How Not to Measure Latency

An attempt to confer wisdom...

Gil Tene, CTO & co-Founder, Azul Systems

AZUL SYSTEMS

This Talk's Purpose / Goals

- This is not a "there is only one right way" talk
- This is a talk about the common pitfalls people run into when measuring latency
- It will hopefully get you to critically examine both WHY and HOW you measure latency
- It will discuss some generic tools that could help
- The "Azul makes the world's best JVM for latency-sensitive applications" stuff will only come towards the end, I promise...

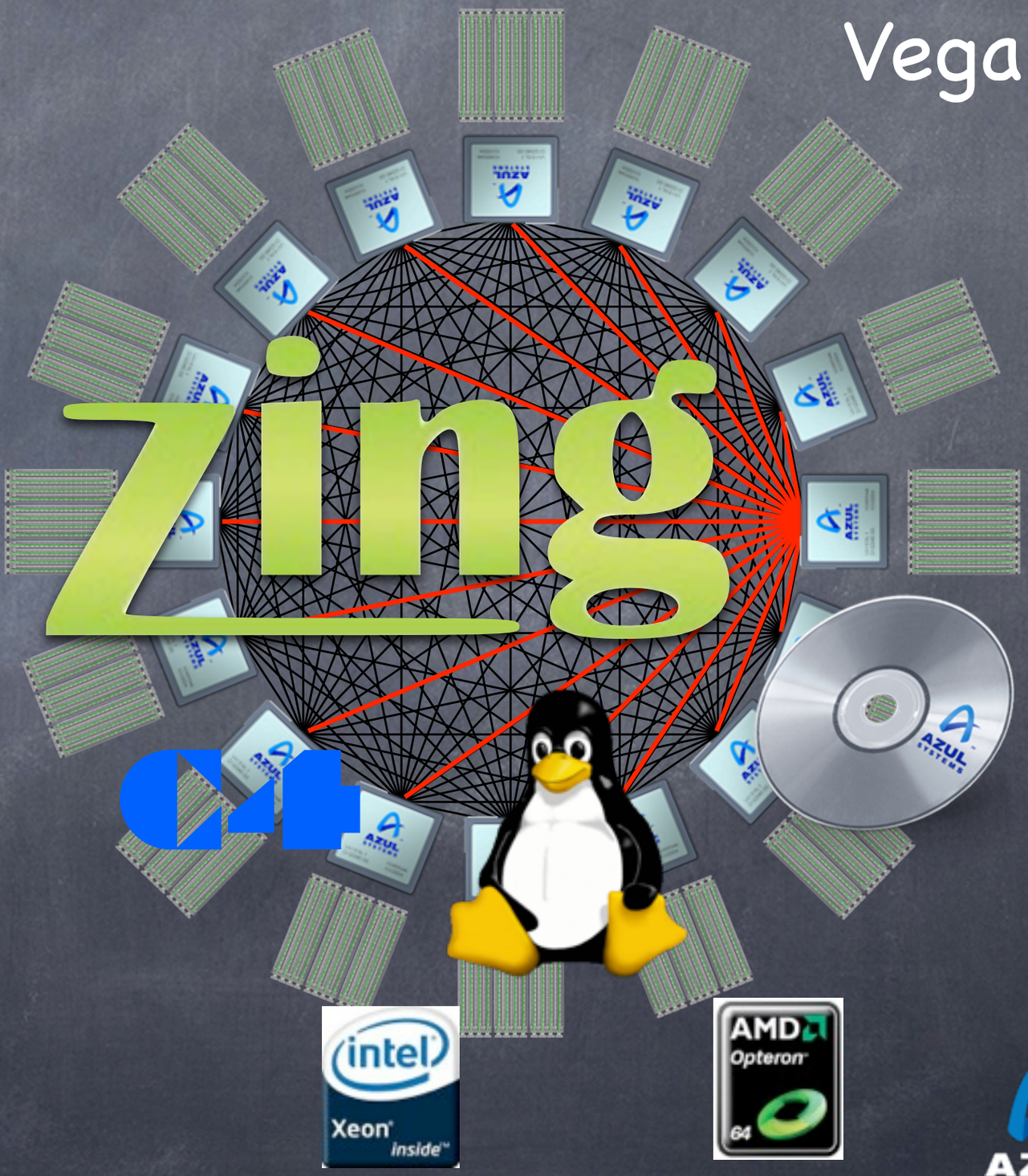
About me: Gil Tene

- co-founder, CTO @Azul Systems
- Have been working on a "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...

* working on real-world trash compaction issues, circa 2004

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)
- **Now Pure software for commodity x86 (Zing)**
- "Industry firsts" in Garbage collection, elastic memory, Java virtualization, memory scale



High level agenda

- Some latency behavior background
- Latency "philosophy" questions
- The pitfalls of using "statistics"
- The Coordinated Omission Problem
- Some useful tools
- Demonstrate what tools can tell us about a latency-friendly JVM...

A classic look at response time behavior

Response time as a function of load

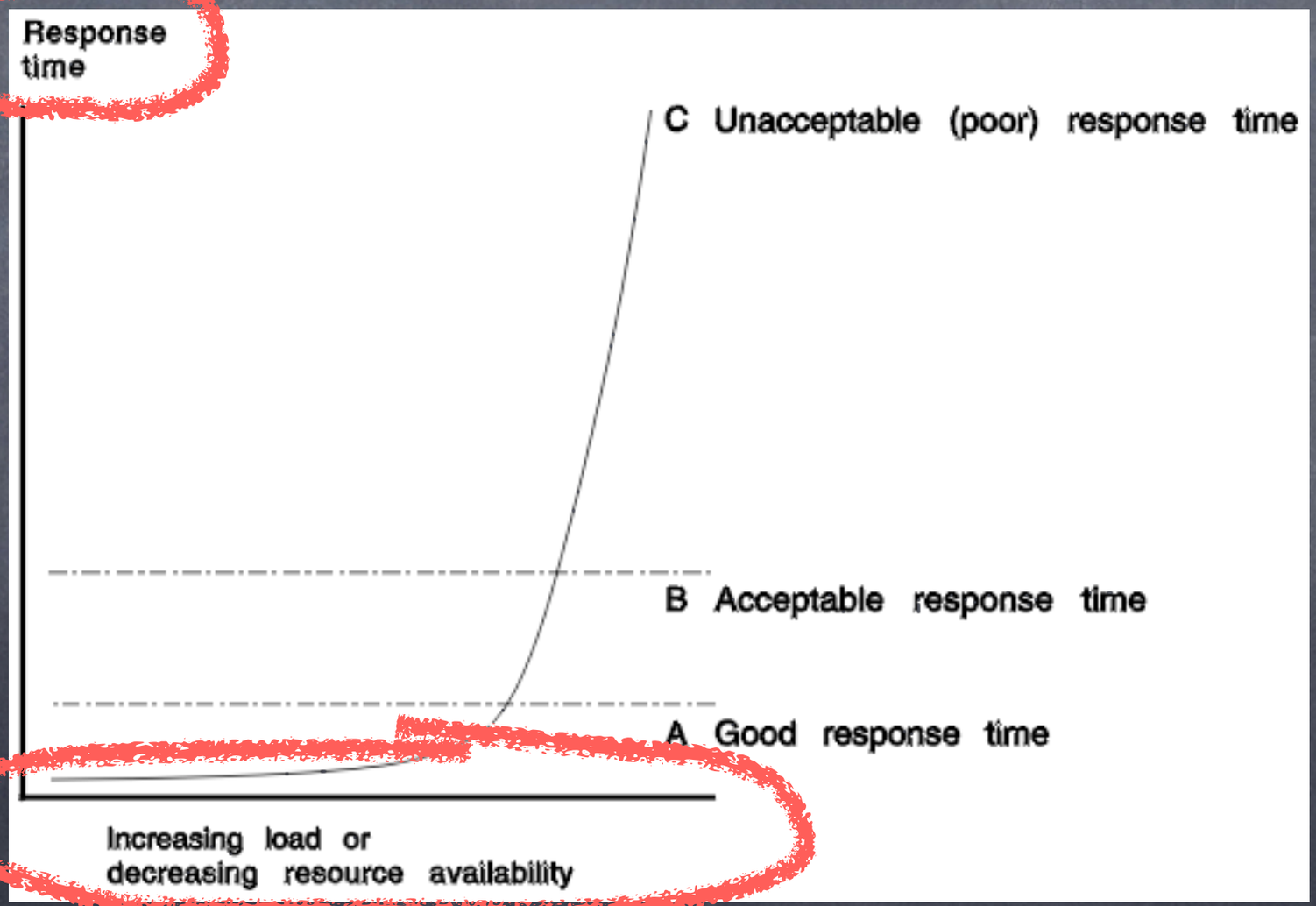
Average?

Max?

Median?

90%?

99.9%

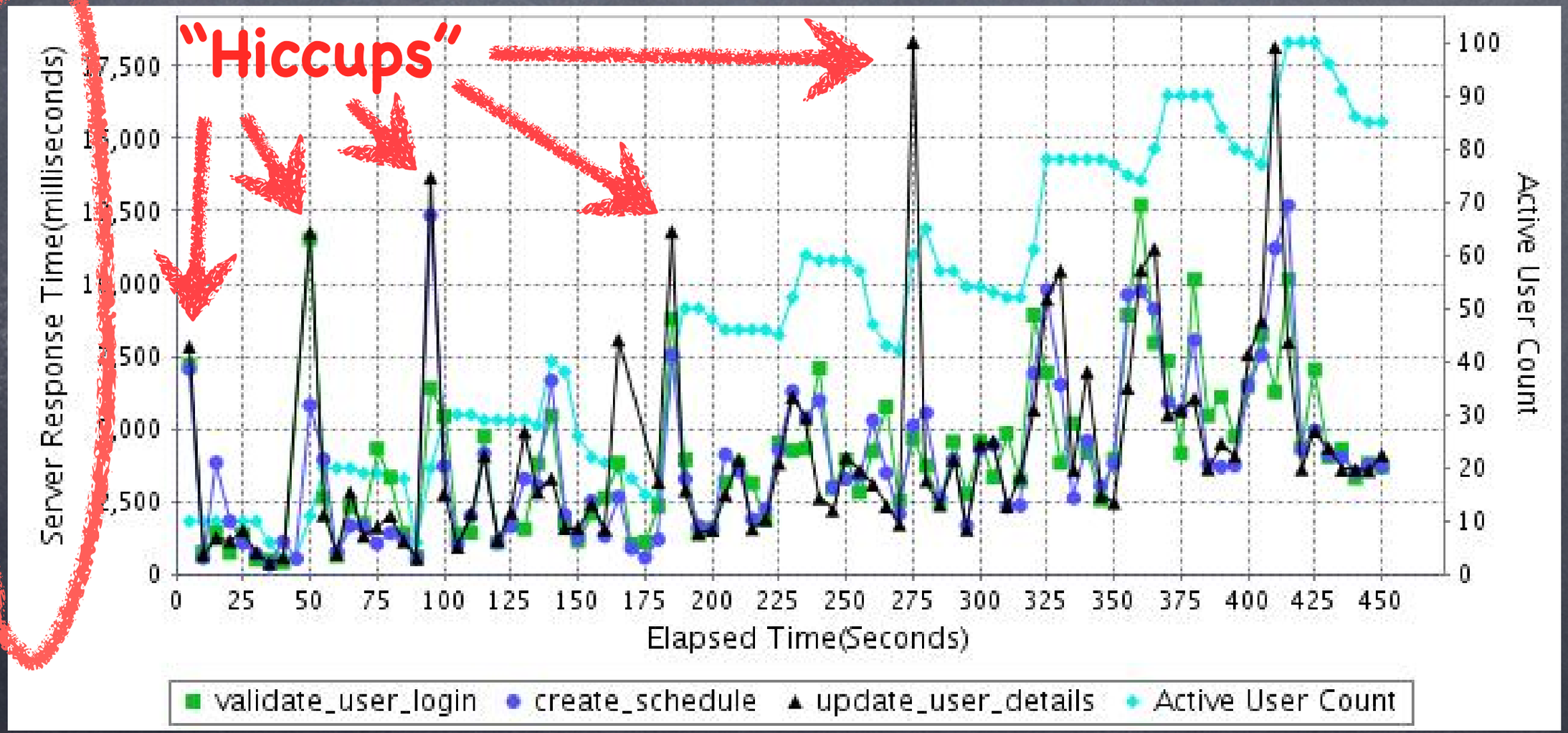


* source: IBM CICS server documentation, "understanding response times"

AZUL SYSTEMS

Response time over time

When we measure behavior over time, we often see:

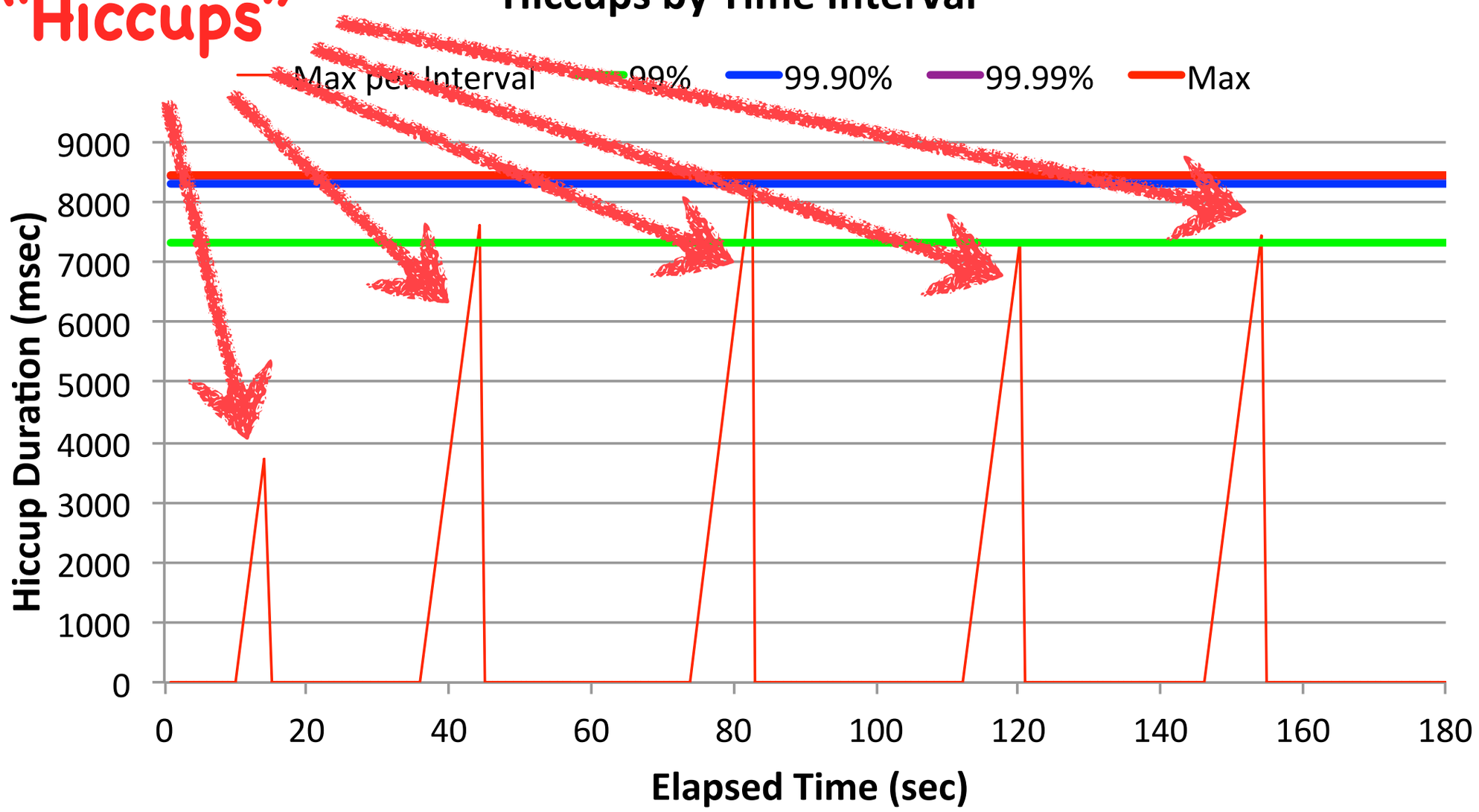


✳ source: ZOHO QEngine White Paper: performance testing report analysis

AZUL SYSTEMS

What happened here?

"Hiccups"



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

AZUL SYSTEMS

Common fallacies

- Computers run application code continuously
- Response time can be measured as work units/time
- Response time exhibits a normal distribution
- "Glitches" or "Semi-random omissions" in measurement don't have a big effect.

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", …
- ….

