# Algoritmos Paralelos

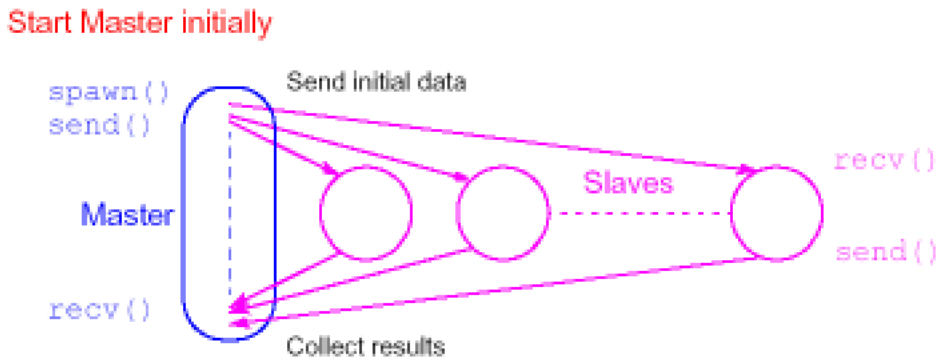
(15.09.15)

Prof. J.Fiestas

# Paralelismo directo:

Problemas, que pueden ser separados en tareas independientes relativamente fácil, tal que cada tarea sea realizada por un proceso

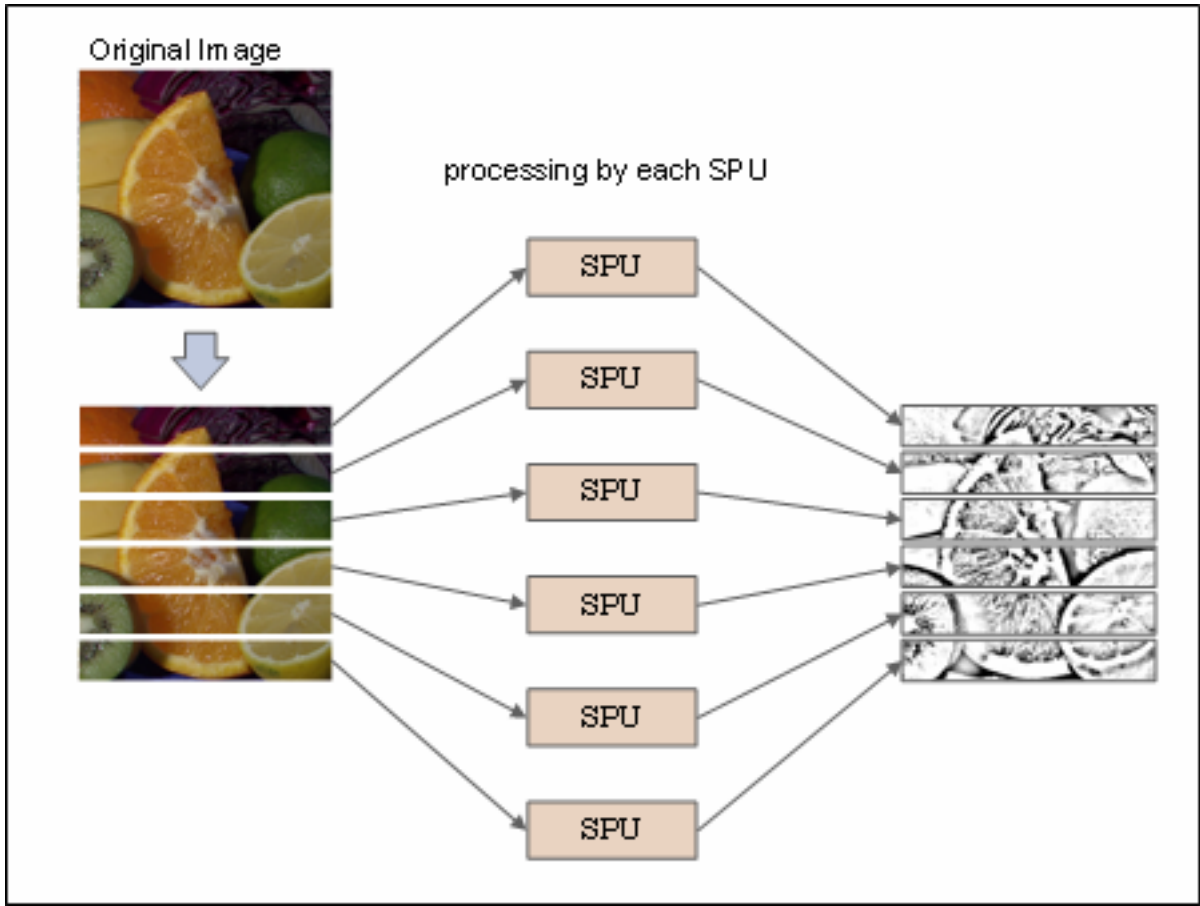
**Paralelismo ideal:**
Donde las tareas son completamente separables. Incluso no es necesaria la comunicación entre procesos



# Paralelismo ideal:

## Ejemplo: procesamiento de imágenes

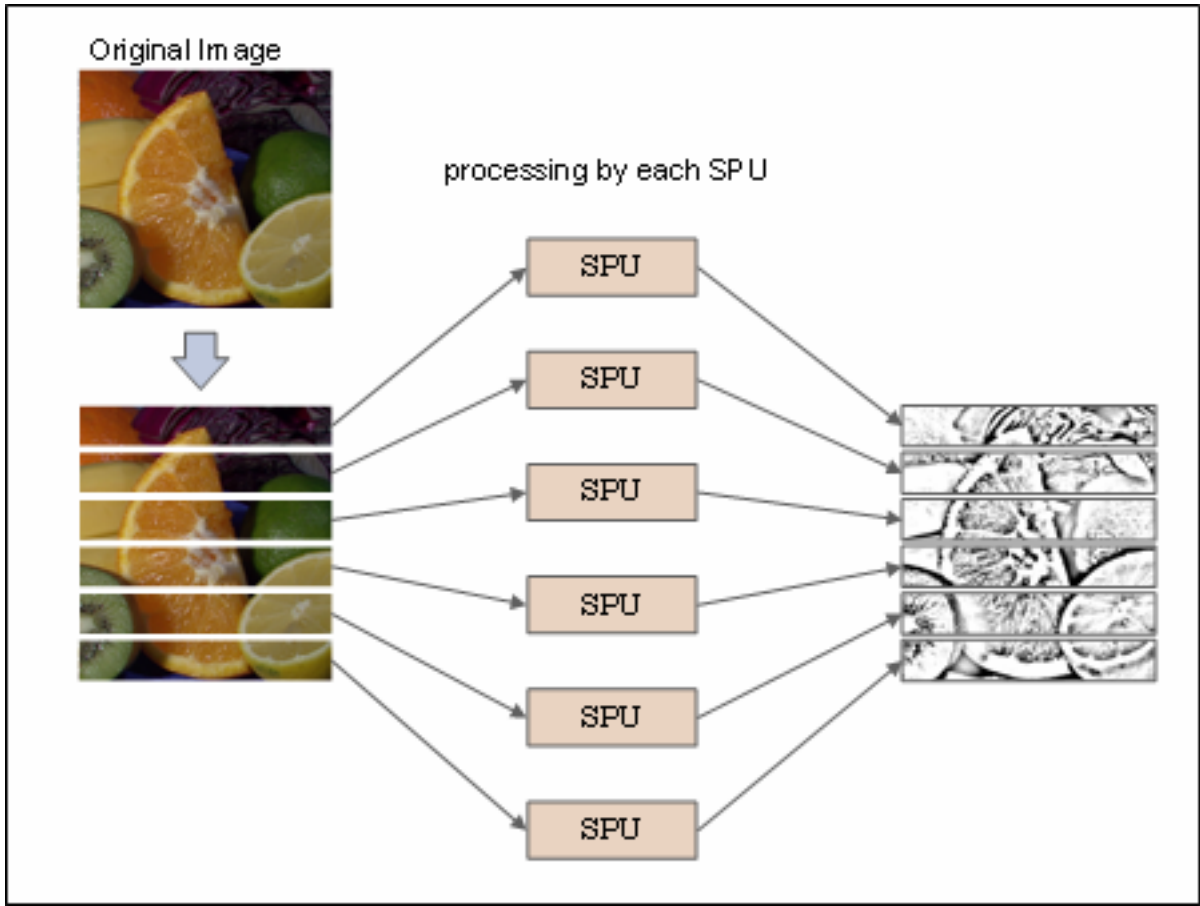
Solo las coordenadas del pixel son modificadas, independiente-mente de los pixels vecinos (desplazamiento, escala, rotación, etc)



