# Prometheus: A practical guide to alerting at scale

Jamie Wilkinson <jaq@google.com>

# Prometheus: Yet Another Effing Stream Processor

*YAESP*

# Prometheus: A practical guide to alerting at scale

Jamie Wilkinson <jaq@google.com>


THE SEARCH FOR OUR BEGINNING
COULD LEAD TO OUR END
PROMETHEUS
6.08.12
IMAX 3D


# WE'RE HIRING

http://prometheus.io

# Who am I?

- 1999 became aware of monitoring
- 2002 deployed Nagios for the first time
- 2004 tried to build my own
- 2006 disappeared into black hole
- 2007 first learned about Borgmon
- 2009 finally proficient in Borgmon
- 2012 PuppetConf 2012 on “timeseries alerting”

# TRIGGER WARNING

High school calculus

# TRIGGER WARNING

Material you've already seen

# TRIGGER WARNING

Material I’ve used before

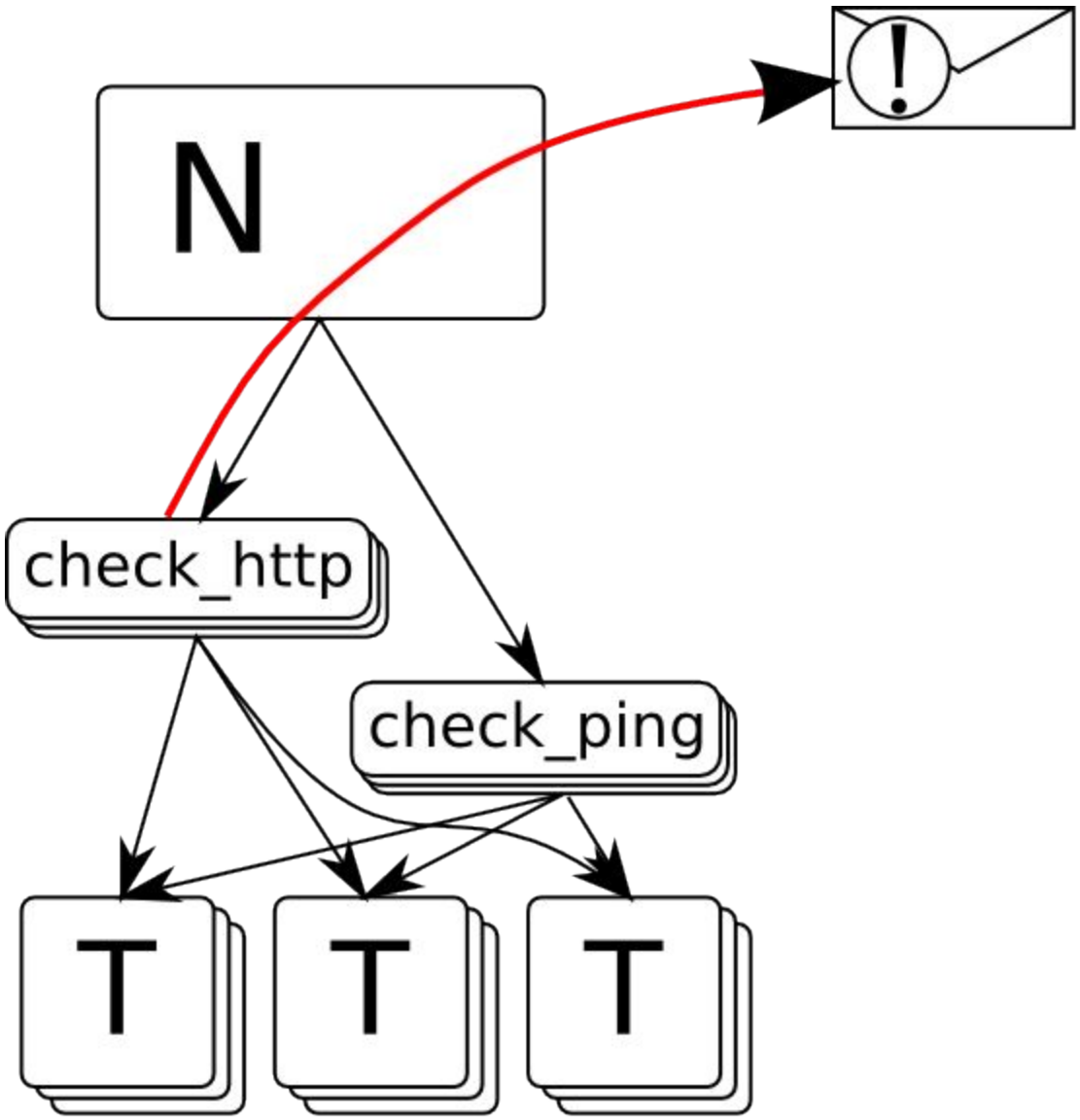
# TRIGGER WARNING

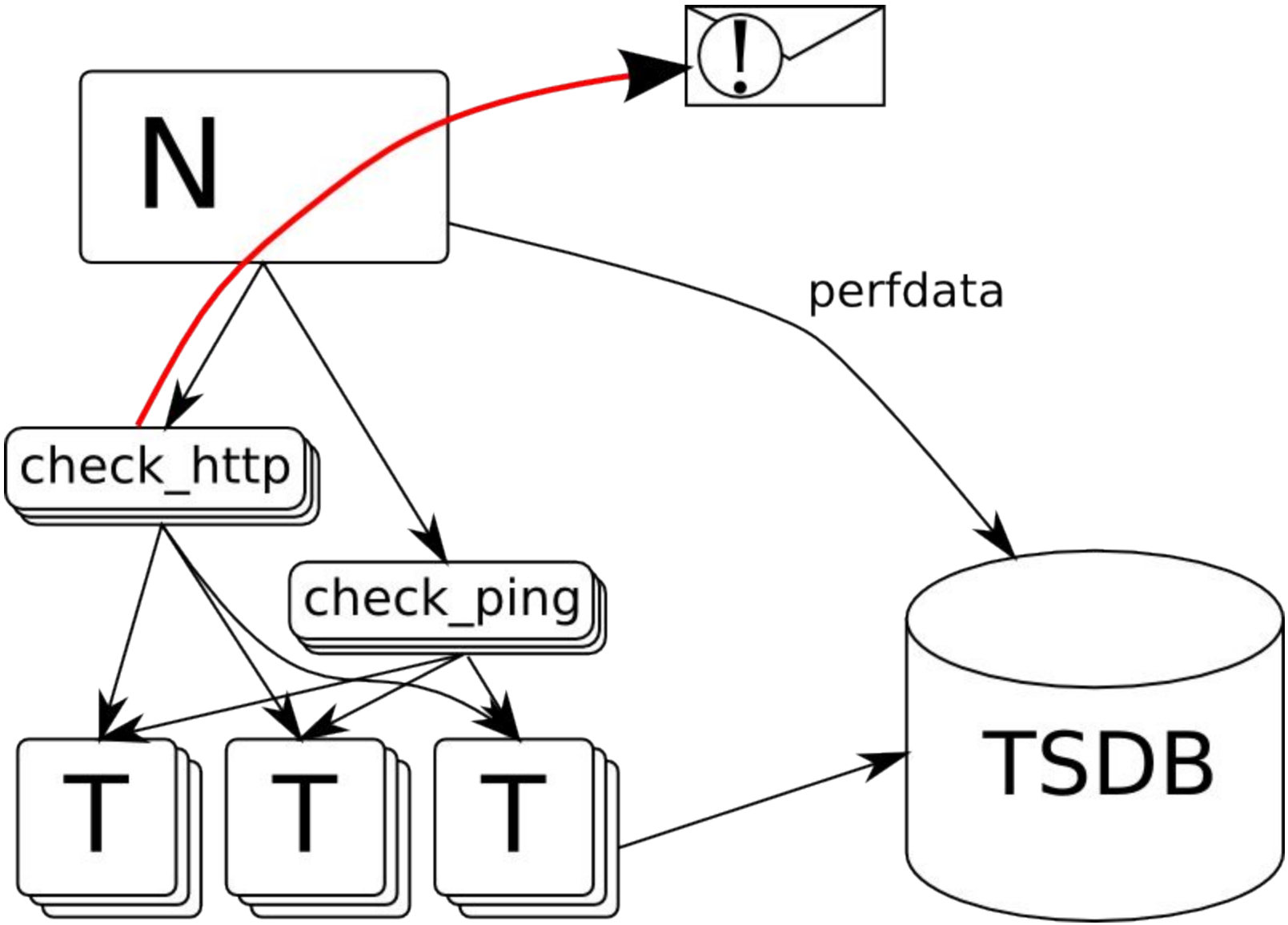
Borgmon

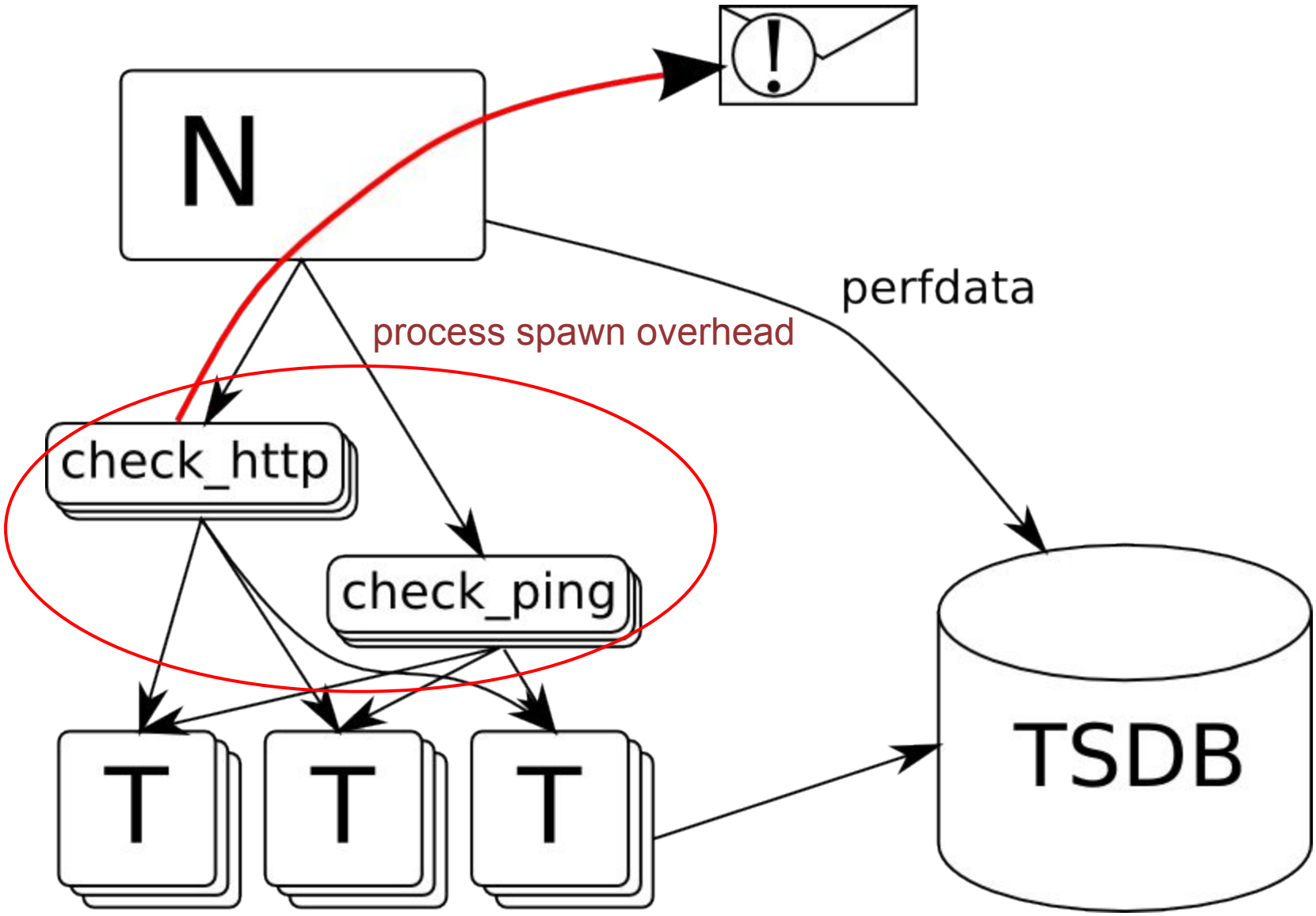
# Questions?

