

Tema 03: Paralelismo y Pipelining

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

<http://microprocesadores.unt.edu.ar/arqcom/>

¿Cuáles temas veremos?



- ▶ Primera parte: **Generalidades**
 - ▶ Introducción. Conceptos generales.
 - ▶ **P**erformance, **P**rice & **P**ower.
 - ▶ Conceptos de **P**aralelismo y **P**ipelining.
- ▶ Segunda parte: **Procesadores**
 - ▶ ISA: qué es, cómo se diseña.
 - ▶ Diseños de procesadores para *performance*.
- ▶ Tercera parte: **Subsistema de Memoria**
 - ▶ Memorias Caché.
 - ▶ Memoria Virtual.
- ▶ Cuarta parte: **Sistema completo**
 - ▶ Subsistemas de Entrada/Salida y de Comunicaciones.
 - ▶ Consumo de energía.
- ▶ Quinta parte: **Nuevos paradigmas**
 - ▶ Arquitecturas específicas.
 - ▶ Seguridad.

Temas que veremos

- ▶ Fundamentos de Paralelismo y Pipelining.
- ▶ Medidas de Performance.
 - ▶ Aceleración y Eficiencia (rendimiento).
- ▶ Herramientas
 - ▶ Perfil de la tarea.
 - ▶ Diagrama de dependencias.
- ▶ Problemas reales
 - ▶ Cuellos de Botella
 - ▶ Dependencias.
 - ▶ Colisiones. Riesgos.
- ▶ Ejemplos de Aplicación

Introducción

- ▶ $t = CI * CPI * T$
- ▶ *¿Cómo mejorar?*
- ▶ Pero... hay límites a la tecnología
 - ▶ Disipación de Calor, Velocidad de la luz.
- ▶ La clave: trabajar en cosas simultáneas.
 - ▶ Cerebro humano: mala tecnología, muy rápido. Mucho paralelismo.
- ▶ La mayoría de las máquinas actuales emplean:
 - ▶ Paralelismo, Pipelining y ambas (procesadores superescalares).
- ▶ Son temas **fundamentales** en Computación.
 - ▶ Dos de las Grandes Ideas en Arquitectura de Computadoras.

Paralelismo – Idea clave

- ▶ Agregar estaciones de trabajo.
- ▶ Hacer más trabajo en igual cantidad de tiempo.
- ▶ *¿Mejora latencia o productividad?*

Paralelismo – Ejemplo

- ▶ Para la fabricación de un determinado producto se requieren tres etapas:
 - ▶ 2 tareas independientes de 9 hs. cada una.
 - ▶ 5 tareas independientes de 3 hs. cada una.
 - ▶ 2 tareas independientes de 12 hs. cada una.
- ▶ Las tareas de cada etapa son independientes entre sí, pero las tareas de una etapa deben completarse antes de iniciar la etapa siguiente.
- ▶ Cada tarea es realizada por un trabajador.
- ▶ *Si tengo un solo trabajador, ¿cuánto demora en completar el trabajo?*
- ▶ *¿Y si tuviese dos trabajadores?*

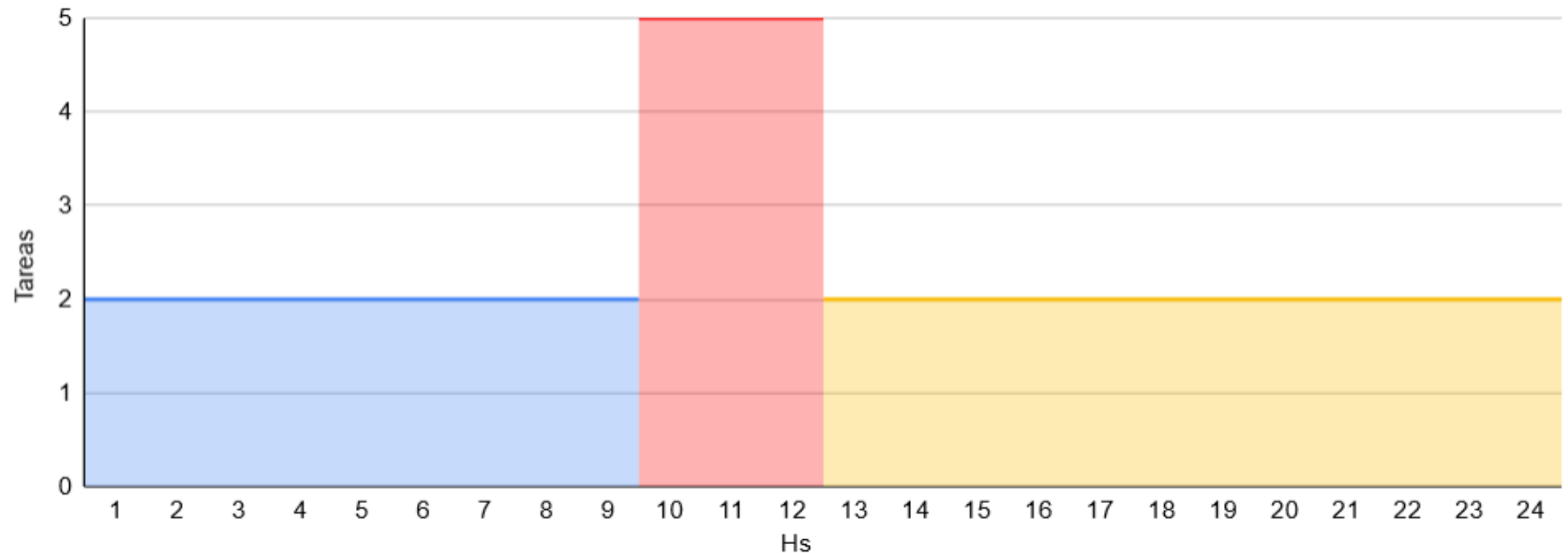
Paralelismo – Métricas

- ▶ Factor de Aceleración
 - ▶ $A = t(\text{serie}) / t(\text{paralelo})$
 - ▶ *¿Cuánto para el ejemplo?*
- ▶ Eficiencia (o rendimiento)
 - ▶ $\text{Eficiencia} = \text{Trabajo realizado} / \text{Máximo trabajo posible}$
 - ▶ *¿Cuánto para el ejemplo?*

Paralelismo – Ejemplo (cont.)

- ▶ Al aumentar la cantidad de trabajadores disminuye el tiempo total para hacer la tarea.
 - ▶ ¡Sigamos aumentando la cantidad de trabajadores!
 - ▶ Si pusiese diez trabajadores, *¿cuánto demoran en completar el trabajo?*
 - ▶ *¿Aceleración?*
 - ▶ *¿Rendimiento?*
- ▶ Es claro que el paralelismo tiene un límite
 - ▶ *¿Cómo podemos descubrirlo?*

Perfil de la Tarea



- ▶ Muestra **tareas independientes** que se pueden realizar **en función del tiempo**.
- ▶ Muestra dependencias (no se puede adelantar).
- ▶ Determina **duración mínima** de la tarea y cantidad máxima de estaciones de trabajo útiles (**grado de paralelismo**)
- ▶ Eficiencia = trabajo realizado/máximo posible (Área coloreada / Área total)

Perfil de la Tarea (limitado)

- ▶ El perfil anterior es el ideal. A veces es necesario construir un perfil con limitaciones.
- ▶ En nuestro ejemplo, suponiendo que tenemos como máximo 4 trabajadores disponibles en simultáneo, sería así:
 - ▶ La duración ya no es la mínima.
 - ▶ Podemos calcular la eficiencia de la misma manera.

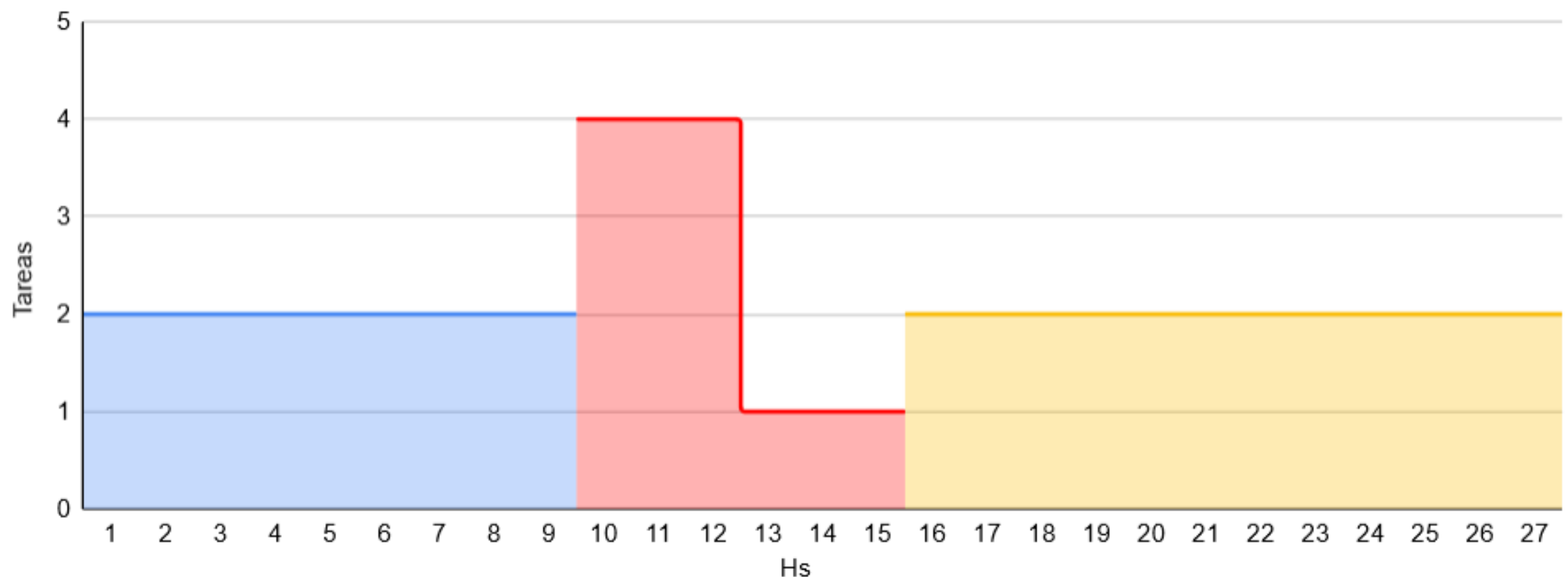


Diagrama de Dependencias

- ▶ Es un grafo dirigido, usado para determinar la dependencia entre tareas.
 - ▶ Determina cuántas tareas pueden ejecutarse en paralelo, lo que determina el **grado de paralelismo**.
 - ▶ Complementa al perfil de la tarea.
- ▶ Existen tres tipos de dependencias:
 - ▶ Dependencias verdaderas.
 - ▶ Hay un pase de información real entre las tareas.
 - ▶ Dependencias de salida.
 - ▶ Anti dependencias.
- ▶ Las dependencias verdaderas permiten establecer el **camino crítico** del diagrama.
 - ▶ Duración mínima de la tarea.

