NETFLIX

Atlas

Scalable time-series management

Brian Harrington

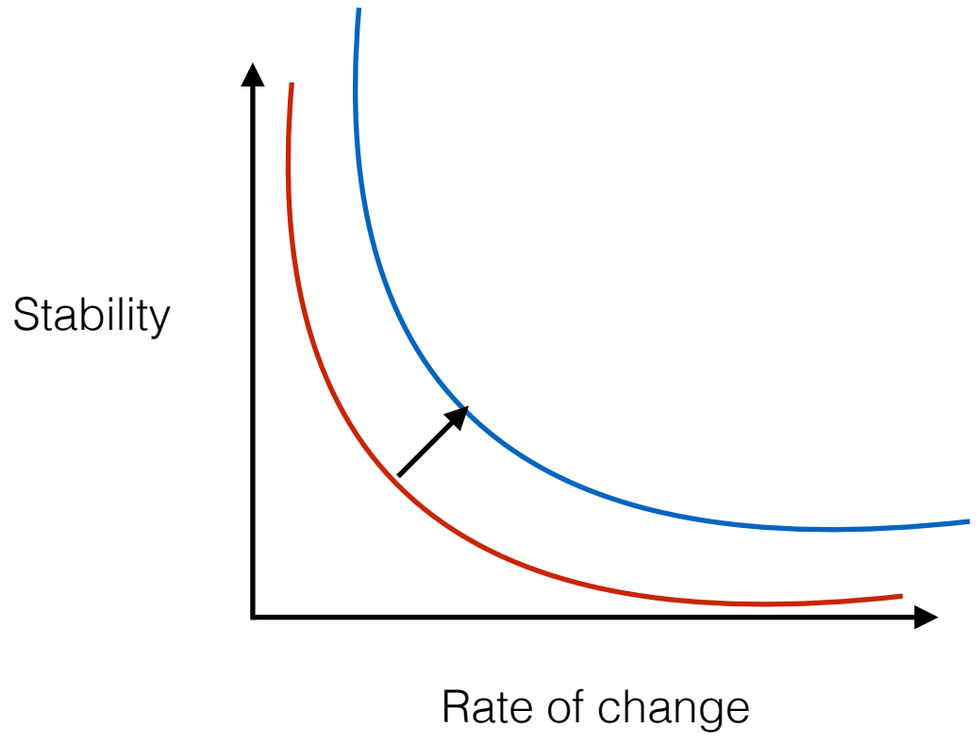
December 16th, 2014

About me

- Brian
  - 4 years in May
  - Mostly focus on backend
- Insight engineering
  - Enables and drives continuous improvement of real-time operational insight into our customer experience across operational environments.

Our role

- Prevention
  - Is my system working?
  - Test > Canary > Prod
- MTTD - mean time to detect
- MTTR - mean time to resolution

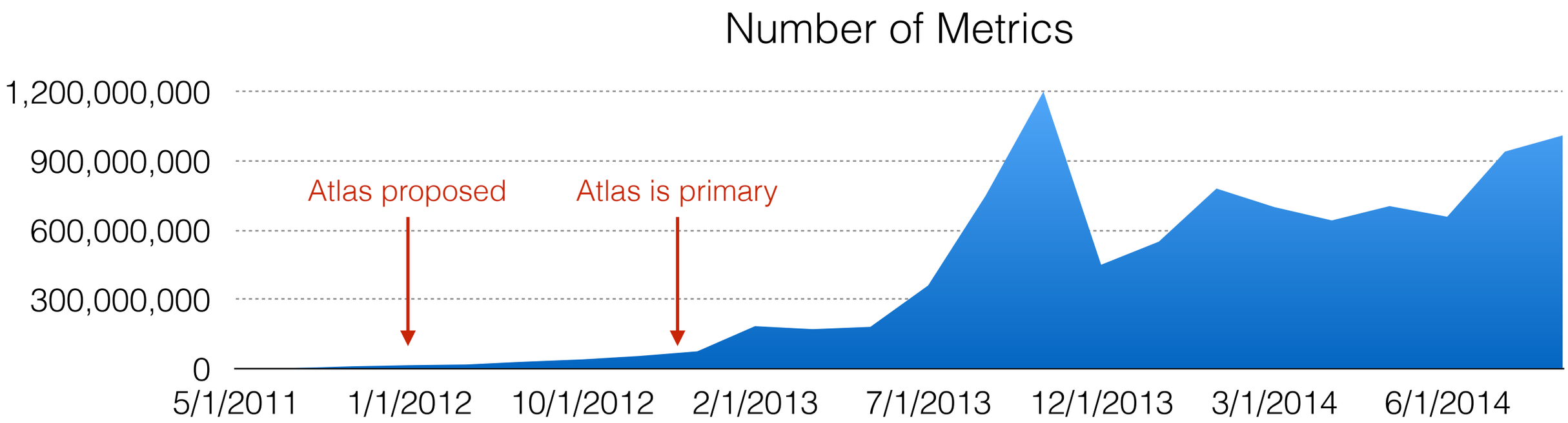


Netflix likes monitoring

- Hadoop, Hive, Spark, …
- CloudWatch, Boundary, AppDynamics, Teradata, SumoLogic, …
- JMX, SNMP, sar, …
- Atlas, Chronos, Edda, Mantis, Turbine, Chukwa, …

What is Atlas?

- Atlas is the system Netflix uses to manage dimensional time series data for near real-time operational insight.
- Metric volume has doubled almost every quarter since I started. We have grown from 2M to 1.2B.



Insight Categories

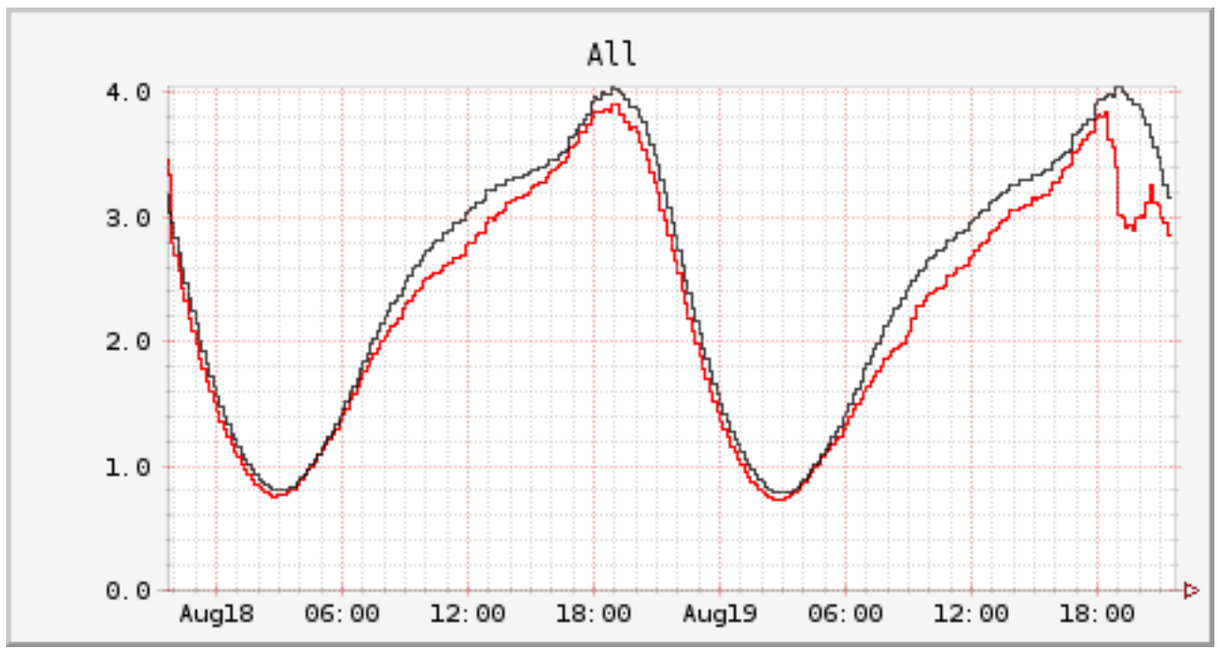
- Operational vs Business intelligence
  - Operations: What is happening now?
  - BI: What are the trends over time?
- Time series vs Events
  - Do you need to query for a particular event?
  - Or just see a summary of events over time?

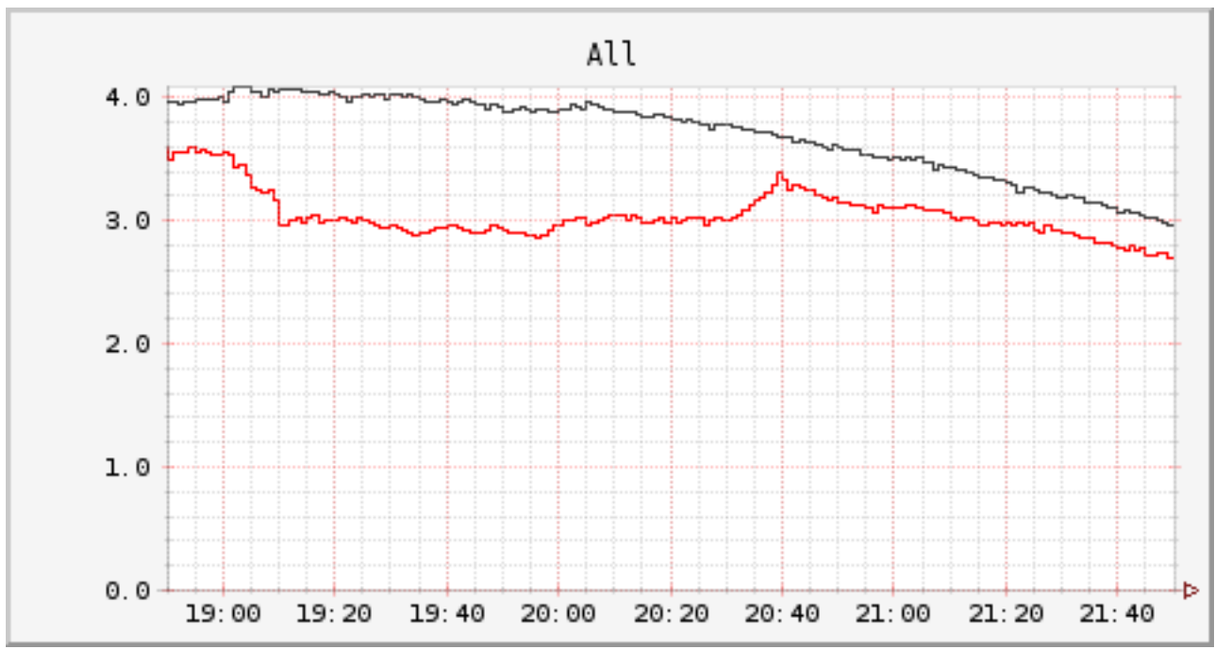
Where we started

- Epic
  - Predecessor to Atlas
    - CGI script in front of RRDTool
    - MySQL for metadata and RRD files on disk
    - Data center
    - Falling over at around 2M metrics

Requirements

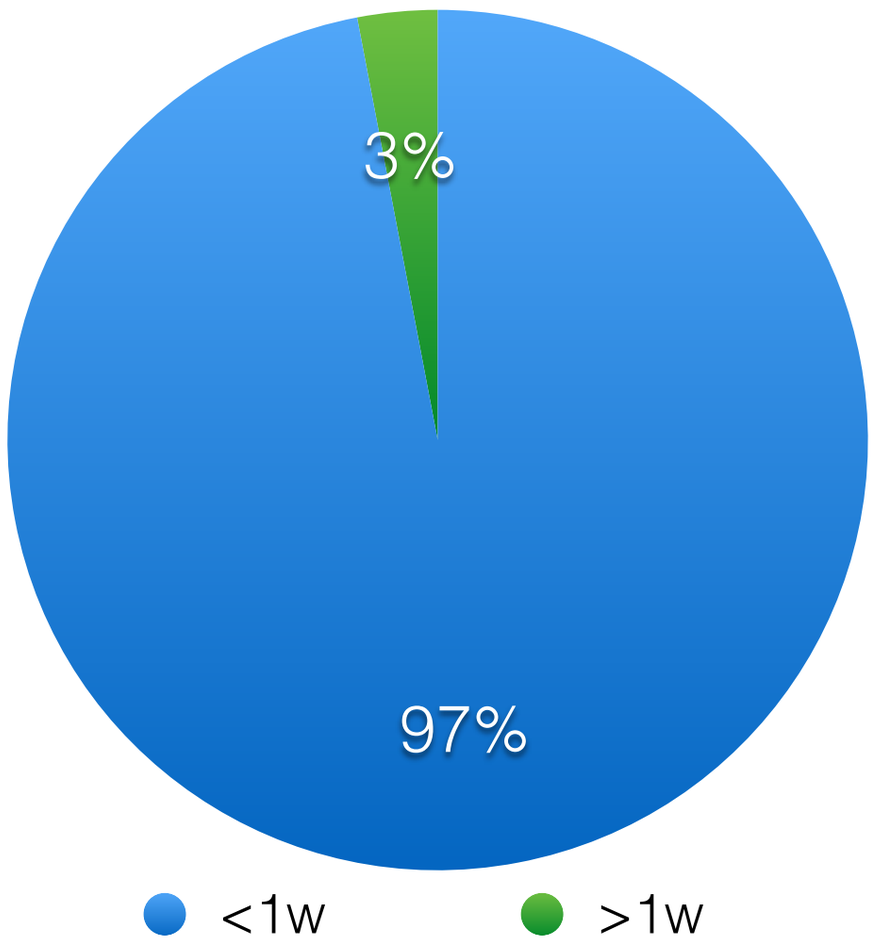
- Don’t lose functionality
- Retention: 2w + a few days
- Scale
- Query explicitly based on dimensions



