



CIRCONUS

*Insight  
Into Failures*



## Context

We take measurements from things.

We make pictures from that data.

We do offline analytics on that data.

We do stream-based analytics for real-time alerting.

Simple, right?



# Constraints

Billions of unique things being measured.

Millions of measurements per second per thing.

Things can be disconnected for a while.

Multi-tenant.



# Agenda

## Components

Message Queues

Storage operations

Storage performance

Analytics

## Problems

Performance and isolation

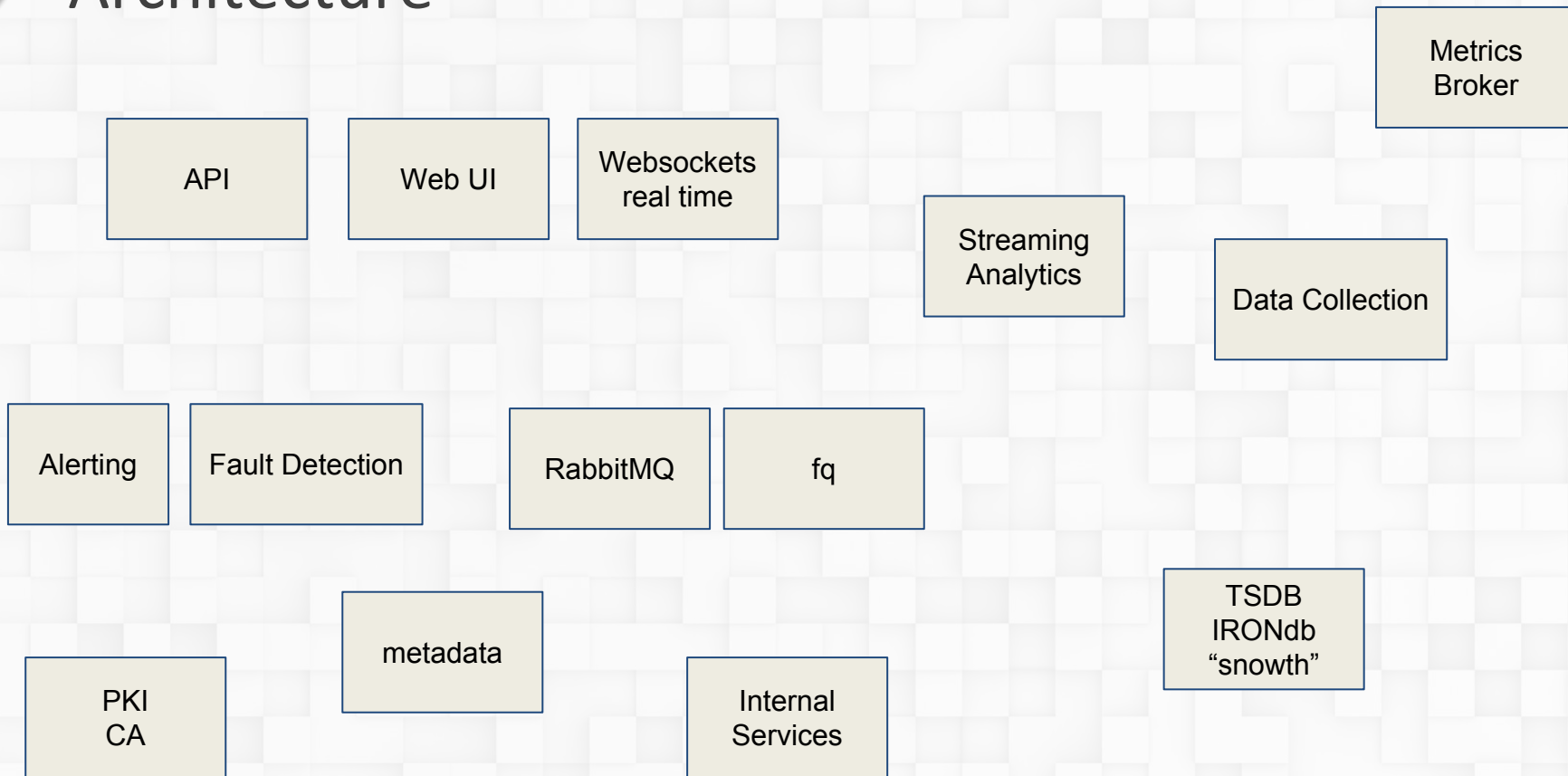
Space, ops burden

performance, and scale

Streaming vs. historic, model, performance



# Architecture





# Message Queues

More macroservice than microservice.

Components learn via control messages.

The streaming analytics and fault-detection services see measurement data.



# The failure of RabbitMQ

All software will break outside of operating parameters.

RabbitMQ has/had:

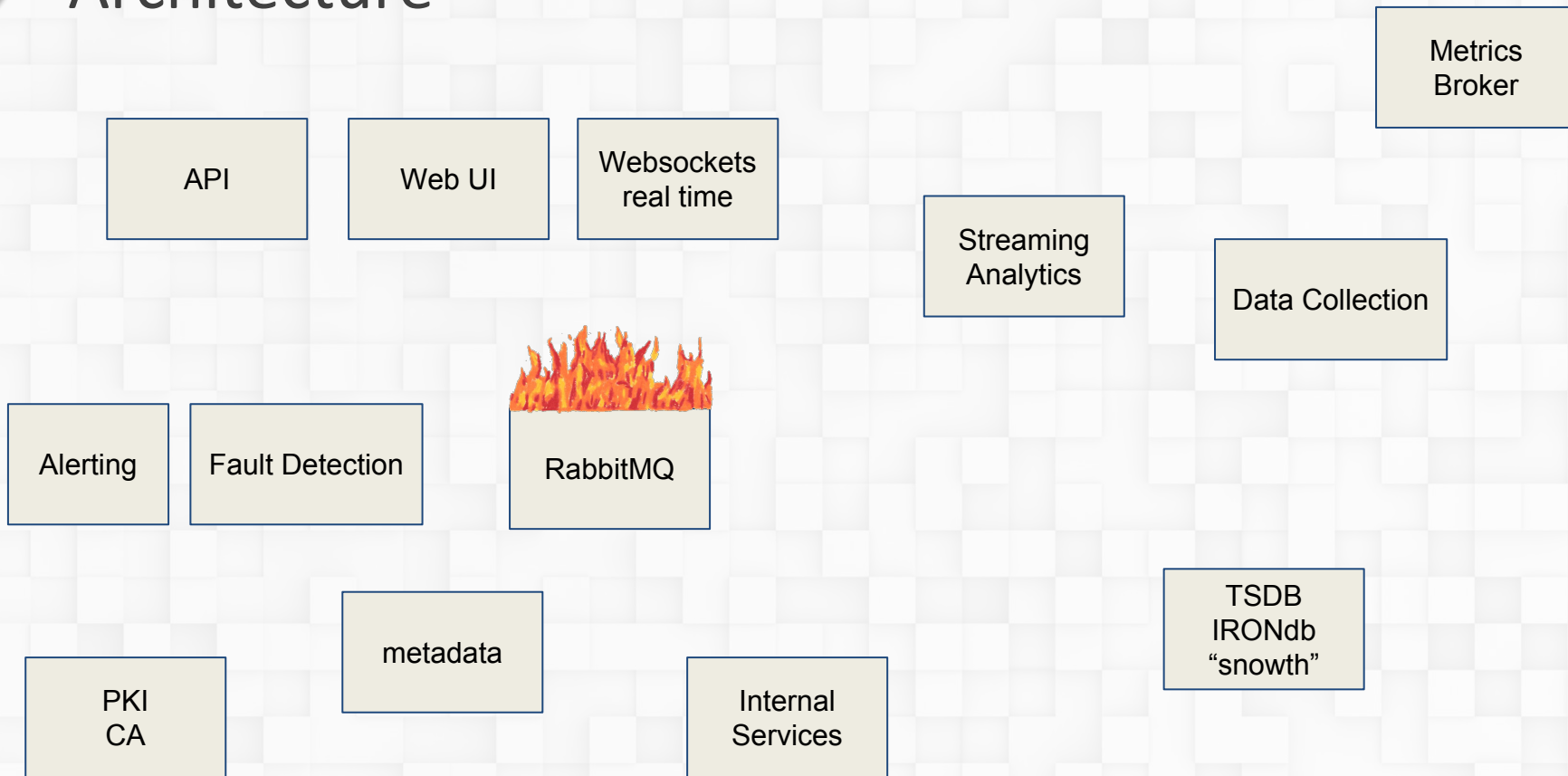
- 1) poorly documented operating parameters
- 2) horrendous failure pathologies.

A failed data plane is disruptive.

A failed control plane is debilitating.



