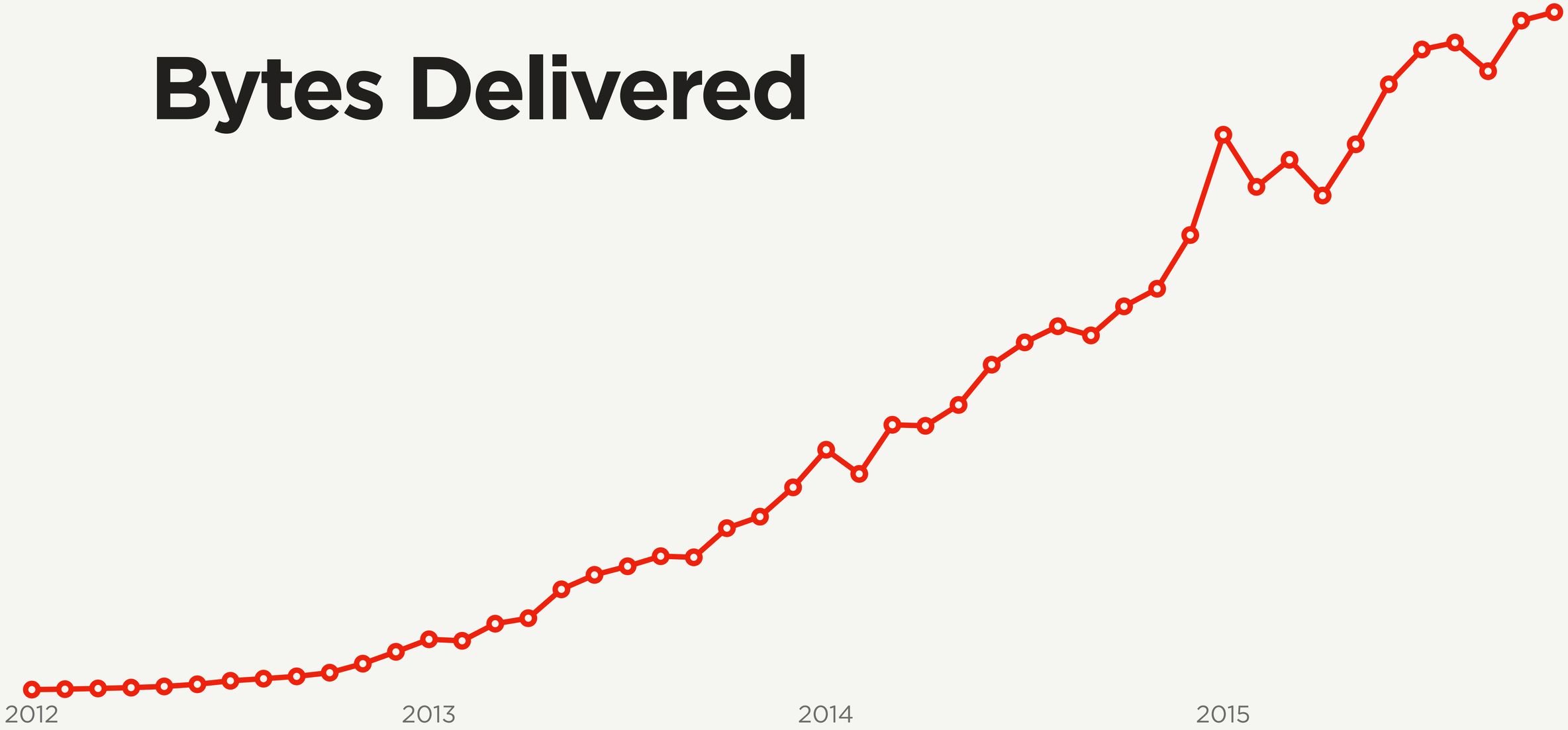
Zuul's Journey to Non-Blocking

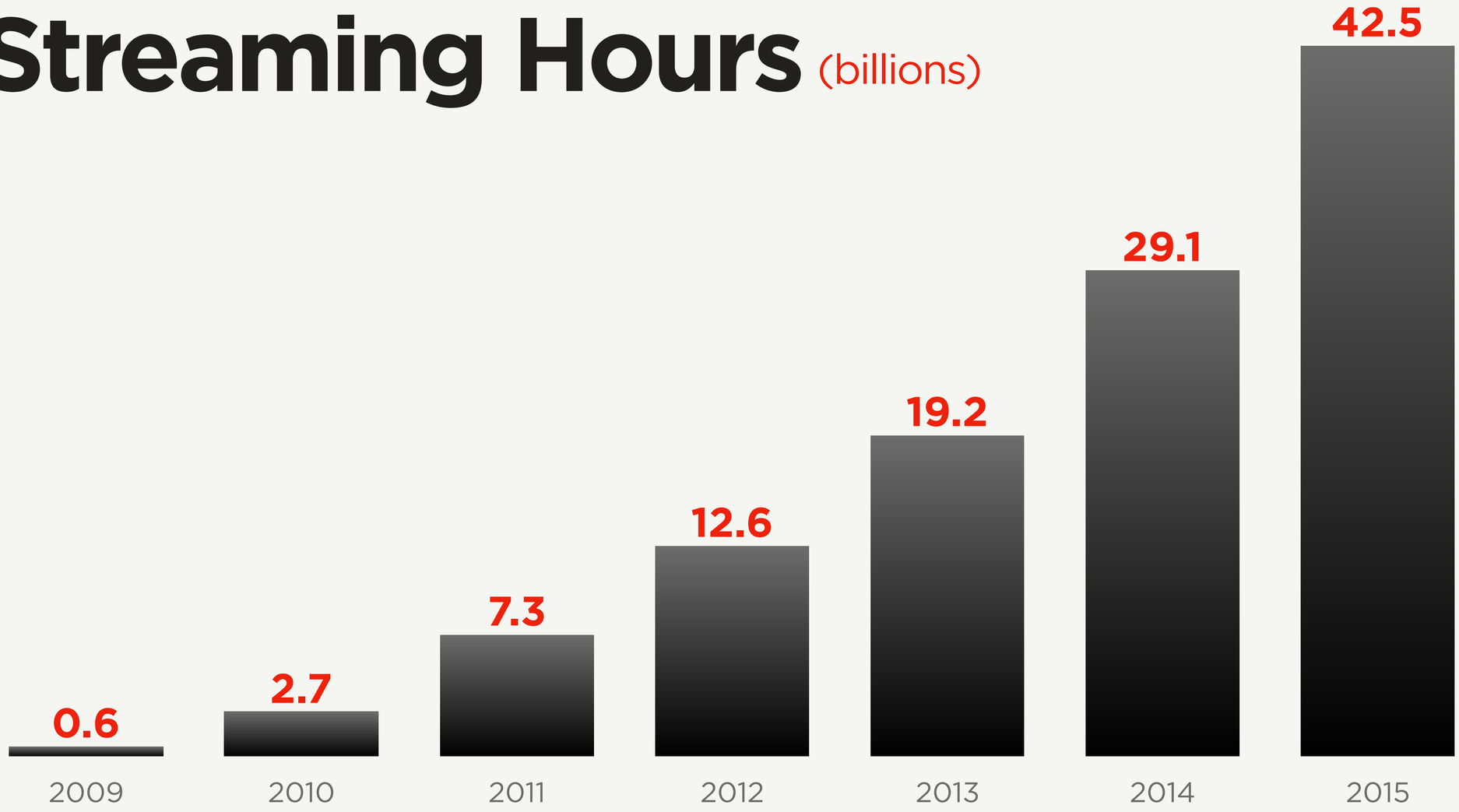
Arthur Gonigberg
Cloud Gateway

NETFLIX

Bytes Delivered



Streaming Hours (billions)



Zuul.

