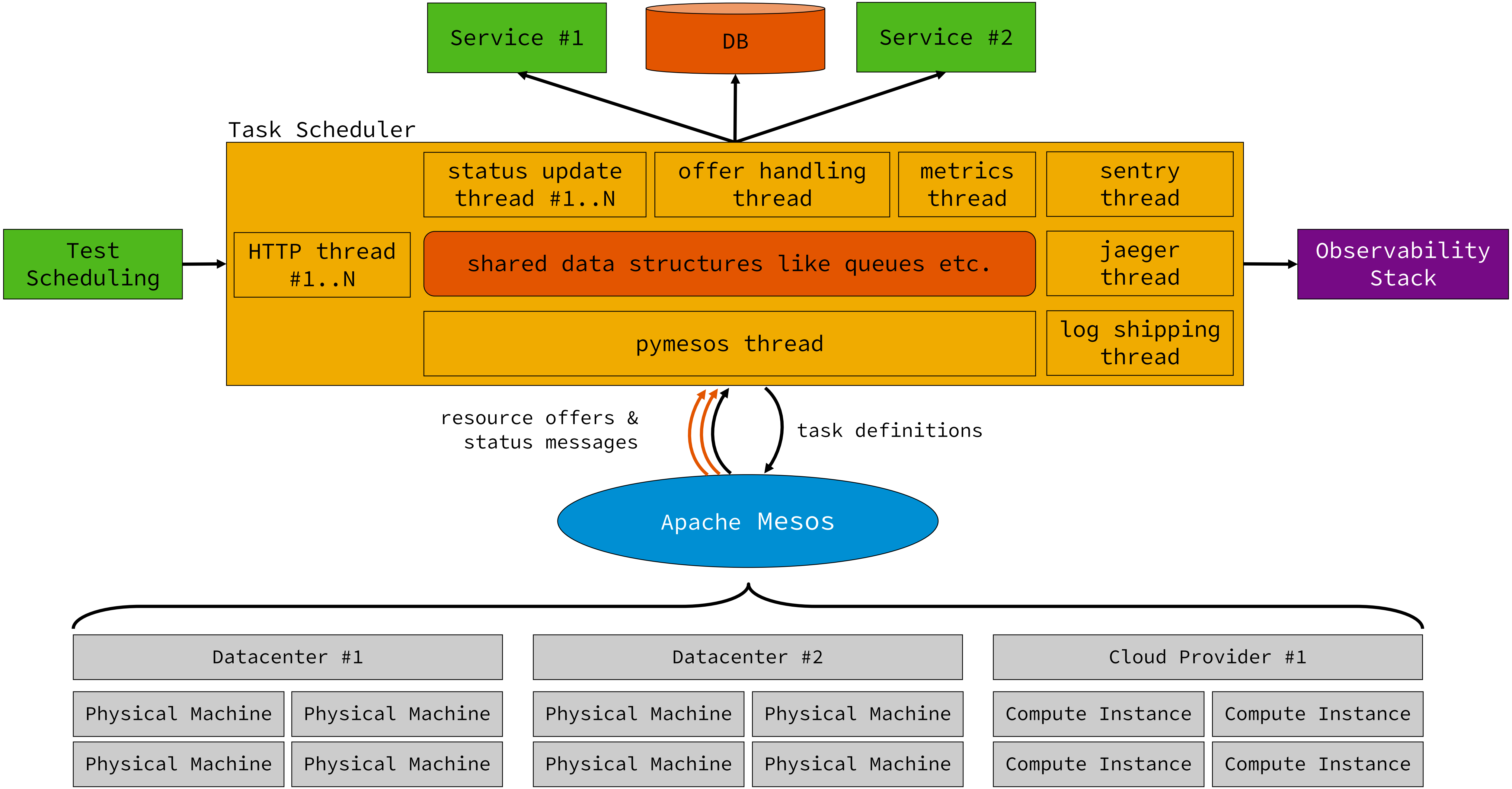
Is it me, or the GIL?

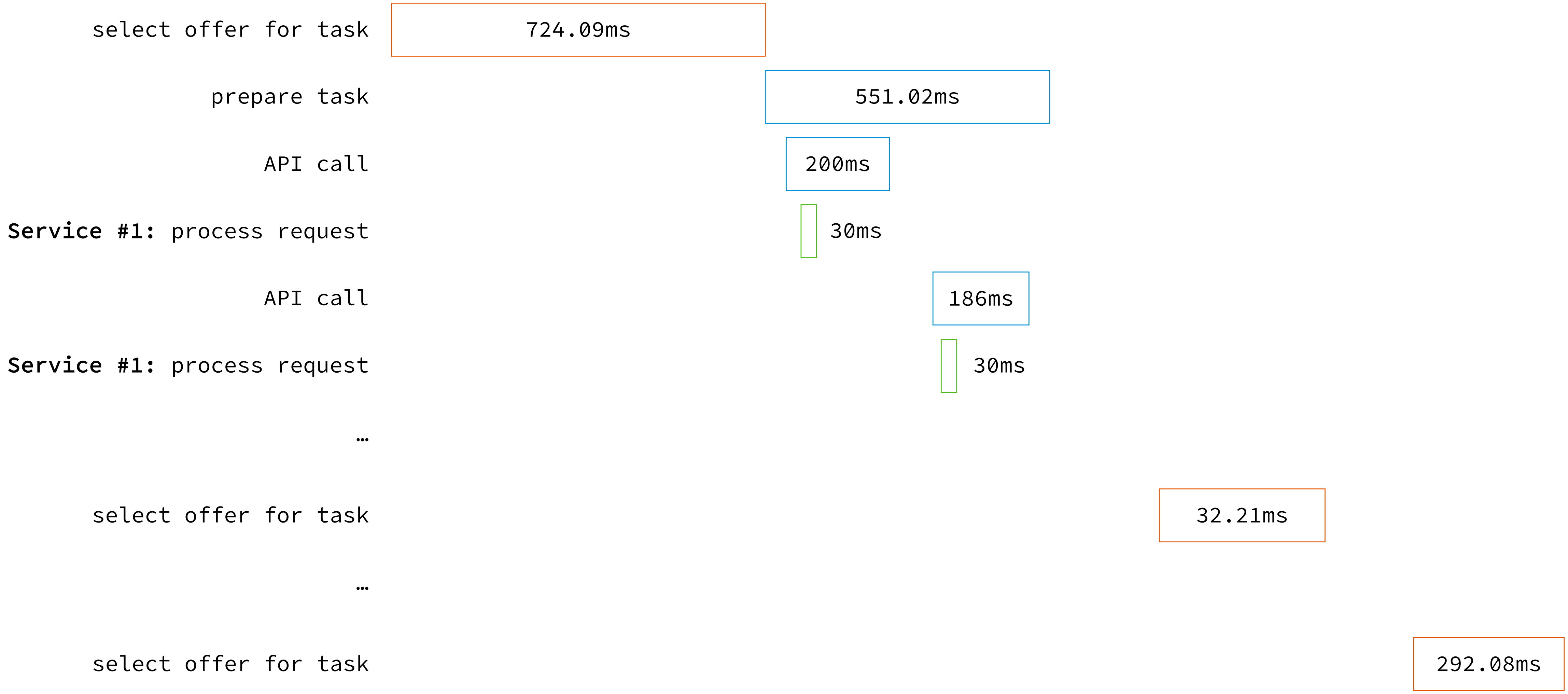
Christoph Heer
EuroPython 2019 – 10.07.2019

