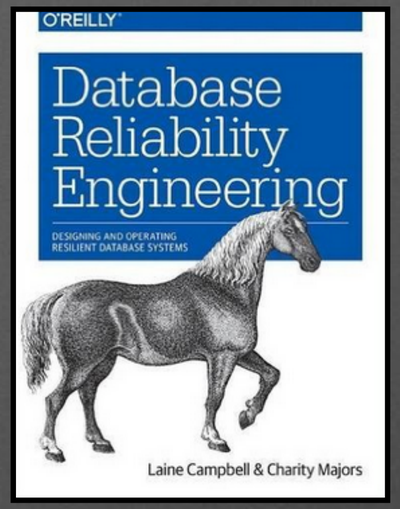
Observability and Emerging Infrastructures

*Not just an ops thing.

honeycomb

honeycomb



@mipsytipsy

engineer/cofounder/CEO

“the only good diff is a red diff”

@mipsytipsy

Hates monitoring

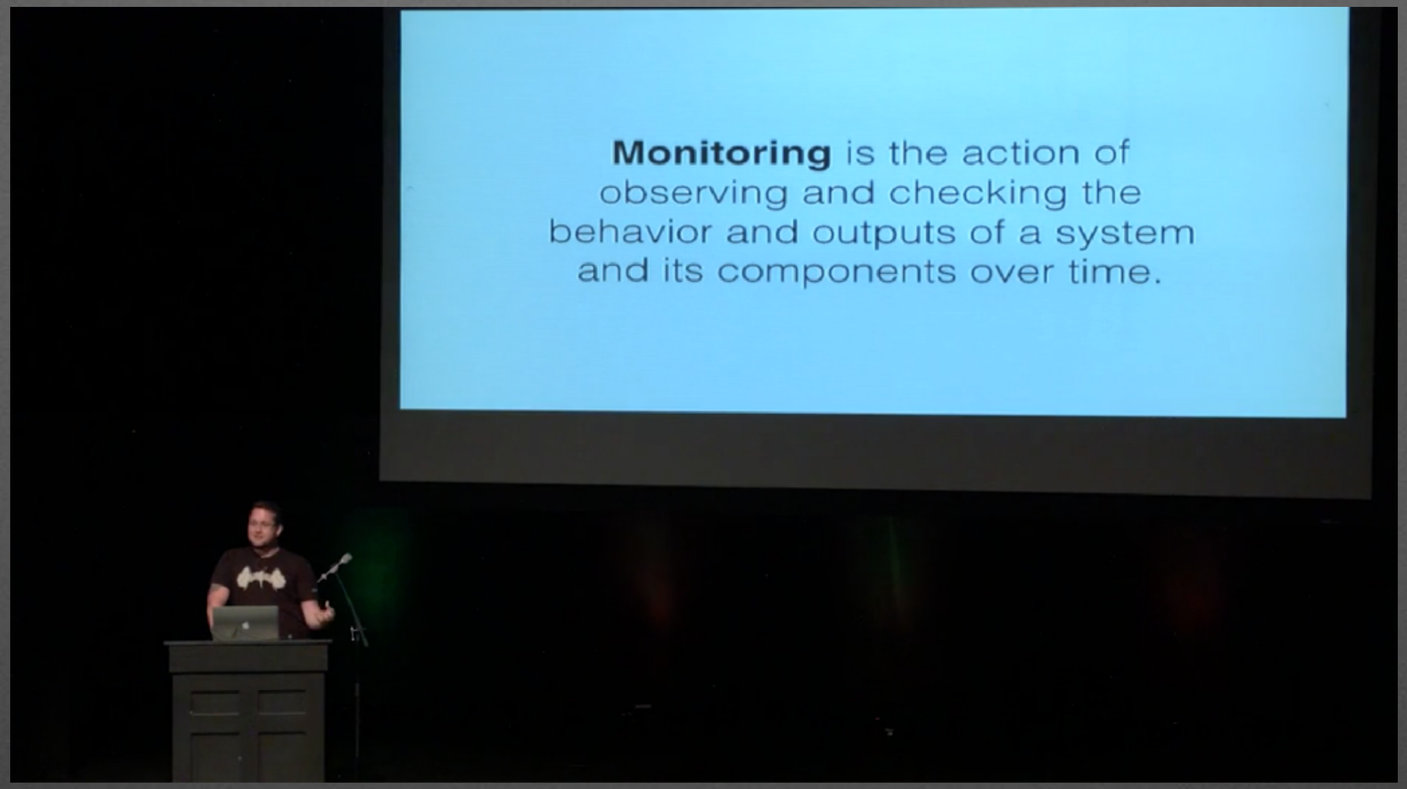
honeycomb

Not a monitoring company

“Monitoring is dead.”

“Monitoring systems have not changed significantly in 20 years and has fallen behind the way we build software. Our software is now large distributed systems made up of many non-uniform interacting components while the core functionality of monitoring systems has stagnated.”