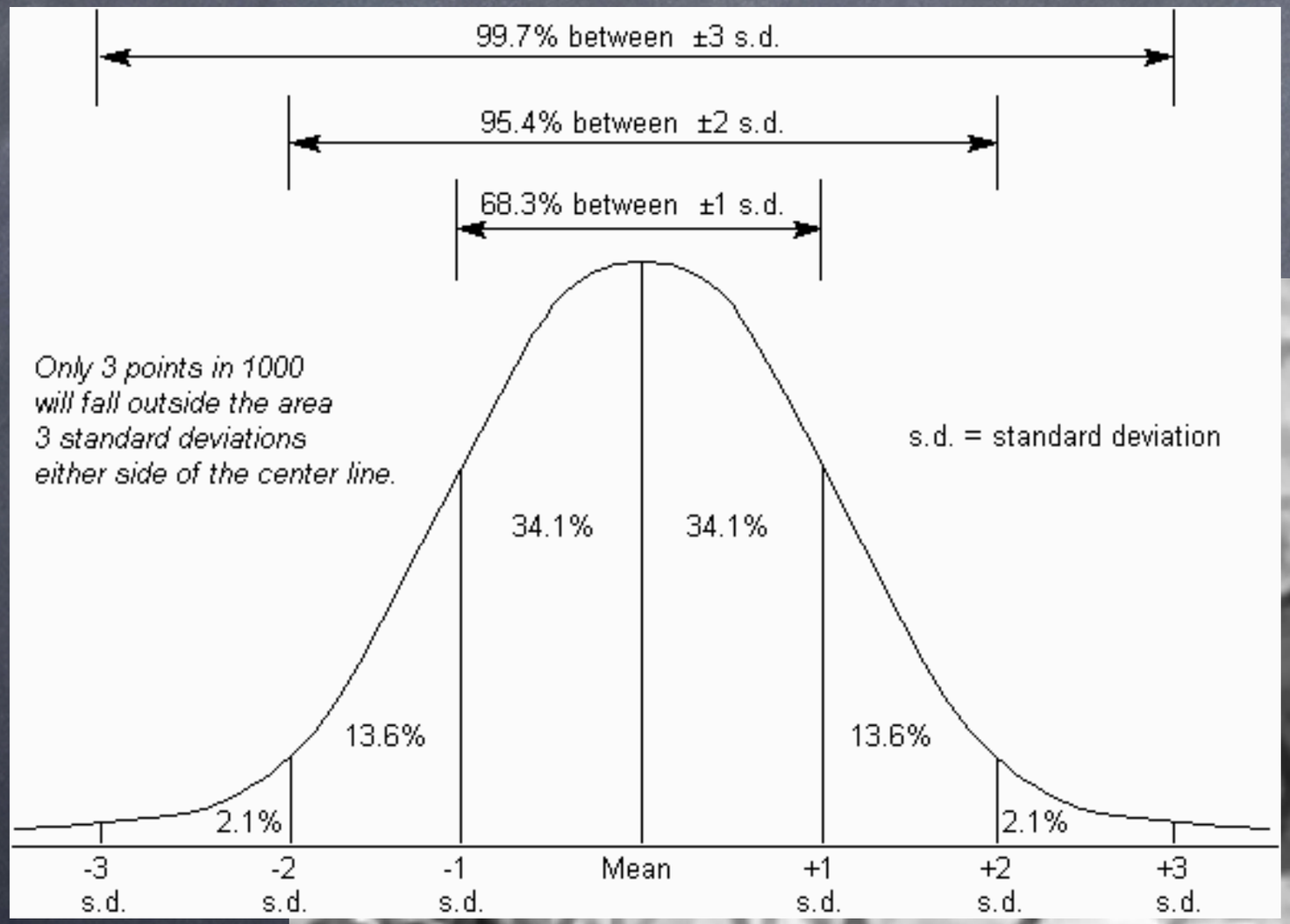
AZUL SYSTEMS

Common ways people deal with hiccups

AZUL SYSTEMS®

Common ways people deal with hiccups

Averages and Standard Deviation

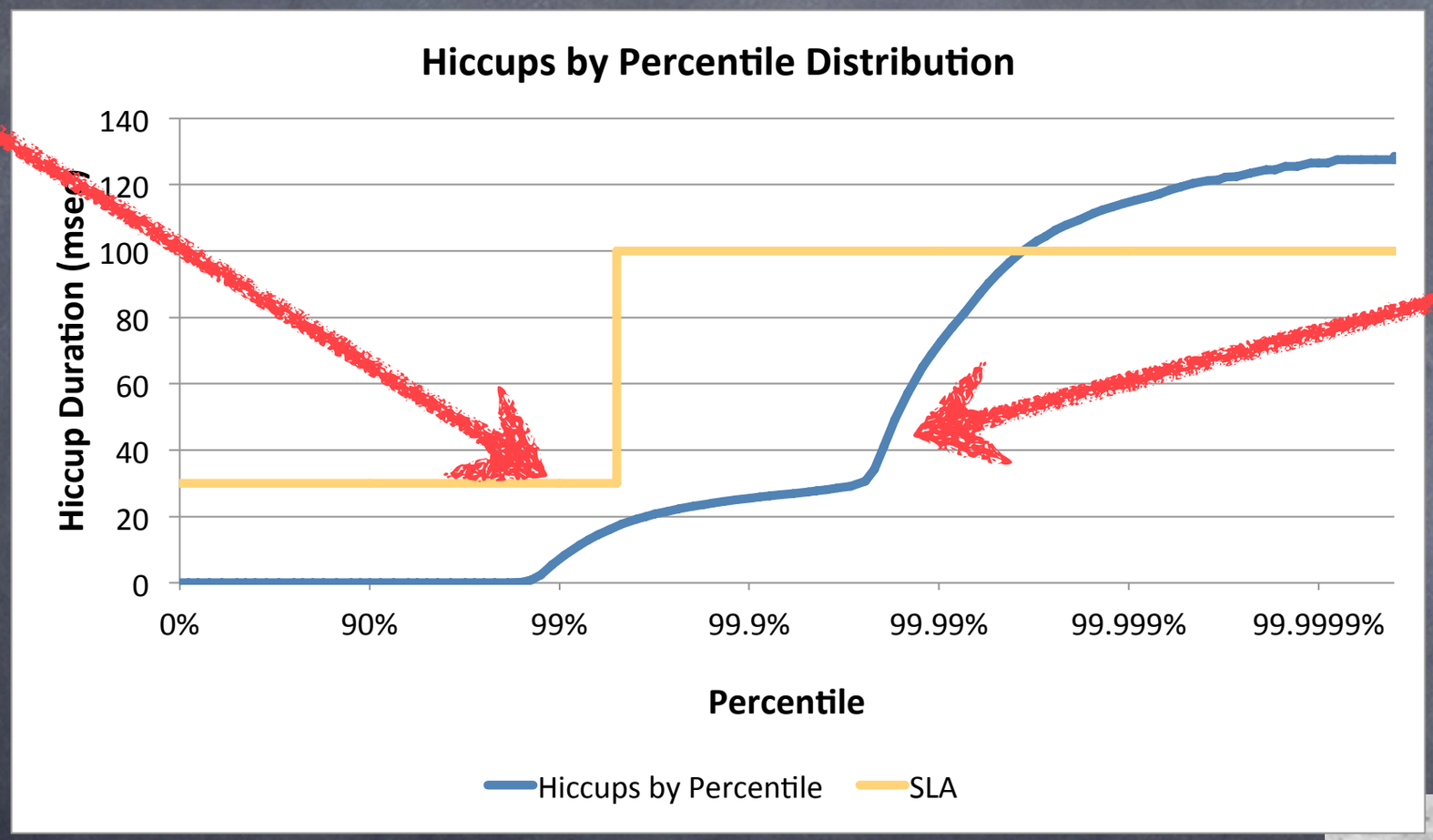


AZUL SYSTEMS

Better ways people deal with hiccups

Actually measuring percentiles

SLA



Response Time Percentile plot line

AZUL SYSTEMS

Requirements

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

AZUL SYSTEMS®

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

AZUL SYSTEMS

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

AZUL SYSTEMS

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 very reassuring
- Having a good average and a nice standard deviation don't matter or help
- The worst case is all that matters

AZUL SYSTEMS

"Low Latency" Trading

aka "soft" real time

- Goal: Be fast enough to make some good plays
- Goal: 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 30msec stale information
- So we want a very good "typical" (median, 50%'ile)
- But we also need a reasonable Max, or 99.99%'ile

AZUL SYSTEMS

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 0.5sec, 99% should be below 2 seconds, 99.9 should be better than 5 seconds. And a >10 sec. response should never happen.
- Remember: A single user may have 100 interactions per session...

Establishing Requirements

an interactive interview (or thought) process

- Q: What are your latency requirements?
- A: We need an avg. response of 20msec
- Q: Ok. Typical/average of 20msec... 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!
- Q: So I'll write down "5 hours worst case..."
- A: No... Make that "nothing worse than 100 msec"
- Q: Are you sure? Even if it's only three times a day?
- A: Ok... Make it "nothing worse than 2 seconds..."

AZUL SYSTEMS

Establishing Requirements

an interactive interview (or thought) process

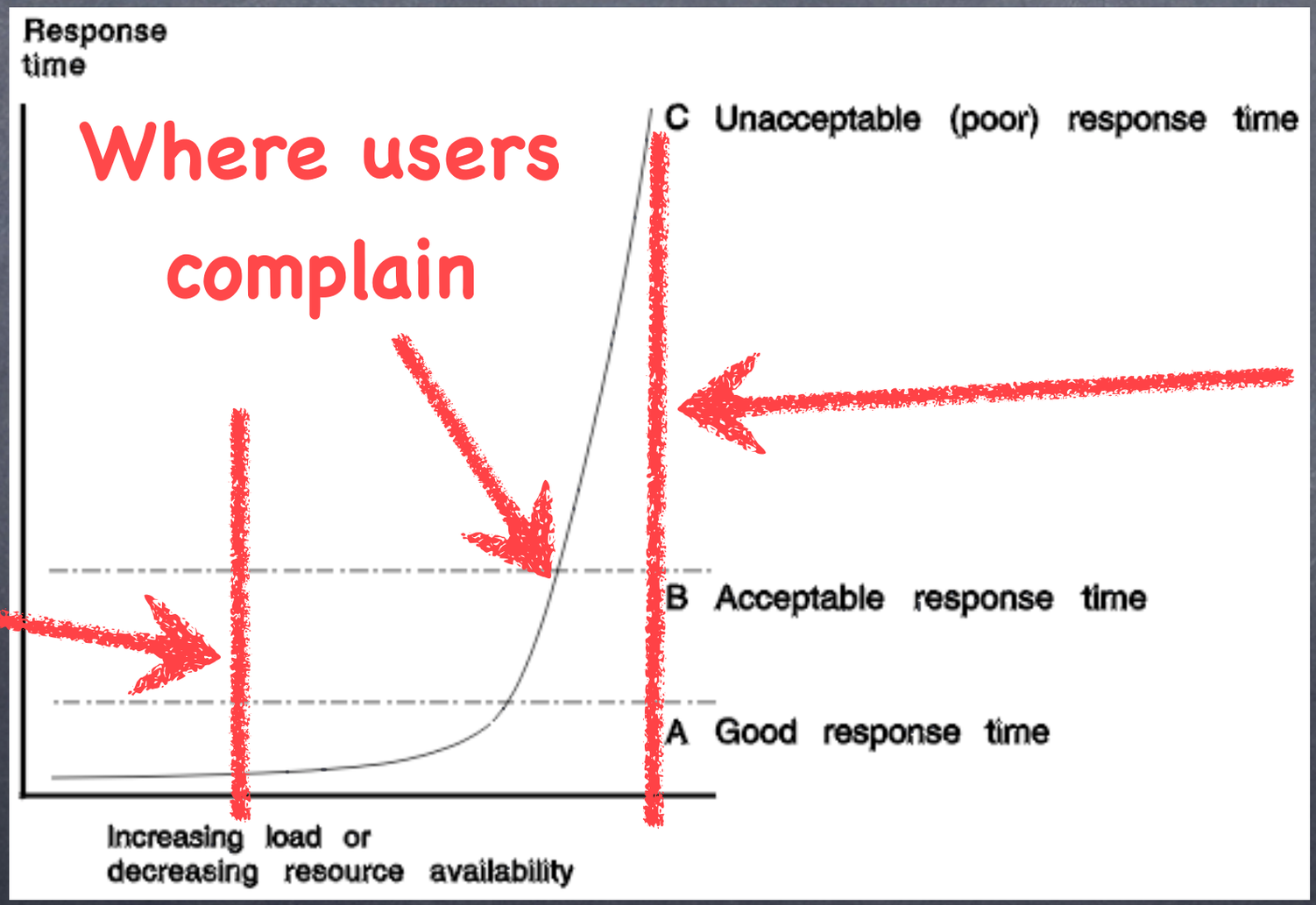
- Ok. So we need a typical of 20msec, and a worst case of 2 seconds. How often is it ok to have a 1 second response?
- A: (Annoyed) I thought you said only a few times a day
- Q: Right. For the worst case. But if half the results are better than 20msec, is it ok for the other half to be just short of 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 50msec 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 500msec 99.9% of the time, or our customers will complain and go elsewhere
- Now we have a service level expectation:
  - 50% better than 20msec
  - 90% better than 50msec
  - 99.9% better than 500msec
  - 100% better than 2 seconds

Latency does not live in a vacuum

AZUL SYSTEMS®

Remember this?

How much load can this system handle?



Where users complain

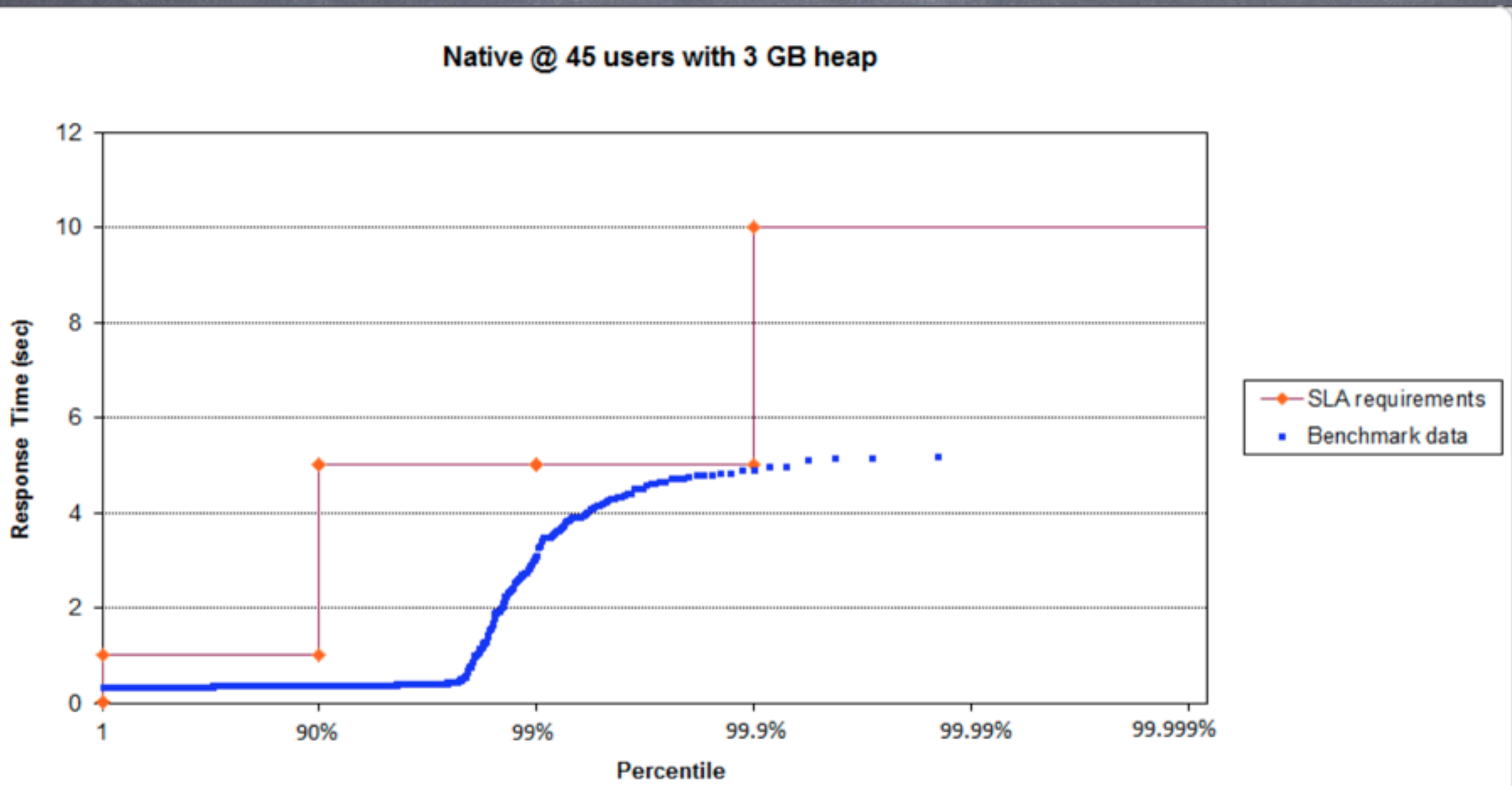
Where the sysadmin is willing to go

What the marketing benchmarks will say

AZUL SYSTEMS

Sustainable Throughput:
The throughput achieved while safely maintaining service levels

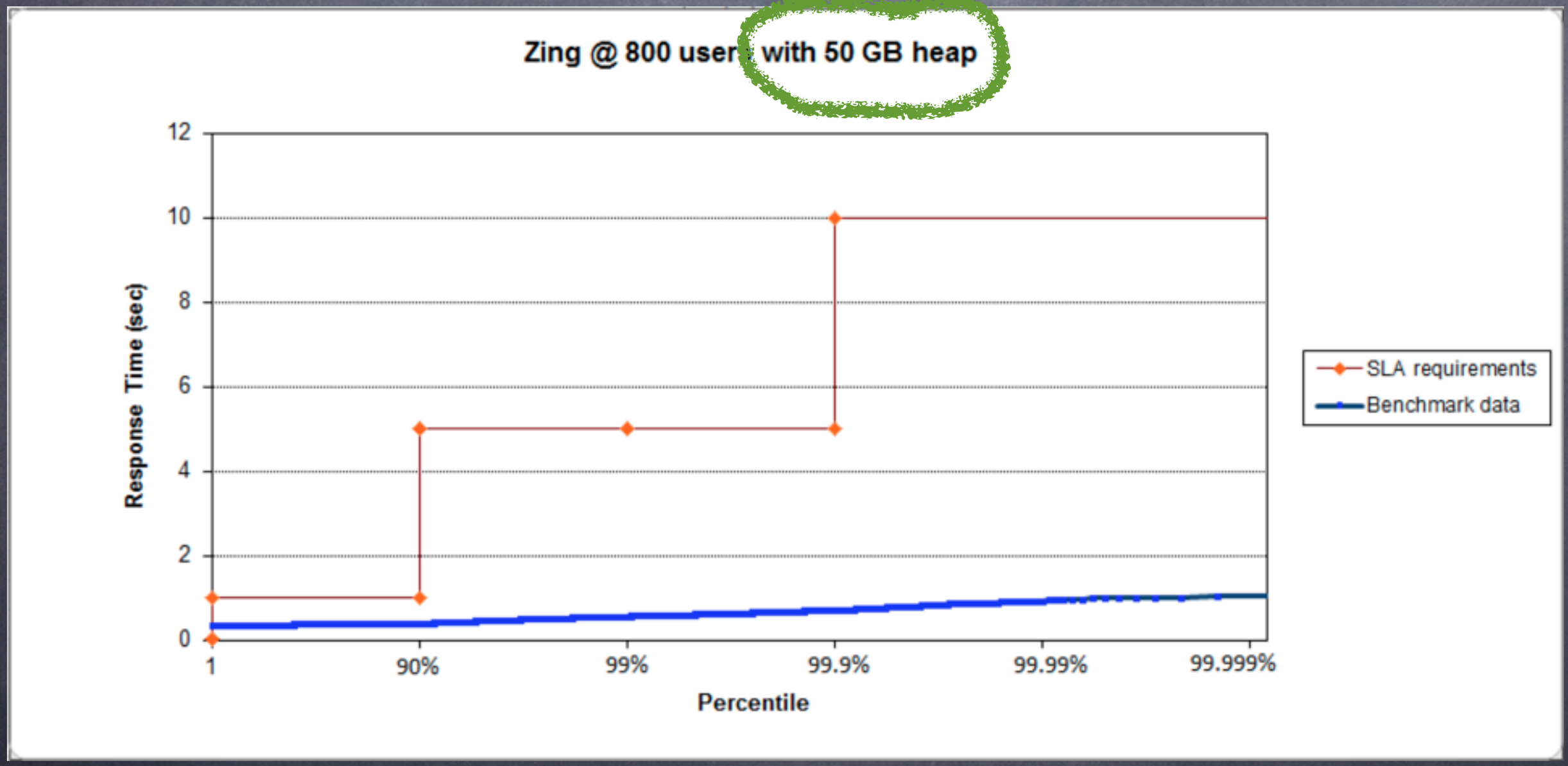
Instance capacity test: "Fat Portal"
HotSpot CMS: Peaks at ~ 3GB / 45 concurrent users



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

AZUL SYSTEMS

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



AZUL SYSTEMS

The coordinated omission problem

An accidental conspiracy...

