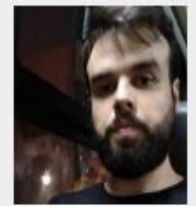
Live de Python #53

Usando Processos - Multiprocessamento #3

APOIE O CANAL
apoia.se/livedepython




Muito obrigado <3

Edimar Fardim
Eliabe Silva
Fabiano Teichmann
João Lugão
Maria boladona
Paulo Tadei
Regis Santos
Willian Lopes

1º Sorteio da Live de Python

https://goo.gl/forms/wiNGNZaXboZ5GxC12

Nome:
Eduardo Mendes

Instituição:
Unicamp / Diebold Nixdorf

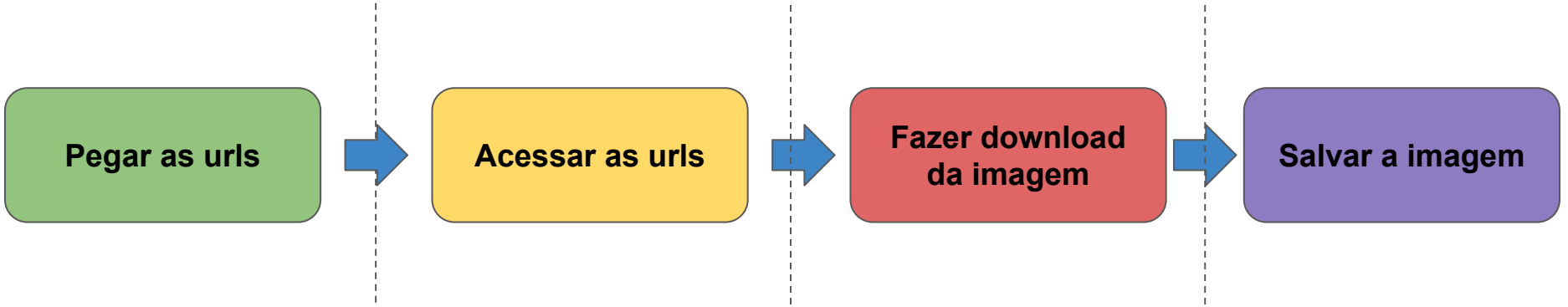
Contatos:
{facebook, github, gist
instagram, linkedin,
telegram, twitter}/dunossauro

Roteiro

- Relembrando o problema
- Relembrando nossas soluções até então
- Entendendo os fundamentos de processos
  - Similaridade de APIS (threads, multiprocess)
  - Memória compartilhada?

