## Grado en Ingeniería Información

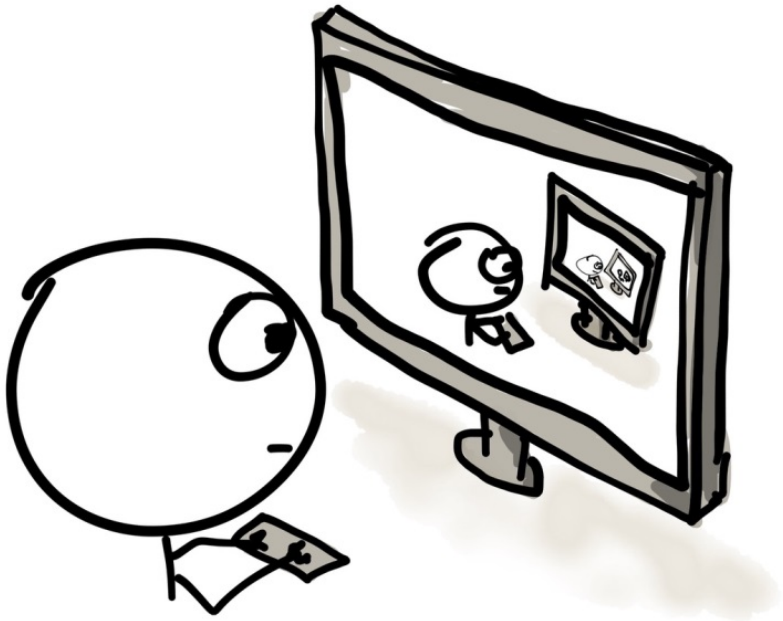
# Estructura de Datos y Algoritmos

## Sesión 2

**Curso 2023-2024**

Marta N. Gómez

UNIVERSIDAD NEBRIJA



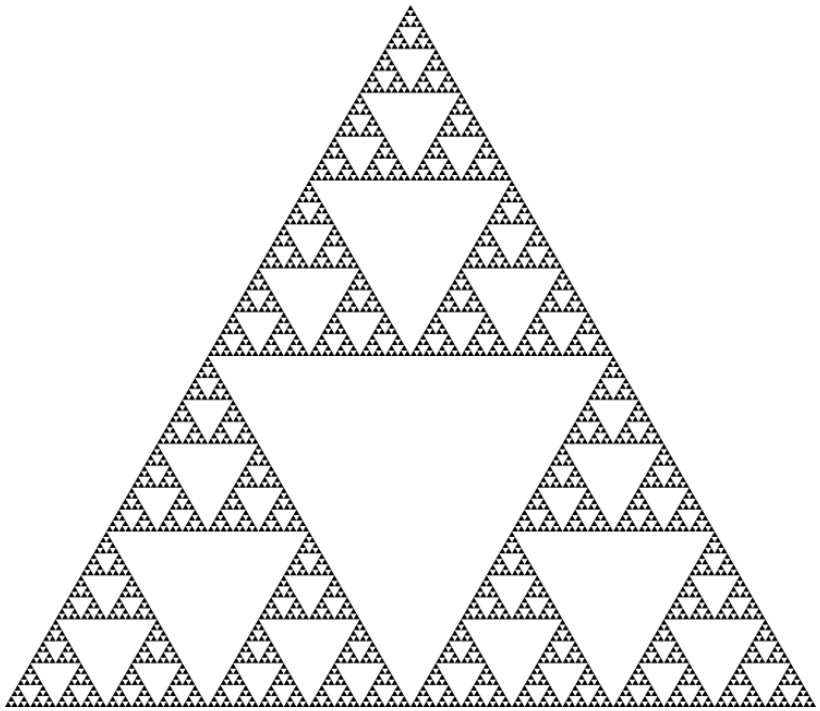
- Algoritmos
- Algoritmos iterativos
- **Algoritmos recursivos**

UNIVERSIDAD NEBRIJA

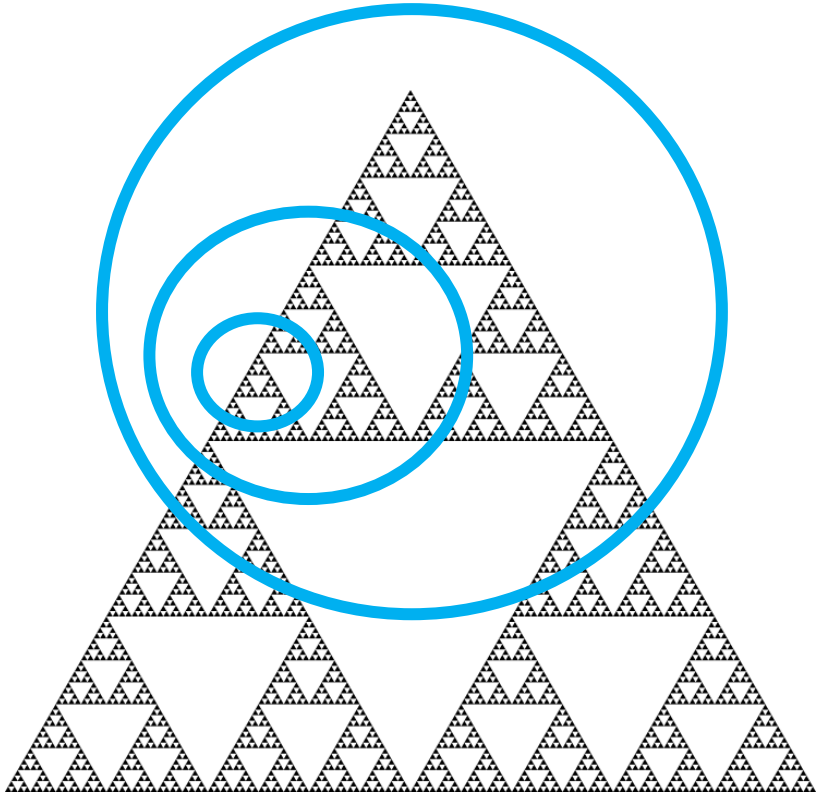
# ÍNDICE

# Definición de Recursividad

**Técnica** que sirve para **definir conceptos o diseñar procesos** incluyendo, en la **propia definición** o **diseño,** el **propio concepto** o **proceso** definido.

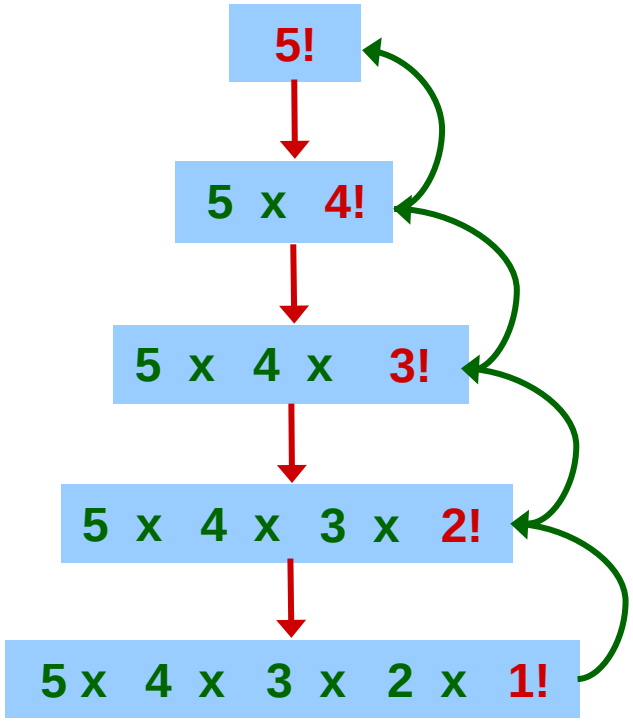
**Técnica** que sirve para **definir conceptos o diseñar procesos** incluyendo, en la **propia definición** o **diseño,** el **propio concepto** o **proceso** definido.

# Definición de Recursividad

Una **función** es **recursiva** cuando dicha función ***se llama a sí misma***.

Un **proceso** que utiliza la **recursividad** es ***sustituir*** la ***iteración*** o los ***bucles***.



# Definición de Recursividad

Una ***función recursiva*** siempre:

- Tiene una **condición de salida** de la **recursividad**, es decir, un **caso base.** El **caso base** hace que **no se produzca la llamada recursiva** y por tanto, **finalice la ejecución de la función.**
- **Modifica los parámetros** de la **llamada recursiva**, es decir, se va **aproximando** cada vez más al **caso base** (**condición de salida**) y **evita** la **recursividad infinita**.

# Tipos de Recursividad: Recursividad Lineal

**Recursividad Lineal:** produce, como **máximo**, **otra llamada recursiva**.

Según **cómo** se obtiene el **resultado**:

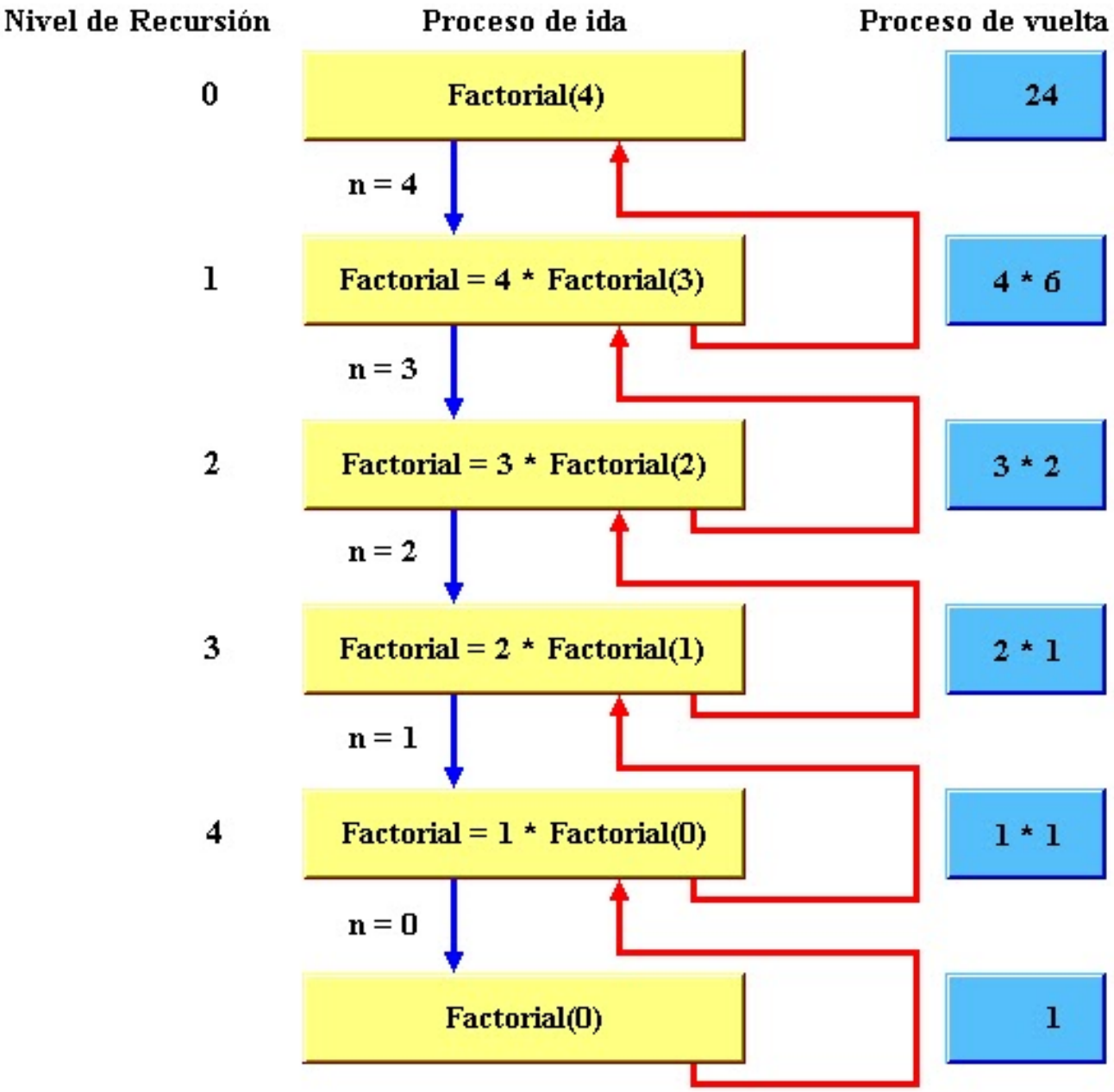
- **Recursividad Lineal No Final**
- **Recursividad Lineal Final**

## Recursividad Lineal No Final:

**El resultado final de la función que llama *se obtiene evaluando una expresión que contiene una llamada recursiva más simple.***

**Ejemplo: Cálculo del factorial de un número**

# Tipos de Recursividad: Recursividad Lineal No Final



# Tipos de Recursividad: Recursividad Lineal No Final

```
#include <iostream>

using namespace std;

long factorial (int numero);

int main()
{
    int n;
    char resp;

    do {
        cout << "\n\tIntroduza un numero: ";
        cin >> n;
        cout <<"\n\tEl factorial de " << n << " es " << factorial(n);

        cout << "\n\n\tDesea repetir el factorial con otro valor (S/N)? ";
        cin >> resp;

        resp = toupper(resp);
    } while (resp == 'S');

     cout << "\n\n\t";
     return (0);
}

long factorial (int numero)
{
    if (numero > 1) return (numero * factorial (numero -1));
    else return (1);
}
```

**Recursividad Lineal Final:**

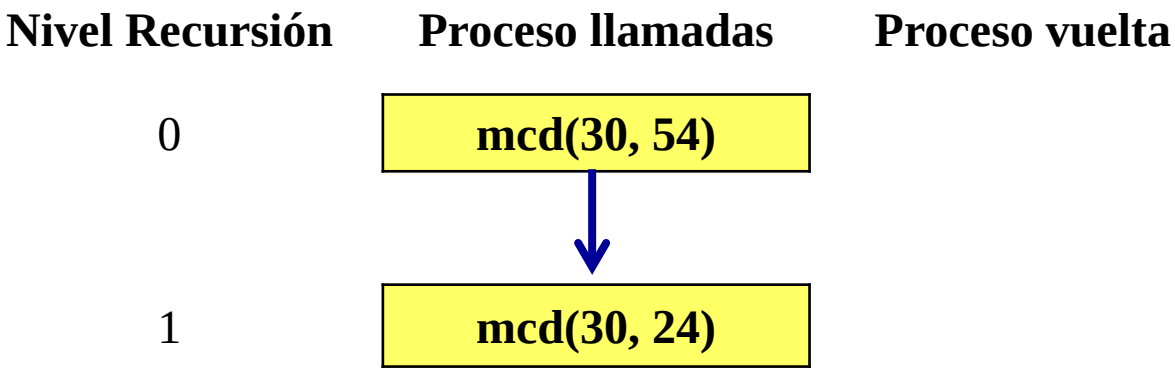
**El resultado final de la función que llama *se obtiene en la ejecución de la última llamada recursiva*.**

**Ejemplo: Cálculo del máximo común divisor de dos números.**

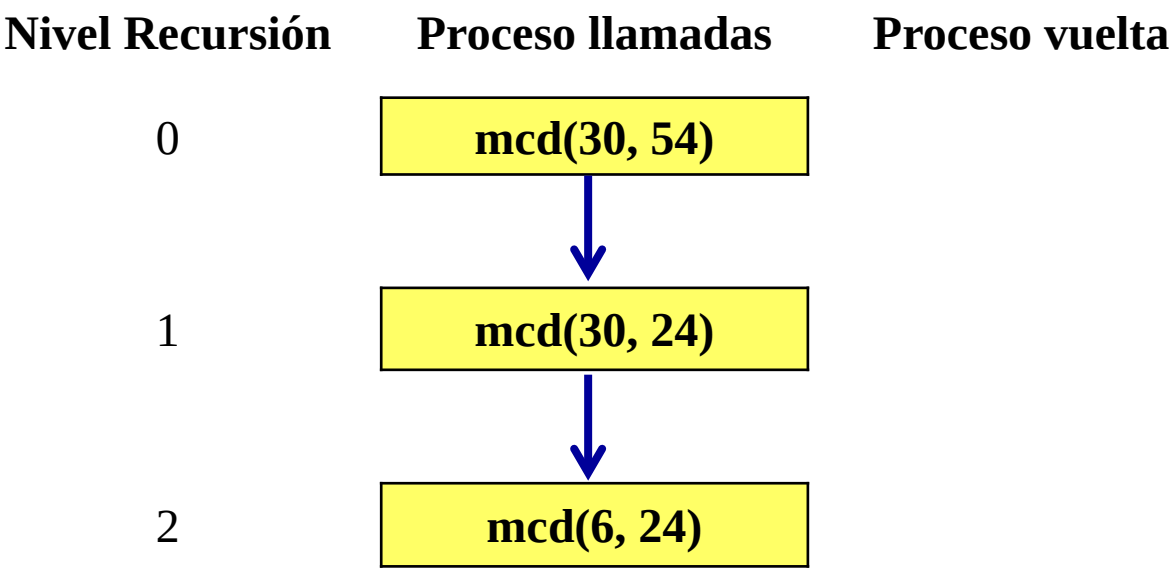
# Tipos de Recursividad: Recursividad Lineal Final

| Nivel Recursión | Proceso llamadas | Proceso vuelta |
|---|---|---|
| 0 | **mcd(30, 54)** | |