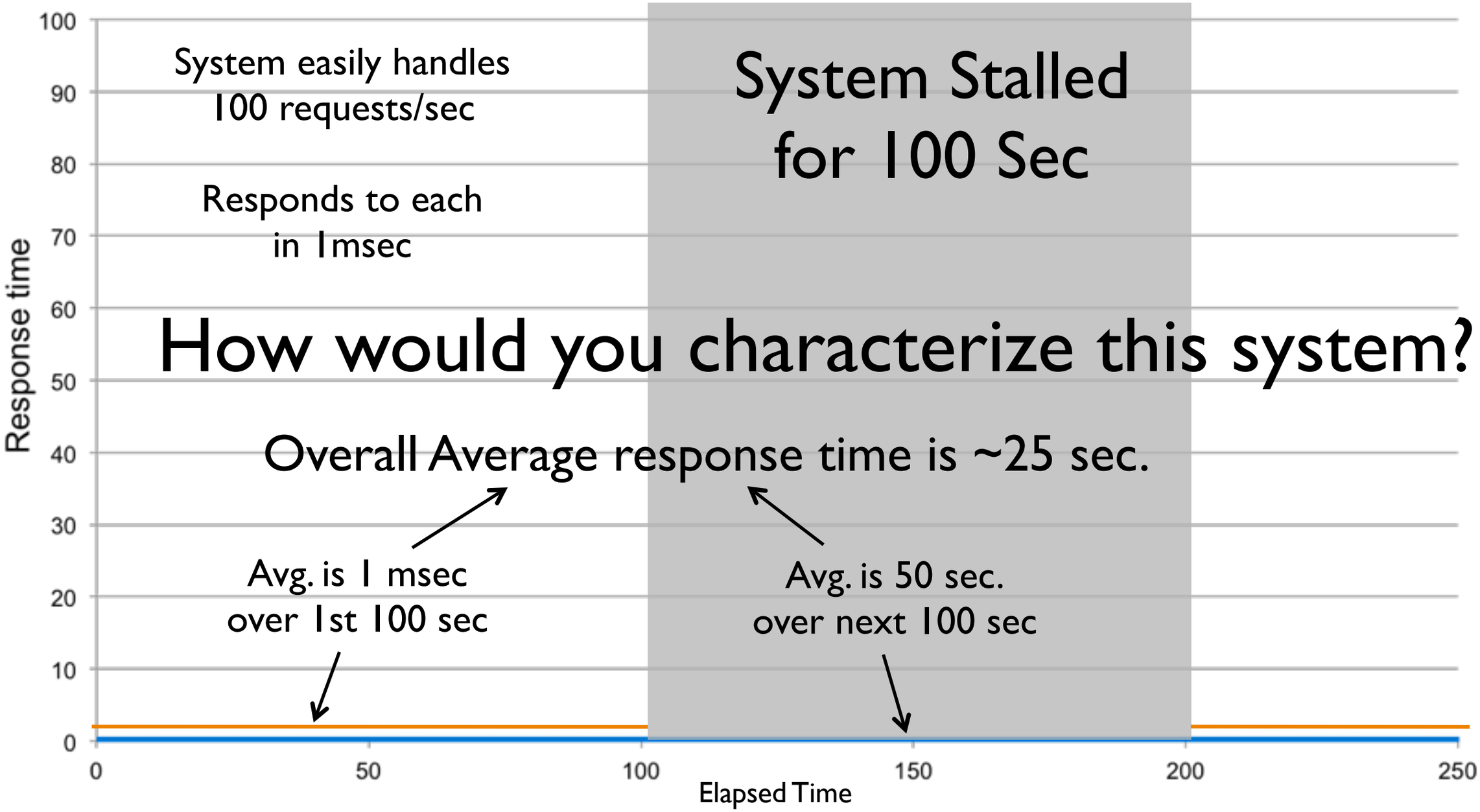
AZUL SYSTEMS®

The coordinated omission problem

- Common Example:
  - build/buy simple load tester to measure throughput
  - issue 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 only when all responses fit within rate interval
  - technique includes implicit "automatic backoff" and coordination
  - But requirements interested in random, uncoordinated requests
- How bad can this get, really?

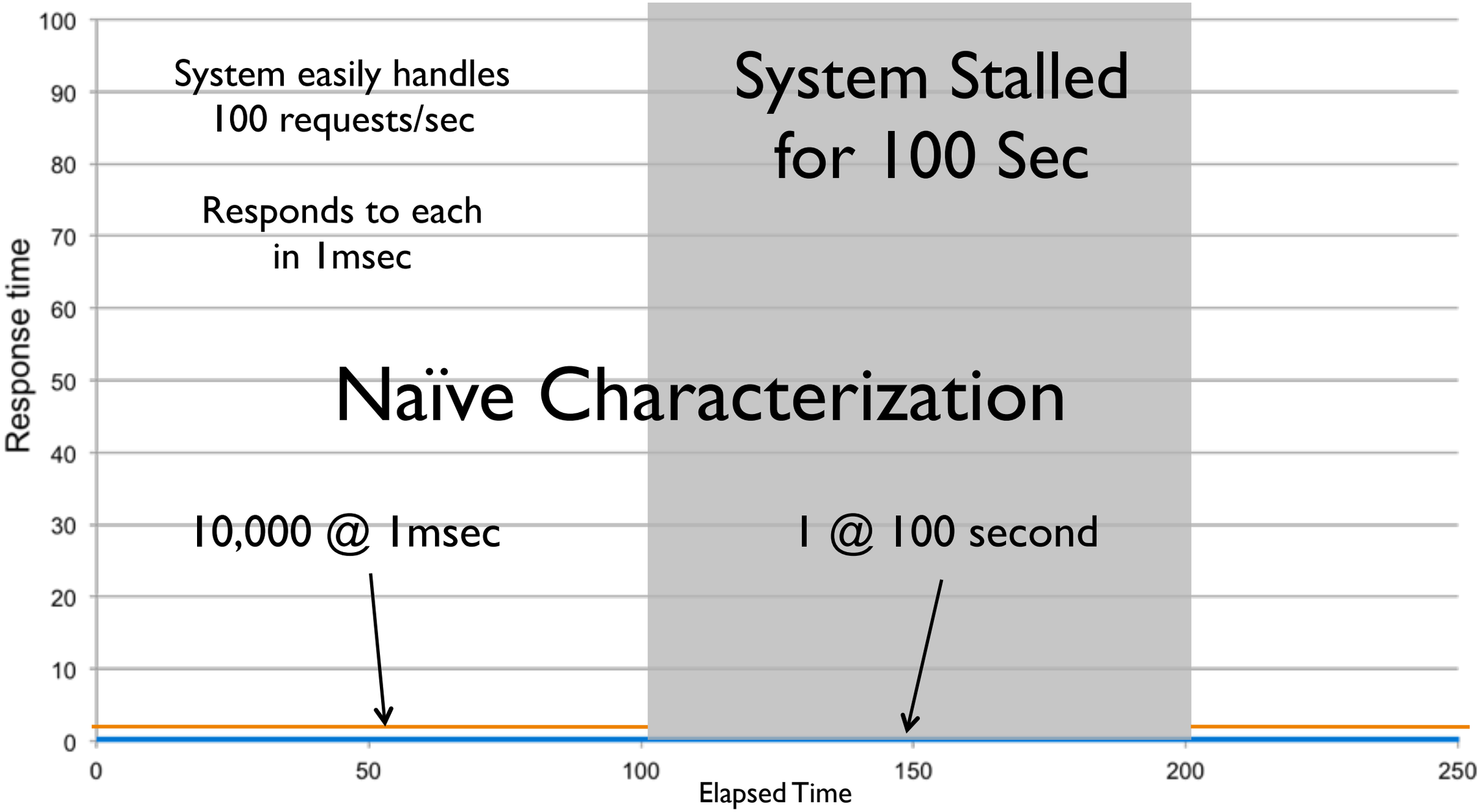
AZUL SYSTEMS

Example of naïve %'ile



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

Example of naïve %'ile

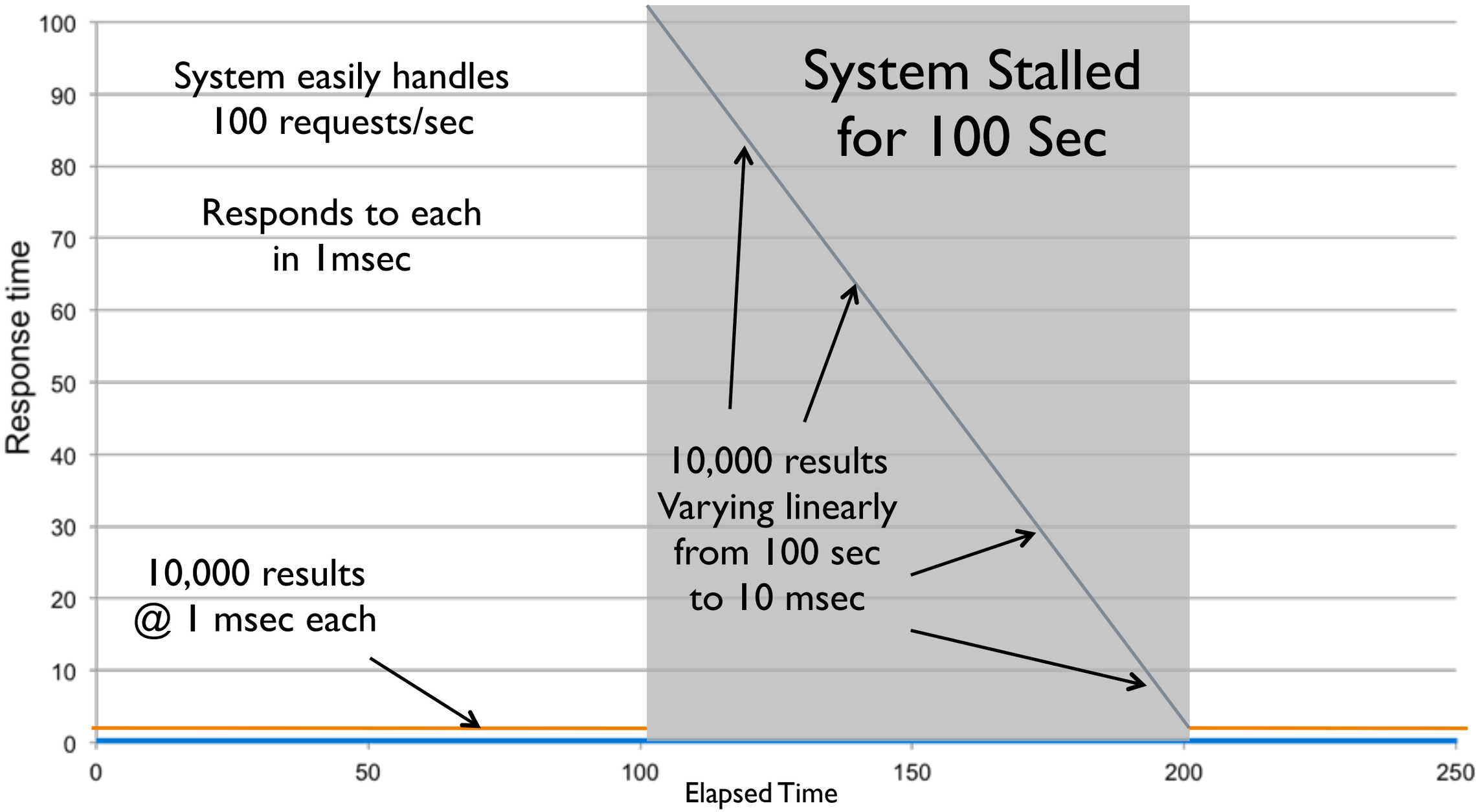


99.99%'ile is 1 msec! (should be ~100sec)

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

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), 2.606116218695308
[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

AZUL SYSTEMS

The real world

A world record SPECjEnterprise2010 result

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

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

305.197 seconds represents 8.4% of the timing run

AZUL SYSTEMS

Suggestions

Lessons learned

- Whatever your measurement technique is, test it.
- Run your measurement method against artificial system that creates the 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 use or derive from std. deviation.
- Always measure Max time. Consider what it means.
- Measure %'iles. Lots of them.

AZUL SYSTEMS

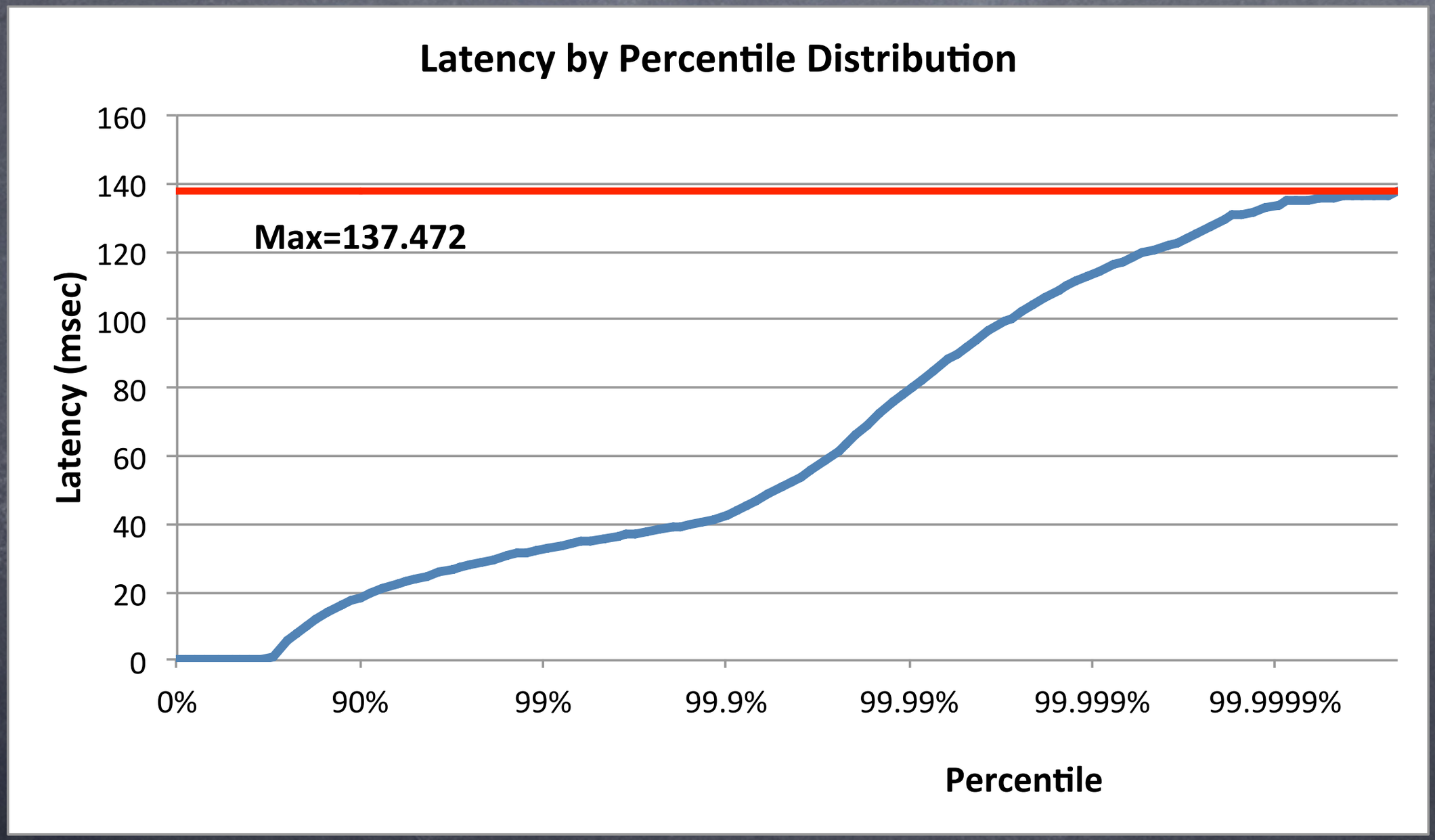
Some Tools

HdrHistogram

AZUL SYSTEMS®

HdrHistogram

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



You need a good dynamic range, and good resolution, at the same time

AZUL SYSTEMS

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 compensation for Coordinated Omission
- Open Source
  - On github, released to the public domain, creative commons CC0

AZUL SYSTEMS

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 use 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