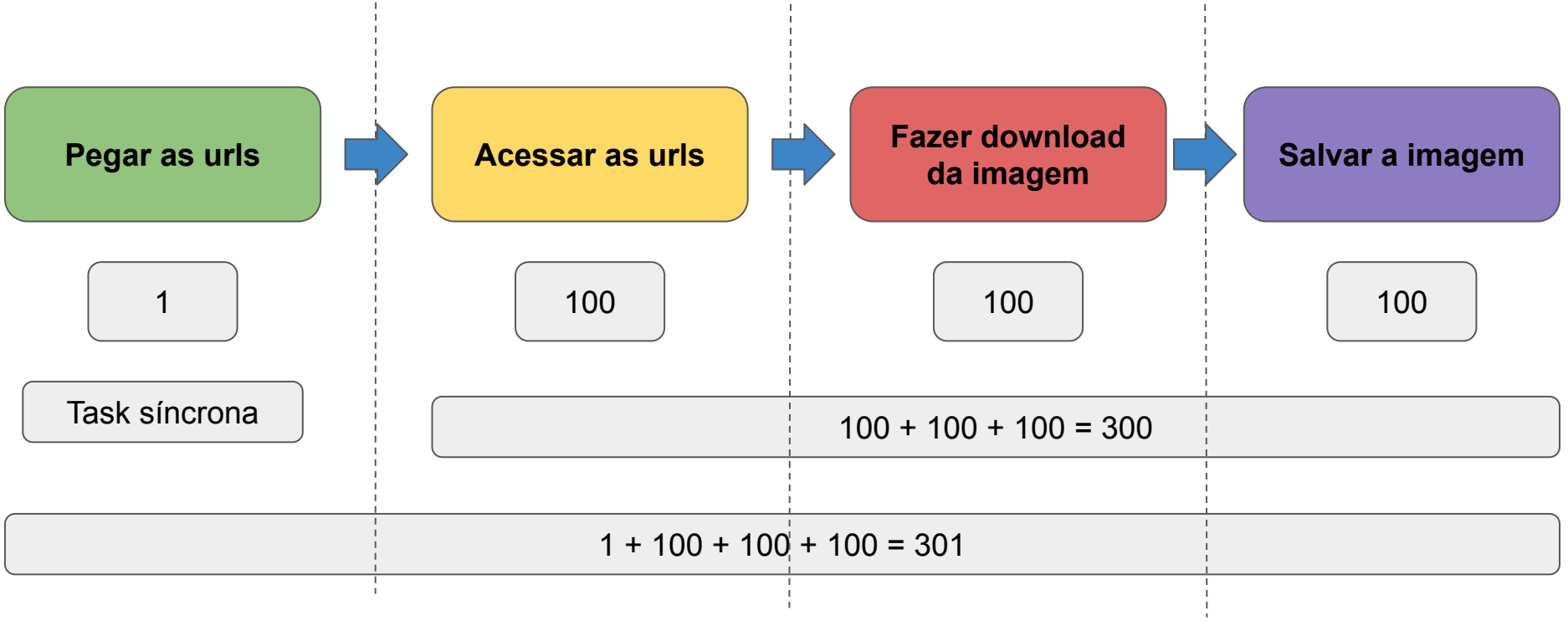
Relembrando o problema

Fazer o download de um sprite dos primeiros 100 pokémons da pokeapi

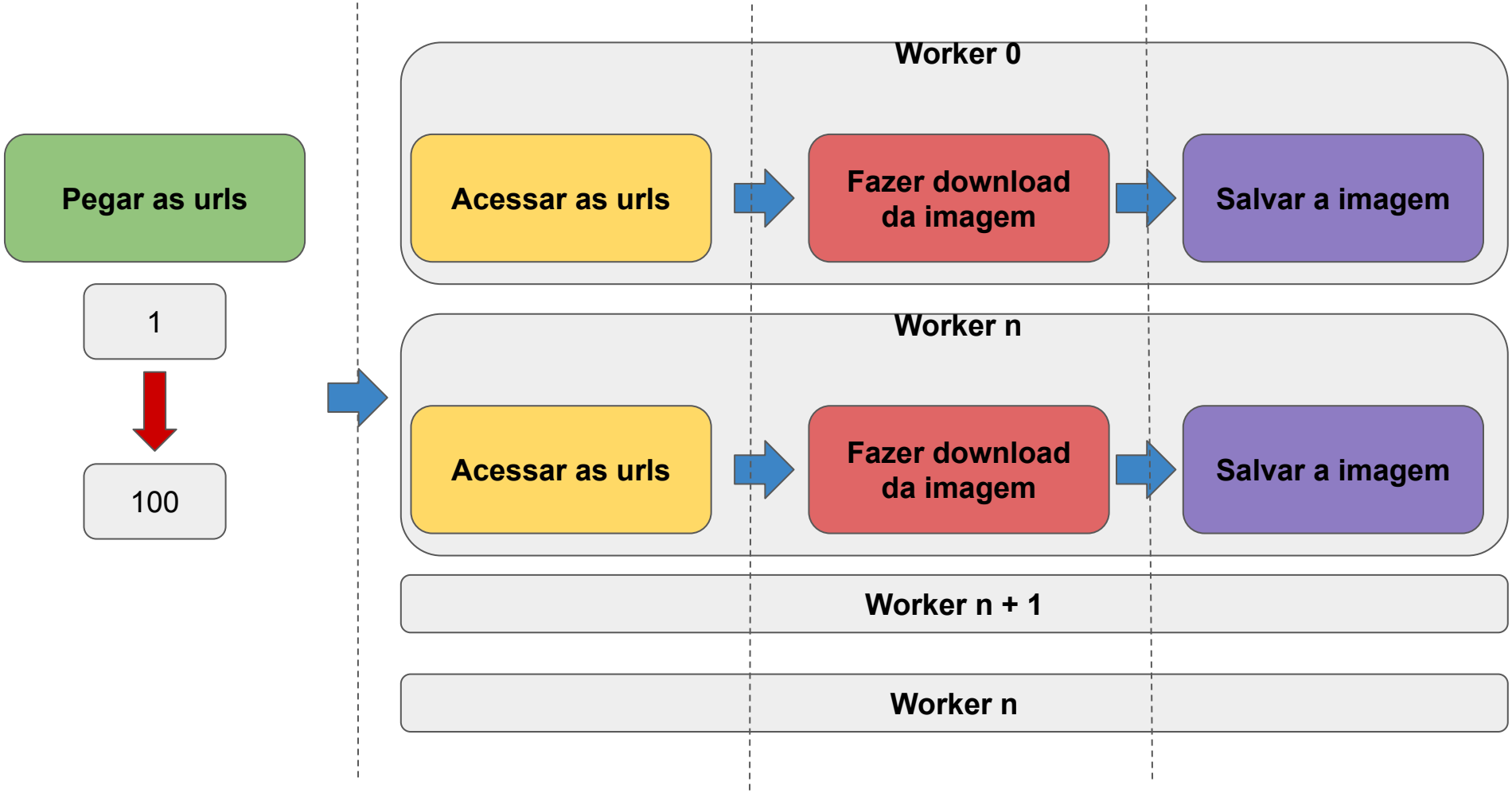


Decomposição

A decomposição inside em pegar um gargalo do processamento e decompor ele em tarefas menores. Vamos chamar tarefas de ‘tasks’ e coisas que resolvem as tasks de workers.

