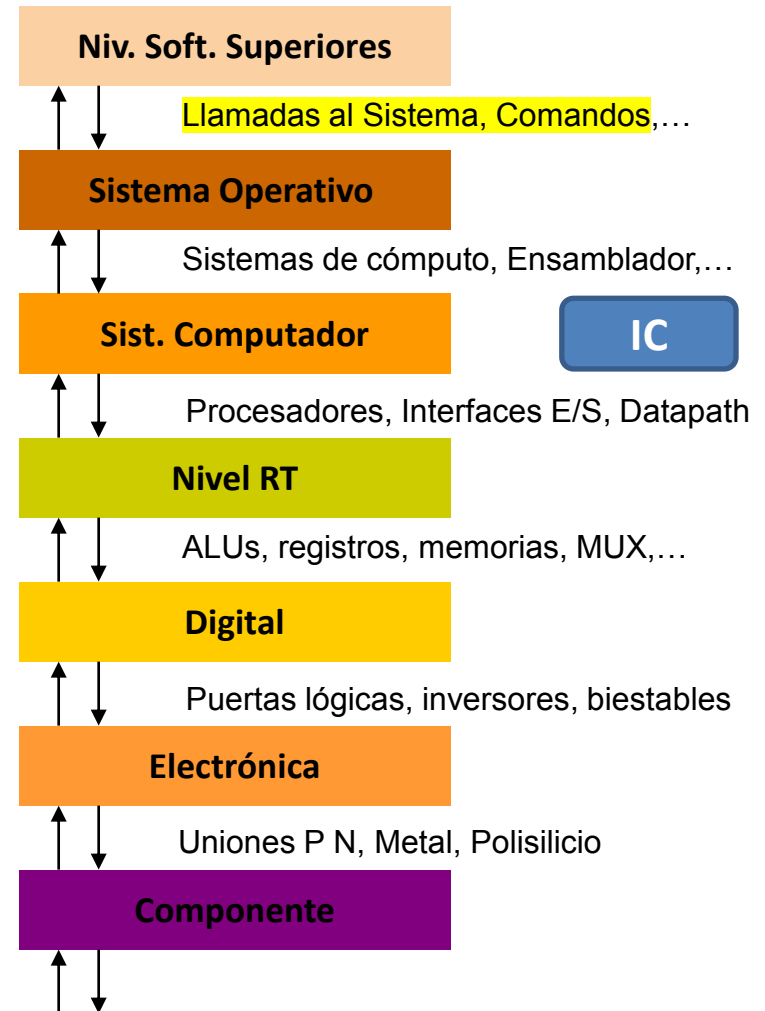
Integra la orientación **estructural** de los niveles de Bell y Newell y el punto de vista **funcional** de los niveles de Levy y Tanenbaum.

SW

ARQUITECTURA

TECNOLOGÍA

HW



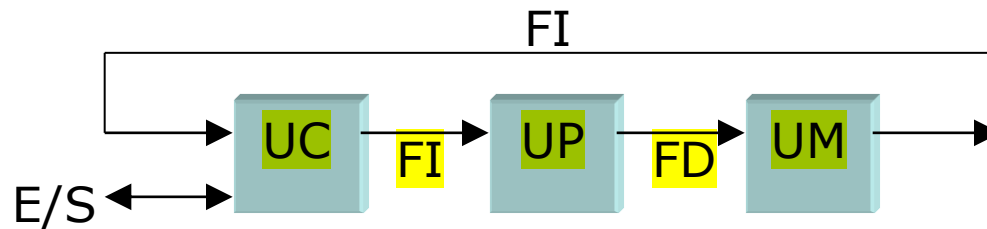
# Ingeniería de los Computadores

## Sesión 1. Introducción

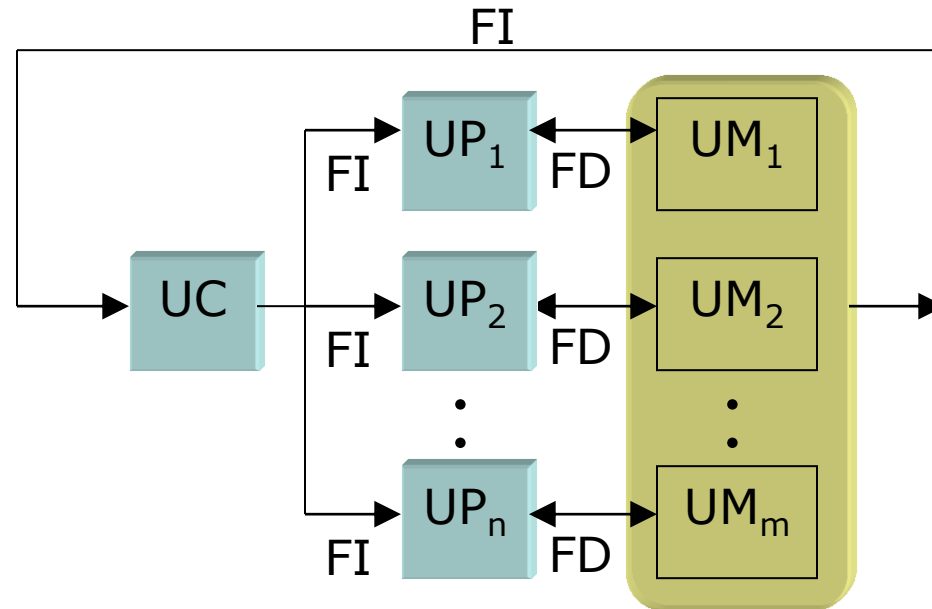
¿Qué?

