## 

Relatório trabalho Computação Escalável:

Plataforma de E-commerce e Gestão de Cadeia de Suprimentos

21 de Abril de 2025

# Integrantes:

Daniel de Miranda Almeida

João Felipe Vilas Boas

Lívia Verly

Luís Felipe de Abreu Marciano

Paulo César Gomes Rodrigues

# Visão Geral

Este trabalho propõe o desenvolvimento de um micro-framework para a construção de pipelines de processamento de dados, com foco em eficiência e suporte a paralelismo. A proposta é criar uma solução escalável, considerando a capacidade de processamento da máquina do usuário, e que permita o processamento de múltiplos componentes de forma concorrente. Para demonstrar sua aplicabilidade, será utilizado um projeto de exemplo nas áreas de E-commerce e Gestão da Cadeia de Suprimentos. É possível ver todas implementações que serão explicadas neste relatório em nosso [repositório GitHub](https://github.com/Vilasz/A1_comp_esc) (<https://github.com/Vilasz/A1_comp_esc>).

# Objetivos

O principal objetivo deste trabalho é demonstrar, de forma eficiente, como o processamento de dados pode ser potencializado por meio do uso de técnicas de paralelismo e concorrência em uma única máquina. A proposta é evidenciar os ganhos de desempenho obtidos a partir da execução simultânea de tarefas em diferentes núcleos de processamento. Para isso, será desenvolvido um micro-framework que incorpora esses conceitos, permitindo a construção de pipelines modulares e escaláveis.

# Python ou C++?

Uma das primeiras escolhas de projeto que tivemos foi decidir entre Python e C++. A linguagem de programação escolhida foi Python. Sabemos que, quando se trata de programação eficiente, C++ é preferível à Python. Não só porque é uma linguagem de mais baixo nível, em que temos mais garantias de atomicidade nas operações, mas também por conta do GIL (Global Interpreter Lock) do Python, que nos impede de trabalhar com multithreading real, que seria uma boa opção dado o objetivo do nosso trabalho. Entretanto, ainda sim optamos por utilizar Python por conta da afinidade dos membros do grupo com a linguagem e também pela aula sobre paralelismo e concorrência em Python que tivemos.

Por consequência da escolha de linguagem de programação, para que tenhamos processos realmente rodando em paralelo em nossas implementações, o multithreading não é mais uma opção viável. Em seu lugar, precisamos utilizar o módulo multiprocessing, que permite a criação de múltiplos processos independentes, cada um com sua própria memória e interpretador Python.

O multiprocessing consegue contornar o problema do GIL criando vários processos separados, cada um com seu próprio interpretador Python. Isso faz com que eles possam rodar ao mesmo tempo, usando diferentes núcleos do processador, o que garante paralelismo de verdade. Além disso, o módulo oferece ferramentas como Queue, Pipe e Pool que ajudam na comunicação entre processos e na divisão das tarefas. Por outro lado, como esses processos não compartilham a mesma memória (ao contrário das threads), é preciso criar formas específicas para que eles troquem informações, o que pode deixar o sistema mais lento por causa da necessidade de copiar e transformar os dados.

Já o multithreading, quando usado em linguagens como C++ que não possuem GIL, tem algumas vantagens importantes. As threads são mais leves, mais rápidas de criar e podem compartilhar a memória entre si, o que facilita a comunicação.

Por isso, escolhemos usar multiprocessing, pois ele permite um paralelismo real nas tarefas mais pesadas, mesmo que isso torne o projeto mais complexo. Essa foi uma decisão importante para que nosso pipeline conseguisse aproveitar melhor o desempenho dos processadores, mesmo com as limitações do Python.

# Micro-framework

O micro-framework desenvolvido tem como objetivo facilitar a construção de pipelines de processamento de dados, permitindo o encadeamento de etapas de forma paralela (via processos), utilizando recursos como queues, workers, dispatchers, além de abstrações como DataFrame, repositórios de dados, tratadores e triggers.

O módulo dataframe.py define uma classe DataFrame que representa um conjunto de dados em memória, estruturado em colunas e linhas. Essa classe é central para a manipulação de dados ao longo do pipeline. Foi escolhida uma representação com uma lista python em que cada elemento é outra lista que representa uma linha do dataframe. Nessa representação, todas as listas devem ter o mesmo tamanho, correspondendo com o número de colunas do dataframe. Dessa forma, as operações que consistem em percorrer as linhas do DataFrame são facilitadas, visto que basta percorrer uma lista de listas.