# Architecture







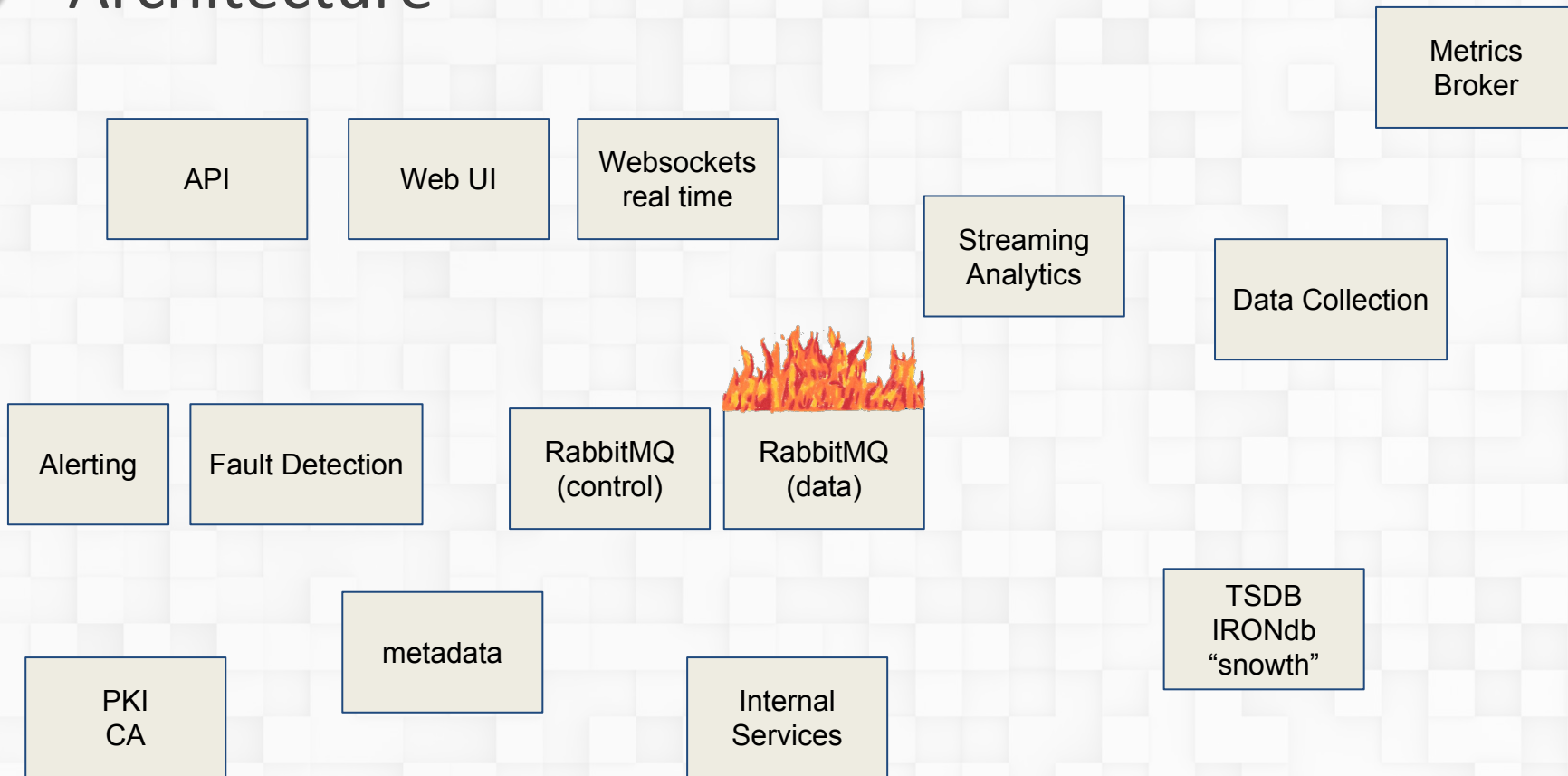
# Step 1

Separate our control plane and our data plane by running two completely independent RabbitMQ instances.

This successfully isolated tragedy in our environment to the data plane.



# Architecture





## Step 2

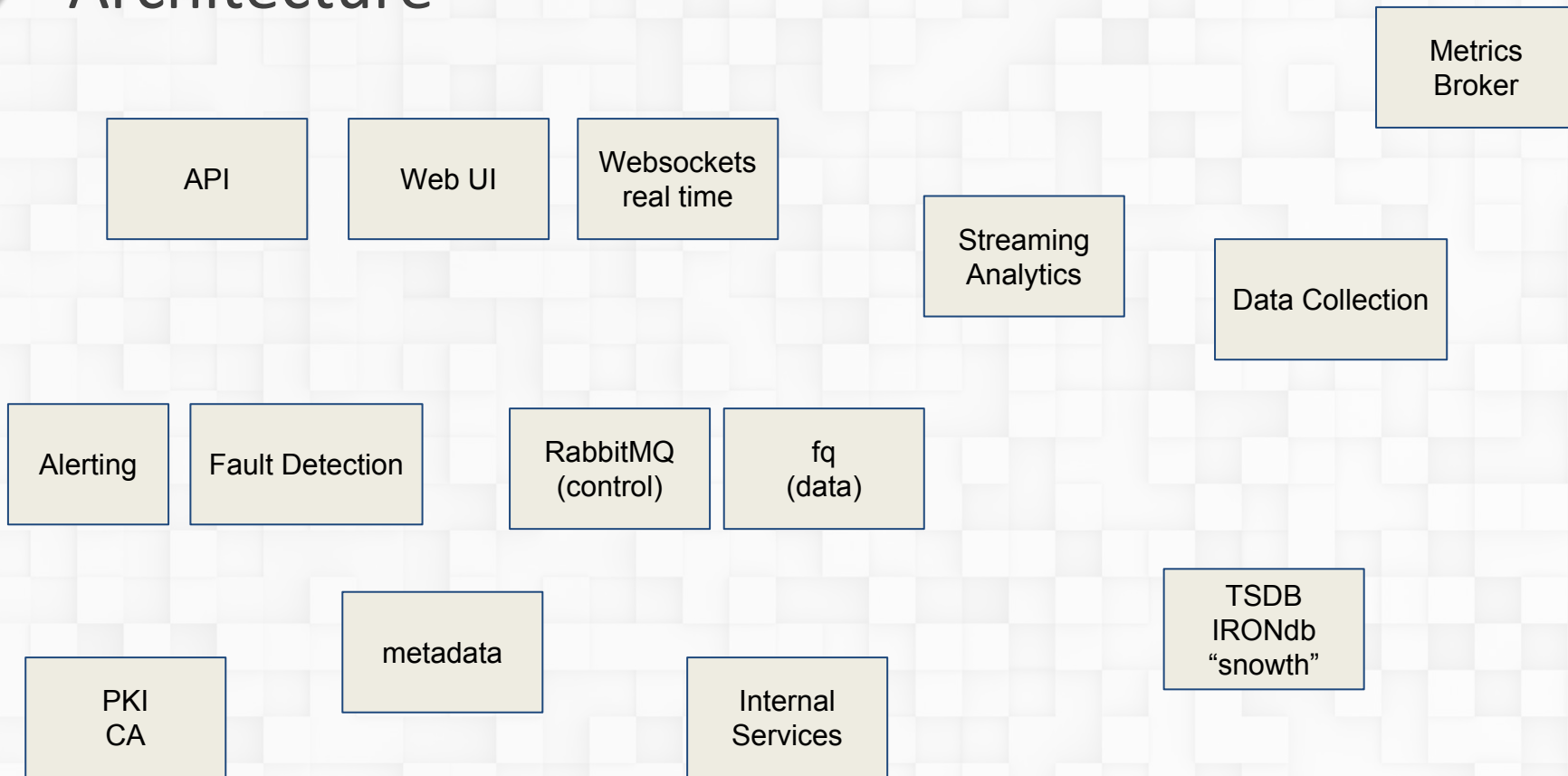
Authored and deployed fq for the data plane.

It is faster and more accommodating of our data patterns than Kafka.

If Kafka were available at the time, we'd have gone with that and not looked back... "good enough."

