

# Introducción a las Arquitecturas Paralelas



Arquitectura de Computadoras II

Fac. Cs. Exactas

UNCPBA

Prof. Marcelo Tosini

2015

# Procesamiento Paralelo

---

Uso de muchas unidades de proceso independientes para ejecutar distintas partes de una tarea en simultáneo

**Principal objetivo:** Aumento del RENDIMIENTO. Aumento de la capacidad para resolver problemas computacionales grandes

## ¿Cómo?

- División del trabajo en tareas mas pequeñas e independientes
- Asignación de las tareas a distintas unidades de proceso
- Resolución de las tareas en simultaneo.

## Problemas:

- Sincronización de las tareas.
- control de ejecución simultanea
- conflictos debidos a dependencias

# Procesamiento Paralelo

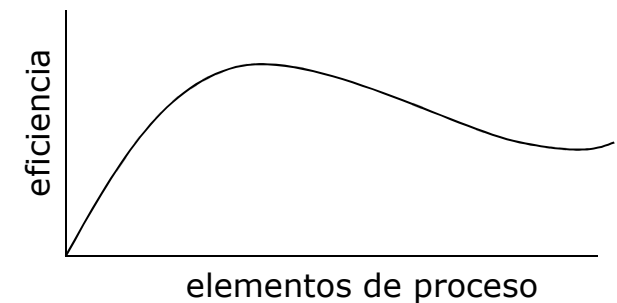
---

## Limitaciones:

En algunos problemas el incremento del número de procesadores no mejora el rendimiento global, incluso empeora la eficiencia del sistema.

La eficiencia se mejora cuando:

- se logra un balance de carga entre procesadores: igual número de tareas de igual tamaño
- Se minimiza la interacción entre tareas: se minimiza la comunicación o, al menos, se mejoran los canales de comunicación



# Sistema paralelo

---

**Conjunto de elementos de proceso que, operando juntos, permiten resolver problemas computacionales complejos de forma eficiente**

## **Características de un sistema paralelo:**

- Cantidad y potencia de los elementos de proceso
- Tipo y Tamaño de la memoria
- Forma de comunicación entre los elementos de proceso
- Rendimiento
- Escalabilidad del sistema
- Recursos de potencia requeridos

# Niveles de paralelismo

---

El paralelismo puede estudiarse a varios niveles:

- **Trabajo:** Dos programas distintos pueden ejecutarse en paralelo
- **Tarea:** En este nivel se consideran varias tareas independientes entre si formando parte de un programa determinado. Es posible la interacción de las tareas
- **Proceso:** Varios procesos componen una tarea. Son bloques con funcionalidad bien definida.
- **Variable:** El paralelismo puede darse a nivel de variables ya que varias instrucciones pueden ser ejecutadas en paralelo siendo el punto de conflicto las variables en común
- **Bit:** Todos los computadores usan paralelismo a nivel de bit

# Arquitecturas de procesadores

---

## Complejidad del procesador:

Arquitectura característica y estructura de cada procesador del sistema.

Íntimamente ligado con la funcionalidad  
(variedad de operaciones y cantidad de instrucciones)

Arreglos sistólicos —————> homogéneos —————> complejidad baja  
MIMD —————> Heterogéneos —————> complejidad alta

# Arquitecturas de procesadores

---

## Modo de operación:

Forma de controlar la secuencia de operaciones a realizar para llevar adelante la ejecución de una tarea

### **Control flow**

Las instrucciones se ejecutan en el orden dispuesto por el algoritmo

### **Data flow**

Las operaciones se realizan según la disponibilidad de datos

### **Demand flow**

Los resultados parciales se calculan por demanda, o sea cuando se los necesita

# Arquitecturas de procesadores

---

## Organización de la memoria:

Tipo de memoria utilizada en el sistema

### **Direccionable**

Accedida por referencias a los datos

### **Asociativa**

Accedida por contenido

### **Interconectada**

Accedida por cualidades de los datos

(redes neuronales)



# Arquitecturas de procesadores

---

## Red de interconexión:

Conexión de hardware entre procesadores y entre procesadores y memorias

La arquitectura de conexión debe ajustarse lo mejor posible a la topología de un algoritmo para mejorar la performance

# Arquitecturas de procesadores

---

Número de procesadores y tamaño de la memoria:

Potencia de cálculo del sistema y capacidad de almacenamiento de datos del mismo

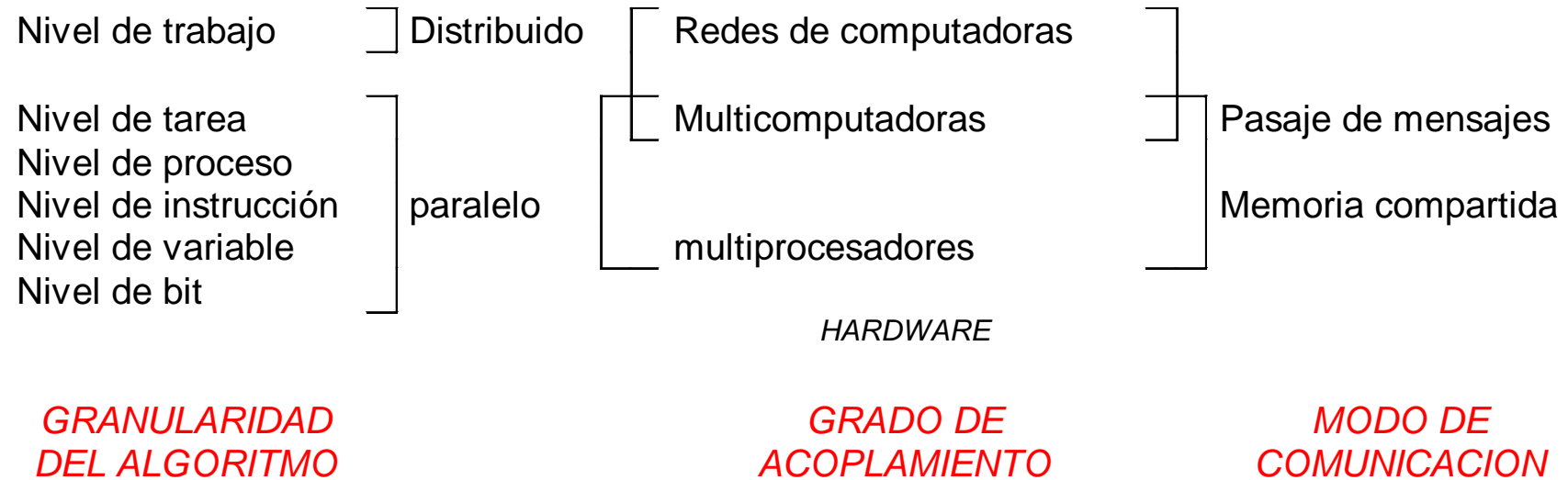
Clasificación:

**Sistemas grandes:** más de 1000 procesadores

**Sistemas medios:** de 100 a 1000 procesadores

**Sistemas chicos:** hasta 100 procesadores

# Organización de las arquitecturas

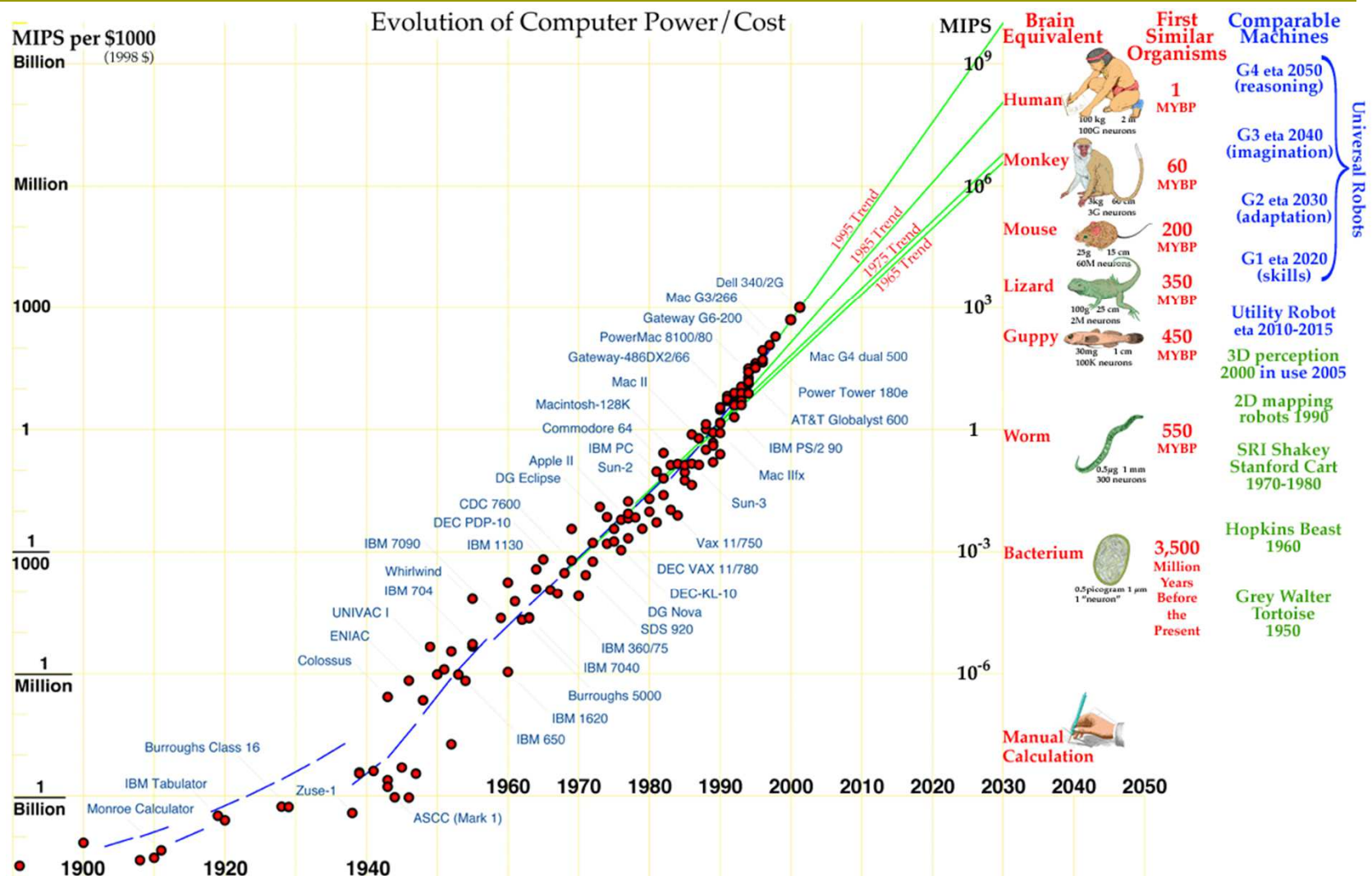


# Ámbitos de uso de la computación paralela

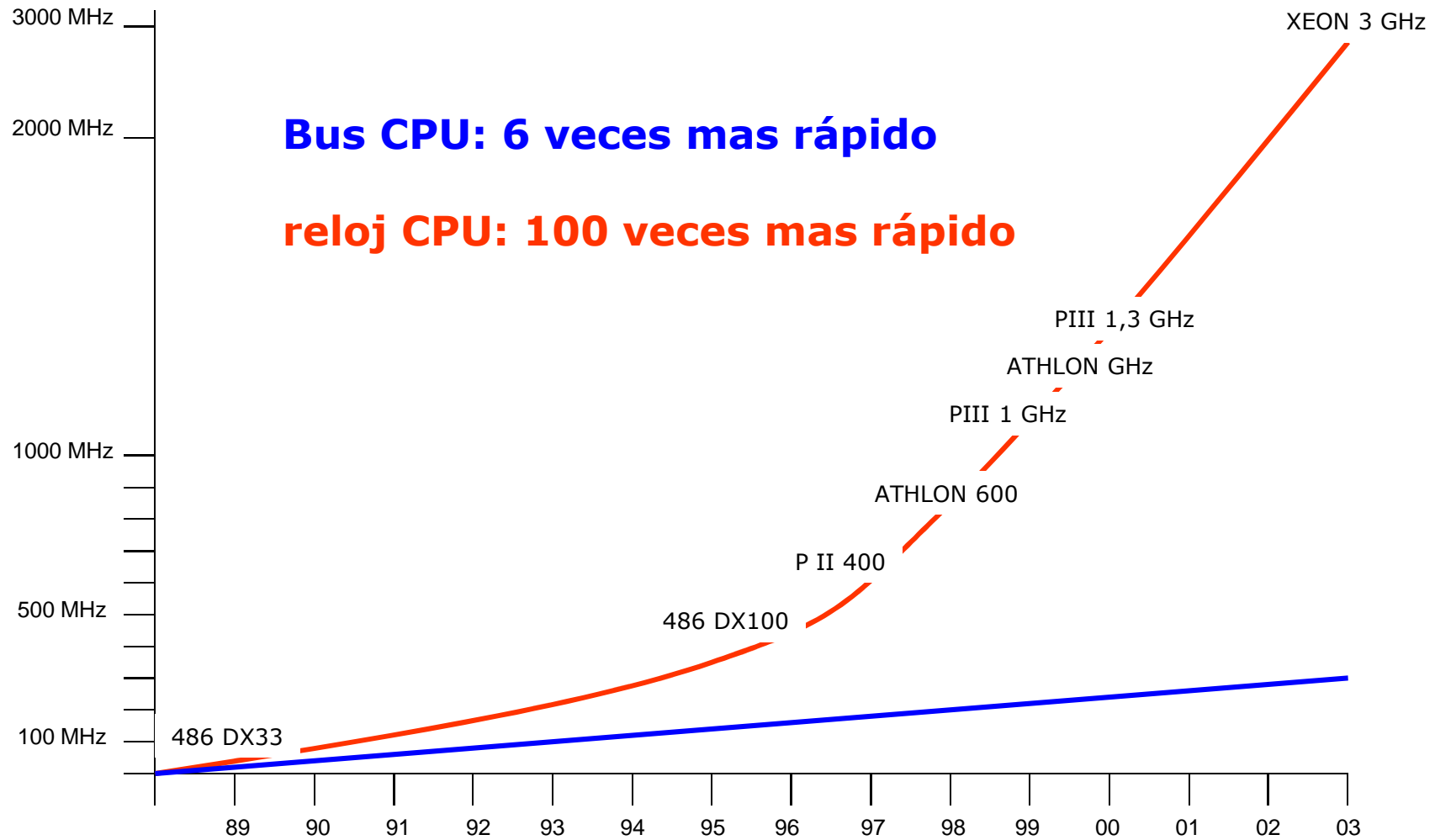
---

- Simulación de modelos complejos
- Diseño y automatización de proyectos de ingeniería
- Exploración petrolera y minera
- Medicina
- Área militar
- Cine: efectos visuales, animación 3D
- Realidad Virtual
- Comercio electrónico
- Mega bases de datos (google, youtube, rapidshare)

