How NOT to Measure Latency

Matt Schuetze

Product Management Director, Azul Systems

QCon NY

Brooklyn, New York

@azulsystems | azulsystems.com

Understanding Latency and Application Responsiveness

Matt Schuetze

Product Management Director, Azul Systems

QCon NY

Brooklyn, New York

@azulsystems | azulsystems.com

The Oh $@%T! talk.

Matt Schuetze

Product Management Director, Azul Systems

QCon NY

Brooklyn, New York

@azulsystems | azulsystems.com

About me: Matt Schuetze

- Product Management Director at Azul Systems
- Translate Voice of Customer into Zing and Zulu requirements and work items
- Sing the praises of Azul efforts through product launches
- Azul alternate on JCP exec committee, co-lead Detroit Java User Group
- Stand on the shoulders of giants and admit it

Philosophy and motivation

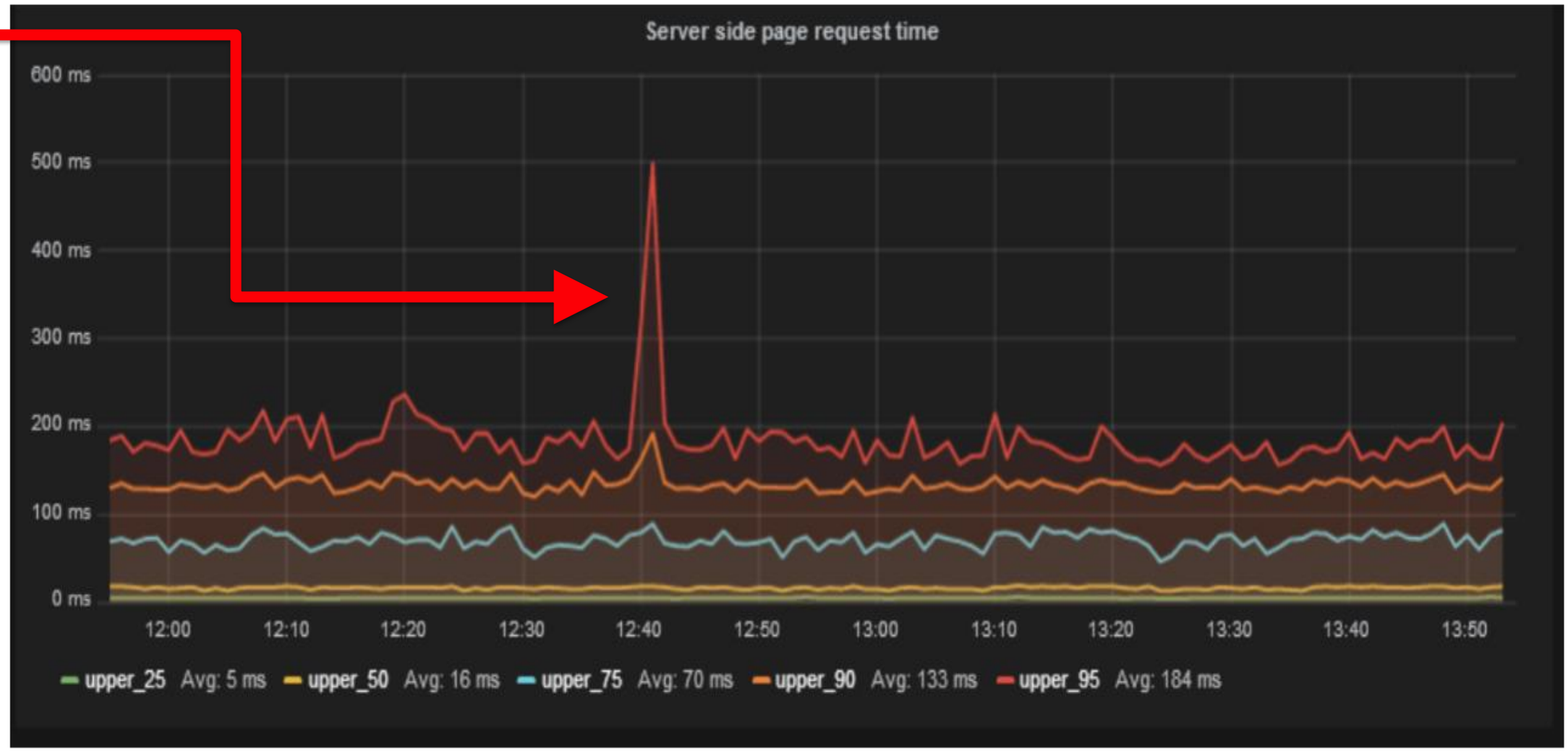
What do we actually care about. And why?

Latency Behavior

- Latency: The time it took one operation to happen
- Each operation occurrence has its own latency
- What we care about is how latency behaves
- Behavior is a lot more than “the common case was X”

We like to look at charts

95%’ile



The “We only want to show good things” chart

What do you care about?

Do you :

- ☞ **Care about latency in your system?**
- ☞ **Care about the worst case?**
- ☞ **Care about the 99.99%’ile?**
- ☞ **Only care about the fastest thing in the day?**
- ☞ **Only care about the best 50%**
- ☞ **Only need 90% of operations to meet requirements?**

We like to rant about latency

About Me

Gil Tene

CTO and co-founder of Azul Systems.

View my complete profile

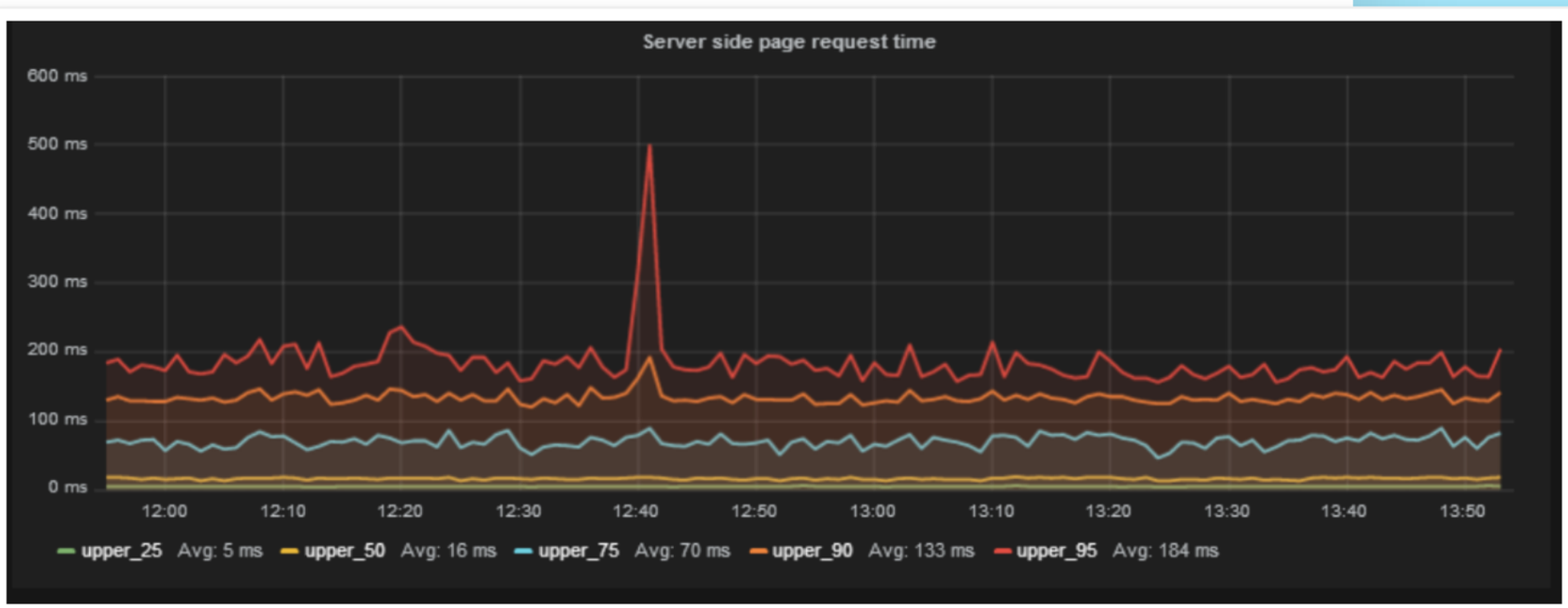
Blog Archive

- 2014 (8)
  - June (8)
    - #LatencyTipOfTheDay: Median Server Response Time: ...
    - #LatencyTipOfTheDay: MOST page loads will experien...
    - #LatencyTipOfTheDay: Q: What's wrong with this pic...
    - #LatencyTipOfTheDay: If you are not measuring and/...
    - #LatencyTipOfTheDay : Measure what you need to mon...
    - #LatencyTipOfTheDay: Average (def): a random numbe...

Saturday, June 21, 2014

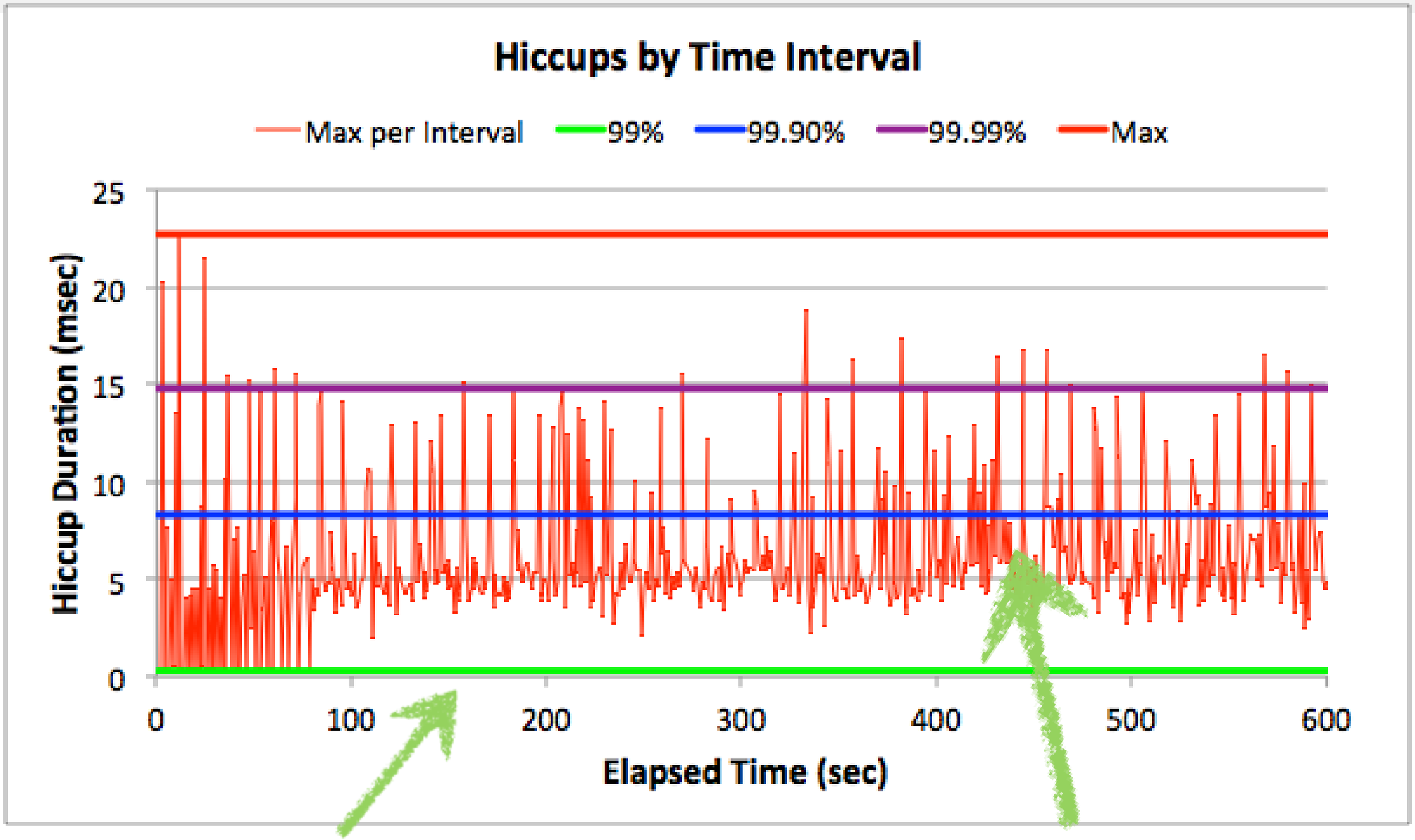
#LatencyTipOfTheDay: Q: What's wrong with this picture? A: Everything!

Question: What's wrong with this picture:



Answer: Everything!

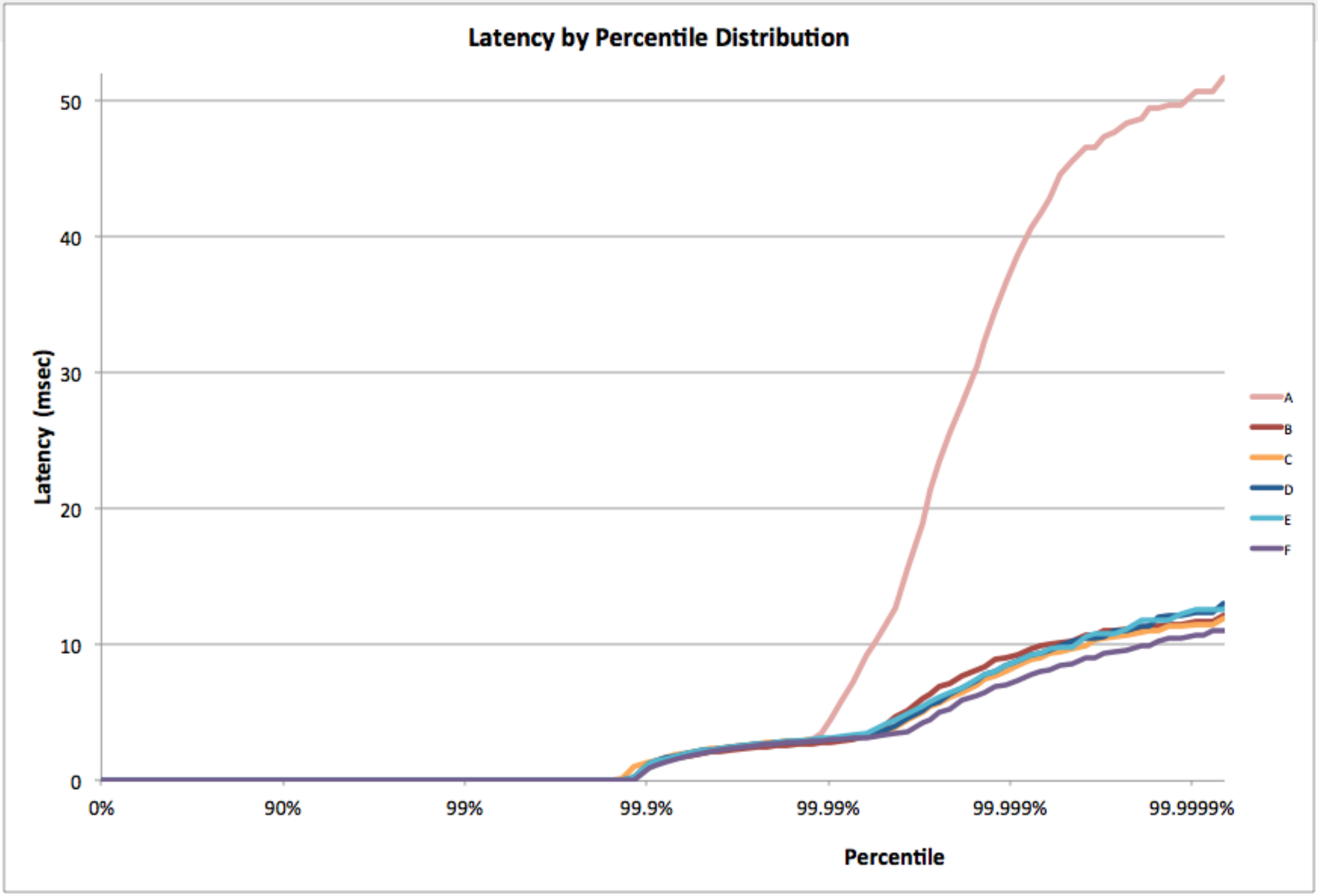
“outliers”, “averages” and other nonsense



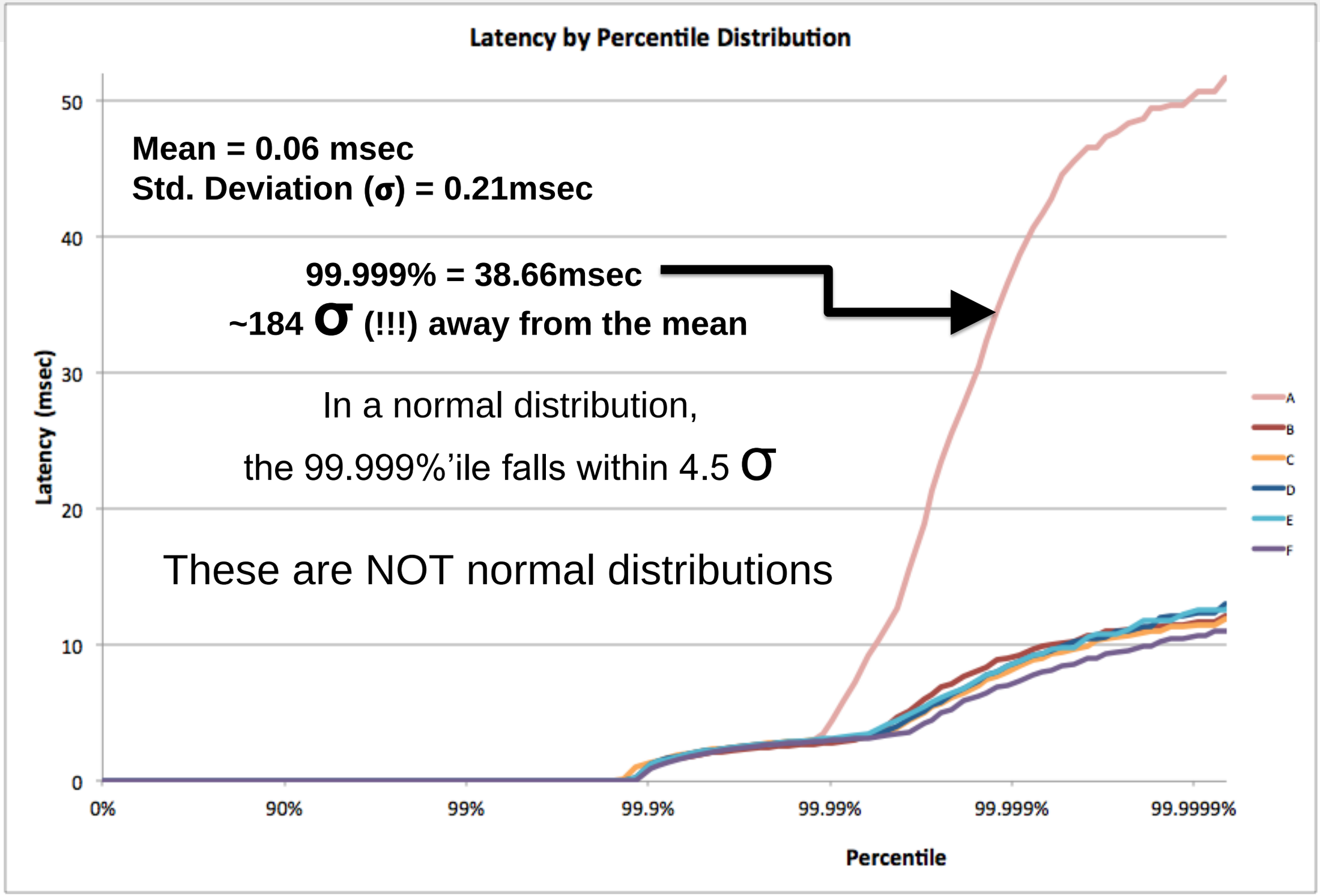
99%‘ile is ~60 usec.
(but mean is ~210usec)

We nicknamed these spikes “hiccups”

Dispelling standard deviation



Dispelling standard deviation



Is the 99%’ile “rare”?

Cumulative probability…

What are the chances of a single web page view experiencing the 99%'ile latency of:

- A single search engine node?

- A single Key/Value store node?

- A single Database node?

- A single CDN request?

Site	# of requests	page loads that would experience the 99%'lie [(1 - (.99 ^ N)) * 100%]
amazon.com	190	85.2%
kohls.com	204	87.1%
jcrew.com	112	67.6%
saksfifthavenue.com	109	66.5%
--	--	--
nytimes.com	173	82.4%
cnn.com	279	93.9%
--	--	--
twitter.com	87	58.3%
pinterest.com	84	57.0%
facebook.com	178	83.3%
--	--	--
google.com (yes, that simple noise-free page)	31	26.7%
google.com search for "http requests per page"	76	53.4%

Which HTTP response time metric is more “representative” of user experience?

The 95%’ile or the 99.9%’ile

Gauging user experience

Example: A typical user session involves 5 page loads, averaging 40 resources per page.

