Stop the Guessing

Performance Methodologies for Production Systems

Brendan Gregg

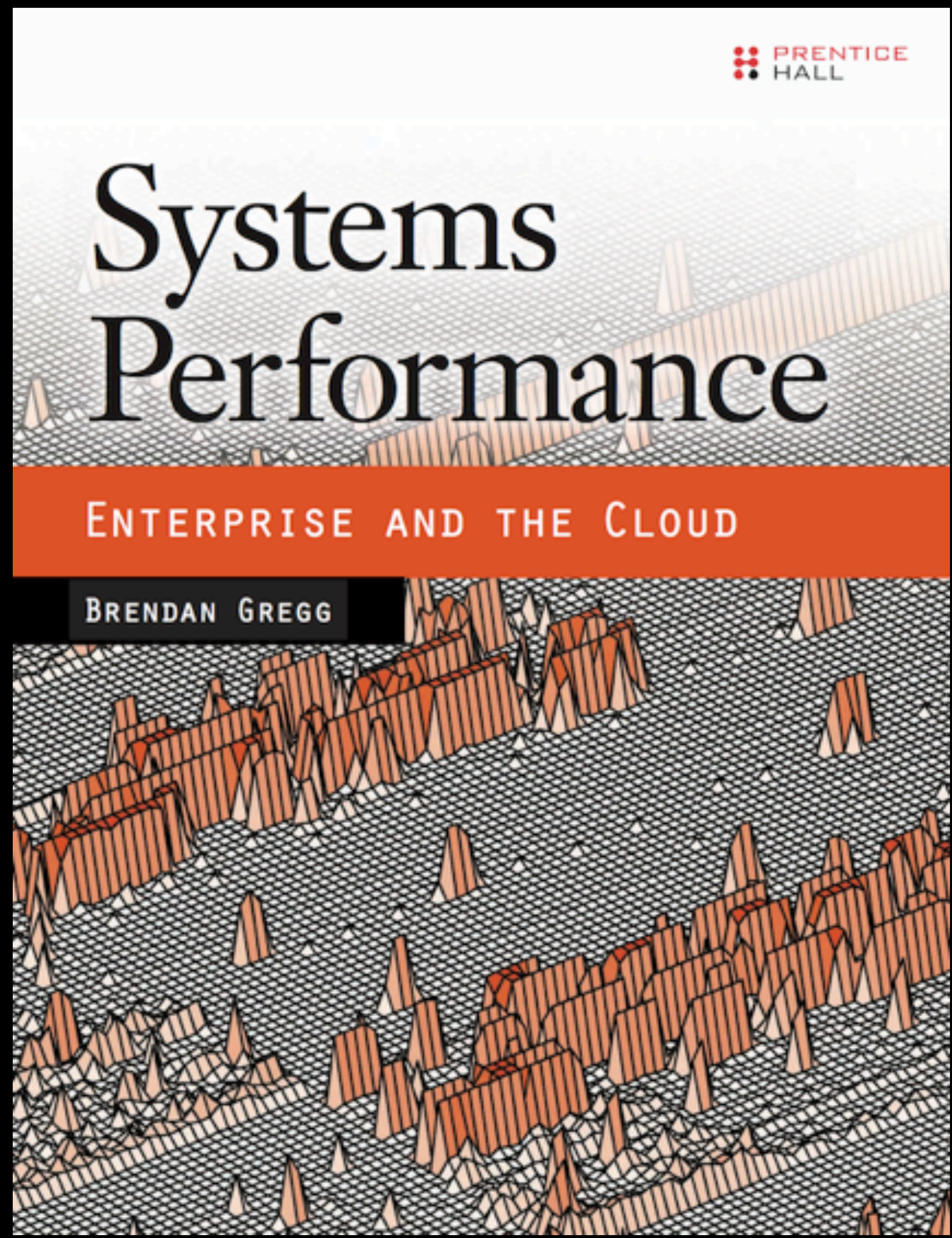
Lead Performance Engineer, Joyent

Audience

- This is for developers, support, DBAs, sysadmins
- When perf isn't your day job, but you want to:
  - Fix common performance issues, quickly
  - Have guidance for using performance monitoring tools
- Environments with small to large scale production systems

whoami

- Lead Performance Engineer: analyze everything from apps to metal
- Work/Research: tools, visualizations, methodologies
- Methodologies is the focus of my next book



Joyent

- High-Performance Cloud Infrastructure
  - Public/private cloud provider
- OS Virtualization for bare metal performance
- KVM for Linux and Windows guests
- Core developers of SmartOS and node.js

Performance Analysis

- Where do I start?
- Then what do I do?

Performance Methodologies

- Provide
  - Beginners: a starting point
  - Casual users: a checklist
  - Guidance for using existing tools: pose questions to ask
- The following six are for production system monitoring

Production System Monitoring

- Guessing Methodologies
  - 1. Traffic Light Anti-Method
  - 2. Average Anti-Method
  - 3. Concentration Game Anti-Method
- Not Guessing Methodologies
  - 4. Workload Characterization Method
  - 5. USE Method
  - 6. Thread State Analysis Method

Traffic Light Anti-Method

Traffic Light Anti-Method

- 1. Open monitoring dashboard
- 2. All green? Everything good, mate.

= BAD

= GOOD

Traffic Light Anti-Method, cont.

- Performance is subjective
  - Depends on environment, requirements
  - No universal thresholds for good/bad
- Latency outlier example:
  - customer A) 200 ms is bad
  - customer B) 2 ms is bad (an “eternity”)
- Developer may have chosen thresholds by guessing

Traffic Light Anti-Method, cont.

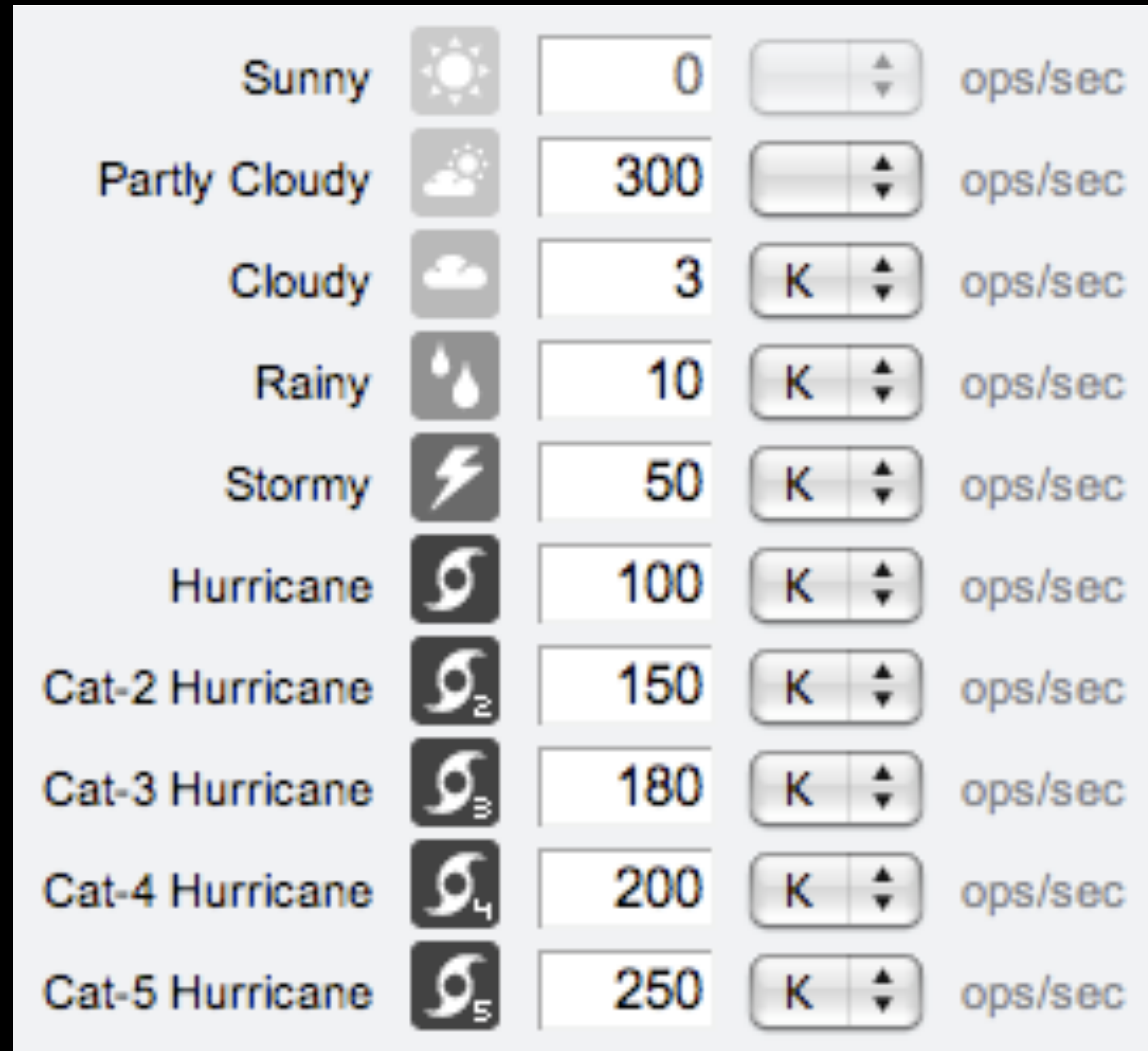
- Performance is complex
  - Not just one threshold required, but multiple different tests
- For example, a disk traffic light:
  - Utilization-based: one disk at 100% for less than 2 seconds means green (variance), for more than 2 seconds is red (outliers or imbalance), but if all disks are at 100% for more than 2 seconds, that may be green (FS flush) provided it is async write I/O, if sync then red, also if their IOPS is less than 10 each (errors), that's red (sloth disks), unless those I/O are actually huge, say, 1 Mbyte each or larger, as that can be green, ... etc ...
  - Latency-based: I/O more than 100 ms means red, except for async writes which are green, but slowish I/O more than 20 ms can red in combination, unless they are more than 1 Mbyte each as that can be green ...

