

# ALGORITMOS PARALELOS


## *FUNDAMENTOS DE DISEÑO*

[ABOUT ACM](#) [MEMBERSHIP](#) [PUBLICATIONS](#) [SIGS](#) [CONFERENCES](#) [CHAPTERS](#) [AWARDS](#) [EDUCATION](#) [LEARNING CENTER](#) [PUBLIC POLICY](#) [DIVERSITY, EQUITY & INCLUSION](#)

Association for Computing Machinery

## Advancing Computing as a Science & Profession

We see a world where computing helps solve tomorrow's problems – where we use our knowledge and skills to advance the profession and make a positive impact.



AWARDS & RECOGNITION

### [Ian Foster Recognized with Ken Kennedy Award](#)

Ian Foster, a Professor at the University of Chicago and Division Director at Argonne National Laboratory, has been named the recipient of the 2022 [ACM-IEEE CS Ken Kennedy Award](#). The Ken Kennedy Award recognizes pathbreaking achievements in parallel (high-performance) computing. Foster is cited for contributions to programming and productivity in computing via the establishment of new programming models and foundational science services. The award will be formally presented to Foster in November at [The International Conference for High-Performance Computing, Networking, Storage and Analysis \(SC22\)](#).

[Abrir el archivo adjunto](#)

¿ Le suena de algo? Revise la bibliografía

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

**Paralelizar.** Dividir el problema en subproblemas susceptibles de resolverse a la vez (concurrentes).

## Objetivos

- Desarrollar algoritmos paralelos de forma sistemática y metódica.
- Reducir el impacto de la “inspiración” del programador.
- Enfocar el problema desde la definición: no considerar la “transformación del secuencial” como único punto de partida.

## Nivel de complejidad superior al secuencial

- La concurrencia (comunicar/sincronizar).
- La asignación de datos/programas a procesadores.
- La escalabilidad, etc.

## El rendimiento dependerá (entre otras):

- Del balanceado: que todas las tareas sean de igual tamaño.
- De la concurrencia: que se ejecuten todas a la vez.
- De las dependencias: minimizar las dependencias entre tareas.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
<pre>a = 0.0; b = 0.0; for (i=0; i&lt;n; i++)   a = a + x[i];  for (i=0; i&lt;n; i++)   b = b + y[i];  for (i=0; i&lt;n; i++)   z[i] = x[i]/b + y[i]/a;  for (i=0; i&lt;n; i++)   y[i] = (a+b)*y[i];</pre>

Sea el coste computacional por iteración de cada uno de los 4 bucles 1, 1, 3 y 2 *flop*, respectivamente.

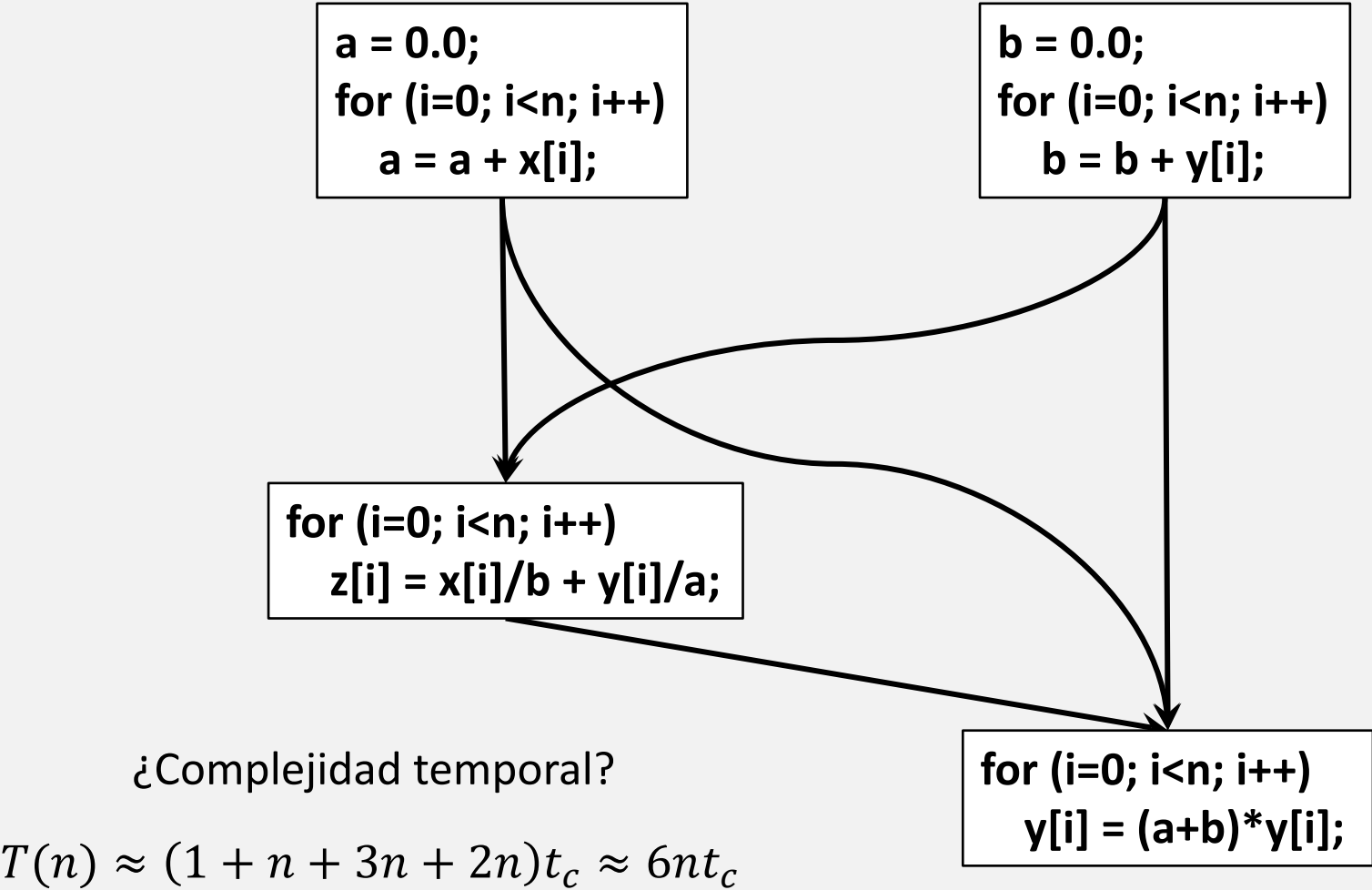
Algoritmo secuencial ¿complejidad temporal?

$$T(n) = (2 + n + n + 3n + 2n)t_c \approx 7nt_c$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
a = 0.0; b = 0.0; for (i=0; i<n; i++) a = a + x[i];  for (i=0; i<n; i++) b = b + y[i];  for (i=0; i<n; i++) z[i] = x[i]/b + y[i]/a;  for (i=0; i<n; i++) y[i] = (a+b)*y[i];



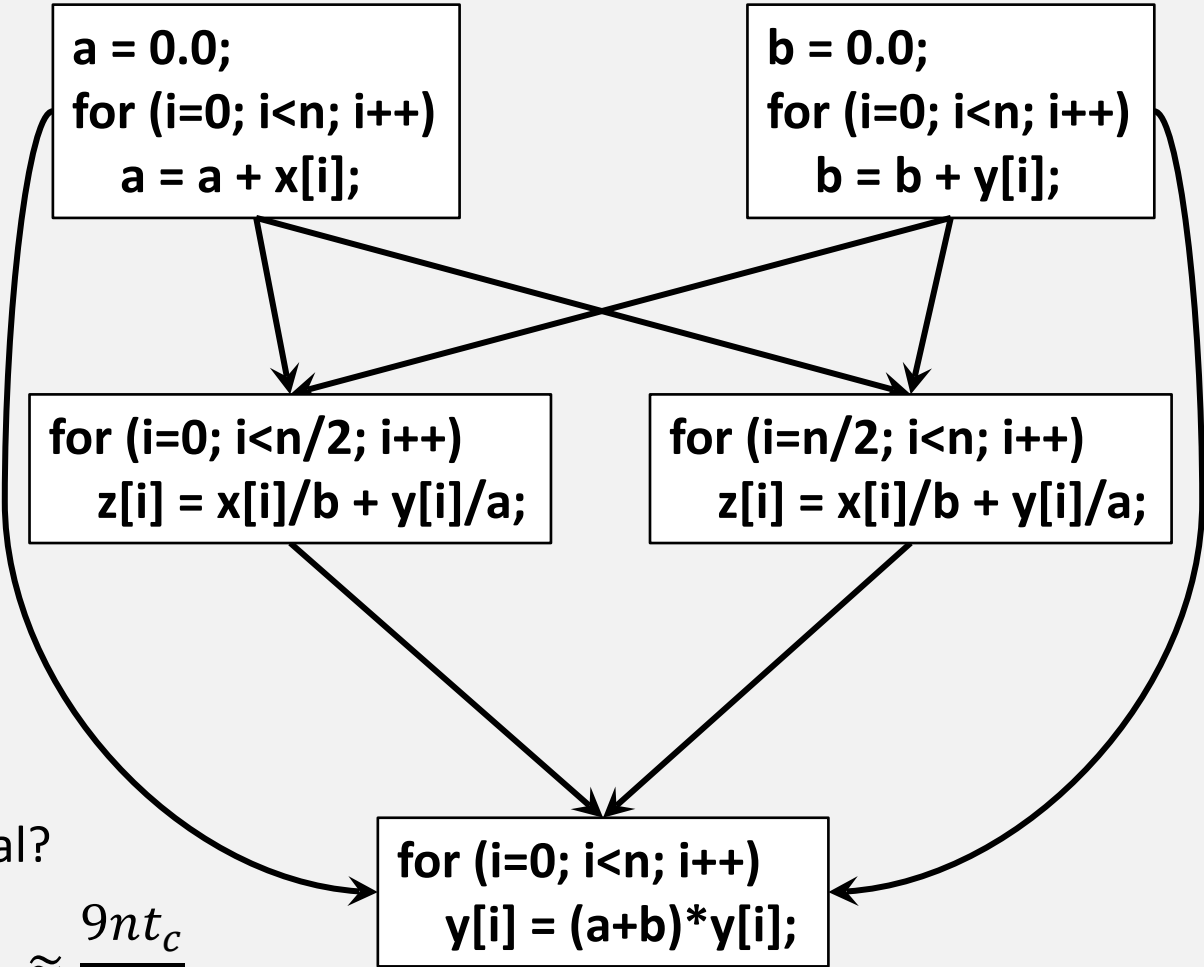
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
a = 0.0; b = 0.0; for (i=0; i<n; i++) a = a + x[i];  for (i=0; i<n; i++) b = b + y[i];  for (i=0; i<n; i++) z[i] = x[i]/b + y[i]/a;  for (i=0; i<n; i++) y[i] = (a+b)*y[i];

¿Complejidad temporal?

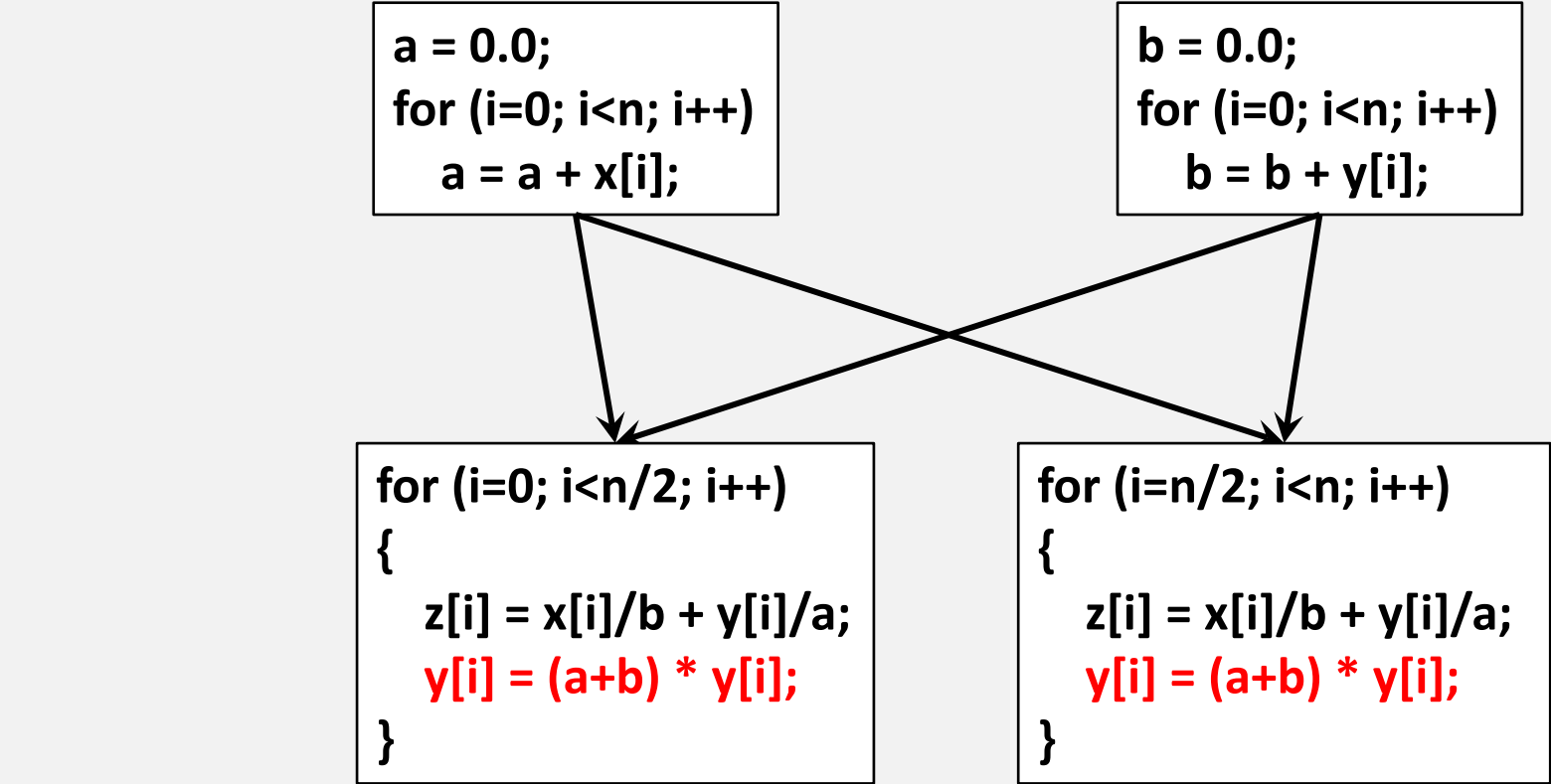
$$T(n) \approx \left( 1 + n + \frac{3n}{2} + 2n \right) t_c \approx \frac{9nt_c}{2}$$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
a = 0.0; b = 0.0; for (i=0; i<n; i++) a = a + x[i];  for (i=0; i<n; i++) b = b + y[i];  for (i=0; i<n; i++) z[i] = x[i]/b + y[i]/a;  for (i=0; i<n; i++) y[i] = (a+b)*y[i];



¿Complejidad temporal?

$$T(n) \approx \left(1 + n + \frac{5n}{2}\right) t_c \approx \frac{7nt_c}{2}$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

## Estrategias

- Grafos de dependencias: descomposición, análisis y optimización, asignación, etc.
- Esquemas Algorítmicos
  - Un esquema se puede aplicar a varios problemas y un problema puede requerir varios esquemas.
  - El alumno ya conoce una gran variedad de esquemas (Algoritmia, Paradigmas, ED, etc.).
    - Árboles y grafos (divide y vencerás).
    - Programación dinámica, ramificación y poda, etc.
    - Algoritmos de relajación y Maestro-Esclavo (asíncronos).
  - Pero otros más específicos no, como, por ejemplo:
    - **Paralelismo segmentado (*pipelining*) y síncrono (sistólicos).**
    - **Granja Procesos, Trabajadores Replicados, etc.**
    - **Paralelismo de datos.**
- Enfoques Metodológicos (p. ej. Foster I.)
- Etc.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

## Dependencias

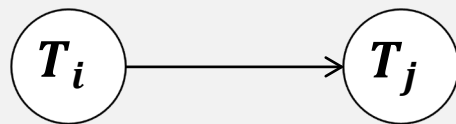
- Se puede conocer analizando los datos de entrada/salida.

## Condiciones de Bernstein

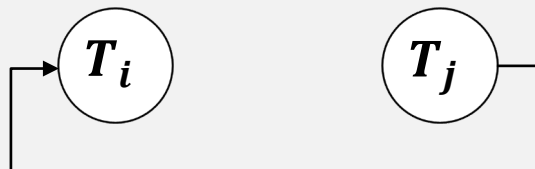
- Sean  $T_i$  y  $T_j$  dos tareas tal que  $T_i$  precede a  $T_j$ . Sean  $I_i$  y  $O_i$  los conjuntos que representan las variables leídas y escritas, respectivamente, por la tarea  $i$ . De forma análoga,  $I_j$  y  $O_j$  para la tarea  $j$ . Se dice que  $T_i$  y  $T_j$  son independientes  $\Leftrightarrow I_j \cap O_i = \emptyset, I_i \cap O_j = \emptyset$  y  $O_j \cap O_i = \emptyset$

## Tipos

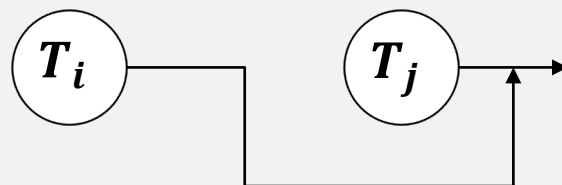
- De flujo ( $I_j \cap O_i \neq \emptyset$ )



- Anti-dependencia ( $I_i \cap O_j \neq \emptyset$ )



- De salida ( $O_j \cap O_i \neq \emptyset$ )





# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

EJEMPLO: DEPENDENCIA DE FLUJO

```
for (i=1; i<n; i++)  
    b[i] = b[i] + a[i-1];  
    a[i] = a[i] + c[i];
```

- La iteración *i-ésima* modifica a[i] que es leída por la iteración *i+1*.
- Algunas se pueden eliminar y otras son difíciles.

```
b[1] = b[1] + a[0];  
a[1] = a[1] + c[1];  
b[2] = b[2] + a[1];  
a[2] = a[2] + c[2];  
b[3] = b[3] + a[2];  
a[3] = a[3] + c[3];  
...  
b[n] = b[n] + a[n-1];  
a[n] = a[n] + c[n];
```

SIN DEPENDENCIA DE FLUJO

```
b[1]=b[1] + a[0]  
for (i=1; i<n-1; i++)  
    a[i]=a[i]+c[i]  
    b[i+1]=b[i+1]+a[i];  
a[n-1]=a[n-1] + c[n-1];
```

```
b[1] = b[1] + a[0];  
a[1] = a[1] + c[1];  
b[2] = b[2] + a[1];  
a[2] = a[2] + c[2];  
b[3] = b[3] + a[2];  
a[3] = a[3] + c[3];  
b[4] = b[4] + a[3];  
...  
b[n] = b[n] + a[n-1];  
a[n] = a[n] + c[n];
```

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

## Grafo de Dependencias

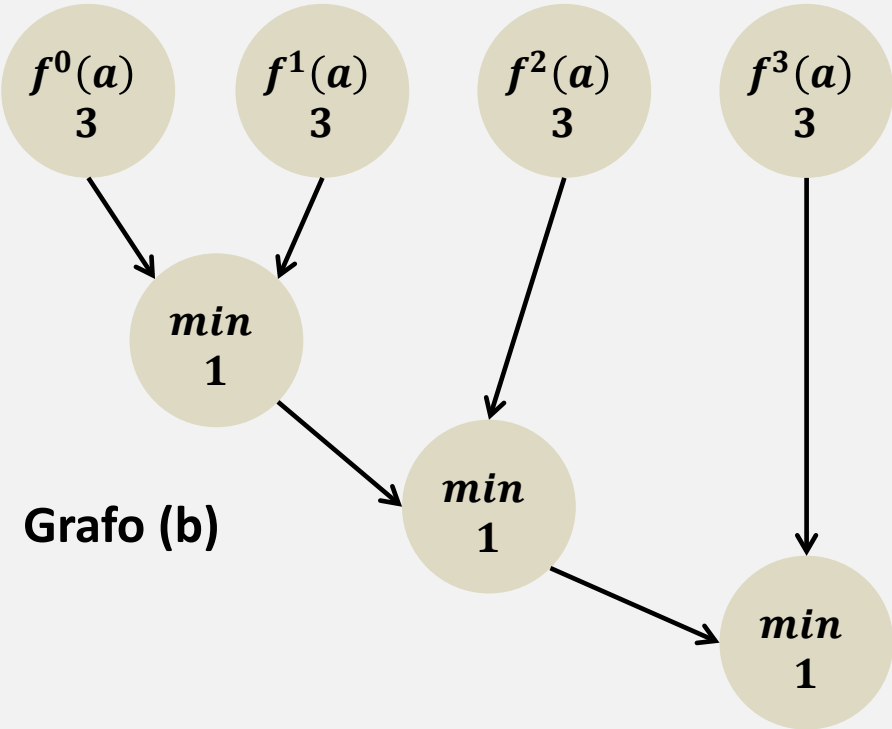
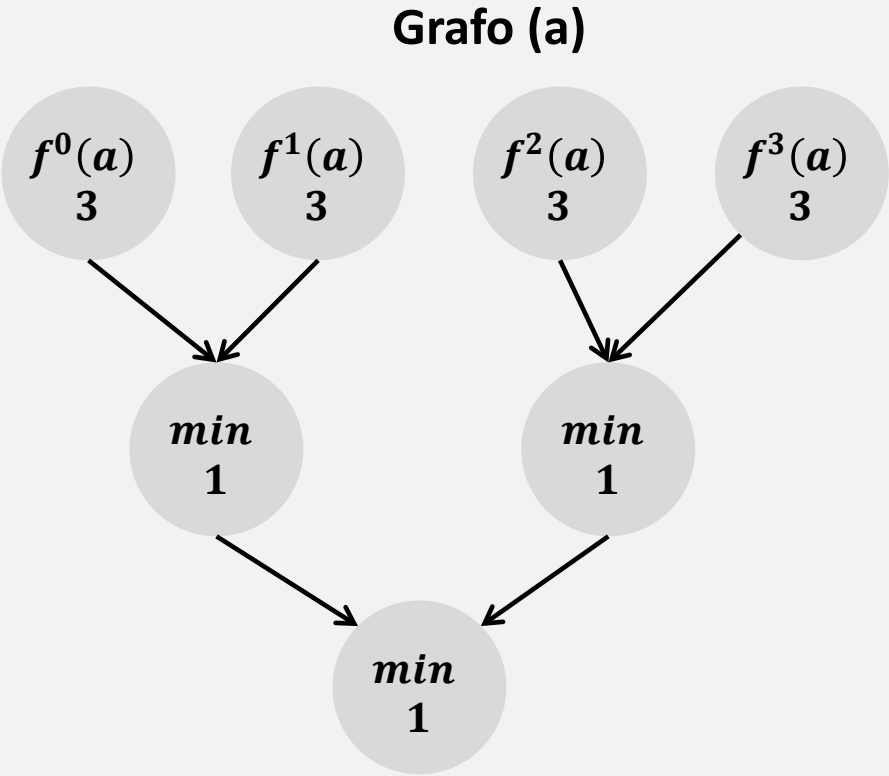
- Abstracción utilizada para expresar dependencias entre las tareas y, consecuentemente, su orden relativo de ejecución.

