

Optimización de Código

Conceptos, Tipos y Estrategias

Una guía práctica para mejorar la eficiencia sin comprometer la funcionalidad.

```
1 def optimize_process(data):
2     if data.is_valid():
3         result = process(data) # initial
4         -/- + -
5     result = optimized_process(data) // faster!
6 else:
7     handle_error()
8 ...
9 return result
```

Basado en los principios de rendimiento de software.

La Regla de Oro: Velocidad ≠ Cambio de Lógica

Optimizar no es "hacer el código más corto". Es reducir recursos manteniendo el comportamiento intacto.



Advertencia: Si el resultado cambia (ej. el precio final varía), no es optimización; es un cambio de lógica.

Los 4 Pilares de la Optimización Web



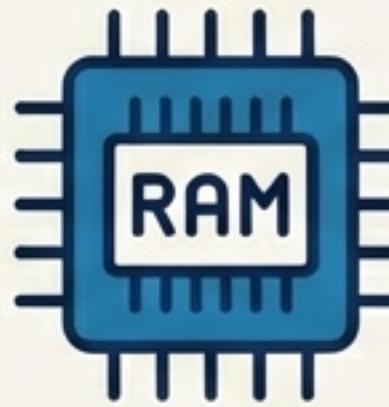
Tiempo de Respuesta

Carga de páginas, latencia de APIs.



Base de Datos

Cantidad de consultas, peso de las queries, datos repetidos.



Memoria

Carga de objetos enormes, Garbage Collection, datos no usados.



Red

Tamaño de payloads, transferencias innecesarias.

Tip: La mejora más rentable suele ser reducir las consultas a base de datos.

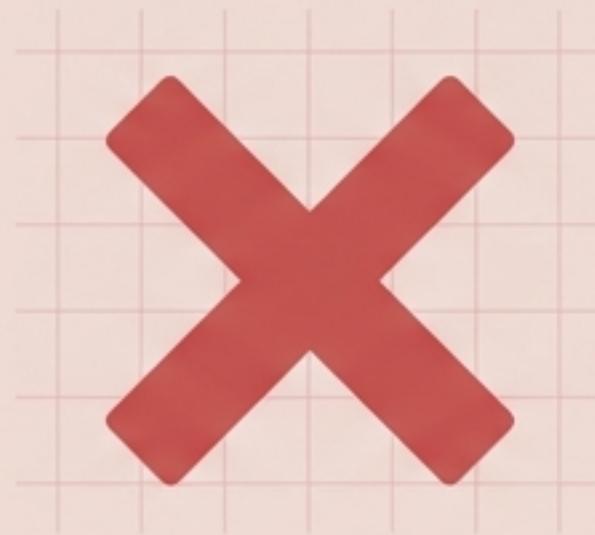
Estrategia: ¿Cuándo conviene optimizar?

SÍ OPTIMIZAR (Proceder)



- **Lentitud real** (usuarios esperando).
- **Cuello de botella** en Base de Datos.
- El servidor degrada con concurrencia.
- **Código crítico** en cada petición.

NO OBSESIONARSE (Detener)

