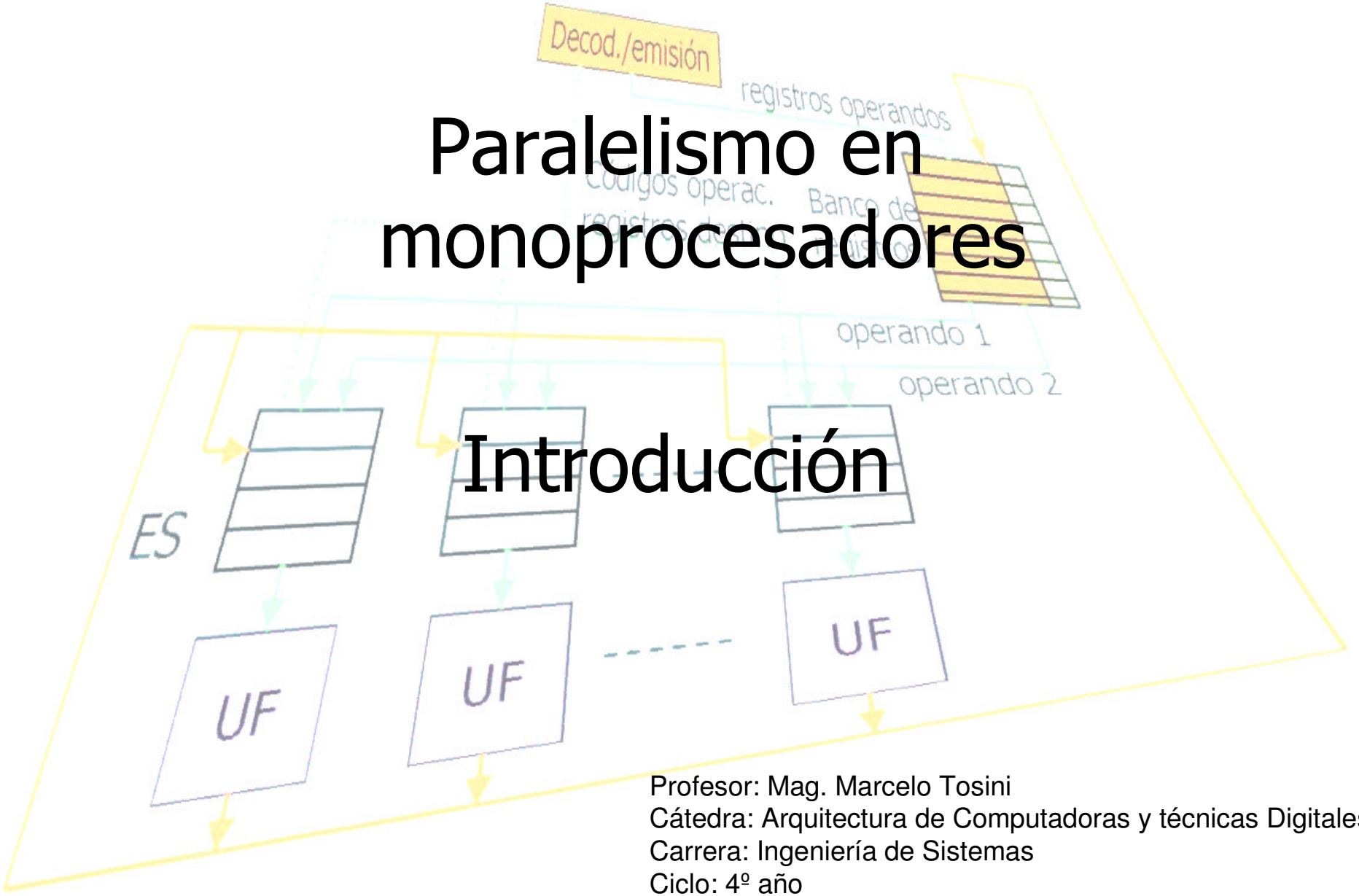
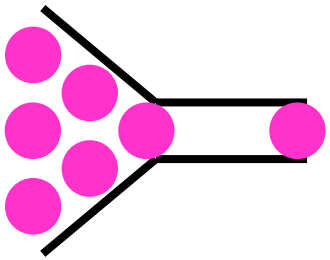


