Hashtag #cassandra12

Hastur: Open-Source Scalable Metrics with Cassandra

Noah Gibbs | August 8, 2012

noah@ooyala.com

@codefolio

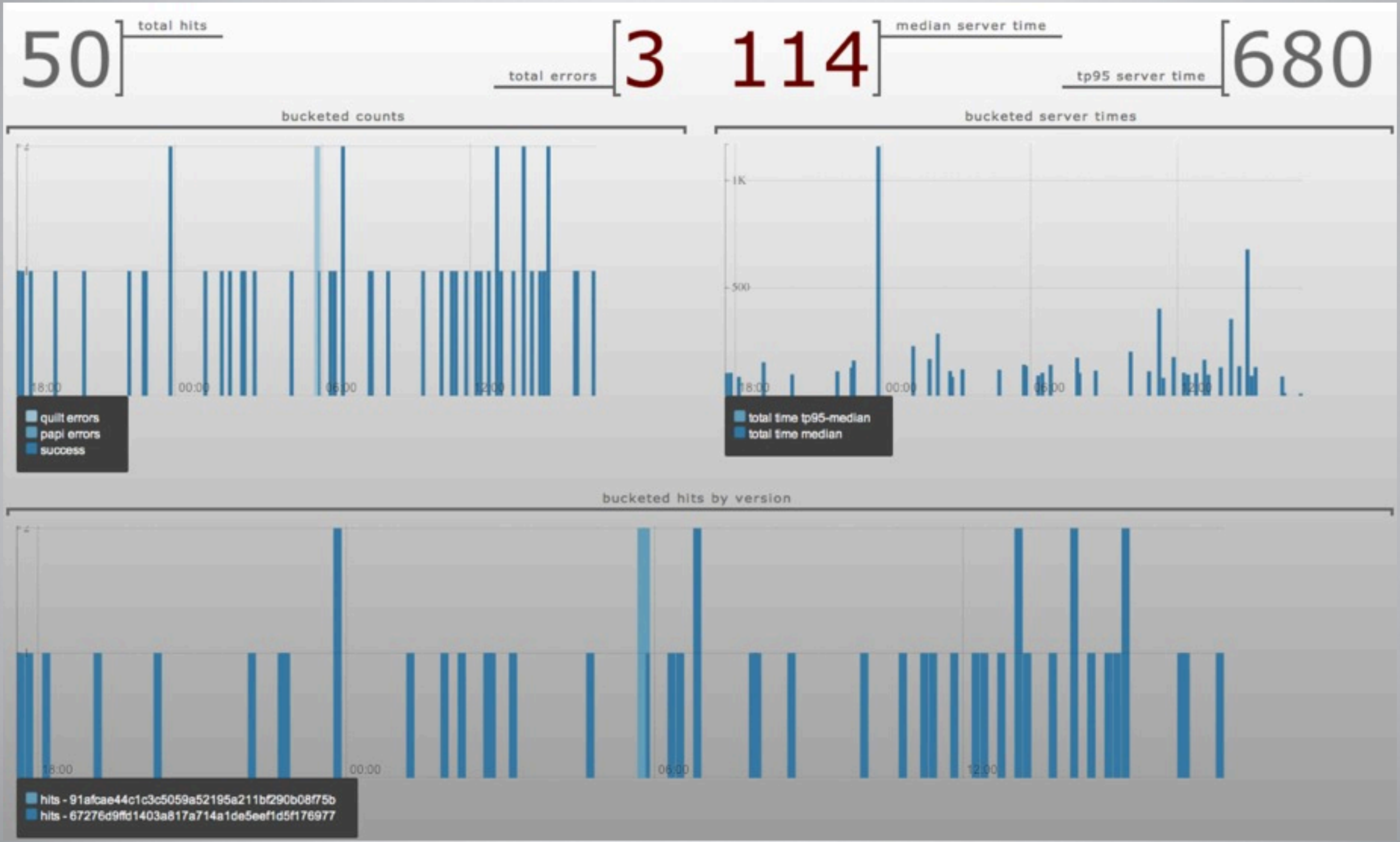
OOYALA

http://github.com/ooyala/hastur-server

In this Talk:

- What is Hastur? Quick Intro.
- What Cassandra Schema? In Depth.
- What's In Progress?

