



Desenvolvimento de Aplicações Embarcadas utilizando Python

Flávio Ribeiro

III Encontro do Grupo de Usuários de Python de Pernambuco
Abril de 2010

Quem sou?

Flávio Ribeiro

Trezeano

Graduando em Engenharia Elétrica (IFPB)

Engenheiro de Software (Avaty! Tecnologia)

Python, Sistemas Embarcados, Disp. Móveis, Robótica, Automação

<http://www.flavioribeiro.com>

<http://www.twitter.com/flavioribeiro>

email@flavioribeiro.com

flavioribeiro @ freenode #python-br



Agenda

Conceitos de Sistemas Embarcados

Desenvolvimento de Software para esses Sistemas

Por que Python?

Algumas Dicas :-)



Conceitos de Sistemas Embarcados

Sistemas Embarcados são sistemas eletrônicos microprocessados (computadores) encapsulados e dedicados ao dispositivo em que reside e são desenvolvidos para exercer especialmente uma atividade específica.



Conceitos de Sistemas Embarcados

- Escassez em recursos de Processamento, Armazenamento e Autonomia
- Funcionalidade Única, executada repetidamente
- Forte Comunicação com o ambiente
- Propósito de Existência concreto
- Heterogêneos





• Divertidos de Programar e Manipular!

Desenvolvendo Para esses Sistemas

- Equilíbrio entre legibilidade x qualidade & velocidade do código



Desenvolvendo Para esses Sistemas

- Equilíbrio entre legibilidade x qualidade & velocidade do código
 - "Otimização prematura é a raiz de todo mal na programação."



Desenvolvendo Para esses Sistemas

- Equilíbrio entre legibilidade x qualidade & velocidade do código
 - "Otimização prematura é a raiz de todo mal na programação."
- Think Embedded



Desenvolvendo Para esses Sistemas

- Equilíbrio entre legibilidade x qualidade & velocidade do código
 - "Otimização prematura é a raiz de todo mal na programação."
- Think Embedded
- Conhecimento da plataforma de hardware (capacidade)



Desenvolvendo Para esses Sistemas

- Equilíbrio entre legibilidade x qualidade & velocidade do código
 - "Otimização prematura é a raiz de todo mal na programação."
- Think Embedded
- Conhecimento da plataforma de hardware (capacidade)
- Noções de Sistema Operacional e Arquitetura de Computadores



Por que Python?

- Encapsulamento e Controle de Acesso



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!
- Portabilidade



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!
- Portabilidade
- Melhor testabilidade



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!
- Portabilidade
- Melhor testabilidade
- Integração fácil com C\C++



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!
- Portabilidade
- Melhor testabilidade
- Integração fácil com C\C++
- Muito divertido de programar



Por que Python?

- Encapsulamento e Controle de Acesso
- Mais reuso do código!
- Portabilidade
- Melhor testabilidade
- Integração fácil com C\C++
- Muito divertido de programar
- Processo de desenvolvimento\teste rápido (no host)