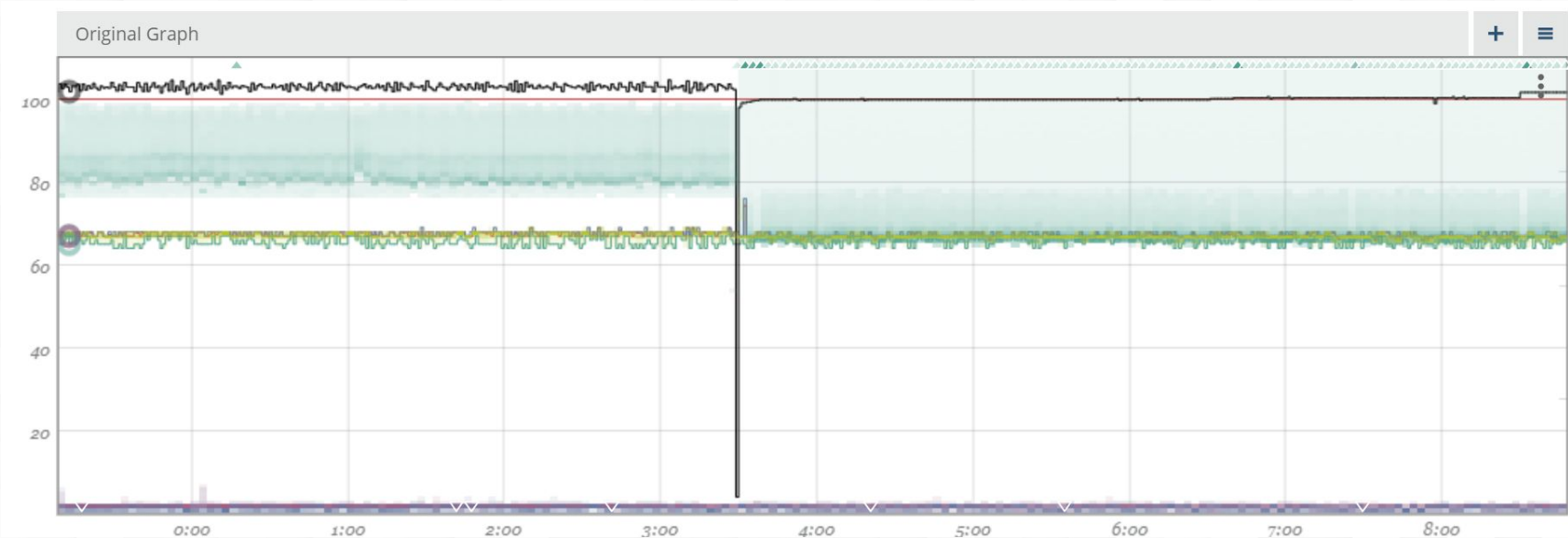


# Architecture





# Tracking end-to-end latencies





# Storage

At launch (2010), we stored metrics in a highly dense columnar format in PostgreSQL (similar to how Timescaledb works)

Operational burden upon node failure was too high.

Storage considerations were too inefficient.

All “fixable,” but it was clearly the wrong tool for the job.



# Storage

We defined our own storage format.

- “NNT” statistical aggregate storing 7 aggregation facets per rollup.
- LMDB-backed block-extent format.
- OpenZFS safety and laz4 compression.
- 8.5 bits per facet in practice.



# Storage

We defined our own storage format + histograms

- **llhist** a sparse 46k bin histogram format with an upper limit of 5% error.
- **rocksdb** stores time-series histograms in llhist format.





# Storage

We built our own database: “snowth” now “IRONdb”

This was a huge endeavour with bold operational goals:

- Zero points of failure.
- Recovery and rebuild with no operator intervention.
- Resizing with minimal operator intervention.



# Storage

Wait. Stop. Building a database is a cardinal sin.

Why build a DB when we've got:

InfluxDB, Cassandra, Riak, OpenTSDB, etc.?

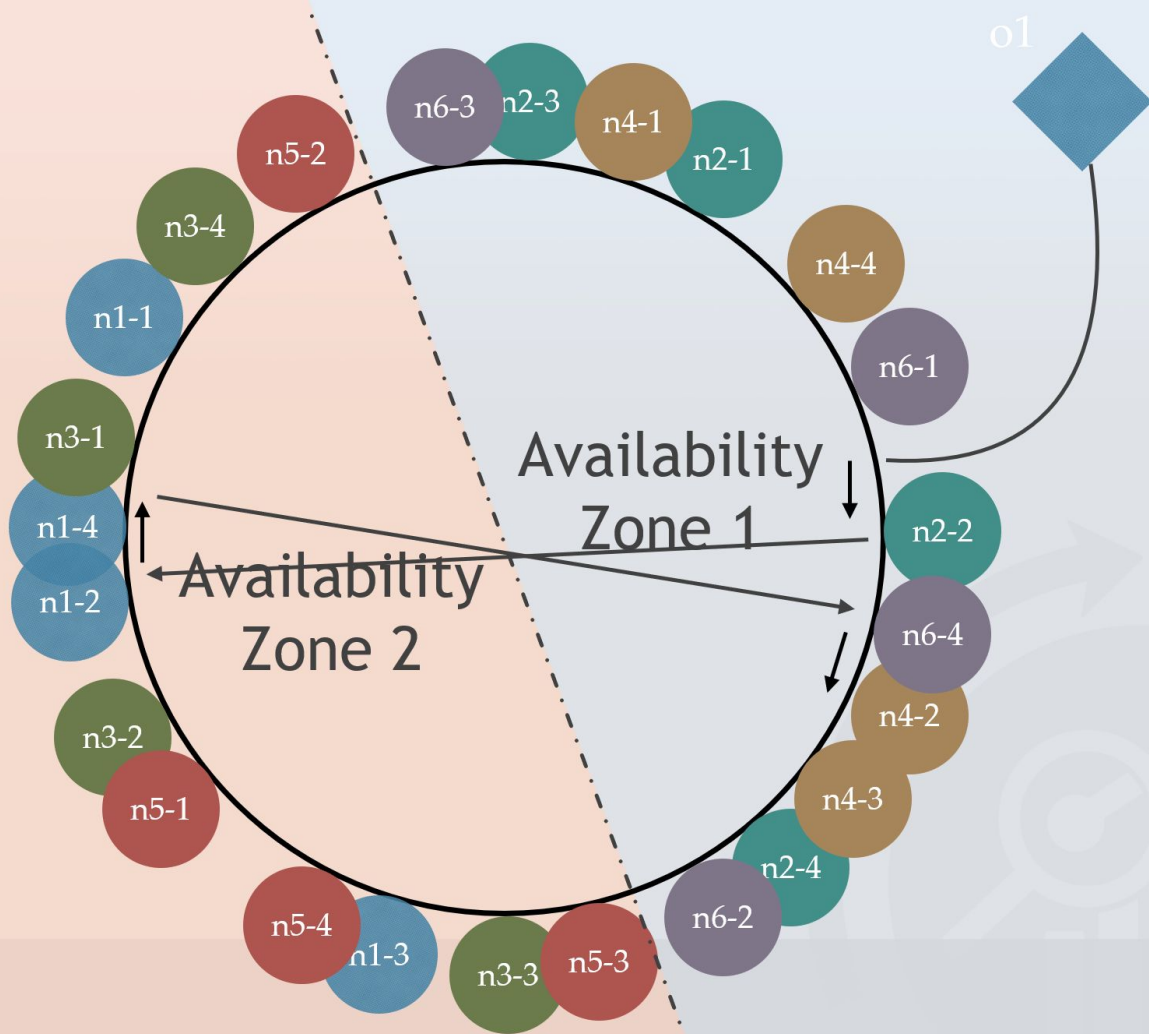
- 1) It's 2010
- 2) We realized our problem scope was smaller.
  - a) Incredibly predictable write patterns
  - b) Commutative operations



# Storage

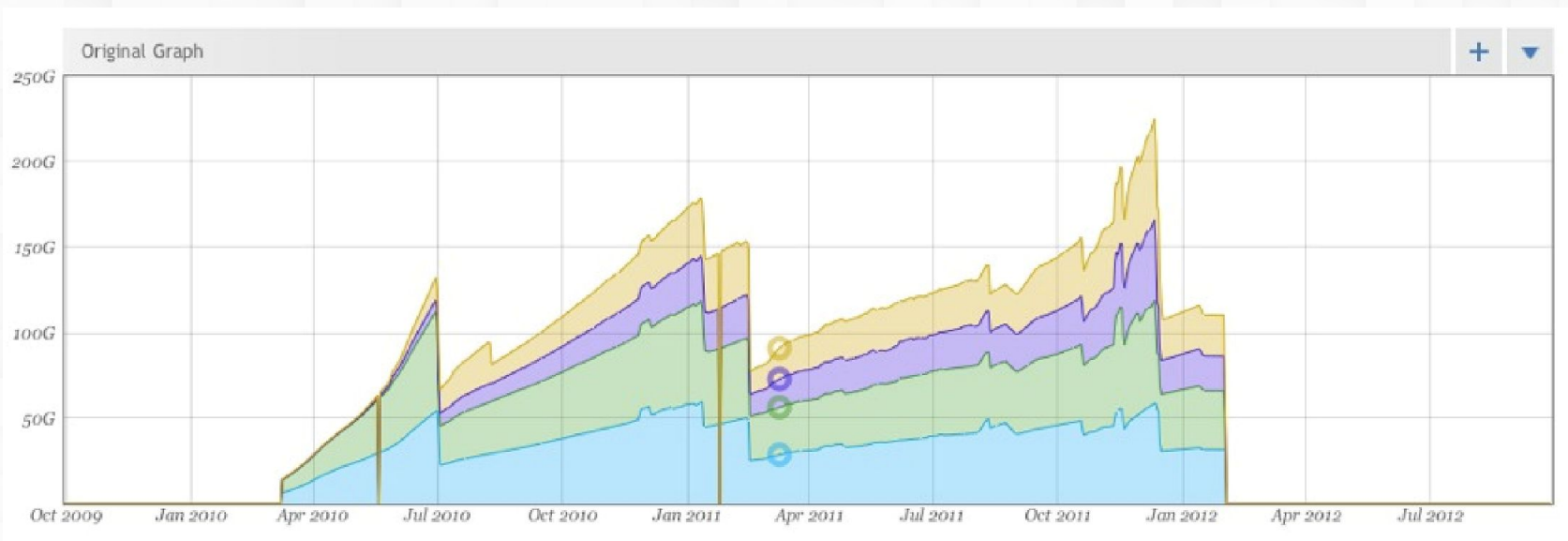
Performance, scale, and resiliency requirements:

- 1) Nodes on different continents
- 2) Full region failure
- 3) Uninterrupted ingest on failure
- 4) Uninterrupted read/analytics on failure
- 5) More than 1MM records/second/node sustained ingest



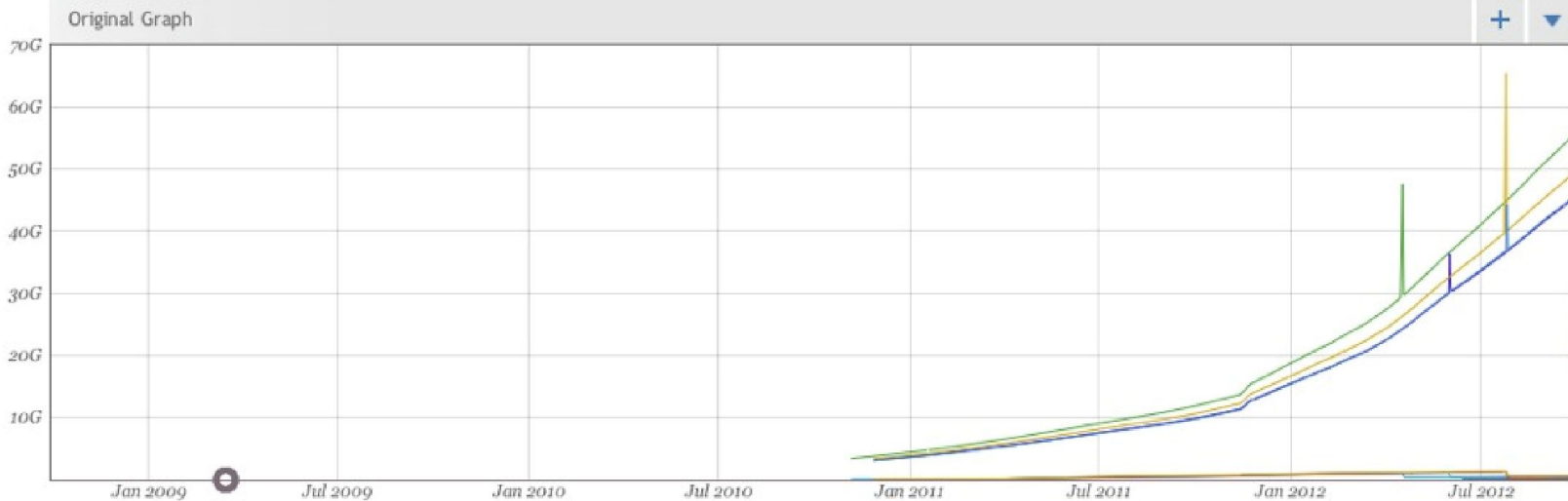


# PostgreSQL



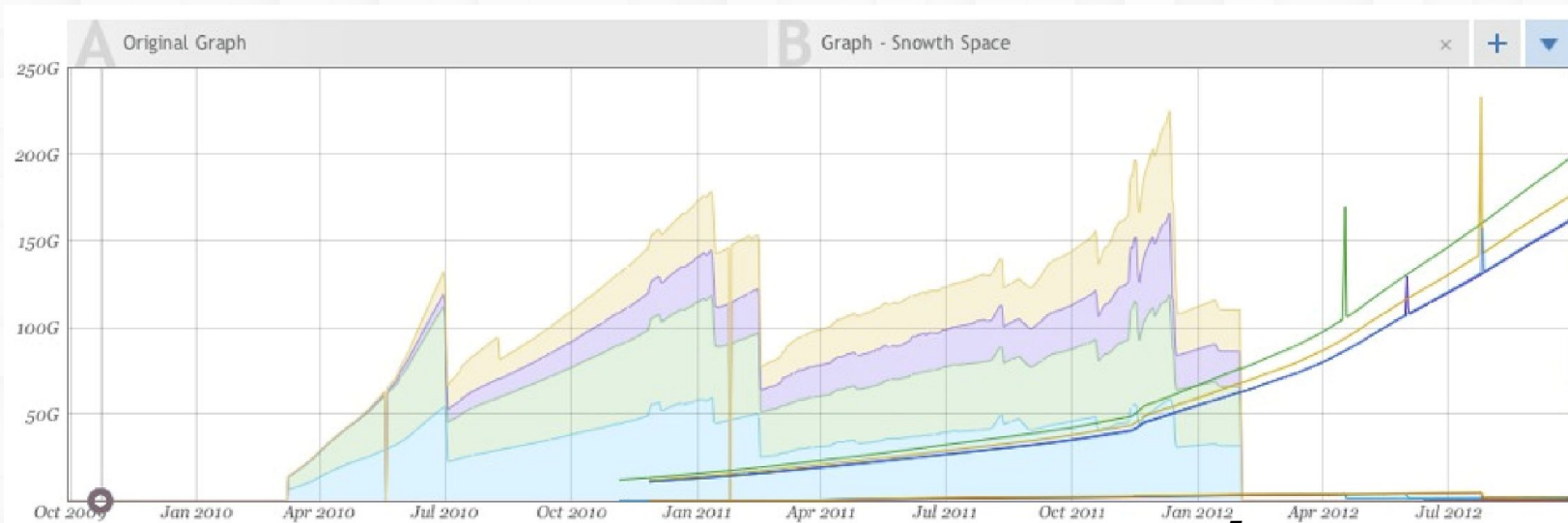


# snrowth





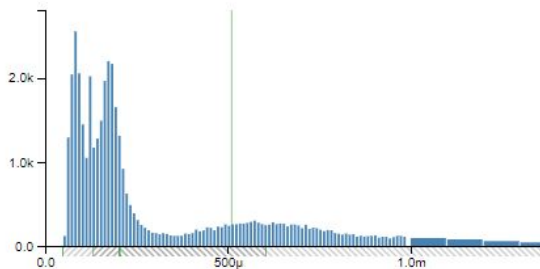
# 11 months of safety



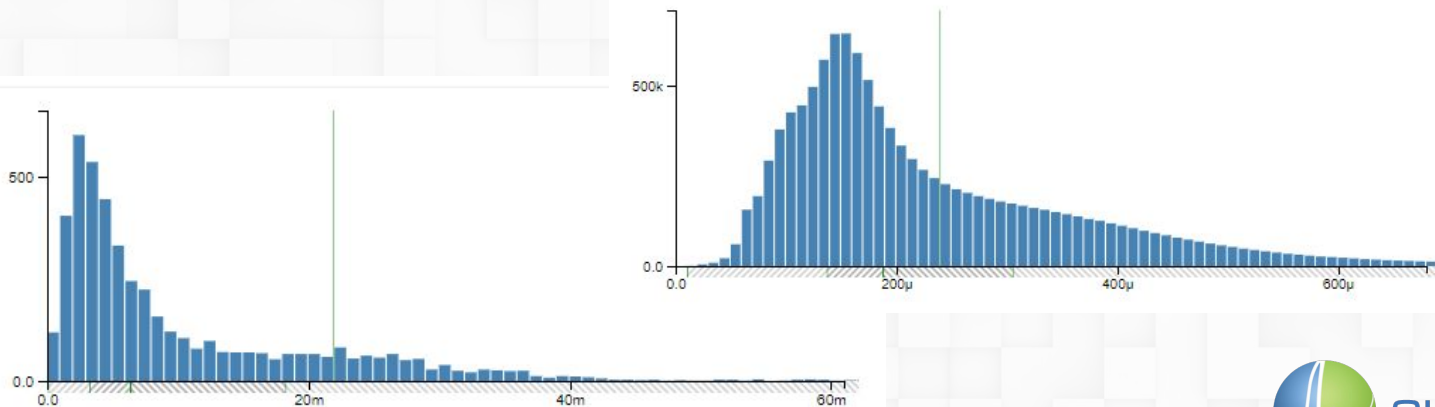


# Histogram instrumentation

Latency of snowth.db.hist.60.get`latency



Latency of put`latency







# Zipkin instrumentation

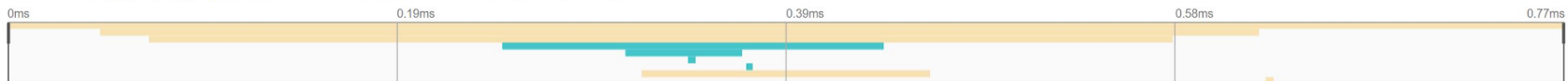
▼ snowth-00011116: /rollup/^(^([0-9a-fA-F]{4}(?:[0-9a-fA-F]{4}-){4}[0-9a-fA-F]{12})/(.+)\$



Search...

