

Ingeniería de los Computadores

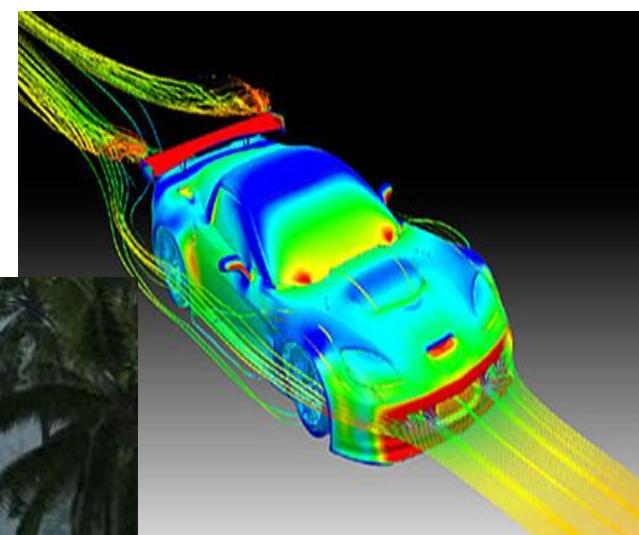
Unidad 1. Introducción al paralelismo

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

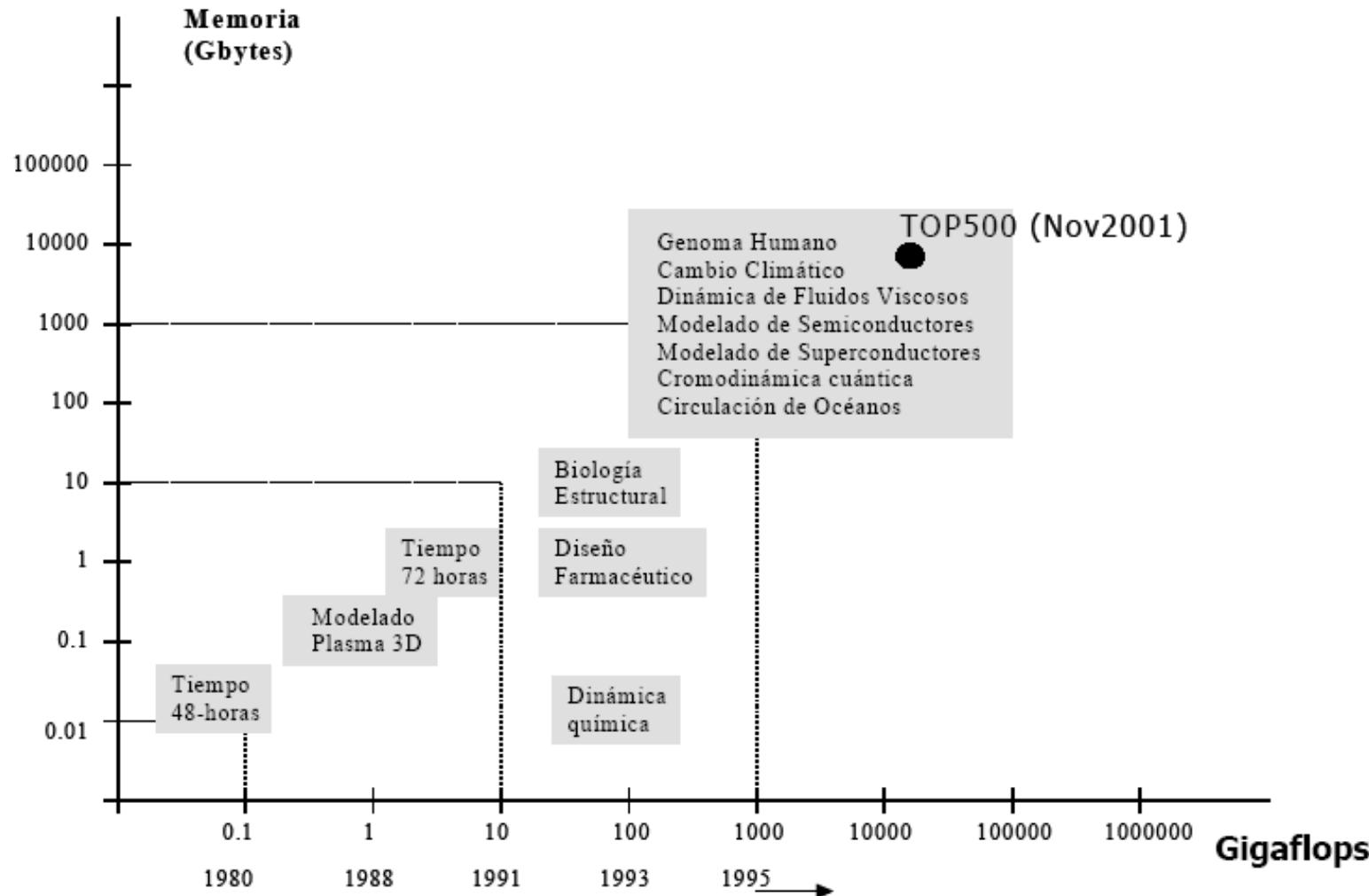
- Evolución de la informática \longleftrightarrow Necesidades computacionales



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

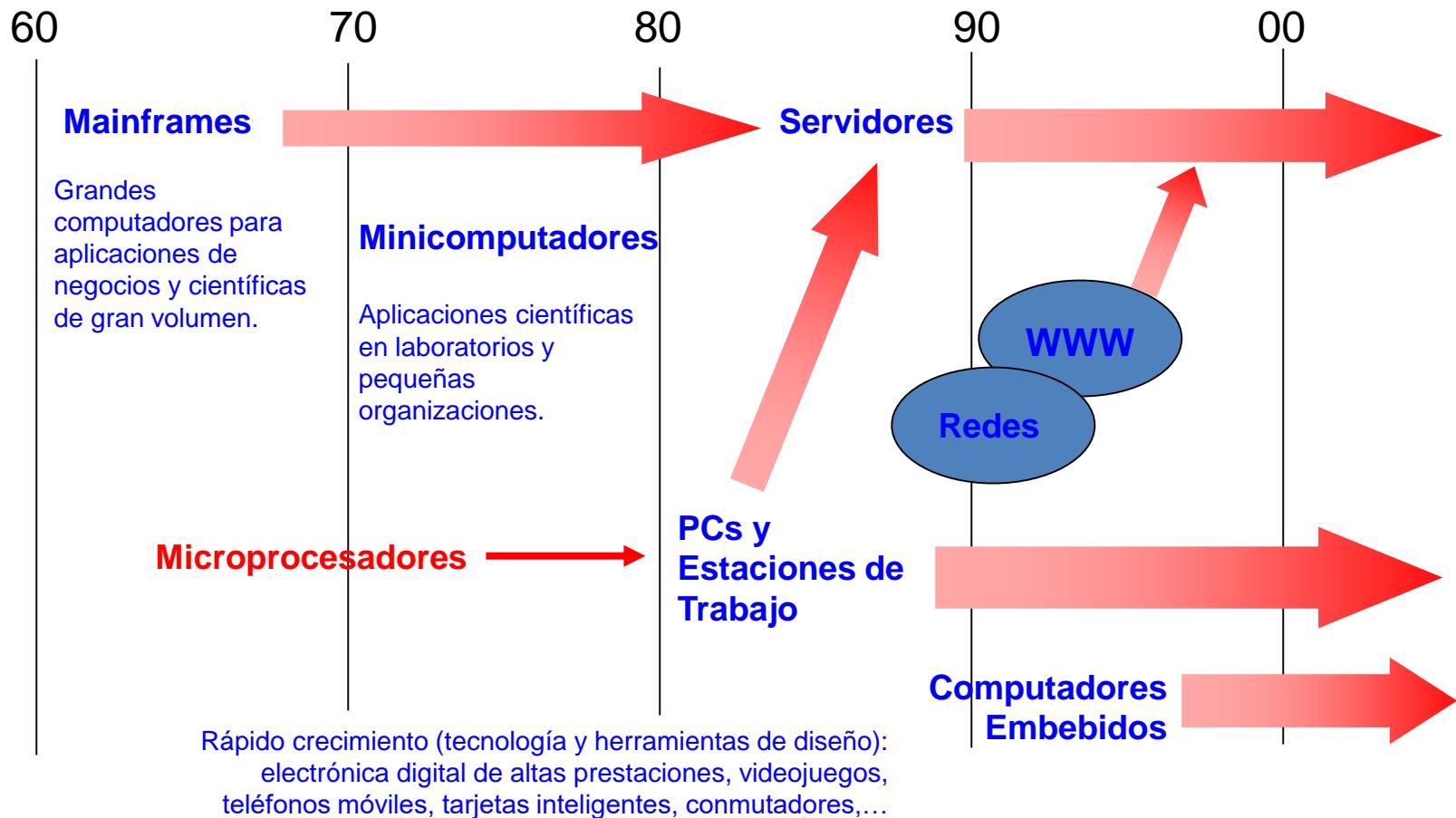
¿Qué?



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

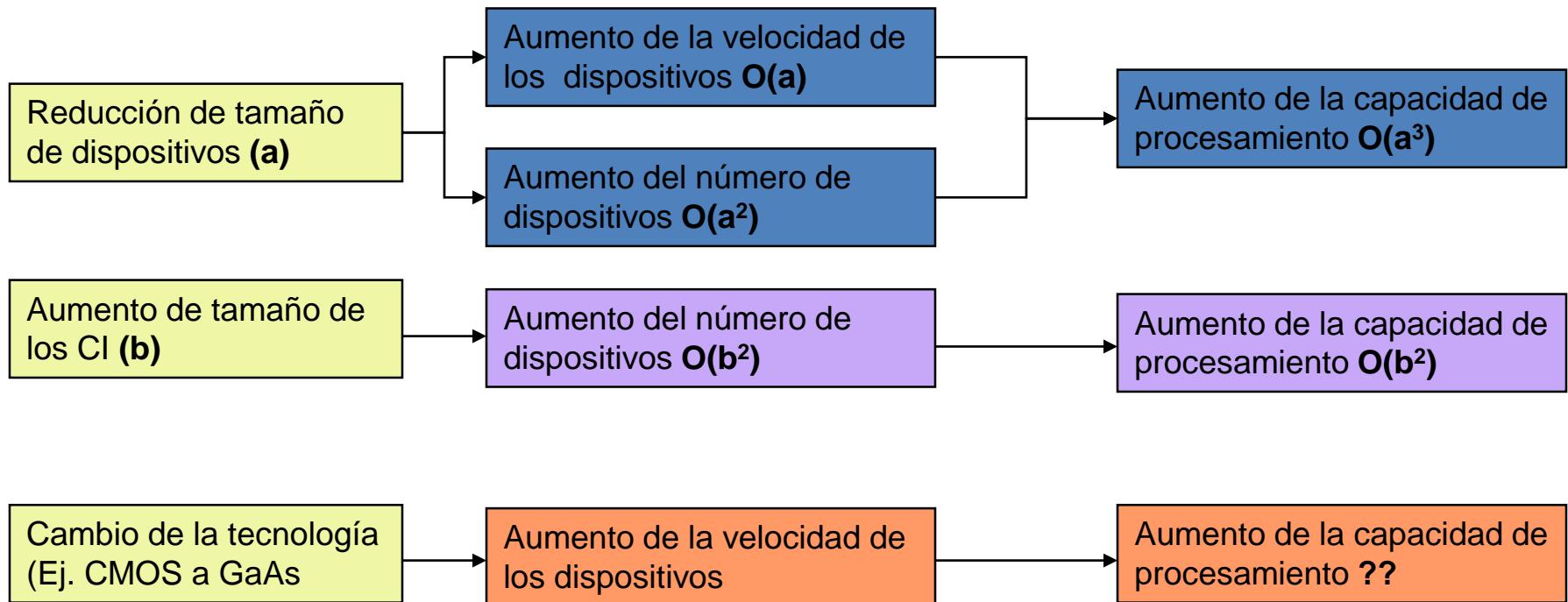


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

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



Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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).

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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, ...)

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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)

Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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

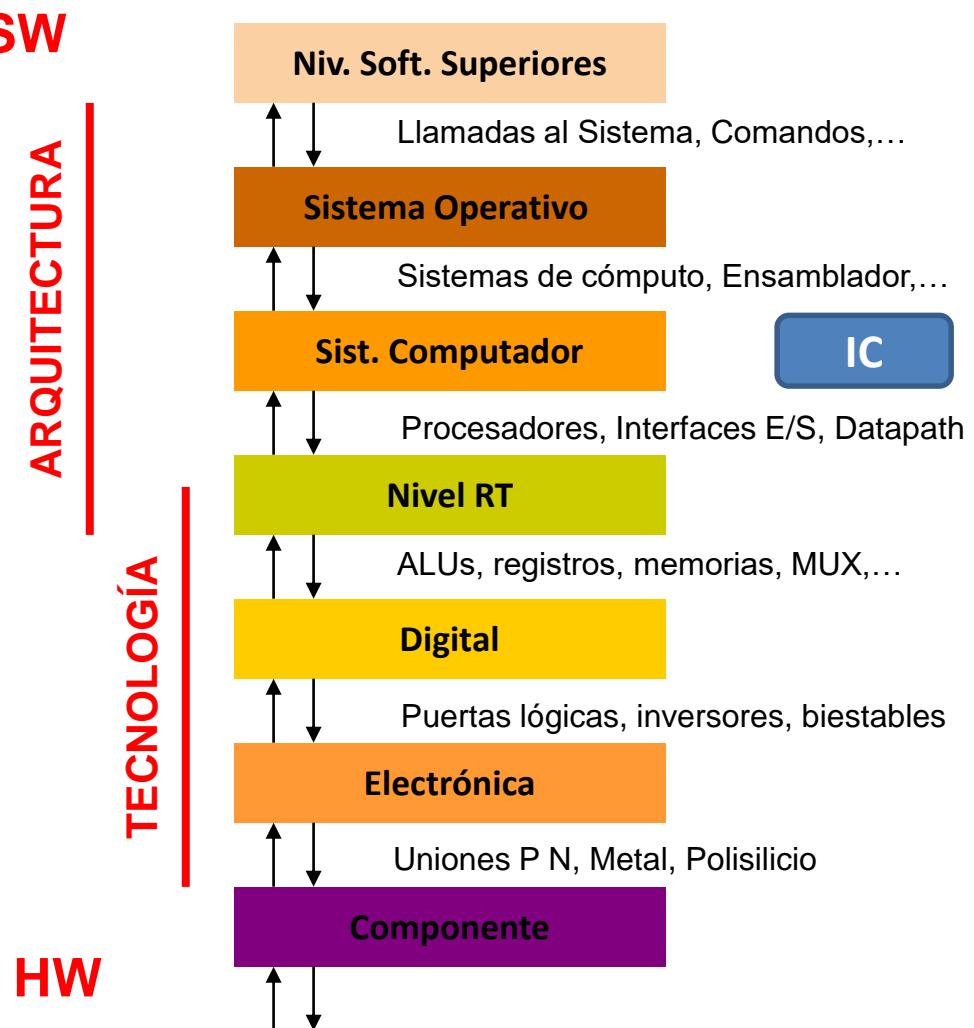
Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿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.



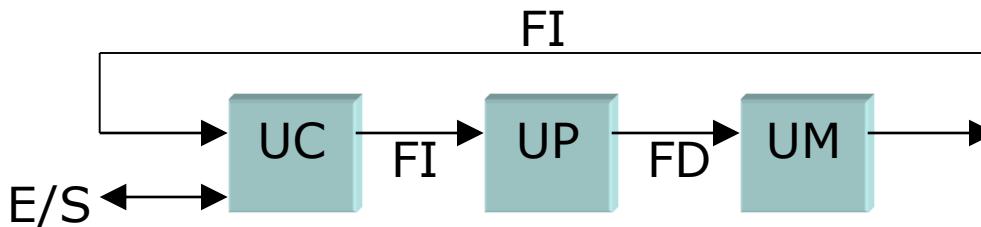
Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

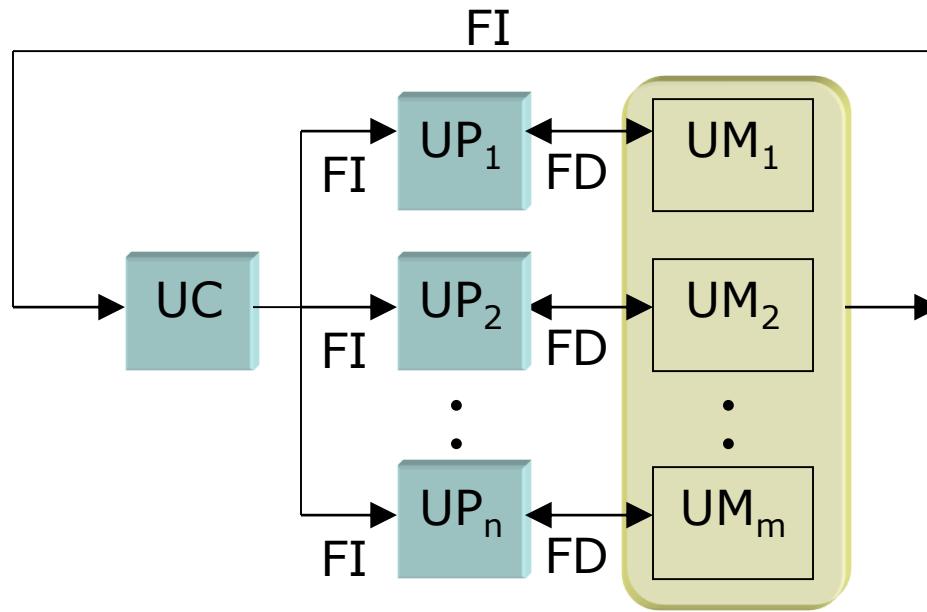
¿Qué?

- Taxonomía de Flynn

➤ SISD



➤ SIMD

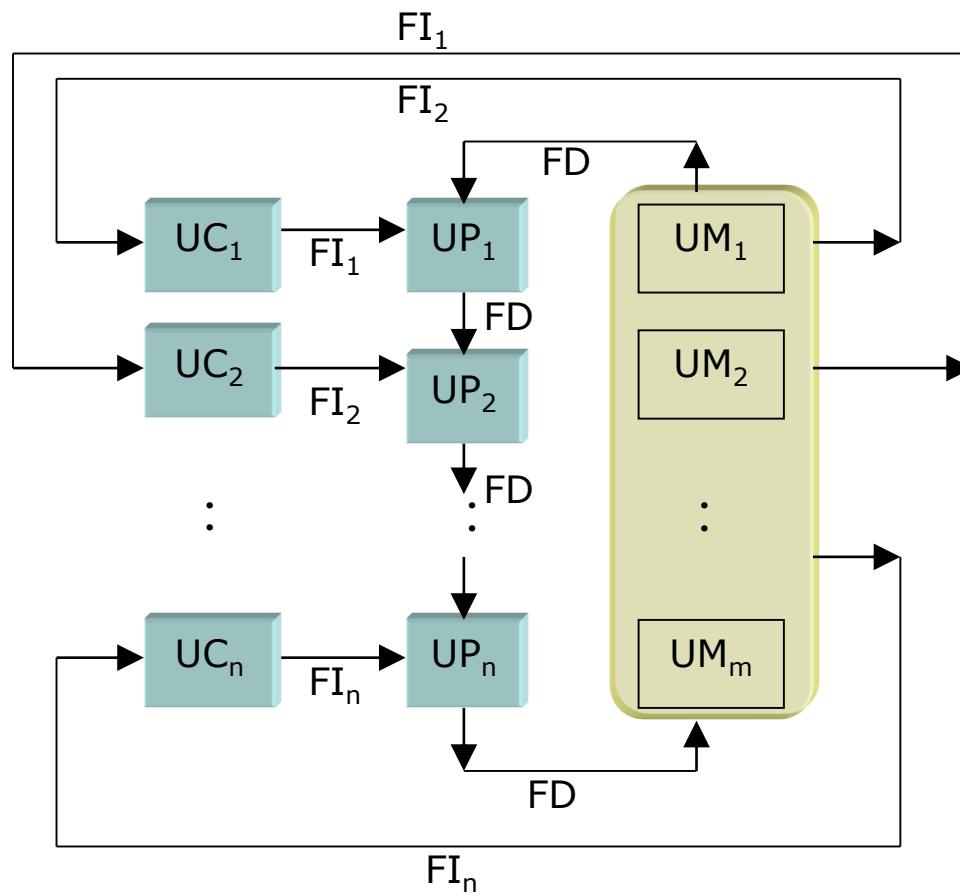


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn
- MISD

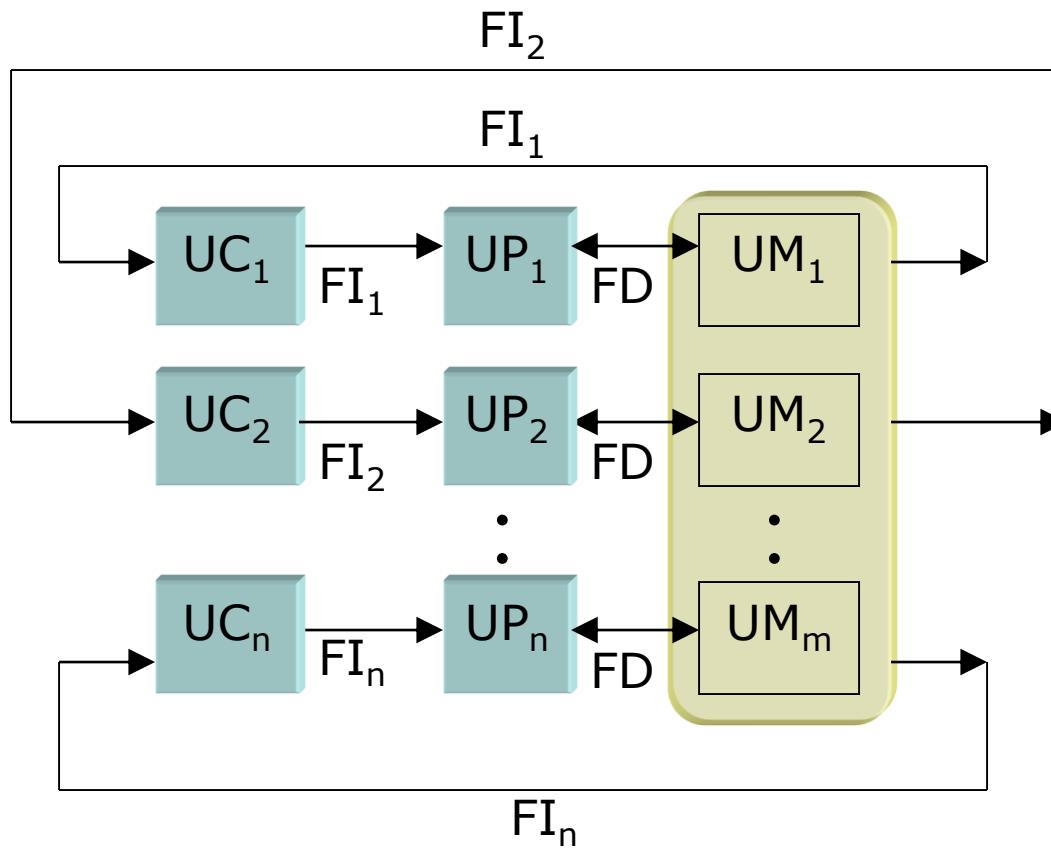


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn
- MIMD

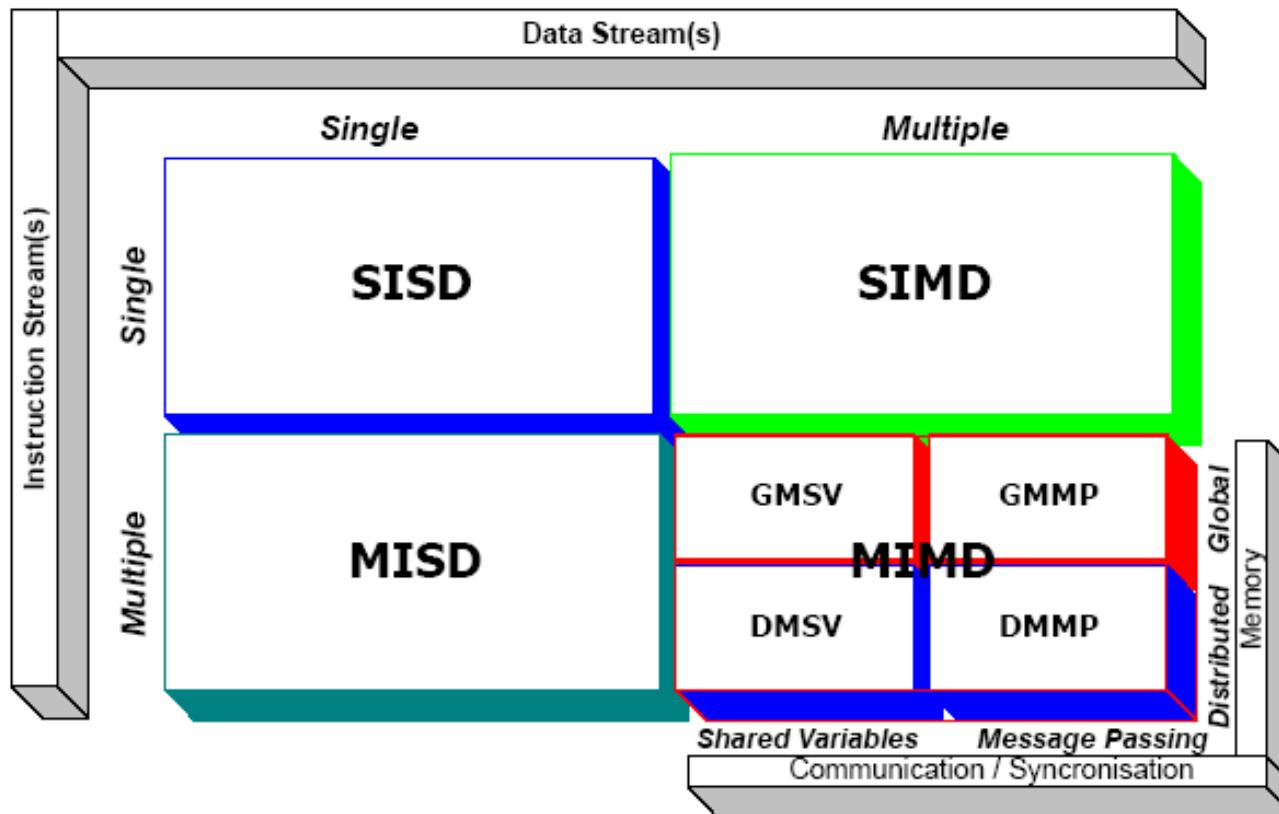


Unidad 1. Introducción al paralelismo

1.1 Introducción y motivación

¿Qué?

- Taxonomía de Flynn-Johnson



Tipos de paralelismo

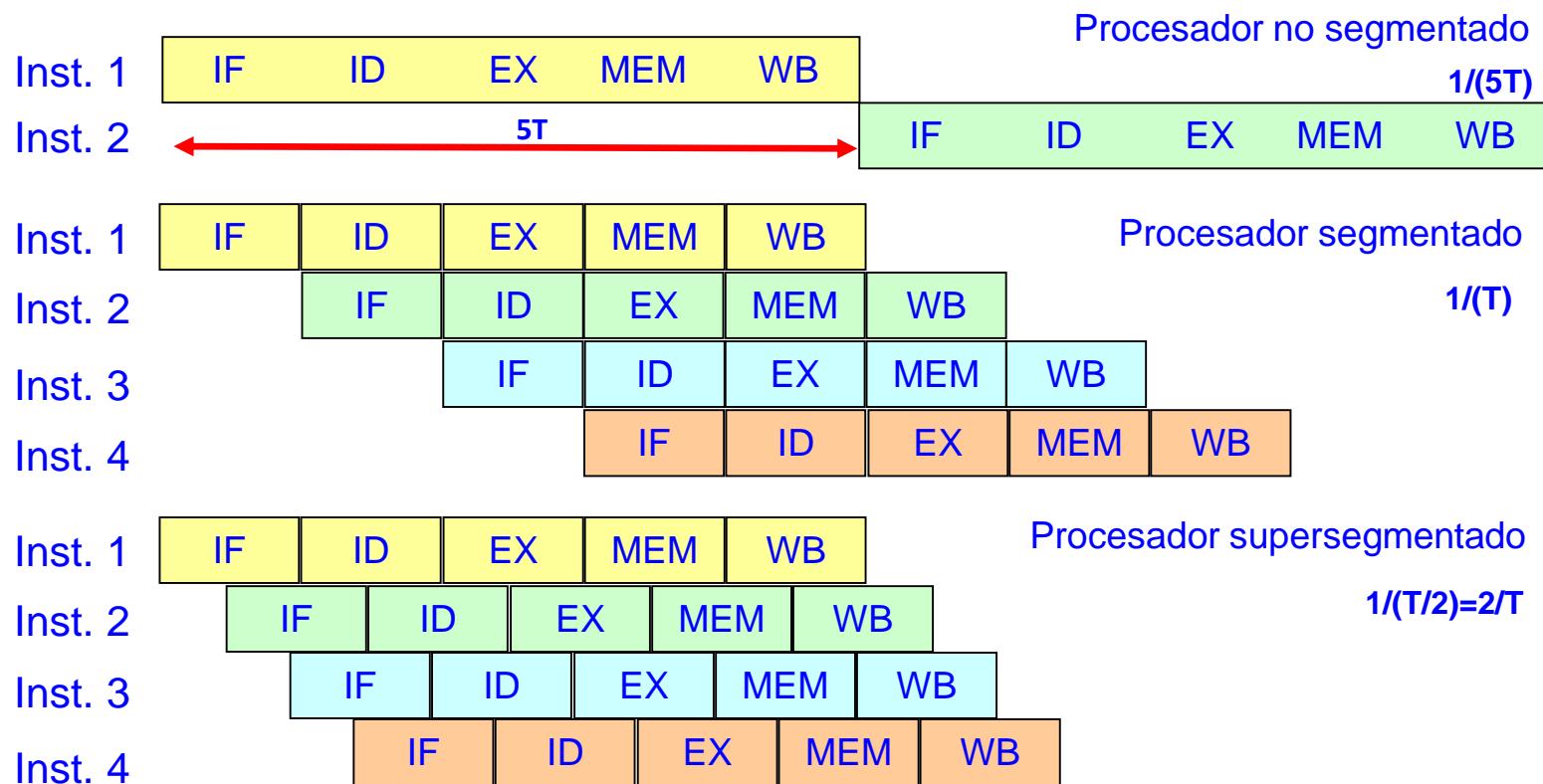
- Tipos de paralelismo
 - **Paralelismo de datos:** La misma función, instrucción, etc. se ejecuta en paralelo pero en cada una de esas ejecuciones se aplica sobre un conjunto de datos distinto.
 - **Paralelismo funcional:** Varias funciones, tareas, instrucciones, etc. (iguales o distintas) se ejecutan 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.

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Segmentación: ILP



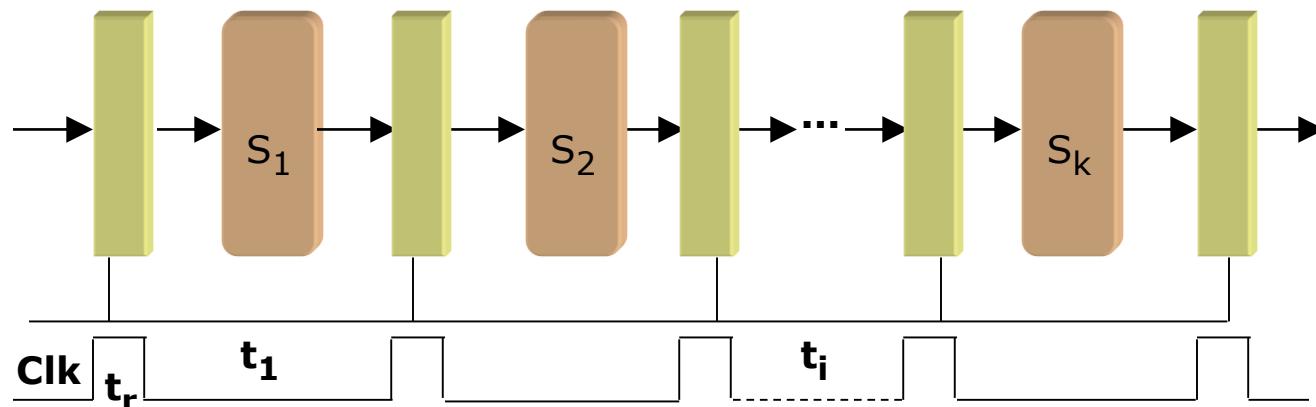
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

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.



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Segmentación

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

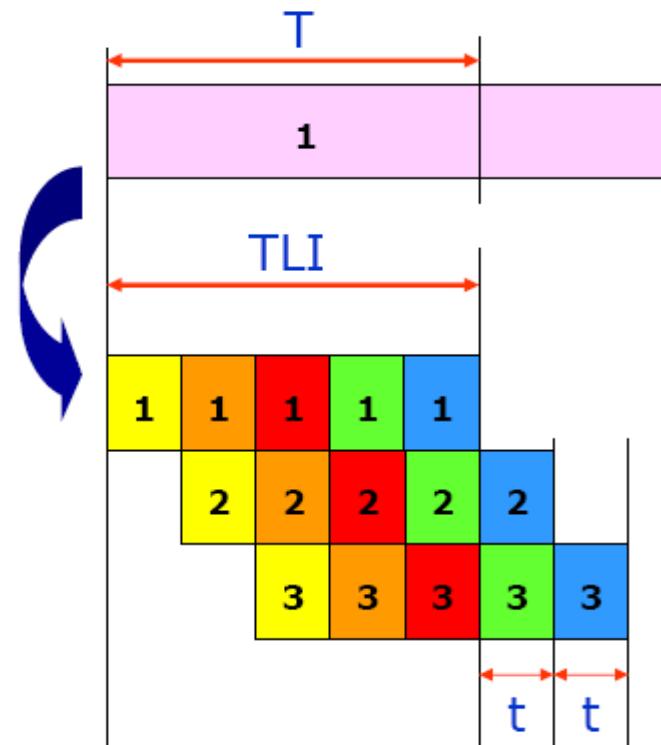
TLI = $k \cdot t$, siendo **k** el nº de etapas del cauce, y **t** la duración de cada etapa

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

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

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

Normalmente, ¿ $TLI > T$ ó $TLI < T$?



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Ganancia real

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

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

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

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Riesgos de datos



RAW (Read After Write)

$$\begin{array}{l} R2 := R1 + R2 \\ R1 := R2 + R3 \end{array}$$



WAR (Write After Read)

$$\begin{array}{l} R2 := R1 + R2 \\ R1 := R2 + R3 \end{array}$$



WAW (Write After Write)

$$\begin{array}{l} R2 := R1 + R2 \\ R2 := R4 + R3 \end{array}$$

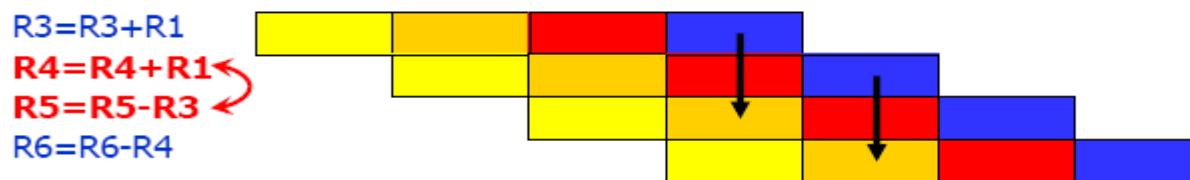
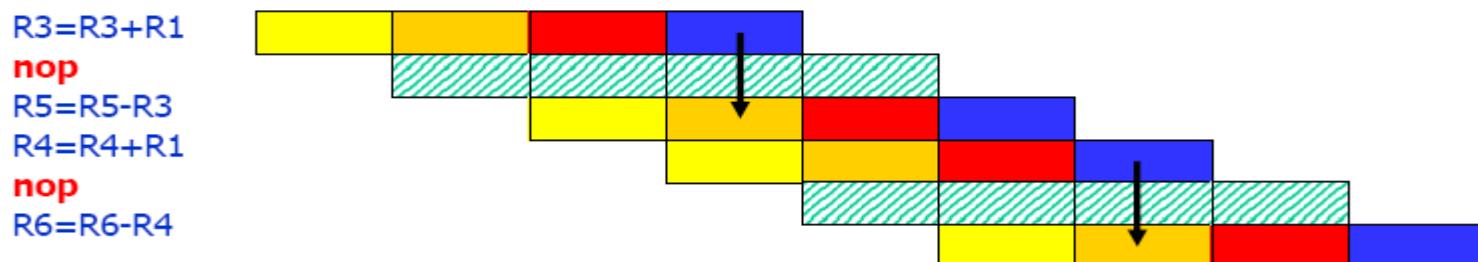
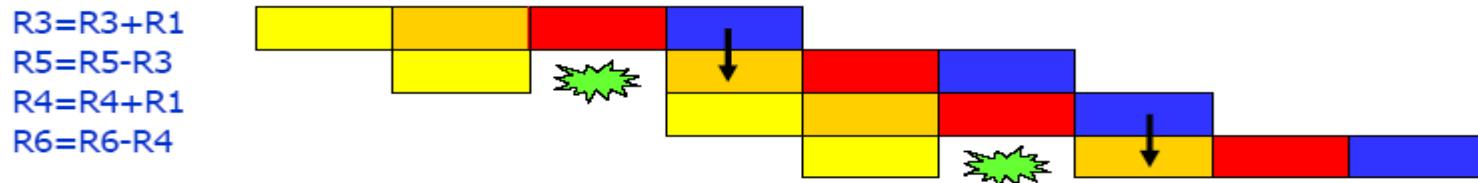
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de datos

- Reorganización de código (intercambio de instrucciones e inserción de NOP)



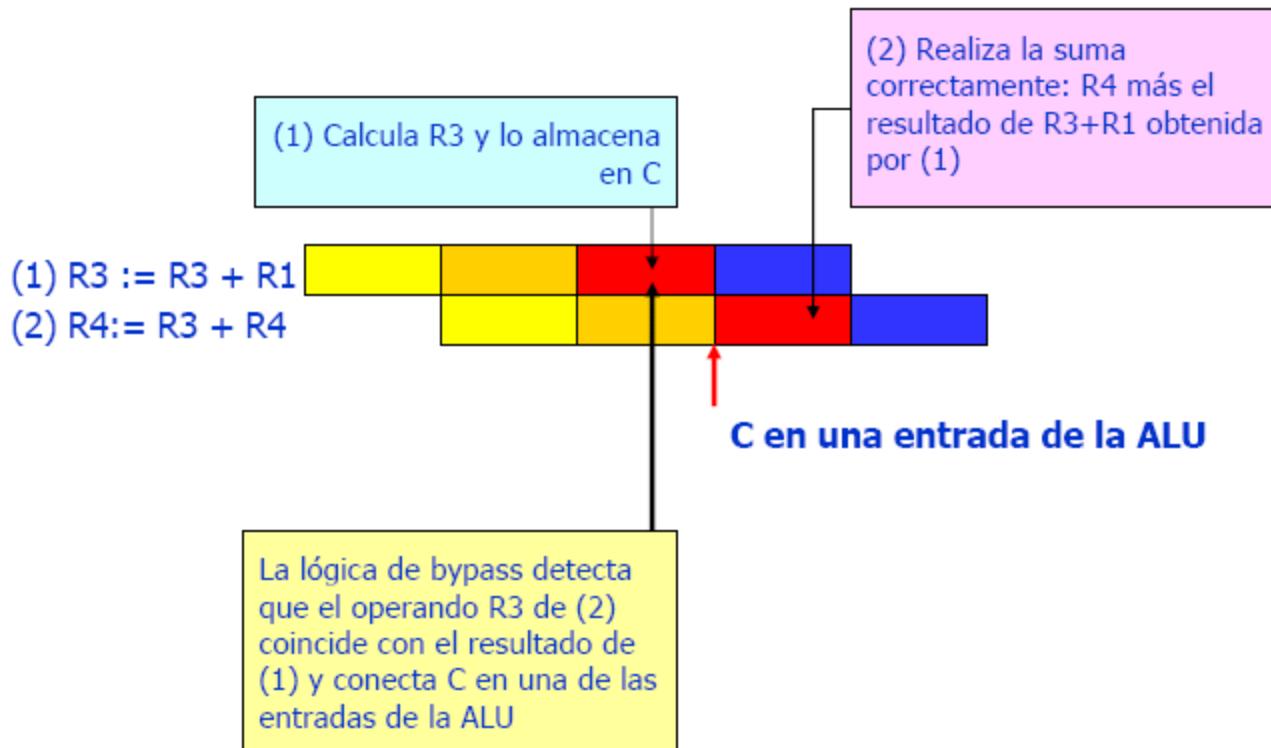
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de datos

- Forwardings

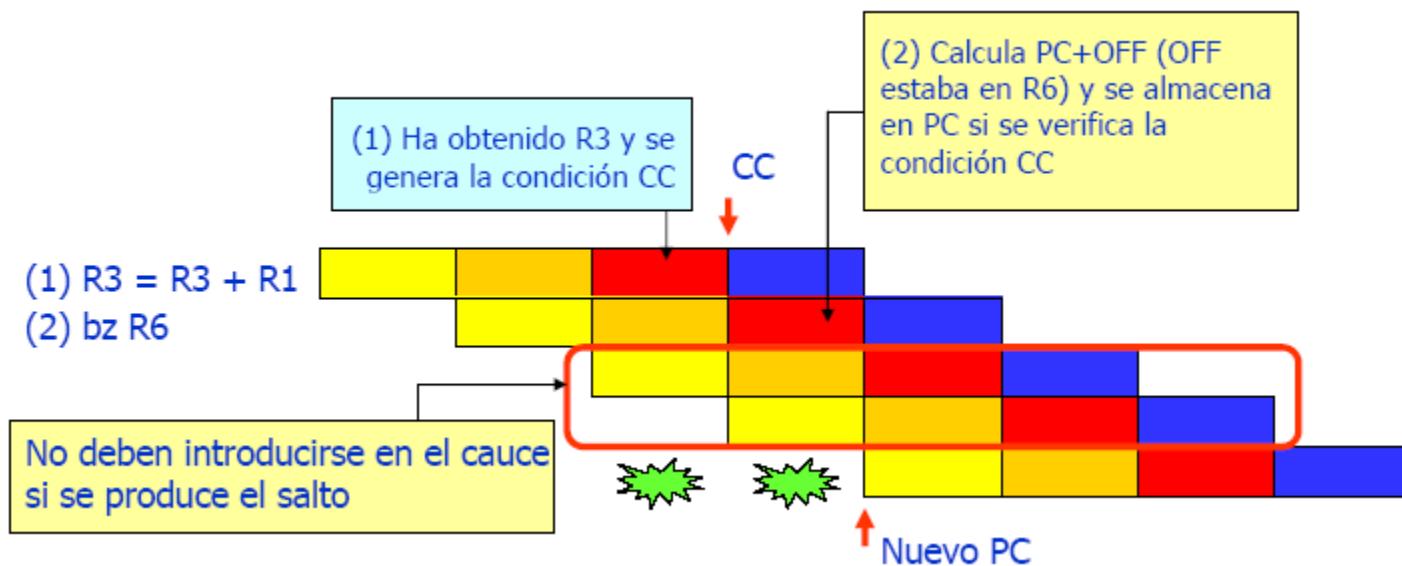


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

- Soluciones a los riesgos de control
 - Abortar operaciones

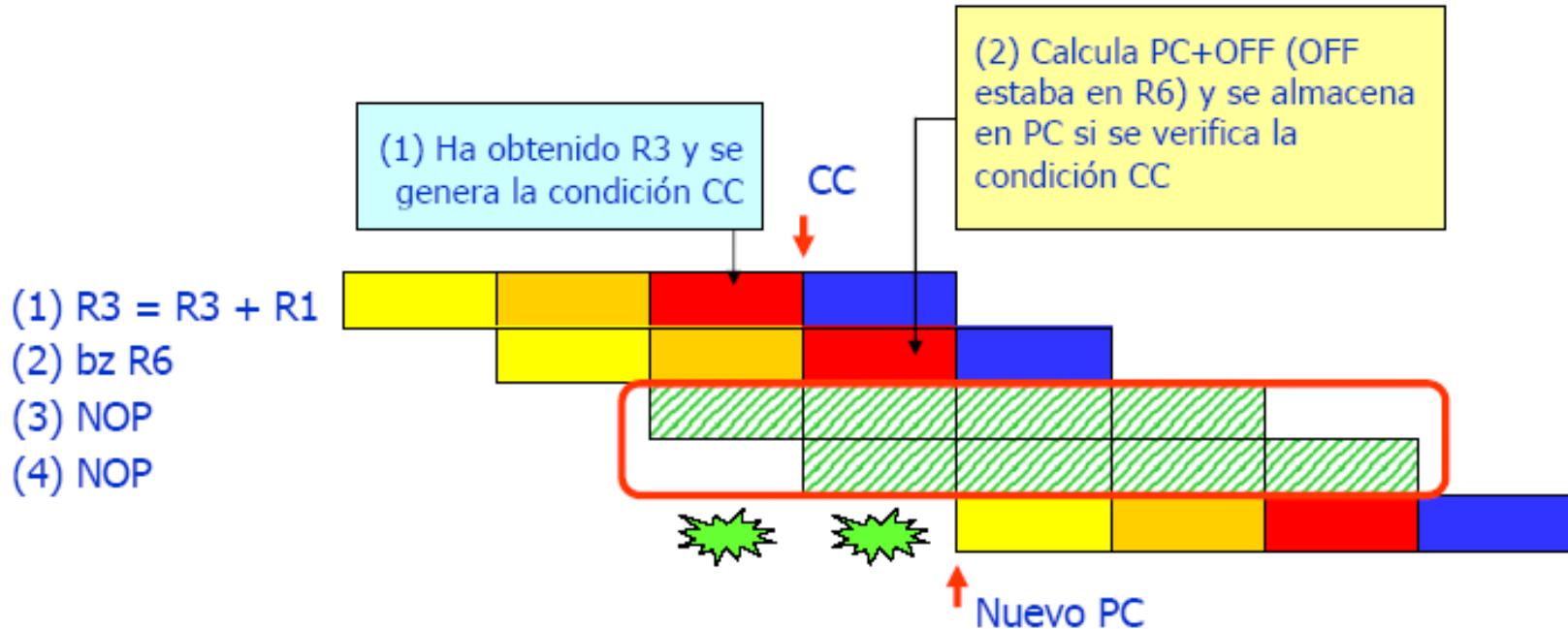


Unidad 1. Introducción al paralelismo

1.2 Paralelismo

ILP

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



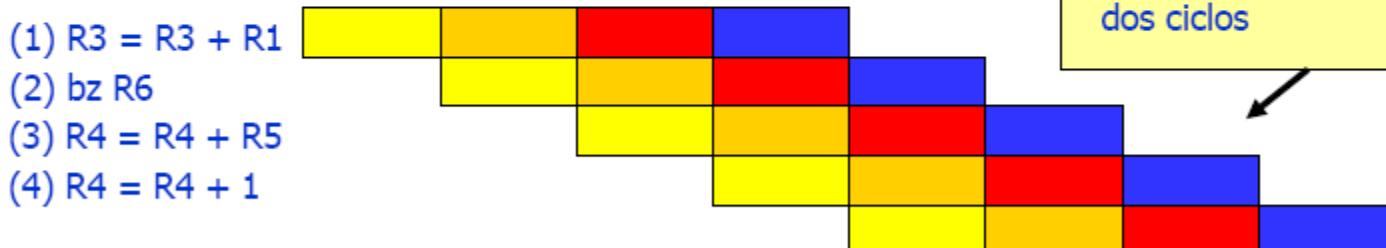
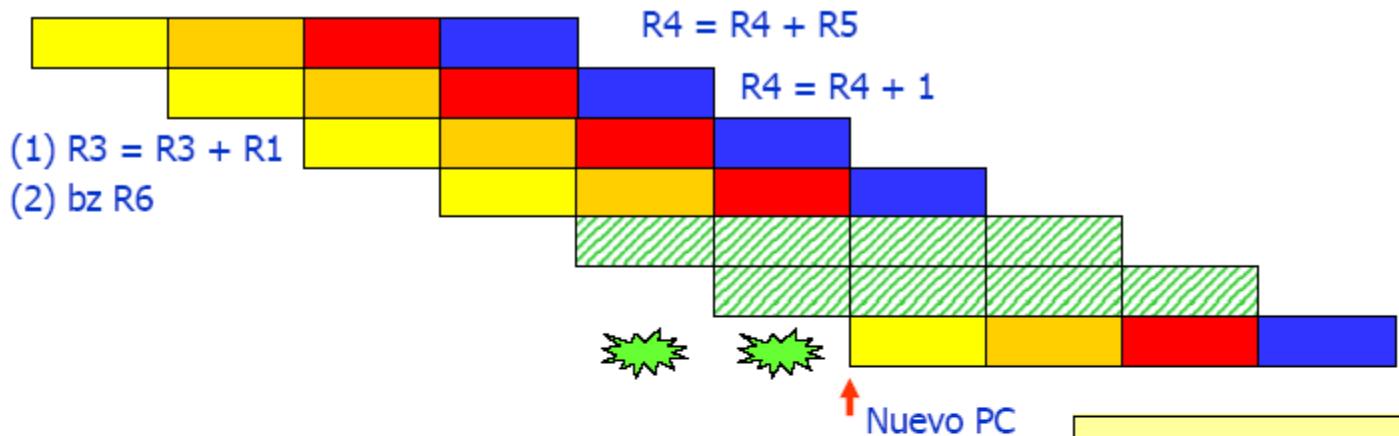
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

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)

Nuevo PC

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

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

$$\text{Tiempo de CPU (T}_{\text{CPU}}\text{)} = \text{Ciclos_del_Programa} \times T_{\text{CICLO}} = \frac{\text{Ciclos_del_Programa}}{\text{Frecuencia_de_Reloj}}$$

$$\text{Ciclos por Instrucción (CPI)} = \frac{\text{Ciclos_del_Programa}}{\text{Número_de_Instrucciones (NI)}}$$

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

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

F=frecuencia

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

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.

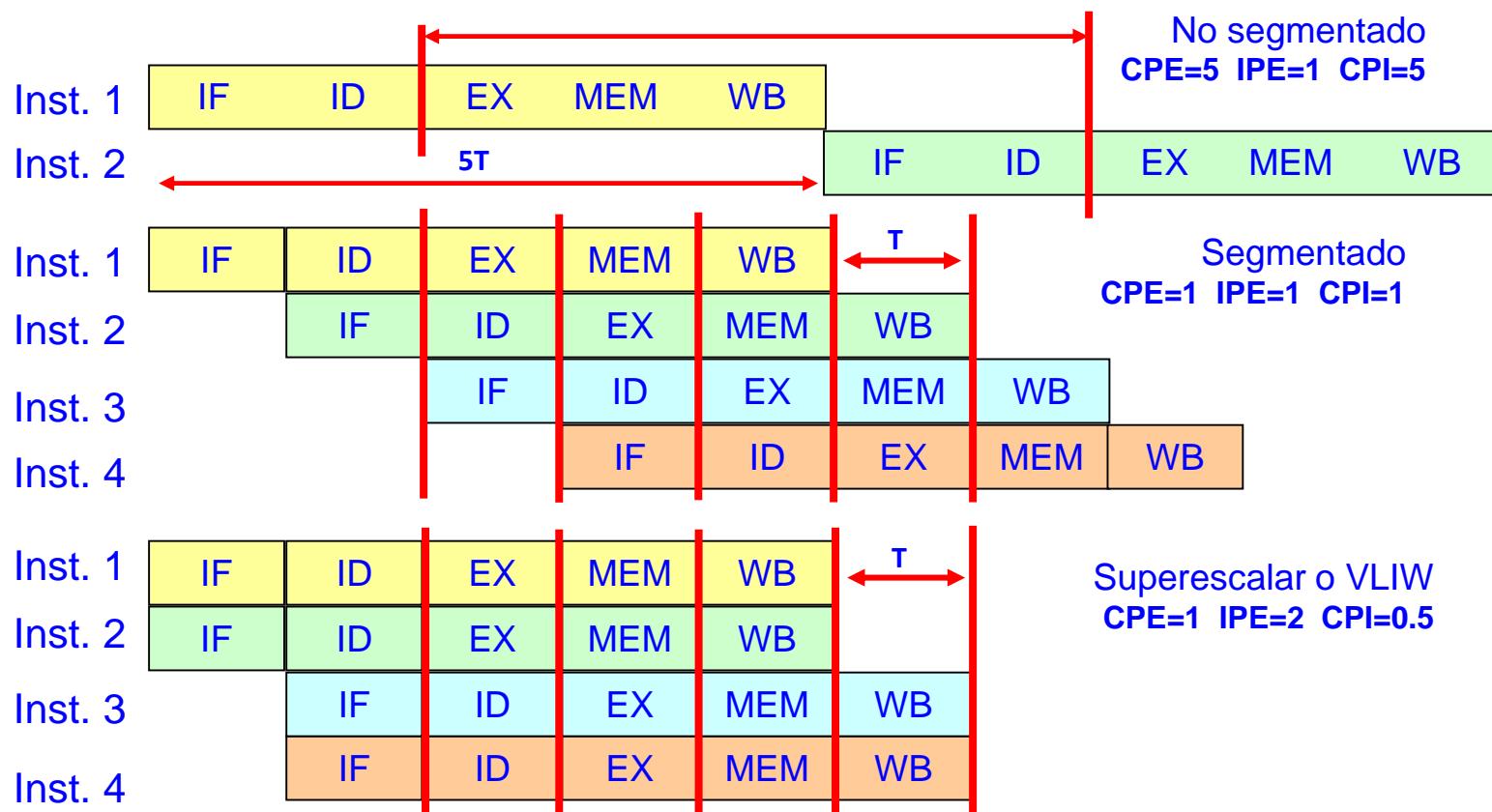
Unidad 1. Introducción al paralelismo

1.2 Paralelismo

Rendimiento

Ejemplo:

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



Rendimiento

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

$$T_{CPU} = (\text{Noper} / \text{Op_instr}) \times \text{CPI} \times T_{CICLO}$$


NI

Noper = número de operaciones que realiza el programa.

Op_instr = número de operaciones que puede codificar una instrucción.

Unidad 1. Introducción al paralelismo

1.2 Paralelismo

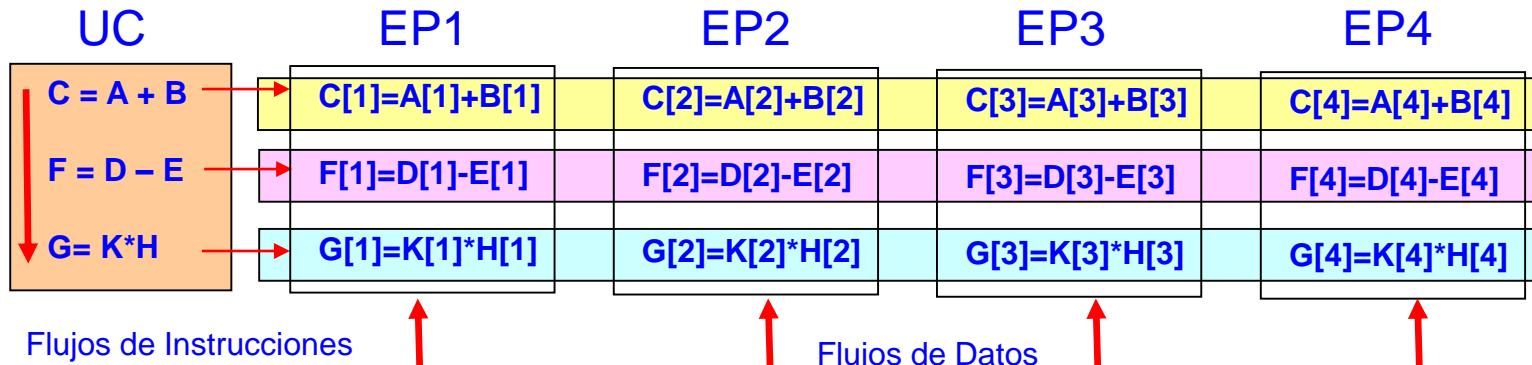
Rendimiento

Ejemplo:

$$NI = \text{Noper}/\text{Op_instr}$$

Ejemplo paralelismo datos

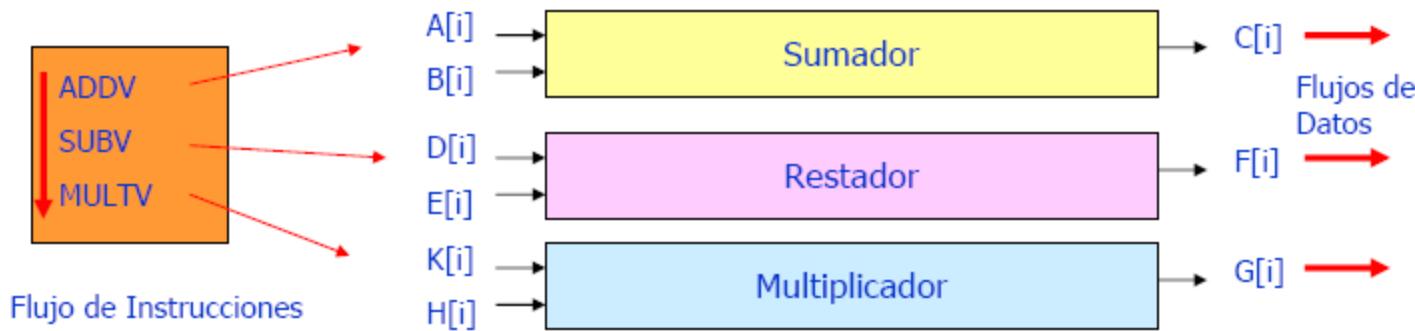
Procesador Matricial



Ejemplo paralelismo instrucciones

Procesador Vectorial

$$\text{Noper}=12 \text{ Op_instr}=4 \text{ NI}=3$$



Unidad 1. Introducción al paralelismo

1.2 Paralelismo

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

Unidad 1. Introducción al paralelismo

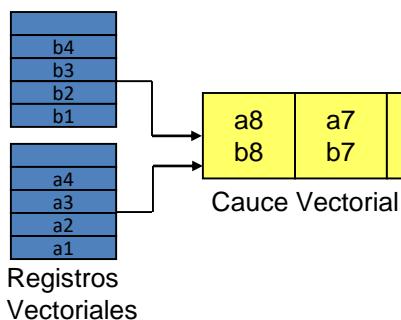
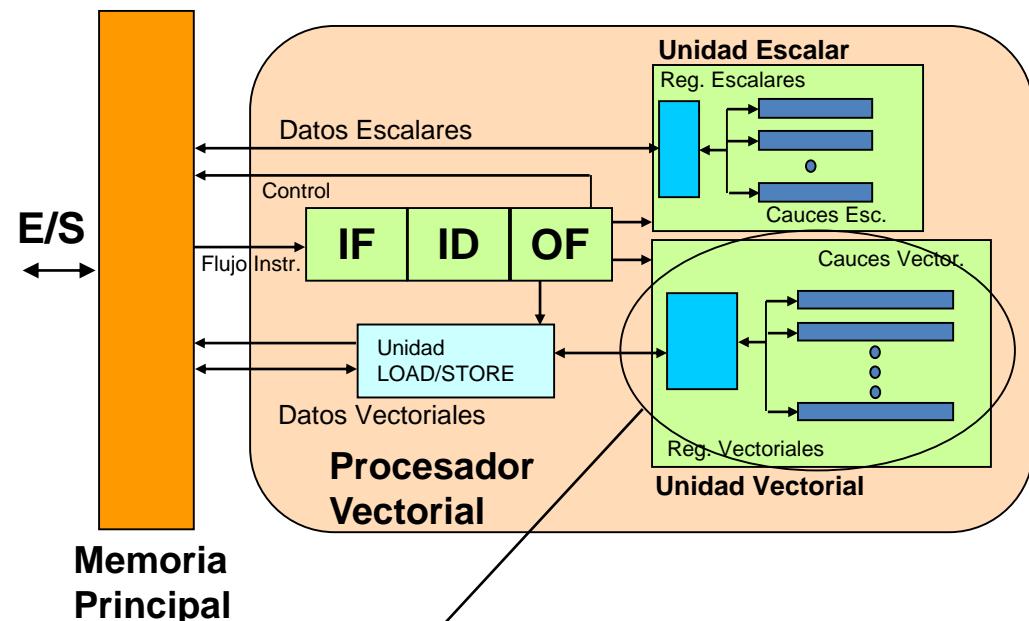
1.3 Arquitecturas vectoriales

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

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

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)