- How many of our users will NOT experience something worse than the 95%’ile?

Answer: **~0.003%**

- How many of our users will experience at least one response that is longer than the 99.9%’ile?

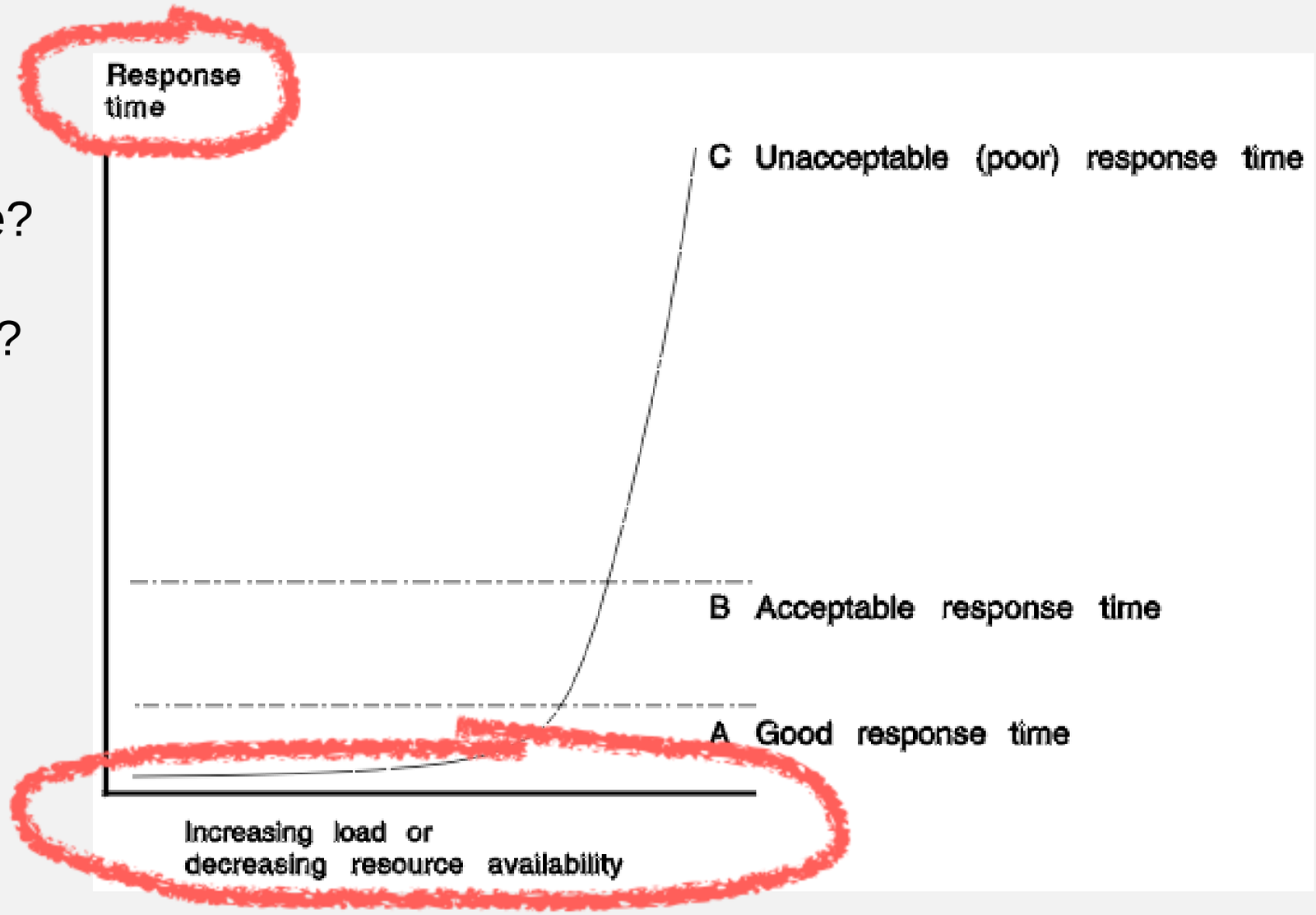
Answer: **~18%**

Classic look at response time behavior

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

Response time as a function of load

Average?
Max?
Median?
90%?
99.9%



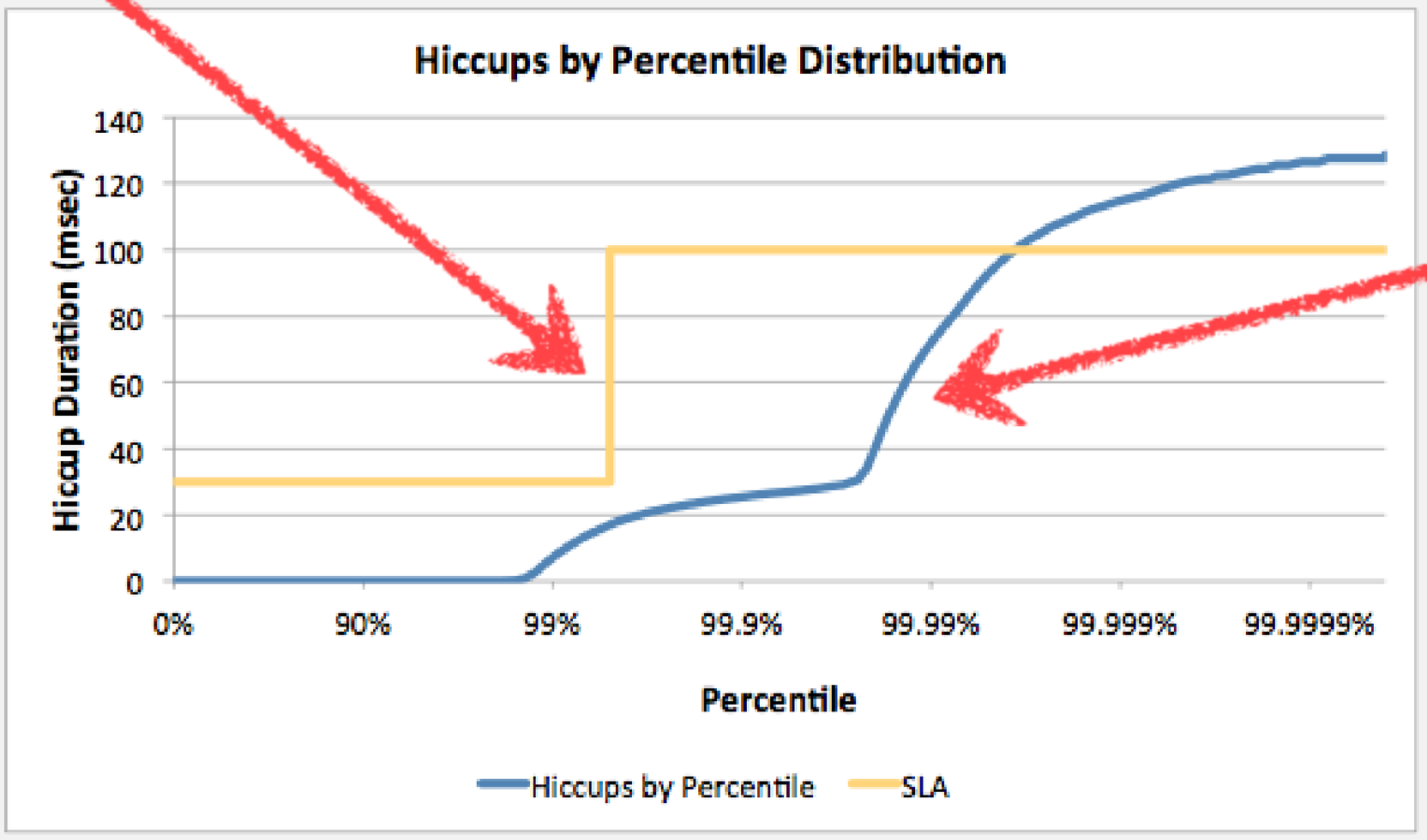
Hiccups are strongly multi-modal

- They don’t look anything like a normal distribution
- A complete shift from one mode/behavior to another
- Mode A: “good”.
- Mode B: “Somewhat bad”
- Mode C: “terrible”, ...
- The real world is not a gentle, smooth curve
- Mode transitions are “phase changes”

Proven ways to deal with hiccups

Actually characterizing latency

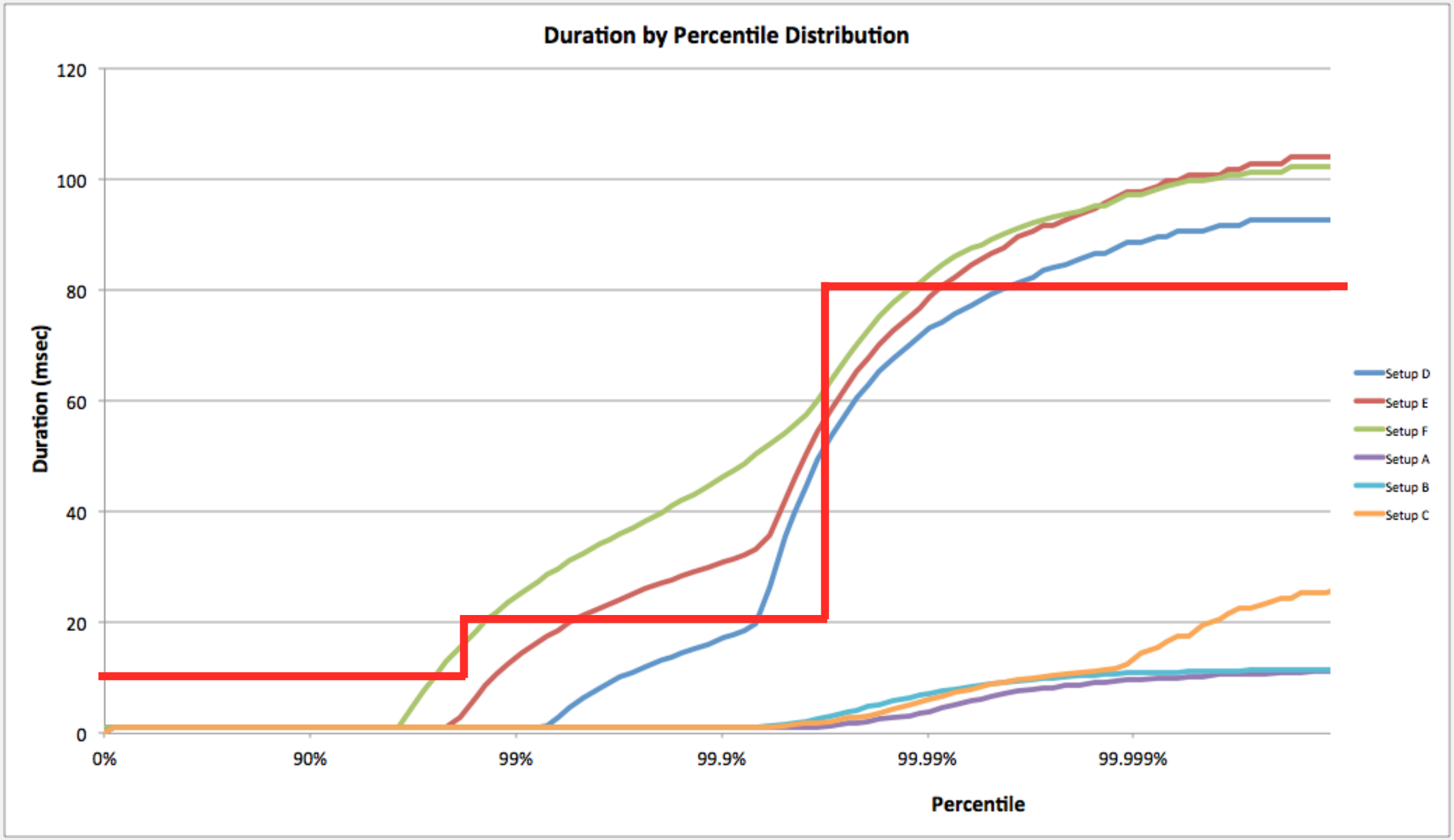
Requirements



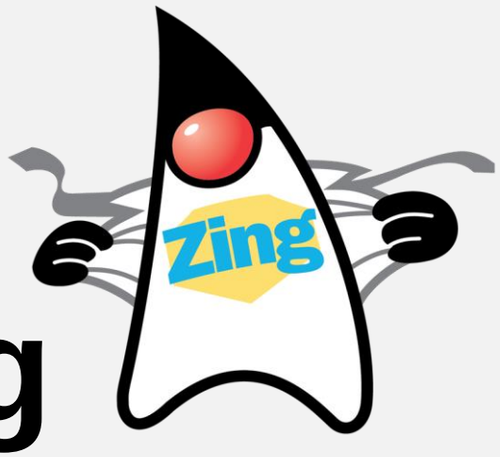
Response Time Percentile plot line

Comparing Behavior

Different throughputs, configurations, or other parameters on one graph

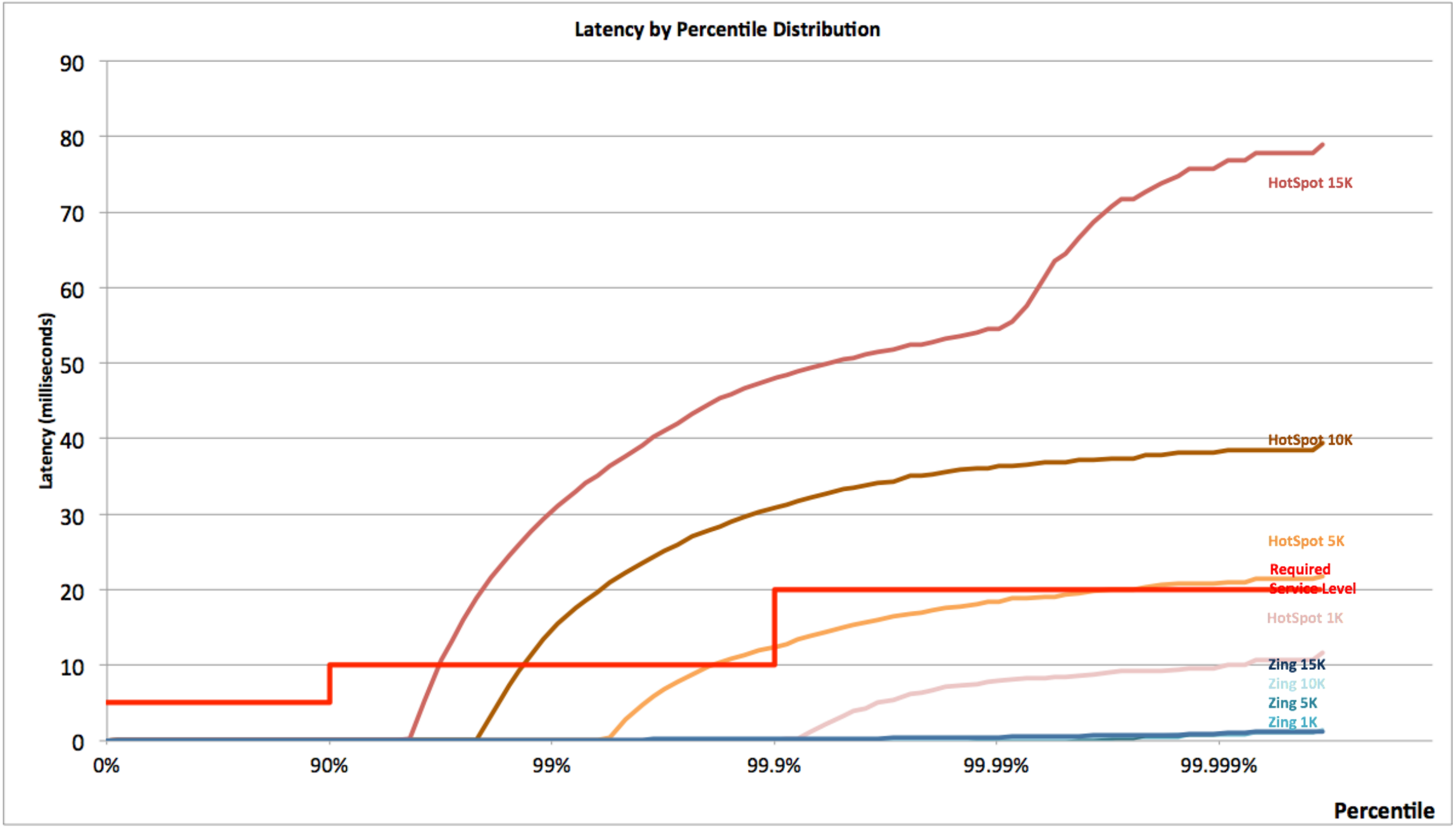


Shameless Bragging

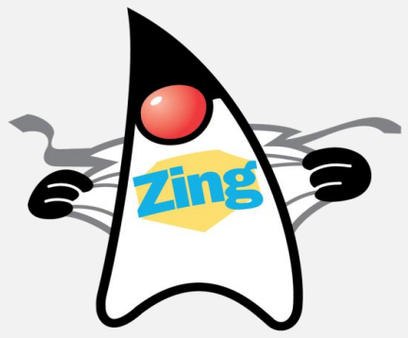


Comparing Behaviors - Actual

Latency sensitive messaging distribution application: HotSpot vs. Zing



Zing



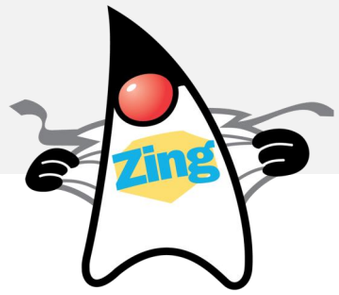
- A standards-compliant **JVM** for Linux/x86 servers
- **Eliminates** Garbage Collection as a concern for enterprise applications in Java, Scala, or any JVM language
- Very wide operating range: Used in both low latency and large scale enterprise application spaces
- Decouples scale metrics from response time concerns

  Transaction rate, data set size, concurrent users, heap size, allocation rate, mutation rate, etc.
- Leverages elastic memory for resilient operation

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, up to as high as 1TB memory,
- Then Zing will likely deliver superior behavior metrics

Where Zing shines



- Low latency
  - Eliminate behavior blips down to the sub-millisecond-units level
- Machine-to-machine “stuff”
  - Support higher *sustainable* throughput (one that meets SLAs)
  - Messaging, queues, market data feeds, fraud detection, analytics
- 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.
  - Cassandra, Spark, Solr, DataGrid, any large dataset in fast motion

The coordinated omission problem

An accidental conspiracy...

The coordinated omission problem

- Common load testing example:
  - each “client” issues requests at a certain rate
  - measure/log response time for each request
- So what’s wrong with that?
  - works only if ALL responses fit within interval
  - implicit “automatic back off” coordination
- Begin audience participation exercise now…

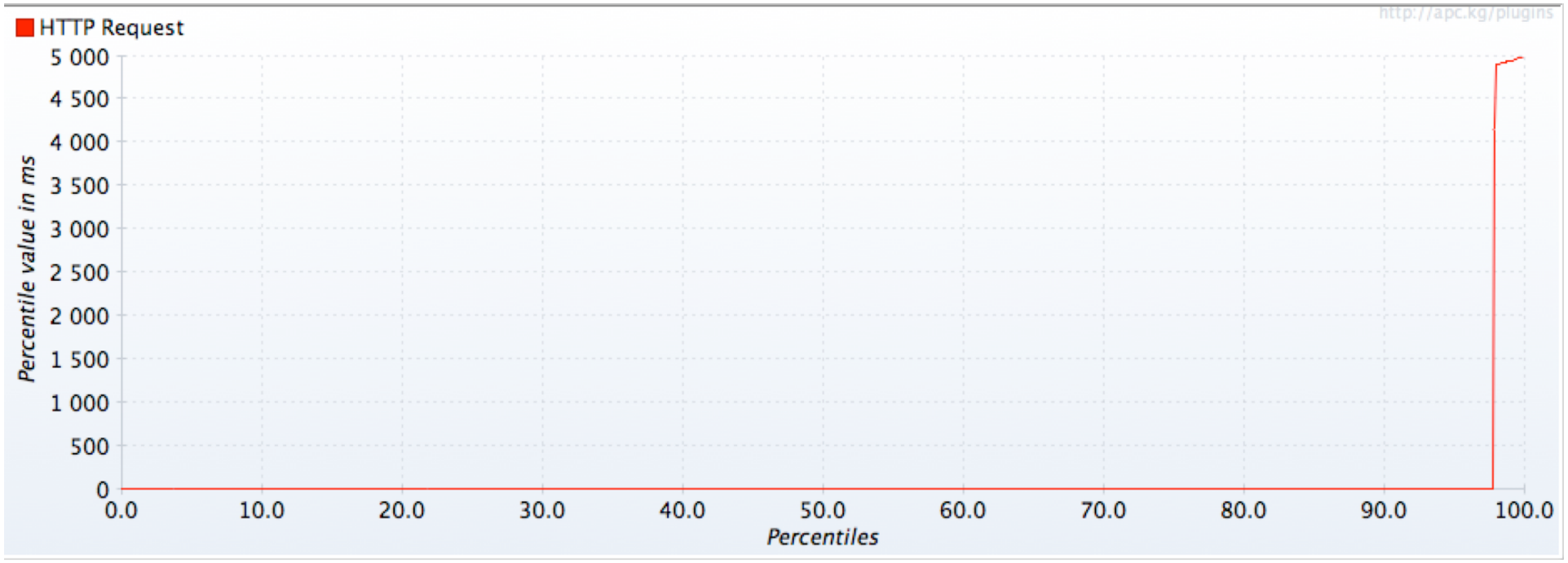
Is coordinated omission rare?