View Options ▾

Trace Start: August 22, 2018 11:41 AM | Duration: 0.77ms | Services: 2 | Depth: 5 | Total Spans: 9

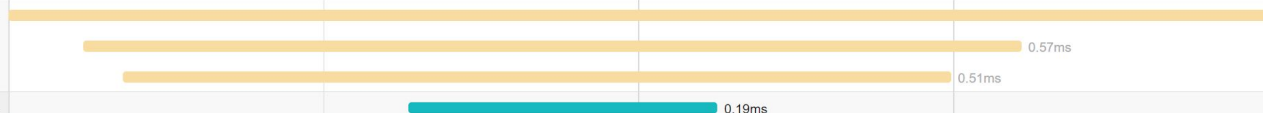


Service & Operation



0ms | 0.19ms | 0.39ms | 0.58ms | 0.77ms

- ▼ snowth-00011116 /rollup/^(^([0-9a-fA-F]{4}(?:[0-9a-fA-F]{4}-){4}[0-9a-fA-F]{12})/(.+)\$
- ▼ snowth-00011116 eventer\_callback
- ▼ snowth-00011116 curl: request
- ▼ snowth-00011114 /rollup/^(^([0-9a-fA-F]{4}(?:[0-9a-fA-F]{4}-){4}[0-9a-fA-F]{12})/(.+)\$



/rollup/^(^([0-9a-fA-F]{4}(?:[0-9a-fA-F]{4}-){4}[0-9a-fA-F]{12})/(.+)\$

Service: snowth-00011114 | Duration: 0.19ms | Start Time: 0.25ms

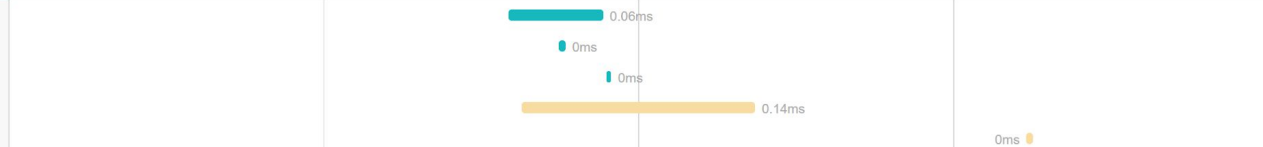
> Tags: http.bytes\_out = 469 | http.status\_code = 200 | http.hostname = 127.0.0.1:11114 | http.method = GET | http.uri = /rollup/85b75ea7-aa60-442e-afe4-d8eb69ff0cb8/nnt\_...

> Process: ip = 127.0.0.1

> Logs (2)

SpanID: 1

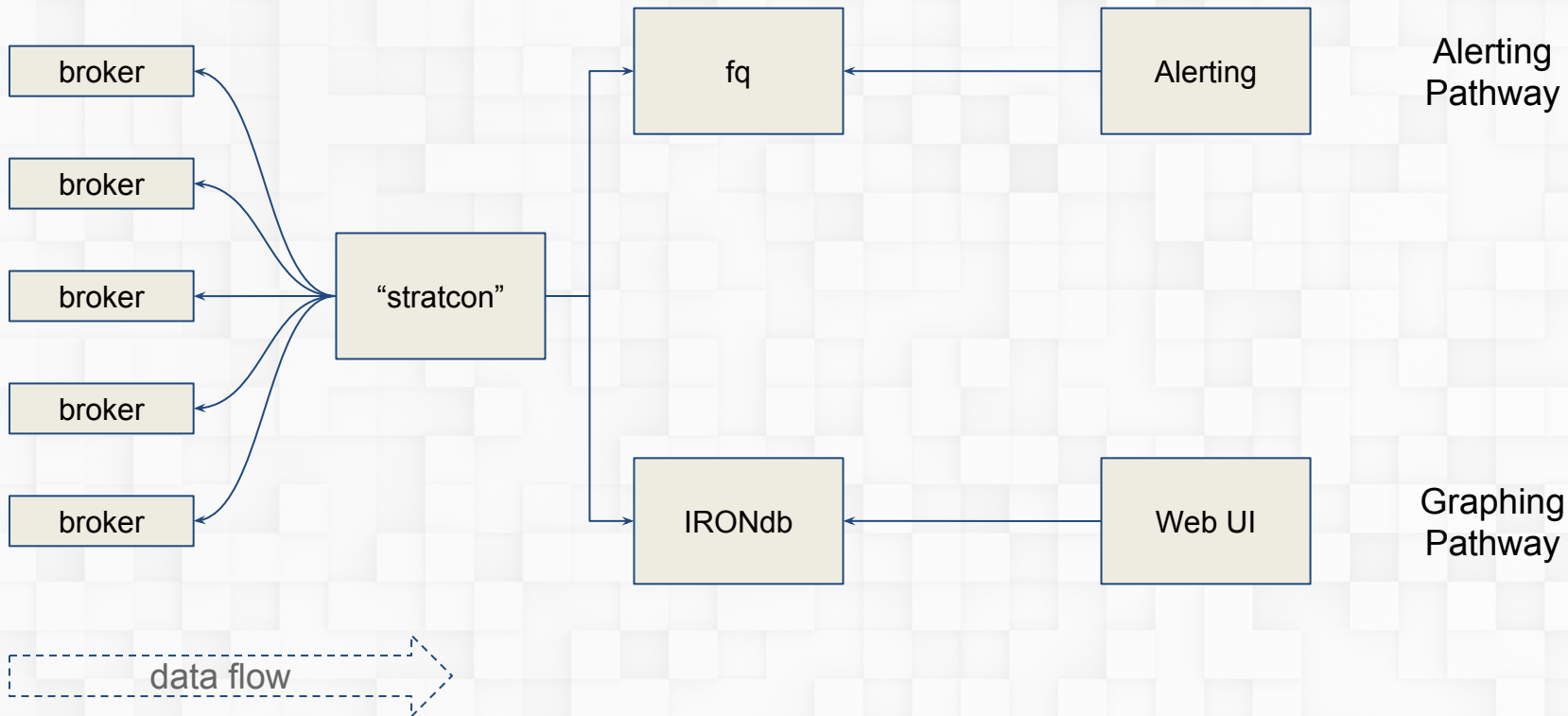
- snowth-00011114 eventer\_callback
- snowth-00011114 eventer\_callback
- snowth-00011114 eventer\_callback
- snowth-00011116 eventer\_callback
- snowth-00011116 eventer\_callback



CIRCONUS



# Circonus in 2010: No Analytics.





# Feature: Stream Analytics

Allow the user to:

- Alert on forecasted values
- Alert on histogram percentiles

...

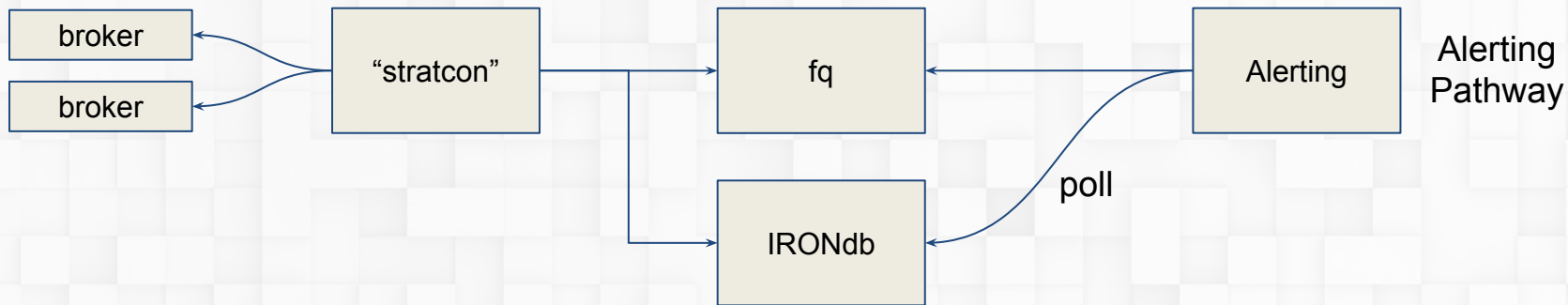
- Alert on custom analytics transformations (CAQL)

...

