What I’ve learned doing chaos at Netflix

Lorin Hochstein (@lhochstein)

Chaos engineering should not be endorsed by the ICSE community. Accepting a workshop pretty much endorses the topic.

-- Reviewer, rejected ICSE'16 chaos workshop proposal

Some context about Netflix

We care about availability

Whoops, something went wrong…

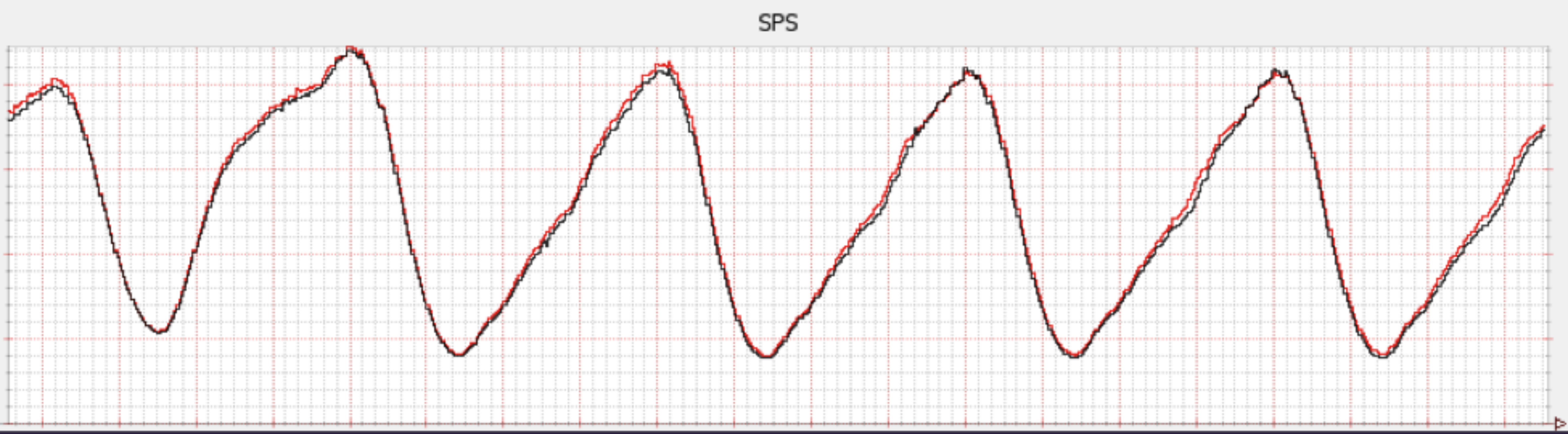
Netflix Streaming Error

We’re having trouble playing this title right now. Please try again later or select a different title.

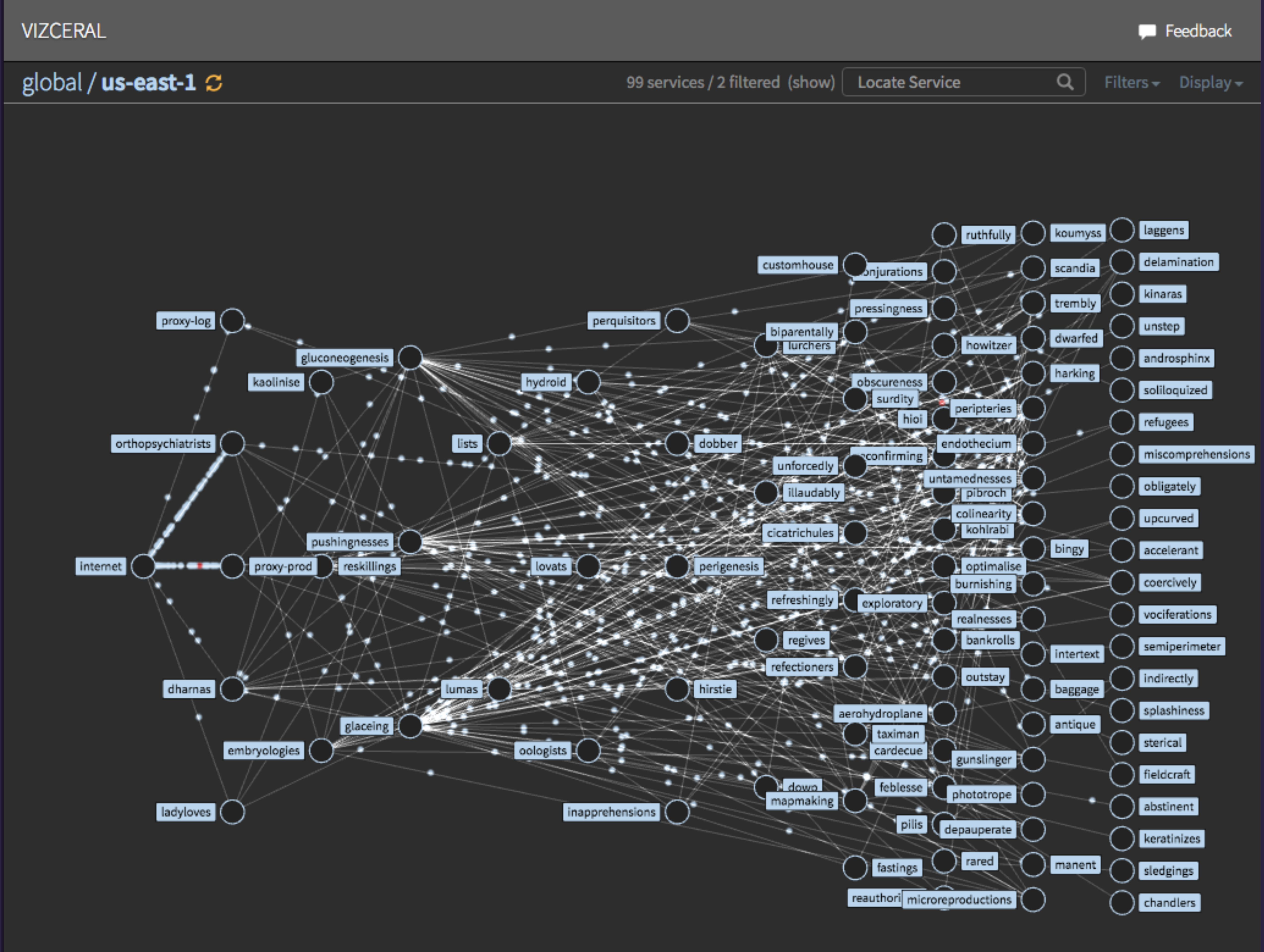
SPS: Stream starts Per Second

Number of people who hit the "play" button and successfully started

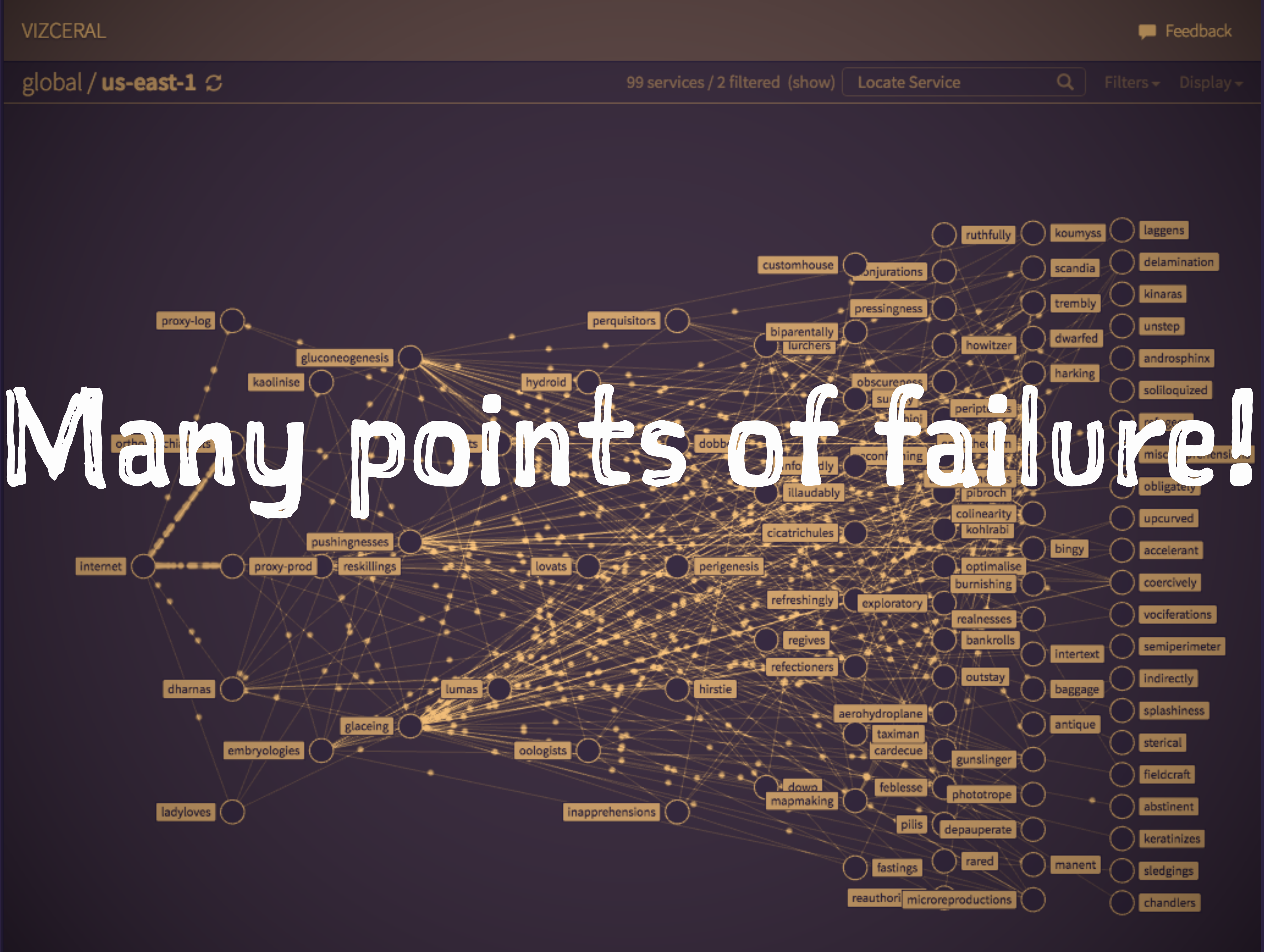
99.95%


SPS

Microservice architecture


VIZCERAL
Feedback
global / us-east-1
99 services / 2 filtered (show)
Locate Service
Filters
Display
internet
proxy-log
orthopsychiatrists
dharnas
ladyloves
kaolinise
proxy-prod
reskillings
embryologies
gluconeogenesis
pushingnesses
glaceing
hydroid
lists
lovats
lumas
oologists
perquisitors
dobber
perigenesis
hirstie
inapprehensions
customhouse
biparentally
lurchers
unforcedly
illaudably
cicatrichules
refreshingly
regives
refectioners
dowp
mapmaking
njurations
pressingness
obscureness
surdity
hioi
confirming
exploratory
aerohydroplane
taximan
cardecue
feblesse
pilis
fastings
reauthori
microreproductions
ruthfully
howitzer
peripteries
endothecium
untamednesses
pibroch
colinearity
kohlrabi
optimalise
burnishing
realnesses
bankrolls
outstay
gunslinger
phototrope
depauperate
rared
koumyss
scandia
trembly
dwarfed
harking
bingy
intertext
baggage
antique
manent
laggens
delamination
kinaras
unstep
androsphinx
soliloquized
refugees
miscomprehensions
obligately
upcurved
accelerant
coercively
vociferations
semiperimeter
indirectly
splashiness
sterical
fieldcraft
abstinent
keratinizes
sledgings
chandlers

Many points of failure!



