

Auxiliar N°3

Programación paralela

Auxiliar: Pablo Pizarro R. [@ppizarro](#)

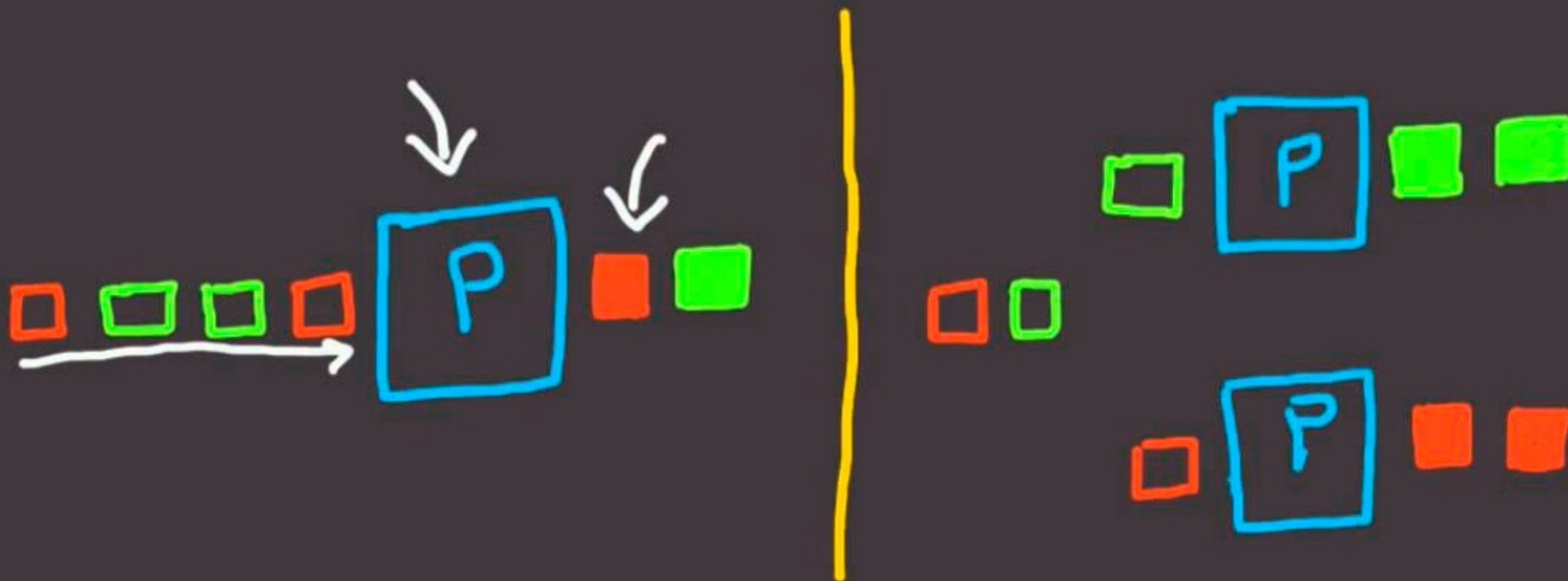
Introducción

- Capítulo 1: Qué es la programación paralela
- Capítulo 2: *Data parallel vs Task parallel*
- Capítulo 3: Patrones programación paralela
- Capítulo 4: ¿Con qué se programa en paralelo?
- Capítulo 5: Programación en GPU

1 - ¿Qué es la programación paralela?

Programa que ejecuta una serie de operaciones en paralelo (al mismo tiempo)

Parallel Computing



¿Qué es la programación paralela?

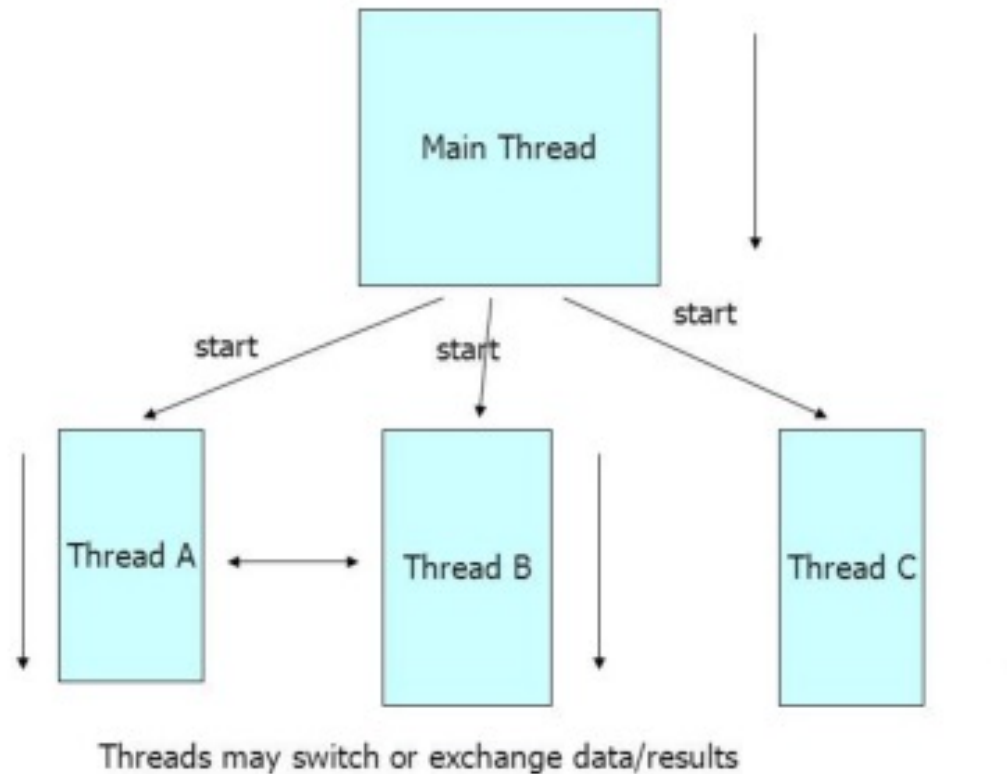
- Puede convertir la ejecución a una no determinista.
- Toma ventaja de las arquitecturas actuales.
- Hay que mirar los problemas desde otro punto de vista.

Nociones básicas

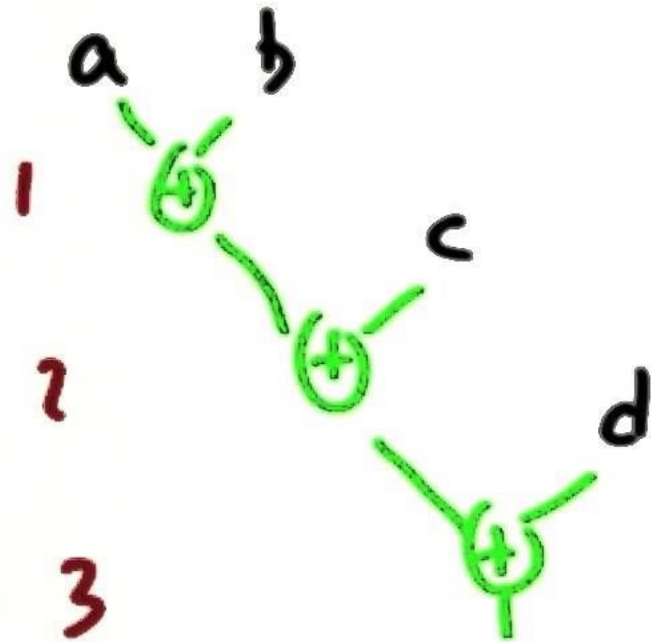
- **Task:** Secuencia de instrucciones que deben ejecutarse secuencialmente.
- **Ejecución concurrente:** Múltiples tareas independientes que **pueden** ejecutarse simultáneamente. Si dos tareas son dependientes entonces no son concurrentes.