Hastur Live Dashboard



50 total hits

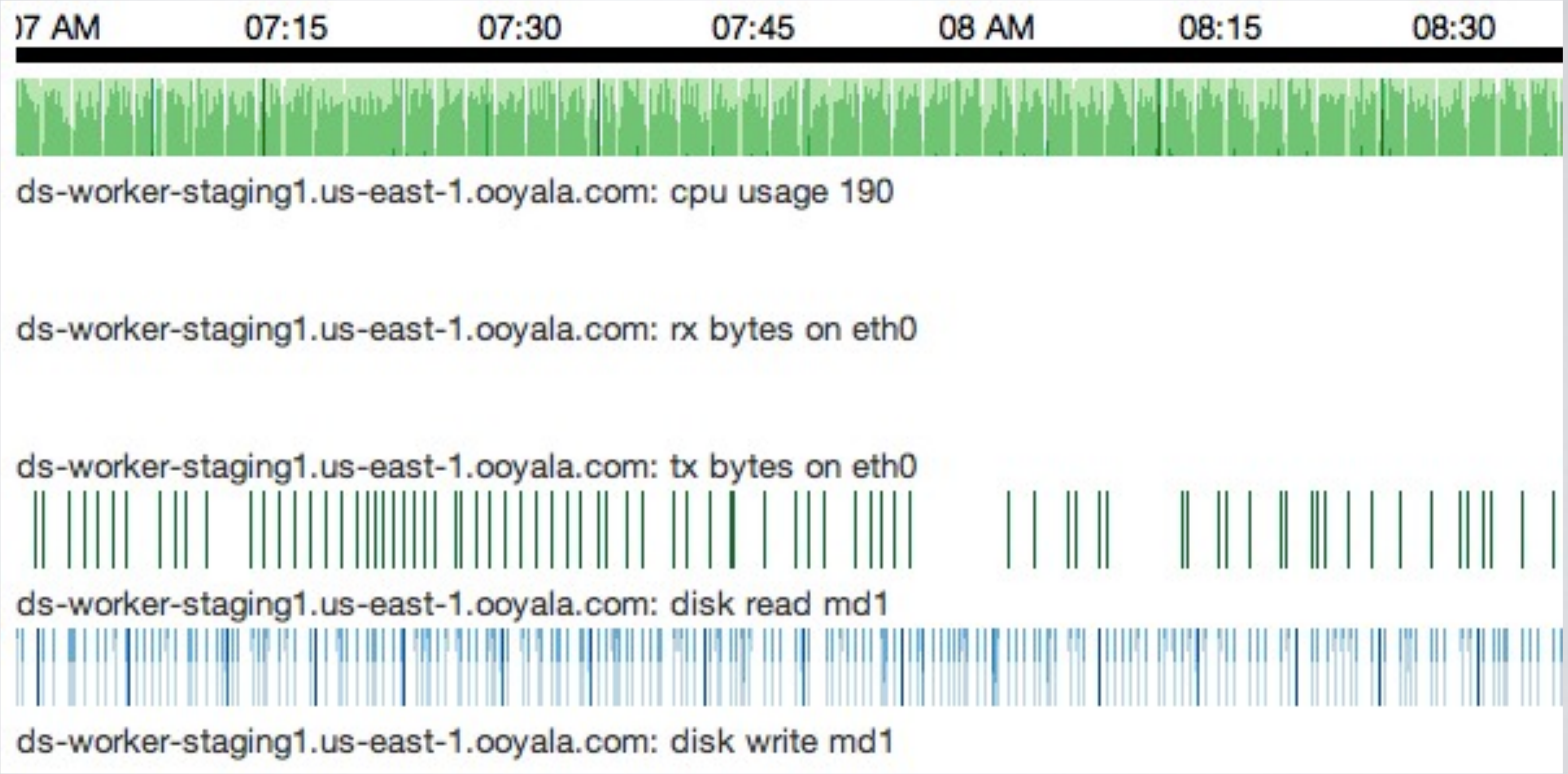
total errors 3

114 median server time

tp95 server time 680

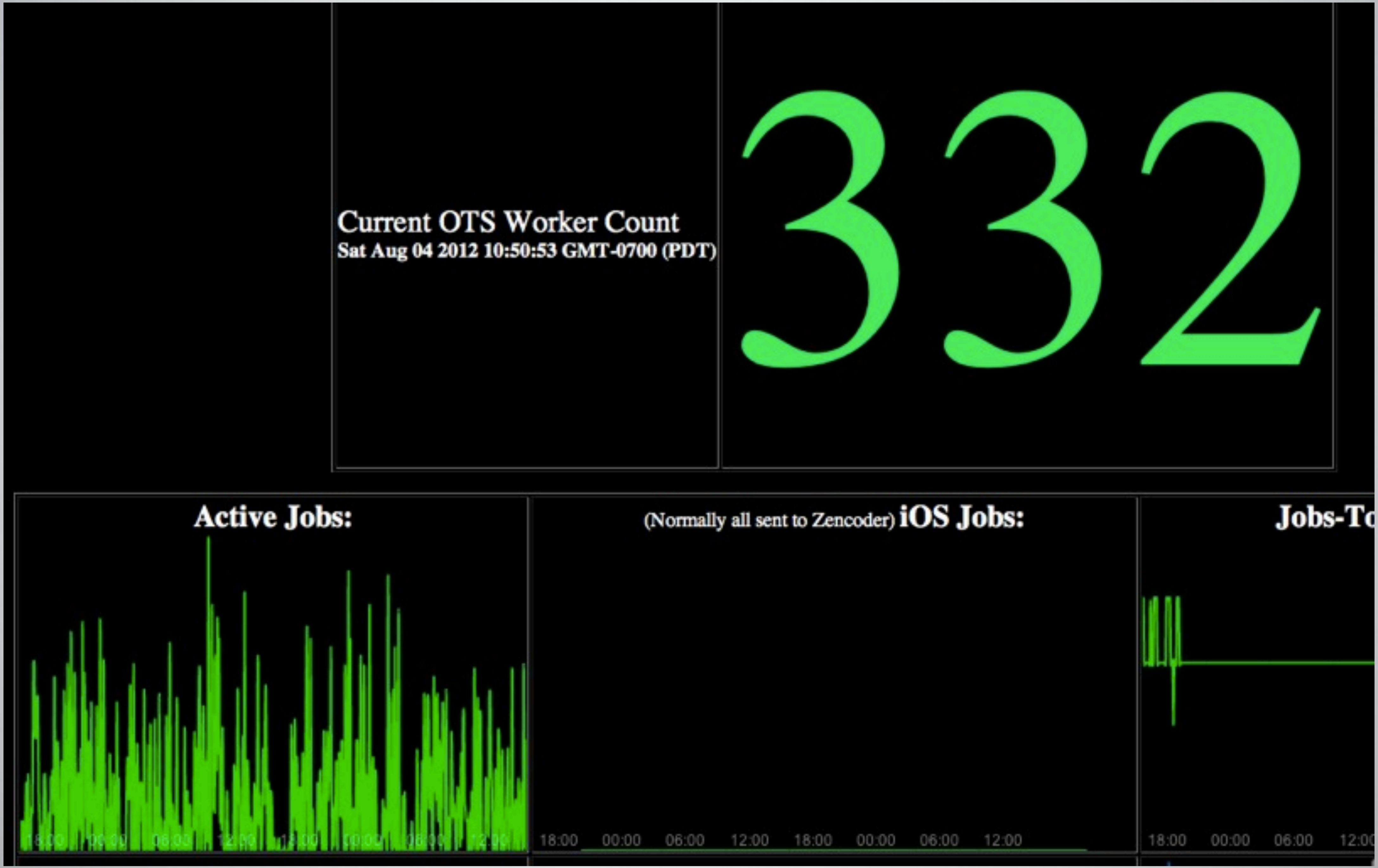
OOYALA

Hastur Live Dashboard


)7 AM
07:15
07:30
07:45
08 AM
08:15
08:30
ds-worker-staging1.us-east-1.ooyala.com: cpu usage 190
ds-worker-staging1.us-east-1.ooyala.com: rx bytes on eth0
ds-worker-staging1.us-east-1.ooyala.com: tx bytes on eth0
ds-worker-staging1.us-east-1.ooyala.com: disk read md1
ds-worker-staging1.us-east-1.ooyala.com: disk write md1

Bindings for D3, Cubism and Rickshaw. Easy to support other JavaScript graphing libs. The JavaScript directly queries Hastur's REST retrieval service.

Hastur Live Dashboard



Current OTS Worker Count
Sat Aug 04 2012 10:50:53 GMT-0700 (PDT)

332

Hastur

- Metrics, like StatsD and Graphite
- CollectD-Style System Statistics
- REST Interface, JS Dashboards
- Replicated, Fault-Tolerant, Scalable

Cassandra Challenges:

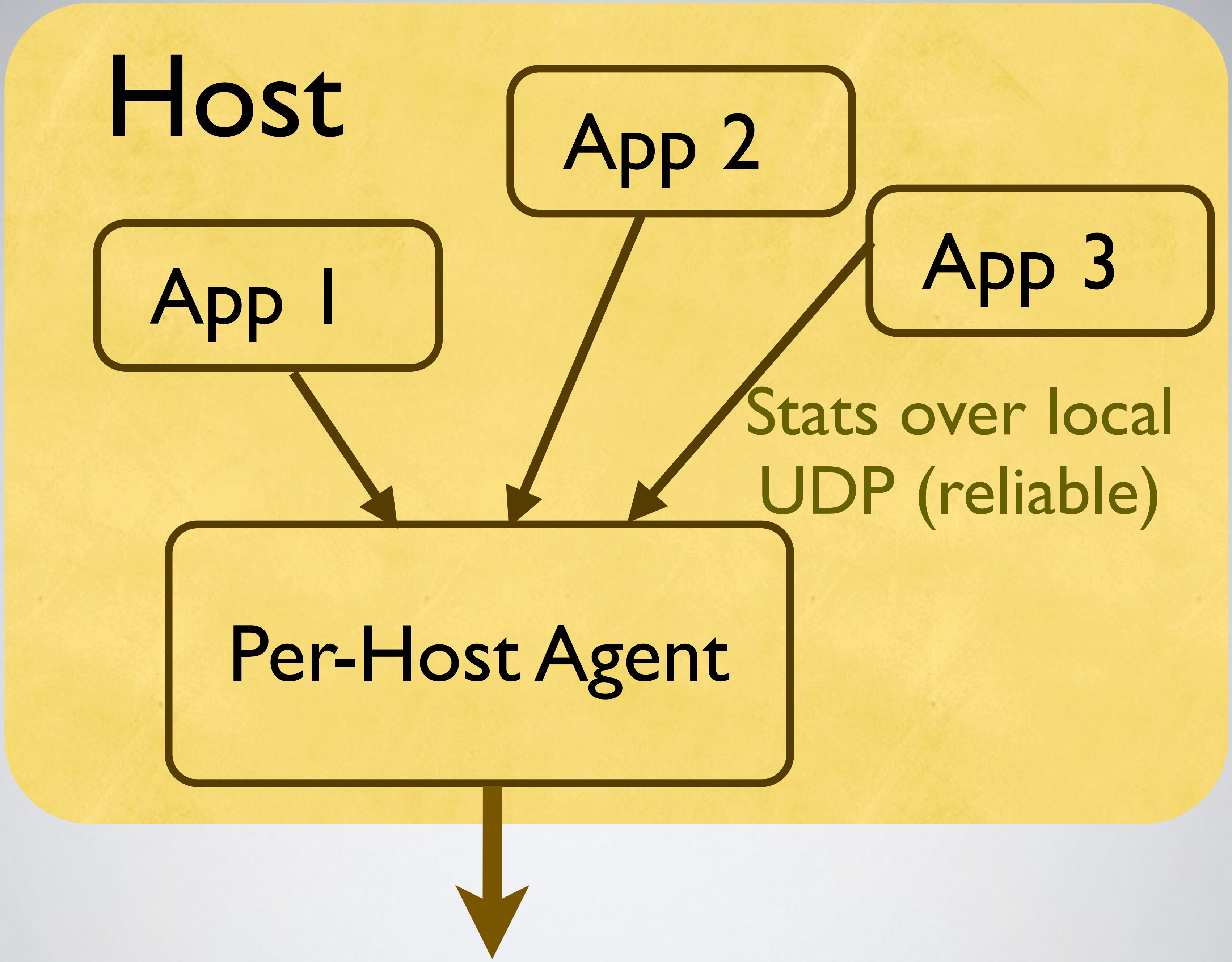
- High, Unpredictable Write Volume
- Varying Schema, Variable Msg Size
- 2 Types of Series - Data, Lookups
- All time-series, even metadata - no supplemental DB

Sample Hastur Message

```
{
  "type": "gauge",
  "uuid": "91c61ff0-8740-012f-e54a-64ce8f3a9dc2",
  "name": "authserver.request.latency",
  "value": 0.3714,
  "timestamp": 1329858724285438,
  "labels": {
    "app": "authserver",
    "pid": 138423,
    "req_type": "anon_user"
  }
}
```

