# Dominando la Recursividad

Estructuras de control avanzado y gestión de memoria.

# Toda función recursiva opera bajo dos principios innegociables

FUNCTION BLOCK

FUNCTION BLOCK

CONDITIONAL LOGIC

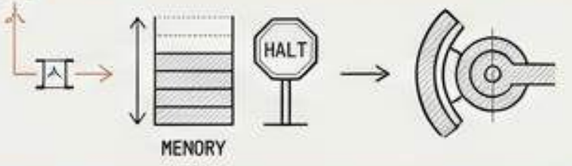
```
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```

RECURSIVE CALL

DETAIL

```
if (n == 0) {
    return 1;
}
```

**CASO BASE:** El freno de emergencia. Detiene la recursión y evita el colapso de la memoria.

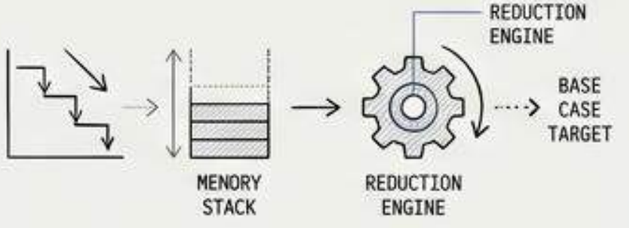



DETAIL

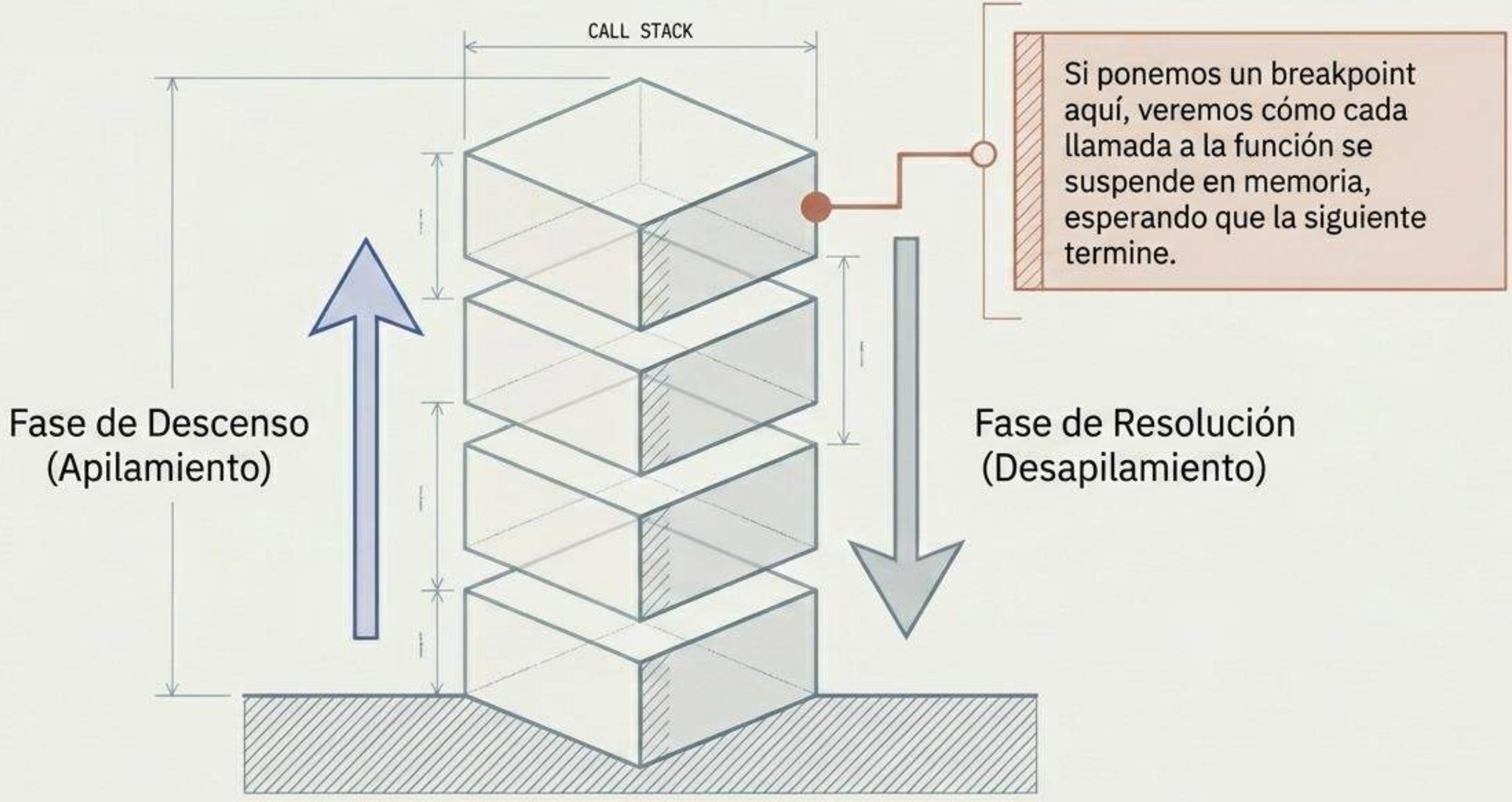
```
return n * factorial(n - 1);
```

STRUCTURAL ANALYSIS - RECURSIVE LOGIC