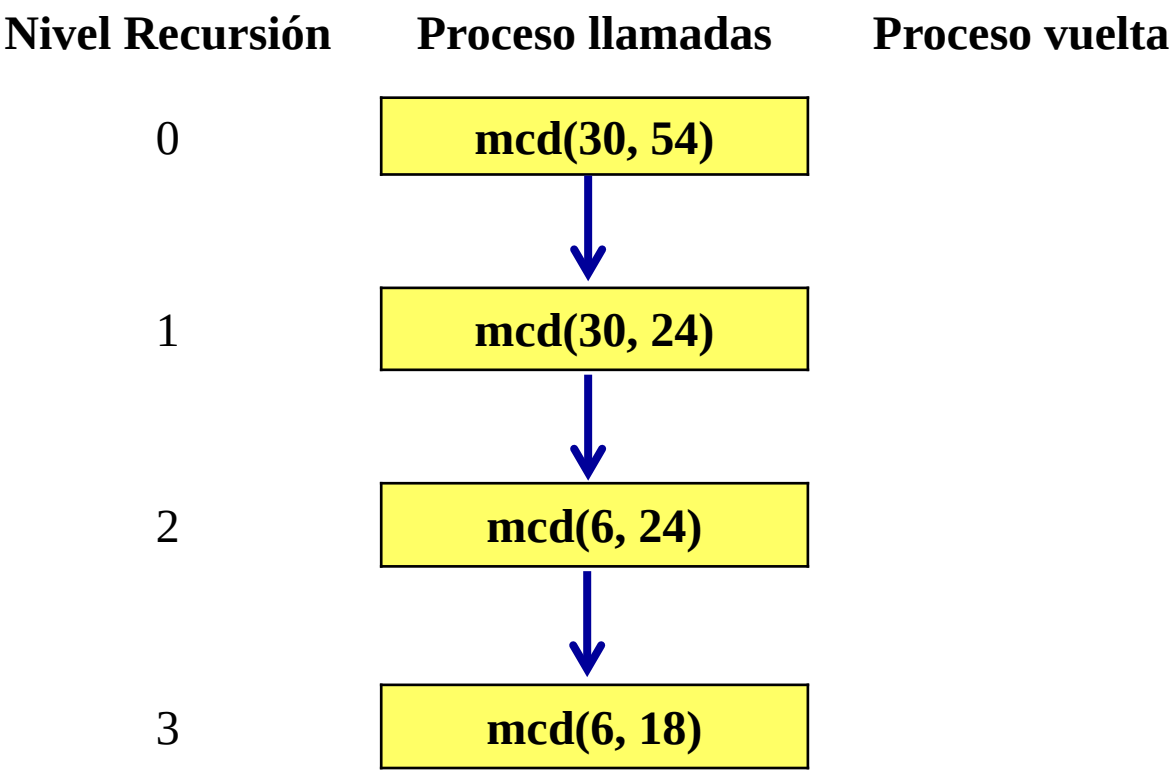
# Tipos de Recursividad: Recursividad Lineal Final



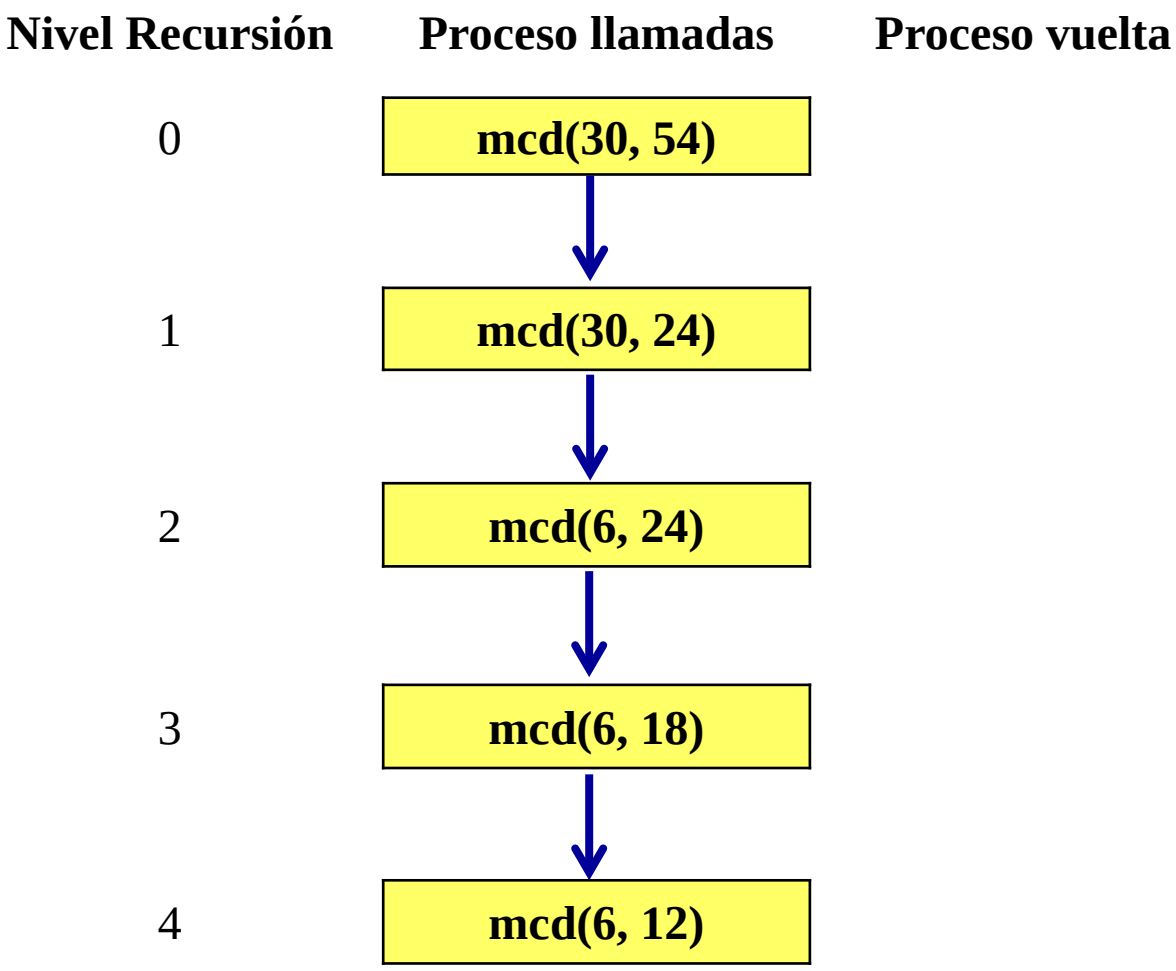
# Tipos de Recursividad: Recursividad Lineal Final



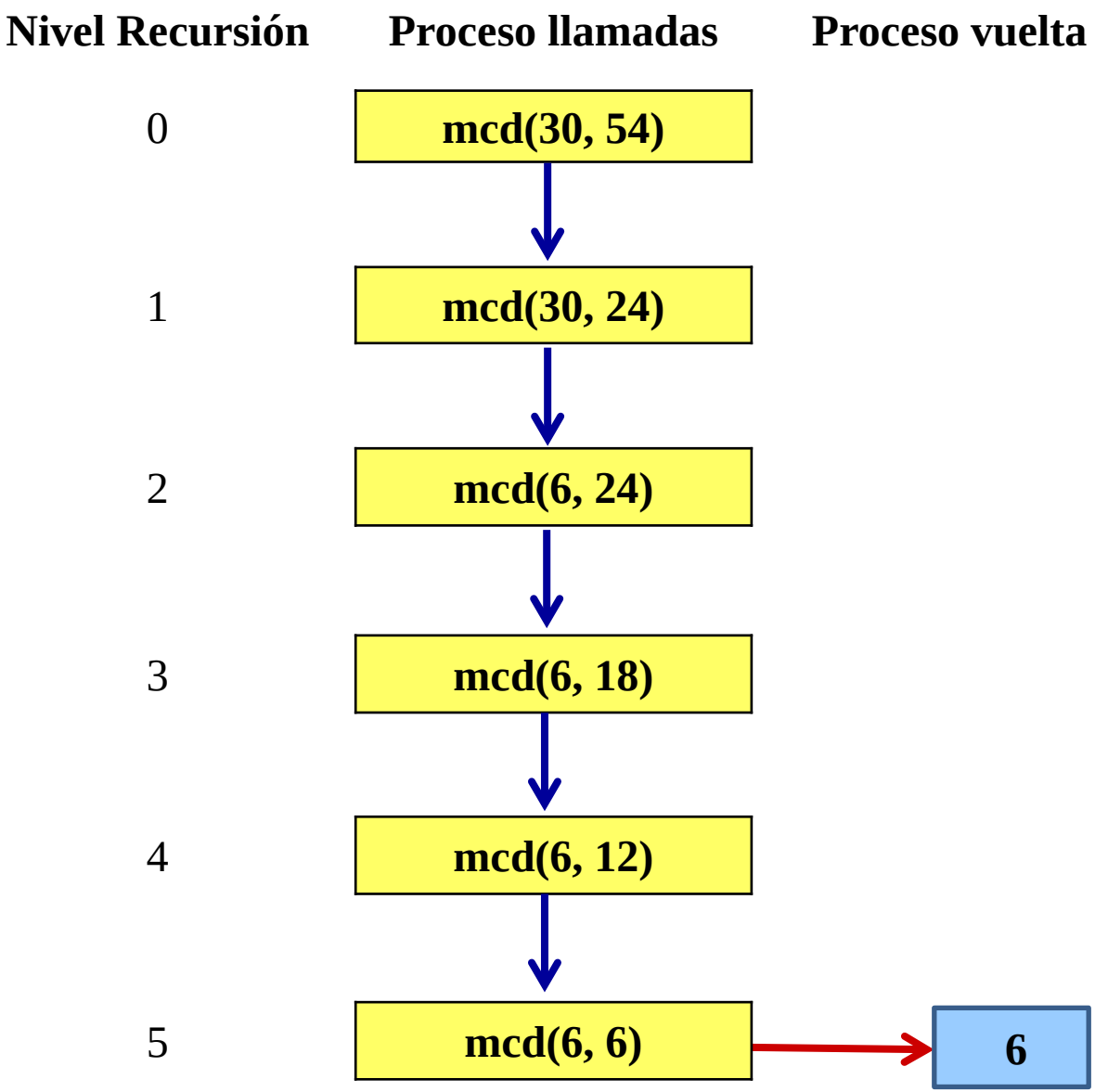
# Tipos de Recursividad: Recursividad Lineal Final



# Tipos de Recursividad: Recursividad Lineal Final



# Tipos de Recursividad: Recursividad Lineal Final



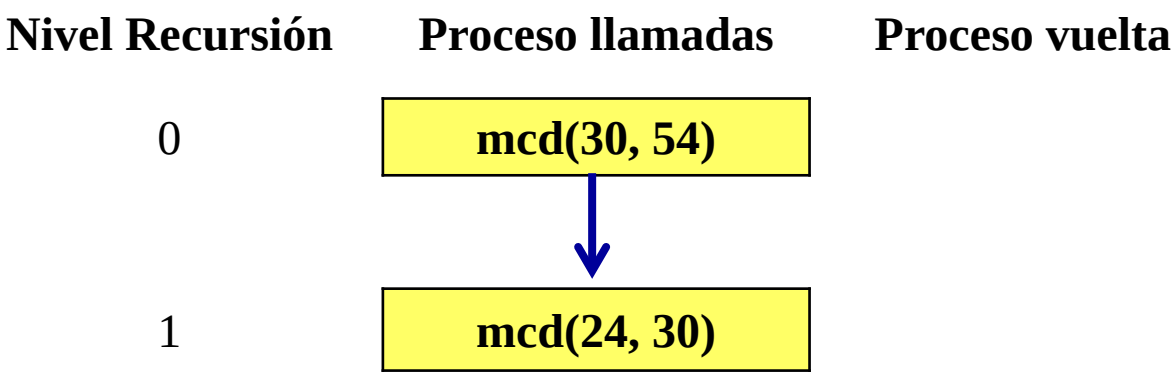
# Tipos de Recursividad: Recursividad Lineal Final

```
#include <iostream>

using namespace std;

int mcd (int a, int b);

int main()
{
    int n1, n2;
    char resp;

    do {
            cout << "\n\tIntroduza el primer numero: ";
            cin >> n1;
            cout << "\n\tIntroduza el segundo numero: ";
            cin >> n2;
            cout << "\n\tEl m.c.d de " << n1 << " y " << n2 << " es: " << mcd(n1, n2);
            cout << "\n\n\tDesea repetir el mcd con otros valores (S/N)? ";
            cin >> resp;

            resp = toupper(resp);
        } while (resp == 'S');

    cout << "\n\n\t";
    return 0;
}

int mcd (int a, int b)
{  if (a == b) return (a);
   else if ( a > b) return (mcd(a-b, b));
   else return (mcd(a, b-a));
}
```

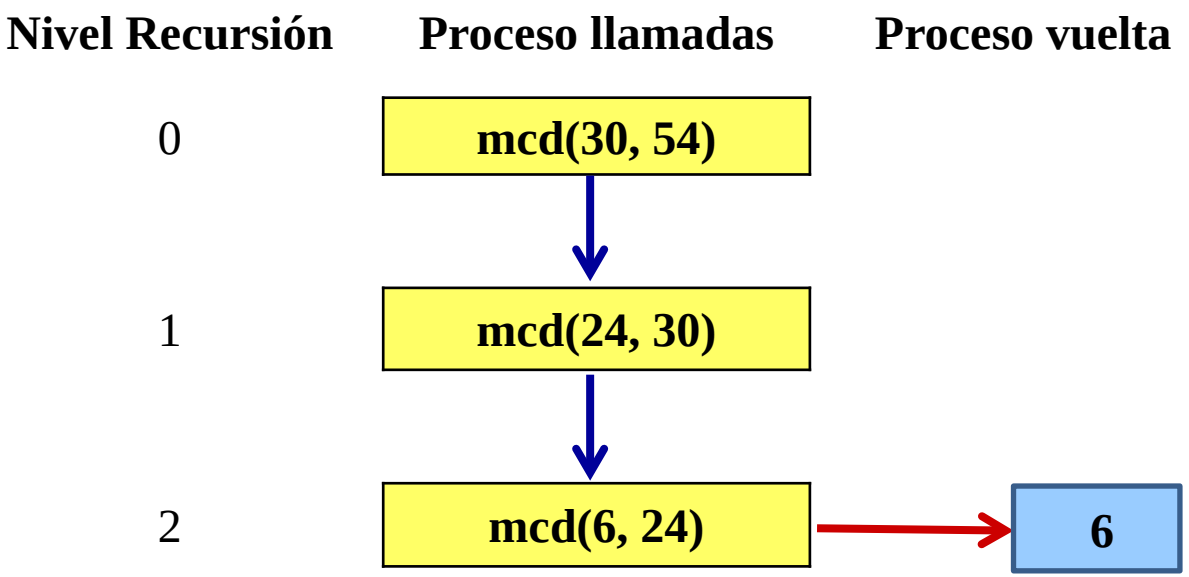
# Tipos de Recursividad: Recursividad Lineal Final

| Nivel Recursión | Proceso llamadas | Proceso vuelta |
|---|---|---|
| 0 | **mcd(30, 54)** | |

# Tipos de Recursividad: Recursividad Lineal Final



# Tipos de Recursividad: Recursividad Lineal Final



## Recursividad Múltiple:

Cuando ***produce más de una llamada recursiva*** **hacia sí misma** con ***diferentes parámetros***.