## Características

- Grafo dirigido acíclico. Los nodos denotan tareas y las aristas conectan una tarea fuente con otra destino. La semántica de la arista es *“para que la tarea destino se pueda ejecutar tiene que haberse ejecutado previamente la tarea fuente”*.
- A veces las aristas representan comunicaciones. En este caso las aristas se etiquetan con pesos representativos del volumen de datos que comunican o mueven.
- Cada nodo del grafo suele etiquetarse con un valor proporcional al coste computacional de la tarea.
- Varios grafos para el mismo problema según el planteamiento seguido.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Ejemplo



## FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

- **Longitud de un camino.** Suma de los costes asociados a los nodos (y/o los arcos) que lo componen.
- **Camino Crítico  $L(G)$ .** Camino más largo (*más costoso*) entre cualquier nodo inicial y cualquiera final.
- **Grado Medio de Concurrencia  $M(G)$ .** Número medio de tareas que se pueden ejecutar en paralelo:

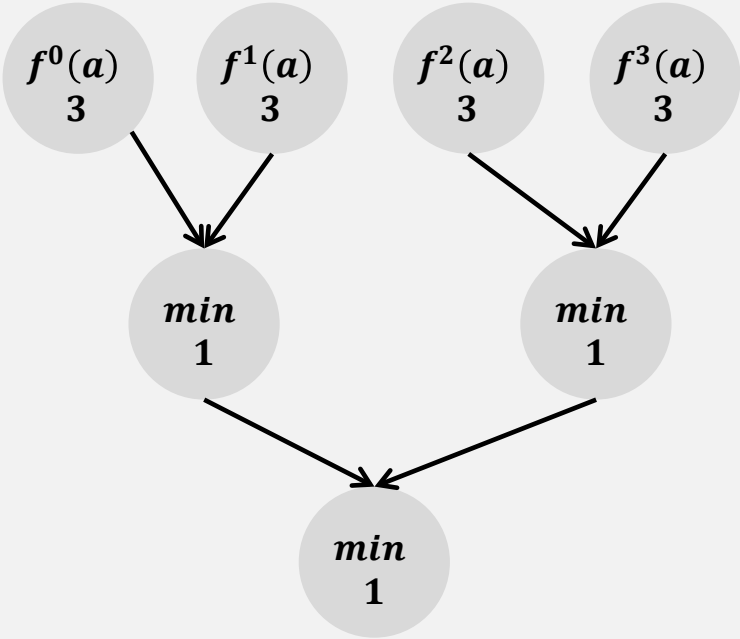
$$M(G) = \sum_{i=1}^N \frac{c_i}{L(G)} = \frac{1}{L(G)} \sum_{i=1}^N c_i$$

siendo  $N$  el número de nodos del grafo  $G$  y  $c_i$  el coste asociado al nodo  $i$ .

- **Máximo Grado de Concurrencia.** Número máximo de tareas cuya ejecución se puede realizar simultáneamente en el grafo de dependencias.

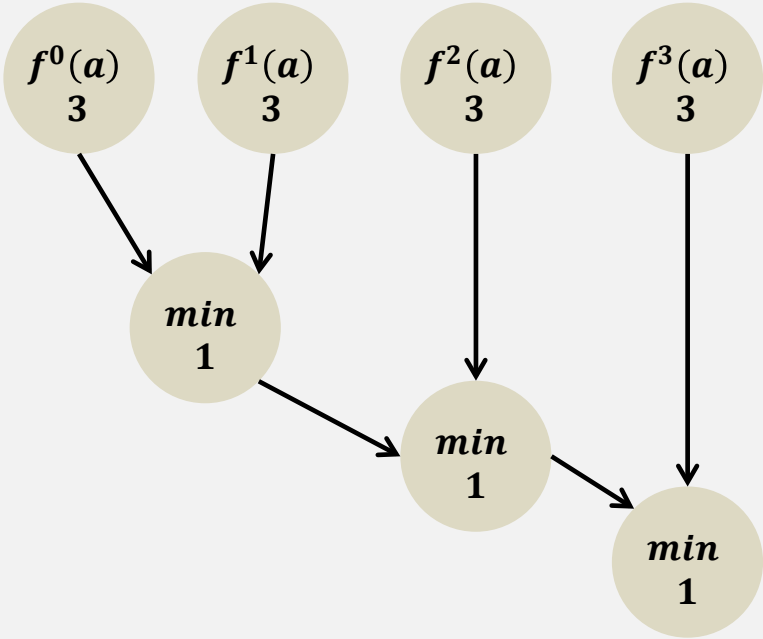
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Grafo (a)



$L(a) = 5$

Grafo (b)



$L(b) = 6$

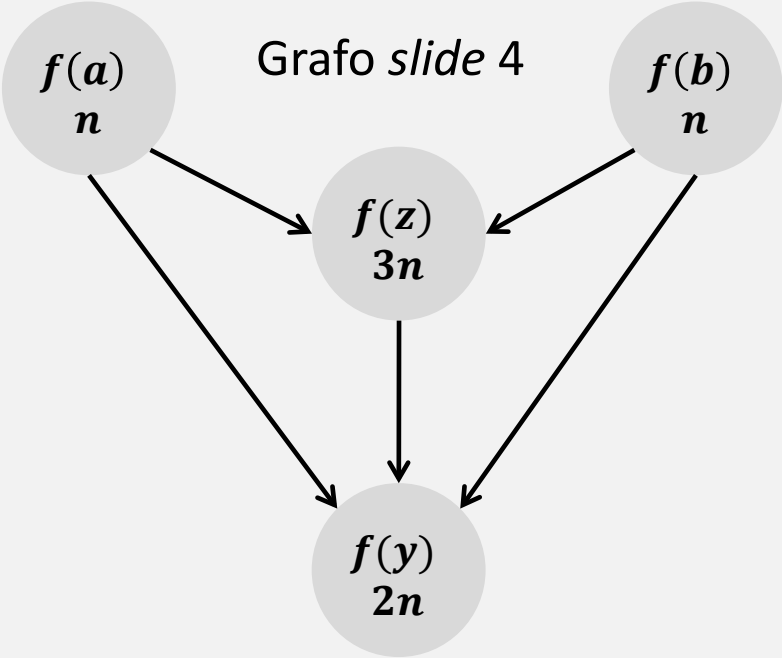
$$M(a) = \frac{3 + 3 + 3 + 3 + 1 + 1 + 1}{5} = \frac{15}{5} = 3$$

$$M(b) = \frac{3 + 3 + 3 + 3 + 1 + 1 + 1}{6} = \frac{15}{6} = 2.5$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
<pre>a = 0.0; b = 0.0; for (i=0; i&lt;n; i++)   a = a + x[i];  for (i=0; i&lt;n; i++)   b = b + y[i];  for (i=0; i&lt;n; i++)   z[i] = x[i]/b + y[i]/a;  for (i=0; i&lt;n; i++)   y[i] = (a+b)*y[i];</pre>



$$L = n + 3n + 2n = 6n$$
$$M = \frac{n + n + 3n + 2n}{n + 3n + 2n} = \frac{7}{6} \cong 14\%$$

FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo Ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

**Ejemplo:**

```
a = 0.0; b = 0.0;
for (i=0; i<n; i++)
  a = a + x[i];

for (i=0; i<n; i++)
  b = b + y[i];

for (i=0; i<n; i++)
  z[i] = x[i]/b + y[i]/a;

for (i=0; i<n; i++)
  y[i] = (a+b)*y[i];
```

```
a = 0.0; for (i=0; i<n; i++) a = a + x[i];
b = 0.0; for (i=0; i<n; i++) b = b + y[i];
for (i=0; i<n; i++) z[i] = x[i]/b + y[i]/a;
for (i=0; i<n; i++) y[i] = (a+b)*y[i];
```

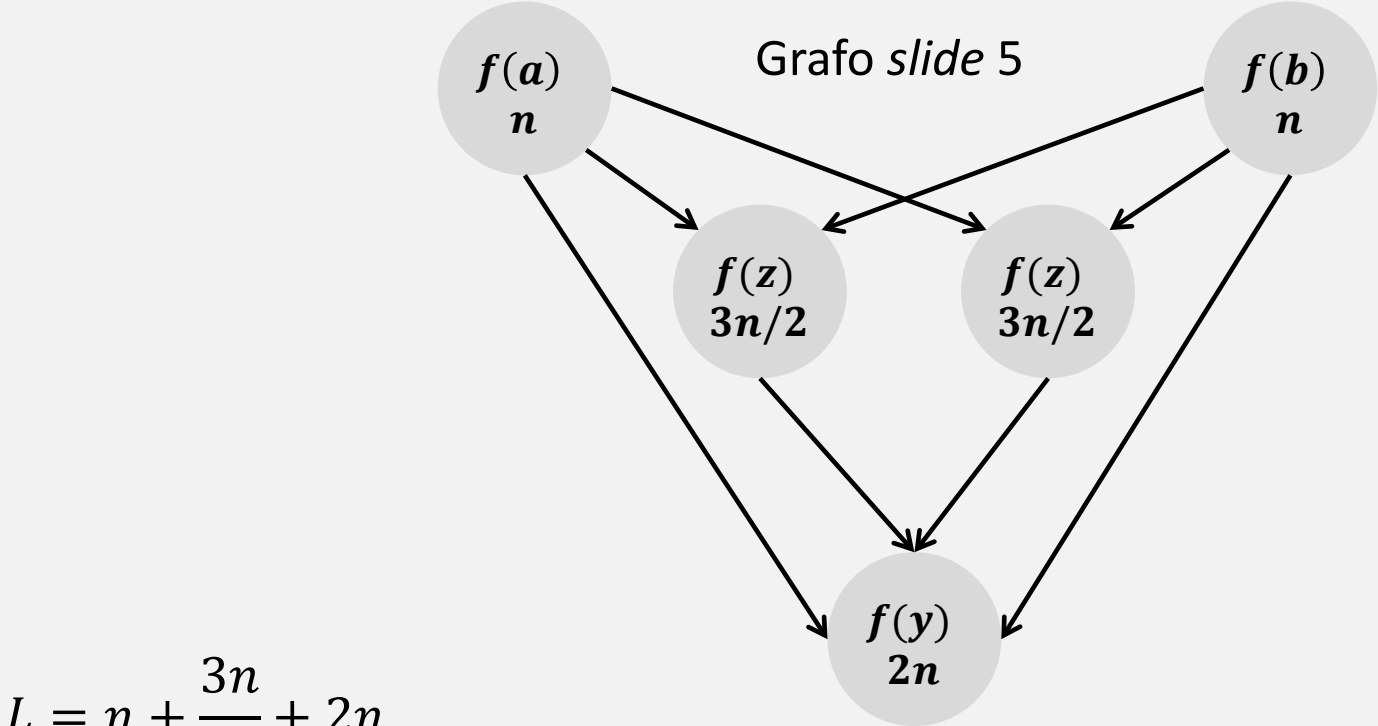
¿Complejidad temporal?

$T(n) \approx (1 + n + 3n + 2n)T_c \approx 6nT_c$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
<pre>a = 0.0; b = 0.0; for (i=0; i&lt;n; i++)   a = a + x[i];  for (i=0; i&lt;n; i++)   b = b + y[i];  for (i=0; i&lt;n; i++)   z[i] = x[i]/b + y[i]/a;  for (i=0; i&lt;n; i++)   y[i] = (a+b)*y[i];</pre>



$$L = n + \frac{3n}{2} + 2n$$
$$M = \frac{n + n + \frac{3n}{2} + \frac{3n}{2} + 2n}{n + \frac{3n}{2} + 2n} = \frac{7}{4.5} \cong 35\%$$

FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo Ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

**Ejemplo:**

```
a = 0.0; b = 0.0;
for (i=0; i<n; i++)
  a = a + x[i];

for (i=0; i<n; i++)
  b = b + y[i];

for (i=0; i<n; i++)
  z[i] = x[i]/b + y[i]/a;

for (i=0; i<n; i++)
  y[i] = (a+b)*y[i];
```

¿Complejidad temporal?

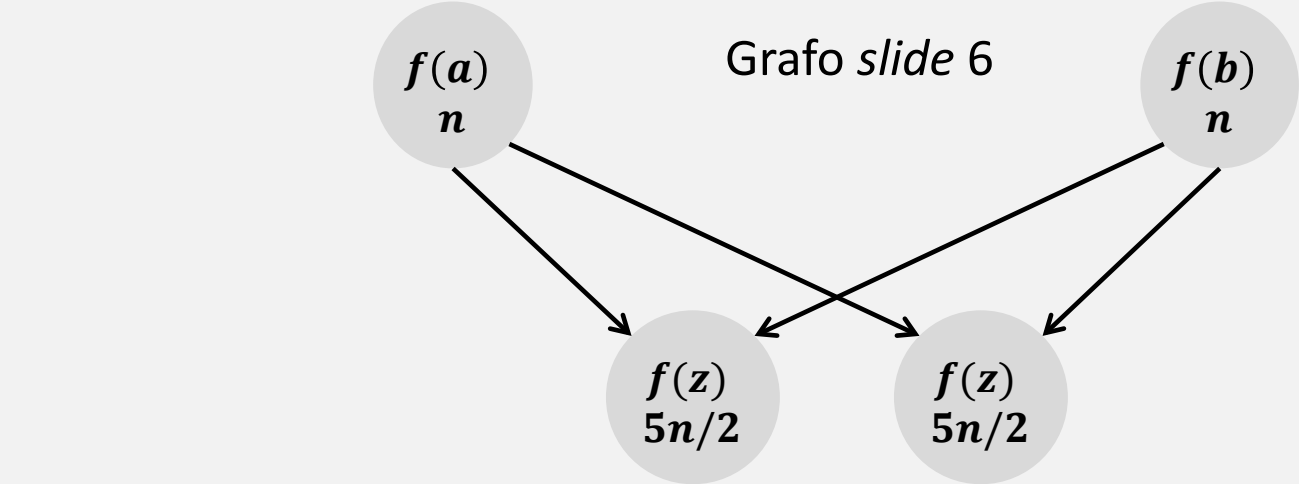
$T(n) \approx \left(1 + n + \frac{3n}{2} + 2n\right) t_c \approx \frac{9nt_c}{2}$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Ejemplo ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

EJEMPLO
<pre>a = 0.0; b = 0.0; for (i=0; i&lt;n; i++)   a = a + x[i];  for (i=0; i&lt;n; i++)   b = b + y[i];  for (i=0; i&lt;n; i++)   z[i] = x[i]/b + y[i]/a;  for (i=0; i&lt;n; i++)   y[i] = (a+b)*y[i];</pre>



$$L = n + \frac{5n}{2}$$
$$M = \frac{n + n + \frac{5n}{2} + \frac{5n}{2}}{n + \frac{5n}{2}} = \frac{7}{3.5} \cong 50\%$$

FUNDAMENTOS DEL DISEÑO DE ALGORITMOS

Ejemplo Ilustrativo de qué implica “Balanceado + Concurrencia + Dependencias”.

**Ejemplo:**

```
a = 0.0; b = 0.0;
for (i=0; i<n; i++)
  a = a + x[i];

for (i=0; i<n; i++)
  b = b + y[i];

for (i=0; i<n; i++)
  z[i] = x[i]/b + y[i]/a;

for (i=0; i<n; i++)
  y[i] = (a+b)*y[i];
```

```
a = 0.0;
for (i=0; i<n; i++)
  a = a + x[i];

b = 0.0;
for (i=0; i<n; i++)
  b = b + y[i];

for (i=0; i<n; i++)
  z[i] = x[i]/b + y[i]/a;

for (i=0; i<n; i++)
  y[i] = (a+b)*y[i];
```

¿Complejidad temporal?

$T(n) \approx \left(1 + n + \frac{5n}{2}\right) T_c \approx \frac{7nT_c}{2}$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Sea  $s = \min_{i=0 \dots m-1} \{P_i(b)\}$  donde  $P_i(x) = a_{i,0} + a_{i,1}x + a_{i,2}x^2 + \dots + a_{i,n}x^n, i = 0 \dots m - 1$ . Considerando que los coeficientes de los polinomios están almacenados en una matriz de la siguiente manera,

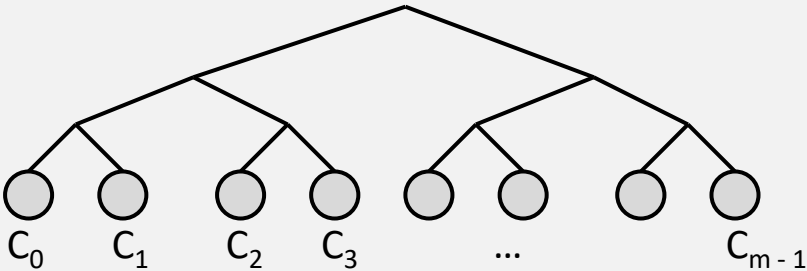
$$A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,n} \end{pmatrix}$$

Plantear un posible grafo de dependencias. Una solución:

- 1. Asignar la evaluación de cada polinomio a una tarea

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$\dots$	$a_{0,n}$	---> Tarea <sub>0</sub> : $c_0 = P_0(b)$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$\dots$	$a_{1,n}$	---> Tarea <sub>1</sub> : $c_1 = P_1(b)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	
$a_{m-1,0}$	$a_{m-1,1}$	$a_{m-1,2}$	$\dots$	$a_{m-1,n}$	---> Tarea <sub>m-1</sub> : $c_{m-1} = P_{m-1}(b)$

- 2. Combinar la solución: **Reducción**



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Resolver  $x = v + w$ , siendo  $v, w, x \in \mathbb{R}^n \wedge n > 0$

- Algoritmo Secuencial

```
for (i=0; i<n; i++)  
  x[i] = v[i] + w[i];
```

- Grafo de Dependencias



- Algoritmo paralelo con un número de procesadores  $p$  igual a  $n$ .

```
En cada procesador de ID  $i$  hacer  
  x[i] = v[i] + w[i];
```

- ¿ Algoritmo paralelo con un número de procesadores  $p < n$  ?

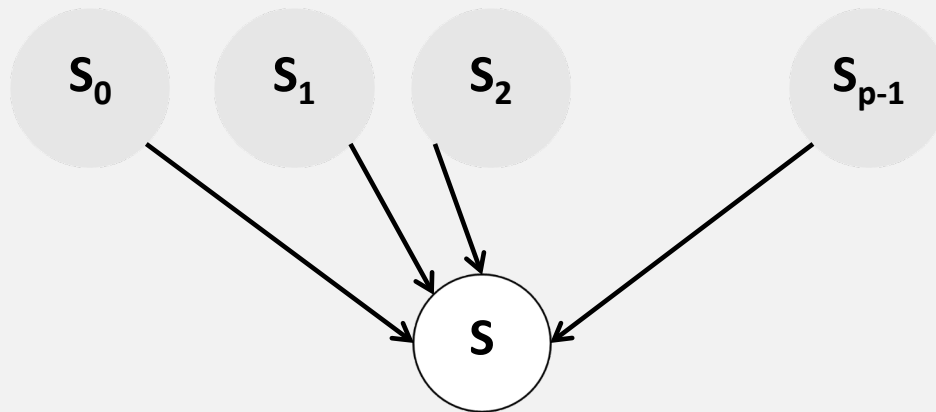
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Resolver  $s = \sum v_i$  con  $v \in \mathbb{R}^n$

- Algoritmo Secuencial

```
s=0  
for (i=0; i<n; i++)  
  s = s + v[i];
```

- Grafo de Dependencias (operador de reducción)



- Sea  $p \leq n$  y  $n$  divisible por  $p$  para que todos los procesadores realicen aproximadamente el mismo número de sumas parciales.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Resolver  $s = \sum v_i$  con  $v \in \mathbb{R}^n$

- Algoritmo paralelo  $p \leq n \wedge (n \bmod p = 0)$

```

En cada procesador Proc hacer (con Proc = 0 hasta  $p - 1$ )
{
  s[Proc] = 0
  for(i = n/p*Proc; i < n/p*(proc+1)-1; i++)
    s[Proc] = s[Proc] + v[i]

  "esperar por todos aquí"

  if (Proc == 0)      // haciendo la operación de reducción
  {
    sf = s[0]
    for (i=1; i<p; i++)
      sf = sf + s[i]
  }
}

```

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: GRAFOS DE DEPENDENCIAS

Resolver  $s = \sum v_i$  con  $v \in \mathbb{R}^n$

- Algoritmo paralelo  $p \leq n \wedge (n \bmod p = 0)$

En OpenMP

```
...
sum=0;
#pragma omp parallel private(local_sum)
{
    local_sum=0;
    #pragma omp for
    for(i=0; i<n; i++)
        local_sum +=v[i];

    #pragma omp atomic
    sum+=local_sum;
}
```

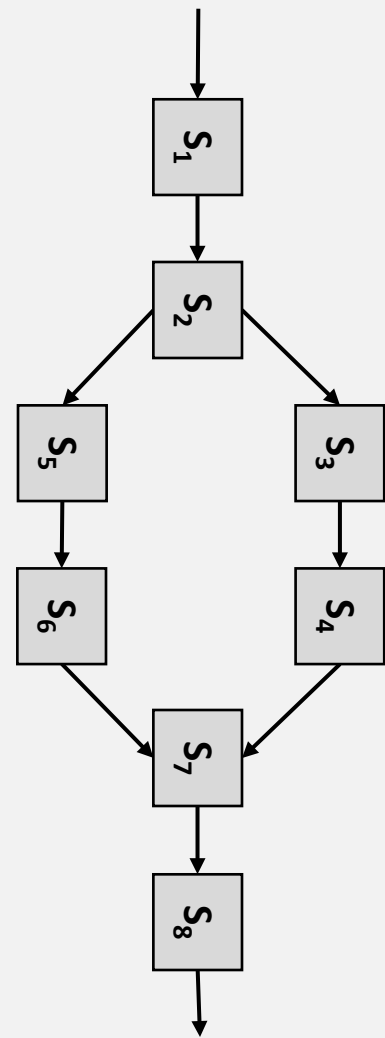
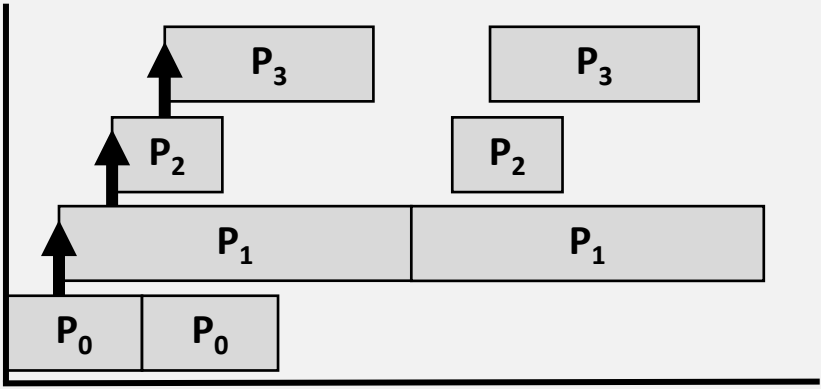
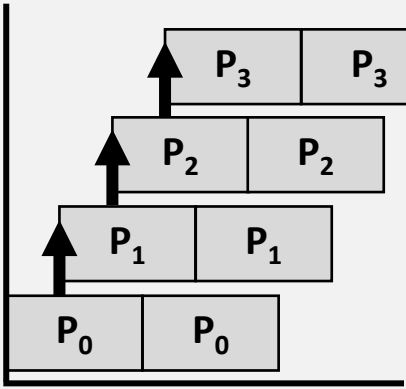
En OpenMP

```
...
sum=0;
#pragma omp parallel for reduction(+: sum)
for(i=0; i<n; i++)
    sum +=v[i];
```

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: PIPELINE

## Pipeline

- Conjunto ordenado de segmentos tales que la salida de uno es la entrada del siguiente.
- Pueden ejecutarse simultáneamente varios segmentos si la salida de unos es irrelevante para la entrada de los otros.
- Útil cuando se puede ejecutar más de una instancia, cuando una serie de datos debe ser tratada con múltiples operaciones o cuando la información para el siguiente proceso está disponible antes de que la necesite.
- Importante: **equidad** en la granularidad de los segmentos.





# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: PIPELINE

Resolver un sistema triangular usando la técnica pipeline.

