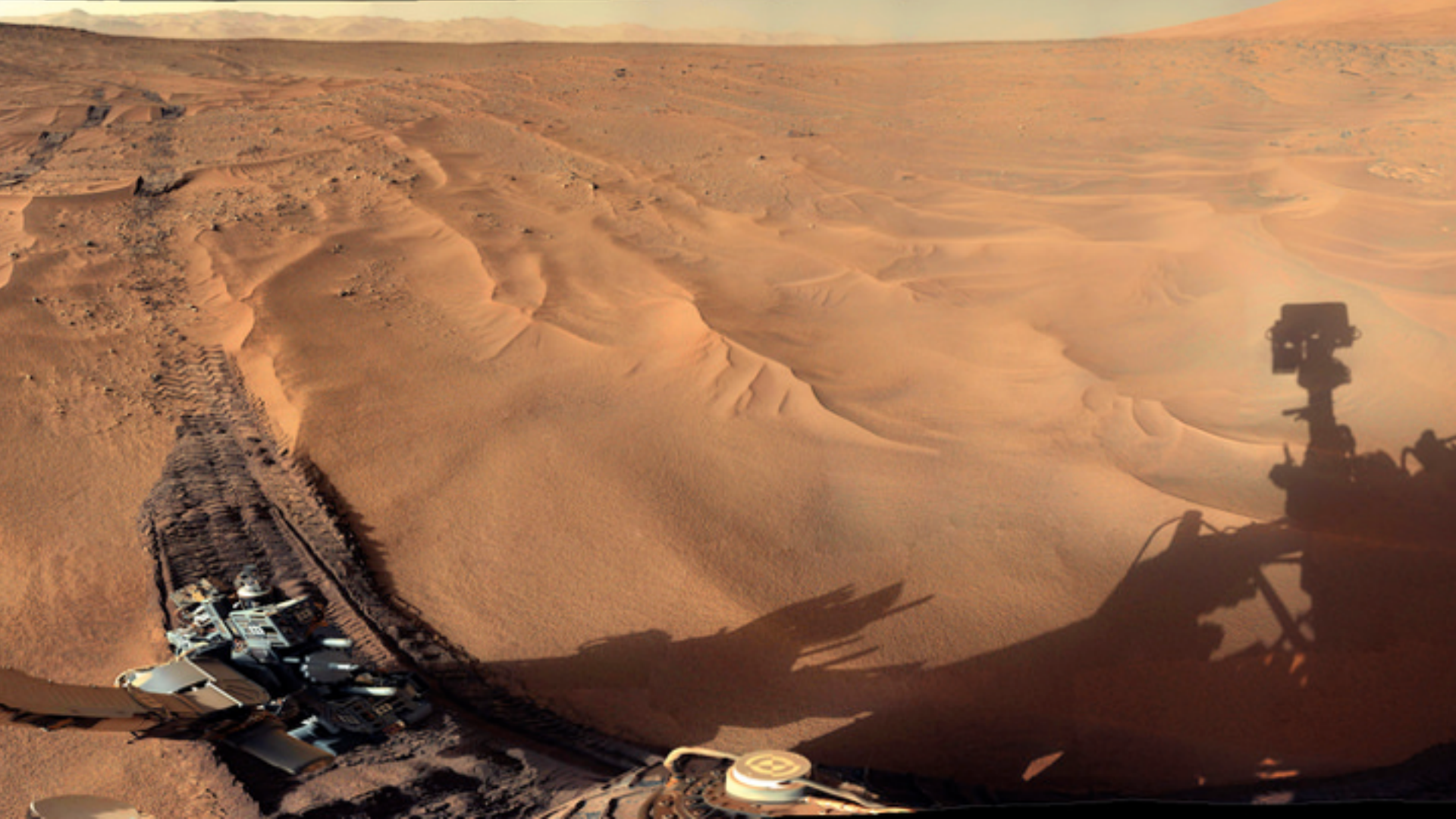
Benchmarking Elasticsearch with Rally

Daniel Mitterdorfer
@dmitterd

“Elasticsearch is just a search engine, isn't it?”

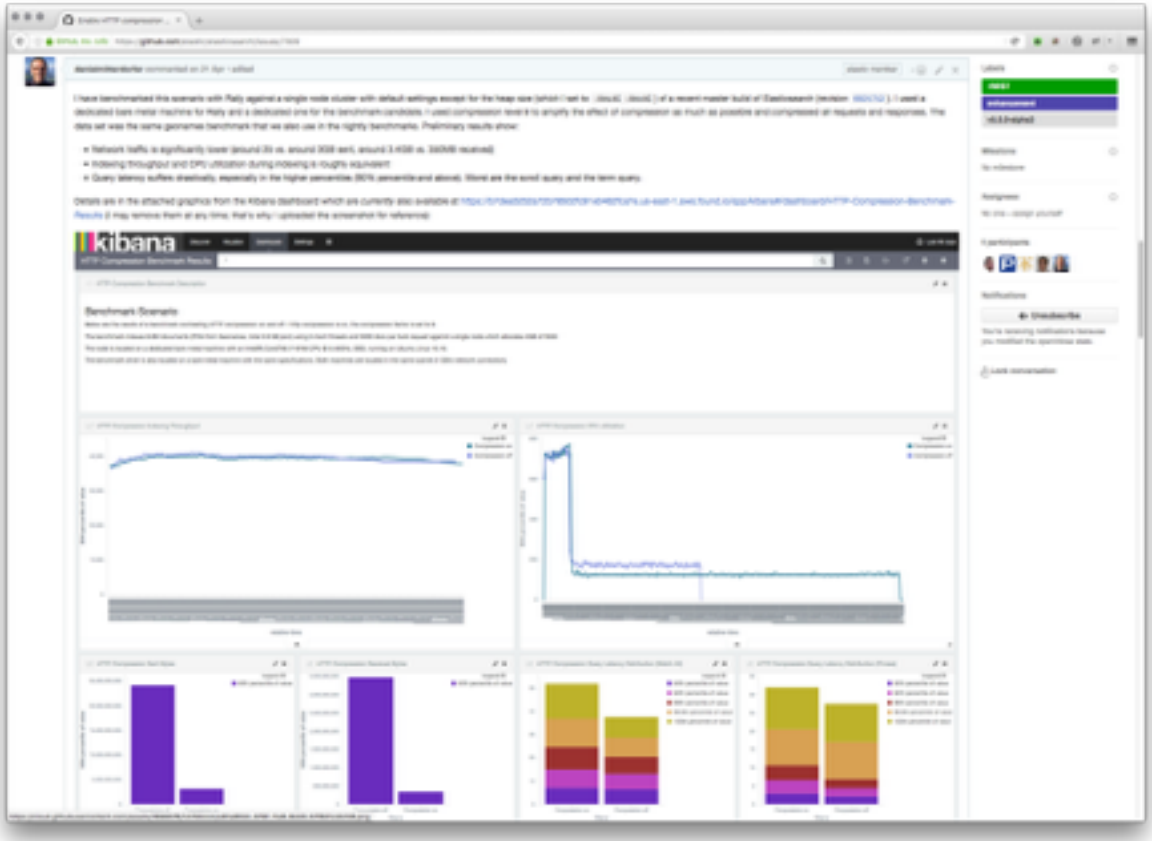

skyblueseed



How do you evaluate performance for all these use-cases?