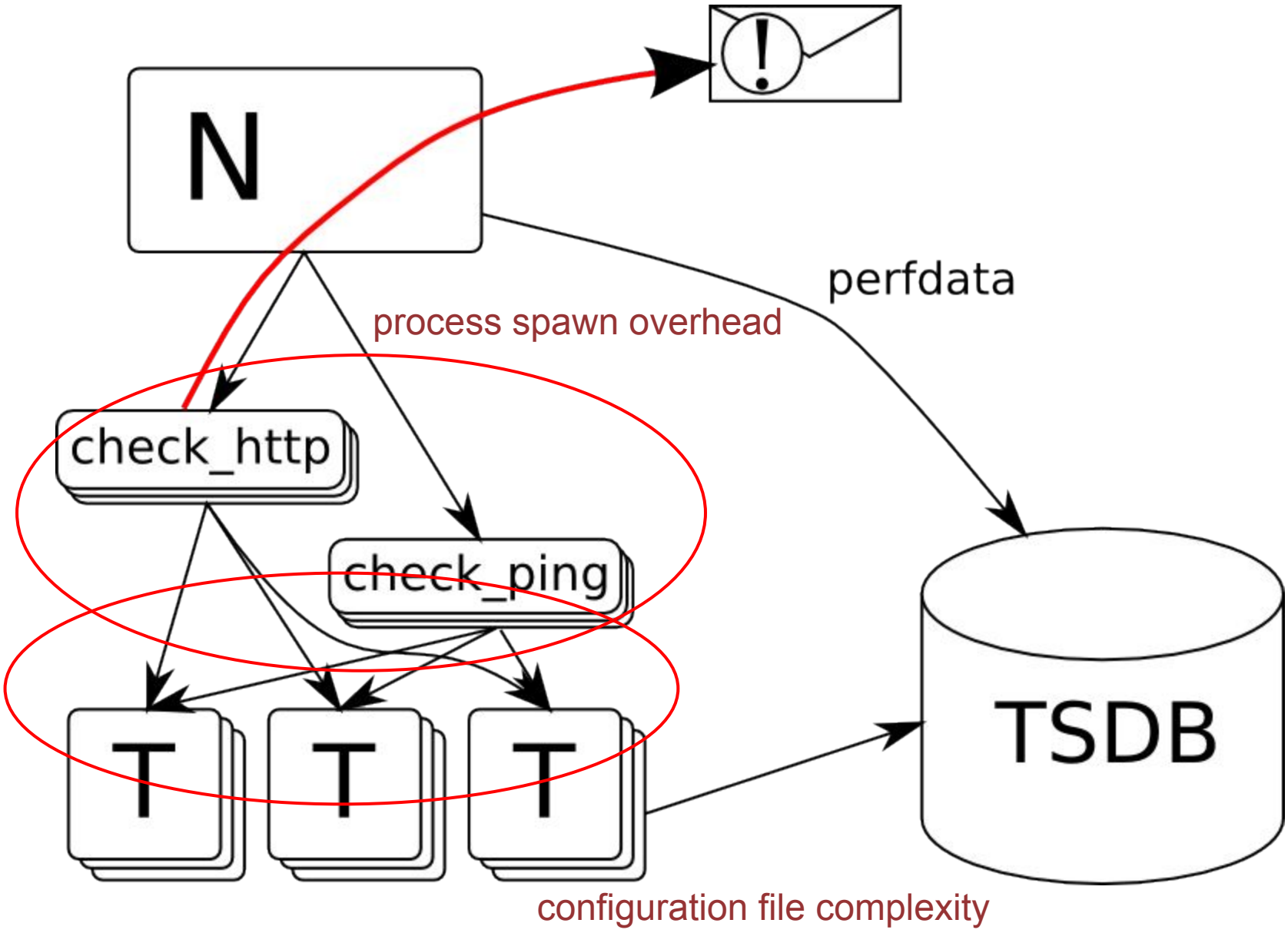
# The traditional* model of monitoring

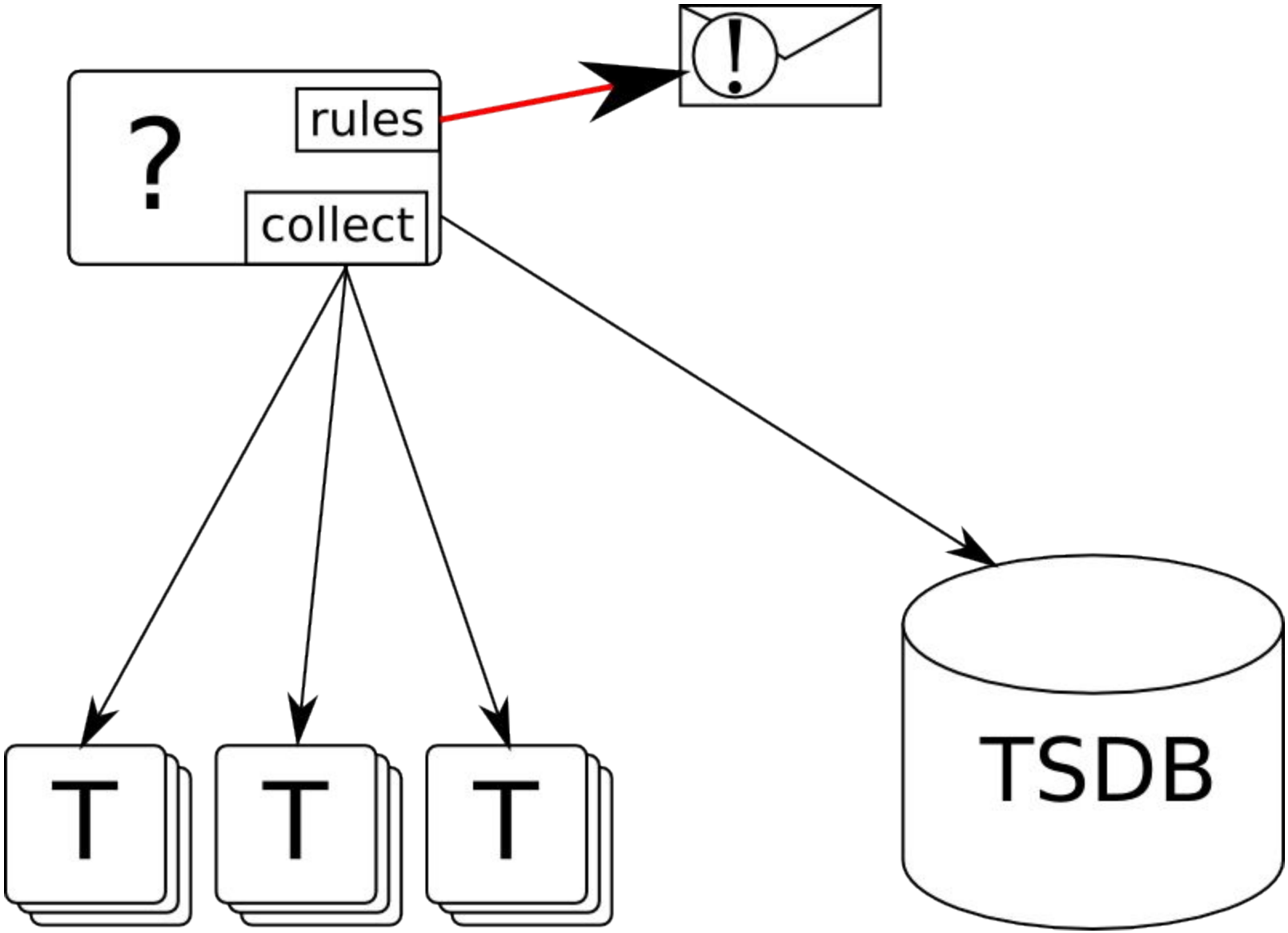
1. blackbox monitoring resulting in alerts
2. whitebox monitoring resulting in charts


N
!
check_http
check_ping
T
T
T


N
!
perfdata
check_http
check_ping
T
T
T
TSDB


N
!
perfdata
process spawn overhead
check_http
check_ping
T
T
T
TSDB


N
!
perfdata
process spawn overhead
check_http
check_ping
T
T
T
TSDB
configuration file complexity


?
rules
collect
!
T
T
T
TSDB

# Large-scale cluster management at Google with Borg

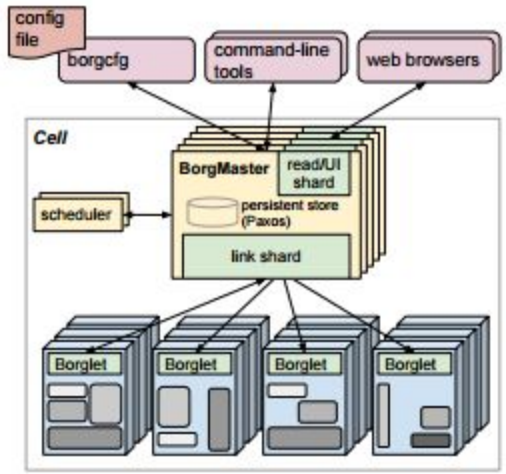
Abhishek Verma[†] Luis Pedrosa[‡] Madhukar Korupolu
David Oppenheimer Eric Tune John Wilkes
Google Inc.

## Abstract

Google's Borg system is a cluster manager that runs hundreds of thousands of jobs, from many thousands of different applications, across a number of clusters each with up to tens of thousands of machines.

It achieves high utilization by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation. It supports high-availability applications with runtime features that minimize fault-recovery time, and scheduling policies that reduce the probability of correlated failures. Borg simplifies life for its users by offering a declarative job specification language, name service integration, real-time job monitoring, and tools to analyze and simulate system behavior.

We present a summary of the Borg system architecture and features, important design decisions, a quantitative anal-



# Borgmon is

- a metric collector
- an in-memory timeseries database
- a interactive query tool
- a programmable calculator for discrete multidimensional vector streams
- something I've been wanting to discuss outside of Google for 8 years
- and, the inspiration for Prometheus

# Borgmon is not

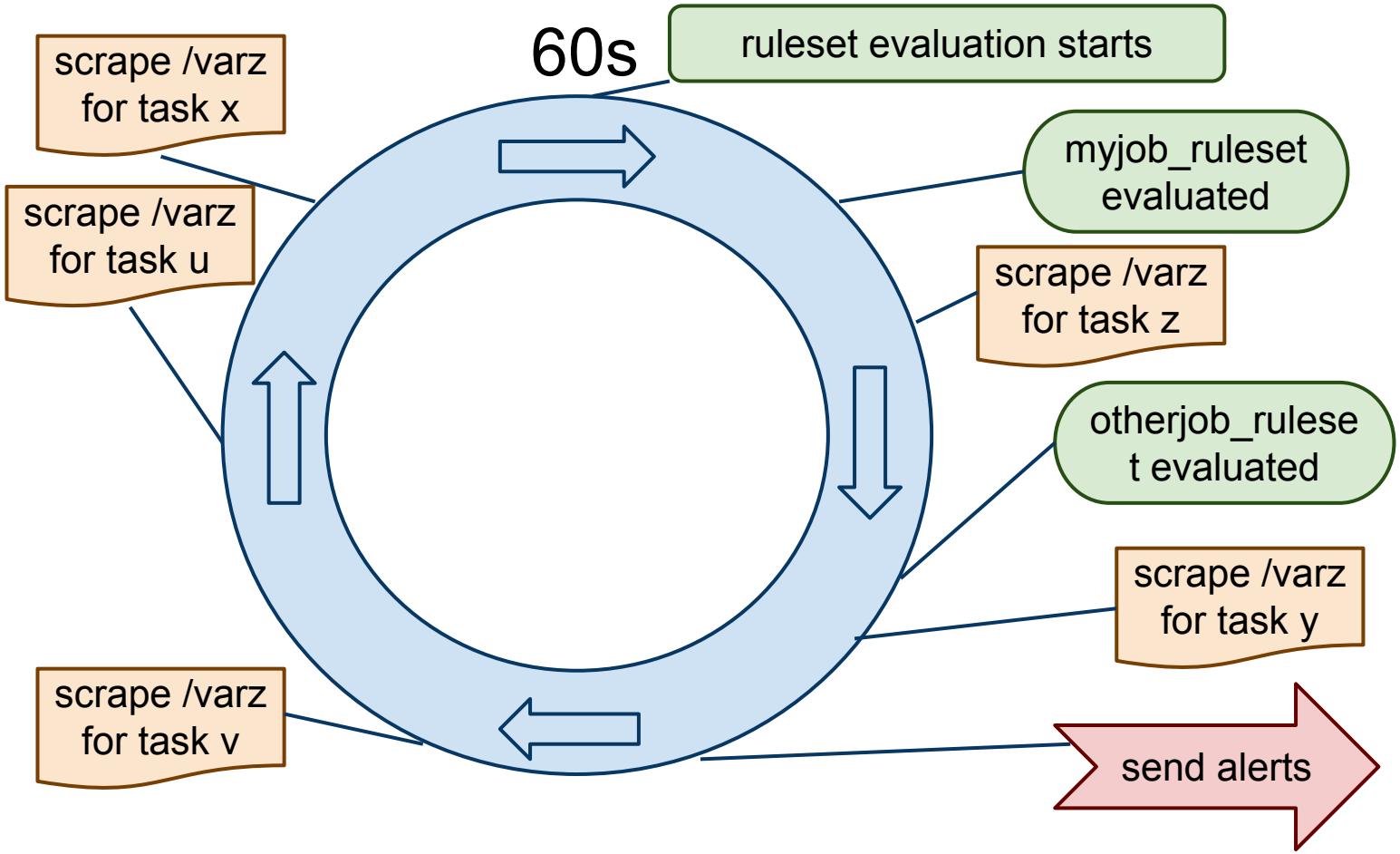
- a system profiler
- an exception catcher
- a load tester
- a log processor