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

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.EQUALINI
```

Pseudo-código vectorial

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

Ensamblador vectorial

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

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

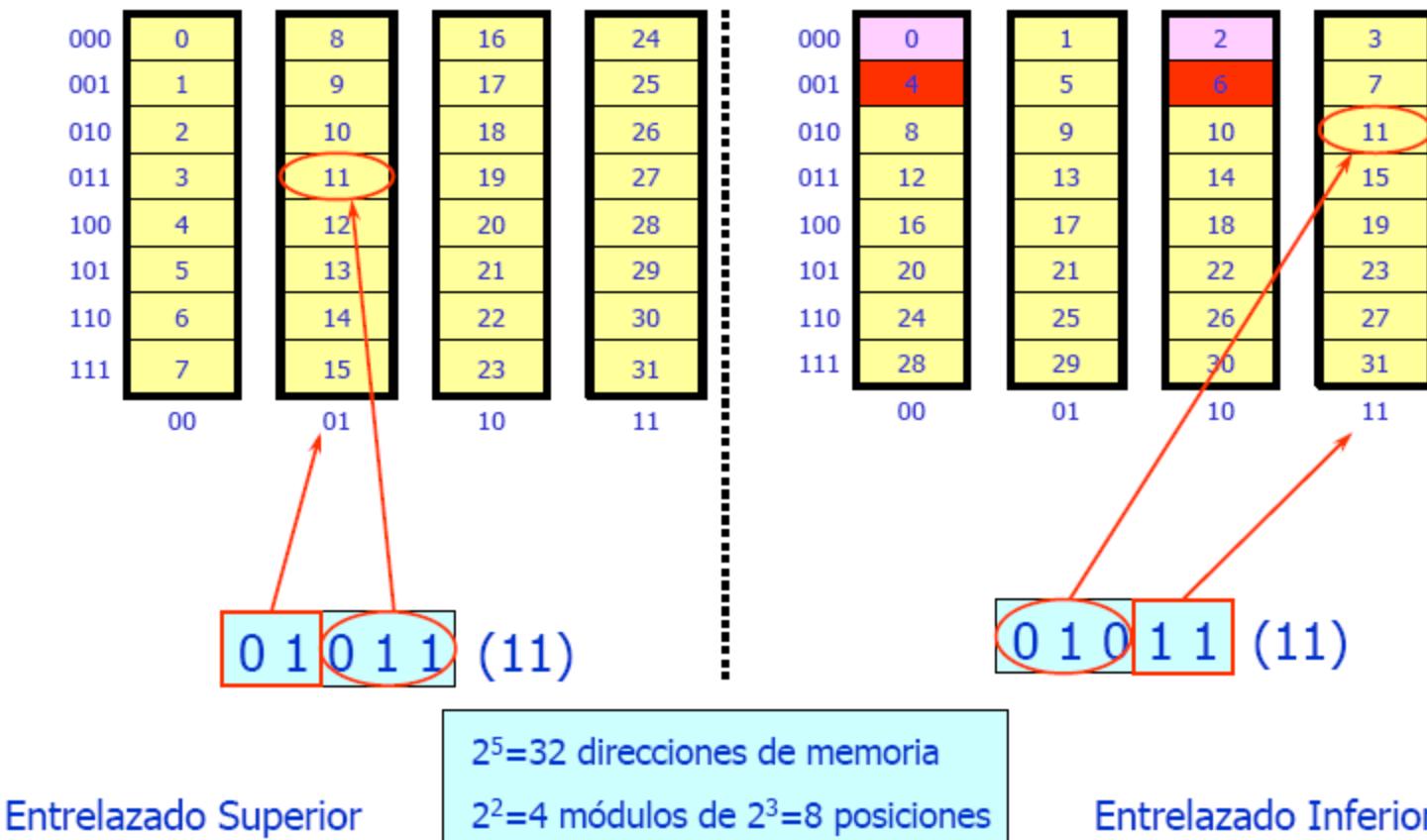
Vector instruction	Operands	Function
ADDV	V1, V2, V3	Add elements of V2 and V3, then put each result in V1.
ADDSV	V1, F0, V2	Add F0 to each element of V2, then put each result in V1.
SUBV	V1, V2, V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS	V1, V2, F0	Subtract F0 from elements 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	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.
MOVSI2	R1, VLR	Move the contents of the vector-length register to R1.
MOVEF2S	VM, F0	Move contents of F0 to the vector-mask register.
MOVFS2F	F0, VM	Move contents of vector-mask register to F0.

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Entrelazado de memoria

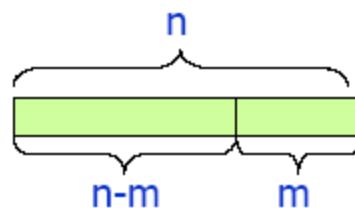
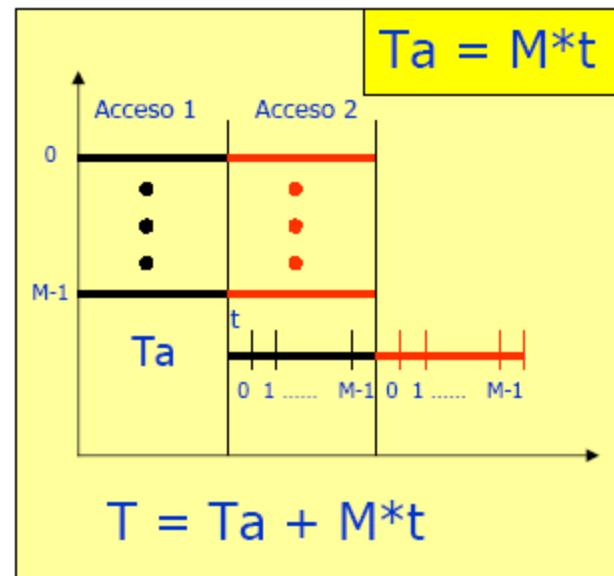
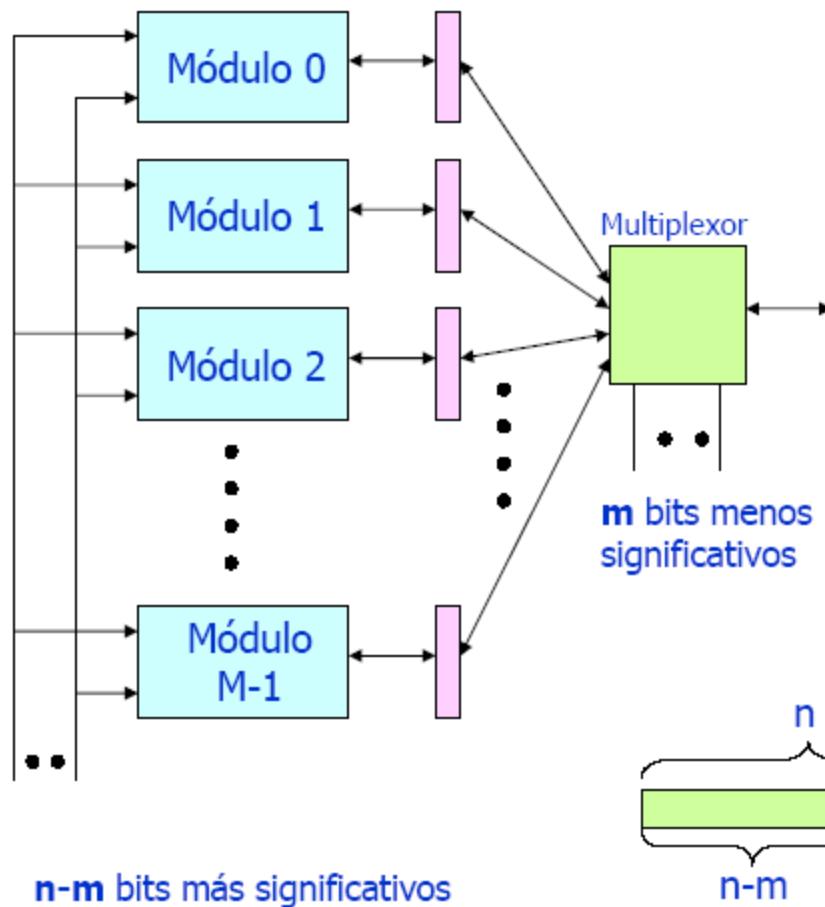


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Acceso a memoria simultáneo o tipo S



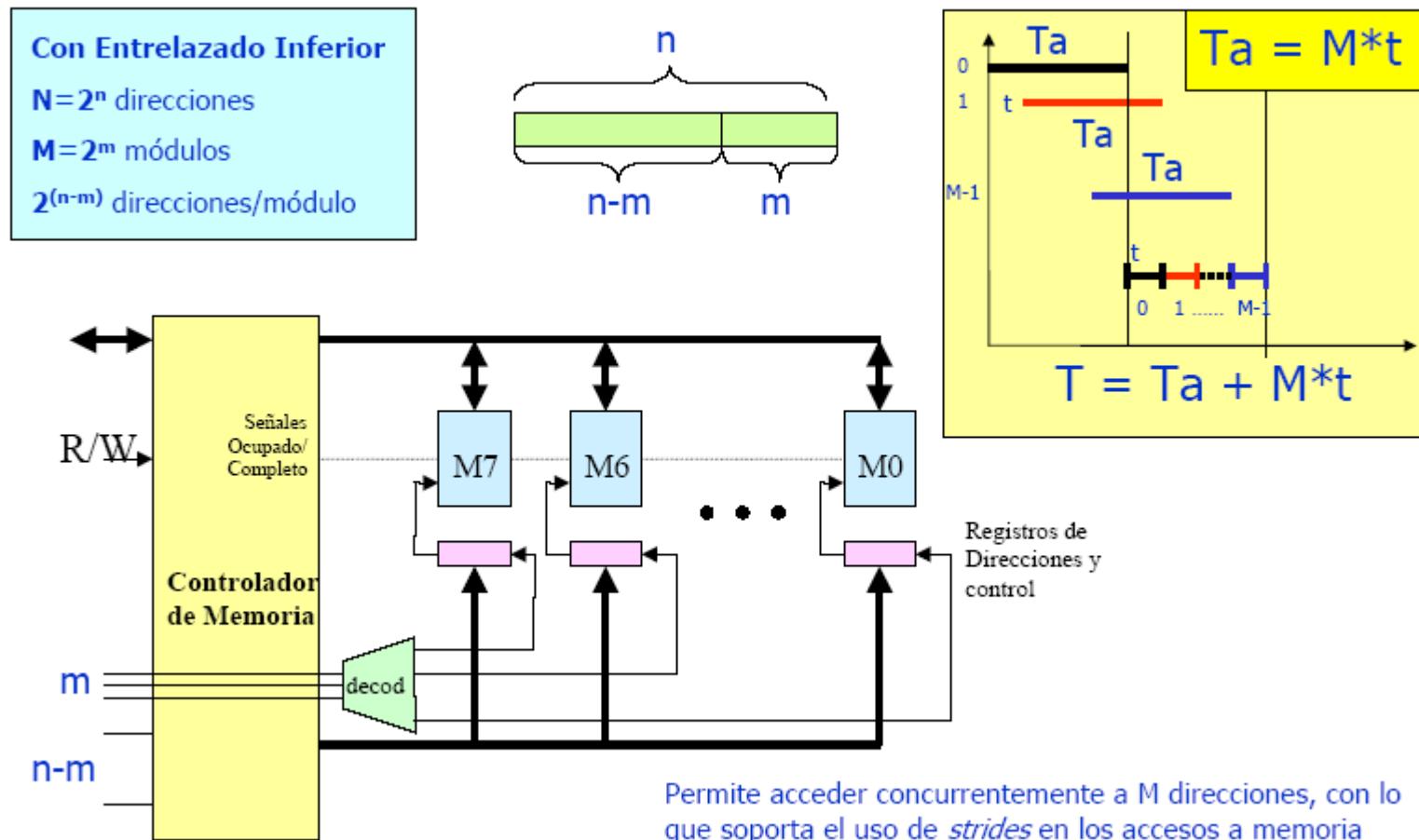
Con Entrelazado Inferior
 $N=2^n$ direcciones
 $M=2^m$ módulos
 $2^{(n-m)}$ direcciones/módulo

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Acceso a memoria concurrente o tipo C

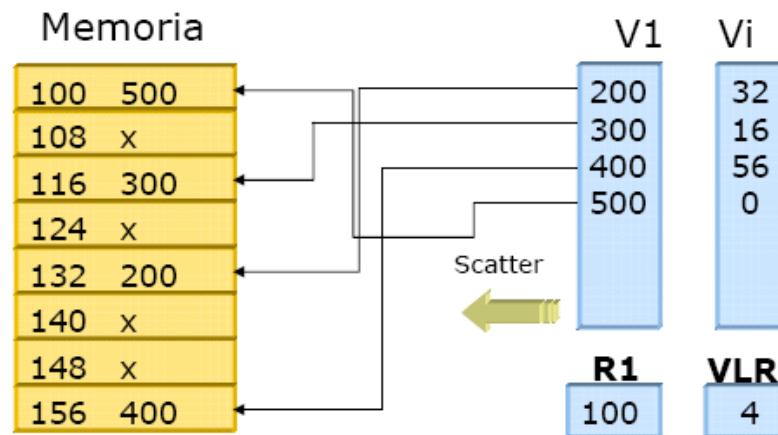
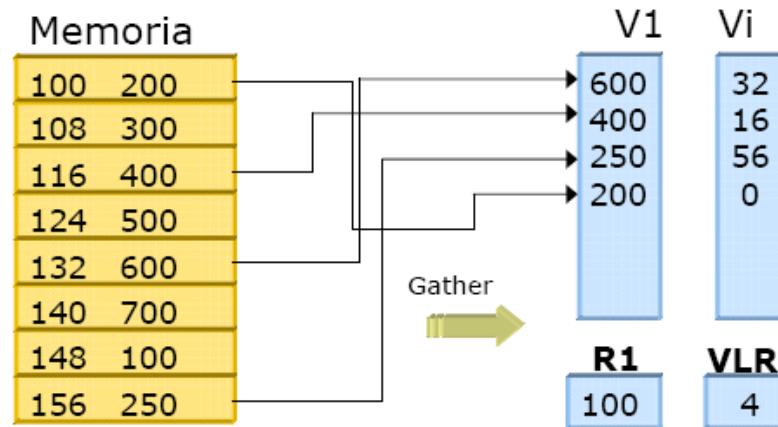


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Operaciones gather-scatter

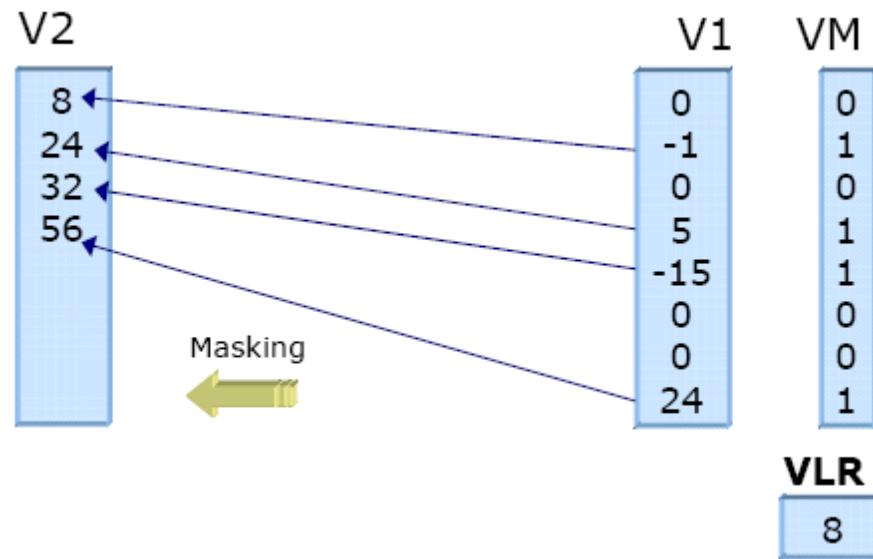


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Enmascaramiento (gestión de matrices dispersas)

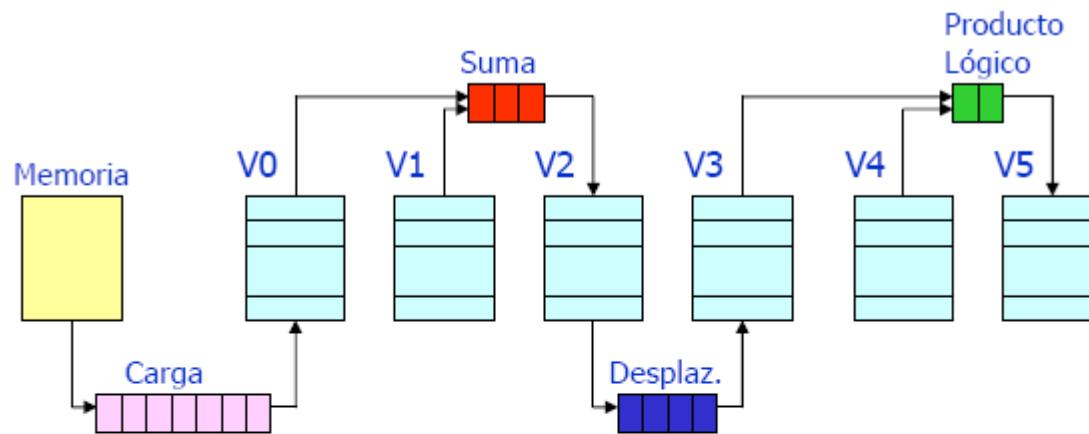


Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales

- Rendimiento: encadenamiento de cauce



$V0 = \text{Load}(\text{Memoria})$	(TLI=7)
$V2 = V0 + V1$	(TLI=3)
$V3 = V1 < A3$	(TLI=4)
$V5 = V3 \wedge V4$	(TLI=2)

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

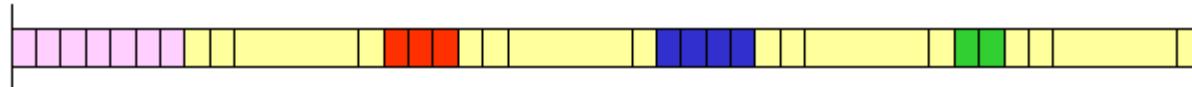
Vectoriales

- Rendimiento: encadenamiento de cauce

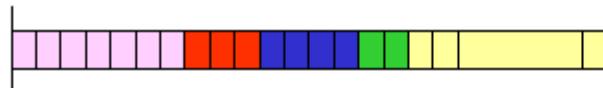


Si se espera que termine una operación vectorial para que empiece otra

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



$$TCV = TLI(\text{carga}) + TLI(\text{suma}) + TLI(\text{desplaz.}) + TLI(\text{Prod.Log}) + K$$

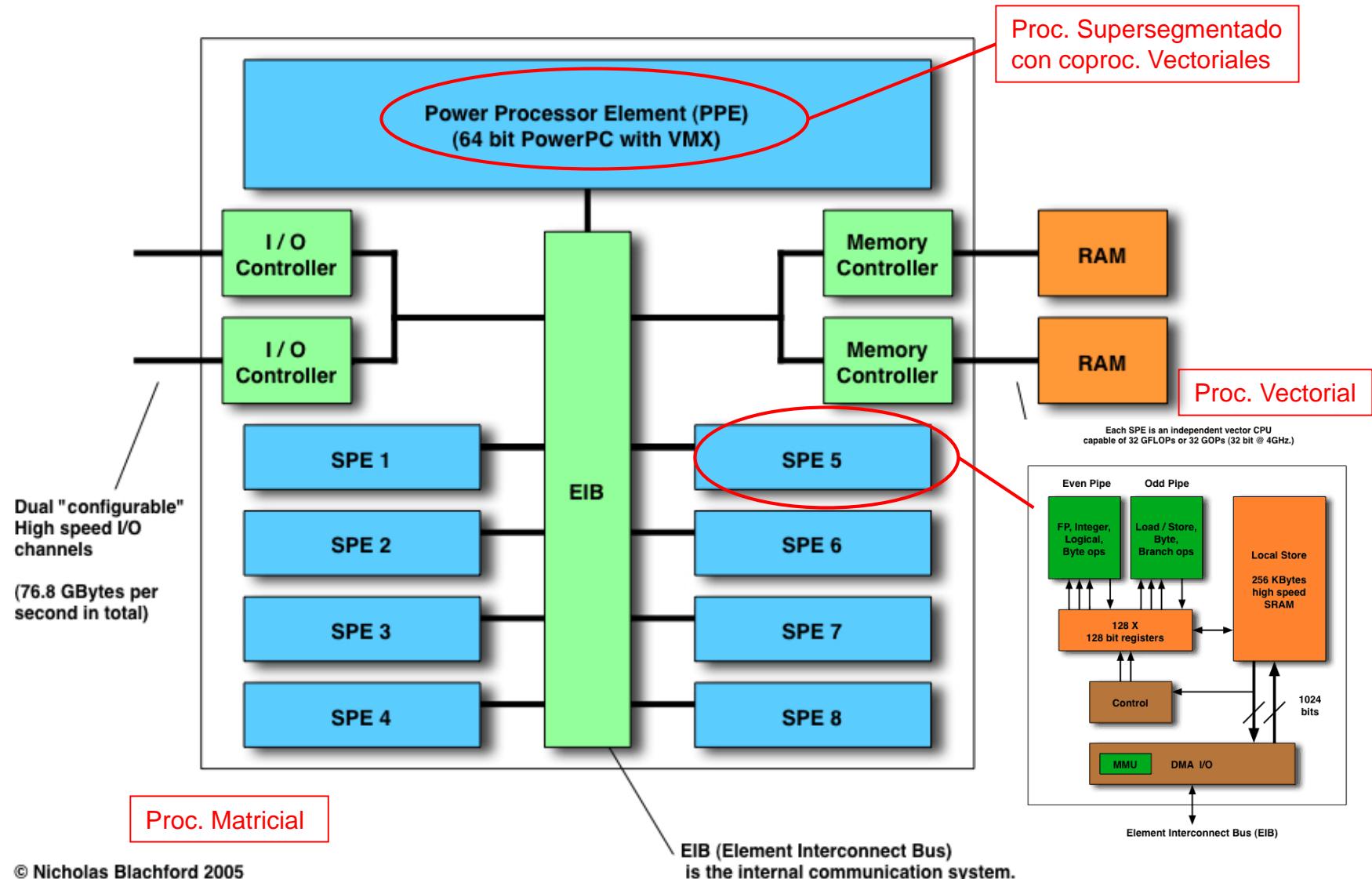


Si se encadenan los cauces de las distintas operaciones

Unidad 1. Introducción al paralelismo

1.3 Arquitecturas vectoriales

Vectoriales



Ingeniería de los Computadores

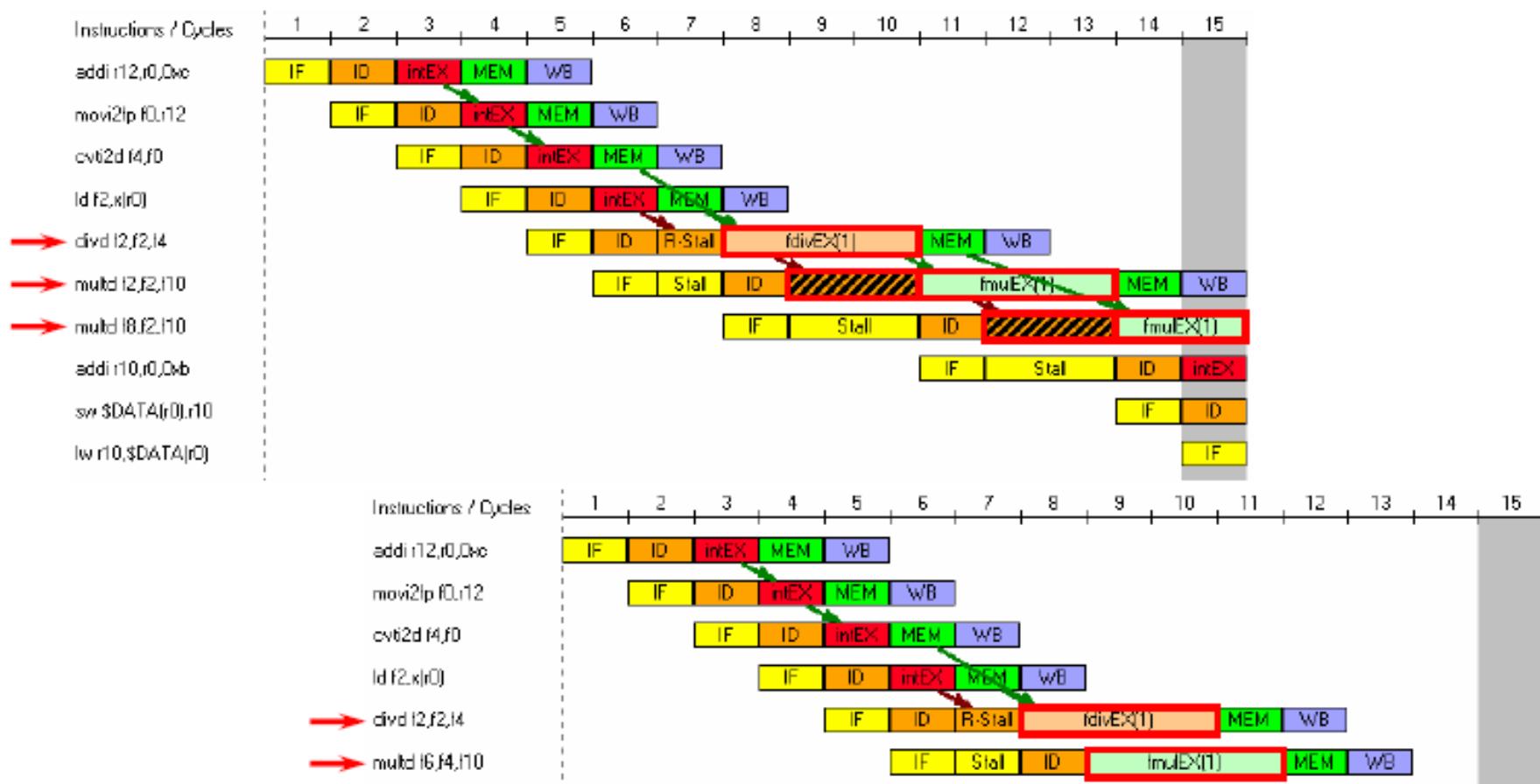
Unidad 2. Superescalares

Unidad 2. Superescalares

2.1. Introducción y motivación

Motivación

- Dependencias estructurales provocan pérdidas de ciclos
 - Ejemplo de una unidad FP vs. Varias unidades FP

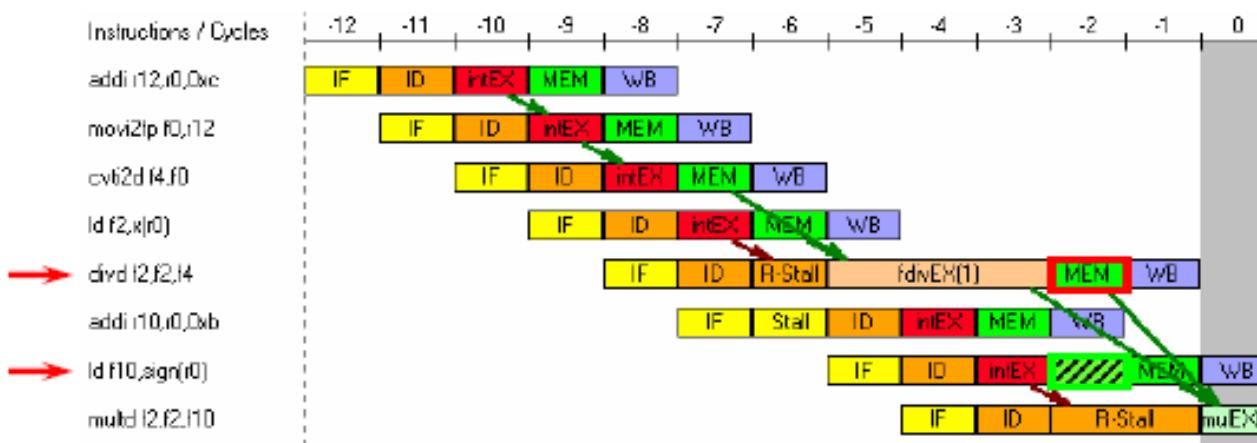


Unidad 2. Superescalares

2.1. Introducción y motivación

Motivación

- Varias unidades funcionales permiten la ejecución fuera de orden
 - Validar riesgos WAR y WAW



- Se obtienen mejores prestaciones si se pueden procesar varias instrucciones en la misma etapa → **procesamiento superescalar**

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

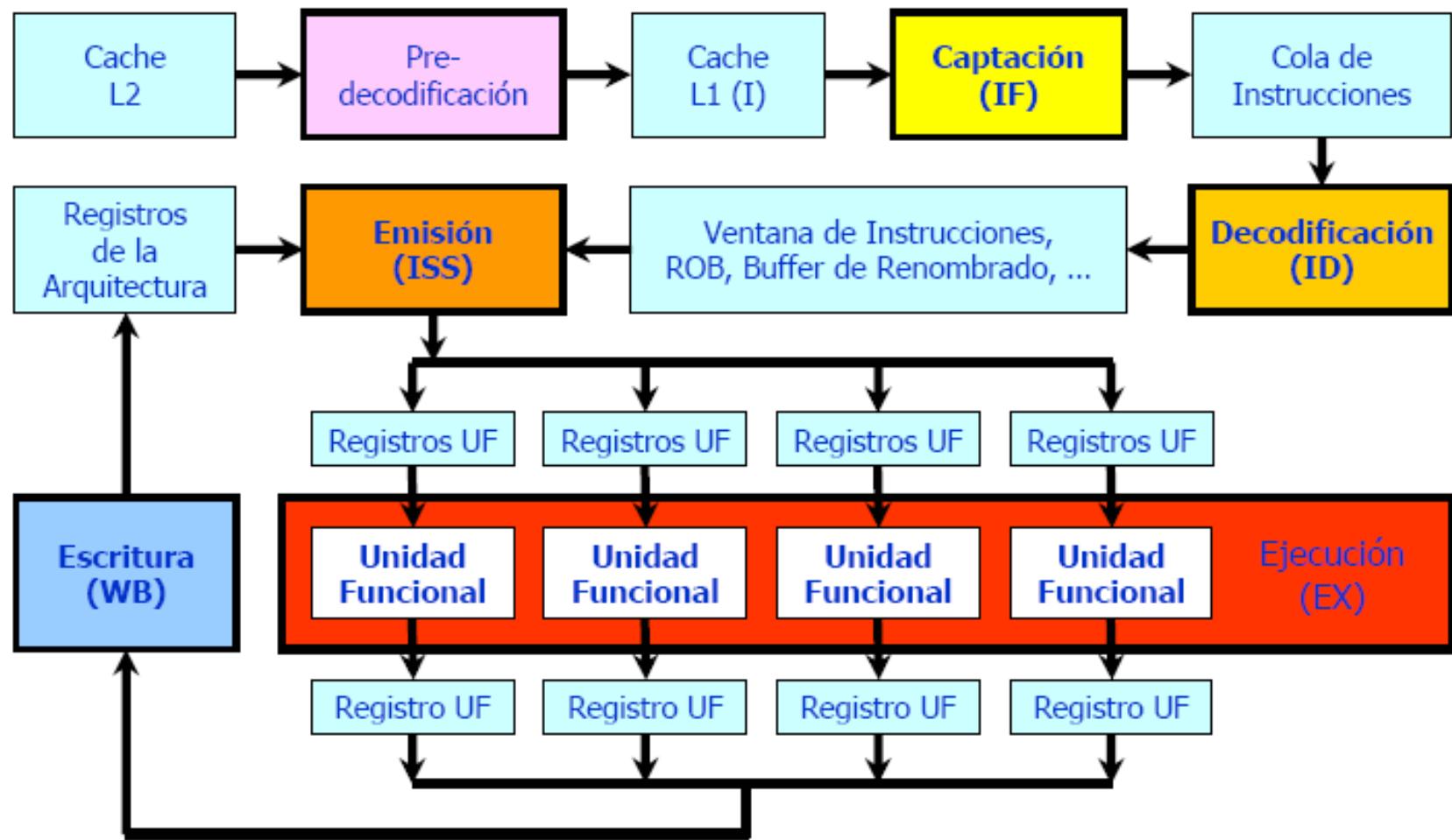
- Etapas
 - Captación de instrucciones (IF)
 - Decodificación de instrucciones (ID)
 - Emisión de instrucciones (ISS)
 - Ejecución de instrucciones (EX) – Instrucción finalizada o “finish”
 - Escritura (WB) – Instrucción completada o “complete”
- Características del procesamiento superescalar
 - Diferentes tipos de órdenes: orden de captación, orden de emisión, orden de finalización
 - Capacidad para identificar ILP existente y organizar el uso de las distintas etapas para optimizar recursos

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

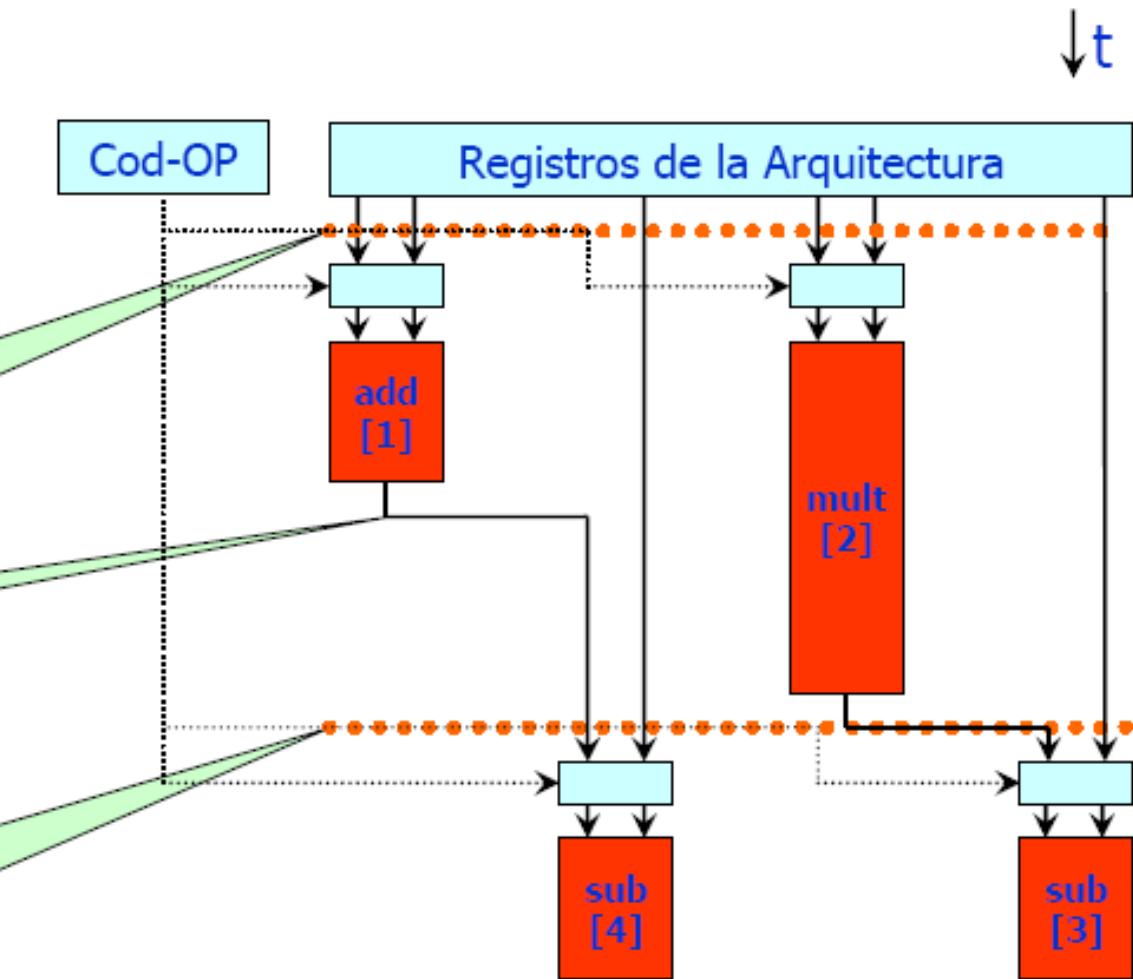
- Emisión ordenada

<i>i</i>	Instr. <i>i</i>	Latencia
[1]	add r4,r1,r2	(2)
[2]	mult r5,r1,r5	(5)
[3]	sub r6,r5,r2	(2)
[4]	sub r5,r4,r3	(2)

Emisión de [1] y [2] (tienen sus operandos)

Aunque [4] tiene sus operandos debe esperar

Emisión de [3] (tiene sus operandos al terminar [2]) y [4]



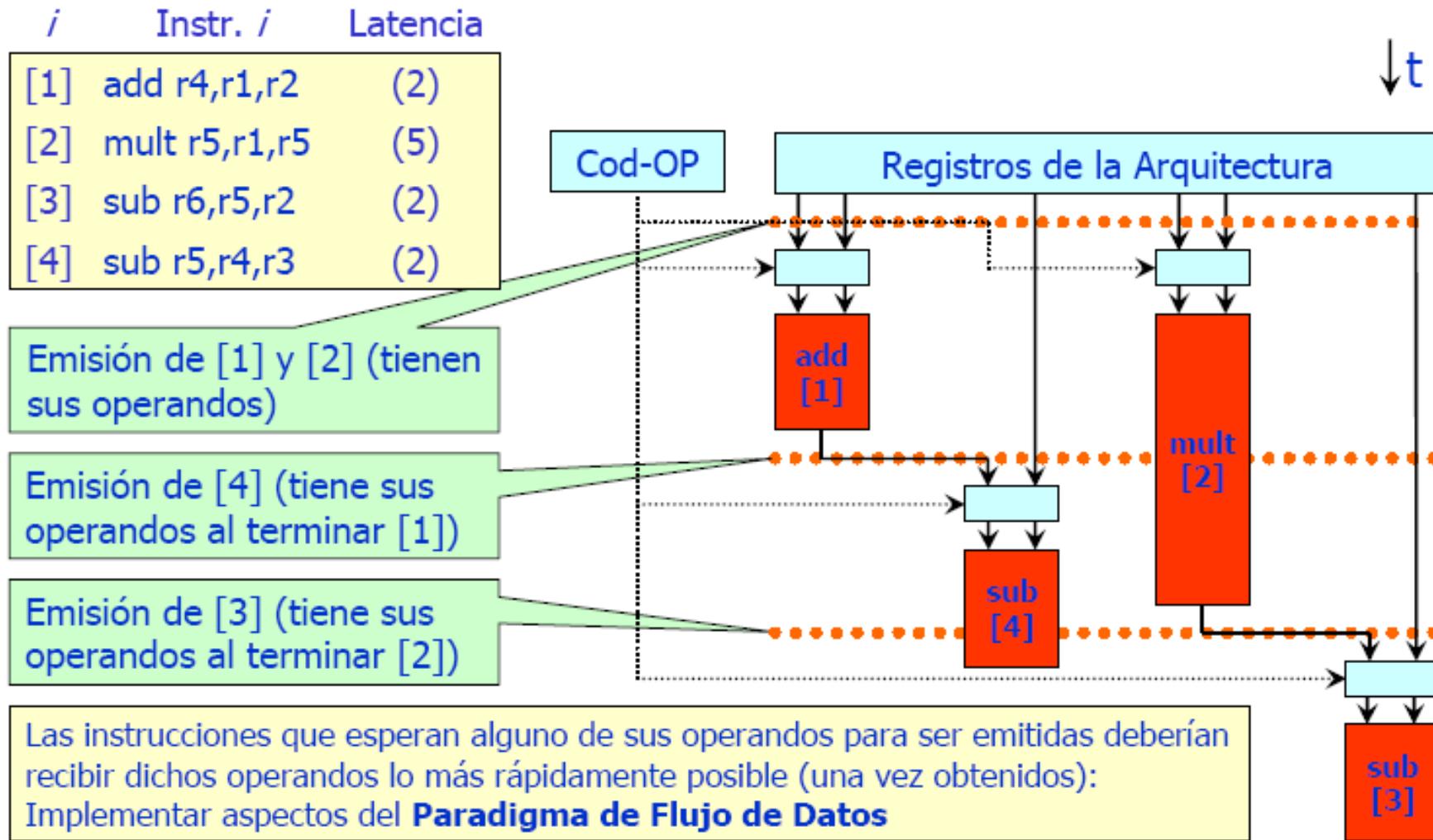
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Emisión desordenada



Unidad 2. Superescalares

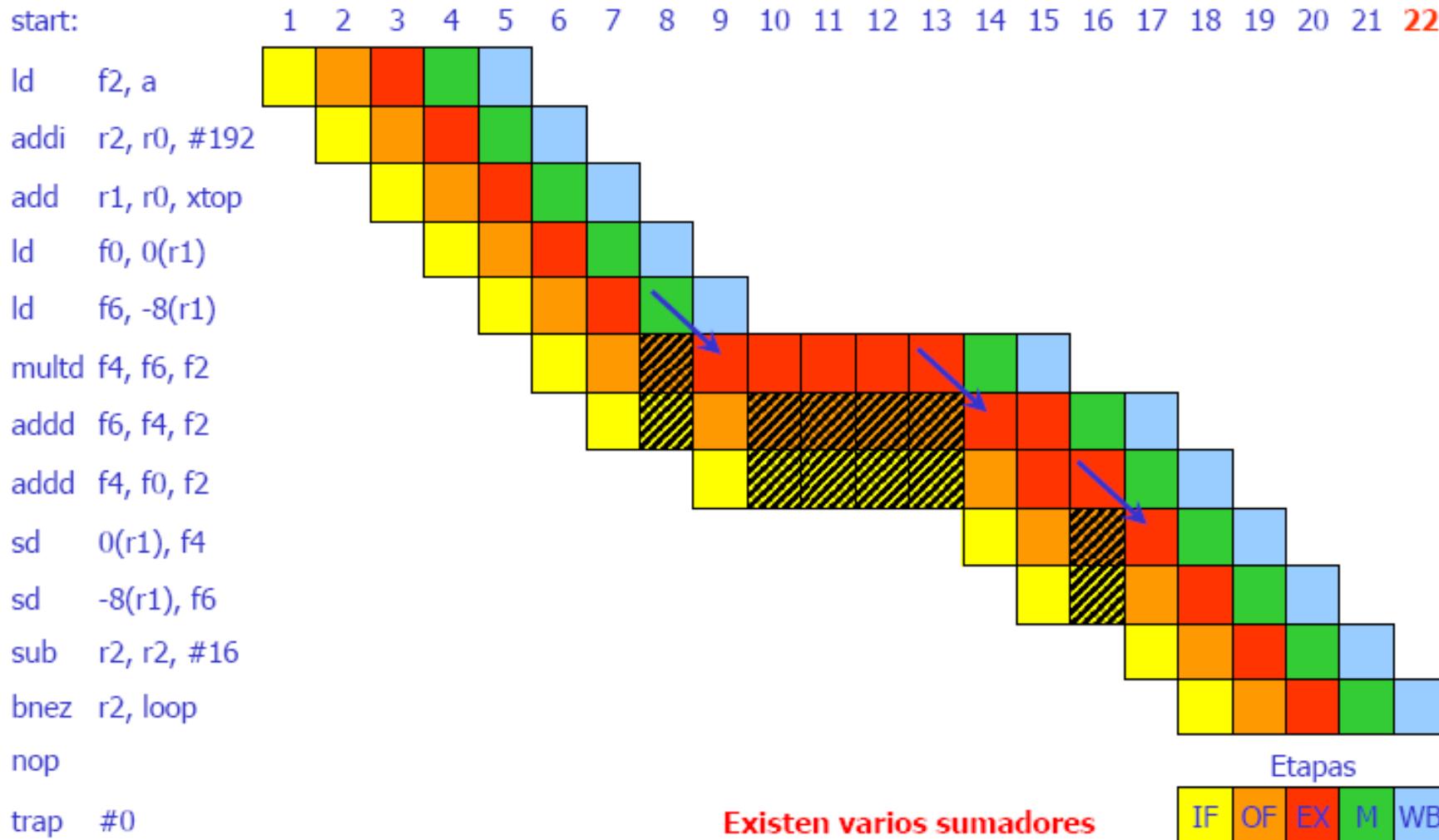
2.2. Cauce superescalar

Motivación

Cauce

- Procesamiento segmentado

start:



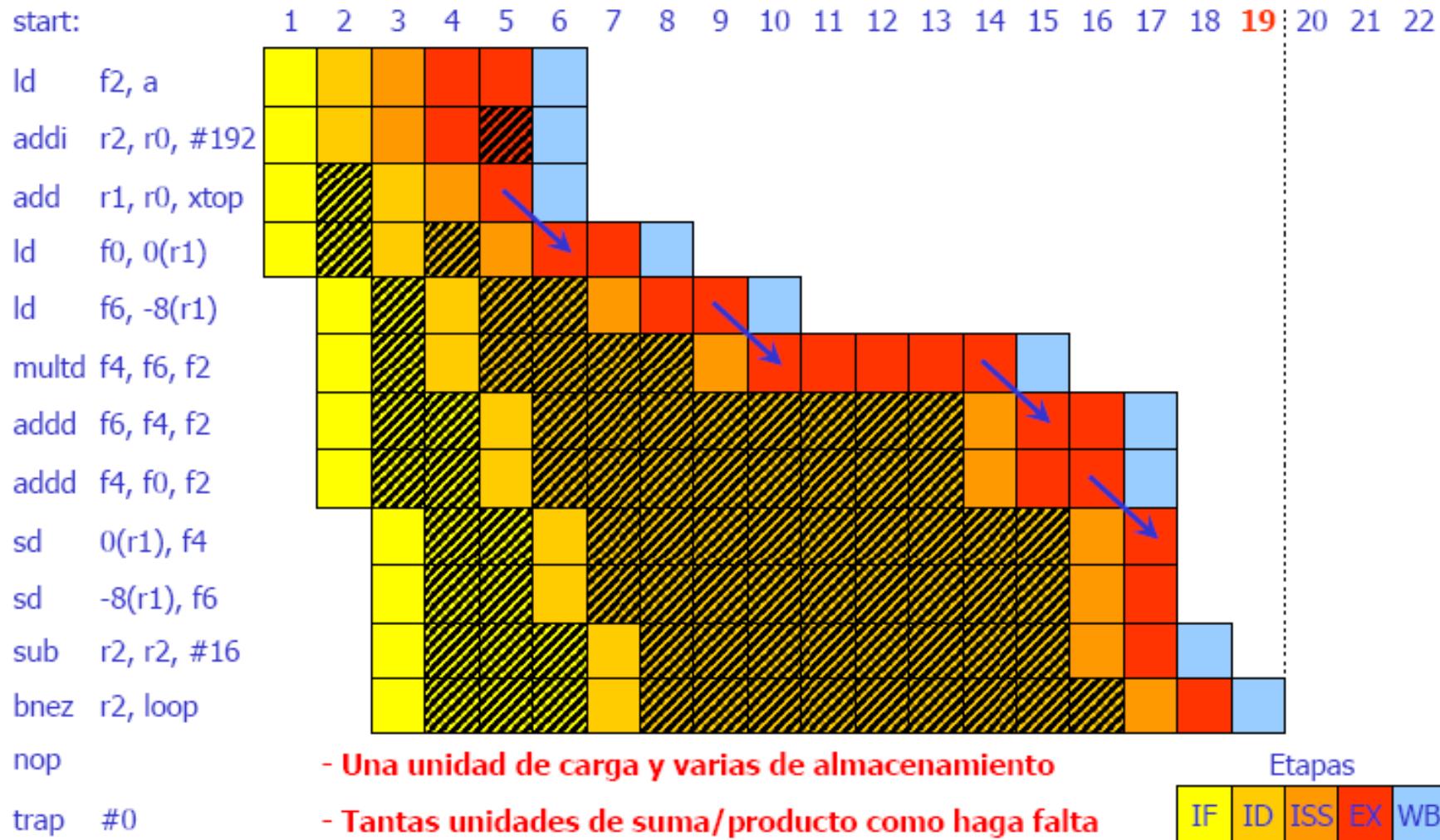
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión ordenada/finalización ordenada



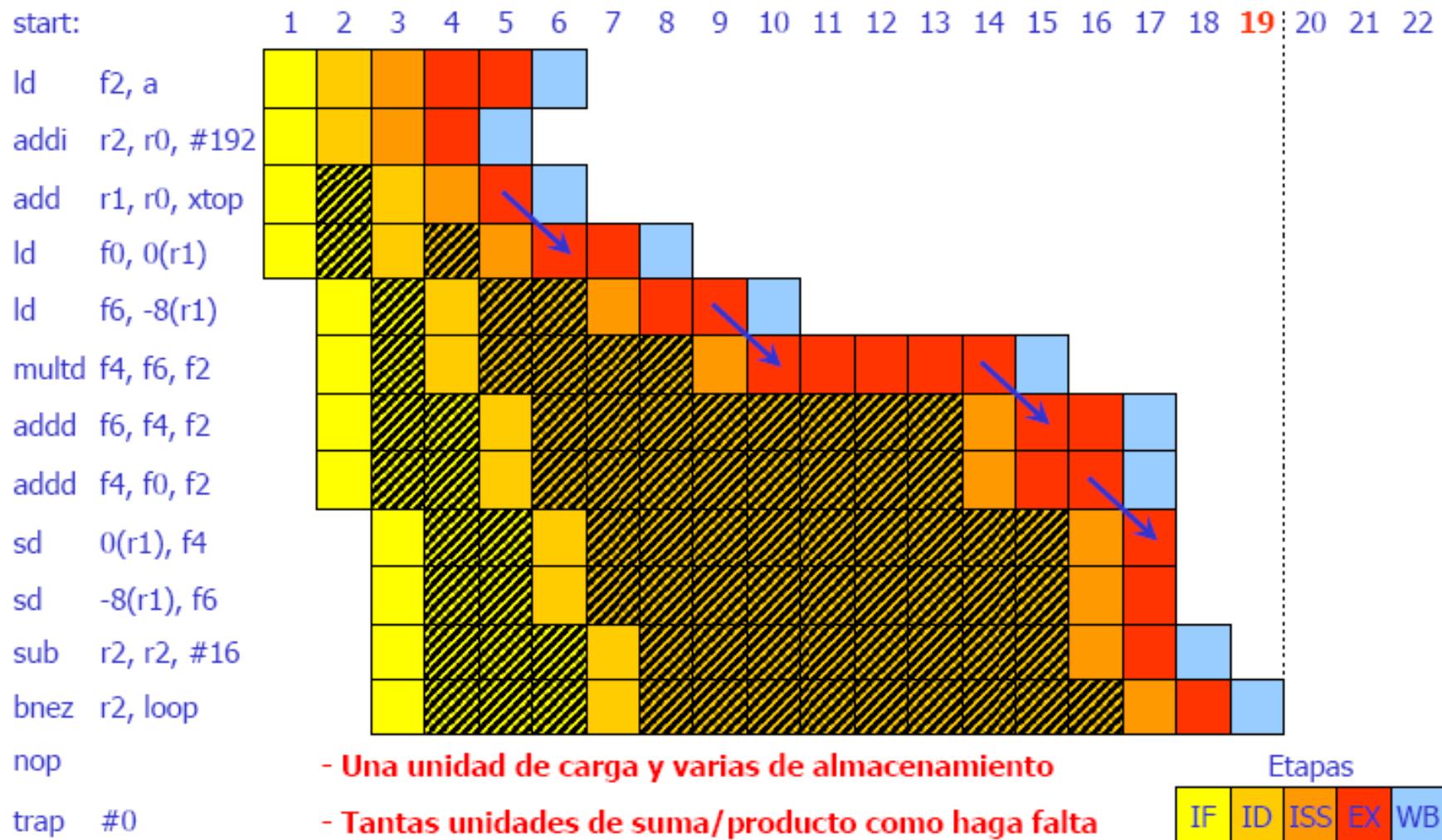
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión ordenada/finalización desordenada



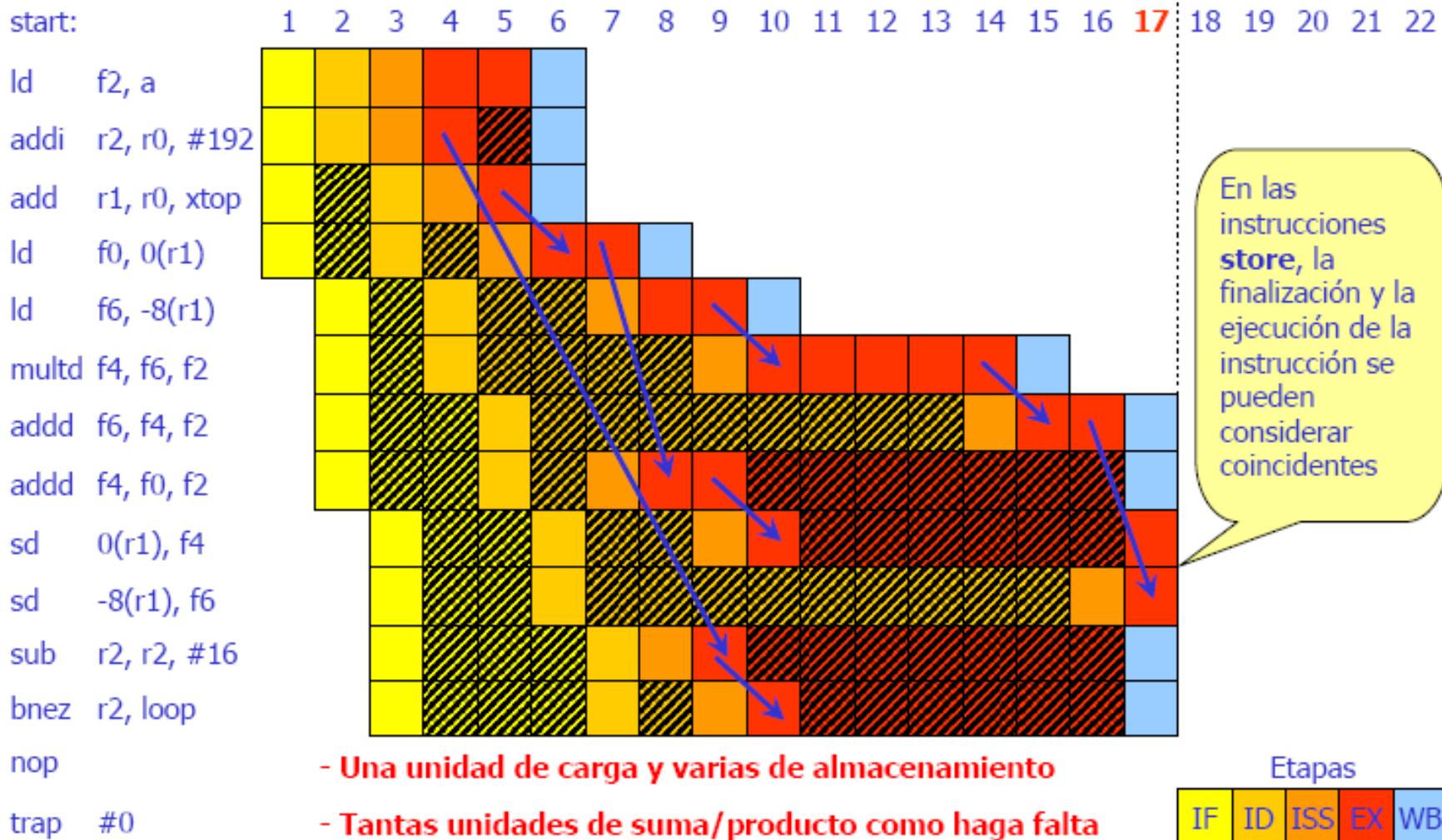
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión desordenada/finalización ordenada



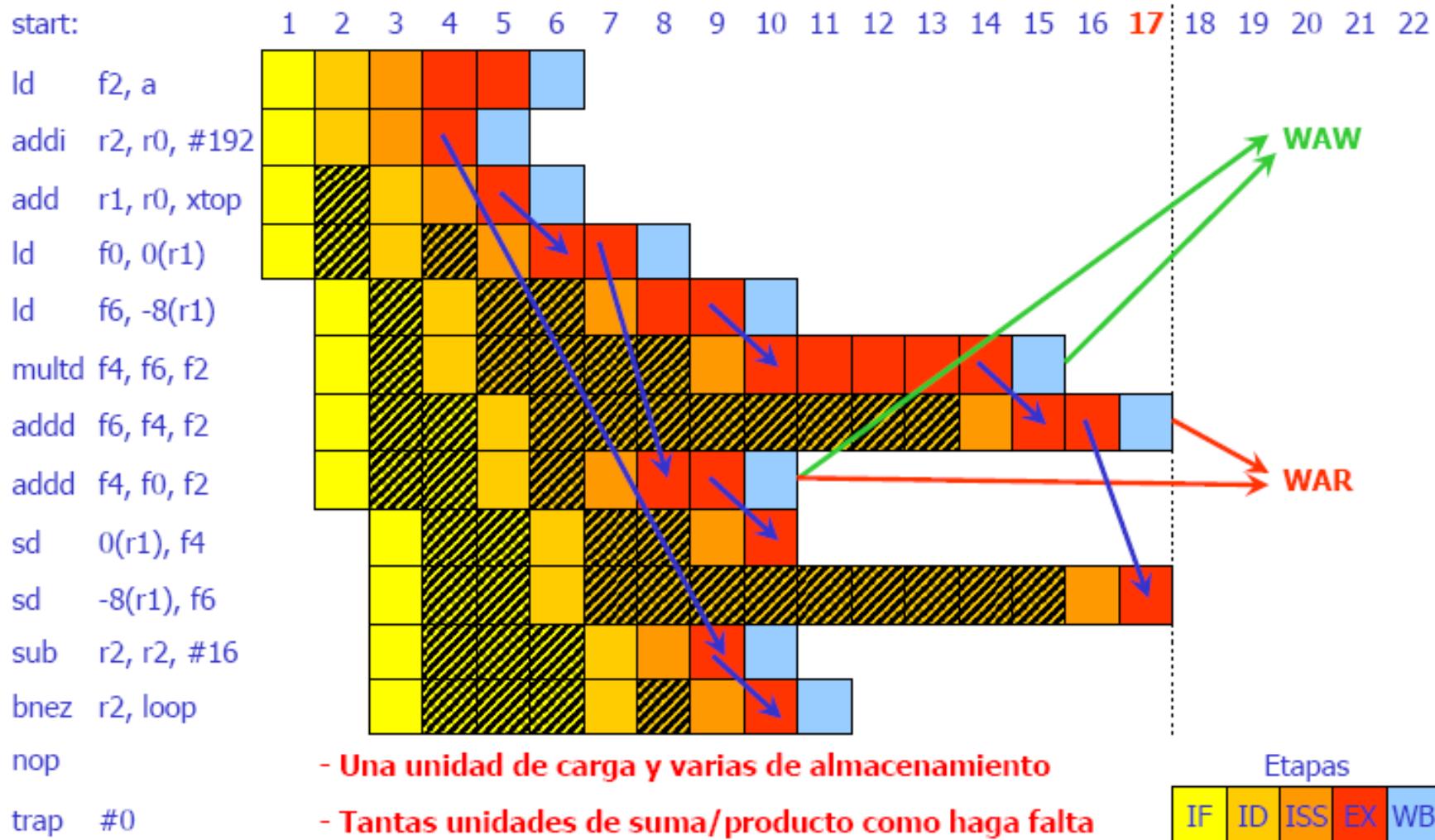
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

- Superescalar: emisión desordenada/finalización desordenada



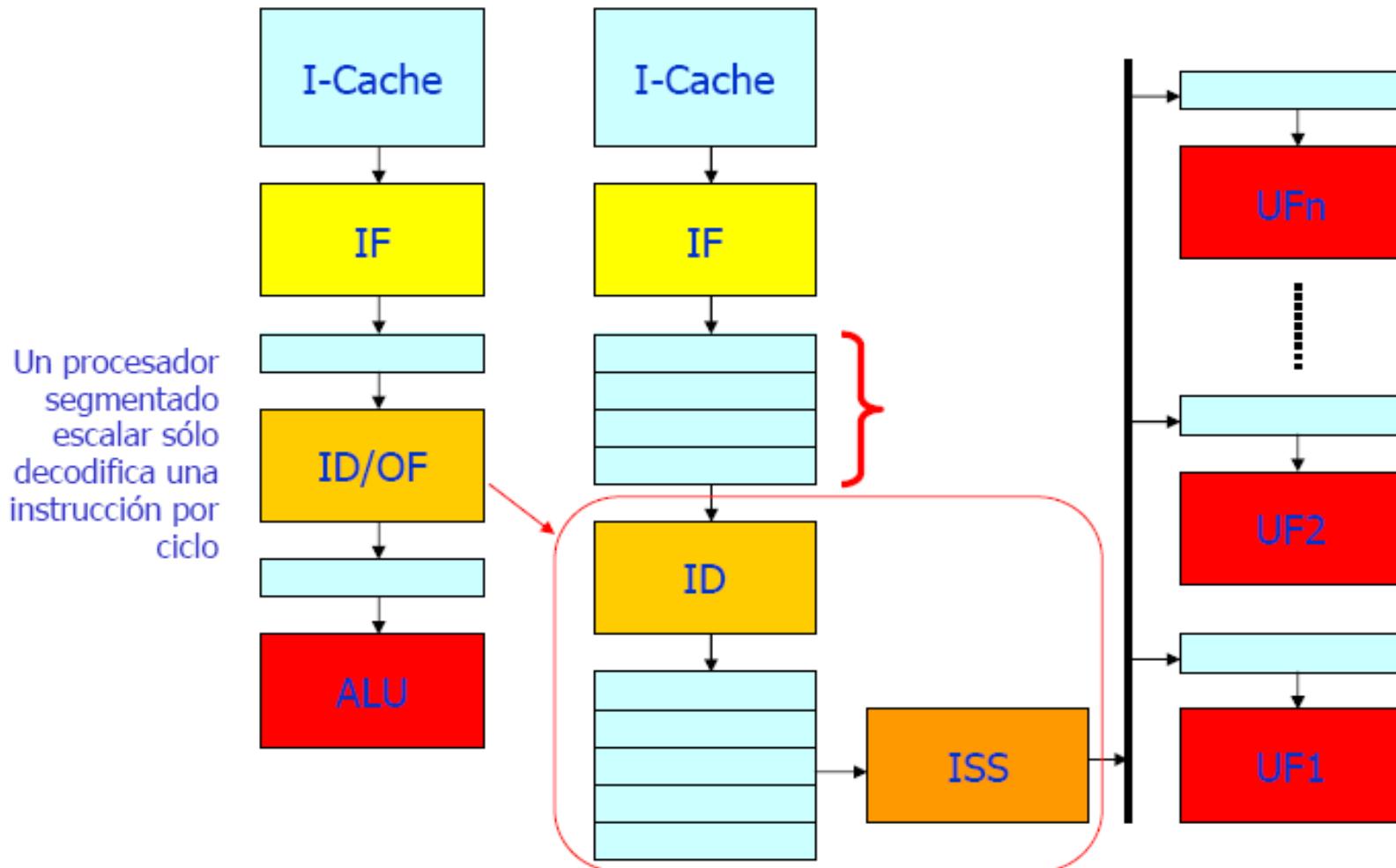
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación



En un procesador superescalar se han de decodificar varias instrucciones por ciclo (y comprobar las dependencias con las instrucciones que se están ejecutando)

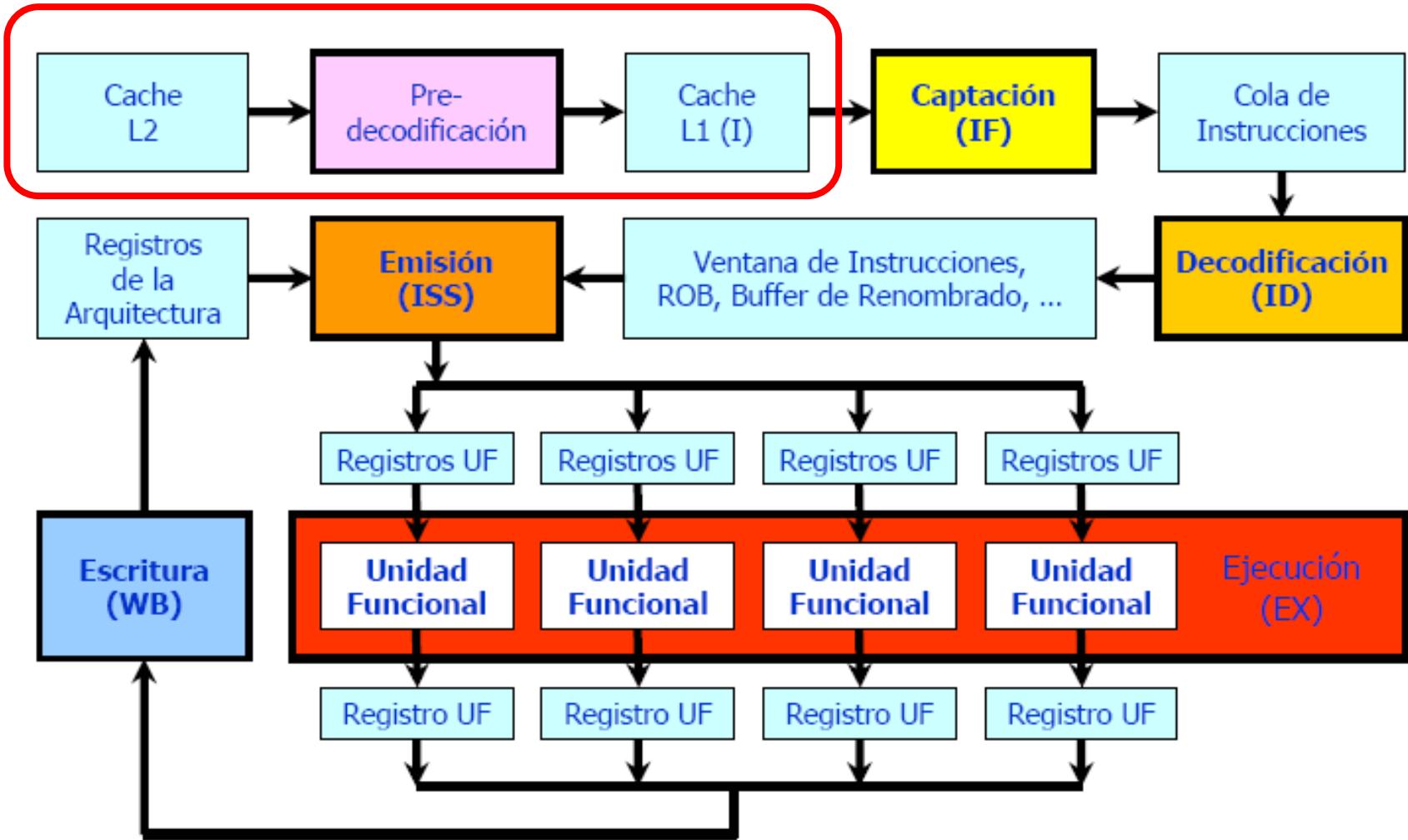
Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

- Bits de predecodificación pueden indicar:
 - Si es una instrucción de salto o no (se puede empezar su procesamiento antes)
 - El tipo de unidad funcional que va a utilizar (se puede emitir más rápidamente si hay cauces para enteros o coma flotante...)
 - Si hace referencia a memoria o no

Unidad 2. Superescalares

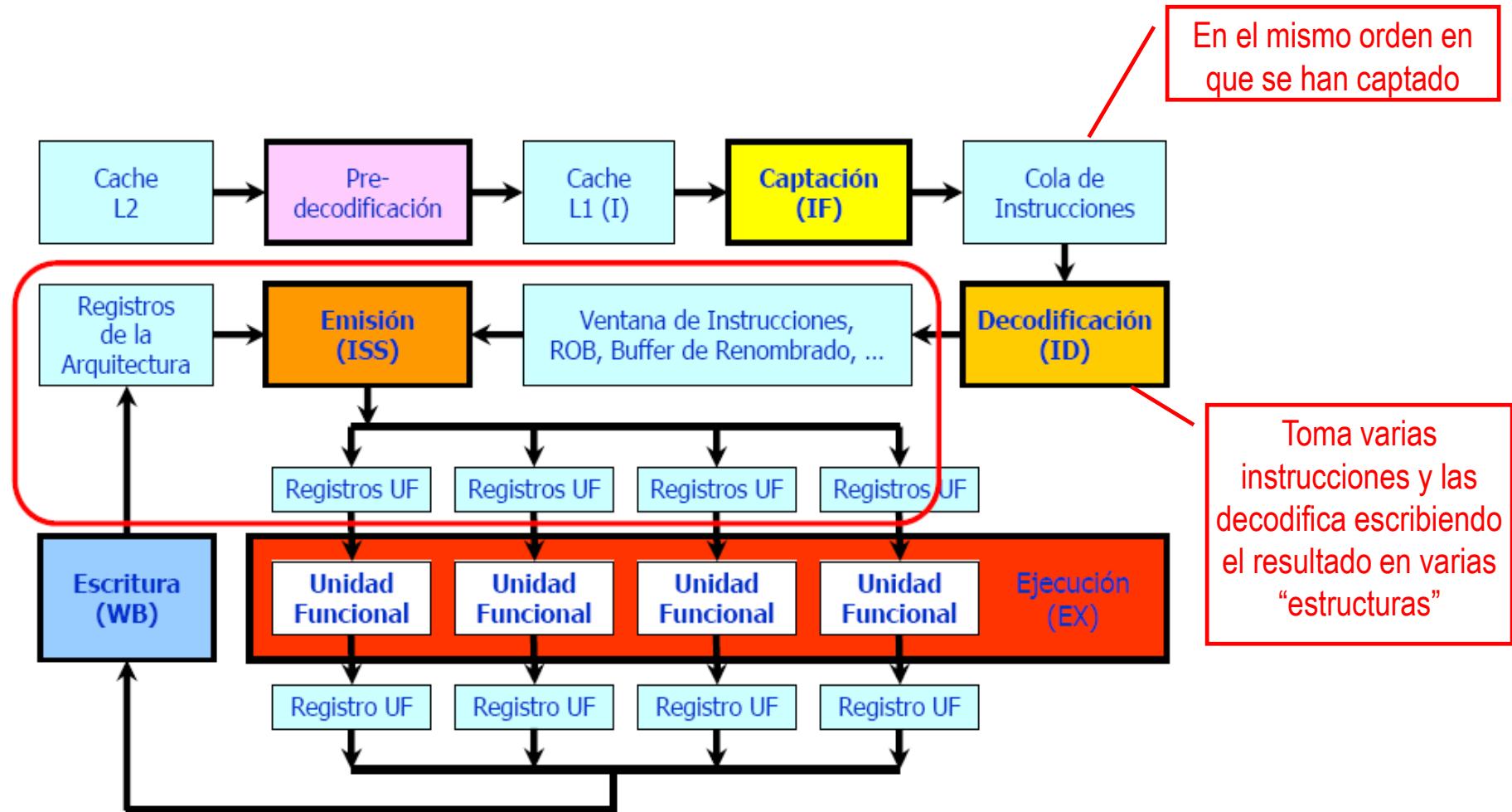
2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Ventana de instrucciones:

- La ventana de instrucciones almacena las instrucciones pendientes (todas, si la ventana es centralizada o las de un tipo determinado, si es distribuida)
- Las instrucciones se cargan en la ventana una vez decodificadas y se utiliza un bit para indicar si un operando está disponible (se almacena el valor o se indica el registro desde donde se lee) o no (se almacena la unidad funcional desde donde llegará el operando)
- Una instrucción puede ser emitida cuando tiene todos sus operandos disponibles y la unidad funcional donde se procesará. Hay diversas posibilidades para el caso en el que varias instrucciones estén disponibles (características de los buses, etc.)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

Ejemplo de Ventana de Instrucciones

#	opcode	address	rb_entry	operand1	ok1	typ1	operand2	ok2	typ2	pred
2	MULTD	loop + 0x4	2	1	0	FPD	0	0	FPD	-
1	LD	loop	1	0	0	INT	0	1	IMM	-

Lugar donde se
almacenará el resultado

Dato no válido
(indica desde dónde se recibirá el dato)

Dato válido
(igual a 0)

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

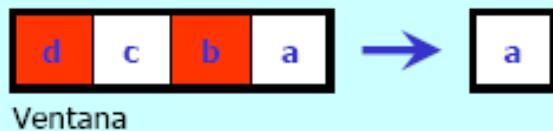
- **Orden:** Emisión Ordenada o Desordenada
- **Alineamiento:** Emisión Alineada o No alineada

Ejemplos:

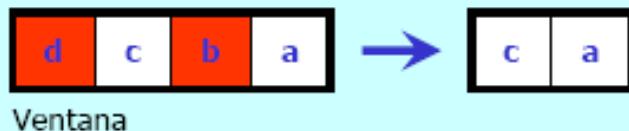
Alineada: SuperSparc (92), PowerPC (93, 95, 96), PA8000 (96), Alpha (92, 94, 95), R10000 (96)

No Alineada: MC88110 (93), PA7100LC (93), R8000 (94), UltraSparc (95)

Emisión Ordenada

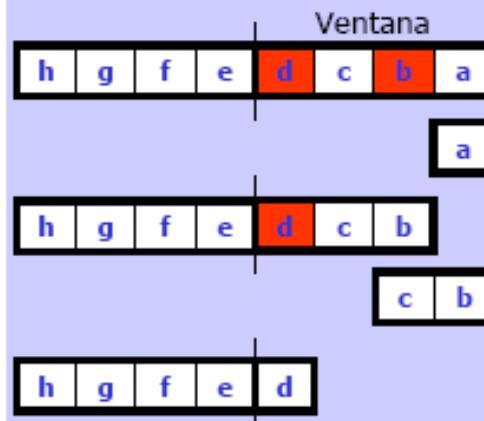


Emisión Desordenada



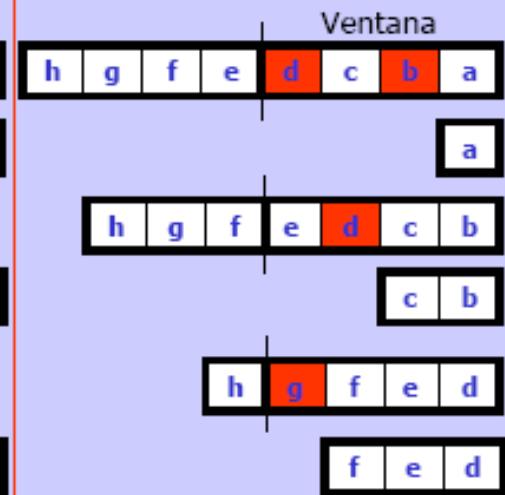
Instrucción no preparada para la emisión

Emisión Alineada



Ordenada

Emisión No alineada



Ordenada

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

add/sub: 2
mult: 1

Ventana de Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

[4] puede emitirse pero debe esperar a la [3]

add/sub: 2
mult: 1

Ventana de Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(7)	(8)-(9)

Ha terminado [2]: pueden emitirse [3] y [4]

sub r5 [r4] 1 - [r3] 1 -
sub r6 [r5] 1 - [r2] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -



Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -
sub r6 r5 0 mult [r2] 1 -



sub r6 [r5] 1 - [r2] 1 -

add/sub: 2
mult: 1

Ventana de
Registros

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)

No afecta
al tiempo

Se ha emitido [4] y ha terminado
[2]: puede emitirse [3]

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

↓ Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

↓ Ha terminado [1]