# Evolución del rendimiento



# Incremento de velocidad



# Límites tecnológicos

El *feature size* ( $d$ ) determina el tamaño de las compuertas en la tecnología CMOS de manera que:

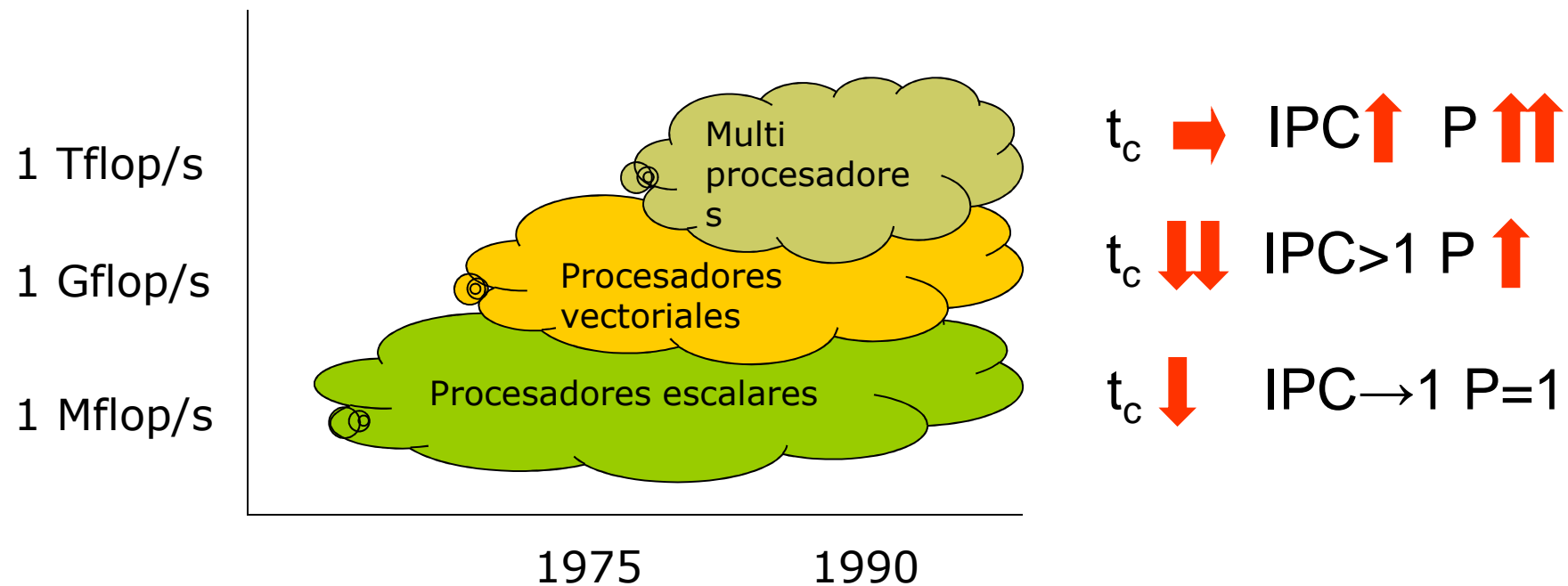
- Un aumento de la velocidad de reloj es proporcional a  $\lambda=1/d$
- Un aumento del número de transistores es proporcional a  $\lambda^2$

## Hasta cuanto puede disminuir $d$ ??

| característica / año             | 1997    | 1999    | 2001    | 2003    | 2006    | 2009    | 2012    |
|----------------------------------|---------|---------|---------|---------|---------|---------|---------|
| Feature Size ( $\mu\text{m}$ )   | 0.25    | 0.18    | 0.15    | 0.13    | 0.10    | 0.07    | 0.05    |
| Voltaje                          | 1.8-2.5 | 1.5-1.8 | 1.2-1.5 | 1.2-1.5 | 0.9-1.2 | 0.6-0.9 | 0.5-0.6 |
| Nº transistores                  | 11M     | 21M     | 40M     | 76M     | 200M    | 520M    | 1.4B    |
| DRAM bits/chip                   | 167M    | 1.07G   | 1.7G    | 4.29G   | 17.2G   | 68.7G   | 275G    |
| Tamaño Die ( $\text{mm}^2$ )     | 300     | 340     | 385     | 430     | 520     | 620     | 750     |
| Frecuencia local de reloj (MHz)  | 750     | 1250    | 1500    | 2100    | 3500    | 6000    | 10000   |
| Frecuencia global de reloj (MHz) | 750     | 1200    | 1400    | 1600    | 2000    | 2500    | 3000    |

# Evolución de las arquitecturas

$$T = \frac{N}{P} * \frac{1}{IPC} * t_c$$





# Medidas de performance

---

## Tiempo promedio de ejecución

- Instrucciones por segundo  
Útil en SISD y en MIMD, pero no en SIMD
- Operaciones por segundo  
No considera tamaño de palabra
- Punto flotante por segundo  
No es útil en compiladores y en AI
- Inferencias por segundo  
Útil en inteligencia artificial

# Medidas de performance

---

Speedup ( $S_p$  - para P procesadores)

$$S_p = \frac{T_1}{T_p}$$

Promedio entre el tiempo de proceso secuencial y paralelo en P procesadores

$T_1$  : tiempo en 1 procesador

$T_p$  : tiempo en P procesadores

$$S_p < P$$

# Medidas de performance

---

Eficiencia ( $E_p$  - para P procesadores)

$$E_p = \frac{S_p}{P}$$

$$0 < E_p < 1$$

Cociente entre  $S_p$  y  $P$ .

Medida de la relación costo/efectividad de la computación

$P$  : número de procesadores

$S_p$  : Speedup con  $P$  procesadores

# Medidas de performance

---

Redundancia ( $R_p$  - para P procesadores)

$$R_p = \frac{O_p}{O_1}$$

$$R_p > 1$$

Promedio entre el número total de operaciones ejecutadas con P proc. y el número de operaciones necesarias en 1 procesador

$O_p$  : número de operaciones en P procesadores

$O_1$  : Número de operaciones en un procesador

# Medidas de performance

---

Utilización ( $U_p$  - para P procesadores)

$$U_p = R_p * E_p = \frac{O_p}{P \cdot T_p}$$

Número de operaciones totales ejecutadas con P procesadores ponderada por la eficiencia de trabajo en esos P procesadores

$O_p$  : número de operaciones en P procesadores

$$U_p < 1$$

# Medidas de performance

---

Calidad del paralelismo ( $Q_p$  - para P procesadores)

$$Q_p = \frac{S_p * E_p}{R_p}$$

La calidad de paralelismo es proporcional al Spedup y a la Eficiencia.

La calidad de paralelismo decrece al aumentar la Redundancia

$$Q_p < 1$$

# Límites de la computación paralela

---

La idea es modelar lo más aproximadamente la operación en un entorno multiprocesador

Premisas:

Un programa paralelo es una serie de instancias de tareas de sincronización seguidas de cálculos reales (programa) distribuidos entre los procesadores.

