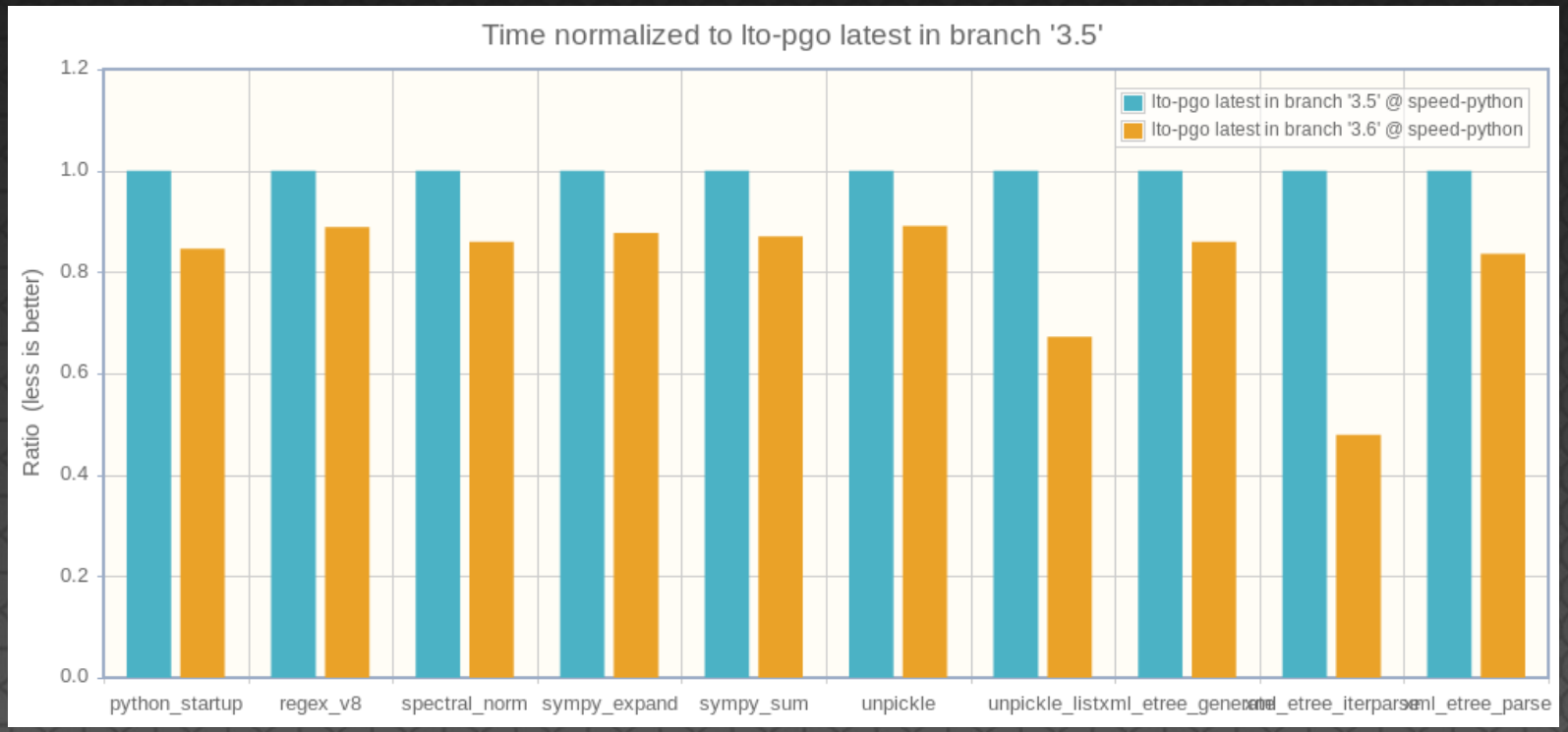
Optimizations which made Python 3.6 faster than Python 3.5



Pycon US 2017, Portland, OR

redhat®

Victor Stinner
vstinner@redhat.com

Agenda

(1) Benchmarks

(2) Benchmarks results

(3) Python 3.5 optimizations

(4) Python 3.6 optimizations

(5) Python 3.7 optimizations

Agenda

(1) Benchmarks

(1) Unstable benchmarks

- March 2016, no developer trusted the Python benchmark suite
- Many benchmarks were unstable
- It wasn't possible to decide if an optimization makes CPython faster or not...