A play in three acts

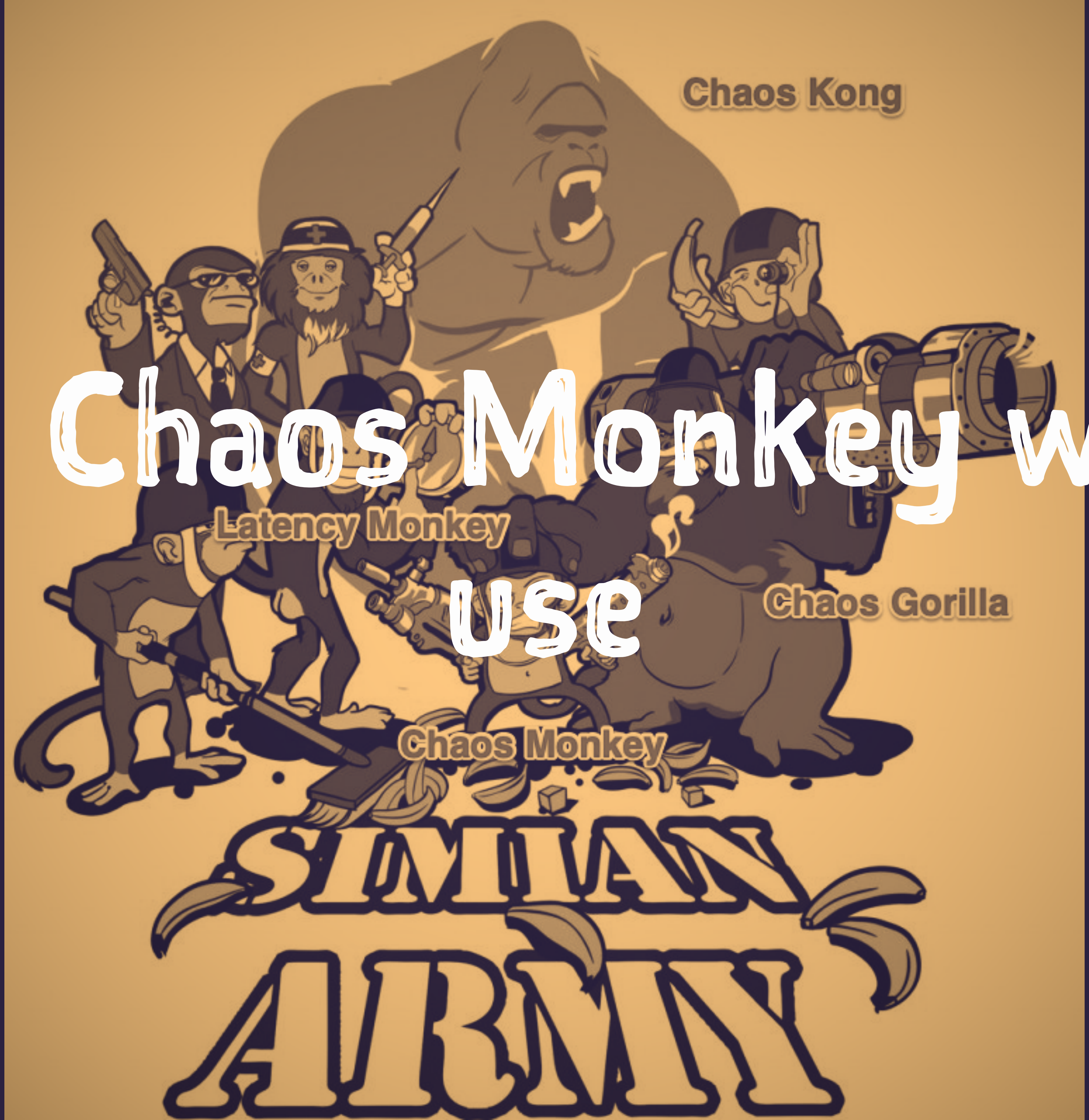
- Act I: Chaos at Netflix when I got there
- Act II: Chaos as experimentation
- Act III: Lessons learned

Act I: Chaos at Netflix when I got there

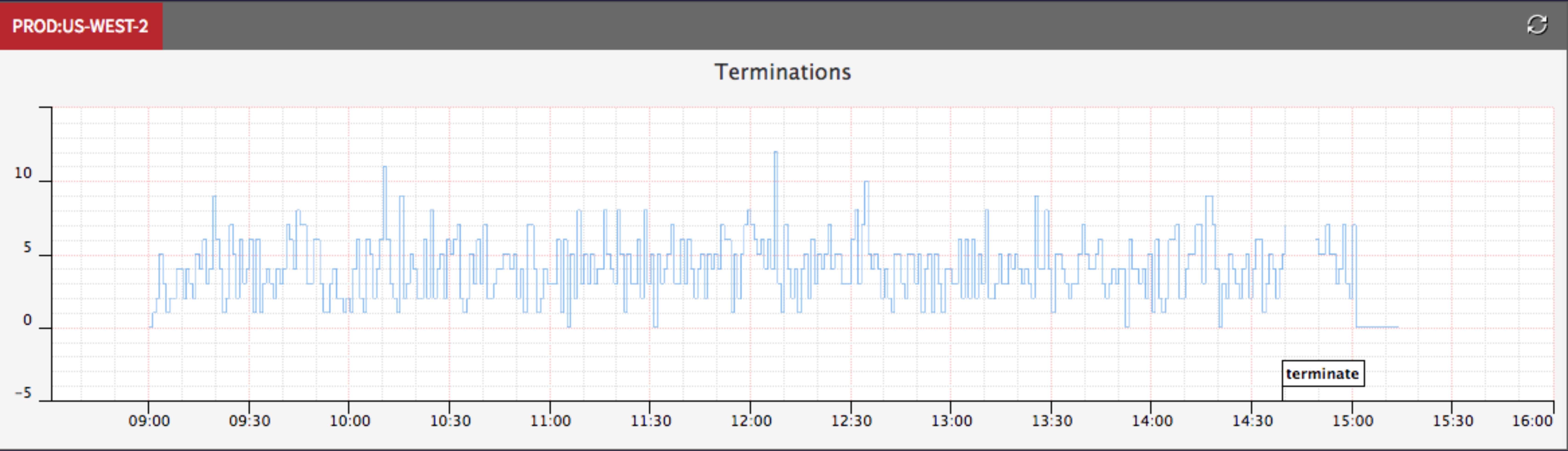
SIMIAN
ARMY

Chaos Kong
Latency Monkey
Chaos Gorilla
Chaos Monkey
SIMIAN
ARMY

Only Chaos Monkey was in use



Chaos Monkey randomly terminates instances in production


PROD:US-WEST-2
Terminations
10
5
0
-5
09:00
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
terminate

Chaos Monkey had already exposed single-instance termination weaknesses

Latency monkey was too dangerous

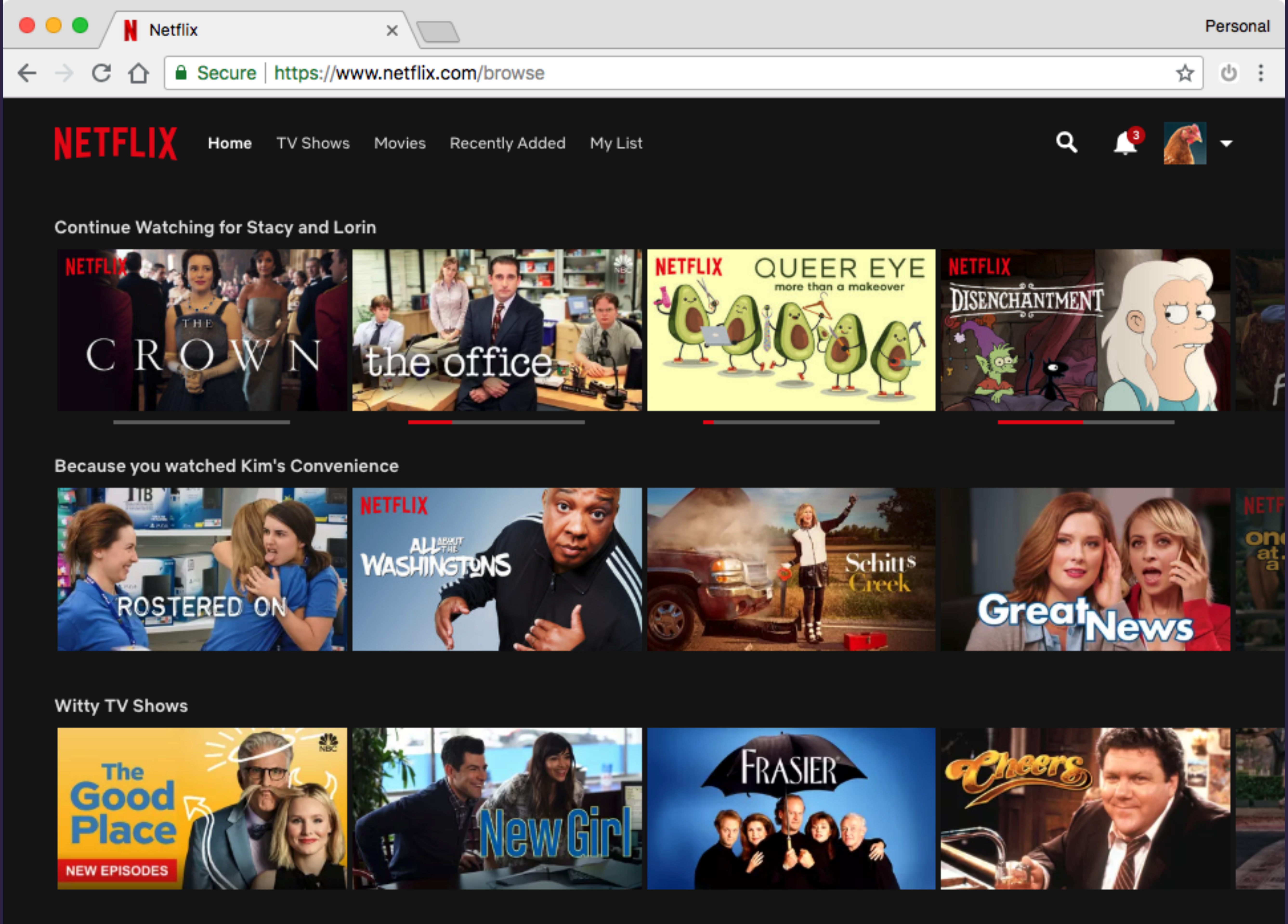
FIT: Failure Injection Testing

Inject failure or latency at "injection points" in code

Example injection point: remote procedure call

Failures are scoped, not random

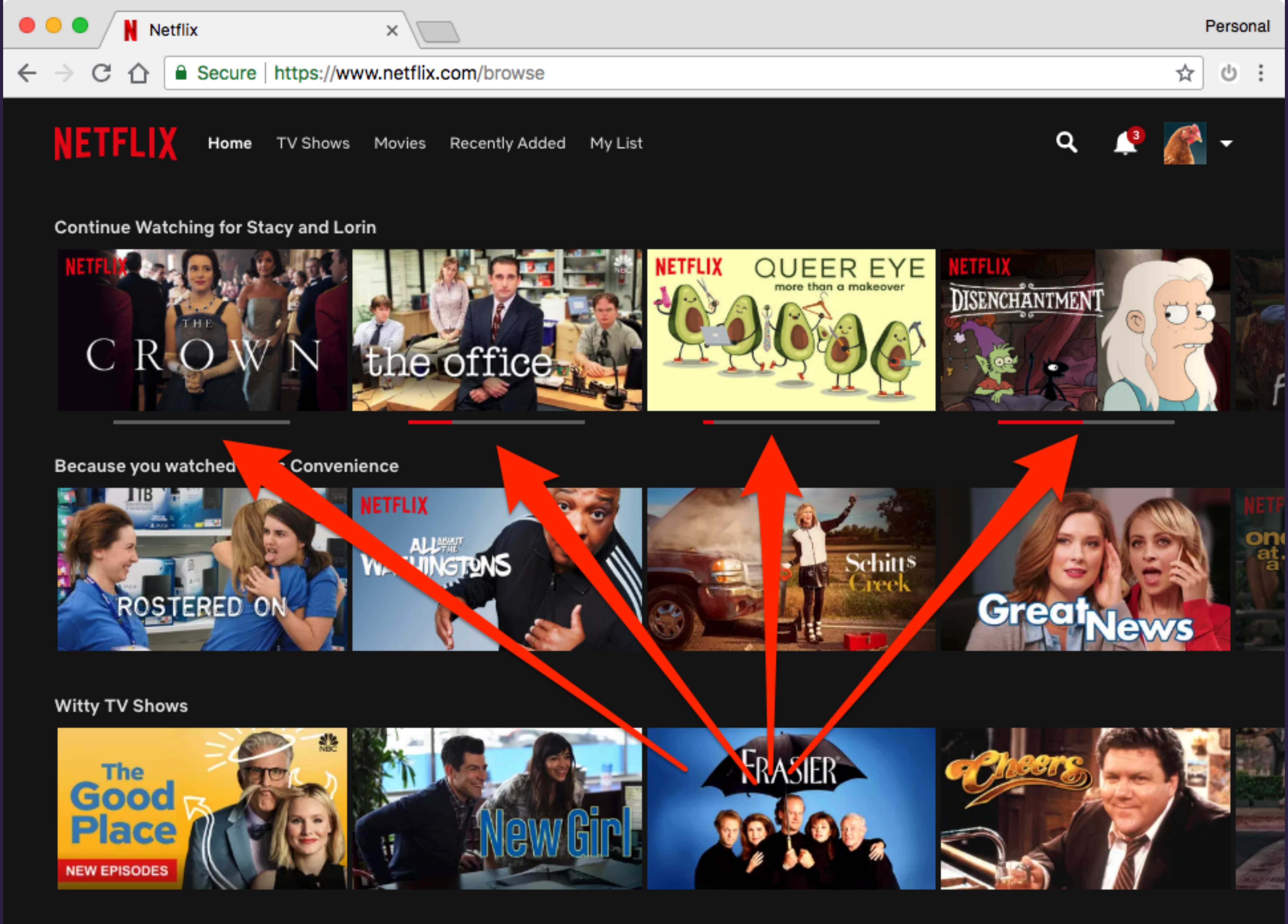
Example: Is the bookmarks service critical?



