Programação Avançada

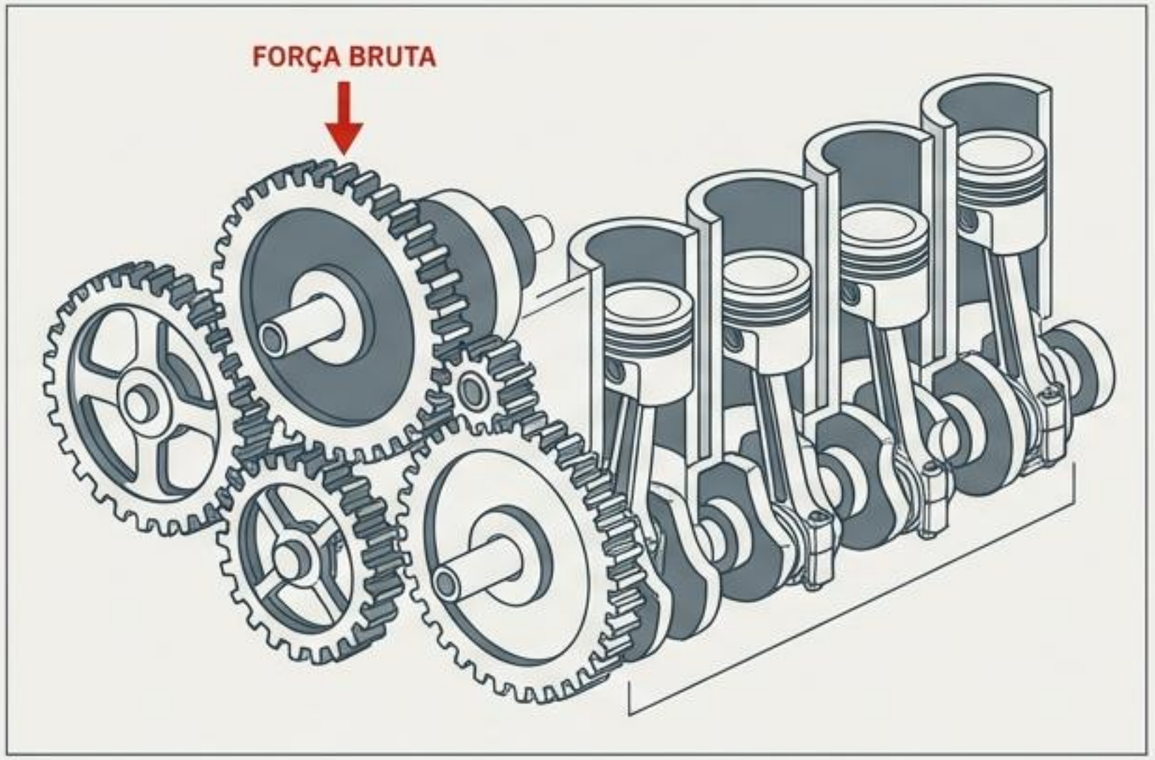
# TOPIC 3: ASYNC PROGRAMMING VS MULTITHREADING FOR WEB REQUESTS

Alunos: Rui Duarte, nº 190200190
Laysa Siqueira, nº 220000005

Janeiro, 2026

# O Desafio da Concorrência Web: Bloqueio vs. Fluxo

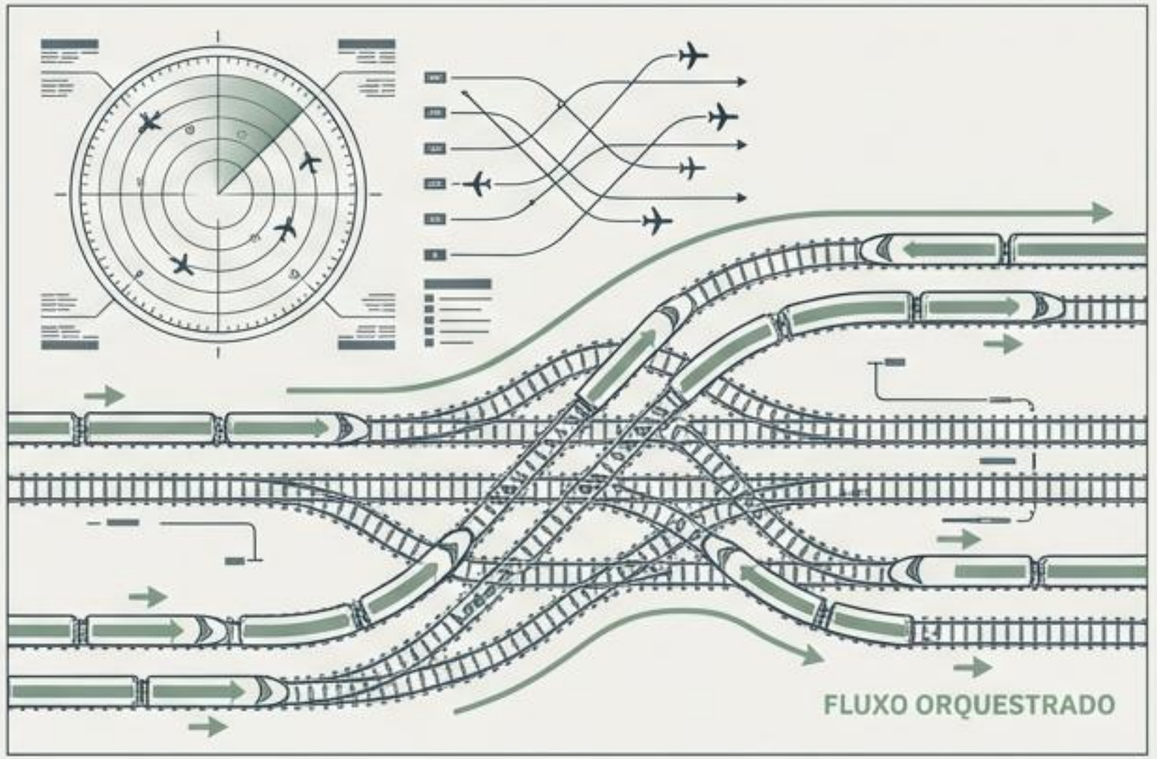
Uma análise comparativa entre Multithreading e Programação Assíncrona em Python.




## MULTITHREADING

⚙ Gestão: Sistema Operativo

⇄ Mecanismo: Context Switching (**Força Bruta**)