Paralelismo en monoprocesadores

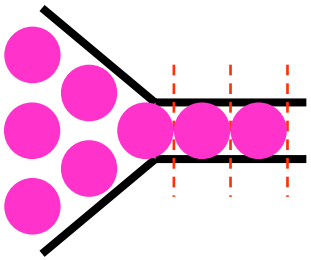
Introducción

Profesor: Mag. Marcelo Tosini
Cátedra: Arquitectura de Computadoras y técnicas Digitales
Carrera: Ingeniería de Sistemas
Ciclo: 4º año

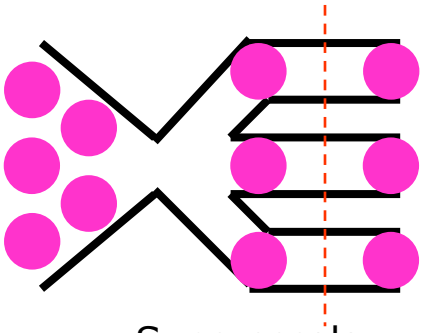
Técnicas para mejorar el rendimiento de un procesador



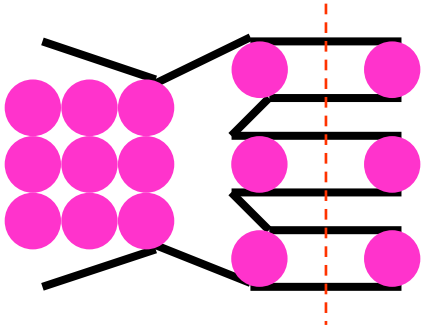
Sin mejoras



Segmentado
(Escalar)

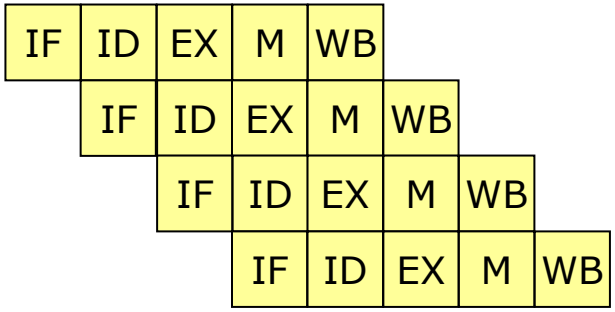


Superescalar



VLIW (EPIC)

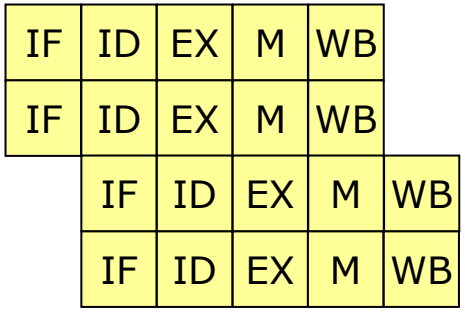
Comparación de procesadores



Segmentado

Características
- Emisión de instrucciones cada ciclo

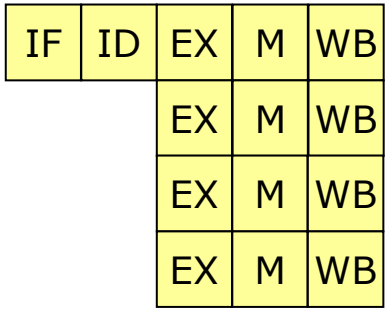
Limitaciones
- Frecuencia de emisión
- Paradas de UF (Riesgos)
- Profundidad de UF