Background

- Quality assurance for SAP HANA
- Automated testing of ~800 commits per day
- Mainly testing with physical hardware: ~1600 machines, 610 TB RAM (256 GB – 8TB)
- Development of optimized tools and services


Service #1
DB
Service #2
Task Scheduler
status update thread #1..N
offer handling thread
metrics thread
sentry thread
Test Scheduling
HTTP thread #1..N
shared data structures like queues etc.
jaeger thread
Observability Stack
pymesos thread
log shipping thread
resource offers & status messages
task definitions
Apache Mesos
Datacenter #1
Datacenter #2
Cloud Provider #1
Physical Machine
Physical Machine
Physical Machine
Physical Machine
Physical Machine
Physical Machine
Physical Machine
Physical Machine
Compute Instance
Compute Instance
Compute Instance
Compute Instance

Inspect offer handling thread



Inspect offer handling thread

select offer for task 724.09ms

prepare task 551.02ms

API call 200ms

Service #1: process request 30ms

API call 186ms

Service #1: process request 30ms

…

select offer for task 32.21ms

…

select offer for task 292.08ms

Observations

- Different function runtimes

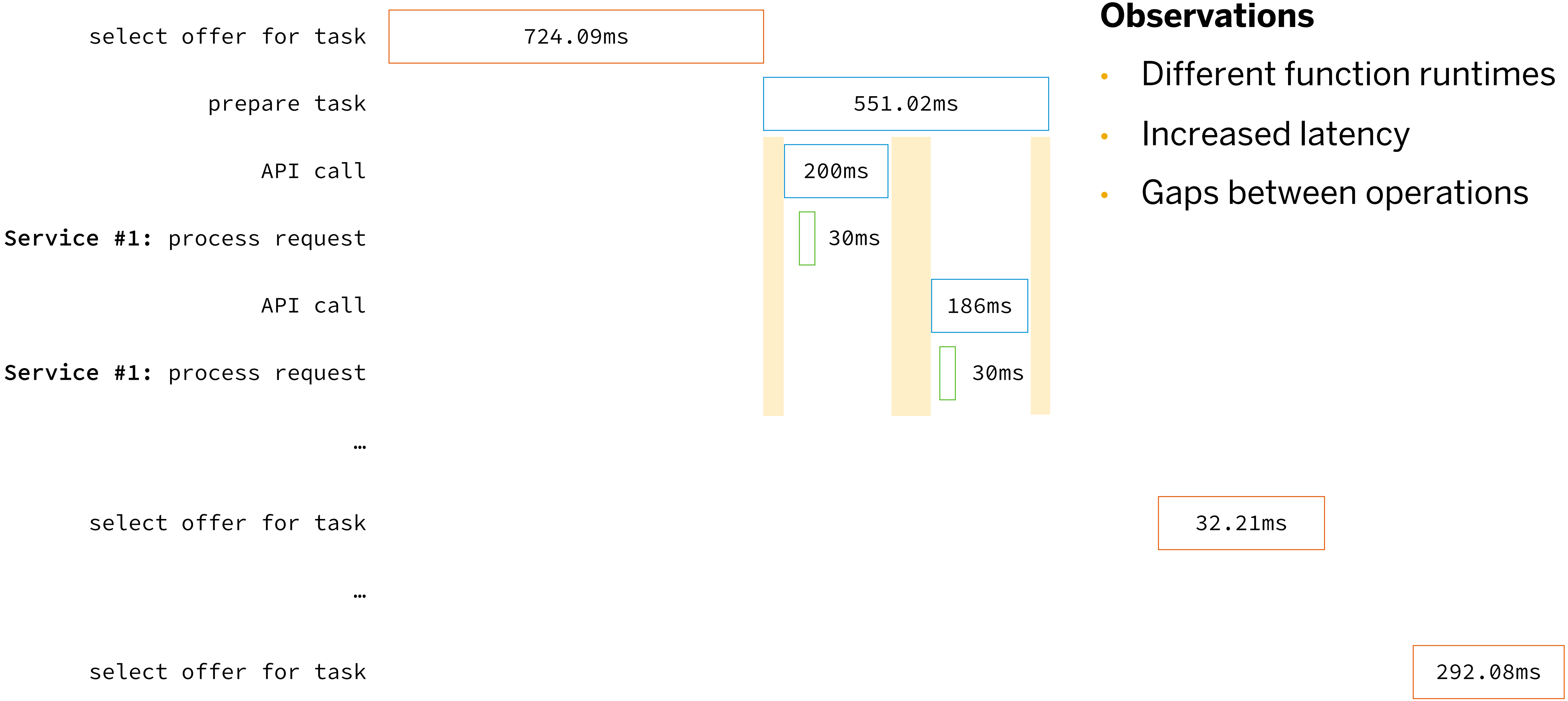
Inspect offer handling thread

select offer for task	724.09ms
prepare task	551.02ms
API call	200ms
Service #1: process request	30ms
API call	186ms
Service #1: process request	30ms
…	
select offer for task	32.21ms
…	
select offer for task	292.08ms

Observations

- Different function runtimes
- Increased latency

Inspect offer handling thread



Observations

- Different function runtimes
- Increased latency
- Gaps between operations

Inspect offer handling thread

select offer for task 724.09ms

prepare task 551.02ms

API call 200ms

Service #1: process request 30ms

API call 186ms

Service #1: process request 30ms

…

select offer for task 32.21ms

…

select offer for task 292.08ms

Observations

- Different function runtimes
- Increased latency
- Gaps between operations

Assumption: Thread released GIL multiple times and had to wait for re-acquire

Mitigate GIL contention

Python’s ecosystem offers various ways:

- multithreading => asyncio
- multithreading => multiprocessing (+ asyncio)
- CPU-intensive functions => Cython without the GIL
- Python => $faster language
- …

