A Thundering Herd in the Wild

Bloomberg

Engineering

SREcon25 Americas
March 24, 2025

Nicolas Arroyo
Software Architect, Order & Trade Capture
narroyo1@bloomberg.net
www.linkedin.com/in/nicolas-arroyo-duran

TechAtBloomberg.com

What is a ‘thundering herd problem’?

The ‘thundering herd problem’ is a performance issue that happens when multiple threads wait on the same event and are all woken up at the same time. If only one thread can handle the event, the others waste resources with empty and unnecessary context switches.

Bloomberg

Engineering

The way it used to be

- Multiple threads running accept() on the same socket
  - Task exclusive wakeups
- Multiple threads running epoll_wait()
  - Edge-triggered events
- Condition variable broadcast
  - FUTEX_REQUEUE/FUTEX_CMP_REQUEUE functionality
- Networking/system code is usually buried in low-level libraries

Bloomberg

Engineering

The story begins …

The story

FRIEND

Hey Nick, can you talk?

NICK

Sure! What's up?

FRIEND

I have a strange performance problem. Service *unicorn* fetches data from datastore *rainbow* as you know …

FRIEND

And the mean latency is around 80 microseconds.

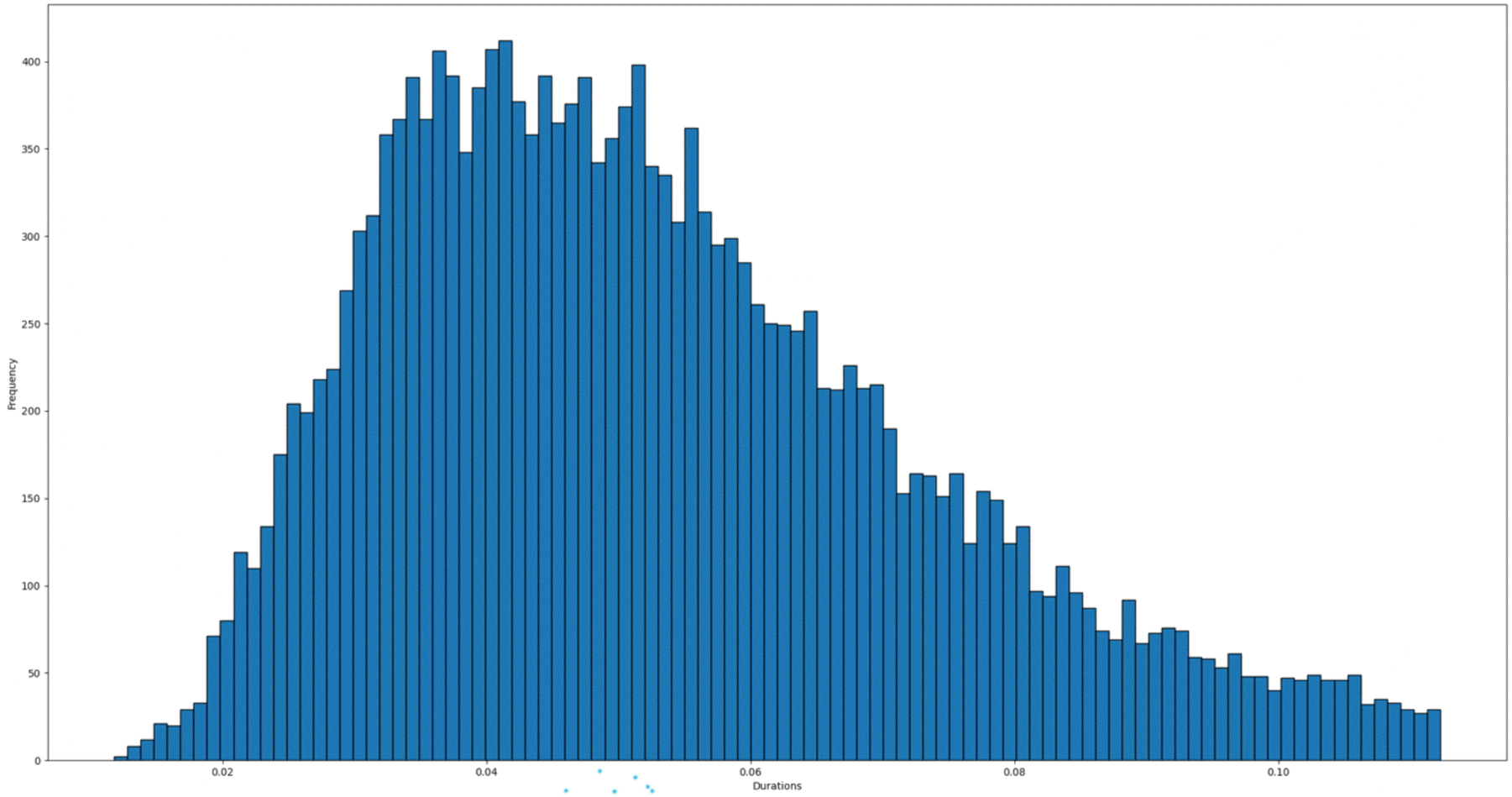
NICK

That sounds reasonable.

FRIEND

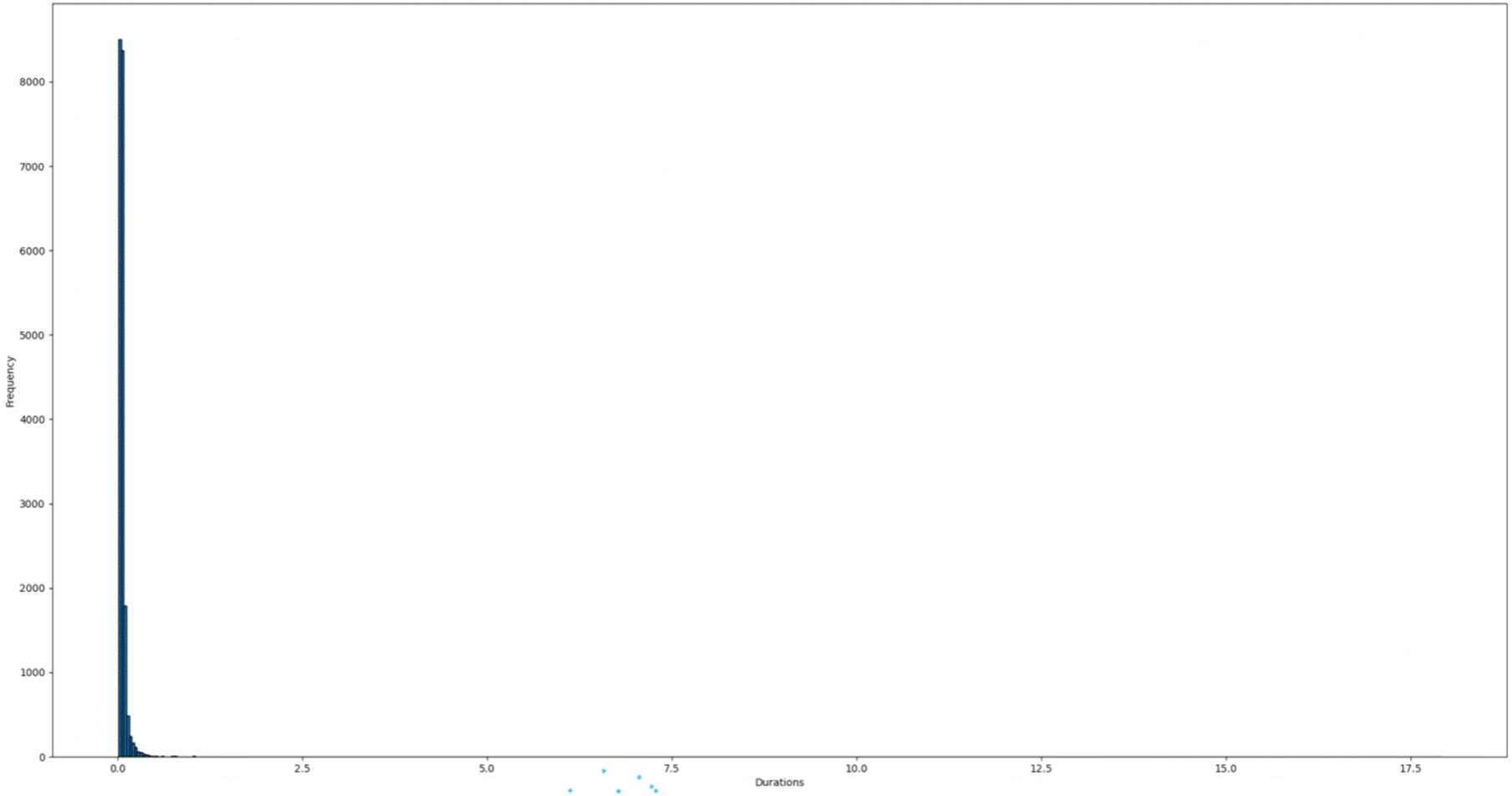
Yes, but the problem is that I'm seeing some outliers that take as much as 20 milliseconds! And they happen daily.

Extreme latency outliers in datastore fetches



P95 Histogram

Extreme latency outliers in datastore fetches



Complete Histogram

The story continues …

Sounds like *rainbow* is saturated. Have you looked at the service metrics?

NICK

FRIEND

I did, it's not that. The problem happens regularly, regardless of system activity.

Are the records huge when these outliers happen? Or are there maybe a lot of them?

NICK

FRIEND

Nope, I checked and the record sizes for this outliers are average. Also, there's only 1 fetch per request.

Then maybe rainbow datastore is running background jobs when this happens?

NICK

FRIEND

It only has a background job related to redundancy that runs once a day.

What is sampling profiling?

- Take a thread's state snapshot with a given frequency
- Can be time-based using OS timers or event-based relying on software/hardware events
- Samples are typically aggregated into a statistical model
- Points to hot code that can benefit from optimization

Bloomberg

Engineering

Why sampling?