Algumas Dicas

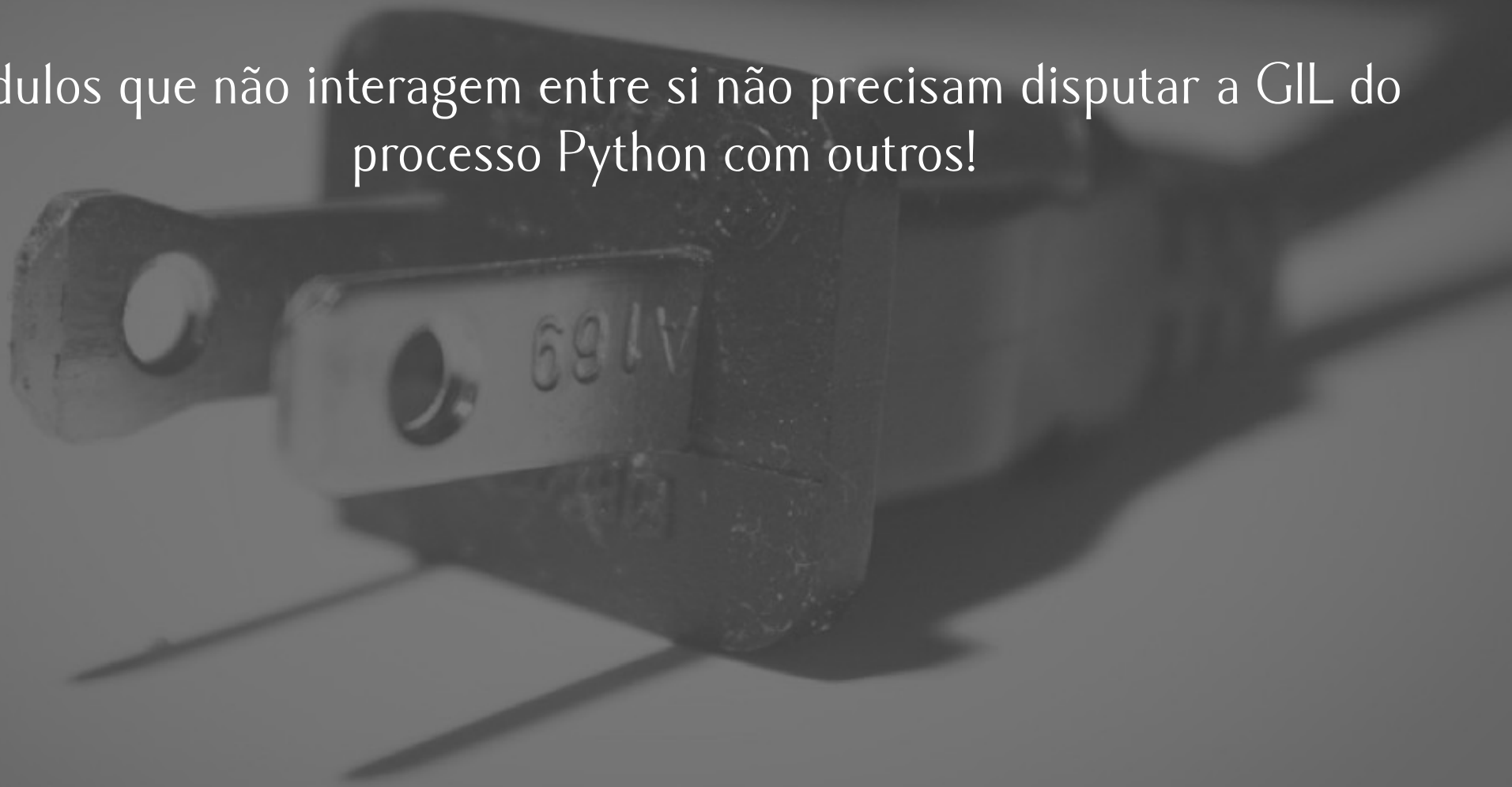


Desacople!



Desacople!

Módulos que não interagem entre si não precisam disputar a GIL do processo Python com outros!



Desacople!

Módulos que não interagem entre si não precisam disputar o GIL do processo Python com outros!

Se sua aplicação tiver muitos serviços\threads, avalie rodar mais de um processo python com serviços distintos

Faça benchmarks com o módulo multiprocessing para versões do CPython que suportam



E o que
é o GIL?



Em poucas palavras...

GIL – Global Interpreter Lock, é usado para proteger objetos Python de serem modificados entre vários processos de uma vez.

Somente o segmento que tem o bloqueio (GIL) pode acessar com segurança os objetos.

E o que
é o GIL?



“It’s simple : threads hold the GIL when running”

David Beazley (www.dabeaz.com)

E o que
é o GIL?




“It limits thread performance”

David Beazley (www.dabeaz.com)

E o que
é o GIL?

