How NOT to Measure Latency

An attempt to share wisdom...

Matt Schuetze, Product Management Director, Azul Systems

©2013 Azul Systems, Inc.

High level agenda

- Some latency behavior background
- The pitfalls of using “statistics”
- Latency “philosophy” questions
- The Coordinated Omission Problem
- Some useful tools
- Use tools for bragging

©2013 Azul Systems, Inc.

About Gil Tene – Intended Speaker

- co-founder, CTO @Azul Systems
- Have been working on “think different” GC approaches since 2002
- Created Pauseless & C4 core GC algorithms (Tene, Wolf)
- A Long history building Virtual & Physical Machines, Operating Systems, Enterprise apps, etc...
- JCP EC Member...
- Not Cloned Yet…

©2013 Azul Systems, Inc.

About me: Matt Schuetze

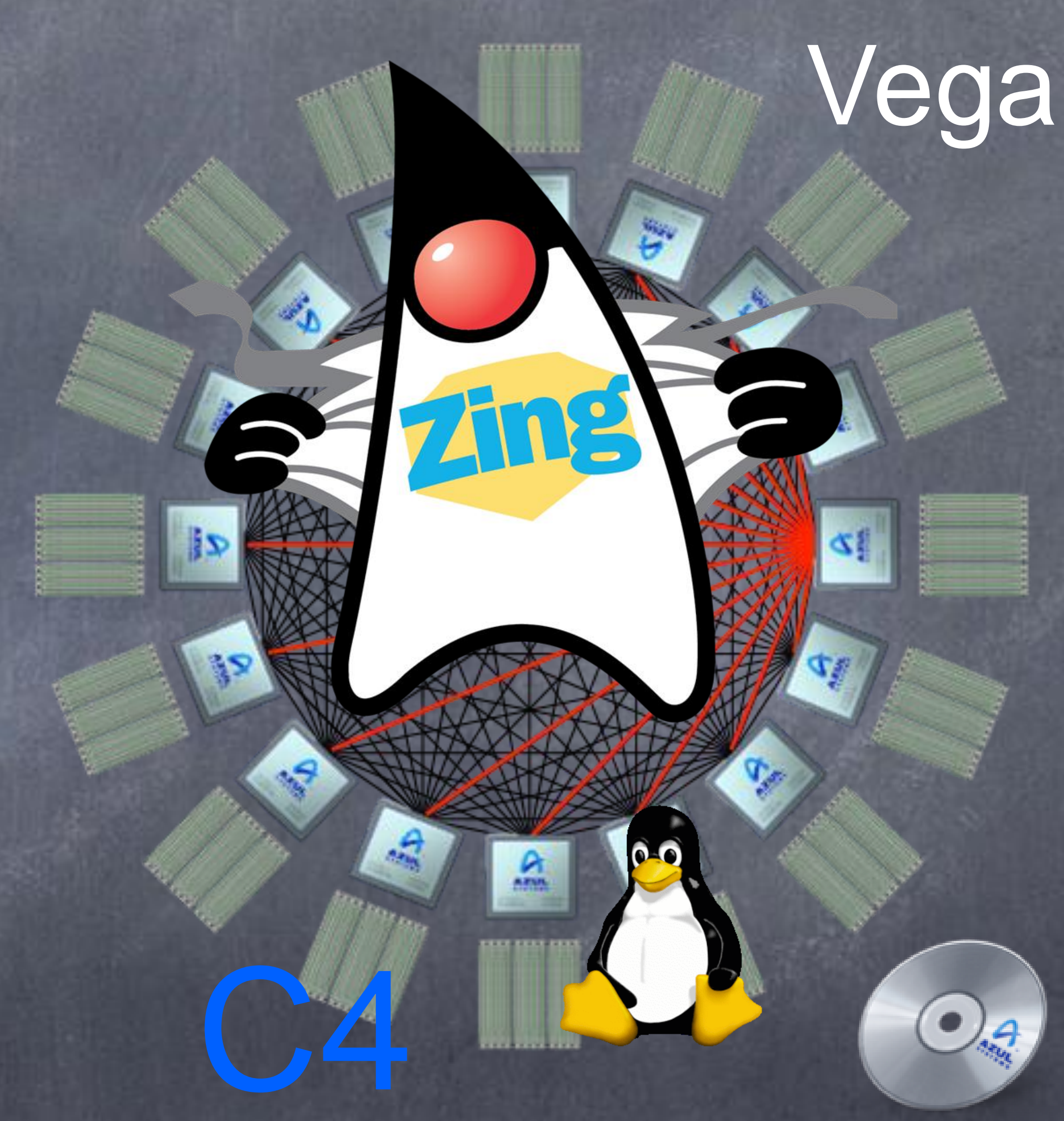
- Product Manager @Azul Systems
- Stewardship of Azul's product roadmap.
- Started career in radar systems. Measured how "stealthy" is an aircraft.
- Moved to enterprise software development in 2000
- Built professional grade monitoring and profiling tools. More types of measurements.
- Measure my clone: Ben Affleck

©2013 Azul Systems, Inc.

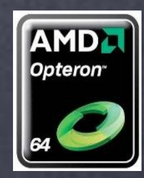
About Azul

- We make scalable Virtual Machines
- Have built "whatever it takes to get job done" since 2002
- 3 generations of custom SMP Multi-core HW (Vega)
- Zing: Pure software for commodity x86
- Known for Low Latency, Consistent execution, and Large data set excellence

Vega



C4



©2013 Azul Systems, Inc.

Common fallacies

- Computers run application code continuously

Wrong!

- Response time can be measured as work units/time

Wrong!

- Response time exhibits a normal (or Gaussian or Poisson) distribution

Wrong!

- “Glitches” or “Semi-random omissions” in measurement don’t have a big effect.

Wrong!

©2013 Azul Systems, Inc.

A classic look at response time behavior

Response time as a function of load

Average?
Max?
Median?
90%?
99.9%

Response time

C Unacceptable (poor) response time

B Acceptable response time

A Good response time

Increasing load or
decreasing resource availability

* source: IBM CICS server documentation, “understanding response times”

©2012 Azul Systems, Inc.

Response time over time

When we measure behavior over time, we often see:

"Hiccups"

Server Response Time(milliseconds)

Elapsed Time(Seconds)

Active User Count

validate_user_login • create_schedule ▲ update_user_details • Active User Count

* source: ZOHO QEngine White Paper: performance testing report analysis

©2012 Azul Systems, Inc.

What happened here?

"Hiccups"

Hiccups by Time Interval

Max per Interval — 99% — 99.90% — 99.99% — Max

Hiccup Duration (msec): 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000

Elapsed Time (sec): 0, 20, 40, 60, 80, 100, 120, 140, 160, 180

* Source: Gil running an idle program and suspending it five times in the middle

©2012 Azul Systems, Inc.

The real world (a low latency example)

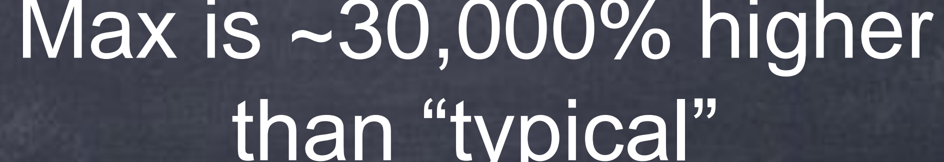
Hiccups by Time Interval

Max per Interval — 99% — 99.90% — 99.99% — Max

Hiccup Duration (msec): 0, 5, 10, 15, 20, 25

Elapsed Time (sec): 0, 100, 200, 300, 400, 500, 600

99%'ile is ~60 usec



©2012 Azul Systems, Inc.

Hiccups are [typically] strongly multi-modal

- They don’t look anything like a normal distribution
- They usually look like periodic freezes
- A complete shift from one mode/behavior to another
- Mode A: “good”.
- Mode B: “Somewhat bad”
- Mode C: “terrible”, ...
- ....

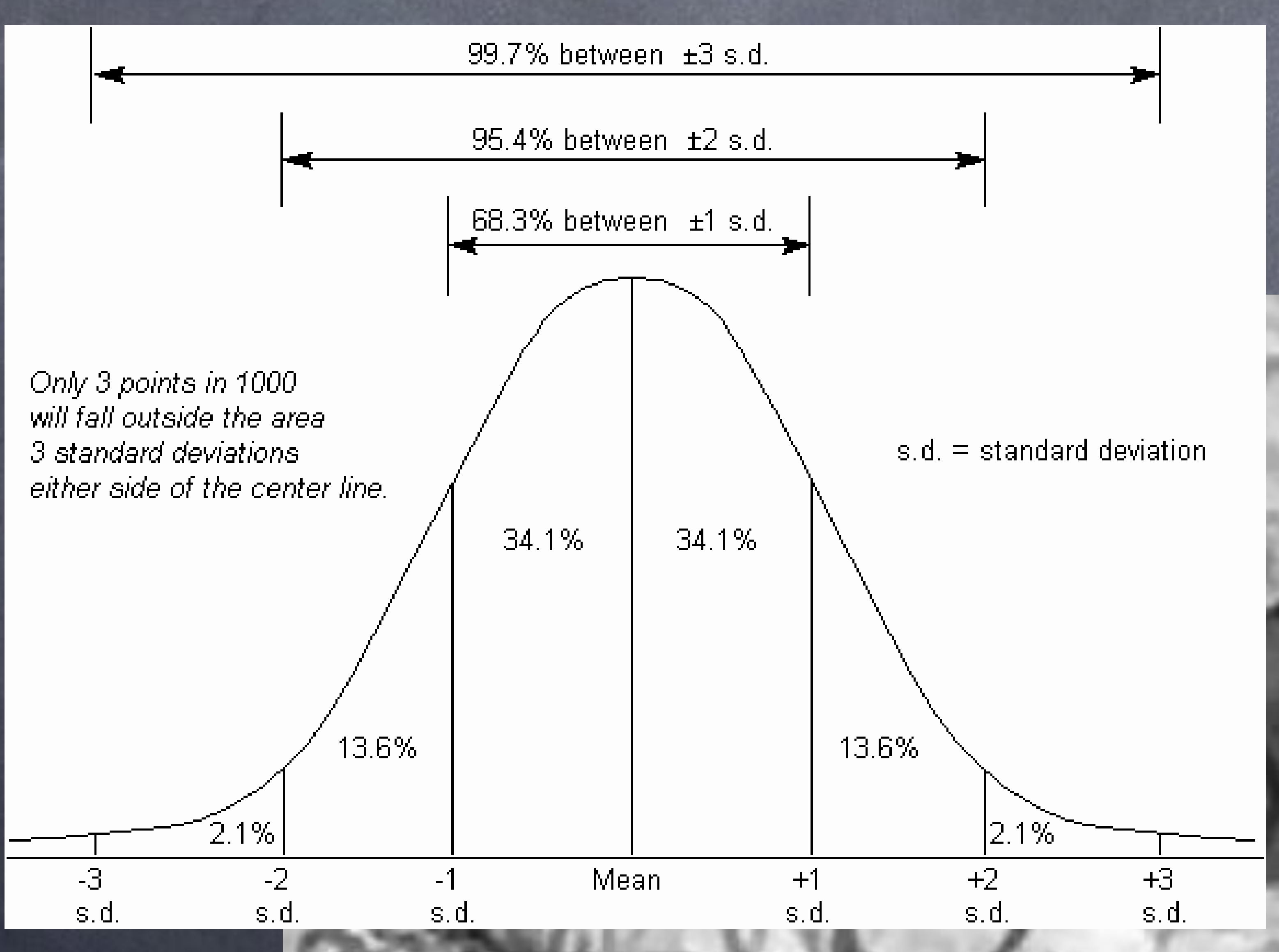
©2012 Azul Systems, Inc.

Common ways people deal with hiccups

©2012 Azul Systems, Inc.

Common ways people deal with hiccups

Averages and Standard Deviation



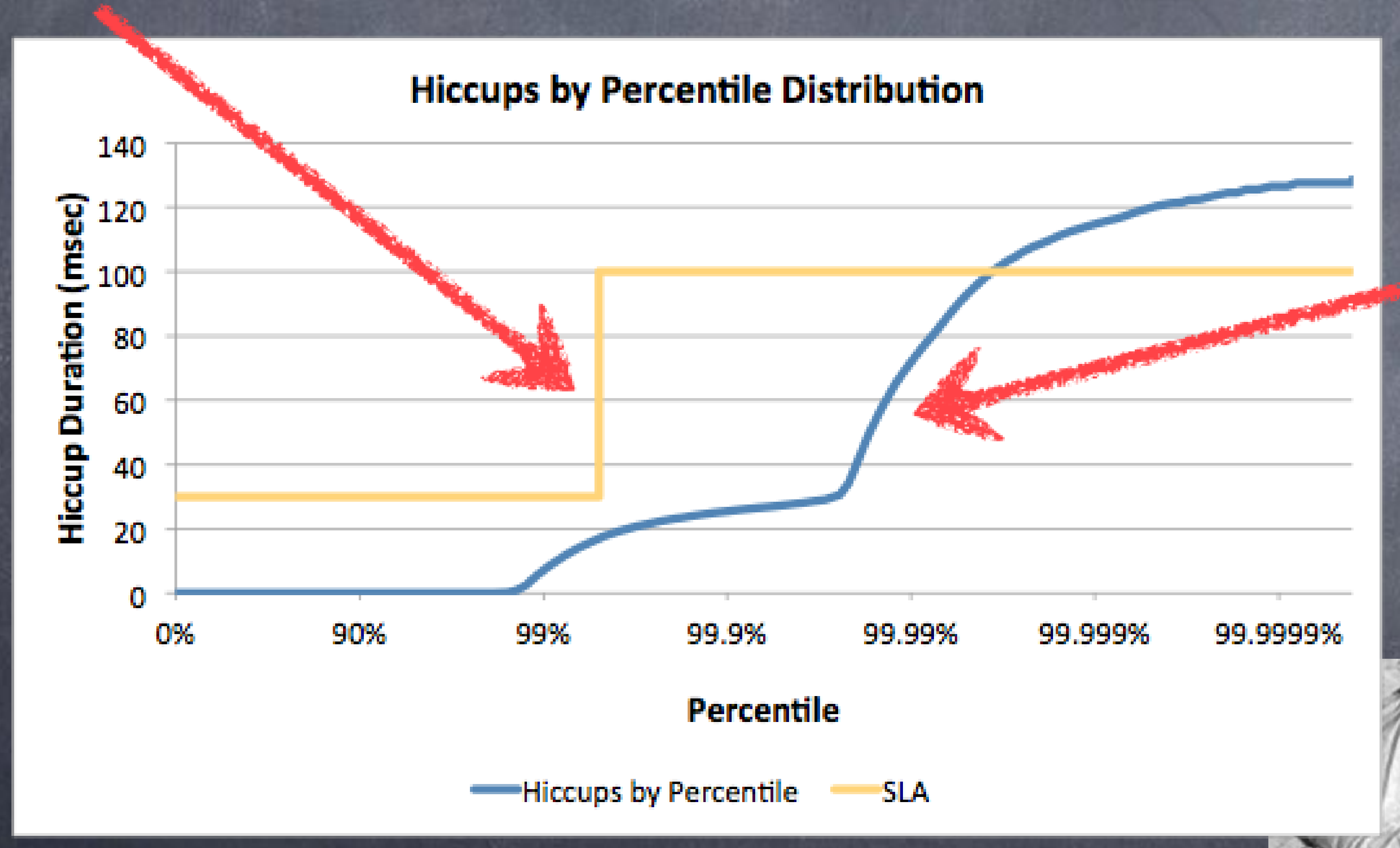
Always Wrong!

©2012 Azul Systems, Inc.

Better ways people can deal with hiccups

Actually measuring percentiles

Requirements



Response Time Percentile plot line

©2012 Azul Systems, Inc.

Requirements

Why we measure latency and response times to begin with...

©2013 Azul Systems, Inc.

Latency tells us how long something took

- But what do we WANT the latency to be?
- What do we want the latency to BEHAVE like?
- Latency requirements are usually a PASS/FAIL test of some predefined criteria
- Different applications have different needs
- Requirements should reflect application needs
- Measurements should provide data to evaluate requirements

©2012 Azul Systems, Inc.

The Olympics
aka “ring the bell first”

- Goal: Get gold medals
- Need to be faster than everyone else at SOME races
- Ok to be slower in some, as long as fastest at some (the average speed doesn’t matter)
- Ok to not even finish or compete (the worst case and 99%‘ile don’t matter)
- Different strategies can apply. E.g. compete in only 3 races to not risk burning out, or compete in 8 races in hope of winning two

©2013 Azul Systems, Inc.