@grepory, Monitorama 2016



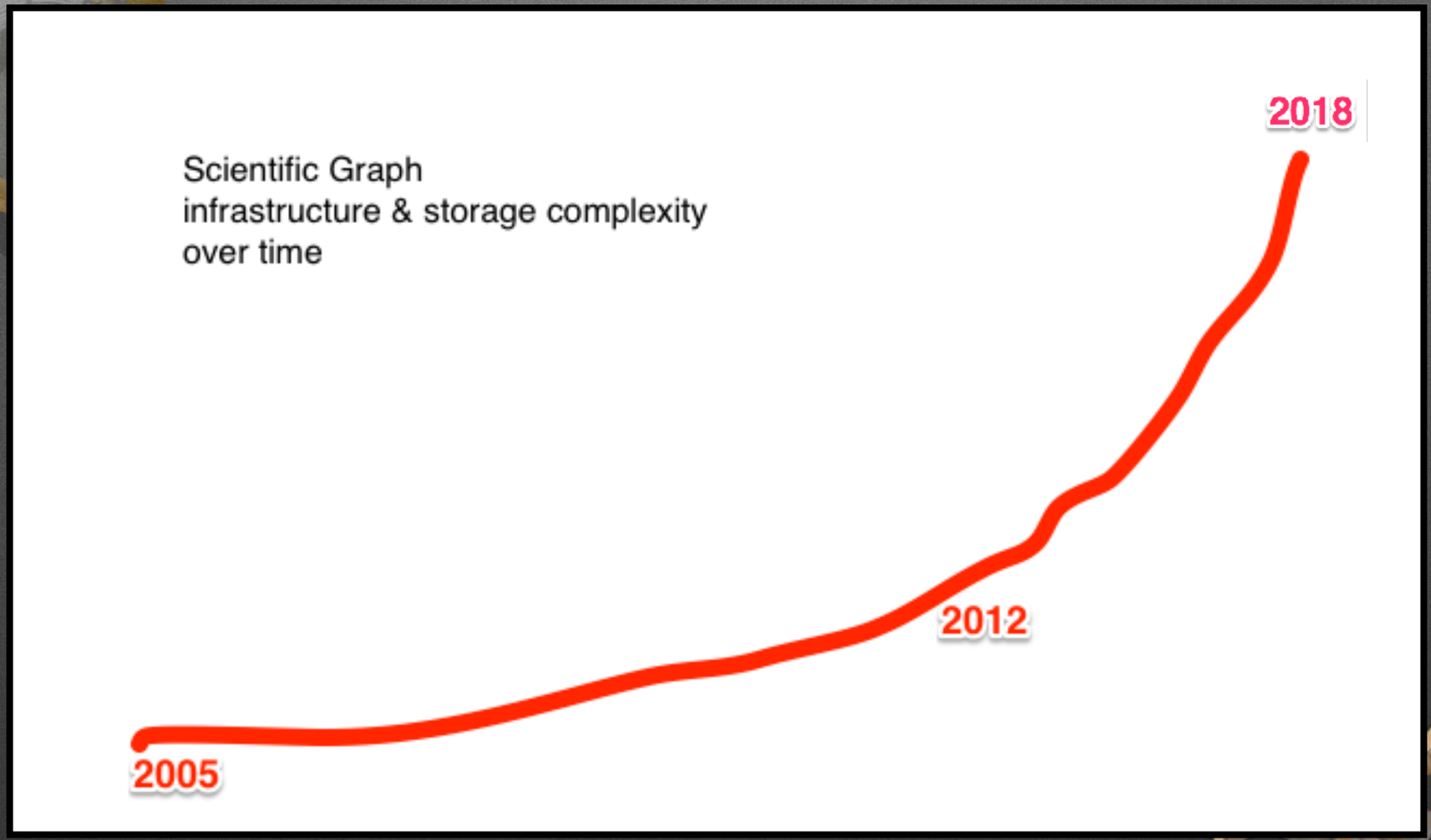
@grepory, 2016

This is a outdated model for complex systems.

We don’t *know* what the questions are, all we have are unreliable symptoms or reports.

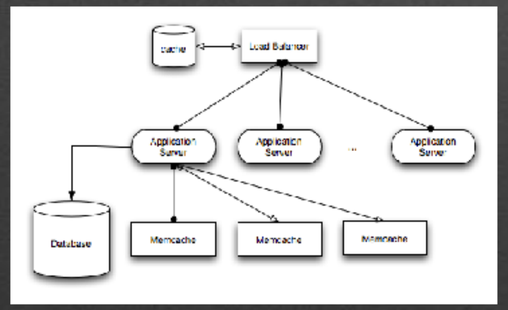
Complexity is exploding everywhere, but our tools are designed for a predictable world.

As soon as we know the question, we usually know the answer too.