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

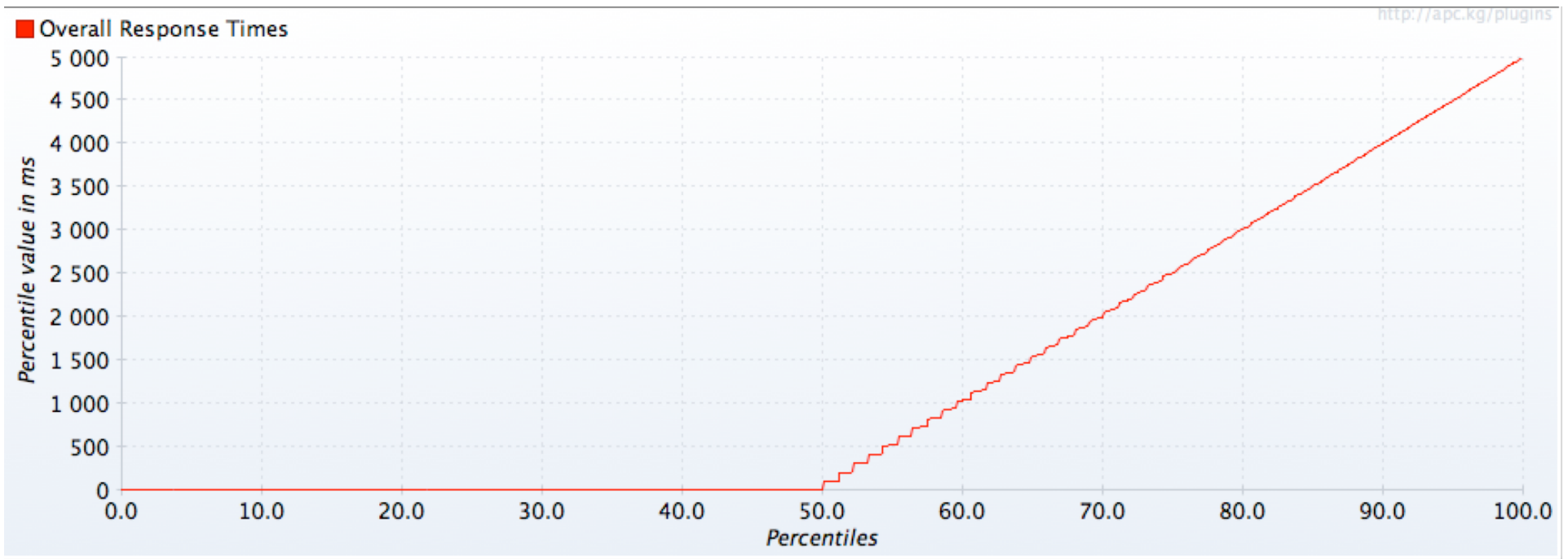
JMeter makes this mistake...

And so do other tools!