Pacemakers

aka “hard” real time

- Goal: Keep heart beating
- Need to never be slower than X
- “Your heart will keep beating 99.9% of the time” is not reassuring
- Having a good average and a nice standard deviation don’t matter or help
- The worst case is all that matters

©2013 Azul Systems, Inc.

“Low Latency” Trading

aka “soft” real time

- Goal A: Be fast enough to make some good plays
- Goal B: Contain risk and exposure while making plays
- E.g. want to “typically” react within 200 usec.
- But can’t afford to hold open position for 20 msec, or react to 30 msec stale information
- So we want a very good “typical” (median, 50%‘ile)
- But we also need a reasonable Max, or 99.99%‘ile

©2013 Azul Systems, Inc.

Interactive applications

aka “squishy” real time

- Goal: Keep users happy enough to not complain/leave
- Need to have “typically snappy” behavior
- Ok to have occasional longer times, but not too high, and not too often
- Example: 90% of responses should be below 0.2 sec, 99% should be below 0.5 sec, 99.9 should be better than 2 seconds. And a >10 second response should never happen.
- Remember: A single user may have 100s of interactions per session...

©2013 Azul Systems, Inc.

Establishing Requirements

an interactive interview (or thought) process

- Q: What are your latency requirements?
- A: We need an avg. response of 20 msec
- Q: Ok. Typical/average of 20 msec... So what is the worst case requirement?
- A: We don’t have one
- Q: So it’s ok for some things to take more than 5 hours?
- A: No way in H%%&!
- Q: So I’ll write down “5 hours worst case...”
- A: No. That’s not what I said. Make that “nothing worse than 100 msec”
- Q: Are you sure? Even if it’s only two times a day?
- A: Ok... Make it “nothing worse than 2 seconds...”

©2013 Azul Systems, Inc.

Establishing Requirements

an interactive interview (or thought) process

- Ok. So we need a typical of 20msec, and a worst case of 2 seconds. How often is it ok to have a 1 second response?
- A: (Annoyed) I thought you said only a few times a day
- Q: That was for the worst case. But if half the results are better than 20 msec, is it ok for the other half to be just short of 2 seconds? What % of the time are you willing to take a 1 second, or a half second hiccup? Or some other level?
- A: Oh. Let's see. We have to better than 50 msec 90% of the time, or we'll be losing money even when we are fast the rest of the time. We need to be better than 500 msec 99.9% of the time, or our customers will complain and go elsewhere
- Now we have a service level expectation:
    - 50% better than 20 msec
    - 90% better than 50 msec
    - 99.9% better than 500 msec
    - 100% better than 2 seconds

©2013 Azul Systems, Inc.

Latency does not live in a vacuum

©2013 Azul Systems, Inc.

Remember this?

How much load can this system handle?

Sustainable Throughput Level

Response time

Where users complain

C Unacceptable (poor) response time

What the marketing benchmarks will say

Where the sysadmin is willing to go

B Acceptable response time

A Good response time

Increasing load or decreasing resource availability

©2013 Azul Systems, Inc.

Sustainable Throughput: The throughput achieved while safely maintaining service levels

Unsustainable Throughout

©2013 Azul Systems, Inc.

Comparing behavior under different throughputs and/or configurations

Duration by Percentile Distribution

Duration (msec)

120
100
80
60
40
20
0

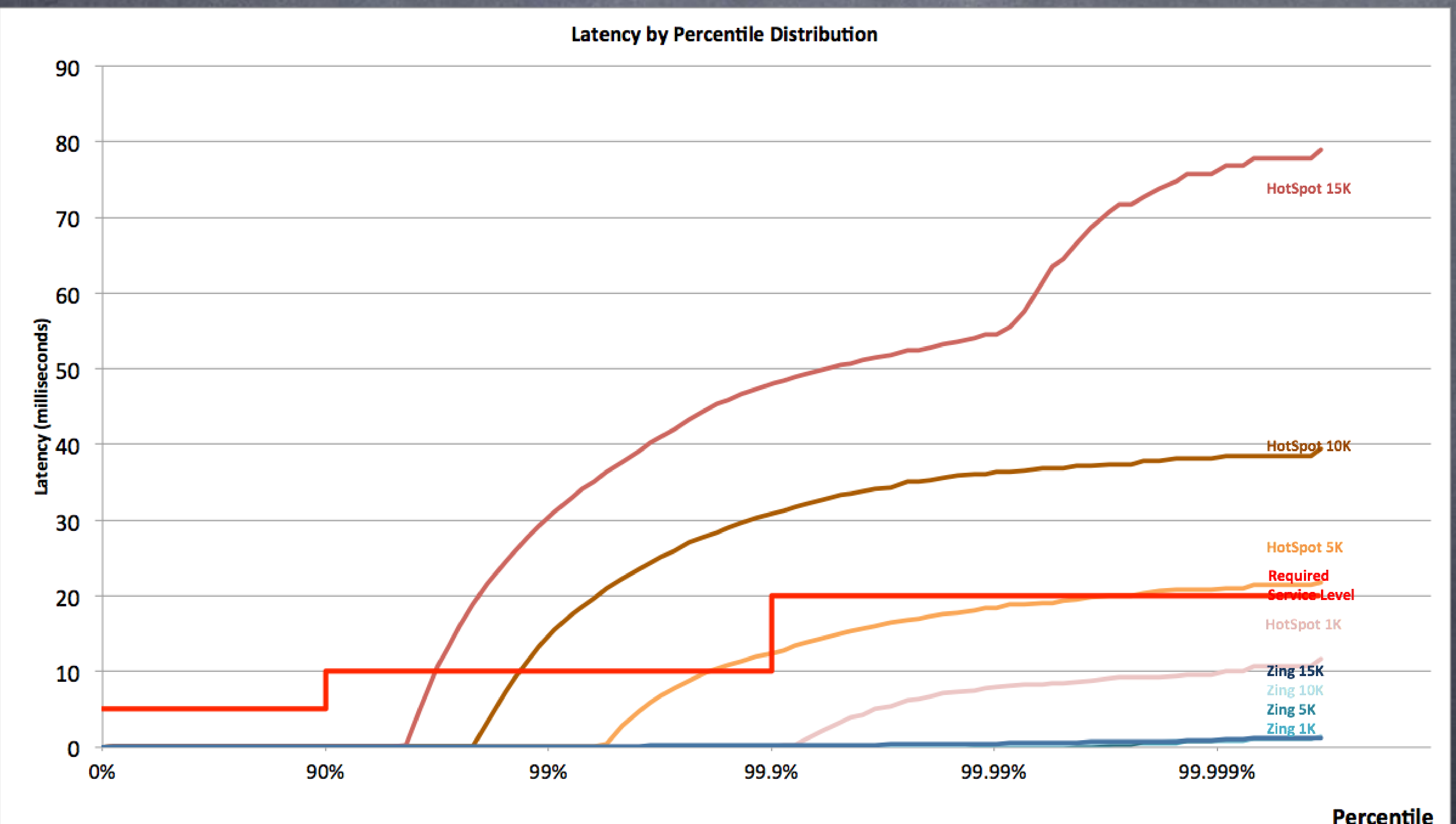
0% 90% 99% 99.9% 99.99% 99.999%

Percentile

Setup D
Setup E
Setup F
Setup A
Setup B
Setup C

©2013 Azul Systems, Inc.

Comparing latency behavior under different throughputs, configurations latency sensitive messaging distribution application



©2013 Azul Systems, Inc.

AZUL SYSTEMS

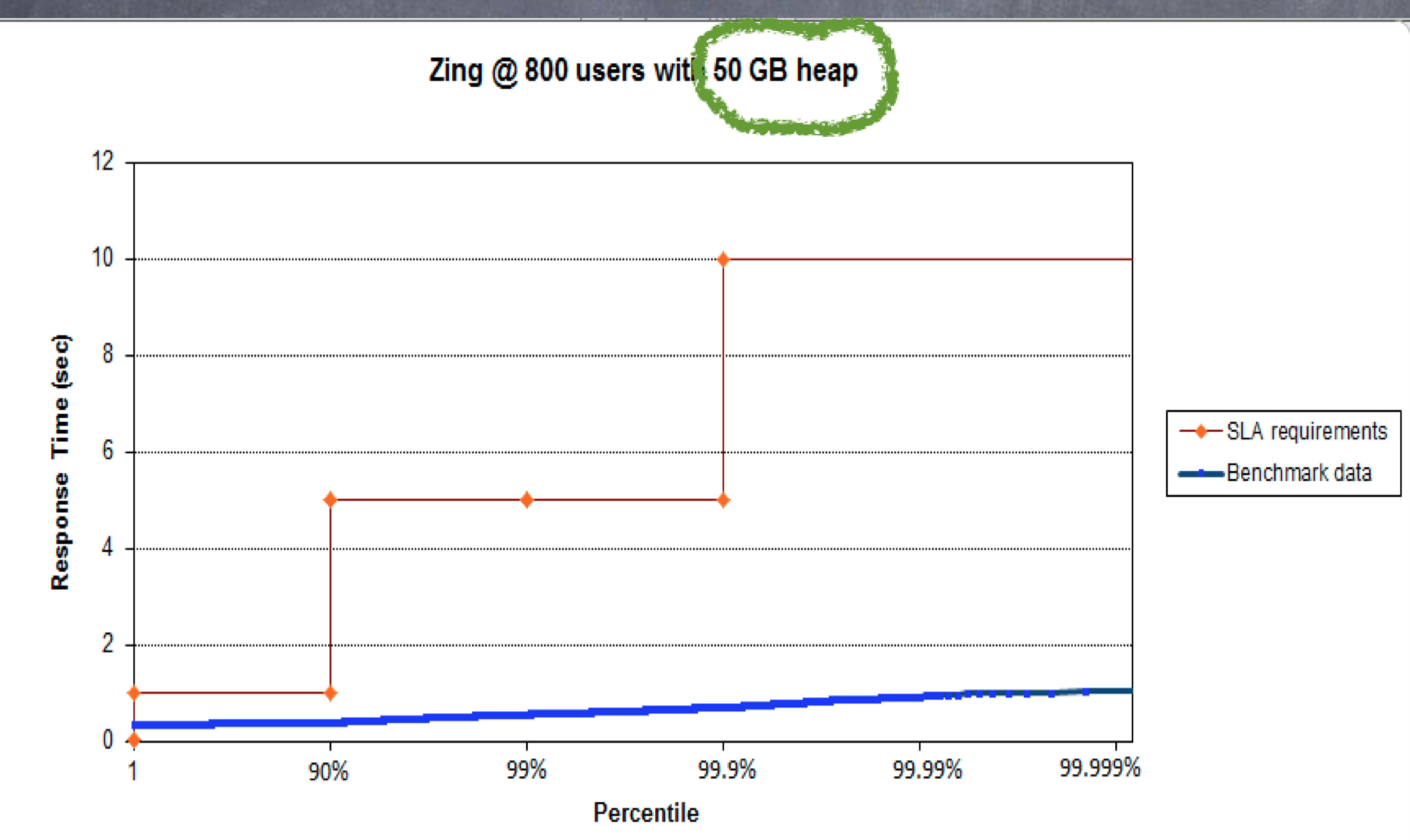
Instance capacity test: “Fat Portal”
HotSpot CMS: Peaks at ~3GB / 45 concurrent users

Native @ 45 users with 3 GB heap

* LifeRay portal on JBoss @ 99.9% SLA of 5 second response times

©2013 Azul Systems, Inc.

Instance capacity test: “Fat Portal”
C4: still smooth @ 800 concurrent users



* LifeRay portal on JBoss @ 99.9% SLA of 5 second response times

©2013 Azul Systems, Inc.

AZUL SYSTEMS

The coordinated omission problem

An accidental conspiracy...

©2013 Azul Systems, Inc.

The coordinated omission problem

- Common Example A (load testing):
  - build/buy load tester to measure system behavior
  - each “client” issues requests one by one at a certain rate
  - measure and log response time for each request
  - results log used to produce histograms, percentiles, etc.
- So what’s wrong with that?
  - works well <u>only</u> when all responses fit within rate interval
  - technique includes implicit “automatic backoff” and coordination
  - But requirements interested in random, uncoordinated requests

©2013 Azul Systems, Inc.

The coordinated omission problem

- Common Example B (monitoring):
  - System monitors and records each transaction latency
  - Latency measured between start and end of each operation
  - keeps observed latency stats of some sort (log, histogram, etc.)
- So what’s wrong with that?
  - works correctly well only when no queuing occurs
  - Long operations only get measured once
  - delays outside of timing window do not get measured at all
  - queued operations are measured wrong

©2013 Azul Systems, Inc.

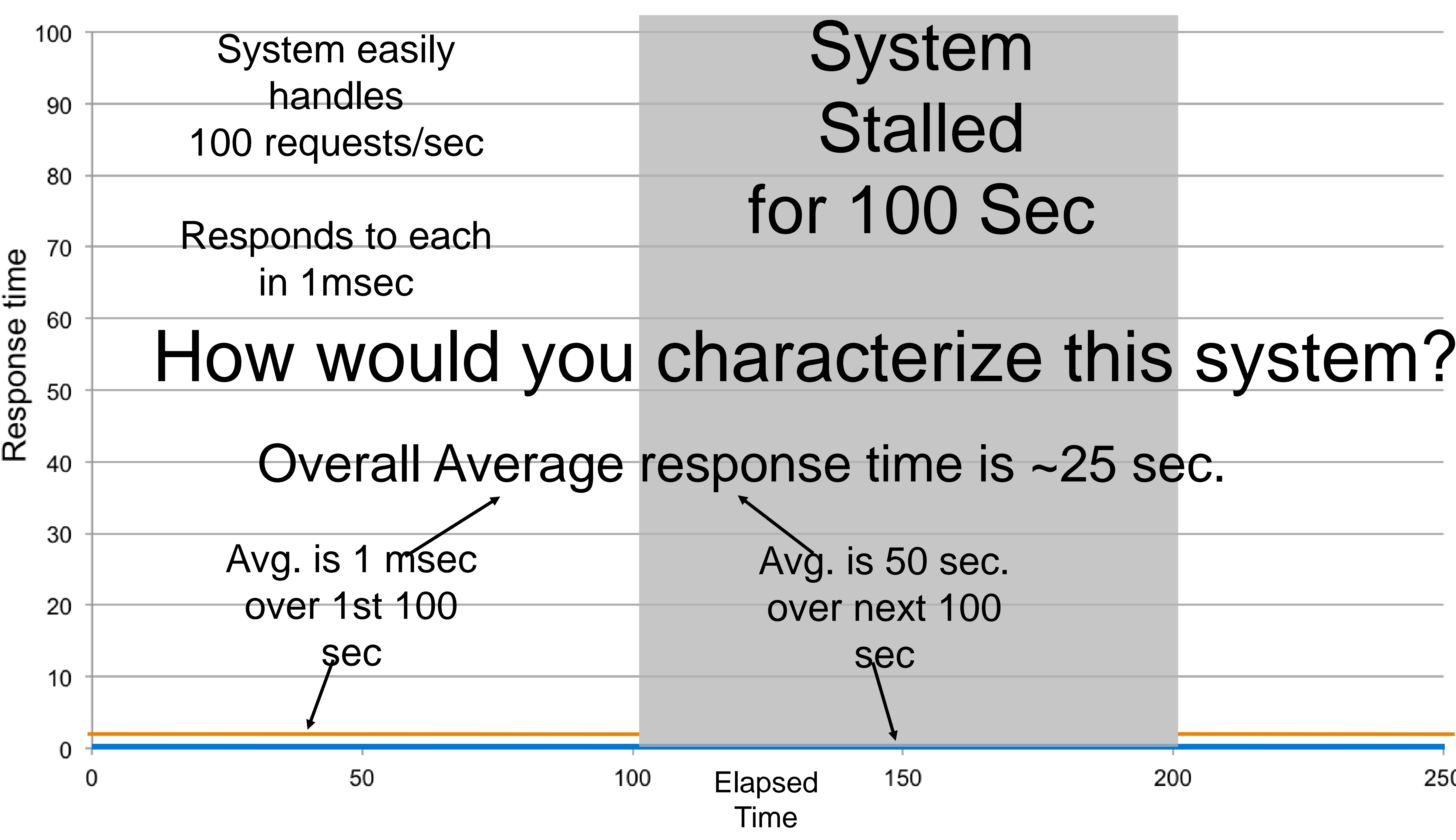
Common Example B: Coordinated Omission in Monitoring Code

```java
/**
 * Performs the actual reading of a row out of the StorageService, fetching
 * a specific set of column names from a given column family.
 */
public static List<Row> read(List<ReadCommand> commands, ConsistencyLevel consistency_level)
        throws UnavailableException, IsBootstrappingException, ReadTimeoutException
{
    if (StorageService.instance.isBootstrapMode())
        throw new IsBootstrappingException();
    long startTime = System.nanoTime();
    List<Row> rows;
    try
    {
        rows = fetchRows(commands, consistency_level);
    }
    finally
    {
        readMetrics.addNano(System.nanoTime() - startTime);
    }
    return rows;
}
```