Debido al overhead el tiempo total de ejecución de las tareas distribuidas en los procesadores es mayor que si se hubiese ejecutado en un único procesador

# Límites de la computación paralela

---

Variables de cálculo:

- $t_s$  = tiempo de sincronización
- $t$  = granularidad de tarea (tiempo de ejecución promedio de las tareas)
- $t_o$  = overhead de tareas causado por la ejecución paralela
- $N$  = cantidad de tareas entre puntos de sincronización
- $P$  = número de procesadores



# Límites de la computación paralela

---

El tiempo de ejecución secuencial de  $N$  tareas de tiempo  $t$  será

$$T_1 = N.t$$

En un ambiente paralelo cada tarea requiere  $(t + t_o)$  unidades de tiempo  
Si hay  $N$  tareas en  $P$  procesadores, entonces el número de pasos paralelos será  $\lceil N/P \rceil$ . Entonces el tiempo de ejecución paralelo será:

$$T_{N,P} = t_s + \lceil N/P \rceil . (t + t_o)$$

Si  $N$  es múltiplo de  $P$  no hay penalizaciones de balance de carga al final de cada computación

# Límites de la computación paralela

---

El Speedup del sistema será:

$$\begin{aligned} S_{N,p} &= \frac{T_1}{T_{N,p}} = \frac{N \cdot t}{ts + \lceil N/P \rceil \cdot (t + t_o)} \\ &= \frac{1}{ts/(N \cdot t) + (1/N) \lceil N/P \rceil \cdot (1 + t_o/t)} \end{aligned}$$

# Límites de la computación paralela

---

La eficiencia del sistema será:

$$E_{N,p} = \frac{S_{N,p}}{P} = \frac{N \cdot t}{ts + \lceil N/P \rceil \cdot (t + t_o)} / P$$
$$= \frac{1}{ts/(N \cdot t) + (1/N) \lceil N/P \rceil \cdot (1 + t_o/t)} / P$$

# Límites de la computación paralela

---

| Métrica   | $P \rightarrow \infty$ , N fijo | $N \rightarrow \infty$ , P fijo |
|-----------|---------------------------------|---------------------------------|
| $S_{N,p}$ | $N/(1 + (t_s + t_o)/t)$         | $P/(1 + t_o/t)$                 |
| $E_{N,p}$ | 0                               | $1/(1 + t_o/t)$                 |

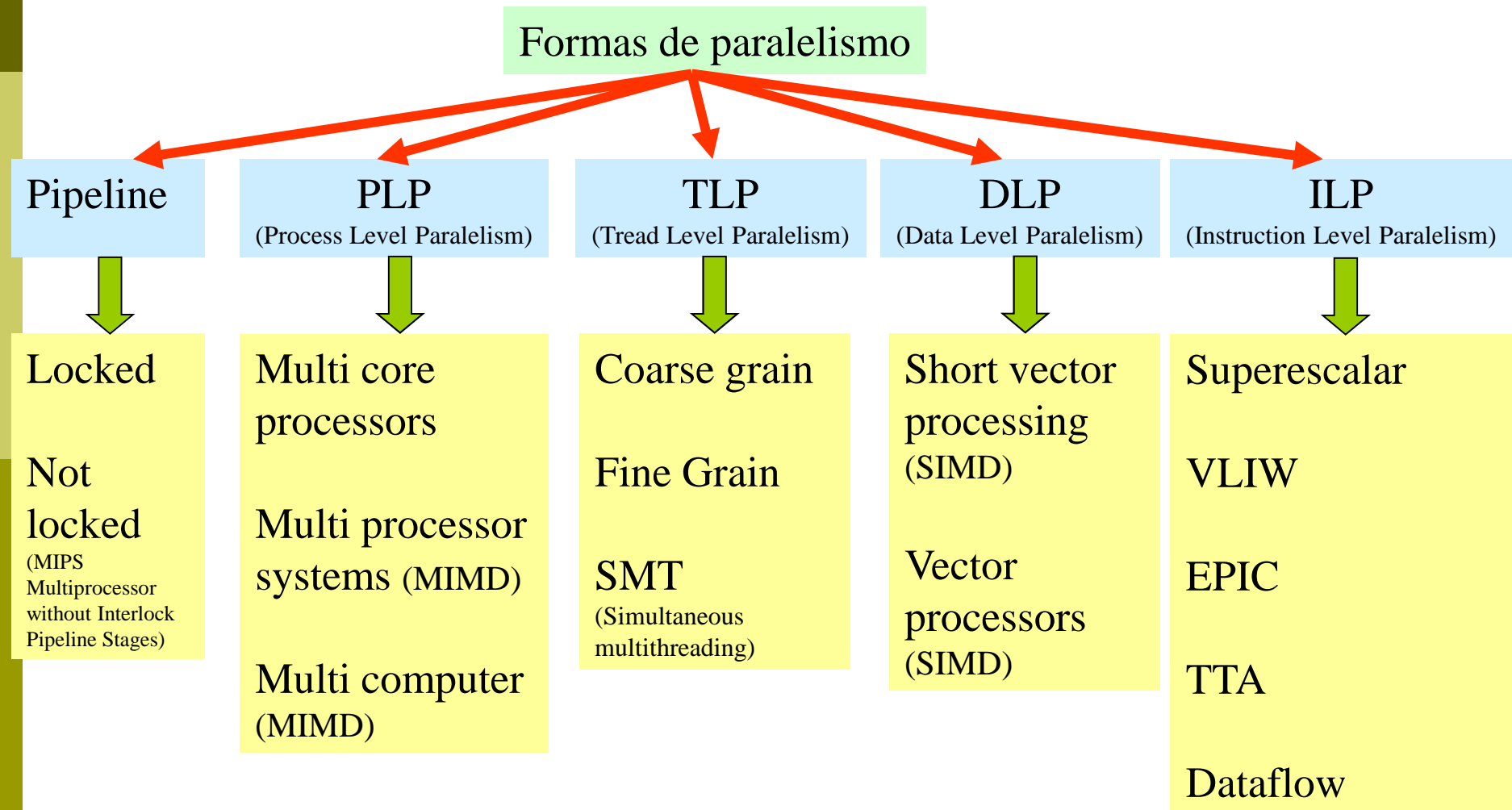
# Límites de la computación paralela

---

## Conclusiones:

- La primera columna muestra que el **speedup** resultante de incrementar el número de procesadores está **limitado por el número de tareas  $N$** , mientras que la eficiencia tiende a 0.
- La segunda columna muestra que un **Speedup igual a la cantidad de procesadores** puede ser logrado realizando un gran **número de tareas**, siempre y cuando el overhead sea ínfimo respecto a la granularidad de tareas

# Clasificación de las arquitecturas de computadoras



# PLP - Process level paralelism

---

Distintos procesos se ejecutan en diferentes procesadores paralelos o en diferentes cores de un mismo procesador

## Clasificados de acuerdo al modelo de Flynn

Modelo que permite clasificar a todas las computadoras basándose en el estudio del paralelismo de los flujos de instrucciones y datos exigidos por las instrucciones en los componentes más restringidos de la máquina

- Flujo único de instrucciones, flujo único de datos.(SISD)
- Flujo único de instrucciones, flujo múltiple de datos.(SIMD)
- Flujo múltiple de instrucciones, flujo único de datos.(MISD)
- Flujo múltiple de instrucciones, flujo múltiple de datos.(MIMD)

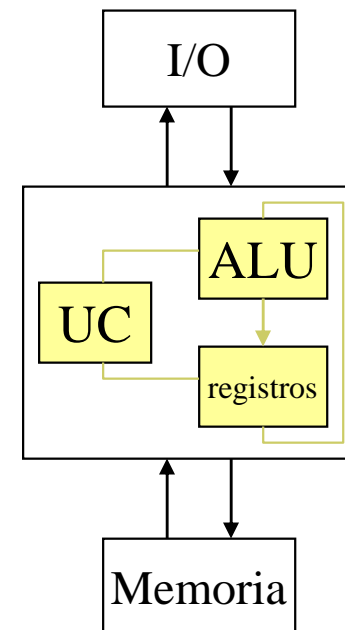
# SISD. Flujo único de instrucciones y datos

---

La CPU controla todas las operaciones que se realizan en la máquina extrayendo secuencialmente las instrucciones de programa desde la memoria.