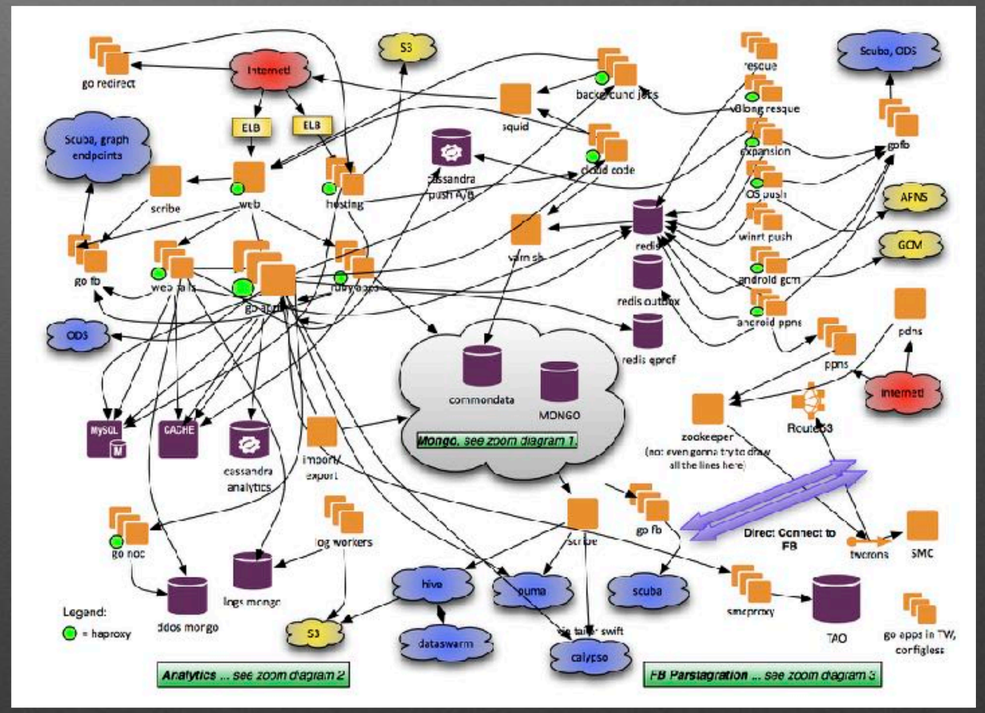


“Complexity is increasing” - Science

Architectural complexity



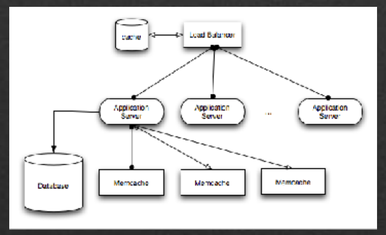
LAMP stack, 2005

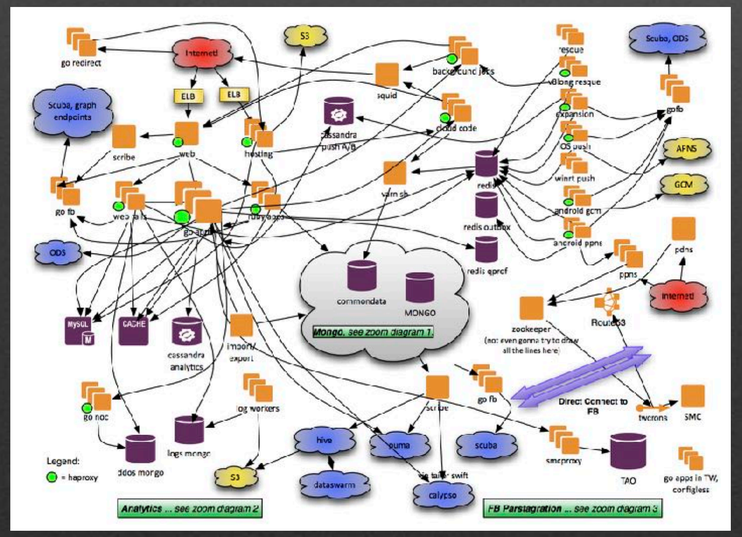


Parse, 2015

monitoring => observability

known unknowns => unknown unknowns

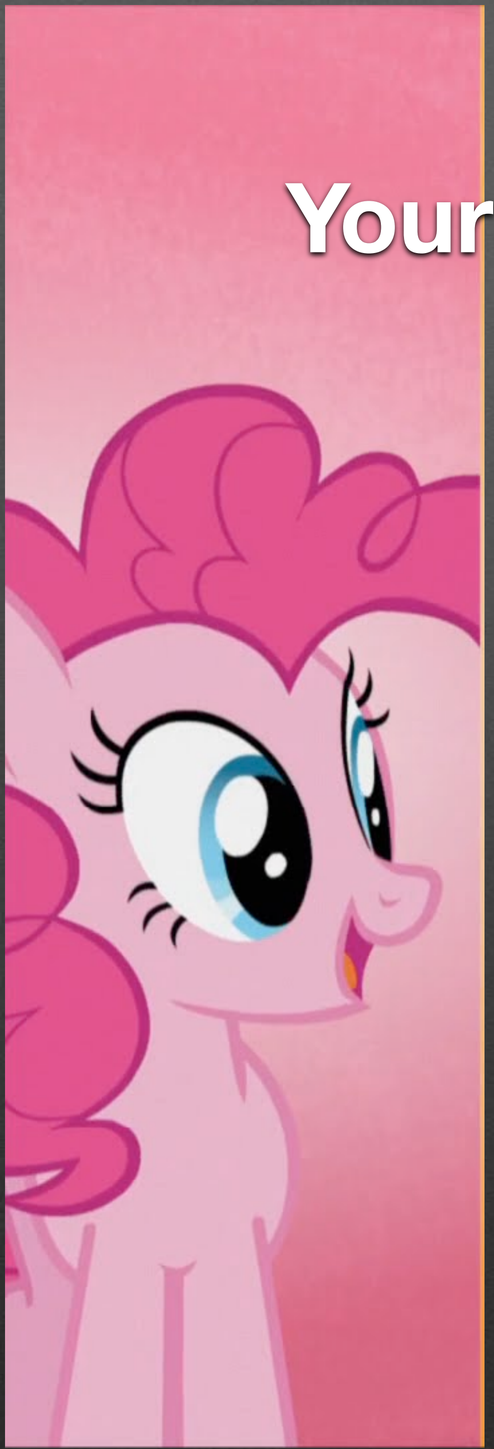




Welcome to distributed systems.

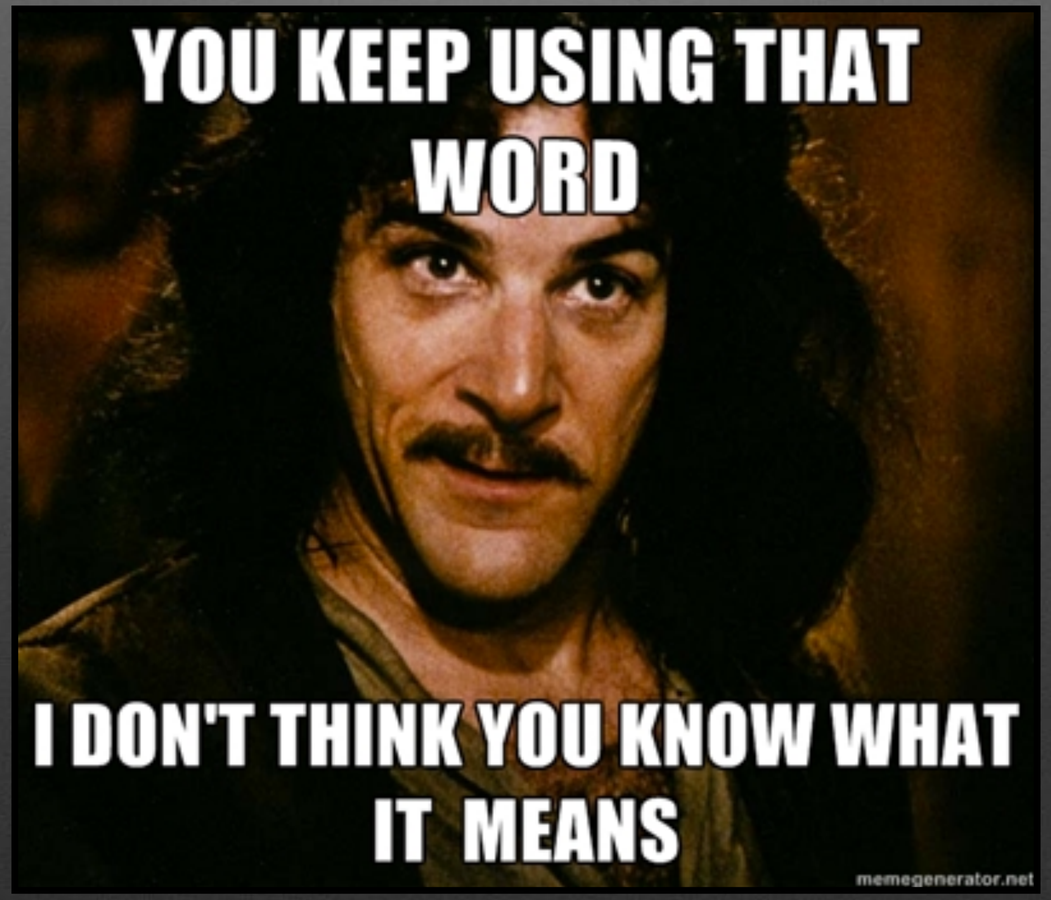
it’s probably fine.

(it might be fine?)



Your system is never entirely ‘up’

Many catastrophic states exist at any given time.



Exponentially more possible outcomes and # of components, ephemeral architecture, flexibility

Unknown-unknowns increasingly dominate

Monitoring

The system experienced as magic. Thresholds, alerts, watching the health of a system by checking for a long list of symptoms. Black box-oriented.

Observability

The world as it really is. What can you learn about the running state of a program by observing its outputs? (Instrumentation, tracing, debugging)

~~Monitoring~~

Observability

Observability

“In control theory, **observability** is a measure of how well internal states of a system can be inferred from knowledge of its external outputs. The observability and controllability of a system are mathematical duals." — wikipedia

… translate??!?

Observability

Can you understand what's happening inside your code and systems, simply by asking questions using your tools? Can you answer any new question you think of, or only the ones you prepared for?

Having to ship new code every time you want to ask a new question … SUCKS.

Observability:Software Engineers::Monitoring:Operations

You have an observable system
when your team can quickly and reliably track down any new problem with no prior knowledge.

Let’s try some examples!

LAMP stack vs distributed system

“Photos are loading slowly for some people. Why?”

(LAMP stack)

The app tier capacity is exceeded. Maybe we rolled out a build with a perf regression, or maybe some app instances are down.

monitor these things

Errors or latency are high. We will look at several dashboards that reflect common root causes, and one of them will show us why.