Tipos de Dependencias

- ▶ Verdaderas dependencias: RAW (*Read-After-Write*):
 - ▶ Cuando una instr. j usa (lee) un dato que es escrito por una instr. i previa.
 - ▶ lw **x1**, 0 (x2)
add x1, **x1**, x3
- ▶ Dependencias de salidas: WAW (*Write-After-Write*):
 - ▶ Cuando una instr. j escribe un dato que es escrito por una instr. i previa.
 - ▶ lw **x1**, 0 (x2)
add **x1**, x1, x3
- ▶ Anti dependencias: WAR (*Write-After-Read*):
 - ▶ Cuando una instr. j escribe un dato que es leído por una instr. i previa.
 - ▶ add x1, x1, **x3**
lw **x3**, 4 (x2)
- ▶ Las dependencias de salida y las anti dependencias son agrupadas como “dependencias de nombre”.
 - ▶ A pesar de no ser verdaderas, pueden causar problemas.

Diagrama de Dependencias

- ▶ Veamos un ejemplo sencillo, con una secuencia de instrucciones de RV32I.
 - ▶ Por simplicidad, supondremos que todas las instrucciones demoran 1 T en ejecutarse.
1. add x1, x2, x3
 2. sub x4, x1, x5
 3. add x1, x6, x7
 4. add x8, x4, x9
 5. add x10, x11, x12
 6. add x4, x13, x14
 7. add x15, x1, x16
 8. sw x8, 0 (x17)
- ▶ *¿Cuánto demora en ejecutarse la secuencia?*

Diagrama de Dependencias – Ejemplo

1. add x1, x2, x3
2. sub x4, x1, x5
3. add x1, x6, x7
4. add x8, x4, x9
5. add x10, x11, x12
6. add x4, x13, x14
7. add x15, x1, x16
8. sw x8, 0 (x17)

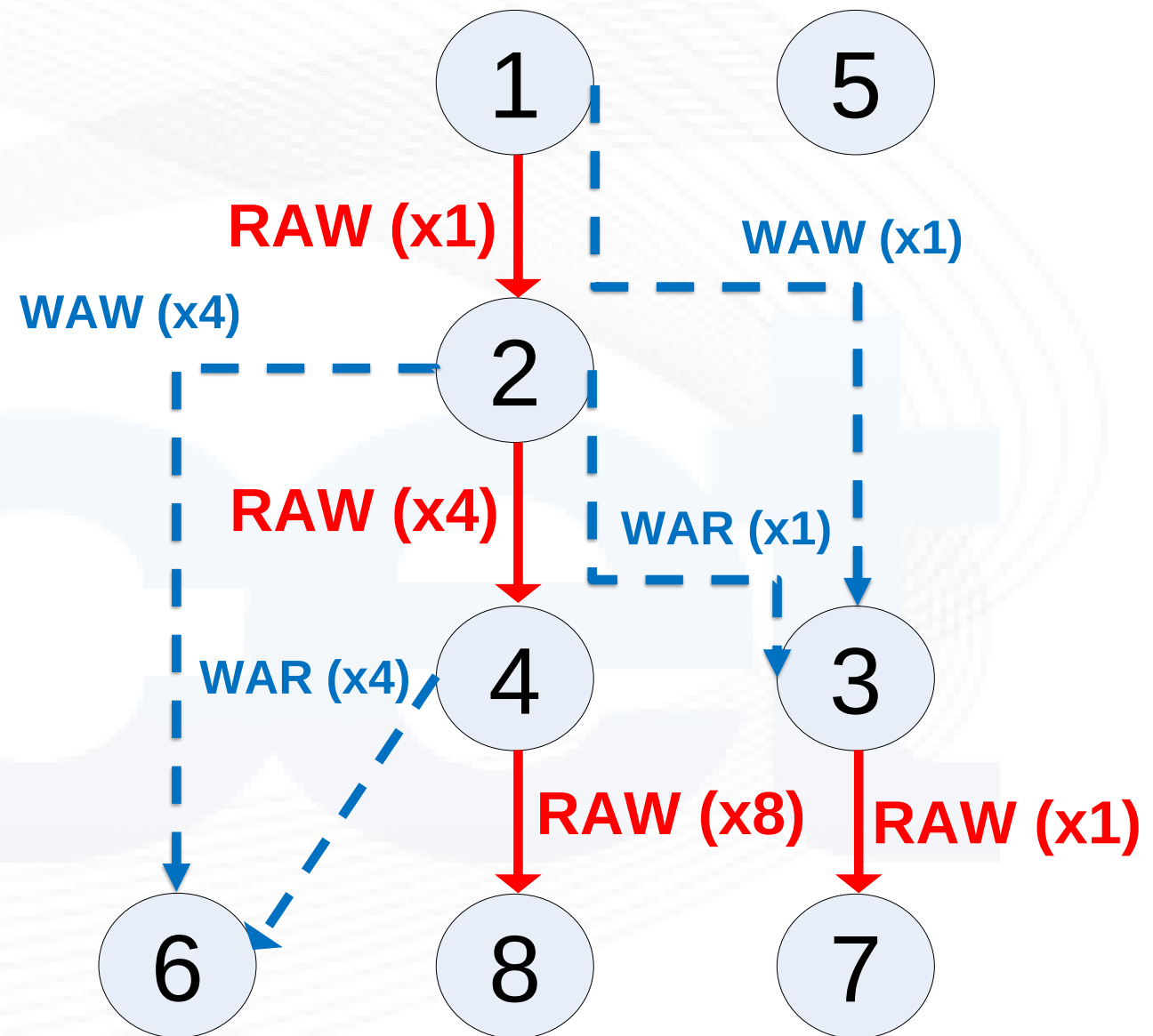


Diagrama de Dependencias – Ejemplo

- ▶ **¿Cuál es el grado de paralelismo?**
- ▶ **¿Cuál es el camino crítico?**
 - ▶ La secuencia más larga de dependencias RAW.
 - ▶ No necesariamente es único.
- ▶ **¿Cuánto es la duración mínima posible de la secuencia?**
- ▶ **¿Cuánto sería la máxima aceleración posible?**
- ▶ Implica que:
 - ▶ Se deberían poder ejecutar 3 instrucciones al mismo tiempo.
 - ▶ Se deberían poder ejecutar instrucciones fuera de orden.
- ▶ **¿Cuánto sería la eficiencia en este caso?**

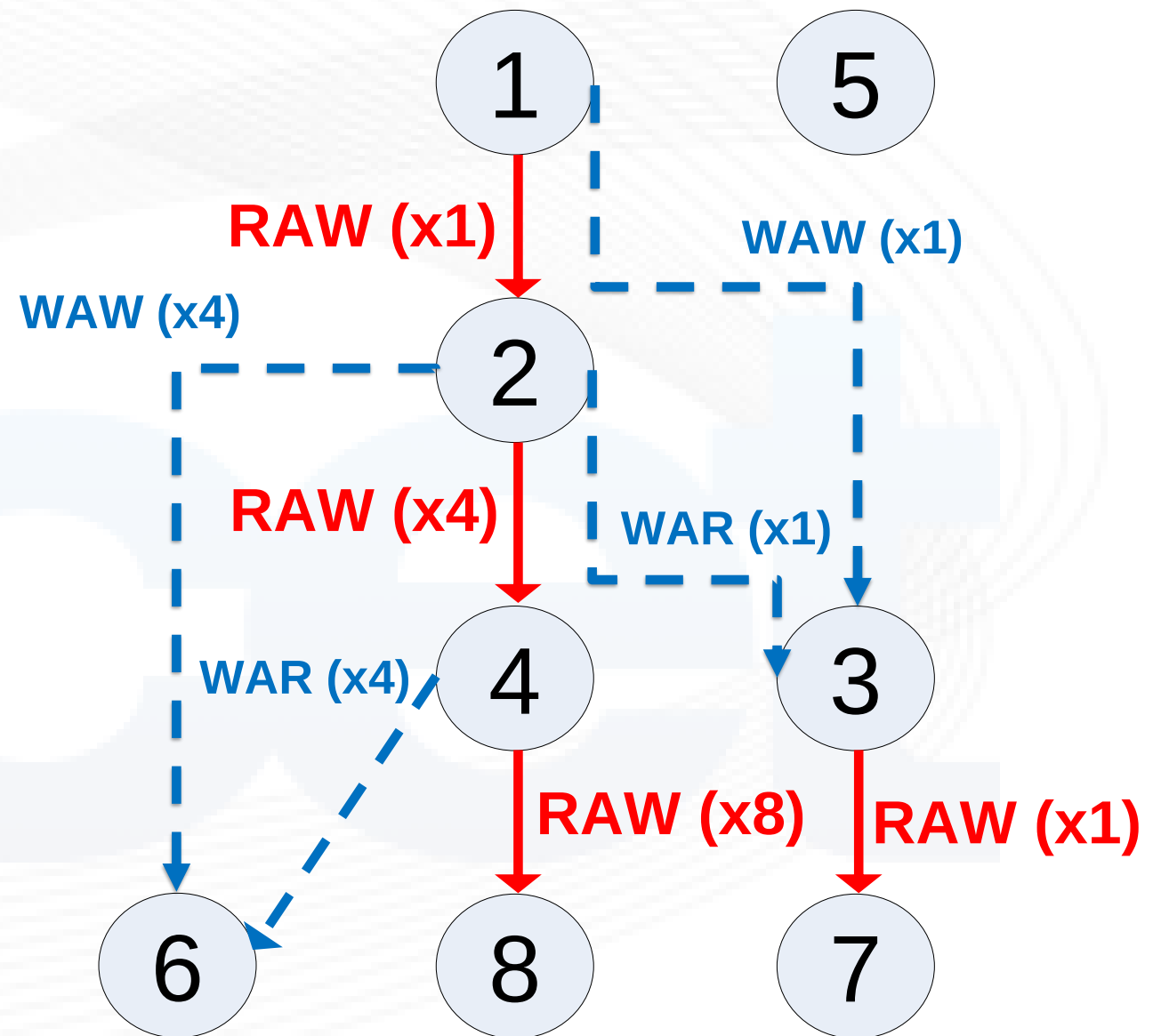


Diagrama de Dependencias – Ejemplo

- ▶ Si solamente se pudieran ejecutar dos instrucciones en paralelo:

- ▶ *¿Cuánto es la duración de la secuencia?*
- ▶ *¿Cuánto sería la aceleración?*
- ▶ *¿Cuánto sería la eficiencia en este caso?*

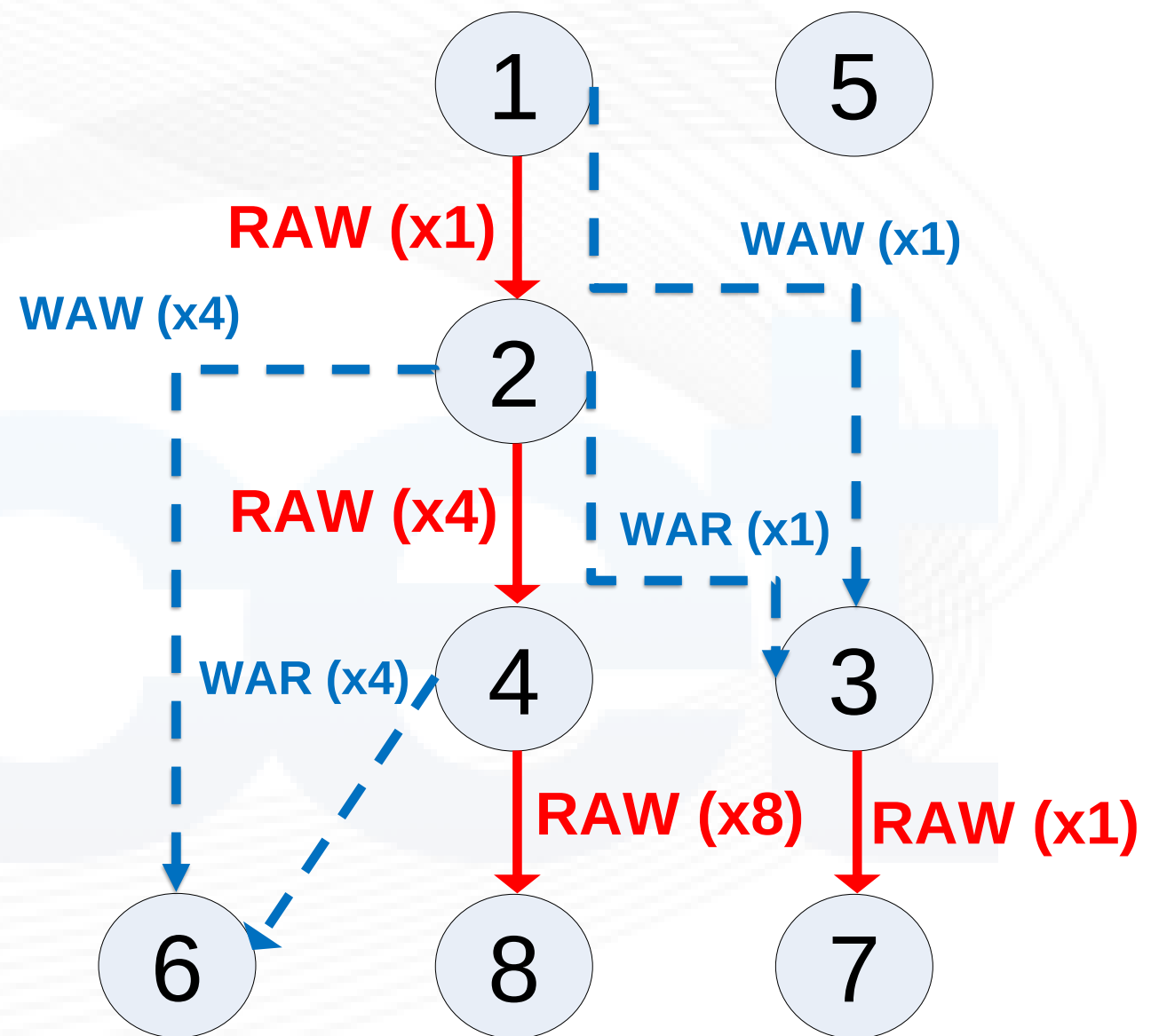


Diagrama de Dependencias – Ejemplo 2

1. add x1, x2, x3
2. sub x4, x1, x5
3. add **x20**, x6, x7
4. add x8, x4, x9
5. add x10, x11, x12
6. add **x21**, x13, x14
7. add x15, **x20**, x16
8. sw x8, 0 (x17)

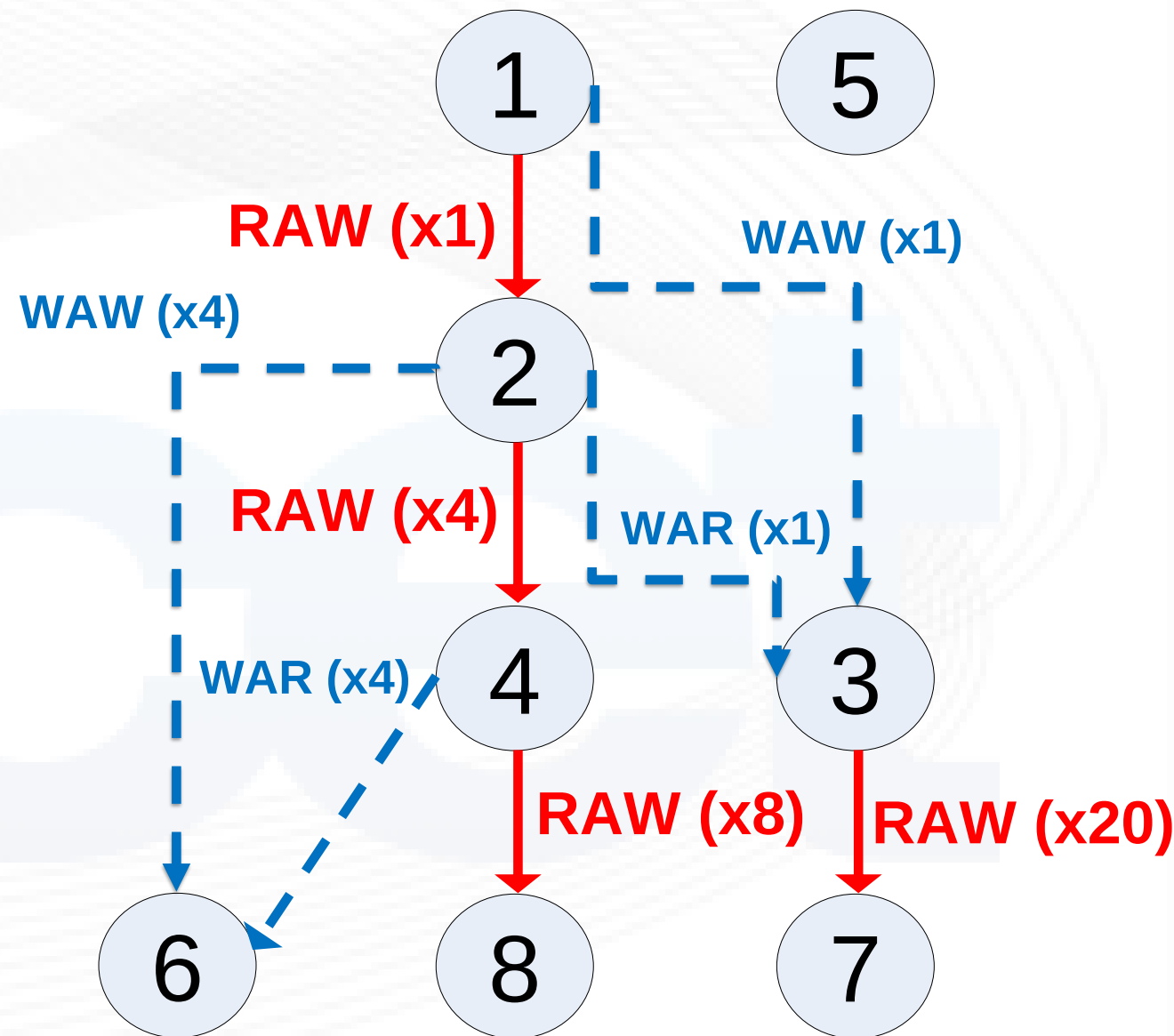


Diagrama de Dependencias – Ejemplo 2

- ▶ *¿Cuál es el grado máximo de paralelismo?*
- ▶ *¿Cuál es el camino crítico?*
- ▶ *¿Cuánto es la duración mínima posible de la secuencia?*
- ▶ *¿Cuánto sería la máxima aceleración posible?*
- ▶ *¿Cuánto sería la eficiencia en este caso?*