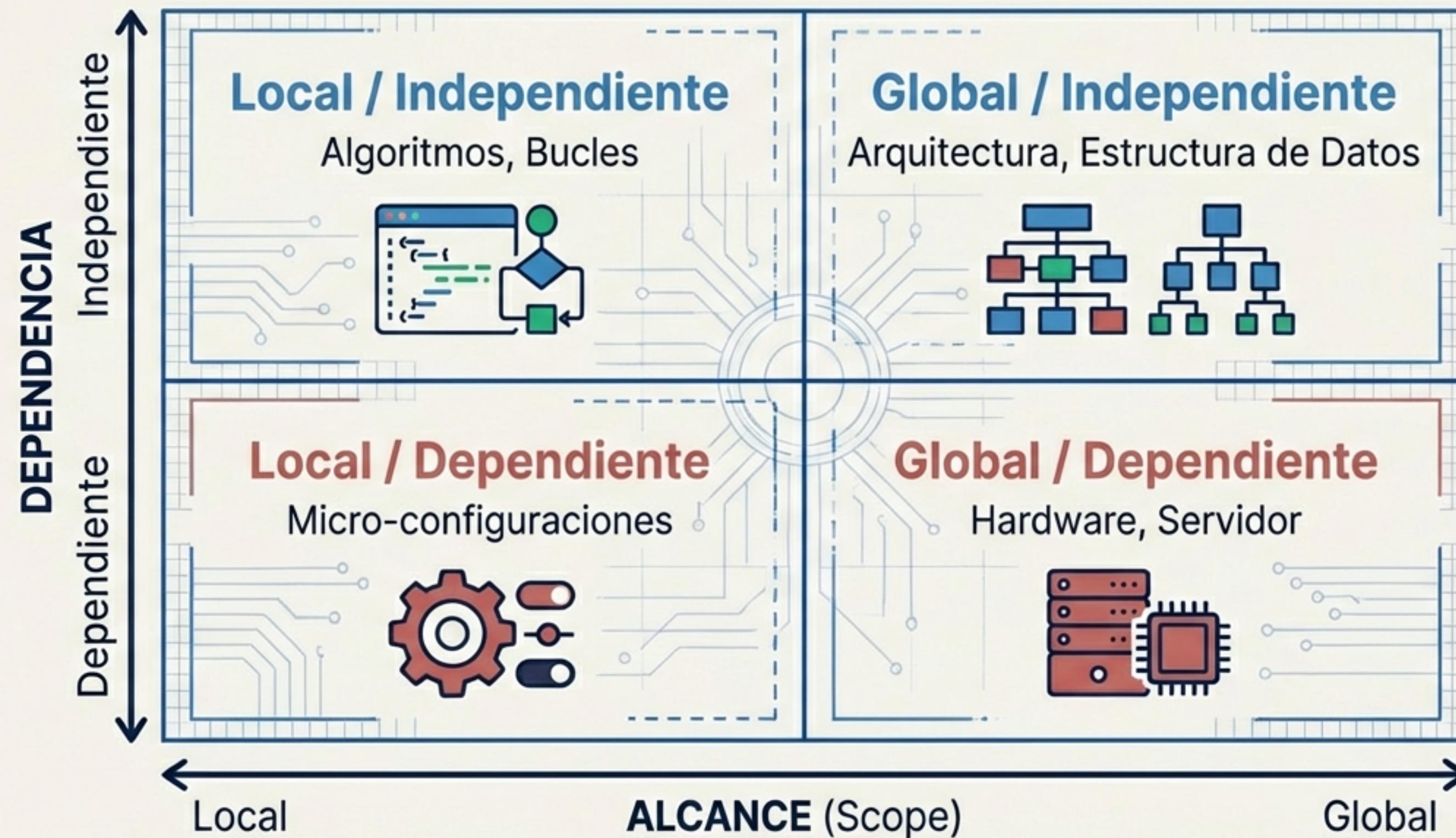


- El proyecto es **pequeño** y fluido.
- La optimización complica la lectura del código.
- **Premature optimization is the root of all evil.**

El Ciclo de Vida de la Optimización



Clasificación: La Matriz de Optimización



1. Optimización Local (Micro)

Definición: Mejoras que afectan a una parte pequeña del código.

- Se aplica a una función o bucle concreto.
- Rápida de implementar.
- Mantiene el resultado reduciendo el trabajo.



Ideal para: Arreglar bucles infinitos o fugas de memoria.

Caso Práctico Local: Salida Temprana (Early Exit)

✗ Ineficiente

```
for (String rol : roles) {  
    if ("ADMIN".equals(rol)) {  
        encontrado = true;  
        // Sigue recorriendo...  
    }  
}  
return encontrado;
```

✓ Optimizado

```
for (String rol : roles) {  
    if ("ADMIN".equals(rol)) {  
        return true; // Salida inmediata  
    }  
}  
return false;
```

