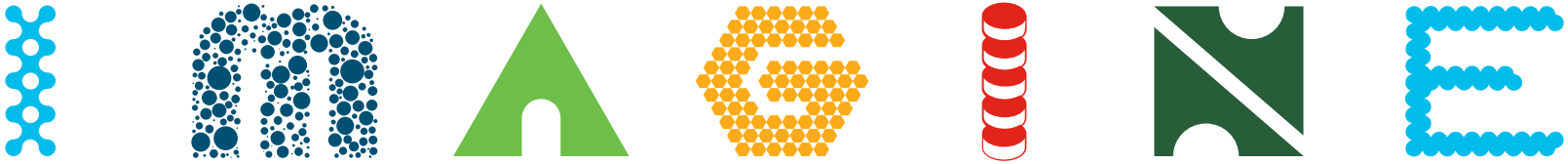
Cisco live!

January 28 - February 1, 2019 • Barcelona

cisco



INTUITIVE

DEVWKS-3627

Make Python applications faster with asyncio!

Dmitry Figol, WW Enterprise Sales

Systems Engineer – Programmability

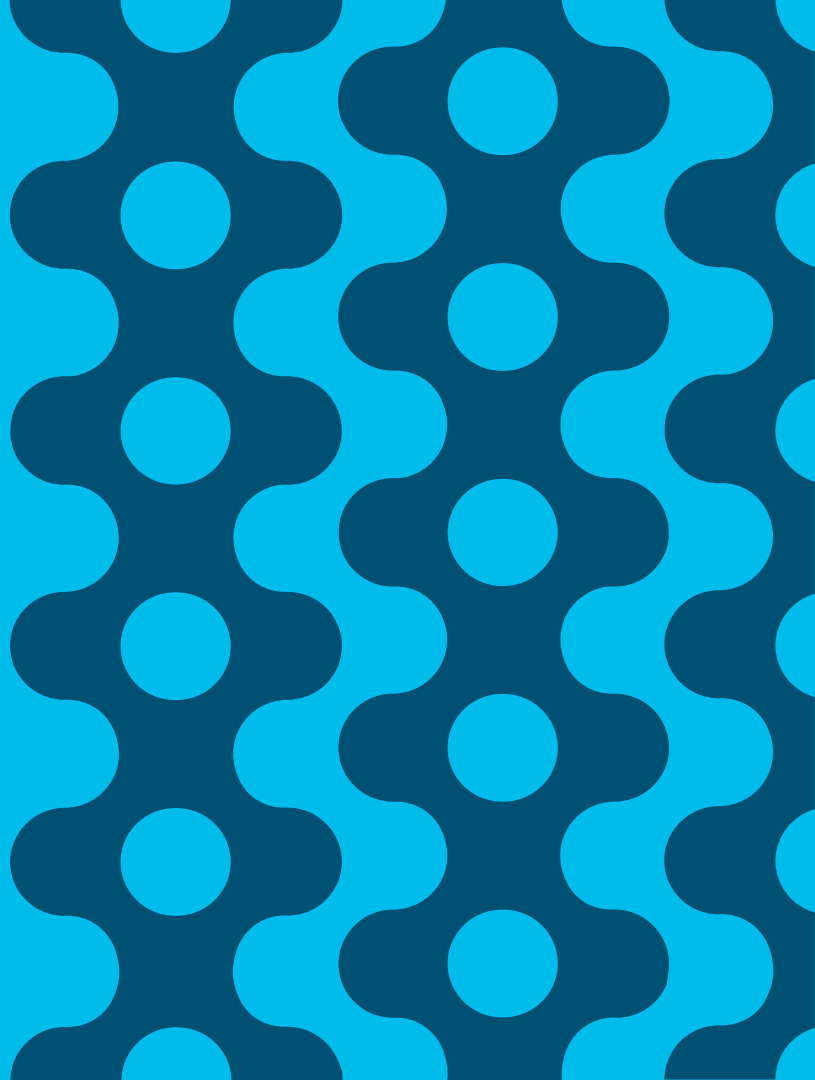
About me

@dmfigol
dmfigol
dmfigol.me
dmfigol
dmfigol

Agenda

- Python performance
- Concurrency and parallelism in Python
- asyncio library building blocks
- asyncio examples in networking

Python performance



Python speed

- Everyone says Python is slow, but …
  - Is it really?
- What is the bottleneck?
  - Often it is not CPU, it is input/output (I/O) – database queries, network

If Python was fast, it wouldn't really help

Takeaway: Python is actually slow, but it doesn't matter for many use-cases

Python implementations

- CPython (default)
- Cython
- PyPy

Use Python implementations to increase Python performance!

