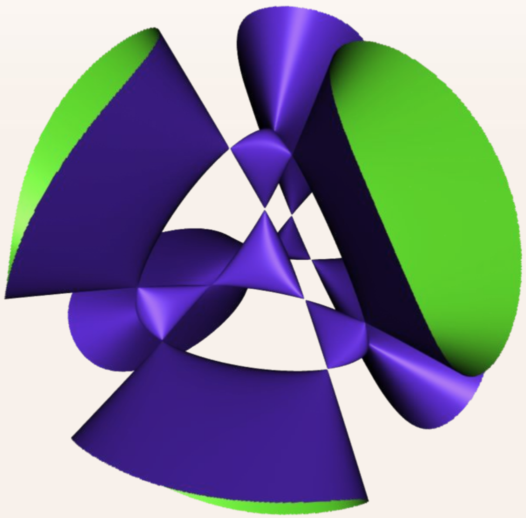
Scalable Online Analytics for Monitoring

LISA15, Nov. 13, 2015, Washington, D.C.

Heinrich Hartmann, PhD, Chief Data Scientist, Circonus

I'm Heinrich

Heinrich.Hartmann@Circonus.com



- From Mainz, Germany
- Studied Math. in Mainz, Bonn, Oxford
- PhD in Algebraic Geometry
- Sensor Data Mining at Univ. Koblenz
- Freelance IT consultant
- Chief Data Scientist at Circonus

Circonus is ...

- monitoring and telemetry analysis platform
- scalable to millions of incoming metrics
- available as public and private SaaS
- built-in histograms, forecasting, anomaly-detection, ...

This talk is about...

I. The Future of Monitoring

II. Patterns of Telemetry Analysis

III. Design of the Online Analytics Engine ‘Beaker’

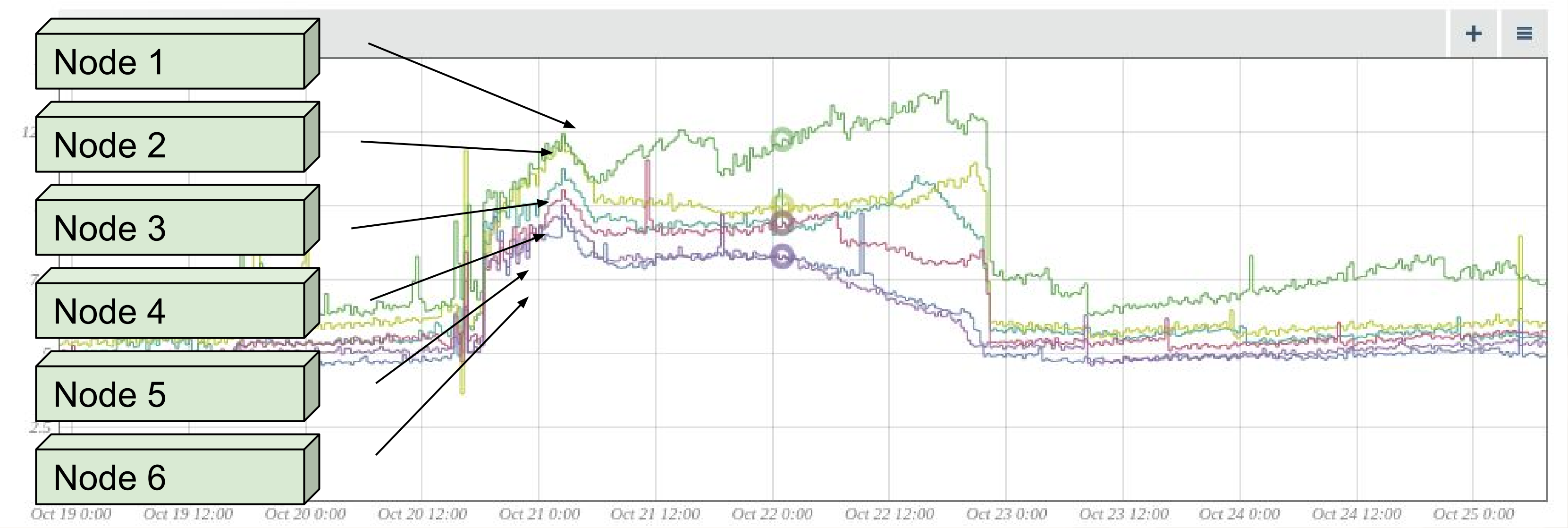
Part I - The Future of Monitoring

The “cloud monitoring challenge” 2015

Monitor this:

