



# Processos e Threads

Implementação e exemplos em C, Java, Python e Go

Por: Paulo Ricardo, Eros Ryan, Breno Gomes e Francisco Antonio

# Temas

Características e  
peculiaridades da implementação  
de processos e threads em:

1

C

2

Java

3

Python

4

Go

c



# Processos

- Cada processo tem seu próprio espaço de endereçamento
- Comunicação entre processos requer mecanismos como pipes, sinais, memória compartilhada ou sockets
- Criar processos é mais "pesado" para o sistema

# Threads

- Compartilham memória e recursos do processo pai
- São mais leves que processos (criação e troca de contexto)
- Requer sincronização (mutexes, semáforos) para acesso a recursos compartilhados

# Características e Peculiaridades



Processos e Threads  
têm uma biblioteca  
própria

`unistd.h` e `pthread.h`

Oferece acesso direto às  
chamadas de sistema do SO  
(`fork()`,  
`pthread_create()`)

Precisa gerenciar  
manualmente a memória  
compartilhada entre  
threads/processos  
(Sem GC)

Precisa usar primitivas  
como `mutex`, `semáforos`  
manualmente

Multiprocessamento (`fork`)  
e `multithreading`  
(`pthreads`)

Java



Java

# Processos

## Implementação

- A própria aplicação Java roda como um **processo JVM**, criado pelo sistema operacional;
- Java pode criar processos externos com **ProcessBuilder** ou **Runtime.exec()**.

## Sincronização

- Não há sincronização automática;
- Usa-se **waitFor()** para aguardar fim do processo externo.
- Controle manual com **destroy()**, **exit code**, e **sleep**.




# Processos

## Comunicação

- Comunicação com processos externos ocorre via:
  - **Entrada padrão (stdin)** - envia dados ao processo.
  - **Saída padrão (stdout)** - lê dados do processo.
  - **Erro padrão (stderr)** - lê mensagens de erro.
- Dentro da JVM, usa-se comunicação entre threads (não entre processos).



# Características e Peculiaridades

- 
- Cada processo possui espaço de memória separado
  - Comunicação é mais lenta e complexa
  - Pesado em termos de recursos (CPU, memória)
  - Ideal para automações, integração com SO ou scripts externos