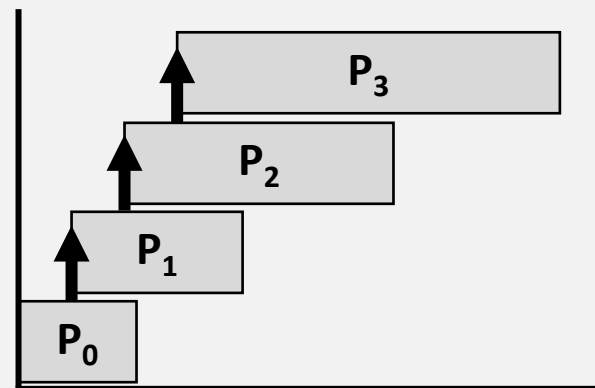
Complejidad temporal 
$$\sum_{i=1}^n \left( 2 + \sum_{j=1}^{i-1} 1 \right) = \frac{1}{2}n^2 + \frac{3}{2}n$$

- A cada  $P_i$  ( $1 \leq i \leq n$ ) se le asigna el proceso  $i$ : “obtener  $x_i$ ”.  $P_i$  necesita la salida de  $P_1, \dots, P_{i-1}$

```
Proceso pipeline(i)
  suma = 0
  para j=1 hasta (i-1) hacer
    recibir x[j] de la izquierda; enviar x[j] a la derecha
    suma = suma + a[i][j]*x[j]
  fin para
  x[i] = (b[i] – suma) / a[i][i]
  enviar x[i] a la derecha
```

Algoritmo secuencial

```
para i=1 hasta n hacer
  suma = 0
  para j=1 hasta (i-1) hacer
    suma = suma + a[i][j]*x[j]
  fin para
  x[i] = (b[i] – suma) / a[i][i]
fin para
```

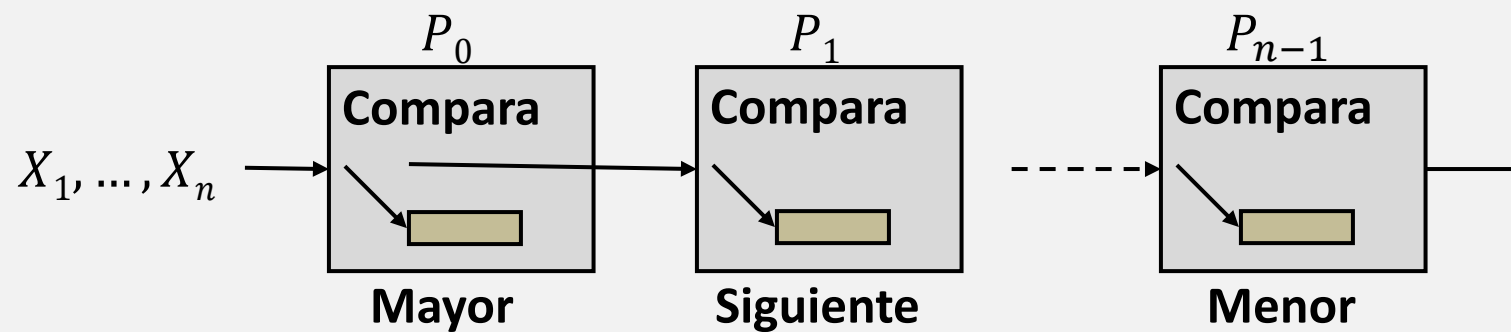


- Complejidad temporal  $\in O(n)$ . Comunicaciones y Computación del más lento de orden  $n$ . El Speedup  $0.37n$  (Lester) debido a la no equidad en la granularidad.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMO: PIPELINE

Ordenar un vector de  $n$  elementos usando la técnica pipeline.

- Complejidad secuencial  $\in O(n \log 2n)$ .

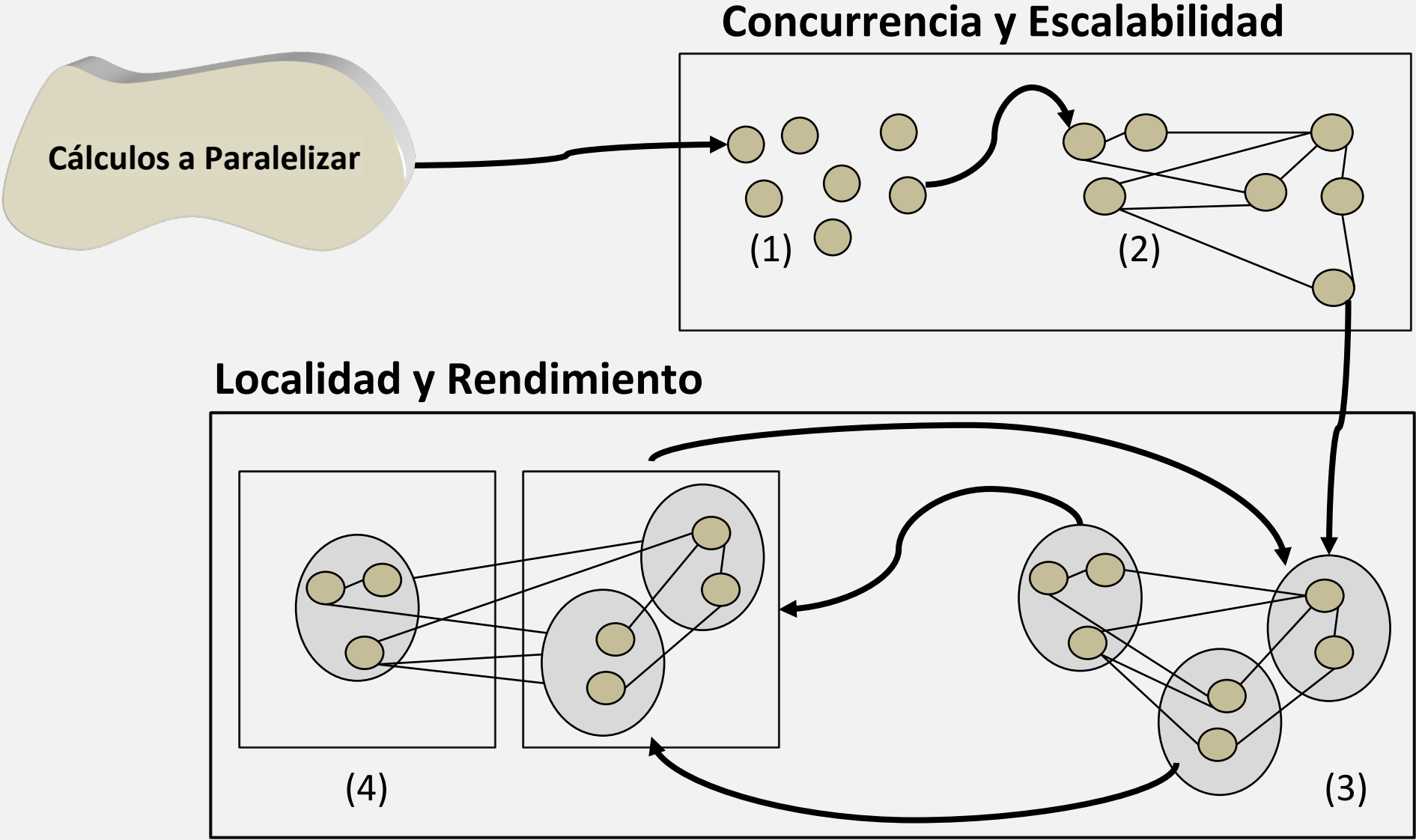


```

recibir( $P_{i-1}$ , numero)
si (numero > x)
    enviar( $P_{i+1}$ , x)
    x = numero
si no
    enviar( $P_{i+1}$ , numero)
    
```

- Complejidad  $\in O(n)$ . Eficiencia baja: 0.5 el valor máximo para  $n$  grande.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## 1. Descomposición

- En un **número elevado** de tareas con un grado de concurrencia alto.
- Flexibilidad. Considerar un computador “*ideal*”, sin limitaciones.
- Estática, las tareas se crean al inicio, o dinámica, en tiempo de ejecución.

### Interesa

- Obtener conjuntos disjuntos de computaciones y datos.
- Que el número de tareas “*escale*” al crecer el tamaño del problema.
- Que el peso de las tareas sea similar y que compartan poco datos/cálculos.

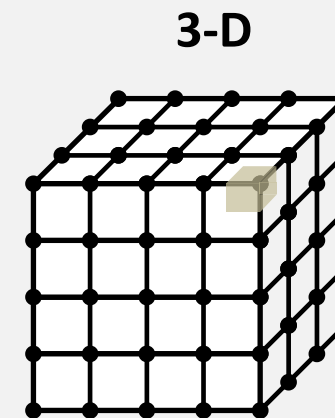
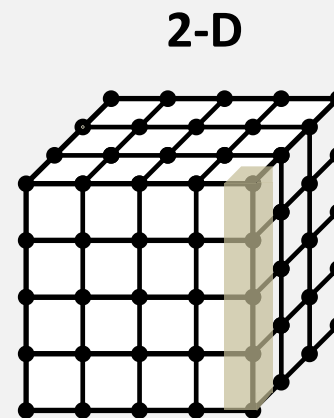
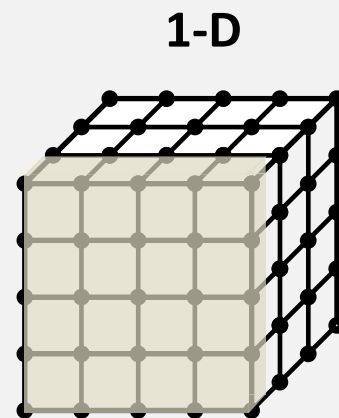
### Tipos

- Generales: Dominio, Funcional y Recursiva.
- Específicos: Explorativa, Especulativa, etc.
- Mixtos: Combinación de los anteriores.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Descomposición del Domino

- ¿Cuándo? Cuando sea posible aplicar el mismo conjunto de operación sobre los datos de cada subdominio.
- ¿Cómo? Dividiendo los datos en pequeñas subconjuntos homogéneos y estudiar la computación aplicable a cada subconjunto. La tarea es la encargada de gestionar la computación sobre sus datos.
- ¿Dónde? Sobre los datos de entrada, de salida (resultados), intermedios, basada en bloques, etc.
- ¿Consejos? Fijarse en los datos de mayor dimensión, frecuencia de acceso y/o reflejen la evolución en la resolución del problema. Obtener subconjuntos disjuntos y de igual tamaño.

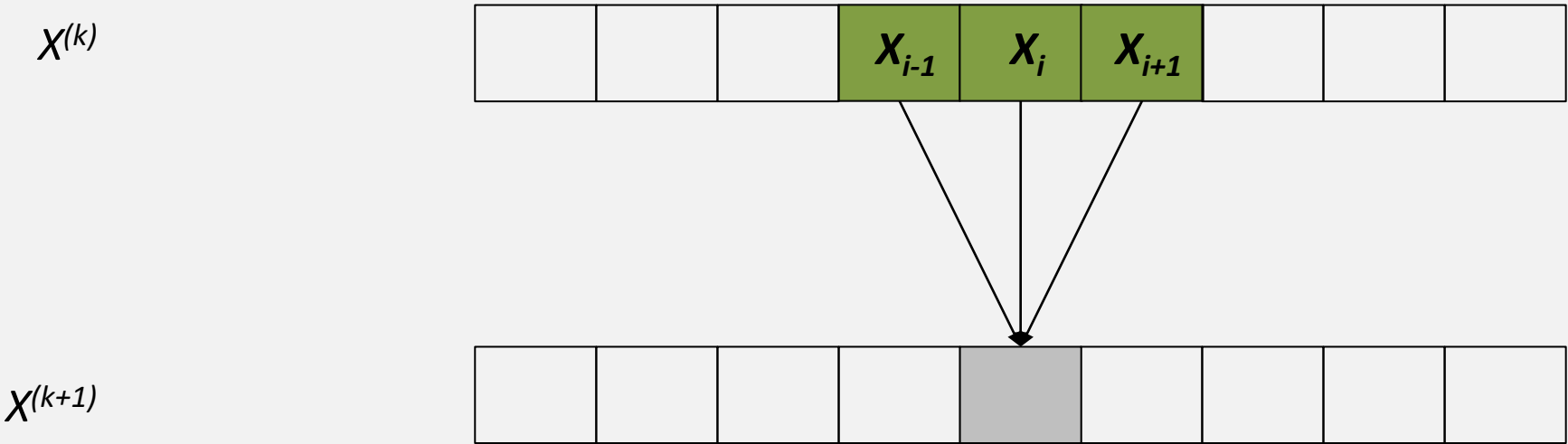


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

Ejemplo 1. Diseñar un algoritmo paralelo iterativo que calcule los vectores  $X^{(0)}, X^{(1)}, \dots, X^{(k)}, X^{(k+1)}$  donde el vector  $X^{(0)}$  es conocido y el resto:

$$X_i^{k+1} = \frac{X_{i-1}^k + X_i^k + X_{i+1}^k}{2}, i = 0, \dots, n - 1$$

$$X_{-1}^k = X_{n-1}^k \qquad X_n^k = X_0^k$$



Centrada en los Datos de Salida

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

Ejemplo 1. Diseñar un algoritmo paralelo iterativo que calcule los vectores  $X^{(0)}, X^{(1)}, \dots, X^{(k)}, X^{(k+1)}$  donde el vector  $X^{(0)}$  es conocido y el resto:

$$X_i^{k+1} = \frac{X_{i-1}^k + X_i^k + X_{i+1}^k}{2}, i = 0, \dots, n-1$$

$$X_{-1}^k = X_{n-1}^k \quad X_n^k = X_0^k$$



Dependencias de flujo (tipo 1) y anti-dependencia (tipo 2).



## FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

Ejemplo 2. Producto escalar de vectores, suponiendo que la dimensión de los vectores  $n$  es divisible entre el número de tareas  $t$ .

$$\sum_{j=1}^n x_j \cdot y_j$$

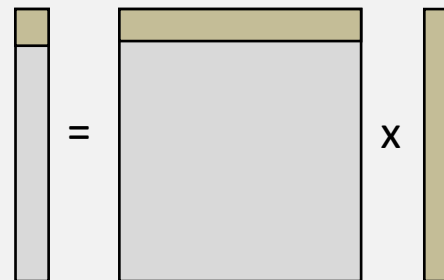
Centrada en los datos de entrada. Cada tarea  $j$  calcula

$$\sum_{j=i\frac{n}{t}}^{(i+1)\frac{n}{t}-1} x_j \cdot y_j$$

Reducción final de los resultados parciales. Problema conocido y visto para MC en las transparencias [19 a 21](#).

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

Ejemplo 3. Resolver  $x = A * b$ , siendo  $x \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{n \times m}$  y  $p = n$ .



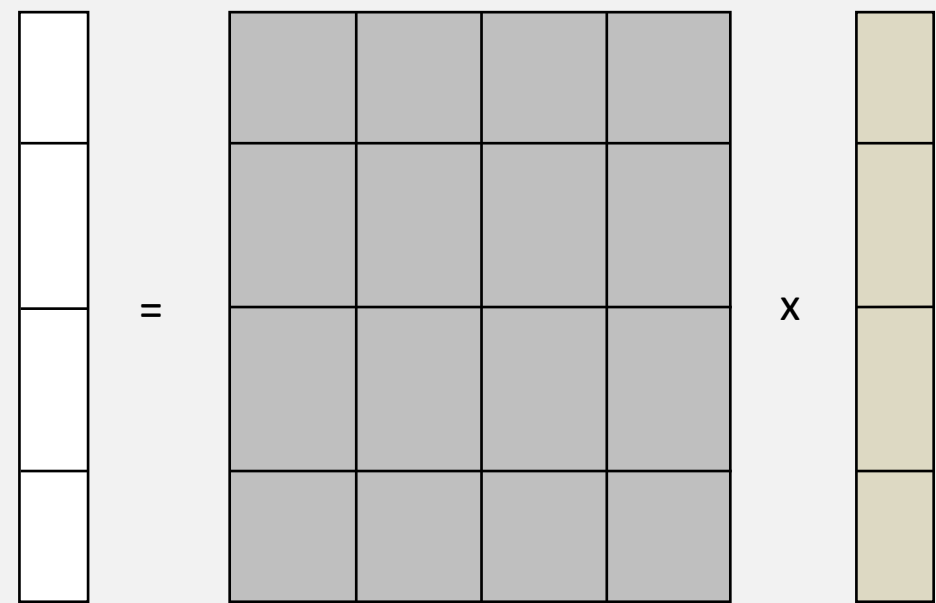
## Algoritmo Secuencial

```
for (i=0; i<n; i++)  
{  
    x[i]=0;  
    for (j=0; j<m; j++)  
        x[i] = x[i] + A[i,j]*b[j];  
}
```

Grafo de Dependencias: similar al ejemplo de la transparencia [17](#) sin reducción final. También al de la [18](#).

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

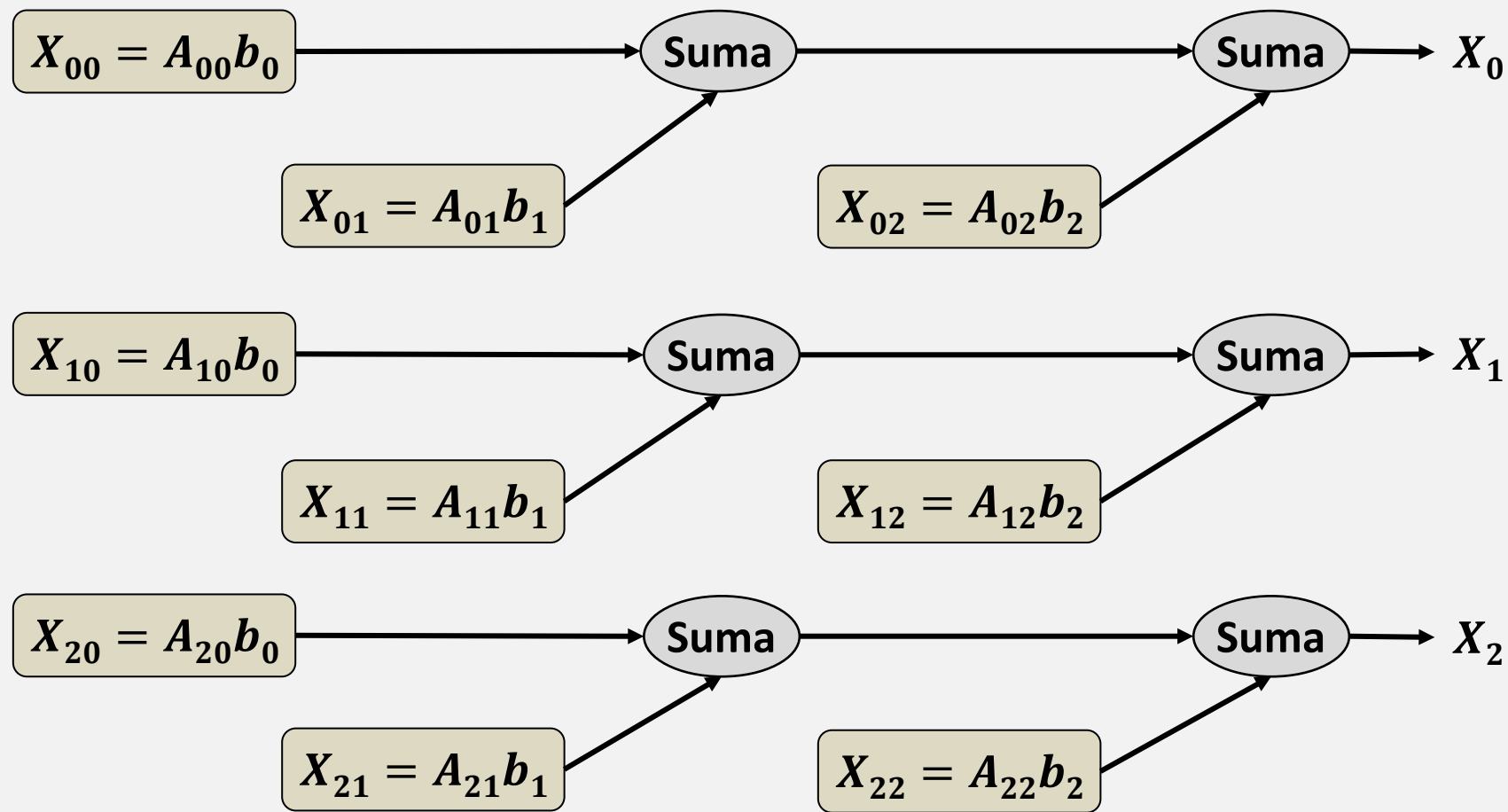
Ejemplo 3. Resolver  $x = A * b$ , siendo  $x \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{n \times m}$  y  $p = n$ . Enfoque por bloques centrado en los datos de salida.



**Tarea 0** →  $X_0 = A_{00} * b_0 + A_{01} * b_1 + A_{02} * b_2 + A_{03} * b_3$   
**Tarea 1** →  $X_1 = A_{10} * b_0 + A_{11} * b_1 + A_{12} * b_2 + A_{13} * b_3$   
**Tarea 2** →  $X_2 = A_{20} * b_0 + A_{21} * b_1 + A_{22} * b_2 + A_{23} * b_3$   
**Tarea 3** →  $X_3 = A_{30} * b_0 + A_{31} * b_1 + A_{32} * b_2 + A_{33} * b_3$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

Ejemplo 3. Resolver  $x = A * b$ , siendo  $x \in \mathbb{R}^n$ ,  $b \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{n \times m}$  y  $p = n$ . Enfoque por bloques centrado en resultados intermedios (*pipeline*).



