METRICS

METRICS EVERYWHERE

METRICS

METRICS EVERYWHERE

Make better **decisions**

by using **numbers**.

Coda Hale

@coda

github.com/codahale

The enterprise social network.

www.yammer.com

I write code.

But that's not actually my job.

code

code

business value

What the hell is

business value?

A new feature.

An improved existing feature.

Fewer bugs.

Not pissing our users off with a slow site.

Not pissing our users

off with a ~~slow~~ site.

ugly

Not pissing our users

off with a ~~slow~~ site.

~~ugly~~

pretty

Making future changes easier.

Adding a unit test before fixing that bug.

Business value is anything which makes people more likely to give us **money**.

We want to generate

more **business value**.

We need to make
better **decisions**
about our **code**.

Our **code** generates

business value

when it ***runs***.

Our **code** generates

business value

when it ***runs***,

not when we **write** it.

We need to know
what our **code** does
when it **runs**.

We can’t do this unless

we **measure** it.

Why **measure** it?

map ≠ territory

map ≠ city

of of

San San

Francisco Francisco

the ≠ the

way	way
we	it
talk	is

the ≠ the

thing	thing
we	in
think of	itself

perception ≠ reality

Saturday, April 9, 2011

We have a

mental model

of what our **code** does.

It's a **mental model.**

It's not the **code.**

It is often **wrong**.

Confusion.

“This code can’t possibly work.”

(It works.)

Saturday, April 9, 2011

“This code can’t possibly fail.”

(It fails.)

Saturday, April 9, 2011

Which is faster?

Which is faster?

```
items.sort_by { |i| i.name }
```

Which is faster?

```
items.sort_by { |i| i.name }
```

```
items.sort { |a, b| a.name <=> b.name }
```

We don’t know.

```ruby
def sort_by(&blk)
  sleep(100) # FIXME: I AM POISON
  super(&blk)
end
```

We don't know.

```ruby
def sort_by(&blk)
  sleep(100) # FIXME: I AM POISON
  super(&blk)
end
```

We don't know.

```ruby
def sort(&blk)
  # TODO: make not explode
  raise Exception.new("Haw haw!")
end
```

We **can't** know until we **measure** it.

This affects how we make **decisions**.

“Our application is slow.
This page takes 500ms.
Fix it.”

Find the bottleneck!

Find the bottleneck!

SQL Query

Find the bottleneck!

SQL Query

Template Rendering

Find the bottleneck!

SQL Query

Template Rendering

Session Storage

We don’t know.

Find The Bottleneck *2.0*!

SQL Query

Template Rendering

Session Storage

Find The Bottleneck *2.0*!

SQL Query 53ms

Template Rendering

Session Storage

Find The Bottleneck *2.0*!

SQL Query 53ms

Template Rendering 1ms

Session Storage

Find The Bottleneck *2.0*!

SQL Query	53ms
Template Rendering	1ms
Session Storage	315ms

Find The Bottleneck *2.0*!

SQL Query	53ms
Template Rendering	1ms
Session Storage	315ms

Confusion.

We made a better

decision.

We improve our **mental model** by **measuring** what our **code** does.

map ≠ territory

map → territory

We use our

mental model

to **decide** what to **do**.

A better
mental model
makes us better at
deciding what to **do**.

A better
mental model
makes us better at
generating
business value.

Measuring makes your **decisions** better.

But only if we're

measuring

the **right thing**.

We need to **measure** our **code** where it *matters*.

In the **wild.**

Generating

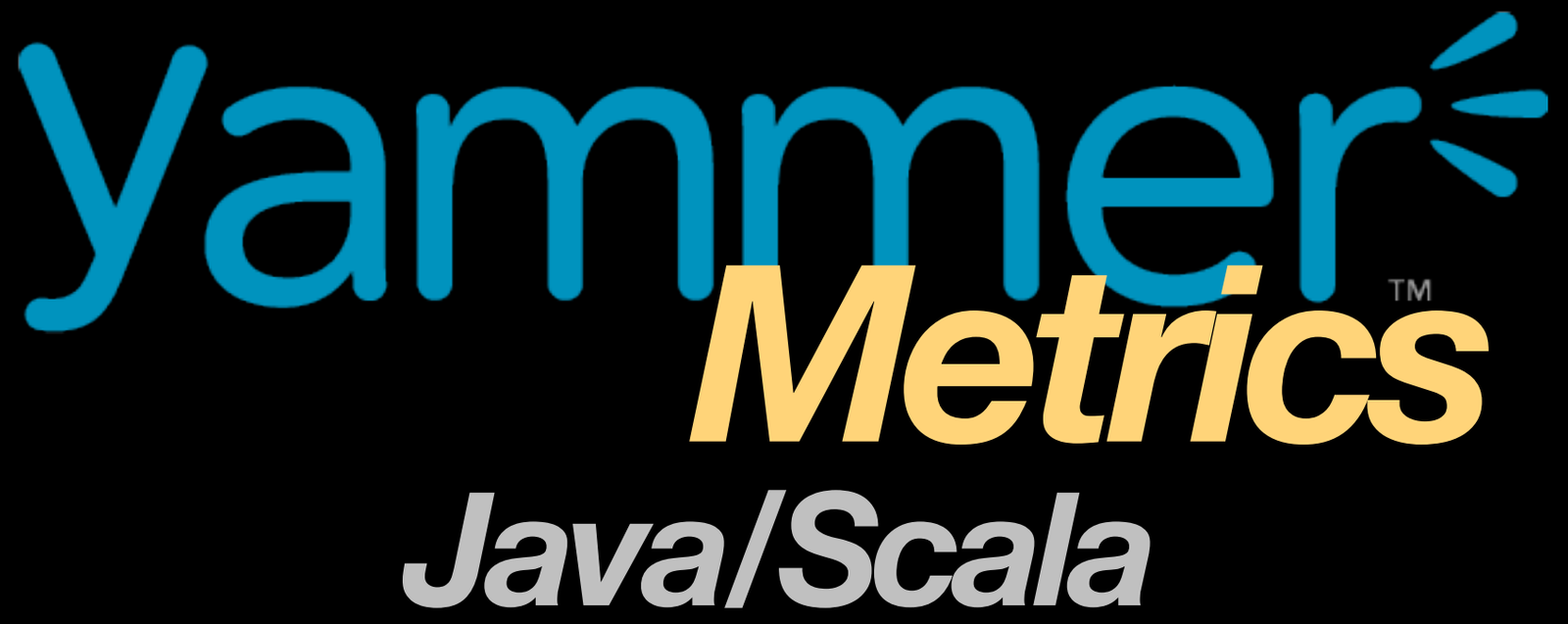
business value.

PRODUCTION

Continuously measuring code in production.

yammer™

Metrics


yammer
Metrics™
Java/Scala

Java/Scala

github.com/codahale/metrics

Gauges
Counters
Meters
Histograms
Timers

Each metric is associated with a **class** and has a **name**.

An autocomplete service for city names.

An autocomplete service for city names.

```
> GET /complete?q=San%20Fra
```

An autocomplete service for city names.

```
> GET /complete?q=San%20Fra
<
< HTTP/1.1 200 RAD
<
< ["San Francisco"]
```

What does this **code**
do that affects its
business value?

And how can we **measure** that?

Gauges
Counters
Meters
Histograms
Timers

Gauges

Counters

Meters

Histograms

Timers

Gauge

The instantaneous value of something.

# of cities

```
metrics.gauge("cities") { cities.size }
```

```
metrics.gauge("cities") { cities.size }
```

```
metrics.gauge("cities") { cities.size }
```

“The service has 589 cities registered.”

Gauges

Counters

Meters

Histograms

Timers

Gauges

Counters

Meters

Histograms

Timers

Counter

An incrementing and decrementing value.

# of open connections

```
val counter = metrics.counter("connections")

counter.inc()

counter.dec()
```

```
val counter = metrics.counter("connections")

counter.inc()

counter.dec()
```

```
val counter = metrics.counter("connections")

counter.inc()

counter.dec()
```

```
val counter = metrics.counter("connections")

counter.inc()

counter.dec()
```

“There are 594 active sessions on that server.”

Gauges

Counters

Meters

Histograms

Timers

Gauges
Counters
Meters
Histograms
Timers

Meter

The average rate of events over a period of time.

# of requests/sec

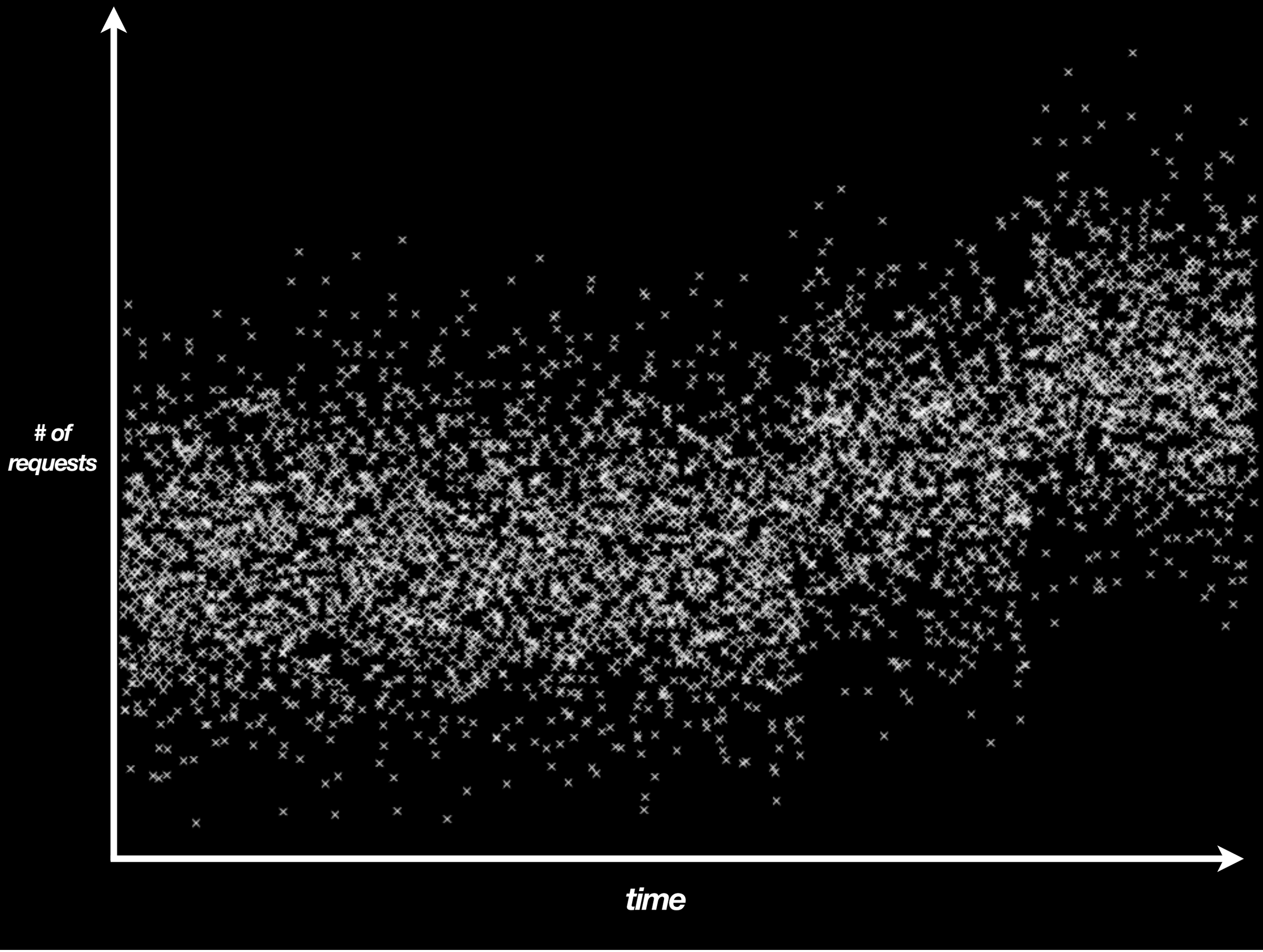
```
val meter = metrics.meter("requests",
                          SECONDS)
meter.mark()
```