# Paralelismo ideal:

## Ejemplo: procesamiento de imágenes

Se toma especial cuidado en la distribución de la imagen entre procesos, normalmente se separa la imagen en regiones cuadradas, o en lineas y columnas.



# Paralelismo ideal:

## Ejemplo: procesamiento de imágenes

Si se necesitan dos pasos de transformación para cada pixel, $(x' = x + \Delta x; y' = y + \Delta y)$

Tendremos una secuencia de cálculos de nxn Pixels en tiempo $ts=2n^2$, es decir $O(n^2)$

# Paralelismo ideal:

**Ejemplo: procesamiento de imágenes**

Para p procesos, cada uno con 4 resultados (antiguas y nuevas coordenadas), es el tiempo de comunicación:

$$t_{comm} = p(t_{startup} + t_{data}) + n^2(t_{startup} + 4t_{data}) = O(p + n^2)$$

Y el tiempo de cómputo:

$$t_{comp} = \frac{2n^2}{p} = O(\frac{n^2}{p})$$

Por consiguiente:

$$t_p = t_{comm} + t_{comp} = O(n^2)$$

# Paralelismo ideal:

**Ejemplo: procesamiento de imágenes**

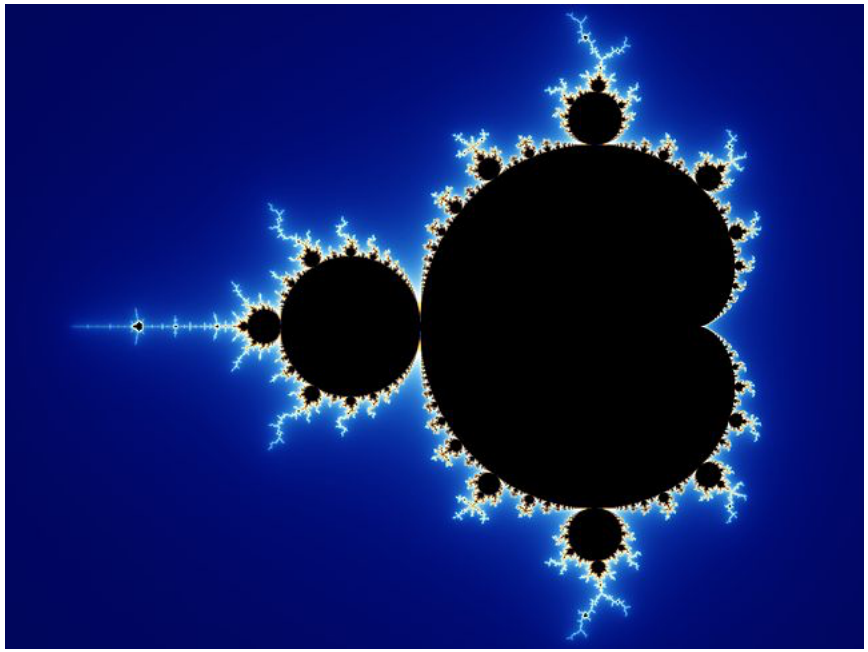
Asimismo, el ratio cómputo/communicación es constante $O(\frac{n^2 / p}{p + n^2})$, si p es constante

Ya que la comunicación a los procesos y el retorno es lo que más cuesta, arquitectura de memoria compartida es óptima para este tipo de problemas.

# Ejemplo: Mandelbrot set

Es un set de puntos complejos, que se calculan a través de la iteración de una función compleja de forma

$$z_{k+1} = z_k^{\,2} + c$$



# Ejemplo: Mandelbrot set

donde zk+1 es la iteración (k+1) del número complejo

$$z = a + bi \ \ (\text{mit } i = \sqrt{-1})$$

asi como c es un número complejo.
Se itera hasta que la longitud del vector

$$z_{len} = \sqrt{z_{real}^{\ 2} + z_{imag}^{\ 2}}$$

es mayor que 2

# Ejemplo: Mandelbrot set

## Paralelismo:

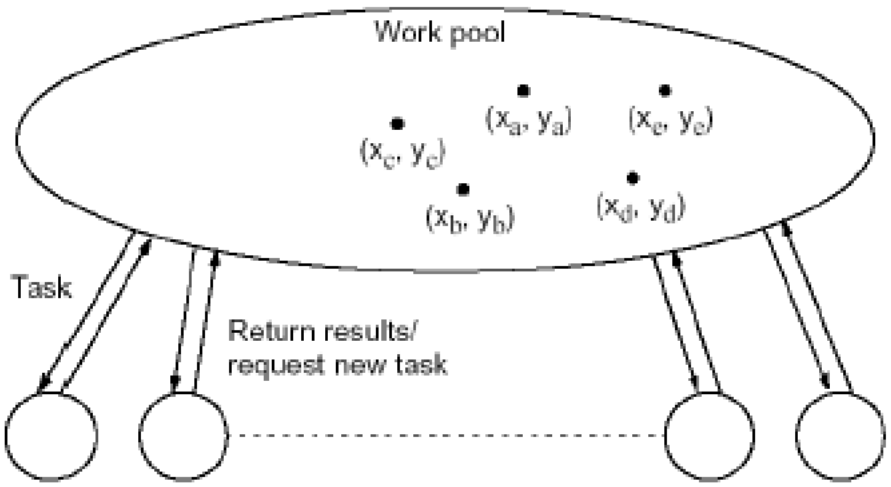
Cada pixel puede ser calculado independientemente del resto (el cálculo individual del pixel no es paralelizable)

## Distribución estática de tareas:

Cada pixel necesita un número distinto de iteraciones, por lo que cada proceso recibe una porción de la imágen, y no terminan sincronizadas entre ellas, lo que torne ineficiente al algoritmo.