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Descomposición Funcional

- ¿Cuándo? Cuando la resolución del problema se puede descomponer en fases y en cada fase se ejecuta un algoritmo distinto.
- ¿Cómo? Identificando las fases: fase=tarea. Dividir, con posterioridad, los datos necesarios para cada tarea.
- ¿Consejos? Si los cjtos. de datos son disjuntos (solapamiento / dependencias  $\Rightarrow \oslash$ ) fin. En caso contrario, usar otro tipo de descomposición.
- Ventajas. Útil en problemas de gran complejidad. Reduce la complejidad al paralelizar estructuras grandes con accesos múltiples.
- Inconvenientes. No  $\Rightarrow$  descomposiciones de grano-fino. Menos flexible, menos escalable. La división no disjunta  $\Rightarrow$  comunicaciones complejas.
- Ejemplo. Sea una serie de  $m$  números primos y  $n$  una lista de número enteros. Buscar los números de la lista múltiplos de todos los primos.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Descomposición Recursiva

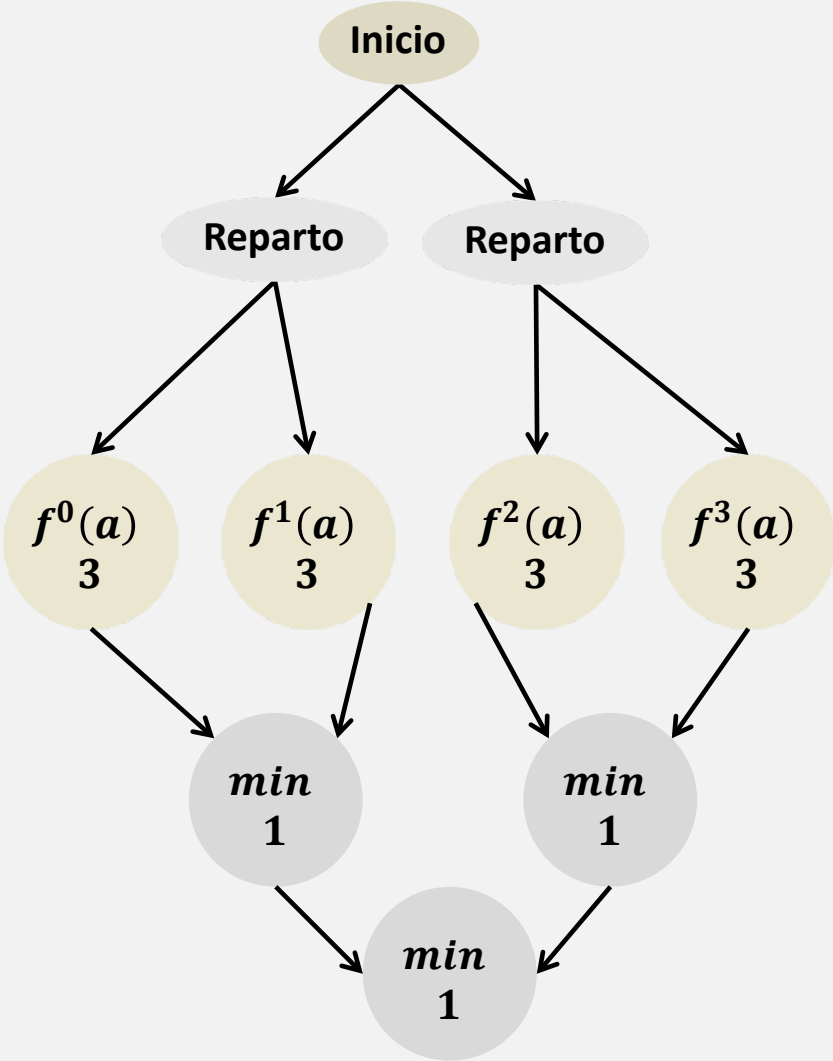
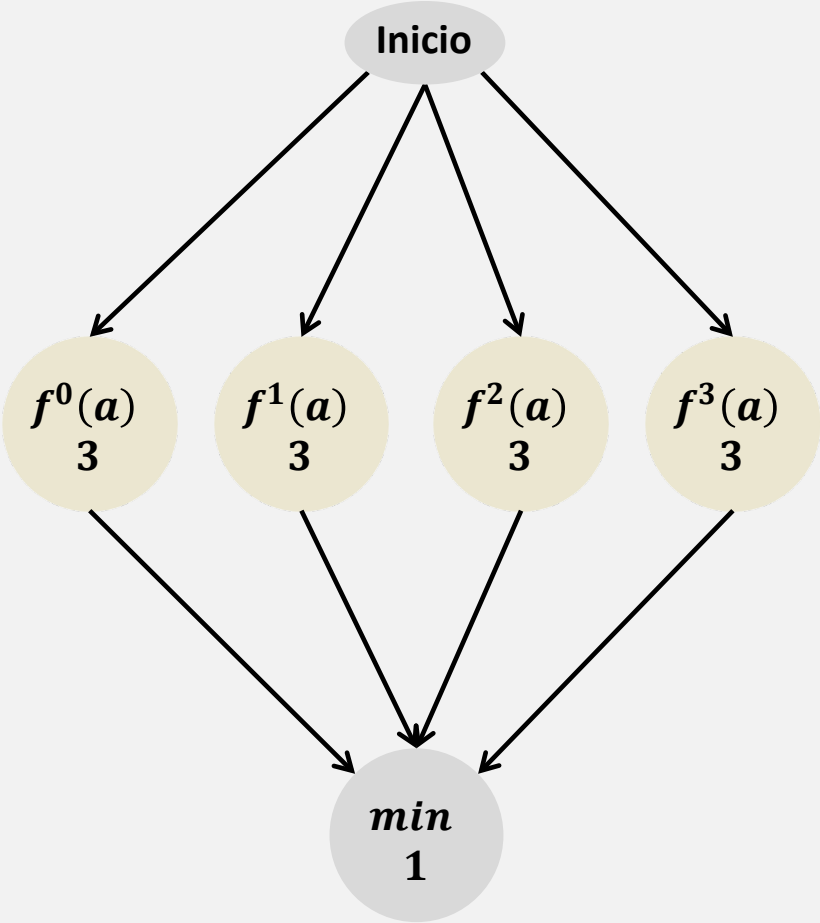
- Basada en la técnica de Divide y Vencerás. La generación recursiva de los subproblemas crea la concurrencia.
- ¿Cómo? Dividiendo recursivamente el problema hasta alcanzar el criterio de parada. Los resultados parciales se combinan para obtener el resultado final (salvo en recursión final).
- ¿Dónde? Trabajadores replicados con bolsa de tareas (bolsa tareas = grupo de tareas que cargan subproblemas/datos de estructuras compartidas). Algoritmos recursivos. Todo lo aprendido en la asignatura Algoritmia y otras.
- El ejemplo de la transparencia [30](#) se puede resolver de esta forma.
- Calcular el mínimo de una secuencia de reales de longitud  $m$ , tal que:
 
$$\text{MIN}(a_1) = a_1$$

$$\text{MIN}(a_1, a_2) = \text{mínimo}(a_1, a_2)$$

$$\text{MIN}(a_1, a_2 \dots a_m; m > 2) = \text{mínimo}(\text{MIN}(a_1, \dots, a_{m/2}), \text{MIN}(a_{m/2+1}, \dots, a_m))$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Descomposición Recursiva





# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Descomposición Explorativa

- ¿Cuándo? Búsqueda de soluciones en un espacio de estados.
- ¿Cómo? Descomposición dinámica del espacio de búsqueda en partes menores más exploración concurrente de cada parte por tareas diferentes. Establecer estructuras tipo árbol con las posibles soluciones.
- ¿Estrategia? Generar niveles de estados desde el estado inicial. Cuando se tengan suficientes nodos cada tarea explora un conjunto de nodos diferente.
- Casos Particulares
  - Búsqueda exhaustiva: finaliza cuando no existan nodos.
  - Búsqueda primera solución: cuando se encuentra la solución se informa al resto y fin.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## 2. Comunicaciones

- Las tareas creadas se pueden ejecutar concurrentemente, pero no de forma independiente.

### Etapas en el estudio de las comunicaciones

- Definir, según la tecnología, la estructura de los enlaces (canales) y las tareas productor y consumidor.
- Definir los tipos de mensajes que circulan por cada tipo de canal.

### Particionado y complejidad de las comunicaciones

- Funcional: particionado complejo y comunicaciones simples.
- Domino: lo contrario.

### Incidir en

- Existen relaciones de sincronización y comunicación entre tareas que pueden no aparecer en los grafos.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Tipos de comunicaciones

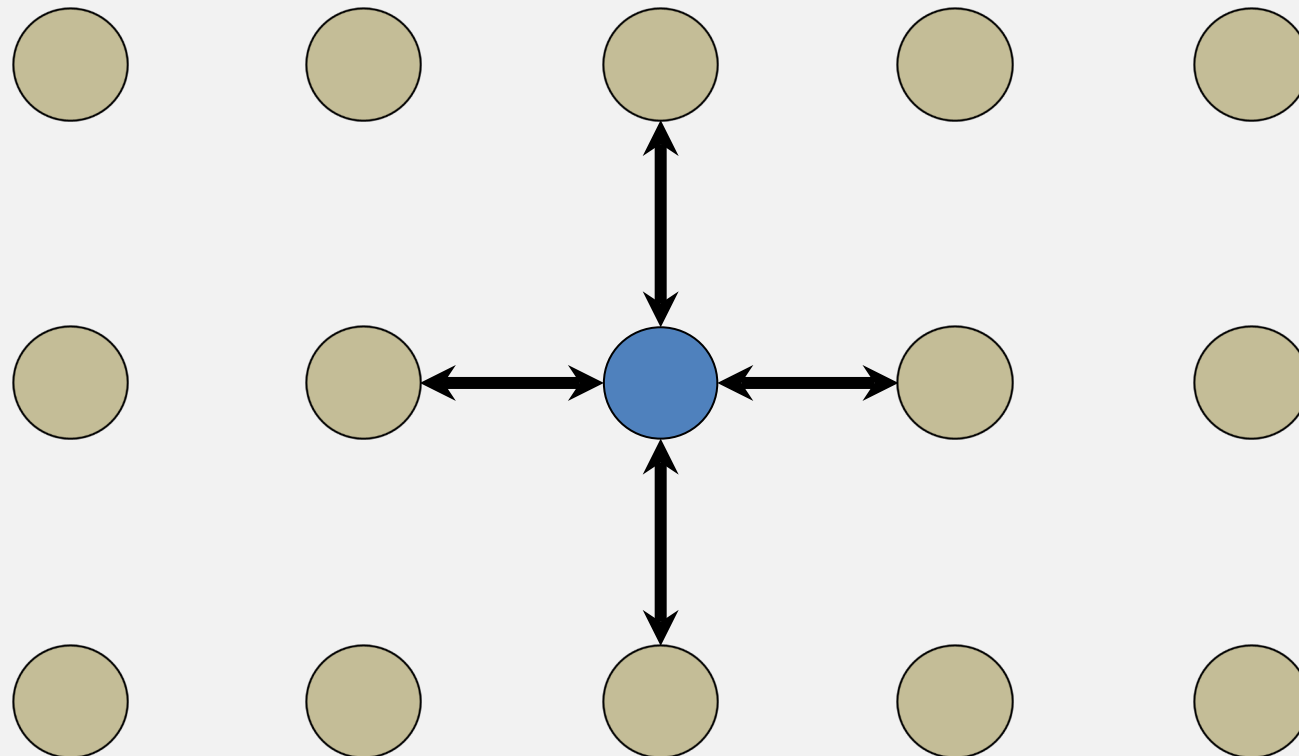
- **Locales.** Cada tarea solo necesita interactuar con un conjunto pequeño de tareas “*vecinas*”. Sencillez en su definición:
  - Definir las aristas en la estructura de comunicación.
  - Definir las operaciones de envío/recepción (paso de mensajes) o de sincronización (memoria compartida)
- **Globales.** Múltiples tareas deben aportar datos para realizar un cálculo en paralelo. Origen en análisis puramente localistas.
- **Estáticas.** Para cada tarea son conocidas las tareas con las que se puede comunicar y los tiempos en que se comunican.
- **Dinámicas.** Lo contrario.
- **Regulares.** Cuando la estructura espacial de las interacciones permite una eficiente implementación.
- **Asíncrona/Síncrona – Unilateral/Bilateral – Lectura/Lectura-Escritura**

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i, j]^{t+1} = \frac{4X[i, j]^t + X[i - 1, j]^t + X[i + 1, j]^t + X[i, j - 1]^t + X[i, j + 1]^t}{8}$$

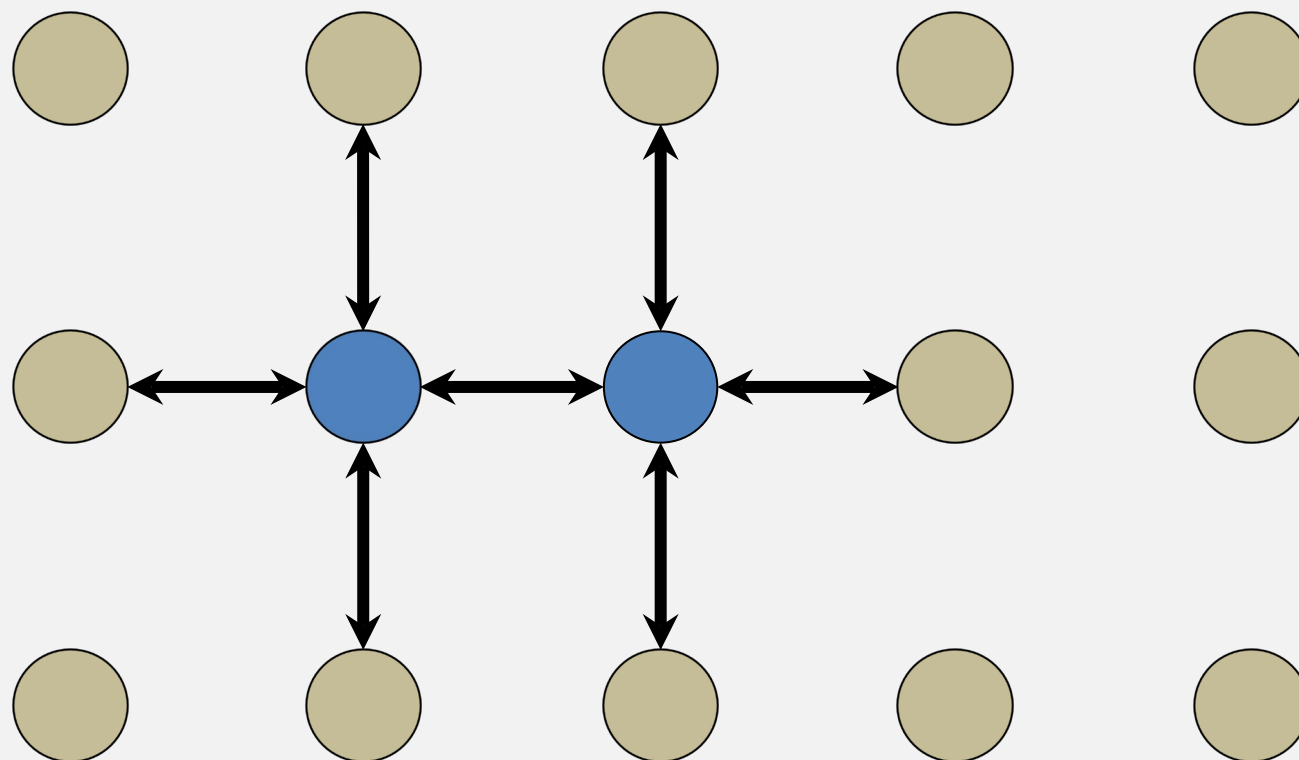


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i, j]^{t+1} = \frac{4X[i, j]^t + X[i - 1, j]^t + X[i + 1, j]^t + X[i, j - 1]^t + X[i, j + 1]^t}{8}$$

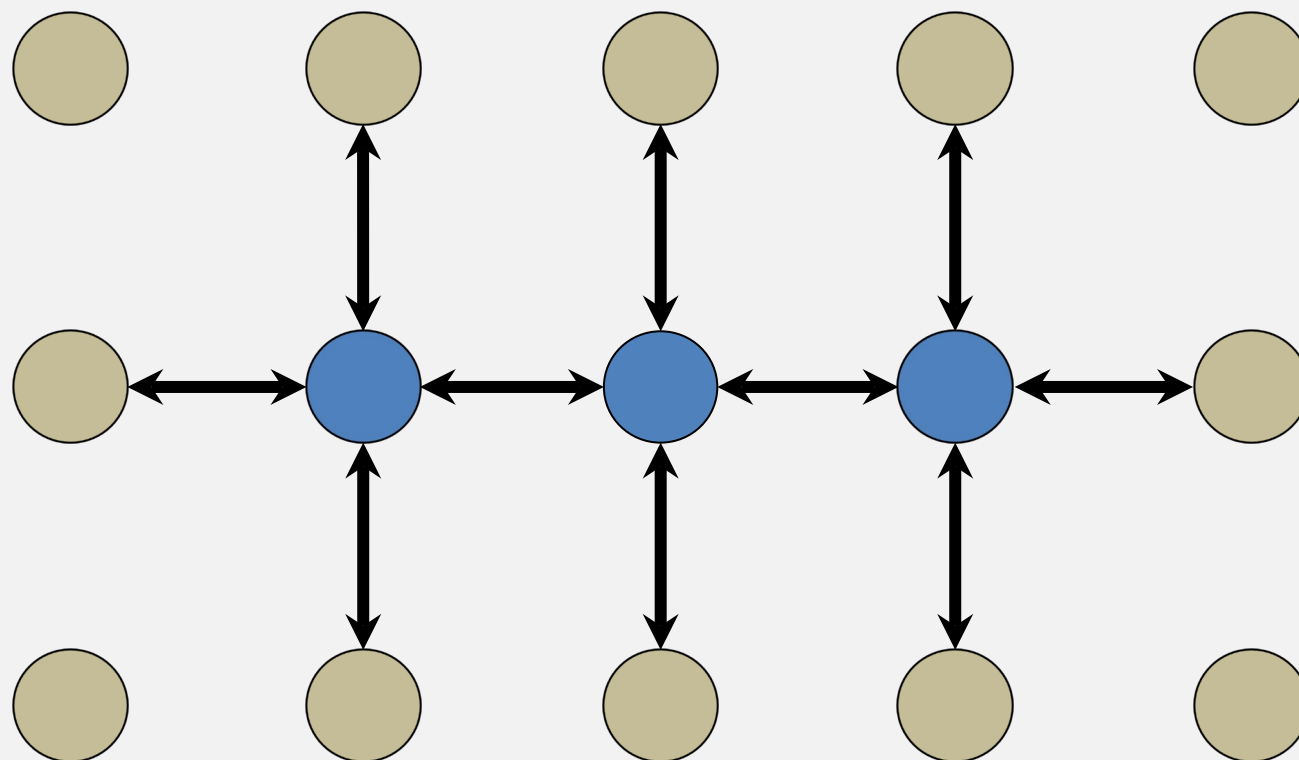


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i, j]^{t+1} = \frac{4X[i, j]^t + X[i - 1, j]^t + X[i + 1, j]^t + X[i, j - 1]^t + X[i, j + 1]^t}{8}$$

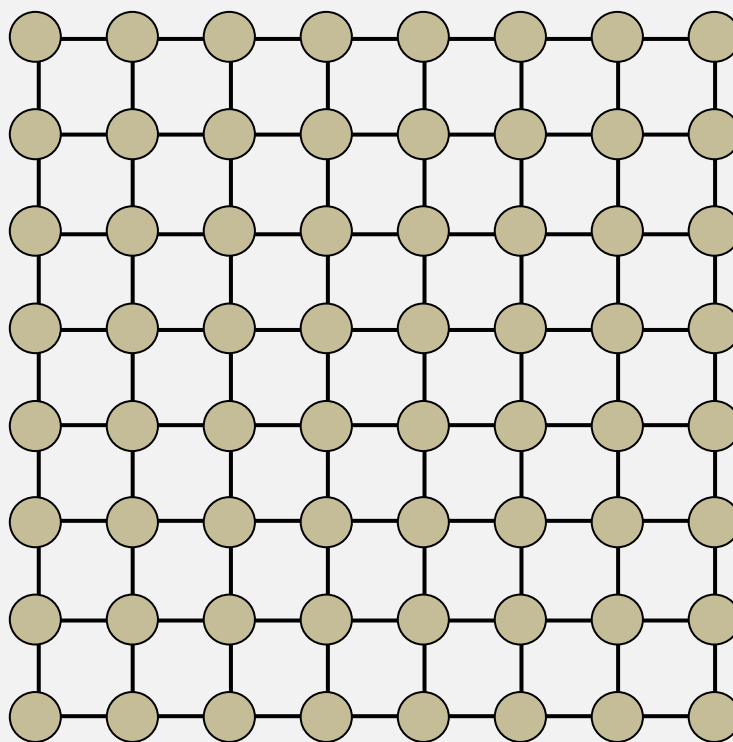


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i, j]^{t+1} = \frac{4X[i, j]^t + X[i - 1, j]^t + X[i + 1, j]^t + X[i, j - 1]^t + X[i, j + 1]^t}{8}$$

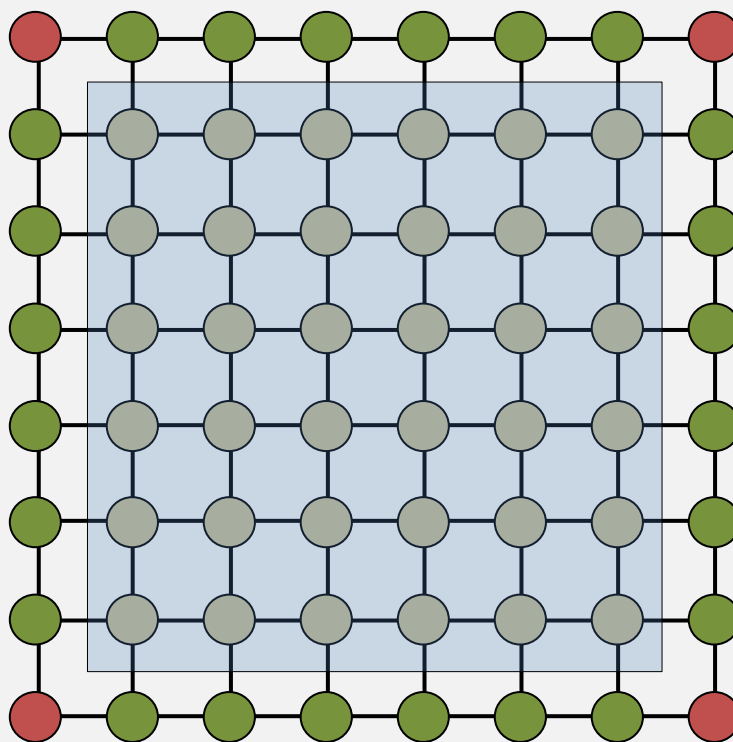


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i,j]^{t+1} = \frac{4X[i,j]^t + X[i-1,j]^t + X[i+1,j]^t + X[i,j-1]^t + X[i,j+1]^t}{8}$$



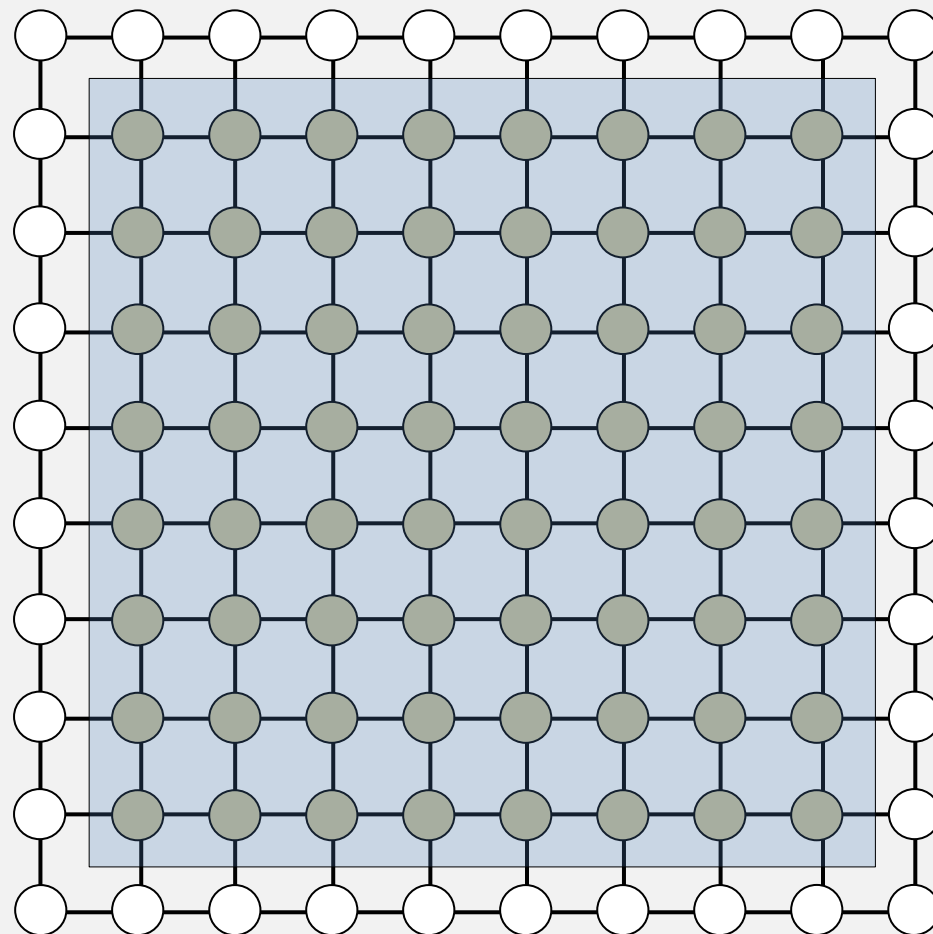


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración ...

$$X[i, j]^{t+1} = \dots$$



Aplicar técnicas para simplificar la implementación: p. ej. *Padding* en MC

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i, j]^{t+1} = \frac{4X[i, j]^t + X[i - 1, j]^t + X[i + 1, j]^t + X[i, j - 1]^t + X[i, j + 1]^t}{8}$$