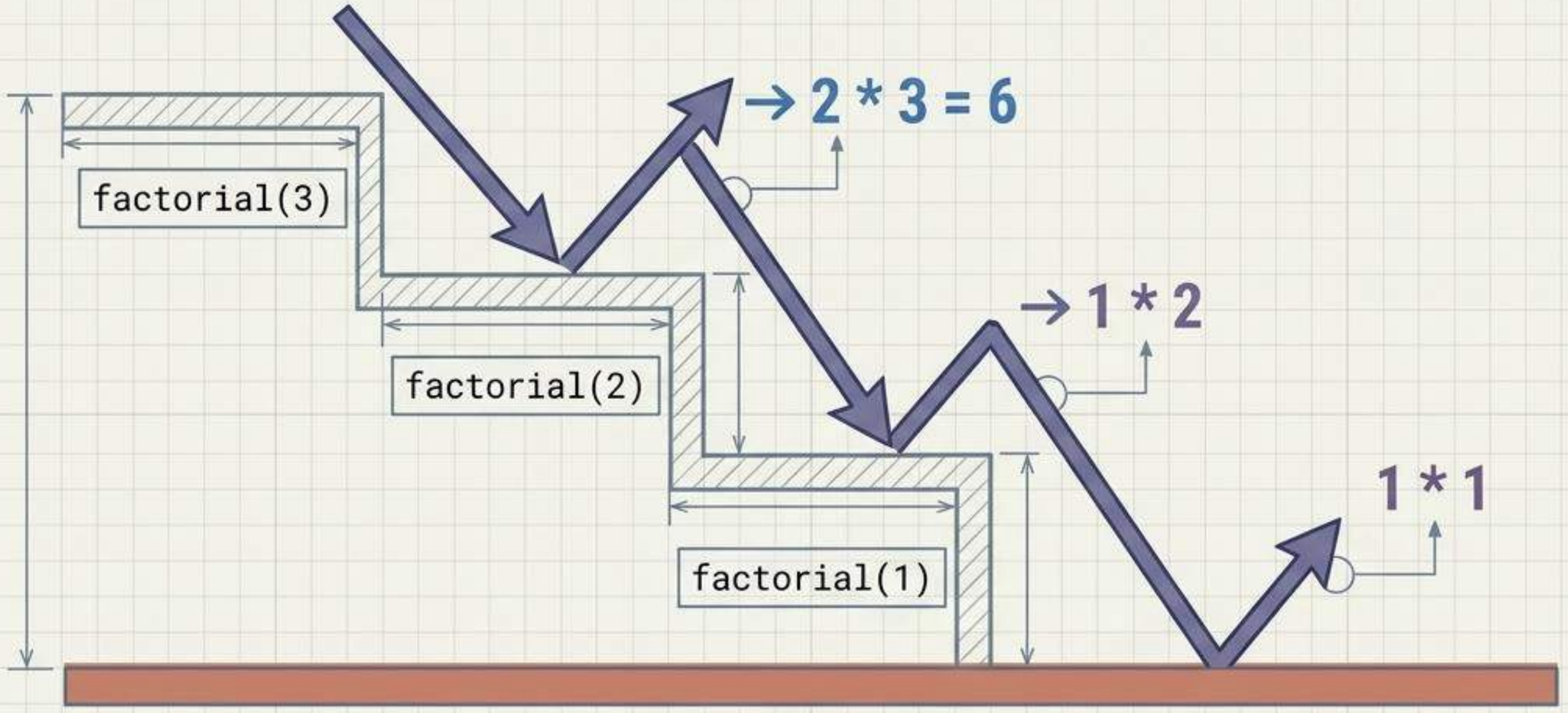
**CASO RECURSIVO:** El motor. Reduce el problema original, acercándolo forzosamente un paso más hacia el caso base.



# La ejecución ocurre dentro de una pila de llamadas temporal



# La