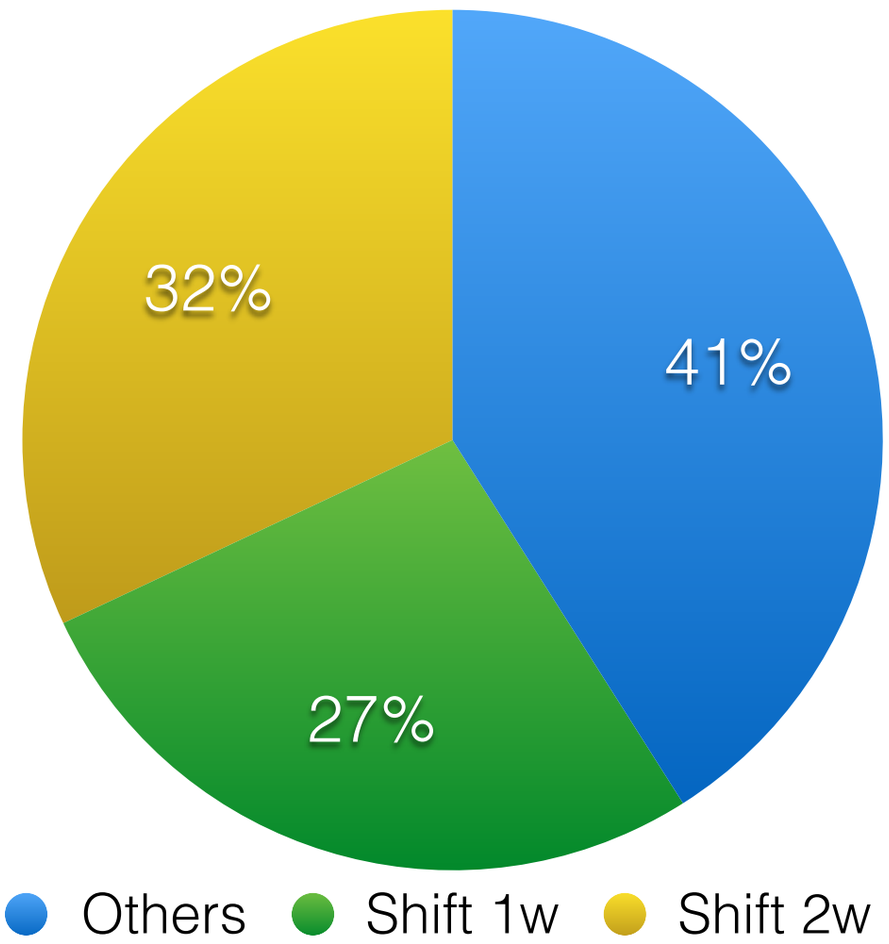


Amount of time

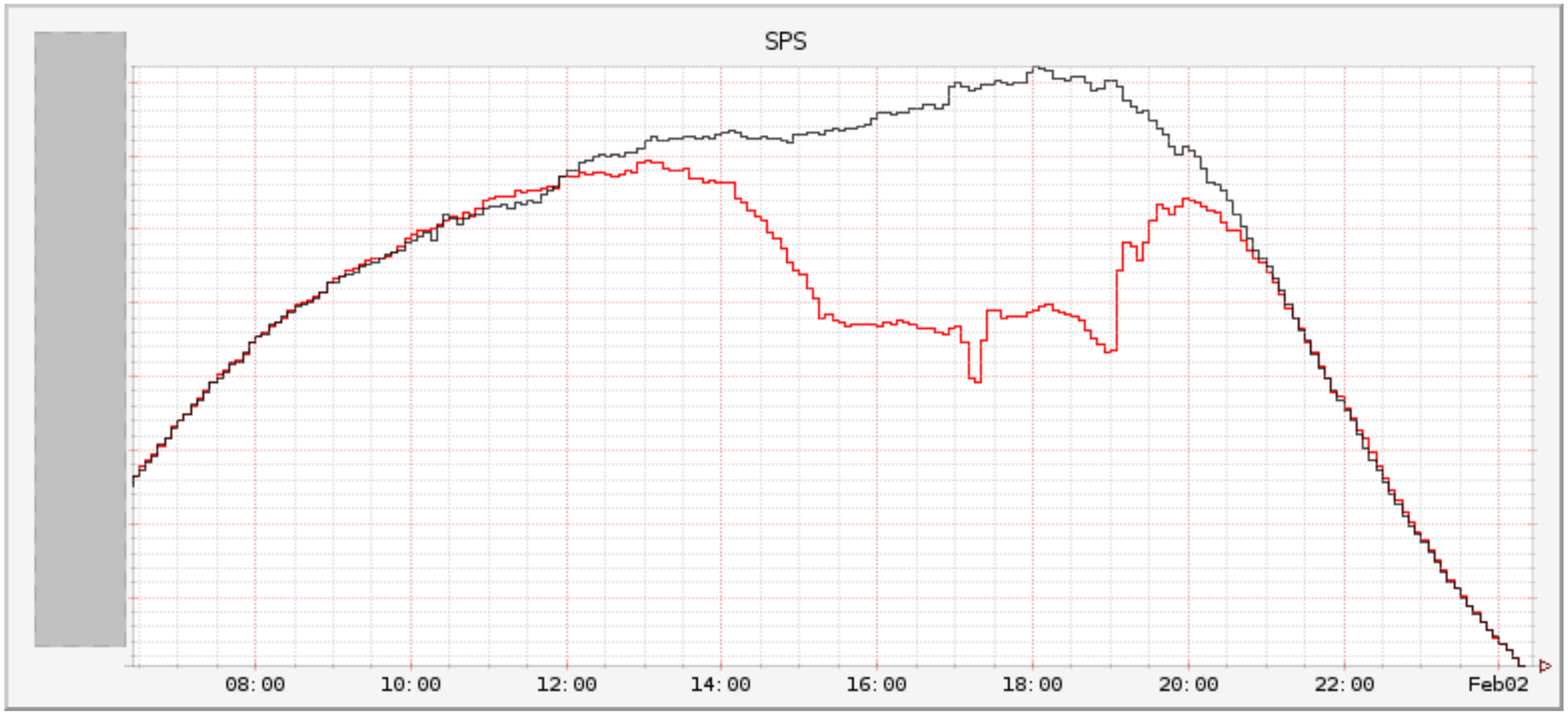
Time range for graph requests



Time range for graph requests with shifts



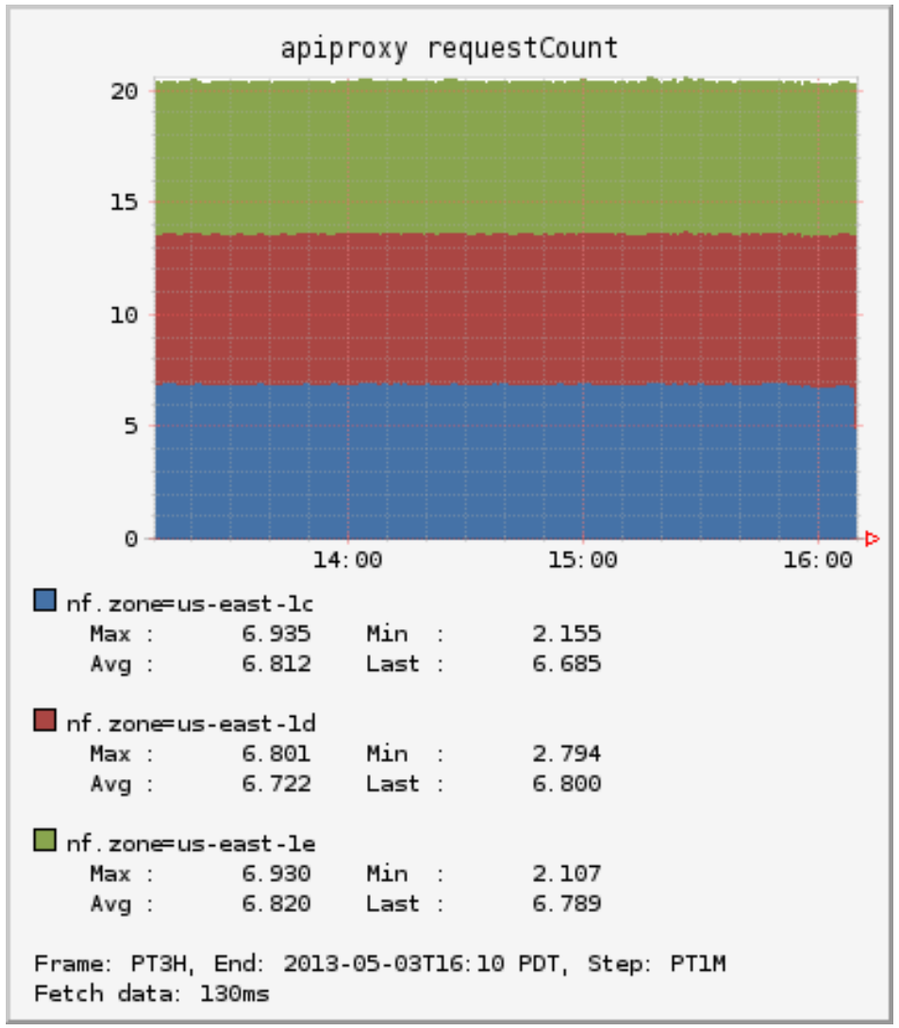
Any guesses?

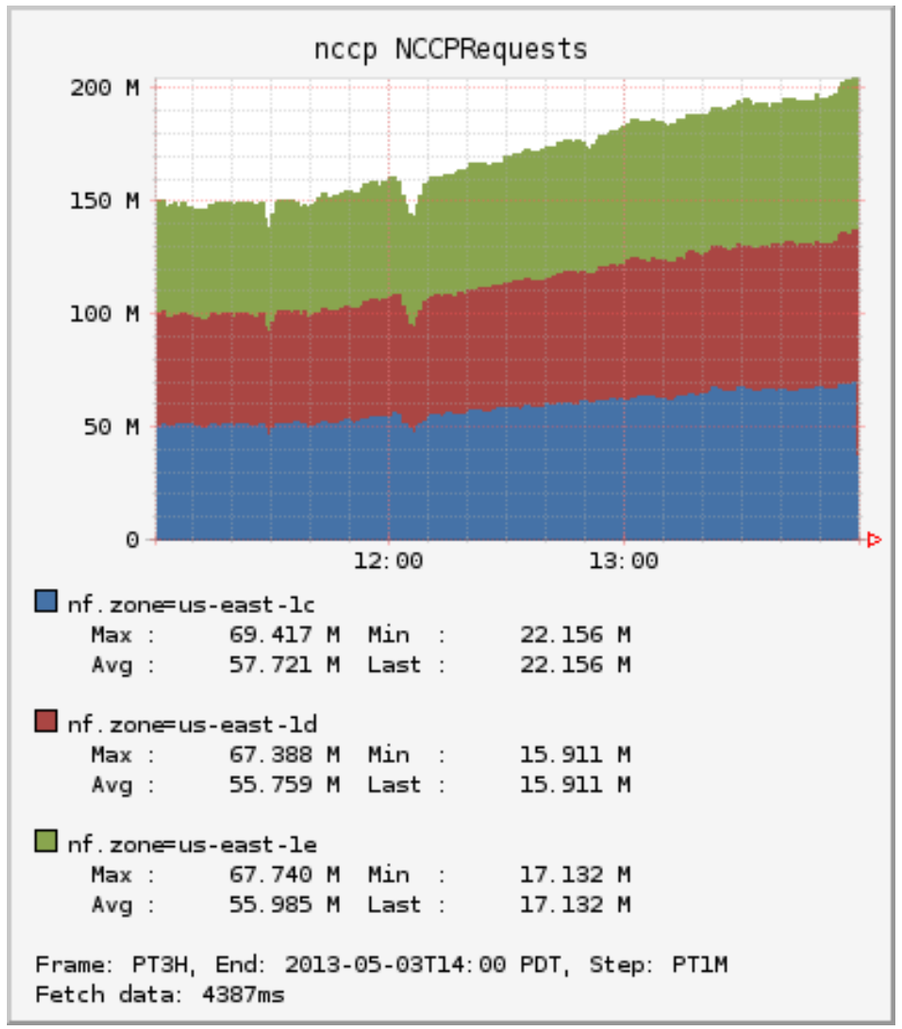


Scale

- Define scalable?
    - We can throw hardware at it
- Write volume
- Read volume

How much input data?



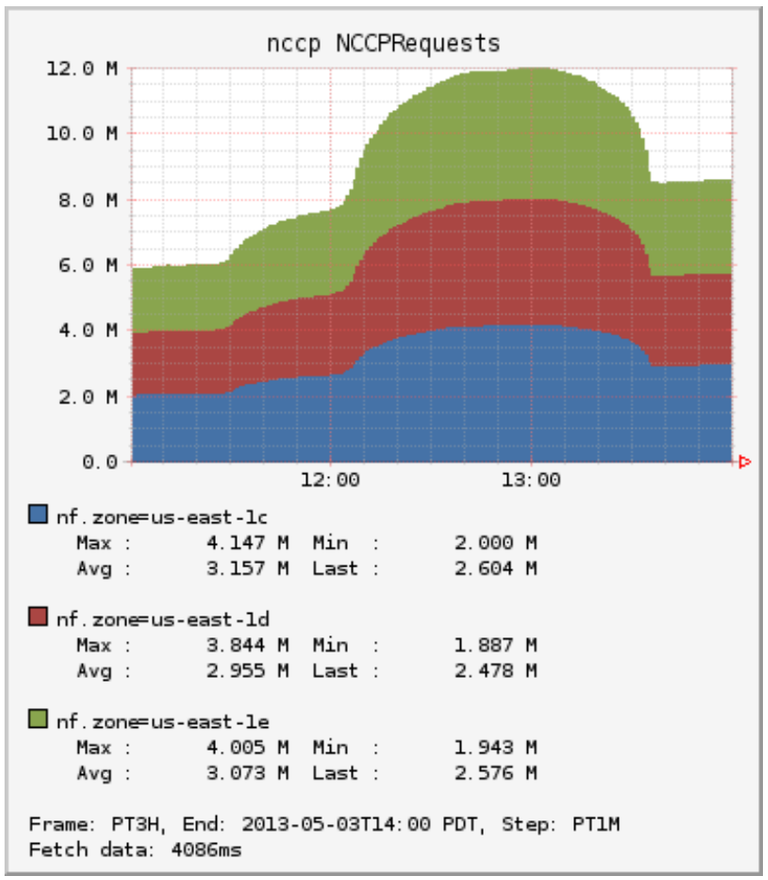


Graph 1: apiproxy

- Number of time series matched: 206
- Number of blocks: 824
- Number of input data points: 37,080
- Number of output data points: 540
- Number of output lines: 3

Graph 2: nccp

- Number of time series matched: 12M
- Number of blocks: 48M
- Number of input data points: 2.16B
- Number of output data points: 540
- Number of output lines: 3



Why dimensions?

- Example metric name
  - com.netflix.eds.nccp.successful.requests.uiversion.nccprt-authorization.devtypid-101.clver-PHL_0AB.uiver-UI_169_mid.geo-US
- How do you query this?

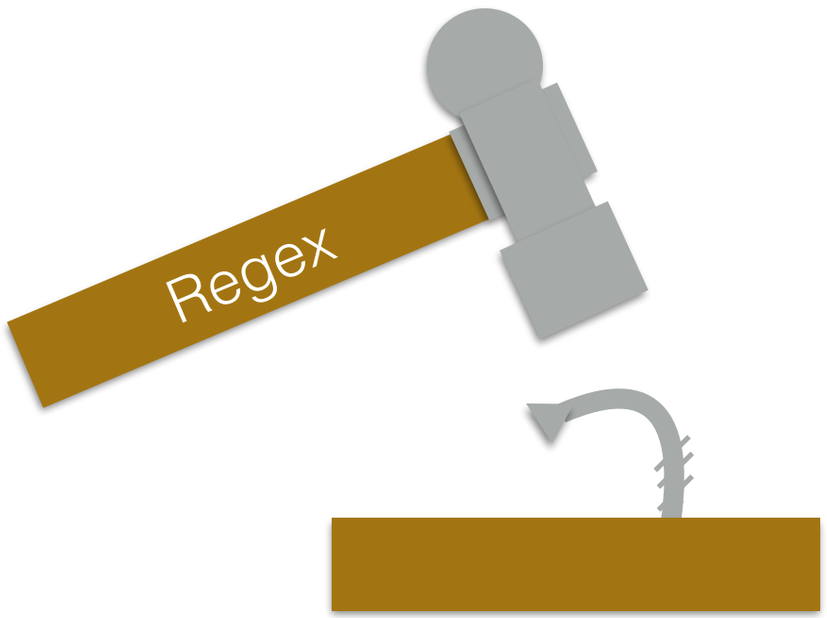
Why dimensions?

- Example metric name
  - com.netflix.eds.nccp.successful.requests.uiversion.nccprt-authorization.devtypid-101.clver-PHL_0AB.uiver-UI_169_mid.geo-US
- How do you query this?

Key	Value
name	nccp.successful.requests
nccprt	authorization
devtypid	101
clver	PHL_0AB
uiver	UI_169_mid
geo	US

Why dimensions?

- Example metric name
  - com.netflix.eds.nccp.successful.requests.uiversion.nccprt-authorization.devtypid-101.clver-PHL_0AB.uiver-UI_169_mid.geo-US
- How do you query this?