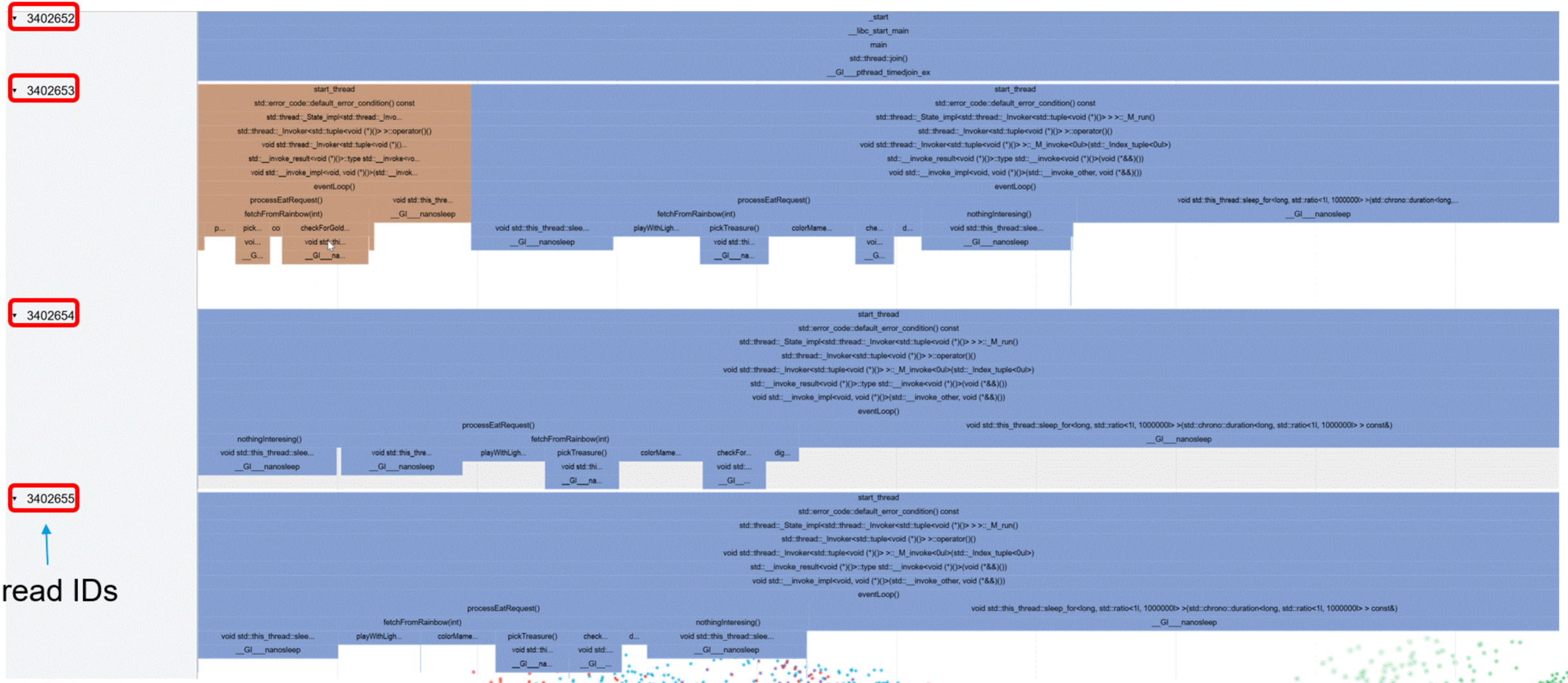
- Aggregating samples will destroy information
- Sampling will point out which callstacks are involved
- May still help if the problem is related to scaling
- This profiler is both on- and off-CPU
- Problem is probably I/O related
- Doing off-CPU by probing the scheduler is not convenient
- It is user-mode, and uses OS timers