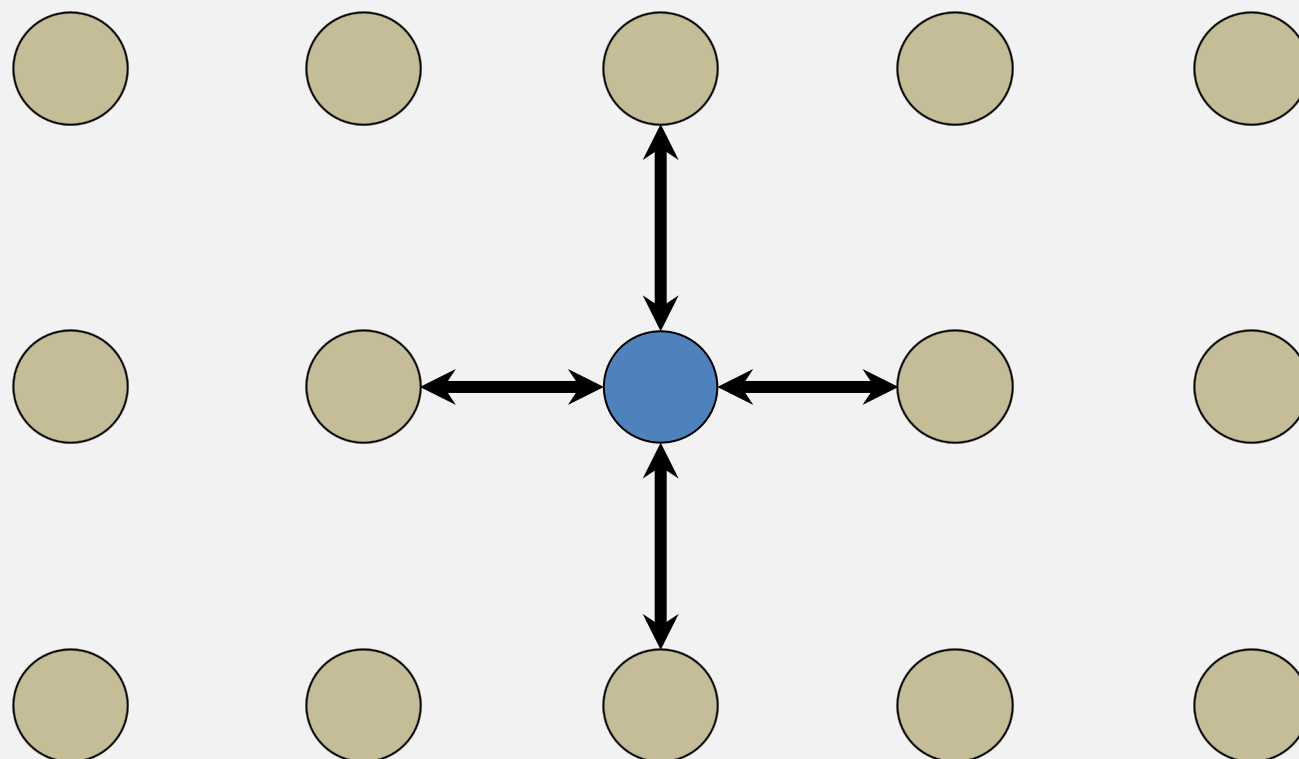
- En cada tarea  $X[i, j]$  hacer
  - para  $t=1$  hasta  $k$ 
    - enviar valor a cada vecino afectado
    - recibir datos de los vecinos afectados
    - actualizar valor
  - fin para
- Fácil de paralelizar
- Todas las tareas concurrentes y balanceo de la carga
- Tiempo de ejecución:  $T(n, p) = k(4(t_s + t_w) + 4(t_s + t_w) + 6t_c)$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i,j]^{t+1} = \frac{4X[i,j]^t + X[i-1,j]^{t+1} + X[i+1,j]^t + X[i,j-1]^{t+1} + X[i,j+1]^t}{8}$$

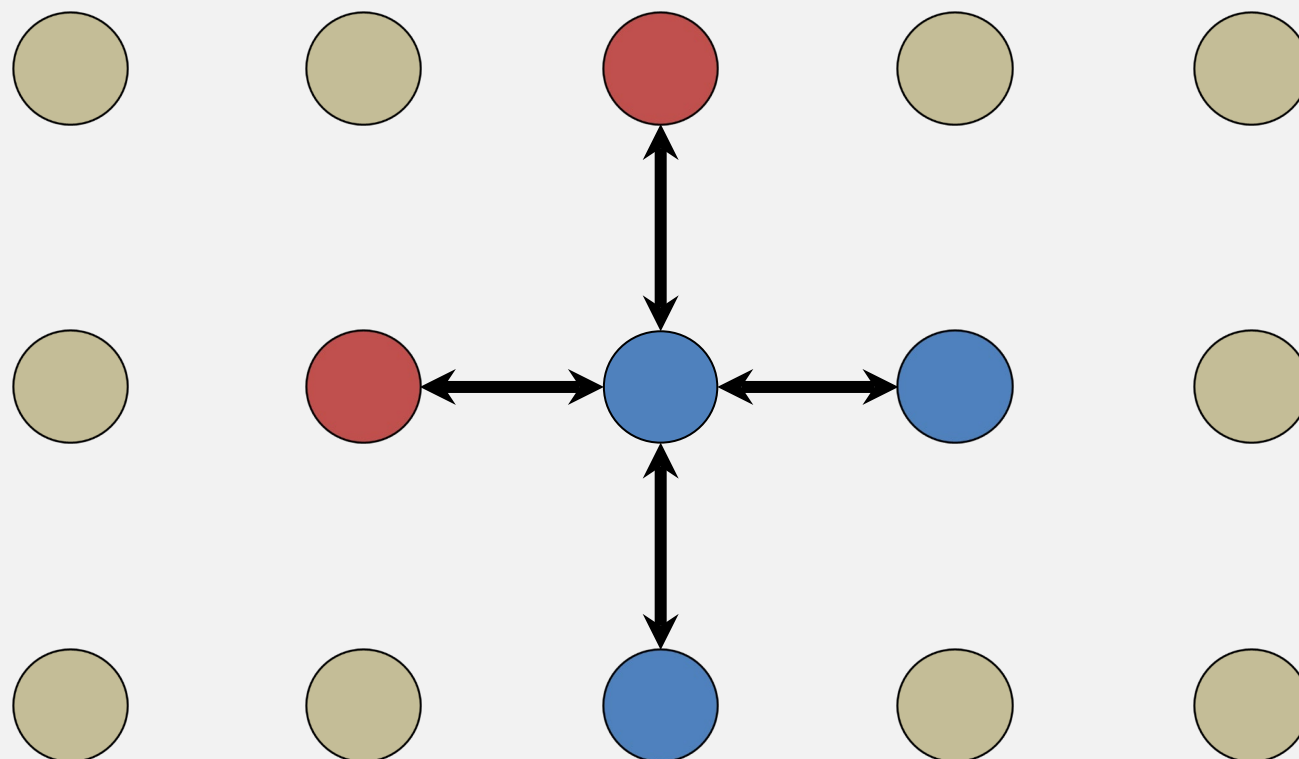


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz modifican su valor según la expresión adjunta. Se asume descomposición del dominio.

$$X[i,j]^{t+1} = \frac{4X[i,j]^t + X[i-1,j]^{t+1} + X[i+1,j]^t + X[i,j-1]^{t+1} + X[i,j+1]^t}{8}$$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz... **Primera** aproximación.

(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(1)	(1)	(1)	(0)
(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(1)	(1)	(1)	(1)
(1)	(1)	(1)	(0)
(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)

(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)
(1)	(1)	(1)	(0)
(1)	(1)	(0)	(0)

(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)
(1)	(1)	(1)	(0)

(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)
(1)	(1)	(1)	(1)

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz... **Segunda** aproximación.

(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(2)	(1)	(1)	(0)
(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)
(0)	(0)	(0)	(0)

(2)	(2)	(1)	(1)
(2)	(1)	(1)	(0)
(1)	(1)	(0)	(0)
(1)	(0)	(0)	(0)

(3)	(2)	(2)	(1)
(2)	(2)	(1)	(1)
(2)	(1)	(1)	(0)
(1)	(1)	(0)	(0)

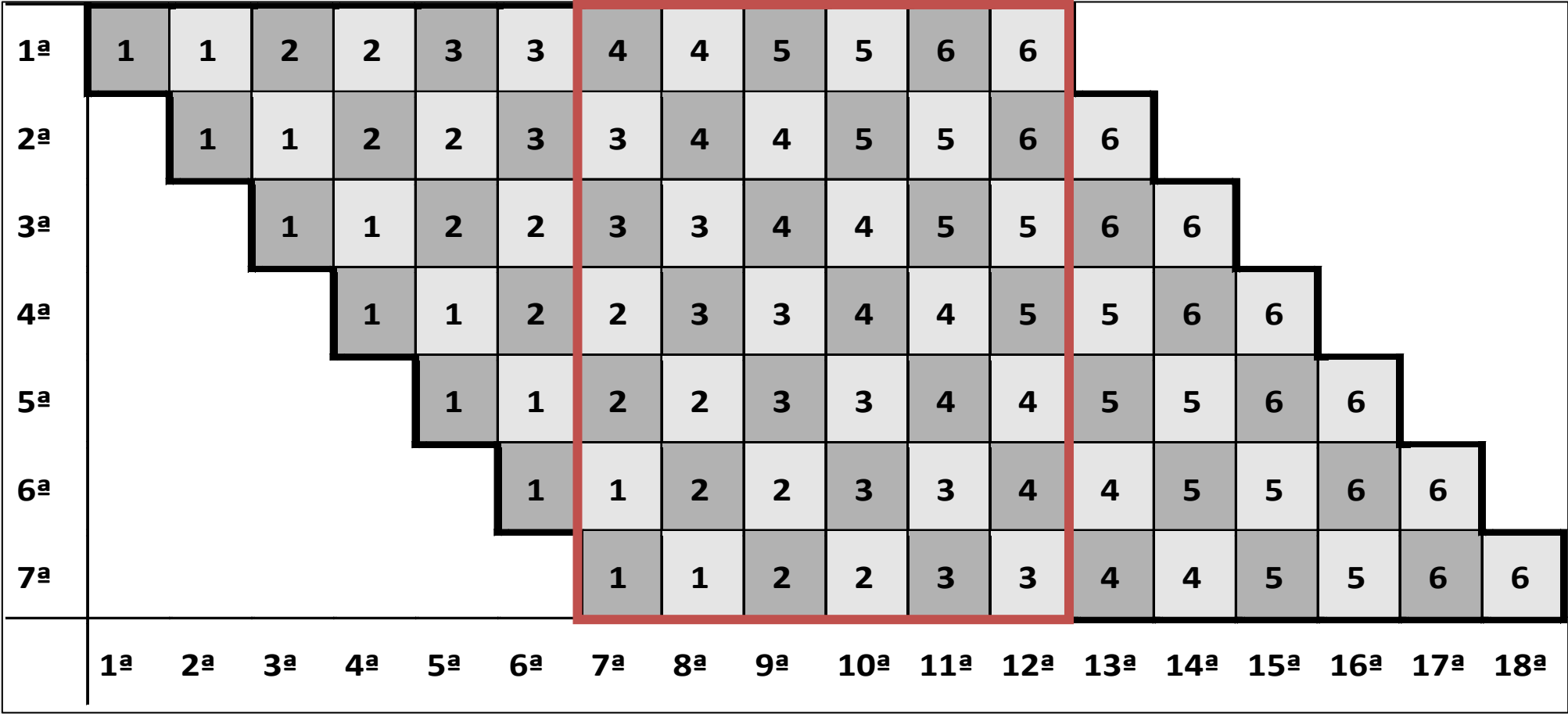
(3)	(3)	(2)	(2)
(3)	(2)	(2)	(1)
(2)	(2)	(1)	(1)
(2)	(1)	(1)	(0)

(4)	(3)	(3)	(2)
(3)	(3)	(2)	(2)
(3)	(2)	(2)	(1)
(2)	(2)	(1)	(1)

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz... **Segunda** aproximación. Completo para  $n = m = 4$  y  $k = 6$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

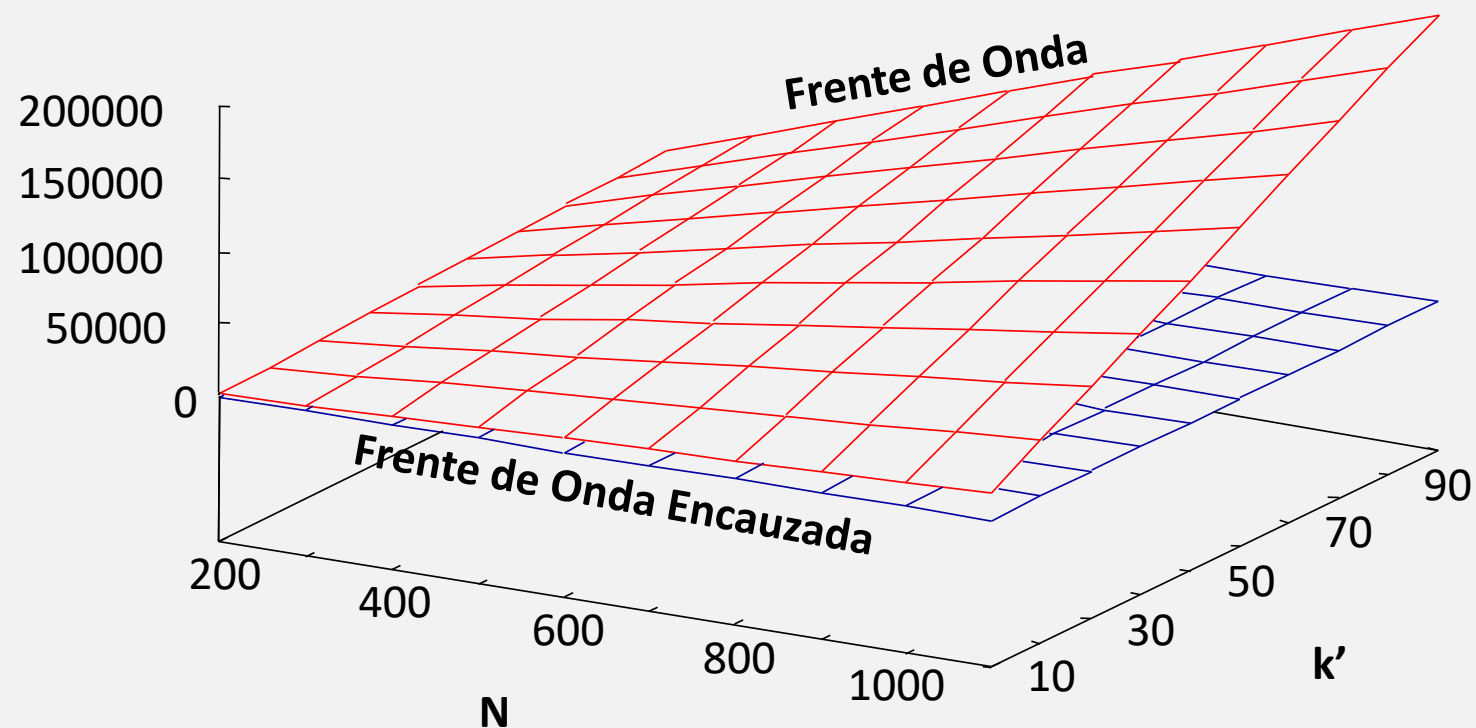
## Ejemplo Comunicaciones Locales

- Gauss-Seidel por frente de onda

$$T \cong k'(n + m)(4(t_s + t_w) + 4(t_s + t_w) + 6t_c)$$

- Gauss-Seidel por frente de onda encauzado

$$T \cong (n + m)(4(t_s + t_w) + 4(t_s + t_w) + 6t_c) + k'(4(t_s + t_w) + 4(t_s + t_w) + 6t_c)$$

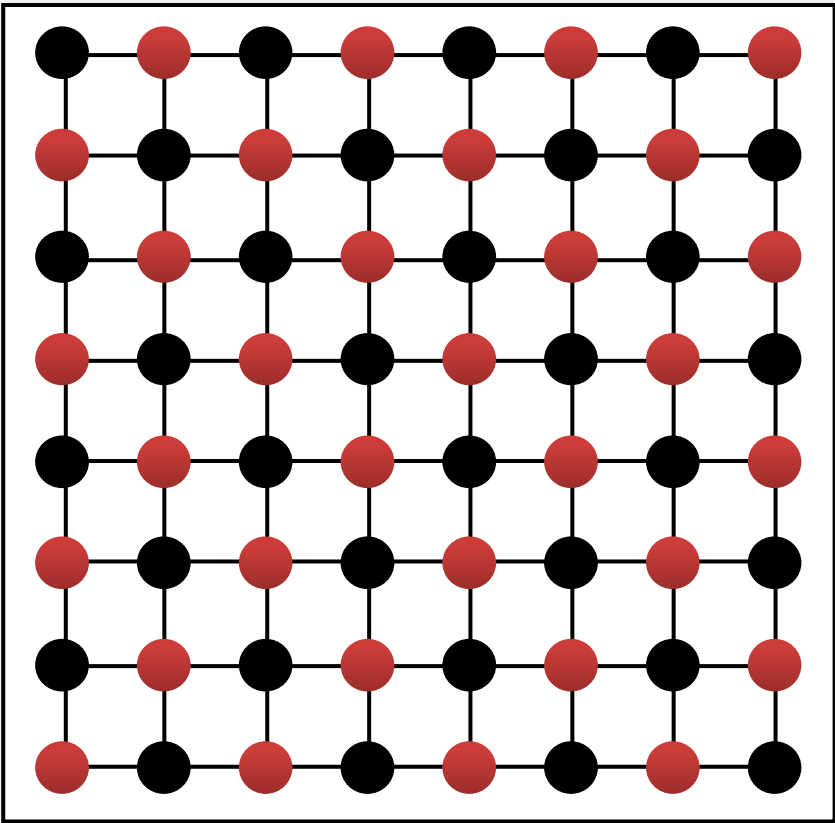




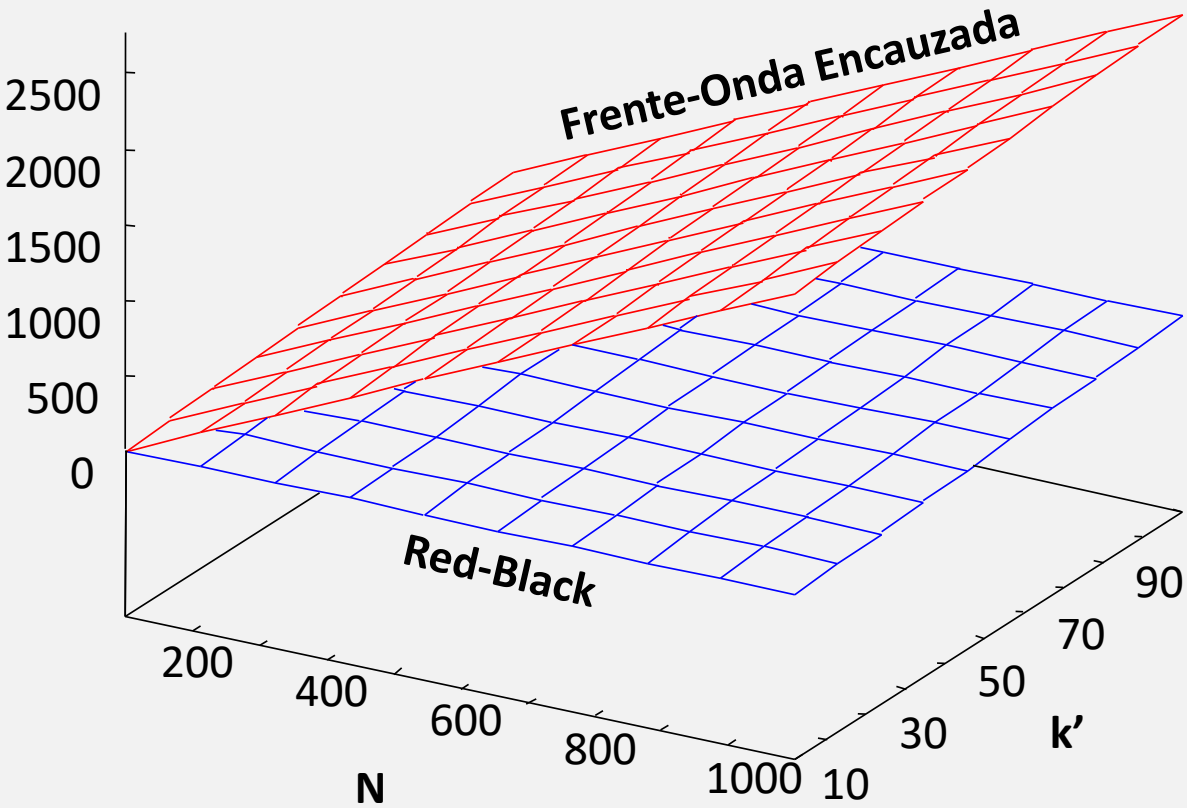
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Comunicaciones Locales

- En cada iteración todos los elementos de una matriz... **Tercera** aproximación: *red-black*. Crear dos grupos de  $n/2$  tareas sin dependencias intra-grupo. Actualizar alternadamente cada grupo.



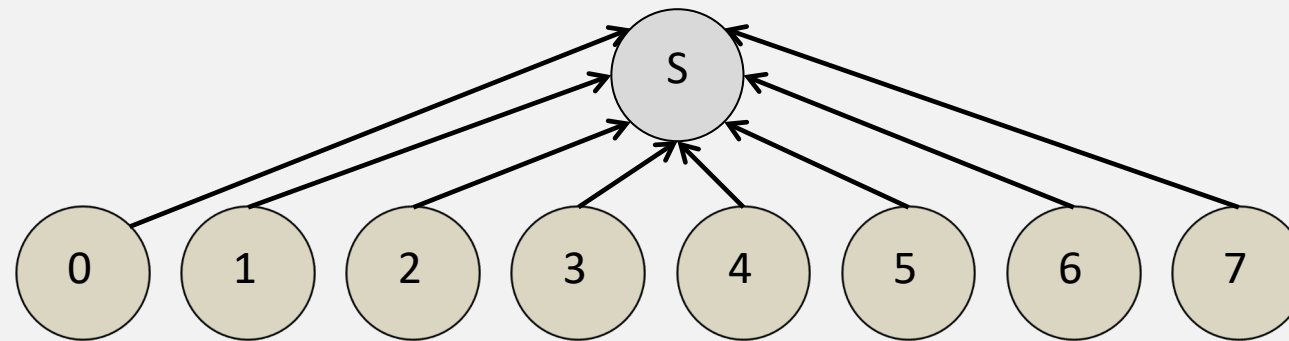
$$T \cong 2k'(4(t_s + t_w) + 4(t_s + t_w) + 6t_c)$$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

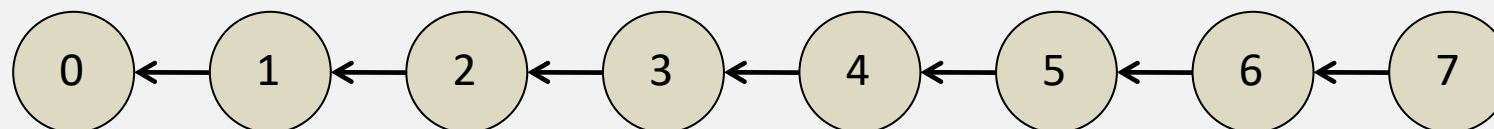
## Ejemplo Globales

- Realizar de forma paralela la operación de reducción  $S = \sum_{i=0}^{N-1} X_i$ 
  - Primera aproximación:  $\in \theta(n)$



Algoritmo paralelo centralizado con elevada componente secuencial: muy ineficiente. Conclusión: No es suficiente con identificar pares productor / consumidor individuales.

