Throttling: New Developments in Application Performance with CPU Limits

Dave Chiluk, Linux Platform Software Engineer

Welcome. Here's what we'll cover today.

- How container CPU constraints work
- Reproducing the throttling problem
- The root cause
- Solutions and workarounds

indeed

Throttling: New Developments in Application Performance with CPU Limits

Dave Chiluk

Linux Platform Software Engineer, Indeed

Dave Chiluk
Linux Platform Software Engineer


indeed

We help people get jobs.

250M unique monthly visitors[1]

25M jobs

1M employers with sponsored jobs

10k employees

[1] Source: Google Analytics, Unique Visitors, September 2018

Roadmap

CPU Limit Basics

The Problem

Reproducing the Problem

Solution and Workarounds

Roadmap

CPU Limit Basics

The Problem

Reproducing the Problem

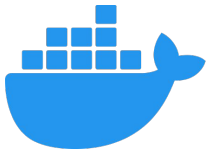
Solution and Workarounds

Who should care about CPU limits?

EVERY **CONTAINER ORCHESTRATOR** ON THE PLANET

kubernetes

Apache MESOS™



docker

Setting CPU Limits in Kubernetes

```
"resources": {
        "limits": {
          "cpu": "200m",
        },
        "requests": {
          "cpu": "134m",
        }
      }
```

Hard Limit

Soft Limit

`Located in pod definition @spec.containers[].resources.`

Soft Limits

.requests = Soft Limits - Cgroup cpu.shares

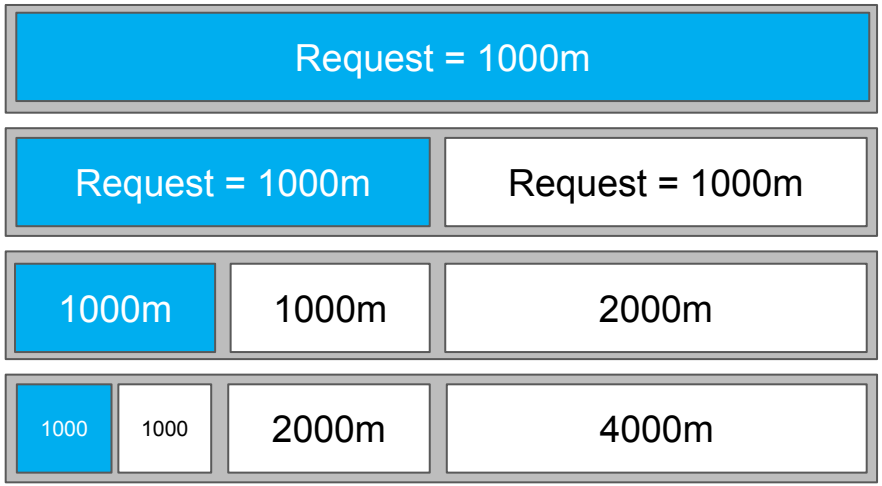
```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.shares
1024
```

Soft Limits

$$CpuTime = \frac{shares}{\sum shares} * NumCPUs$$

Soft Limits

Actual Usable CPU on 88 Core Machine



Max 88 Cores, Min 88

Max 88 Cores, Min 44

Max 88 Cores, Min 22

Max 88 Cores, Min 11

The Floor for Usable CPU

Hard Limits

.limits = Hard Limits - Cgroup CFS Bandwidth Control

Containers are limited to using quota amount of CPU time in a period.

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.cfs_period_us
100000
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.cfs_quota_us
10000
```

THROTTLING

The Ceiling for Usable CPU

Throttling metrics

Throttled time

nr_periods

nr_throttled

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.stat
nr_periods 128
nr_throttled 124
throttled_time 5664985136
```

Throttling metrics

Throttled time

nr_periods

nr_throttled

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.stat
nr_periods 128
nr_throttled 124
throttled_time 5664985136
```

Throttled time is the sum total time a thread in a cgroup was throttled

Throttling metrics

Throttled time

nr_periods

nr_throttled

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.stat
nr_periods 128
nr_throttled 124
throttled_time 5664985136
```

Throttled time is the sum total time a thread in a cgroup was throttled

nr_periods is the number of periods the application was running

Throttling metrics

Throttled time

nr_periods

nr_throttled

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.stat
nr_periods 128
nr_throttled 124
throttled_time 5664985136
```

Throttled time is the sum total time a thread in a cgroup was throttled

nr_periods is the number of periods the application was running

nr_throttled is number of those periods the application was throttled

Throttling metrics