Bloomberg

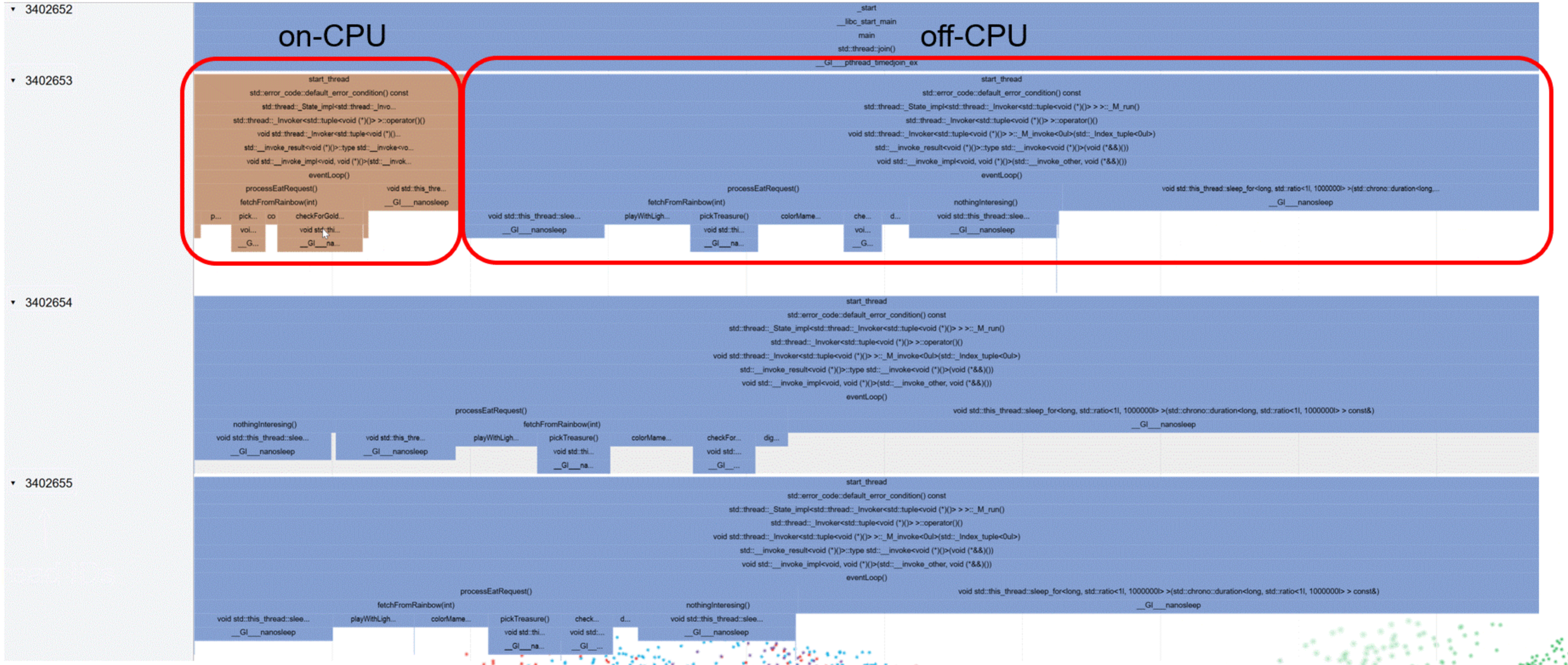
Engineering

Let’s try sampling …

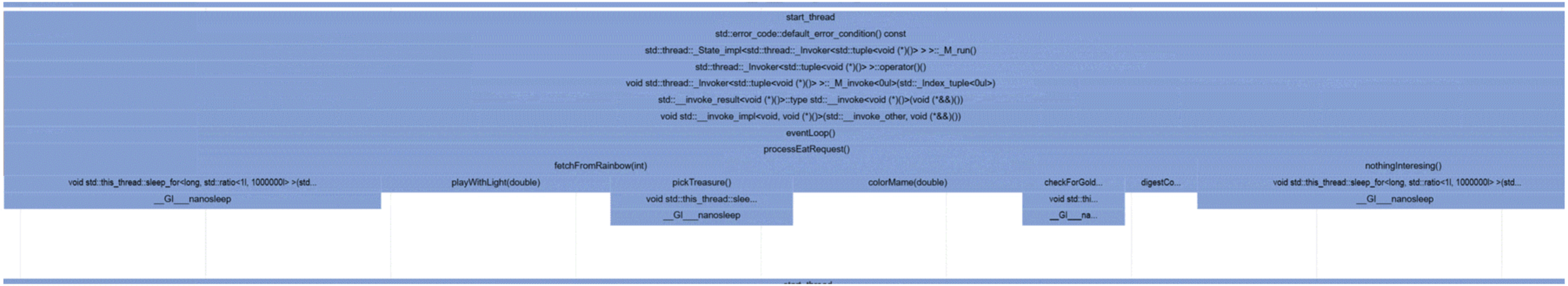
Let’s try sampling …



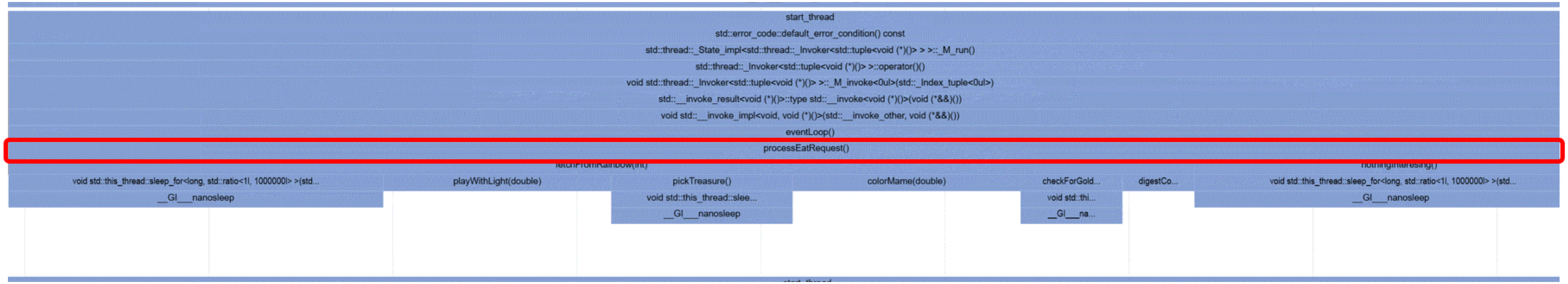
Let’s try sampling …



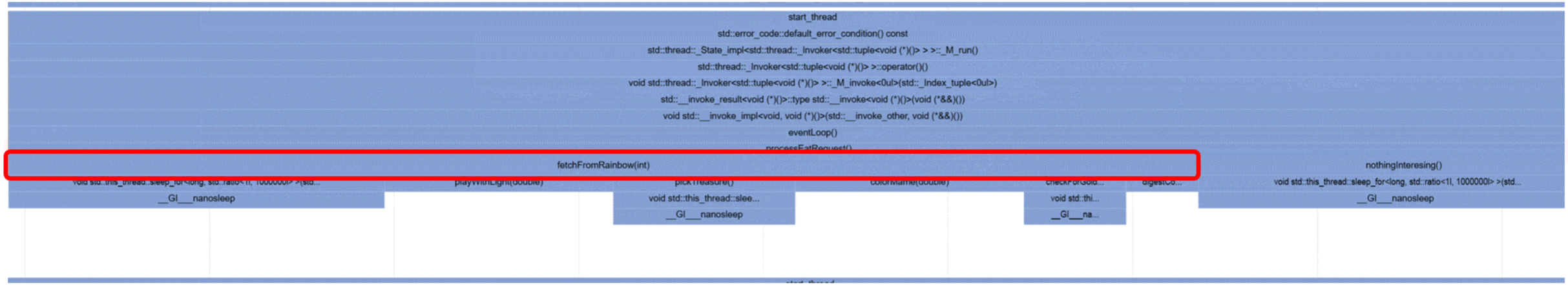
Let’s try sampling …



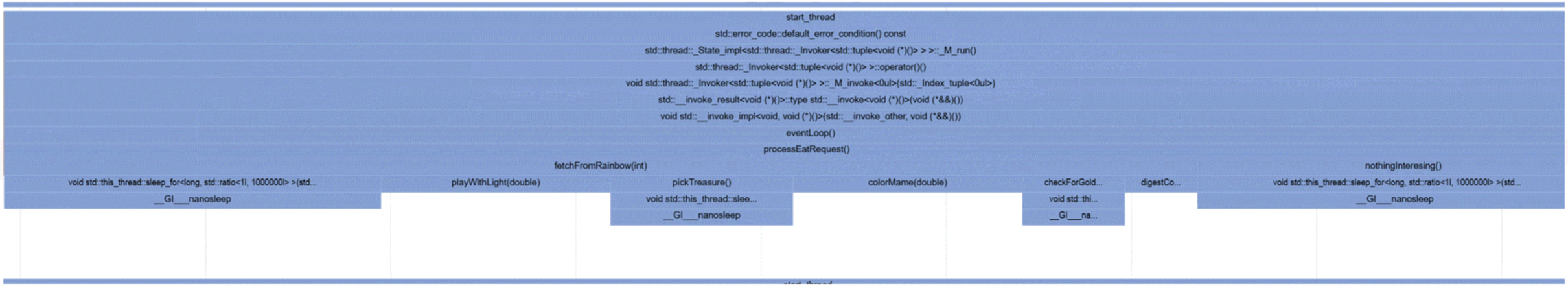
Let’s try sampling …



Let’s try sampling …

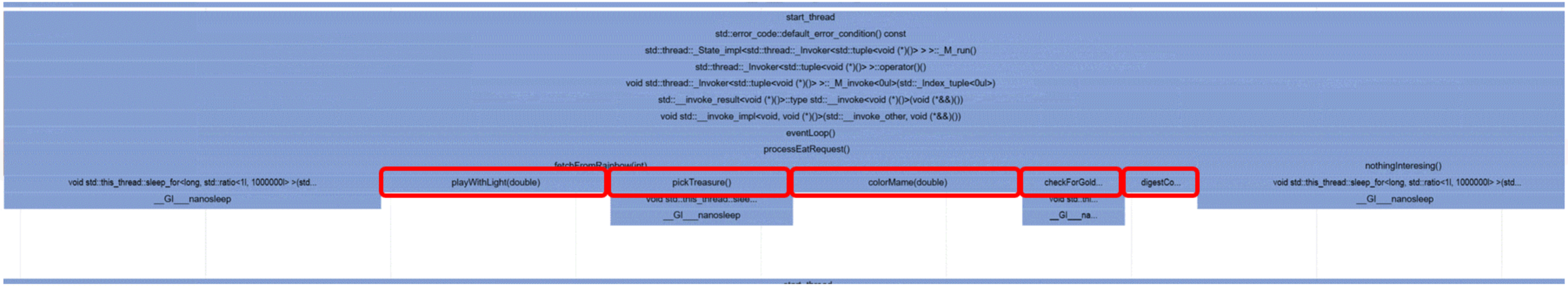


Let’s try sampling …



Suspects:
- fetchFromRainbow()

Let’s try sampling …



Suspects:
- fetchFromRainbow()
- playWithLight()
- pickTreasure()
- colorMane()
- checkForGoldPot()
- digestColors()

How about code instrumentation?

```
void runProgram() {
    initializeSystem();

    std::string input = getUserInput();
    processInput(input);

    std::string output = "Processed: " + input;
    displayOutput(output);

    cleanupSystem();
}
```

How about code instrumentation?

```
auto start = std::chrono::system_clock::now();
void runProgram() {
    initializeSystem();

    std::string input = getUserInput();
    processInput(input);

    std::string output = "Processed: " + input;
    displayOutput(output);

    cleanupSystem();
}
auto duration = std::chrono::system_clock::now() - start;
```

How about code instrumentation?

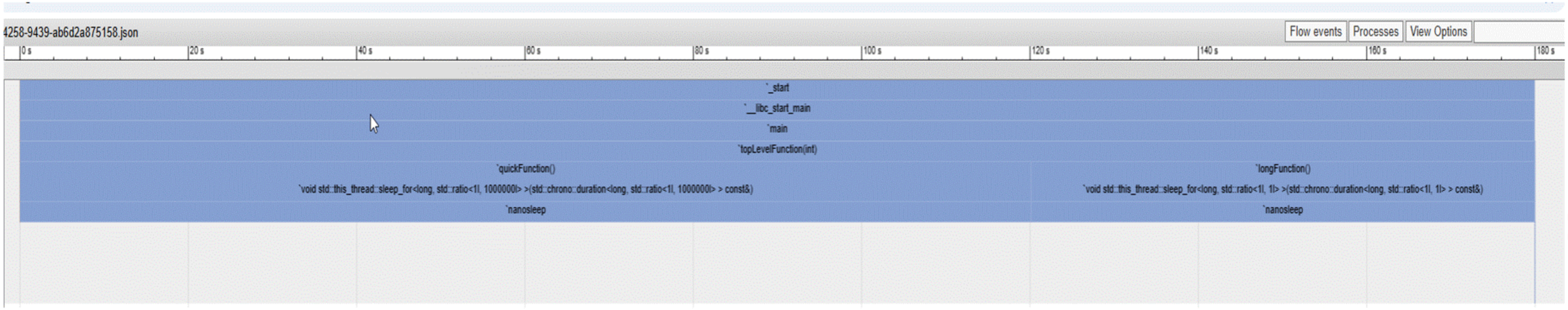
- Manual code modifications
- Using libraries, typically with scope guards
- Statically modifying the binary
- Dynamically modifying the running task