## PROGRAMAÇÃO ASSÍNCRONA

⏱ Gestão: Event Loop

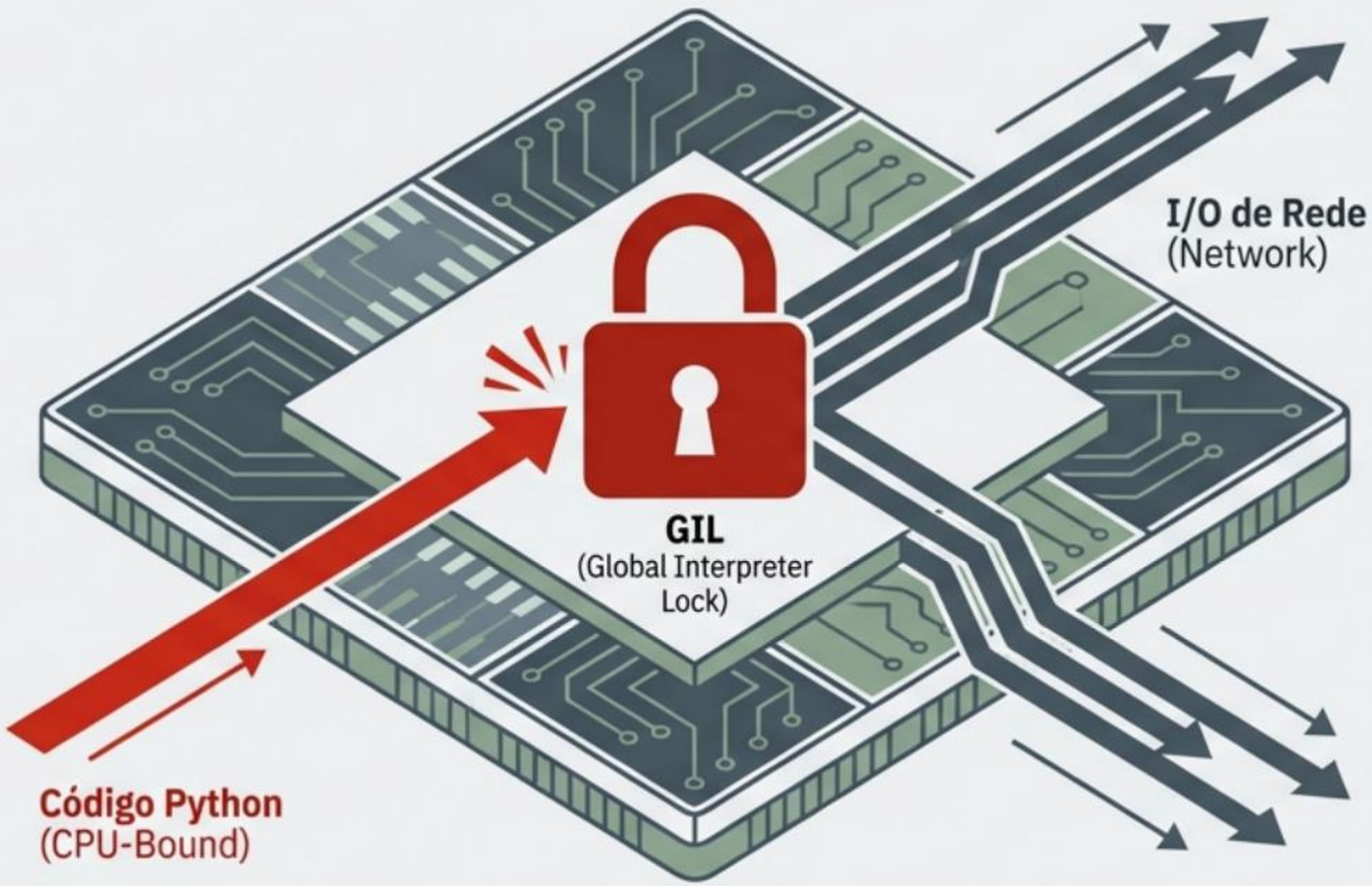
⇄ Mecanismo: Suspensão Cooperativa (**Fluxo**)

**Contexto:** A batalha pelo processamento eficiente de 1700 pedidos HTTP em ambiente controlado.

# O Problema: O Dilema do GIL e a Ilusão do Paralelismo

## O OBSTACULO

OGIL impede a execução simultânea de código Python puro em múltiplas threads, limitando o paralelismo em tarefas de CPU.




## A OPORTUNIDADE

Em tarefas de I/0 (rede), o GIL é libertado.
A questão central:

OMultithreading tradicional consegue competir com a eficiência moderna do Asyncio apesar do peso do Sistema Operativo?

Objetivo do Estudo: Comparar empiricamente o desempenho (Tempo vs. Recursos) em 1700 pedidos concorrentes.

# Metodologia: O Laboratório (Ambiente Experimental)

Isolamento de variáveis para garantir justiça científica.

## INFRAESTRUTURA

17 Contentores Docker Locais

Simulação de API pública sem latência de internet real.

## CARGA TOTAL

1700 Pedidos HTTP

Volume suficiente para stressar o interpretador.

## SIMULAÇÃO

Delay Artificial: 1.5s

Imita o tempo de processamento real de um servidor.