DB queries are slower than normal. Maybe we deployed a bad new query, or there is lock contention.

Monitoring

Characteristics

(LAMP stack)

- Known-unknowns predominate
- Intuition-friendly
- Dashboards are valuable.
- Monolithic app, single data source.
- The health of the system more or less accurately represents the experience of the individual users.

Monitoring

Best Practices

- Lots of actionable active checks and alerts
- Proactively notify engineers of failures and warnings
- Maintain a runbook for stable production systems
- Rely on clusters and clumps of tightly coupled systems all breaking at once

Monitoring

“Photos are loading slowly for some people. Why?”

(microservices)

Any microservices running on c2.4xlarge instances and PIOPS storage in us-east-1b has a 1/20 chance of running on degraded hardware, and will take 20x longer to complete for requests that hit the disk with a blocking call. This disproportionately impacts people looking at older archives due to our fanout model.

wtf do i ‘monitor’ for?!

Canadian users who are using the French language pack on the iPad running iOS 9, are hitting a firmware condition which makes it fail saving to local cache … which is why it FEELS like photos are loading slowly

Our newest SDK makes db queries sequentially if the developer has enabled an optional feature flag. Working as intended; the reporters all had debug mode enabled. But flag should be renamed for clarity sake.

Monitoring?!?

~~Problems~~ Symptoms
(microservices)

"I have twenty microservices and a sharded db and three other data stores across three regions, and everything seems to be getting a little bit slower over the past two weeks but nothing has changed that we know of, and oddly, latency is usually back to the historical norm on Tuesdays.

"All twenty app micro services have 10% of available nodes enter a simultaneous crash loop cycle, about five times a day, at unpredictable intervals. They have nothing in common afaik and it doesn't seem to impact the stateful services. It clears up before we can debug it, every time."

"Our users can compose their own queries that we execute server-side, and we don't surface it to them when they are accidentally doing full table scans or even multiple full table scans, so they blame us."

Observability

Still More Symptoms

(microservices)

"Several users in Romania and Eastern Europe are complaining that all push notifications have been down for them … for days."

"Sometimes a bot takes off, or an app is featured on the iTunes store, and it takes us a long long time to track down which app or user is generating disproportionate pressure on shared components of our system (esp databases). It's different every time."

"Disney is complaining that once in a while, but not always, they don't see the photo they expected to see — they see someone else's photo! When they refresh, it's fixed. Actually, we've had a few other people report this too, we just didn't believe them."

"We run a platform, and it's hard to programmatically distinguish between problems that users are inflicting themselves and problems in our own code, since they all manifest as the same errors or timeouts."

Observability

These are all unknown-unknowns

that may have never happened before, or ever happen again

(They are also the overwhelming majority of what you have to care about for the rest of your life.)

Characteristics

(microservices/complex systems)

- Unknown-unknowns are most of the problems
- “Many” components and storage systems
- You cannot model the entire system in your head. Dashboards may be actively misleading.
- The hardest problem is often identifying which component(s) to debug or trace.
- The health of the system is irrelevant. The health of each individual request is of supreme consequence.

Observability

Best Practices

(microservices/complex systems)

- Rich instrumentation.
- Events, not metrics.
- Sampling, not write-time aggregation.
- Few (if any) dashboards.
- Test in production.. a lot.
- Very few paging alerts.

Observability

Known-unknowns

- A support problem
- Predictable time scale
- Use a fucking dashboard, then automate it out of existence

Unknown-unknowns

- An engineering problem
- Open-ended time scale
- Require creativity

Why:

Instrumentation?
Events, not metrics?
No dashboards?
Sampling, not time series aggregation?
Test in production?
Fewer alerts?

7 commandments for a Glorious Future™

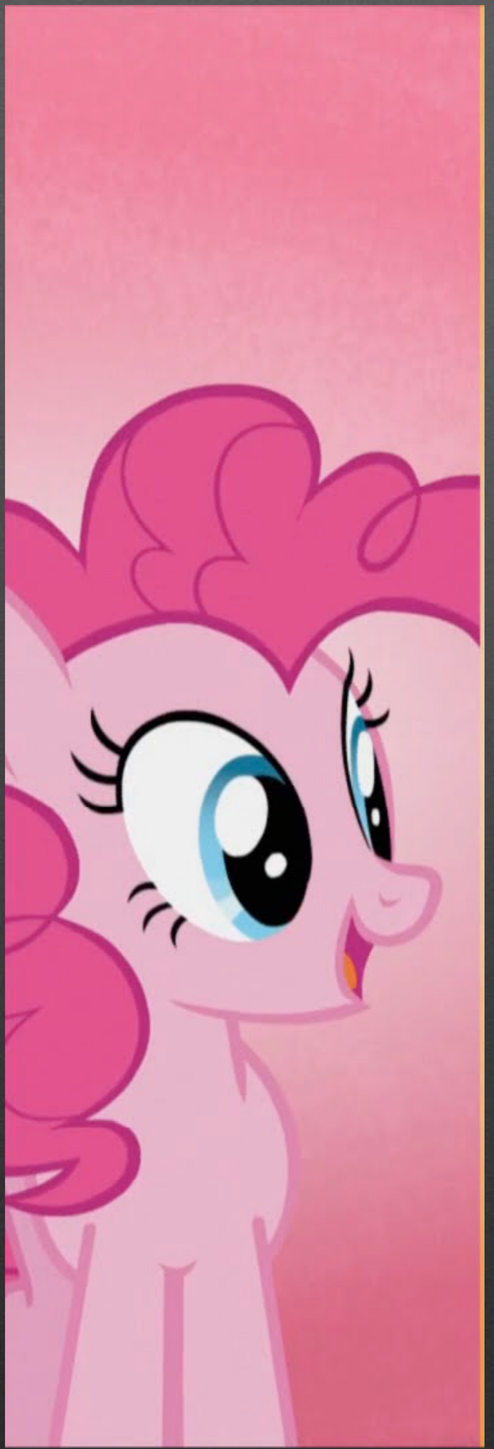
well-instrumented
high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
tested in prod.

7 commandments for a Glorious Future™

well-instrumented
high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
tested in prod.