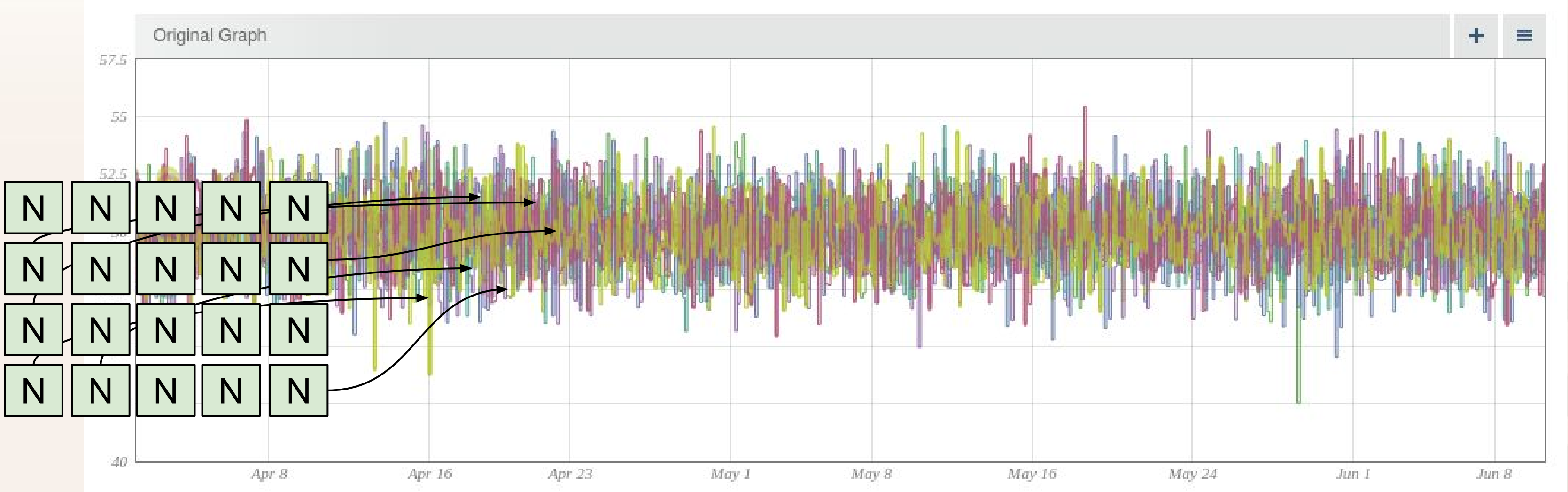
- 100 containers in one fleet
- 200 metrics per-container
- 50 code pushes a day
- For each push all containers are recreated

Line-plots work well for small numbers of nodes



CPU utilization for a DB cluster. Source: Internal.

... but can get polluted easily



CPU utilization for a DB cluster. Source: Internal.

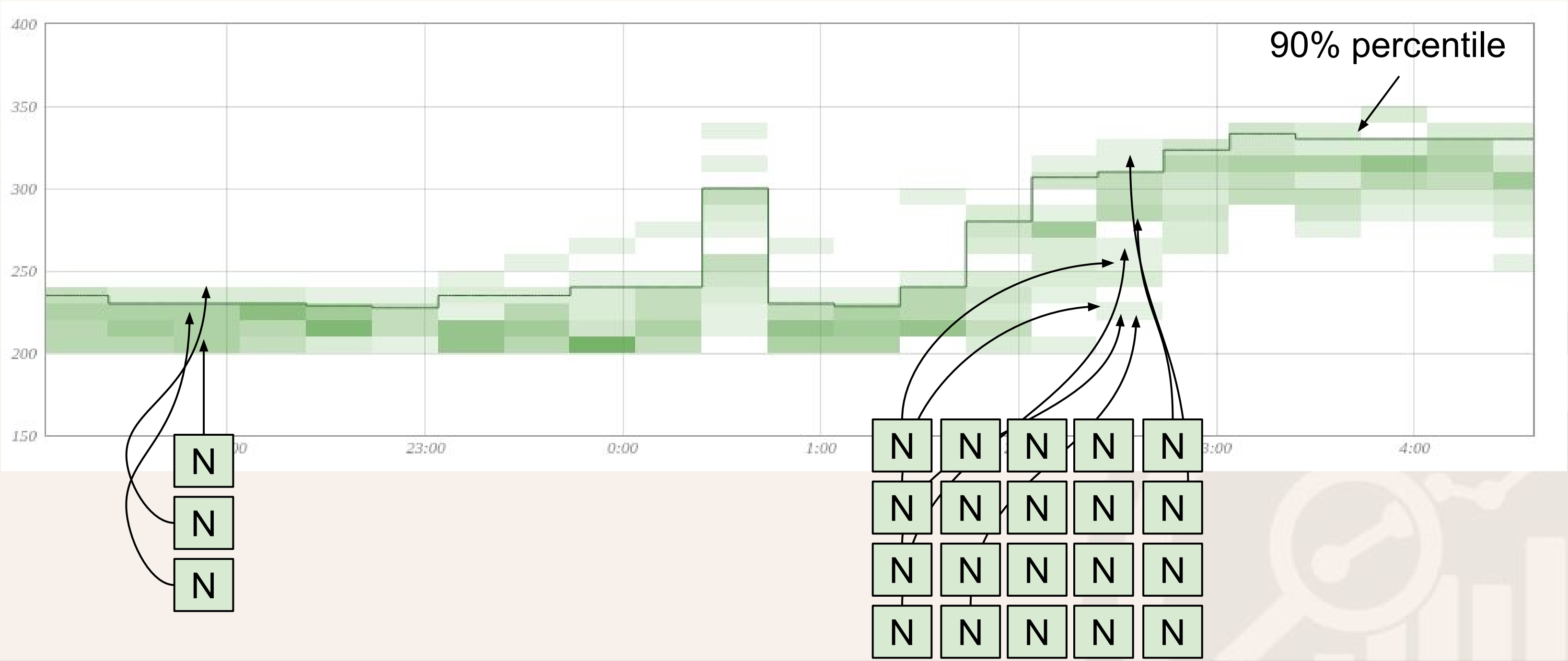
“Information is not a scarce resource. Attention is.”

