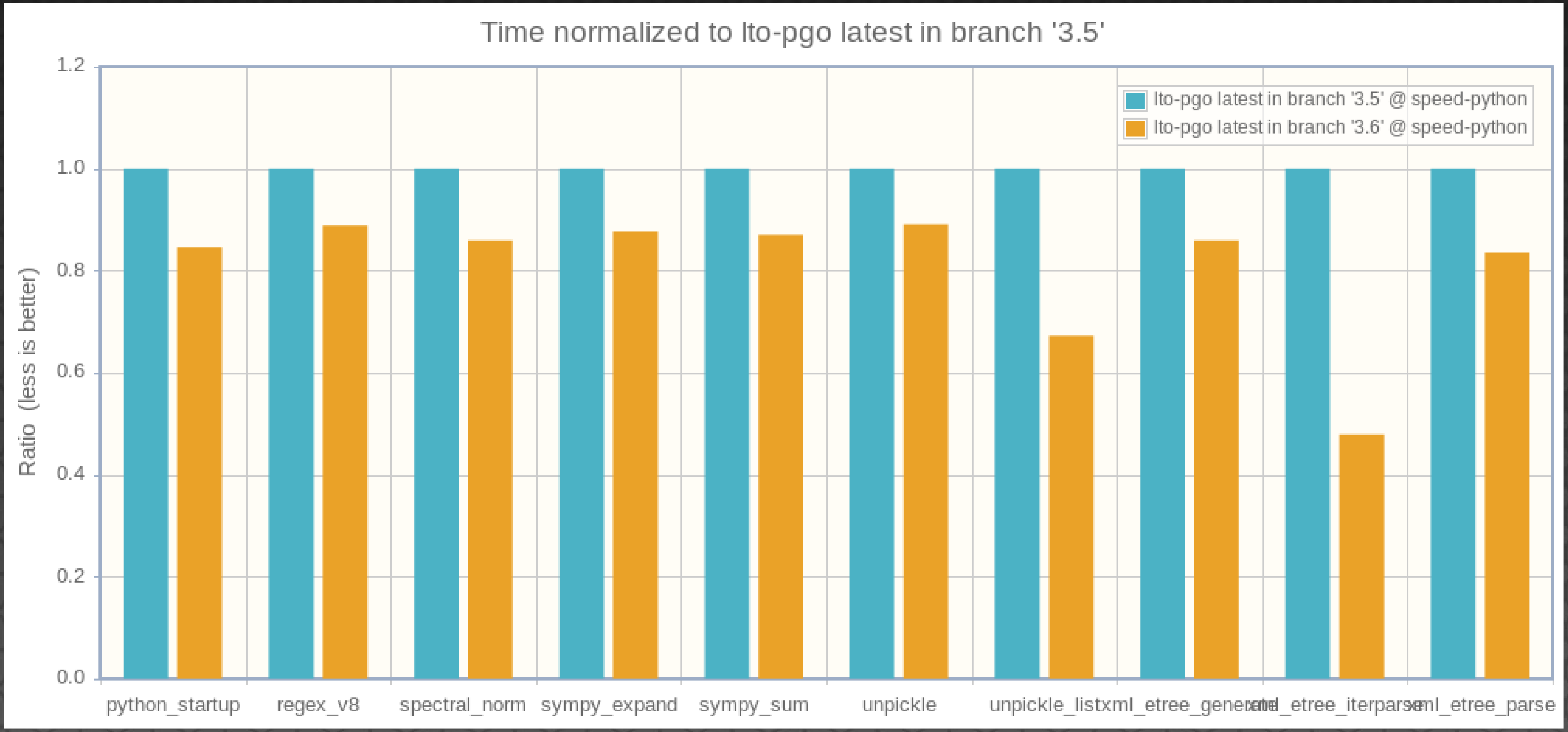
Optimizations which made Python 3.6 faster than Python 3.5



Pycon US 2017, Portland, OR

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

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...