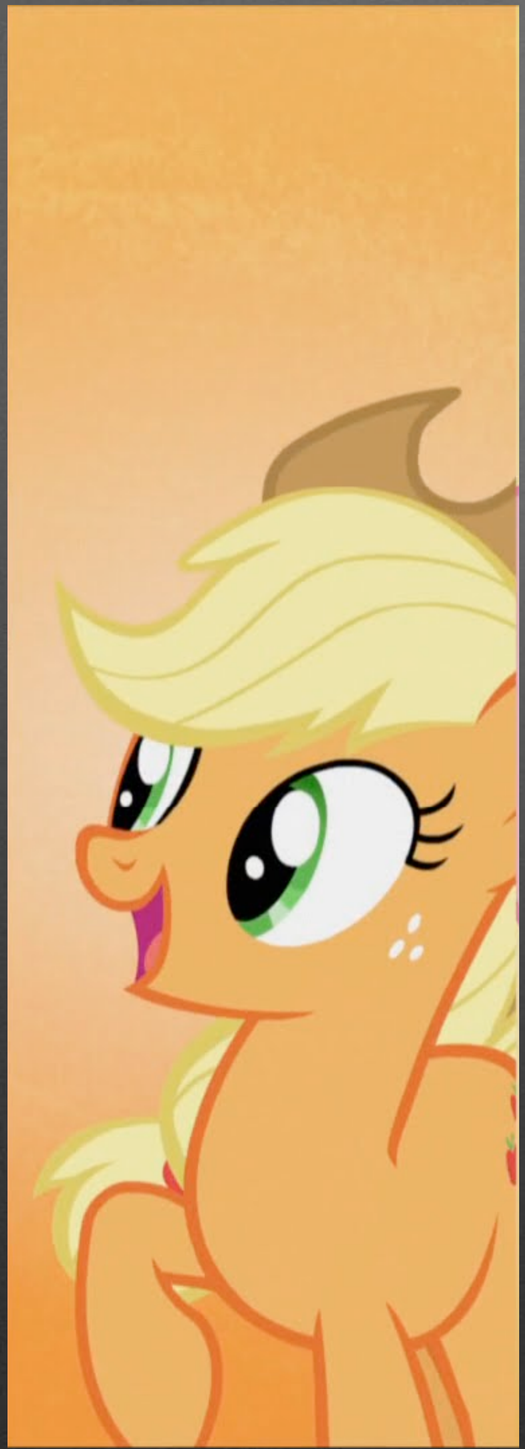
Glorious Future™

well-instrumented
high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
tested in prod.



Instrumentation?

Start at the edge and work down
Internal state from software you didn’t write, too
Wrap every network call, every data call
Structured data only
`gem install` magic will only get you so far



Events, not metrics?

Cardinality
Context
Structured data

(trick question.. you'll need both
but you'll rely on events more and more)

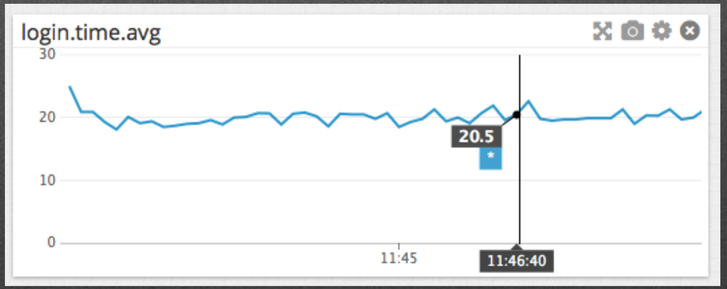
Metrics (+tags)

```
<bucket>:<value>|<type>|@<sample rate>
```

```
login.time:22|ms   # record a login.time event that took 22 ms
```

```
import statsd
statsd_client = statsd.StatsClient('localhost', 8125)

@statsd_client.timer('login.time')
def login(username, password):
    statsd_client.incr('login.invocations')
    if password_valid(username, password):
        render_welcome_page()
```



Metrics are cheap, but terribly limited in context or cardinality.

Metrics (+tags)

```
http.get.200.count   [2][1][4][5][6]
http.get.302.count   [ ][ ][3][1][ ]
http.get.404.count   [ ][1][2][ ][2]
http.post.201.count  [9][7][8][6][9]
http.post.403.count  [ ][1][3][1][ ]
http.post.500.count  [ ][ ][9][1][ ]
```

```
http.get.200.user8996.count  [2][1][ ][ ][ ]
http.get.200.user9b93.count  [ ][ ][3][2][3]
http.get.200.userca49.count  [ ][ ][1][ ][ ]
http.get.200.user2b85.count  [ ][ ][ ][3][3]
http.get.302.user8996.count  [ ][ ][ ][1][ ]
http.get.302.user9b93.count  [ ][ ][1][ ][ ]
http.get.302.userca49.count  [ ][ ][1][ ][ ]
http.get.302.user2b85.count  [ ][ ][1][ ][ ]
http.get.404.user8996.count  [ ][1][ ][ ][ ]
http.get.404.user9b93.count  [ ][ ][2][ ][ ]
http.get.404.userca49.count  [ ][ ][ ][ ][1]
http.get.404.user2b85.count  [ ][ ][ ][ ][1]
http.post.201.user8996.count [1][2][1][ ][3]
http.post.201.user9b93.count [1][2][ ][2][4]
http.post.201.userca49.count [4][1][5][1][1]
http.post.201.user2b85.count [3][2][2][3][1]
http.post.403.user8996.count [ ][ ][1][ ][ ]
http.post.403.user9b93.count [ ][1][ ][ ][ ]
http.post.403.userca49.count [ ][ ][1][1][ ]
http.post.403.user2b85.count [ ][ ][1][ ][ ]
http.post.500.user8996.count [ ][ ][ ][1][ ]
http.post.500.user9b93.count [ ][ ][1][ ][ ]
http.post.500.userca49.count [ ][ ][7][ ][ ]
http.post.500.user2b85.count [ ][ ][1][ ][ ]
```

```
statsd.increment(`http.${method}.${status_code}.count`)
```

Metrics are cheap, but terribly limited in context or cardinality.

Metrics (cardinality)

Some of these …
might be …
useful …
YA THINK??!

High cardinality will save your ass.

UUIDs
db raw queries
normalized queries
comments
firstname, lastname
PID/PPID
app ID
device ID
HTTP header type
build ID
IP:port
shopping cart ID
userid
... etc

High cardinality is not a nice-to-have

You must be able to break down by 1/millions and
THEN by anything/everything else

‘Platform problems’ are now everybody’s problems

Looking for a needle in your haystack? Be descriptive, add unique identifiers.