Herbert A. Simon

Source: http://www.circonus.com/the-future-of-monitoring-qa-with-john-allspaw/

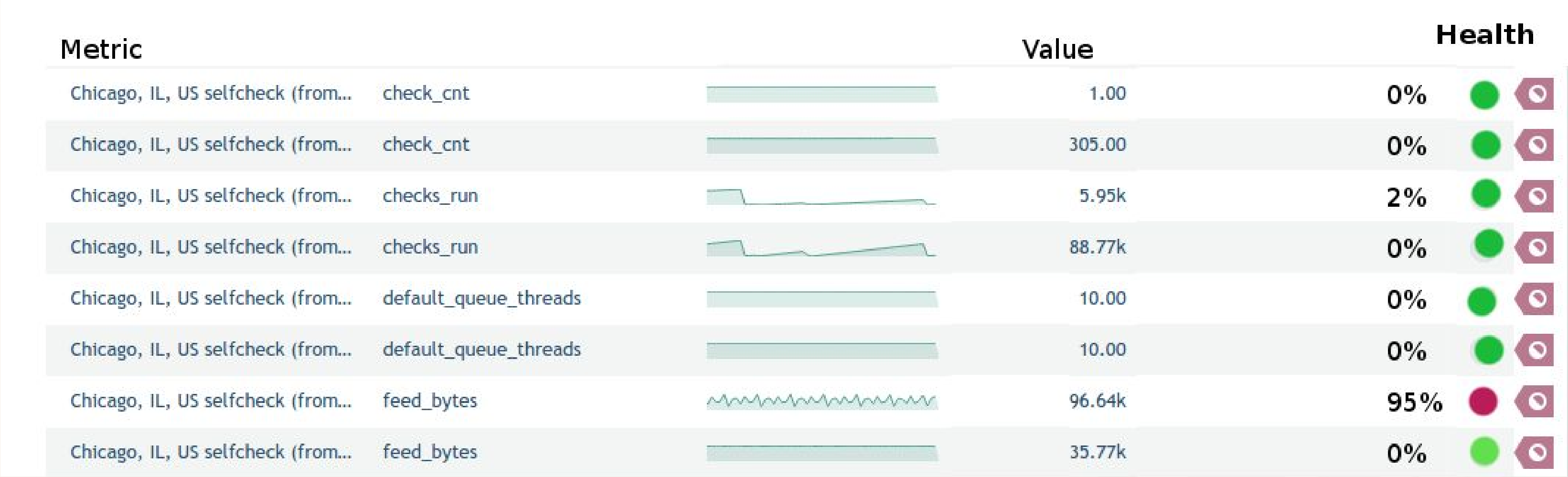
Store Histograms and Alert on Percentiles

CPU Utilization of a db service with a variable number of nodes.



CIRCONUS

Anomaly Detection for Surfacing relevant data



Metric		Value	Health	
Chicago, IL, US selfcheck (from...	check_cnt	1.00	0%	
Chicago, IL, US selfcheck (from...	check_cnt	305.00	0%	
Chicago, IL, US selfcheck (from...	checks_run	5.95k	2%	
Chicago, IL, US selfcheck (from...	checks_run	88.77k	0%	
Chicago, IL, US selfcheck (from...	default_queue_threads	10.00	0%	
Chicago, IL, US selfcheck (from...	default_queue_threads	10.00	0%	
Chicago, IL, US selfcheck (from...	feed_bytes	96.64k	95%	
Chicago, IL, US selfcheck (from...	feed_bytes	35.77k	0%	

Mockup of an metric overview page. Source: Circonus

Part II - Patterns of Telemetry Analysis

Telemetry analysis comes in two forms

Online Analytics

- Anomaly detection
- Percentile alerting
- Smart alerting rules
- Smart dashboards

Stream Processing

Offline Analytics

- Post mortem analysis
- Assisted thresholding

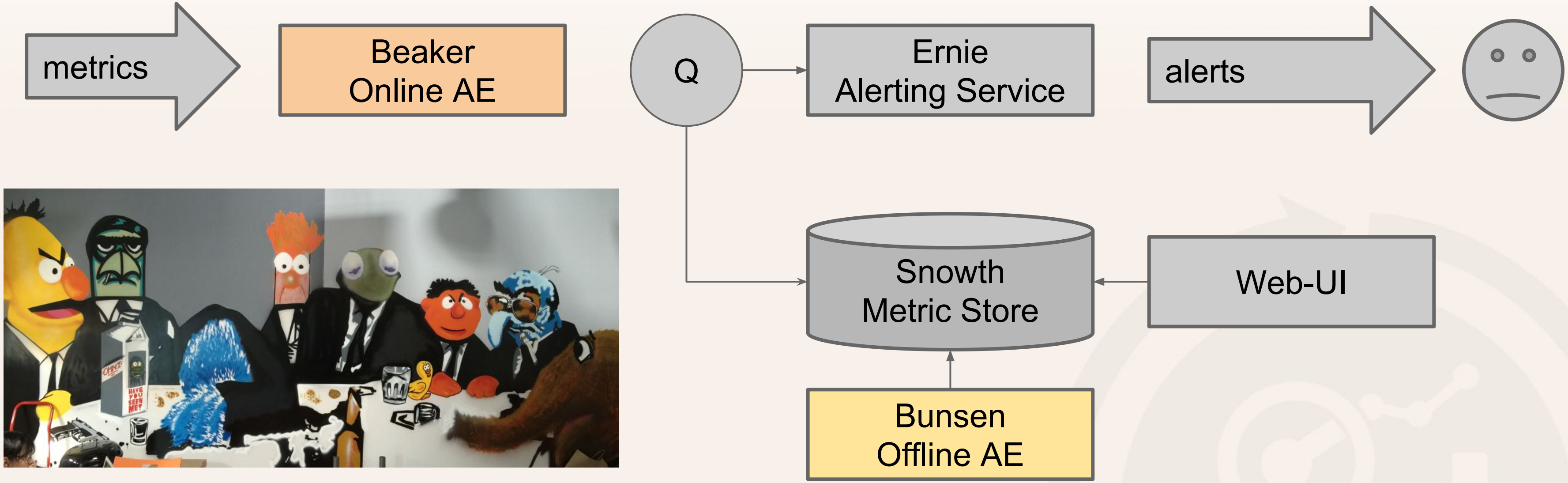
'Big Data'