(1) perf module

- New **perf** project: spawn **20 processes**, each runs the benchmark once to warmup and then run it 3 times
- Total of 60 timings: compute **average** (mean) and **standard deviation**
- Statistics to check for outliers, min/max, percentiles, etc.

(1) performance project

- Benchmarks rewritten using perf: new project **performance** on GitHub
- `http://speed.python.org` now runs performance
- CPython is now compiled with Link Time Optimization (**LTO**) and Profile Guided Optimization (**PGO**)

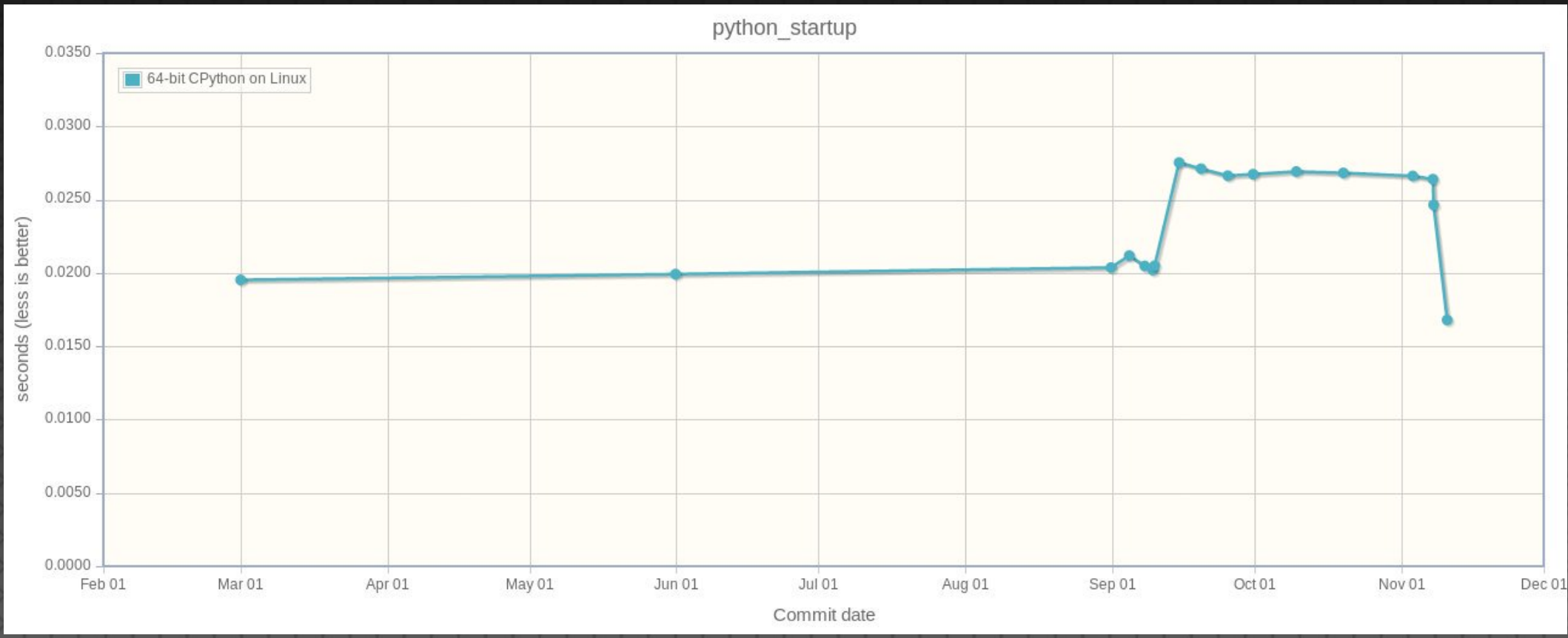
(1) Linux and CPUs

- **`python3 -m perf system tune`**
- Use fixed CPU frequency, disable Intel Turbo Boost
- Use CPU isolation (Linux `isolcpus` and `rcu_nocbs`), pin benchmark process and IRQs on CPUs
- Reduce Linux perf sampling rate
- Laptop: check that power cable is plugged