Read-time aggregation lets you compute percentiles, derived columns.


```
[{"Timestamp":"Sat Sep  2 00:30:56 UTC 2017"
"api_version":"1"
"availability_zone":"us-east-1b"
"batch":false
"build_id":"4715"
"chosen_partition":31
"content_length":156785
"dataset_columns":35
"dataset_expand_json_depth":0
"dataset_id":3915
"dataset_name":"kubernetes-resource-metrics"
"dataset_partitions":"[3,31,35]"
"dynsample_key":"batch,3915,784,202"
"dynsample_rate":1
"env":"production"
"event_columns":23
"event_time":"0001-01-01T00:00:00Z"
"extra_headers":"Accept-Encoding:gzip,Connection:keep-alive,Content-Length:1
nt-Type:application/json,X-Forwarded-Port:443,X-Forwarded-Proto:https"
"instance_type":"c4.large"
"json_decoding_dur_ms":278.703396
"memory_inuse":1.1720848
"method":"POST"
"nested_json":false
"nested_json_depth":0
"num_goroutines":270
"oversize_len_longest_string_column":90
"oversize_num_columns":0
"oversize_total_bytes":0
"prep_partition_info_dur_ms":0.004373
"process_uptime_seconds":2545
"remote_addr":"10.0.54.83:2158"
```


Structured Data

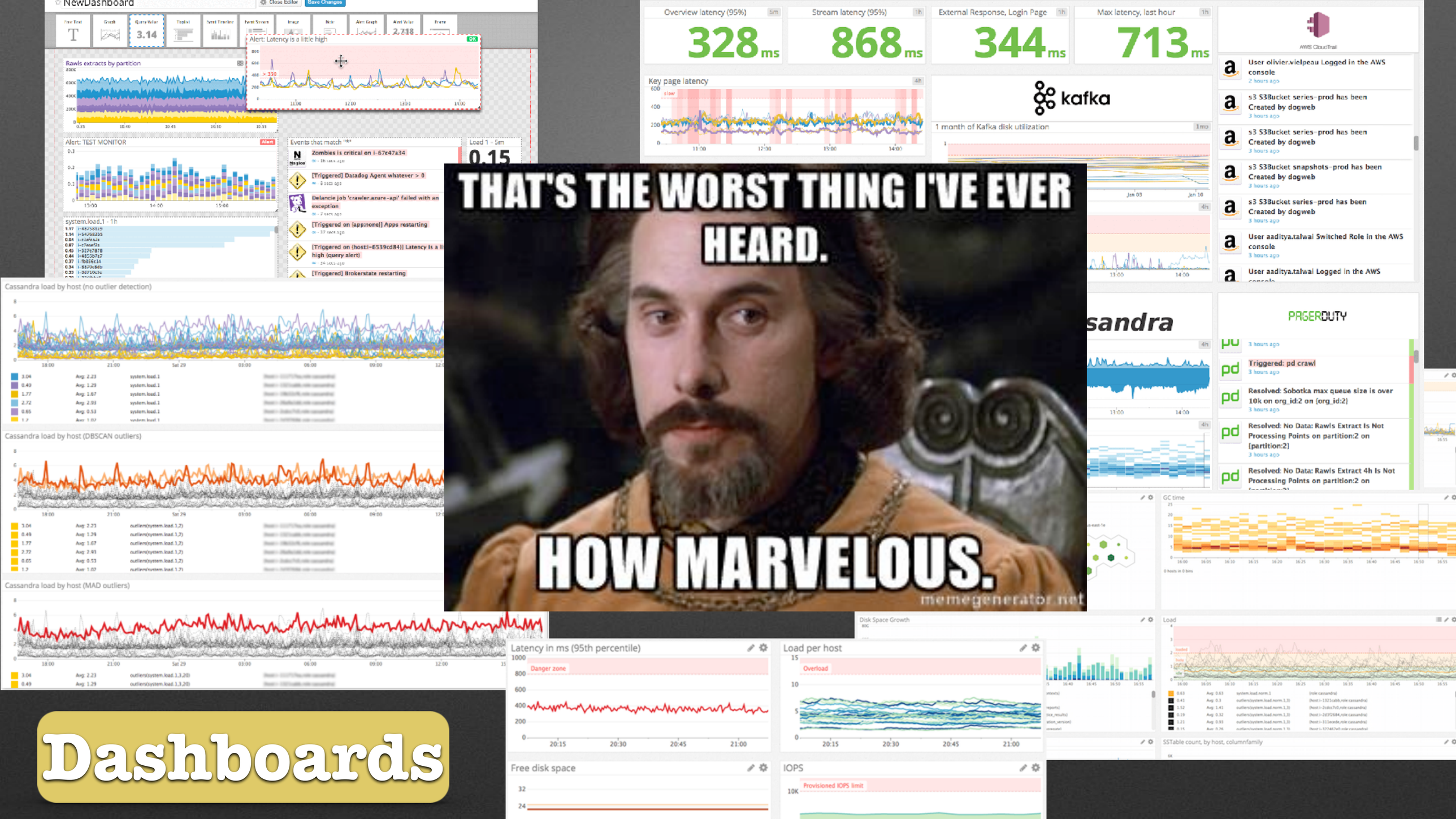
Events tell stories.

Arbitrarily wide events mean you can amass more and more context over time. Use sampling to control costs and bandwidth.

Structure your data at the source to reap massive efficiencies over strings.

("Logs" are just a transport mechanism for events)

Dashboards??


NewDashboard
Overview latency (95%)
328 ms
Stream latency (95%)
868 ms
External Response, Login Page
344 ms
Max latency, last hour
713 ms
Key page latency
kafka
1 month of Kafka disk utilization
Alert: Latency is a little high
Rawls extracts by partition
Alert: TEST MONITOR
Events that match "*"
Load 1 - 5m
0.15
Cassandra load by host (no outlier detection)
Cassandra load by host (DBSCAN outliers)
Cassandra load by host (MAD outliers)
PAGERDUTY
Triggered: pd crawl
Resolved: Sobotka max queue size is over 10k on org_id:2 on {org_id:2}
Resolved: No Data: Rawls Extract Is Not Processing Points on partition:2 on {partition:2}
Resolved: No Data: Rawls Extract 4h Is Not Processing Points on partition:2 on
THAT'S THE WORST THING I'VE EVER HEARD.
HOW MARVELOUS.
memegenerator.net
Latency in ms (95th percentile)
Danger zone
Load per host
Overload
Disk Space Growth
Load
Free disk space
IOPS
Provisioned IOPS limit
SSTable count, by host, columnfamily

Dashboards

Dashboards

Artifacts of past failures.

Jumps to an answer, instead of starting with a question

You don’t know what you don’t know.

Exploratory

Interactive

Iterative

Fast

Raw

Dashboard overuse must die

Unknown-unknowns demand explorability and an open mind.

Raw requests:

sampling, not aggregation

Aggregation is a one-way trip

Destroying raw events eliminates your ability to ask new questions. Forever.

Aggregates are the devil

Aggregates destroy your precious details.

You need MORE detail and MORE context.

Aggregates

You can’t hunt needles if your tools don’t handle extreme outliers, aggregation by arbitrary values in a high-cardinality dimension, super-wide rich context…

Black swans are the norm

you must care about max/min, 99%, 99.9th, 99.99th, 99.999th …

“Sum up all the time spent holding the user.* table lock by INSERT queries, broken down by user id and the size of the object written, and show me any users using more than 30% of the overall row lock.”

“Latency seems elevated for HTTP requests. Requests can loop recursively back into the API multiple times; are requests getting progressively slower as the iteration stack gets deeper? What is the MAX recursive call depth, and max latency over the past day? Is it still growing? What do the 100 slowest have in common?”