sub r5 [r4] 1 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -

Ventana de Registros

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(10)	(11)-(12)

Ha terminado [2] y se ha emitido [3].
Cuando la unidad quede libre se emitirá [4]



sub r5 [r4] 1 - [r3] 1 -

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

sub r5 r4 0 add [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		
mult r5 [r1] 1 - [r5] 1 -		
add r4 [r1] 1 - [r2] 1 -		



Se han emitido [1] y [2]

sub r5 r4 0 add [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		



Ha terminado [1]

sub r5 [r4] 1 - [r3] 1 -		
sub r6 r5 0 mult [r2] 1 -		

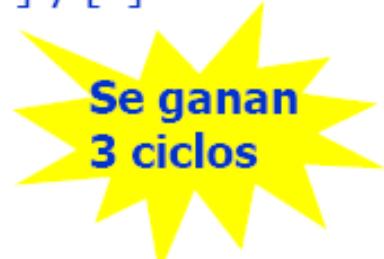


Ventana de Registros

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)

Instrucc.	ISS	EXE
add	(1)	(2)-(3)
mult	(1)	(2)-(6)
sub	(7)	(8)-(9)
sub	(4)	(5)-(6)



Se ha emitido [4] y ha terminado [2]: puede emitirse [3]

sub r6 [r5] 1 - [r2] 1 -		

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva

sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

Cola de
Instrucciones

add/sub: 1
mult: 1

[1] add r4,r1,r2 (2)
[2] mult r5,r1,r5 (5)
[3] sub r6,r5,r2 (2)
[4] sub r5,r4,r3 (2)



sub r5 r4 0 add [r3] 1 -
sub r6 r5 0 mult [r2] 1 -
mult r5 [r1] 1 - [r5] 1 -
add r4 [r1] 1 - [r2] 1 -

ID/ISS

sub r6 r5 0 mult [r2] 1 -
add r4 [r1] 1 - [r2] 1 -

Estaciones de Reserva
(con capacidad para dos
instrucciones)

mult r5 [r1] 1 - [r5] 1 -

Cola de
Instrucciones

Unidad 2. Superescalares

2.2. Cauce superescalar

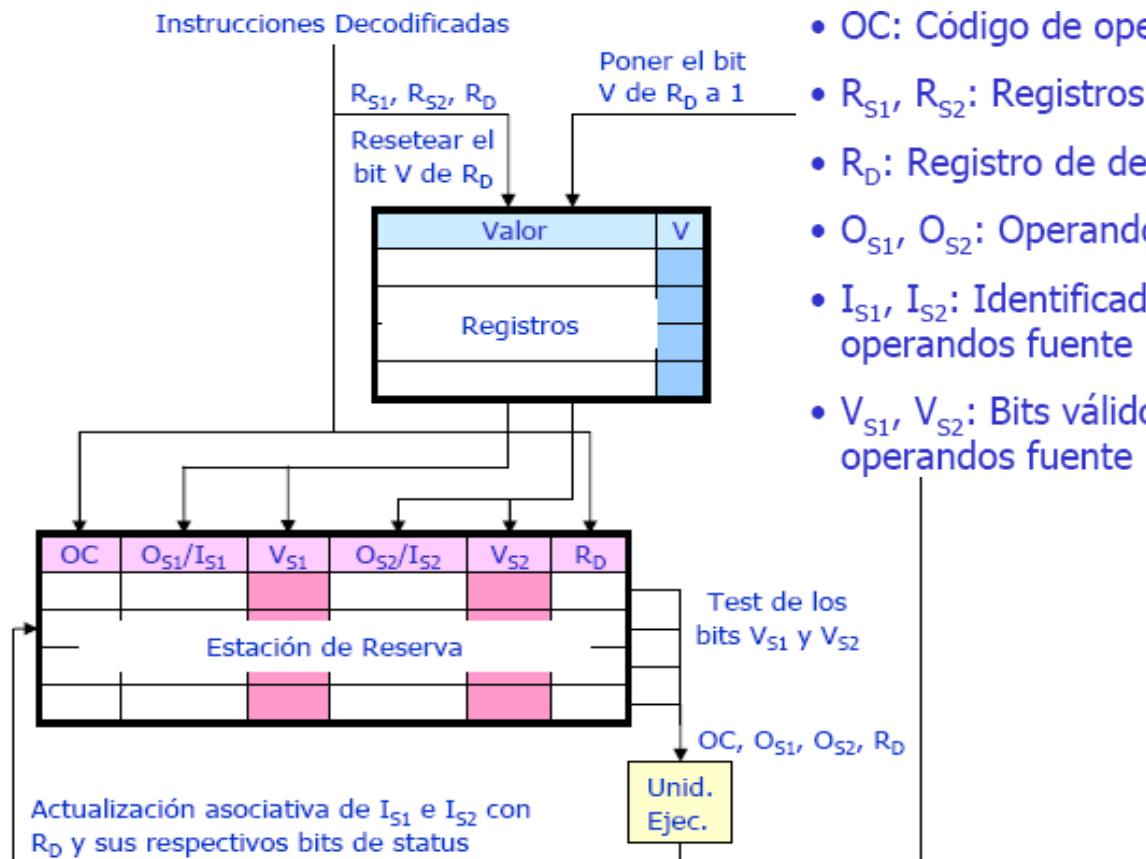
Motivación

Cauce

Decodificación

Emisión

- Estaciones de reserva



- OC: Código de operación
- R_{S1}, R_{S2} : Registros fuente
- R_D : Registro de destino
- O_{S1}, O_{S2} : Operandos fuente
- I_{S1}, I_{S2} : Identificadores de los operandos fuente
- V_{S1}, V_{S2} : Bits válidos de los operandos fuente

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

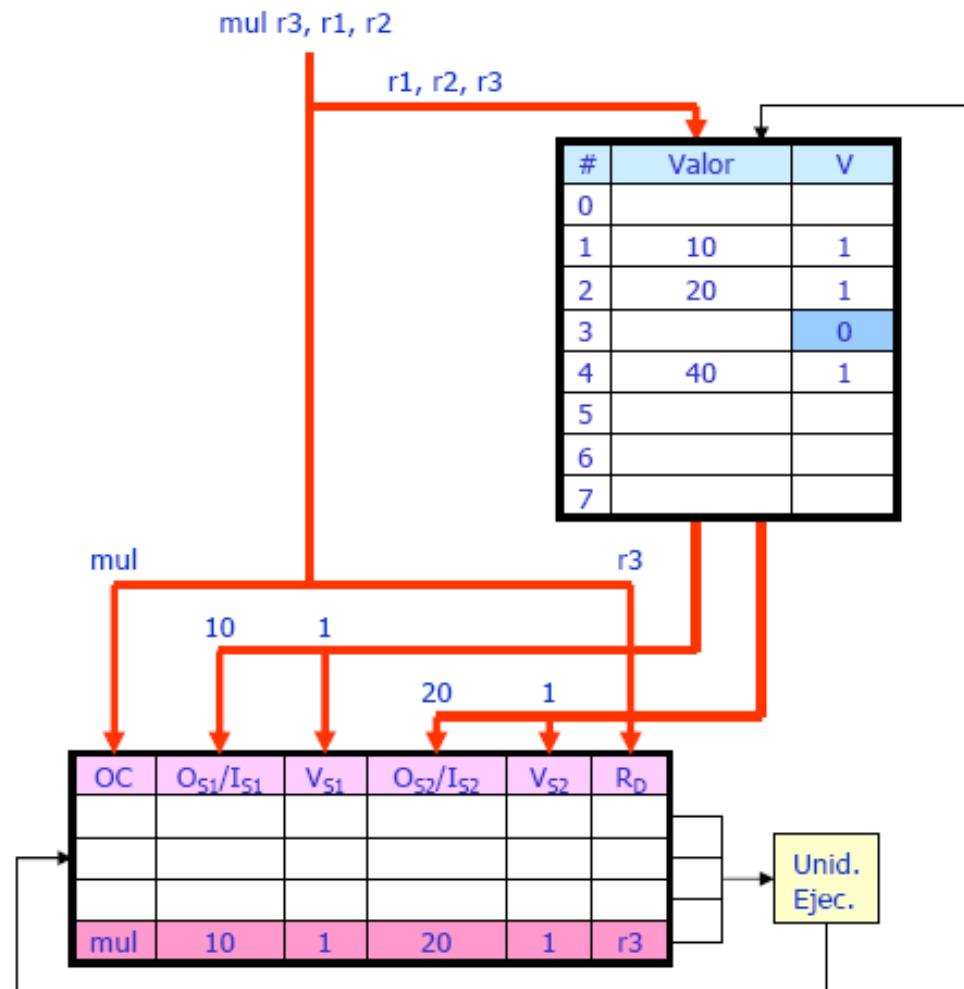
Emisión

- Estaciones de reserva. Ejemplo de uso (1)

Ciclo i:	mul r3, r1, r2
Ciclo i+1:	add r5, r2, r3
	add r6, r3, r4

Ciclo i:

- Se emite la instrucción de multiplicación, ya decodificada, a la estación de reserva
- Se anula el valor de r3 en el banco de registros
- Se copian los valores de r1 y r2 (disponibles) en la estación de reserva



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

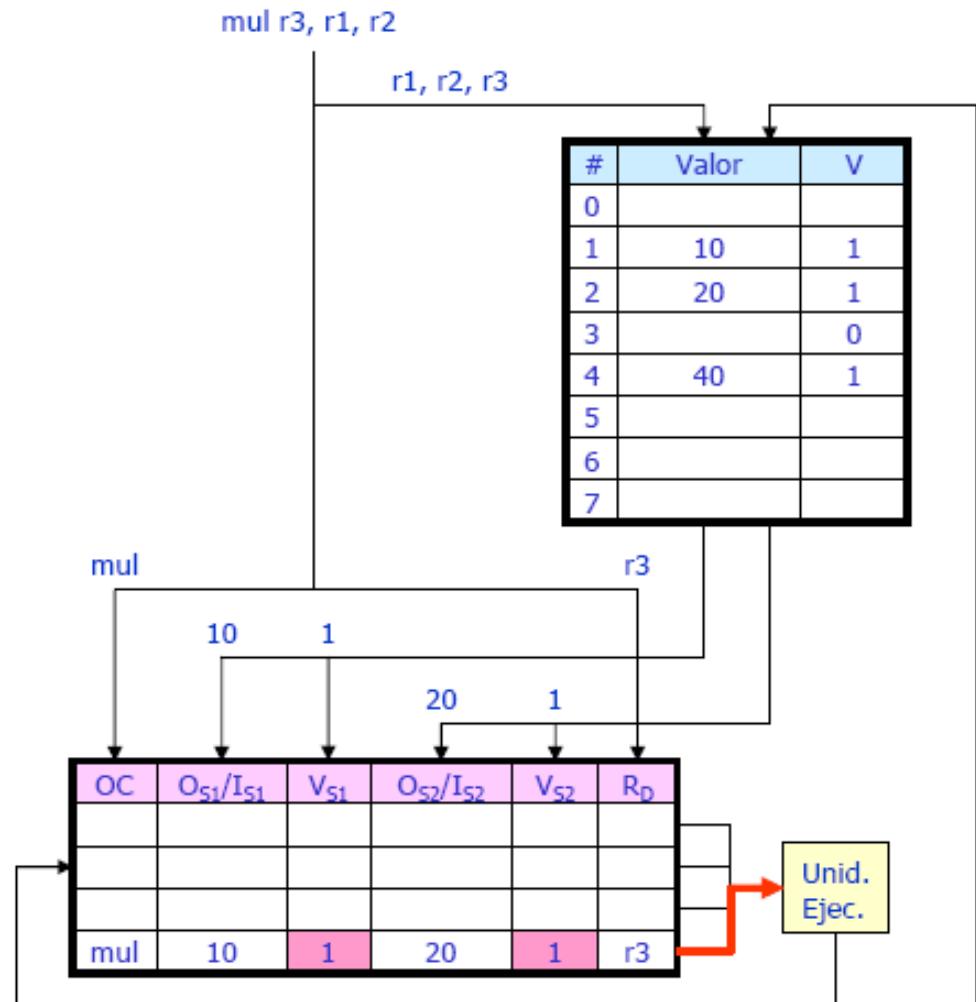
Emisión

- Estaciones de reserva. Ejemplo de uso (2)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
add r6, r3, r4

Ciclo i + 1:

- La operación de multiplicación tiene sus operadores preparados ($V_{S1} = 1$ y $V_{S2} = 1$)
- Así que puede enviarse a la unidad de ejecución



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

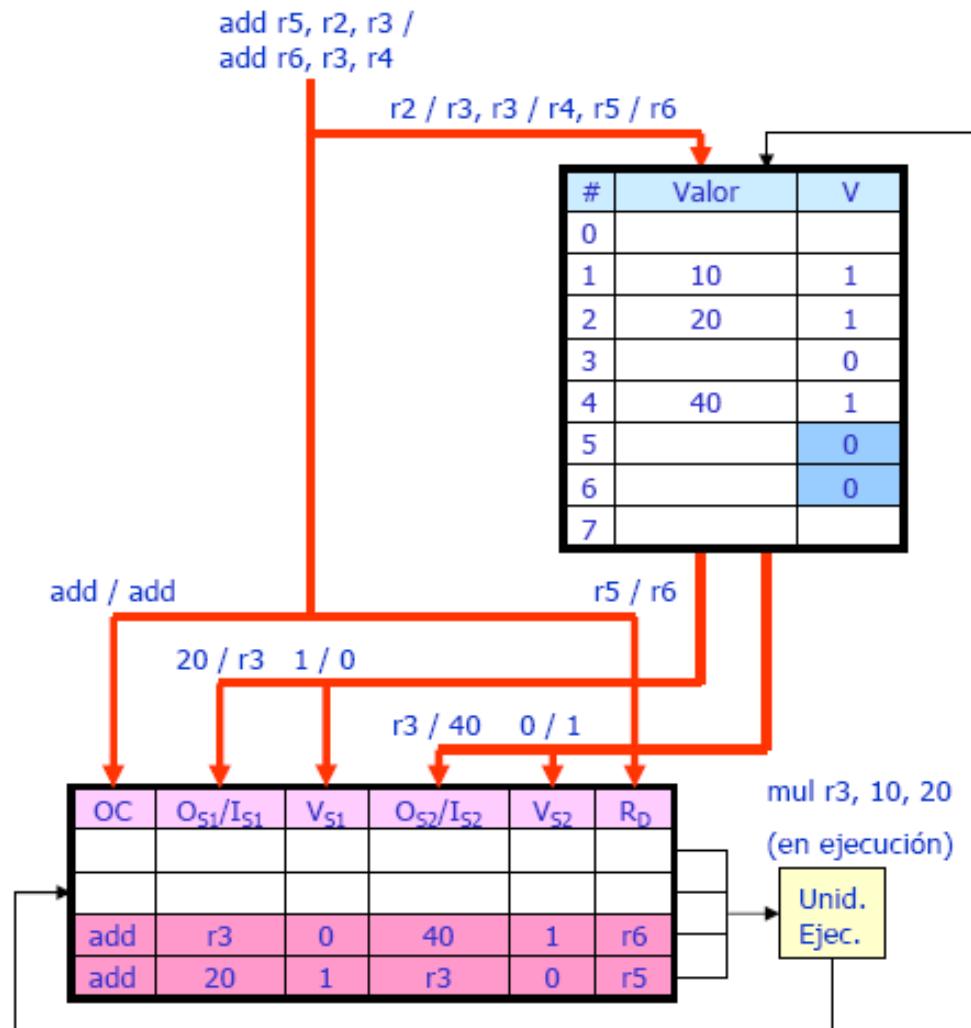
Emisión

- Estaciones de reserva. Ejemplo de uso (3)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclo i + 1 (*cont.*):

- Se emiten las dos instrucciones de suma a la estación de reserva
- Se anulan los valores de r5 y r6 en el banco de registros
- Se copian los valores de los operandos disponibles y los identificadores de los operandos no preparados



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

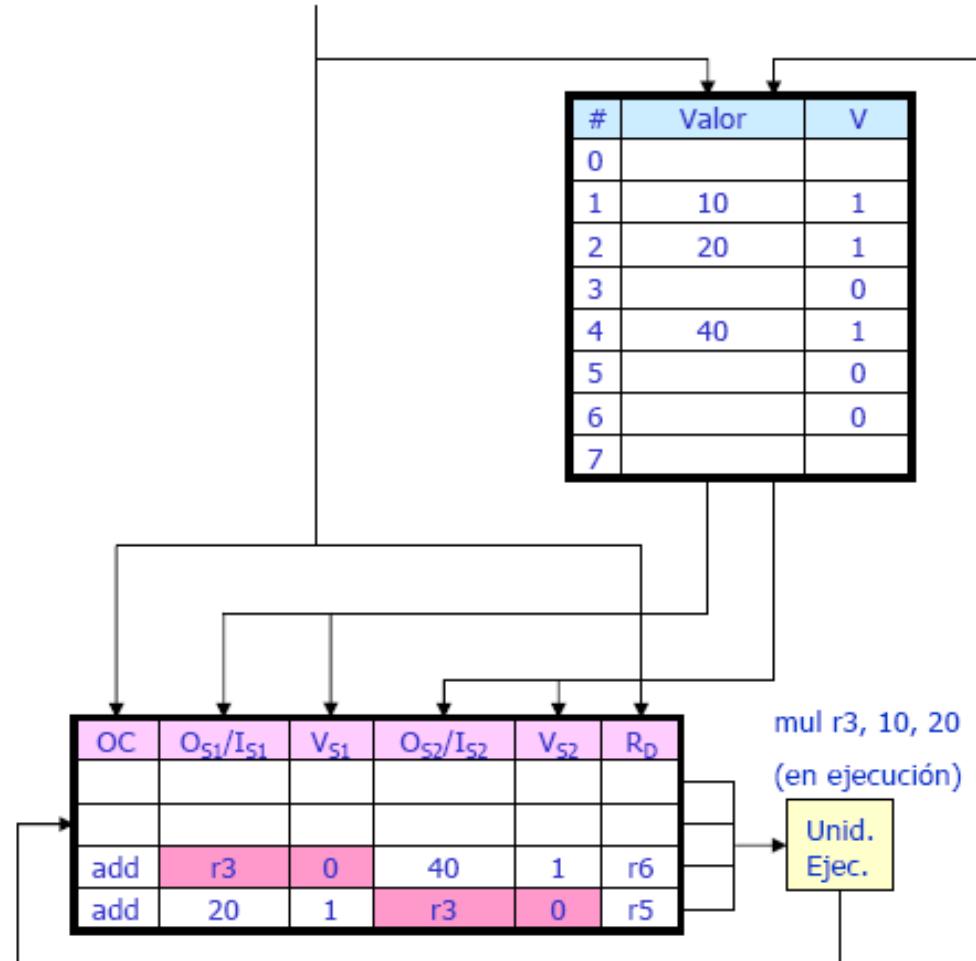
Emisión

- Estaciones de reserva. Ejemplo de uso (4)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclos i + 2 .. i + 5:

- La multiplicación sigue ejecutándose
- No se puede ejecutar ninguna suma hasta que esté disponible el resultado de la multiplicación (r3)



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

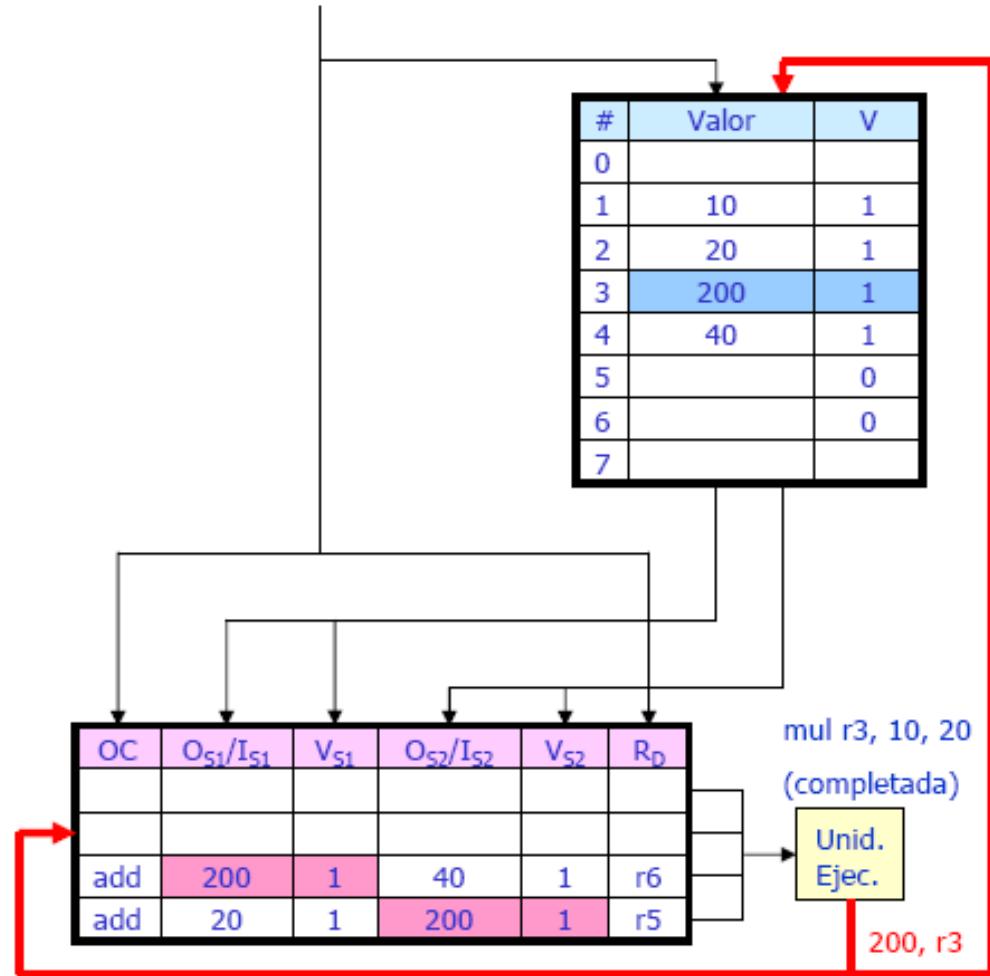
Emisión

- Estaciones de reserva. Ejemplo de uso (5)

Ciclo i: mul r3, r1, r2
 Ciclo i+1: add r5, r2, r3
 add r6, r3, r4

Ciclo i + 6:

- Se escribe el resultado de la multiplicación en el banco de registros y en las entradas de la estación de reserva
- Se actualizan los bits de disponibilidad de r3 en el banco de registros y en la estación de reserva



Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

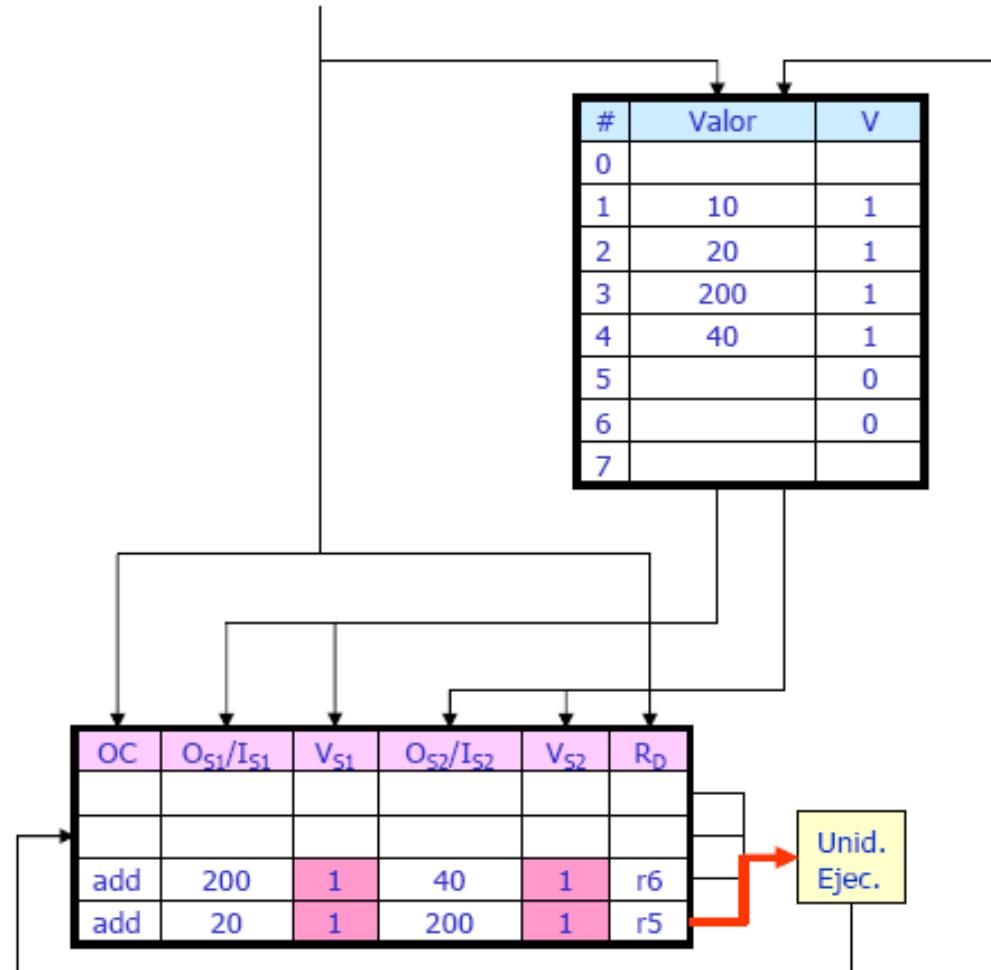
Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (6)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
add r6, r3, r4



Ciclo i + 6 (*cont.*):

- Las sumas tienen sus operadores preparados ($VS_1 = 1$ y $VS_2 = 1$)
- Así que pueden enviarse a la unidad de ejecución

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

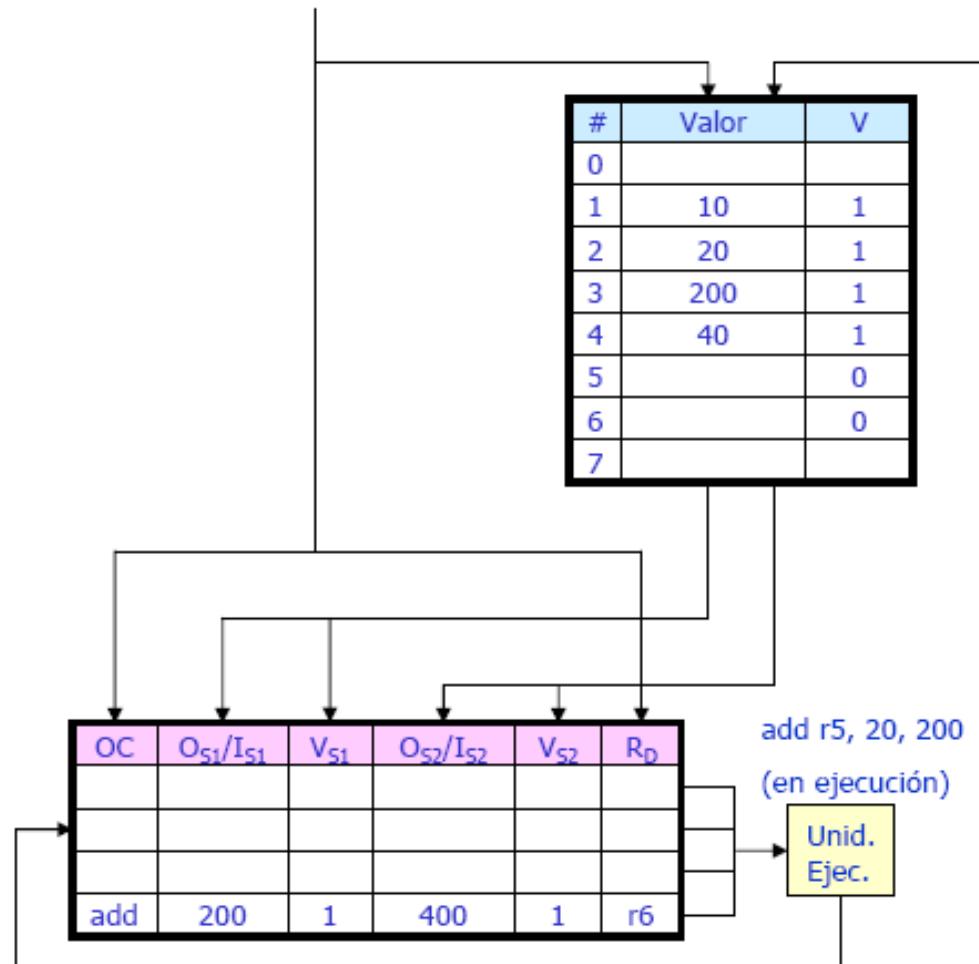
Cauce

Decodificación

Emisión

- Estaciones de reserva. Ejemplo de uso (7)

Ciclo i: mul r3, r1, r2
Ciclo i+1: add r5, r2, r3
add r6, r3, r4



Ciclo i + 6 (*cont.*):

- Como sólo hay una unidad de ejecución, se envía la instrucción más antigua de la estación de reserva, la primera suma

Unidad 2. Superescalares

2.2. Cauce superescalar

Motivación

Cauce

Decodificación

Emisión

- Ejercicio
 - Emisión ordenada/desordenada
 - Recursos: 1 lw/sw, 1 add/sub, 1 mult/div
 - Recursos: 1 lw/sw, 2 add/sub, 2 mult/div
 - Latencia: lw/sw (5 ciclos), add/sub (2 ciclos), mult/div (5 ciclos)

Lw r1,0(r2)

Add r2,r1,r3

Mult r3,r1,r2

Sub r4,r1,r2

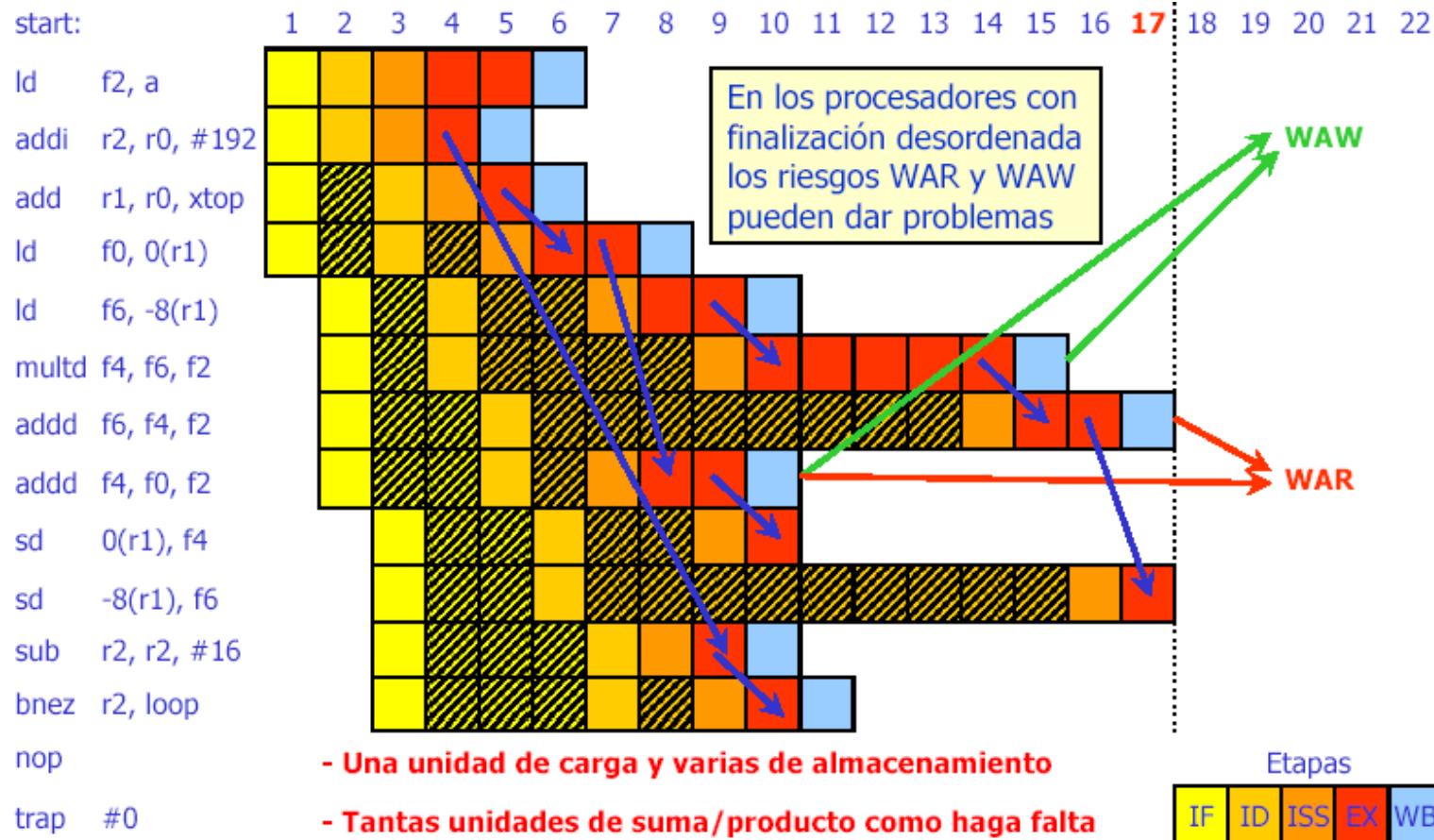
Add r4,r1,r2

Div r4,r5,r6

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Renombrado de registros (motivación)



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Técnica para **evitar el efecto de las dependencias WAR, o Antidependencias** (en la emisión desordenada) y **WAW, o Dependencias de Salida** (en la ejecución desordenada).

$$\begin{aligned} R3 &:= R3 - R5 \\ R4 &:= R3 + 1 \\ R3 &:= R5 + 1 \\ R7 &:= R3 * R4 \end{aligned}$$

Cada escritura se asigna
a un registro físico distinto

$$\begin{aligned} R3b &:= R3a - R5a \\ R4a &:= R3b + 1 \\ R3c &:= R5a + 1 \\ R7a &:= R3c * R4a \end{aligned}$$

Sólo RAW

R.M. Tomasulo (67)

Implementación Estática: Durante la Compilación

Implementación Dinámica: Durante la Ejecución (circuitería adicional y registros extra)

Características de los Buffers de Renombrado

- **Tipos de Buffers** (separados o mezclados con los registros de la arquitectura)
- **Número de Buffers de Renombrado**
- **Mecanismos para acceder a los Buffers** (asociativos o Indexados)

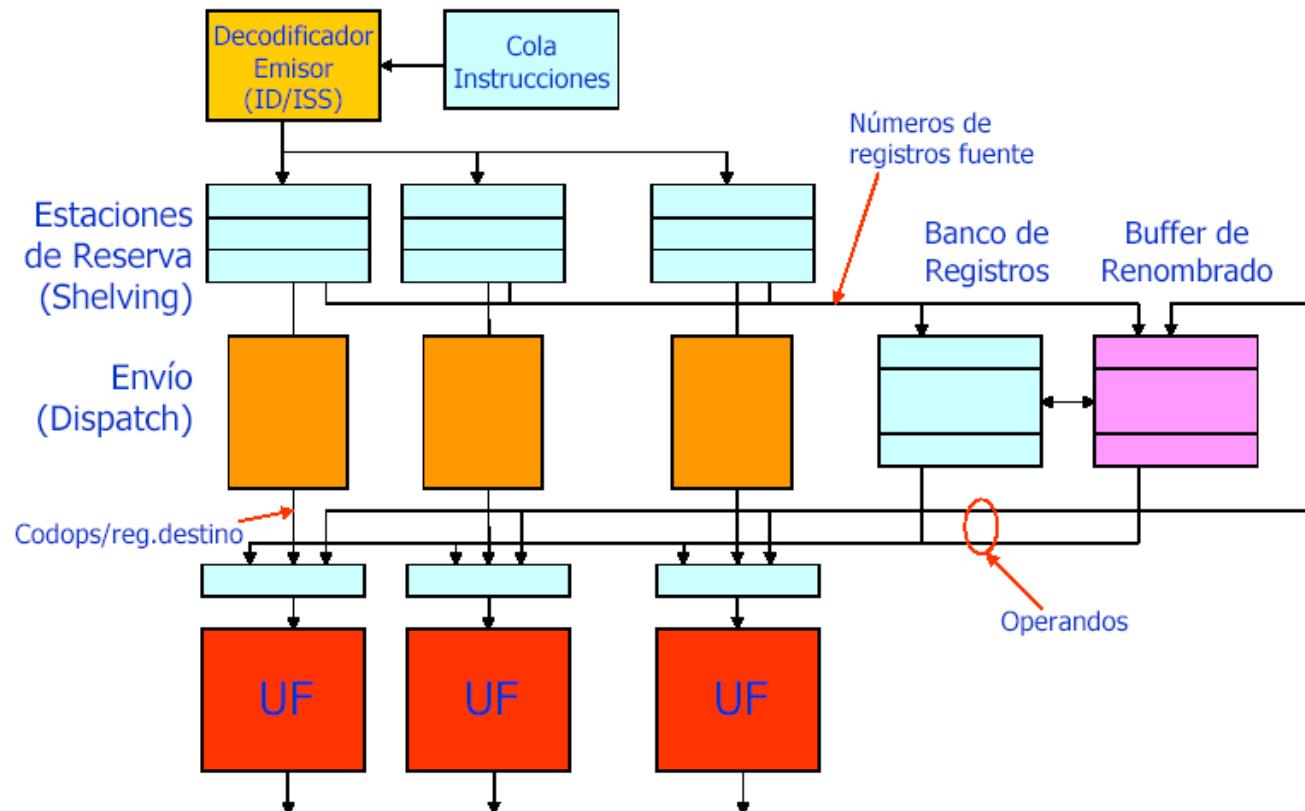
Velocidad del Renombrado

- **Máximo número de nombres asignados por ciclo** que admite el procesador

2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Renombrado de registros
 - Estaciones de reserva + buffer de renombrado



2.3. Gestión de riesgos (dependencia datos)

Renombrado

- Tipos de buffers de renombrado

Buffer de Renombrado con Acceso Asociativo

Búffer de Renombrado

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
1	5	50	1	1
1	12	1200	1	1
1	2	20	1	1
1	1	3	1	1
:	:	:	:	:

Buffer de Renombrado con Acceso Indexado

Índices

Búsq. de r2 →

Entrada Válida	Índice
0	2
1	5
1	3
3	12
:	:

Valor	Valor Válido
45	1
320	1
30	1
20	1
:	:

- Permite varias escrituras pendientes a un mismo registro
- Se utiliza el bit último para marcar cual ha sido la más reciente

- Sólo permite una escritura pendiente a un mismo registro
- Se mantiene la escritura más reciente

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (I)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	0			
5	0			
6	0			
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

$$\begin{aligned} r2a &= r0a * r1a \\ r3a &= r1a + r2a \\ r2b &= r0a - r1a \end{aligned}$$

Situación Inicial:

- Existen dos renombrados de r1 en las entradas 2 y 3 del buffer
- La última está en la entrada 3

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (II)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

 r0: 0, "válido" r1: 15, "válido" 4

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo i:

- Se emite la multiplicación
- Se accede a los operandos de la multiplicación, que tienen valores válidos en el buffer de renombrado
- Se renombra r2 (el destino de mul)

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (III)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	1
5	3		0	1
6				
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido" r2: "no válido" 5

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo $i + 1$:

- Se emite la suma
- Se accede a sus operandos, pero r2 no estará preparado hasta que termine la multiplicación
- Se renombra r3 (destino de add)

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (IV)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2		0	0
5	3		0	1
6	2		0	1
7	0			
8	0			
9	0			
10	0			
11	0			
12	0			
13	0			
14	0			

 r0: 0, "válido" r1: 15, "válido" 6