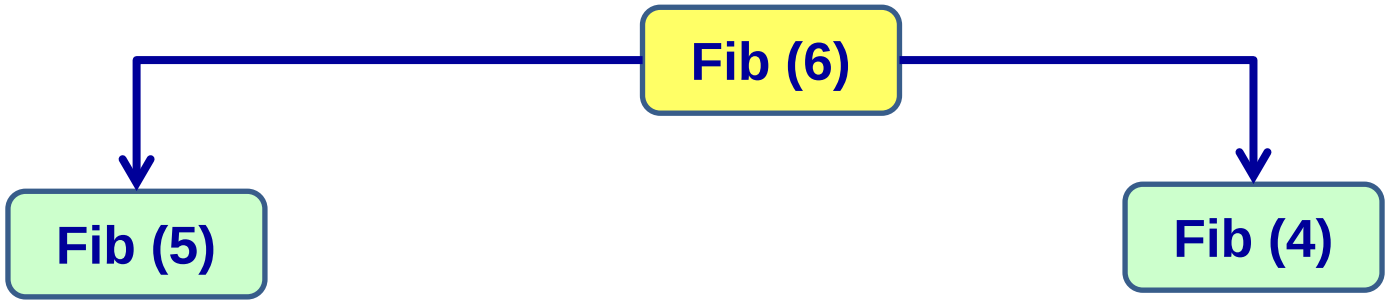
**Ejemplo: Cálculo de un determinado número de la serie de Fibonacci:**

$$f(0) = 0 \qquad f(1) = 1 \qquad f(n) = f(n-1)+f(n-2)$$

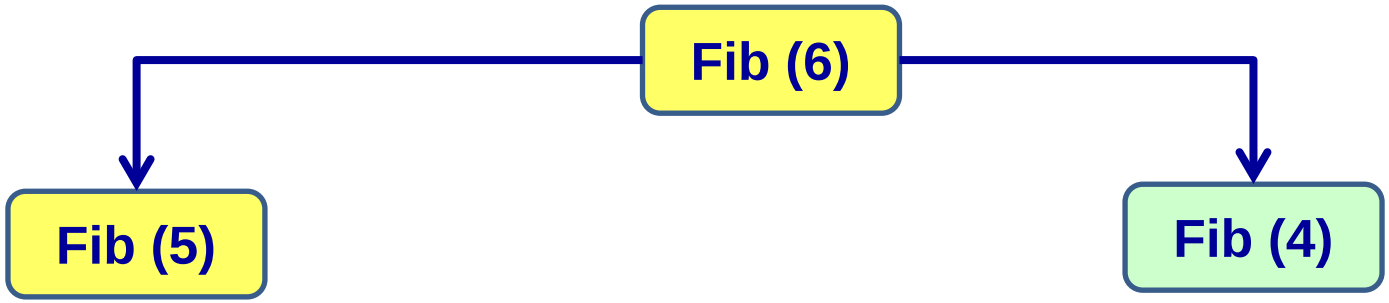
# Tipos de Recursividad: Recursividad Múltiple

**Fib (6)**

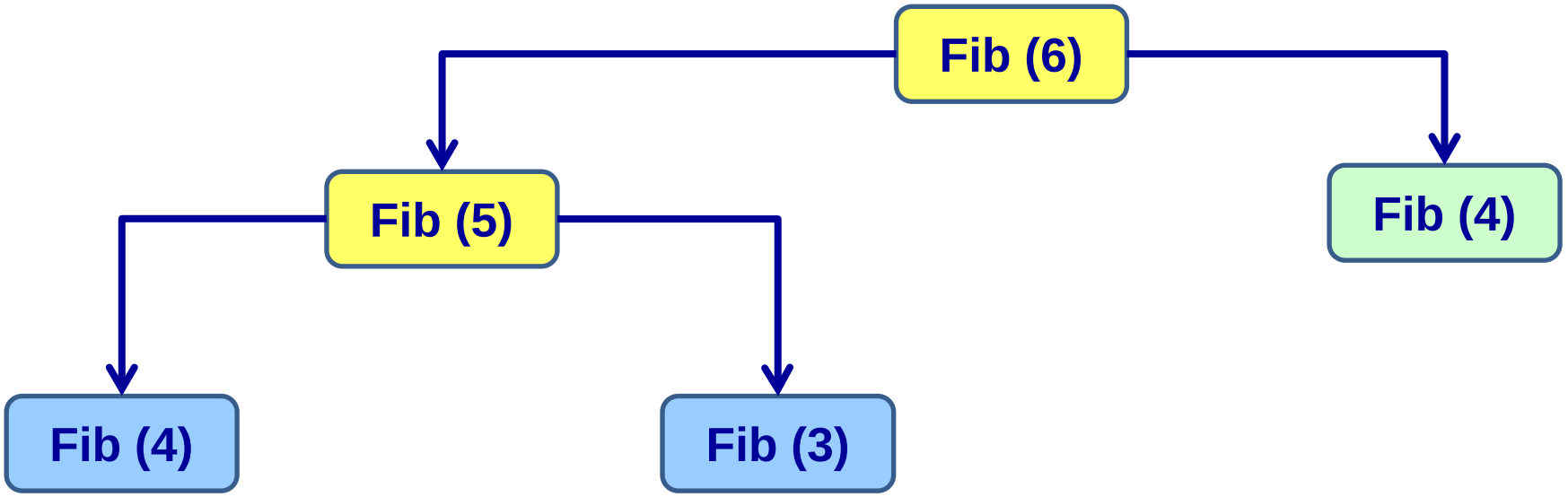
# Tipos de Recursividad: Recursividad Múltiple



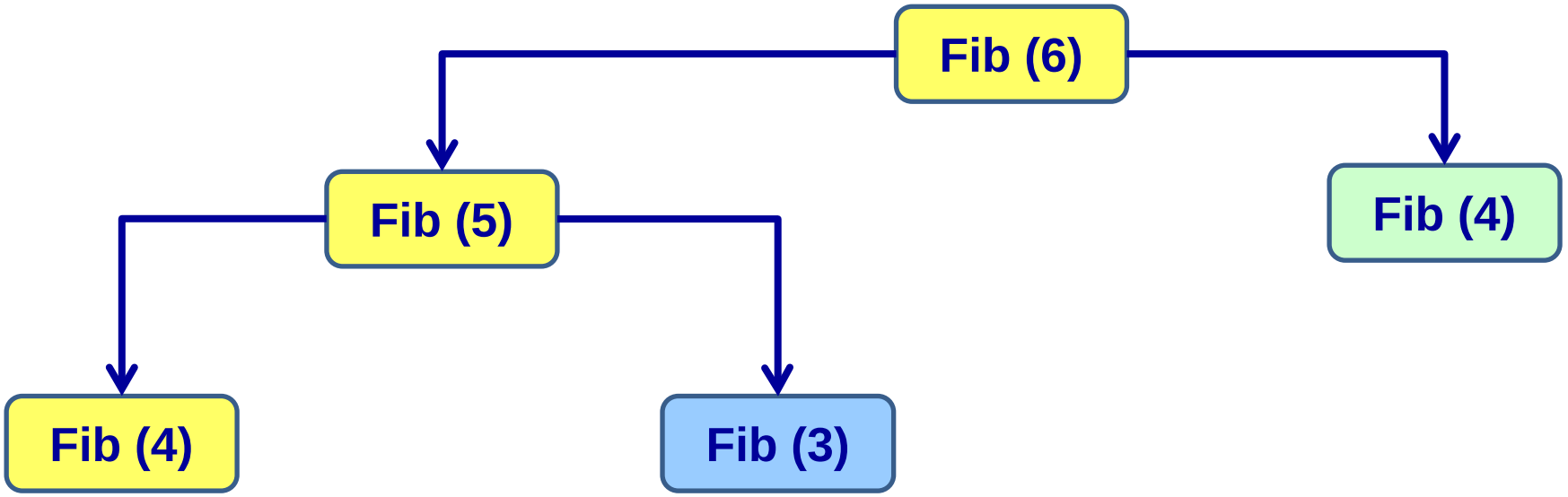
# Tipos de Recursividad: Recursividad Múltiple



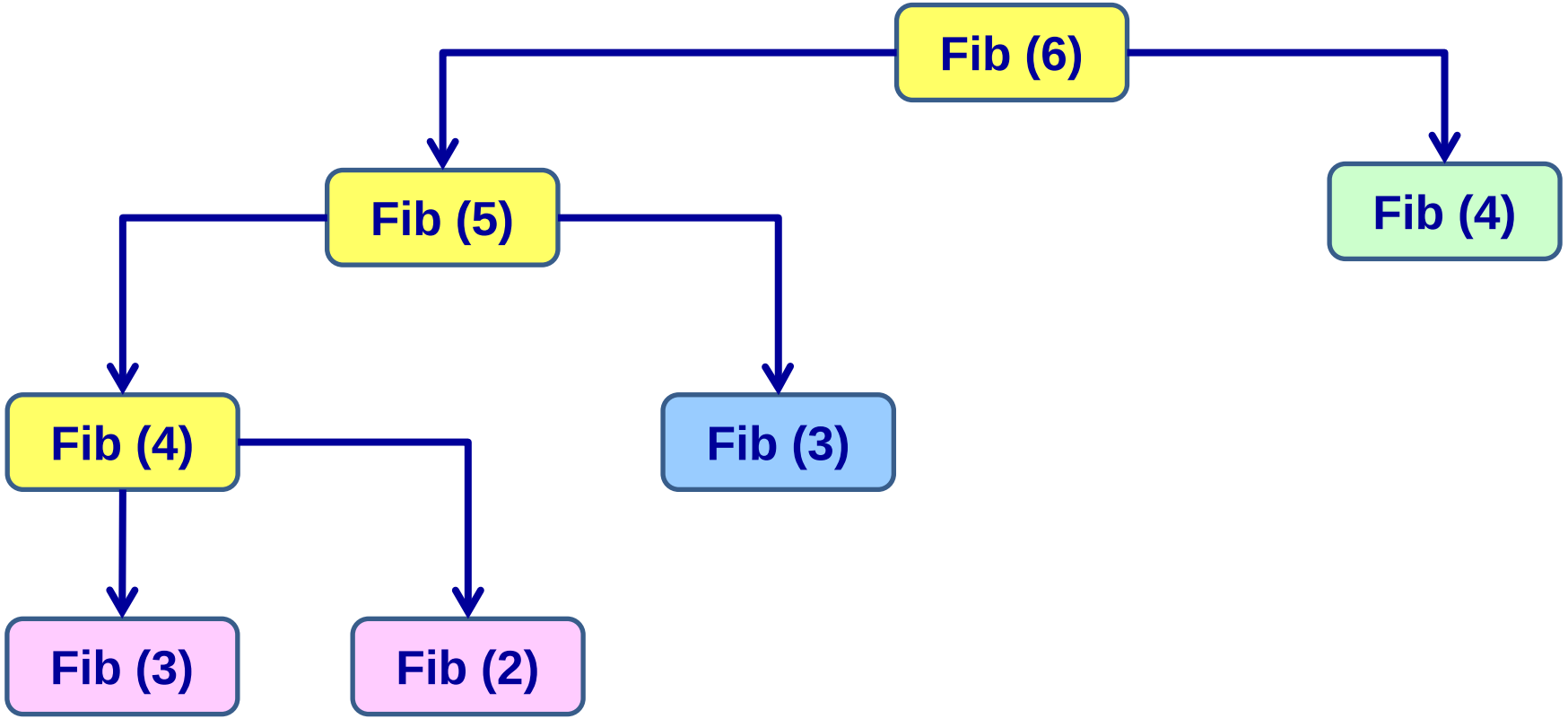
# Tipos de Recursividad: Recursividad Múltiple



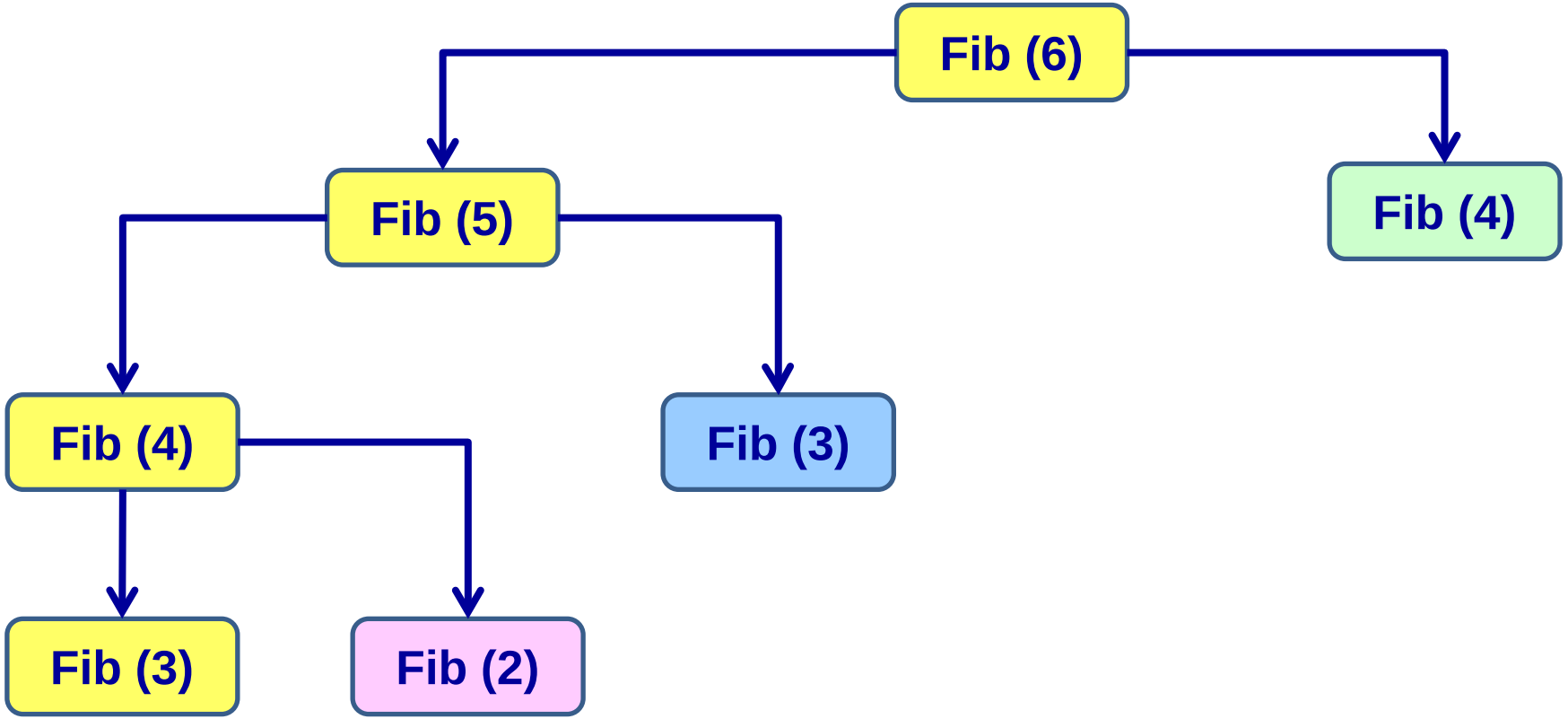
# Tipos de Recursividad: Recursividad Múltiple



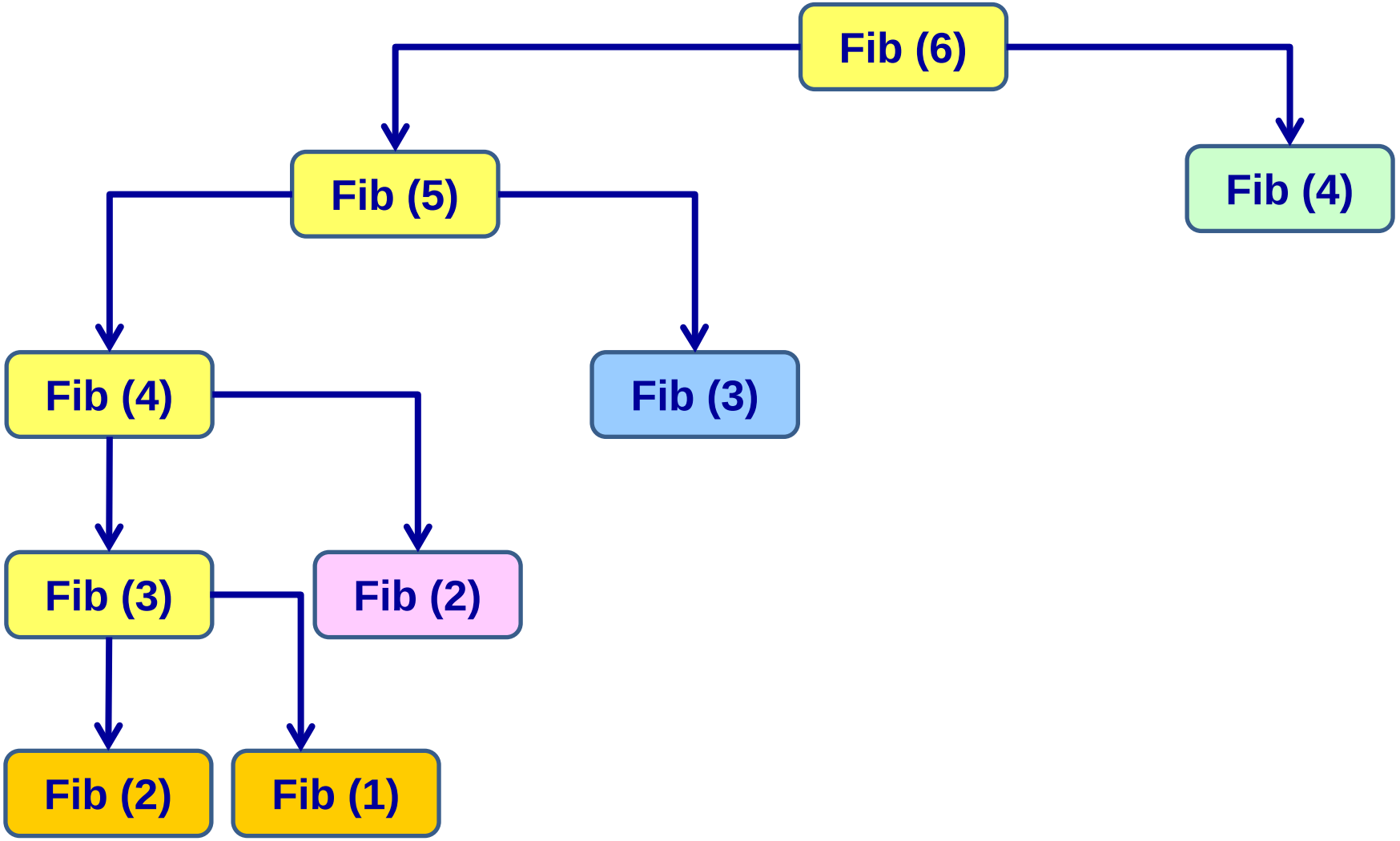
# Tipos de Recursividad: Recursividad Múltiple



