

Understanding Latency and Response Time Behavior

Pitfalls, Lessons and Tools

Matt Schuetze
Director of Product Management
Azul Systems











Latency Behavior



- Latency: The time it took one operation to happen
- Each operation occurrence has its own latency
- So we need to measure again, and again, and again...
- What we care about is how latency behaves
- Behavior is more than “what was the common case?”

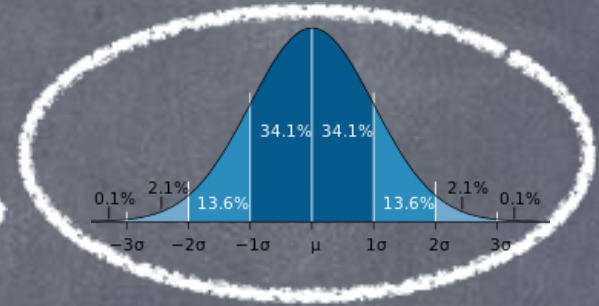
What do you care about?

Do you :

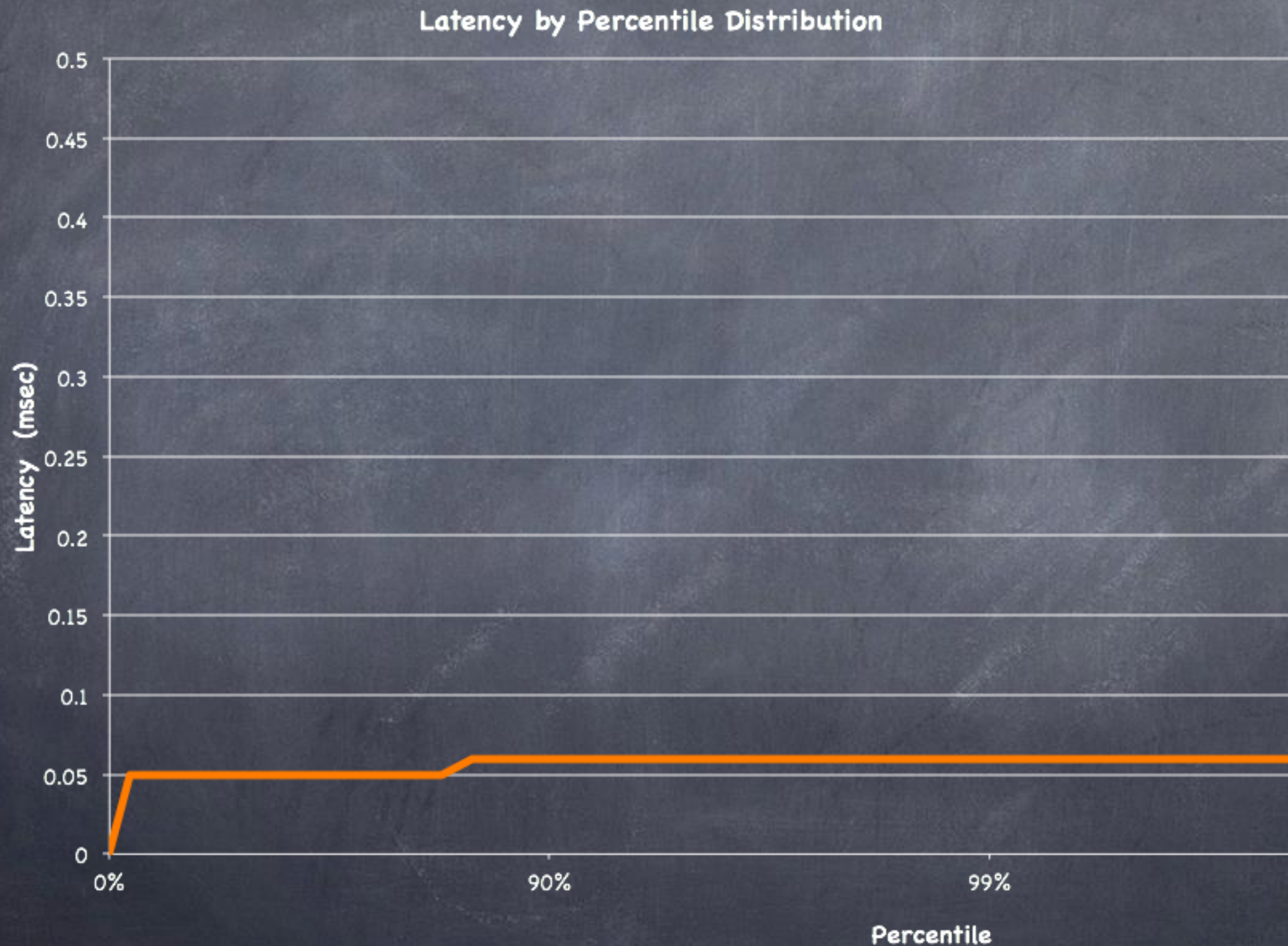
-  Care about latency in your system?
-  Care about the worst case?
-  Care about the 99.99%?
-  Only care about the fastest thing in the day?
-  Only care about the best 50%
-  Only need 90% of operations to meet needs?
-  Care if “only” 1% of operations are painfully slow?
-  Care if 90% of users see outliers every hour?

Latency “wishful thinking”

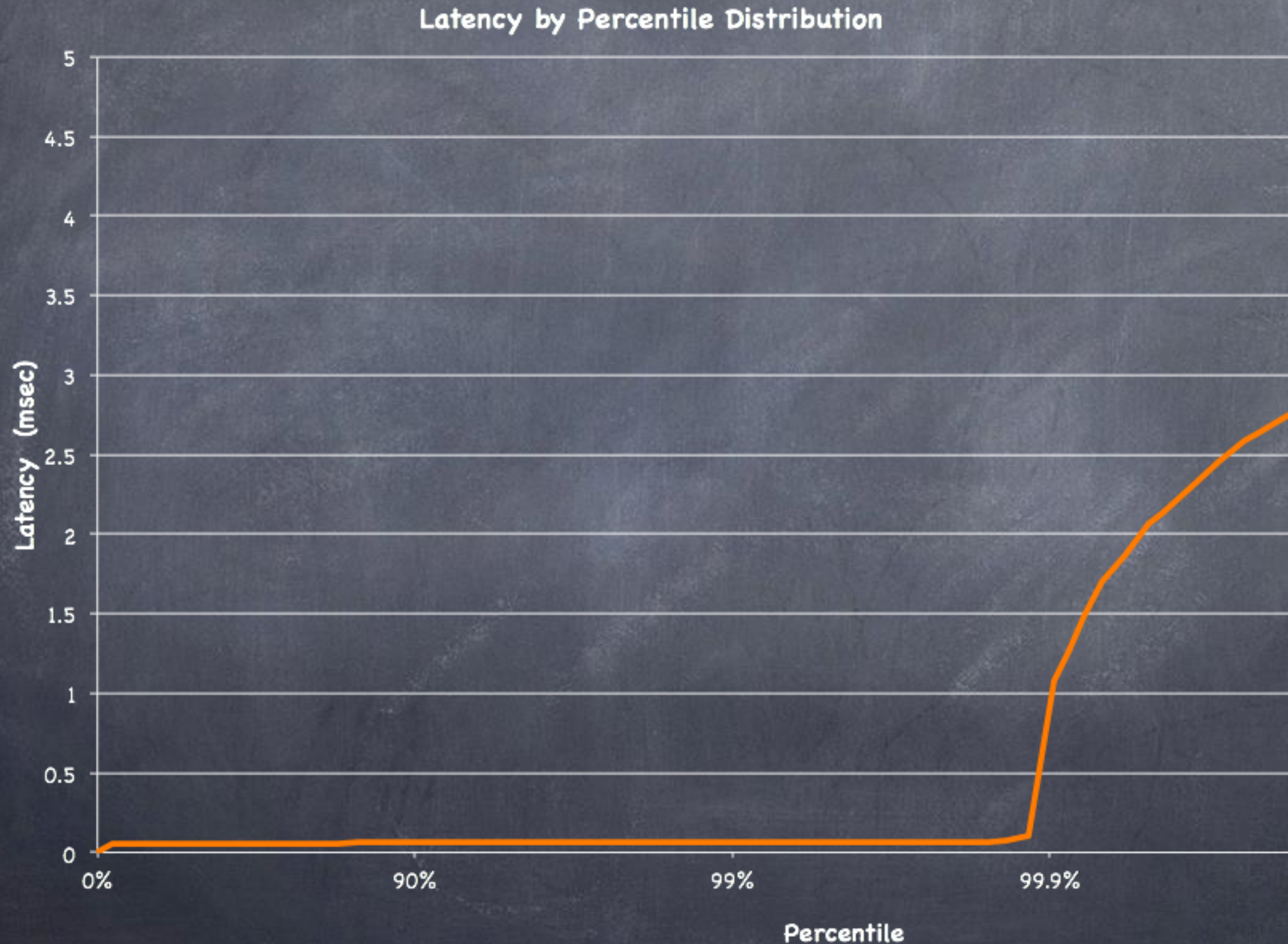
- We know how to compute averages & std. deviation, etc.
- Wouldn't it be nice if latency had a normal distribution?
- The average, 90%, 99%, std. deviation, etc. can give us a “feel” for the rest of the distribution, right?
- If 99% of the stuff behaves well, how bad can the rest be, really?