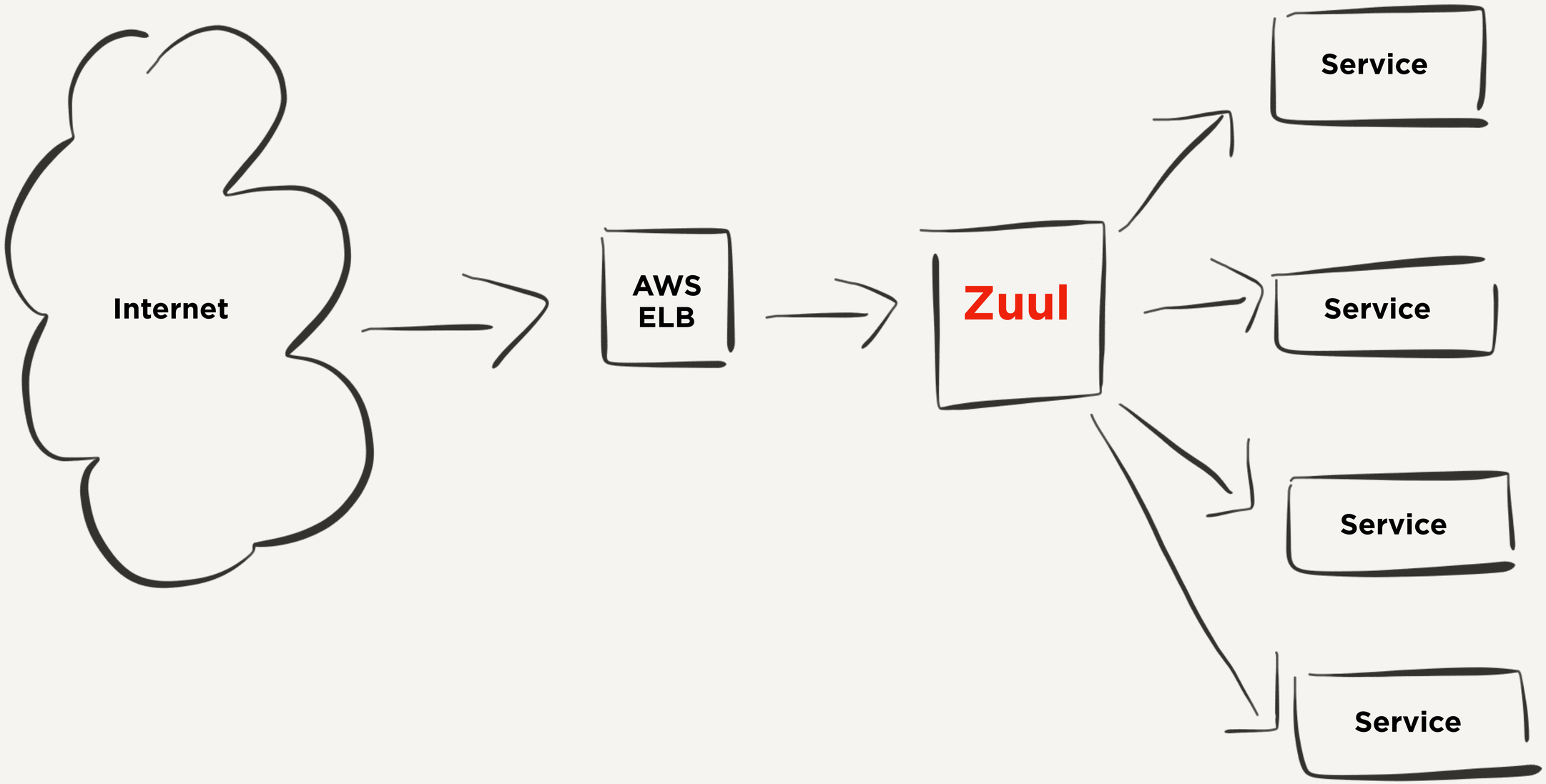
Routing

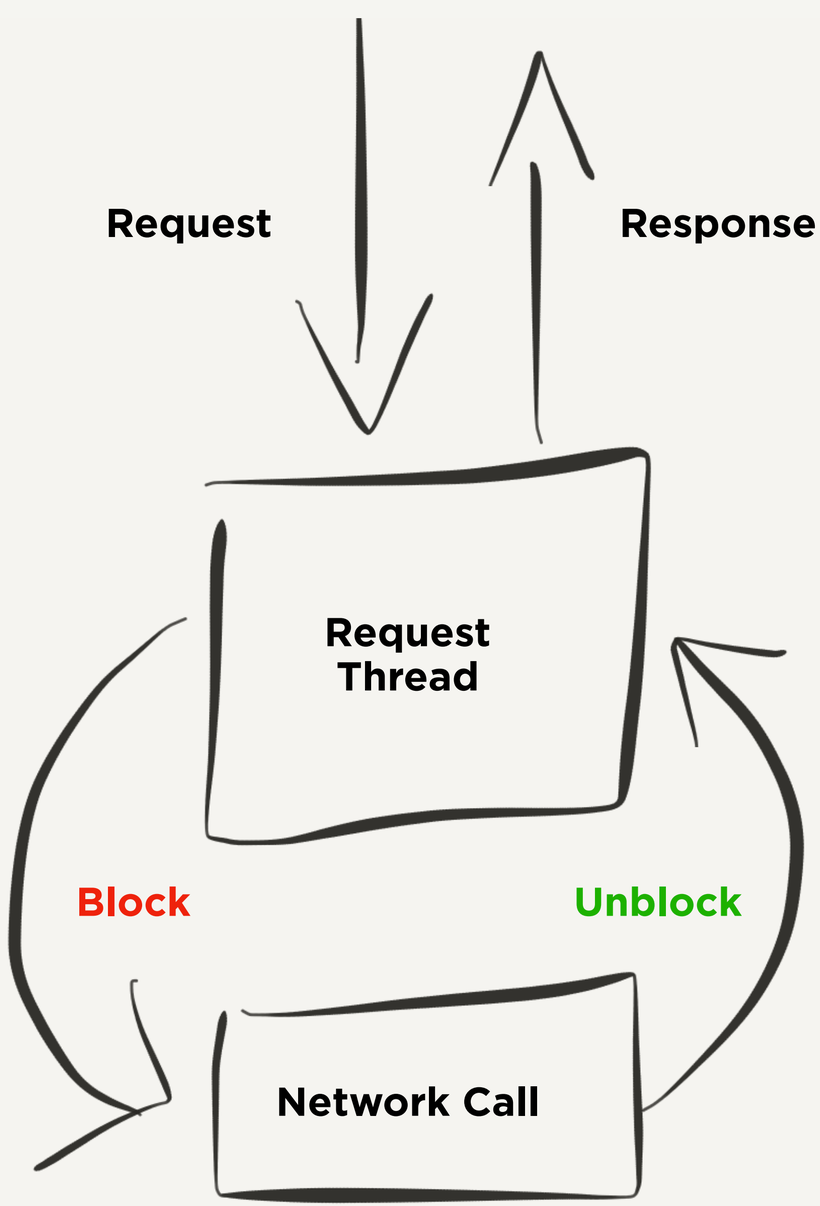
Monitoring

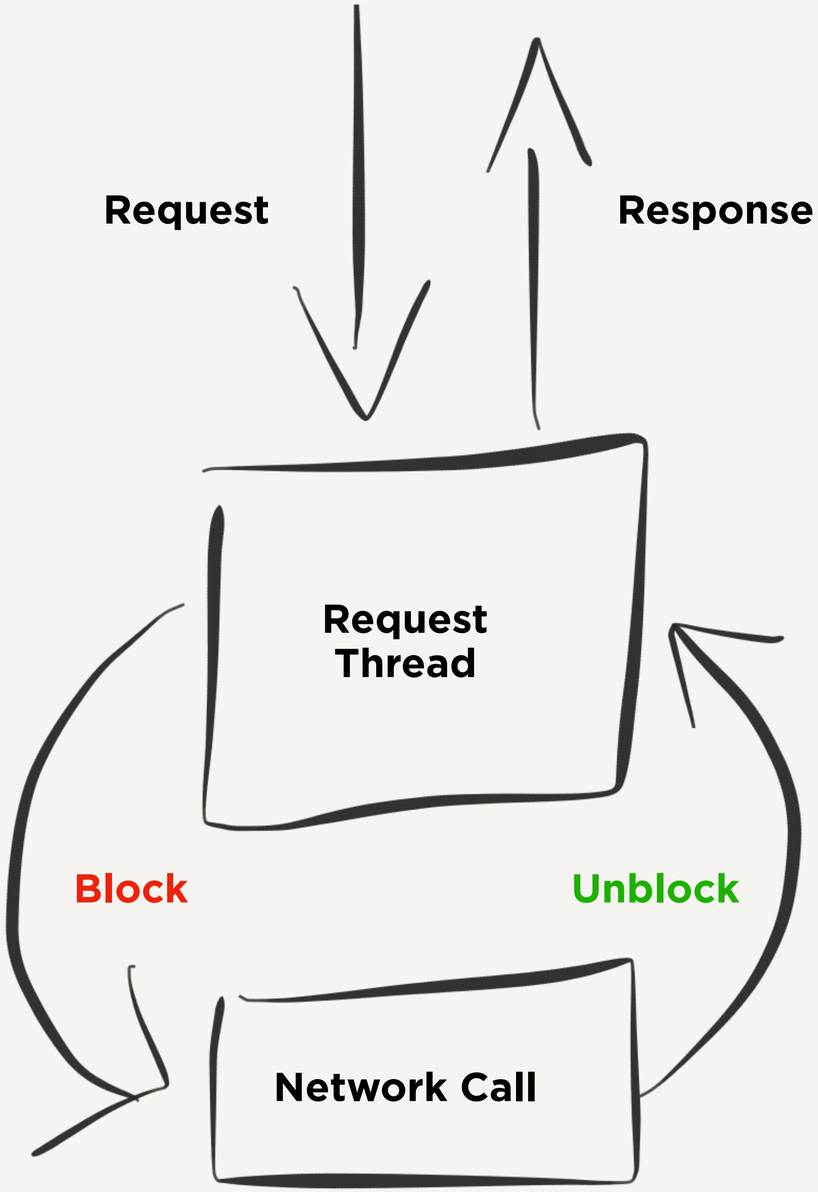
Security

Resiliency

Flexibility

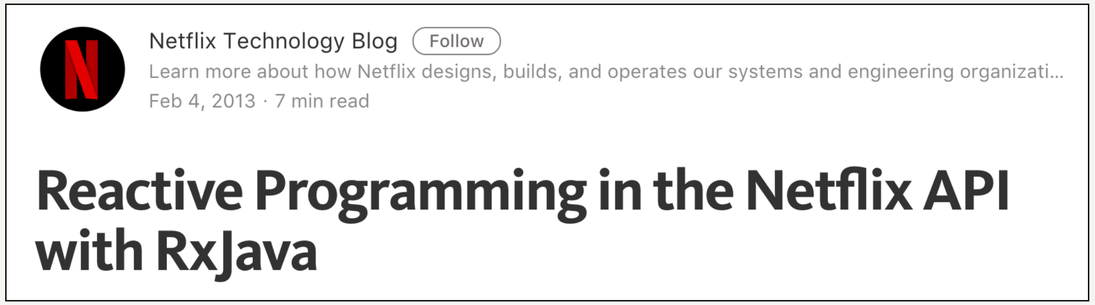

Internet
AWS
ELB
Zuul
Service
Service
Service
Service


Request
Response
Request Thread
Block
Unblock
Network Call


Request
Response
Request Thread
Block
Unblock
Network Call


Netflix Technology Blog
Follow
Learn more about how Netflix designs, builds, and operates our systems and engineering organizati...
Nov 26, 2012 · 2 min read
Introducing Hystrix for Resilience Engineering


Netflix Brings Reactive Extensions to Java
Like
| by Jonathan Allen on Feb 06, 2013. Estimated reading time: 1 minute |
Discuss


Netflix Technology Blog
Follow
Learn more about how Netflix designs, builds, and operates our systems and engineering organizati...
Feb 4, 2013 · 7 min read
Reactive Programming in the Netflix API with RxJava


The Reactive Manifesto
Published on September 16 2014. (v2.0)