Superescalar

Características
- Emisión múltiple de instrucciones

Limitaciones
- Resolución de riesgos
- Búsqueda de ILP

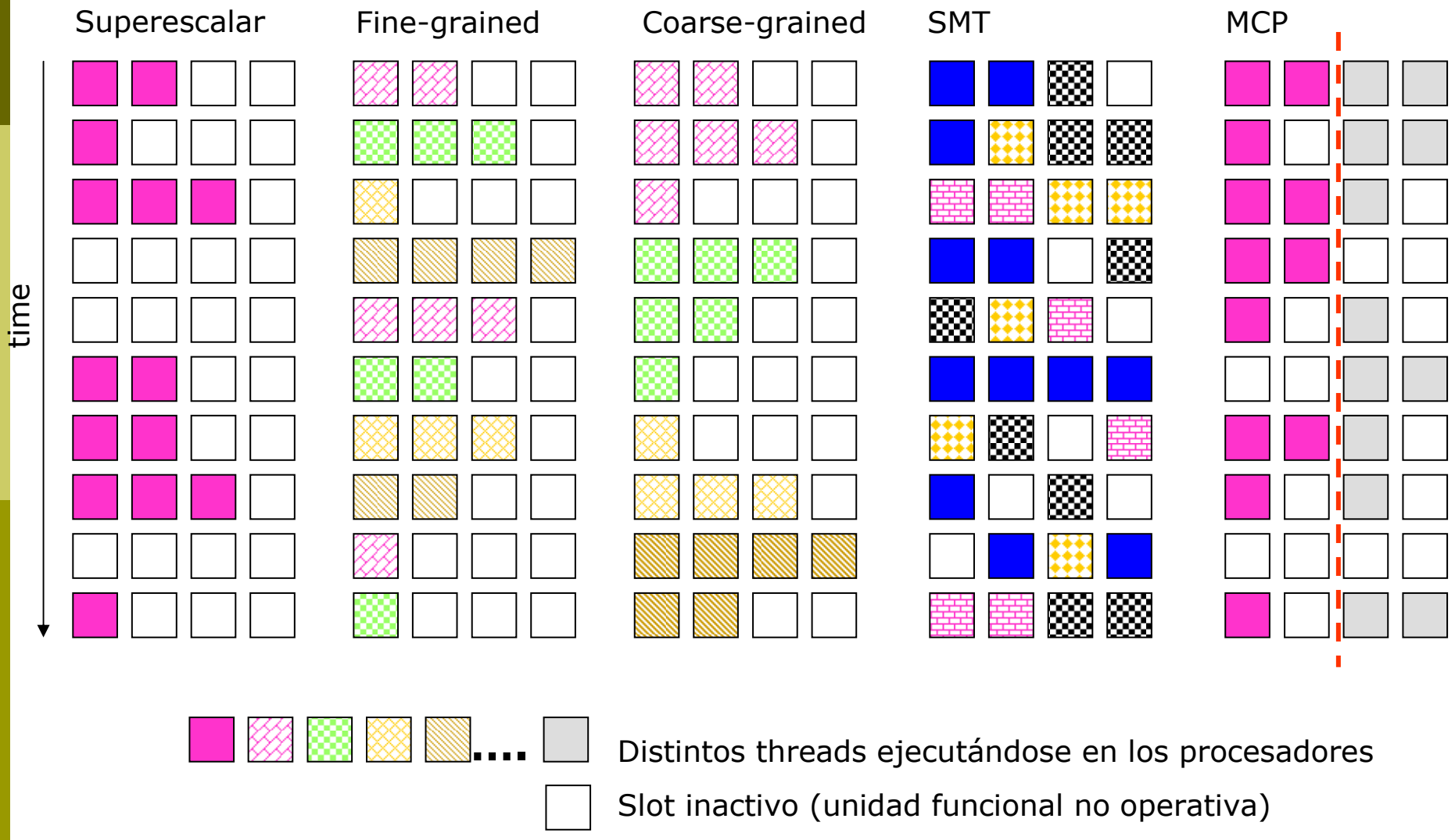


VLIW

Características
- El compilador determina el grado de paralelismo
- Una instrucción implica múltiples operaciones

Limitaciones
- Empaquetamiento
- Compatibilidad de código

Ejecución en las distintas arquitecturas



Dependencias de un programa

Una dependencia entre dos sentencias de un programa es algún tipo de conflicto que evita que las mismas puedan ejecutarse concurrentemente.

Clasificación:

- Estructurales (de recursos)

  consecuencia de las limitaciones de hardware disponible en un sistema de computación. ocurre cuando dos sentencias intentan usar el mismo recurso simultáneamente

