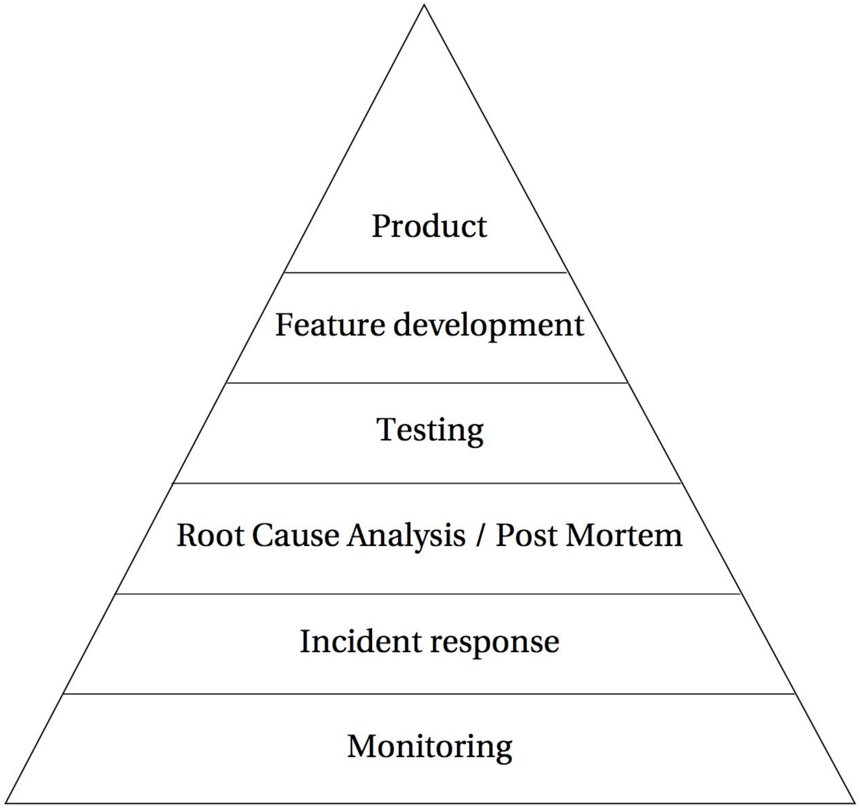
A practical guide to monitoring and alerting with time series at scale

SREcon17 Americas

Jamie Wilkinson <jaq@google.com>
Site Reliability Engineering, Google



Dickerson Hierarchy of Needs

Why does #monitoringsuck?

TL;DR:

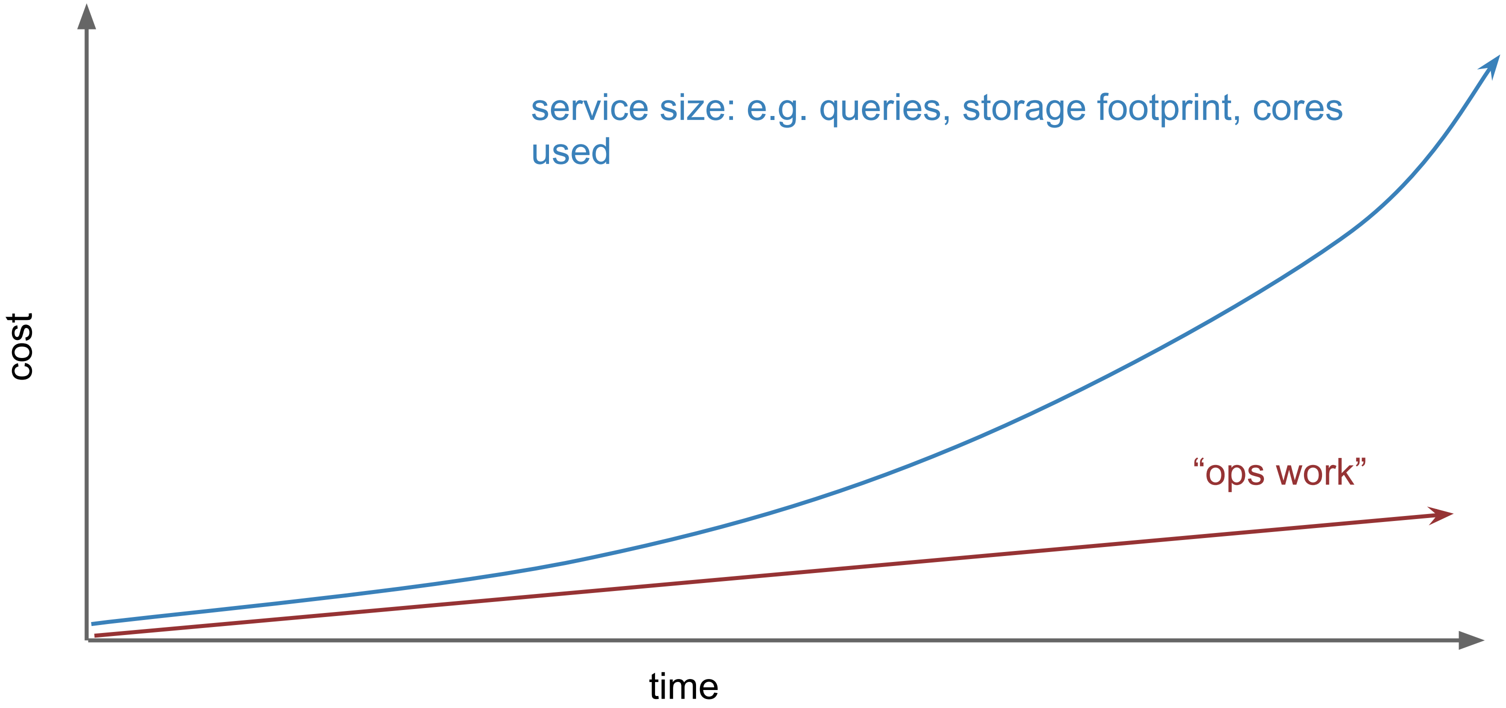
when the cost of maintenance is too high

- to improve the quality of alerts
- to improve exploratory tools

Nagios: Part of your complete ecosystem

Why *does* $X \; \forall \; X \in \{\text{Ops}\}$ suck?