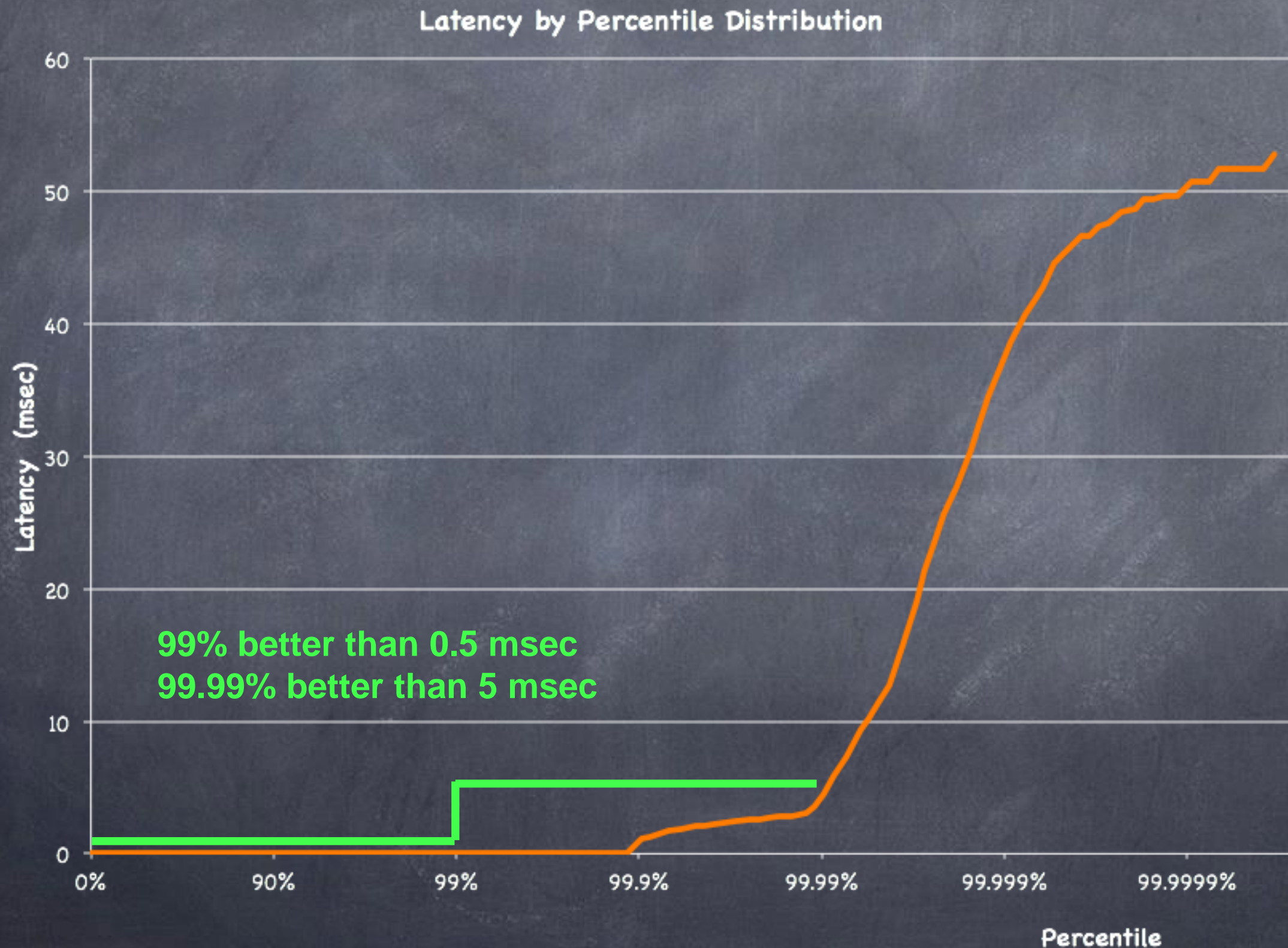
The real world: latency distribution



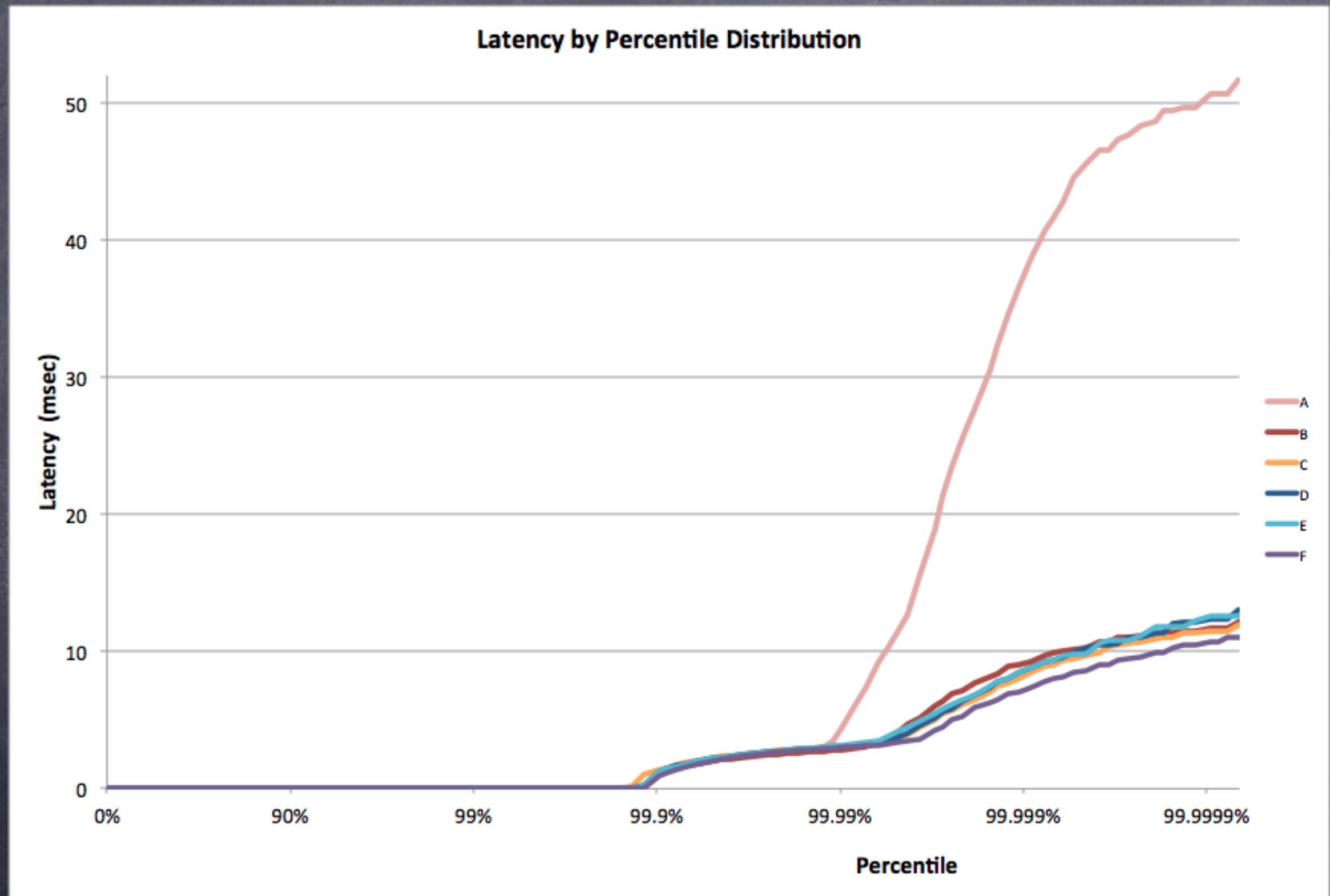
The real world: latency distribution



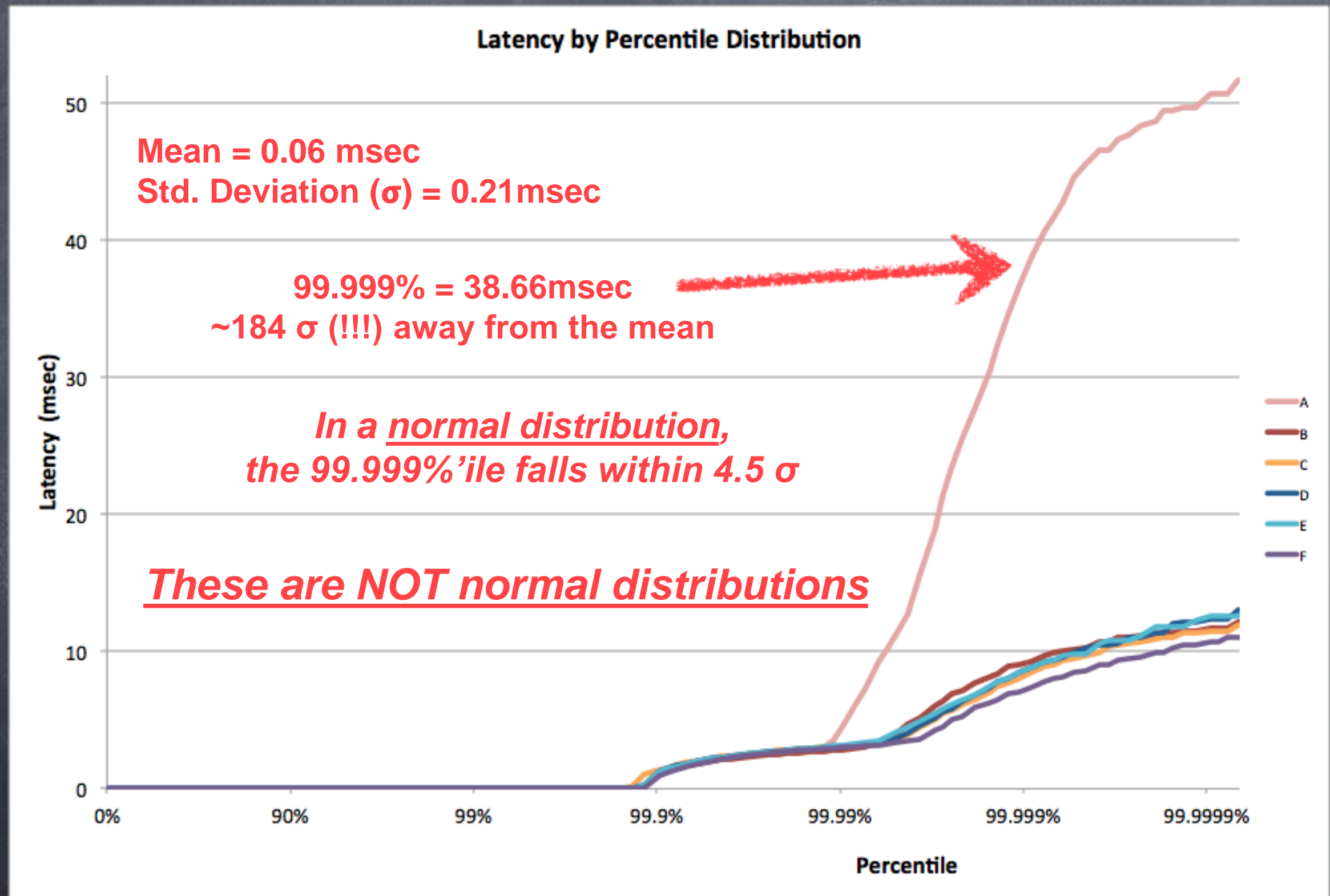
The real world: latency distribution



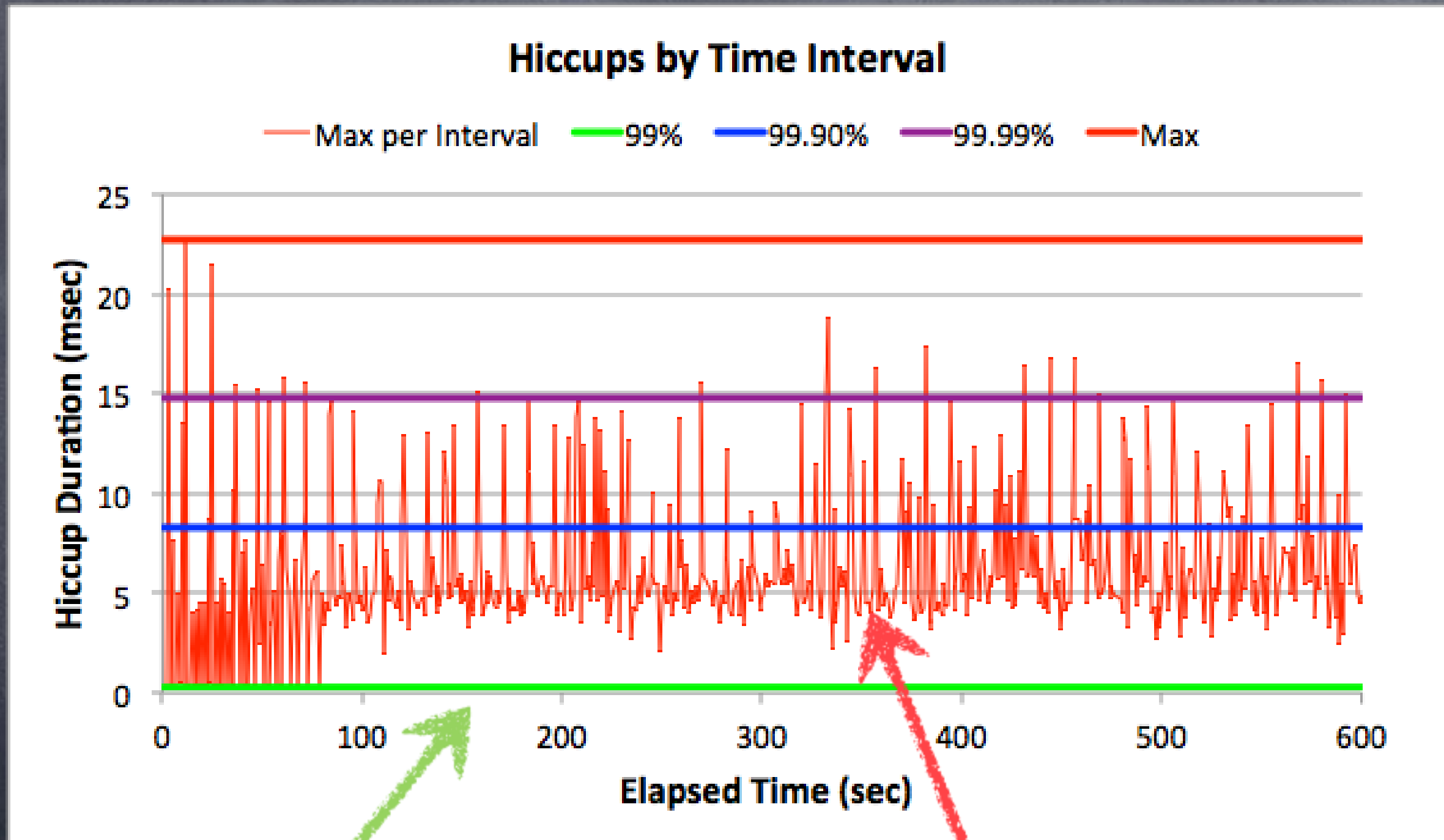
The real world: latency distribution



The real world: latency distribution



The real world: “outliers”



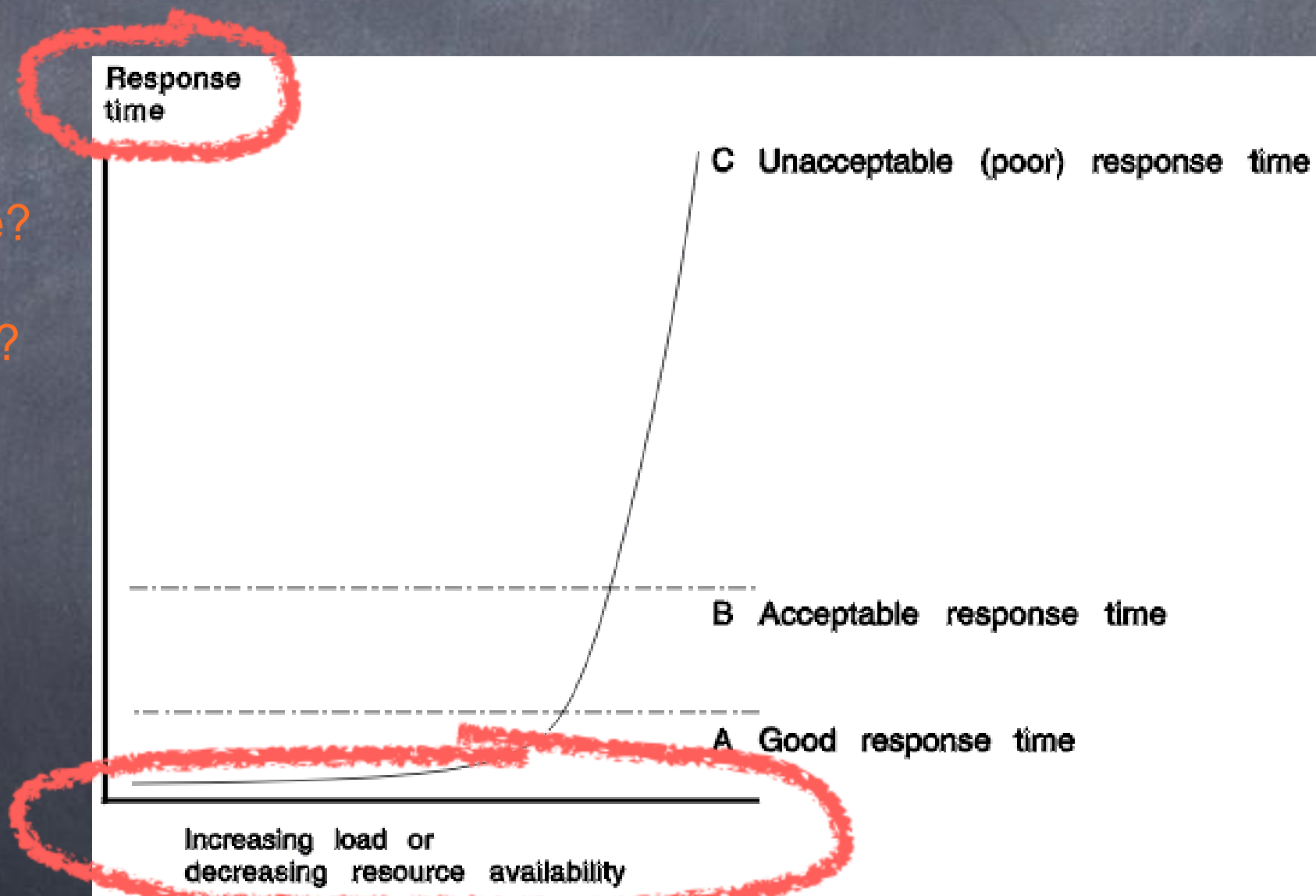
99%ile is ~60 usec

Max is ~30,000% higher than “typical”

A classic look at response time behavior

Response time as a function of load

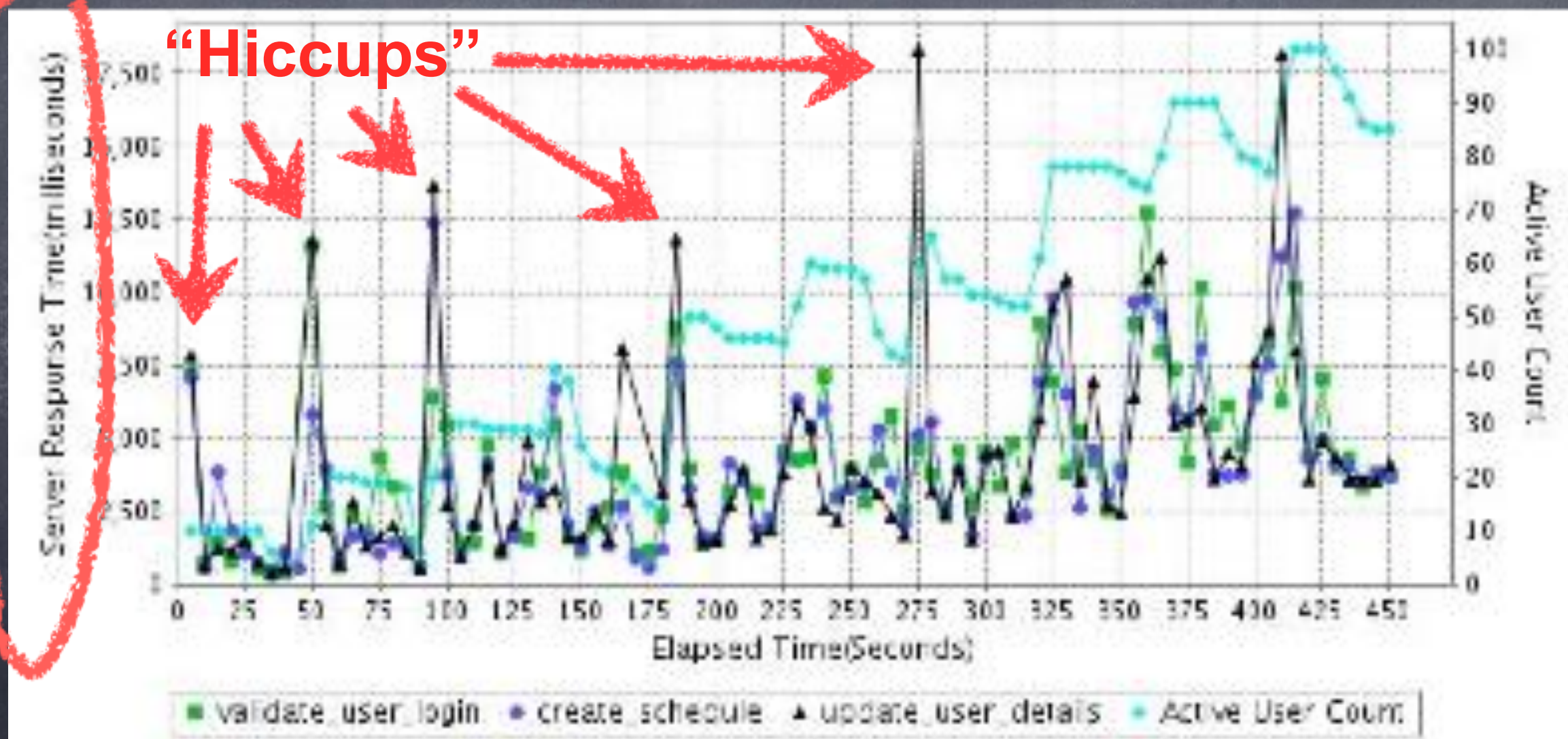
Average?
Max?
Median?
90%?
99.9%



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

Response time over time

When we measure behavior over time, we often see:



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

Hiccups are [typically] strongly multi-modal

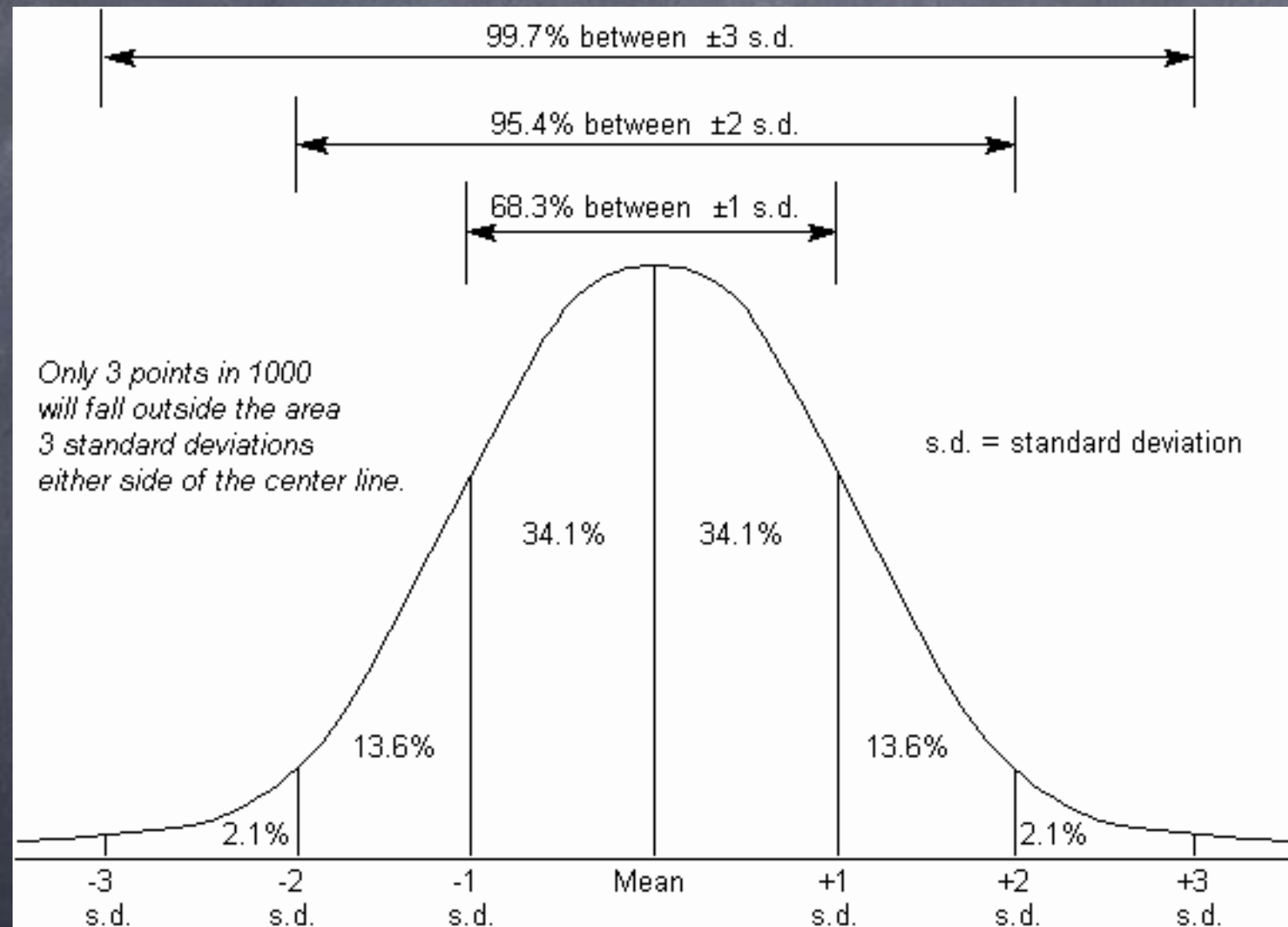
- They don't look anything like a normal distribution
- They usually look like periodic freezes
- A complete shift from one mode/behavior to another
- Mode A: "good".
- Mode B: "Somewhat bad"
- Mode C: "terrible", ...
-

Common ways people deal with hiccups



Common ways people deal with hiccups

Averages and Standard Deviation



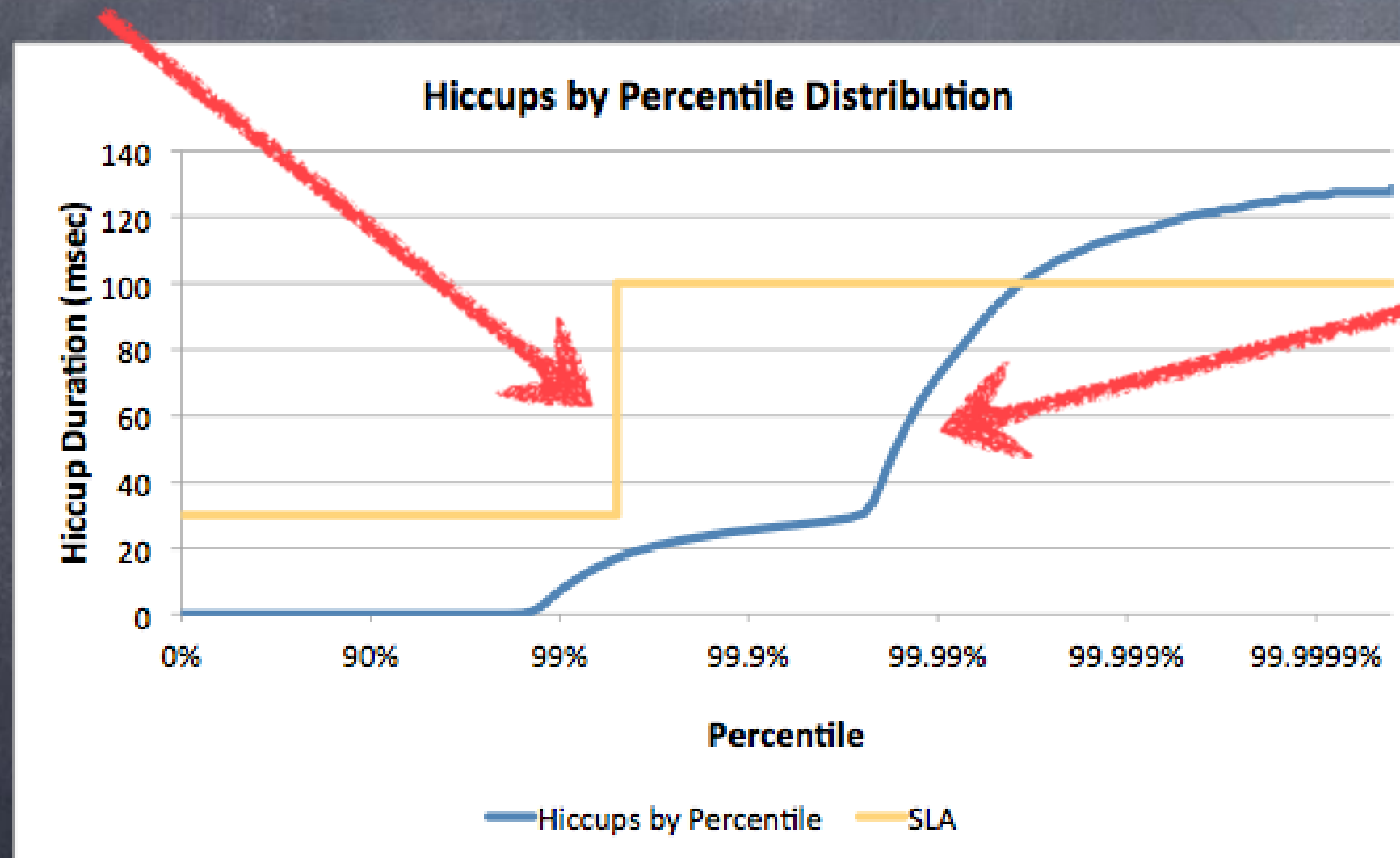
Always Wrong!