New Components in Circonus

Beaker

Bunsen

Beaker & Bunsen in the Circonus Architecture



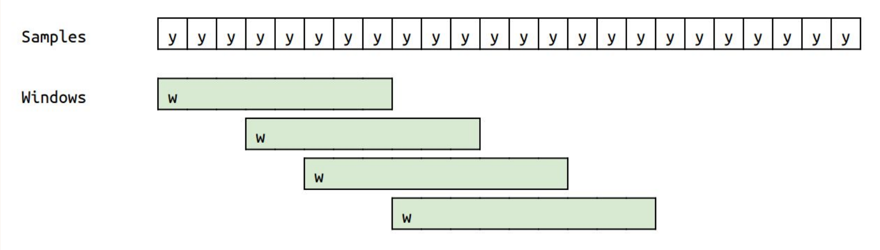
Pattern 1: Windowing

```
windows = window(y_stream)

def results():

  for w in windows:

    yield z = method(w)
```



Examples

- Supervised Machine Learning
- Fourier Transformation
- Anomaly Detection (etsy)

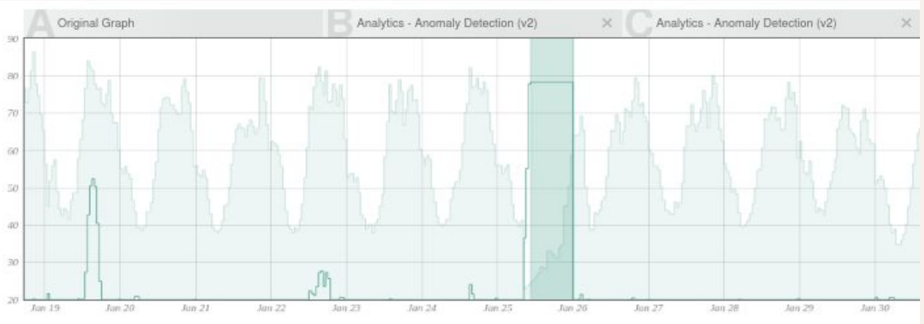
Remarks

- Tradeoff: window size
  rich features vs. memory

- Tradeoff: overlap
  latency vs. CPU

Pattern 3: Processing Units

```
processing_unit = {
  state = ...
  update = function(self, y) ... end
}
```



Circonus Anomaly Detection

Example

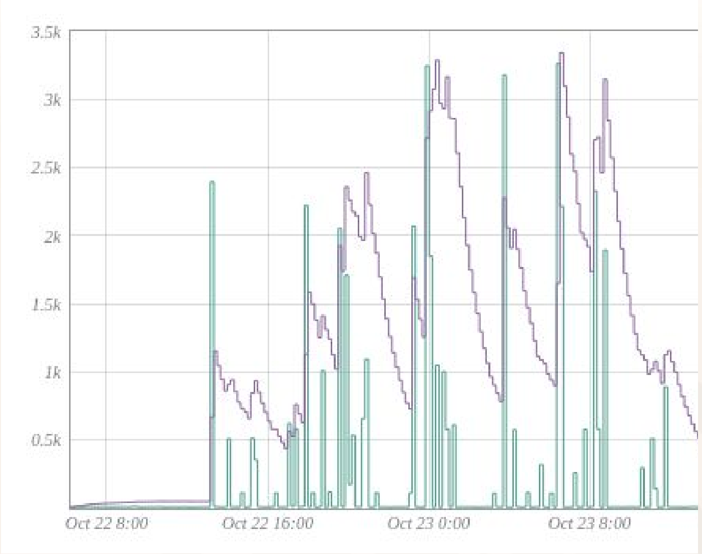
- Exponential Smoothing
- Holt Winters forecasting
- Anomaly Detection (Circonus)

Remarks

- Fast updates
- Fully general
- Cost: maintains state

A Processing Unit for Exponential Smoothing

```
local q = 0.9
exponential_smoothing = {
    s = 0,
    update = function(self, y)
      self.s = (y * q) + (self.s * (1 - q))
      return self.s
    end
}
```



Exponential smoothing applied to a dns-duration metric

For

Processing units are convenient

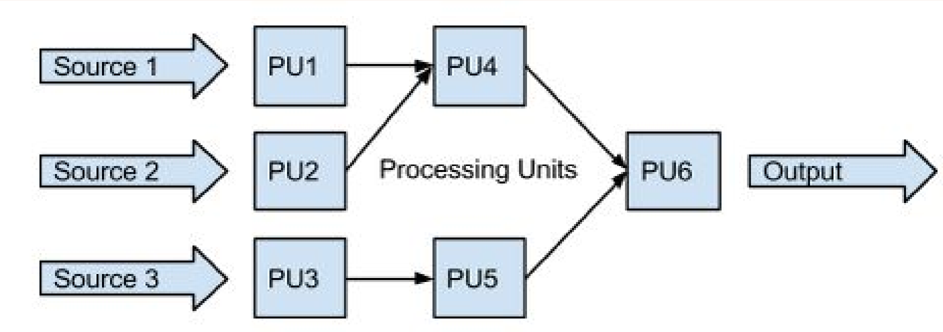
- Primitive transformation are readily implemented: Arithmetic, Smoothing, Forecasting, ...

- Fully general. Allow window-based processing as well

- Composable. Compose several PUs to get a new one!