Continue Watching for Stacy and Lorin

Because you watched Kim's Convenience

Witty TV Shows



Continue Watching for Stacy and Lorin

Because you watched Convenience

Witty TV Shows

```
-------        -------------
| api | -----> | bookmarks |
-------        -------------
```

Fail calls from the "api" service to the "bookmarks" service for account "123456"

```
-------        -------------
| api | --×--> | bookmarks |
-------        -------------
```

Many service failures look like errors or latency

Great for testing with a single device

Some problems only appear when many calls fail

503 Service Unavailable

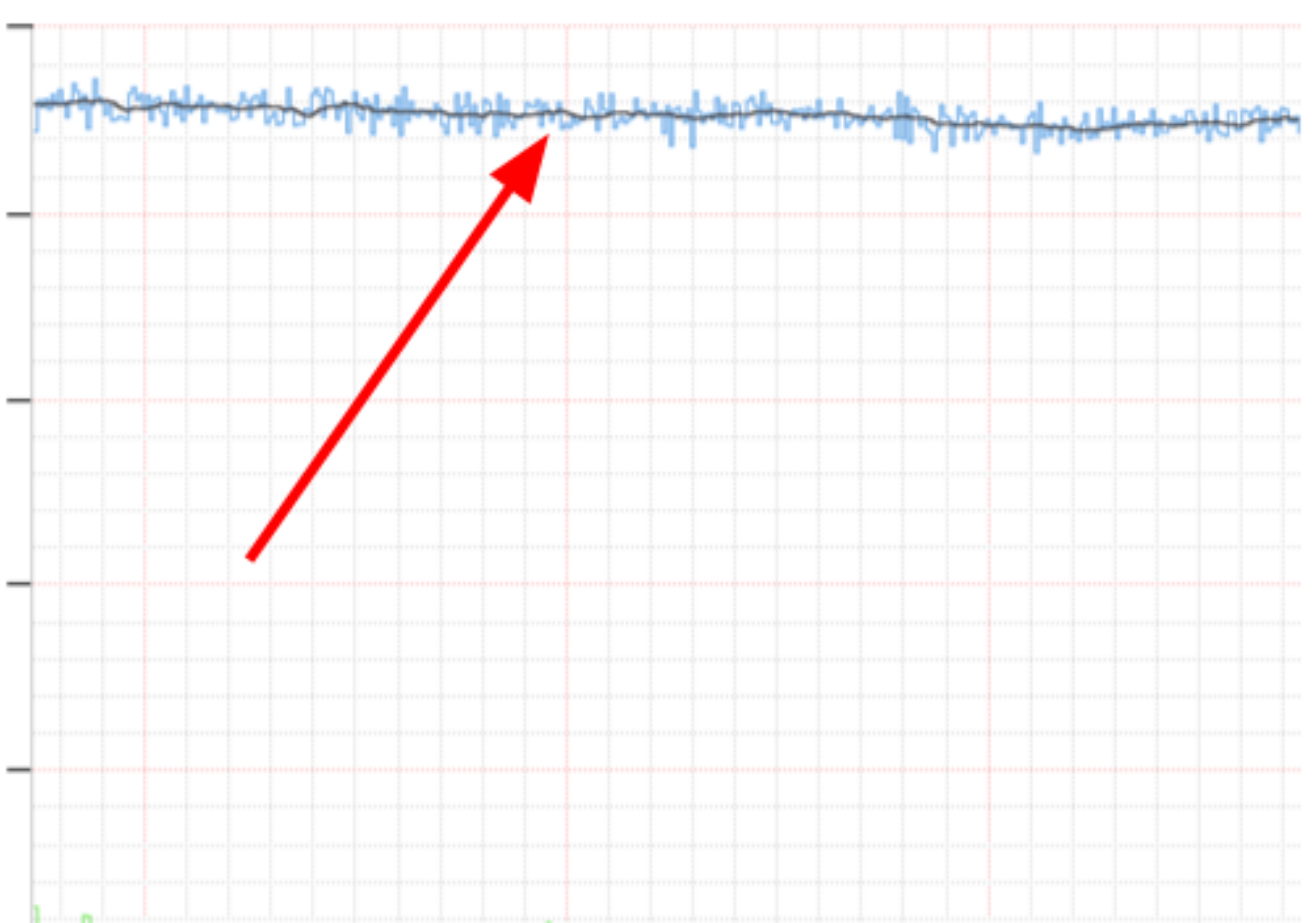
FIT supported large-scale failure injection

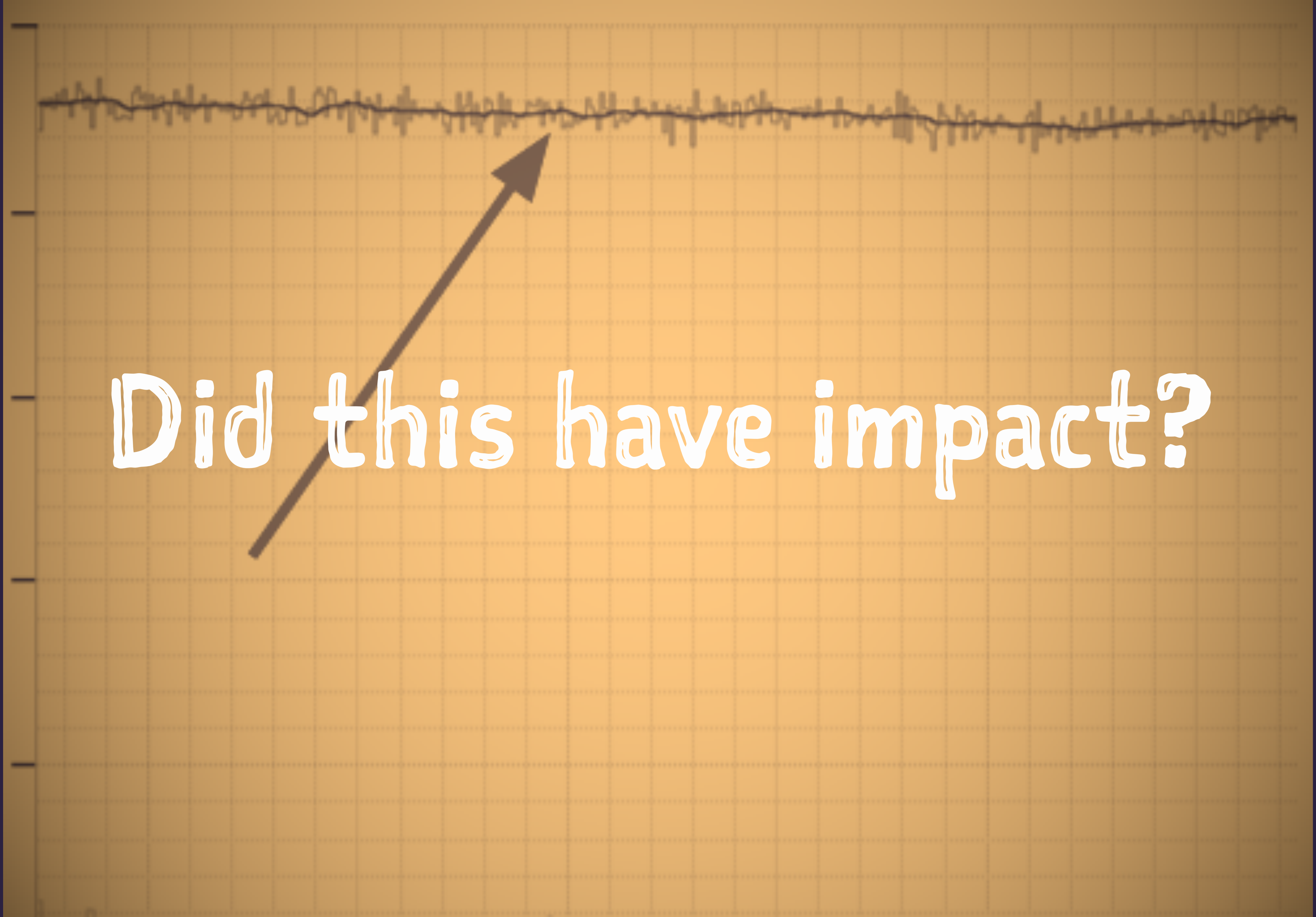
Example: Inject failure for 10% of customer traffic

How much should you inject?

Too much: unnecessary customer pain

Too little: can't tell if there's a vulnerability





Did this have impact?

Act II: Chaos as experimentation

ChAP

Chaos Automation Platform

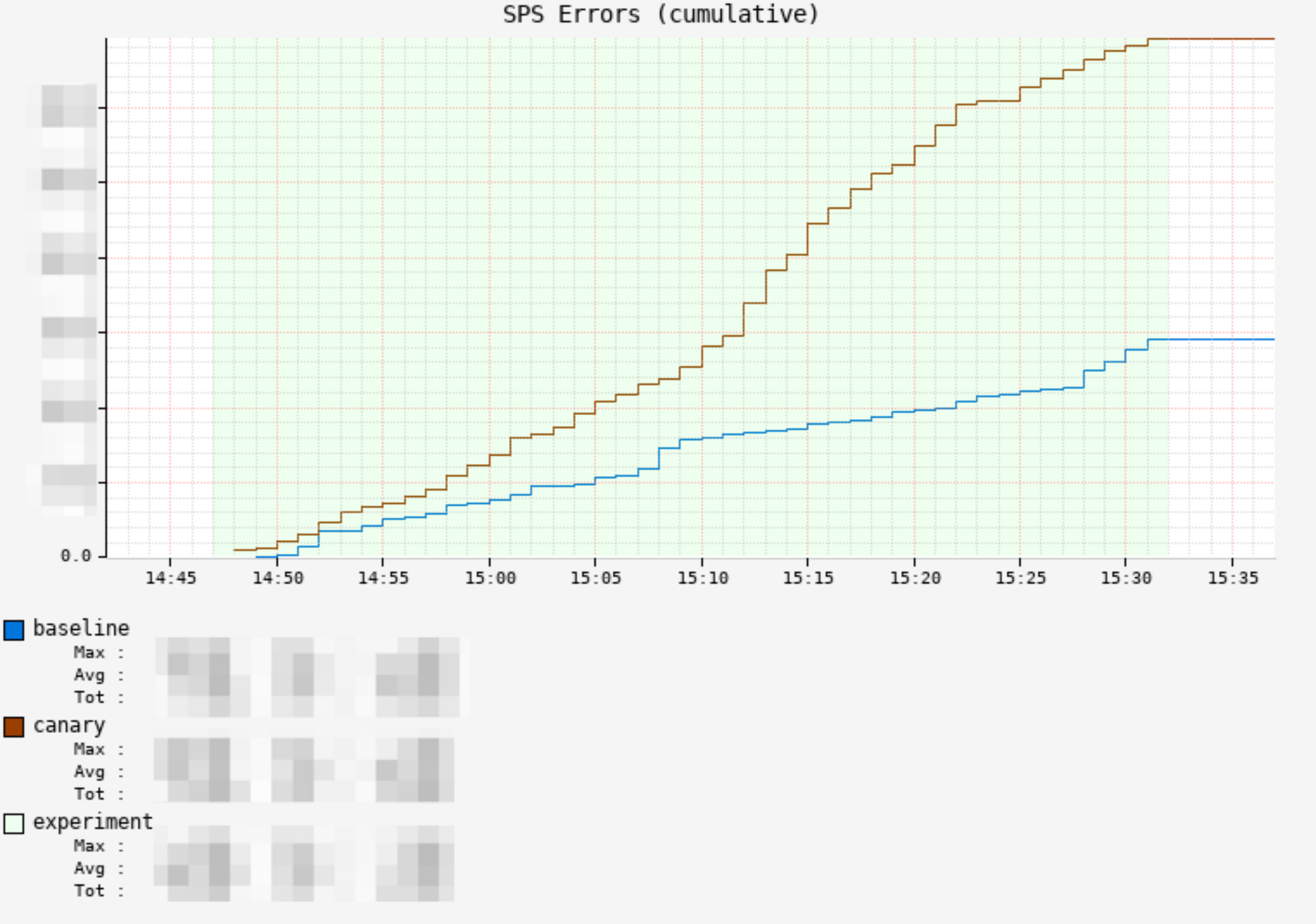
Want: clear signal if failure injection having negative impact...