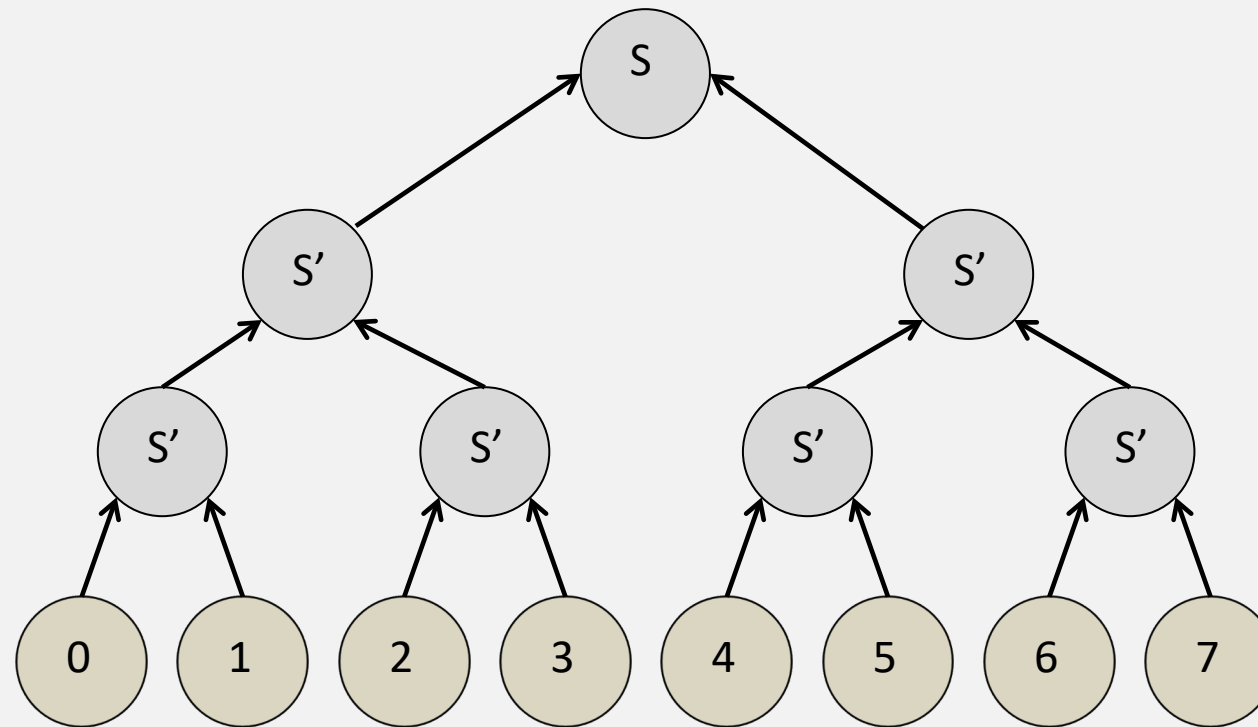
- Segunda aproximación: distribuir la computación balanceando la carga. Problema: sigue siendo  $\in \theta(n)$



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Ejemplo Globales

- Realizar de forma paralela la operación de reducción  $S = \sum_{i=0}^{N-1} X_i$ 
  - Tercera aproximación: explotar la concurrencia con particionado recursivo, esto es, *“Divide y Vencerás”*.



Es de  $\theta(\log n)$ , pero el problema es que no es balanceado.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## 3. Agrupar

- Las prestaciones se degradan por dependencias, alto porcentaje de tiempo secuencial, comunicaciones, mala distribución de la carga, etc.
- Aumenta la granularidad de las tareas puede mejorar la eficiencia (minimizar el tiempo de ejecución) de los algoritmos paralelos.
- Esta fase, y la siguiente, comparten objetivos. La diferencia está en su esencia. En esta fase el foco es el algoritmo/diseño, que se puede (o debe) modificar para conseguir el fin último. En la siguiente la distribución de las tareas en los procesadores físicos, sin modificar el diseño/programa.
- Al agrupar se reduce el número de tareas con objeto de:
  - Limitar costes de creación y destrucción de tareas.
  - Minimizar los retardos debidos a la interacción entre tareas (acceso a datos locales frente a remotos).

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Qué Maximizar/Minimizar al Agrupar

- Maximizar el tiempo de computación, la concurrencia, asignando tareas independientes a diferentes procesos.
- Minimizar el tiempo de comunicación asignando tareas que se comuniquen mucho al mismo proceso.
- Minimizar el tiempo de ocio evitando las fuentes de inactividad.

## Estrategias

- **Minimizar el volumen de datos transferidos.** Distribución basada en bloques de datos, agrupamiento de tareas no concurrentes (grafos de tareas estáticos), almacenamiento temporal de resultados, etc.
- **Reducir la frecuencia de interacciones.** Minimizar el número de transferencias y aumentar el volumen de datos a transferir.
  - En paso de mensajes significa reducir el número de mensajes y aumentar su tamaño. En memoria compartida reducir el número de fallos de caché.

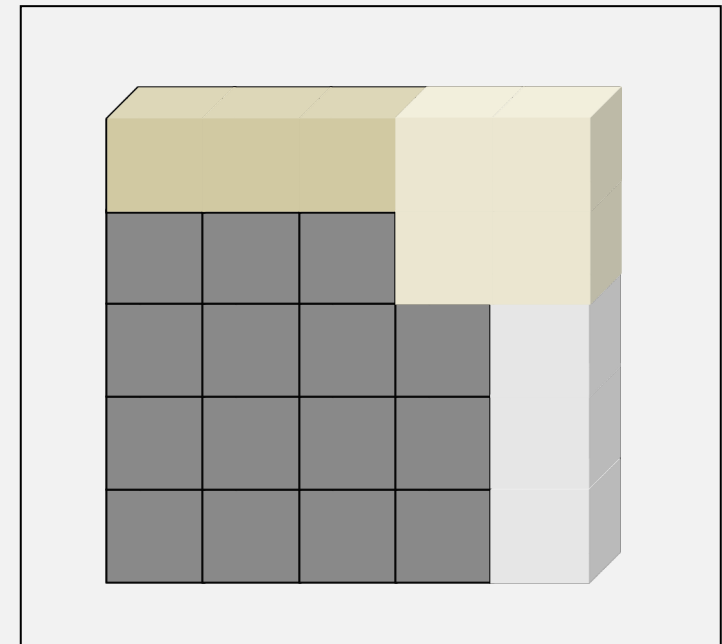
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## El Problema al Agrupar

- ¿Cómo mantener la flexibilidad respecto a la escalabilidad?
- ¿Cómo reducir el número de tareas (aumentar granularidad) reduciendo el coste de las comunicaciones?
- ¿Qué tareas a qué procesos?
- ¿Qué orden de ejecución de las tareas?
- Etc.

## Algunas Estrategias

- Relación Superficie / Volumen.
- Replicación de datos / computación y comunicaciones.
- Tratamiento de tareas no concurrentes.
- El grafo de dependencias es un buen punto de partida.
- El análisis de la eficiencia de los algoritmos es una buena herramienta.
- Etc.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- **Definición**

- El número de comunicaciones que requiere cada tarea es proporcional a la superficie de su dominio. La computación es proporcional al volumen de su dominio.

- **Objetivo**

- Poca superficie y mucho volumen.

- **Consideraciones**

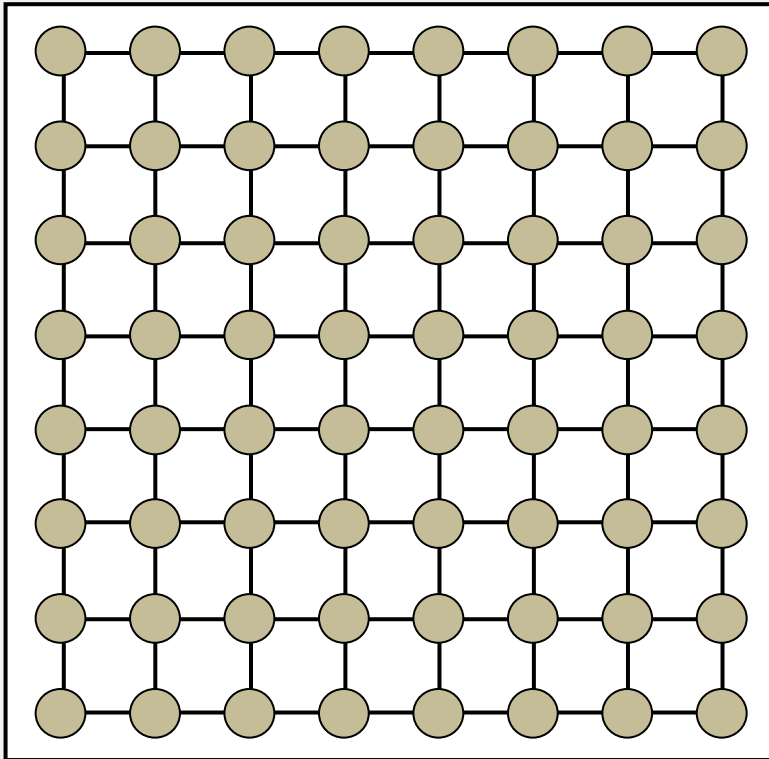
- Presente en descomposición del dominio.
- Aumentar la granularidad sin reducir la dimensionalidad del problema. Como siempre, puede haber excepciones.

En el ejemplo ilustrativo (transparencia [43](#))

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Información movida por iteración en el ejemplo ilustrativo.



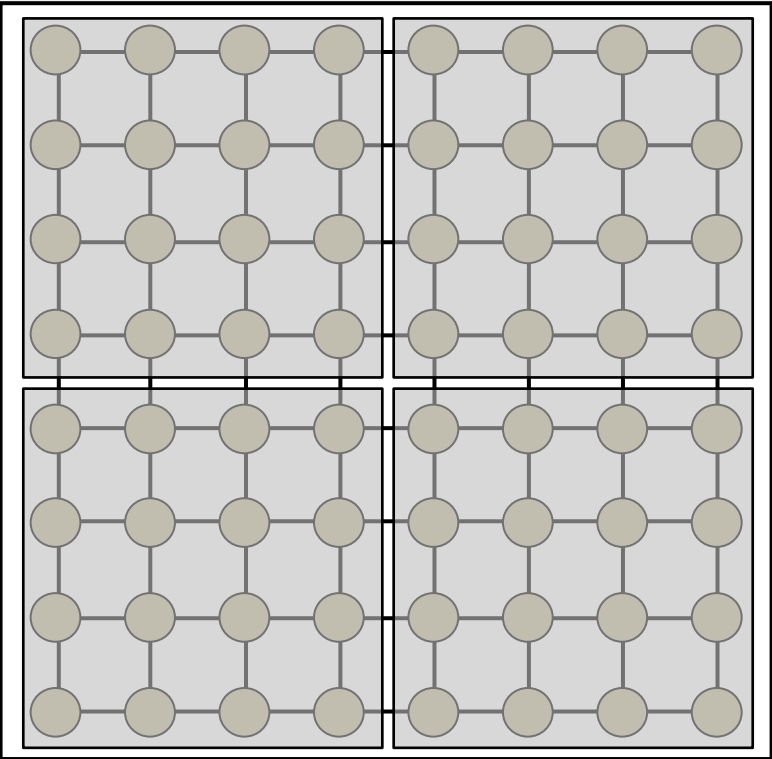
$$64 \times 8(t_s + t_w) = 512t_s + 512t_w$$



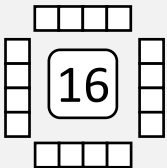
# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Información movida por iteración en el ejemplo ilustrativo.



$$64 \times 8(t_s + t_w) = 512t_s + 512t_w$$

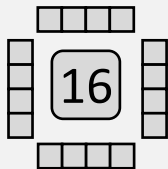
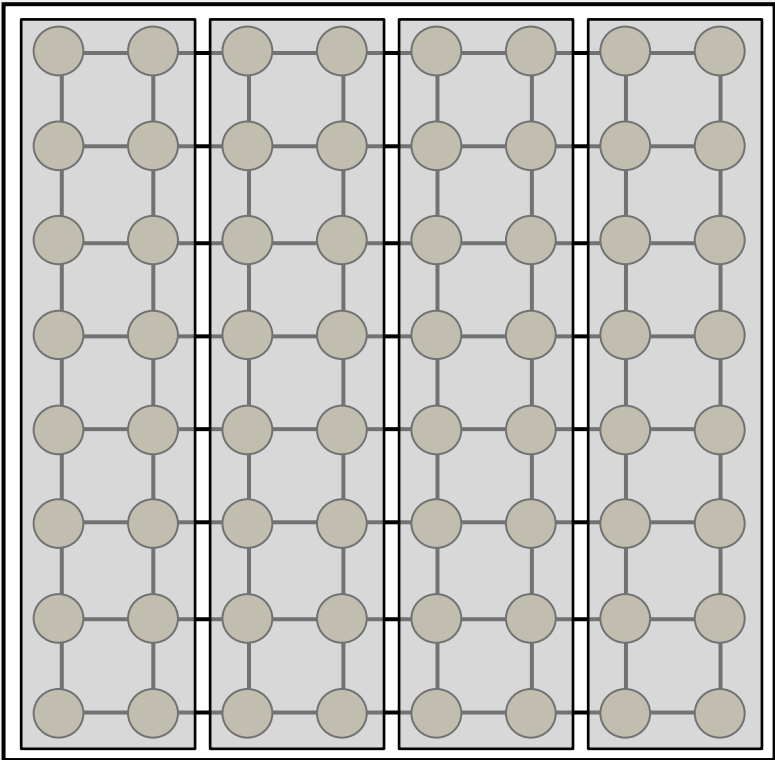


$$4 \times 8(t_s + 4t_w) = 32t_s + 128t_w$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

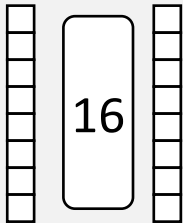
## Relación Superficie / Volumen

- Información movida por iteración en el ejemplo ilustrativo.



$$4 \times 8(t_s + 4t_w) = 32t_s + 128t_w$$

$$64 \times 8(t_s + t_w) = 512t_s + 512t_w$$

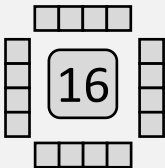
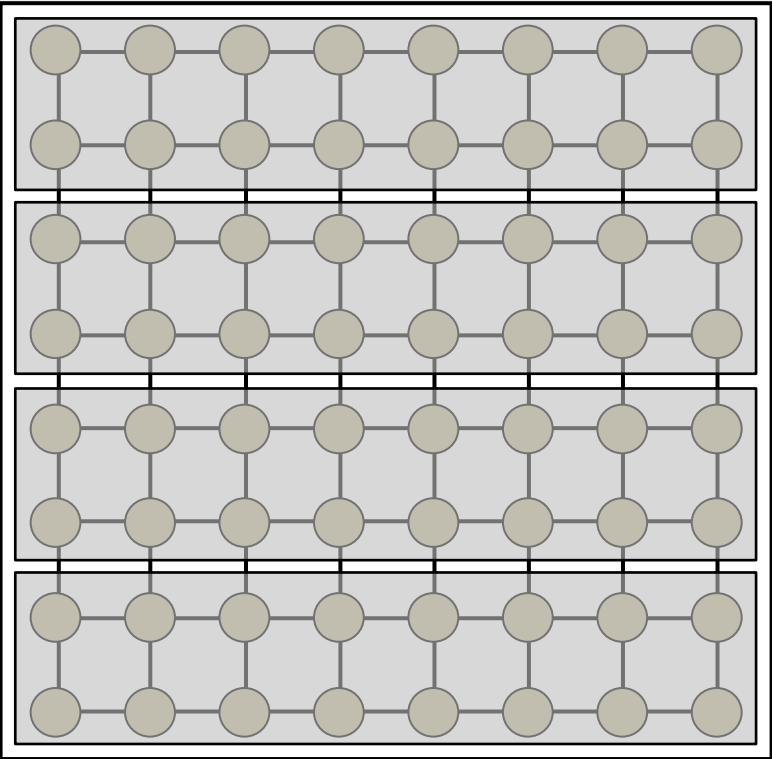


$$4 \times 4(t_s + 8t_w) = 16t_s + 128t_w$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

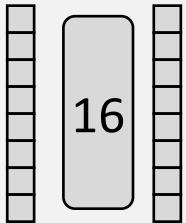
## Relación Superficie / Volumen

- Información movida por iteración en el ejemplo ilustrativo.

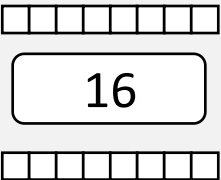


$$4 \times 8(t_s + 4t_w) = 32t_s + 128t_w$$

$$64 \times 8(t_s + t_w) = 512t_s + 512t_w$$



$$4 \times 4(t_s + 8t_w) = 16t_s + 128t_w$$

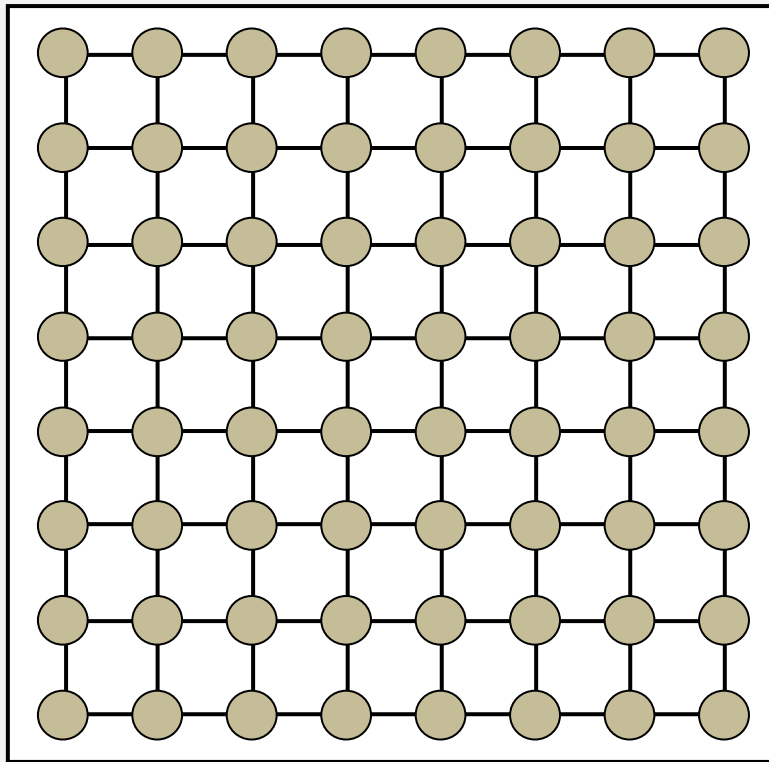


$$4 \times 4(t_s + 8t_w) = 16t_s + 128t_w$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Análisis de la relación en el ejemplo ilustrativo.

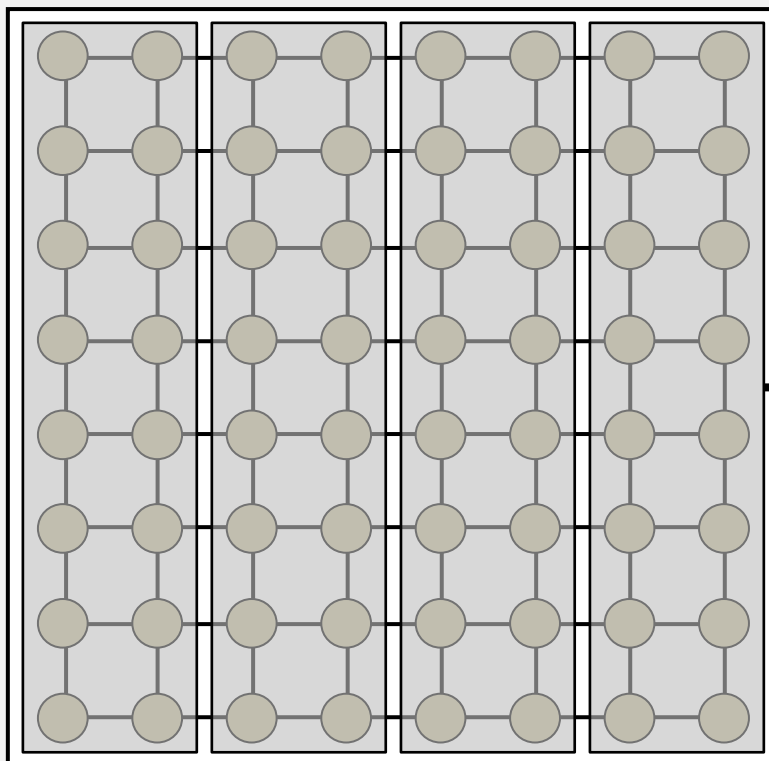


$$\frac{8(t_s + t_w)}{6t_c}$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Análisis de la relación en el ejemplo ilustrativo.