But, Rewriting and major refactoring are expensive

- Verify the assumption and measure GIL contention
- Decide about solution based on collected metrics instead of intuition

Look at the GIL

```
$ python3.7
Python 3.7.1 (default, Oct 22 2018, 13:16:18) [GCC] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.get_gil_stats()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute 'get_gil_stats’
```

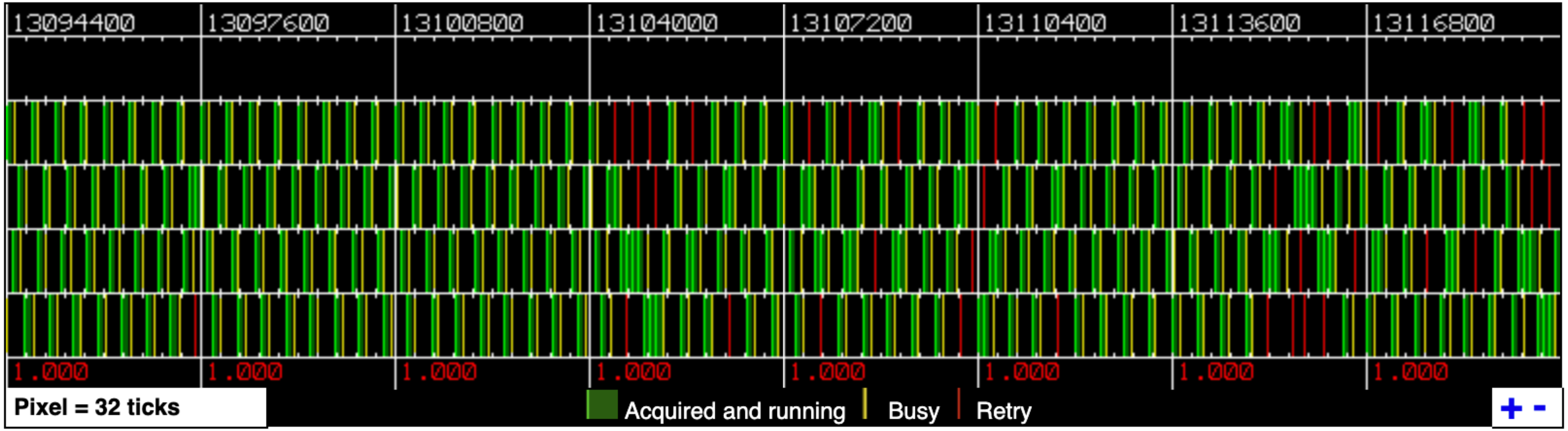
Wish list

- Provides metrics about GIL contention
  - wait time
  - hold time
- Additional context
  - Thread identifier/name
  - Python/C-Function
  - Trace/Request Id
- Usable for productive environments
  - Low overhead
  - Dynamically attachable to running Python processes
- Integration into existing observability stack

Related work and instrumentation approaches

A [Zoomable Interactive Python Thread Visualization]() by Dave Beazley

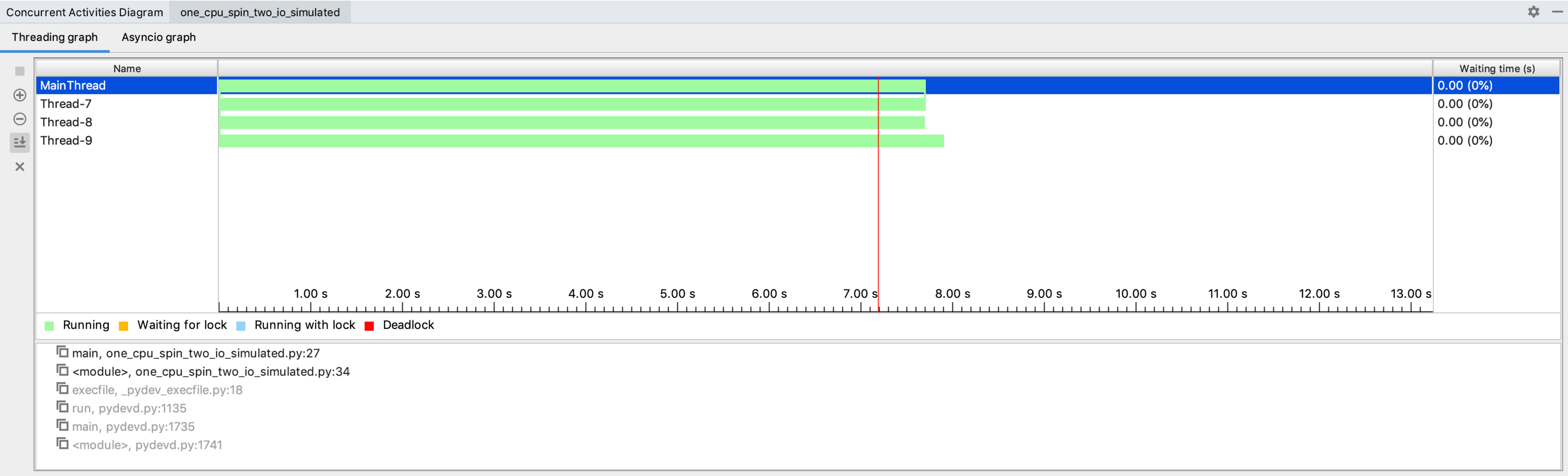
- Understanding the Python GIL (PyCon 2010)
- Adjustments in CPython 2.6 to store events about the GIL



