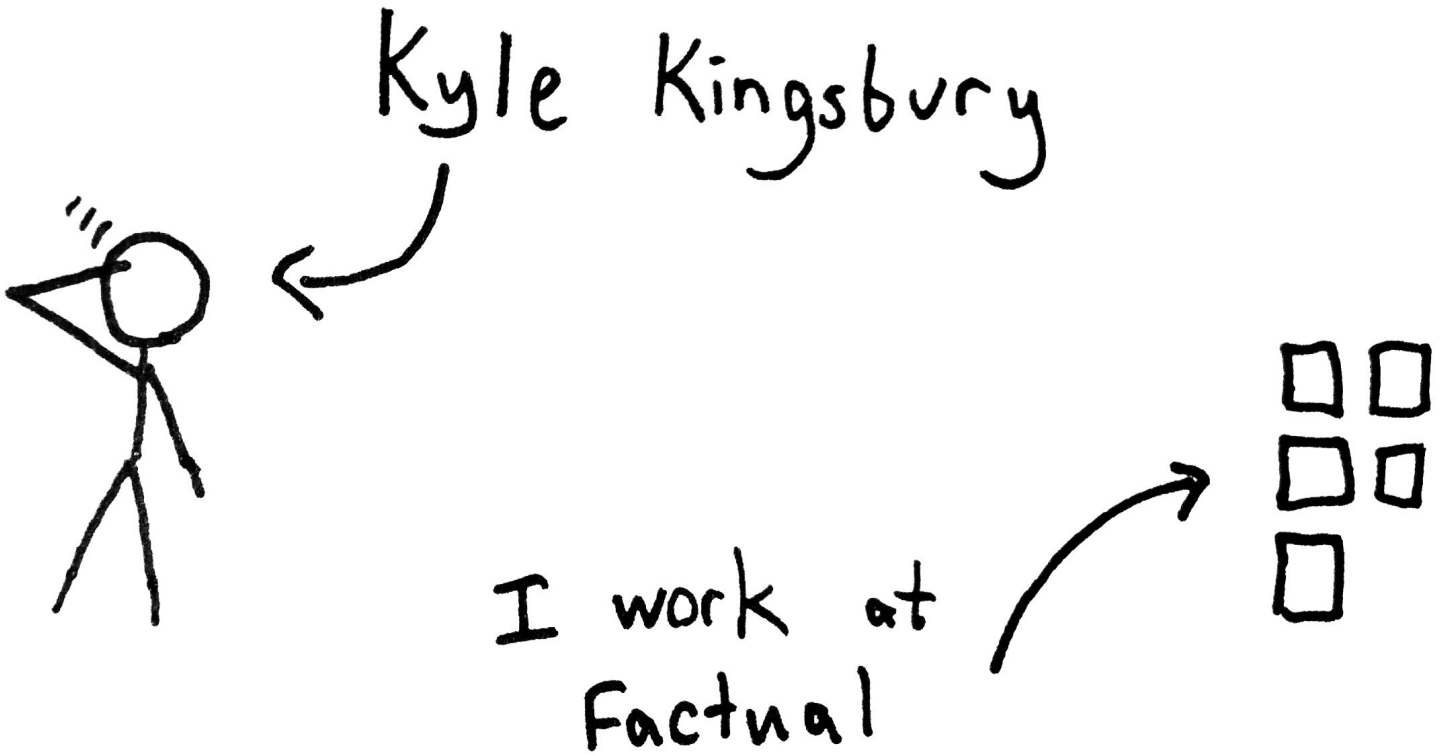
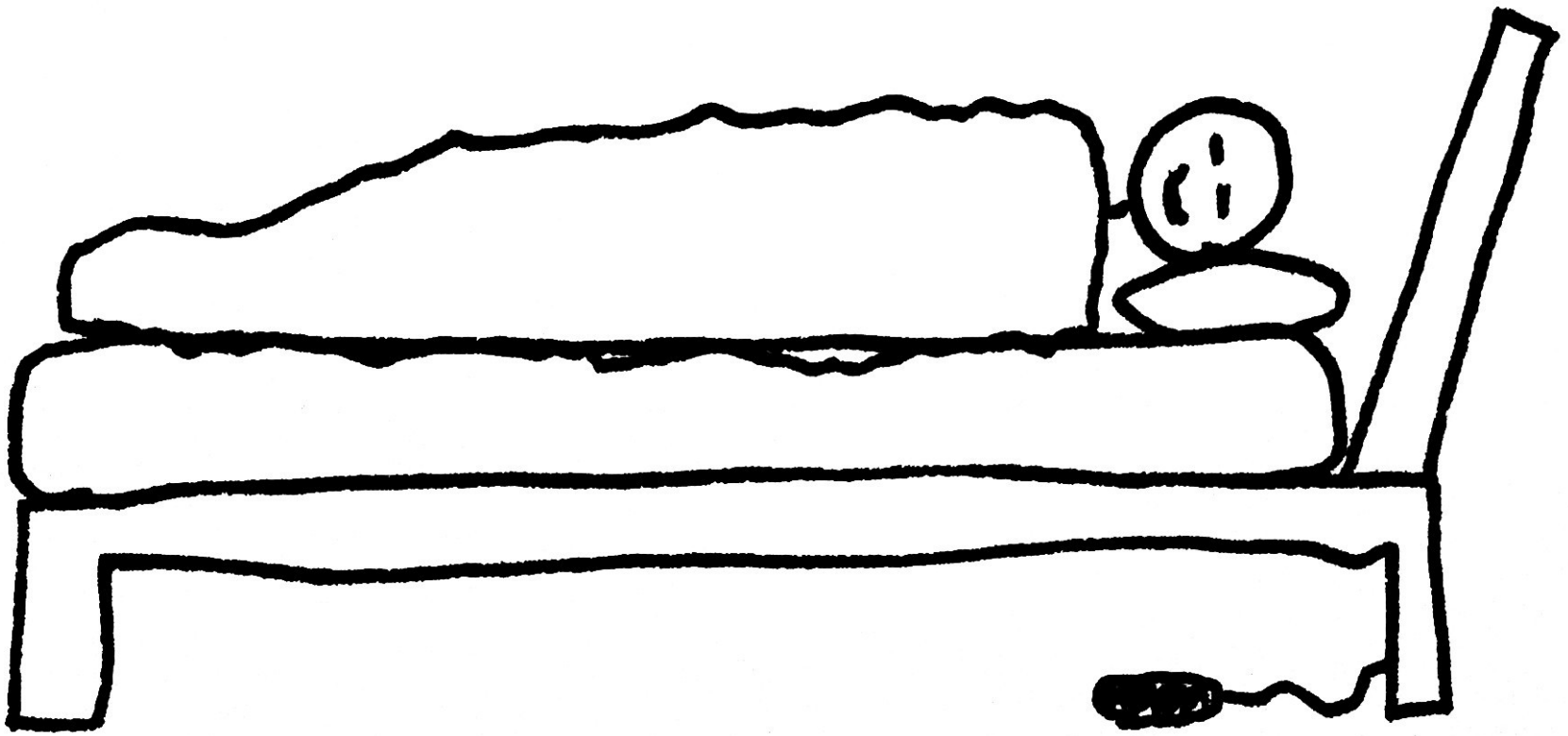
RIEMANN

version 0.2.0

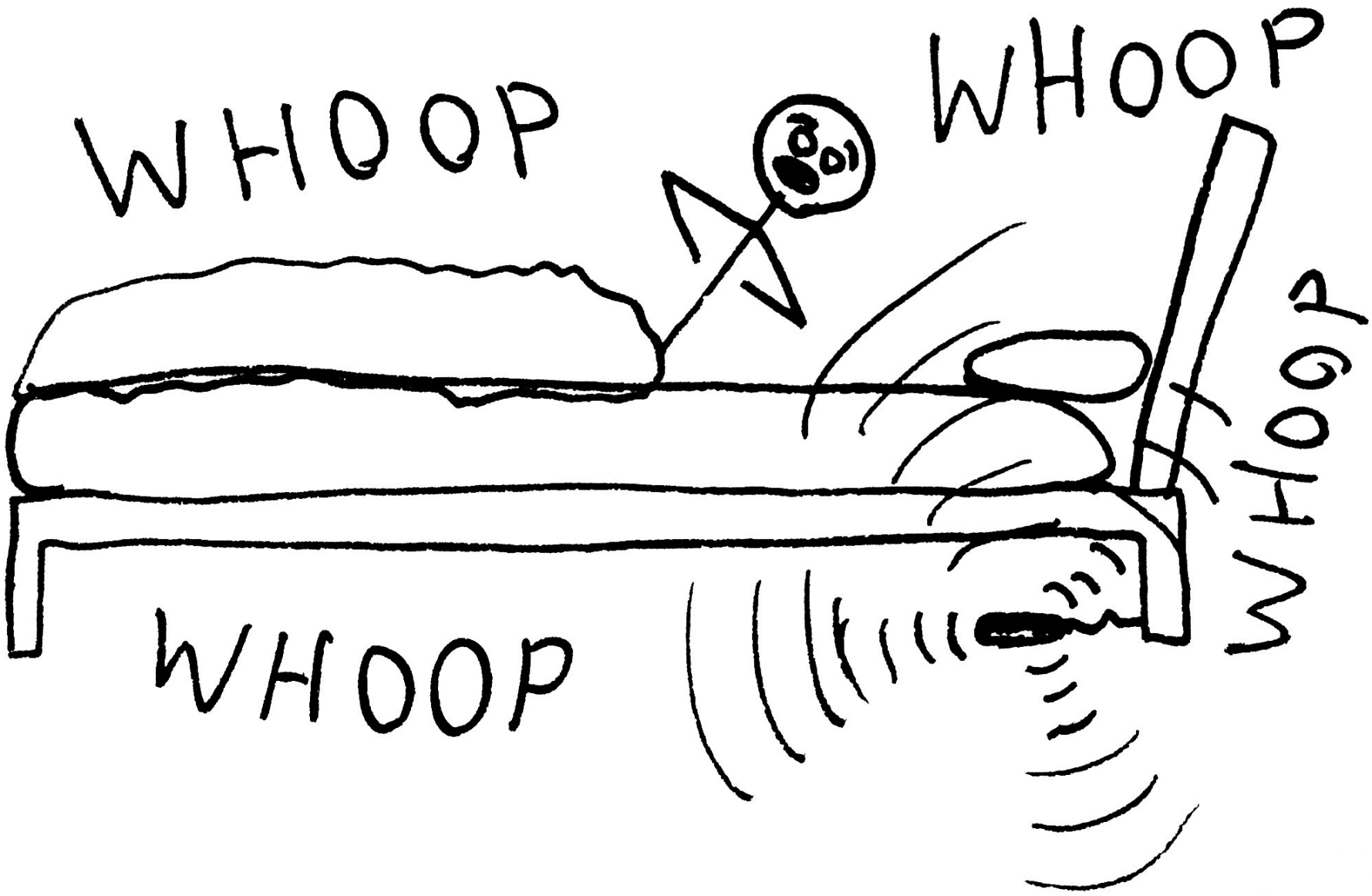

Kyle Kingsbury
I work at Factual

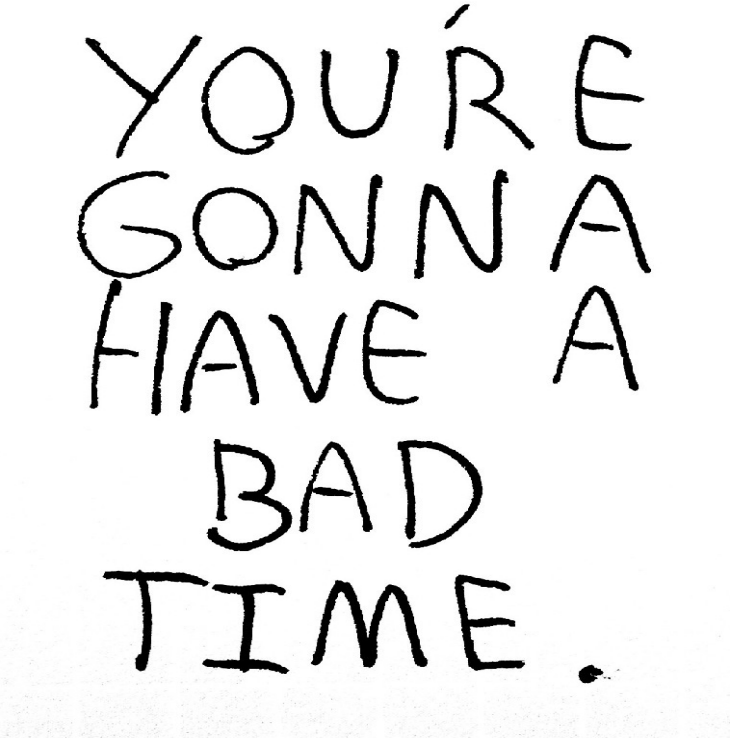
A Story!

02:51 am



02:52 am

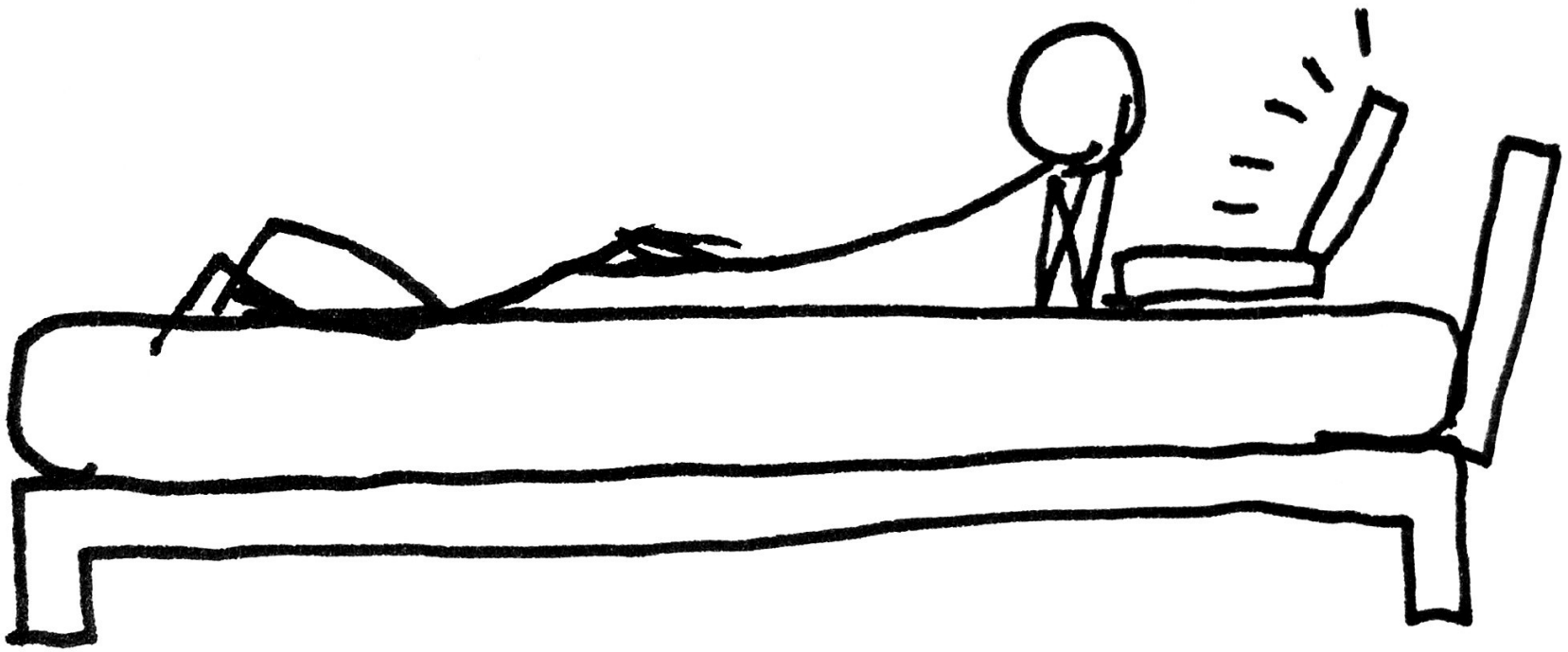



YOU'RE
GONNA
HAVE A
BAD
TIME.

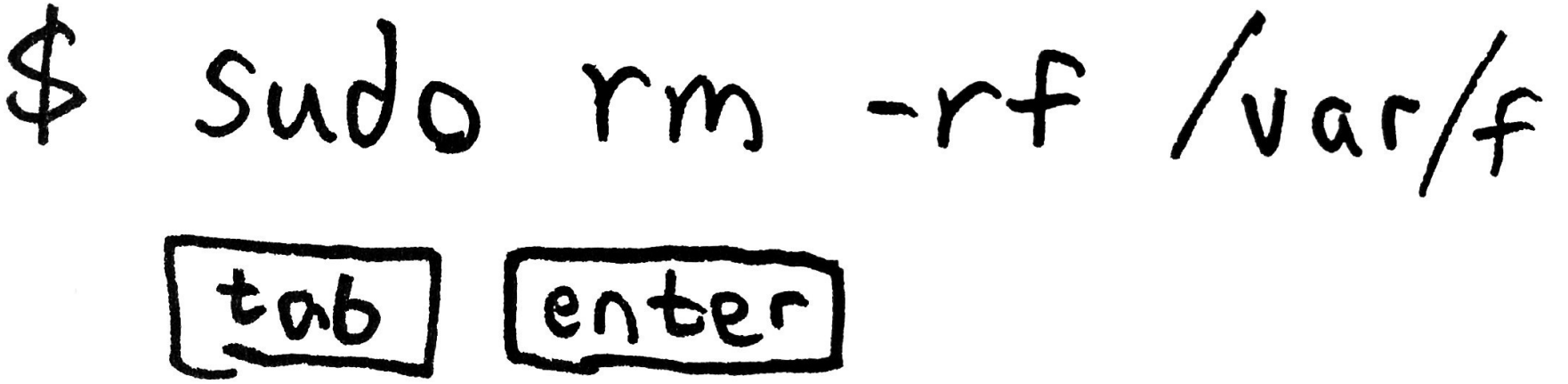

PACER DUTY
AL EE D-
OWN I, STS ON
SERV. AGIOS

WHAT DO YOU DO?

03:28 AM



Concerns, in order of importance	1. Why me? 2. What even is broken? 3. Rational thought

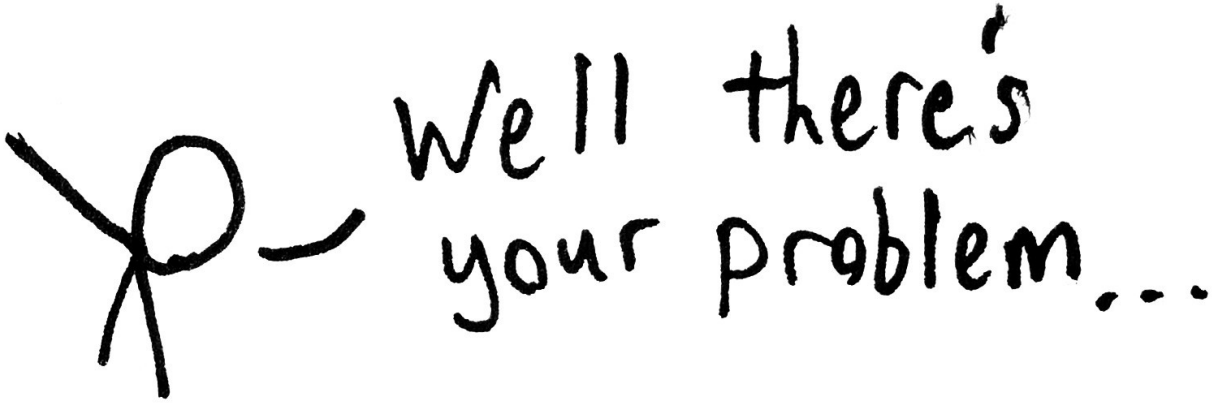

$ sudo rm -rf /var/f
tab
enter

Can't change your job.

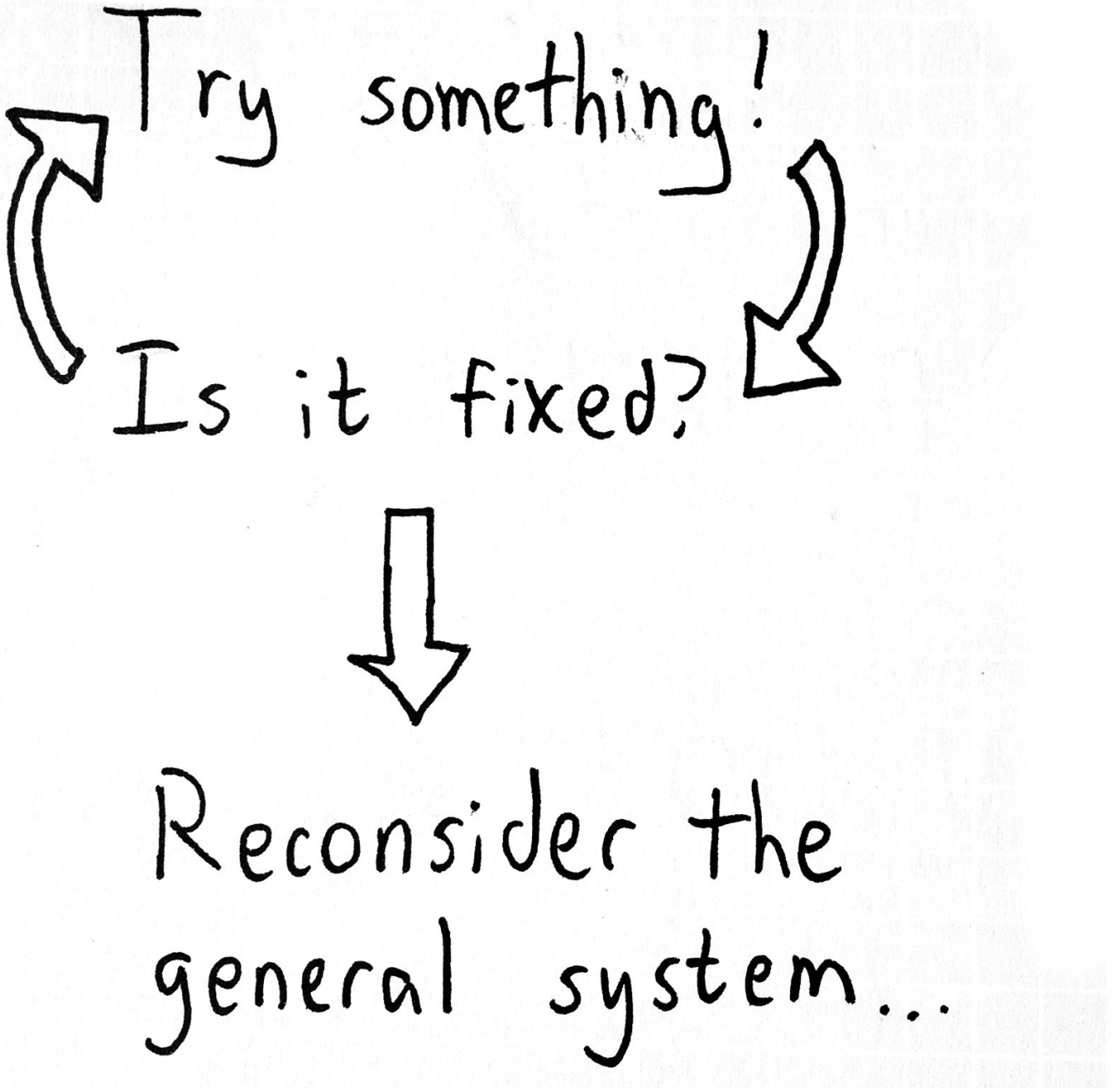
but we can...

- Break things less often.
- Make them easier to fix.
- Reduce the need for hard thinking.

OBSERVE
ORIENT
DECIDE
ACT


Well there's
your problem...

What's broken?
Where did it break?
When did it happen?
What's the cause?


Try something!
Is it fixed?
Reconsider the general system...

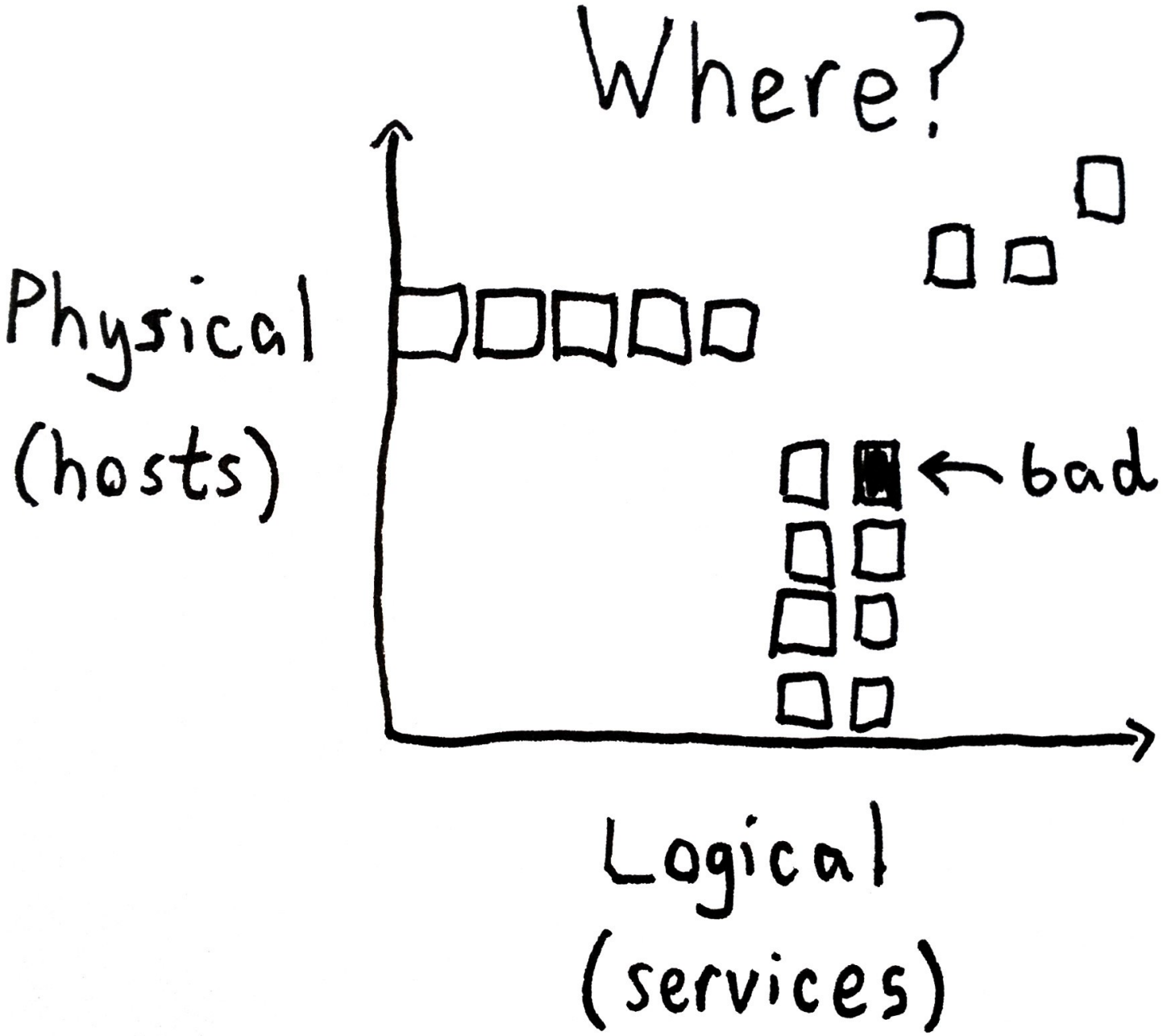
TWO FUNDAMENTAL PROCESSES

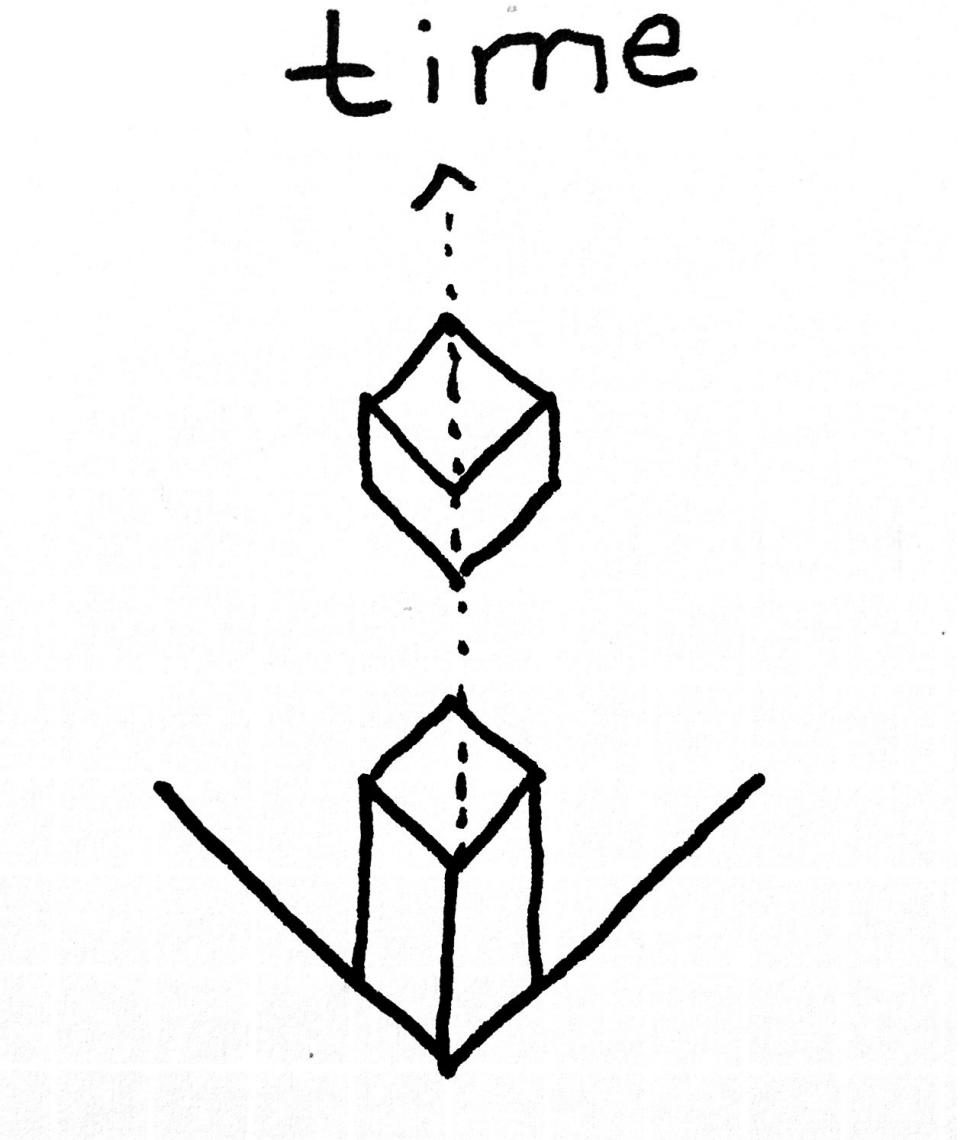
1. LOCALIZE
2. FIX

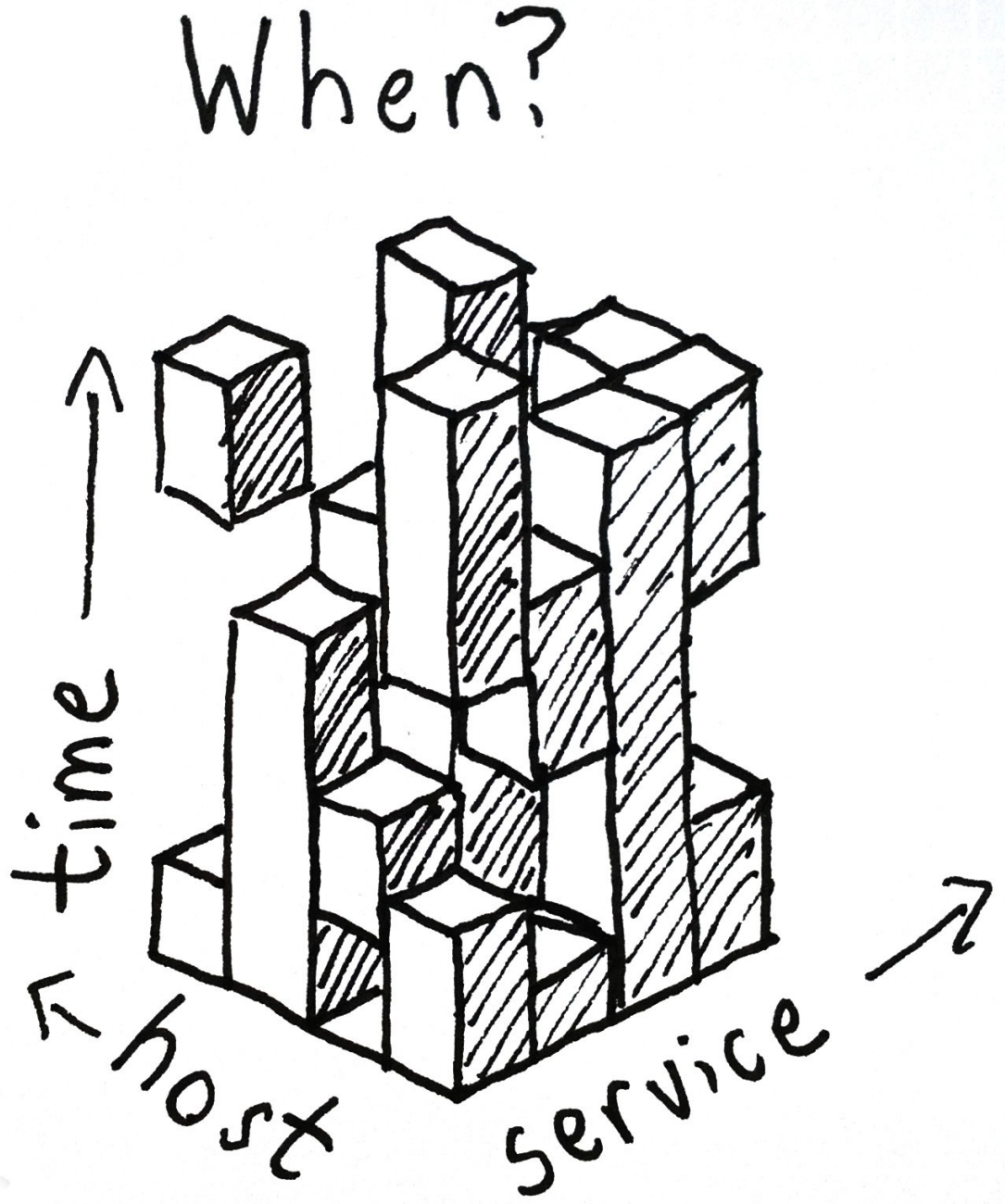
(repeat)

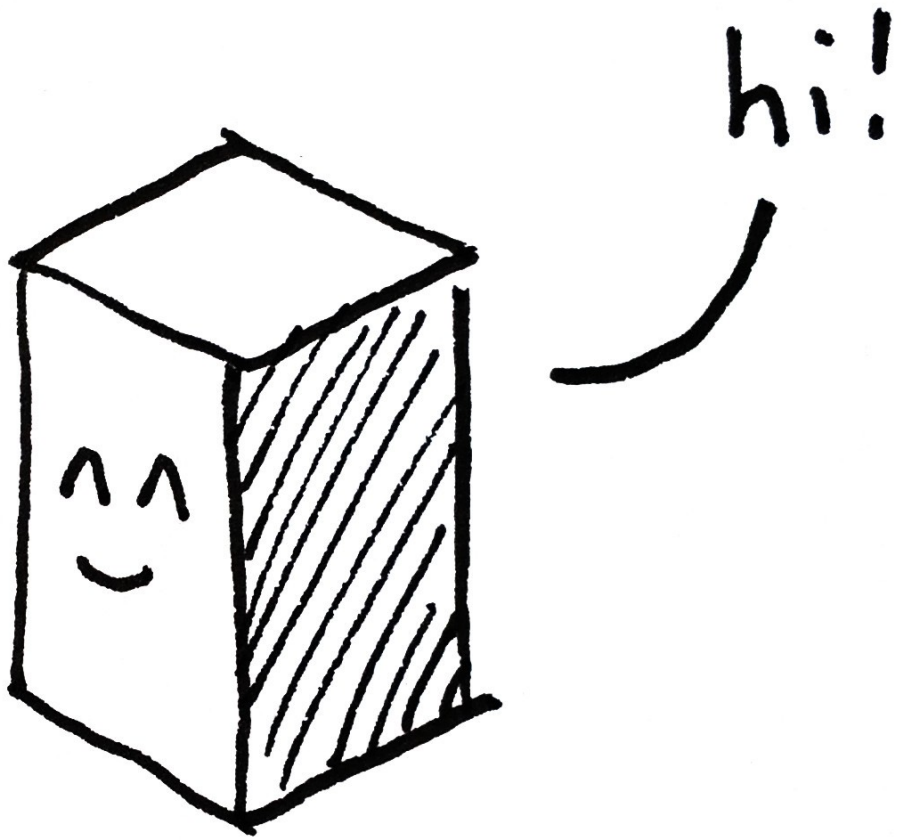
Lets Talk

Theory!

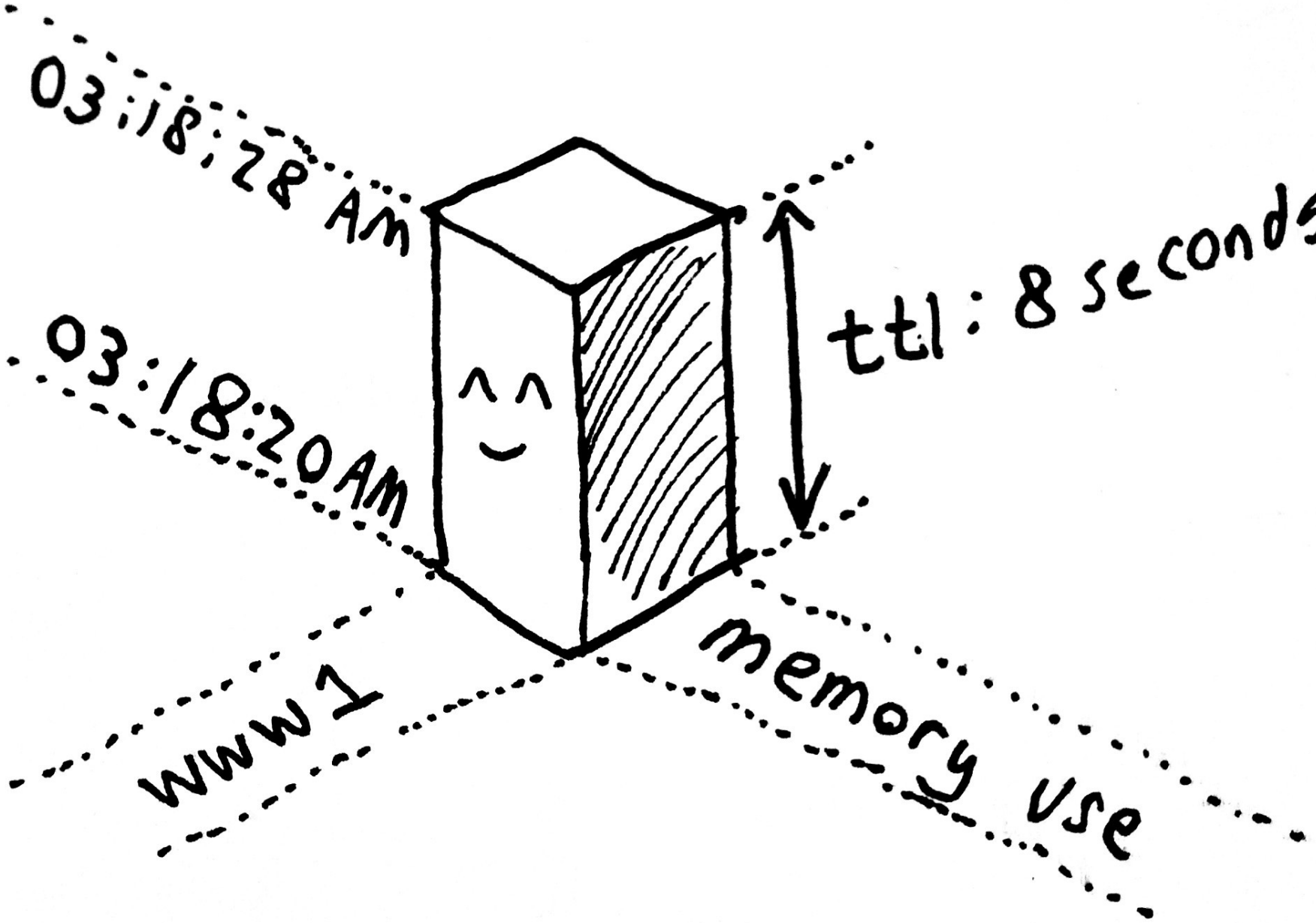

Where?
Physical
(hosts)
bad
Logical
(services)


time

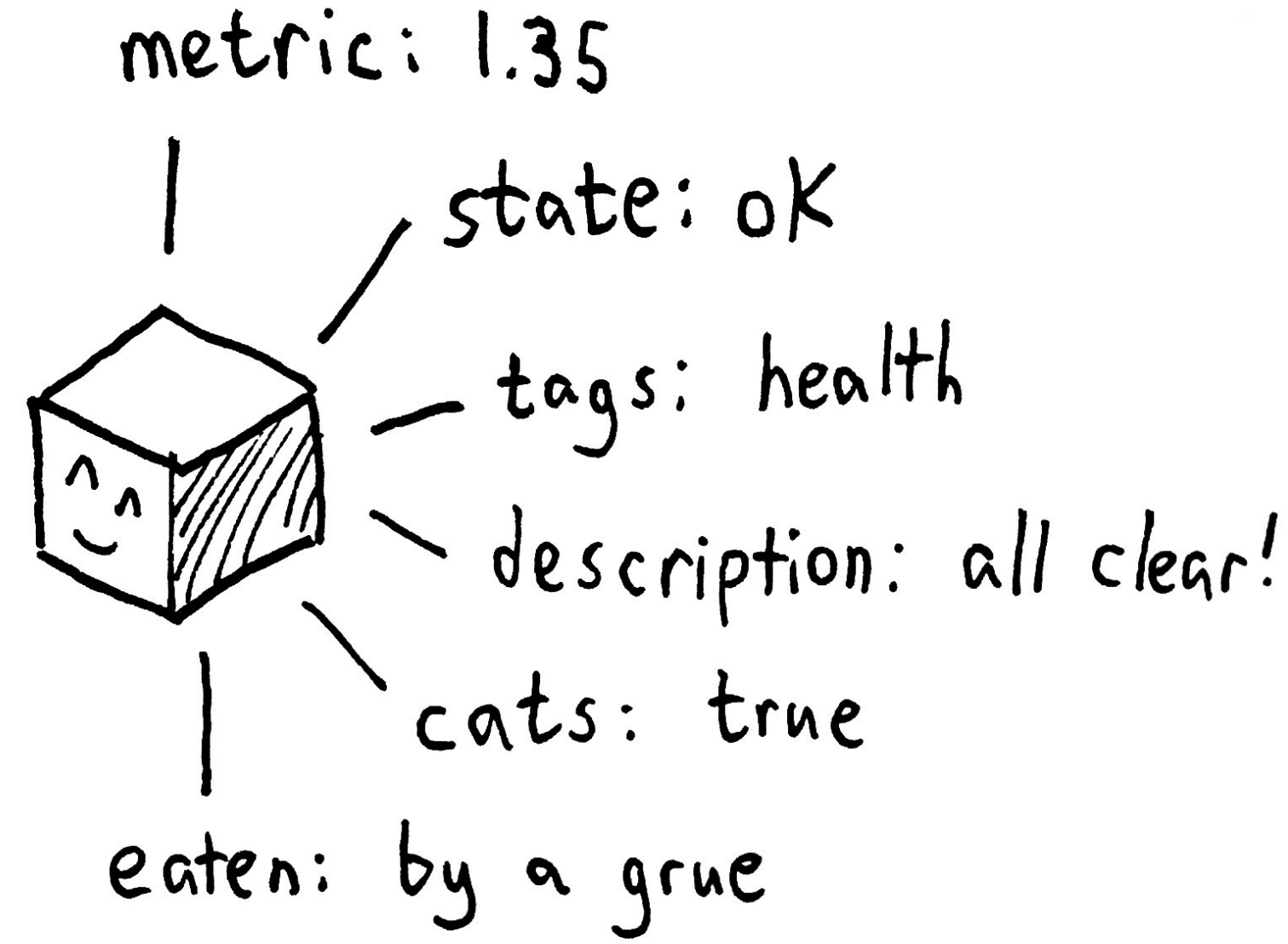

When?
time
host
service



An Event

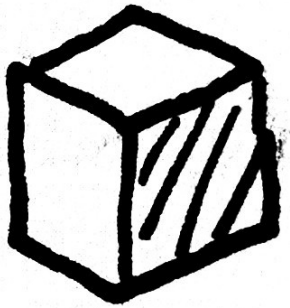


IDENTITY


metric: 1.35
state: oK
tags: health
description: all clear!
cats: true
eaten: by a grue

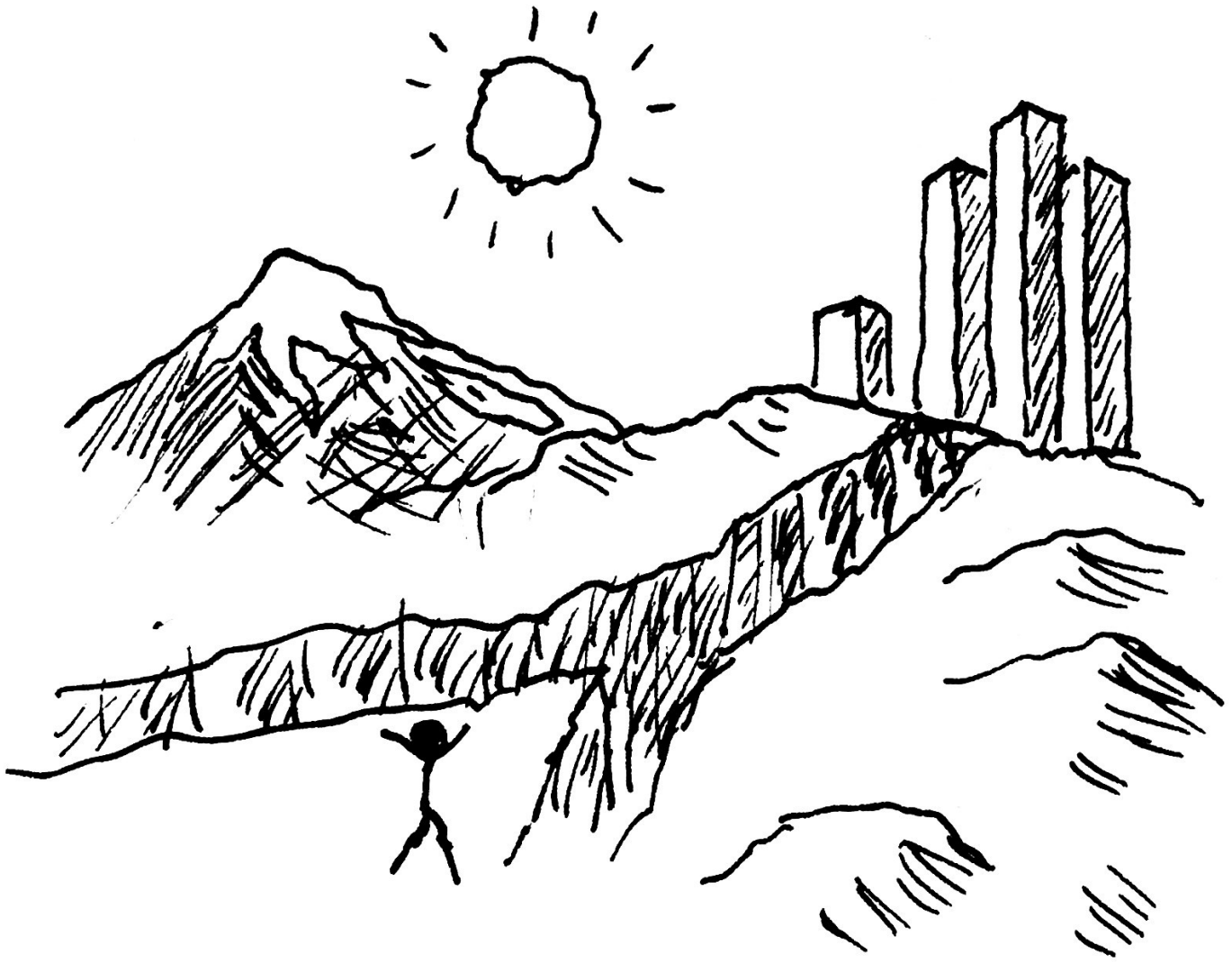
STATE

One event, alone,



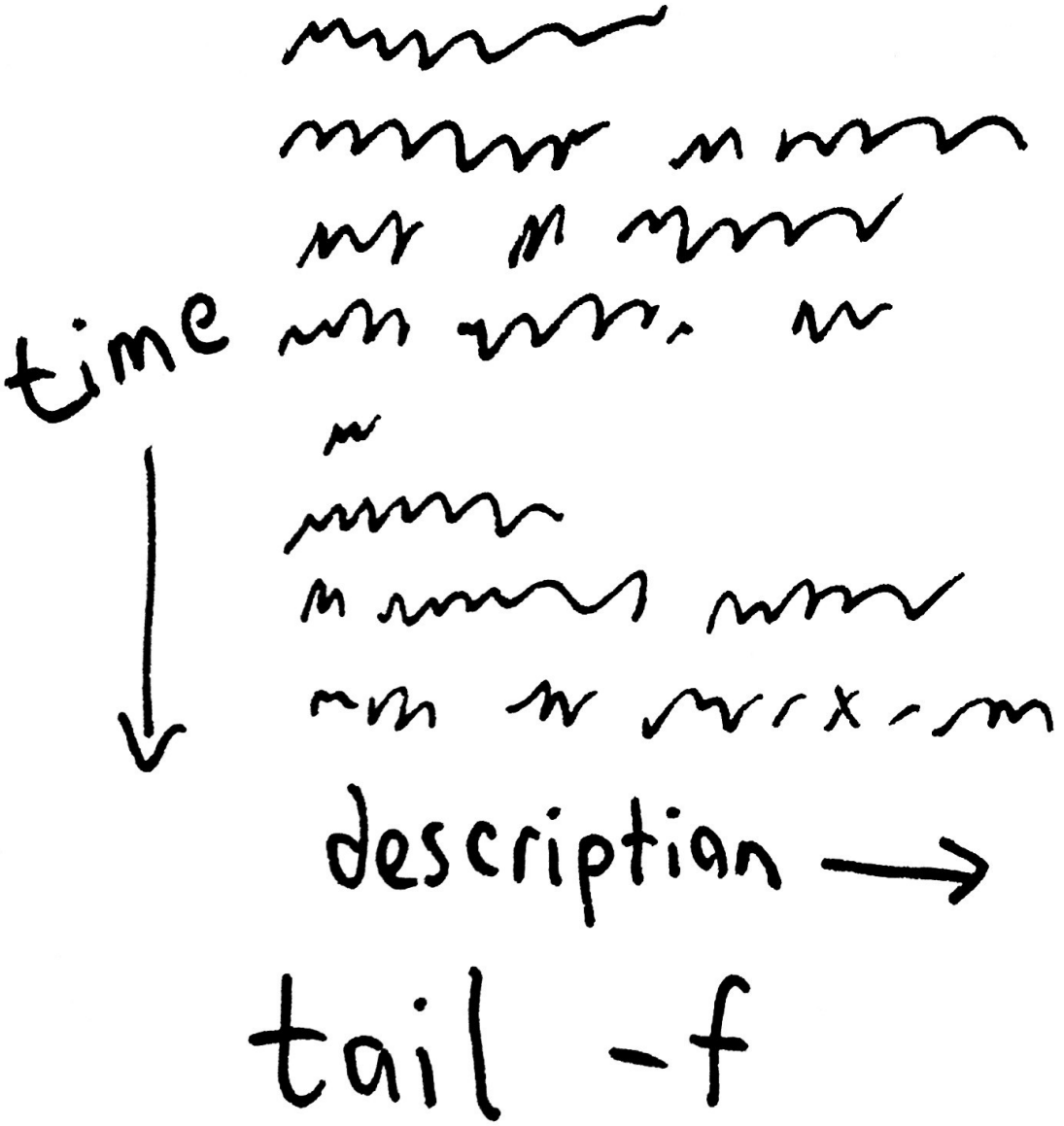
is boring.

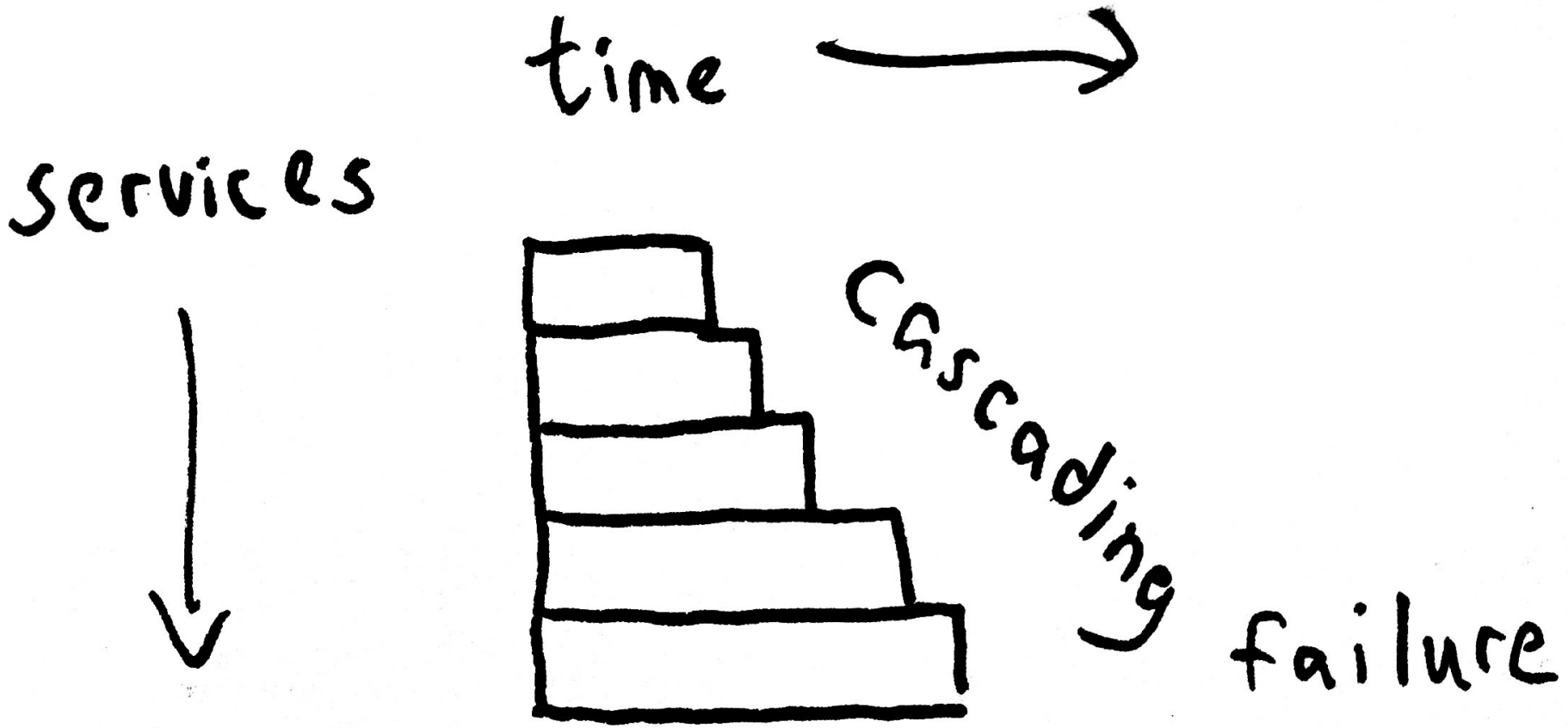
Many events in context

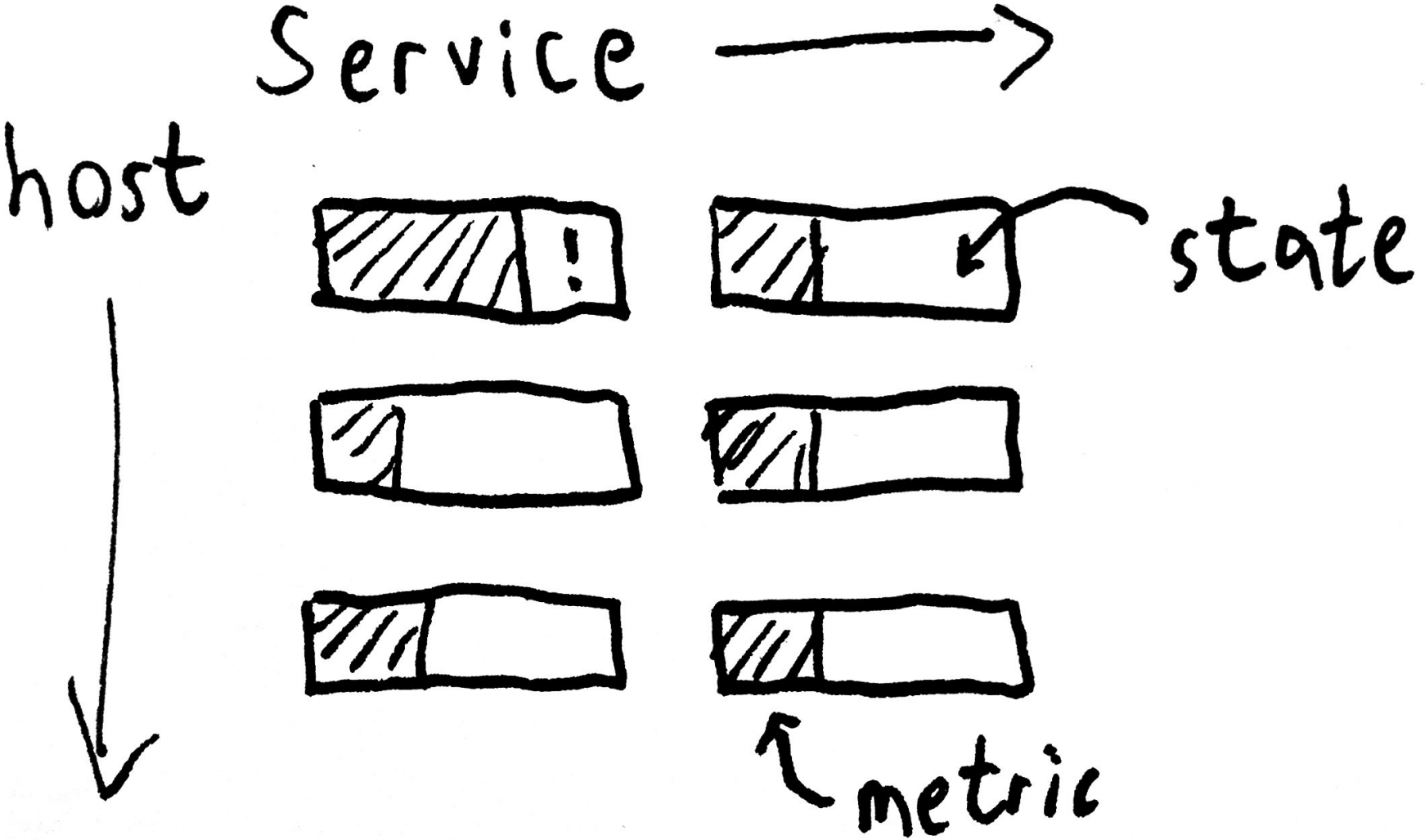


Form a Landscape

PROJECTIONS


time
description →
tail -f


time
services
cascading
failure


Service
host
!
state
metric


metric
time

RIEMANN'S JOB

is to

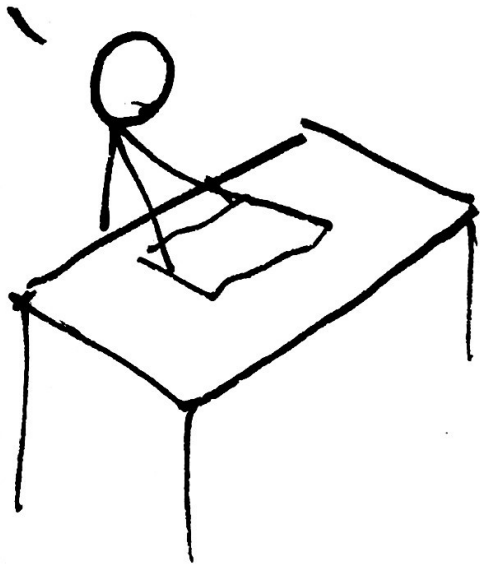
PROCESS, ANALYZE, & REACT

to the event stream

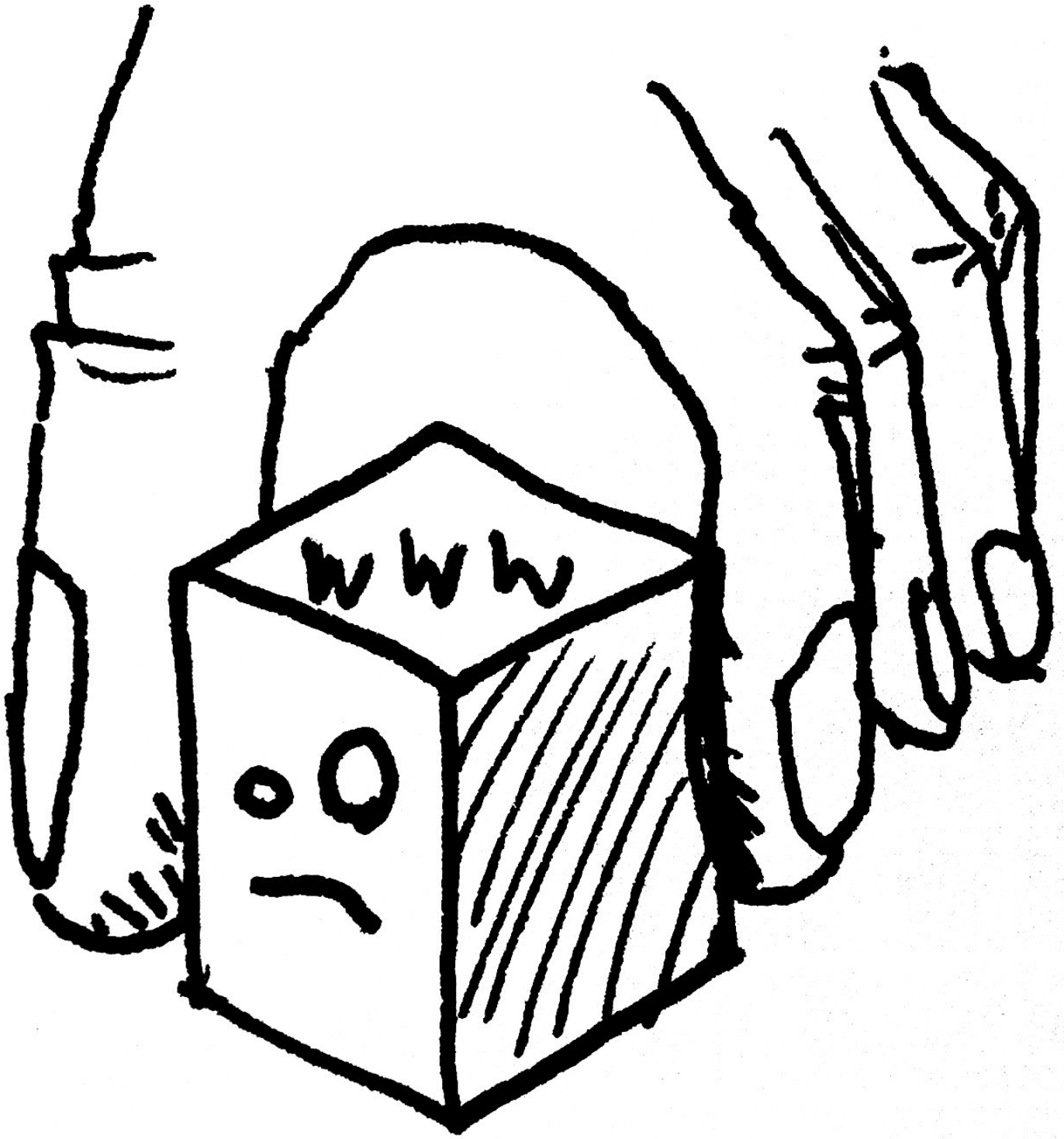
HOW?

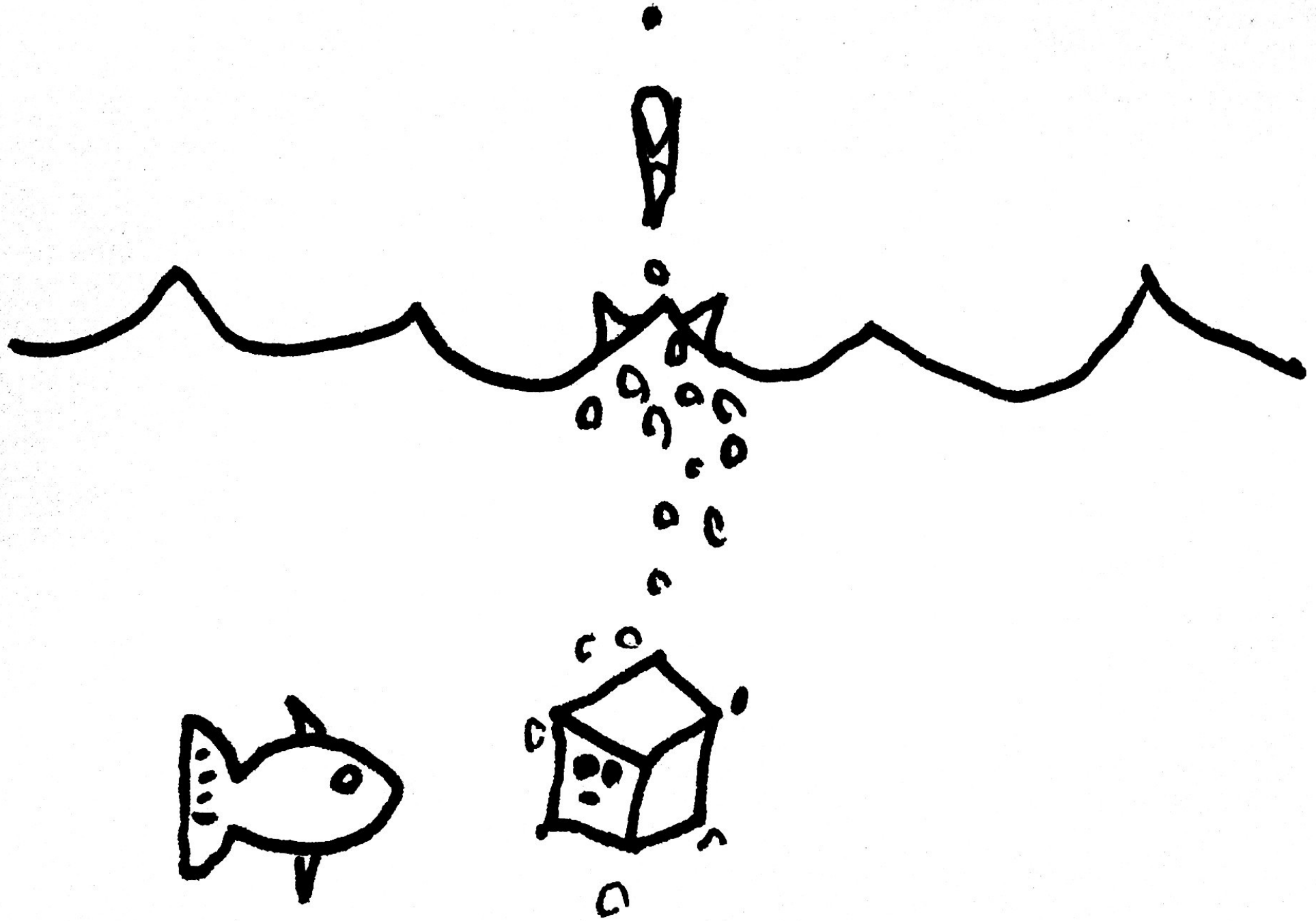
(L & P)

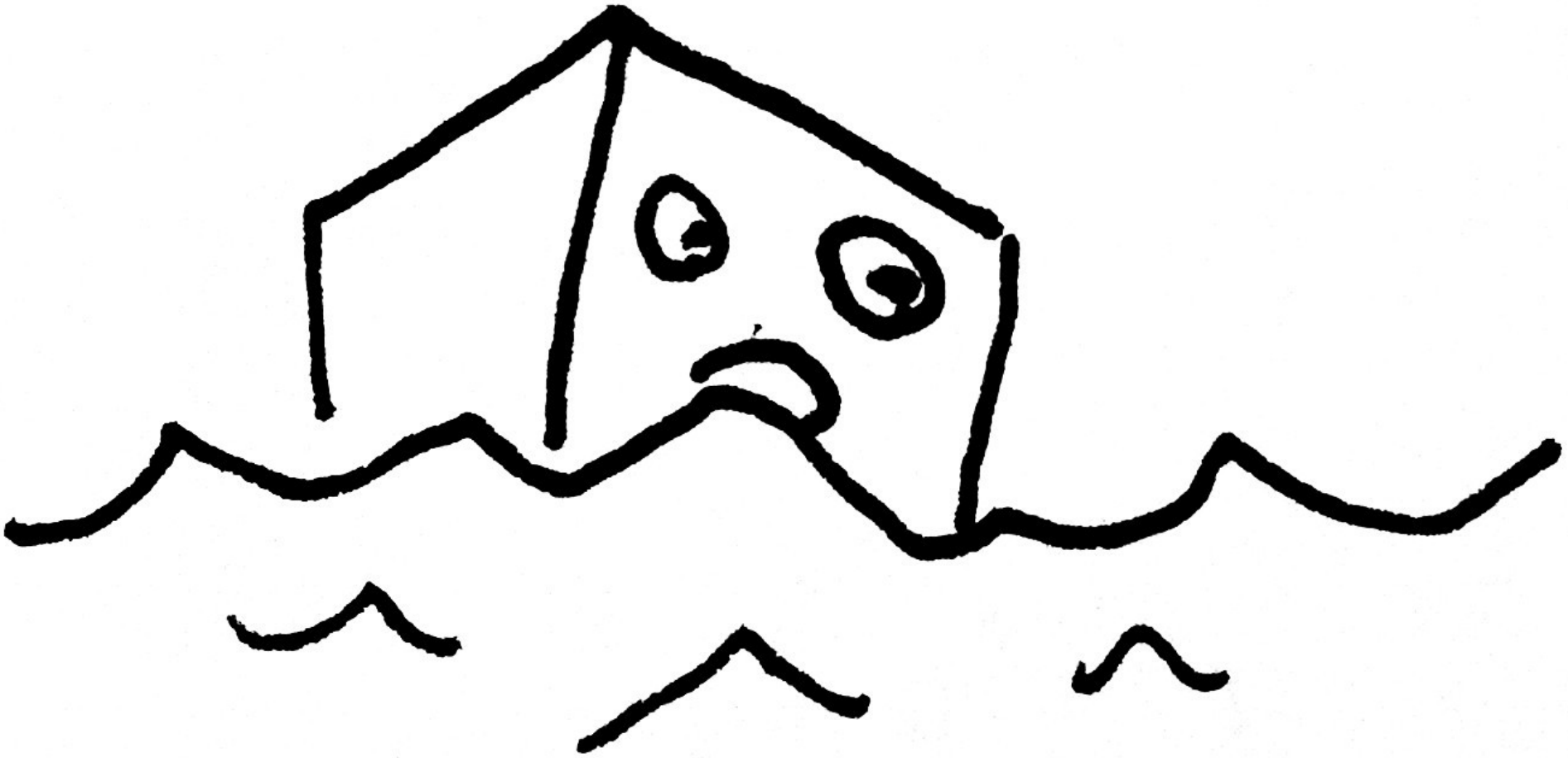
"did you just tell me to go fuck myself?"

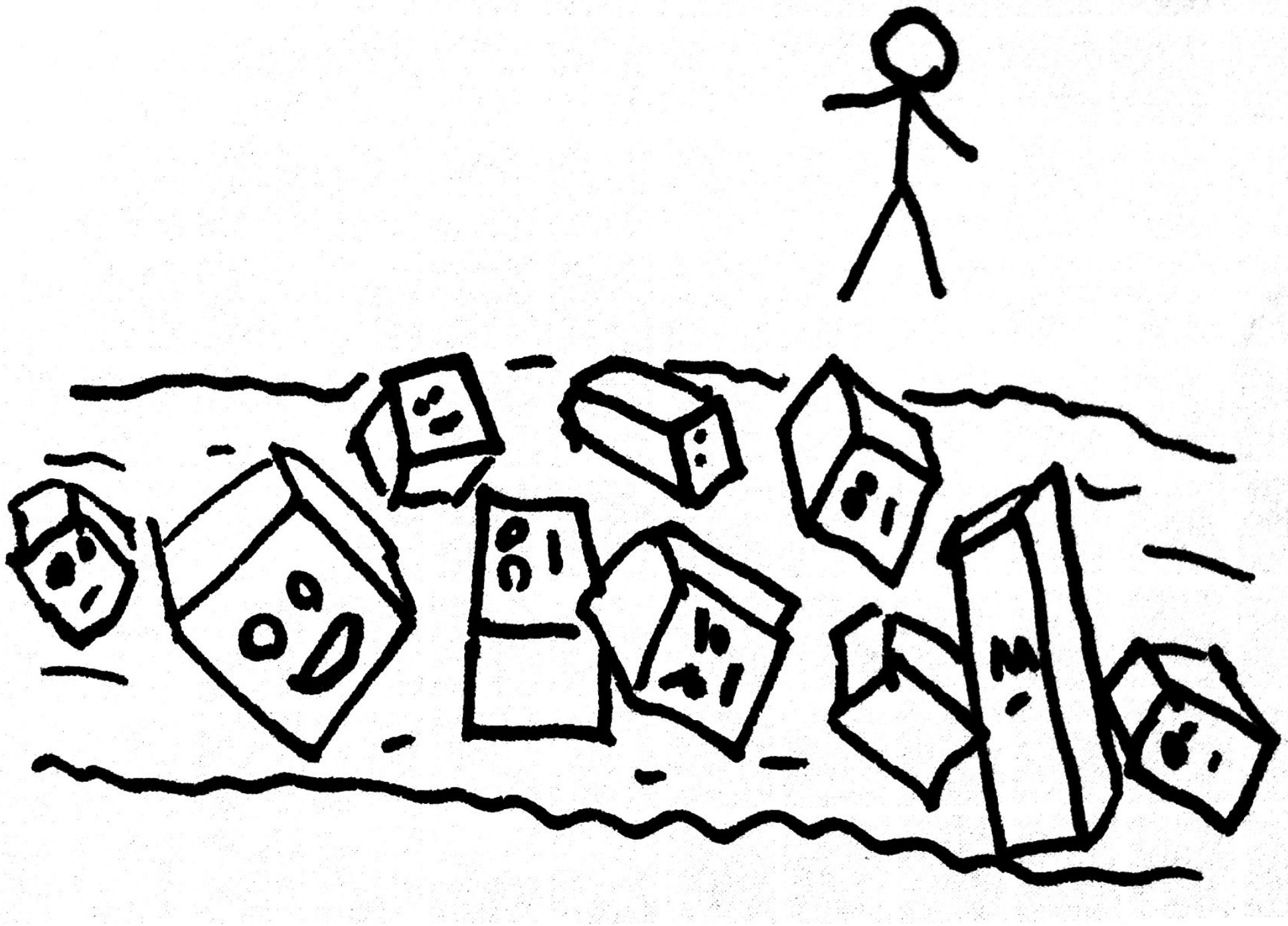


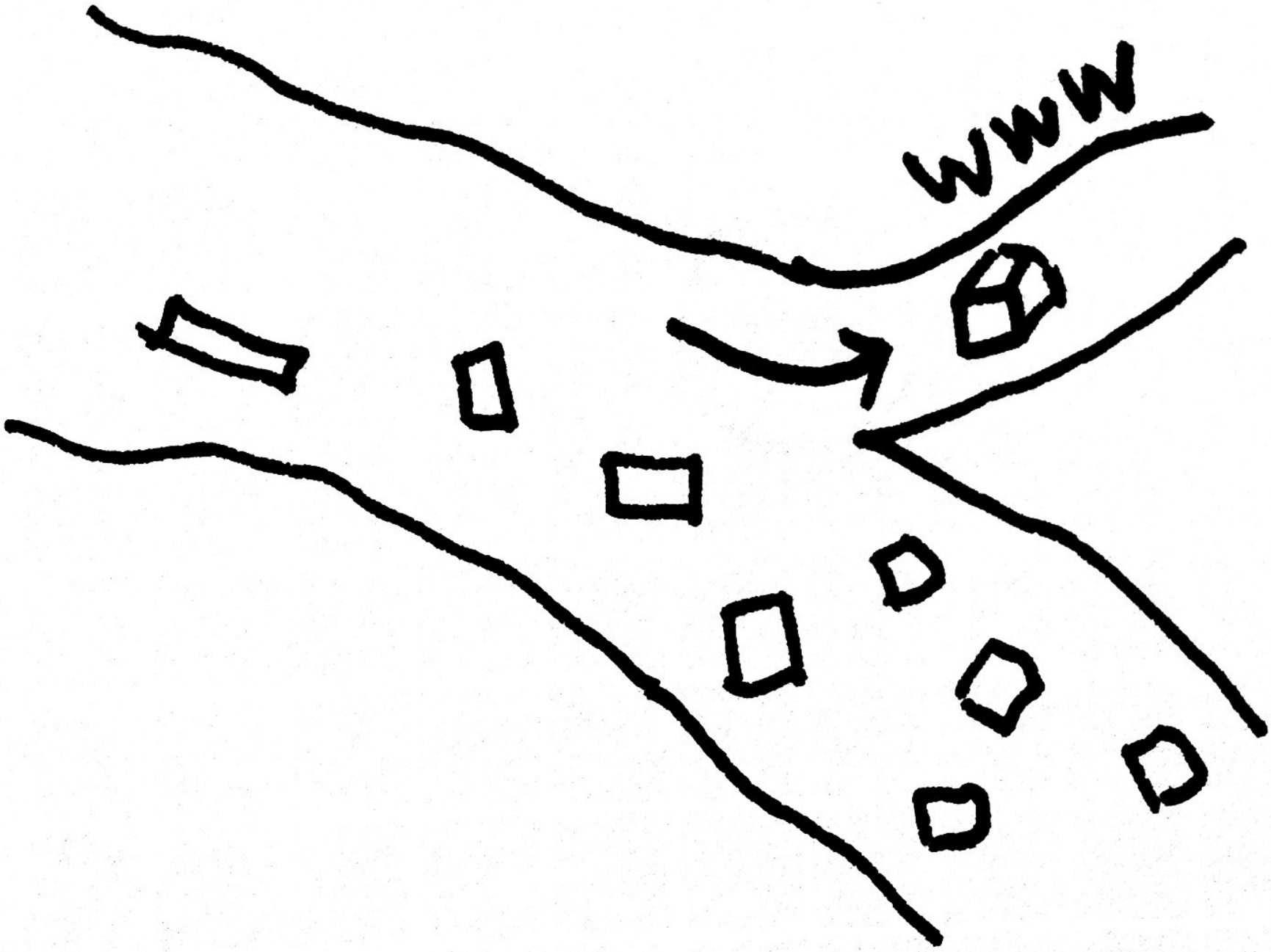
"i believe i did, bob."

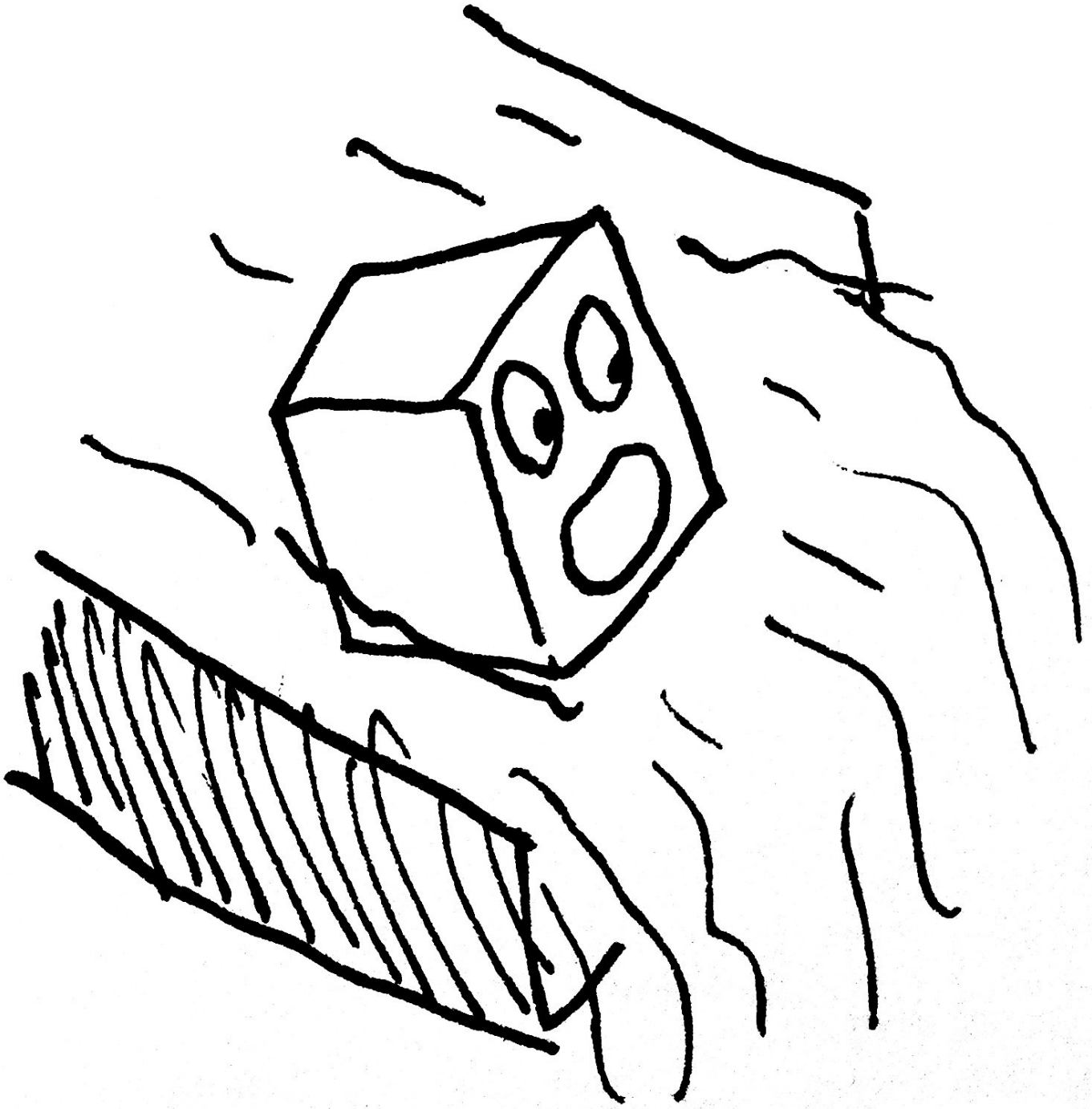

WWW

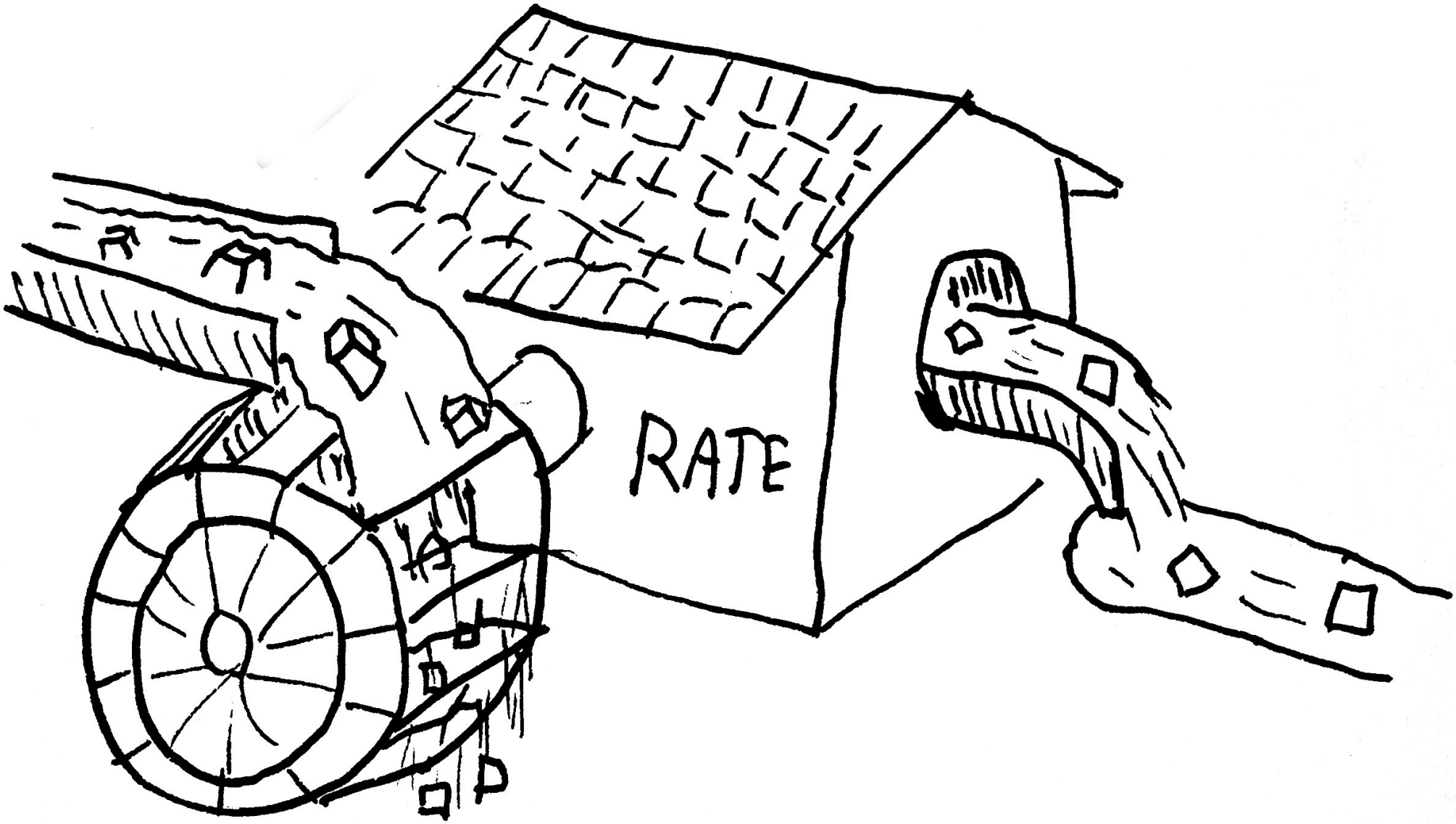


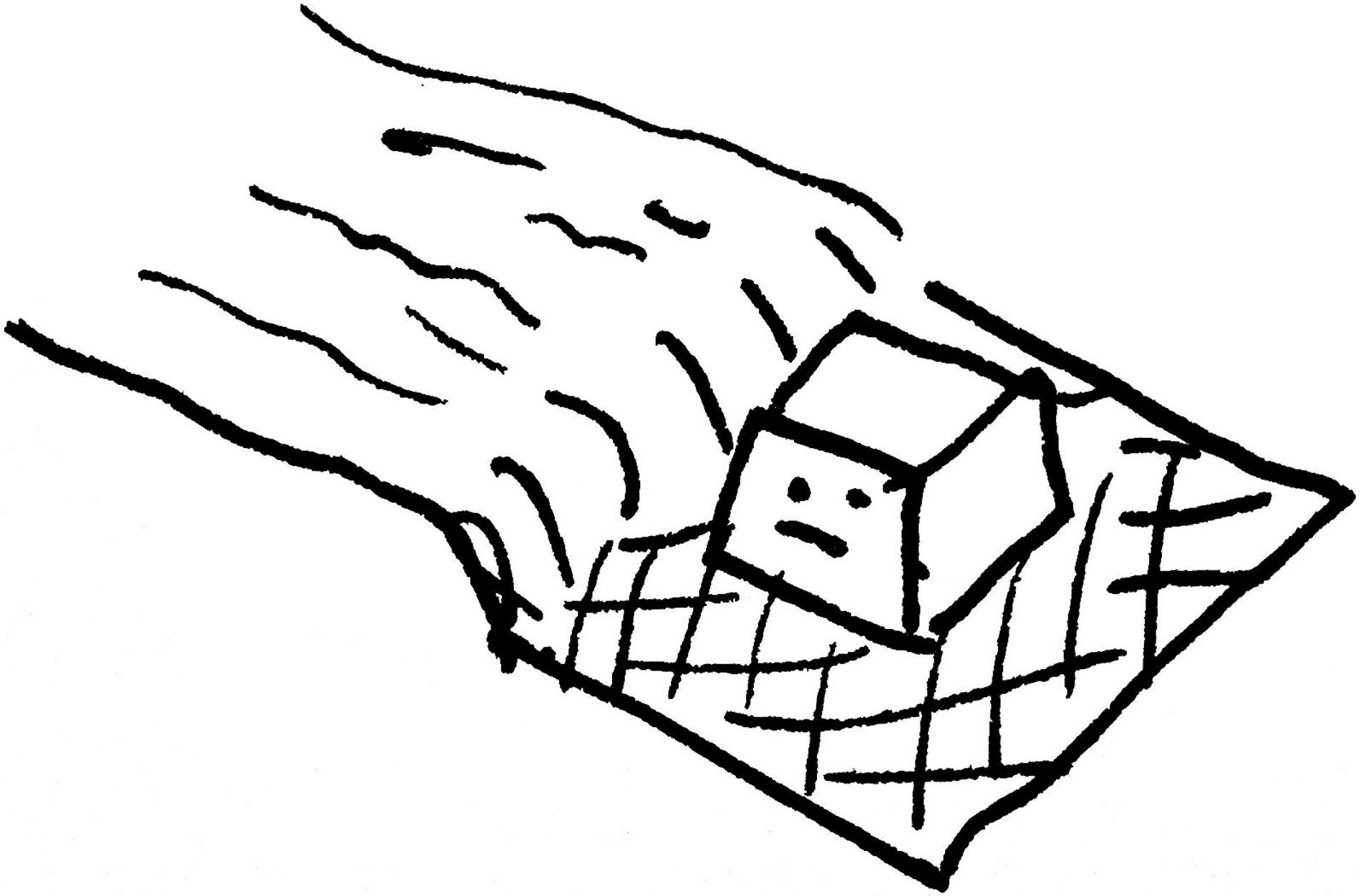


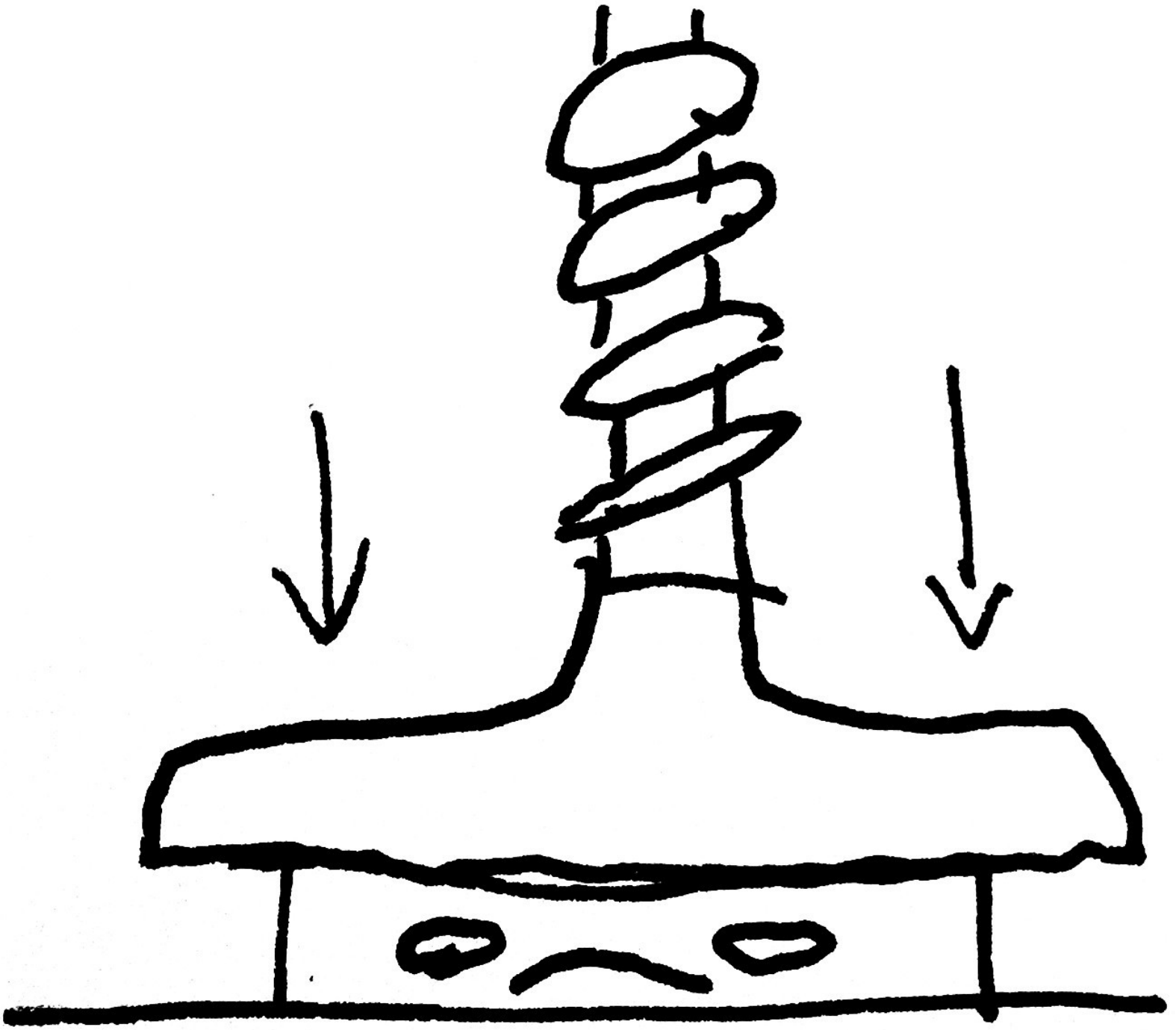


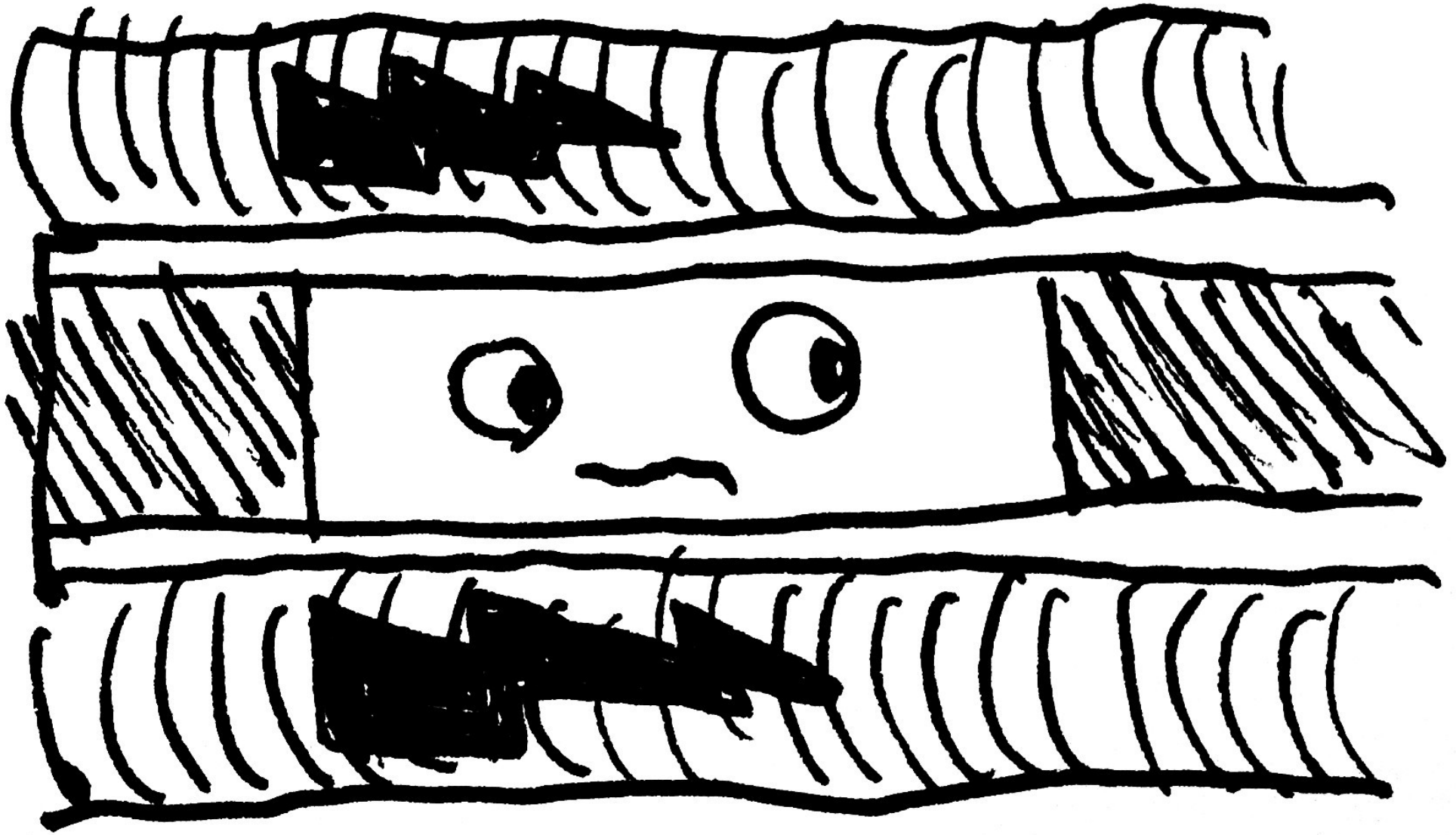

WWW

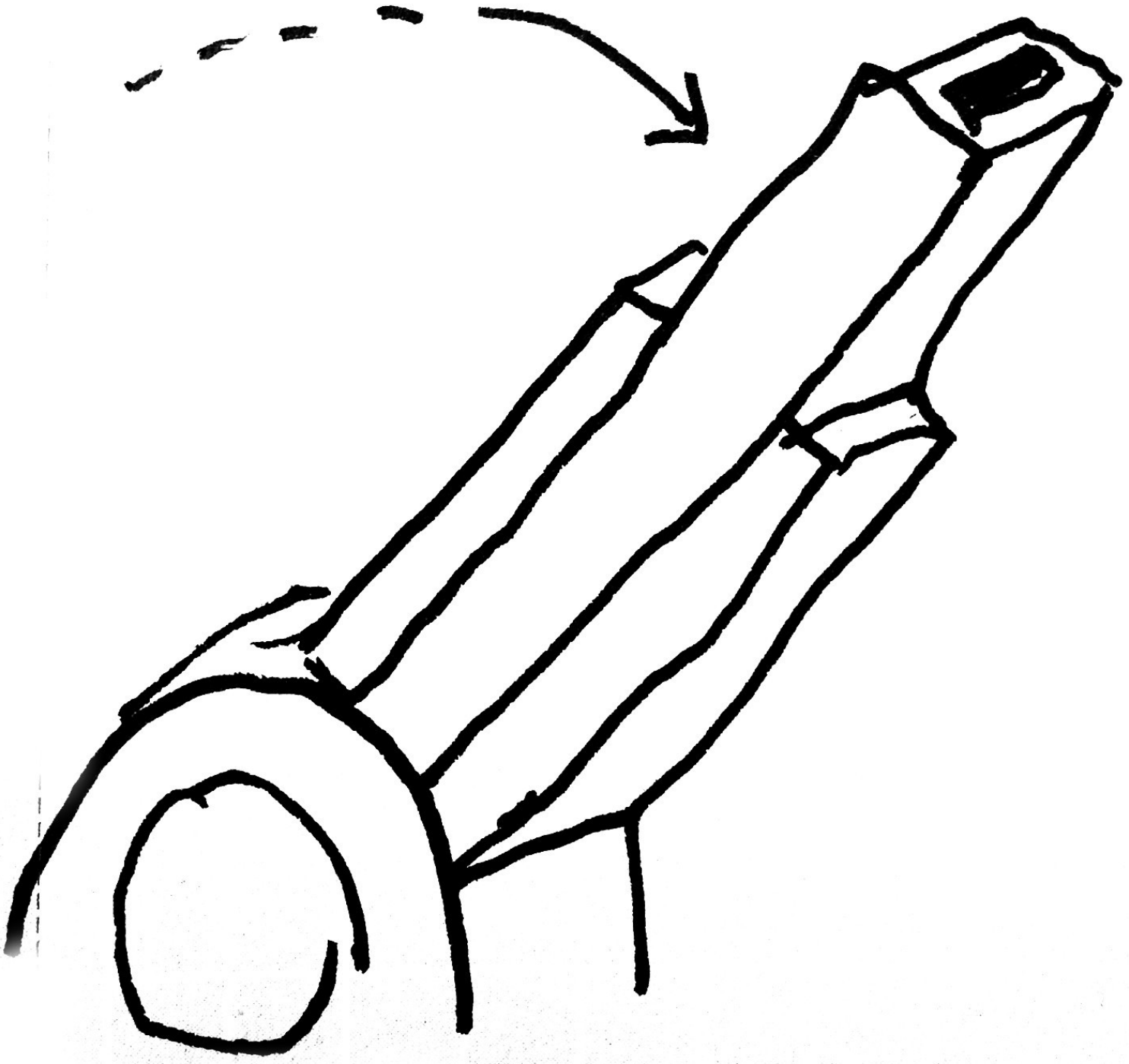


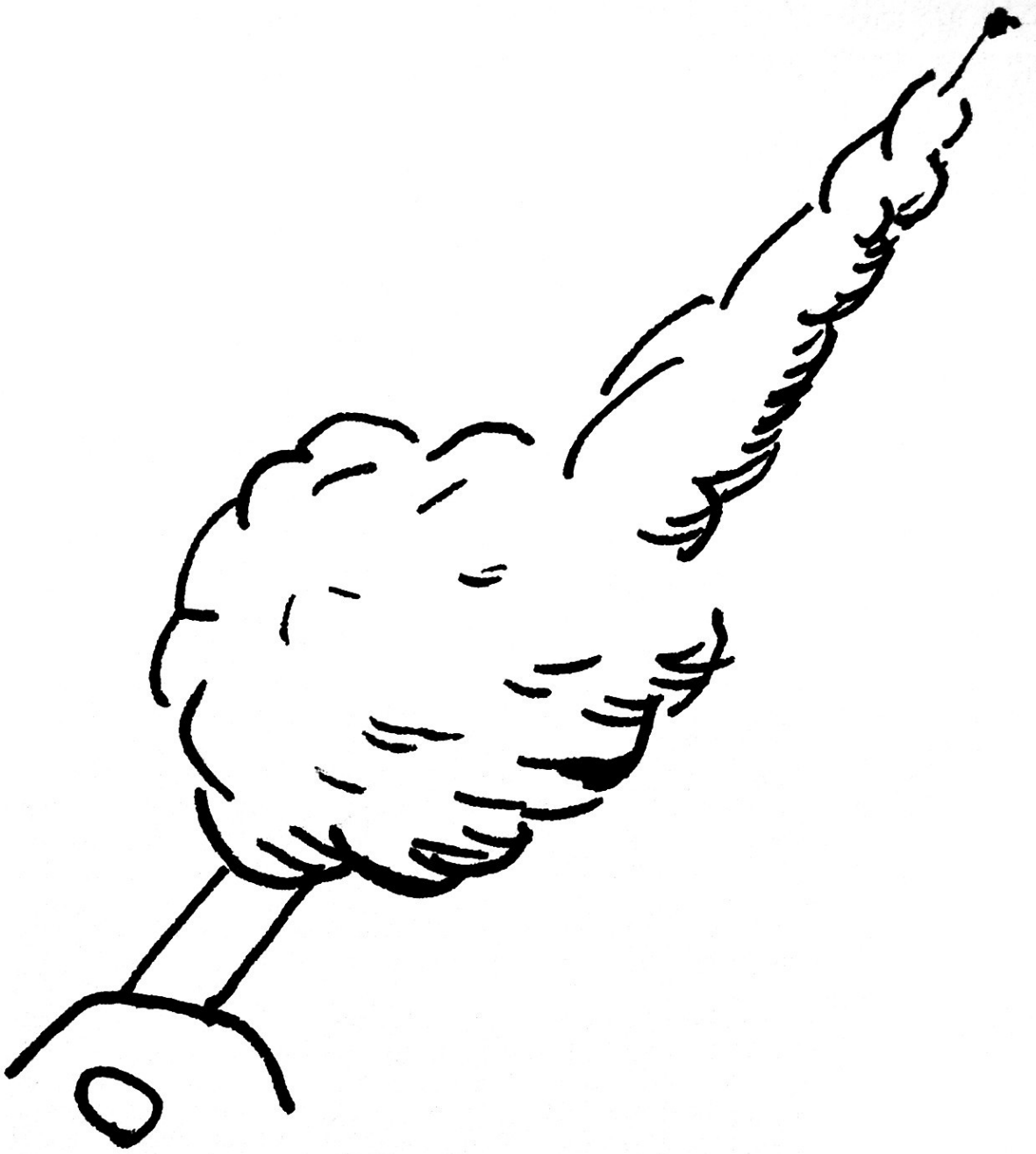

RATE

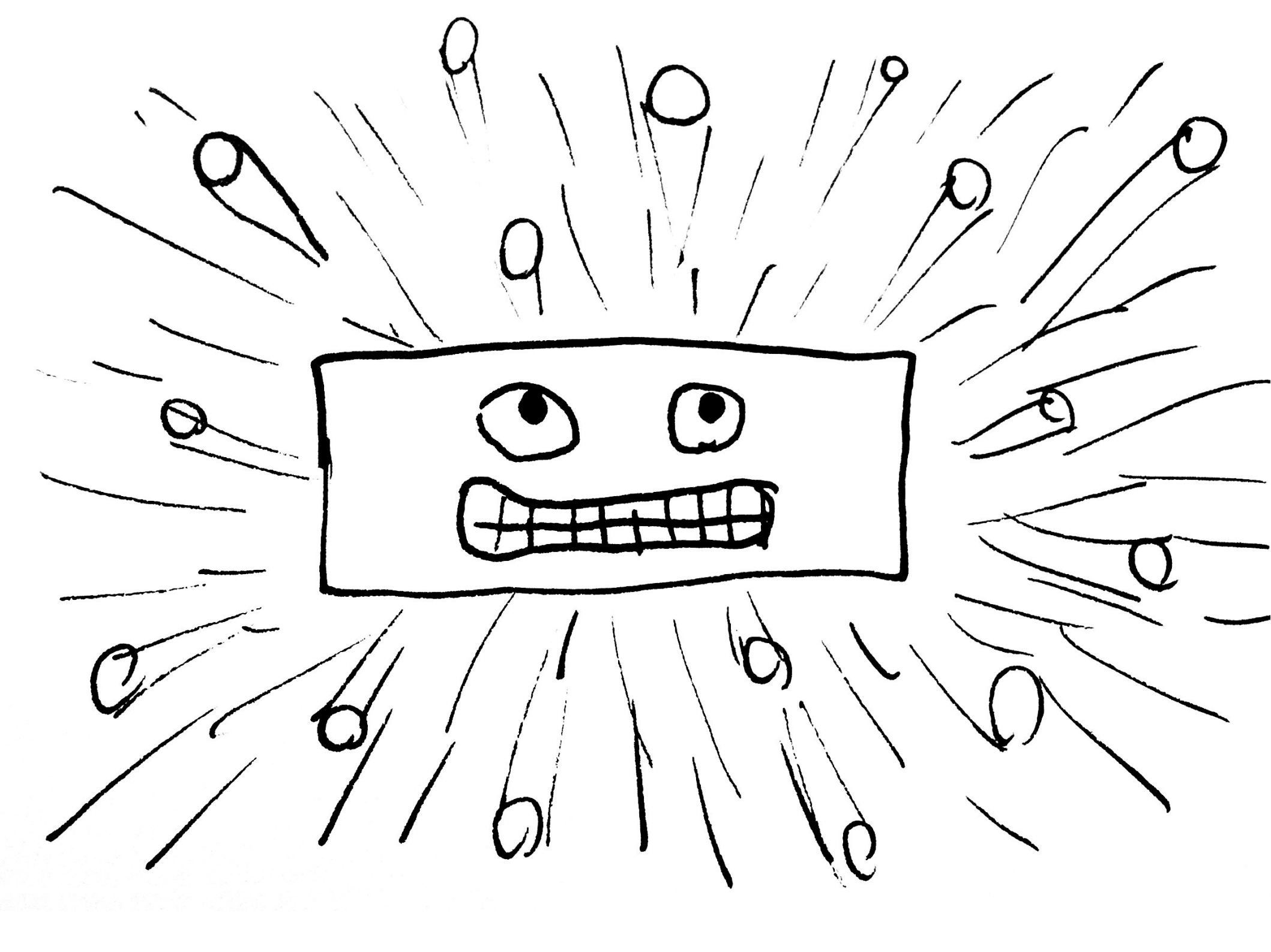


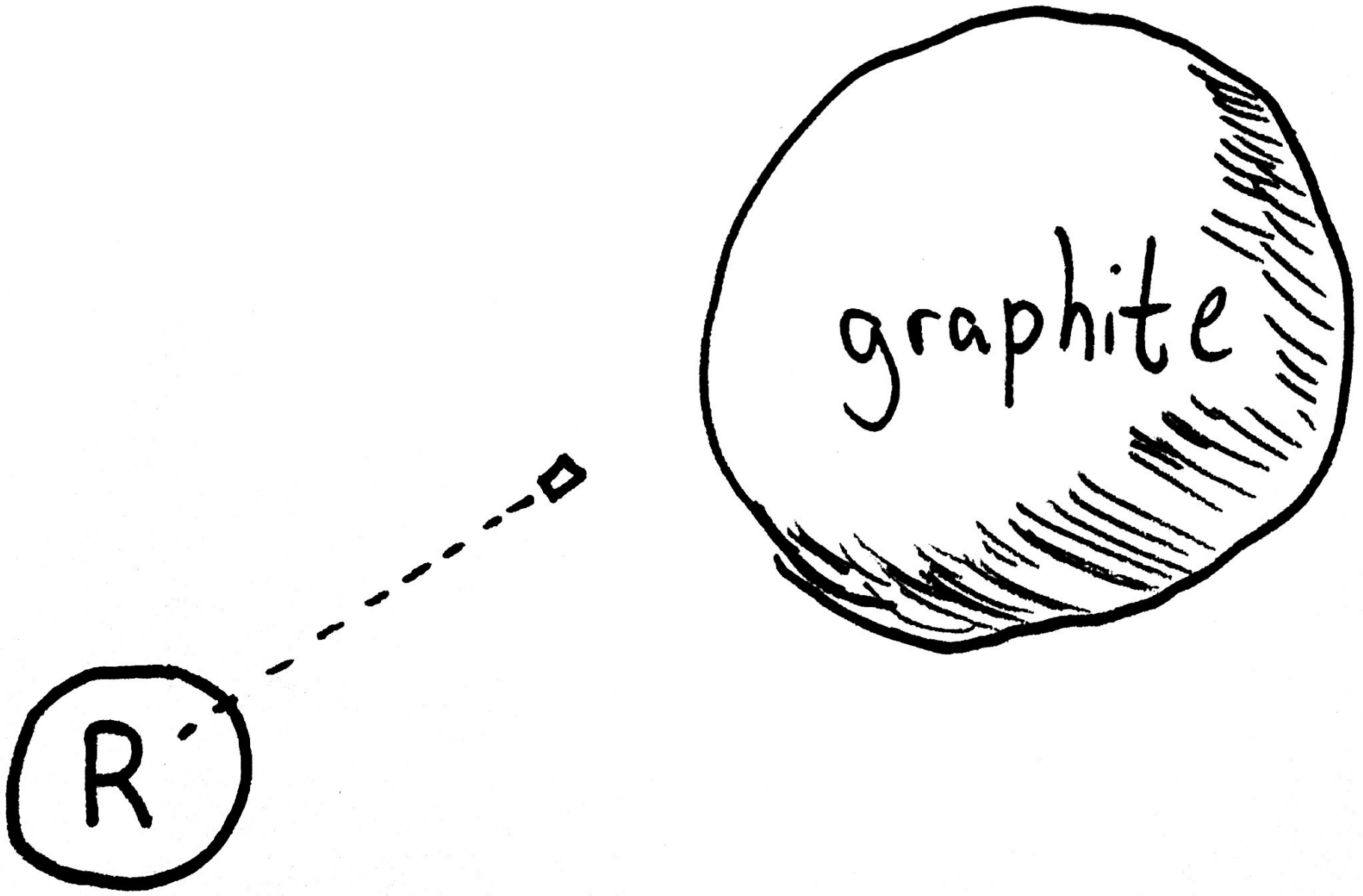


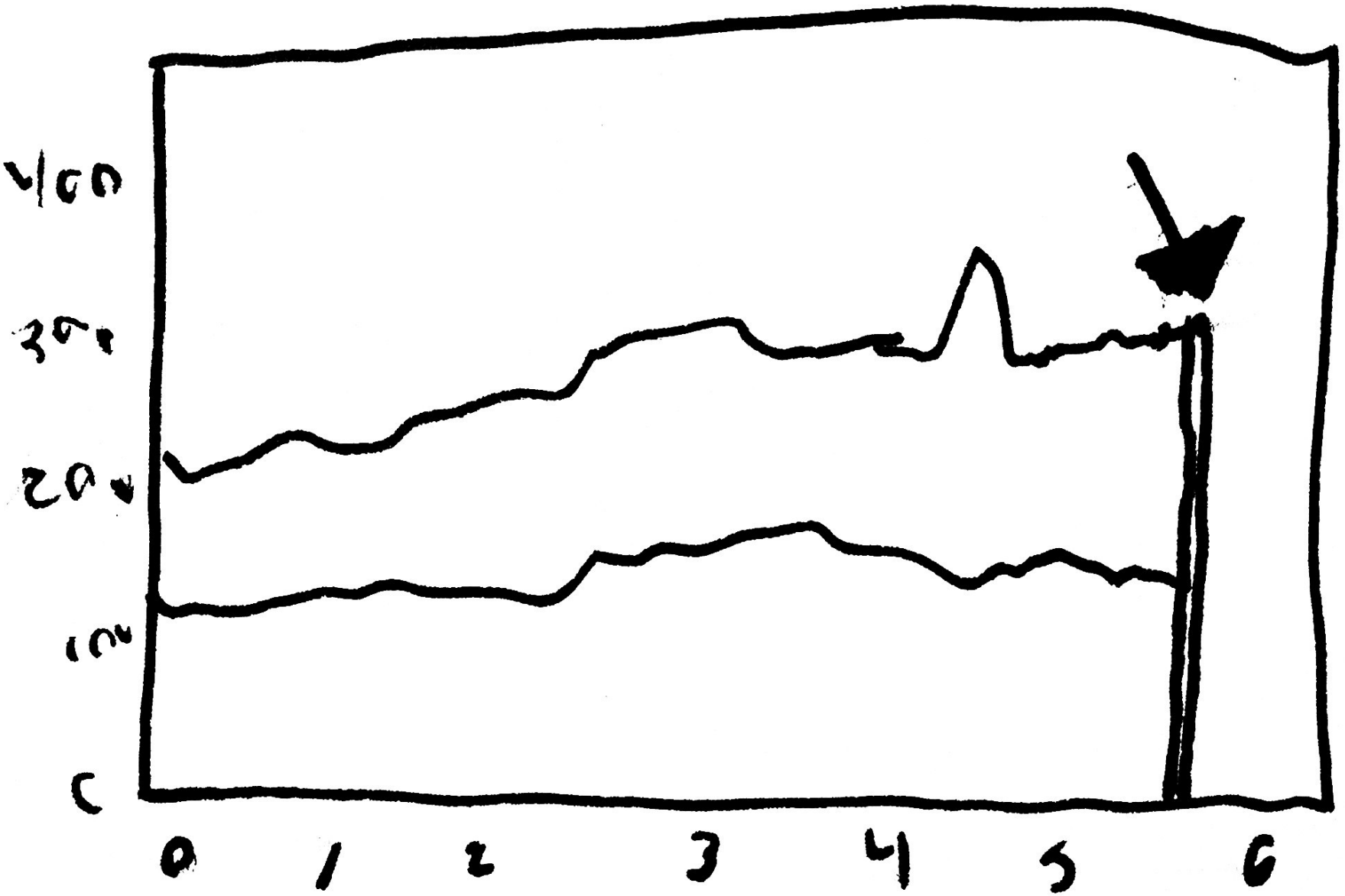


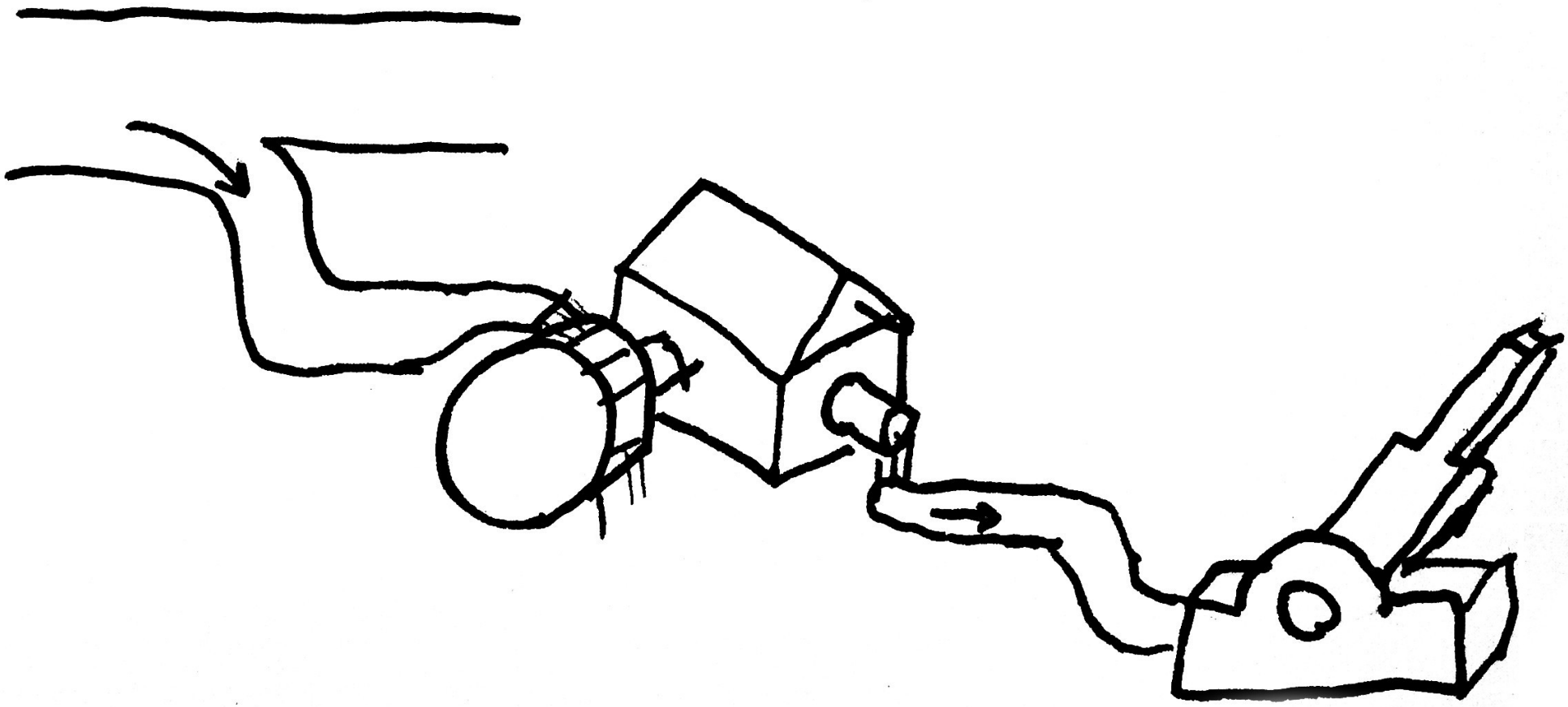







graphite
R·


40
30
20
10
0
0
1
2
3
4
5
6



streams

↳ where

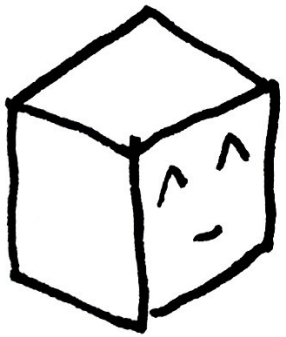
↳ rate

↳ graphite

```
(streams
  (where (service "www")
    (rate 5
      graphite)))
```

In Riemann,

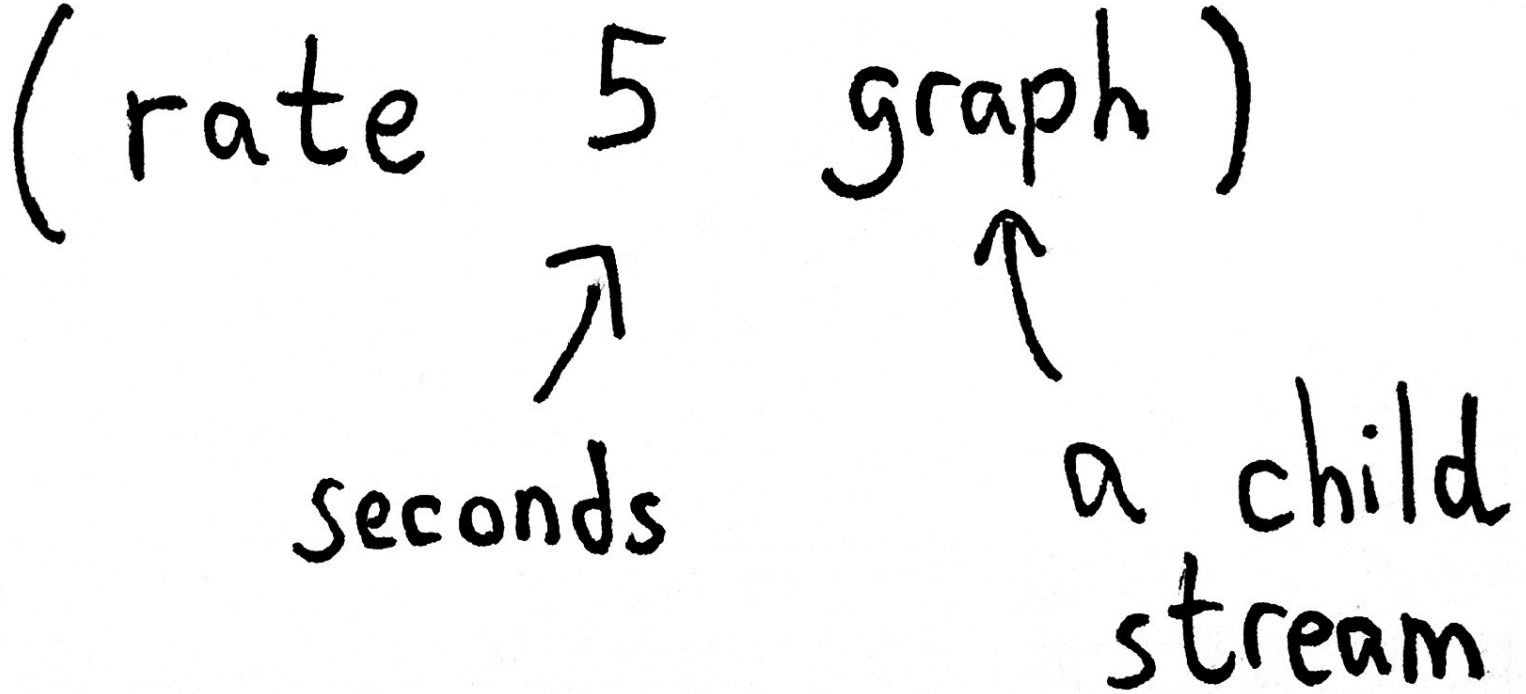
Events are just maps.

 = {:state "ok" ...}

Streams are just functions

(rate 5) →  = (fn [event]
... do stuff)

Stream composition is function composition!



Second derivatives?

(ddt 10 graph) ← 1st deriv.

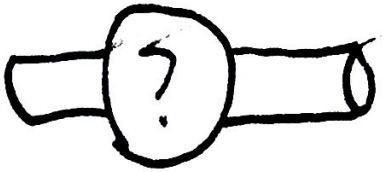
(ddt 10 (ddt 10 graph)) ← 2nd

A Whirlwind Tour of Riemann's Streams

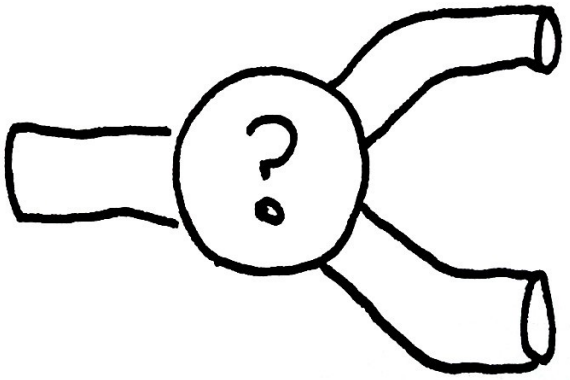
```
(where   (state "error")
    ...)
(where (and (not (service "special"))
            (state "exception"))
    ...)
```

```
(where   (and  (service #"queue .*")
               (< 5  metric  15))
     (email  "aphyr@aphyr.com")
     (pagerduty  :trigger))
```

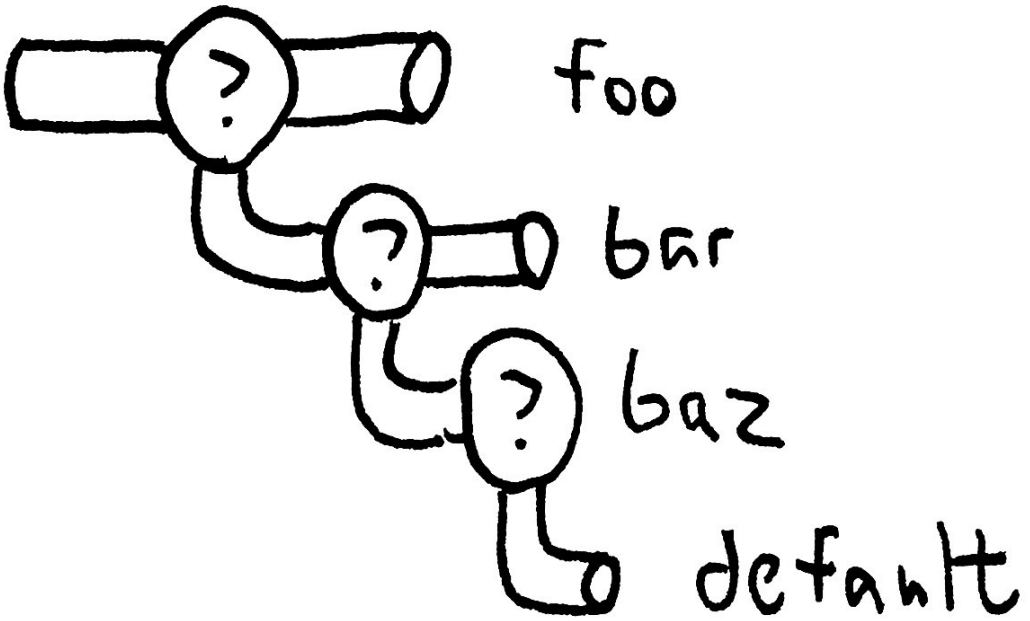
```
(where (and (service #"queue .*")
            (< 5 metric 15))
       (email "aphyr@aphyr.com")
       (pagerduty :trigger))
```



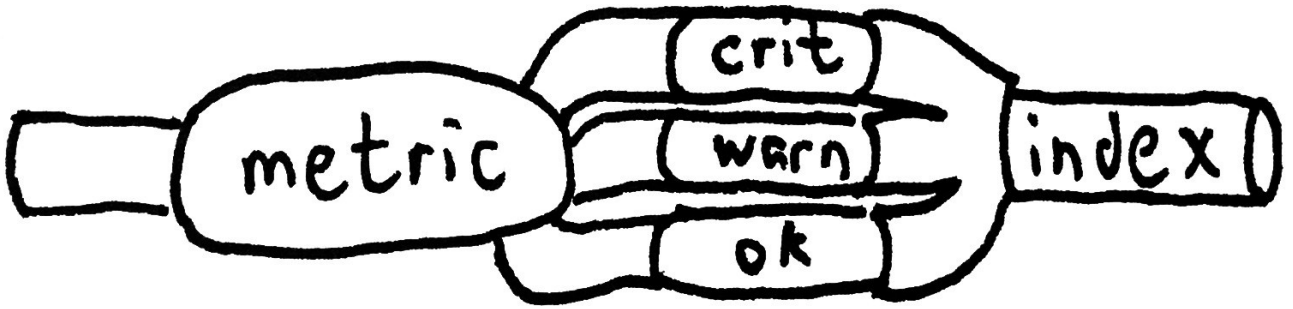

```
(where (some-fn event)
       foo
       (else
         bar))
```



```
(split
    (service "foo")      foo
    (state "bar")        bar
    (metric 3)           baz
    default-stream)
```

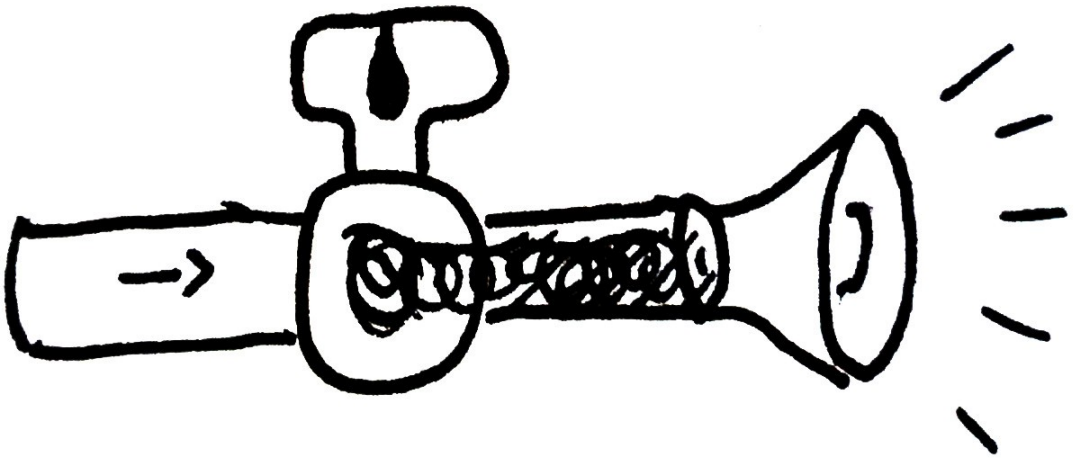


```
(splitp < metric
    .95 (with state "critical" index)
    .80 (with state "warning" index)
        (with state "ok" index))
```

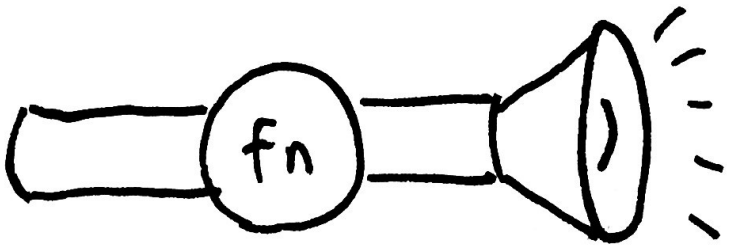


(with :service "http req rate"

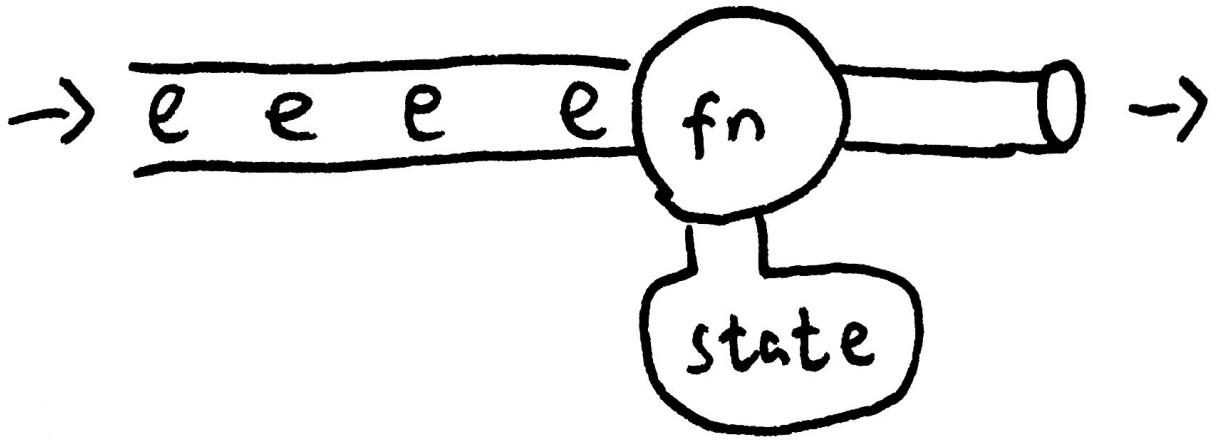
prn)



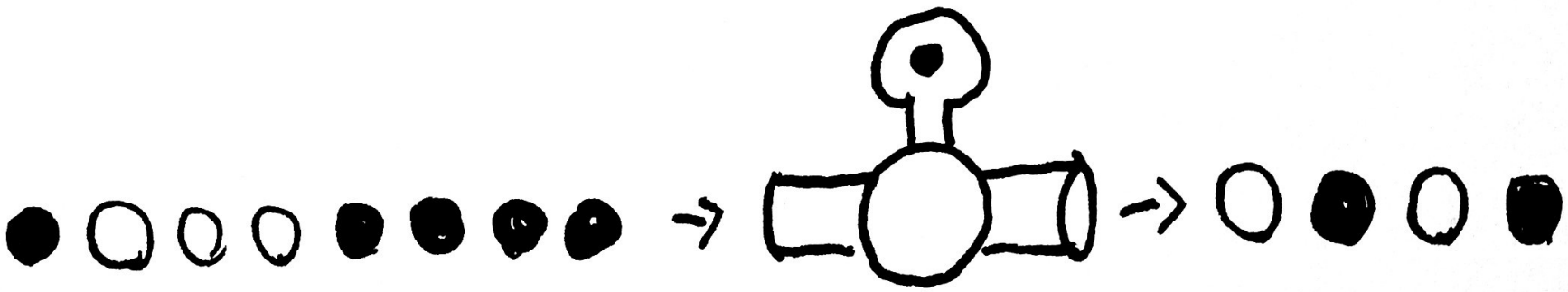
```
(smap  (fn [event]
         (assoc event :queue "my-jobs"))

  prn)
```



```
(sreduce (fn [acc event]
           ...)
         initial-value
         index)
```



(changed :state)



(counter ...)

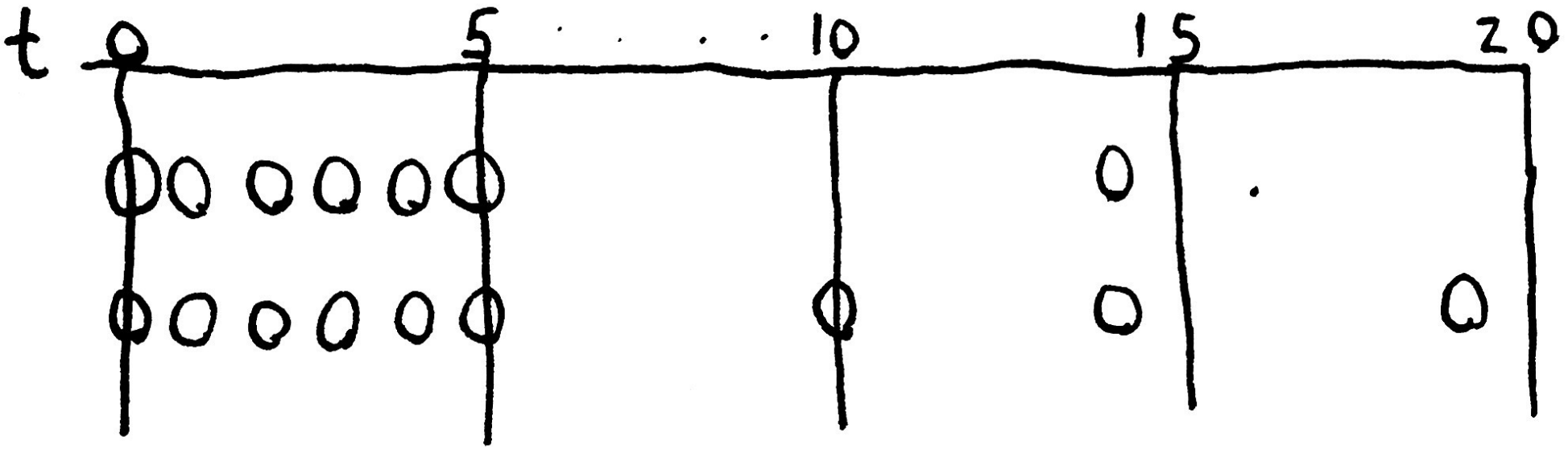
1 → 1
2 → 3
1 → 4
3 → 7

(ddt)

1 →
2 → 1
1 → -1
3 → 2

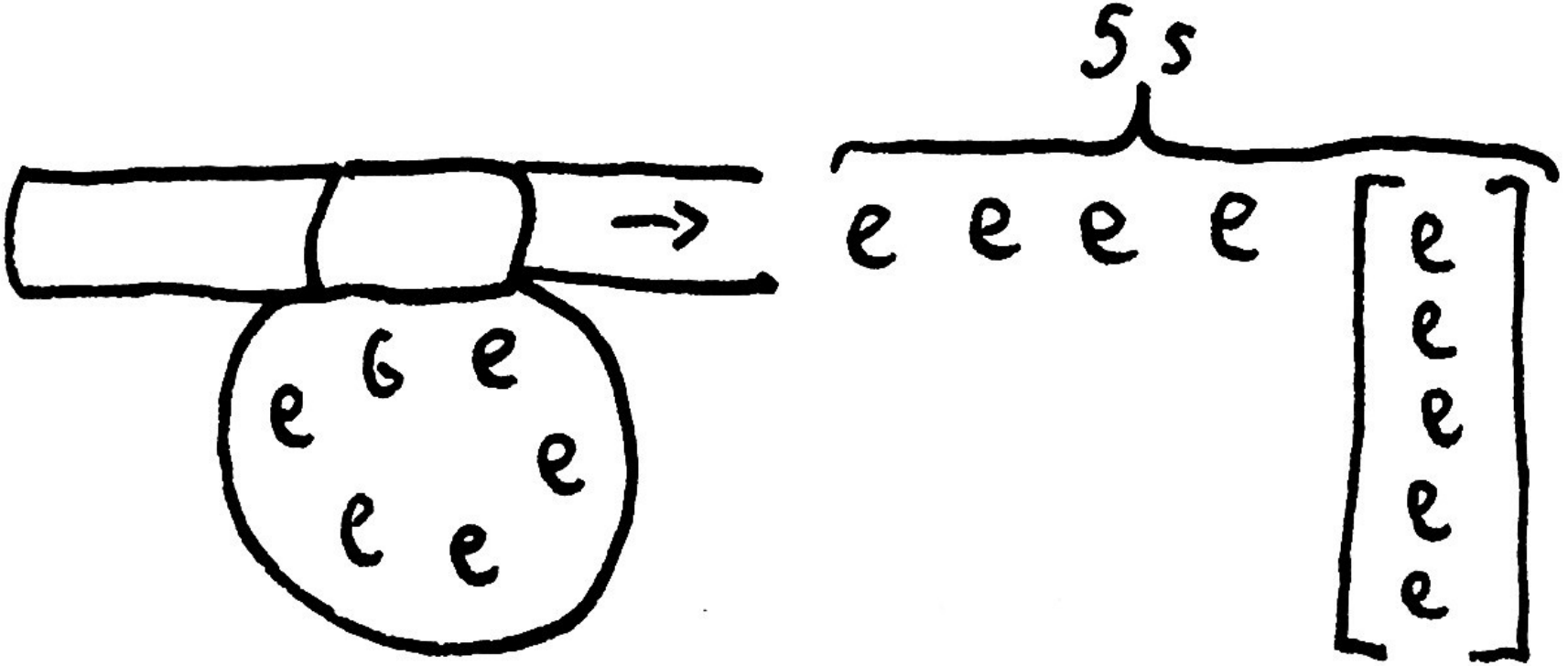
(ddt s...)

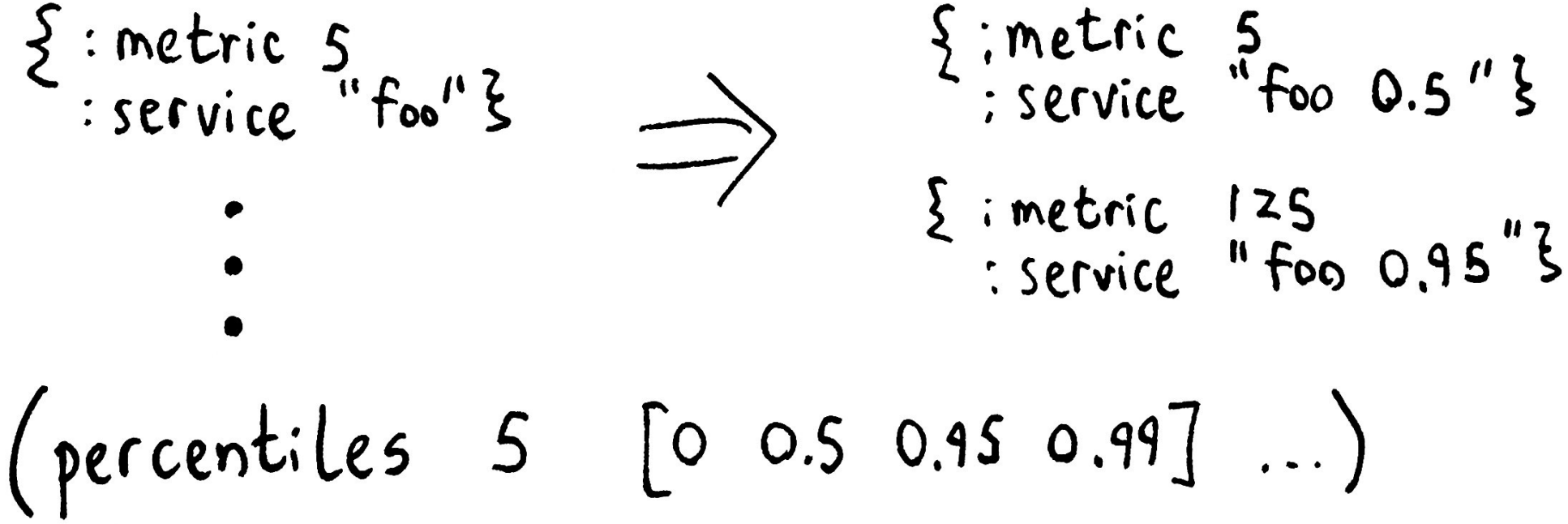
(interpolate-constant 5 ...)



(throttle 5 3600)

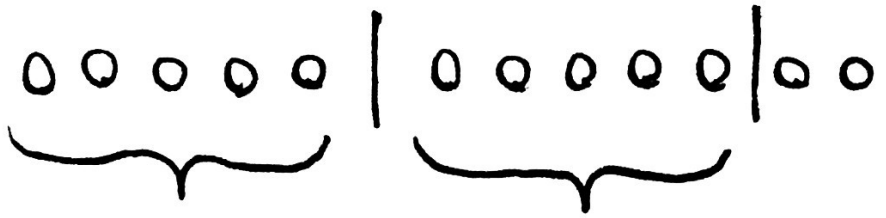
(rollup 5 3600 ...)



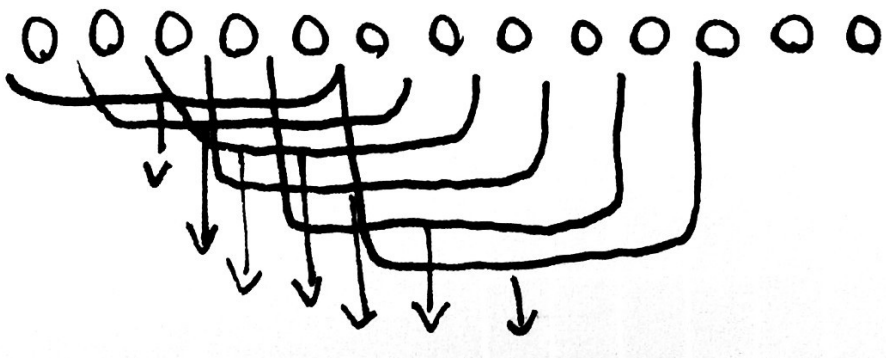

{:metric 5
:service "foo"}
{:metric 5
:service "foo 0.5"}
{:metric 125
:service "foo 0.95"}
(percentiles 5 [0 0.5 0.95 0.99] ...)

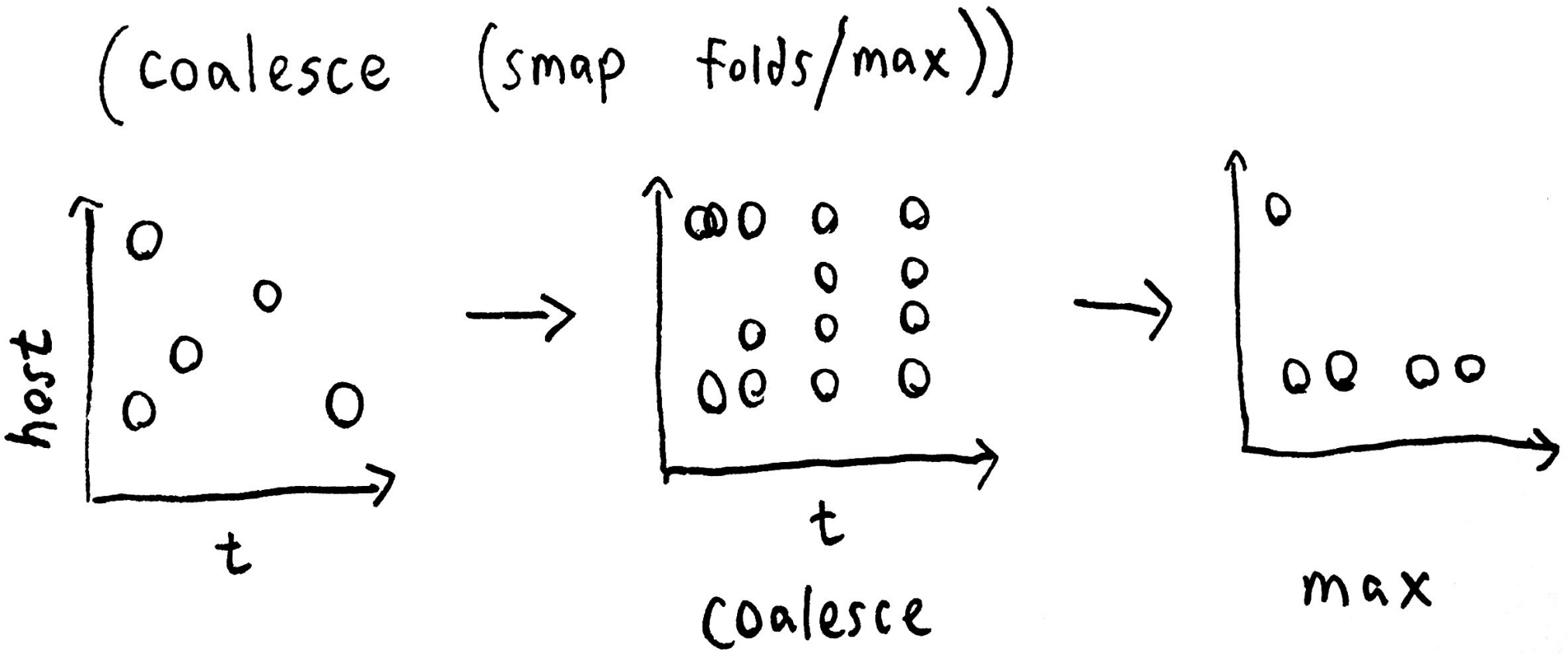
(top 10 :metric index)

(fixed-event-window 5)

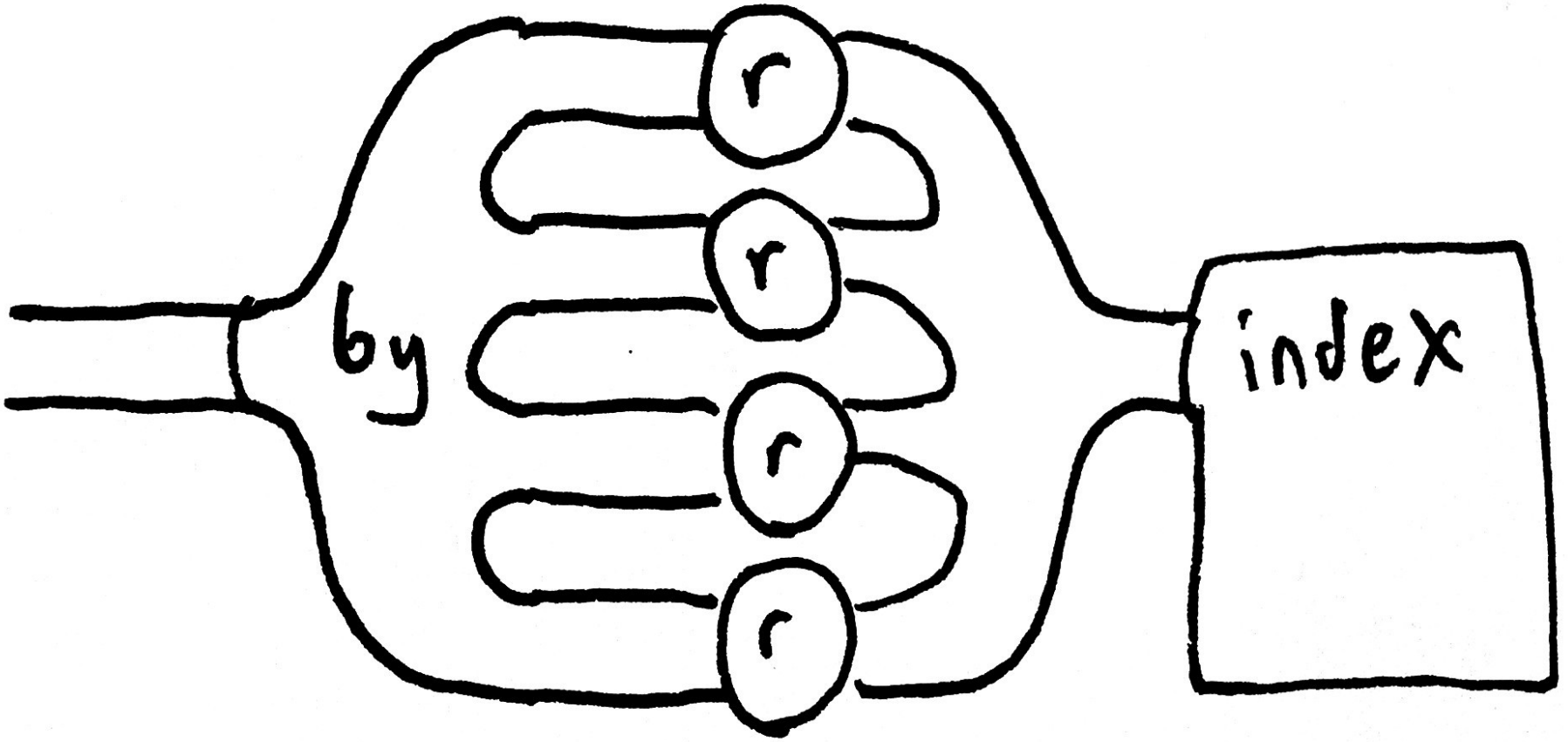


(moving-event-window 5)

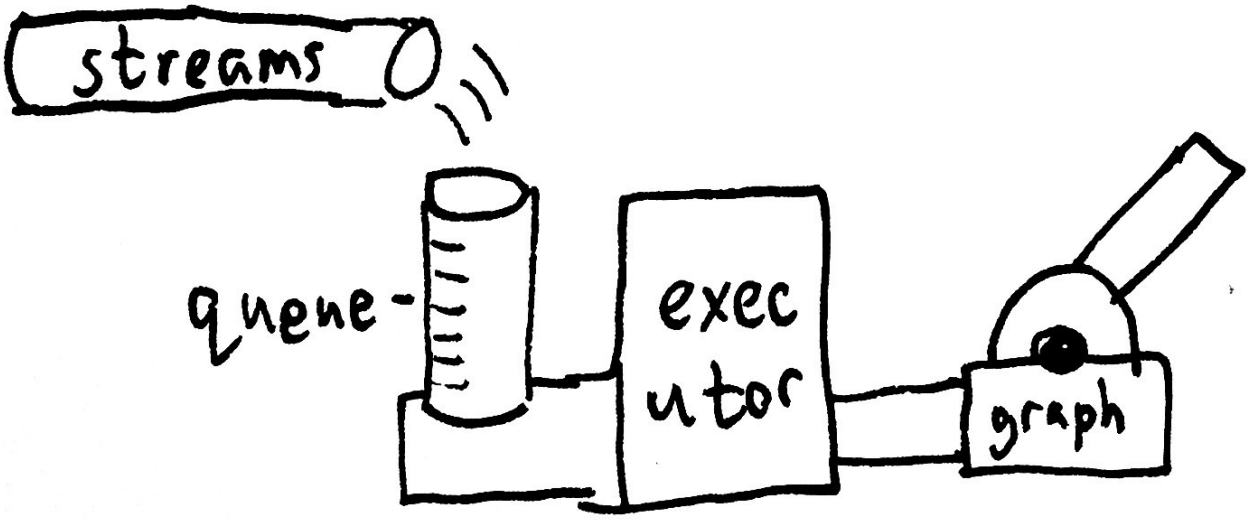


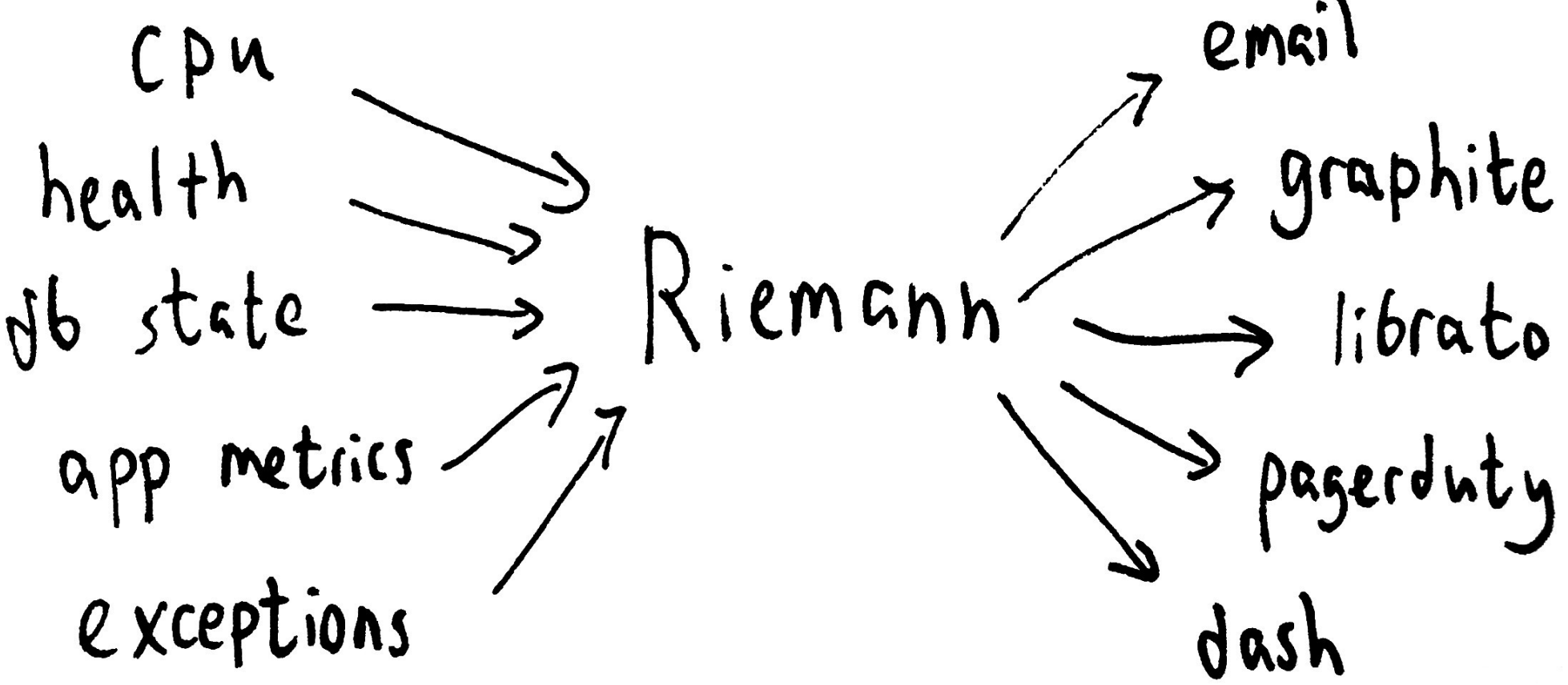

(coalesce (smap folds/max))
host
t
t
coalesce
max

(by :service (rate 5 index))

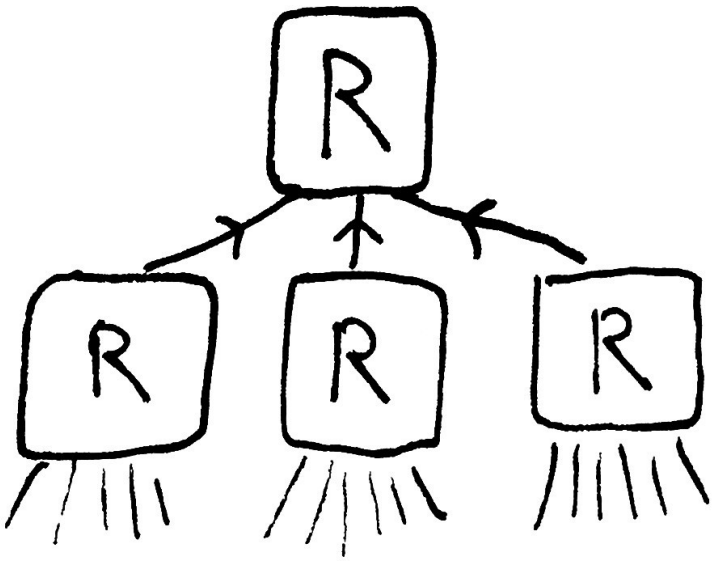


```
(let [graph (async-queue!
              (graphite {:host "1.2.3.4"}))]
  (streams
    (where (tagged "graphite")
           graph)))
```




cpu
health
jvm state
app metrics
exceptions
Riemann
email
graphite
librato
pagerduty
dash

```
(forward (tcp-client {:host "agg-node.tx"}))
```



Configuration as CODE

Use functions, variables, and namespaces to organize the PROBLEM

```
(include "foo.clj")

(ns foocorp.amqp)

(def alert-ops
  (rollup 10 3600
    (email "ops@foo.com")))
```

$ killall -sighup riemann

Hot config reloading!

Some things you just can't do in NAGIOS...

Eventually, in every monitoring system, you hit A WALL

Riemann aims to make the COMPLEX achievable...

Even if the SIMPLE takes a little more... work.

Riemann is a complex event stream processor for monitoring distributed systems at scale.

Designed to bridge the gap between disparate components.

In a low-latency, highly configurable way.

Things Riemann is not:

- A queue
- A database
- A historical store
- A dessert topping

Not designed for:

- Implicit distribution
- Clustering for HA
- Perfect reliability
- Multi-event transactions
- Long-term state

When your monitoring problems are complex,

Riemann gives you the *Flexibility* to *Adapt*.

Thank You!

@aphyr

riemann.io

Questions?