Better ways people can deal with hiccups

Actually characterizing latency

Requirements



Response Time
Percentile plot
line



Requirements

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

Latency: Stating Requirements

- Requirements describe how latency should behave
- Useful Latency requirements are usually stated as a PASS/FAIL test against some predefined criteria
- Different applications have different needs
- Requirements should reflect application needs
- Measurements should provide data to evaluate requirements

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..."

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

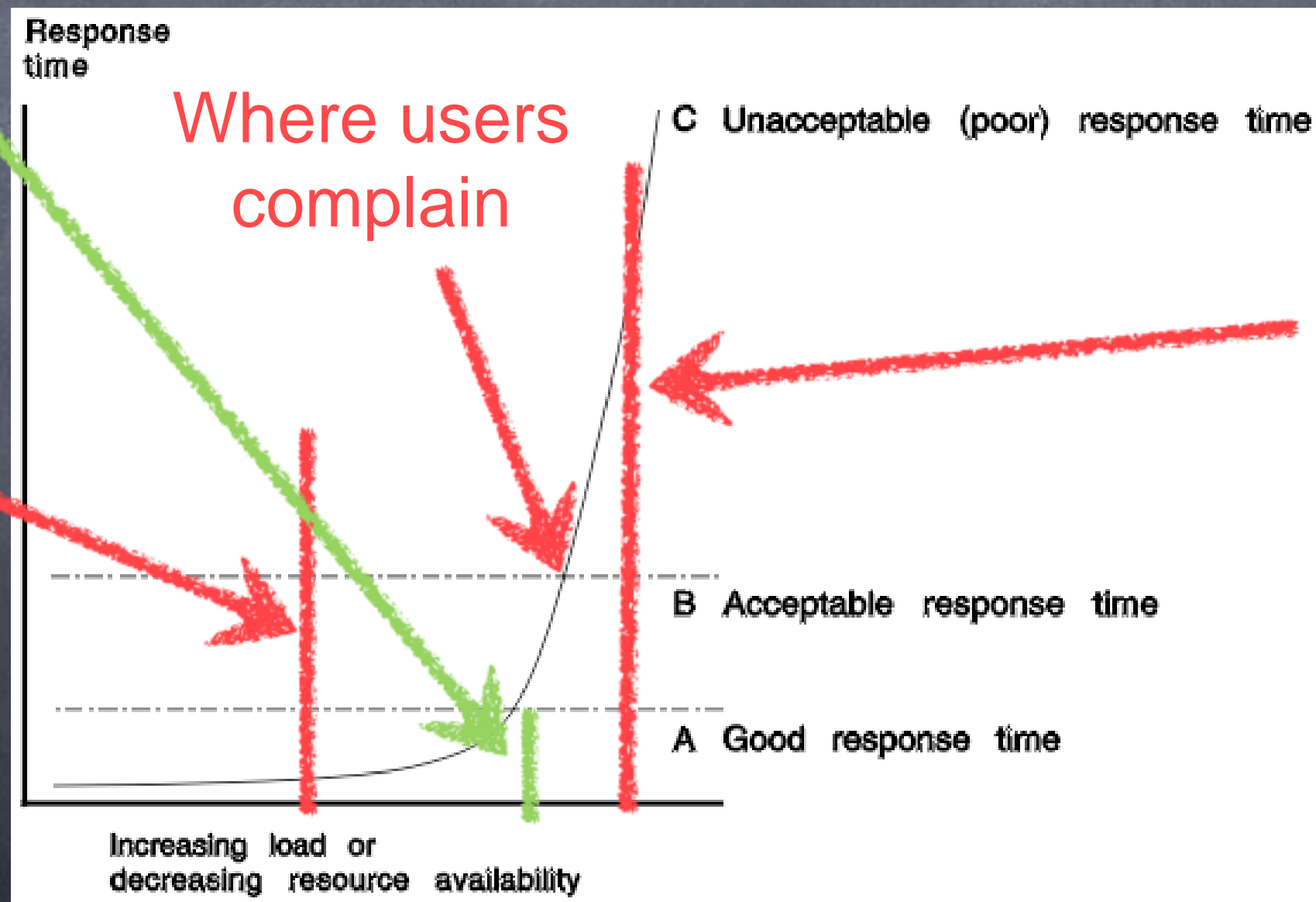
Latency does not live in a vacuum

Remember this?

How much load can this system handle?

Sustainable
Throughput
Level

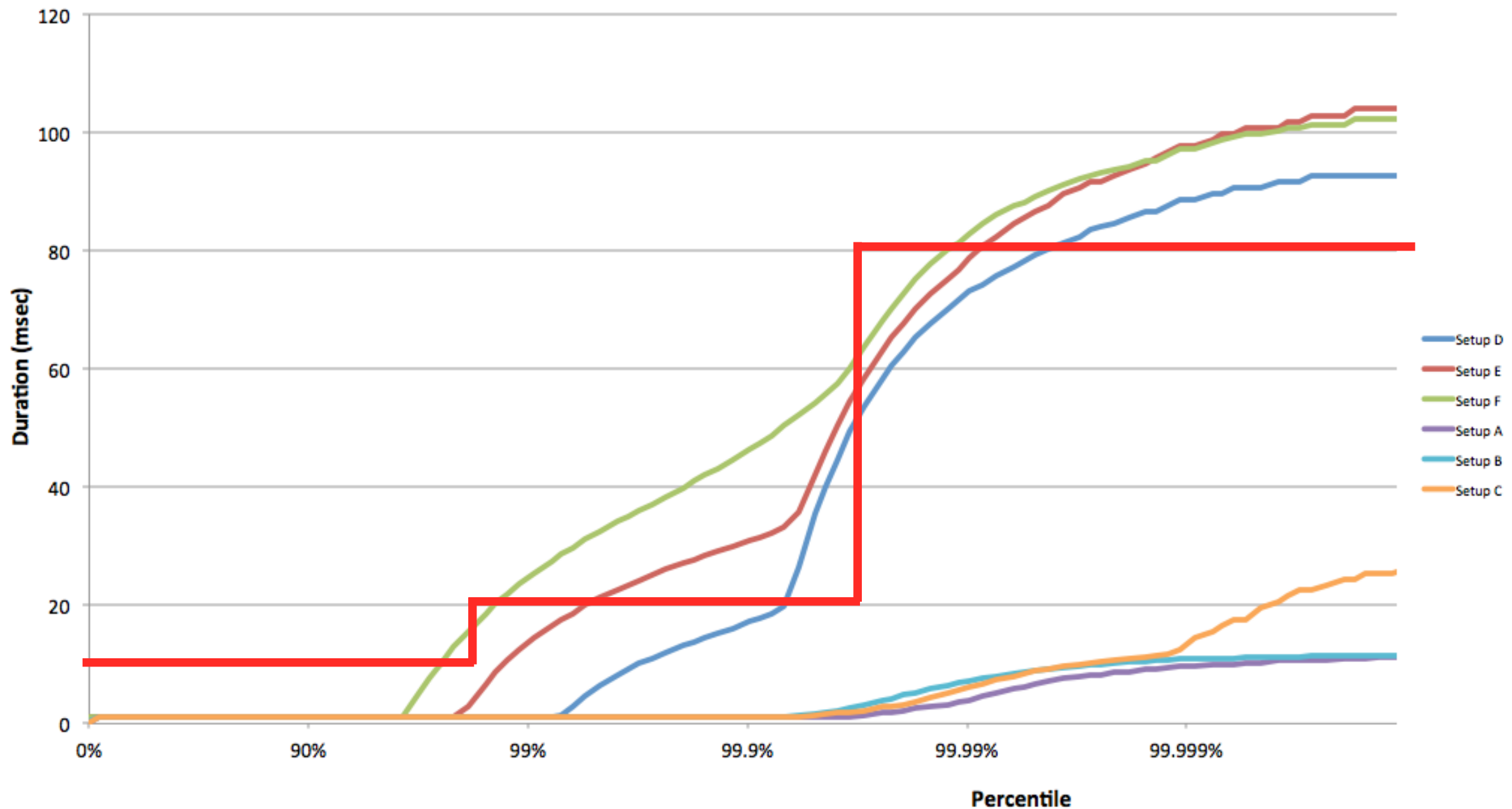
Where the
sysadmin is
willing to go



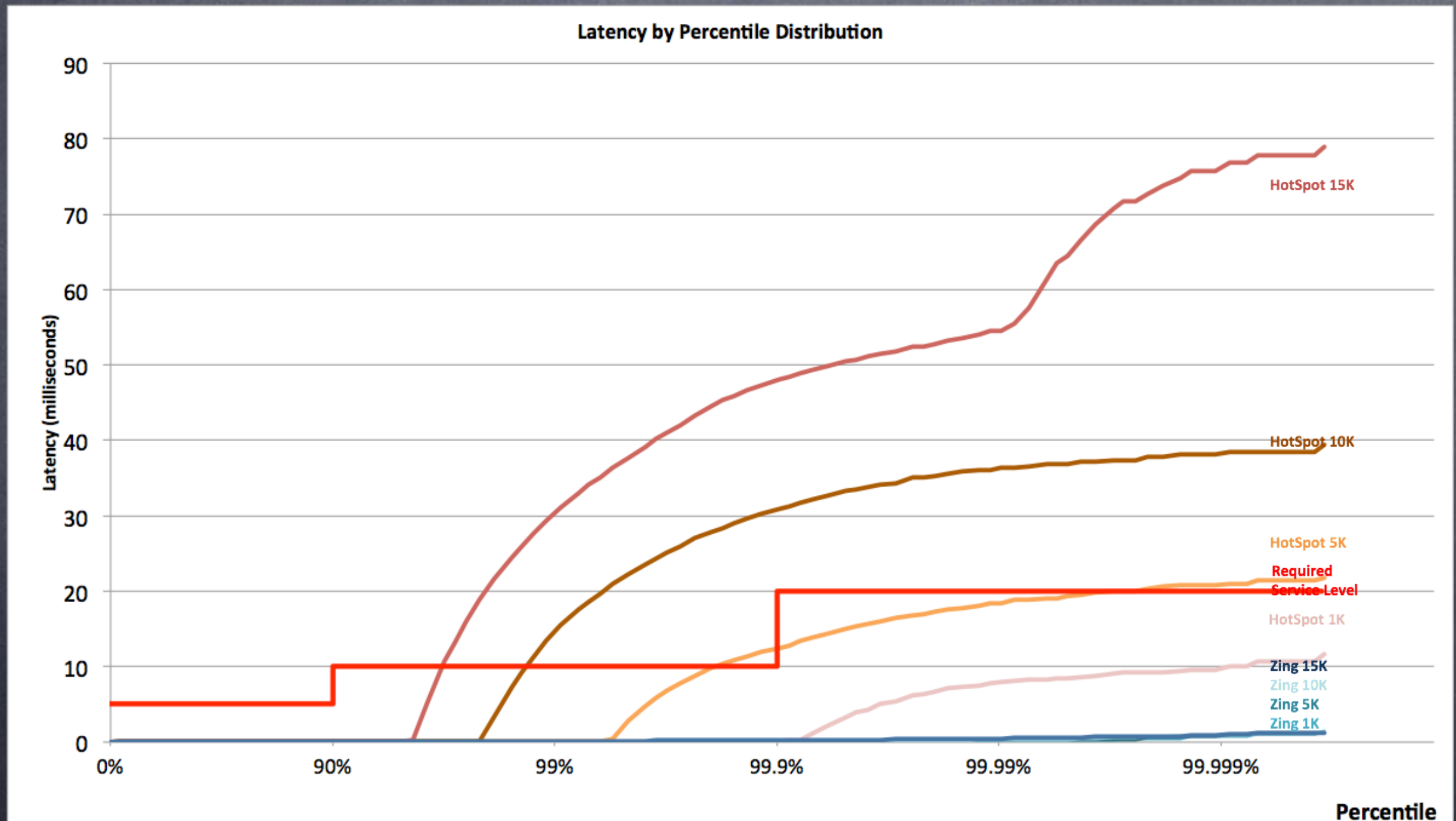
What the
marketing
benchmarks
will say

Comparing behavior under different throughputs and/or configurations

Duration by Percentile Distribution



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



The coordinated omission problem

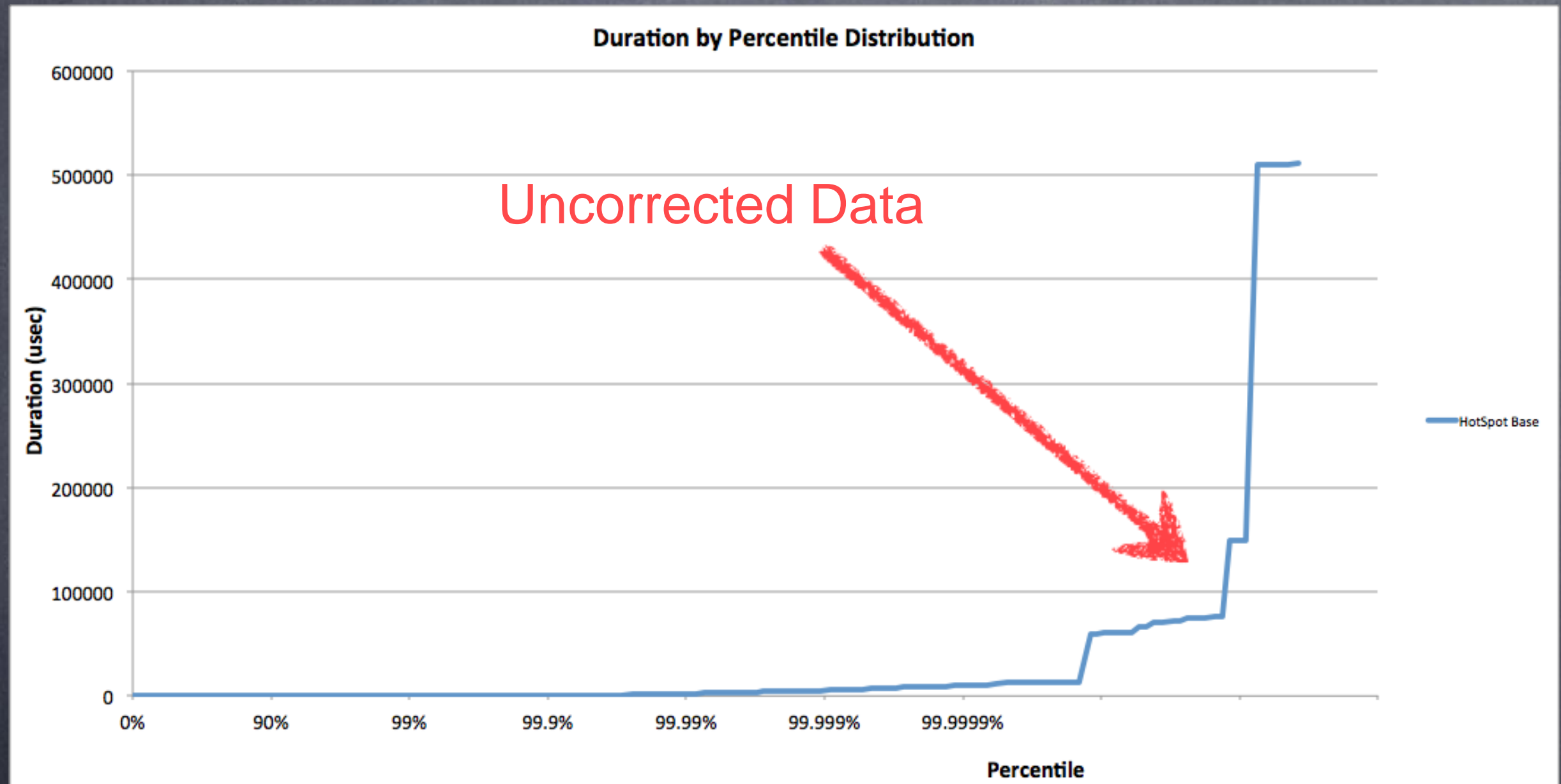
An accidental conspiracy...

Synopsis of Coordinated Omission

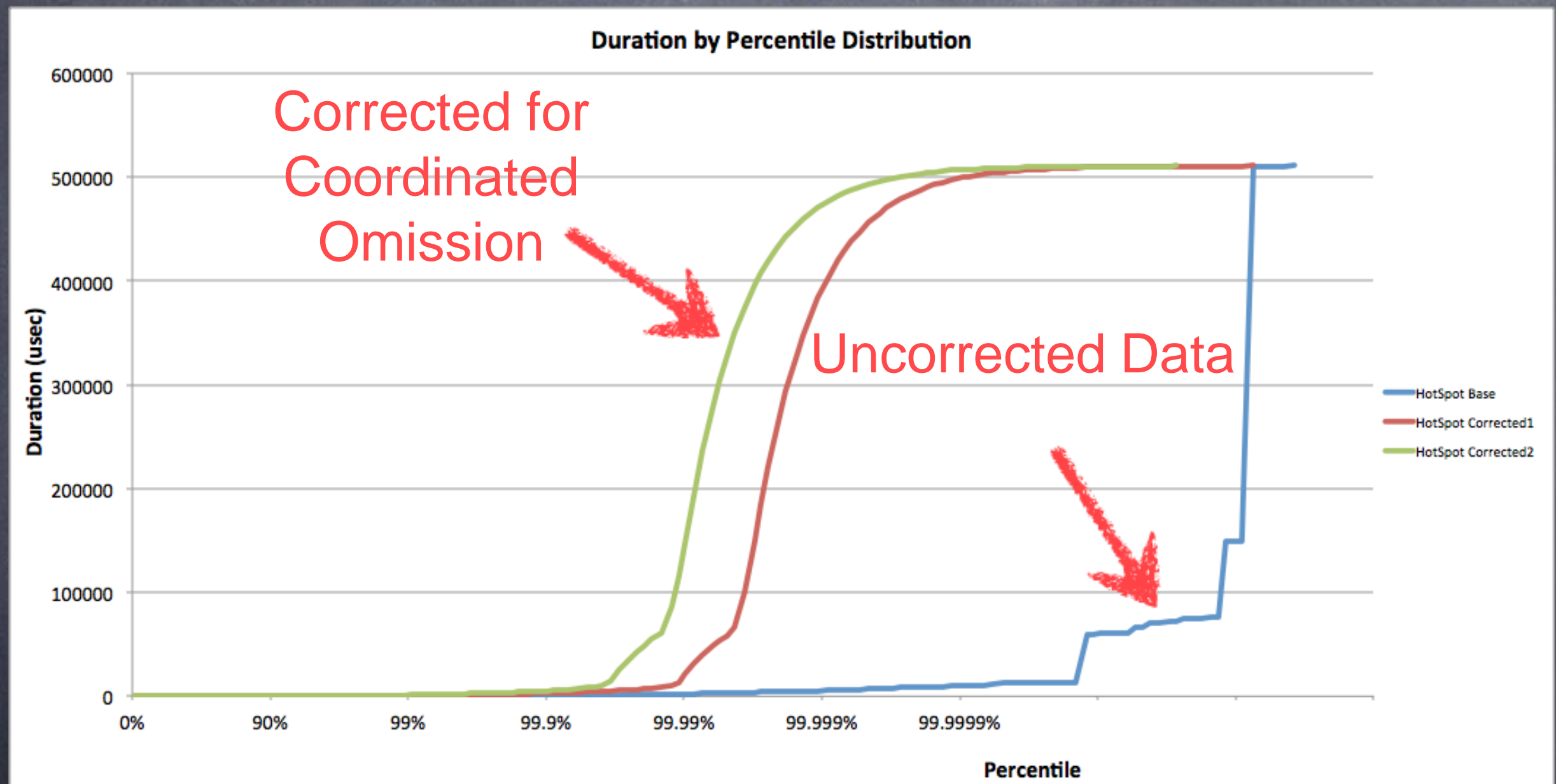
Coordinated Omission (“CO”) is the measurement error which is introduced by **naively recording** requests, sorting them and reporting the result as the percentile distribution of the request latency.