## CONTROLO

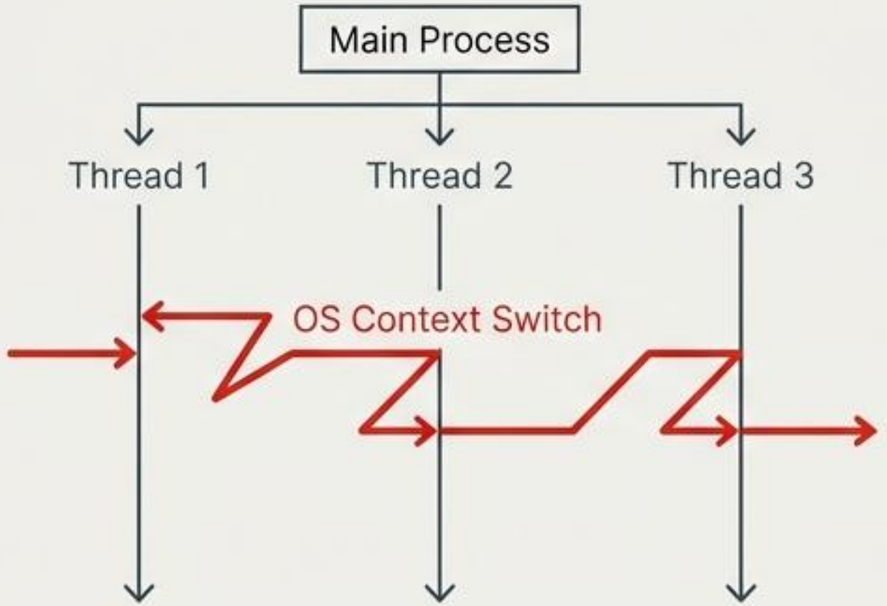
Concorrência Máxima: 24

Limite fixo para Threads e Semáforo Async.

Nota: O uso de ambiente local elimina a variabilidade da rede externa (jitter), focando o teste puramente na eficiência do código Python.

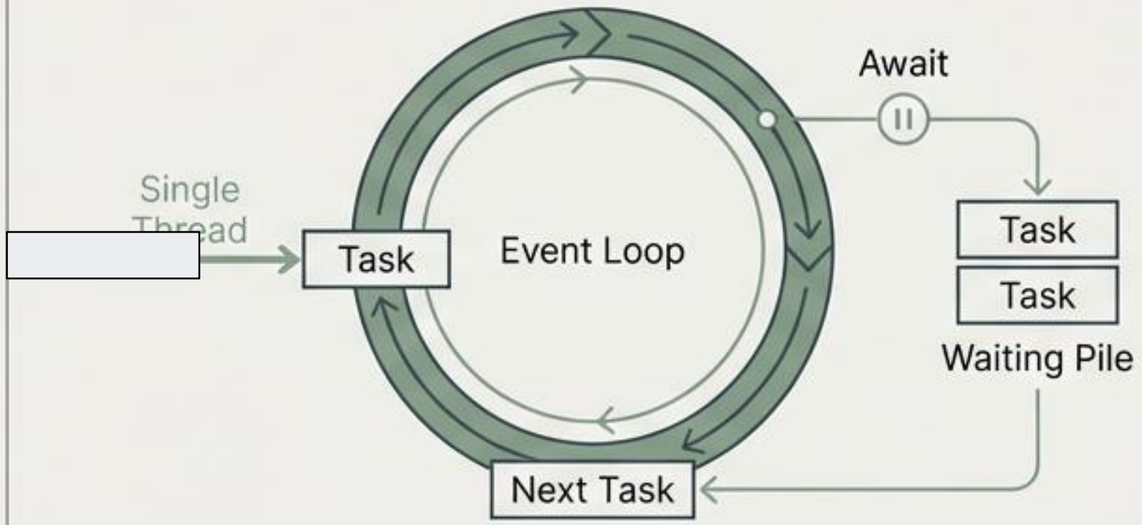
# Metodologia: O Duelo Arquitetural

## MULTITHREADING (OS MANAGED)




ThreadPoolExecutor: O Sistema Operativo força a alternância. Overhead elevado.