the cost of maintenance must scale sublinearly
with the growth of the service

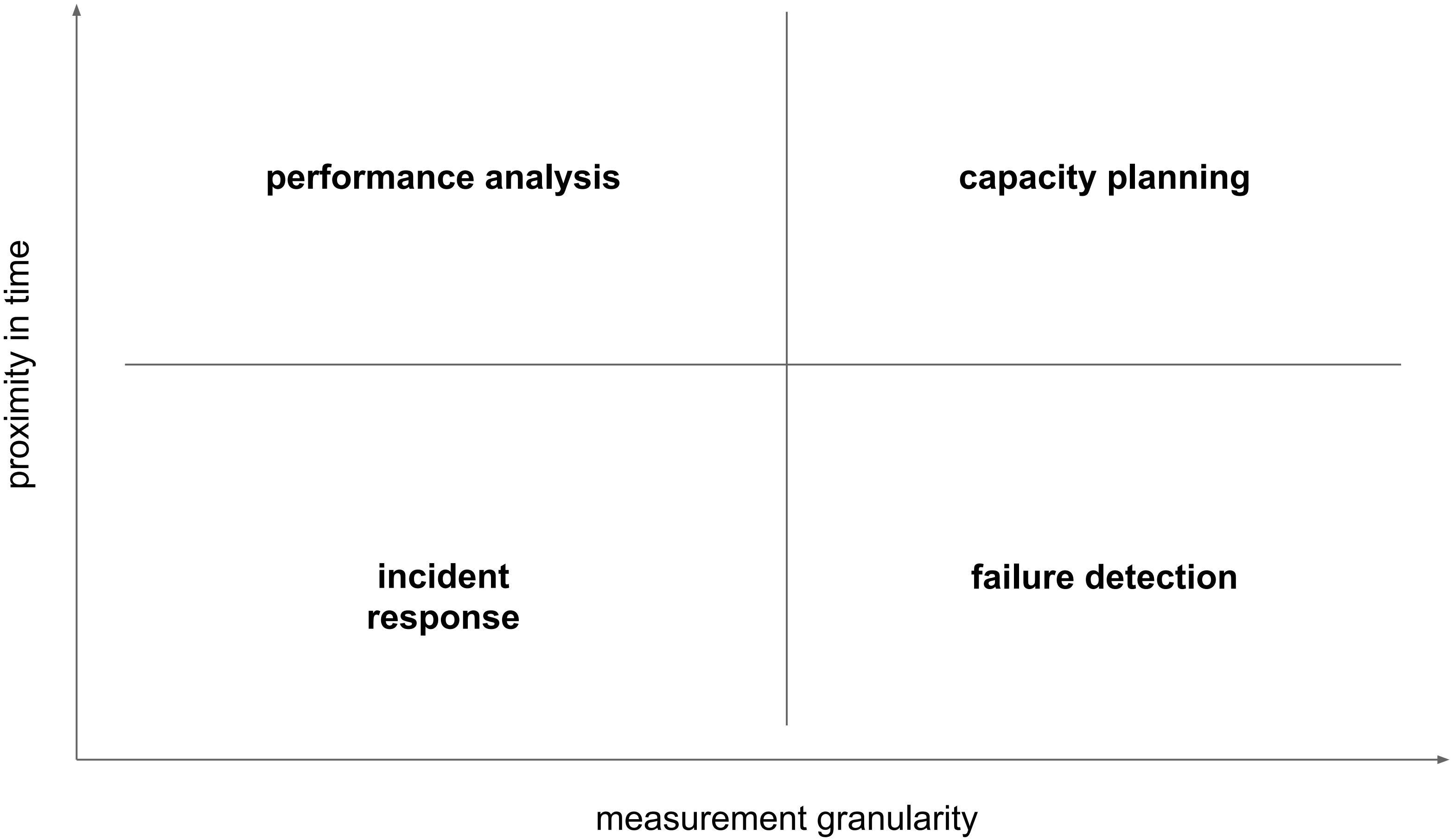

service size: e.g. queries, storage footprint, cores used
"ops work"
cost
time

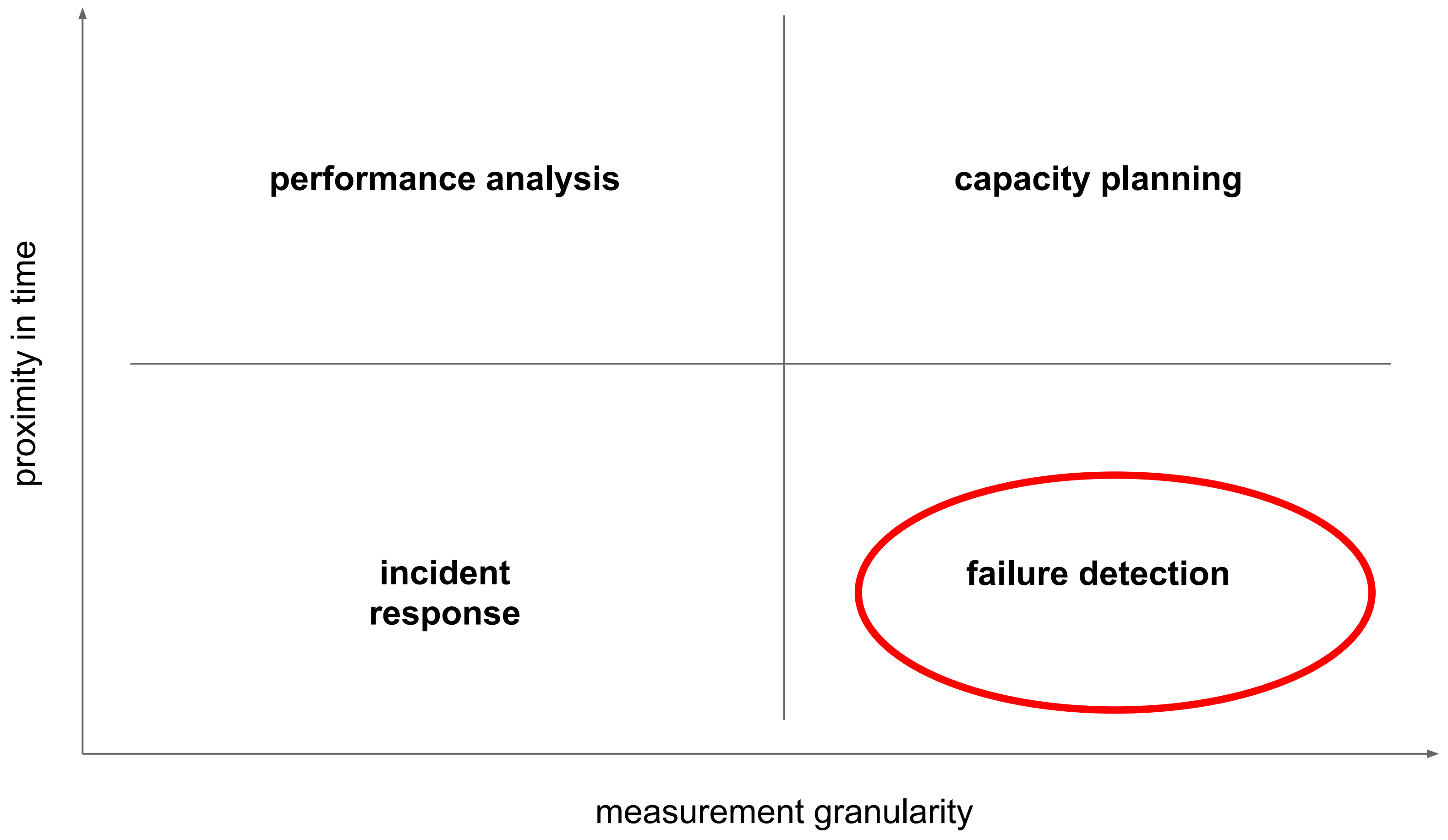
Automate yourself out of a job

- Homogenity, Configuration Management
- Abstractions, Higher level languages
- Convenient interfaces in tools
  - scriptable
  - Service Oriented Architectures

What is “monitoring”

- incident response
- performance analysis
- capacity planning
- failure detection

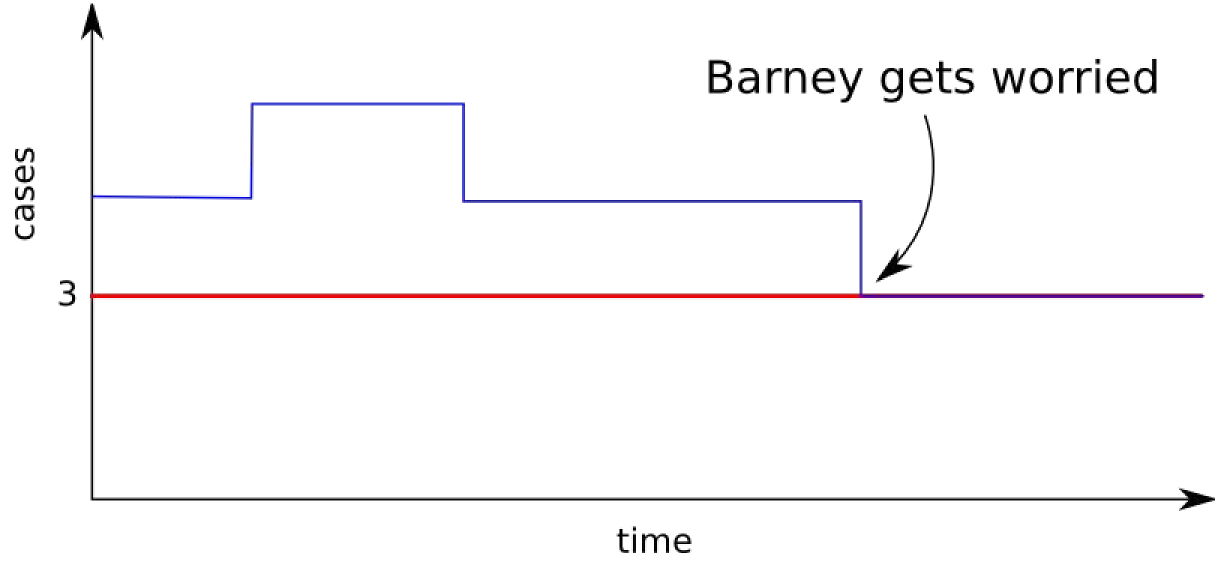

performance analysis
capacity planning
incident response
failure detection
proximity in time
measurement granularity


performance analysis
capacity planning
incident
response
failure detection
proximity in time
measurement granularity

Alerting on thresholds

Alert when the beer supply is low

$$ALERT \rightarrow \begin{cases} true & cases - 1 - 1 = 1 \\ false & otherwise \end{cases}$$



Alert when beer supply low

```
ALERT BarneyWorriedAboutBeerSupply
IF cases - 1 - 1 = 1
ANNOTATIONS {
 summary = “Hey Homer, I’m worried about the beer supply.”
 description = “After this case, and the next case, there’s only
   one case left! Yeah yeah, Oh Barney's right. Yeah, lets get
   some more beer.. yeah.. hey, what about some beer, yeah
   Barney's right…”
}
```

Disk full alert

Alert when 90% full

Different filesystems have different sizes

10% of 2TB is 200GB

False positive!

Alert on absolute space, < 500MB

Arbitrary number

Different workloads with different needs: 500MB might not be enough warning

Disk full alert

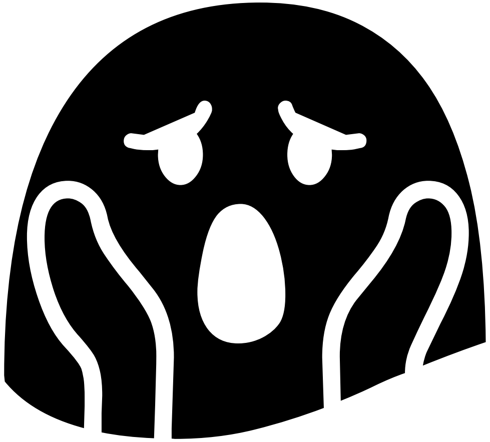
More general alert based on human interactions:

How long before the disk is full?

and

How long will it take for a human to remediate a full disk?