New perf module

- Calibrate the number of loops
- Spawn 20 processes sequentially, 3 values per process, total: 60 values
- Compute average (mean) and standard deviation

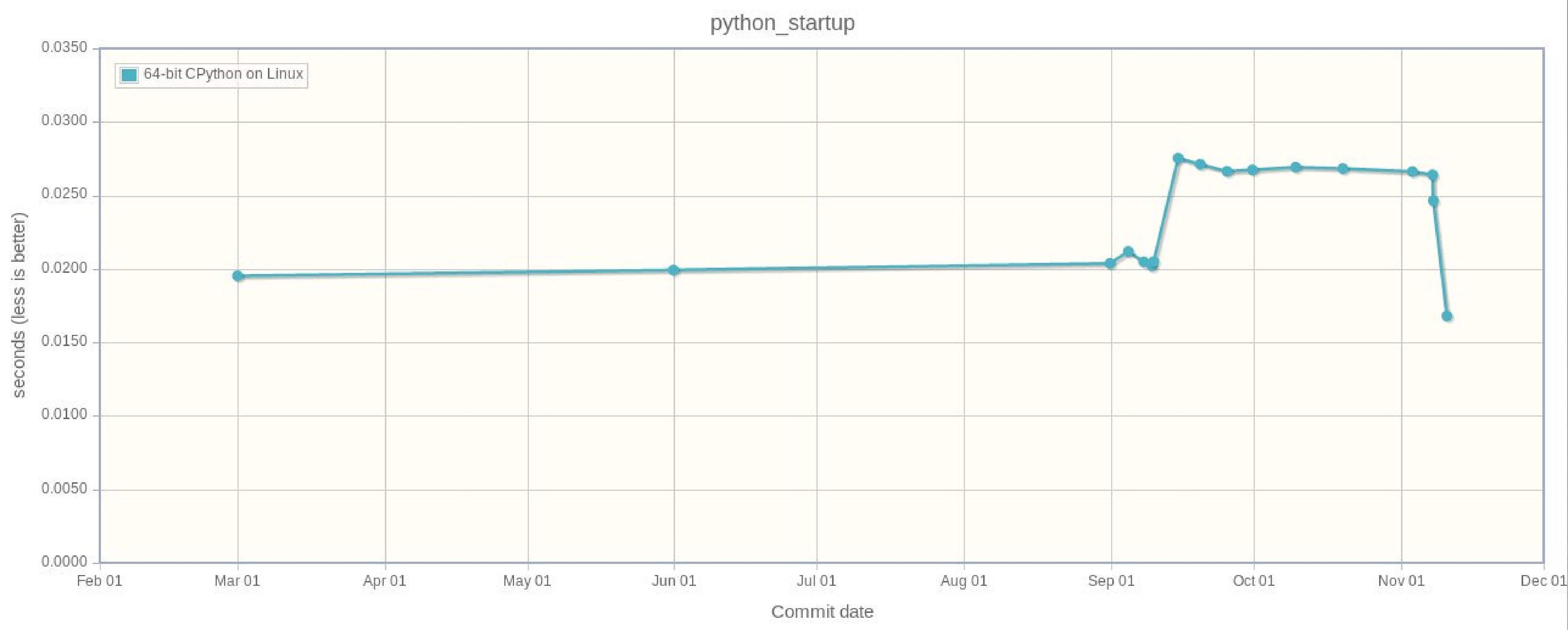
performance project

- 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**)