# Threads

## Implementação

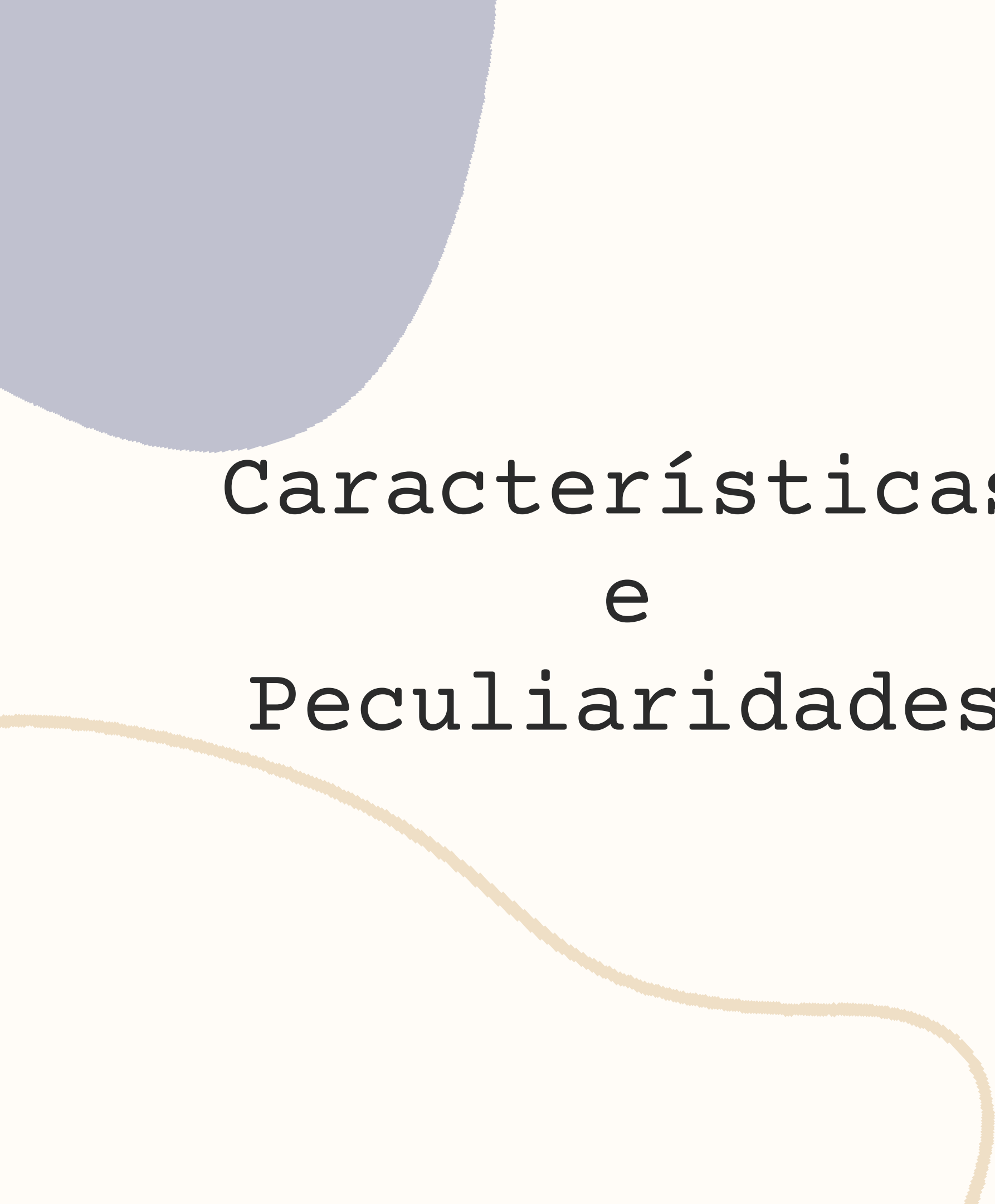
- Criadas via:
  - Herança de **Thread**;
  - Implementação de **Runnable**;
  - Uso de **ExecutorService**, **Callable**, **Future** (moderno);
- Permite multitarefa dentro da JVM

## Sincronização


- Via palavras-chave (**synchronized**, **volatile**);
- Uso de APIs: **Semaphore**, **Lock**, **CountDownLatch**, **CyclicBarrier**;
- Evita condições de corrida (race conditions);

## Comunicação

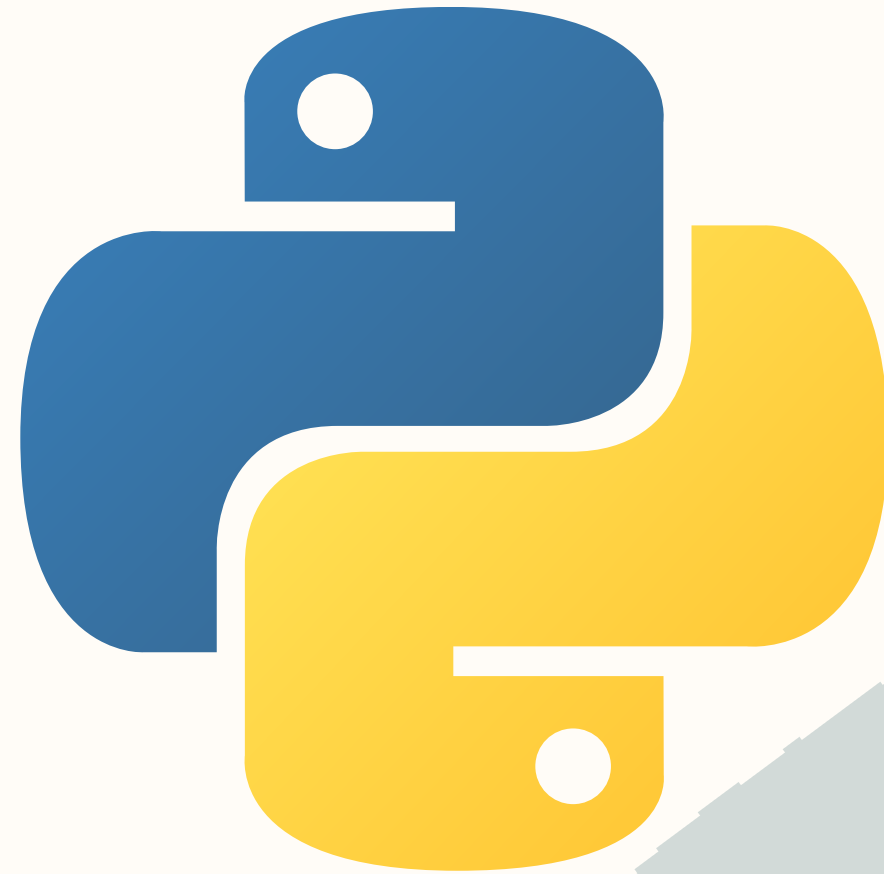
- Compartilham o mesmo espaço de memória
- Comunicação direta por variáveis compartilhadas
- Mais eficiente que comunicação entre processos