What we do: Measure, Measure, Measure

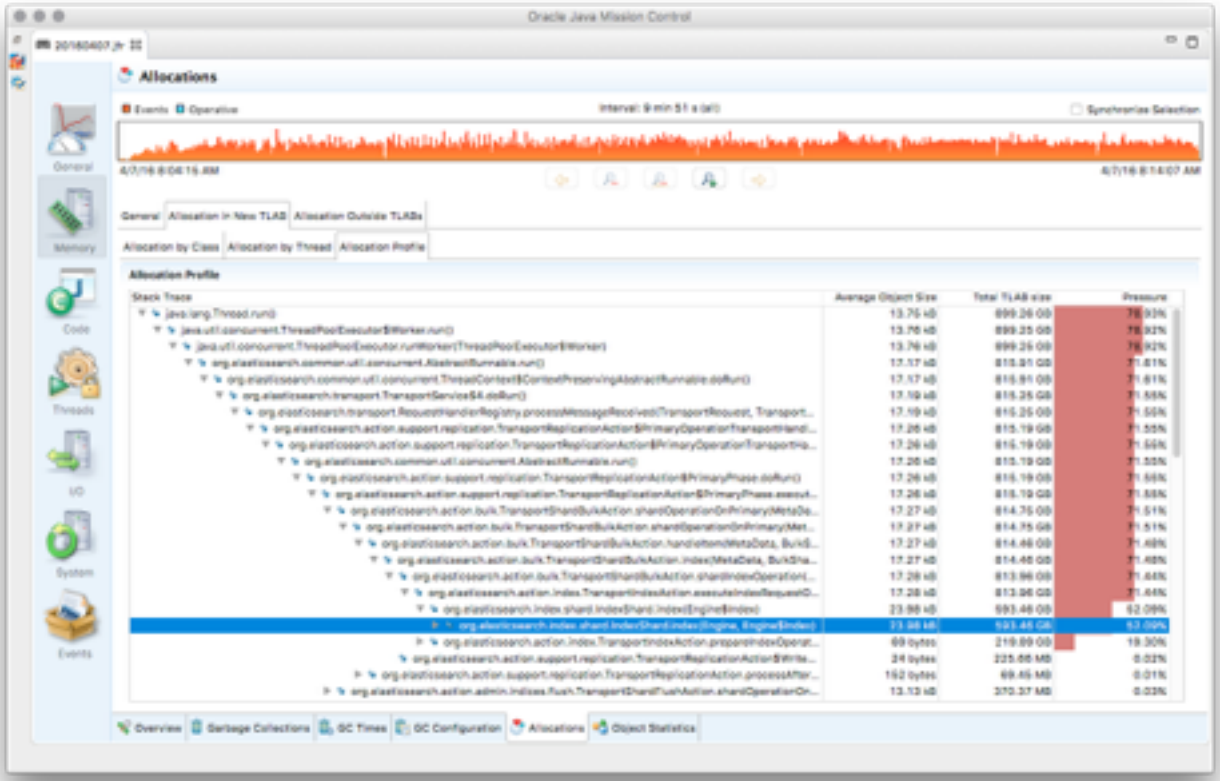
During Development



elastic

What we do: Measure, Measure, Measure

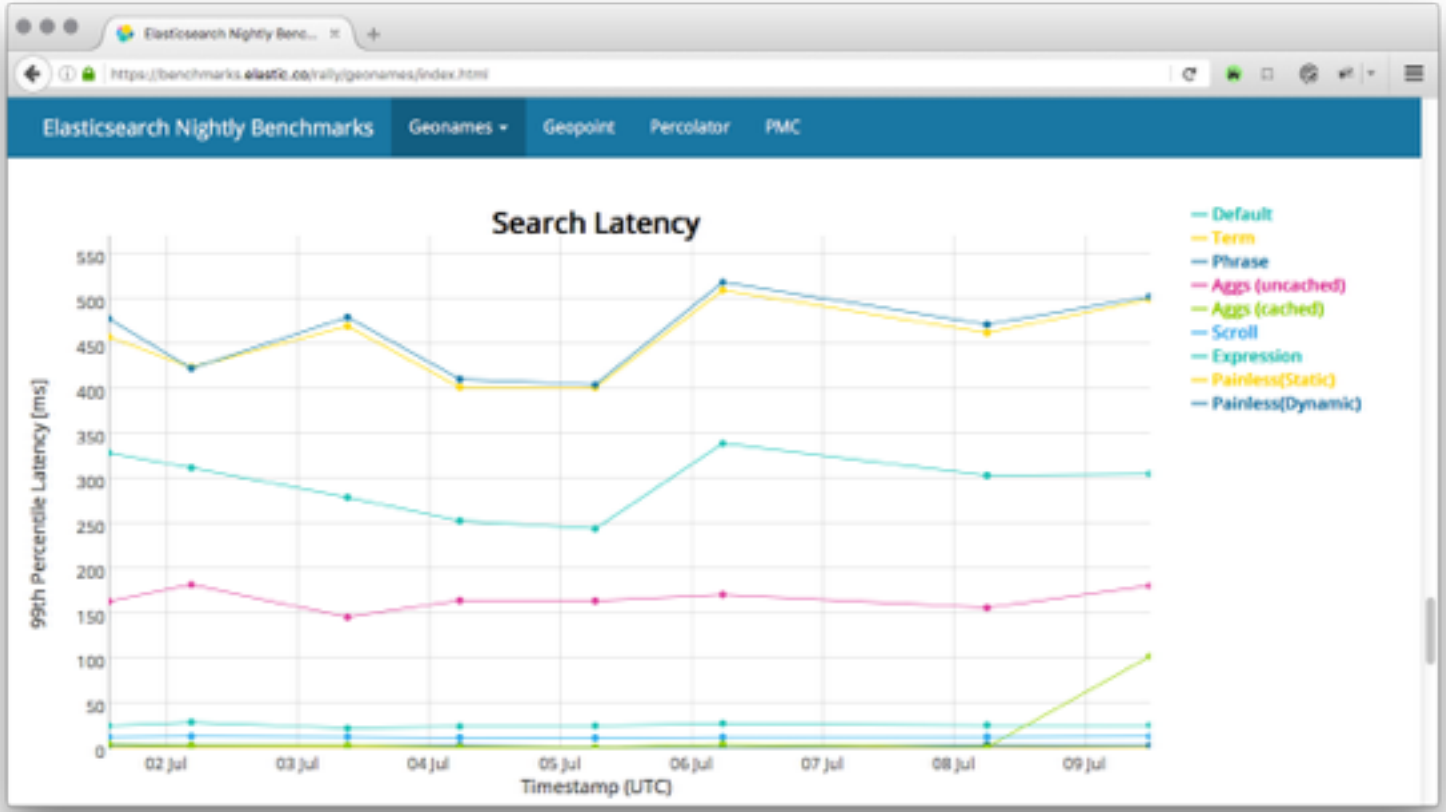
During Development



elastic

What we do: Measure, Measure, Measure

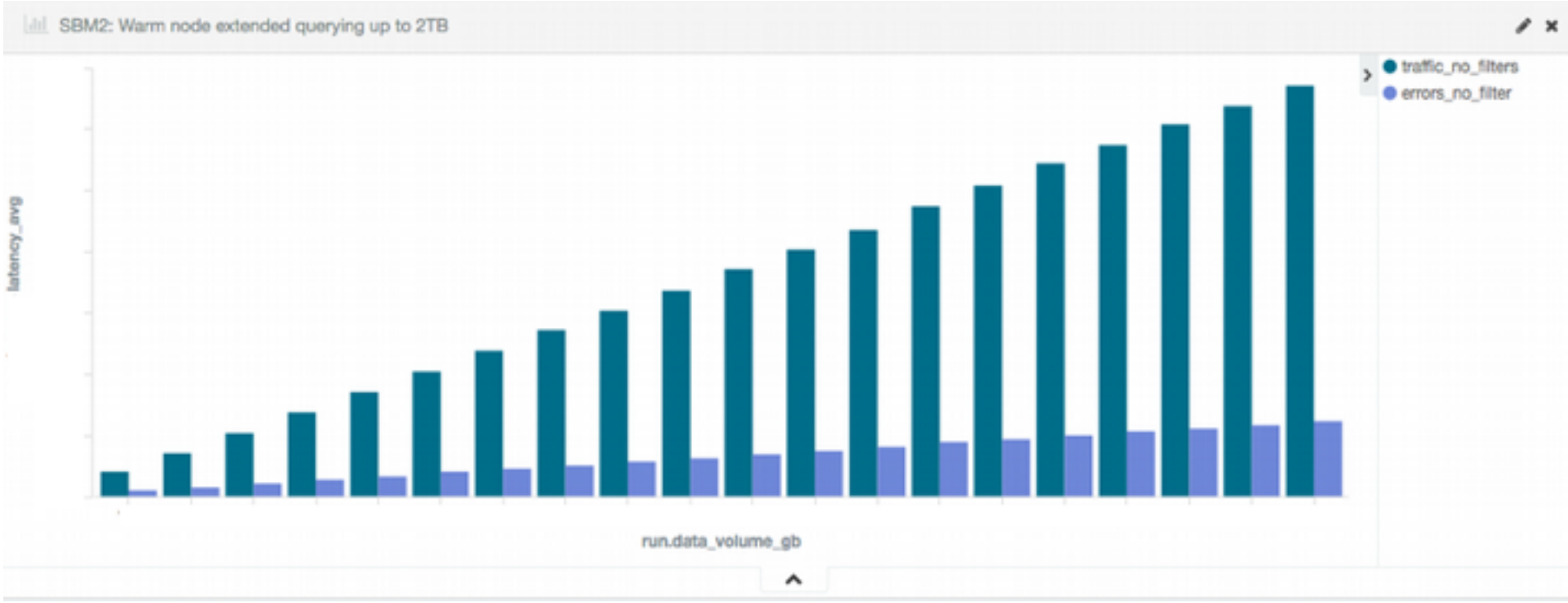
Nightly benchmarks



elastic

What we do: Measure, Measure, Measure

Sizing benchmarks for specific scenarios*

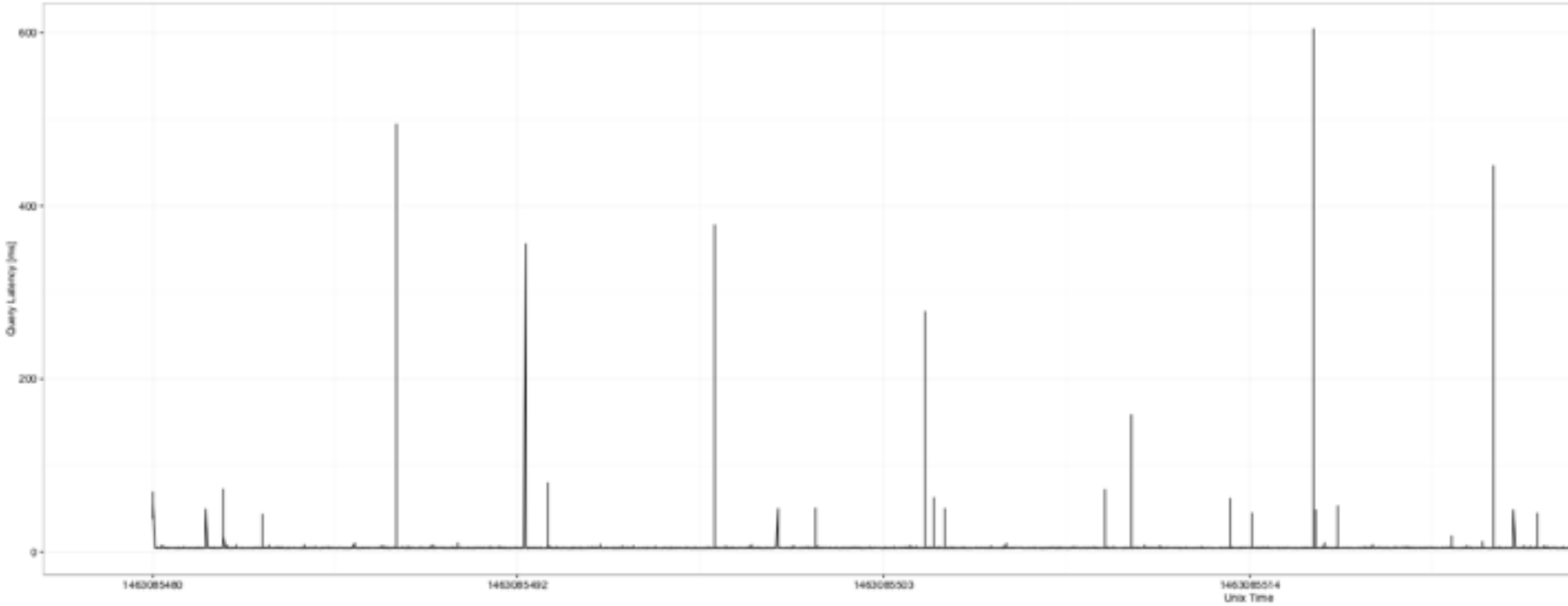


 *) numbers on axis intentionally stripped to avoid interpretation out of context

elastic

What we do: Measure, Measure, Measure