Linux and CPUs

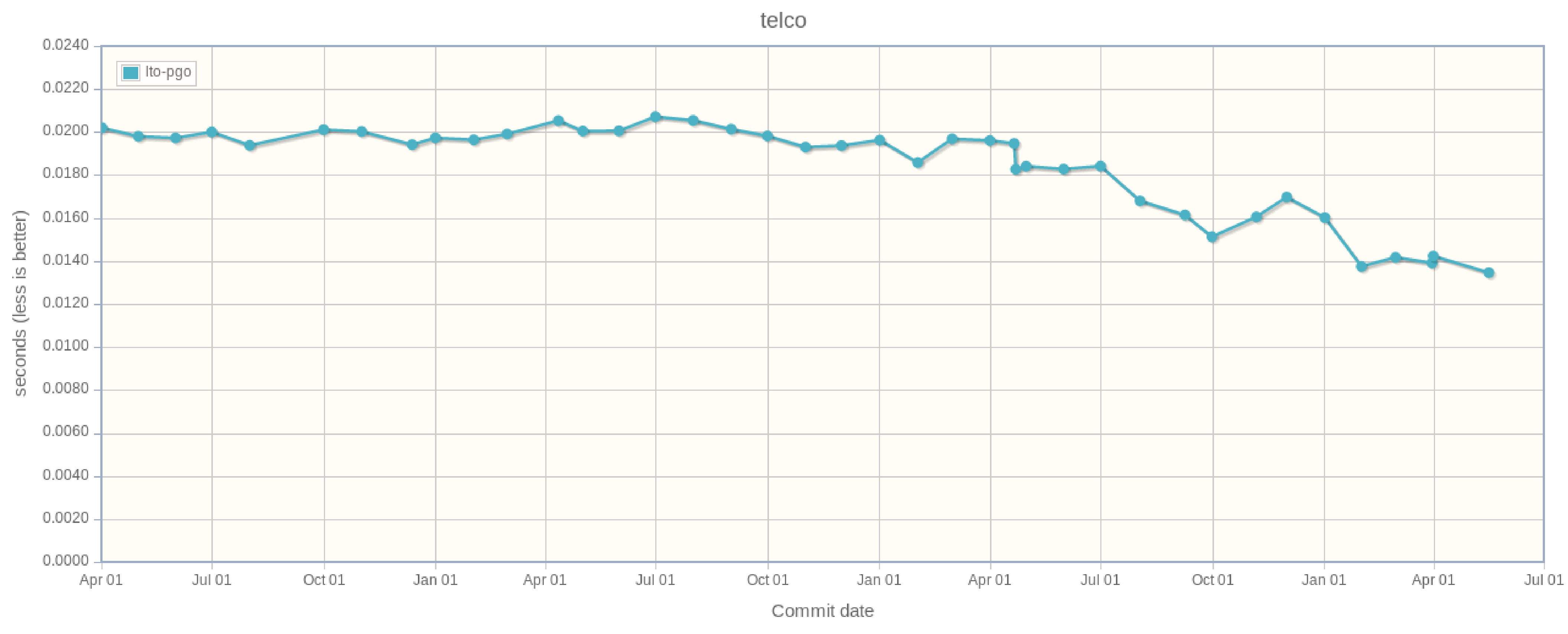
- `sudo python3 -m perf system tune`
- Use fixed CPU frequency
- Disable Intel Turbo Boost
- If CPU isolation is enabled, Linux kernel options `isolcpus` and `rcu_nocbs`, use CPU pinning
- CPU isolation helps a lot to reduce operation system jitter