“Show me all the 50x errors broken down by user id or app id. Show me all the abandoned carts with the most items in them. Show me the users rate limited in the past hour, broken down by browser type or mobile device type and release version string.”

Raw data examples

Zero users care what the “system” health is

All users care about THEIR experience.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Raw Requests

atter if users aren’t happy.

Nines don’t matter if users aren’t happy.

Test in production

SWEs own their own services

Services need owners, not operators.

Observability:

must be designed for generalist SWEs.

SaaS, APIs, SDKs.

Ops lives on the other side of an API

Engineers

Operations skills are not optional for software engineers in 2016. They are not “nice-to-have”,

they are table stakes.

Engineers

~~Monitoring~~ **Instrumentation** is part of building software

Engineers

Software engineers spend too much time looking at code in elaborately falsified environments, and not enough time observing it in the real world.

- Real users
- Real data
- Real infra
- Other real services

~~Test-Driven Development~~

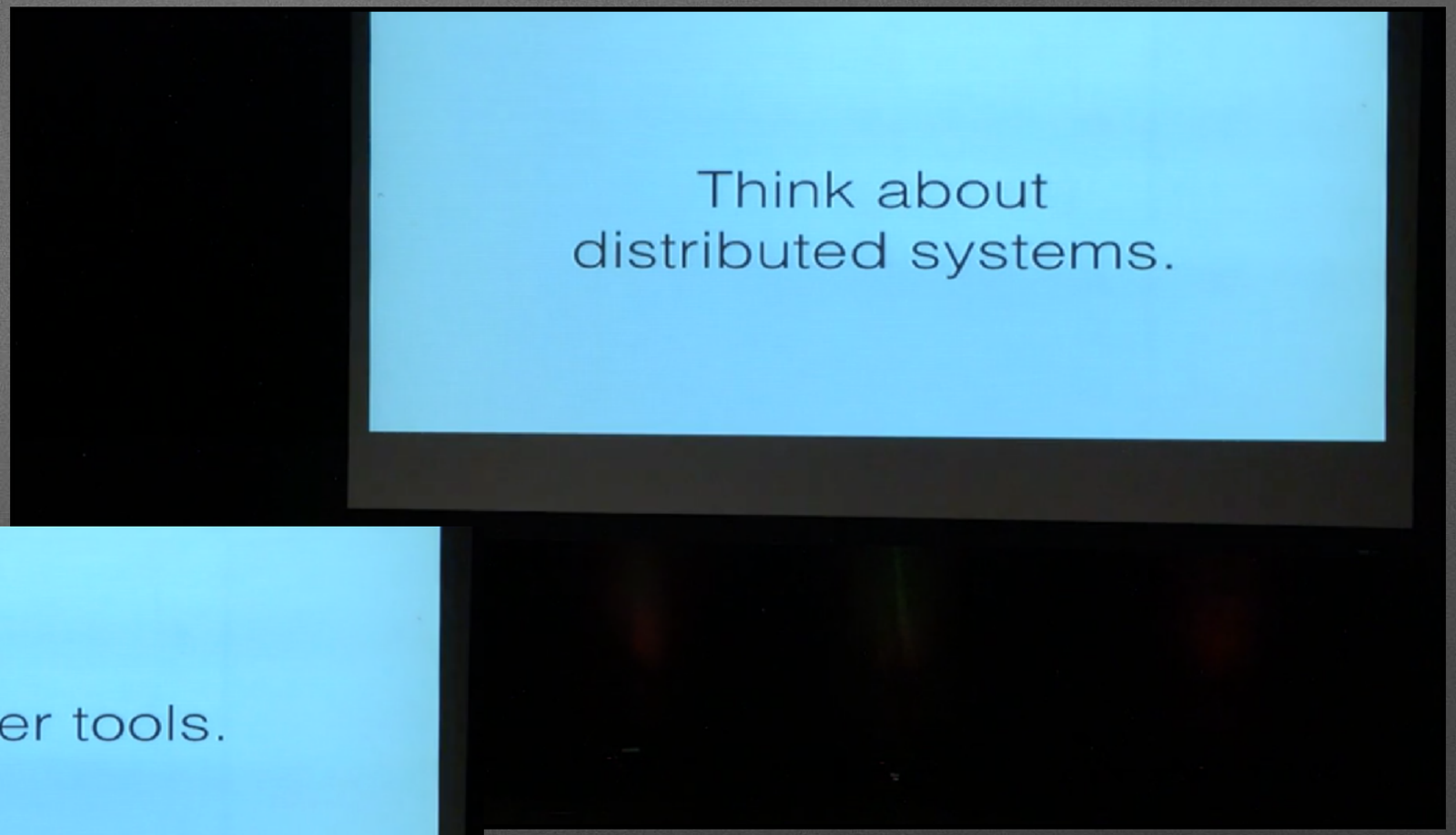
Observability-Driven Development

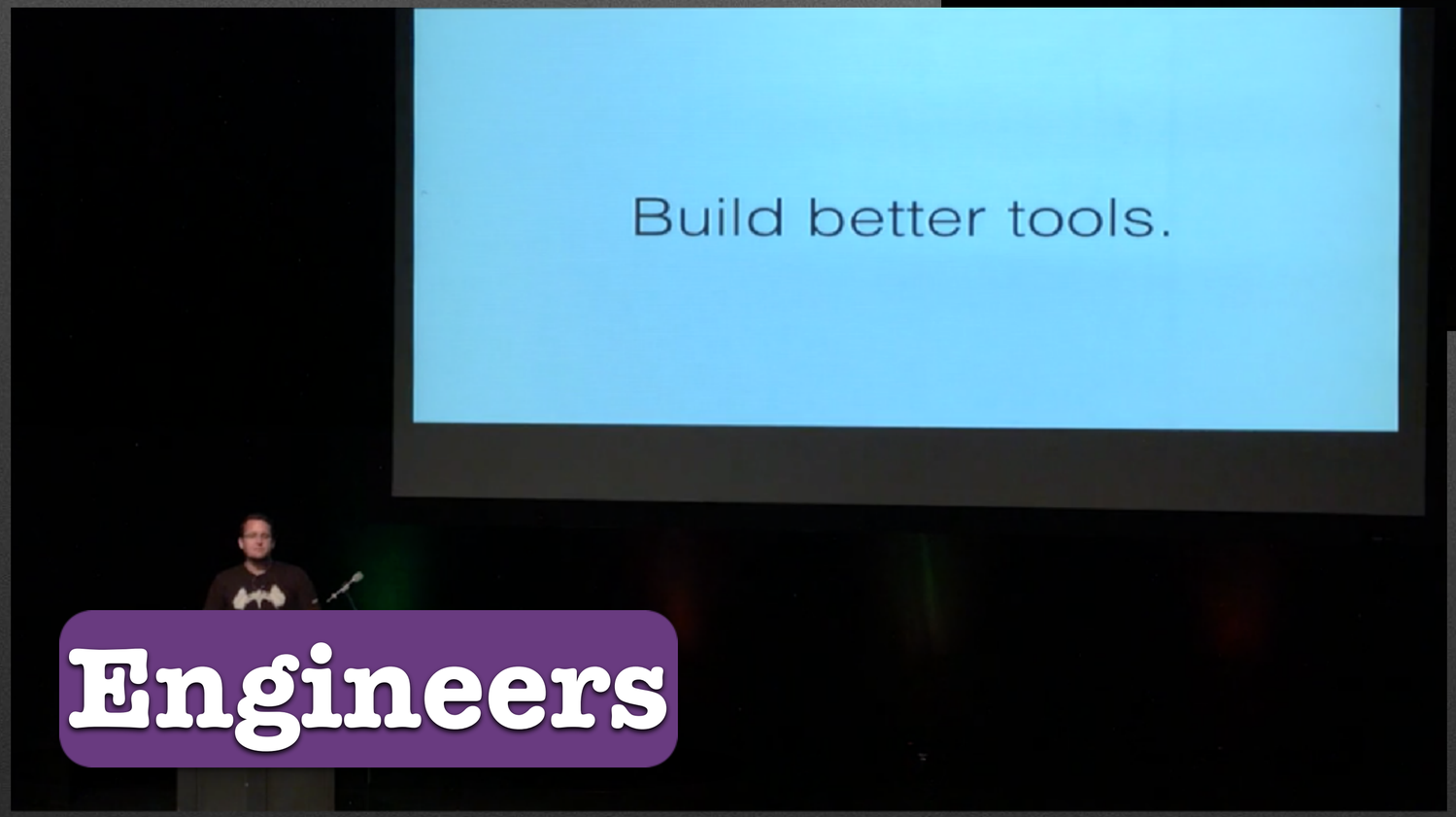
Watch it run in production.