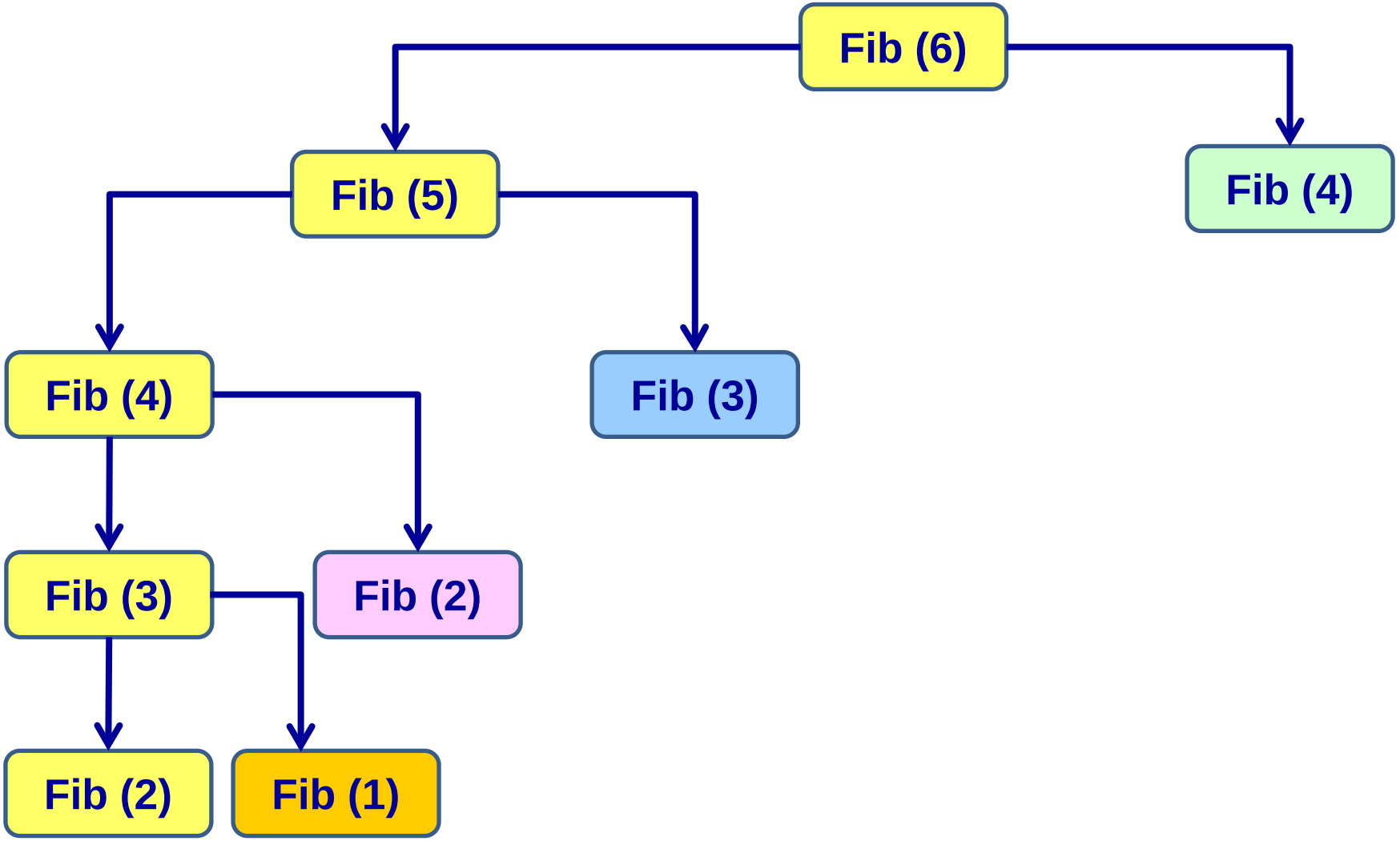
# Tipos de Recursividad: Recursividad Múltiple



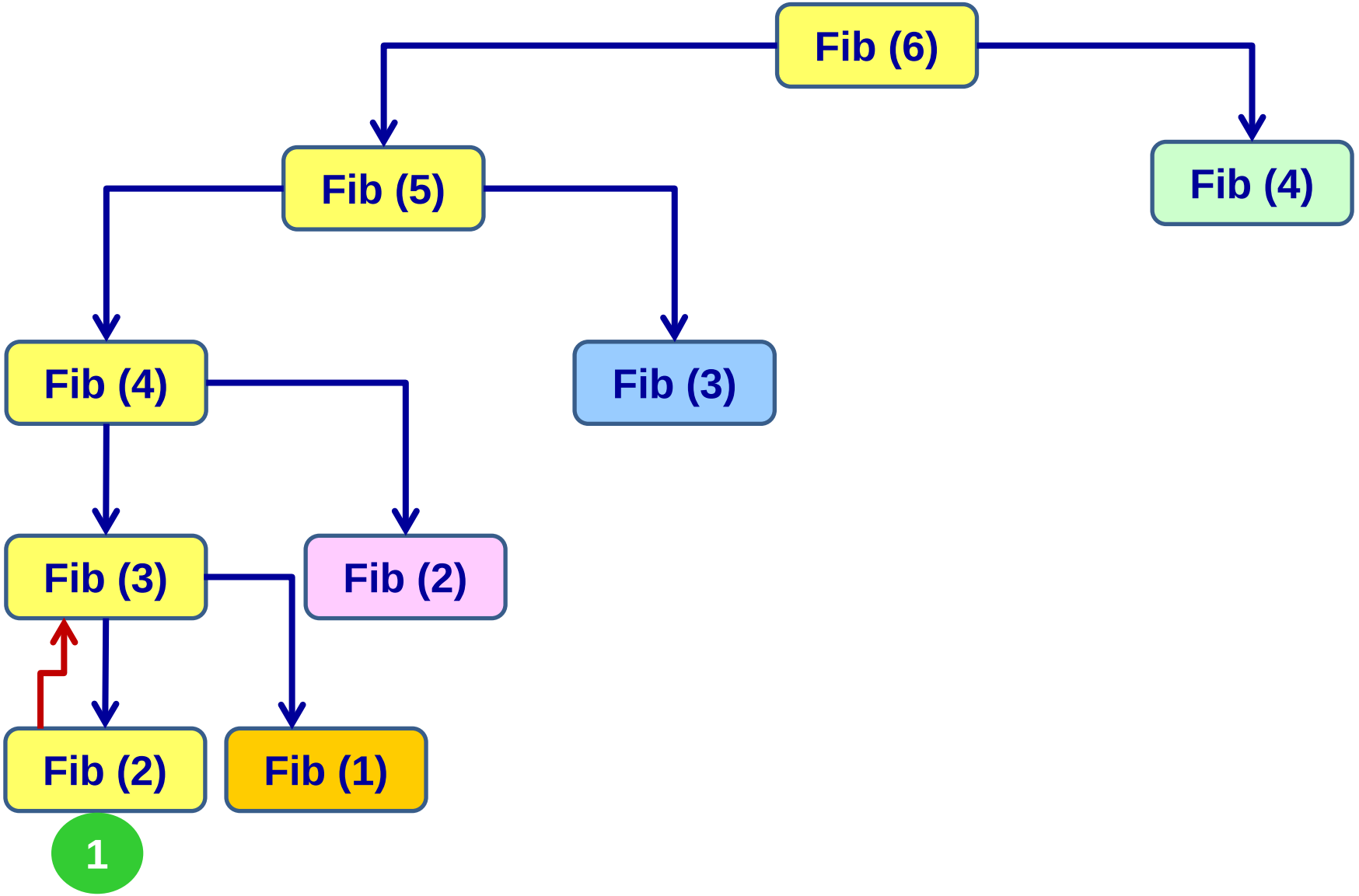
# Tipos de Recursividad: Recursividad Múltiple



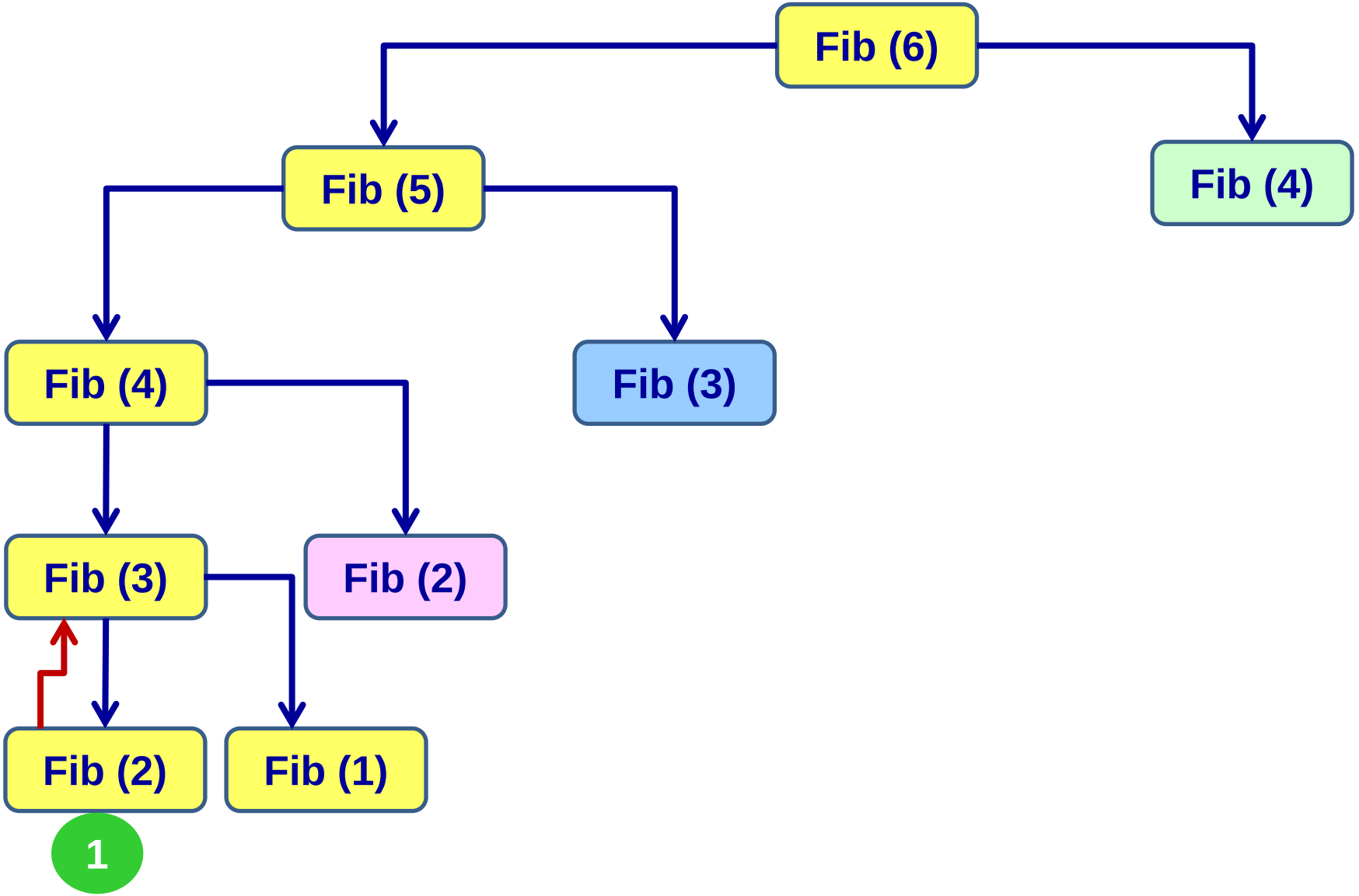
# Tipos de Recursividad: Recursividad Múltiple



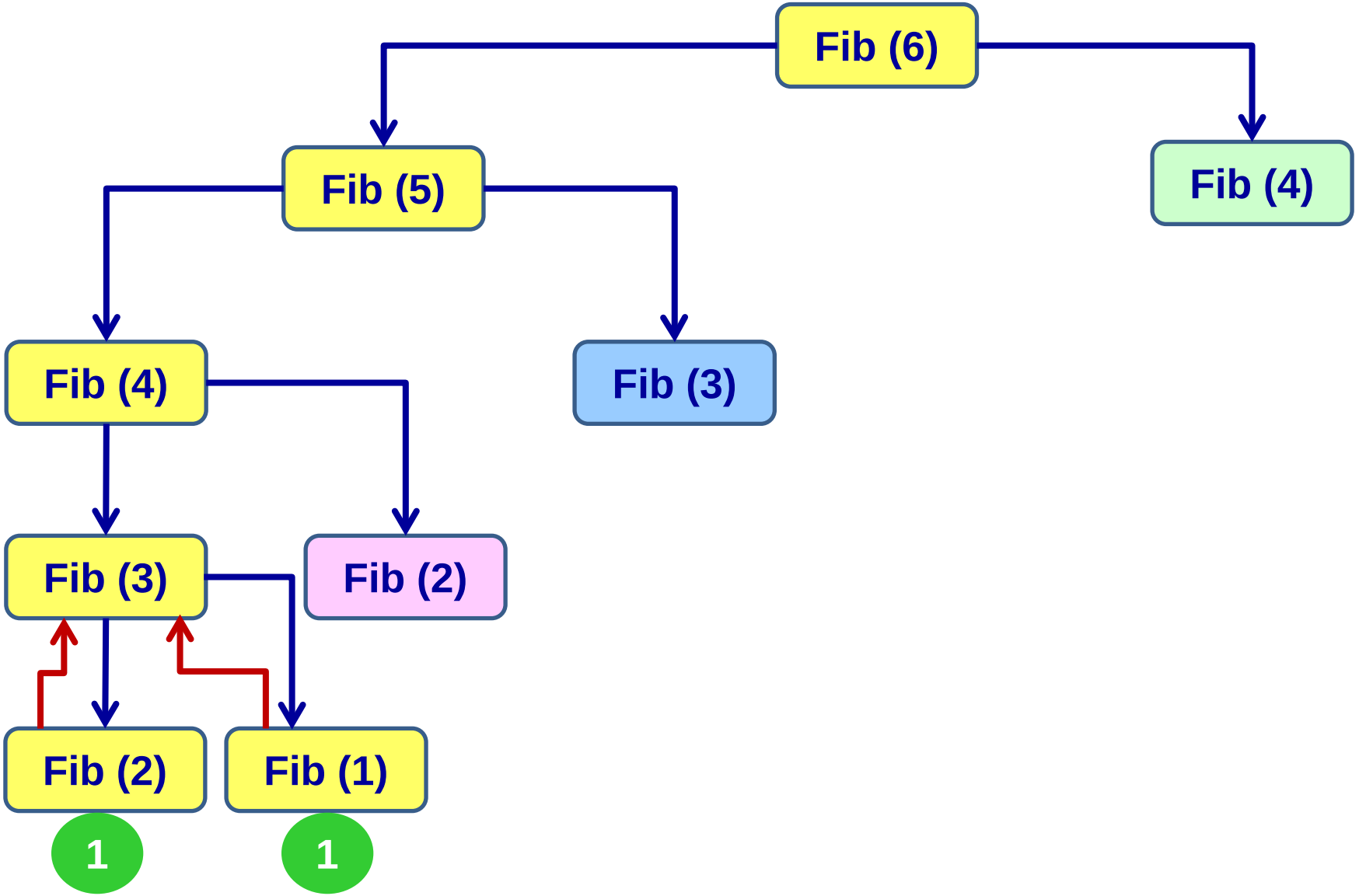
# Tipos de Recursividad: Recursividad Múltiple



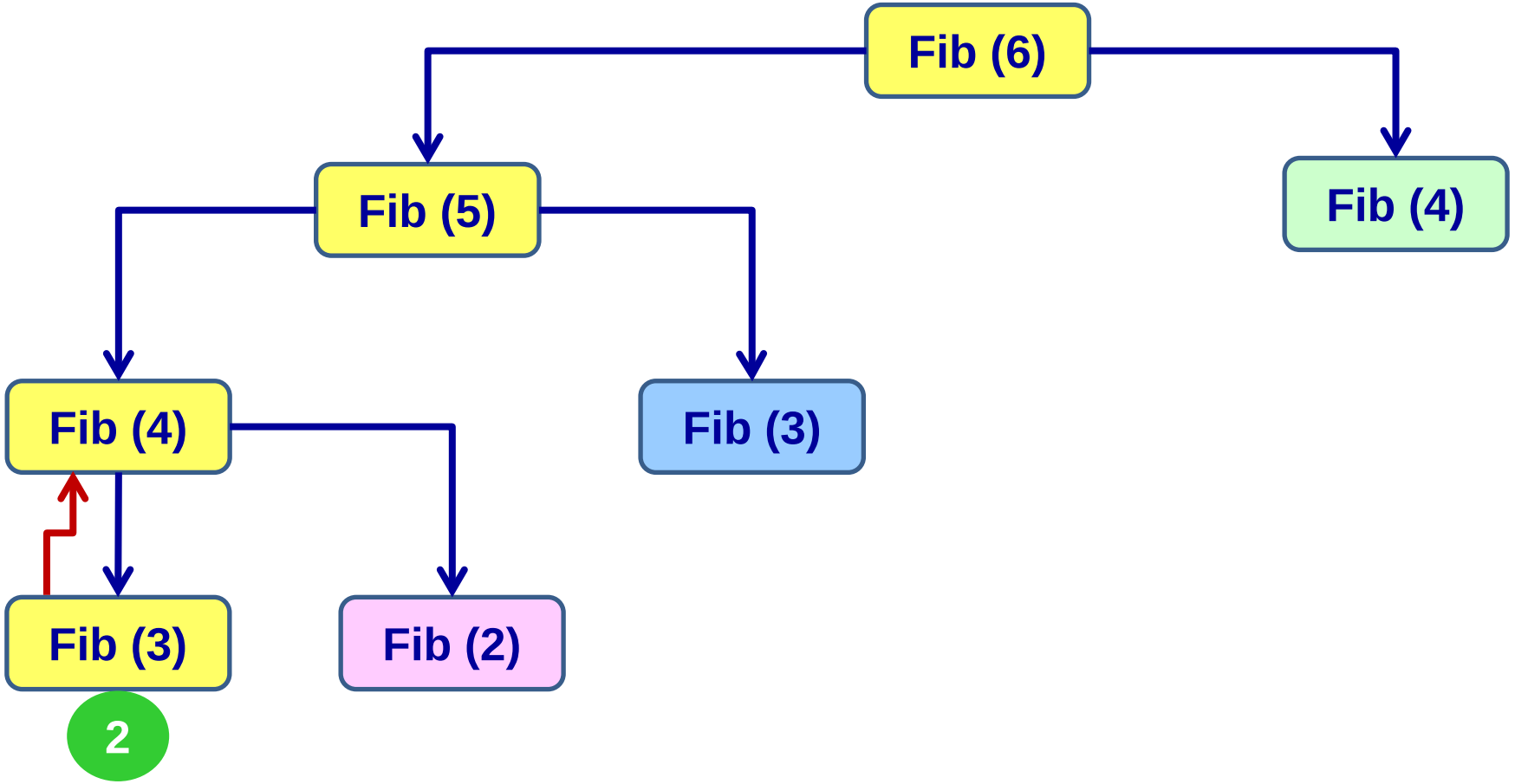
# Tipos de Recursividad: Recursividad Múltiple