CALCULUS



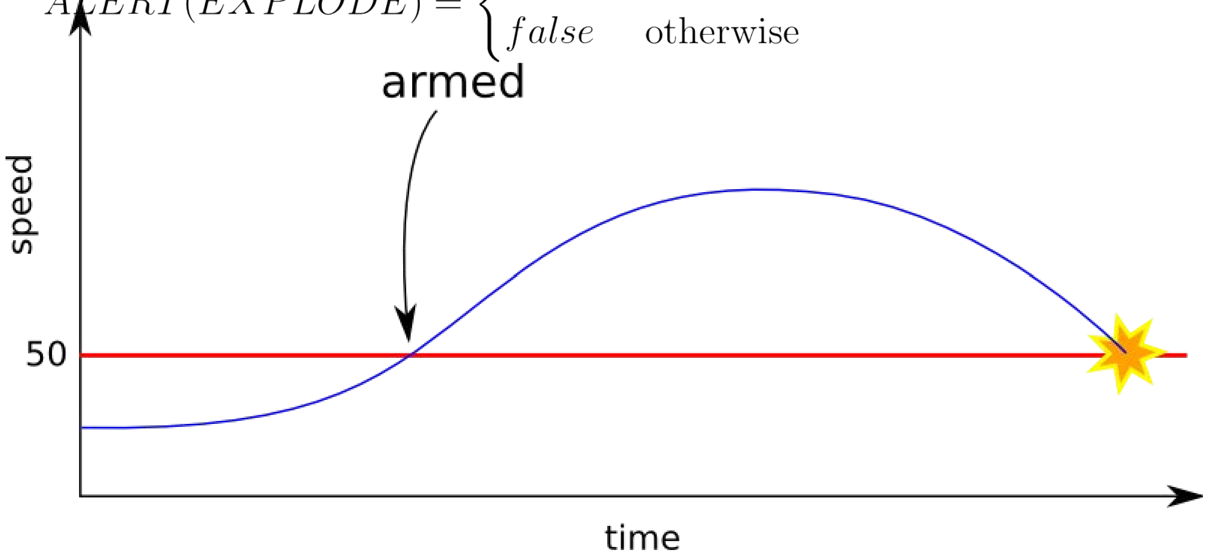
Alerting on rates of change

Dennis Hopper's Alert

$$speed\_mph = \frac{\partial miles}{\partial t}$$

$$ALERT(BombArmed) = \begin{cases} true & \text{if } speed\_mph >= 50 \\ false & \text{otherwise} \end{cases}$$

$$ALERT(EXPLODE) = \begin{cases} true & \text{if } BombArmed \cdot speed\_mph < 50 \\ false & \text{otherwise} \end{cases}$$

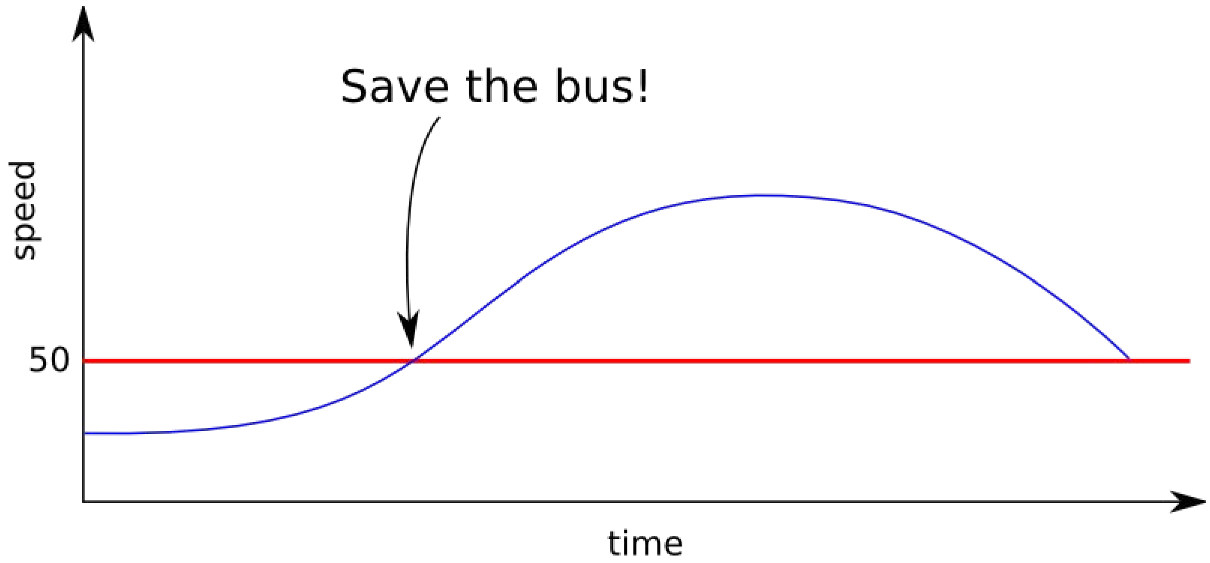


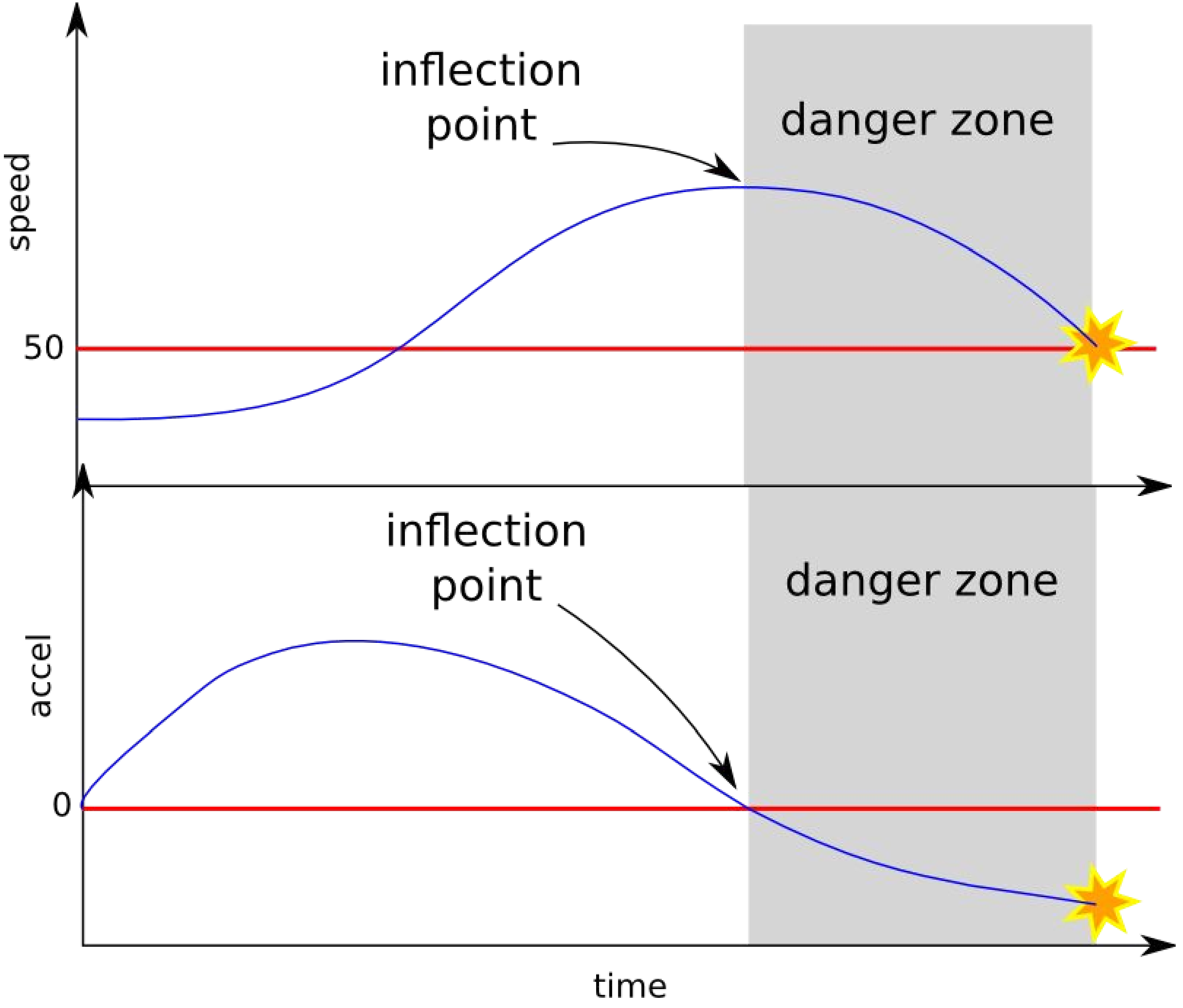
Dennis Hopper's Alert

```
ALERT BombArmed
IF speed_mph >= 50
ANNOTATIONS {
  summary = “Pop quiz, hotshot!”
}
ALERT EXPLODE
IF max(ALERTS{alertname=BombArmed,
alertstate=firing}[1d]) > 0 and speed_mph < 50
```

Keanu's Alert

$$ALERT(SaveTheBus) = \begin{cases} true & \text{if } speed\_mph >= 50 \\ false & \text{otherwise} \end{cases}$$





inflection
point
danger zone
speed
50
inflection
point
danger zone
accel
0
time

Keanu's alert

$$v - at = 50$$

$$50 - v = -at$$

$$\frac{v - 50}{a} = t$$

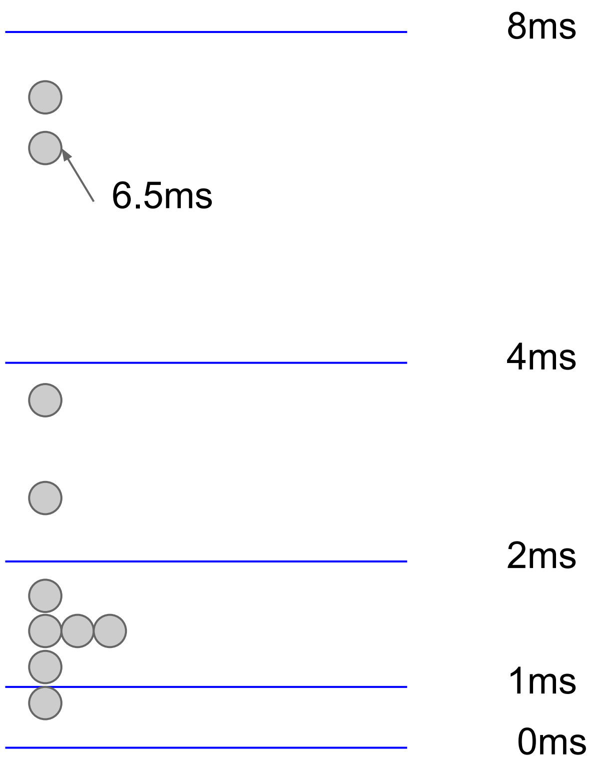
$$ALERT(...) = \begin{cases} true & \text{if } \frac{v-50}{a} <= T \\ false & \text{otherwise} \end{cases}$$