Nociones básicas

A Multithreaded Program



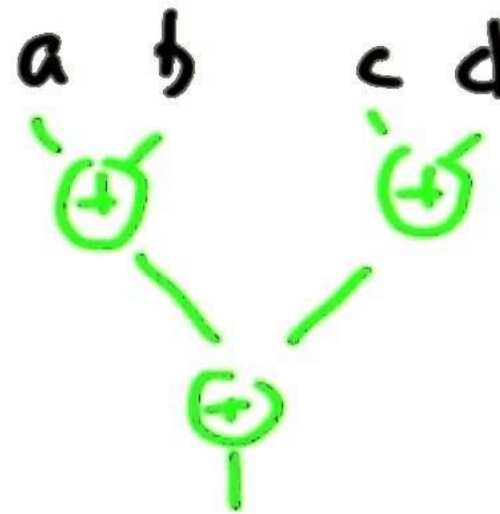
SERIAL REDUCE



$$((a+b)+c)+d$$

3 WORK
STEPS

PARALLEL REDUCE




$$(a+b) + (c+d)$$

3 WORK
STEPS

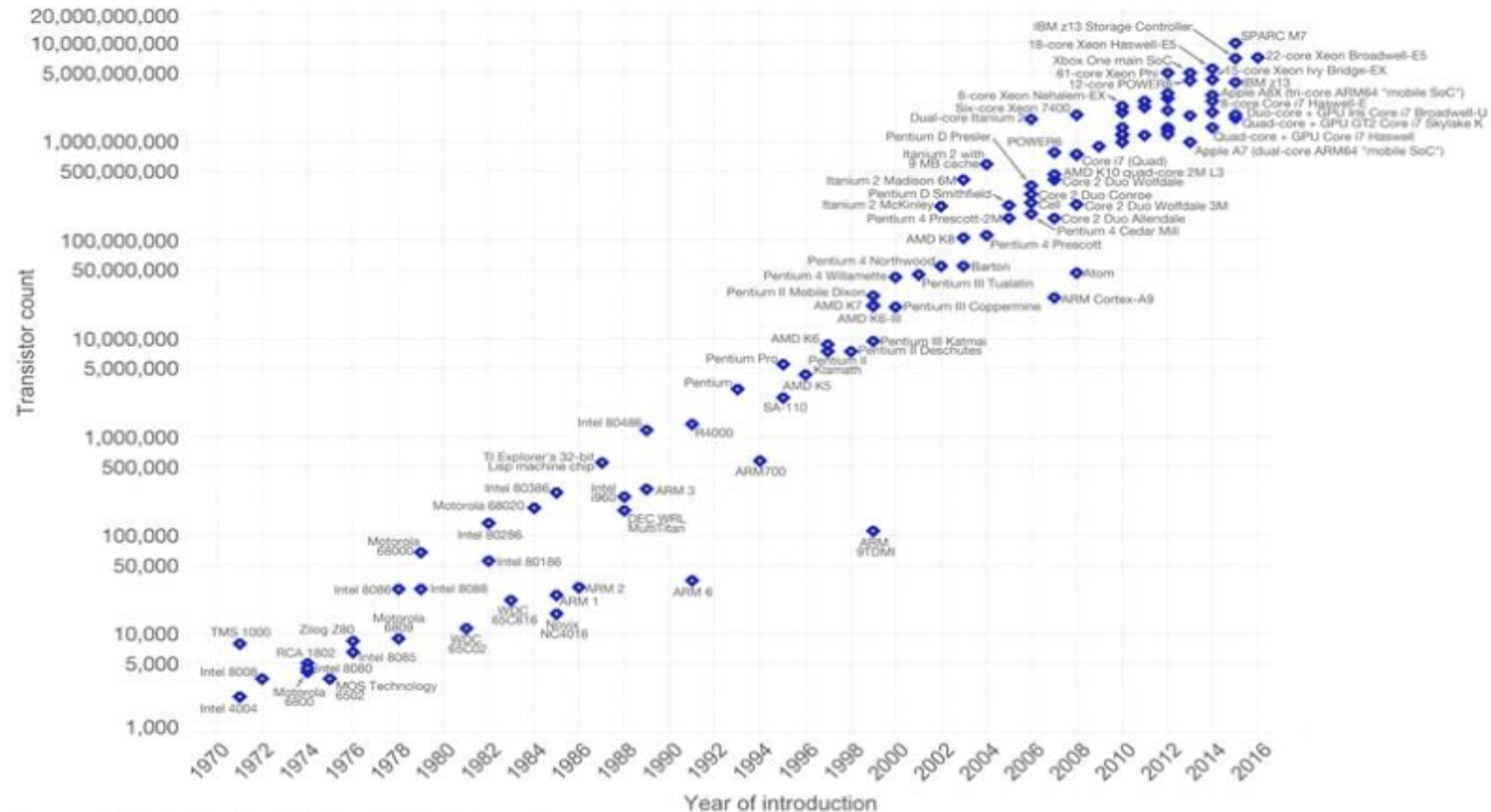
¿Qué ganamos con el paralelismo?

- Una solución más rápida.
- Resolver problemas más grandes.
- Uso efectivo de los recursos del computador.

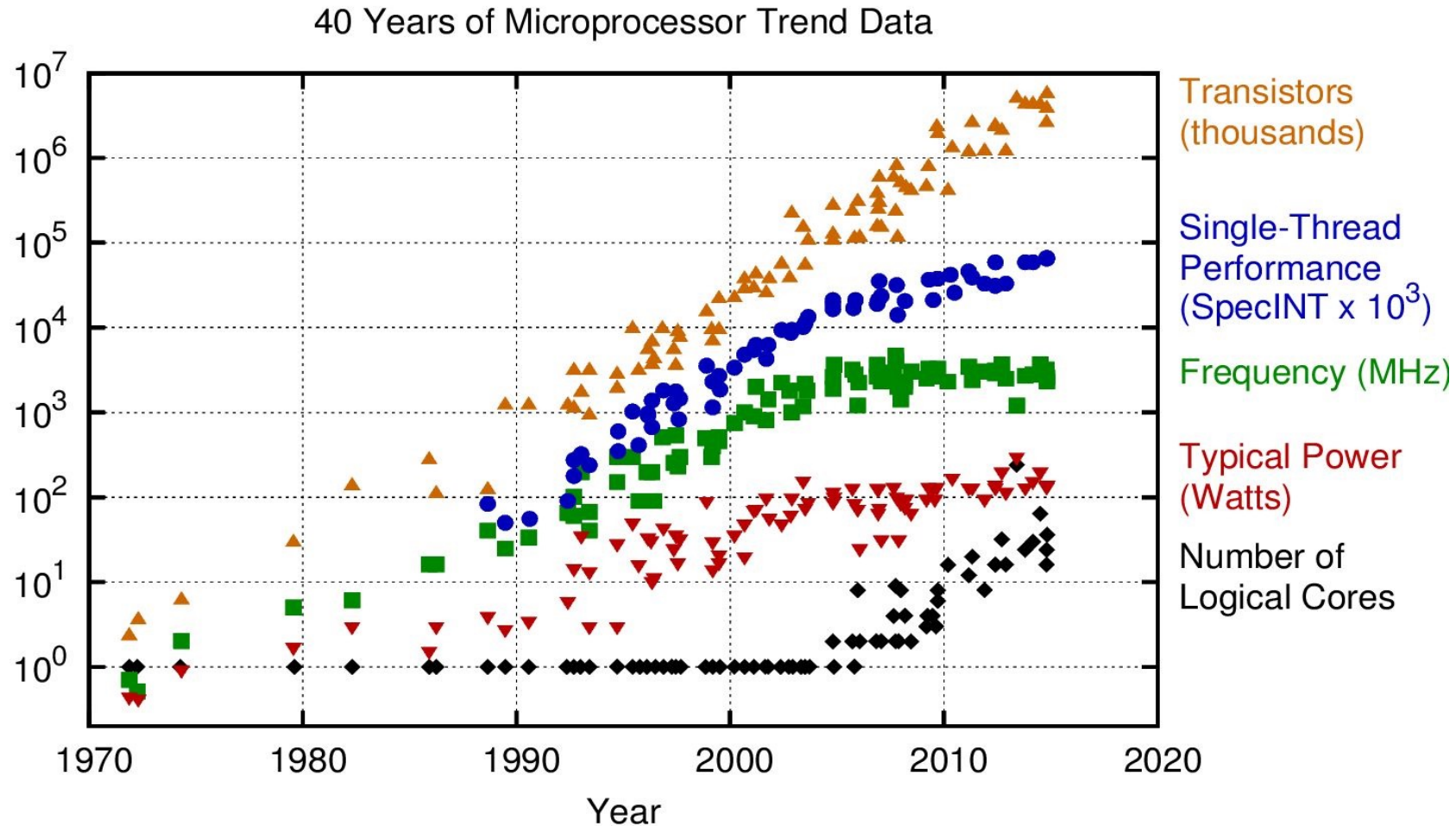
¿Para qué usar paralelismo si podemos mejorar los procesadores?

 Intel Core-X Series (Kabylake-X, Skylake-X)							
Processor	Cores/ Threads	L3 Cache	PCIe Lanes	Base Clock	Turbo Clock 2.0	Turbo Clock 3.0	Launch
Core i9-7920X	12C/24T	16.5 MB	44	TBD	TBD	TBD	August
Core i9-7900X	10C/20T	13.75 MB	44	3.3 GHz	4.3 GHz	4.5 GHz	June
Core i9-7820X	8C/16T	11 MB	28	3.6 GHz	4.3 GHz	4.5 GHz	June
Core i9-7800X	6C/12T	8.25 MB	28	3.5 GHz	4.0 GHz	-	June
Core i7-7740K	4C/8T	8 MB	16	4.3 GHz	4.5 GHz	-	June
Core i7-7640K	4C/4T	6 MB	16	4.0 GHz	4.2 GHz	-	June

¿Qué nos impide aumentar la cantidad de transistores?