Spot perf regression



python_startup: **20 ms => 27 ms**, fix: 17 ms

Timeline

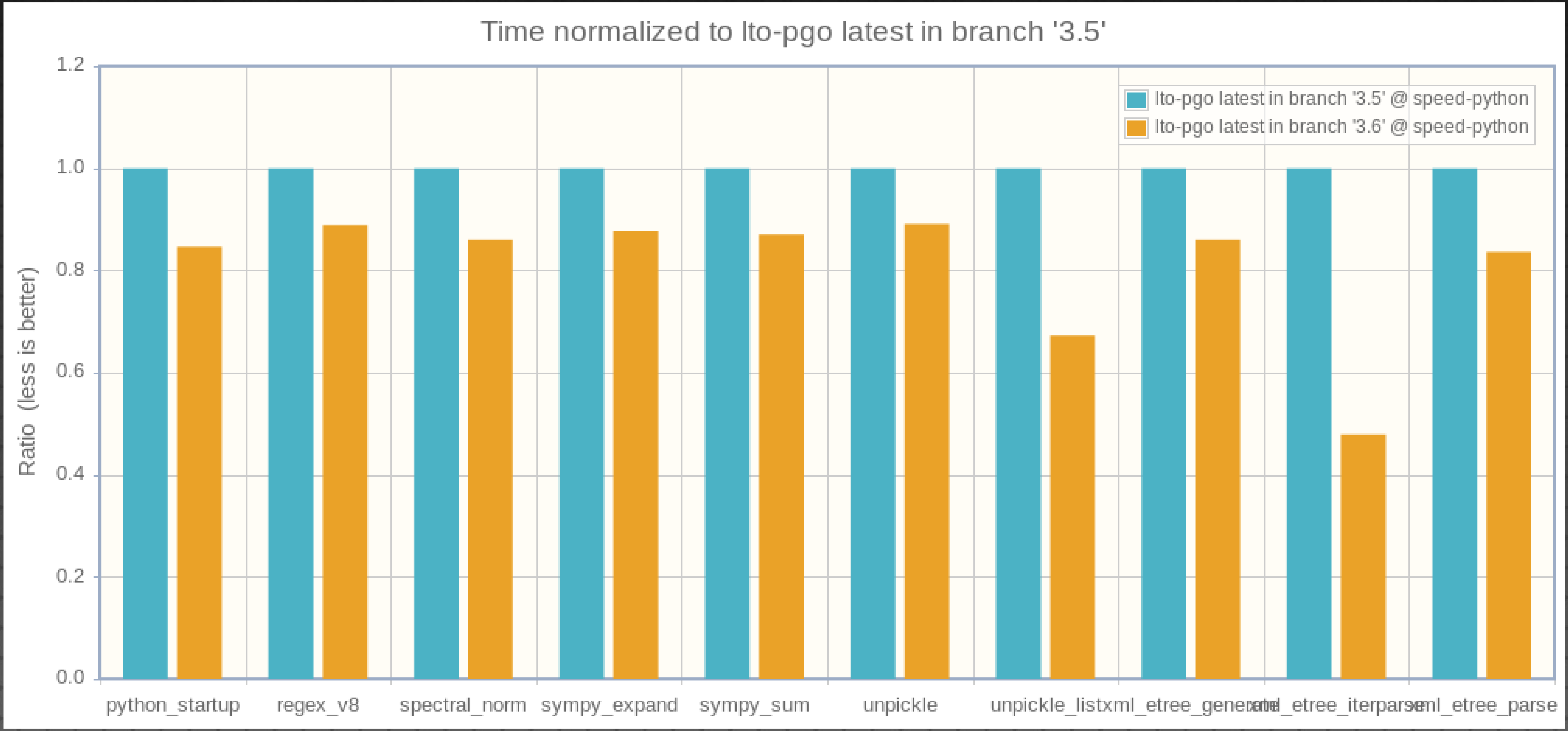


April, 2014 – May, 2017: 3 years

Agenda

(2) Benchmarks results

3.6 faster than 3.5