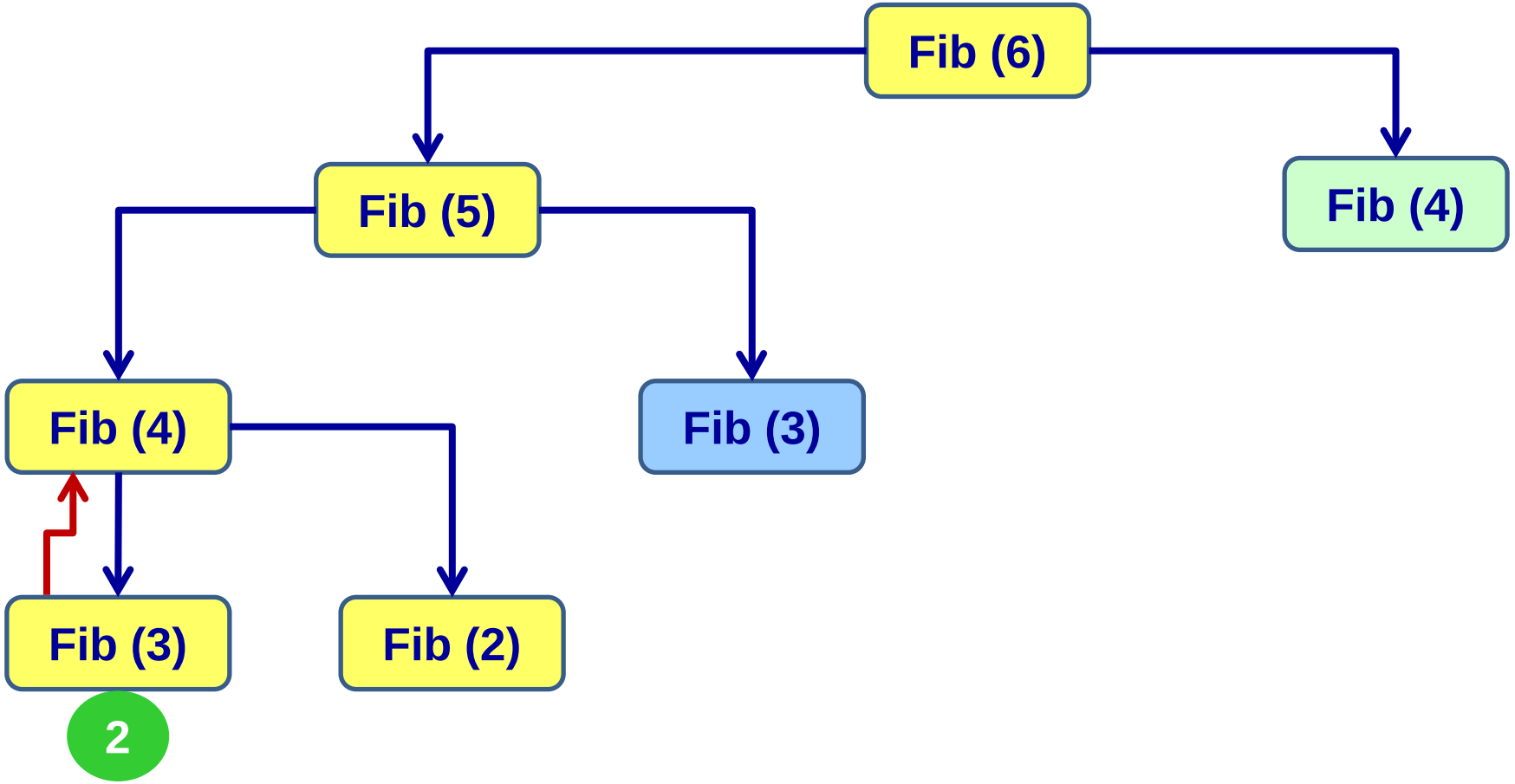
# Tipos de Recursividad: Recursividad Múltiple



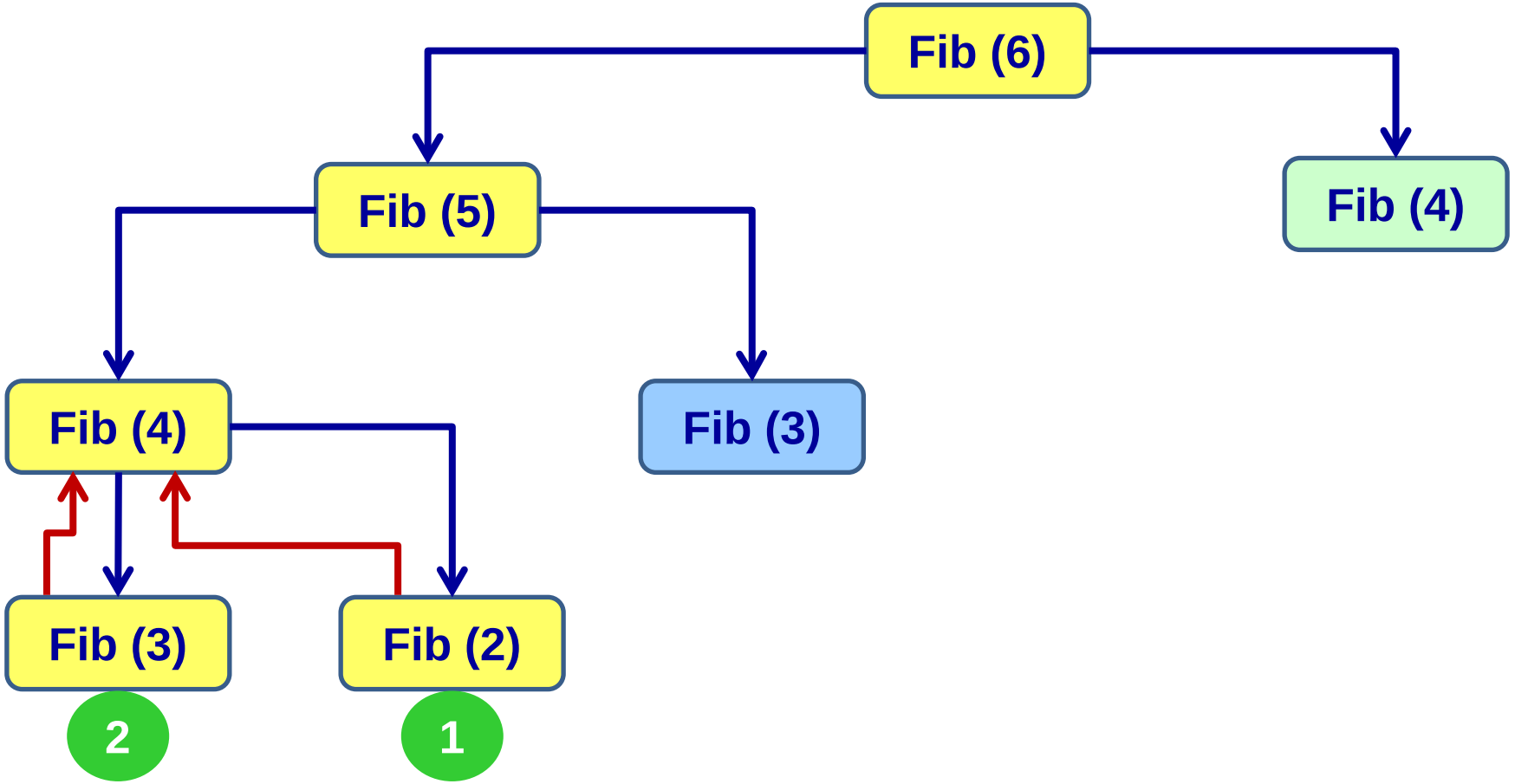
# Tipos de Recursividad: Recursividad Múltiple



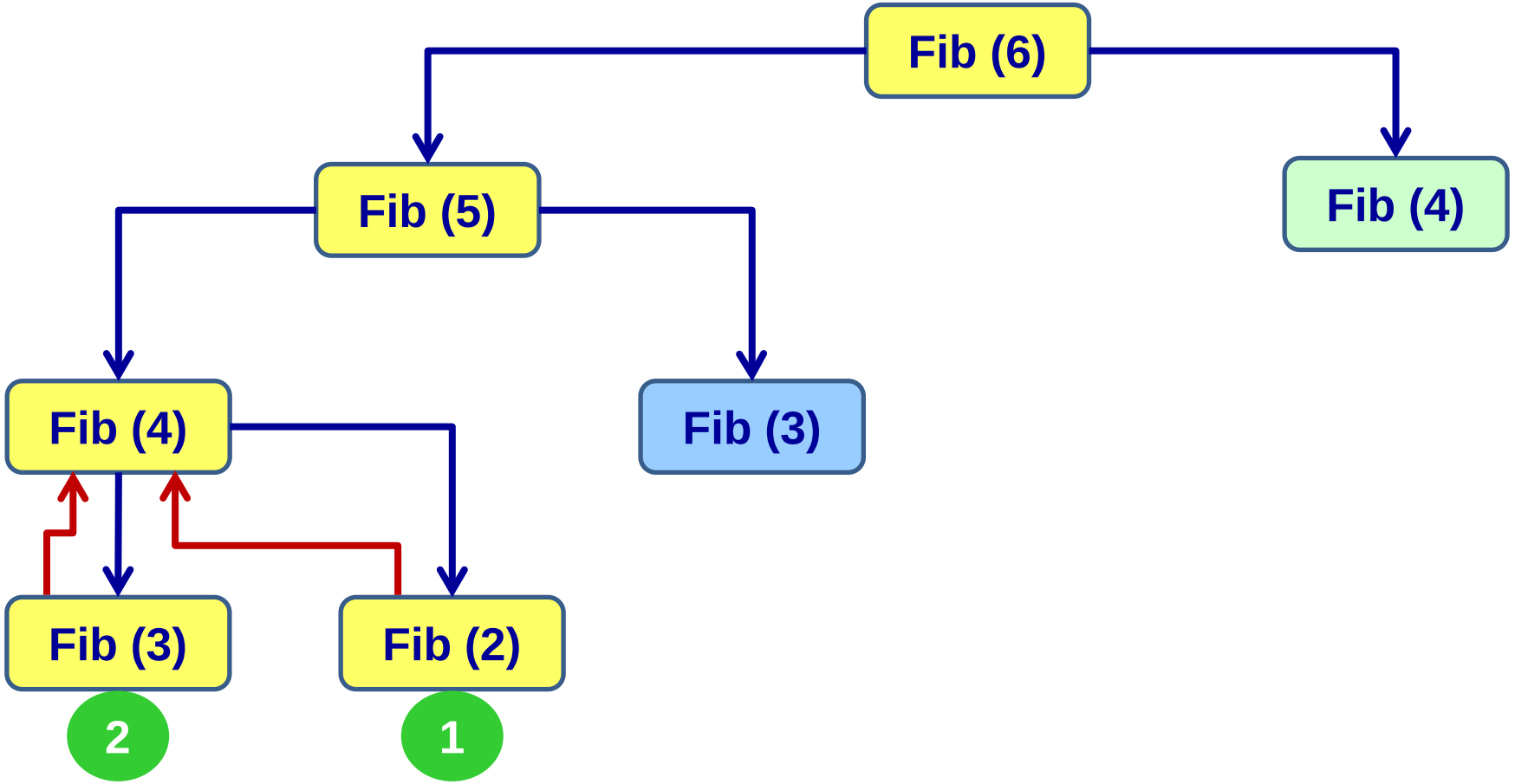
# Tipos de Recursividad: Recursividad Múltiple



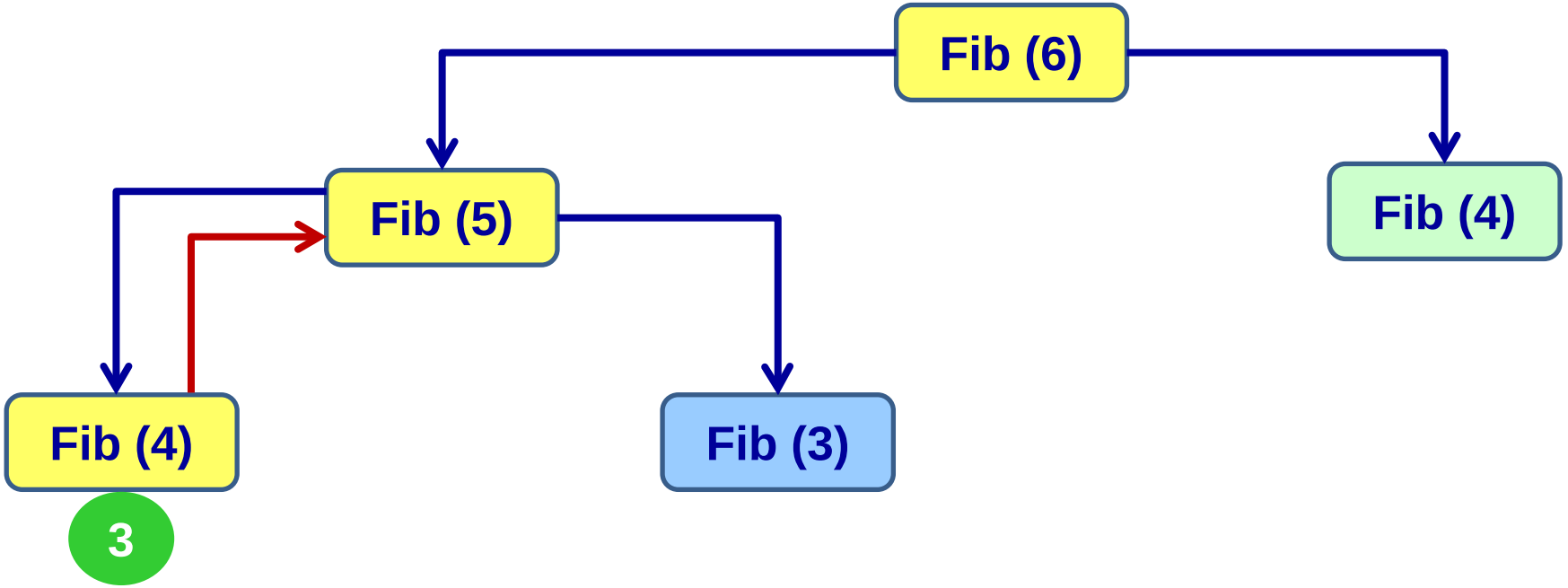
# Tipos de Recursividad: Recursividad Múltiple