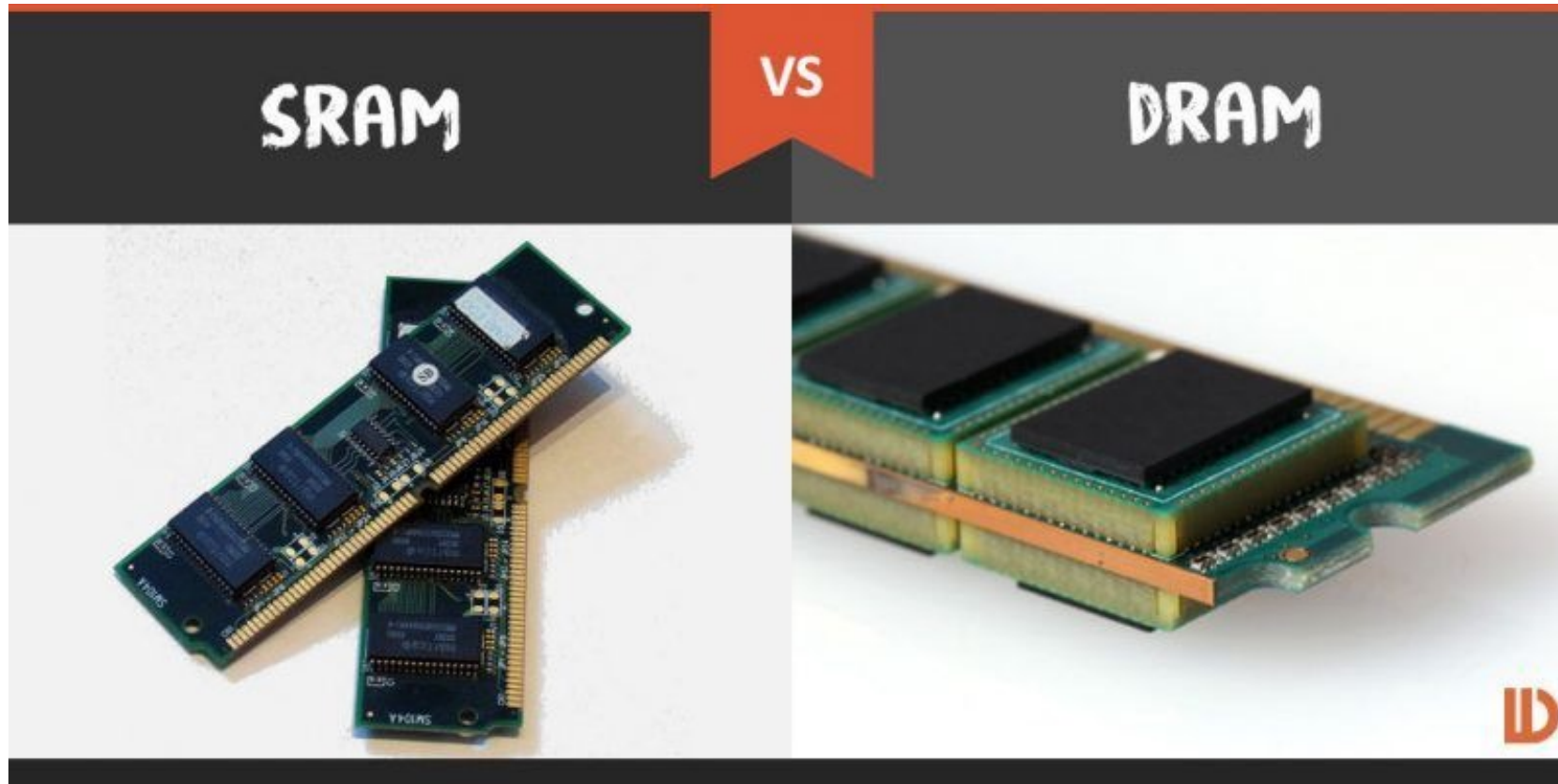


¿Qué nos impide aumentar la cantidad de transistores?



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

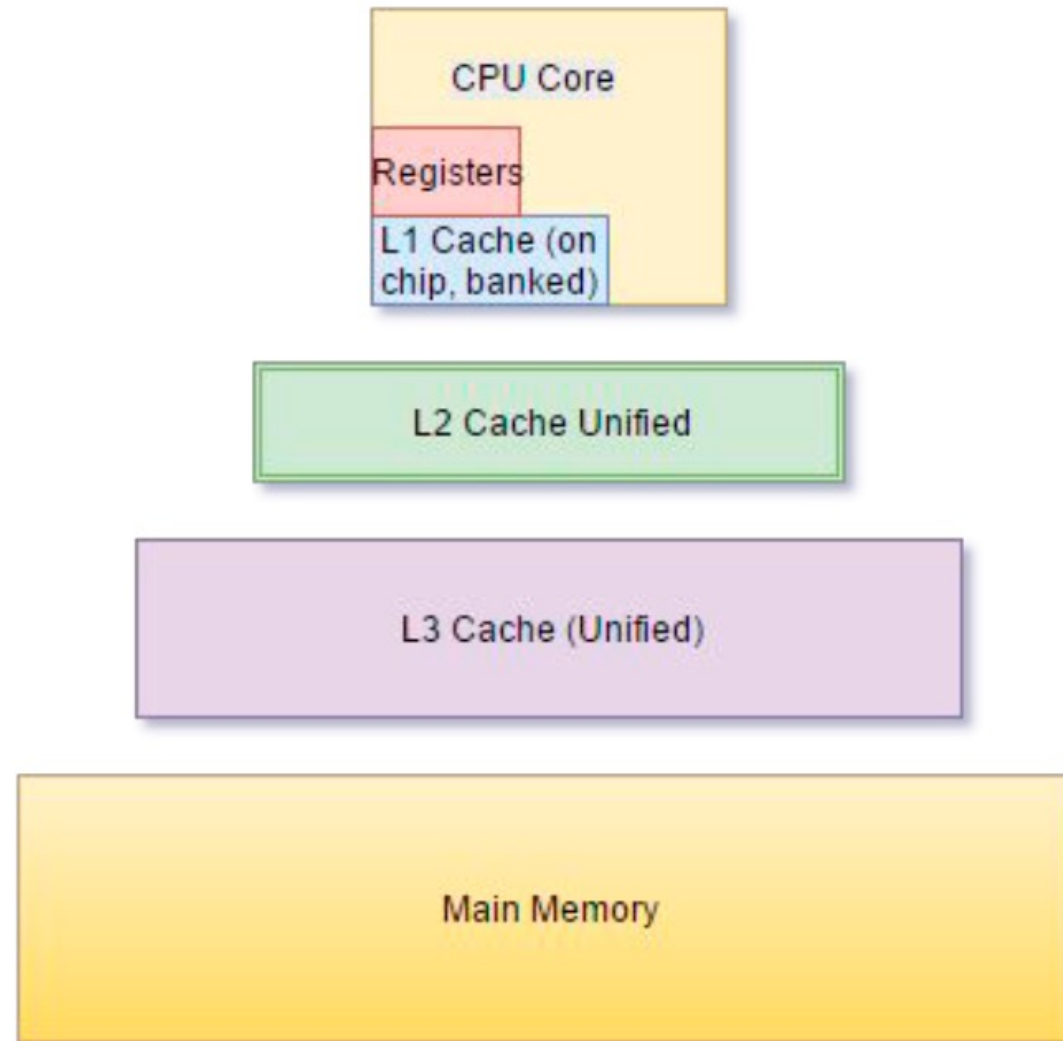
¿Cuáles son las memorias del procesador?



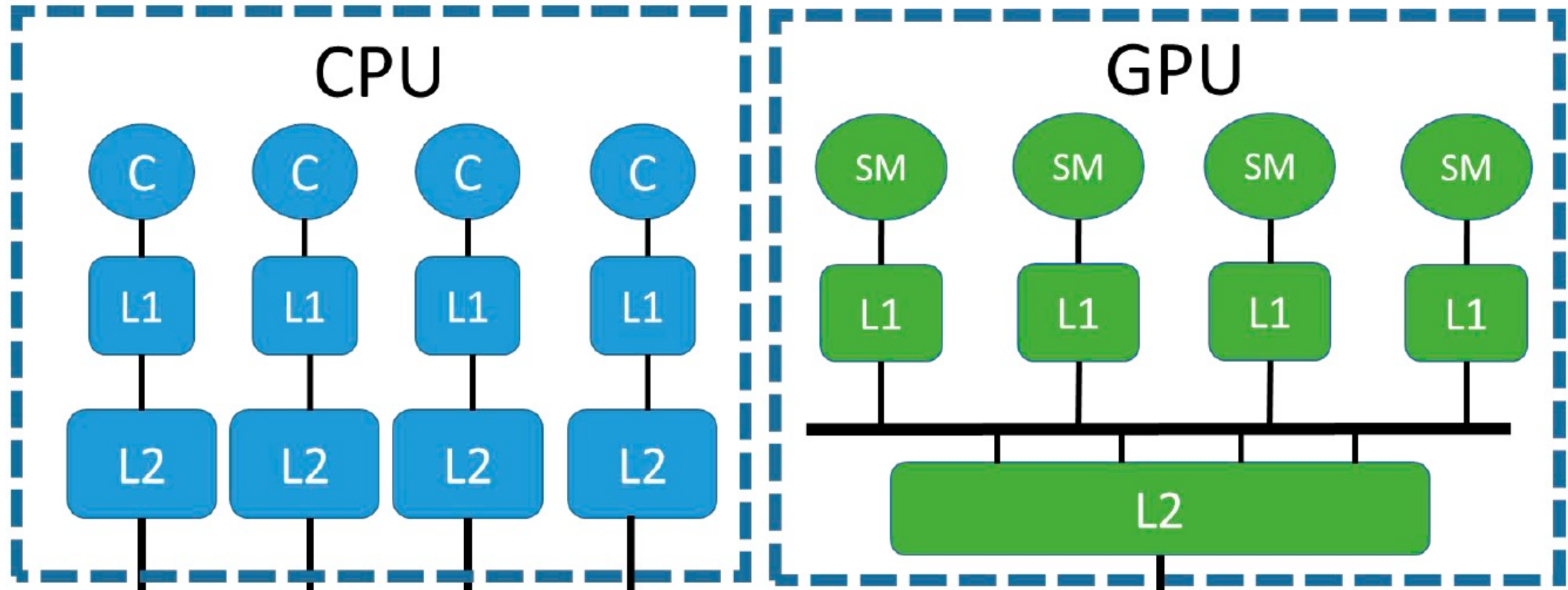
¿Qué tipos existen?