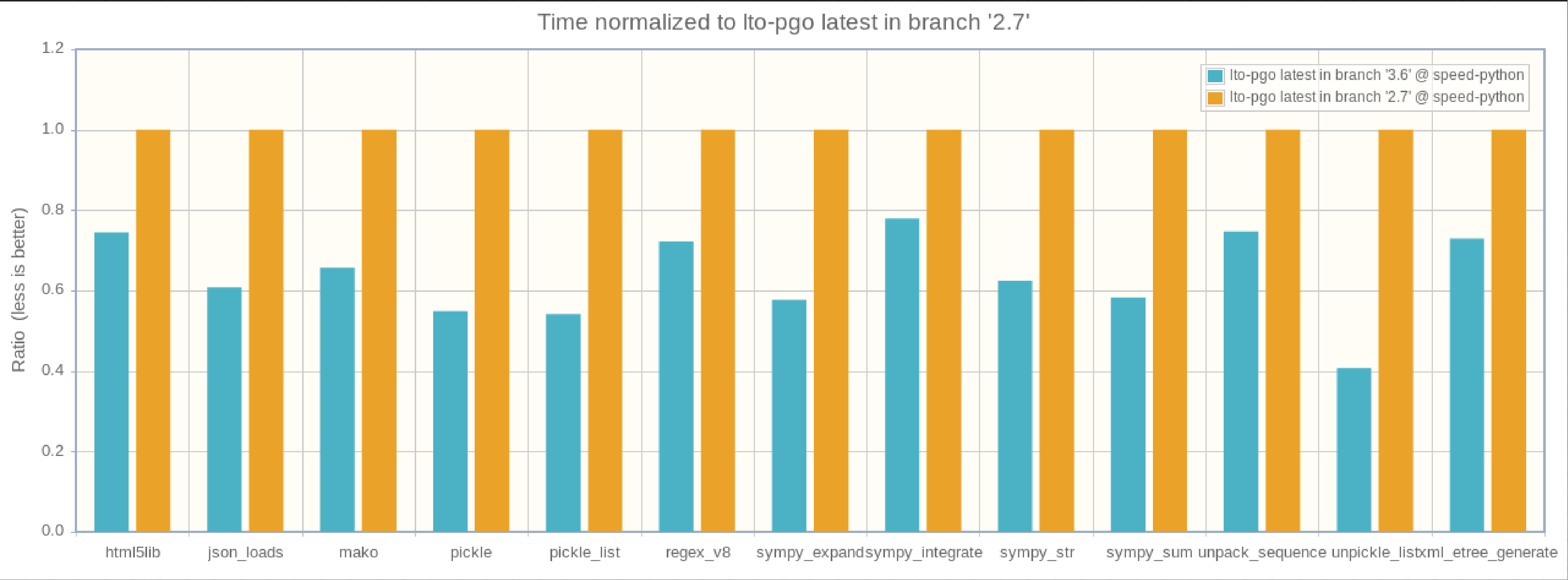


Results normalized to Python 3.5

lower = faster

redhat. python™

3.6 faster than 2.7



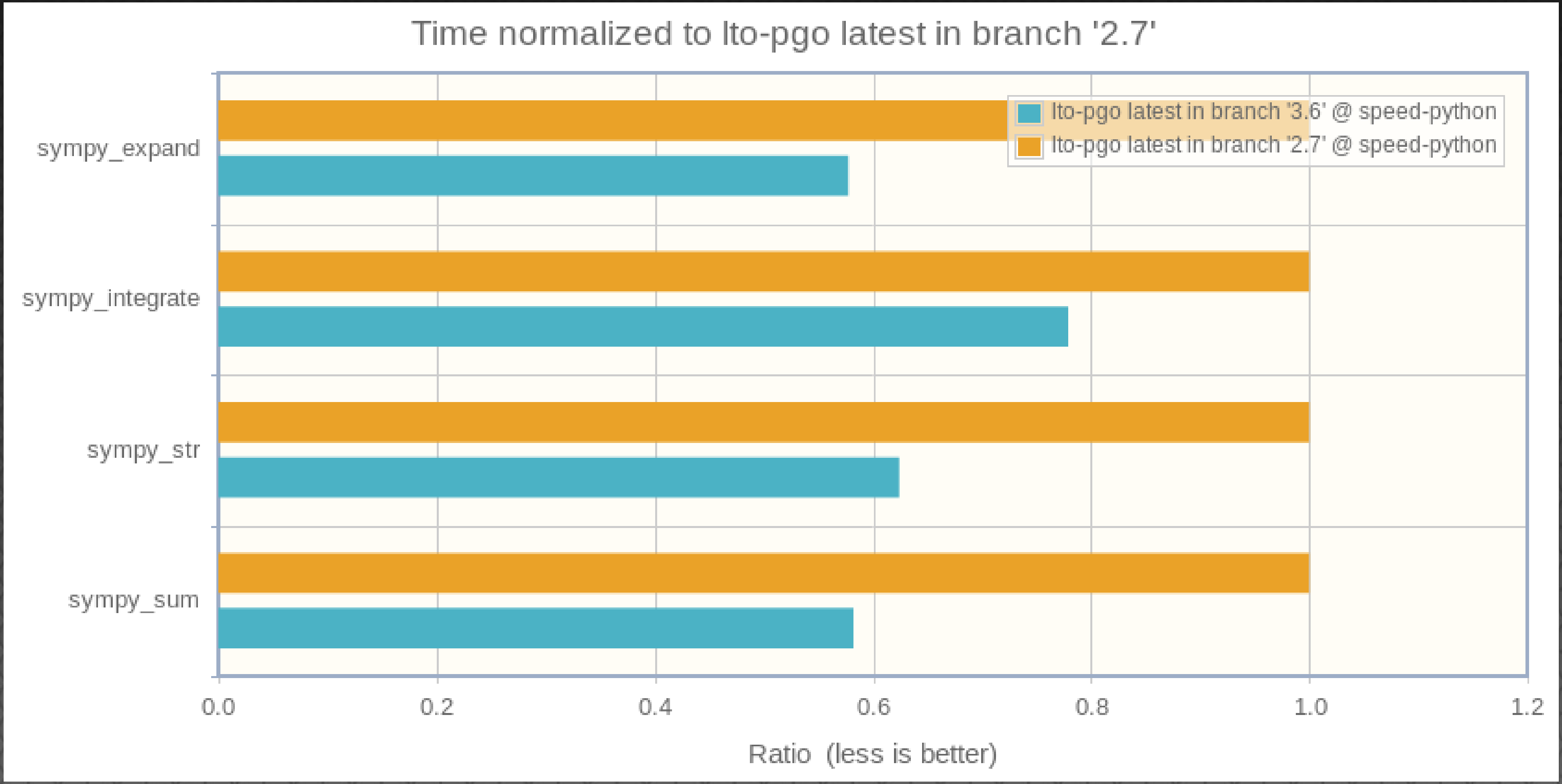
Results normalized to Python 2.7

lower = faster

redhat.