Traffic Light Anti-Method, cont.

- Types of error:
  - I. False positive: red instead of green
    - Team wastes time
  - II. False negative: green insead of red
    - Performance issues remain undiagnosed
    - Team wastes *more* time looking elsewhere

Traffic Light Anti-Method, cont.

- Subjective metrics (opinion):
  - utilization, IOPS, latency
- Objective metrics (fact):
  - errors, alerts, SLAs
- For subjective metrics, use weather icons
  - implies an inexact science, with no hard guarantees
  - also attention grabbing
- A dashboard can use both as appropriate for the metric



http://dtrace.org/blogs/brendan/2008/11/10/status-dashboard

Traffic Light Anti-Method, cont.

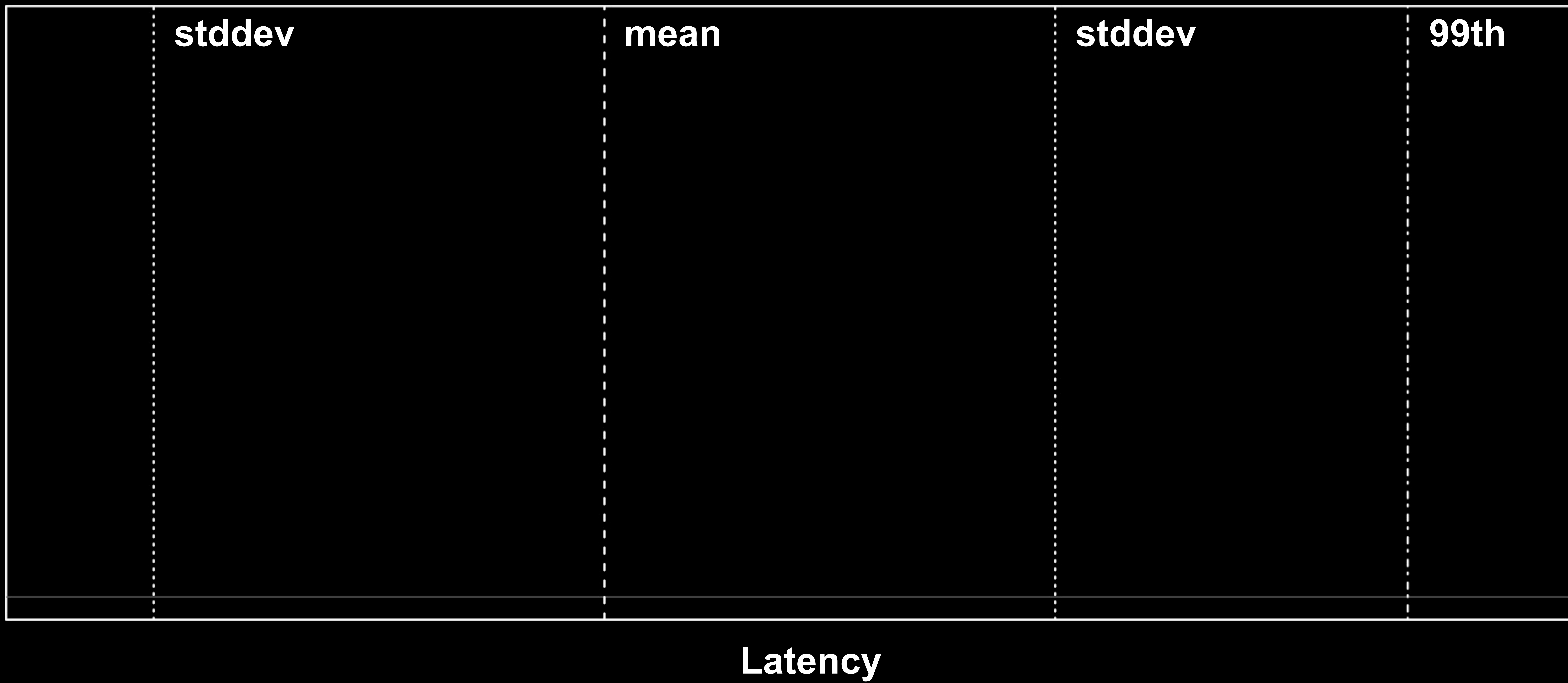
- Pros:
  - Intuitive, attention grabbing
  - Quick (initially)
- Cons:
  - Type I error (red not green): time wasted
  - Type II error (green not red): more time wasted & undiagnosed errors
  - Misleading for subjective metrics: green might not mean what you think it means - depends on tests
  - Over-simplification