```
mul r2, r0, r1
add r3, r1, r2
sub r2, r0, r1
```

Ciclo i + 2:

- Se emite la resta
- Se accede a sus operandos
- Se vuelve a renombrar r2 (destino de sub)

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado (V)

Entrada Válida	Registro Destino	Valor	Valor Válido	Último
0	4	40	1	1
1	0	0	1	1
2	1	10	1	0
3	1	15	1	1
4	2	0	1	0
5	3		0	1
6	2		0	1
7				
8				
9				
10				
11				
12				
13				
14				

r1: 15, "válido" r2: 0, "válido"

```
mul r2, r0, r1  
add r3, r1, r2  
sub r2, r0, r1
```

Ciclo i + 5:

- Termina la multiplicación
- Se actualiza el resultado en el buffer de renombrado
- Ya se puede ejecutar la suma con el valor de r2 de la entrada 4
- Cuando termine la resta escribirá otro valor para r2 en la entrada 6
- Sólo se escribirá en el banco de registros el último valor de r2

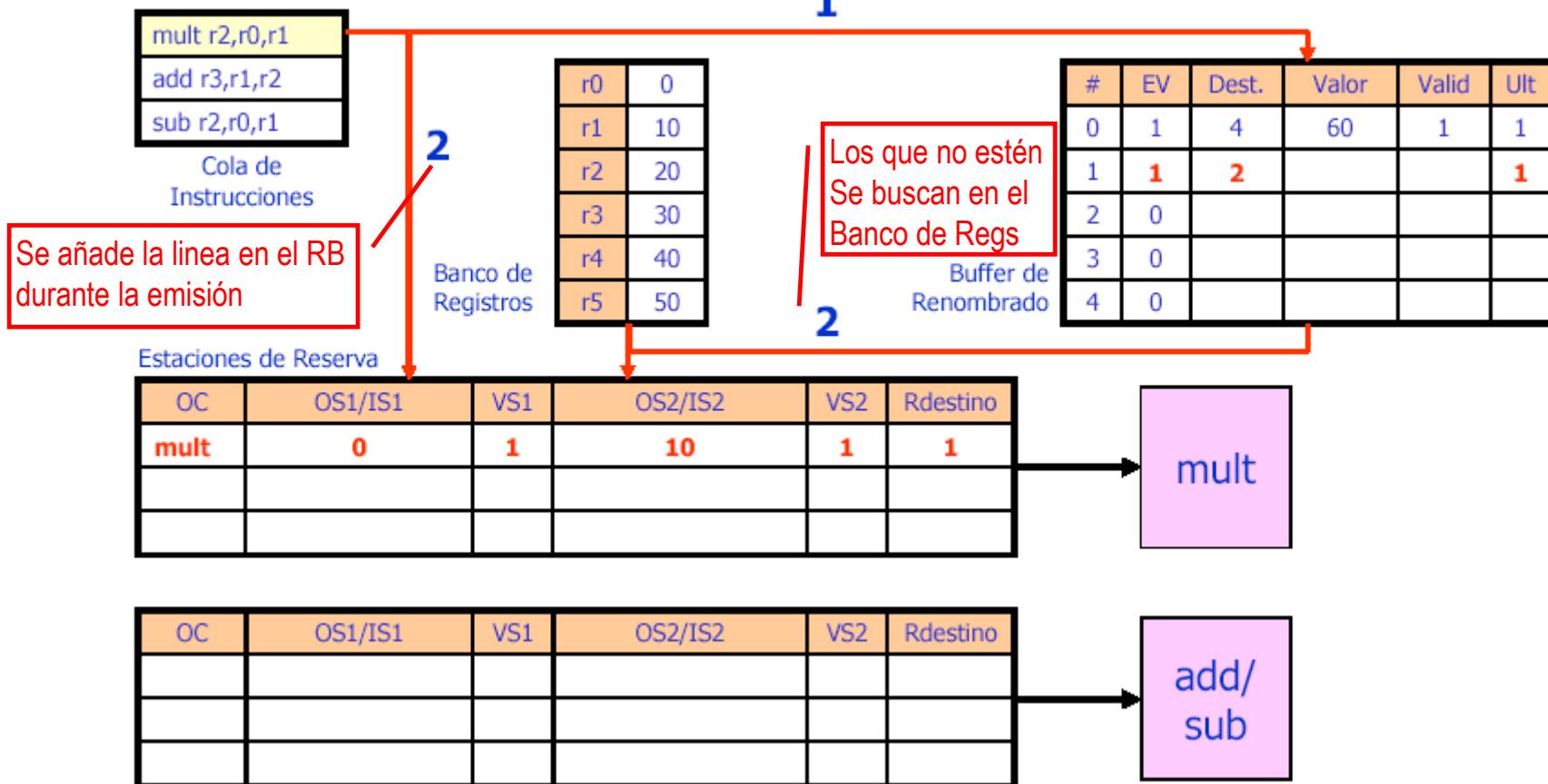
Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva

Emisión de la multiplicación

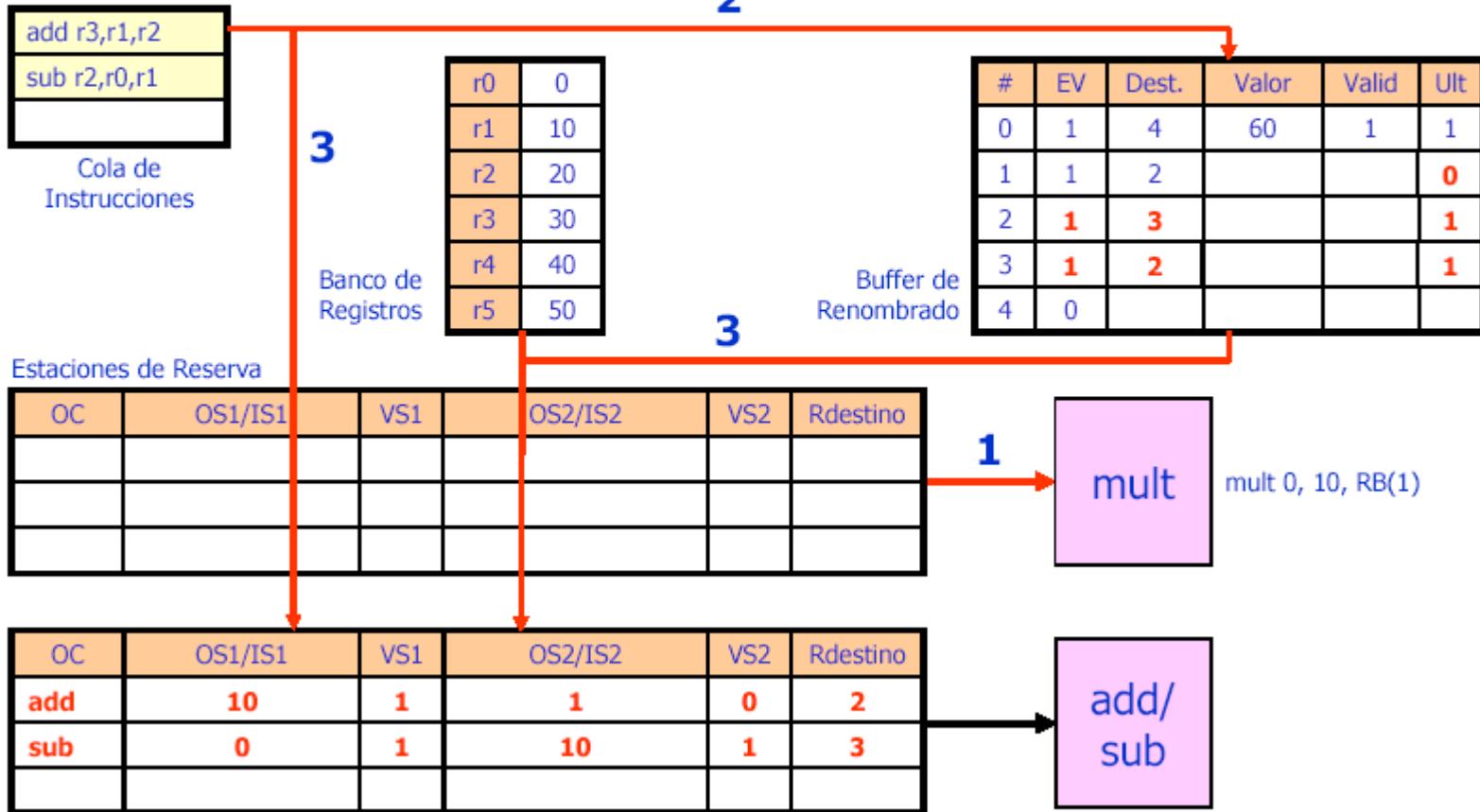


2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (II)

Envío de la multiplicación y emisión de las suma y la resta
2

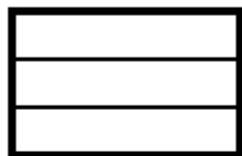


2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (III)

Envío de la resta



Cola de
Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

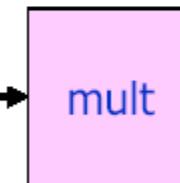
Banco de
Registros

Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2			1
4	0				

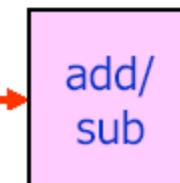
Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



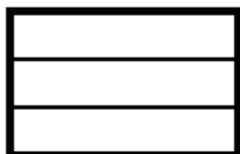
sub 0, 10, RB(3)

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (IV)

Termina la resta



Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

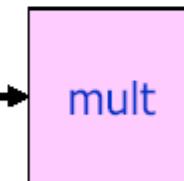
Banco de Registros

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2			0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

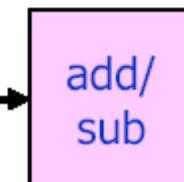
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	1	0	2



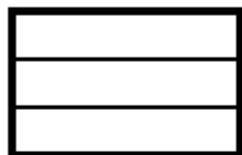
sub 0, 10, RB(3)

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (V)

Termina la multiplicación



Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

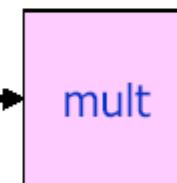
Banco de Registros

Buffer de Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva

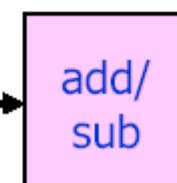
OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



mult 0, 10, RB(1)

1

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino
add	10	1	0	1	2

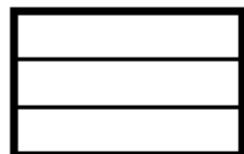


2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VI)

Envío de la suma



Cola de
Instrucciones

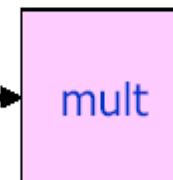
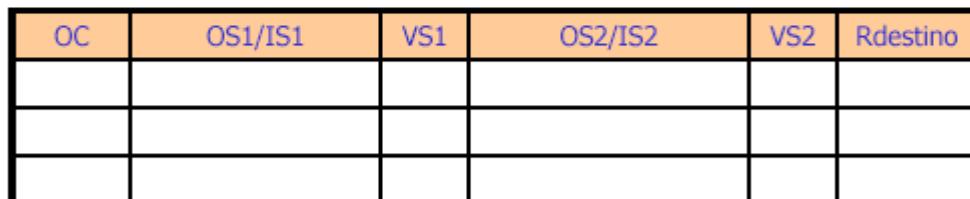
r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Banco de
Registros

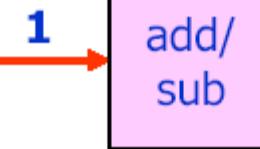
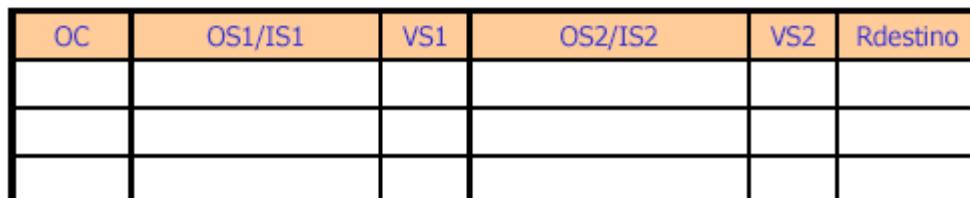
Buffer de
Renombrado

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3			1
3	1	2	-10	1	1
4	0				

Estaciones de Reserva



mult



add/
sub

add 10, 0, RB(2)

1

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Termina la suma

Cola de Instrucciones

r0	0
r1	10
r2	20
r3	30
r4	40
r5	50

Banco de Registros

#	EV	Dest.	Valor	Valid	Ult
0	1	4	60	1	1
1	1	2	0	1	0
2	1	3	10	1	1
3	1	2	-10	1	1
4	0				

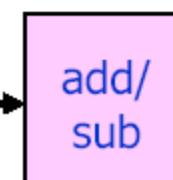
Buffer de Renombrado

Estaciones de Reserva

OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



OC	OS1/IS1	VS1	OS2/IS2	VS2	Rdestino



add 10, 0, RB(2)

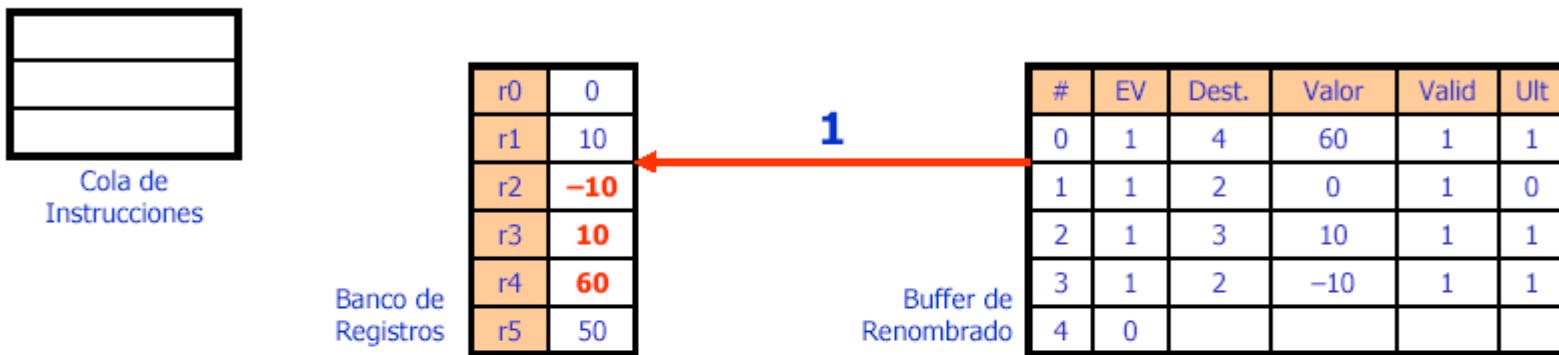
1

2.3. Gestión de riesgos (dependencia datos)

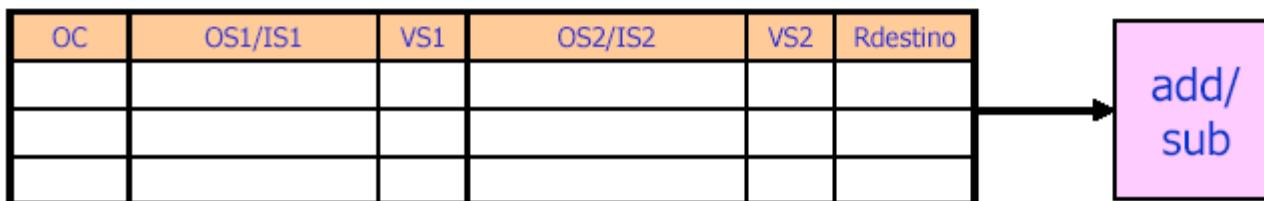
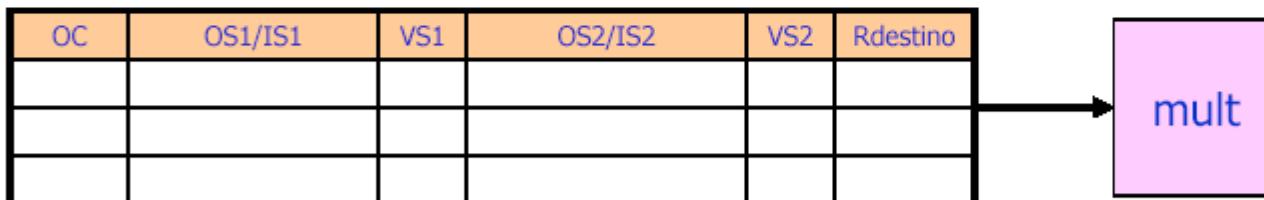
Renombrado

Ejemplo de Renombrado y Estaciones de Reserva (VII)

Se actualizan los registros



Estaciones de Reserva



Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

1			
2	instr(n)	f
3	instr(n+1)	x
4	instr(n+2)	f
5	instr(n+3)	x
6	instr(n+4)	x
7	instr(n+5)	i
8	instr(n+6)	i
9			
10			

Cola
(Las instrucciones se retiran desde aquí)

Cabecera
(Las instrucciones que se decodifican se introducen a partir de aquí)

- El puntero de cabecera apunta a la siguiente posición libre y el **puntero de cola** a la siguiente instrucción a retirar.
- Las instrucciones **se introducen en el ROB en orden de programa estricto** y pueden estar marcadas como **emitidas (issued, i)**, **en ejecución (x)**, o **finalizada su ejecución (f)**
- Las **instrucciones sólo se pueden retirar** (se produce la finalización con la escritura en los registros de la arquitectura) **si han finalizado**, y todas las que les preceden también.
- La **consistencia se mantiene porque sólo las instrucciones que se retiran del ROB se completan** (escriben en los registros de la arquitectura) y se retiran en el orden estricto de programa.

La gestión de interrupciones y la ejecución especulativa se realizan fácilmente mediante el ROB

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (I)

I1: mult r1, r2, r3

I2: st r1, 0x1ca

I3: add r1, r4, r3

I4: xor r1, r1, r3



Dependencias:

RAW: (I1,I2), (I3,I4)

WAR: (I2,I3), (I2,I4)

WAW: (I1,I3), (I1,I4), (I3,I4)

I1: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r2** y **r3**)

I2: Se envía a la unidad de almacenamiento hasta que esté disponible **r1**

I3: Se puede empezar a ejecutar inmediatamente (se suponen disponibles **r4** y **r3**)

I4: Se envía a la estación de reserva de la ALU para esperar a **r1**

Estación de Reserva (Unidad de Almacenamiento)

codop	dirección	op1	ok1
st	0x1ca	3	0

Estación de Reserva (ALU)

codop	dest	op1	ok1	op2	ok2
xor	6	5	0	[r3]	1

Líneas del ROB

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (II)

Ciclo 7

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	-	0	x	9
6	xor	10	r1	int_alu	-	0	i	-

Ciclo 9 No se puede retirar **add** aunque haya finalizado su ejecución

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	-	0	x	10

Ciclo 10 Termina **xor**, pero todavía no se puede retirar

#	codop	Nº Inst.	Reg.Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	-	0	x	12
4	st	8	-	store	-	0	i	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Unidad 2. Superescalares

2.3. Gestión de riesgos (dependencia datos)

Renombrado

Reorden

Ejemplo de Uso del Buffer de Reorden (III)

Ciclo 12

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
3	mult	7	r1	int_mult	33	1	f	12
4	st	8	-	store	-	1	f	12
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

Ciclo 13

#	codop	Nº Inst.	Reg. Dest.	Unidad	Resultado	ok	marca	'ready'
5	add	9	r1	int_add	17	1	f	9
6	xor	10	r1	int_alu	21	1	f	10

- Se ha supuesto que se pueden retirar dos instrucciones por ciclo.
- Tras finalizar las instrucciones **mult** y **st** en el ciclo 12, se retirarán en el ciclo 13.
- Después, en el ciclo 14 se retirarán las instrucciones **add** y **xor**.

Unidad 2. Superescalares

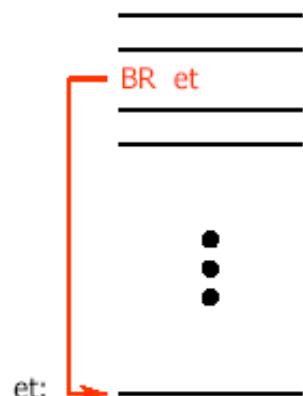
2.3. Gestión de riesgos (control)

Riesgos

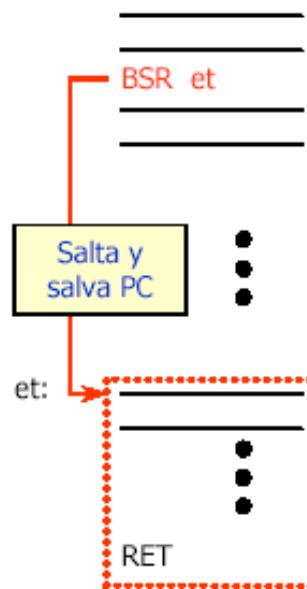
Clasificación de los Saltos

Saltos Incondicionales

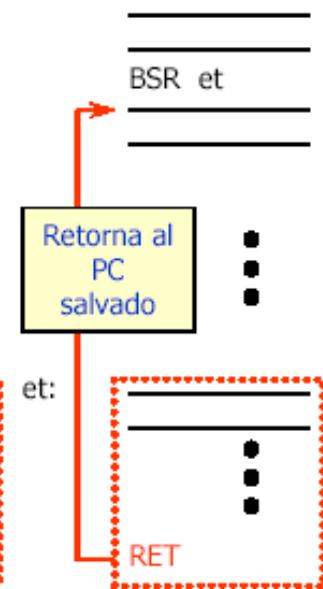
Salto Incondicional



Llamada a Subrutina

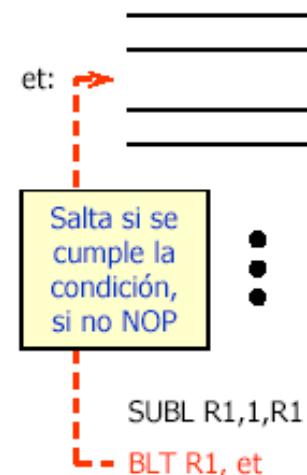


Retorno de Subrutina

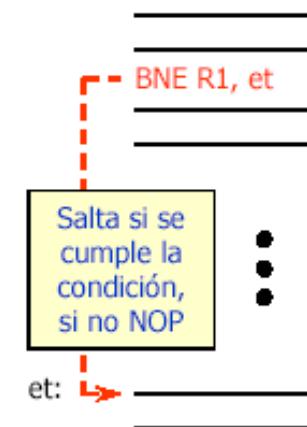


Saltos Condicionales

Condición de Bucle



Otro Salto Condicional



Salto hacia atrás
o hacia adelante

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

El efecto de los saltos en los procesadores superescalares es más pernicioso ya que, al emitirse varias instrucciones por ciclo, prácticamente en cada ciclo puede haber una instrucción de salto.

El salto retardado no tiene mucho interés porque la unidad de emisión decide las instrucciones que pasan a ejecutarse teniendo en cuenta las dependencias.

- **Detección de la Instrucción de Salto**

Cuanto antes se detecte que una instrucción es de salto menor será la posible penalización. Los saltos se detectan usualmente en la fase de decodificación.

- **Gestión de los Saltos Condicionales no Resueltos**

Si en el momento en que la instrucción de salto evalúa la condición de salto ésta no se haya disponible se dice que *el salto o la condición no se ha resuelto*. Para resolver este problema se suele utilizar el **procesamiento especulativo del salto**.

- **Acceso a las Instrucciones destino del Salto**

Hay que determinar la forma de acceder a la secuencia a la que se produce el salto

- **La implementación física de las Unidades de Salto**

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

- Detección

- En etapa de decodificación



- Detección temprana ('early branch detection')
 - Detección paralela a decodificación



- Detección anticipada ('look-ahead branch detection')



- Detección integrada en captación



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos	Gestión	
Uso de los ciclos que siguen a la inst. de salto condicional	Salto Retardado	Se utilizan los ciclos que siguen a la captación de una instrucción de salto para insertar instrucciones que deben ejecutarse independientemente del resultado del salto (Primeras arquitecturas RISC y posteriores)
Gestión de Saltos Condicionales no Resueltos (Una condición de salto no se puede comprobar si no se ha terminado de evaluar)	Bloqueo del Procesamiento del Salto	Se bloquea la instrucción de salto hasta que la condición esté disponible (68020, 68030, 80386)
	Procesamiento Especulativo de los Saltos	La ejecución prosigue por el camino más probable (se especula sobre las instrucciones que se ejecutarán). Si se ha errado en la predicción hay que recuperar el camino correcto. (Típica en los procesadores superescalares actuales)
	Múltiples Caminos	Se ejecutan los dos caminos posibles después de un salto hasta que la condición de salto se evalúa. En ese momento se cancela el camino incorrecto. (Máquinas VLIW experimentales: Trace/500 , URPR-2)
Evitar saltos condicionales	Ejecución Vigilada (<i>Guarded Exec.</i>)	Se evitan los saltos condicionales incluyendo en la arquitectura instrucciones con operaciones condicionales (IBM VLIW, Cydra-5, Pentium, HP PA, Dec Alpha)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Esquemas de Predicción de Salto

Predicción Fija

Se toma siempre la misma decisión: el salto siempre se realiza, '*taken*', o no, '*not taken*'

Predicción Verdadera

La decisión de si se realiza o no se realiza el salto se toma mediante:

- **Predicción Estática:**

Según los atributos de la instrucción de salto (el código de operación, el desplazamiento, la decisión del compilador)

- **Predicción Dinámica:**

Según el resultado de ejecuciones pasadas de la instrucción (historia de la instrucción de salto)



Prestaciones

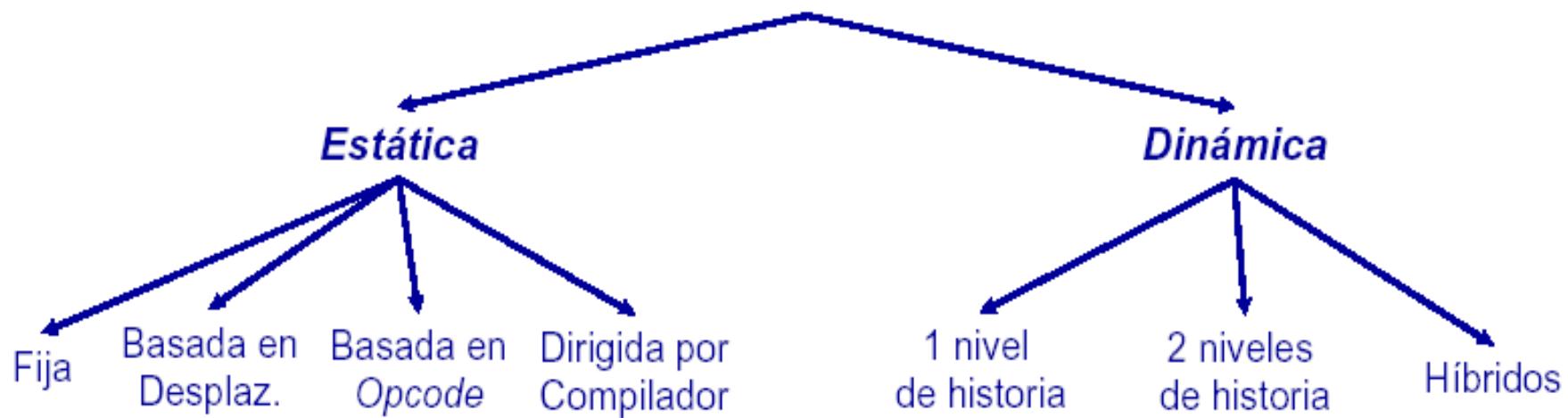
Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Predicción de saltos



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción fija

Aproximación 'Siempre No Tomado'

- Toda condición de salto no resuelta se predice que no da lugar a un salto
- Se continúa la ejecución por donde iba aunque se puede adelantar algo el procesamiento de la secuencia de salto (calculo de la dirección de salto, BTA)
- Cuando se evalúa la condición se comprueba si la predicción era buena.
- Si la predicción era buena el procesamiento continúa y se borra la BTA, y si era mala se abandona el procesamiento de la secuencia predicha (no se considera su efecto) y se captan instrucciones a partir de la BTA

* Es más fácil de implementar que la aproximación de 'Siempre Tomado'

SuperSparc (1992)
(TP: 1; NTP: 0)

Alpha21064
Power I (1990)
(TP: 3; NTP: 0)

Power 2 (1993)
(TP: 1; NTP: 0)

Aproximación 'Siempre Tomado'

- Toda condición de salto no resuelta se predice que da lugar a un salto
- En previsión de error de predicción se salva el estado de procesamiento actual (PC) y se empieza la ejecución a partir de la dirección de salto.
- Cuando se evalúa la condición de salto se comprueba si la predicción era buena
- Si la predicción es correcta se continúa, y si es errónea se recupera el estado almacenado y no se considera el procesamiento de la secuencia errónea

MC68040 (1999)
(TP: 1; NTP: 2)

* Necesita una implementación más compleja que la aproximación anterior aunque suele proporcionar mejores prestaciones

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática

<p>Predicción basada en el Código de Operación</p> <p>Para ciertos códigos de operación (ciertos saltos condicionales específicos) se predice que el salto se toma, y para otros que el salto no se toma</p>	MC88110 (93) PowerPC 603(93)
---	---------------------------------

Ejemplo: Predicción Estática en el MC88110

Formato	Instrucción		Predicción
	Condición Especificada	Bit 21 de la Instr.	
bcnd <i>(Branch Conditional)</i>	$\neq 0$	1	Tomado
	$= 0$	0	No Tomado
	> 0	1	Tomado
	< 0	0	No Tomado
	≥ 0	1	Tomado
	≤ 0	0	No Tomado
	bb1 <i>(Branch on Bit Set)</i>		Tomado
	bb0 <i>(Branch on Bit Clear)</i>		No Tomado

Unidad 2. Superescalares

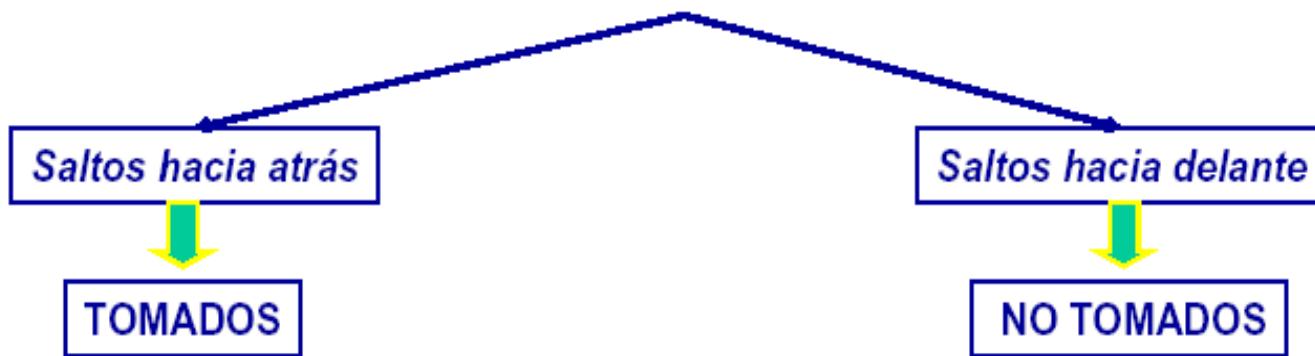
2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática

Predicción basada en la DIRECCIÓN del salto



La mayoría de saltos hacia atrás corresponden a bucles

La mayoría de saltos hacia delante corresponden a IF-THEN-ELSE

Mal comportamiento en programas con pocos bucles y muchos IF-THEN-ELSE

EJEMPLOS

- Alpha 21064 (1992) (Opción seleccionable)
- PowerPC 601/603 (1993)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción estática



- Se añade un **Bit de Predicción** al opcode de la instrucción
- El compilador activa o desactiva este bit para indicar su predicción

EJEMPLOS

- AT&T 9210 Hobbit (1993)
- PowerPC 601/603 (1993)
- PA 8000 (1996)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica

- La predicción para cada instrucción de salto puede cambiar cada vez que se va a ejecutar ésta según la historia previa de saltos tomados/no-tomados para dicha instrucción.
- El presupuesto básico de la predicción dinámica es que es más probable que el resultado de una instrucción de salto sea similar al que se tuvo en la última (o en las n últimas ejecuciones)
- Presenta mejores prestaciones de predicción, aunque su implementación es más costosa

- **Predicción Dinámica Implícita**

No hay bits de historia propiamente dichos sino que se almacena la dirección de la instrucción que se ejecutó después de la instrucción de salto en cuestión

- **Predicción Dinámica Explícita**

Para cada instrucción de salto existen unos bits específicos que codifican la información de historia de dicha instrucción de salto

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

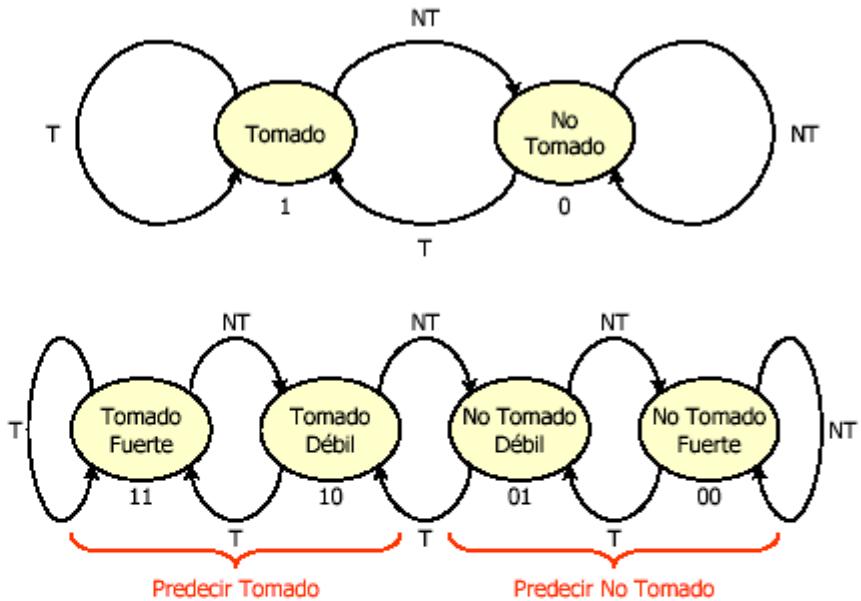
Riesgos

Gestión

- Predicción dinámica explícita

Predicción con 1 bit de historia

La designación del estado, Tomado (1) o No Tomado (0), indica lo que se predice, y las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)



Predicción con 2 bits de historia

Existen cuatro posibles estados. Dos para predecir Tomado y otros dos para No Tomado

La primera vez que se ejecuta un salto se inicializa el estado con predicción estática, por ejemplo 11

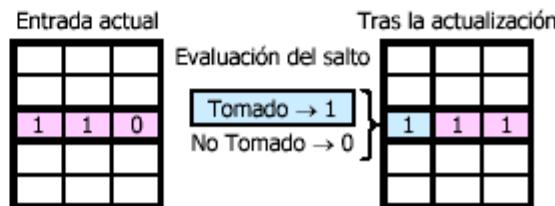
Las flechas indican las transiciones de estado según lo que se produce al ejecutarse la instrucción (T o NT)

Predicción con 3 bits de historia

Cada entrada guarda las últimas ejecuciones del salto

Se predice según el bit mayoritario (por ejemplo, si hay mayoría de unos en una entrada se predice salto)

La actualización se realiza en modo FIFO, los bits se desplazan, introduciéndose un 0 o un 1 según el resultado final de la instrucción de salto



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

Implementación de los Bits de Historia

• En la Cache de Instrucciones	Alpha 21064 (92) (2k x 1 bit) Alpha 21064A (94) (4k x 2 bits) UltraSparc (92) (2k x 2 bits)
• En una Tabla de Historia de Salto (BHT)	PA8000(96) (253 x 3bits) PowerPC 620(95) (2K x 2bits) R10000 (96) (512 x 2bits)
• En una Cache para las Instrucciones a las que se produce el Salto (BTIC) o, • En una Cache para las Direcciones a las que se produce el Salto (BTAC)	Pentium (94) (256 x 2 bits) (BTAC) MC 68069 (93) (256 x 2bits) (BTAC)

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

1) Branch Target Buffer (BTB): bits acoplados

La BTB almacena

- La dirección destino de los últimos saltos tomados
- Los bits de predicción de ese salto

Actualización de la BTB

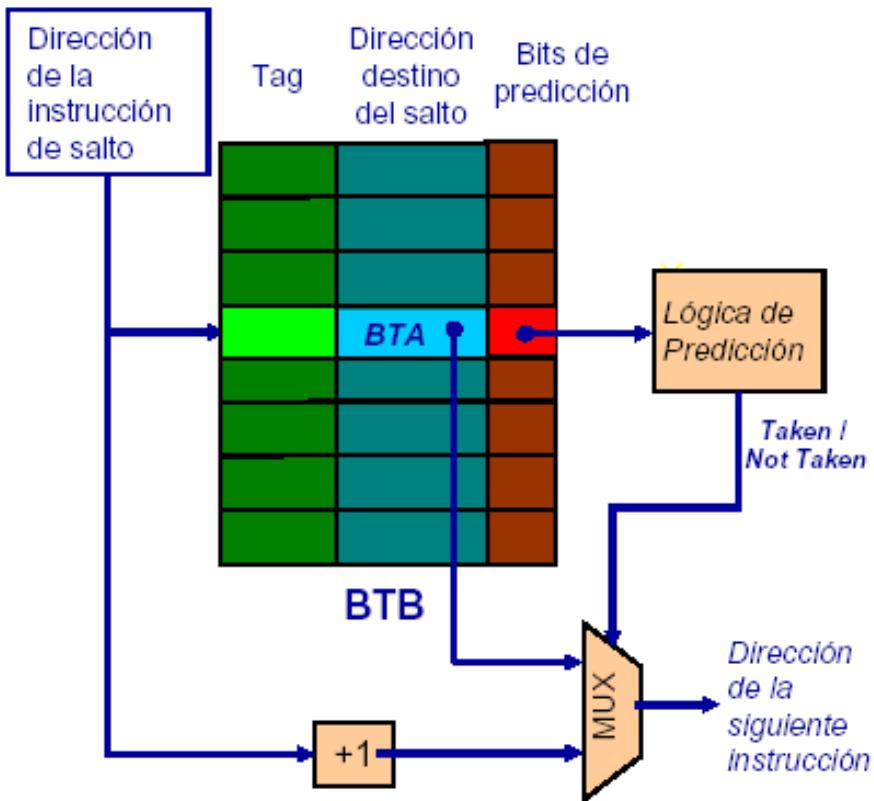
Los campos de la BTB se actualizan después de ejecutar el salto, cuando se conoce:

- Si el salto fue tomado o no \Rightarrow Actualizar bits de predicción
- La dirección destino del salto \Rightarrow Actualizar BTA

Predicción Implícita (sin bits de predicción)

Aplicable con un sólo bit de predicción

- Si la instrucción de salto está en la BTB
 \Rightarrow El salto se predice como tomado
- Si la instrucción de salto no está en la BTB
 \Rightarrow El salto se predice como no tomado



DESVENTAJA: Sólo se pueden predecir aquellas instrucciones de salto que están en la BTB

Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

2) Tabla de historia de saltos (BHT): bits desacoplados

Existen dos tablas distintas:

- La BTAC, que almacena la dirección destino de los últimos saltos tomados
- La BHT, que almacena los bits de predicción de todas las instrucciones de salto condicional

Ventaja

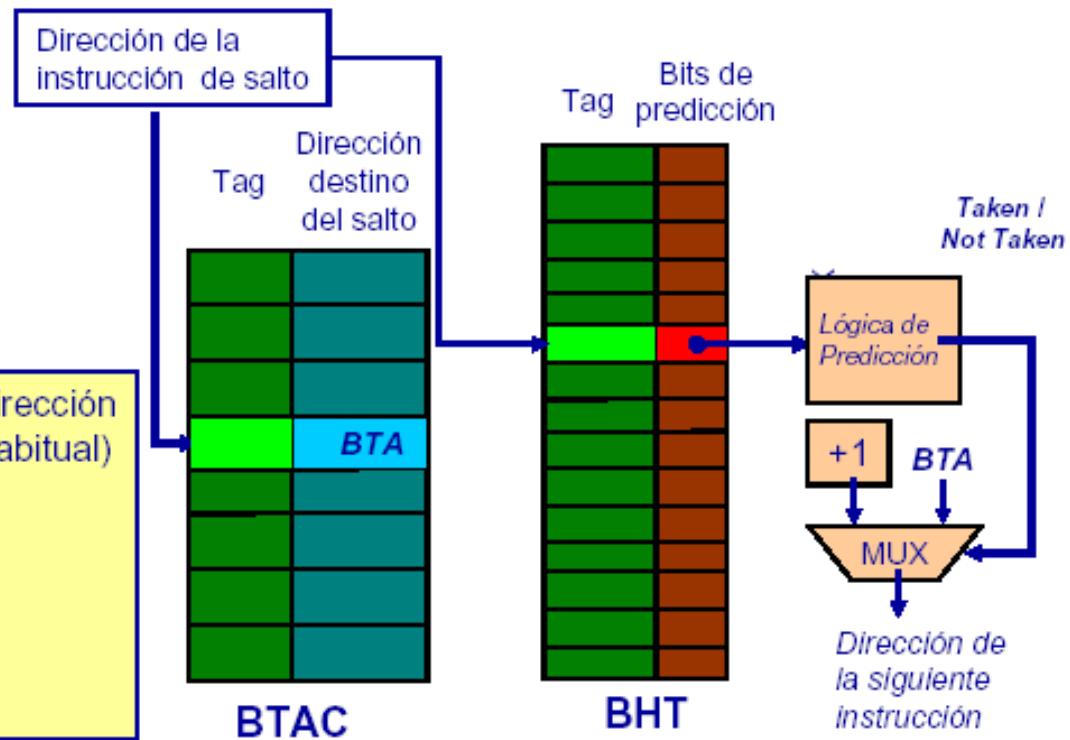
Puede predecir instruc. que no están en la BTAC (más entradas en BHT que en BTAC)

Desventaja

Aumenta el hardware necesario
⇒ 2 tablas asociativas

Acceso a la BHT

- Usando los bits menos significativos de la dirección
 - Sin TAGs ⇒ Menor coste (opción + habitual)
 - Compartición de entradas
 - ⇒ Se degrada el rendimiento
- Asociativa por conjuntos
 - Mayor coste ⇒ Tablas pequeñas
 - Para un mismo coste hardware
 - ⇒ Peor comportamiento



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Predicción dinámica explícita

3) Bits de predicción en la I-cache

Funcionamiento

Cuando se capta la instrucción de la cache
Si se trata de una instrucción de salto condicional

- Se accede en paralelo a los bits de predicción
- Si el salto se predice como tomado se accede a la instrucción destino del salto

Acceso a la instrucción destino del salto

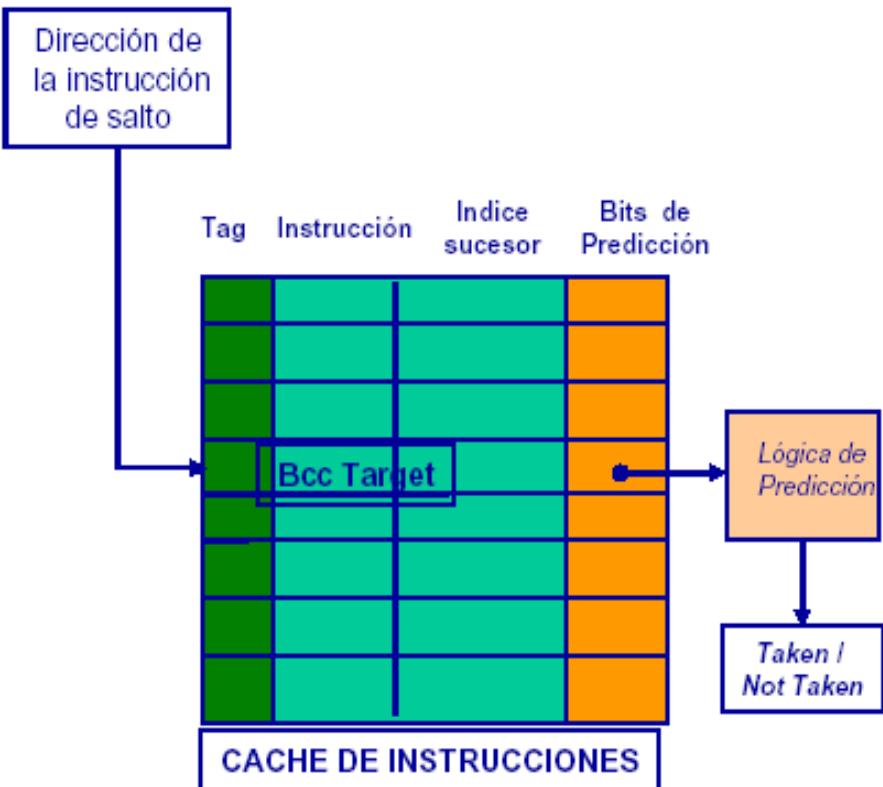
- BTB independiente
- Añadir índice sucesor a la I-cache

Alternativas de diseño

- Bits de predicción por cada instrucción de la cache
- Bits de predicción por cada línea de cache

Ventajas

- Puede predecir instrucciones que no están en la BTB
- No añade una cantidad extra de hardware excesiva



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

Extensión del Procesamiento Especulativo

- Tras la predicción, el procesador continúa ejecutando instrucciones especulativamente hasta que se resuelve la condición.
- El intervalo de tiempo entre el comienzo de la ejecución especulativa y la resolución de la condición puede variar considerablemente y ser bastante largo.
- En los procesadores superescalares, que pueden emitir varias instrucciones por ciclo, pueden aparecer más instrucciones de salto condicional no resueltas durante la ejecución especulativa.
- Si el número de instrucciones que se ejecutan especulativamente es muy elevado y la predicción es incorrecta, la penalización es mayor.

Así, cuanto mejor es el esquema de predicción mayor puede ser el número de instrucciones ejecutadas especulativamente.



- **Nivel de Especulación:** Número de Instrucciones de Salto Condicional sucesivas que pueden ejecutarse especulativamente (si se permiten varias, hay que guardar varios estados de ejecución). Ejemplos: Alpha 21064, PowerPC 603 (1); Power 2 (2); PowerPC 620 (4); Alpha 21164 (6)
- **Grado de Especulación:** Hasta qué etapa se ejecutan las instrucciones que siguen en un camino especulativo después de un salto. Ejemplos: Power 1 (Captación); PowerPC 601 (Captación, Decodificación, Envío); PowerPC 603 (Todas menos la finalización)

Unidad 2. Superescalares

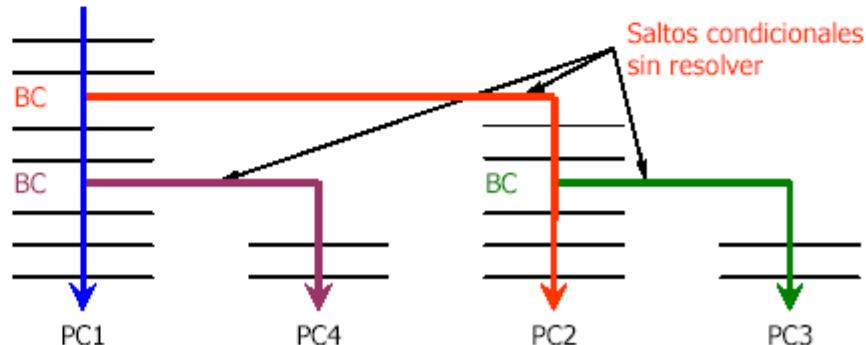
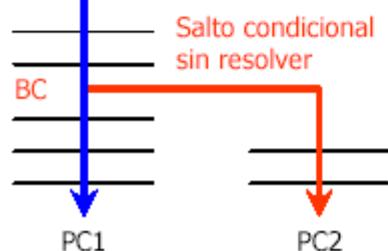
2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Ramificación multicamino

- Se siguen las dos secuencias de instrucciones que aparecen a partir de una instrucción de salto (la correspondiente al salto efectuado y al salto no efectuado). Una vez resuelto el salto la secuencia incorrecta se abandona
- Para implementar esta técnica hacen falta varios contadores de programa
- Se necesitan gran cantidad de recursos hardware y el proceso para descartar las instrucciones que se han ejecutado incorrectamente es complejo



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

Riesgos

Gestión

- Ejecución condicional

Instrucciones de Ejecución Condicional (*Guarded Execution*)

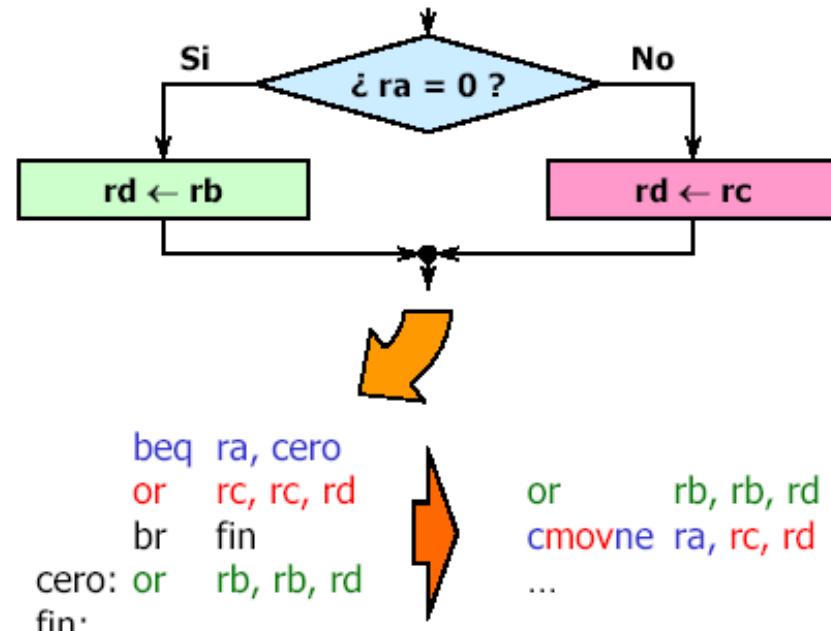
- Se pretende reducir el número de instrucciones de salto incluyendo en el repertorio máquina instrucciones con operaciones condicionales ('conditional operate instructions' o 'guarded instructions')
- Estas instrucciones tienen dos partes: la **condición** (denominada *guardia*) y la parte de **operación**

Ejemplo: cmovxx de Alpha

cmovxx ra.rq, rb.rq, rc.wq

- xx** es una condición
- ra.rq, rb.rq** enteros de 64 bits en registros ra y rb
- rc.wq** entero de 64 bits en rc para escritura
- El registro ra se comprueba en relación a la condición xx y si se verifica la condición rb se copia en rc.

Sparc V9, HP PA, y Pentium ofrecen también estas instrucciones.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

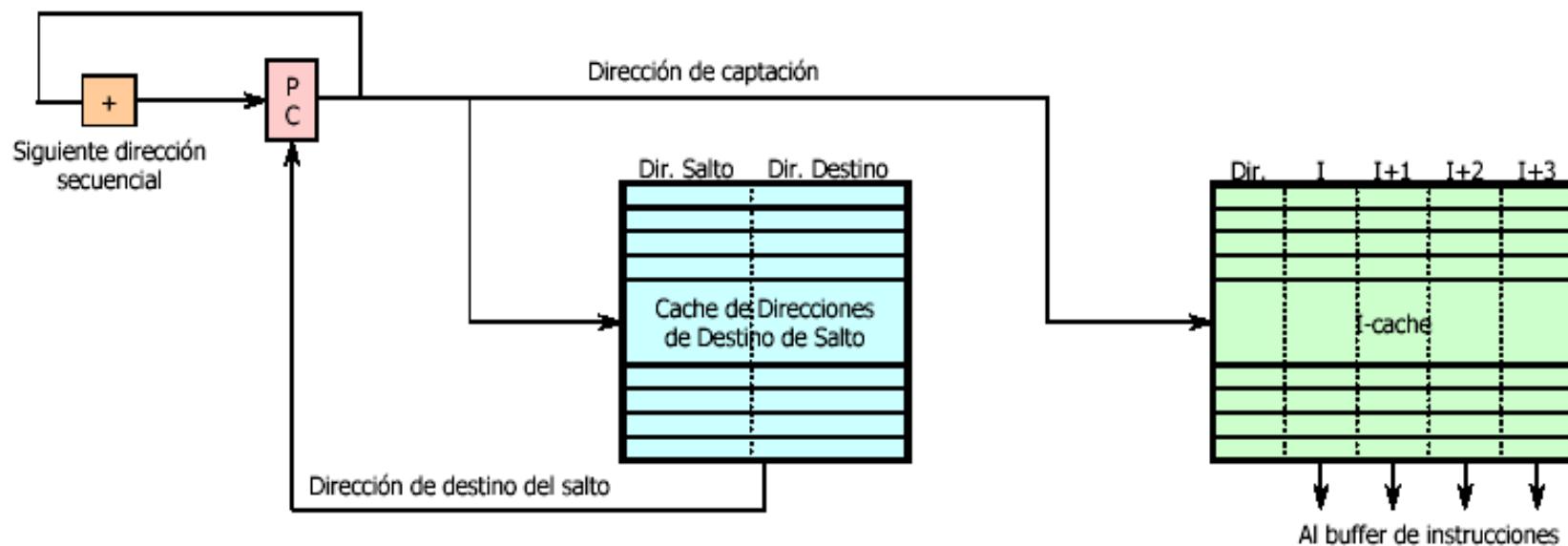
Riesgos

Gestión

Estructuras

Esquema de cache de direcciones de destino de salto (BTAC)

- Se añade una cache que contiene las direcciones de las instrucciones destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Se leen las direcciones al mismo tiempo que se captan las instrucciones de salto.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

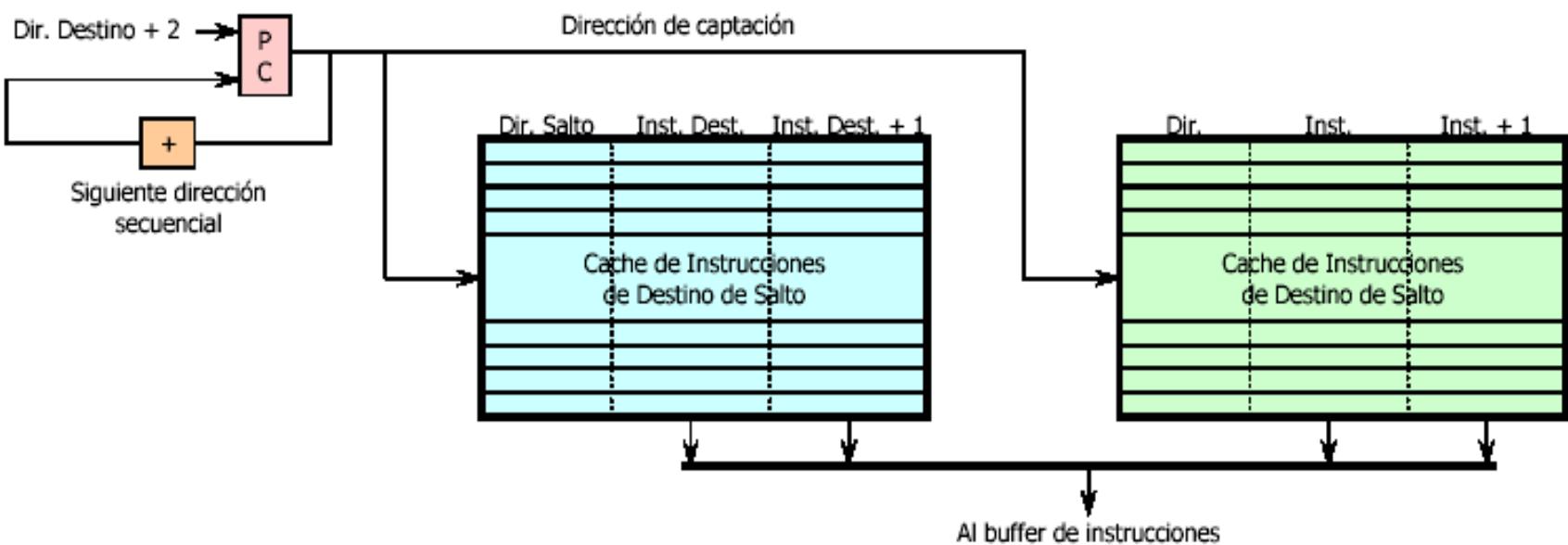
Riesgos

Gestión

Estructuras

Esquema de cache de instrucciones de destino de salto (BTIC)

- Se añade una cache que contiene las instrucciones siguientes a la dirección de destino de los saltos, junto con las direcciones de las instrucciones de salto.
- Sólo tiene sentido si la cache de instrucciones tiene una latencia muy alta.
- Mientras se procesan estas instrucciones se calcula la dirección de las siguientes.



Unidad 2. Superescalares

2.3. Gestión de riesgos (control)

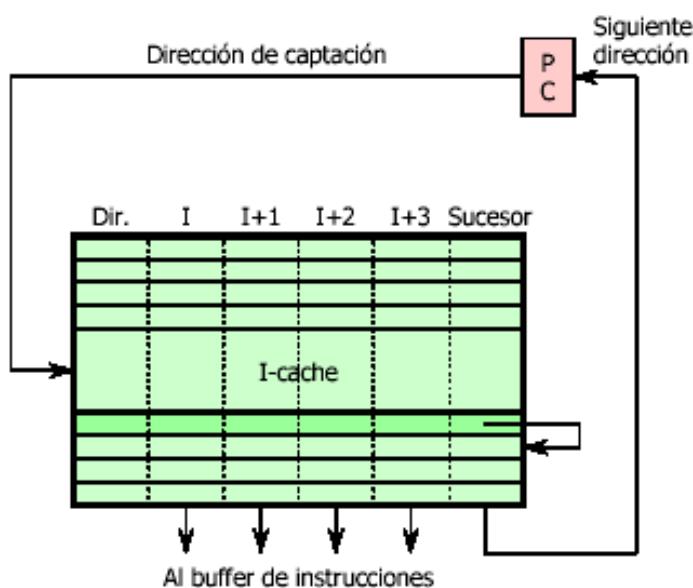
Riesgos

Gestión

Estructuras

Esquema de índice sucesor en la cache de instrucciones

- La cache de instrucciones contiene un índice sucesor que apunta a la siguiente línea de la cache de instrucciones que hay que captar (la siguiente, o la que se predice que se debe captar si hay una instrucción de salto condicional en esa línea).



Ejemplos y evolución de los esquemas de Acceso

Calcular/captar	BTIC	BTAC	Índice sucesor
i486 (1989)	→	Pentium (1993)	
MC68040 (1990)	→	MC 68060 (1993)	
		Am 29000 (1988) →	Am 29000 superscalar (1995)
Sparc CYC 600 (1992) SuperSparc (1992)		→	UltraSparc (1995)
R4000 (1992) R10000 (1996)		→	R8000 (1994)
PowerPC 601 (1993) PowerPC 603 (1993)	→	PowerPC 604 (1995) PowerPC 620 (1996)	

Ingeniería de los Computadores

Unidad 3. Computación paralela

Ingeniería de los Computadores

3.1 Conceptos y motivación

Introducción

¿Dónde se usa la supercomputación?

- **Defensa**
 - Lucha antiterrorista
 - Programas de armas
 - Programa espacial
- **Ingeniería/Academia/Investigación**
 - Prospección de nuevas energías y simulación
 - Nanotecnología
 - Modelado biológico
 - *Deep learning*
 - Investigación genoma
 - Investigación proteínas
 - Simulaciones astrofísicas
 - Predicción meteorológica
 - Diseño de fármacos
 - Modelado geológico
 - ...

Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Procesamiento paralelo versus procesamiento distribuido

Procesamiento paralelo

- Estudia los aspectos relacionados con la **división de una aplicación en unidades independientes** y su ejecución en múltiples procesadores para reducir tiempo y/o aumentar la complejidad del problema.

Procesamiento distribuido

- Estudia los aspectos relacionados con la ejecución de **múltiples aplicaciones** al mismo tiempo utilizando múltiples recursos (procesadores, memorias, discos y bases de datos) situados en distintas localizaciones físicas.

Ingeniería de los Computadores

3.1 Conceptos y motivación

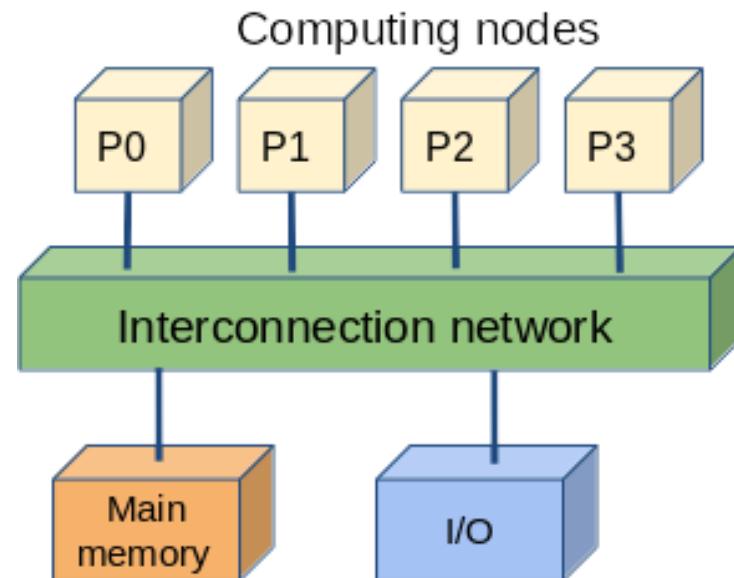
Conceptos

Clasificación de computadores paralelos (según el sistema de memoria):

- **Multiprocesadores:** Comparten el mismo espacio de memoria (el programador no necesita saber dónde están los datos)
- **Multicomputadores:** Cada procesador (nodo) tiene su propio espacio de direcciones (el programador necesita saber dónde están los datos)

Diseño de un computador paralelo:

- Nodos de cómputo
- Sistema de memoria
- Sistema de comunicación
- Sistema de entrada/salida



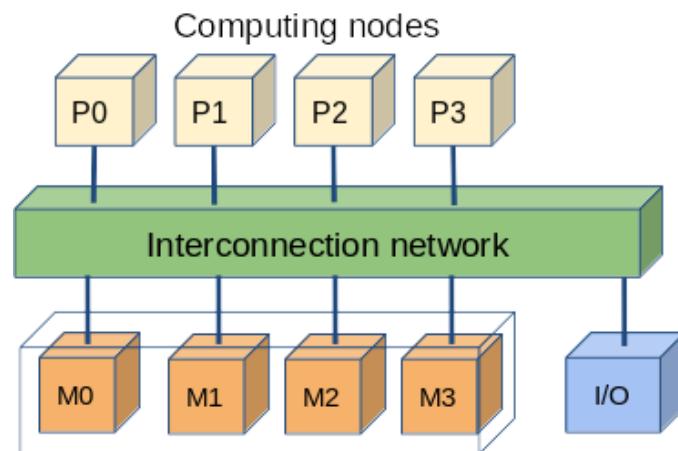
Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Multiprocesador con memoria centralizada (*SMP – Symmetric MultiProcessor*)

- Mayor latencia
- Poco escalable
- Comunicación mediante variables compartidas (datos no duplicados en memoria principal)
- Necesita implementar primitivas de sincronización
- No necesita distribuir código y datos
- Programación más sencilla (generalmente)

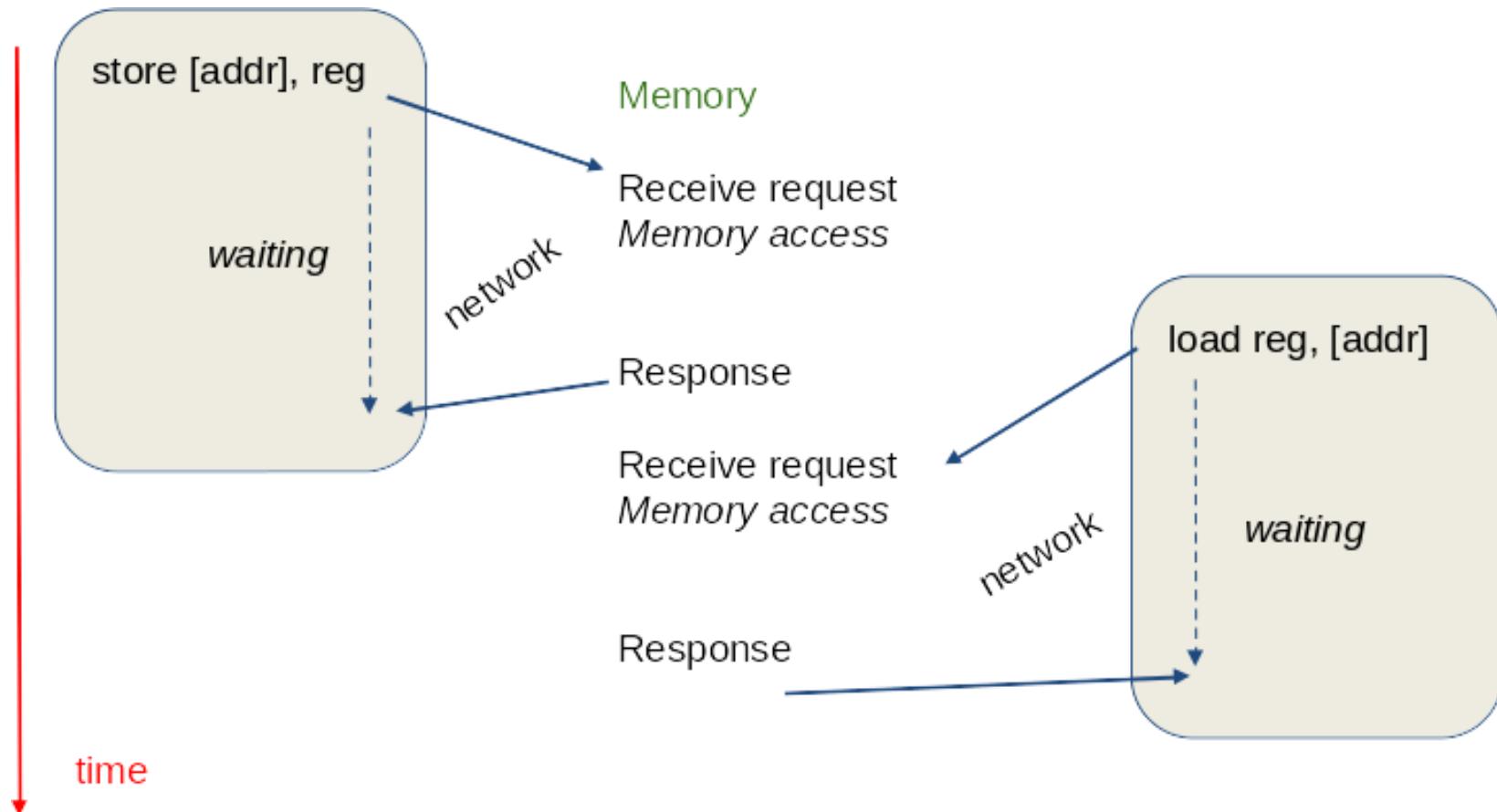


Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Comunicación en un multiprocesador



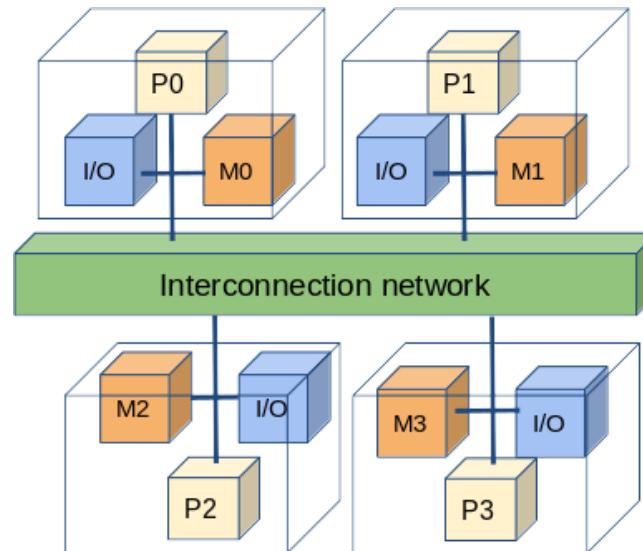
Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Multicomputador:

- Menor latencia
- Mayor escalabilidad
- Comunicación mediante **paso de mensajes** (datos duplicados en memoria)
- Sincronización mediante mecanismos de comunicación
- Distribución de carga de trabajo (código y datos) entre procesadores
- Programación más difícil

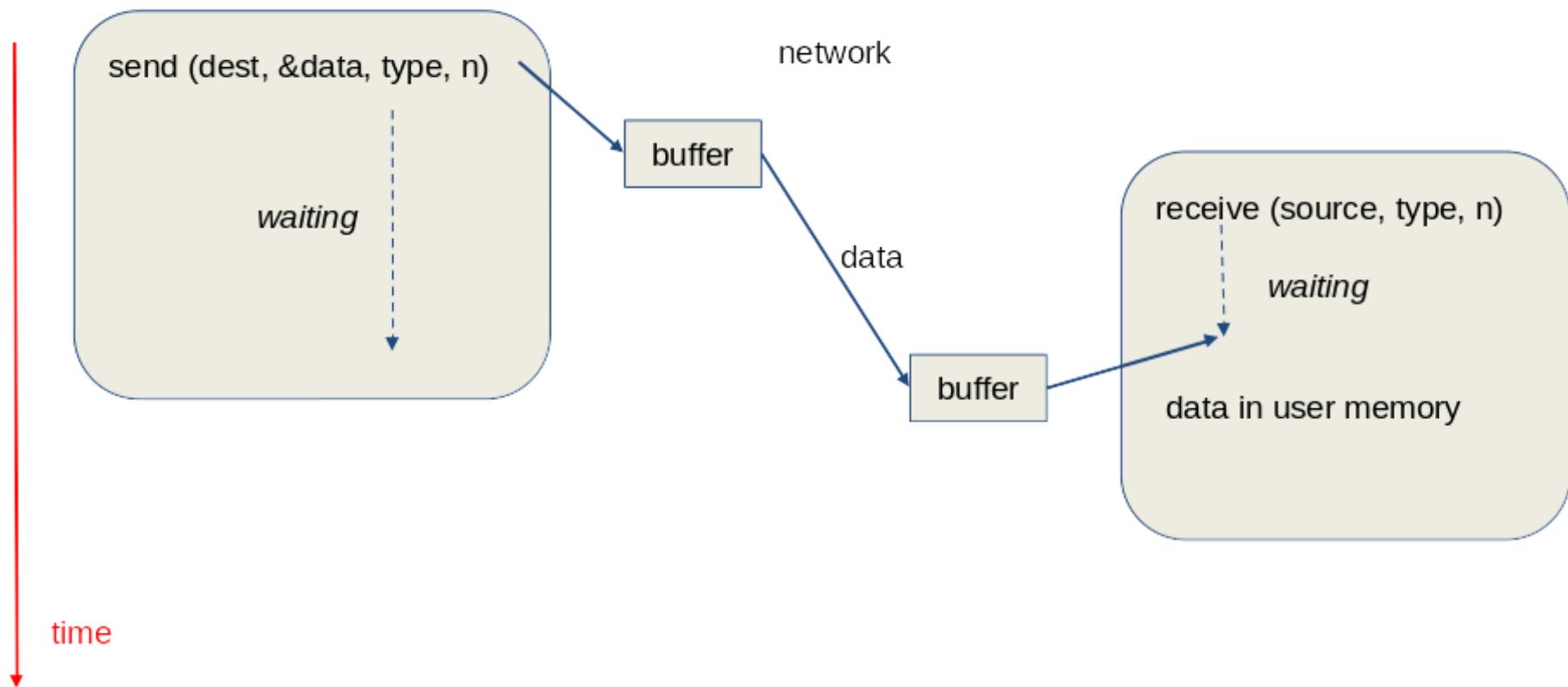


Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Comunicación asíncrona en un multicomputador

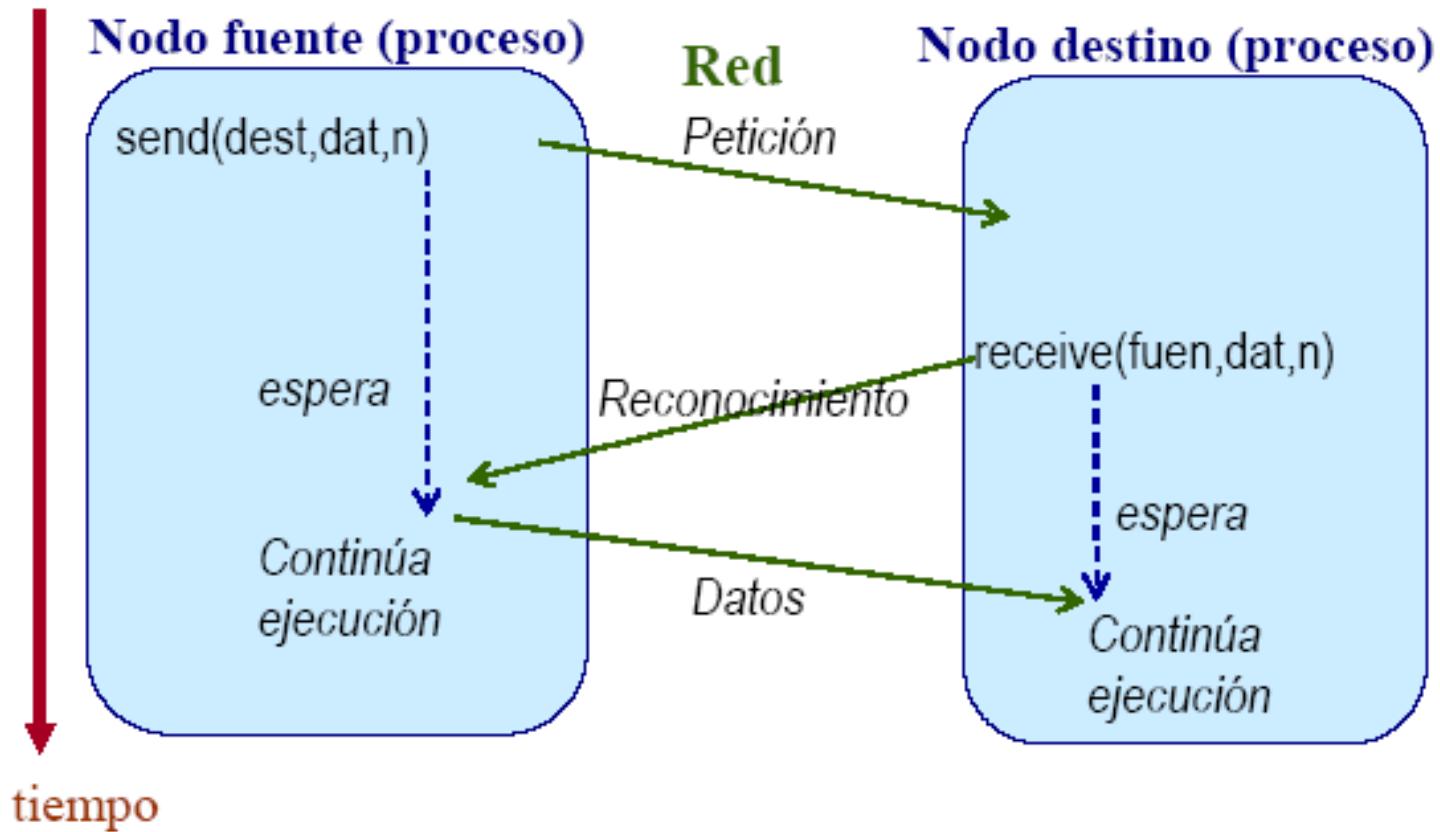


Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Comunicación síncrona en un multicomputador

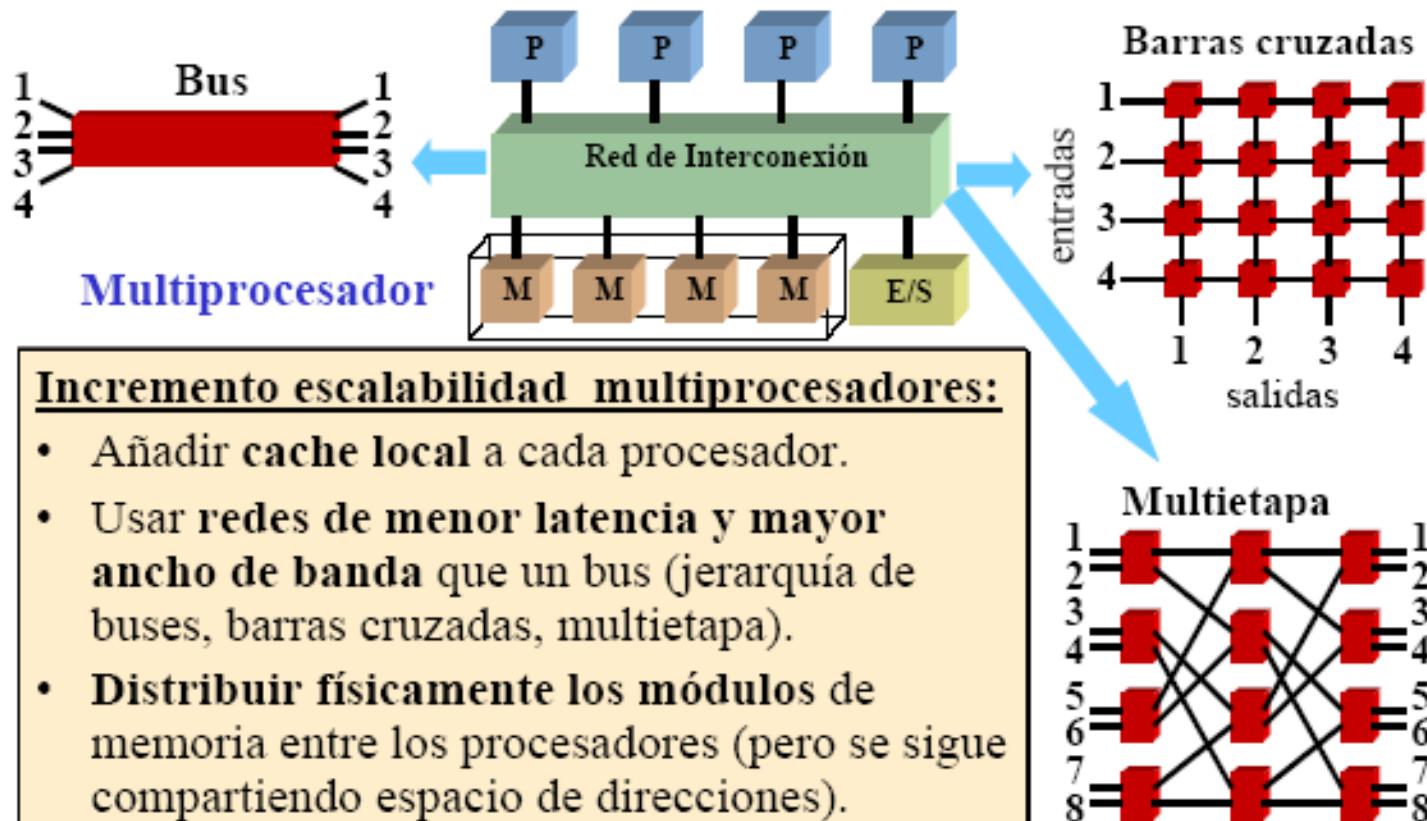


Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Redes de interconexión en multiprocesadores

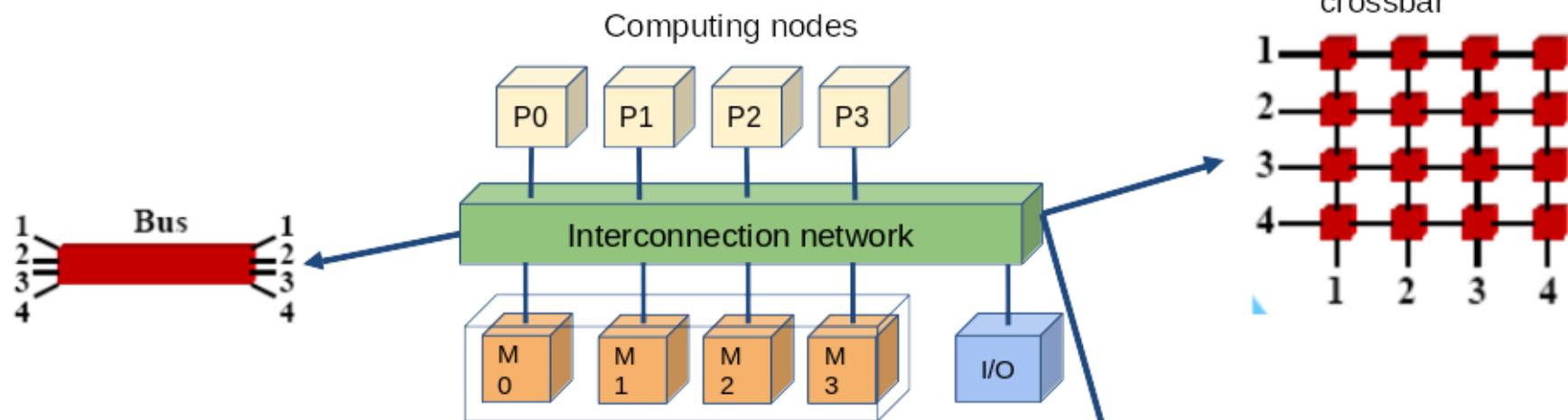


Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Redes de interconexión en multiprocesadores



¿Cómo aumentar la escalabilidad en multiprocesadores?

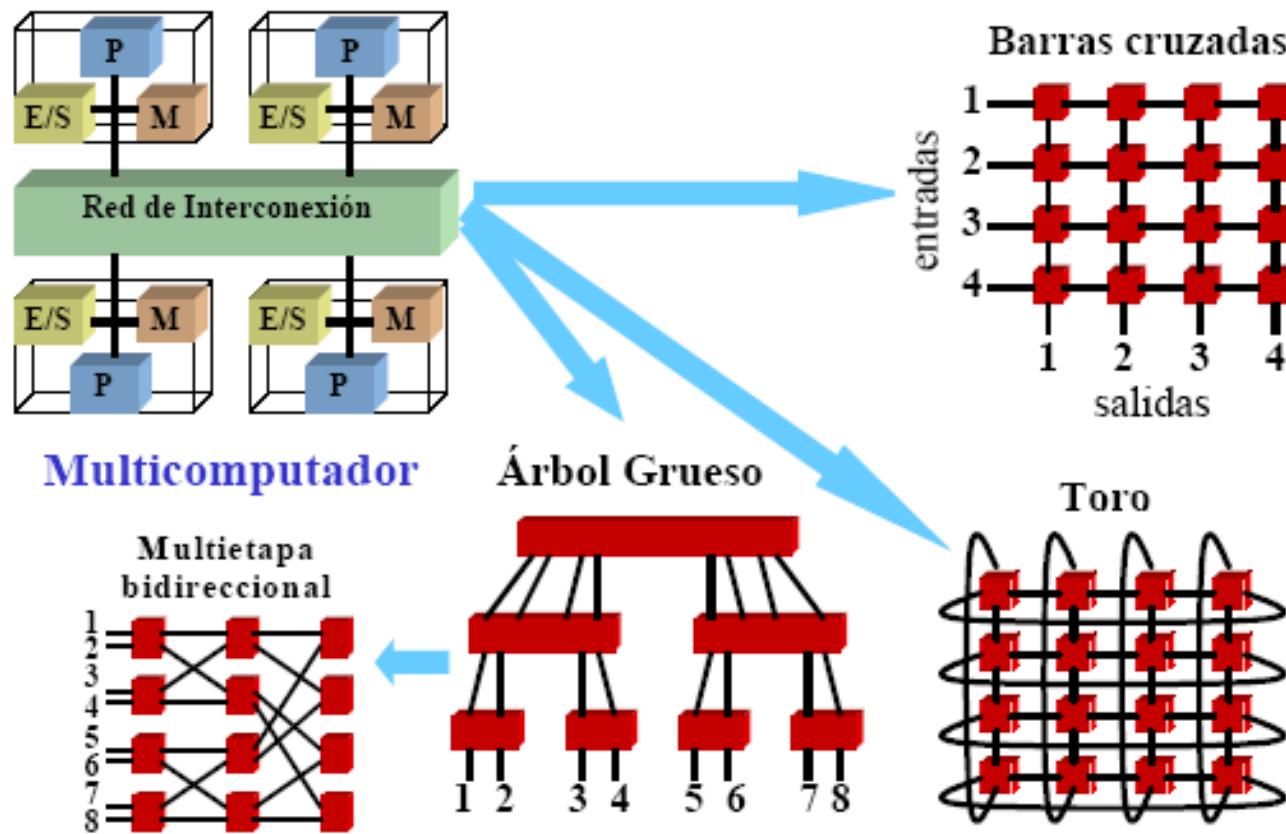
- Usando cachés locales a cada nodo -> NUMA
- Usando redes con menos latencia y más ancho de banda que un bus
- Convertir el UMA a NUMA

Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Redes de interconexión en multicomputadores



Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Clasificación de computadores paralelos

Multicomputadores Memoria no compartida Múltiples espacios de direcciones.	IBM-SP2 HP AlphaServer SC45 (cluster de SMP)	Memoria físicamente distribuida	+
Multiprocesadores Memoria compartida Un único espacio de direcciones.	NUMA (<i>Non-Uniform Memory Access</i>)	NUMA	Cray T3E, Cray X1
	CC-NUMA	CC-NUMA	Origin 3000 HP 9000 Superdome
	COMA	COMA	KSR-1
	UMA (<i>Uniform Memory Access</i>)	SMP	WS, servidores básicos
			Memoria físicamente centralizada

Ingeniería de los Computadores

3.1 Conceptos y motivación

Conceptos

Computadores paralelos y escalabilidad

Podemos ordenar los computadores paralelos en función de su escalabilidad creciente:

UMA/SMP < COMA < CC-NUMA < NUMA < Multicomputers

Ejemplos:

Multicomputers: IBM-SP2, HP AlphaServer SC45 (clúster o SMP)

NUMA : Cray T3E, Cray X1

CC-NUMA : Origin 3000, HP 9000 Superdome

COMA: KSR-1

SMP: ¡Intentad adivinar!

Otros tipos de computadores paralelos

- **MPP (Massive Parallel Processor)**. Número de procesadores superior a 100. Sistemas con redes diseñadas a medida.
- **Cluster**. Computadores completos (PC's, servidores, etc.) conectados con alguna red comercial tipo LAN que se utiliza como recurso de cómputo único. El tráfico de la red se reduce al ocasionado por la aplicación ejecutada (no hay tráfico externo)
- **Cluster Beowulf**. Cluster con sistema operativo libre (Linux) y hardware y software de amplia difusión
- **Constelaciones**. Cluster de SMP con un nº de procesadores en un nodo mayor que el nº de nodos del cluster

Conceptos

Otros tipos de computadores paralelos

- **Redes de computadores.** Conjunto de computadores conectados por una LAN. Por la red circula tanto tráfico de la aplicación paralela como externo
- **GRID.** Conjunto de recursos autónomos distribuidos geográficamente conectados por infraestructura de telecomunicaciones y que conforman un sistema de altas prestaciones virtual

Tipos de paralelismo

Paralelismo funcional:

- Se obtiene a través de la **reorganización de la estructura lógica** de una aplicación. Existen diferentes niveles de paralelismo funcional según las estructuras que se reorganicen.

Paralelismo de datos:

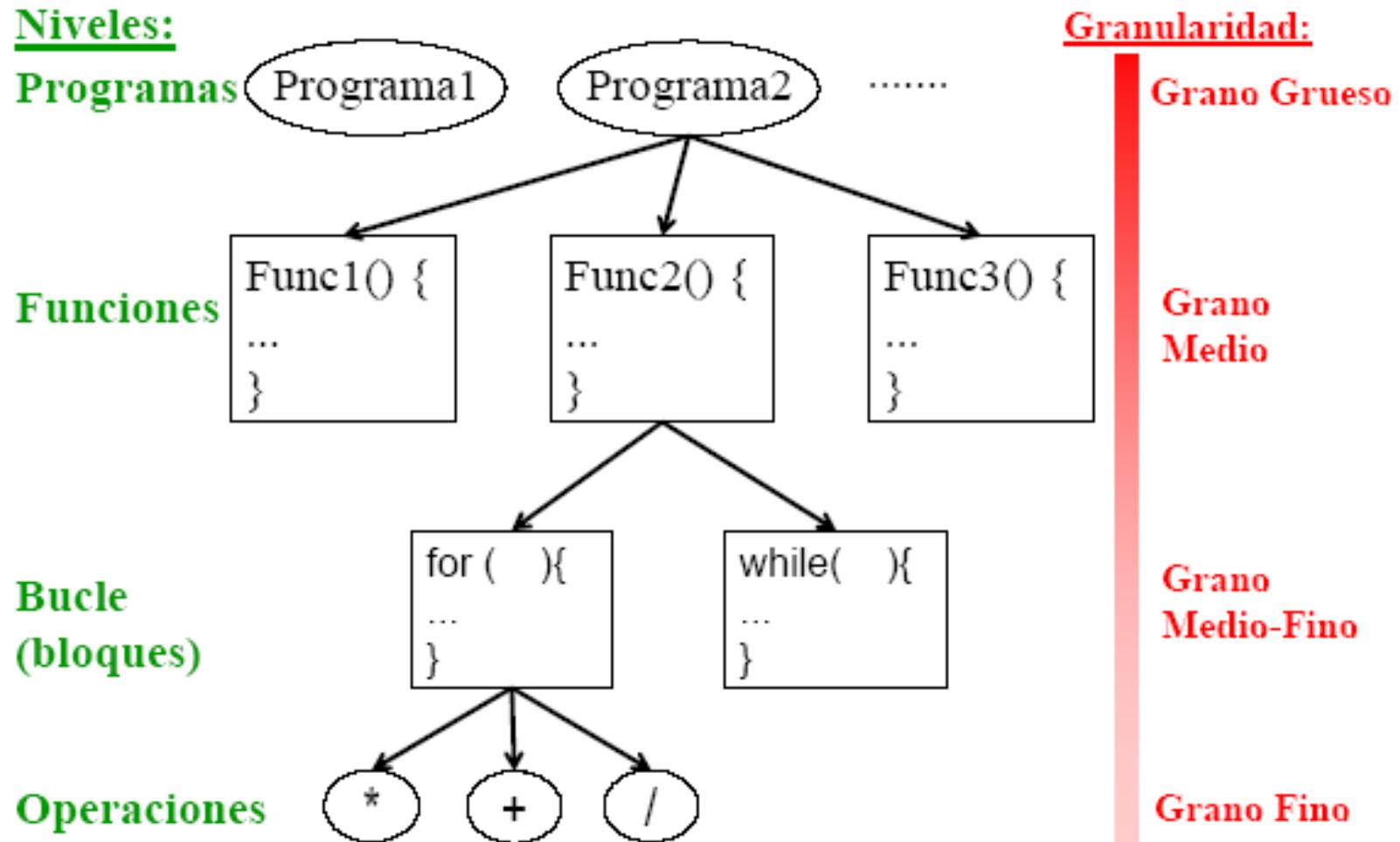
- Implícito en operaciones con estructuras de datos tipo **vector** o **matriz**. Está relacionado con operaciones realizadas sobre grandes volúmenes de datos que sean independientes entre si.

Ingeniería de los Computadores

3.1 Conceptos y motivación

Paralelismo

Niveles y granularidad del paralelismo funcional:



Ingeniería de los Computadores

3.1 Conceptos y motivación

Paralelismo

Tipos de paralelismo con respecto a su visibilidad:

Paralelismo explícito

- Paralelismo no presente de forma inherente en las estructuras de programación y que **se debe indicar expresamente**.

Paralelismo implícito

- Paralelismo presente (aunque puede no estar ejecutándose de forma paralela) debido a la propia estructura de los datos (vectores) o de la aplicación (bucle)

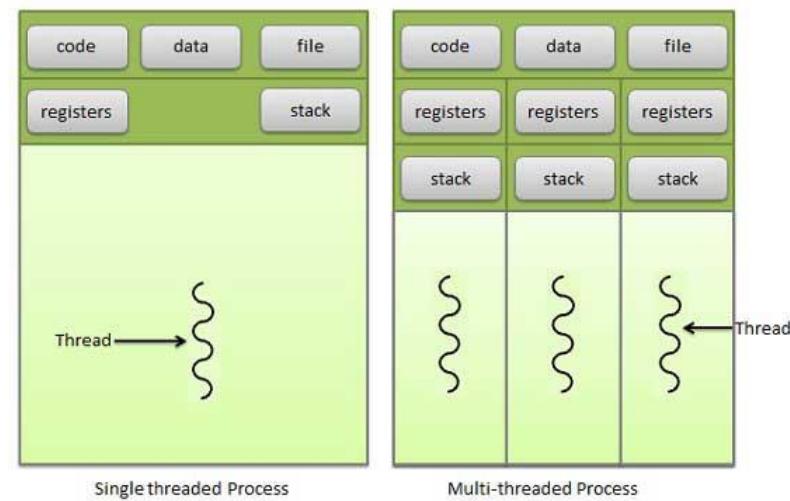
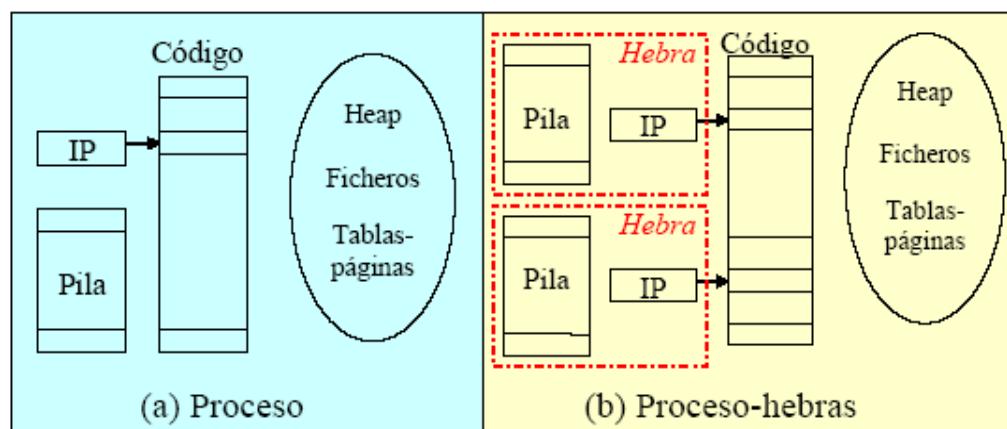
Ingeniería de los Computadores

3.1 Conceptos y motivación

Parallelismo

Unidades de ejecución: hilos y procesos

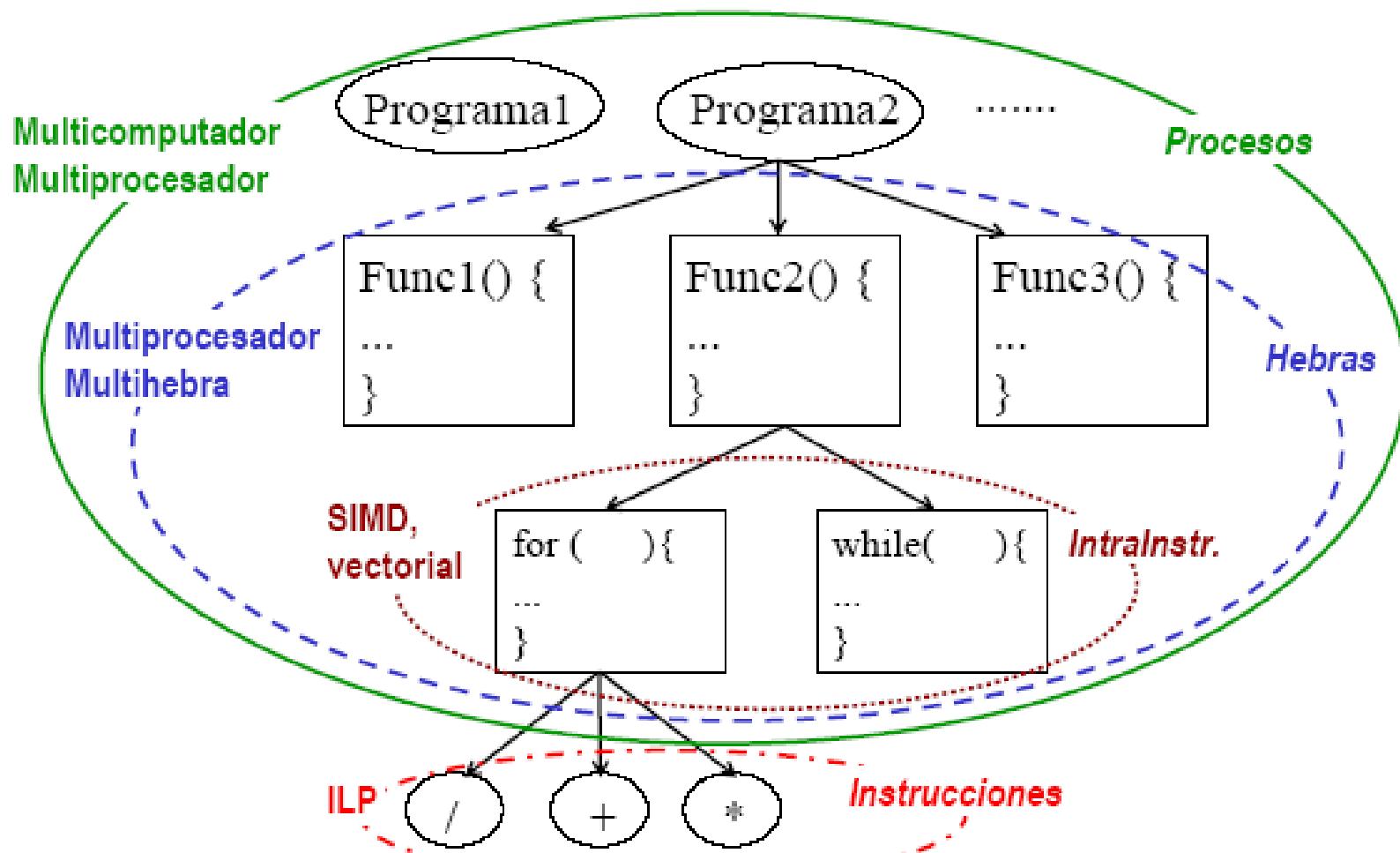
- Hardware (procesador): Gestiona la ejecución de las instrucciones
- Software (SO): Gestiona la ejecución de hilos y procesos
 - **Proceso**: espacio de direcciones virtuales propio
 - **Hilos/hebra/thread**: comparten direcciones virtuales, se crean y destruyen más rápido y la comunicación también es más rápida



Ingeniería de los Computadores

3.1 Conceptos y motivación

Paralelismo

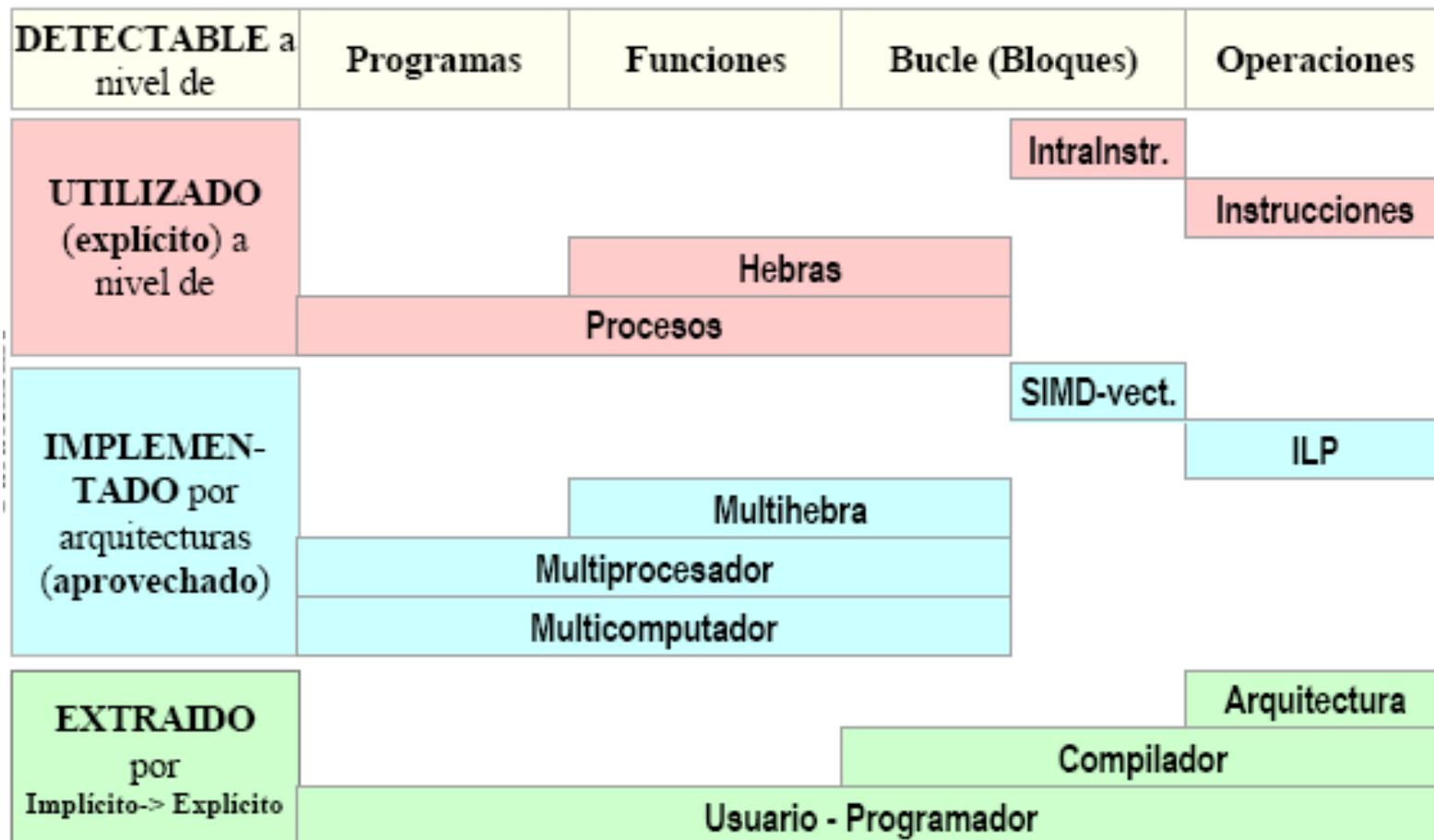


Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Detección y extracción de paralelismo



Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Problemas introducidos por la programación paralela

- División en unidades de cómputo independientes (tareas)
- Agrupación de tareas (código y datos) en procesos/hebras
- Asignación a procesadores
- Sincronización y comunicación

Situación inicial (normalmente)

- Se parte de una versión secuencial (no paralela)
- Se parte de una descripción de la aplicación
- Elementos de apoyo:
 - Programa paralelo que resuelva un problema semejante
 - Librerías de funciones paralelas (BLAS, ScalaPack, OpenMP, Intel TBB)

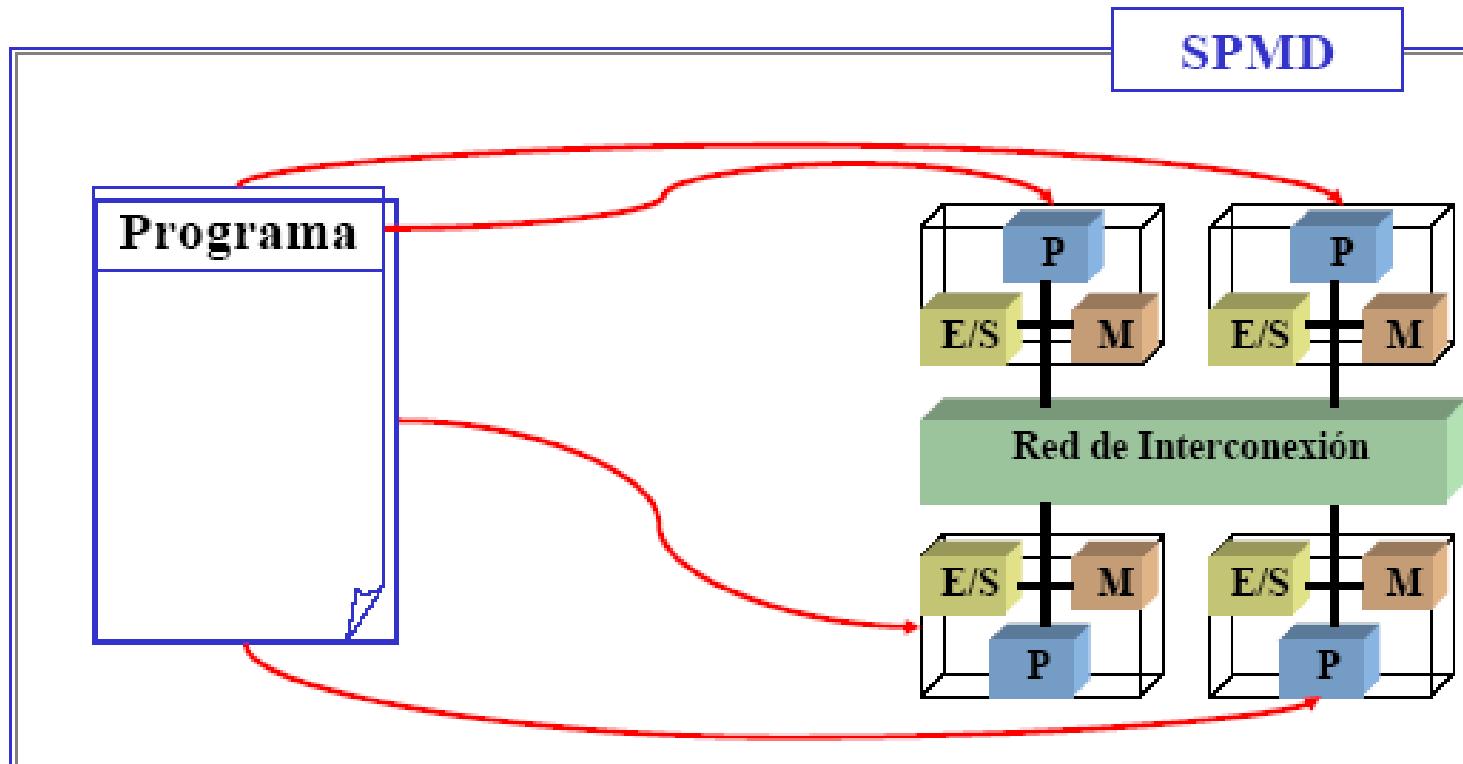
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Modos de programación paralela:

- **SPMD (Single Program Multiple Data)**



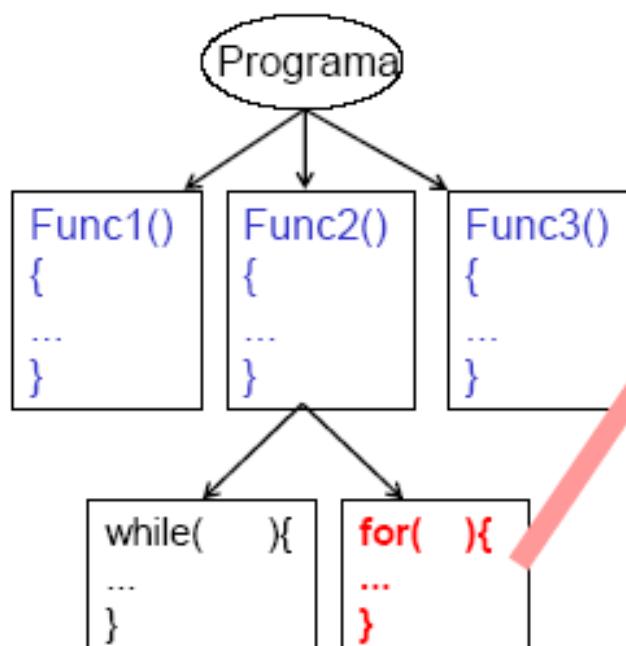
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Modos de programación paralela:

- **SPMD** (Single Program Multiple Data). Ejemplo



SPMD