- Taxonomía de Flynn

➤ SISD



➤ SIMD

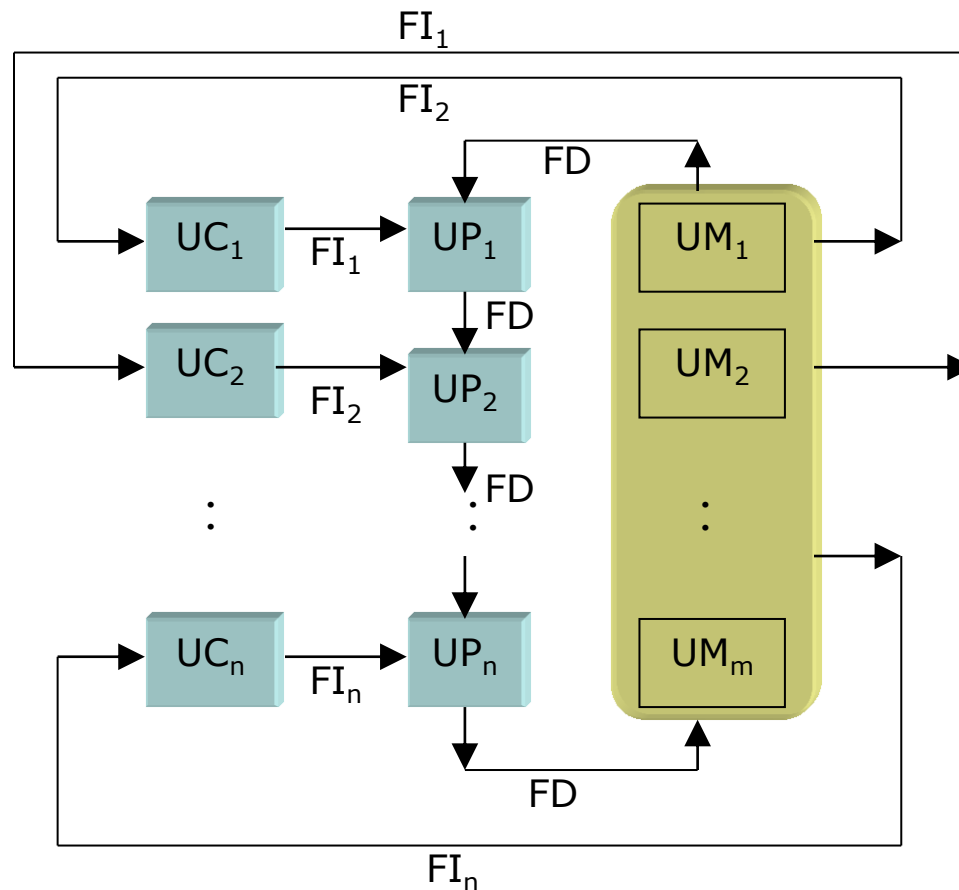


# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn
  - MISD

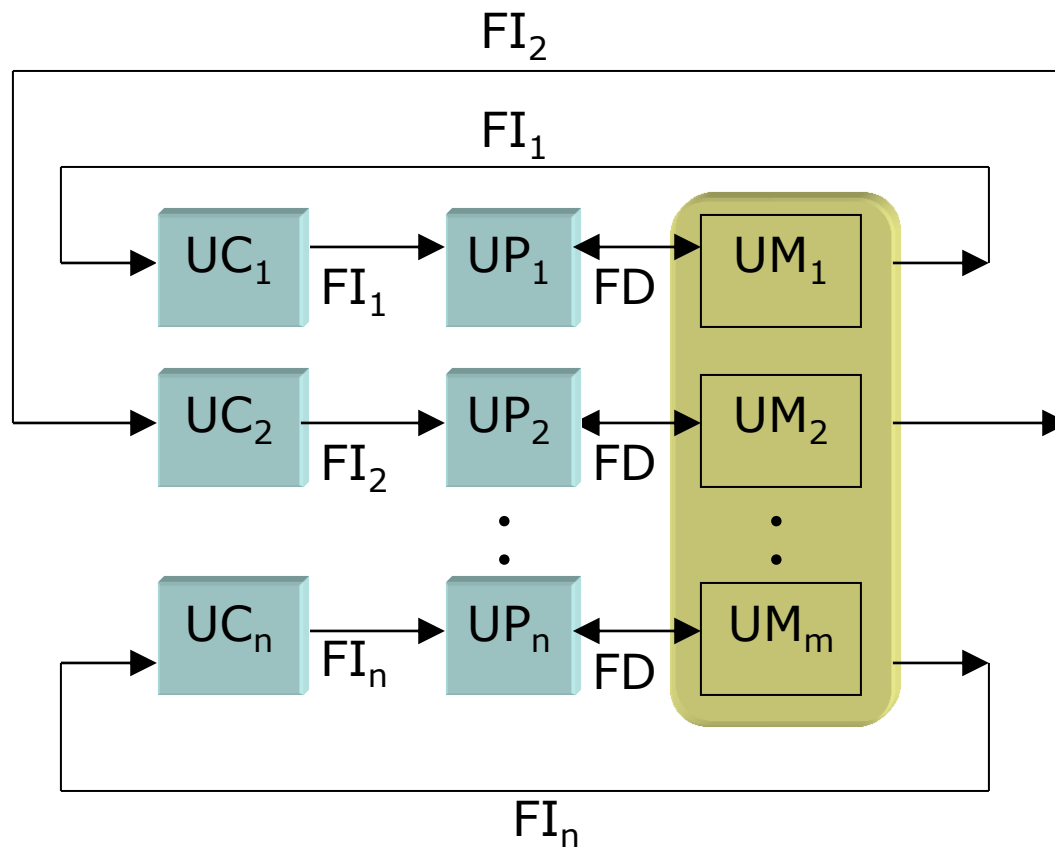


# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn
  - MISD



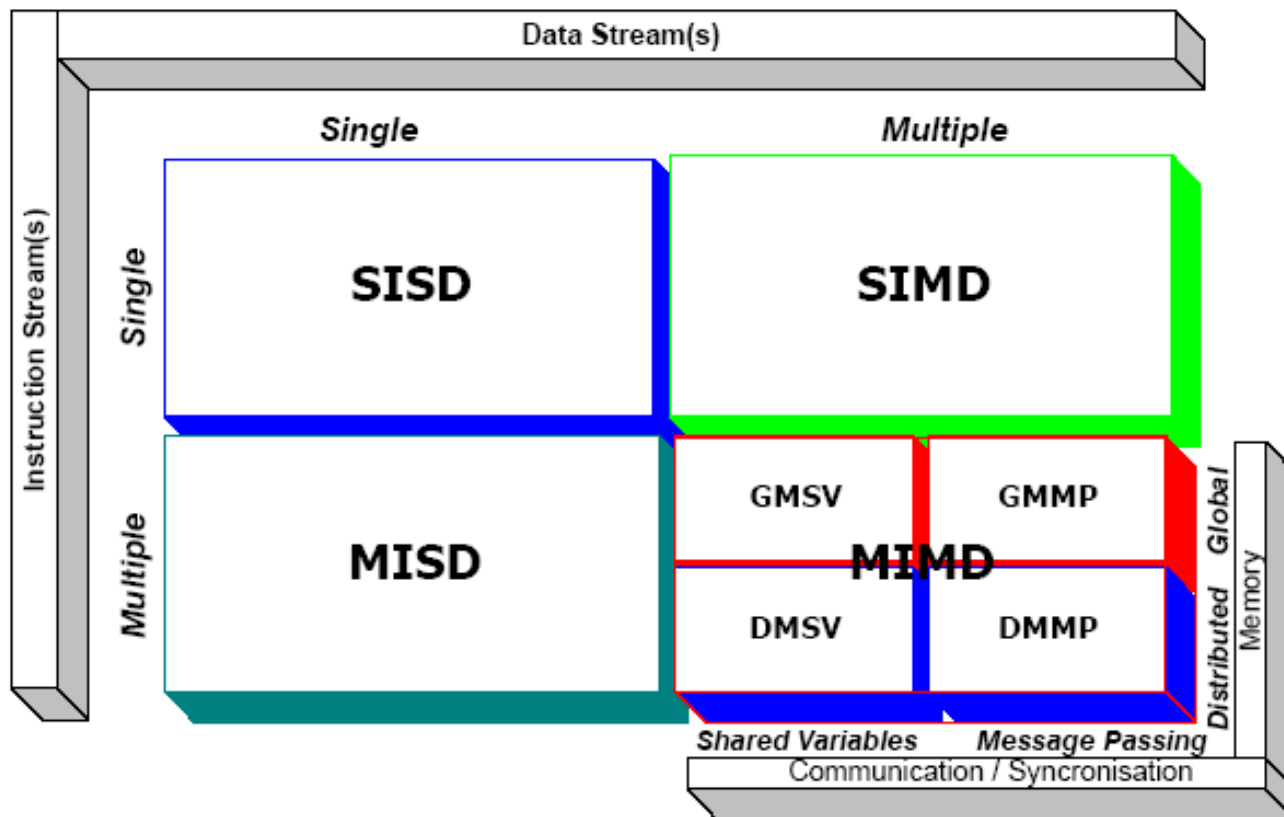


# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

- Taxonomía de Flynn-Johnson



### ¿Qué?

- 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.

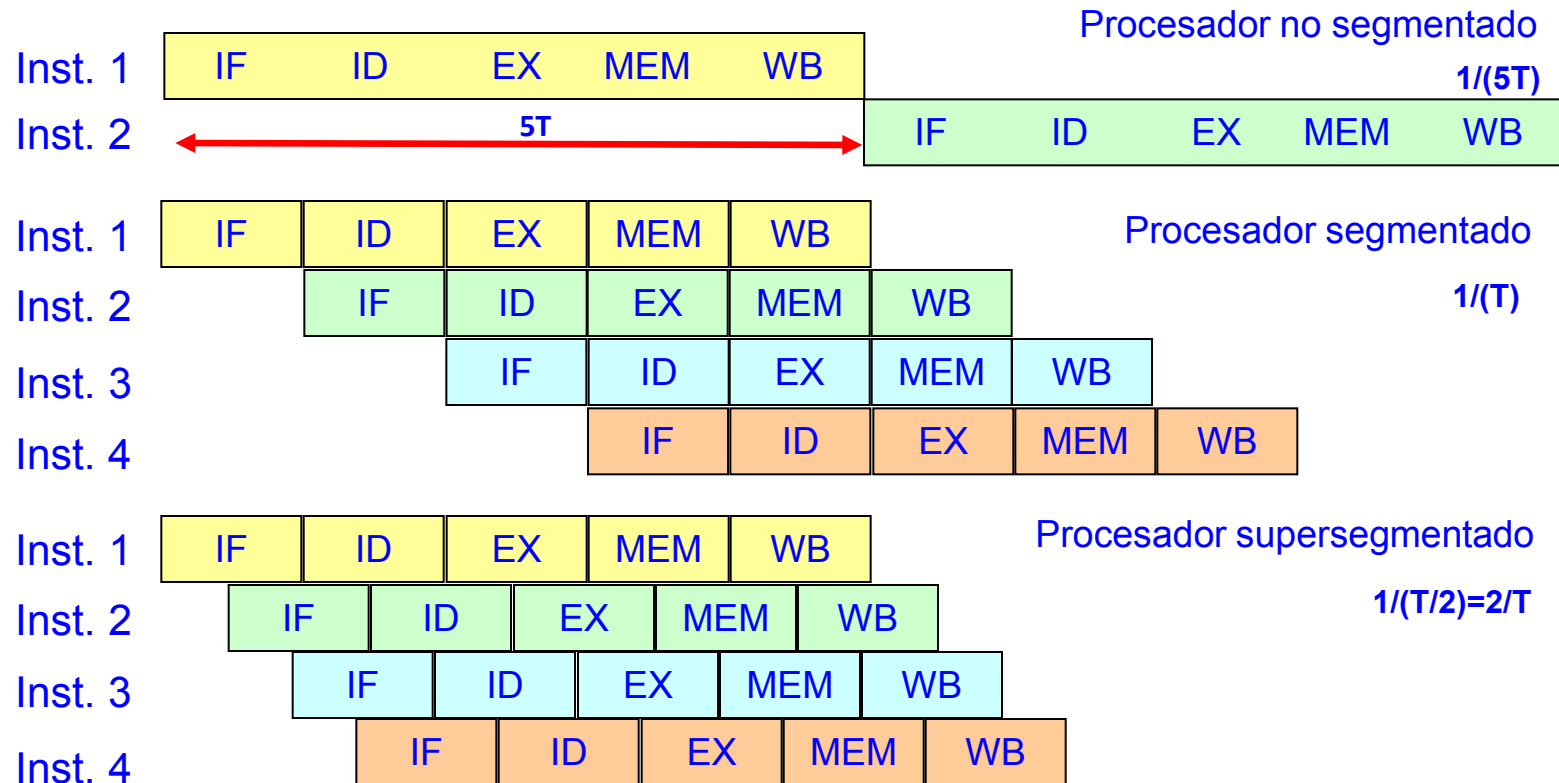
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Segmentación: ILP