recursividad lineal desciende un nivel a la vez hasta rebotar en un límite sólido

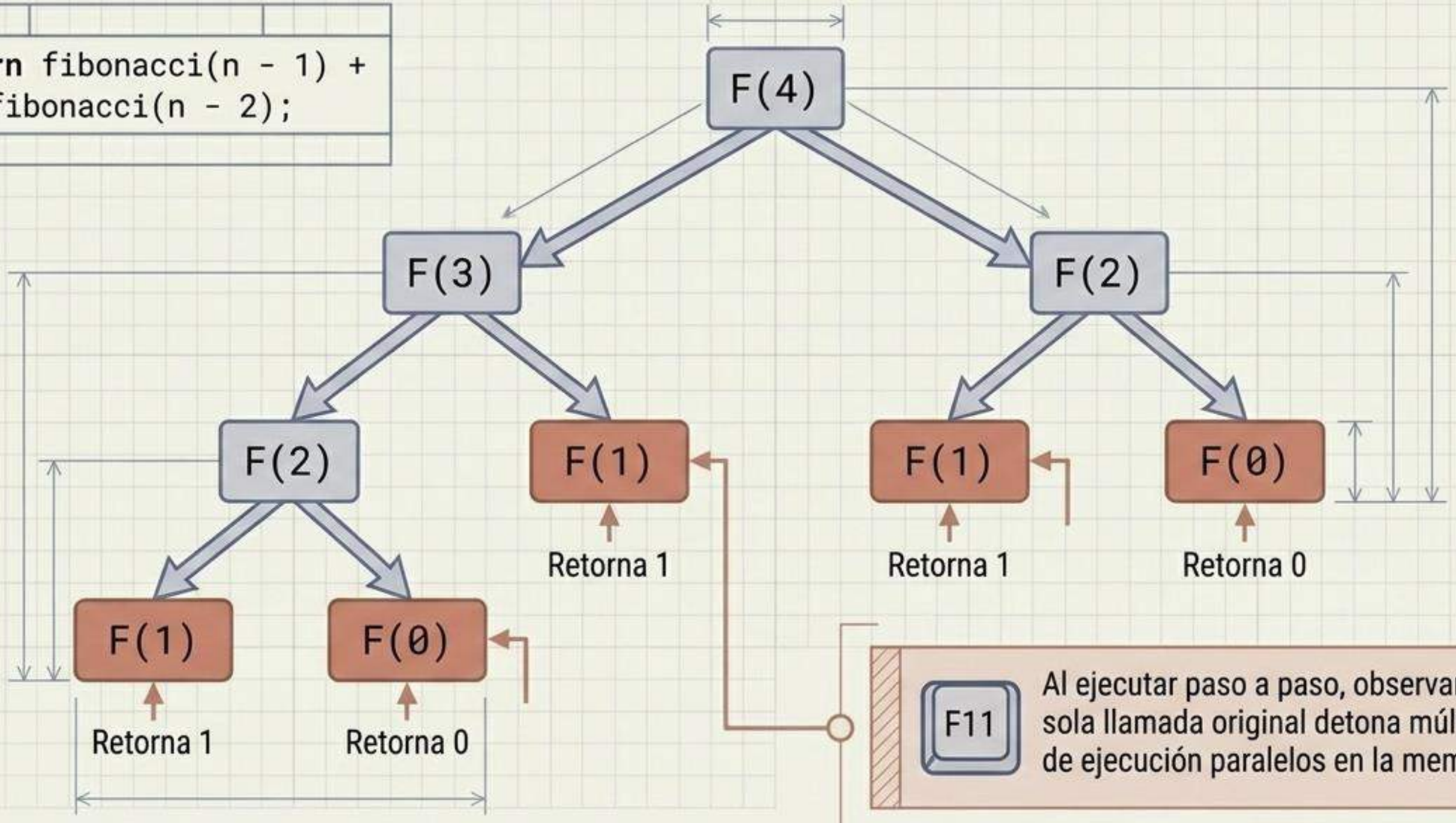



**CASO BASE: n == 0 (Retorna 1)**

**Ejemplo Clásico:** Factorial.
Una sola llamada recursiva por ejecución.