# Características e Peculiaridades

- 
- Rodam dentro do mesmo processo (JVM)
  - Risco de erros de concorrência se não sincronizadas corretamente
  - Modernas APIs (ExecutorService, Virtual Threads - Project Loom)
  - Boa escalabilidade com uso de pools e estratégias reativas

# Python



# Processos

## Implementação

- Utiliza multiprocessing
- Processos independentes
- Evita GIL (Global Interpreter Lock)

## Sincronização

- Evita conflitos
- Utiliza mecanismos: Lock, Queue, etc
- Previne Race Condition

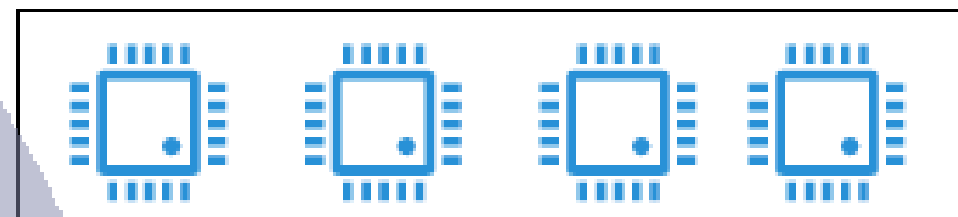
## Comunicação

- Necessitam de Mecanismos para comunicação: Pipe, Queue
- Usados em caso de troca de dados
- Não compartilham memória

# Processos em Python - Funcionamento

## Processo Paralelo

Quad-Core CPU

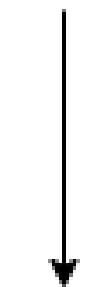
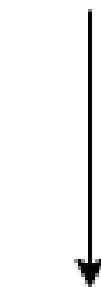