- Long operations only get measured once
- delays outside of timing window do not get measured at all

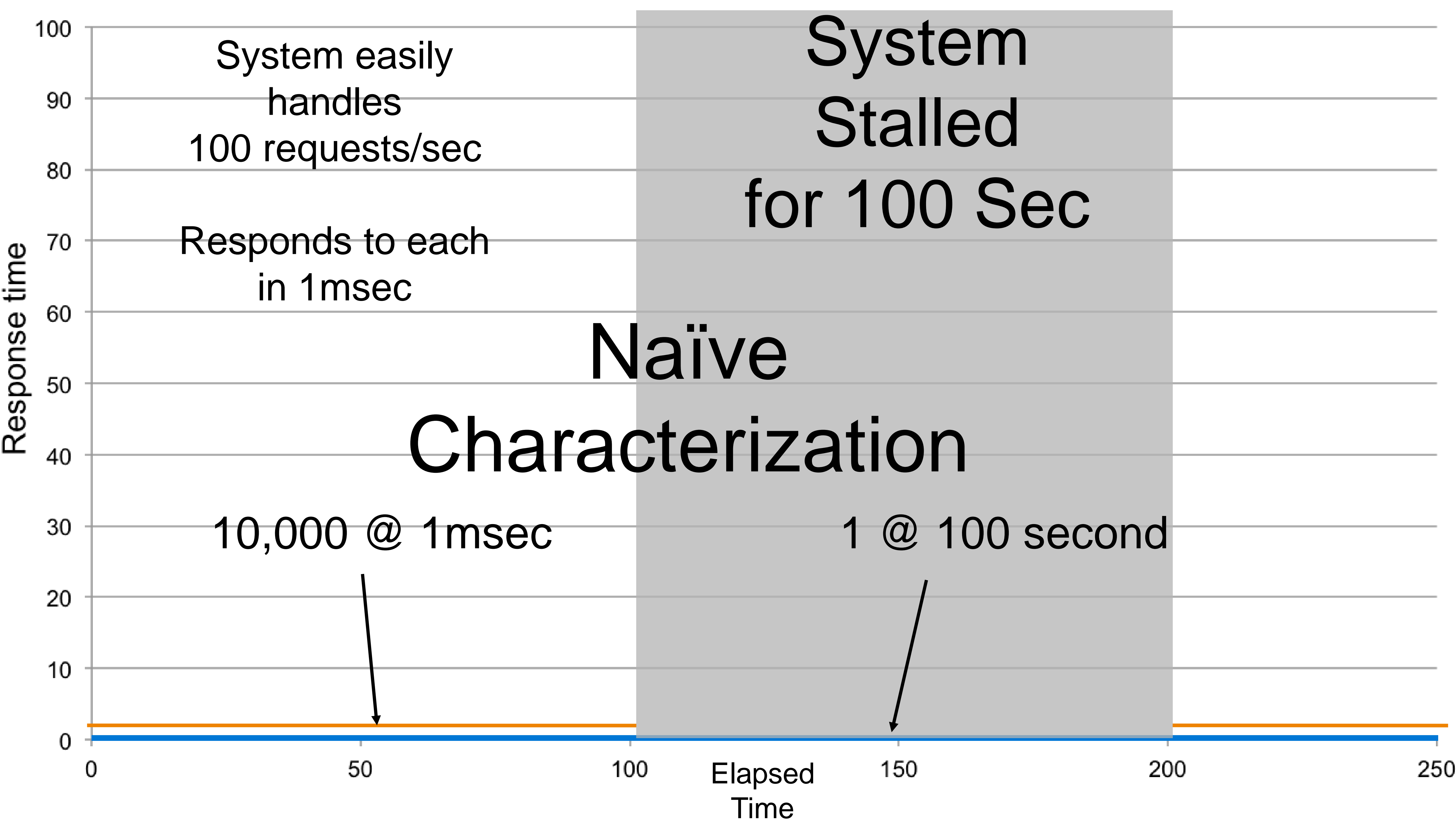
©2013 Azul Systems, Inc.

How bad can this get?



~50%'ile is 1 msec ~75%'ile is 50 sec 99.99%'ile is ~100sec

Measurement in practice

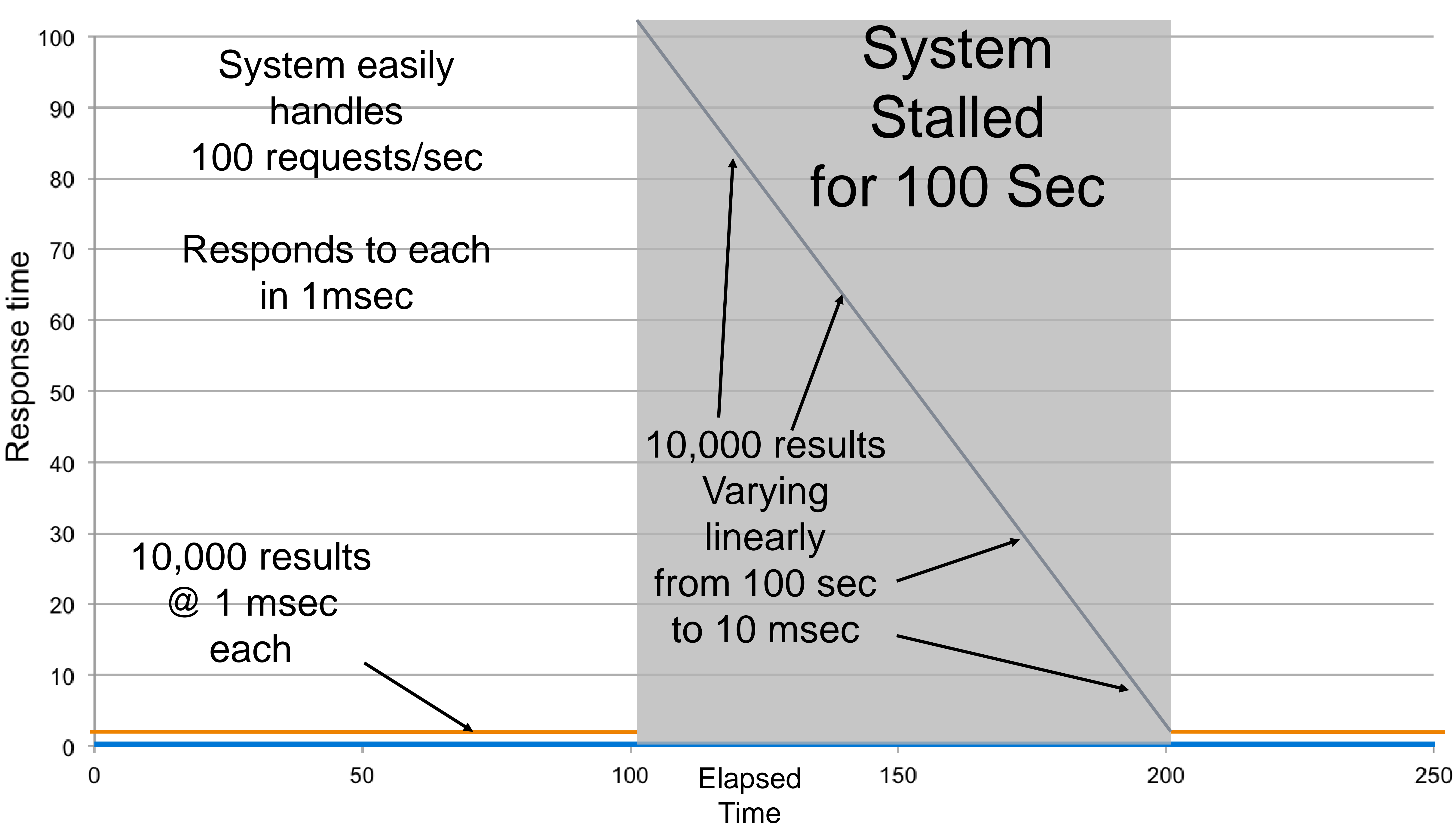


99.99%'ile is 1 msec! (should be ~100sec)

Average. is 10.9msec! (should be ~25 sec)

Std. Dev. is 0.99sec!

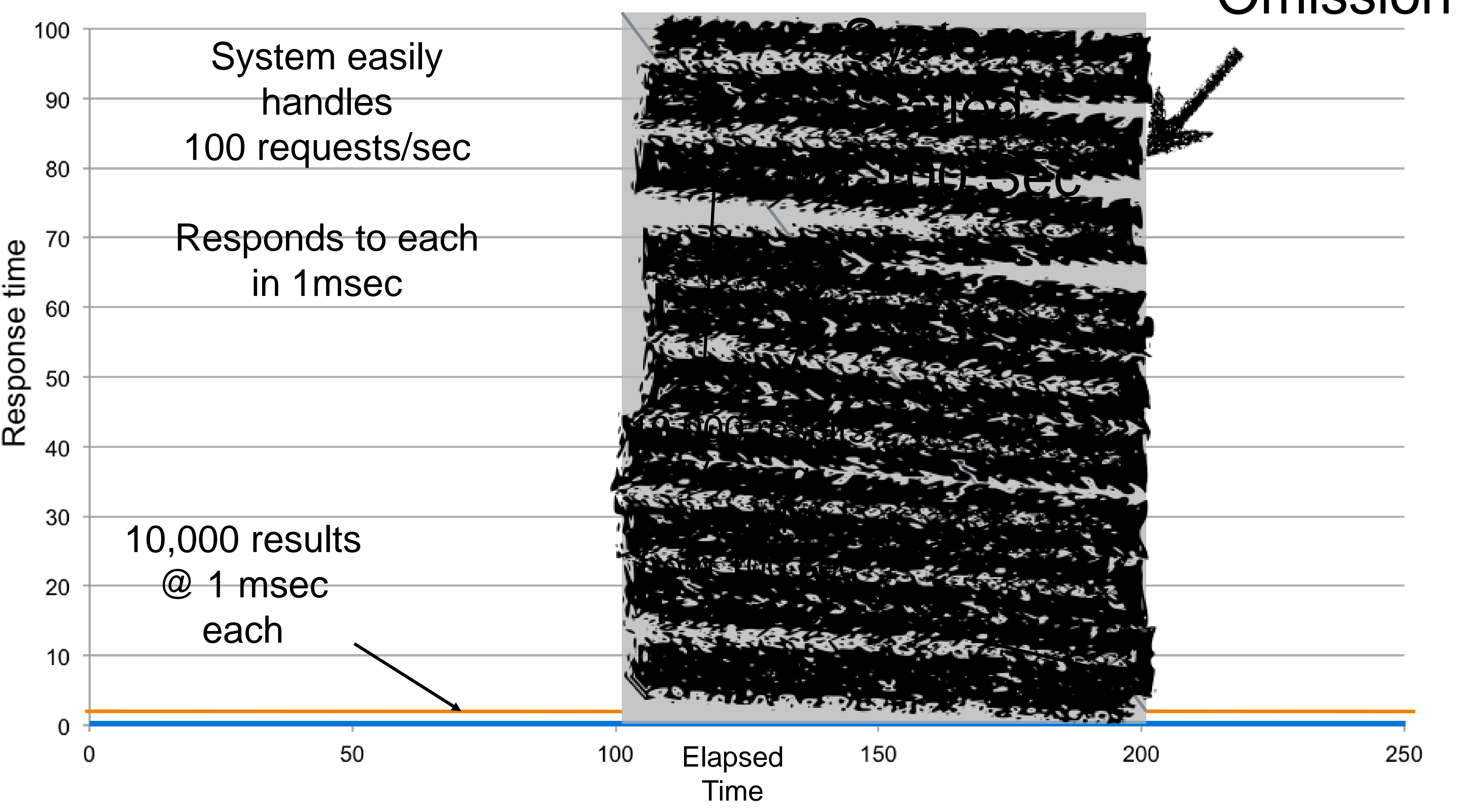
Proper measurement



~50%'ile is 1 msec ~75%'ile is 50 sec 99.99%'ile is ~100sec

Proper measurement

Coordinated Omission



~50%'ile is 1 msec ~75%'ile is 50 sec 99.99%'ile is ~100sec

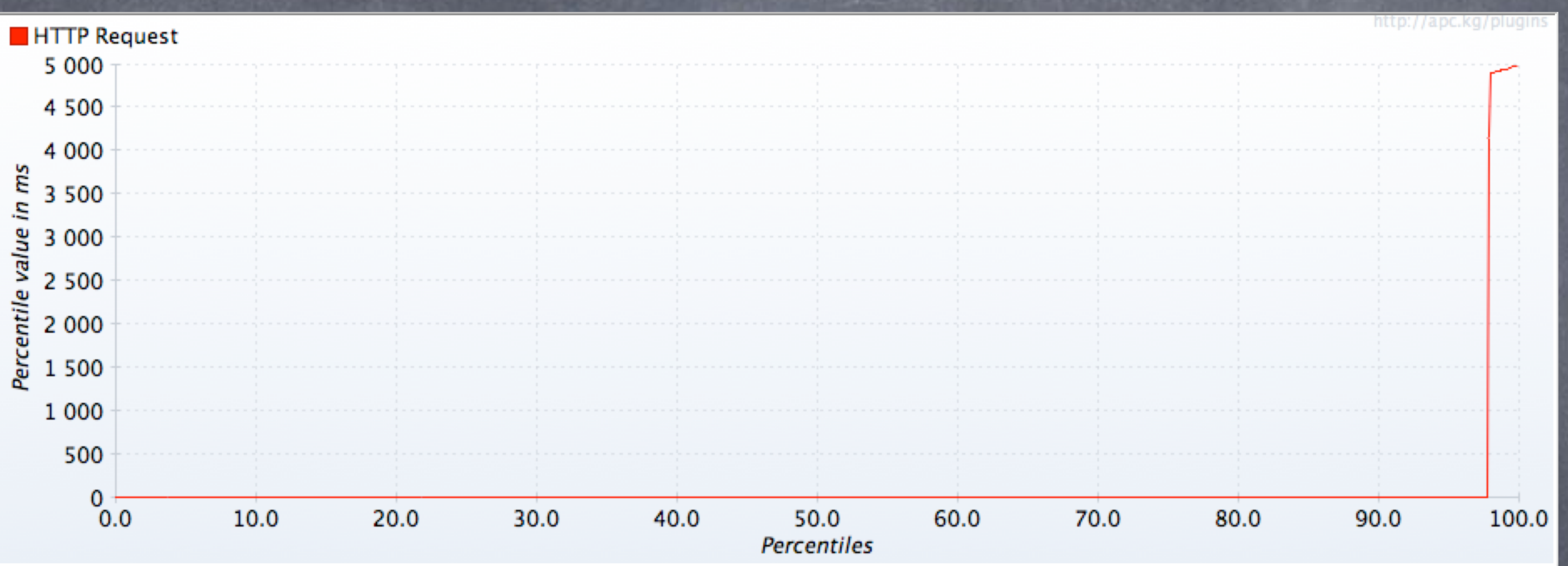
The coordinated omission problem

It is MUCH more common than you may think...

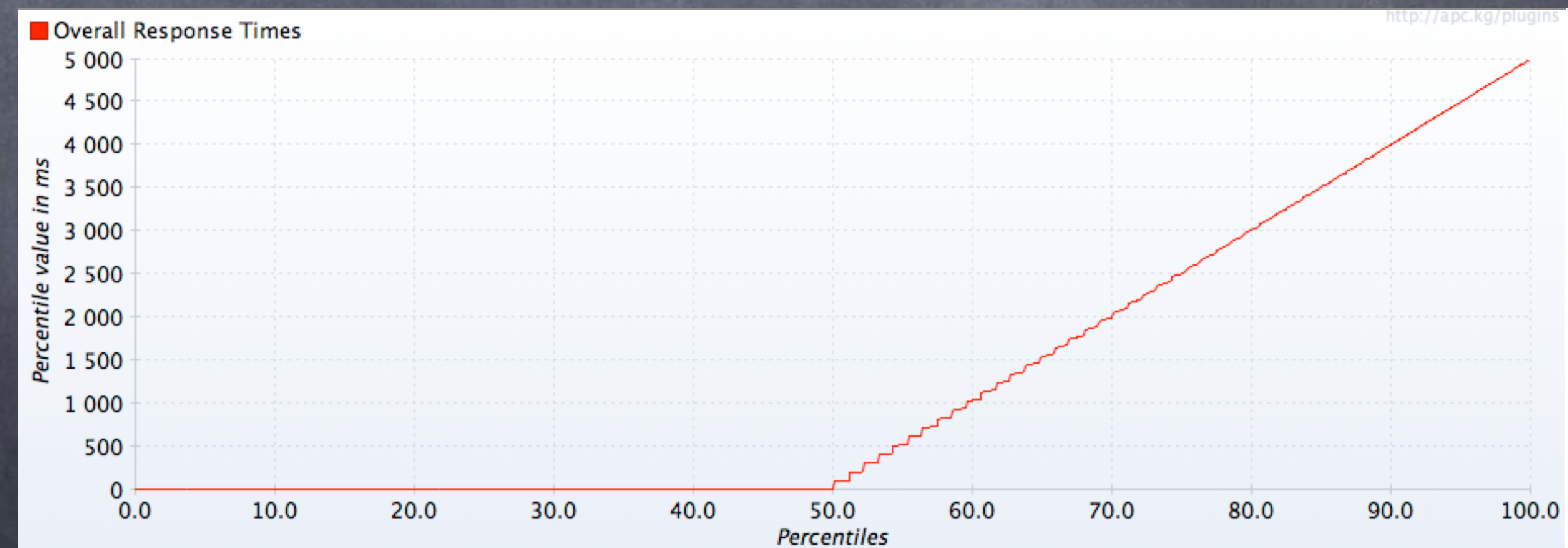
©2013 Azul Systems, Inc.