redhat

python™

(1) Spot perf regression

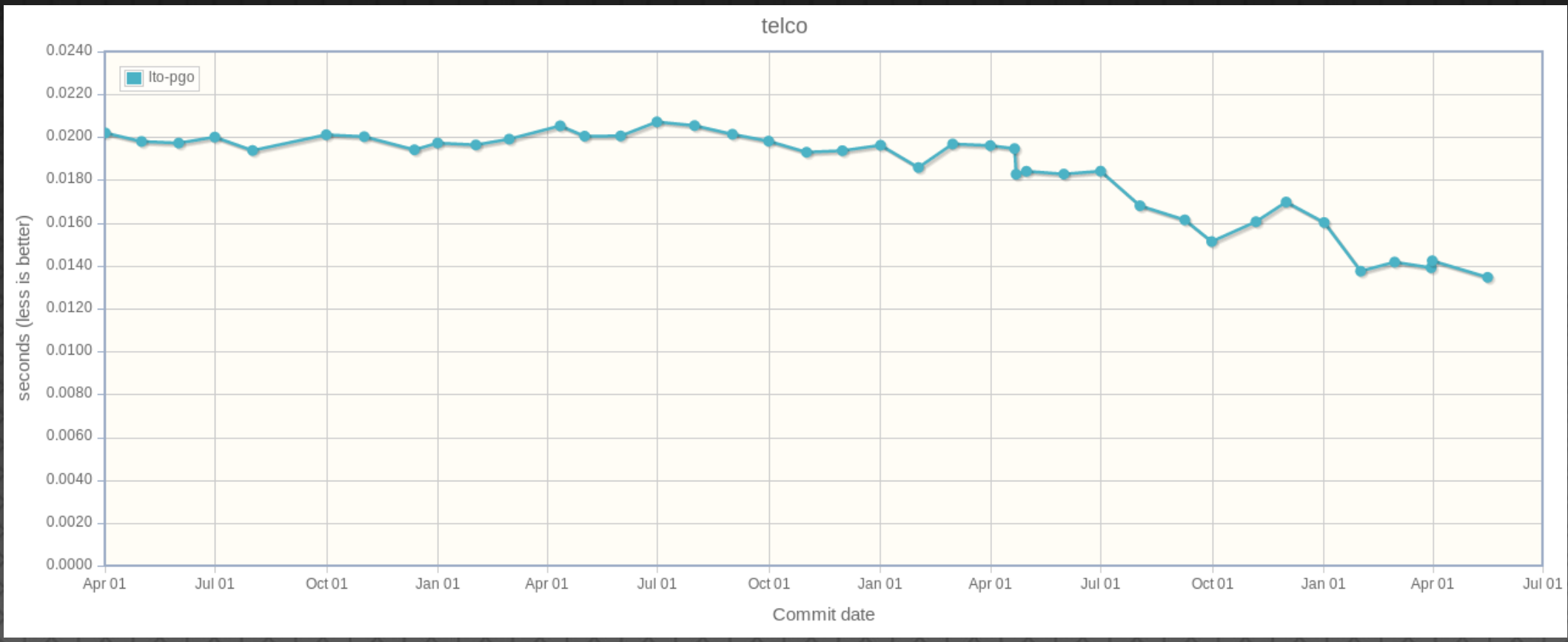


python_startup: 20 ms => 27 ms => 17 ms

redhat

python™

(1) Timeline



April, 2014 – May, 2017: 3 years

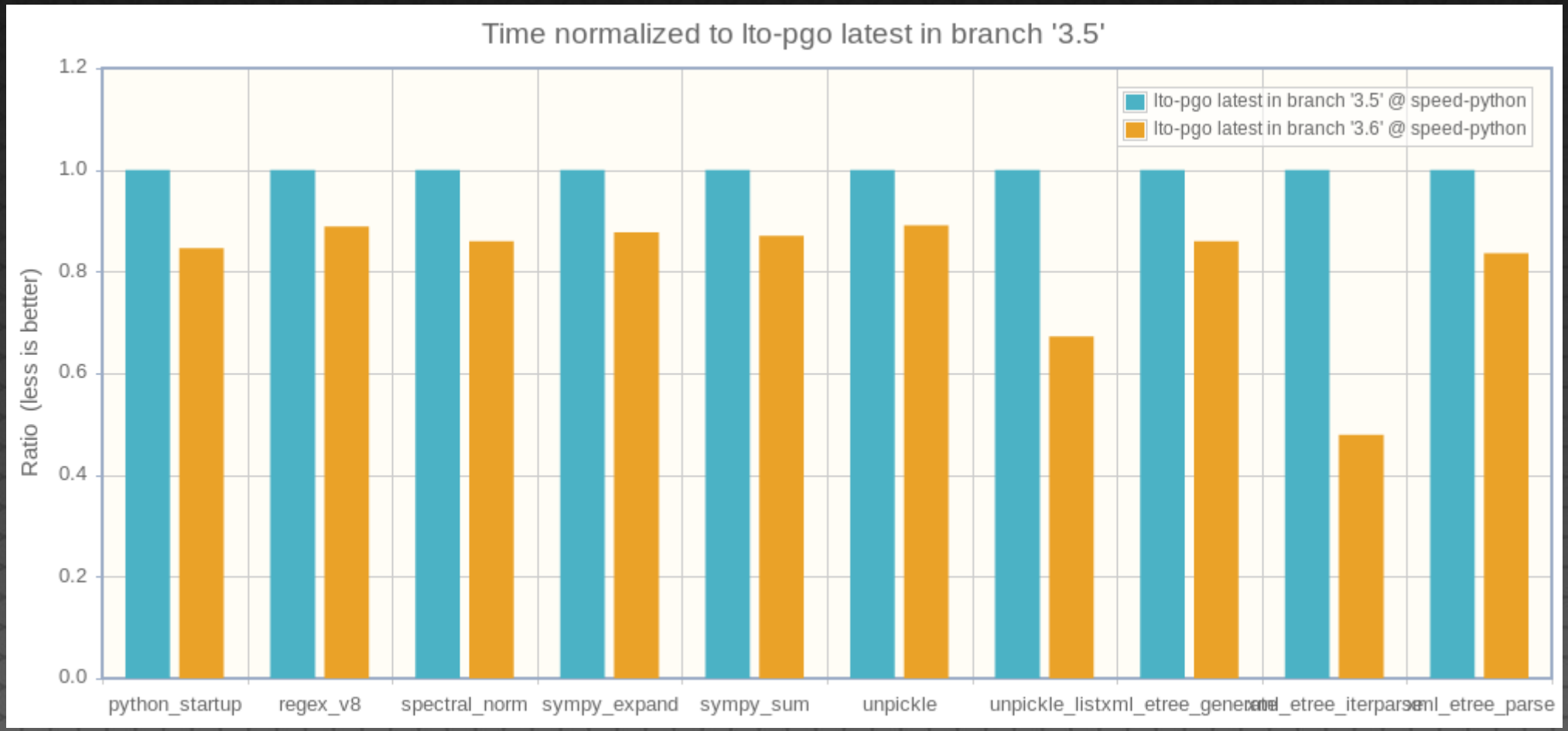
redhat

python™

Agenda

(2) Benchmarks results

(2) 3.6 faster than 3.5



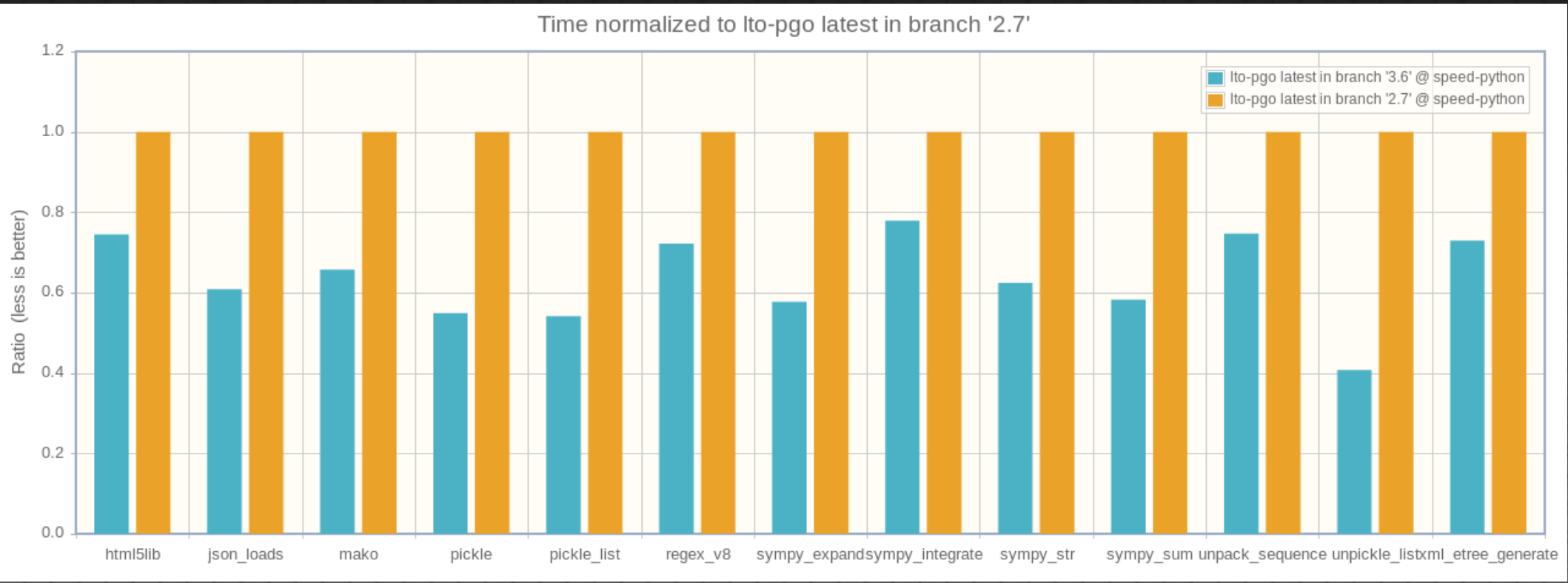
Results normalized to Python 3.5

lower = faster

redhat

python™

(2) 3.6 faster than 2.7



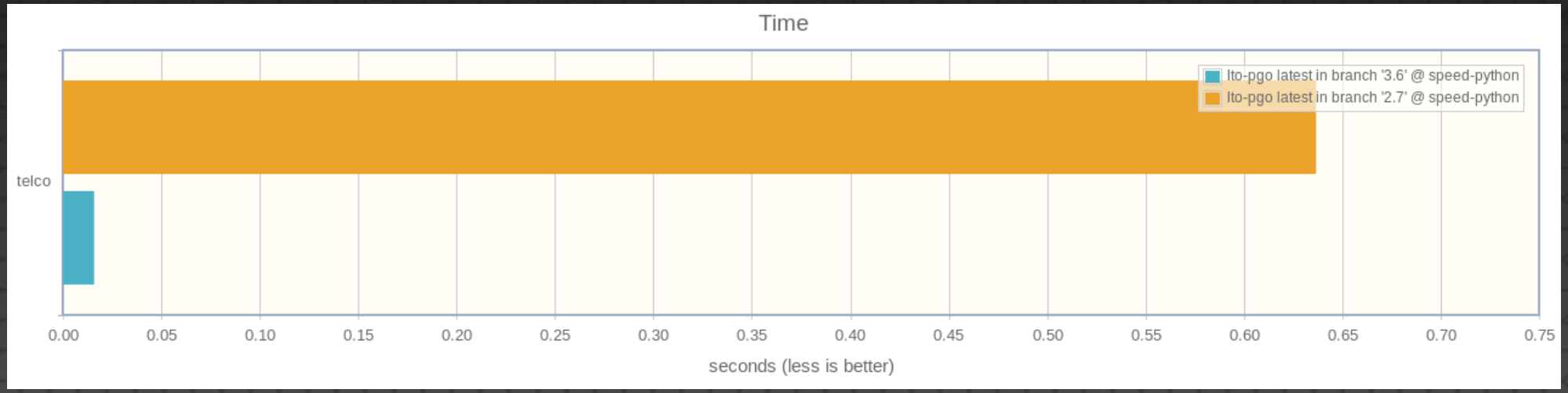
Results normalized to Python 2.7

lower = **faster**

redhat

python™

(2) telco: 3.6 vs 2.7

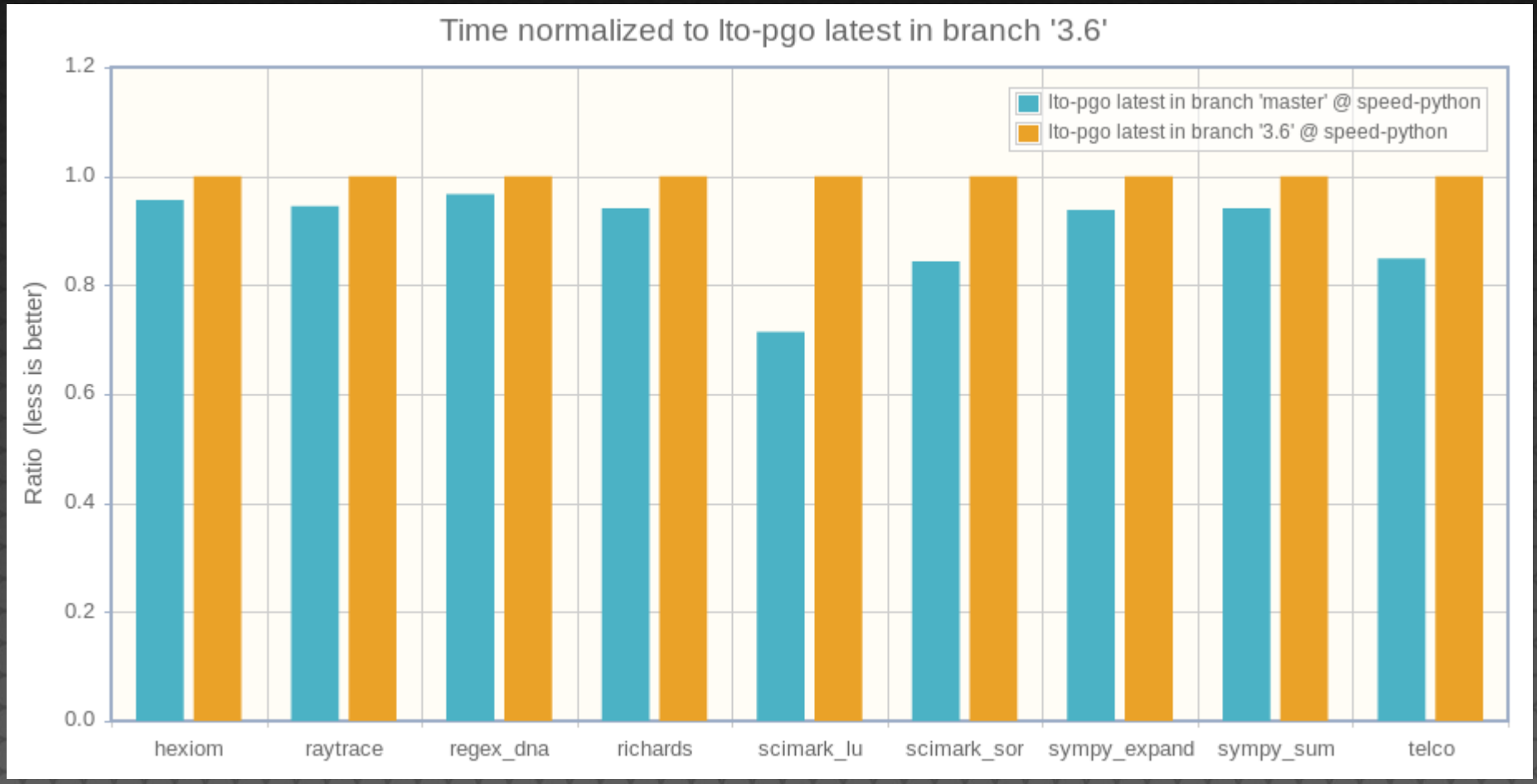


Python 3.6 is **40x** faster than Python 2.7
`decimal` module rewritten in C