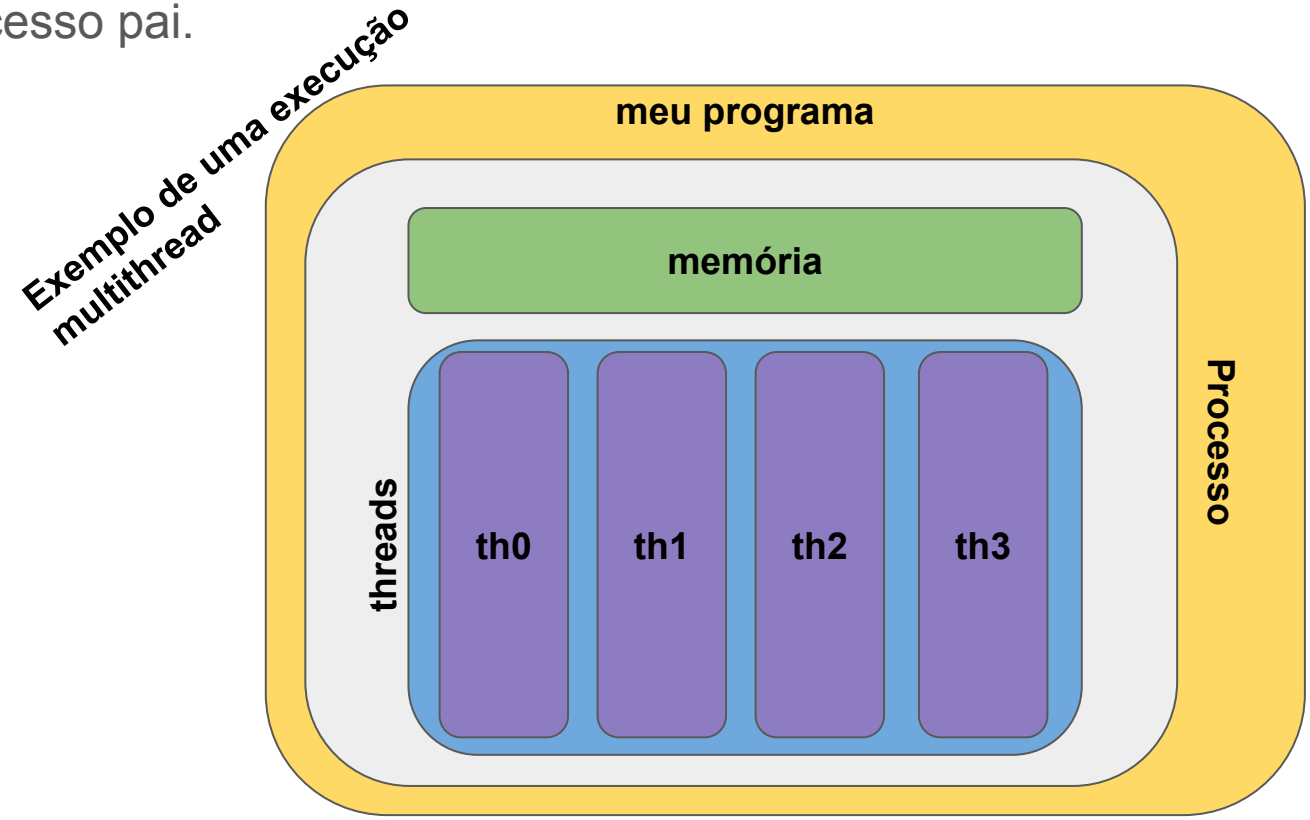


Decomposição



Processo

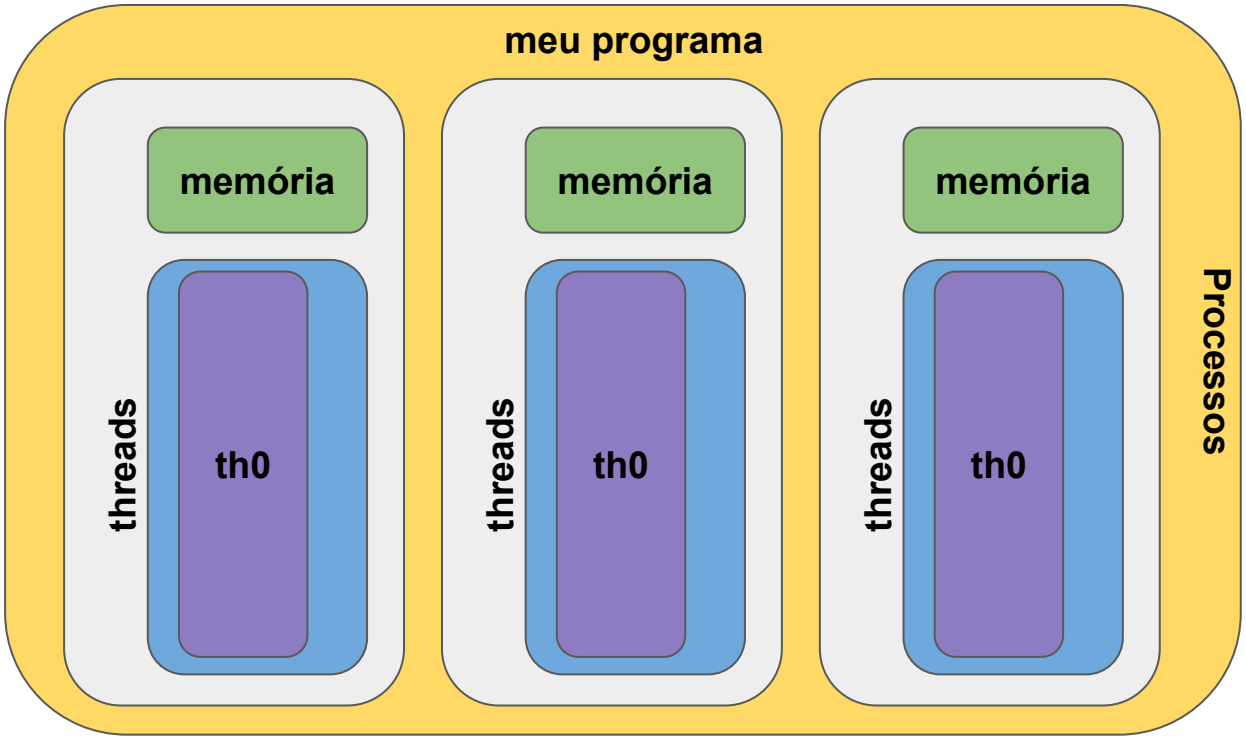
Em comparação com as Threads, processos são completamente independentes do seu “programa”. Ou seja, threads são linhas de execução dentro de um mesmo processo. Processos não compartilham nem mesmo memória com o processo pai.