Any recording method, synchronous or asynchronous, which results in even partially coordinating sample times with the system under test, and as a result **avoids recording some of the originally intended samples** will exhibit CO. Synchronous methods tend to naturally exhibit this sort of nativity when intended sample time are not kept far apart from each other.

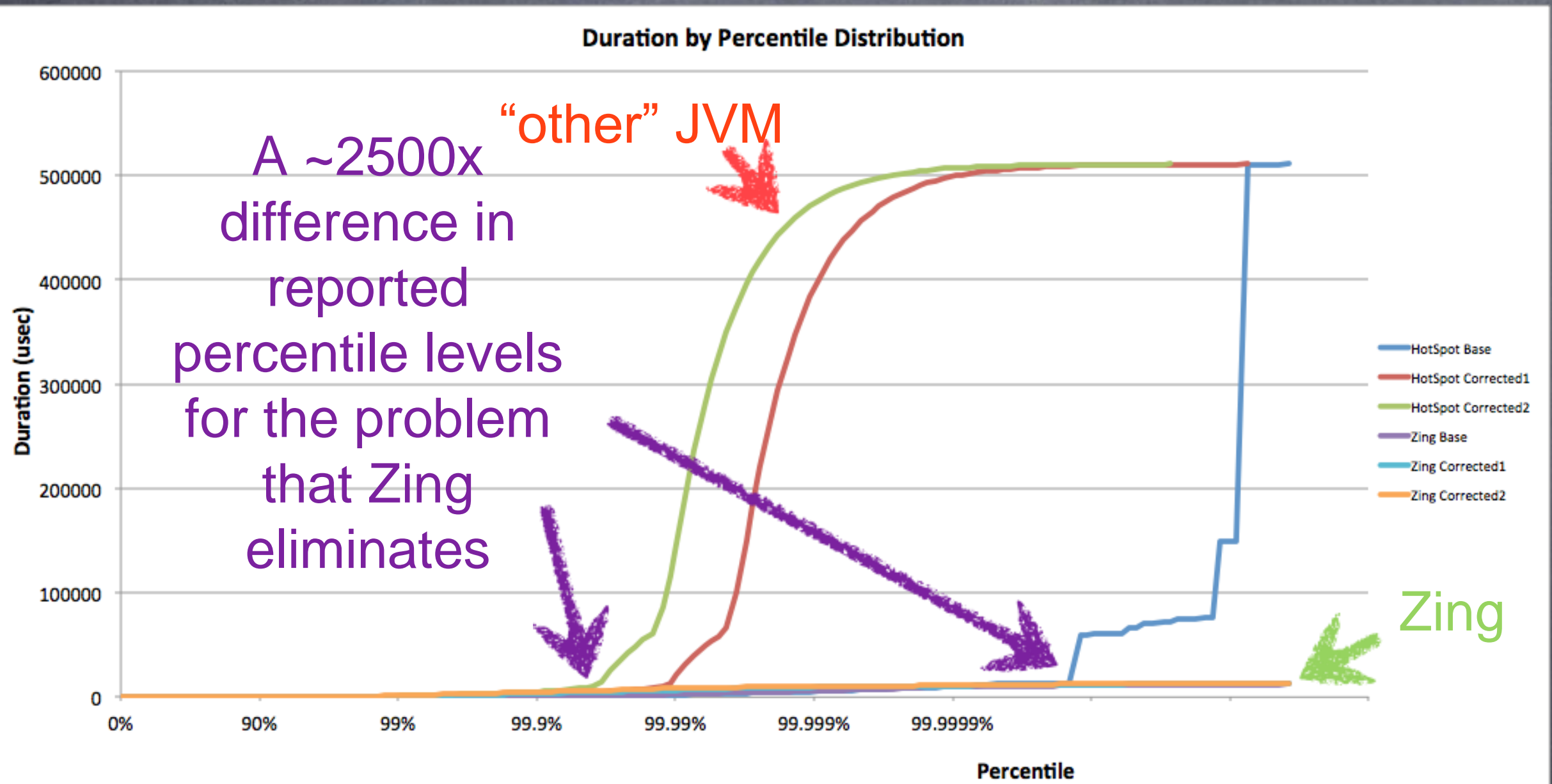
Real World Coordinated Omission effects



Real World Coordinated Omission effects



Real World Coordinated Omission effects (Why I care)



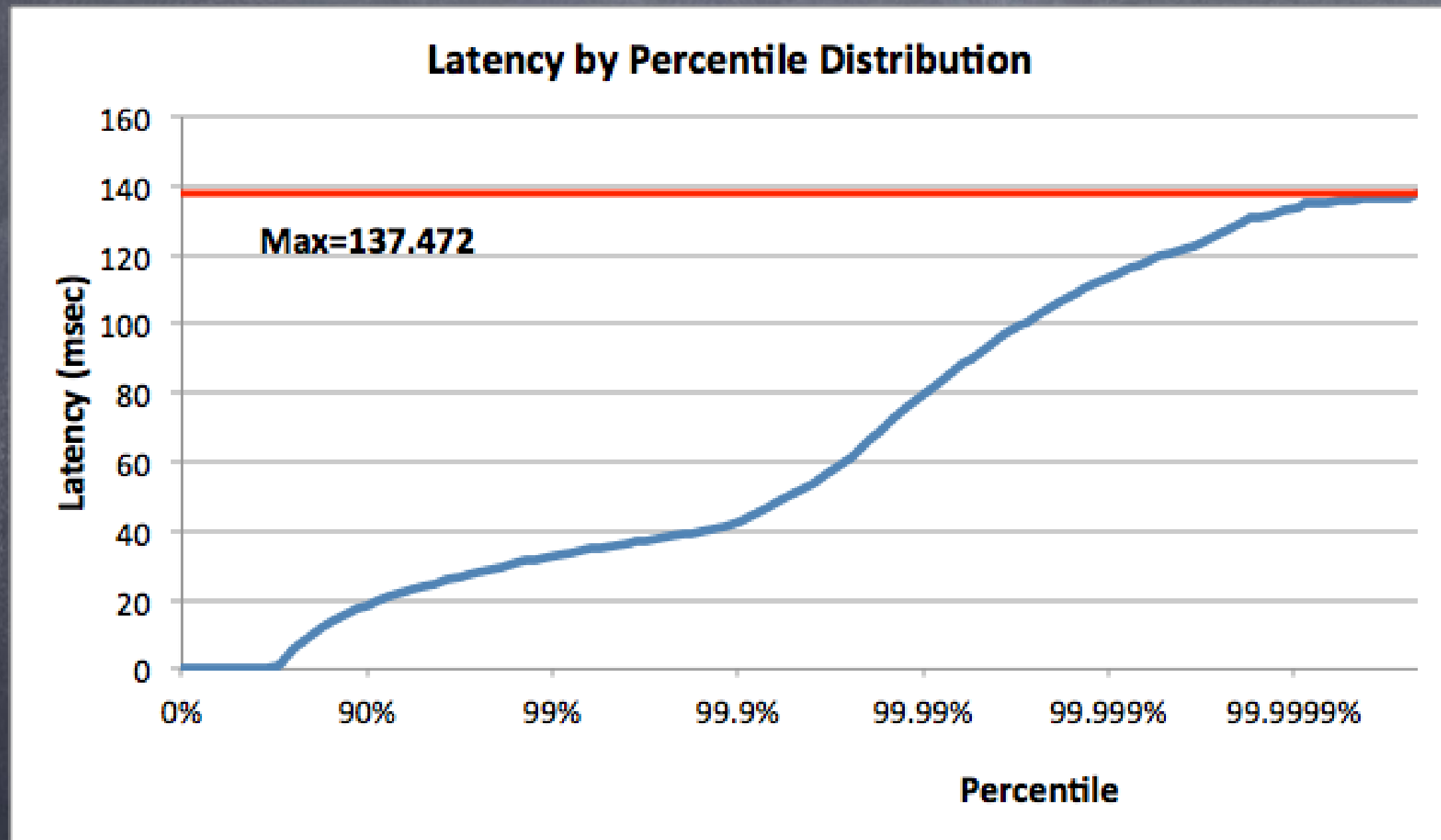
Suggestions

- Whatever your measurement technique is, TEST IT.
- Run your measurement method against artificial systems that create 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.

HdrHistogram

HdrHistogram

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



You need both good dynamic range
and good resolution

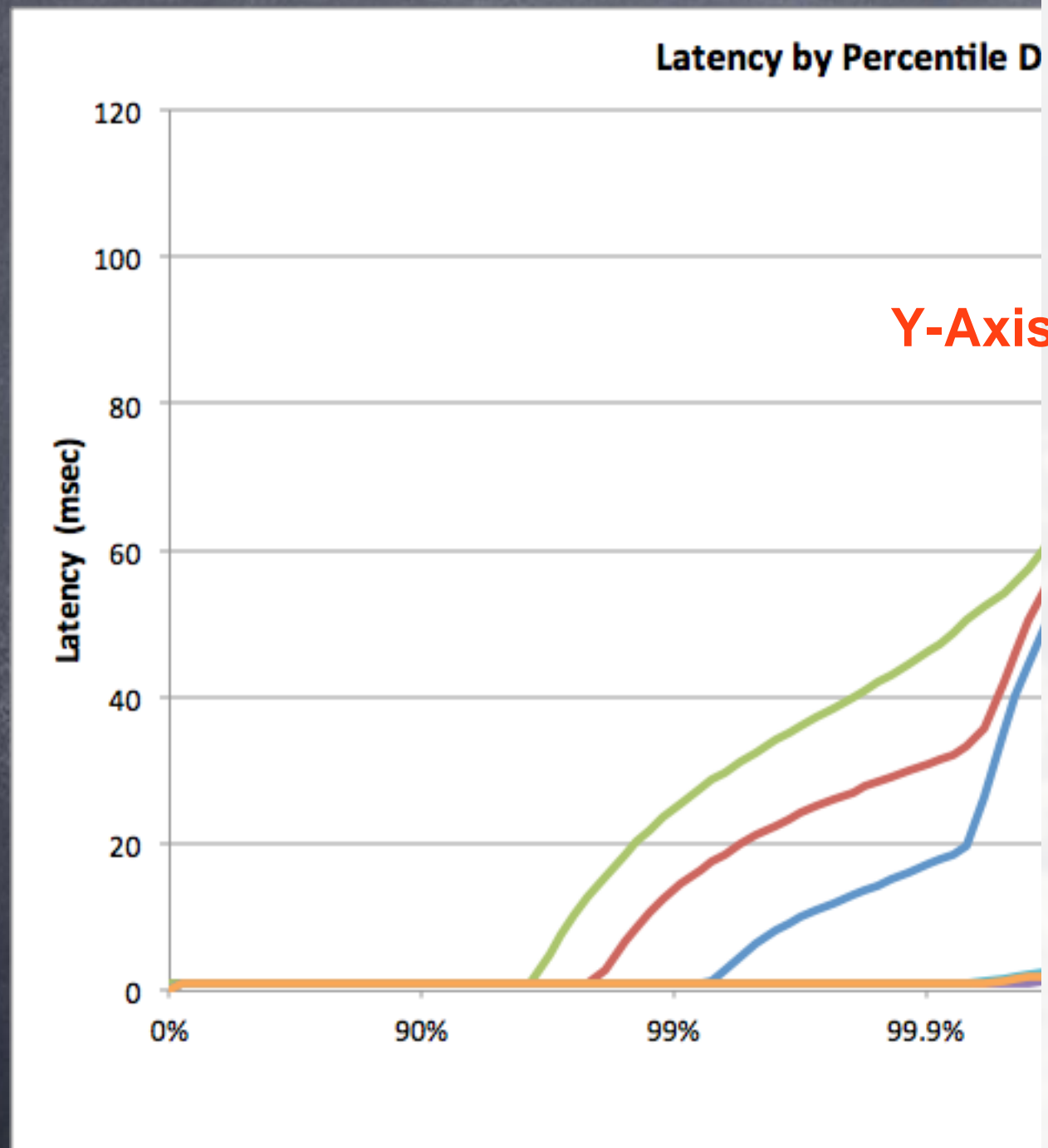
HdrHistogram background

- 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

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

Plotting HdrHistogram output



Value	Percentile	TotalCount	1/(1-Percentile)
0.01	0.000000000000	1	1.00
0.14	0.100000000000	8335	1.11
0.16	0.200000000000	16367	1.25
0.17	0.300000000000	25669	1.43
0.17	0.400000000000	34052	1.67
0.18	0.500000000000	41041	2.00
0.18	0.550000000000	45761	2.22
0.18	0.600000000000	50879	2.50
0.18	0.650000000000	53112	2.86
0.19	0.700000000000	57682	3.33
0.19	0.750000000000	61733	4.00
0.19	0.775000000000	63803	4.44
0.20	0.800000000000	65384	5.00
0.20	0.825000000000	67370	5.71
0.21	0.850000000000	69377	6.67
0.23	0.875000000000	71423	8.00
0.24	0.887500000000	72493	8.89
0.24	0.900000000000	73522	10.00
0.24	0.912500000000	74571	11.43
0.25	0.925000000000	75514	13.33
0.26	0.937500000000	76578	16.00
0.27	0.943750000000	77382	17.78
0.27	0.950000000000	77764	20.00
0.27	0.956250000000	78031	22.86
0.28	0.962500000000	78605	26.67
0.28	0.968750000000	79090	32.00
0.29	0.971875000000	79411	35.56
0.29	0.975000000000	79583	40.00
0.29	0.978125000000	79897	45.71
0.30	0.981250000000	80117	53.33
0.30	0.984375000000	80369	64.00
0.31	0.985937500000	80456	71.11
0.32	0.987500000000	80585	80.00
51.97	0.989062500000	80703	91.43
180.22	0.990625000000	80831	106.67
307.20	0.992187500000	80958	128.00
372.74	0.992968750000	81023	142.22
436.22	0.993750000000	81087	160.00
499.71	0.994531250000	81150	182.86
565.25	0.995312500000	81216	213.33

Value	Percentile	TotalCount	1/(1-Percentile)
0.01	0.000000000000	1	1.00
0.14	0.100000000000	8335	1.11
0.16	0.200000000000	16367	1.25
0.17	0.300000000000	25669	1.43
0.17	0.400000000000	34052	1.67
0.18	0.500000000000	41041	2.00
0.18	0.550000000000	45761	2.22
0.18	0.600000000000	50879	2.50
0.18	0.650000000000	53112	2.86
0.19	0.700000000000	57682	3.33
0.19	0.750000000000	61733	4.00
0.19	0.775000000000	63803	4.44
0.20	0.800000000000	65384	5.00
0.20	0.825000000000	67370	5.71
0.21	0.850000000000	69377	6.67
0.23	0.875000000000	71423	8.00
0.24	0.887500000000	72493	8.89
0.24	0.900000000000	73522	10.00
0.24	0.912500000000	74571	11.43
0.25	0.925000000000	75514	13.33
0.26	0.937500000000	76578	16.00
0.27	0.943750000000	77382	17.78
0.27	0.950000000000	77764	20.00
0.27	0.956250000000	78031	22.86
0.28	0.962500000000	78605	26.67
0.28	0.968750000000	79090	32.00
0.29	0.971875000000	79411	35.56
0.29	0.975000000000	79583	40.00
0.29	0.978125000000	79897	45.71
0.30	0.981250000000	80117	53.33
0.30	0.984375000000	80369	64.00
0.31	0.985937500000	80456	71.11
0.32	0.987500000000	80585	80.00
51.97	0.989062500000	80703	91.43
180.22	0.990625000000	80831	106.67
307.20	0.992187500000	80958	128.00
372.74	0.992968750000	81023	142.22
436.22	0.993750000000	81087	160.00
499.71	0.994531250000	81150	182.86
565.25	0.995312500000	81216	213.33

Convenient for plotting and comparing test results

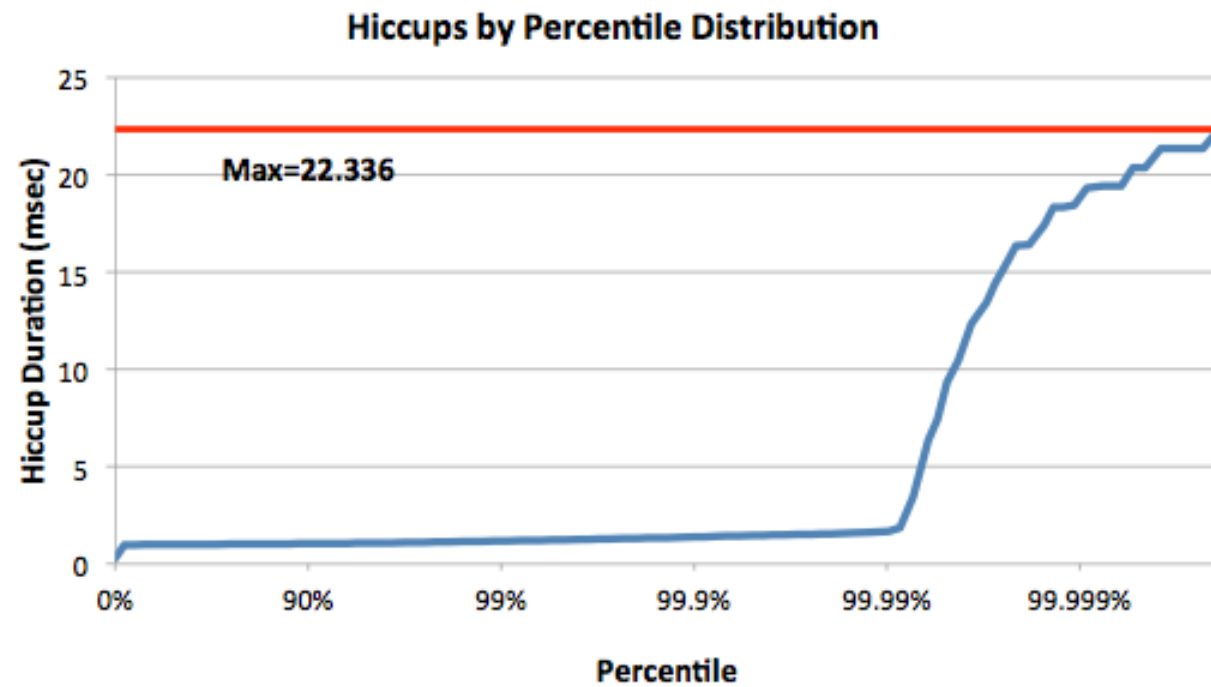
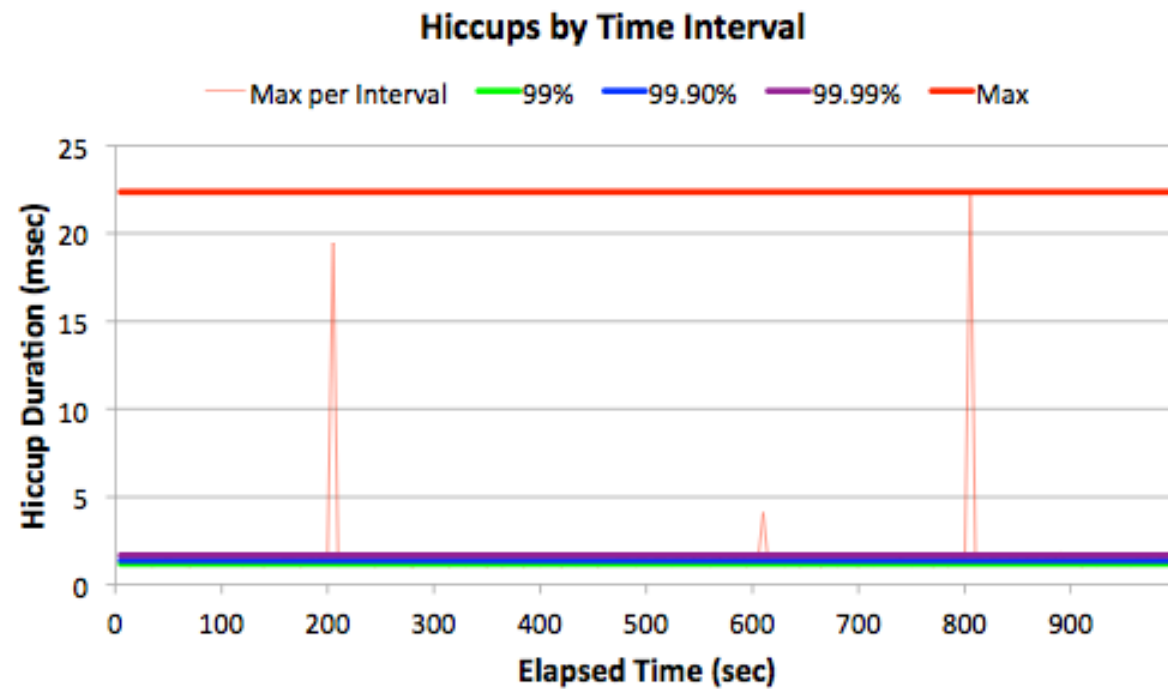
jHiccup