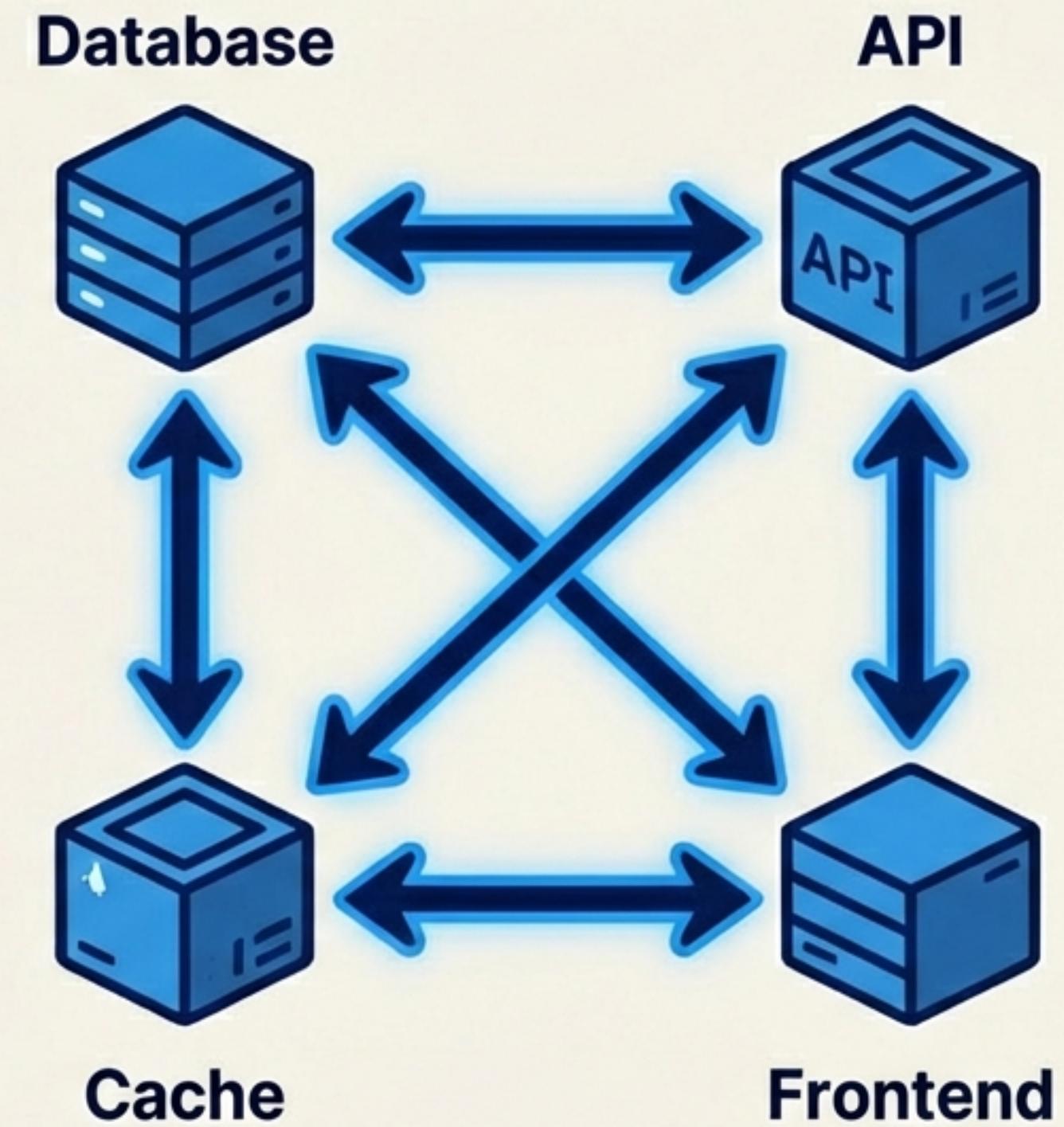


En cuanto lo encuentra, sale.
No consume más CPU.

2. Optimización Global (Macro)

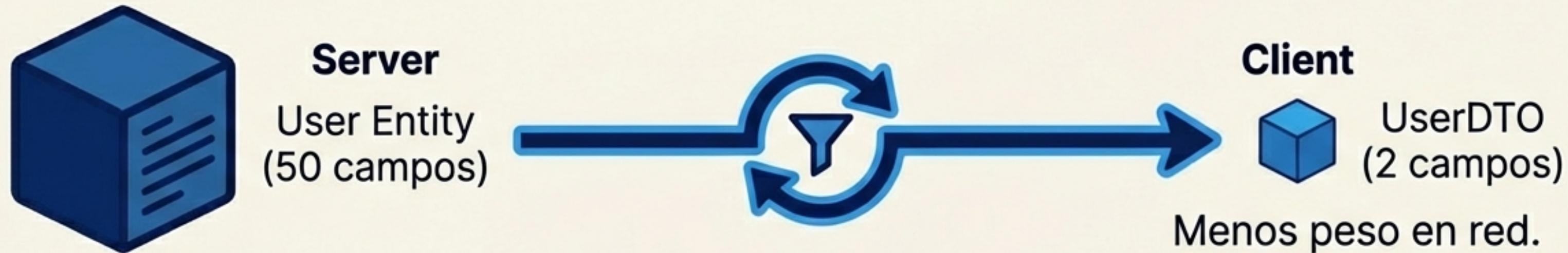
Definición: Mejoras que afectan al sistema como conjunto.