```
Func1()  
...  
Func2()  
...  
for (i=ithread;i<N;i=i+nthread){  
    código para la iteración i  
    ...  
}  
Func3()  
...  
Main () {  
    ...  
switch (iproc) {  
        case 0: Func1(); break;  
        case 1: Func2(); break;  
        case 2: Func3(); break;  
    }  
    ...  
}
```

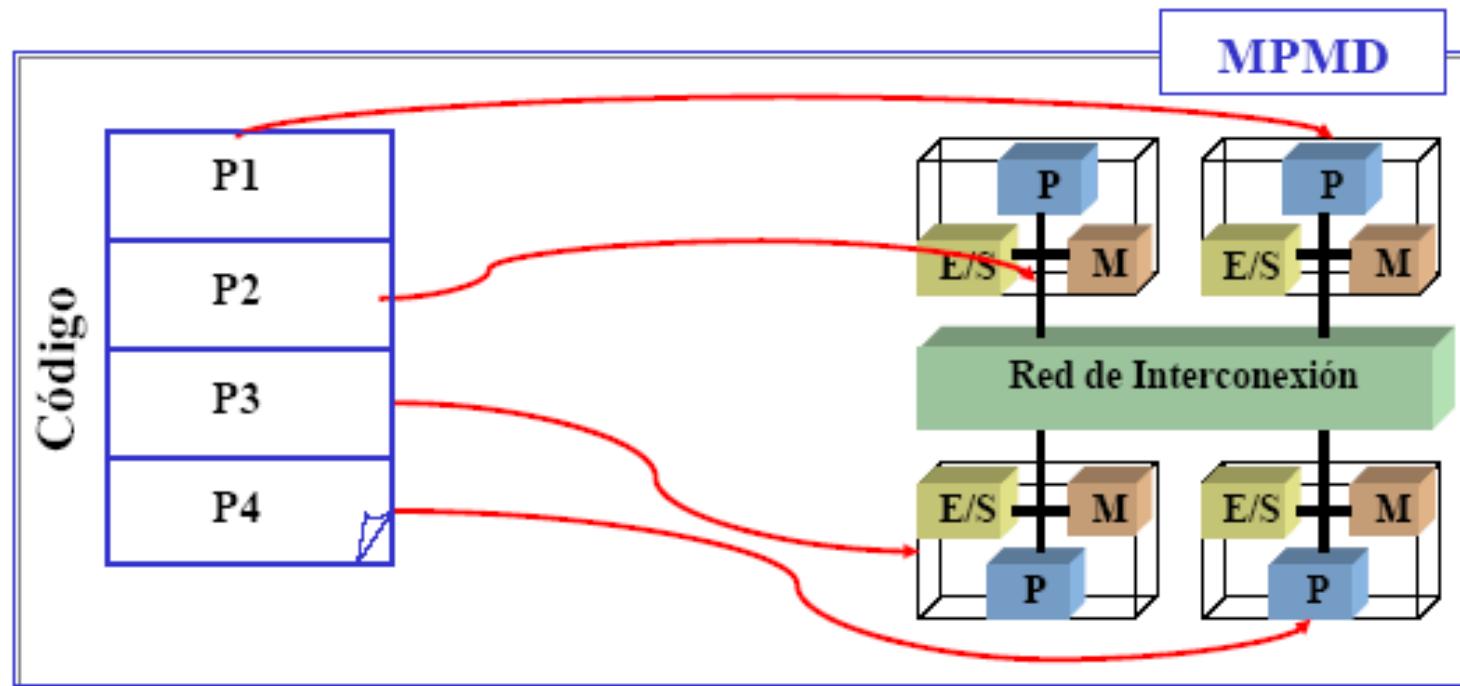
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Modos de programación paralela

- **MPMD** (Multiple Program Multiple Data)



- Modo Mixto SPMD-MPMD

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Herramientas que facilitan la programación paralela

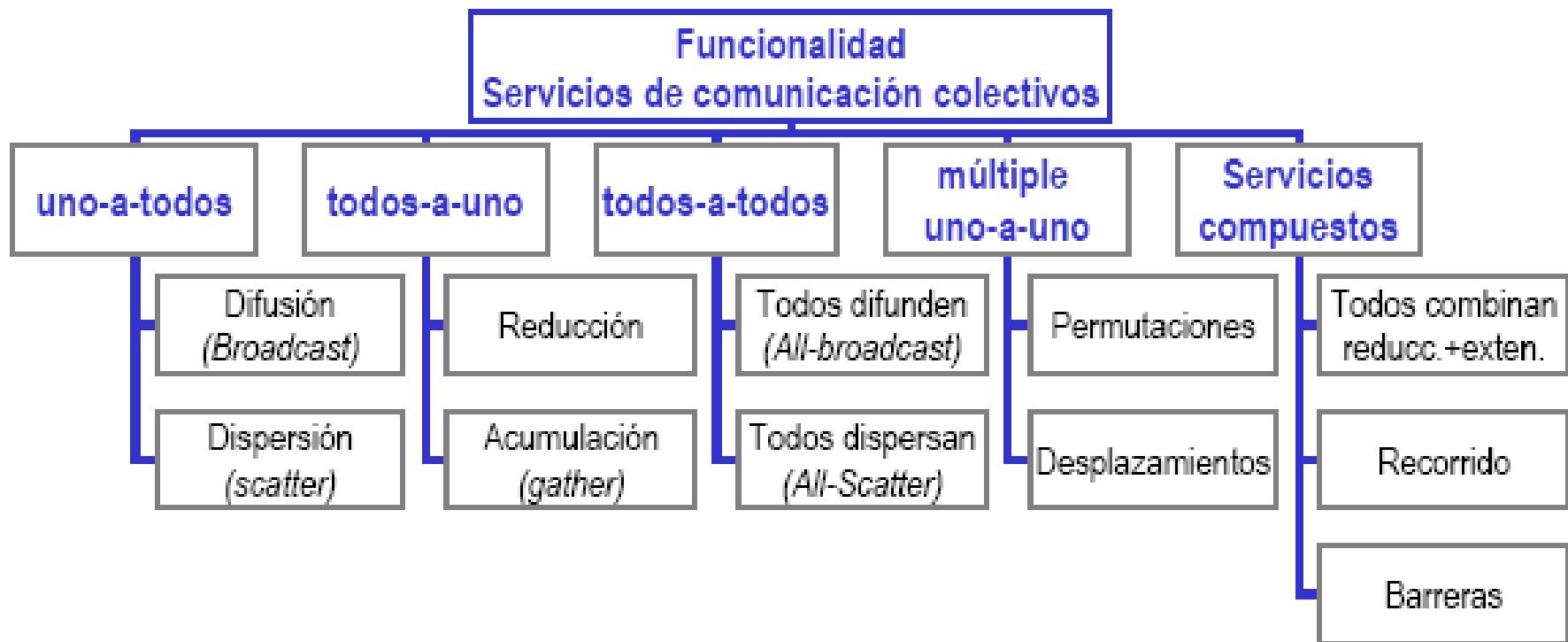
- Compiladores paralelos. Extracción automática del paralelismo
- Directivas del compilador (OpenMP, Intel TBB): lenguaje secuencial + directivas
- Lenguajes paralelos (HPF, Occam, Ada)
- Bibliotecas de funciones (*Pthreads*, MPI, PVM): lenguaje secuencial + funciones de biblioteca como interfaces

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Alternativas de comunicación:



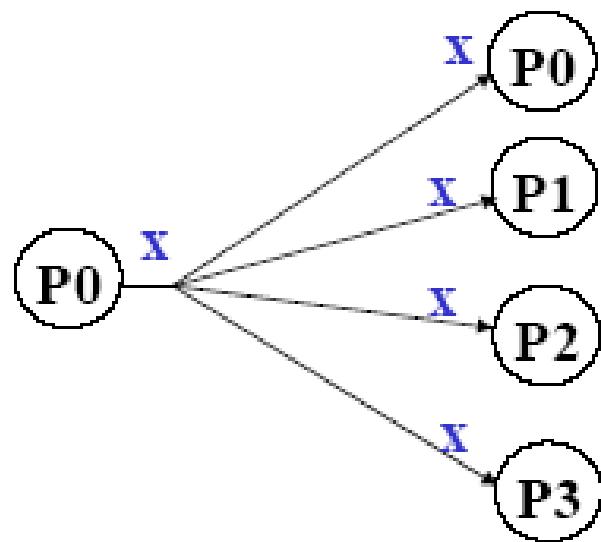
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

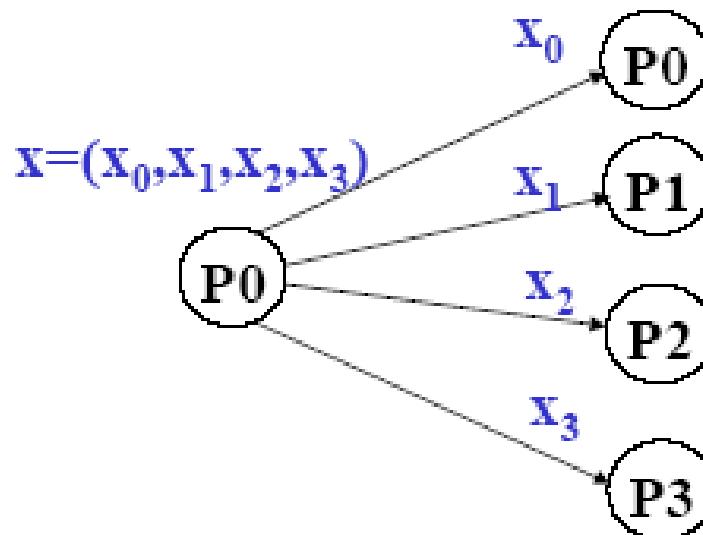
Paralelismo

Comunicación: uno a todos

Difusión (*broadcast*)



Dispersión (*scatter*)



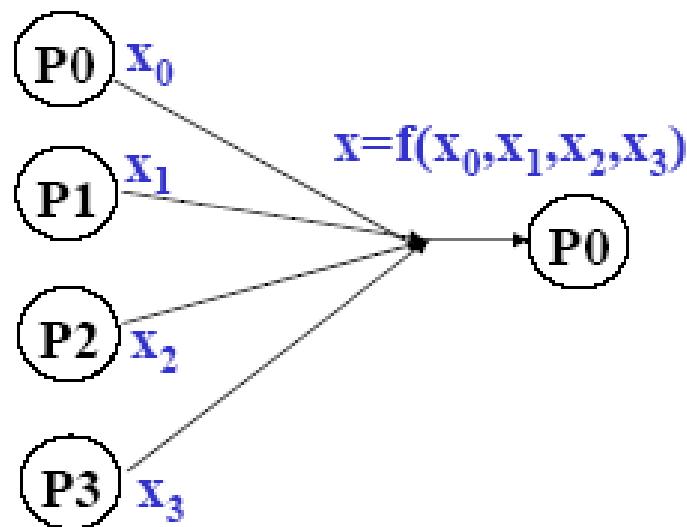
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

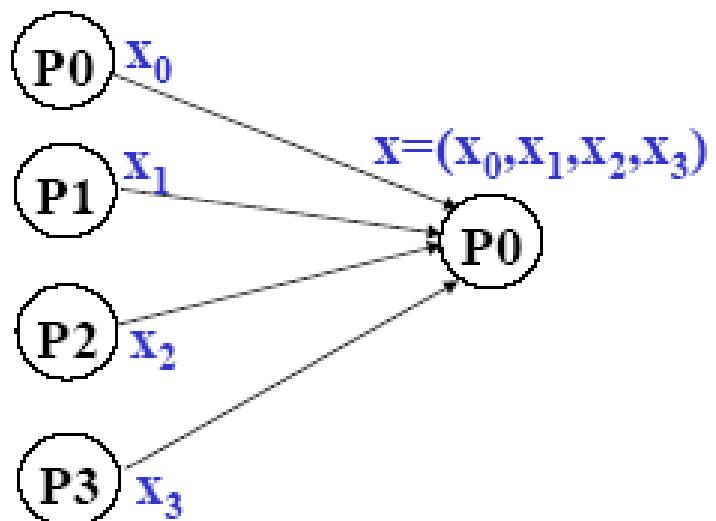
Paralelismo

Comunicación: todos a uno

Reducción



Acumulación (*gather*)

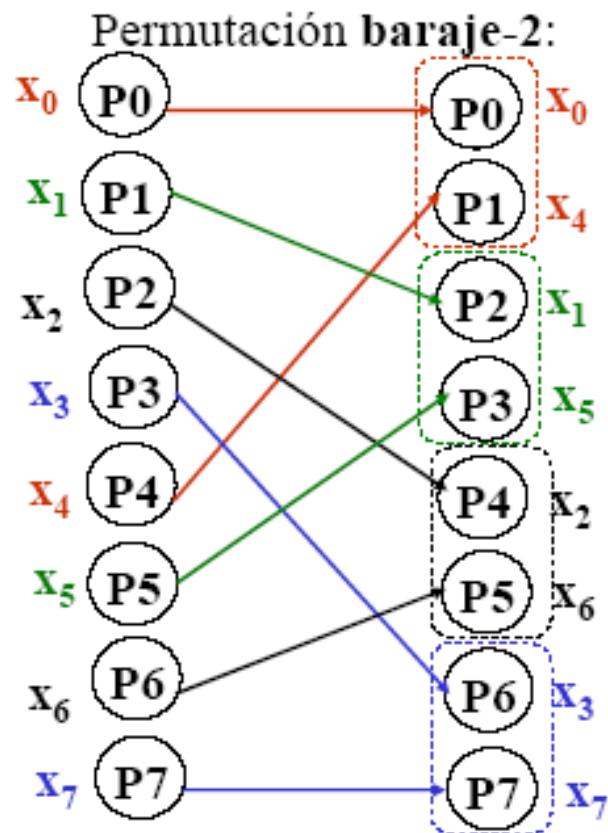
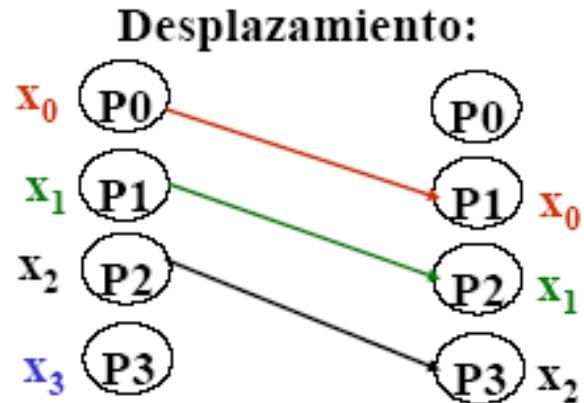
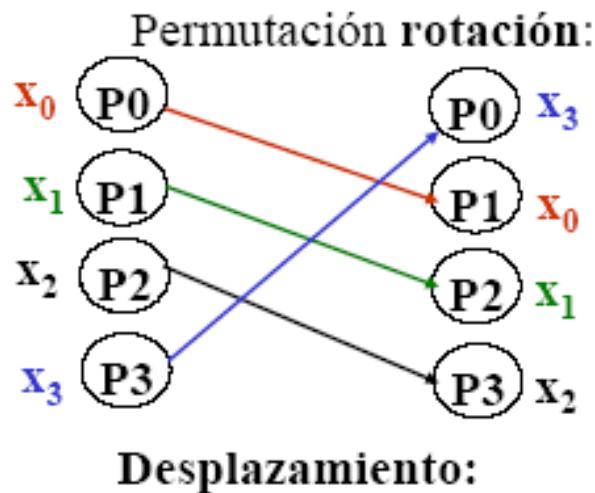


Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Comunicación: múltiple uno a uno



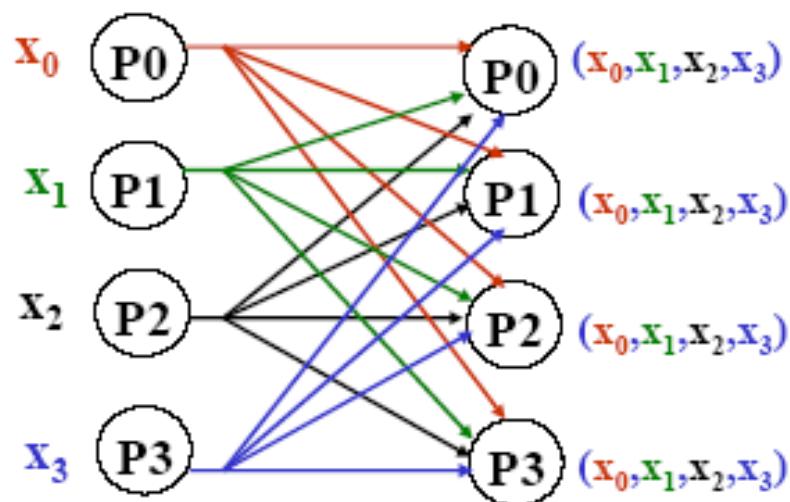
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

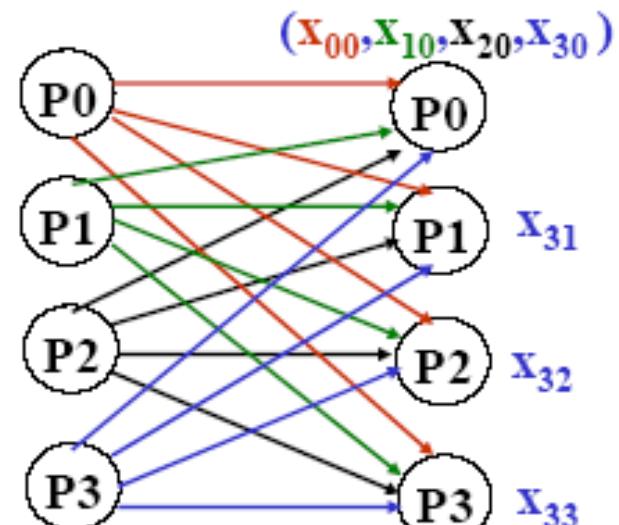
Paralelismo

Comunicación: todos a todos

Todos Difunden (*all-broadcast*)
o chismorreo (*gossiping*)



Todos Dispersan (*all-scatter*)



$$x = (x_{30}, x_{31}, x_{32}, x_{33})$$

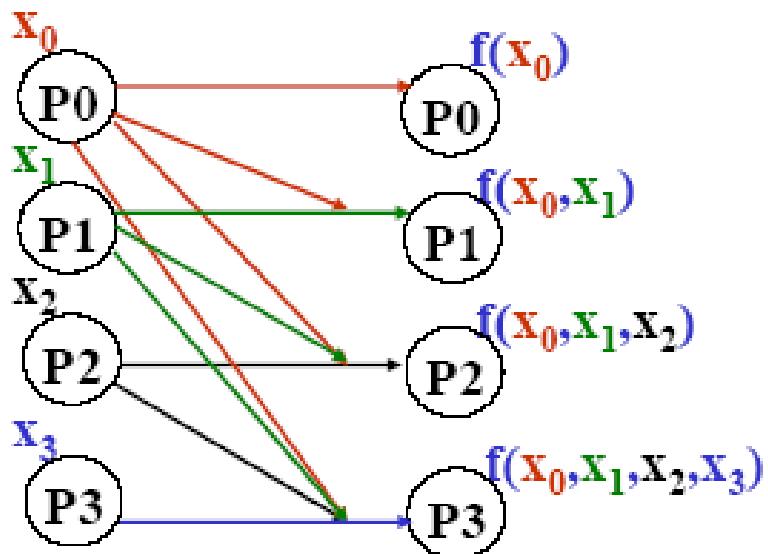
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

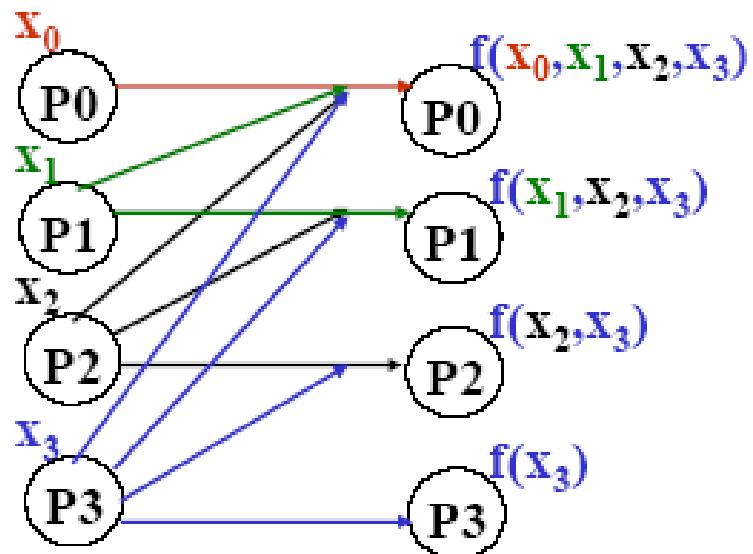
Paralelismo

Comunicación: servicios compuestos

Recorrido prefijo paralelo



Recorrido sufijo paralelo



Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Comunicación: Ejemplos de funciones colectivas usando MPI

Uno-a-todos	Difusión	MPI_Bcast()
	Dispersión	MPI_Scatter()
Todos-a-uno	Reducción	MPI_Reduce()
	Acumulación	MPI_Gather()
Todos-a-todos	Todos difunden	MPI_Allgather()
Servicios compuestos	Todos combinan	MPI_Allreduce()
	Barreras	MPI_Barrier()
Servicios compuestos	Scan	MPI_Scan

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Estilos de programación paralela: paso de mensajes

- Herramientas software
 - Lenguajes de programación (Ada) y librerías (MPI, PVM)
- Distribución de carga de trabajo
 - Uso de librerías: explícita con construcciones del lenguaje secuencial (if, for, ...)
 - Uso de lenguaje paralelo: explícita con construcciones del propio lenguaje (Occam: sentencia par)
- Primitivas básicas de comunicación : Send y Receive
- Funciones de comunicación colectivas
- Sincronización
 - Receive bloqueante. Barreras (MPI_BARRIER)
 - Send-receive bloqueante en ADA

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Estilo de programación paralela: variables compartidas

- Lenguajes de programación (Ada), Funciones de librerías (OpenMP), directivas del compilador (OpenMP)
- Distribución de la carga
 - Librerías: explícita con el lenguaje secuencial
 - Directivas: sincronización implícita (mayor nivel de abstracción)
 - Lenguajes: construcciones del lenguaje
- Comunicación básica: load y store
- Funciones de comunicación colectiva
- Sincronización
 - Semáforos, cerros, variables condicionales

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Estilos de programación paralela: parallelismo de datos

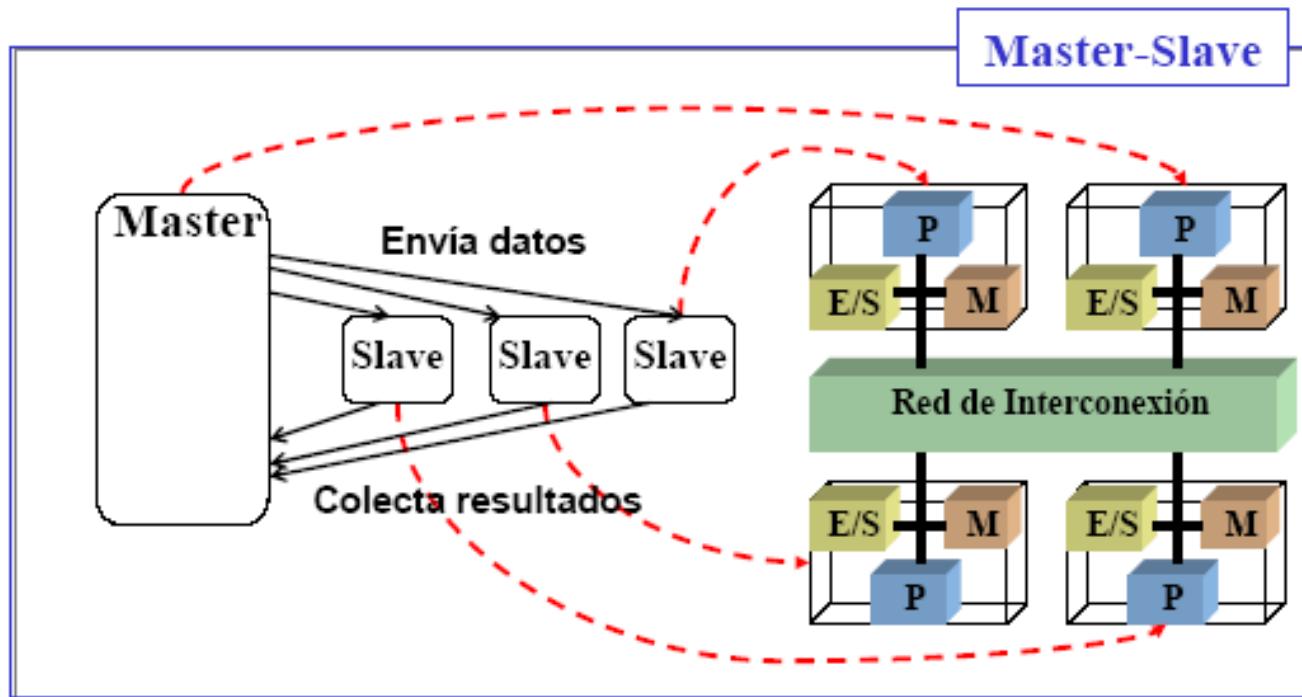
- Herramientas
 - Lenguaje de programación (HPF – High Performance Fortran)
- Distribución de carga
 - Directivas para distribuir datos entre procesadores
 - Directivas para parallelizar bucles
- Comunicación básica: implícita -> $A(i) = A(i-1)$
- Funciones de comunicación colectivas
 - Implícitas en funciones usadas explícitamente por el programador (rotaciones – CSHIFT)
- Sincronización: implícita

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Estructuras de paralelismo: *master-slave*



Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Estructuras de paralelismo: master slave

Master-Slave como MPMD– SPMD

```
main ()  
{   código Master  
}
```

```
main ()  
{   código Slaves  
}
```

Master-Slave como SPMD

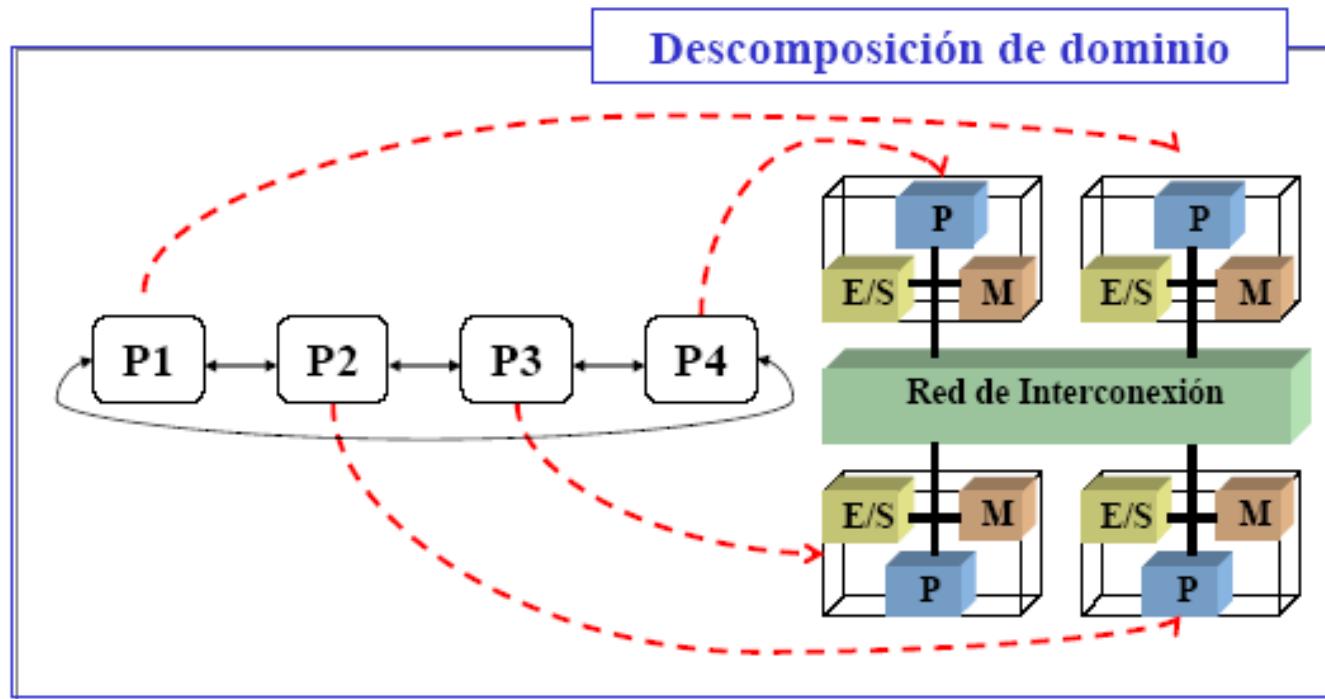
```
main ()  
{ if (iproc=id_Master) {  
    código Master  
} else {  
    código Slaves  
}
```

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Estructuras de paralelismo: descomposición de dominio (paralelismo de datos)

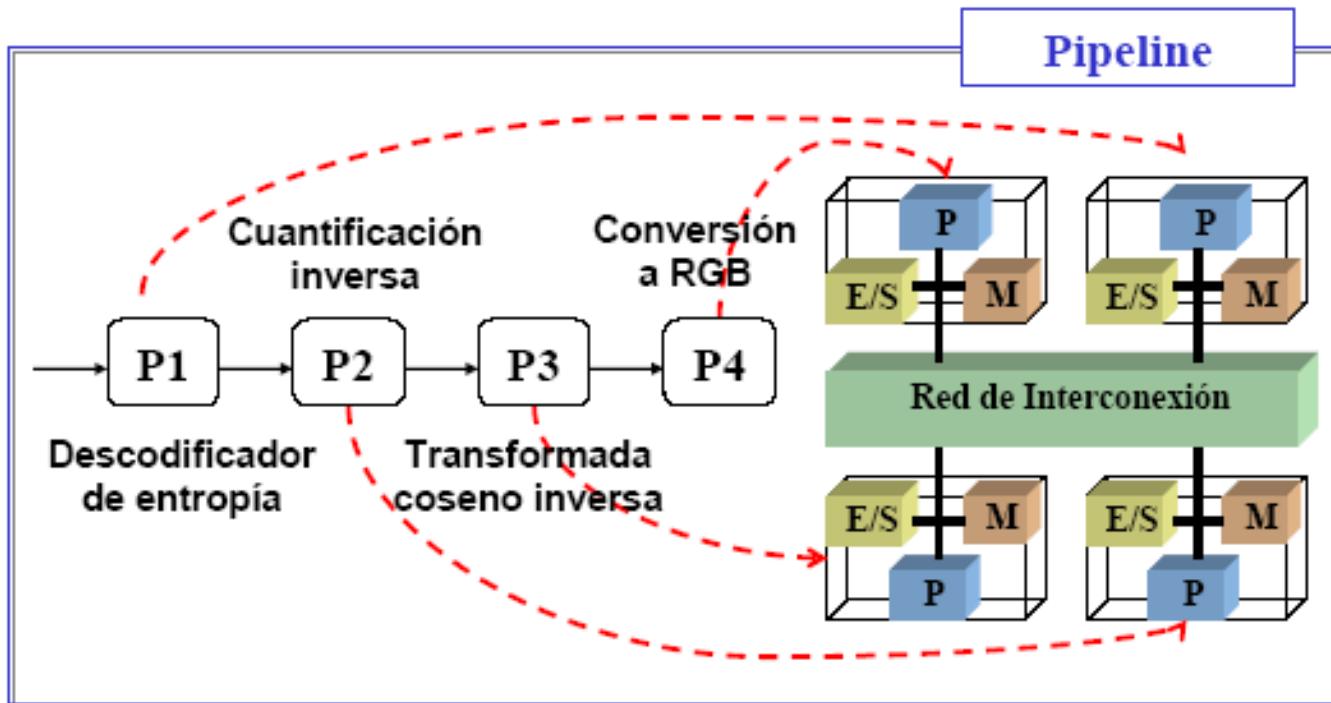


Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Estructuras de paralelismo: segmentada (flujo de datos)

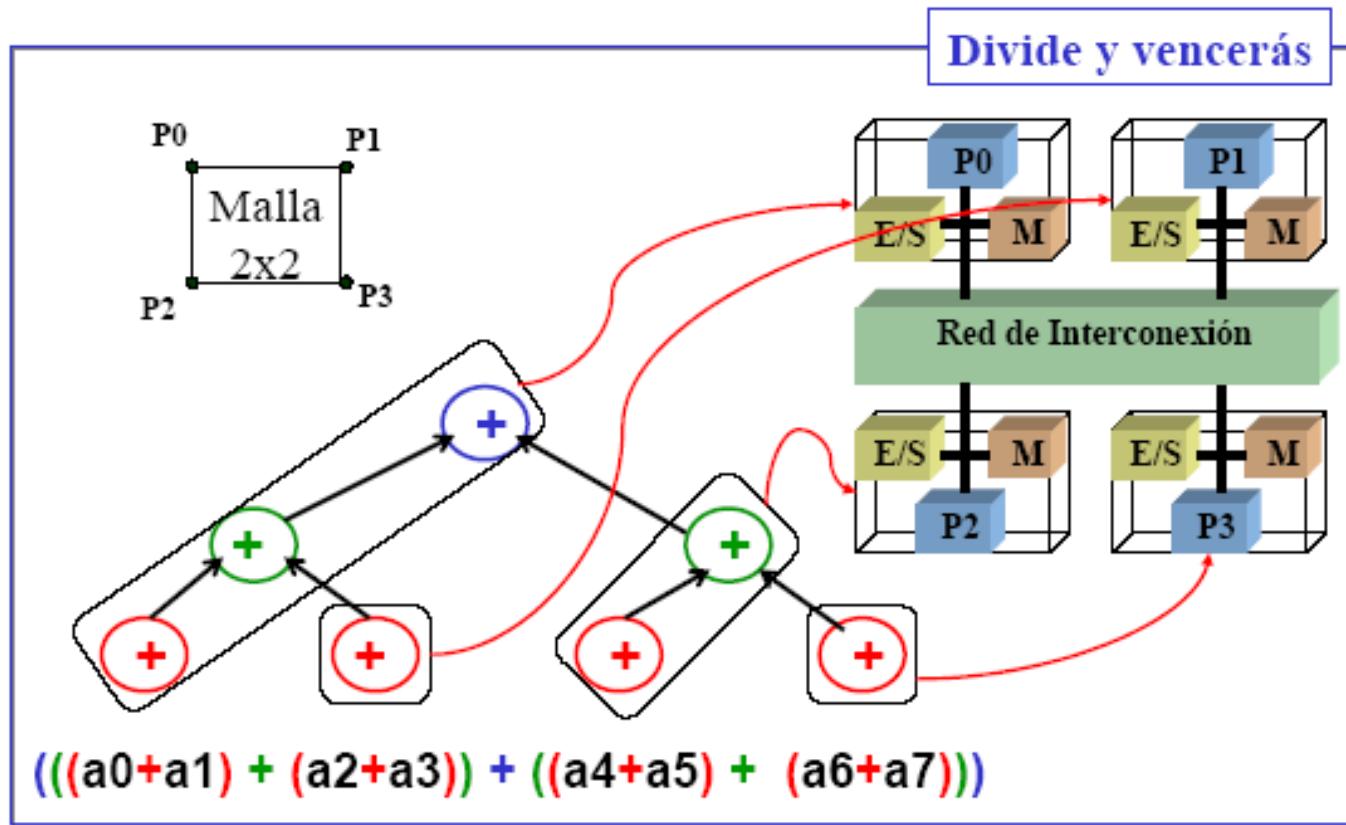


Ingeniería de los Computadores

3.2 Técnicas y procedimientos

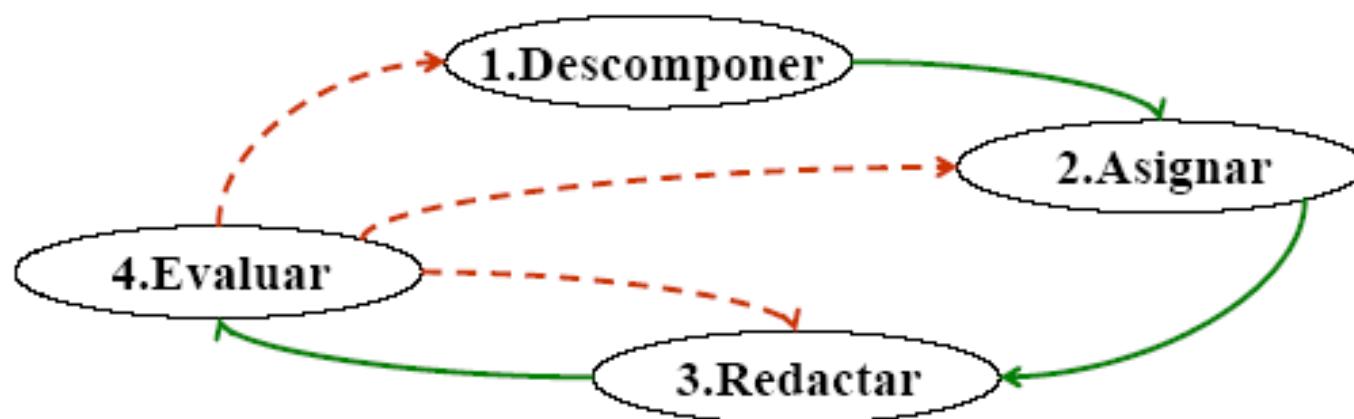
Paralelismo

Estructuras de paralelismo: divide y vencerás (descomposición recursiva)



Proceso de parallelización

- Descomposición en tareas independientes
- **Asignación** de tareas a procesos y/o hebras
- Redacción de código paralelo
- Evaluación de prestaciones



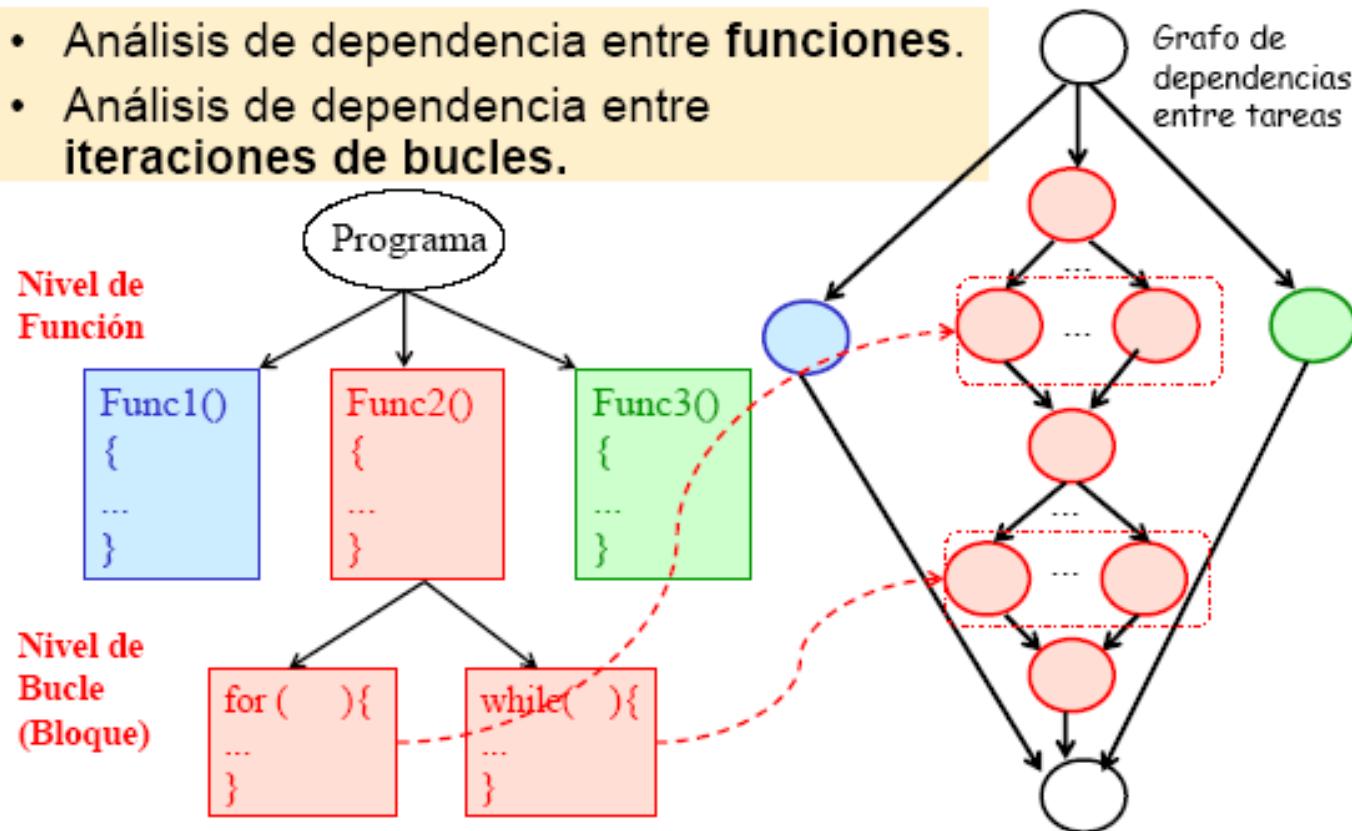
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Proceso de parallelización: descomposición en tareas independientes

- Análisis de dependencia entre funciones.
 - Análisis de dependencia entre iteraciones de bucles.



Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Proceso de parallelización: asignación de tareas

- Normalmente se asignan iteraciones de un ciclo a hebras y funciones a procesos
- La granularidad depende de
 - El número de procesadores
 - El tiempo de comunicación/sincronización frente al tiempo de cálculo
- Equilibrio en la carga de trabajo (que unos procesadores no esperen a otros)
- Tipos de asignación
 - Dinámica (en tiempo de ejecución). Si no se conoce el número de tareas
 - Estática (programador o compilador)

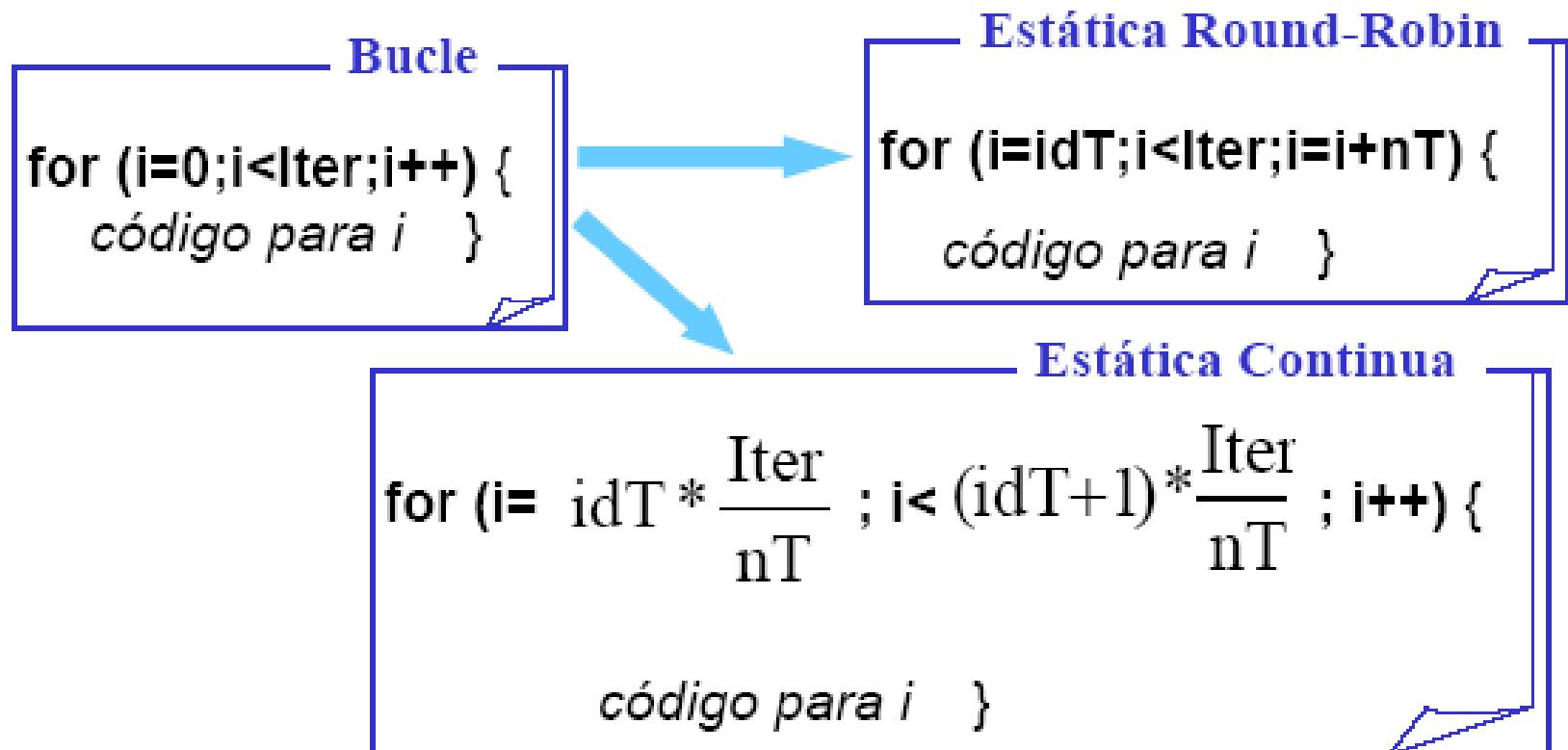
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Proceso de parallelización: asignación de tareas

- Asignación estática



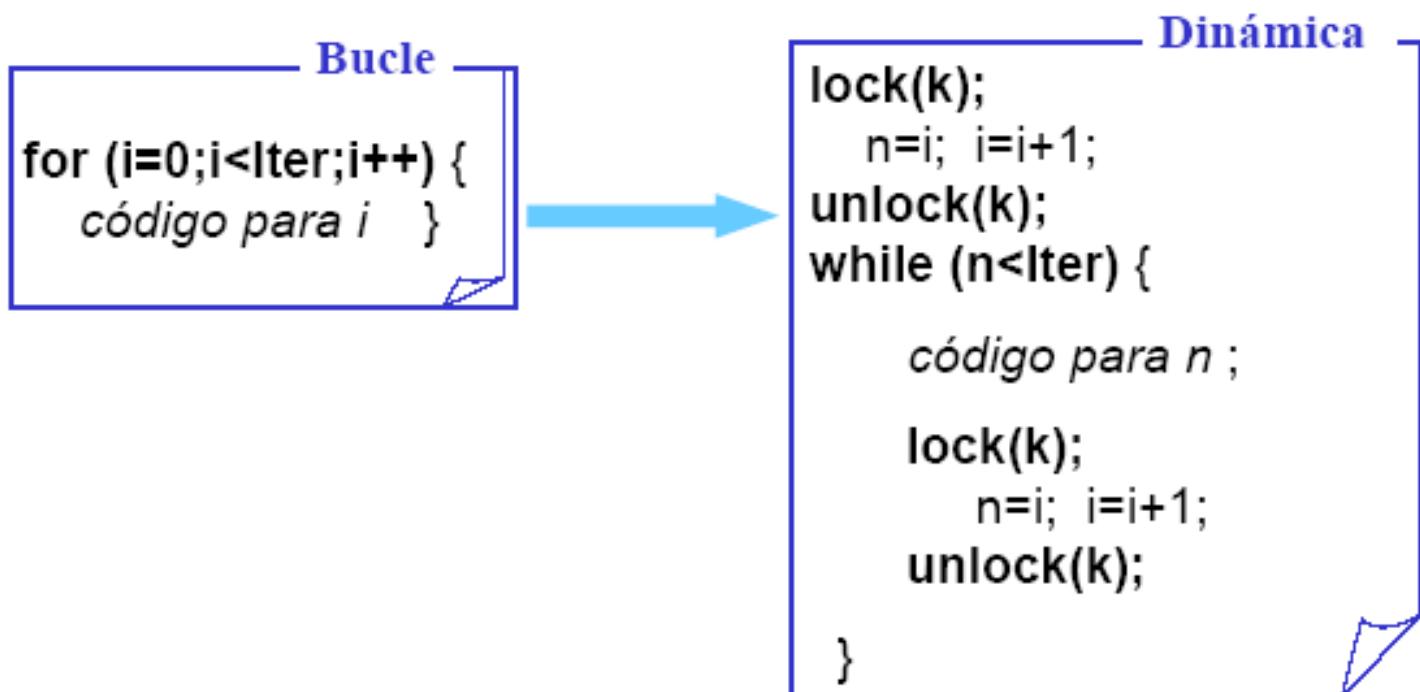
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Proceso de parallelización: asignación de tareas

- Asignación dinámica



Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Proceso de parallelización: redactar código paralelo

- Depende de:
 - Estilo de programación: paso de mensajes, etc.
 - Modo de programación
 - Situación inicial
 - Herramienta utilizada para explicitar el paralelismo
 - Estructura del programa
- Un programa paralelo debe incluir
 - Creación y destrucción de procesos/hebras
 - Asignación de carga de trabajo
 - Comunicación y sincronización

Ingeniería de los Computadores

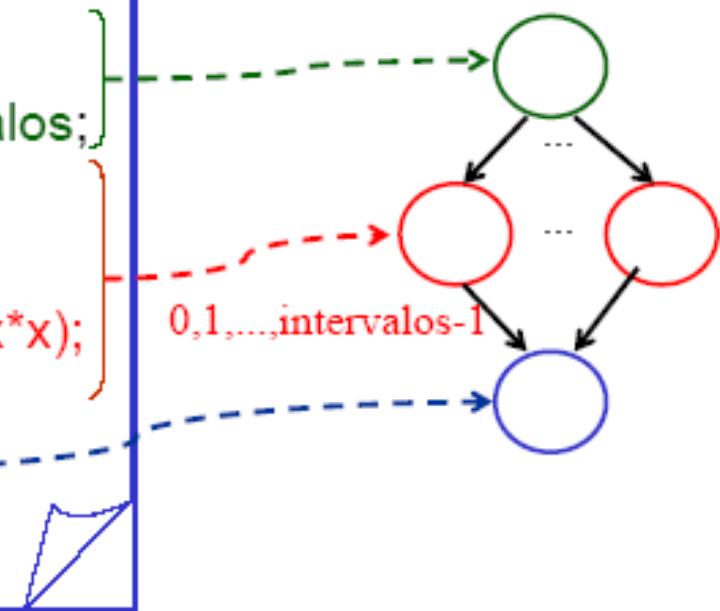
3.2 Técnicas y procedimientos

Paralelismo

Cálculo de Pi por descomposición de tareas