Bloomberg

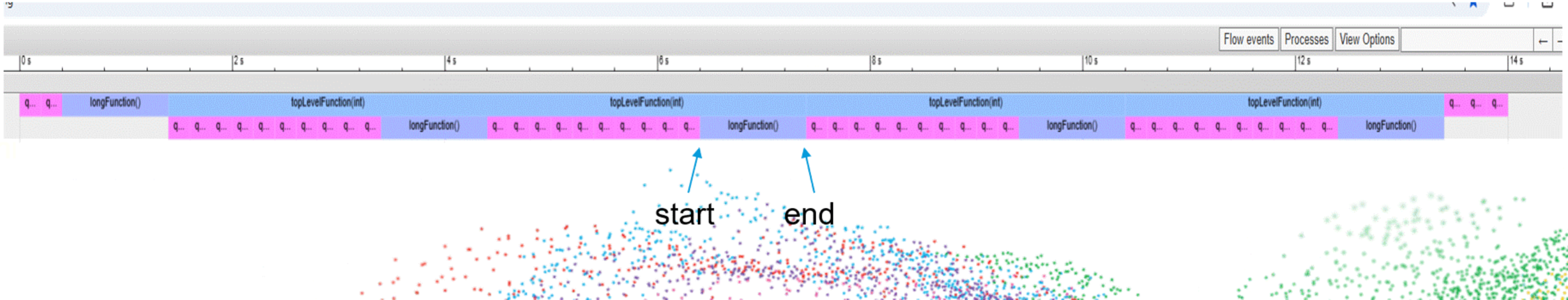
Engineering

Sampling vs. Instrumentation

Sampling

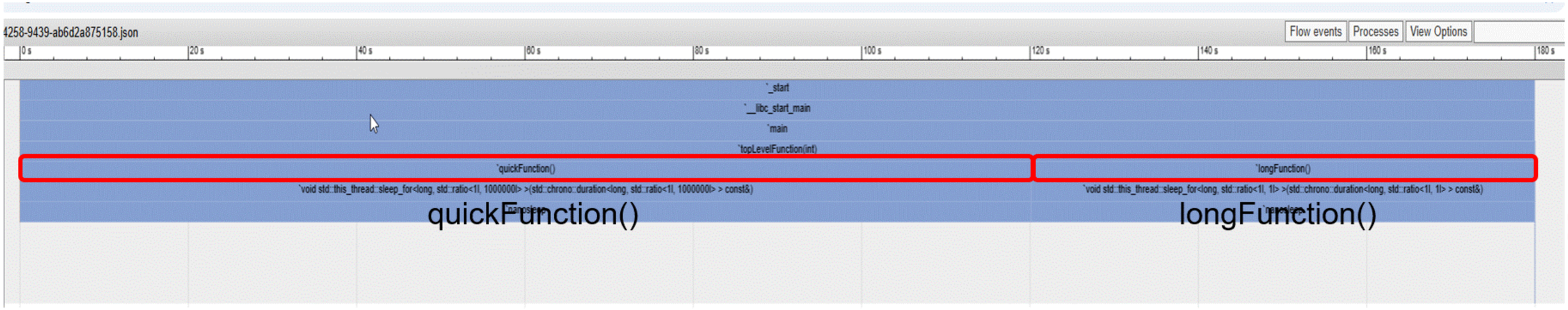


Code instrumentation

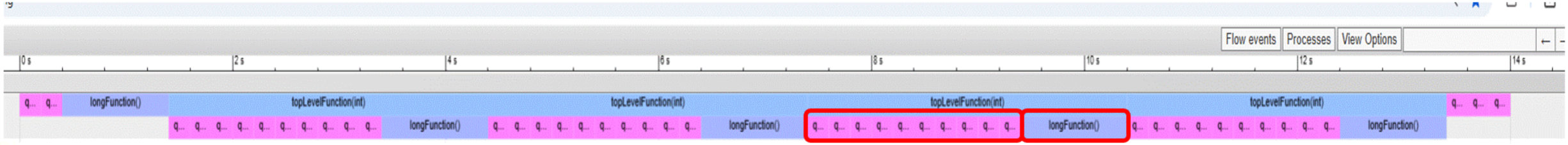


Sampling vs. Instrumentation

Sampling



Code instrumentation



Concurrency analysis

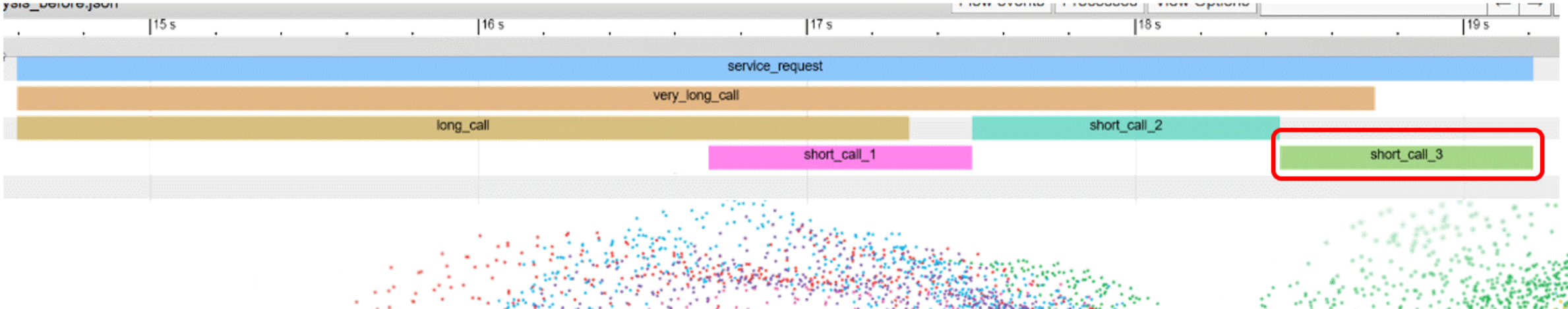
Sampling

Flat Profile

Search: _call

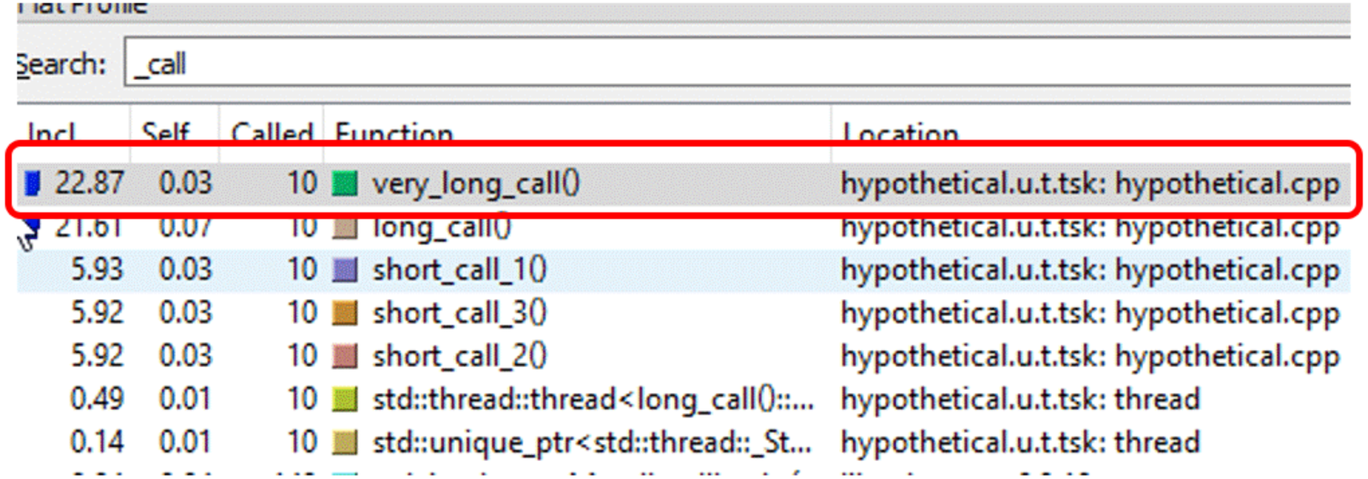
Incl.	Self	Called	Function	Location
22.87	0.03	10	very_long_call()	hypothetical.u.t.tsk: hypothetical.cpp
21.61	0.07	10	long_call()	hypothetical.u.t.tsk: hypothetical.cpp
5.93	0.03	10	short_call_1()	hypothetical.u.t.tsk: hypothetical.cpp
5.92	0.03	10	short_call_3()	hypothetical.u.t.tsk: hypothetical.cpp
5.92	0.03	10	short_call_2()	hypothetical.u.t.tsk: hypothetical.cpp
0.49	0.01	10	std::thread::thread<long_call()::...	hypothetical.u.t.tsk: thread
0.14	0.01	10	std::unique_ptr<std::thread::_St...	hypothetical.u.t.tsk: thread