AZUL SYSTEMS

HdrHistogram

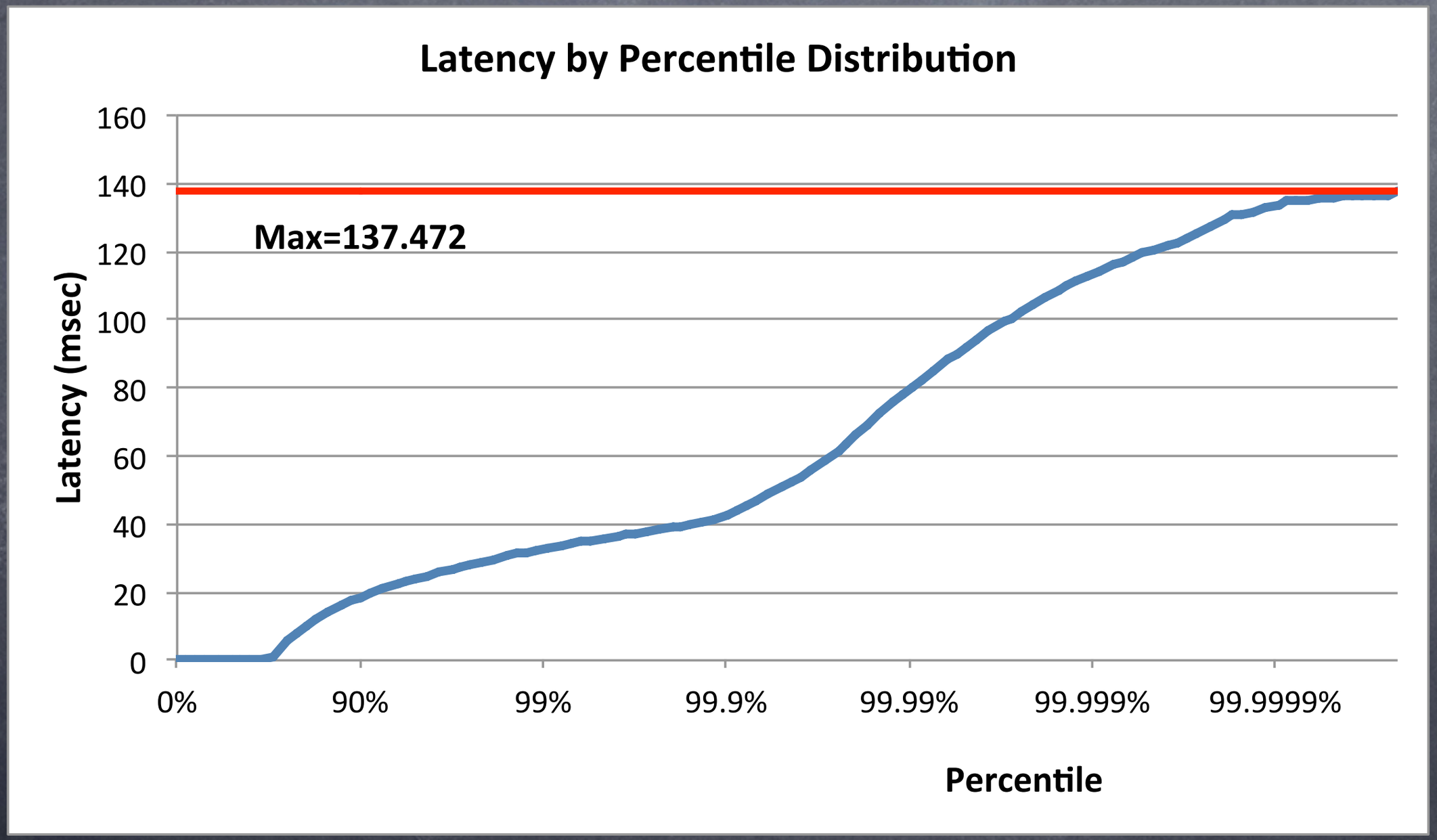
- Provides tools for iteration
  - Linear, Logarithmic, Percentile
- Supports percentile iterators
  - Practical due to high dynamic range
- Convenient percentile output
  - E.g. 10% intervals between 0 and 50%, 5% intervals between 50% and 75%, 2.5% intervals between 75% and 87.5%, ...
  - Very useful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


AZUL SYSTEMS

HdrHistogram

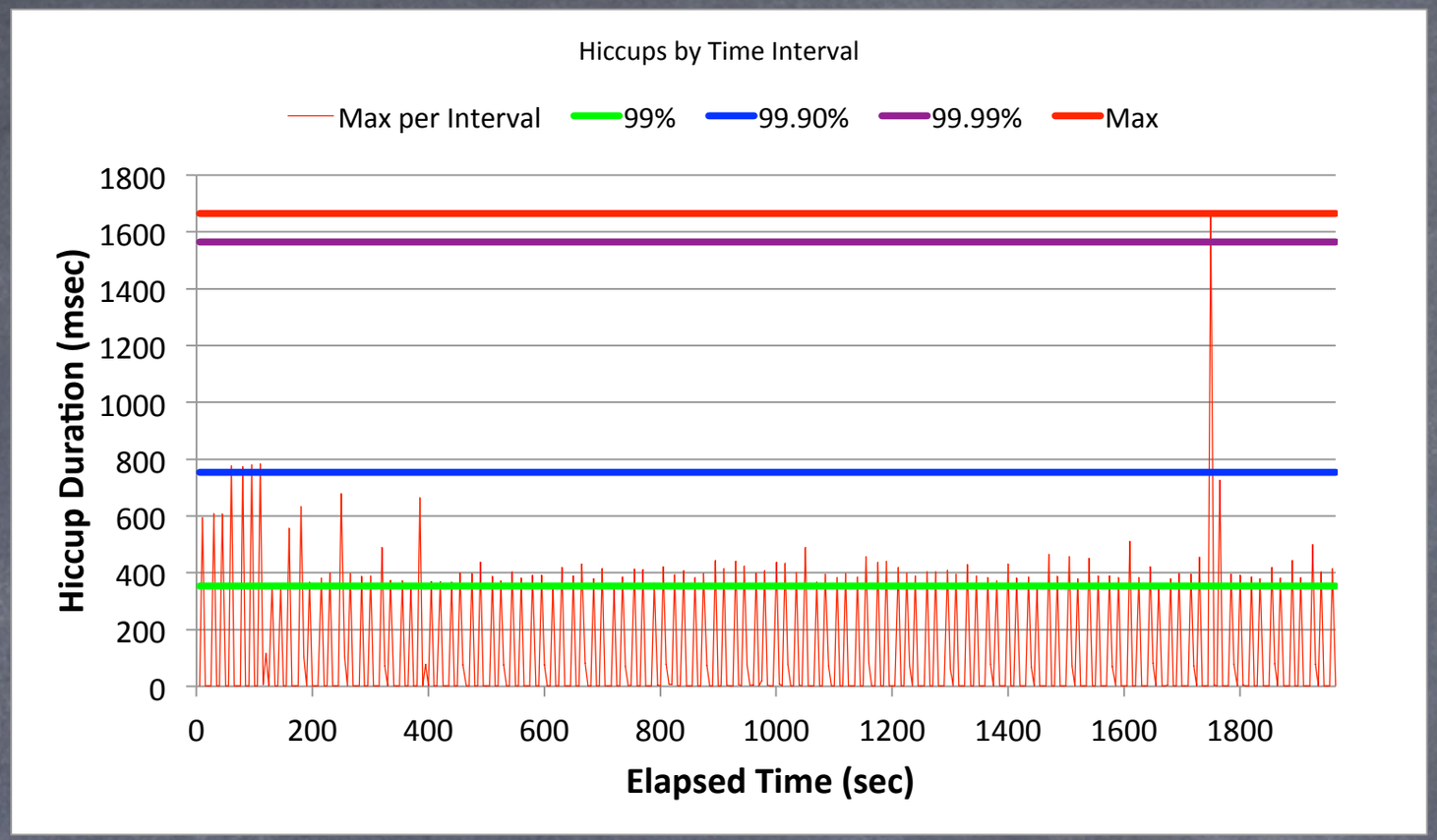


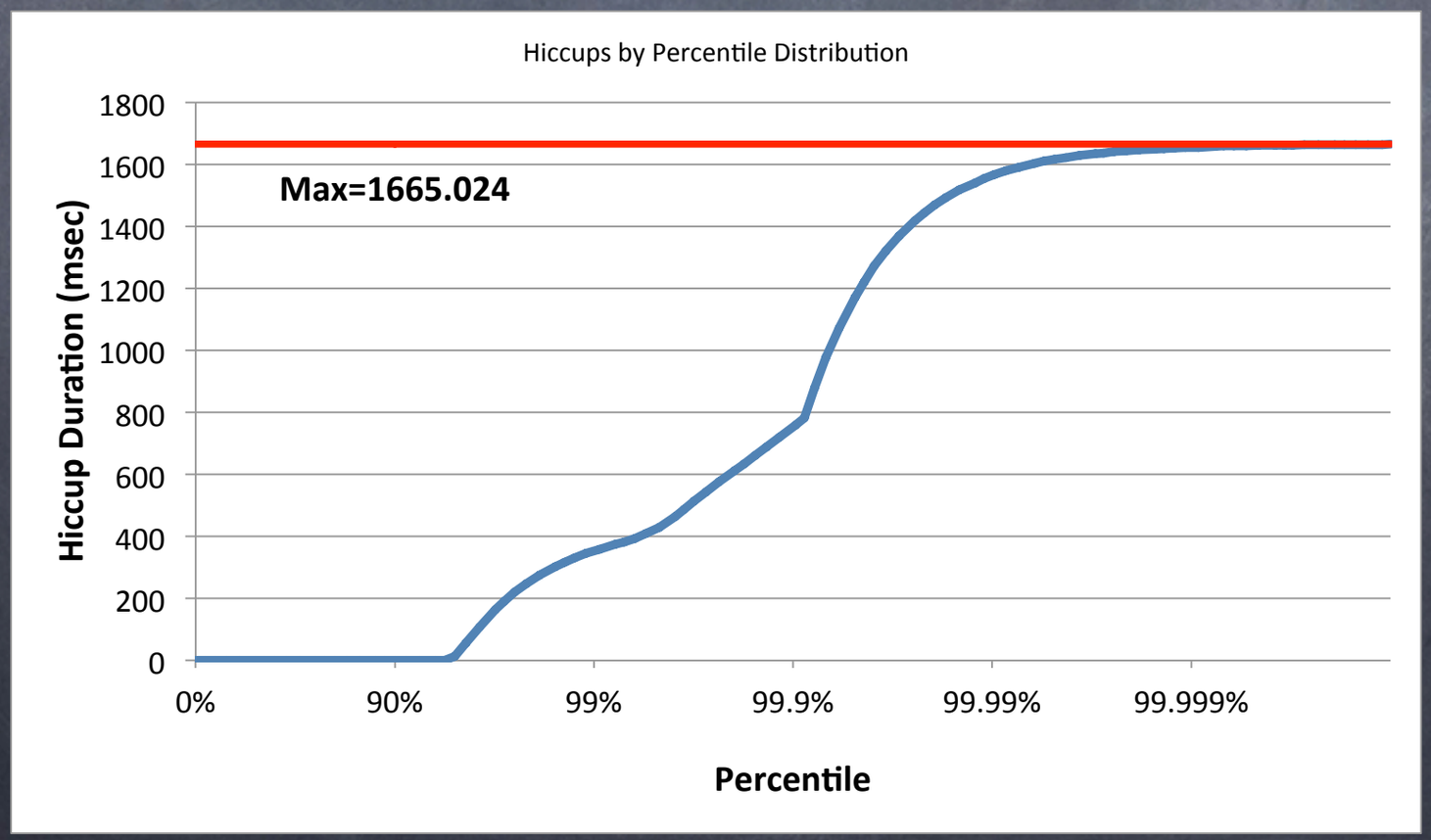
AZUL SYSTEMS

jHiccup

AZUL SYSTEMS®

Incontinuities in Java platform execution





AZUL SYSTEMS

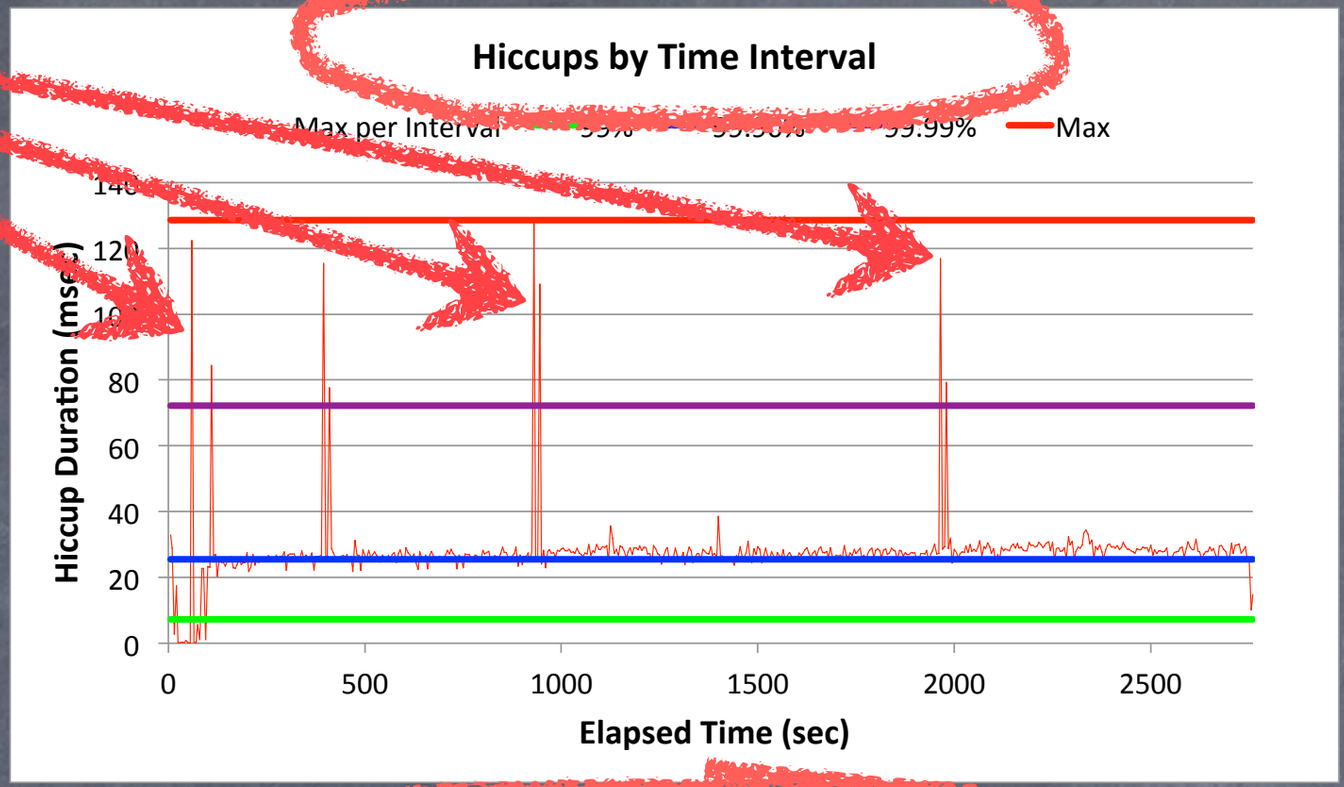
jHiccup

- 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 the word "jHiccup" to your java launch line
  - % jHiccup java myflags myApp
  - (Or use as a java agent)
  - Adds a background thread that samples time @ 1000/sec
- Open Source
  - Released to the public domain, creative commons CC0