Processo

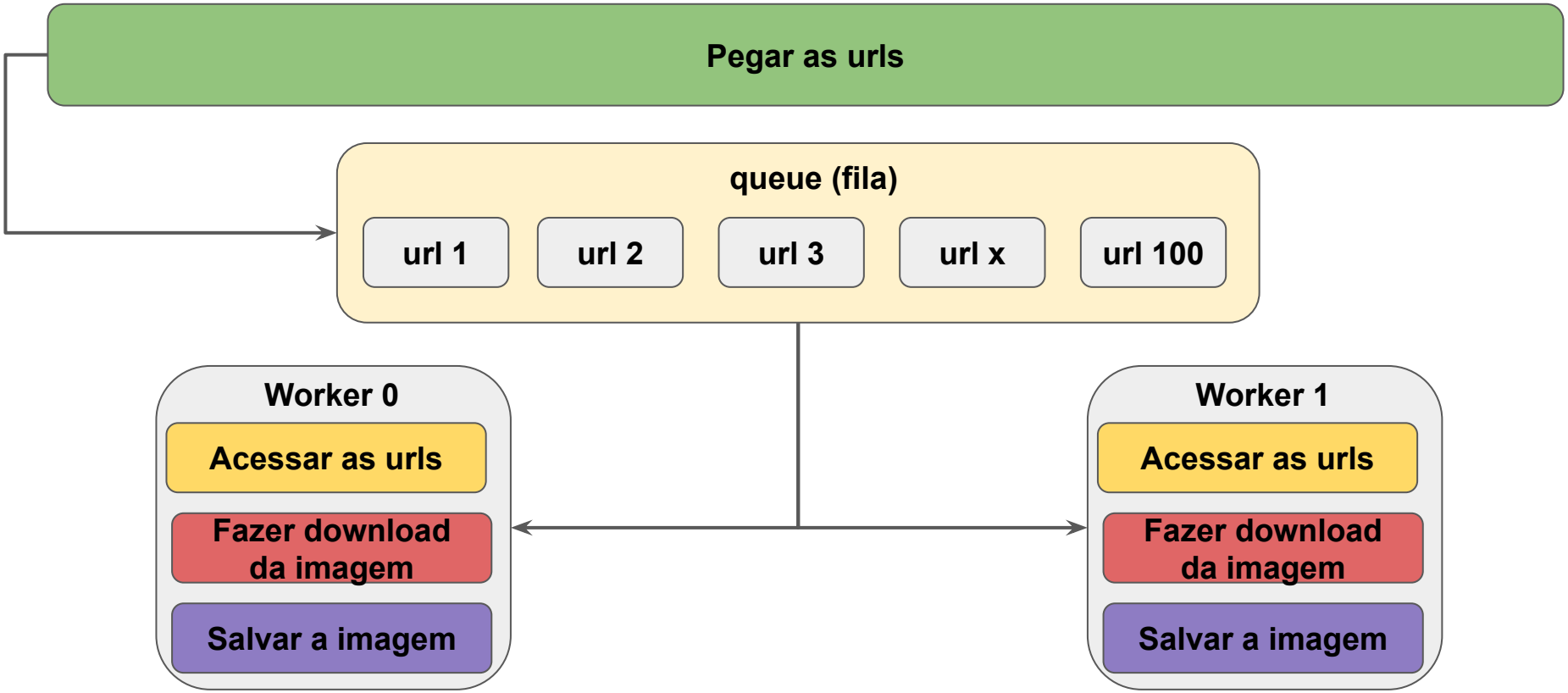
Em comparação com as Threads, processos são completamente independentes do seu “programa”. Ou seja, threads são linhas de execução dentro de um mesmo processo. Processos não compartilham nem mesmo memória com o processo pai.