Code instrumentation

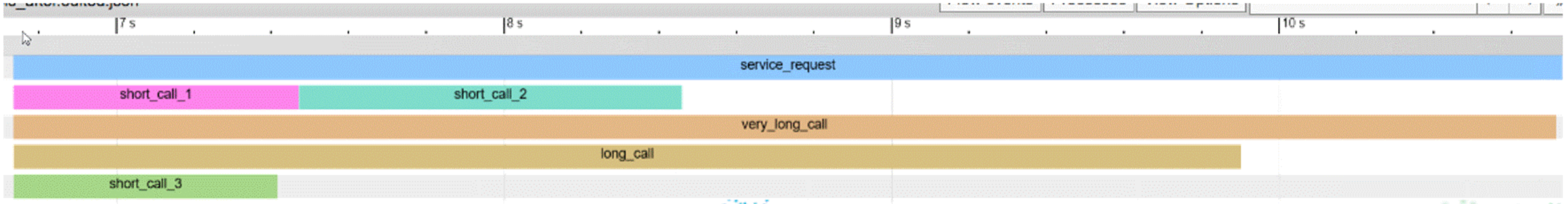


Concurrency analysis

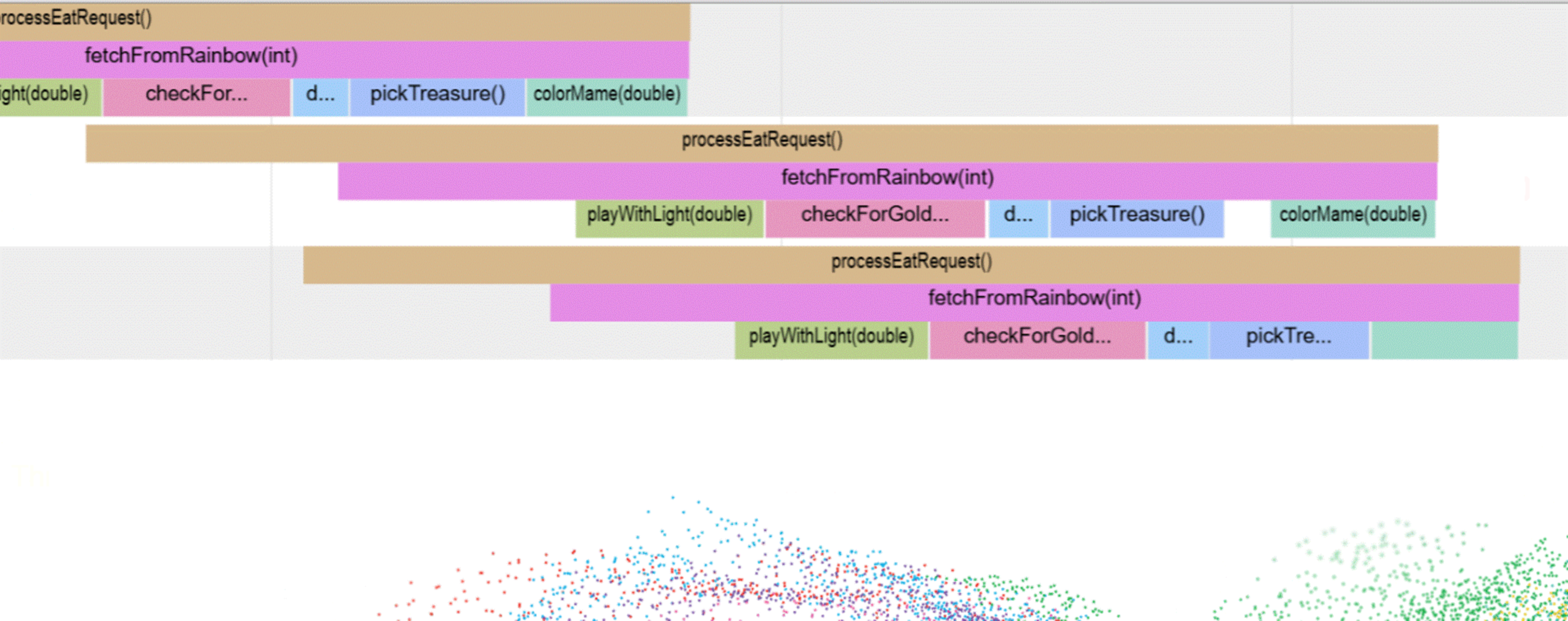
Sampling



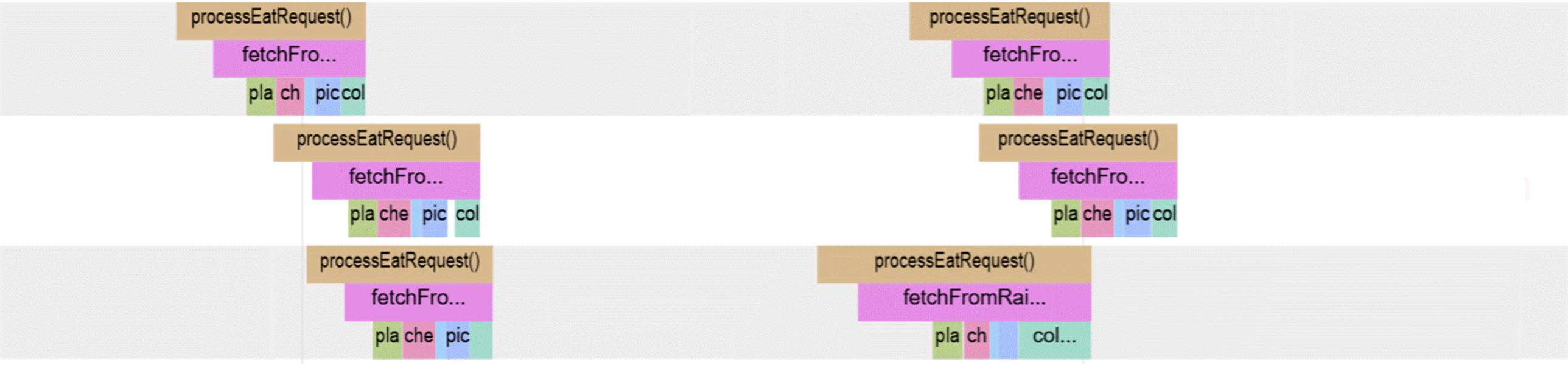
Code instrumentation



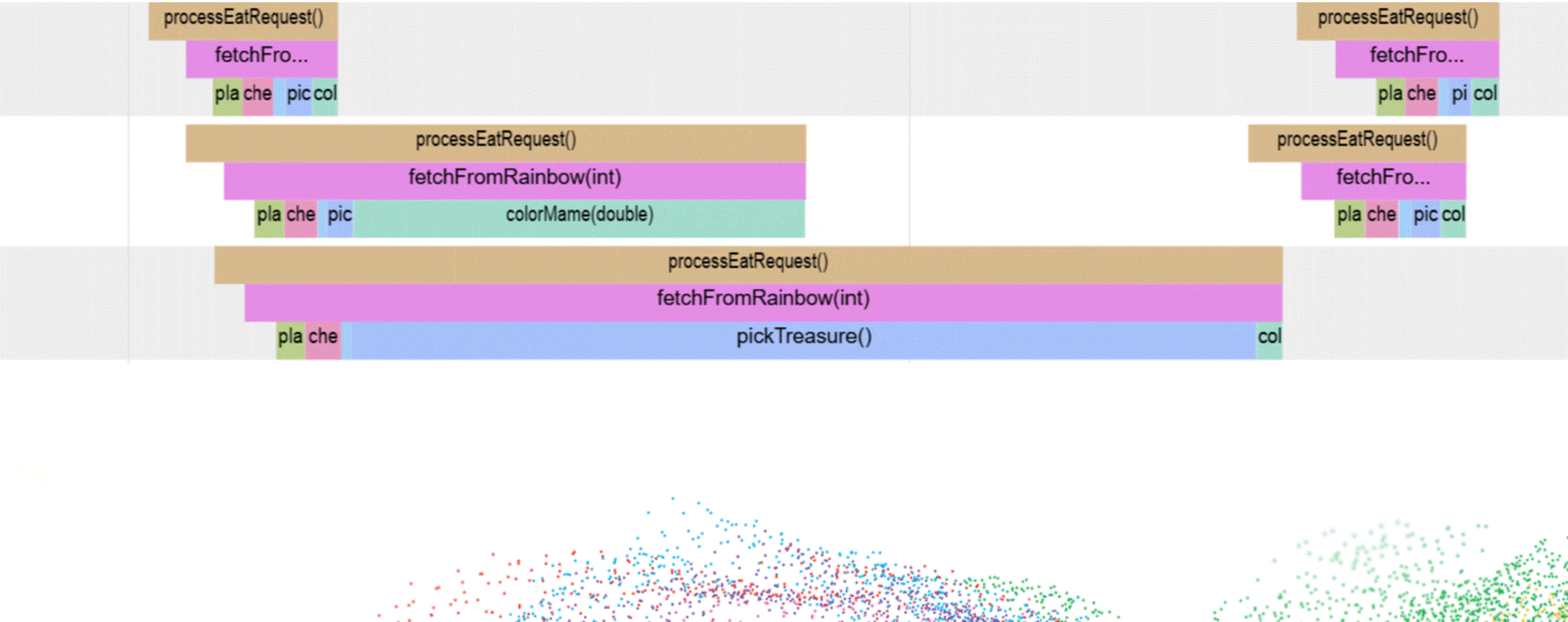
Let’s try instrumentation …



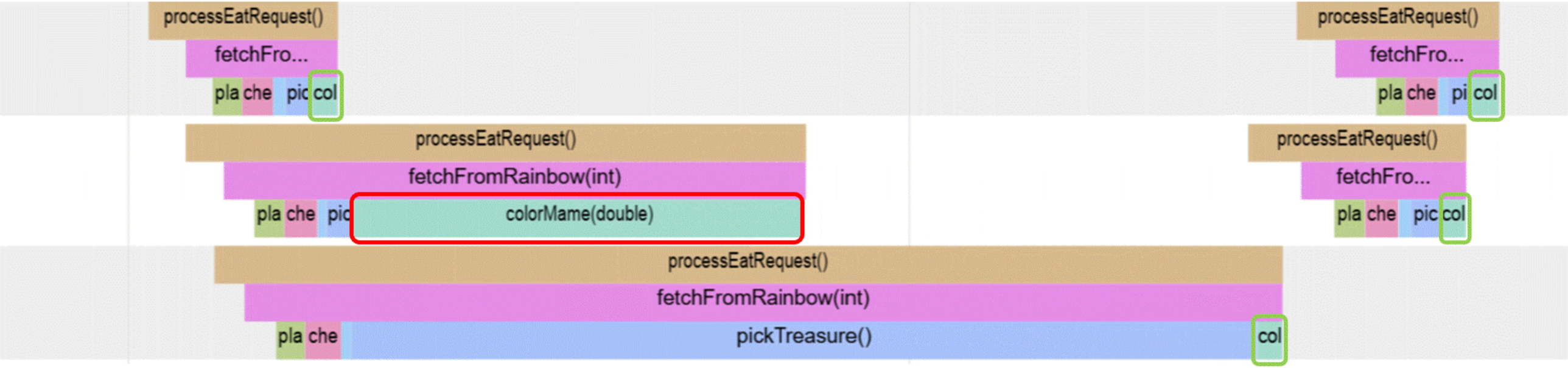
Let’s try instrumentation …



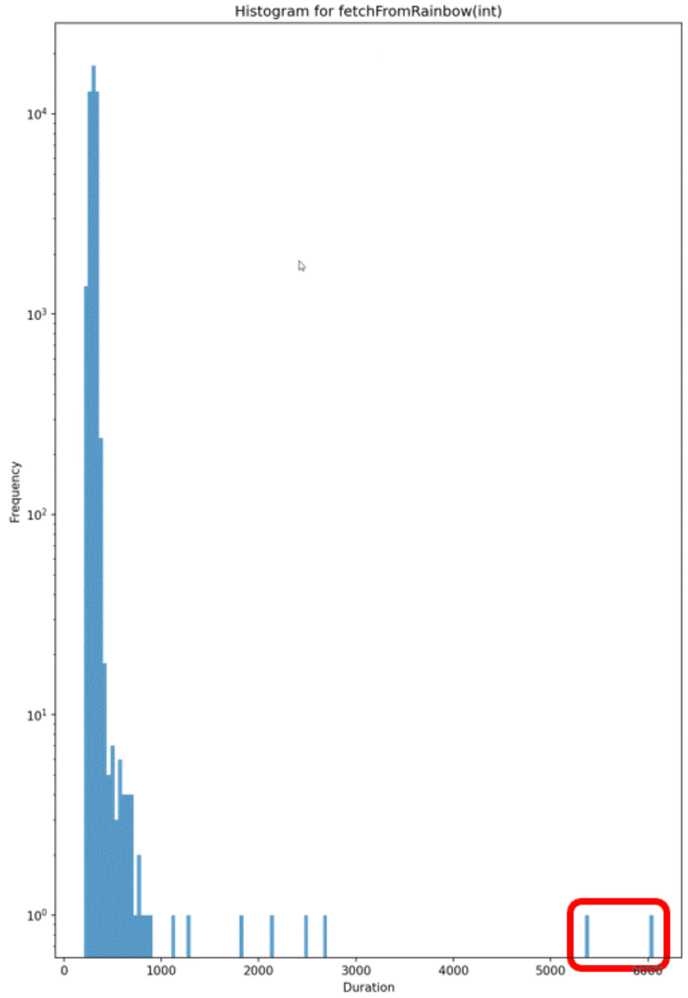
Let’s try instrumentation …

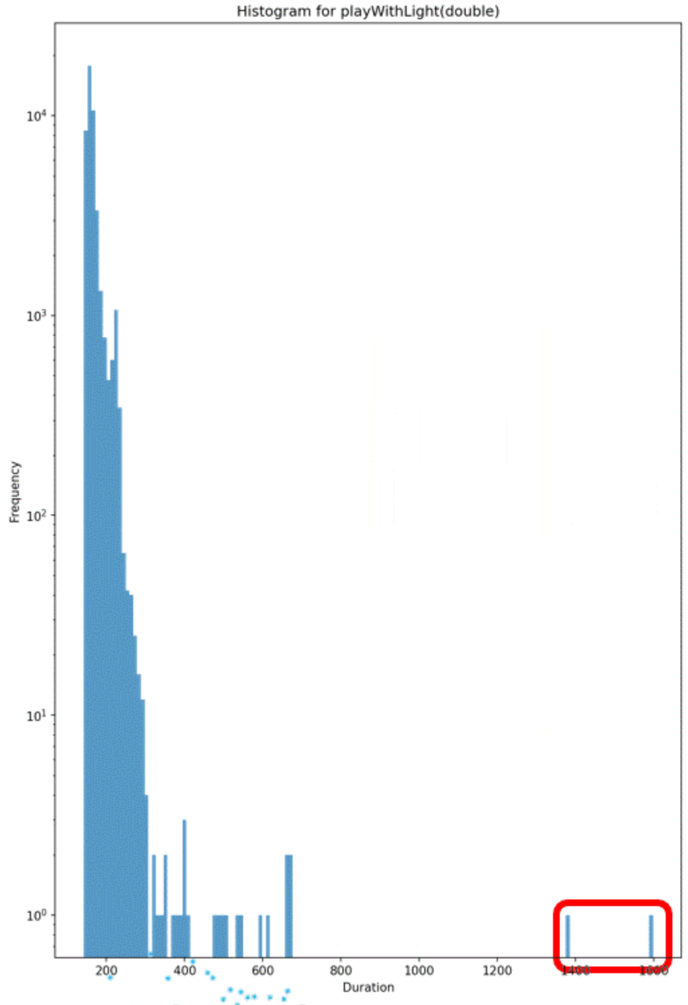


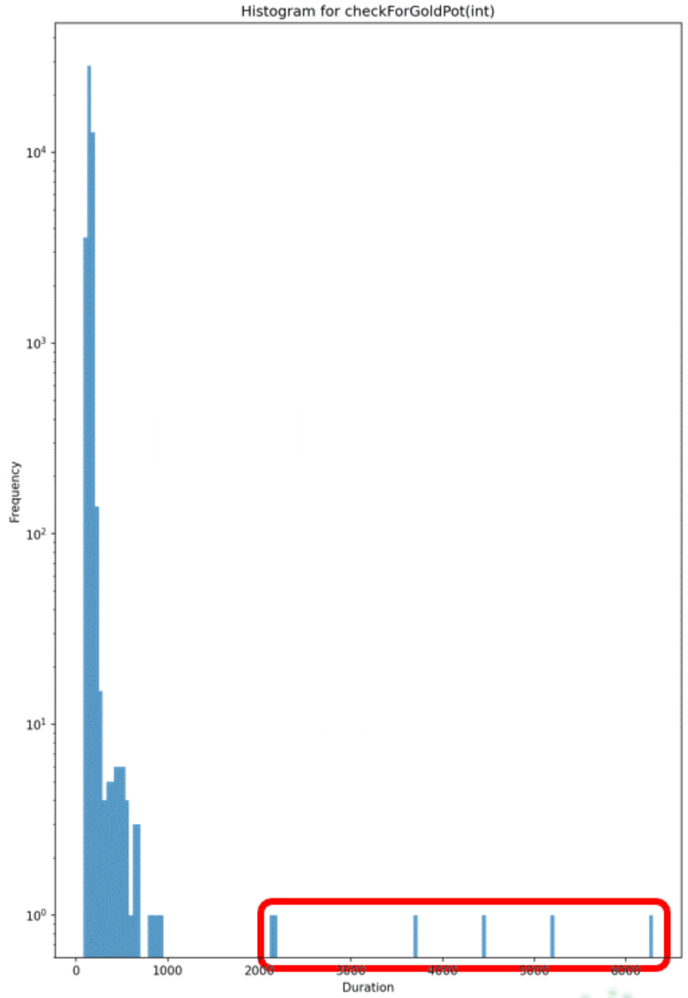
Let’s try instrumentation …



Function histograms







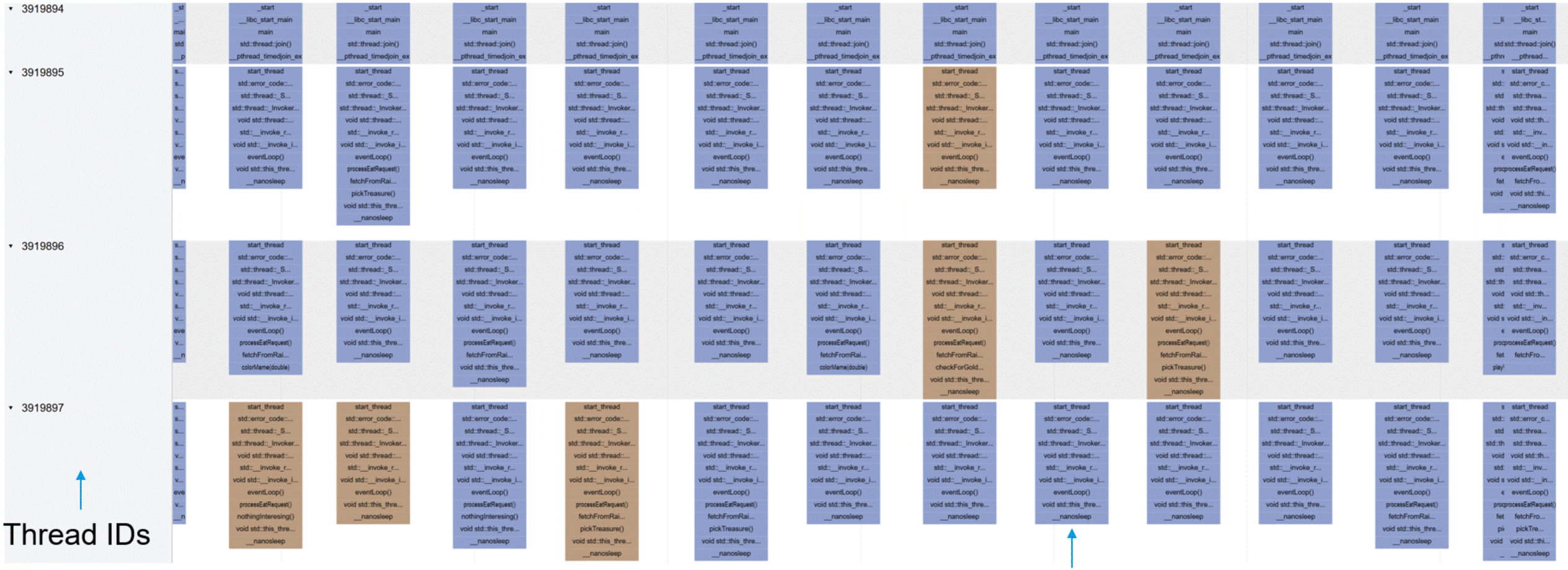
Granular sampling

- Instrumentation only tells you when a function starts and ends