- Compute anomaly detection on all incoming metrics



# First Approach: Polling the DB



- \* Alerting component reaches out to IRONdb every minute
- \* Analytics transformations are computed on IRONdb from stored data

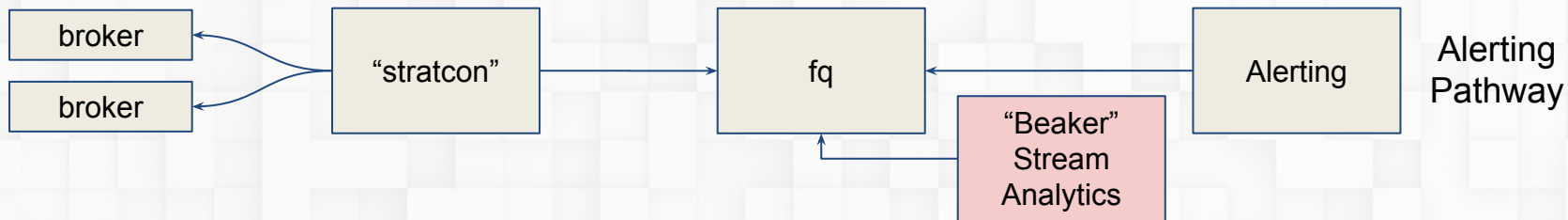
Tech: Java

Pros/Cons:

- + Simple
- High latency on metric data (seconds-minutes)
- Constant load on database
- Not suitable for alerting on all metrics



# Stream Analytics (2015): Beaker



- \* New component “Beaker” executing stream transformations
- \* reads & writes data to fq

Tech: C++/luajit

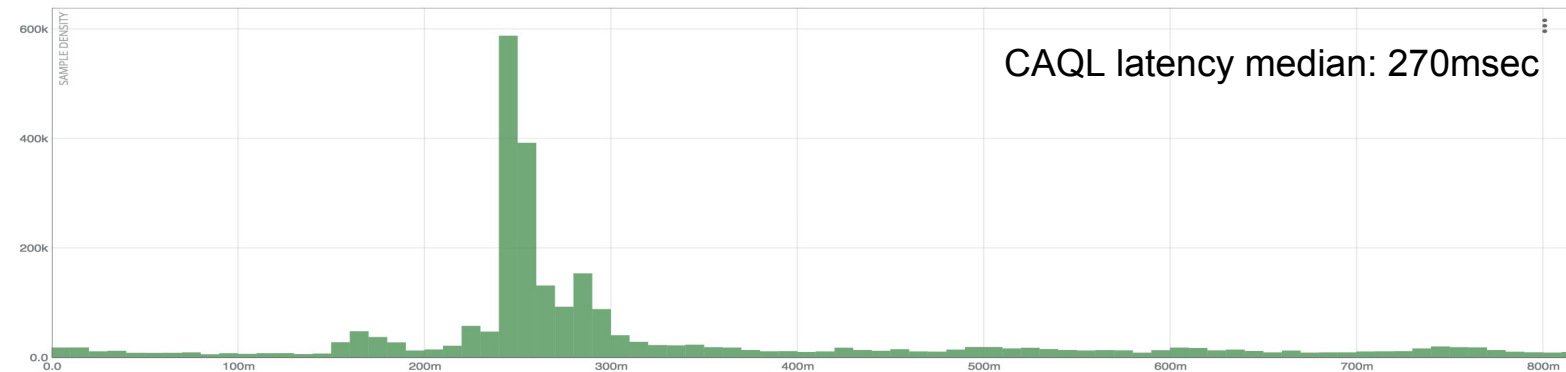
Pros/Cons:

- + Minimally invasive architecture
- + Avoid load on IRONdb
- + Reuse existing analytics code-base (luajit)
- ...

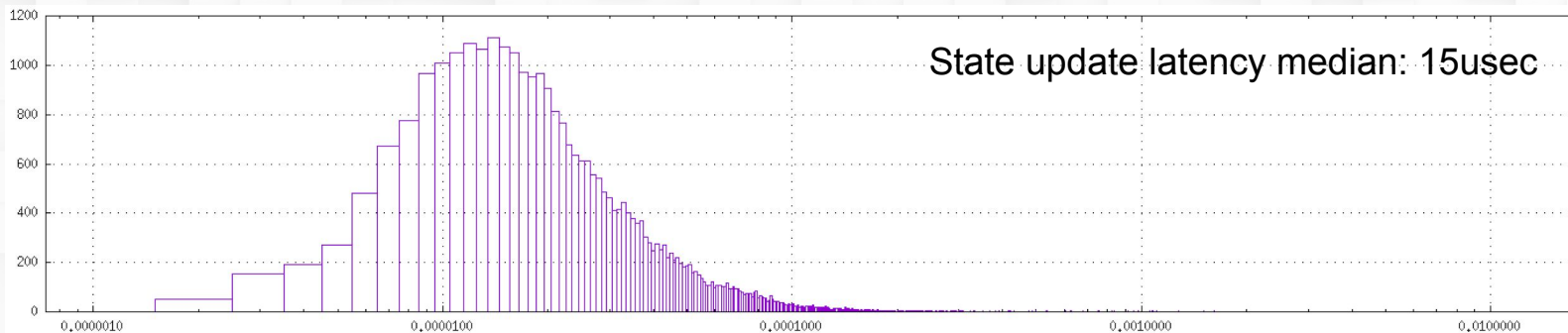


# Performance of Beaker vs IRONdb

## IRONdb: CAQL Query Performance



## Beaker: State Update Performance





# The Regrets of Beaker

- (1) Minimal Architecture Changes  $\neq$  Minimal Product Changes
- (2) Intransparent alerting behavior. Missing History.
- (3) Underestimate development effort to build operations tooling
- (4) Requirement Oversight: Cross metric aggregation



# Regrets of Beaker: Cultural Aspects 2015

New CEO: “When can I sell this?”

New CTO: “Don’t break stuff.”

New Team: “What is OmniOS?”





# Regrets of Beaker: Cultural Aspects 2016

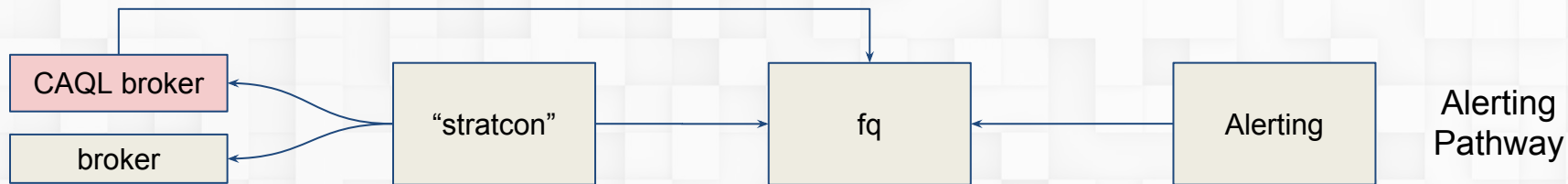
New Old CEO: “We need to redo this.”

Old Team: “Yeah. It’s not pretty.”

New New CTO: “Why not re-use our broker tech?”