Before Correction

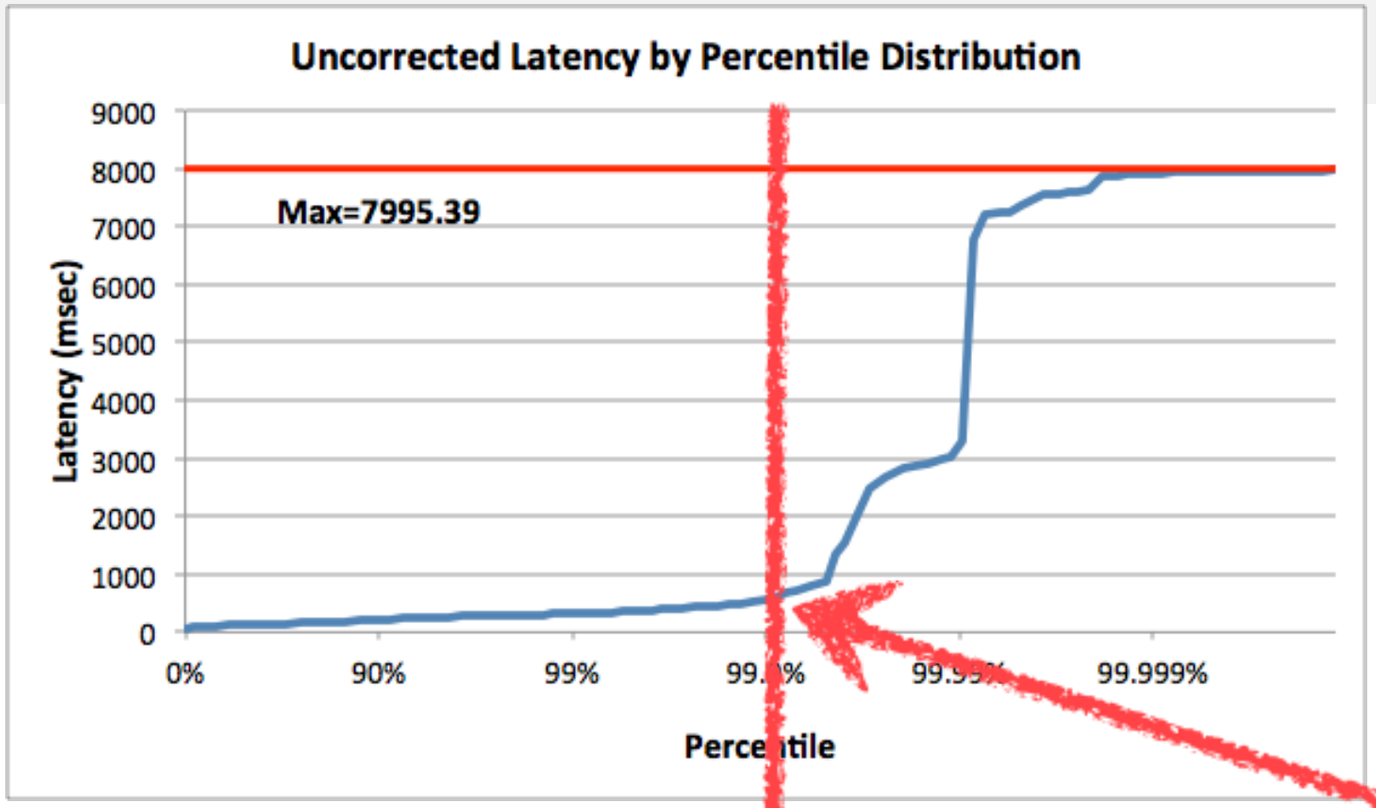


After Correcting for Omission



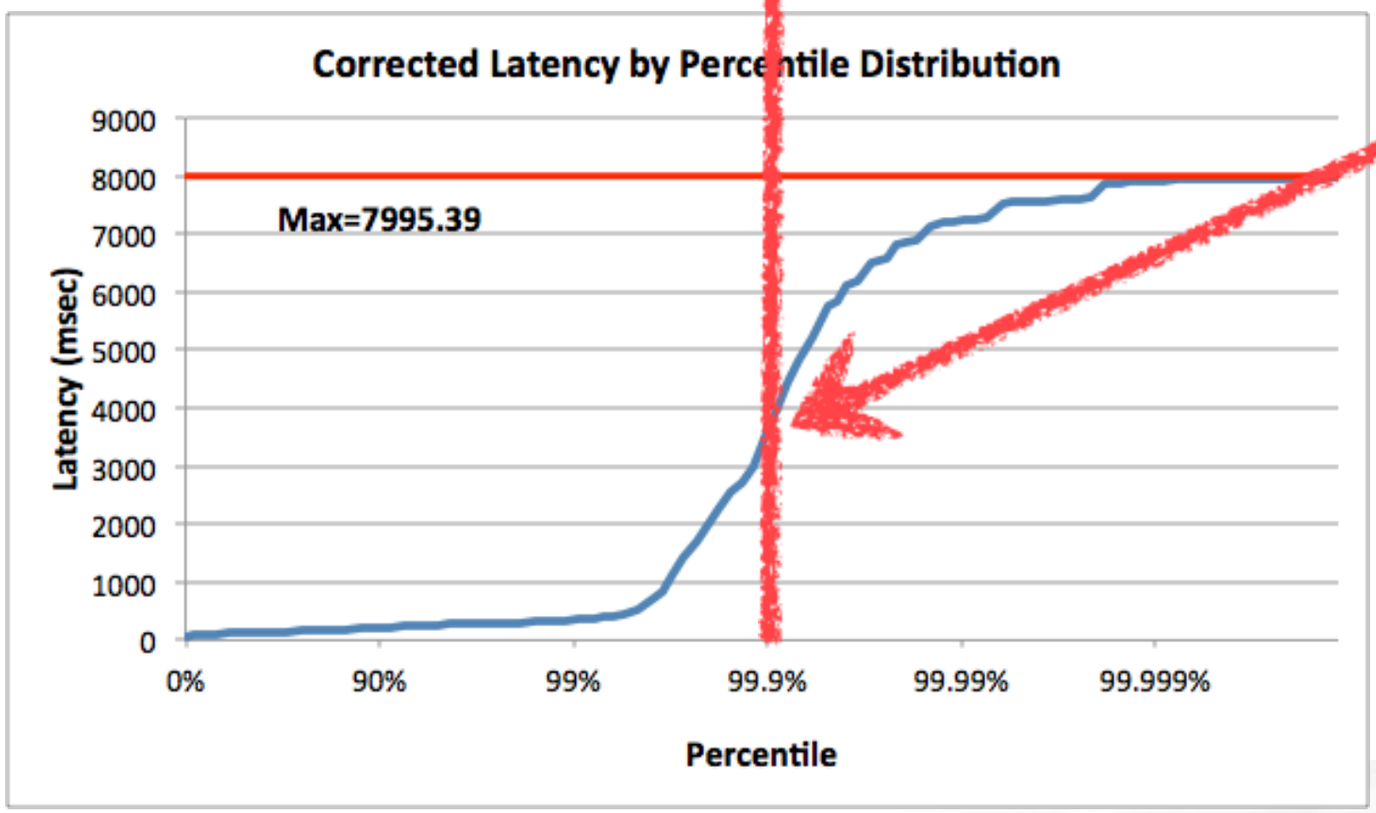
Real World Coordinated Omission effects

Before Correction

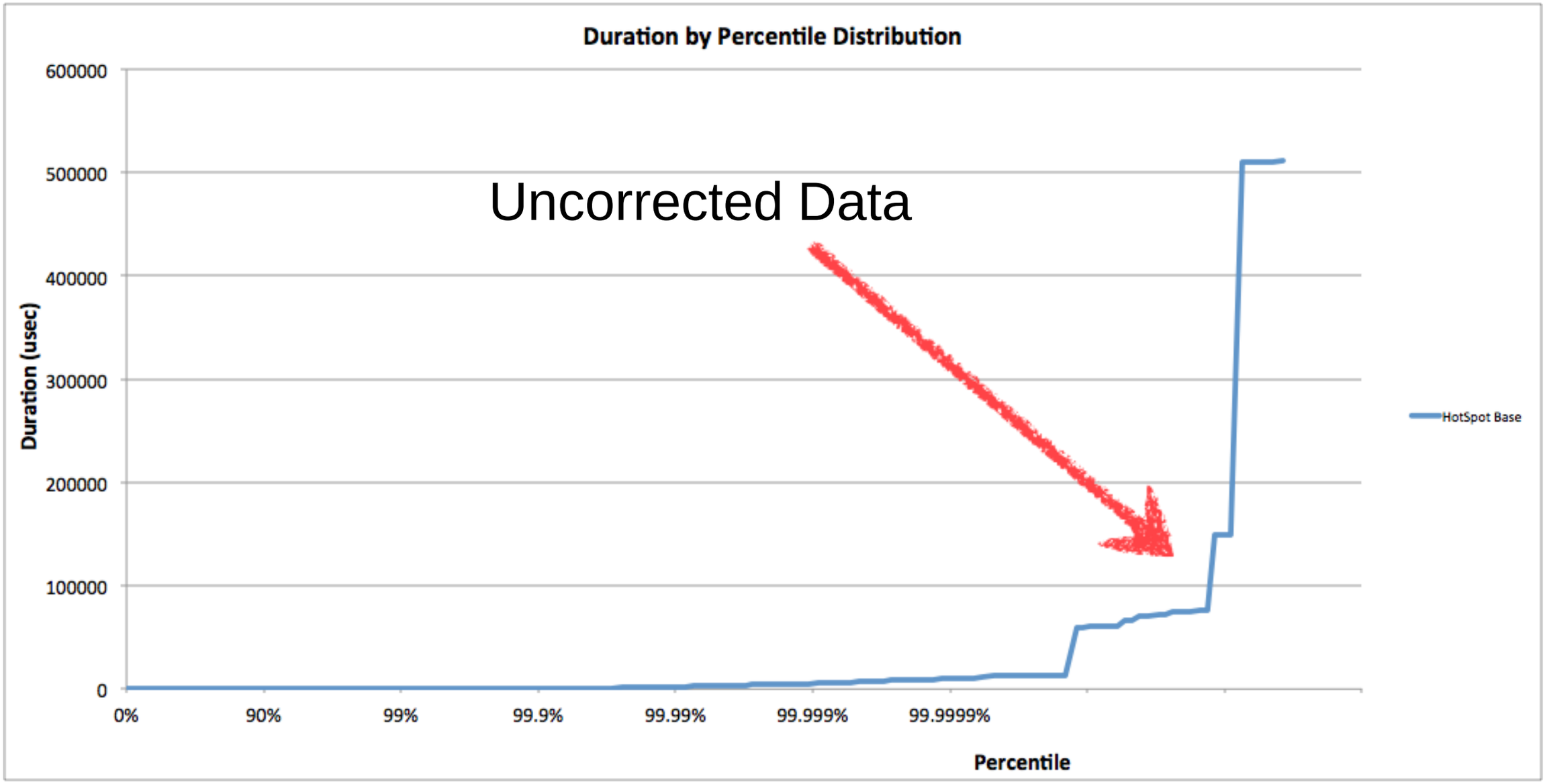


Wrong by 7x

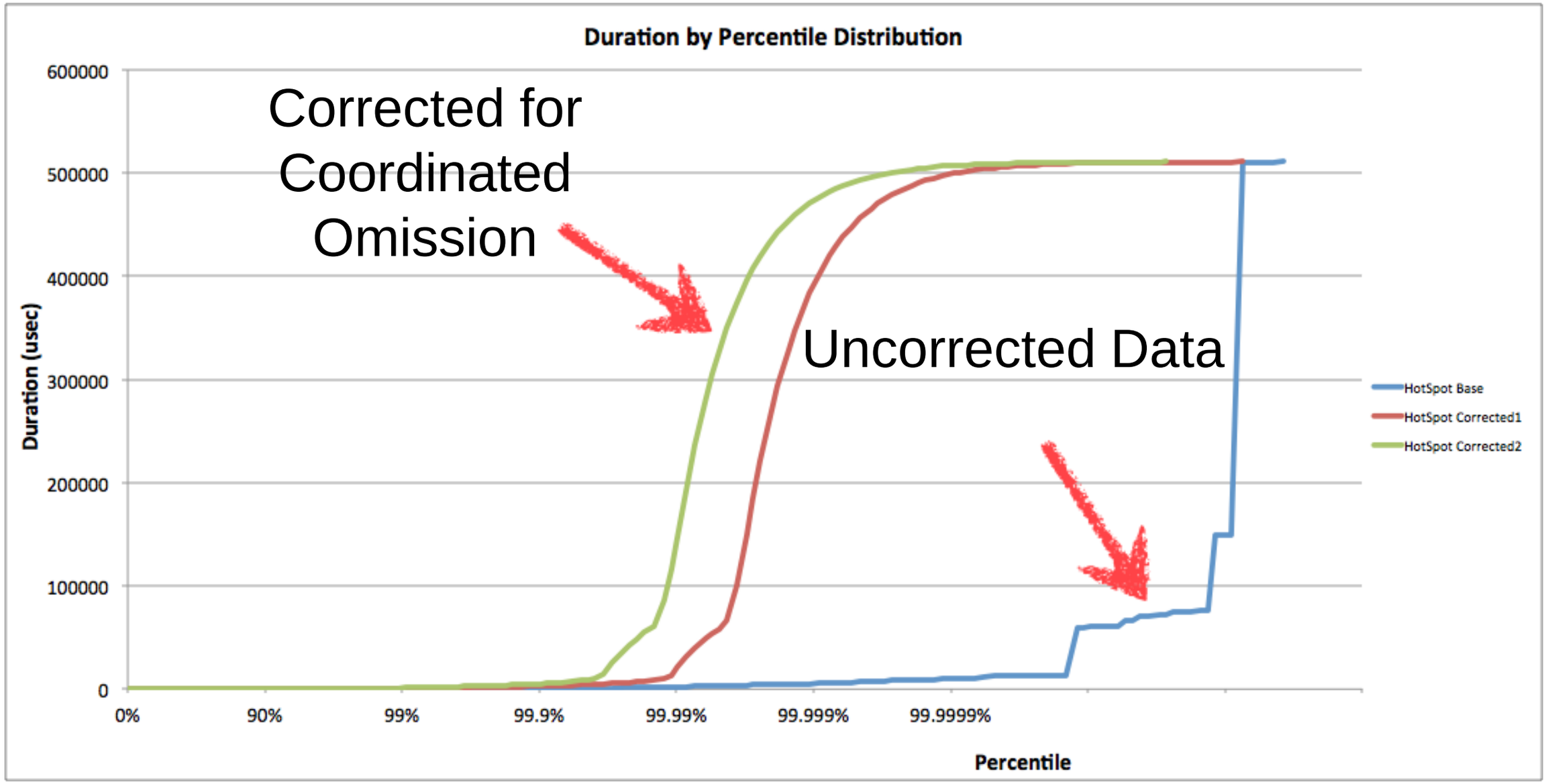
After Correction



Real World Coordinated Omission effects

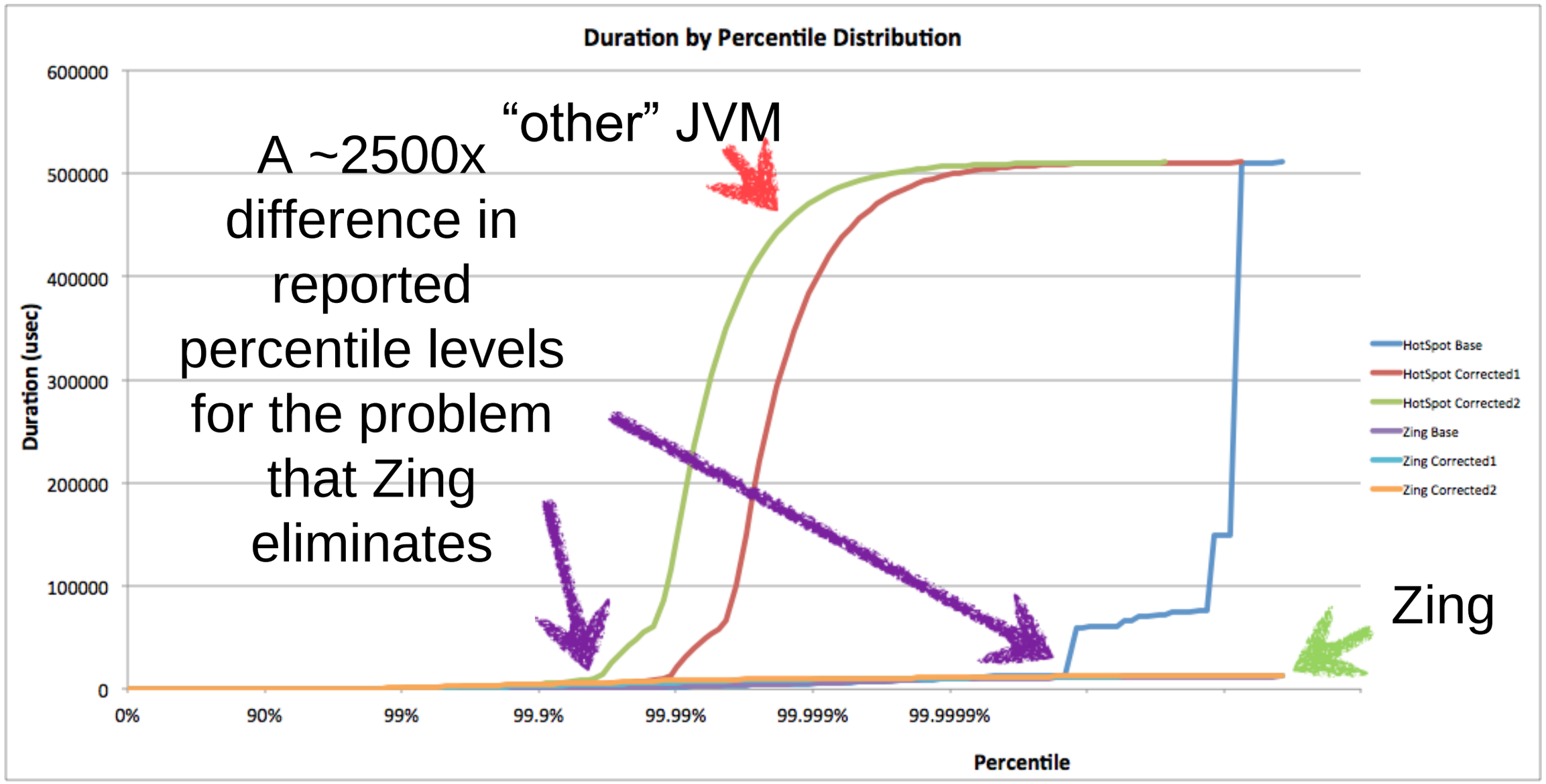


Real World Coordinated Omission effects



Real World Coordinated Omission effects

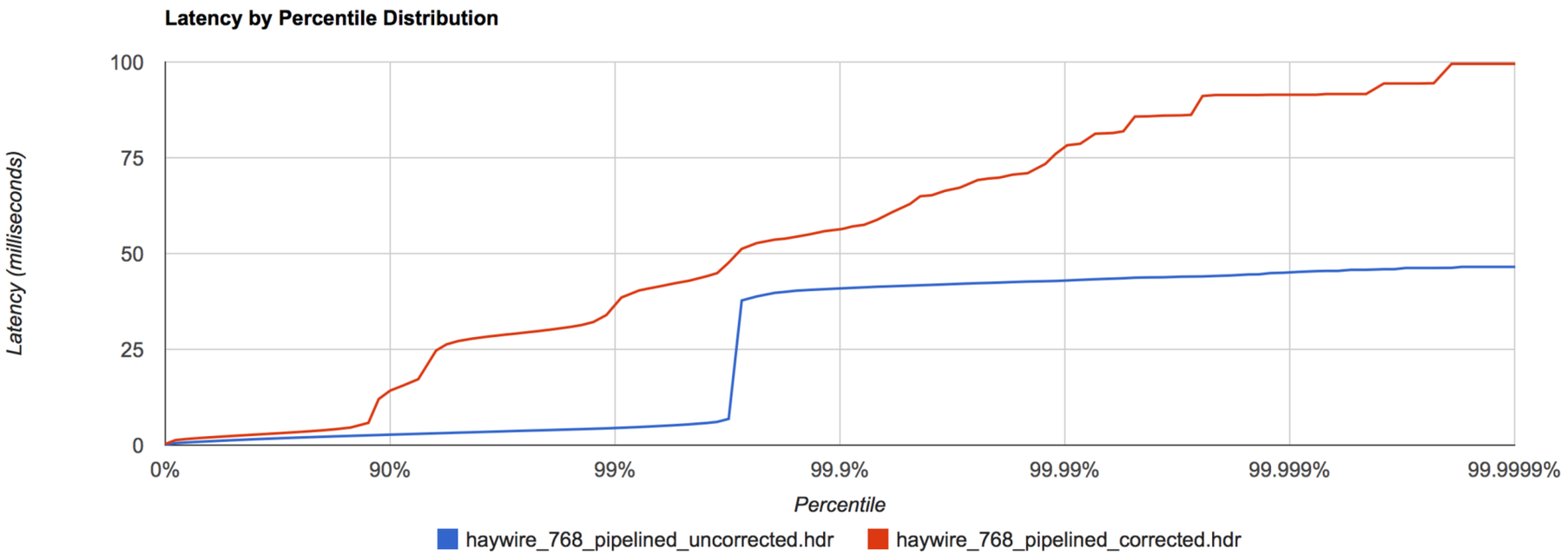
Why I care



How “real” people react

Kelly Sommers @kellabyte 2d

LOL at how badly we all benchmark. Blue is how most of us are benchmarking, Red is the actual truth i.imgur.com/HYoWEu6.png



Leandro Pereira @lafp

@kellabyte Blue, you believe in whatever you want to believe. Red, you wake up in Wonderland and see how deep the rabbit hole goes.

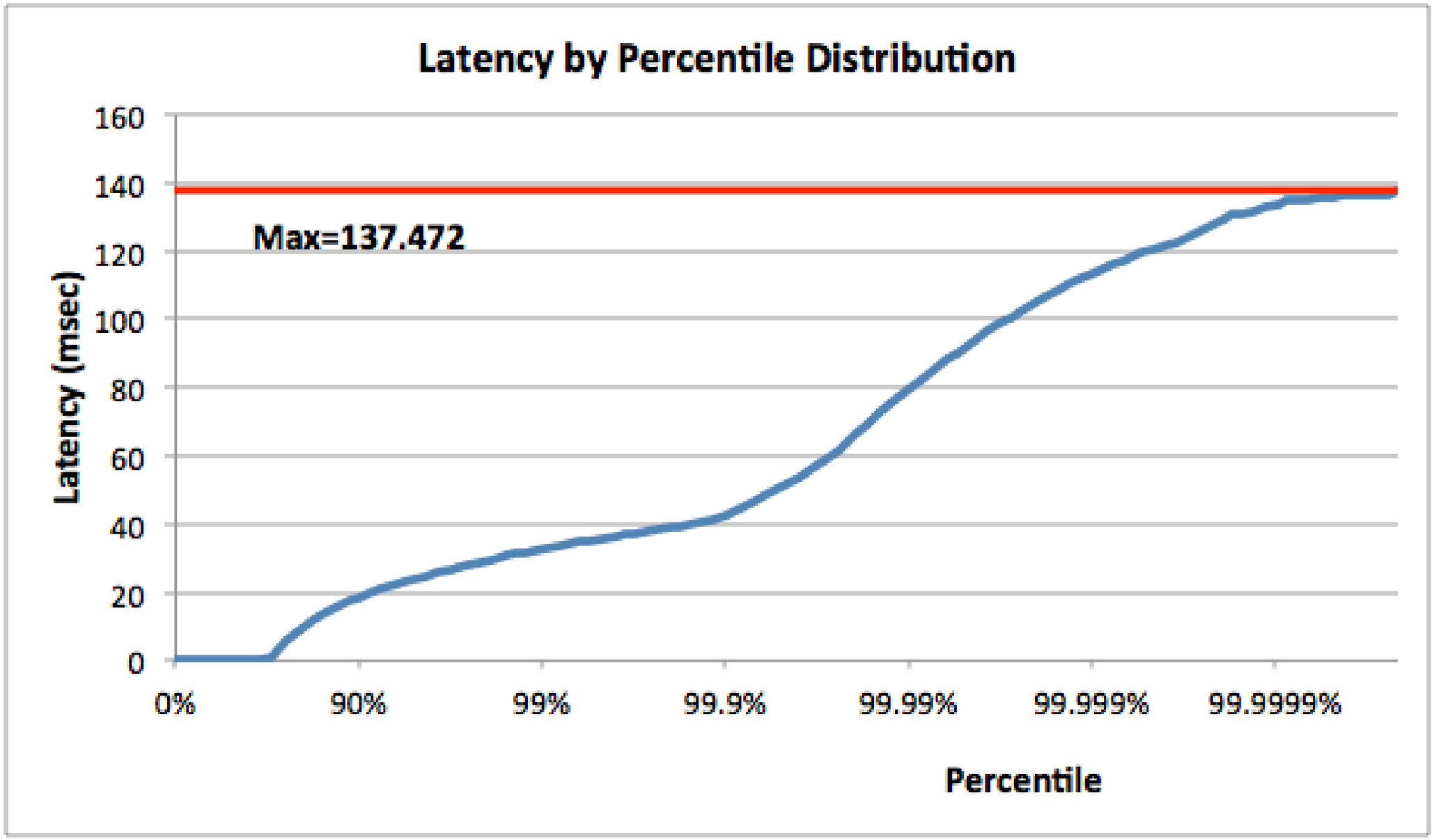
Suggestions

- 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 standard 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 charts like this...



Then you need both good dynamic range and good resolution

hdrhistogram.org

Google Maps | The Register | TheServerSide | Apple | Research | News | UIUC | eBay | Mechanical-Sympathy | FoJC | blogger

gram/HdrHisto... | mint_compare_exchang... | YOW 2014 Speakers Gu... | JetBrains Store | FileChannel (Java Pl

HdrHistogram

A High Dynamic Range (HDR) Histogram

View the Project on GitHub
HdrHistogram/HdrHistogram
(Java, C, and C# versions)

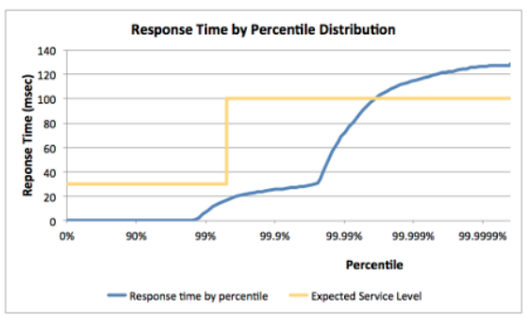
View the Project JavaDoc
hdrhistogram.github.com/HdrHistogram/JavaDoc

Download **ZIP File** | Download **TAR Ball**

View On **GitHub**

Plot histogram file(s)

An example plot of an HdrHistogram based full percentile spectrum plot:



HdrHistogram: A High Dynamic Range Histogram.

A Histogram that supports recording and analyzing sampled data value counts across a configurable integer value range with configurable value precision within the range. Value precision is expressed as the number of significant digits in the value recording, and provides control over value quantization behavior across the value range and the subsequent value resolution at any given level.

For example, a Histogram could be configured to track the counts of observed integer values between 0 and 3,600,000,000 while maintaining a value precision of 3 significant digits across that range. Value quantization within the range will thus be no larger than 1/1,000th (or 0.1%) of any value. This example Histogram could be used to track and analyze the counts of observed response times ranging between 1 microsecond and 1 hour in magnitude, while maintaining a value resolution of 1 microsecond up to 1 millisecond, a resolution of 1 millisecond (or better) up to one second, and a resolution of 1 second (or better) up to 1,000 seconds. At it's maximum tracked value (1 hour), it would still maintain a resolution of 3.6 seconds (or better).

HDR Histogram is designed for recoding histograms of value measurements in latency and performance sensitive applications. Measurements show value recording times as low as 3-6 nanoseconds on modern (circa 2014) Intel CPUs. The HDR Histogram maintains a fixed cost in both space and time. A Histogram's memory footprint is constant, with no allocation operations involved in recording data values or in iterating through them. The memory footprint is fixed regardless of the number of data value samples recorded, and depends solely on the dynamic range and precision chosen. The amount of work involved in recording a sample is constant, and directly computes storage index locations such that no iteration or searching is ever involved in recording data values.

Authors, Contributors, and License

HdrHistogram was authored by Gil Tene (@giltene) (original and Java version), with ports by Mike Barker (@mikeb01) (C), and Matt Warren (@mattwarren) (C#), and placed in the public domain, as explained at http://creativecommons.org/publicdomain/zero/1.0/

Support or Contact

Don't call me, I won't call you.

AZUL SYSTEMS®

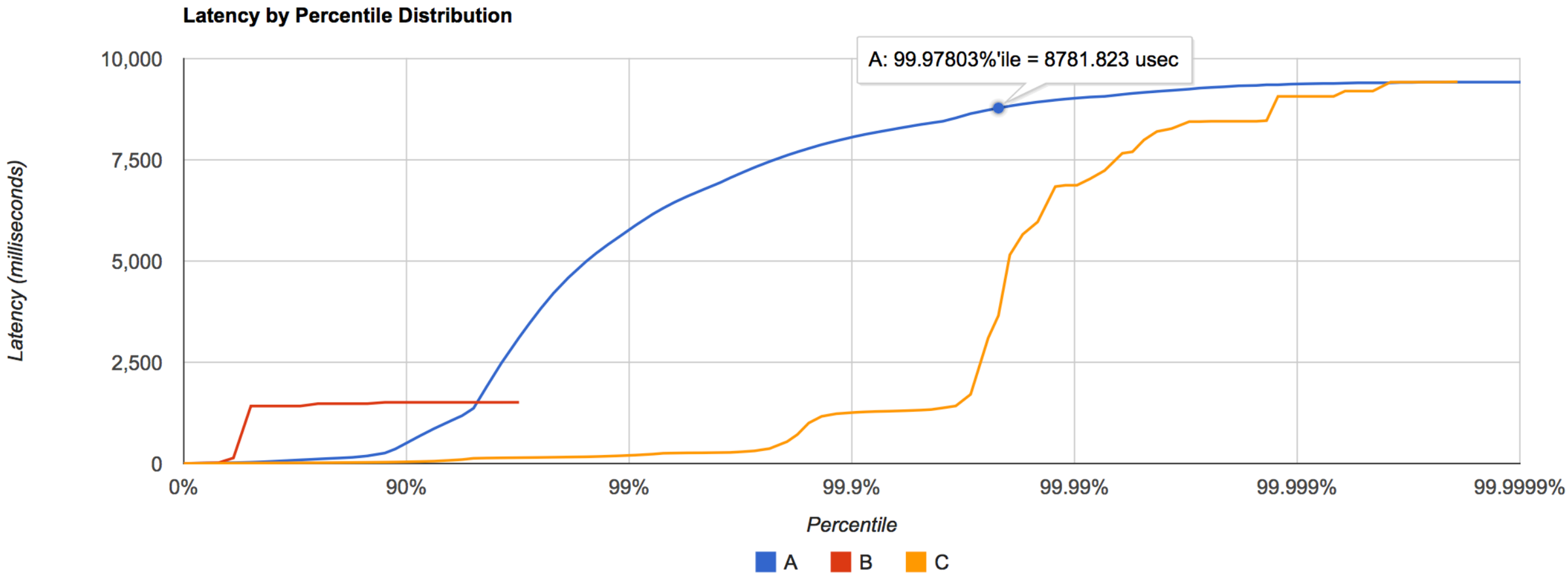
hdrhistogram.github.io

Google Maps | The Register | TheServerSide | Apple | Research | News | UIUC | eBay | Mechanical-Sympathy | FoJC | blogger

HdrHistogram Plotter

Choose Files | files selected

Please select file(s) above.



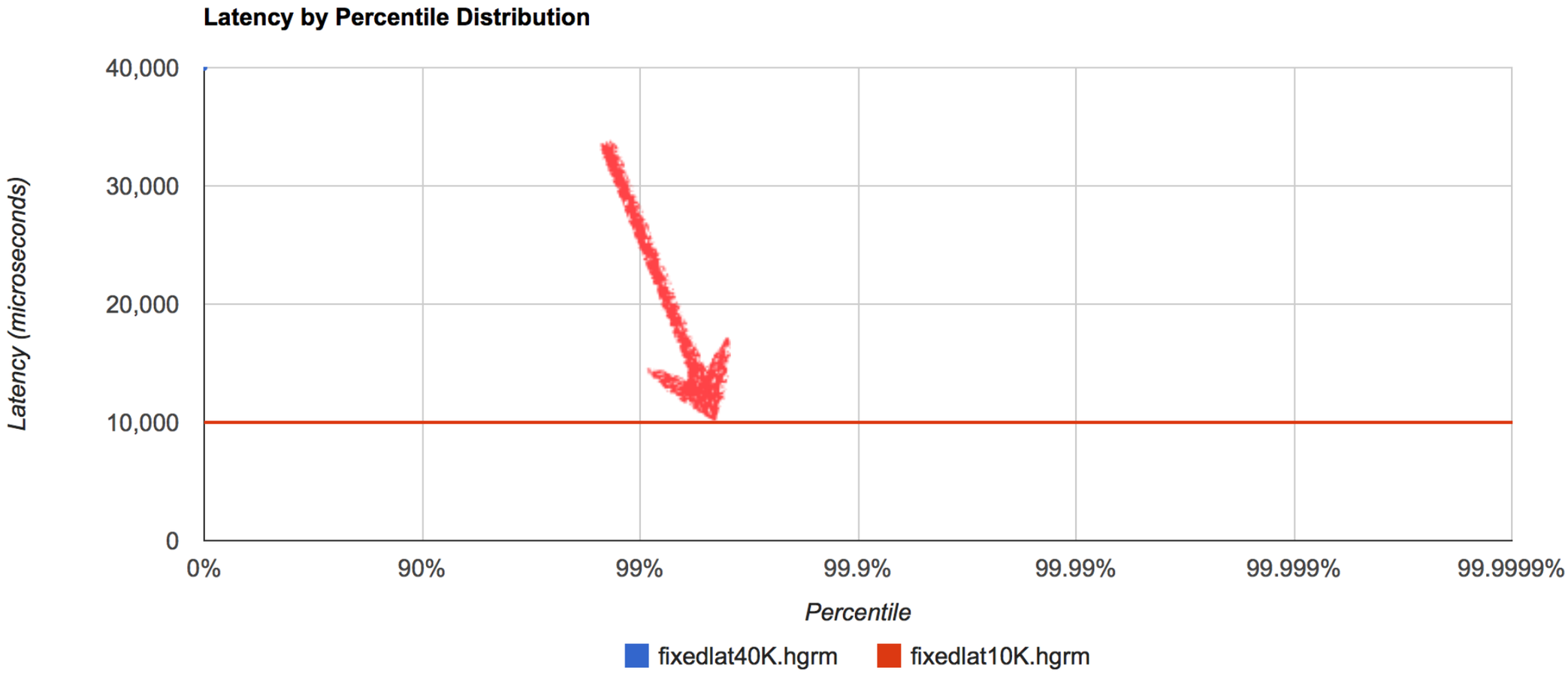
Latency time units: milliseconds | Export Image

Percentile range: 99.99999%

*** Note: Input files are expected to be in the .hgrm format produced by HistogramLogProcessor, or the percentile output format for HdrHistogram. See example file format here