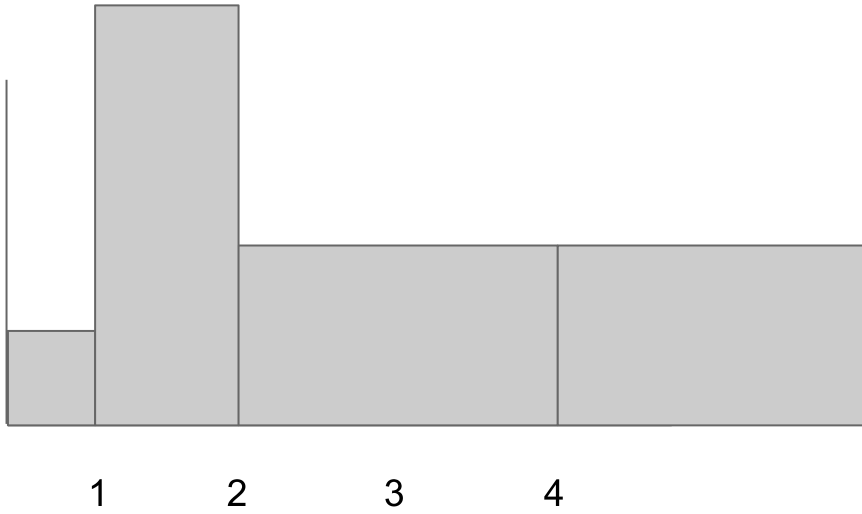
Keanu's alert

```
ALERT StartSavingTheBus
IF (v - 50)/a <= ${threshold}
```

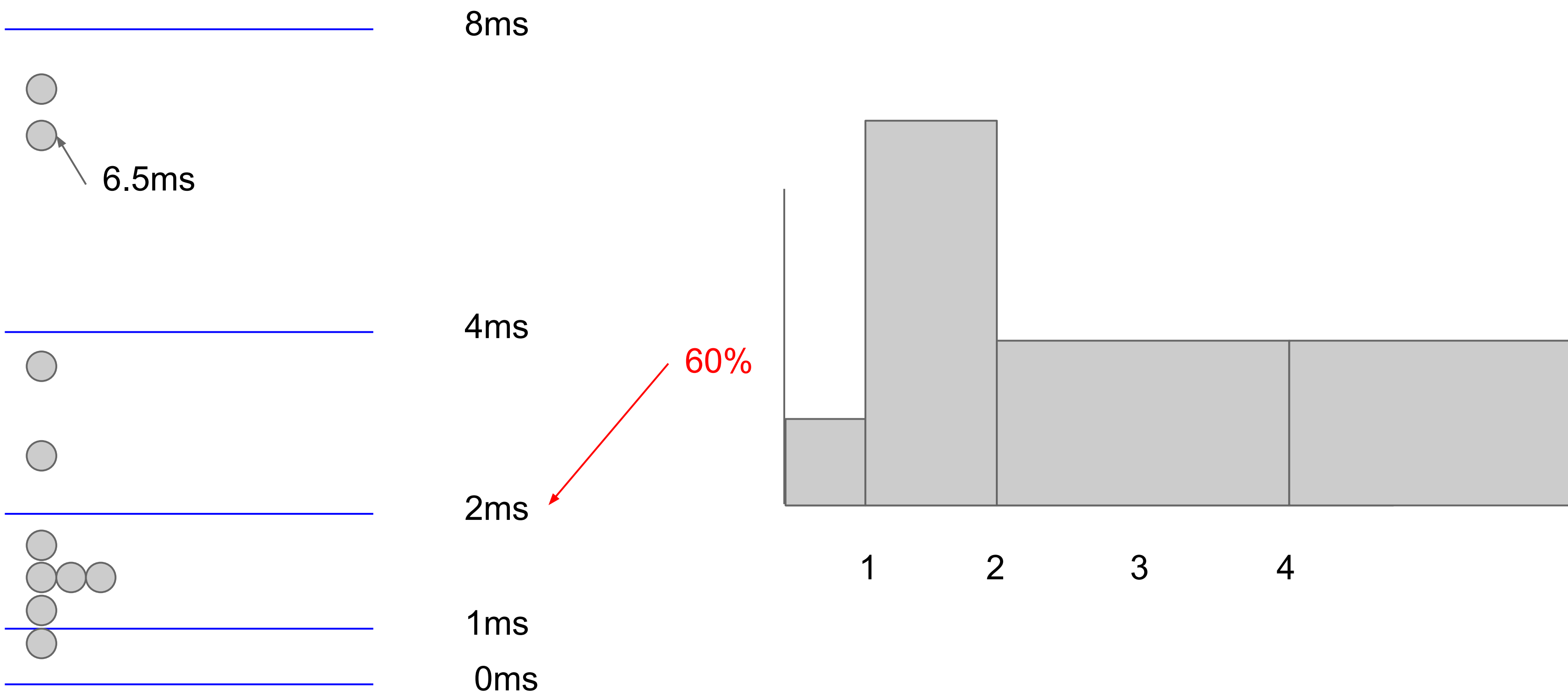
Distributions

Quantisation

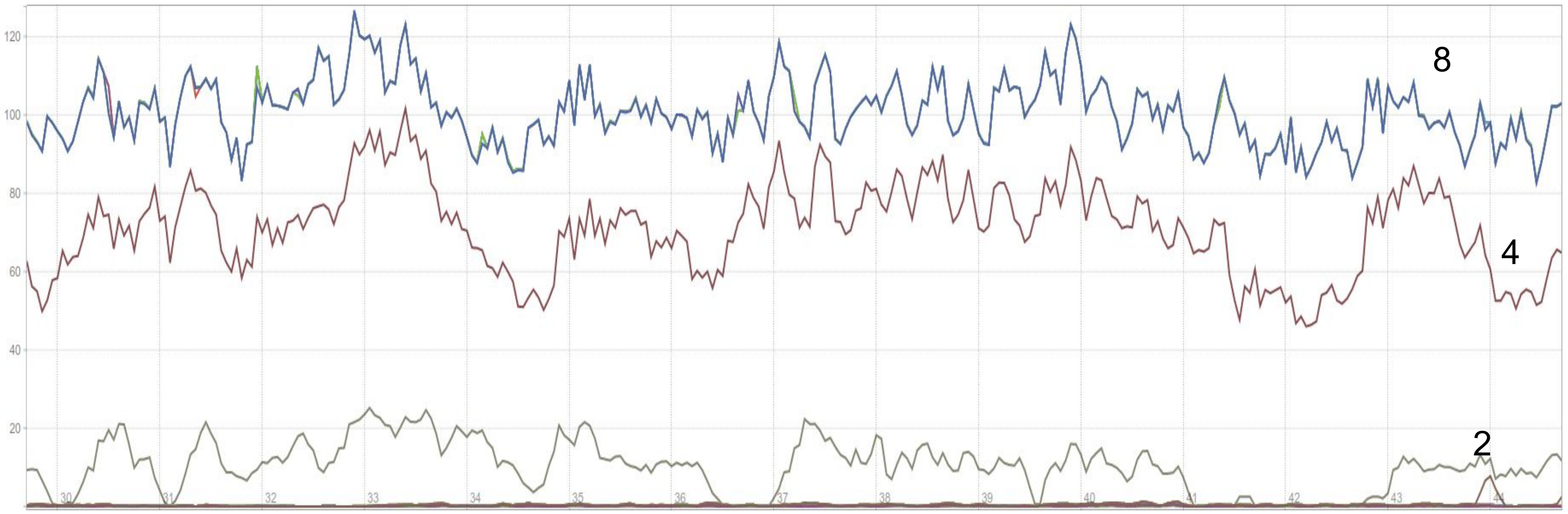




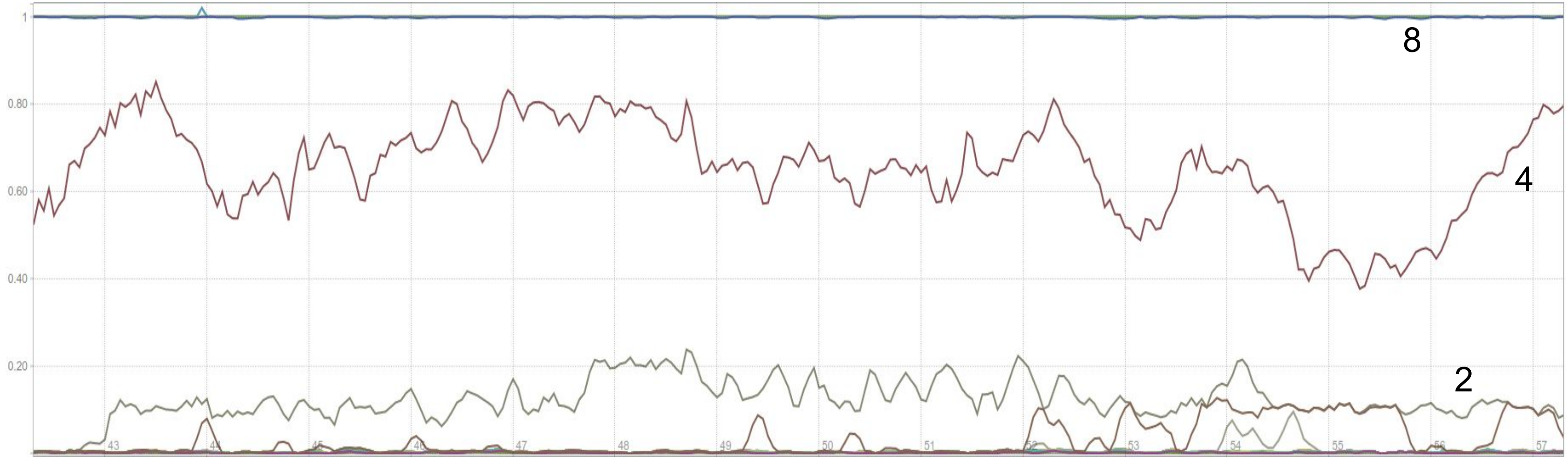
Quantisation



Rate of change in each bucket



Percentile lines of each bucket



Brian Fantana’s Alert

```
ALERT LatencyTooHigh
IF (job:latency_ms_bucket:rate10s{le=”2”}
      / on (job) group_left
    job:latency_ms_bucket:rate10s{le=”+Inf”}
) < 0.6
ANNOTATIONS {
summary=”60% of the time it works every time”
}
```

alert design

SLAs, SLOs, SLIs

- SLI → **I**ndicator: a measurement
  - response latency over 10 minutes
  - error rates over 10 minutes
- SLO → **O**bjective: a goal
  - 99.9th percentile below 5ms
  - less than 1% errors
- SLA → **A**greement: economic incentives
  - or we get paged

Clients provision against SLO

Jeff Dean, “A Reliable Whole From Unreliable Parts”

[*“Achieving Rapid Response Times in Large Online Services”*](http://research.google.com/people/jeff/Berkeley-Latency-Mar2012.pdf)

http://research.google.com/people/jeff/Berkeley-Latency-Mar2012.pdf

Error Budgets

Allowing your service some room to fail to experiment with features

The SLO is as good as your clients need, but no better.