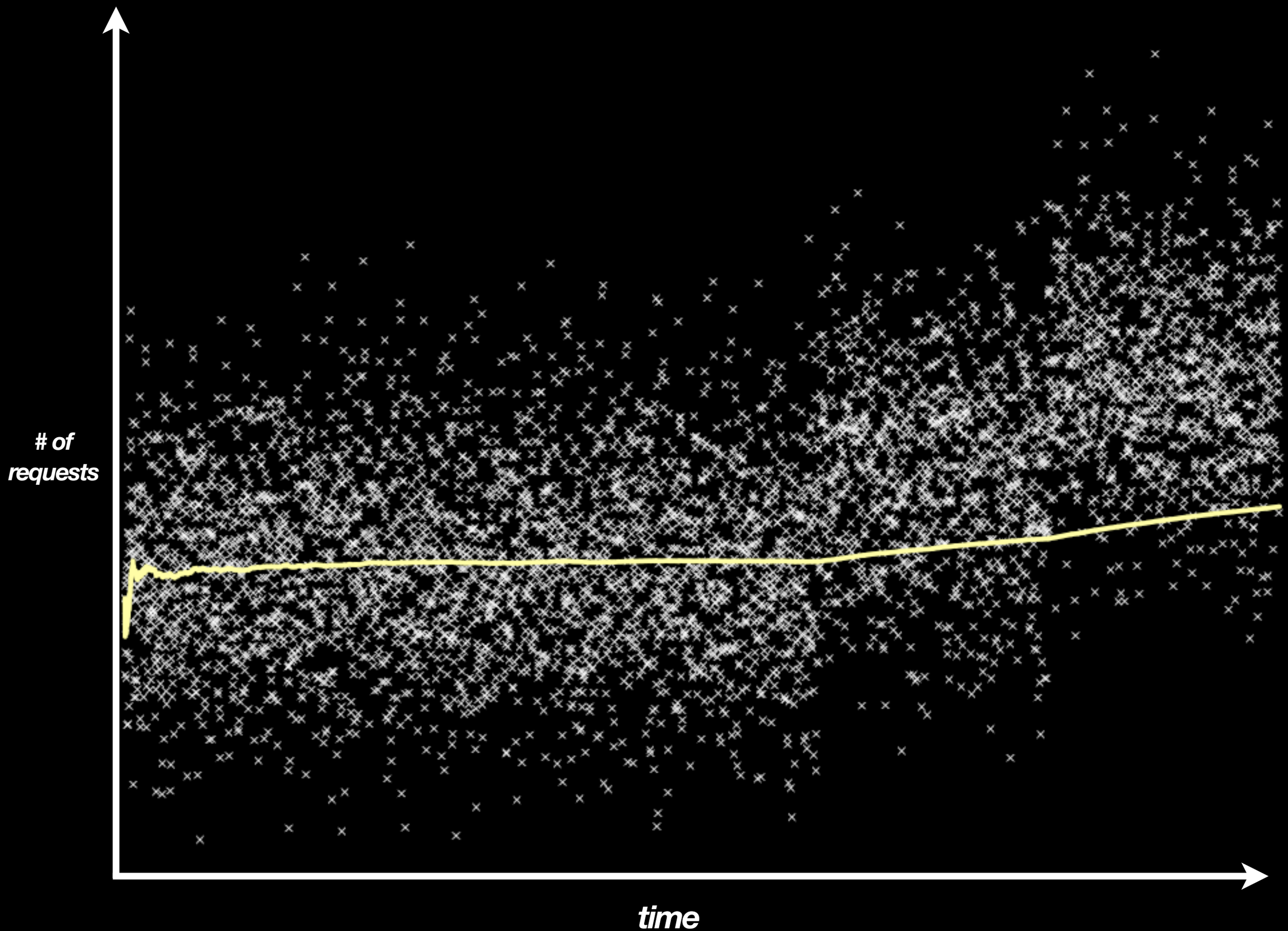
```
val meter = metrics.meter("requests",
                          SECONDS)
meter.mark()
```

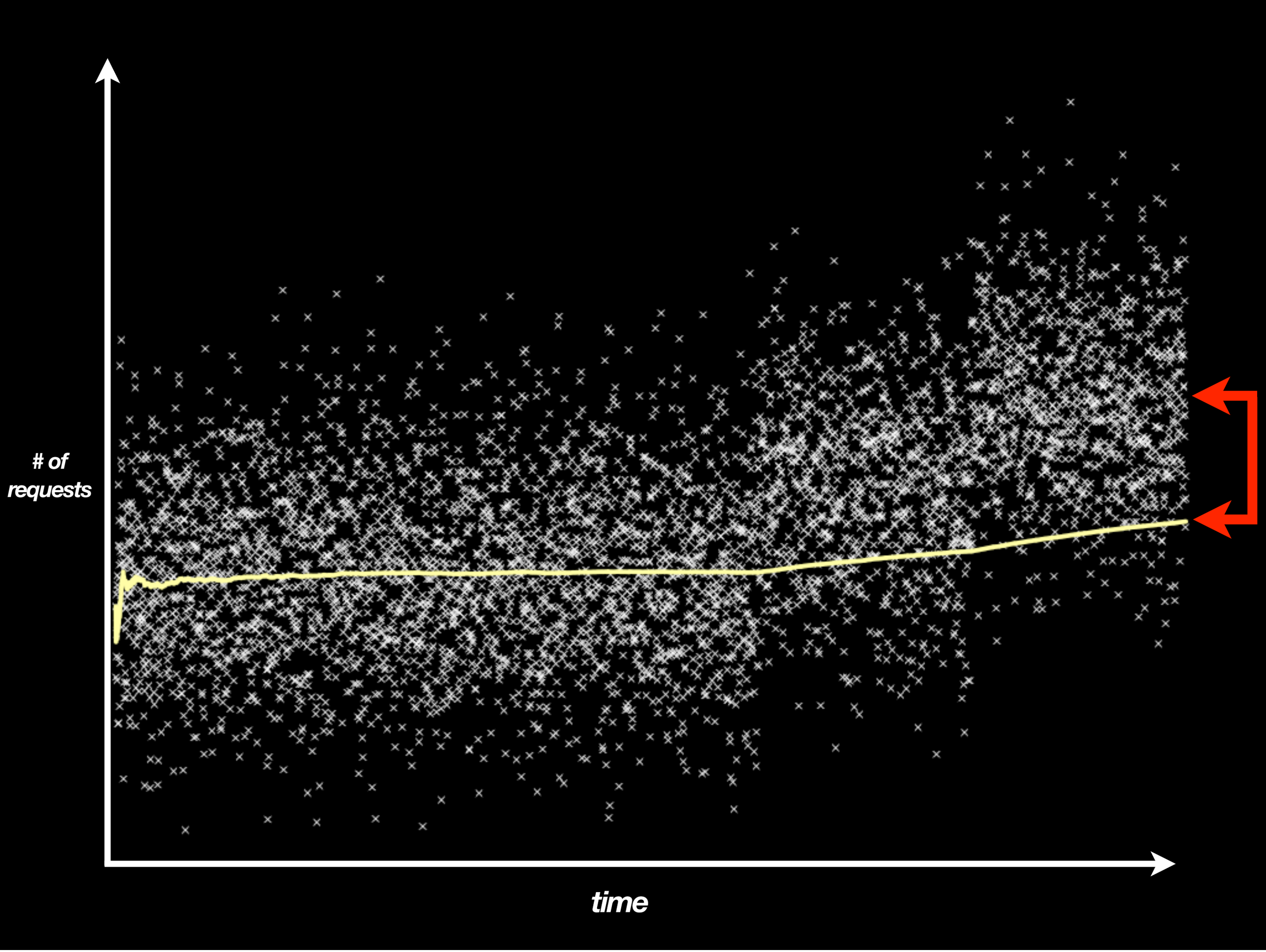
```
val meter = metrics.meter("requests",
                          SECONDS)
meter.mark()
```

```
val meter = metrics.meter("requests",
                          SECONDS)
meter.mark()
```

$$\text{mean rate} = \frac{\#\text{ of events}}{\text{elapsed time}}$$


# of requests
time


# of requests
time


# of requests
time

Saturday, April 9, 2011

Recency.