Related work and instrumentation approaches

Thread Concurrency Visualization by PyCharm

- Visualizes and reveals locking issues but omits GIL



Related work and instrumentation approaches

gil_load by Chris Billington

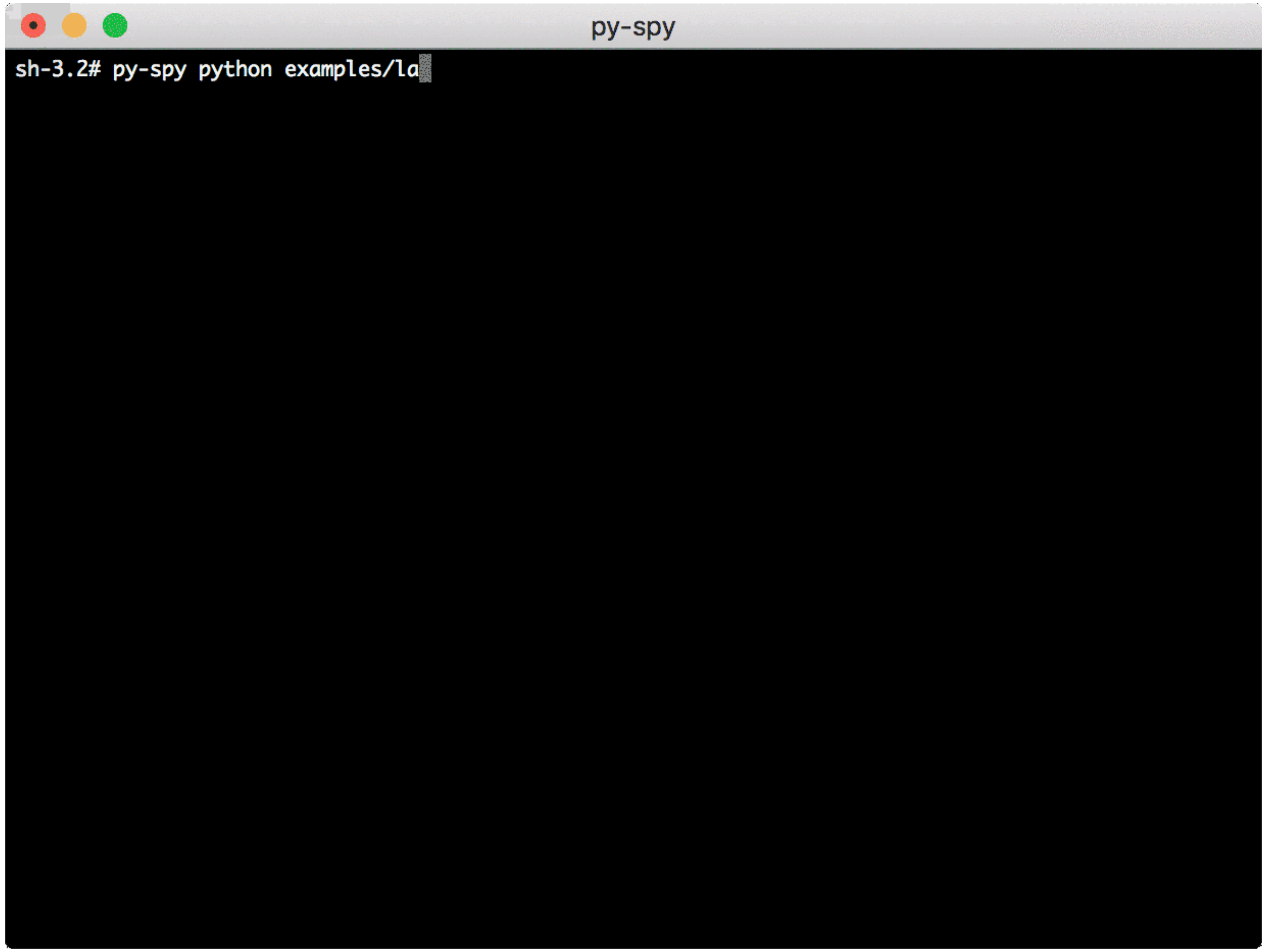
- Statistical profiler based on sampling states of all python threads every 50ms
- Installable python package (not Python 3.7 compatible yet)

```
$ python example_two.py
[2019-06-21 23:23:20]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:21]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:23]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:23]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:25]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:26]  GIL load: 1.00 (1.00, 1.00, 1.00)
[2019-06-21 23:23:27]  GIL load: 1.00 (1.00, 1.00, 1.00)
```

Related work and instrumentation approaches

py-spy by Ben Frederickson

- Promising sampling profiler in Rust for Python applications
- Includes GIL utilization metric but no breakdown into usage per thread



“There is no magic GIL performance analysis tool”

a sad Python developer

Creating a tool that reveals the GIL

- Base: SystemTap
- Analyze Linux applications by attaching handlers to events
- CPython 3.6 introduced DTrace/SystemTap support & markers
  - `function__entry`
  - `function__return`
  - Great documentation: [Instrumenting CPython with DTrace and SystemTap]
- Pre-build Linux packages often compiled without: `--with-dtrace`
- No GIL related markers

Adding SystemTap markers about the GIL

```
diff --git a/Python/ceval_gil.h b/Python/ceval_gil.h
index ef5189068e..aecd3c99aa 100644
@@ static void drop_gil(PyThreadState *tstate)
     MUTEX_LOCK(_PyRuntime.ceval.gil.mutex);
     _Py_atomic_store_relaxed(&_PyRuntime.ceval.gil.locked, 0);
+    if (PyDTrace_GIL_DROP_ENABLED())
+        PyDTrace_GIL_DROP(PyThread_get_thread_ident());
+
     COND_SIGNAL(_PyRuntime.ceval.gil.cond);
     MUTEX_UNLOCK(_PyRuntime.ceval.gil.mutex);

@@ static void take_gil(PyThreadState *tstate)
     if (tstate == NULL)
         Py_FatalError("take_gil: NULL tstate");

+    if (PyDTrace_GIL_CLAIM_ENABLED())
+        PyDTrace_GIL_CLAIM(PyThread_get_thread_ident());
+
     err = errno;
     MUTEX_LOCK(_PyRuntime.ceval.gil.mutex);
```

Only thread identifiers as argument as there is currently no C API for accessing user-defined thread name

Measure time between GIL markers

```
probe process("libpython3.7m.so.1.0").mark("gil_claim") {
    last_timestamp[$arg1] = gettimeofday_ns();
}

probe process("libpython3.7m.so.1.0").mark("gil_acquired") {
    wait_time_ns = gettimeofday_ns() - last_timestamp[$arg1];
    gil_wait_aggregate[$arg1] <<< wait_time_ns;
    last_timestamp[$arg1] = gettimeofday_ns();
}

probe process("libpython3.7m.so.1.0").mark("gil_drop") {
    hold_time_ns = gettimeofday_ns() - last_timestamp[$arg1];
    gil_hold_aggregate[$arg1] <<< hold_time_ns;
}
```

- Attach probes (event handlers) to GIL markers
- Calculate timing of transitions
  - claim - acquired - drop
- Store measurements in SystemTap statistical aggregate type per thread

Produce summary of collected data

```
probe begin {println("Start tracing of Python GIL probes")}

probe end {
    println("Terminate tracing")
    println("Summary (all time measurements in ns)")
    foreach (thread in gil_wait_aggregate) {
        printf("Python Thread %d\n", thread)
        printf(
            "Aggregated GIL wait time: %d\n",
            @sum(gil_wait_aggregate[thread])
        )
        printf(
            "Aggregated GIL hold time: %d\n",
            @sum(gil_hold_aggregate[thread])
        )
        printf("GIL wait latency\n")
        println(@hist_log(gil_wait_aggregate[thread]))
        printf("GIL hold time\n")
        println(@hist_log(gil_hold_aggregate[thread]))
    }
}
```