… but you can hook it up to those things

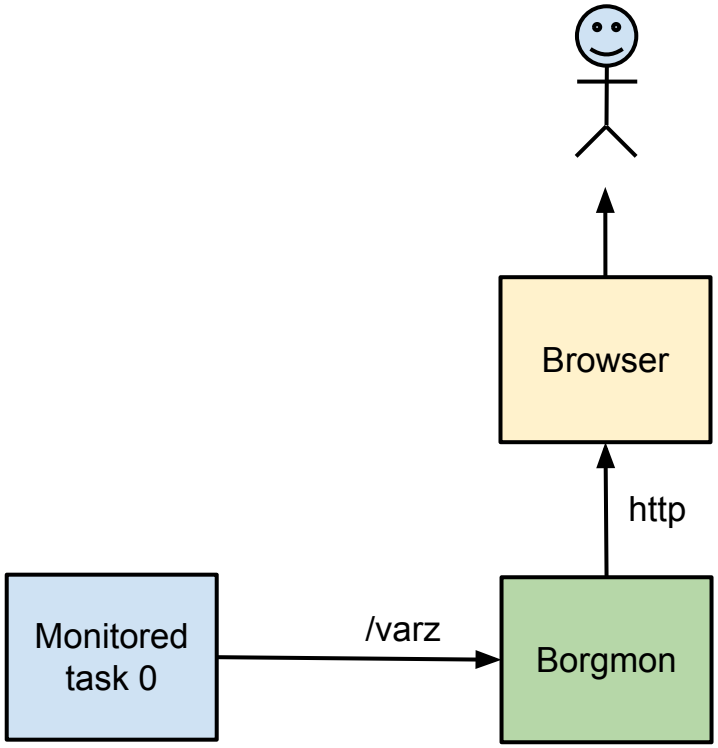
# How it works

- Scrape plain text **/varz** pages
  - randomly distributed across targets
  - read values into memory
- Evaluate rulesets, vector arithmetic
  - typically every 10 or 60 seconds, starting all at once
- Send alerts
- Record to TSDB