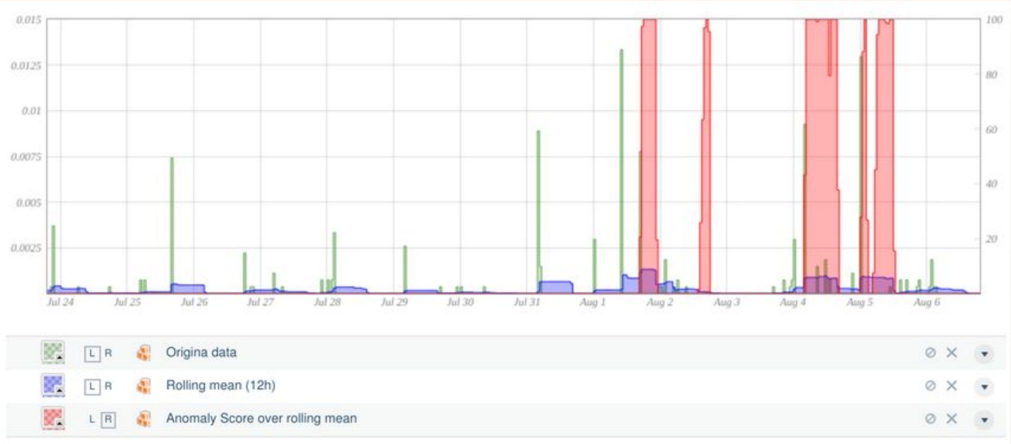
Circonus Analytics Query Language

- Create your own customized processing unit from primitives
- UNIX-inspired syntax with pipes ‘|’
- Native support for histogram metrics



CAQL: Example 1 - Low frequency AD

Pre-process a metric before feeding into anomaly detection




```
metric:average(<>) | rolling:mean(30m) | anomaly_detection()
```

CAQL: Example 2 - Histogram Aggregation

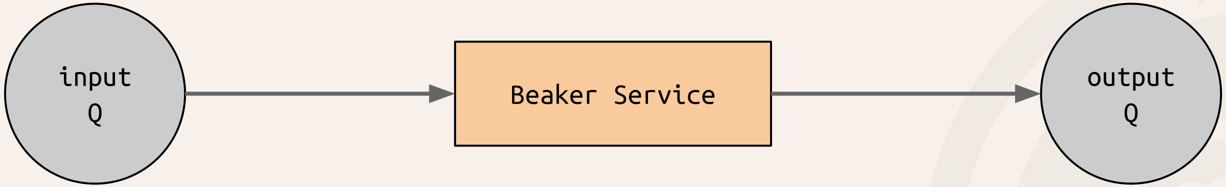
Histograms are first-class citizens


```
metric:average(<uuid>) | window:histogram(1h) | histogram:percentile(95)
```

Part III - The Design of the Online Analytics Engine ‘Beaker’

Beaker v. 0.1: Simple stream processing

1. Read messages from a message queue
2. Execute a processing units over incoming metrics
3. Publish computed values to message queue



Beaker: Basic Data Flow

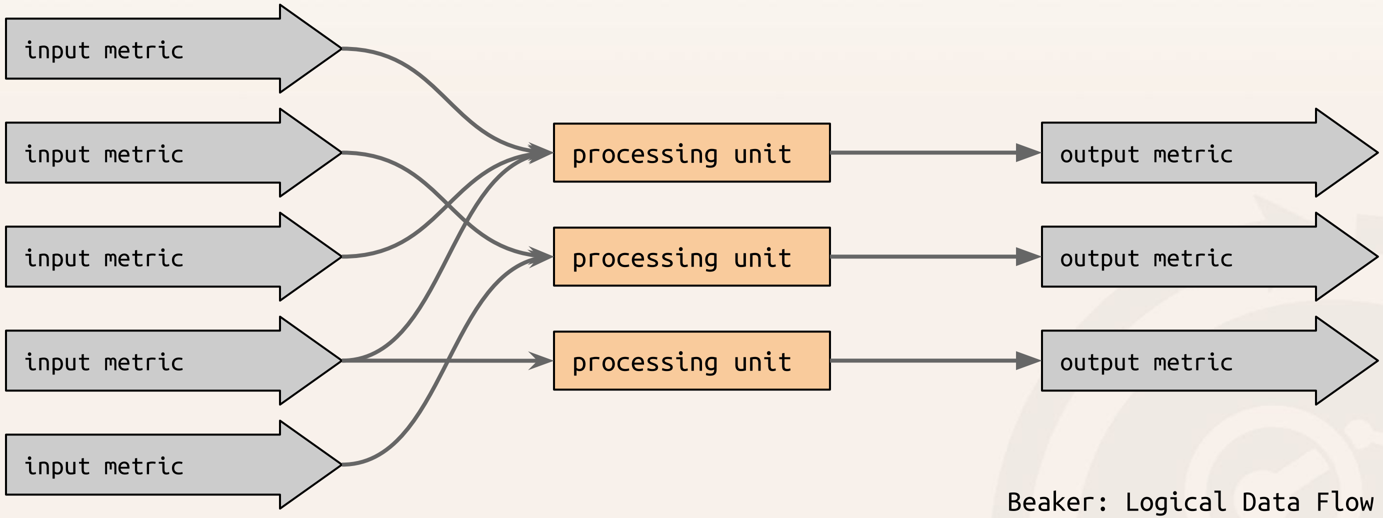
Challenge 1: Rollup metrics by the minute

- Metrics arrive asynchronously on the input queue
- PUs expect exactly one sample per minute
- Rollup logic needs to allow for
  - late arrival, e.g. when broker is behind
  - out of order arrival
  - errors in system time (time-zones, drifts)

Consequence: No real-time processing in Beaker

- Rolling up data in 1m periods causes an average delay of 30sec for data processing.
- Real-time threshold-based alerting still available.
- Approach: Avoid rolling up data for stateless PUs.