```
dchiluk@cando:/sys/fs/cgroup/cpu,cpuacct/user.slice/fibtest$ cat cpu.stat
nr_periods 128
nr_throttled 124
throttled_time 5664985136
```

Throttled time

nr_periods

nr_throttled

Throttled time is the sum total time a thread in a cgroup was throttled

nr_periods is the number of periods the application was running

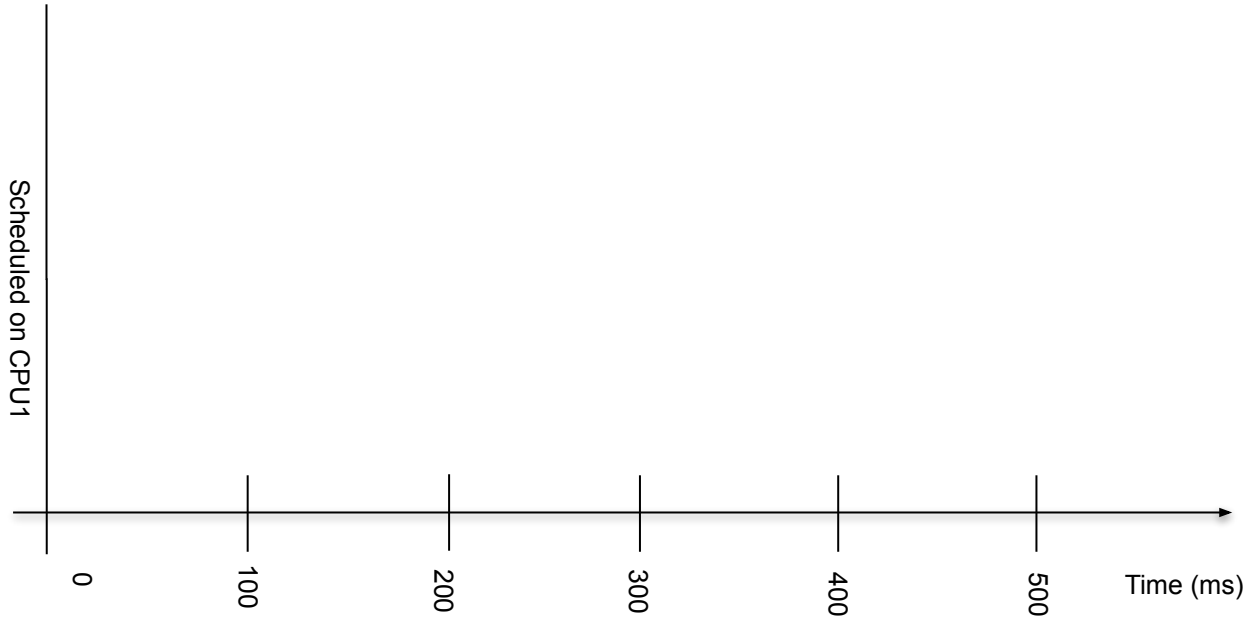
nr_throttled is number of those periods the application was throttled

Throttled percentage

$$throttled\% = \frac{\Delta nr\_throttled}{\Delta nr\_periods}$$

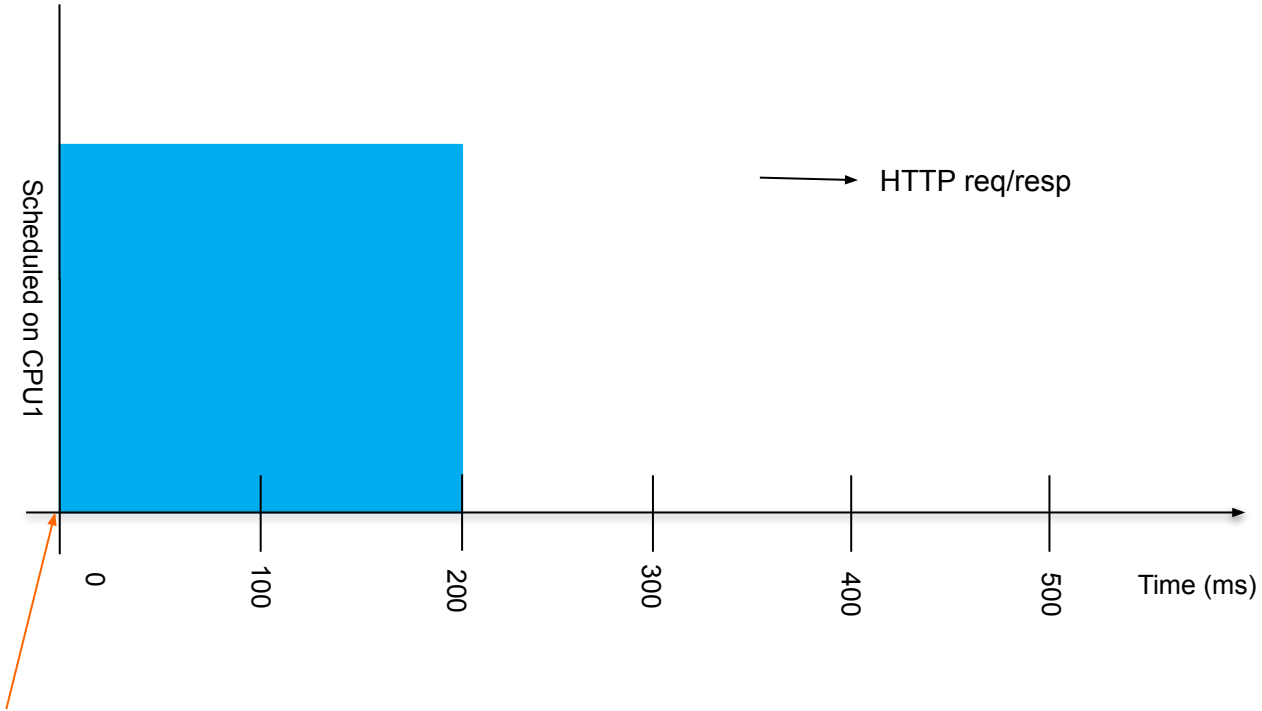
Conceptual Model: Unconstrained

CPU time required 200ms



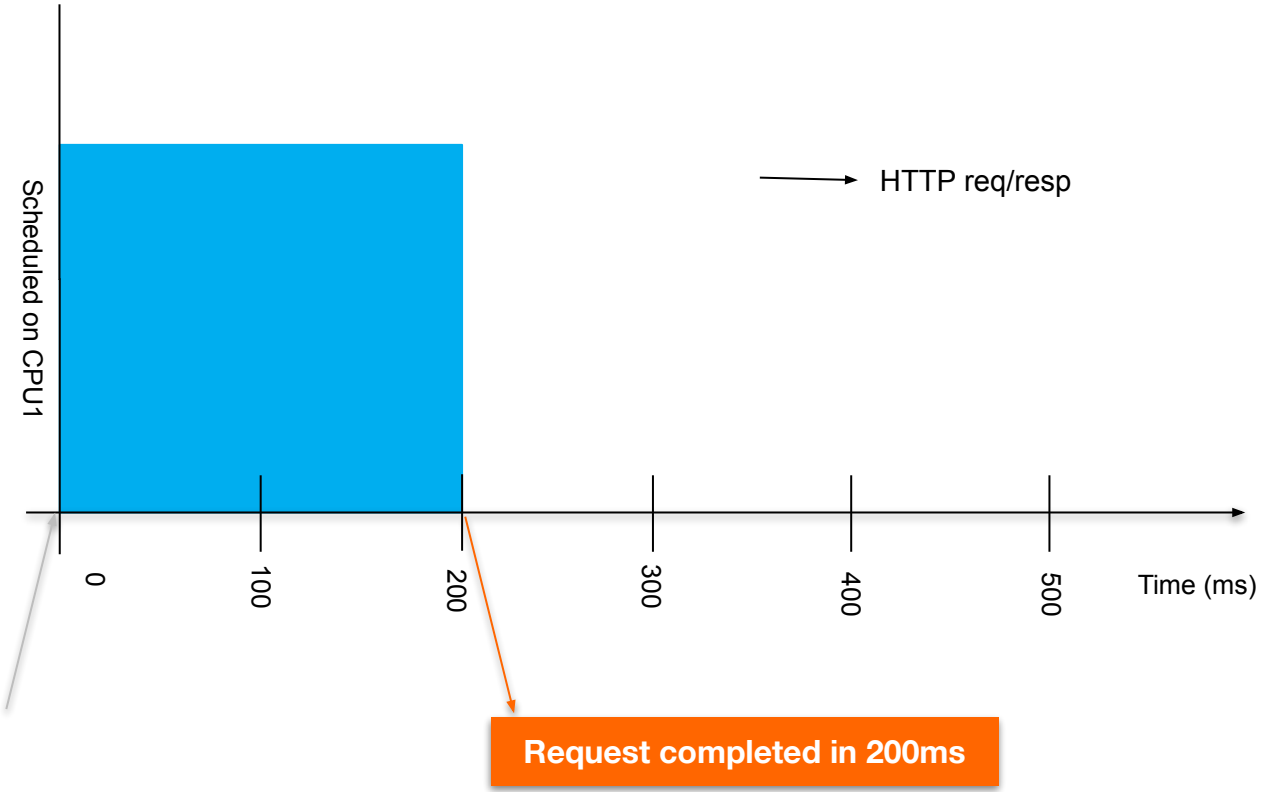
Conceptual Model: Unconstrained

CPU time required 200ms



Conceptual Model: Unconstrained

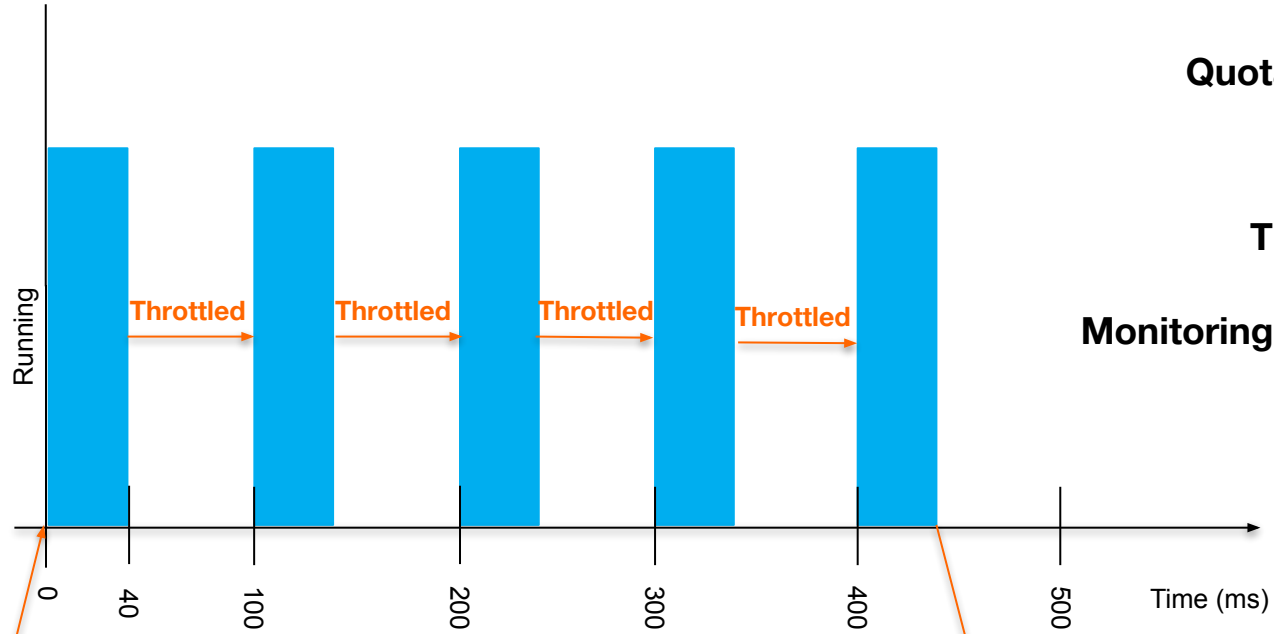
CPU time required 200ms



Conceptual Model: CFS Bandwidth Control

CPU time required 200ms
CPU quota .4 CPU
Quota per period 40ms

CPU Usage 200ms
Throttle Time 240ms
Throttled % 80%
Monitoring CPU Usage .4 CPU



Request

Request completed in 440ms

Roadmap

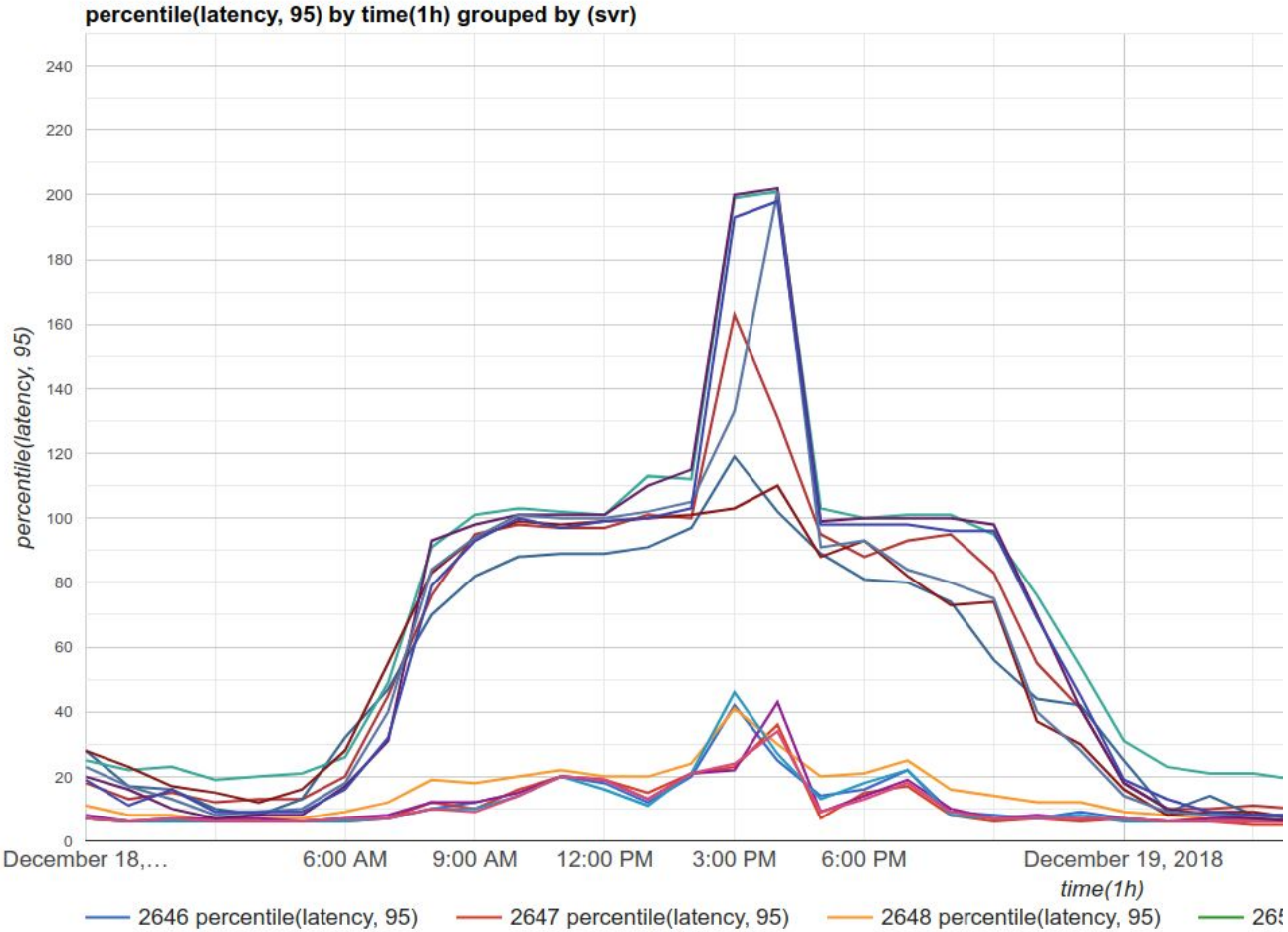
CPU Limit Basics

The Problem

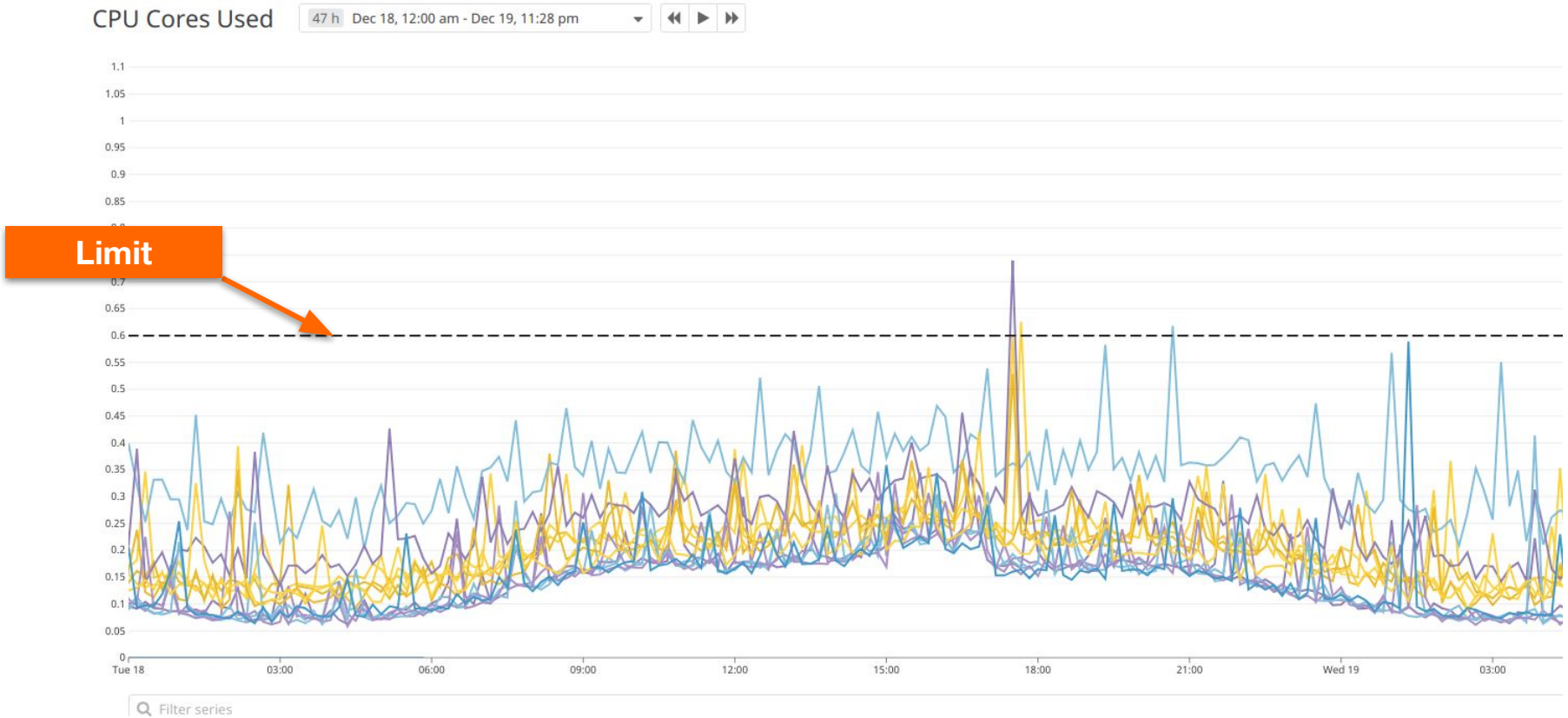
Reproducing the Problem

Solution and Workarounds

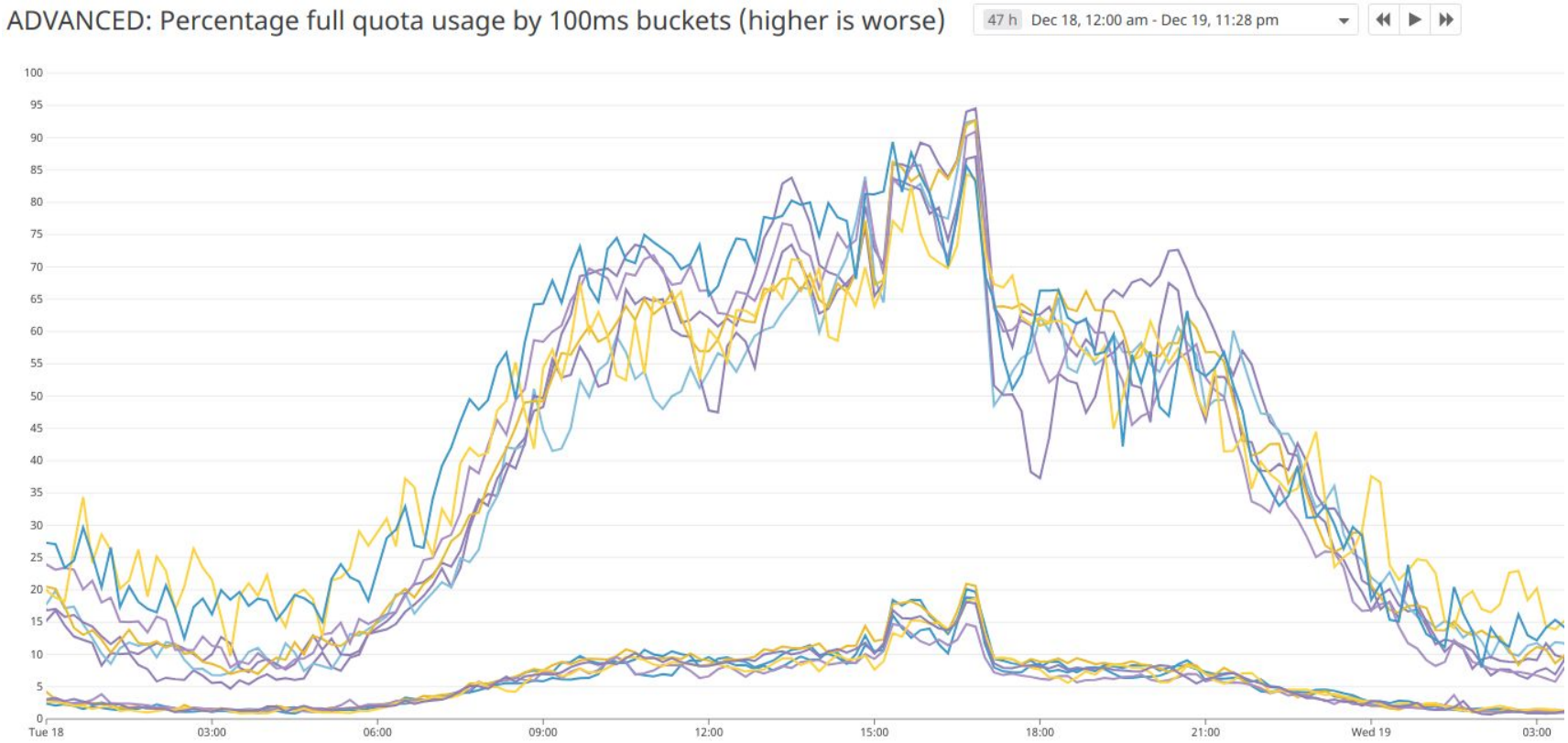
Latency Issues