jHiccup

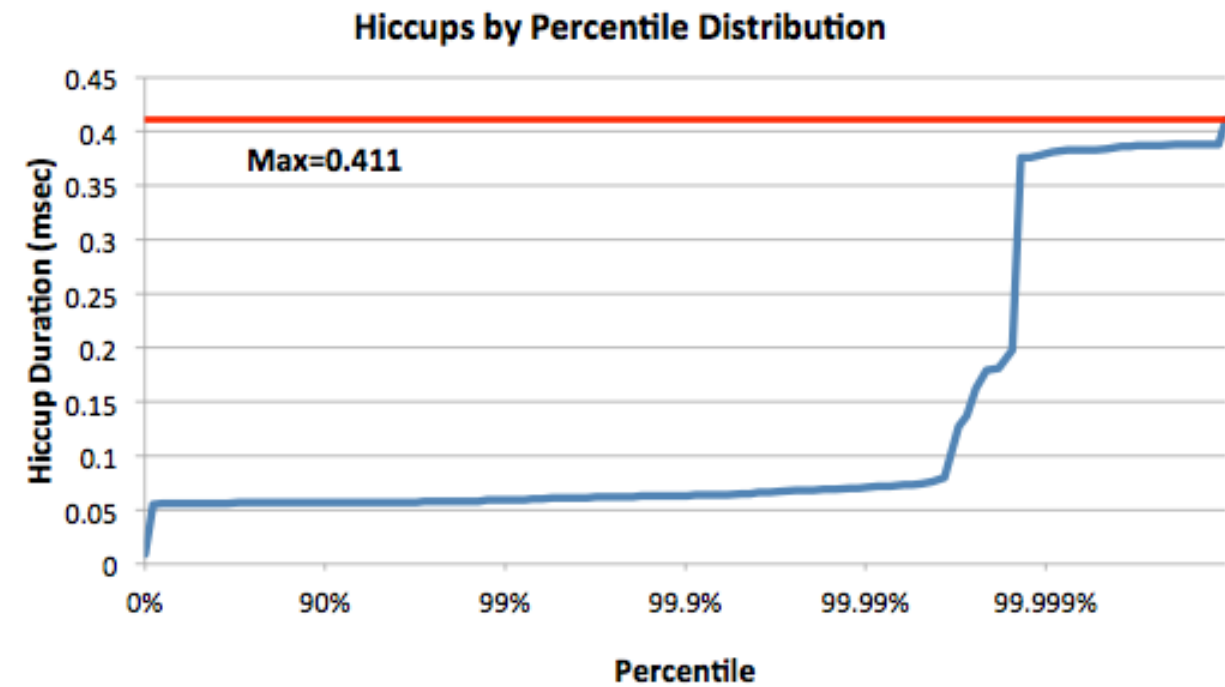
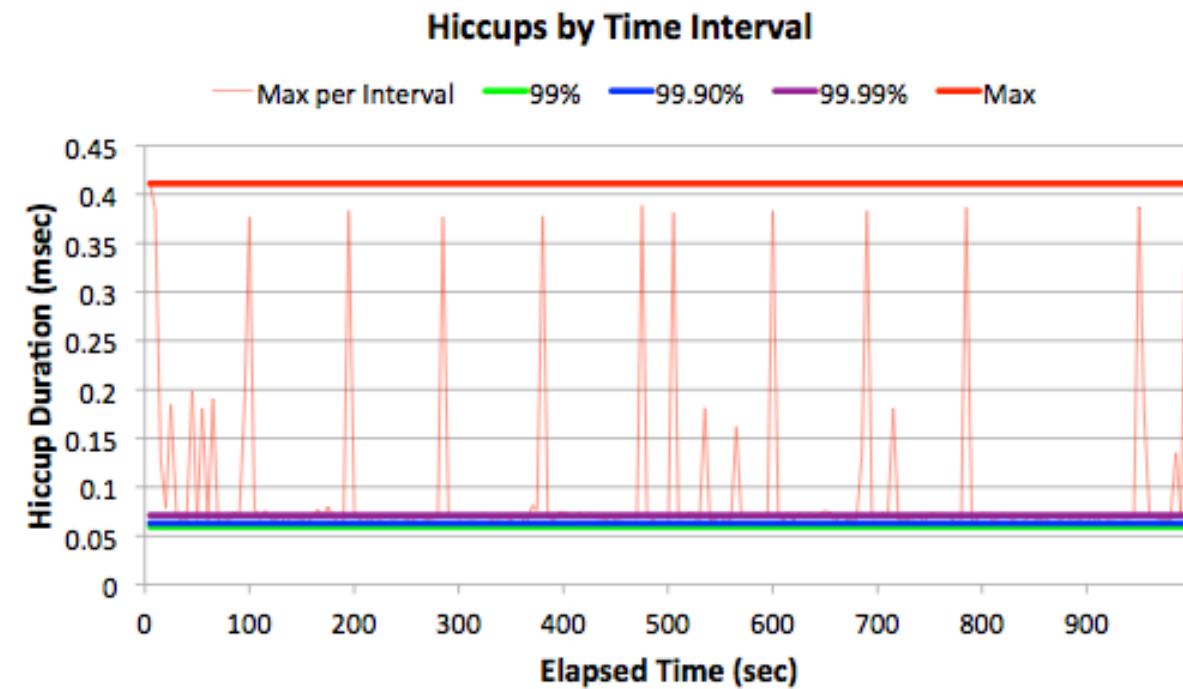
- A tool for capturing and displaying platform hiccups
 - Records any observed non-continuity of the underlying platform
 - Plots results in simple, consistent format
- Simple, non-intrusive
 - As simple as adding jHiccup.jar as a java agent:
 - `% java -javaagent=jHiccup.jar myApp myflags`
 - or attaching 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

Examples

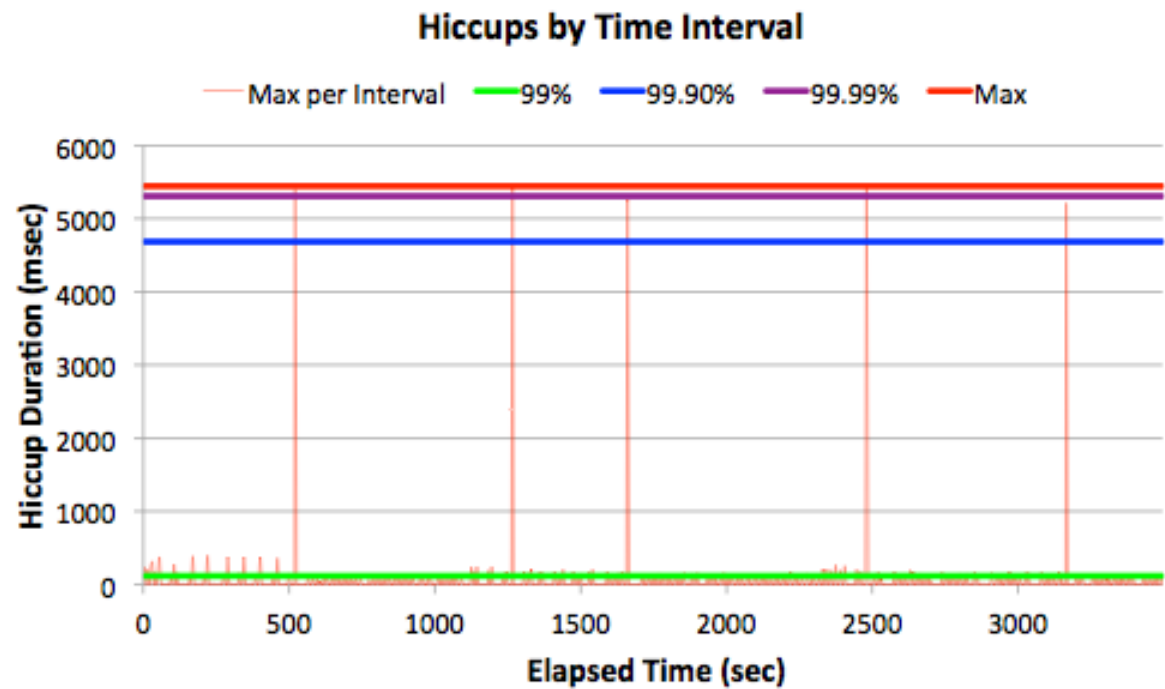
Idle App on Quiet System



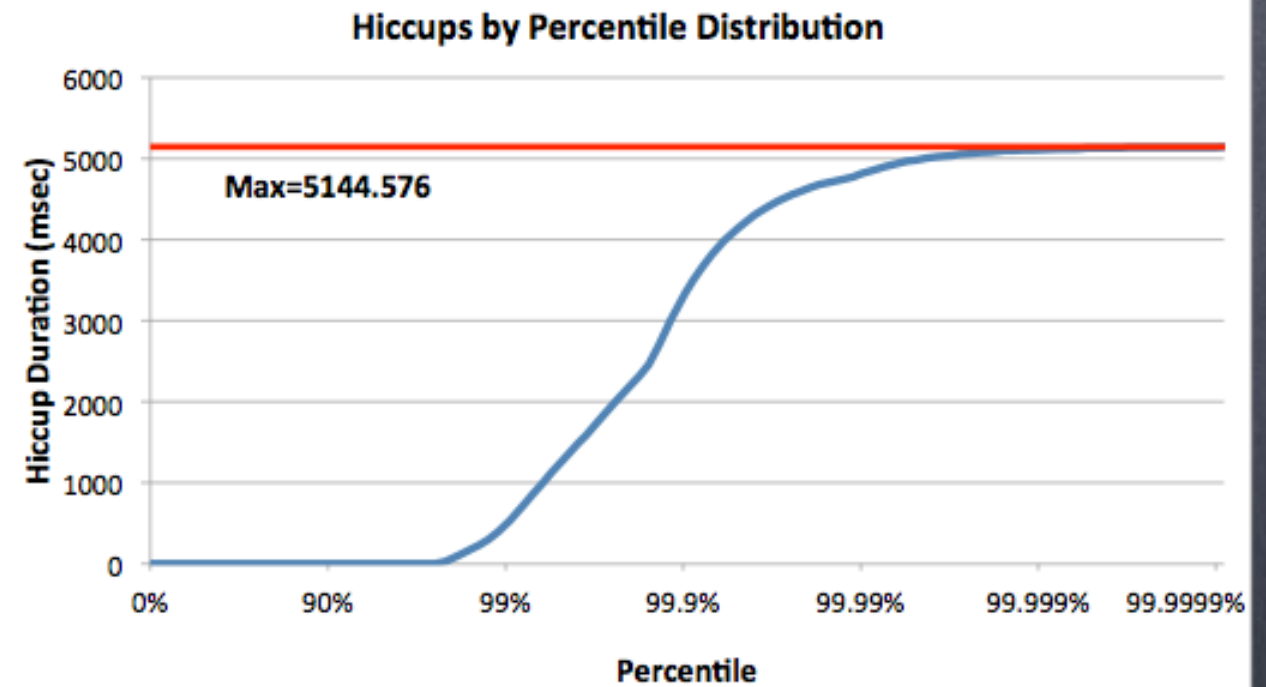
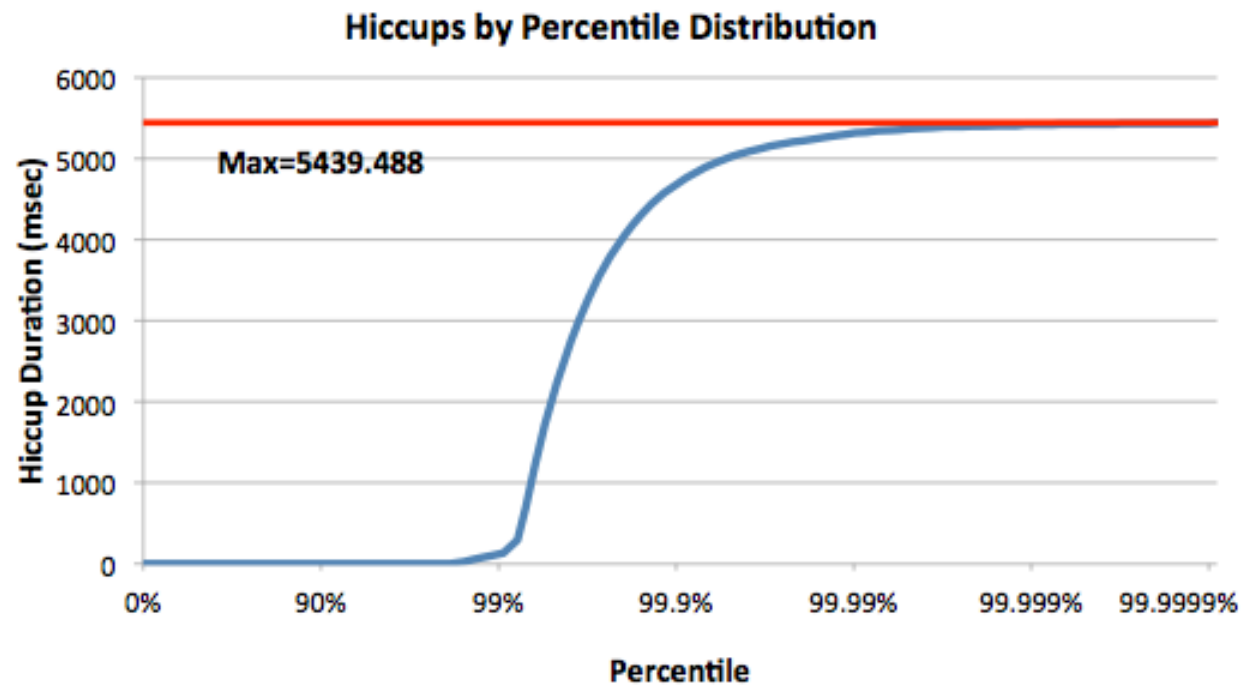
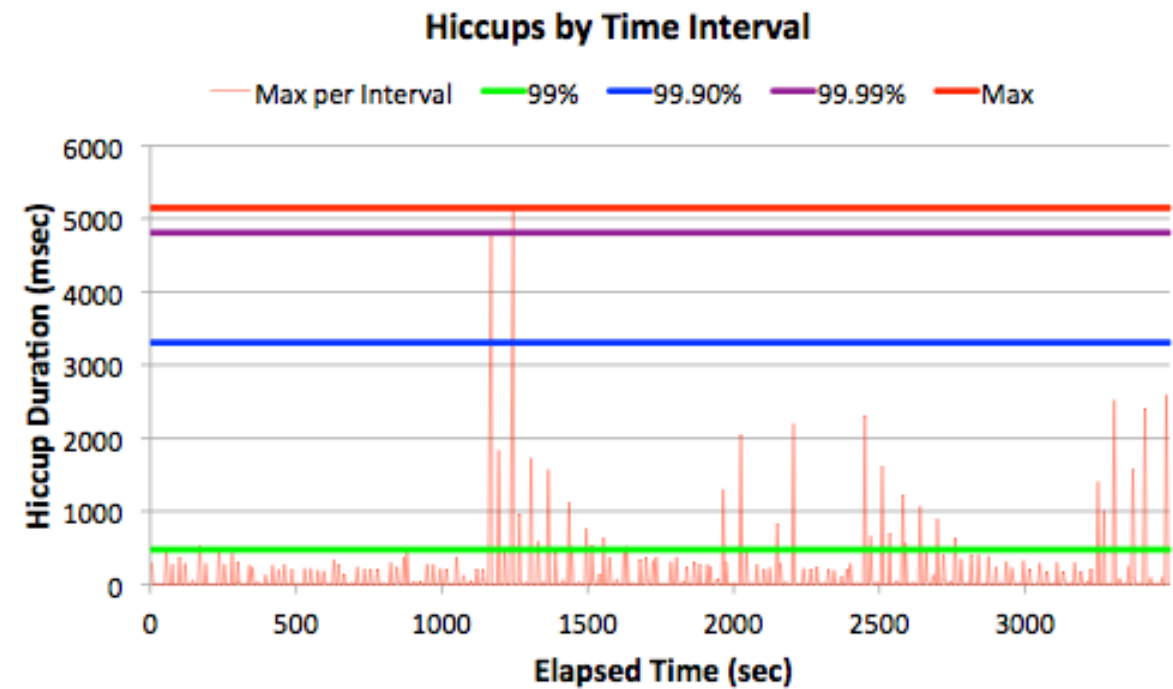
Idle App on Dedicated System