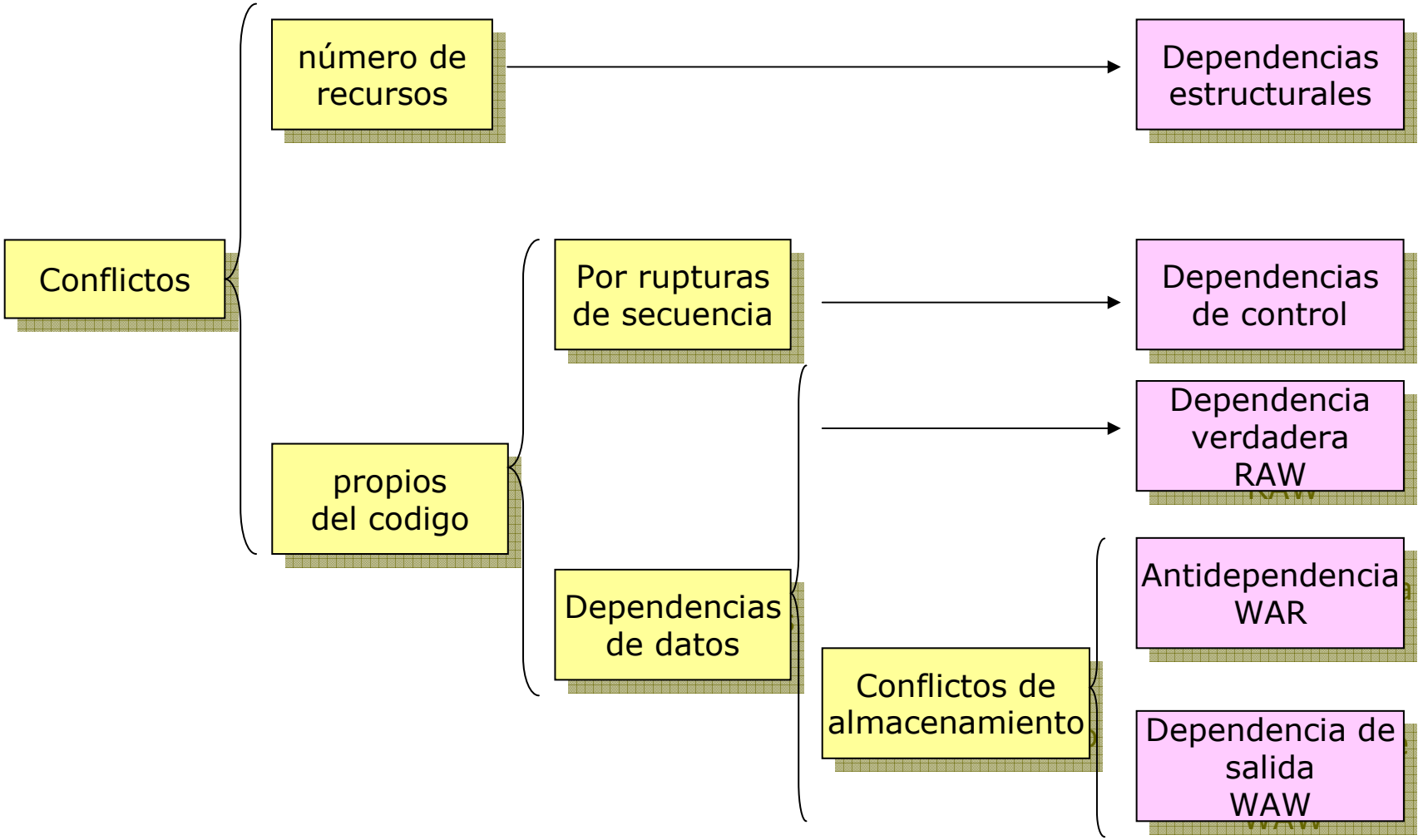
- de control

  Una dependencia de la sentencia S_i a S_j existe cuando la sentencia S_j debiera ser ejecutada sólo si S_i produce un cierto resultado.

- de datos

  se da entre dos sentencias cuando ambas referencian la misma posición de memoria o acceden a un mismo registro

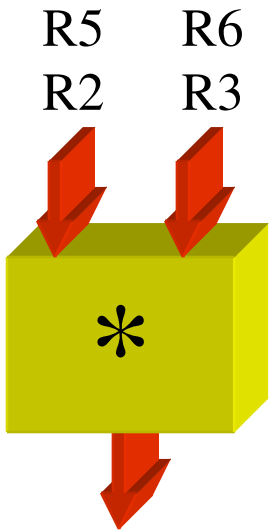
Limitaciones en la ejecución de instrucciones



Dependencias estructurales

Con recursos limitados (uno)

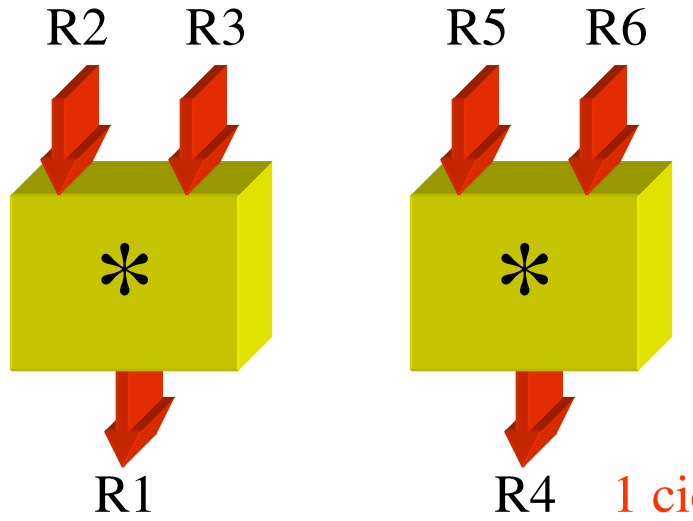
S1 : mul R1, R2, R3
S2 : mul R4, R5, R6



2 ciclos

Con recursos ilimitados (dos)