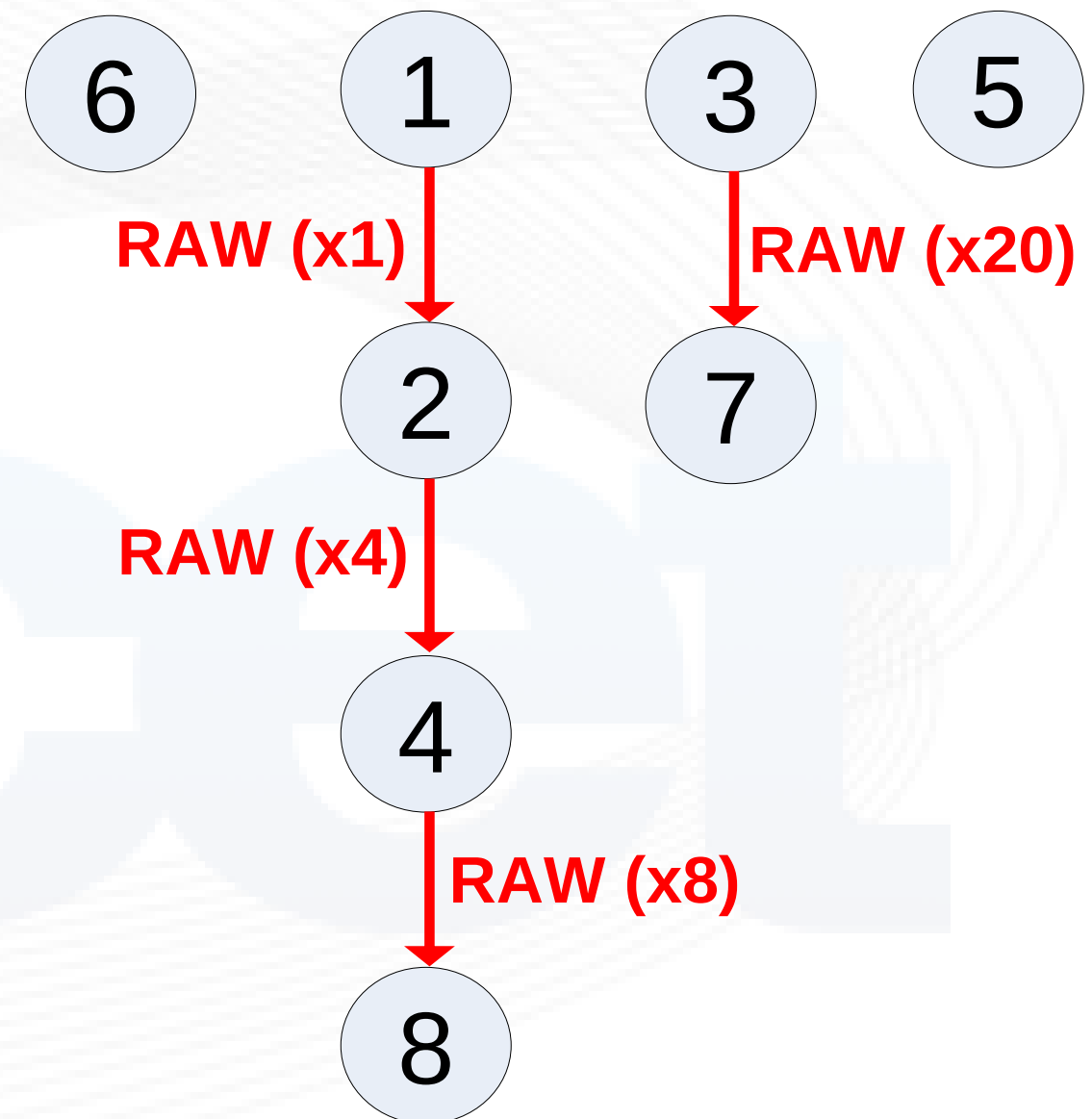


Diagrama de Dependencias – Ejemplo 2

- ▶ Si solamente se pudieran ejecutar dos instrucciones en paralelo:
- ▶ *¿Cuánto es la duración de la secuencia?*
- ▶ *¿Cuánto sería la aceleración?*
- ▶ *¿Cuánto sería la eficiencia en este caso?*

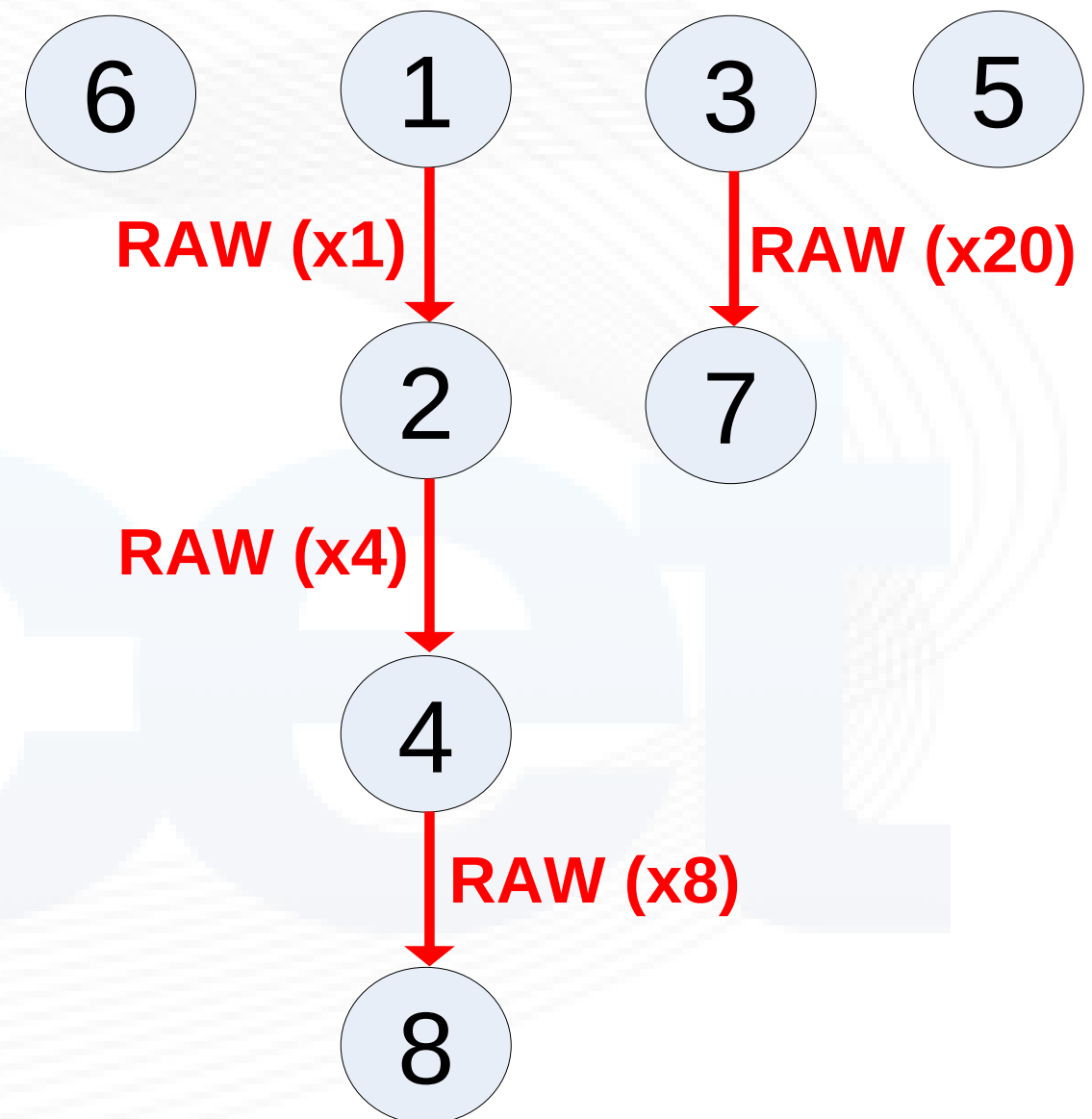


Diagrama de Dependencias - Conclusiones

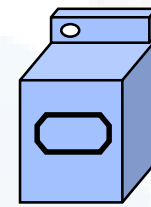
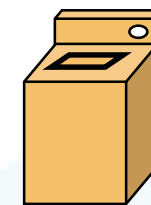
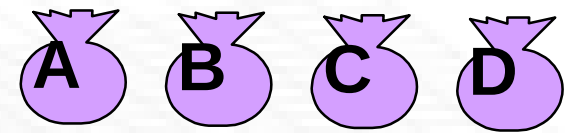
- ▶ El diagrama de dependencias es una herramienta muy importante para explotar el **paralelismo a nivel de instrucciones** dentro de un procesador.
 - ▶ Tiene dos requerimientos importantes:
 - ▶ Poder ejecutar varias instrucciones en simultáneo.
 - ▶ Poder ejecutar instrucciones fuera de orden.
 - ▶ Volveremos sobre esto en el Tema 08.
- ▶ El **renombramiento de registros permite eliminar dependencias de nombre (WAW y WAR)**.
 - ▶ De esa manera puede aumentar el grado de paralelismo disponible.
 - ▶ No siempre reduce el camino crítico.
 - ▶ Puede aumentar la aceleración y la eficiencia en caso de estar limitados en hardware.
 - ▶ Puede hacerlo el programador o el compilador o el mismo hardware.
 - ▶ Requiere tener muchos registros libres disponibles.

Pipelining – Idea clave

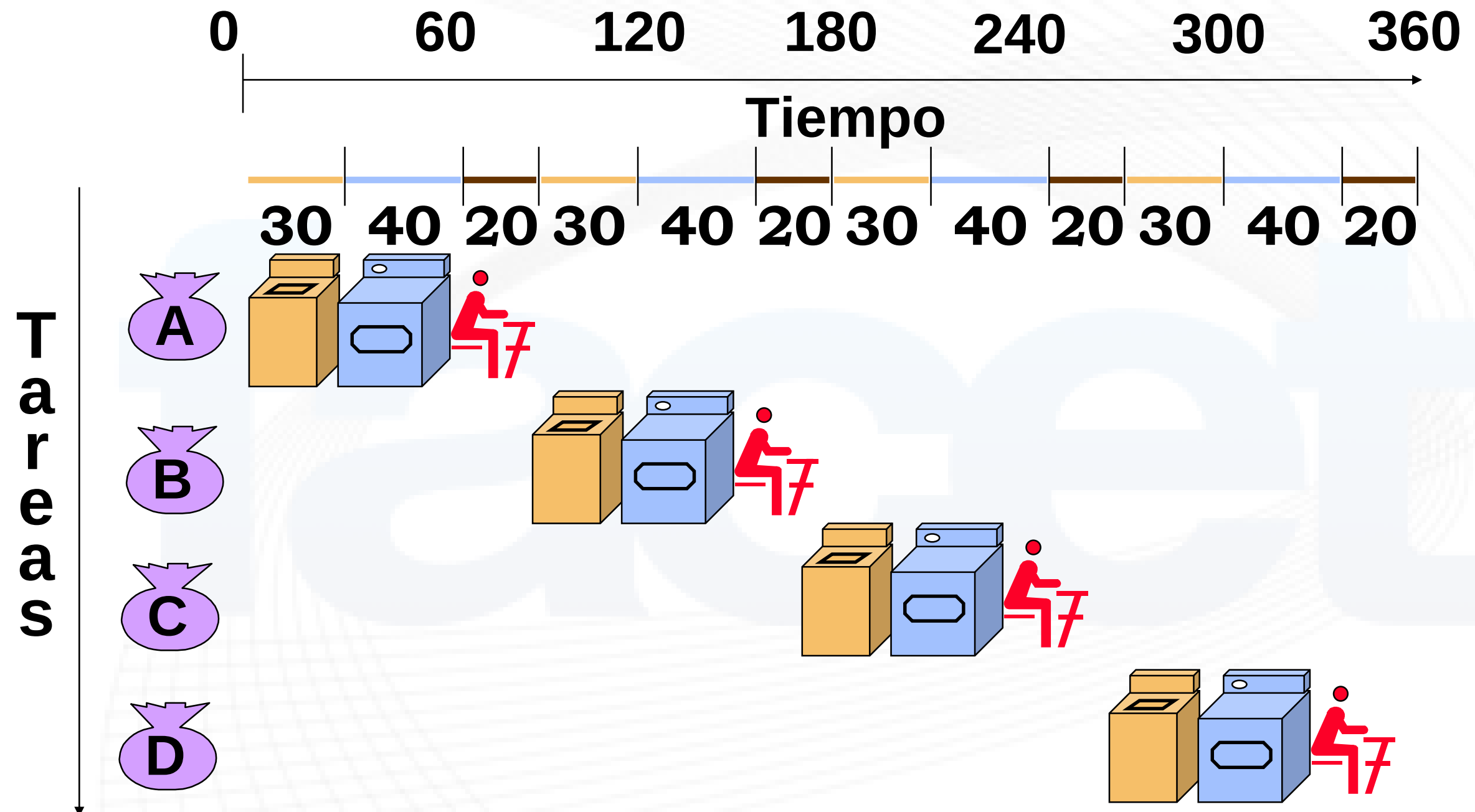
- ▶ Procesamiento en serie.
- ▶ Dividir la tarea en etapas secuenciales, e ir “solapando” su ejecución.
- ▶ Empezar a utilizar un recurso apenas esté disponible.
- ▶ *¿Mejora latencia o productividad?*