- Granular sampling does not aggregate samples
- Will show where the program is at any given time
- Used a modified version of the sampler

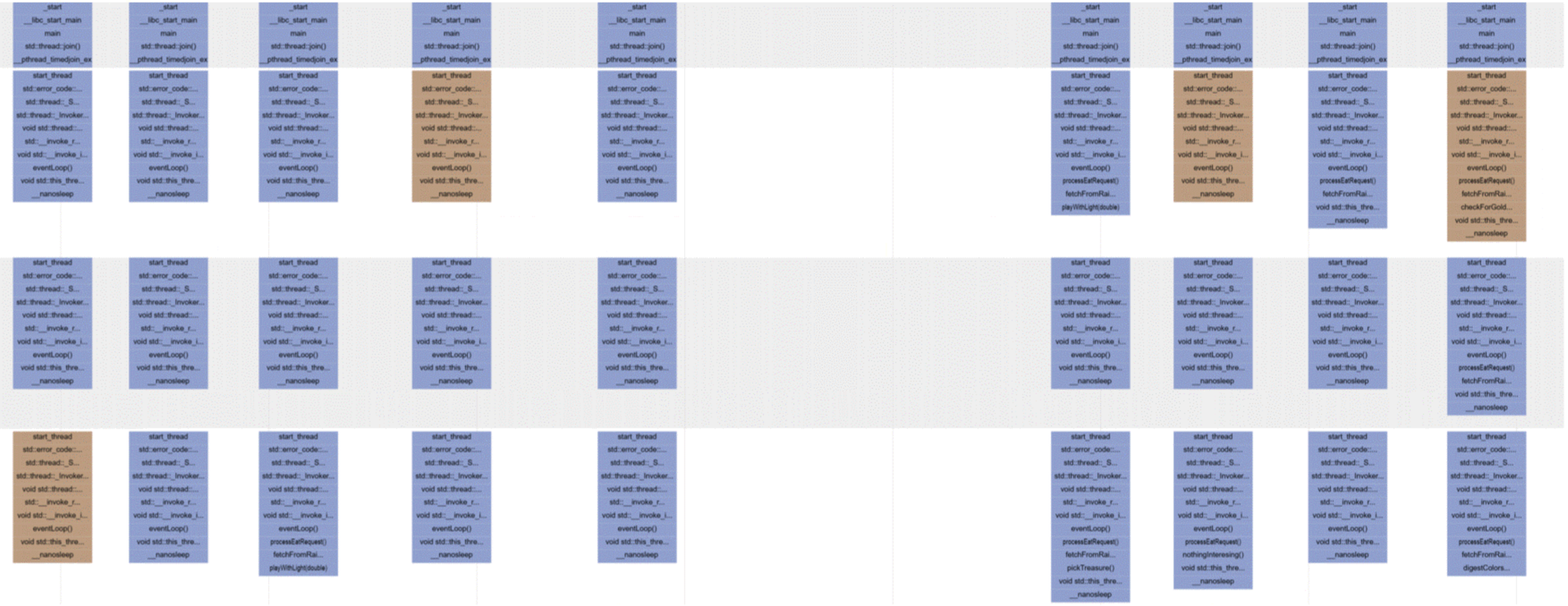
Bloomberg

Engineering

Let’s try sampling again, but granular …



Let’s try sampling again, but granular …



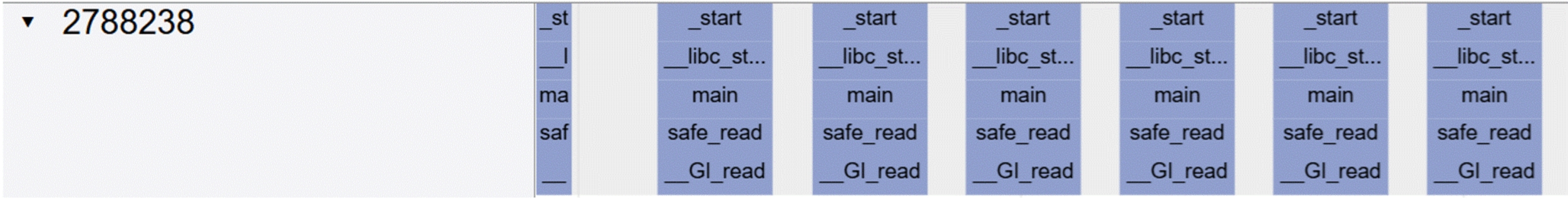
Mysterious gaps

- Threads go into long uninterruptible sleeps?
- Threads trigger spin locks in kernel mode?
- Queries cause major page faults?
- Excessive context switches?

Bloomberg

Engineering

Sampler bias? Or something else?

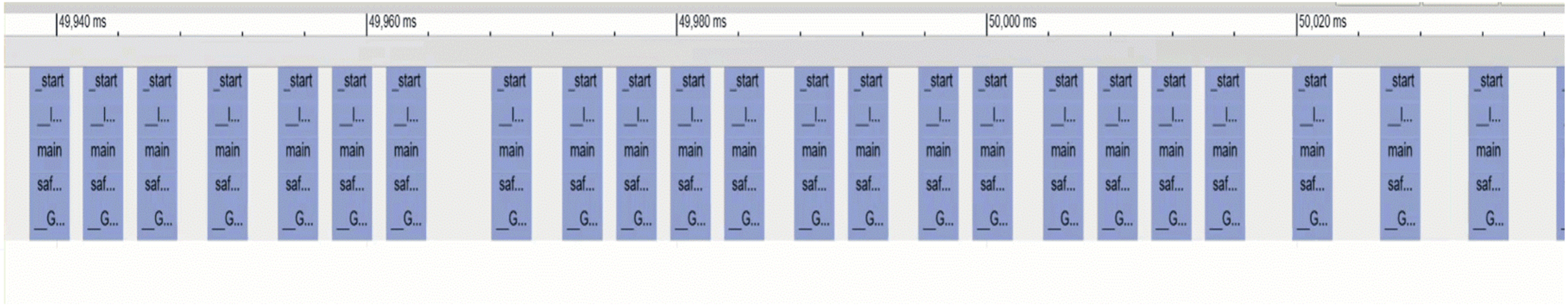


Granular sampling on “cat”

Sampler bias? Or something else?