Average Anti-Method

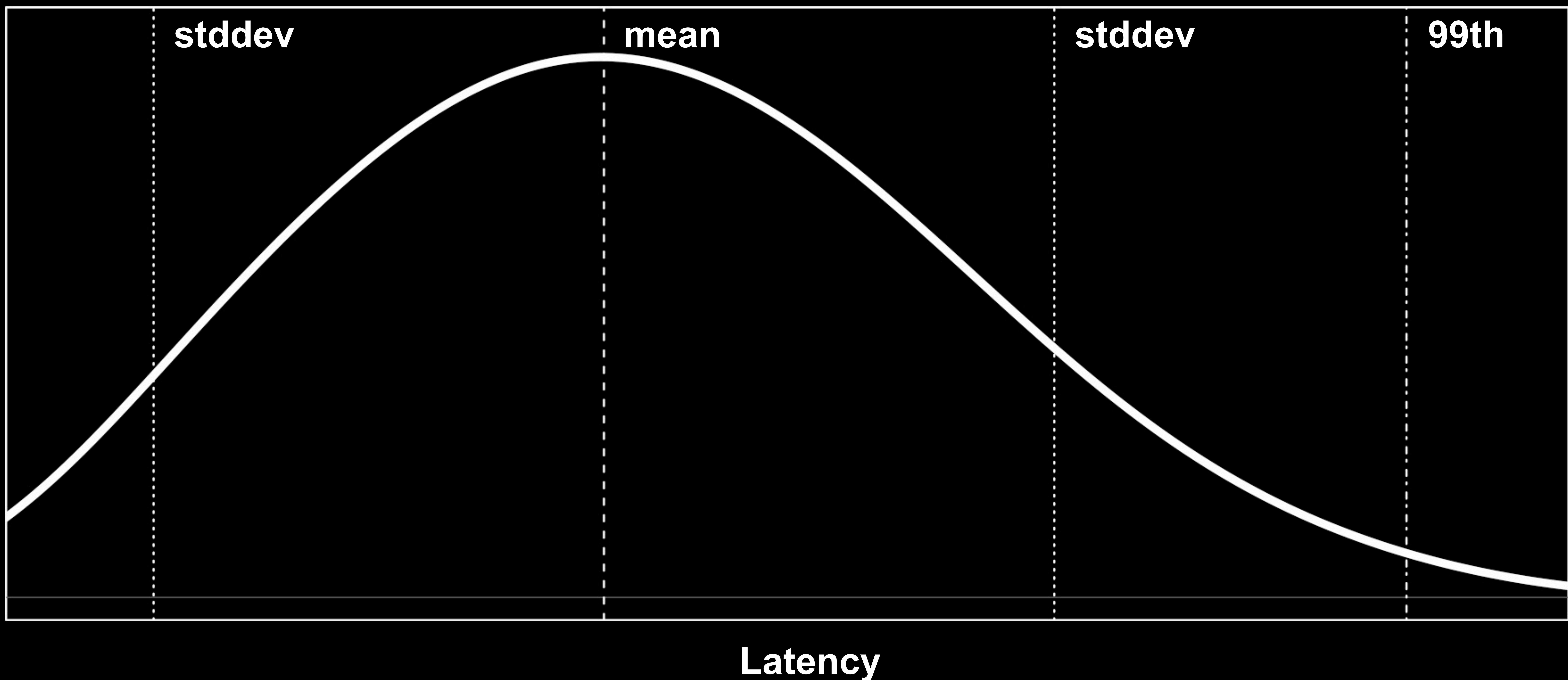
Average Anti-Method

- 1. Measure the average (mean)
- 2. Assume a normal-like distribution (unimodal)
- 3. Focus investigation on explaining the average