Performance measurement / tuning at customer site



elastic

How we measure

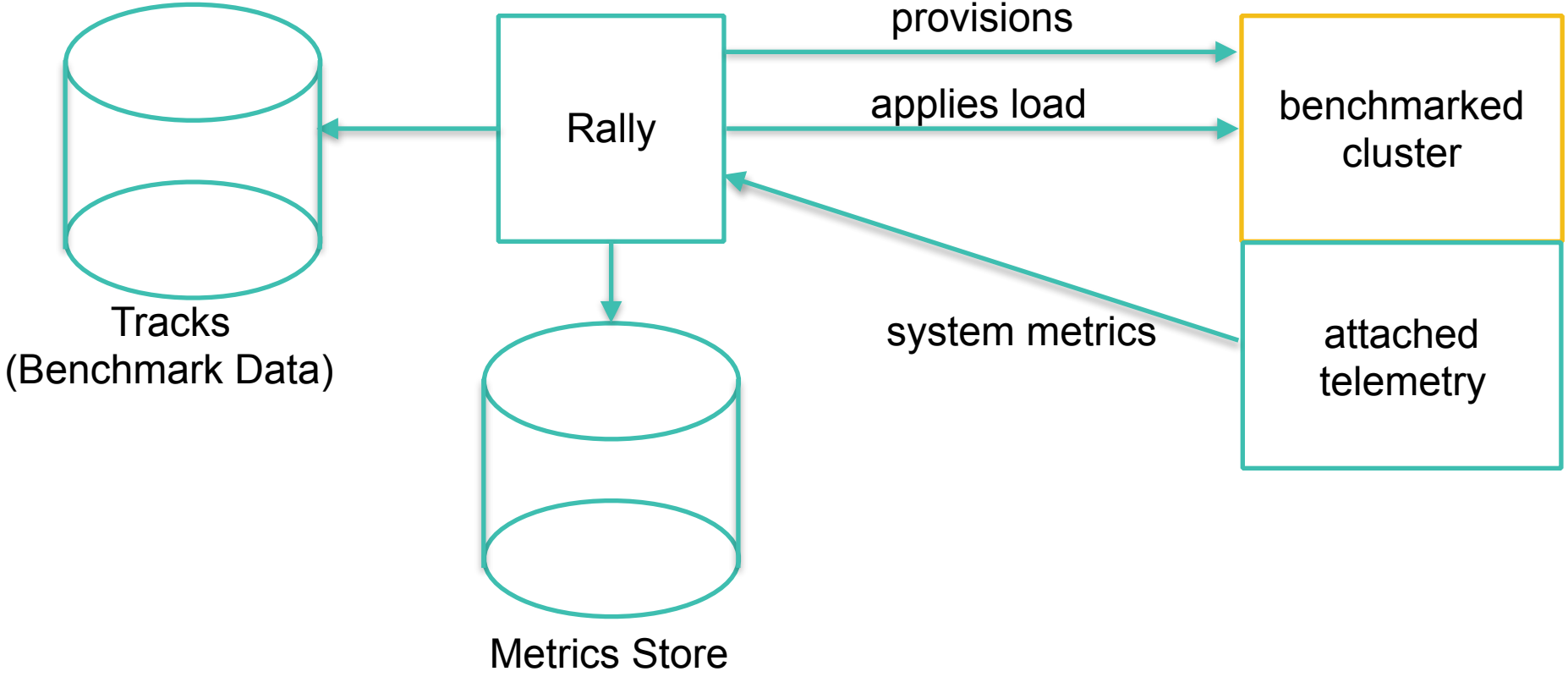

OJB 93B
12

Rally

You know … for benchmarking Elasticsearch

https://github.com/elastic/rally

10.000 feet view of Rally



elastic

Demo

7 Deadly Benchmark Sins

Sin #1: Not paying attention to system setup

Hardware

- Bare-metal
- SSDs
- Server-class CPU
- Single socket, multi socket?
- Enough memory head-room for FS cache

Sin #1: Not paying attention to system setup

Operating System and JVM

- Linux, Windows
- No Swap
- Check network configuration