Key	Value
name	nccp.successful.requests
nccprt	authorization
devtypid	101
clver	PHL_0AB
uiver	UI_169_mid
geo	US

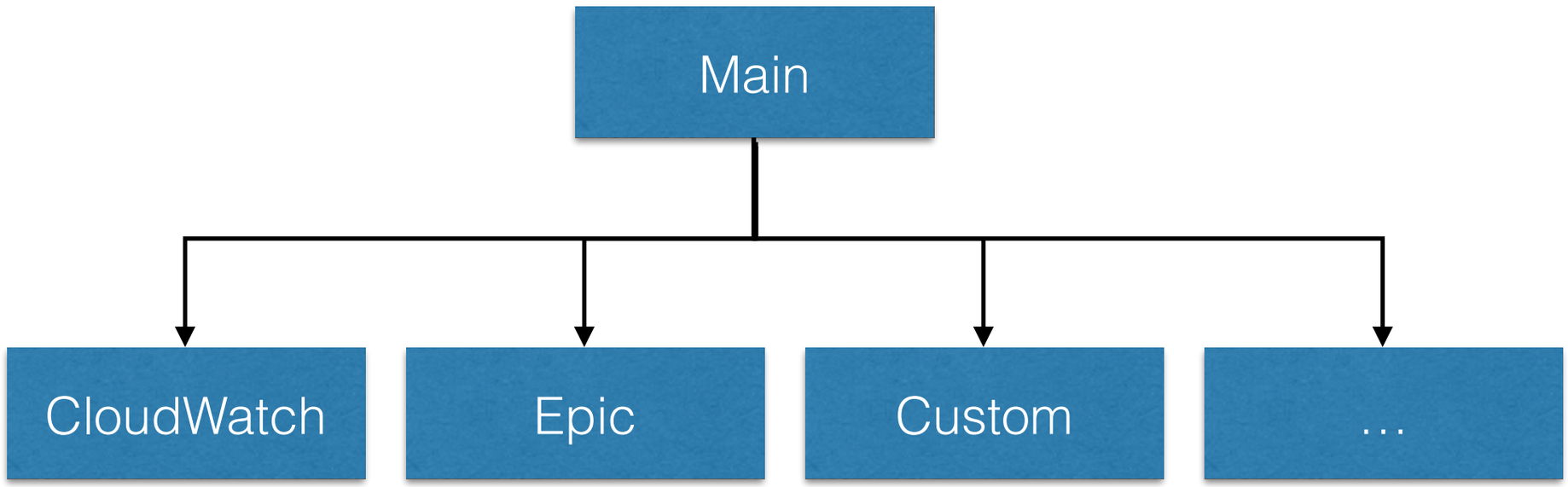
Perspective

- Service owner
- Library owner
- UI team
- CDN team managing caches in ISPs
- Cross-functional
  - Performance and capacity team
  - Site reliability
- Exploratory

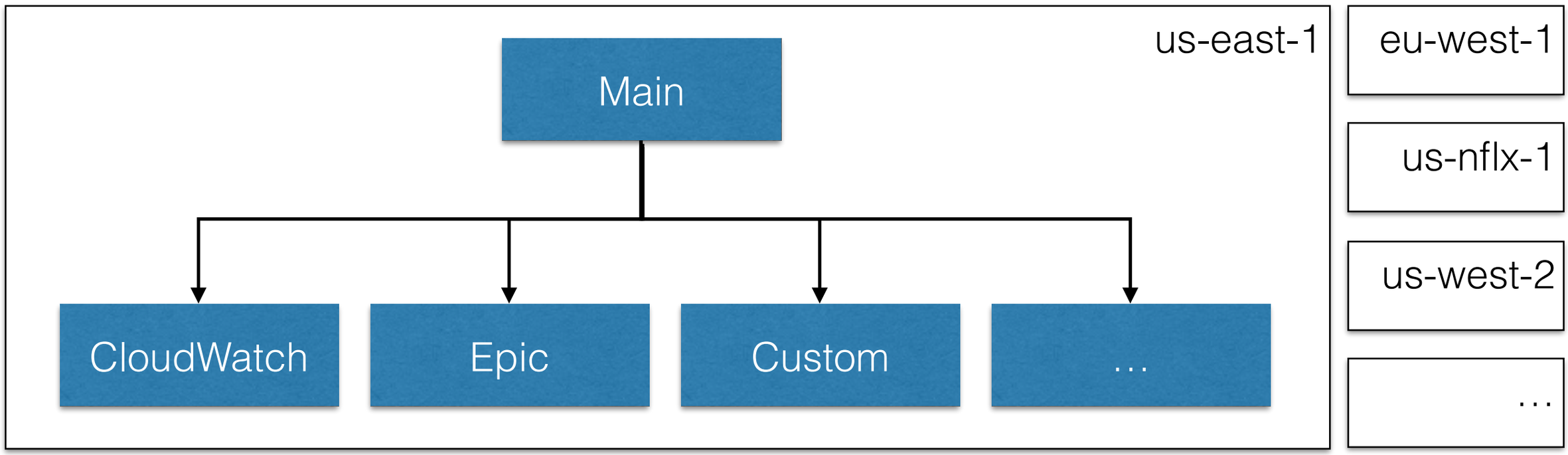
Problem 1: parity

- Normalization and consolidation
- Flexible legends, scale independently of chart
- Math, in particular handling of NaN values
- Holt-Winters
- Visualization options
- Deep linking

General query layer

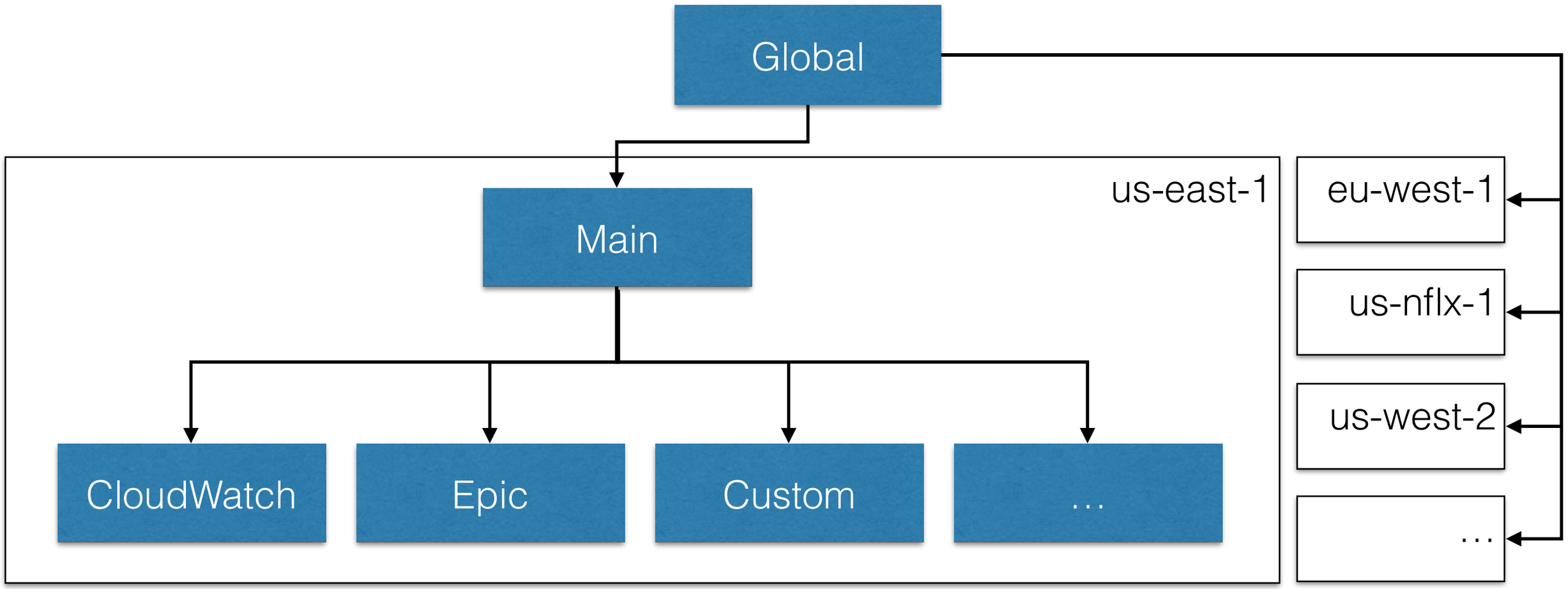


General query layer



Island model: geographic regions should be isolated

General query layer



Island model: geographic regions should be isolated

Stack language

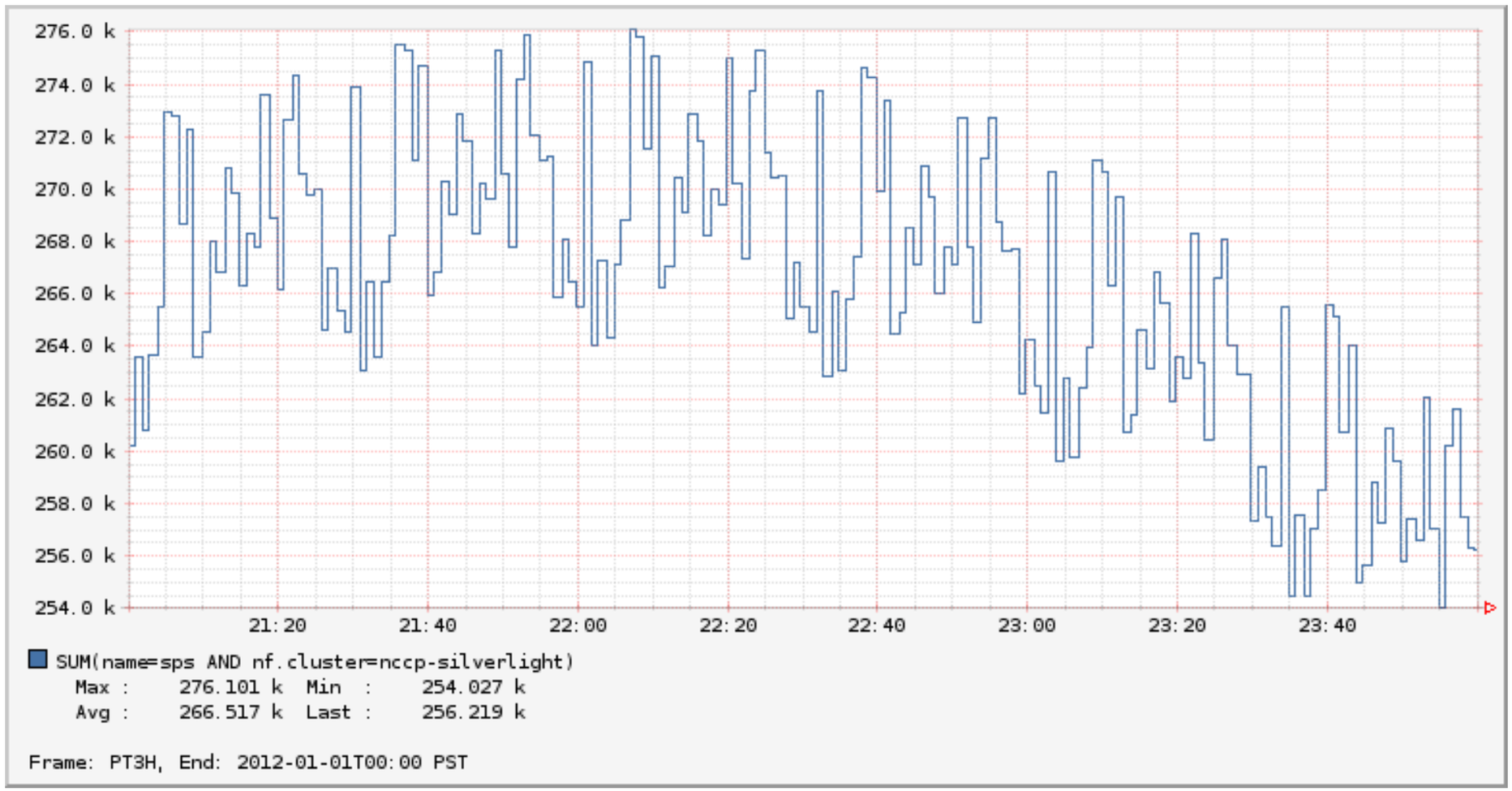
- Embedding and linking is import to us
- GET request
- URL friendly stack language
  - Few special symbols (comma, colon, parenthesis)
  - Easy to extend
  - Usability
- Basic operations
  - Query: and, or, equal, regex, has key, not
  - Aggregation: sum, count, min, max
  - Consolidation: aggregate across time
  - Math: add, subtract, multiply, etc
  - Boolean: and, or, lt, gt, etc
  - Graph settings: legends, area, transparency

Stack language summary

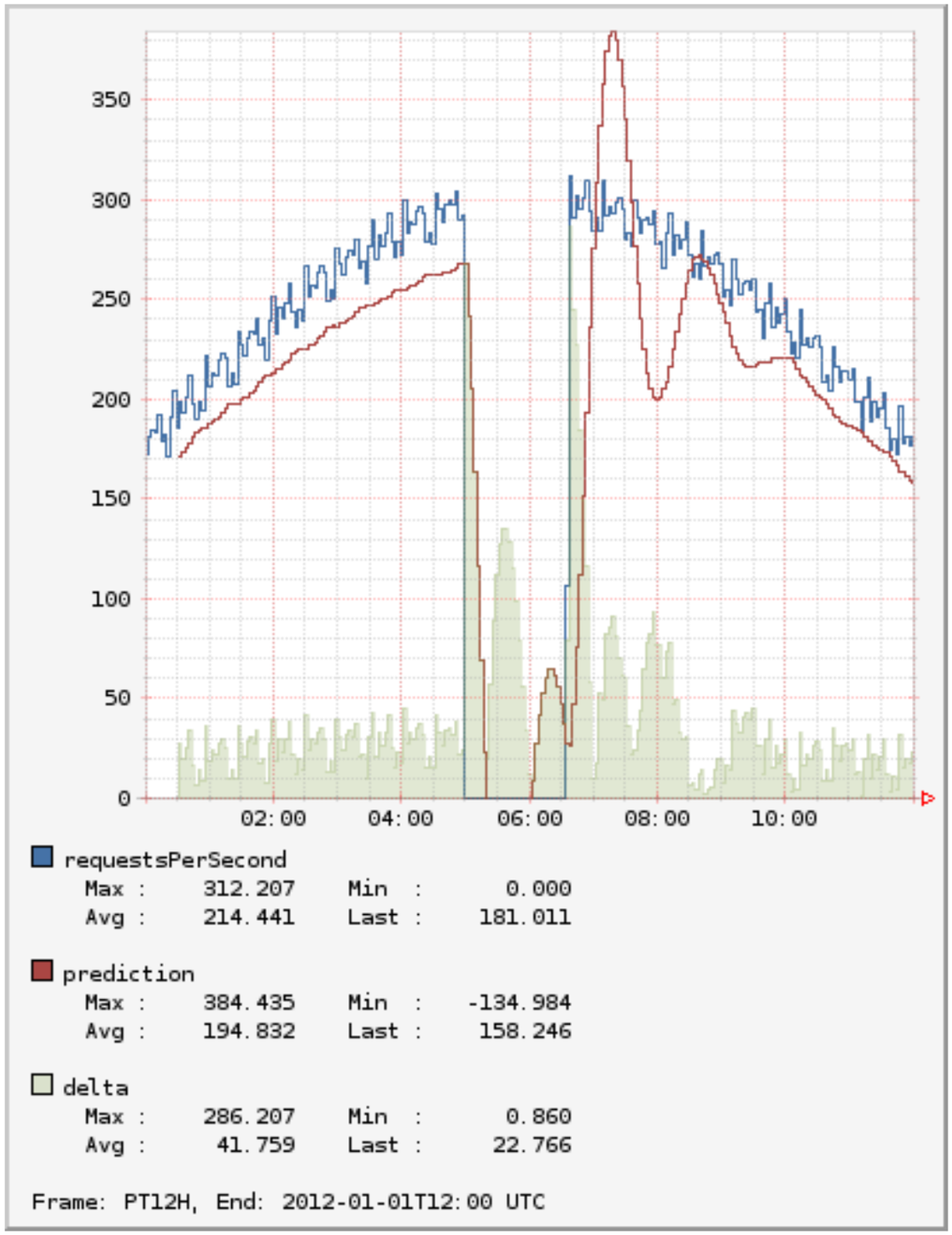
- Punctuation: comma, colon, and parenthesis
- Operations start with colon
- Comma is the separator
- Parenthesis used for lists
- Example:
  - nf.cluster,discovery,:eq,(,nf.zone,),:by
  - select * where nf.cluster == “discovery” group by nf.zone

Simple graph

/api/v1/graph?
e=2012-01-01T00:00&
q=name,sps,:eq,nf.cluster,nccp-silverlight,:eq,:and,:sum