Fields vary by msg type

Arbitrary per-msg labels



Host

App 1

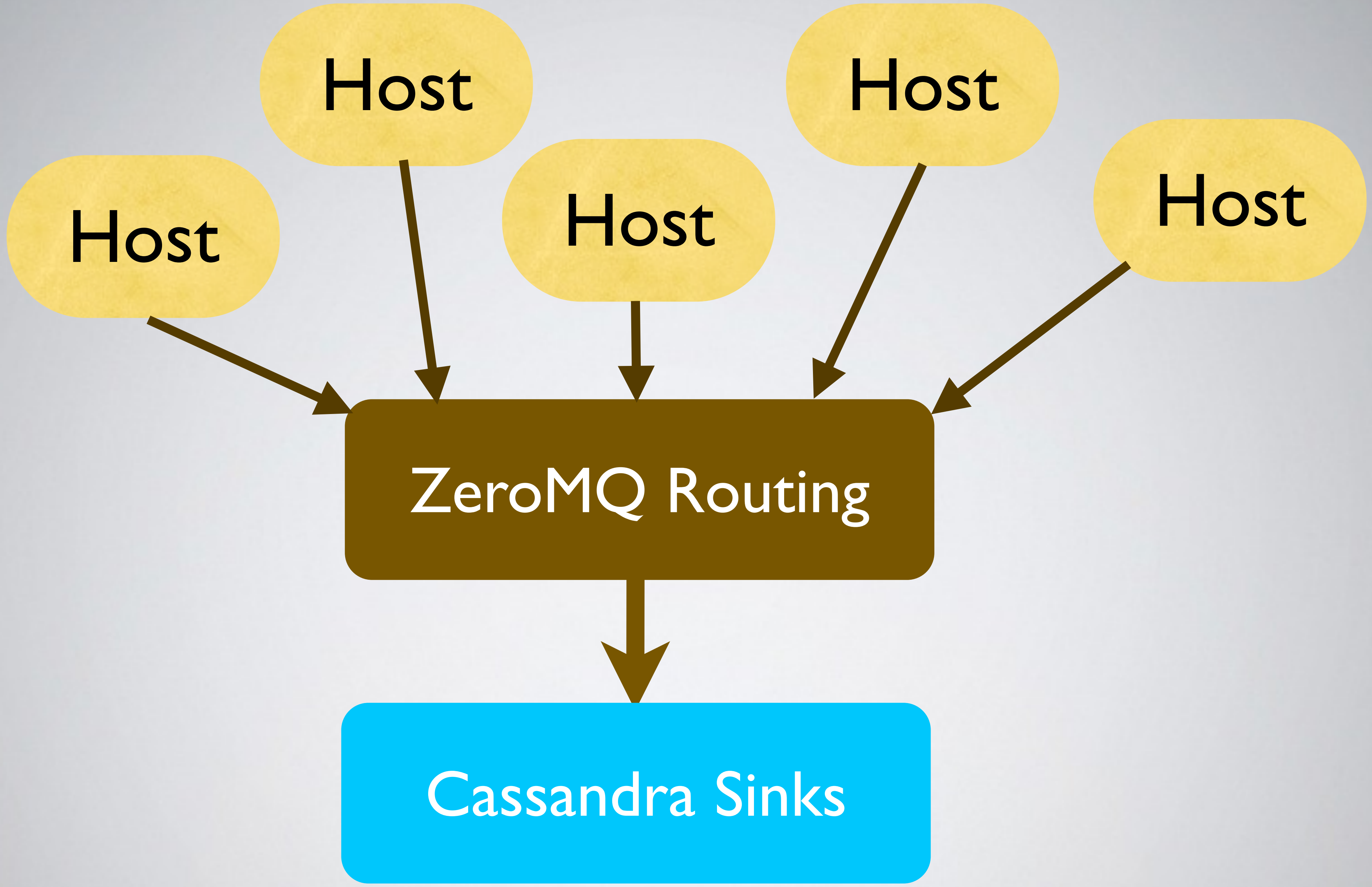
App 2

App 3

Stats over local UDP (reliable)

Per-Host Agent

Stats over ZeroMQ (redundant, failover)