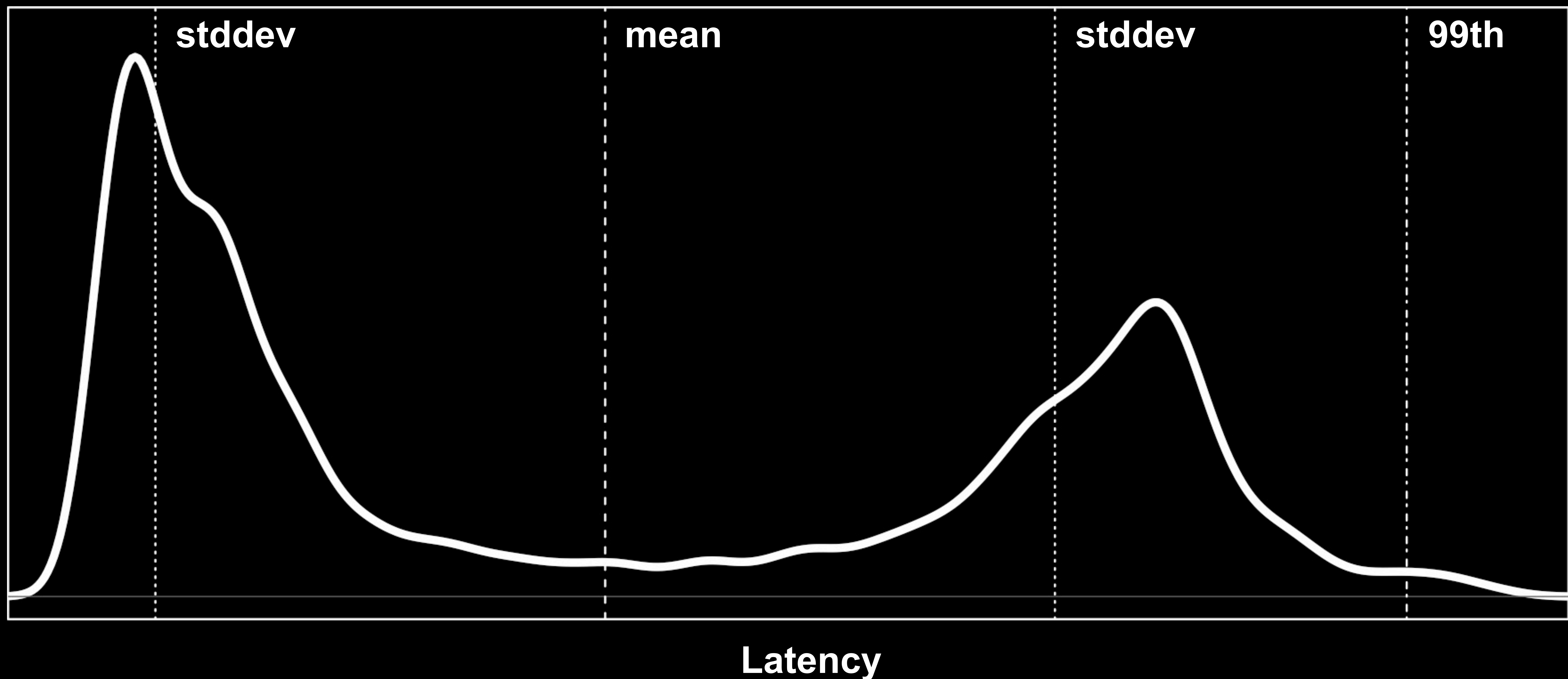
Average Anti-Method: You Have



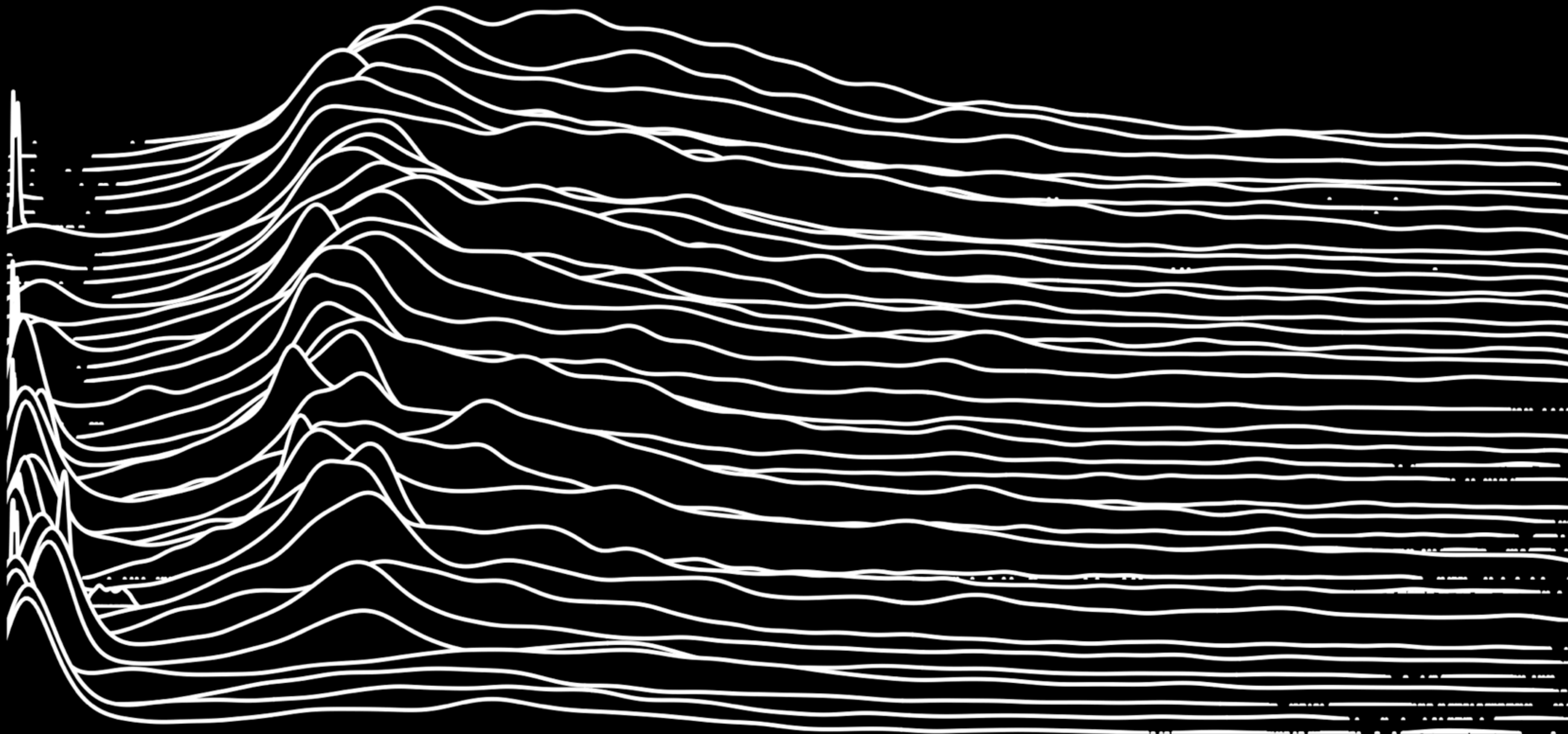
Average Anti-Method: You Guess



Average Anti-Method: Reality



Average Anti-Method: Reality x50

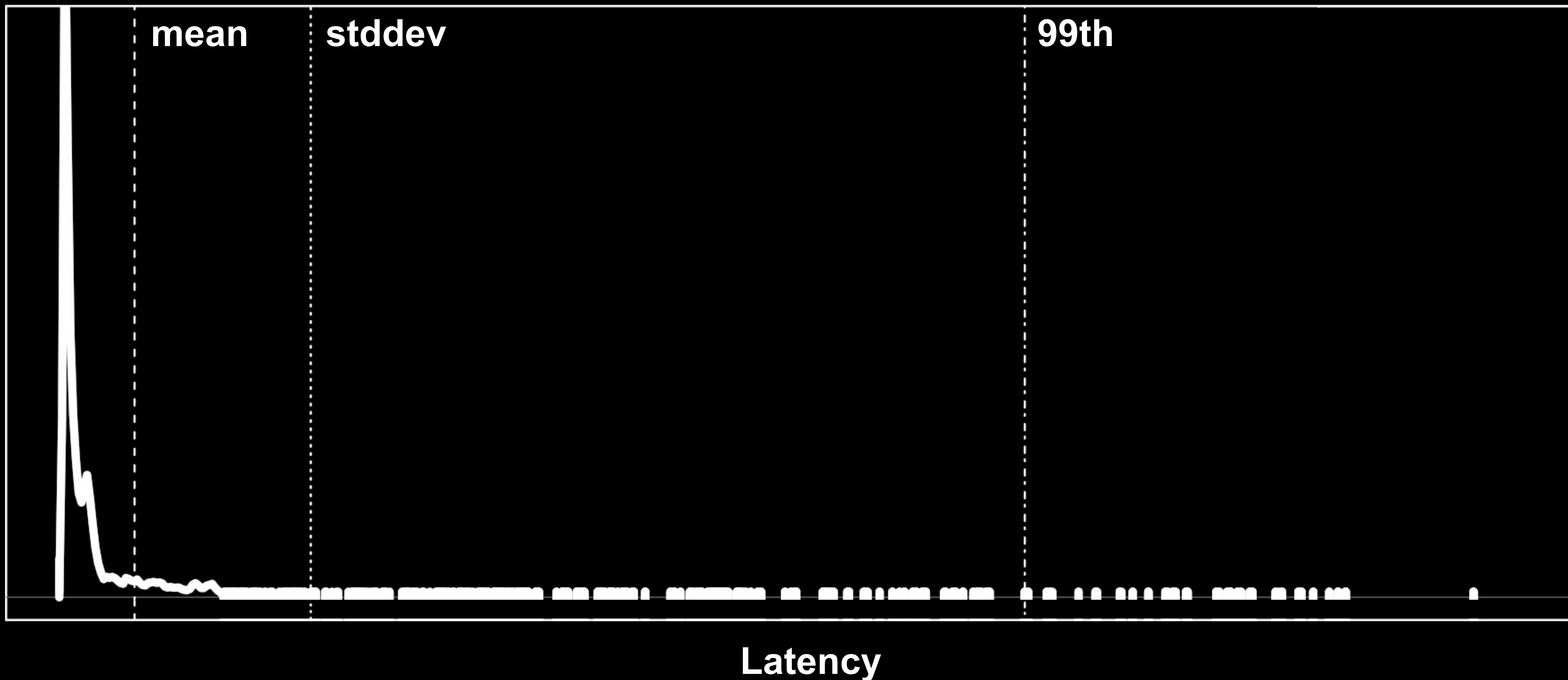


http://dtrace.org/blogs/brendan/2013/06/19/frequency-trails

Average Anti-Method: Examine the Distribution

- Many distributions aren’t normal, gaussian, or unimodal
- Many distributions have outliers
  - seen by the max; may not be visible in the 99...th percentiles
  - influence mean and stddev