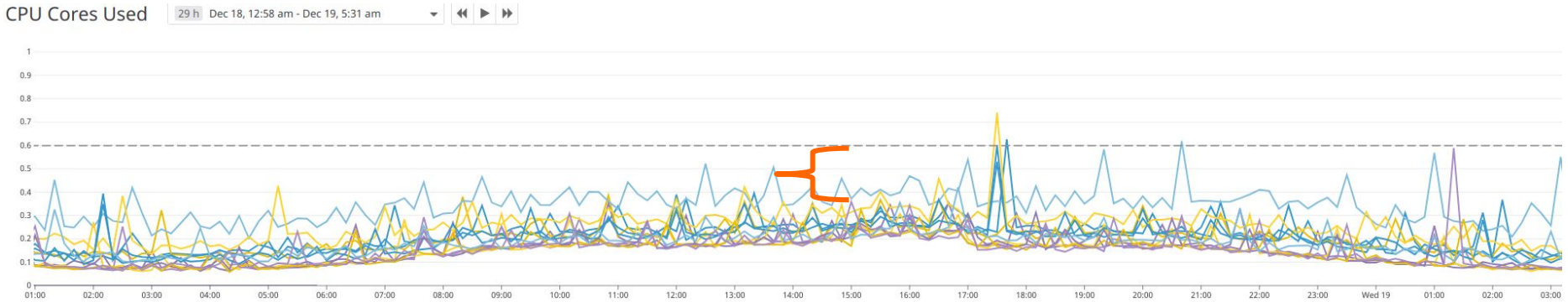
CPU Usage

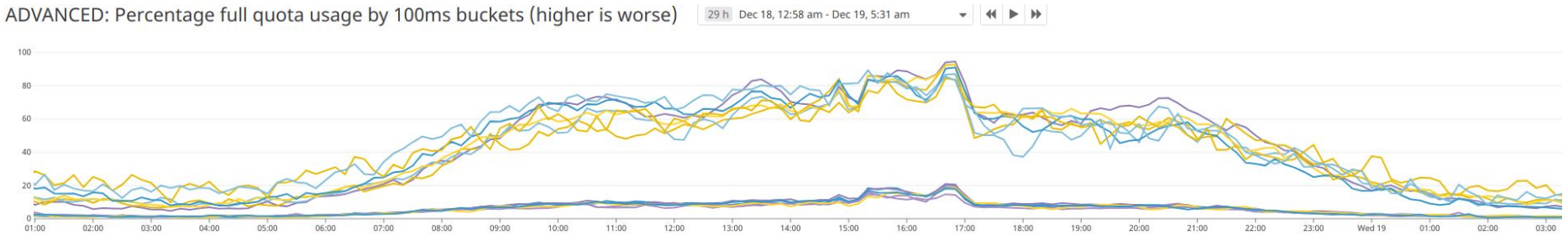

CPU Cores Used
47 h
Dec 18, 12:00 am - Dec 19, 11:28 pm
Limit
1.1
1.05
1
0.95
0.9
0.85
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Tue 18
03:00
06:00
09:00
12:00
15:00
18:00
21:00
Wed 19
03:00
Filter series

Throttling

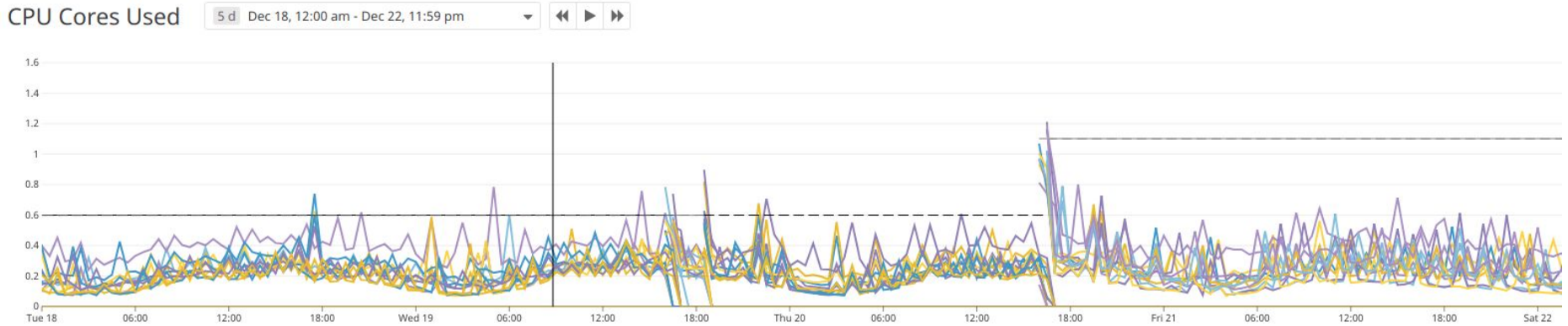

ADVANCED: Percentage full quota usage by 100ms buckets (higher is worse)
47 h
Dec 18, 12:00 am - Dec 19, 11:28 pm
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
0
Tue 18
03:00
06:00
09:00
12:00
15:00
18:00
21:00
Wed 19
03:00

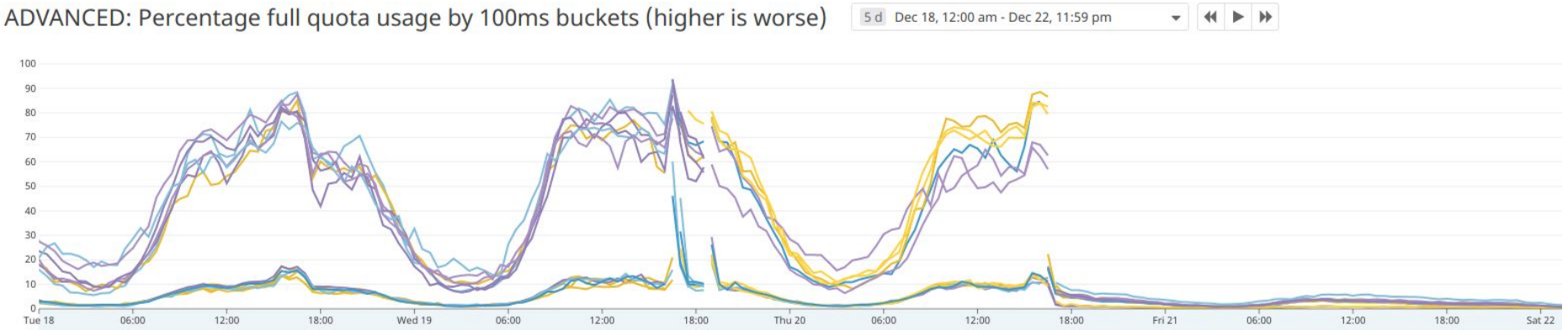
Low CPU Usage but High Throttling?



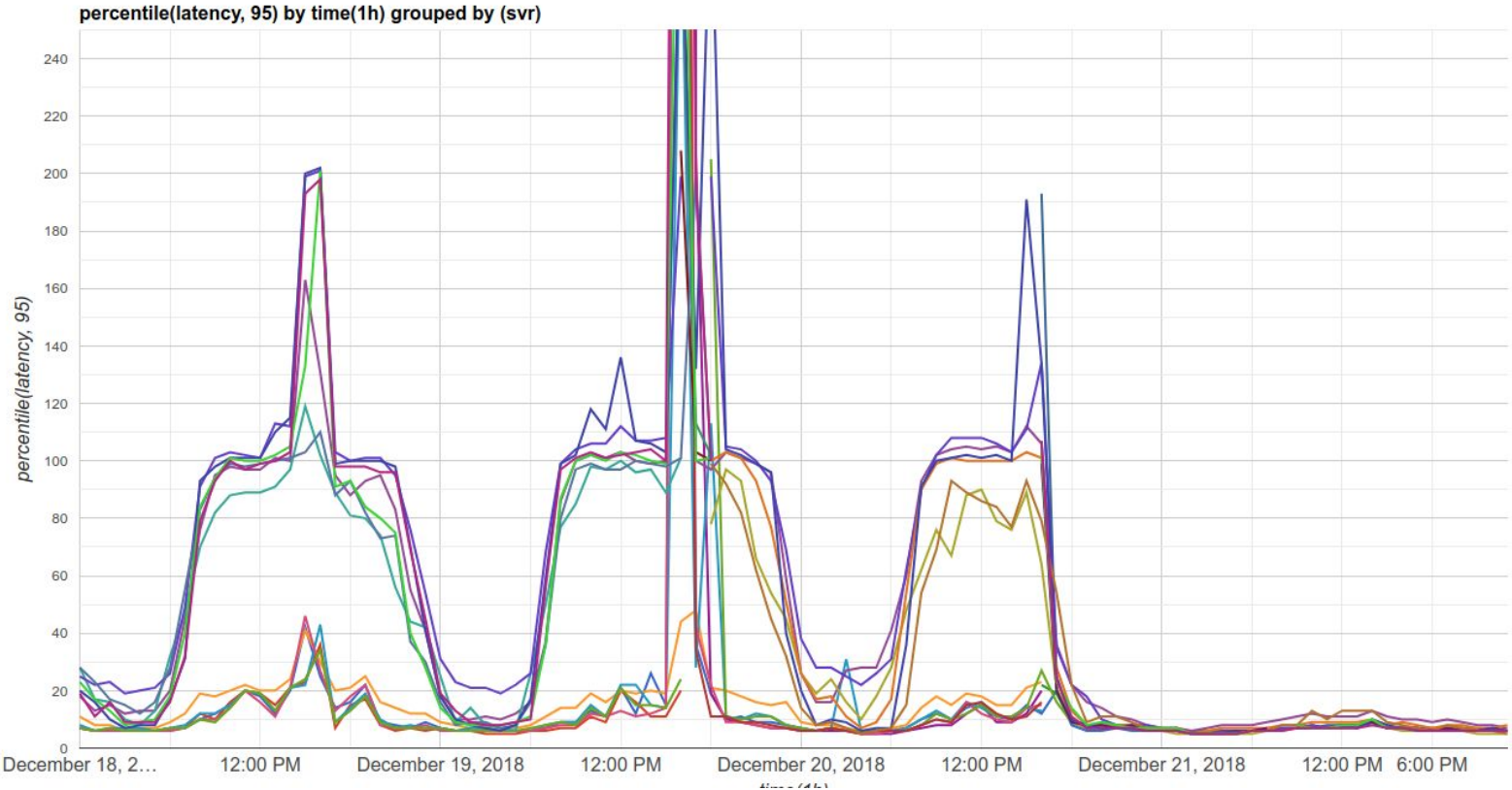


After increasing Limit

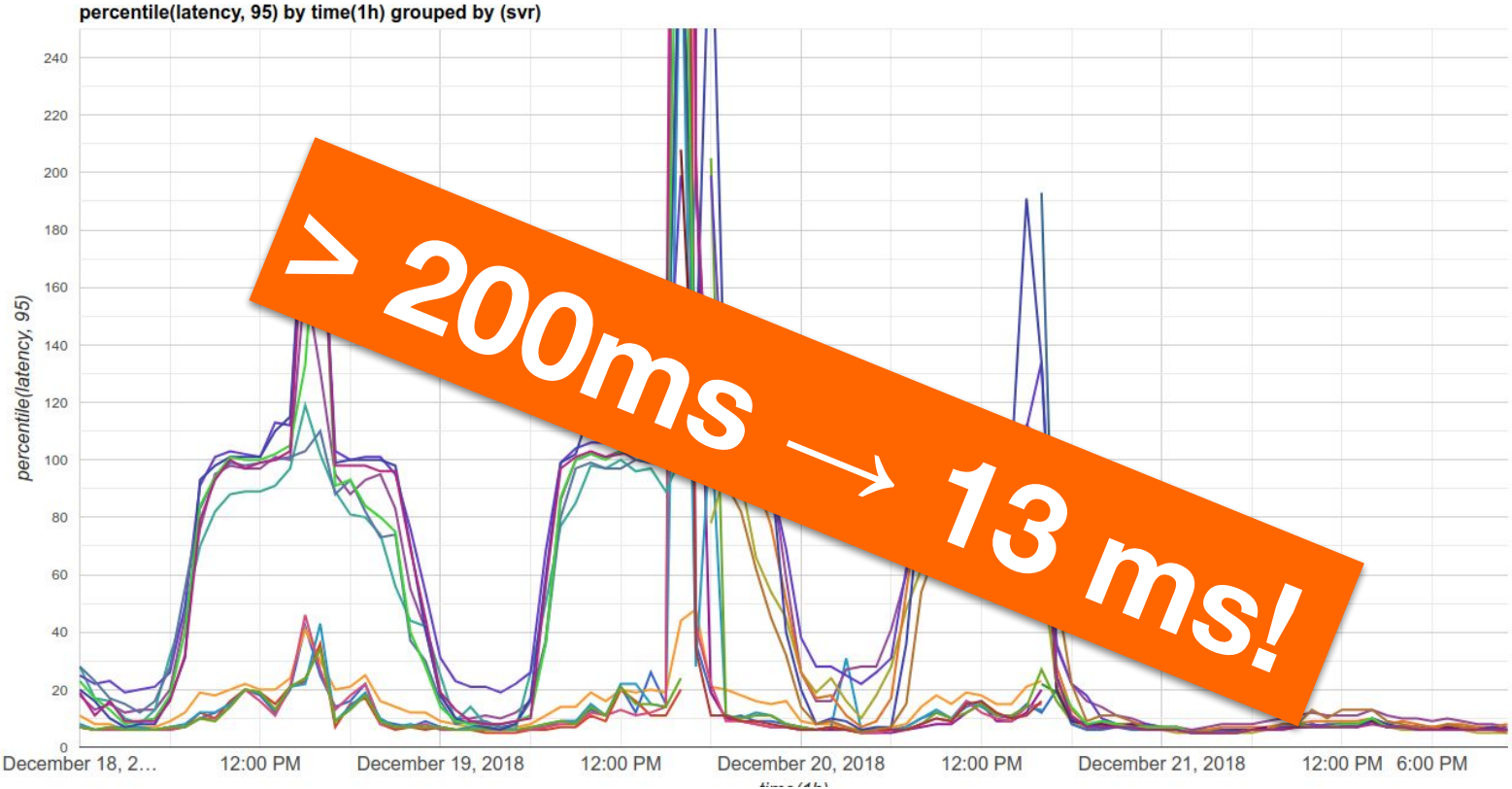




After increasing Limit



After increasing Limit



What we know

What we know

➔ Workarounds

+ Increasing CPU quota mitigates throttling

What we know

➔ Workarounds

+ Increasing CPU limit mitigates throttling

➔ Possible root causes

+ High Core Count ?
+ CPU architecture ?
+ Kernel version ?
+ Spectre-meltdown mitigations ?

Roadmap

CPU Limit Basics

The Problem

Reproducing the Problem

Solution and Workarounds

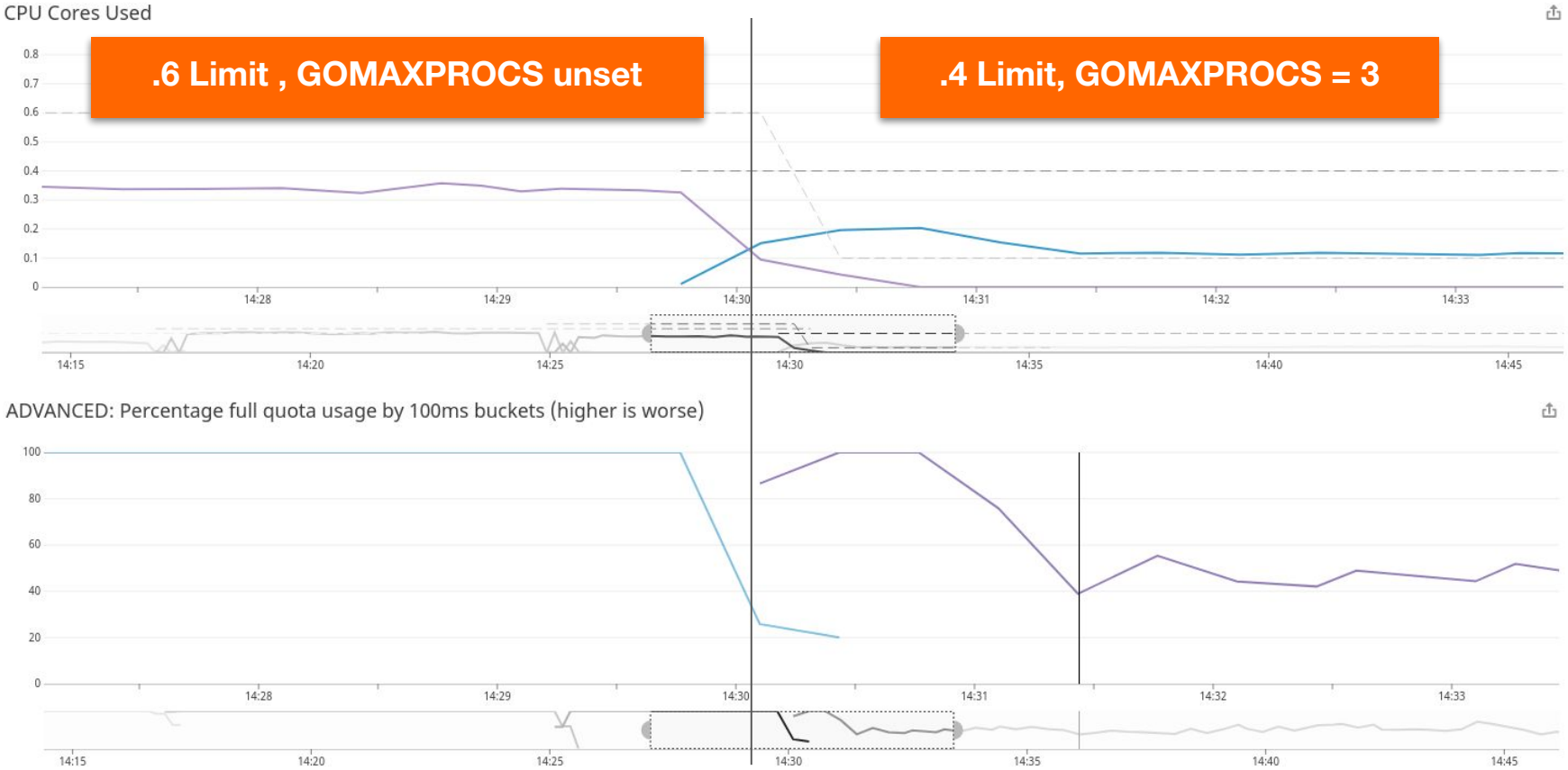
Reproducing

+ **ab**
+ **stress-ng**
+ <bash>