## CPU:

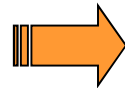
- **Unidad de control:** ejecuta una a una las instrucciones de programa
- **Unidad lógico/aritmética:** realiza las operaciones sobre los datos
- **Registros internos:** se almacenan datos parciales y direcciones.





# SIMD. Flujo único de instrucciones, flujo múltiple de datos

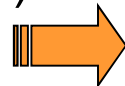
```
for (i = 1 ; i < MaxElem ; i ++)  
    A[i] = 2 * a[i-1];
```



```
A[1] = 2 * a[0];  
A[2] = 2 * a[1];  
.  
.  
A[n-1] = 2 * a[n-2];  
A[n] = 2 * a[n-1];
```

Distintas operaciones  
sobre  
distintos datos

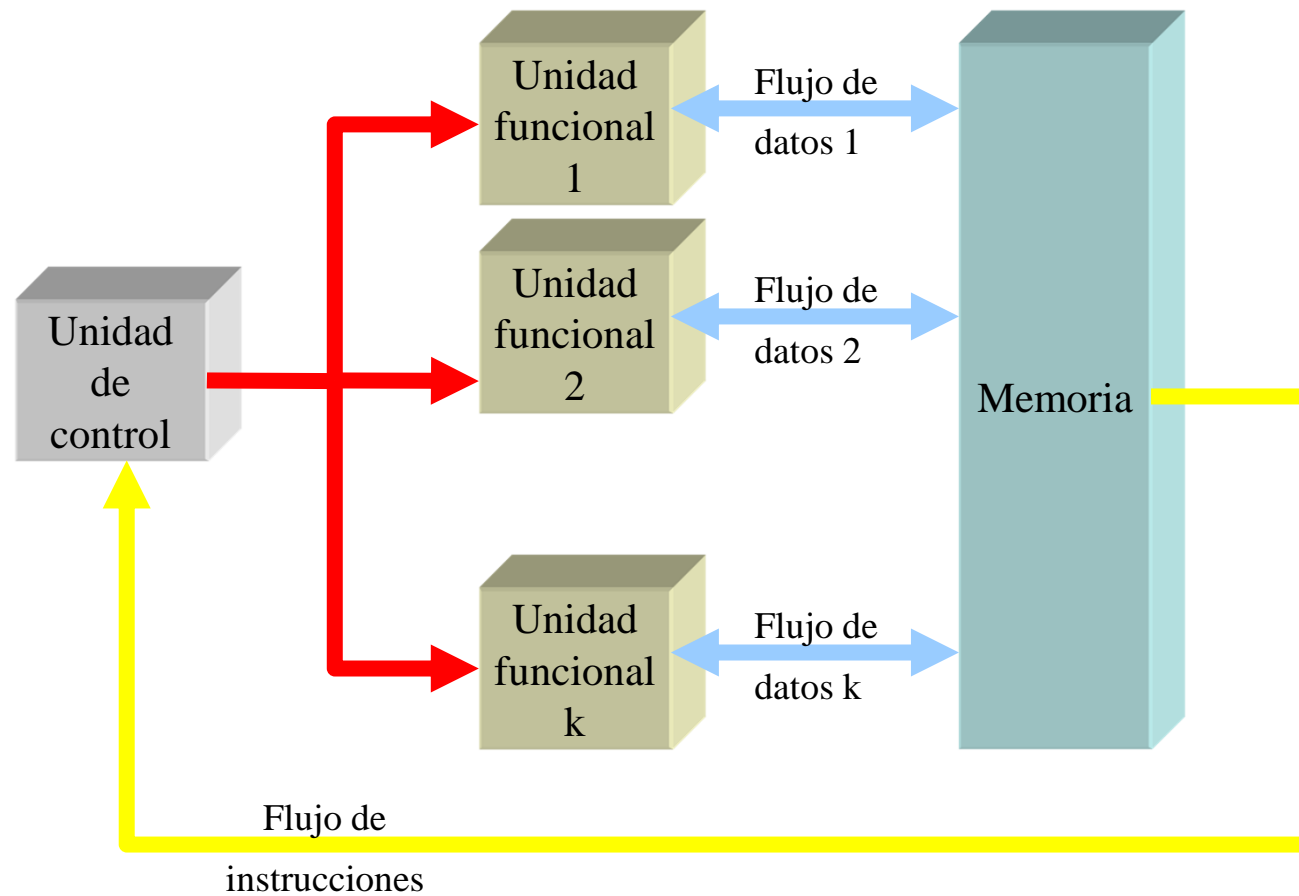
```
for (i = 1 ; i < MaxElem ; i ++)  
    A[i] = 2 * b[i];
```



```
A[1] = 2 * b[1];  
A[2] = 2 * b[2];  
.  
.  
A[n-1] = 2 * b[n-1];  
A[n] = 2 * b[n];
```

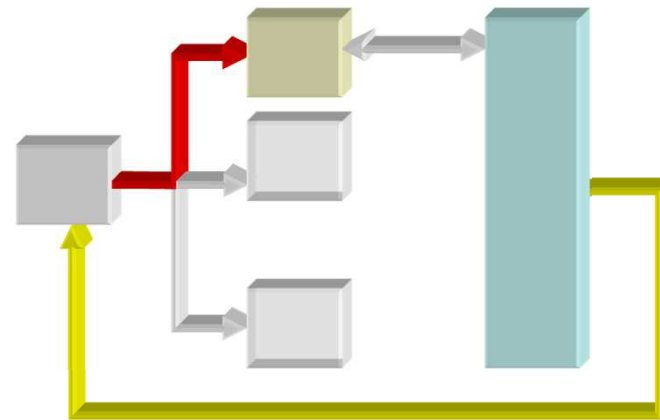
Mismas operaciones  
sobre  
distintos datos

# Arquitectura SIMD



# Arquitectura SIMD

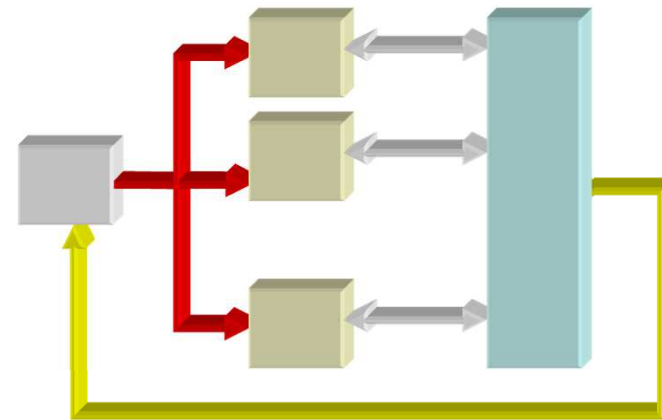
```
for (i = 1 ; i < MaxElem ; i ++)  
    A[i] = 2 * a[i-1];
```



|                    |               |               |               |               |               |
|--------------------|---------------|---------------|---------------|---------------|---------------|
| Unidad funcional 1 | $A[1]=2*A[0]$ | $A[2]=2*A[1]$ | $A[3]=2*A[2]$ | $A[4]=2*A[3]$ | $A[5]=2*A[4]$ |
| Unidad funcional 2 | idle          | idle          | idle          | idle          | idle          |
| . . . . .          |               |               |               |               |               |
| Unidad funcional k | idle          | idle          | idle          | idle          | idle          |
| <b>Ciclo</b>       | <b>0</b>      | <b>1</b>      | <b>2</b>      | <b>3</b>      | <b>4</b>      |