Memory technology	Typical access time	\$ per GB in 2004
SRAM	0.5–5 ns	\$4000–\$10,000
DRAM	50–70 ns	\$100–\$200
Magnetic disk	5,000,000–20,000,000 ns	\$0.50–\$2

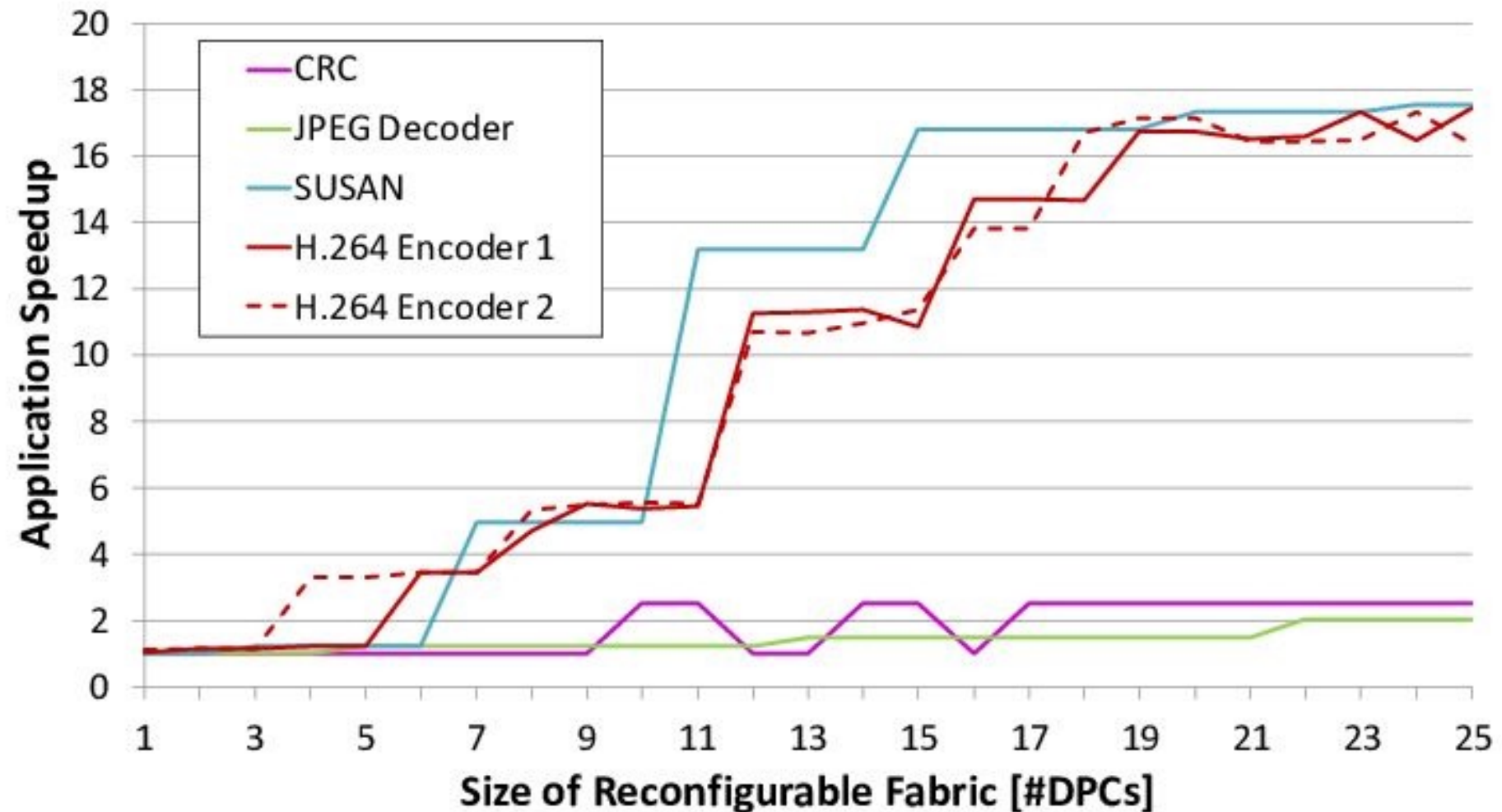
Niveles de memoria



Jerarquía de memoria en la GPU



Efectos del cache en los resultados

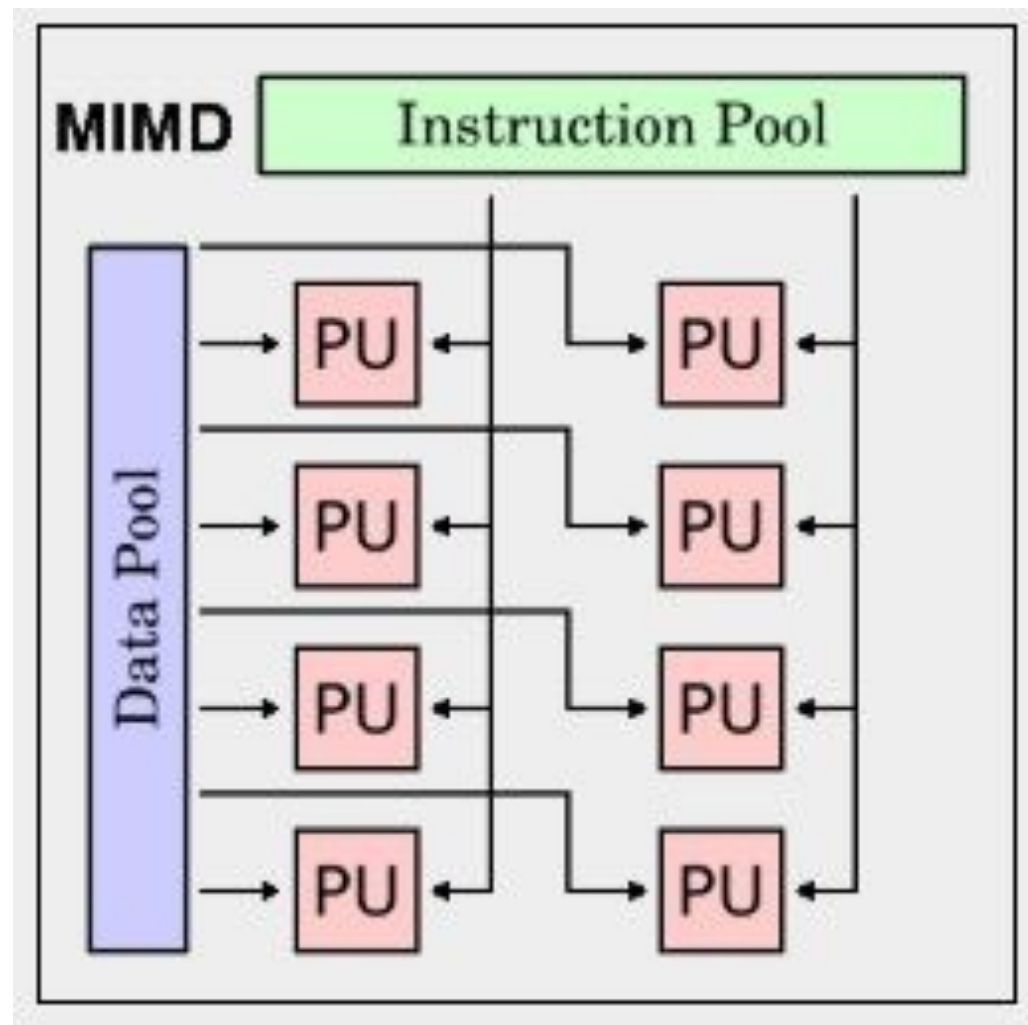


2 - *Data parallel vs Task parallel*

Flynn's taxonomy of processors

		Instruction Streams	
		one	many
Data Streams	one	SISD traditional von Neumann single CPU computer	MISD May be pipelined Computers
	many	SIMD Vector processors fine grained data Parallel computers	MIMD Multi computers Multiprocessors

Task Parallel



Task Parallel - Fibonacci

step1) $x = []$
 $a = 1$
 $b = 1$
 $c = a + b$

step2)