```
for (( i=1 ; i <= 1000 ; i++ )) ; do
    curl -s http://127.0.0.1:8888/info/healthcheck 2>&1 >/dev/null &
    sleep .005s
done

# Report amount of throttling
</bash>
```

GOMAXPROCS


CPU Cores Used
.6 Limit , GOMAXPROCS unset
.4 Limit, GOMAXPROCS = 3
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
14:28
14:29
14:30
14:31
14:32
14:33
14:15
14:20
14:25
14:30
14:35
14:40
14:45
ADVANCED: Percentage full quota usage by 100ms buckets (higher is worse)
100
80
60
40
20
0
14:28
14:29
14:30
14:31
14:32
14:33
14:15
14:20
14:25
14:30
14:35
14:40
14:45

What We Know

Workarounds

+ Increase CPU quota
+ Decrease number of threads in the application
    + Golang - set GOMAXPROCS
    + Java - move to newer JVMs that are cgroup aware.
+ Move from fractional to whole cpu shares.

To Do

+ Create Custom Reproducer
+ Fix Kernel Scheduler

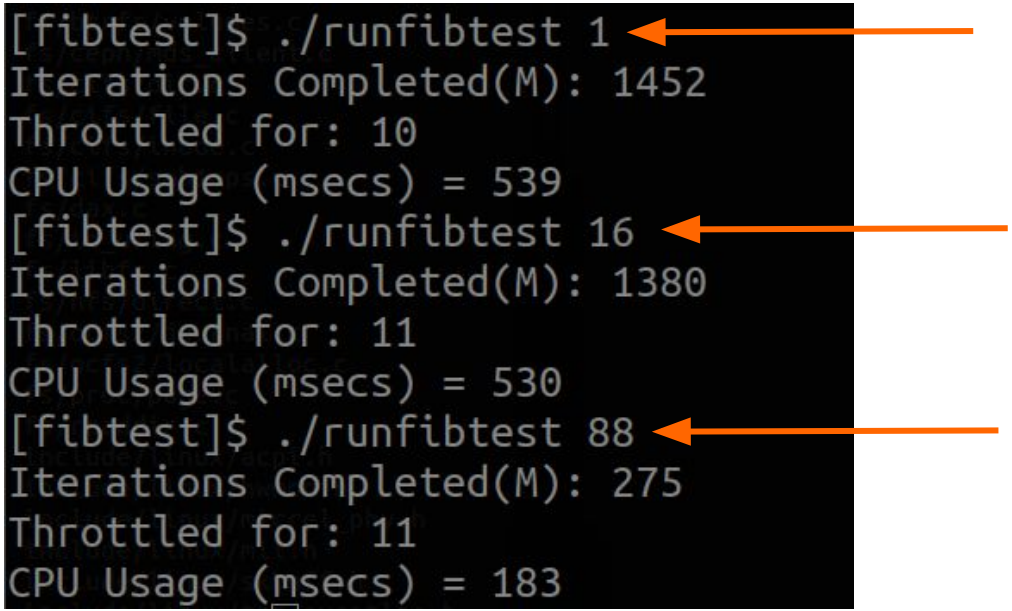
The Reproducer: Fibtest

Fibtest

Multithreaded calculation of the Fibonacci sequence

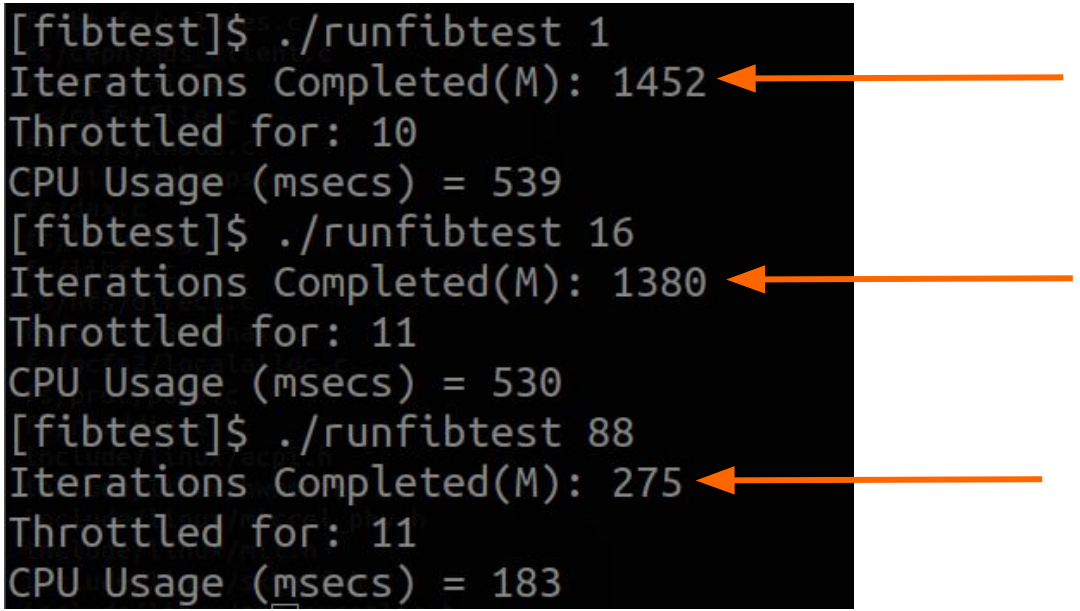
+ Fast threads – calculate as fast as possible
+ Slow threads – calculate 100 iterations, then sleep for 10ms
+ Each thread is pinned to it's own CPU
+ https://github.com/indeedeng/fibtest

Running Fibtest

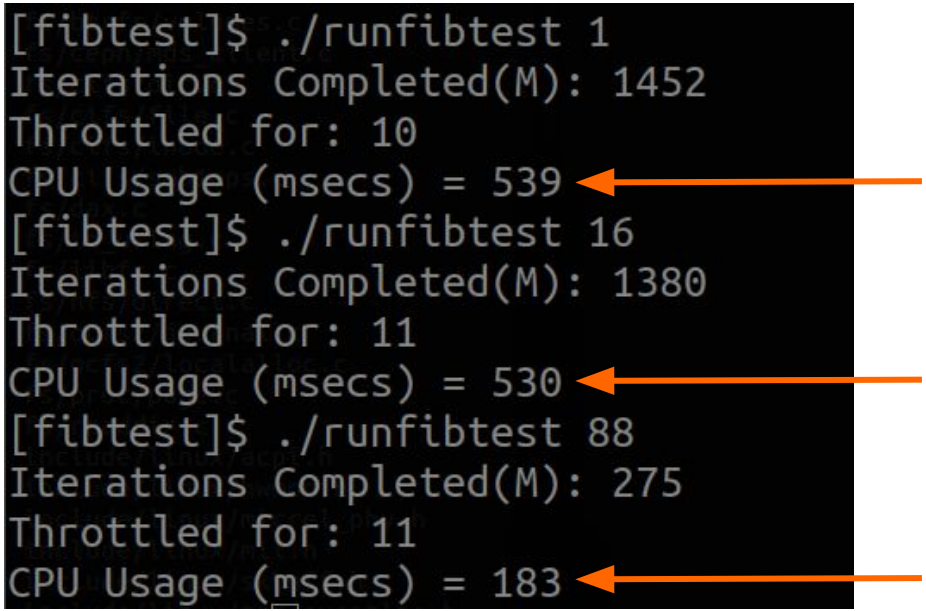


Running Fibtest