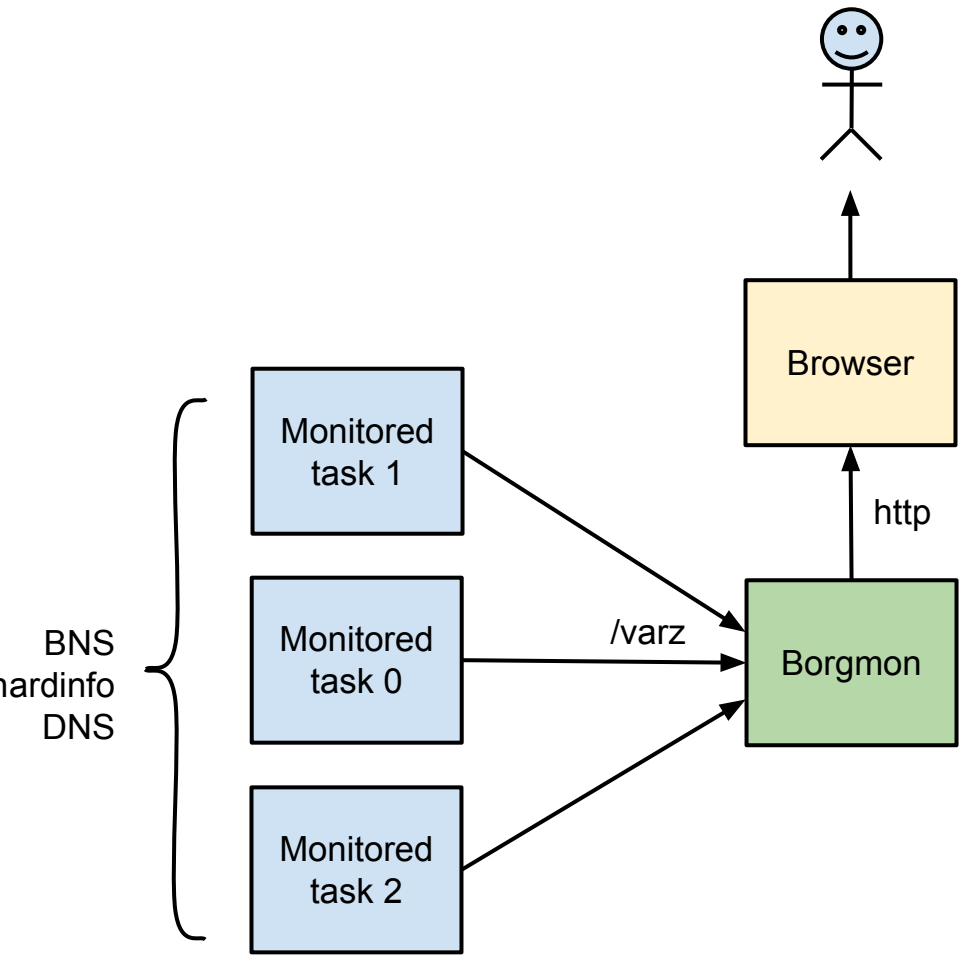
# Borgmon duty cycle



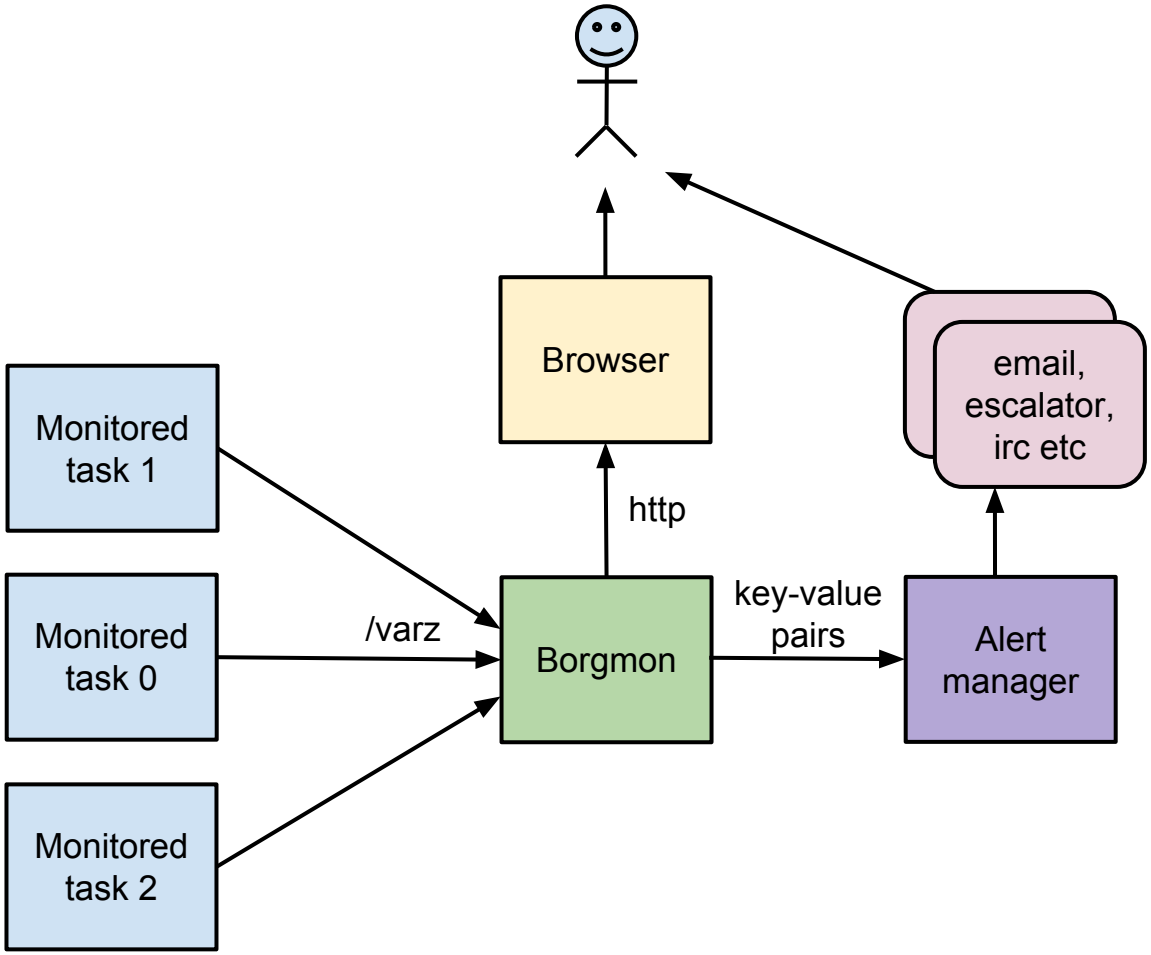
# Borgmon Process Overview



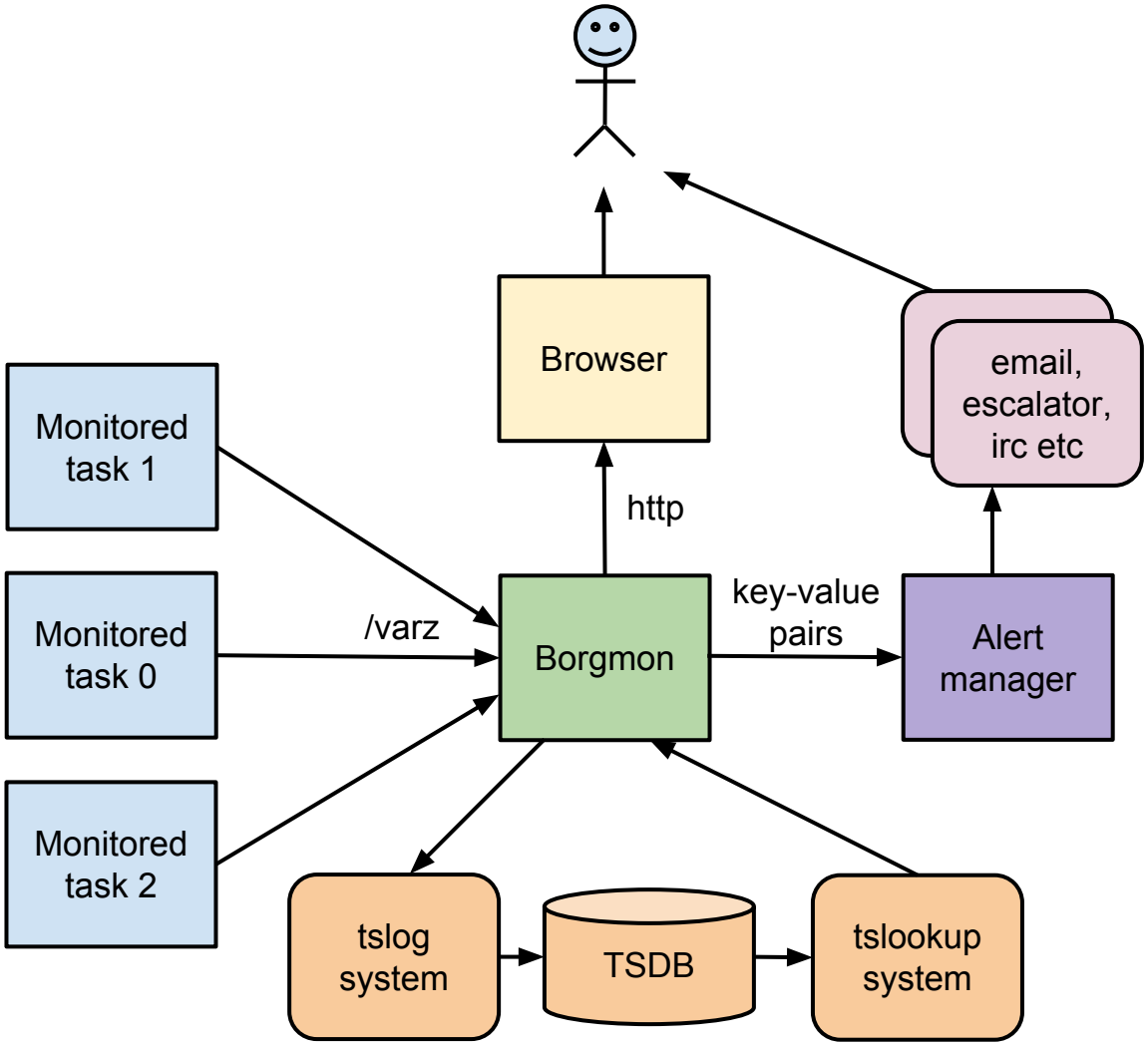
# Add Multiple targets



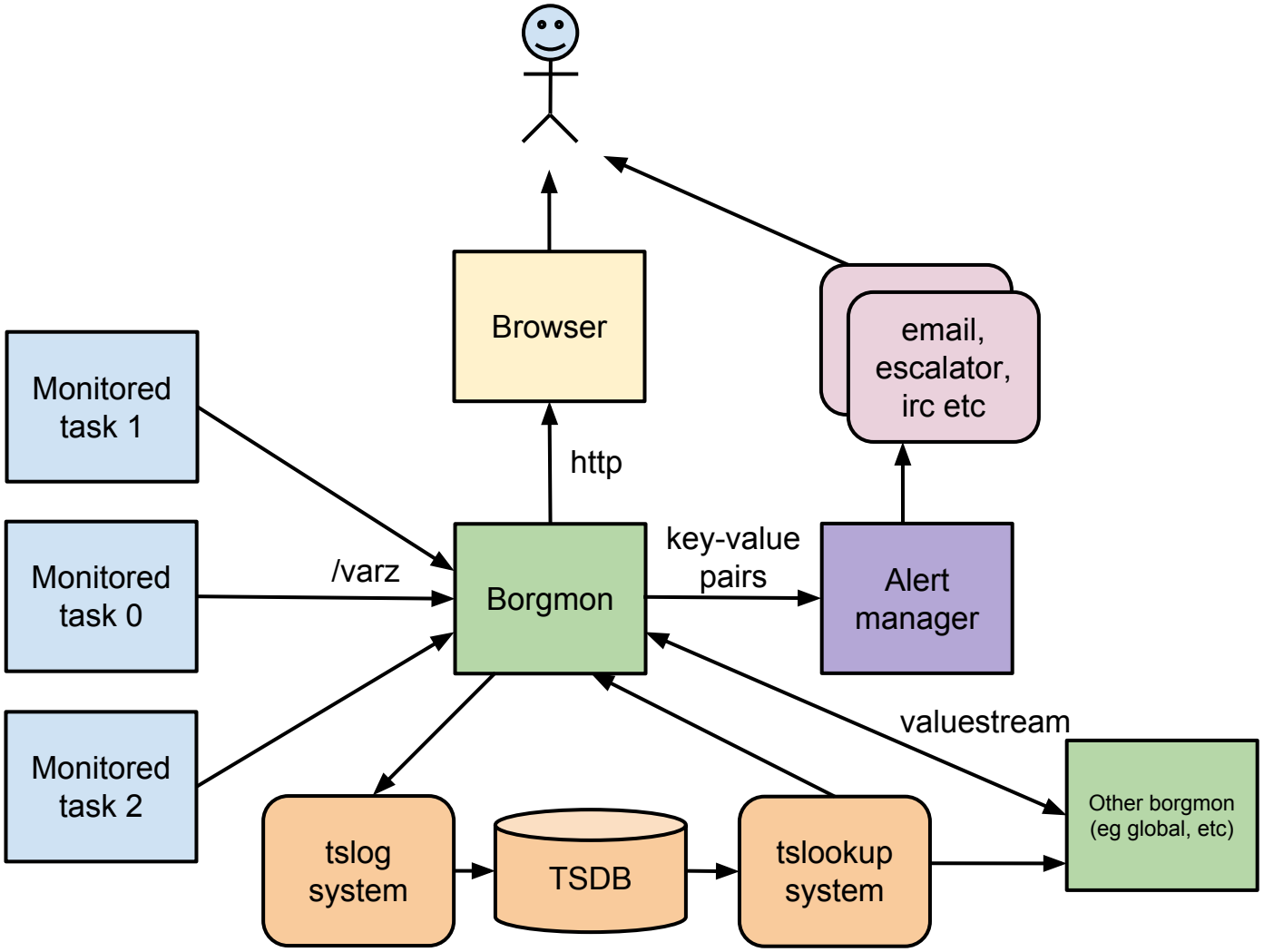
# Add Alert Notifications



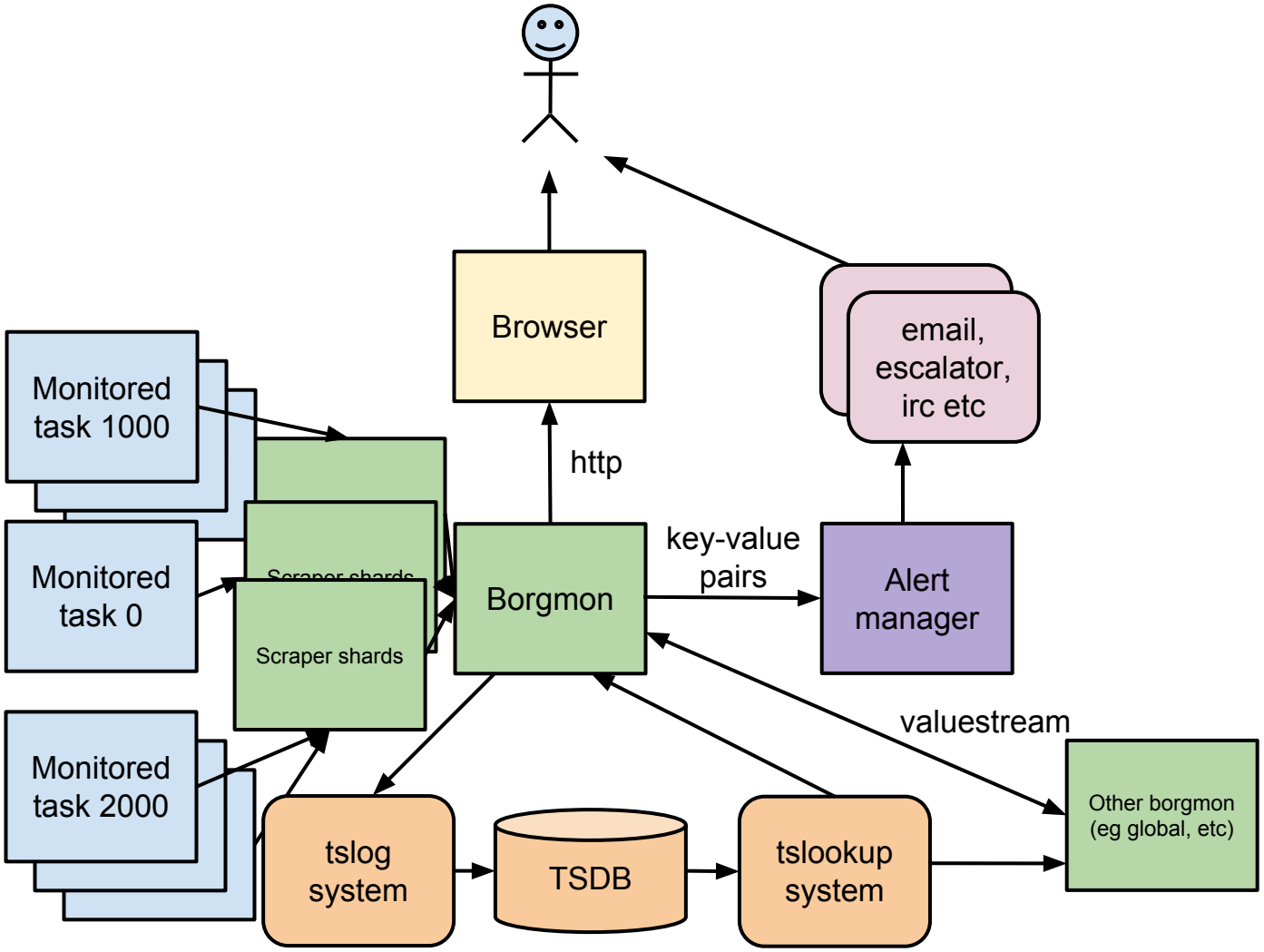
# Add Long-term storage



# Add Global & other monitoring



# Sprinkle some shards on it



# alert design

# SLAs, SLOs, SLIs

- SLA → economic incentives
- SLO → a goal
- SLI → a measurement

I promise to do 5ms @99.9%ile, or else

# 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

# “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

Alerts that page should indicate violations of SLO.

“Traditional” Alerts can be used to help diagnosis by firing but being routed to nowhere, instead displayed on the debugging console.

# Alerts don’t need 1s resolution

- Human response time SLA is typically in minutes.
- You don’t need hi-res input to make quality decisions about SLO.

# instrumentation

# Google’s “framework”

- Google’s origin in HTTP servers
- Common library for RPC and HTTP handling
- Many useful functions are linked in “for free.”
- /varz handler exports the instrumentation

# Simple API

https://golang.org/pkg/expvar/

## Package expvar

```
import "expvar"
```

Overview
Index

### Overview ▾

Package expvar provides a standardized interface to public variables, such as operation counters in servers. It exposes these variables via HTTP at /debug/vars in JSON format.

# 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)
 …
```

# /metric handlers are plain text

```
# HELP requests total requests
# TYPE requests counter
requests 20056
# HELP errors total errors served
# TYPE errors counter
errors{code="Bad Request"} 2027
errors{code="Internal Server Error"}
824
```

# Tips for effective metric setup

- Let Prometheus aggregate for you
- Prefer numbers over strings
- Avoid timestamps
- Initialize variables on program start

# Tips for effective metric setup

Let Prometheus aggregate for you

- Fine:

```
queries_total 1234.0
errors_total  12.0
```

- Questionable:

```
qps 18.0
error_rate 0.2
```

Why?

# Tips for effective metric setup

Prefer numbers over strings

- Helpful:

  **`exceptions{name="GksSqlPermissionsException"}`** `142`
- Less Helpful:

  **`last_exception`**
  `GksSqlPermissionsException`

Why?

# Tips for effective metric setup

Avoid timestamps

- Usually unnecessary:

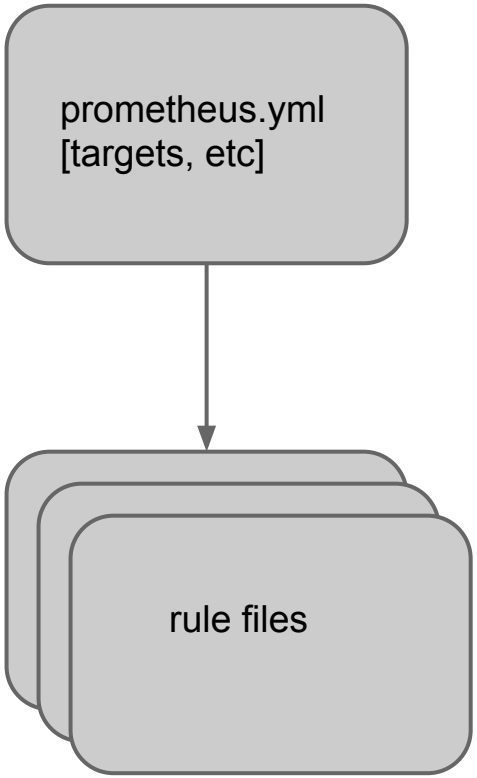
  **`last_sync_time`** `1209770045.0`
- Usually better:

  **`sync_total`** `534.0`

Why?

# collection

# Configuring Prometheus



# Configuring Prometheus

prometheus.yml:

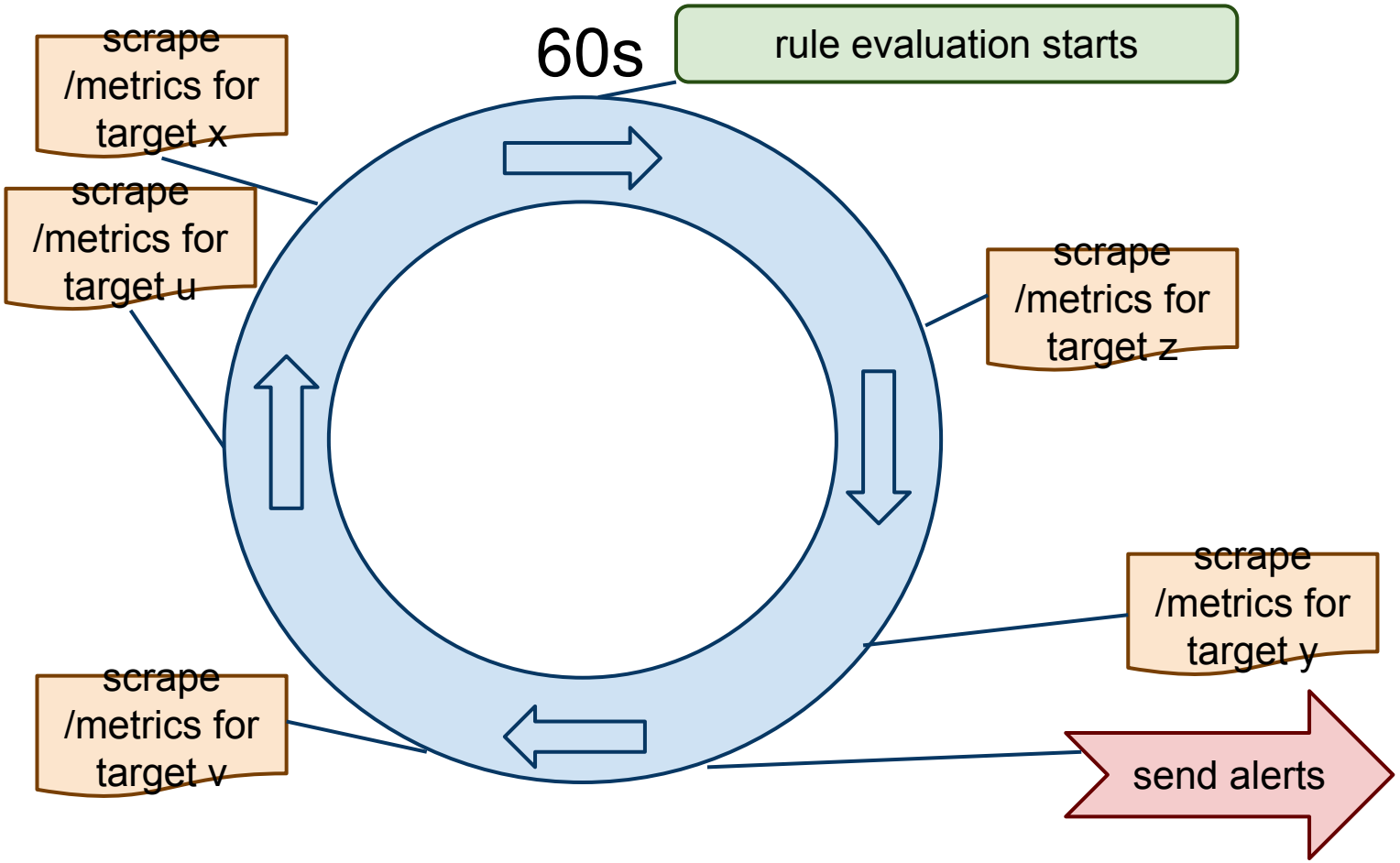
```
global:
  scrape_interval: 1m # duty cycle
  labels:  # Added to all targets
    zone: us-east
rule_files:
  [ - <filepath> ... ]
scrape_configs:
  [ - <scrape_config> ... ]
```

# Finding Targets

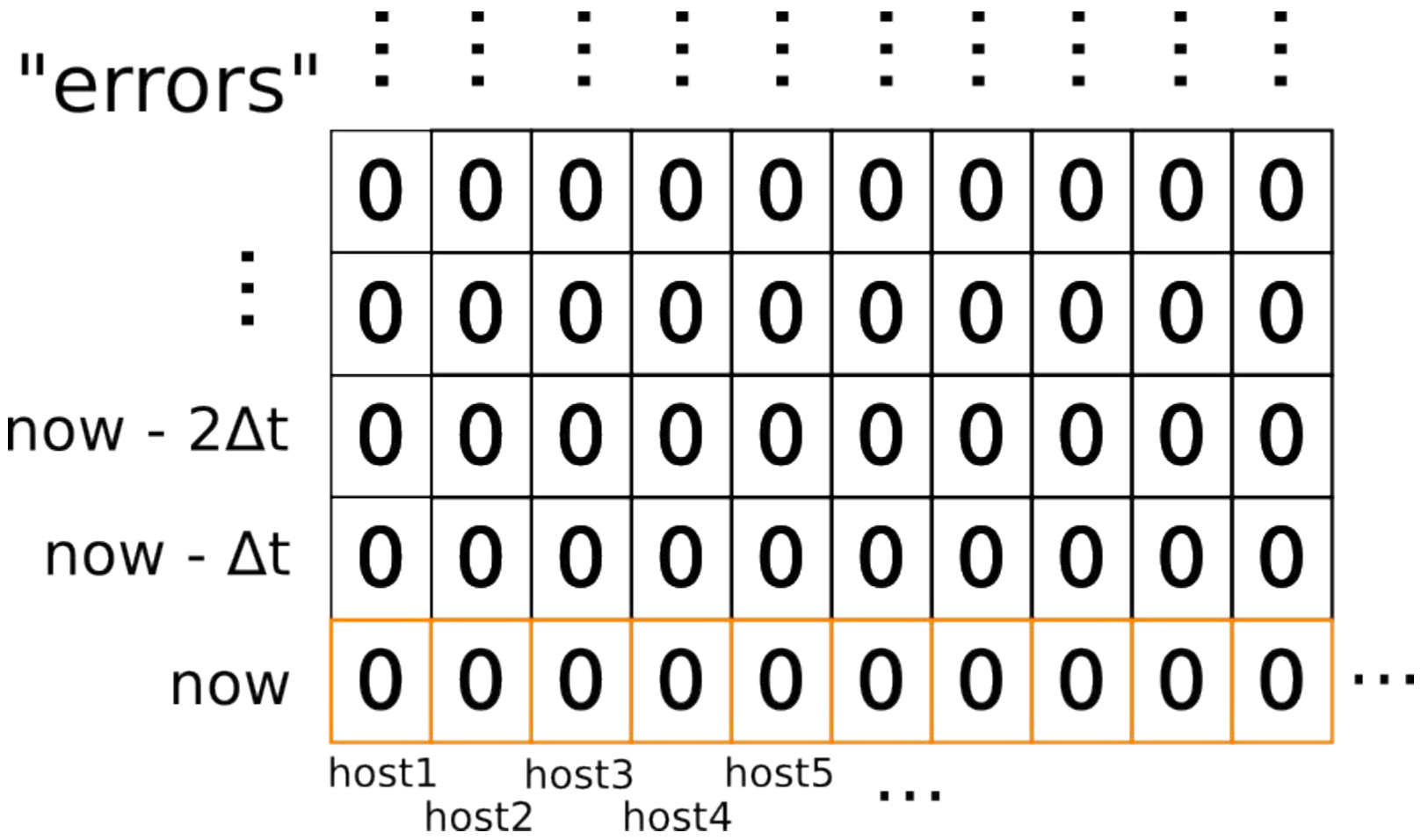
```
scrape_configs:
 - job_name: “smtp”
   target_groups:
   - targets:
     - ‘mail.example.com:3903’