Exemplo de uma execução multiprocessada



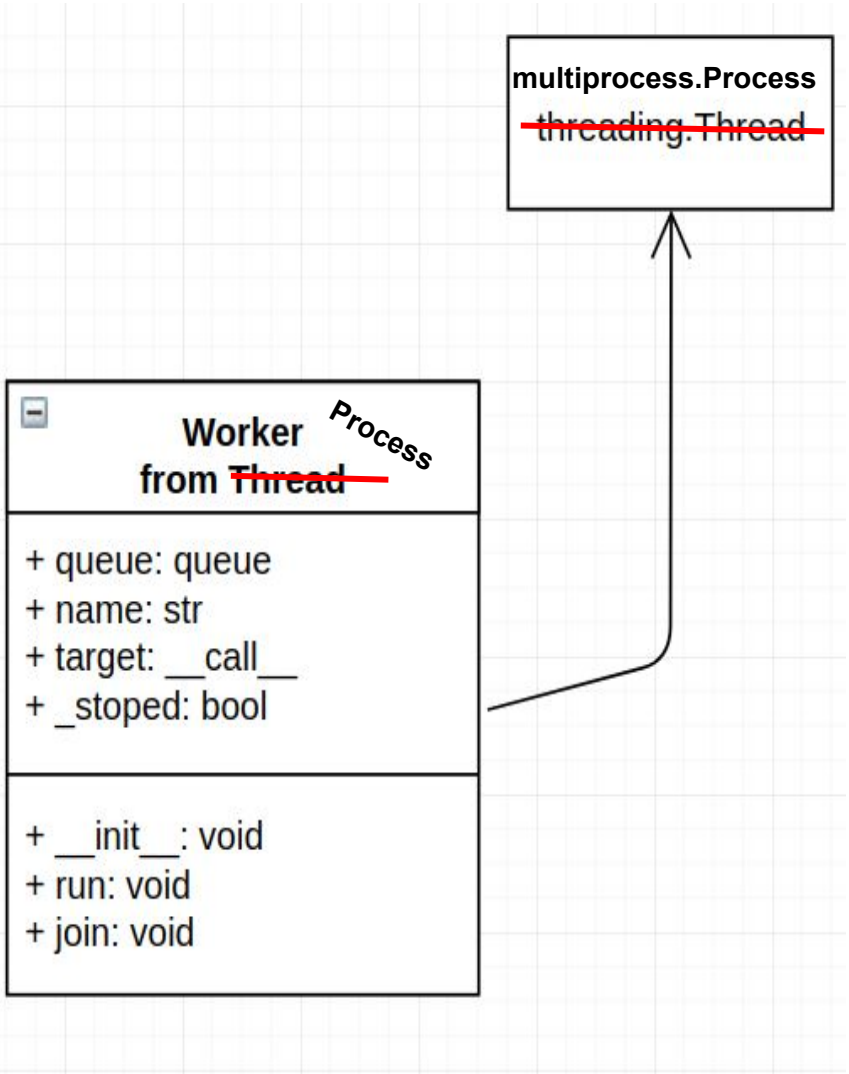
Processo

Ou seja, nessa linha de execução, não existe compartilhamento de escopo entre processos. Ou seja, nossa queue da live passada falharia. Ao menos que a queue esteja pronta múltiplos processos.



Exemplo sobre as queues

Construindo um worker (similaridade de APIs)