CPU 1

CPU 2

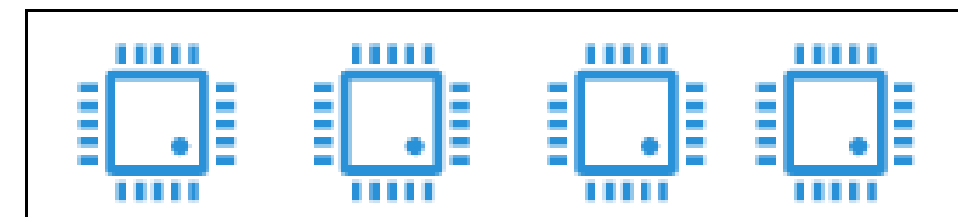
CPU 3

CPU 4



## Processo Serial

Quad-Core CPU

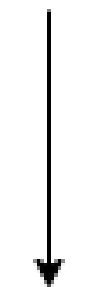


CPU 1

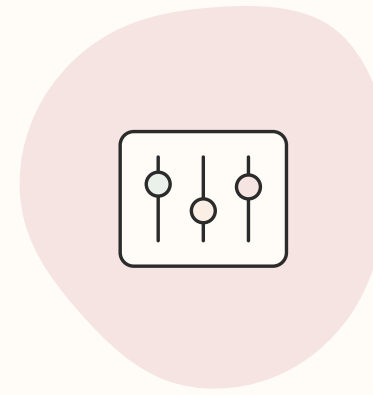
CPU 2

CPU 3

CPU 4

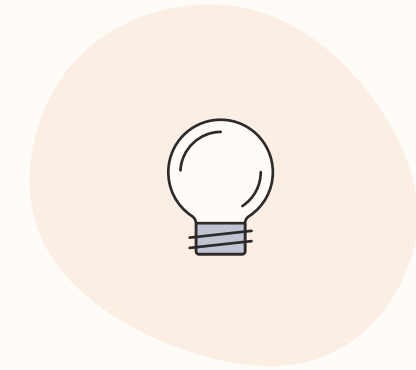


# Vantagens



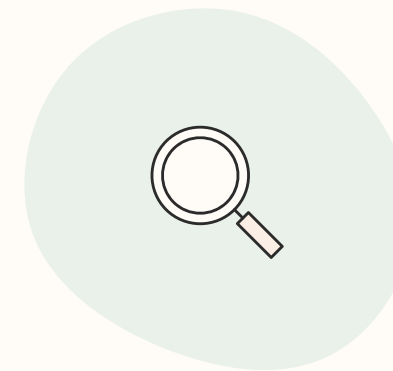
## Paralelismo Real

Executam em  
múltiplos núcleos de  
CPU



## Isolamento de Memória

Cada processo possui  
sua própria memória



## Segurança de Dados

Menor risco de  
interferência entre  
tarefas



## Estabilidade

Se um falhar, não  
interfere em outro



# Características e Peculiaridades

## Limitação de Recursos

Limitado pelo So e recursos de máquinas

## Ideal para CPU\_bound

Melhor desempenho em tarefas que exigem processamento intensivo

## Alto Overhead

Processos consomem mais recursos que threads.

## Pool de processos

`multiprocessing.Pool`  
distribue tarefas automaticamente



# Threads



## Implementação

- Utiliza threading
- Utiliza a classe Thread para criar
- Compartilham memória

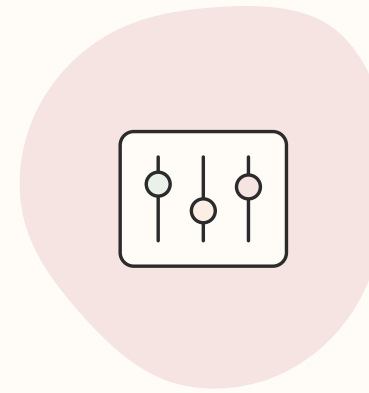
## Sincronização

- Para evitar conflitos, utilizam mecanismos
- Lock, Event, Semaphore

## Comunicação

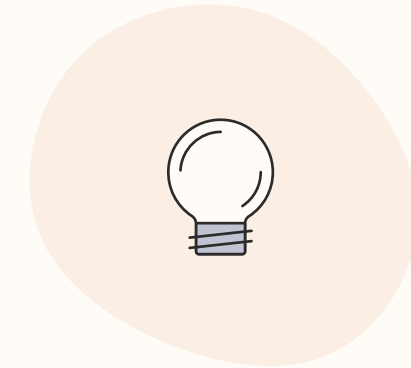
- Direta por utilizar variáveis compartilhadas
- Foco principal:

# Vantagens



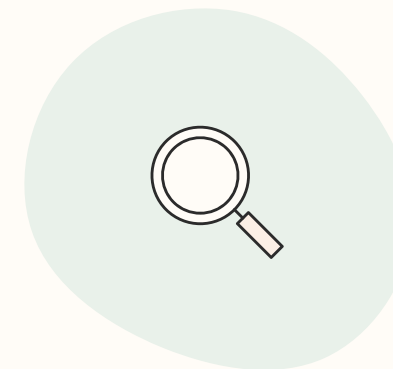
## Eficiência em Uso de Recursos

Consome menos  
recursos que  
múltiplos processos



## Fácil Comunicação Interna

A troca de informações  
é mais rápida e direta



## Maior Desempenho

Permite execução de  
múltiplas tarefas



## Aplicações em tempo real

Útil para sistemas  
que precisam de  
respostas rápidas

# Características e Peculiaridades

Gerenciadas pela  
**threading**

Python oferece suporte  
nativo a threads.