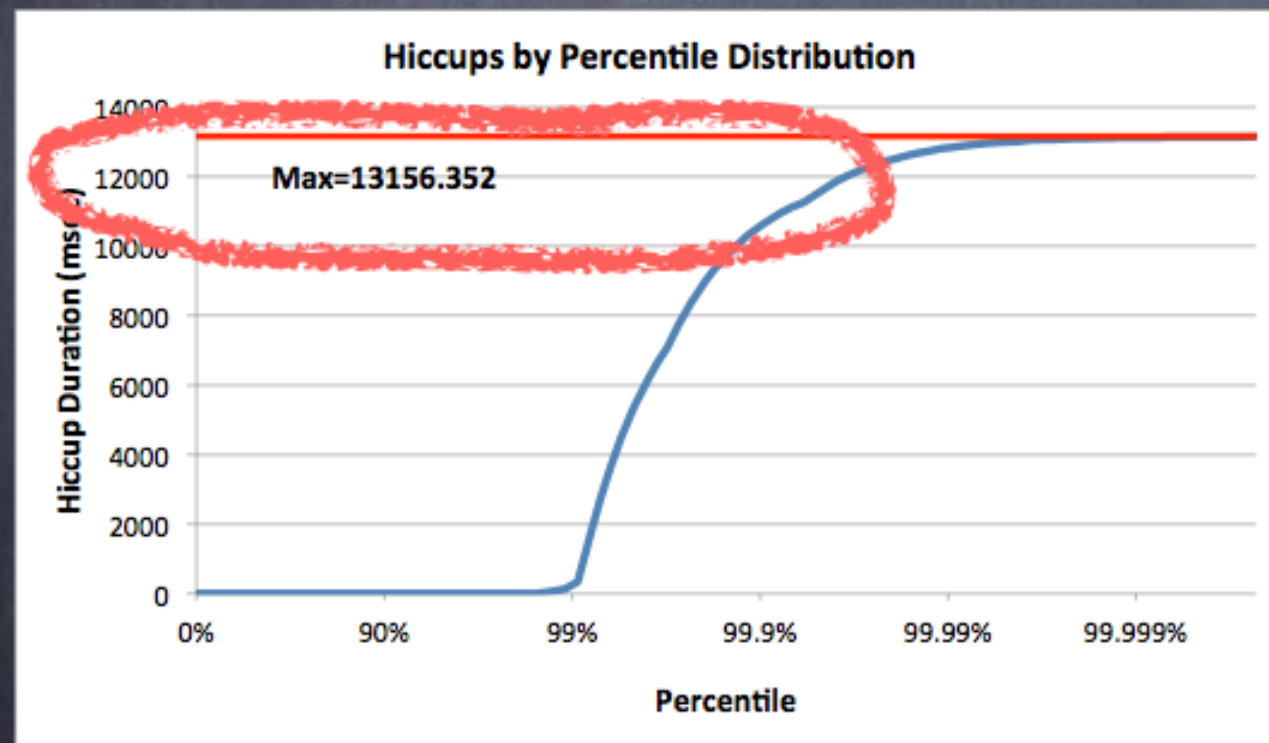
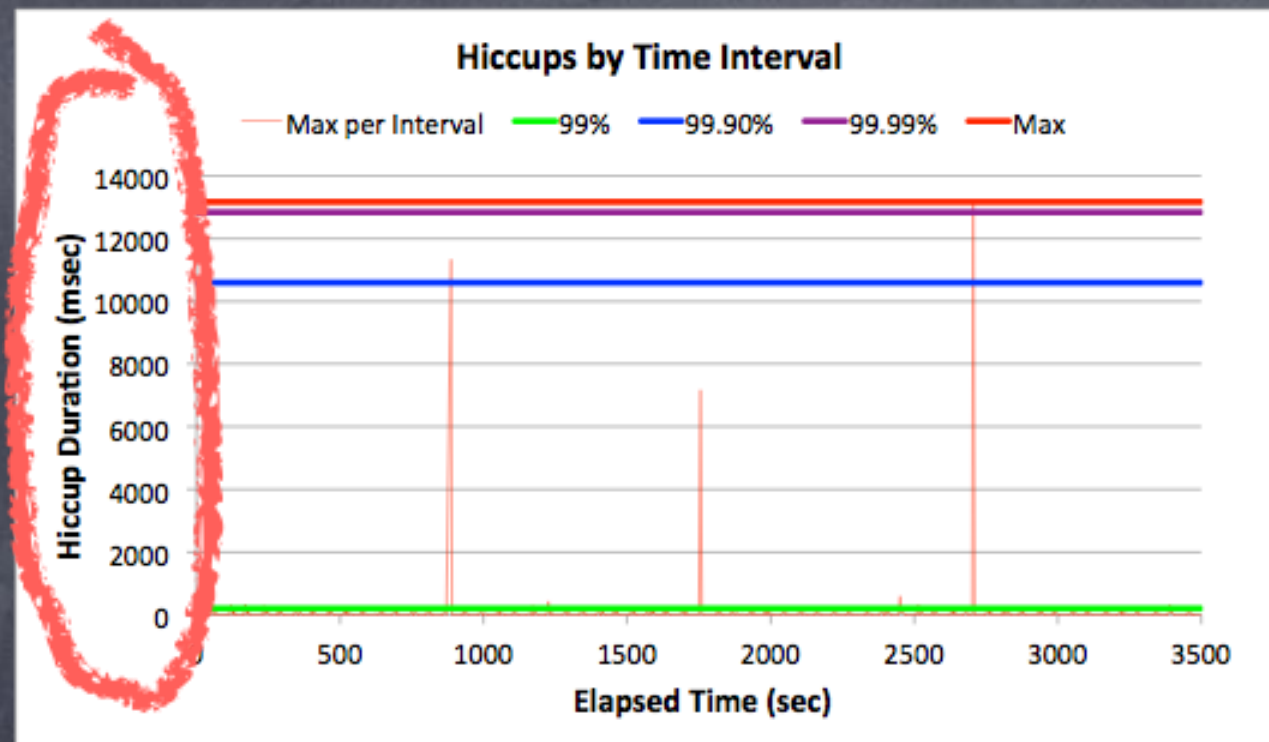
Oracle HotSpot ParallelGC, 1GB in 8GB heap



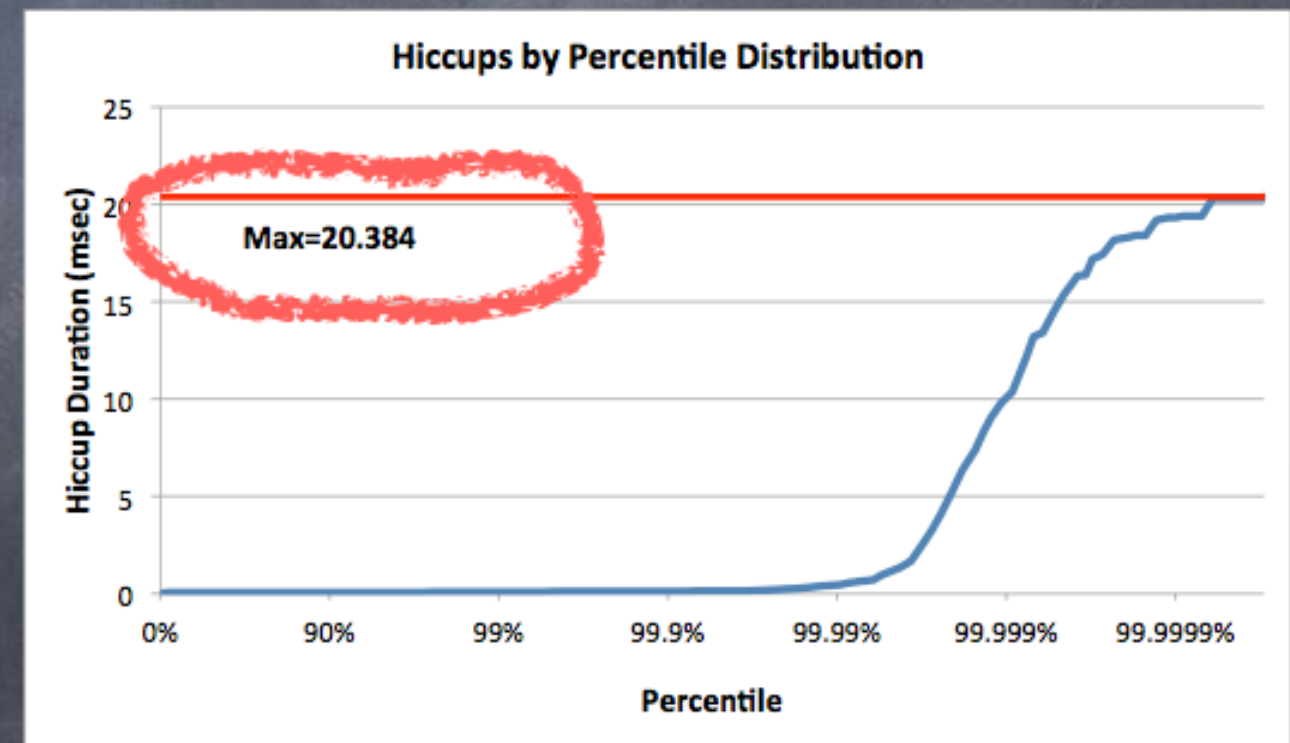
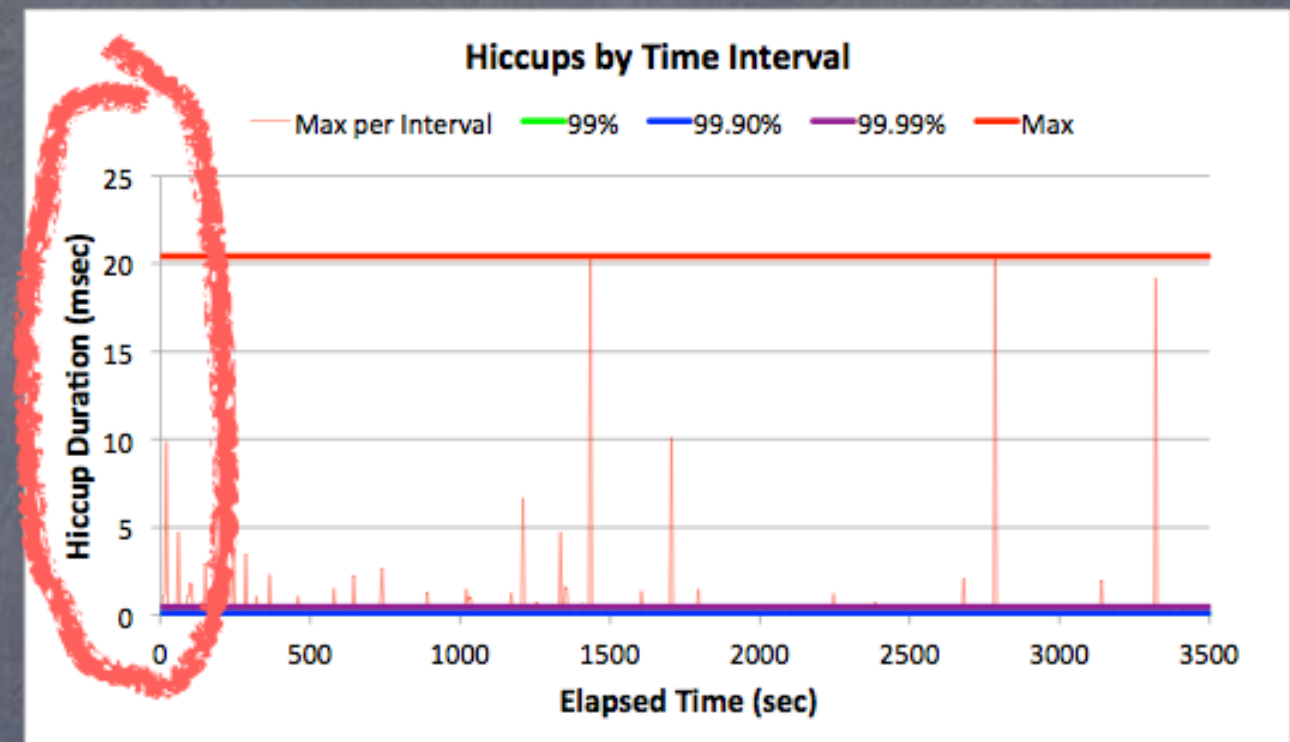
Oracle HotSpot G1, 1GB in 8GB heap



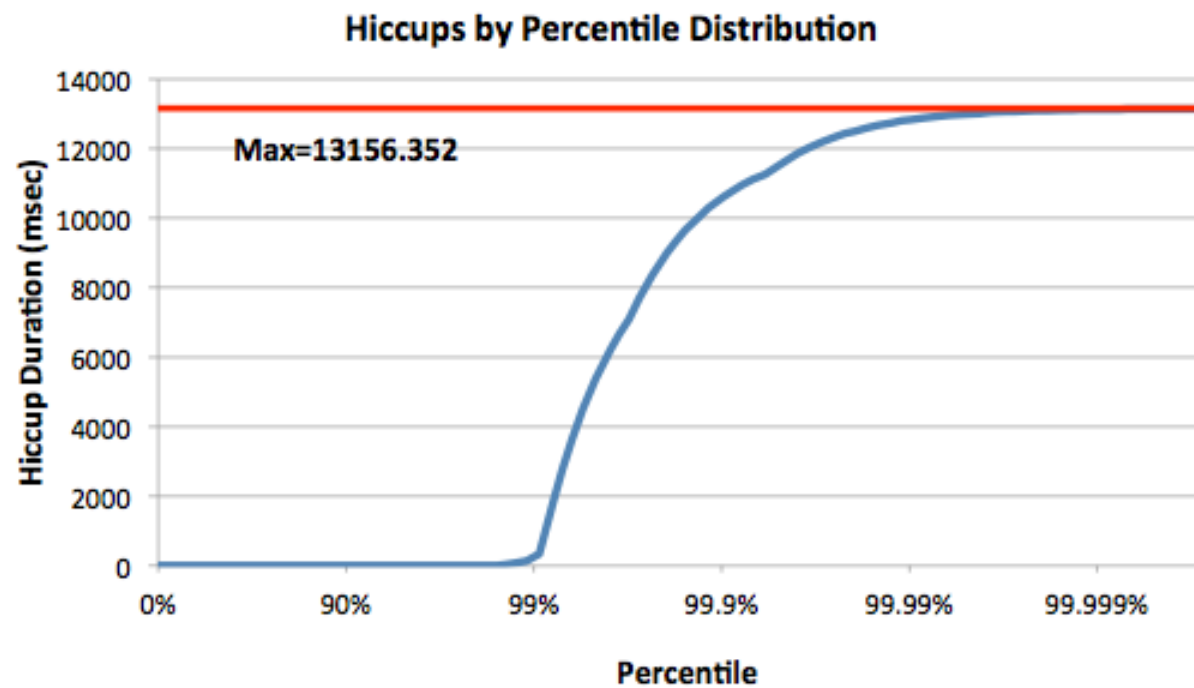
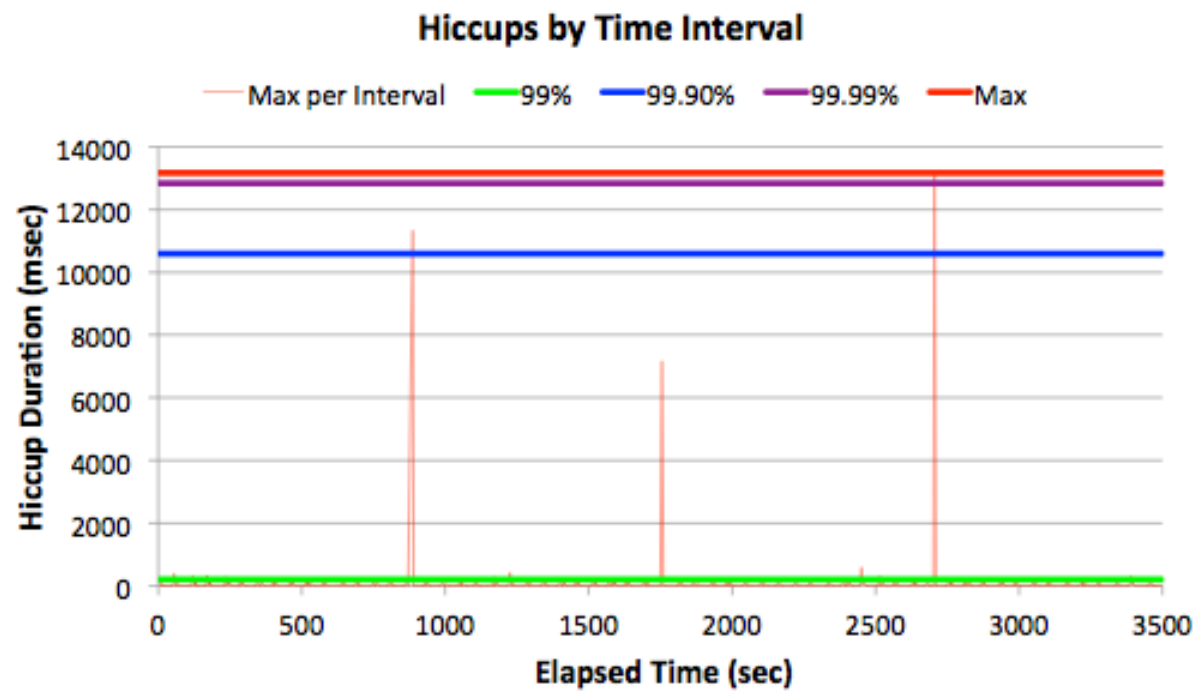
Oracle HotSpot CMS, 1GB in an 8GB heap



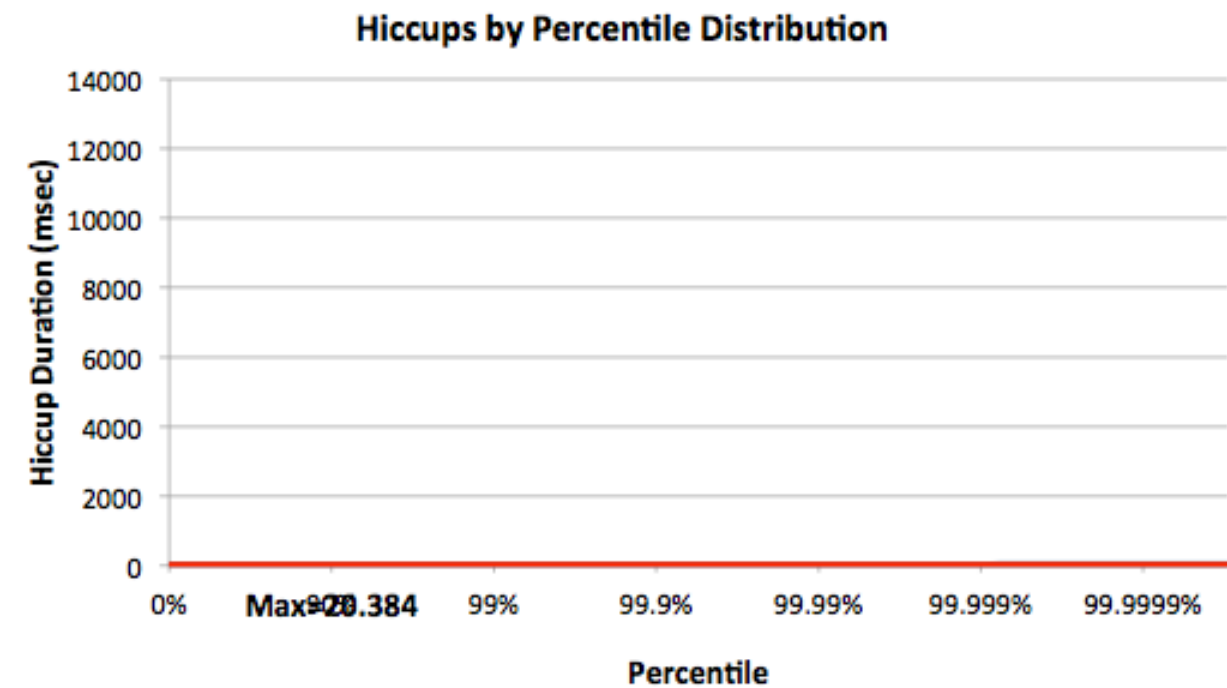
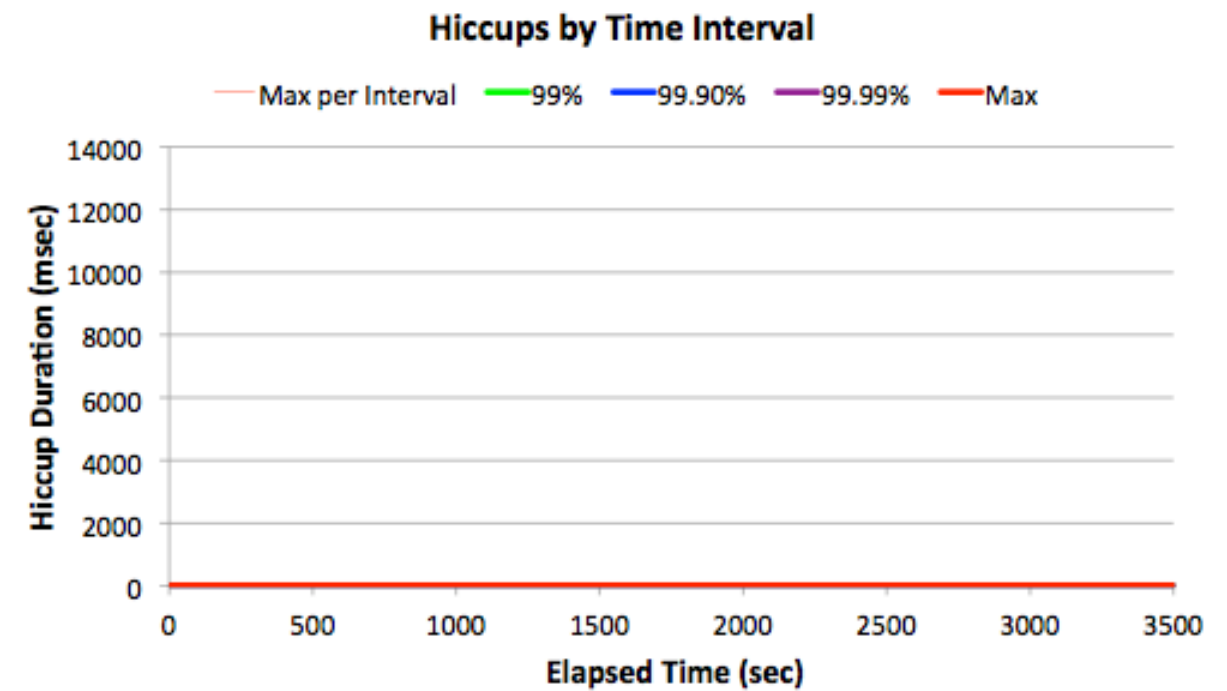
Zing 5, 1GB in an 8GB heap