Pipelining – Ejemplo básico

- ▶ Ejemplo de una lavandería.
 - ▶ Ana, Belén, Carlos y Daniel.
 - ▶ Cada uno tiene una carga de ropa
 - ▶ Los pasos son lavar, secar y doblar.
- ▶ Lavarropa demora 30 min.
- ▶ Secarropa demora 40 min.
- ▶ Planchado y embolsado demora 20 min.
- ▶ *¿Cuánto tiempo demora en total cada carga?*
- ▶ *¿Y las cuatro cargas?*

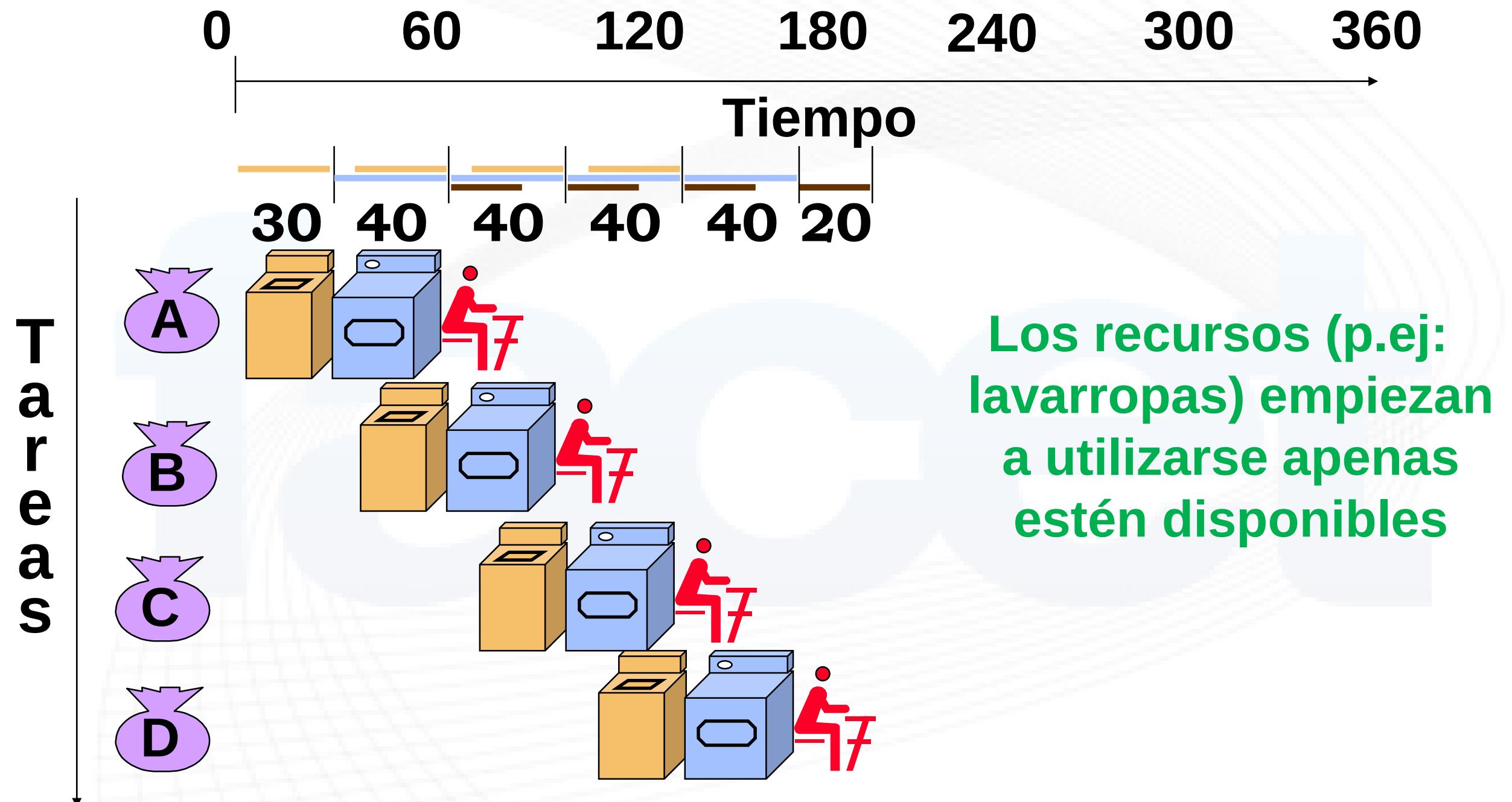


Pipelining – Ejemplo básico



Lavado Secuencial demora 360 minutos para 4 cargas.

Pipelining – Ejemplo básico

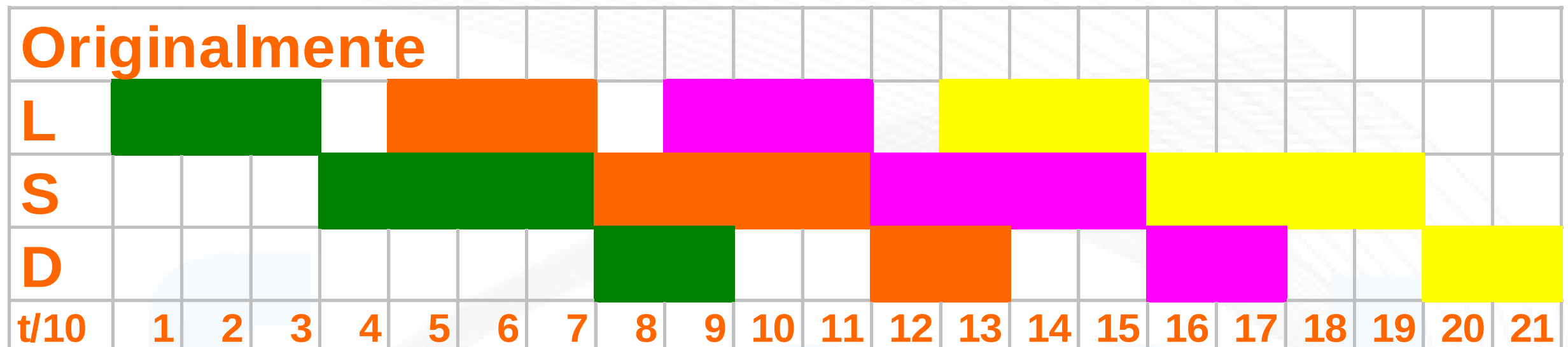


Lavado en Pipelining demora 210 minutos para 4 cargas.

Pipelining – Métricas

- ▶ Productividad = Tasa de salida
 - ▶ Tareas completadas por unidad de tiempo.
- ▶ **Aceleración = $t(\text{serie}) / t(\text{pipe})$**
- ▶ Latencia = tiempo total que demora una tarea
 - ▶ Desde que ingresa al pipeline hasta que sale.
- ▶ Eficiencia = Trabajo realizado / Máx posible.
- ▶ *¿Cuáles son estos valores para el ejemplo?*

Pipelining – Visualización



- ▶ En el eje vertical se ubican los distintos puestos de trabajo.
- ▶ En el eje horizontal, el tiempo.
- ▶ Las tareas se identifican con diferentes colores (o letras o números).
- ▶ Las tareas se van desplazando por los puestos de trabajo, **si es que el siguiente está disponible**.
 - ▶ Por ello es que hay una demora (“hueco”) al ingresar la segunda tarea.
- ▶ Se obtienen fácilmente: la duración total, la latencia de cada tarea, la productividad.
- ▶ Se puede calcular el rendimiento: área ocupada / área total.

Eficiencia vs Productividad

- ▶ *¿Cuál es la eficiencia de un único trabajador en un sistema serie?*
- ▶ Y si tengo m trabajadores en pipelining, con una eficiencia del 100%, ¿cuánto es la aceleración?
- ▶ Idealmente, en un pipeline se cumple que:
 - ▶ $A = m * \text{eficiencia}$
- ▶ Ojo, eficiencia del pipeline no es lo mismo que eficiencia de cada recurso.
 - ▶ ¿Cuál es la eficiencia en el uso del lavarropas?

Pipeline – Observaciones adicionales

- ▶ **Pipelining no mejora la latencia de una tarea particular, pero sí la productividad de la carga total de trabajo.**
- ▶ **La productividad está limitada por la etapa más lenta.**
- ▶ **En un mismo instante de tiempo, hay múltiples tareas ejecutándose simultáneamente (¿paralelismo?).**
- ▶ **Idealmente, la máxima aceleración posible es igual al número de etapas.**
- ▶ Si la duración de las etapas es desbalanceada, se reduce la aceleración (y el rendimiento).
- ▶ La aceleración también se reduce por tiempos de “llenado” y de “vaciado” del pipeline.