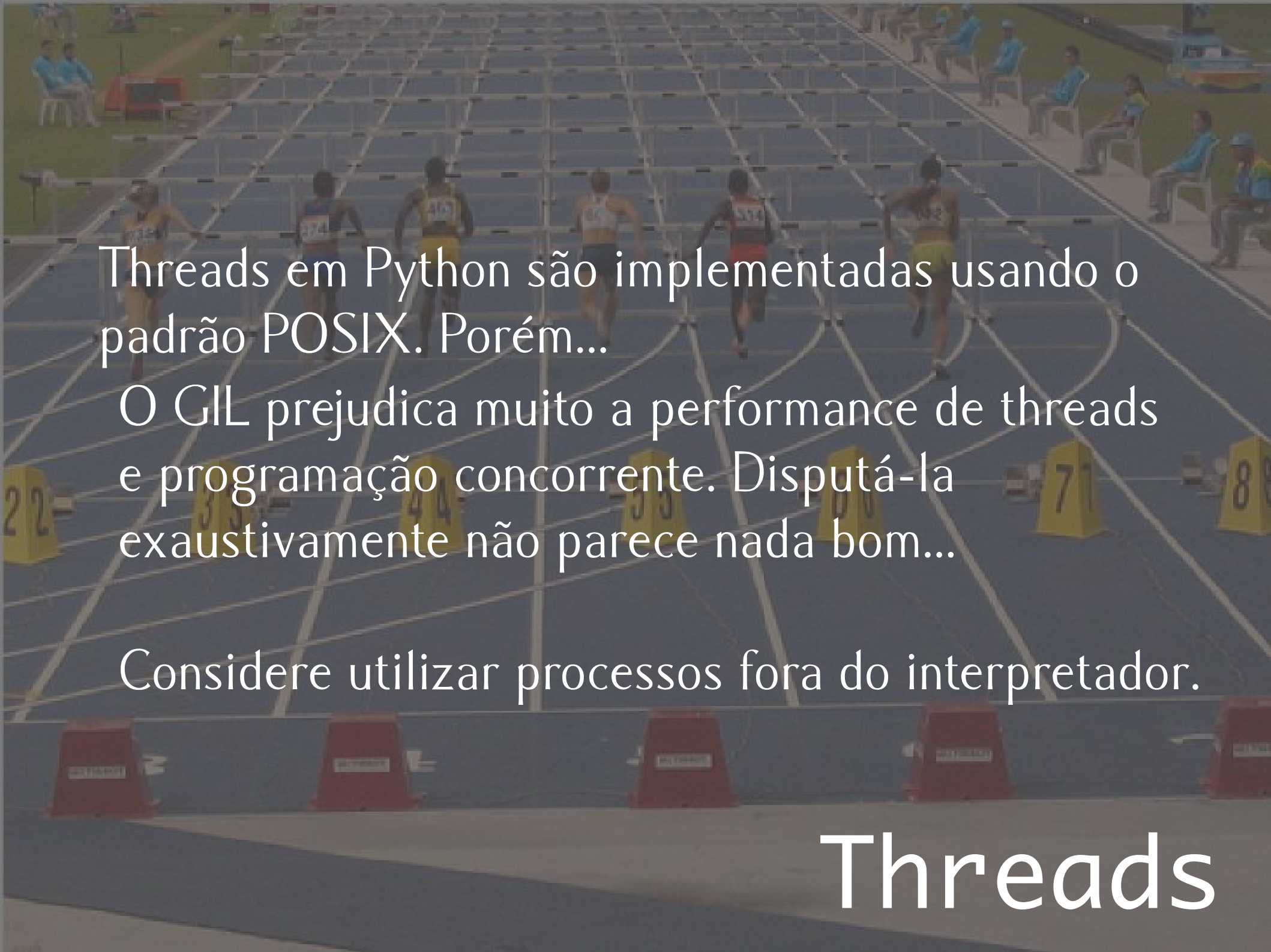


Threads



Threads em Python são implementadas usando o padrão POSIX. Porém...

Threads

A background image of a track and field race. Several runners are visible on a blue track, wearing various colored uniforms. Spectators are seated in the stands on the right side. The image is slightly blurred and has a dark overlay to make the text stand out.

Threads em Python são implementadas usando o padrão POSIX. Porém...

O GIL prejudica muito a performance de threads e programação concorrente. Disputá-la exaustivamente não parece nada bom...

Considere utilizar processos fora do interpretador.

Threads

Evite o excesso de
tratamentos de erro!



Sua classe
parece
Com algum tipo
nativo?

Herde-o!





Evite Heranças Múltiplas

Mais dicas...

- Quando testar `'a in b'`, `b` deve ser um set ou dicionário
- Concatenação de strings é melhor com `"".join(seq)`, ao invés de `+` e `+=`
- Iteradores ao invés de grandes listas
- Variáveis locais são acessadas mais rápidas que variáveis globais
- `X = 3` é bem melhor que `X = 1 + 2`
- List Comprehensions são bem melhores que for loops
- `x,y = a,b` é mais lento que `x = a; y = b`

Agora sim, otimize!