$$\text{mean rate} = \frac{\#\text{ of events}}{\text{elapsed time}}$$

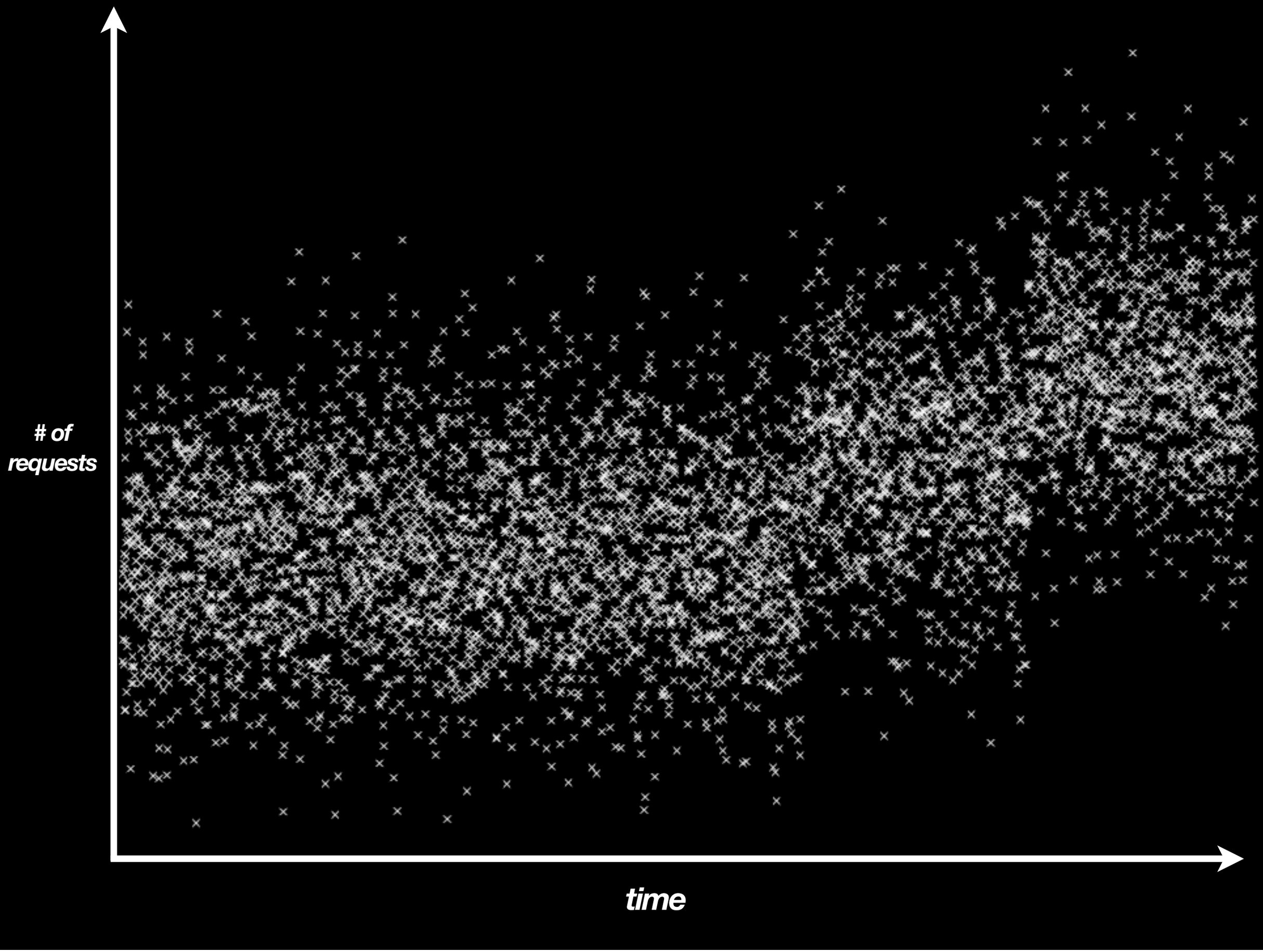

mean rate =
of events
elapsed time

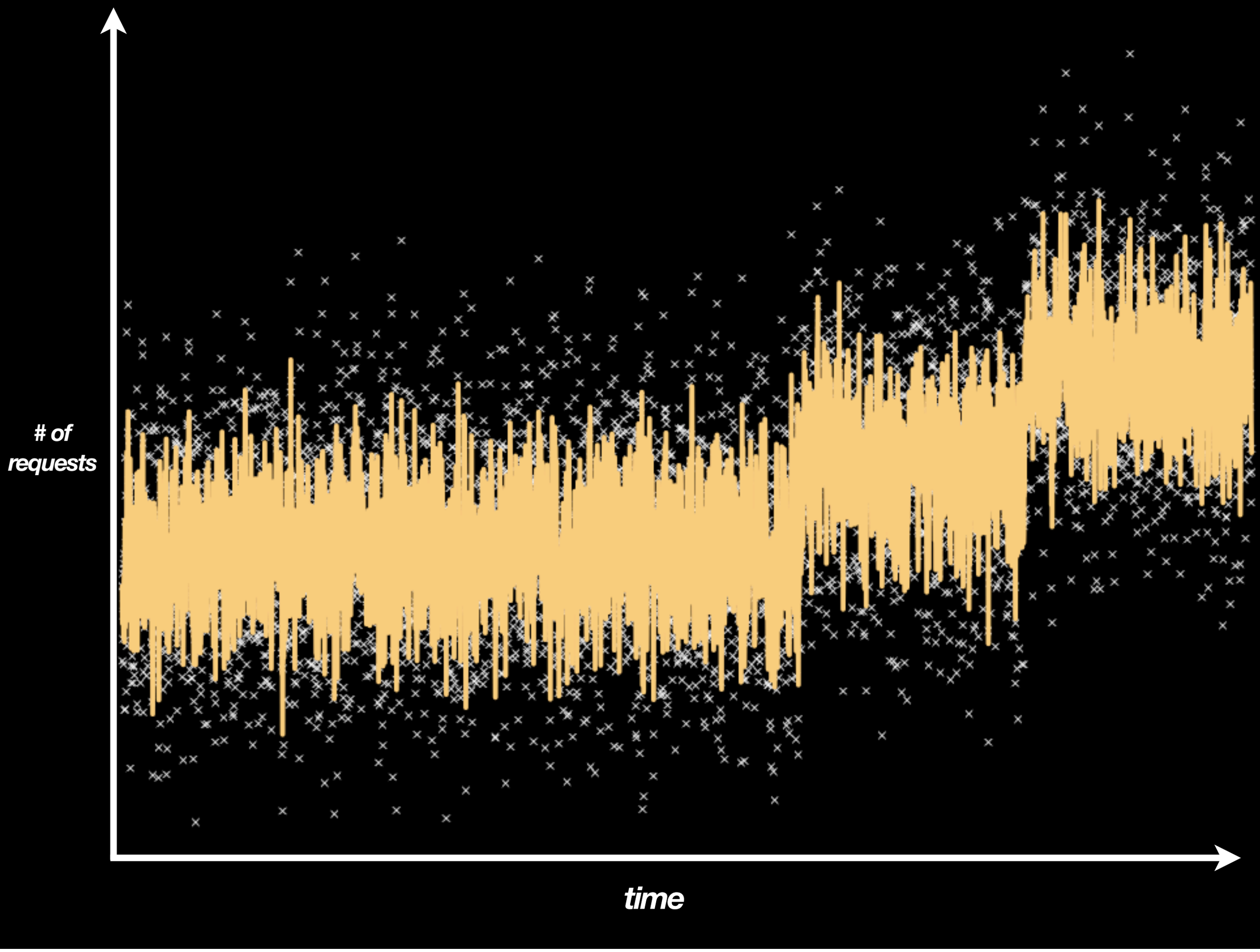
COGNITIVE HAZARD

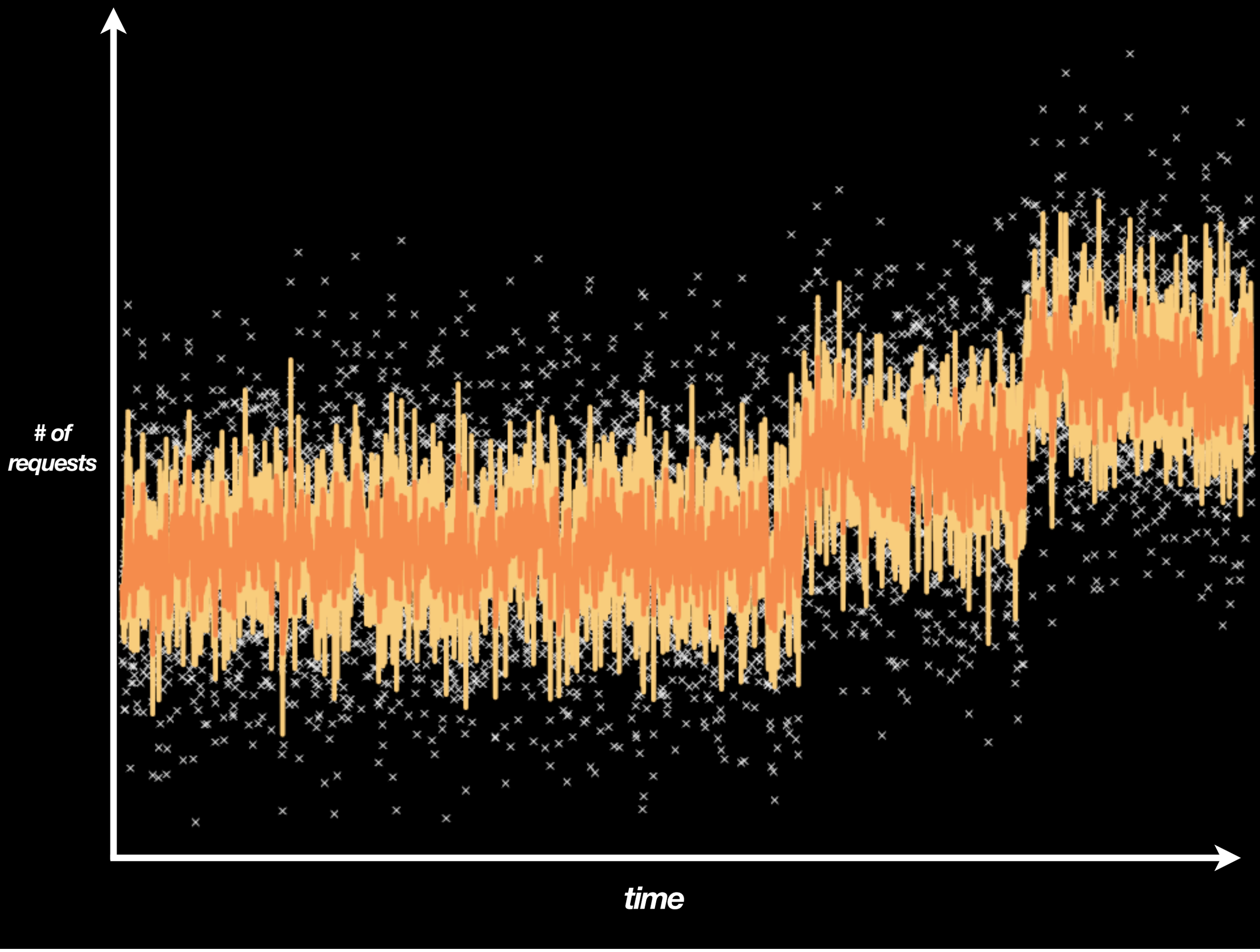
Exponentially weighted moving average.