Usadas com funções  
ou classes

Podem ser criadas usando  
funções ou subclasses

Comunicação

Queue é o módulo  
reponsável por fazer a  
comunicação segura.

Fácil interrupção

Se o processo principal  
encerra, todas as  
threads filhas encerram

# Situação: Problema e Solução

## Sistema de Monitoramento

Imagine uma estufa inteligente, utilizada para cultivo de plantas, onde é essencial manter o ambiente sob controle rigoroso para garantir o crescimento saudável das espécies cultivadas. Nesse contexto, é necessário monitorar constantemente variáveis ambientais como:

- Temperatura
- Umidade
- Luminosidade

O sistema precisa coletar os dados desses sensores, processar essas informações em tempo real, e responder rapidamente a possíveis anomalias que possam comprometer o equilíbrio da estufa.

Go



# Go: Concorrência com Goroutines e Canais

**`"Do not communicate by sharing memory; share memory by communicating"`**

`"Não comunique compartilhando memória, compartilhe memória comunicando"`

O mantra da linguagem de programação Go, que encapsula sua filosofia de concorrência. Ela enfatiza que, em vez de múltiplas goroutines acessarem a mesma memória compartilhada e usarem mecanismos de sincronização complexos (como travas), é preferível que as goroutines se comuniquem trocando mensagens através de canais.

# Goroutines

(Equivalente a Threads)

## Implementação

- Unidades leves gerenciadas pelo runtime Go (não pelo SO)
- Iniciadas com `“go func()”`
- Baixo custo (stack inicial de ~2KB)

## Sincronização

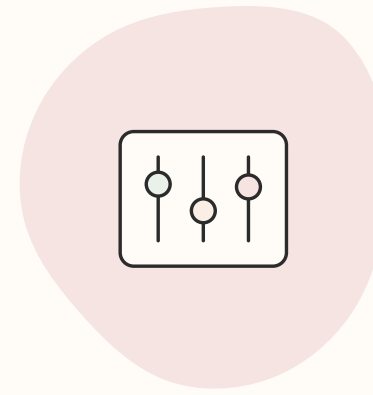
- Primariamente através de canais (comunicação segura)
- Alternativas: `sync.Mutex`, `sync.WaitGroup`

## Comunicação

- Canais como mecanismo nativo (`chan Name`)
- Princípio: "Não comunique compartilhando memória; compartilhe memória comunicando"