 - 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

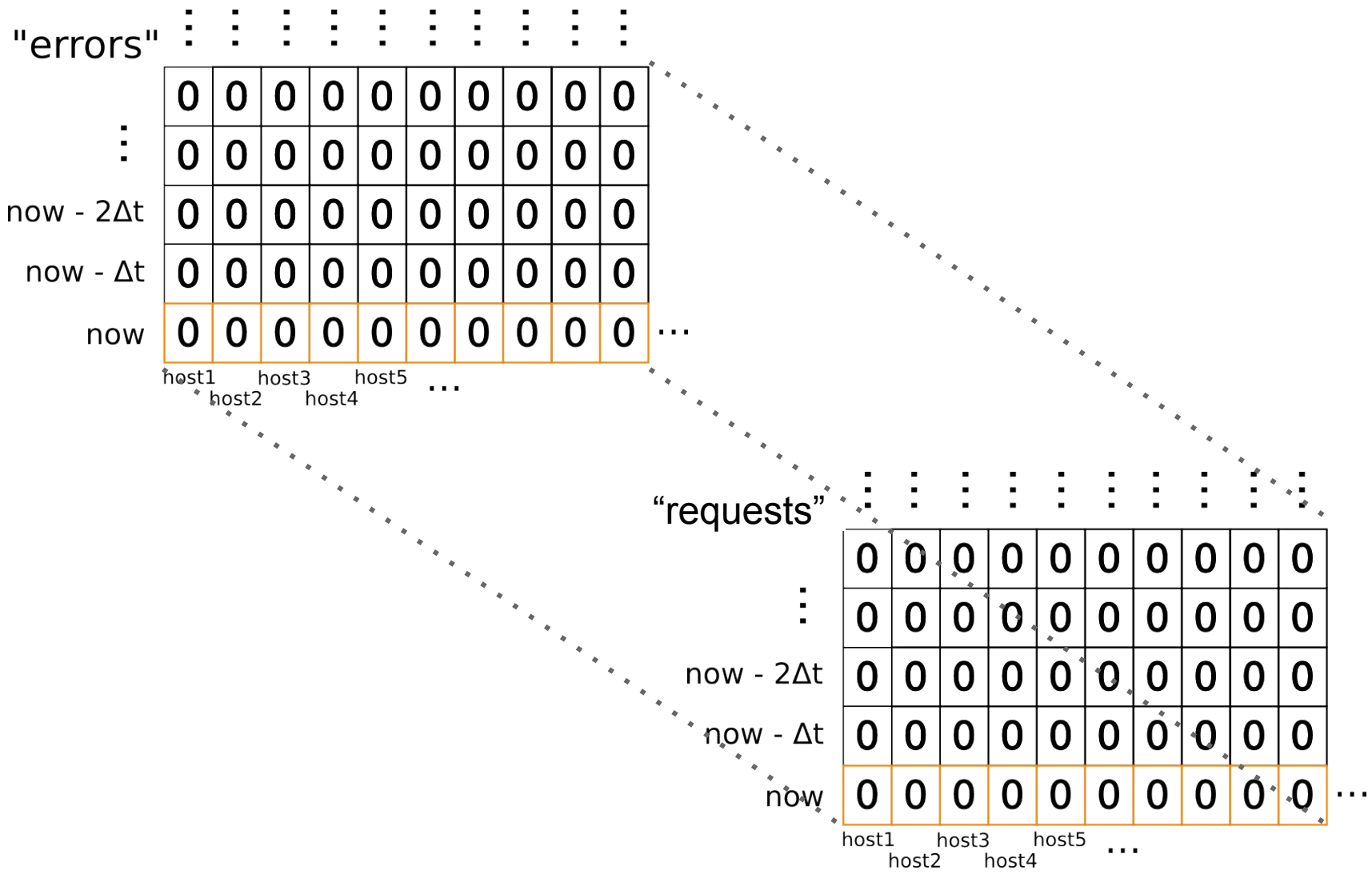
# Prometheus duty cycle



# Structure of timeseries



# Structure of timeseries



# Data Storage Requirements

- A 'service' can consist of:
  - multiple binaries (jobs) running many tasks
  - multiple machines
  - multiple datacenters

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

# Solution: The timeseries arena

- Data is stored in one global database in memory
- 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.

# 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
```

**{var=”errors”,job=”web”}**

or

**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
```