python™

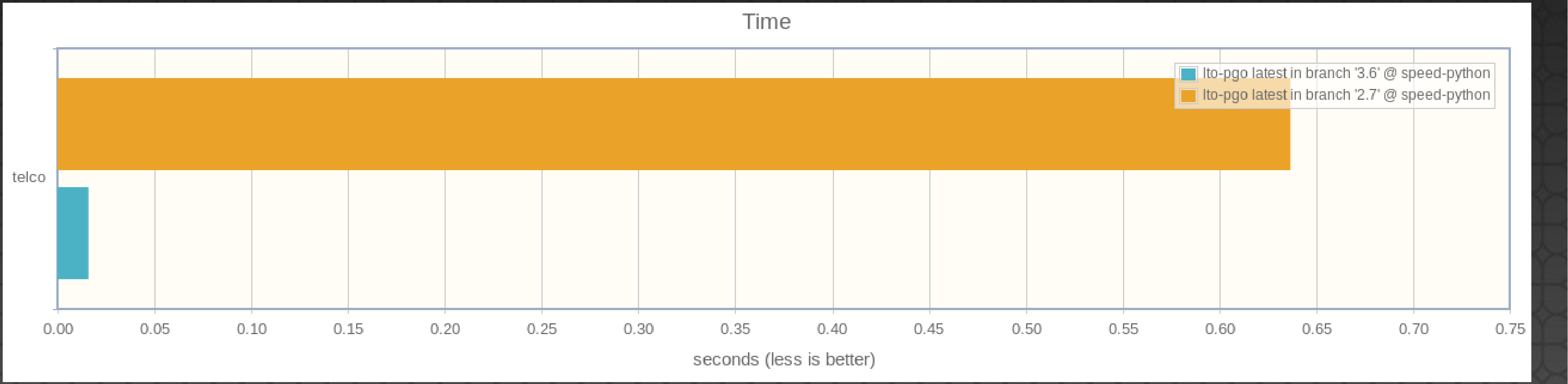
3.6 faster than 2.7



Sympy: 22% - 42% faster

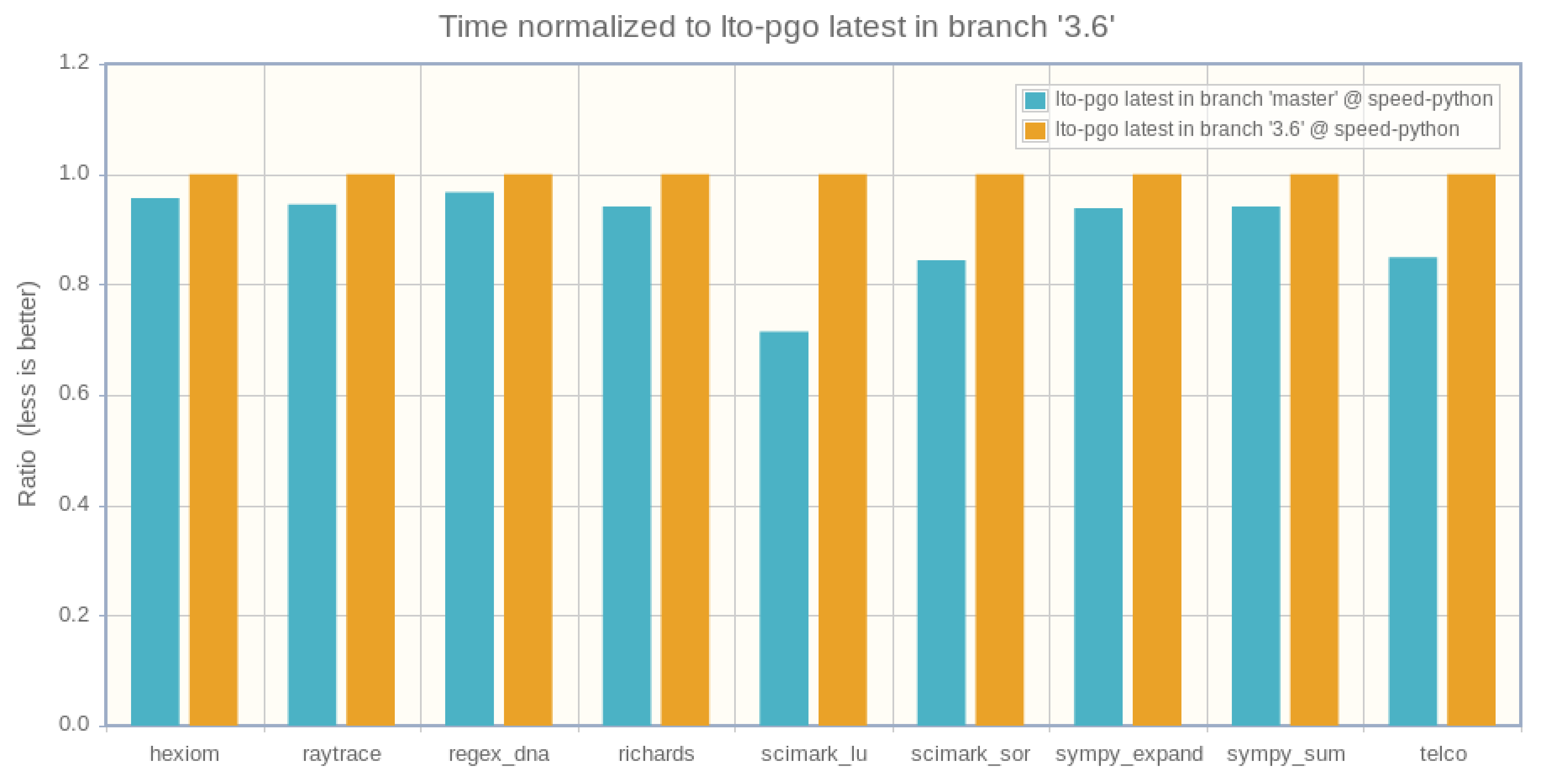
redhat. python™

telco: 3.6 vs 2.7



Python 3.6 is **40x** faster than Python 2.7
(`decimal` module rewritten in C
by Stefan Krah in Python 3.3)

3.7 faster than 3.6



Results normalized to Python 3.6

lower = faster

Agenda

(3) Python 3.5 optimizations

lru_cache()

- Matt Joiner, Alexey Kachayev and Serhiy Storchaka reimplemented `functools.lru_cache()` in C
- sympy: 20% faster
- scimark_lu: 5% faster
- Tricky C code, hard to get it right: 3 years ½ to close the bpo-14373

OrderedDict

- Eric Snow reimplemented collections.OrderedDict in C
- html5lib: **20% faster**
- Reuse C implementation of dict
- Again, tricky C code: 2 years ½ to close the bpo-16991

Agenda

(4) Python 3.6 optimizations

PyMem_Malloc()

- Victor Stinner changed `PyMem_Malloc()` to use Python fast memory allocator
- Many benchmarks: **5% - 22% faster**
- Check if the GIL is held in debug hooks
- Only numy misused the API (fixed)
- **PYTHONMALLOC=debug** now available in release builds to detect memory corruptions, bpo-26249

ElementTree parse

- Serhiy Storchaka optimized ElementTree.iterparse()
- 2x faster
- Follow-up of Brett Canon's Pycon Canada 2015 keynote :-)
- bpo-25638

PGO uses tests

- Brett Canon modified the Profile Guided Optimization (PGO)