Granular sampling on “cat”

It was something else …



It was something else …

Investigating the gaps …

```
import time

now = time.time()
end = now + 600.0
worst = 0.0

print("Starting ...")
iterations = 0
while now < end:
    prev = now
    now = time.time()
    elapsed = now - prev
    worst = max(elapsed, worst)

    iterations += 1
    if elapsed > 0.001:
        print(f"Lag found. {now=:<26} {elapsed=:<26} {iterations=:<26}")

print(f"Finished {worst=} {iterations=}")
```

Investigating the gaps …

```
Starting ...
Lag found. now=1718391470.006907        elapsed=0.006851911544799805      iterations=179564
Lag found. now=1718391505.014046        elapsed=0.013989686965942383      iterations=67476614
Lag found. now=1718391510.0080965       elapsed=0.008043527603149414      iterations=77137766
Lag found. now=1718391515.0076745       elapsed=0.007618427276611328      iterations=86748032
Lag found. now=1718391515.0215437       elapsed=0.013869285583496094      iterations=86748033
Lag found. now=1718391520.0146332       elapsed=0.014580011367797852      iterations=96288849
Lag found. now=1718391525.0168524       elapsed=0.016796350479125977      iterations=105913104
Lag found. now=1718391530.0134668       elapsed=0.01341104507446289       iterations=115572973
Lag found. now=1718391535.0213733       elapsed=0.02131962776184082       iterations=125183675
Lag found. now=1718391540.0199544       elapsed=0.019899368286132812      iterations=134847627
Lag found. now=1718391545.0077271       elapsed=0.0076711177825927734     iterations=144248105
Lag found. now=1718391550.0122104       elapsed=0.012154579162597656      iterations=153787796
Lag found. now=1718391555.021808        elapsed=0.021755218505859375      iterations=163208708
Lag found. now=1718391560.017184        elapsed=0.017131567001342773      iterations=172746202
Lag found. now=1718391565.0175037       elapsed=0.017450809478759766      iterations=182168608
Lag found. now=1718391570.0083096       elapsed=0.008256673812866211      iterations=191699086
Lag found. now=1718391575.0074887       elapsed=0.007422924041748047      iterations=201328540
...
Lag found. now=1718391720.0135837       elapsed=0.013527393341064453      iterations=479019089
Lag found. now=1718391725.0105941       elapsed=0.010539531707763672      iterations=487991883
Lag found. now=1718391730.0101373       elapsed=0.010082244873046875      iterations=497494294
Lag found. now=1718391735.015129        elapsed=0.015075922012329102      iterations=506850184
Lag found. now=1718391805.014935        elapsed=0.014881372451782227      iterations=640403993
Lag found. now=1718391810.0159037       elapsed=0.015850067138671875      iterations=650052585
Lag found. now=1718391815.015559        elapsed=0.015505313873291016      iterations=659684604
Lag found. now=1718391820.01105         elapsed=0.01099705696105957       iterations=669311384
Lag found. now=1718391820.0213754       elapsed=0.007797956466674805      iterations=669313870
Lag found. now=1718391825.0236914       elapsed=0.023638248443603516      iterations=678800399
Finished worst=0.02818918228149414 iterations=688259366
```

Investigating the gaps …

```
Starting ...
Lag found. now=1718391470.006907        elapsed=0.006851911544799805      iterations=179564
Lag found. now=1718391505.014046        elapsed=0.013989686965942383      iterations=67476614
Lag found. now=1718391510.0080965       elapsed=0.008043527603149414      iterations=77137766
Lag found. now=1718391515.0076745       elapsed=0.007618427276611328      iterations=86748032
Lag found. now=1718391515.0215437       elapsed=0.013869285583496094      iterations=86748033
Lag found. now=1718391520.0146332       elapsed=0.014580011367797852      iterations=96288849
Lag found. now=1718391525.0168524       elapsed=0.016796350479125977      iterations=105913104
Lag found. now=1718391530.0134668       elapsed=0.01341104507446289       iterations=115572973
Lag found. now=1718391535.0213733       elapsed=0.02131962776184082       iterations=125183675
Lag found. now=1718391540.0199544       elapsed=0.019899368286132812      iterations=134847627
Lag found. now=1718391545.0077271       elapsed=0.0076711177825927734     iterations=144248105
Lag found. now=1718391550.0122104       elapsed=0.012154579162597656      iterations=153787796
Lag found. now=1718391555.021808        elapsed=0.021755218505859375      iterations=163208708
Lag found. now=1718391560.017184        elapsed=0.017131567001342773      iterations=172746202
Lag found. now=1718391565.0175037       elapsed=0.017450809478759766      iterations=182168608
Lag found. now=1718391570.0083096       elapsed=0.008256673812866211      iterations=191699086
Lag found. now=1718391575.0074887       elapsed=0.007422924041748047      iterations=201328540
...
Lag found. now=1718391720.0135837       elapsed=0.013527393341064453      iterations=479019089
Lag found. now=1718391725.0105941       elapsed=0.010539531707763672      iterations=487991883
Lag found. now=1718391730.0101373       elapsed=0.010082244873046875      iterations=497494294
Lag found. now=1718391735.015129        elapsed=0.015075922012329102      iterations=506850184
Lag found. now=1718391805.014935        elapsed=0.014881372451782227      iterations=640403993
Lag found. now=1718391810.0159037       elapsed=0.015850067138671875      iterations=650052585
Lag found. now=1718391815.015559        elapsed=0.015505313873291016      iterations=659684604
Lag found. now=1718391820.01105         elapsed=0.01099705696105957       iterations=669311384
Lag found. now=1718391820.0213754       elapsed=0.007797956466674805      iterations=669313870
Lag found. now=1718391825.0236914       elapsed=0.023638248443603516      iterations=678800399
Finished worst=0.028189182281494 14 iterations=688259366
```

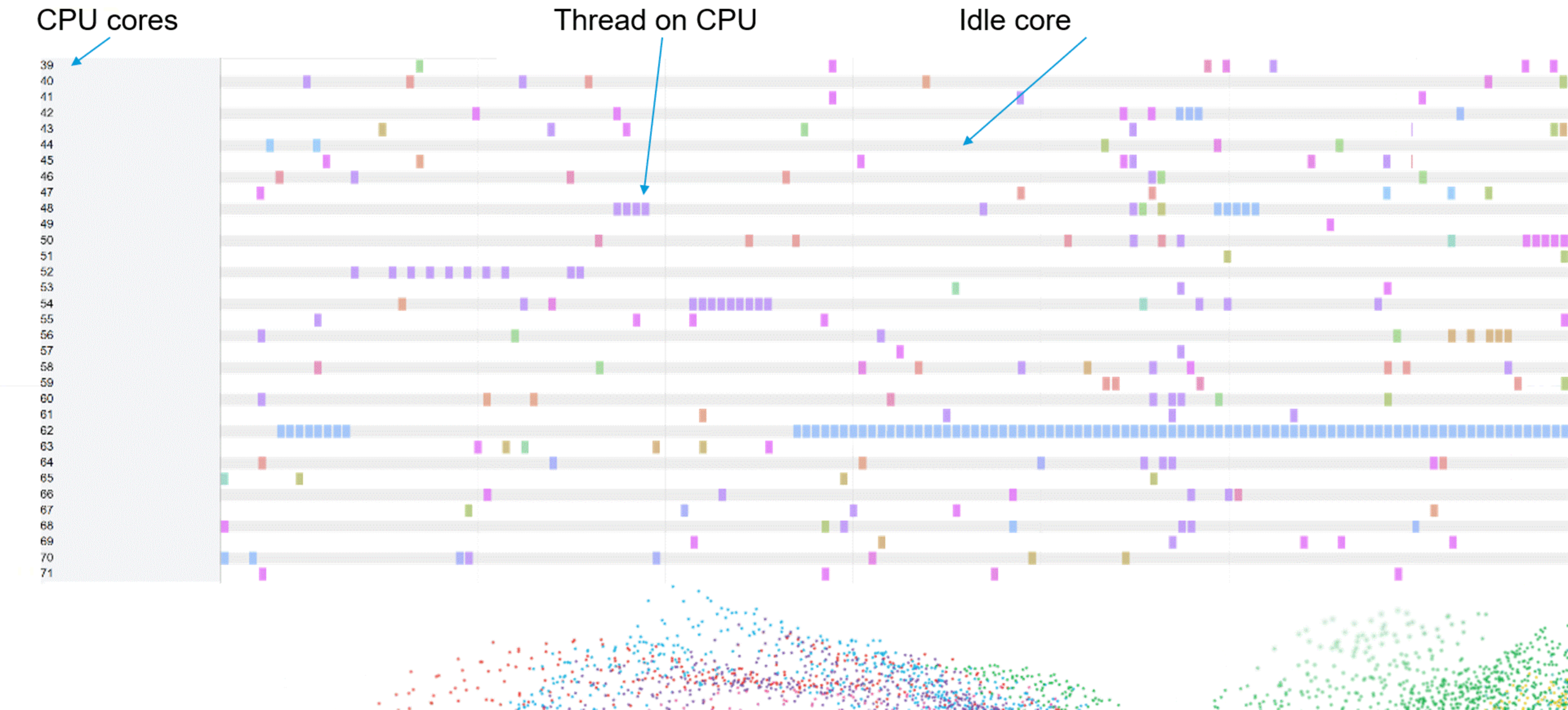
Root powers!

- eBPF (Extended Berkely Packet Filter)
  - Allows you to run sandboxed programs in the kernel
- WhoisonCPU script
  - Periodically sample every core
  - Get the thread name and ID of the thread running on it
  - Get the PID and the PID of the parent
  - Emit the information with a timestamp
- Parse the output into a traceevents file