Host

Host

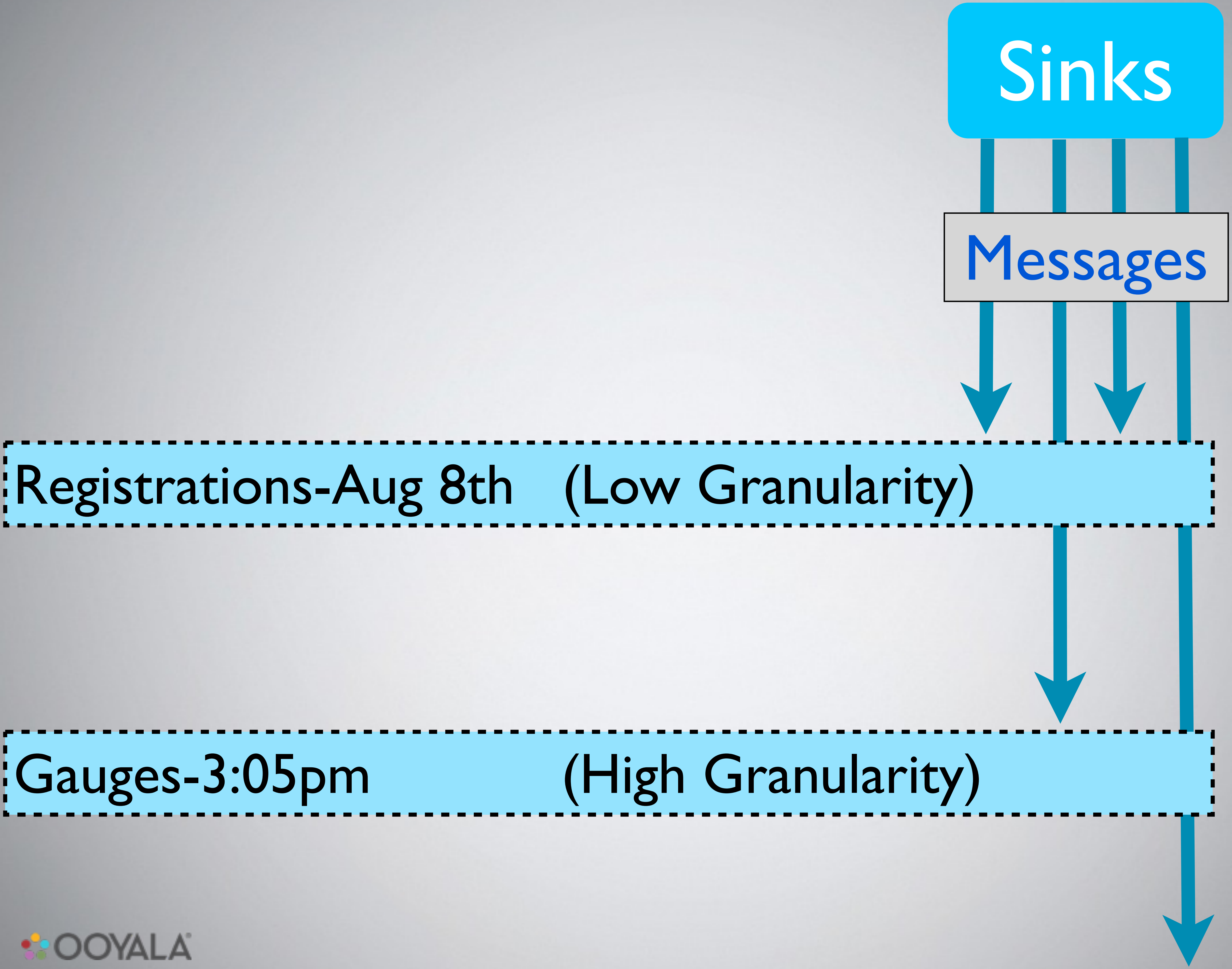
Host

Host

Host

ZeroMQ Routing

Cassandra Sinks



Sinks

Messages

Registrations-Aug 8th (Low Granularity)

Gauges-3:05pm (High Granularity)

Gauges-3:05pm (High Granularity)

This writes several things to several different rows:

Location	Value
5-min archive row	JSON struct
5-min value row	0.3714 (latency value)
message names row	authserver.request.latency
UUIDs row	host’s UUID
app-name row	app name, UUID

Columns and Comparators

- Use reversed comparator - return most recent first when limited.
- Composite keys are great, but Ruby support is mixed. We use Bytes.
- Column keys make the easiest and fastest indices.
- Timestamp everything, modify nothing.