- Think about file system, LVM, etc.
- I/O scheduler: cfq, noop, deadline
- CPU governor: powersave, performance
- JVM version

Sin #1: Not paying attention to system setup

Benchmark Setup

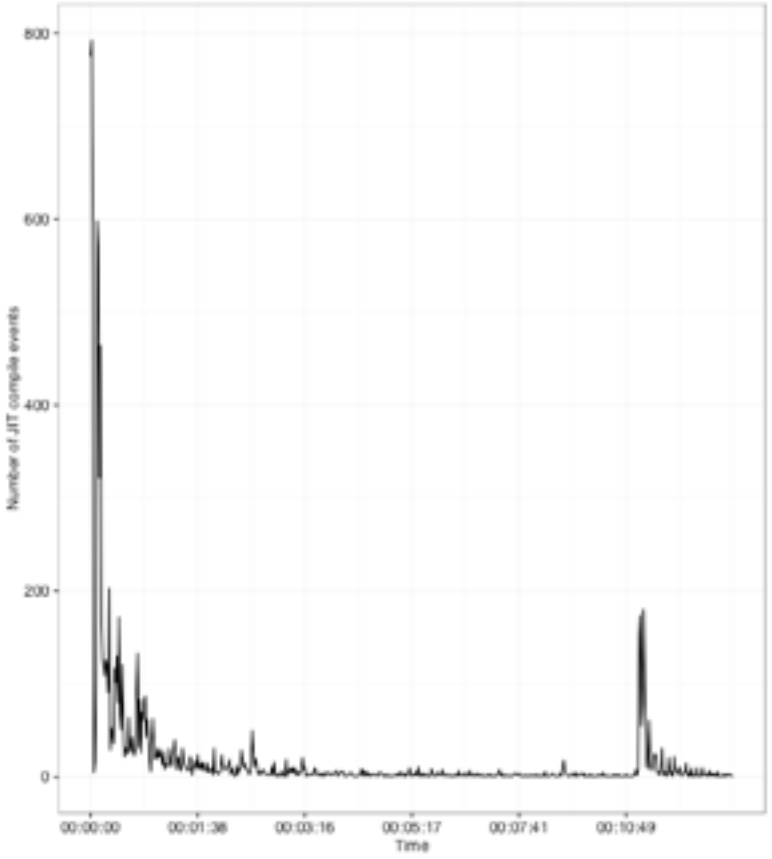
- Beware of unwanted caching effects (FS cache, …)
- Benchmark driver and ES on separate machines
- One node per machine (or adjust JVM parameters (GC threads))
- Low-latency, high-throughput network between benchmark driver and ES
- No traffic on this network

Sin #2: No warmup

Awake before your first coffee? Elasticsearch isn’t either.