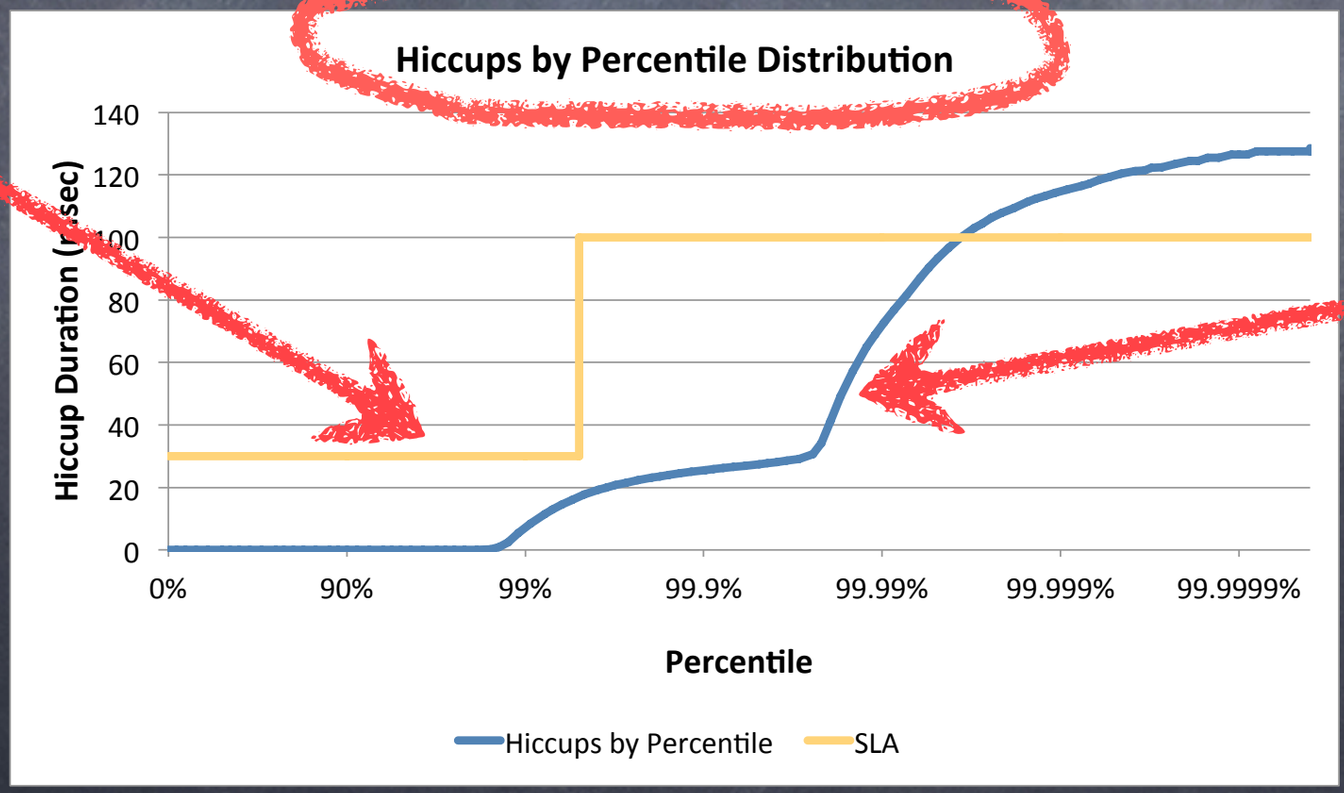
AZUL SYSTEMS

Telco App Example

Max Time per interval



Optional SLA plotting



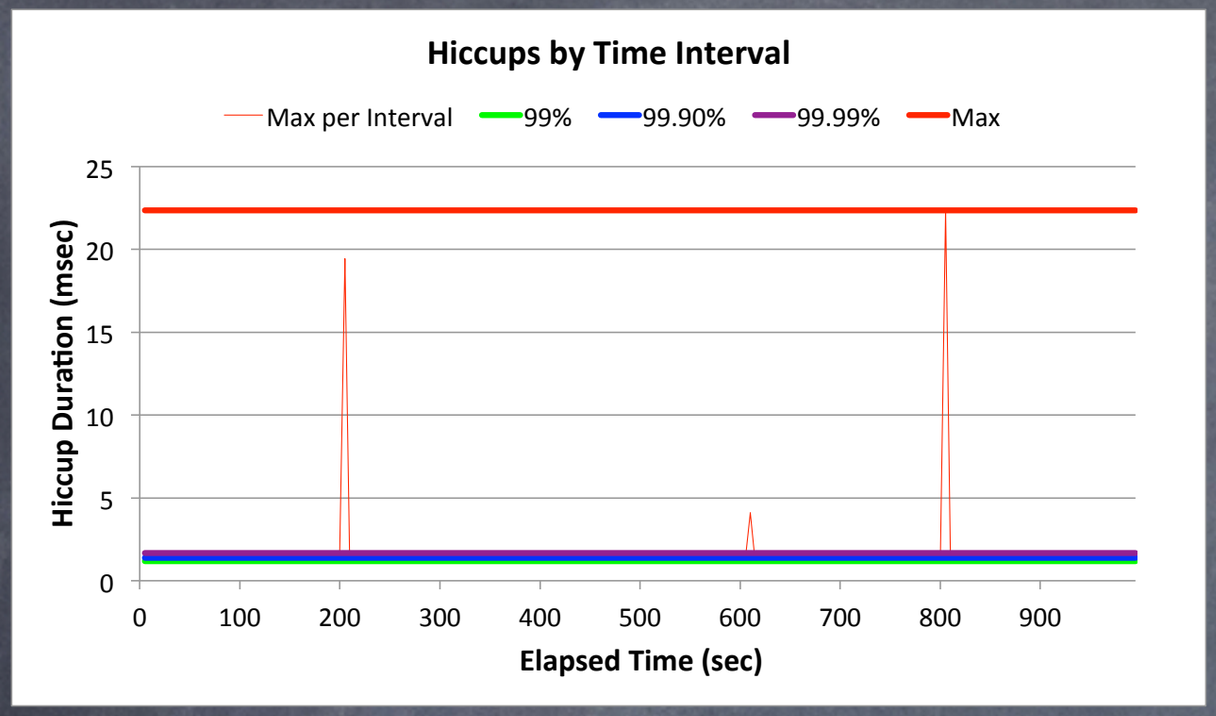
Hiccup duration at percentile levels

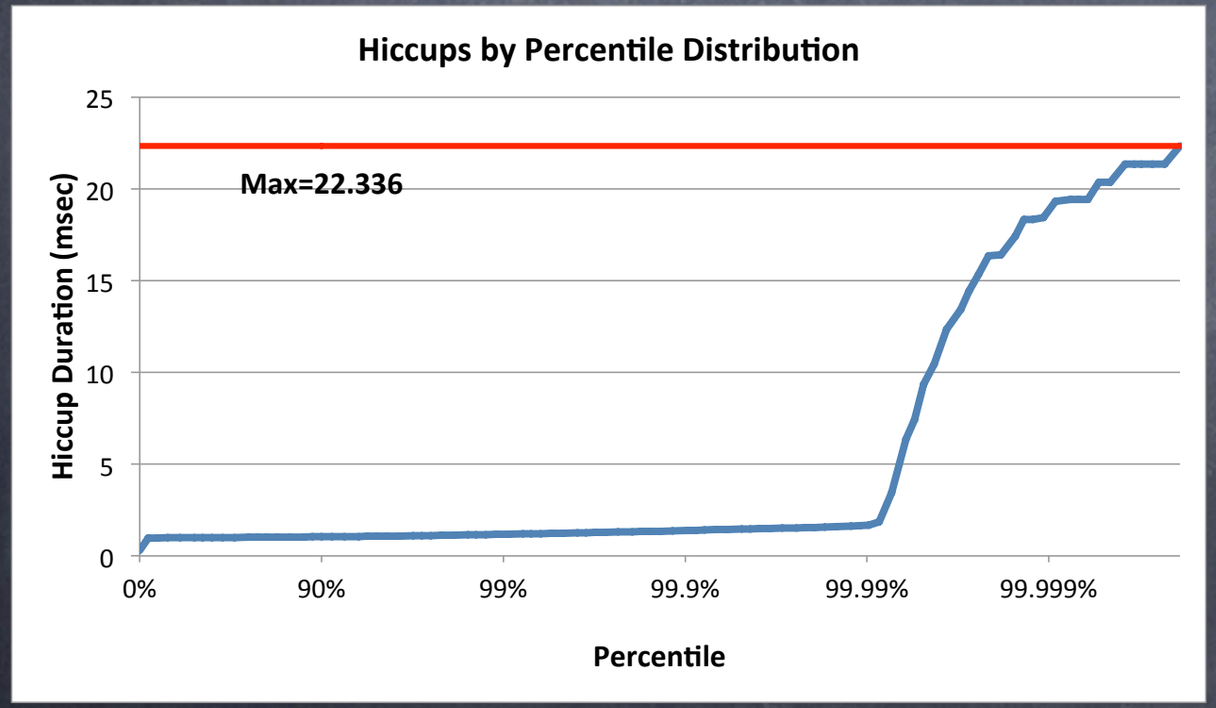
AZUL SYSTEMS

Examples

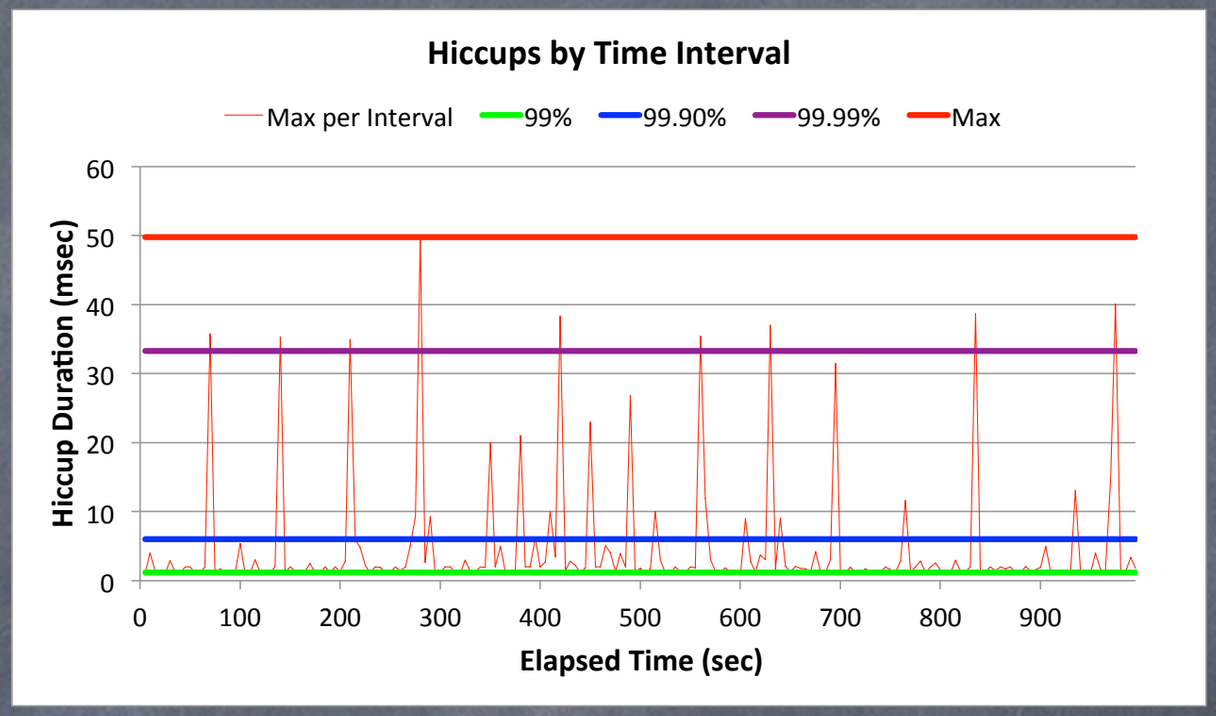
AZUL SYSTEMS®

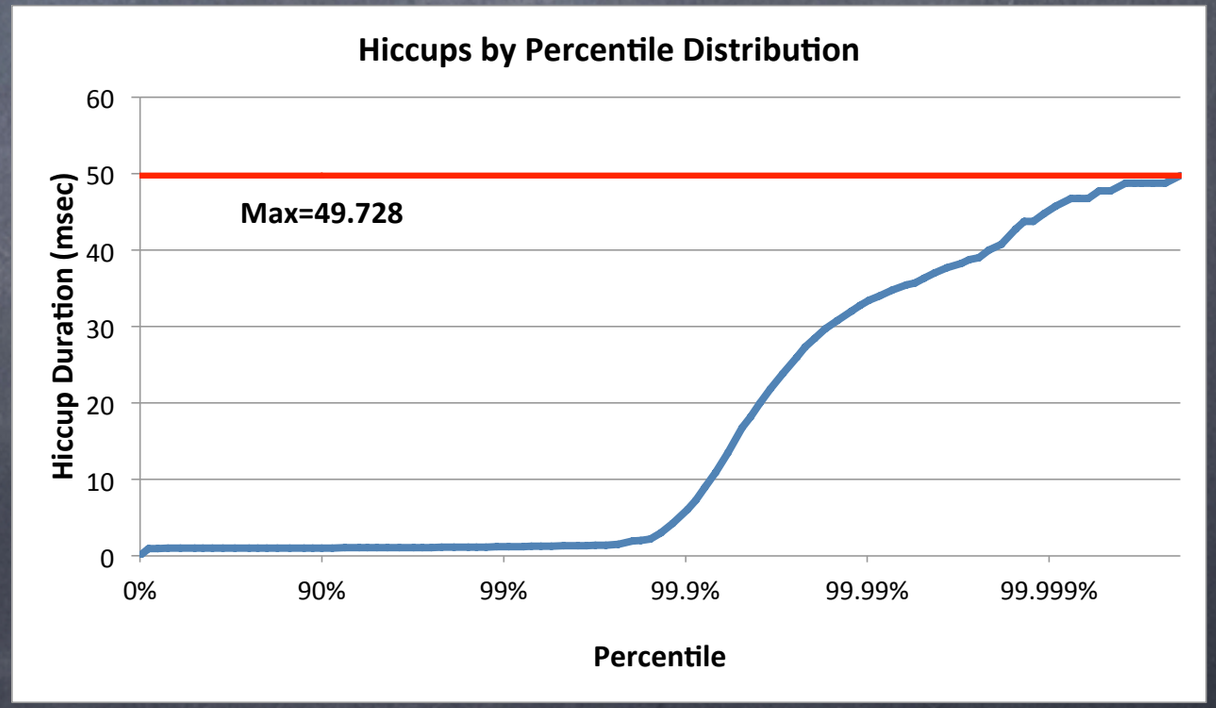
Idle App on Quiet System





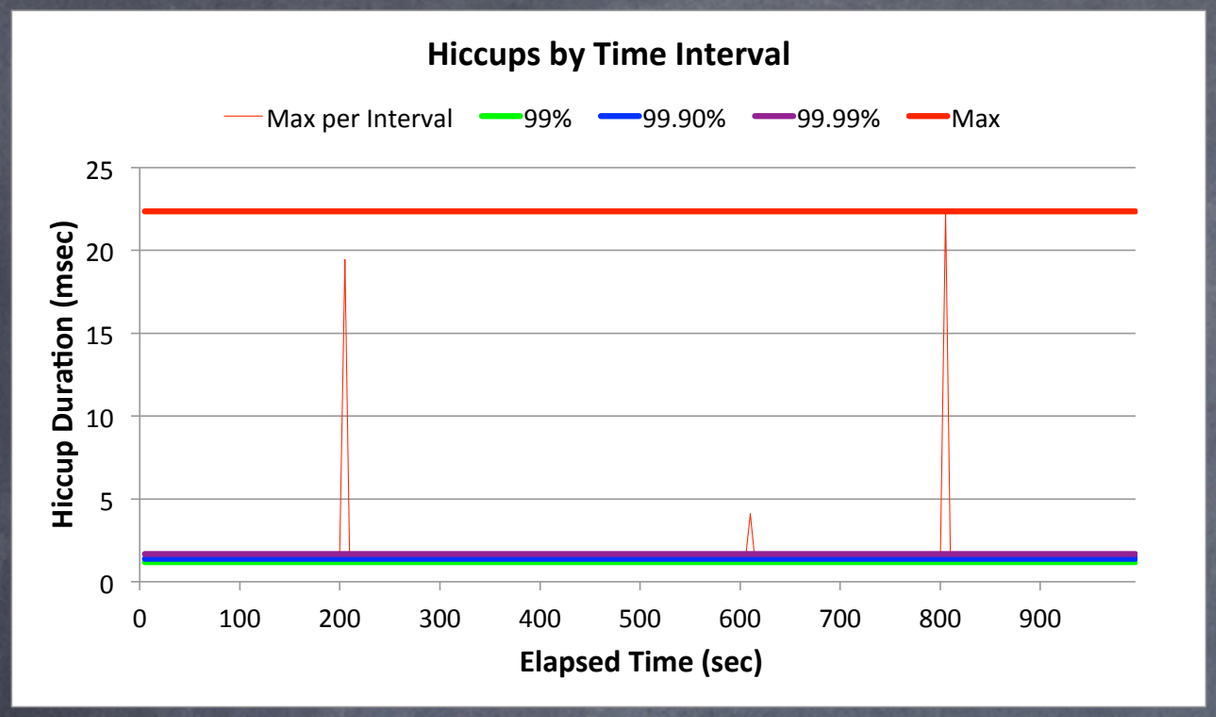
Idle App on Busy System

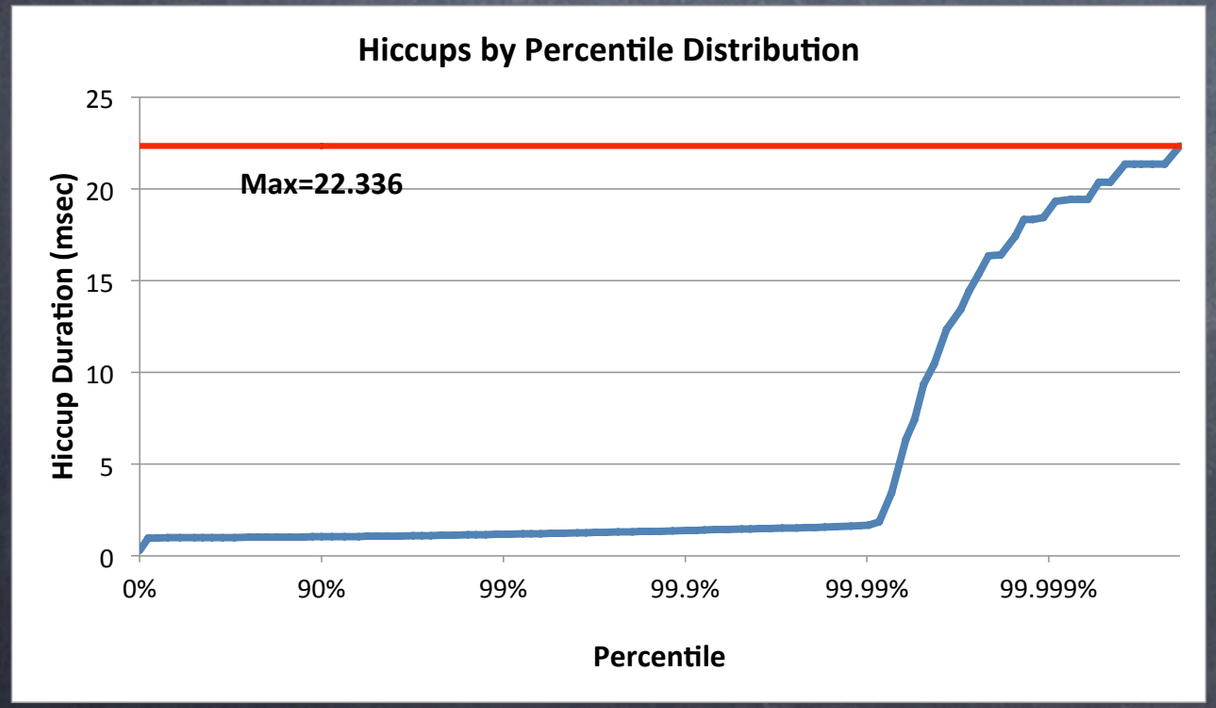




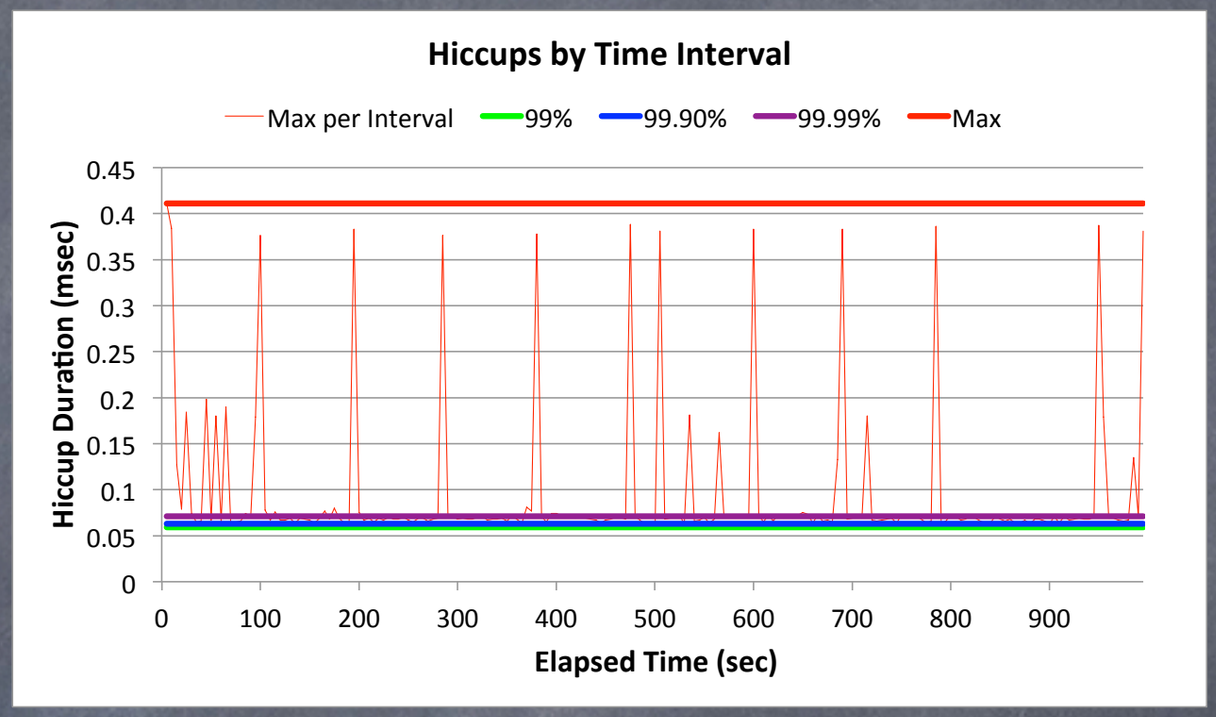