JMeter makes this mistake... (so do others)

Before Correction



After Correcting for Omission



©2012 Azul Systems, Inc.

The “real” world

Results were collected by a single client thread

[OVERALL], RunTime(ms), 2028755.0
[OVERALL], Throughput(ops/sec), 49291.31413108039
[UPDATE], Operations, 89999169
[UPDATE], AverageLatency(ms), 2.6061162186953O8
[UPDATE], MinLatency(ms), 0
[UPDATE], MaxLatency(ms), 26182
[UPDATE], 95thPercentileLatency(ms), 3
[UPDATE], 99thPercentileLatency(ms), 5

26.182 seconds represents 1.29% of the total time

99%‘ile MUST be at least 0.29% of total time (1.29% - 1%) which would be 5.9 seconds

wrong by a factor of 1,000x

©2012 Azul Systems, Inc.

The “real” world

A world record SPECjEnterprise2010 result

Response Times	Average	Std. dev.	Maximum	90th%	Reqd 90th%
Purchase	0.211	0.63	284.121	0.280	2.000
Manage	0.133	0.52	298.800	0.210	2.000
Browse	0.260	0.82	300.078	0.320	2.000
CreateVehicleEJB	0.303	0.50	11.647	0.610	5.000
CreateVehicleWS	0.276	0.40	305.197	0.520	5.000

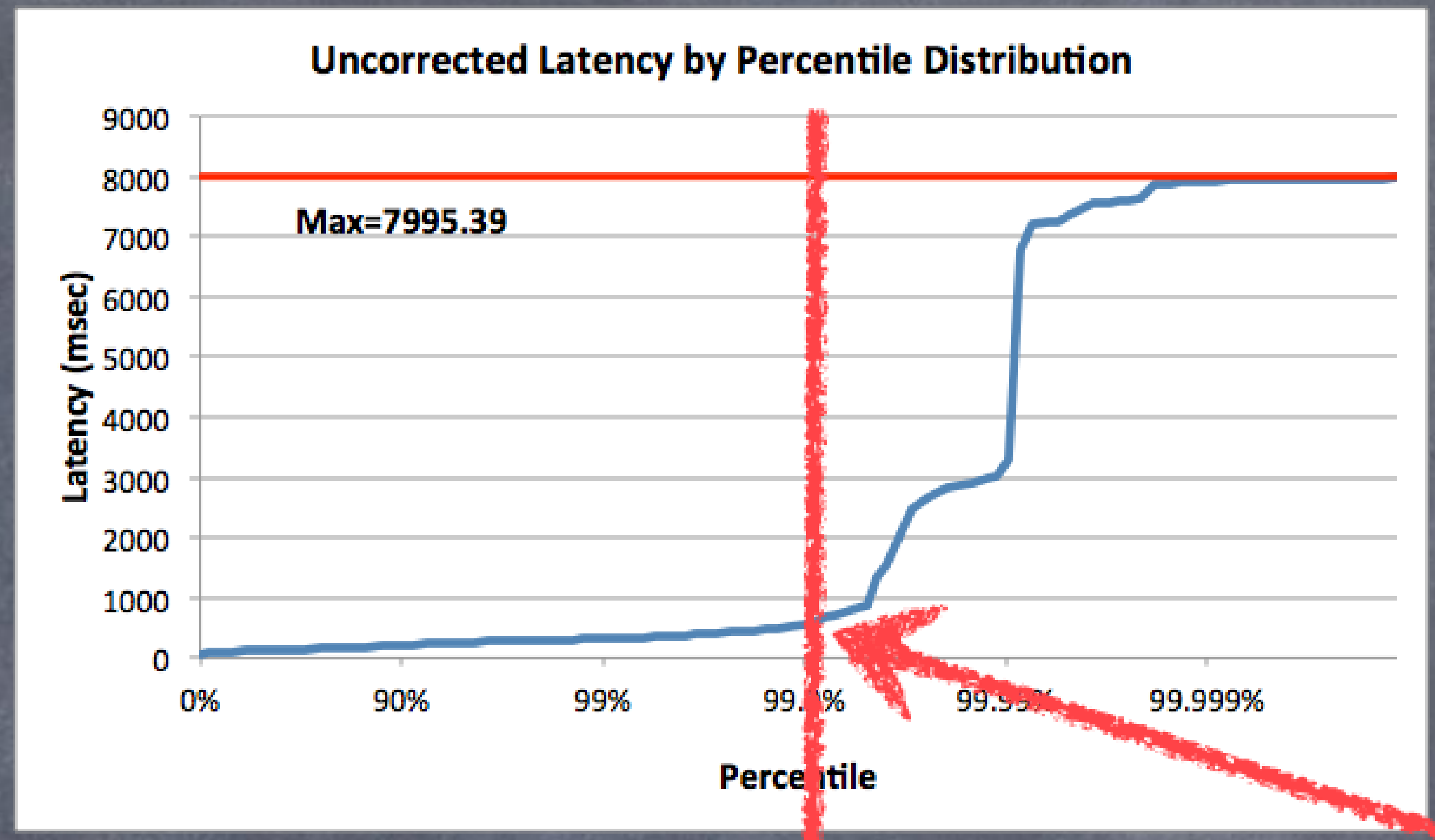
The max is 762 (!!!) standard deviations away from the mean

305.197 seconds represents 8.4% of the timing run

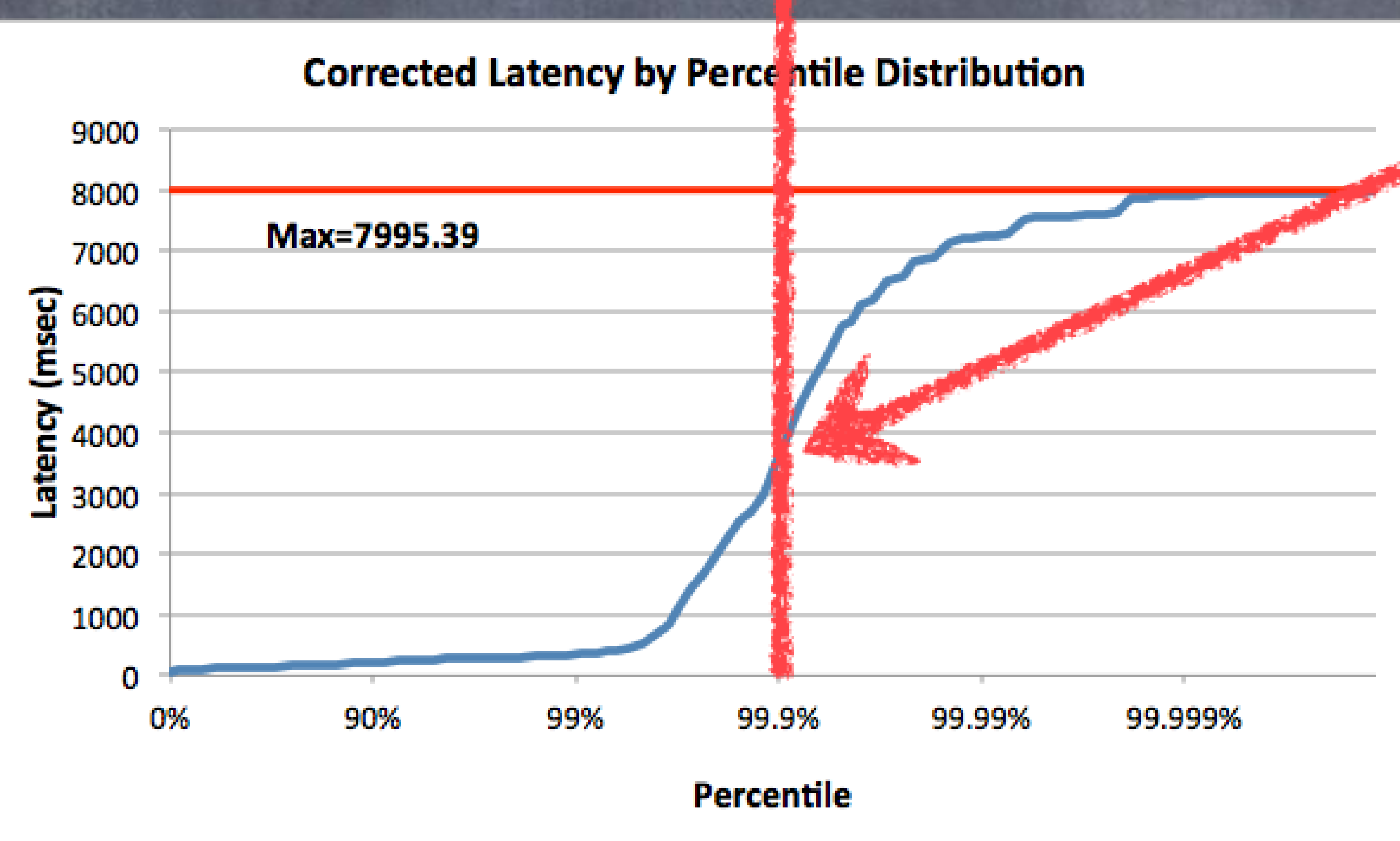
©2012 Azul Systems, Inc.

Real World Coordinated Omission effects

Before Correction



After Correction



Wrong by 7x

©2012 Azul Systems, Inc.

Real World Coordinated Omission effects

Duration by Percentile Distribution

Uncorrected Data

Duration (usec)

600000
500000
400000
300000
200000
100000
0

0% 90% 99% 99.9% 99.99% 99.999% 99.9999%

Percentile

HotSpot Base

©2013 Azul Systems, Inc.

Real World Coordinated Omission effects

Duration by Percentile Distribution

Corrected for Coordinated Omission

Uncorrected Data

Duration (usec)

600000
500000
400000
300000
200000
100000
0

0%
90%
99%
99.9%
99.99%
99.999%
99.9999%

Percentile

HotSpot Base
HotSpot Corrected1
HotSpot Corrected2

©2013 Azul Systems, Inc.

Real World Coordinated Omission effects (Why I care)

Duration by Percentile Distribution

Duration (usec)

600000
500000
400000
300000
200000
100000
0

0% | 90% | 99% | 99.9% | 99.99% | 99.999% | 99.9999%

Percentile

- HotSpot Base
- HotSpot Corrected1
- HotSpot Corrected2
- Zing Base
- Zing Corrected1
- Zing Corrected2

"other" JVM

A ~2500x difference in reported percentile levels for the problem that Zing eliminates

Zing

©2013 Azul Systems, Inc.

Suggestions

- Whatever your measurement technique is, TEST IT.
- Run your measurement method against an artificial system that creates hypothetical pauses scenarios. See if your reported results agree with how you would describe that system behavior
- Don’t waste time analyzing until you establish sanity
- Don’t EVER use or derive from std. deviation
- ALWAYS measure Max time. Consider what it means... Be suspicious.
- Measure %‘iles. Lots of them.

©2013 Azul Systems, Inc.

Some Tools

©2013 Azul Systems, Inc.

HdrHistogram

©2013 Azul Systems, Inc.

HdrHistogram

If you want to be able to produce graphs like this...

Latency by Percentile Distribution

Latency (msec)

160
140
120
100
80
60
40
20
0

Max=137.472

0%
90%
99%
99.9%
99.99%
99.999%
99.9999%

Percentile

You need both good dynamic range
and good resolution

©2012 Azul Systems, Inc.

HdrHistogram background

- Goal: Collect data for good latency characterization...
  - Including acceptable precision at and between varying percentile levels
- Existing alternatives
  - Record all data, analyze later (e.g. sort and get 99.9%'ile).
  - Record in traditional histograms
- Traditional Histograms: Linear bins, Logarithmic bins, or Arbitrary bins
  - Linear requires lots of storage to cover range with good resolution
  - Logarithmic covers wide range but has terrible precisions
  - Arbitrary is.... arbitrary. Works only when you have a good feel for the interesting parts of the value range

©2013 Azul Systems, Inc.

HdrHistogram

- A High Dynamic Range Histogram
  - Covers a configurable dynamic value range
  - At configurable precision (expressed as number of significant digits)
- For Example:
  - Track values between 1 microsecond and 1 hour
  - With 3 decimal points of resolution
- Built-in [optional] compensation for Coordinated Omission
- Open Source
  - On github, released to the public domain, creative commons CC0

©2013 Azul Systems, Inc.

HdrHistogram

- Fixed cost in both space and time
  - Built with “latency sensitive” applications in mind
  - Recording values does not allocate or grow any data structures
  - Recording values uses a fixed computation to determine location (no searches, no variability in recording cost, FAST)
  - Even iterating through histogram can be done with no allocation
- Internals work like a “floating point” data structure
  - “Exponent” and “Mantissa”
  - Exponent determines “Mantissa bucket” to use
  - “Mantissa buckets” provide linear value range for a given exponent. Each have enough linear entries to support required precision

©2013 Azul Systems, Inc.

HdrHistogram

- Provides tools for iteration
  - Linear, Logarithmic, Percentile
- Supports percentile iterators
  - Practical due to high dynamic range
- Convenient percentile output
  - 10% intervals between 0 and 50% 5% intervals between 50% and 75% 2.5% intervals between 75% and 87.5%...
  - Very useful for feeding percentile distribution graphs...

```
Value, Percentile, TotalCountIncludingThisValue

    0.003 0.000000000000          7
    0.057 0.100000000000     807222
    0.058 0.200000000000    1235747
    0.059 0.300000000000    1694413
    0.060 0.400000000000    1994719
    0.062 0.500000000000    2373326
    0.064 0.550000000000    2620309
    0.066 0.600000000000    2795011
    0.070 0.650000000000    3036116
    1.280 0.700000000000    3228296
    5.552 0.750000000000    3458862
    7.712 0.775000000000    3574491
    9.856 0.800000000000    3689655
   12.016 0.825000000000    3805210
   14.176 0.850000000000    3920746
   16.320 0.875000000000    4036366
   17.408 0.887500000000    4094471
   18.464 0.900000000000    4150910
   19.584 0.912500000000    4209006
   20.832 0.925000000000    4267165
   22.208 0.937500000000    4324157
   22.976 0.943750000000    4352952
   23.808 0.950000000000    4381652
   24.736 0.956250000000    4410732
   25.760 0.962500000000    4439554
   26.880 0.968750000000    4467918
   27.488 0.971875000000    4482272
   28.160 0.975000000000    4496805
   28.896 0.978125000000    4511389
   29.696 0.981250000000    4525422
   30.656 0.984375000000    4539989
   31.200 0.985937500000    4547261
   31.776 0.987500000000    4554465
   32.384 0.989062500000    4561828
   33.088 0.990625000000    4569070
```

©2012 Azul Systems, Inc.

