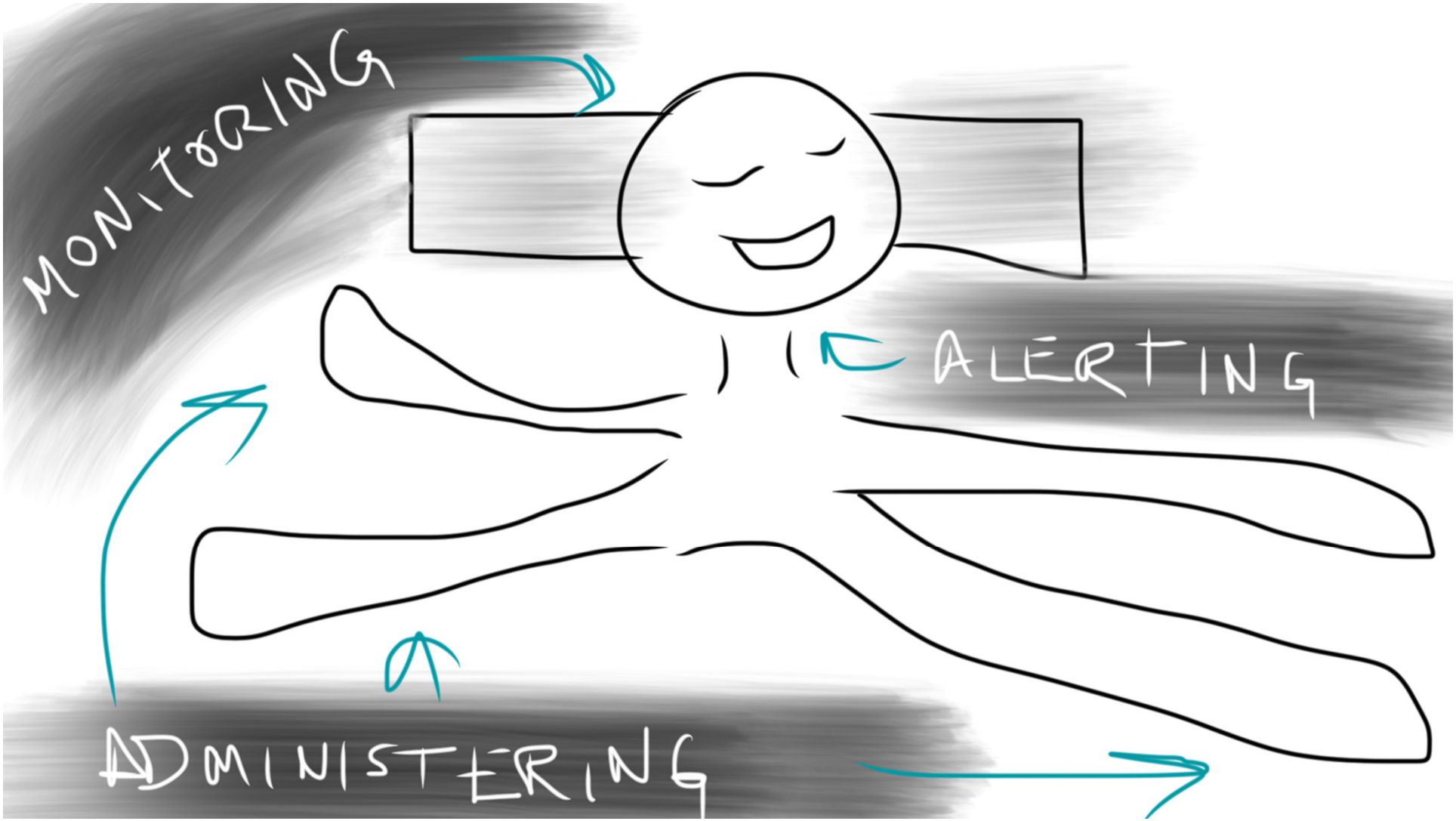
SATELLITE

MESOSCON 2015


@sabraham
20


/twosigma
/satellite

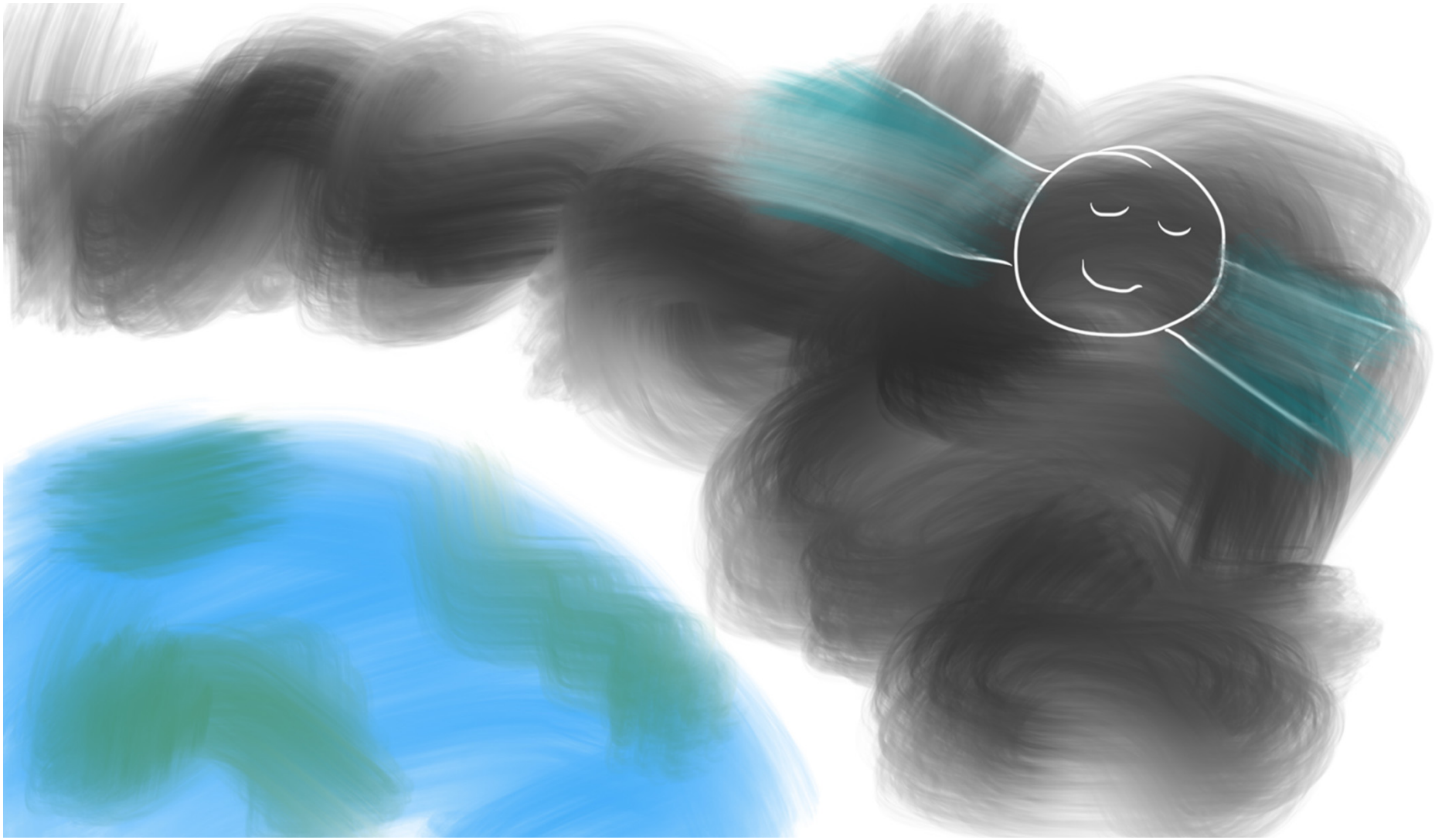

MONITORING
ALERTING
ADMINISTERING

SUMMARY

Example of what Satellite can do: When swap falls below 1 day, turn the host off; if 20% of the cluster is turned off, send a pagerduty alert.

Satellite is an application Two Sigma wrote to monitor, alert, and auto-administer our Mesos clusters.

- Monitor: provide a global view of the cluster
- Alert: communicate status changes to the outside world
- Administer: perform actions on status changes that affect the cluster



SUMMARY

Mesos exposes limited information through its HTTP REST API. With Satellite, you can expose arbitrary host metrics either at the host level or aggregated.