Challenge 2: Multiple input slots



Challenge 3:
Synchronize roll-ups for multiple inputs is tricky

Challenge 4: Fault Tolerance

Definition (Birman): A service is fault tolerant if it tolerates the failure of nodes without producing wrong output messages.

Failed nodes must be able to recover from a crash and rejoin the cluster.

The time to recovery should be as low as possible.

Source: K. Birman - Reliable Distributed Systems, Springer, 2005

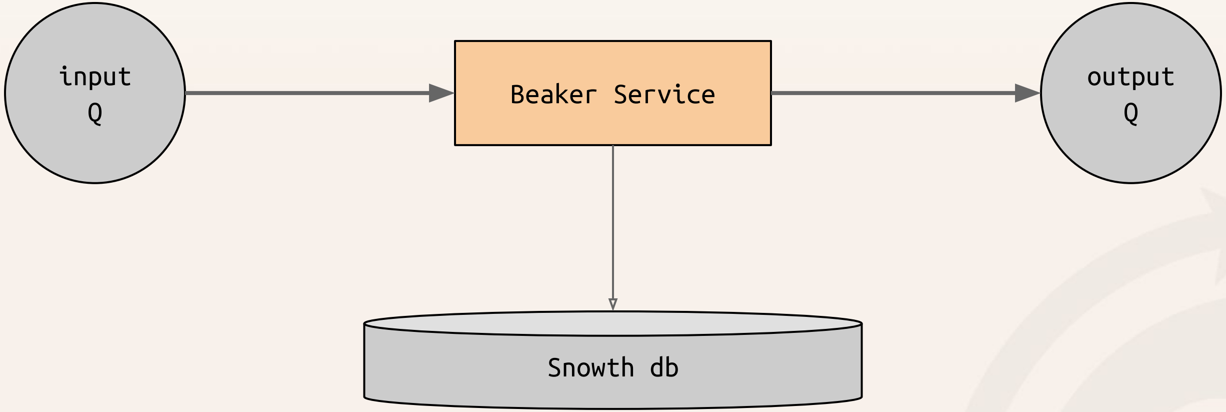
Beaker v0.5:

- Automated restarts
  a. Service Management Facilities (svcadm) in OmniOS
  b. systemd or watchdogs in Linux

- But, recovery can take a **long** time
  State has to be rebuilt from input stream.

- Need a way to recover faster from errors:
  a. Persist processing unit state (software updates!)
  b. Access persisted metric data