HdrHistogram

Latency by Percentile Distribution

Latency (msec)

160
140
120
100
80
60
40
20
0

Max=137.472

0% 90% 99% 99.9% 99.99% 99.999% 99.9999%

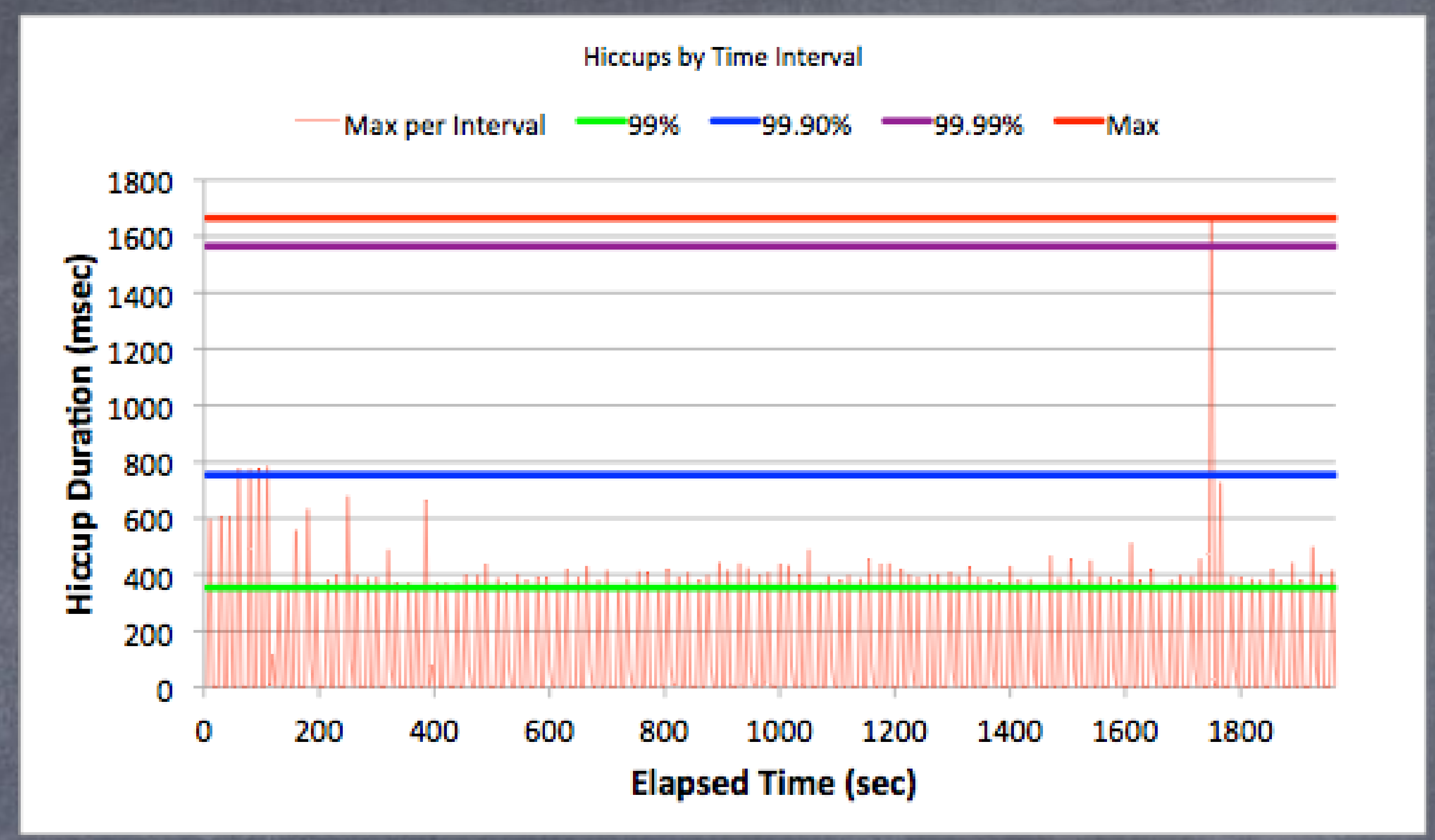
Percentile

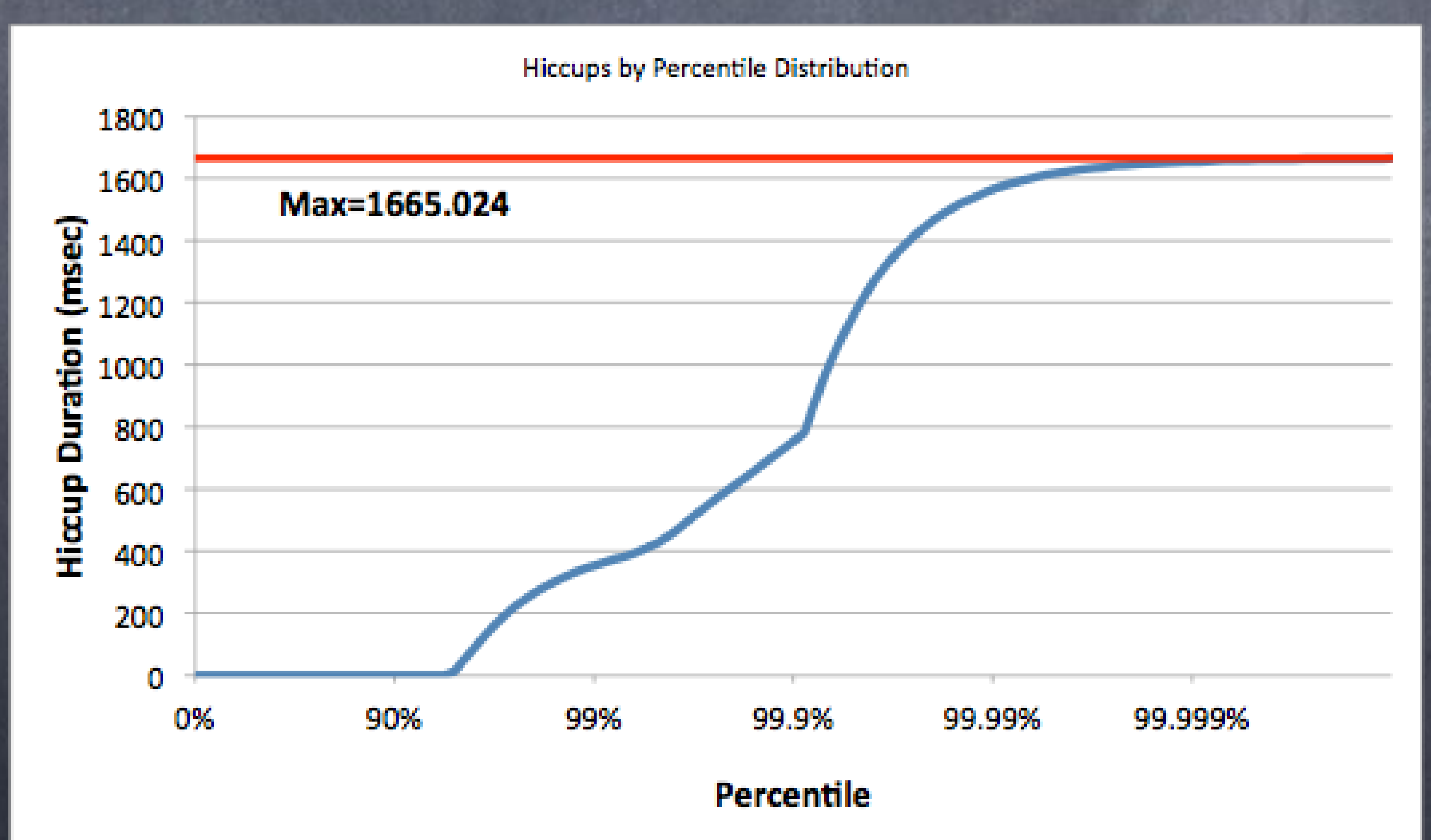
©2012 Azul Systems, Inc.

jHiccup

©2013 Azul Systems, Inc.

Incontinuities in Java platform execution





©2012 Azul Systems, Inc.

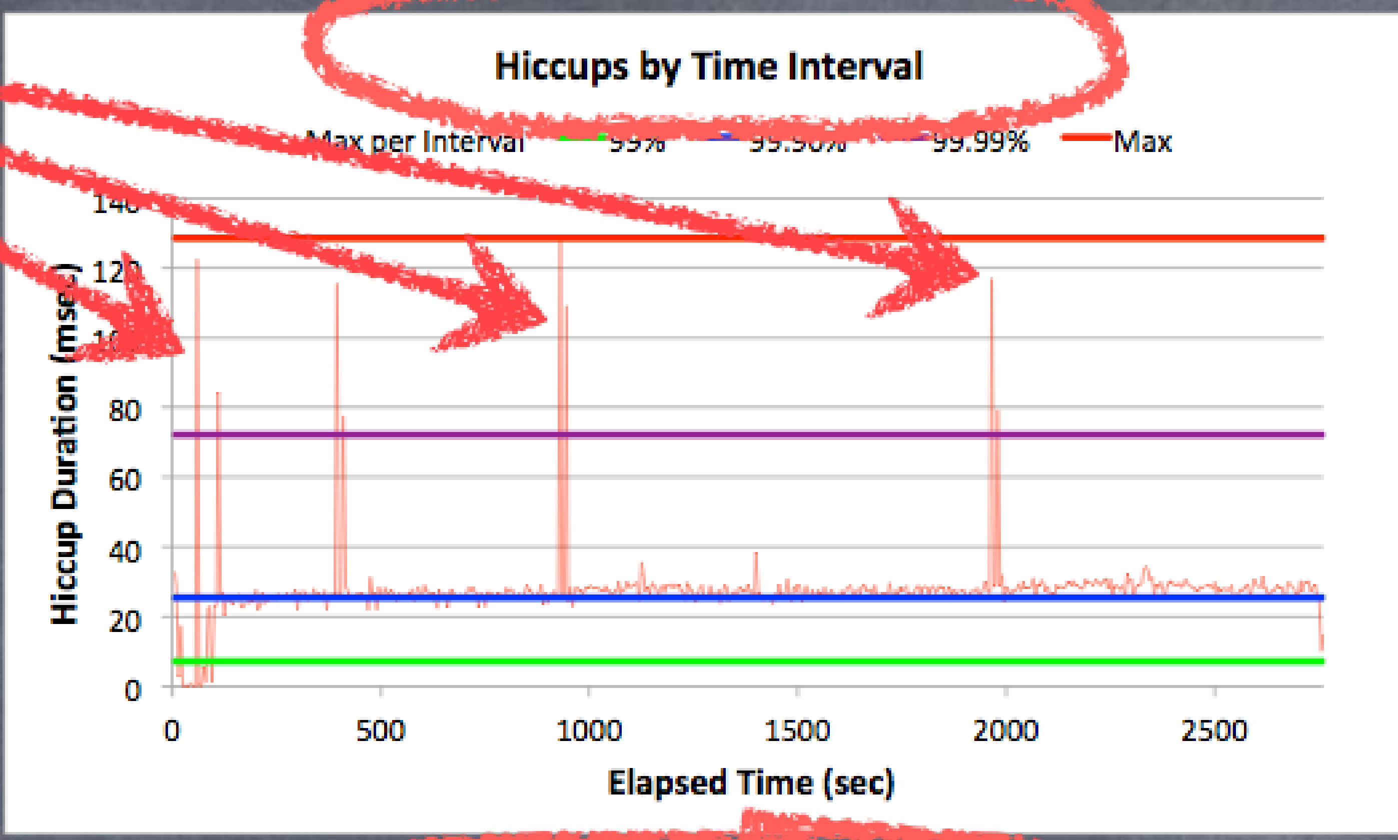
jHiccup

- A tool for capturing and displaying platform hiccups
  - Records any observed non-continuity of the underlying platform
  - Plots results in simple, consistent format
- Simple, non-intrusive
  - As simple as adding jHiccup.jar as a java agent:
    - % java -javaagent=jHiccup.jar myApp myflags
  - or attaching jHiccup to a running process:
    - % jHiccup -p <pid>
  - Adds a background thread that samples time @ 1000/sec into an HdrHistogram
- Open Source. Released to the public domain

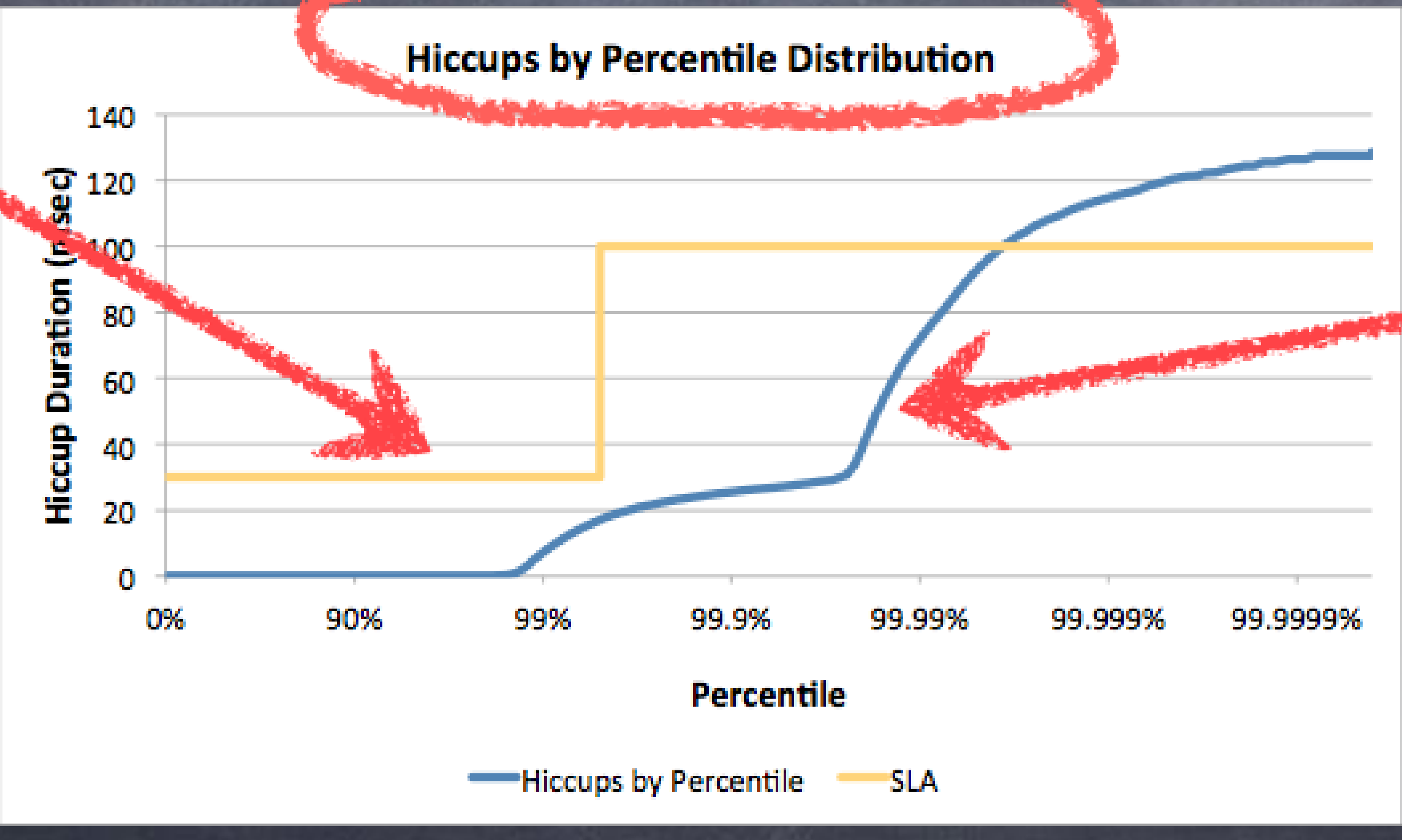
©2013 Azul Systems, Inc.

Telco App Example

Max Time per interval



Optional SLA plotting



Hiccup duration at percentile levels

©2012 Azul Systems, Inc.