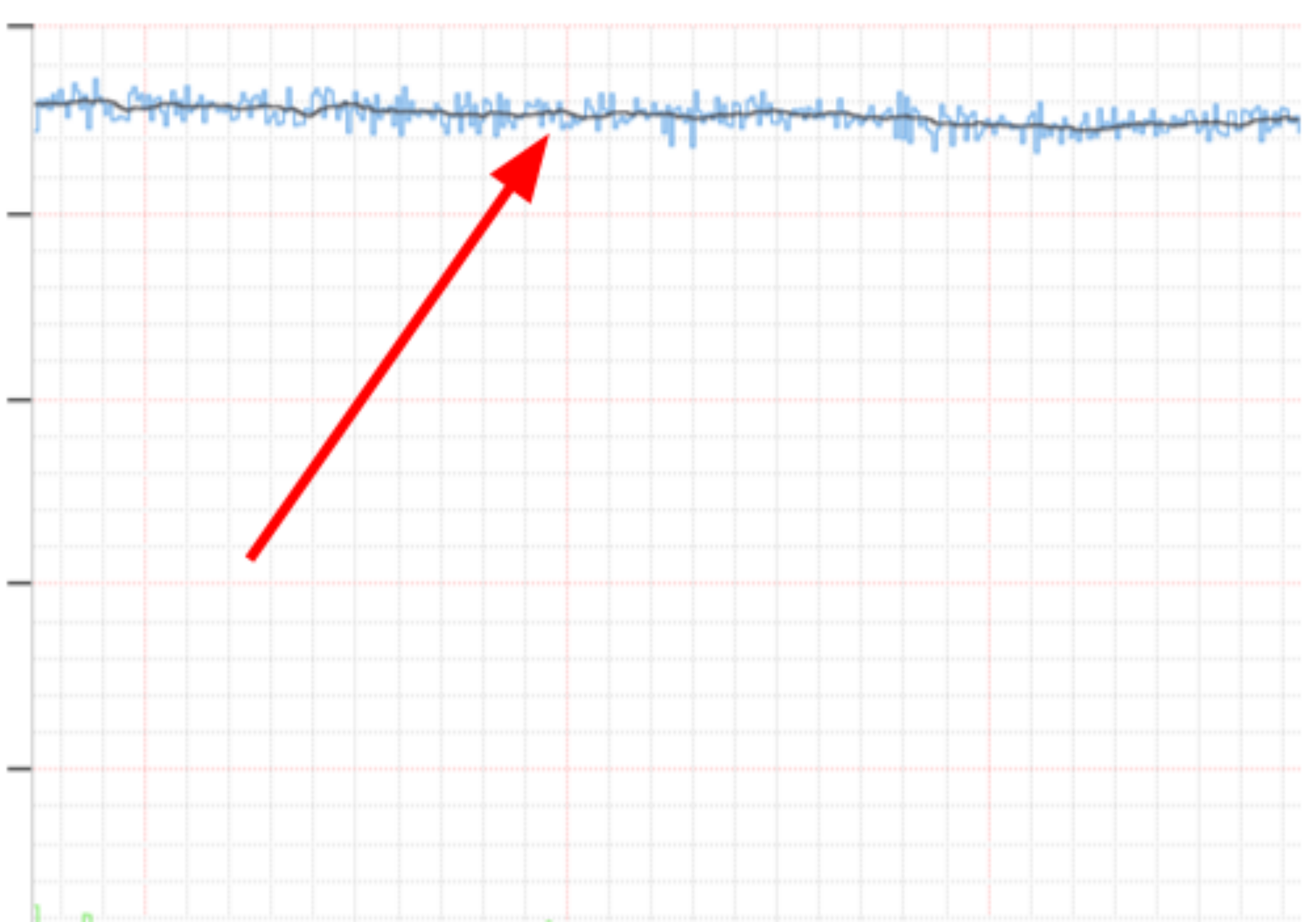
...on customers...

...and on services

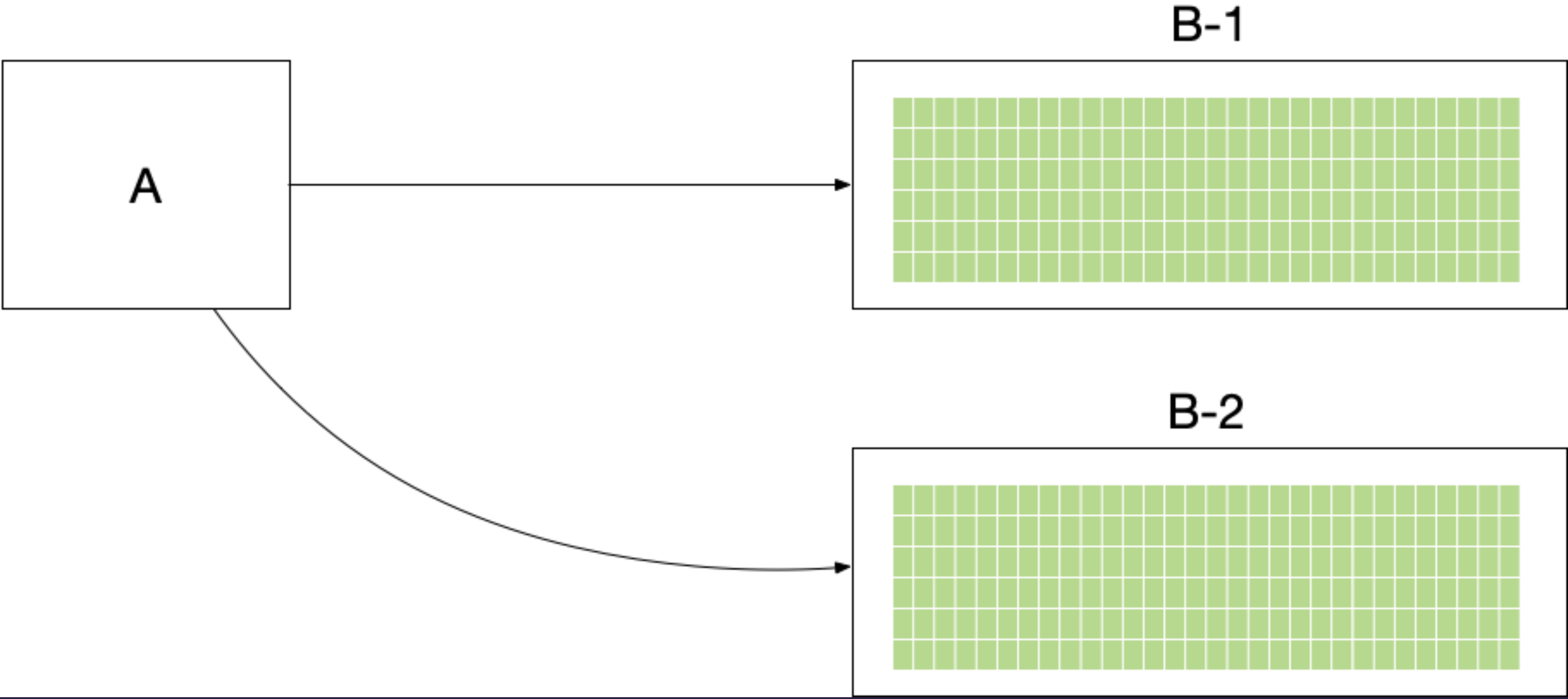
Big idea: stickiness

Failure injection sessions are sticky to users

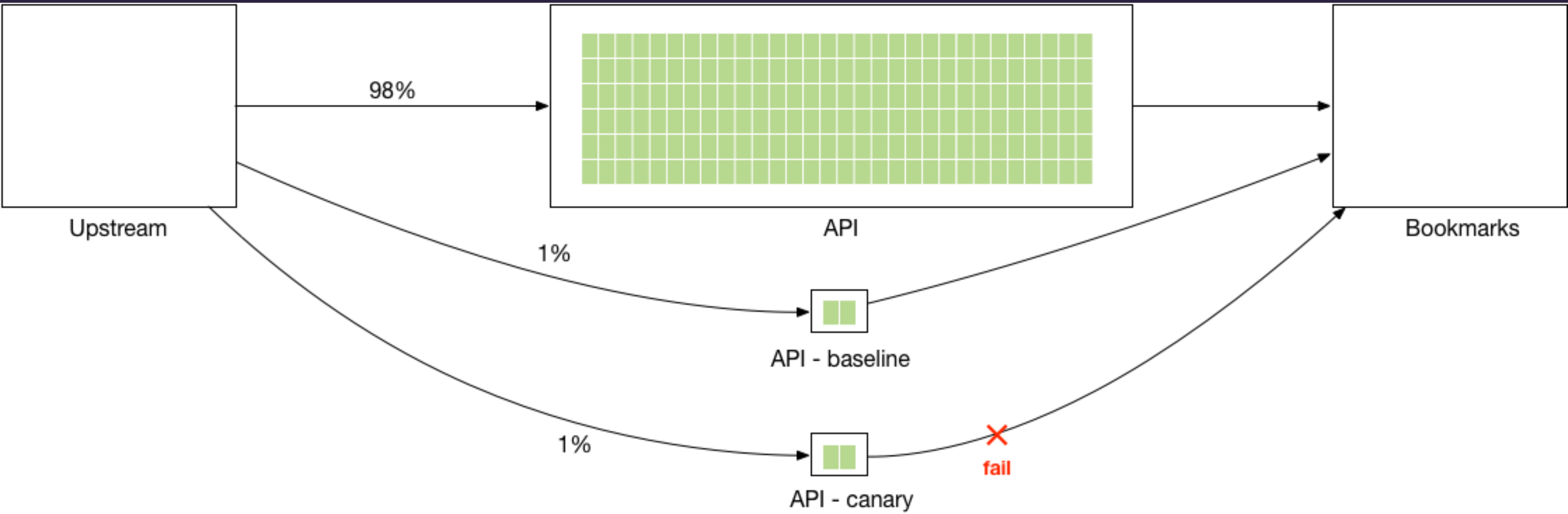

SPS Errors (cumulative)
0.0
14:45
14:50
14:55
15:00
15:05
15:10
15:15
15:20
15:25
15:30
15:35
🟦 baseline
Max :
Avg :
Tot :
🟫 canary
Max :
Avg :
Tot :
□ experiment
Max :
Avg :
Tot :

