# Faster CPython

Guido van Rossum

Python Language summit

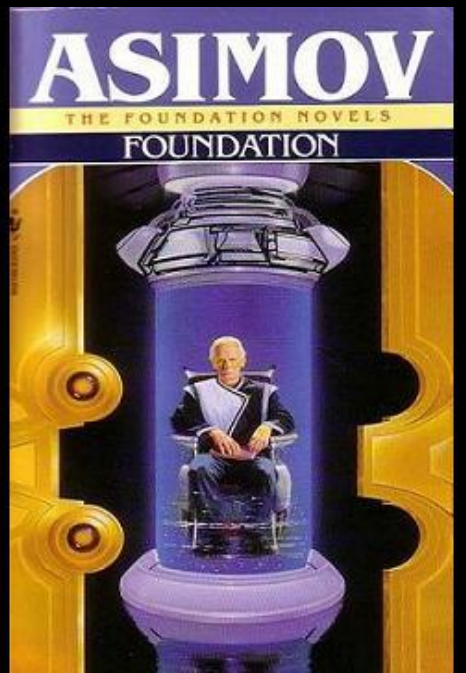
May 2021

# Can we make CPython faster?

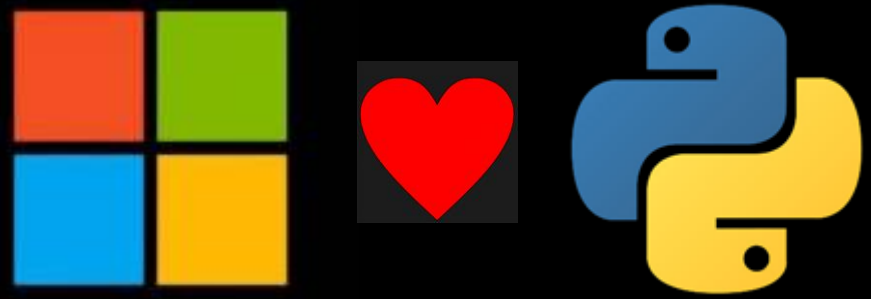
- How much can we speed up CPython?
- By a factor 2?
- By a factor 10??
- Without breaking anybody’s code?

# The “Shannon Plan”

- Posted to GitHub and python-dev last October
  - github.com/markshannon/faster-cpython
- Based on experience with “HotPy”, “HoyPy 2”
- Promises 5x in 4 years (1.5x per year)
- Looking for funding



# Thank you, Python!



- (And thank you pandemic :-)
- I got bored sitting at home while retired
- I applied at Microsoft and got hired
- I was given freedom to pick a project
- I chose to go back to my roots
- This is Microsoft’s way of giving back to Python

# Open discussions, open source

- Small team funded by Microsoft
  - Eric Snow, Mark Shannon, myself (might grow)
- Fully open collaboration with core devs
- Incremental changes to CPython
  - No long-lived forks/branches, no surprise 6,000 line PRs
  - We'll take care of maintenance and support too
- Everything open source
  - All project-specific repos open to the world
  - All discussions in trackers on open GitHub repos

# Constraints

- Don’t break stable ABI compatibility
- Don’t break limited API compatibility
- Don’t break or slow down extreme cases
  - e.g. push 1,000,000 items on eval stack
- Keep code maintainable

- This is hard!

# What can/can’t we do?

- Can’t change base object, type layout
  - E.g. must keep reference counting semantics
- Can change bytecode, stack frame layout
- Can change compiler, interpreter
- Can change most objects’ internals
  - Not all objects’ layout is public

# How to reach 2x speedup in 3.11

- Adaptive, specializing bytecode interpreter
  - Like inline cache but more comprehensive
- Various other separate optimizations
  - E.g. optimize frame stack, faster calls, tweak allocation
  - “Zero overhead” exception handling
- We’re far from certain we will reach 2x!
  - But optimistic and curious

# Other stuff we could do

- Improve startup time
- Change pyc file format (no more marshal?)
- Faster integer internals
- Put __dict__ at a fixed offset (-1?)
- “Hidden classes”
- Tagged numbers (but... ABI issues)

# Plans post 3.11

- Depends on what we manage for 3.11
- For 5x speedup we'll have to be creative
- There's machine code generation in our future
- Maybe evolve stable ABI/ limited API

# Who will benefit

- Users running CPU-intensive pure Python code
- Users of websites built in Python
- Users of tools that happen to use Python

# Not much benefit

- Code that’s already in C
  - numpy, tensorflow etc.
- I/O-bound code
- Multi-threading code
- Code that’s algorithmically inefficient

# Where’s our stuff

- PEP 659: Specializing Adaptive Interpreter
- repos (in a new org, faster-cpython):
  - github.com/faster-cpython/cpython
    - fork of cpython, for staging branches
  - github.com/faster-cpython/ideas
    - tracker for discussions
  - github.com/faster-cpython/tools
    - tools for analysis, benchmarking, whatever

# Q&A