```
main(int argc, char **argv) {  
    double ancho, sum;  
    int intervalos, i;  
  
    intervalos = atoi(argv[1]);  
    ancho = 1.0/(double) intervalos;  
    for (i=0;i< intervalos; i++){  
        x = (i+0.5)*ancho;  
        sum = sum + 4.0/(1.0+x*x);  
    }  
    sum* = ancho;    }  
}
```

Grafo de dependencias entre tareas



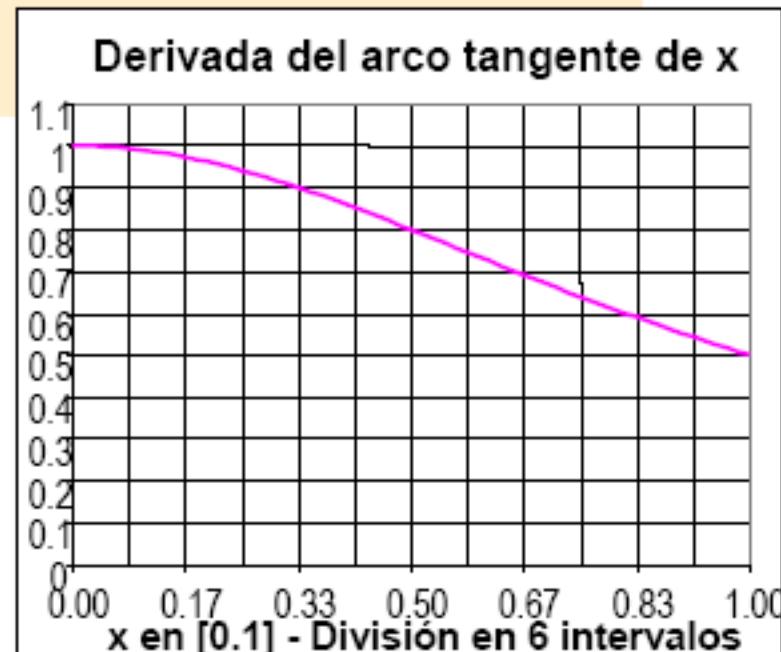
Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Cálculo de Pi por descomposición de tareas

$$\left. \begin{array}{l} \arctg'(x) = \frac{1}{1+x^2} \\ \arctg(1) = \frac{\pi}{4} \\ \arctg(0) = 0 \end{array} \right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \arctg(x) \Big|_0^1 = \frac{\pi}{4} - 0$$



- PI se puede calcular por integración numérica.

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

- Cálculo de Pi por descomposición de tareas
 - Asignación estática de iteraciones del bucle (asignación *Round Robin*)
 - Redacción de código paralelo
 - Estilo de programación: paso de mensajes
 - Modo de programación: SPMD
 - Situación inicial: versión secuencial
 - Herramienta: MPI
 - Estructura del programa: paralelismo de datos o divide y vencerás

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Parallelismo

Cálculo de Pi por descomposición de tareas

```
#include <mpi.h>
main(int argc, char **argv) {
    double ancho,x, sum, tsum; int intervalos, i; int nproc, iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc); → Enrolar
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervalos = atoi(argv[1]); ancho = 1.0 / (double) intervalos; lsum = 0;
    for (i=iproc; i<intervalos; i+=nproc) { → Asignar/
        x = (i + 0.5) * ancho; lsum += 4.0 / (1.0 + x * x); Localizar
    }
    lsum *= ancho;
    MPI_Reduce(&tsum, &sum, 1, MPI_DOUBLE, → Comunicar/
               MPI_SUM,0,MPI_COMM_WORLD); sincronizar
    MPI_Finalize(); → Desenrolar
}
```

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Cálculo de Pi por descomposición de tareas

- Asignación dinámica de iteraciones del bucle
- Redacción de código paralelo
 - Estilo de programación: directivas
 - Modo de programación: SPMD
 - Situación inicial: versión secuencial
 - Herramienta: OpenMP
 - Estructura del programa: paralelismo de datos o divide y vencerás

Ingeniería de los Computadores

3.2 Técnicas y procedimientos

Paralelismo

Cálculo de Pi por descomposición de tareas

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv) {
    double ancho,x, sum; int intervalos, i;
    intervalos = atoi(argv[1]); ancho = 1.0/(double) intervalos;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
        #pragma omp for reduction(+:sum) private(x)
            schedule(dynamic)
                for (i=0;i< intervalos; i++) {
                    x = (i+0.5)*ancho; sum = sum + 4.0/(1.0+x*x);
                }
                sum* = ancho;
}
```

Localizar →

Crear/Terminar →

Comunicar/sincronizar →

Asignar →

Ingeniería de los Computadores

3.3 Arquitecturas de altas prestaciones

Ejemplos del Top500

#1 TOP500 desde 2013 (10/2019) - Tianhe 2A (Milkiway-2)

- Fabricante: NUDT (*National University of Defense Technology*)
- #Cores: 4.981.760
- OS: Linux
- Red de interconexión: propietaria
- Máximo rendimiento: 61,444 TF
- Pico de rendimiento: 100,678 TF
- Potencia disipada: 18,482 kW
- Situación: *National Supercomputing Center* en Guangzhou
- <http://www.nscc-tj.gov.cn/en/>
- Especificación detallada: <https://www.top500.org/site/50365>

Ingeniería de los Computadores

3.3 Arquitecturas de altas prestaciones

Introducción

#5 TOP500 (6/2012) – Tianhe 1A

- #1 TOP500 (6/2012)
- Fabricante: NUDT (National University of Defense Technology)
- #cores: 186368
- OS: Linux
- Red de interconexión: propietaria
- Rendimiento máximo: 2566 TF
- Rendimiento pico: 4701 TF
- Potencia disipada: 4040 kW
- Situsción: *National Supercomputing Center* en Tianjin
- <http://www.nscc-tj.gov.cn/en/>
- Especificaciones detalladas: http://www.nscc-tj.gov.cn/en/resources/resources_1.asp

Ingeniería de los Computadores

3.3 Arquitecturas de altas prestaciones

Introducción

- Visita a la web del Top500: <https://www.top500.org>

Ingeniería de los Computadores

3.4 Evaluación de rendimiento

Prácticas

- Competencias adquiridas en las prácticas de la asignatura
- Debate: técnicas y problemas en la evaluación del rendimiento de una aplicación paralela.

Ingeniería de los Computadores

Unidad 4. Redes de interconexión.

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Introducción

Redes de interconexión en supercomputación:

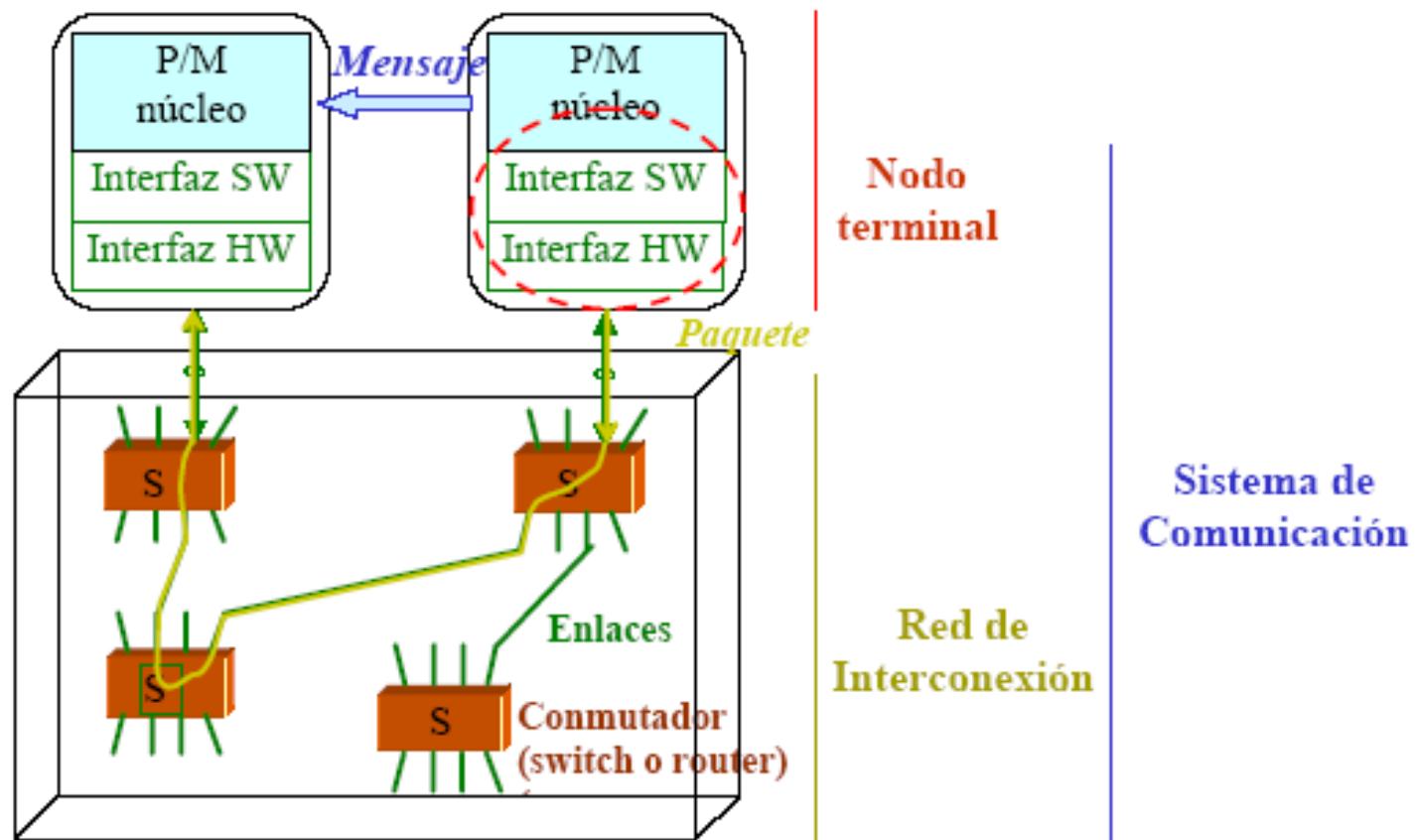
- Elemento fundamental en arquitecturas paralelas con **varios elementos de proceso que se comunican**
- La eficiencia en la comunicación es crítica: multiprocesadores, multicomputadores
- El diseño de la red condiciona: escalabilidad de la arquitectura, complejidad, tolerancia a fallos, etc.
- Aspectos relacionados: control de flujo y encaminamiento

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Introducción

Estructura general del sistema de comunicación



Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Parámetros básicos:

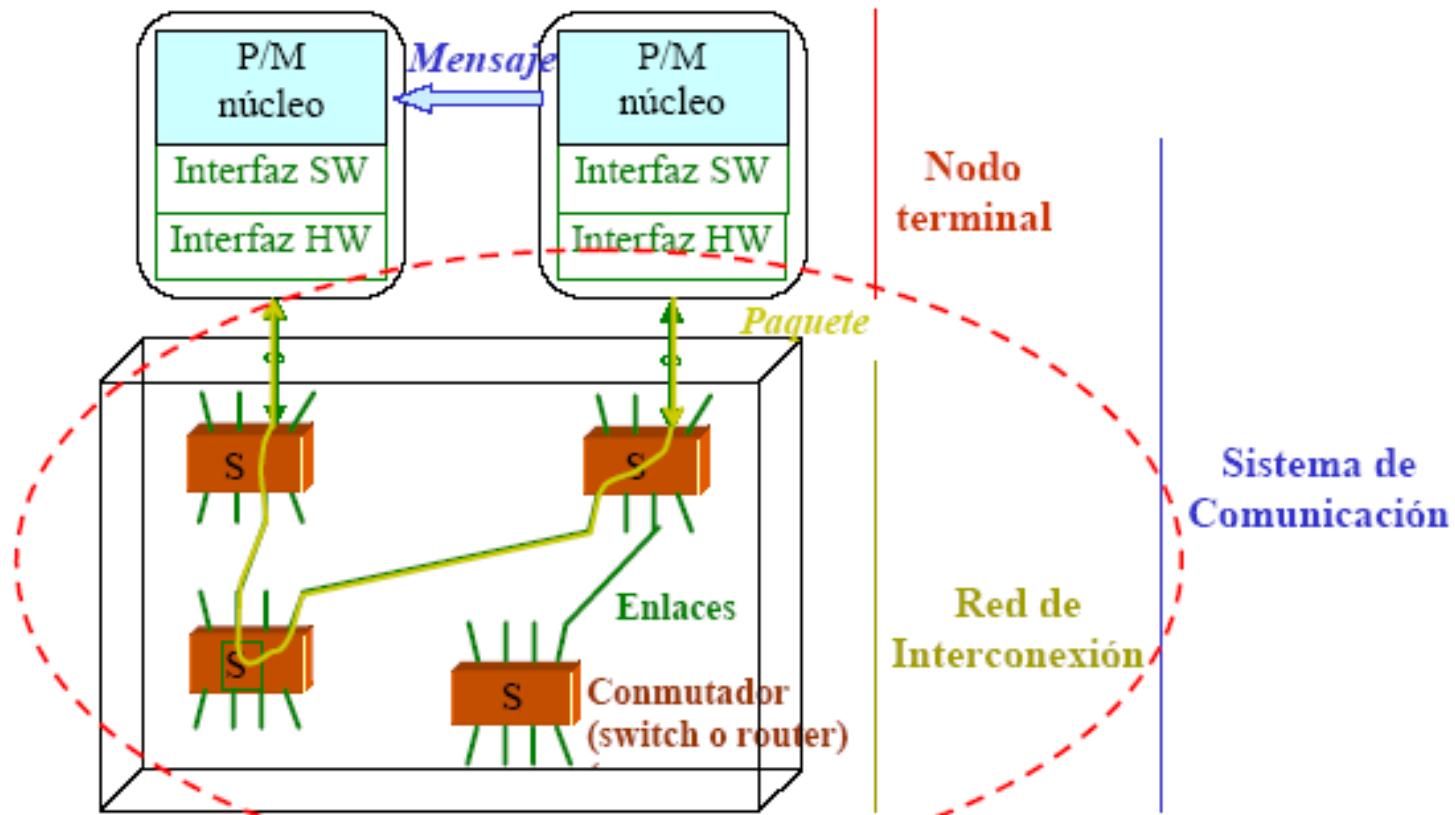
- Tamaño de la red: número de nodos (EPs, memorias, computadores)
- Grado del nodo (d - *degree*): número de canales de entrada y salida
- Nodos unidireccionales: grado de salida y grado de entrada
- Grado del nodo -> puertos de E/S (¿coste?)
- Diámetro de red: longitud máxima del camino más corto entre dos nodos cualquiera de una red.

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión



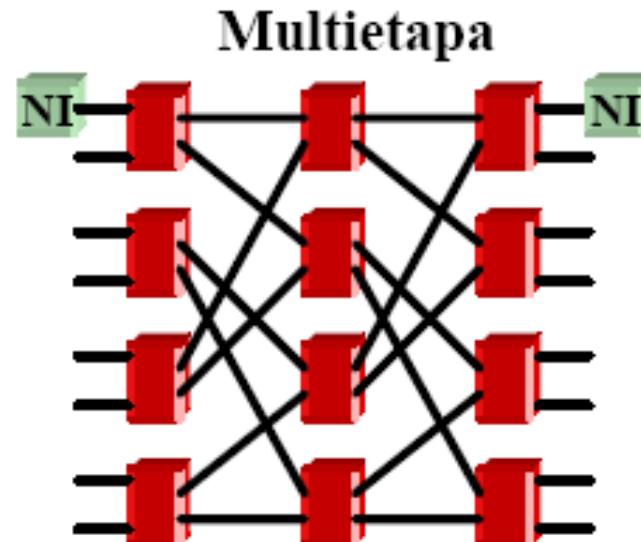
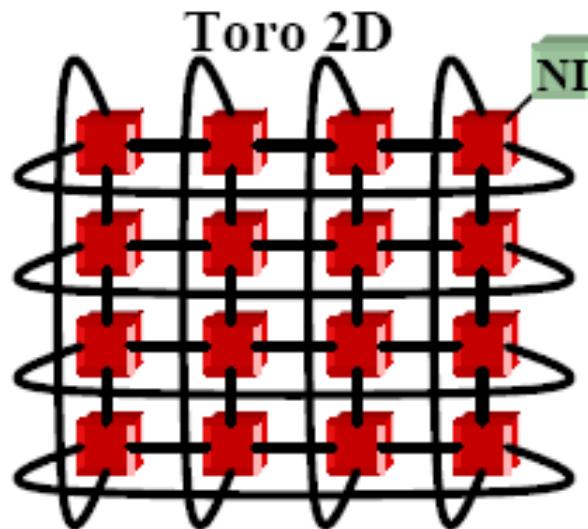
Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión: **Topología**

- **Estructura** de interconexión física de la red. Se puede modelar mediante un grafo cuyos **vértices** son **comunicadores** o **interfaces de red** (a nodos de cómputo, a módulos de memoria, o a dispositivos de E/S) y los **aristas** son los **enlaces**.



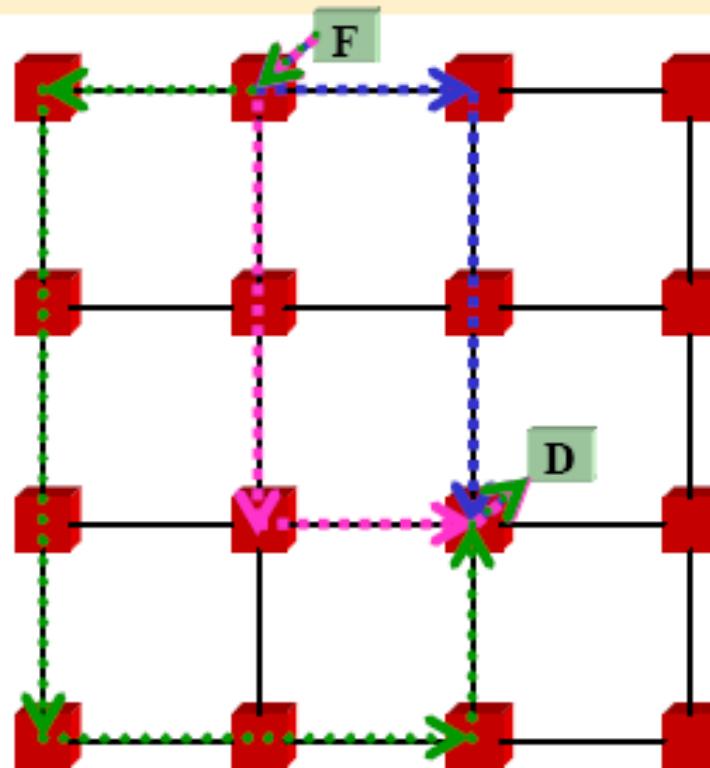
Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión: Encaminamiento

- Determina el **camino** a seguir por un paquete desde el fuente al destino.



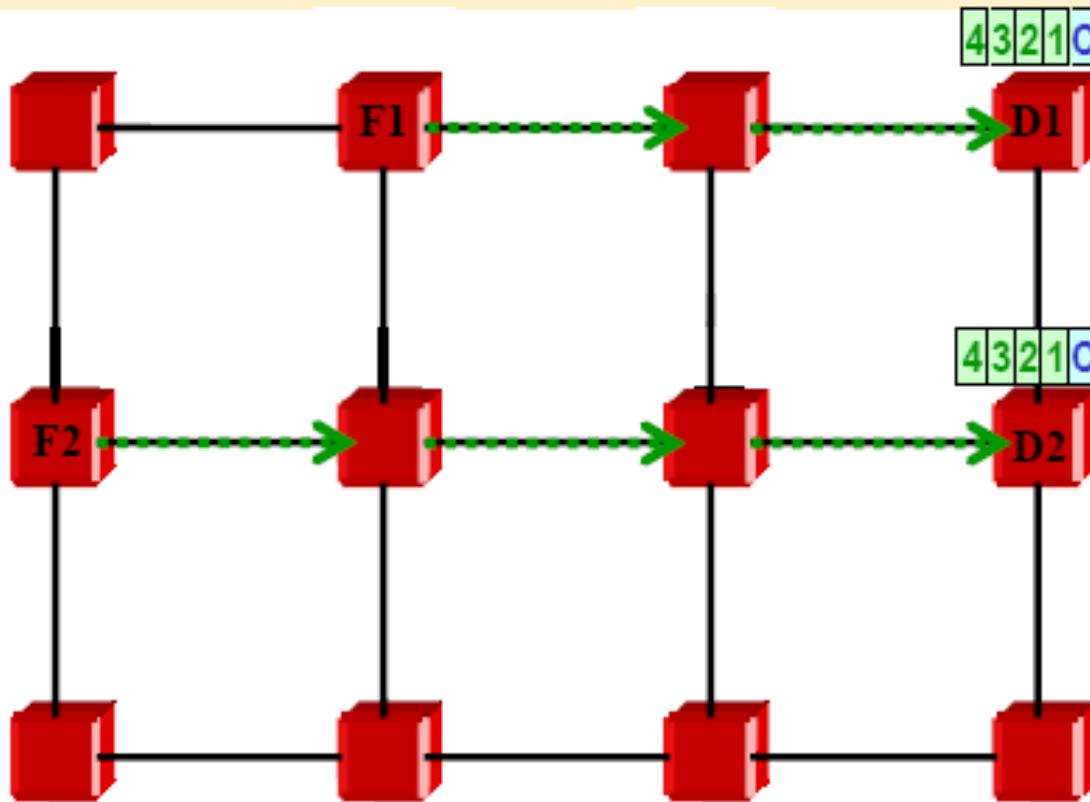
Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión: Estrategia de conmutación

- Determina **cómo** los datos en un paquete atraviesan el camino hacia el destino.



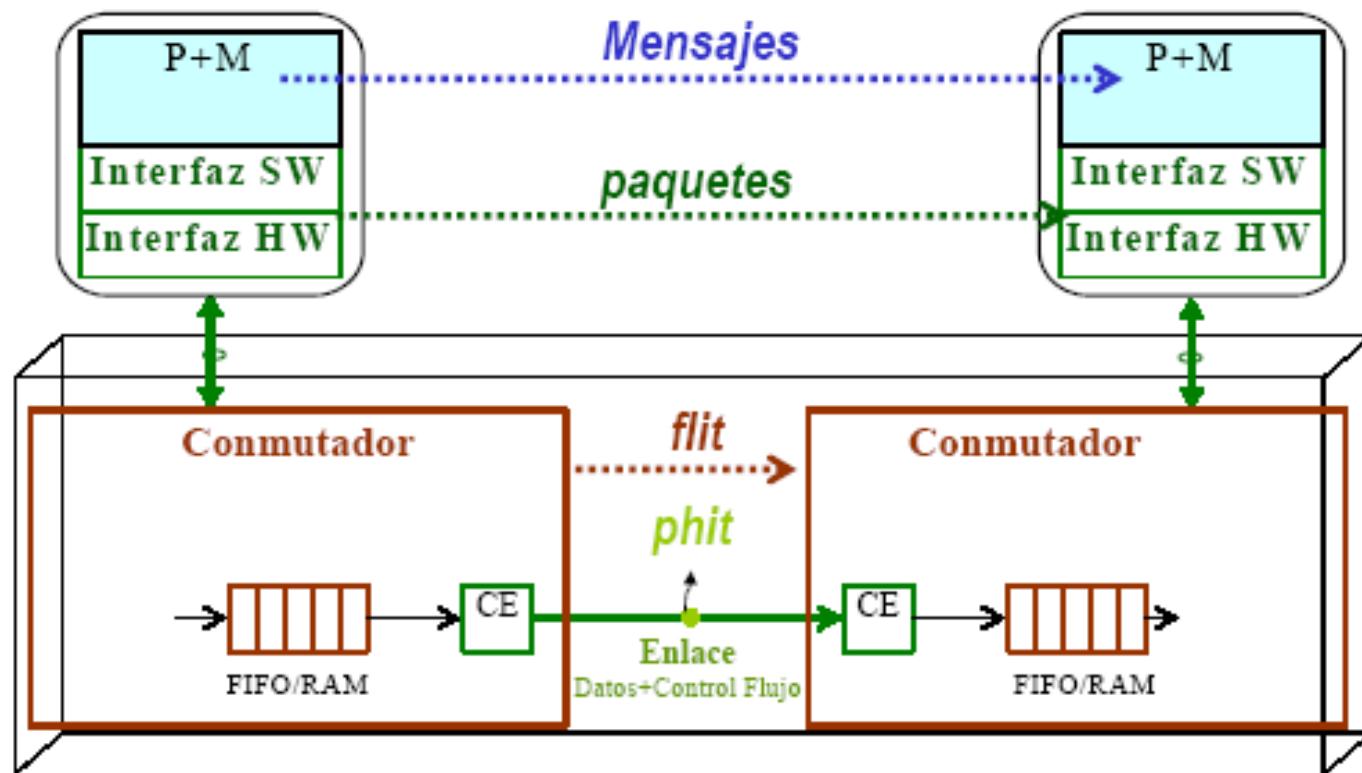
Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión: Control de flujo

Determinan **cuándo** una unida se mueven entre componentes del Sist. Comunicación, avanzando hacia el destino. **Arbitra** ante colisiones. Determina cómo y cuándo se asignan recursos (intra- e inter-conmutadores)

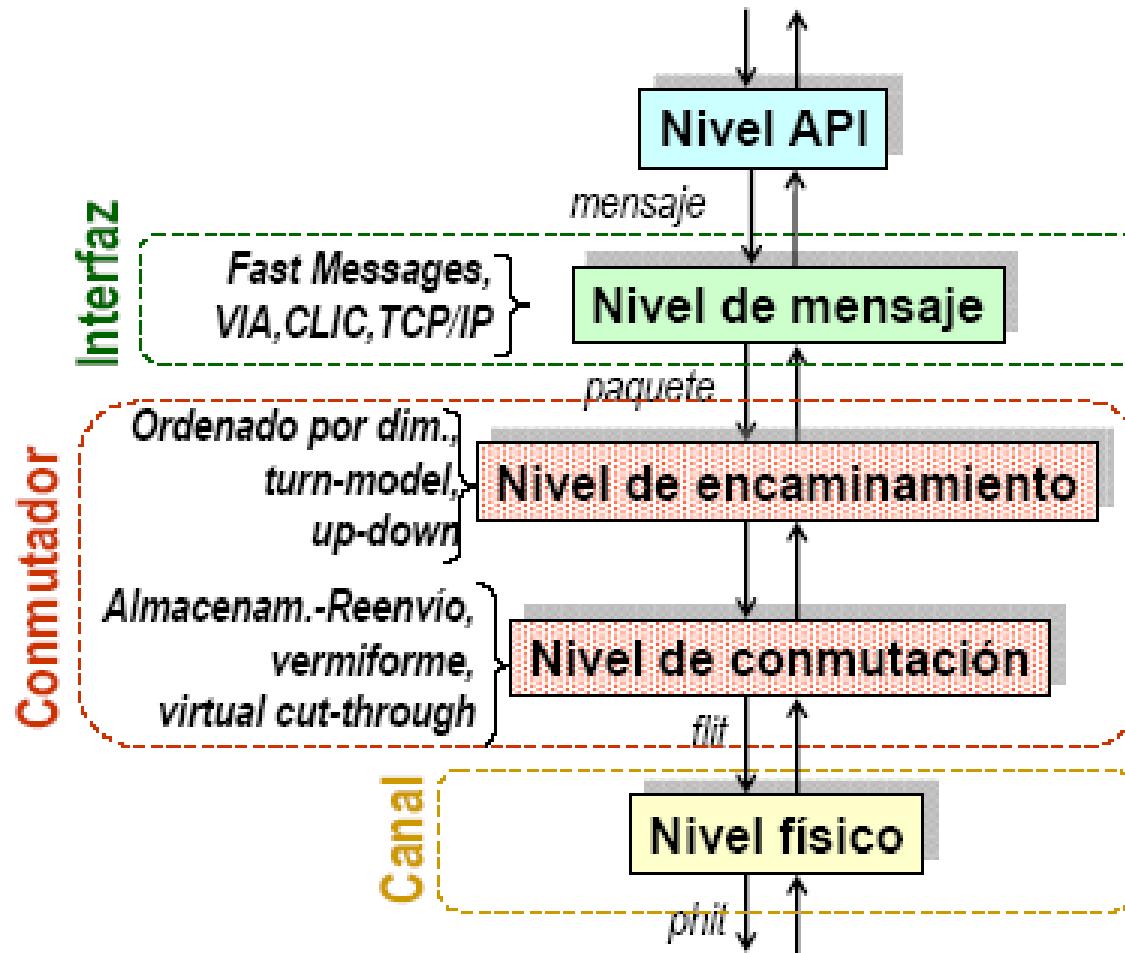


Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Conceptos

Diseño de una red de interconexión: Niveles de servicios



Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación

Clasificación de redes de interconexión

CLASES	Nº NODOS Y DISTANCIA	UTILIZACIÓN	DESARROLLO	EJEMPLOS
Diseñadas a medida	Nodos: unos pocos-decenas-cientos-miles	Multiprocesadores Multicomputadores Proc. matriciales	Arquitecturas de altas prestaciones.	-Cray X1 -Origin SGI -Sun Fire 15K
SAN: System Area Network	Nodos: decenas-cientos-miles Dist. decenas o cientos metros	Conecta comp. en habitación Interfaz software "ligera" (<i>lightweight</i>)	Redes a medida y LAN	-Estándares: SCI, Infiniband -OEM: Myrinet, QsNet
LAN: Local Area Network	Nodos: cientos Dist<decenas km	Conecta comp. en edificio o campus	Estaciones de trabajo	-Fast Eth. -Gigabit Eth.
WAN: Wide Area Network	Nodos: miles Dist. miles km	Conecta comp. a nivel mundial	Telecomunicaciones	-ATM

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación

Clasificación de redes de interconexión (1/2):

- Redes de **medio compartido**
 - ➔ Redes de área local
 - Bus de contención (Ethernet)
 - Token bus (Arcnet)
 - Token ring (IBM Token ring)
 - ➔ Bus de sistema (backplane bus) (Sun Gigaplane)
- Redes **directas** (estáticas basadas en router)
 - ➔ Topologías ortogonales (Malla, Toro, Hipercubo)
 - ➔ Otras topologías (Árbol, CCC, Estrella, ...)
- Redes **indirectas** (dinámicas basadas en conmutador)

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación

Clasificación de redes de interconexión (2/2)

- Redes **indirectas** (dinámicas basadas en conmutador)
 - ➔ Topologías regulares
 - Barras cruzadas (Crossbar)
 - Redes de interconexión multietapa (MIN)
 - Con bloqueos (unidireccionales y bidireccionales)
 - Sin bloqueos (red de Clos)
 - ➔ Topologías irregulares
- Redes híbridas (redes jerárquicas)

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación

Otra clasificación:

- Redes **directas**
 - Nodos conectados a subconjuntos de nodos
 - Escalabilidad
 - Comunicación entre los nodos mediante *routers*.
 - Canales unidireccionales o bidireccionales
- Redes **indirectas**
 - Comunicación a través de conmutadores
 - Topologías regulares (matriciales) e irregulares (NOWs)
- Redes **híbridas** (combinación de las anteriores)
- Redes **multibus**
- Redes **jerárquicas** (jerarquía de buses conectados mediante routers)
- Redes basadas en **clusters**
 - Nodos conectados (buses – fácil *broadcasting*) formando *clusters*
 - Clusters conectados entre sí (red directa - escalabilidad)

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido

- Medio de transmisión compartido
- Arbitraje (resolución de conflictos)
- Sencillo Broadcast
- Ancho de banda limitado (escalabilidad limitada) -> cuello de botella
- Bus de sistema (arquitectura UMA: Proc -> Mem)
- Redes de área local
 - ➔ Ethernet (no determinista)
 - ➔ Token bus (determinista → aplicaciones de tiempo real)
 - ➔ Token ring (estructura en anillo)

Ingeniería de los Computadores

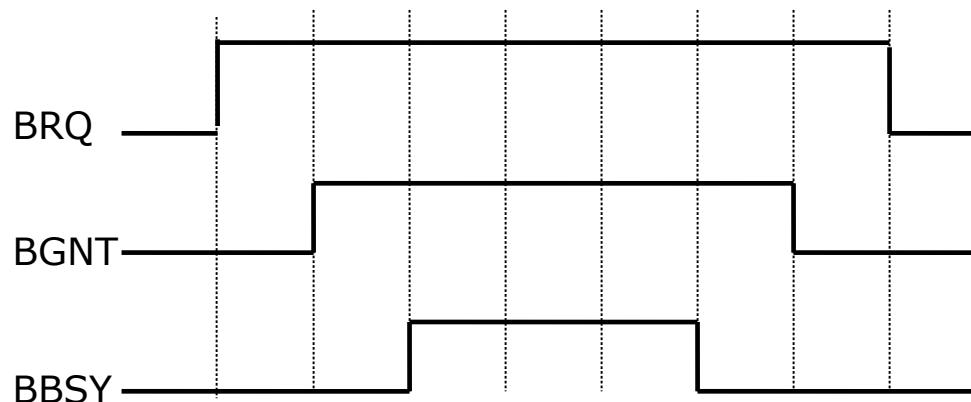
4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Prioridad estática. Señales de control:

- BRQ
- BGNT
- BBSY común



Ingeniería de los Computadores

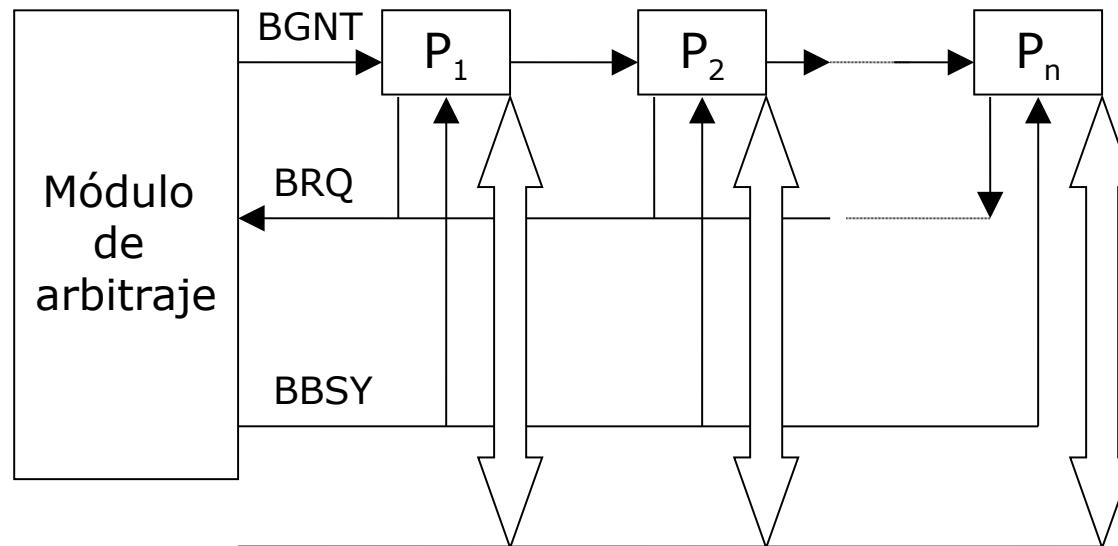
4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Prioridad estática. *Daisy Chain (centralizada-serie)*:

- BRQ común
- BGNT propagada
- BBSY común



Ingeniería de los Computadores

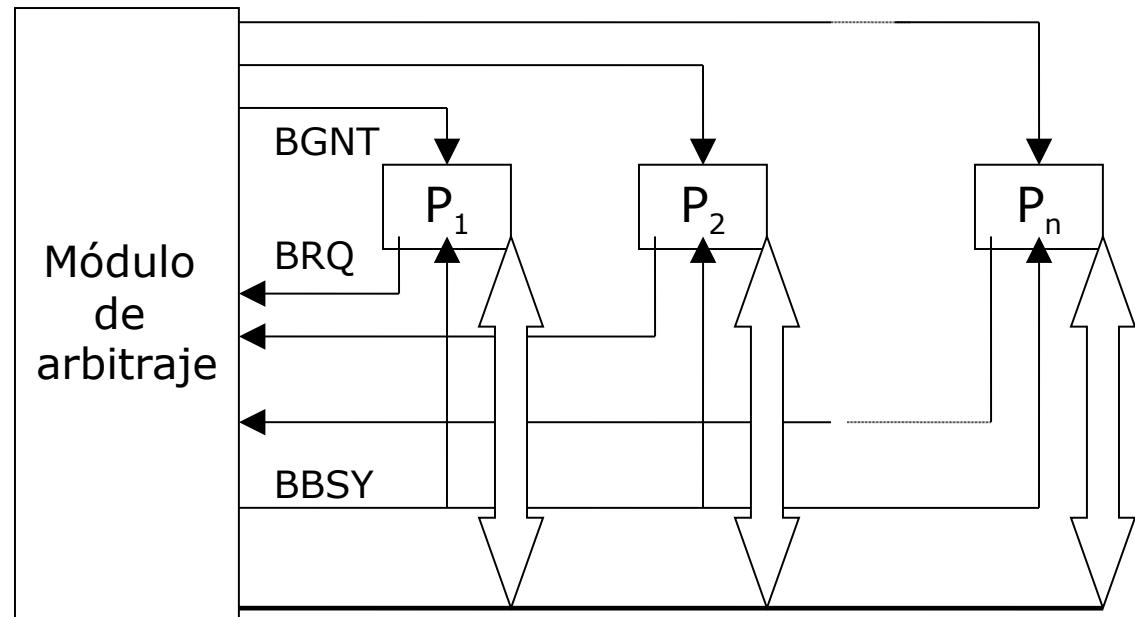
4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Prioridad estática. Codificador-decodificador de prioridad
(centralizada-paralela)

- BRQ individual
- BGNT individual
- BBSY común



Ingeniería de los Computadores

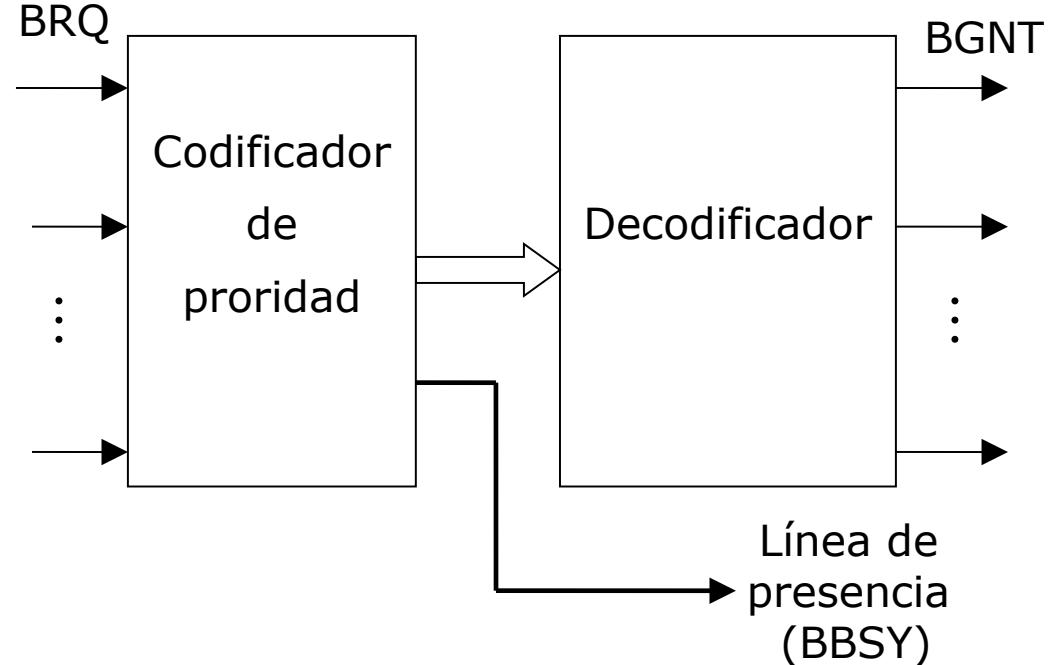
4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Prioridad estática. Codificador-decodificador de prioridad
(centralizada-paralela)

- BRQ individual
- BGNT individual
- BBSY común



Ingeniería de los Computadores

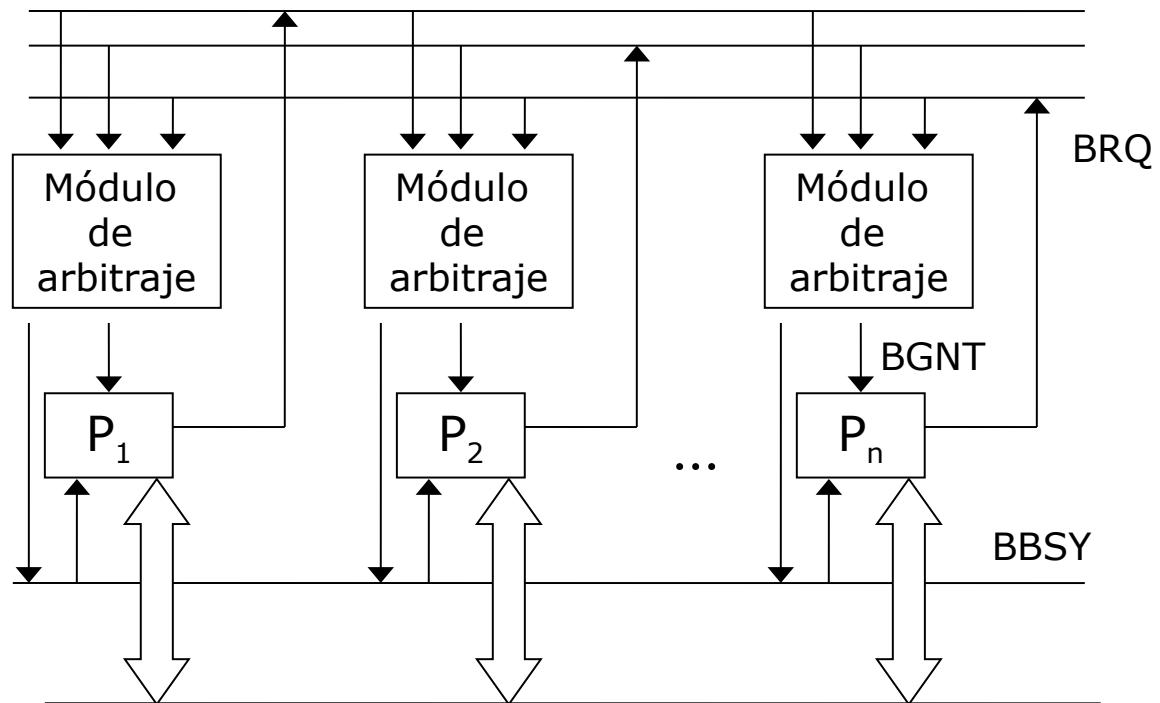
4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Prioridad estática. Autoarbitraje (distribuido-paralelo)

- BRQ individual
- BGNT individual
- BBSY común



Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

- Multiplexación temporal

Ventajas:

- Asignación equitativa
- Simplicidad

Inconvenientes:

- Infrautilización del ancho de banda

- Prioridad dinámica

- LRU
- RDC (Rotating Daisy Chain)
- FCFS

Ingeniería de los Computadores

4.1 Introducción, conceptos y clasificación

Clasificación: redes de medio compartido

Redes de medio compartido (arbitraje del bus)

Ejemplo: Prioridad dinámica (LRU)

P_0	P_1	P_2	P_3	Acción
0	1	2	3	P_0 utiliza bus
0	1	2	3	P_2 solicita bus
1	2	0	3	P_2 utiliza bus
1	2	0	3	P_1 y P_3 solicitan bus
2	3	1	0	P_3 utiliza bus

¿Qué afecta al rendimiento?

Parámetros básicos

- Anchura de la bisección (B): mínimo número de canales que, al cortar, separa la red en dos partes iguales
Nota: El número de cables que cruzan la bisección es una cota inferior de la densidad de cableado
- Longitud del cable: efectos sobre la latencia
- Simetría: Una red es simétrica si es isomorfa a ella misma independientemente del nodo considerado origen
- Rendimiento
 - ➔ Funcionalidad: Indica cómo la red soporta el encaminamiento de datos, tratamiento de las interrupciones y sincronización.
 - ➔ Latencia: Indica el retraso de un mensaje

¿Qué afecta al rendimiento?

Parámetros básicos:

- Rendimiento
 - Ancho de banda. Velocidad máxima de transmisión de datos
 - Complejidad hardware. Coste de implementación (cables, conmutadores, conectores, etc.)
 - Escalabilidad. Capacidad de la red para expandirse de forma modular
 - Capacidad de transmisión. Número total de datos que se pueden transmitir a través de la red en una unidad de tiempo. (Punto caliente)

¿Qué afecta al rendimiento?

Diseño de una red de interconexión

- Topología -> grafo de interconexión
- Control de flujo -> método usado para regular el tráfico en la red
 - ➔ Mensaje
 - ➔ Paquete
 - ➔ Flit (FLow control unIT)
- Encaminamiento -> método usado por un mensaje para elegir un camino entre los canales de la red
 - ➔ Determinista
 - ➔ Adaptativo

Topologías: Redes estáticas o directas

Redes estáticas o directas

Clasificación:

- Estrictamente ortogonales (malla, hipercubo, toro)
 - (**Estrictamente**) Cada nodo tiene al menos un enlace en cada dimensión.
 - (**Ortogonal**) Cada enlace supone un desplazamiento en **una dimensión**.
- No ortogonales (árbol)
- Propiedades
 - Grado (número de enlaces con otros nodos)
 - Diámetro (máximo camino más corto entre dos nodos)
 - Regularidad (todos los nodos tienen el mismo grado)
 - Simetría (se ve semejante desde cualquier nodo)

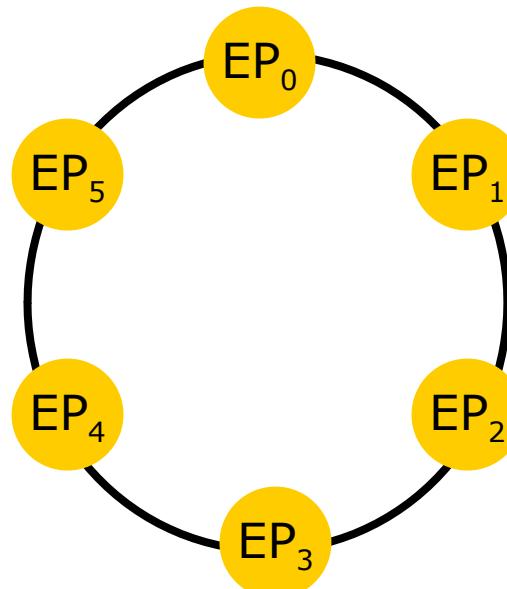
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Anillo unidireccional

- Función de interconexión: $F_{+1}(i) = (i+1) \bmod N$
- Grado de entrada/salida: 1/1
- Diámetro: $N-1$



- ¿Anillo bidireccional?

Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Malla abierta

- F. interconexión:

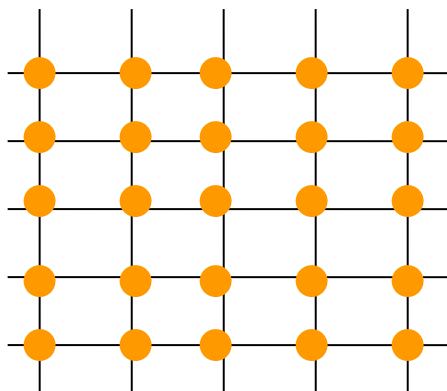
$$F_{+1}(i) = (i+1) \text{ si } i \bmod r <> r-1$$

$$F_{-1}(i) = (i-1) \text{ si } i \bmod r <> 0$$

$$F_{+r}(i) = (i+r) \text{ si } i \bmod r <> r-1$$

$$F_{-r}(i) = (i-r) \text{ si } i \bmod r <> 0$$

- Grado: 4
- Diámetro: $2(r-1)$, donde $N=r^2$



Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Malla Illiac

- F. interconexión:

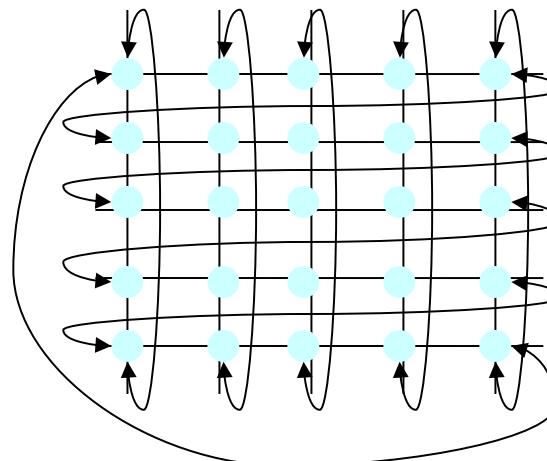
$$F_{+1}(i) = (i+1) \bmod N$$

$$F_{-1}(i) = (i-1) \bmod N$$

$$F_{+r}(i) = (i+r) \bmod N$$

$$F_{-r}(i) = (i-r) \bmod N$$

- Grado: 4
- Diámetro: $(r-1)$, donde $N=r^2$



Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Redes n-cubos k-arias ó toros

- n dimensiones, k nodos
- Función de interconexión toro 2D:

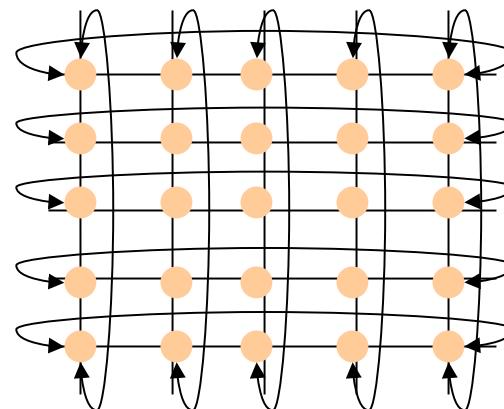
$$F_{+1}(i) = (i+1) \bmod r + (i \text{ DIV } r) \cdot r$$

$$F_{-1}(i) = (i-1) \bmod r + (i \text{ DIV } r) \cdot r$$

$$F_{+r}(i) = (i+r) \bmod N$$

$$F_{-r}(i) = (i-r) \bmod N$$

- Grado: 4
- Diámetro: $2 \cdot \left\lfloor \frac{r}{2} \right\rfloor$, donde $N=r^2$



Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Desplazador barril

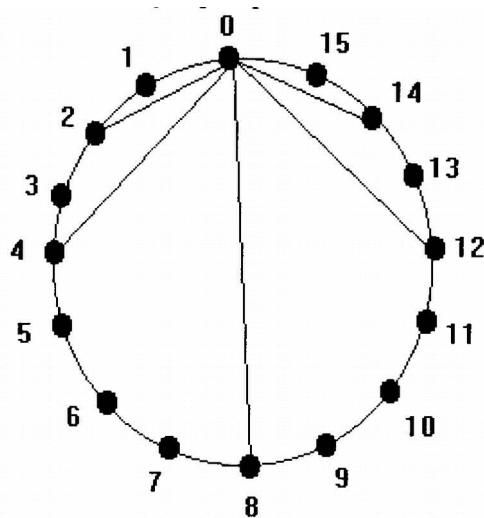
- F. interconexión:

$$B_{\pm k}(i) = (i+2^k) \bmod N$$

$$B_{-k}(i) = (i - 2^k) \bmod N$$

$K=0 \dots n-1$, $n = \log N$, $i = 0 \dots N-1$

- Grado: $2n - 1$
 - Diámetro: $n/2$



Ingeniería de los Computadores

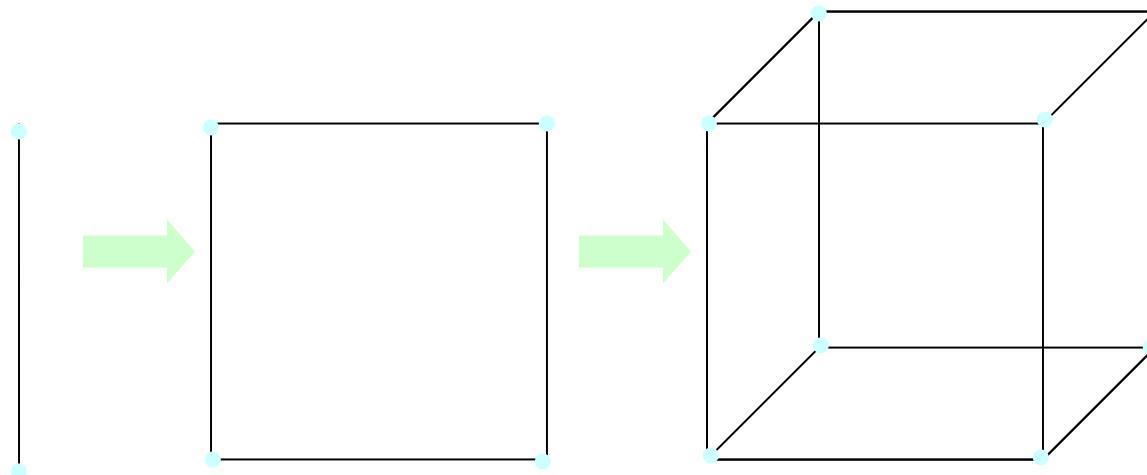
Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

- Redes estáticas o directas. Hipercubo

➤ F. interconexión:

- $F_i(h_{n-1}, \dots, h_i, \dots h_0) = h_{n-1}, \dots, \bar{h}_i, \dots h_0$
- Grado: n ($n=\log N$)
- Diámetro: n

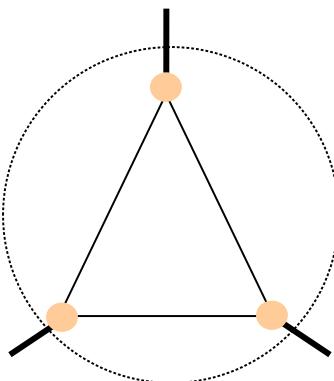


Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ciclo cubo conectado (CCC) (red jerárquica)

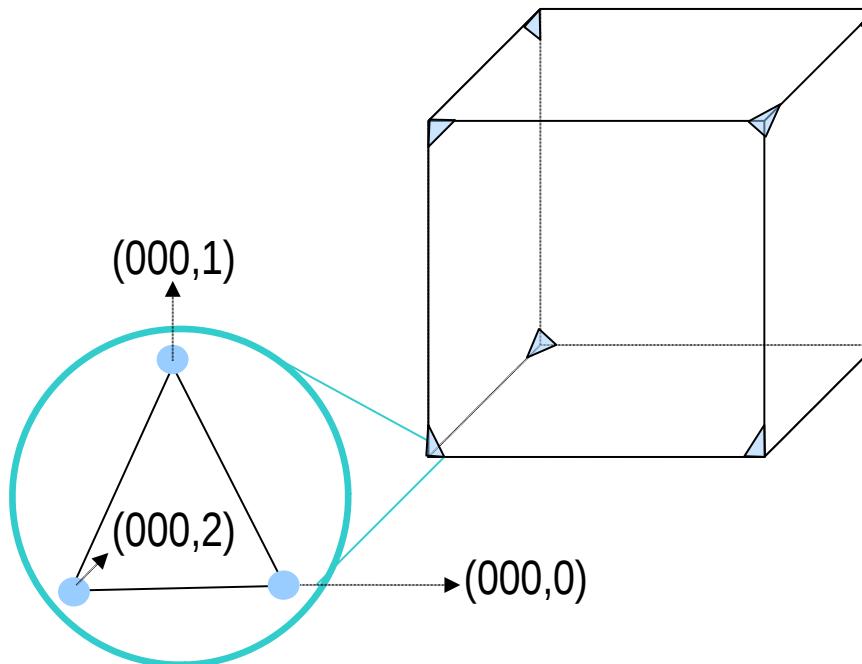


Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Red CCC (Ciclo-Cubo-Conectada)



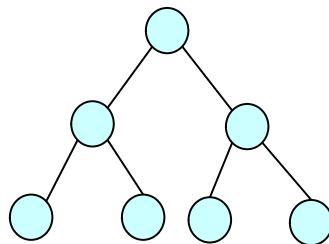
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes estáticas o directas

Redes estáticas o directas. Ej. Árbol binario

- Balanceado: todas las ramas del árbol tienen la misma longitud
- Cuello de botella → nodo raíz
- N (balanceado) = $2^k - 1$ (k = niveles del árbol)
- Grado: 3
- Diámetro: $2(k-1)$



Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas

- Uso de conmutadores y árbitros
- Ejemplos
 - Redes crossbar
 - Redes de conexión multietapa (MIN)
- Modelo: $G(N,C)$
 - N, conjunto de conmutadores
 - C, enlaces (unidireccionales o bidireccionales) entre conmutadores
 - Canal bidireccional → dos canales unidireccionales
 - Un conmutador puede tener conectados 0, 1 o más elementos (Procesadores, memorias, etc.)
- Distancia entre dos nodos: distancia entre los conmutadores que conectan los nodos más 2.