**`{var=”errors”,job=”web”,zone=”us-west”}`**

or

**`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
```

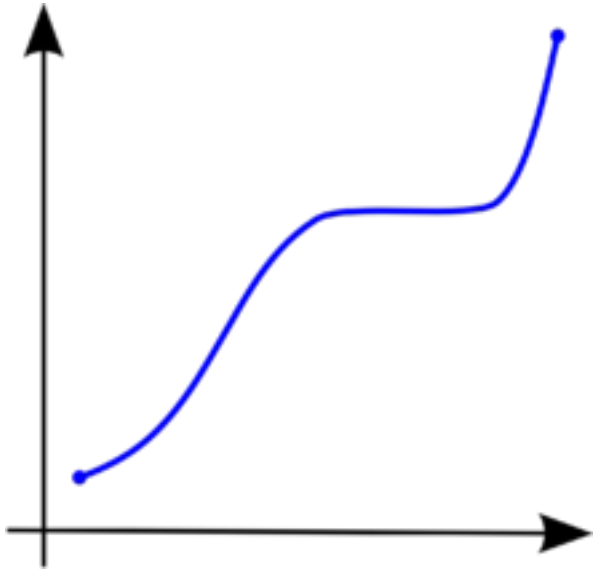
**{var=”errors”,job=”web”,zone=”us-east”, instance=”server01:8000”,code=”500”}**

# Timeseries Have Types

Counter: monotonically nondecreasing

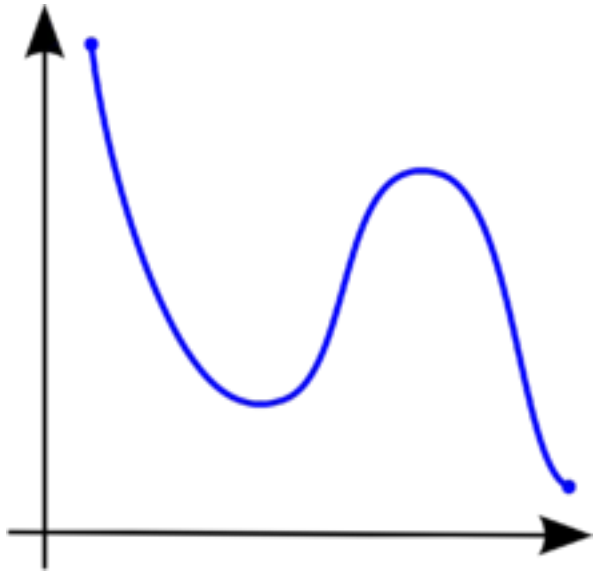
"preserves the order" i.e. UP

"nondecreasing" can be flat

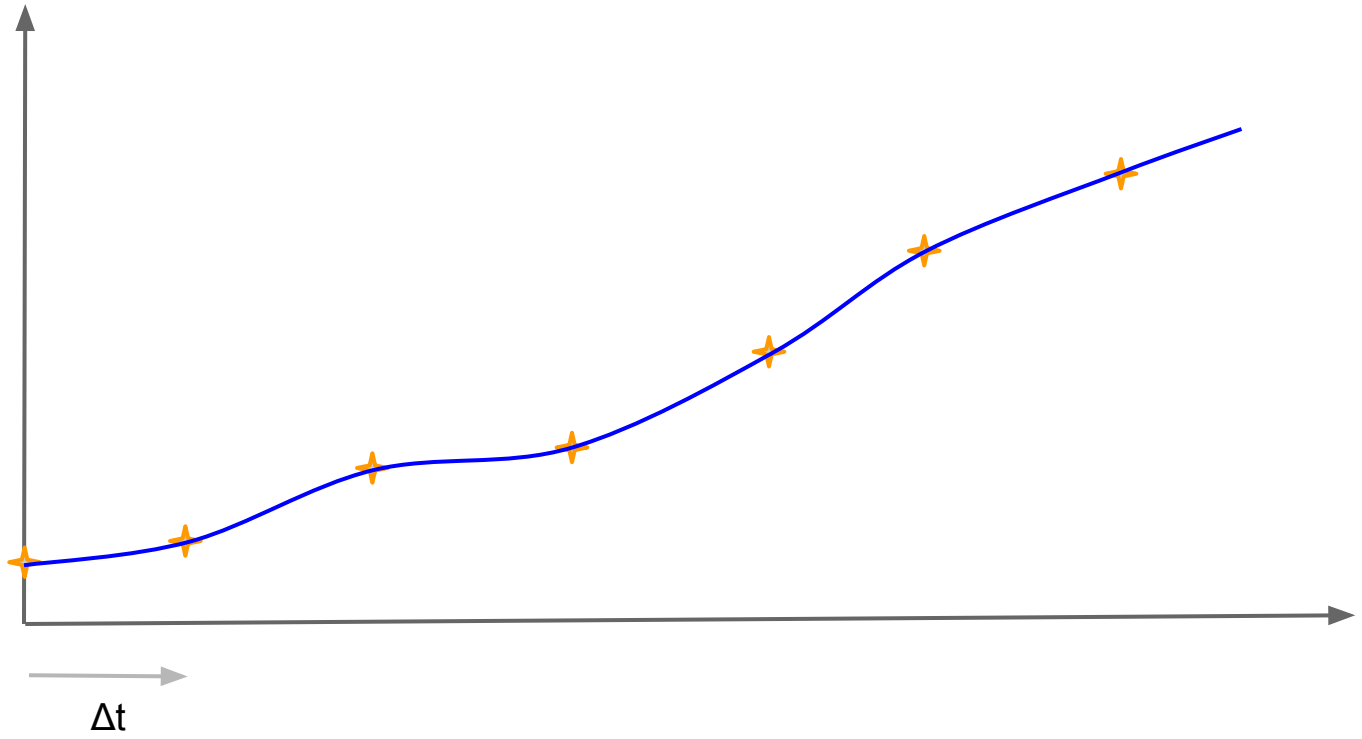


# Timeseries Have Types

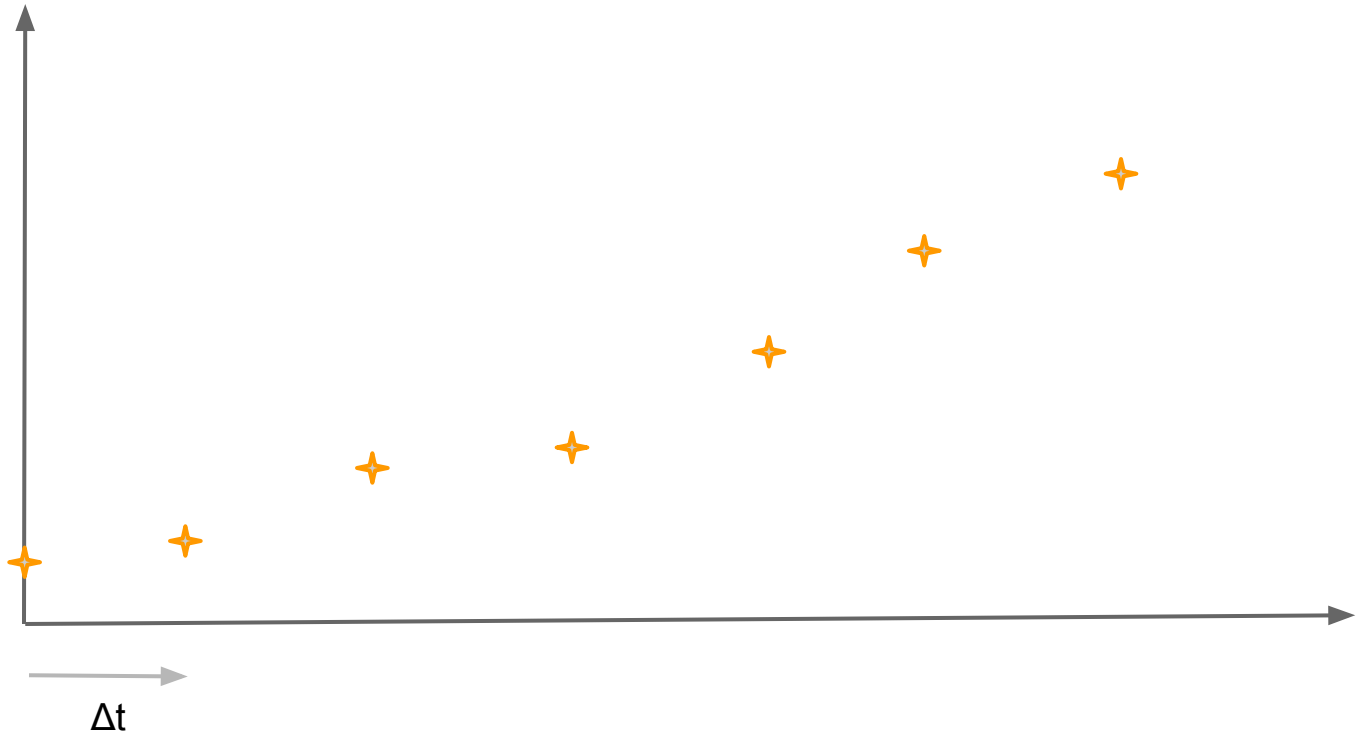
Gauge: everything else... not monotonic



# Counters FTW

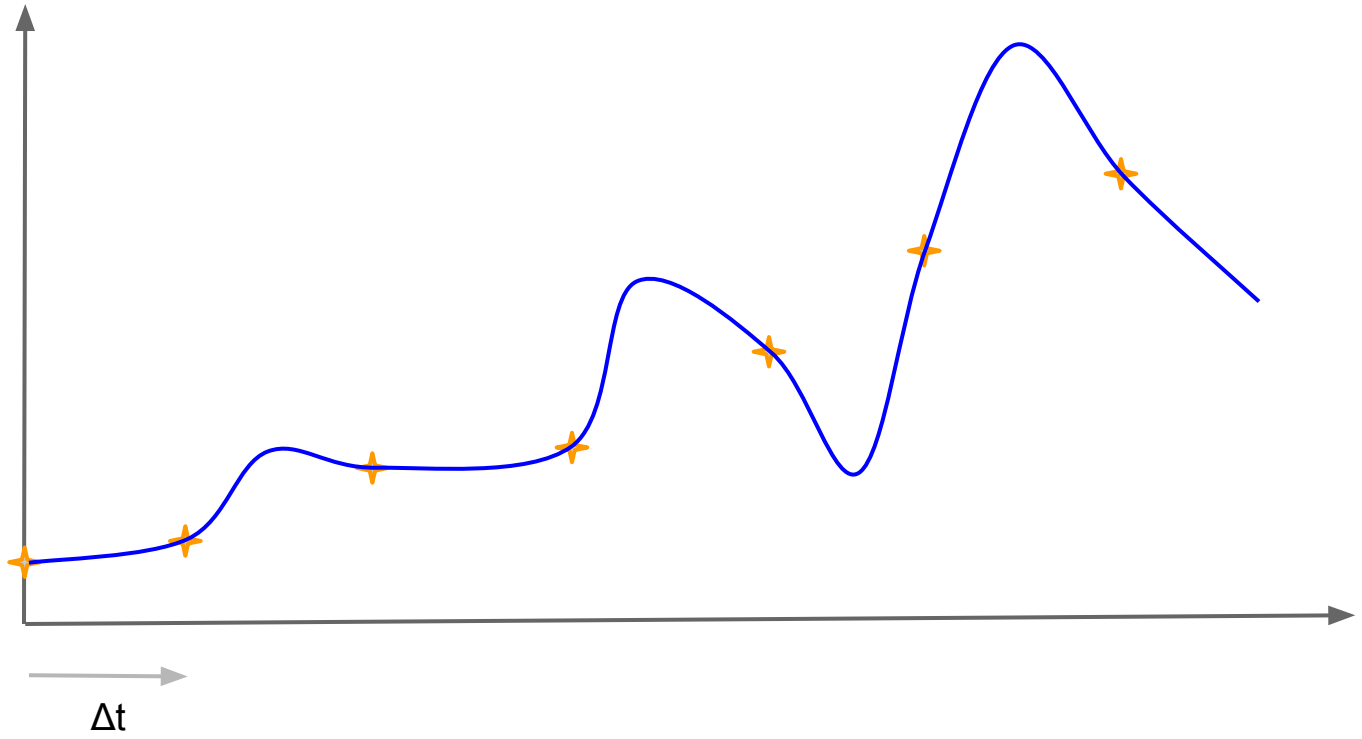


# Counters FTW

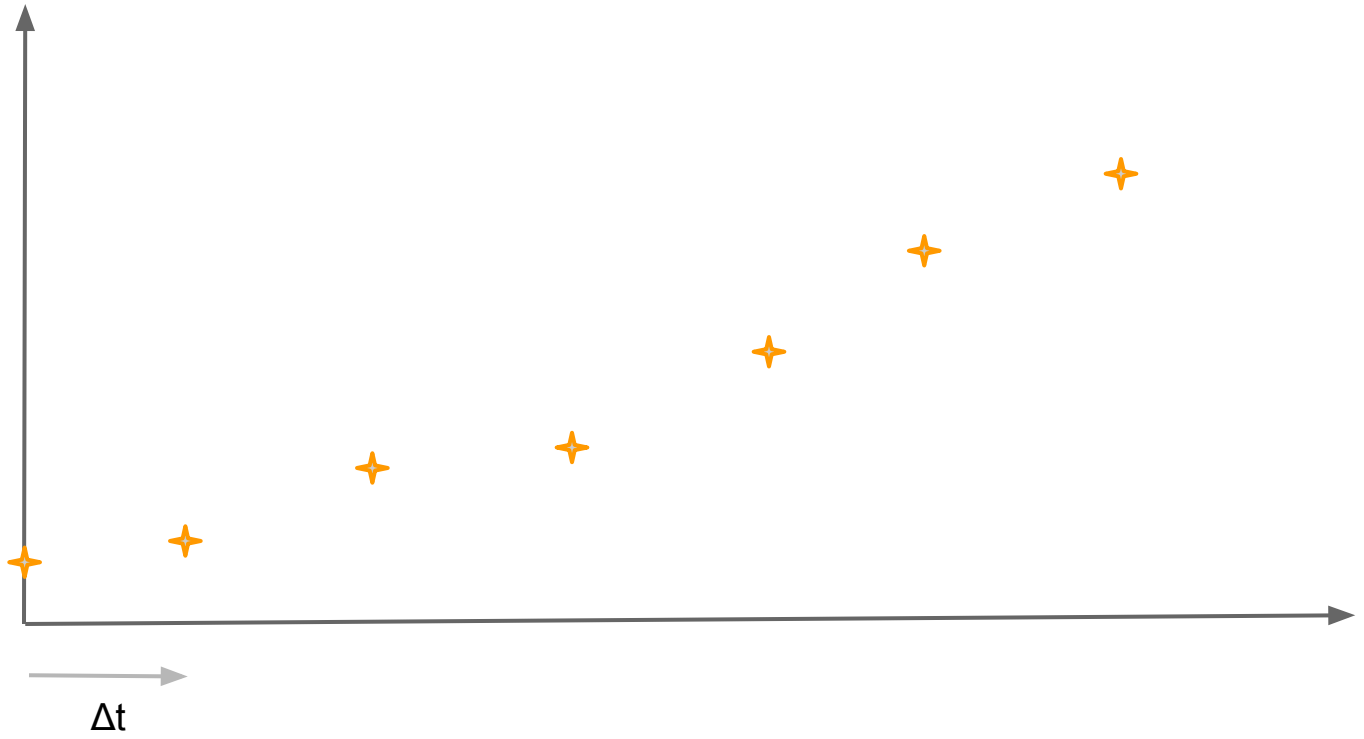



no loss of meaning after sampling

# Gauges FTL



# Gauges FTL




lose spike events shorter than sampling interval

# rule evaluation

# recording rules

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

```
requests{instance="localhost:8001",job="web"} 21235
requests{instance="localhost:8005",job="web"} 21211
→
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

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

“operation”

# recording rules

```
task:requests:sum =
  sum(requests{job=”web”})
```

“operation”

# recording rules

```
task:requests:delta10s =
  delta(requests{job=”web”}[10s])
```

“operation”

# aggregation based on topology

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

dc:requests:rate10s =
  sum by (job,zone)(
    task:requests:rate10s)

global:requests:rate10s =
  sum by (job)(job:requests:rate10s)
```

# aggregation based on topology

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

dc:requests:rate10s =
  sum by (job,zone)(
    task:requests:rate10s)

global:requests:rate10s =
  sum by (job)(job:requests:rate10s)
```