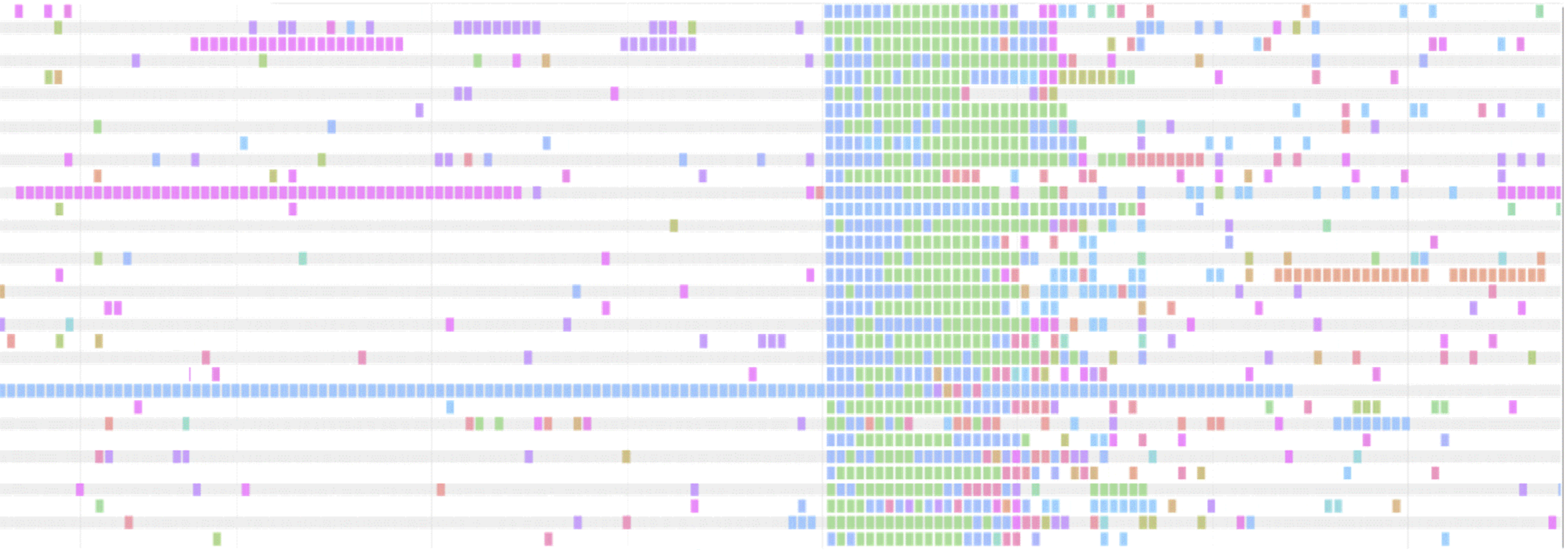
Bloomberg

Engineering

Sampling, every core



Sampling, every core



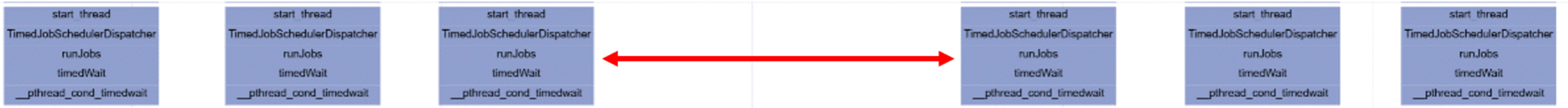
Some naughty threads

- ~~Short lived tasks created by an uncivilized application~~
- Threads created by some common libraries
- A sampling granular trace might reveal what is going on

Bloomberg

Engineering

Some naughty threads



Some naughty threads

TimedJobSchedulerDispatcher()

What is a Timed Job Scheduler?

- Can go by *Timer, Event Scheduler, Event Manager, etc. …*
- Can schedule jobs to run in the future
- Some support starting at a specific time
- Some can schedule recurring jobs
- Some event loops and thread pools have similar functionality

Bloomberg

Engineering

But why the lags?

- Large number of tasks use scheduling for polling, alarms, operation timeouts, heartbeats, batching, etc. …
- These are typically recurring jobs, probably with delays that are multiples of 5 seconds
- Start them at a specific time or aligned at common times
- You'll have a systemic problem

A timed ‘thundering herd’

- A thundering herd can be caused without a common event
- Threads can wake up at the same time by having synchronized recurring jobs
- While they’re not looking at the same event, they are looking at the same clock

Bloomberg

Engineering

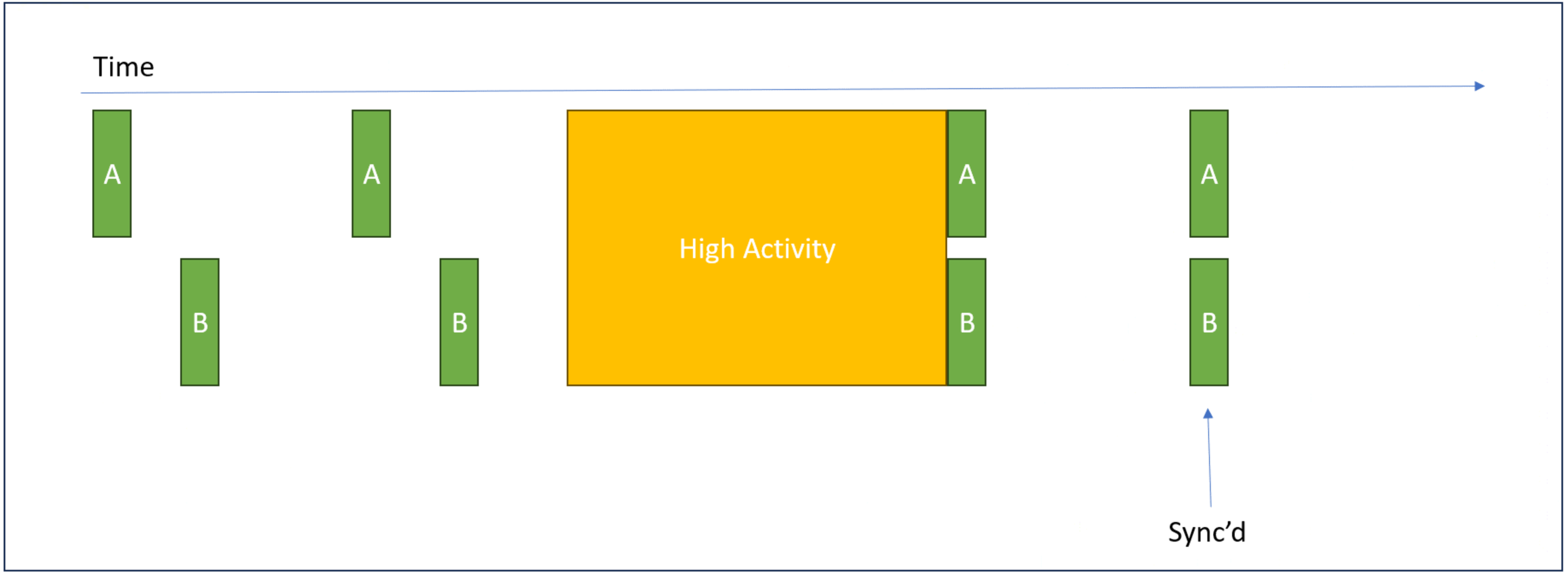
Not only for aligned timers

- ~~Recurring jobs started at random times aren't synchronized~~
- They can become synchronized during the day

Bloomberg

Engineering

Not only for aligned timers



Bloomberg

Engineering

Not only for aligned timers

- ~~Recurring jobs started at random times aren't synchronized~~
- They can become synchronized during the day
- The period of "high activity" can itself be a timed thundering herd

Bloomberg

Engineering

Reflecting on the problem

- Do you really need aligned start timers in your recurring jobs?
- If yes, can they be started at irregular times?
- If polling, can you use some form of subscription instead?
- If batching, can you schedule only if there is work available?
- Can you use prime numbers as delays? Or add jitter?
- Can your custom scheduler calculate all wake times based on a single start time?

Bloomberg

Engineering

Detecting and monitoring this problem

- Continuous monitoring
  1. Launch the script a few times per day at random times
  2. Run the script for 5 minutes (can vary)
  3. Have the script report any lags to your monitoring system

- Assumptions
  1. A timed ‘thundering herd’ with a delay of more than 5 minutes is not a problem
  2. A timed ‘thundering herd’ that happens sporadically is not a problem
  3. A problematic timed thundering herd’ should be caught eventually

Bloomberg

Engineering

If you see something, say something!

narroyo1@bloomberg.net

TechAtBloomberg.com

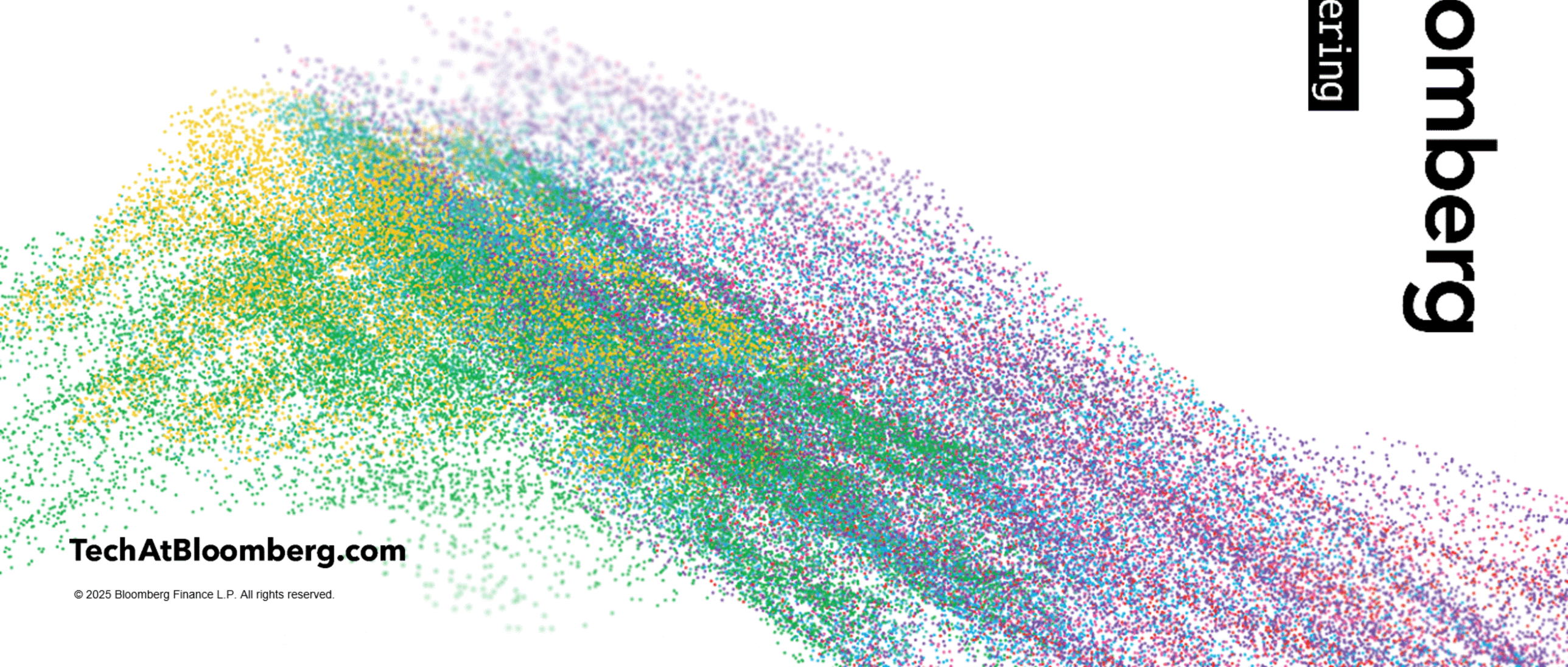
Bloomberg

Engineering

Questions?

Bloomberg

Engineering



TechAtBloomberg.com

Bloomberg Career Opportunities

Bloomberg

Engineering

TechAtBloomberg.com