# Arquitectura SIMD

```
for (i = 1 ; i < MaxElem ; i ++)  
    A[i] = 2 * a[i];
```



|                    |               |                   |                     |                     |               |
|--------------------|---------------|-------------------|---------------------|---------------------|---------------|
| Unidad funcional 1 | $A[1]=2*A[1]$ | $A[k+1]=2*A[k+1]$ | $A[2k+1]=2*A[2k+1]$ | $A[3k+1]=2*A[3k+1]$ | $A[n]=2*A[n]$ |
| Unidad funcional 2 | $A[2]=2*A[2]$ | $A[k+2]=2*A[k+2]$ | $A[2k+2]=2*A[2k+2]$ | $A[3k+2]=2*A[3k+2]$ | idle          |
| . . . . .          |               |                   |                     |                     |               |
| Unidad funcional k | $A[k]=2*A[k]$ | $A[2k]=2*A[2k]$   | $A[3k]=2*A[3k]$     | $A[4k]=2*A[4k]$     | idle          |
| <b>Ciclo</b>       | <b>0</b>      | <b>1</b>          | <b>2</b>            | <b>3</b>            | <b>4</b>      |

**MISD.** Flujo múltiple de instrucciones, flujo único de datos

---

**Conceptualmente, varias instrucciones ejecutándose paralelamente sobre un único dato.**

Arquitecturas desacopladas y los arreglos sistólicos

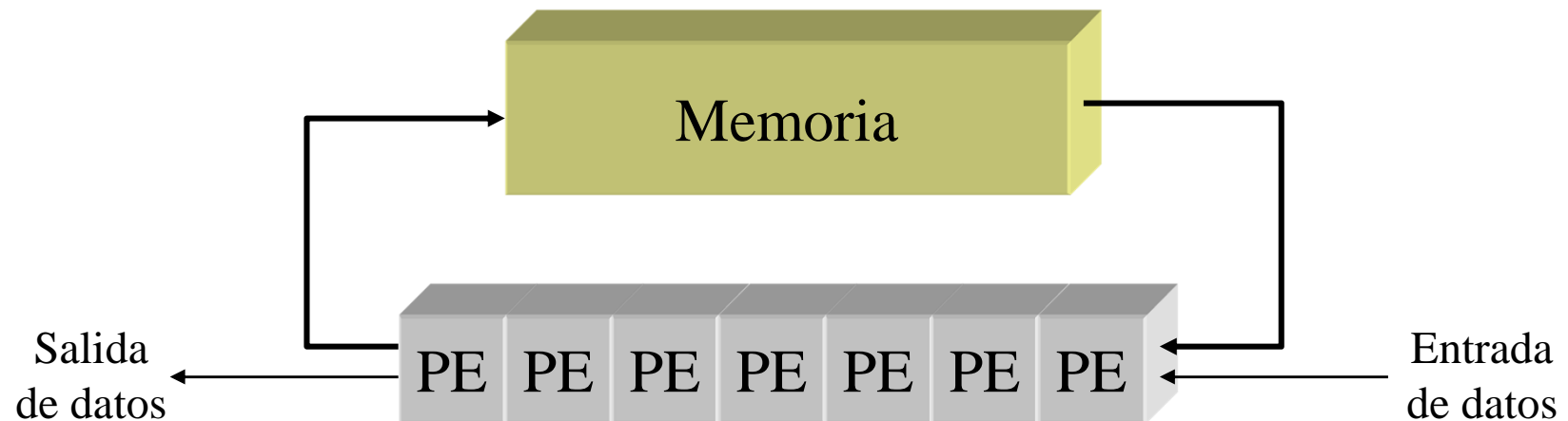
Funcionan con el principio de ‘bombear’ los datos a través de una hilera de procesadores escalares donde en cada uno de ellos se realizan paralelamente operaciones sobre distintos datos.

Desde el punto de vista de cada dato, éste pasa de un procesador al siguiente para transformarse de acuerdo a la operación que realice cada procesador.

# Arquitecturas clásicas MISD

---

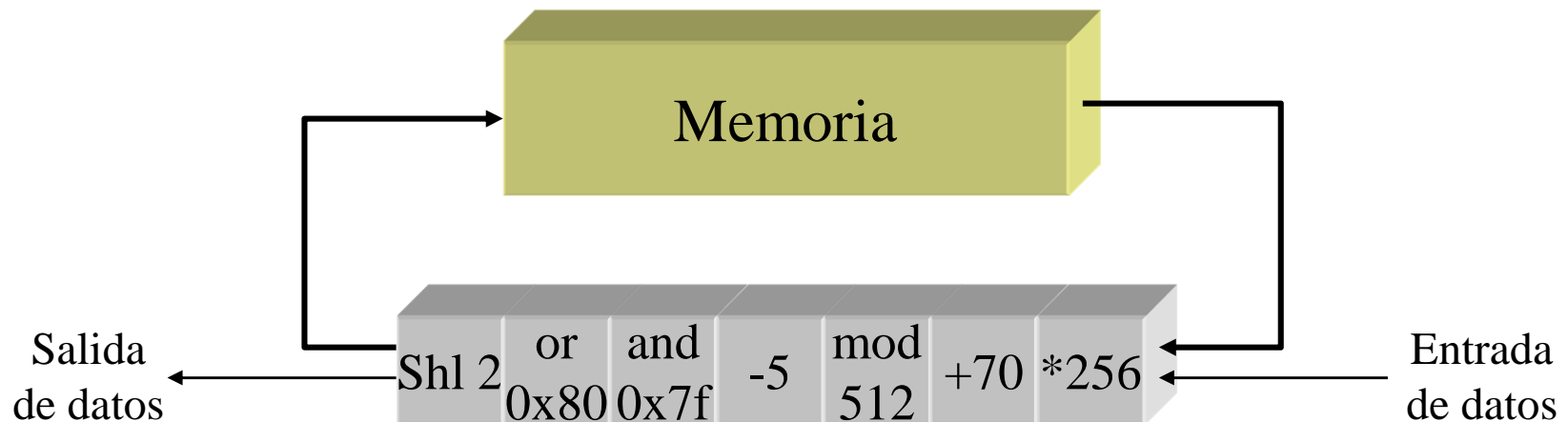
- La información circula entre las celdas como en un pipeline
- La comunicación con el exterior se produce en las celdas frontera



# Arquitecturas clásicas MISD

## Ejemplo

$$M[i] = (((M[i] * 256 + 70) \bmod 512 - 5) \text{ and } 0x7f) \text{ or } 0x80) \text{ shl } 2$$