But this talk is about I/O

I/O in Python - scenario

- A network engineer just recently learned Python and how to interact with network devices using SSH or API
- Needs to collect memory statistics, ARP table and parse this data
- One week of trial and error… and success! Working great for one device
- …but need to do the same for 1000+ devices
- Let's add for loop and it should be fine… right?

RIGHT??

Demo – sequential SSH to network devices with netmiko

I/O can be very slow

Introducing concurrency

- Concurrency – several tasks are running at overlapping time periods
- Parallelism – several tasks happening at the same time

Typical approaches:

- Multiple processes
- Threads
- Asynchronous programming

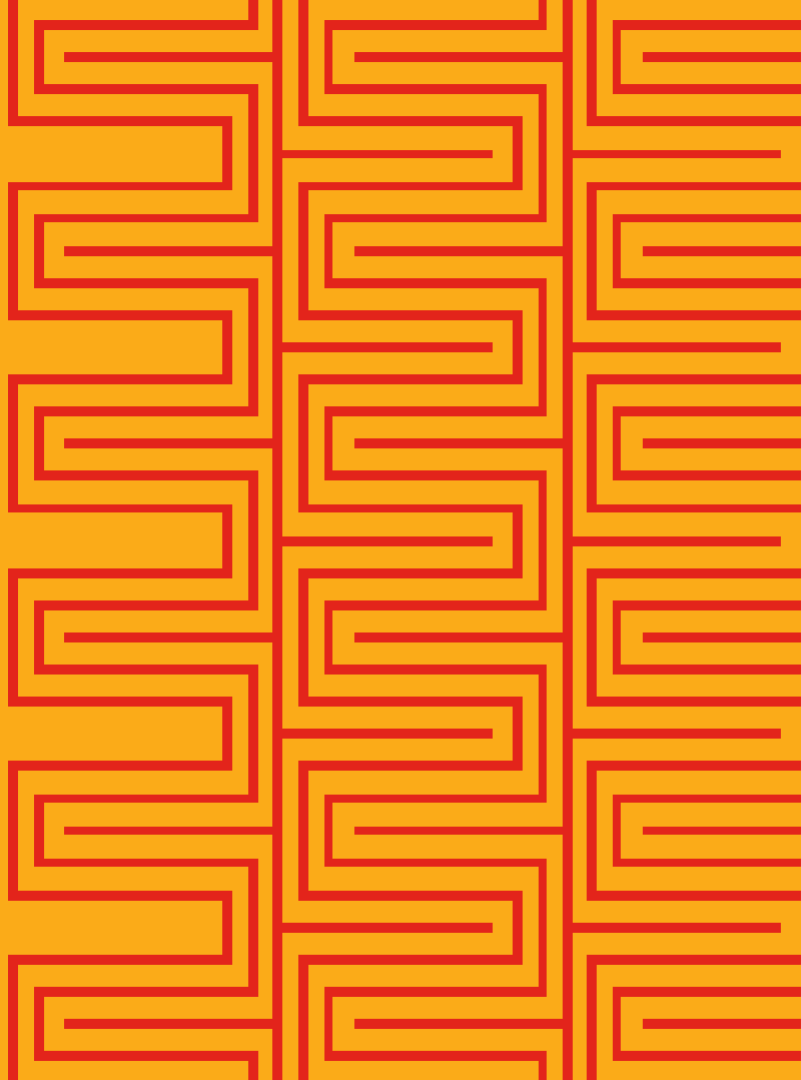
Multiprocessing in Python

- Spawns multiple Python interpreters as forks which may run across multiple CPU cores
- Higher overhead than threads
- Harder to communicate between processes
- Higher memory footprint
- Effective to distribute CPU heavy (computation) load across several cores