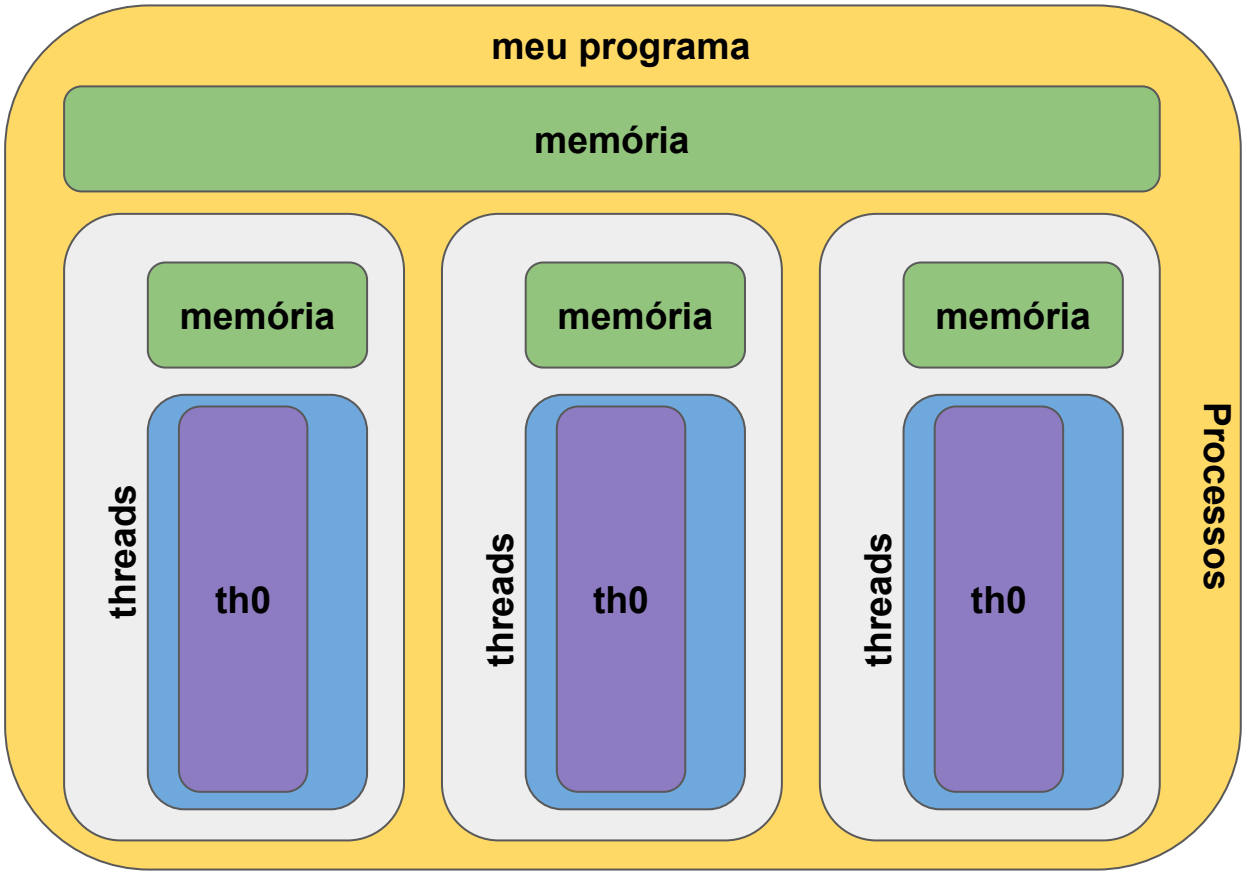



```python
class worker(Process):
    def __init__(self, target, queue, *, name):
        super.__init__()
        self.target
        self.queue = queue
        self.name = name

    def run(self):
        pass

    def join(self):
        pass
```

Exemplo sobre a API

Memória isolada entre processos

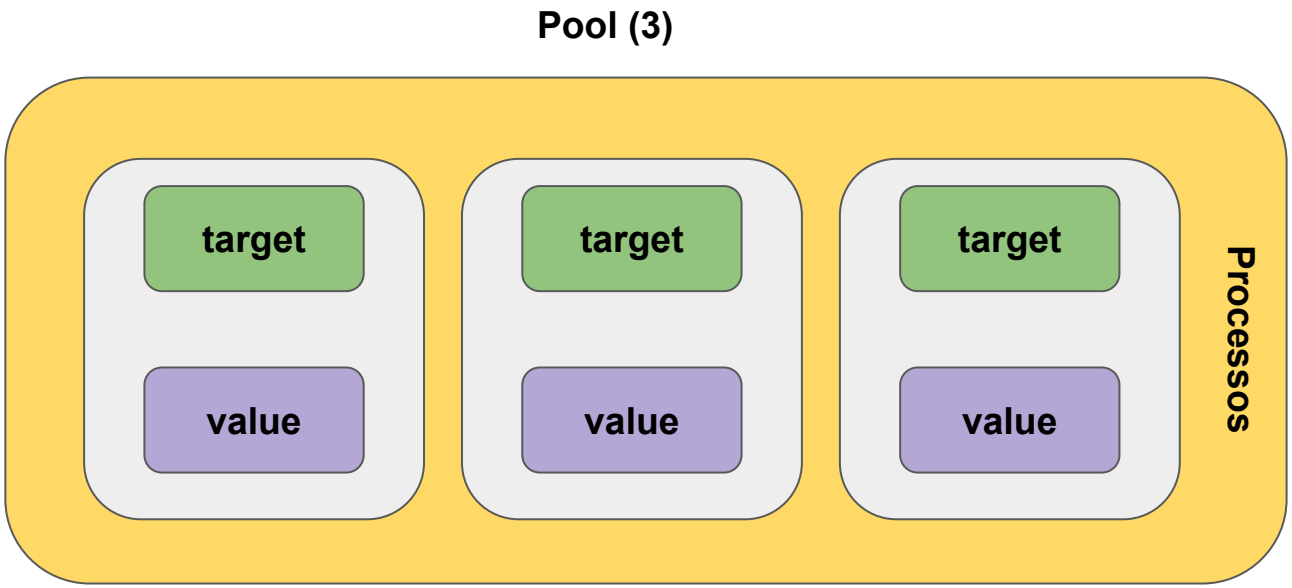
Quando um processo é iniciado ele tenta isolar o que vai precisar serializando isso usando **pickle (live 49)**.



Exemplo sobre serialização

Pools

Na **live 52**, fizemos um pool de threads, o que facilitaria nossa execução usando decomposição de dados usando nosso pipeline. Como o uso da queue do escopo compartilhado não poderia ser usada, os processos tem uma coisa mágica.