- The Python test suite is now used, rather than pidigits, to guide the compiler
- Many benchmarks: 5% – 27% faster
- bpo-24915

Wordcode

- Demur Rumed and Serhiy Storchaka modified the bytecode to always use 2 bytes opcodes
- Before: 1 (no arg) or 3 bytes (with arg)
- Removed an if from ceval.c hotcode for better CPU branch prediction:

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

- bpo-26647

FASTCALL

- Victor Stinner wrote a new C API to avoid the creation of temporary tuples to pass function arguments
- Many microbenchmarks: **12% – 50% faster**
- `obj[0], getattr(obj, "attr"), {1: 2}.get(1), list.count(0), str.replace("a","b"), …`
- Avoid **20 ns** per modified function call

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

bytes % args

- PEP 461 added back bytes % args to Python 3.5
- Victor Stinner wrote a new `_PyBytesWriter` API to optimize functions creating bytes and bytearray strings
- bytes % args: 2x faster
- bytes.fromhex(): 3x faster

Globbing

- Serhiy Storchaka optimized glob.glob(), glob.iglob() and pathlib globbing using os.scandir() (new in Python 3.5)
- glob: 3x - 6x faster
- Pathlib glob: 1.5x - 4x faster
- Avoid one `stat()` per directory entry
- bpo-25596, bpo-26032

asyncio

- Yury Selivanov and Naoki INADA reimplemented asyncio Future and Task classes in C
- Asyncio programs: 30% faster
- bpo-26081, bpo-28544