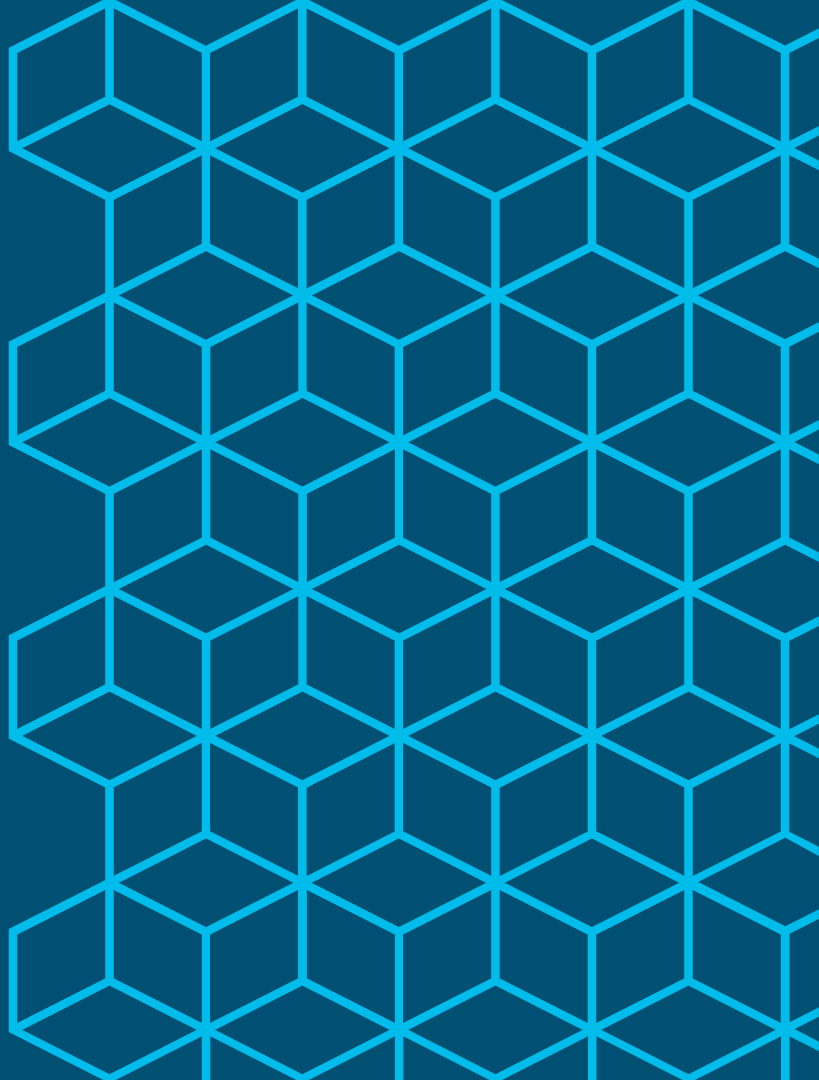
Threading in Python

- Running multiple threads (functions calls) concurrently
- Shared state
- Low overhead
- Thread switching at random time by the scheduler
- Global Interpreter Lock (GIL) – only one thread can run at the same time
  - Because of GIL, makes sense only for I/O heavy applications
- Easy to add, hard to get right – shared state can cause racing conditions
- Requires locks and queues to be safe, but most importantly experience

Demo – SSH to network device with netmiko and threads



Python asyncio



Asynchronous programming

- Intuition: when I am doing I/O and waiting for the response, I could do something else instead!
- By default, runs a single thread and single process
- Python 3.4 introduced new standard library: asyncio
- Python 3.5 introduced new syntax: async/await
- Python 3.7 – asyncio API was cleaned, new asyncio.run() entrypoint
- Completely different programming style which relies on Coroutines, Futures and Event Loop