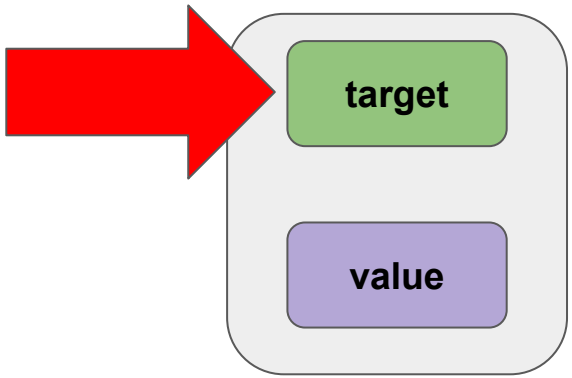


Pools

Na **live 52**, fizemos um pool de threads, o que facilitaria nossa execução usando decomposição de dados usando nosso pipeline. Como o uso da queue do escopo compartilhado não poderia ser usada, os processos tem uma coisa mágica.

Target:

Uma função ou uma classe que implemente __call__ e seja serializável

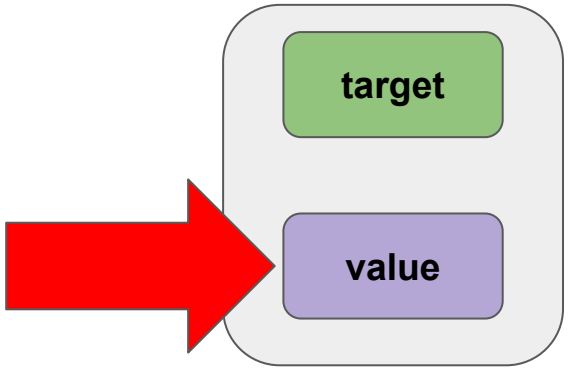


Pools

Na **live 52**, fizemos um pool de threads, o que facilitaria nossa execução usando decomposição de dados usando nosso pipeline. Como o uso da queue do escopo compartilhado não poderia ser usada, os processos tem uma coisa mágica.

Value:

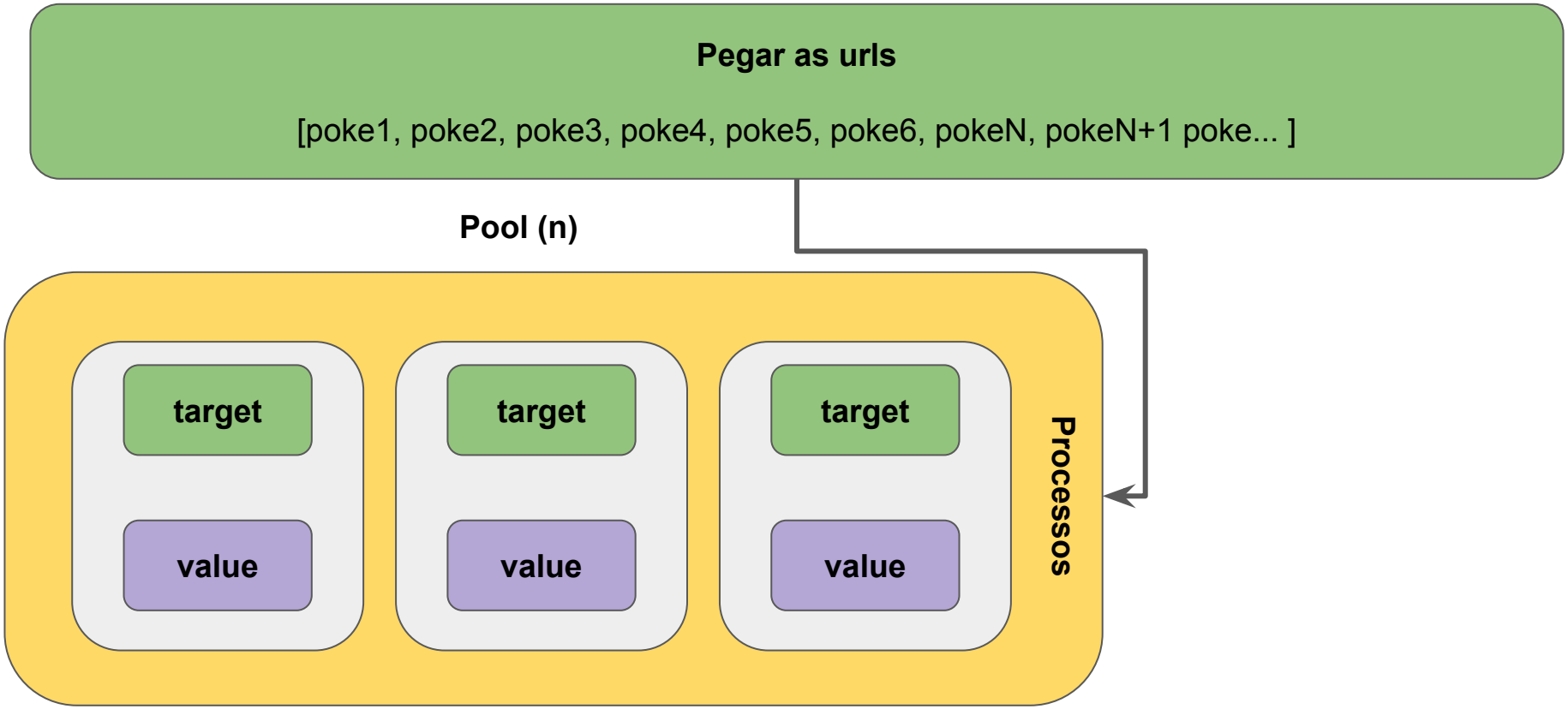
Um valor que também possa ser serializável e possa ser processado pelo target, ou seja, sem levantar exceções