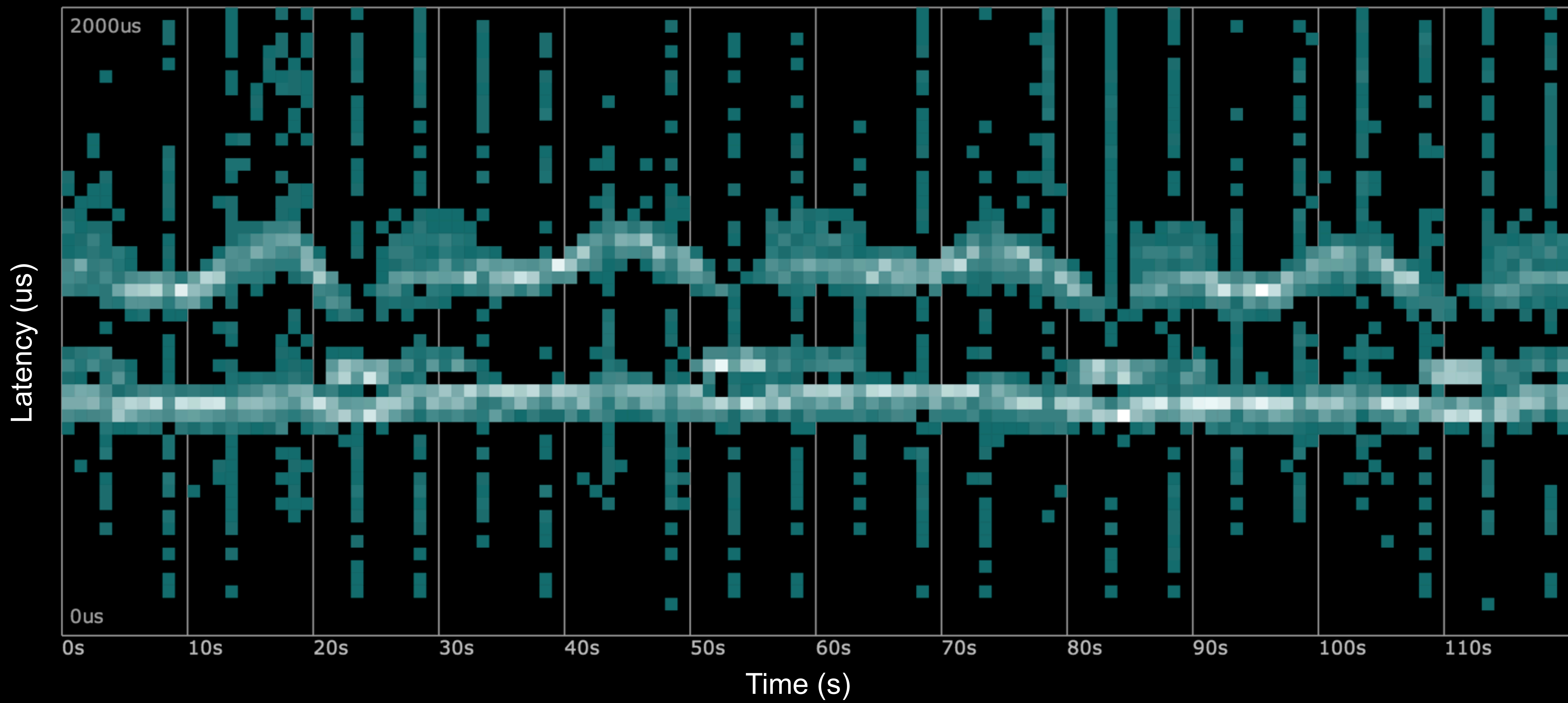
Average Anti-Method: Outliers



Average Anti-Method: Visualizations

- Distribution is best understood by examining it
  - Histogram — summary
  - Density Plot — detailed summary (shown earlier)
  - Frequency Trail — detailed summary, highlights outliers (previous slides)
  - Scatter Plot — show distribution over time
  - Heat Map — show distribution over time, and is scaleable

Average Anti-Method: Heat Map



http://dtrace.org/blogs/brendan/2013/05/19/revealing-hidden-latency-patterns
http://queue.acm.org/detail.cfm?id=1809426

Average Anti-Method

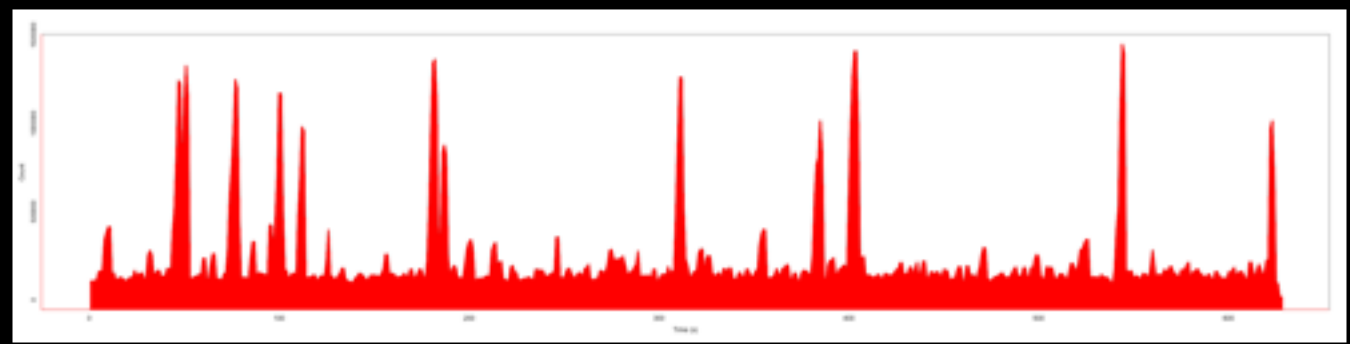
- Pros:
  - Averages are versitile: time series line graphs, Little's Law
- Cons:
  - Misleading for multimodal distributions
  - Misleading when outliers are present
  - Averages are average

Concentration Game Anti-Method

Concentration Game Anti-Method

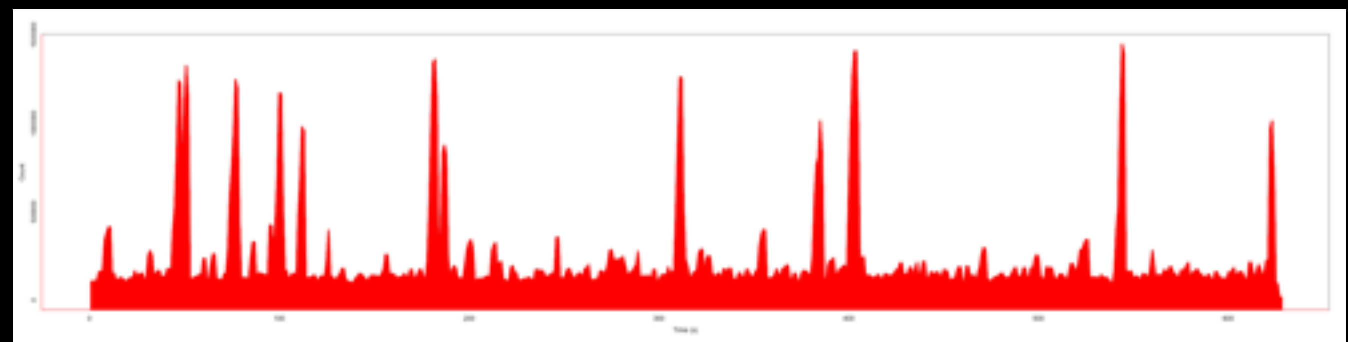
- 1. Pick one metric
- 2. Pick another metric
- 3. Do their time series look the same?
  - If so, investigate correlation!
- 4. Problem not solved? goto 1

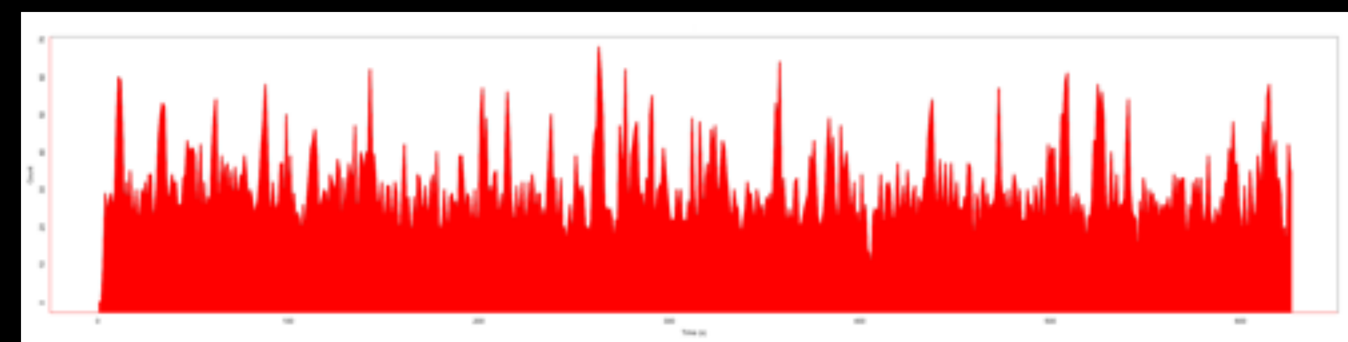
Concentration Game Anti-Method, cont.



App Latency

Concentration Game Anti-Method, cont.