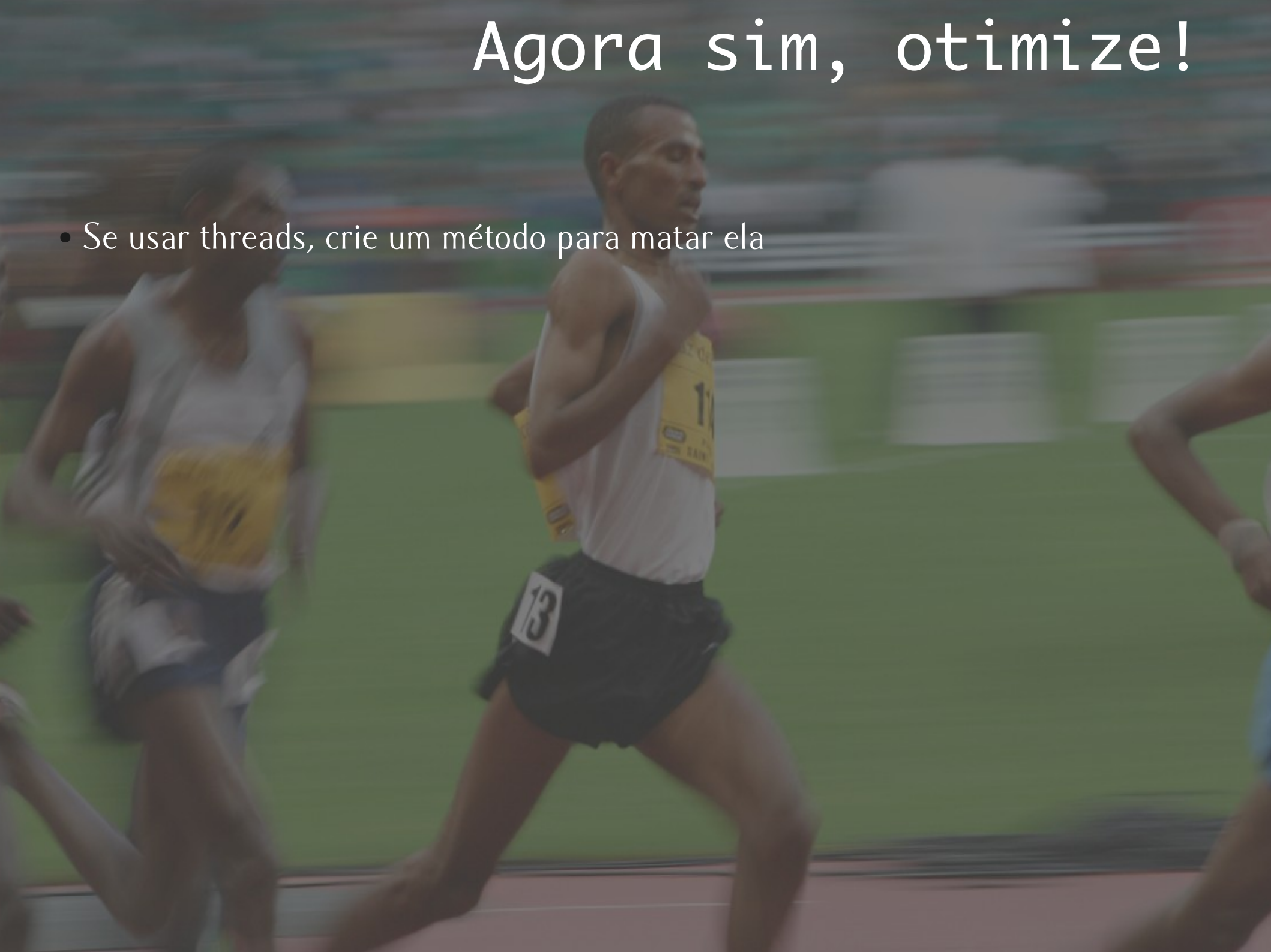
Agora sim, otimize!

- Se usar threads, crie um método para matar ela:

```
class MinhaThread(threading.Thread):  
  
    def __init__(self):  
        threading.Thread.__init__(self)  
  
        self._is_alive = True  
        self.start()  
  
    def run(self):  
        while self._is_alive:  
            time.sleep(1)  
  
    def stop(self):  
        self._is_alive = not self._is_alive
```