## ASYNCIO (EVENT LOOP)

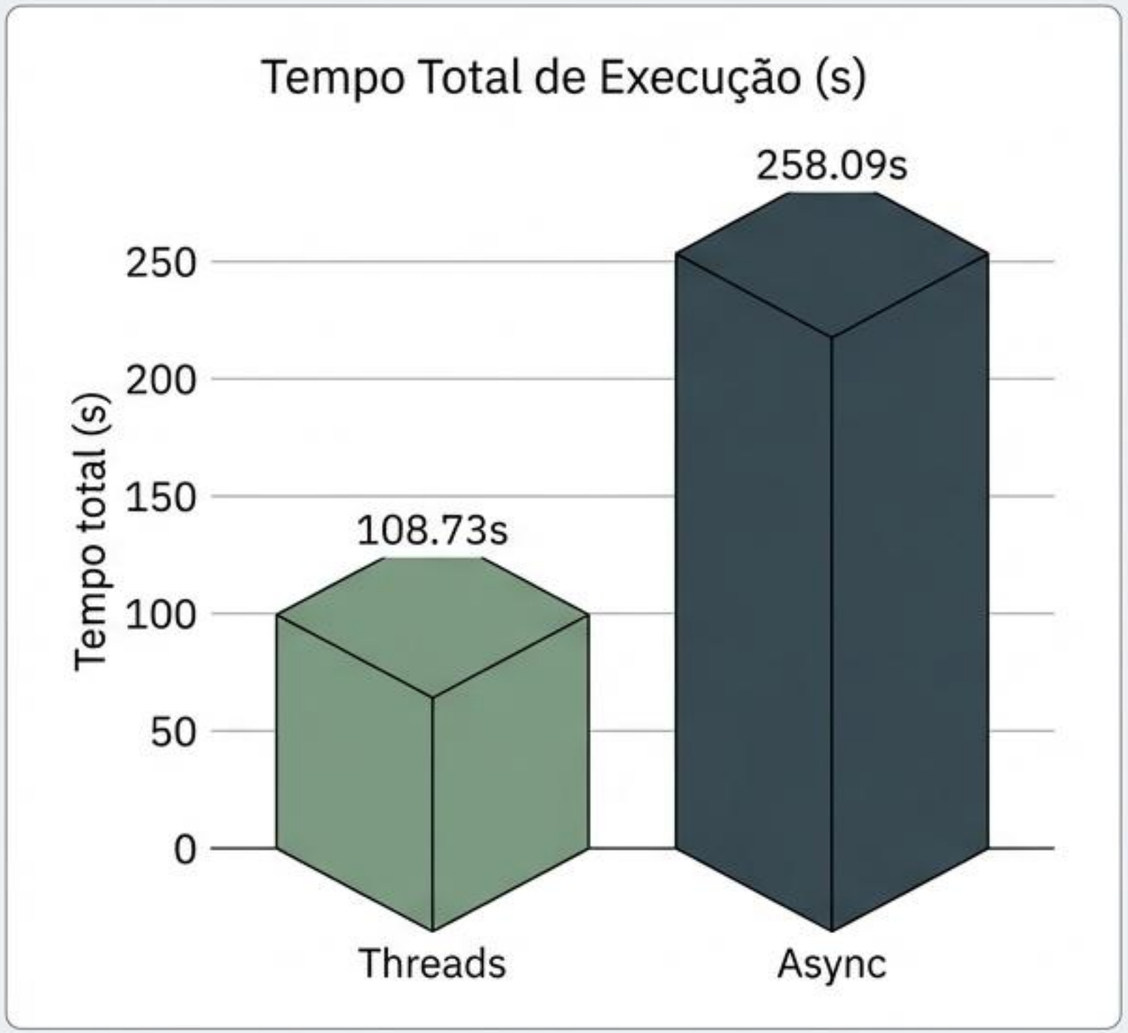



Cooperative Multitasking: O código cede controlo voluntariamente. Overhead mínimo.

Ambas as arquiteturas limitadas a 24 tarefas ativas simultâneas.

# Resultados: Velocidade Bruta (Tempo de Execução)

## Comparação de Desempenho Direto




## O VENCEDOR: MULTITHREADING

## 137.4% Mais Rápido

Contra todas as expectativas, a tecnologia “mais antiga” venceu em velocidade pura. O agendamento agressivo do Sistema Operativo e a otimização da biblioteca `requests` superaram os micro-atrasos de estabilidade necessários no loop assíncrono.