# Stream Analytics (2016): CAQL Broker



- \* Repurpose existing broker tech for stream analytics
- \* Publish results as metrics into the system

Tech: C/mtev/reconnoiter/luajit

Pros/Cons:

- + Reuse existing analytics code-base (luajit)
- + Reuse existing server framework inc. ops tooling (libmtev)
- + No new concepts for the UI
- High observability needs



# CAQL Broker: Observability is Key

- CAQL state is kept over days and weeks
  - Many issues can't be replicated in dev
- > Need to understand the system while it's running!
- Extensive logging of state changes
  - Instrumentation (/stat.json, RED, USE)
  - Introspection capabilities (telnet, web)

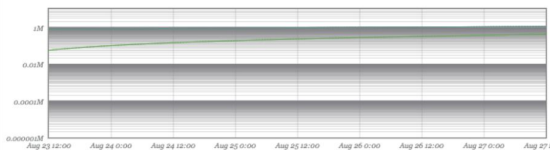


# CAQL Broker: Monitoring

Uptime in Days

# 12.6

Uptime



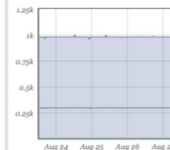
Active Checks

# 965

Messages IN



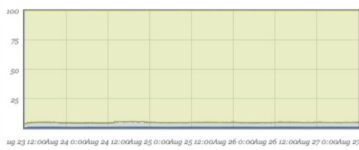
Metrics OUT



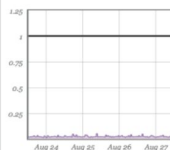
CPU Utilization %

# 4

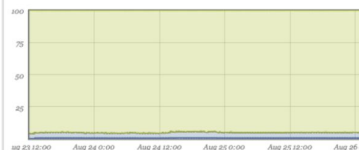
caqlbroker1-gcp-ia CPU Utilization



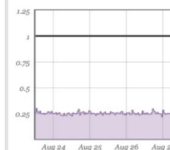
CPU Saturation



caqlbroker2-gcp-ia CPU Utilization



caqlbroker2-gcp-ia System Load



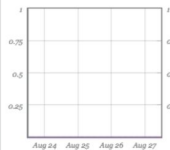
Out of Memory in

# >17.0 HRS

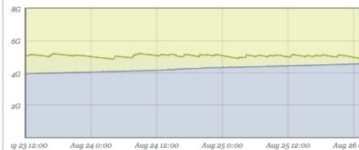
Memory Utilization



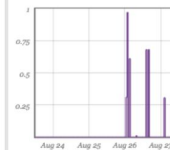
Memory Saturation



caqlbroker2-gcp-ia Memory Utilization



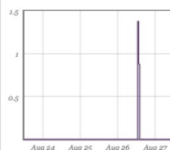
caqlbroker2-gcp-ia Memory Satur



Network Utilization



Network Saturation



caqlbroker2-gcp-ia Network Utilization

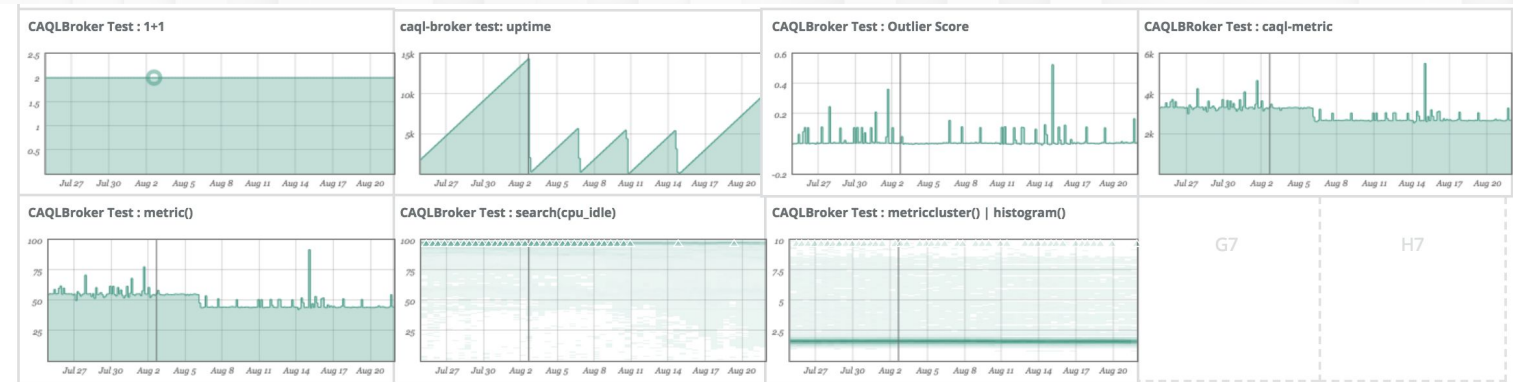


caqlbroker2-gcp-ia Network Satur





# CAQL Broker: Self Monitoring





# Log Analysis: “Poor man’s Splunk / ELK”


```
./harvest.sh caqlbroker{1..5}.dev.circonus.net
make log.sqlite

# Top-5 slow queries
sqlite3 log.sqlite ".mode line" \
  'select duration, query from log order by duration desc limit 5'

# Duration histogram
sqlite3 log.sqlite ".mode csv" 'select duration from log' \
  | feedgnuplot --histogram 0 --binwidth 0.1 --with boxes
```



# CAQL Broker: Check inspection

 Circonus CAQL Broker [Overview](#) [Checks](#) [Internals](#)

ID	Name	Module	Period	Status
5250ee71-d30c-4c3d-8dc9-96d6239d0730	c_1906_233206::caql	caql	q_caql	1m/10s -M
6971a639-d07d-466f-8dfb-f984c01703a0	c_1_168429::caql	caql	q_caql	1m/10s -M

### Check Details

**ID** 6971a639-d07d-466f-8dfb-f984c01703a0

**Target** q\_caql

**Name** c\_1\_168429::caql

**Module** caql

**IP**

**Period** 1m

**Timeout** 10s

**Filterset** filterset\_23419

**Last run** 2018/07/23 15:14:21 (3m25s ago)

**Next run** 2018/07/23 15:15:21

**Status** available good

### Check Config

```
query metric:counter("e0cb68de94d0-42a7-a46e-dd808f460ce2","exchange | delay(0,1d,2d,3d,4d,5d,6d) | outlier:std_score(trim=1, normalize=2)
```

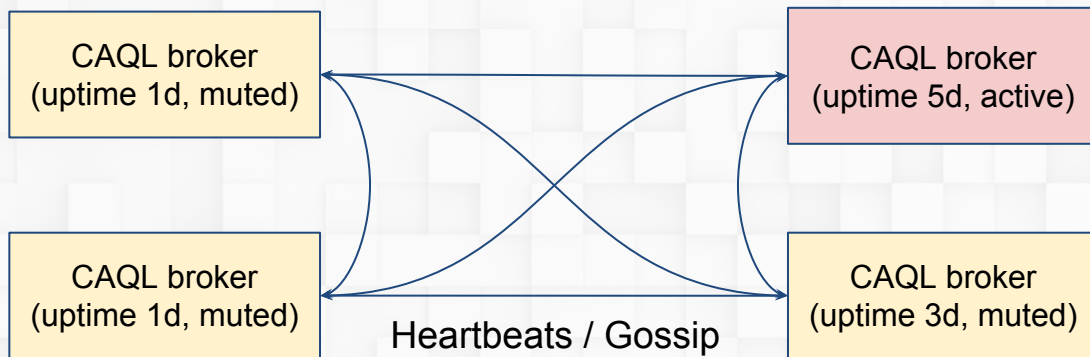
### Metrics

Name	Type	Value
output[1]	double	0.000000000000e+00



# CAQL Broker: High Availability

- \* Oldest node sends data
- \* Checks are replicated among cluster nodes
- \* Versioning of checks is operator provided
- \* Network partitions result in double submissions (OK)







# HA is a Game Changer for Ops

Without HA:

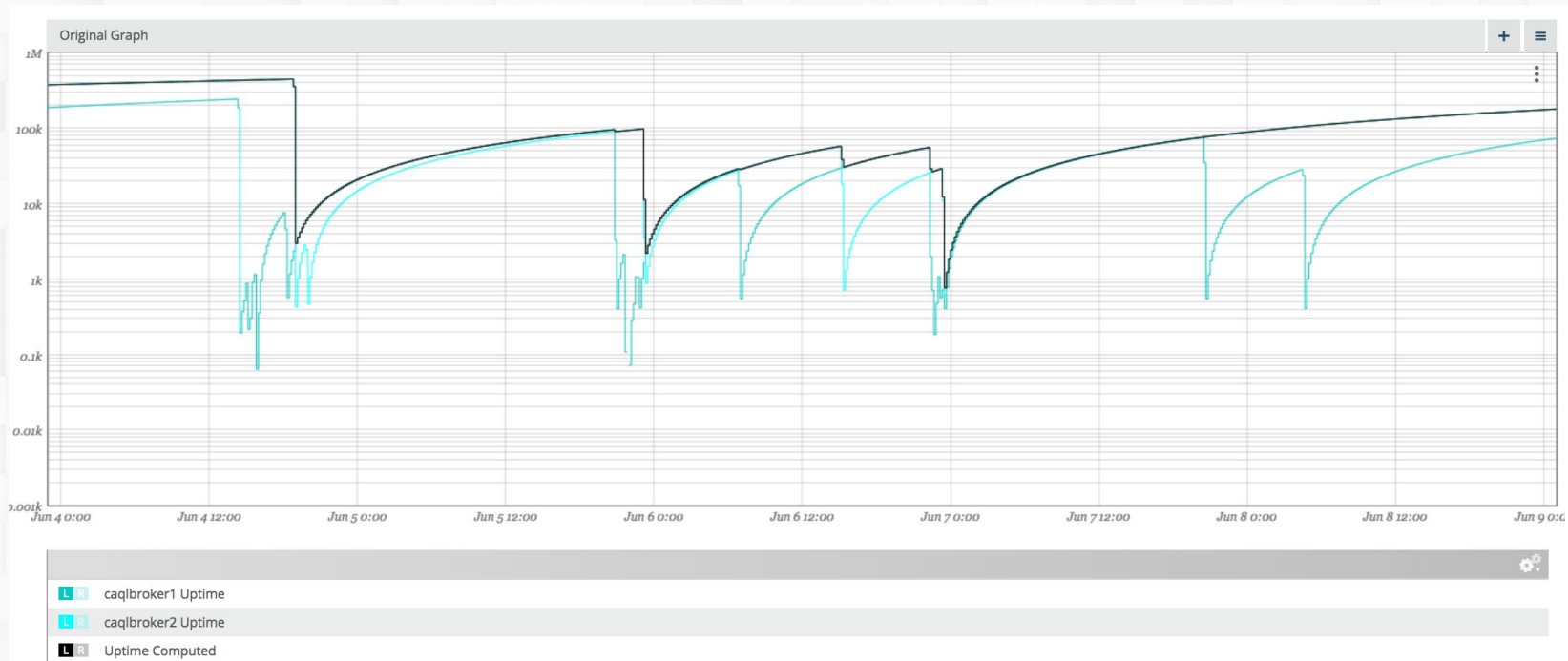
- Every update means downtime (1-3Min)
- Immediate rollback if any issue is hit (<5M)
- Try to reproduce in dev/stage