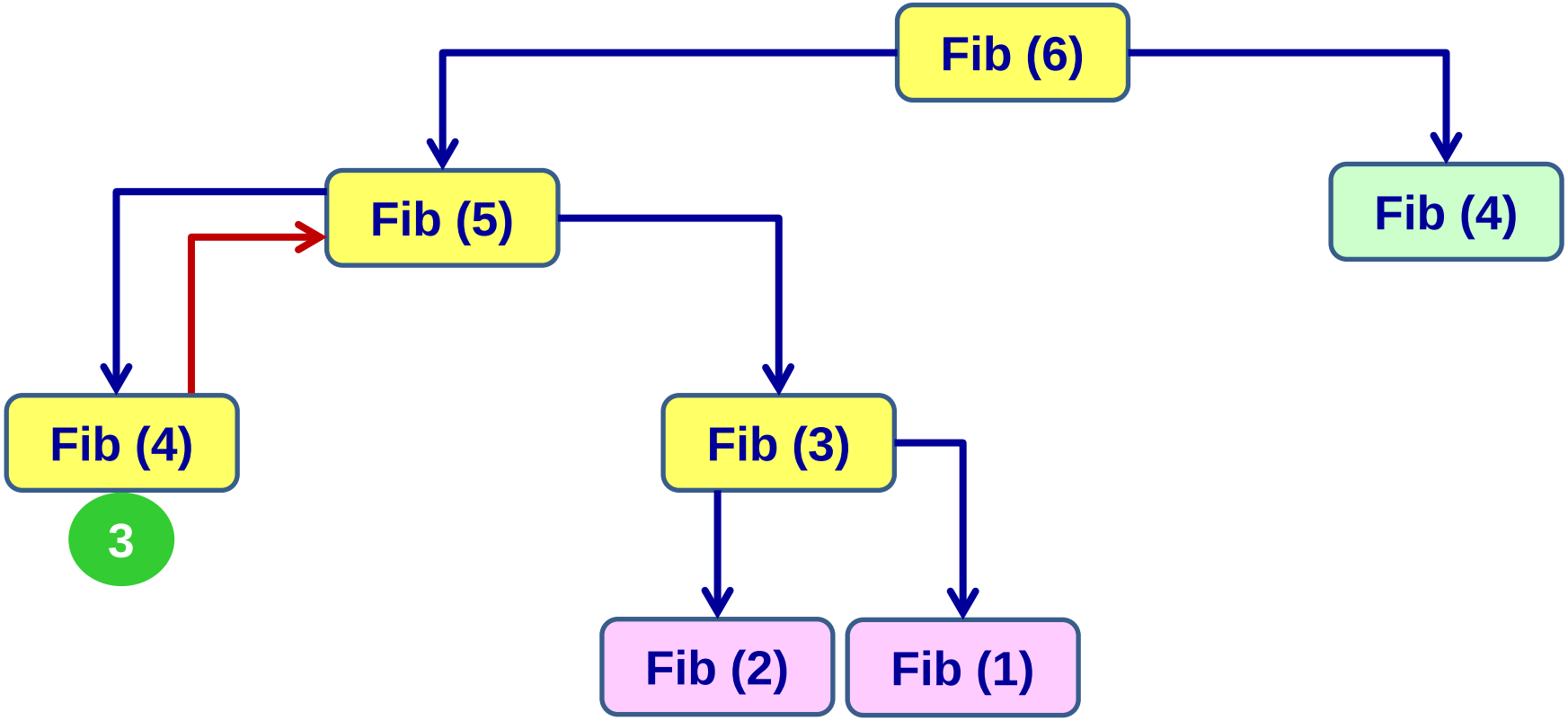
# Tipos de Recursividad: Recursividad Múltiple



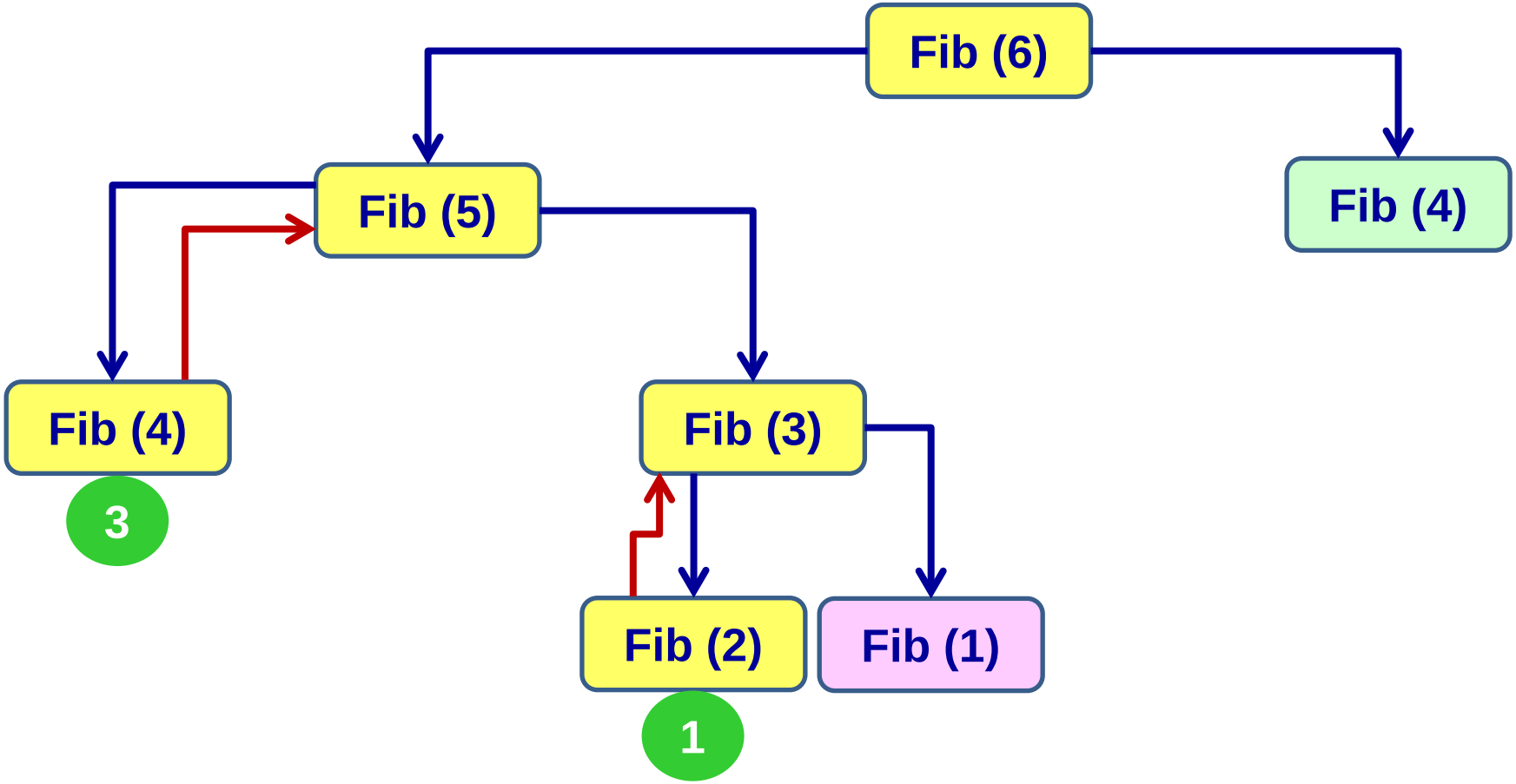
# Tipos de Recursividad: Recursividad Múltiple



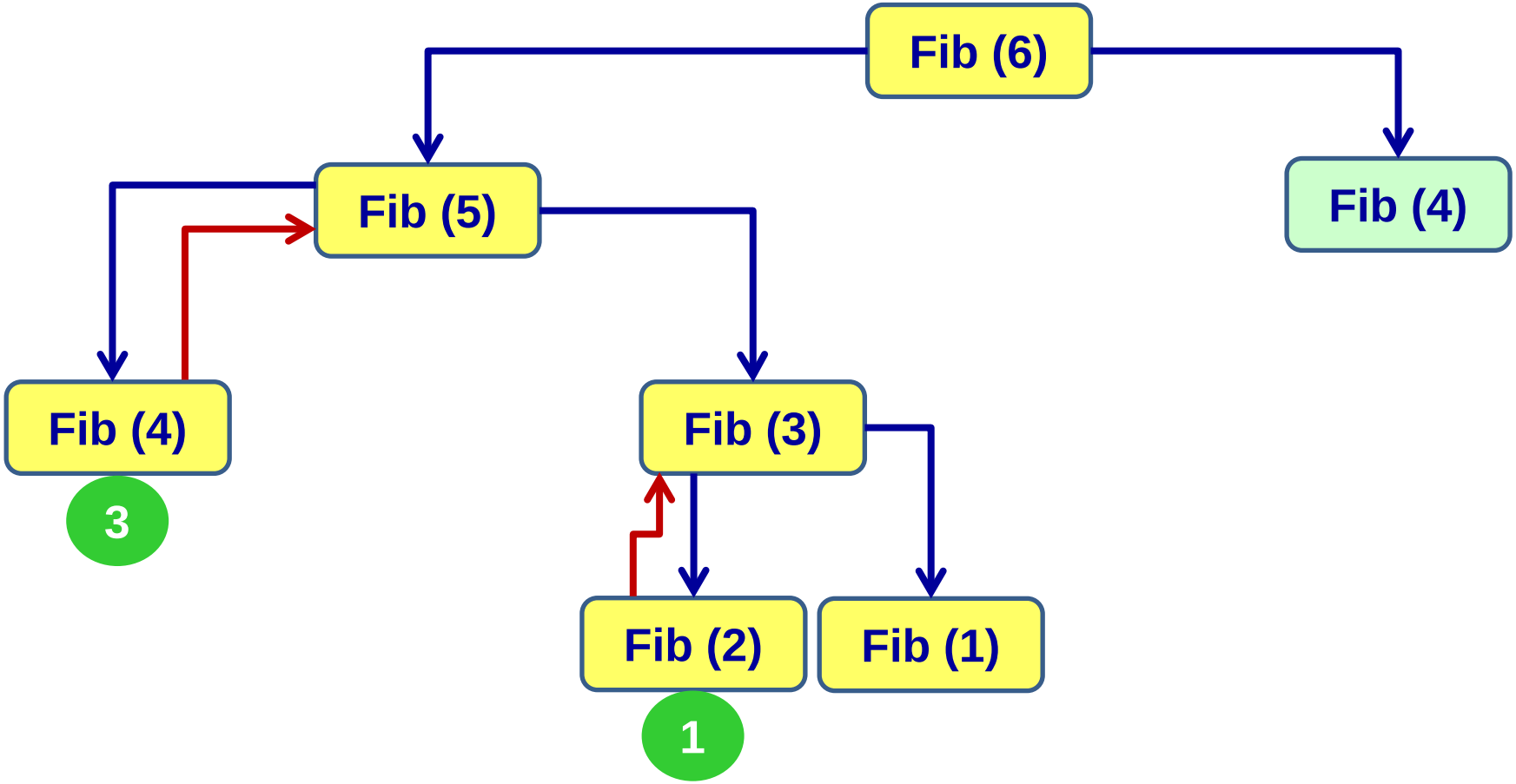
# Tipos de Recursividad: Recursividad Múltiple



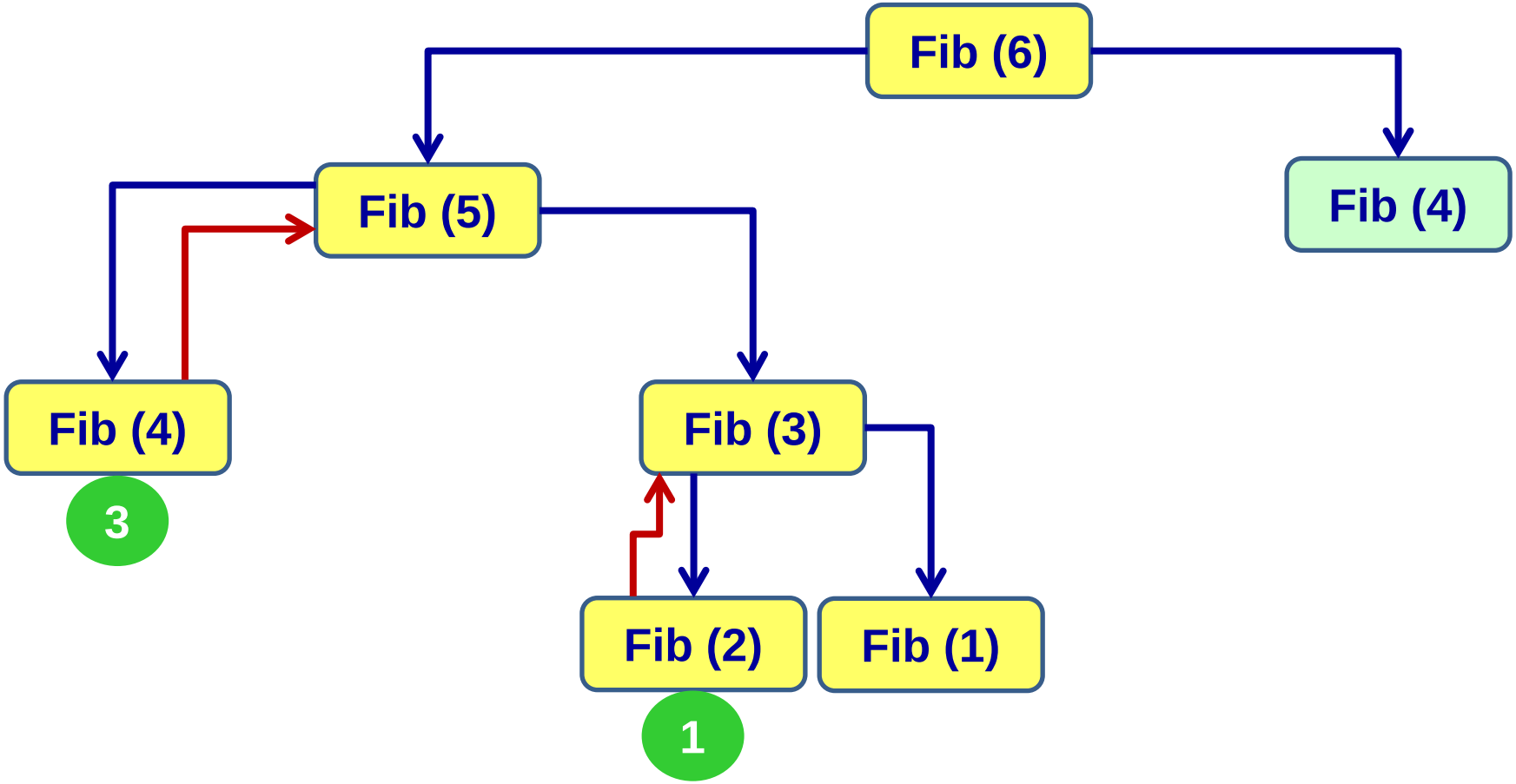
# Tipos de Recursividad: Recursividad Múltiple



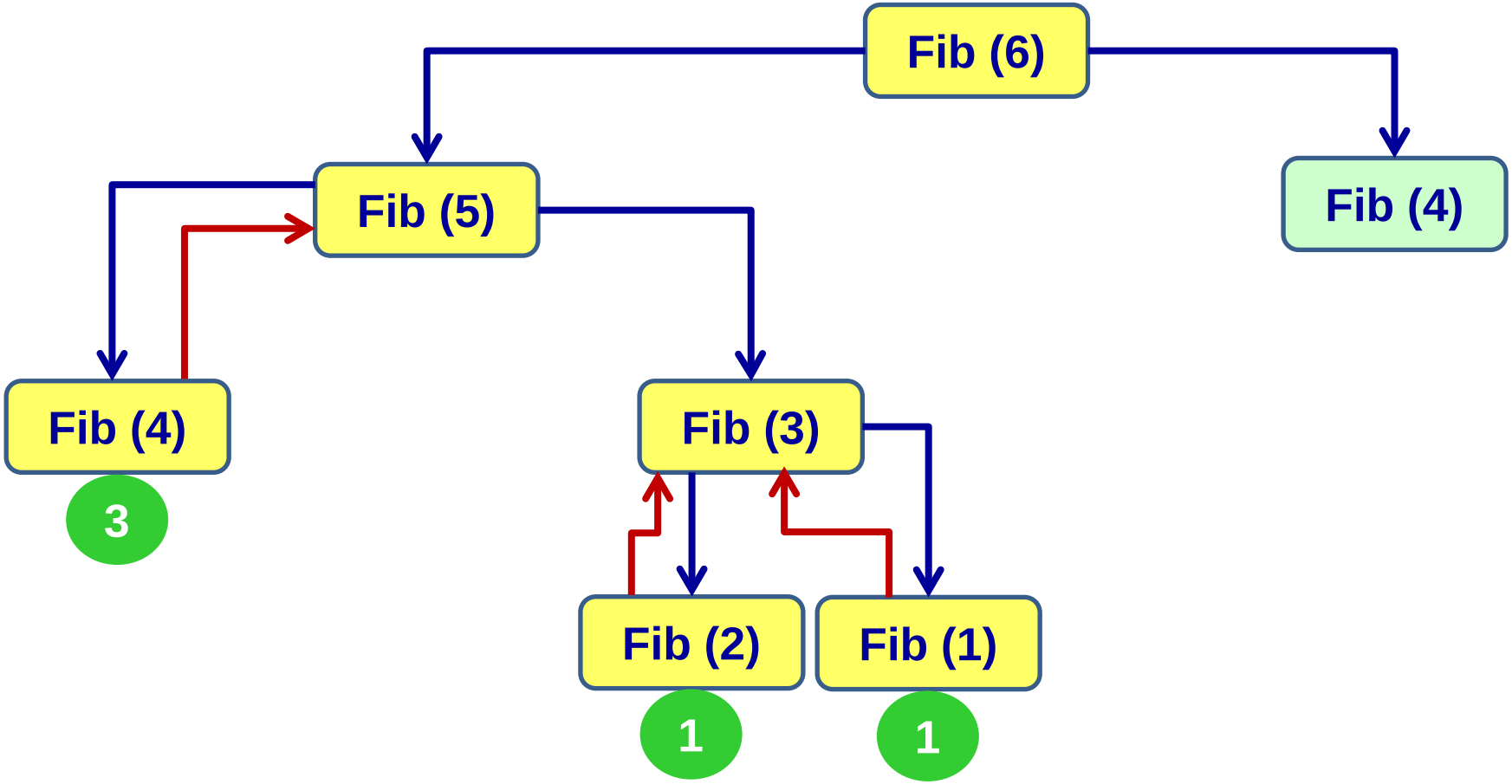
# Tipos de Recursividad: Recursividad Múltiple