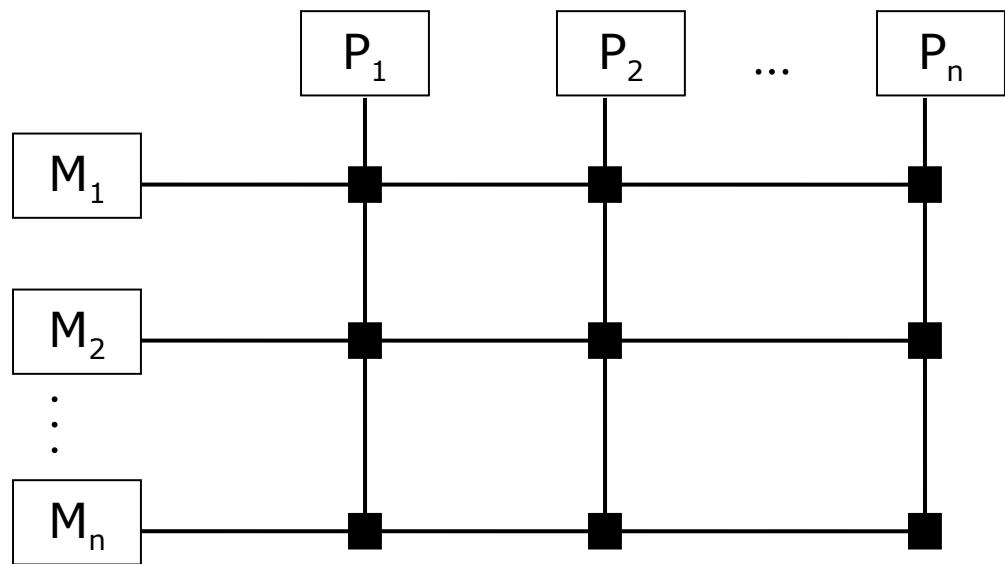
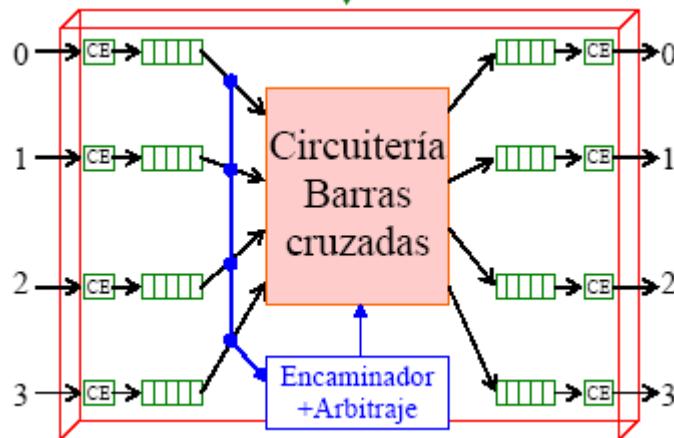
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Ej. Redes *crossbar*

- Conexión directa nodo-nodo
- Gran ancho de banda y capacidad de interconexión
- Conexión Proc. – Mem. → limitado por los accesos a memoria (columnas)
- Conexión Proc(N) – Proc(N) → máximo de N conexiones
- Coste elevado: $O(N \cdot M)$



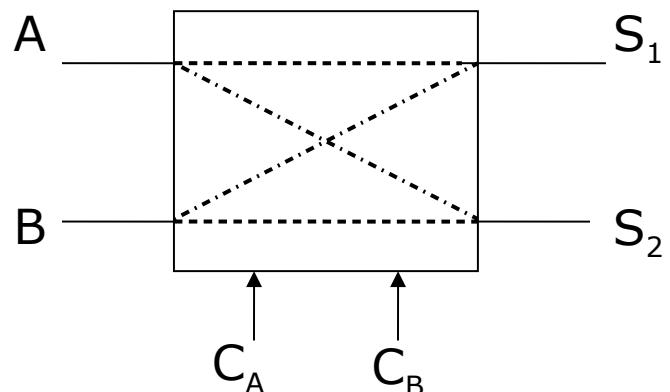
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Ej. Redes MIN

- Conectan dispositivos de entrada con dispositivos de salida mediante un conjunto de etapas de conmutadores, donde cada conmutador es una red *crossbar*.
- Concentradores → n^o entradas > n^o salidas
- Expansores → n^o salidas > n^o entradas



Ingeniería de los Computadores

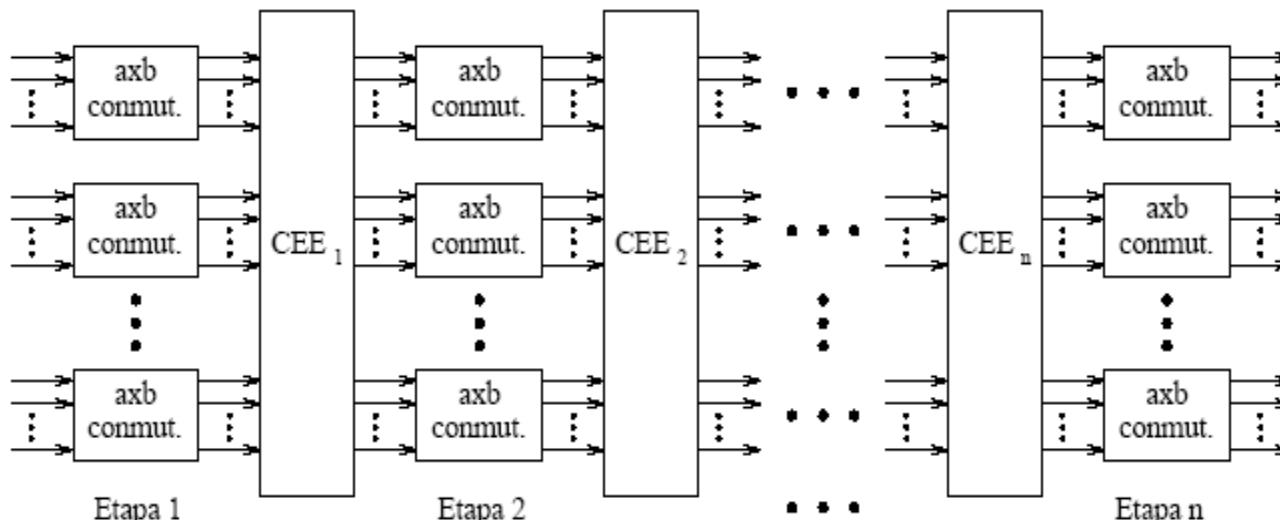
Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Redes MIN

- Conexión de etapas adyacentes → Patrón de conexión
- Patrón basado en permutaciones: conmutadores con el mismo número de entradas y salidas.
- Ejemplo: barajado perfecto.

$$B(a_{n-1}, a_{n-2}, \dots, a_0) = (a_{n-2}, \dots, a_0, a_{n-1})$$



Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Redes MIN

- Número de entradas a^n y número de salidas b^n (red $a^n \times b^n$)
 - ✓ n etapas de conmutadores (C_0, C_1, \dots, C_{n-1})
 - ✓ Conmutadores $a \times b$
 - ✓ $a^{n-1-i} \times b^i$ conmutadores en la etapa C_i
- Funcionalidad de los conmutadores: barras cruzadas, reducción, difusión
- Subred de interconexión entre etapas: R_0, R_1, \dots
- Tipos de canales: unidireccionales, bidireccionales

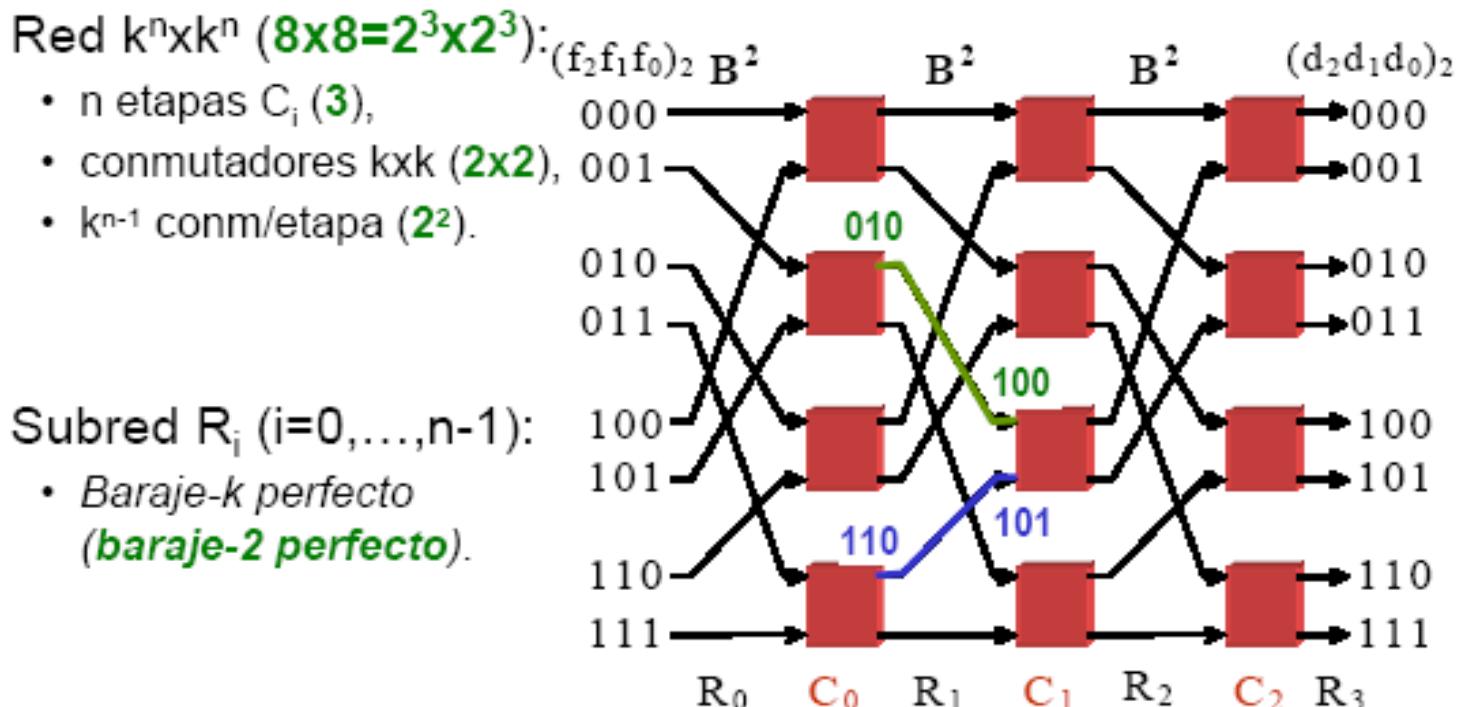
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Redes MIN - red Omega

El patrón de conexión C_i es una permutación k -baraje perfecto a excepción del último (R_n) que es permutación 0



$$B^k ((f_{n-1}, f_{n-2}, \dots, f_1, f_0)_k) = (f_{n-2}, \dots, f_1, f_0, f_{n-1})_k$$

$$B^2 ((f_2, f_1, f_0)_2) = (f_1, f_0, f_2)_2$$

Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

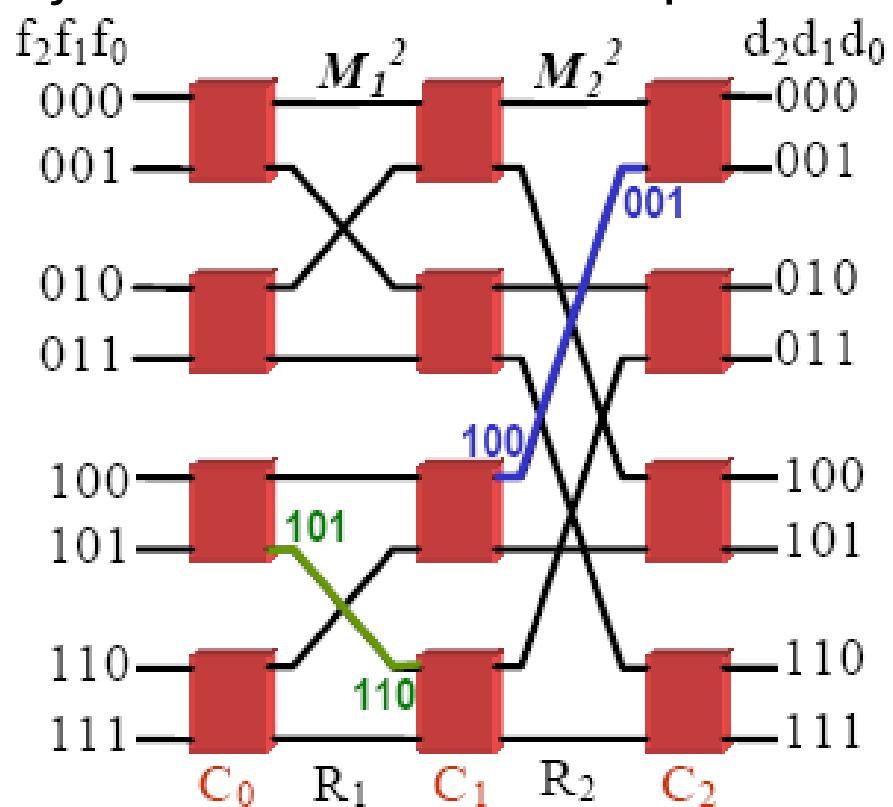
Redes indirectas o dinámicas. Ej. Redes MIN - red mariposa

– Red $k^n \times k^n$ ($8 \times 8 = 2^3 \times 2^3$):

- n etapas C_i (3),
- commutadores $k \times k$ (2×2),
- k^{n-1} comm/etapa (2^2).

– Subred R_i ($i=0, \dots, n-1$):

- Mariposa M_i^k



$$M_i^k ((f_{n-1}, f_{n-2}, \dots, f_{i+1}, \underline{f_i}, f_{i-1}, \dots, f_1, \underline{f_0})_k) = (f_{n-1}, f_{n-2}, \dots, f_{i+1}, \underline{f_0}, f_{i-1}, \dots, f_1, \underline{f_i})_k$$

$$i=0, \dots, n-1$$

$$M_2^2 ((\underline{f_2}, f_1, \underline{f_0})_2) = (\underline{f_0}, f_1, \underline{f_2})_2$$

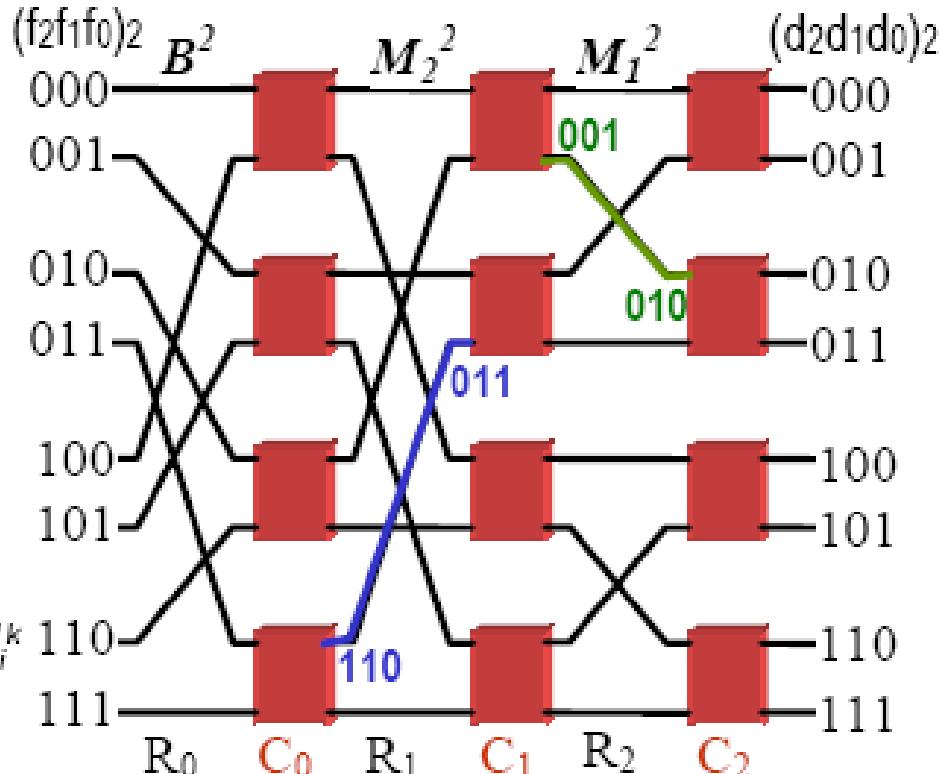
Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Ej. Redes MIN - red cubo

- Red $k^n \times k^n$ ($8 \times 8 = 2^3 \times 2^3$):
 - n etapas C_i (3),
 - commutadores $k \times k$ (2x2),
 - k^{n-1} comm/etapa (2^2).
- Subred R_i ($i=0, \dots, n-1$):
 - R_0 : Baraje-k perfecto (**baraje-2 perfecto**).
 - R_{n-i} ($i=1, \dots, n-1$): Mariposa M_i^k



$$M_i^k ((f_{n-1}, f_{n-2}, \dots, f_{i+1}, \textcolor{blue}{f}_i, f_{i-1}, \dots, f_1, \textcolor{blue}{f}_0)_k) = (f_{n-1}, f_{n-2}, \dots, f_{i+1}, \textcolor{blue}{f}_0, f_{i-1}, \dots, f_1, \textcolor{blue}{f}_i)_k$$

$$i=0, \dots, n-1$$

$$M_1^2 ((f_2, \textcolor{blue}{f}_1, \textcolor{blue}{f}_0)_2) = (f_2, \textcolor{blue}{f}_0, f_1)_2$$

Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Redes indirectas o dinámicas. Ej. Redes MIN - red delta

Red $a^n \times b^n$ ($16 \times 9 = 4^2 \times 3^2$):

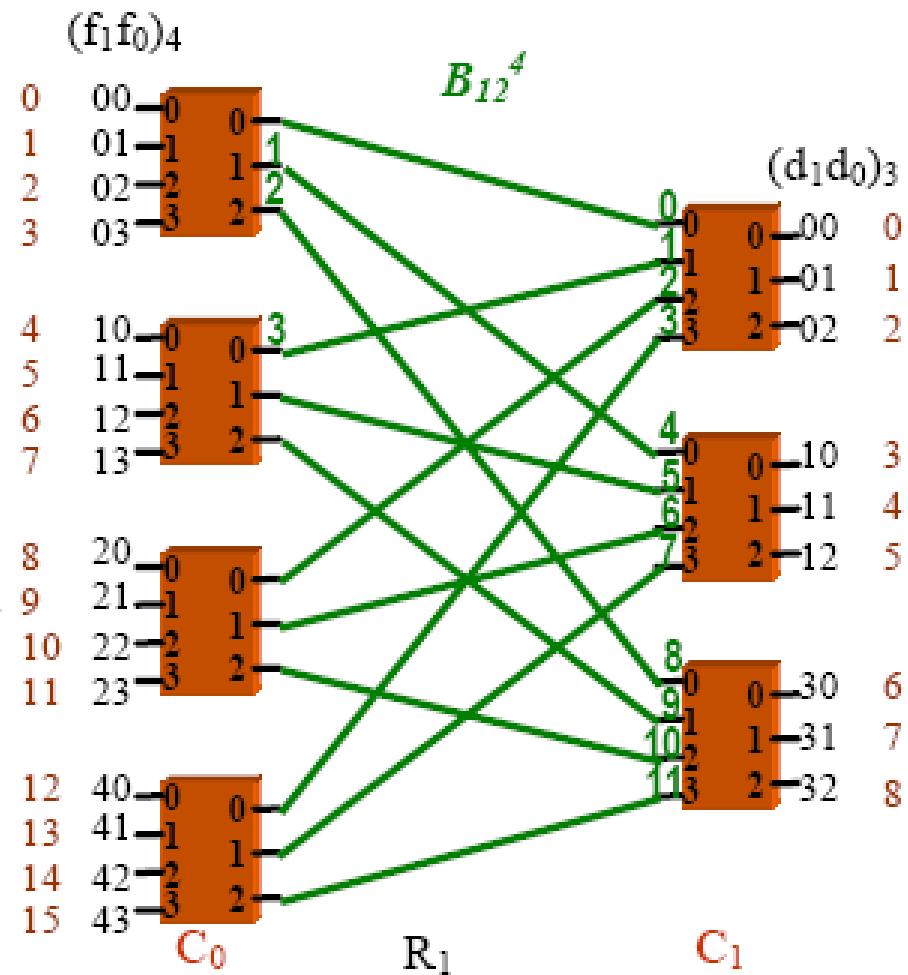
- n etapas C_i (2),
- conmutadores $a \times b$ (4×3),
- $a^{n-1-i} \cdot b^i$ conn / C_i (4, 3).

Subred R_i ($i=0 \text{ o } 1, \dots, n-1$):

- Baraje-a de c elementos
- R_1 : (**baraje-4 de 12 elementos**)

$$B_c^a(s) = \begin{cases} a \cdot s \bmod (c-1) & \text{si } s < c-1 \\ c-1 & \text{si } s = c-1 \end{cases}$$

$$B_{12}^4(s) = \begin{cases} 4 \cdot s \bmod (11) & \text{si } s < 11 \\ 11 & \text{si } s = 11 \end{cases}$$

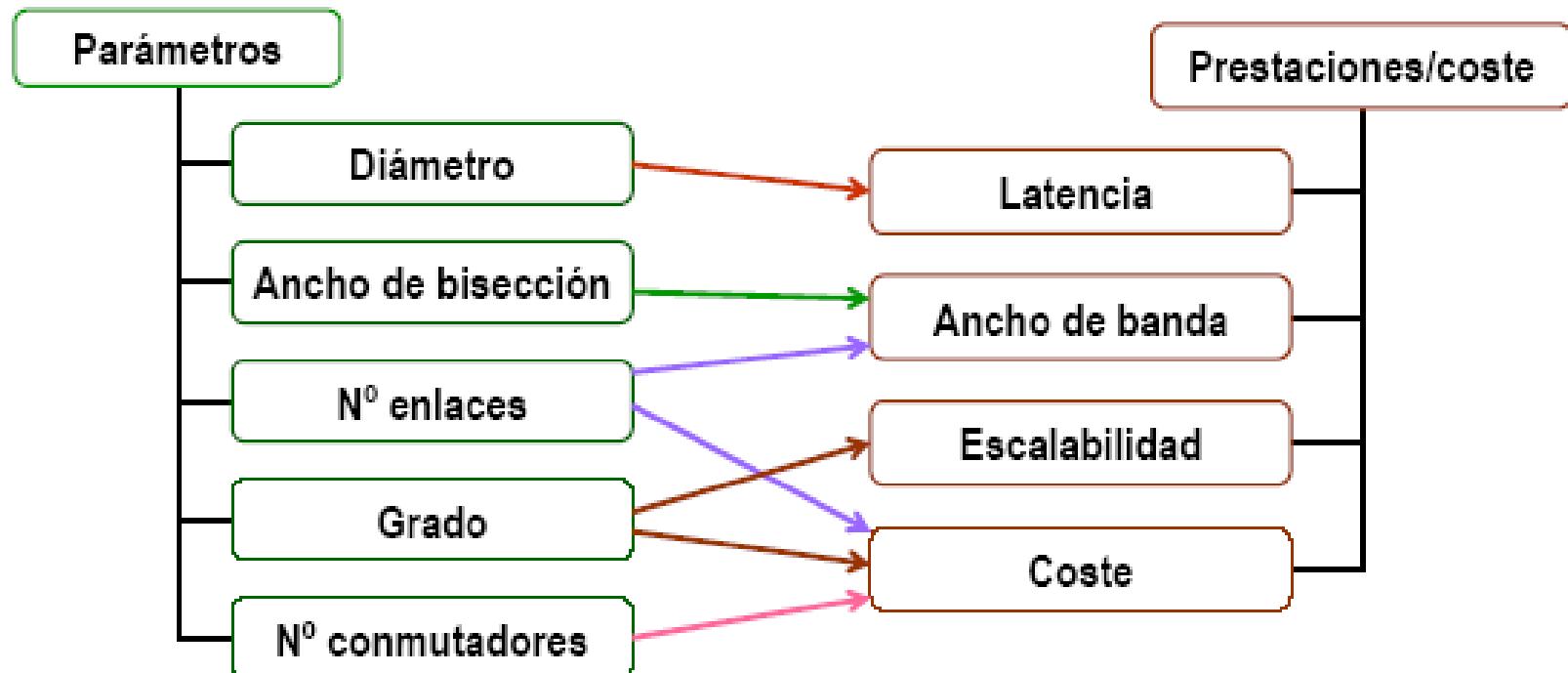


Ingeniería de los Computadores

Unidad 4.3 Topologías.

Topologías: Redes indirectas o dinámicas

Prestaciones:

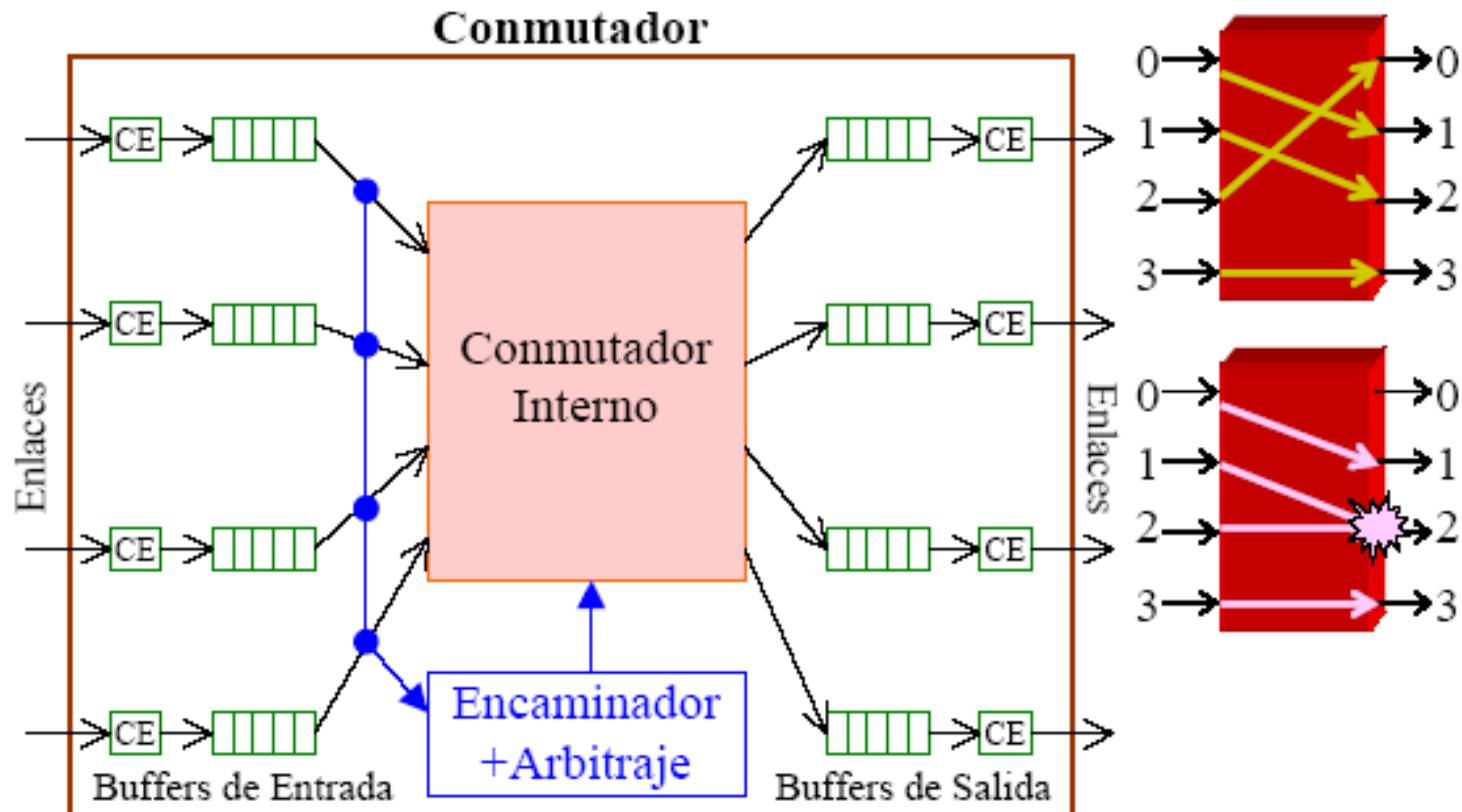


Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

El conmutador

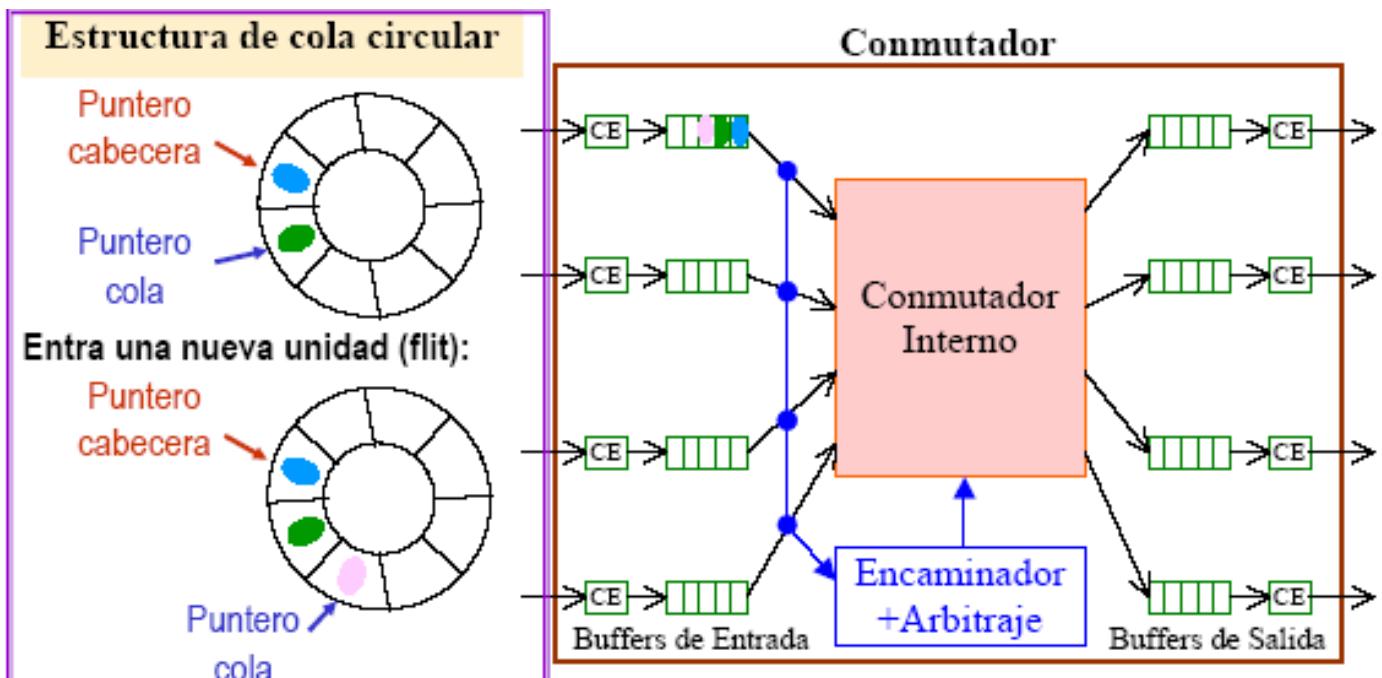


Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

Buffers de entrada



Otra alternativa es una estructura de datos de lista enlazada

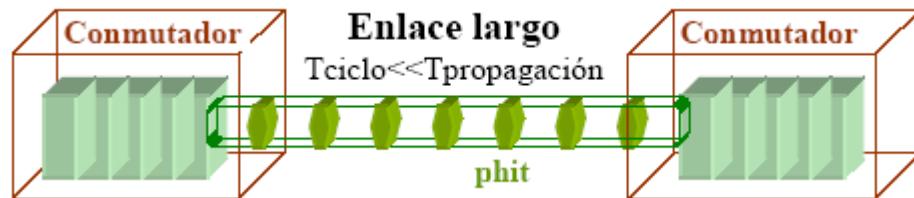
Coste de gestión mayor. Además de actualizar punteros de cabecera y cola hay que modificar punteros entre celdas y gestionar una lista con celdas libres.

Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

- Enlaces y canales.
 - ✓ Infraestructura: hilos eléctricos (cobre), fibras ópticas, etc.
- Anchura
 - ✓ Anchos: Si se transmite simultáneamente datos y control
 - ✓ Estrechos: Cuando se multiplexa en el tiempo datos y control
- Longitud
 - ✓ Cortos: 1 símbolo
 - ✓ Largos: Varios símbolos de forma simultánea



Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

- Enlaces: longitud
 - Cortos: El ciclo de red depende del retardo de propagación
 - Largos: Ciclo de red << retardo de propagación
- Velocidad del canal depende:
 - Energía empleada para transmitir por una línea
 - Distancia a atravesar
 - Ruido
 - Desplazamiento entre líneas de un enlace
 - Tamaño del *buffer* destino (enlaces largos)

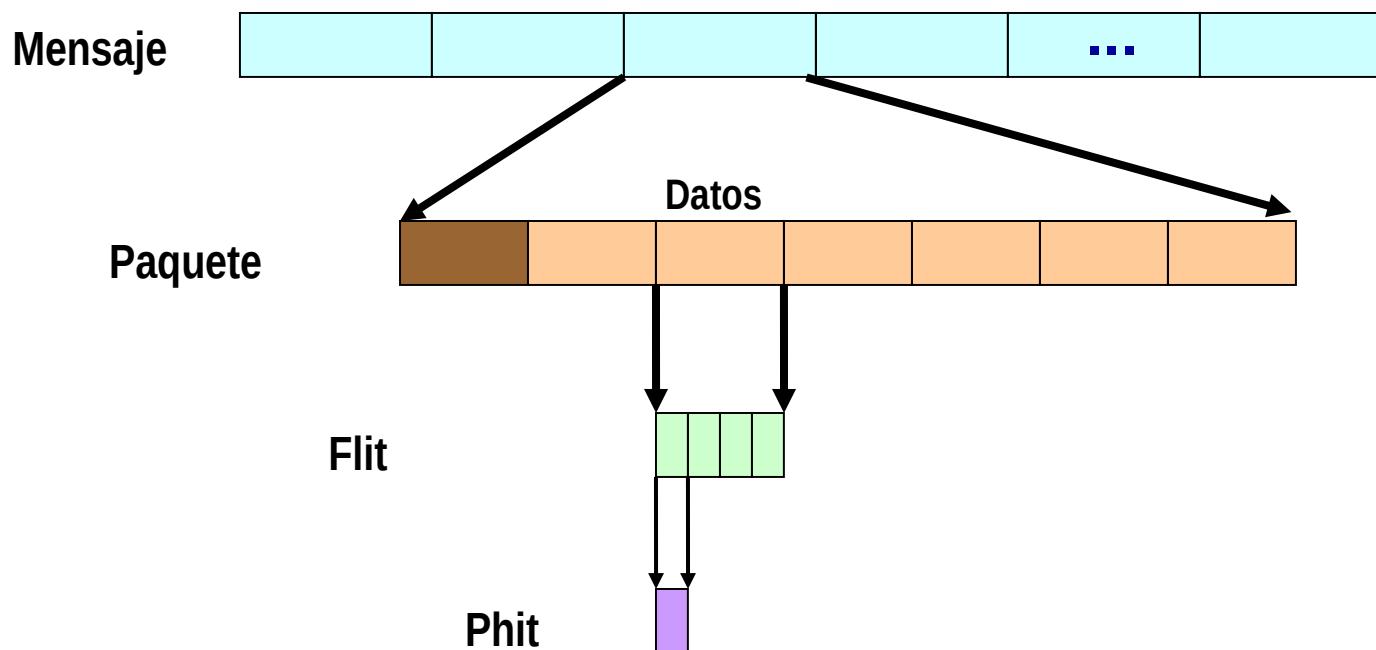
Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

Técnicas de conmutación:

- Cuándo y cómo se conectan entradas y salidas de routers
- Cuándo se transfiere el mensaje por los caminos



Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

- Tipos de técnicas de conmutación
 - Almacenamiento y reenvío (S&F, Store and Forward)
 - Vermiforme (Wormhole)
 - *Virtual Cut-Through* (VCT)
 - Conmutación de circuitos (CC, *Circuit Switching*) (Origen en redes telefónicas)
 - Canales virtuales
- Comparación entre técnicas
 - Comparación cuantitativa: latencia de transporte
 - Comparación cualitativa: ancho de banda global

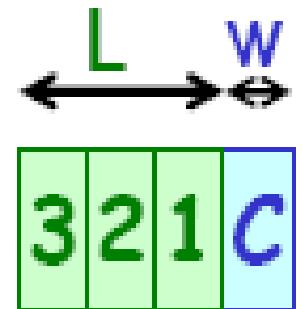
Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

Se considera (a efectos de explicación teórica siguientes transparencias):

- 1 phit = 1 flit = W bits
- Cabecera = 1 flit
- Tamaño total del paquete = **L** bits + **W** bits (cabecera)
- Distancia fuente-destino = **D** parejas comutador-en
- Comutadores con *buffer* independiente para cada entrada y salidas sin *buffer*
- T_w = tiempo para que un phit atraviese una etapa comutador/enlace
- T_r = tiempo de encaminamiento (*routing*)



Store & forward (almacenamiento y reenvío)

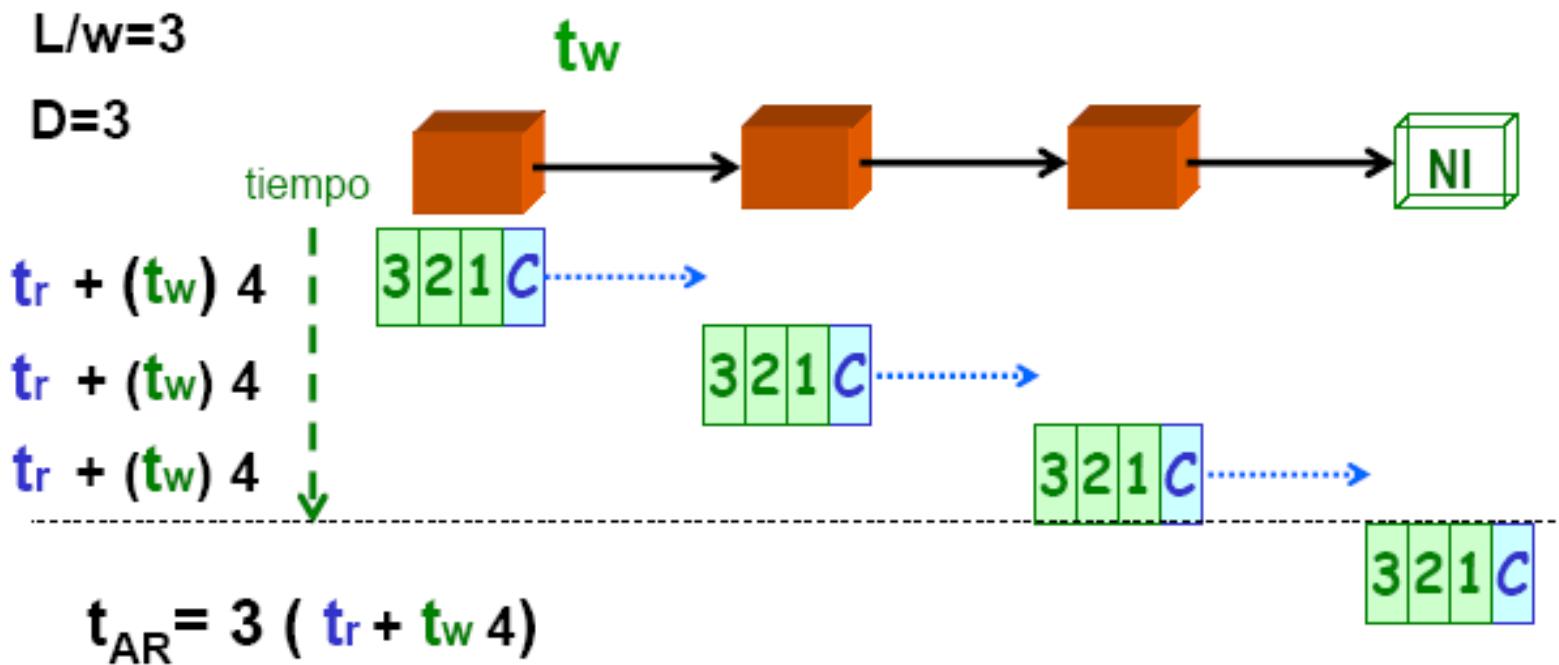
- El conmutador almacena el paquete completo antes de ejecutar el algoritmo de encaminamiento y reenviar
- La unidad de transferencia (**paquete**) entre interfaces ocupa sólo un canal en cada instante
- Almacenamiento en conmutadores: múltiples de un paquete (mínimo 1 paquete)
- Ancho de banda: El número de enlaces ociosos influye en este parámetro: para un tamaño de *buffer* mínimo (1 paquete), un paquete bloqueado deja ocioso un canal

Técnicas de conmutación

Store & forward (almacenamiento y reenvío)

- Latencia de transporte:

$$t_{AR} = D \cdot \left[t_r + t_w \cdot \left(\left\lceil \frac{L}{W} \right\rceil + 1 \right) \right]$$



Wormhole

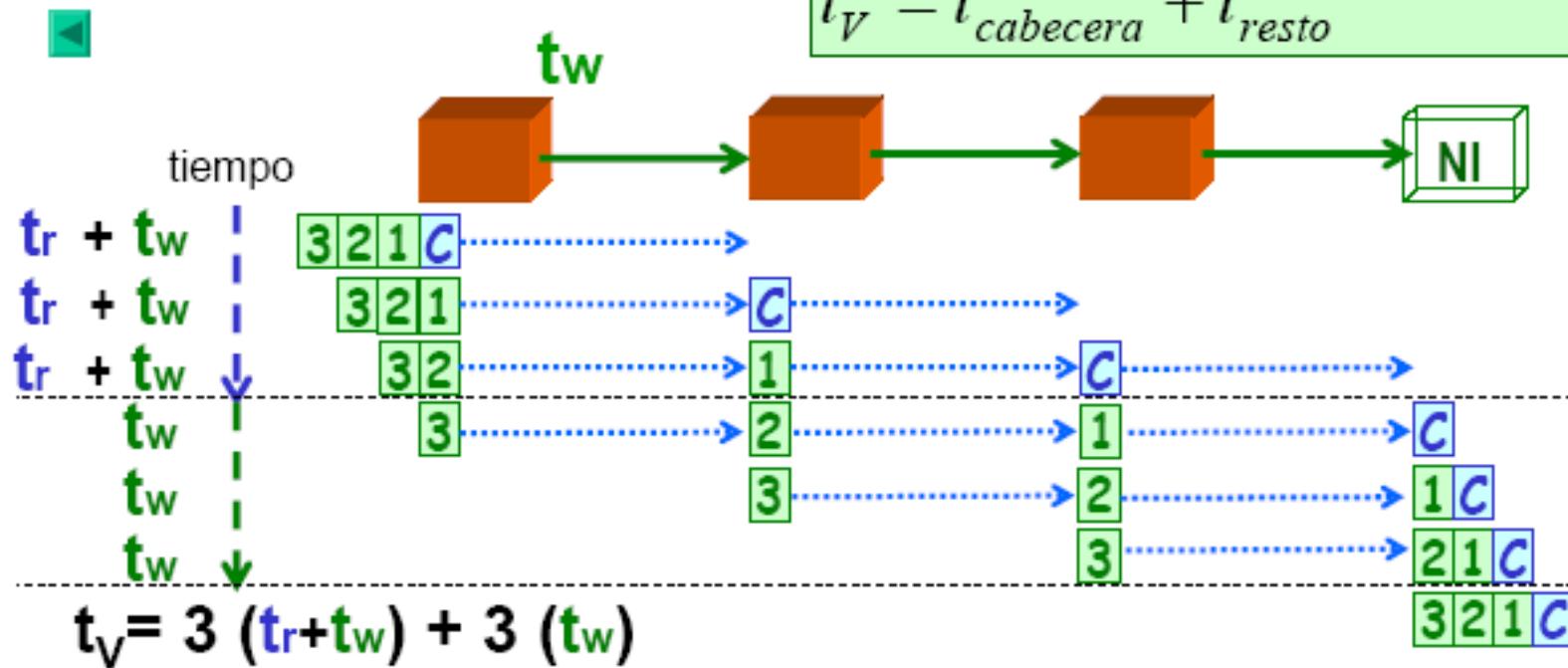
- En cuanto llega la cabecera al conmutador se ejecuta el algoritmo de encaminamiento y se reenvía
- La unidad de transferencia es el **mensaje**
- La transferencia se hace a través de un camino segmentado (n^o etapas depende del n^o de buffer). La unidad de transferencia puede ocupar varios canales en un instante
- Almacenamiento en conmutadores: múltiples de un flit (mínimo 1 flit)
- Ancho de banda: El número de enlaces ociosos influye en este parámetro: para un tamaño de *buffer* mínimo (1 flit), un paquete bloqueado deja ociosos varios canales

Técnicas de conmutación

Wormhole

- Latencia de transporte
(buffer sólo en entradas de conmutadores)

$$t_V = D \cdot (t_r + t_w) + t_w \cdot \left\lceil \frac{L}{W} \right\rceil$$
$$t_V = t_{cabecera} + t_{resto}$$



Ingeniería de los Computadores

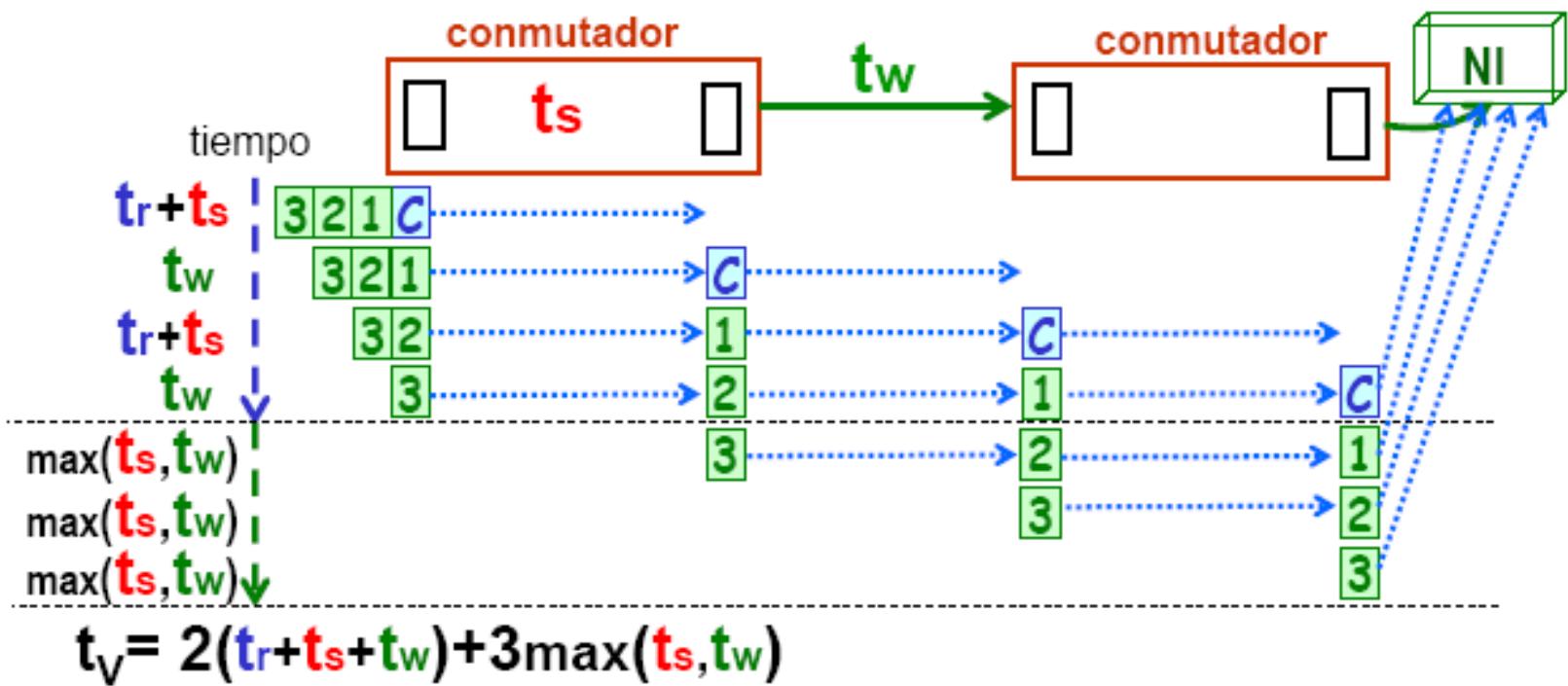
Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

Wormhole

- Latencia de transporte:
(buffer en entradas y salidas)

$$t_V = D \cdot (t_r + t_s + t_w) + \max(t_s, t_w) \cdot \left\lceil \frac{L}{W} \right\rceil$$
$$t_V = t_{cabecera} + t_{resto}$$



Virtual Cut-Through

- En cuanto llega la cabecera al conmutador se ejecuta el algoritmo de encaminamiento y se reenvía
- La unidad de transferencia es **el paquete**
- La transferencia se hace a través de un camino segmentado (n^o etapas depende del n^o de *buffer*). La unidad de transferencia puede ocupar varios canales en un instante
- Almacenamiento en conmutadores: múltiples de un paquete (mínimo 1 paquete)
- Prestaciones:
 - Latencia = **Wormhole**
 - Ancho de banda = **Store & Forward**

Conmutación de circuitos

- Desde el fuente se envía una sonda (flit) que reserva el camino. El destino devuelve una señal de reconocimiento y el fuente comienza la transmisión
- La unidad de transferencia es el **mensaje**
- La transferencia se hace a través del canal entre fuente y destino (o un camino segmentado) reservado por la sonda
- Almacenamiento en conmutadores: los *buffers* almacenan la sonda
- Ancho de banda : Cuando la sonda queda bloqueada deja ociosos múltiples canales (tantos como la distancia del punto de bloqueo al fuente)

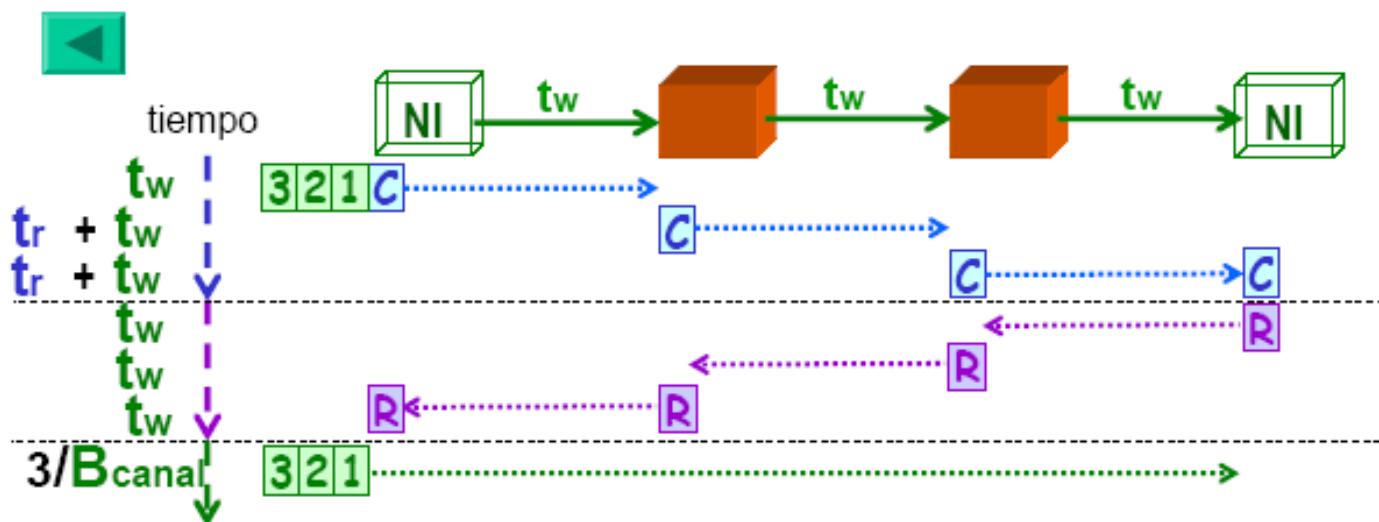
Técnicas de conmutación

Conmutación de circuitos

- Latencia de transporte (si se establece 1 canal):

$$t_{CC} = [t_w + D \cdot (t_r + t_w)] + [D \cdot (t_w) + t_w] + [1/B_{canal} \cdot \lceil L/W \rceil]$$

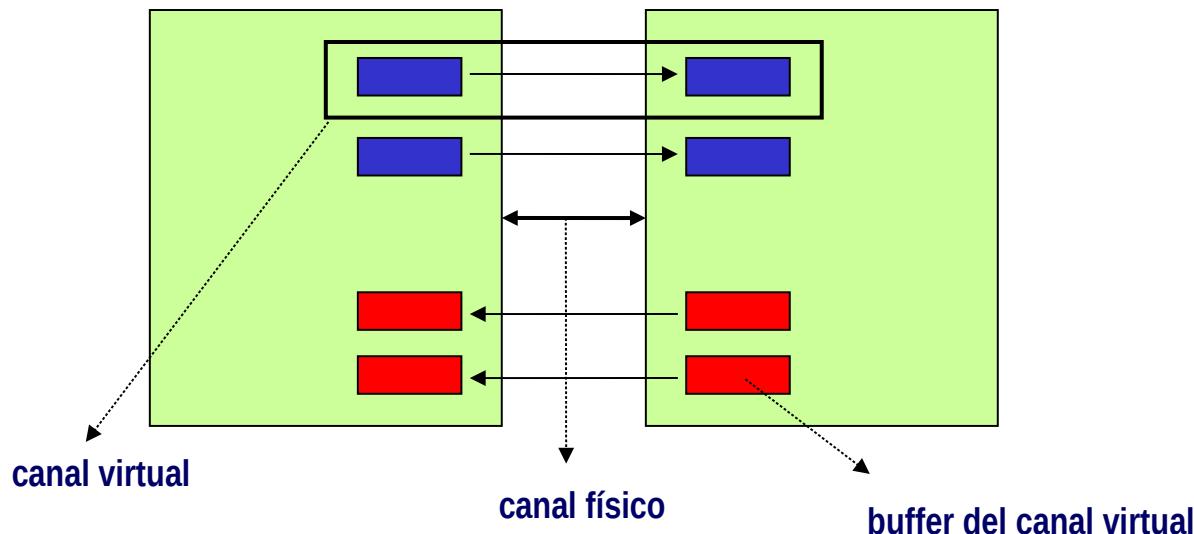
$$t_{CC} = t_{sonda} + t_{reconocimiento} + t_{datos}$$



$$t_w = (t_w + 2(t_r + t_w)) + (2(t_w) + t_w) + 3/B_{canal}$$

Canales virtuales

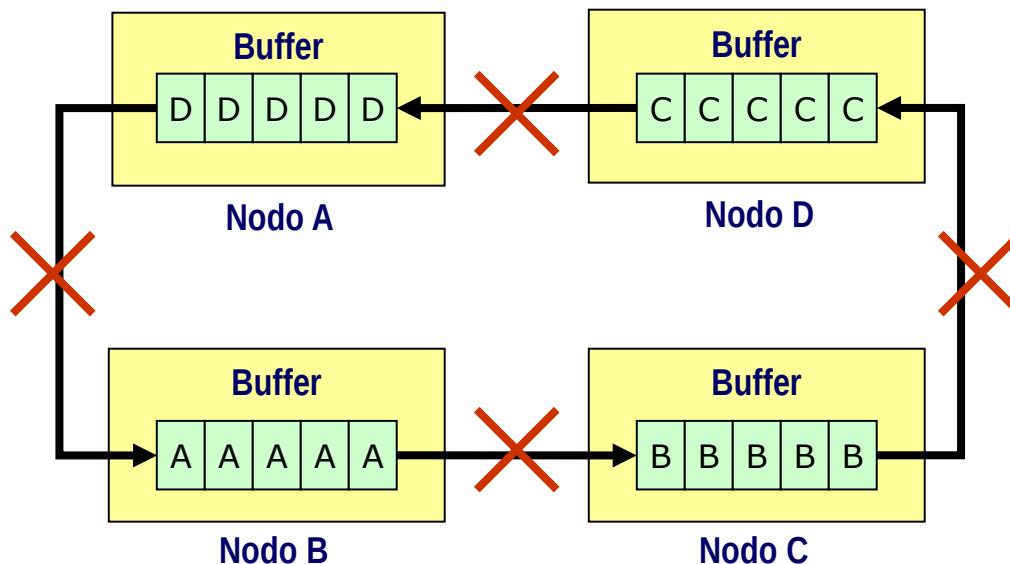
- Permiten que varios paquetes compartan el mismo enlace (a nivel de flit)
- Mejoran el ancho de banda y la latencia al disminuir la probabilidad de bloqueos
- Se aplica con el resto de mecanismos



Técnicas de conmutación

¿Bloqueos?