Demo – asyncio building blocks

Coroutines

- Python 3.5 – async/await syntax for coroutines
- await expects values from I/O (e.g. HTTP response)
- I/O places in the code are marked explicitly using await
- await is used only in async coroutines marked by async def
- Usual functions can't call coroutine functions, only coroutine functions can call coroutine functions
  - This poisons the codebase, requiring coroutines everywhere
  - Loop entrypoint (`loop.run_until_complete` or `asyncio.run`) calls coroutines from synchronous context

Task/Future (asyncio)

- Task/Future – an object that wraps coroutine
  - Returned immediately
  - Placeholder for the future result or exception
  - Can be cancelled

Schedule tasks:

- `loop.create_task()`
- `asyncio.ensure_future()`
- `asyncio.create_task` – Python 3.7+ only

Event loop (asyncio)

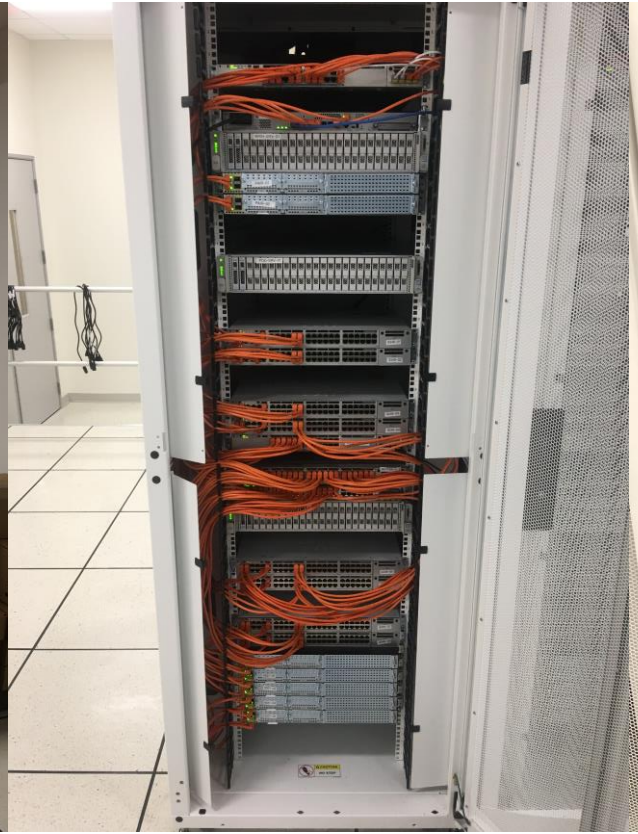
- Event loop – is a loop that executes tasks
  - Different ways to mix tasks, schedule, start and stop them
  - Directly impacts the performance
- Different loop implementations:
  - asyncio base loop – default
  - uvloop – speed increase x2-x4
  - trio event loop

Demo – asynchronous SSH to network devices with netdev

Demo – asynchronous HTTP using asyncio and aiohttp

Real world use-case

- Automated lab management system
- 5 racks of gear, 100 devices, 700+ cables



Real world use-case (cont.)

- 2 weeks for racking and stacking
- Did we actually do cabling correctly?
- Wrote a script to check that using CDP

```
-> % pipenv run python check_cabling.py
Loading .env environment variables…
Cabling 100% coincides with the documentation! Congratulations!
It took 17.43 seconds to run for 88 devices, which is 0.20 seconds per device
```

Demo – compare actual cabling with desired cabling using CDP output asynchronously

Wrap-up

Use-cases

All I/O heavy use-cases:

- Web back-end
- REST API interaction
- Database interaction
- Interaction with network devices

Problems with asyncio

- Shift in perspective and approach
- Hard to migrate existing codebase
  - Mixing synchronous and asynchronous code is complex
- Requires different libraries for I/O operations: aiohttp (HTTP client), asyncssh and netdev (SSH), aiofiles (files read/write), etc.