```
[fibtest]$ ./runfibtest 1
Iterations Completed(M): 1452
Throttled for: 10
CPU Usage (msecs) = 539
[fibtest]$ ./runfibtest 16
Iterations Completed(M): 1380
Throttled for: 11
CPU Usage (msecs) = 530
[fibtest]$ ./runfibtest 88
Iterations Completed(M): 275
Throttled for: 11
CPU Usage (msecs) = 183
```

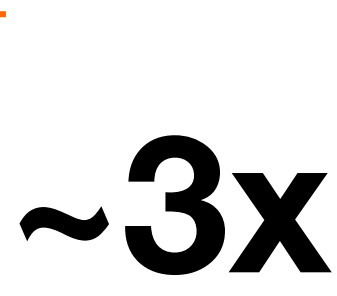


Running Fibtest



Running Fibtest

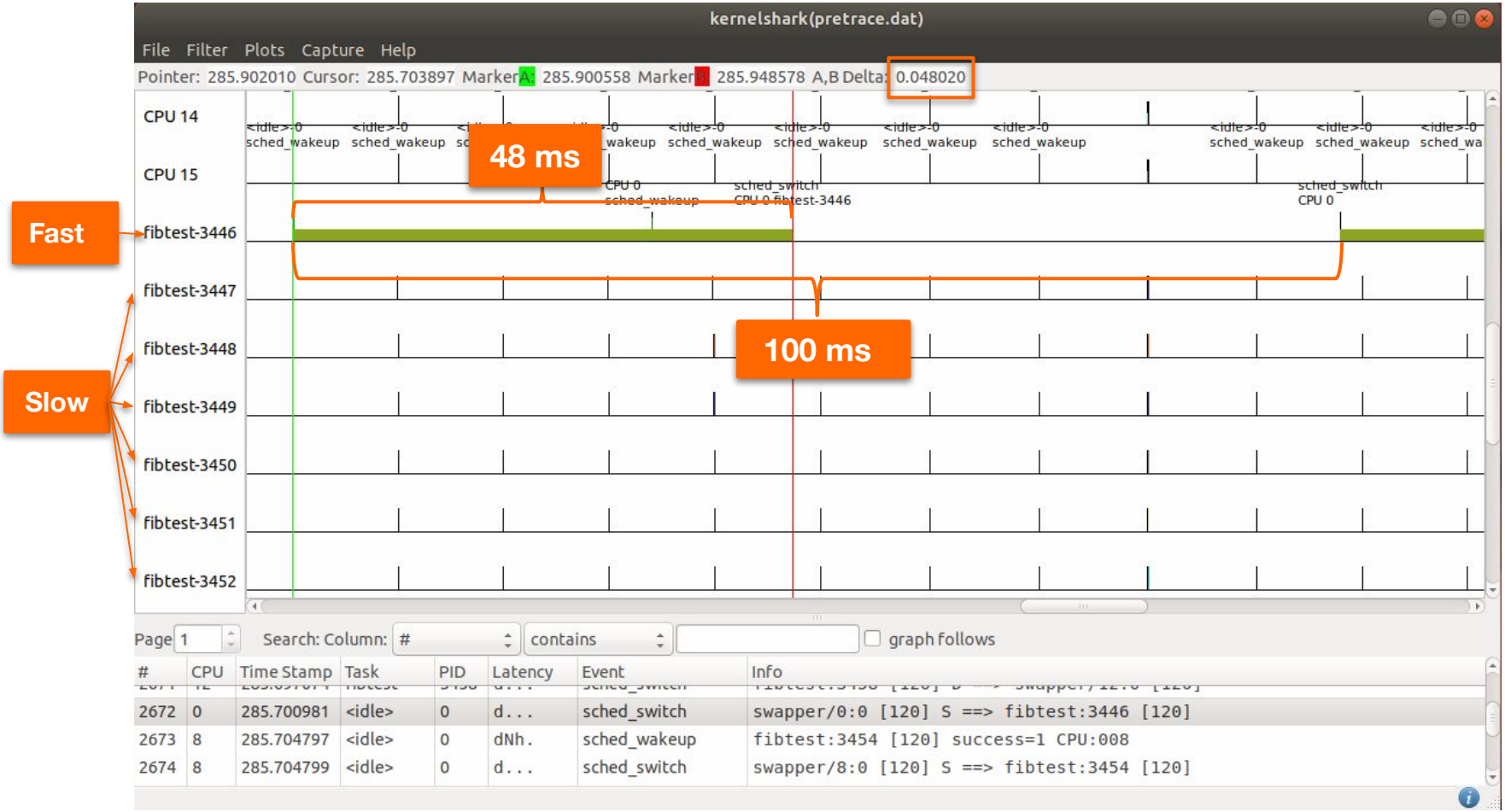
```
[fibtest]$ ./runfibtest 1
Iterations Completed(M): 1452
Throttled for: 10
CPU Usage (msecs) = 539
[fibtest]$ ./runfibtest 16
Iterations Completed(M): 1380
Throttled for: 11
CPU Usage (msecs) = 530
[fibtest]$ ./runfibtest 88
Iterations Completed(M): 275
Throttled for: 11
CPU Usage (msecs) = 183
```



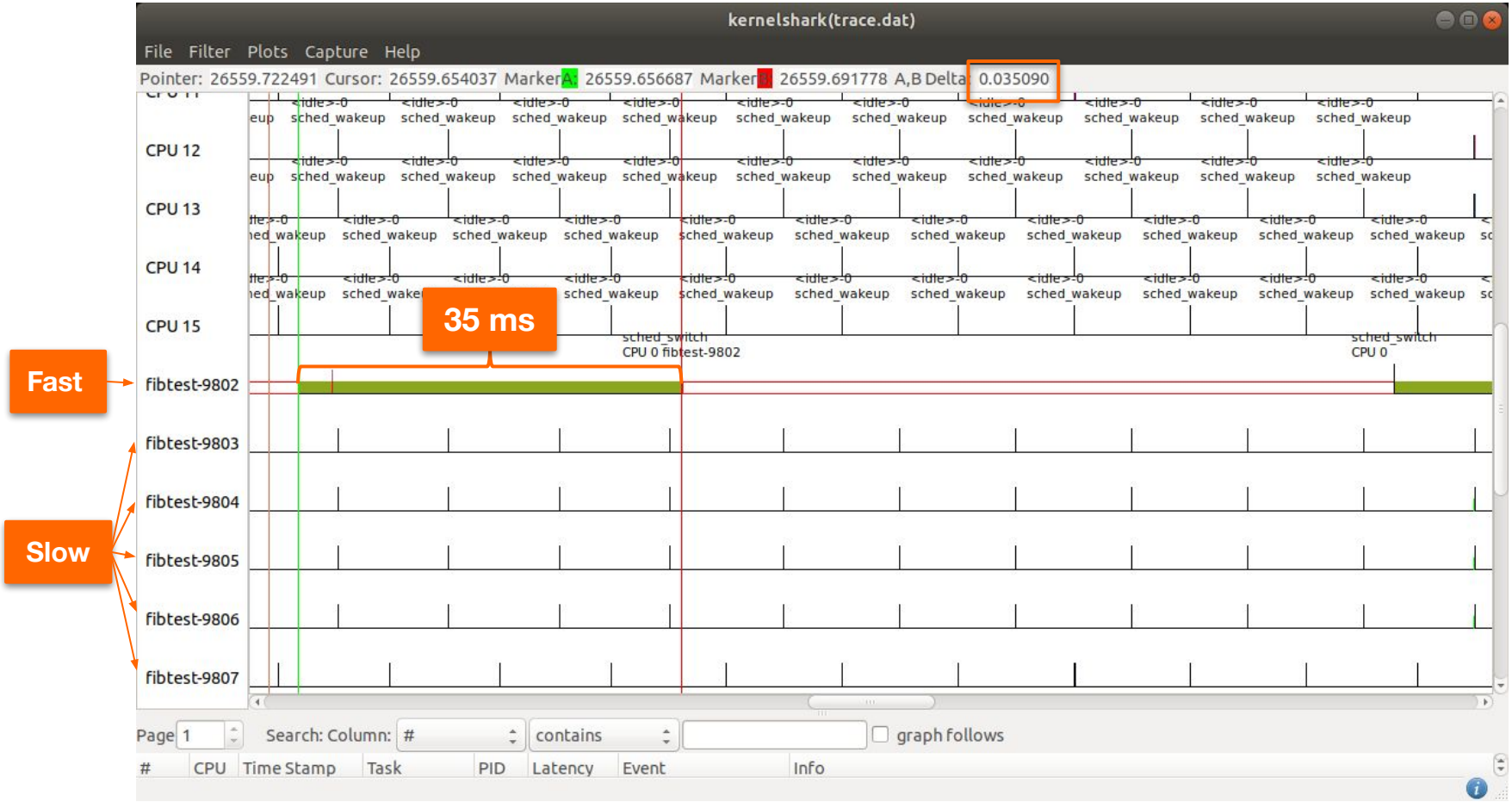
Fibtest Tracing