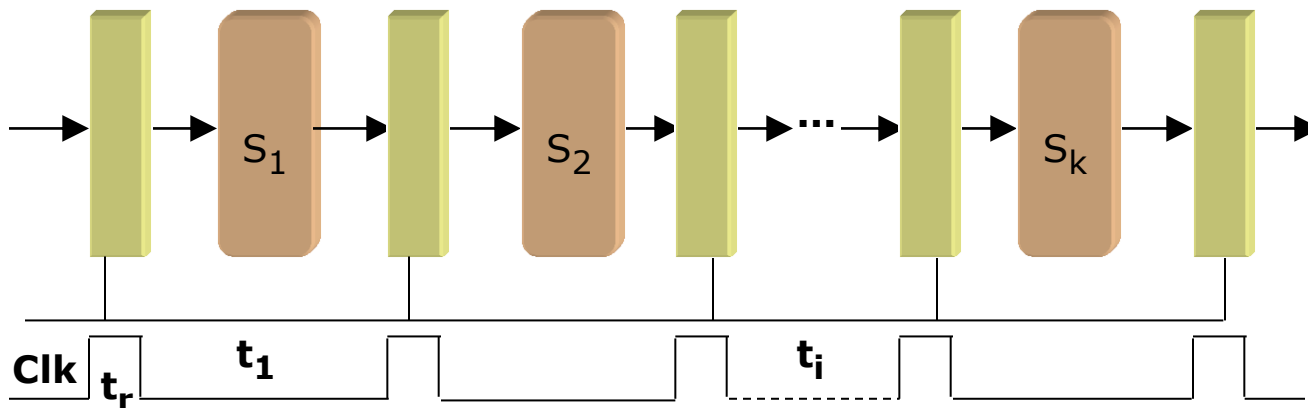
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- 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.



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Segmentación

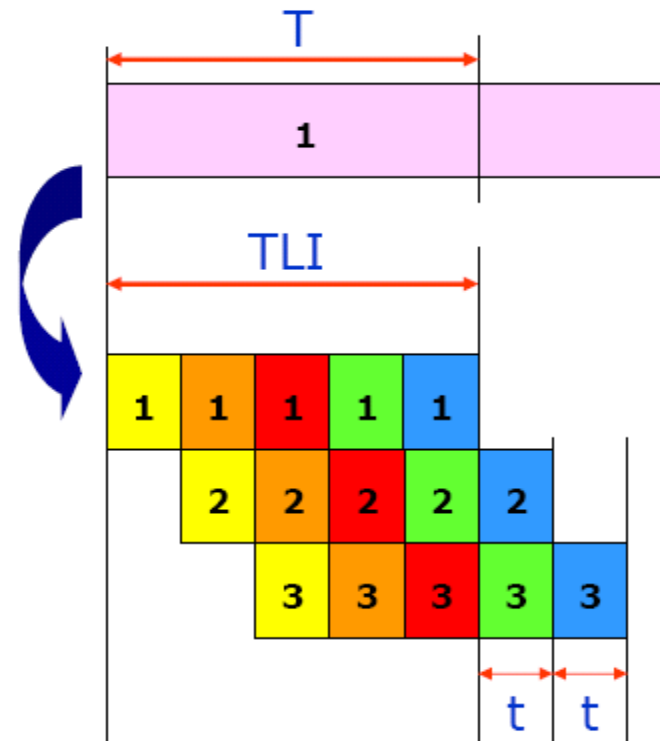
**Ganancia.** Suponemos que TLI (tiempo de latencia de inicio) = T, tiempo que tarda en ejecutarse una operación en una unidad sin segmentar.

TLI =  $k \cdot t$ , siendo  $k$  el nº de etapas del cauce, y  $t$  la duración de cada etapa

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

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

Normalmente,  **$TLI > T$**  ó  $TLI < T$ ?



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Ganancia real

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

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

↑  
 $\text{TLI} = kt$

↑  
 $T < \text{TLI} = kt$

- 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

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Riesgos de datos



RAW (Read After Write)

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

The diagram shows the second instruction `R1 := R2 + R3` with a red arrow pointing from the `R2` operand to the `R2` in the first instruction `R2 := R1 + R2`, indicating that the second instruction depends on the value written by the first.



WAR (Write After Read)

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

The diagram shows the first instruction `R2 := R1 + R2` with a red arrow pointing from the `R1` operand to the `R1` in the second instruction `R1 := R2 + R3`, indicating that the second instruction depends on the value read by the first.



WAW (Write After Write)

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

The diagram shows the second instruction `R2 := R4 + R3` with a red arrow pointing from the `R2` operand to the `R2` in the first instruction `R2 := R1 + R2`, indicating that the second instruction depends on the value written by the first.



# Ingeniería de los Computadores

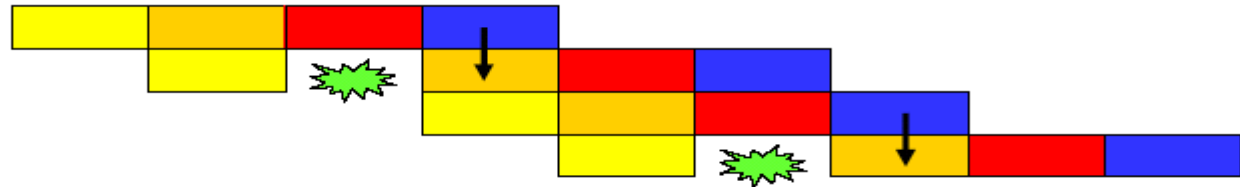
## Sesión 1. Introducción

¿Qué?

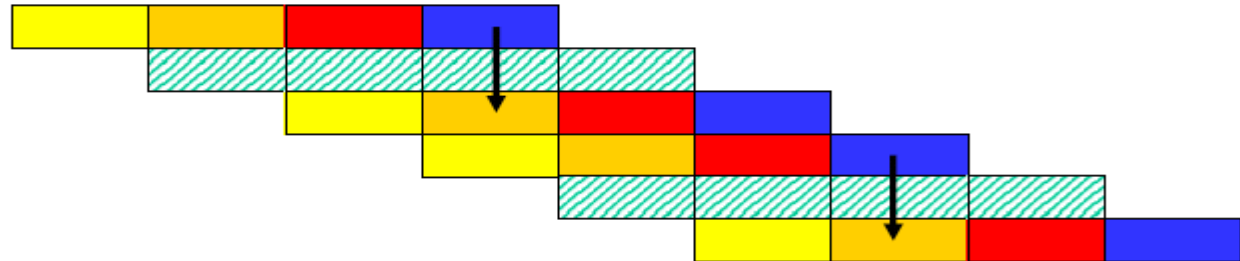
Segmentación

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

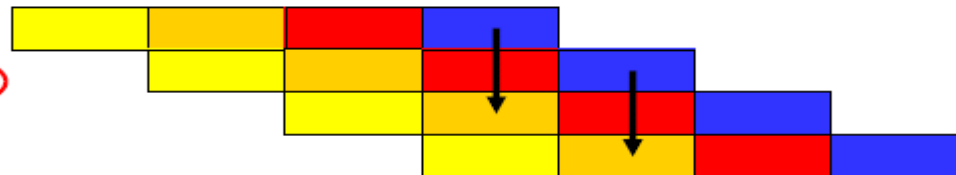
R3=R3+R1  
R5=R5-R3  
R4=R4+R1  
R6=R6-R4



R3=R3+R1  
**nop**  
R5=R5-R3  
R4=R4+R1  
**nop**  
R6=R6-R4



R3=R3+R1  
**R4=R4+R1**  
**R5=R5-R3**  
R6=R6-R4



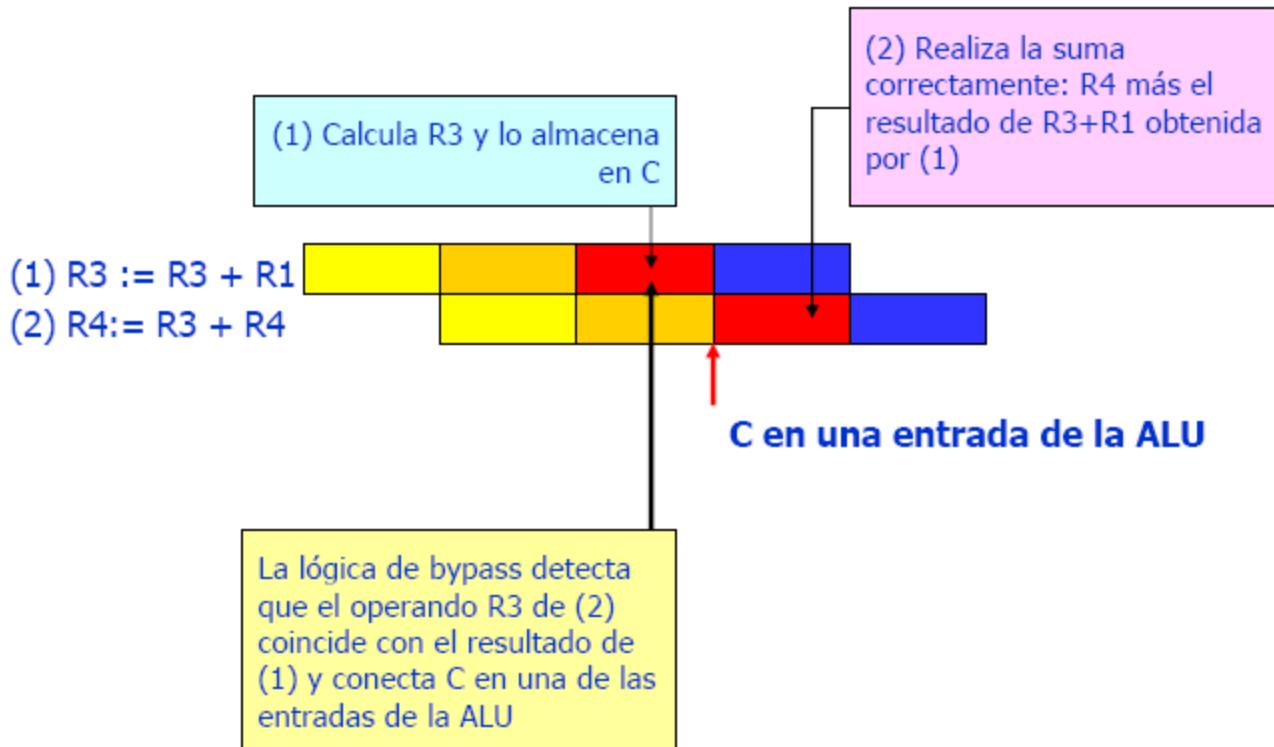
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de datos
  - Forwardings