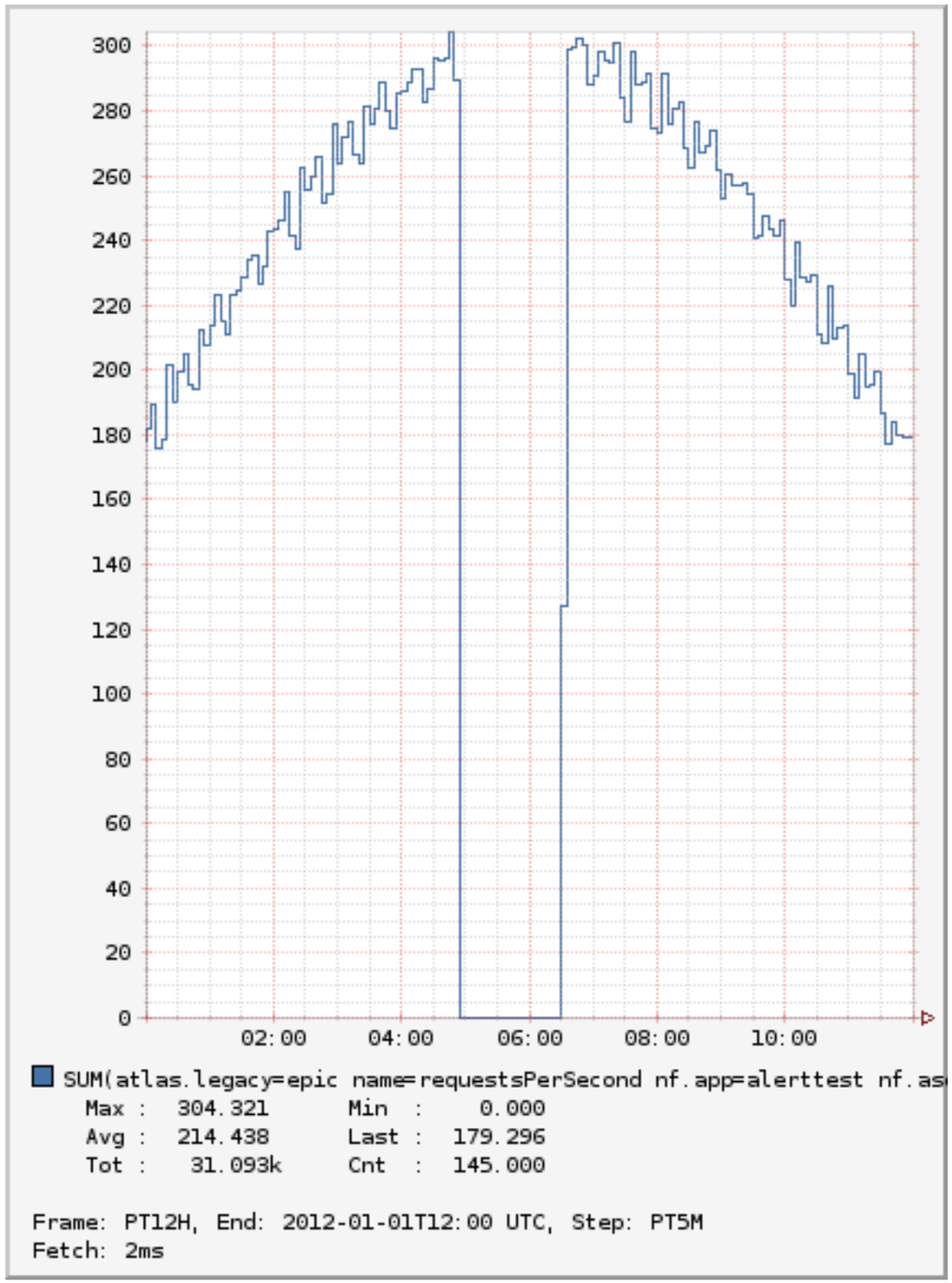


More complex graph



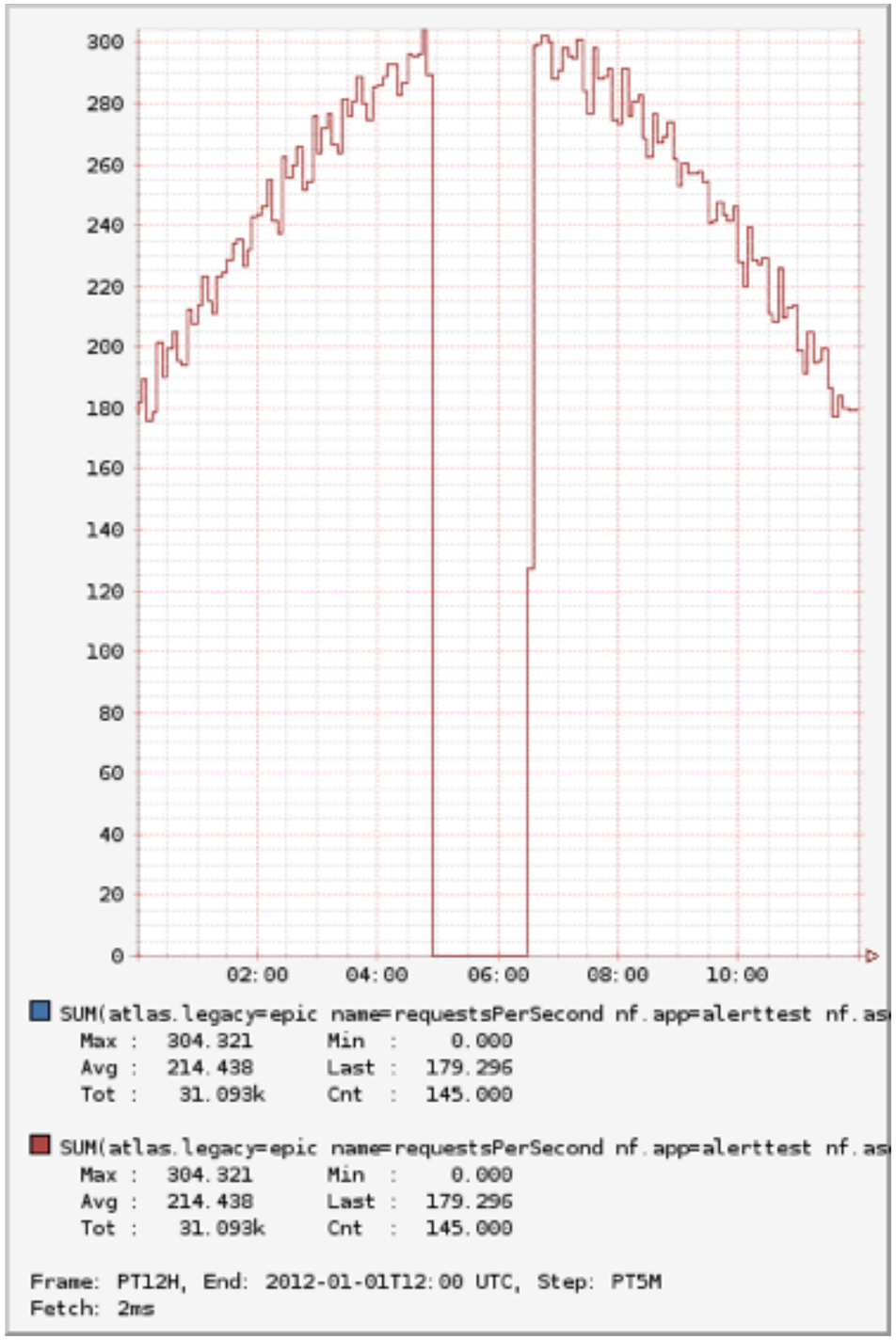
More complex graph

Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,



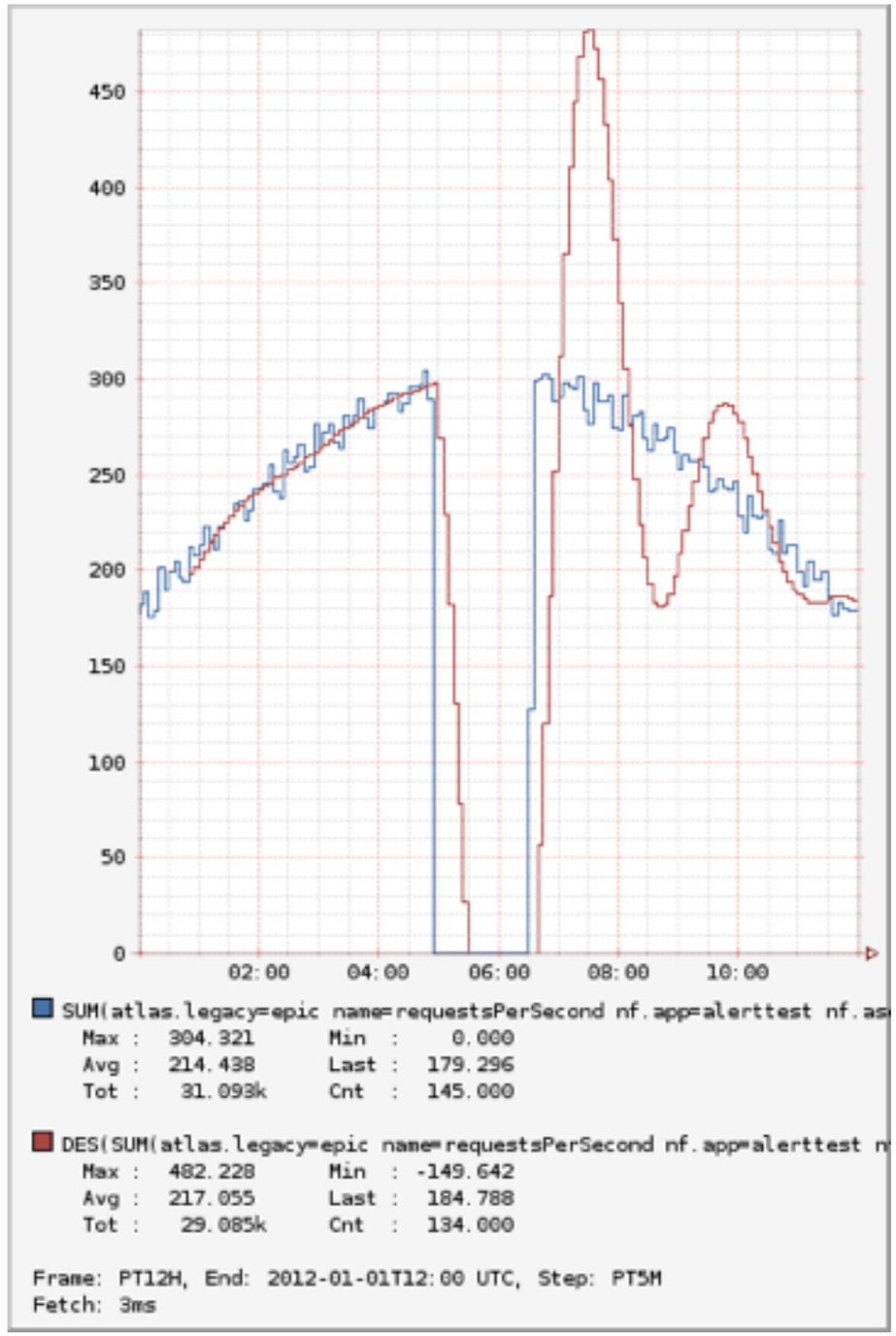
More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,
```



More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,
```