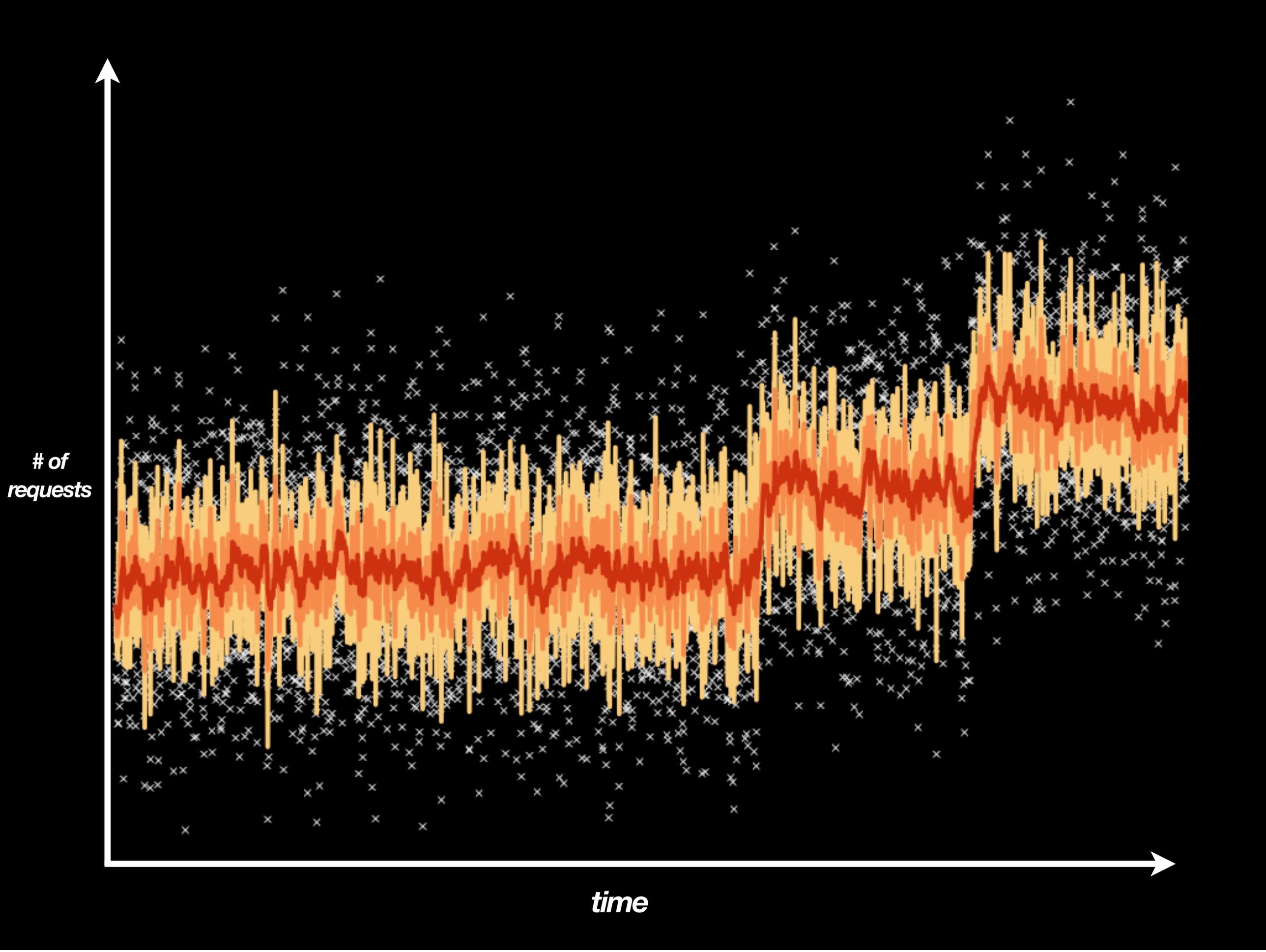
$$-(1-\alpha)^{k}\mathrm{m}_{t-1} + \frac{(1-(1-\alpha)^{k})\mathrm{Y}_{t}}{k}$$

$$-(1-\alpha)^{k}\mathrm{m}_{t\text{-}1} + \frac{(1-(1-\alpha)^{k})\mathrm{Y}_{t}}{k}$$


# of requests
time


# of requests
time


# of requests
time


# of requests
time

1-minute rate

1-minute rate
5-minute rate

1-minute rate

5-minute rate

15-minute rate

“We went from 3,000 requests/sec to <500 a second.”

Gauges
Counters
Meters
Histograms
Timers

Gauges
Counters
Meters
Histograms
Timers

Histogram

The statistical distribution of values in a stream of data.

of cities returned

```
val histogram =
  metrics.histogram("response-sizes")

histogram.update(response.cities.size)
```

```
val histogram =
  metrics.histogram("response-sizes")

histogram.update(response.cities.size)
```

```
val histogram =
  metrics.histogram("response-sizes")

histogram.update(response.cities.size)
```

minimum

minimum

maximum

minimum

maximum

mean

minimum

maximum

mean

standard deviation

Quantiles

Quantiles

median

Quantiles

median

75th percentile

Quantiles

median

75th percentile

95th percentile

Quantiles

median

75th percentile

95th percentile

98th percentile

Quantiles

median

75th percentile

95th percentile

98th percentile

99th percentile

Quantiles

median

75th percentile

95th percentile

98th percentile

99th percentile

99.9th percentile

We can’t keep all of these values.

1,000 req/sec

1,000 req/sec

×

1,000 req/sec

×

1,000 actions/req

1,000 req/sec

×

1,000 actions/req

×

1,000 req/sec

×

1,000 actions/req

×

1 day

1,000 req/sec

×

1,000 actions/req

×

1 day

=

1,000 req/sec

×

1,000 actions/req

×

1 day

=

>86 billion values

1,000 req/sec

×

1,000 actions/req

×

1 day

=

>86 billion values

>640GB of data/day

1,000 req/sec

×

1,000 actions/req

×

1 day

=

>86 billion values

>640GB of data/day

Not gonna happen.