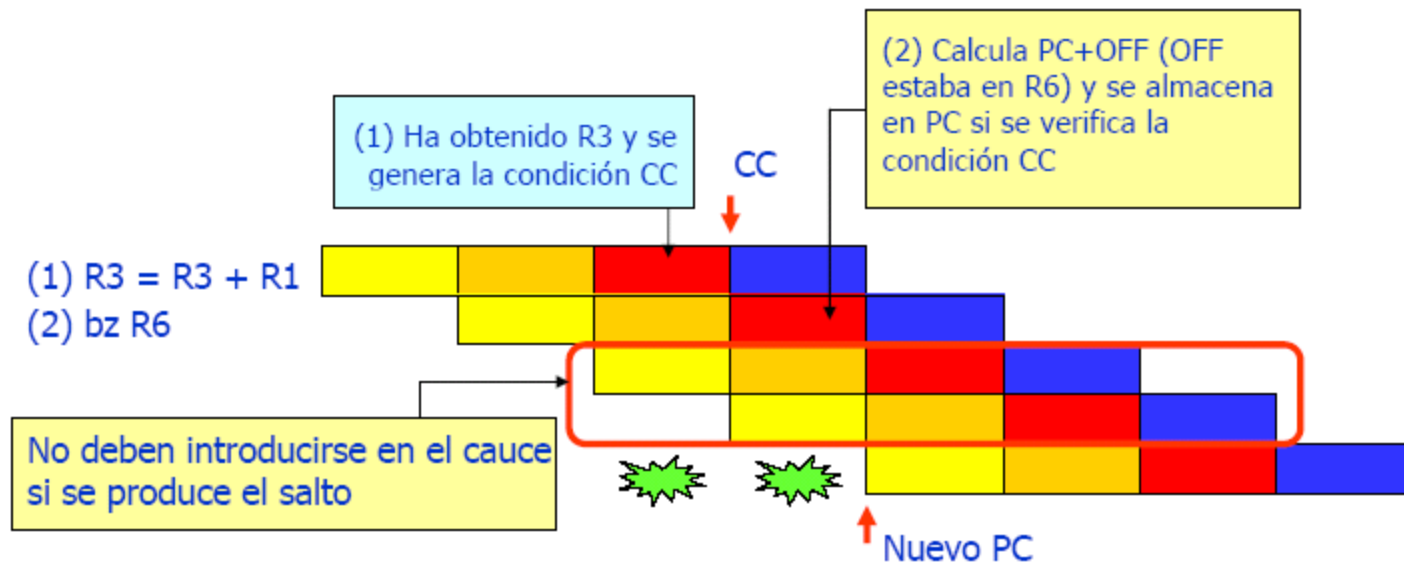
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de control
  - Abortar operaciones



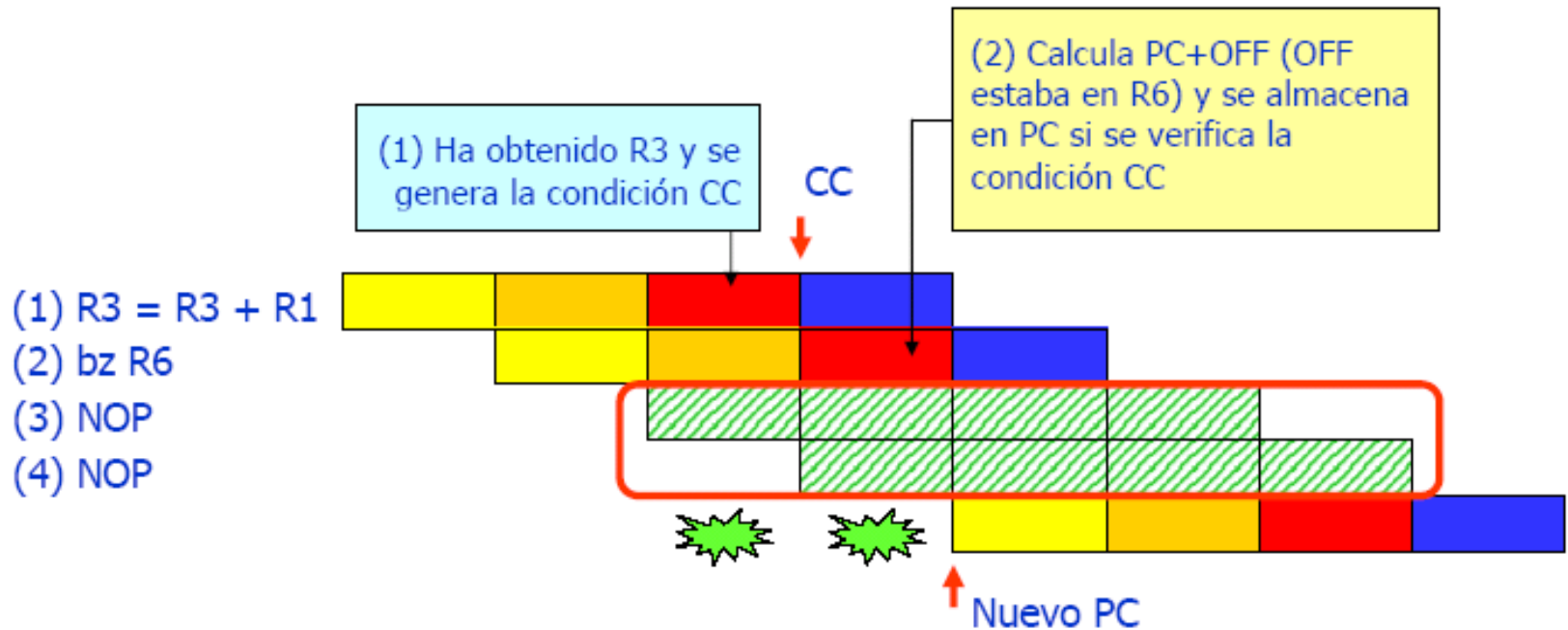
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

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



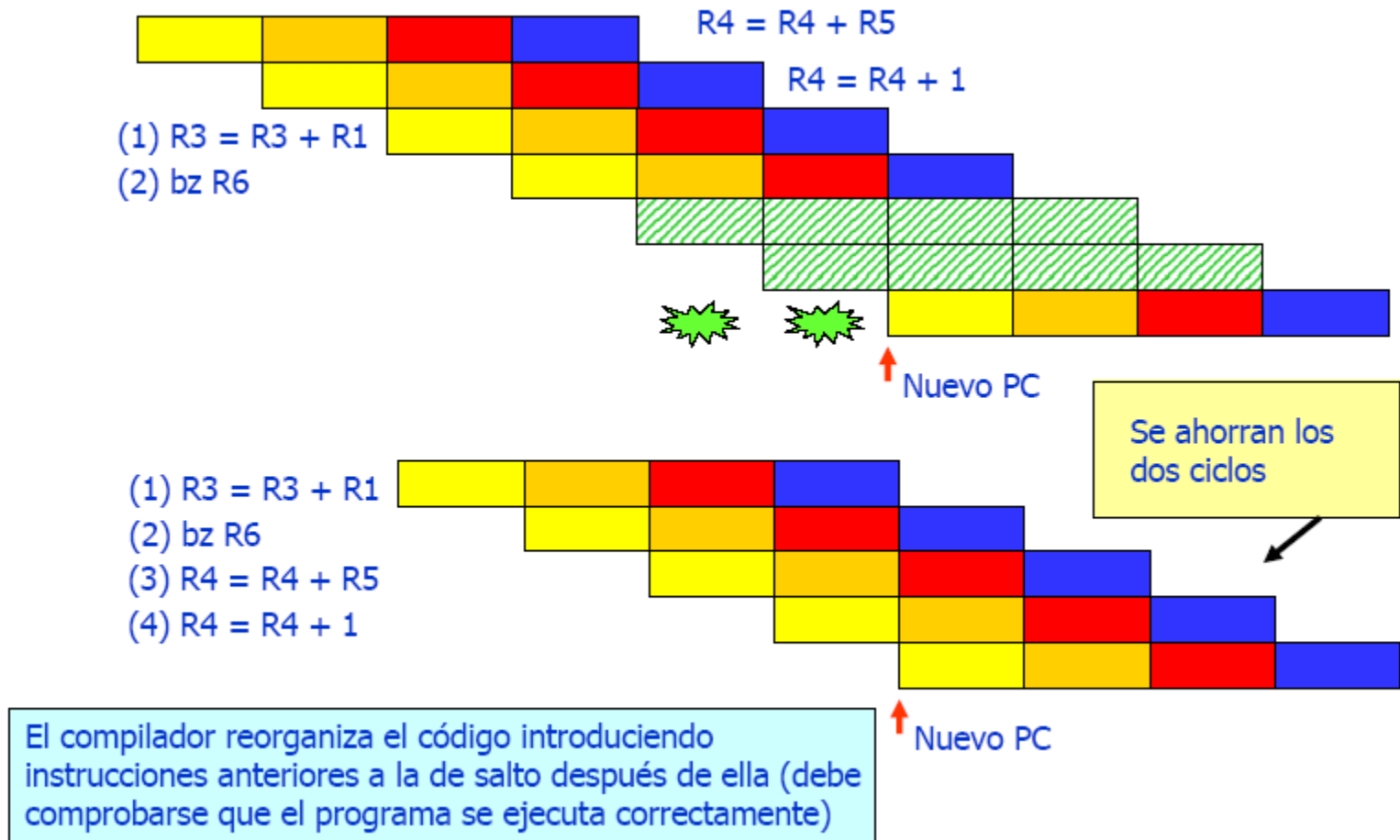
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