Pipeline ideal

- ▶ Posee m etapas equilibradas, de duración igual a T .
- ▶ Ejecuta una cantidad enorme de tareas idénticas, $n \rightarrow$ infinito.
 - ▶ Esto se denomina **estado de régimen**.
- ▶ Latencia = $m * T$
- ▶ Tiempo(serie) = $n * m * T$
- ▶ Tiempo(pipeline) = $(m-1) * T + n * T$
 - ▶ Una vez que se llena, sale una tarea cada T .
- ▶ Aceleración = $m / [1 + (m - 1) / n] \rightarrow m$
- ▶ Productividad = $1 / T$
- ▶ Rendimiento = $A / m = 100\%$

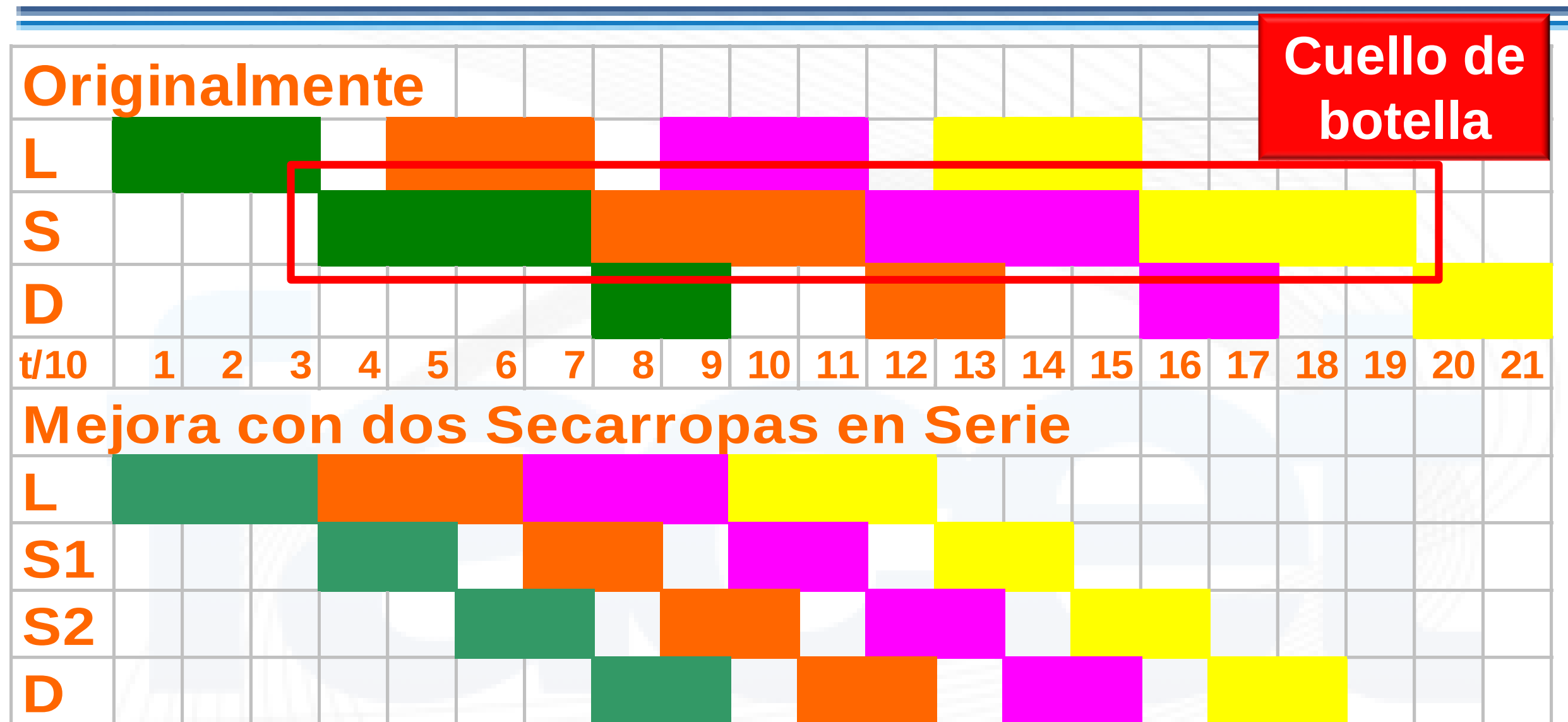
Pipelining – Problemas comunes

- ▶ No contar con suficientes tareas para entrar en estado de régimen.
 - Hay que considerar tiempos de llenado y vaciado.
- ▶ No contar con etapas de igual duración.
 - Produce cuellos de botella.
- ▶ Tener distintos tipos de tareas que utilicen distintas combinaciones de etapas.

Pipelining – Cuellos de botella

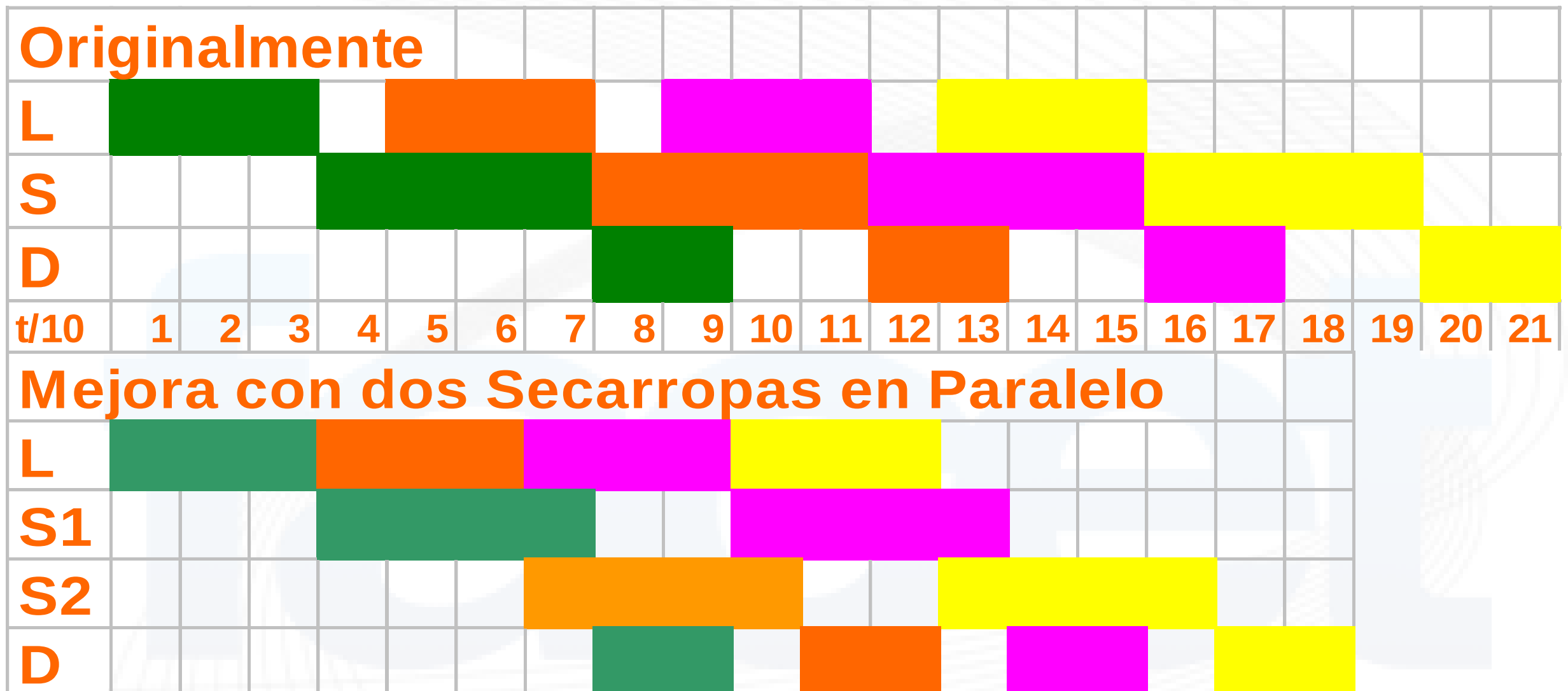
- ▶ En el ejemplo de la lavandería, una etapa era más lenta que las demás.
 - ▶ El secarropas demora 40 minutos.
 - ▶ En estado de régimen limita la productividad, disminuye el rendimiento y la aceleración.
- ▶ *¿Cómo podemos mejorarlo?*
 - ▶ Comprando un secarropas más rápido.
 - ▶ Comprando otro secarropas igual, y poniéndolos en serie.
 - ▶ Dividiendo la etapa de secado en dos etapas de 20 minutos.
 - ▶ Comprando otro secarropas igual y poniéndolos en paralelo.
 - ▶ Alternando el uso entre uno y otro.

Cuellos de botella – Solución serie



- ▶ *¿Nueva productividad?*
- ▶ ¡Hay un nuevo cuello de botella! *¿Alternativas?*

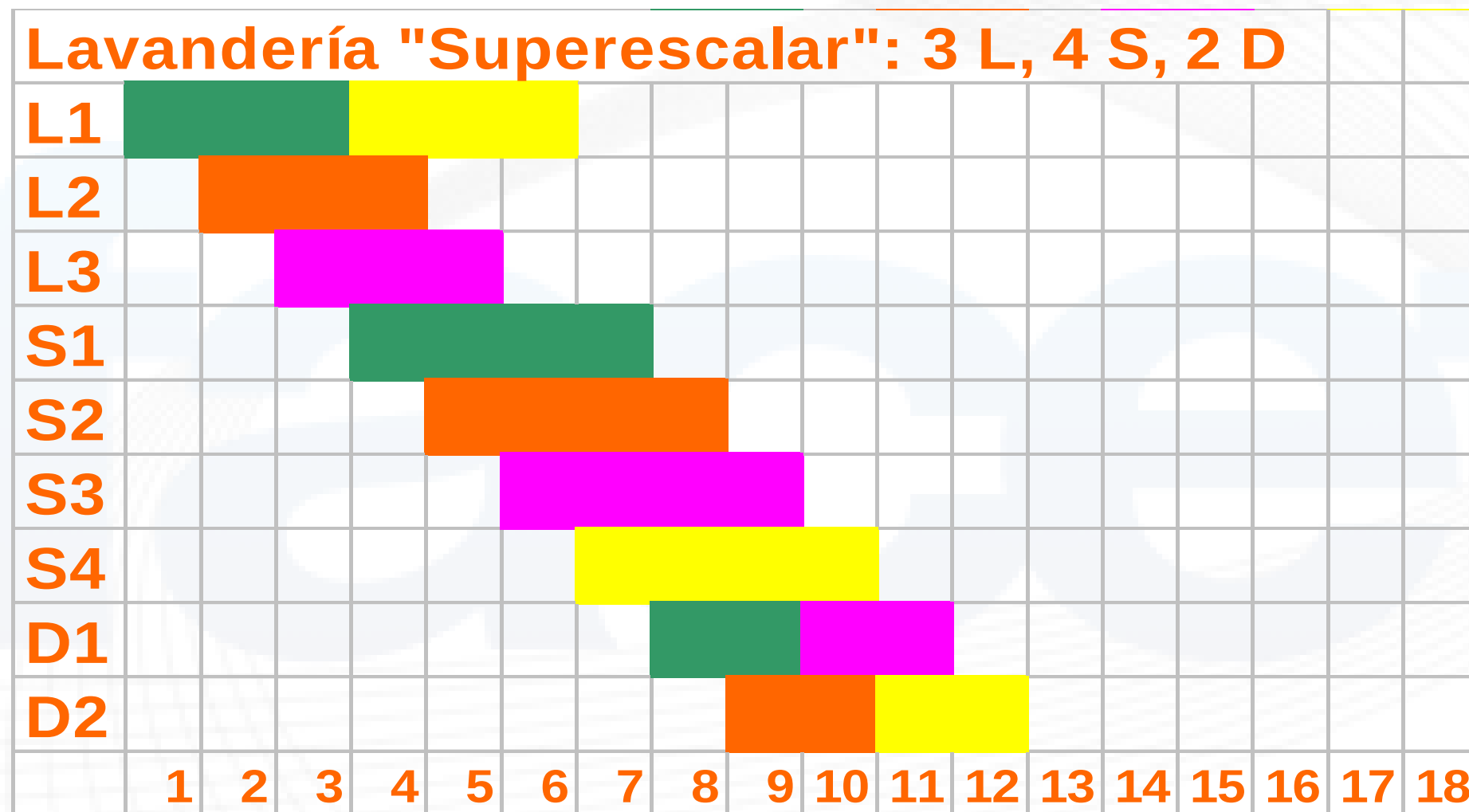
Cuellos de botella – Solución paralela



- ▶ *¿Nueva productividad?*
- ▶ Si se usa una alternativa en serie o en paralelo, depende exclusivamente de la tarea en sí.