# Ejemplo: Mandelbrot set
# Paralelismo:
## Distribución dinámica de tareas:

Las tareas en paralelo se distribuyen en un área común (work pool), desde donde cada proceso obtiene una nueva tarea en caso quede inactivo



# Ejemplo: Mandelbrot set

## Paralelismo:

Dados n Pixel y un número máximo de iteraciones que depende de c y por ello el tiempo es O(n)

$$t_S \leq \max\_iter \times n$$

Existen 3 pasos:

- Transferencia de datos a p-1 procesos

$$t_{comm\_1} = (p-1)(t_{startup} + t_{data})$$

# Ejemplo: Mandelbrot set

# Paralelismo:

Existen 3 pasos:

- Cálculo en paralelo $t_{comp} \leq \frac{\max\_iter \times n}{p-1}$
- Transferencia de resultados al master

  $t_{comm\_2} = k$ constante

En total: $t_p = t_{comp} + t_{comm\_1} + t_{comm\_2}$

Y la eficiencia se acerca a *p* si *max_iter* es grande. El ratio cálculo/comunicación es O(n), si p es constante.

# Ejemplo: Método Monte Carlo

Utiliza principios de cálculo probabilístico y estadística, para aproximar soluciones de problemas complejos.

Por ello, los resultados son correctos con cierta probabilidad. Se utilizan:

- Confiabilidad de sistemas técnicos
- Operatividad (almacén, transporte)
- Estudio de sismos o fenómenos naturales
- Finanzas

# Ejemplo: Método Monte Carlo

## Paralelismo:

Dada una integral $I = \int_{x1}^{x2} (x^2 - 3x)dx$

Esta puede calcularse a través de la generación de N valores aleatorios entre $x_1$ y $x_2$, y ser aproximados con $I_{MC} = \sum_{N} (x^2 - 3x)$

Ya que las iteraciones para suma o integrales son independientes, se paralelizan fácilmente asignando a cada proceso un conjunto de números random entre $x_1$ y $x_2$

# Ejemplo: Método Monte Carlo

## Paralelismo:

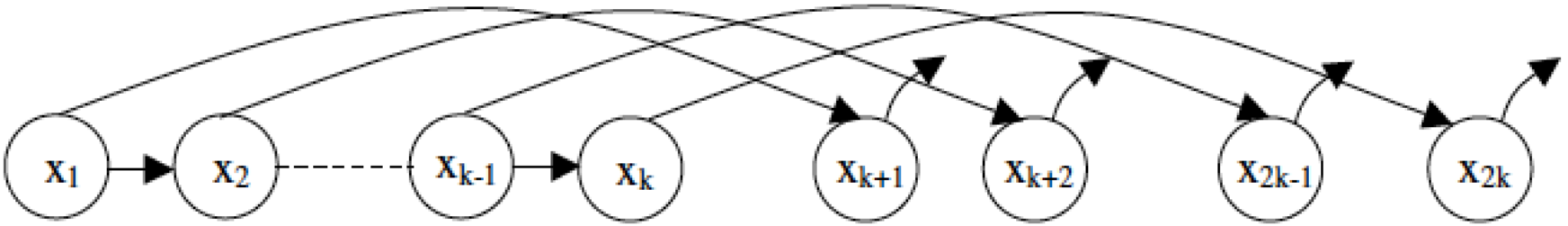
Estas se ejecutarán en cada proceso hasta que el criterio de aproximación se cumpla.
Luego se recopila y suma la información en el Master.
La generación aleatoria puede delegarse a cada proceso, evitando asi tiempos de comunicación.

# Ejemplo: Método Monte Carlo

## Generación random en paralelo:

El generador 'linear-congruente' produce números pseudo-aleatorios, utilizando la función, $x_{i+1} = (ax_i + c) \bmod m$

Con a,c y m constantes, e.g. A=16807, $m=2^{31}-1$, c=0

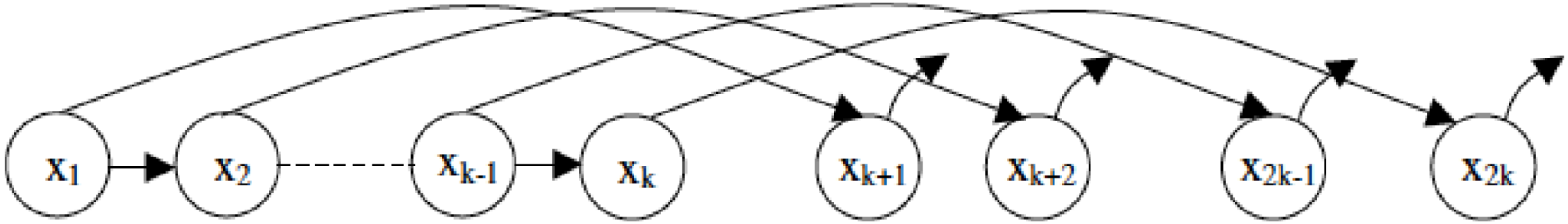


# Ejemplo: Método Monte Carlo

## Generación random en paralelo:

Se generan los primeros k valores en k procesos, que se utilizan para los siguientes k valores.

Se logra una simplificación al ser m una potencia de 2



# Usos de random en paralelo:

## Sampling:

Seleccionar una muestra representativa de un set de elementos y luego aplicarla al set original. Se usa en gemometría, grafos, y algoritmos de string matching.

## Usos de random en paralelo:

## Symmetry breaking:

Por ejemplo, seleccionando un número elevado de vértices independientes en un grafo. Si uno decide unirse al set, el resto no deberá unirse. Es dificil de paralelizar, si la estructura de cada vértice es similar. El problema se resuelve usando randomización para romper la simetría entre los vértices.

# Usos de random en paralelo:

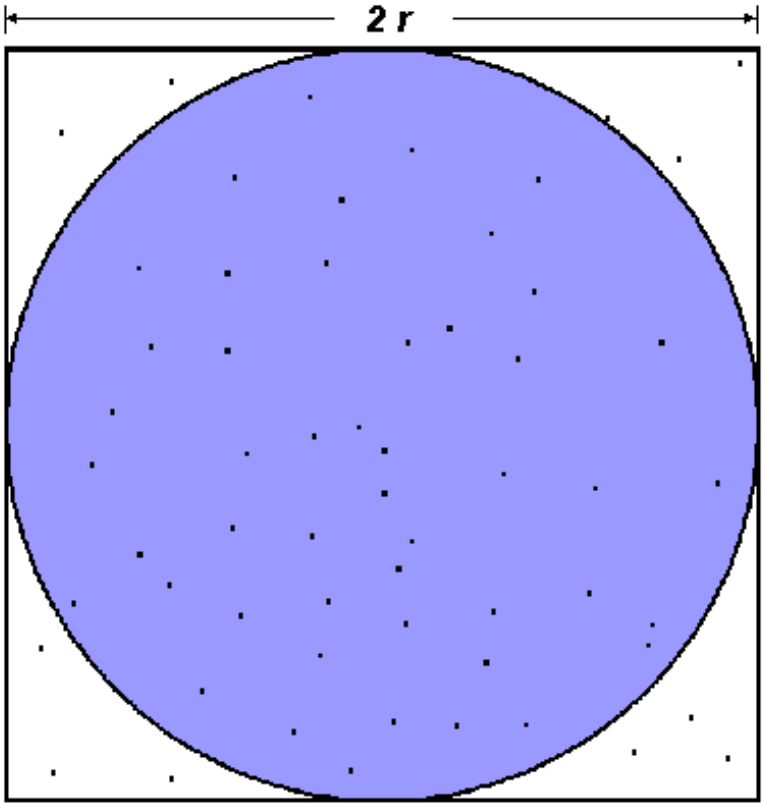
# Load balancing:

Una forma de particionar un número grande de elementos en una colección más homogénea, es asignando aleatoriamente cada elemento a un subgrupo

# Ejemplo:

## Método de Monte Carlo para calcular PI

- Genera N puntos random en el cuadrado exterior
- Determina el número de puntos n dentro del círculo




$$A_S = (2r)^2 = 4r^2$$

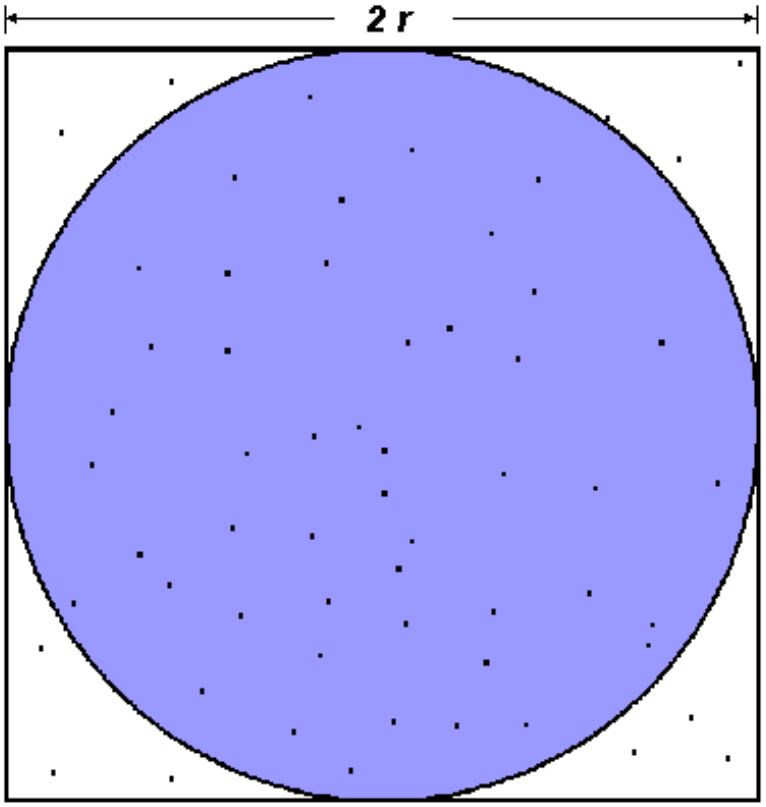
$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

# Ejemplo:

## Método de Monte Carlo para calcular PI

- Siendo el área del círculo pi*r^2, y el área del cuadrado 4*r^2
- Entonces n/N = pi/4, o pi=4*n/N
- Generando un número suficiente de puntos se logra un resultado mas exacto




$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

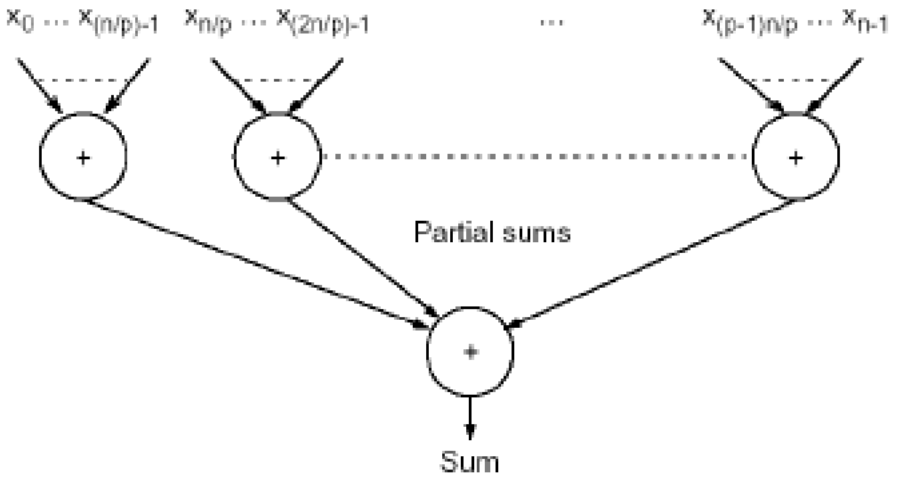
$$\pi = 4 \times \frac{A_C}{A_S}$$

# Particionamiento:

Las tareas se dividen y se ejecutan independientemente. Esto se aplica:

- A la data (data partitioning, domain decomposition)
- A las funciones del programa (functional decomposition)

Aqui también la comunicación es lo mas costoso



# Particionamiento:

El cálculo secuencial necesita n-1 adiciones, con complejidad O(n). Con p procesos se obtiene:

- Envío de data $t_{comm\_1} = p(t_{startup} + (n/p)t_{data})$
- Calculo de cada proceso $t_{comp\_1} = n/p - 1,$
- Recopilación de sumas parciales