- Handlers for start and end of tracing session
- Report GIL wait and hold time as sum and histogram

Experiment 1: Process with 2 IO-bound threads

```python
def io_work(n=150):
    while n > 0:
        n -= 1
        time.sleep(0.1)

def main():
    io1 = Thread(name='io1', target=io_work)
    io2 = Thread(name='io2', target=io_work)

    io1.start()
    io2.start()
    io1.join()
    io2.join()
```

```
Summary (all time measurements in ns)
Python Thread 140604373862144         # MainThread
Aggregated GIL wait time: 1102539     #  1.10ms
Aggregated GIL hold time: 27794967    # 27.79ms

Python Thread 140604351805184         # io1
Aggregated GIL wait time: 452269      #  0.45ms
Aggregated GIL hold time: 1490119     #  1.49ms

Python Thread 140604343412480         # io2
Aggregated GIL wait time: 462690      #  0.46ms
Aggregated GIL hold time: 1382457     #  1.38ms
```

```
App. runtime:  15064.3ms
GIL Hold time:     29.7ms 0.20% of runtime
GIL Wait time:      1.9ms 0.01% of runtime
```

Experiment 2: CPU-bound thread

```python
def io_work(n=150):
    while n > 0:
        n -= 1
        time.sleep(0.1)

def cpu_spin():
    while True:
        pass

def main():
    io1 = Thread(name='io1', target=io_work)
    io2 = Thread(name='io2', target=io_work)
    cpu1 = Thread(
        name='cpu1', target=cpu_spin, daemon=True
    )

    ...
```

```
Summary (all time measurements in ns)
Python Thread 140347214374656         # MainThread
Aggregated GIL wait time: 45613707    #    45.61ms
Aggregated GIL hold time: 28039210    #    28.04ms

Python Thread 140347192317696         # io1
Aggregated GIL wait time: 760017132   #   760.02ms
Aggregated GIL hold time: 2109572     #     2.11ms

Python Thread 140347116091136         # io2
Aggregated GIL wait time: 770118858   #   770.01ms
Aggregated GIL hold time: 1504611     #     1.50ms

Python Thread 140347107698432         # cpu1
Aggregated GIL wait time: 3816216     #     3.81ms
Aggregated GIL hold time: 15790884948 # 15790.88ms
```

```
App. runtime:   15822ms
GIL hold time: 15821ms 99.99% of runtime
GIL wait time:  1578ms   9.97% of runtime
```

Experiment 2: GIL wait latency for IO threads

```
Python Thread 140347192317696 # io1
GIL wait latency
   value |-------------------------------------------------- count
     512 |                                                    0
    1024 |                                                    0
    2048 |                                                    1
    4096 |                                                    0
    8192 |                                                    0
         ~
 1048576 |                                                    0
 2097152 |                                                    0
 4194304 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 150
 8388608 |                                                    0
16777216 |                                                    0
```

```
Python Thread 140347116091136 # io2
GIL wait latency
   value |-------------------------------------------------- count
     512 |                                                    0
    1024 |                                                    0
    2048 |                                                    1
    4096 |                                                    0
    8192 |                                                    0
         ~
 1048576 |                                                    0
 2097152 |                                                    0
 4194304 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@   149
 8388608 |                                                    1
16777216 |                                                    0
33554432 |                                                    0
```

Stable wait latency between 4 – 8ms

```
GIL hold time
  value |--------------------------------
   1024 |
   2048 |
   4096 |@@@@@@@@@@@@@@
   8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  16384 |
  32768 |                                                   2
  65536 |                                                   2
 131072 |                                                   0
 262144 |                                                   1
```

```
--------------------------------- count
                                    0
                                    0
                                   34
@@@@@@@@@@@@@@@@@@@@@             115
  16384 |                           0
  32768 |                           1
  65536 |                           1
```

Experiment 2: Evaluation

- GIL contention affects overall application performance
  - Additional 5ms latency on each GIL acquire attempt after one blocking IO operation

```
>>> import sys
>>> sys.getswitchinterval()
0.005
```

- `switchinterval` defines the grace period before a waiting thread requests GIL drop
- GIL-holding thread may keep the GIL longer due to
  - long-running bytecode operation
  - external C function

Analyze production application with SystemTap

- **Plan**
  - Deploy container with customized CPython and SystemTap
  - Attach to process
  - Get clear and precise insides about GIL contention

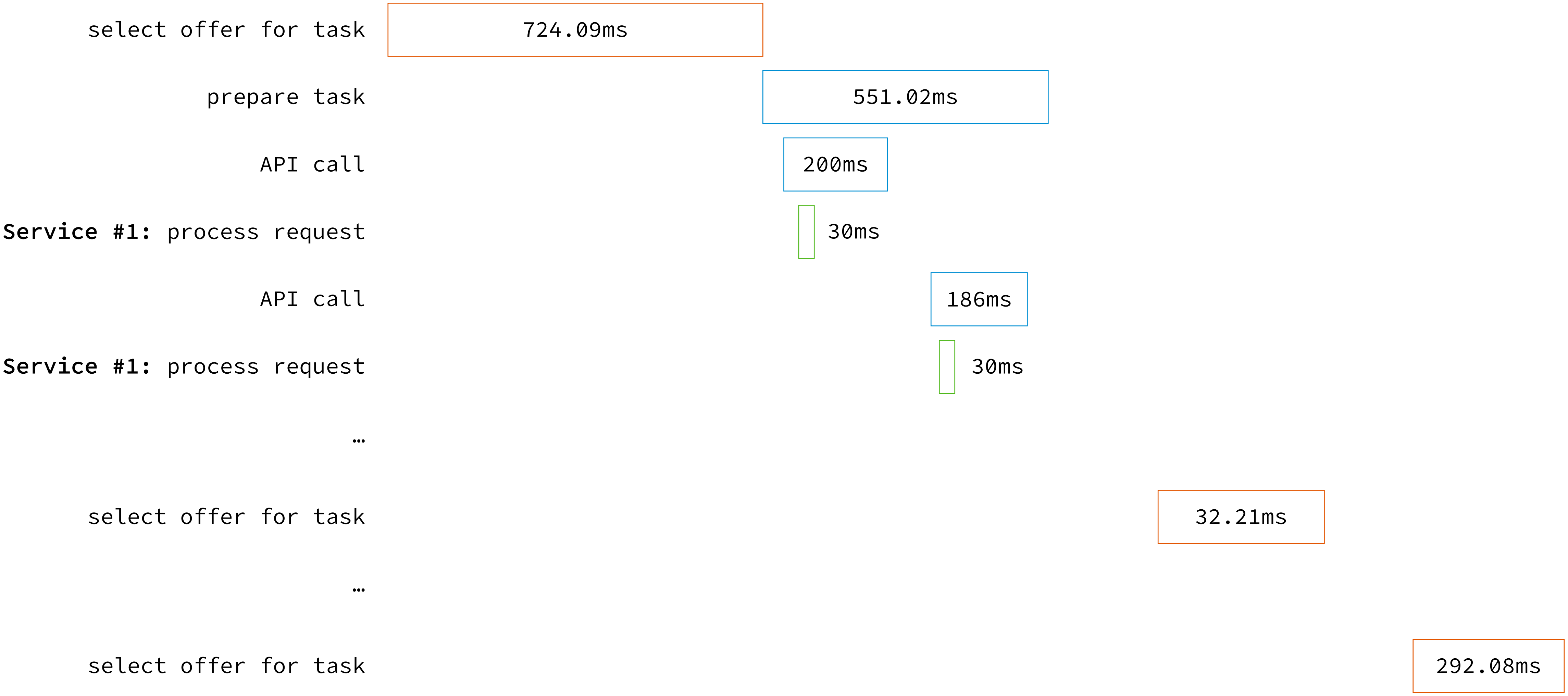
- **Relativity**
  - Deploy container with customized CPython
  - Install SystemTap on host with kernel sources, compiler toolchain etc.
  - Copy libpython3.7m.so.1.0 from container into host filesystem
  - Attach to process = Load custom kernel extension with your systemtap handlers
  - Get huge text file with report