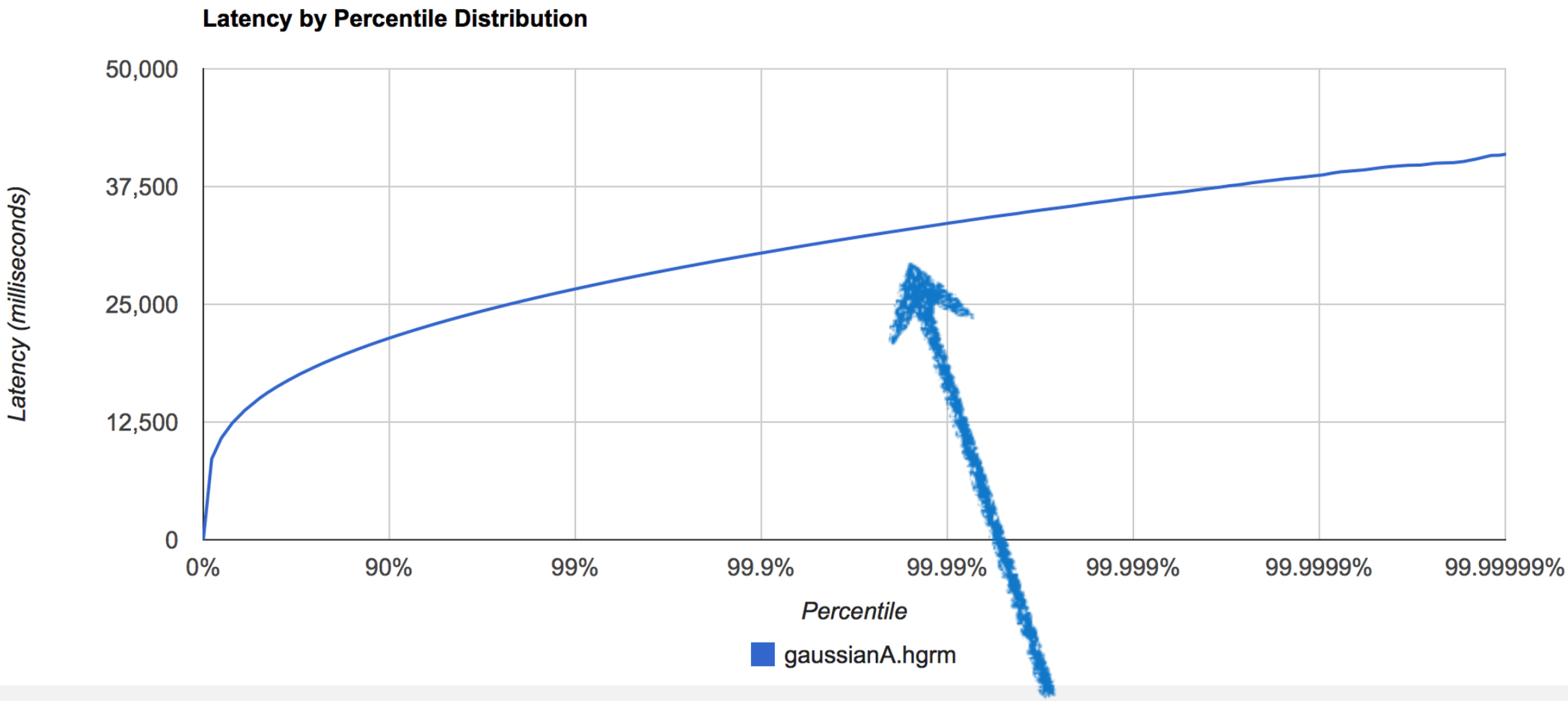
AZUL SYSTEMS®

Shape of *Constant* latency



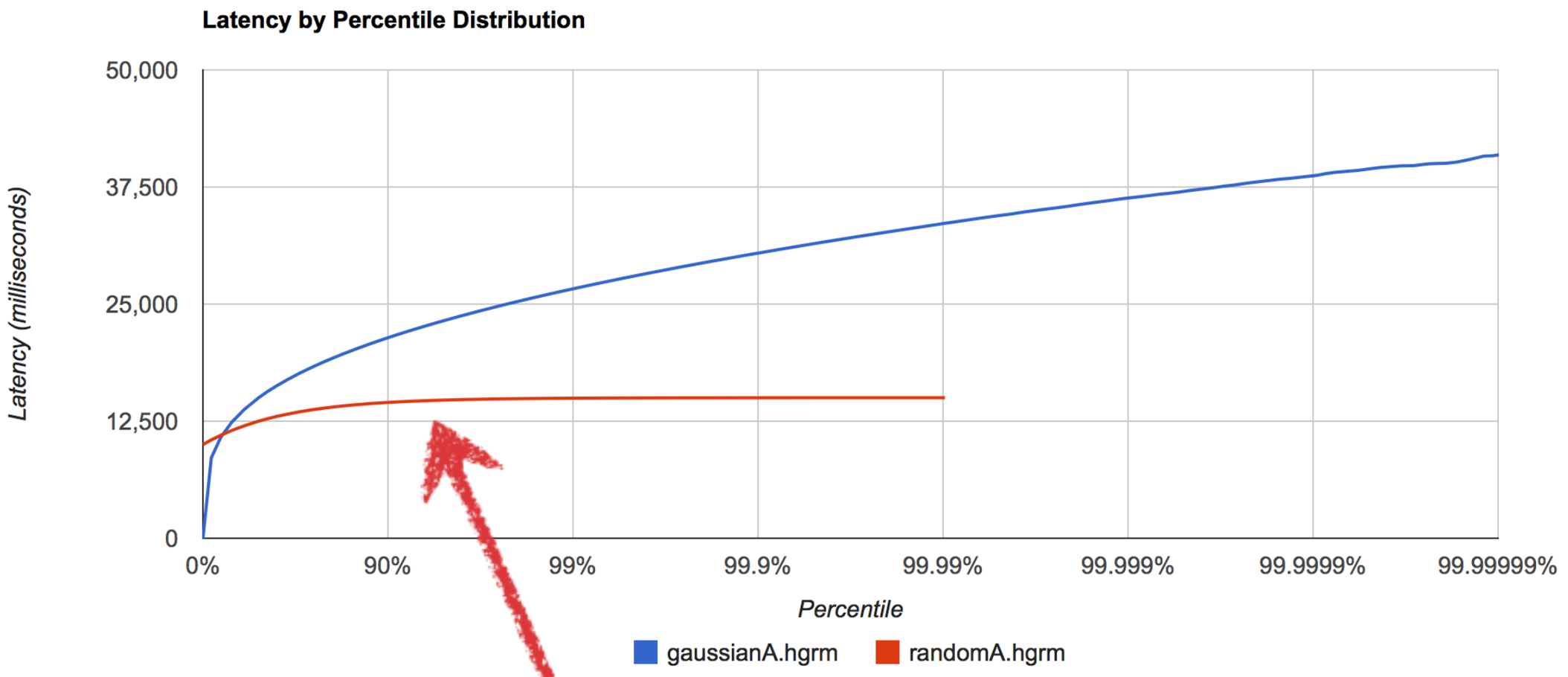
10K fixed line latency

Shape of *Gaussian* latency



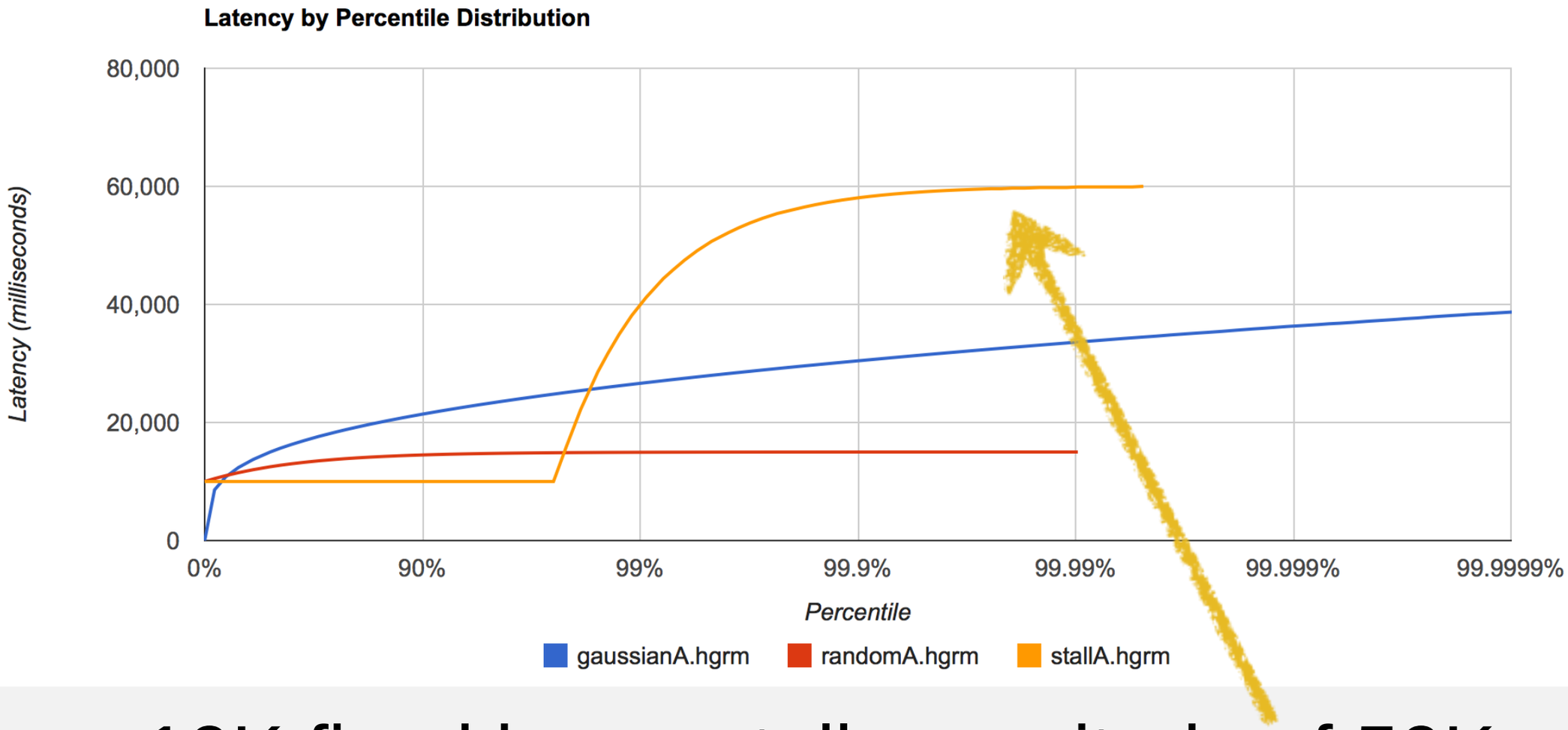
10K fixed line latency with added Gaussian noise (std dev. = 5K)

Shape of ***Random*** latency



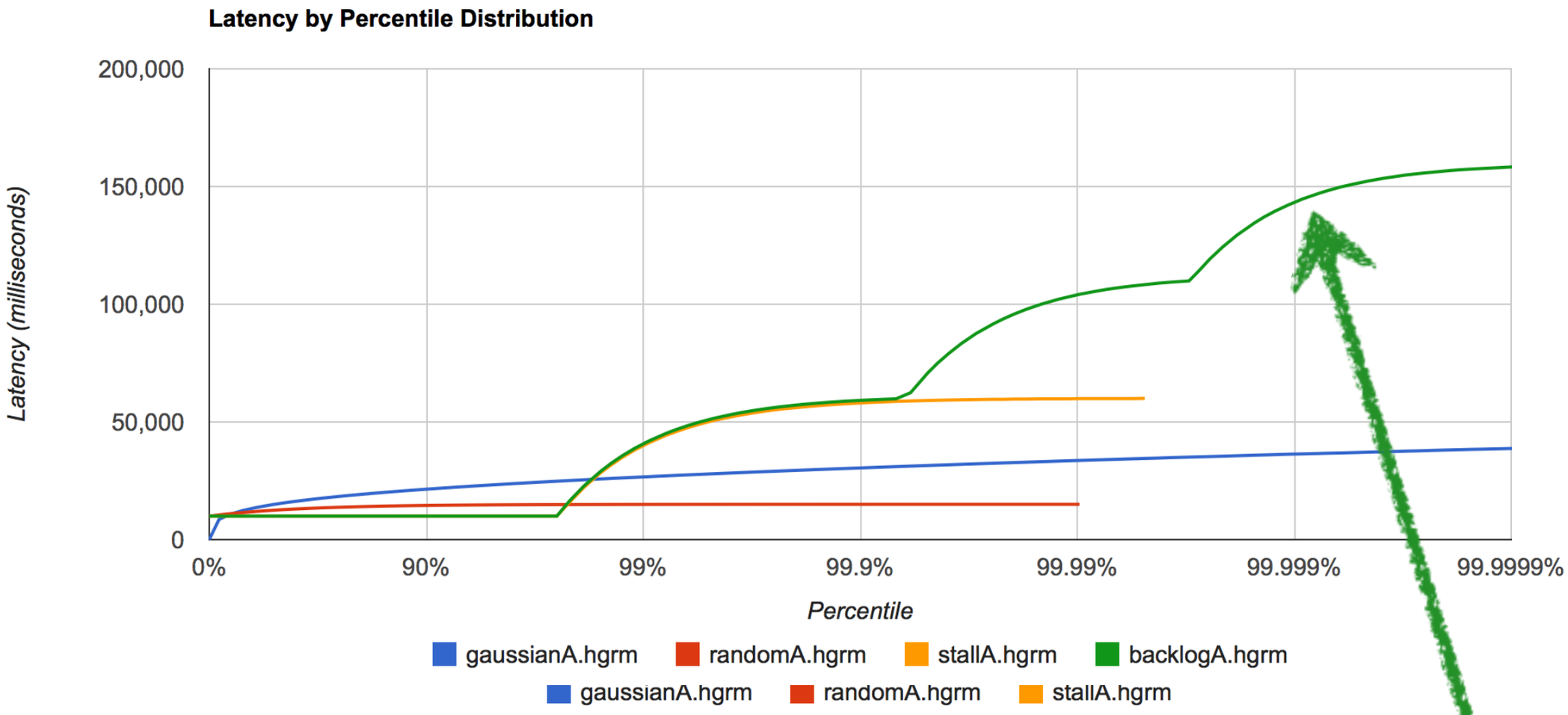
10K fixed line latency with added Gaussian (std dev. = 5K) vs. random (+5K)

Shape of *Stalling* latency



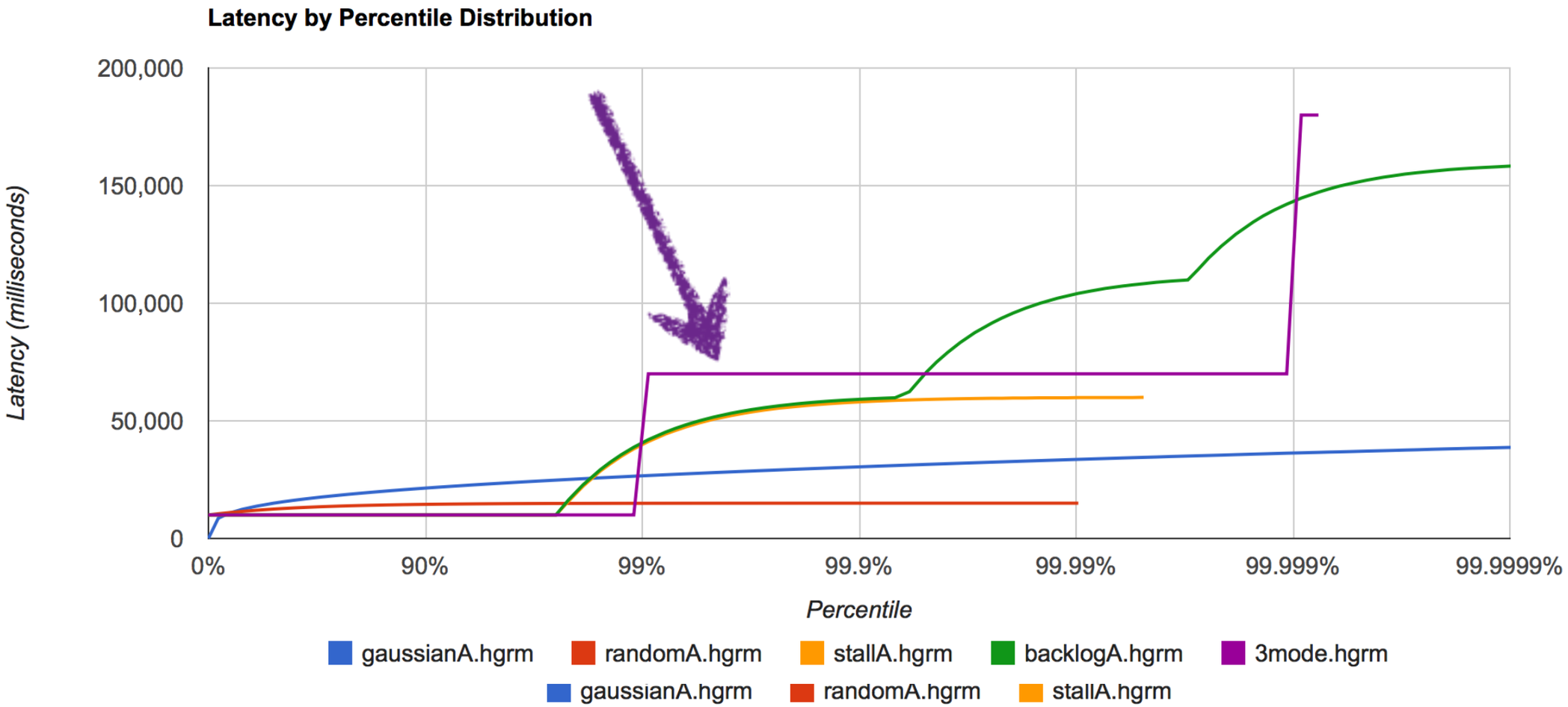
10K fixed base, stall magnitude of 50K
stall likelihood = 0.00005 (interval = 100)

Shape of *Queuing* latency



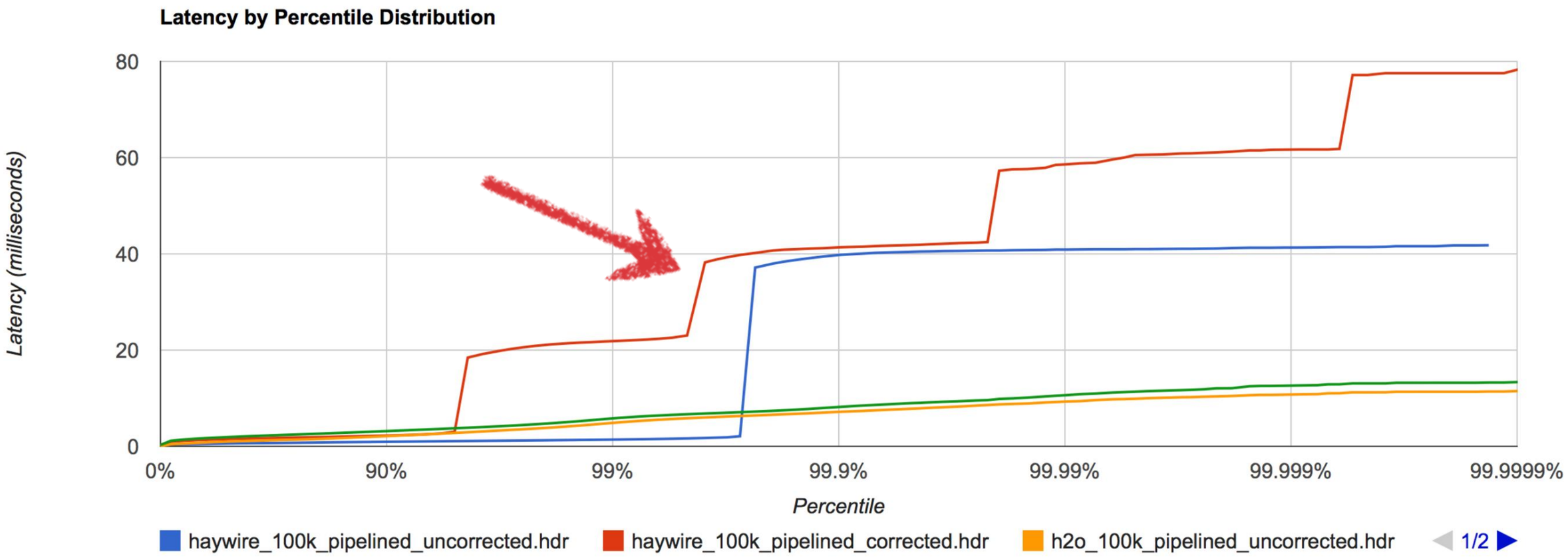
10K fixed base, occasional bursts of 500 msgs
handling time = 100, burst likelihood = 0.00005

Shape of ***Multi Modal*** latency



10K mode0 70K mode1 (likelihood 0.01)
180K mode2 (likelihood 0.00001)

And this what the real(?) world sometimes looks like…

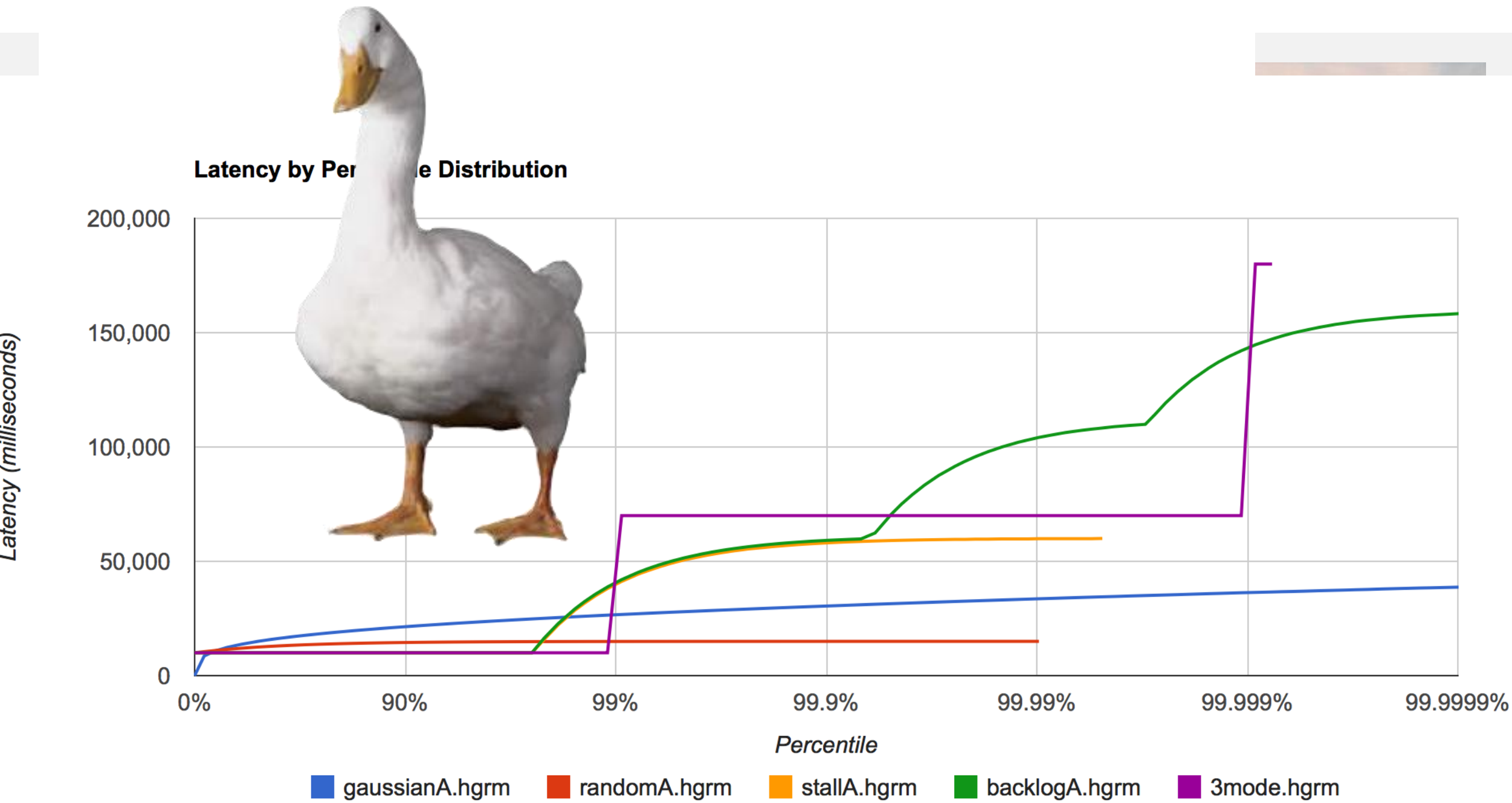


Kelly Sommers
@kellabyte

@giltene Yeah. I have no idea how to do that yet. Like, wtf is going on in the code here?! i.imgur.com/XMYH62w.png