Fun with jHiccup

Charles Nutter @headius 20 Jan

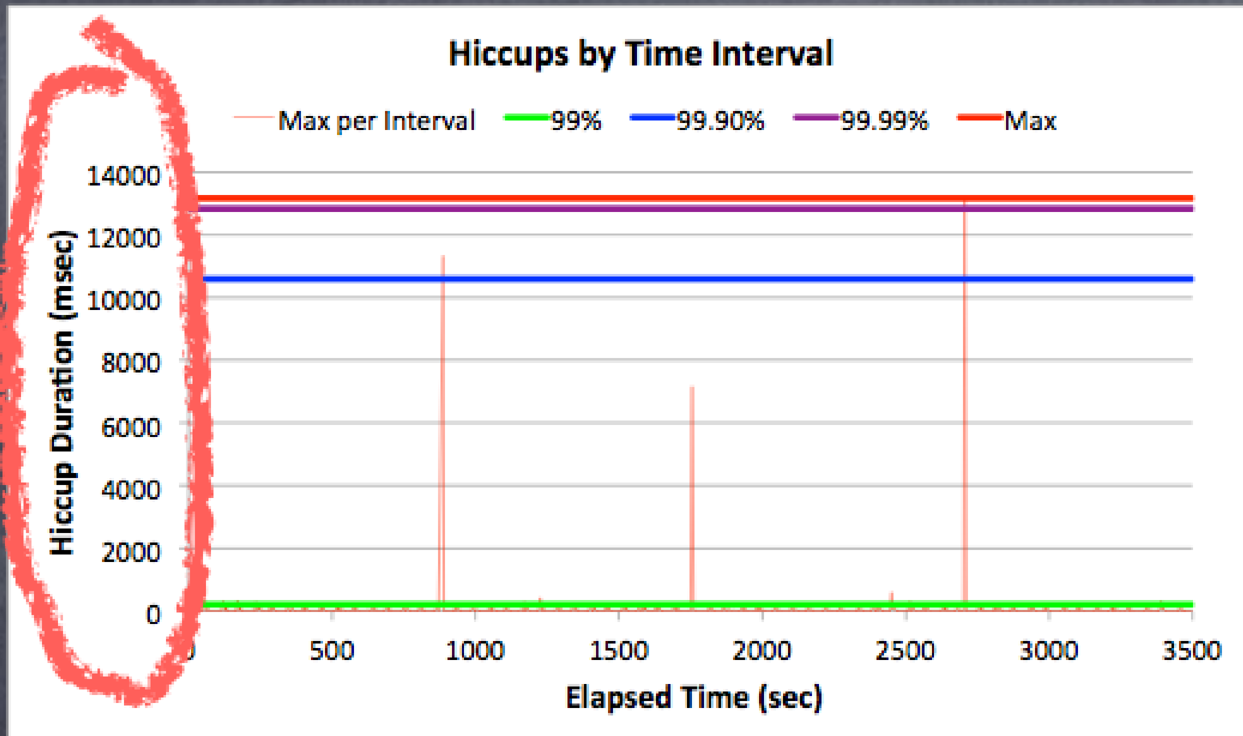
jHiccup, @AzulSystems' free tool to show you why your JVM sucks compared to Zing: bit.ly/wsH5A8 (thx @bascule)

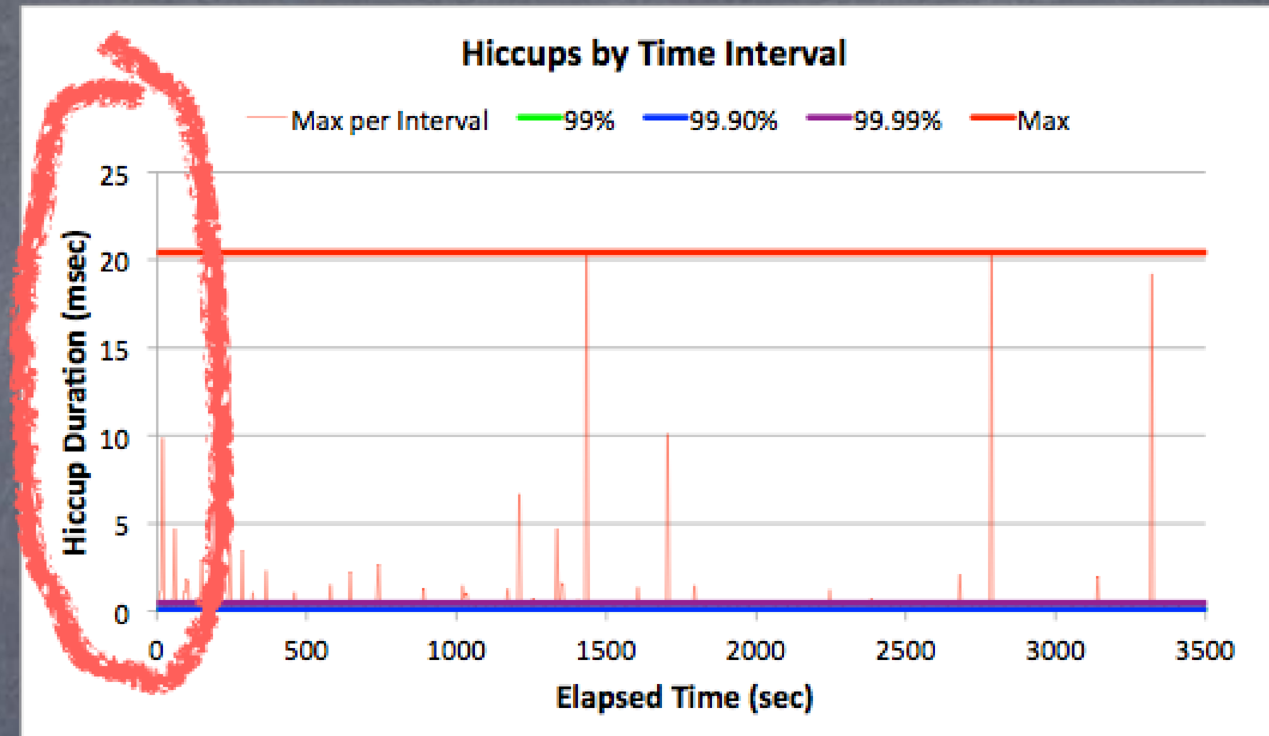
Retweeted by Gil Tene

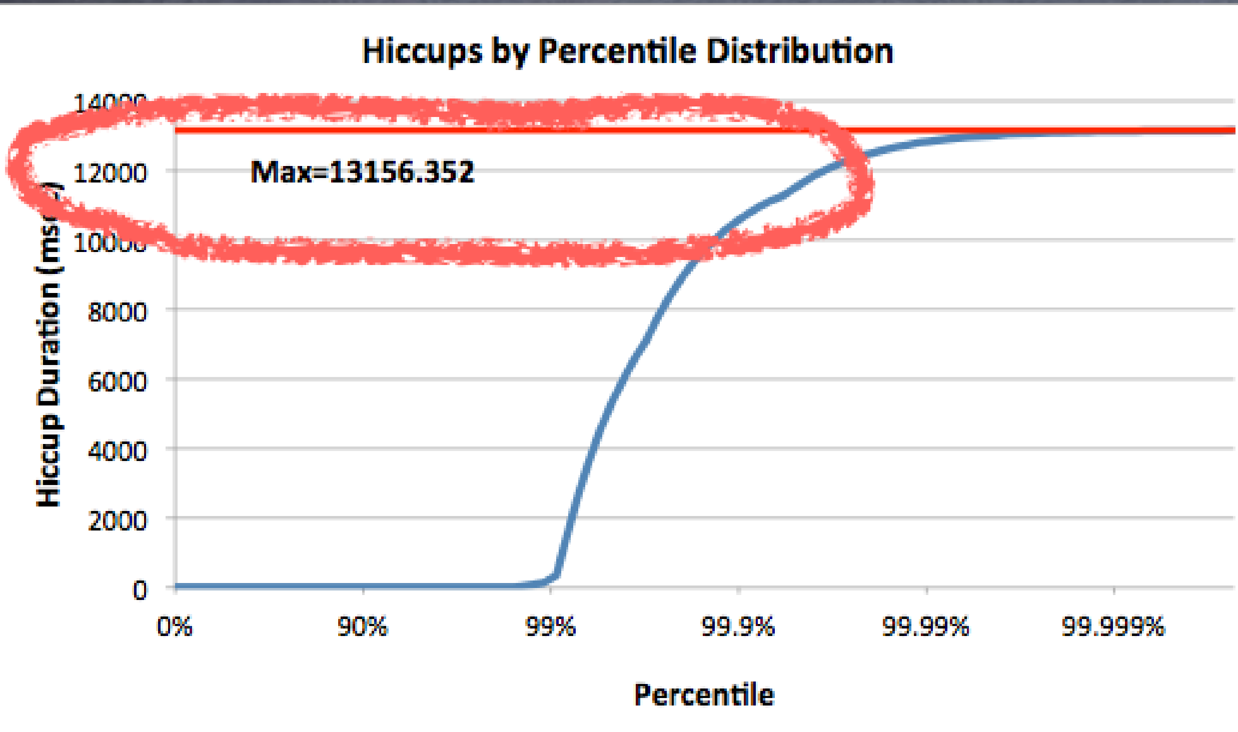
©2012 Azul Systems, Inc.