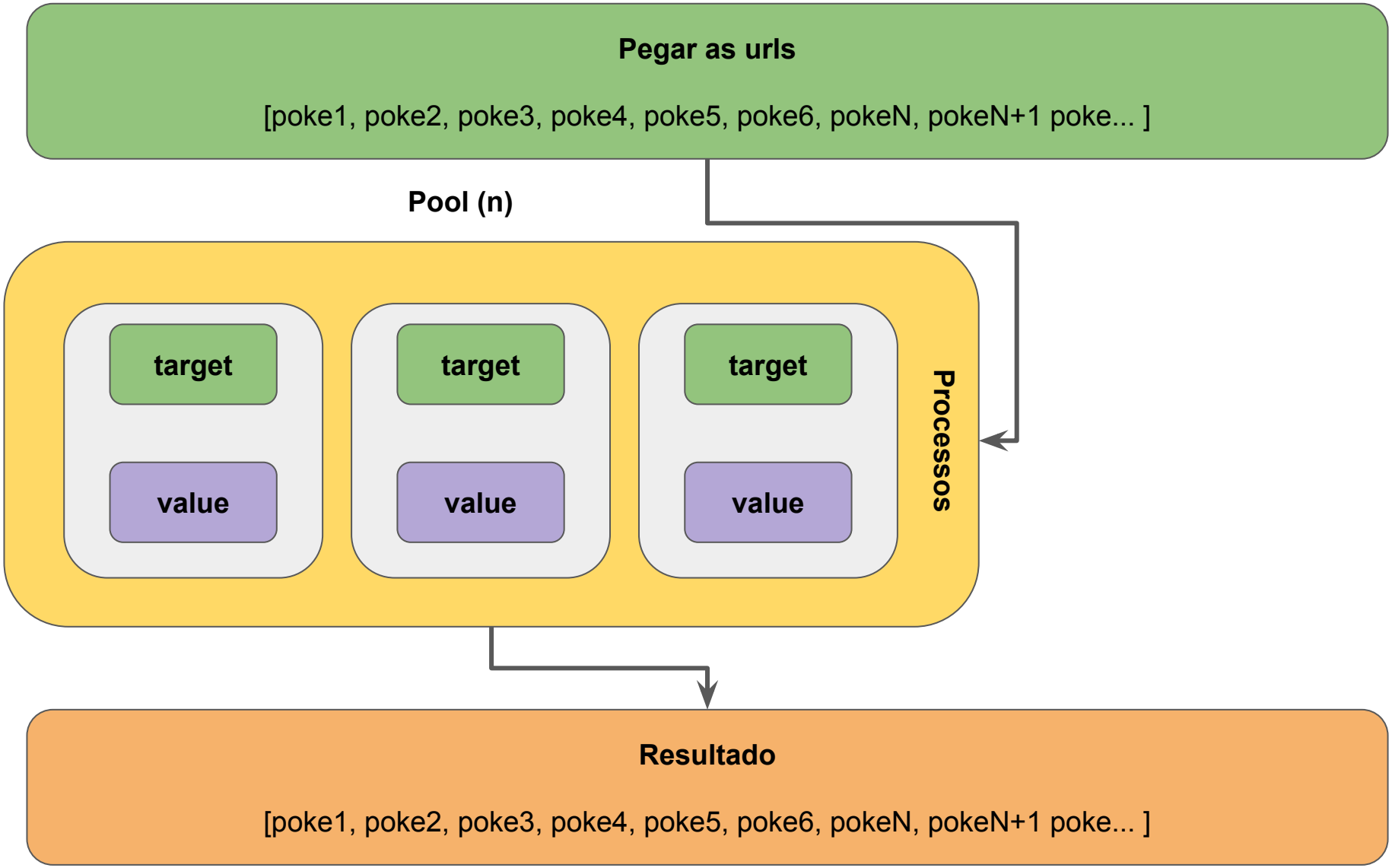
Exemplo usando o Pool

Resolução com Pool de processos

Vamos novamente compor as funções e o resultado da composição deverá ser serializável, após isso, o pool vai consumir uma lista, sem intervenção de eventos e vai consumir uma lista do Python e não uma queue



Resolução com Pool de processos



XOXO

Dúvidas?

Nome:
Eduardo Mendes

Instituição:
Unicamp / Diebold Nixdorf

Contatos:
{facebook, github, gist
instagram, linkedin,
telegram, twitter}/dunossauro