- Soluciones a los riesgos de control
  - Delayed branch



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

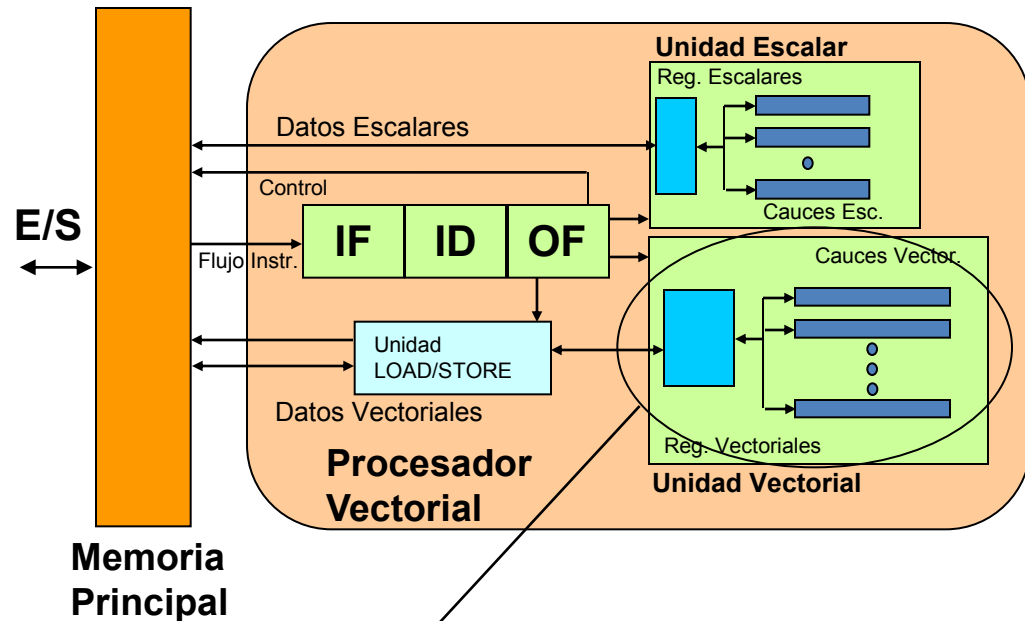
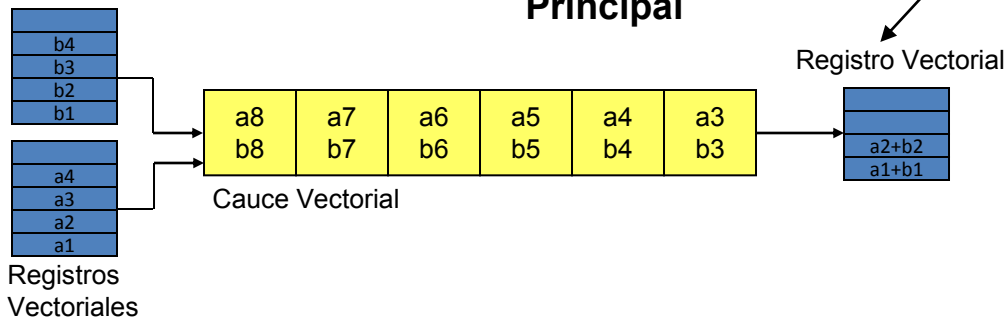
Segmentación

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

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

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)