# La recursividad múltiple genera ramificaciones exponenciales en el árbol de ejecución

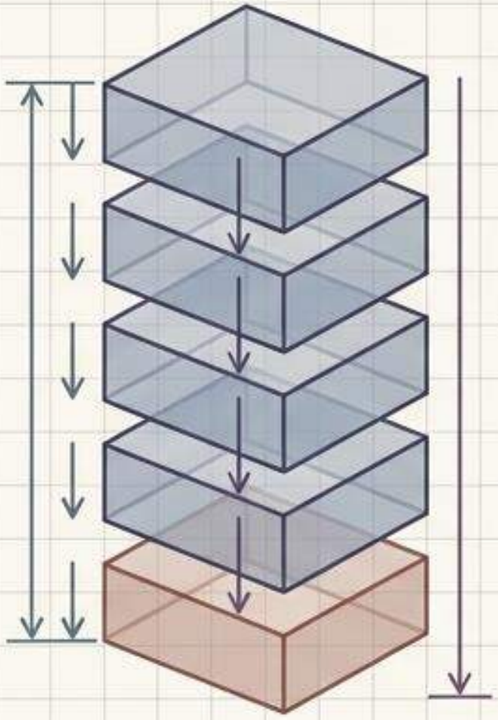
```
return fibonacci(n - 1) +
    fibonacci(n - 2);
```



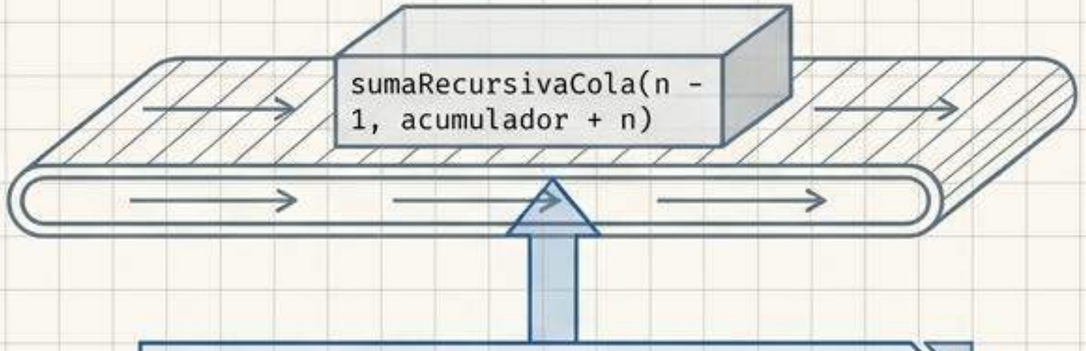

F11

Al ejecutar paso a paso, observamos cómo una sola llamada original detona múltiples caminos de ejecución paralelos en la memoria.