AZUL SYSTEMS

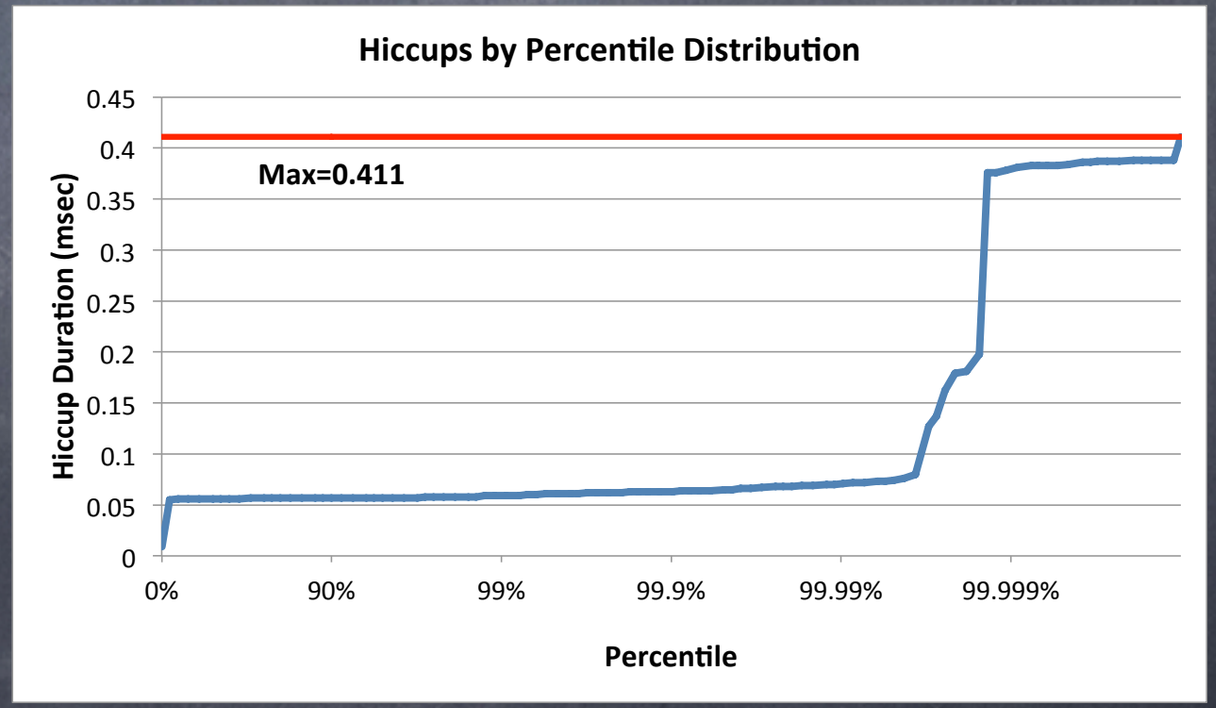
Idle App on Quiet System





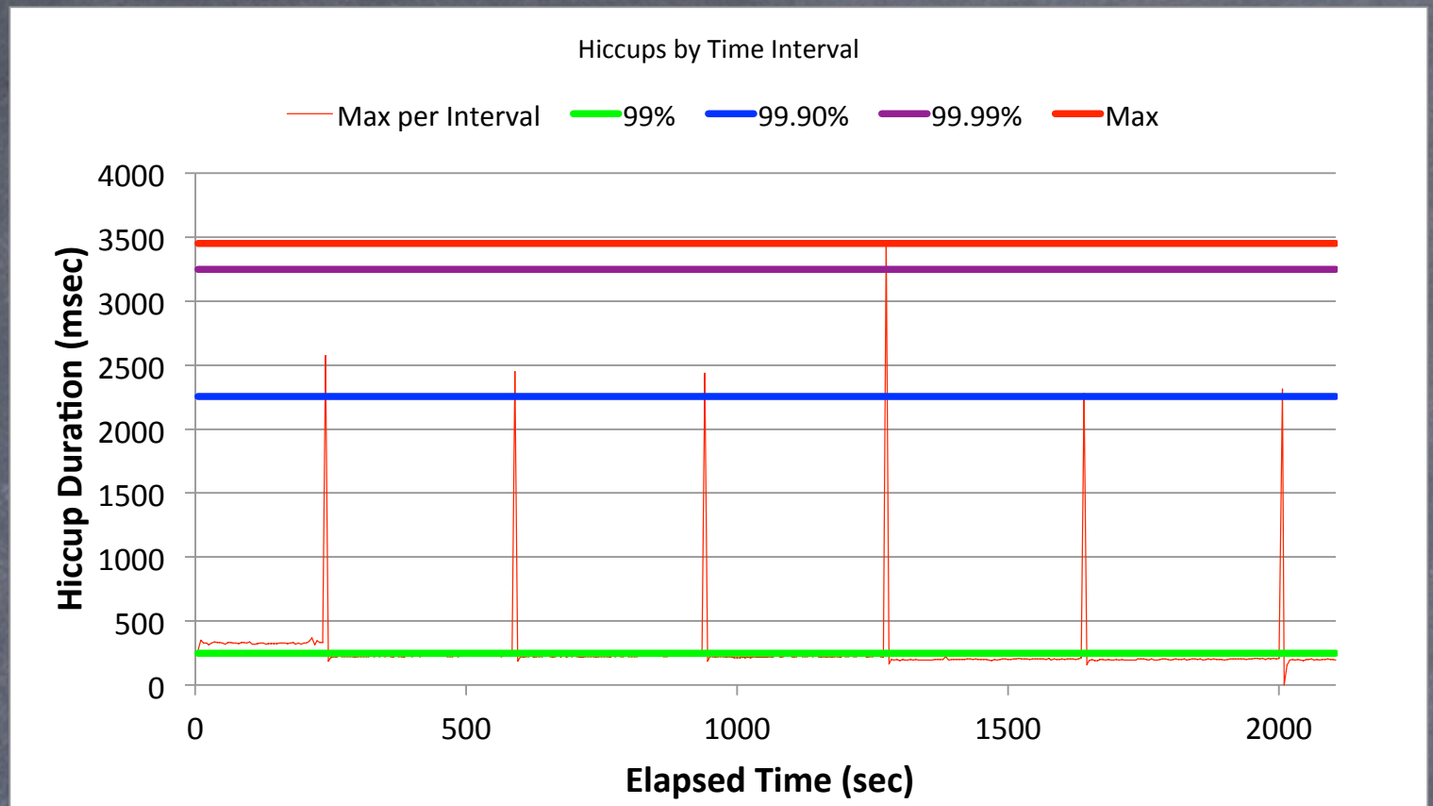
Idle App on Dedicated System

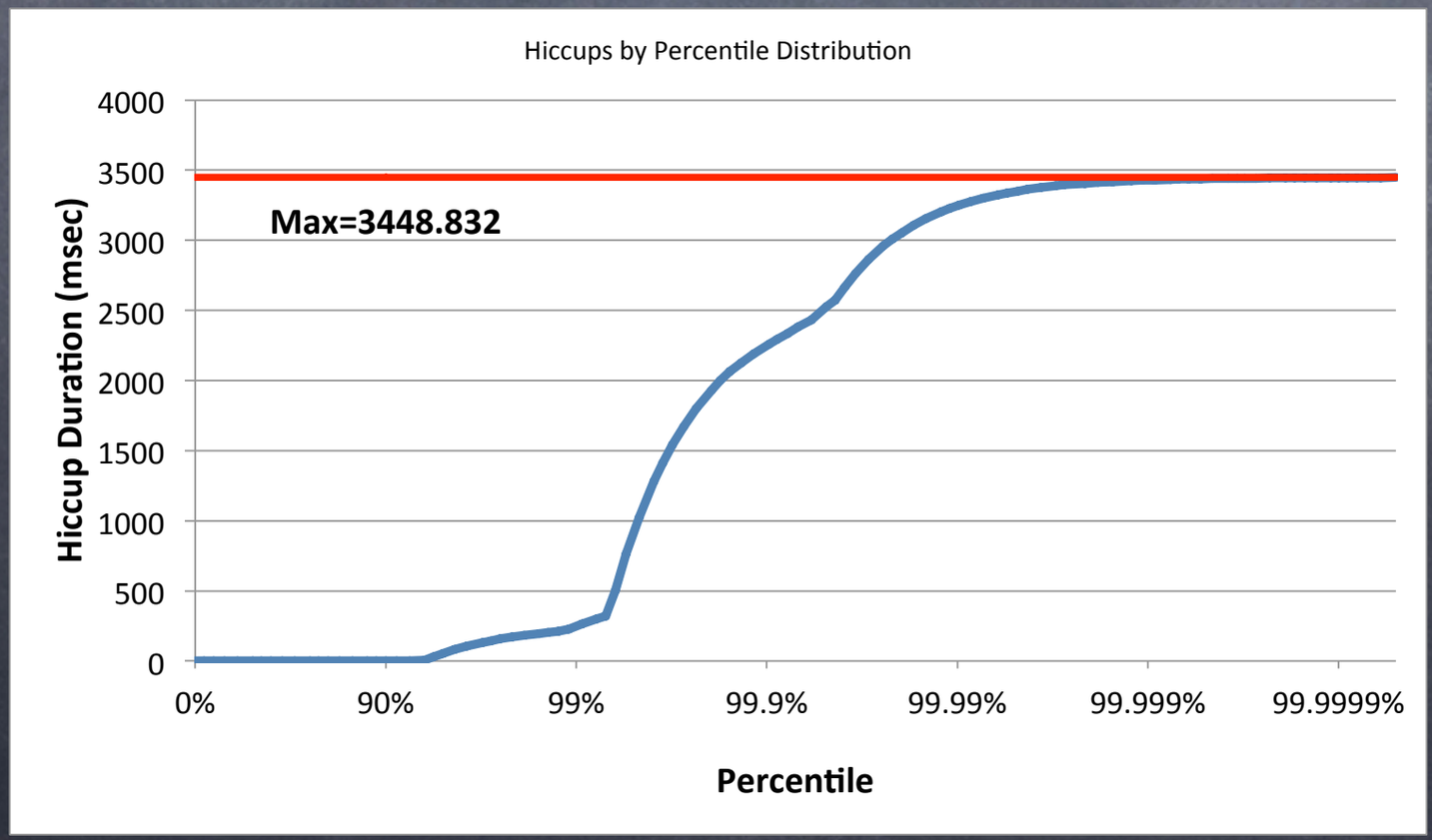




AZUL SYSTEMS

EHCache: 1GB data set under load





AZUL SYSTEMS

Fun with jHiccup

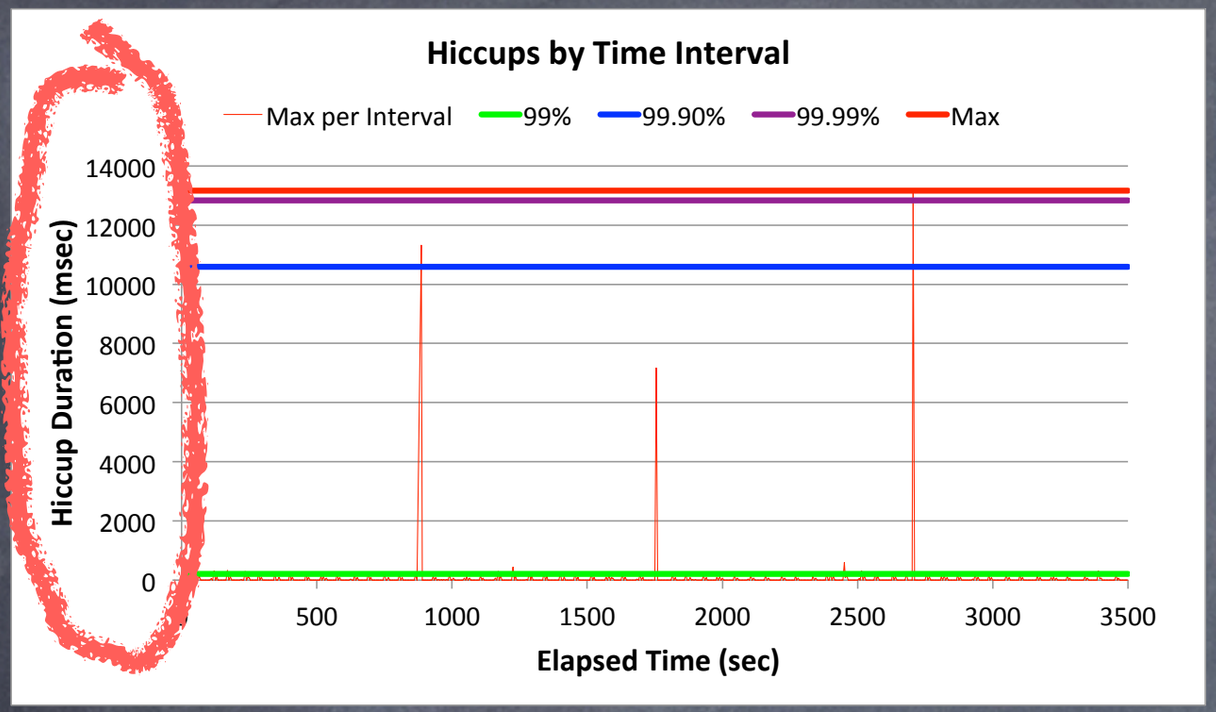
Charles Nutter @headius 20 Jan

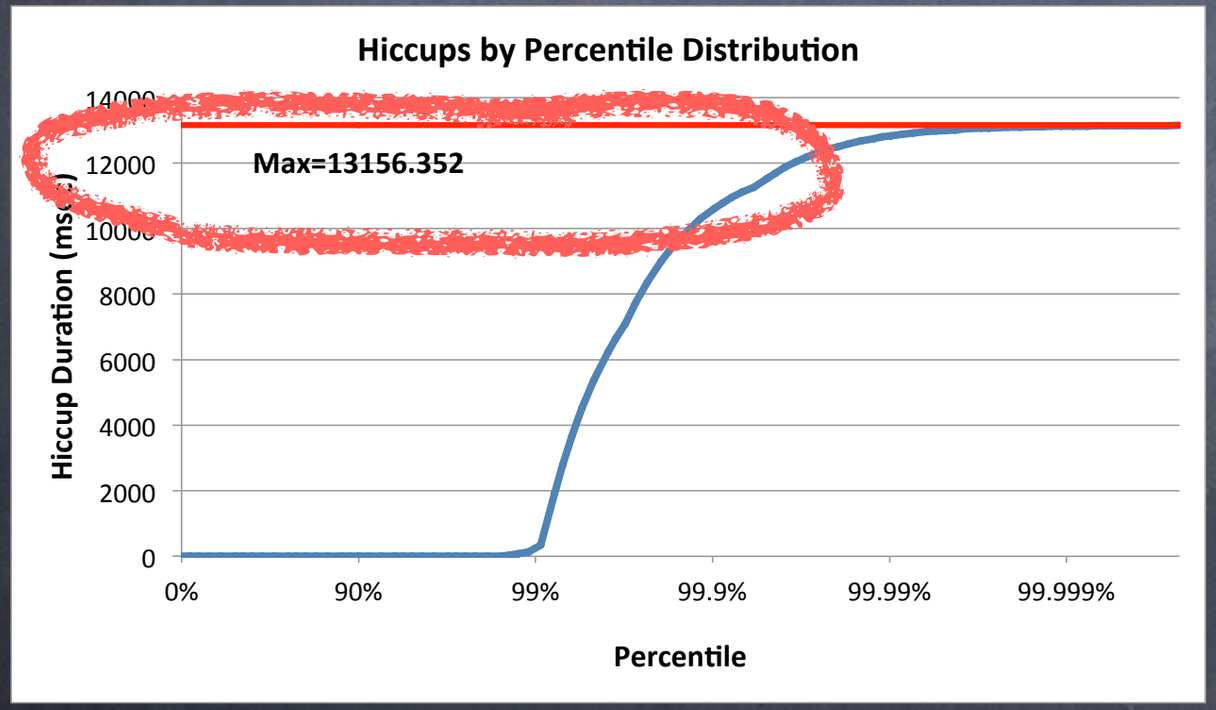
jHiccup, @AzulSystems' free tool to show you why your JVM sucks compared to Zing: bit.ly/wsH5A8 (thx @bascule)