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

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
```



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

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., $R1+i \cdot R2$ .
SVWS	(R1, R2), V1	Store V1 from address at R1 with stride in R2, i.e., $R1+i \cdot R2$ .
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 $0, 1 \cdot R1, 2 \cdot R1, \dots, 63 \cdot R1$ into V1.
S_V	V1, V2	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.
S_SV	F0, V1	
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.
MOV2I	R1, VLR	Move the contents of the vector-length register to R1.
MOV2S	VM, F0	Move contents of F0 to the vector-mask register.
MOV2F	F0, VM	Move contents of vector-mask register to F0.

# Ingeniería de los Computadores

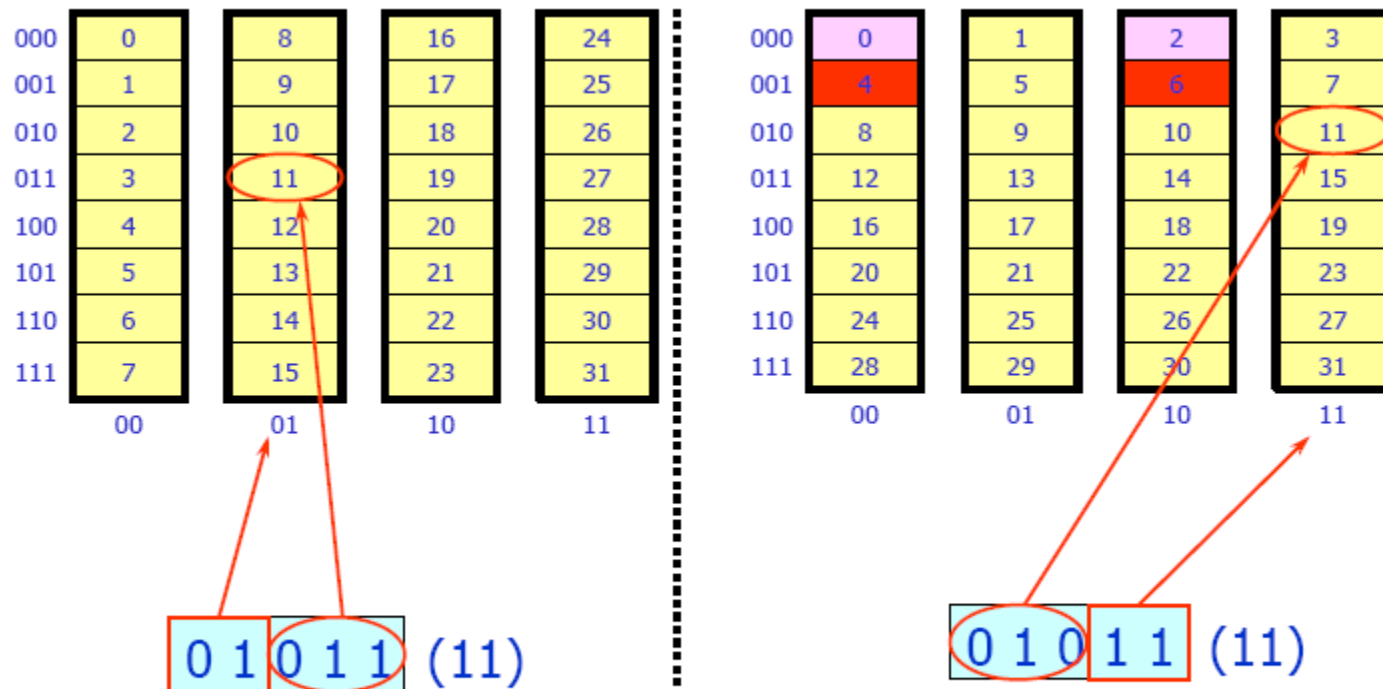
## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Entrelazado de memoria



$2^5=32$  direcciones de memoria

$2^2=4$  módulos de  $2^3=8$  posiciones

Entrelazado Superior

Entrelazado Inferior

# Ingeniería de los Computadores

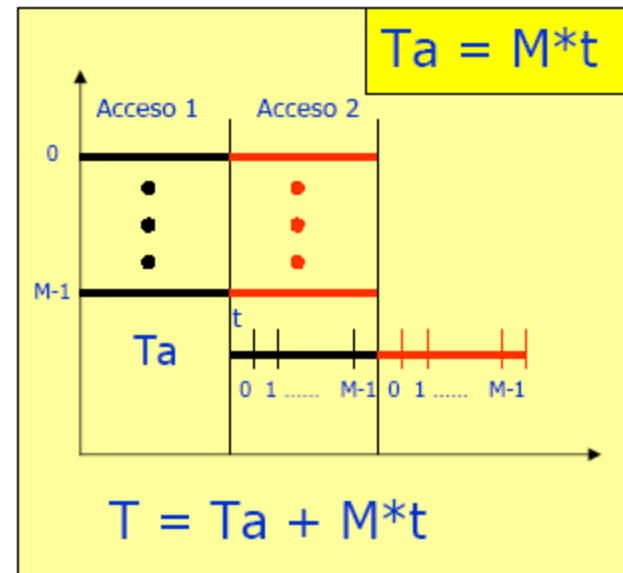
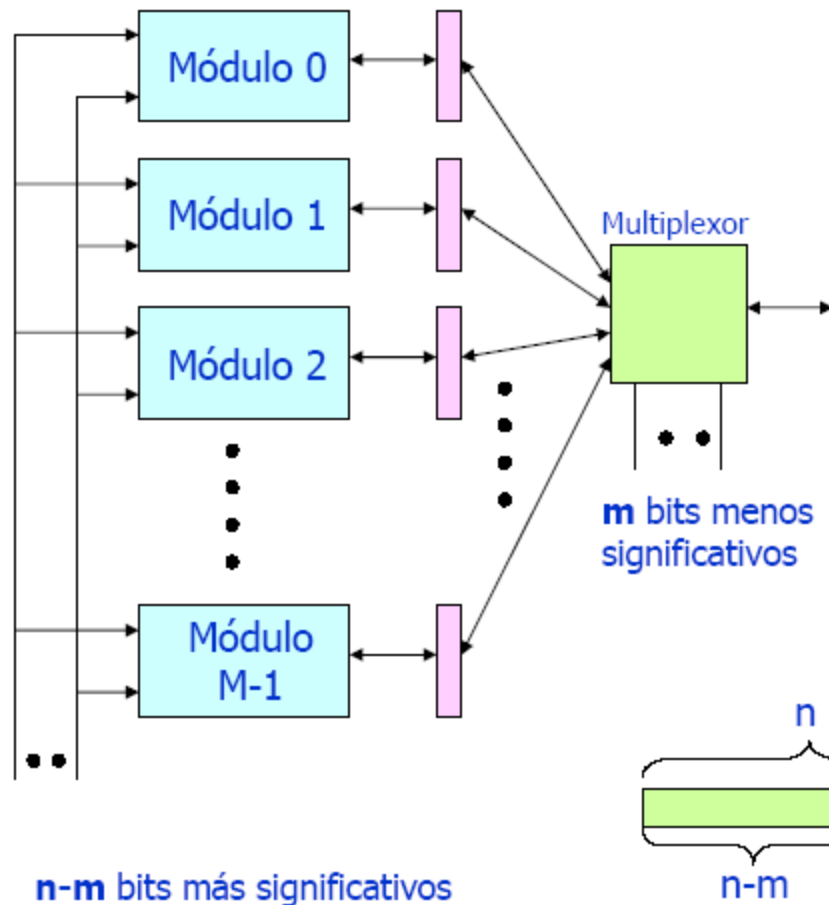
## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Acceso a memoria simultáneo o tipo S



Con Entrelazado Inferior

$N = 2^n$  direcciones

$M = 2^m$  módulos

$2^{(n-m)}$  direcciones/módulo

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

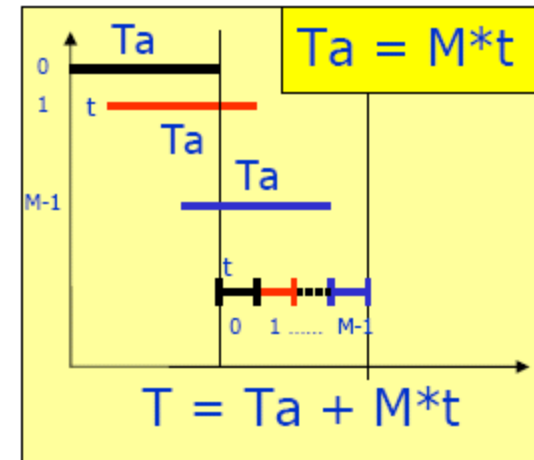
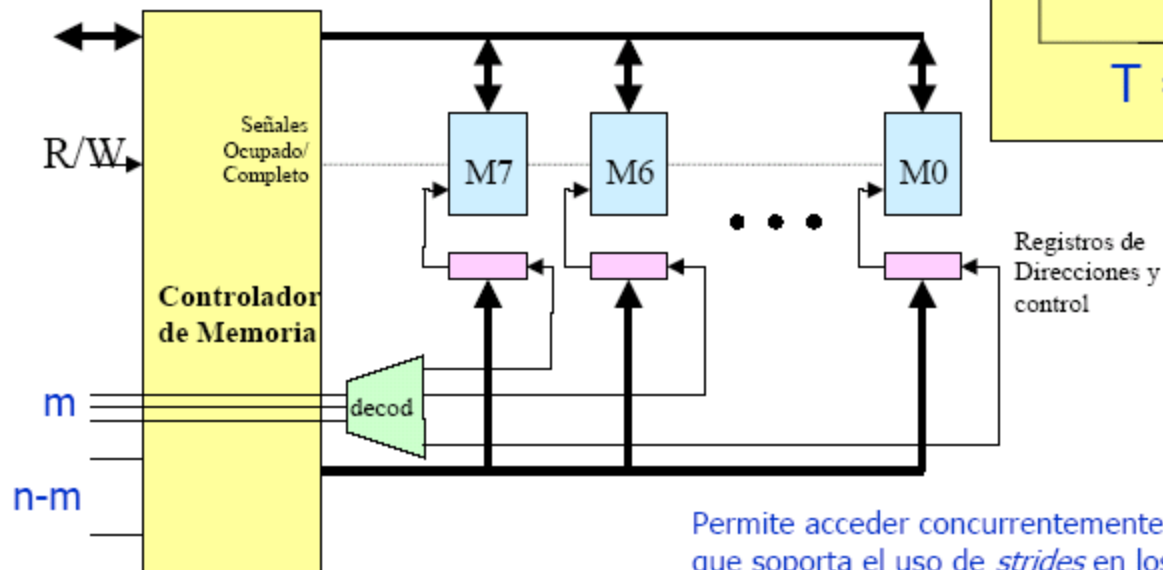
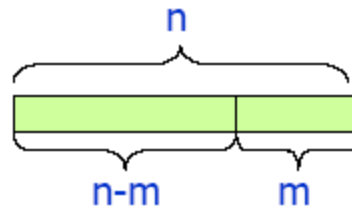
- Acceso a memoria concurrente o tipo C

Con Entrelazado Inferior

$N = 2^n$  direcciones

$M = 2^m$  módulos

$2^{(n-m)}$  direcciones/módulo



Permite acceder concurrentemente a  $M$  direcciones, con lo que soporta el uso de *strides* en los accesos a memoria

# Ingeniería de los Computadores

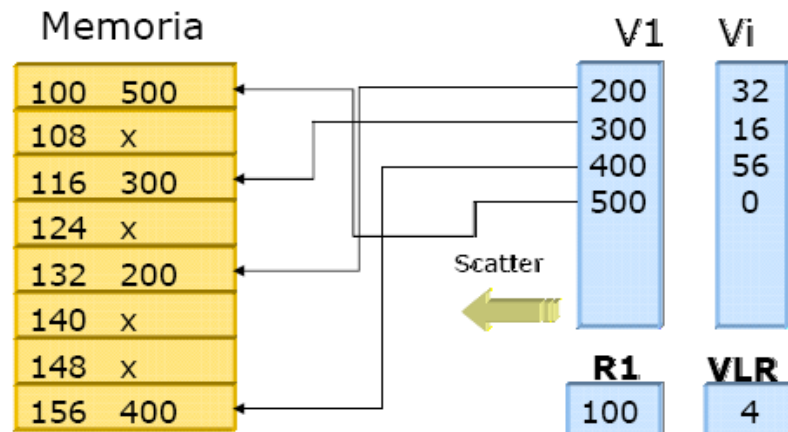
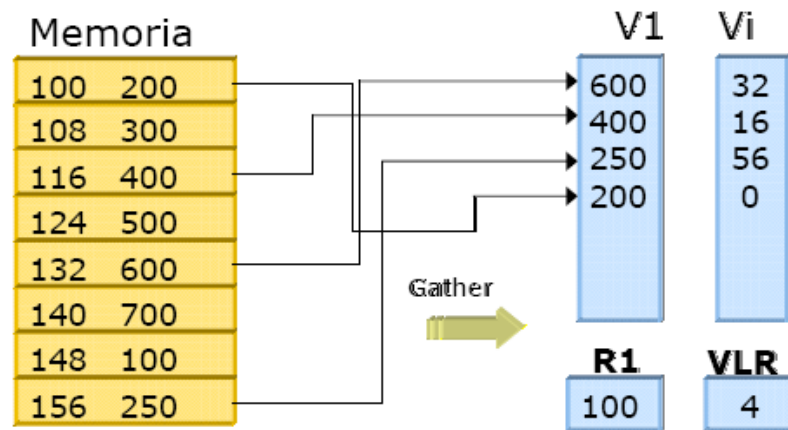
## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Operaciones gather-scatter



# Ingeniería de los Computadores

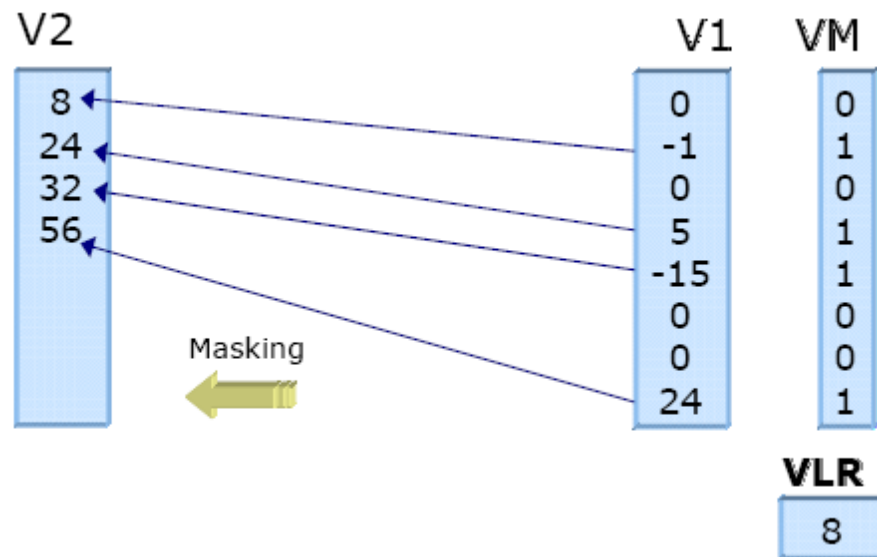
## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Enmascaramiento (gestión de matrices dispersas)