# relations based on schema

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

# relations based on schema

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

# relations based on schema

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

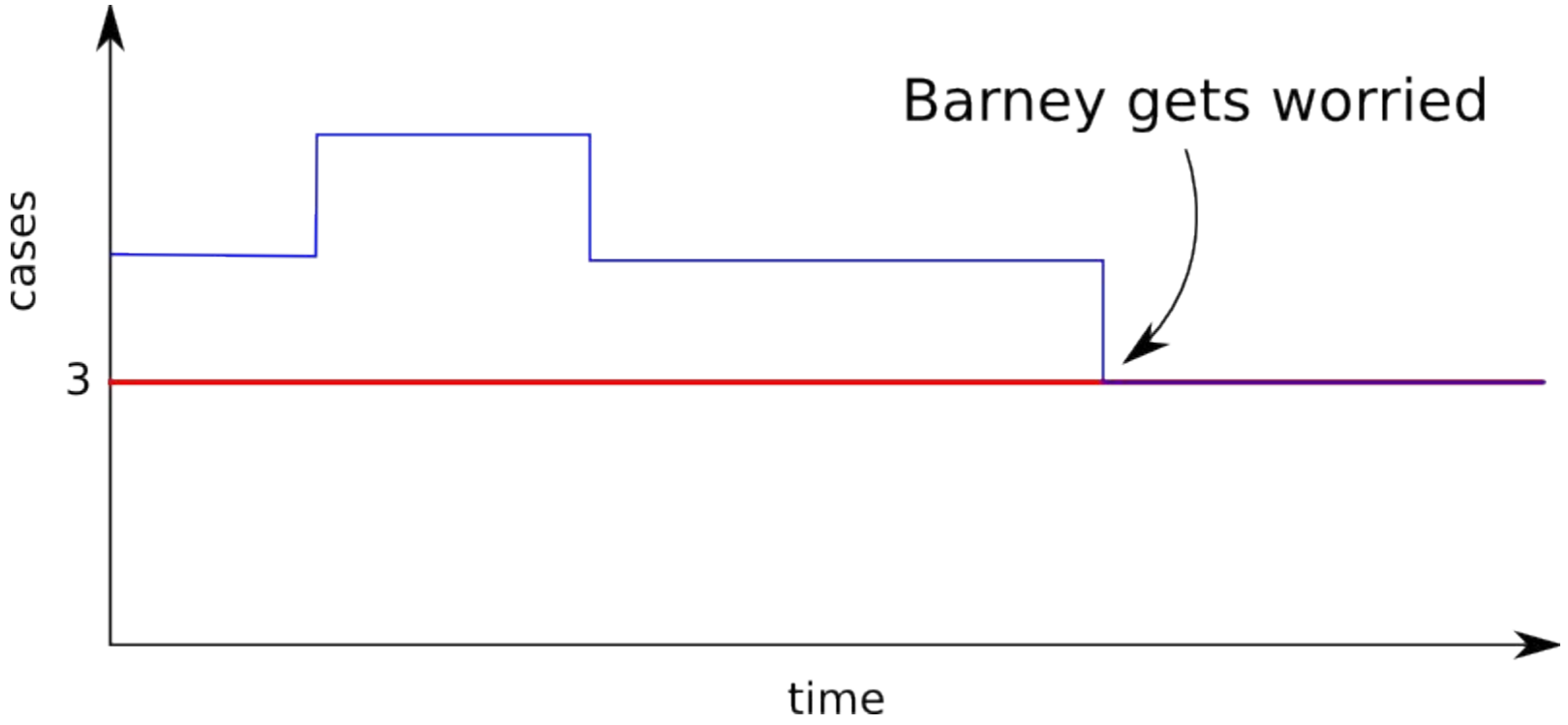
# timeseries based alerting

(the actual talk)

# 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
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:

**How long before the disk is full?**

and

**How long will it take for a human to fix an (almost) full disk?**

# CALCULUS

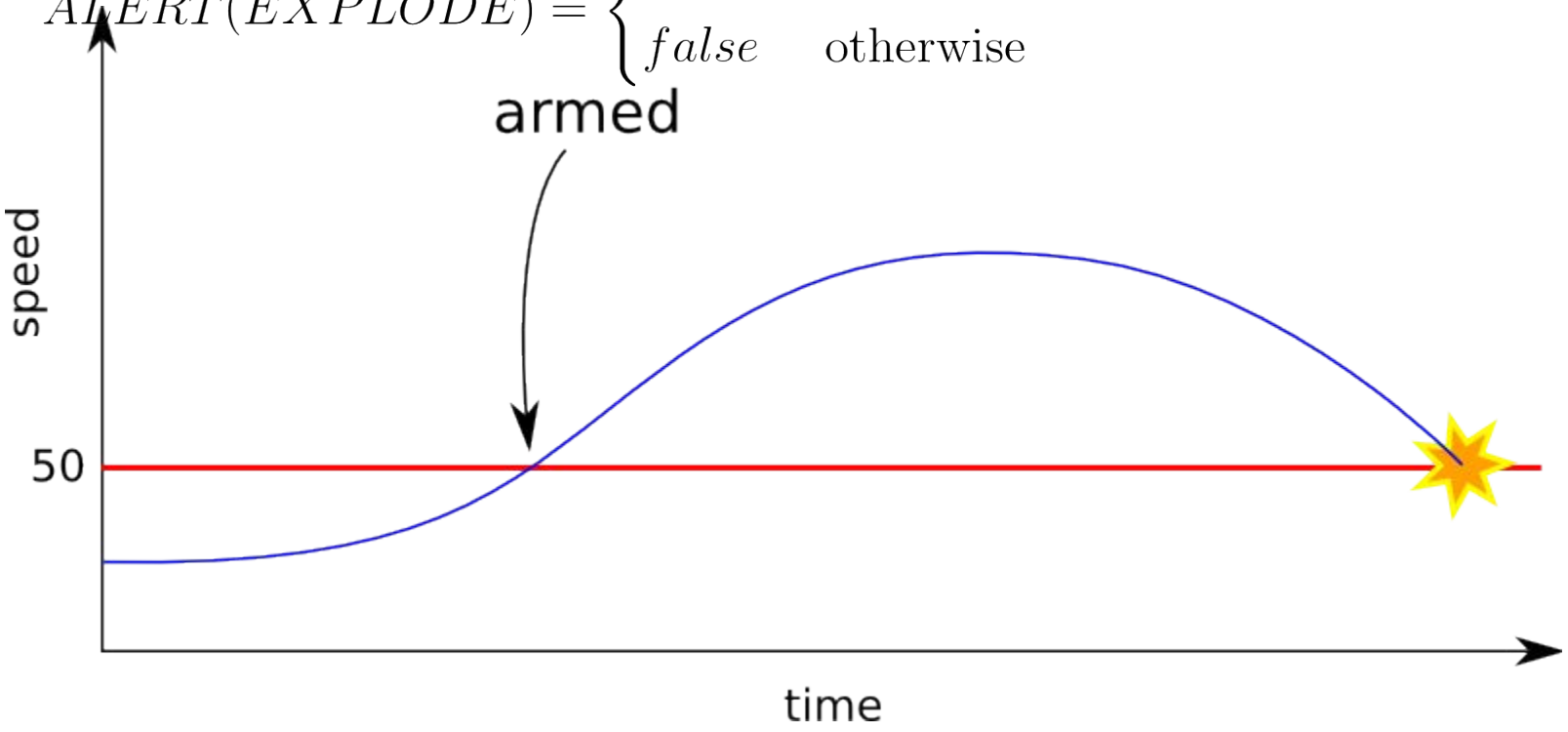
# Alerting on rates of change

# Dennis Hopper's Alert

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

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

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

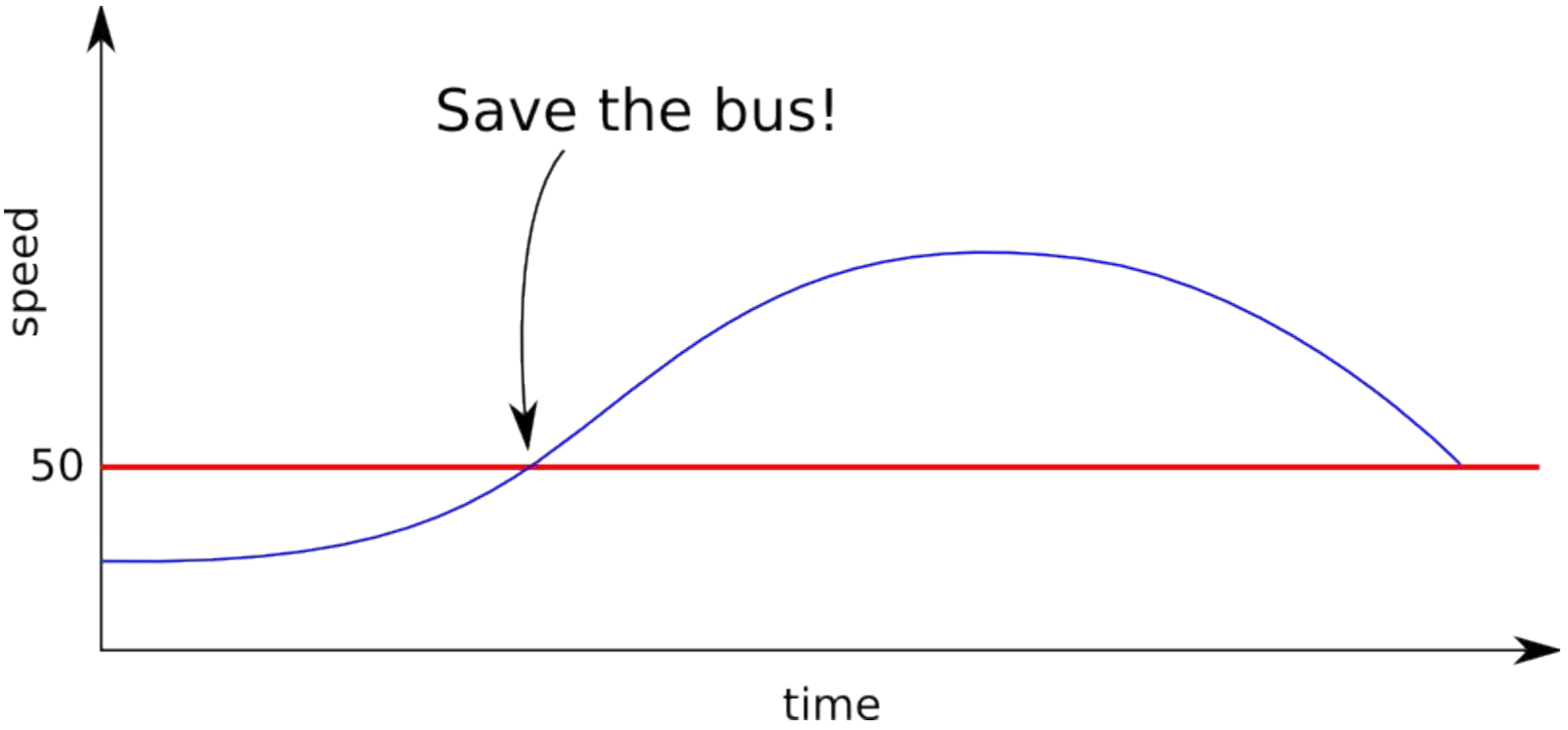


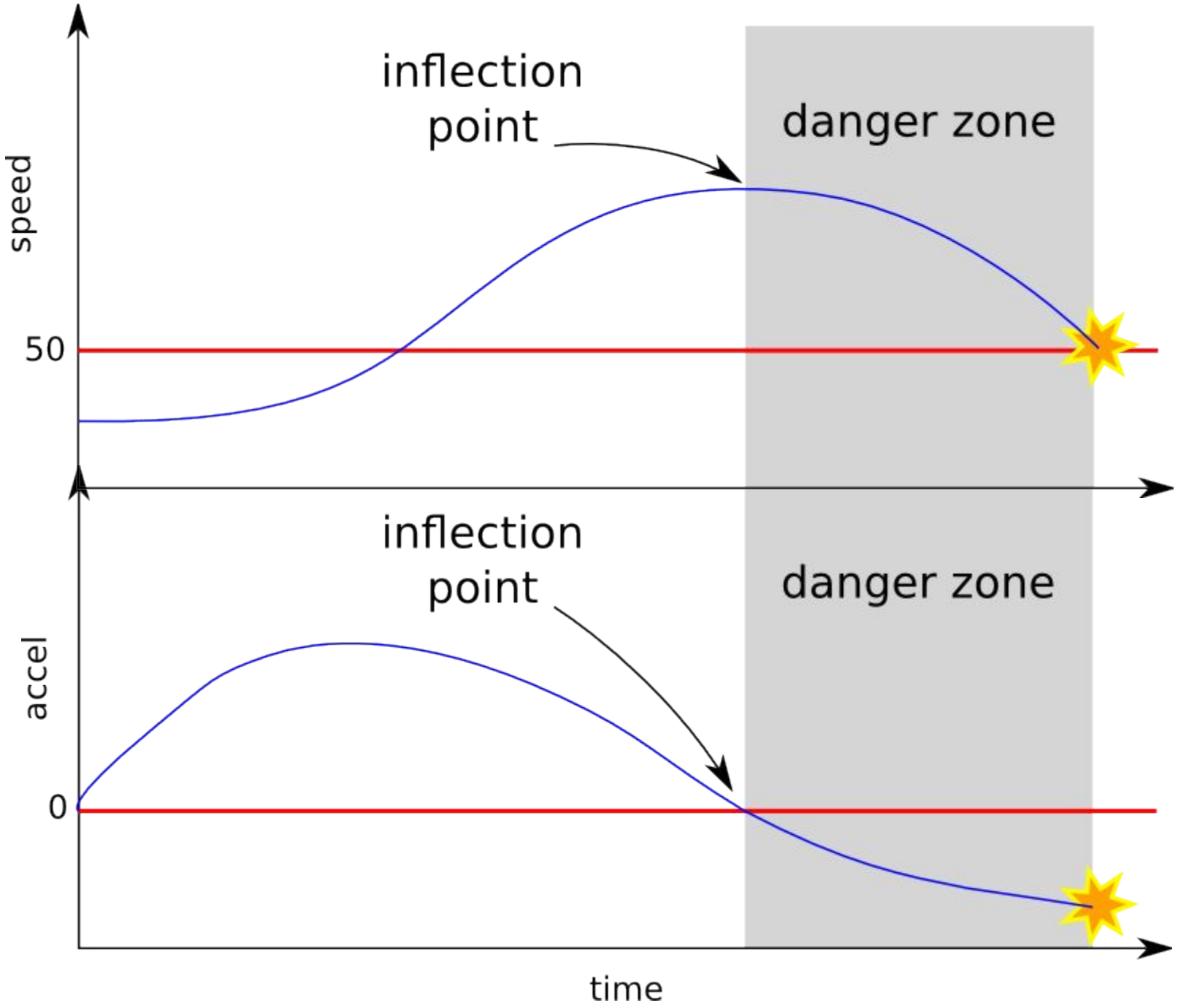
# Dennis Hopper's Alert

```
ALERT BombArmed
IF speed_mph >= 50
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}
SUMMARY “I’d want to know what bus it was.”
```