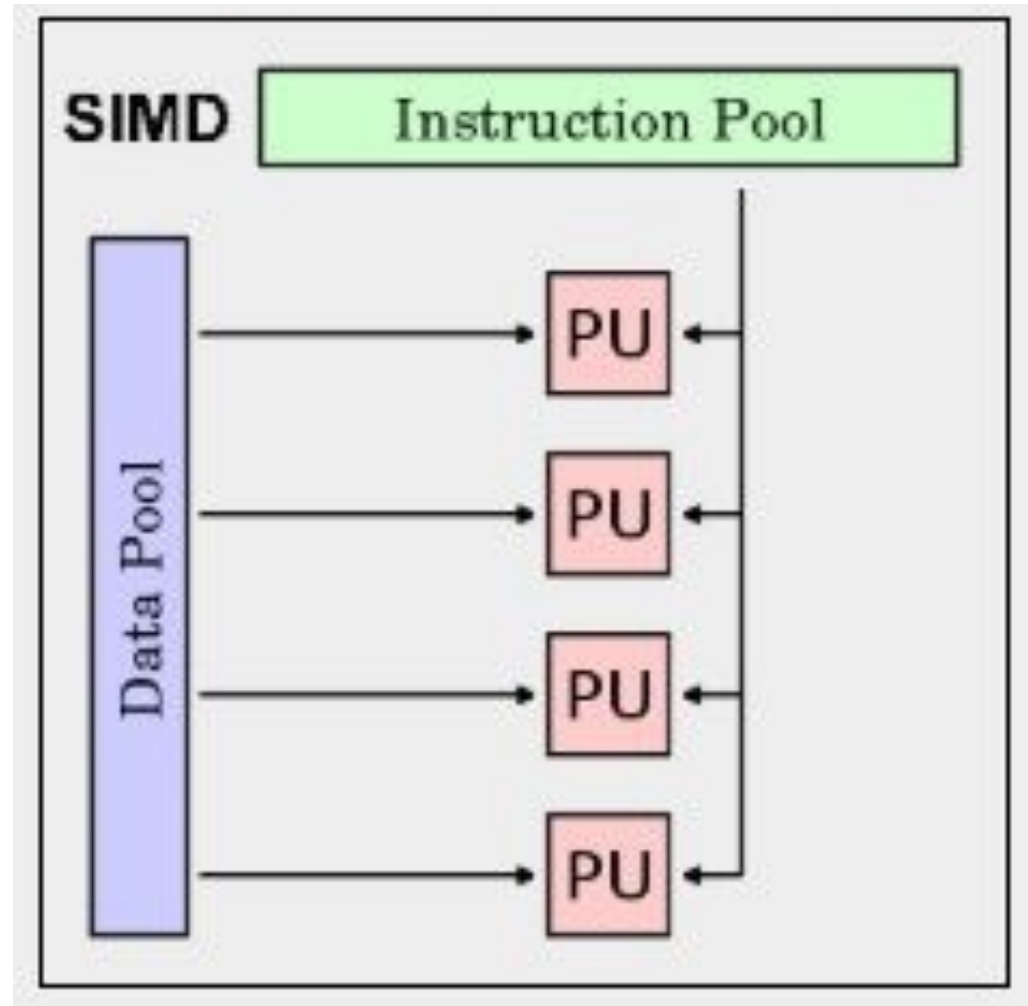
for loop

107006626638275893676498058445739688...

[1,1,2,3,5,8,13,21,34...]