- Algunos paquetes no pueden alcanzar el destino
- Capacidad de los *buffers* finita
- Canales ocupados



Bloqueos. Clasificación:

- **Interbloqueos (deadlocks)**
 - Recursos no disponibles para el avance de los paquetes
 - *Buffers* ocupados
 - Bloqueo permanente
- **Bloqueos activos (livelocks)**
 - Los paquetes nunca llegan a su destino
 - Canales ocupados por otros paquetes
 - Sólo ocurre si se permiten caminos no mínimos
- **Inanición** (los recursos siempre se asignan a otros paquetes)

Técnicas de conmutación

Bloqueos. Soluciones:

- Inanición
 - Emplear un esquema de asignación de recursos correcto
 - Cola circular con distinta prioridad
- Bloqueos activos
 - Usar solo rutas mínimas
 - Usar rutas no mínimas restringidas
 - Dar mayor probabilidad a caminos mínimos respecto a no mínimos
- Interbloqueos
 - Prevención
 - Evitación
 - Recuperación

Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

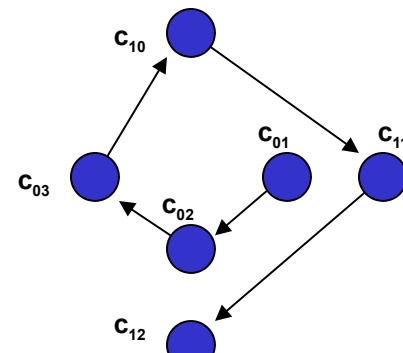
Técnicas de conmutación

Prevención de interbloqueos:

- Estrategia muy conservadora: se asignan todos los recursos para transmitir un mensaje antes de iniciar la transmisión
- Un flit de sondeo establece el camino
- Si existe bloqueo, retrocede y libera recursos

Evitación de interbloqueos

- Los recursos se asignan a medida que el mensaje atraviesa la red
- Un recurso se asigna a un paquete si el estado resultante es seguro (grafo de dependencias acíclico)

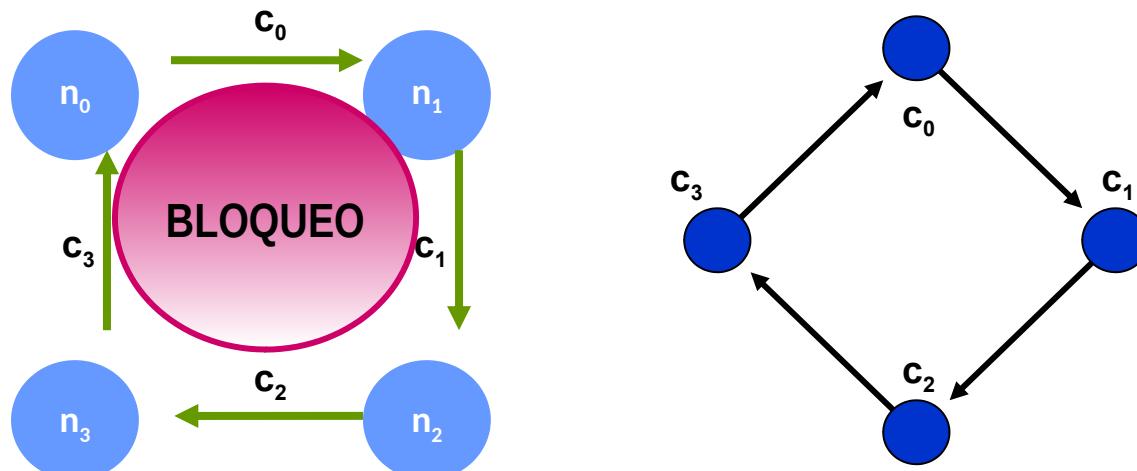


Técnicas de conmutación

Evitación de interbloqueos - grafo de dependencias

Teorema: Una función de encaminamiento determinista F para una red R está libre de interbloqueos si sólo si no existen ciclos en su grafo de dependencia de canales

Ej. Anillo unidireccional



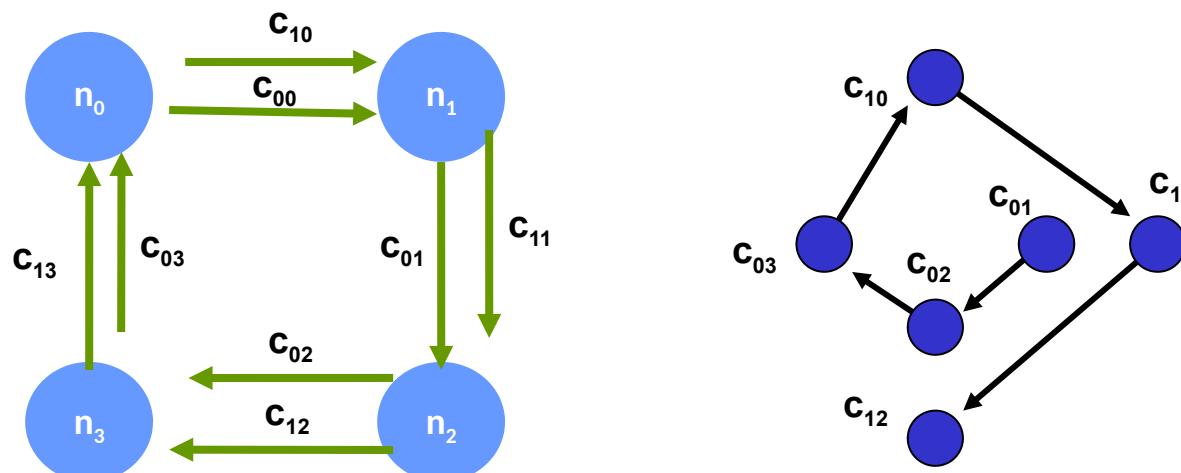
Ingeniería de los Computadores

Unidad 4.4 Técnicas de conmutación

Técnicas de conmutación

Evitación de interbloqueos - grafo de dependencias

Ej. Anillo unidireccional con canales virtuales



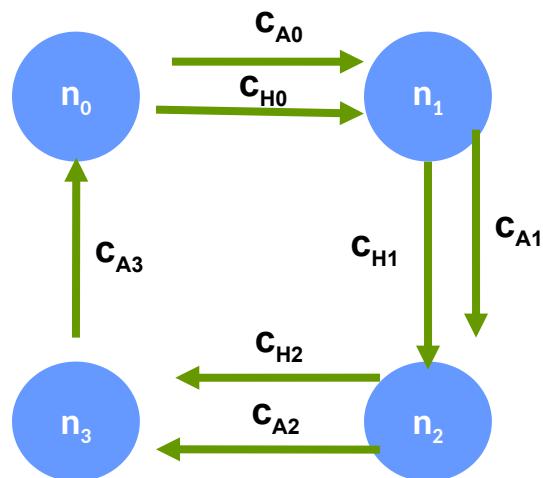
Función de encaminamiento:

Usar c_{0i} si $j < i$, o c_{1i} si $j > i$

Técnicas de conmutación

Evitación de interbloqueos - grafo de dependencias

Teorema: Una función de encaminamiento adaptativa F para una red R está libre de interbloqueos si, existiendo dependencias cíclicas, cada paquete encuentra un caminos libre de bloqueos para llegar al destino



Función de encaminamiento:
Usar o c_{Ai} si $j \neq i$, o c_{Hi} si $j > i$

Técnicas de conmutación

Recuperación de interbloqueos

- Estrategia optimista: supone que rara vez ocurre un bloqueo
- Los recursos se asignan a los mensajes sin ninguna comprobación adicional
- Si existe bloqueo, se liberan bloqueos
- Se reasignan los recursos a otro mensaje
- Detección de bloqueos en el nodo origen

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento: Los algoritmos de encaminamiento establecen el camino que sigue cada mensaje o paquete

Propiedades derivadas:

- Conectividad: capacidad de encaminar desde cualquier nodo origen a cualquier nodo destino
- Adaptabilidad: capacidad de encaminar a través de caminos alternativos
- Evitación de bloqueos: capacidad de garantizar que los mensajes no se bloquearán en la red
- Tolerancia a fallos: capacidad de encaminar en presencia de componentes defectuosos

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmos de encaminamiento. Criterios de clasificación

- El número de destinos
- Quién toma la decisión del encaminamiento
- Cómo se realiza la implementación
- La adaptabilidad
- La progresividad
- La minimalidad del encaminamiento
- El número de caminos proporcionados

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmos de encaminamiento. Clasificación

Según número de destinos:

- Monodestino (*unicast*)
- Multidestino (*multicast*)

Según decisión de encaminamiento:

- Centralizados
- En origen (El nodo fuente especifica el camino y la ruta se almacena en la cabecera del paquete) → encaminamiento *street-sign*
- Distribuidos (Los nodos intermedios deciden hacia dónde encaminar) → Idóneo para topologías irregulares
- Multifase

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmos de encaminamiento. Clasificación

Según la implementación:

- Tablas (encaminamiento por intervalos)
- Máquinas de estados finitos (FSM) → topologías ortogonales (encaminamiento por orden de dimensión)

Según adaptabilidad

- Deterministas:
 - Siempre suministran el mismo camino
 - Rendimiento pobre si tráfico no uniforme
- Adaptativos:
 - Consideran el estado de la red
 - Totalmente adaptativos: pueden usar todos los canales
 - Parcialmente adaptativos: usan un subconjunto

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmos de encaminamiento. Clasificación

Según progresividad:

- Progresivos
- *Backtracking: EPB (Exhaustive Profitable Backtracking)*

Según minimalidad:

- Mínimos
 - ¿Algoritmos deterministas progresivos y mínimos?
- No mínimos
 - Mayor flexibilidad
 - Encaminamiento tolerante a fallos

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmo determinista: encaminamiento por orden de dimensión

- Topologías ortogonales
- Selección de canales sucesivos con orden específico
- Tipo determinístico
- La diferencia en una dimensión se anula antes de pasar a la siguiente
- Ejemplos:
 - *Street-sign* (fuente y sin tabla)
 - encaminamiento XY (distribuido y sin tabla)
 - encaminamiento e-cube
 - Intervalo (distribuido y con tabla de consulta)
- Libre de interbloqueos en mallas e hipercubos (en toros es necesario usar canales virtuales y establecer un orden en su utilización)

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento XY (ordenado por dimensión)

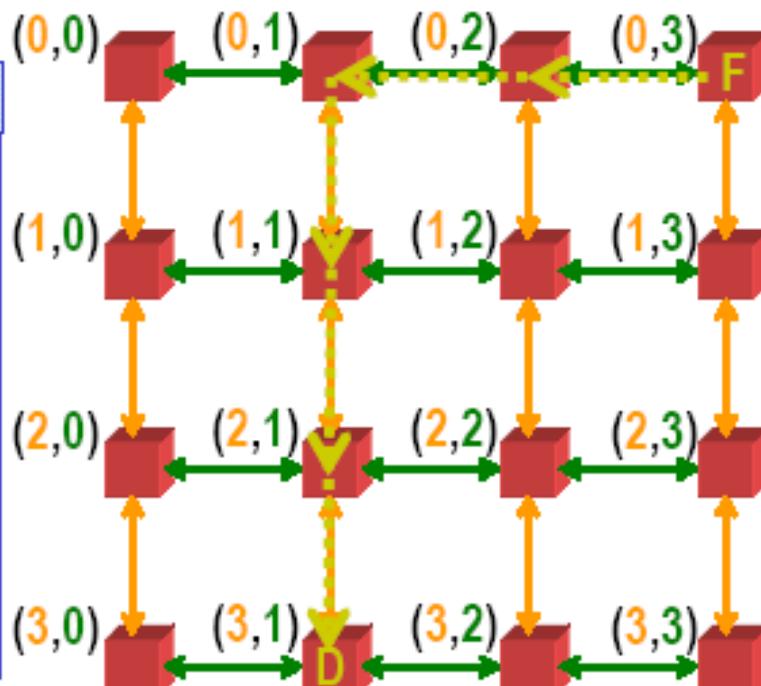
Ej. 3: malla 2D Ord. por dimensión 0-1

Entrada: actual $A = (a_1, a_0)_k$
destino $D = (d_1, d_0)_k$

Salida: Canal $cs = (D0-, D0+, D1-, D1+, I)$.

Procedimiento:

```
dist0 =  $d_0 - a_0$ ; dist1 =  $d_1 - a_1$ ;
if ( dist0 < 0 ) cs = D0- ;
if ( dist0 > 0 ) cs = D0+ ;
if ( dist0 = 0 & dist1 < 0 ) cs = D1- ;
if ( dist0 = 0 & dist1 > 0 ) cs = D1+;
if ( dist0 = 0 & dist1 = 0 ) cs = I ;
```



Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento ordenado por dimensión en tablas



Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Modelo de giros (*turn-model*)

- Redes estáticas (topologías ortogonales) y redes dinámicas
- Ejemplo:
 - West-First en mallas 2D (distribuido, sin tablas, parcialmente adaptativo y puede ser mínimo o no mínimo)
- Interbloqueos
 - Ciclos que engloban varias direcciones → Se evitan prohibiendo al menos un cambio de dirección para cada ciclo
 - Ciclos sin cambio de dirección → Se evitan añadiendo canales virtuales y estableciendo un orden de uso

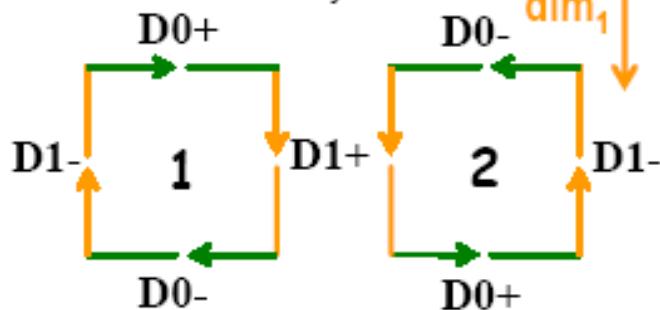
Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

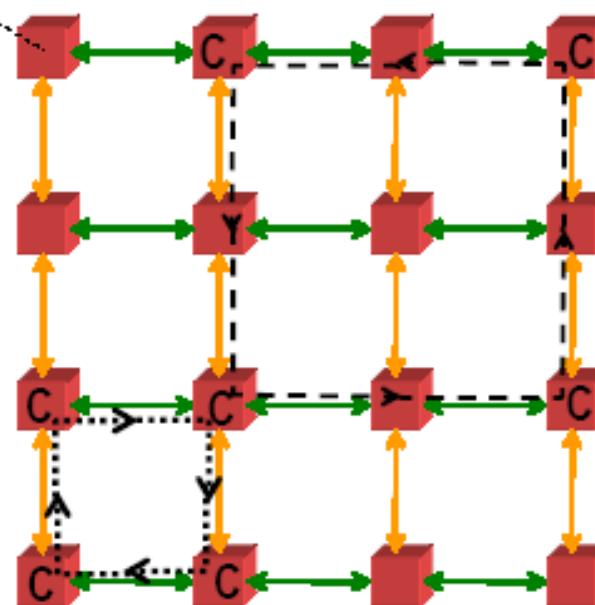
Técnicas de encaminamiento

Algoritmo West-First

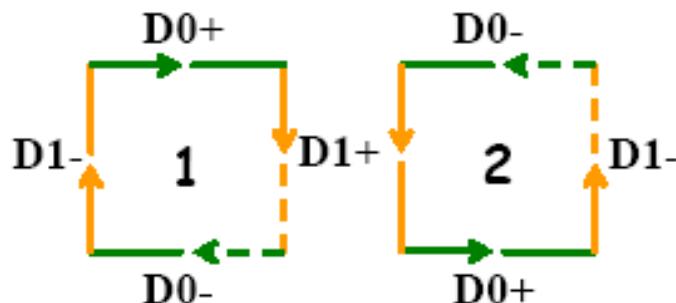
2 y 3. Identificar **cambios de dirección** y **ciclos**:



1. **Clasificación** de los enlaces de acuerdo a su **dirección**:



4. **Prohibir** un cambio de dirección en cada **ciclo**:

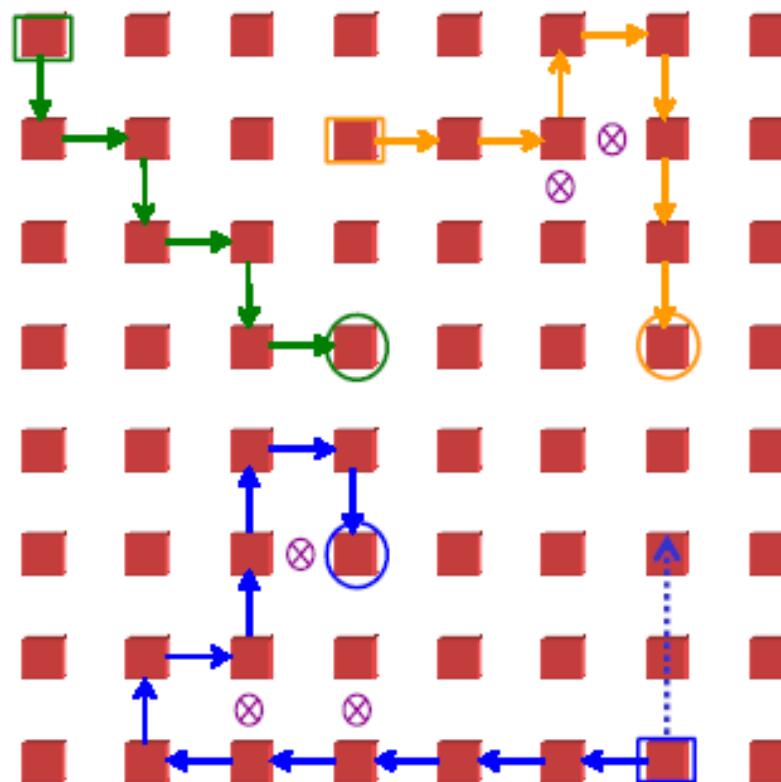


Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmo West-First – implementación no mínima



Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmo West-First para mallas 2D

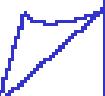
Entrada: Actual A = (a_1, a_0)

Destino D = (d_1, d_0)

Salida: Canal cs

Procedimiento:

```
dist0 =  $d_0 - a_0$ ; dist1 =  $d_1 - a_1$ ;
if ( dist0 < 0 ) cs = D0-;
if ( dist0 > 0 & dist1 > 0 ) cs = Sel(D0+,D1+);
if ( dist0 > 0 & dist1 < 0 ) cs = Sel(D0+,D1-);
if ( dist0 > 0 & dist1 = 0 ) cs = D0+;
if ( dist0 = 0 & dist1 > 0 ) cs = D1+;
if ( dist0 = 0 & dist1 < 0 ) cs = D1-;
if ( dist0 = 0 & dist1 = 0 ) cs = I;
```

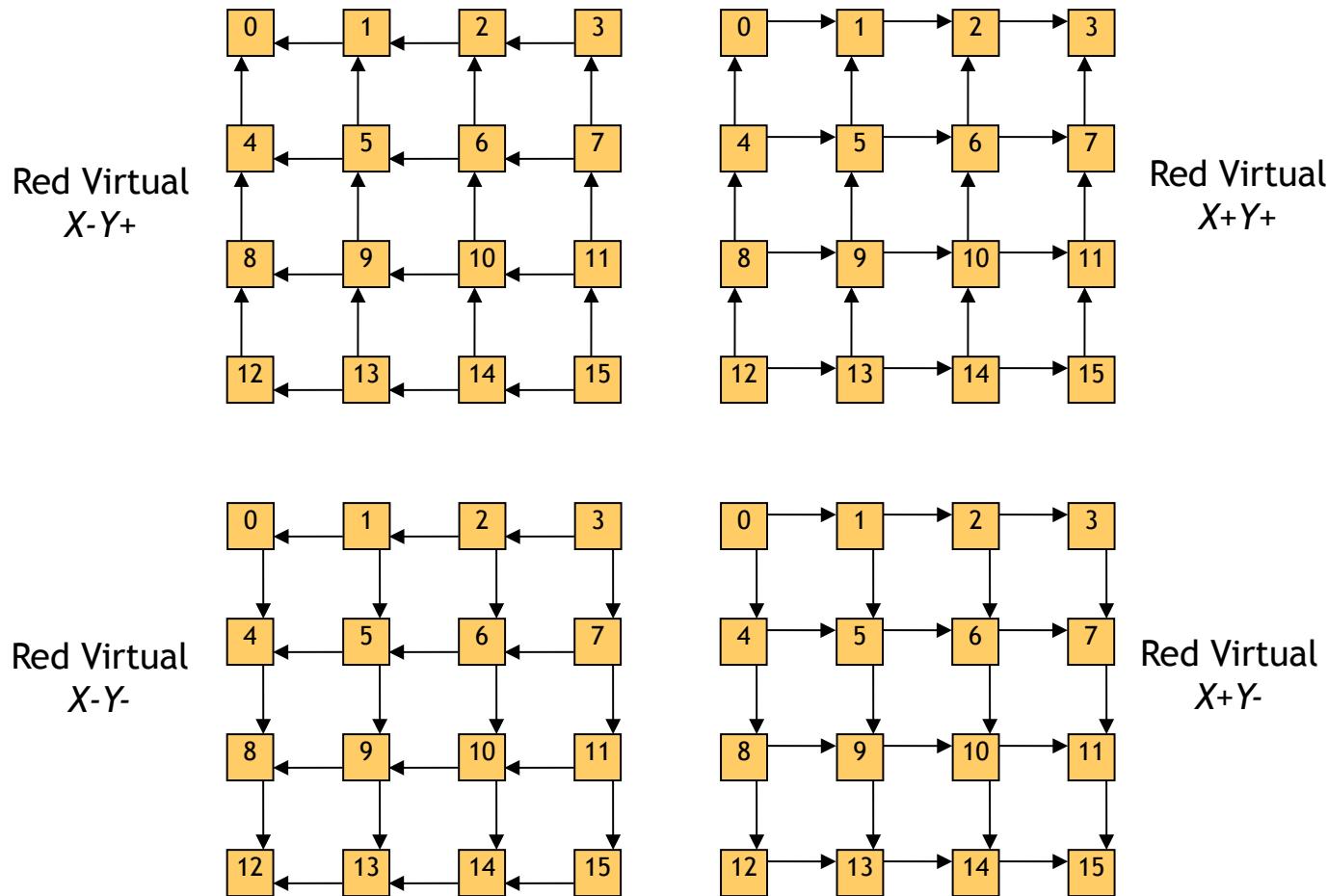


Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Algoritmo totalmente adaptativo: redes virtuales



Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento en redes tipo mariposa

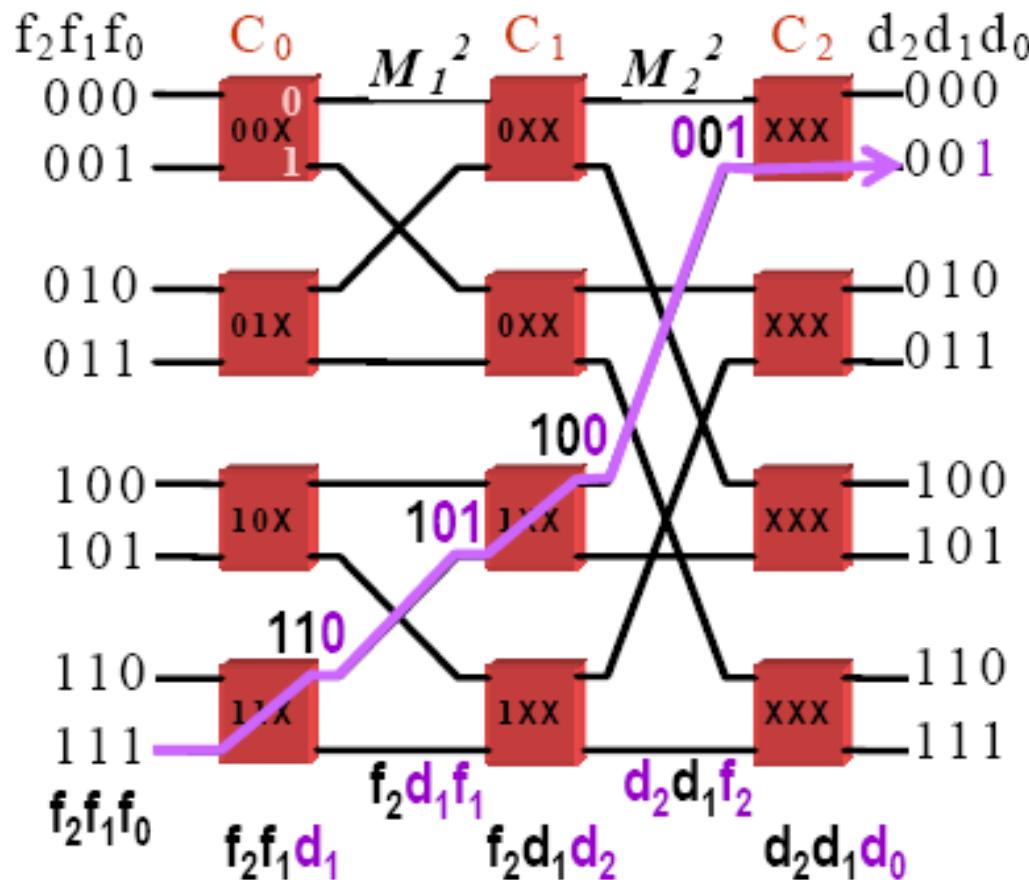
- Tipo determinístico
- Se usa la dirección del destino D en base b. Cada dígito controla una etapa de conmutadores
- No hay interbloqueo porque la topología no presenta ciclos
- Ejemplos:
 - ✓ Red Omega y red Cubo: d_i controla la etapa $n-i-1$
 - ✓ Red mariposa: d_i controla la etapa $i-1$ y d_0 la etapa $n-1$

Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento en redes tipo mariposa

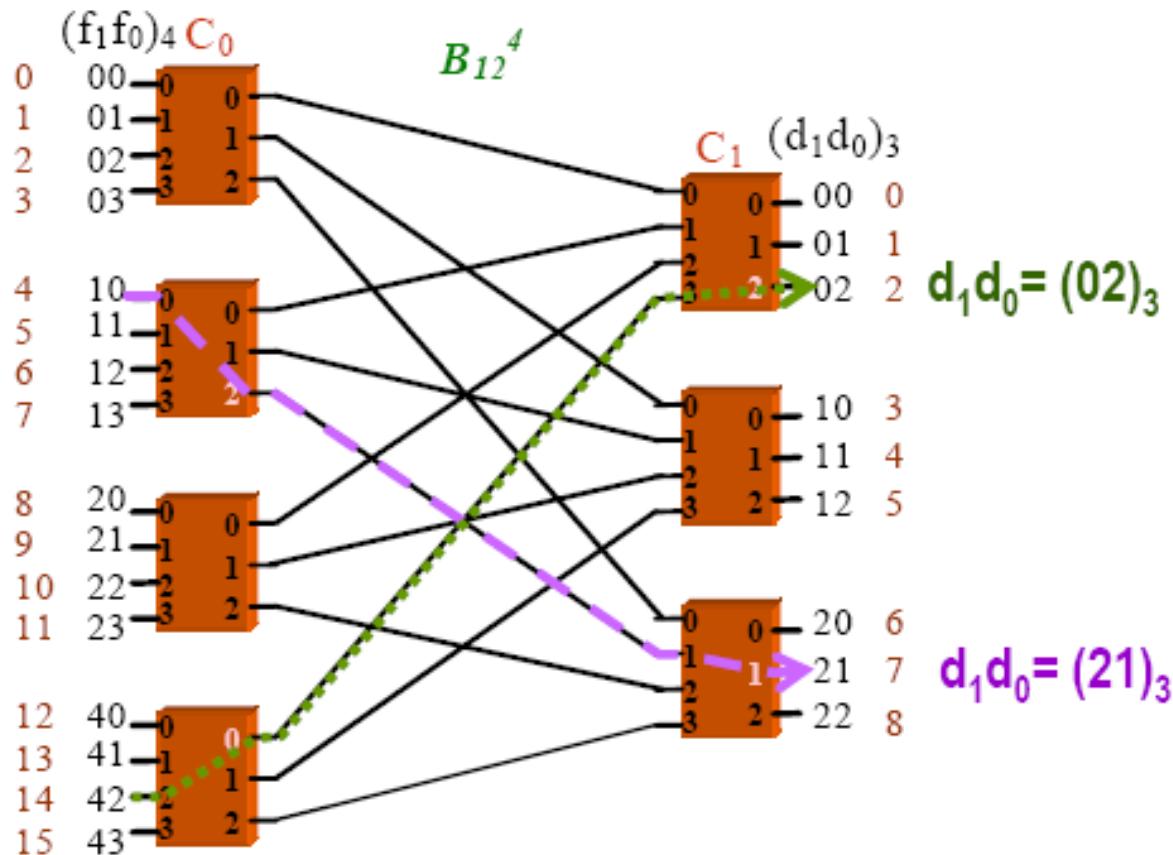


Ingeniería de los Computadores

Unidad 4.5 Técnicas de encaminamiento

Técnicas de encaminamiento

Encaminamiento en redes tipo mariposa



Ingeniería de los Computadores

Unidad 5. Memoria en arquitecturas
paralelas

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Consistencia de memoria

Problema: Transmisión de información entre procesadores en memoria compartida

Ejemplo:

```
/* El valor inicial de A y flag es 0 */
P1
A=1
flag=1

P2
while (flag == 0); /*bucle vacío*/
print A
```

P2 ha de realizar una **espera activa** hasta que *flag* cambie a '1'.

Después imprimirá A = '1' (suponiendo que en P1 "A=1" es una instrucción anterior a "flag=1") →

¿COHERENCIA DEL SISTEMA DE MEMORIA? → Escrituras en orden de programa.

Pero ... y si "flag=1" se ejecuta antes de "A=1" ¿?

En un **espacio de direcciones compartido** es necesario un **modelo de consistencia de la memoria**. Su objetivo es **especificar restricciones en el orden de las operaciones en la memoria** y proporcionar una visión uniforme de las mismas a los demás procesadores, logrando una abstracción de las operaciones de memoria independiente de la localización de las operaciones de memoria (módulos) y de los procesos involucrados.

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Consistencia de memoria

- “Un modelo de consistencia de memoria especifica el **orden** en el cual las operaciones de acceso a memoria deben **parecer** haberse realizado (operaciones de lectura y escritura)”
- En un procesador (o sistema **monoprocesador**), el orden en el que deben parecer haberse ejecutado los accesos a memoria es el **orden secuencial especificado por el programador** (o la herramienta de programación en el código que añade), denominado **orden de programa**.
- Tanto el hardware como la herramienta de programación si que pueden alterar dicho orden, para mejorar las prestaciones, pero debe parecer en la ejecución del programa que no se ha alterado.

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Consistencia de memoria → Se debe garantizar que:

- Cada lectura de una dirección, proporcione el último valor escrito en dicha dirección (**dependencia de datos lectura después de escritura**)
- Tras varias escrituras en una dirección se debe retornar el último valor escrito (**dependencia escritura después de escritura**)
- Si se escribe en una dirección a la que previamente se ha accedido para leer, no se debe obtener en la lectura previa el valor escrito posteriormente en la secuencia del programa especificada por el programador (**dependencia escritura después de lectura**)
- No se puede escribir en una dirección si la escritura depende de una condición que no se cumple (**dependencias de control**)

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Ejemplo 1: 2P, caché write-through (cada vez que se escribe una línea de caché se actualiza la memoria principal)

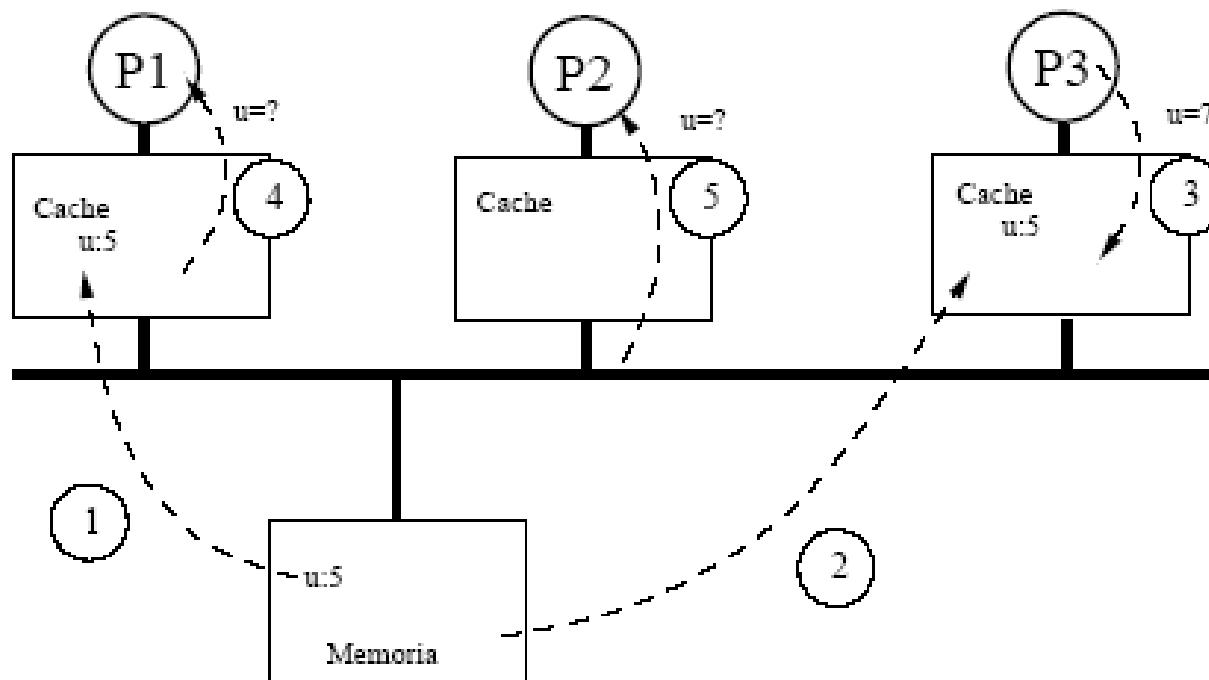
Secuencia	Acción	Caché A	Caché B	X en MP
1	CPU A lee X	1	-	1
2	CPU B lee X	1	1	1
3	CPU A escribe 0 en X	0	1	0

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Ejemplo 2: 3P, caché write-through/write-back (se actualiza la memoria principal escribiendo todo el bloque cuando se desaloja de la caché → aún más problemática)



Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

- Un sistema de memoria es **coherente** si cualquier lectura de un dato devuelve el **valor más reciente** escrito de ese dato
- Aspectos críticos del sistema de memoria compartida: los datos devueltos por una lectura (coherencia) y cuándo un valor escrito será devuelto por una lectura (consistencia)
- Un sistema de memoria es coherente si cumple:
 - ✓ **Preservación del orden del programa:** una lectura por un procesador P de una posición X, que sigue a una escritura de P a X, sin que ningún otro procesador haya escrito nada en X entre la escritura y la lectura de P, siempre devuelve el valor escrito por P.
 - ✓ **Visión coherente de la memoria:** una lectura por un procesador de la posición X, que sigue a una escritura por otro procesador a X, devuelve el valor escrito si la lectura y escritura están suficientemente separados y no hay otras escrituras sobre X entre los dos accesos.
 - ✓ **Serialización de operaciones concurrentes de escritura:** Las escrituras a la misma posición por cualquiera dos procesadores se ven en el mismo orden por todos los procesadores.

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Un sistema de memoria multiprocesador es **coherente** si el resultado de cualquier ejecución de un programa es tal que, para cada localización **es posible construir una hipotética ordenación secuencial de todas las operaciones realizadas sobre dicha localización** que sea consistente con los resultados de la ejecución y en el cuál:

1. Las **operaciones** emitidas por un procesador particular ocurren en **la secuencia indicada** y en el orden en el que dicho procesador las emite al sistema de memoria
2. El valor devuelto por cada operación de lectura es el valor escrito por la última escritura en esa localización en la secuencia indicada

Ingeniería de los Computadores

5.1 Introducción y motivación

Introducción

Dos estrategias para abordar la coherencia de las cachés:

- Resolución **software**: el **compilador** y el programador evitan la incoherencia entre cachés de datos compartidos.
- Hardware: transparente al programador mediante la provisión de mecanismos **hardware**, esta es la solución más utilizada para mantener la coherencia.

Políticas para mantener la coherencia:

- **Invalidación de escritura o coherencia dinámica (write invalidate)**: siempre que un procesador modifica un dato de un bloque en la caché, invalida todas las demás copias de ese bloque guardadas en las otras cachés de los procesadores.
- **Actualización en escritura (write-update o write-broadcast)**: esta política lo que hace es actualizar la copia en las demás cachés en vez de invalidarla.

Ingeniería de los Computadores

5.2 Memoria en multiprocesadores

Clasificación

Clasificación de multiprocesadores atendiendo a la distribución de memoria

- **UMA** (Uniform Memory Access)
- **NUMA** (Non-Uniform Memory Access)
- **COMA** (Cache-Only Memory Architecture)

Ingeniería de los Computadores

5.2 Memoria en multiprocesadores

Multiprocesadores UMA

Tipo UMA

- Memoria **centralizada, uniformemente compartida** entre procesadores (todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de la memoria)
- Sistema fuertemente acoplado: alto grado de compartición de recursos (memoria) entre los procesadores → dependencia funcional entre ellos
- Cada procesador puede disponer de caché
- Periféricos compartidos entre procesadores
- Aplicaciones de propósito general y de tiempo compartido por múltiples usuarios
- Sincronización y comunicación entre procesadores utilizando variables compartidas
- Acceso a memoria, periféricos y distribución de procesos S.O.:
 - ✓ Equitativo → Symmetric (shared-memory) Multi Processors (SMPs)
 - ✓ No equitativo → Asymmetric (shared-memory) Multi Processors (AMPs)CC-UMA (Caché-Coherent Uniform Memory Access)

Ingeniería de los Computadores

5.2 Memoria en multiprocesadores

Multiprocesadores NUMA

Tipo NUMA

- Memoria compartida con tiempo de acceso dependiente de la ubicación de procesadores
- Sistema débilmente acoplado: cada procesador dispone de una memoria local a la que puede acceder más rápidamente
- Sistema de acceso global a memoria: local, global, local de otros módulos
- CC-NUMA (Caché-Coherent Non-Uniform Memory Access) → Memoria compartida distribuida y directorios de caché
- Ventajas
 - ✓ Escalado de memoria con + coste/rendimiento
 - ✓ Reducción de latencia de acceso a memorias locales

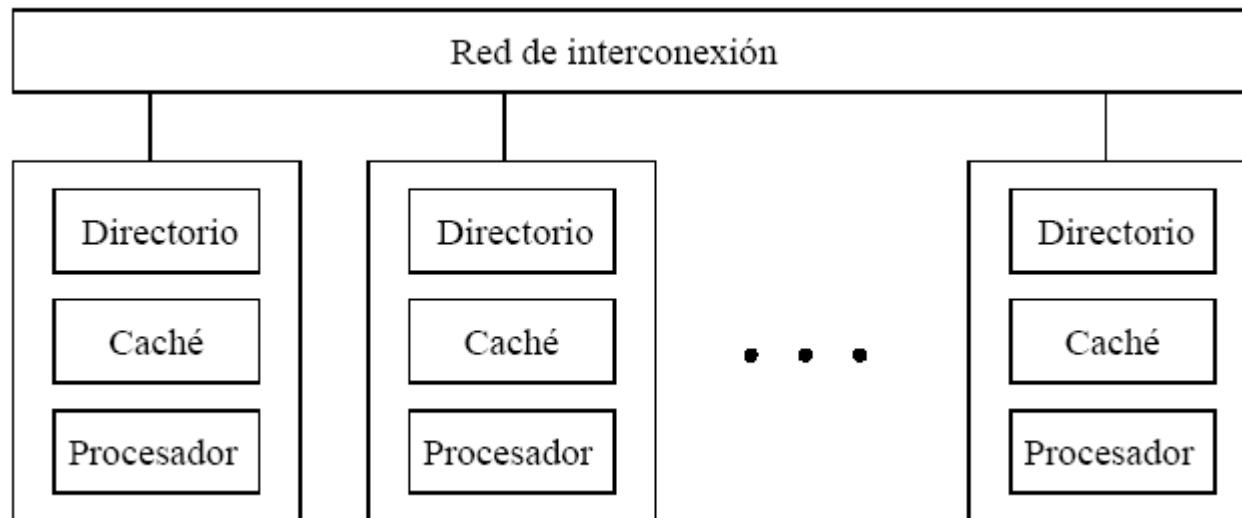
Ingeniería de los Computadores

5.2 Memoria en multiprocesadores

Multiprocesadores COMA

Tipo COMA

- Sólo se usa una caché como memoria
- Es un caso particular de NUMA donde las memorias distribuidas se convierten en cachés
- Las cachés forman un mismo espacio global de direcciones
- Acceso a las cachés por **directorio distribuido**



Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

- Protocolos de **sondeo** o *snoopy*:
 - Utilizado en redes donde la difusión (broadcast) es posible
 - Redes basadas en bus
 - Cada caché monitoriza el estado del bus y las transacciones de las demás cachés
- Protocolos basados en directorio:
 - Utilizado en redes donde el broadcast no es posible a causa de la degradación
 - Redes multietapa
 - Se utiliza para ello un **directorío centralizado o distribuido**

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

- **Protocolos de sondeo o snoopy**
 - **Objetivo:** garantizar las transacciones necesarias en el bus para operaciones de memoria y que los controladores de caché observen y actúen en transacciones relevantes
 - **Todas las transacciones** aparecen en el bus y **son visibles** para los procesadores en el mismo orden en el que se producen
 - Utilizado en sistemas multiprocesador con bus y pocos elementos conectados
 - Cada procesador indica el estado de cada línea de su caché
 - La red de interconexión debe permitir broadcast (difusión)
 - Cada caché monitoriza las transacciones de las demás cachés observando el bus
 - Se utiliza un algoritmo distribuido representado como un conjunto de máquinas de estados finitos que cooperan entre sí:
 - Conjunto de estados asociado con los bloques de memoria en las cachés
 - Diagrama de transición entre estados
 - Acciones asociadas a las transiciones entre estados

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolos de sondeo o snoopy – Protocolo de **invalidación en escritura** (**write invalidate**)

- Constituyen las estrategias **más robustas y extendidas**
- Basado en asegurar que un procesador tiene acceso exclusivo a un dato antes de que acceda a él
- Se consigue invalidando todas la líneas de todas las cachés que contengan ese dato que esta siendo escrito en ese momento
- Cada controlador de caché observa las transacciones de memoria (observa el bus) de los otros controladores para mantener su estado interno
- Control de coherencia de caché mediante transacciones de lectura y de lectura exclusiva
- Cuando un procesador quiera leer un dato invalidado, falle su caché y tenga que ir a memoria a buscarlo
- Ejemplo: Protocolos **MSI**, **MESI**, Write Once y Berkeley

Protocolos de coherencia

Protocolos de sondeo o snoopy – Protocolo de **actualización en escritura (*write-update*)**

- Menos utilizados que el anterior (*write-invalidate*)
- Cada vez que un procesador escribe un dato, se actualizan las cachés que contienen el dato en los demás procesadores
- Problemas para mantener el ancho de banda bajo control debido a la alta cantidad de transacciones que se generan
- Mejora: escritura en una localización → actualización de cachés relacionadas
- Ejemplo: Protocolos *Dragon* y *Firefly*

Protocolos de coherencia

Protocolos de sondeo (snoopy) - **MSI**

- Protocolo de invalidación básico para cachés write-back
 - Estados: Inválido (**I**), Compartido (**S**), Modificado (**M**)
-
- ✓ **Inválido:** no es válido el bloque
 - ✓ **Compartido:** el bloque esta presente en la caché y no ha sido modificado, **la memoria principal esta actualizada y cero o más** cachés pueden tener también una copia actualizada (compartida)
 - ✓ **Modificado:** únicamente este procesador tiene una copia válida, la copia de la memoria principal esta anticuada y ninguna otra caché puede tener una copia válida del bloque (ni en estado modificado ni compartido)

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - **MSI**

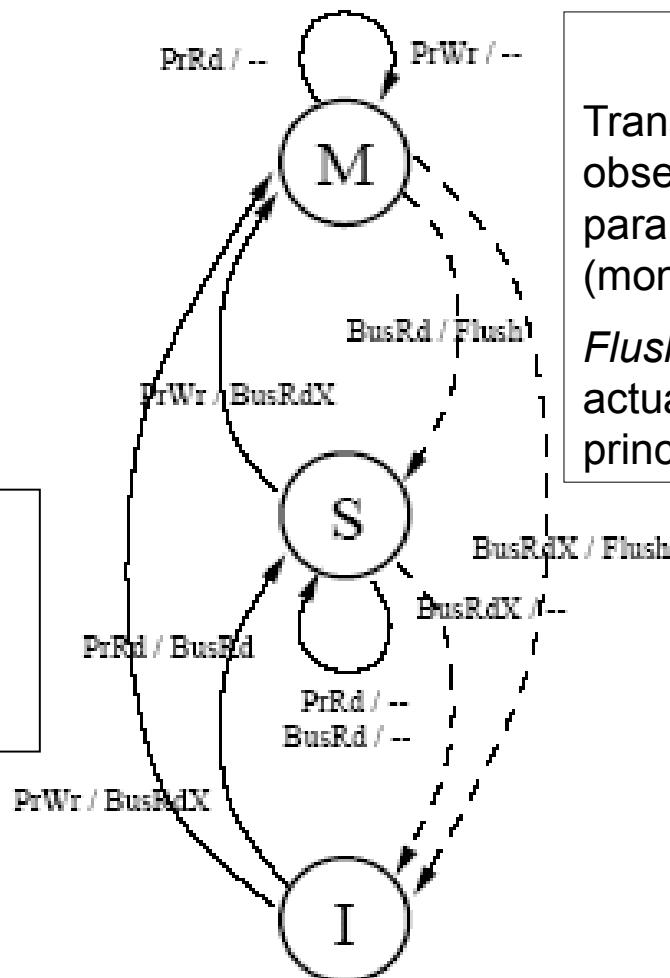
Operaciones del procesador
+ transacciones que se generan en el bus por dichas operaciones de procesador

Fallo lectura [**miss**] (I) → M,S

Fallo escritura [**miss**] (I) → M,S

Acierto escritura [**hit**] (S) → S, (M) → M

Acierto lectura [**hit**] (S,M)



Transacciones en el bus observadas por las cachés para cambiar su estado (monitorización)

Flush → dato en bus + actualización de memoria principal

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - MESI (o Illinois)

- Ampliación del protocolo de invalidación de 3 estados. Refinamiento para aplicaciones “secuenciales” que corren en **multiprocesadores** (carga común usada en multiprocesadores de pequeña escala)
- En el MSI el programa cuando lee y modifica un dato tiene que generar 2 transacciones incluso en el caso de que no exista compartición (solo presente en una caché) del dato (*BusRd* y *BusRdX*)
- Estados: Modificado (**M**), *Exclusivo (**E**), Compartido (**S**), Inválido (**I**)
El estado exclusivo (E): indica que el bloque es la **única copia** (y por tanto exclusiva) del sistema multiprocesador, que **no está modificada** y la **memoria principal está actualizada**
- Al ser exclusivo es posible realizar una escritura o pasar al estado modificado sin ninguna transacción en el bus, al contrario que en el caso de estar en el estado compartido; pero no implica pertenencia, así que al contrario que en el estado modificado la caché no necesita responder al observar una petición de dicho bloque (la memoria tiene una copia válida)
- Publicado por la Universidad de Illinois en 1984

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - MESI

BusRd(S) → cuando ocurre la transacción de lectura en el bus, se activa la señal S.

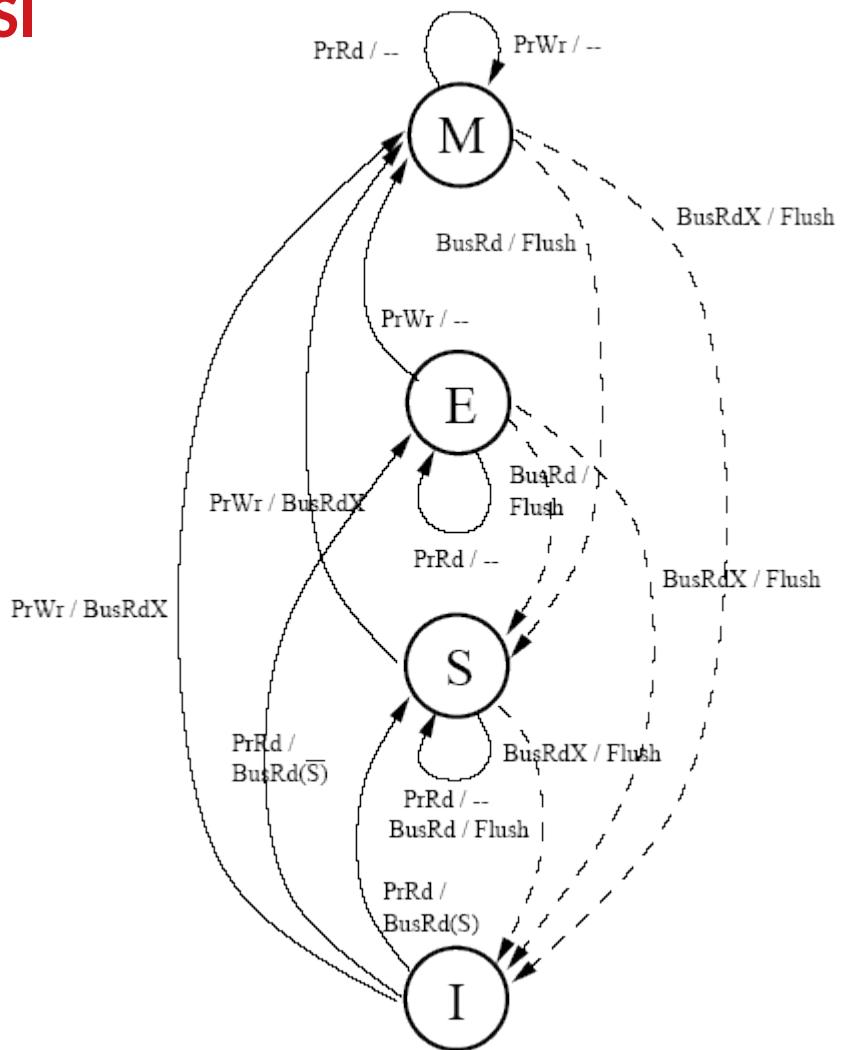
Este protocolo necesita una señal adicional en el bus (señal S) para que los controladores puedan determinar en una transacción BusRd si existe otra caché que tenga el mismo bloque

Fallo lectura (I) → M,S,E

Fallo escritura (I) → M,S,E

Acierto escritura (S) → S, (M,E) → O

Acierto lectura (S,M,E)



Protocolos de coherencia

Protocolo de sondeo (snoopy) - **Write Once**

- Cada línea de caché tiene **dos bits** extra para almacenar el estado de esa línea
- Líneas adicionales de control para inhibir la memoria principal
- Estados líneas de caché:
 - ✓ Válida (**V**): la línea de caché, es consistente con la copia de memoria, ha sido leída de la memoria principal y no ha sido modificada
 - ✓ Inválida (**I**): la línea no se encuentra en la caché o no es consistente con la copia en memoria
 - ✓ Reservada (**R**): los datos han sido escritos una única vez desde que se leyó de la memoria compartida, la línea de caché es consistente con la copia en memoria que es la única otra copia
 - ✓ Sucia (**S**): la línea de caché ha sido escrita más de una vez, y la copia de la caché es la única en el sistema (por lo tanto inconsistente con el resto de copias)
- Transiciones según operaciones sobre cachés: fallo de lectura, acierto de escritura, fallo de escritura, acierto de lectura y cambio de línea

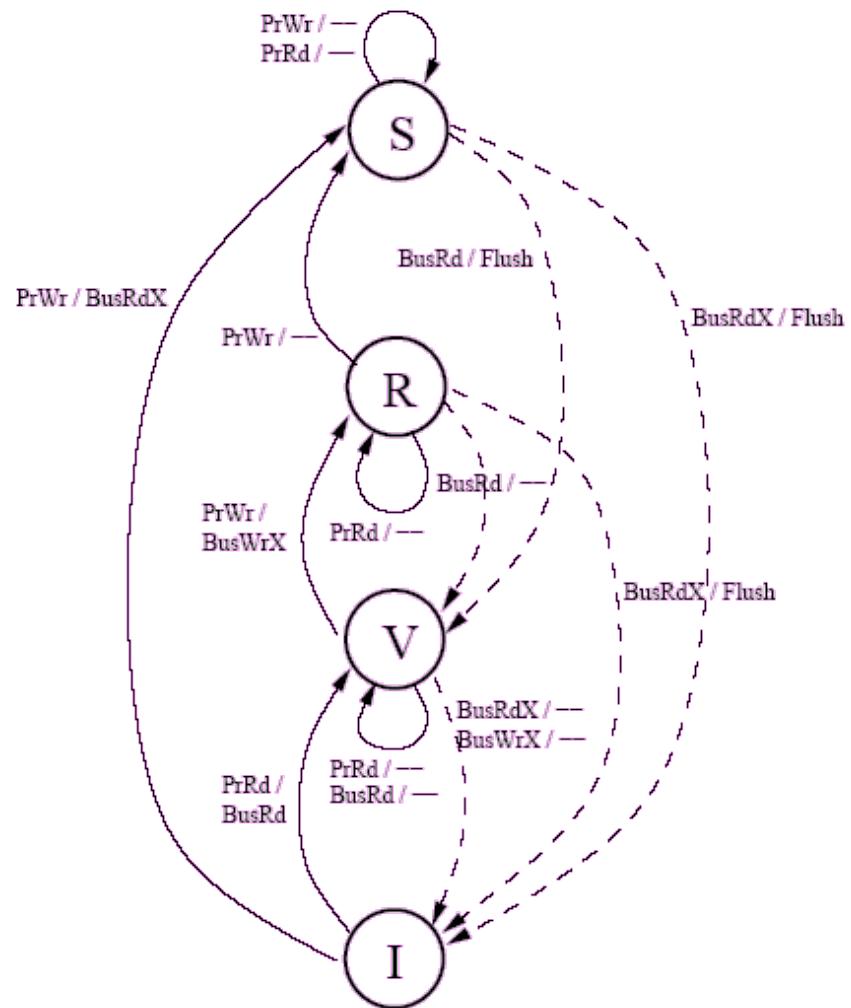
Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - **Write Once**

Diagrama de transiciones entre estados



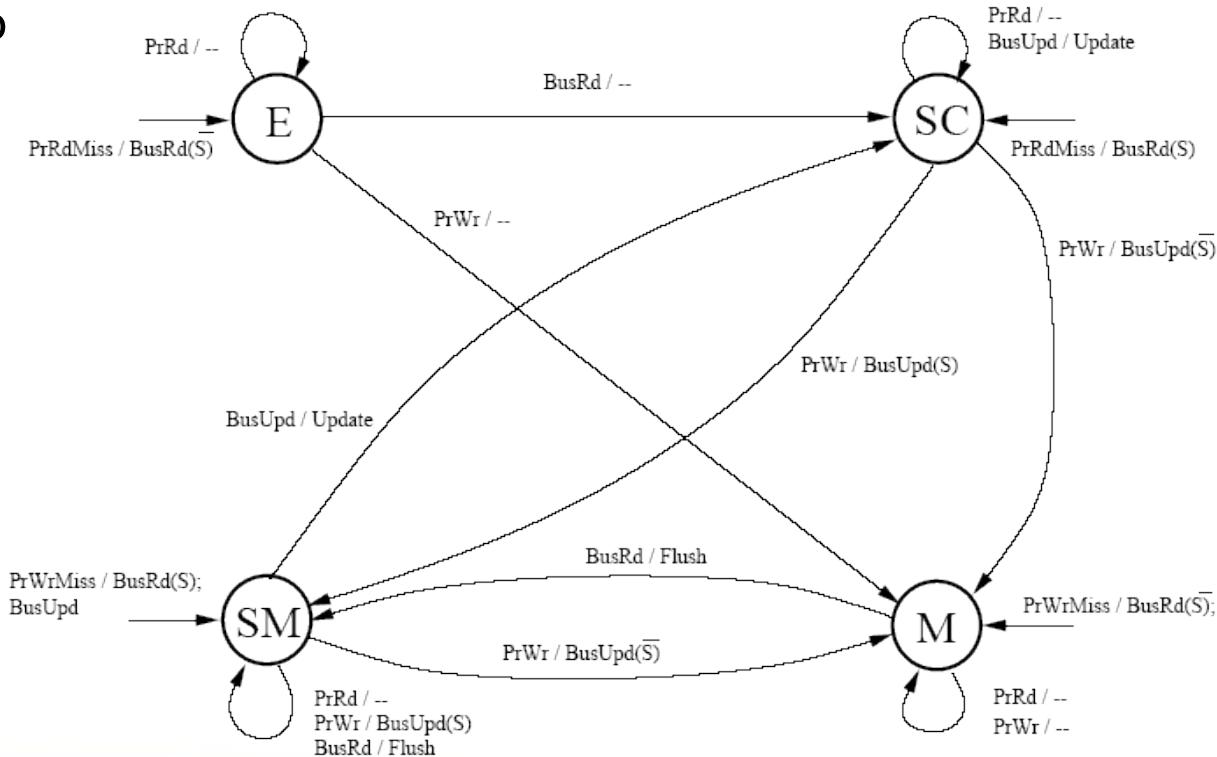
Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - **Dragon**

- Protocolo de actualización en escritura básico para cachés *write-back*
- Estados: Exclusivo (**E**), Compartido (**C**), Compartido_Modificado (**SM**) y Modificado (**M**)
- Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



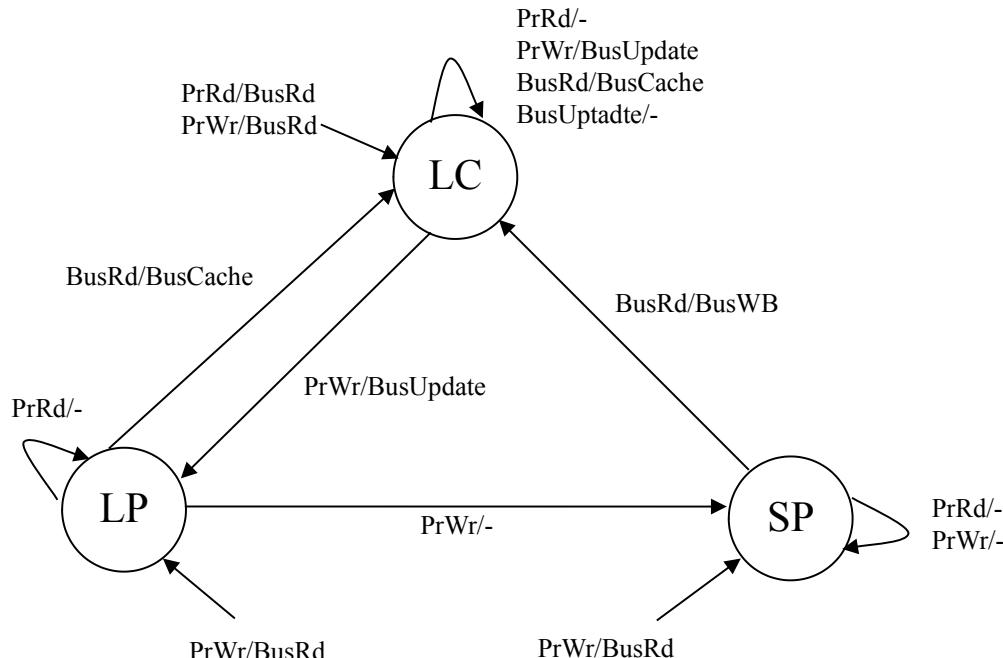
Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de coherencia

Protocolo de sondeo (snoopy) - **Firefly**

- Protocolo de actualización en escritura, que hace uso de una línea compartida especial de bus
- Estados de línea de caché: Lectura_Privada(**LP**) (Exclusive), Lectura_Compartida (**LC**) (Shared) y Sucia_Privada (**SP**) (Modified) (MESI)
- Transiciones según operaciones sobre cachés: fallo de lectura, escritura, reemplazo



Protocolos de coherencia

Protocolos de sondeo (snoopy): Rendimiento

- Consideraciones a tener en cuenta para evaluar el rendimiento de un protocolo:
 - ✓ Tráfico causado por fallos de caché
 - ✓ Tráfico de comunicación entre cachés
- Diferencias de rendimiento entre protocolos de invalidación y actualización:
 - ✓ Varias escrituras a la misma palabra sin lecturas intermedias
 - ✓ Líneas de caché de varias palabras: los de invalidación trabajan sobre el bloque y los de actualización sobre palabras para aumentar la eficiencia
 - ✓ Retraso entre escritura de palabra en un procesador y la lectura por parte de otro → mejor en los protocolos de actualización
- Rendimiento según los requisitos:
 - ✓ **Para aprovechar mejor el ancho de banda del bus y la memoria, se utilizan los protocolos de invalidación**
 - ✓ Migración de procesos o sincronización intensivas: cuando se requiera mucha migración de procesos o mucha sincronización, un protocolo de invalidación va a trabajar mucho mejor que uno de actualización

Protocolos de directorio

Protocolos de directorio (centralizado o distribuido)

- Sistemas multiprocesador con varias subredes locales de interconexión y muchos elementos conectados
- Directorio común que almacena el estado de las líneas de caché
- La red de interconexión no soporta *broadcast* o provoca mucha degradación
- El comando de invalidación/actualización se envía a aquellas caches que disponen de copia local de un bloque
- La entrada del directorio tiene un **bit de permiso de actualización**
- Existen **varias copias compartidas de la misma línea de caché** para mejorar el rendimiento sin incrementar el tráfico

Protocolos de directorio

Protocolos de **directorío centralizado**

- Tabla **centralizada** que almacena ubicación de cada copia de caché
- Tamaño grande → búsqueda asociativa
- Inconvenientes: competencia de acceso y tiempos largos de búsqueda
- Protocolos de mapeado completo o limitado

Protocolos de directorio

Protocolos de **directorio de mapeado completo**

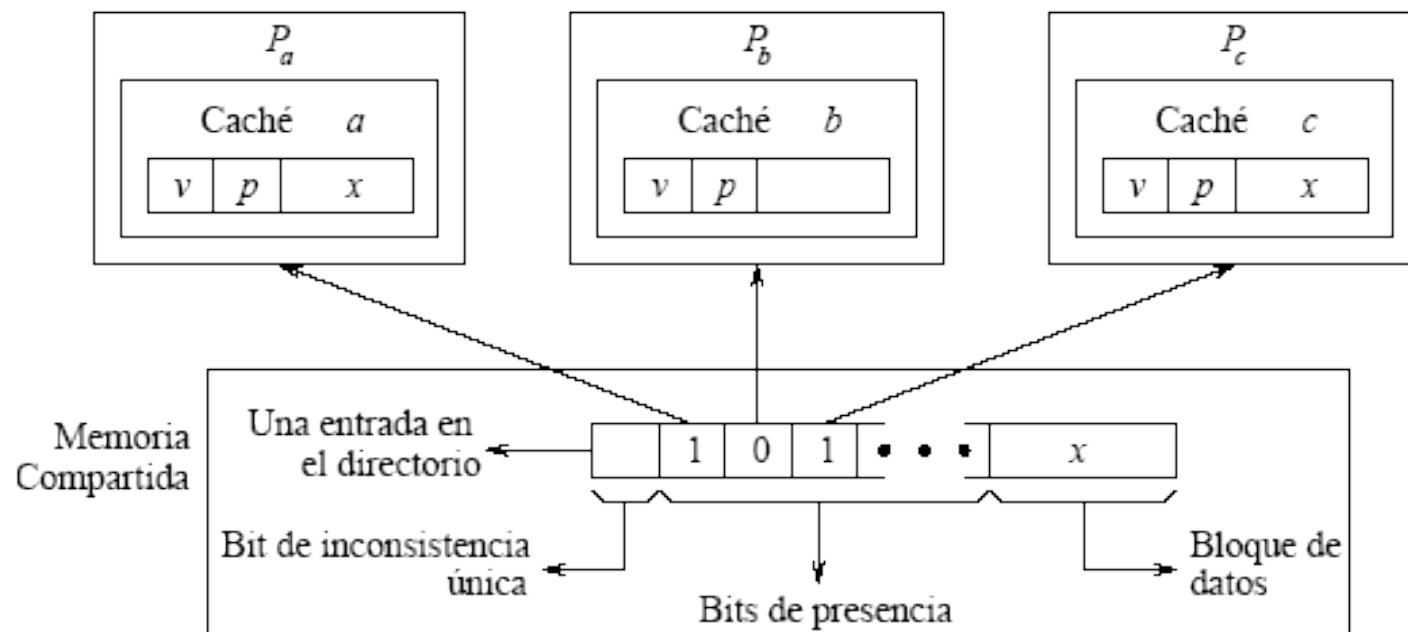
- Protocolo Dir_N NB
- En el directorio: cada entrada de bloque tiene unos bits de presencia por cada caché y un bit de inconsistencia única
 - ✓ Bits de presencia: especifica la presencia en las cachés de copias del bloque de memoria
 - ✓ Bit de inconsistencia única: cuando este bit está activado, sólo uno de los bits presencia está a uno, es decir, sólo existe una caché con la copia de ese bloque o línea, con lo que sólo esa caché tiene permiso para actualizar la línea
- Por cada caché:
 - ✓ Bit de validación (**v**): indica si la copia es válida
 - ✓ Bit de privacidad (**p**): indica si la copia tiene permiso de escritura, es decir, cuando este bit es 1 entonces es la única copia que existe de esta línea en las cachés y por tanto tiene permiso para escribir

Ingeniería de los Computadores

5.3 Protocolos de coherencia

Protocolos de directorio

Protocolos de directorio de mapeado completo



Protocolos de directorio

Protocolos de **directorío de mapeado completo**

- Fallo de lectura:
 - Supongamos que una caché envía una petición de fallo de lectura a la memoria
 - Si el bit de inconsistencia simple está activado, la memoria envía una petición de actualización a la caché que tenga el bit de privado activo, o sea, a la única caché que tenga la línea
 - ✓ La caché devuelve el bloque a la memoria y desactiva su bit de privado
 - ✓ El bit de inconsistencia simple también es desactivado del directorio central
 - ✓ La memoria activa el bit de presencia de la caché correspondiente que solicitó el bloque y envía una copia a dicha caché
 - ✓ Cuando la caché recibe la copia, activa el bit de válido y desactiva el de privado
 - Si el bit de inconsistencia simple está desactivado, trae el bloque de la memoria activando el de válido y desactivando el de privado como en el caso anterior

Protocolos de directorio

Protocolos de **directorío de mapeado completo**

- Fallo de escritura:
 - Supongamos que una caché envía una petición de fallo de escritura a la memoria
 - La memoria envía peticiones de invalidación a todas las cachés que tienen copias de ese bloque poniendo a cero sus bits de presencia
 - Las cachés aludidas invalidan sus líneas poniendo a 0 el bit de línea válida
 - Si hay alguna caché con una copia del bloque con el bit de privado activado, la memoria actualiza su copia
 - Una vez la memoria recibe todos los reconocimientos, activa el bit de presencia en la caché y manda la copia dicha caché
 - El bit de inconsistencia simple se pone a 1 puesto que solo hay una copia
 - La caché que recibe la copia, se modifica y pone los bit de válido y privado a uno

Protocolos de directorio

Protocolos de **directorío de mapeado completo**

- Acierto de escritura:
 - Si el bit de privado es 0, la caché envía una petición de privacidad a la memoria
 - ✓ La memoria invalida todas las demás cachés que tienen una copia del bloque
 - ✓ Se pone el bit de inconsistencia a 1
 - ✓ Cuando la caché recibe el reconocimiento de que las demás cachés han sido invalidadas, modifica su bloque pone el bit de privado a uno
 - Si el bit de privado es 1, entonces escribe sin más puesto que es la única caché que tiene copia de esa línea

Protocolos de directorio

Protocolos de **directorío de mapeado completo**

- Principales problemas:
 - ✓ Tamaño_directorio = $(1 + \text{Núm_cachés}) \cdot \text{Núm_líneas_memoria}$
 - ✓ Crecimiento del directorio proporcional a N^2
 - ✓ Para evitar el crecimiento cuadrático del mapeado completo es posible restringir el número de copias de caché activas de forma simultánea de un bloque. De esta forma se limita el crecimiento del directorio hacia un valor constante

Protocolos de directorio

Protocolos de **directorio de mapeado limitado**

- $Dir_i NB, i < N$
- Idéntico al protocolo de mapeado completo pero reduciendo el número de copias de caché activas simultáneas de un bloque:
 - ✓ Cada entrada de bloque tiene un **bit de presencia** en cada caché y un **bit de inconsistencia única (BIU)**
 - ✓ Cada caché tiene un bit de validación (**v**) y un bit de privacidad (**p**)
- Punteros limitados
 - ✓ Llamada de desalojo: la memoria invalida una de las cachés que utilizaba el puntero y le asigna ese puntero libre a la nueva caché
 - ✓ Tamaño punteros: $\log_2 N$ bits
- Tamaño del directorio = $(1 + \text{Punteros} \cdot \log_2 N) \cdot \text{Bloques_memoria}$
- Crecimiento proporcional a $N \cdot \log_2 N$
- Protocolo “escalable”

Protocolos de directorio

Protocolos de **directorío de distribuido**

- Tabla distribuida entre cachés
- Cada tabla local almacena el estado de la caché y de las copias de los bloques
- Protocolos de mapeado distribuidos de directorio:
 - ✓ **Encadenados**: listas de cachés encadenadas
 - ✓ **Jerárquicos**: división del directorio entre grupos de procesadores interconectados (*clusters*)

Protocolos de directorio

Protocolos de **directorío de distribuido encadenado**

Lista simple o doble enlazada de punteros entre distintas cachés con copia de un mismo bloque:

- ✓ Cada entrada del directorio apunta a una caché con copia del bloque y esta caché a su vez tiene un puntero que apunta a otra, etc.
- ✓ Una entrada en el directorio contiene un único puntero que apunta a la cabeza de la lista