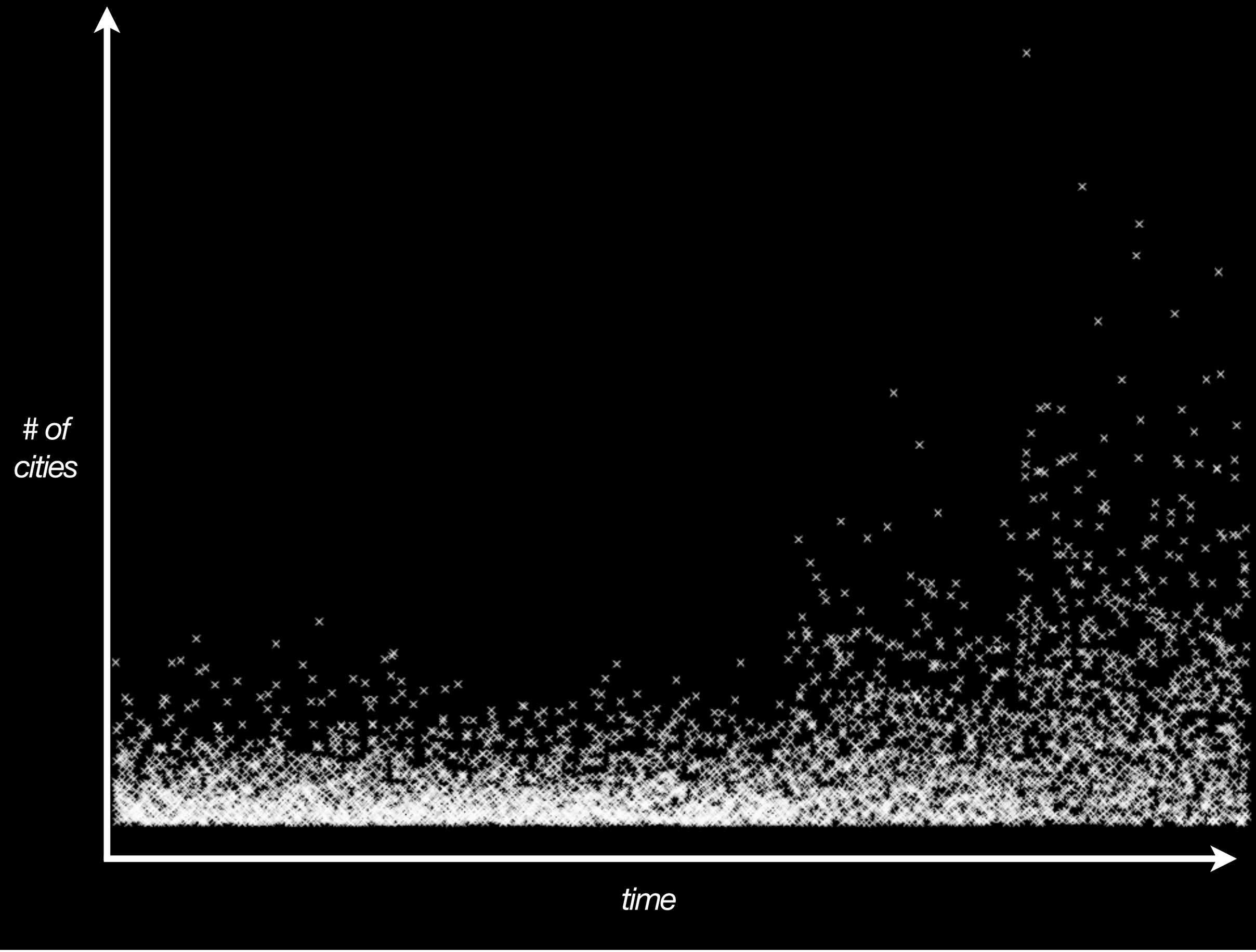
COGNITIVE HAZARD

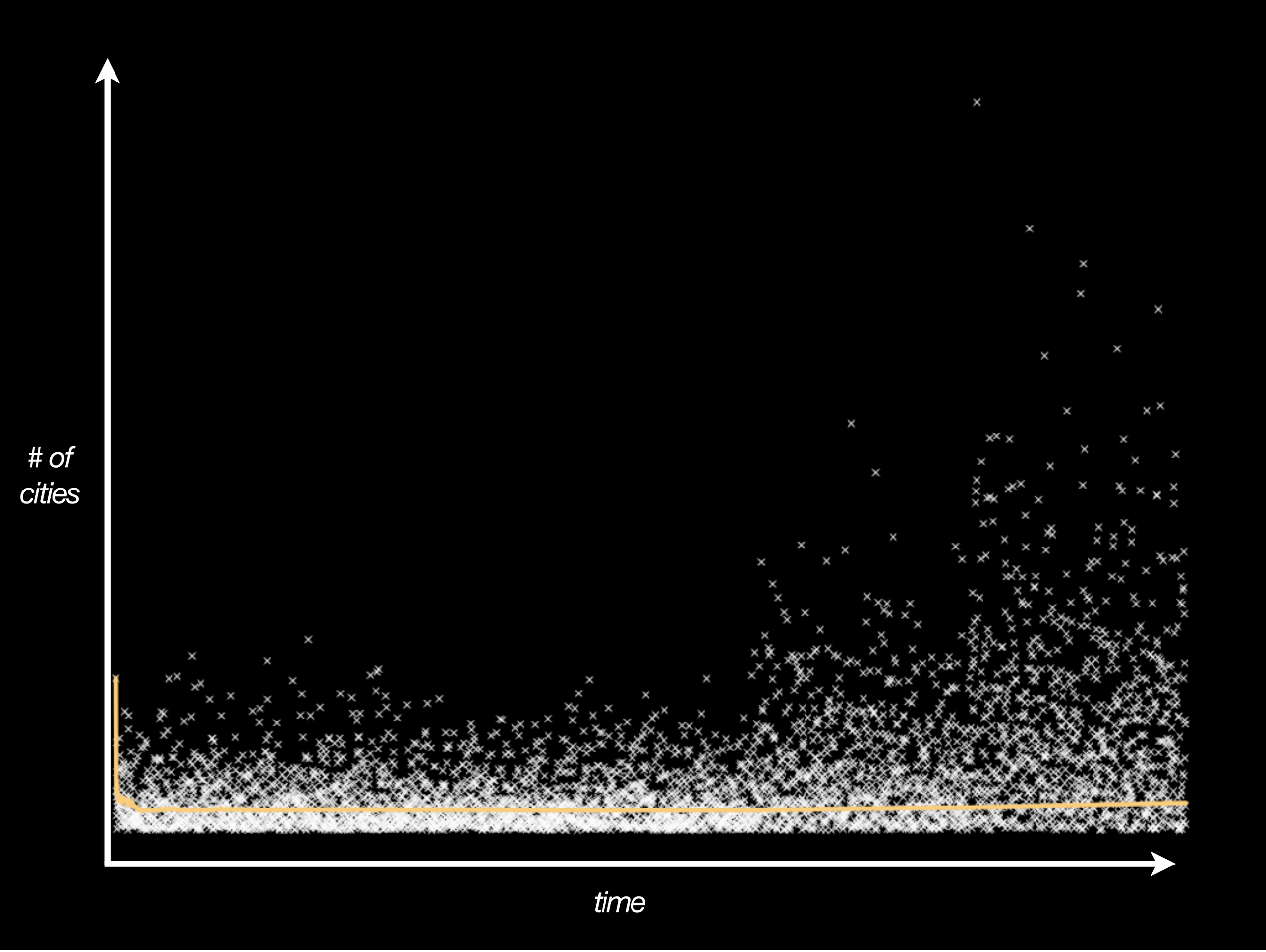
Reservoir sampling.

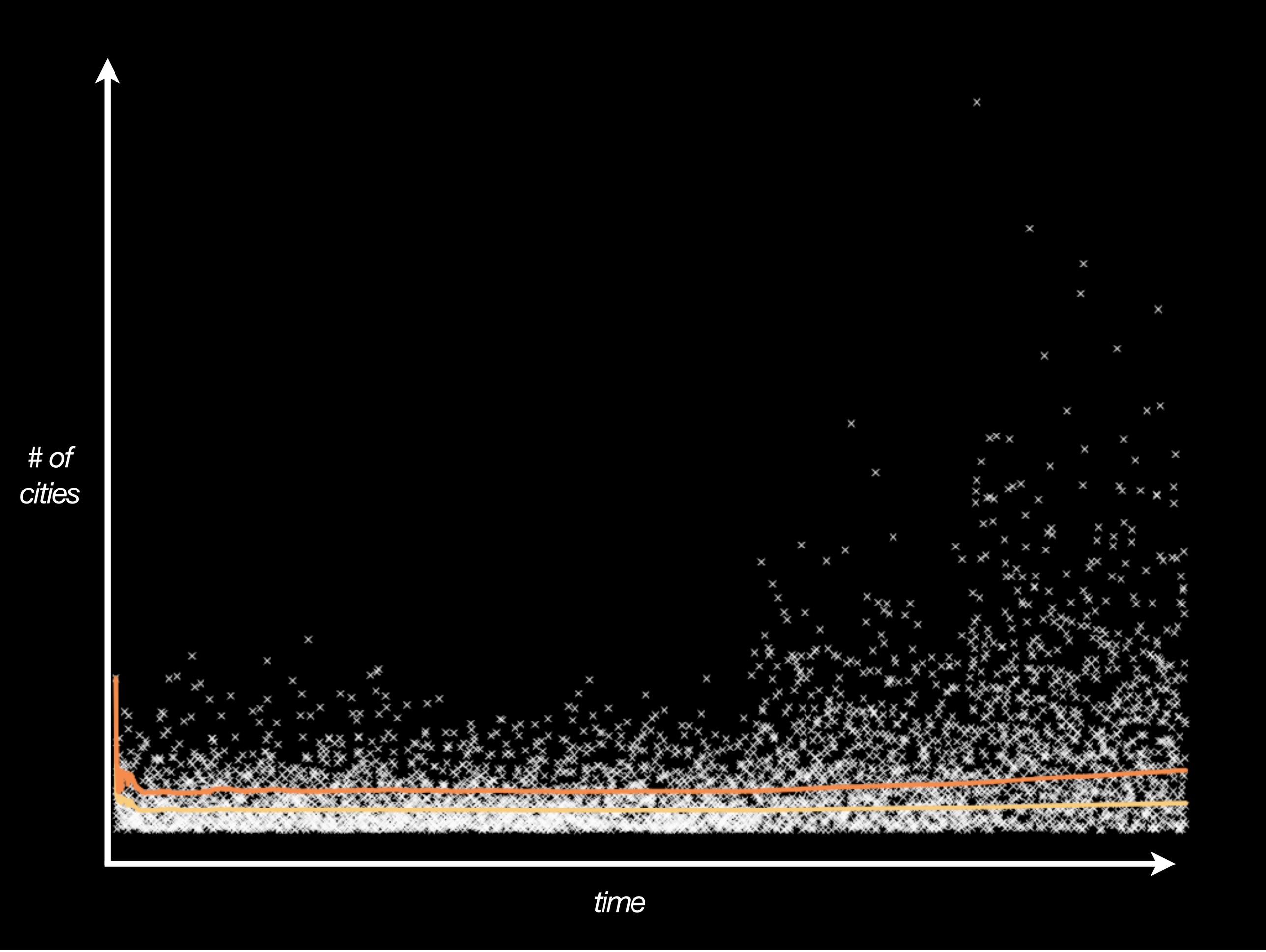
Keep a statistically representative sample
of measurements as they happen.

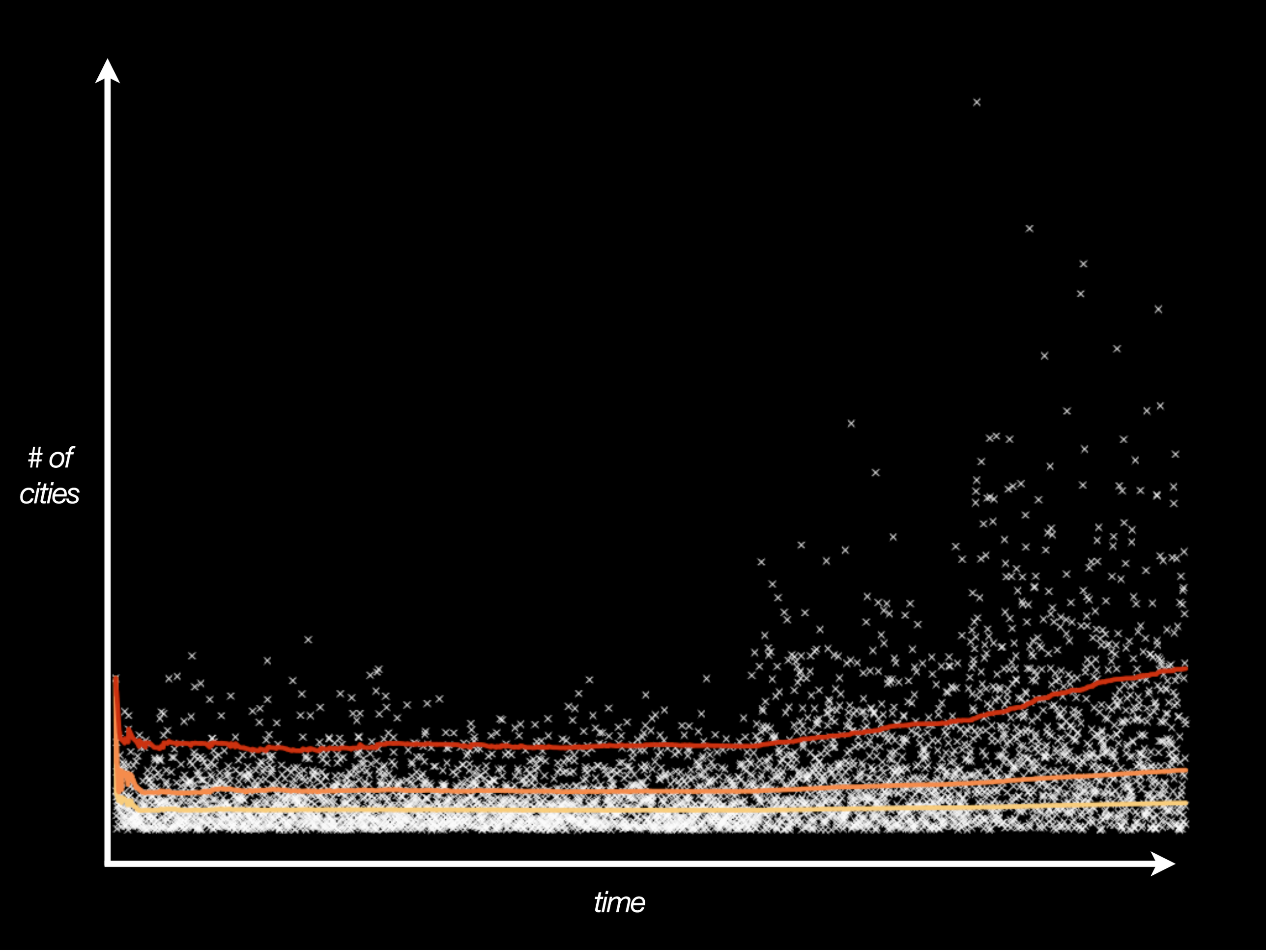
Vitter's *Algorithm R*.

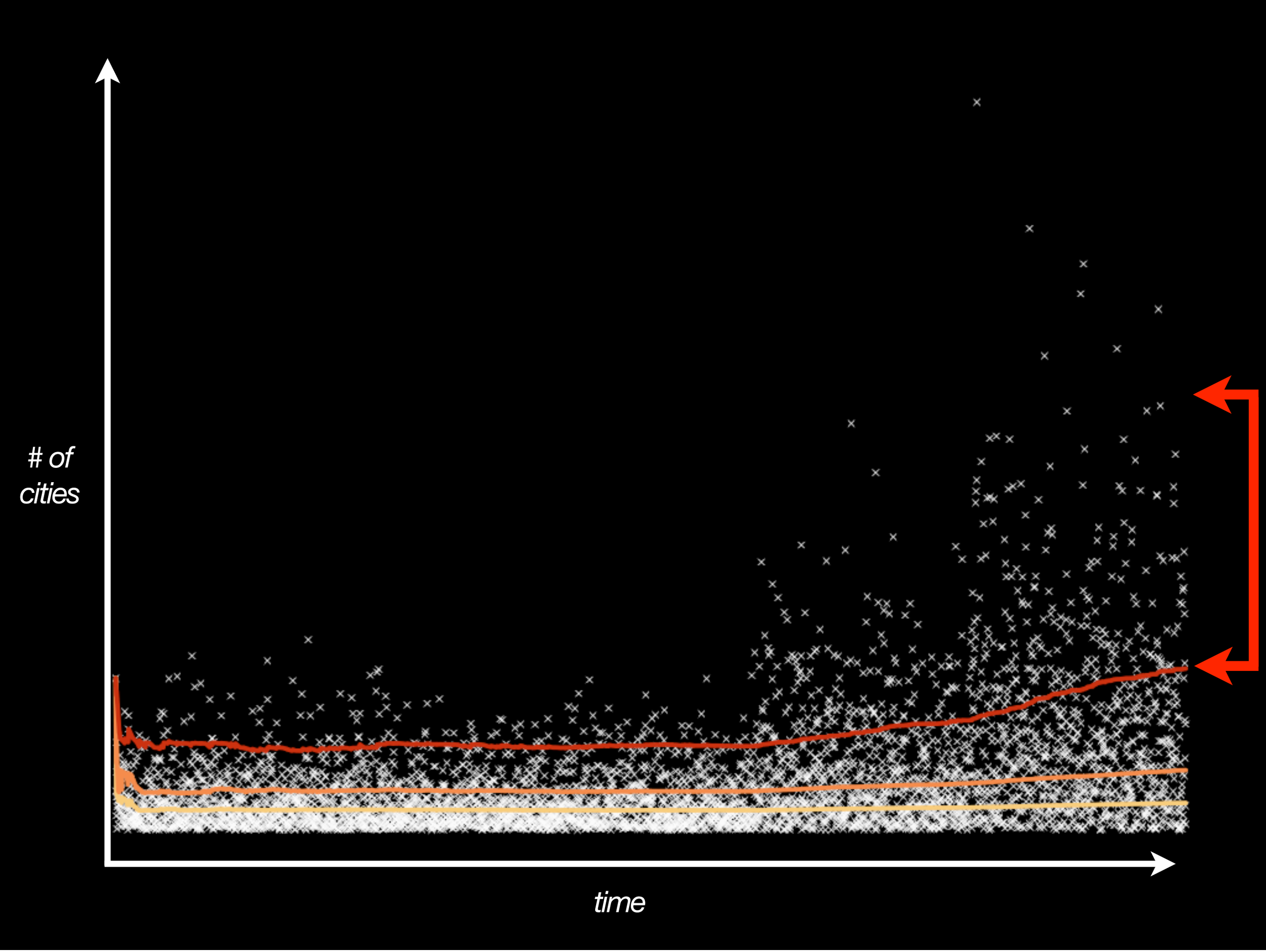
Vitter, J. (1985).
Random sampling with a reservoir.
ACM Transactions on Mathematical Software (TOMS), 11(1), 57.


# of cities
time


# of cities
time


# of cities
time


# of cities
time


# of cities
time

Saturday, April 9, 2011

Vitter's *Algorithm R* produces **uniform** samples.

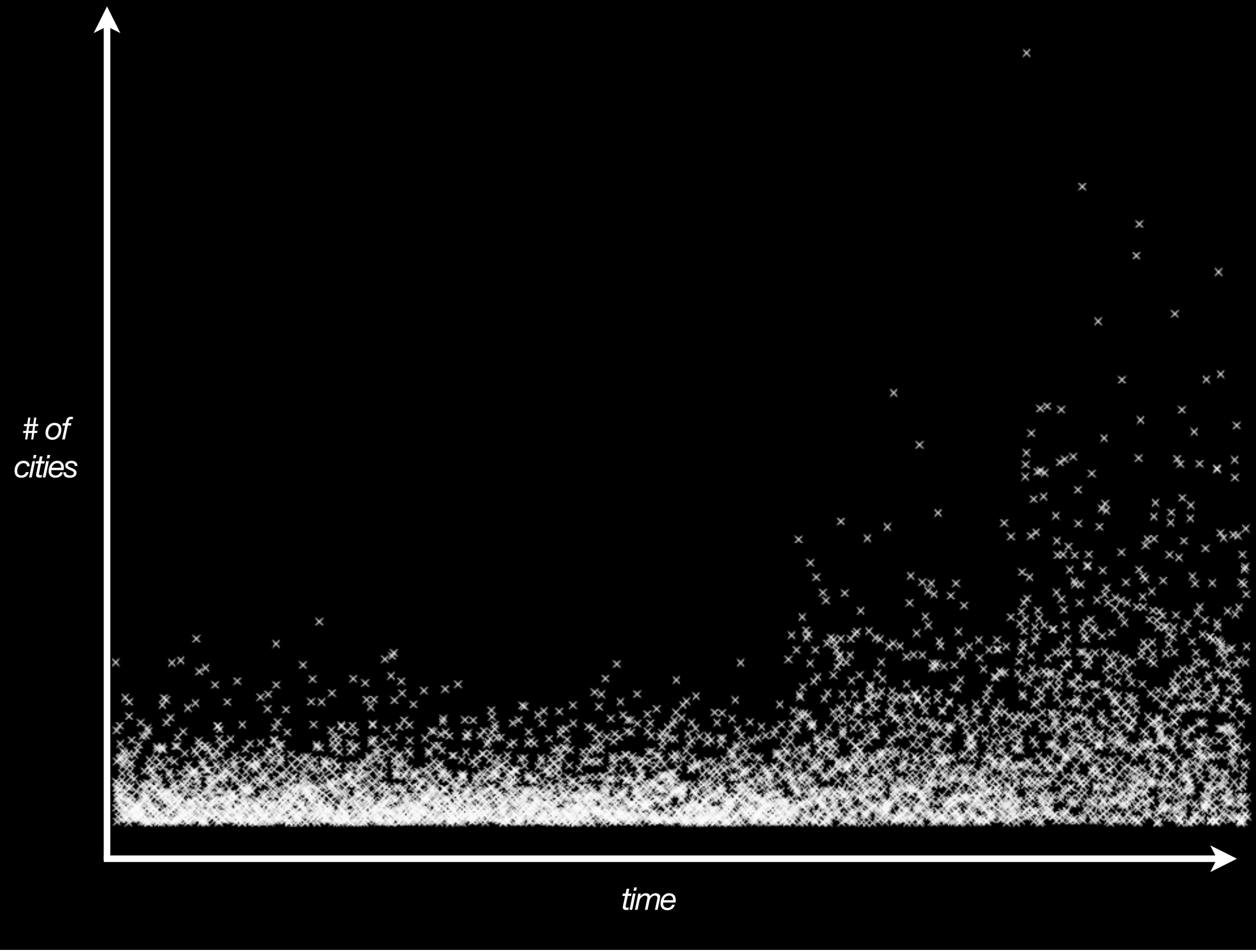
Recency.

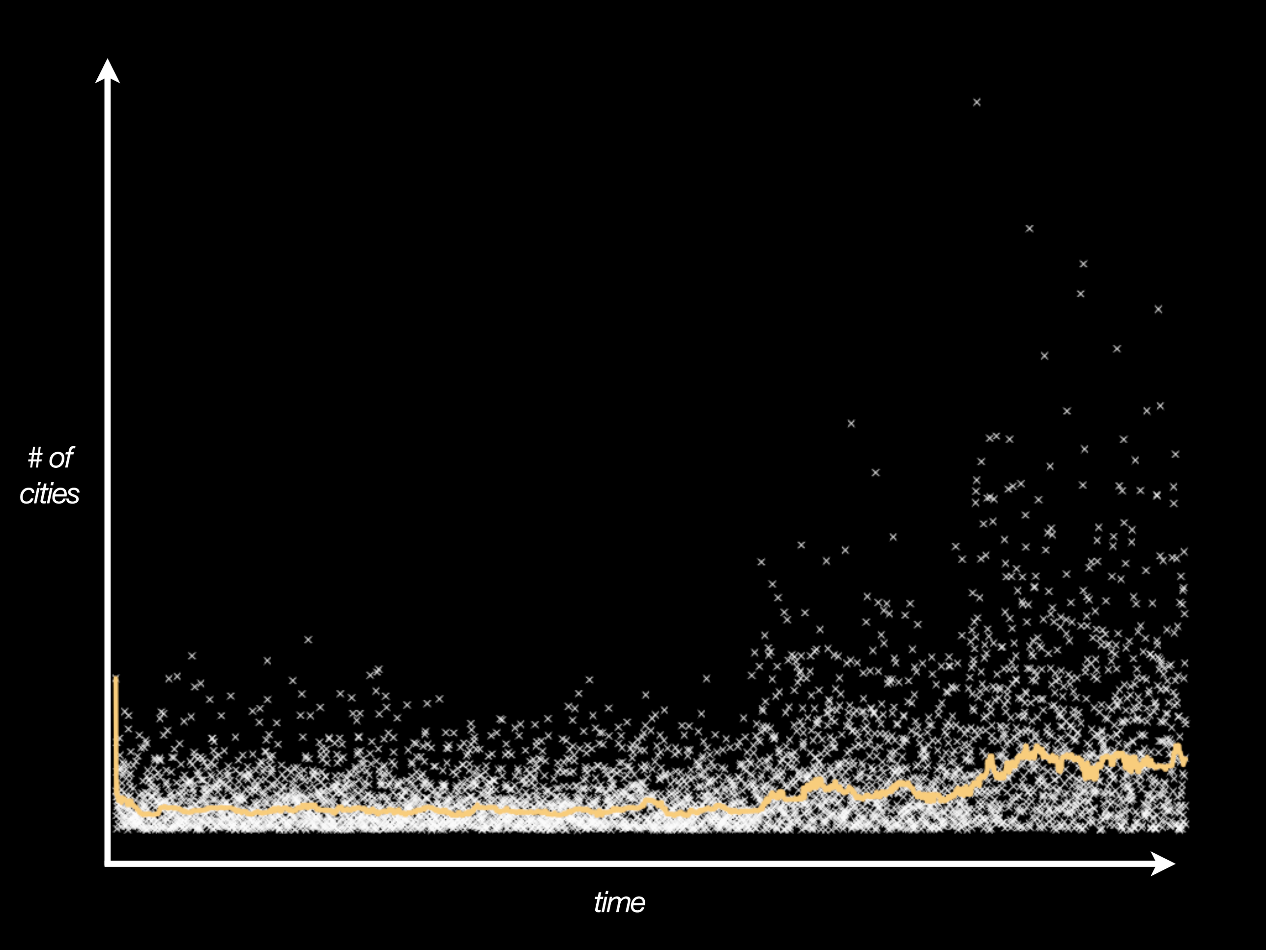
SUPER-DUPER COGNITIVE HAZARD

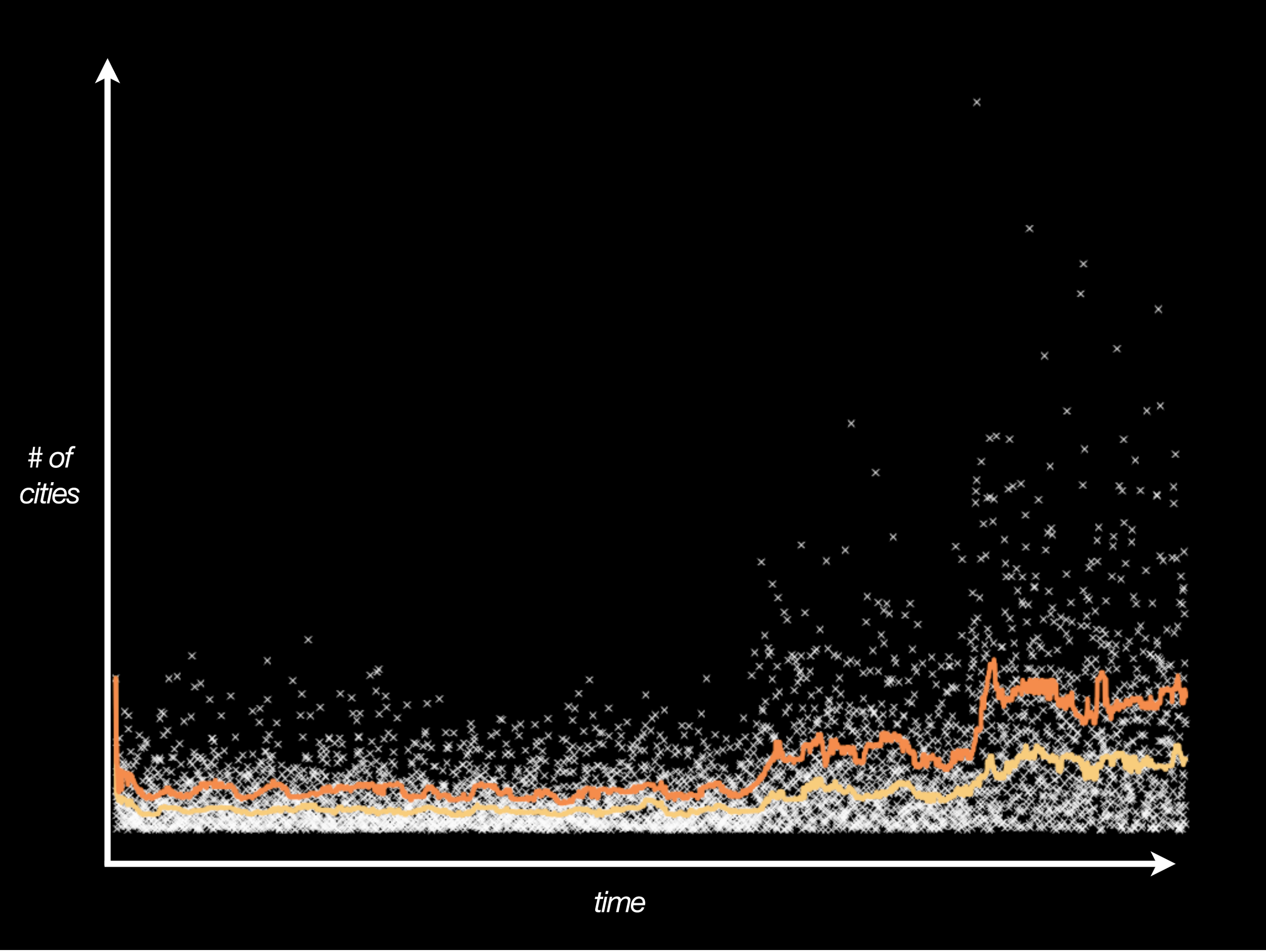
Forward-decaying priority sampling.

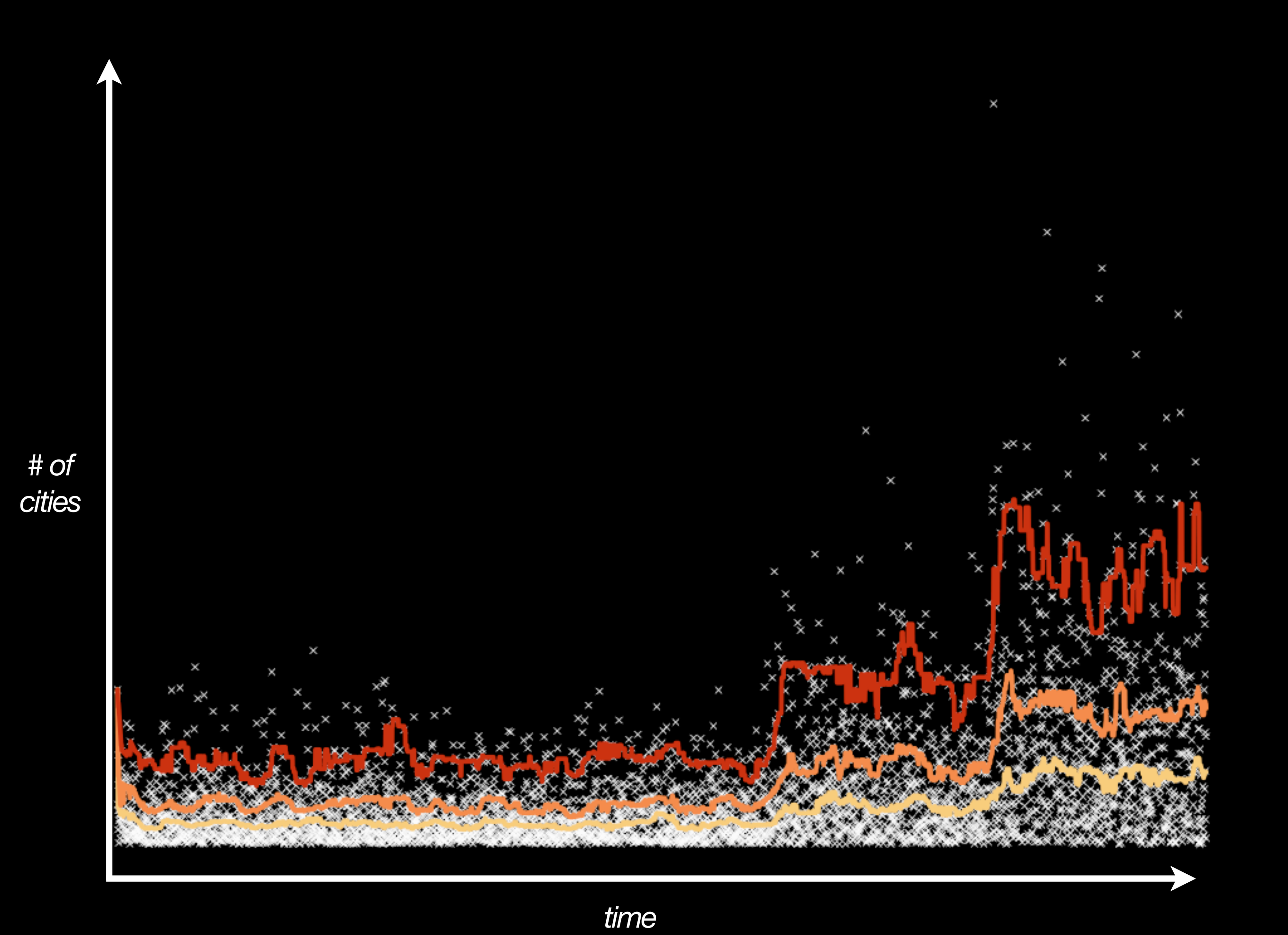
Cormode, G., Shkapenyuk, V., Srivastava, D., & Xu, B. (2009).
Forward Decay: A Practical Time Decay Model for Streaming Systems.
ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering.

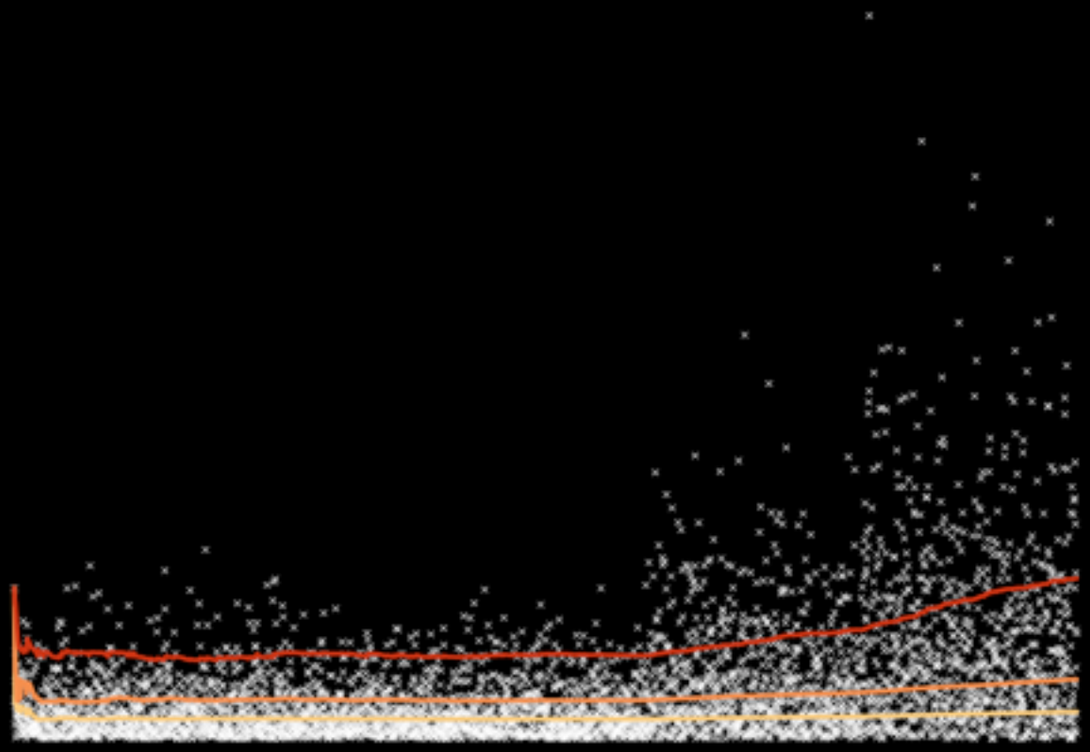
Maintain a statistically representative sample of the *last 5 minutes*.


# of cities
time

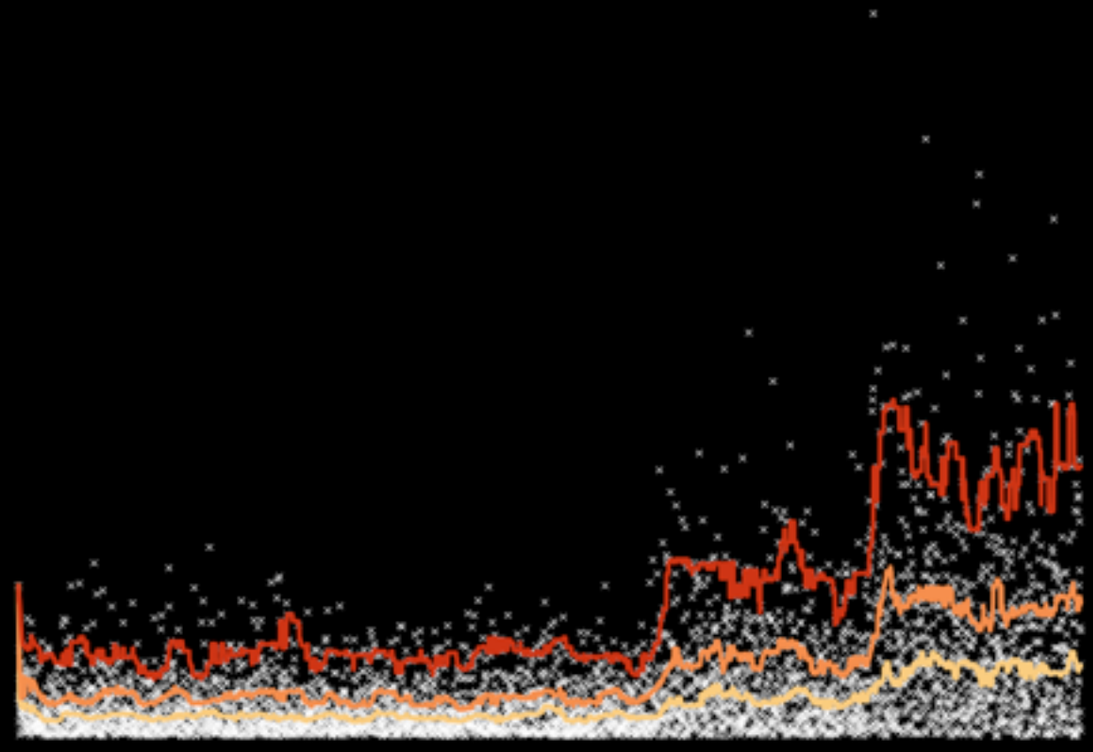

# of cities
time


# of cities
time


# of cities
time



Uniform



Biased

“95% of autocomplete results return 3 cities or less.”

Gauges
Counters
Meters
Histograms
Timers

Gauges
Counters
Meters
Histograms
Timers

Timer

A histogram of durations and a meter of calls.

of *ms* to respond

```
val timer = metrics.timer("requests",
                          MILLISECONDS,
                          SECONDS)

timer.time { handle(req, resp) }
```

```
val timer = metrics.timer("requests",
                          MILLISECONDS,
                          SECONDS)

timer.time { handle(req, resp) }
```

```
val timer = metrics.timer("requests",
                          MILLISECONDS,
                          SECONDS)

timer.time { handle(req, resp) }
```

```
val timer = metrics.timer("requests",
                          MILLISECONDS,
                          SECONDS)

timer.time { handle(req, resp) }
```

```
val timer = metrics.timer("requests",
                          MILLISECONDS,
                          SECONDS)

timer.time { handle(req, resp) }
```

“At ~2,000 req/sec, our 99% latency jumps from 13ms to 453ms.”

Gauges
Counters
Meters
Histograms
Timers

Gauges
Counters
Meters
Histograms
Timers

Now what?

Instrument it.

Instrument it.

If it could affect your **code**'s
business value, add a metric.

Instrument it.

If it could affect your **code**'s
business value, add a metric.
Our services have 40-50 metrics.

Collect it.

Collect it.

JSON via HTTP.

Collect it.

JSON via HTTP.
Every minute.

Monitor it.

Monitor it.

Nagios/Zabbix/Whatever

Monitor it.

Nagios/Zabbix/Whatever
If it affects **business value**,
someone should get **woken up**.

Aggregate it.

Aggregate it.

Ganglia/Graphite/Cacti/Whatever

Aggregate it.

Ganglia/Graphite/Cacti/Whatever

Place current values in historical context.

Aggregate it.

Ganglia/Graphite/Cacti/Whatever

Place current values in historical context.

See long-term patterns.

Go *faster*.

Shorten our

decision-making cycle.

Observe

Observe
Orient

Observe
Orient
Decide

Observe
Orient
Decide
Act

Observe
Orient
Decide
Act

Observe

What is the 99% latency of our autocomplete service right now?

Observe

What is the 99% latency of our autocomplete service right now?

~500ms

Orient

How does this compare to other parts of our system, both currently and historically?

Orient

How does this compare to other parts of our system, both currently and historically?

way slower

Decide

Should we make it faster?

Or should we add feature X?

Decide

Should we make it faster?

Or should we add feature X?

make it faster

Act!

Write some code.

Act!

Write some code.

```
def sort_by(&blk)
  #sleep(100) # WTF DUDE
  super(&blk)
end
```

```
10 Print "Rinse"
20 Print "Repeat"
30 Goto 10
```

If we do this **faster**

we will **win.**

Fewer bugs.

More features.

Happier

users.

Money:

tl;dr

We might write **code**.

We have to generate

business value.

In order to know how well our **code** is generating **business value**, we need **metrics**.

Gauges
Counters
Meters
Histograms
Timers

Monitor them for current problems.

Aggregate them for historical perspective.

map ≠ territory

map → territory

Improve our **mental**

model of our **code**.

Saturday, April 9, 2011

Observe
Orient
Decide
Act

If you're on the JVM,

use ***Metrics***.

If you're on the JVM,

use ***Metrics***.

github.com/codahale/metrics

If not,

you can build this.

Please build this.

Make better **decisions**

by using **numbers**.

Thank you.