The SLO is also as bad as necessary to prevent humans being overloaded.

“My Philosophy on Alerting”

Rob Ewaschuk

- *Every time my pager goes off, I should be able to **react with a sense of urgency**. I can only do this a few times a day before I get fatigued.*
- *Every page should be **actionable**; simply noting "this paged again" is not an action.*
- *Every page should **require intelligence** to deal with: no robotic, scriptable responses.*

“Alerts” don’t have to page you

Alerts that **do** page should indicate violations of SLO.

Put diagnostics on a console to look at when the pager goes off

- disk fullness
- task crashes
- backend slowness

http://prometheus.io

How it works

- Dynamically discover target addresses
- Scrape **/metrics** pages
  - evenly distributed load across targets
- Evaluate **rulesets** mapped to targets
  - vector arithmetic
- Send alerts
- Record to Timeseries Database (TSDB)

Prometheus Client API

```
import “github.com/prometheus/client_golang/prometheus”

var request_count =
prometheus.NewCounter(prometheus.CounterOpts{
  Name: “requests”, Help: “total requests”})

func HandleRequest … {
 …
   request_count.Add(1)
 …
```

/metrics handlers can be plain text

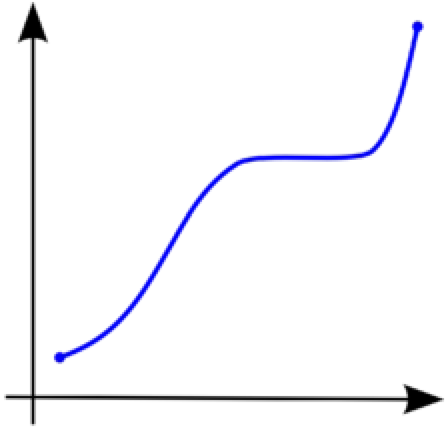
```
# HELP requests total requests
# TYPE requests counter
requests 20056
# HELP errors total errors served
# TYPE errors counter
errors{code="400"} 2027
errors{code="500"} 824
```