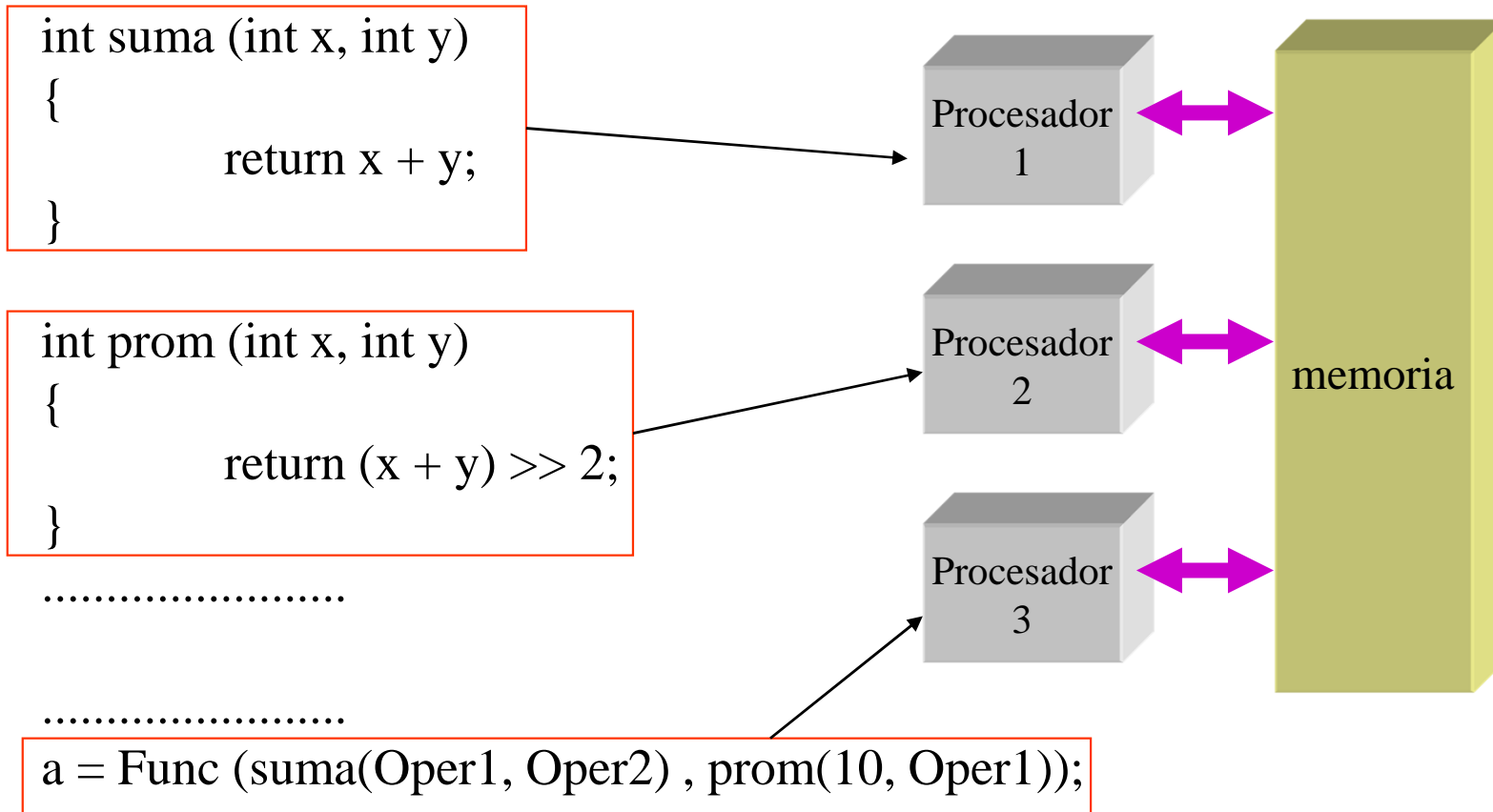
## MIMD. Flujo múltiple de instrucciones, flujo múltiple de datos

---

- Es la mejor estrategia de diseño orientada a obtener el más alto rendimiento y la mejor relación costo/rendimiento.
- Idea general: conectar varios procesadores para obtener un rendimiento global lo más cercano a la suma de rendimientos de cada procesador por separado.
- La filosofía de trabajo plantea la división de un problema en varias tareas independientes y asignar a cada procesador la resolución de cada una de estas tareas.



# MIMD. Flujo múltiple de instrucciones, flujo múltiple de datos



# TLP – Thread level parallelism

---

En TLP las unidades de ejecución de un procesador se comparten entre los threads independientes de un proceso (o threads de diferentes procesos)

**COARSE GRAIN:** En coarse grain multi-threading los threads son desalojados del procesador con baja frecuencia, usualmente cuando el thread realiza alguna I/O, o ante un fallo de cache.

**FINE GRAIN:** En fine grain multi-threading el thread en ejecución es cambiado (thread swaping) en cada ciclo de reloj

**SMT:** Simultaneous multi-threading es similar a fine grain, pero permite ejecutar múltiples threads en cada ciclo de reloj.

SMT permite concurrencia física, a diferencia de los anteriores que solo manejan Concurrencia virtual (multiplexado por división de tiempo)

# TLP – Thread level paralelism

|   |   |   |   |
|---|---|---|---|
| A | A | A | A |
| A | A | A |   |
| A |   |   |   |
| A | A |   |   |
| B | B | B |   |
| B |   |   |   |
| B | B |   |   |
| C | C | C | C |
| C | C | C |   |
| C |   |   |   |
| D | D |   |   |
| D | D | D | D |
| D | D | D |   |
| D |   |   |   |

Coarse grain

|   |   |   |   |
|---|---|---|---|
| A | A | A | A |
| B | B | B |   |
| C | C | C | C |
| D | D |   |   |
| A | A | A |   |
| B |   |   |   |
| C | C | C |   |
| D | D | D | D |
| A |   |   |   |
| B | B |   |   |
| C |   |   |   |
| D | D | D |   |
| A | A |   |   |
| D |   |   |   |

Fine grain

|   |   |   |   |
|---|---|---|---|
| A | A | A | A |
| A | B | B | C |
| C | C | C | D |
| A | D | B | D |
| B | A | A | C |
| C | C | C |   |
| D | D | D | B |
| D | B | A |   |
| A | D | D | D |

SMT

# DLP – Data level paralelism

---

- La operación se aplica a varios ítems de dato en lugar de uno
- Implementado con rutas de datos divisibles
- Por ejemplo: una ruta de 64 bits permite realizar 1 operación de 64 bits; 2 de 32 bits; 4 de 16 bits; etc.

## Tipos:

- **Short vector processing:** uso de operadores de M bits para realizar N operaciones de M/N bits.
- **Vector processors:** la ruta de datos se multiplexa en tiempo entre los elementos del vector de operandos. No ahorra tiempo de proceso, solo permite disminuir el tamaño del código por el uso de instrucciones vectoriales.

# ILP – instruction level paralelism

---

## Ejecución paralela e instrucciones completas u operaciones

Aproximaciones: Superescalar

VLIW (Very Long Instruction Word)

EPIC (Explicit parallel Instruction Computer)

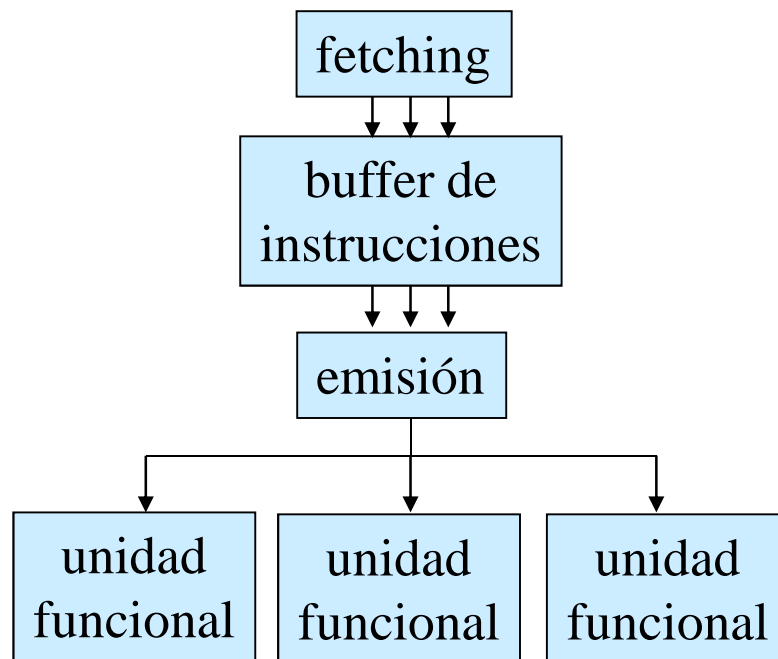
TTA (Transport Triggered Architecture)

DataFlow

Si bien todas se basan en la paralelización de instrucciones para su ejecución difieren en la forma de emisión de las mismas

# ILP - Superescalar

Los procesadores superescalares leen varias instrucciones a la vez en su cola de instrucciones y dinámicamente emiten cierto número de ellas en cada ciclo de reloj. El número y tipo de instrucciones emitidas depende de cada arquitectura.