Summary: Why use asyncio

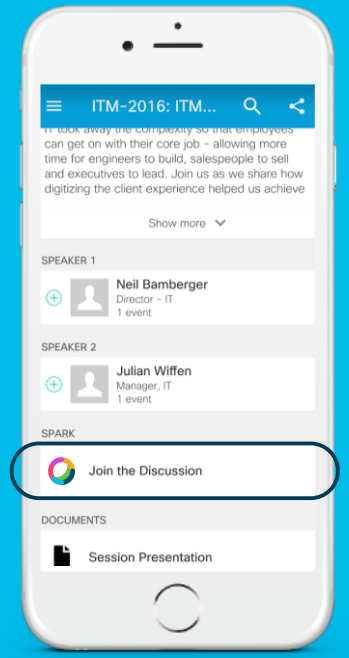
- Explicit control for task switching using await keyword
- Less overhead and slightly faster than threads
- Great scalability approach
- Easy for simple use-cases

Get Started with asyncio

- Raymond Hettinger – Keynote on Concurrency – PyBay 2017
- Async Techniques and Examples in Python course
- Miguel Grinberg – Asynchronous Python for the Complete Beginner – PyCon 2017
- Łukasz Langa – Thinking In Coroutines – PyCon 2016
- Armin Ronacher – I don't understand Python's Asyncio
- asyncio documentation
- Nathaniel J Smith – Python Concurrency for Mere Mortals – Pyninsula #10
- Useful asyncio libraries: aiohttp, asyncssh, netdev, sanic

Code

https://github.com/dmfigol/devwks-3627.git



cs.co/ciscolivebot#DEVWKS-3627

Cisco Webex Teams

Questions?

Use Cisco Webex Teams (formerly Cisco Spark) to chat with the speaker after the session

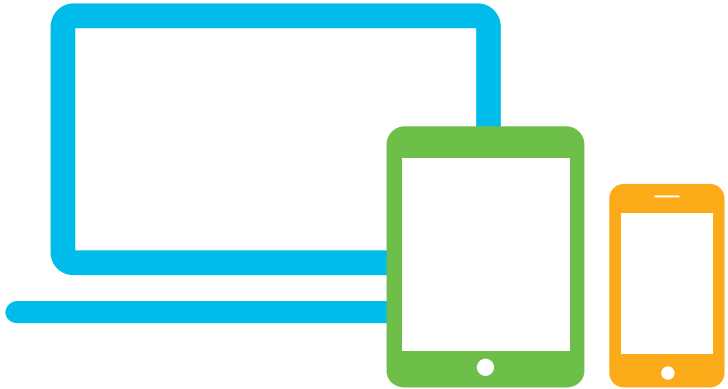
How

1. Find this session in the Cisco Events Mobile App
2. Click “Join the Discussion”
3. Install Webex Teams or go directly to the team space
4. Enter messages/questions in the team space

Complete your online session survey

- Please complete your Online Session Survey after each session
- Complete 4 Session Surveys & the Overall Conference Survey (available from Thursday) to receive your Cisco Live T-shirt
- All surveys can be completed via the Cisco Events Mobile App or the Communication Stations

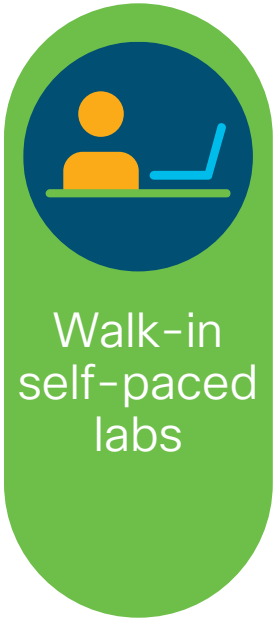
Don't forget: Cisco Live sessions will be available for viewing on demand after the event at ciscolive.cisco.com



Thank you

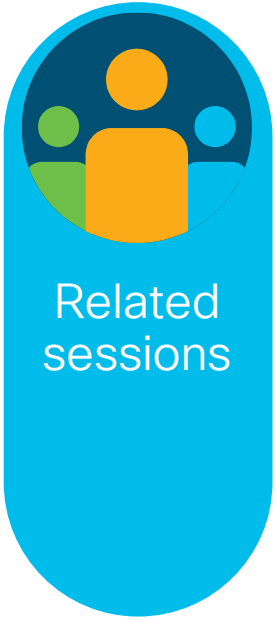
Continue Your Education

Demos in the Cisco Showcase



Walk-in self-paced labs

Meet the engineer 1:1 meetings



Related sessions

CISCO

IMAGINE

INTUITIVE