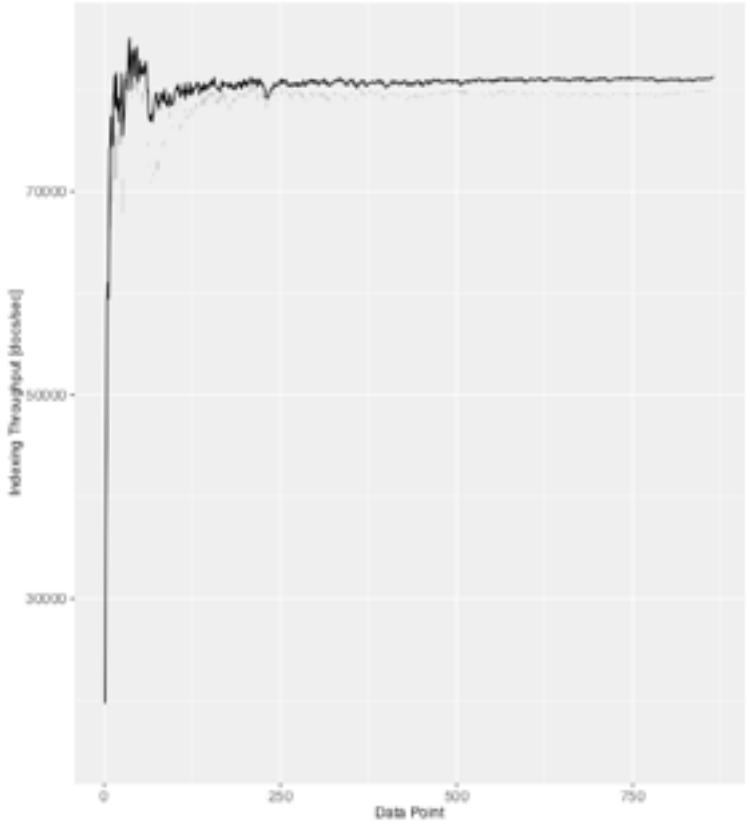
- JIT compiler needs to run first
- Creation of long-living data structures
- FS cache for Lucene segments (memory-mapped IO)
- Benchmark driver needs to reach stable state too

Warmup Behaviour: C2 Compilation Events/s



elastic

Warmup Behaviour: Benchmark Driver Throughput



elastic

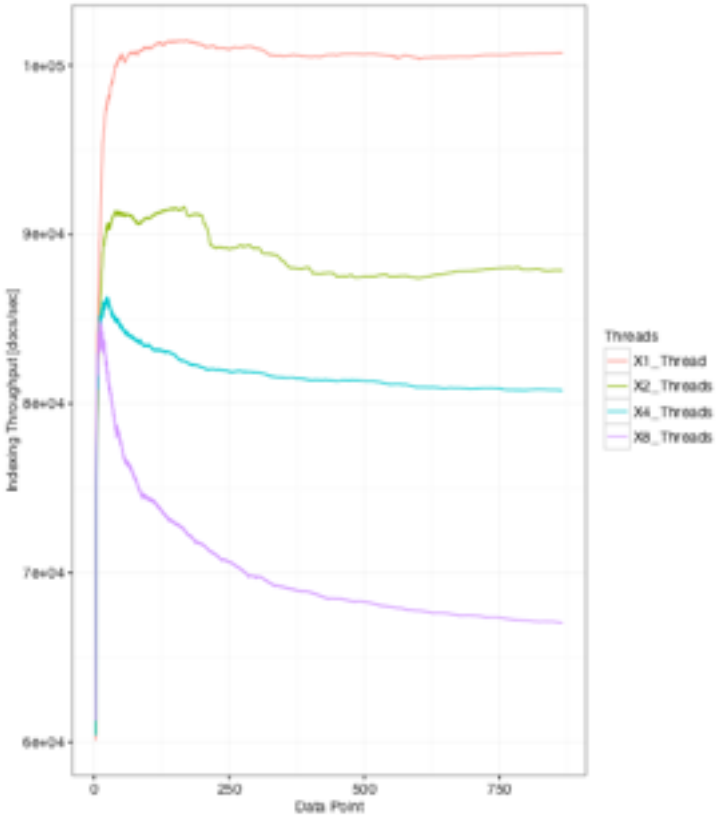
Sin #3: No bottleneck analysis

Are you really benchmarking what you think you're benchmarking?

- Benchmark driver
- System setup: analysis of system background noise (jhiccup)
- Network

First Driver Stress Tests

Contention all over the place



elastic

Sin #4: The divine benchmarking script

"After all, it produces numbers with 6 decimal places!"

- Not paying attention how metrics are gathered
  - `System.currentTimeMillis()` vs. `System.nanoTime()`
- Not checking measurement overhead
- No return code checks: the fast 404
- Blind trust in tools: No cross-verification

Cross-Validation of Metrics

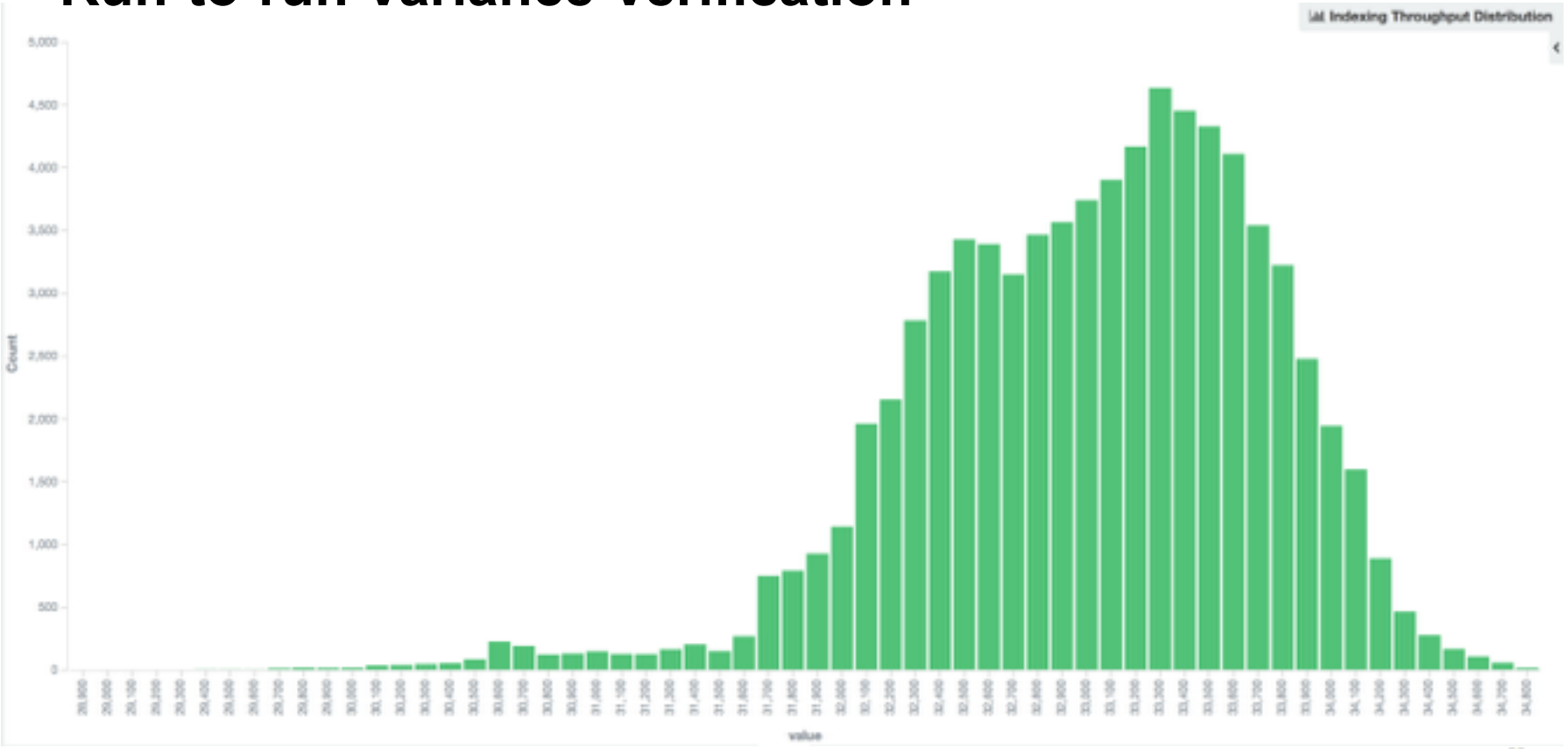
Metric	Rally	Flight Recorder	GC log
Young Gen GC	79,416 ms	**89,003 ms(?)**	80,853 ms
Old Gen GC	23,964 ms	**156,630 ms(?)**	23,989 ms