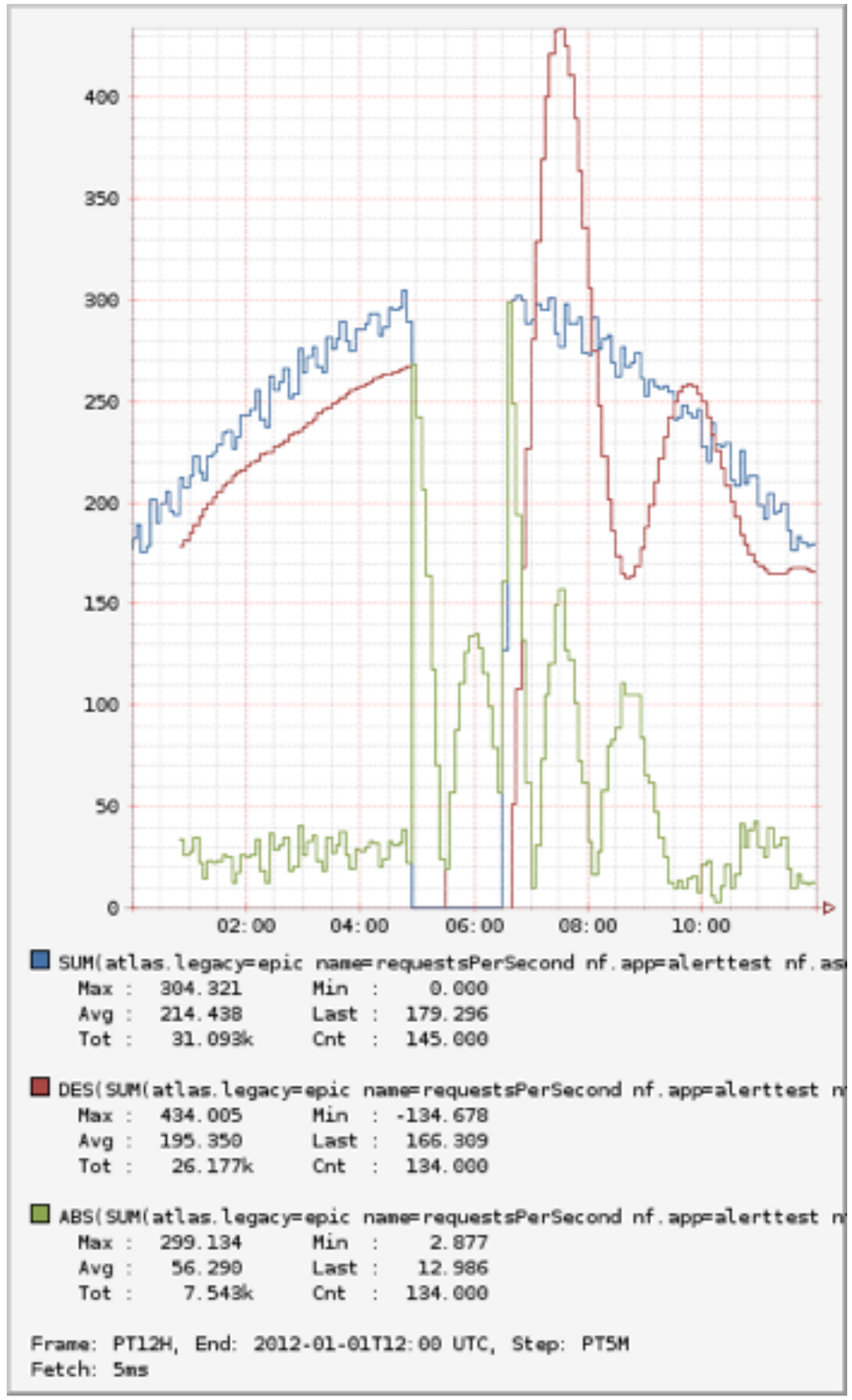


More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,

# Mutliply, used to set threshold
0.9,:mul,
```

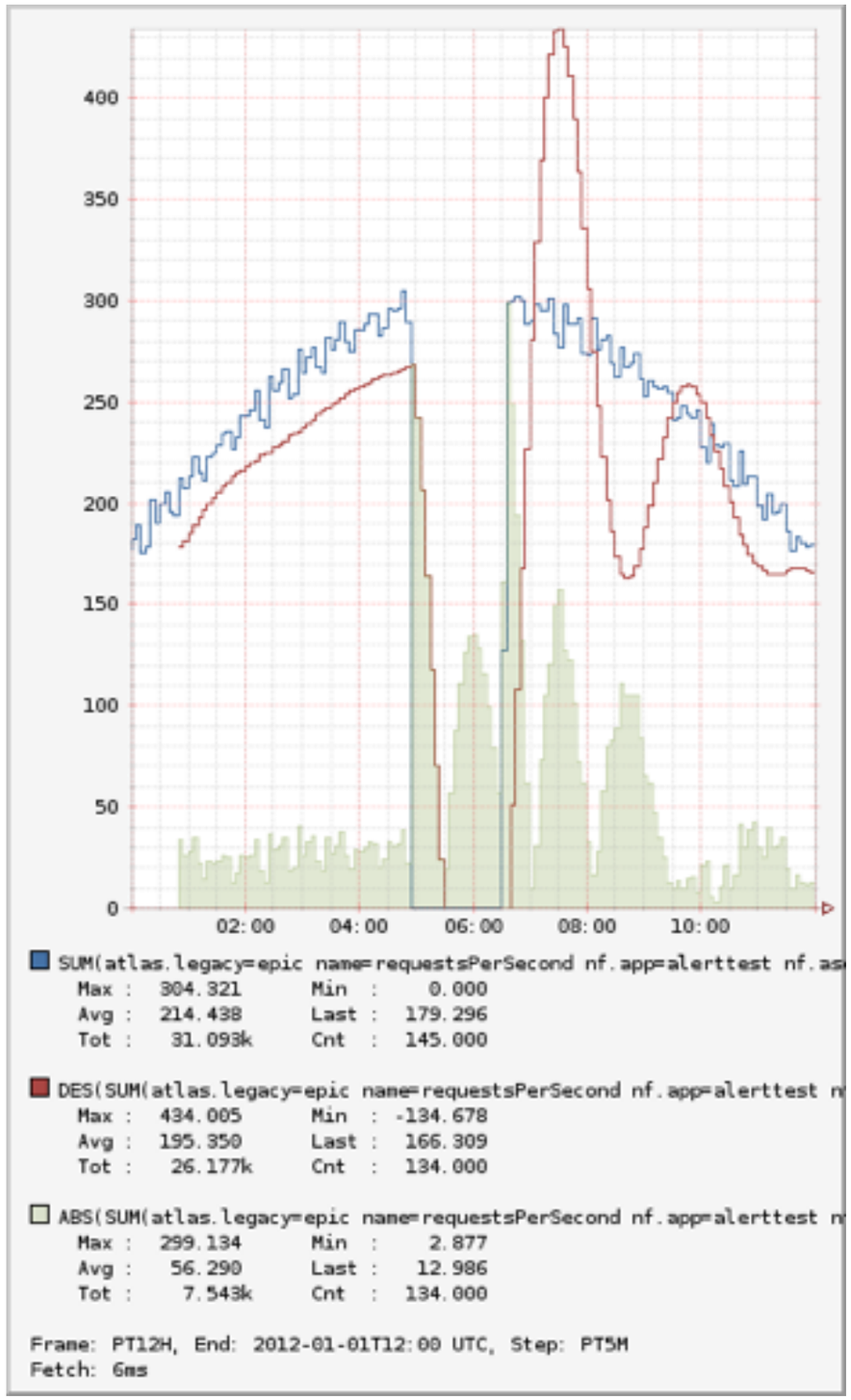
More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,

# Mutliply, used to set threshold
0.9,:mul,

# a b => a b abs(a - b)
:2over,:sub,:abs,
```



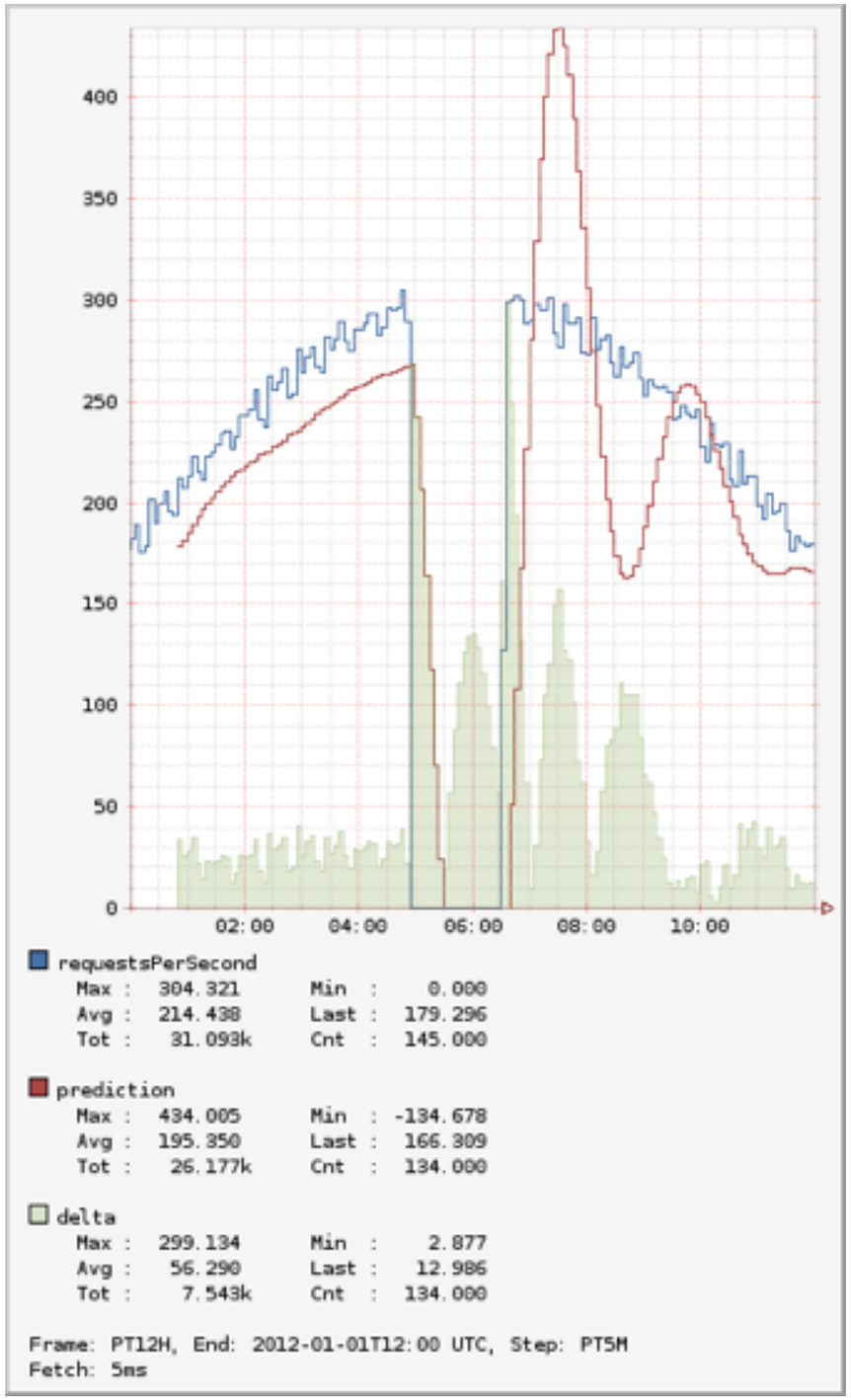
More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,

# Mutliply, used to set threshold
0.9,:mul,

# a b => a b abs(a - b)
:2over,:sub,:abs,

# Take line on top of stack
# and set it to area with transparency
:area,40,:alpha,
```



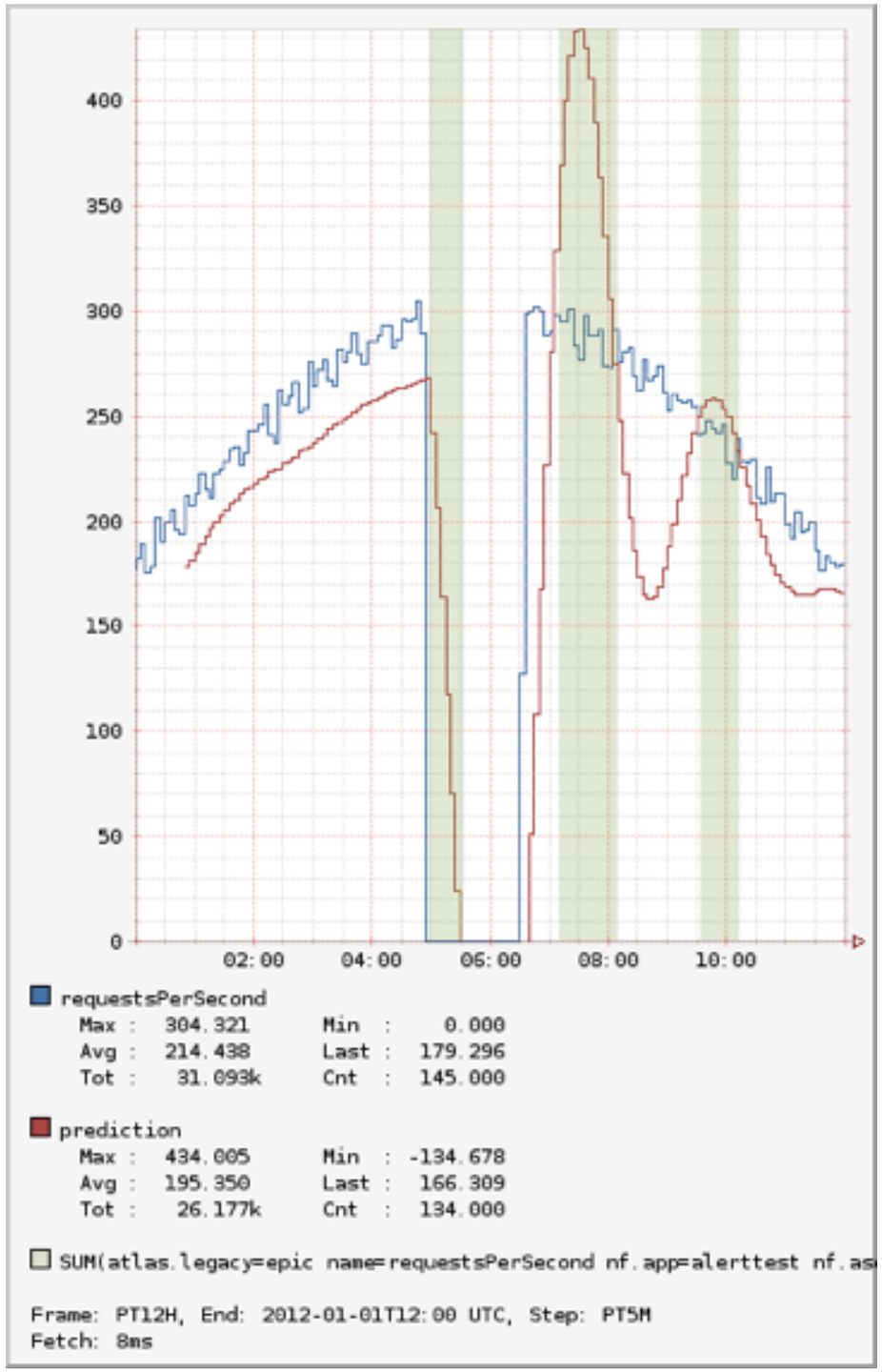
More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,

# Mutliply, used to set threshold
0.9,:mul,

# a b => a b abs(a - b)
:2over,:sub,:abs,

# Take line on top of stack
# and set it to area with transparency
:area,40,:alpha,

# Item on bottom of stack moved to
# top, set legend
:rot,$name,:legend,
:rot,prediction,:legend,
:rot,delta,:legend
```



More complex graph

```
# Query for input line
nf.cluster,alerttest,:eq,
name,requestsPerSecond,:eq,
:and,:sum,

# Create a copy on the stack
:dup,

# Create a DES line using the expr
# on top of the stack
:des-simple,

# Mutliply, used to set threshold
0.9,:mul,

# a b => a b (a < b)
:2over,:lt

# Take line on top of stack
# and set it to area with transparency
:area,40,:alpha,

# Item on bottom of stack moved to
# top, set legend
:rot,$name,:legend,
:rot,prediction,:legend,
:rot,:vspan,40,:alpha
```



Problem 2: storage

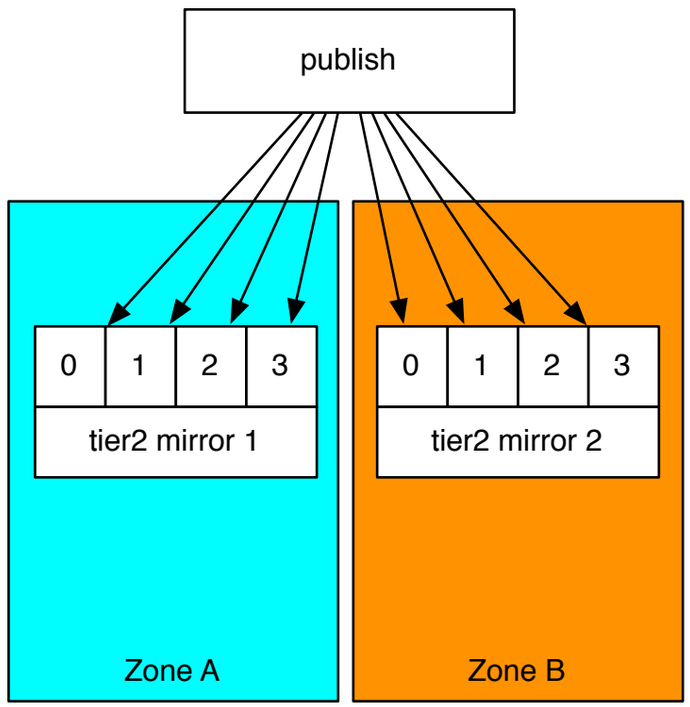
- What backend can effectively execute our queries over a large data set?
- What dependencies are required for monitoring to work?
  - E.g.: OpenTSDB > HBase > ZooKeeper

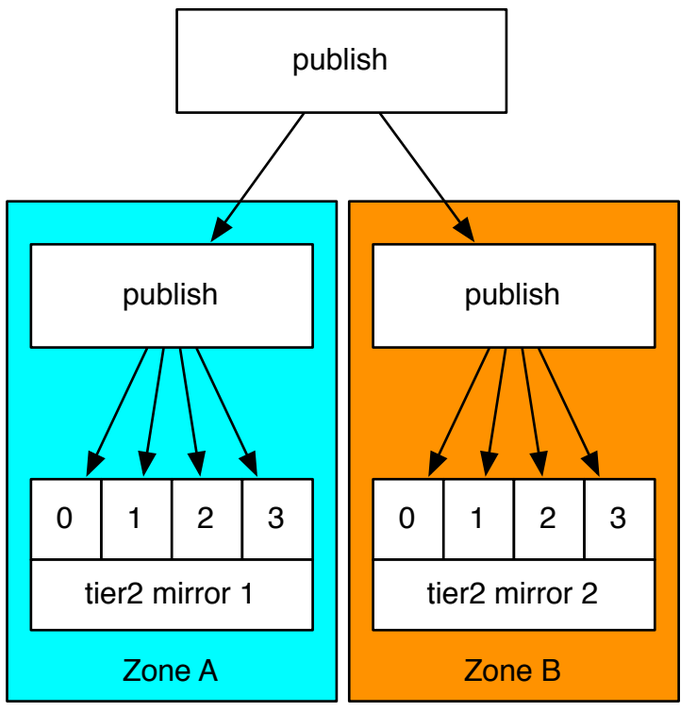
Problem 2: storage

- Split the problem
  - Short term data with minimal dependencies
  - Separate solution for longer term persistence