```
$ sudo trace-cmd record -e 'sched_wakeup*' -e sched_switch -F -c ./runfibtest 16
```

ftrace/kernelshark of Fibtest on 3.16-4.17



ftrace/kernelshark of Fibtest on 4.18-5.3.8

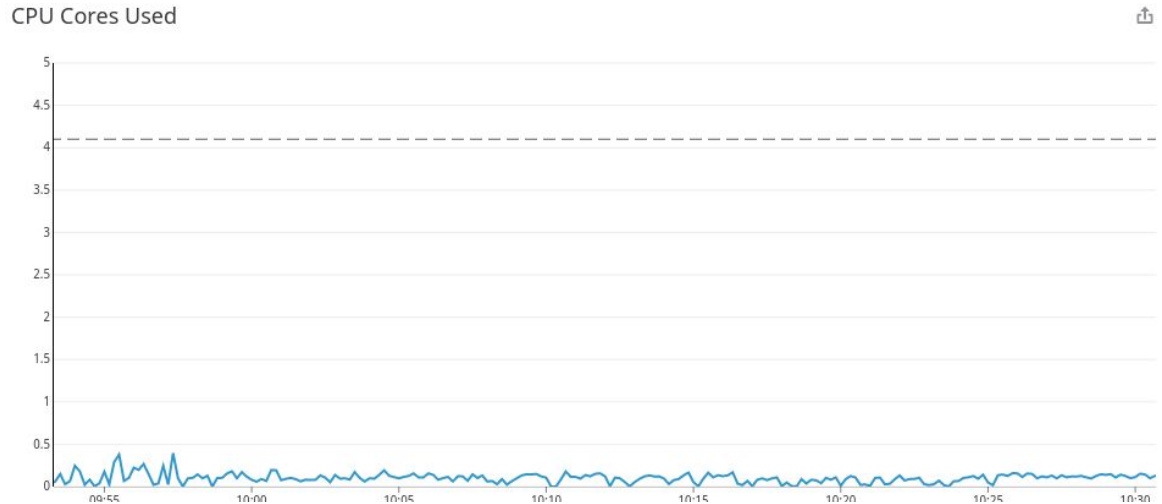


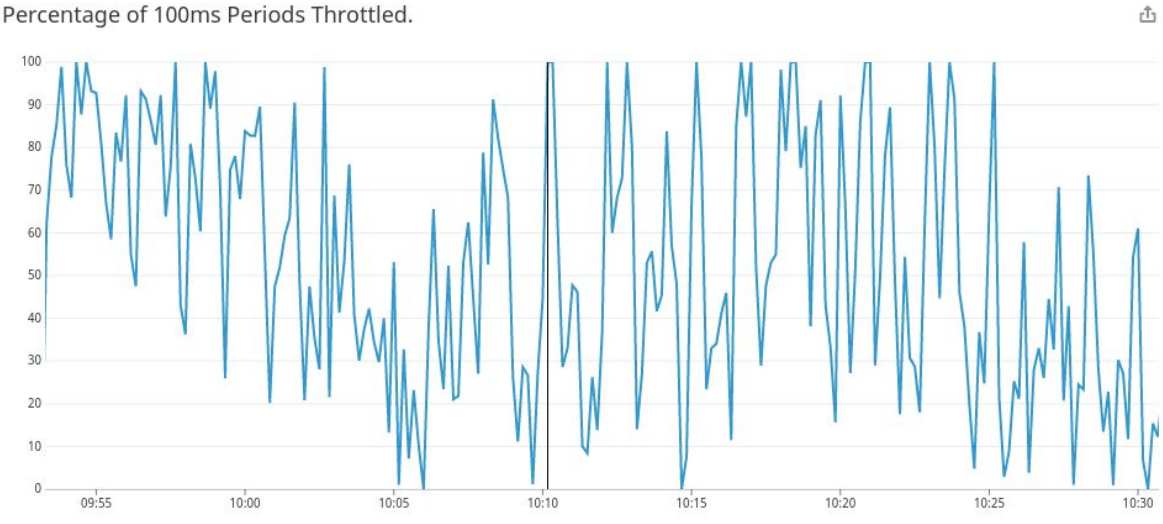
The Causal Commit

512ac999

➔ **Fix for inadvertent throttling due to clock-drift**

512ac999
Clock-drift problem





512ac999

➔ Fix for inadvertent throttling due to clock-drift

➔ **Fixed per-cpu quota to expire on period boundaries**

Real World vs. Conceptual Model

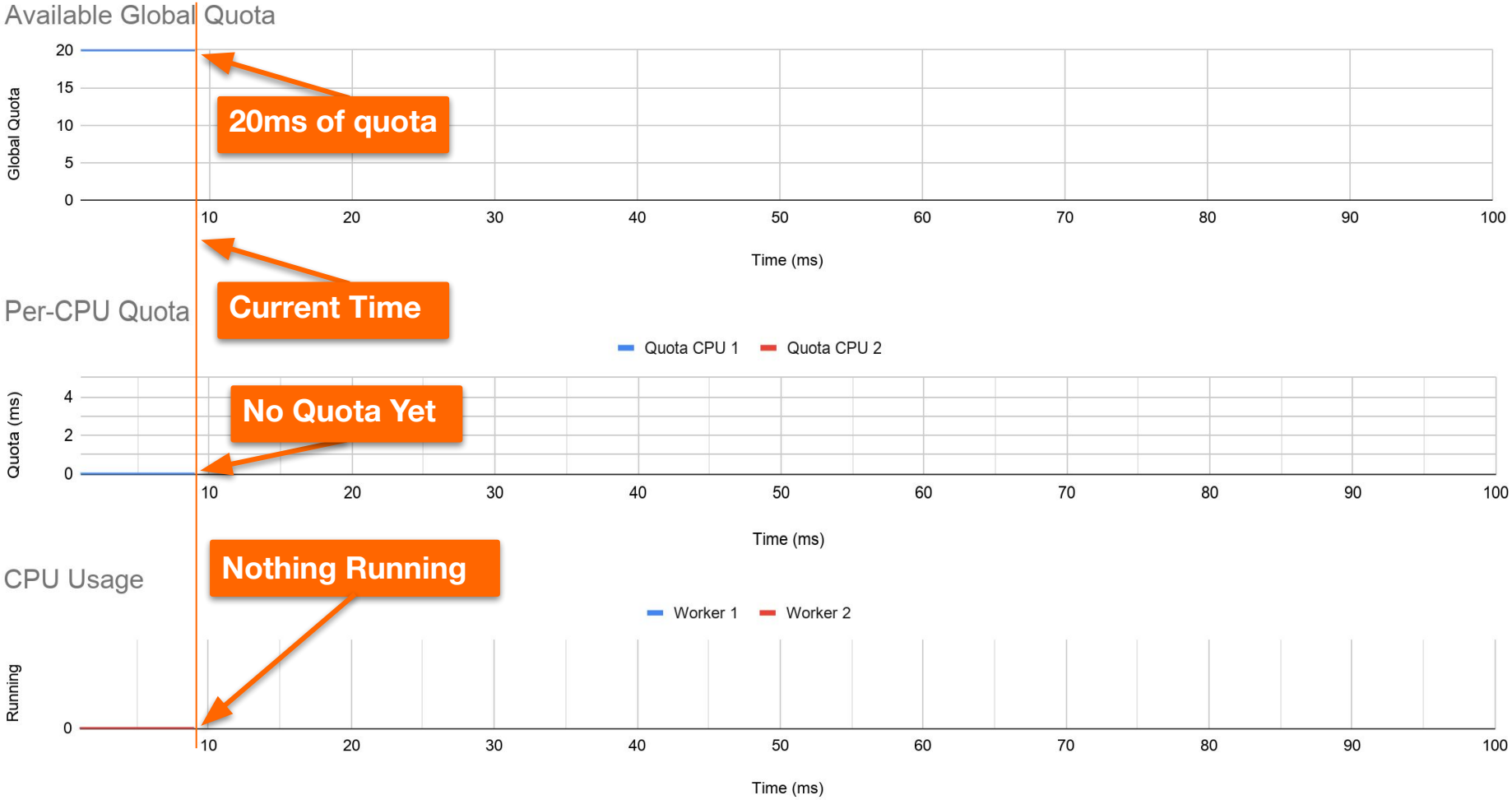
- **Multiple CPUs**
- **Many threads *(sometimes thousands)**
- **Cores run at different speeds - Use performance mode**
- **Schedulers are hard**

  Quota is split into 5ms slices and assigned to individual CPUs

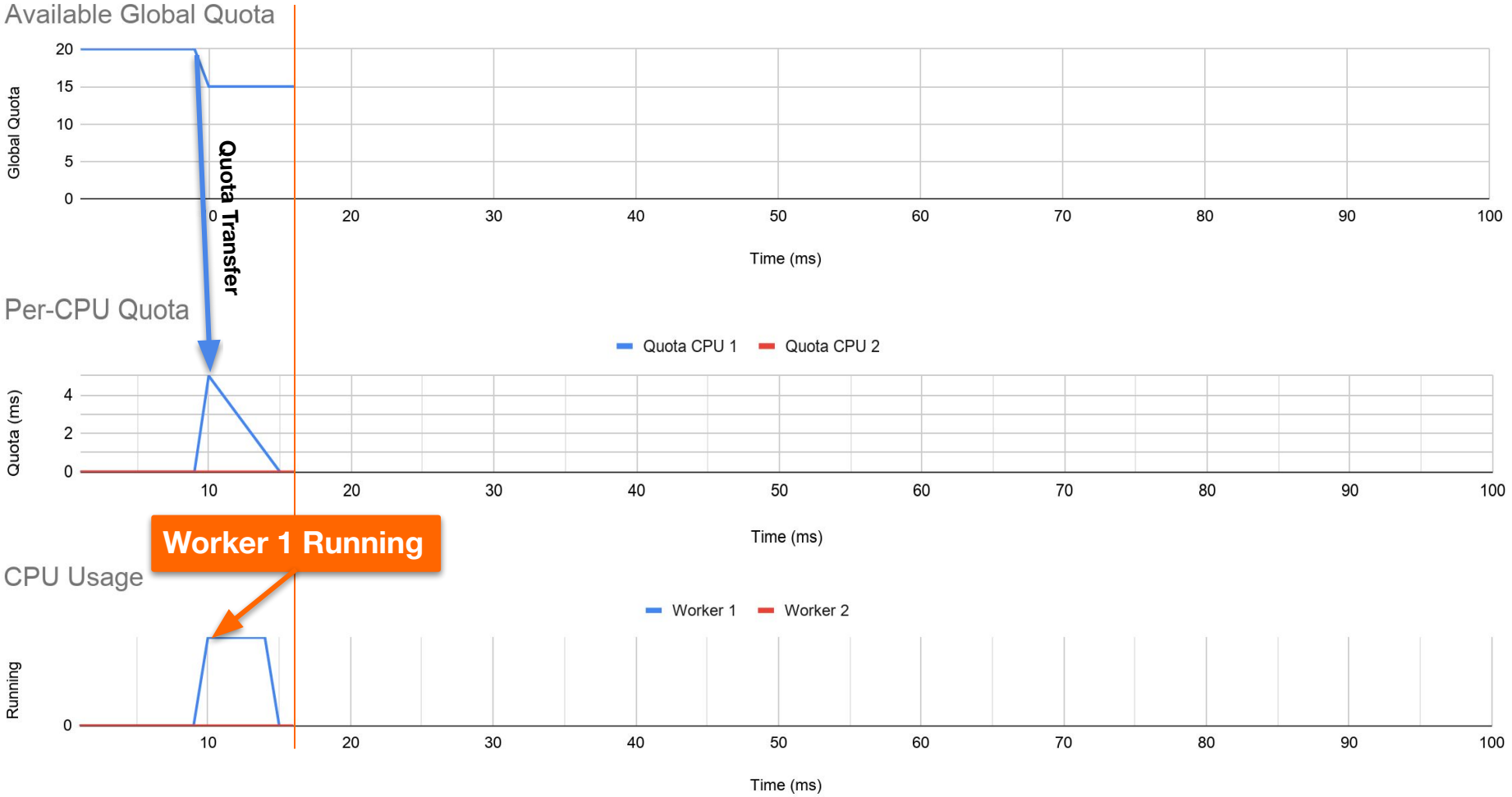
  (1 CPU of quota = 100ms/period = 20 slices/period) = not enough for large machines

  Per-cpu quota will expire if not used within a period