$$t_{comm\_2} = t_{startup} + pt_{data},$$

- Suma total $t_{comp\_2} = p - 1$

Complejidad es $t_p = O(n)$

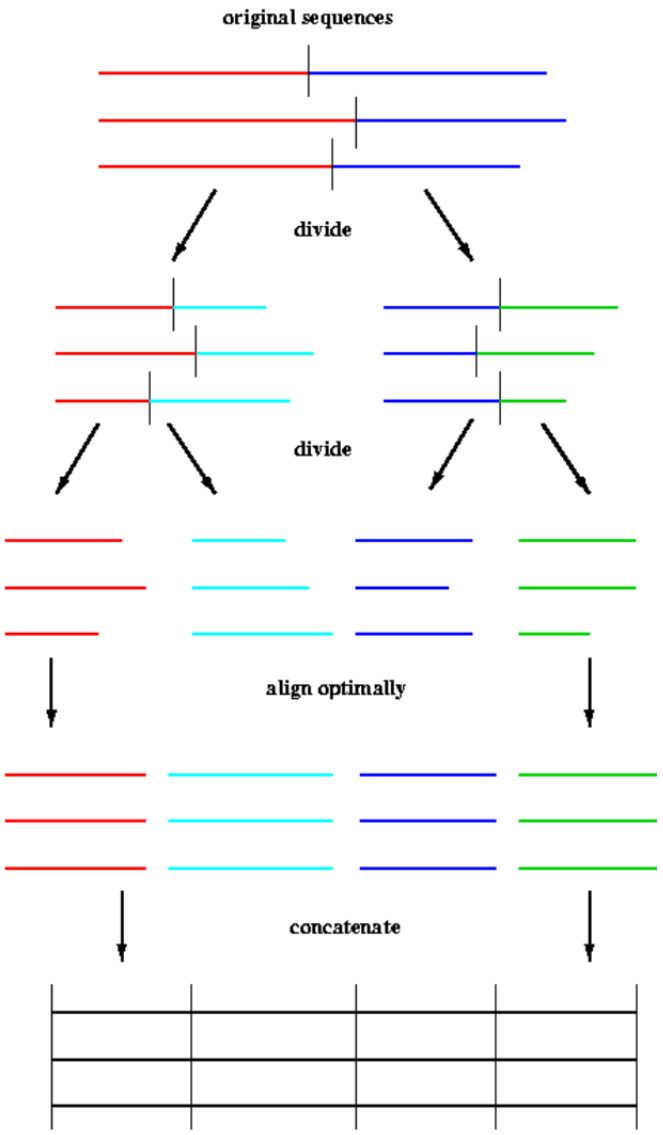
# **Paradigma Divide y Vencerás** (divide and conquer)

Algoritmo heuristico acelerado para solución de secuencias múltiples y homologas. Funciona:

- Separando las secuencias en partes reduciendo su longitud. Esto es, dividiendo el problema en sub-problemas

# Paradigma Divide y Vencerás (divide and conquer)

- Optimizar su distribución o resolver los sub-problemas (recursiva, directamente)
- Concatenar resultados o combinar soluciones de subproblemas en solución general



# **Paradigma Divide y Vencerás** (divide and conquer)

Se utiliza para operaciones globales en listas (ordenamiento), cálculo de máximo/mínimo, o búsqueda

En su versión recursiva, se generan dos sub-problemas en cada separación. Esto se representa con un árbol, que se divide para abajo y se recopila para arriba.

# **Paradigma Divide y Vencerás** (divide and conquer)

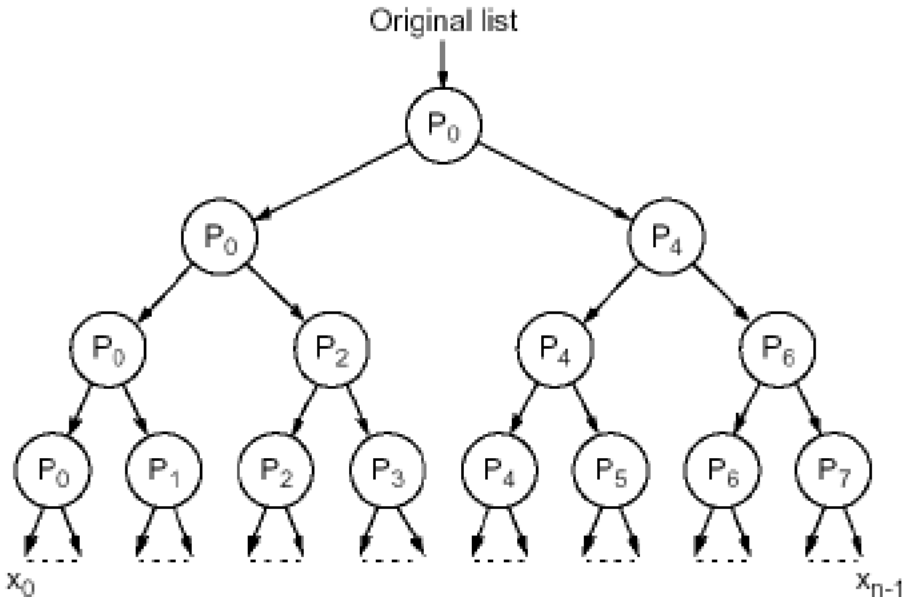
## **Paralelismo:**

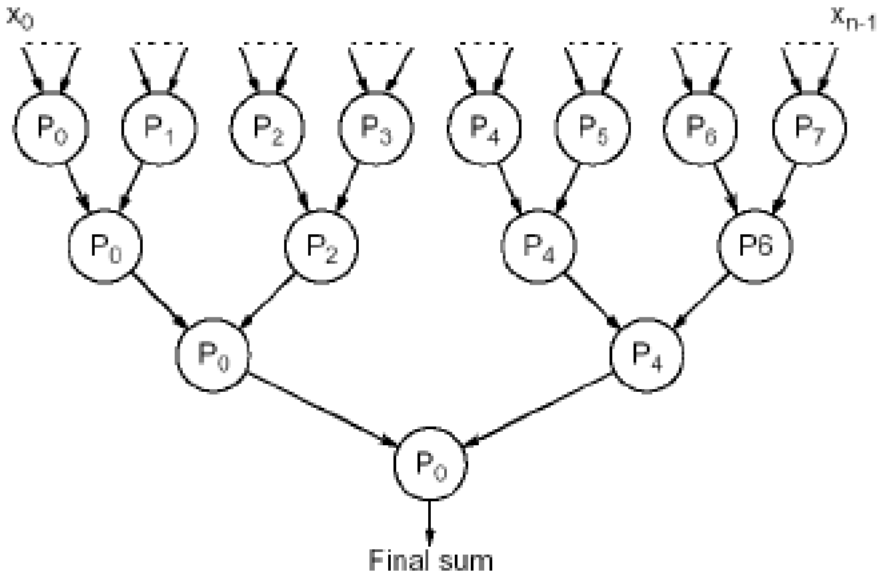
Partes distintas del árbol pueden ser ejecutadas a la vez. Una solución sería asignar un proceso a cada nodo del árbol. Pero sería ineficiente, ya que para $2^m$ sub-problemas se necesitan $2^{m+1}$-1 procesos

# Paradigma Divide y Vencerás

## Paralelismo:

División y conquista en paralelo



# Paradigma Divide y Vencerás

## Paralelismo:

Sea la data n una potencia de 2 y p procesos, para la primera etapa (división) se necesitan log(p) pasos

$$t_{comm\_1} = \frac{n}{2}t_{data} + \frac{n}{4}t_{data} + \ldots + \frac{n}{p}t_{data} = \frac{n(p-1)}{p}t_{data}$$

Mientras que en la combinación de resultados

$$t_{comm\_2} = (\log p)t_{data}$$

Es decir, $t_{comm} = O(n)$

# Paradigma Divide y Vencerás

# Paralelismo:

El cálculo suma n/p números.

$$t_{comp} = \frac{n}{p} + \log p = O(n)$$

La eficiencia máxima es p, si todos los procesos cálculan su suma parcial.