S1 : mul R1, R2, R3 S2 : mul R4, R5, R6

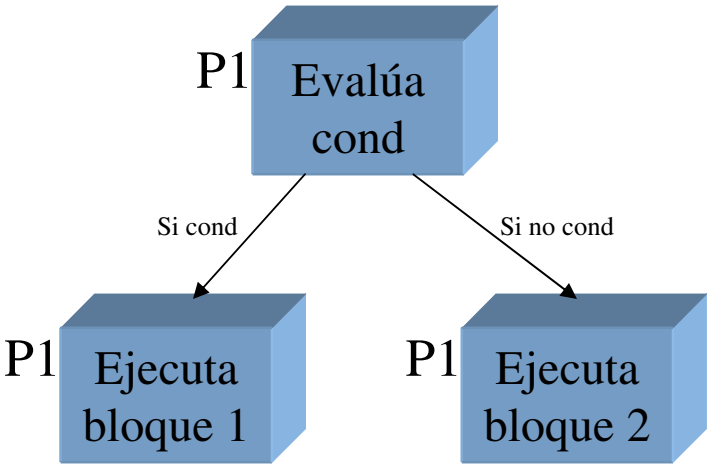


1 ciclo

Dependencias de control

Monoprocesador

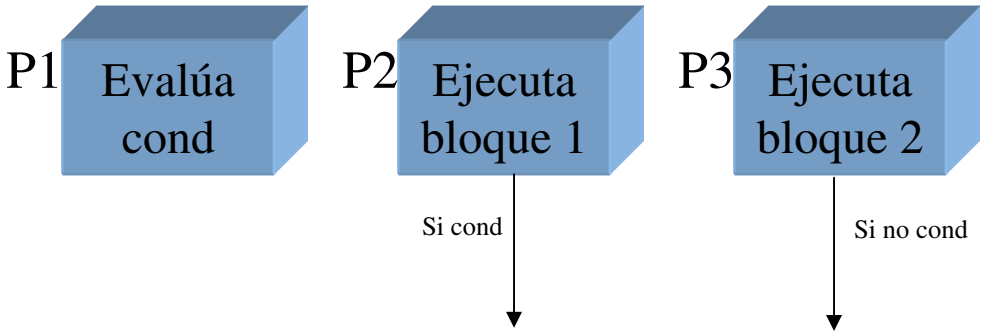
```
IF (cond)
    bloque 1
else
    bloque2
```



2 ciclos

Multiprocesador

```
IF (cond)
    bloque 1
else
    bloque2
```



Continuación del flujo de ejecución
un solo procesador continua. El otro se anula

1 ciclo

Dependencias de datos

Flujo (read after Write hazard): Se da entre S1 y S2 cuando S2 necesita el como operando el resultado de S1

S1 : A = B + C
S2 : D = A - E

Salida (Write after Write hazard): Se da entre S1 y S3 cuando ambas escriben en la misma posición de memoria

S1 : X = Y + Z
S2 : C = X * 22
S3 : X = A - B

Antidependencia (Write after read hazard): Se da entre S2 y S3 ya que S2 necesita leer un valor antes que S3 lo escriba

Dependencias de datos

NOTAS

- Las dependencias de flujo son las únicas dependencias verdaderas en las que el resultado producido por la primer sentencia es usado como valor por la segunda sentencia.

- Las antidependencias y la dependencia de salida ocurren cuando el programador o el compilador reusan espacio de memoria para reducir el espacio de variables del programa.

En estos casos, renombrar variables es una buena política para eliminar estas dependencias.