Foram implementados também métodos tradicionais de um DataFrame, tais como *group\_by*, *sort\_by*, *vstack* e *to\_dicts* que tem funções intuitivas de acordo com seus nomes e serão essenciais para realizar análises posteriormente.

No arquivo *pedidos\_handler.py*, os repositórios são representados pelas classes *CSVHandler* e *JSONHandler*. Esses componentes encapsulam a lógica de extração e carregamento de dados a partir de fontes externas(CSV e JSON). Os dados extraídos são padronizados para a estrutura DataFrame por meio da função *merge\_dataframes*.

A classe *ConcurrentPipeline* implementa toda a lógica de execução concorrente e paralela do pipeline. Cada etapa do pipeline é representada por uma função tratadora, que implementa uma transformação específica nos dados. Essas funções são conectadas à estrutura do pipeline por meio do método *add\_stage*, que adiciona uma nova etapa com seu respectivo tratador. Cada etapa é executada em paralelo utilizando *ProcessPoolExecutor*, garantindo escalabilidade.

O *trigger* é feito de forma manual e é representado pelas chamadas à função *start()* e pelo método *feed\_data()* ou *add\_batch()*, que alimentam o pipeline com dados. O *trigger* inicializa todo o pipeline, cria os dispatchers, ativa os pools de processos e inicia o processamento de forma assíncrona, até o envio do sentinela por *end()*.

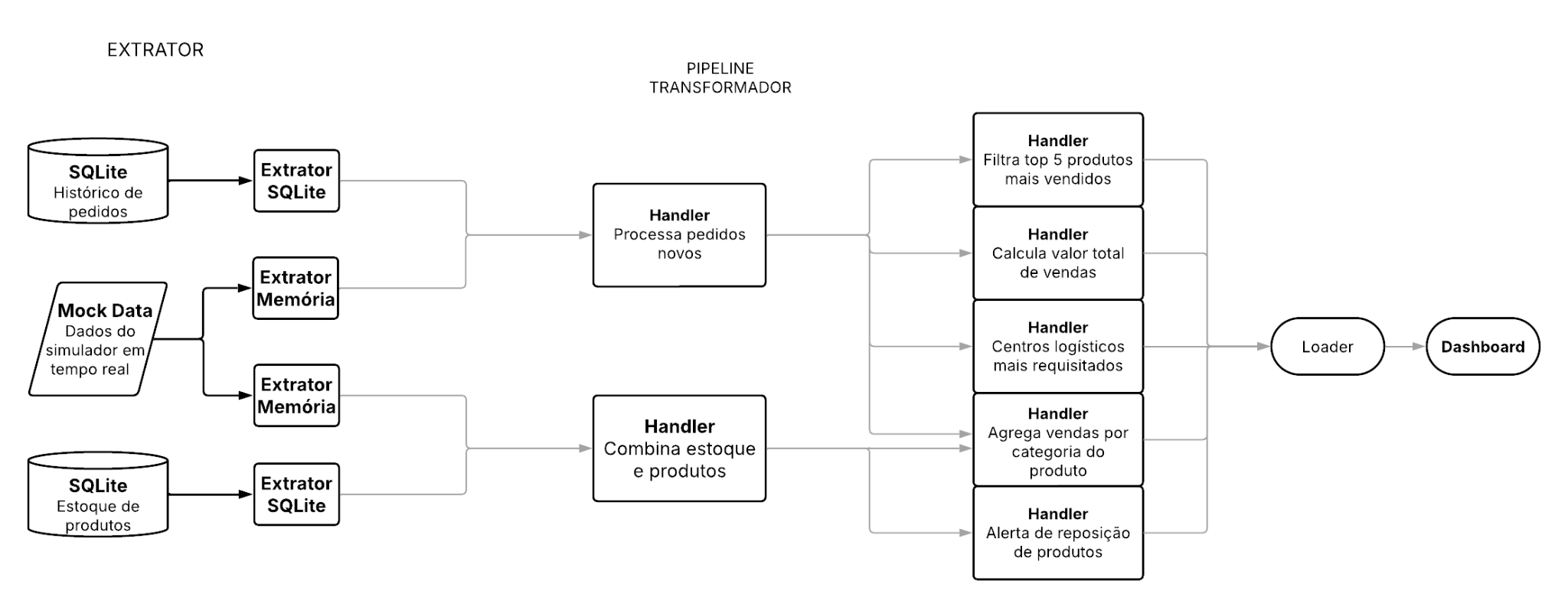
# Paralelismo e Concorrência

Por isso, escolhemos usar multiprocessing, pois ele permite um paralelismo real nas tarefas mais pesadas, mesmo que isso torne o projeto mais complexo. Essa foi uma decisão importante para que nosso pipeline conseguisse aproveitar melhor o desempenho dos processadores, mesmo com as limitações do Python.