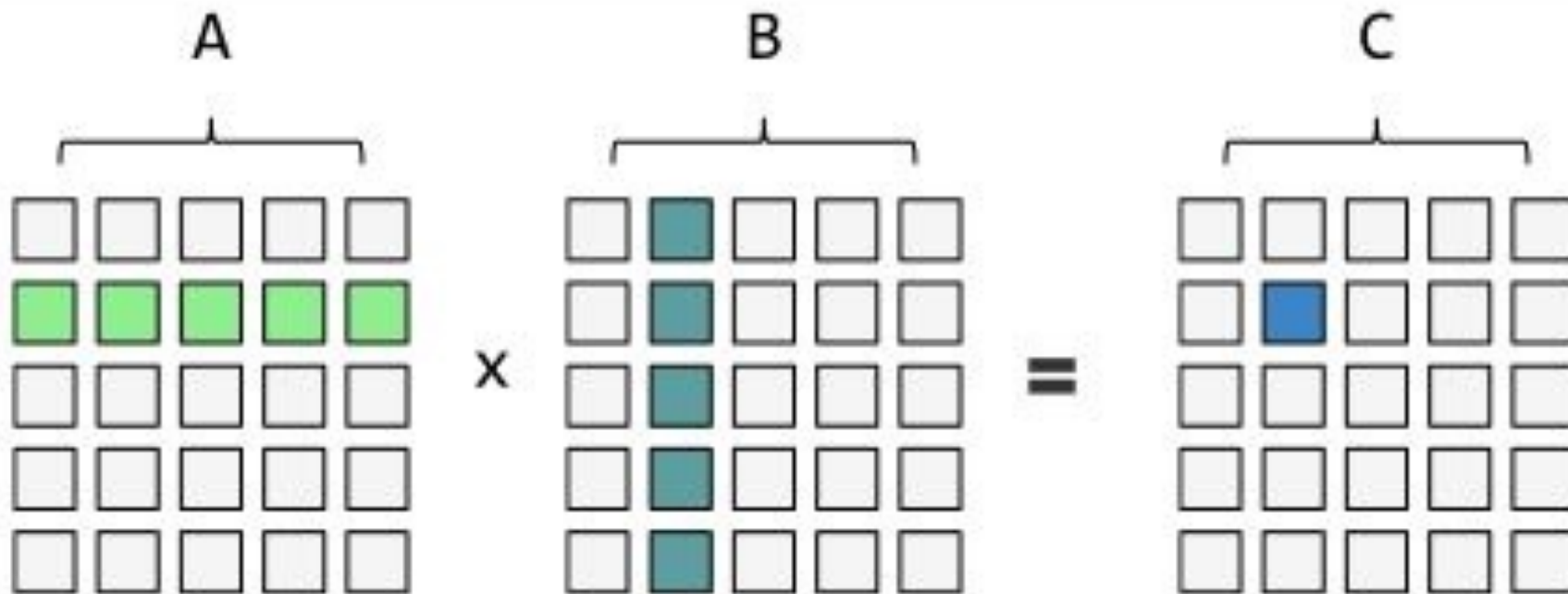
fibonacci sequence

Data Parallel



Data Parallel - Multiplicación de matrices

MATRIX MULTIPLICATION

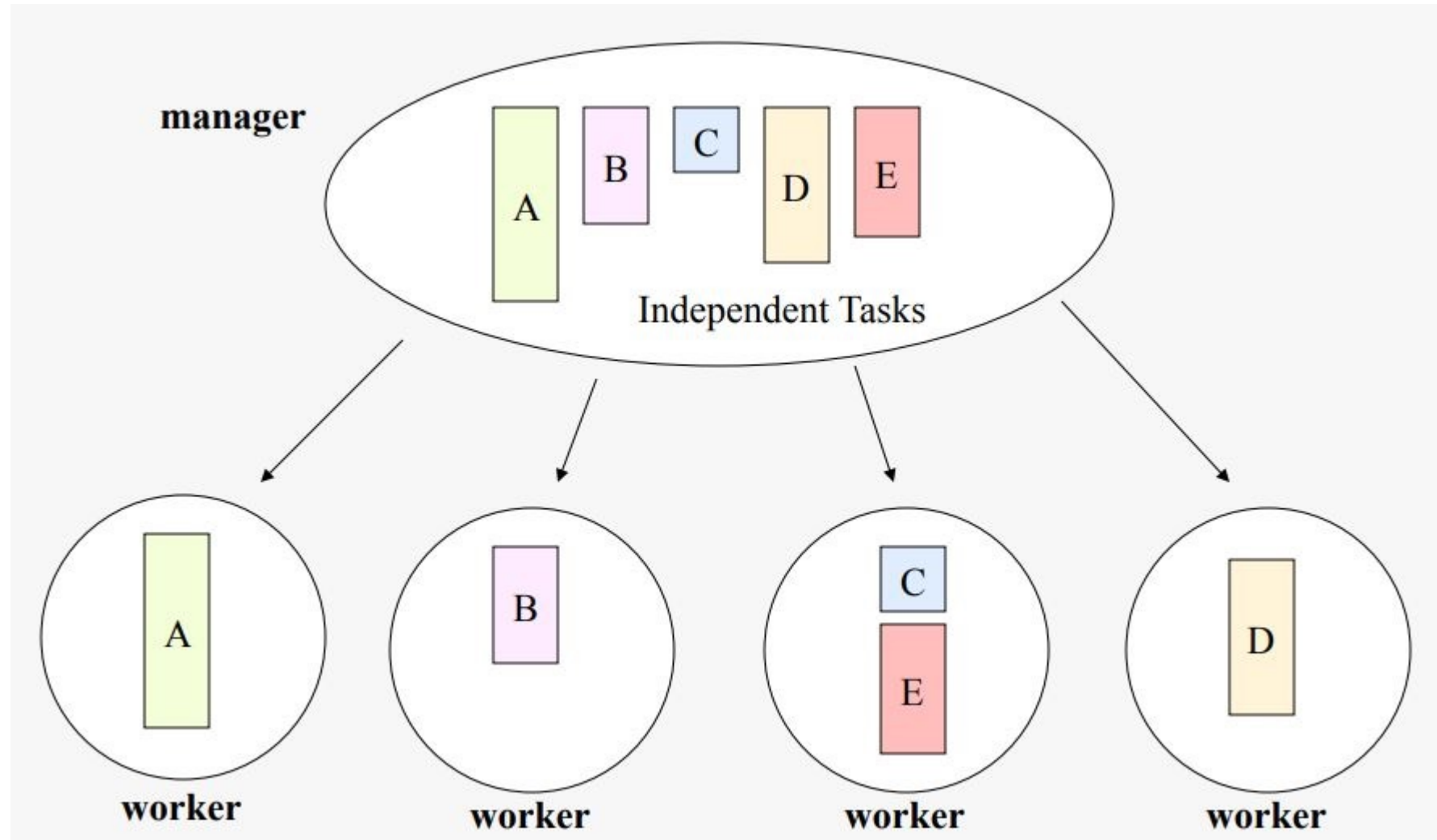


3 - Patrones programación paralela

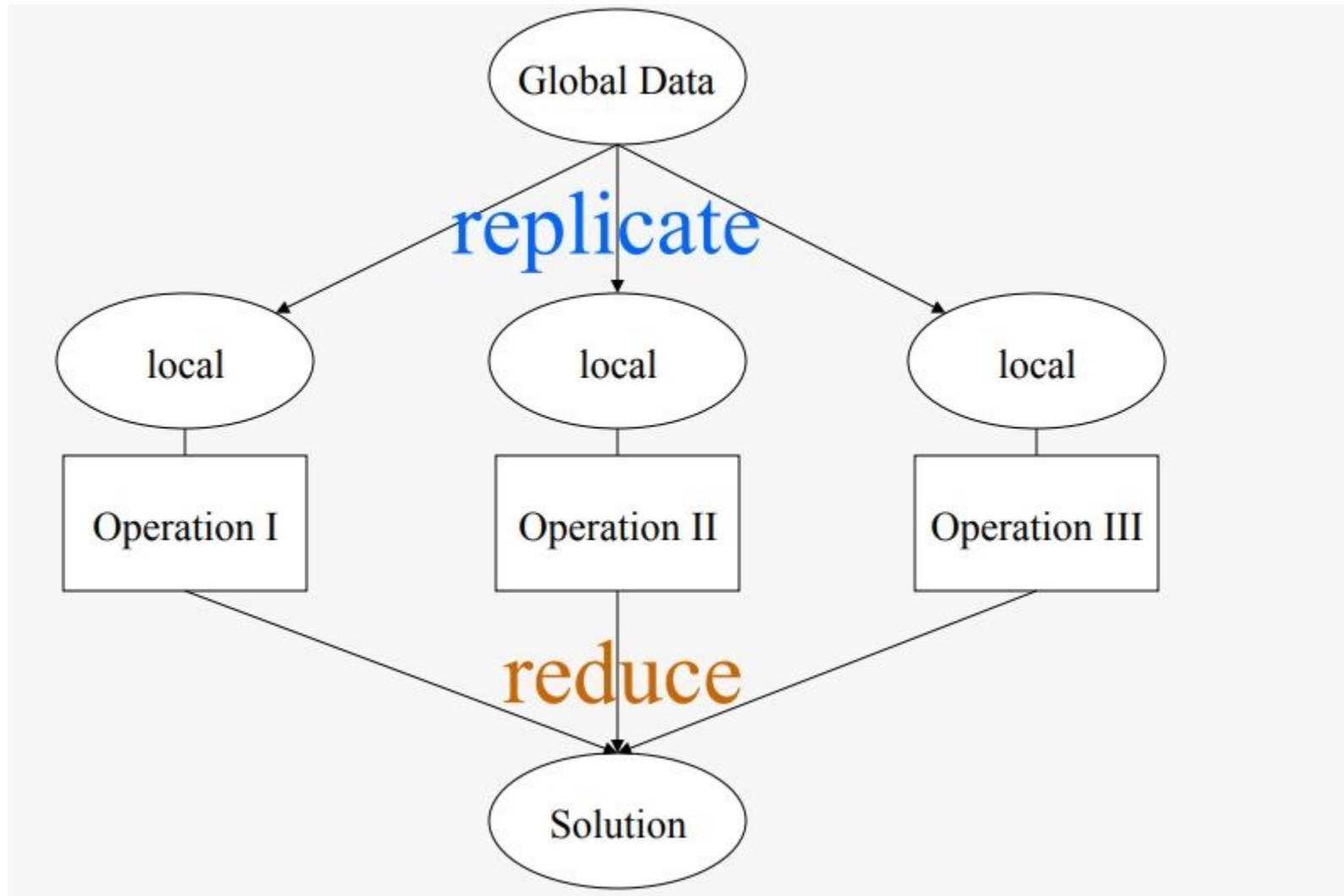
Patrones programación paralela

- Embarrassingly Parallel
- Replicable (reduce)
- Divide & Conquer
- Recursive Data

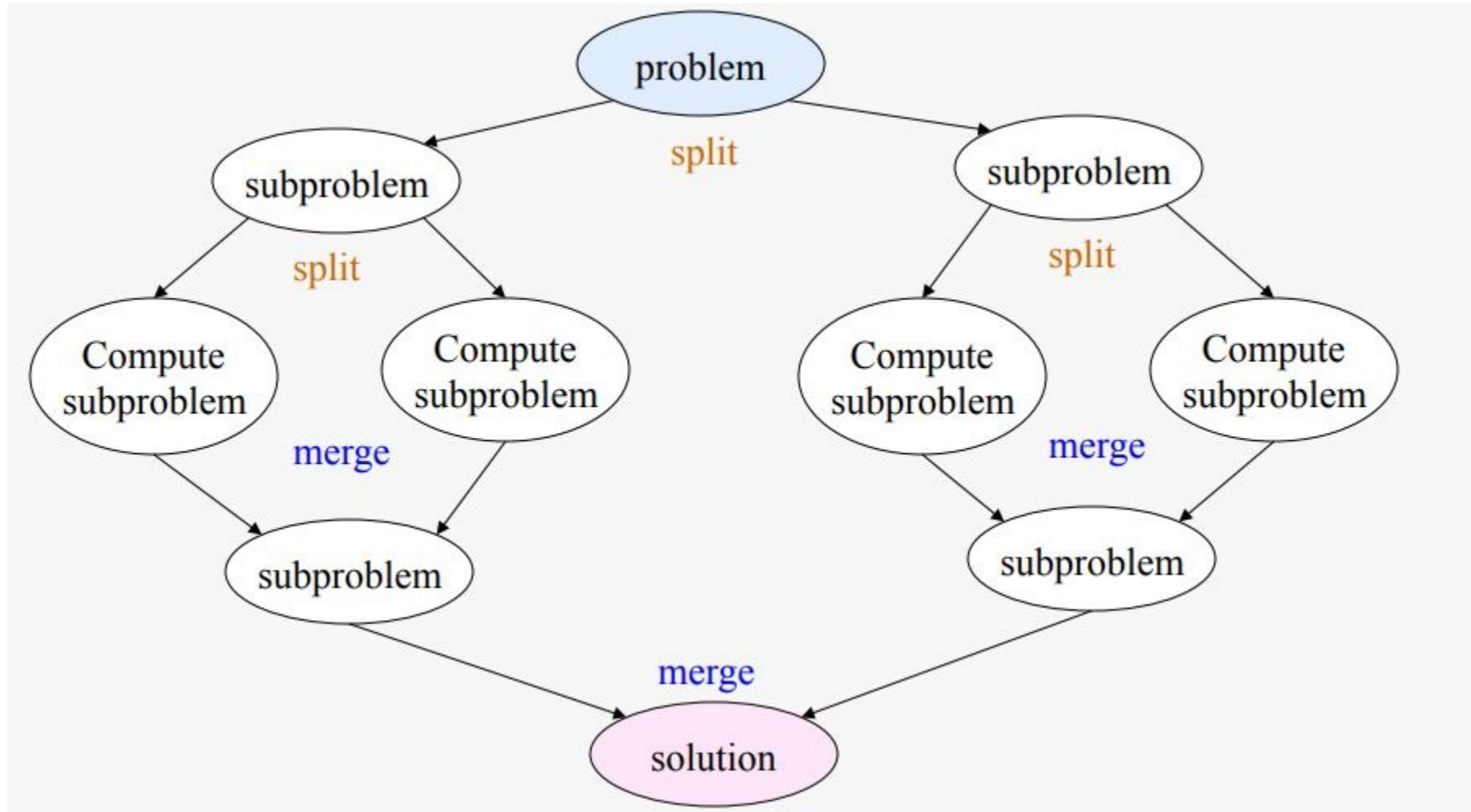
Embarrassingly Parallel



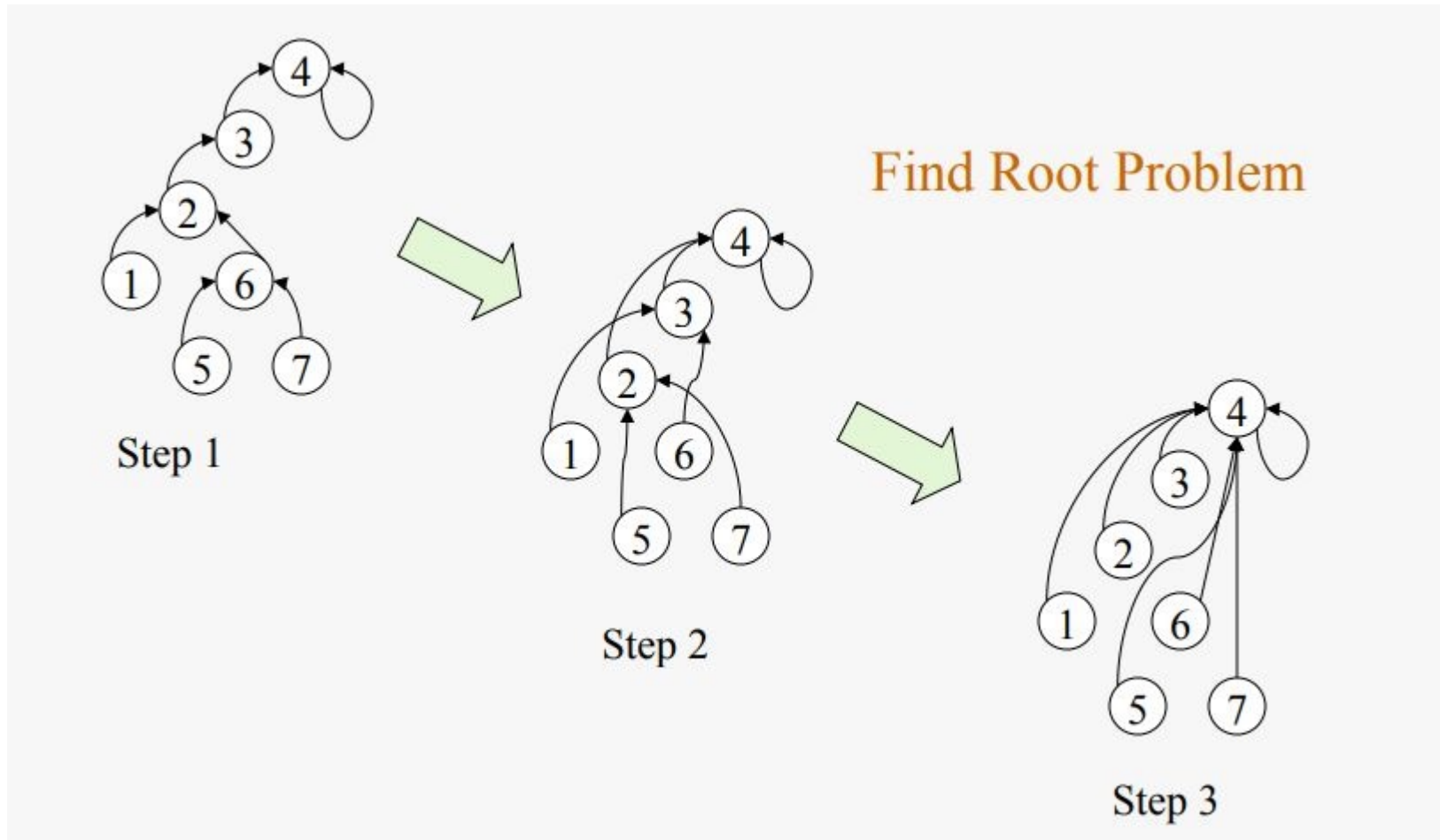
Replicable (reduce)