Sin #5: Denying Statistics

Run-to-run variance

- How is run-to-run variance distributed?
- Multiple trial runs and t-test

Run-to-run Variance Verification



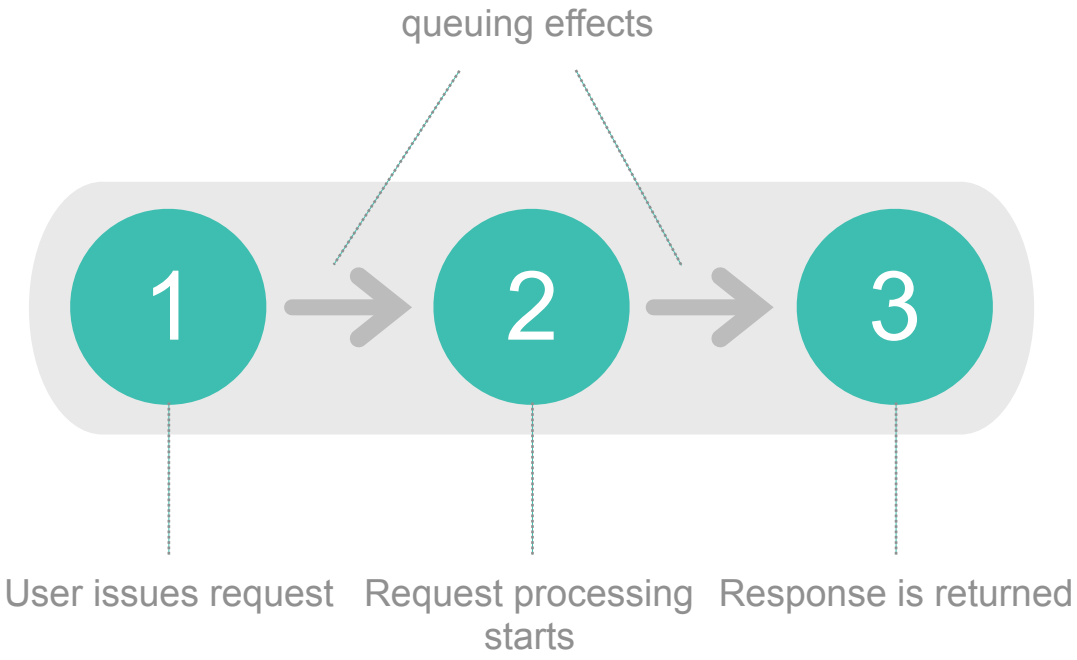
elastic

Sin #5: Denying Statistics

Latency Measurement

- The meaningless mean: Half of the responses are worse than the mean
- Cannot calculate 99.99th percentile from 10 samples
- Don’t average percentiles
- Latency distribution is multi-modal