Eliminación de dep. de datos

Sin renombre

```
aux = B + C
X = aux * 2
write (X)
aux = J + 8
X = aux + J
write (X)
```

P1	P2
aux = B + C X = aux * 2 write (X)	aux = J + 8 X = aux + J write (X)

5 ciclos

Con renombre

```
aux = B + C
X = aux * 2
write (X)
aux2 = J + 8
X2 = aux2 + J
write (X2)
```

P1	P2
aux = B + C X = aux * 2 write (X)	aux2 = J + 8 X2 = aux2 + J write (X2)

3 ciclos

Máxima aceleración de LOOPs

2 motivos principales:

- Los loops se repiten varias veces por lo que su optimización representa un gran aumento de performance
- Permite comparar diferentes técnicas de paralelización sin importar la arquitectura subyacente

Premisas de análisis

El análisis se concentrará en las dependencias de datos

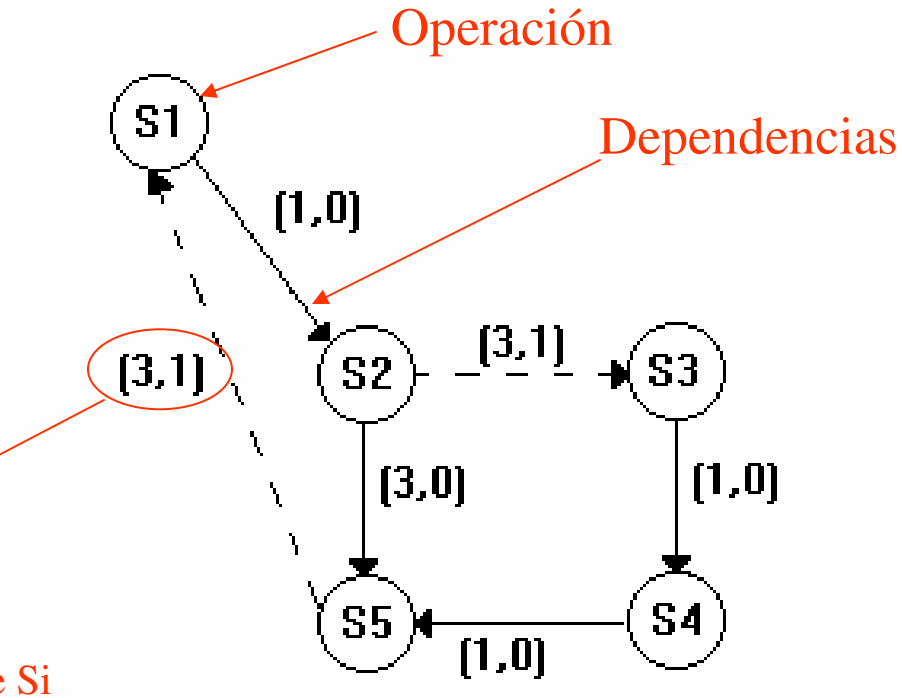
Se puede suponer que las dependencias de recursos se eliminan con hardware adicional

Por simplicidad analizaremos ciclos normalizados con índices de 1 hasta N con incrementos de 1 paso, aunque el análisis puede extenderse a ciclos más complejos posteriormente.

Máxima aceleración de LOOPs

Grafos orientados representan las dependencias de datos

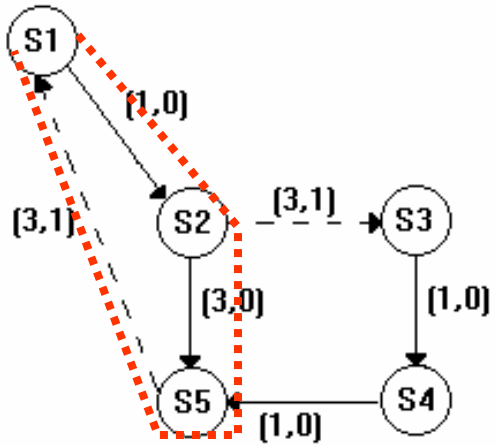
```
For (i=1; i<=N ; i++) {
    S1 : A(i) = E(i-1) + 6
    S2 : B(i) = A(i) * Z
    S3 : C(i) = B(i-1) + X
    S4 : D(i) = C(i) + Y
    S5 : E(i) = B(i) * D(i)
}
```