- Impacto alto en el rendimiento general.
- Toca múltiples módulos del programa (Acceso a datos, flujos).
- Requiere mayor verificación (Testing de regresión).

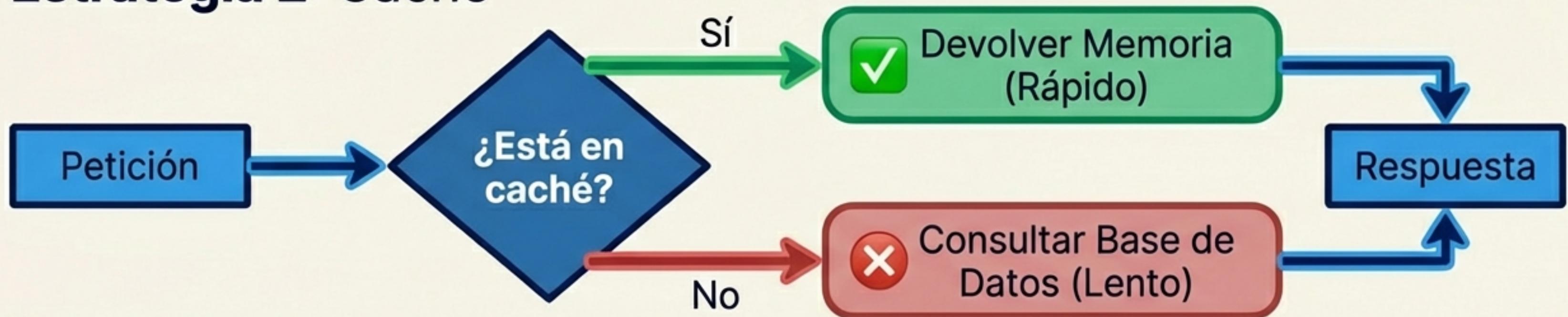


Estrategias Globales: DTOs y Caché

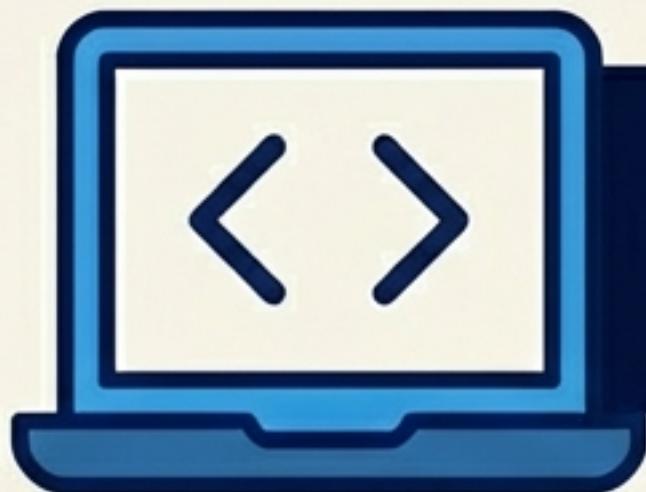
Estrategia 1: Data Transfer Objects (DTO)



Estrategia 2: Caché

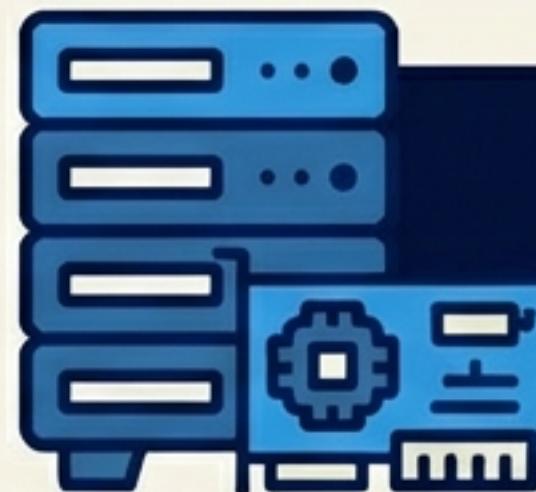


3. Independiente vs. Dependiente de la Máquina



Independiente
(Prioridad)

- **Qué es:** Lógica, algoritmos, diseño.
- **Ejemplos:** Paginación, DTOs, evitar loops anidados.
- **Verdad:** Funciona igual en local que en producción.



