ENGAGE THE HYPER-PYTHON

A RATTLE-THROUGH MANY OF THE WAYS TO MAKE YOUR PYTHON PROGRAM FASTER

DANIELE RAPATI

notonthehighstreet.com

github

gitpitch.com/danielerapati/faster-python

SLOW IS BORING

short attention span

Bottleneck / Solution	CPU	RAM	Disk	Network
Threads	x	x	V	V
Processes	v	x	V	V
Async	x	x	V	V
Pypy	V	x!!	x	x
Cython	V	x	x	x

We are going to build up to this table ...

The magic table of "not so slow anymore"

WHY SLOW?

LIMITED RESOURCES

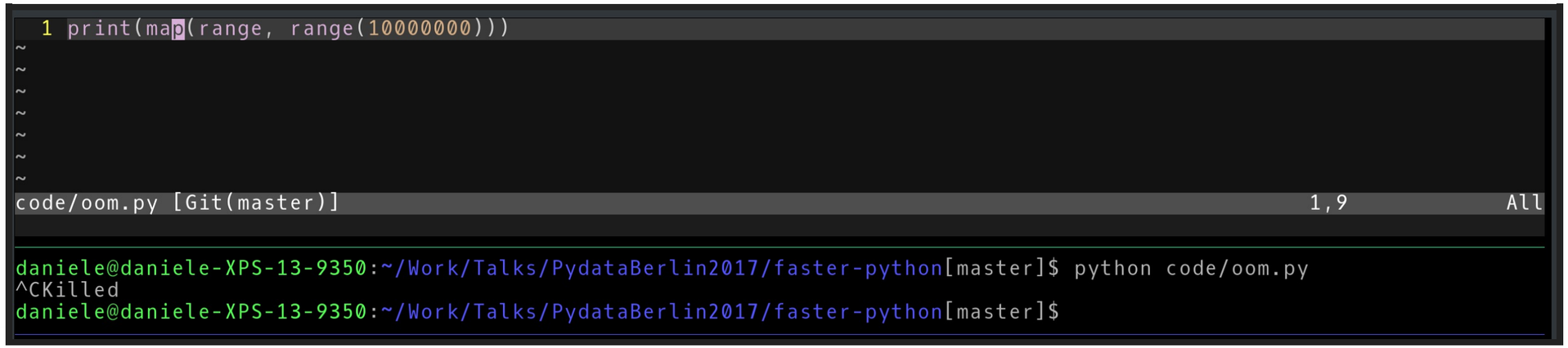
What happens when you use up a resource?

CPU

you go slow

RAM

out of memory error



DISK

I/O is slow

do you wait while reading/writing?

NETWORK

is super slow (bandwidth)

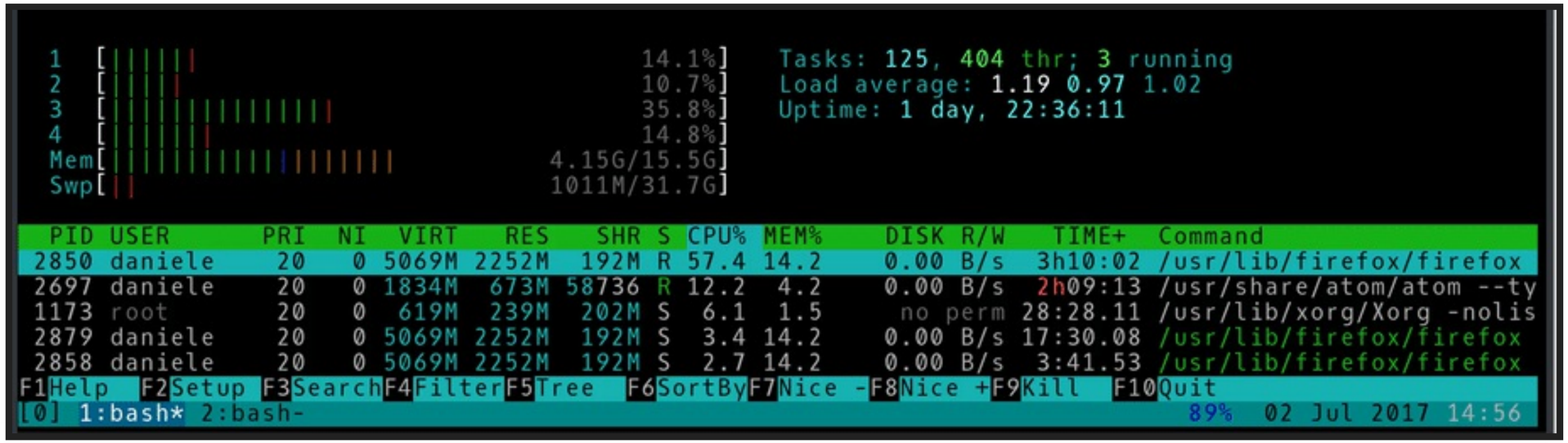
do you wait for response?

HOW TO TELL WHY

- know your program
- know it better: log
- know it even better: profile

HOW TO TELL WHY: THE EASY WAY

use a system monitor



(disclaimer: will not always work)

WHAT CAN YOU DO?