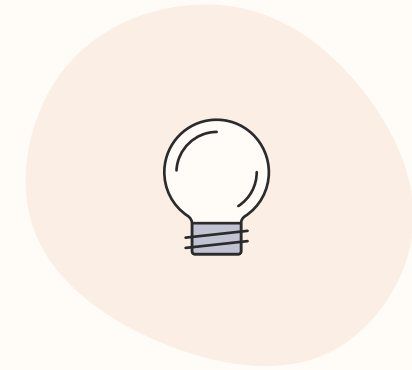


# Vantagens



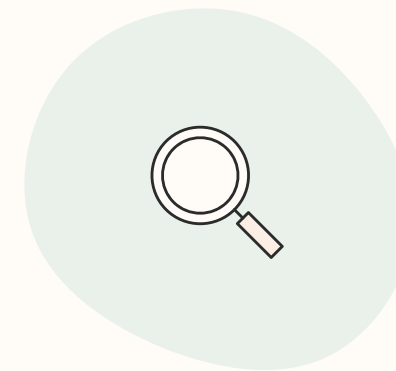
## Concorrência Eficiente

Baixo custo (criação de  
milhões de goroutines)  
Otimizada para multi-core



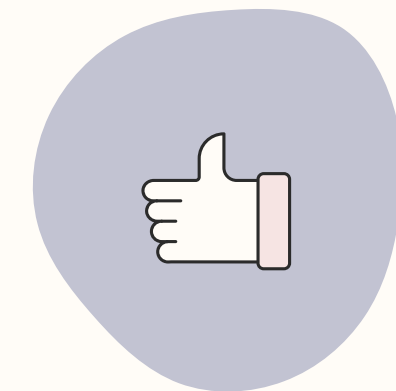
## Gerenciamento Automático

Escalonador integrado ao  
runtime Go



## Comunicação Segura

Canais eliminam necessidade  
de locks explícitos



## Integração Nativa

Projetada para concorrência  
desde sua origem

# Características e Peculiaridades

## Leveza

Stack inicial de 2KB (cresce dinamicamente)

## Modelo de Comunicação

Canais bloqueantes por padrão (unbuffered)

Canais com buffer opcional (buffered)

## Padrões de Concorrência

Pipelines, worker pools, fan-out/fan-in

## Finalização Controlada

Uso de context para cancelamentos e timeouts

# Caso Prático: Problema e Implementação

## Sistema de Processamento de Pedidos Online

Um e-commerce de grande escala recebe centenas de pedidos por minuto. Cada pedido precisa passar por múltiplas etapas interdependentes:

- Validação: Verificar dados do cliente, disponibilidade de produtos e regras de negócio
- Pagamento: Processar transações financeiras com segurança
- Estoque: Atualizar inventário e preparar itens para envio
- Notificação: Comunicar status em tempo real ao cliente

O sistema precisa coordenar essas etapas interdependentes de forma concorrente, lidar com picos de demanda imprevisíveis e garantir a integridade de cada operação mesmo em situações de falha parcial, evitando perdas financeiras e insatisfação do cliente.

# Comparação entre Processos

Aspecto	C	JAVA	PYTHON	GO
Criação	<code>fork()</code>	<code>ProcessBuilder</code>	<code>os.fork()</code> / <code>multiprocessing</code>	<code>exec.Command()</code>
Controle	Direto	Abstração	Flexível	Simples
Portabilidade	Unix only	Multiplataforma	Limitada	Multiplataforma
Performance	Máxima	Boa	Boa	Excelente

# Comparação entre Threads/Goroutines

Aspecto	C	JAVA	PYTHON	GO
Criação	<code>pthread_create()</code>	<code>Thread / Runnable</code>	<code>threading.Thread</code>	<code>go</code> keyword
Sincronização	<code>pthread_mutex</code>	<code>synchronized</code>	<code>threading.Lock</code>	<code>channels</code>
Paralelismo	Real	Real	Limitado (GIL)	Real
Facilidade	Complexa	Simples	Simples	Muito Simples
Overhead	Médio	Médio	Médio	Muito Baixo