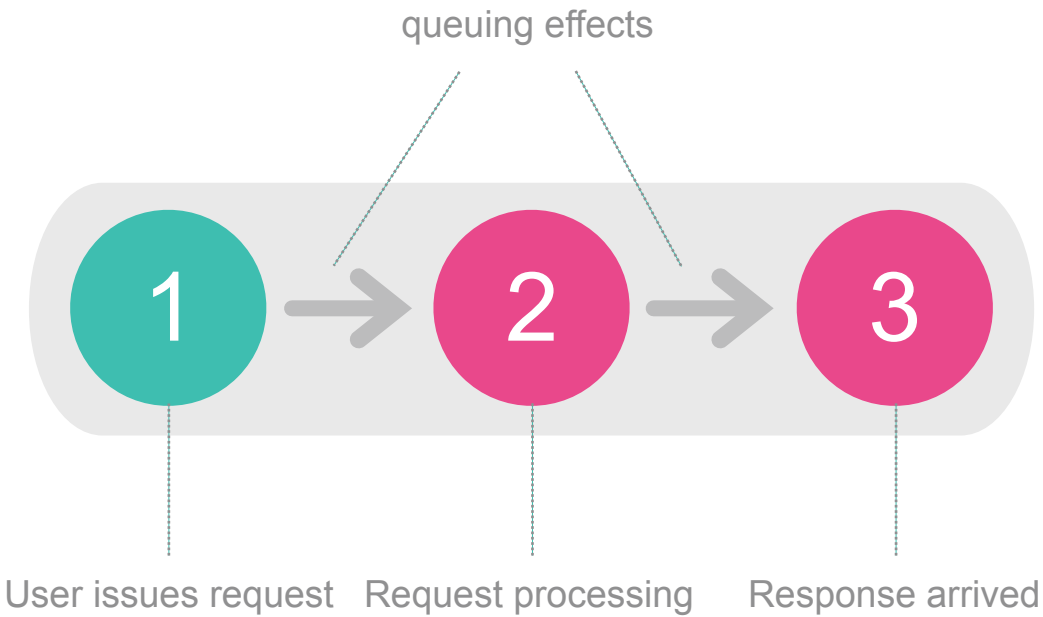
Sin #6: Vague metrics



elastic

Sin #6: Vague metrics

Service Time



elastic

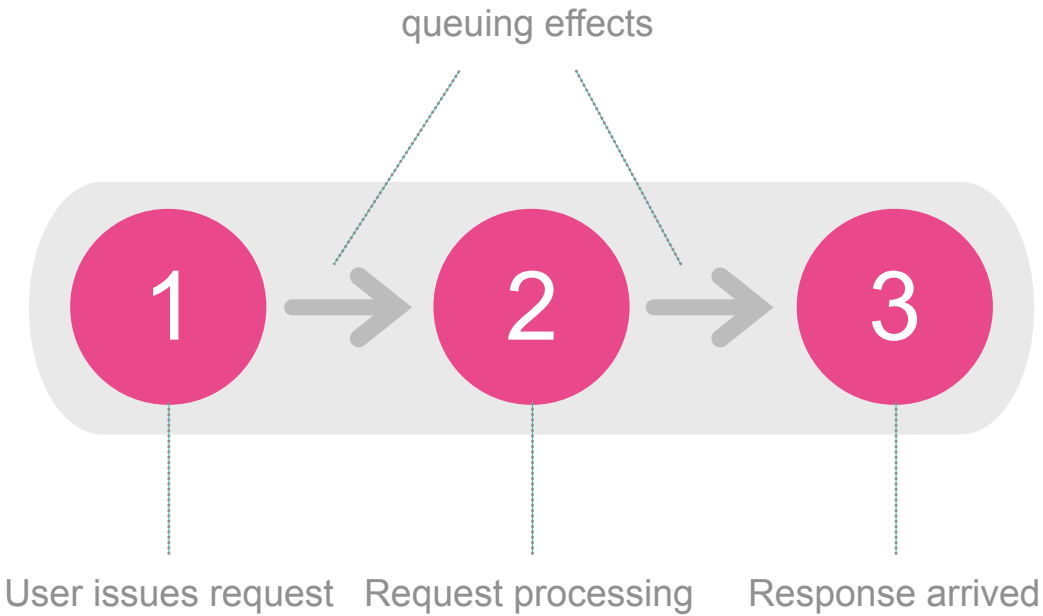
Sin #6: Vague metrics

Service Time

```
while (!isDone()) {
    long start = System.nanoTime();
    // block until the request has finished
    send(createRequest());
    long end = System.nanoTime();
    long serviceTime = end - start;
}
```

Sin #6: Vague metrics

Latency



elastic

Sin #6: Vague metrics

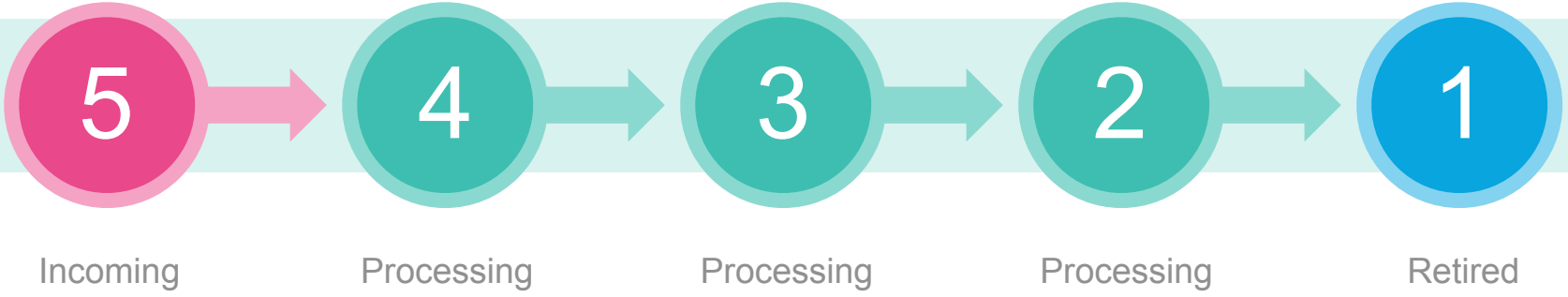
Latency: Include wait time

```
// generator thread
while (!isDoneGenerating()) {
    long start = System.nanoTime();
    queue.put(createRequest(), start);
}

// request issuing thread
while (!isDoneSending()) {
    request, start = queue.take();
    send(request);
    long end = System.nanoTime();
    long latency = end - start;
}
```