Beaker v. 0.5: Use db to rebuild state on startup



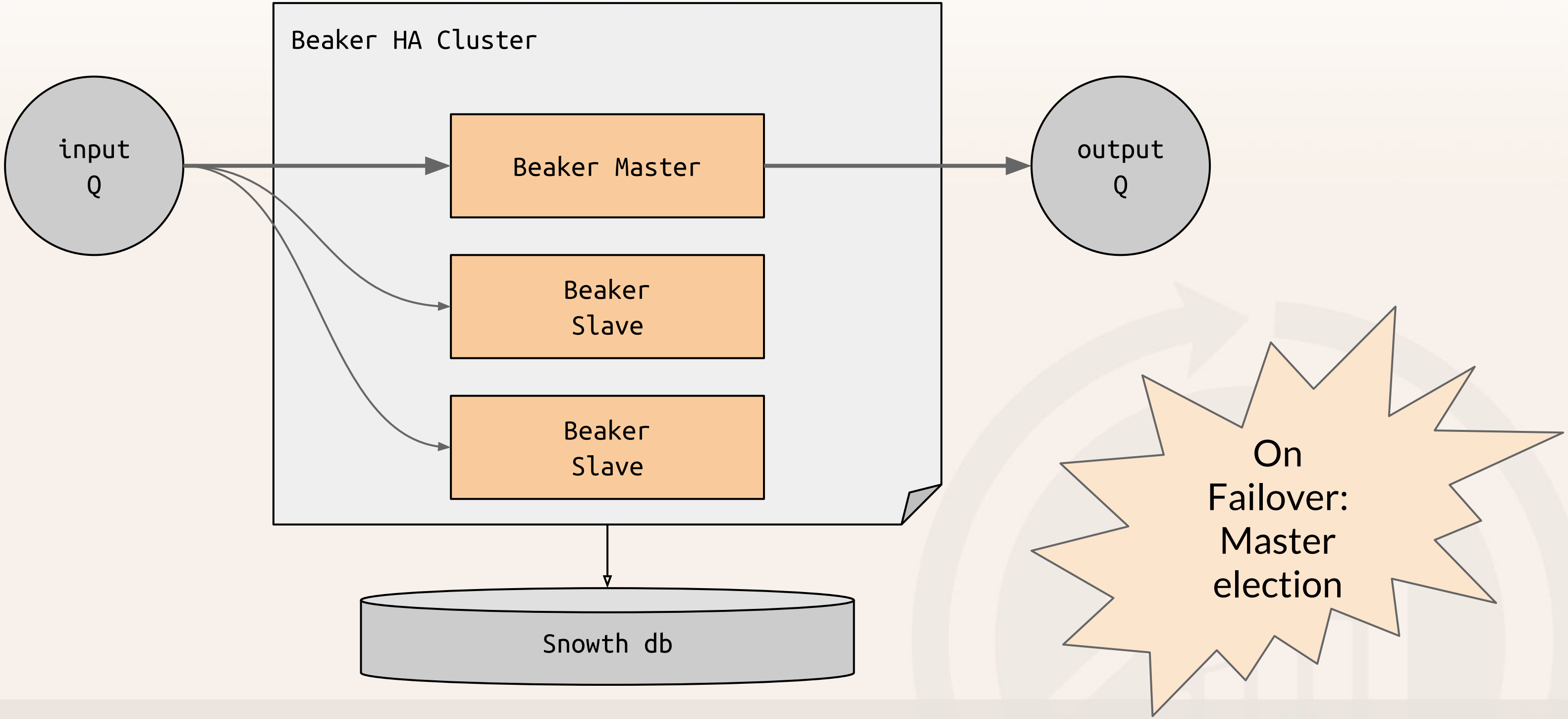
Challenge 5: High Availability

Definition (Birman): A service is highly available it continues to publish valid messages during a node failure after a small reconfiguration period.

For Beaker we require a reconfiguration period of less than 1 minute. In this time messages may be delayed (e.g. 30sec) and duplicated messages may be published.

Source: K. Birman - Reliable Distributed Systems, Springer, 2005

Beaker v. 0.5 -- HA Cluster

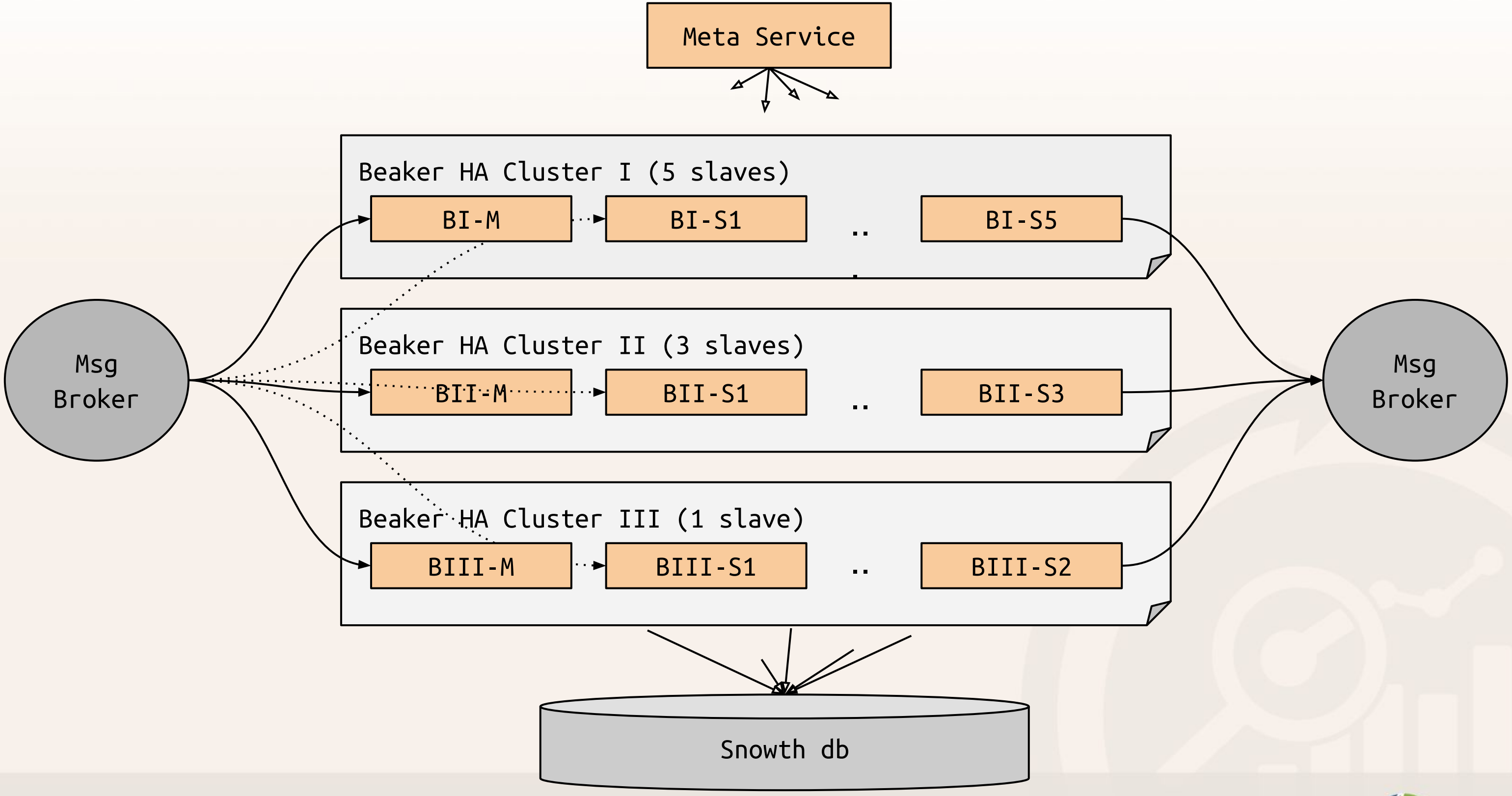


Challenge 6: Scalability

Beaker needs to scale in the following dimensions

- Number of processing units up to an unlimited amount.
- In the number of incoming metrics up to ~100M metrics.