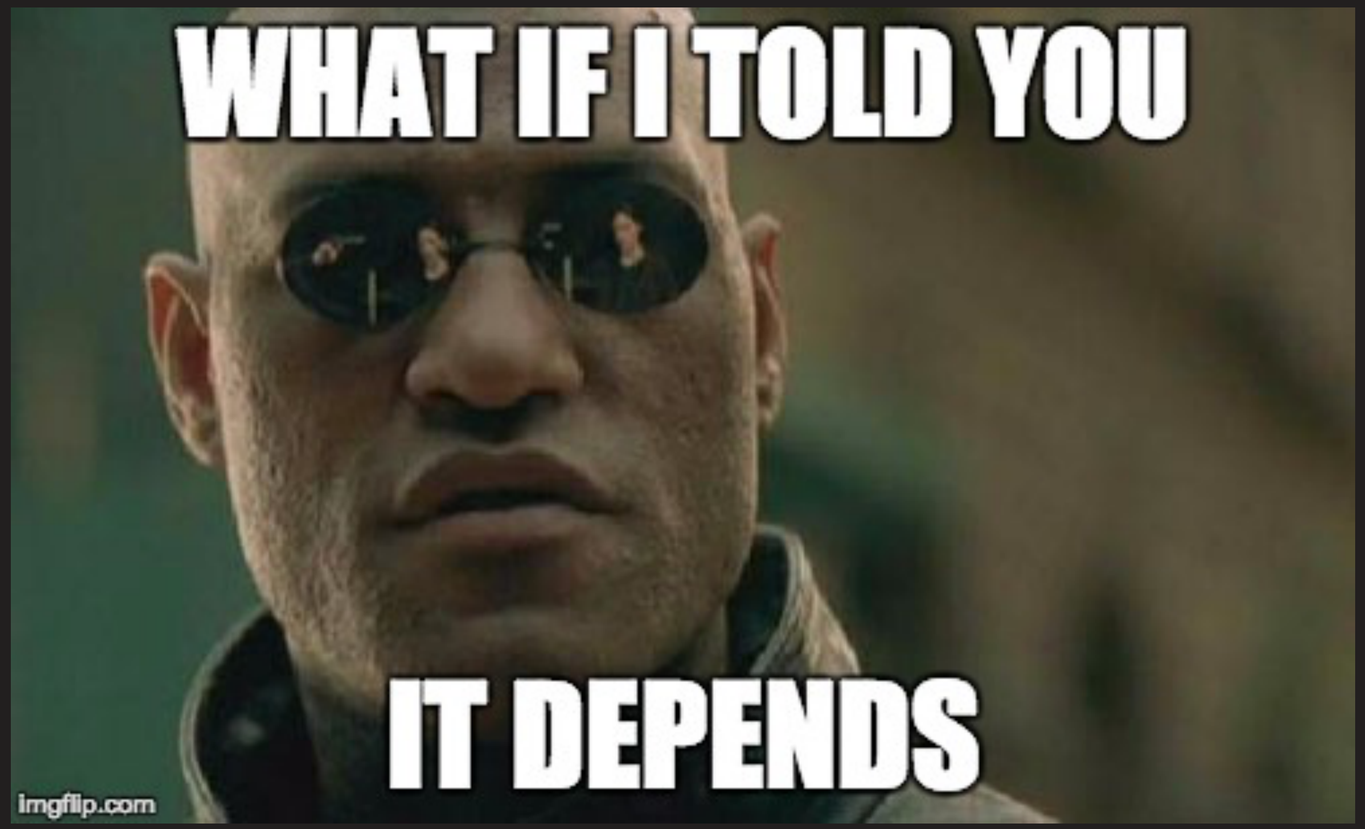
Hello Netty

```
public class HttpServerHandler extends SimpleChannelInboundHandler<HttpObject> {

    @Override
    protected void channelRead0(ChannelHandlerContext ctx, HttpObject msg) throws Exception {
        if (msg instanceof LastHttpContent) {
            ByteBuf content = Unpooled.copiedBuffer("Hello World.", CharsetUtil.UTF_8);
            FullHttpResponse response = new DefaultFullHttpResponse(HttpVersion.HTTP_1_1,
                                                                    HttpResponseStatus.OK, content);
            response.headers().set(HttpHeaders.Names.CONTENT_TYPE, "text/plain");
            response.headers().set(HttpHeaders.Names.CONTENT_LENGTH, content.readableBytes());
            ctx.write(response);
        }
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) throws Exception {
        ctx.flush();
    }
}
```

```
-> % ab -c 200 -n 1000000 -k http://localhost:8080/
This is ApacheBench, Version 2.3 <$Revision: 1757674 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100000 requests
Completed 200000 requests
Completed 300000 requests
Completed 400000 requests
Completed 500000 requests
Completed 600000 requests
Completed 700000 requests
Completed 800000 requests
Completed 900000 requests
Completed 1000000 requests
Finished 1000000 requests


Server Software:
Server Hostname:        localhost
Server Port:            8080

Document Path:          /
Document Length:        11 bytes

Concurrency Level:      200
Time taken for tests:   13.587 seconds
Complete requests:      1000000
Failed requests:        0
Keep-Alive requests:    1000000
Total transferred:      100000000 bytes
HTML transferred:       11000000 bytes
Requests per second:    73602.19 [#/sec] (mean)
Time per request:       2.717 [ms] (mean)
Time per request:       0.014 [ms] (mean, across all concurrent requests)
Transfer rate:          7187.71 [Kbytes/sec] received
```