Agenda

(5) Python 3.7 optimizations

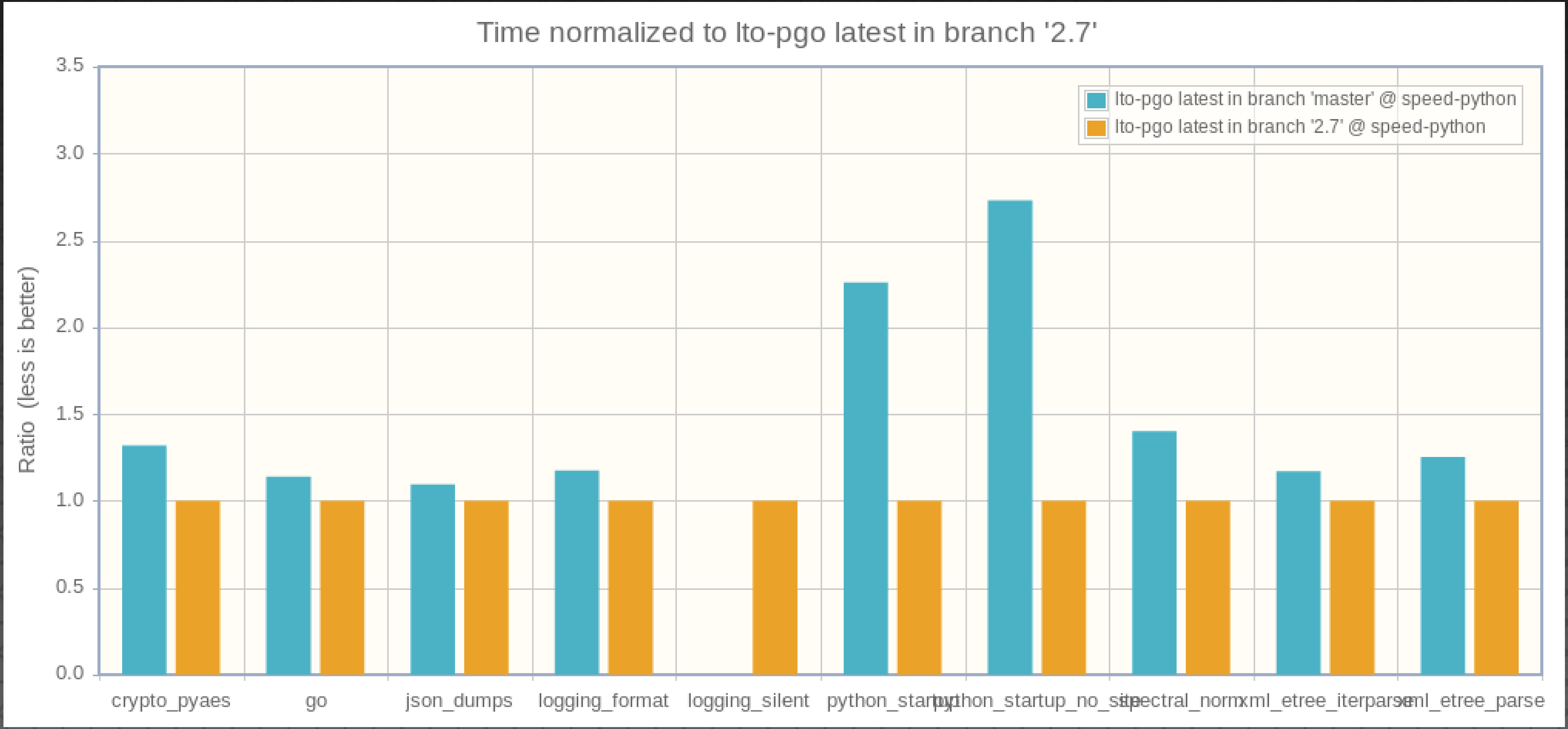
Method calls

- Yury Selivanov and Naoki INADA added LOAD_METHOD and CALL_METHOD opcodes
- Methods calls: 10% - 20% faster
- Idea coming from PyPy, bpo-26110

Future optimizations

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

3.7 slower than 2.7 :-(



Results normalized to Python 2.7

higher = slower

Questions?

http://speed.python.org/

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