Retweeted by Gil Tene

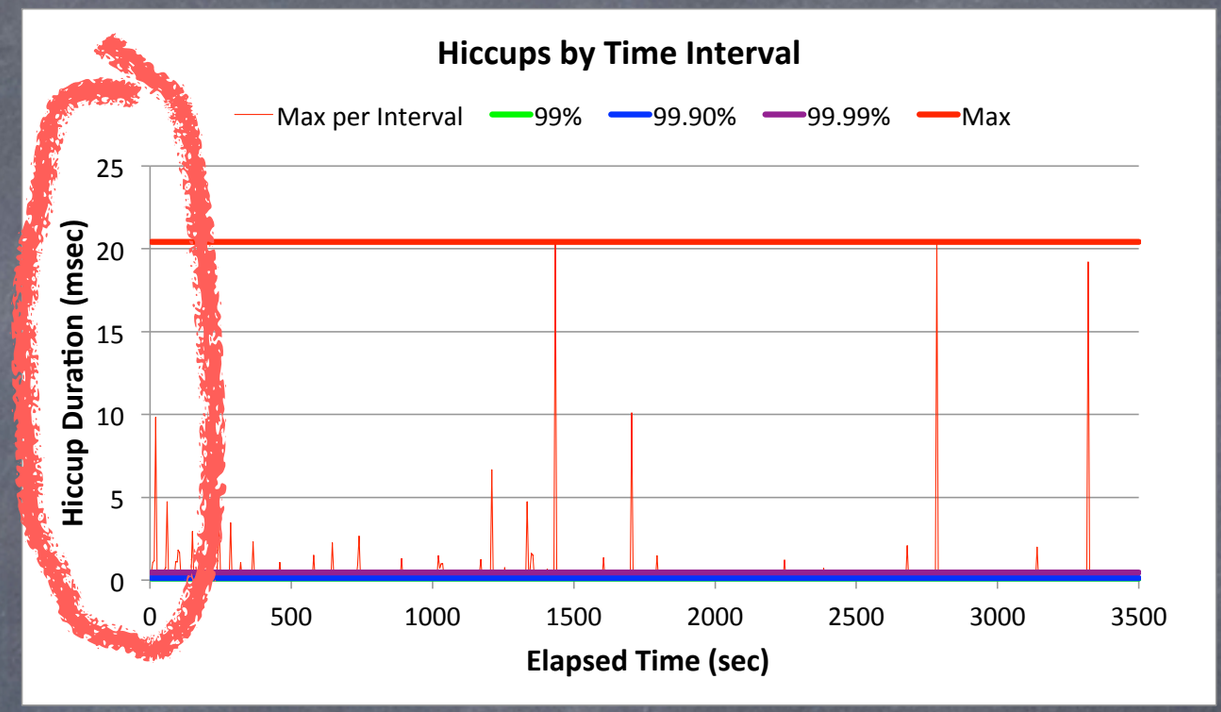
AZUL SYSTEMS

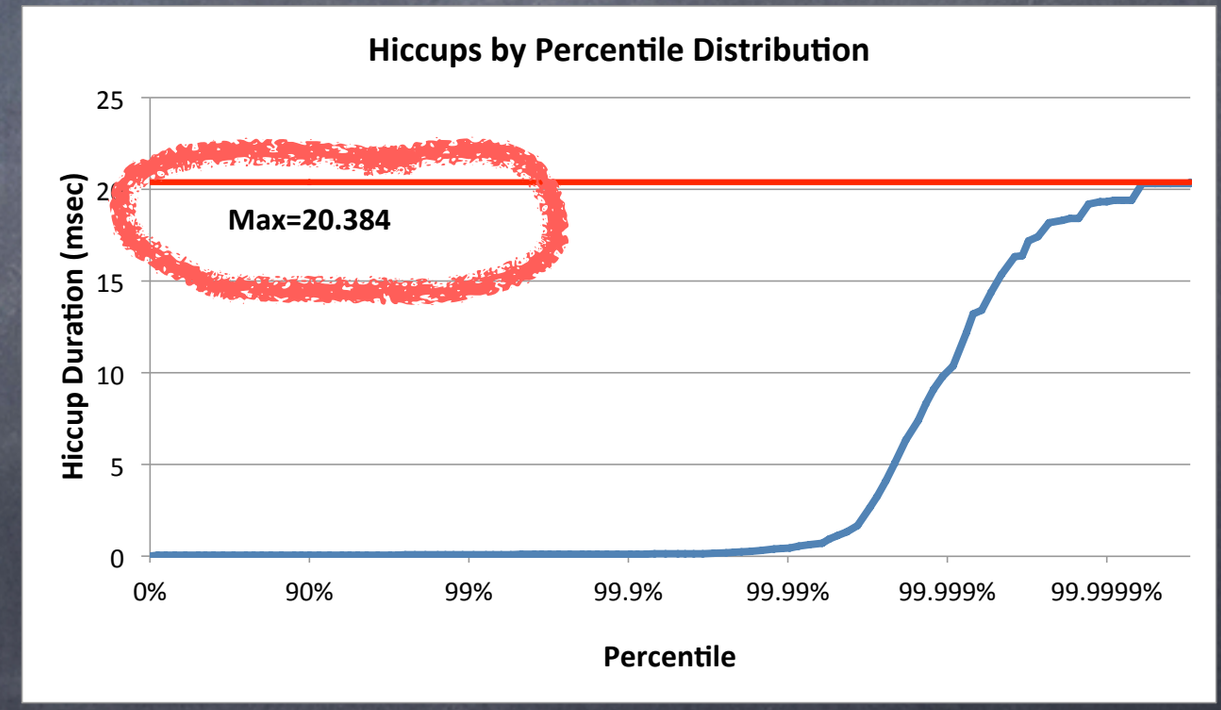
Oracle HotSpot CMS, 1GB in an 8GB heap





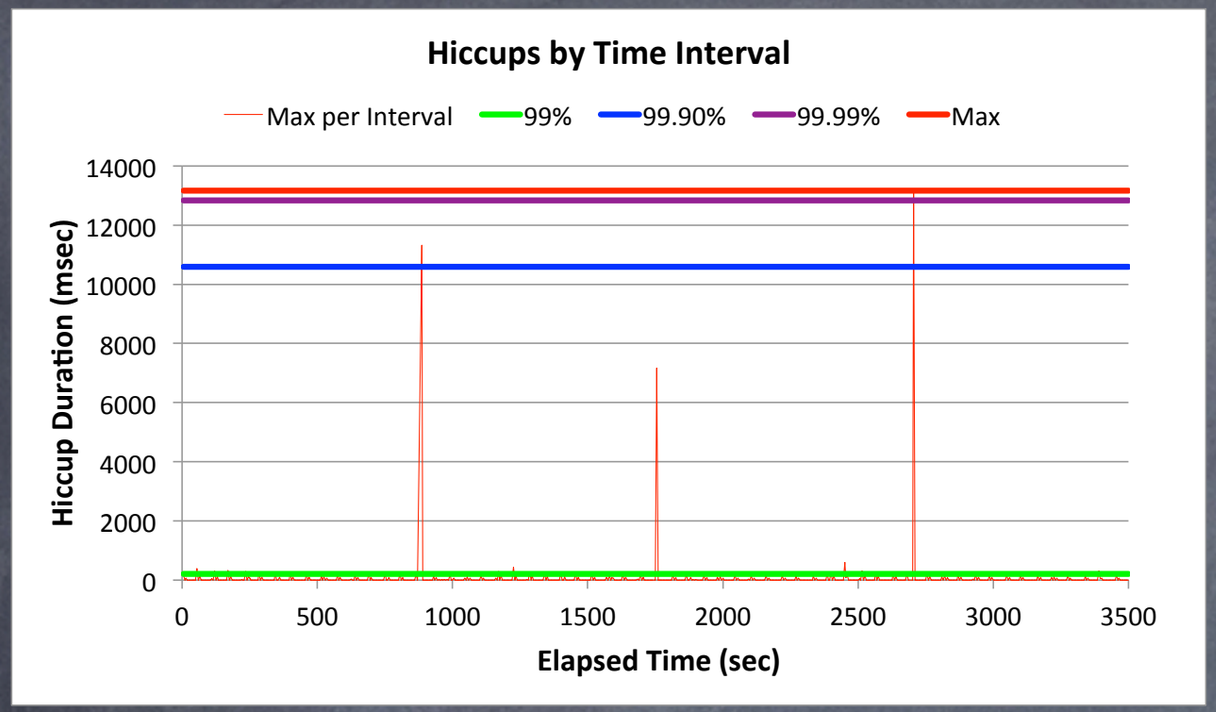
Zing 5, 1GB in an 8GB heap

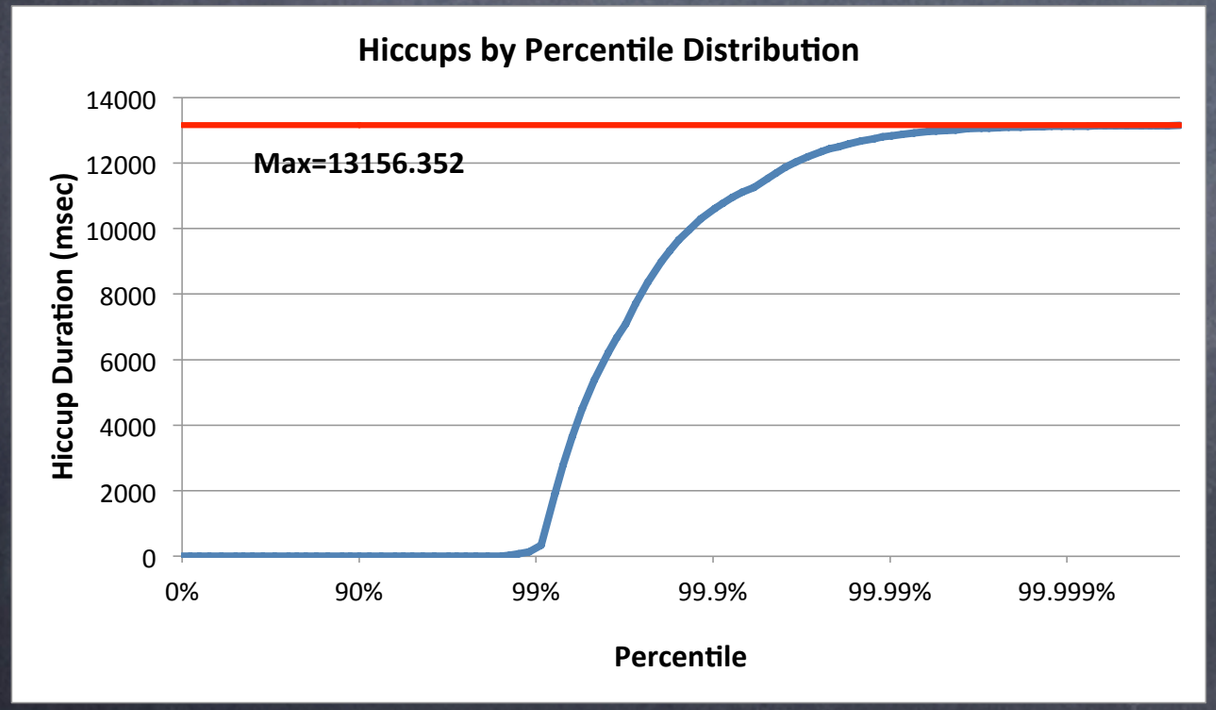




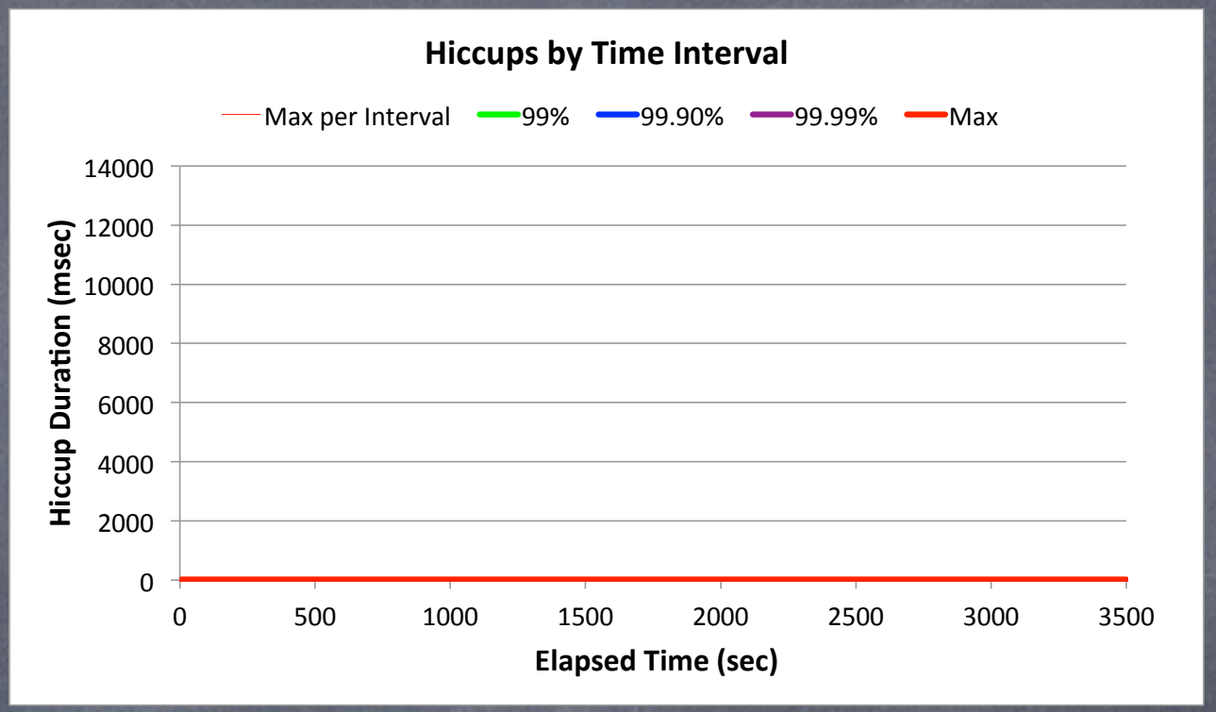
AZUL SYSTEMS

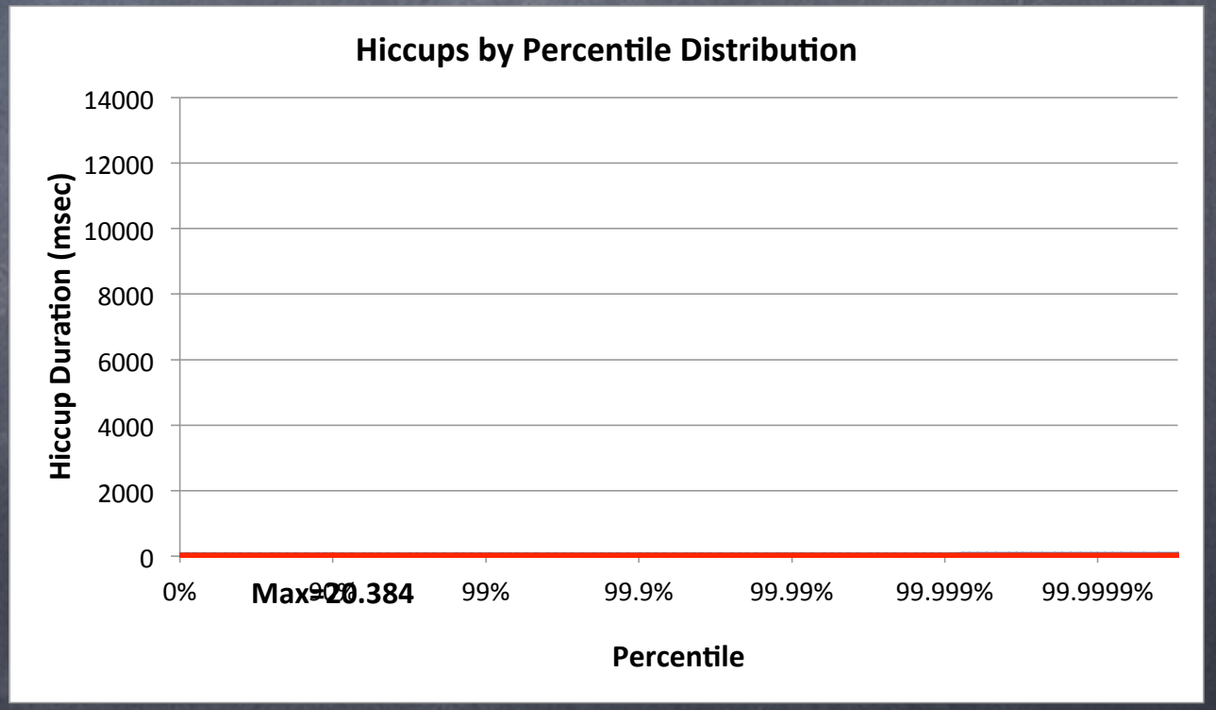
Oracle HotSpot CMS, 1GB in an 8GB heap





Zing 5, 1GB in an 8GB heap





Drawn to scale

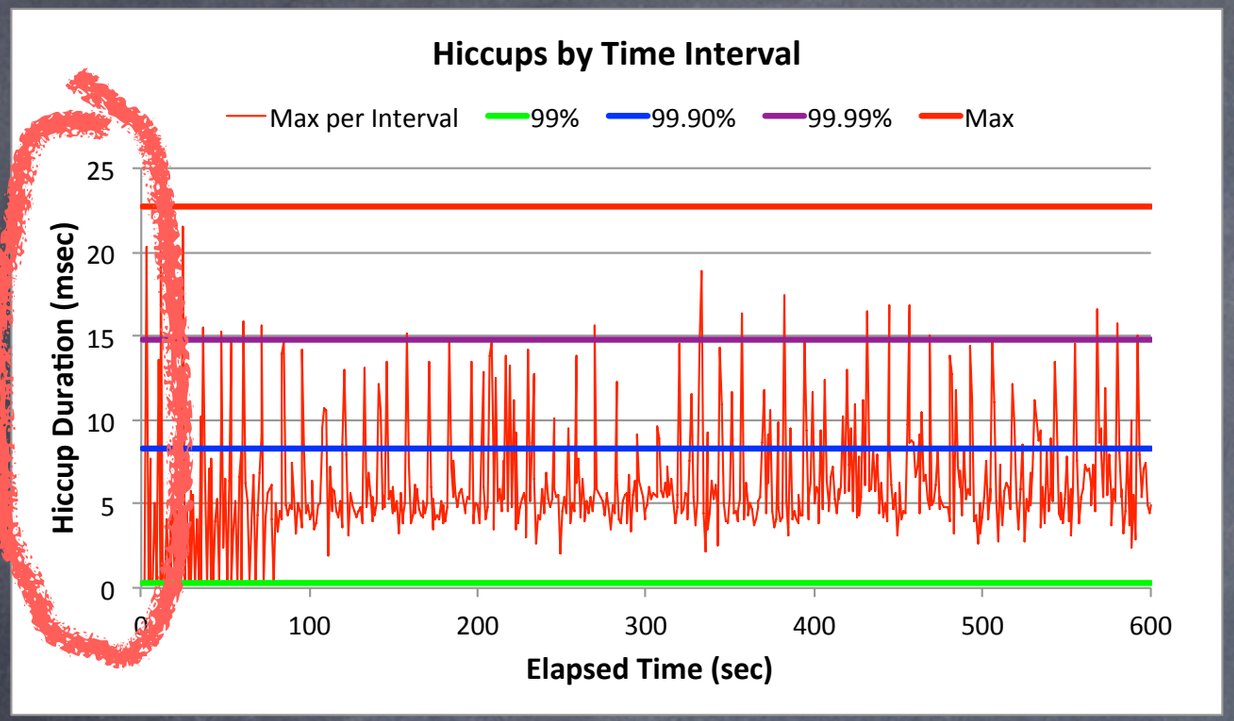
AZUL SYSTEMS

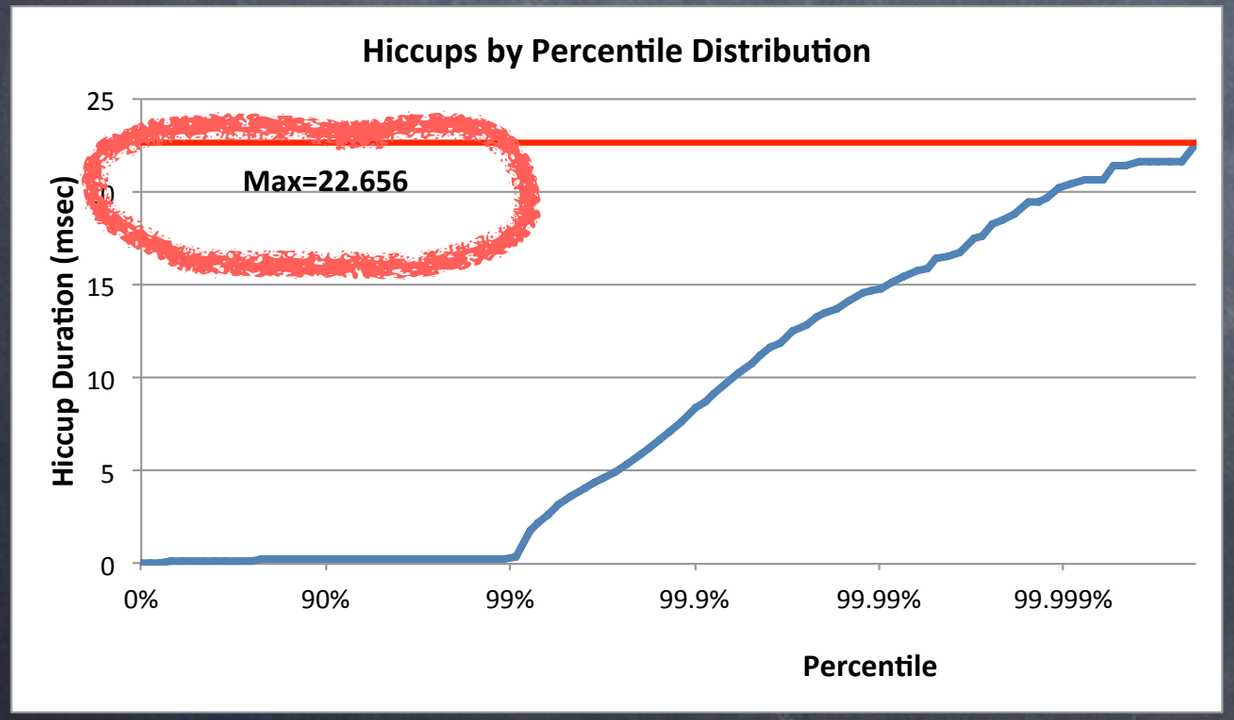
What you can expect (from Zing) in the low latency world