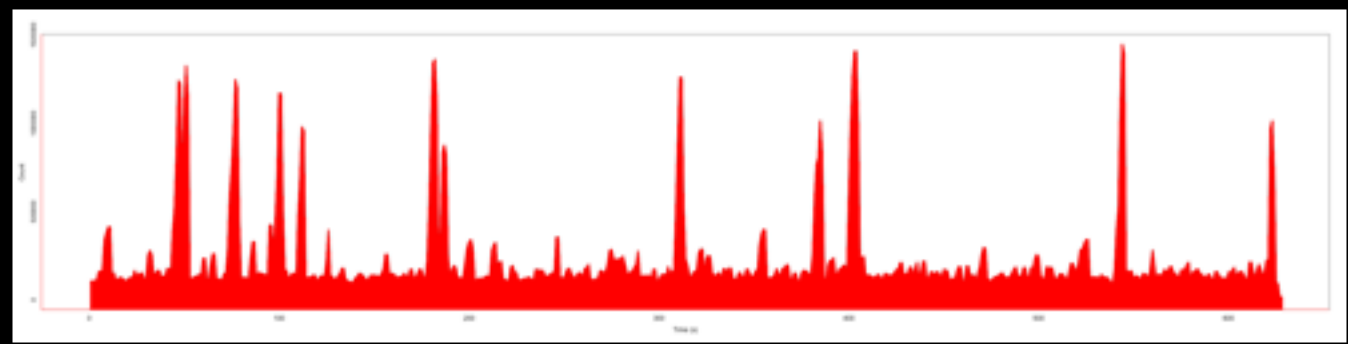




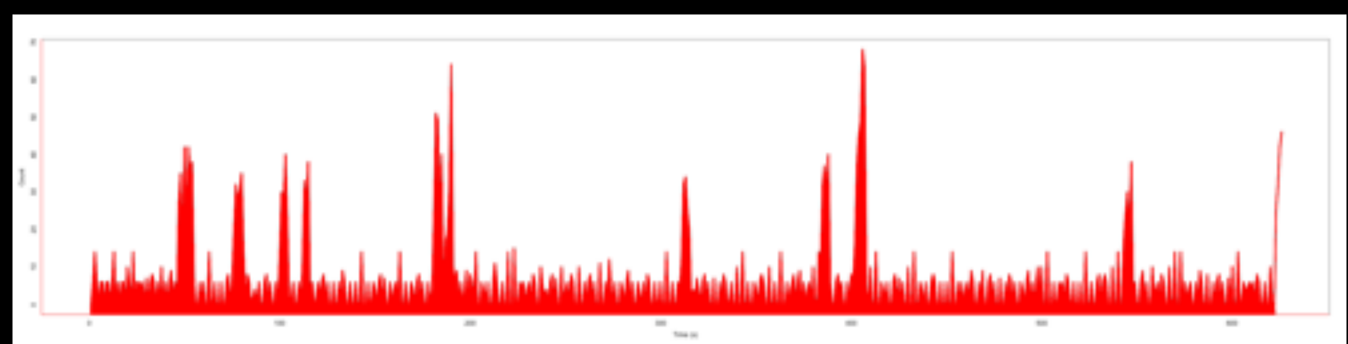
App Latency

NO

Concentration Game Anti-Method, cont.



App Latency



YES!

Concentration Game Anti-Method, cont.

- Pros:
  - Ages 3 and up
  - Can discover important correlations between distant systems
- Cons:
  - Time consuming: can discover many symptoms before the cause
  - Incomplete: missing metrics

Workload Characterization Method

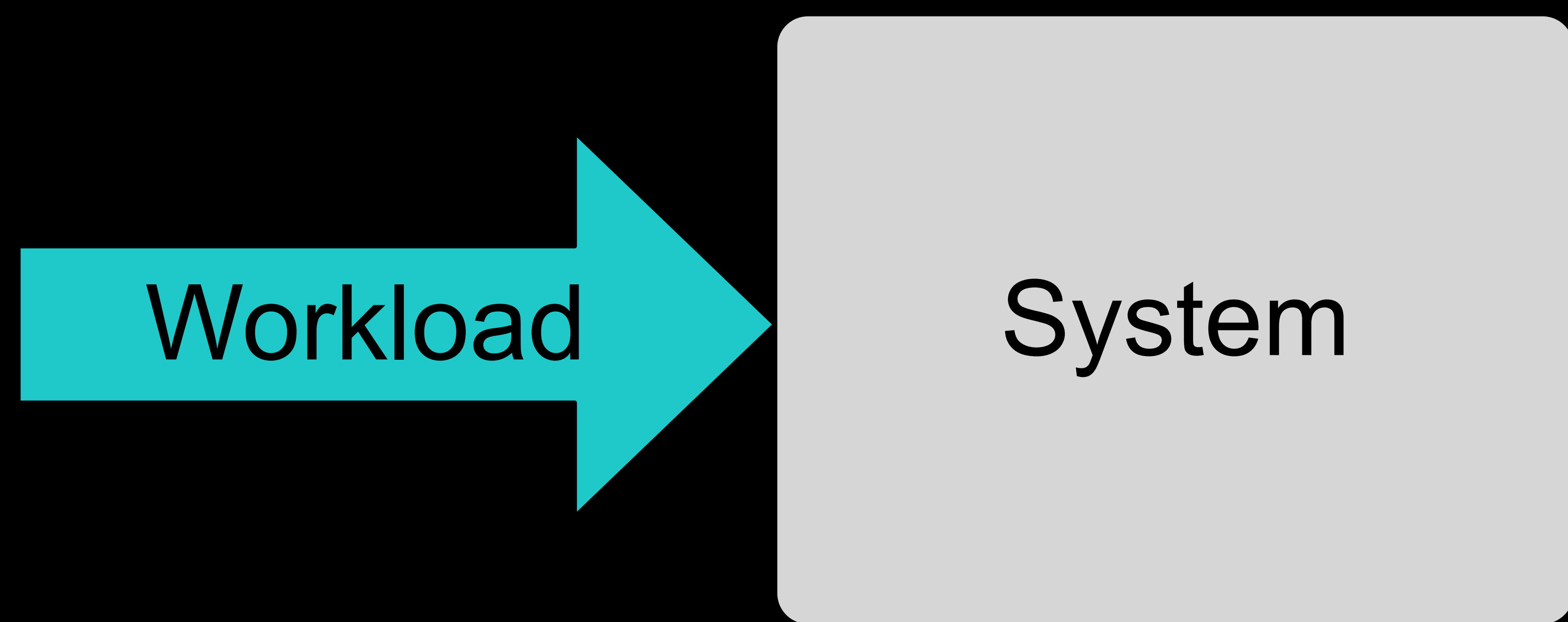
Workload Characterization Method

- 1. Who is causing the load?
- 2. Why is the load called?
- 3. What is the load?
- 4. How is the load changing over time?

Workload Characterization Method, cont.

- 1. Who: PID, user, IP addr, country, browser
- 2. Why: code path, logic
- 3. What: targets, URLs, I/O types, request rate (IOPS)
- 4. How: minute, hour, day

- The target is the system input (the workload) not the resulting performance



Workload Characterization Method, cont.

- Pros:
  - Potentially largest wins: eliminating unnecessary work
- Cons:
  - Only solves a class of issues – load
  - Can be time consuming and discouraging – most attributes examined will not be a problem