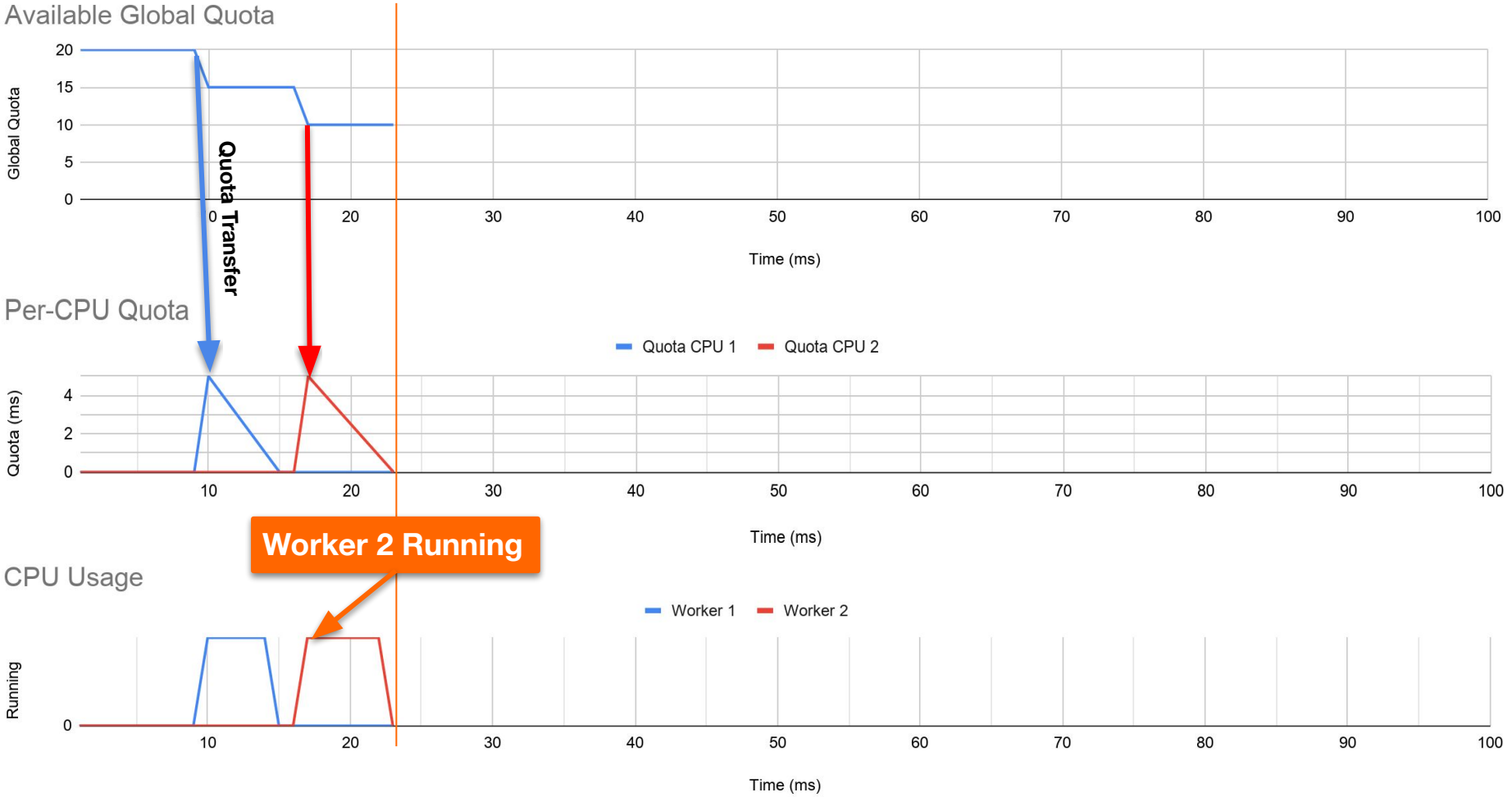
Real World Example



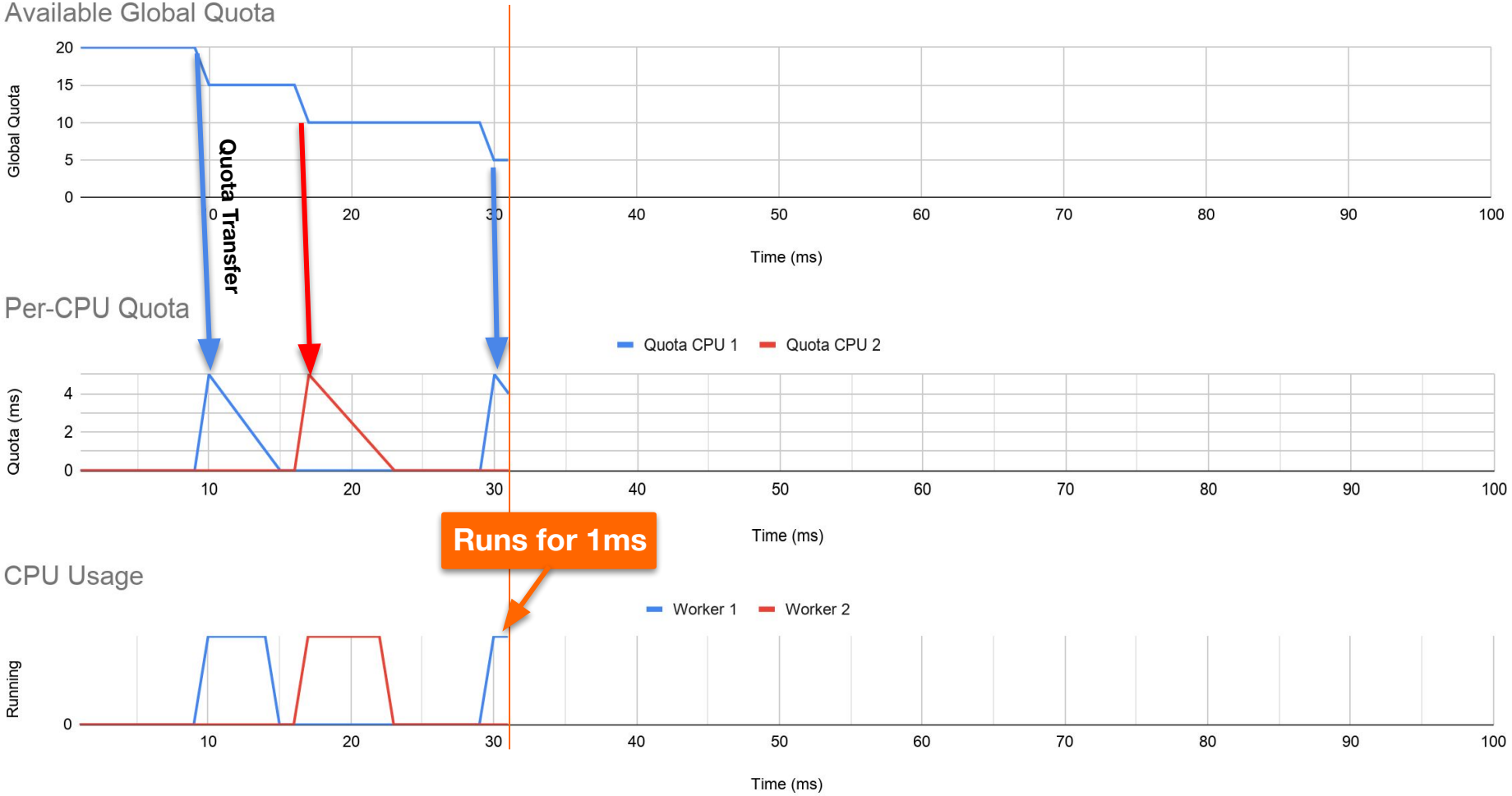
Real World Example



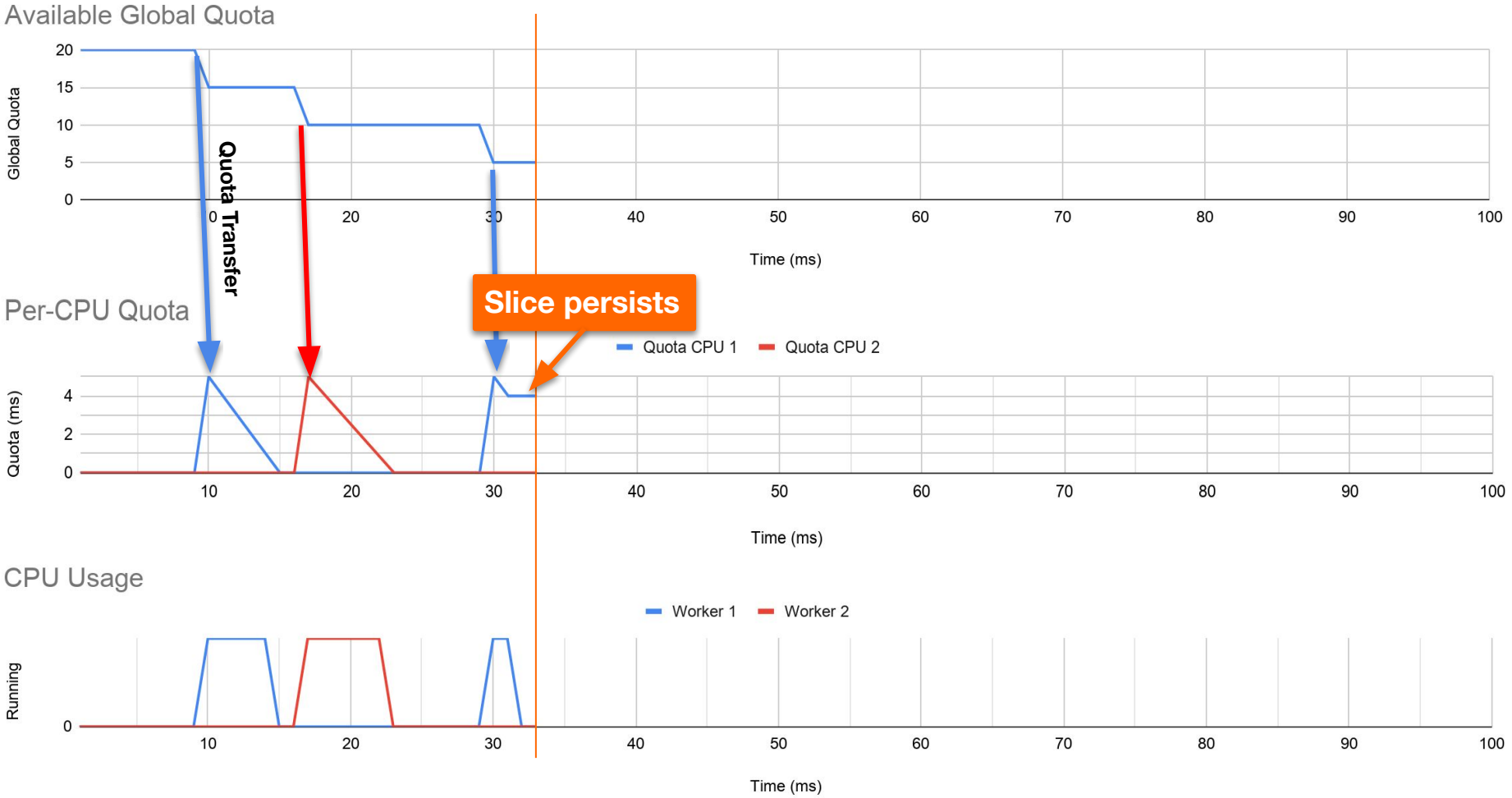
Real World Example



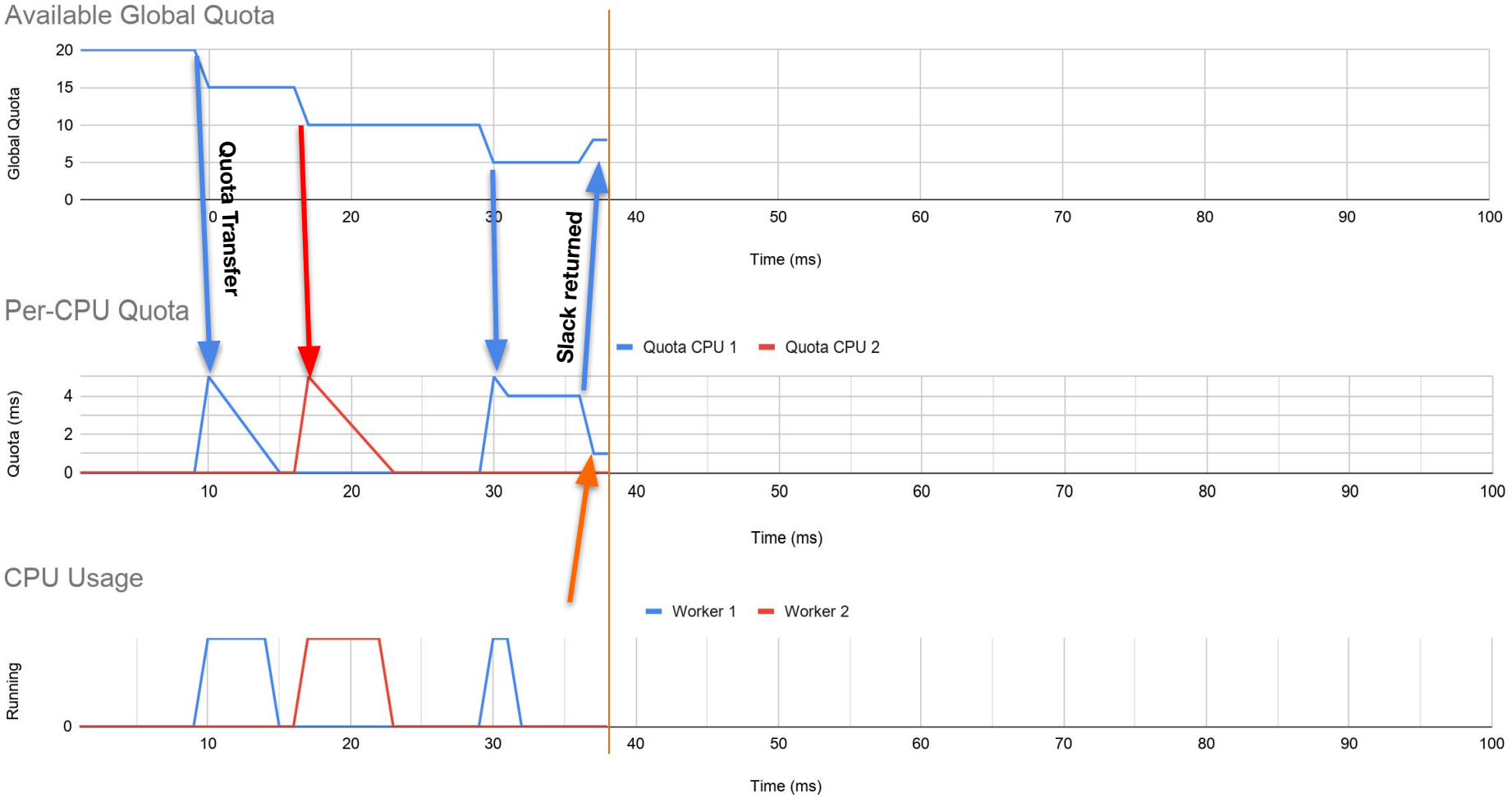
Real World Example



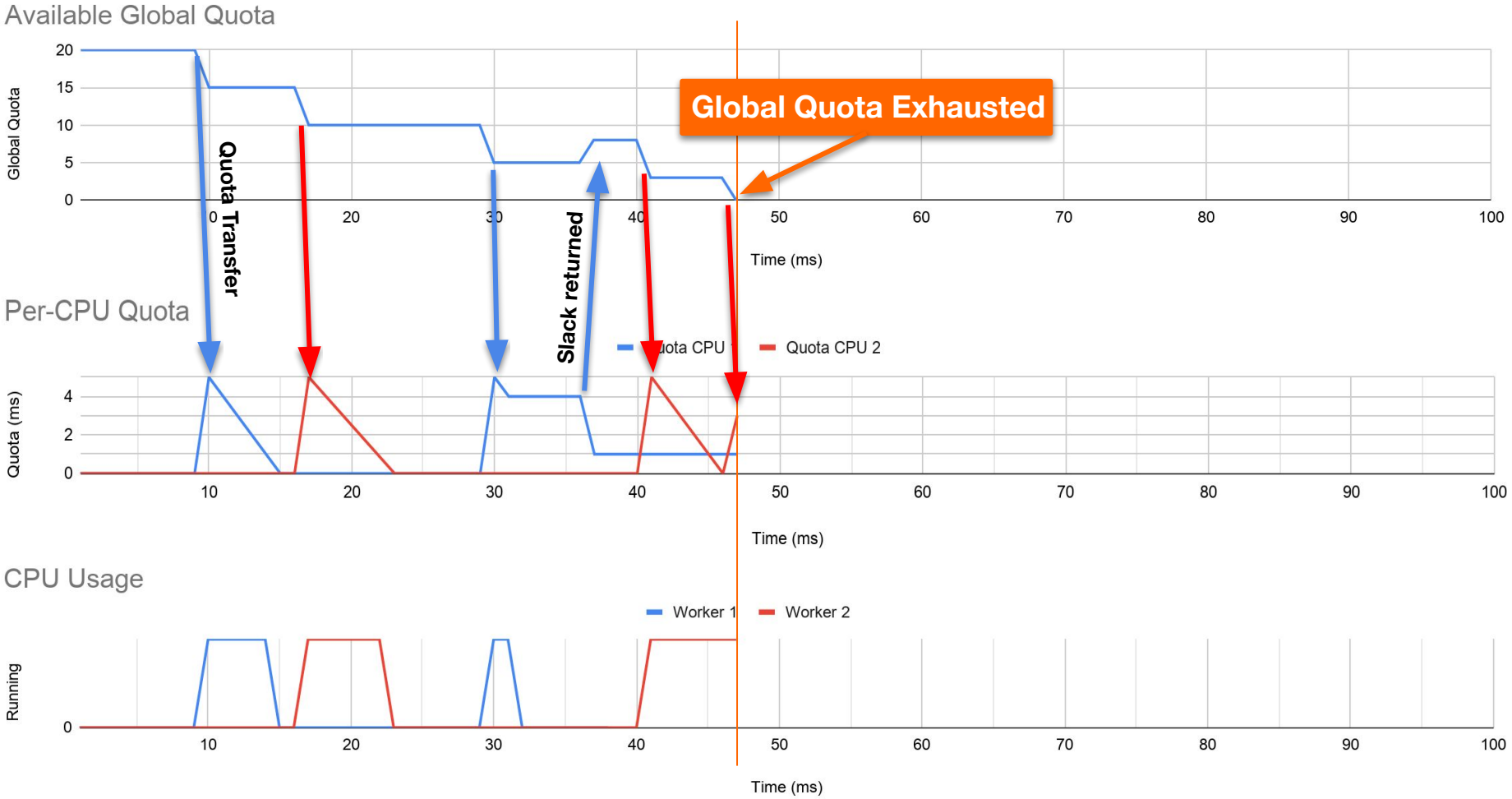
Real World Example



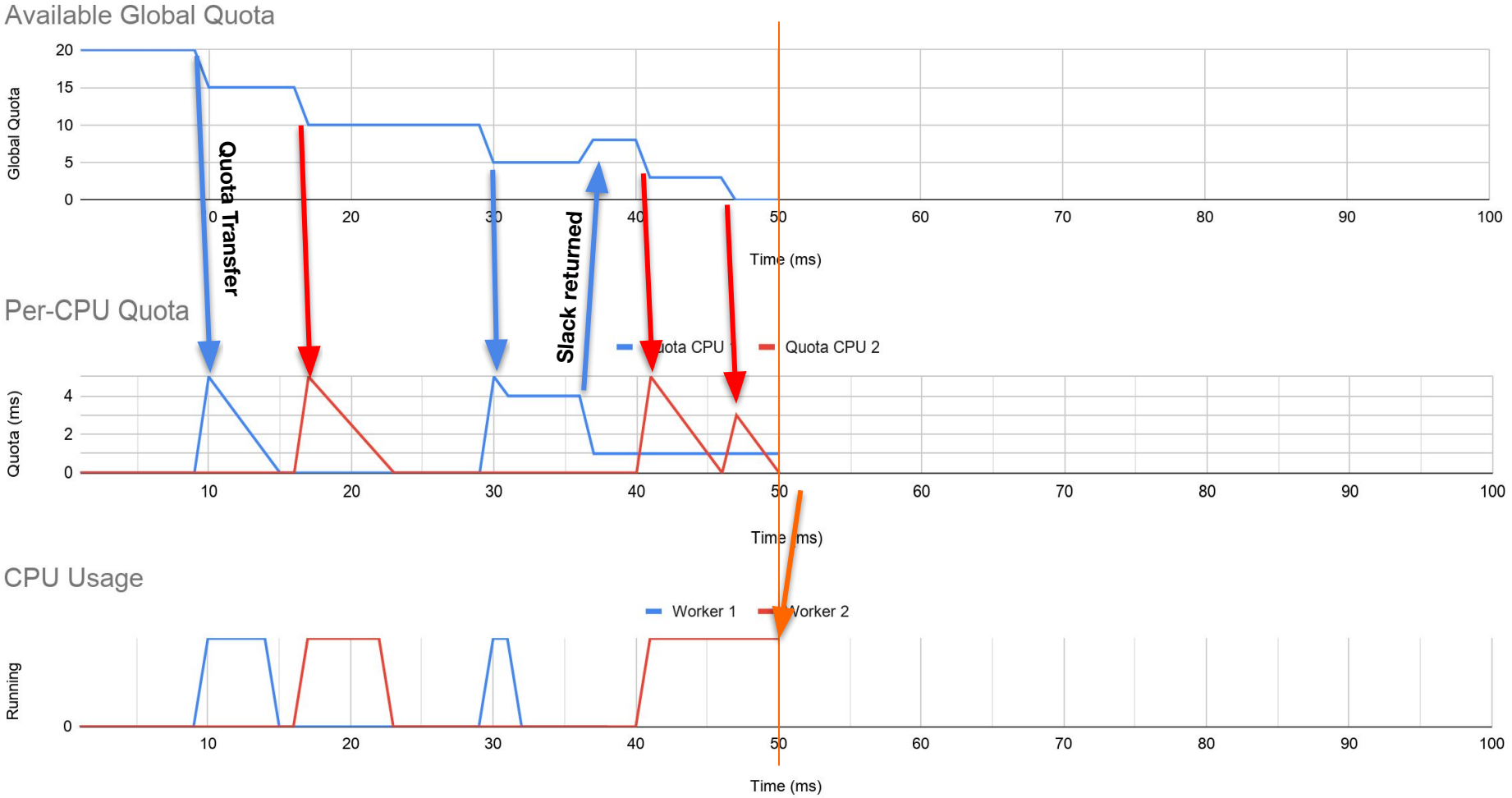
Real World Example



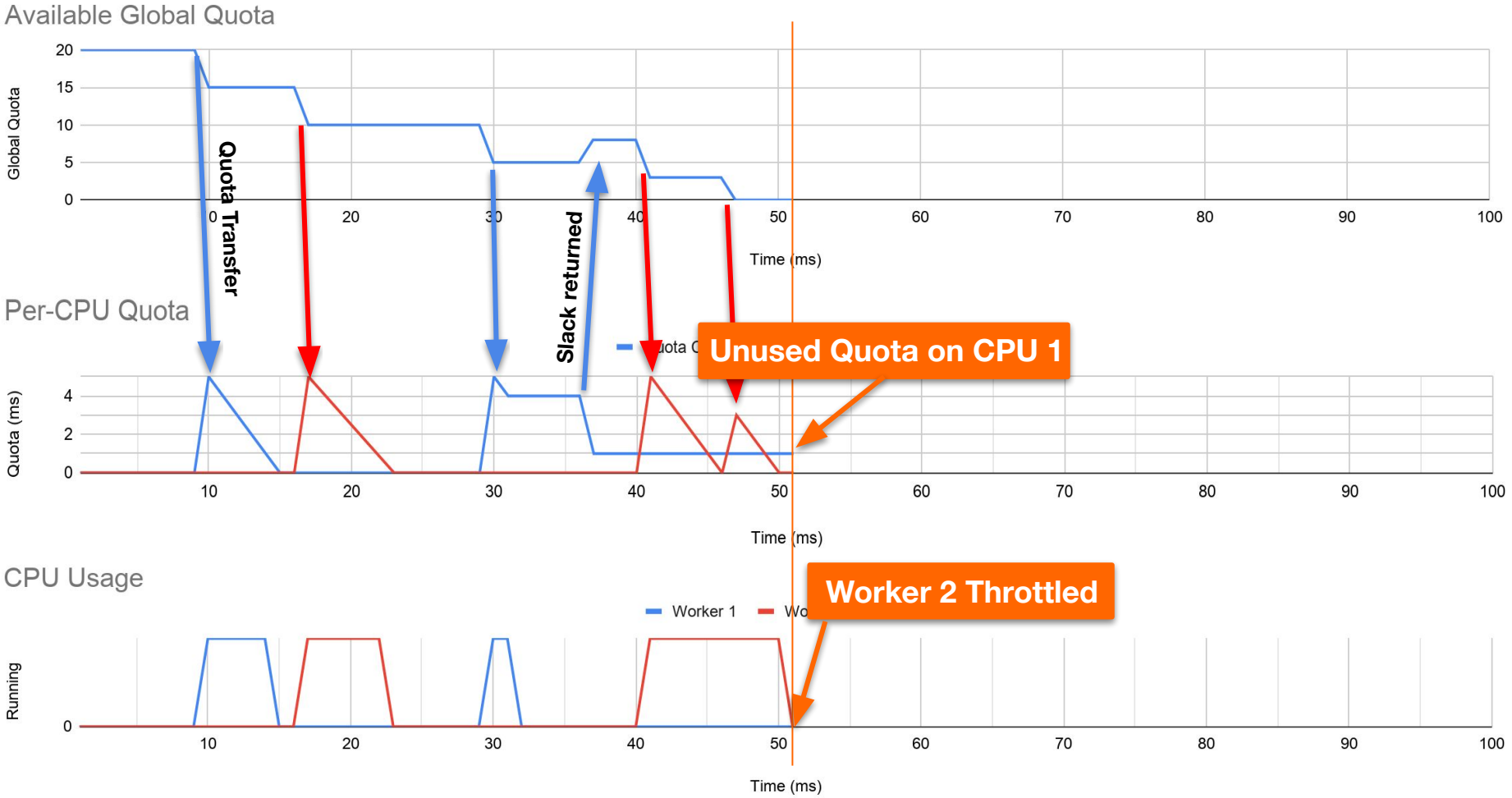
Real World Example



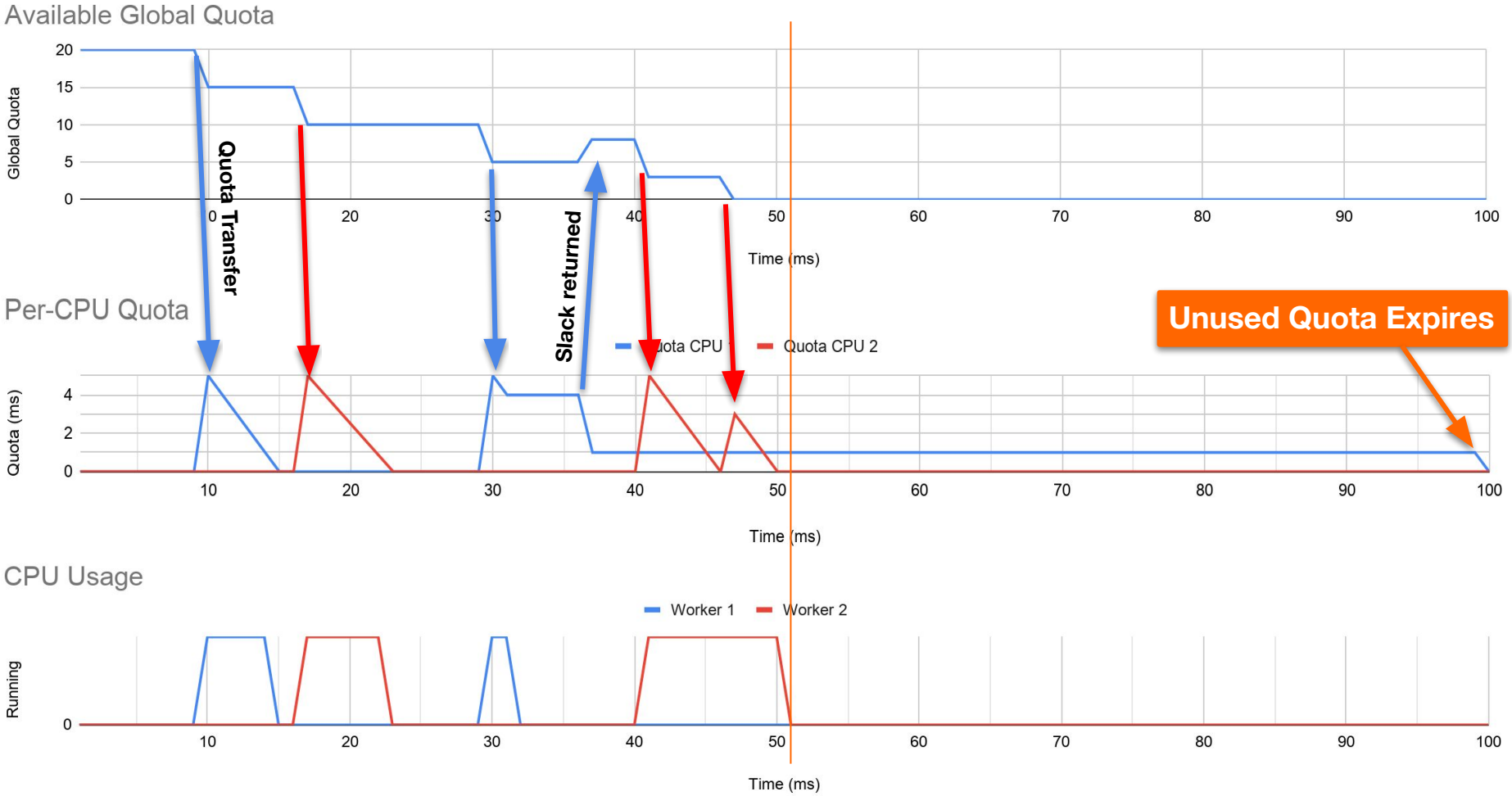
Real World Example



Real World Example



Real World Example



IMPACT

1ms/100ms * (88 CPUs-1) = 87ms/100ms

= 870m = .87 CPU

Roadmap

CPU Limit Basics

The Problem

Reproducing the Problem

Solution and Workarounds

Possible Solutions

- Remove 512ac999
- Burst bank / Rollover minutes
- Remove all per-CPU expiration logic

The Solution

Remove all per-CPU expiration logic

- ➔ **5 months of debate**
- ➔ **6 patch iterations**

The Solution

Commits: de53fd7aedb1 & 763a9ec06c40