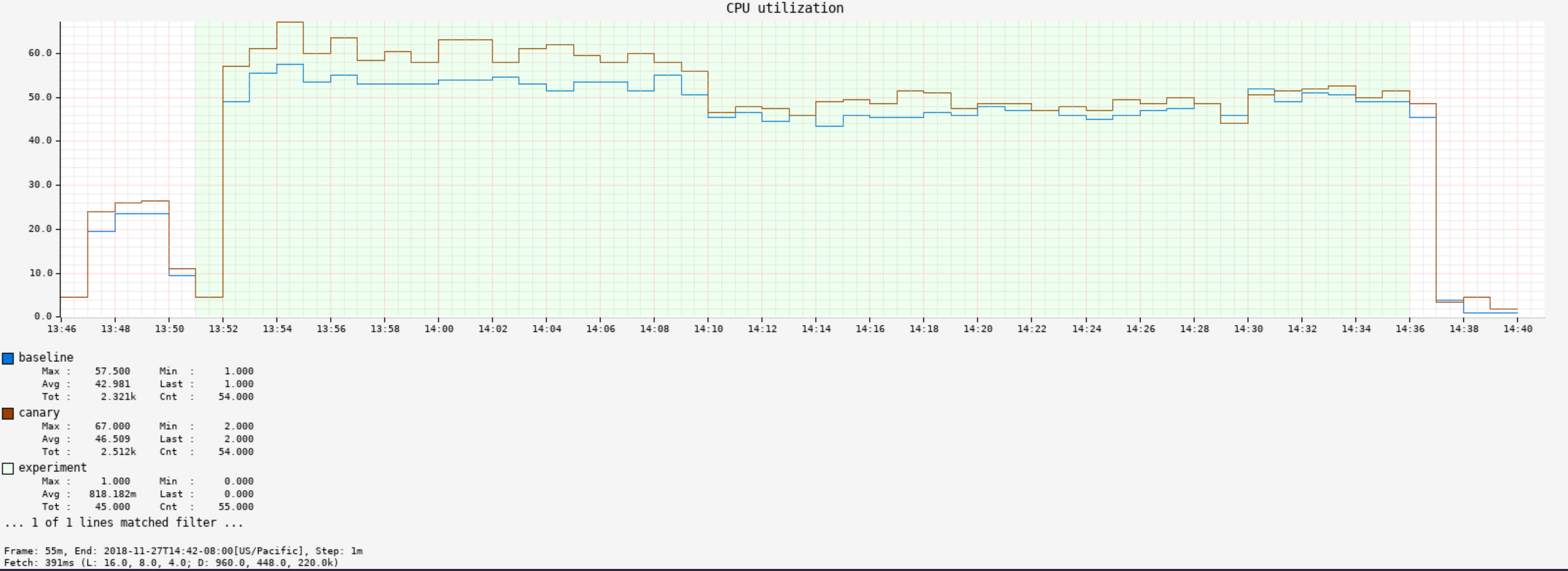


Failure injection sessions are sticky to clusters


B-1
A
B-2

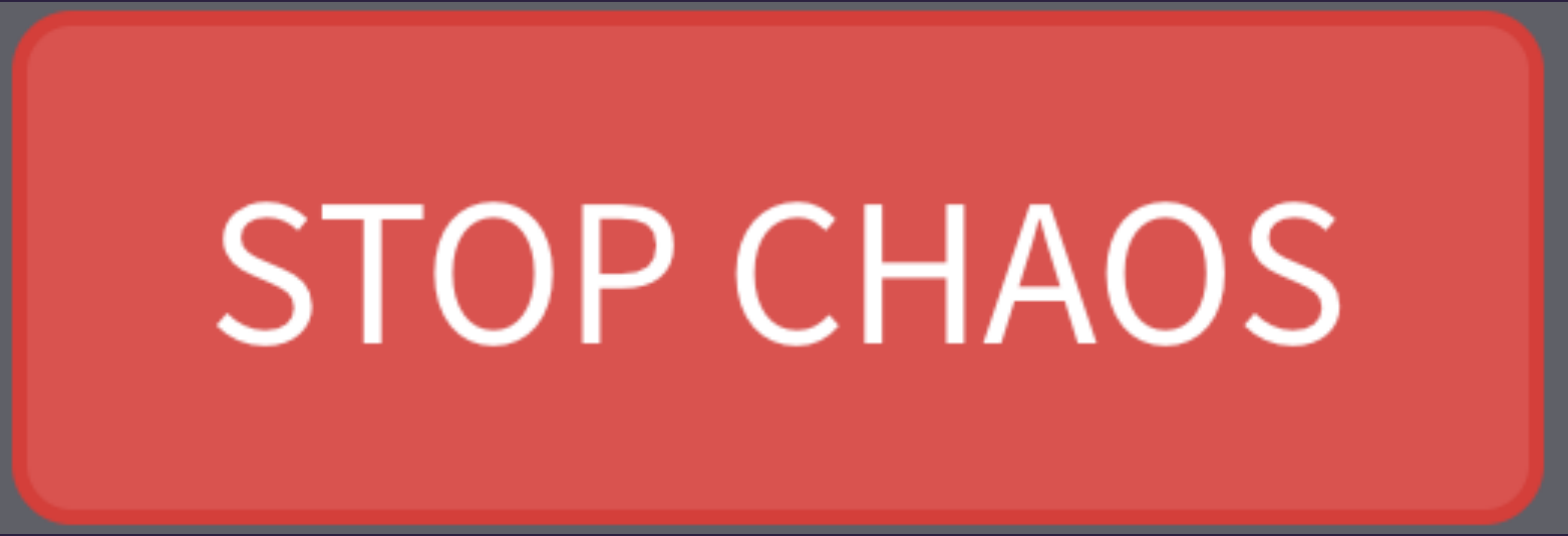
We can do controlled experiments!


98%
Upstream
API
Bookmarks
1%
API - baseline
1%
fail
API - canary



```
🟦 baseline
     Max :    57.500     Min  :      1.000
     Avg :    42.981     Last :      1.000
     Tot :     2.321k    Cnt  :     54.000
🟫 canary
     Max :    67.000     Min  :      2.000
     Avg :    46.509     Last :      2.000
     Tot :     2.512k    Cnt  :     54.000
□ experiment
     Max :     1.000     Min  :      0.000
     Avg :   818.182m    Last :      0.000
     Tot :    45.000     Cnt  :     55.000
... 1 of 1 lines matched filter ...

Frame: 55m, End: 2018-11-27T14:42-08:00[US/Pacific], Step: 1m
Fetch: 391ms (L: 16.0, 8.0, 4.0; D: 960.0, 448.0, 220.0k)
```

How do we do this safely?


STOP CHAOS

Automatic stop

(<5 minutes)

Business hours only

Limit number of simultaneous runs

How do we scale this?

First attempt: self-serve

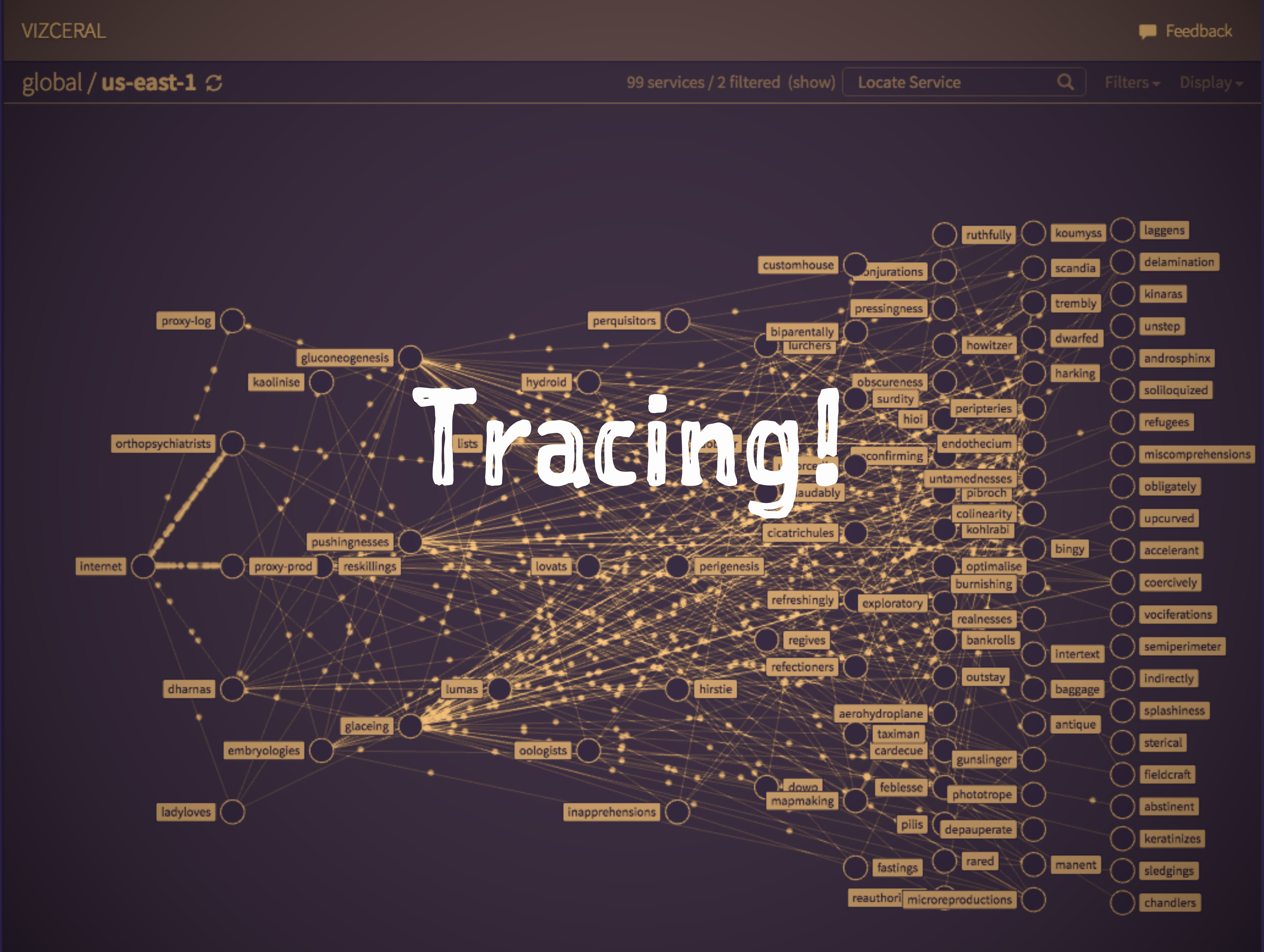
Actively engage with multiple teams

Didn't see uptake after engagements ☹️

Second attempt: automatically generate experiments

Problem: need to understand services to design experiments

What other services do they communicate with?


VIZCERAL
Feedback
global / us-east-1
99 services / 2 filtered (show)
Locate Service
Filters
Display
internet
proxy-log
orthopsychiatrists
proxy-prod
dharnas
ladyloves
kaolinise
embryologies
gluconeogenesis
pushingnesses
reskillings
glaceing
perquisitors
hydroid
lists
lovats
lumas
oologists
inapprehensions
perigenesis
hirstie
customhouse
biparentally
lurchers
cicatrichules
refreshingly
regives
refectioners
mapmaking
conjurations
pressingness
obscureness
surdity
hioi
confirming
audably
exploratory
aerohydroplane
taximan
cardecue
feblesse
pilis
fastings
reauthori
ruthfully
howitzer
peripteries
endothecium
untamednesses
pibroch
colinearity
kohlrabi
optimalise
burnishing
realnesses
bankrolls
outstay
gunslinger
phototrope
depauperate
rared
microreproductions
koumyss
scandia
trembly
dwarfed
harking
bingy
intertext
baggage
antique
manent
laggens
delamination
kinaras
unstep
androsphinx
soliloquized
refugees
miscomprehensions
obligately
upcurved
accelerant
coercively
vociferations
semiperimeter
indirectly
splashiness
sterical
fieldcraft
abstinent
keratinizes
sledgings
chandlers

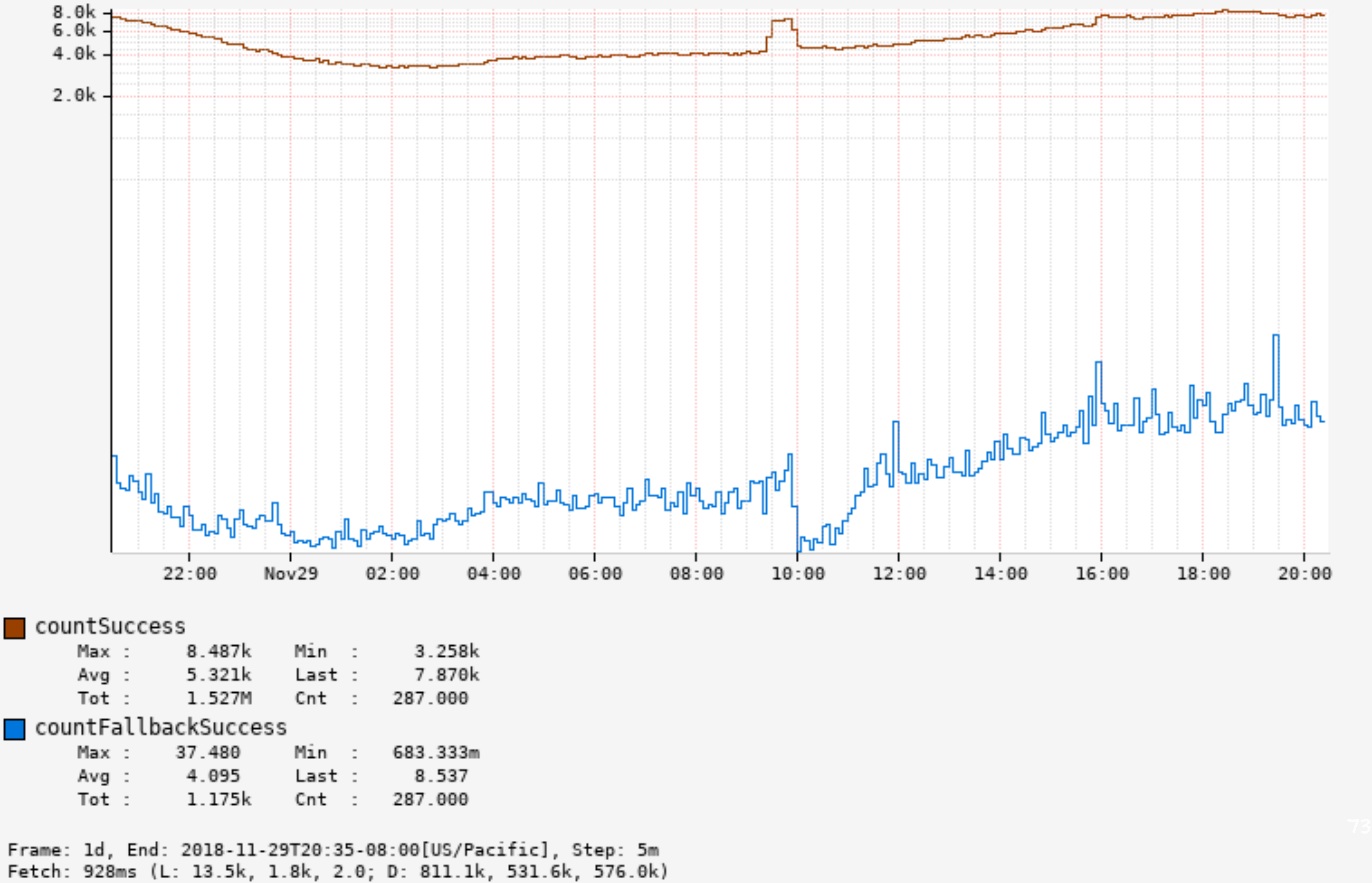
Tracing!