Accept no substitute.

Get used to observing your systems when they AREN’T on fire.

Engineers



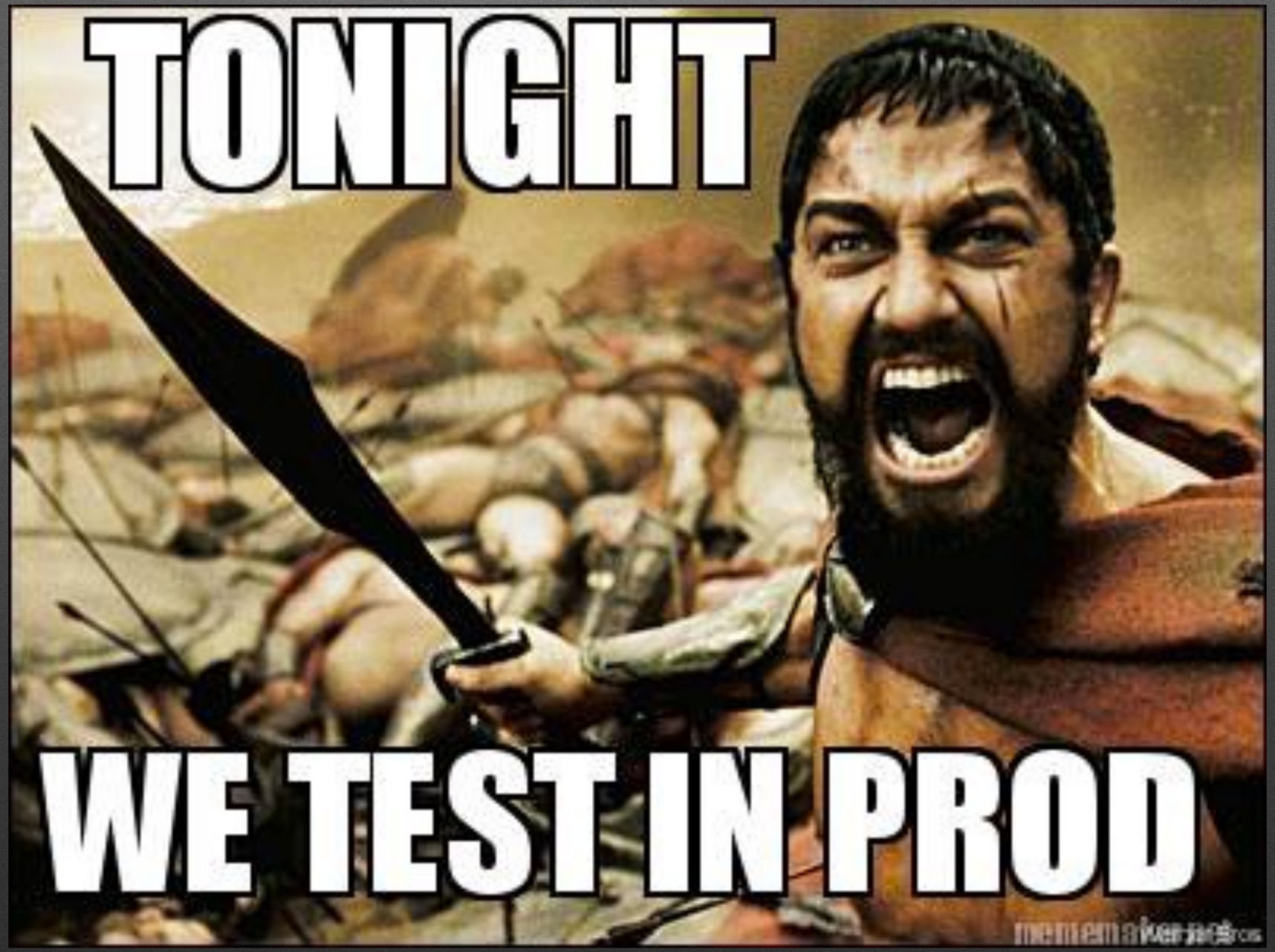


Engineers

Let's build tools that don’t lie to us.

Let’s get comfortable with the messiness of reality

Let’s automate ourselves out of a fucking job.


TONIGHT
WE TEST IN PROD

Glorious Future™

high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
fun.


You win …
Drastically fewer paging alerts!

Black swans are the norm

you must care about max/min, 99%, 99.9th, 99.99th, 99.999th …

You can’t hunt needles if your tools don’t handle extreme outliers, aggregation by arbitrary values in a high-cardinality dimension, super-wide rich context…

you must be able to explore any individual event.

find and describe any needle in the haystack

Metrics:System::Events:Request

converging trends:

monolith => microservices

“the database” => polyglot persistence

users => developers

single tenant => multi tenancy app

could reason about => def cannot reason about

distributed systems:

it is often harder to find out <u>where</u> the problem is, than <u>what</u> the problem is.

7 commandments for a Glorious Future™

well-instrumented
high cardinality
high dimensionality
event-driven
structured
well-owned
sampled
tested in prod.

Charity Majors
@mipsytipsy