Short-term storage

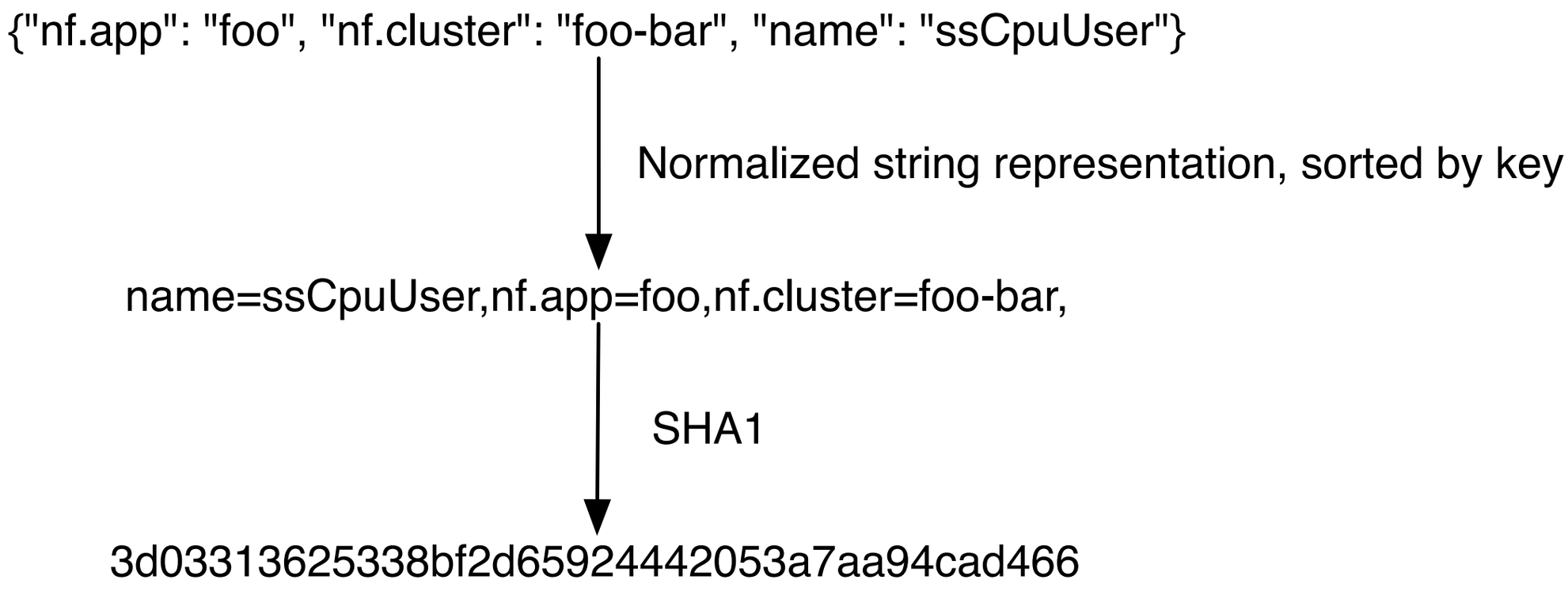
- What is short-term? ~6h
  - Transient time series, organize in 1h blocks, allocated as needed
  - Blocks can be compressed after 1 hour (array, constant, sparse)
- Built in-house, all data kept in memory





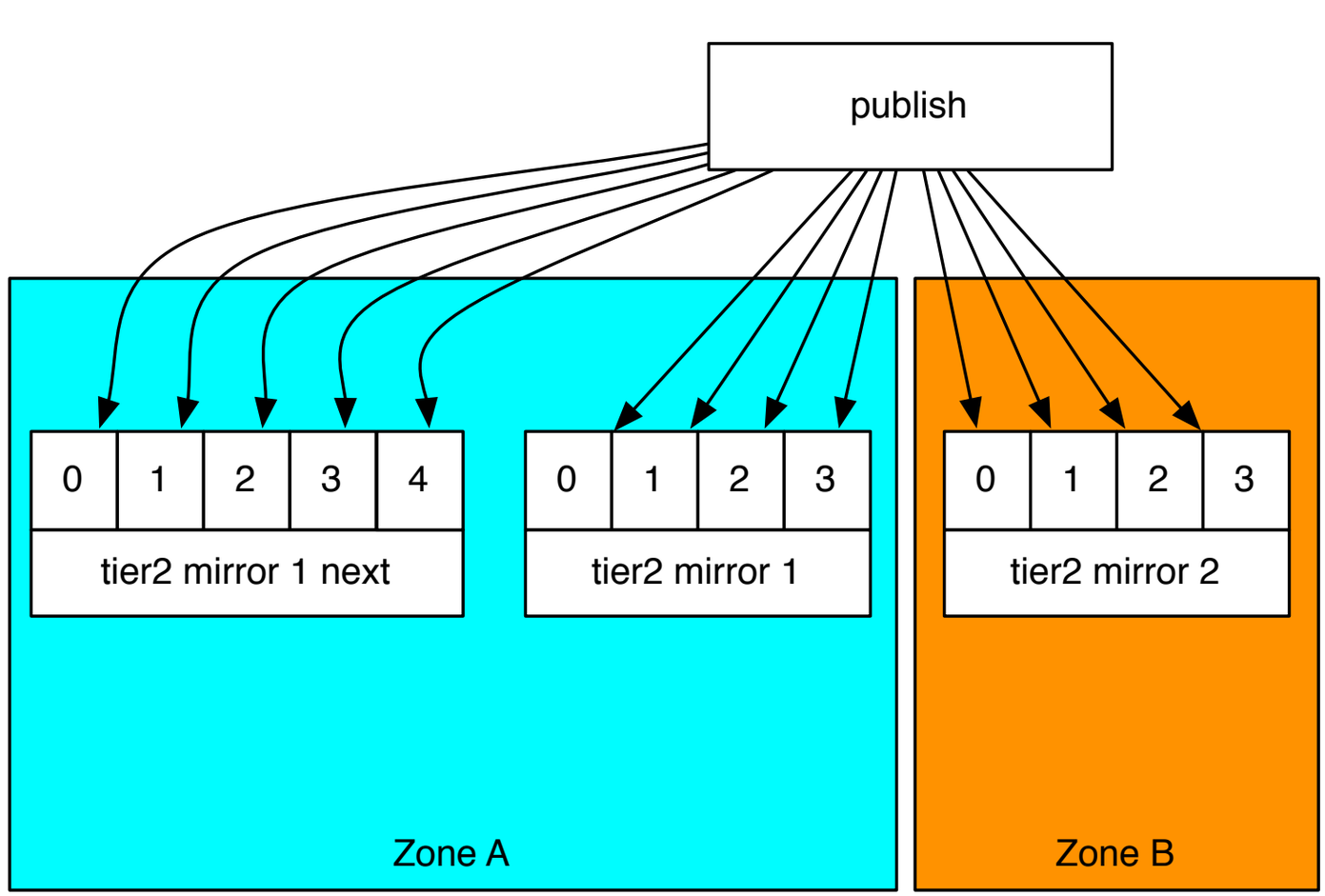
Short-term storage

- How do we shard the data?



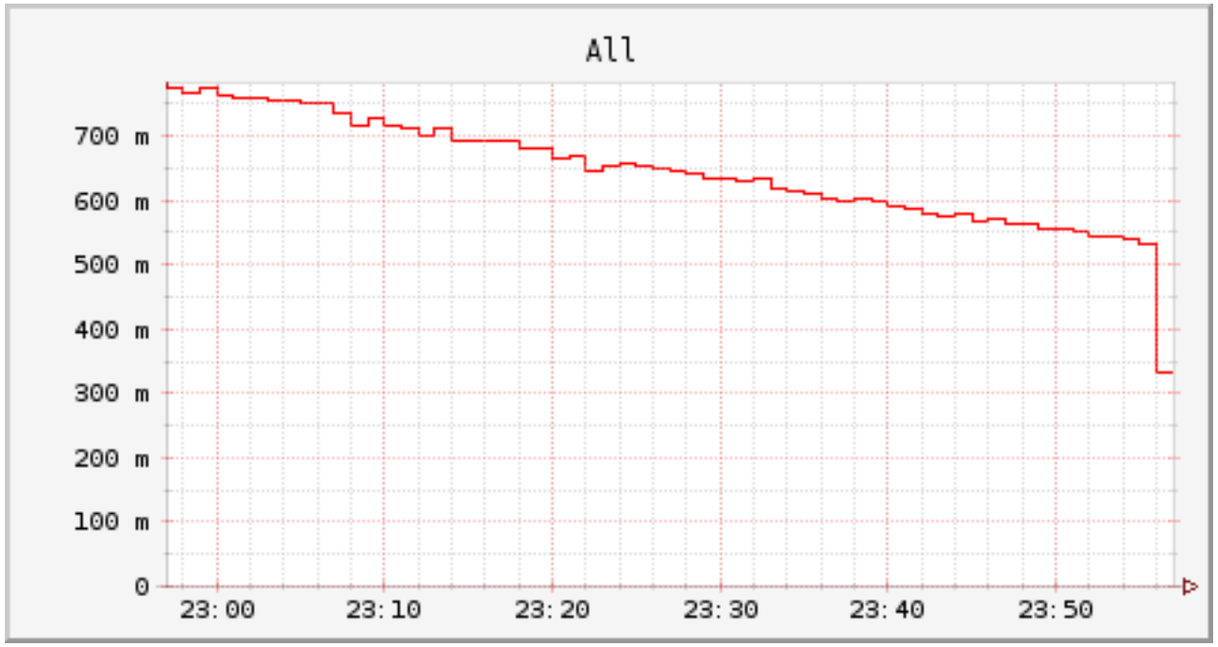
Short-term storage

- How do we deploy?
- Small window, after a few hours new deployment will have data



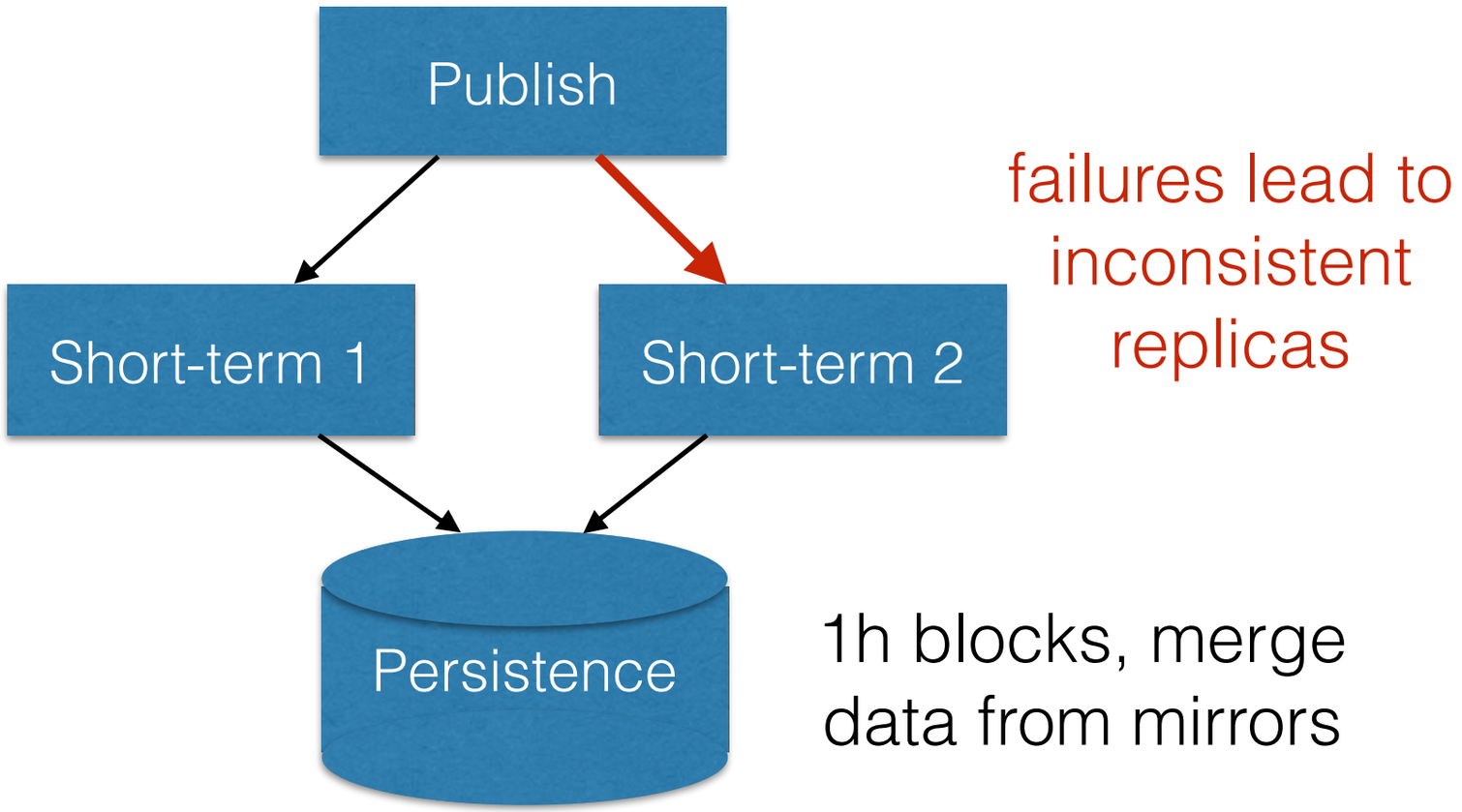
Short-term storage

- When is data visible?
- When is data actionable?



Short-term storage

- How does data get to long-term storage?



failures lead to inconsistent replicas

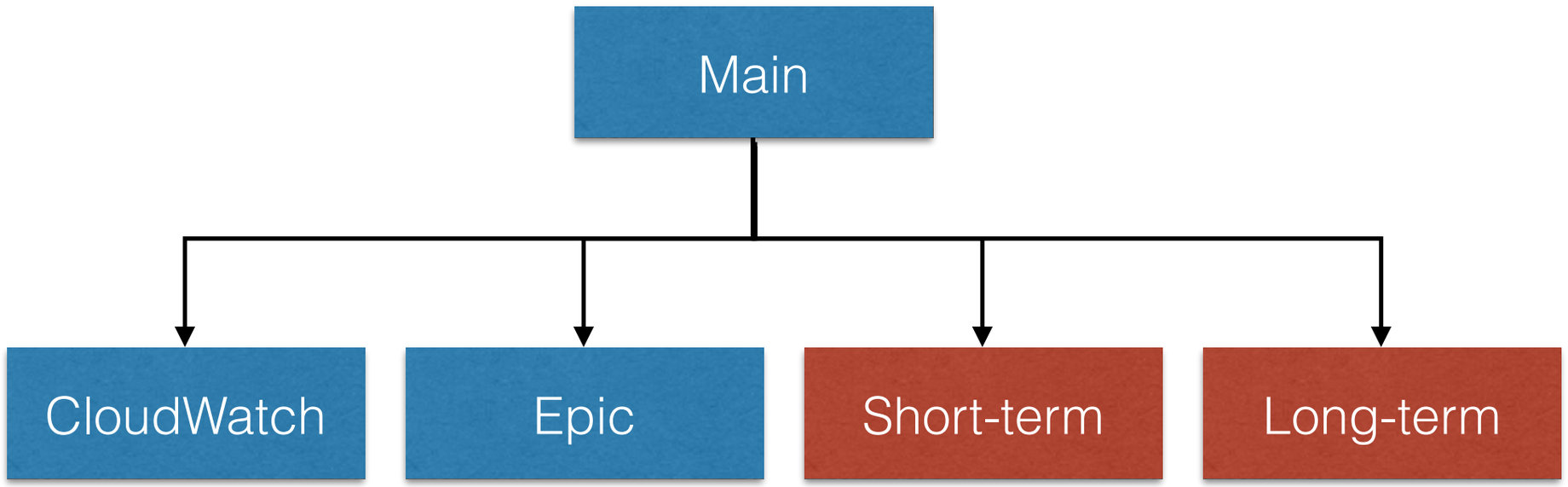
1h blocks, merge data from mirrors

Merging blocks

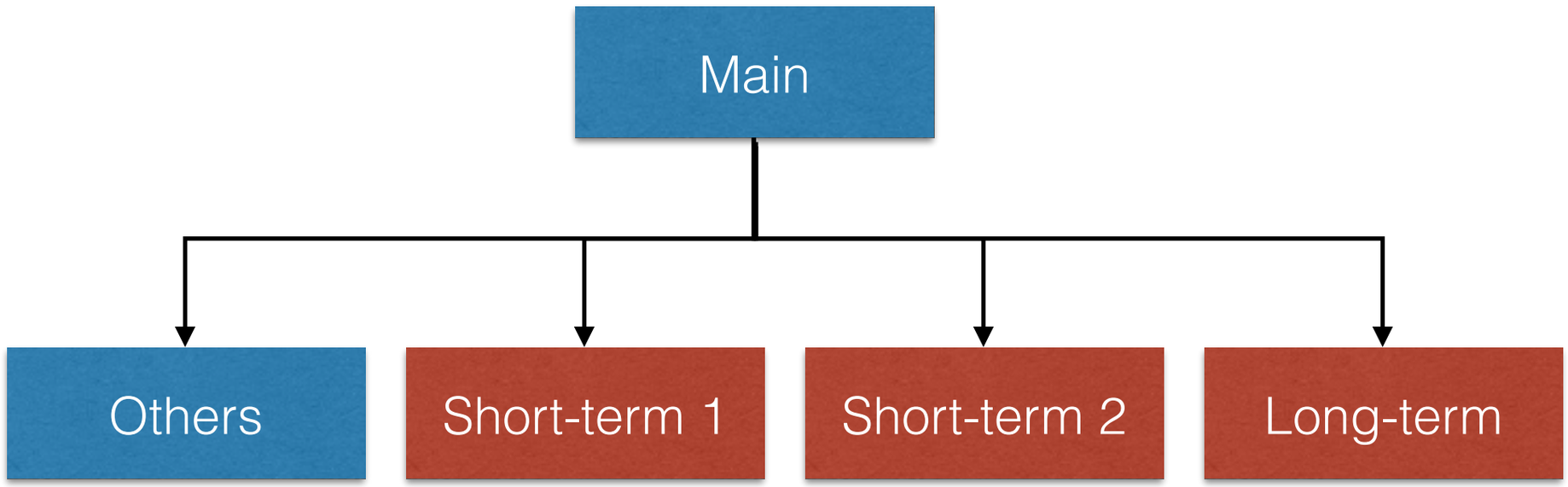
- Simple policy
  - Prefer a value to NaN
  - Prefer a larger value over a smaller one
  - Assumes data-loss is more likely to result in smaller values