Which RPCs do we believe are safe to fail?

Heuristics!

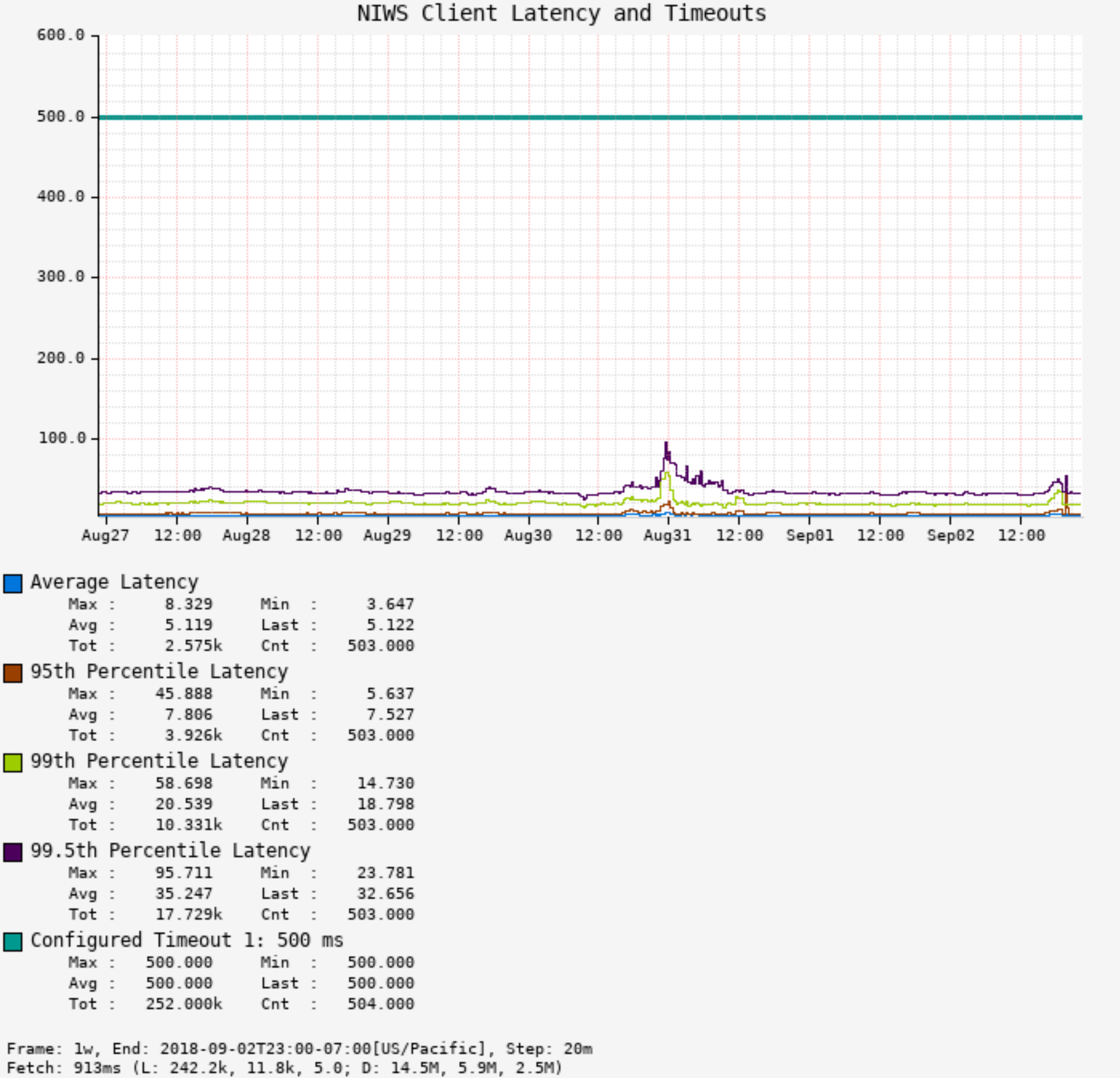
Is there a fallback?

Does the fallback ever get invoked?



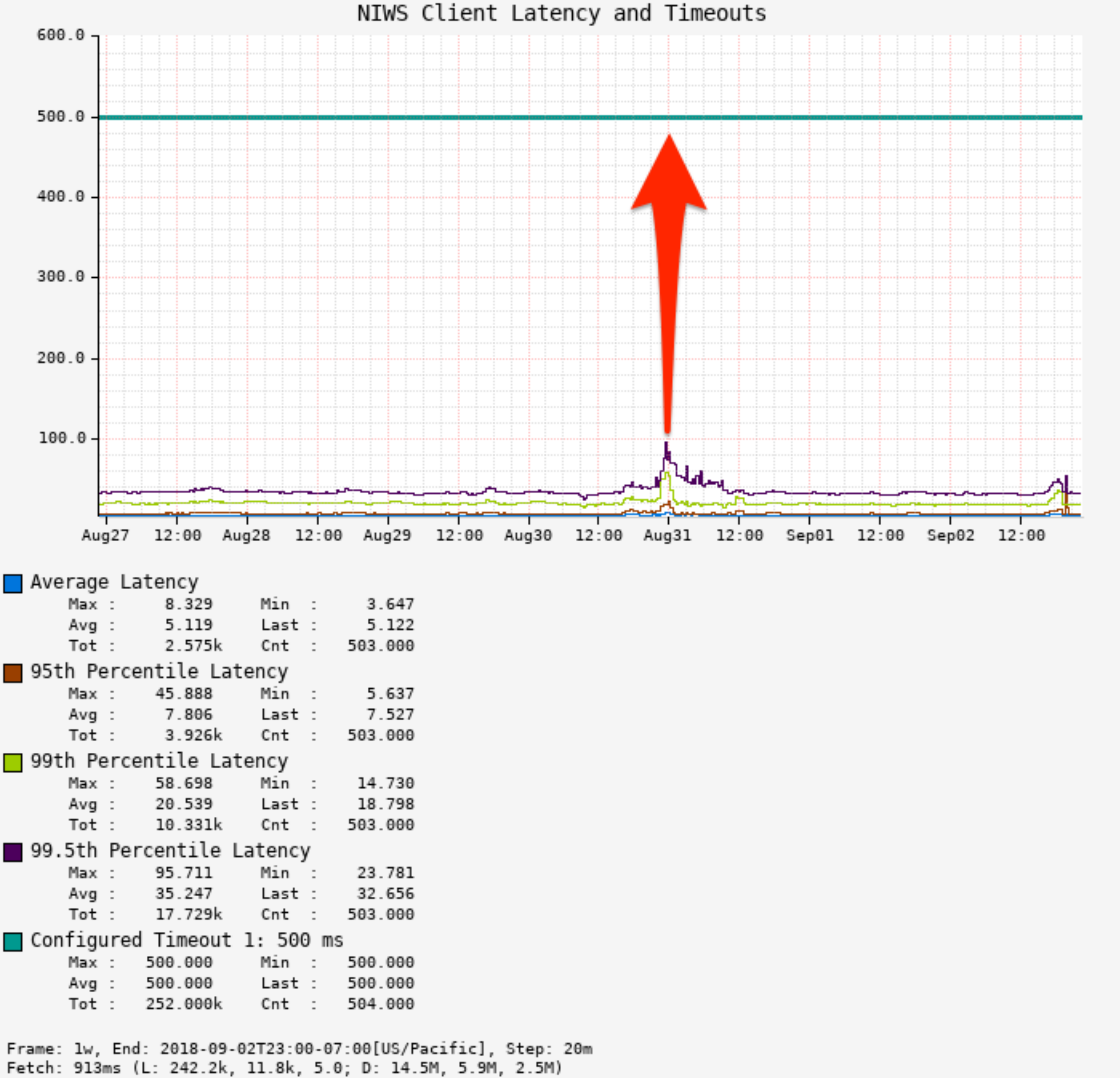
```
🟫 countSuccess
     Max :     8.487k    Min  :     3.258k
     Avg :     5.321k    Last :     7.870k
     Tot :     1.527M    Cnt  :   287.000
🟦 countFallbackSuccess
     Max :    37.480     Min  :   683.333m
     Avg :     4.095     Last :     8.537
     Tot :     1.175k    Cnt  :   287.000

Frame: 1d, End: 2018-11-29T20:35-08:00[US/Pacific], Step: 5m
Fetch: 928ms (L: 13.5k, 1.8k, 2.0; D: 811.1k, 531.6k, 576.0k)
```

How much latency should we inject?



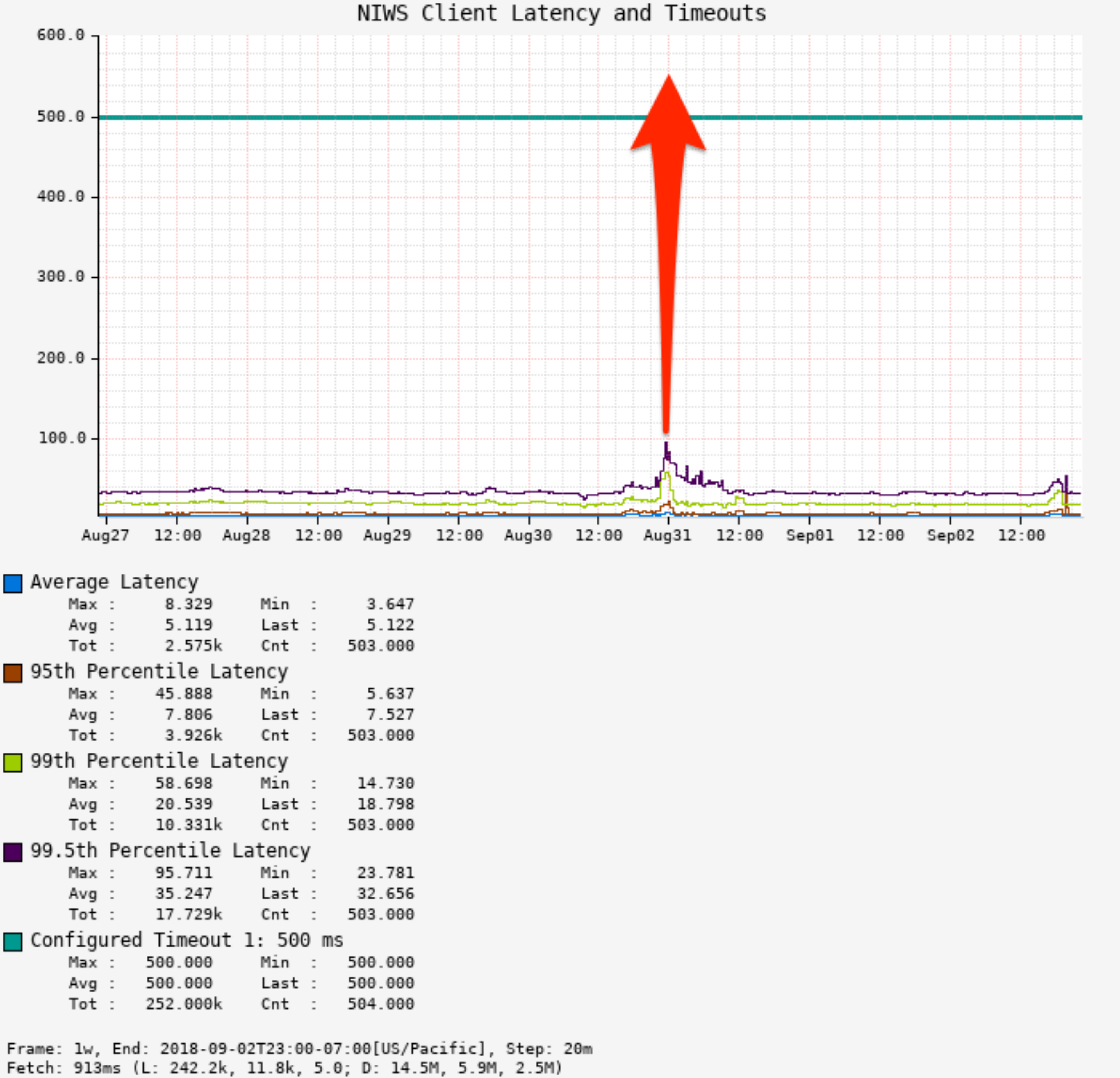
NIWS Client Latency and Timeouts

```
🟦 Average Latency
      Max :     8.329      Min  :     3.647
      Avg :     5.119      Last :     5.122
      Tot :     2.575k     Cnt  :   503.000
🟫 95th Percentile Latency
      Max :    45.888      Min  :     5.637
      Avg :     7.806      Last :     7.527
      Tot :     3.926k     Cnt  :   503.000
🟩 99th Percentile Latency
      Max :    58.698      Min  :    14.730
      Avg :    20.539      Last :    18.798
      Tot :    10.331k     Cnt  :   503.000
🟪 99.5th Percentile Latency
      Max :    95.711      Min  :    23.781
      Avg :    35.247      Last :    32.656
      Tot :    17.729k     Cnt  :   503.000
🟩 Configured Timeout 1: 500 ms
      Max :   500.000      Min  :   500.000
      Avg :   500.000      Last :   500.000
      Tot :   252.000k     Cnt  :   504.000

Frame: 1w, End: 2018-09-02T23:00-07:00[US/Pacific], Step: 20m
Fetch: 913ms (L: 242.2k, 11.8k, 5.0; D: 14.5M, 5.9M, 2.5M)
```