redhat

python™

(2) 3.7 faster than 3.6



Results normalized to Python 3.6

lower = faster

redhat

python™

Agenda

(3) Python 3.5 optimizations

(3) C lru_cache()

- Matt Joiner, Alexey Kachayev and Serhiy Storchaka reimplemented `functools.lru_cache()` in C
- sympy: 1.2x faster
- scimark_lu: 1.06x faster
- bpo-14373

(3) C OrderedDict

- Eric Snow reimplemented collections.OrderedDict in C
- html5lib: 1.2x faster
- bpo-16991

Agenda

(4) Python 3.6 optimizations

(4) PyMem_Malloc()

- Victor Stinner modified `PyMem_Malloc()` to reuse Python fast memory allocator
- Many benchmarks: **1.05x - 1.36x faster**
- **PYTHONMALLOC=debug** now available in release builds to detect memory corruptions
- bpo-26249

redhat

python™

(4) ElementTree parse

- Serhiy Storchaka optimized ElementTree.iterparse()
- 2x faster
- bpo-25638

(4) PGO uses tests

- Brett Canon modified the build system to guide the compiler using the Python test suite rather than pidigits for the Profile Guided Optimization (PGO)
- Many benchmarks: 1.05x - 1.28x faster
- bpo-24915

(4) Wordcode

- Demur Rumed and Serhiy Storchaka modified the bytecode to always use 16-bit instructions:
  8 bit op, 8 bit arg (arg=0 if unused)
- Removed an if from ceval.c hotcode for **better CPU branch prediction**:

```
if (HAS_ARG(opcode))
    oparg = NEXTARG();
```

- bpo-26647

redhat.

python™

(4) FASTCALL