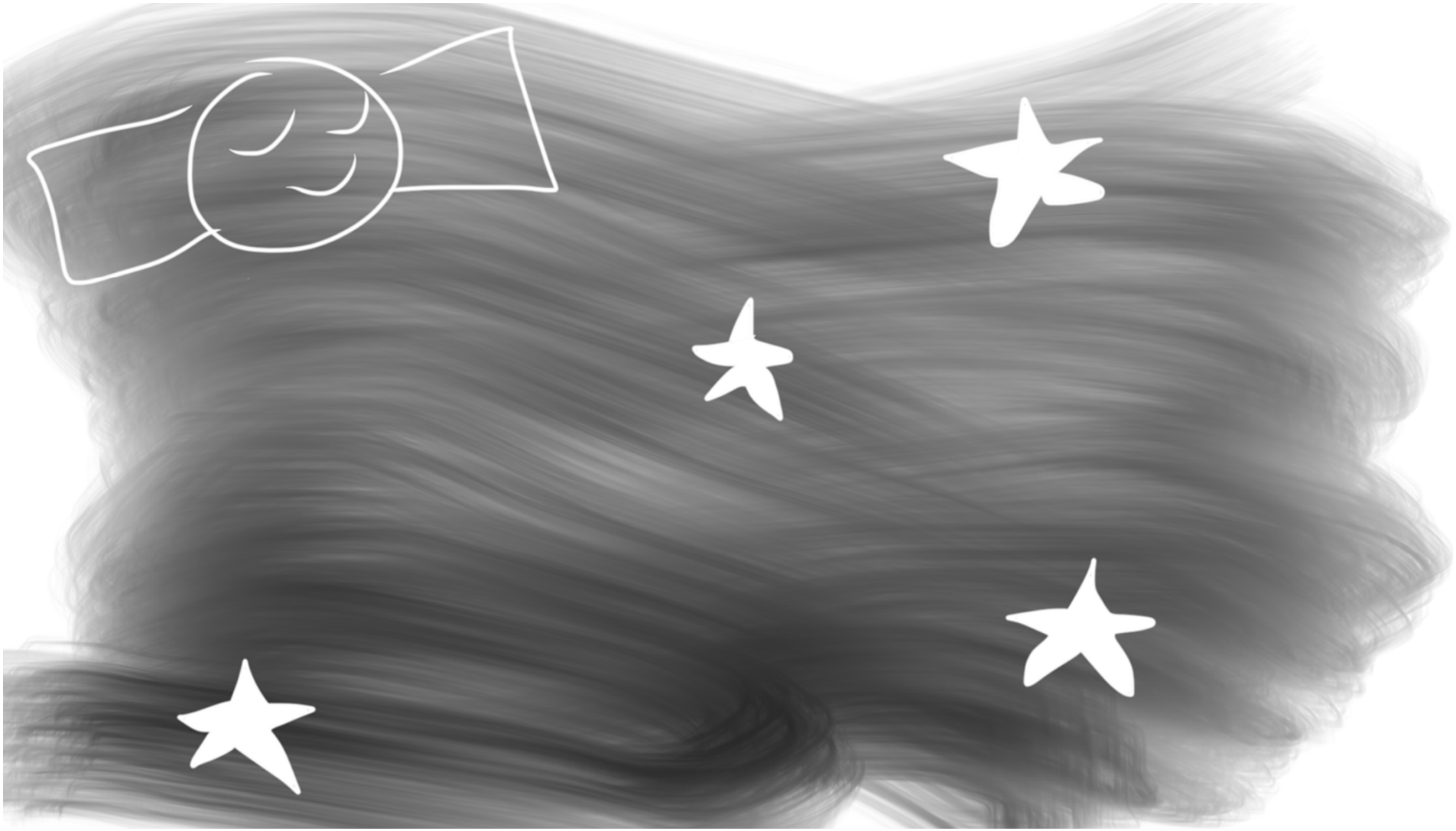
As an example, I'd like to know in real-time

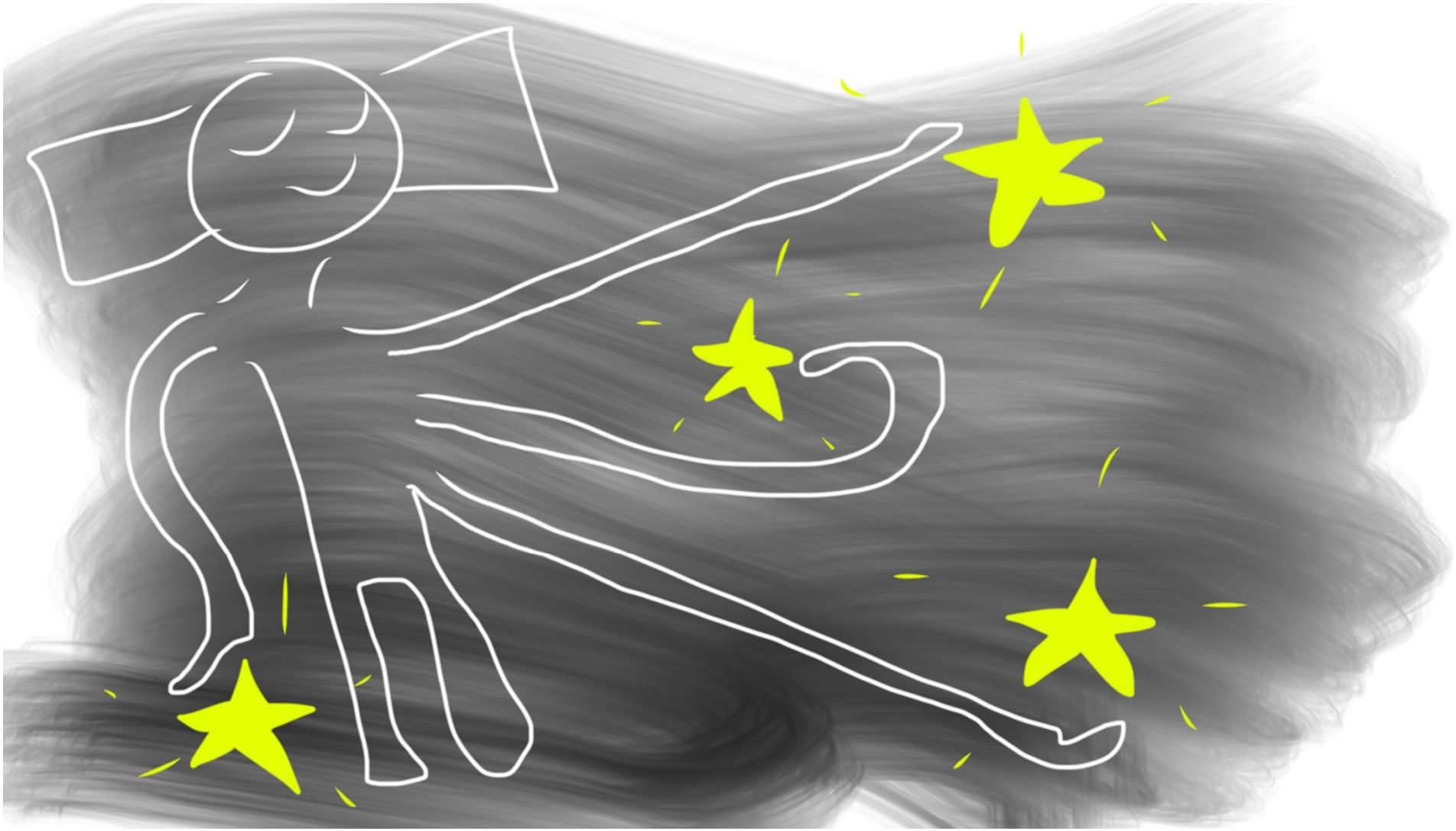
- What percent of the cluster had high swap utilization
- What the median `max_allowed_age` is on the cluster
- How many slaves have a `max_allowed_age` less than 1 day

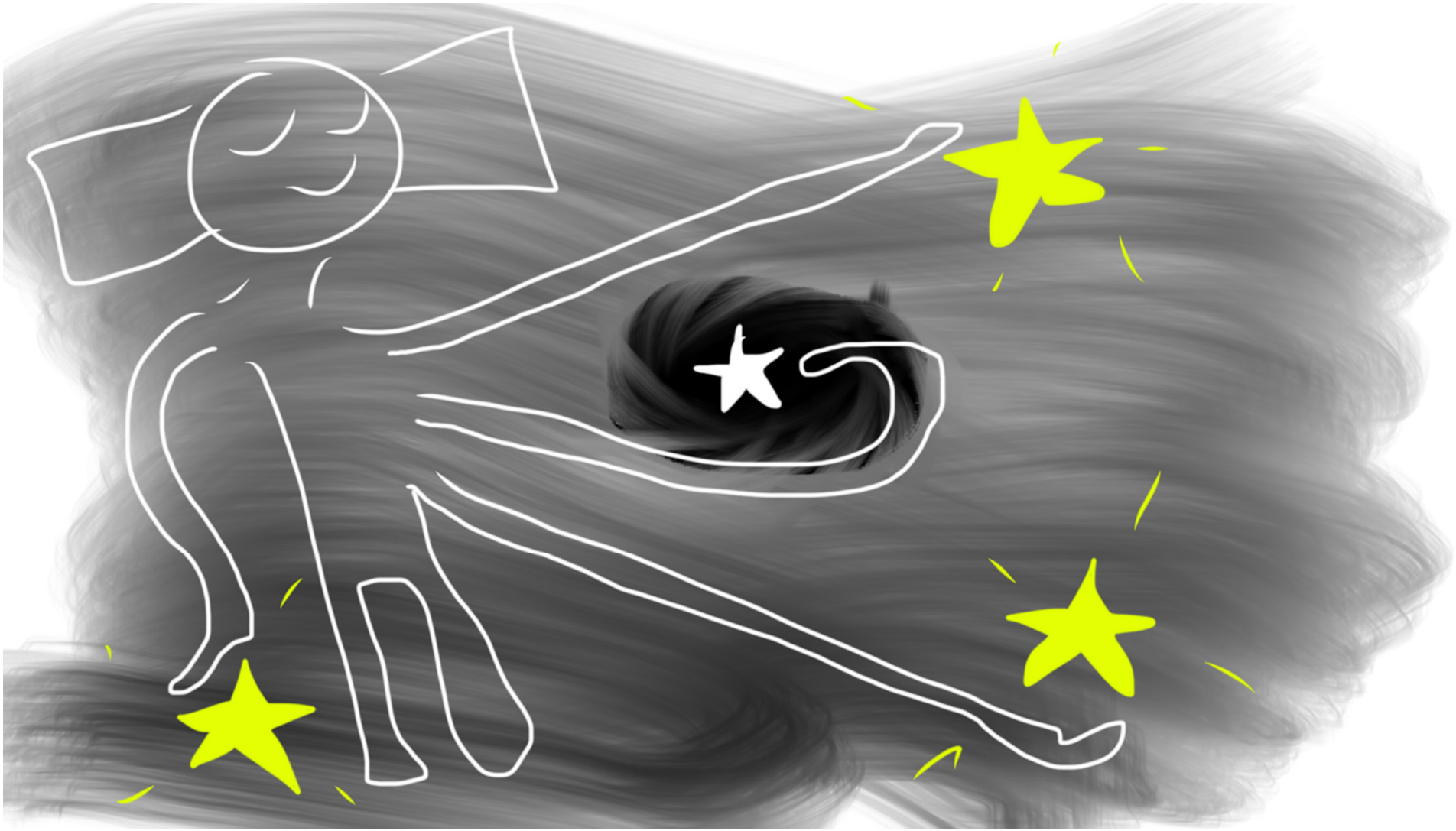
SUMMARY

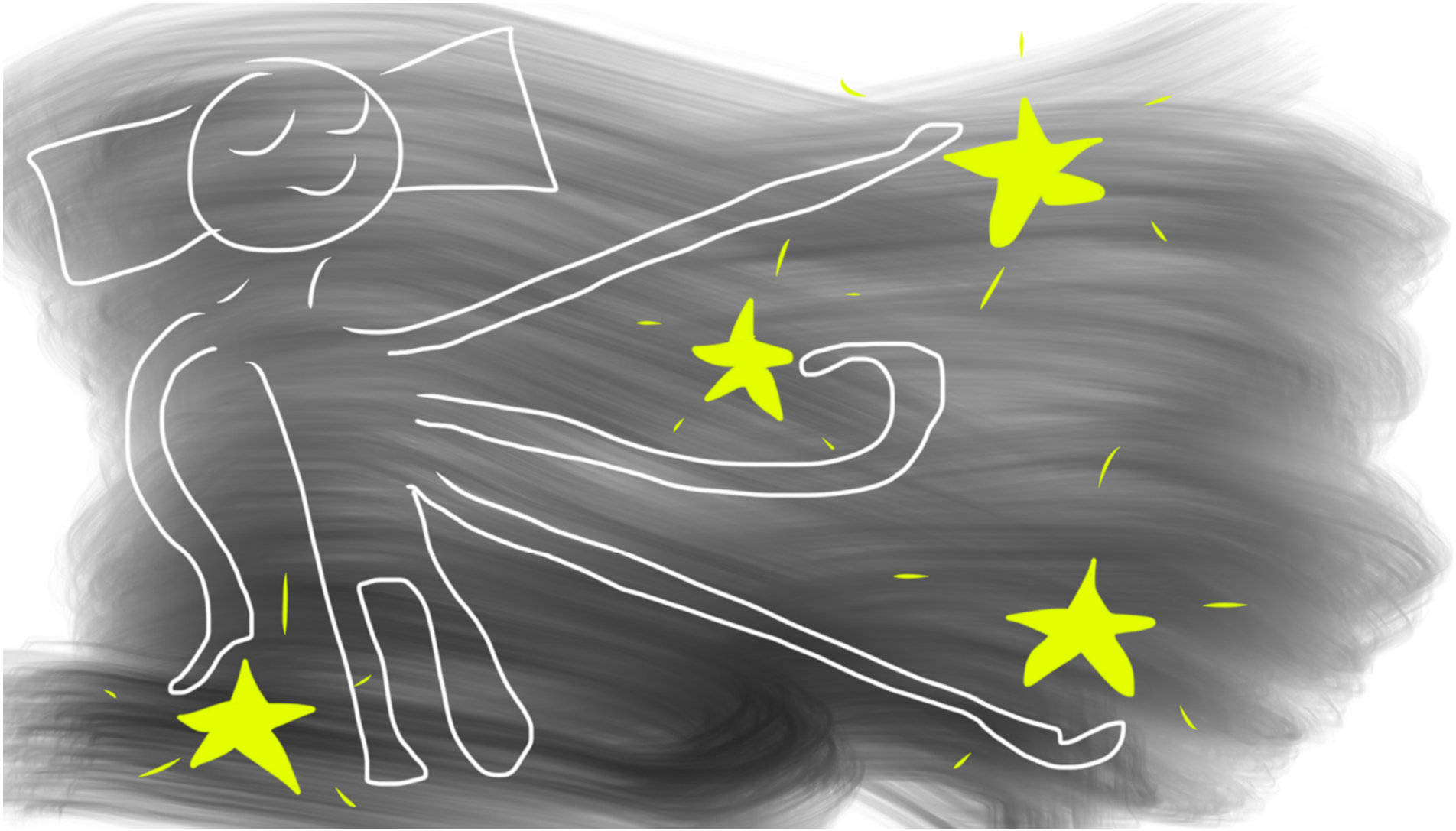
Alerting means communicating this aggregate view you have derived.

Ex: In the swap case, say we want to receive an email whenever a host makes a state transition from < 90% swap utilization to >= 90% swap utilization. And we want to get a pagerduty alert when 50% of the cluster is >= 90%.







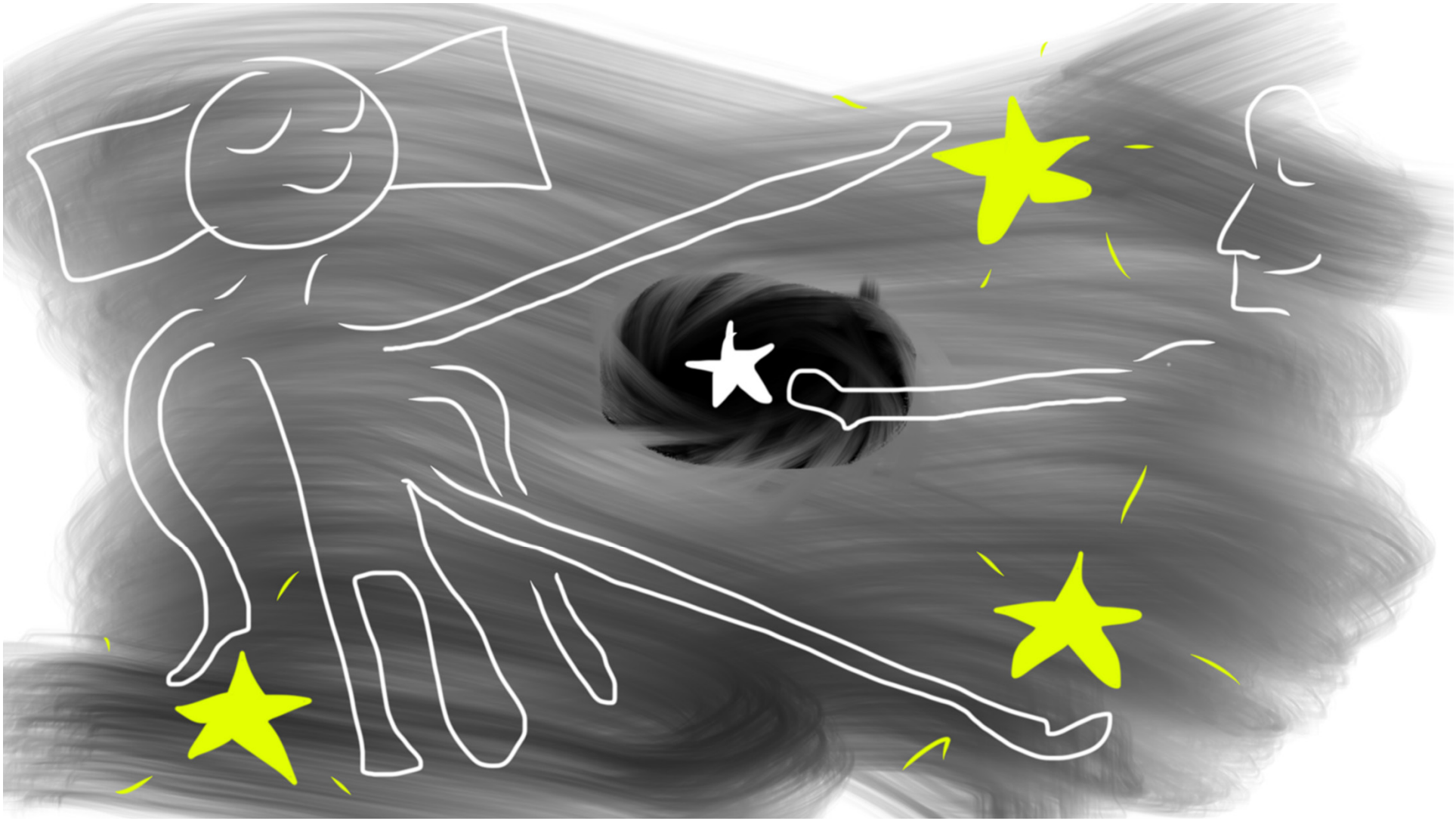


SUMMARY

Auto-administration is the ability to programmatically control when a host will receive new tasks.

Satellite offers two special primitives, **off-host** and **on-host,** that allow you to stop sending new tasks to a given host and re-commence sending tasks to a host, respectively.

Ex: Suppose when a host has 90% of swap used, you want to turn it off, and when pressure relieves, you want it to turn back on, automatically. This something you can do easily within satellite.

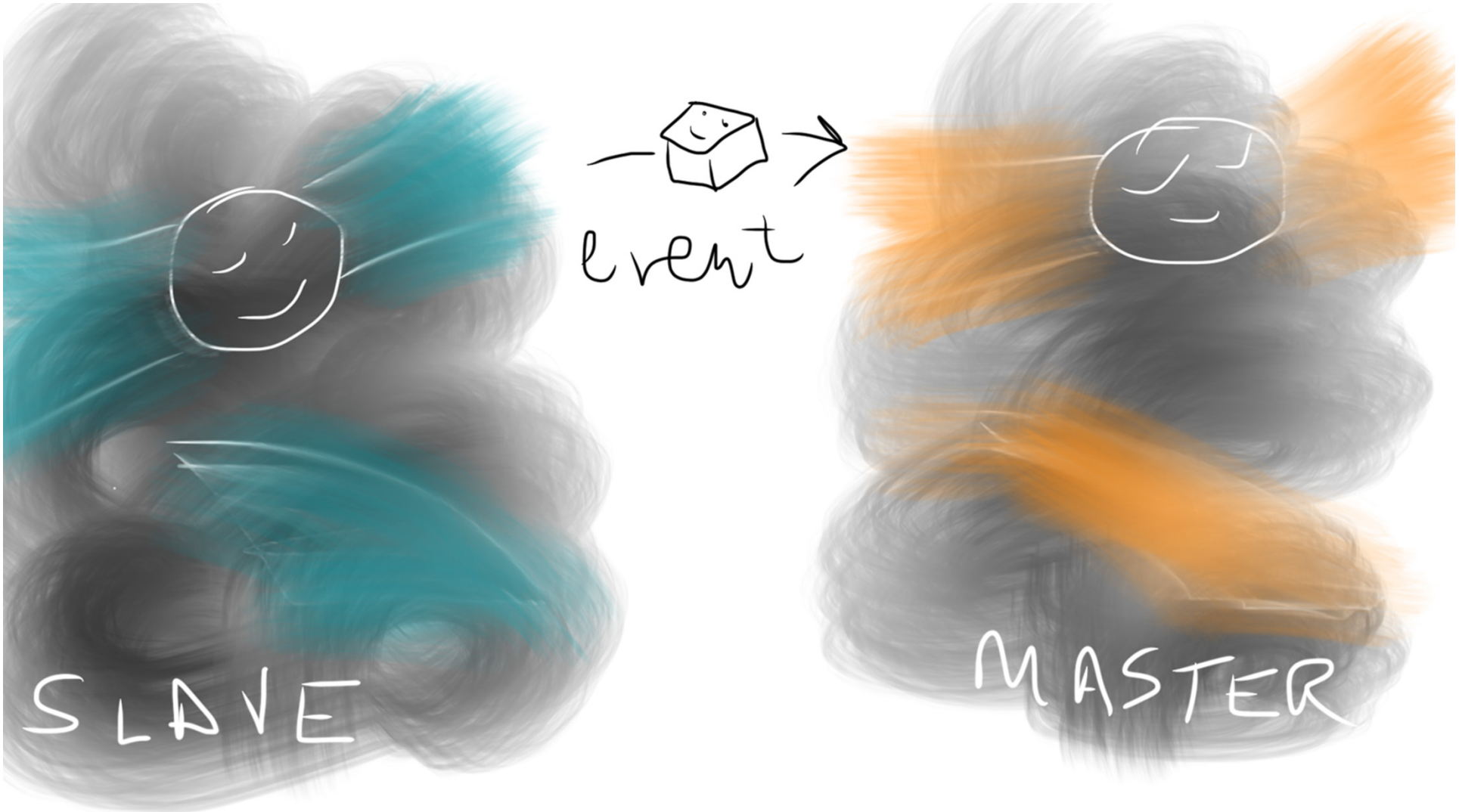


SUMMARY

Automation without overrides is trouble. We initially wrote Satellite without manual overrides and found times we wanted to turn hosts off – say there was a bad deployment on that host – that were task black holes. Unfortunately, Satellite would turn them back on immediately.

Satellite lets you set manual overrides over an HTTP REST interface; Satellite will ignore the auto-administration command, while you have set a host to be on/off manually.

Ex: If you want to perform some maintenance work on a set of hosts – you can loop through those hosts in a bash script and make sure no new tasks are sent to them during your maintenance window.


event
SLAVE
MASTER

SUMMARY

High level view of the Satellite architecture:

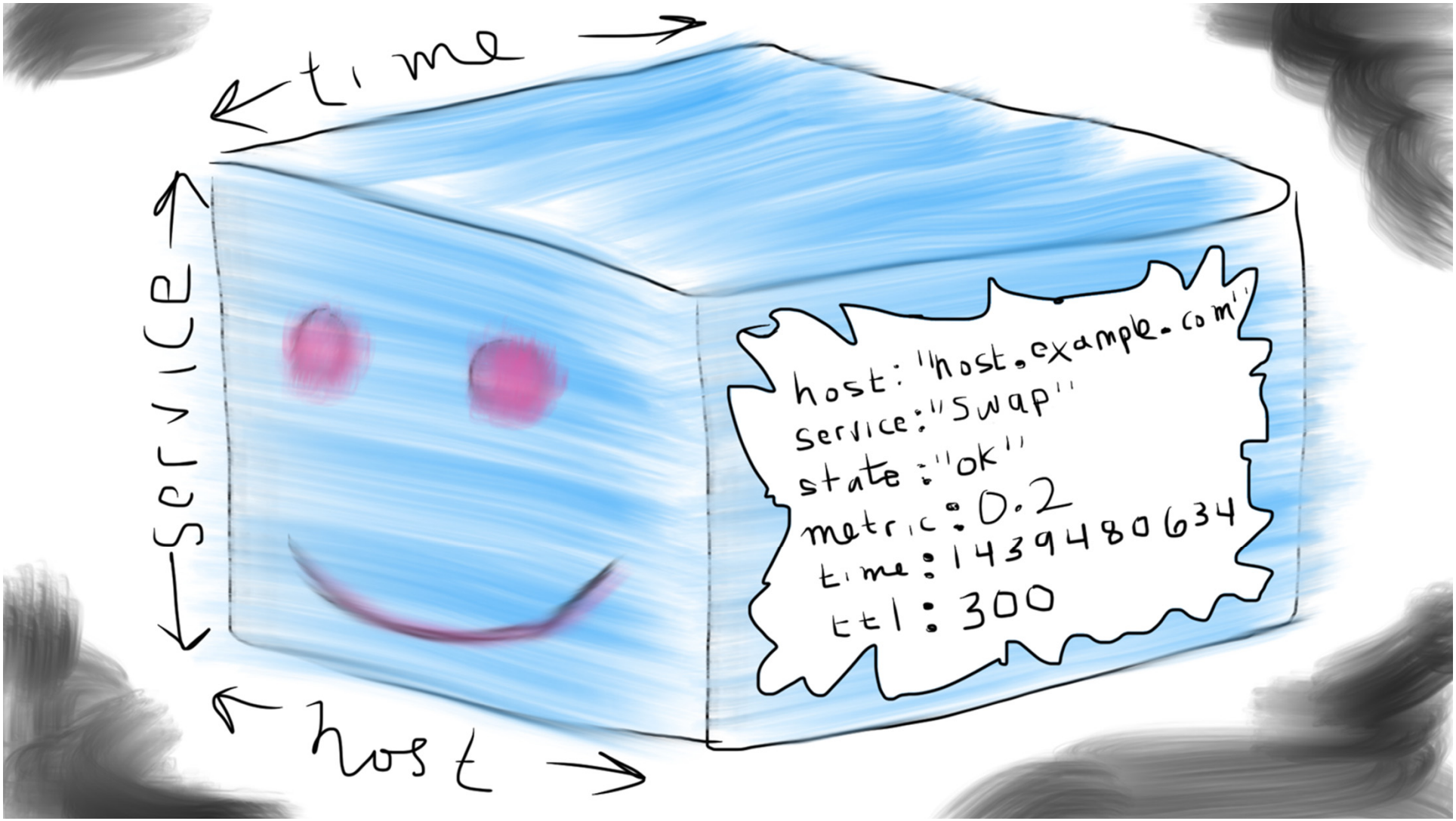
In the normal Mesos architecture there are two type of hosts, master and slave hosts, on which mesos-master and mesos-slave processes reside, respectively.

In Satellite, this architecture is preserved; there is a satellite-master process that co-exists on each master host, and a satellite-slave process that co-exists on each slave host.

The Satellite slave periodically pushes to the Satellite masters an update of its status.

The Satellite slave communicates to the satellite master over TCP; its message is a Riemann event.


event
SLAVE
MASTER

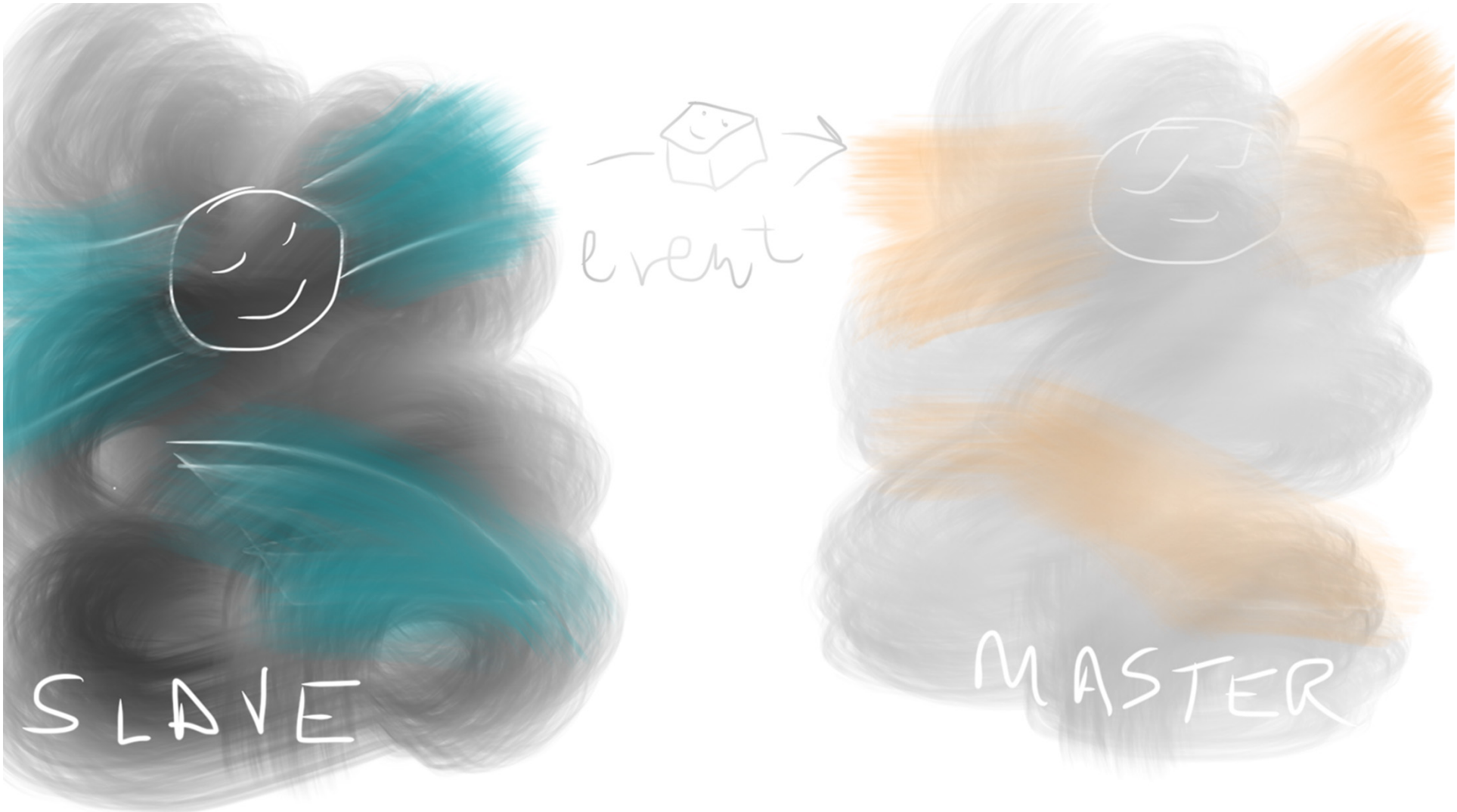

time
service
host
host: "host.example.com"
service: "swap"
state: "ok"
metric: 0.2
time: 1439480634
ttl: 300

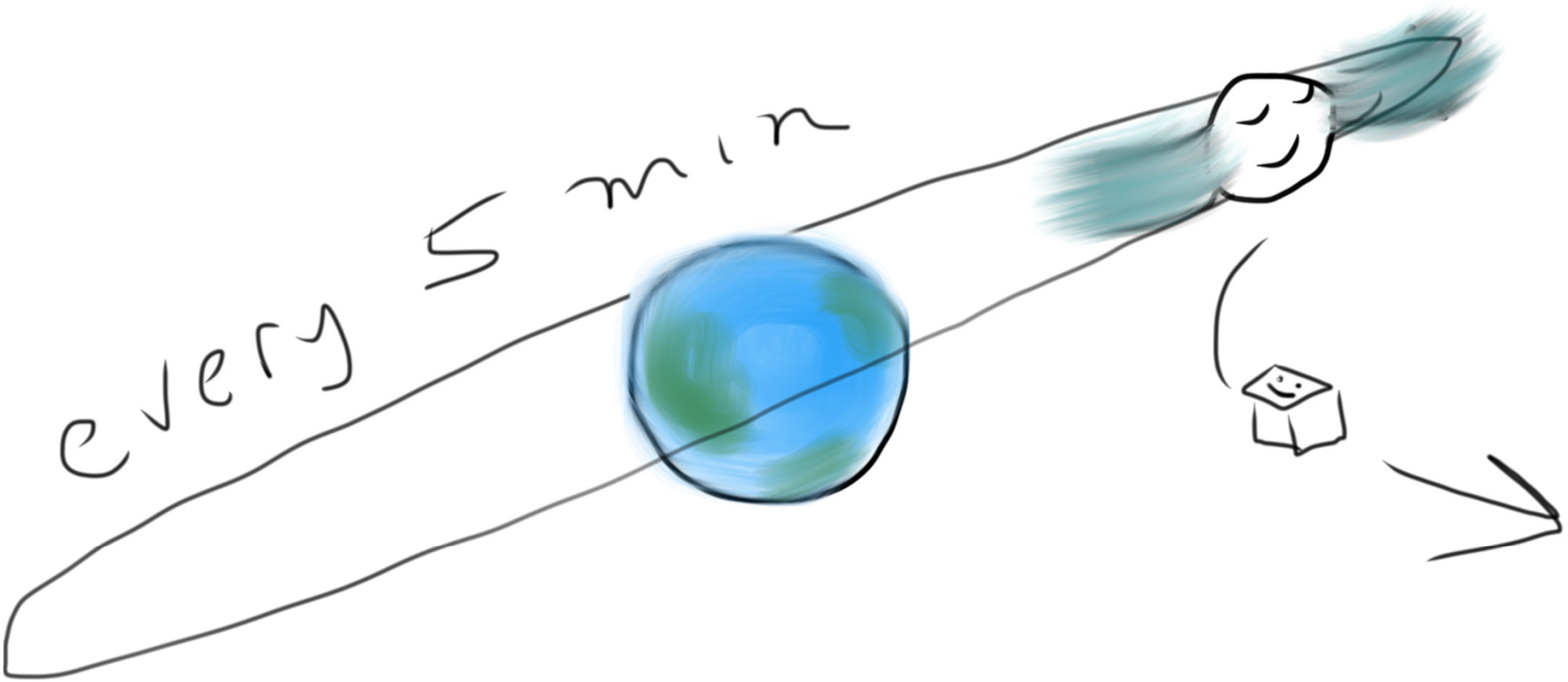
SUMMARY

The update is a Riemann event. Riemann events are just key-value maps.

A Riemann event is identified by the host it is coming from, the service, which is a string name, and the time the event is valid for.

Conventionally, and optionally, there are also fields like “state” and “metric” that we will talk about later.


event
SLAVE
MASTER


every 5 min

SUMMARY

The satellite-slave takes a user specified list of tests.

A comet (test) is just a shell command we run periodically and whose output we convert into a list of Riemann events

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

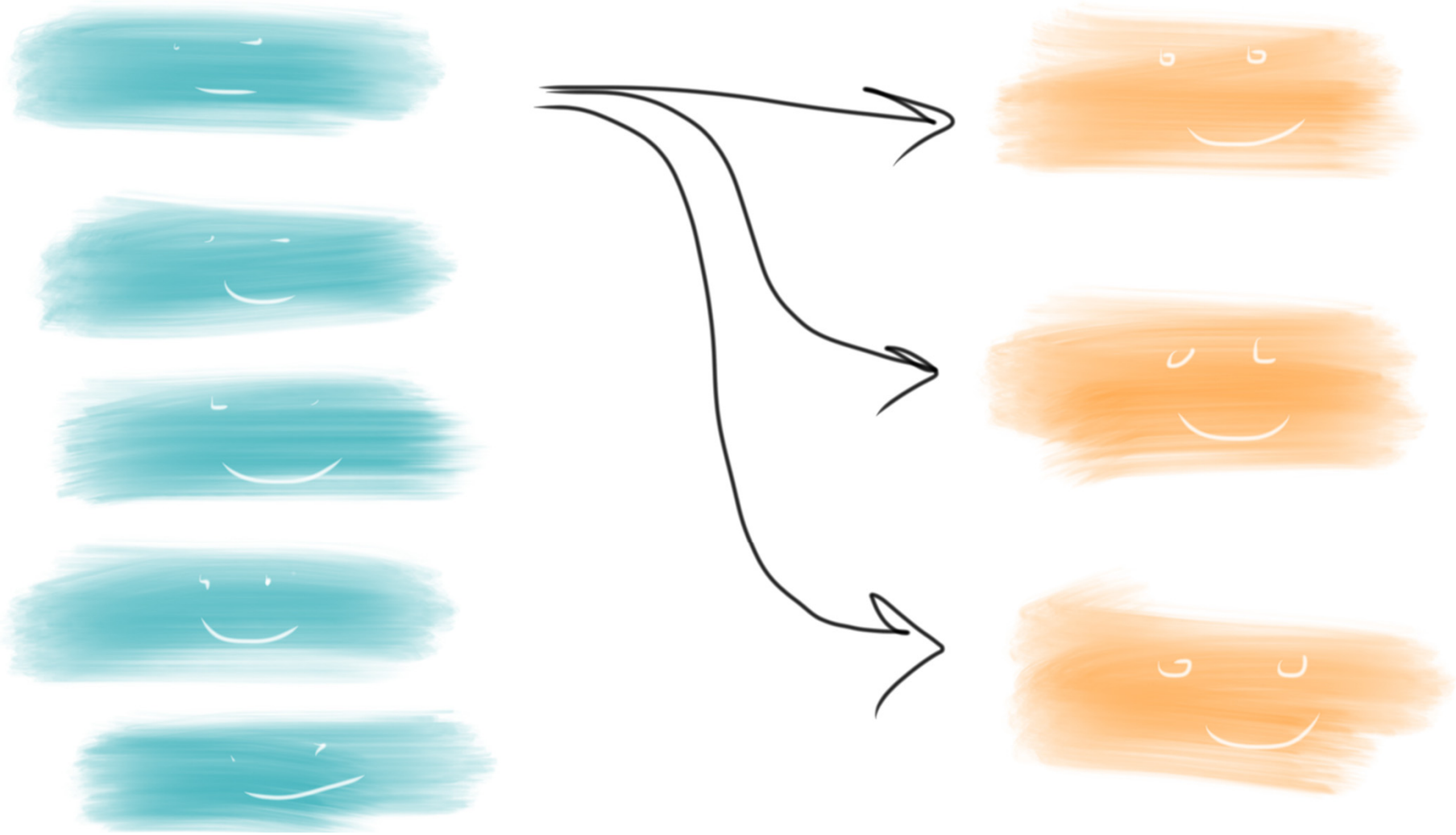
```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric exit
             :ttl 300
             :service “echo returns”}])}
```

```
;; A comet (a Satellite Slave test)
{:command “echo 17”
 :schedule (every (-> 60 seconds))
 :output (fn [out err exit]
           ...
           [{:state (if (zero? exit) “ok” “critical”)
             :metric (if (zero? exit) 1 0)
             :ttl 300
             :service “num echo returns”}])}
```

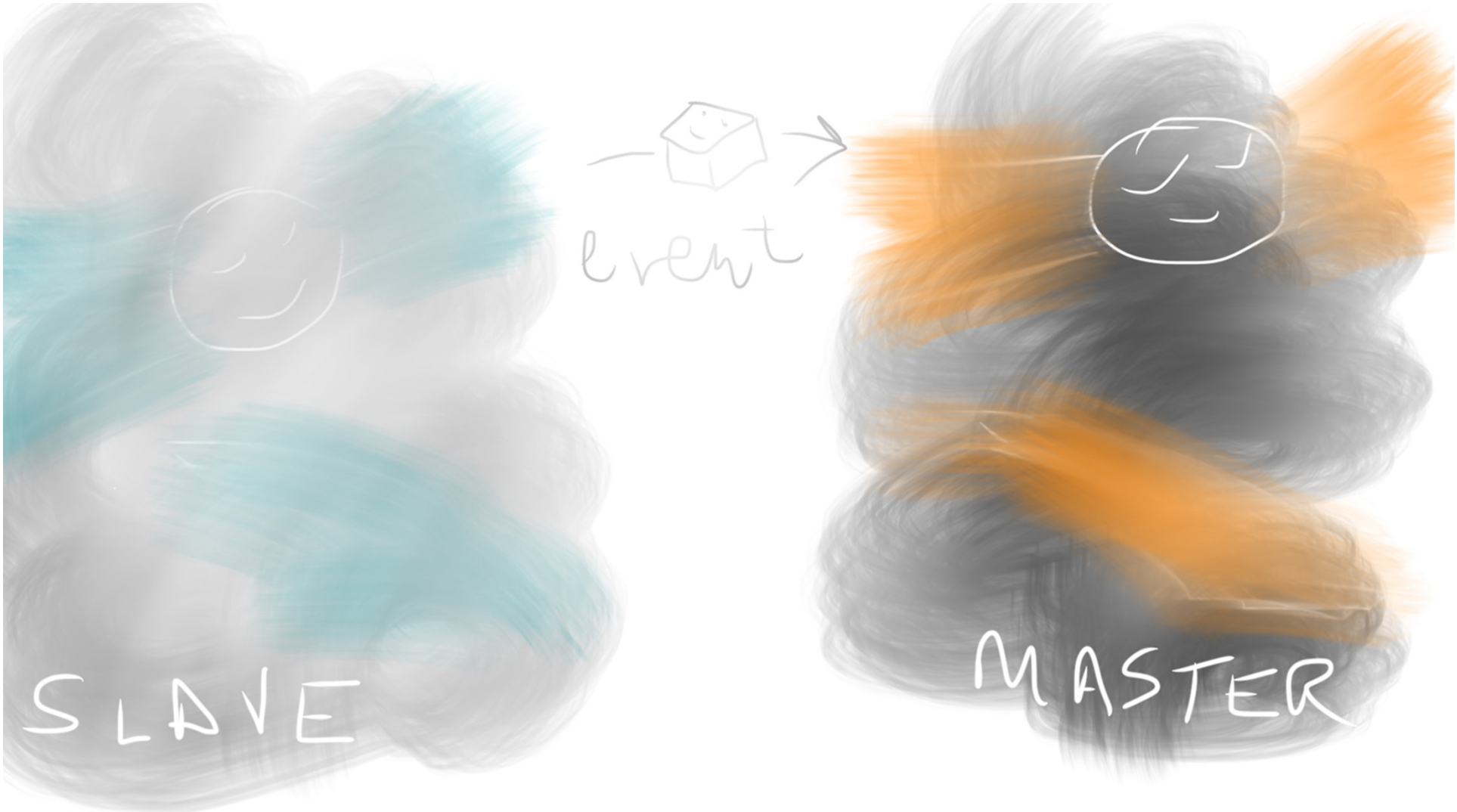
SUMMARY

Overview of a Satellite test



SUMMARY

Each slave emits its events to the masters. Now we're finished with the slaves.


event
SLAVE
MASTER


Riemann

SUMMARY

Satellite is able to perform its monitoring and alerting capabilities because we embed Riemann, a stream processor written by Kyle Kingsbury aka @aphyr, in the same JVM that Satellite runs in.

Riemann is a stream processing system that provides many primitives / functions for monitoring and alerting.

What you don't find in Riemann, you can make yourself – every Riemann config is a clojure/java program, so you have a full programming language available to you.

These are the reasons we choose Riemann – it was easy to extend, its data model suits our use case, and we already had experience with it. It also explains why the satellite project is written in Clojure – because Riemann is too.

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

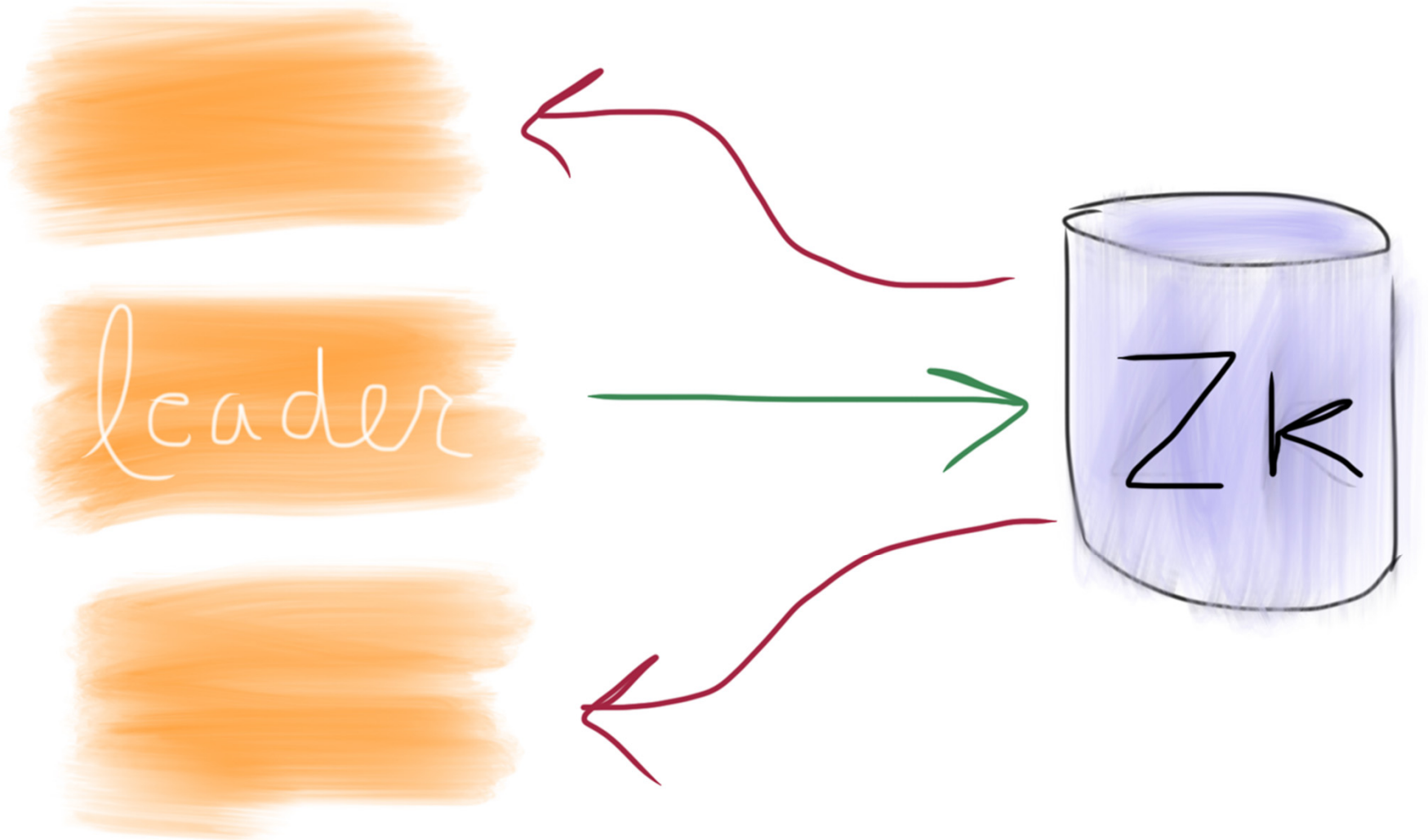
```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

```
;; only send tasks if enough swap
(where (service #”mesos/slave/swap”)
       (where (> metric 0.9)
              (off-host host)
              (else (on-host host))))
...
(def pd (pagerduty MY-SWEET-API-KEY))

(where (service #”mesos/prop-available-hosts”)
       (where (< metric 0.7)
              (:trigger pd)
              (else (:resolve pd))))
```

SUMMARY

Example of writing a Riemann config for Satellite that auto-administers for high swap utilization and sends an alert via Pagerduty if there cluster availability falls below a threshold.

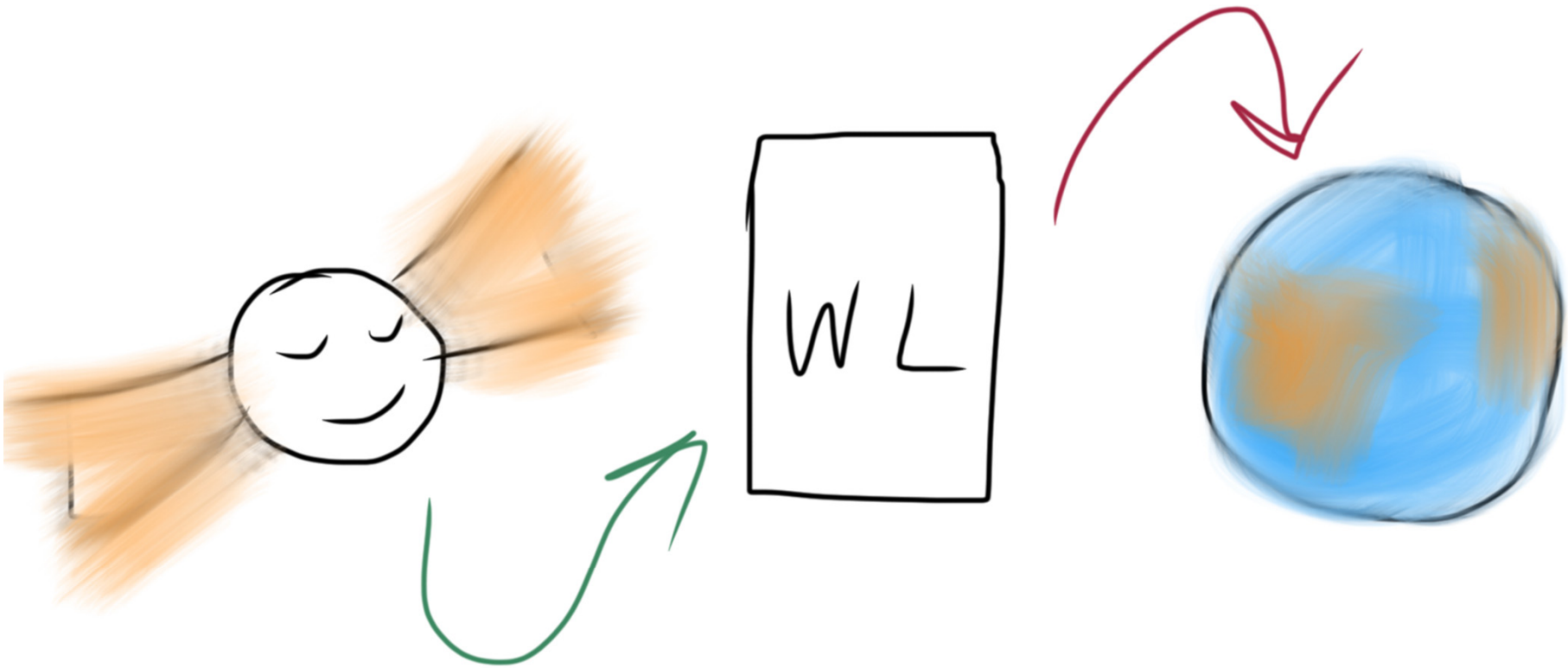

leader
Zk

SUMMARY

Every master should see every message.

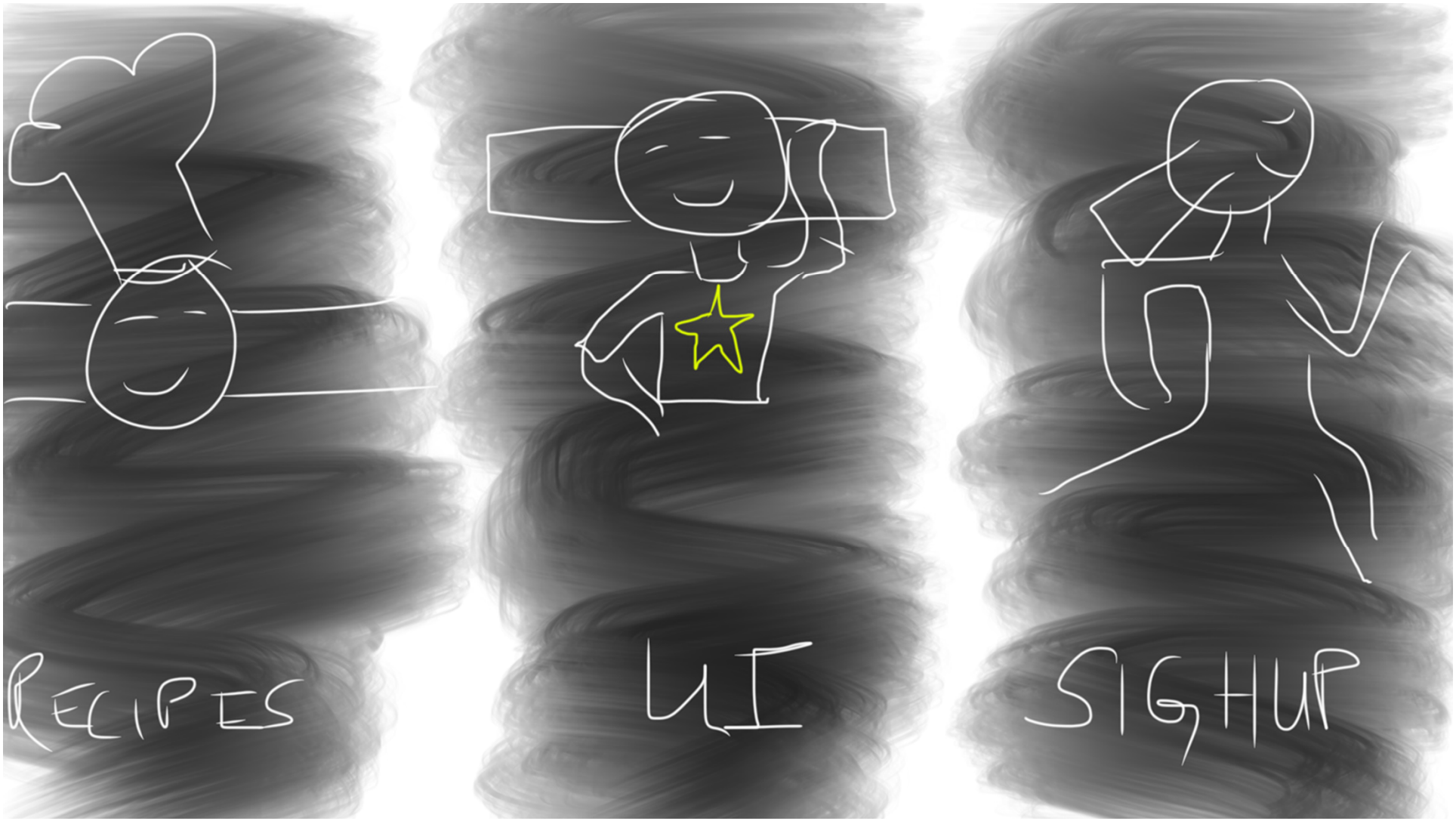
To ensure that, generally, there is one and only one action is by having the leader be the stream processor. Any state changes – hosts that are turned on/off and the reasons for why the transition happened – are written to Zookeeper by the leader.

These changes are read by the follower masters, so they are up to date during a Mesos failover.


W L

SUMMARY

Satellite communicates to Mesos through the whitelist file


RECIPES
LI
SIGHUP

SUMMARY

Future work includes

- More recipes
- Improving the web UI
- Being able to hot reload configs through a SIGHUP


THIS IS FINE.

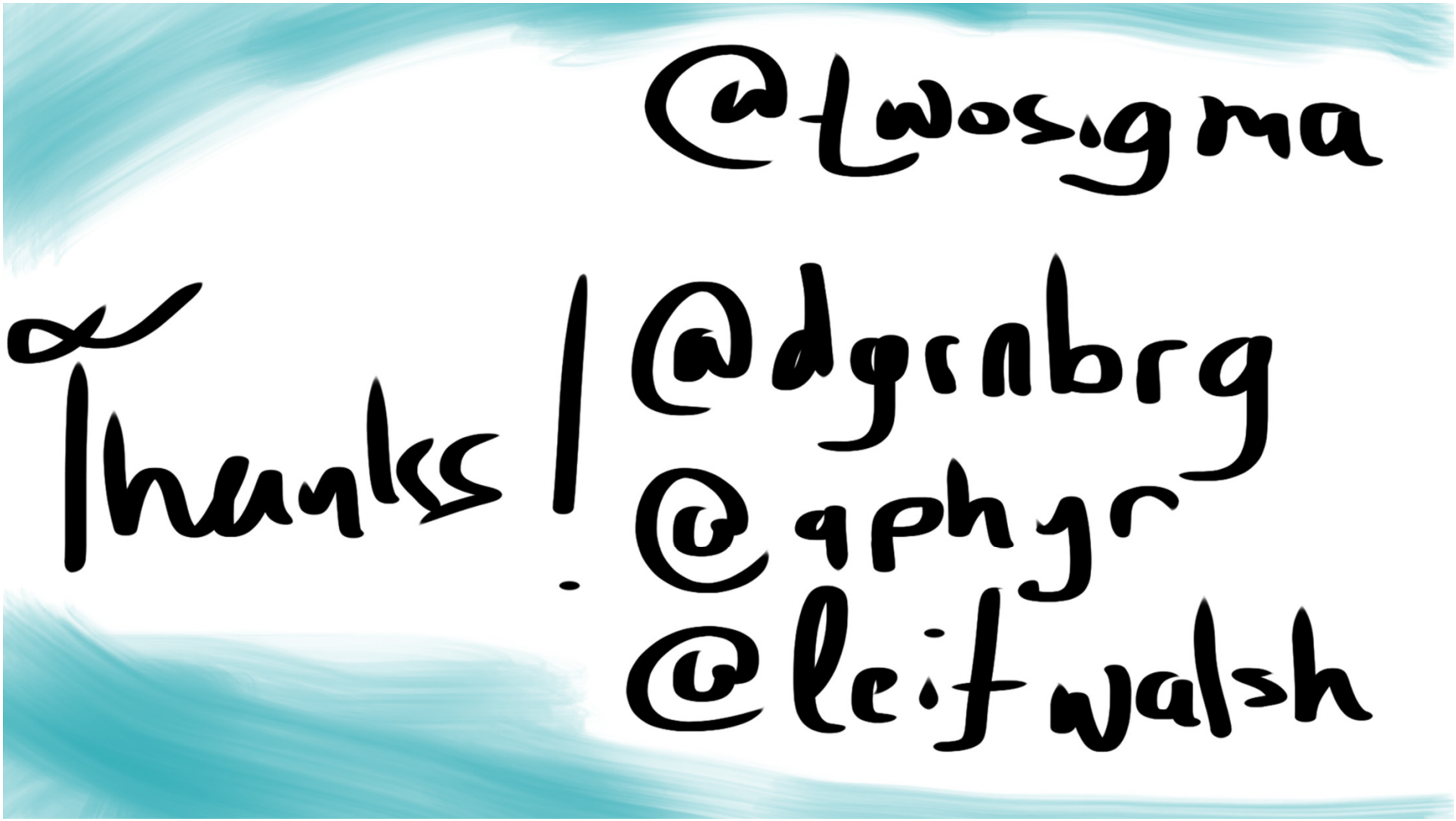
SUMMARY

- In production for almost a year at Two Sigma
- Manages multiple clusters and thousands of non-commodity hosts

When I deployed Satellite the first time, 20% of the cluster was turned off immediately. At first I thought I had done something wrong, but Satellite was doing everything what it should.

In fact, we had a lot of hosts whose swap was completely utilized. It turns out there were a number of jobs that were stuck, and thrown into swap – they had been sitting there for weeks. What's worse is that this was cutting into the quota of the users who had originally launched those jobs.

This fast failure doesn't extinguish any fires, but rather forces you to pay attention when things are truly awry.


@lwosigma
Thanks! @dgrnbrg
@aphyr
@leif_walsh


/twosigma
/satellite