NIWS Client Latency and Timeouts

```
🟦 Average Latency
       Max :     8.329      Min  :     3.647
       Avg :     5.119      Last :     5.122
       Tot :     2.575k     Cnt  :   503.000
🟫 95th Percentile Latency
       Max :    45.888      Min  :     5.637
       Avg :     7.806      Last :     7.527
       Tot :     3.926k     Cnt  :   503.000
🟩 99th Percentile Latency
       Max :    58.698      Min  :    14.730
       Avg :    20.539      Last :    18.798
       Tot :    10.331k     Cnt  :   503.000
🟪 99.5th Percentile Latency
       Max :    95.711      Min  :    23.781
       Avg :    35.247      Last :    32.656
       Tot :    17.729k     Cnt  :   503.000
🟦 Configured Timeout 1: 500 ms
       Max :   500.000      Min  :   500.000
       Avg :   500.000      Last :   500.000
       Tot :   252.000k     Cnt  :   504.000

Frame: 1w, End: 2018-09-02T23:00-07:00[US/Pacific], Step: 20m
Fetch: 913ms (L: 242.2k, 11.8k, 5.0; D: 14.5M, 5.9M, 2.5M)
```



```
■ Average Latency
      Max :      8.329      Min  :      3.647
      Avg :      5.119      Last :      5.122
      Tot :      2.575k     Cnt  :    503.000
■ 95th Percentile Latency
      Max :     45.888      Min  :      5.637
      Avg :      7.806      Last :      7.527
      Tot :      3.926k     Cnt  :    503.000
■ 99th Percentile Latency
      Max :     58.698      Min  :     14.730
      Avg :     20.539      Last :     18.798
      Tot :     10.331k     Cnt  :    503.000
■ 99.5th Percentile Latency
      Max :     95.711      Min  :     23.781
      Avg :     35.247      Last :     32.656
      Tot :     17.729k     Cnt  :    503.000
■ Configured Timeout 1: 500 ms
      Max :    500.000      Min  :    500.000
      Avg :    500.000      Last :    500.000
      Tot :    252.000k     Cnt  :    504.000

Frame: 1w, End: 2018-09-02T23:00-07:00[US/Pacific], Step: 20m
Fetch: 913ms (L: 242.2k, 11.8k, 5.0; D: 14.5M, 5.9M, 2.5M)
```

We found vulnerabilities!

Still requires human effort to interpret results

Experimental design limited by our heuristics

Current state: hybrid approach

Busy season: right before the holidays

Act III: Lessons learned

Safety

It needs to be safe, or nobody will use it

Safe = limited impact

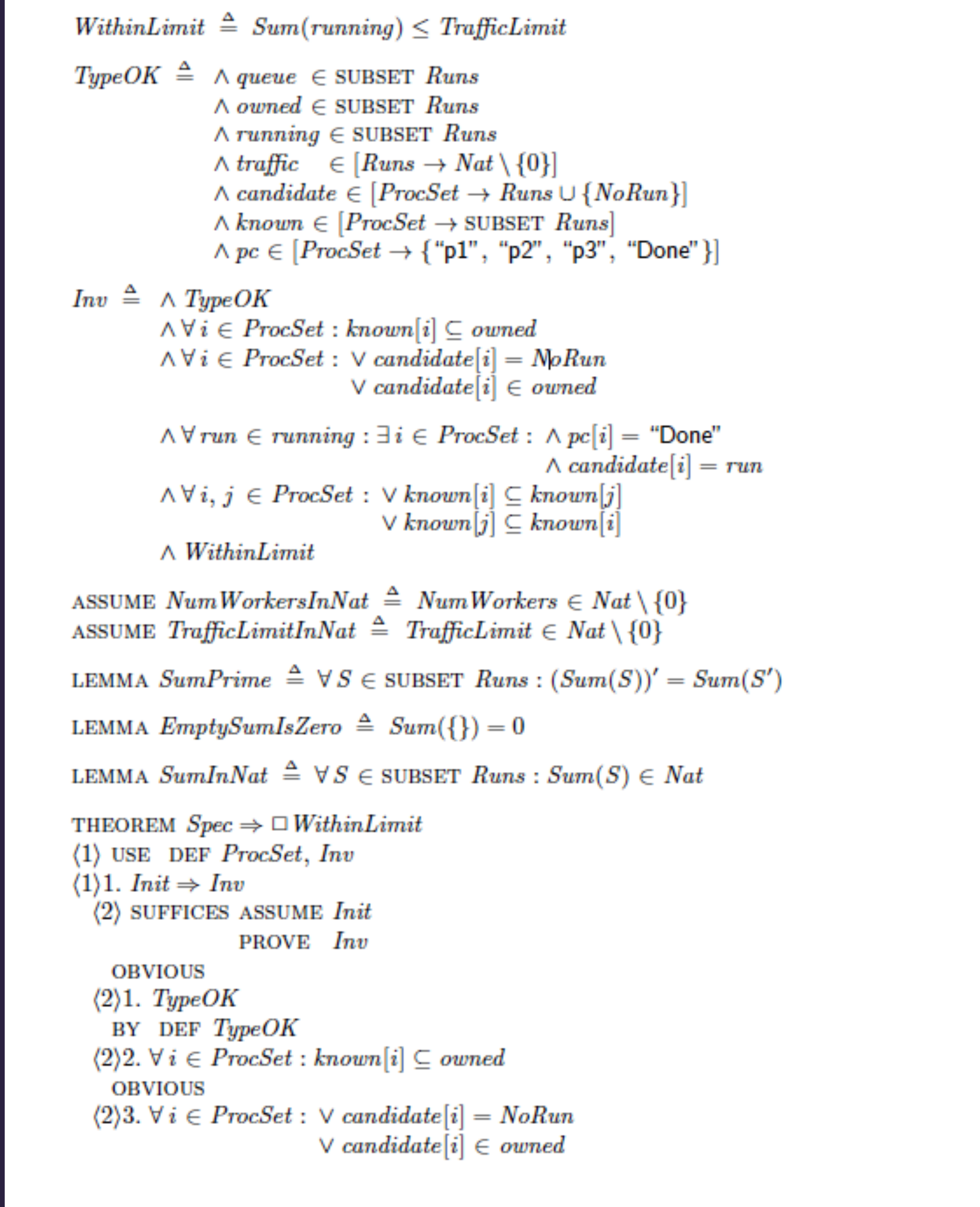
Simplicity is prerequisite for reliability

-- Edsger Dijkstra

https://en.wikipedia.org/wiki/Edsger_W._Dijkstra#/media/File:Edsger_Wybe_Dijkstra.jpg

No!

Safety adds complexity



$$WithinLimit \triangleq Sum(running) \leq TrafficLimit$$

$$\begin{array}{lll} TypeOK \triangleq & \land\, queue & \in \text{SUBSET } Runs \\ & \land\, owned & \in \text{SUBSET } Runs \\ & \land\, running & \in \text{SUBSET } Runs \\ & \land\, traffic & \in [Runs \rightarrow Nat \setminus \{0\}] \\ & \land\, candidate & \in [ProcSet \rightarrow Runs \cup \{NoRun\}] \\ & \land\, known & \in [ProcSet \rightarrow \text{SUBSET } Runs] \\ & \land\, pc & \in [ProcSet \rightarrow \{\text{“p1”}, \text{“p2”}, \text{“p3”}, \text{“Done”}\}] \end{array}$$

$$\begin{array}{ll} Inv \triangleq & \land\, TypeOK \\ & \land\, \forall\, i \in ProcSet : known[i] \subseteq owned \\ & \land\, \forall\, i \in ProcSet : \begin{array}{l} \lor\, candidate[i] = NoRun \\ \lor\, candidate[i] \in owned \end{array} \\ & \\ & \land\, \forall\, run \in running : \exists\, i \in ProcSet : \begin{array}{l} \land\, pc[i] = \text{“Done”} \\ \land\, candidate[i] = run \end{array} \\ & \land\, \forall\, i, j \in ProcSet : \begin{array}{l} \lor\, known[i] \subseteq known[j] \\ \lor\, known[j] \subseteq known[i] \end{array} \\ & \land\, WithinLimit \end{array}$$

ASSUME $NumWorkersInNat \triangleq NumWorkers \in Nat \setminus \{0\}$
ASSUME $TrafficLimitInNat \triangleq TrafficLimit \in Nat \setminus \{0\}$

LEMMA $SumPrime \triangleq \forall\, S \in \text{SUBSET } Runs : (Sum(S))' = Sum(S')$

LEMMA $EmptySumIsZero \triangleq Sum(\{\}) = 0$

LEMMA $SumInNat \triangleq \forall\, S \in \text{SUBSET } Runs : Sum(S) \in Nat$

THEOREM $Spec \Rightarrow \Box\, WithinLimit$
$\langle 1 \rangle$ USE DEF $ProcSet, Inv$
$\langle 1 \rangle 1.\ Init \Rightarrow Inv$
  $\langle 2 \rangle$ SUFFICES ASSUME $Init$
                PROVE Inv
    OBVIOUS
  $\langle 2 \rangle 1.\ TypeOK$
    BY DEF $TypeOK$
  $\langle 2 \rangle 2.\ \forall\, i \in ProcSet : known[i] \subseteq owned$
    OBVIOUS
  $\langle 2 \rangle 3.\ \forall\, i \in ProcSet : \begin{array}{l} \lor\, candidate[i] = NoRun \\ \lor\, candidate[i] \in owned \end{array}$

You better have damn good tests around your failure injection logic...

...especially if it's a shared library in every app!

```
18:18:00,094 ERROR FitContextImpl:195 - Fit Error checking or injecting failure
java.lang.NullPointerException
        at com.netflix.fit.InjectionPointImpl.wildcardMatch(InjectionPointImpl.java:133)
        at com.netflix.fit.scenario.FitScenarioImpl.shouldImpact(FitScenarioImpl.java:45)
        at com.netflix.fit.FitContextImpl.shouldInjectFailure(FitContextImpl.java:130)
        at com.netflix.fit.FitContextImpl.checkAndInjectFailure(FitContextImpl.java:191)
        at com.netflix.fit.FitContext.checkAndInjectFailure(FitContext.java:40)
        at com.netflix.server.base.fit.FitHandler.handle(FitHandler.java:34)
        at com.netflix.server.base.NFFilter.safeDoFilter(NFFilter.java:574)
        at com.netflix.server.base.NFFilter.access$200(NFFilter.java:234)
        at com.netflix.server.base.NFFilter$3.call(NFFilter.java:482)
        at com.netflix.server.base.NFFilter$3.call(NFFilter.java:479)
        at com.netflix.lang.BindingContexts.callWithNewContext(BindingContexts.java:182)
        at com.netflix.server.base.NFFilter.doFilter(NFFilter.java:479)
        at com.google.inject.servlet.FilterChainInvocation.doFilter(FilterChainInvocation.java:82)
        at com.google.inject.servlet.ManagedFilterPipeline.dispatch(ManagedFilterPipeline.java:120)
        at com.google.inject.servlet.GuiceFilter.doFilter(GuiceFilter.java:135)
        at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:240)
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:207)
        at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:212)
        at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:106)
        at org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:502)
        at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:141)
```