# Ingeniería de los Computadores

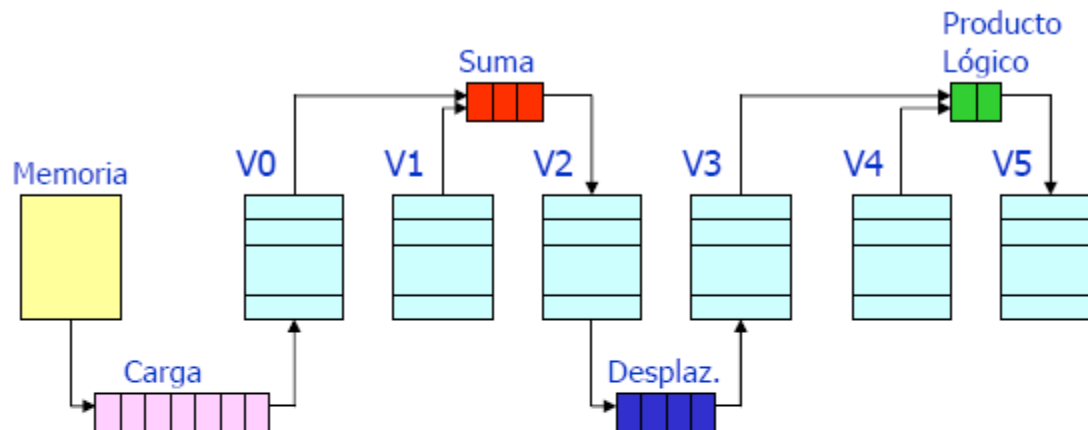
## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

- Rendimiento: encadenamiento de cauce



<b>V0 = Load(Memoria)</b>	<b>(TLI=7)</b>
<b>V2 = V0 + V1</b>	<b>(TLI=3)</b>
<b>V3 = V1 &lt; A3</b>	<b>(TLI=4)</b>
<b>V5 = V3 ^ V4</b>	<b>(TLI=2)</b>

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

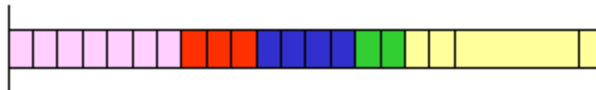
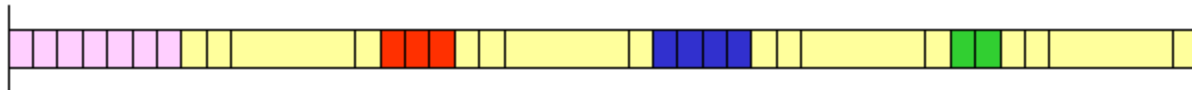
Segmentación

Vectoriales

- Rendimiento: encadenamiento de cauce

Si se espera que termine una operación vectorial para que empiece otra

$$TCV = TLI(carga) + TLI(suma) + TLI(desplaz.) + TLI(Prod.Log) + 4 * K$$



$$TCV = TLI(carga) + TLI(suma) + TLI(desplaz.) + TLI(Prod.Log) + K$$

Si se encadenan los cauces de las distintas operaciones



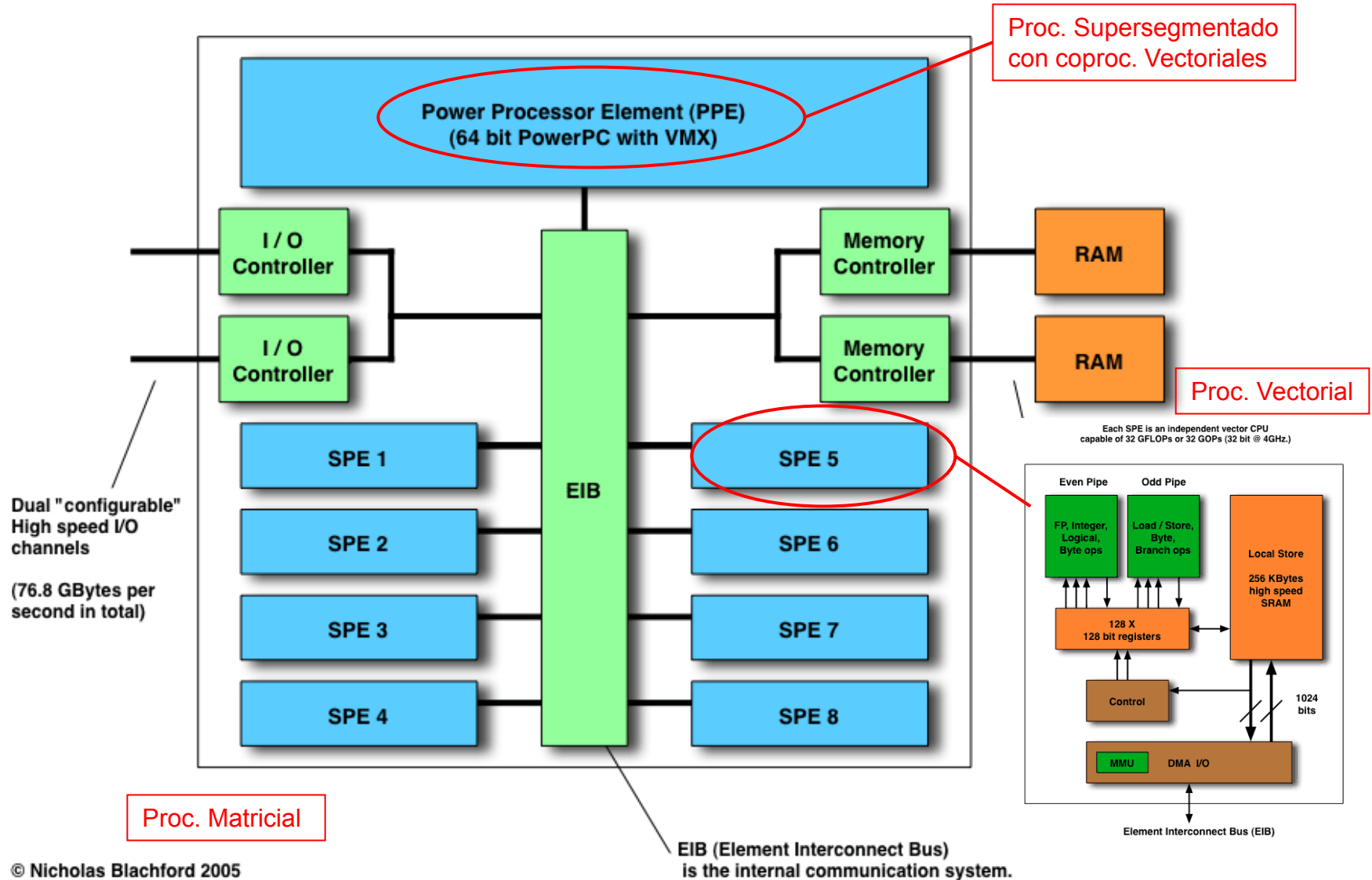
# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

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

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

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

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

$$T_{\text{CICLO}} = 1/F$$

F=frecuencia

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

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

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

**CPI**

**CPE** = ciclos entre inicio de emisión de instrucciones.

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

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

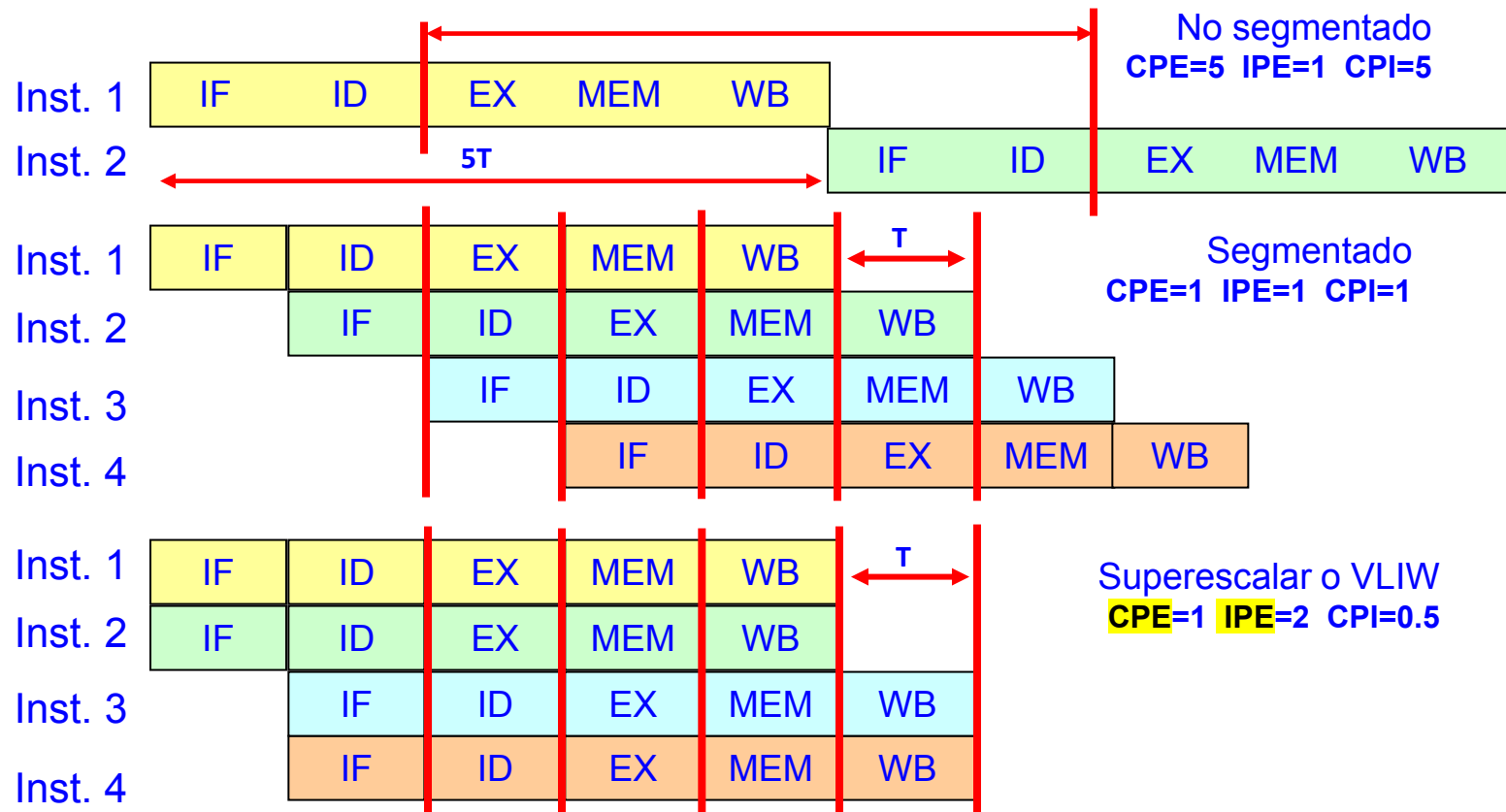
Segmentación

Vectoriales

Rendimiento

Ejemplo:

$$\text{CPI} = \text{CPE} / \text{IPE}$$



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

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

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

**Noper** = número de operaciones que realiza el programa.

**Op\_instr** = número de operaciones que puede codificar una instrucción.

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

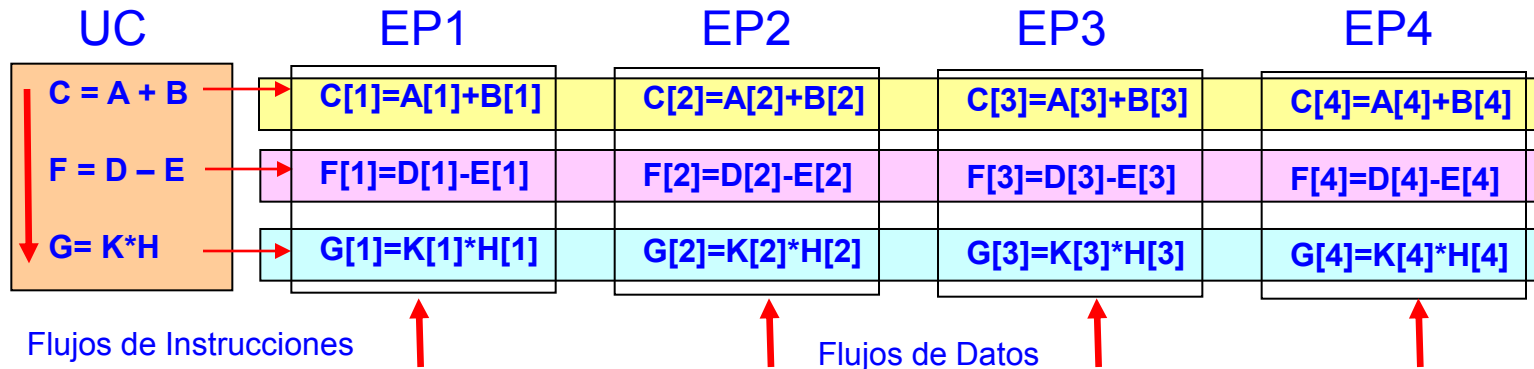
### Ejemplo:

$$NI = \text{Noper}/\text{Op\_instr}$$

Ejemplo paralelismo datos

Procesador Matricial