# Ejemplo: Bucket Sort

Utiliza un método de **particionamiento**.
Funciona mejor con conjuntos homogéneos.
Un intervalo a se separa en regiones m (**buckets**), con n/m valores en cada una.
Luego son los grupos en cada bucket ordenados (con quicksort o mergesort) en **O((n/m) log(n/m))** y los resultados concatenados en una lista ordenada.
Tiempo secuencial:

$$t_S = n + m(n/m \log(n/m)) = O(n \log(n/m))$$

# Ejemplo: Bucket Sort

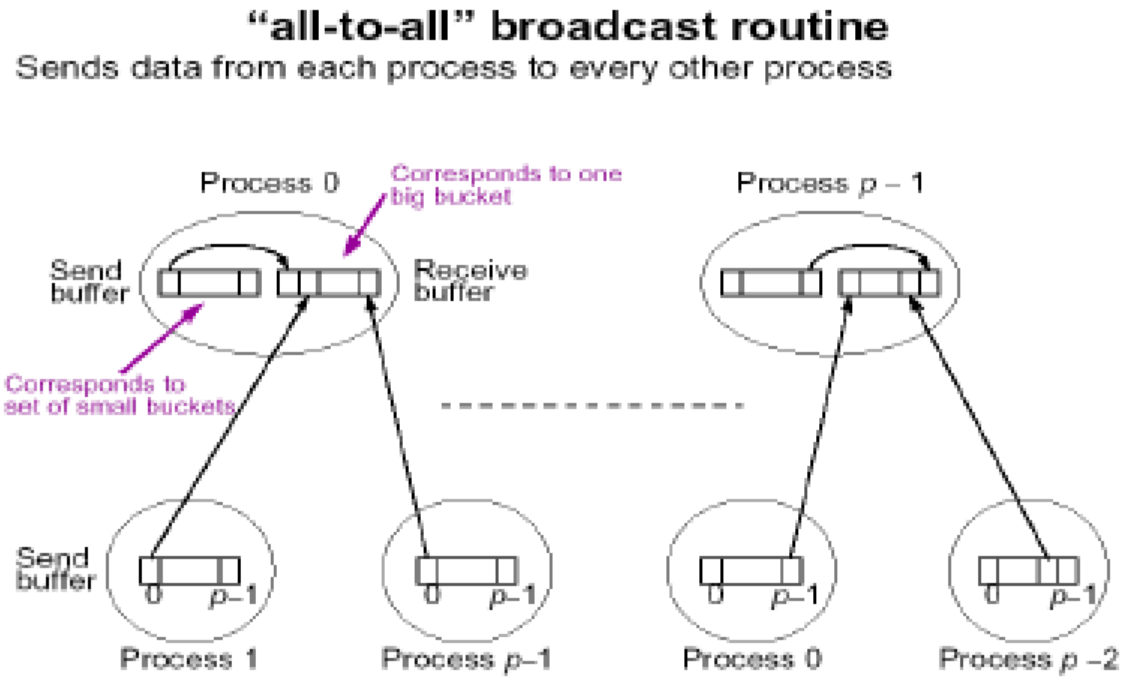
**En paralelo:**

Cada bucket pertenece a un proceso, con lo que se logra $O(n/p \log(n/p))$ con p=m procesos.
Pero cada proceso debe controlar todos los números no-ordenados para su selección.
Si se separa la lista en p regiones, cada una para cada proceso, esta se separa en cada bucket y se envía luego a cada otro proceso, para ser ordenada finalmente ahi.

# Ejemplo: Bucket Sort

## En paralelo:

Esto implica una comunicación all-to-all



# Ejemplo: Bucket Sort

**En paralelo:**

Primero se envía la data a los procesos y estos la classifican. De ahi, los pasos son:

- Comunicación, $t_{comm\_1} = t_{startup} + nt_{data}$
- 1.cálculo, n/p números en p buckets pequeños $t_{comp\_2} = n/p$
- Comunicación, p-1 buckets son enviados a p-1 procesos $t_{comm\_3} = p(p-1)(t_{startup} + (n/p^2)t_{data})$
- 2.cálculo, los n/p numeros en los buckets grandes se ordenan $t_{comp\_4} = (n/p)\log(n/p)$

# Ejemplo: Integración numérica

Para el cálculo de una integral de la forma $I = \int_a^b f(x)dx$ se puede dividir el intervalo a-b en partes, tal que cada una se aproxima con un rectángulo, asignado a cada proceso

**Método estático:**

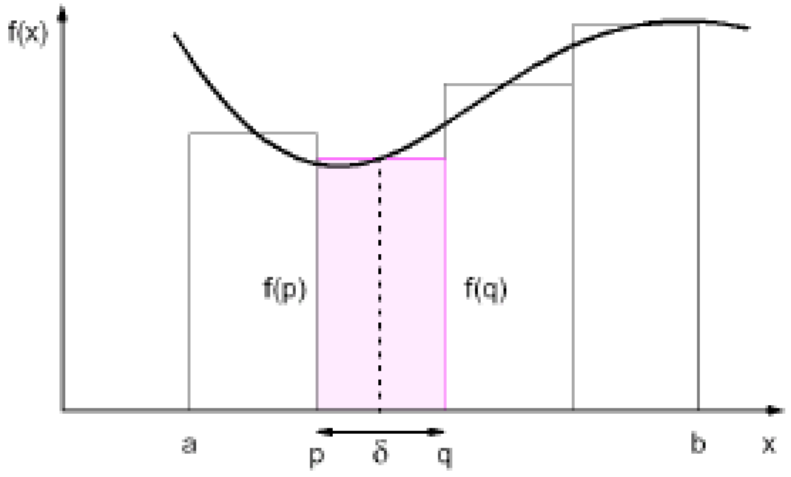
En el método del trapecio, cada área se calcula con la fórmula

$$1/2(f(p) + f(q))\delta$$

# Ejemplo: Integración numérica

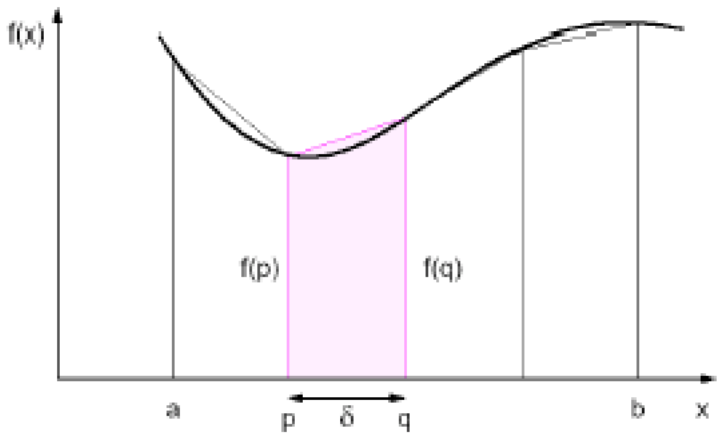
**Método estático:**

Si se conoce δ de antemano, se comunica a cada proceso. Con p procesos,  es (b-a)/p