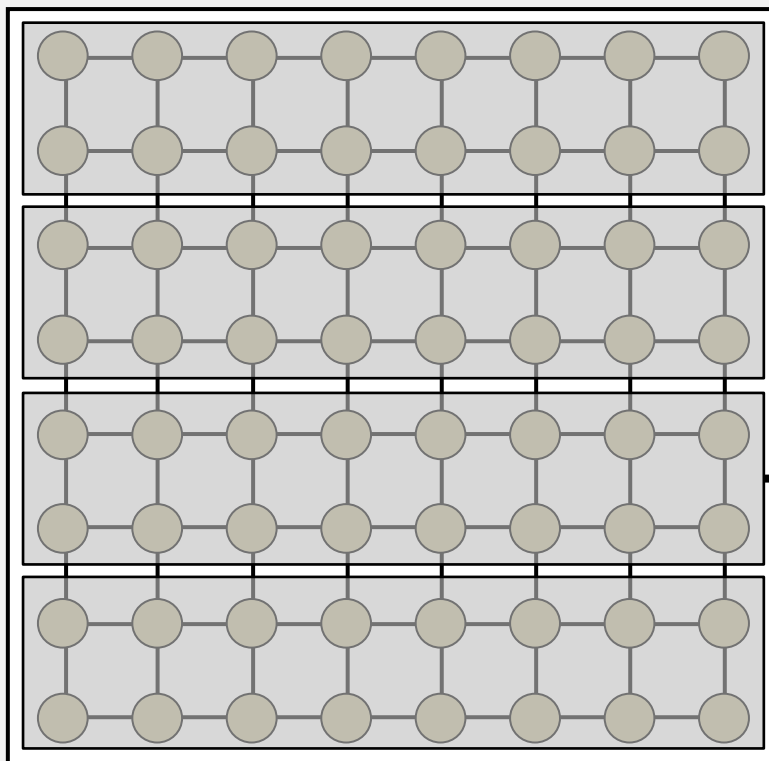
$$\frac{4(t_s + 8t_w)}{16 \times 6t_c} = \frac{4t_s + 32t_w}{96t_c}$$

$$\frac{8(t_s + t_w)}{6t_c}$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Análisis de la relación en el ejemplo ilustrativo.



$$\frac{4(t_s + 8t_w)}{16 \times 6t_c} = \frac{4t_s + 32t_w}{96t_c}$$

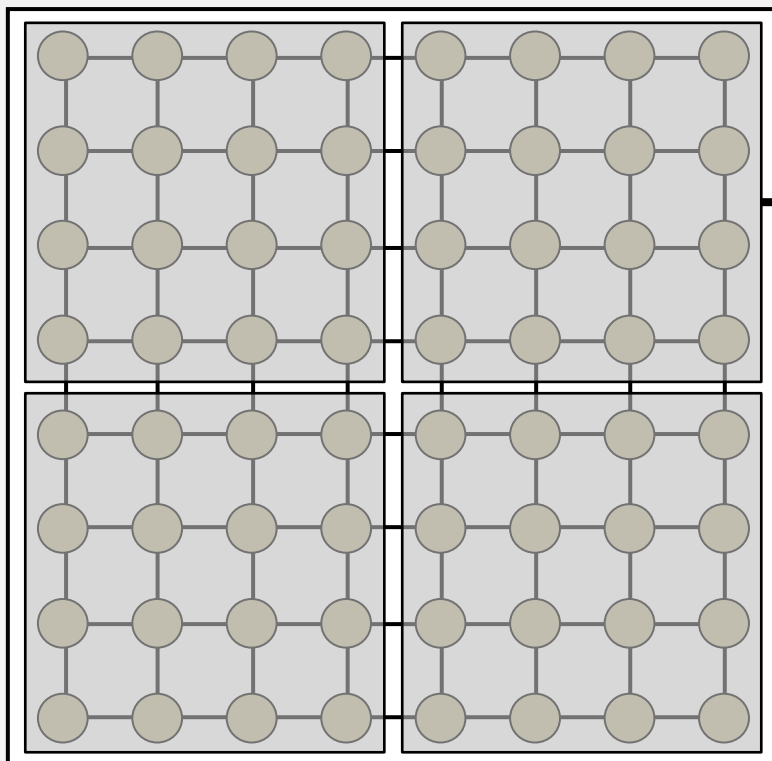
$$\frac{4(t_s + 8t_w)}{16 \times 6t_c} = \frac{4t_s + 32t_w}{96t_c}$$

$$\frac{8(t_s + t_w)}{6t_c}$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Relación Superficie / Volumen

- Análisis de la relación en el ejemplo ilustrativo.



$$\frac{8(t_s + 4t_w)}{16 \times 6t_c} = \frac{8t_s + 32t_w}{96t_c}$$

$$\frac{4(t_s + 8t_w)}{16 \times 6t_c} = \frac{4t_s + 32t_w}{96t_c}$$

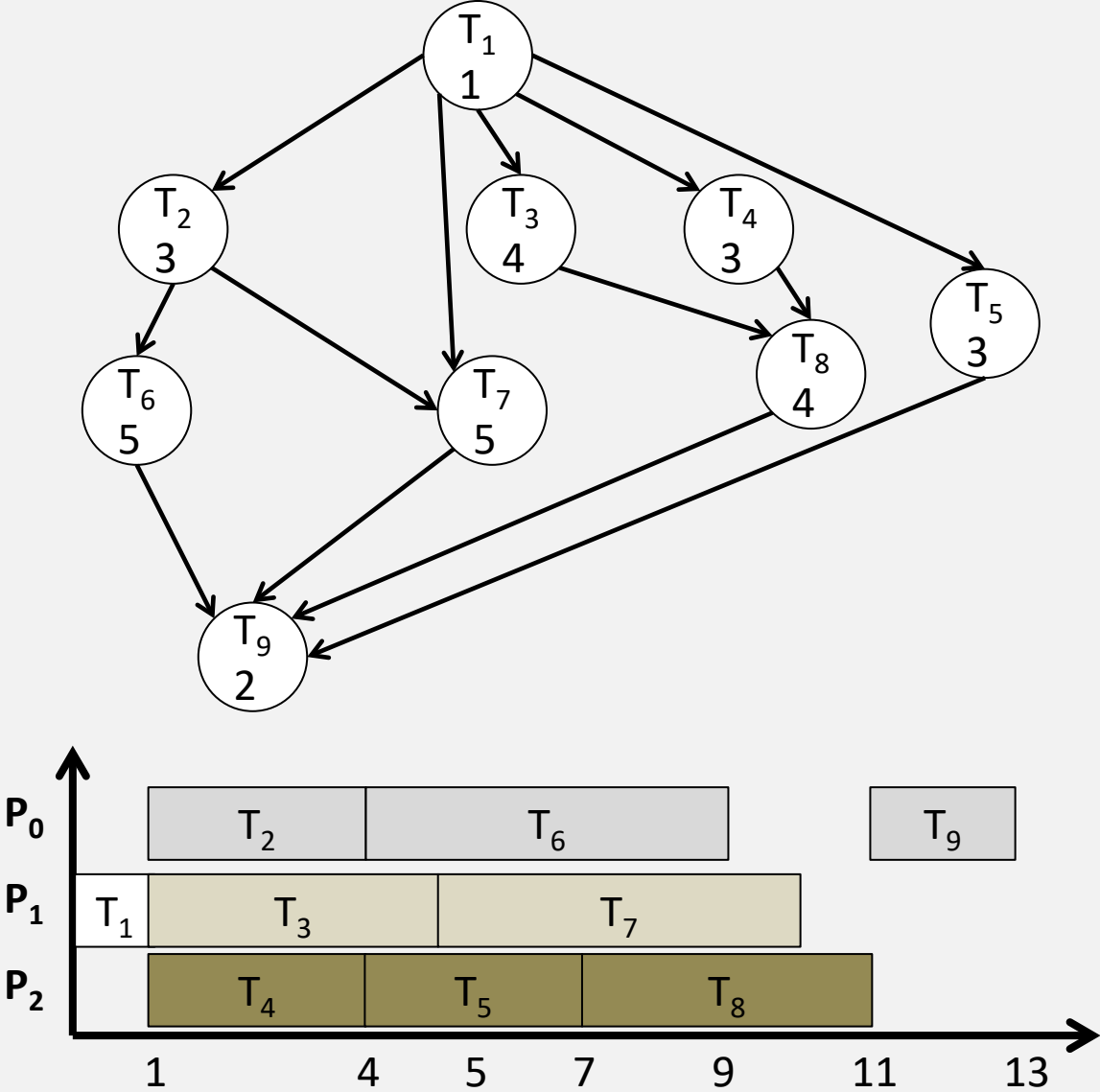
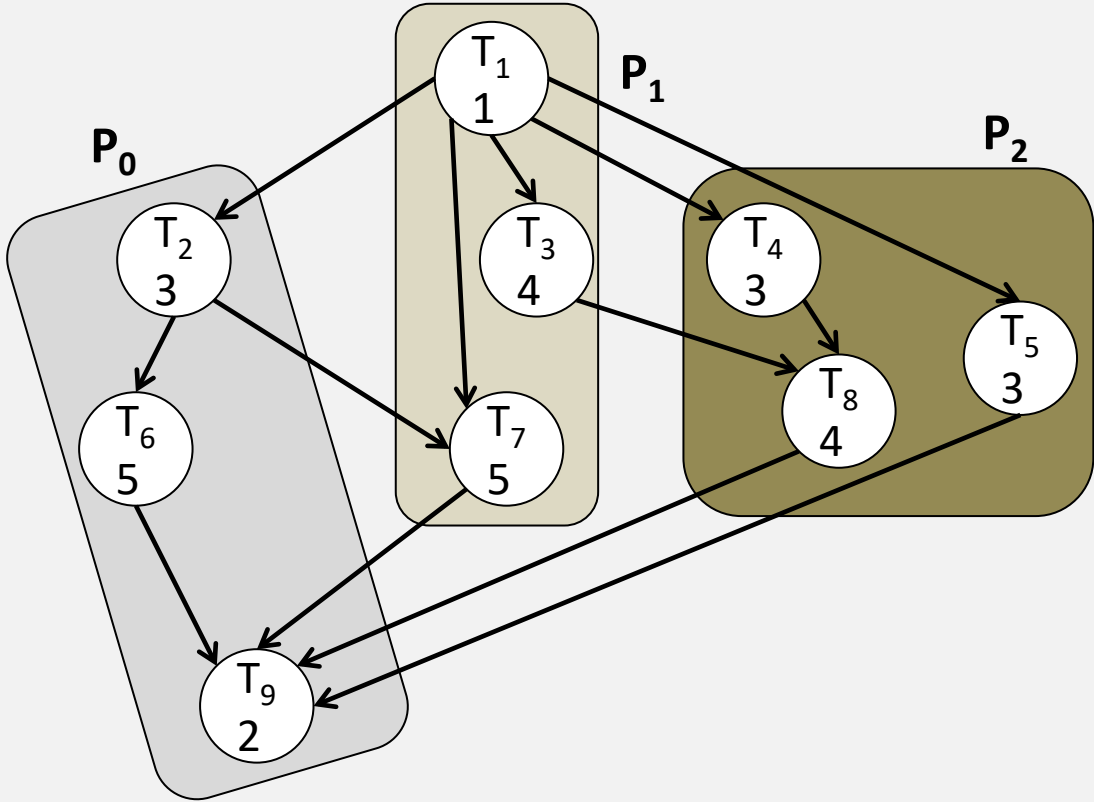
$$\frac{4(t_s + 8t_w)}{16 \times 6t_c} = \frac{4t_s + 32t_w}{96t_c}$$

$$\frac{8(t_s + t_w)}{6t_c}$$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Replicar

- Datos en paso de mensajes
- Computación y/o comunicaciones

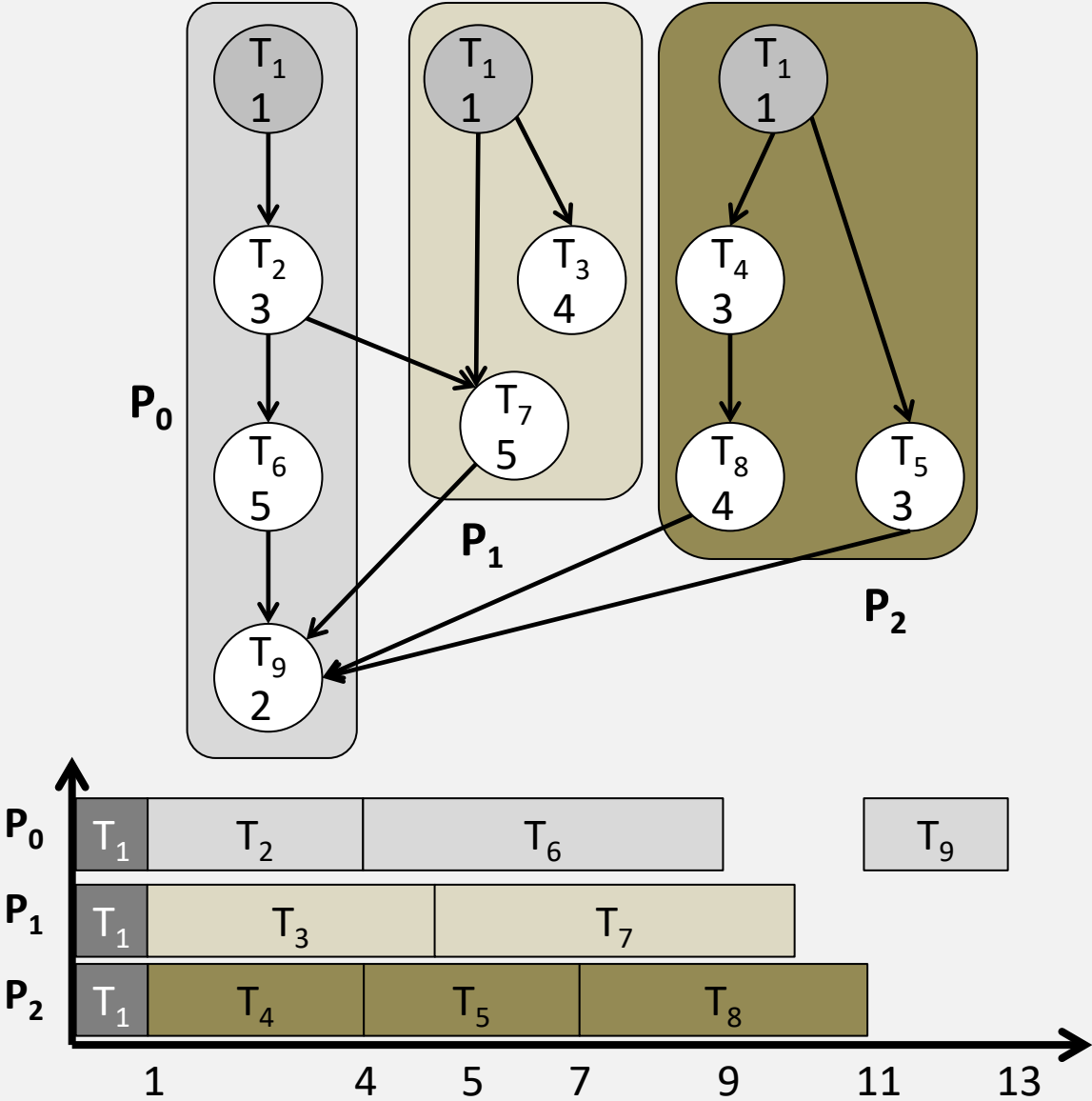
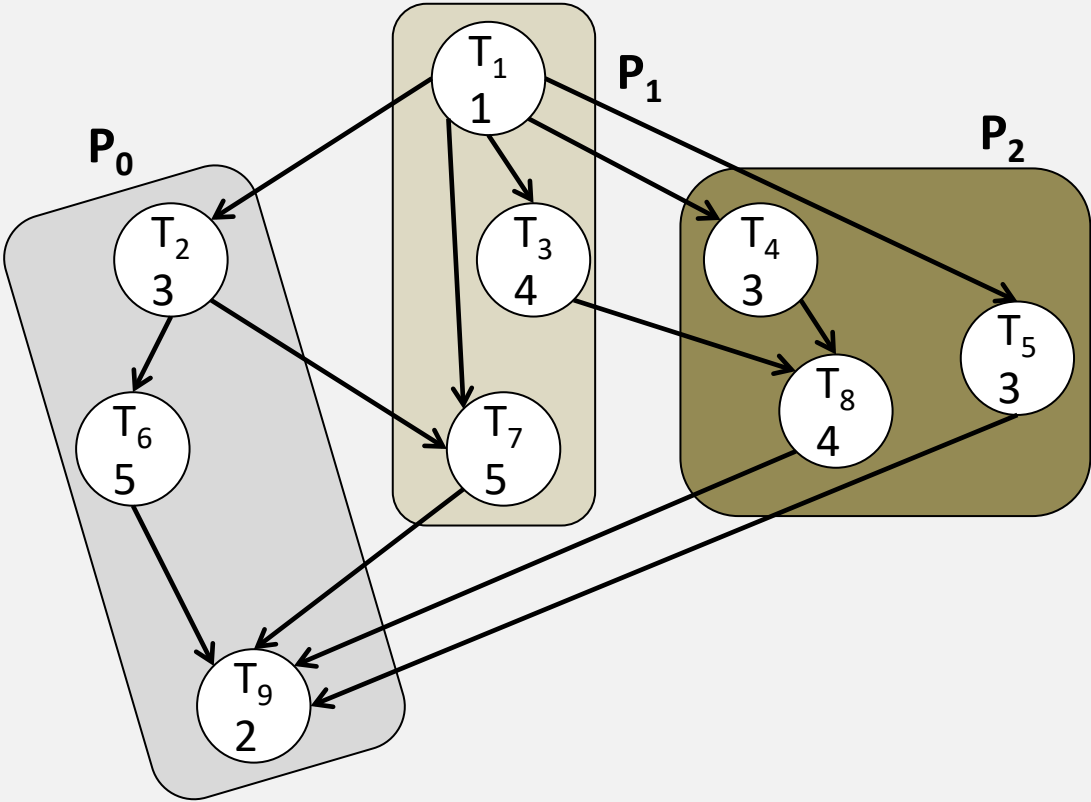




# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

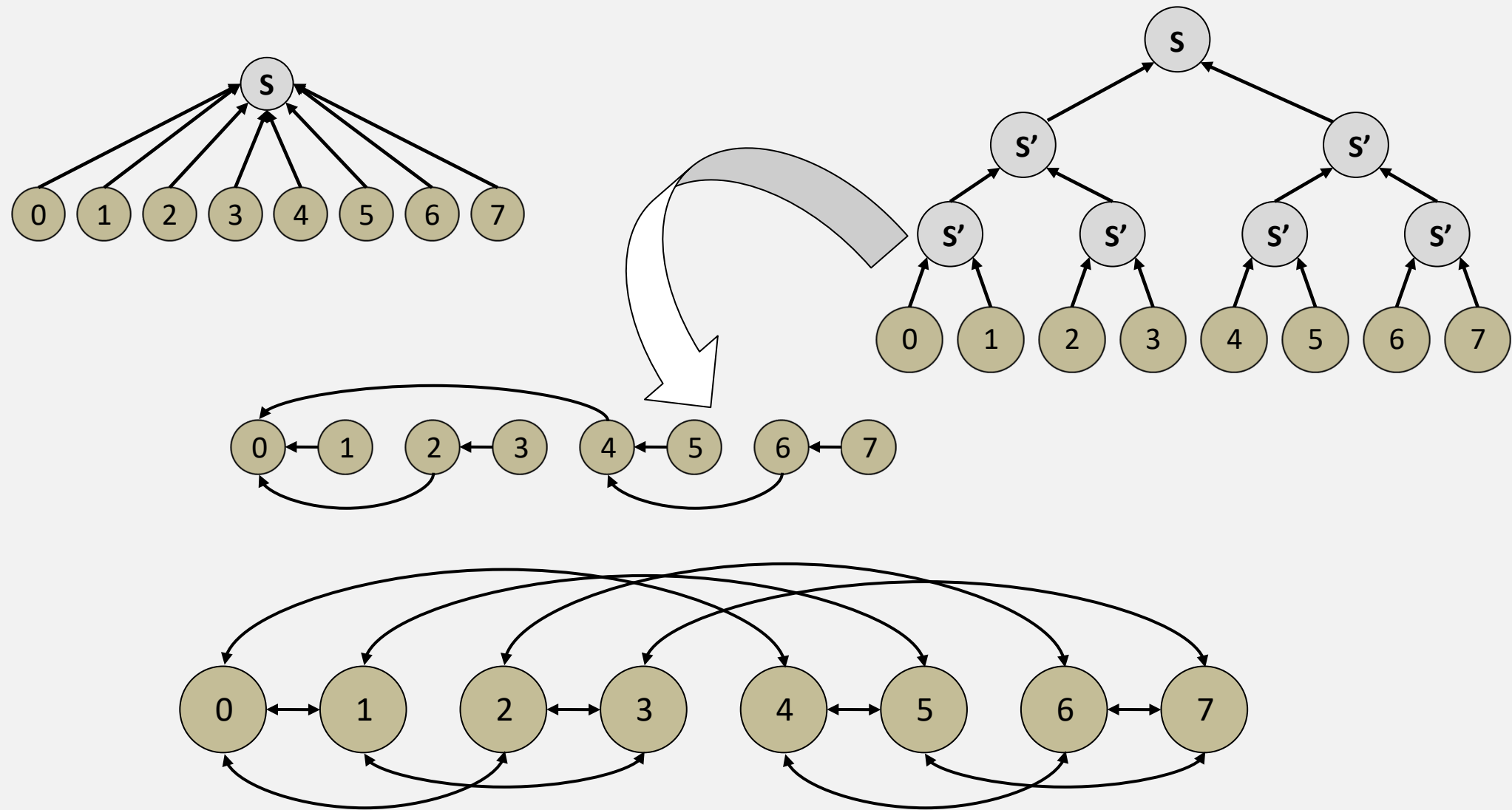
## Replicar

- Datos en paso de mensajes
- Computación y/o comunicaciones



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Replicar



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Agrupar: tareas no concurrentes y comunicaciones

- Agrupar tareas que no se pueden ejecutar concurrentemente disminuye el número de comunicaciones sin degradar el nivel de paralelismo.
- Agrupar datos por bloques de elementos contiguos.
- Agrupar tareas que se comuniquen mucho.
- Usar datos locales para almacenar resultados intermedios.

## Preservar la flexibilidad

- Un agrupamiento basado únicamente en el incremento del rendimiento puede limitar innecesariamente la escalabilidad, como por ejemplo:

Descomposición multidimensional  $\rightarrow$  a una dimensión

## Número de tareas óptimo

- $f(n, p, \text{analítico}, \text{empírico})$

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## 4. Asignar

- Determinar dónde y en qué orden se debe ejecutar.
  - En diseño: asignar o planificar tareas en procesos.
  - En ejecución: asignar procesos a procesadores.

## Objetivo

- Minimizar el tiempo global de ejecución. ¿Cómo?
  - Computación: tareas concurrentes en procesos (procesadores) distintos.
  - Comunicación: tareas que se comunican frecuentemente en el mismo proceso (procesador) o en procesos (procesadores) vecinos.
  - Inactividad: minimizar el desequilibrio de carga y las esperas.

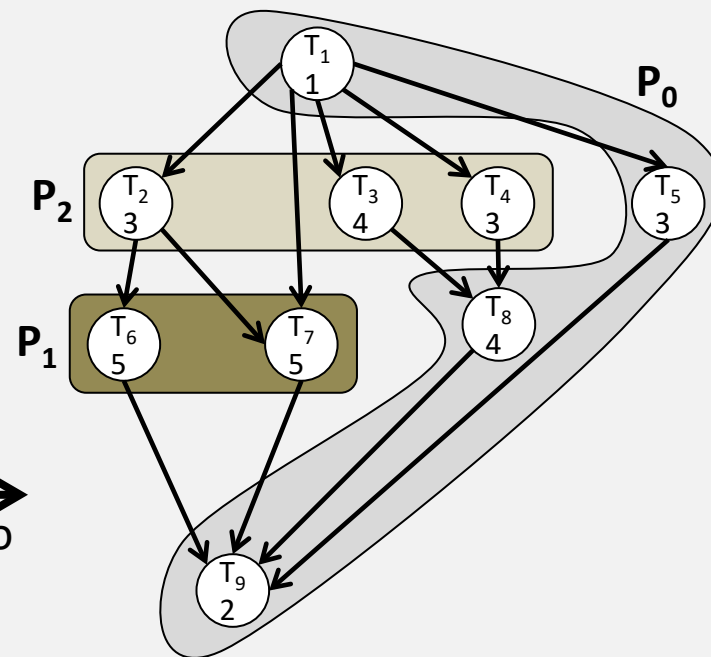
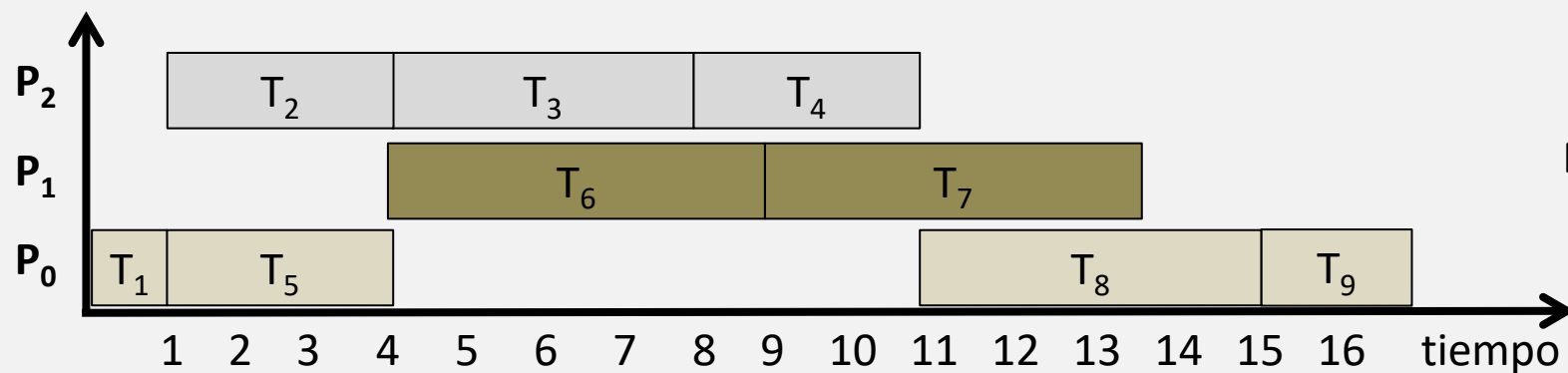
## Problemática

- Conflictos entre las estrategias.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