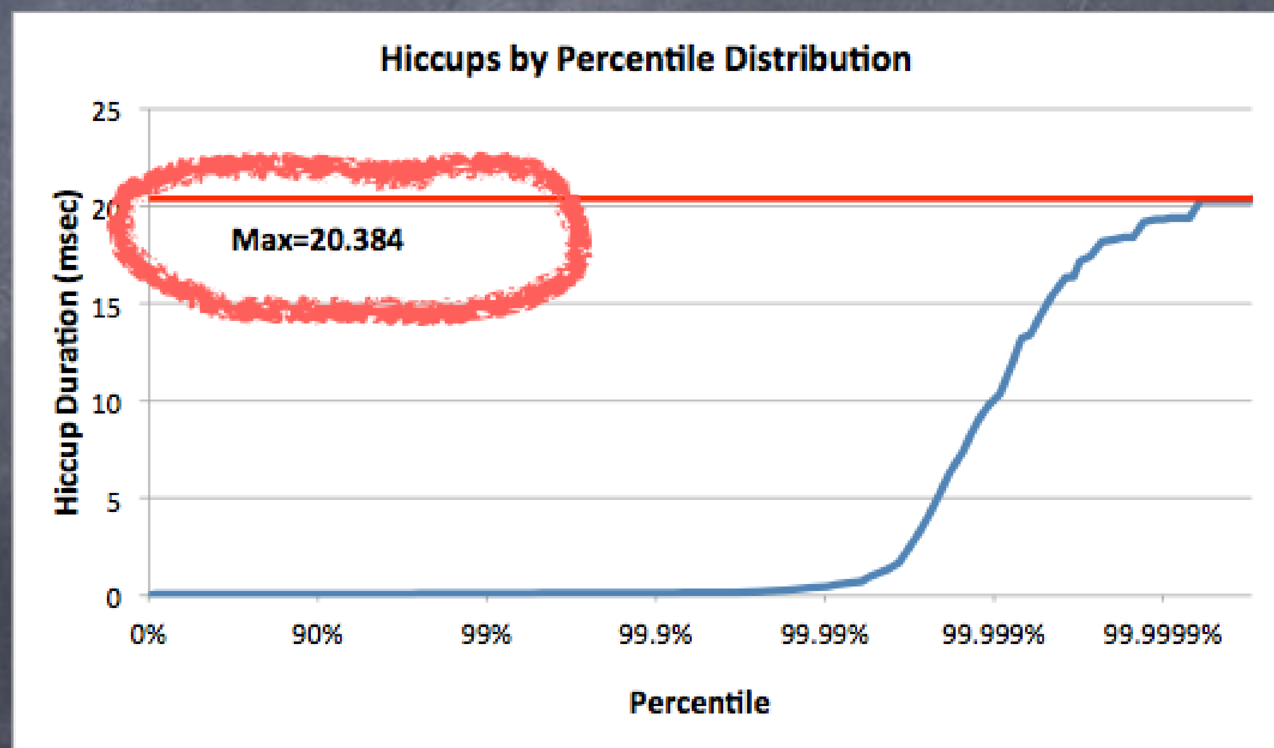
Oracle HotSpot CMS, 1GB in an 8GB heap

Zing 5, 1GB in an 8GB heap





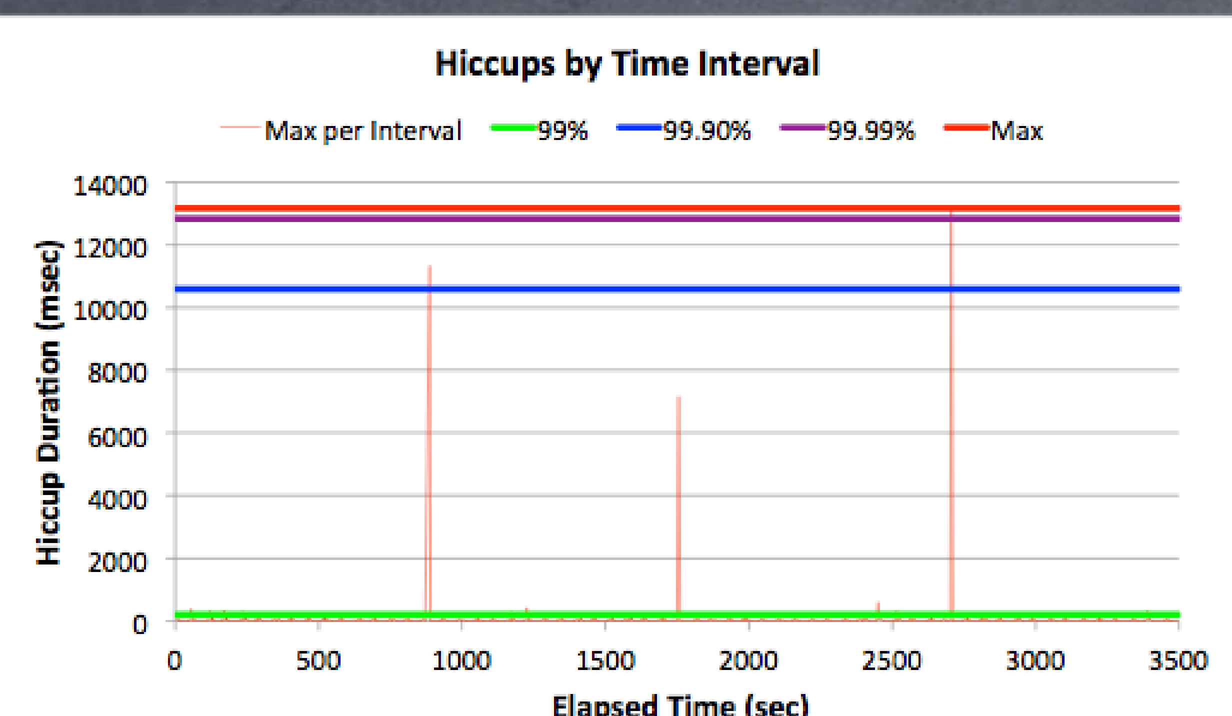


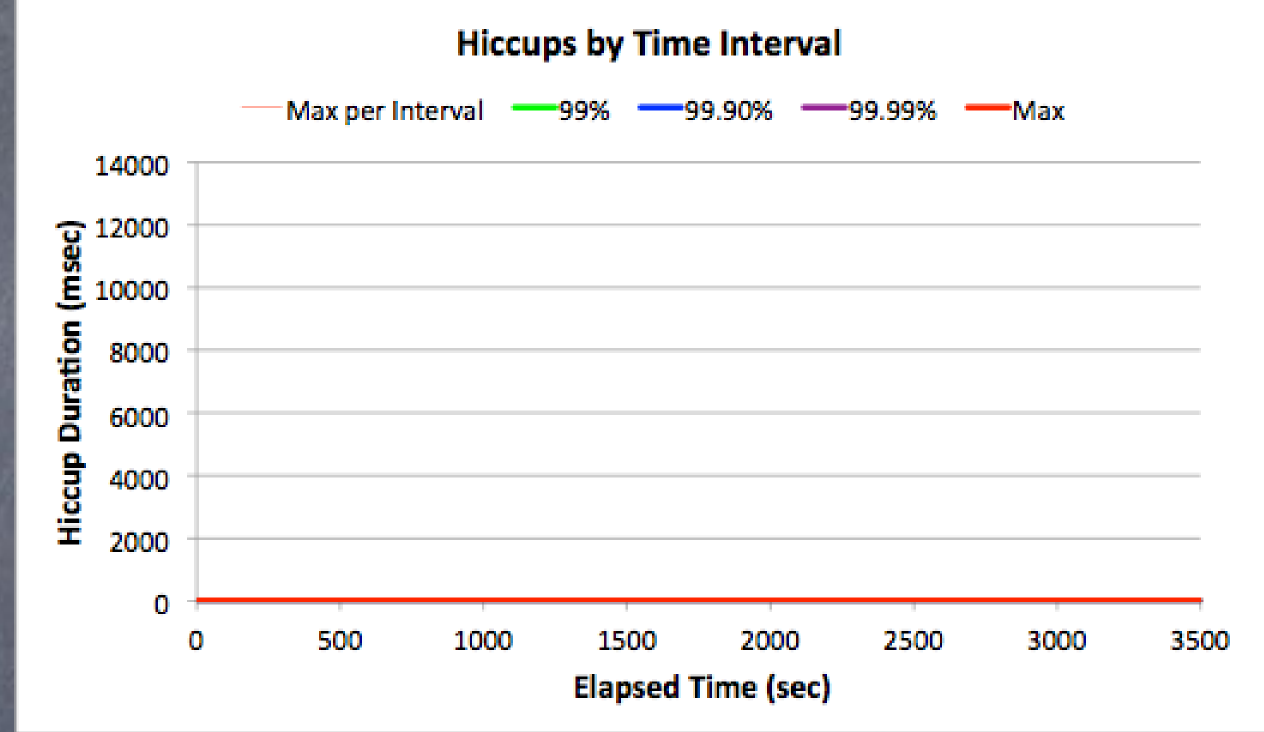


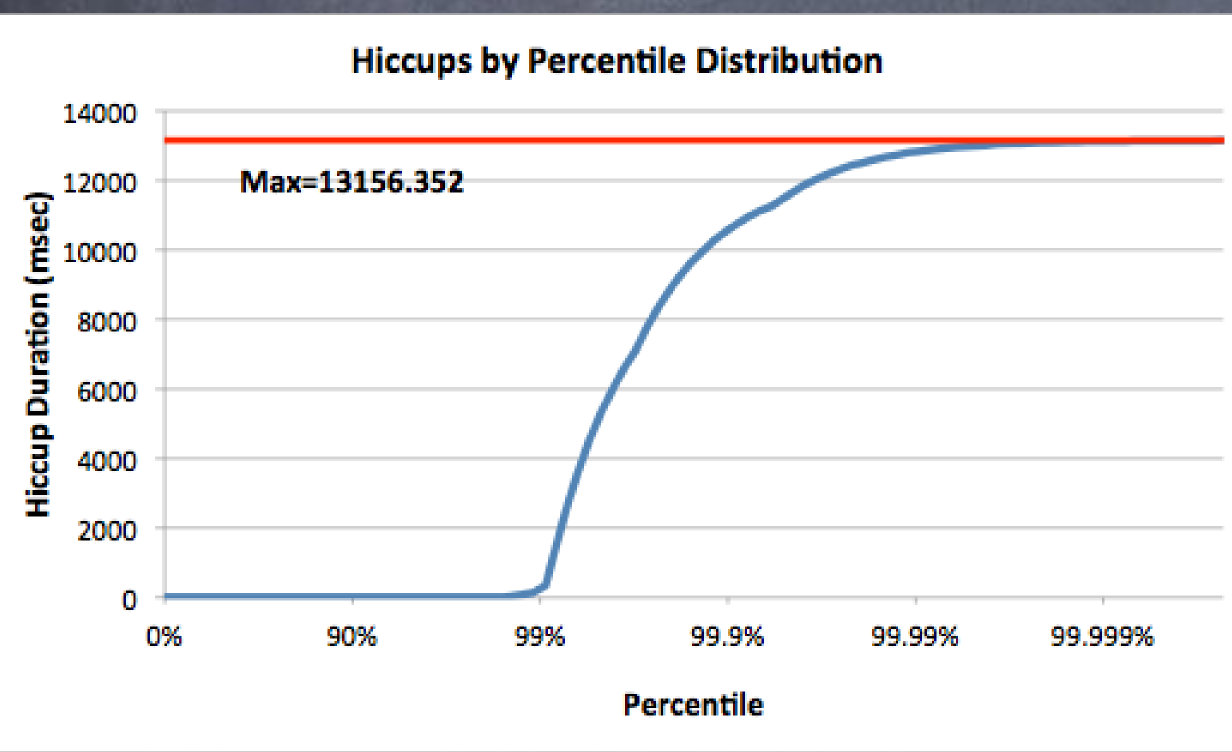
©2012 Azul Systems, Inc.

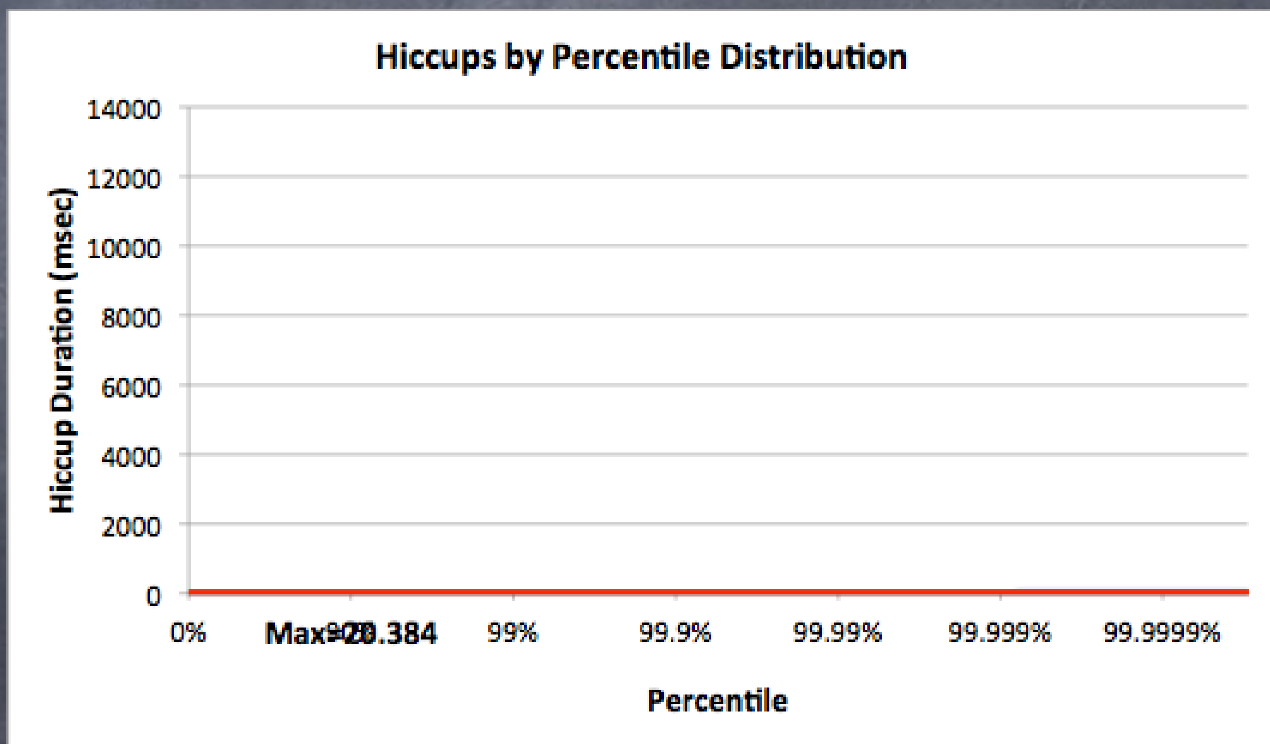
Oracle HotSpot CMS, 1GB in an 8GB heap

Zing 5, 1GB in an 8GB heap









©2012 Azul Systems, Inc.

Good for both

"squishy" real time
(human response times)

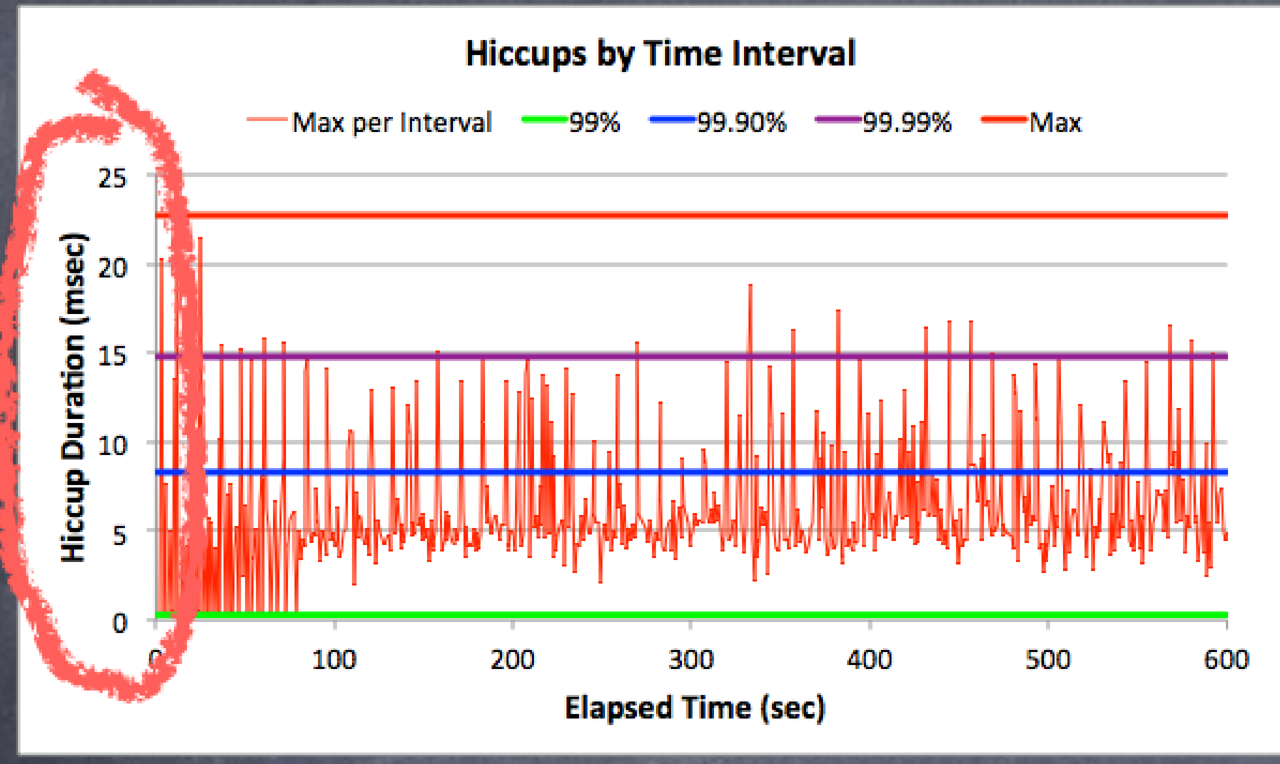
and

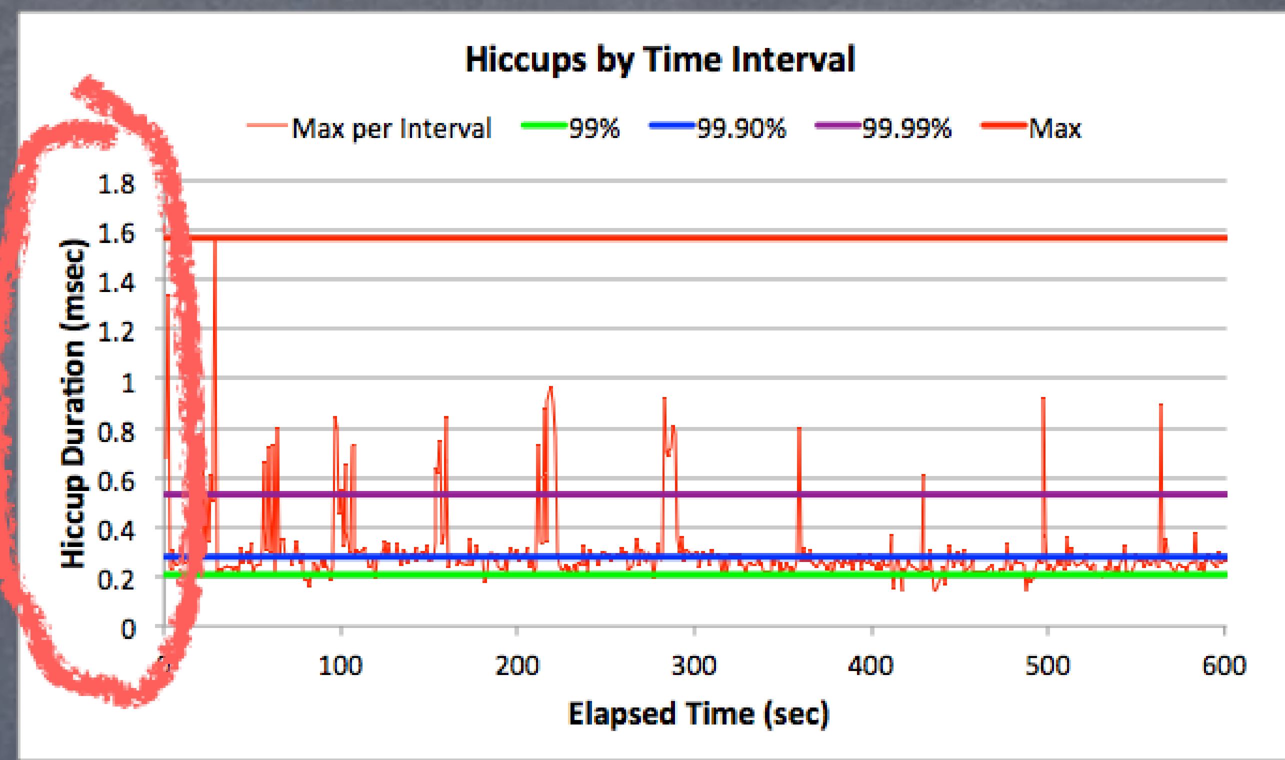
"soft" real time
(low latency software systems)

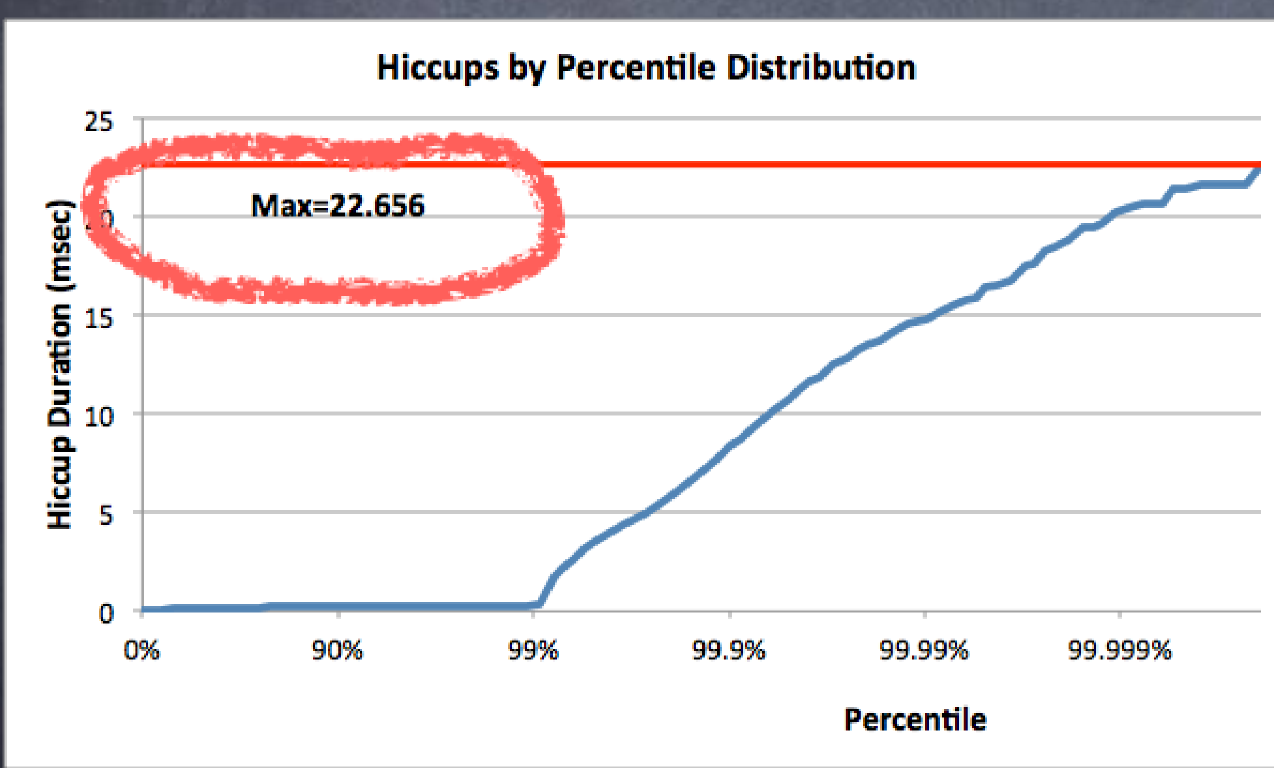
©2012 Azul Systems, Inc.