11/15/14, 1:29 AM

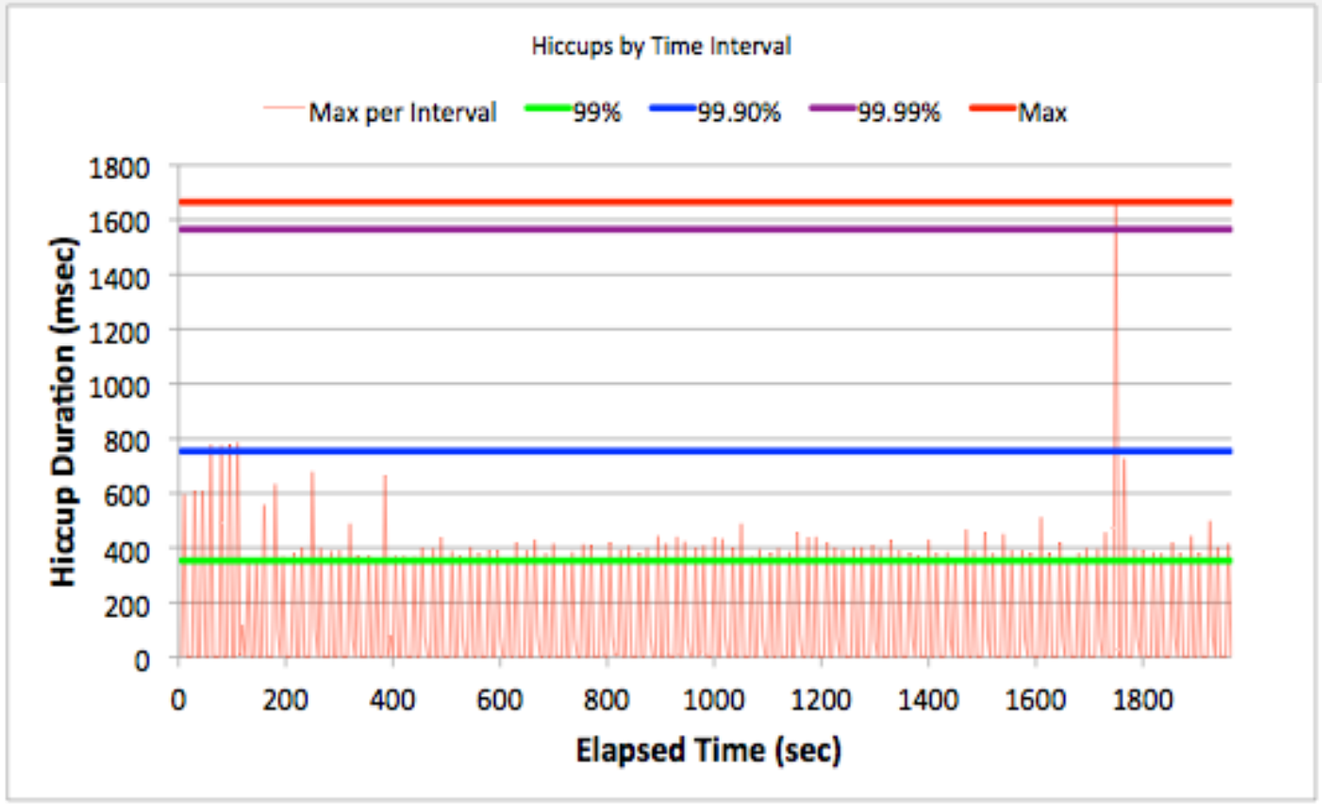
Real world “deductive reasoning”