Timeseries Have Types

Counter: monotonically nondecreasing

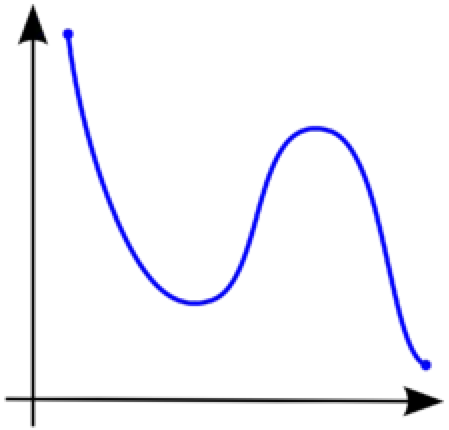
"preserves the order" i.e. UP

"nondecreasing" can be flat

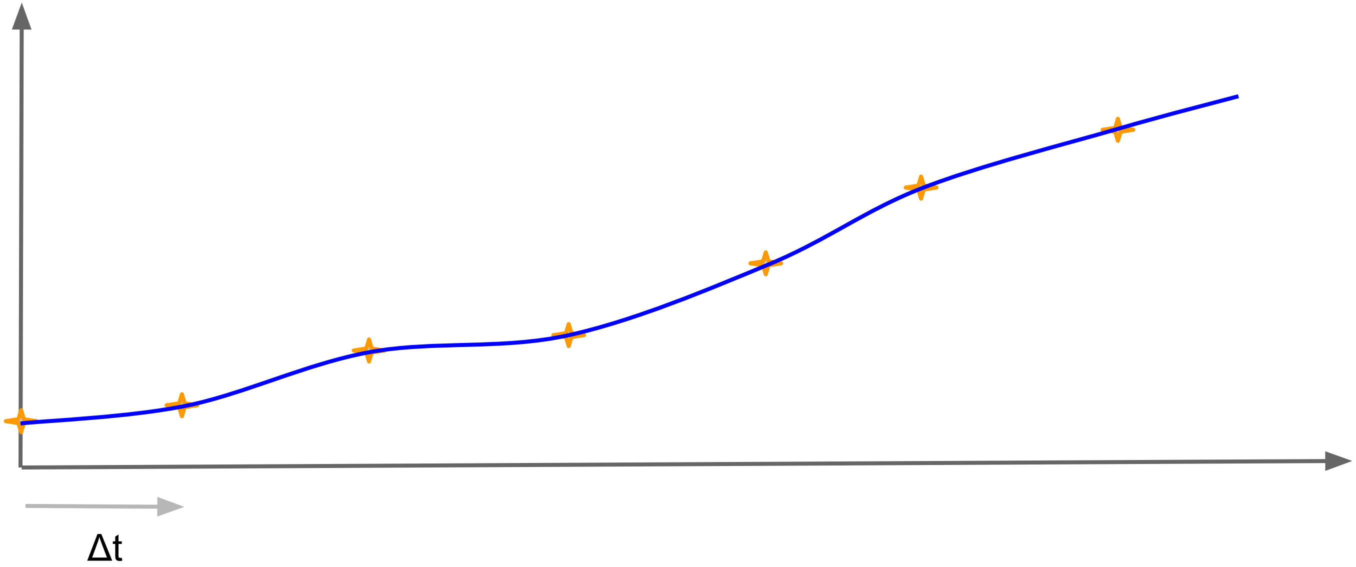


Timeseries Have Types

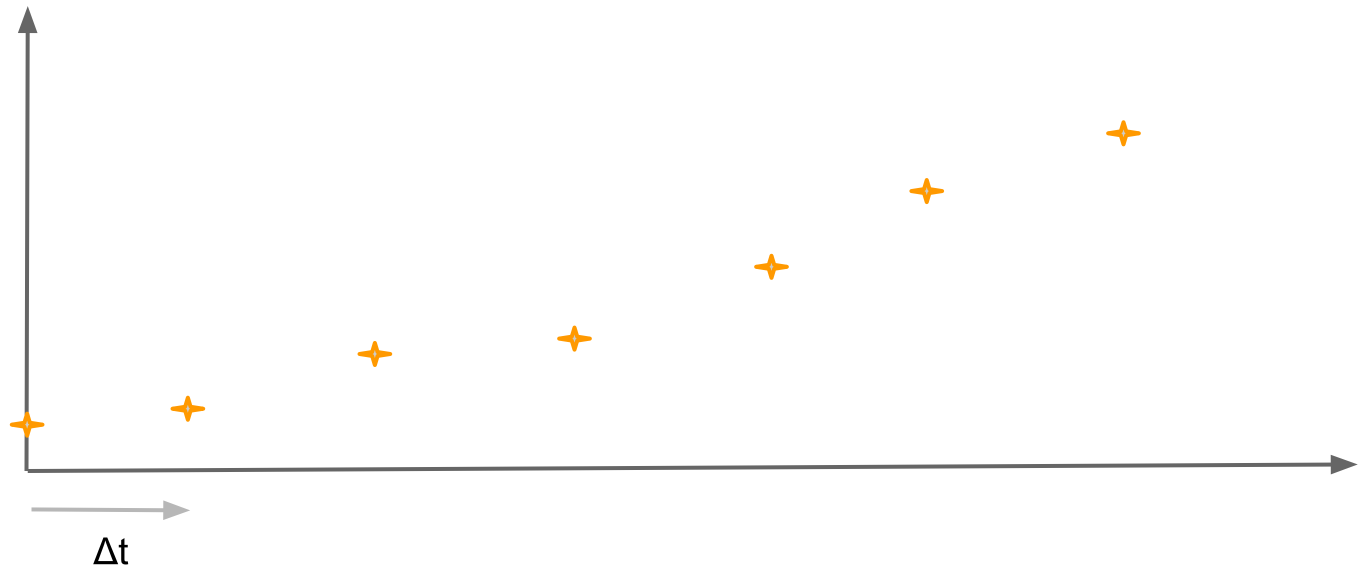
Gauge: everything else... not monotonic



Counters FTW

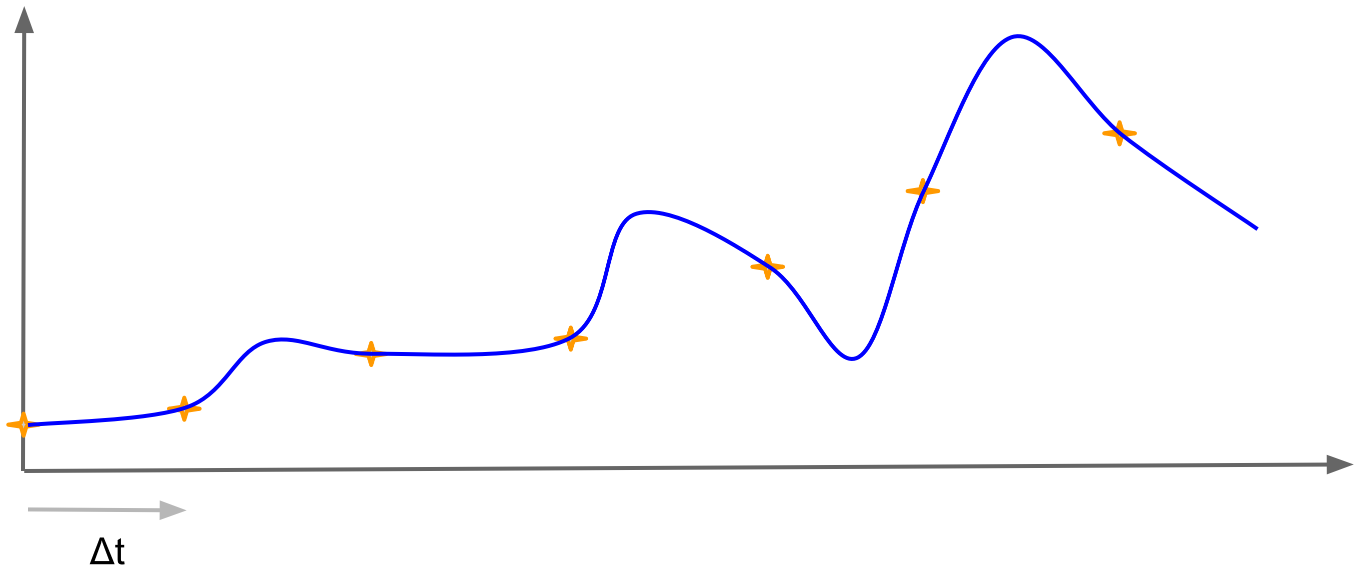


Counters FTW

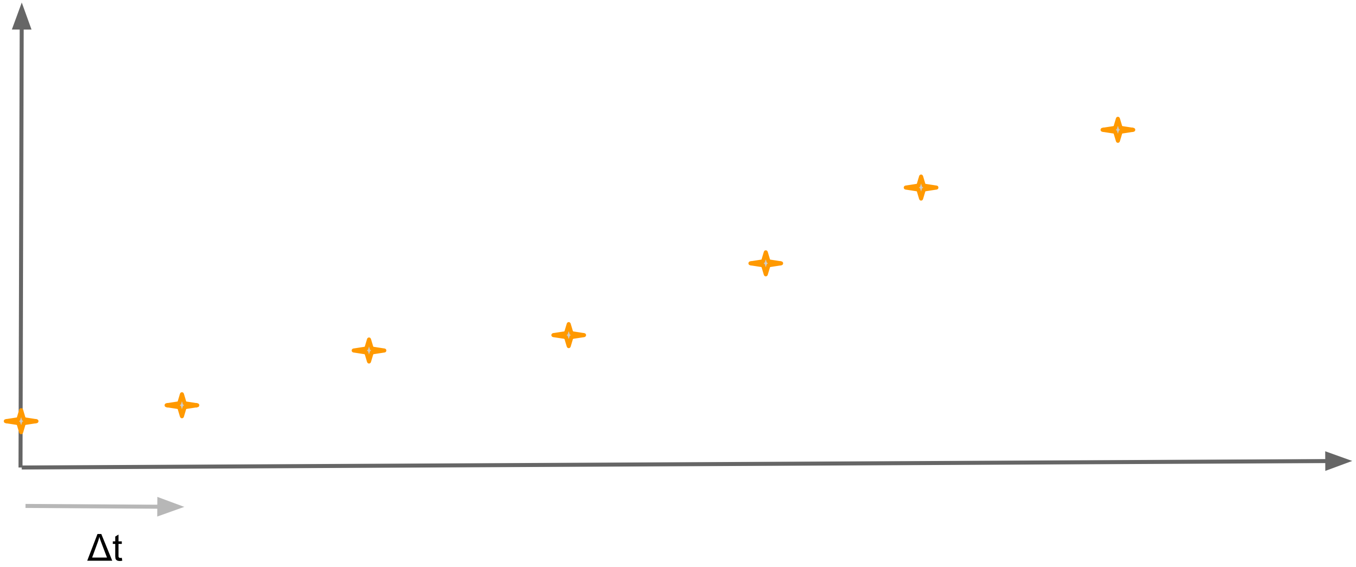


no loss of meaning after sampling

Gauges FTL

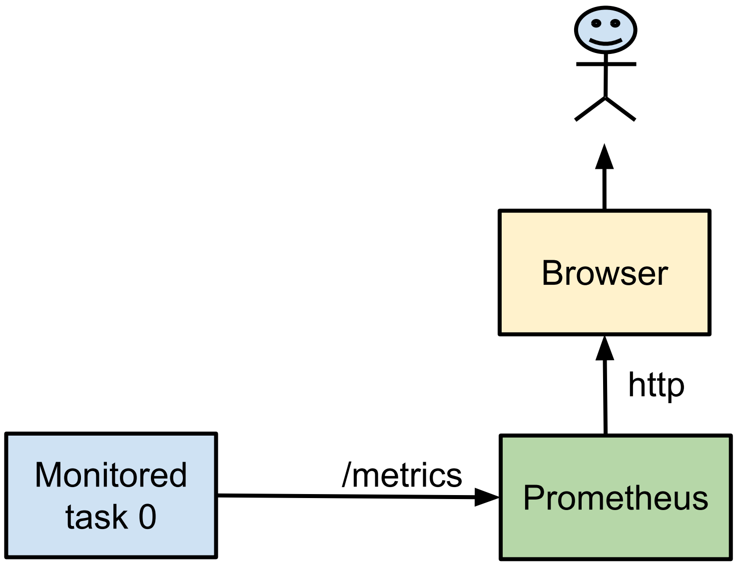


Gauges FTL

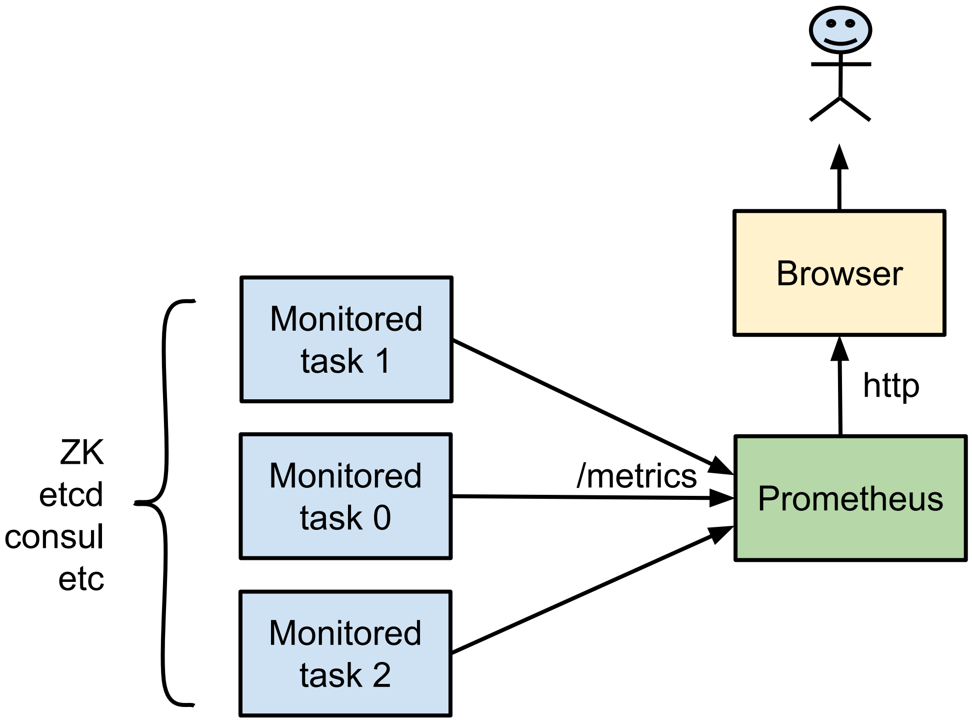


lose spike events shorter than sampling interval

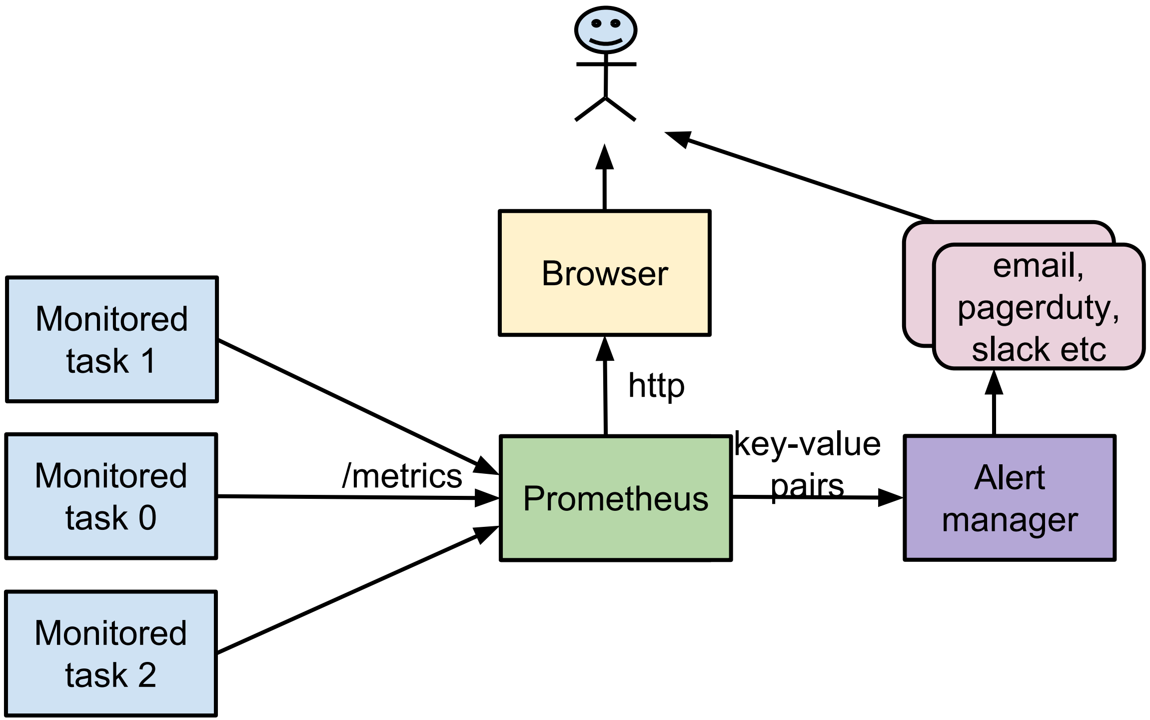
Process Overview



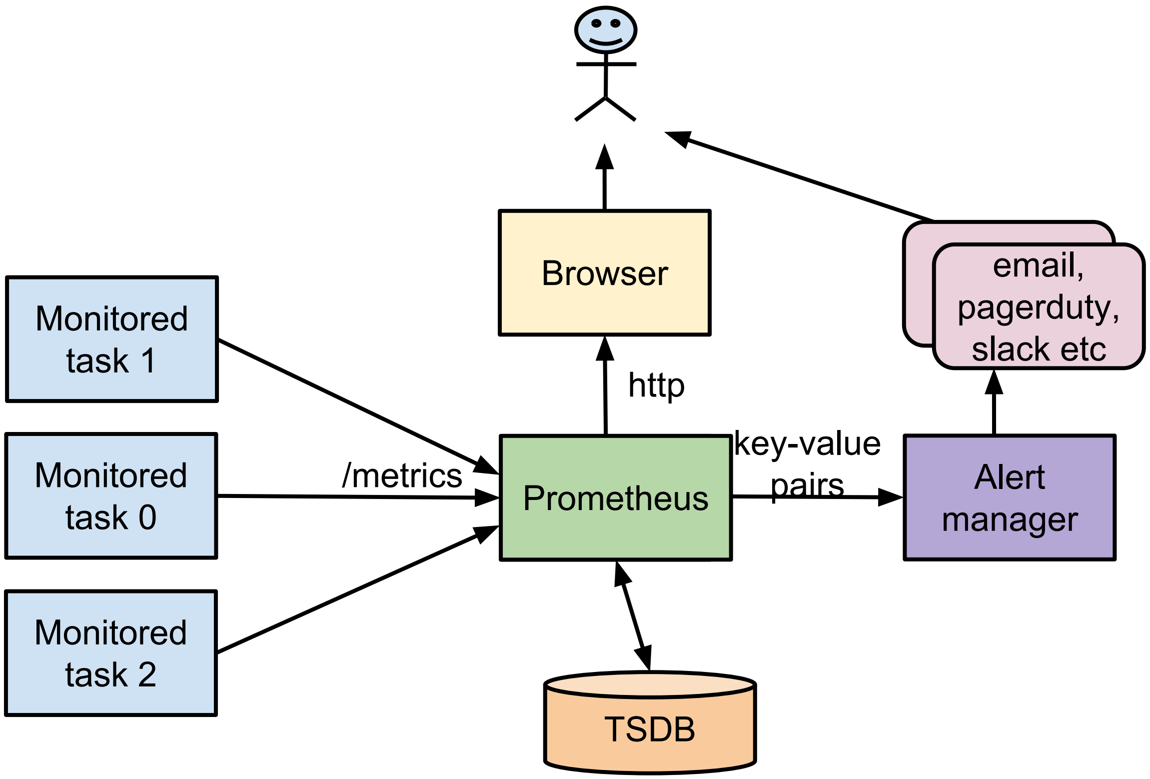
Service Discovery



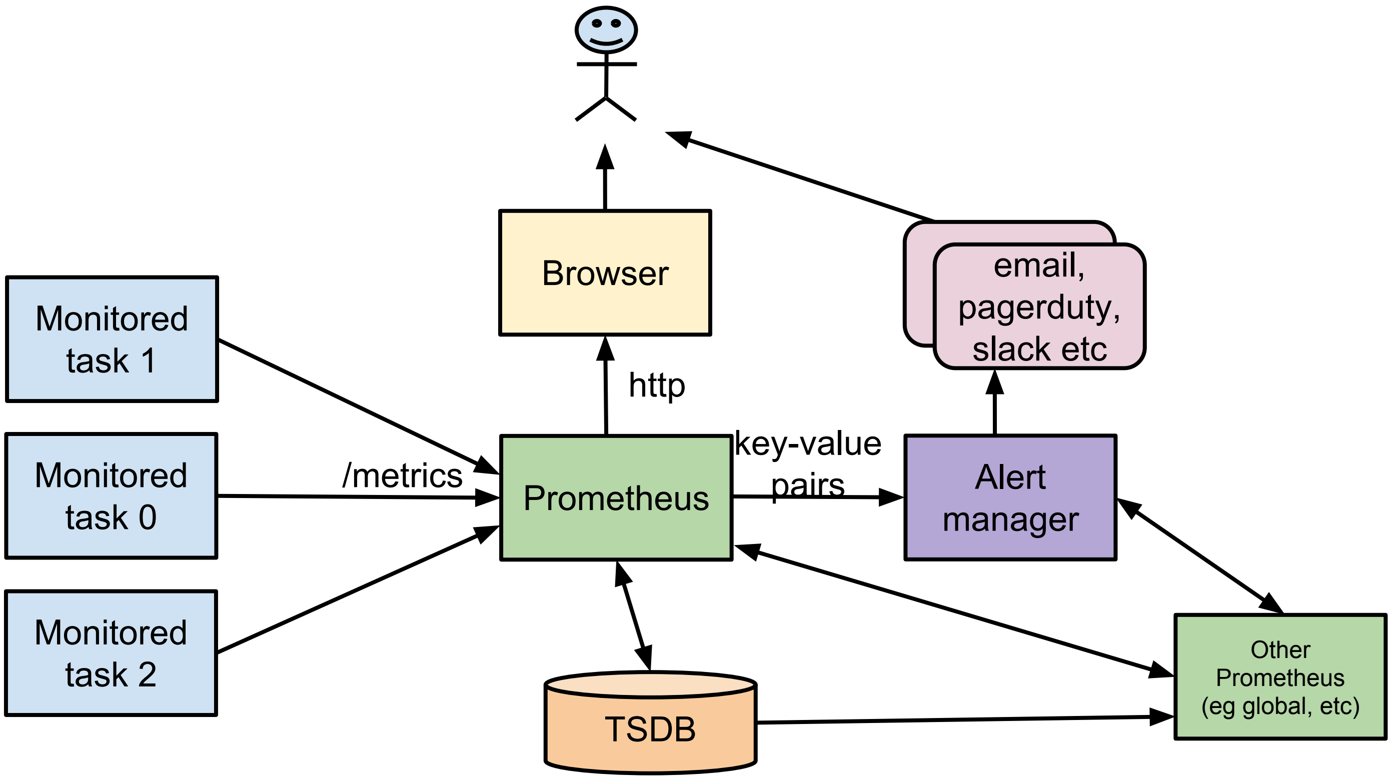
Alert Notifications



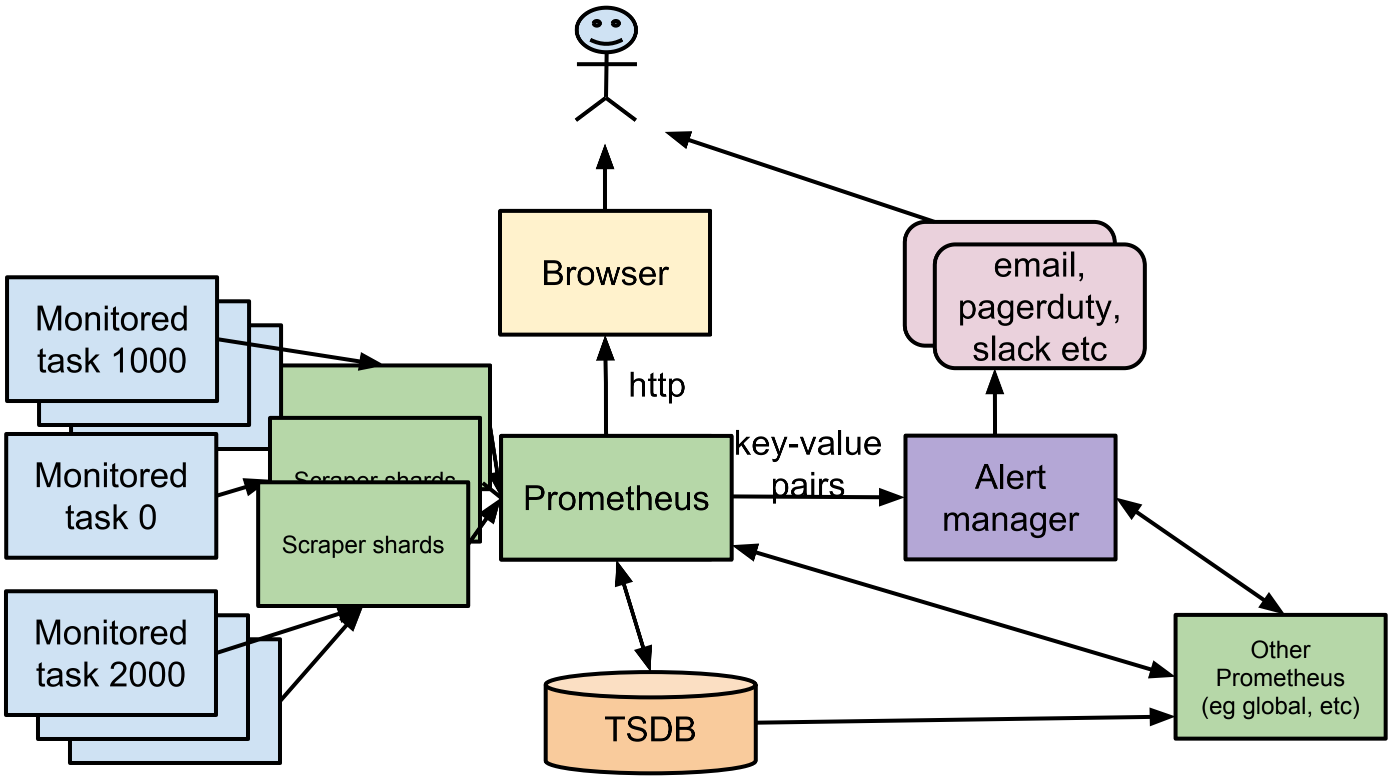
Long-term storage



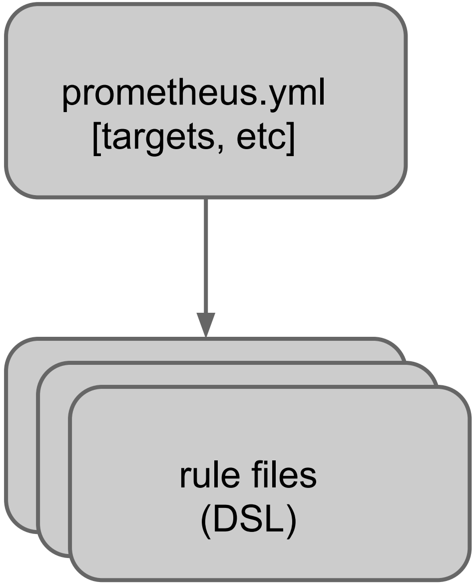
Global & other monitoring



Sprinkle some shards on it



Configuring Prometheus



Configuring Prometheus

prometheus.yml:

```
global:
  scrape_interval: 1m
  labels:  # Added to all targets
    zone: us-east
rule_files:
  [ - <filepath> ... ]
scrape_configs:
  [ - <scrape_config> ... ]
```

Finding Targets

```
scrape_configs:
 - job_name: “smtp”
   static_configs:
   - targets:
     - ‘mail.example.com:3903’

 - job_name: “barserver”
   file_sd_configs:
   - [json_filenames generated by, e.g. puppet]

 - job_name: “webserver”
   dns_sd_configs:
   - names: # DNS SRV lookup
     - web.example.com

 - job_name: “fooserver”
   consul_sd_configs: # autodiscovery from consul queries
```

Labels & Vectors

Data Storage Requirements

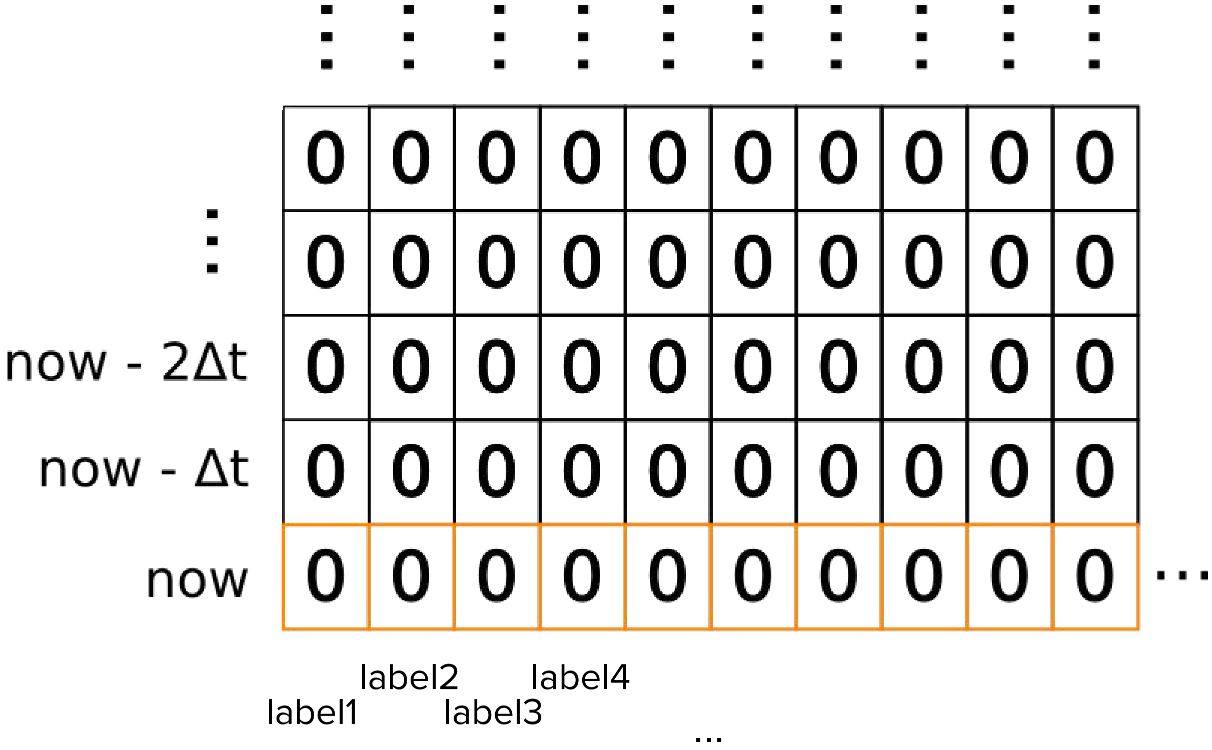
- A 'service' can consist of:
  - multiple processes running many operations
  - multiple machines
  - multiple datacenters

- The solution needs to:
  - Keep high-dimension data organized
  - Allow various aggregation types (max, average, percentile)
  - Allow flexible querying and slicing of data (by machine, by datacenter, by error type, etc)

The timeseries arena

- Data is stored in one global database in memory (checkpointed to disk)
- Each data point has the form: (timestamp, value)
- Data points are stored in chronological lists called timeseries.
- Each timeseries is named by a set of unique labels, of the form name=value
- Timeseries data can be queried via a variable reference (a specification of labels and values).