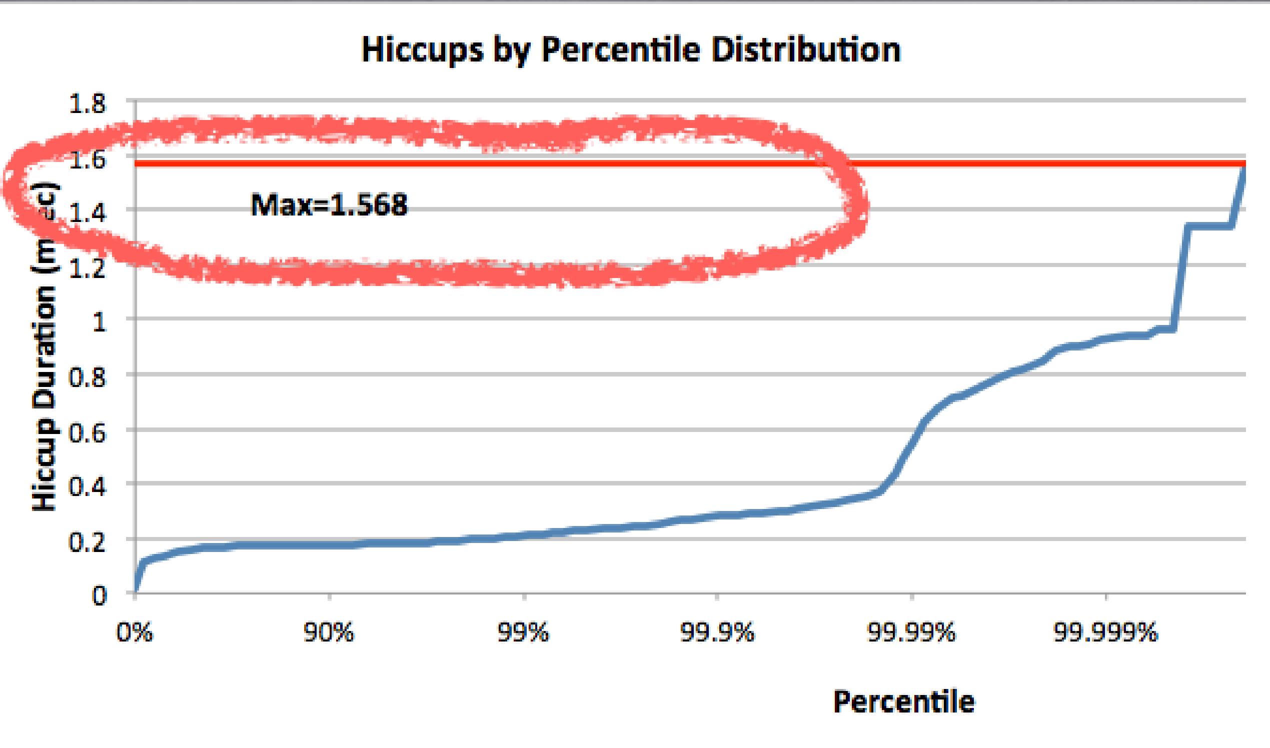
Oracle HotSpot (pure newgen)

Zing





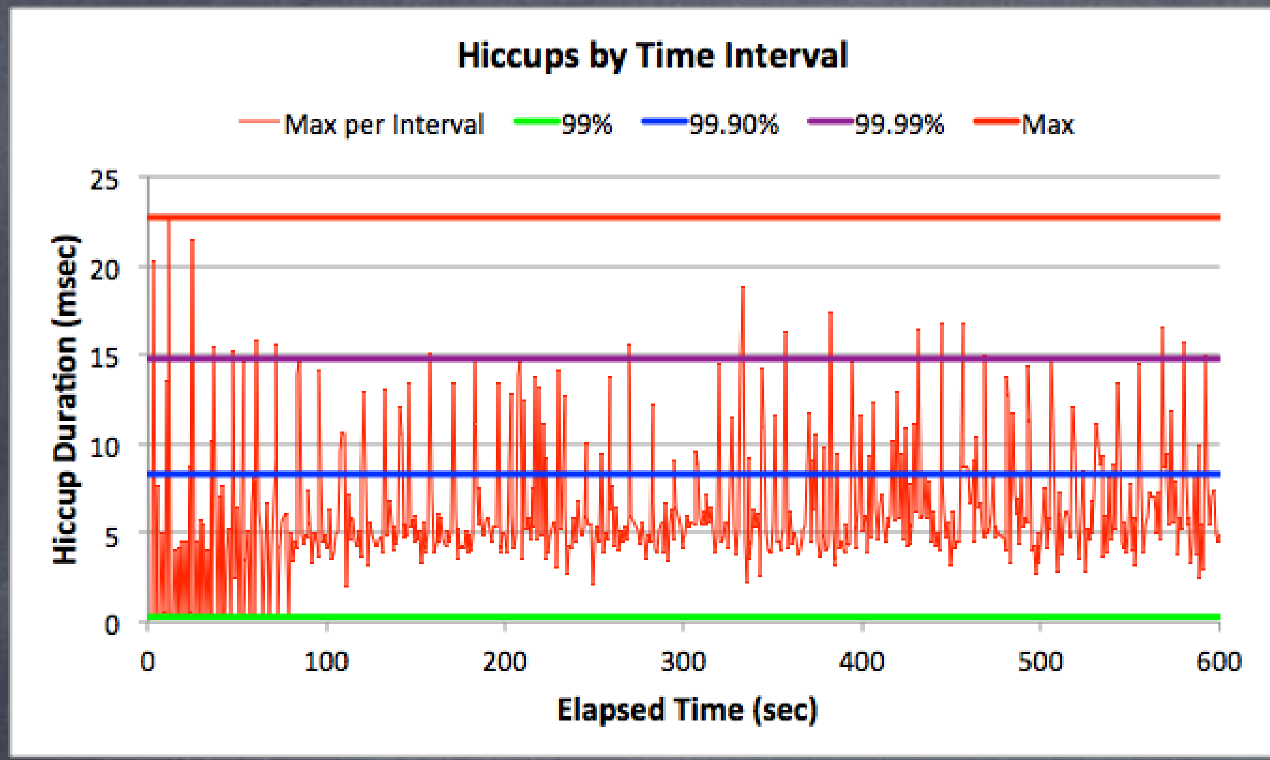


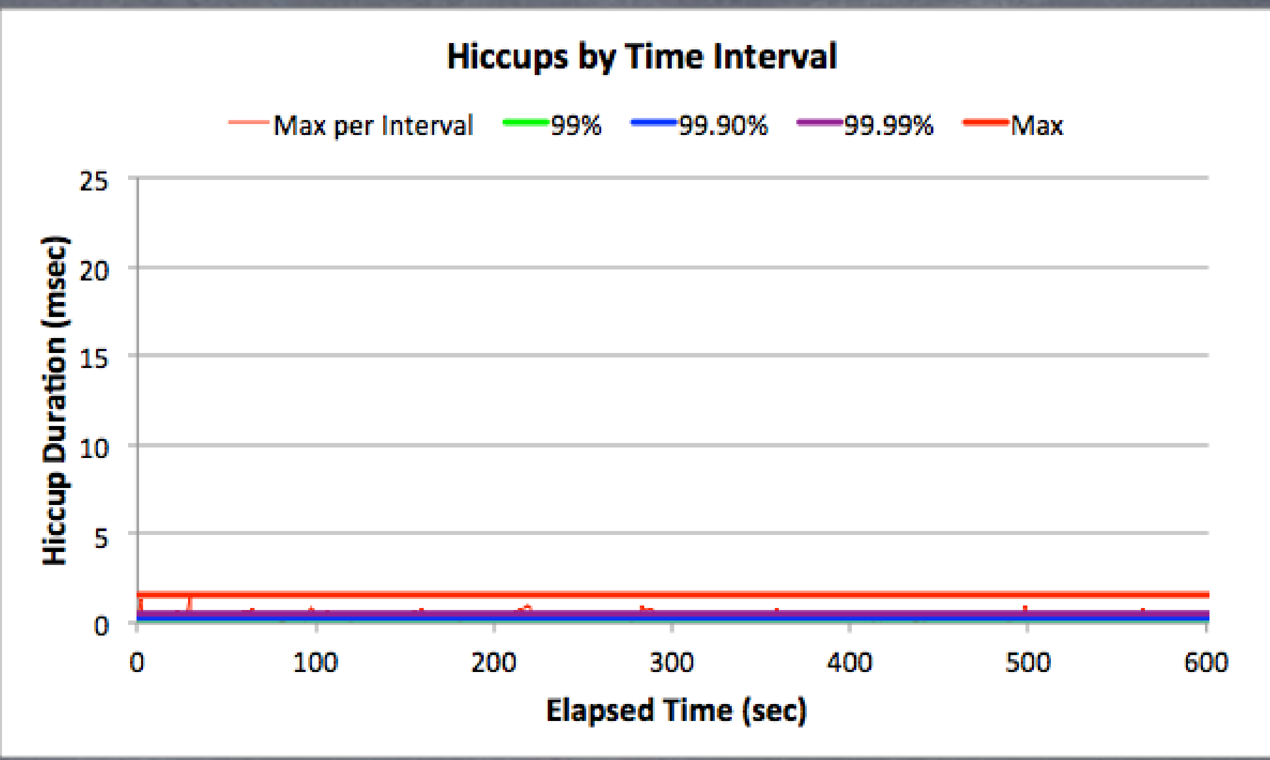


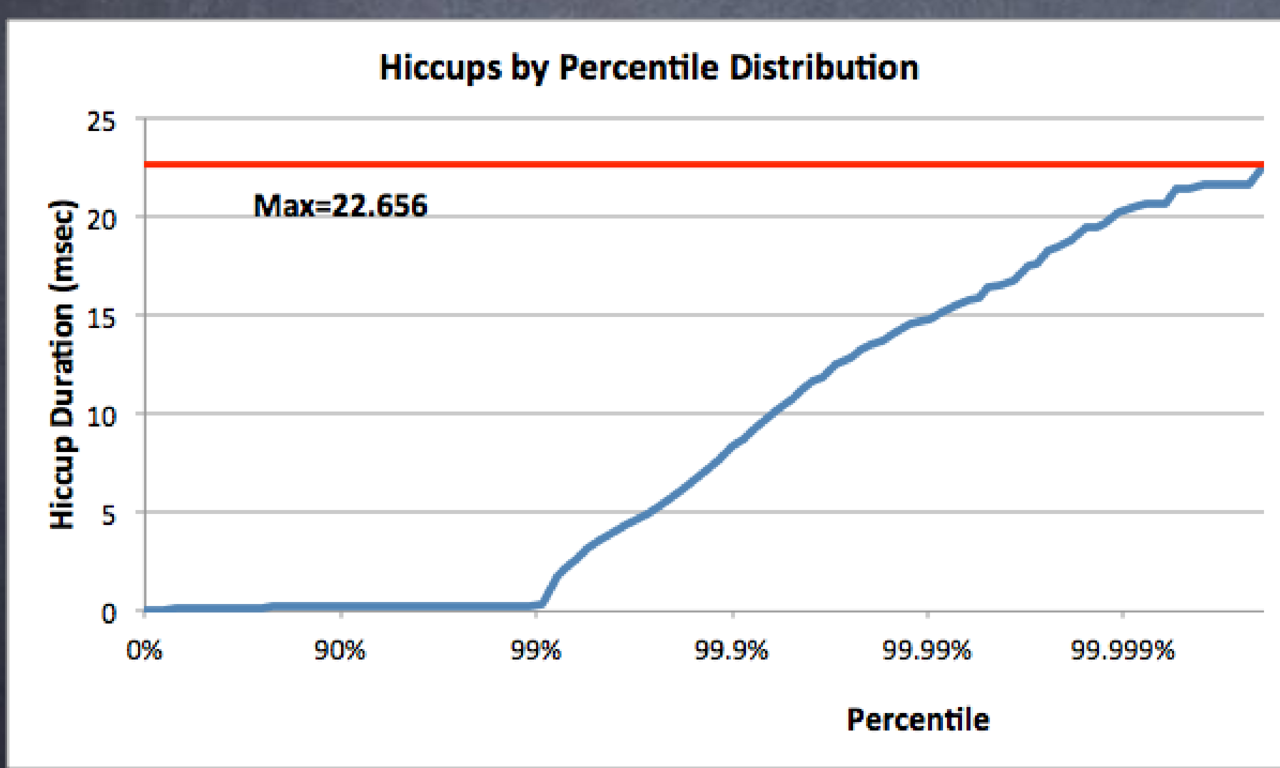
Low latency trading application

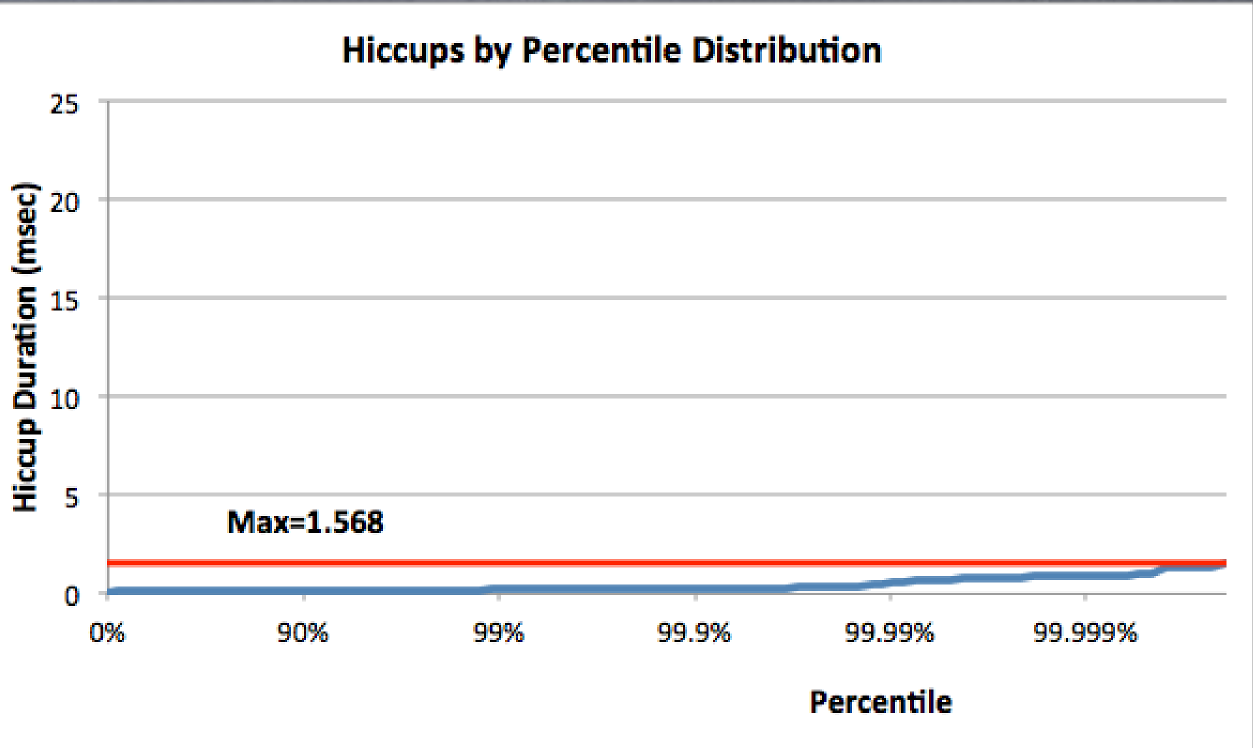
©2012 Azul Systems, Inc.

Oracle HotSpot (pure newgen) Zing









Low latency - Drawn to scale

©2012 Azul Systems, Inc.

Shameless bragging

©2013 Azul Systems, Inc.

Zing

- A JVM for Linux/x86 servers
- ELIMINATES Garbage Collection as a concern for enterprise applications
- Very wide operating range: Used in both low latency and large scale enterprise application spaces
- Decouples scale metrics from response time concerns
    - Transaction rate, data set size, concurrent users, heap size, allocation rate, mutation rate, etc.
- Leverages elastic memory for resilient operation

©2013 Azul Systems, Inc.

What is Zing good for?

- If you have a server-based Java application
- And you are running on Linux
- And you use using more than ~300MB of memory
- Then Zing will likely deliver superior behavior metrics

©2013 Azul Systems, Inc.

Where Zing shines

- Low latency
  - Eliminate behavior blips down to the sub-millisecond-units level
- Machine-to-machine “stuff”
  - Support higher *sustainable* throughput (the one that meets SLAs)
- Human response times
  - Eliminate user-annoying response time blips. Multi-second and even fraction-of-a-second blips will be completely gone.
  - Support larger memory JVMs *if needed* (e.g. larger virtual user counts, or larger cache, in-memory state, or consolidating multiple instances)
- “Large” data and in-memory analytics
  - Make batch stuff “business real time”. Gain super-efficiencies.

©2013 Azul Systems, Inc.

Takeaways

- Standard Deviation and application latency should never show up on the same page...
- If you haven't stated percentiles and a Max, you haven't specified your requirements
- Measuring throughput without latency behavior is [usually] meaningless
- Mistakes in measurement/analysis can cause orders-of-magnitude errors and lead to bad business decisions
- jHiccup and HdrHistogram are pretty useful
- The Zing JVM is cool...

©2013 Azul Systems, Inc.

Q & A

http://www.azulsystems.com

http://www.jhiccup.com
http://giltene.github.com/HdrHistogram

©2013 Azul Systems, Inc.