Messages, Values - Data Series

Row Key

91c61ff0-8740-012f-e54a-64ce8f3a9dc2-1329858600000000

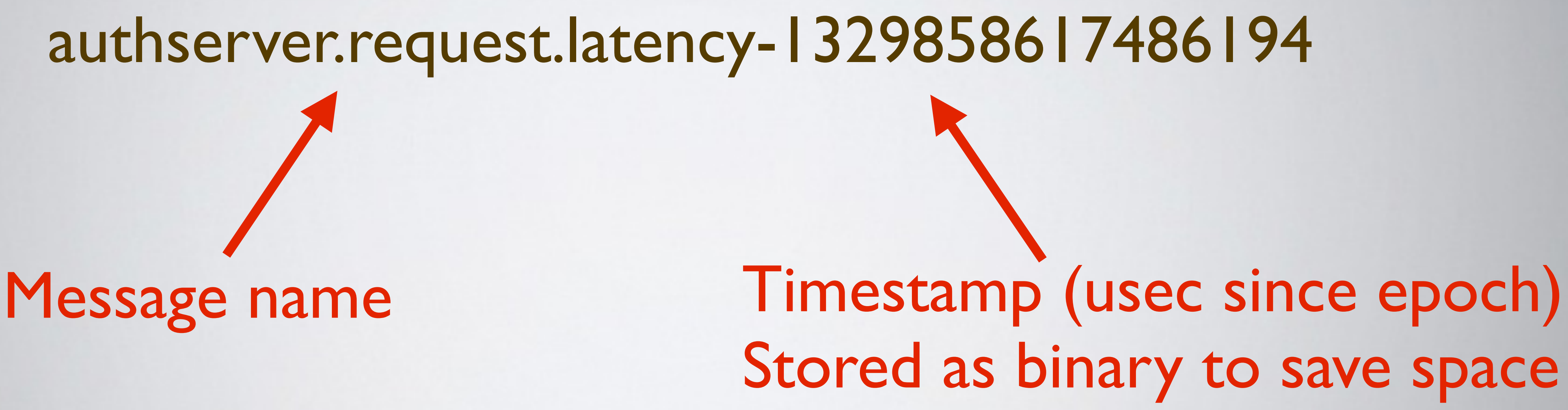
UUID

Timestamp, to 5 minutes precision

Different message types have different time intervals. Stats are 5 minutes, low-frequency message types are up to one day.

Messages, Values - Data Series

Column Key



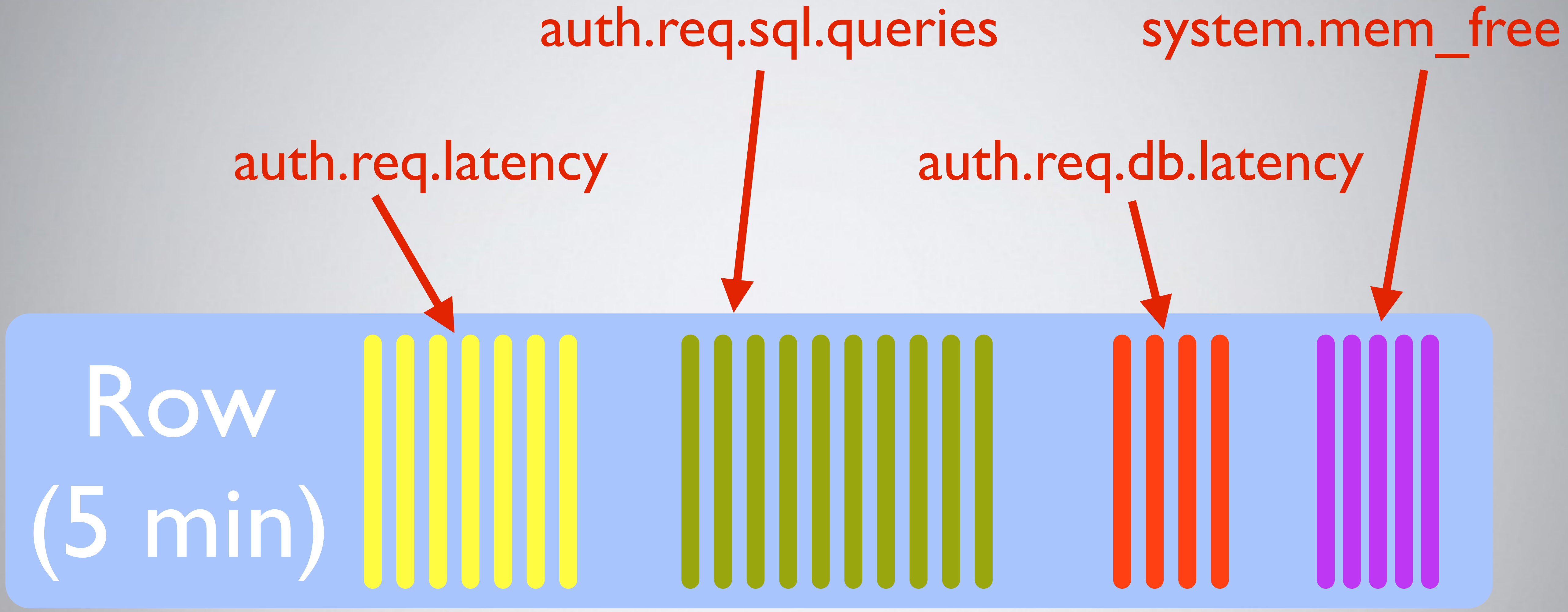
authserver.request.latency-1329858617486194

Message name

Timestamp (usec since epoch)
Stored as binary to save space

Column_slice allows searching by message name or message prefix - e.g. “authserver.*”

Data Series



This row contains all gauges (a statistic type) for this host for this five minute period.