**Threads: 108.73s**

**Async: 258.09s**

# **Resultados:** Eficiência de CPU (O Custo do Processamento)

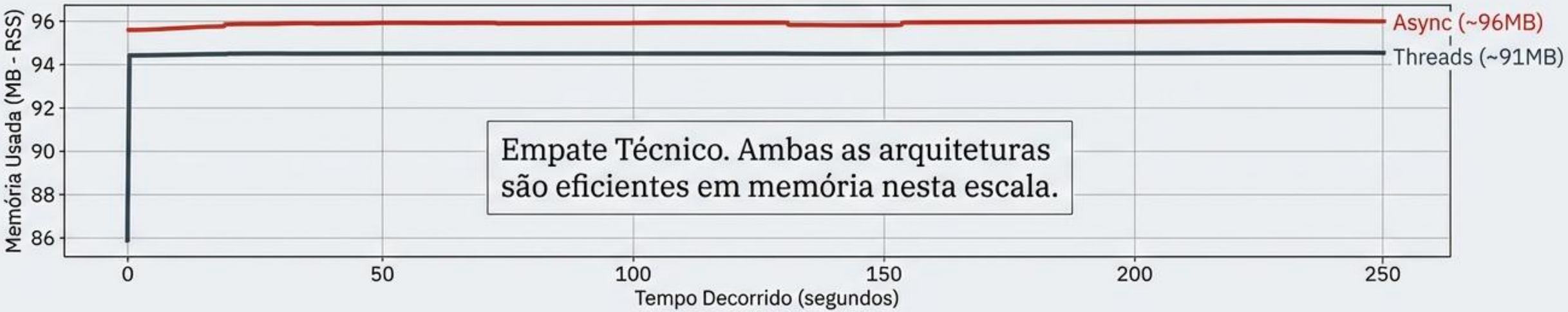
Comparação de Desempenho Direto


**A REDENÇÃO DO ASYNC**

Enquanto as Threads ganham no tempo, o Async ganha na sustentabilidade. O modelo assíncrono realiza o trabalho com uma fração do “esforço” computacional, evitando o custo violento do Context Switching.

# Resultados: Memória e Fiabilidade

## Memória (RAM)




## Taxa de Sucesso (Fiabilidade)

**100%**

**Multithreading**

Robustez imediata (Out-of-the-box).

**99.12%**

**Async**

Ligeira perda de pacotes. Requer afinação do TCPConnector em cargas altas.

# Conclusões Práticas: O Veredicto

## Quando utilizar cada arquitetura?

| CENÁRIO A: VELOCIDADE E AUTOMAÇÃO | CENÁRIO B: ESCALA MASSIVA E EFICIÊNCIA |
| --- | --- |
|  |  |
| **Vencedor: Multithreading** | **Vencedor: Async (Assíncrono)** |
| ➢ **Use quando:** Precisa de throughput máximo em escala moderada e tem CPU disponível.<br>➢ **Ideal para:** Scripts de automação, scraping rápido, ferramentas internas. | ➢ **Use quando:** O hardware é limitado ou precisa de suportar 10.000+ conexões.<br>➢ **Ideal para:** Backends de APIs de alto tráfego, micro-serviços, aplicações realtime. |

> ***"O Multithreading ganha no cronómetro; o Async ganha na eficiência."***

# Trabalho Futuro: Para Além do Horizonte

### Escala Extrema

Testar 10.000+ pedidos (Ponto de rutura da memória das Threads).

### Ambiente Cloud

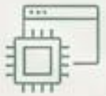
Introduzir latência de rede real (AWS/GCP) e Jitter.

### Novas Bibliotecas

Comparação com HTTPX e frameworks modernos.

### Cargas Mistas

Híbrido: Processamento pesado de CPU + Pedidos Web.



IBM Plex Serif

A escolha do modelo de concorrência é uma decisão arquitetural crítica que define a capacidade de crescimento da sua aplicação.