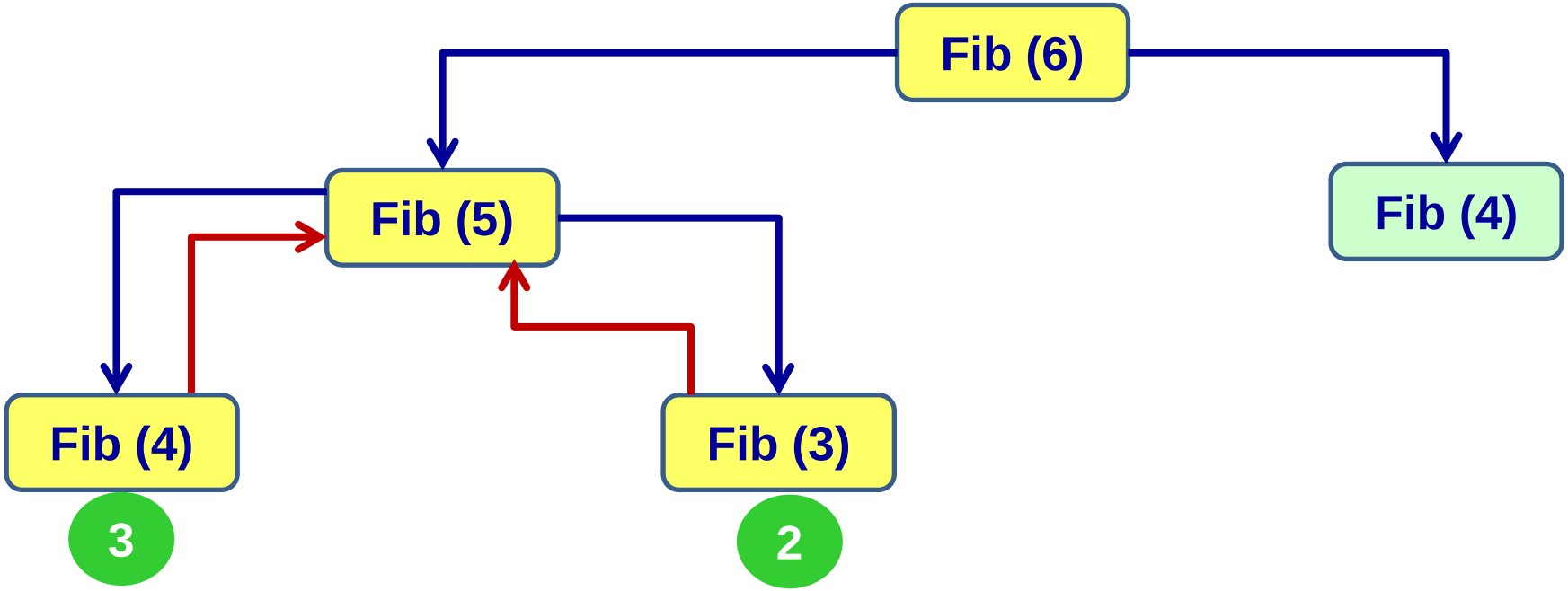
# Tipos de Recursividad: Recursividad Múltiple



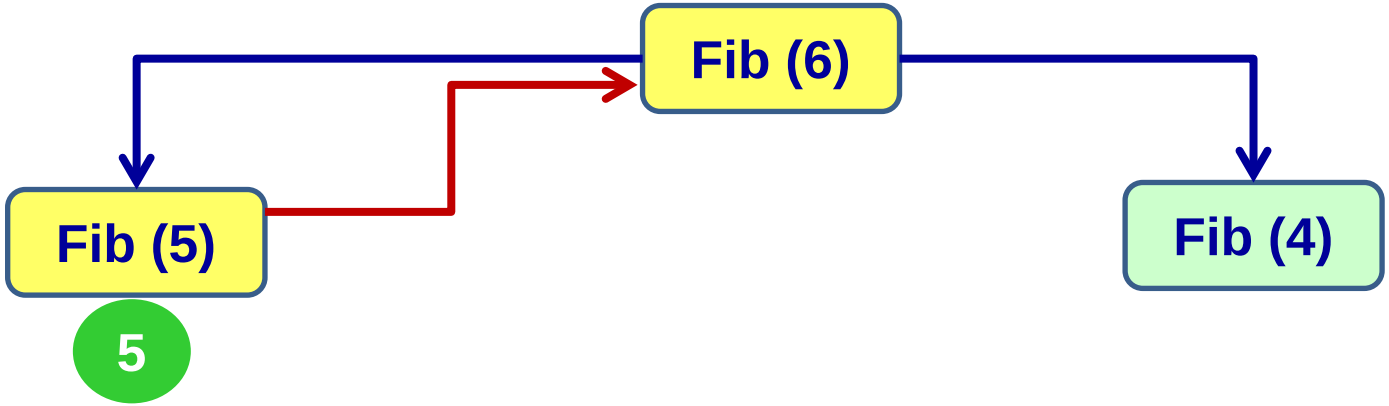
# Tipos de Recursividad: Recursividad Múltiple



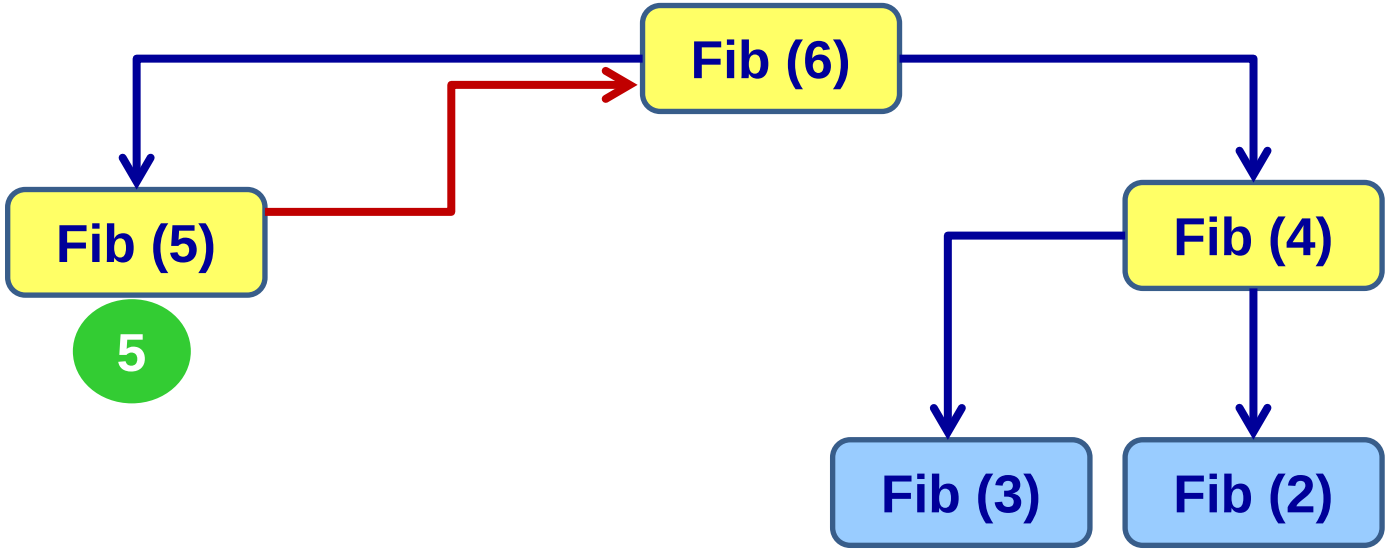
# Tipos de Recursividad: Recursividad Múltiple



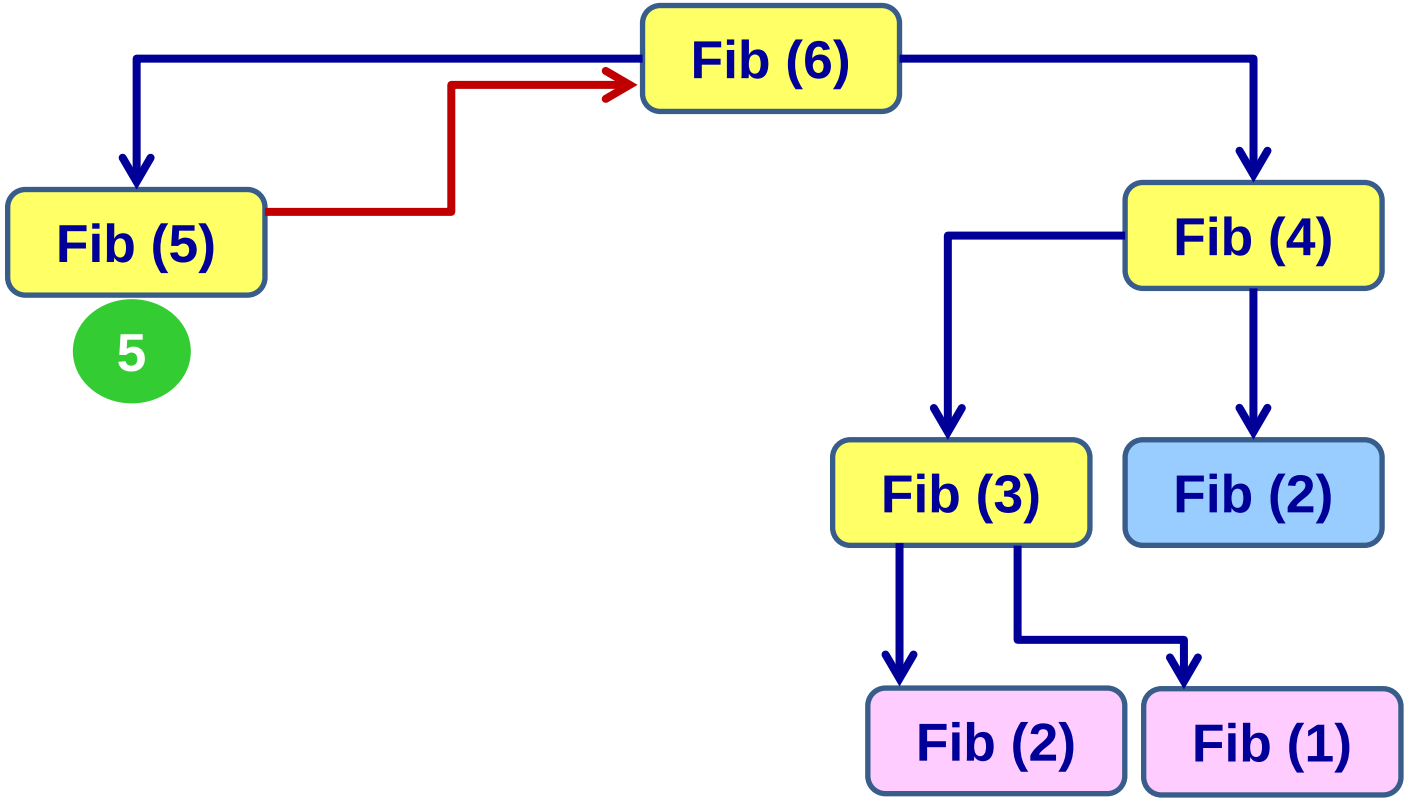
# Tipos de Recursividad: Recursividad Múltiple



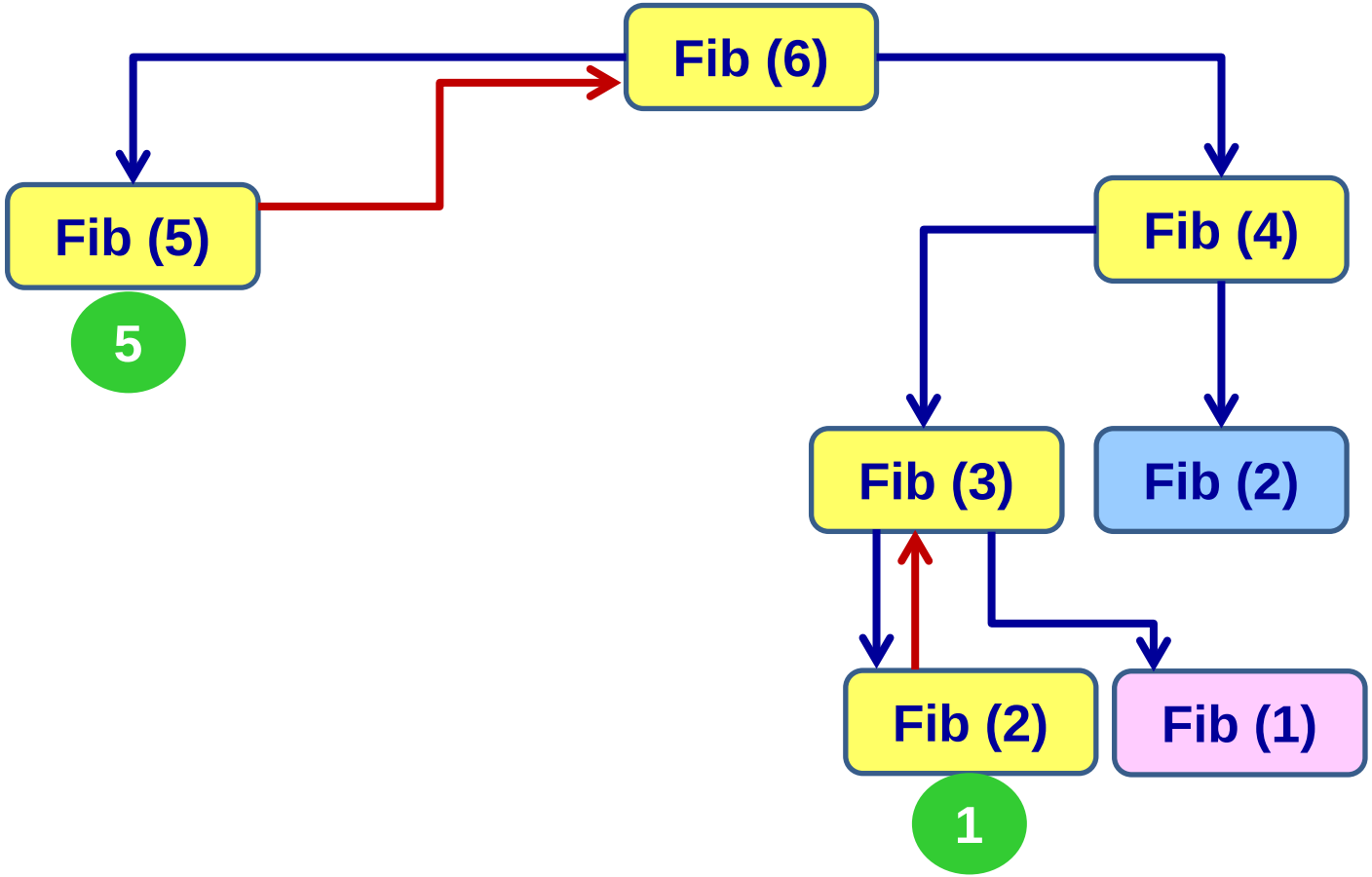
# Tipos de Recursividad: Recursividad Múltiple



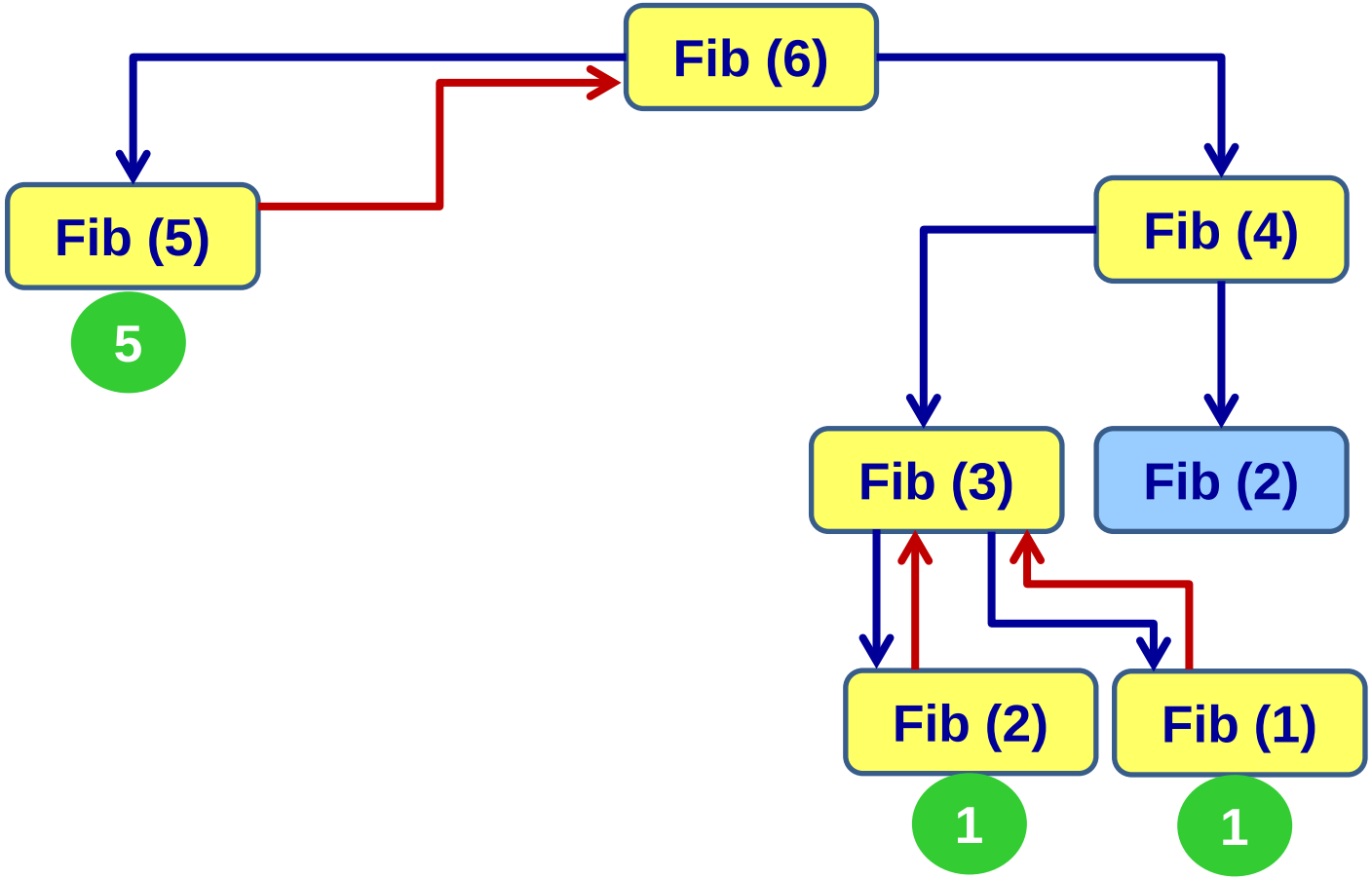
# Tipos de Recursividad: Recursividad Múltiple