Result: Analysis of productive task scheduler

```
Observation period:   120091.89ms
GIL hold time:        105680.91ms   88.00% of timeframe
GIL wait time:        352997.64ms  293.94% of timeframe
```

- **Proved, our application suffers from GIL contention**
- More questions:
  - Are there threads that hold the GIL longer than 5ms?
  - If yes, which functions are so expensive?
  - Is it possible to identify time-based patterns with higher contention?
- **Problem: With 31 threads the text report isn't well understandable**

Timelines are easier to understand



Replace text report with some colorful charts

Collect and store data with SystemTap

↓

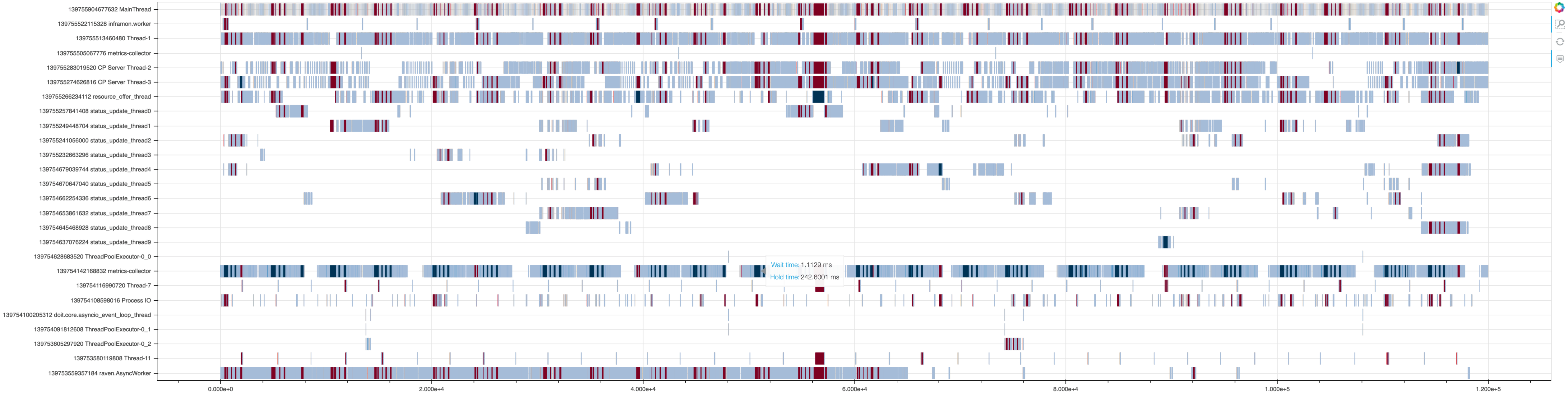
Load data into Jupyter notebook

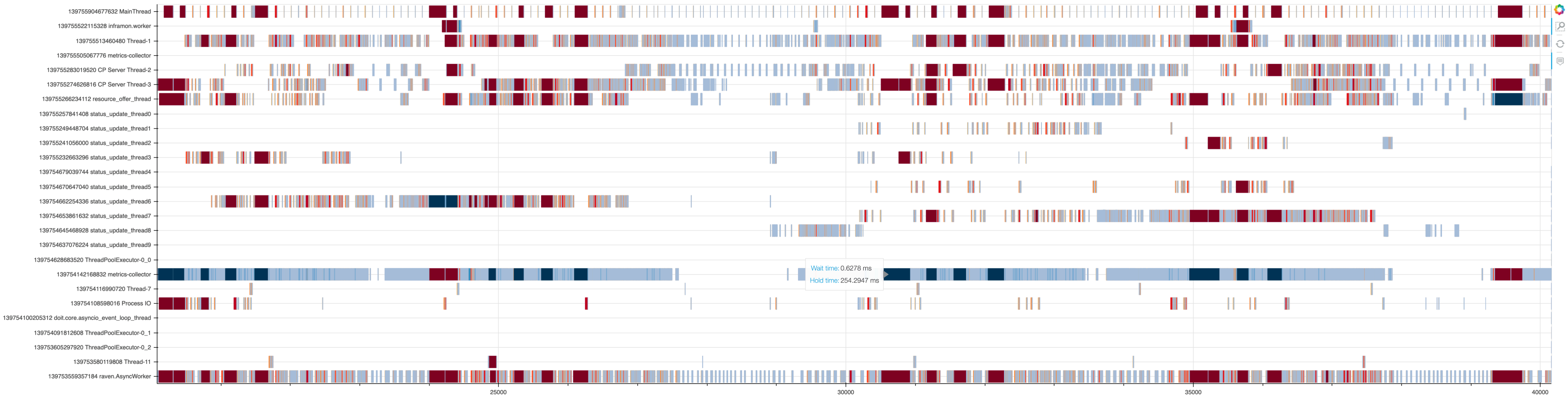
↓

Transform data

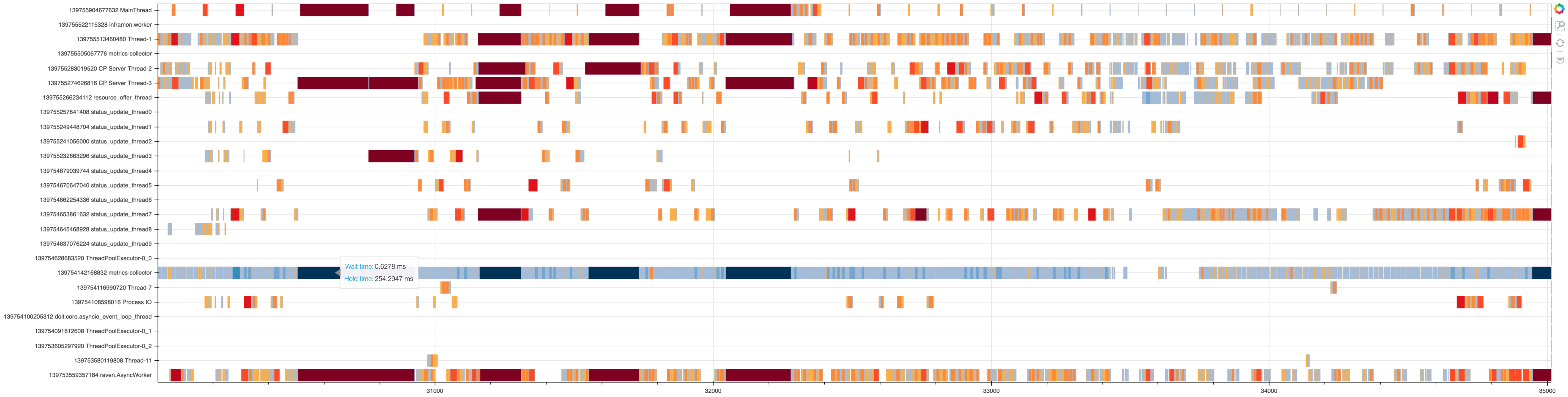
↓

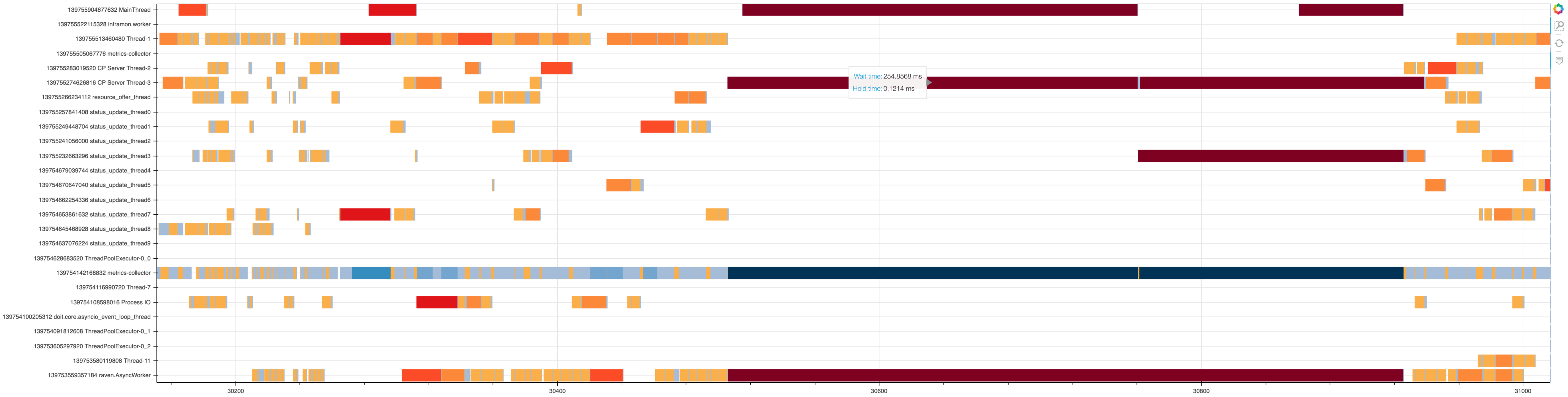
Visualize data with Bokeh


139755904677632 MainThread
139755522115328 inframon.worker
139755513460480 Thread-1
139755505067776 metrics-collector
139755283019520 CP Server Thread-2
139755274626816 CP Server Thread-3
139755266234112 resource_offer_thread
139755257841408 status_update_thread0
139755249448704 status_update_thread1
139755241056000 status_update_thread2
139755232663296 status_update_thread3
139754679039744 status_update_thread4
139754670647040 status_update_thread5
139754662254336 status_update_thread6
139754653861632 status_update_thread7
139754645468928 status_update_thread8
139754637076224 status_update_thread9
139754628683520 ThreadPoolExecutor-0_0
139754142168832 metrics-collector
139754116990720 Thread-7
139754108598016 Process IO
139754100205312 doit.core.asyncio_event_loop_thread
139754091812608 ThreadPoolExecutor-0_1
139753605297920 ThreadPoolExecutor-0_2
139753580119808 Thread-11
139753559357184 raven.AsyncWorker
Wait time: 1.1129 ms
Hold time: 242.6001 ms
0.000e+0
2.000e+4
4.000e+4
6.000e+4
8.000e+4
1.000e+5
1.200e+5