With HA:

- Deploy to standby node first
- Can debug stuff in prod for a good hour with low risk

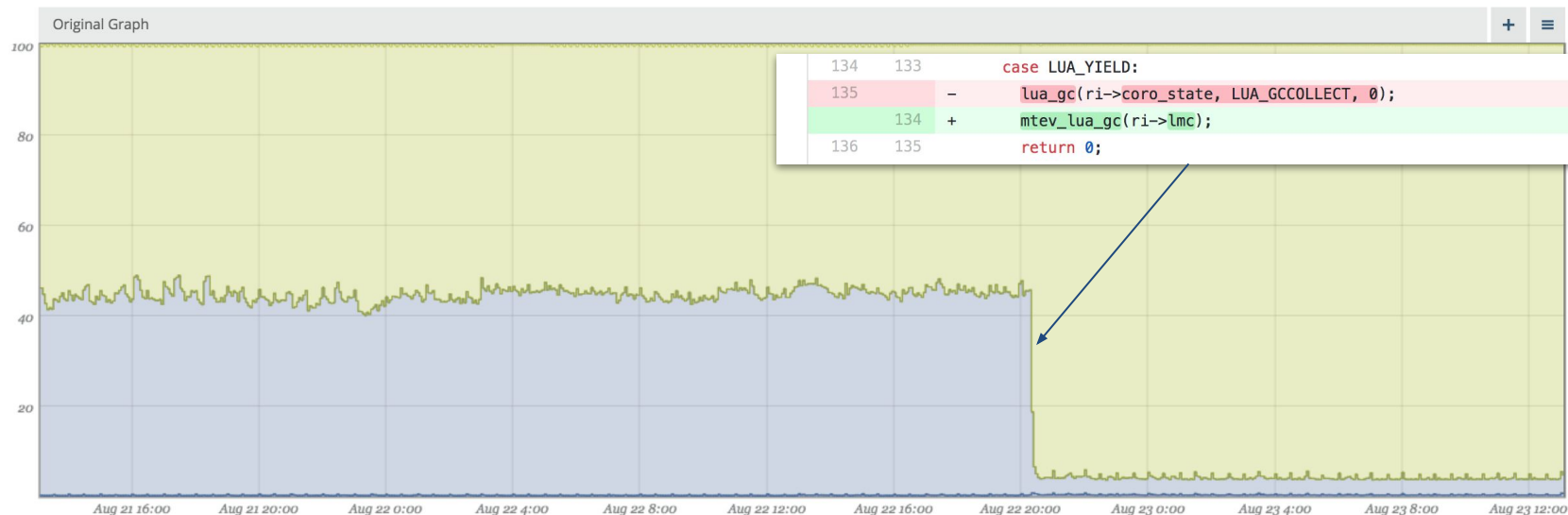


# CAQL Broker: Deployments gone wrong





# Reminder: Don't run GC ... ALL THE TIME.

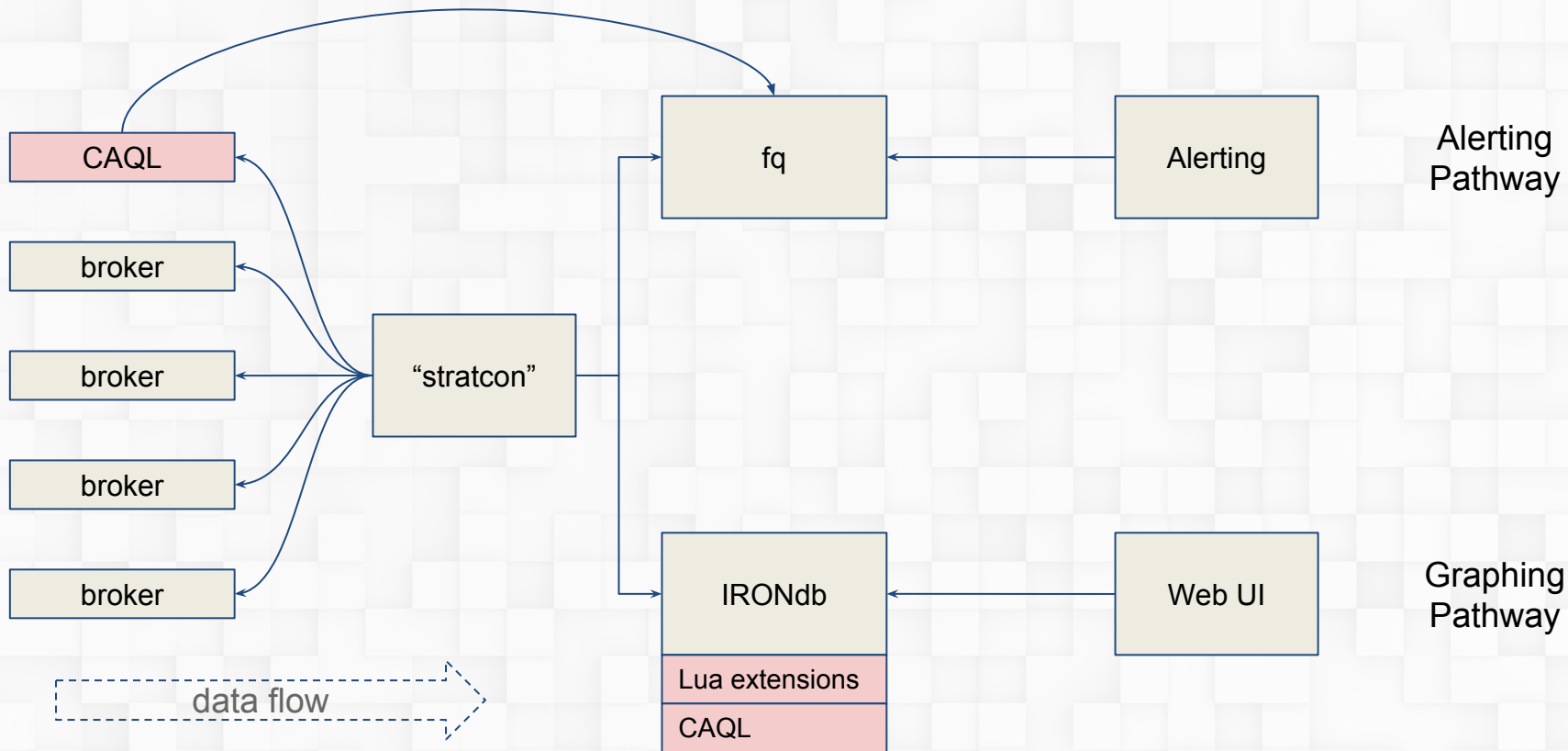


Aug 23 2018, 13:08 (4M)

cpu `kernel	0.79
cpu `intr	0.12
cpu `user	4.74
cpu `wait_io	0
cpu `idle	94.15



# Cirronus Analytics Architecture 2018





# Lessons Learned

- With time as a real imposed constraint, your mistakes will include mistakes you knew would bite you. This is technical debt leveraged for time-to-market.
- Once you've been served a collection notice, focus on operability with a higher priority than correctness; you're unlikely to get either perfect & operability will pay dividends sooner.
- Instrumenting systems and retaining performance data is the only way to know you've actually succeeded, so get on that.
- Build systems resilient enough to allow risky behavior in production up to and including developing in production.