- parallelize: threads, processes and beyond
- async
- alternative interpreter (pypy)
- push to C (Cython, etc.)

THE MAGIC TABLE OF "NOT SO SLOW ANYMORE"

Bottleneck / Solution	CPU	RAM	Disk	Network
Threads	x	x	V	V
Processes	v	x	V	V
Async	x	x	V	V
Pypy	V	x!!	x	x
Cython	V	x	x	x

CONCURRENCY AND PARALLELIZATION

Ideal situation: outputs that do not require previous outputs

PARALLEL

```
def where(city):
  r = requests.get(
      'https://maps.googleapis.com/maps/api/geocode/json',
      params={"address": city}
  ).content
  return json.loads(r)['results'][0]['geometry']['location']

[where(thing) for thing in
  ['London', 'Berlin', 'Milan',
   'my plane ticket', 'my keys']]
```

SEQUENTIAL

```
def fib(n):
  if n in [0,1]: return n
  else: return fib(n-1) + fib(n-2)

def fib_for(n):
  a,b = 0,1
  for i in range(n):
    a,b = b, a + b
  return a

fib_for(200)
```

THREADS

```
#ideally

from multiprocessing.pool import ThreadPool

outputs = ThreadPool().map(operate, inputs)
```

THREADS: HOW DO THEY WORK?

independent subset of program

implementation depends on OS

shared memory

shared interpreter

THREADS: LIMITATIONS

- manual synchronization of variable values can be messy
- shared interpreter with GIL has performance overhead

THREADS

Bottleneck / Solution	CPU	RAM	Disk	Network
Threads	x	x	V	V

PROCESSES

```
from multiprocess import Pool

outputs = Pool().map(operate, inputs)
```

PROCESSES: HOW DO THEY WORK?

spawns new python interpreters (new OS process)

implementation depends on OS

PROCESSES: LIMITATIONS

- startup time
- memory footprint
- no shared memory*

PROCESSES: WATCH OUT FOR

- make your code importable

```
if __name__ == '__main__':
# create and use the Pool here
```

- avoid shared stateful objects (e.g. db connections)

PROCESSES

Bottleneck / Solution	CPU	RAM	Disk	Network
Processes	v	x	V	V

CONCURRENCY: CELERY

pip install -U celery

- many workers
- shared task queue (Redis, RabbitMQ, SQS, a DB ...)

CONCURRENCY: CELERY

multiple machines

ASYNC

```
import asyncio

async def operate(thing):
  res = await slow_async_operation(thing)
  return res

loop = asyncio.get_event_loop()

tasks = [loop.create_task(where(thing)) for thing in inputs]

loop.run_until_complete(asyncio.gather(*tasks))
```

ASYNC: HOW IT WORKS

ASYNC: BENEFITS

can handle much more concurrent tasks than with Threads

ASYNC: LIMITATIONS

- use async versions of common sync operations / libraries
- explicit exception handling
- use semaphores to limit concurrency

ASYNC

Bottleneck / Solution	CPU	RAM	Disk	Network
Async	x	x	V	V

PYPY

use instead of calling python

pip install pypy

PYPY: HOW IT WORKS

Just In Time compilation

PYPY: LIMITATIONS

- python2 (3 is in beta)
- not all libraries will work
- tuning of parameters / RAM usage

PYPY

Bottleneck / Solution	CPU	RAM	Disk	Network
Pypy	V	x!!	x	x

PUSH EXECUTION TO C

C extensions

CYTHON

pip install cython

call with cython

CYTHON: USAGE

- define function: cdef or cpdef
- add static types

CYTHON

Bottleneck / Solution	CPU	RAM	Disk	Network
Cython	V	x	x	x

THE MAGIC TABLE OF "NOT SO SLOW ANYMORE"

Bottleneck / Solution	CPU	RAM	Disk	Network
Threads	x	x	V	V
Processes	v	x	V	V
Async	x	x	V	V
Pypy	V	x!!	x	x
Cython	V	x	x	x

DANIELE RAPATI

notonthehighstreet.com

github

BONUS

THREADS CAN USE MORE THAN 1 CPU

<--- and here there is an animated gif proving it

TRADE-OFFS: CACHING IN MEMORY

use RAM instead of CPU

```
cache = {}
def fib(n):
  if n not in cache:
    if n in [0,1]: cache[n] = n
    else: cache[n] = fib(n-1) + fib(n-2)

  return cache[n]

fib(200)
```

TRADE-OFFS: CACHING IN MEMORY 2

cache information that you retrieve from network or disk

TRADE-OFFS: RAM AND DISK

OS: ram-disks vs swap

Python: mmap

ASYNC: FULL EXAMPLE

```
import asyncio
import aiohttp
import json

async def where(city):
  async with aiohttp.ClientSession() as session:
    r = await session.get(
      'https://maps.googleapis.com/maps/api/geocode/json',
      params={"address": city})
    c = await r.text()
    return json.loads(c)['results'][0]['geometry']['location']

inputs = ['London', 'Berlin', 'Milan',
          'my plane ticket', 'my keys']
```