Oracle HotSpot CMS, 1GB in an 8GB heap



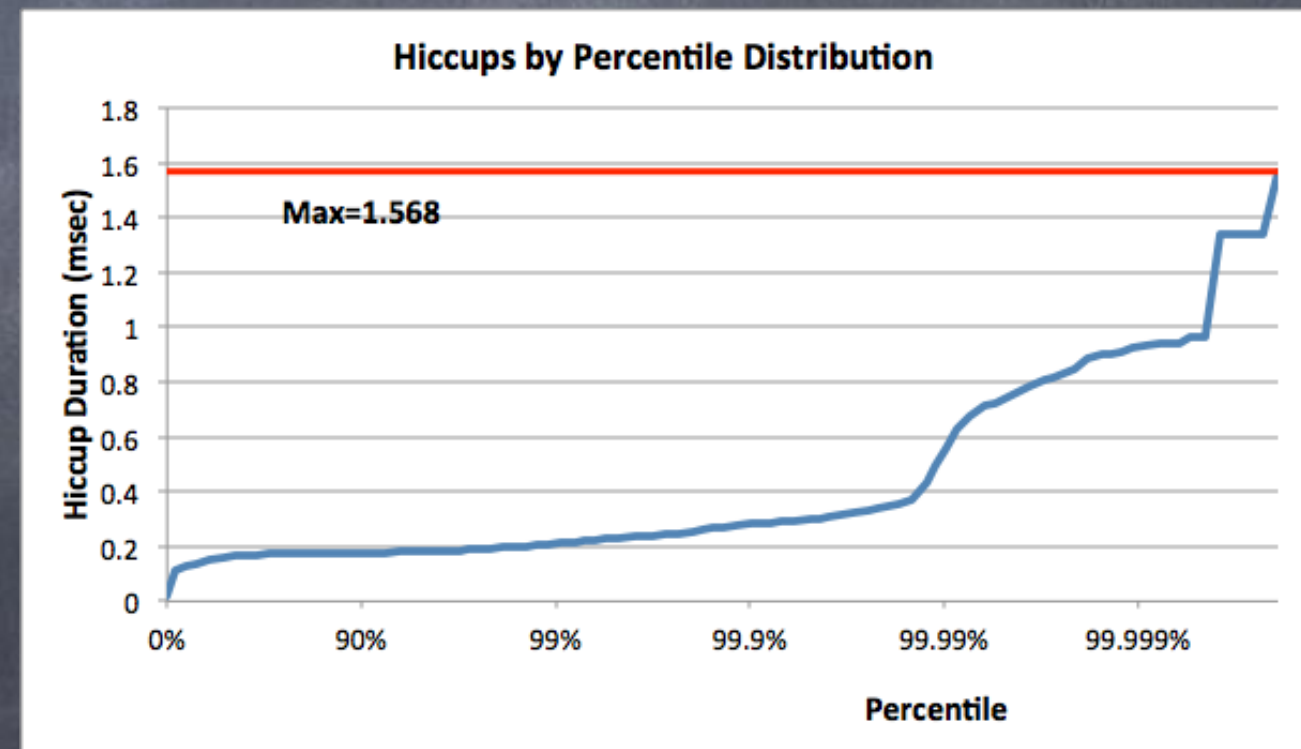
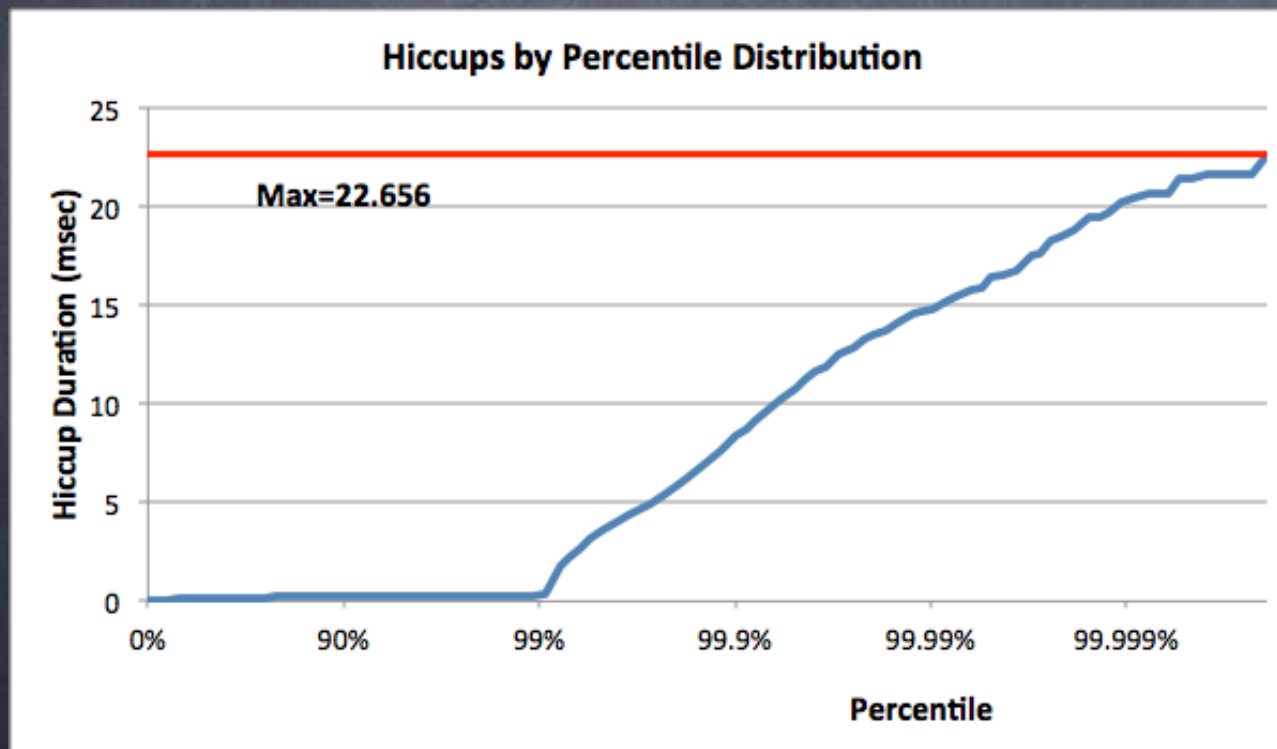
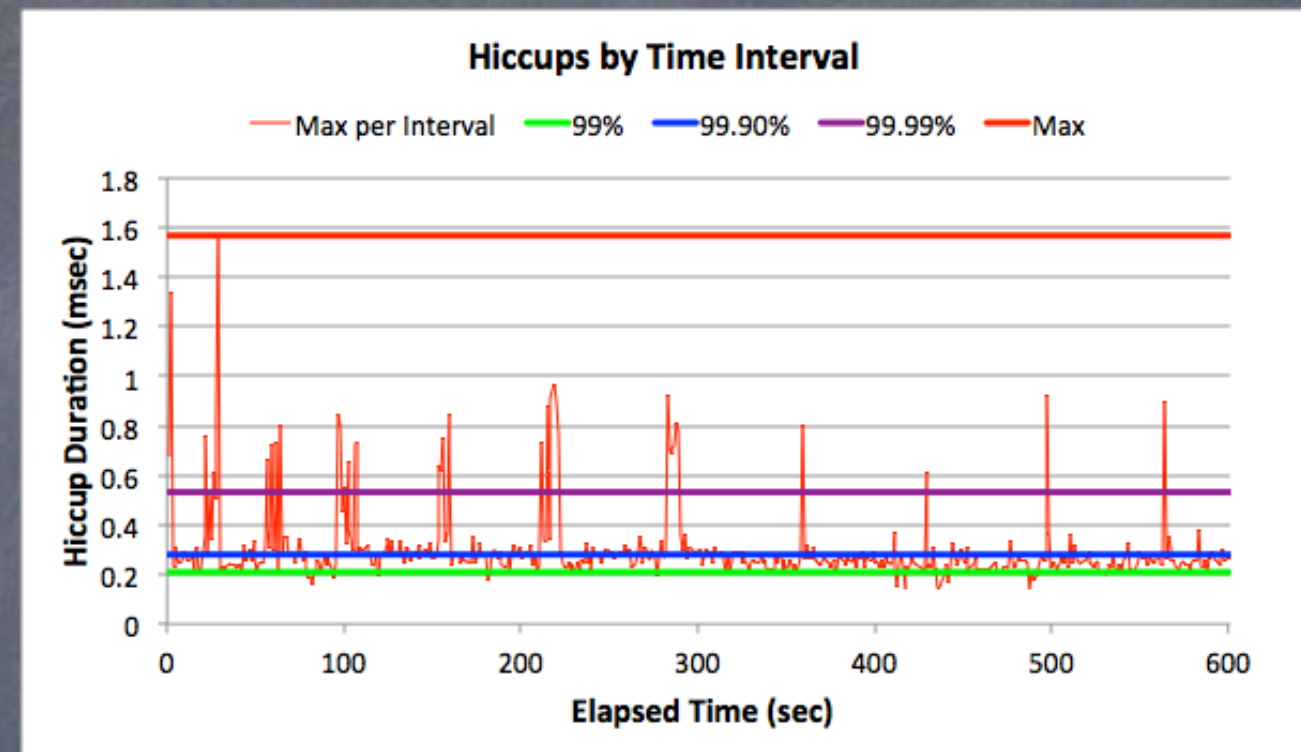
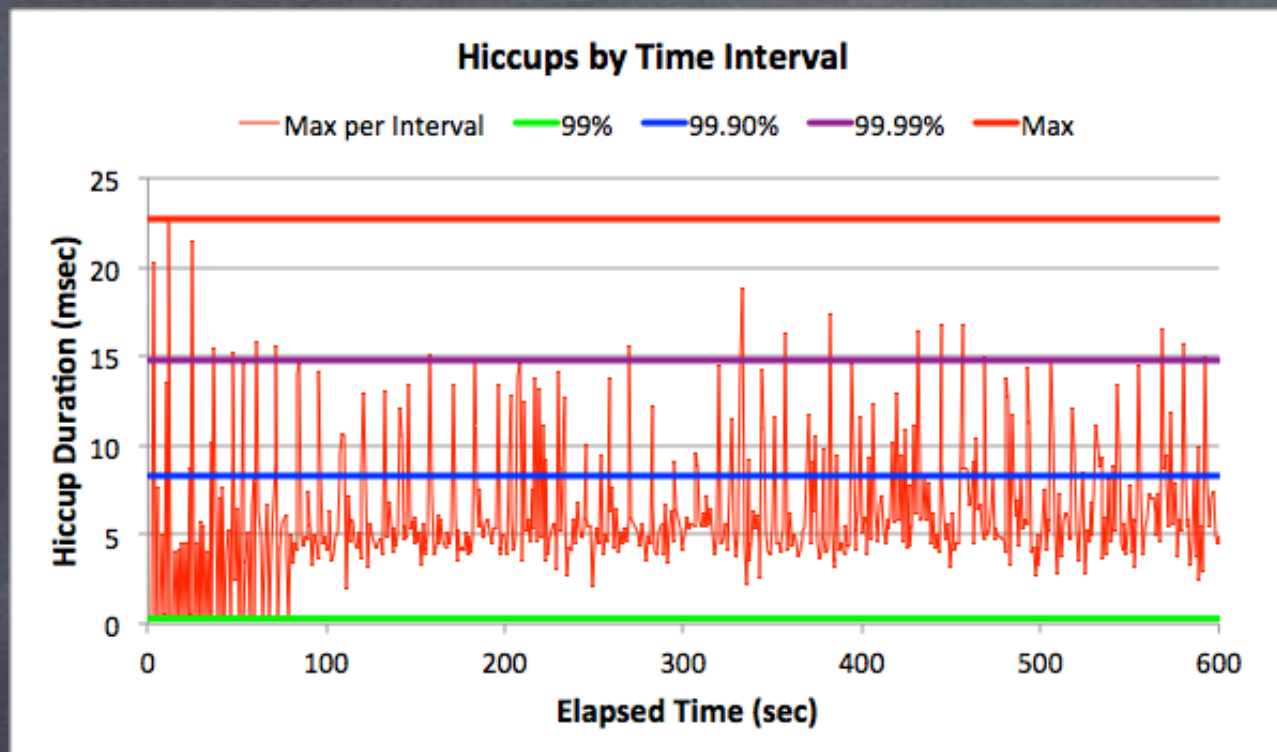
Zing 5, 1GB in an 8GB heap



Drawn to scale

Oracle HotSpot (pure newgen)