  - The result is a vector or matrix.

Structure of timeseries



Variables and Labels

Labels come from

- the target's name: `job`, `instance`
- the target's exported metrics
- the configuration: `labels`, `relabels`
- the processing rules

Variables and labels

```
{var=”errors”,job=”web”,instance=”server01:8000”,zone=”us-east”,code=”500”}  16
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}   0
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-east”,code=”500”}  12
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}  10
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-east”}        50456
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-west”}        12432
{var=”requests”,job=”web”,instance=”server02:8080”,zone=”us-west”}        43424
```

Variables and labels

```
{var=”errors”,job=”web”,instance=”server01:8000”,zone=”us-east”,code=”500”}  16
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}   0
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-east”,code=”500”}  12
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}  10
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-east”}        50456
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-west”}        12432
{var=”requests”,job=”web”,instance=”server02:8080”,zone=”us-west”}        43424
```

errors{job=”web”}

Variables and labels

```
{var=”errors”,job=”web”,instance=”server01:8000”,zone=”us-east”,code=”500”}  16
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}   0
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-east”,code=”500”}  12
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}  10
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-east”}        50456
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-west”}        12432
{var=”requests”,job=”web”,instance=”server02:8080”,zone=”us-west”}        43424
```

errors{job=”web”,zone=”us-west”}

Single-valued Vector

```
{var=”errors”,job=”web”,instance=”server01:8000”,zone=”us-east”,code=”500”}  16
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}   0
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-east”,code=”500”}  12
{var=”errors”,job=”web”,instance=”server01:8080”,zone=”us-west”,code=”500”}  10
{var=”errors”,job=”web”,instance=”server02:8080”,zone=”us-west”,code=”500”}  10
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-east”}        50456