- Assuming individual transaction work is "short" (on the order of 1 msec), and assuming you don't have 100s of runnable threads competing for 10 cores...
- "Easily" get your application to < 10 msec **worst case**
- With some tuning, 2-3 msec **worst case**
- Can go to below 1 msec **worst case**...
  - May require heavy tuning/tweaking
  - Mileage WILL vary

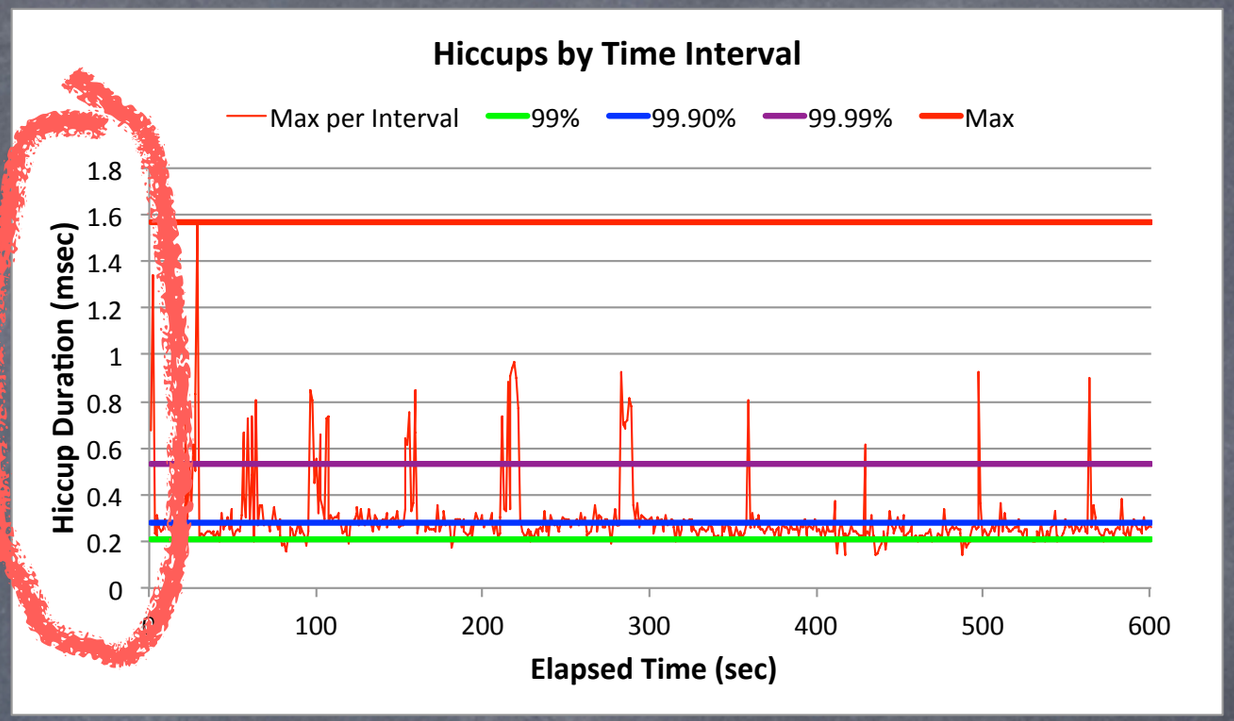
AZUL SYSTEMS

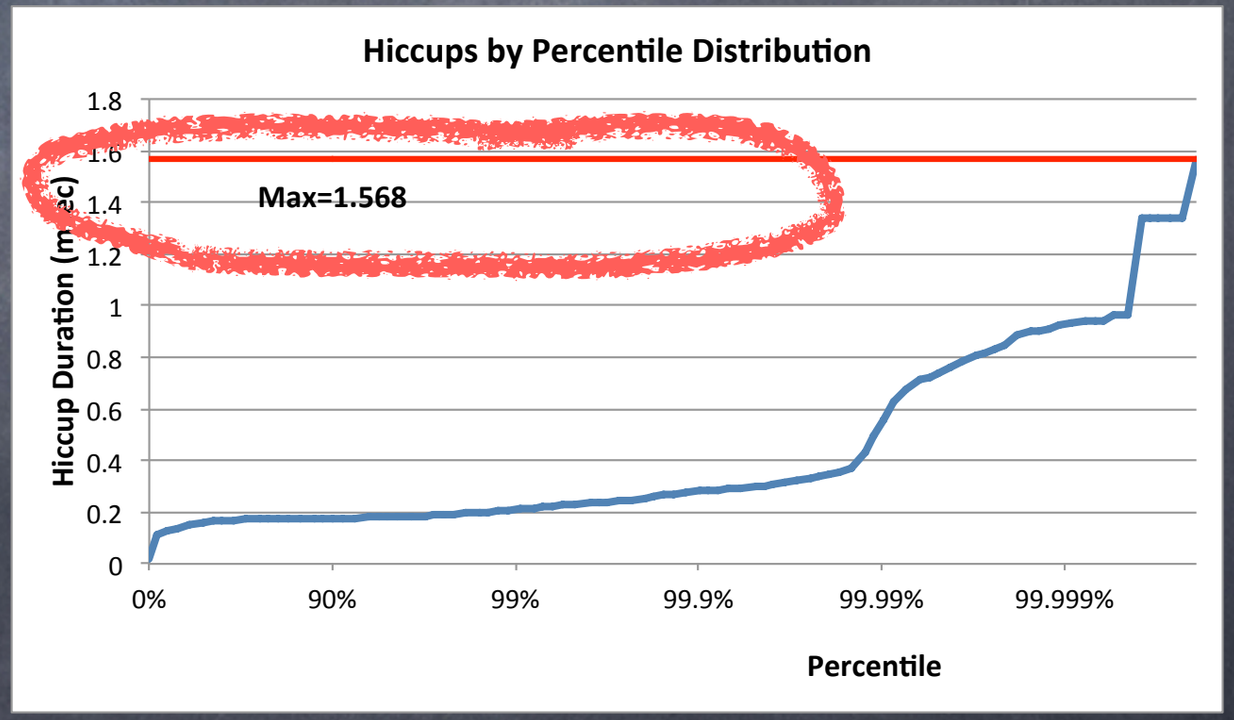
Oracle HotSpot (pure newgen)





Zing

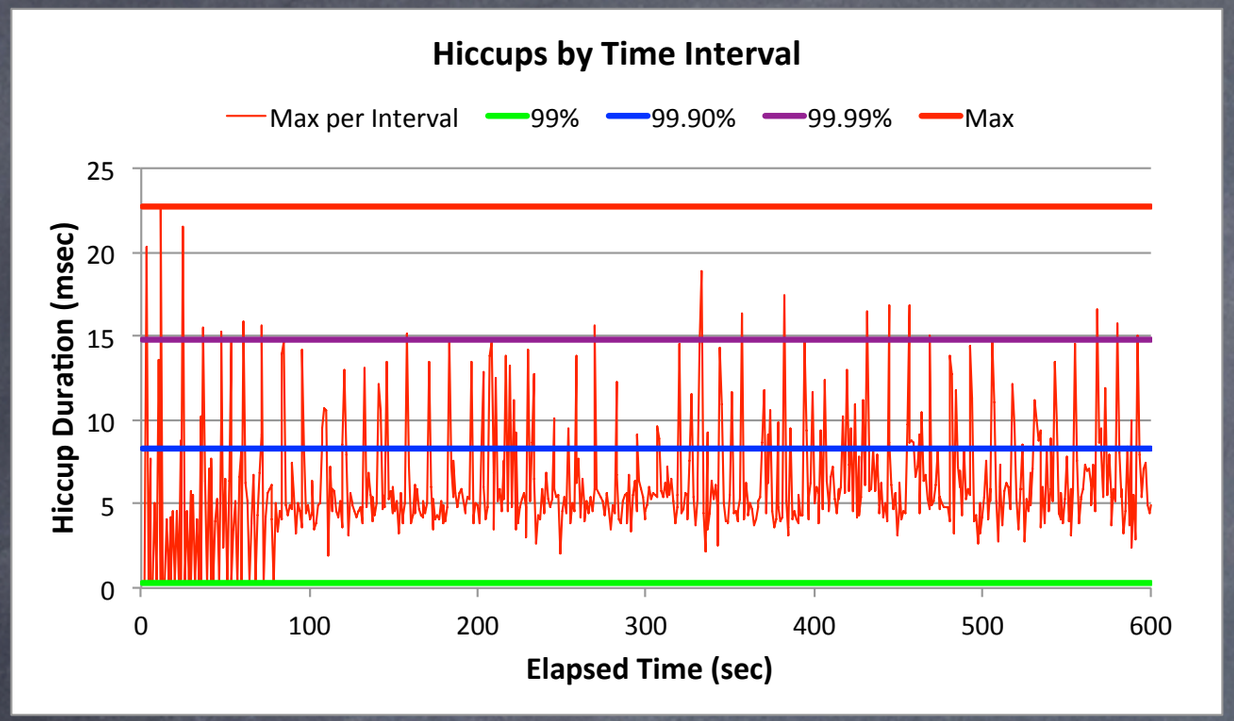


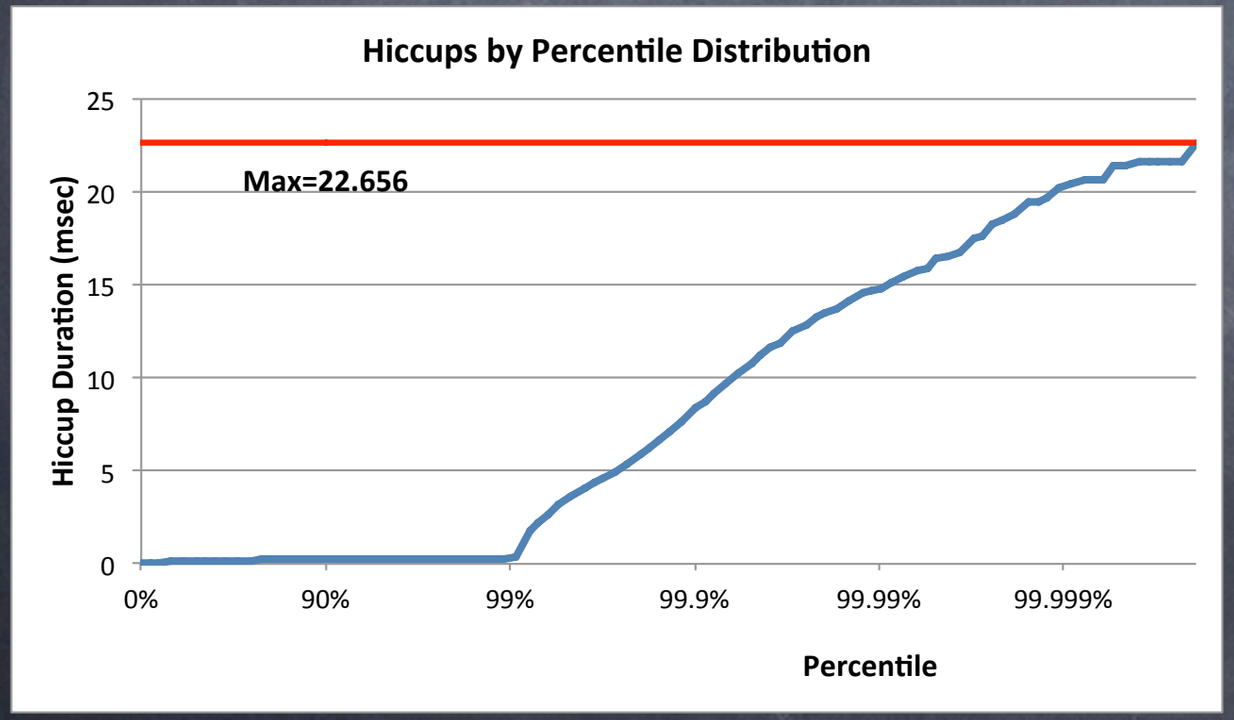


Low latency trading application

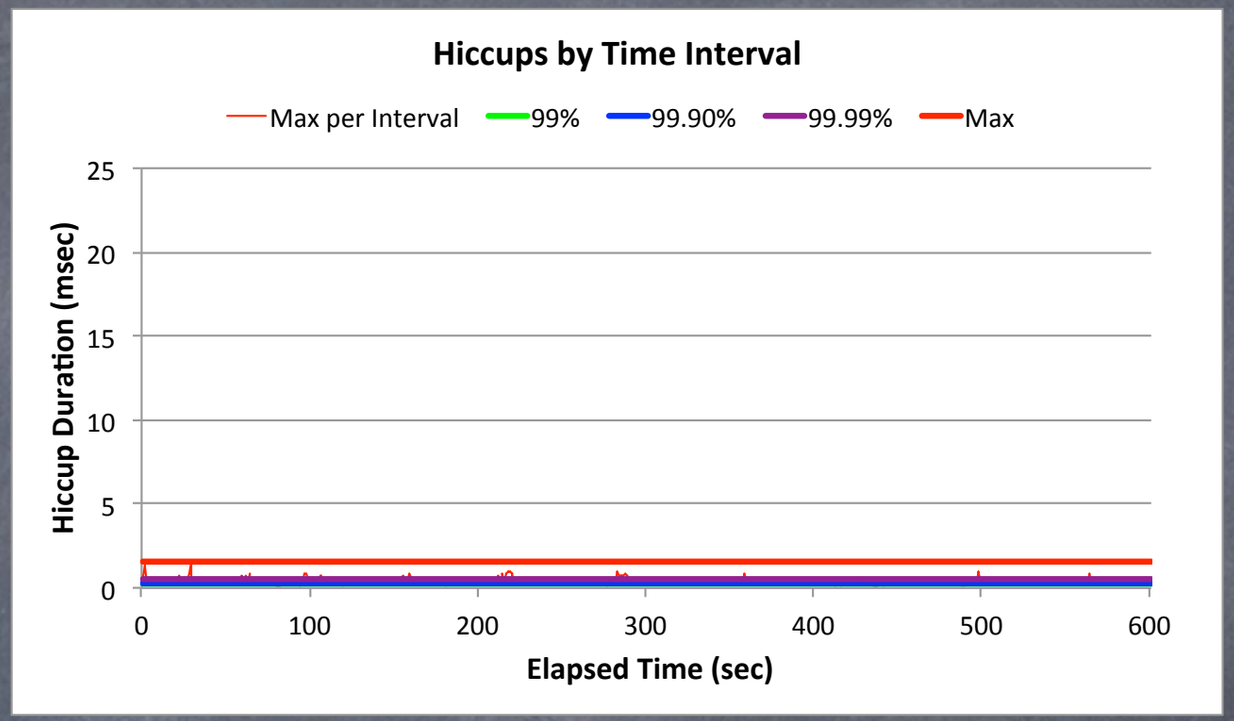
AZUL SYSTEMS

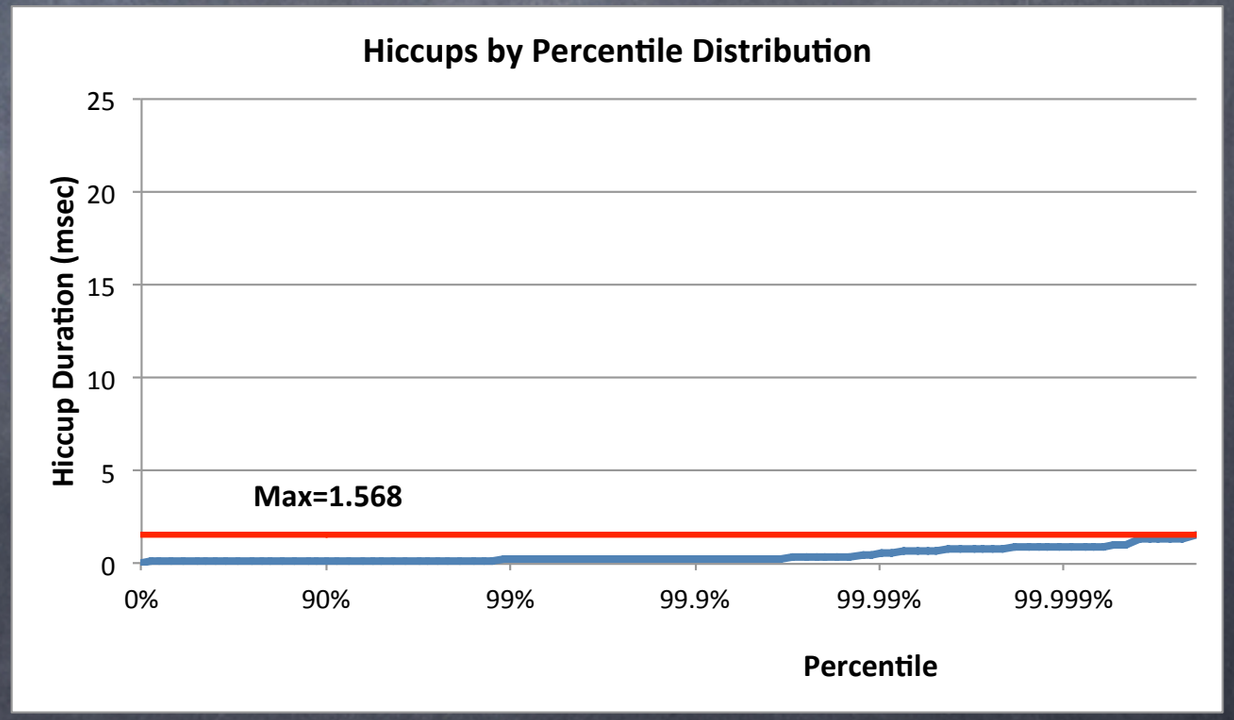
Oracle HotSpot (pure newgen)





Zing





Low latency - Drawn to scale

AZUL SYSTEMS

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 lead to orders-of-magnitude errors and lead to bad business decisions
- jHiccup and HdrHistogram are generically useful
- The Zing JVM is cool...

AZUL SYSTEMS®

Q & A

http://www.azulsystems.com

http://www.jhiccup.com

http://giltene.github.com/HdrHistogram