Data Series are Huge!

- JSON gives great flexibility, easy labels
- But data series are huge writing JSON!
- Cass over Btrfs - compress w/LZO.
- Repetitive JSON = huge compression! Specific data on a later slide.

Lookup Series

Row Key

name-1329782400000000

app-name-1329782400000000

uuid-1329782400000000

Timestamp, truncated to day

Look up message name, application name or UUID, always per day.

Lookup Series

Column Key

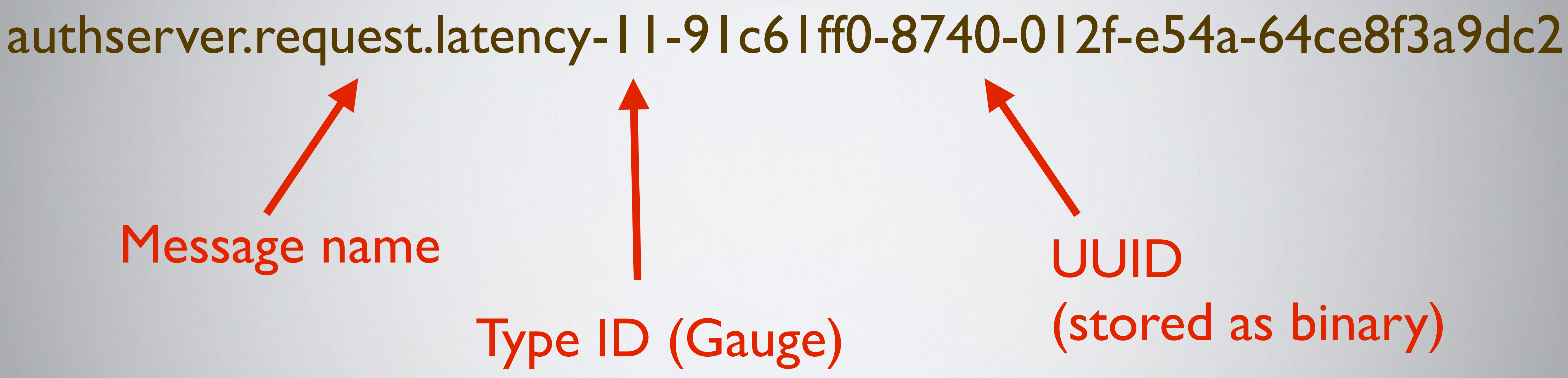
For app name or UUID, just use the app name or UUID itself as the column key.

That app name or UUID is written many times... Always with no column value. Cassandra combines writes and SSTables stay tiny.

The CF with all lookup tables is eleven MB on our benchmark node. The data is 200GB.

Lookup Series

The Rebel: Message Names



authserver.request.latency-11-91c61ff0-8740-012f-e54a-64ce8f3a9dc2

Message name

Type ID (Gauge)

UUID
(stored as binary)

The message-name column ID is larger because you need to know what column family to look in... Since you can't range-scan row keys, more info is needed.

No Cassandra Built-In Indices?

We range-scan almost everything to get double- and triple-duty out of our indices. Cassandra built-in indices aren't bad, but they don't do that.

No Cassandra Compression?

Built-in Cassandra compression claims to compress across columns with identical names. All our data columns are timestamped, so no two will ever have identical names.

Numbers

“Benchmark” Cassandra node
Size: JSON vs Value

	Size	% of full size
Gauge JSON, raw	34 GB	
Gauge values	14 GB	41%
Counter JSON, raw	100 GB	
Counter values	23 GB	23%

Real Production Data

Numbers

“Benchmark” Cassandra node
LZO Compression

	Size	% of full size
Cassandra Size	199 GB	
On-Disk Size	111 GB	56%

Real Production Data

Quick Summary: Future Directions

- Automatic Retention Policy - Delete or move to long-term S3 storage
- Alerting - scan in arrival order, and check automatic thresholds
- On-Demand rollups instead of manual
- Smart label queries - a huge job!

github.com/ooyala/hastur-server

Questions?

Thanks to Al Tobey, co-architect of Hastur. Benchmark numbers are his!

noah@ooyala.com

@codefolio

#cassandra12

github.com/ooyala/

hastur-server (infrastructure)

hastur (ruby client)

hastur-c (C client)

OOYALA

THANK YOU