Divide & Conquer



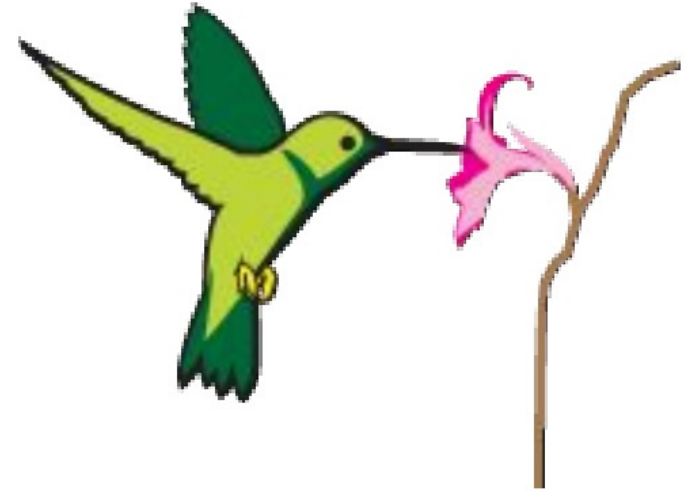
Recursive Data



4 - ¿Con qué se programa en paralelo?

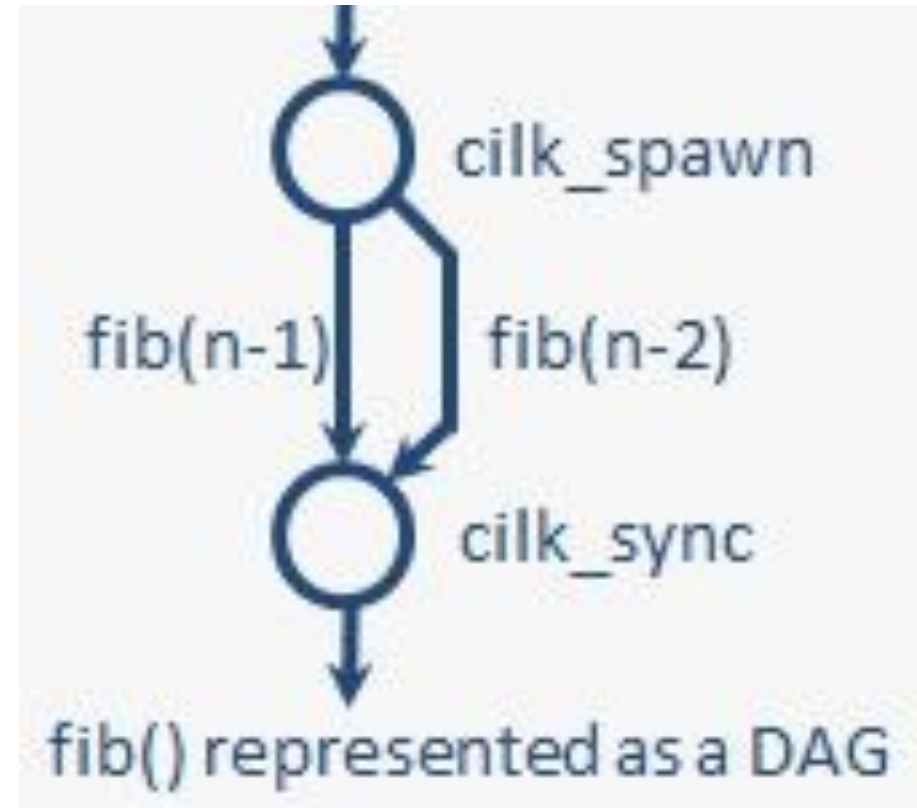
Multicore: Cilk Plus

- Extensión de C y C++ para soportar programación paralela.
- <https://www.cilkplus.org/cilk-plus-tutorial>



Keywords importantes

- `parallel_for`
- `spawn`
- `sync`

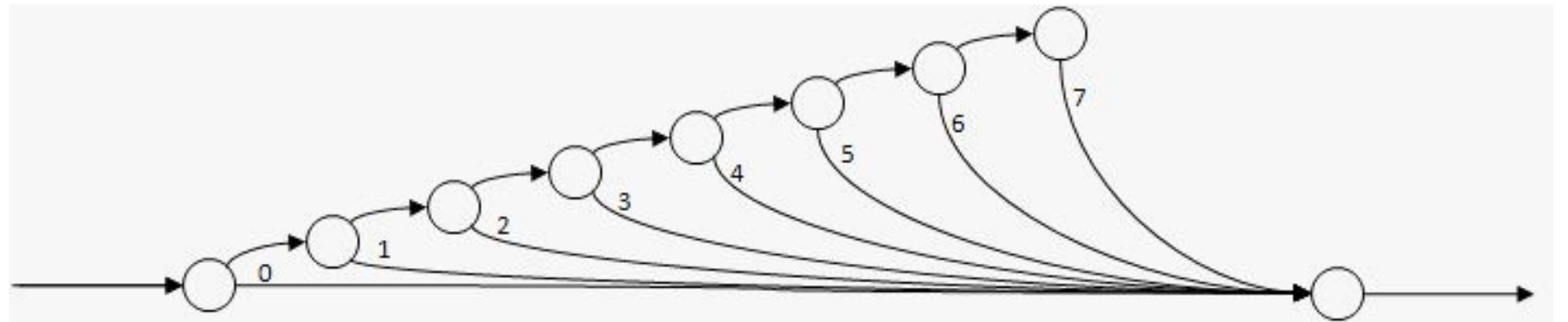


¿Cómo paralelizamos algo simple?

```
for (int i = 0; i < 8; ++i) {  
    do_work(i);  
}
```

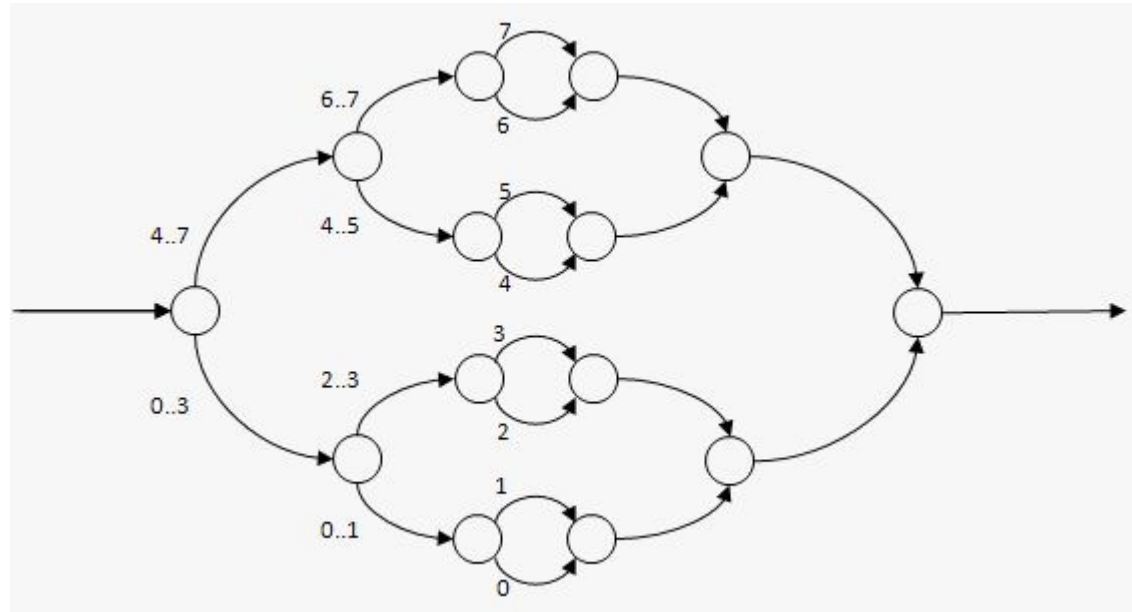
¿Cómo paralelizamos algo simple?

```
for (int i = 0; i < 8; ++i) {  
    cilk_spawn do_work(i);  
}  
cilk_sync;
```



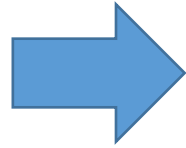
¿Cómo paralelizamos algo simple?

```
cilk_for for (int i = 0; i < 8; ++i)
{
    do_work(i);
}
```