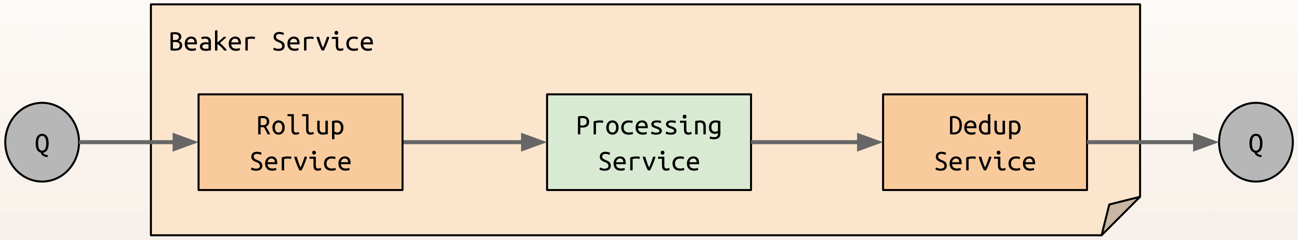
Beaker v.0.6 -- Multiple - HA Clusters



Done! Great. This works...

Can we do better?

Beaker v. 1.0 -- Divide and Conquer

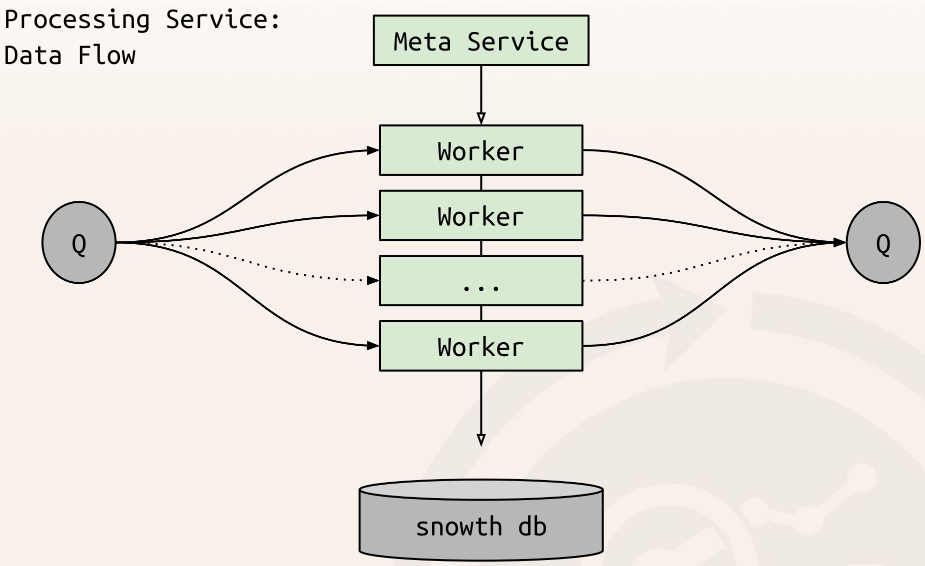


- Divide Beaker into multiple services
- Avoid master election in processing service, by allowing duplicates
- Upside: Only the processing service needs to scale out

Scaling the Processing Service is simplified

- No rollup logic in workers
- All workers publish messages
- No failover logic necessary
- Replicate each PUs on multiple workers

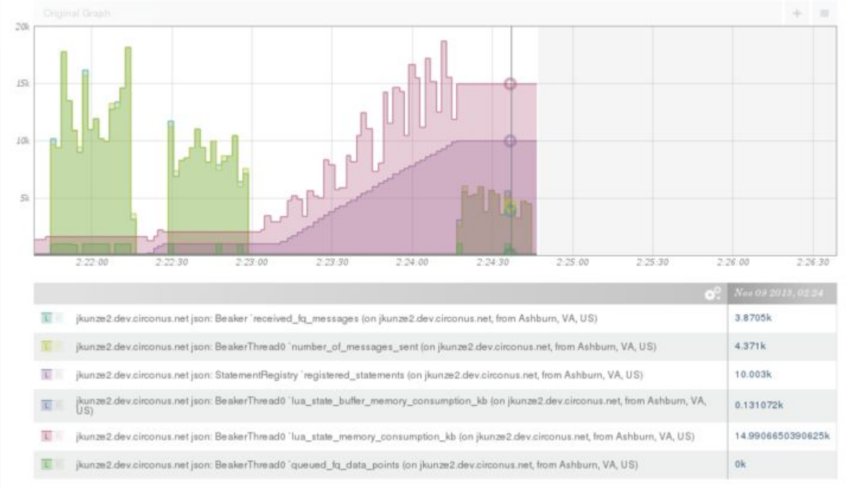
- No crosstalk between nodes (κ = 0 in USL).



Benchmark results on prototype

- PU type anomaly_detection
  PU count 100
  Throughput 15 kHz

- PU type anomaly_detection
  PU count 10k
  Throughput 4.2 kHz



Machine: 6-core Xeon E5 2630 v2 at 2.6 GHz

Conclusion

- Stateful processing units allow implementation of next generation of monitoring analytics
- Use CAQL to build your own processing units
- Service orientation facilitates scaling
- Beaker will be out soon

Credits

Joint work with:

- [Jonas Kunze]
- [Theo Schlossnagle]

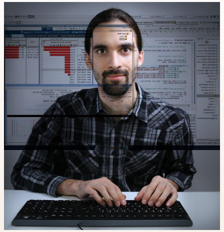


Image Credits:

Example of Kummer Surface, Claudio Rocchini, CC-BY-SA, https://en.wikipedia.org/wiki/File:Kummer_surface.png

Indian truck, by strudelt, CC-BY, https://commons.wikimedia.org/wiki/File:Truck_in_India_-_overloaded.jpg

Kafka, Public Domain, https://en.wikipedia.org/wiki/File:Kafka_portrait.jpg

Thinker, Public Domain, https://www.flickr.com/photos/mustangjoe/5966894496