Dependiente
(Entorno)

- **Qué es:** Configuración, RAM, JVM flags.
- **Ejemplos:** Aumentar memoria, tunear servidor web.
- **Verdad:** El efecto varía según el hardware.

Prioriza siempre la optimización independiente (código) primero.

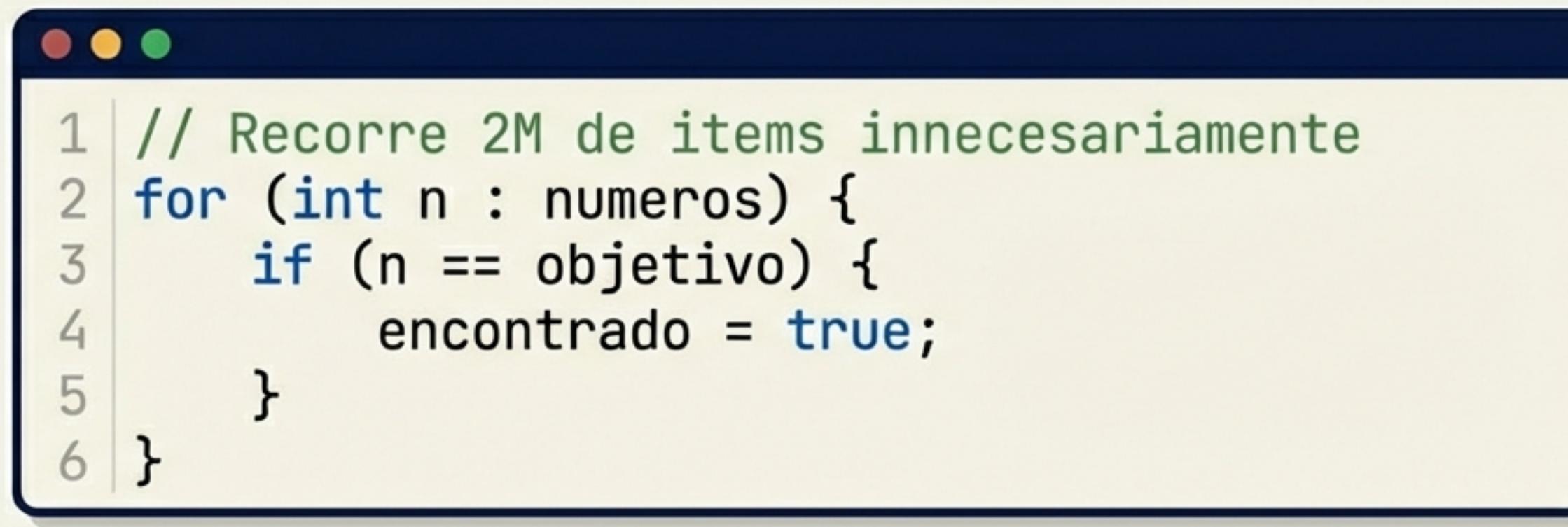
Checklist de Decisión

- ¿El problema es local (bucle) o global (arquitectura)?
- ¿Puedo reducir consultas/red antes de tocar micro-código?
- ¿Es una optimización independiente (lógica)?
- ¿El cambio es pequeño y verificable?
- CRÍTICO:** ¿He verificado que el resultado final es idéntico?



Desafío: Optimiza este Bucle

Contexto: Buscamos el número 10 en una lista de 2 millones de registros. El 10 está al principio.



```
1 // Recorre 2M de items innecesariamente
2 for (int n : numeros) {
3     if (n == objetivo) {
4         encontrado = true;
5     }
6 }
```

Pregunta

¿Cómo harías que este código sea instantáneo?

Solución

Añadir "break;" dentro del if.

Resumen y Puntos Clave

01

Comportamiento Intacto

Si el resultado cambia,
no es optimización.

02

Mide, no adivines

Detecta el cuello de
botella antes de tocar
código.

03

Lógica sobre Hardware

Mejora el algoritmo
antes de añadir más
RAM.