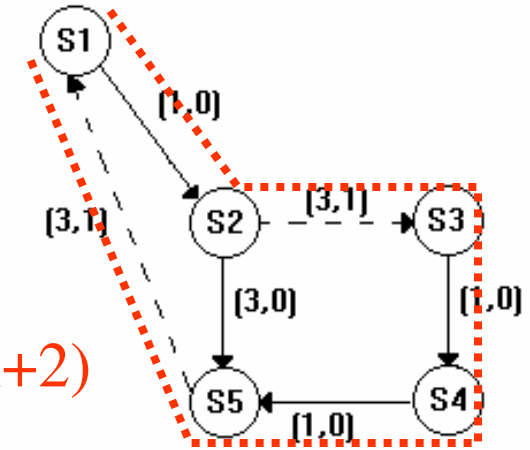
(T_k,C_k)
entre Si y Sj
T_k: tiempo de ejecución de Si
C_k: cantidad de iteraciones posteriores a Si
en que se ejecuta Sj

Análisis del ejemplo…

El grafo de dependencia de la figura anterior posee dos ciclos



S1(i) - S2(i) - S5(i) - S1(i+1)



S3(i) - S4(i) - S5(i) - S1(i+1) - S2(i+1) - S3(i+2)

Ciclo 1...

S1(i) - S2(i) - S5(i) - S1(i+1)

Tiempo	Sentencia
1	S1(1)
2	S2(1)
3	
4	
5	S5(1)
6	
7	
8	S1(2)
9	S2(2)
10	
11	
12	S5(2)
13	
14	
15	S1(3)
...	

Las sentencias se ejecutan N veces

Cada repetición del ciclo requiere

$T_c = \Sigma T_k = 7$ unidades de tiempo

El tiempo total para ejecutar las sentencias será

$Tt1 = N * T_c = N * 7$ unidades de tiempo.

Ciclo 2...

S3(i) - S4(i) - S5(i) - S1(i+1) - S2(i+1) - S3(i+2)

Tiempo	Sentencia
1	S3(1)
2	S4(1)
3	S5(1)
4	
5	
6	S1(2)
7	S2(2)
8	
9	
10	S3(3)
11	S4(3)
12	S5(3)
13	
14	
15	S1(4)
16	S2(4)
17	
18	
19	S3(5)
...	

La ejecución de las sentencias requiere

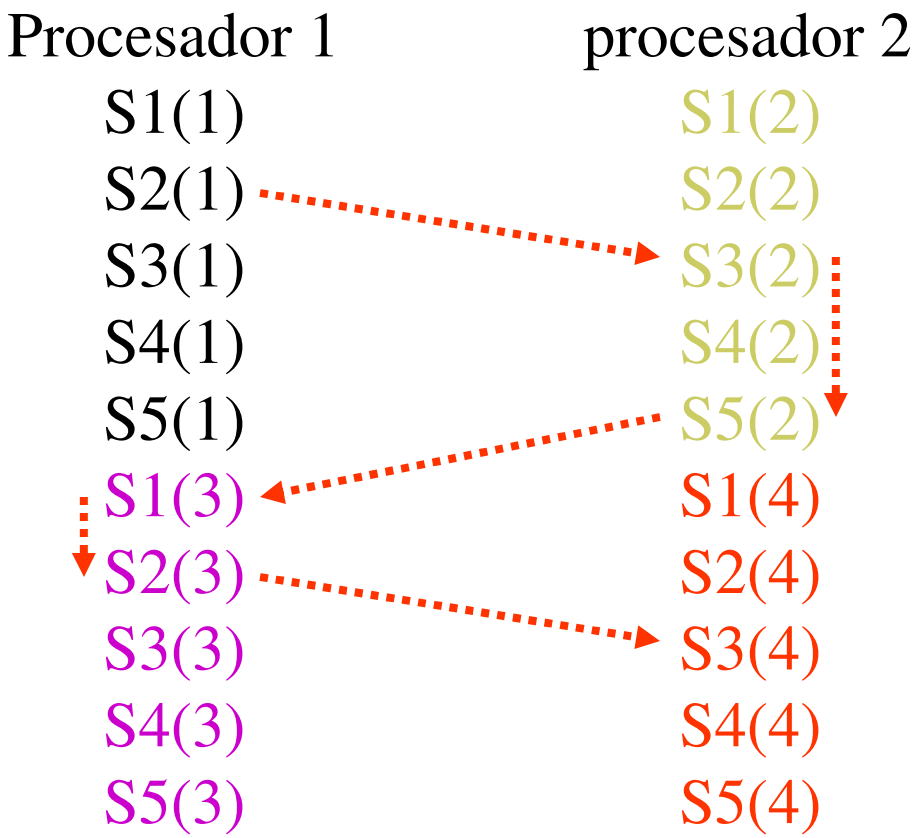
$T_c = \Sigma T_k = 9$ unidades de tiempo

Puesto que el patrón se repite cada dos iteraciones, la versión desenrrollada del ciclo se ejecuta N/2 veces

El tiempo total de ejecución para este ciclo es

$Tt2 = T_c * (N/2) = 9N/2$ unidades de tiempo.