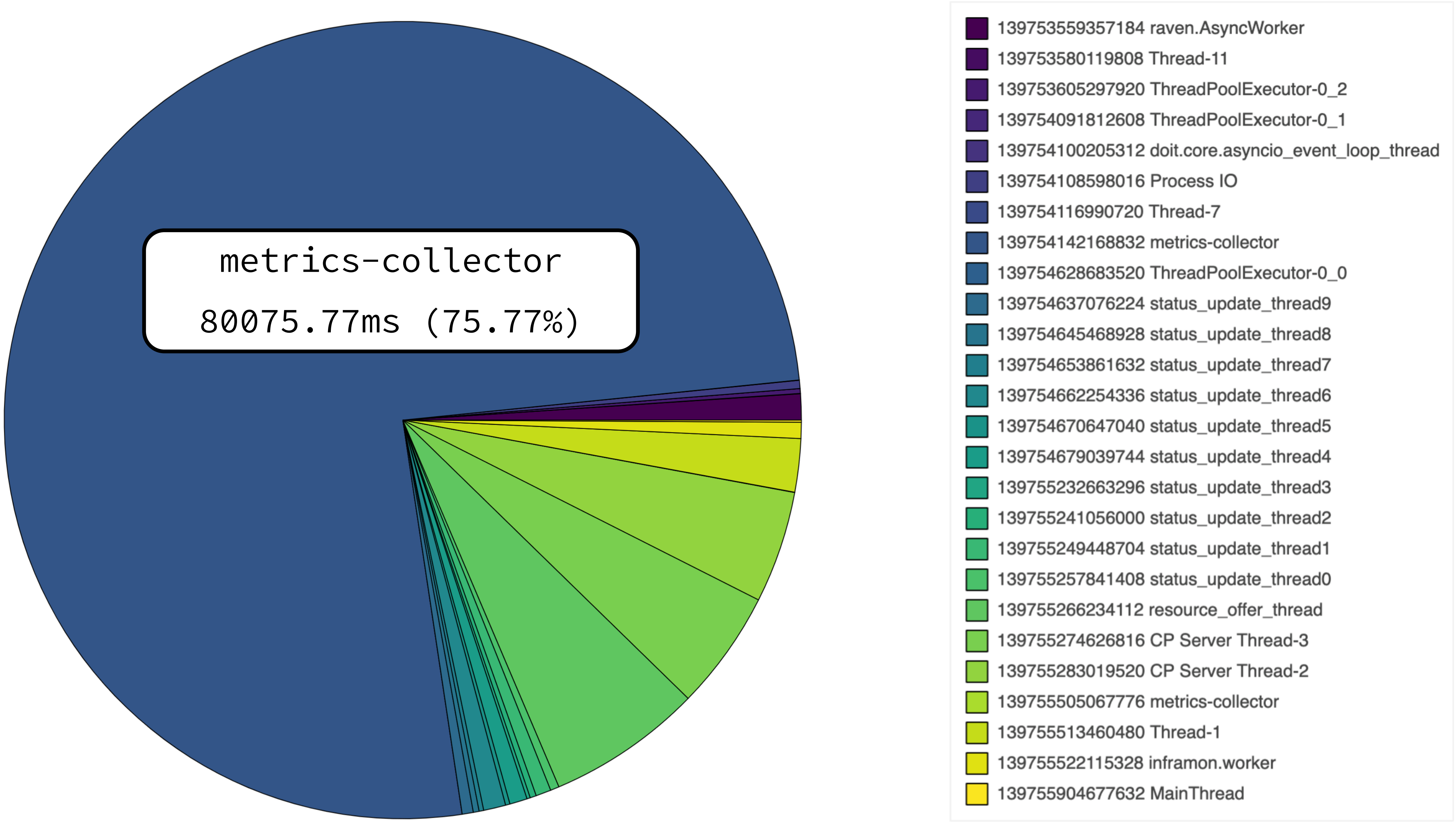
139755904677632 MainThread
139755522115328 inframon.worker
139755513460480 Thread-1
139755505067776 metrics-collector
139755283019520 CP Server Thread-2
139755274626816 CP Server Thread-3
139755266234112 resource_offer_thread
139755257841408 status_update_thread0
139755249448704 status_update_thread1
139755241056000 status_update_thread2
139755232663296 status_update_thread3
139754679039744 status_update_thread4
139754670647040 status_update_thread5
139754662254336 status_update_thread6
139754653861632 status_update_thread7
139754645468928 status_update_thread8
139754637076224 status_update_thread9
139754628683520 ThreadPoolExecutor-0_0
139754142168832 metrics-collector
139754116990720 Thread-7
139754108598016 Process IO
139754100205312 doit.core.asyncio_event_loop_thread
139754091812608 ThreadPoolExecutor-0_1
139753605297920 ThreadPoolExecutor-0_2
139753580119808 Thread-11
139753559357184 raven.AsyncWorker
Wait time: 0.6278 ms
Hold time: 254.2947 ms
25000
30000
35000
40000


139755904677632 MainThread
139755522115328 inframon.worker
139755513460480 Thread-1
139755505067776 metrics-collector
139755283019520 CP Server Thread-2
139755274626816 CP Server Thread-3
139755266234112 resource_offer_thread
139755257841408 status_update_thread0
139755249448704 status_update_thread1
139755241056000 status_update_thread2
139755232663296 status_update_thread3
139754679039744 status_update_thread4
139754670647040 status_update_thread5
139754662254336 status_update_thread6
139754653861632 status_update_thread7
139754645468928 status_update_thread8
139754637076224 status_update_thread9
139754628683520 ThreadPoolExecutor-0_0
139754142168832 metrics-collector
139754116990720 Thread-7
139754108598016 Process IO
139754100205312 doit.core.asyncio_event_loop_thread
139754091812608 ThreadPoolExecutor-0_1
139753605297920 ThreadPoolExecutor-0_2
139753580119808 Thread-11
139753559357184 raven.AsyncWorker
Wait time: 0.6278 ms
Hold time: 254.2947 ms
31000
32000
33000
34000
35000



139755904677632 MainThread
139755522115328 inframon.worker
139755513460480 Thread-1
139755505067776 metrics-collector
139755283019520 CP Server Thread-2
139755274626816 CP Server Thread-3
139755266234112 resource_offer_thread
139755257841408 status_update_thread0
139755249448704 status_update_thread1
139755241056000 status_update_thread2
139755232663296 status_update_thread3
139754679039744 status_update_thread4
139754670647040 status_update_thread5
139754662254336 status_update_thread6
139754653861632 status_update_thread7
139754645468928 status_update_thread8
139754637076224 status_update_thread9
139754628683520 ThreadPoolExecutor-0_0
139754142168832 metrics-collector
139754116990720 Thread-7
139754108598016 Process IO
139754100205312 doit.core.asyncio_event_loop_thread
139754091812608 ThreadPoolExecutor-0_1
139753605297920 ThreadPoolExecutor-0_2
139753580119808 Thread-11
139753559357184 raven.AsyncWorker
Wait time: 254.8568 ms
Hold time: 0.1214 ms
30200
30400
30600
30800
31000

Distribution of GIL hold time



Fixing our GIL contention

- Started replacing of an expensive C-Extension that doesn’t release the GIL
- Increased sleep time of metrics-collector thread from 10sec to 120sec

```
Before:
Observation period:   120091.89ms
GIL hold time:        105680.91ms   88.00% of timeframe
GIL wait time:        352997.64ms  293.94% of timeframe

After:
Observation period: 300112.84ms
GIL hold time:      130806.91ms     43.59% of timeframe
GIL wait time:      240568.60ms     80.16% of timeframe
```

- **Without major refactoring the system is now able to utilize all available resources**
- Collected data will helps us to decide about coming architectural changes

Many additional ideas

- Bring toolset (systemtap script & visualization) in a public usable state
- Enhance CPython for data collection with SystemTap
  - Integrate GIL markers into CPython
  - C API for thread names
- Maybe collect and provide GIL metrics directly with CPython
  - sys.get_gil_stats()
  - Easier integration into existing observability tooling like distributed tracing
  - No need to compile custom kernel extensions in your productive environment
- **If you are also interested in that area, let's talk!**

Thank you.

Christoph Heer

christoph.heer@sap.com
@ChristophHeer

THE BEST RUN

© 2019 SAP SE or an SAP affiliate company.