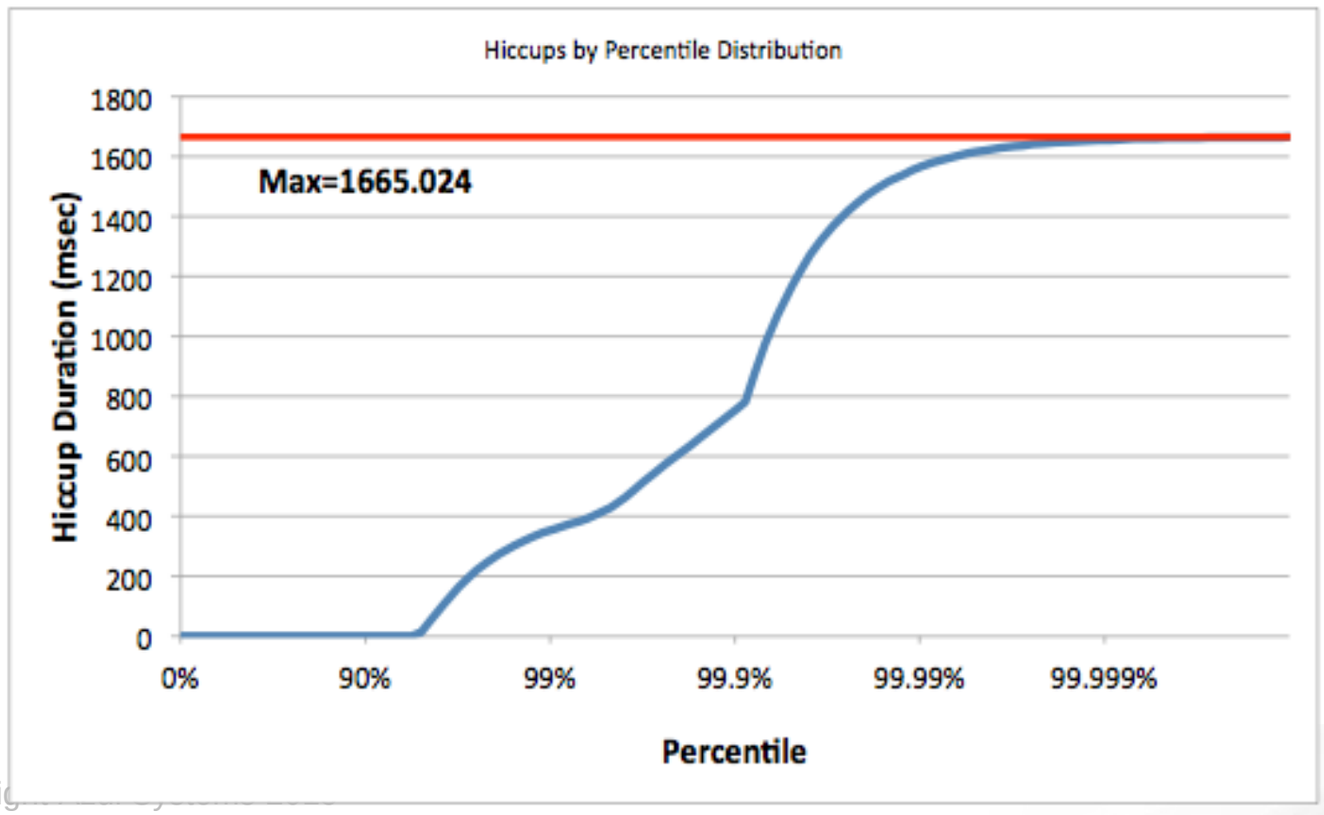


http://www.jhiccup.org

jHiccup

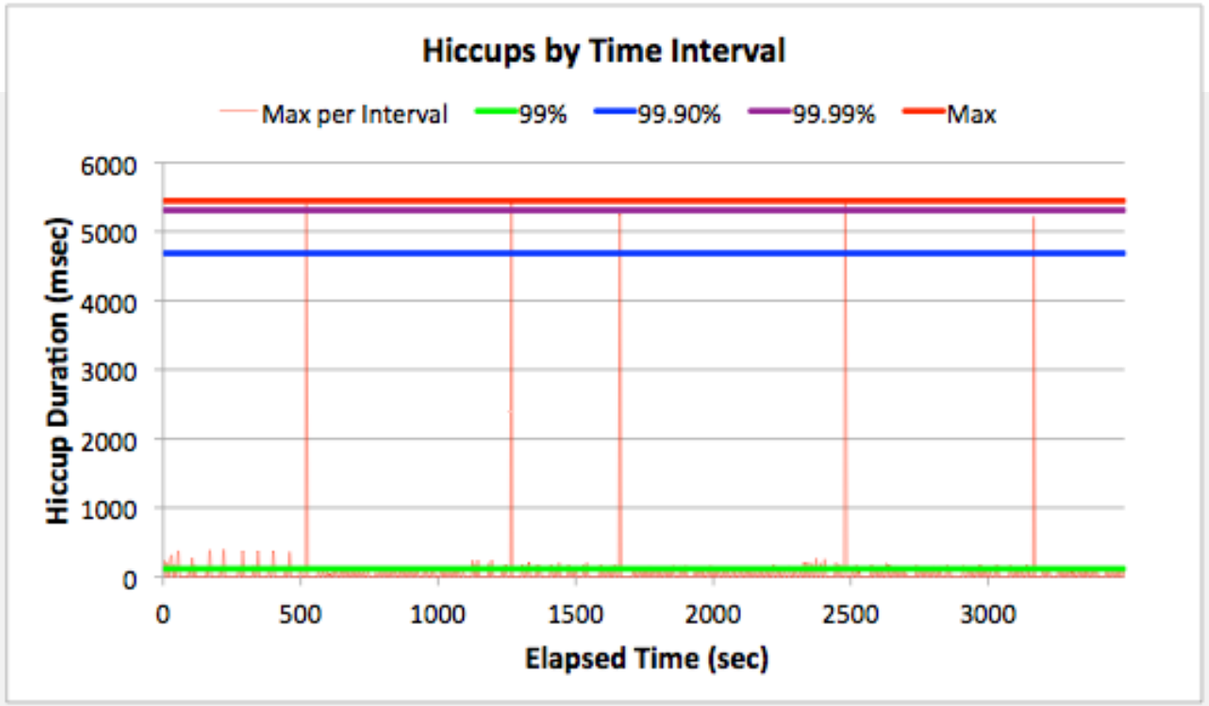
Discontinuity in Java execution

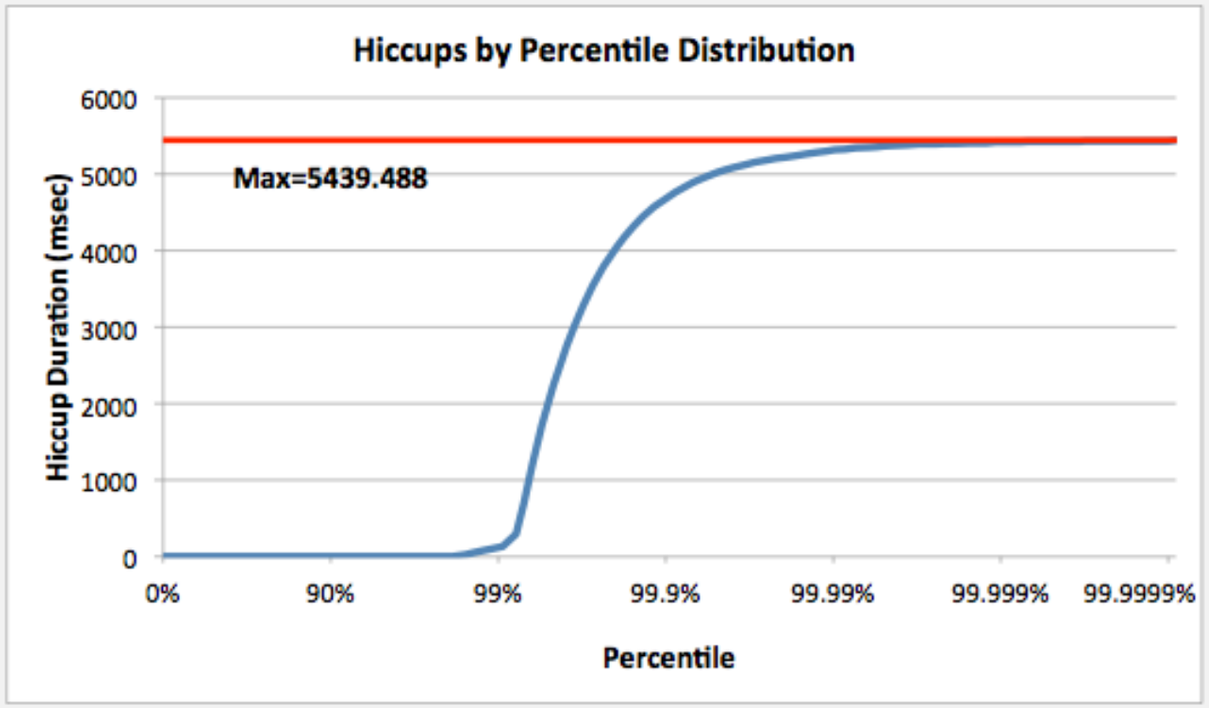




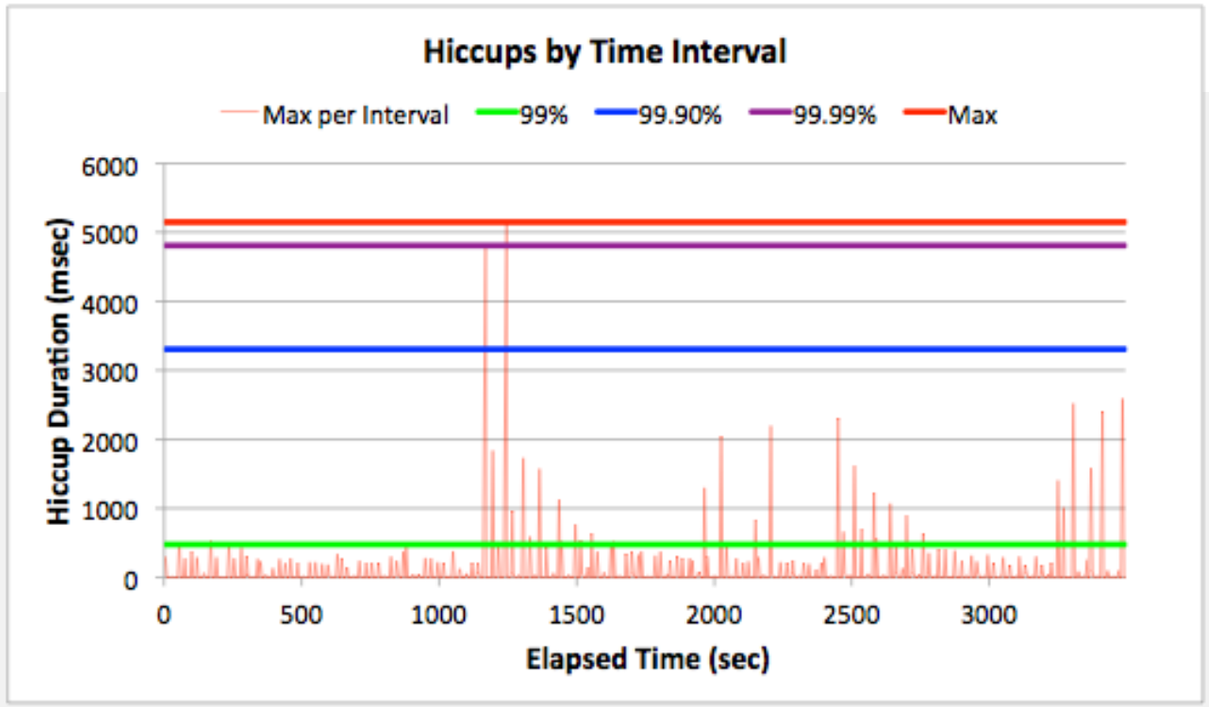
Examples

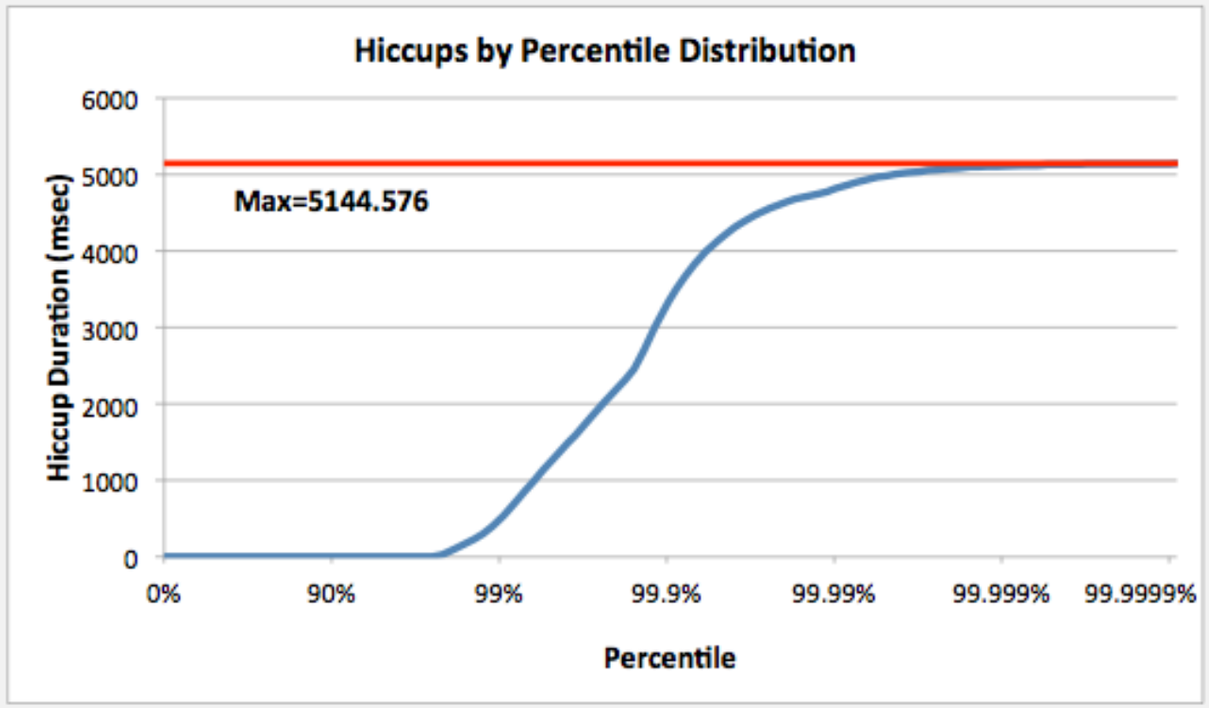
Oracle HotSpot ParallelGC





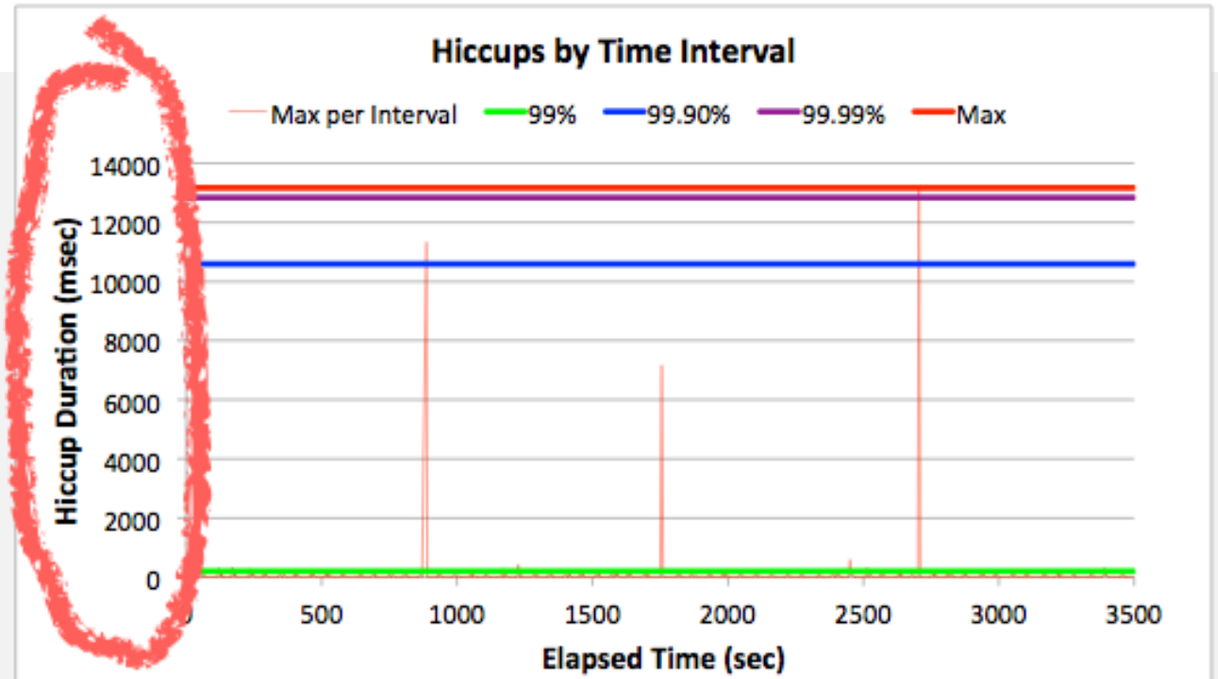
Oracle HotSpot G1

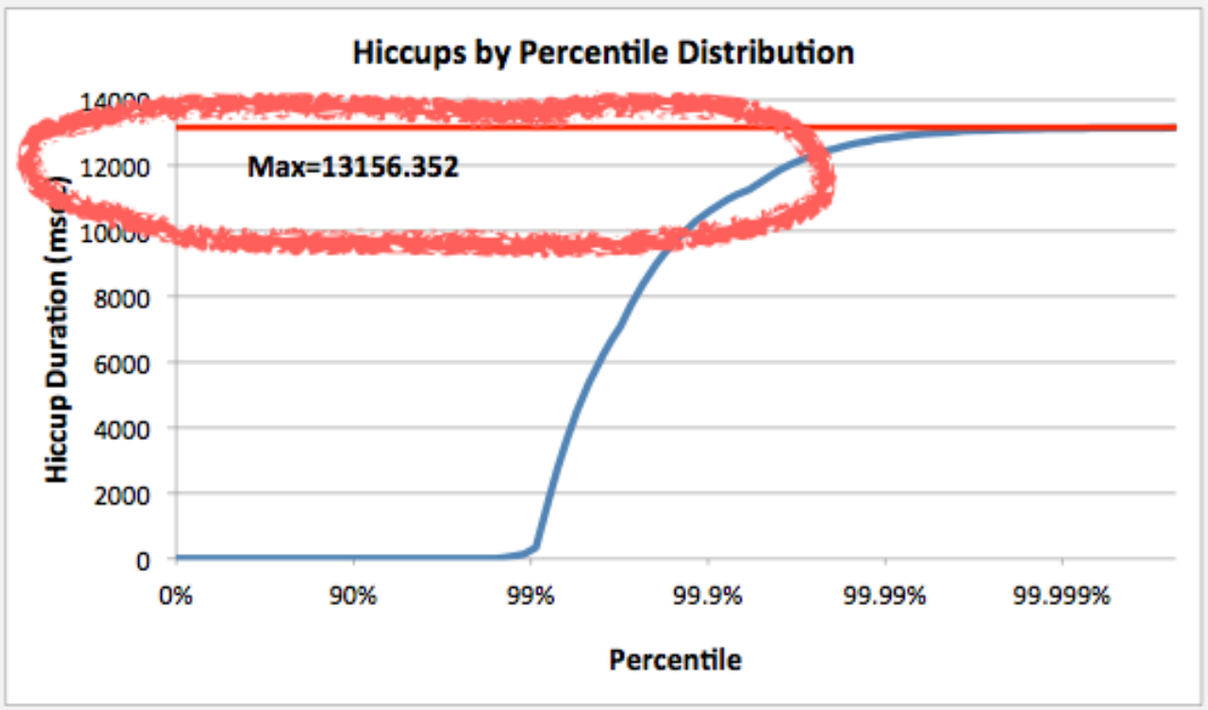




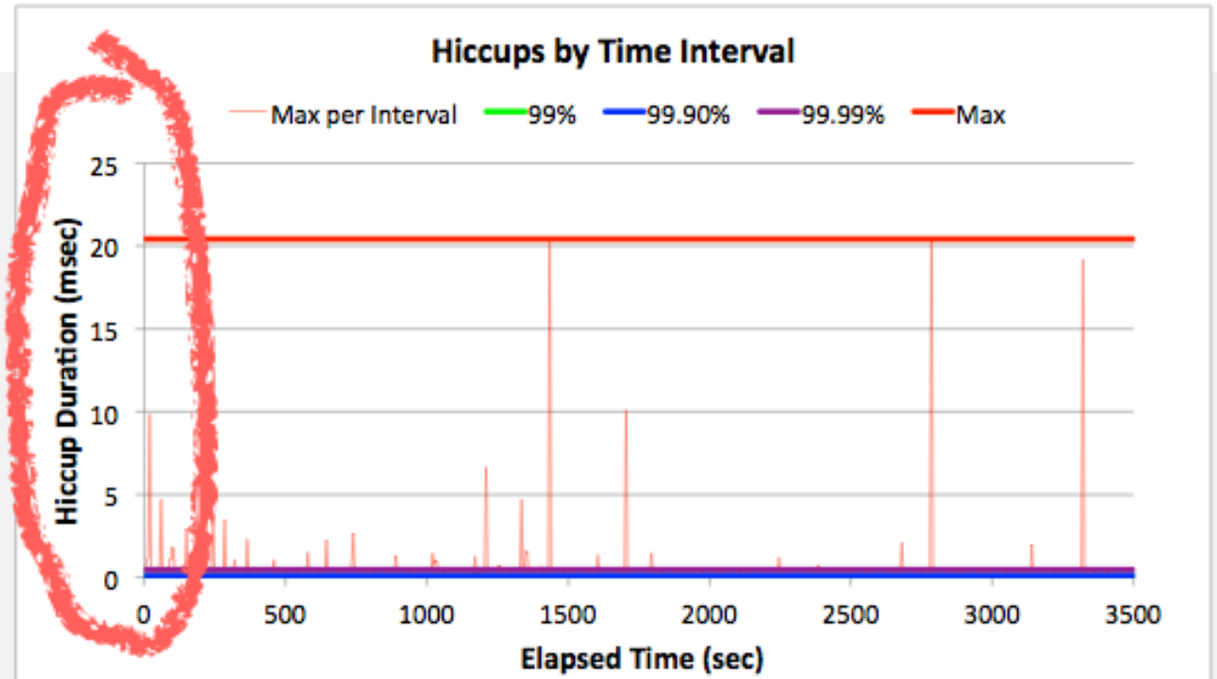
1GB live set in 8GB heap, same app, same HotSpot, different GC