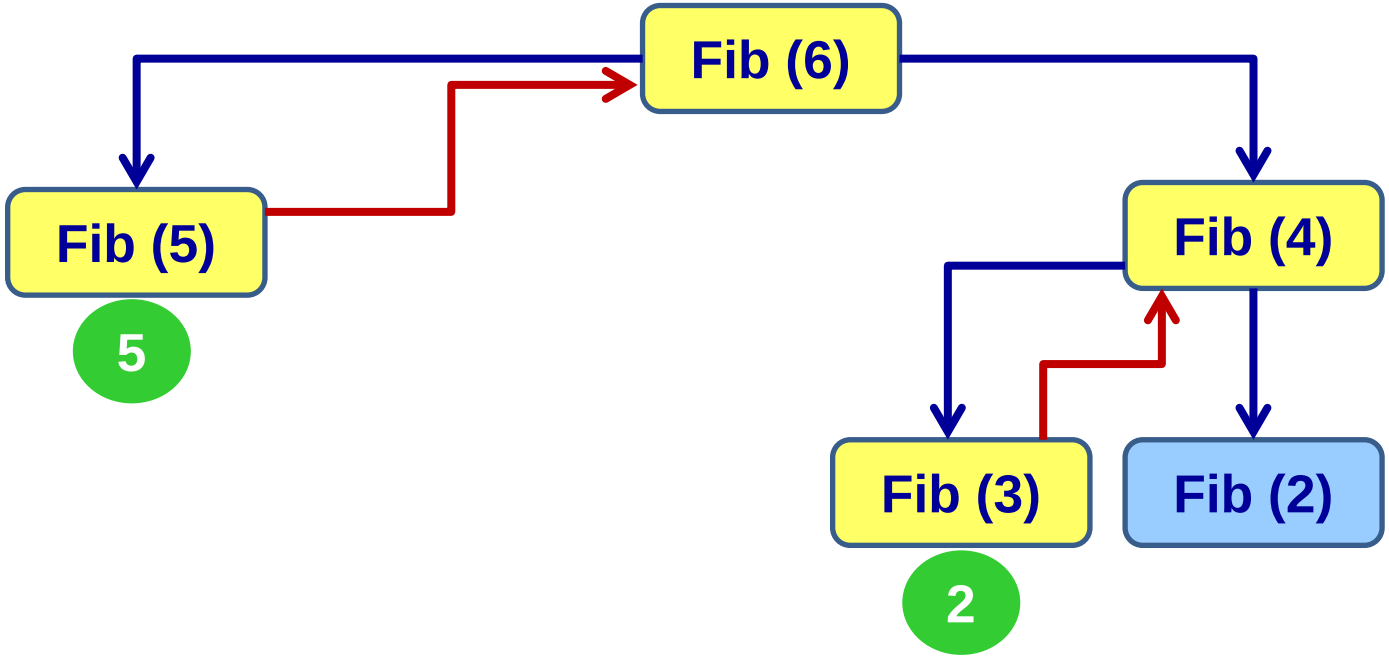
# Tipos de Recursividad: Recursividad Múltiple



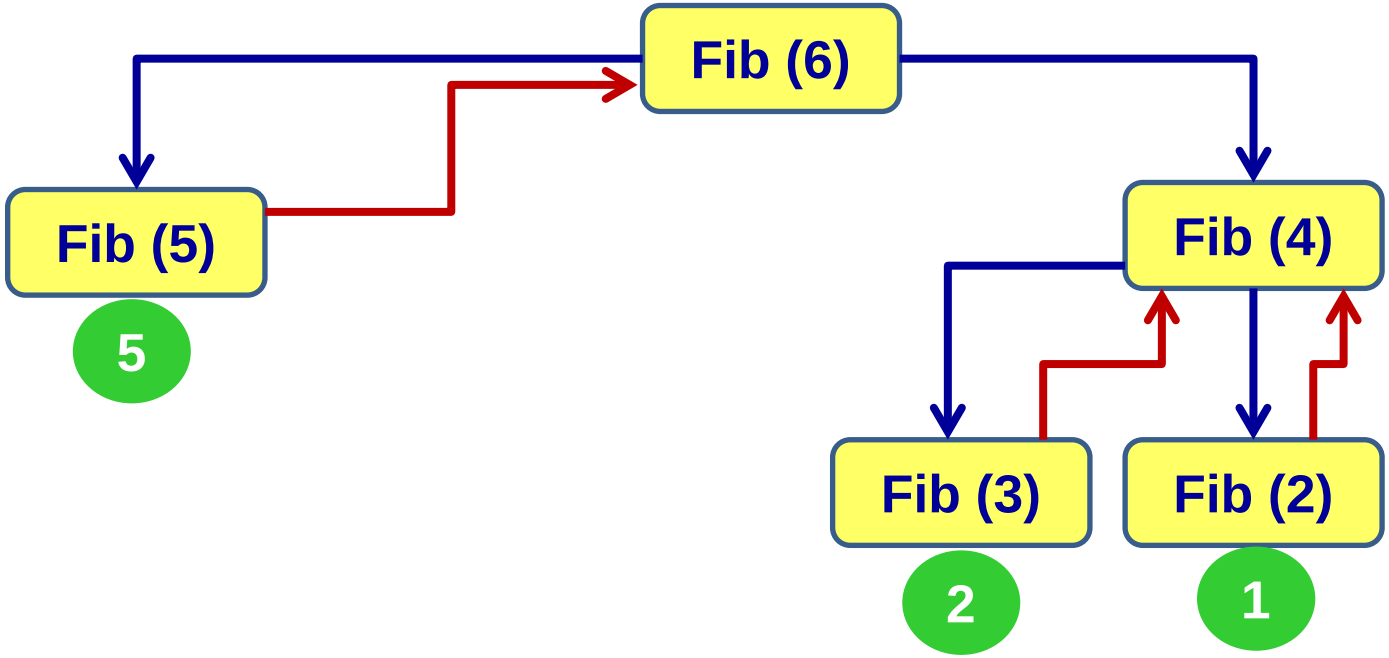
# Tipos de Recursividad: Recursividad Múltiple



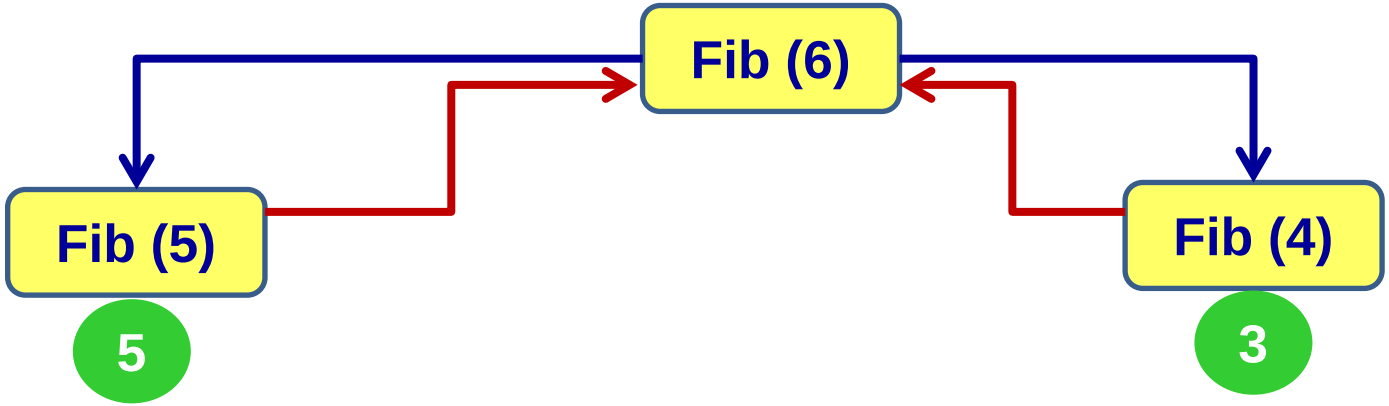
# Tipos de Recursividad: Recursividad Múltiple



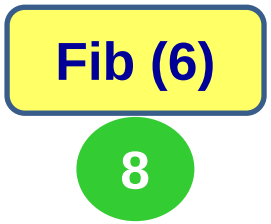
# Tipos de Recursividad: Recursividad Múltiple



# Tipos de Recursividad: Recursividad Múltiple



# Tipos de Recursividad: Recursividad Múltiple



# Tipos de Recursividad: Recursividad Múltiple

```
#include <iostream>

using namespace std;

long Fibonacci (int num);

int main()
{
    int n;
    char resp;

    do {
            cout << "\n\tIntroduza el termino que quiere calcular: ";
            cin >> n;
            cout << "\n\tEl termino " << n << " de la serie de Fibonacci es " << Fibonacci(n);

            cout << "\n\n\tDesea calcular otro termino de Fibonacci (S/N)? ";
            cin >> resp;
            resp = toupper(resp);
     } while (resp == 'S');

    cout << "\n\n\t" << endl;
    return 0;
}

long Fibonacci (int num)
{  if ((num == 1)||(num == 2)) return (1);
   else return (Fibonacci(num-1) + Fibonacci(num-2));
}
```

# Tipos de Recursividad: Recursividad Mutua

```cpp
#include <iostream>

using namespace std;

int par(int n);
int impar(int n);

int main()
{
    int n= 30;
    if (par(n))
        cout << "\n\tEl numero es par";
    else
        cout << "\n\tEl numero es impar";

    cout << "\n\n\t";
    return 0;
}

int par(int n){
    if (n==0) return 1;
    else return (impar(n-1));
}

int impar(int n){
    if (n==0) return 0;
    else return(par(n-1));
 }
```

## Recursividad Mutua:

Implica ***más*** de una función que se llaman ***mutuamente***.

## Ventajas de la recursividad:

- **Facilita** la **resolución de problemas.** Tiene una forma **sencilla** (elegante) y es **fácil** de entender.
- **Simplifica** la **complejidad de los algoritmos** y **facilita** su **verificación**.
- **Tamaño** del código **menor**.

## Inconvenientes de la recursividad:

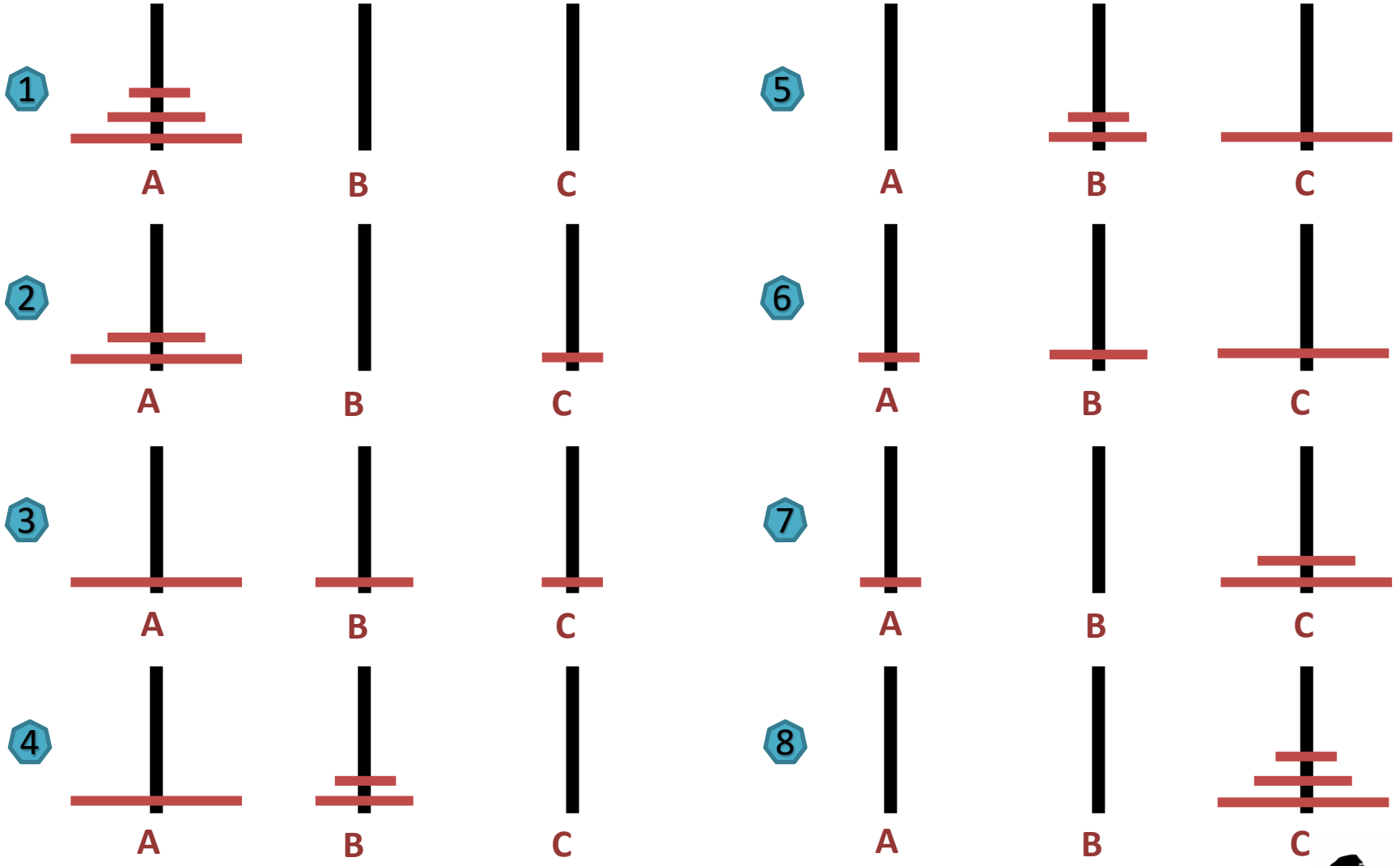
- Los **algoritmos recursivos** son **menos eficientes** que los **algoritmos iterativos**:
  - Consumen **más memoria**
  - Necesitan **más tiempo de ejecución**

# Problema de las Torres de Hanoi

El problema consiste en **mover los discos del palo izquierdo al derecho** respetando las siguientes reglas:

- Sólo se puede mover un disco cada vez.
- No se puede poner un disco encima de otro más pequeño.
- Después de un movimiento todos los discos han de estar en alguno de los tres palos.

# Problema de las Torres de Hanoi



# Problema de las Torres de Hanoi

La solución recursiva a las Torres de Hanoi se plantea como una función con 4 parámetros:

- El número de discos (N)
- Palo origen (A)
- Palo destino (C)
- Palo auxiliar (B)

La solución consiste en leer por teclado un valor N, e imprimir la secuencia de pasos/movimientos para resolver el problema:

- Si N=1 mueva el disco de A a C y parar.
- En caso contrario:
  - ✓ **Mueva los N-1** discos superiores de A a B, con C auxiliar
  - ✓ Mueva el disco restante de A a C
  - ✓ **Mueva los N-1** discos de B a C, usando A como auxiliar

# Problema de las Torres de Hanoi

```
#include <iostream>

using namespace std;

void Mueve(int N,char origen, char destino, char auxiliar);

int main()
{
    int discos;

    do{
    cout << "\n\n\tIndique el numero de discos (valor > 0): ";
    cin >> discos;
    } while (discos <= 0);

    Mueve (discos, 'A', 'C', 'B');


    cout << "\n\n\t";
    return 0;
}


void Mueve(int N,char origen,char destino, char auxiliar)
{
    if (N==1) cout << "\n\n\tSe mueve un disco desde " << origen << " hacia " << destino;
    else
    { Mueve(N-1,origen,auxiliar,destino);
      cout << "\n\n\tSe mueve un disco desde " << origen << " hacia " << destino;
      Mueve(N-1,auxiliar,destino,origen);
    }
}
```