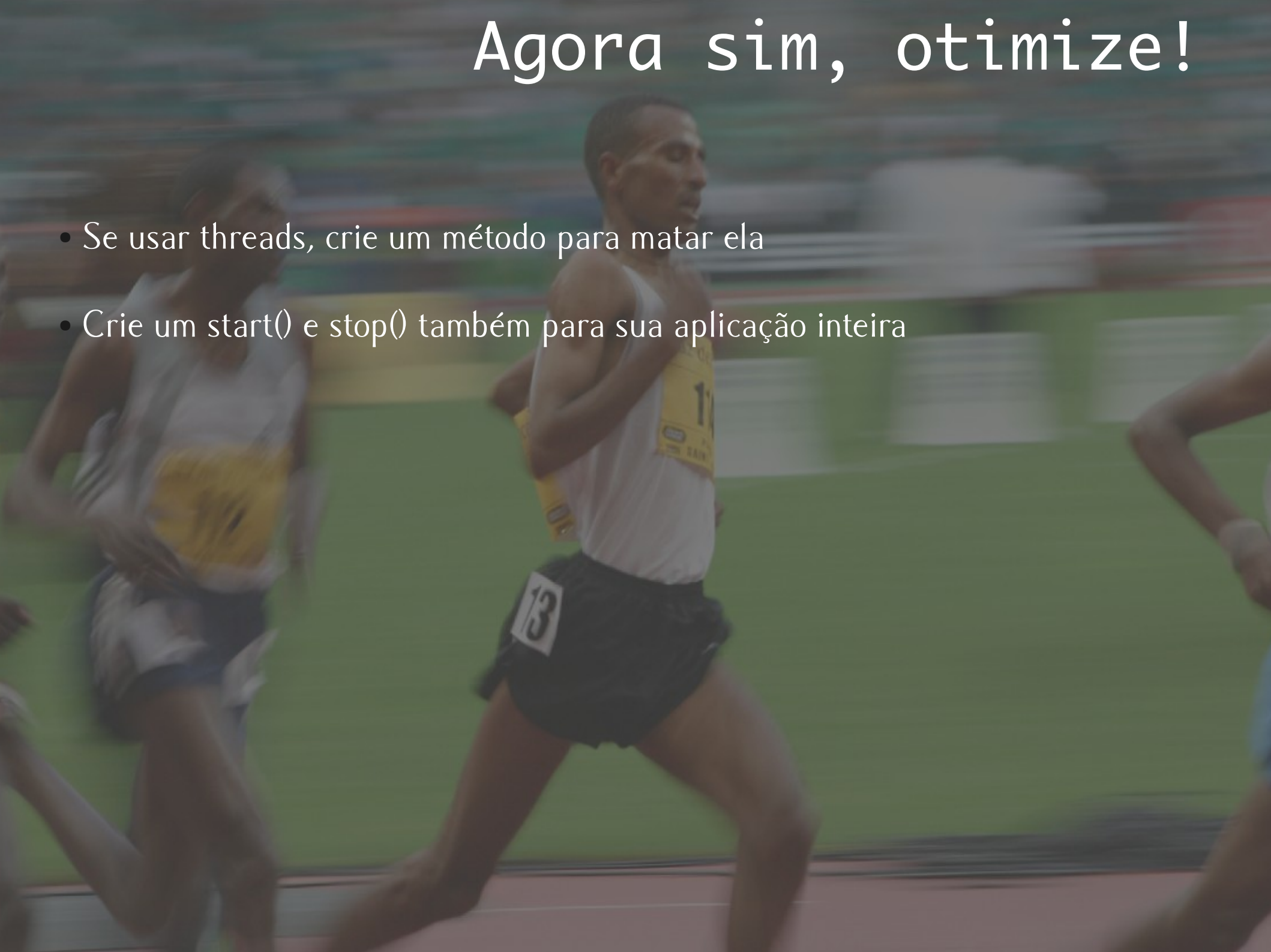

Agora sim, otimize!