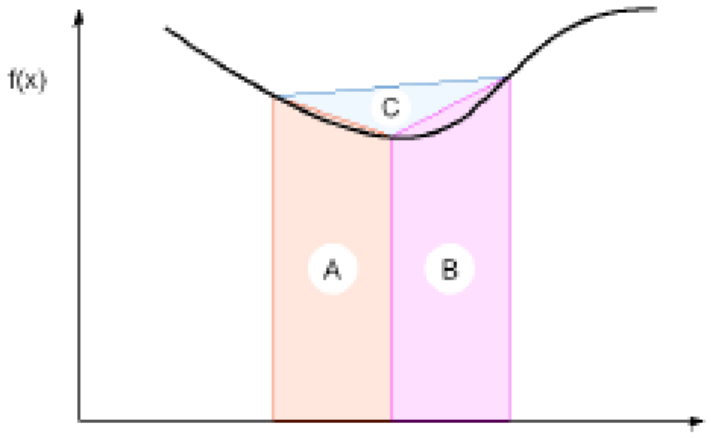
# Ejemplo: Integración numérica

**Cuadratura adaptiva:**
Si no se conoce δ de antemano, se calcula hasta que se logre la convergencia deseada



# Ejemplo: Integración numérica

**Cuadratura adaptiva:**
δ se divide en 3 zonas, y se divide hasta que C sea lo suficientemente pequeño



# Ejemplo: Problema de N-cuerpos

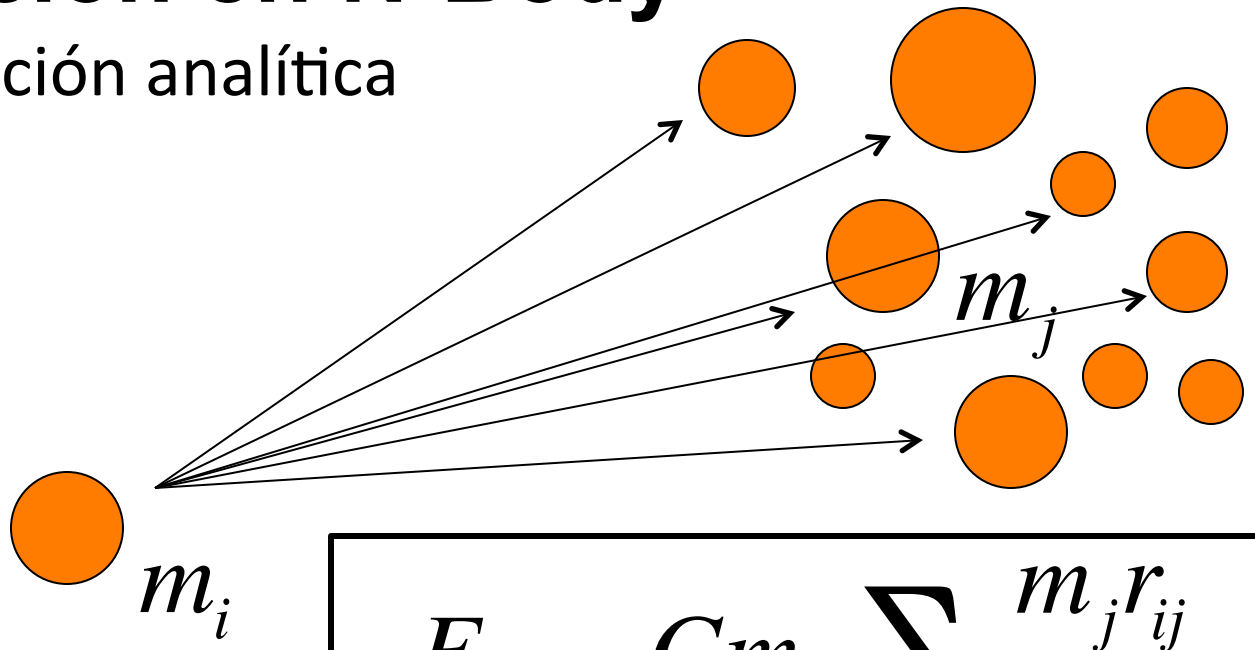
Cálculo de fuerzas entre cuerpos, aplicado a problemas de astronomía o dinámica molecular.

**Gravitación en N-Body**

Se trata de calcular las fuerzas gravitatorias de cuerpos en el espacio (planetas, estrellas, galaxias), con lo que se obtendrán posiciones y velocidades de los cuerpos en un tiempo determinado.

# Gravitación en N-Body

No hay solución analítica
desde N=3




$$F_i = -Gm_i \sum_{\substack{1 \le j \le N \\ j \neq i}} \frac{m_j r_{ij}}{\left\| r_{ij} \right\|^3}$$

Número de operaciones de cálculo de fuerza

**N(N-1)/2**

# Ejemplo: Problema de N-cuerpos

## Gravitación en N-Body

Para una simulación se puede solo aproximar el movimiento en intervalos pequeños.
Sea el intervalo Δt, es para t+1

y con ello, para la velocidad $F = \frac{m(v_{t+1} - v_t)}{\Delta t}$
y la posición:

$$v_{t+1} = v_t + \frac{F\Delta t}{m}, \qquad x_{t+1} = x_t + v\Delta t$$

# Ejemplo: Problema de N-cuerpos

## Gravitación en N-Body

Ya que el cuerpo se mueve a una nueva posición, se modifican su aceleración y fuerza, que deberá volver a calcularse.

Ya que la velocidad no es constante en Δt es solo una aproximación. Por ello se utilizan métodos de interpolación en el tiempo como 'leap-frog'

# Ejemplo: Problema de N-cuerpos secuencial

```
class Nbody
{
public:
float pos[3][N];
float vel[3][N];
Float m[N]
}
```

```
int main (int arg, char
**argv)
{
…
// define class galaxy
Nbody galaxy;
…
// initialize properties
galaxy.init();
…
// integrate forces
galaxy.integr();
return 0;
}
```

```
void integr ()
{
…
// measure CPUtime
start=clock();

force(n, pos, vel, m, dt)

// measure CPUtime
end = clock();
cpuTime= difftime(end,start)/
(CLOCKS_PER_SEC)
…
}
```

```
void force(int n, float pos[][..],
….,float vel[][..], float m[..], float
dt)
{
 // sume over i
  for (int i=0; i<n; i++)
  {
    float my_r_x = r_x[i];
 // sume over j
    for (int j=0; j<n; j++)
    {
        if(j!=i)   // avoid i=j
        {
// compute accelerations
    float d = r_x[j]-my_r_x; // 1 flop

    a_x += G*m[j]/(d*d);  // 4 flops
        …
        }
    }
```

```
//update velocities
v_x[i] += a_x*dt; // 2 flops
// update positions
r_x[i] += v_x[i]*dt; // 2 flops
  }
}
```

# Ejemplo: Problema de N-cuerpos

## En paralelo:

Se puede realizar con partición directa, donde cada proceso se encarga de un grupo de cuerpos y las fuerzas son comunciadas con los otros procesos

La complejidad es $O(n^2)$, ya que cada cuerpo es afectado por los n-1 restantes.