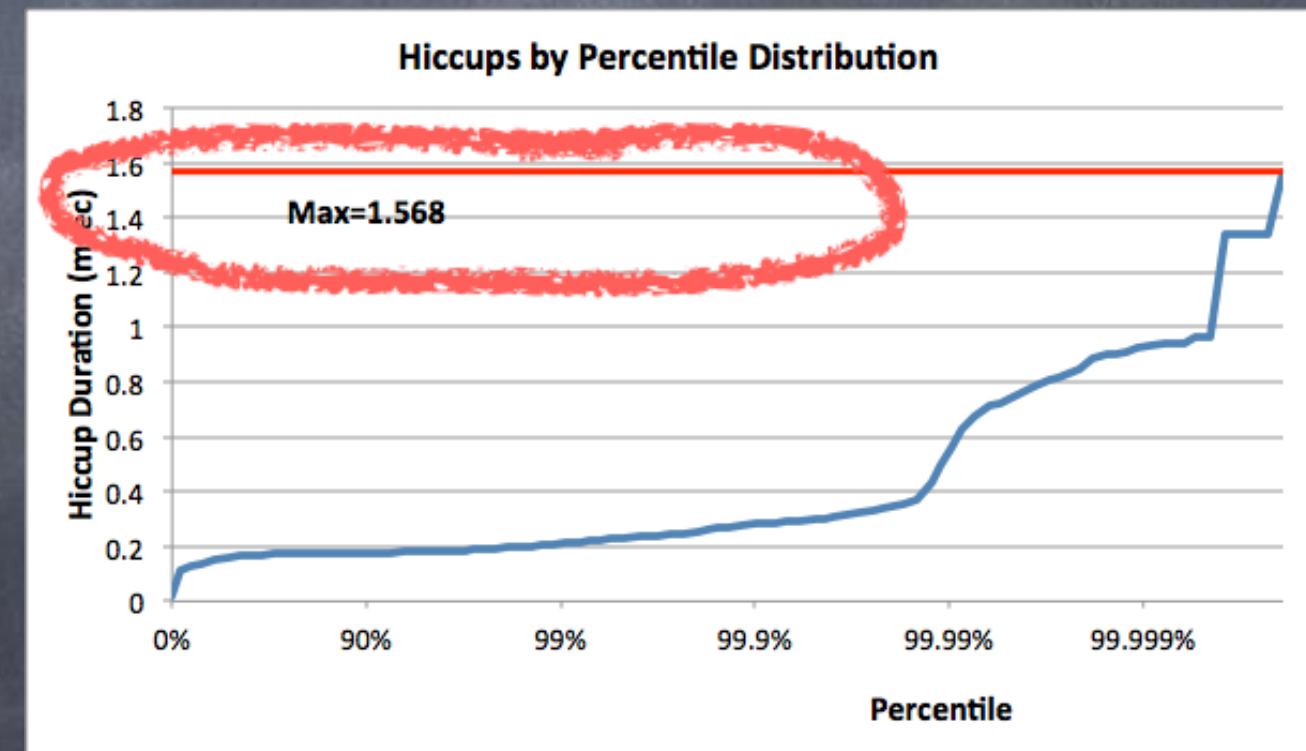
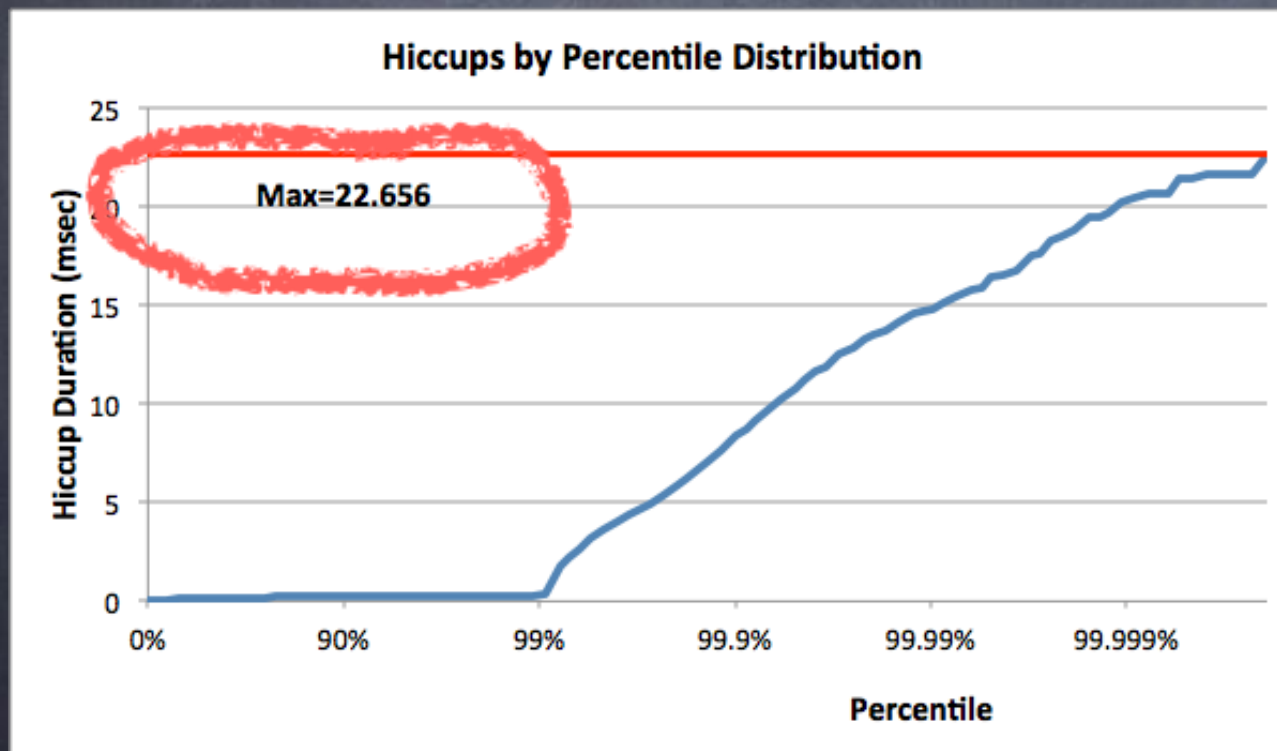
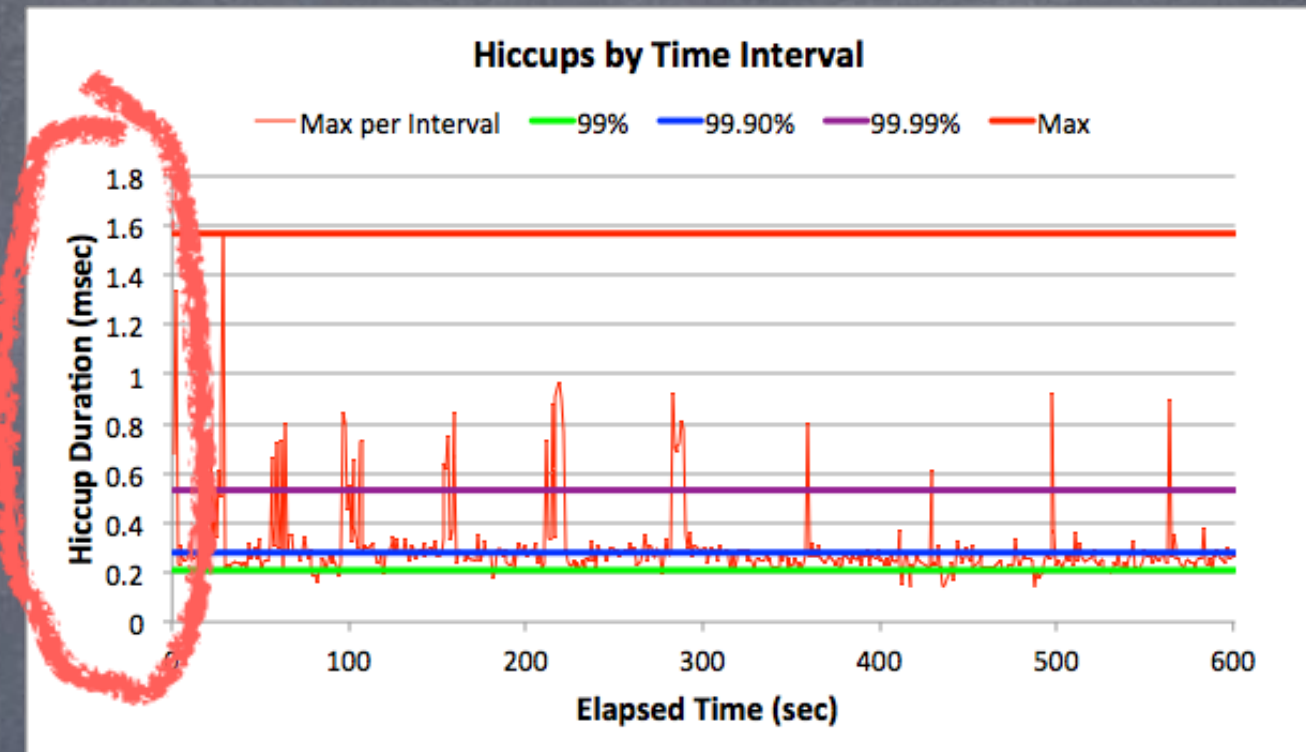
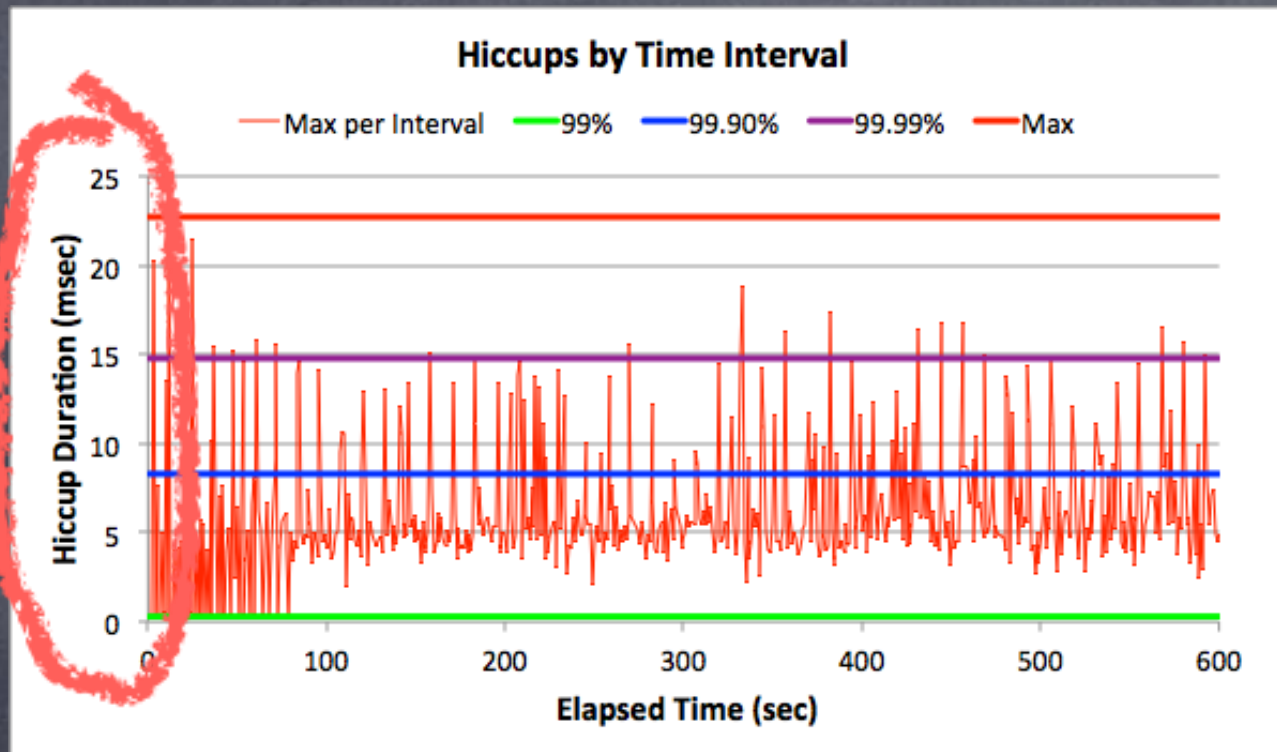
Zing



Low latency trading application

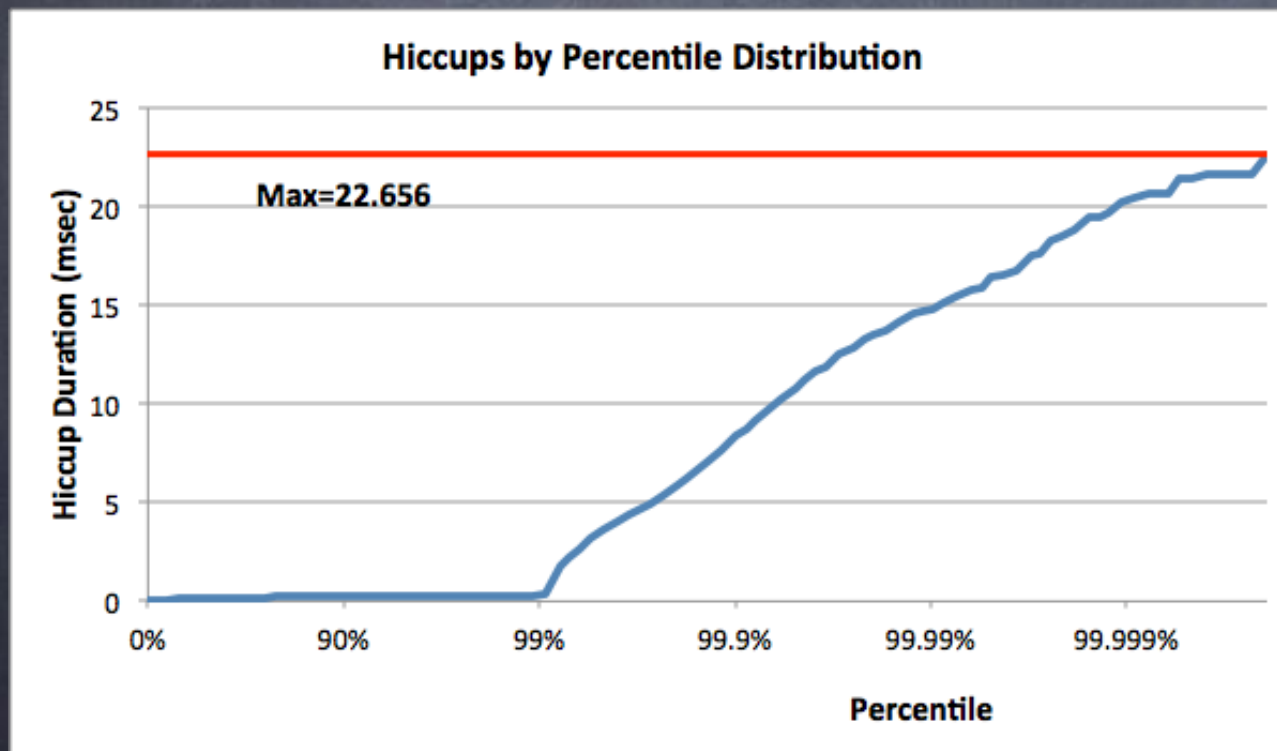
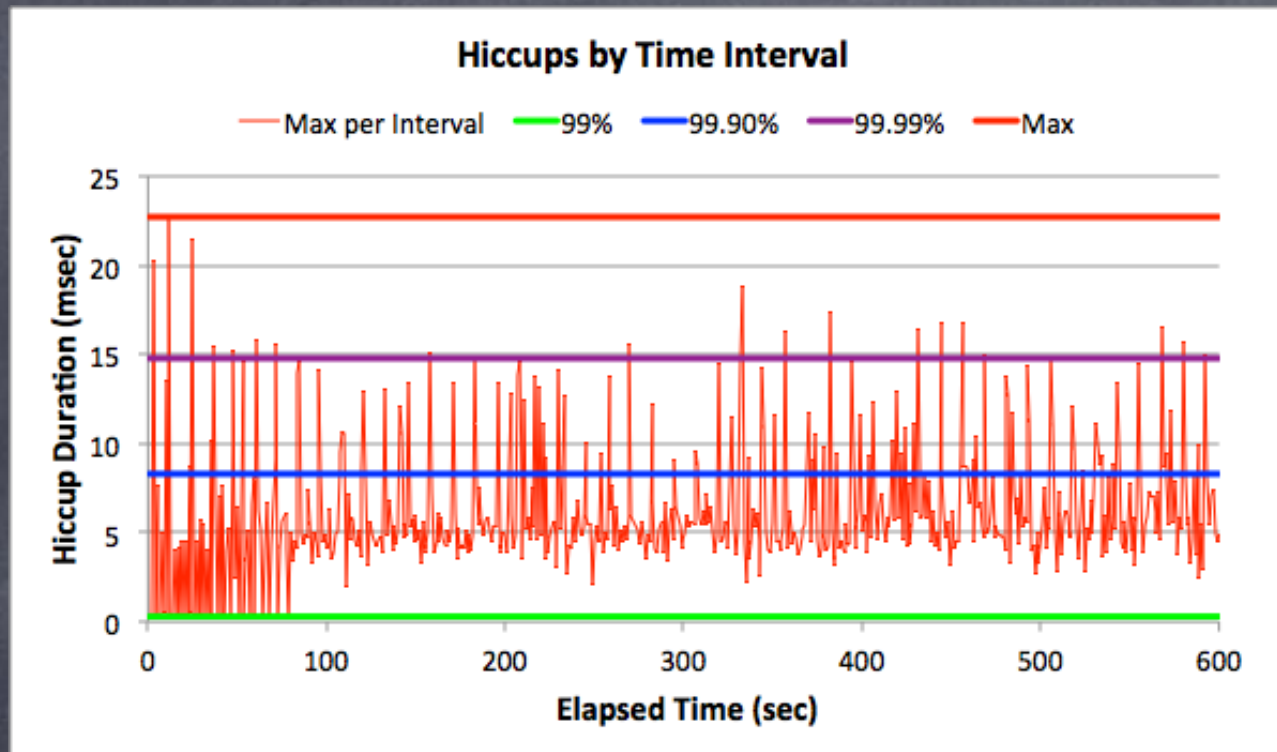
Oracle HotSpot (pure newgen)

Zing

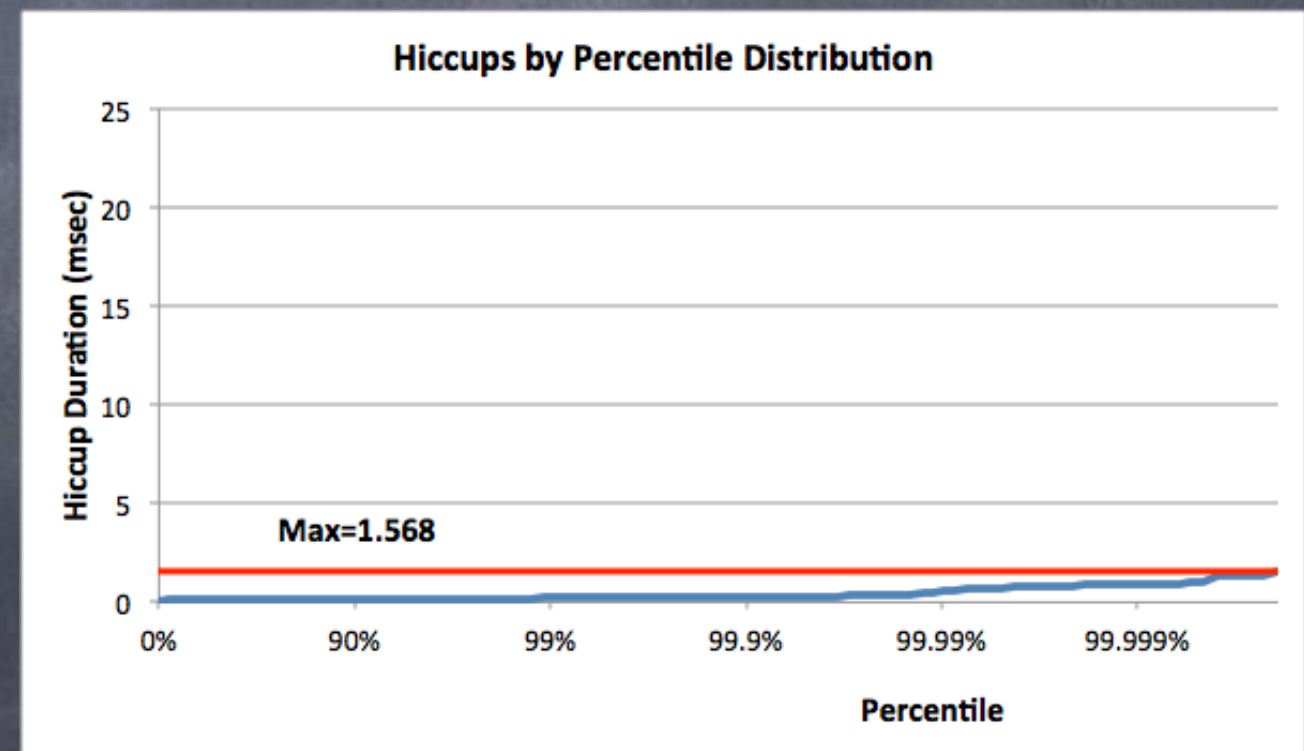
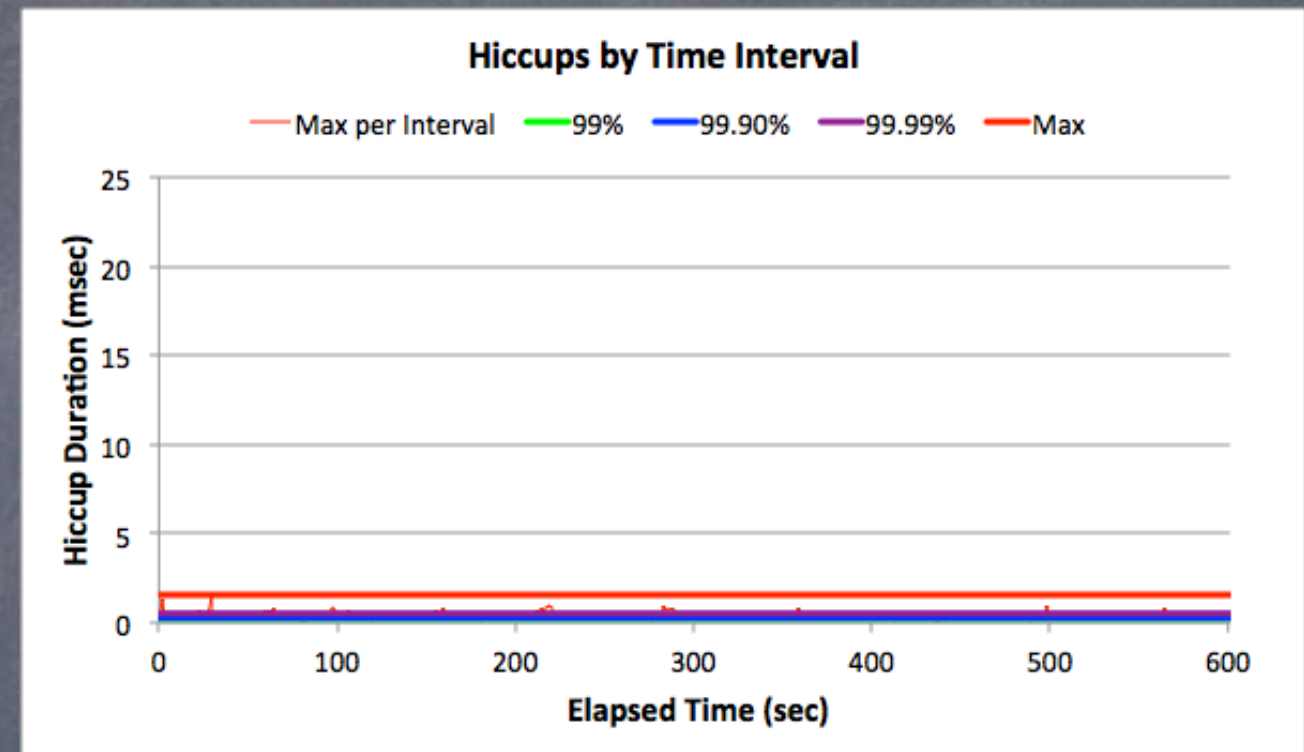


Low latency trading application

Oracle HotSpot (pure newgen)



Zing



Low latency - Drawn to scale

Shameless bragging



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

Zing in Low Latency Java



- A full solution to GC-related issues
- Allows you to code in Java instead of “Java”
 - You can actually use third party code, even if it allocates stuff
 - You can code in idiomatic Java without worries
 - Faster time-to-market
 - less duct-tape engineering
- Not just about GC.
 - We look at anything that makes a JVM “hiccup”.
- “ReadyNow!”: Addresses “Warmup” problems
 - E.g. deoptimization storms at market open

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...



Q & A

<http://www.azul.com>

<http://www.jhiccup.com>

<https://giltene.github.com/HdrHistogram>

<https://github.com/LatencyUtils/LatencyUtils>