- ➔ **Applied to 5.4 Kernel**
- ➔ **linux-stable**
  - ◆ **4.14.154+, 4.19.84+, 5.3.9+**
- ➔ **Distro kernels**
  - ◆ **Ubuntu 5.3.0-24+**
  - ◆ **Ubuntu 4.15.0-67+**
  - ◆ **RHEL7 - kernel-3.10.0-1062.8.1.el7**
  - ◆ **RHEL8.2 - WIP**

The Solution

Kernel 5.3.7

```
[fibtest]$ ./runfibtest 88
Iterations Completed(M): 213
Throttled for: 11
CPU Usage (msecs) = 137
```

Kernel 5.3.9

```
[fibtest]$ ./runfibtest 88
Iterations Completed(M): 1316
Throttled for: 10
CPU Usage (msecs) = 482
```

3x

Takeaways

- Monitor your throttled %
- Upgrade your Kernels
- Use whole cpu quotas
- Increase quota where necessary

Questions?

Dave Chiluk
Linux Platform Software Engineer

@dchiluk

Other Developments:

More developments:

Setting CFS Period (GH #51135)

WIP:

Unset CFS quota with CPU sets (GH #70585) (GH #75682)

Things I’d Like to see

Kernel: C-state aware quotas

Kernel: Burstable Cgroup CPU Limits

Kubernetes: Pod level Resource constraints

Kubernetes: Node Level CPU Overcommit