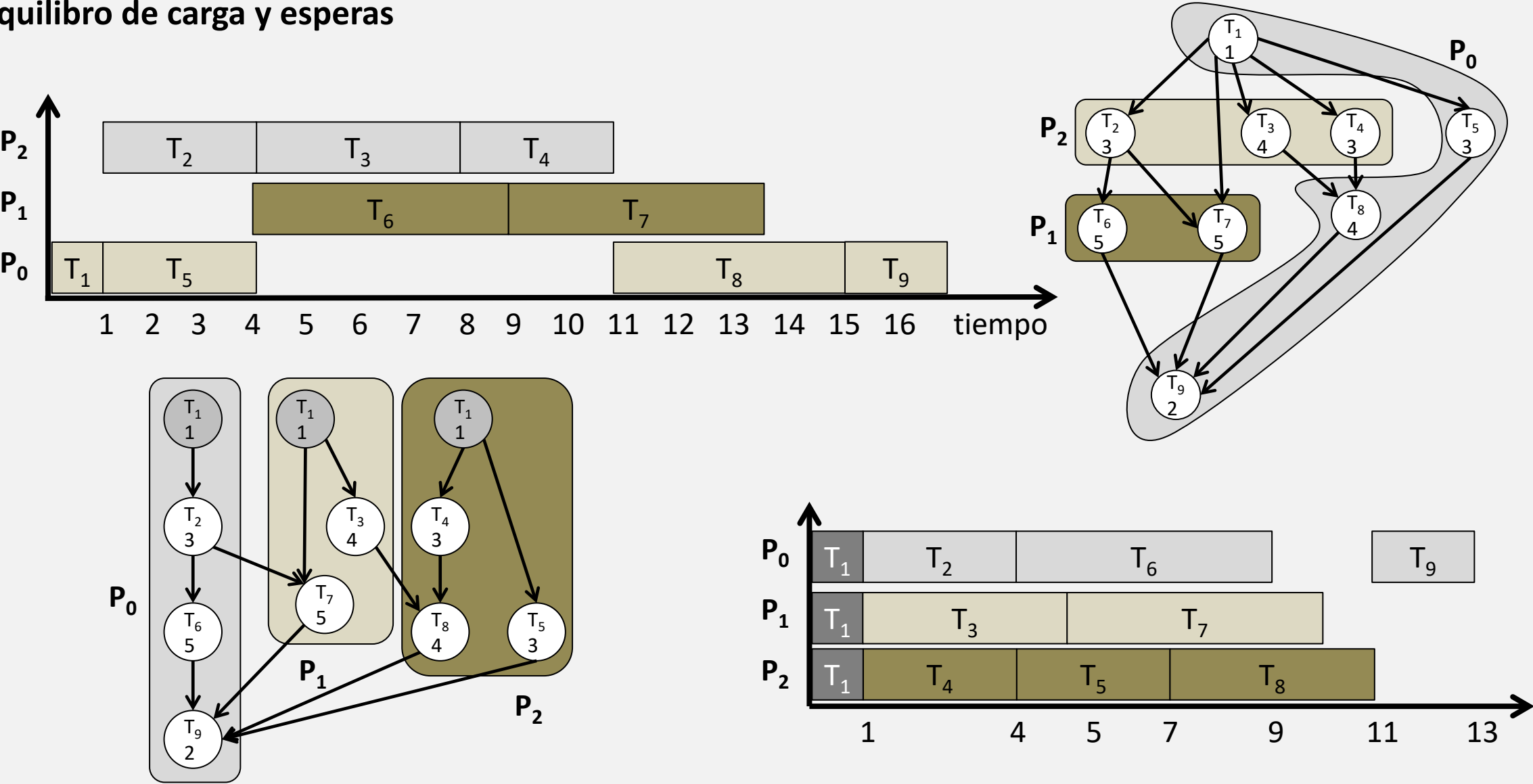
## Desequilibrio de carga y esperas

- Hay que equilibrar tanto los cálculos como las comunicaciones.
- Las esperas son dependencias que se observan en el grafo de dependencias. Asignar de forma balanceada no es suficiente.
- El ejemplo ilustra cómo esta distribución balanceada conlleva a una situación más compleja y con mayor grado de dependencias, respecto a la asignación realizada en [68](#).



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Desequilibrio de carga y esperas



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Estrategias Generales de Asignación

- Asignación estática o planificación determinista
  - Las decisiones de asignación se toman antes de la ejecución.
  - Es NP completo para el caso general. Sencillo de usar y no añade sobrecarga en tiempo de ejecución.
  - *Mapping problems* (Bokhari, 1981) y Lewis & El-Rewini (1992).
- Asignación dinámica
  - Se realiza en tiempo de ejecución, de forma centralizada o distribuida.
  - Sobrecarga debida a las transferencias entre los procesos (procesadores) por la toma de decisiones en tiempo de ejecución.
  - Ventaja: Son flexibles y aptos arquitecturas heterogéneas al no ser necesario conocer el comportamiento a priori.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

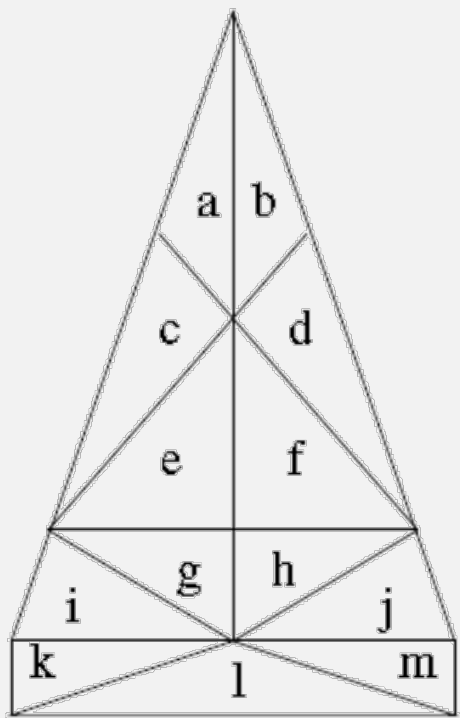
## Estrategias de Asignación Estáticas en descomposición del dominio

- Esquemas de distribución por bloques
  - Aptos con estructuras regulares (vectores, matrices, etc.)
  - Muchos tipos en función del número y uso de las dimensiones: por filas, por columnas, por bloques de filas, por bloques de columnas, por bloques 2D, etc.
- Esquemas de subdivisión de grafos
  - Cuando no son estructuras regulares.
  - Agrupar vértices (subdominios) de forma que cada subdominio tenga aproximadamente el mismo número de vértices y el número de aristas que conectan diferentes subdominios sea mínima.
  - Es, por lo general, NP completo → necesario usar heurísticos.

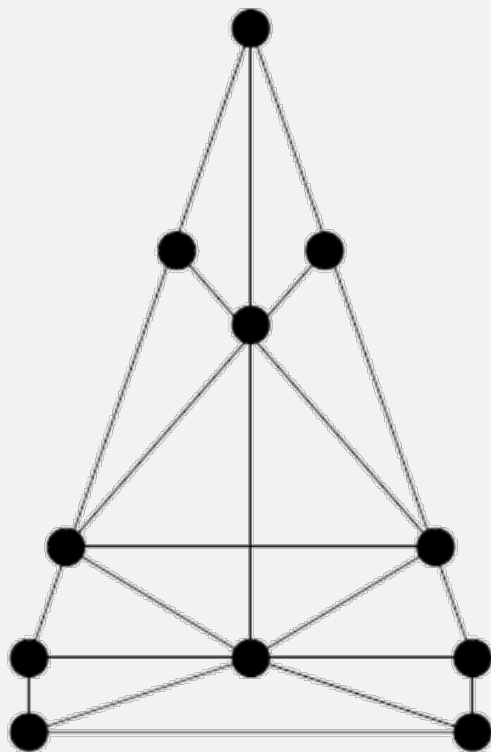


# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

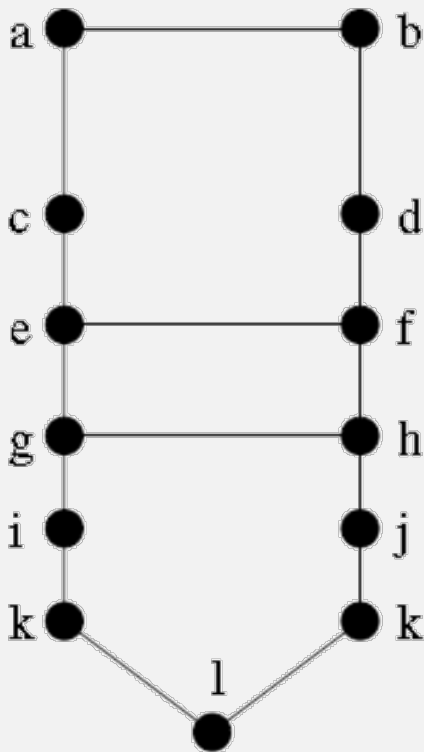
## Estrategias de Asignación Estáticas en descomposición del dominio



Malla Inicial (celdas)



Grafo de nodos (vértices)



Gafo de celdas

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Bisección Recursiva

- Planteamiento - ámbito
  - Divide y Vencerás sobre las computaciones para disminuir el coste de las comunicaciones.
  - Gran rendimiento en Mallas.
- Tipos
  - Bisección Recursiva Coordinada.
  - Bisección Recursiva no Balanceada.
  - Grafo de Bisección Recursiva.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Bisección Recursiva Coordinada

- Ventajas
  - Poco Costo.
  - Respecto a las computaciones realiza un particionado correcto.
- Inconvenientes
  - No disminuye el coste de las comunicaciones.
- Estrategia
  - Realiza cortes basados en las coordenadas físicas de los puntos.
  - En cada paso divide el dominio en dos subdominios a lo largo de una dimensión.
  - Típicamente, el corte se efectúa por el punto medio.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Bisección Recursiva No Balanceada

- Ventajas
  - Mejora el coste de las comunicaciones.
  
- Inconvenientes
  - Mayor coste que el anterior.
  
- Estrategia
  - No divide automáticamente una malla por la mitad.
  - Estudia las  $(p - 1)$  particiones posibles al considerar pares de mallas con:  
$$(1/p, (p - 1)/p), (2/p, (p - 2)/p), \dots, ((p - 1)/p, 1/p)$$
  - Selecciona la partición menos costosa.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Grafos de Bisección Recursiva

- **Ámbito**
  - Mallas complejas y comunicaciones no estructuradas.
- **Estrategia**
  - Se basa en la información relativa a la conectividad de los puntos de la malla.
  - Intenta disminuir el número de canales entre subdominios.
  - Considera la malla como un grafo de  $N$  vértices e identifica los vértices extremos del grafo.
  - Asigna el resto de los vértices a los dominios de los extremos en función de sus distancias.
  - Se repiten los pasos anteriores hasta obtener el número de tareas deseadas.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Algoritmos de balanceo de carga Probabilísticos

- Planteamiento - ámbito
  - Asignación aleatoria procurando un balanceo de las computaciones.
  - Número reducido de comunicaciones locales.
- Ventajas
  - Simple, poco costoso y escalable.
- Inconvenientes
  - Puede ser poco eficiente para las comunicaciones.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Balanceado basados en Distribuciones Cíclicas

- Planteamiento - ámbito
  - Variante del método probabilístico.
  - Parte de la existencia de una numeración u ordenación.
  - Número reducido de comunicaciones locales.
  
- Estrategia
  - Asigna tareas a procesos (procesadores) de igual número.
  - La operación de igualdad se puede definir como: (*número de tareas MOD dimensión*), donde dimensión puede ser:
    - a) el número de nodos de la malla,
    - b) el número de filas o columnas de la malla, etc.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Estrategias de Asignación Estáticas en descomposición funcional

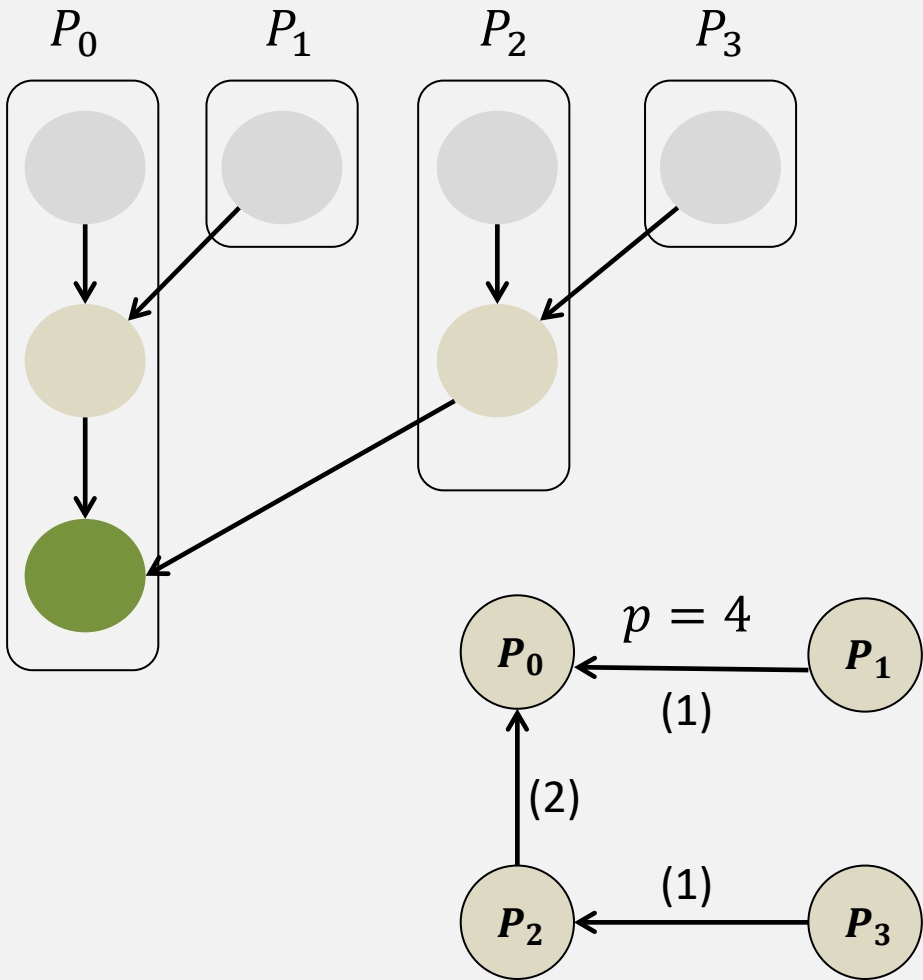
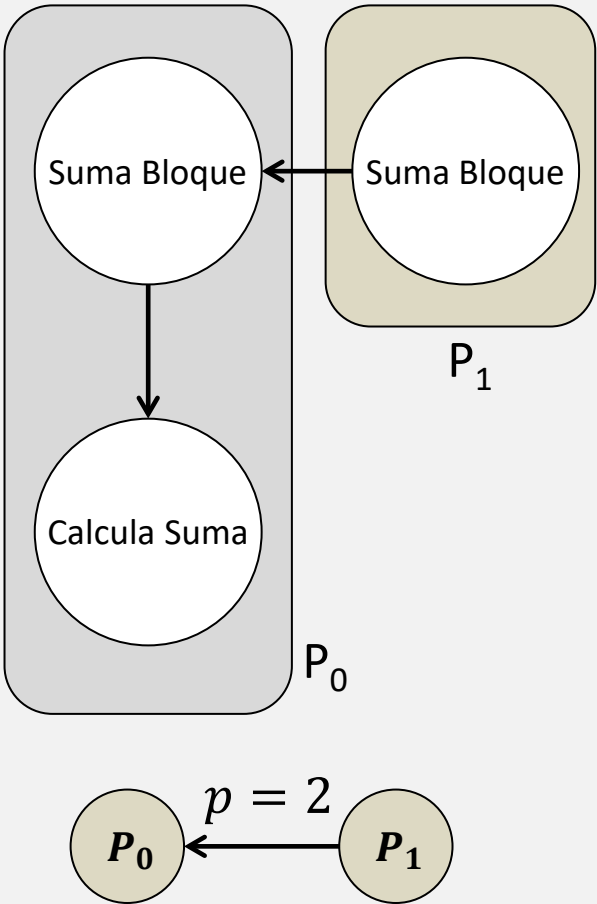
- Se supone un grafo de dependencias estático y costes de las tareas conocidos.
- El problema vuelve a ser NP completo.
- Existen situaciones para las que se conocen algoritmos óptimos o enfoques heurísticos, por ejemplo:
  - Árbol binomial
    - La asignación óptima se obtiene agrupando nodos de diferentes niveles con una relación de dependencia.
  - Hipercubo
    - Generalización del anterior.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Árbol Binomial

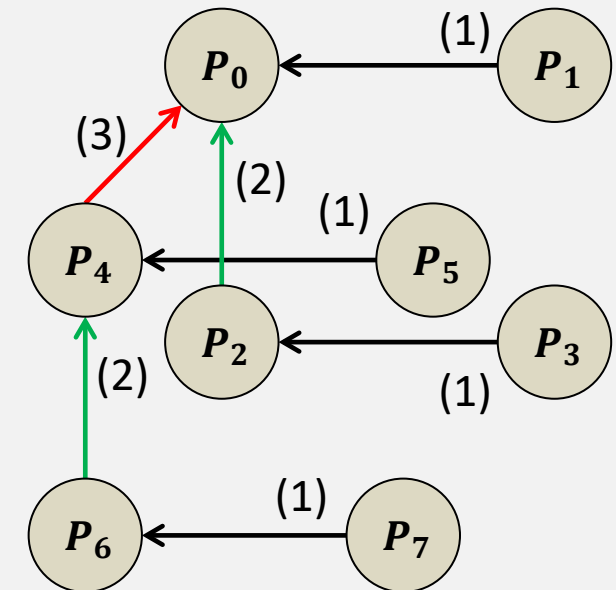
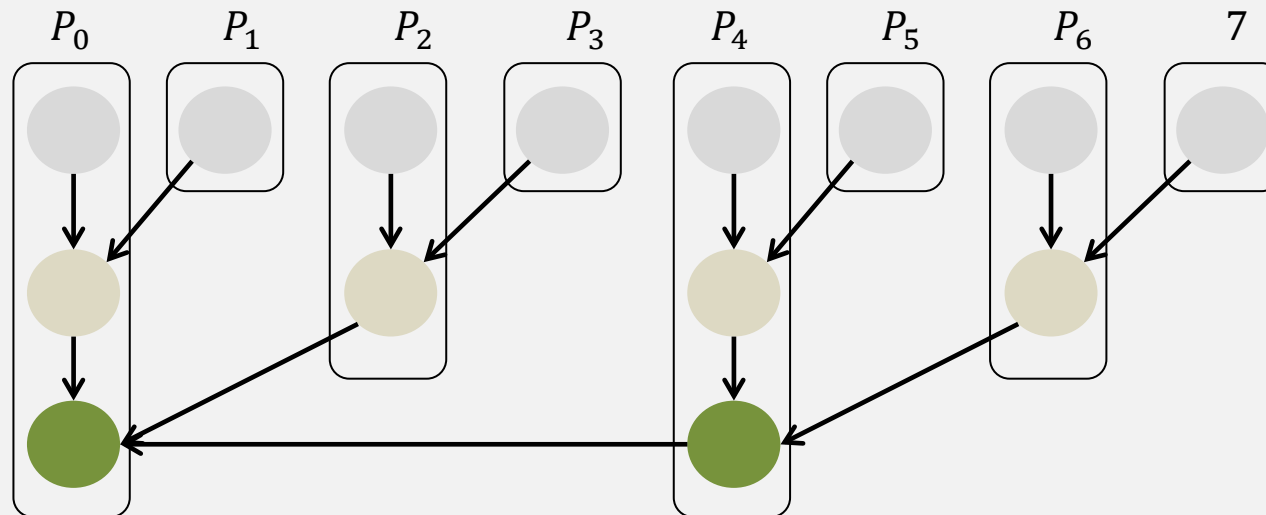
- Ejemplo: producto escalar



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Árbol Binomial

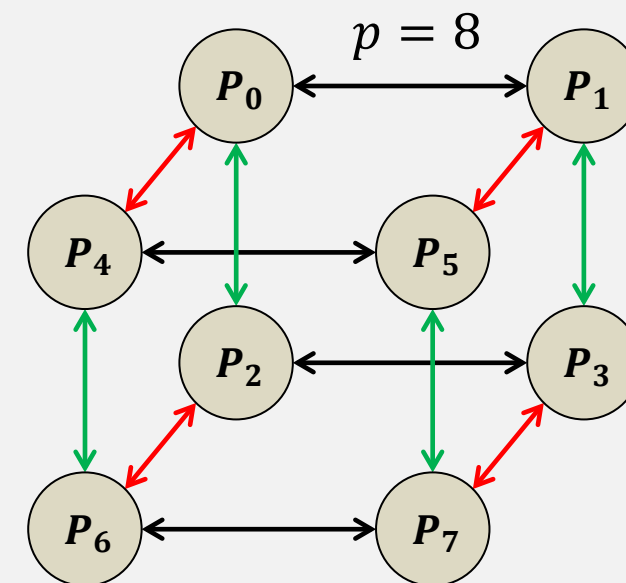
- Es un árbol binario con  $\log(n)$  niveles, siendo  $n$  el número de hojas.
- Un árbol binomial de orden  $k$  tiene  $2^k$  nodos y profundidad  $k$ .
- En un árbol binario con  $p = 2^k$  hoja una asignación óptima consiste en un árbol binomial con  $p$  procesos.
- Un árbol binomial de orden  $k$  se forma uniendo dos árboles binomiales de orden  $k - 1$ . El nodo raíz del nuevo árbol será uno de los nodos raíz de los árboles unidos.
- Utilizando un árbol binomial es posible realizar la suma de  $p$  valores en  $\log(p)$  pasos.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Hipercubo

- ¿Y si se desea el producto escalar con réplica?
- Método 1
  - Recorrer el árbol binomial hasta la raíz para obtener la suma.
  - Recorrer el árbol binomial hasta las hojas para propagar el resultado.
  - $2\log(p)$
- Método 2: extiende el concepto de árbol binomial
  - Replicar las comunicaciones y las computaciones.
  - La distancia máxima entre cualquier par de los  $2^p$  nodos es  $p$  obligando a que cada nodo tenga exactamente  $p$  vecinos: Hipercubo de grado  $p$ .
  - En cada paso los  $p_i$  intercambian su valor con un vecino y realiza la suma de su valor con el recibido. Cada paso usa una dimensión diferente.
  - En  $\log(p)$  todos tienen la suma.



# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Estrategias de Asignación Dinámicas

- Pueden ser centralizados o distribuidos.
- Necesitan un mecanismo de detección de fin para finalizar.

## Estrategia de Asignación Dinámicas basada en Información Local

- Planteamiento - ámbito
  - Problemas con cambios constantes en la carga. No necesita un conocimiento global de la computación.
- Ventajas
  - Coste reducido, frente a los anteriores.
- Inconvenientes
  - Peor rendimientos que los anteriores.
- Estrategia
  - Considerar únicamente un conjunto de vecinos para el estudio de la carga de computación.
  - Periódicamente cada procesador compara con los vecinos y transfiere su exceso de computación.

# FUNDAMENTOS DEL DISEÑO DE ALGORITMOS: ENFOQUE METODOLÓGICO

## Estrategia de Asignación Dinámicas basada en algoritmos de planificación

- **Ámbito**
  - Descomposición funcional. Número reducido de comunicaciones locales.
- **Inconvenientes**
  - Complejidad en la distribución de la computación. Presenta el problema de la parada en algoritmos totalmente distribuidos.
- **Estrategia**
  - Mantener una lista, centralizada o distribuida, de las tareas. Las nuevas tareas se añaden a la lista en espera de que se les asigne un procesador.
- **Modelos estándar**
  - Gestor-Trabajador (efectivo cuando el número de trabajadores es alto) y Jerarquía de Gestores-Trabajadores (con gestores de cachés para evitar conflictos).
  - Esquemas Distribuidos (Cada procesador posee su propia lista).