# La recursividad de cola optimiza la memoria al no dejar operaciones pendientes

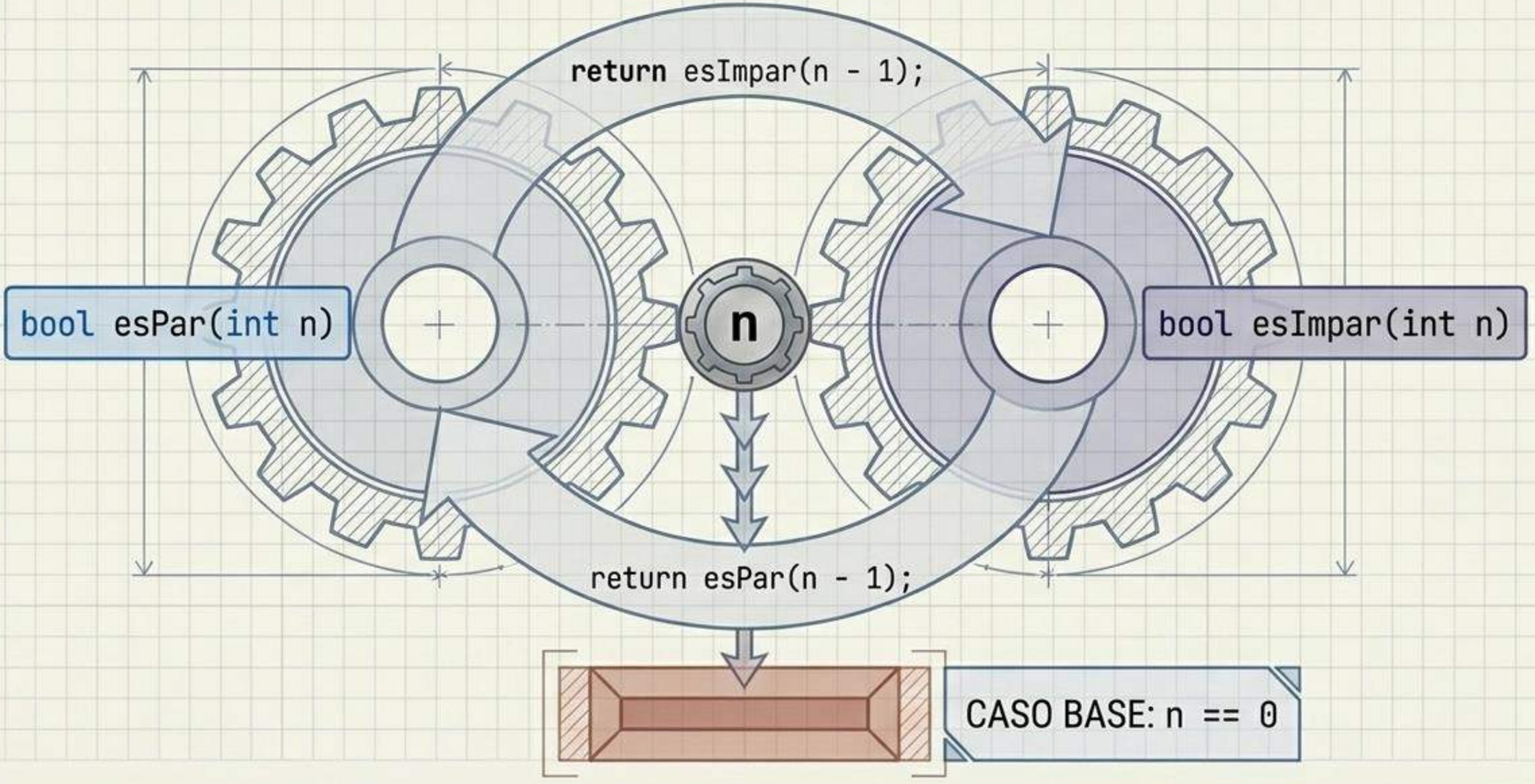


**Recursividad Clásica:** La memoria retiene el estado porque hay multiplicaciones pendientes después de la llamada.




**El Acumulador:** Al pasar el resultado parcial como parámetro, la función actual puede destruirse. Destouirse. No queda nada pendiente en el escritorio.

# La recursividad indirecta delega la ejecución mediante un ciclo mutuo controlado



# Matriz de topología recursiva

| Tipo | Estructura de la Pila | Complejidad Espacial | Ejemplo Clásico |
|---|---|---|---|
| Lineal | Profundidad Lineal | Alta (O(n)) | Factorial |
| Múltiple | Árbol Exponencial | Muy Alta | Fibonacci |
| Cola (Tail) | O(1) Optimizado / Plana | Baja (Sin estado pendiente) | Suma con acumulador |
| Indirecta | Ciclo de delegación | Alta (O(n)) | Validación Par/Impar |

# Reglas de oro para la arquitectura recursiva

01

## Diseña desde el final.

Define siempre tu Caso Base antes de escribir una sola línea de lógica recursiva. Es el único seguro contra un desbordamiento de memoria (Stack Overflow).

02

## Optimiza el estado.

Si la profundidad de la recursión será masiva, utiliza un parámetro acumulador (Recursividad de Cola) para liberar la pila de llamadas.

03

## Visualiza la topología.

Antes de codificar, dibuja mentalmente si el algoritmo requiere un descenso lineal, un ciclo delegado o un árbol árbol ramificado.