- Victor Stinner wrote a new C API to avoid the creation of temporary tuples to pass function arguments
- Many microbenchmarks: 1.13x - 1.92x faster
- `obj[0]`, `getattr(obj, "attr")`, `{1: 2}.get(1)`, …
- Avoid 20 ns per modified function call

redhat

python™

(4) Unicode codecs

- Victor Stinner optimized ASCII and UTF-8 codecs for `ignore`, `replace`, `surrogateescape` and `surrogatepass` error handlers
- UTF-8: decoder 15x faster, encoder 75x faster
- ASCII: decoder 60x faster, encoder 3x faster

redhat

python™

(4) bytes % args

- Victor Stinner wrote a new private `_PyBytesWriter` API to optimize functions creating bytes and bytearray strings
- bytes % args: 2x faster
- bytes.fromhex(): 3x faster

(4) Globbing

- Serhiy Storchaka optimized glob.glob(), glob.iglob() and pathlib globbing
- glob: **3x - 6x faster**
- Pathlib glob: **1.5x - 4x faster**
- bpo-25596, bpo-26032

(4) C asyncio

- Yury Selivanov and Naoki INADA reimplemented asyncio Future and Task classes in C
- Asyncio programs: 1.3x faster
- bpo-26081, bpo-28544

Agenda

(5) Python 3.7 optimizations

(5) Method calls

- Yury Selivanov and Naoki INADA added LOAD_METHOD and CALL_METHOD opcodes
- Methods calls: 1.2x faster
- Idea coming from PyPy, bpo-26110

(5) Future optimizations

- More optimizations are coming in Python 3.7...
- Stay tuned!

Questions?

http://speed.python.org/

http://faster-cpython.readthedocs.io/