- Se usar threads, crie um método para matar ela



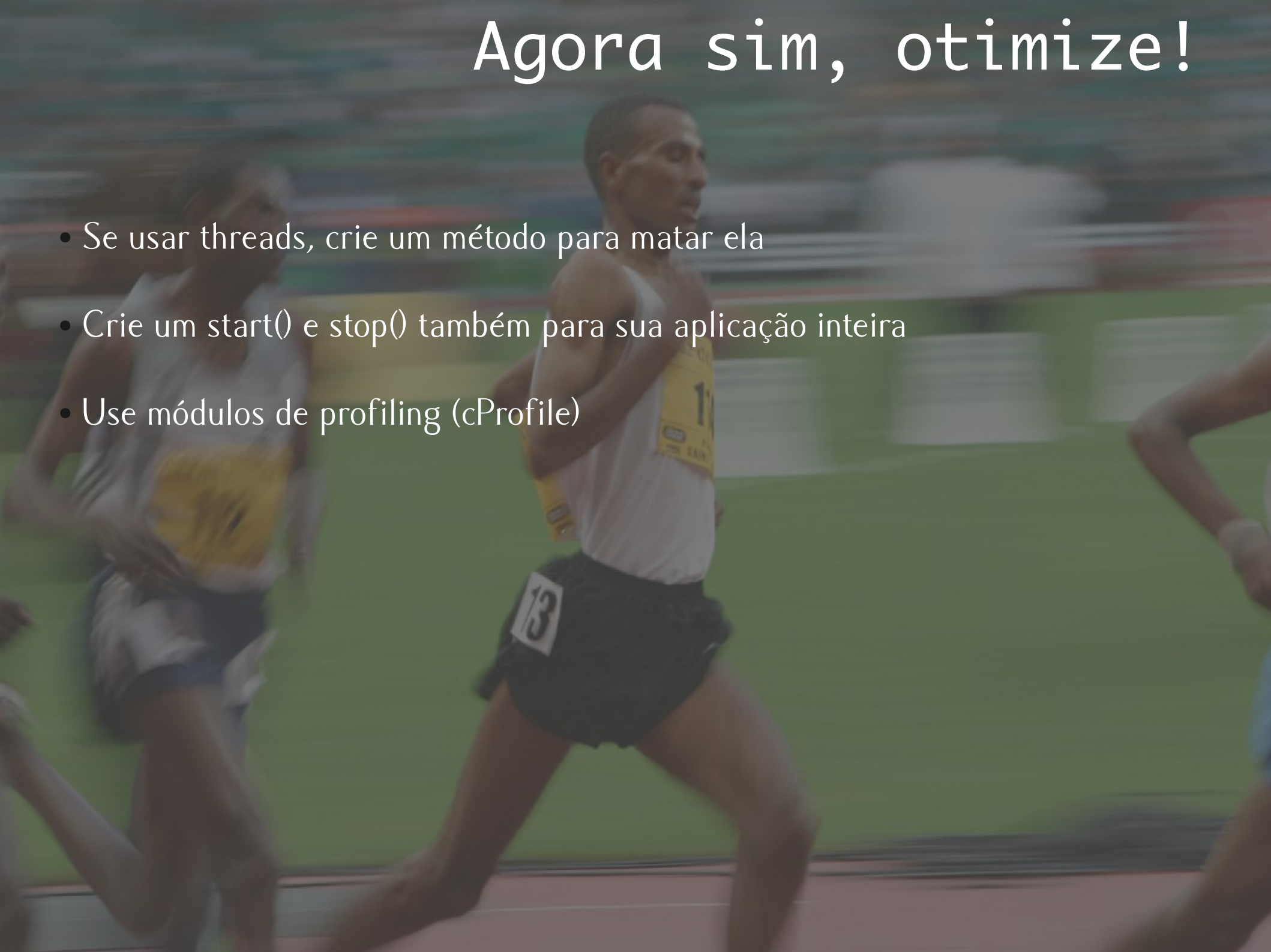
Agora sim, otimize!

- Se usar threads, crie um método para matar ela
- Crie um `start()` e `stop()` também para sua aplicação inteira



Agora sim, otimize!

- Se usar threads, crie um método para matar ela
- Crie um `start()` e `stop()` também para sua aplicação inteira
- Use módulos de profiling (cProfile)



Agora sim, otimize!

- Se usar threads, crie um método para matar ela
- Crie um start() e stop() também para sua aplicação inteira
- Use módulos de profiling (cProfile)
- Use o Gprof2Dot para parsear o arquivo de profile

Agora sim, otimize!

- Se usar threads, crie um método para matar ela
- Crie um start() e stop() também para sua aplicação inteira
- Use módulo de profiling (cProfile)
- Use o Gprof2Dot para parsear o arquivo de profile
- Analise a performance e os bottlenecks (gargálos) graficamente :-)

Dúvidas



Obrigado!

Flávio Ribeiro
email@flavioribeiro.com

Referências

<http://shreevatsa.wordpress.com/2008/05/16/premature-optimization-is-the-root-of-all-evil/>
<http://www.dabeaz.com/python/UnderstandingGIL.pdf>
<http://www.dabeaz.com/python/GIL.pdf>
<http://effbot.org/pyfaq/what-is-the-global-interpreter-lock.htm>
<http://www.flickr.com/photos/pewari/105784022/>
<http://www.flickr.com/photos/randy-shelton/1509012597/sizes//>
<http://wiki.python.org/moin/PythonSpeed>
<http://code.google.com/p/jrfonseca/wiki/Gprof2Dot>
<http://wiki.python.org/moin/PythonSpeed/PerformanceTips>
<http://www.robertostindl.de/multifaces.jpg>