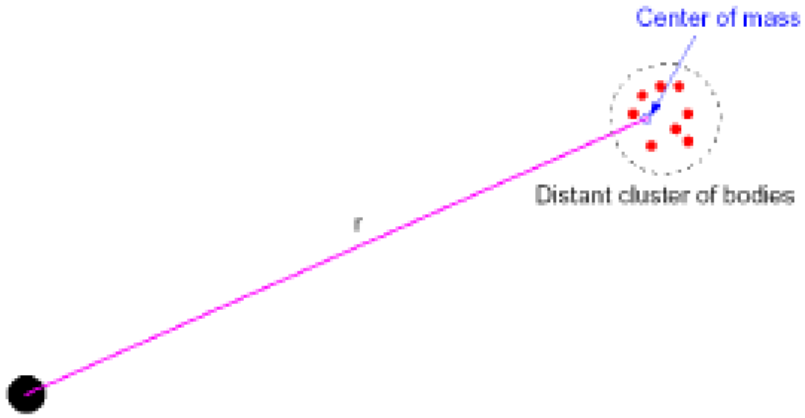
Esto no es eficiente.

# Ejemplo: Problema de N-cuerpos

## En paralelo:

Por ello, se considera que un cluster de cuerpos distantes afecta aproximandamente como un solo cuerpo.

El algoritmo de Barnes-Hut es de divide-and-conquer que funciona dividiendo el dominio en cubos (en 3D) y subdividiendolos en grupos de 8.



# Ejemplo: Fibonacci

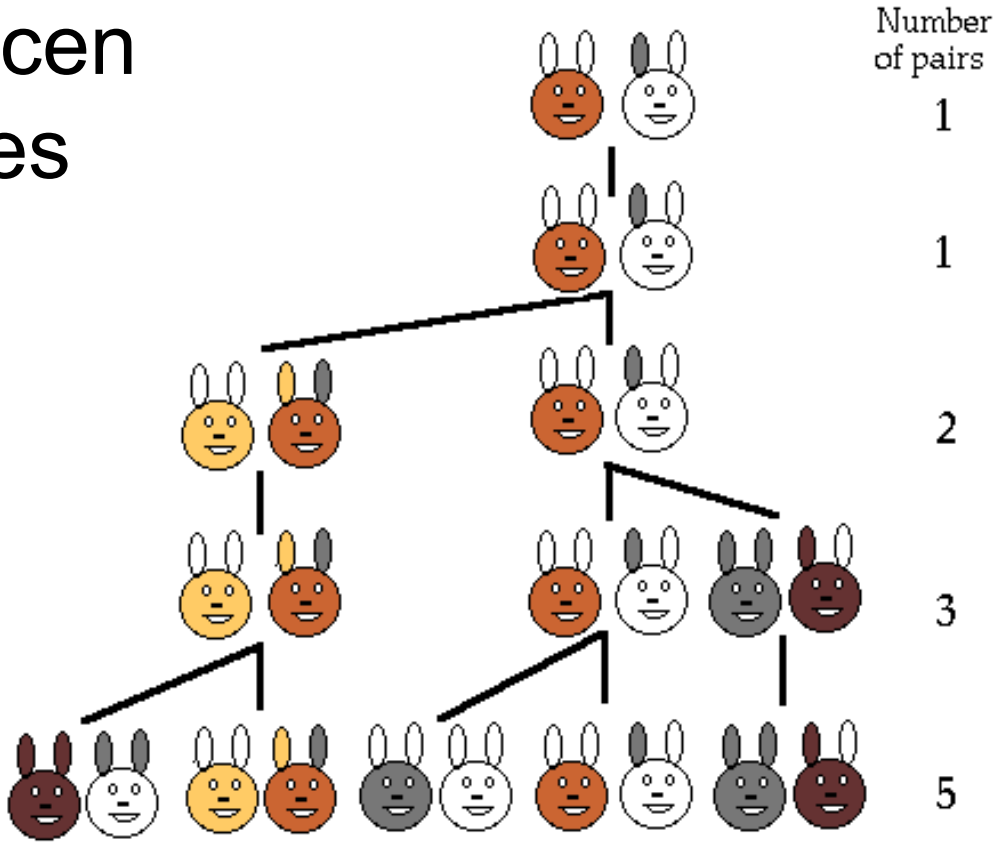
Los números Fibonacci son la secuencia: 0,1,1,2,3,5,8,13,21,34,55,89,144,....

Cada número es las suma de los previos dos números, o $F_n = F_{n-1} + F_{n-2}$, con $F_0=0$, $F_1=0$

# Ejemplo: Fibonacci

Que tan rápido conejos se reproducen en condiciones ideales? (Fibonacci, 1202)

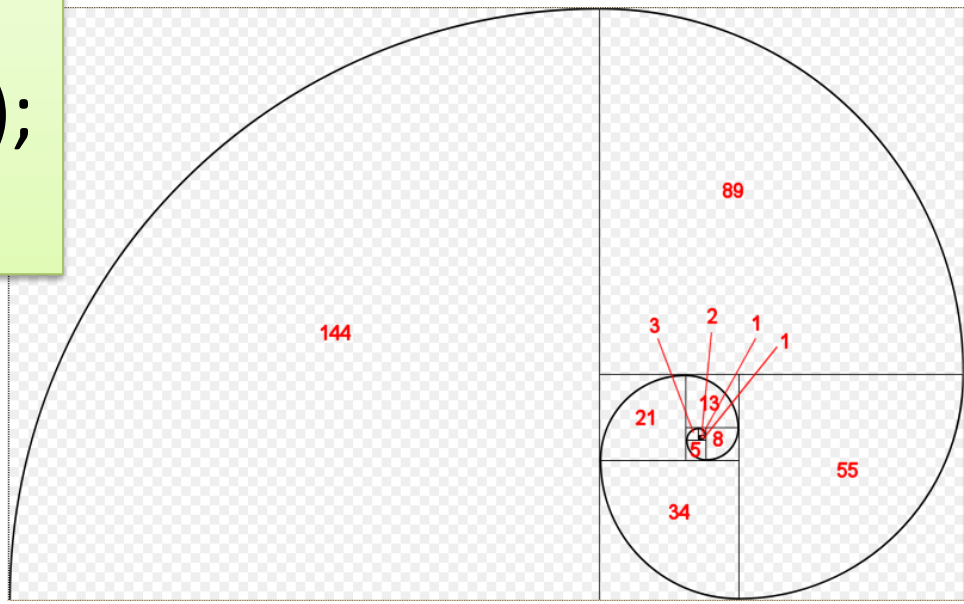
Conejos se reproducen
en parejas cada mes
y nunca mueren



# Ejemplo: Fibonacci

Se utiliza un algoritmo recursivo

```
function fib(n) {
    if (n < 2) {
        return n;
    }
    return fib(n-1) + fib(n-2);
}
```



# Ejemplo: Fibonacci

## Dynamic multithreading

```
Fibonacci(n):
1   if n < 2:                          | thread A
2     return n                         | thread A
3   x = spawn Fibonacci(n-1)           | thread A
4   y = spawn Fibonacci(n-2)           | thread B
5   sync                               | thread C
6   return x + y                       | thread C
```

Aqui, **spawn** paraleliza Fibonacci(n-1)
**sync** sincroniza para garantizar se usen los valores correctos de x,y