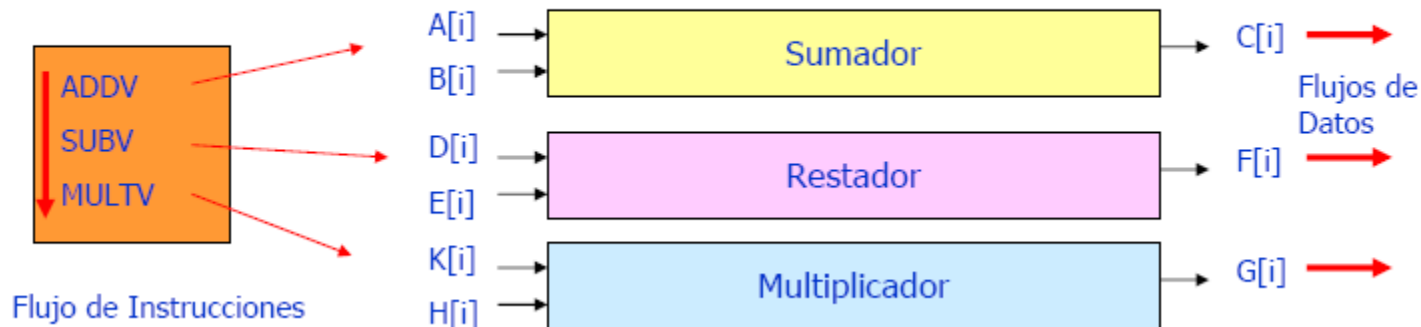
$$\text{Noper}=12 \quad \text{Op\_instr}=4 \quad NI=3$$



Ejemplo paralelismo instrucciones

Procesador Vectorial

$$\text{Noper}=12 \quad \text{Op\_instr}=4 \quad NI=3$$



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

- Evaluación del rendimiento: benchmarks
- Tipos de benchmarks
  - Aplicaciones reales. Compiladores, Word, MathLab, ...
  - Kernels. Trozos de aplicaciones reales seleccionados para evaluar características específicas.
  - Simples (Toys). Pequeños programas fáciles de programar y cuyo resultado es conocido (Quicksort).
  - Sintéticos. Reproducen porcentajes de instrucciones y situaciones de carga reales.
- Suites. Conjuntos de benchmarks que miden las prestaciones de los computadores a través de un conjunto de aplicaciones distintas. Las limitaciones de un benchmark se suplen con la presencia de otros. Se cambian periódicamente para evitar optimizaciones realizadas con el único objetivo de mejorar los resultados del conjunto de benchmark.

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

- SPEC CPU2006
  - Enteros: <http://www.spec.org/cpu2006/CINT2006/>
  - Coma flotante: <http://www.spec.org/cpu2006/CFP2006/>
- Benchmarks para computadores de altas prestaciones
  - SPEC MPI2007: <http://www.spec.org/mpi2007/docs/index.html>
  - Linpack: <http://www.top500.org/project/linpack>



# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

- Medidas de rendimiento:

- Ganancia

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

- Eficiencia

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

- Productividad

# Ingeniería de los Computadores

## Sesión 1. Introducción

¿Qué?

Segmentación

Vectoriales

Rendimiento

<b>Kilobyte (KB)</b>	<b><math>10^3</math> bytes</b>	<b>Kibibyte (KiB)</b>	<b>1024 bytes (<math>2^{10}</math> bytes)</b>
<b>Megabyte (MB)</b>	<b><math>10^6</math> bytes</b>	<b>Mebibyte (MiB)</b>	<b><math>2^{20}</math> bytes</b>
<b>Gigabyte (GB)</b>	<b><math>10^9</math> bytes</b>	<b>Gibibyte (GiB)</b>	<b><math>2^{30}</math> bytes</b>
<b>Terabyte (TB)</b>	<b><math>10^{12}</math> bytes</b>	<b>Tebibyte (TiB)</b>	<b><math>2^{40}</math> bytes</b>