Block 1	Block 2	Merged
NaN	42	42
42	42	42
42	42.1	42.1

Fit into query layer

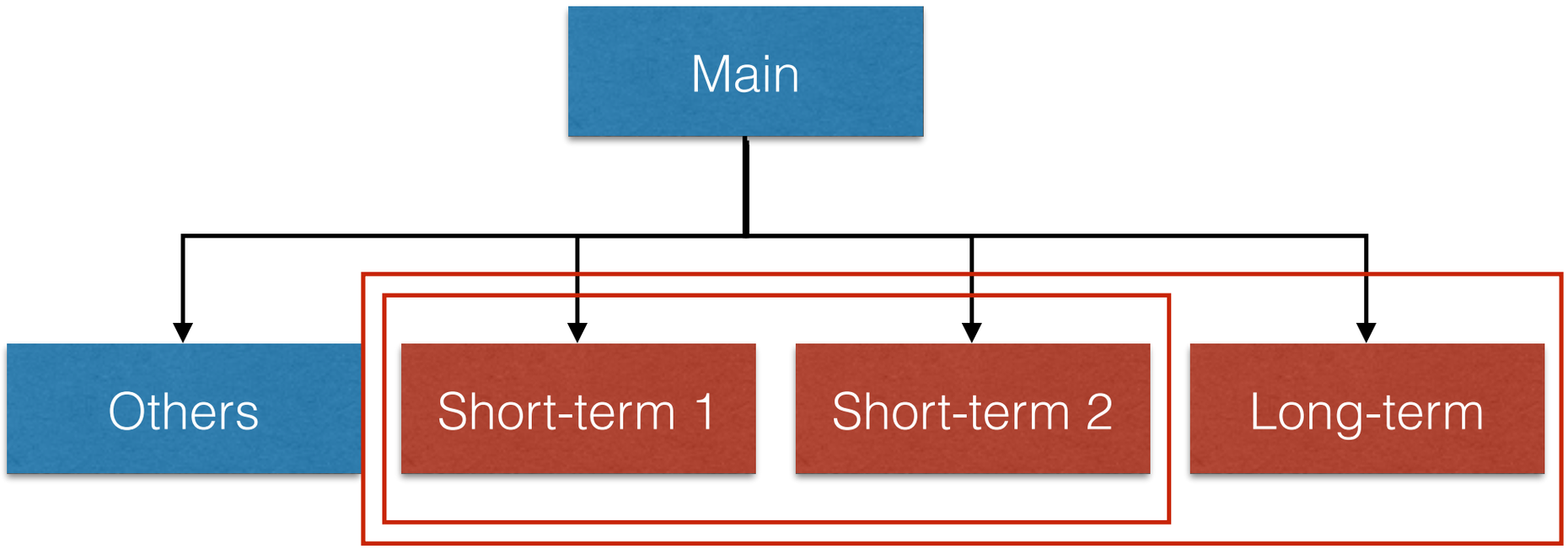


What about redundancy?



Must understand overlap in data

What about redundancy?



Query layer understand overlap in data

Querying Mirrors

- How do we query mirrors?
- Round-robin
- Speculative
  - First(All(A), All(B))
  - All(First(A.0, B.0), …, First(A.N, B.N))
- Correcting - query both and merge

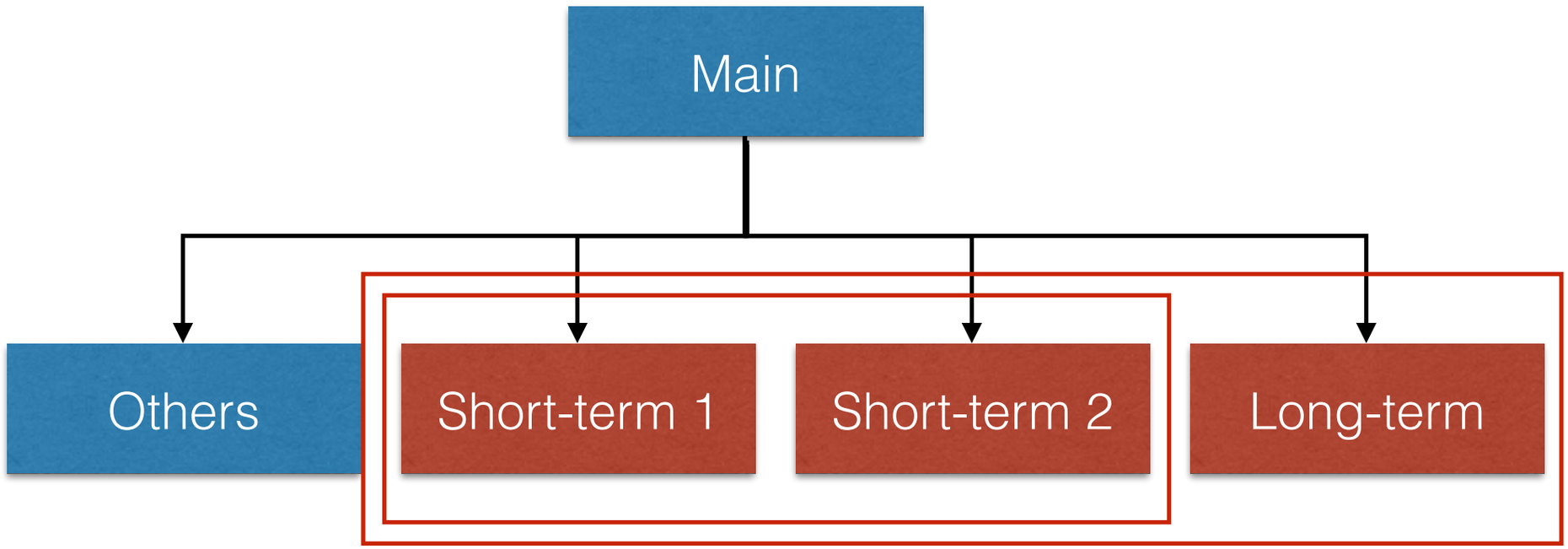
Long-term storage

- What is long-term? >4h
- How fast can it be accessed?
- Initially: MongoDB + Cassandra
  - MongoDB for metadata and expressive queries
  - Cassandra for block storage, lots of internal expertise
  - Disk was too slow for common query patterns
- Now: SQS + S3 + Hadoop

SQS + S3 + EMR

- Pros
  - Flexible processing with Hadoop based tools
  - More powerful inline rollups
  - Scales so far
- Cons
  - More work to build out
  - Really slow to access data that isn't loaded into serving tier

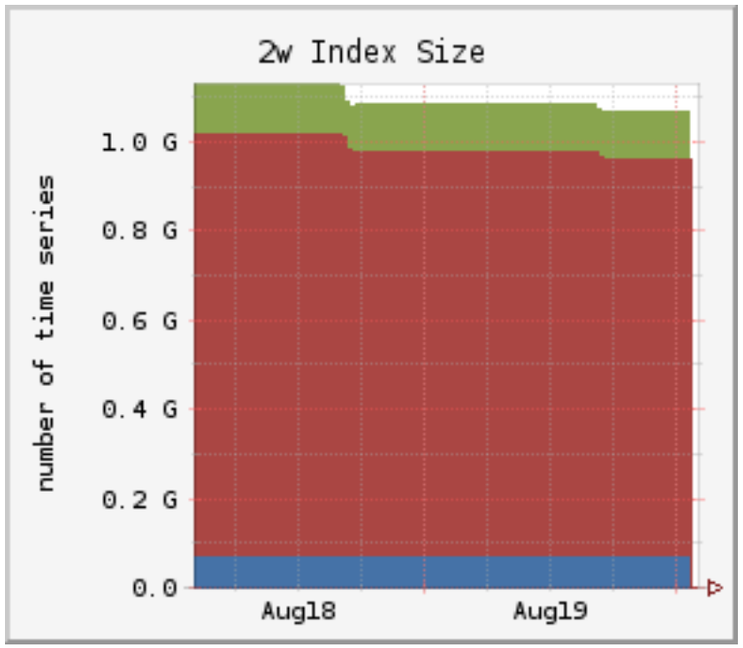
Query blending

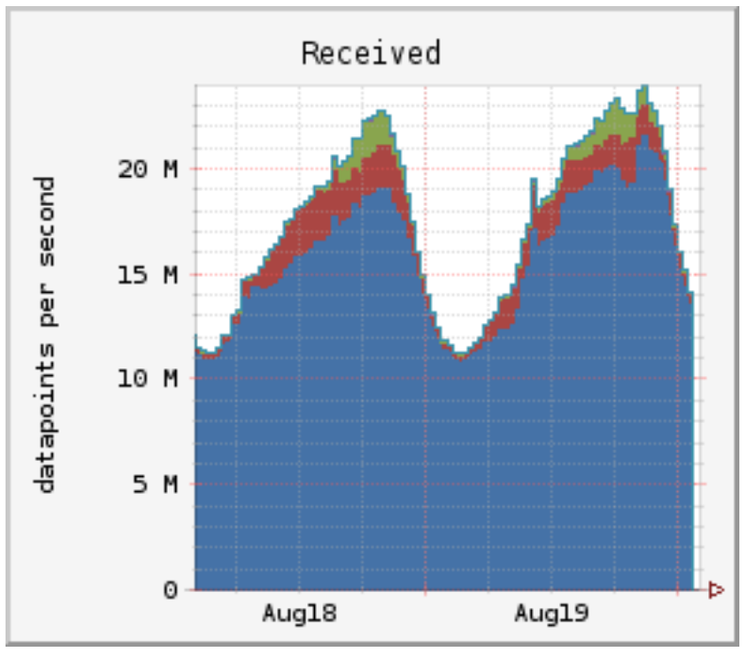


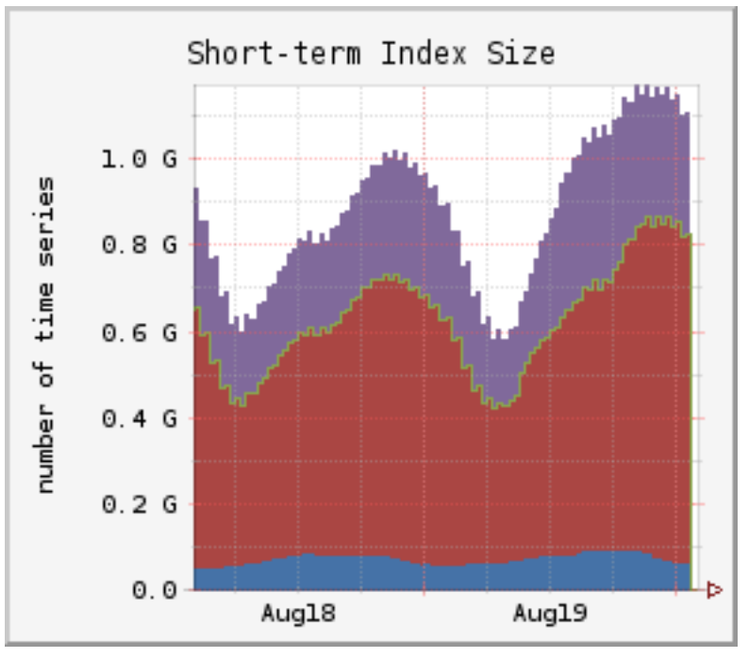
Query layer must match pre-computed rollup with dynamic rollups

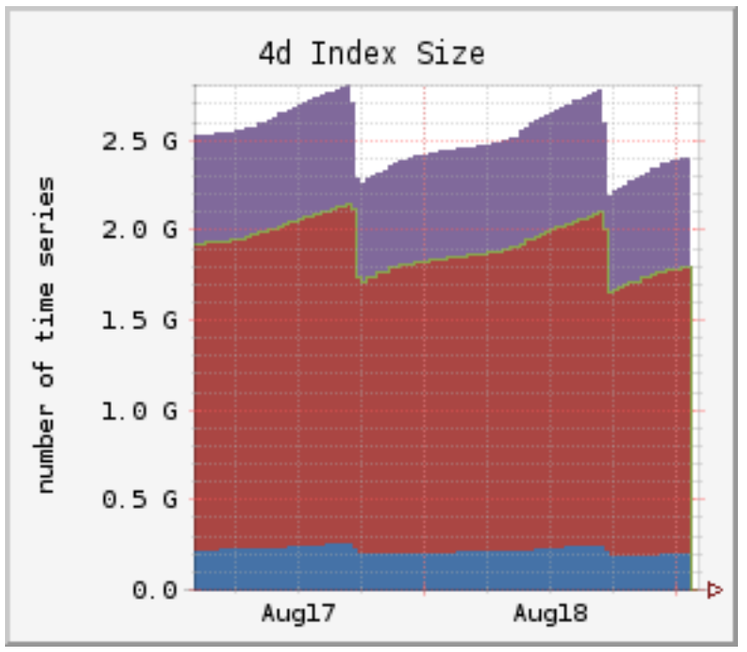
Rollups and Counting

- How many metrics?
  - Number of datapoints per second
  - Number of distinct time series
  - Rollups and retention windows