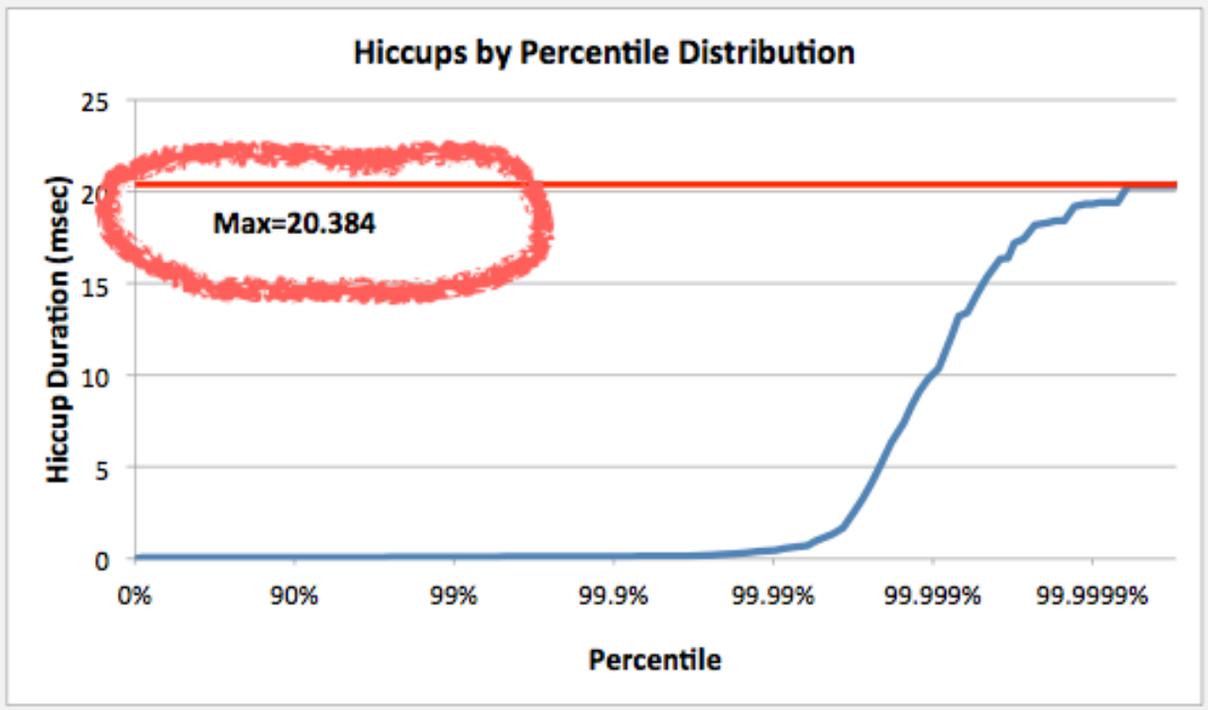
Oracle HotSpot CMS





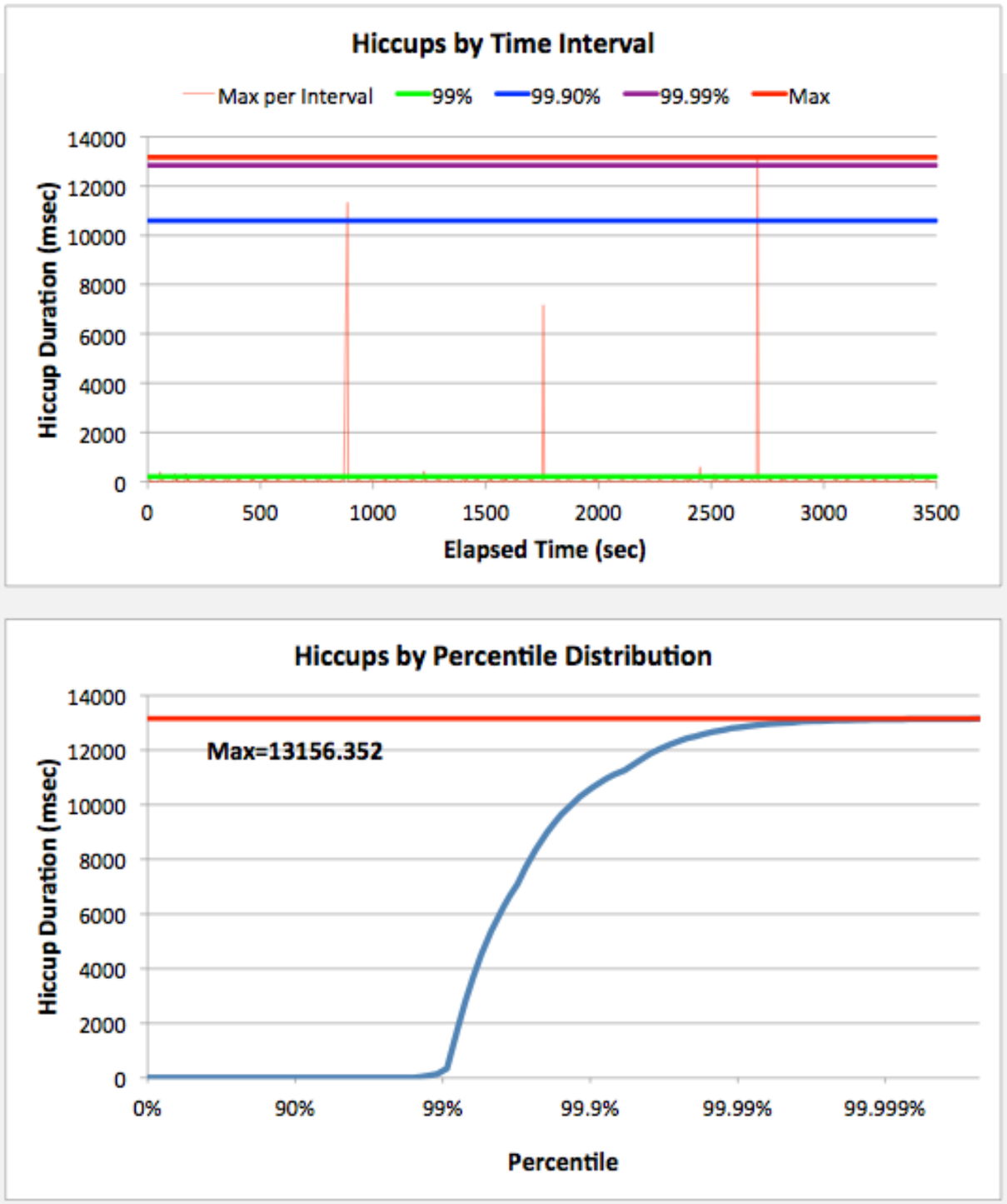
Zing Pauseless GC



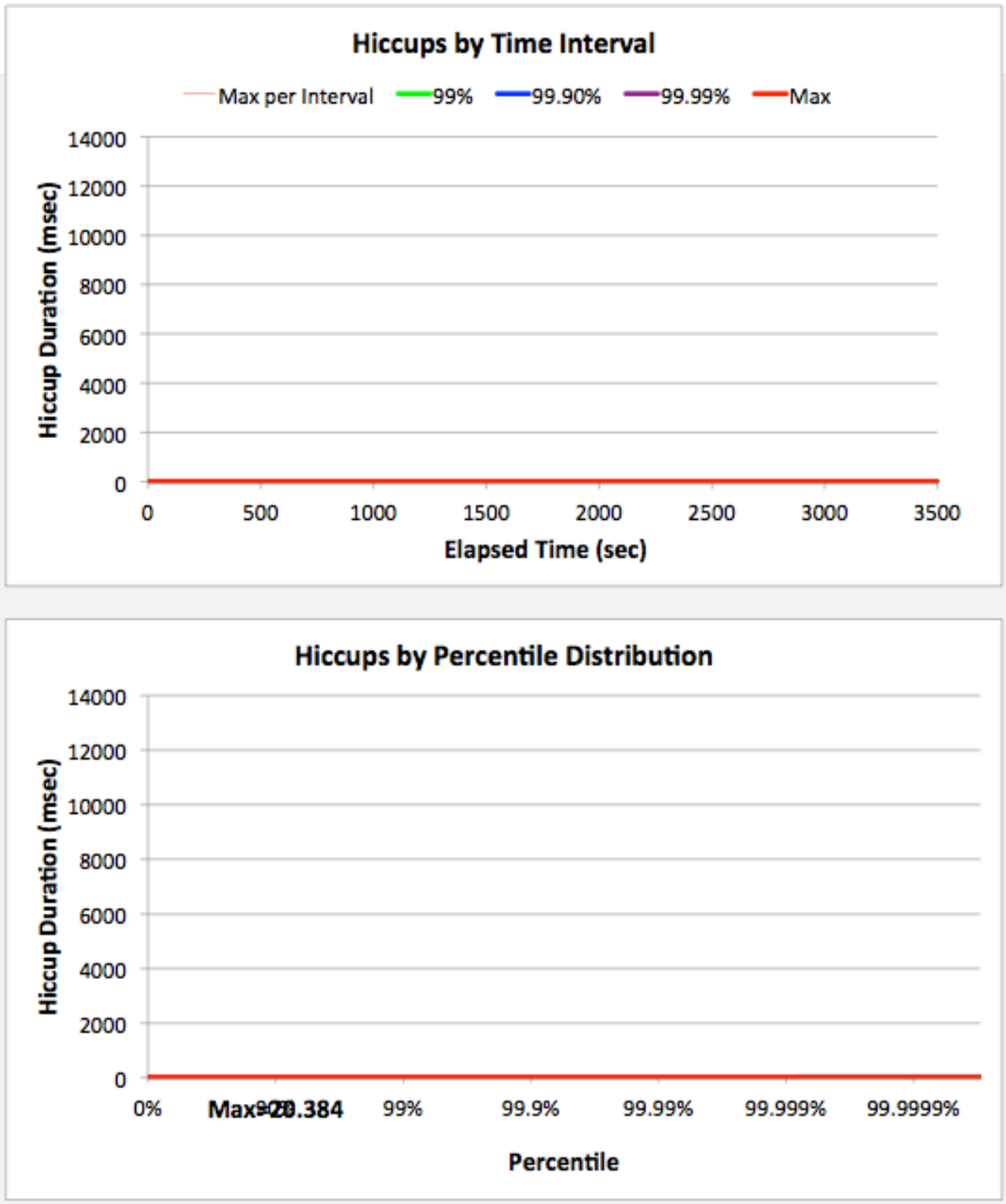


1GB live set in 8GB heap, same app, different JVM/GC

Oracle HotSpot CMS

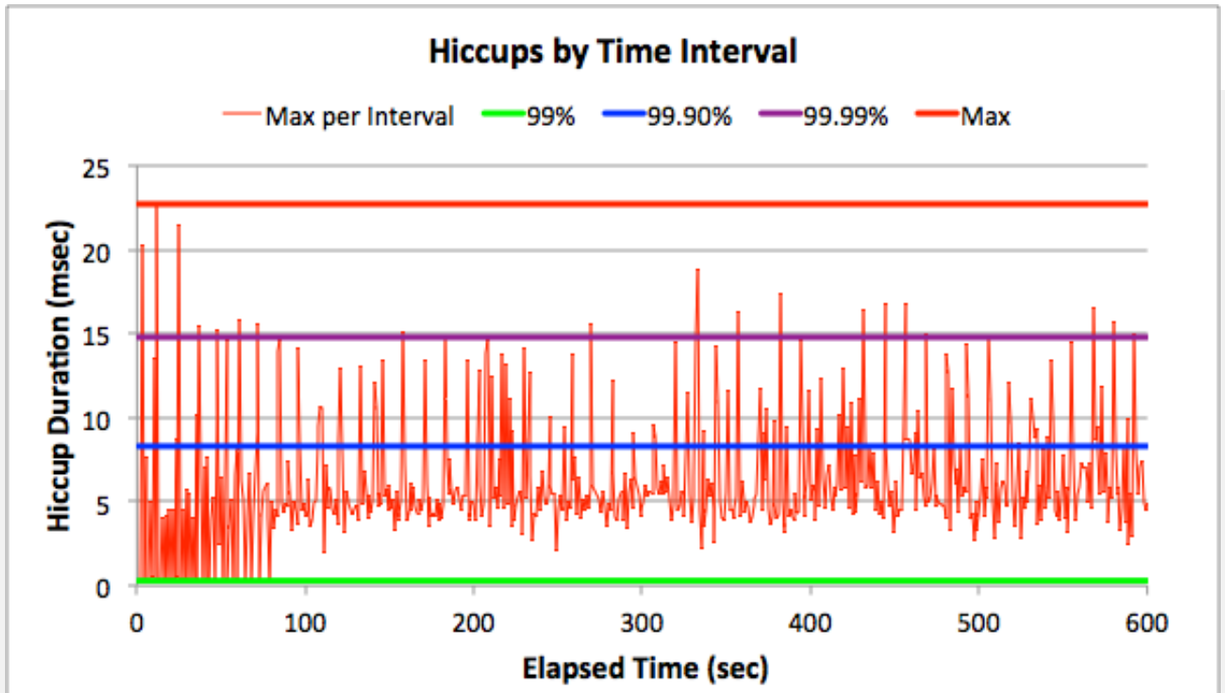


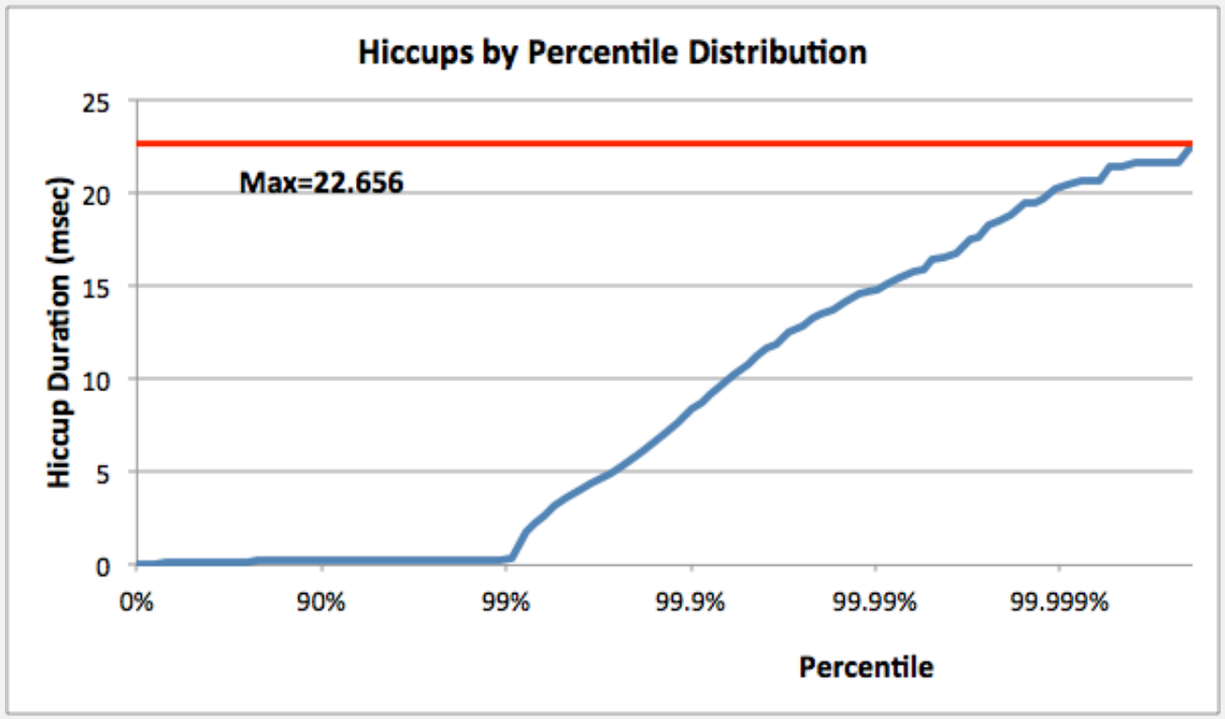
Zing Pauseless GC



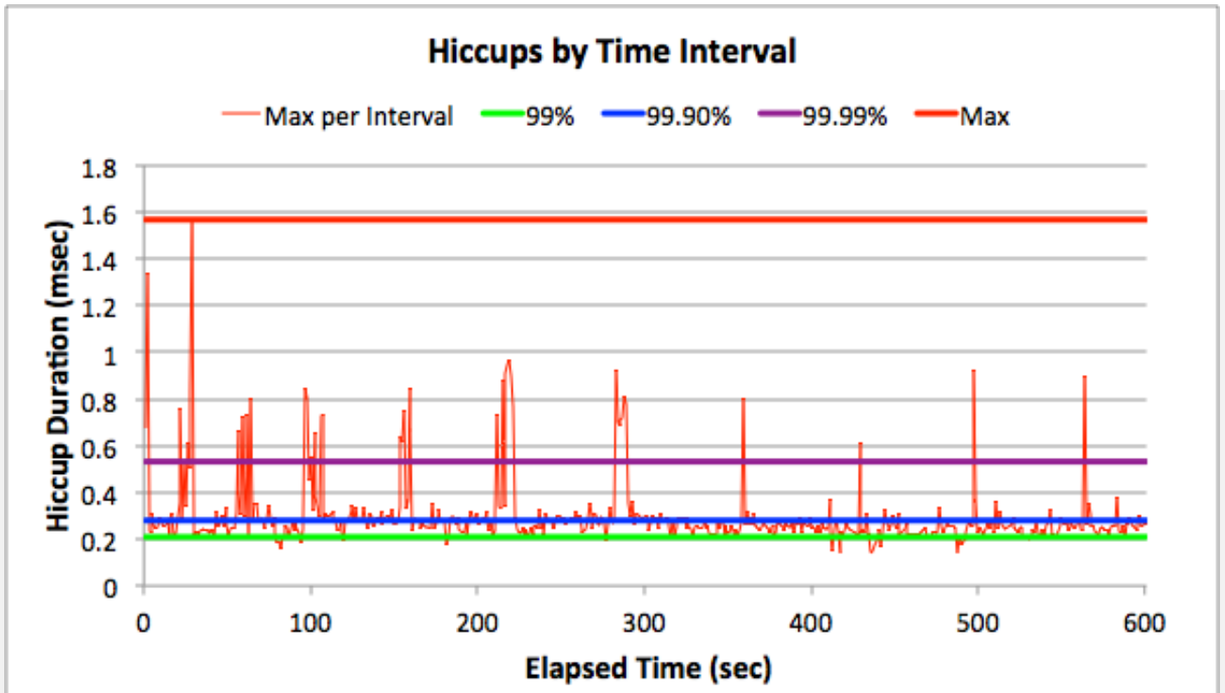
1GB live set in 8GB heap, same app, different JVM/GC- **drawn to scale**

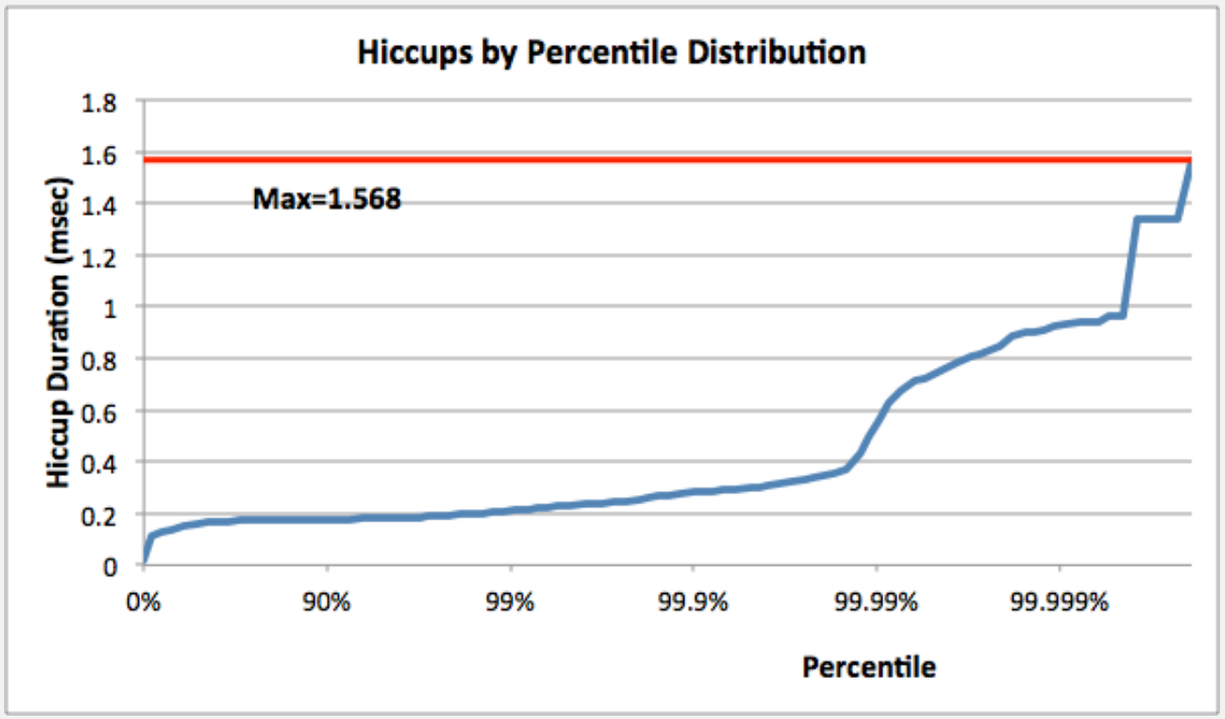
Oracle HotSpot (pure NewGen)





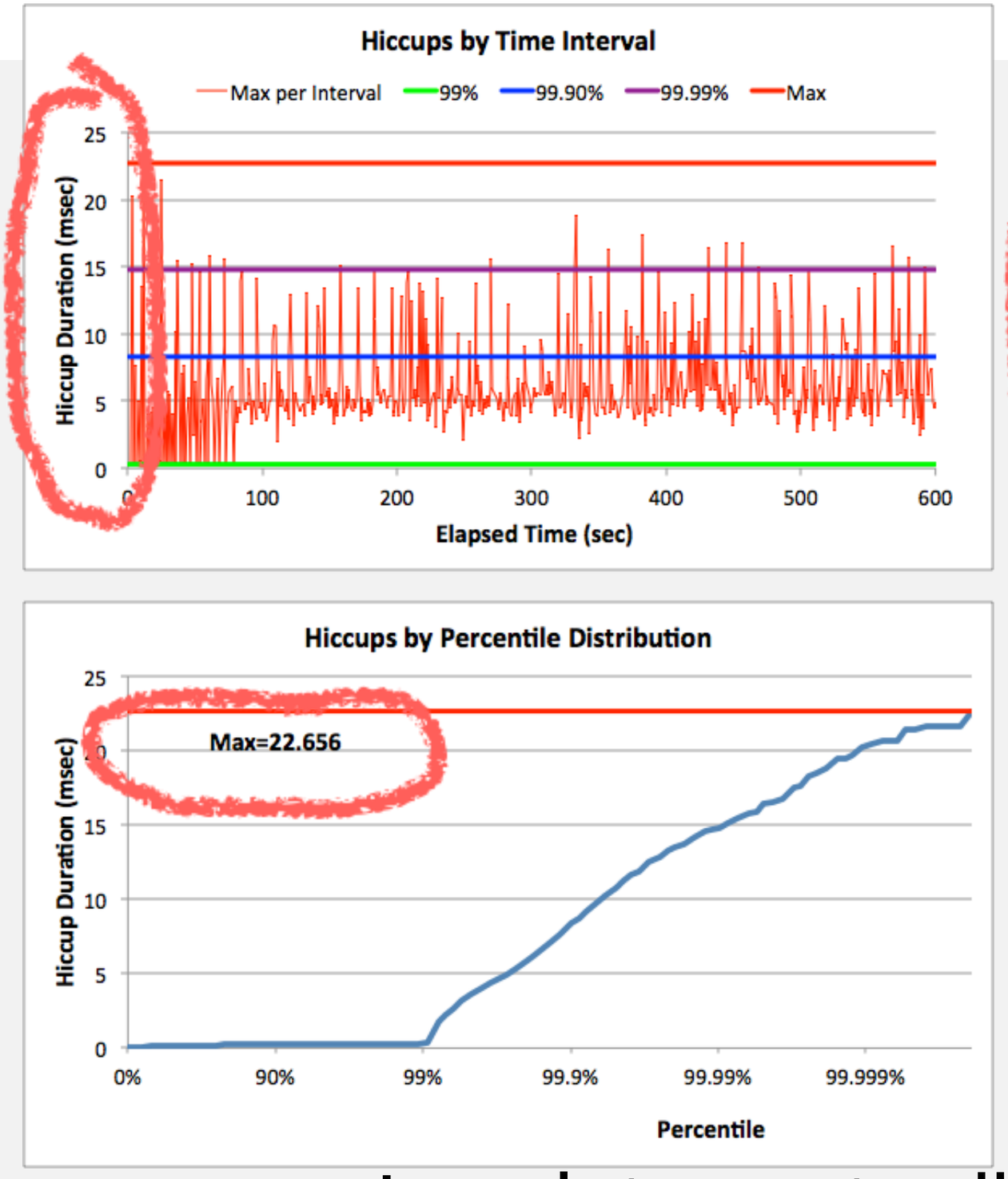
Zing Pauseless GC

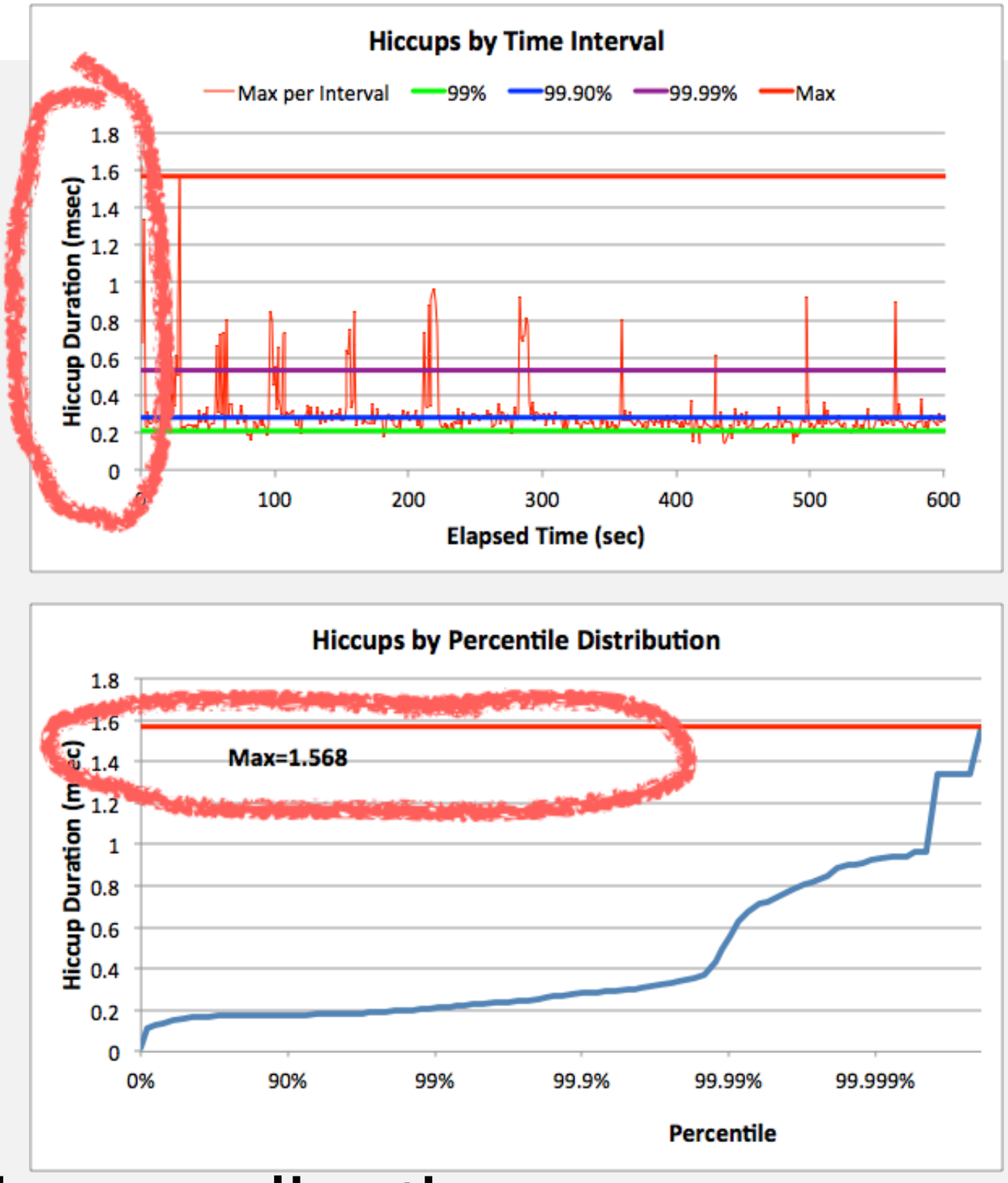




Low latency trading application

Oracle HotSpot (pure NewGen) Zing Pauseless GC

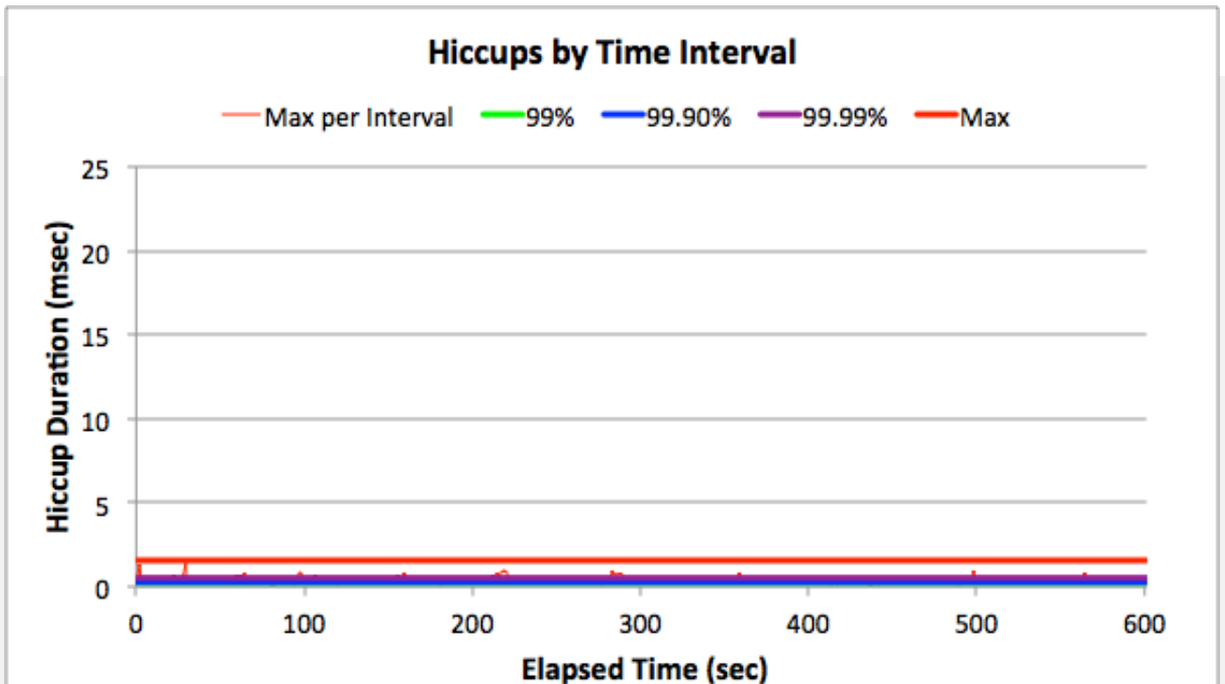


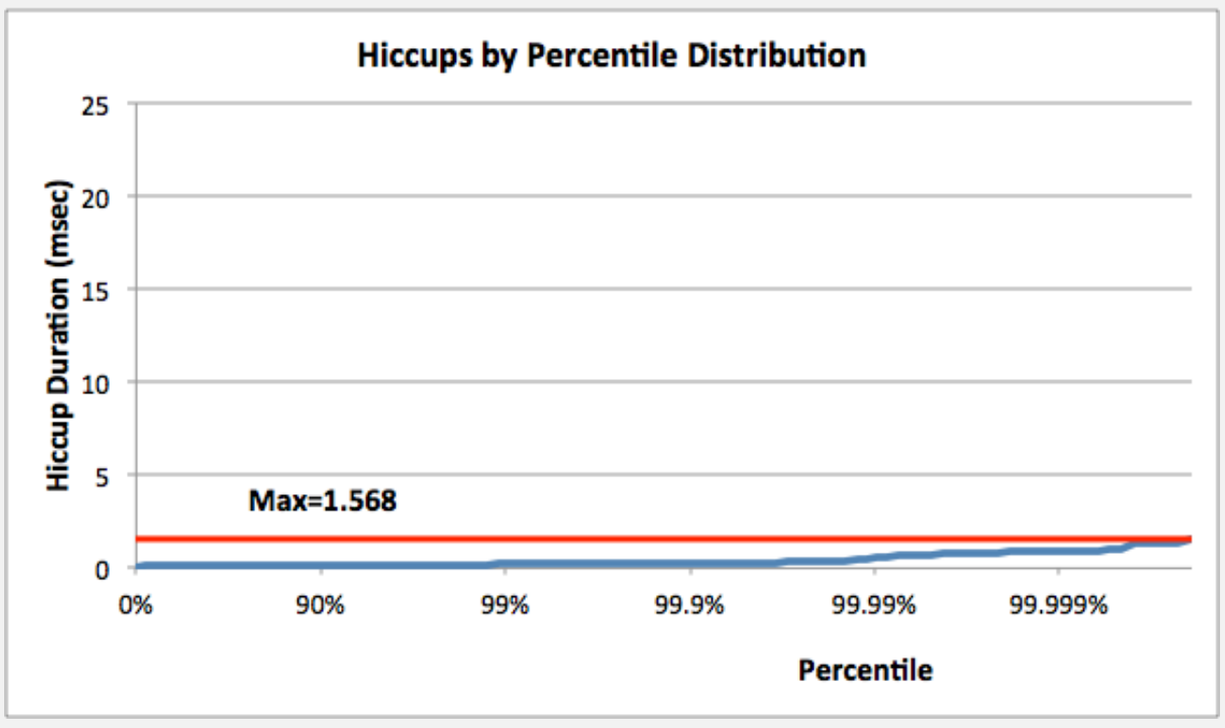


Low latency trading application

Oracle HotSpot (pure NewGen)

Zing Pauseless GC





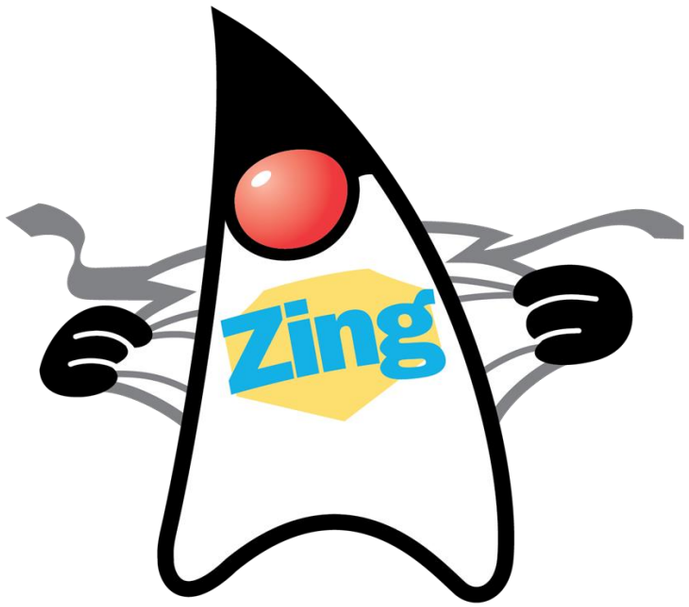
Low latency trading application – **drawn to scale**

Q & A

www.azul.com

www.jhiccup.com

www.hdrhistogram.com



@schuetzematt