Fibonacci

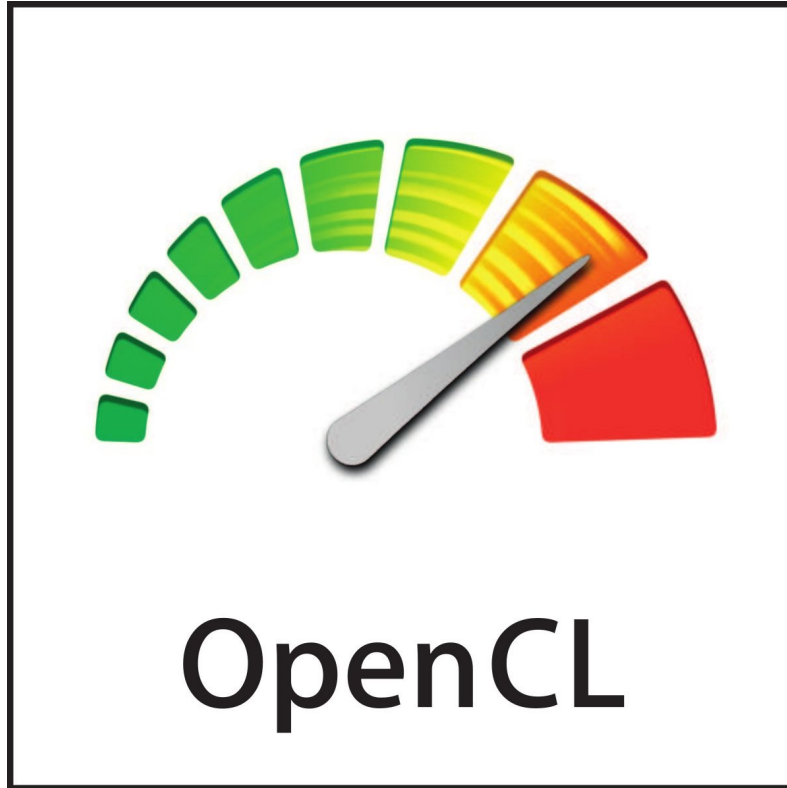
```
int fib(int n) {  
    if (n < 2)  
        return n;  
    int x = fib(n-1);  
    int y = fib(n-2);  
    return x + y;  
}
```



```
int fib(int n) {  
    if (n < 2)  
        return n;  
    int x = cilk_spawn fib(n-1);  
    int y = cilk_spawn fib(n-2);  
    cilk_sync;  
    return x + y;  
}
```

5 - Programación en GPU

GPU: OpenCL y CUDA



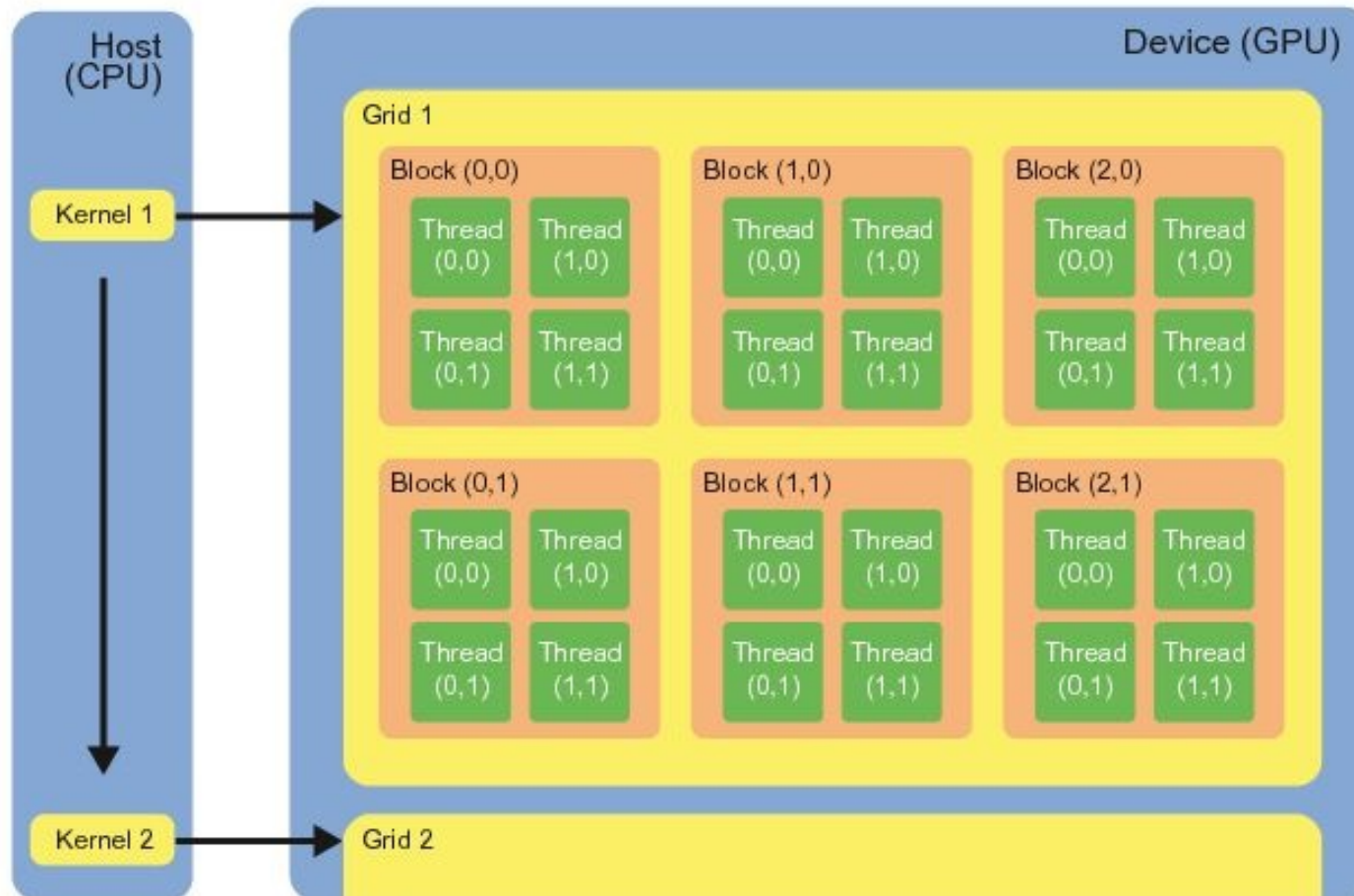
<https://www.khronos.org/opencl/resources>

<https://devblogs.nvidia.com/even-easier-introduction-cuda/>

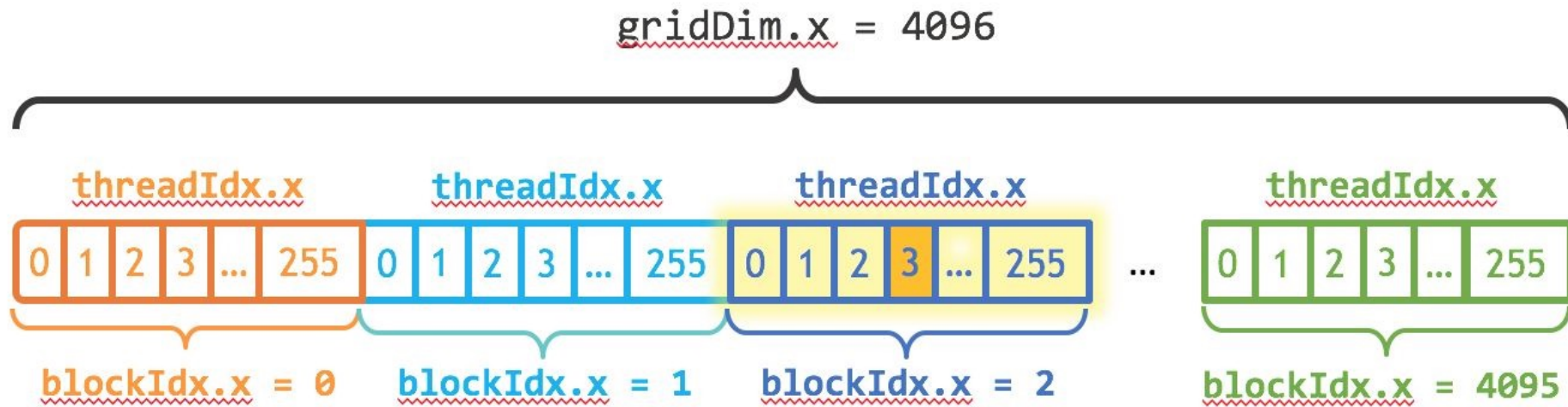
Keywords importantes

- Kernel
- Thread
- Block

Threads & Blocks



Threads & Blocks



$$\text{index} = \text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x}$$

$$\text{index} = (2) * (256) + (3) = 515$$

Single Precision $\alpha X + Y$

$$\mathbf{z} = \alpha \mathbf{x} + \mathbf{y}$$

$\mathbf{x}, \mathbf{y}, \mathbf{z} : \text{vector}$

$\alpha : \text{scalar}$

Ejemplo CUDA

CUDA C



Standard C Code

```
void saxpy_serial(int n,
                  float a,
                  float *x,
                  float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

Parallel C Code

```
__global__
void saxpy_parallel(int n,
                    float a,
                    float *x,
                    float *y)
{
    int i = blockIdx.x*blockDim.x +
            threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096, 256>>>>(n, 2.0, x, y);
```

<http://developer.nvidia.com/cuda-toolkit>

Ejemplo OpenCL

```
__kernel void SAXPY (__global float* x, __global float* y, float a) {  
    const int i = get_global_id (0);  
    y [i] = a * x [i] + y [i];  
}
```