### Principais Aprendizados

Processos oferecem isolamento e segurança, mas com maior overhead

Threads/Goroutines permitem comunicação eficiente e menor uso de recursos

Go revoluciona concorrência com goroutines ultra-leves

A escolha depende dos requisitos específicos da aplicação

### Recomendações de Uso

**C:** Sistemas de baixo nível, performance crítica, controle total

**Java:** Aplicações empresariais, sistemas distribuídos, portabilidade

**Python:** Prototipagem rápida, I/O intensivo, facilidade de desenvolvimento

**Go:** Microserviços, alta concorrência, sistemas modernos e escaláveis

# REFERÊNCIAS

- EMBARCADOS. Threads com padrão POSIX (pthreads). Disponível em: <https://embarcados.com.br/threads-posix/>. Acesso em: 23 jun. 2025. [pt.wikipedia.orgprogramacaoprogessiva.net+4embarcados.com.br+4web.cs.ucla.edu+4](https://embarcados.com.br/threads-posix/)
- W3RESOURCE. Multithreading in C with POSIX Threads: A Complete Guide. Atualizado em 18 set. 2024. Disponível em: <https://www.w3resource.com/c-programming/c-multithreading.php>. Acesso em: 23 jun. 2025. [w3resource.com](https://www.w3resource.com/c-programming/c-multithreading.php)
- WIKIPEDIA. Pthreads. Disponível em: [https://pt.wikipedia.org/wiki/POSIX\\_Threads](https://pt.wikipedia.org/wiki/POSIX_Threads). Acesso em: 23 jun. 2025. [pt.wikipedia.org](https://pt.wikipedia.org/wiki/POSIX_Threads)
- PYTHON SOFTWARE FOUNDATION. multiprocessing – Paralelismo baseado em processo. Versão 3.13.2. Disponível em: <https://docs.python.org/pt-br/3.13/library/multiprocessing.html>. Acesso em: 23 jun. 2025. [medium.com+4docs.python.org+4docs.python.org+4](https://docs.python.org/pt-br/3.13/library/multiprocessing.html)
- PYTHON SOFTWARE FOUNDATION. threading – Thread-based parallelism. Versão 3.13.3. Disponível em: <https://docs.python.org/pt-br/3/library/threading.html>. Atualizado em 28 abr. 2025. Acesso em: 23 jun. 2025.
- DIDÁTICA TECH. Processamento paralelo com Python: usando vários núcleos de CPU. Disponível em: <https://didatica.tech/>.... Acesso em: 23 jun. 2025. [didatica.tech](https://didatica.tech/)
- API BRASIL. Programação Paralela e Concorrente em Python: Técnicas e Estratégias. 2 ago. 2024. Disponível em: <https://apibrasil.blog/>.... Acesso em: 23 jun. 2025. [apibrasil.blog](https://apibrasil.blog/)
- CIÊNCIA DE DADOS BRASIL. Desmistificando Goroutines e Canais em Go: Um Guia para Cientistas de Dados. Disponível em: <https://cienciadedadosbrasil.com.br/>.... Acesso em: 23 jun. 2025. [cienciadedadosbrasil.com.br](https://cienciadedadosbrasil.com.br/)
- ARXIV. SAIoc, G.-V.; SHIRCHENKO, D.; CHABBI, M. Unveiling and Vanquishing Goroutine Leaks in Enterprise Microservices: A Dynamic Analysis Approach. 19 dez. 2023. Disponível em: <https://arxiv.org/abs/2312.12002>. Acesso em: 23 jun. 2025. [arxiv.org](https://arxiv.org/abs/2312.12002)
- WIKIPEDIA. Go (programming language). Disponível em: [https://en.wikipedia.org/wiki/Go\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Go_(programming_language)). Acesso em: 23 jun. 2025.

# REFERÊNCIAS

- IEEE. POSIX.1c-1995: Threads Extensions (IEEE Std 1003.1c-1995). Normas disponíveis via IEEE / The Open Group.
- ORACLE. Java™ Platform, Standard Edition 21 API Specification - java.lang.Thread, java.util.concurrent. Disponível em: <https://docs.oracle.com/en/java/javase/21/docs/api/index.html>. Acesso em: 23 jun. 2025.
- PYTHON SOFTWARE FOUNDATION. threading – Thread-based parallelism. Documentação Python 3.13.3. (ver acima) [en.wikipedia.org+1pt.wikipedia.org+1docs.python.org+2docs.python.org+2docs.python.org+2](https://en.wikipedia.org+1pt.wikipedia.org+1docs.python.org+2docs.python.org+2docs.python.org+2)
- PYTHON SOFTWARE FOUNDATION. multiprocessing – Paralelismo baseado em processo. Documentação Python 3.13.2. (ver acima) [docs.python.org+1docs.python.org+1](https://docs.python.org+1docs.python.org+1)
- GOOGLE. The Go Programming Language Specification, 2023. Disponível em: <https://go.dev/ref/spec>. Acesso em: 23 jun. 2025.
- GOOGLE. Effective Go – seção sobre goroutines e canais. Disponível em: [https://go.dev/doc/effective\\_go](https://go.dev/doc/effective_go). Acesso em: 23 jun. 2025.



Obrigado Pela  
Atenção