Sin #6: Vague metrics

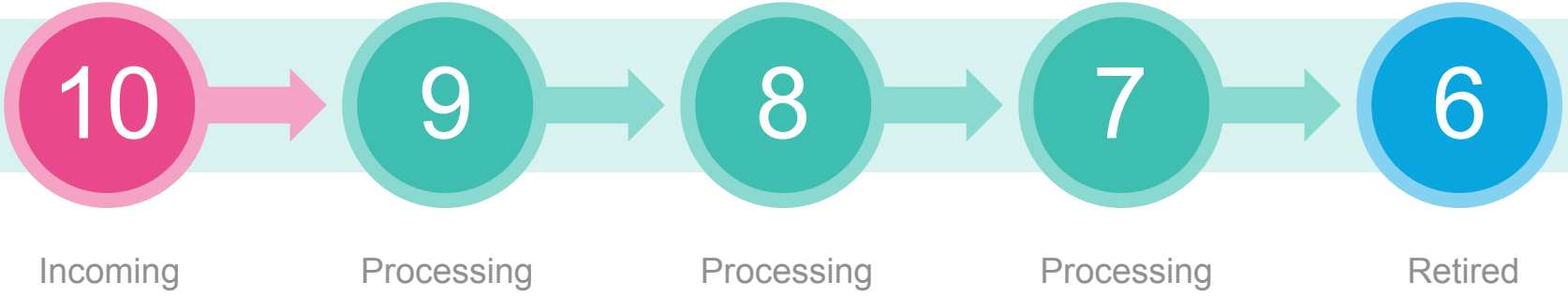
Throughput: System at t = n seconds



elastic

Sin #6: Vague metrics

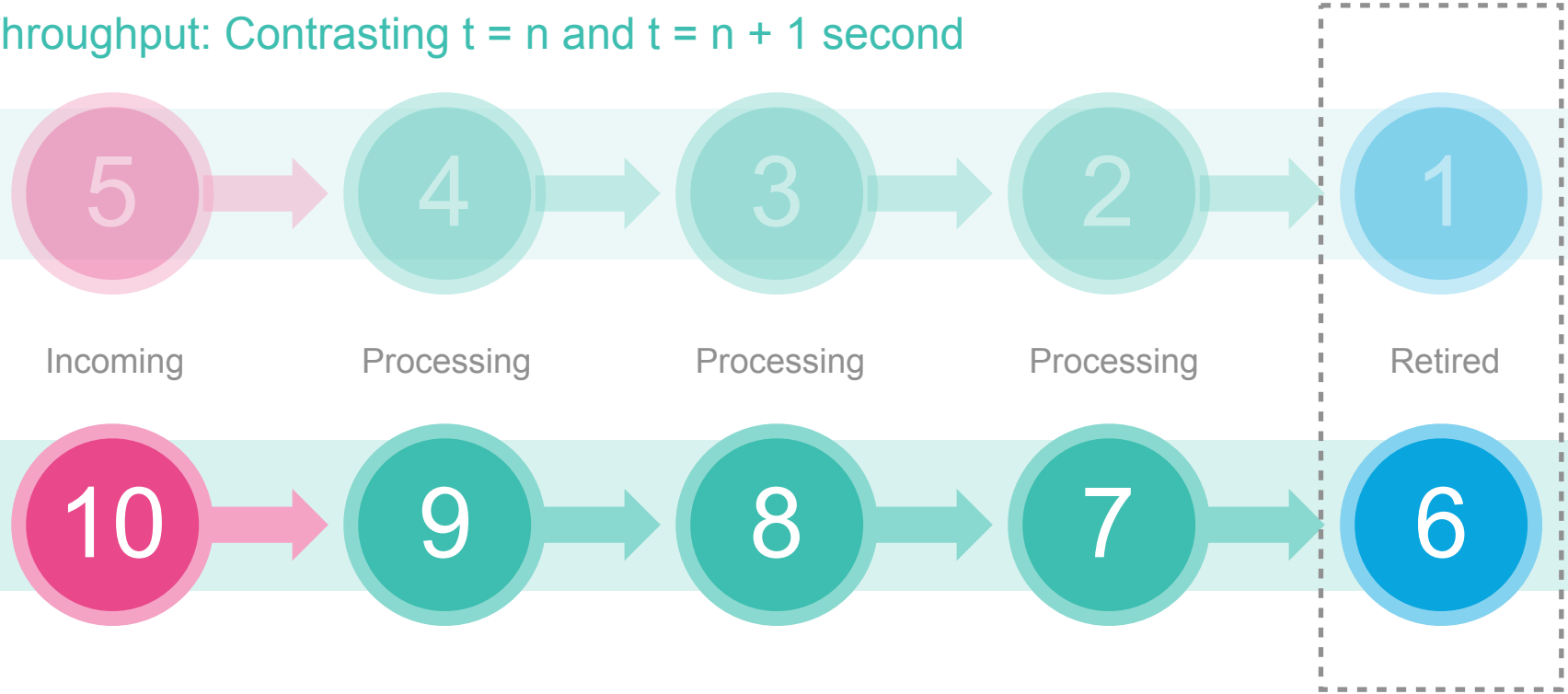
Throughput: System at t = n + 1 second



elastic

Sin #6: Vague metrics

Throughput: Contrasting t = n and t = n + 1 second



elastic

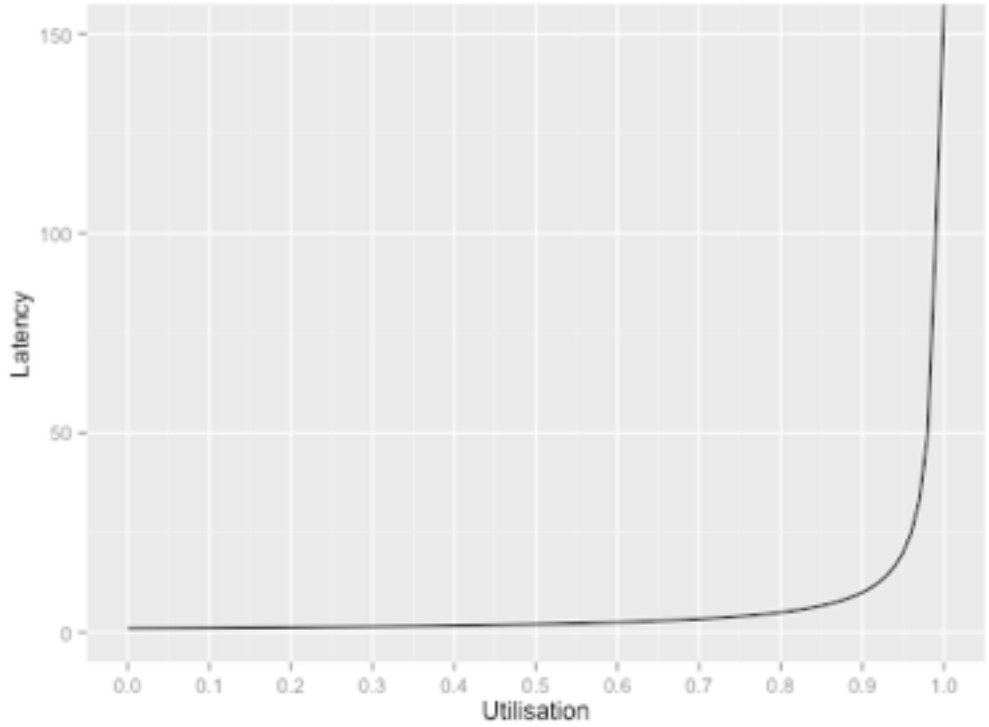
Sin #6: Vague metrics

Resulting Throughput

5 ops/s

Sin #6: Vague metrics

Latency … at which throughput?



 Source: http://robharrop.github.io/maths/performance/2016/02/20/service-latency-and-utilisation.html

Sin #6: Vague metrics

Latency at a defined throughput

```
// generator thread
while (!isDoneGenerating()) {
    long start = System.nanoTime();
    queue.put(createRequest(), start);
    Thread.sleep(waitTime(targetThroughput));
}

// request issuing thread
while (!isDoneSending()) {
    request, start = queue.take();
    send(request);
    long end = System.nanoTime();
    long latency = end - start;
}
```

Sin #7: Treat Performance as One-Dimensional

Vary inputs

- Bulk size
- Query parameters
- Document structure

Sin #7: Treat Performance as One-Dimensional

Vary execution order

- Run queries in different order: Avoid caching effects
- Interfere operations: How does indexing behave with concurrent queries?

Sin #7: Treat Performance as One-Dimensional

And more

- Hardware
- OS
- JDK
- …

Summary


1M
HCl

Performance is easy, all you need to know is everything

Sergey Kuksenko, Oracle Performance Engineer

Questions?

Slides

https://bit.ly/rally-muc

Further Resources

- “How not to measure latency”: http://www.youtube.com/watch?v=lJ8ydIuPFeU
- “Benchmarking Blunders and Things That Go Bump in the Night”: http://arxiv.org/pdf/cs/0404043v1.pdf

Image Sources

- “Searching” by “Skyblueseed”: https://www.flickr.com/photos/skyblueseed/5792500545/ (CC BY-NC-ND 2.0)
- “Curiosity Mastcam L sol 673” by “2di7 & titanio44”: https://www.flickr.com/photos/lunexit/14570422596/ (CC BY-NC-ND 2.0)
- “80's style Hacker Picture” by “Brian Klug”: https://www.flickr.com/photos/brianklug/6870005158/ (CC BY-NC 2.0)
- “Works Mini Cooper S DJB 93B” by “Andrew Basterfield”: https://www.flickr.com/photos/andrewbasterfield/4759364589/ (CC BY-SA 2.0)

Image Sources

- “photo” by “Odi Kosmatos”: https://www.flickr.com/photos/kosmatos/8162850619/ (CC BY 2.0)
- “gags9999”: https://www.flickr.com/photos/gags9999/14124313715/ (CC BY 2.0)
- “Bachelor Students - Chemistry Lab” by “NTNU”: https://www.flickr.com/photos/92416586@N05/12188423293/ (CC BY 2.0)
- “42” by “Lisa Risager”: https://www.flickr.com/photos/risager/5067595483/ (CC BY-SA 2.0)