# Projeto Exemplo: Plataforma de E-commerce e Gestão de Cadeia de Suprimentos

O projeto escolhido para mostrar aplicações do nosso micro-framework foi uma Plataforma de E-commerce e Gestão de Cadeia de Suprimentos. Acreditamos que é um bom exemplo pois pode envolver grandes volumes de dados, múltiplas fontes de entrada e diferentes etapas de processamento, a depender da análise desejada. Além disso, é um cenário realista e de fácil entendimento, o que facilita a demonstração prática dos recursos do framework.   
 Para o desenvolvimento e os testes do projeto, criamos uma simulação com dados fictícios (mock) usando scripts Python. Os arquivos *gera\_csv\_pedidos.py* e *gera\_json\_pedidos.py* geram, respectivamente, pedidos em formato CSV e JSON, com variação na quantidade de produtos, localização de centros logísticos e tipos de itens. Já o arquivo *create\_database.py* define toda a estrutura do banco de dados relacional em SQLite, com tabelas que representam as principais entidades do sistema. Em seguida, o script *populate\_db.py* é responsável por inserir milhares de registros simulados, incluindo clientes, produtos, centros logísticos, pedidos, itens de pedido e entregas, todos com atributos gerados aleatoriamente. Optamos pelo uso do SQLite por ser uma solução leve, de fácil integração com Python e suficiente para representar uma camada de persistência realista no contexto do projeto.

Essa estrutura nos permitiu testar com dados realistas em volume considerável, simulando o comportamento de um sistema real de e-commerce e permitindo validar a eficiência e a escalabilidade do micro-framework proposto. O pipeline ETL proposto foi o seguinte.



O objetivo do pipeline é processar e analisar dados de um sistema de e-commerce, juntando informações antigas com dados simulados em tempo real, e transformando tudo isso em informações e análises úteis que aparecem em um painel (dashboard).

Ele começa com a leitura de dados de três fontes: um banco SQLite com o histórico de pedidos, um conjunto de dados simulados em tempo real (Mock Data), e as informações sobre o estoque dos produtos no mesmo banco de dados. Esses dados são extraídos por módulos extratores. Depois disso, dois blocos principais cuidam do processamento. Um deles recebe os novos pedidos simulados e insere esses pedidos no histórico, adicionando informações como o número do pedido, a data e hora atual, o status de "novo", o cliente, o centro logístico, o valor total e o endereço. Também adiciona os produtos comprados, incluindo o produto, a quantidade e o preço. O outro bloco cuida da atualização do estoque. Para cada pedido novo, ele diminui do estoque a quantidade de produto correspondente à compra.

Com essas atualizações feitas, o pipeline passa a calcular algumas informações importantes. Ele descobre, por exemplo, quais foram os 5 produtos mais vendidos, quanto foi o total em vendas, quais centros logísticos receberam mais pedidos, quantas unidades foram vendidas por categoria de produto, e também alerta quando algum produto está com o estoque muito baixo (10 unidades ou menos). Essas informações são então reunidas e enviadas para o dashboard, que mostra tudo de forma visual. Assim, o pipeline ajuda o sistema a ter uma visão atualizada das vendas e do estoque, usando dados reais e simulados, o que facilita muito a tomada de decisões rápidas e inteligentes.

Vale ressaltar que o objetivo do nosso trabalho é demonstrar como o processamento de dados pode se tornar muito mais eficiente com a aplicação de técnicas de paralelismo e concorrência. Por isso, não nos preocupamos em tornar cada função dos tratadores a mais otimizada possível. Na verdade, podemos considerar até desejável que essas funções realizem um processamento mais intenso, pois isso evidencia melhor o ganho de desempenho proporcionado por uma arquitetura paralela, à medida que aumentamos o número de processos executados simultaneamente.

# Resultados

Com essas atualizações feitas, o pipeline passa a calcular algumas informações importantes. Ele descobre, por exemplo, quais foram os 5 produtos mais vendidos, quanto foi o total em vendas, quais centros logísticos receberam mais pedidos, quantas unidades foram vendidas por categoria de produto, e também alerta quando algum produto está com o estoque muito baixo (10 unidades ou menos). Essas informações são então reunidas e enviadas para o dashboard, que mostra tudo de forma visual. Assim, o pipeline ajuda o sistema a ter uma visão atualizada das vendas e do estoque, usando dados reais e simulados, o que facilita muito a tomada de decisões rápidas e inteligentes.