Ejecución paralela del ciclo 2



Tiempo de dependencia crítico

trayectoria crítica (critical path): cadena de dependencia más larga producida al desenrollar los ciclos de un loop. Su tiempo de ejecución se denota como T_{crit}.

$$T_{crit} = \max(Tti) = \max(7N, 9N/2) = 7N \text{ unidades de tiempo}$$

Aceleración máxima: cociente entre el tiempo de la versión secuencial original y el tiempo de ejecución de la trayectoria crítica

$T_L = 9$

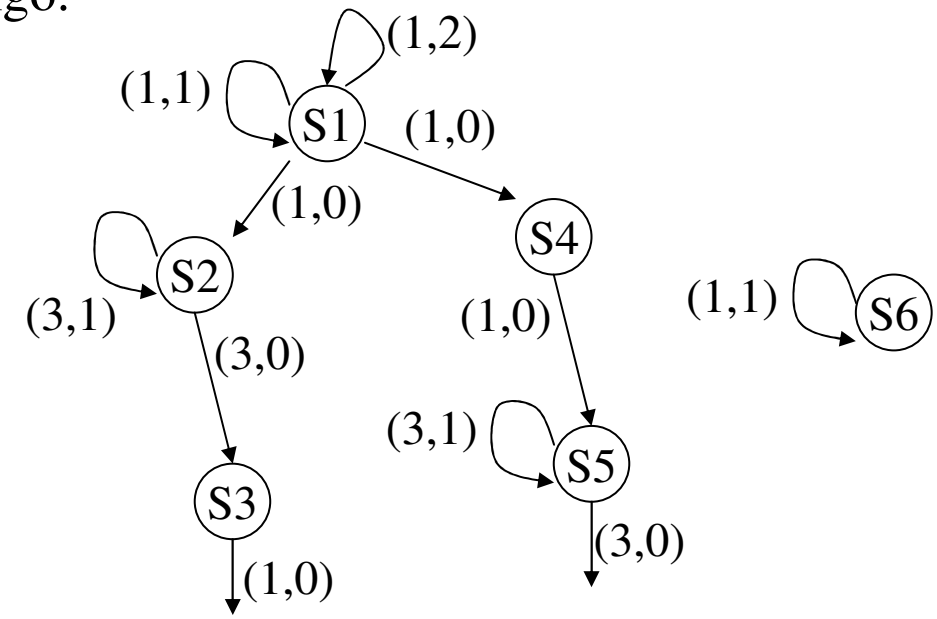
$$S_{max} = (NT_L)/T_{crit} = 9N/7N = 9/7 \approx 1{,}28$$

Segundo ejemplo

NOTA: Vale aclarar que lo que realmente se busca son las ditintas trazas de código desde alguna instrucción de las primeras iteraciones hasta alguna instrucción de lás últimas iteraciones

Sea el siguiente fragmento de código:

```
For (i=2; i<=6 ; i++) {
    S1 : A(i) = A(i-1) + A(i-2)
    S2 : B(i) = A(i) * B(i-1)
    S3 : C(i) = B(i) + X
    S4 : D(i) = A(i) + Y
    S5 : E(i) = E(i-1) * D(i)
    S6 : F(i) = F(i-1) + 5
}
```



Segundo ejemplo

Las trazas son:

C1: N * Ts1 + Ts2 + Ts3	5*1+3+1 = 9
C2: (N/2) * Ts1 + Ts2 + Ts3	2,5*1+3+1 = 6,5
C3: Ts1 + N * Ts2 + Ts3	1+5*3+1 = 17
C4: N * Ts1 + Ts4 + Ts5	5*1+1+3 = 9
C5: (N/2) * Ts1 + Ts4 + Ts5	2,5*1+1+3 = 6,5
C6: Ts1 + Ts4 + N * Ts5	1+1+5*3 = 17
C7: N * Ts6	5*1 = 5

Por ejemplo:
Para C1, se calcula:

S1(2) – S1(3) – S1(4) – S1(5) – S1(6) – S2(6) – S3(6)

Para C3:

S1(2) – S2(2) – S2(3) – S2(4) – S2(5) – S2(6) – S3(6)