{var=”requests”,job=”web”,instance=”server01:8080”,zone=”us-west”}        12432
{var=”requests”,job=”web”,instance=”server02:8080”,zone=”us-west”}        43424
```

```
errors{job=”web”,zone=”us-east”,
instance=”server01:8000”,code=”500”}
```

rule evaluation

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

```
requests{instance="localhost:8001",job="web"} 21235 21244

requests{instance="localhost:8005",job="web"} 21211 21222

→

task:requests:rate10s{instance="localhost:8007",job="web"} 8.777777777777779

task:requests:rate10s{instance="localhost:8009",job="web"} 10.222222222222223
```

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“variable reference”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“range expression”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“function”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“recorded variable”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“level”

recording rules

```
dc:requests:rate10s =
  sum without (instance)
    (task:requests:rate10s)
```

“level”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“operation”

recording rules

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])
```

“name”

aggregation based on topology

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])

dc:requests:rate10s =
  sum without (instance)(
    task:requests:rate10s)

global:requests:rate10s =
  sum without (zone)(dc:requests:rate10s)
```

aggregation based on topology

```
task:requests:rate10s =
  rate(requests{job=”web”}[10s])

dc:requests:rate10s =
  sum without (instance)(
    task:requests:rate10s)

global:requests:rate10s =
  sum without (zone)(dc:requests:rate10s)
```

relations based on schema

```
dc:errors:ratio_rate10s =
  sum by (job)(dc:errors:rate10s)
    / on (job)
  dc:requests:rate10s
```

relations based on schema

```
dc:errors:ratio_rate10s =
  sum by (job)(dc:errors:rate10s)
    / on (job)
  dc:requests:rate10s
```

relations based on schema

```
dc:errors:ratio_rate10s =
  dc:errors:rate10s
    / on (job) group_left
  dc:requests:rate10s
```

Demo

http://github.com/jaqx0r/blts

Recap

- Use “higher level abstractions” to lower cost of maintenance
- Use metrics, not checks, to get Big Data
- Design alerts based on Service Objectives

Fin

jaq@google.com

http://prometheus.io
http://github.com/jaqx0r/blts

"My Philosophy on Alerting"

"Achieving Rapid Response Times in Large Online Services"

Question Time