Cuellos de botella – Solución extrema

- ▶ Al solucionar un cuello de botella, aparece otro. Repetimos la estrategia:



- ▶ *¿Nueva productividad?*

Cuellos de botella – Solución extrema

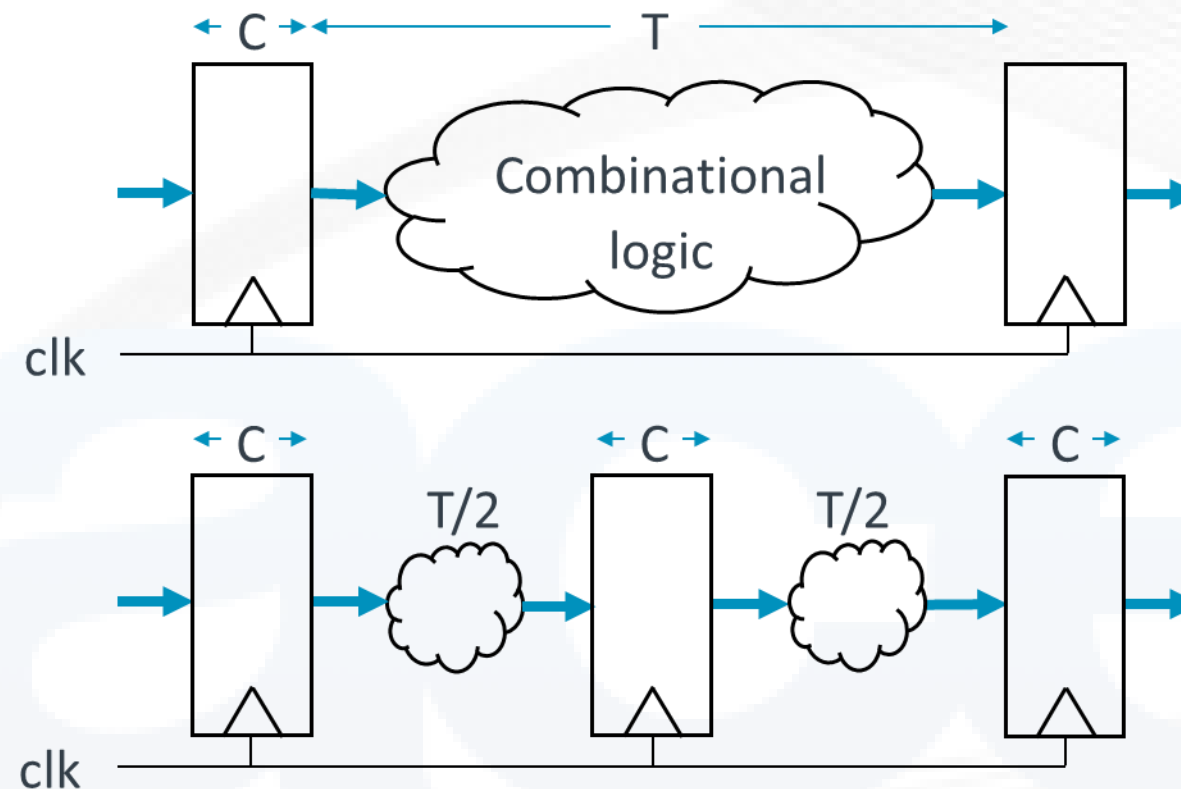
- ▶ La productividad es máxima.
 - ▶ Limitada por el m.c.d. de las duraciones de los componentes.
- ▶ Aumenta significativamente el costo.
 - ▶ Dependerá de los objetivos si vale la pena o no.
- ▶ En microprocesadores, esta solución se conoce como “superescalar”.
 - ▶ La veremos con más detalle en el Tema 08.

Pipelining – Tareas diferentes

- ▶ Suponga una secuencia de tareas de distinto tipo:
 - ▶ Las tareas tipo 1 usan todas las etapas.
 - ▶ Las tareas tipo 2 usan etapas 1, 3 y 5.
 - ▶ Las tareas tipo 3 usan etapas 1, 2 y 5.
- ▶ ¿Alternativas de solución?
 - ▶ En todas se producen choques. Riesgos estructurales.
- ▶ Conviene hacer que todas las tareas pasen por todas las etapas
 - ▶ Aunque no se haga nada útil en las mismas.
- ▶ Ejemplo típico de una mezcla de instrucciones de un procesador.

Pipelining – Ejemplo con circuitos

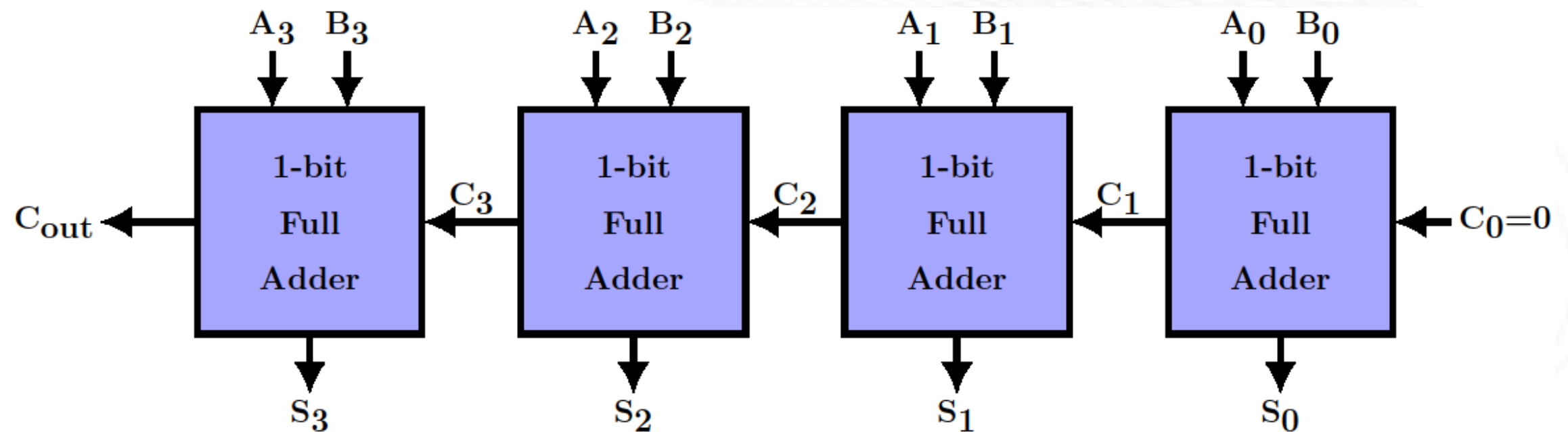
- ▶ Dado un circuito combinacional que posee un retardo T , con un delay de registros C , el tiempo de ciclo sería $T+C$.



- ▶ Se podría agregar un registro intermedio, para dividir la lógica combinacional en dos mitades, y el nuevo tiempo de ciclo pasaría a ser $T/2+C$.
 - ▶ Si C es pequeño, la frecuencia casi que se duplica.

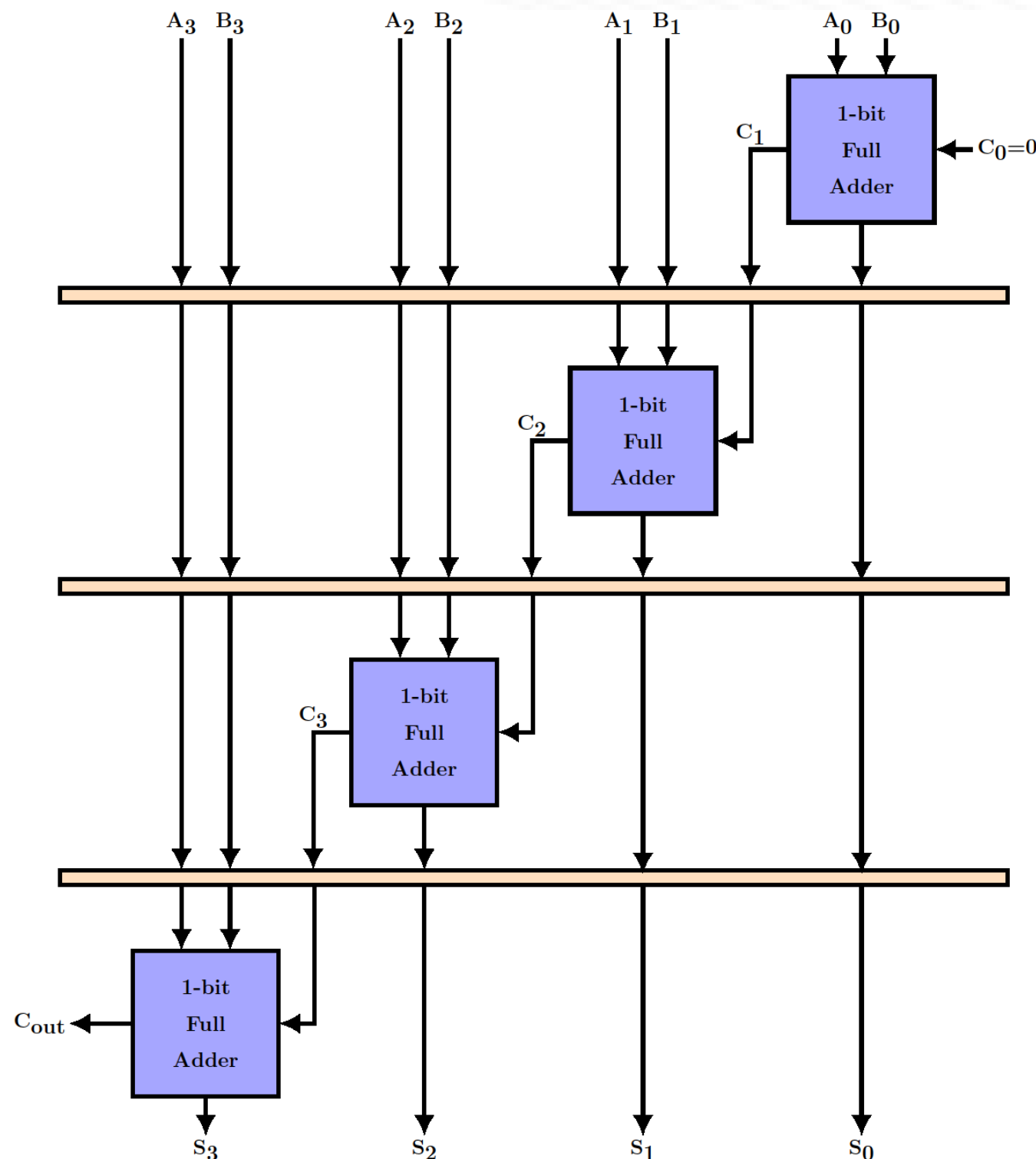
Pipelining – Ejemplo con circuitos

- ▶ Dado un sumador de 4 bits en cascada:



- ▶ Suponiendo que cada sumador de 1 bit tiene un retardo lógico igual a T , *¿cuánto se demora en entregar un resultado estable?*
- ▶ Suponiendo que mi tarea consiste en realizar 1000 sumas, *¿cuánto demoraría?*

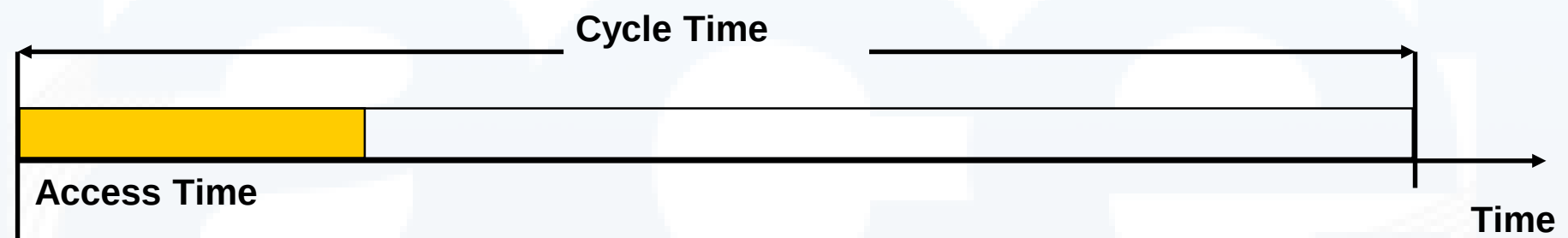
Pipelining – Sumador segmentado



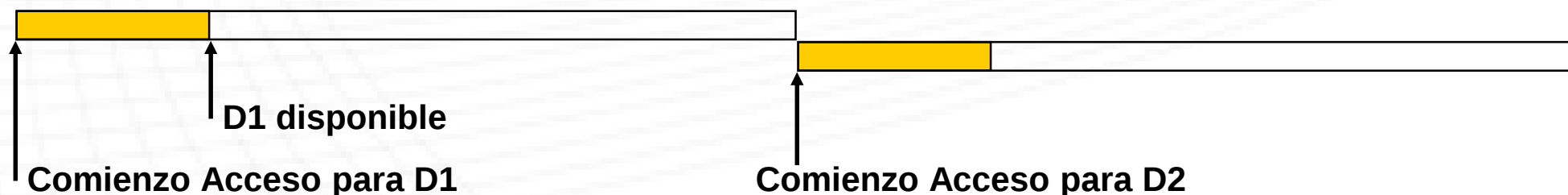
- ▶ Se agregan registros intermedios (buffers) para separar en etapas.
- ▶ En cada etapa los datos siempre se toman desde un registro, y terminan en otro registro.
- ▶ Suponemos que estos registros tienen un retardo despreciable.
- ▶ *¿Latencia de una suma?*
- ▶ *¿Latencia de las 1000 sumas?*
- ▶ *¿Productividad?*
- ▶ **Atención:** no siempre es posible despreciar el retardo de los buffers.

Paralelismo en memorias DRAM

- ▶ Las memorias DRAM tienen un tiempo de ciclo.
 - ▶ Indica cada cuánto se puede iniciar un acceso.
- ▶ También tienen un tiempo de acceso.
 - ▶ Indica cuándo está disponible un dato, desde que se inició el acceso.
- ▶ Normalmente el tiempo de acceso es mucho menor que el tiempo de ciclo.

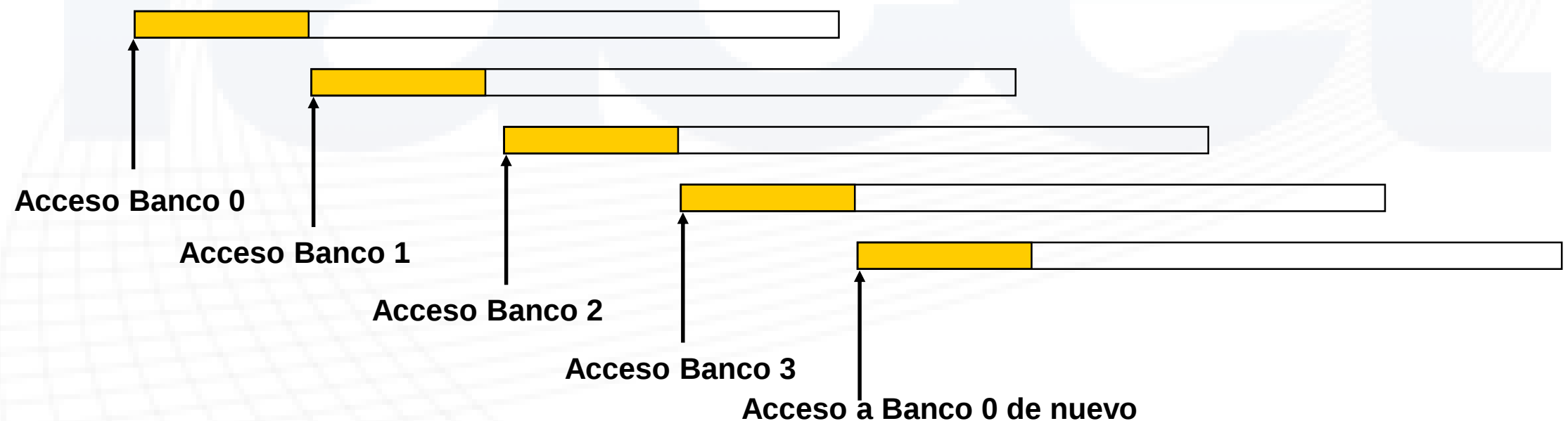


- ▶ Si se tiene un solo módulo de memoria, los accesos serían secuenciales:



Interleaving de memoria

- ▶ Se busca optimizar los accesos a memoria dividiéndola en varios bancos, para aprovechar la diferencia entre tiempo de acceso y tiempo de ciclo.
- ▶ Mientras se espera el tiempo de ciclo, se accede a otro banco.
- ▶ La cantidad de bancos óptima viene dada por la relación entre tiempo de ciclo y tiempo de acceso.
- ▶ Los bancos son conjuntos disjuntos, no copias.
- ▶ Por ejemplo, con cuatro módulos de memoria entrelazados sería algo así:



Consideraciones sobre *Interleaving*

- ▶ Se usan los bits menos significativos de la dirección para seleccionar el banco correspondiente.
- ▶ Aumenta el ancho de banda a memoria.
- ▶ Similar a un pipeline de 4 etapas.
- ▶ Ideal con accesos consecutivos a memoria.
 - ▶ *¿Qué pasa si los accesos no son consecutivos?*
- ▶ Con un factor de *interleaving* de 100 y con accesos al azar, en promedio mejora 12 veces.
 - ▶ *¿Cuánto mejoraría en este caso si los accesos fueran consecutivos?*

Recapitulación

- ▶ Paralelismo y Pipelining son muy útiles cuando hay muchas tareas repetitivas.
- ▶ Paralelismo
 - ▶ Hay que multiplicar recursos.
 - ▶ Tareas **independientes** en un momento dado.
 - ▶ Tasa de salida ideal = $m * \text{tasa}(\text{serie})$
 - ▶ Con adecuado perfil de tarea.
- ▶ Pipelining
 - ▶ Hay que agregar mínimos recursos.
 - ▶ Tareas **secuencialmente dependientes**.
 - ▶ Tasa de salida ideal = $m * \text{tasa}(\text{serie})$
 - ▶ Si las etapas están equilibradas, y trabajando en estado de régimen.
- ▶ *¿Puedo hacer que m sea tan grande como se quiera?*

Coordinación y Validez

- ▶ Dividir demasiado las tareas altera el modelo
 - ▶ ¿Se puede poner menos de un ladrillo?
 - ▶ ¿Ajustar menos una tuerca?
 - ▶ Los retardos de los buffers intermedios dejan de ser despreciables.
- ▶ Se necesitan barreras de sincronización
 - ▶ Esta sincronización debe estar expresada en el perfil de la tarea.
 - ▶ Todas las paredes deben estar listas antes de poner el techo.
 - ▶ Recordar Ley de Amdahl.
- ▶ *“Nueve mujeres no pueden tener un bebé en un mes”.*

Resumen final

- ▶ Métricas importantes:
 - ▶ Productividad, Latencia, Aceleración y Rendimiento.
- ▶ Herramientas importantes:
 - ▶ Perfil de la tarea.
 - ▶ Diagrama de dependencias.
- ▶ Pipelining ideal: 1 tarea en cada T.
 - ▶ Problemas: cuellos de botella, diferentes tareas, pocas tareas.
- ▶ Paralelismo + Pipelining = Superescalar.
- ▶ Ejemplos:
 - ▶ Sumador segmentado
 - ▶ Interleaving de memoria.

Agradecimientos

- ▶ Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Daniel Cohen.
- ▶ A su vez inspiradas en las clases del curso CS152 de la Universidad de Berkeley, California, USA.
 - ▶ Realizadas por los Prof. D. A. Patterson, John Lazzaro, Krste Asanovic.