USE Method

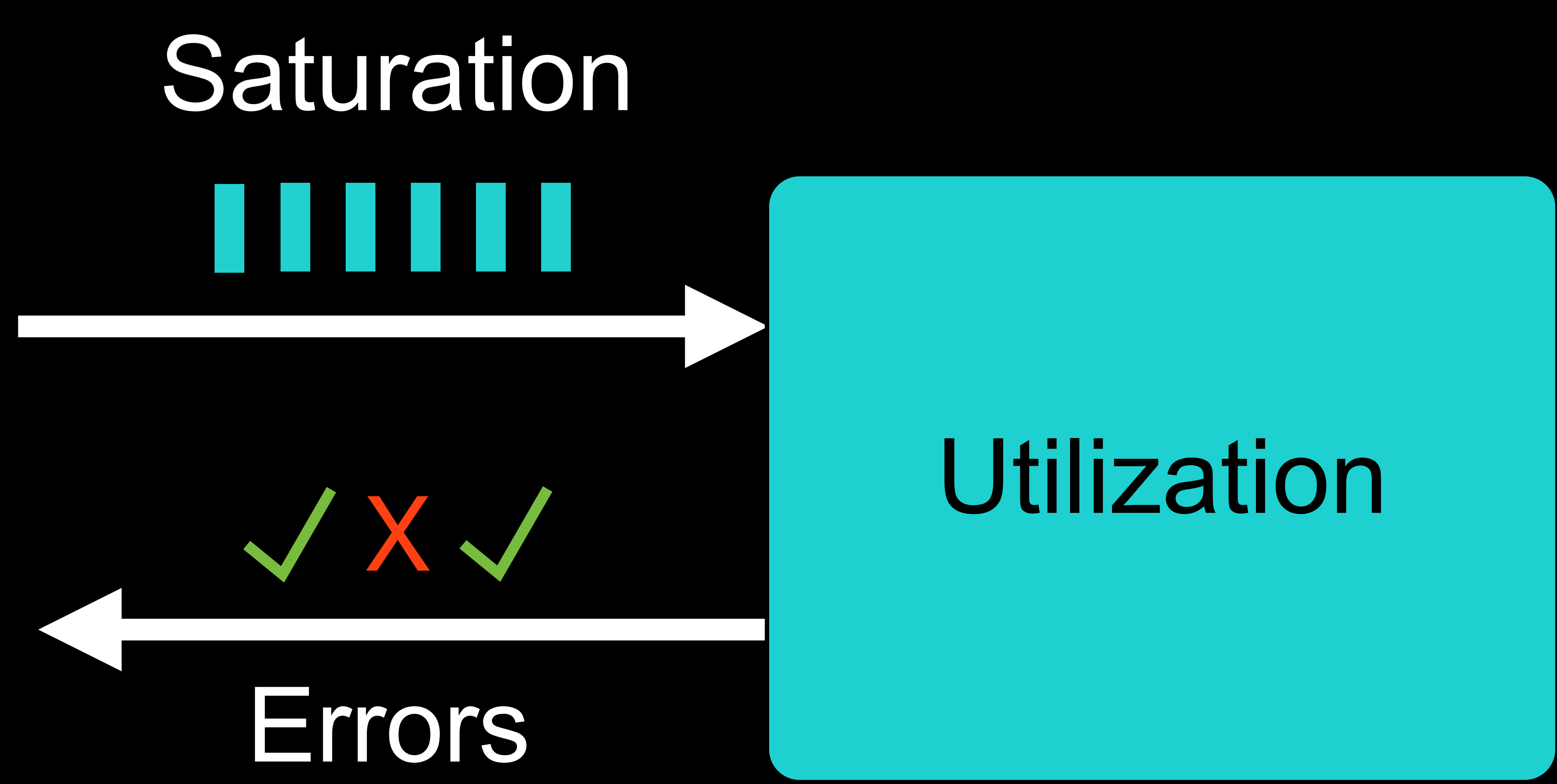
USE Method

- For every resource, check:
- 1. Utilization
- 2. Saturation
- 3. Errors

USE Method, cont.

- For every resource, check:
- 1. Utilization: time resource was busy, or degree used
- 2. Saturation: degree of queued extra work
- 3. Errors: any errors

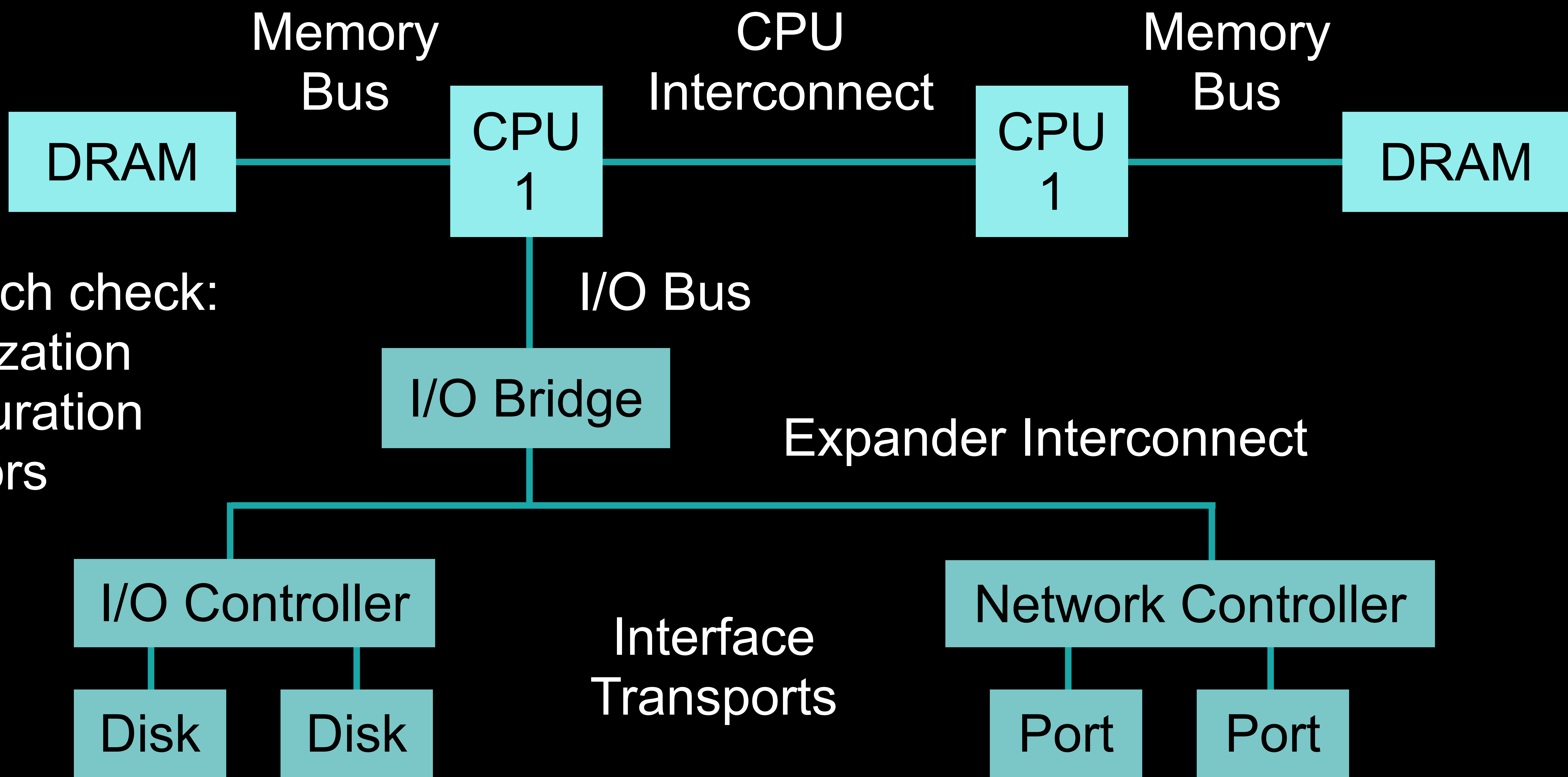
- Identifies resource bottnecks quickly



USE Method, cont.

- Hardware Resources:
  - CPUs
  - Main Memory
  - Network Interfaces
  - Storage Devices
  - Controllers
  - Interconnects
- Find the *functional diagram* and examine every item in the *data path*...