## Ventaja:

- Ejecución masiva en paralelo

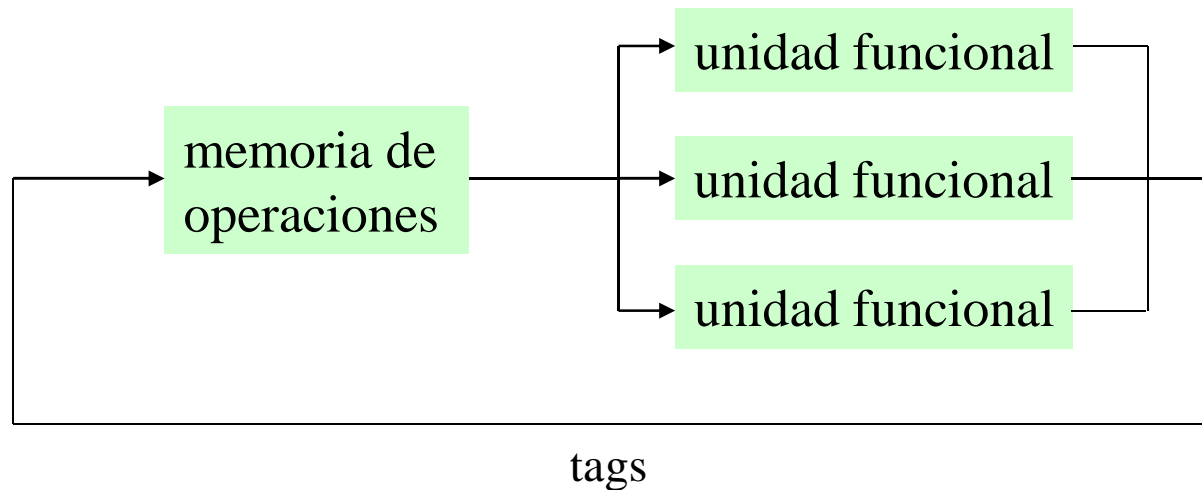
## Desventajas:

- Perdida de orden secuencial
- Problemas de dependencias
- Problemas con los saltos

# ILP - Dataflow

## Controlada por el flujo de los datos en lugar del orden de las instrucciones

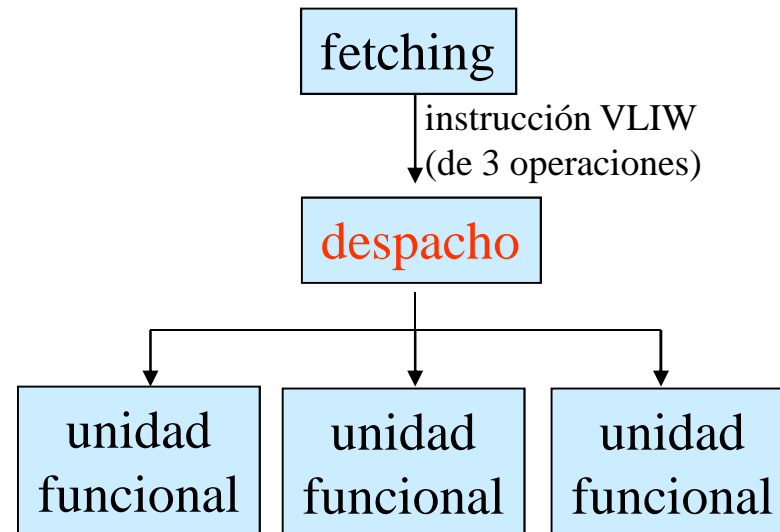
- Las operaciones se almacenan en un buffer a la espera de los datos para operar
- Los resultados viajan en paquetes (tags) que contienen el valor y la lista de operaciones destino (que usan ese valor como operando)
- Cuando una operación tiene todos sus operandos, se dispara y ejecuta.



# ILP - VLIW

## Ejecuta grupos de operaciones empaquetadas en instrucciones compuestas

- Las instrucciones dentro de cada paquete son independientes entre si.
- Todas las instrucciones de un paquete se ejecutan en paralelo y las más rápidas deben esperar la finalización de las más lentas.
- La selección de instrucciones de cada paquete la realiza el compilador.



### Desventajas:

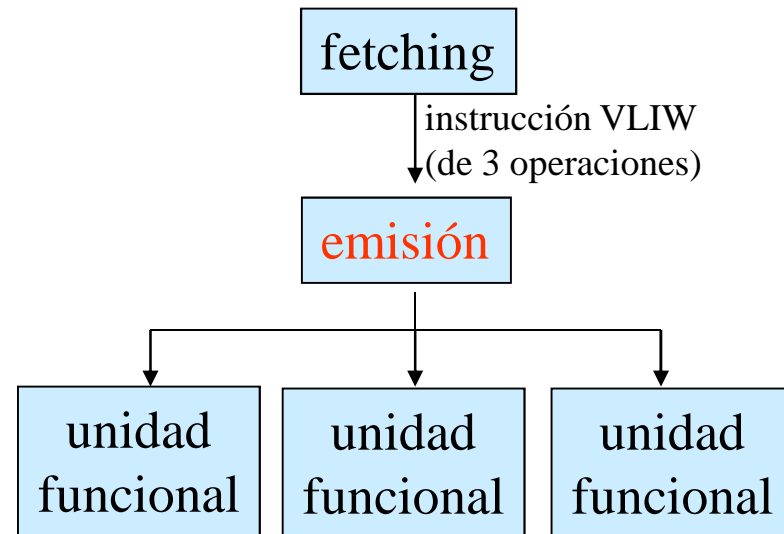
- Mayor ancho del bus de datos desde memoria de instrucciones.
- Banco de registros con varios puertos de lectura/escritura.
- Desperdicio de espacio de memoria por instrucciones VLIW incompletas debido a dependencias.



# ILP - EPIC

## Mejora de VLIW para evitar el desperdicio de espacio debido a dependencias

- Los paquetes siempre están completos (no hay NOOP's)
- Las operaciones dentro de un paquete tienen información adicional de dependencia entre ellas
- Hay una unidad de emisión que decide que instrucciones se emiten y a que unidades



### Desventajas:

- Mayor ancho del bus de datos desde memoria de instrucciones.
- Banco de registros con varios puertos de lectura/escritura.
- La planificación se realiza en el compilador (como en VLIW)

# ILP - TTA

---

La idea básica de TTA es permitir a los programas el control total de los caminos internos de movimiento de datos dentro del procesador.

La arquitectura se compone básicamente de unidades funcionales, buses y registros.

Las entradas de las unidades funcionales tienen puertos disparables (triggering ports) que permiten activar una operación determinada cuando todos los puertos tienen datos válidos para la instrucción a realizar.

Una palabra de instrucción TTA esta compuesta de múltiples slots, uno por bus.

TTA es similar a VLIW pero con mayor control sobre el hardware.

# ILP - TTA

ejemplo:

En RISC

add r3, r1, r2

En TTA

r1 -> ALU.operand1  
r2 -> ALU.add.trigger  
ALU.result -> r3