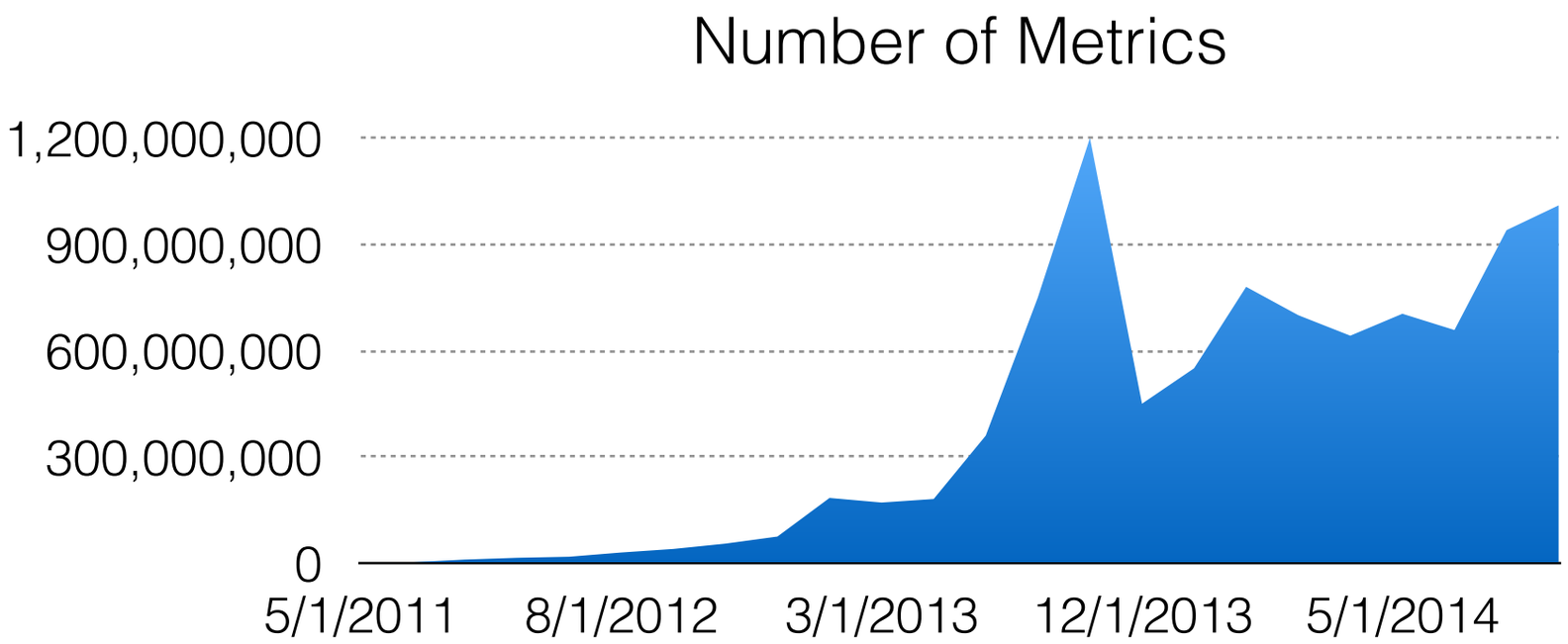


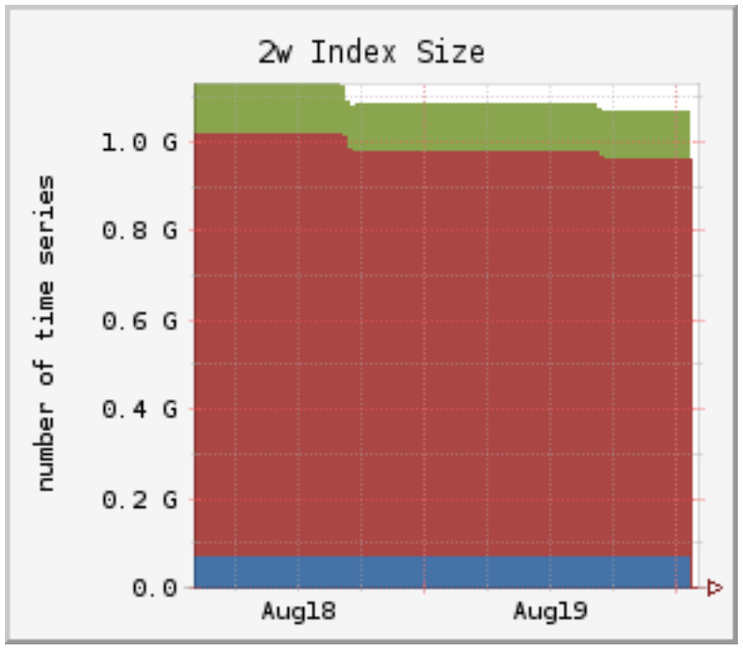


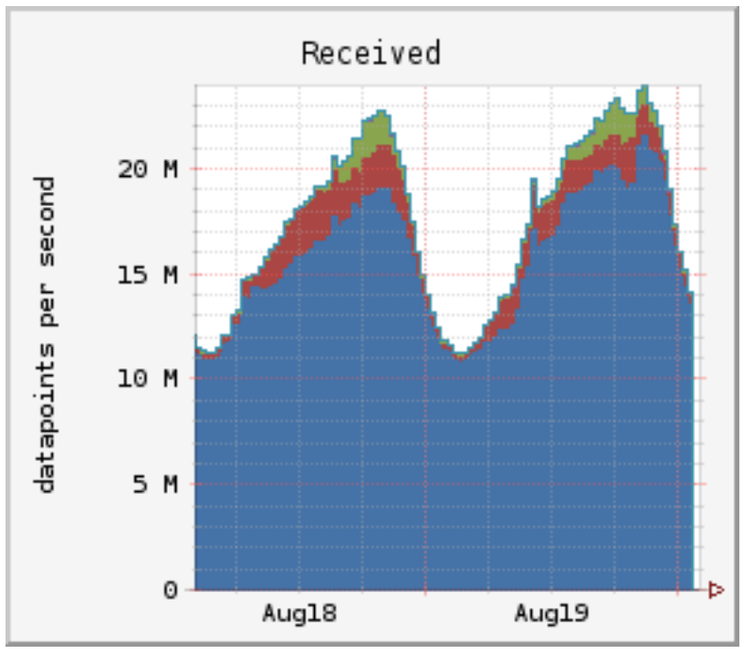


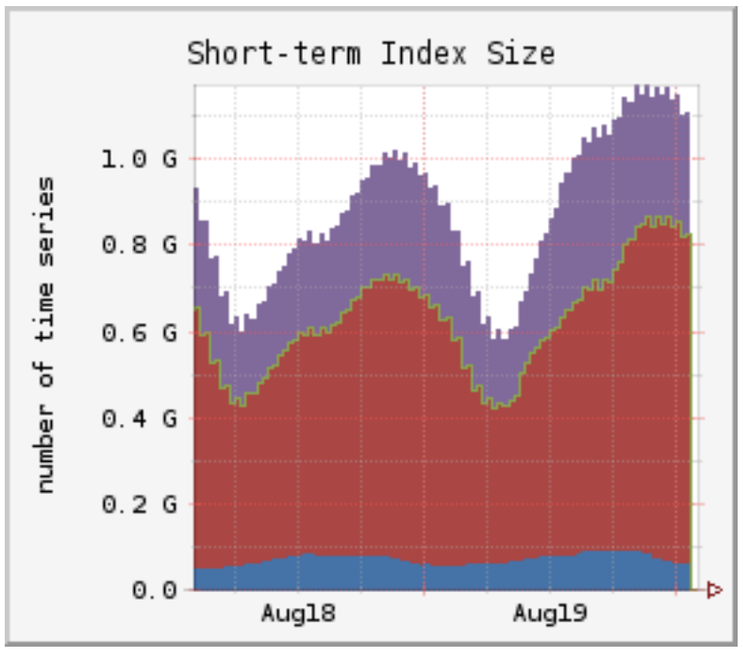


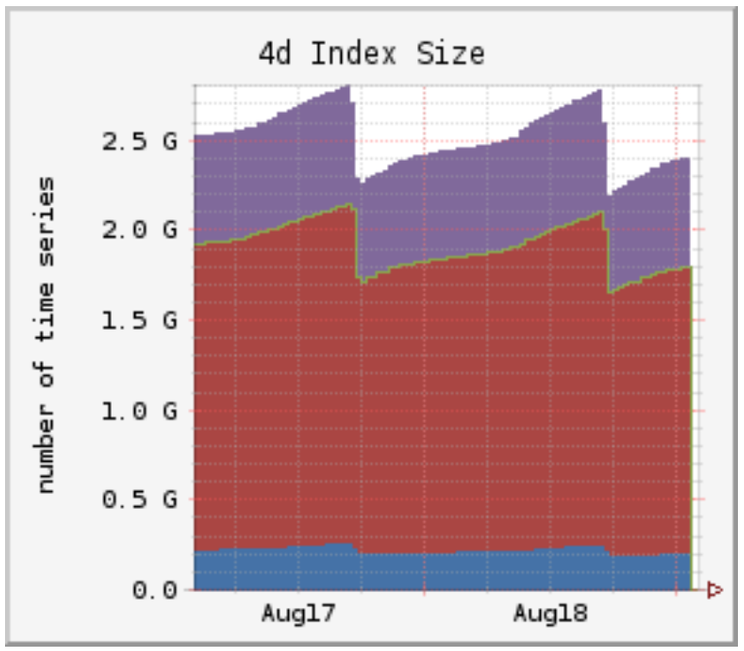
Rollups and Counting





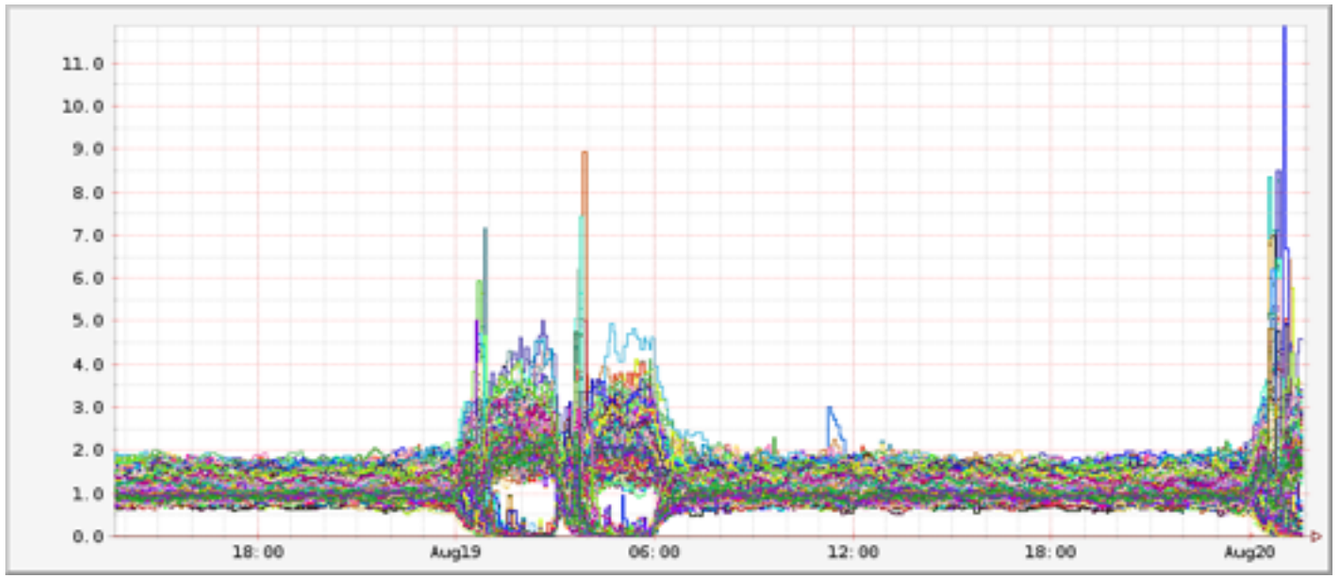


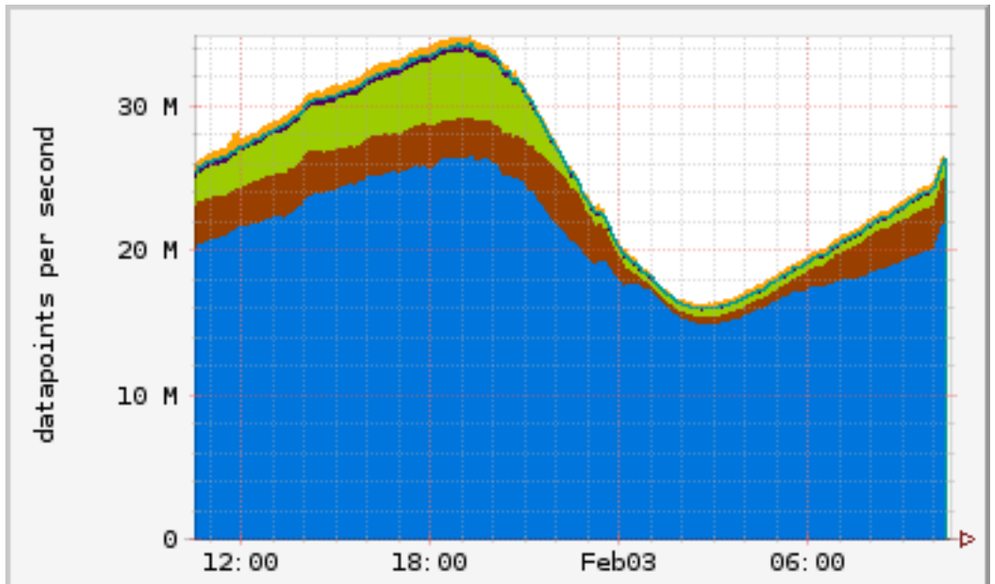




What problems remain?

- Correctness
- Degradation
- Heat maps
- Percentiles
- Dynamic visualization
  - Can we preserve deep linking?
- Streaming
- Scale





bucket=07s
Max : 26.376M Min : 14.790M
Avg : 20.585M Last : 21.828M
Tot : 5.929G Cnt : 288.000

bucket=15s
Max : 3.833M Min : 419.441k
Avg : 2.240M Last : 3.023M
Tot : 645.243M Cnt : 288.000

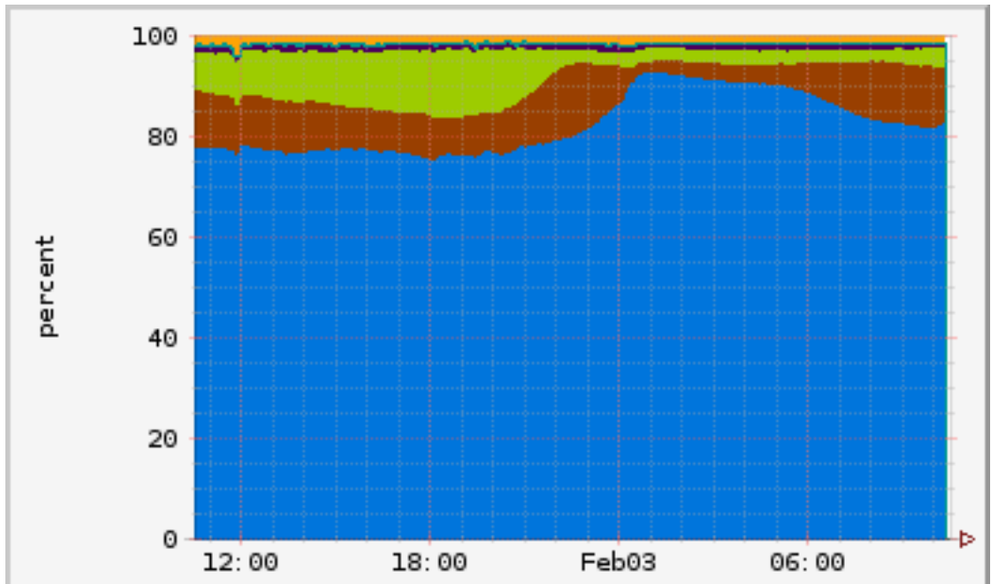
bucket=30s
Max : 4.702M Min : 405.649k
Avg : 1.905M Last : 1.001M
Tot : 548.782M Cnt : 288.000

bucket=60s
Max : 587.110k Min : 189.012k
Avg : 376.784k Last : 327.263k
Tot : 108.514M Cnt : 288.000

bucket=future
Max : 68.027k Min : 2.134k
Avg : 13.353k Last : 5.324k
Tot : 3.846M Cnt : 288.000

bucket=old
Max : 1.132M Min : 222.471k
Avg : 411.190k Last : 335.032k
Tot : 118.423M Cnt : 288.000

Frame: P1D, End: 2015-02-03T10:30 PST, Step: PT5M
Fetch: 640ms (L: 41.7k, 3.7k, 6.0; D: 3.0M, 1.1M, 1.7k)



bucket=07s
Max : 92.438 Min : 75.095
Avg : 81.944 Last : 82.310
Tot : 23.600k Cnt : 288.000

bucket=15s
Max : 14.546 Min : 2.257
Avg : 8.396 Last : 11.399
Tot : 2.418k Cnt : 288.000

bucket=30s
Max : 13.567 Min : 2.367
Avg : 6.539 Last : 3.774
Tot : 1.883k Cnt : 288.000

bucket=60s
Max : 1.885 Min : 1.102
Avg : 1.465 Last : 1.234
Tot : 421.968 Cnt : 288.000

bucket=future
Max : 330.167m Min : 9.508m
Avg : 53.349m Last : 20.078m
Tot : 15.365 Cnt : 288.000

bucket=old
Max : 4.053 Min : 1.201
Avg : 1.603 Last : 1.263
Tot : 461.547 Cnt : 288.000

Frame: P1D, End: 2015-02-03T10:30 PST, Step: PT5M
Fetch: 1536ms (L: 41.9k, 3.7k, 6.0; D: 3.0M, 1.1M, 1.7k)

Tools Using Atlas

- Alerting
  - Threshold
  - RTA (outlier and anomaly detection)
- Dashboards
- Performance

?