USE Method, cont.: System Functional Diagram



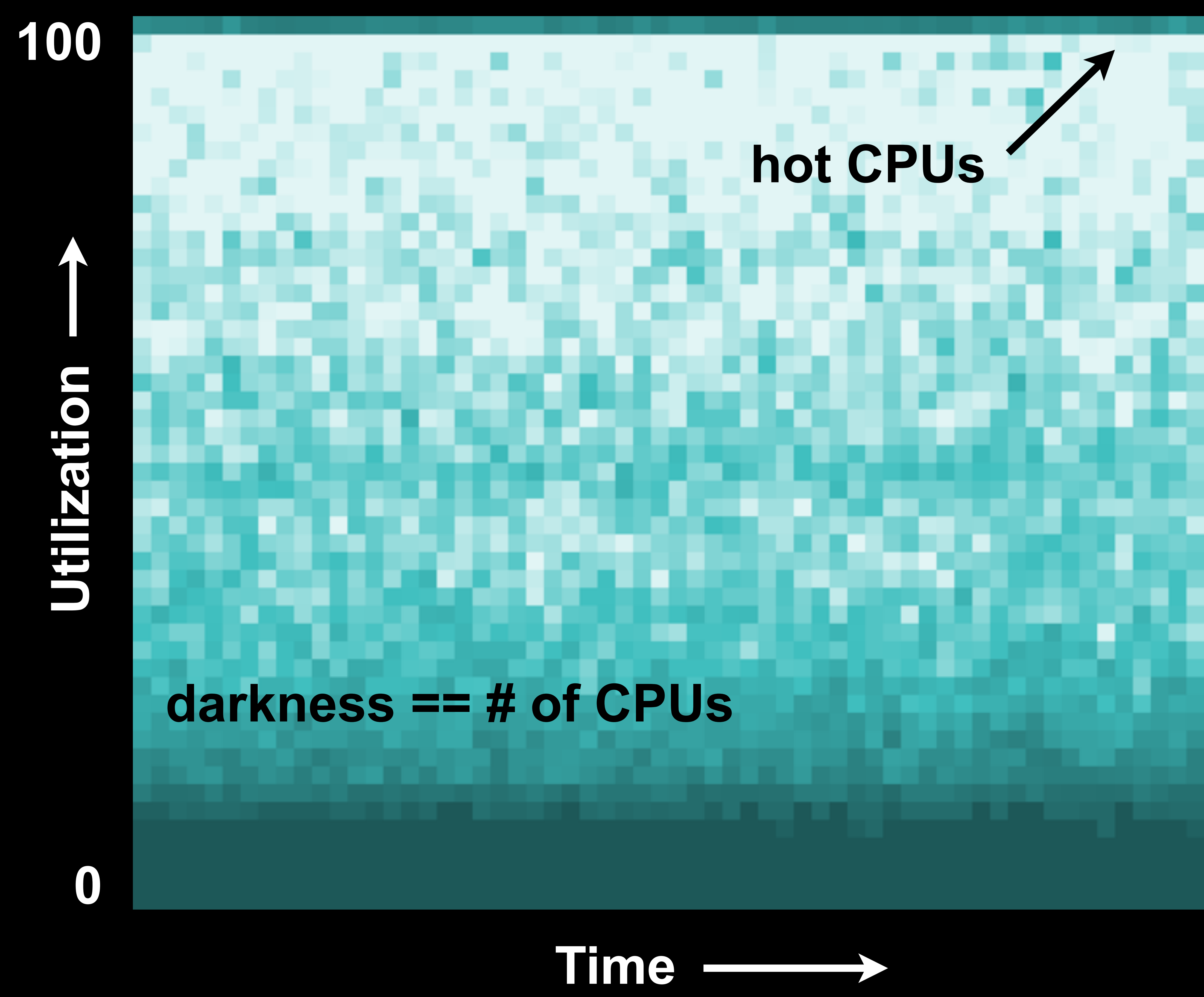
For each check:
1. Utilization
2. Saturation
3. Errors

USE Method, cont.: Linux System Checklist

Resource	Type	Metric
CPU	Utilization	per-cpu: `mpstat -P ALL 1`, "%idle"; `sar -P ALL`, "%idle"; system-wide: `vmstat 1`, "id"; `sar -u`, "%idle"; `dstat -c`, "idl"; per-process:`top`, "%CPU"; `htop`, "CPU%"; `ps -o pcpu`; `pidstat 1`, "%CPU"; per-kernel-thread: `top/htop` ("K" to toggle), where VIRT == 0 (heuristic).
CPU	Saturation	system-wide: `vmstat 1`, "r" > CPU count [2]; `sar -q`, "runq-sz" > CPU count; `dstat -p`, "run" > CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); `perf sched latency` (shows "Average" and "Maximum" delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp "queued(us)"
CPU	Errors	`perf` (LPE) if processor specific error events (CPC) are available; eg, AMD64's "04Ah Single-bit ECC Errors Recorded by Scrubber"
...	...	...

USE Method, cont.: Monitoring Tools

- Average metrics don't work: individual components can become bottlenecks
- Eg, CPU utilization
- Utilization heat map on the right shows 5,312 CPUs for 60 secs; can still identify "hot CPUs"



http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization

USE Method, cont.: Other Targets

- For cloud computing, must study any resource limits as well as physical; eg:
  - physical network interface U.S.E.
  - AND instance network cap U.S.E.
- Other software resources can also be studied with USE metrics:
  - Mutex Locks
  - Thread Pools
- The application environment can also be studied
  - Find or draw a functional diagram
  - Decompose into queueing systems

USE Method, cont.: Homework

- Your ToDo:
  - 1. find a system functional diagram
  - 2. based on it, create a USE checklist on your internal wiki
  - 3. fill out metrics based on your available toolset
  - 4. repeat for your application environment
- You get:
  - A checklist for all staff for quickly finding bottlenecks
  - Awareness of what you cannot measure:
  - *unknown unknowns* become *known unknowns*
  - ... and known unknowns can become feature requests!

USE Method, cont.

- Pros:
  - Complete: all resource bottlenecks and errors
  - Not limited in scope by available metrics
  - No unknown unknowns – at least known unknowns
  - Efficient: picks three metrics for each resource – from what may be hundreds available
- Cons:
  - Limited to a class of issues: resource bottlenecks

Thread State Analysis Method

Thread State Analysis Method

- 1. Divide thread time into operating system states
- 2. Measure states for each application thread
- 3. Investigate largest non-idle state

Thread State Analysis Method, cont.: 2 State

- A minimum of two states:

On-CPU	
Off-CPU	

Thread State Analysis Method, cont.: 2 State

- A minimum of two states:

On-CPU	executing spinning on a lock
Off-CPU	waiting for a turn on-CPU waiting for storage or network I/O waiting for swap ins or page ins blocked on a lock idle waiting for work

- Simple, but off-CPU state ambiguous without further division

Thread State Analysis Method, cont.: 6 State

- Six states, based on Unix process states:

Executing	
Runnable	
Anonymous Paging	
Sleeping	
Lock	
Idle	

Thread State Analysis Method, cont.: 6 State

- Six states, based on Unix process states:

Executing	on-CPU
Runnable	and waiting for a turn on CPU
Anonymous Paging	runnable, but blocked waiting for page ins
Sleeping	waiting for I/O: storage, network, and data/text page ins
Lock	waiting to acquire a synchronization lock
Idle	waiting for work

- Generic: works for all applications

Thread State Analysis Method, cont.

- As with other methodologies, these pose questions to answer
  - Even if they are hard to answer
- Measuring states isn't currently easy, but can be done
  - Linux: /proc, schedstats, delay accounting, I/O accounting, DTrace
  - SmartOS: /proc, microstate accounting, DTrace
- Idle state may be the most difficult: applications use different techniques to wait for work

Thread State Analysis Method, cont.

- States lead to further investigation and actionable items:

Executing	Profile stacks; split into usr/sys; sys = analyze syscalls
Runnable	Examine CPU load for entire system, and caps
Anonymous Paging	Check main memory free, and process memory usage
Sleeping	Identify resource thread is blocked on; syscall analysis
Lock	Lock analysis

Thread State Analysis Method, cont.

- Compare to database query time. This alone can be misleading, including:
  - swap time (anonymous paging) due to a memory misconfig
  - CPU scheduler latency due to another application
- Same for any “time spent in ...” metric
  - is it really *in ...*?

Thread State Analysis Method, cont.

- Pros:
  - Identifies common problem sources, including from other applications
  - Quantifies application effects: compare times numerically
  - Directs further analysis and actions
- Cons:
  - Currently difficult to measure all states

More Methodologies

- Include:
  - Drill Down Analysis
  - Latency Analysis
  - Event Tracing
  - Scientific Method
  - Micro Benchmarking
  - Baseline Statistics
  - Modelling
- For when performance *is* your day job

Stop the Guessing

- The anti-methodolgies involved:
  - guesswork
  - beginning with the tools or metrics (answers)
- The actual methodolgies posed questions, then sought metrics to answer them
- You don't need to guess – post-DTrace, practically everything can be known
- Stop guessing and start asking questions!

Thank You!

- email: brendan@joyent.com
- twitter: @brendangregg
- github: https://github.com/brendangregg
- blog: http://dtrace.org/blogs/brendan
- blog resources:
  - http://dtrace.org/blogs/brendan/2008/11/10/status-dashboard
  - http://dtrace.org/blogs/brendan/2013/06/19/frequency-trails
  - http://dtrace.org/blogs/brendan/2013/05/19/revealing-hidden-latency-patterns
  - http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist
  - http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization