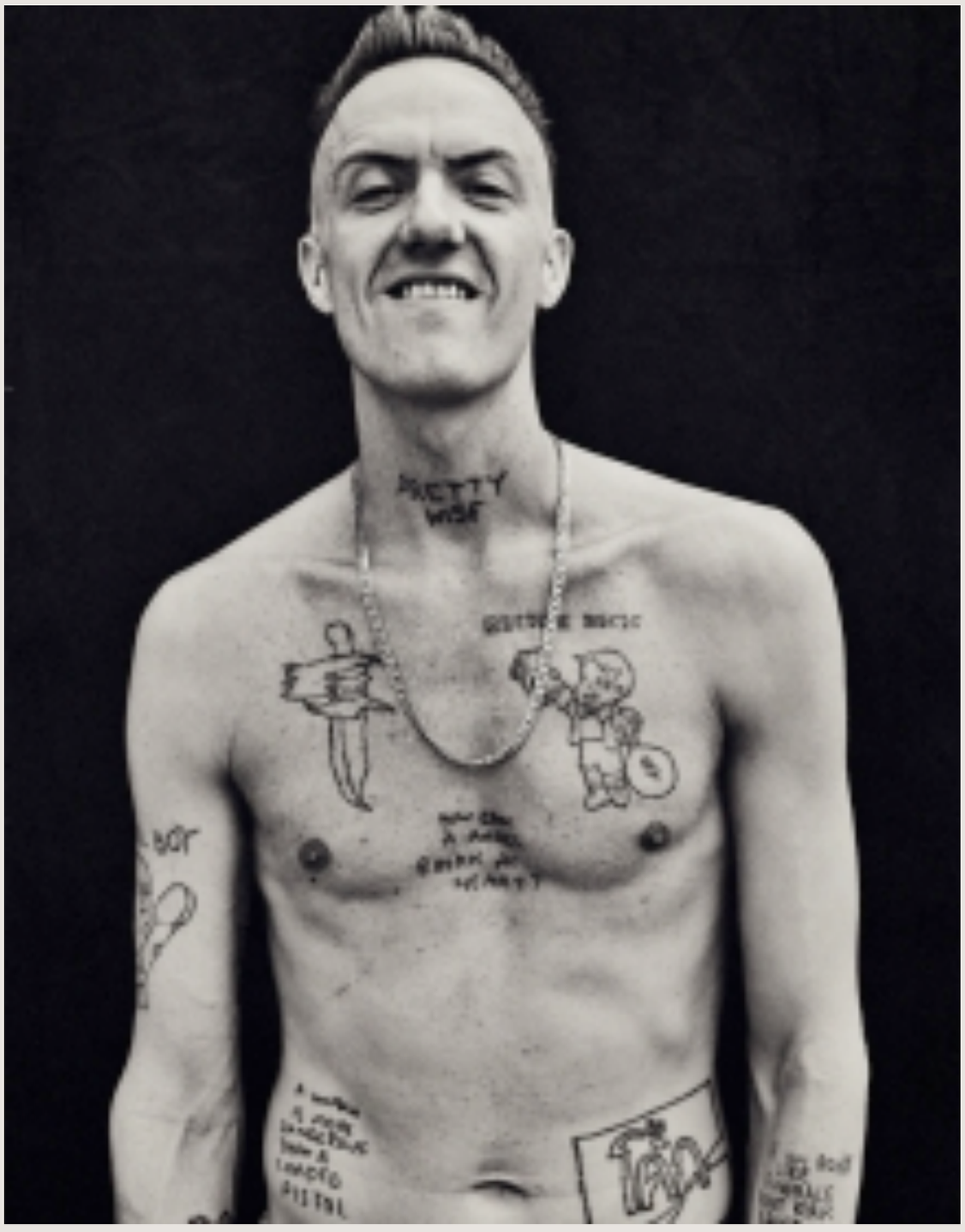
production monitoring. zef side.




CLOJURE
WERKZ

stylefruits.de

here not to teach

but to bring your

attention

how many

servers do you

have?

how many page views do you have?

do you monitor your system extensively?

what do you use?

So,

production?..

looks like that:

yeah, that makes

total sense!..

(c) noone. never.

what's there?...

looking for

patterns

identify

constructs

repetitions

trends

outliers

dispersion

deviation

fragmentation

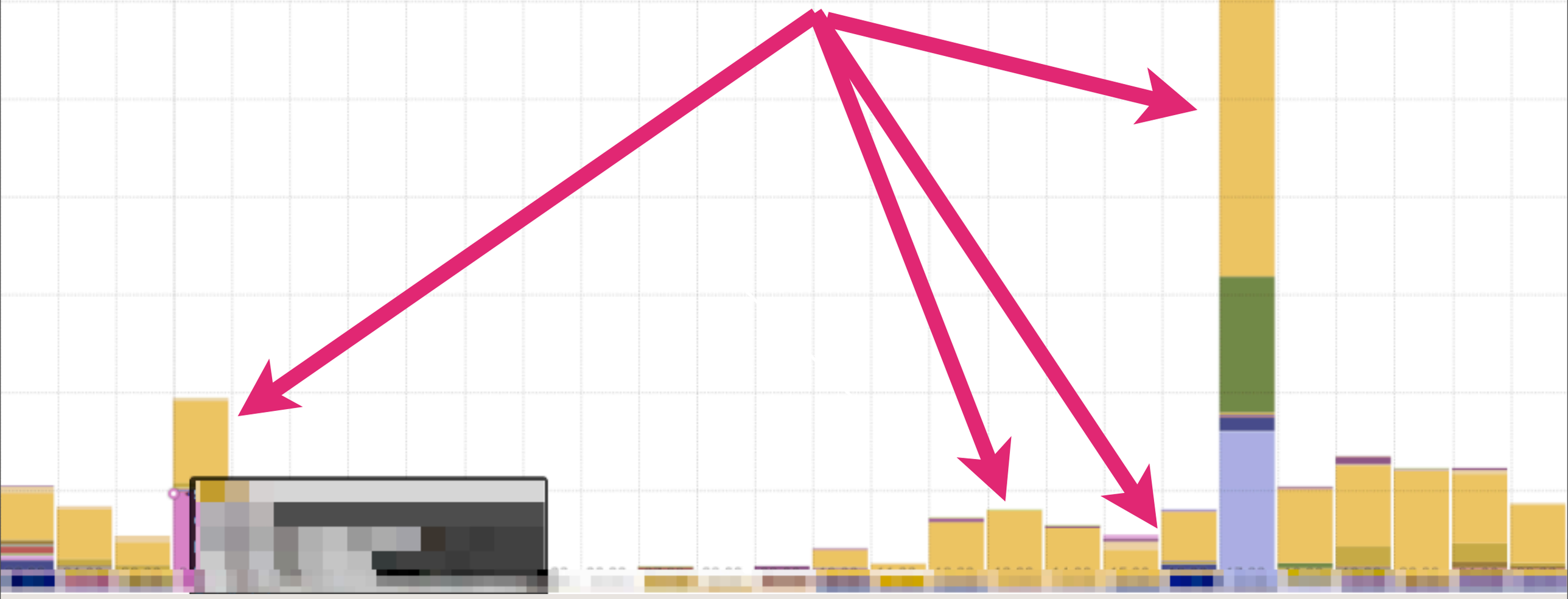
true story:

sittin' there,

gathering exceptions from production

suddenly

Y SO MUCH YELLOW?



check into

exception

details...

3 words:

parse-query-string

Ah shoo!

That's not really an exception!

That's just a 404!

Eliminate the cause

Avoid hitting complete stack when there's no way to do it

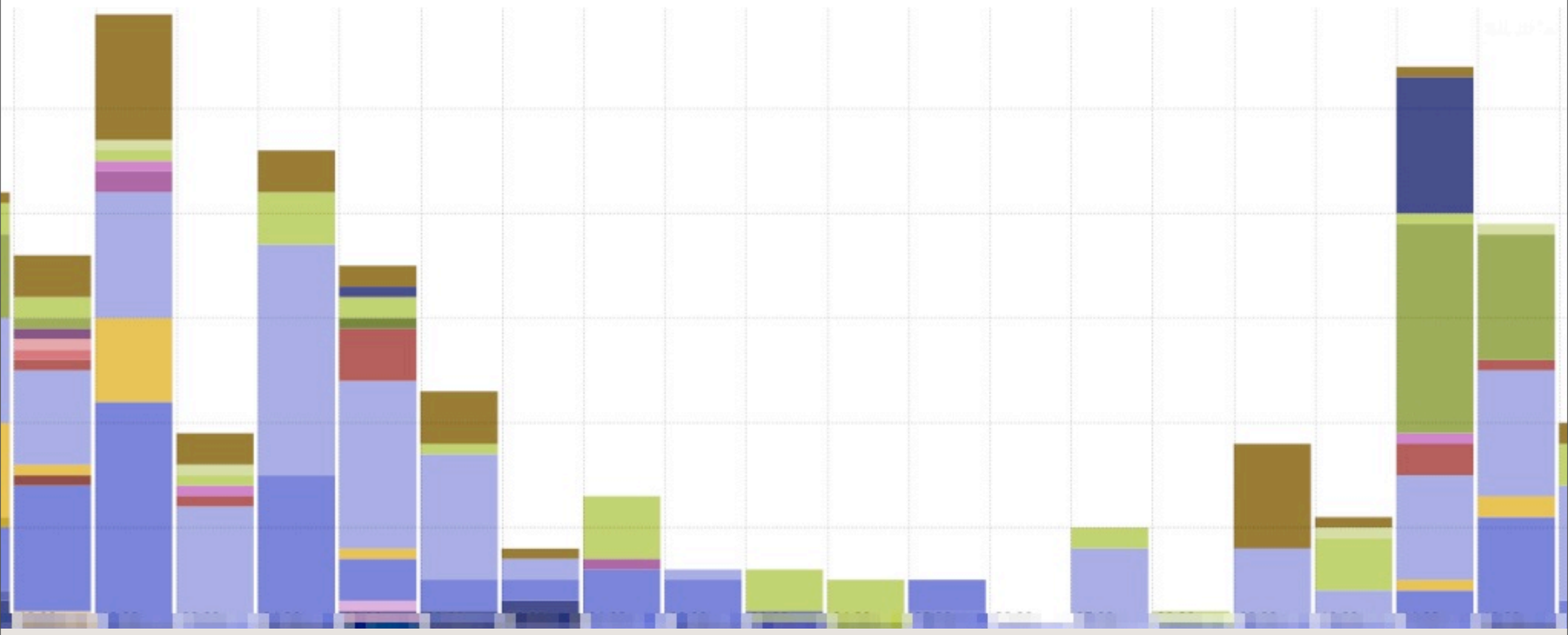
PROFIT!

Also, we know that we now know how to identify **exceptions** that are **404s** in reality

exceptions => 404s

that look like 404s

are actually

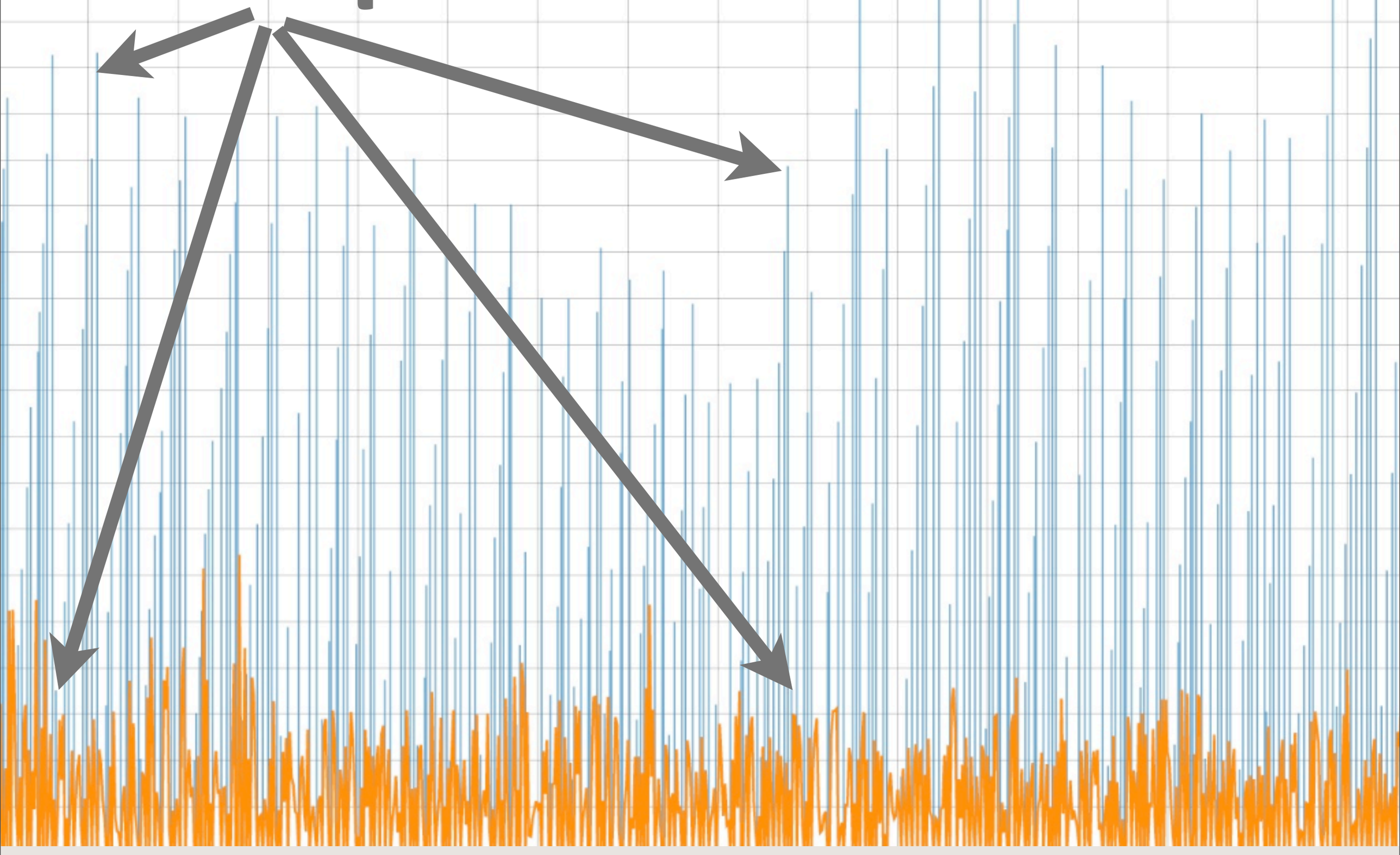


Still some repetition, but much better

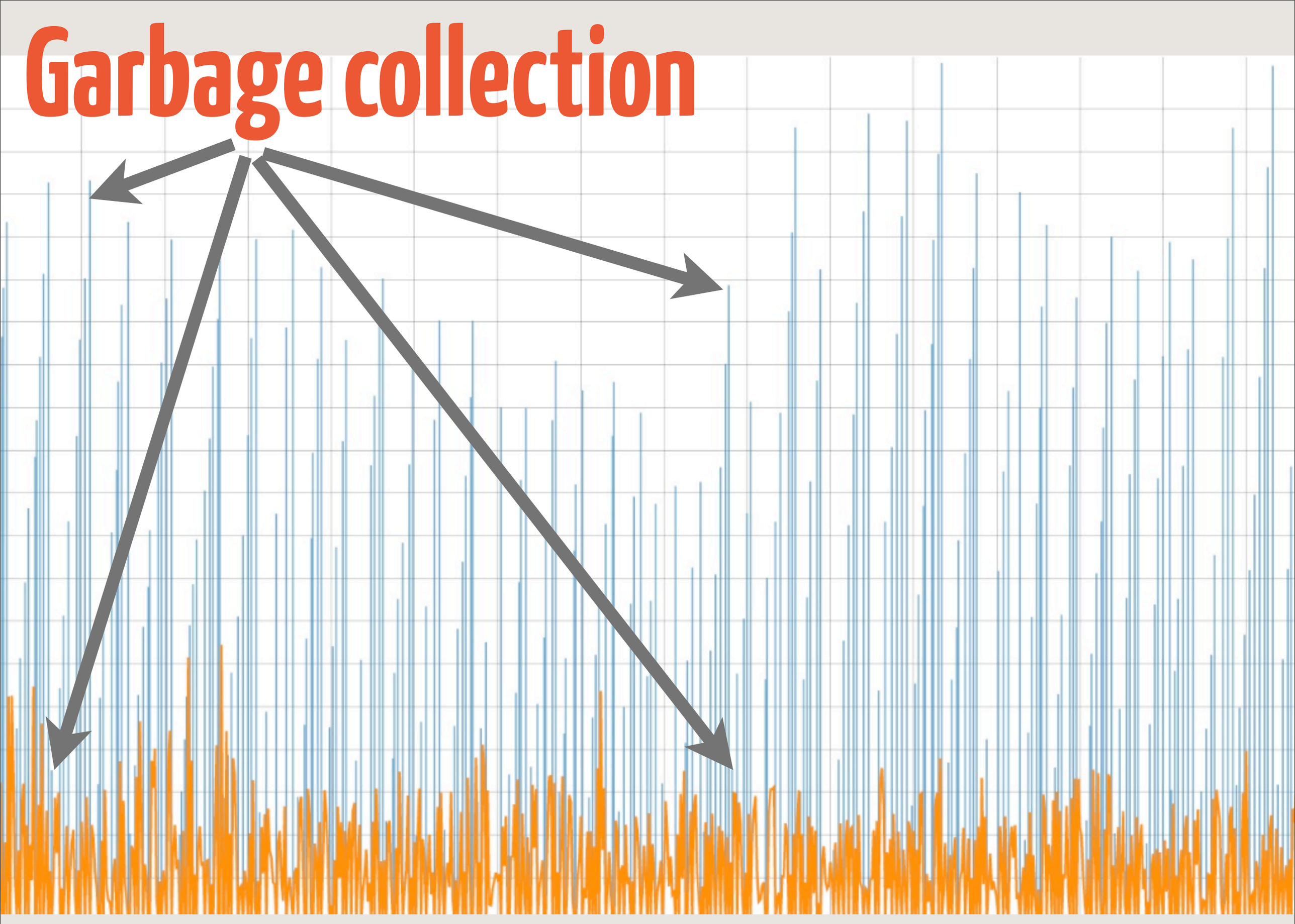
Repetition is not necessarily bad

But you must understand why it occurs

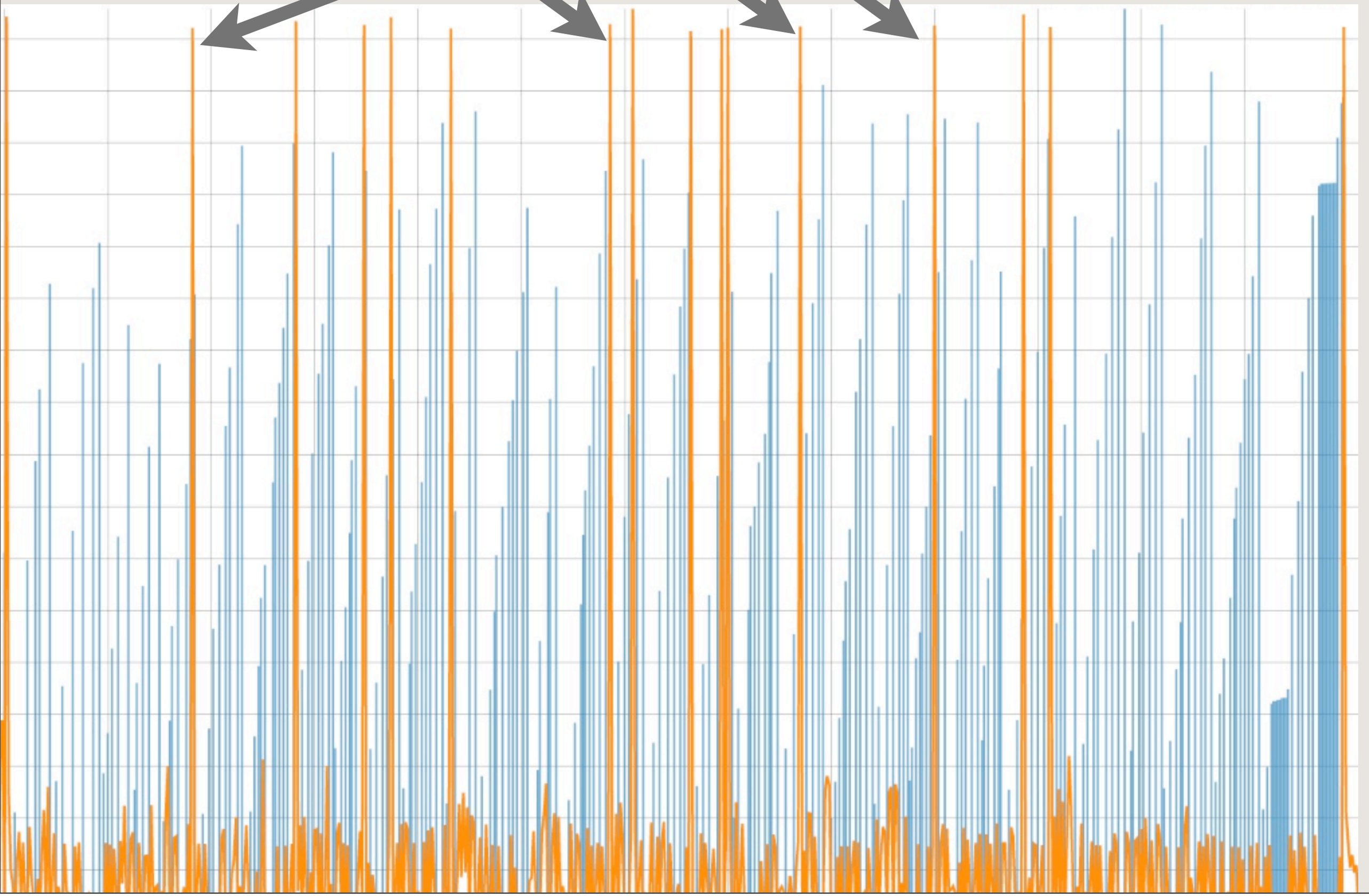
Good repetition.



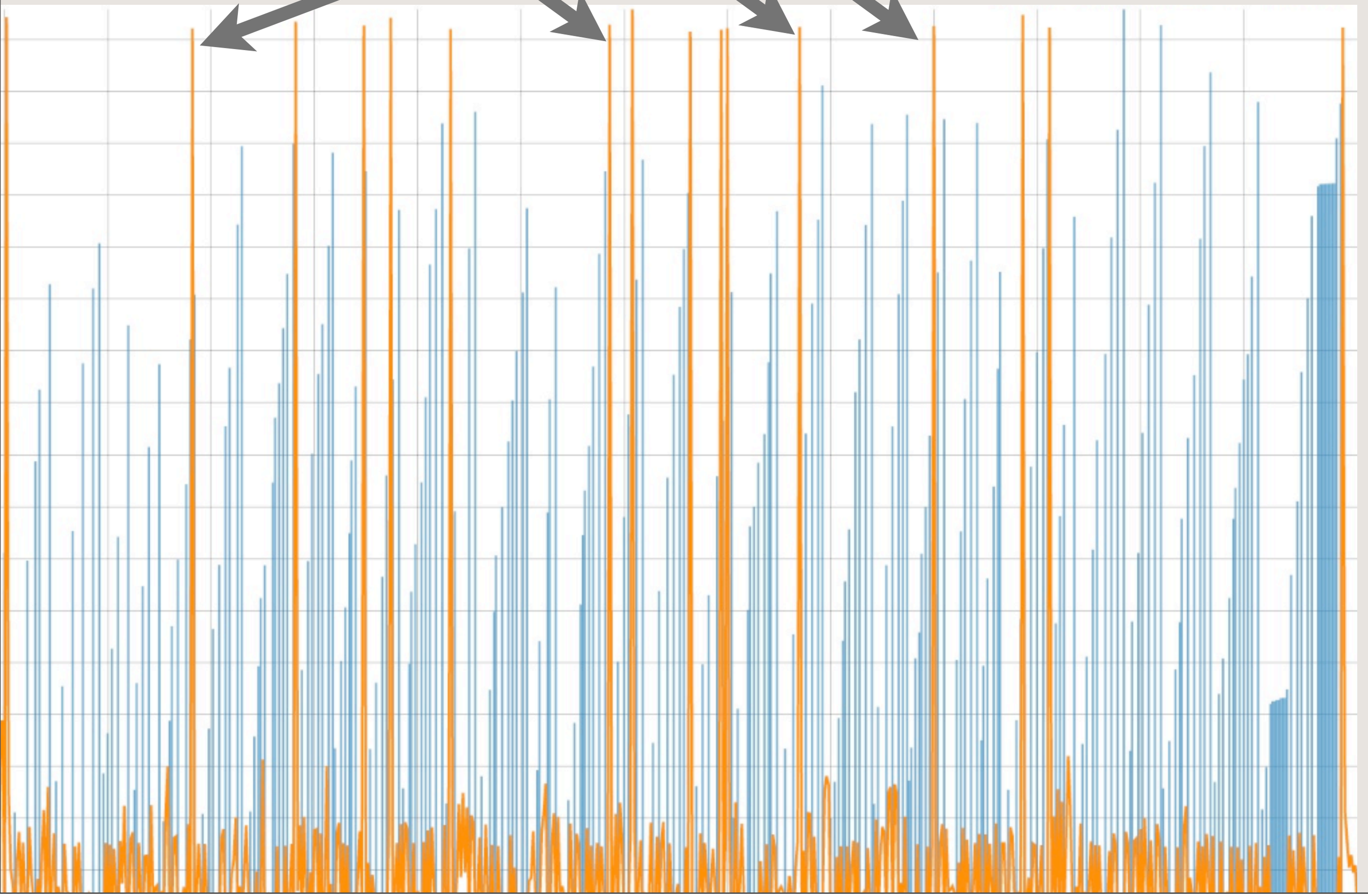
Garbage collection



Bad Repetition



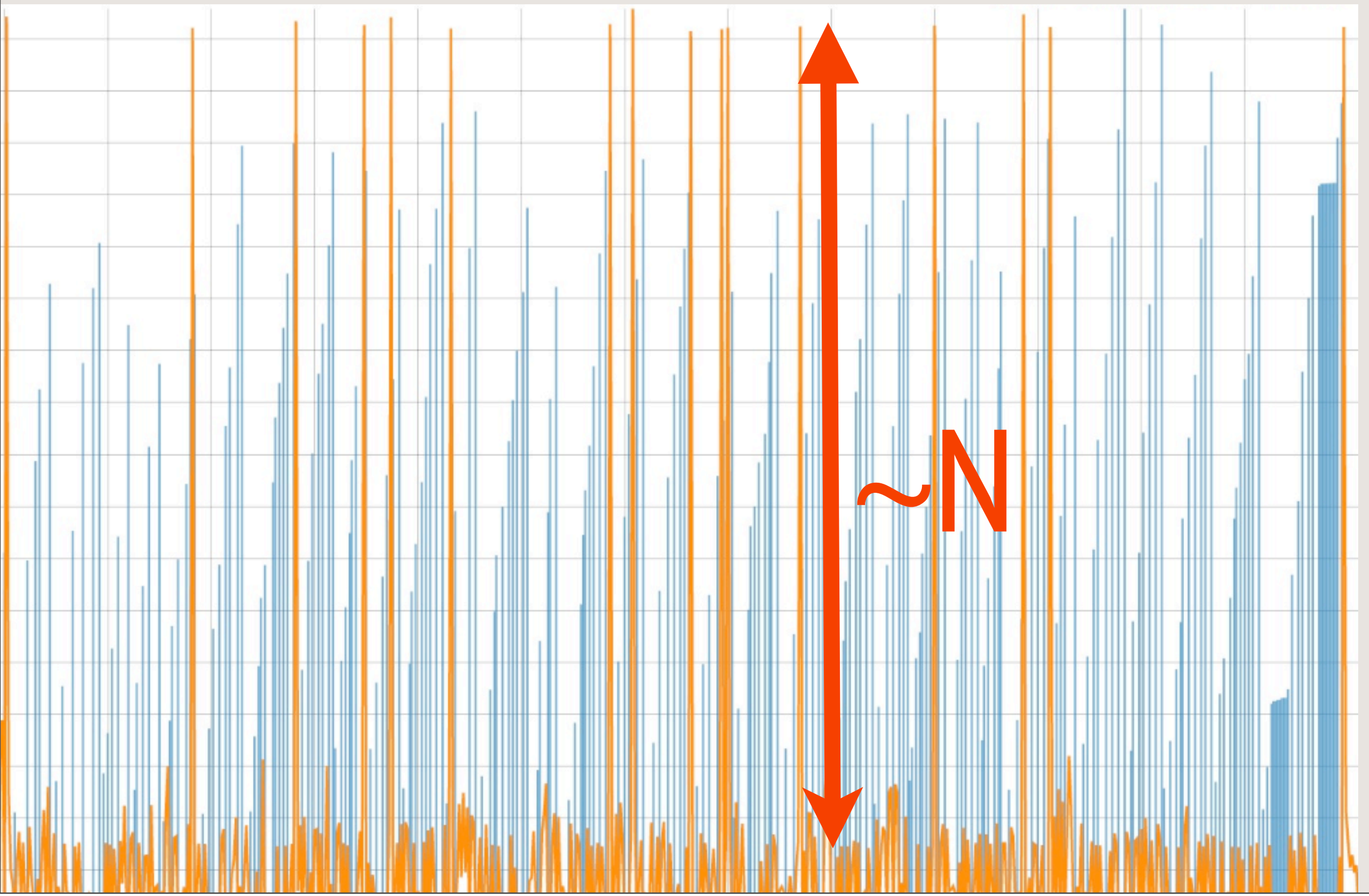
Performance decrease



any other hints?

hmm...

could not obtain database connection within N sec.



Saturday, October 27, 12

tweak parallelism / connection pool size.

PROFIT!

or...

suddenly, application report rate

decreases

server responds only to every third/fourth request

works absolutely fine

after restart

but behavior recurs after 5-7

minutes...

somewhere in log files:

```
Too many open files ....
```

and

```
java.net.UnknownHostException ...
```

lsof

```
java  25673  [redacted]  790u  IPv6  53685187  0t0  UDP *:15596
java  25673  [redacted]  791u  IPv6  53687041  0t0  UDP *:42130
java  25673  [redacted]  792u  IPv6  53685196  0t0  UDP *:20910
java  25673  [redacted]  793u  IPv6  53686984  0t0  UDP *:5284
java  25673  [redacted]  794u  IPv6  53686971  0t0  UDP *:4324
java  25673  [redacted]  795u  IPv6  53686990  0t0  UDP *:62078
java  25673  [redacted]  796u  IPv6  53687213  0t0  UDP *:39681
java  25673  [redacted]  797u  IPv6  53687022  0t0  UDP *:38956
java  25673  [redacted]  798u  IPv6  53687057  0t0  UDP *:56857
java  25673  [redacted]  799u  IPv6  53687051  0t0  UDP *:40148
java  25673  [redacted]  800u  IPv6  53687224  0t0  UDP *:10271
java  25673  [redacted]  801u  IPv6  53687163  0t0  UDP *:46985
java  25673  [redacted]  802u  IPv6  53687197  0t0  UDP *:41271
java  25673  [redacted]  803u  IPv6  53687094  0t0  UDP *:26566
java  25673  [redacted]  804u  IPv6  53687183  0t0  UDP *:45242
java  25673  [redacted]  805u  IPv6  53687170  0t0  UDP *:8359
java  25673  [redacted]  806u  IPv6  53687206  0t0  UDP *:41881
java  25673  [redacted]  807u  IPv6  53687234  0t0  UDP *:20064
java  25673  [redacted]  808u  IPv6  53687232  0t0  UDP *:27985
java  25673  [redacted]  809u  IPv6  53687222  0t0  UDP *:23808
java  25673  [redacted]  810u  IPv6  53687241  0t0  UDP *:1723
java  25673  [redacted]  811u  IPv6  53688629  0t0  UDP *:21399
java  25673  [redacted]  812u  IPv6  53687243  0t0  UDP *:12013
java  25673  [redacted]  813u  IPv6  53687245  0t0  UDP *:50506
java  25673  [redacted]  814u  IPv6  53687247  0t0  UDP *:38468
java  25673  [redacted]  815u  IPv6  53687249  0t0  UDP *:46785
java  25673  [redacted]  816u  IPv6  53687251  0t0  UDP *:11629
java  25673  [redacted]  817u  IPv6  53687254  0t0  UDP *:4663
java  25673  [redacted]  818u  IPv6  53687256  0t0  UDP *:5736
java  25673  [redacted]  819u  IPv6  53687258  0t0  UDP *:60244
java  25673  [redacted]  820u  IPv6  53687260  0t0  UDP *:15657
java  25673  [redacted]  821u  IPv6  53687262  0t0  UDP *:34269
java  25673  [redacted]  822u  IPv6  53687264  0t0  UDP *:10233
java  25673  [redacted]  823u  IPv6  53687266  0t0  UDP *:12452
java  25673  [redacted]  824u  IPv6  53687268  0t0  UDP *:18788
java  25673  [redacted]  825u  IPv6  53687286  0t0  UDP *:7832
java  25673  [redacted]  826u  IPv6  53687288  0t0  UDP *:58249
java  25673  [redacted]  827u  IPv6  53687291  0t0  UDP *:39332
java  25673  [redacted]  828u  IPv6  53687293  0t0  UDP *:38187
java  25673  [redacted]  829u  IPv6  53687295  0t0  UDP *:31435
java  25673  [redacted]  830u  IPv6  53687297  0t0  UDP *:34363
java  25673  [redacted]  831u  IPv6  53687299  0t0  UDP *:49518
java  25673  [redacted]  832u  IPv6  53687301  0t0  UDP *:15172
java  25673  [redacted]  833u  IPv6  53687306  0t0  UDP *:17763
java  25673  [redacted]  834u  IPv6  53687308  0t0  UDP *:37129
```

Occupy UDP or what?

Find connection leak

Fix connection leak

PROFIT!

I would like to

have a production monitoring system

that will allow me to see these

things early

Expectations

- very fast and easy access to recent events (recency is arbitrary)
- fast arbitrary queries on recent events
- real-time analytics, report generation
- easy to go back in time
- scalable queries across an entire dataset

Expectations

- pipelining, ability to inject additional processor
- arbitrary data format
- dumb client
- extensible server (plug-n-play)
- direct access to data, right in my repl

Expectations

- easy, extensible HTML / CSS / JS interface
- graph generation (real-time, pre-calculated)
- websockets for pushing stuff
- re-broadcast of incoming events for custom analytics
- support for multiple incoming channels

Expectations

- extensible alerting system (channels)
- extensible alerting system (rules)
- and (probably) many many more, which may be subsets of the mentioned ones in some way

let's draw a simple schema that will represent such a system

and try our best to make it not

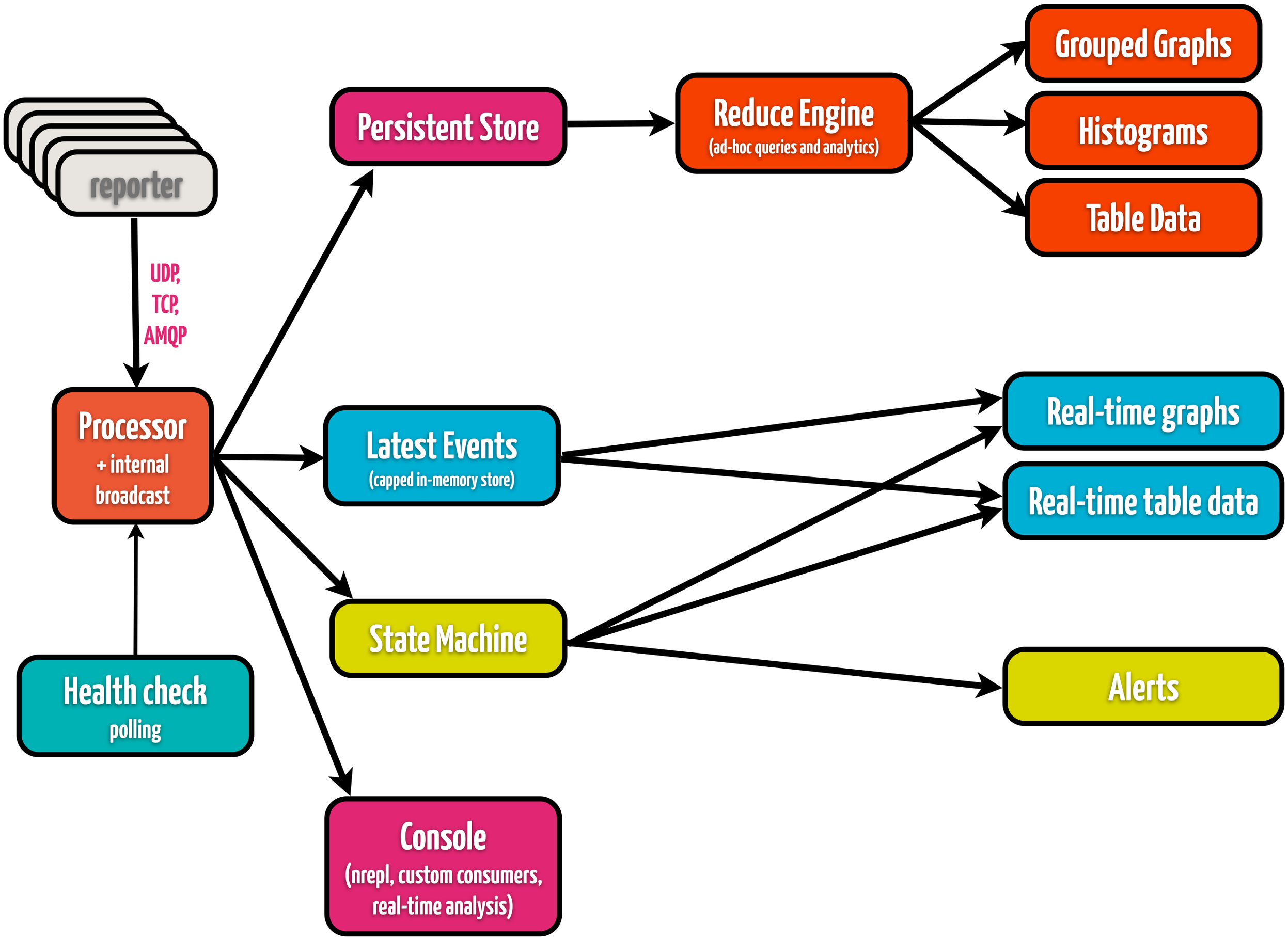
opinionated

opinionated stuff is about tight

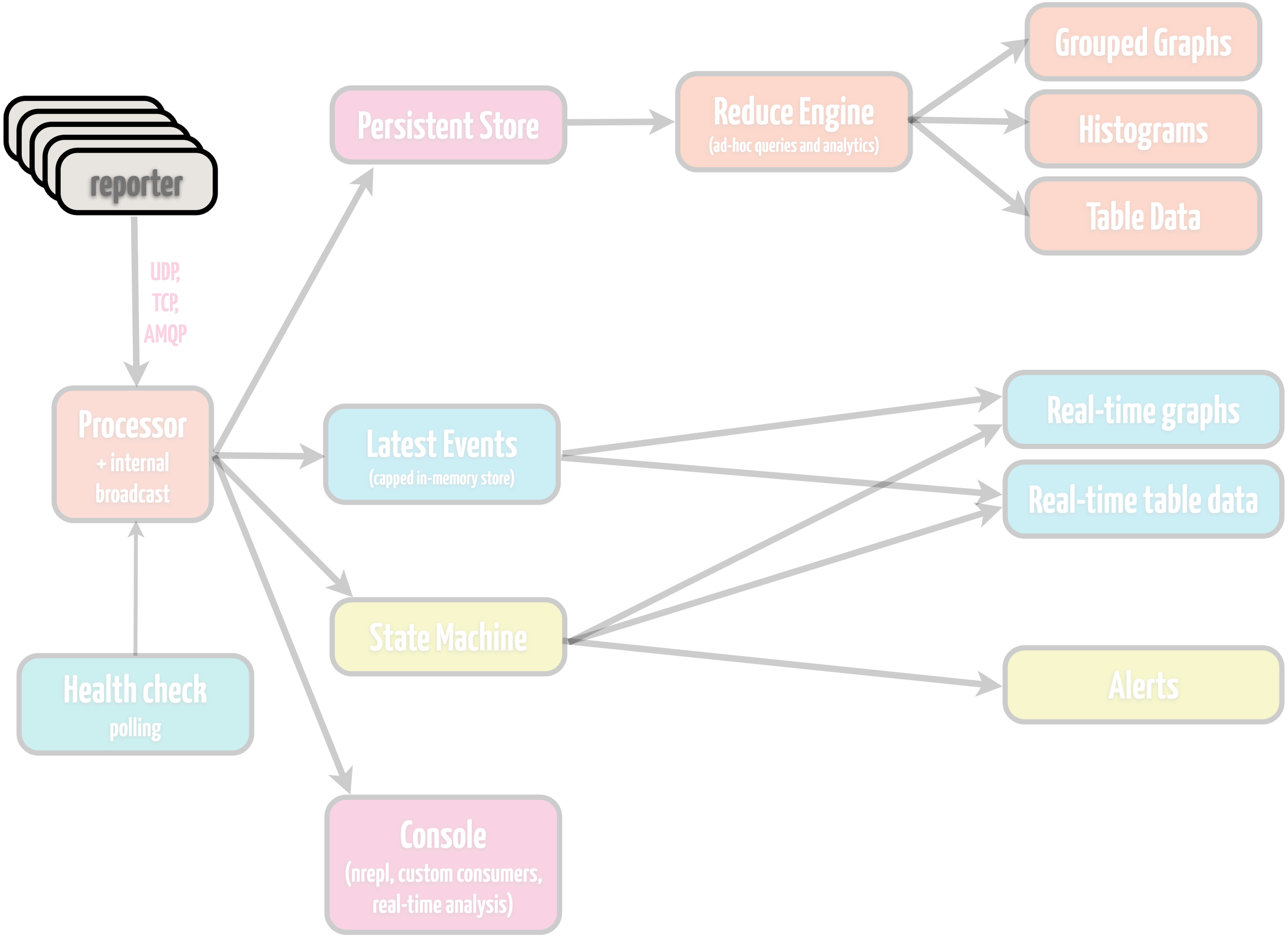
coupling

let's think of it in terms of a

toolchain, not a framework (tm)


reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Health check
polling
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
State Machine
Console
(nrepl, custom consumers,
real-time analysis)
Real-time graphs
Real-time table data
Alerts

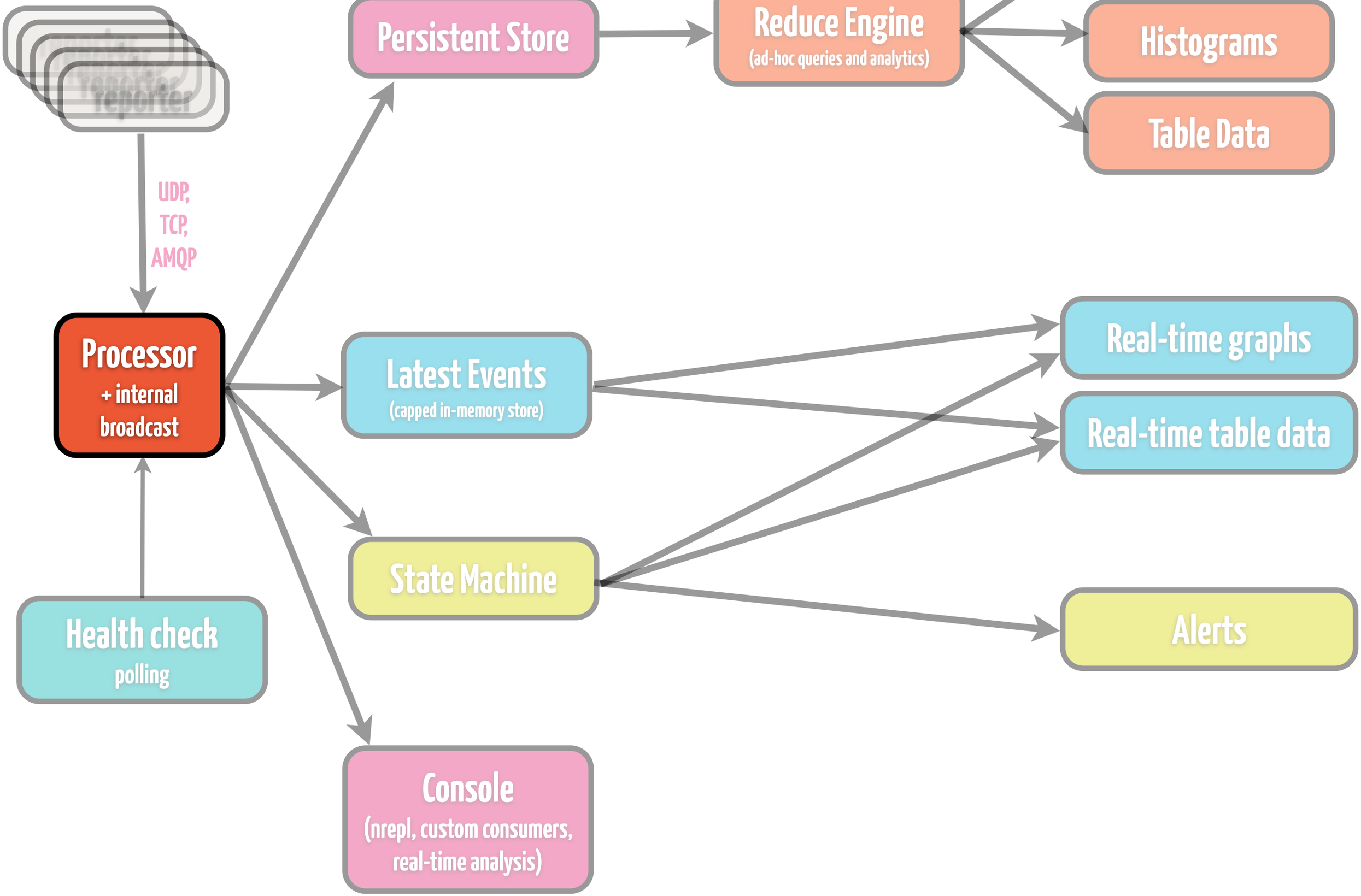
parts


reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
Real-time graphs
Real-time table data
State Machine
Alerts
Health check
polling
Console
(nrepl, custom consumers,
real-time analysis)

simple, < 20 LOC software that sends events to the server

message format

```
{
 ;; Md5 hash of an event, that uniquely identifies it
 :md5 "cd0e351d2eefdf0f79e0b55a0efe543b",
 ;; Arbitrary additional info
 :additional_info
   {:execution_time 0.5469999,
    :url “http://mysite.com/page0” },
 ;; Event Type identifier (404, exception, page load time)
 :type "page_load",
 ;; Dispatcher host name
 :hostname "dc0-web01",
 ;; Time when event was received
 :received_at ...,
 ;; Tags assigned to the event
 :tags ["metrics" "performance"]
}
```


reporter
reporter
reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Health check
polling
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
State Machine
Console
(nrepl, custom consumers,
real-time analysis)
Real-time graphs
Real-time table data
Alerts

In nutshell, it's a pipeline.

Basic properties:

- validate the event
- classify / re-classify event
- run processor based on current event type
- add calculated params
- distribute events internally

Classification

- remember the example with 404 that looked like an exception?
- makes events harvesting and further processing easier
- client will remain dumb, no need to redeploy on type changes
- useful for sub-typing, e.g. page_load could become slow_page_load etc.
- depends on your use case

message format

```
{
 :additional_info
   {:execution_time 0.5469999,
    :url “http://mysite.com/page0” },

 :hostname "dc0-web01",
 :received_at ...,
 :tags ["metrics" "performance"]
}
```

message format

```
{
 :additional_info
   {:execution_time 0.5469999,
    :url “http://mysite.com/page0” },
  :type "page_load",
  :hostname "dc0-web01",
  :received_at ...,
  :tags ["metrics" "performance"]
}
```

message format

```
{
 :additional_info
   {:execution_time 0.5469999,
    :url “http://mysite.com/” },

 :hostname "dc0-web01",
 :received_at ...,
 :tags ["metrics" "performance"]
}
```

message format

```
{
 :additional_info
   {:execution_time 0.5469999,
    :url “http://mysite.com/” },
  :type "landing_page_load",
  :hostname "dc0-web01",
  :received_at ...,
  :tags ["metrics" "performance"]
}
```

Calculated params

- internal example is md5 hash, used to ease finding unique events
- could be used for grouping within a type
- percent of a whole calculation max heap: 4gb, current: 2gb => 50% of a whole
- add flags for further pipeline modules

message format

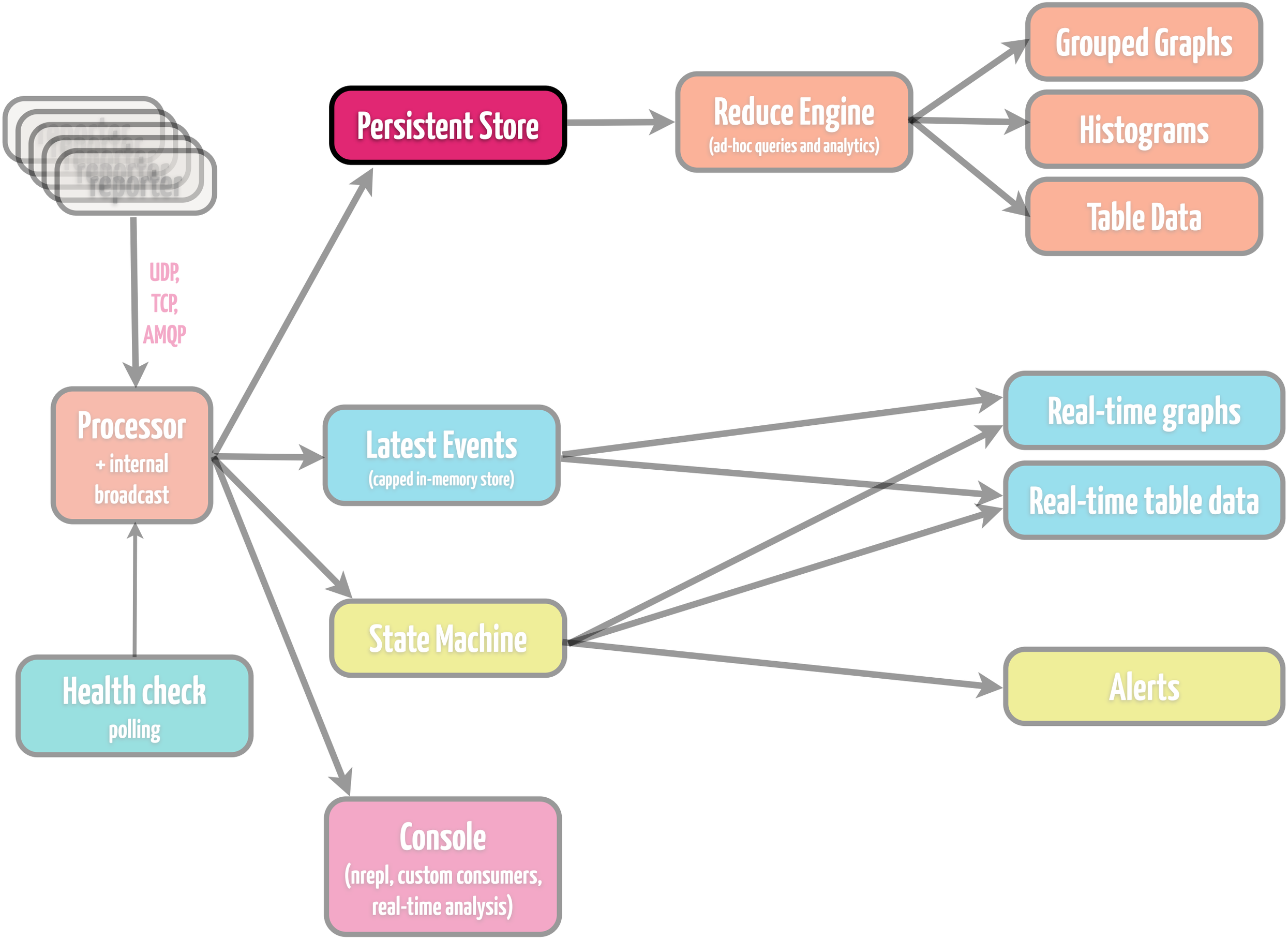
```
{
  :additional_info
    {:backtrace “...” },
  :type "backtrace",
  :hostname "dc0-web01",
  :received_at ...,
  :tags ["metrics" "performance"]
}
```

message format

```
{
 :additional_info
   {:backtrace “...” },
  :md5 (calculate-md5 (:backtrace “...”))
  :type "backtrace",
  :hostname "dc0-web01",
  :received_at ...,
  :tags ["metrics" "performance"]
}
```

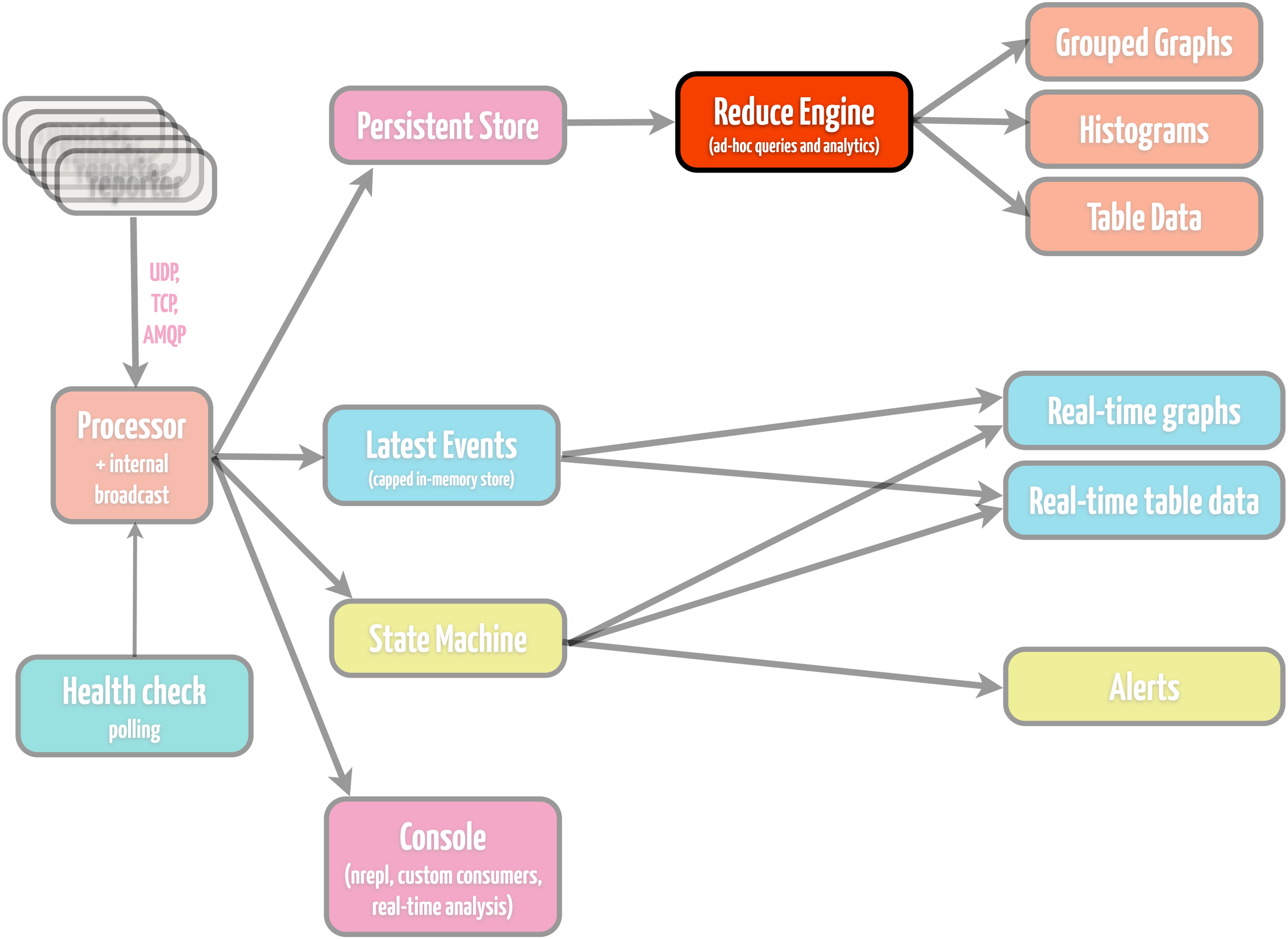
Internal broadcast

- when production system is running, you may want to attach to it in real-time and calculate message rate based on certain rules
- throw events to persistence layer, state machine, arbitrary listeners


reporter
reporter
reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
Real-time graphs
Real-time table data
State Machine
Alerts
Health check
polling
Console
(nrepl, custom consumers,
real-time analysis)

Persistent Store

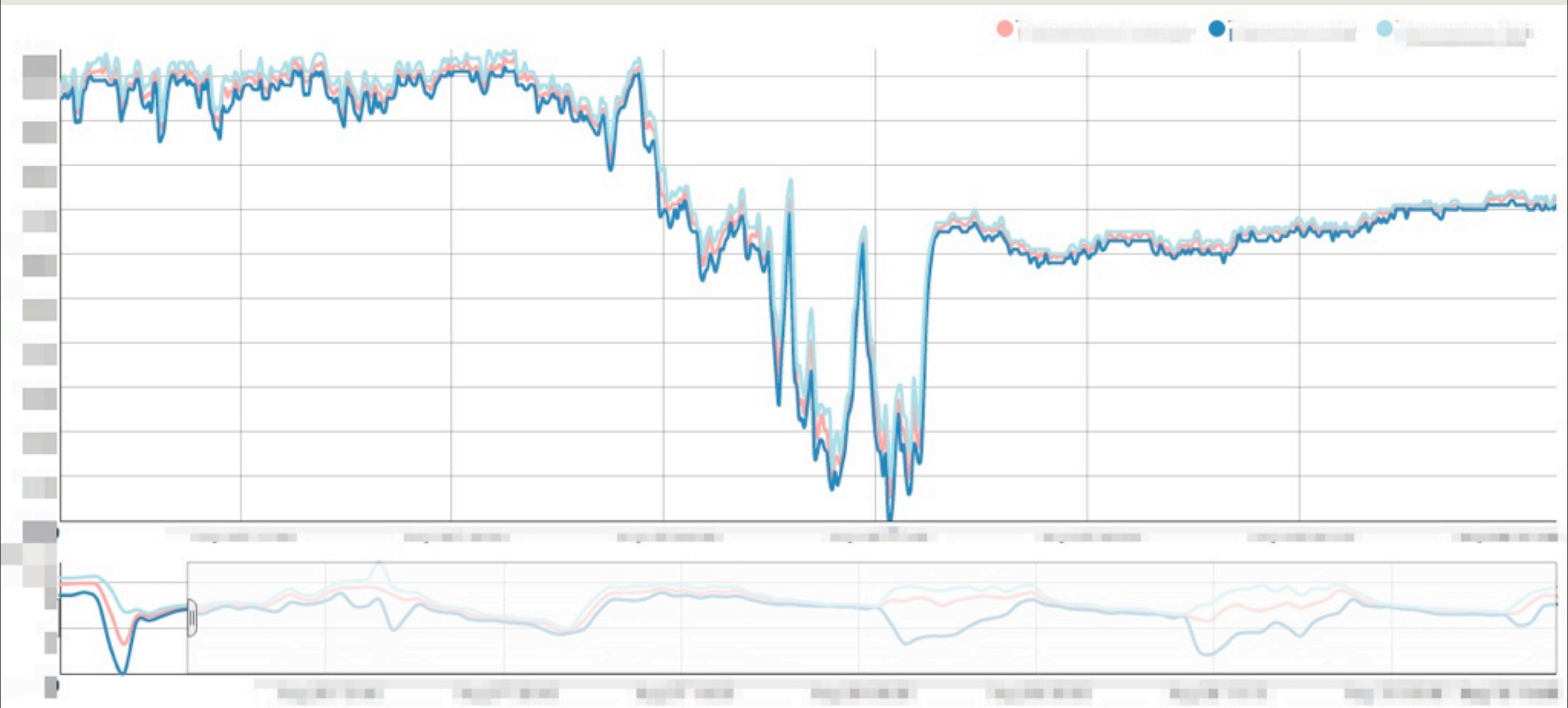
- highly depends on your throughput
- consider something lightweight and scalable
- if you generate fairly large amount of data, don't try to handle processing it by means of your store
- keep it simple
- think about Cassandra or HDFS, depending on how you plan to access data


reporter
reporter
reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
Real-time graphs
Real-time table data
State Machine
Alerts
Health check
polling
Console
(nrepl, custom consumers,
real-time analysis)

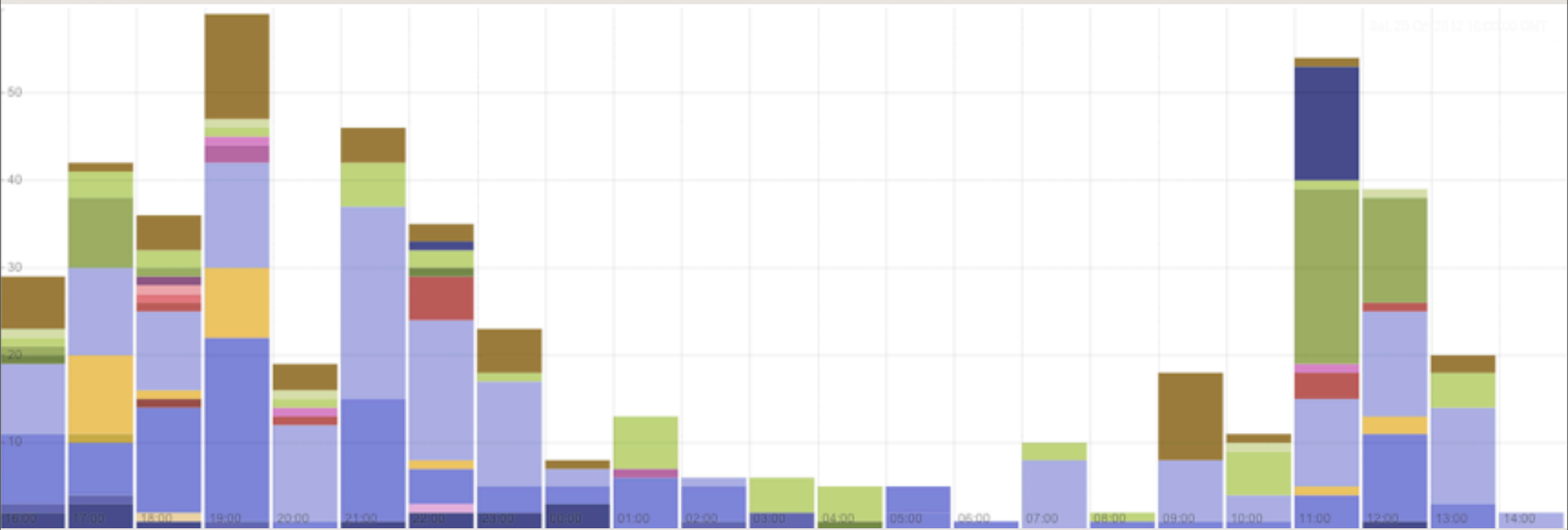
Reduce Engine
(ad-hoc queries and analytics)

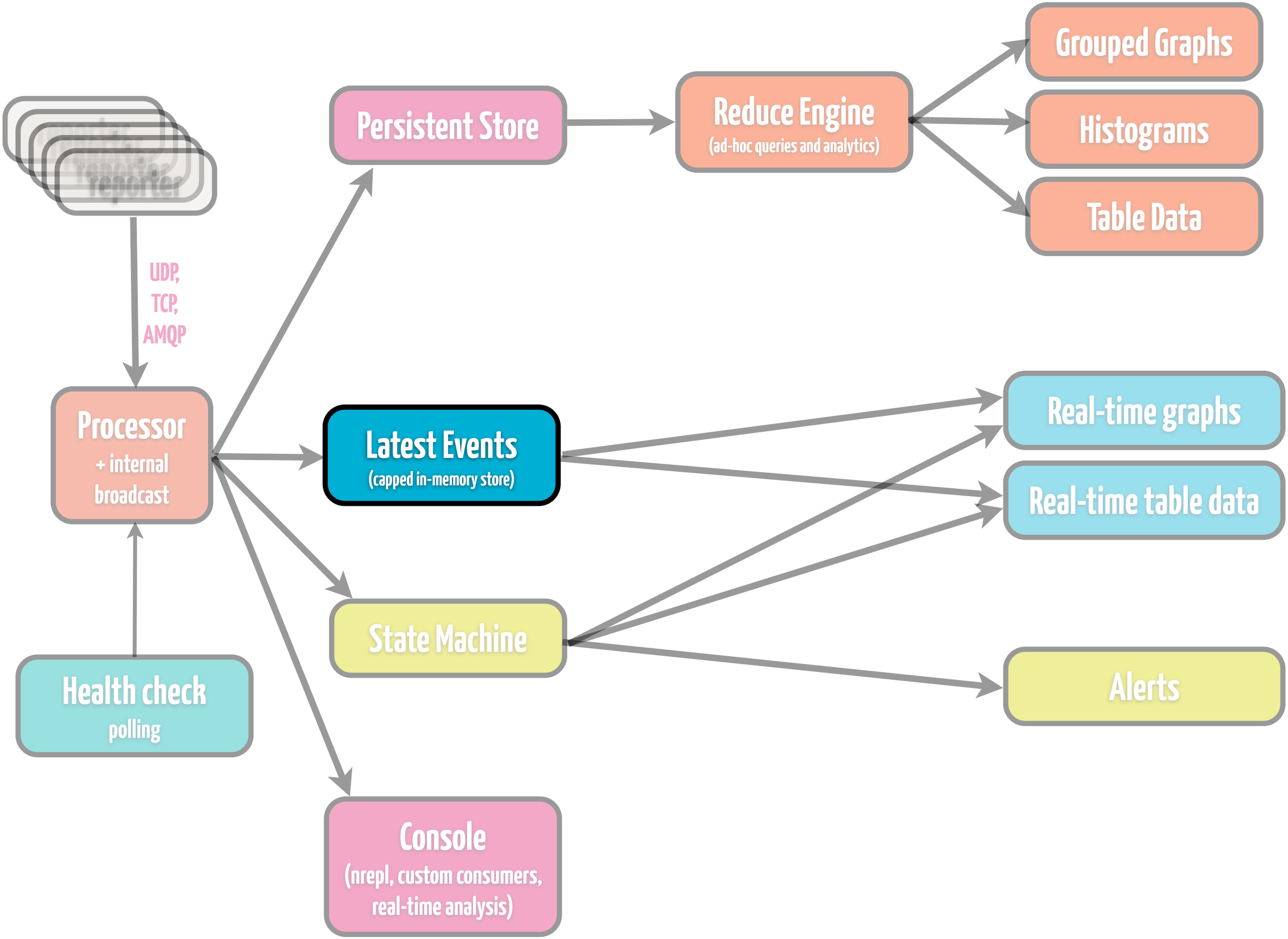
- sometimes you just don't know in real-time
- a new problem pops up, and you have to re-play stuff from a year ago
- find trends, that are simply not visible on a day/week/month scale
- most likely going to be something hadoop-based (YMMV)
- cascalog works quite well for us
- ad-hoc queries
- scheduled analytics
- correlating metrics (hard to do in real time)

Graphs



Graphs


50
40
30
20
10
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
00:00
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00

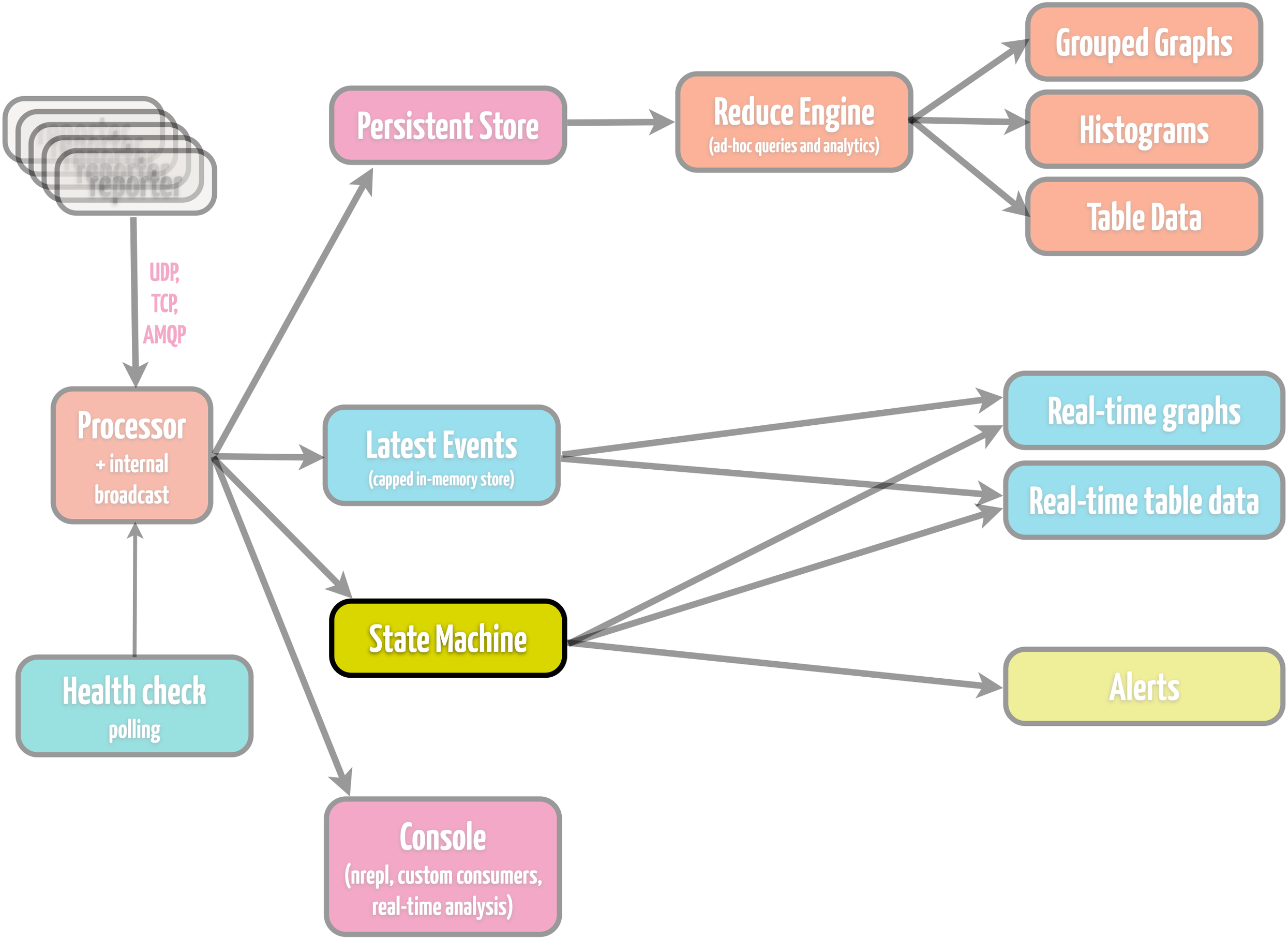

reporter
reporter
reporter
UDP,
TCP,
AMQP
Processor
+ internal
broadcast
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Latest Events
(capped in-memory store)
Real-time graphs
Real-time table data
State Machine
Alerts
Health check
polling
Console
(nrepl, custom consumers,
real-time analysis)

Latest Events
(capped in-memory store)

- that's basically an event cache.
- ring-buffer, whose size is limited by rules
- either time-bound (last hour, 6 hours, 24 hours)
- or size-bound (max 10K events)
- Riemann's indexes are Cliff Click NonBlockingHashMap backed
- you can go with (atom {:keyword (ref [])}) for starters
- fire up an nrepl server on collector
- have complete clojure access to your live data

- very queriable (100K+ events in collection are processed blazingly fast)
- optimize, calculate real-time indexes when needed
- for the most complex queries, that turn out to be a bottleneck, use state machine
- use for anything: stats, graphs, table data, dashboards. It's also very, very easy!

- very queriable (100K+ events in collection are processed blazingly fast)
- optimize, calculate real-time indexes when needed
- for the most complex queries, that turn out to be a bottleneck, use state machine
- use for anything: stats, graphs, table data, dashboards. It's also very, very easy!


reporter
reporter
reporter
UDP,
TCP,
AMQP
Persistent Store
Reduce Engine
(ad-hoc queries and analytics)
Grouped Graphs
Histograms
Table Data
Processor
+ internal
broadcast
Latest Events
(capped in-memory store)
Real-time graphs
Real-time table data
State Machine
Alerts
Health check
polling
Console
(nrepl, custom consumers,
real-time analysis)

State Machine

- counters, gauges, meters
- triggers for events and alarms
- reducers of any type, actually

- if the amount of events of type `T` with md5 `M` (or any md5) exceeds 10, escalate the issue (send alert)
- if we received more than `N` events of type `T` in 1 minute, send alert
- if reponse of page exceeds `X` ms, send alert

State Machine

- You can measure metrics based on any arbitrary rule and make any arbitrary callback, basically.

UDP,
TCP,
AMQP

Persistent Store

Reduce Engine
(ad-hoc queries and analytics)

Grouped Graphs

Histograms

Table Data

Processor
+ internal
broadcast

Latest Events
(capped in-memory store)

Real-time graphs

Real-time table data

State Machine

Alerts

Health check
polling

Console
(nrepl, custom consumers,
real-time analysis)

Console
(nrepl, custom consumers, real-time analysis)

- build processing pipelines in runtime
- run nrepl
- run queries, get necessary information

it's important to

know

what your application does right now

make it easy to

notice failures

to identify

behavior oddities

for that, again

you have to run a lot of ad-hoc queries

make system

more stable

by noticing things that happen

often

but irregular

see what code

doesn't reveal

if the host is up, is "up" what you expect by "up"?

collect

everything

storage is cheap

YMMV if

if you operate on

twitter/facebook

scale, where you need to

seek more global approaches

let your system

talk to you

extract business metrics from tech data

improve business metrics by tech data

make it easy to see what your users see

plan your growth

probably quite hard to do it without knowing your current corellation between performance and number of servers, and extracting trends from historical data

DYI

I've set it already quite a few times, but I won't get tired of saying it: DO IT YOURSELF.
it's possible to find a tool that does most of things for you, but you may not find your own patterns with it. Use libs for visualization, persistency, distribution, transport, processing, but try to avoid coupling. If it's hard to get data in, or get data out of it, good idea to avoid it.

also, do you know about

Clojure


Wait for it...
Citytv

Werkz

It just werkz

A growing collection of open source Clojure libraries that support multiple Clojure & JDK versions, licensed under the EPL, target Clojure 1.3+.


How awesome is that?
Citytv


That's All Folks!

CLOJURE WERKZ

/ ifesdjeen

/ clojurewerkz

Also, you should follow

/ michaelklishin