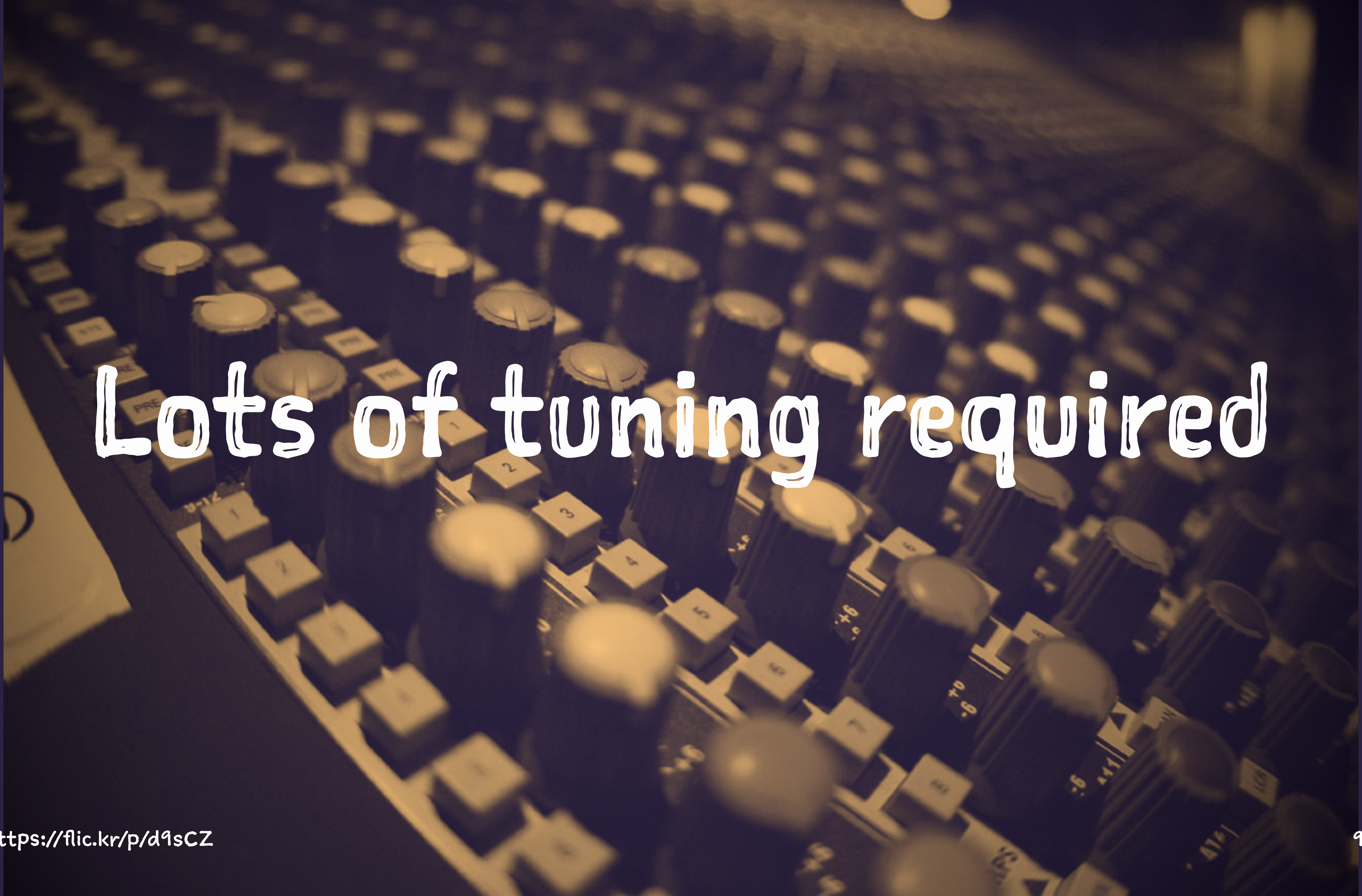
ChAP isn't a "black box"

Experimental design is a skill

Work isn't done when automated experiment reveals a weakness

Confirm it's a genuine problem

Communicate effectively back to service owners



Lots of tuning required

https://flic.kr/p/d9sCZ

Length of experiment

Amount of traffic impacted

Auto-stop thresholds

Error counts are noisy

Leverage your internal tooling ecosystem

ChAP is really an orchestration tool

- Fault injection
- Sticky routing
- Continuous deployment
- Tracing
- Telemetry
- Automated canary analysis

The more heterogeneous your ecosystem, the harder life will be

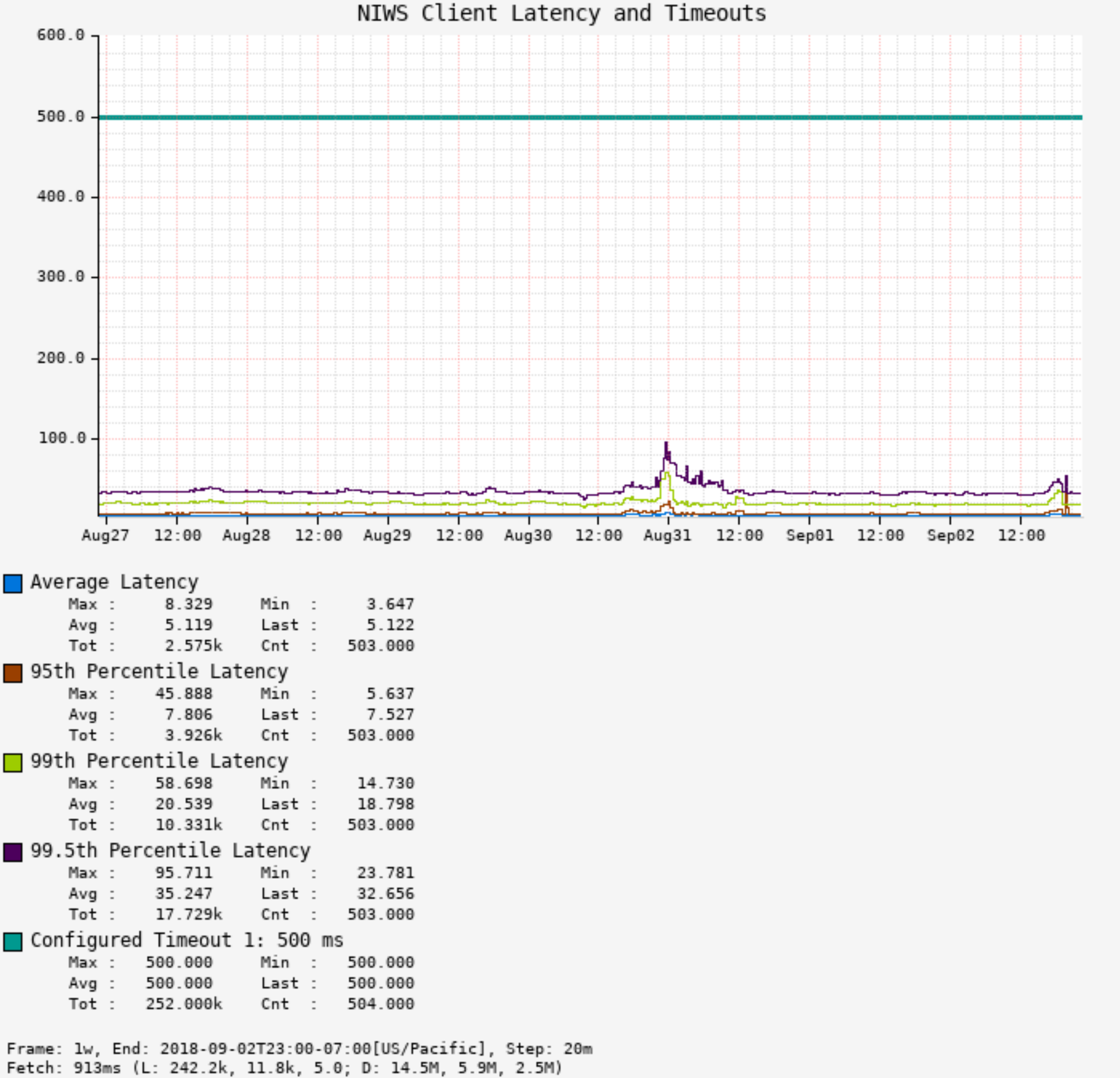
Java -> Node.js

REST -> gRPC

VMs -> containers

Unexpected benefits

Info for experiment generation was useful to service owners



```
🟦 Average Latency
        Max :      8.329      Min  :      3.647
        Avg :      5.119      Last :      5.122
        Tot :      2.575k     Cnt  :    503.000
🟫 95th Percentile Latency
        Max :     45.888      Min  :      5.637
        Avg :      7.806      Last :      7.527
        Tot :      3.926k     Cnt  :    503.000
🟩 99th Percentile Latency
        Max :     58.698      Min  :     14.730
        Avg :     20.539      Last :     18.798
        Tot :     10.331k     Cnt  :    503.000
🟪 99.5th Percentile Latency
        Max :     95.711      Min  :     23.781
        Avg :     35.247      Last :     32.656
        Tot :     17.729k     Cnt  :    503.000
🟦 Configured Timeout 1: 500 ms
        Max :    500.000      Min  :    500.000
        Avg :    500.000      Last :    500.000
        Tot :    252.000k     Cnt  :    504.000

Frame: 1w, End: 2018-09-02T23:00-07:00[US/Pacific], Step: 20m
Fetch: 913ms (L: 242.2k, 11.8k, 5.0; D: 14.5M, 5.9M, 2.5M)
```

Engineers created new use cases (sticky canary)

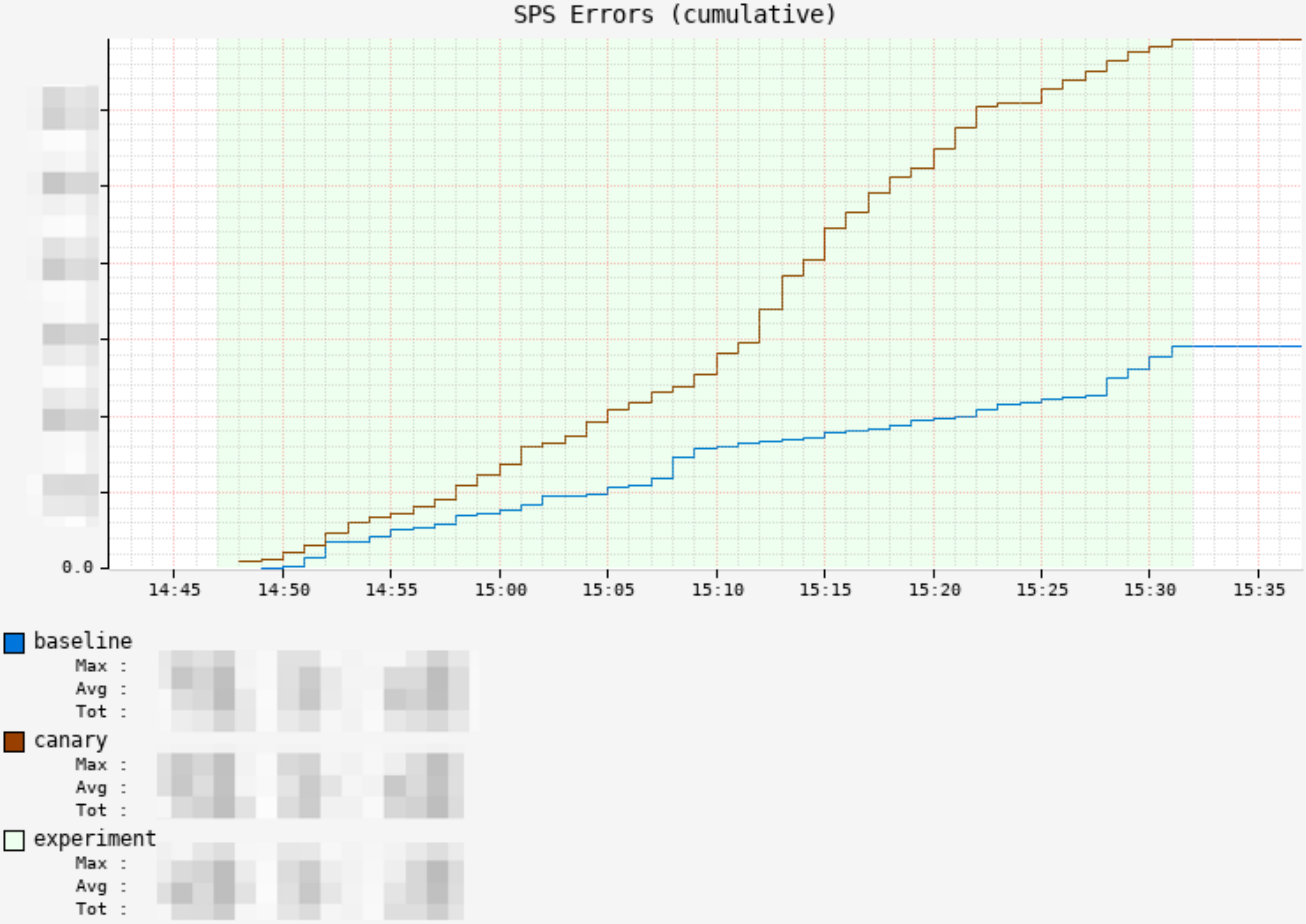

SPS Errors (cumulative)
[redacted]
0.0
14:45
14:50
14:55
15:00
15:05
15:10
15:15
15:20
15:25
15:30
15:35
baseline
Max : [redacted]
Avg : [redacted]
Tot : [redacted]
canary
Max : [redacted]
Avg : [redacted]
Tot : [redacted]
experiment
Max : [redacted]
Avg : [redacted]
Tot : [redacted]

Image credits

- "Knobs", Ian Harding, CC-BY-NC-SA 2.0: https://flic.kr/p/d9sCZ
- "Portrait of Edsger W. Dijkstra", Hamilton Richards, CC BY-SA 3.0: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra#/media/File:EdsgerWybeDijkstra.jpg