# Demo

http://github.com/jaqx0r/blts

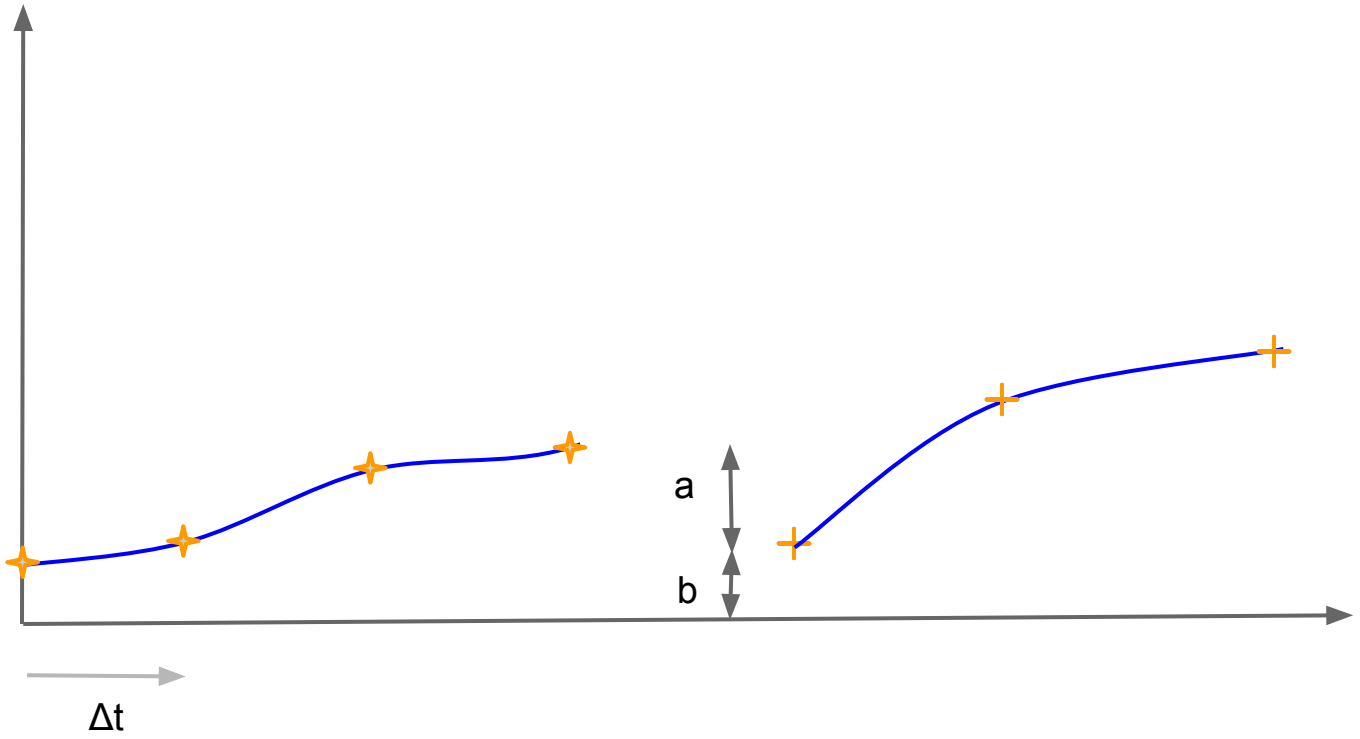
# Question Time

(for realsies this time)

# Backup slides

# counter resets

# Counters



# #monitoringsucks

# Goals of Production Monitoring

- 24/7 Awareness of service health
- Problem -> Owner ASAP
- Record Keeping
- Live "Dashboards"

# Why do we monitor?

The primary function of borgmon rules is to generate alerts.

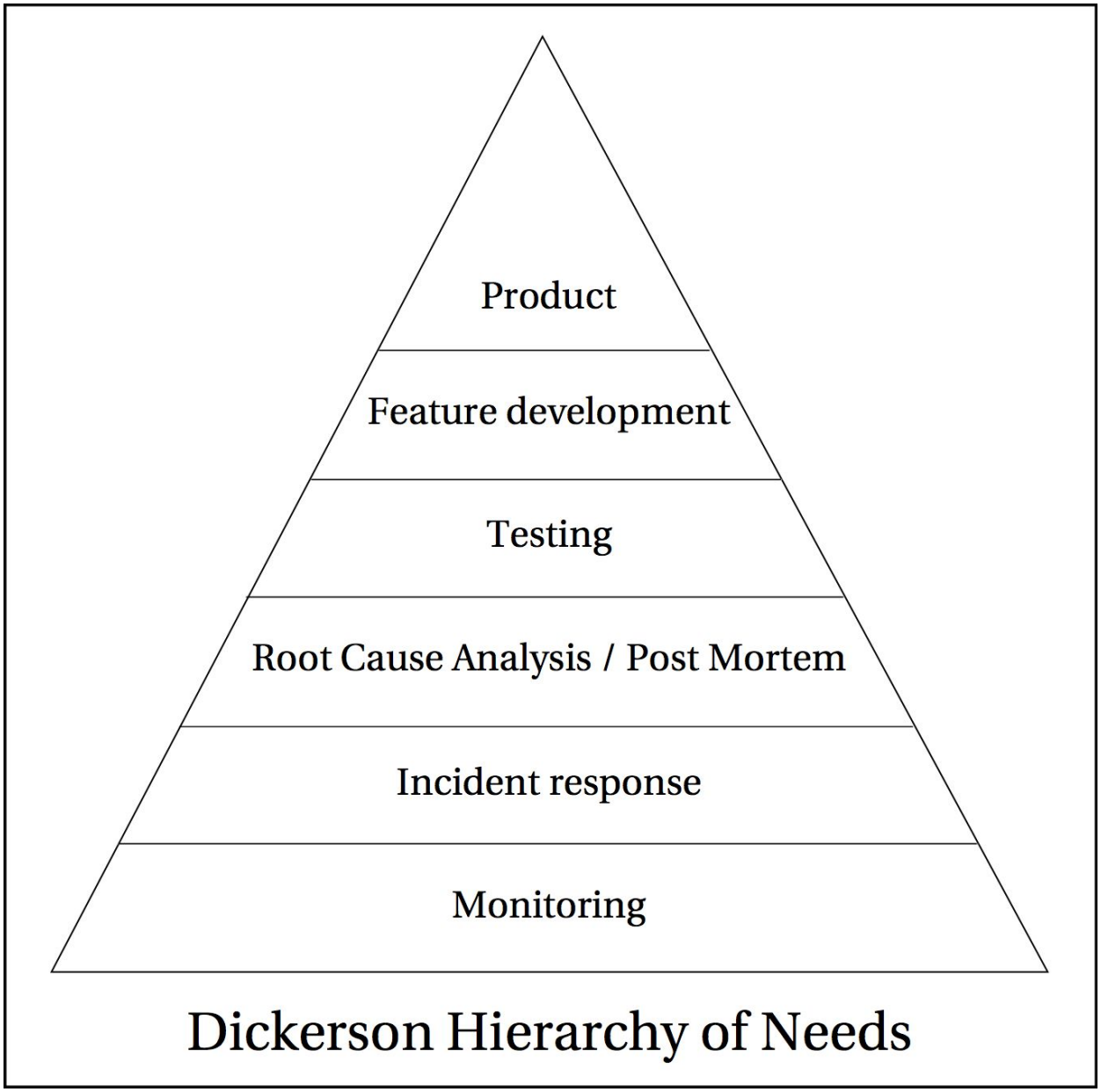
The secondary function of borgmon is to provide debugging information, typically in the form of consoles.

# Monitoring systems

automate boring parts of experimental method

- measuring,
- recording,
- alerting,
- visualisation

so you have more time to do fun things... and debug the occasional emergency.




Dickerson Hierarchy of Needs

# Goals of Production Monitoring

- Automate experimental method, to give
- 24x7 awareness of service health, via
- Generating alerts,
- Record keeping
- Exploratory tools

# Goals of Production Monitoring

- Automate experimental method, to give
- 24x7 awareness of service health, via
- Generating alerts,
- Record keeping
- Exploratory tools
- **… while keeping the maintenance costs low as the system grows**

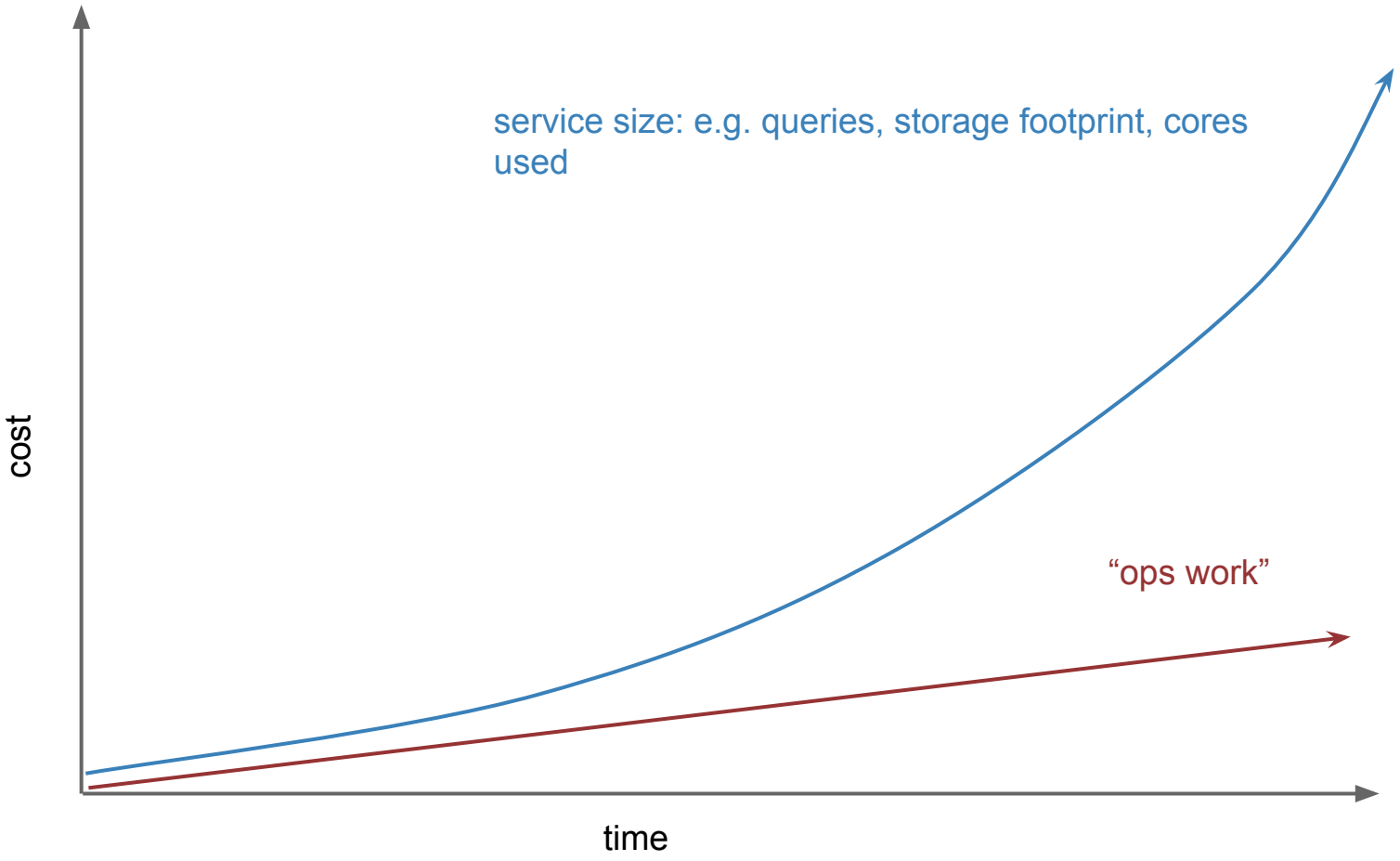
# Why *does* #monitoringsuck?

TL;DR:

when the cost of maintenance is too high to improve the quality of alerts

# Why *does* X ∀ X ∈ {Ops} suck?

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




jaq’s hypothesis

# Automate yourself out of a job

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