Real World Performance

Requests per second per node


WHAT IF I TOLD YOU
IT DEPENDS
imgflip.com

Today's Journey

Rebuilding Zuul on Netty and RxJava

Operating Zuul 2.0 in the Real World

When to go Blocking or Non-Blocking

Starting the Journey

Rebuilding Zuul on Netty and RxJava

Zuul

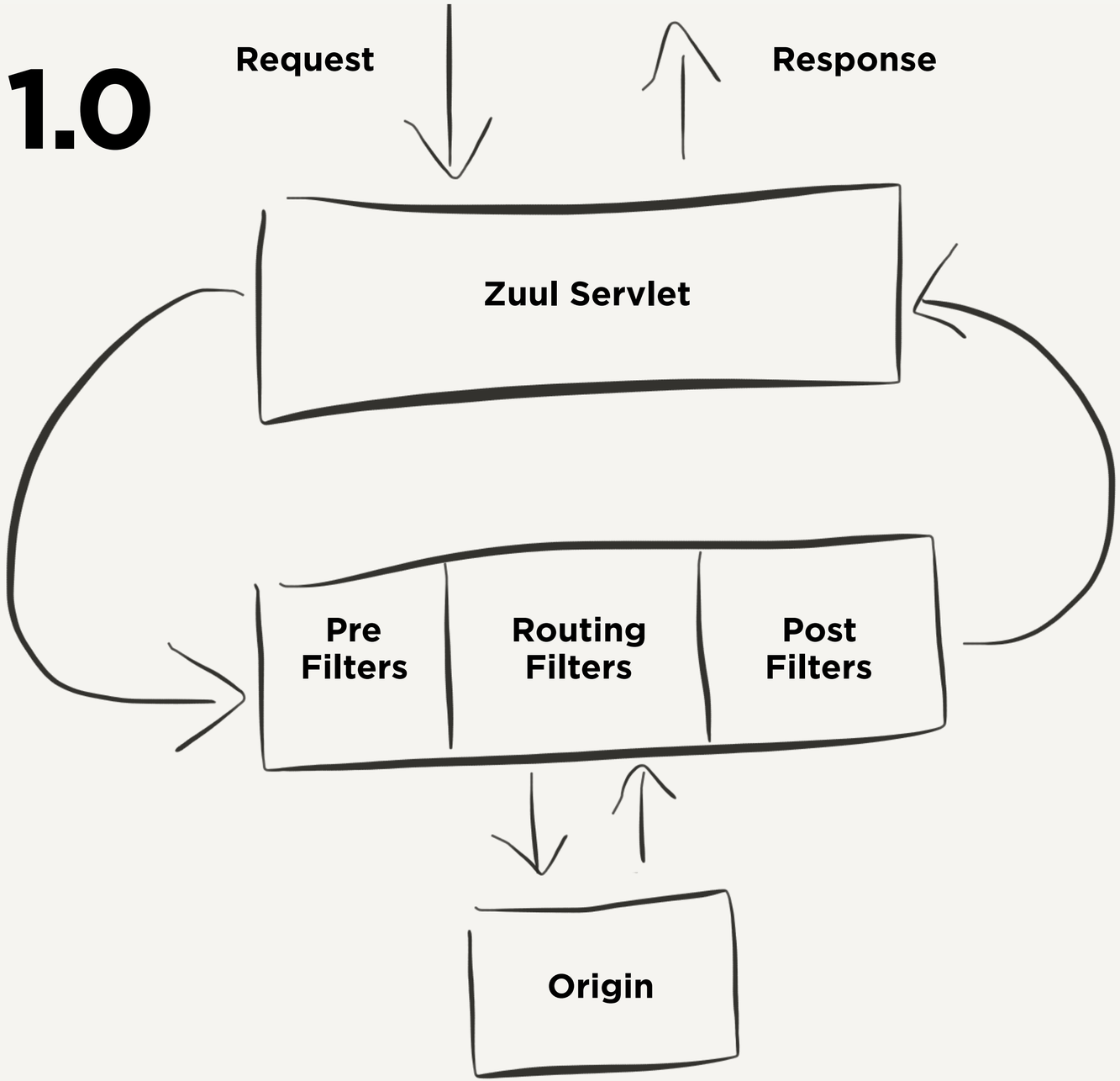
Proxy all traffic coming into AWS

Traffic from (almost) every country

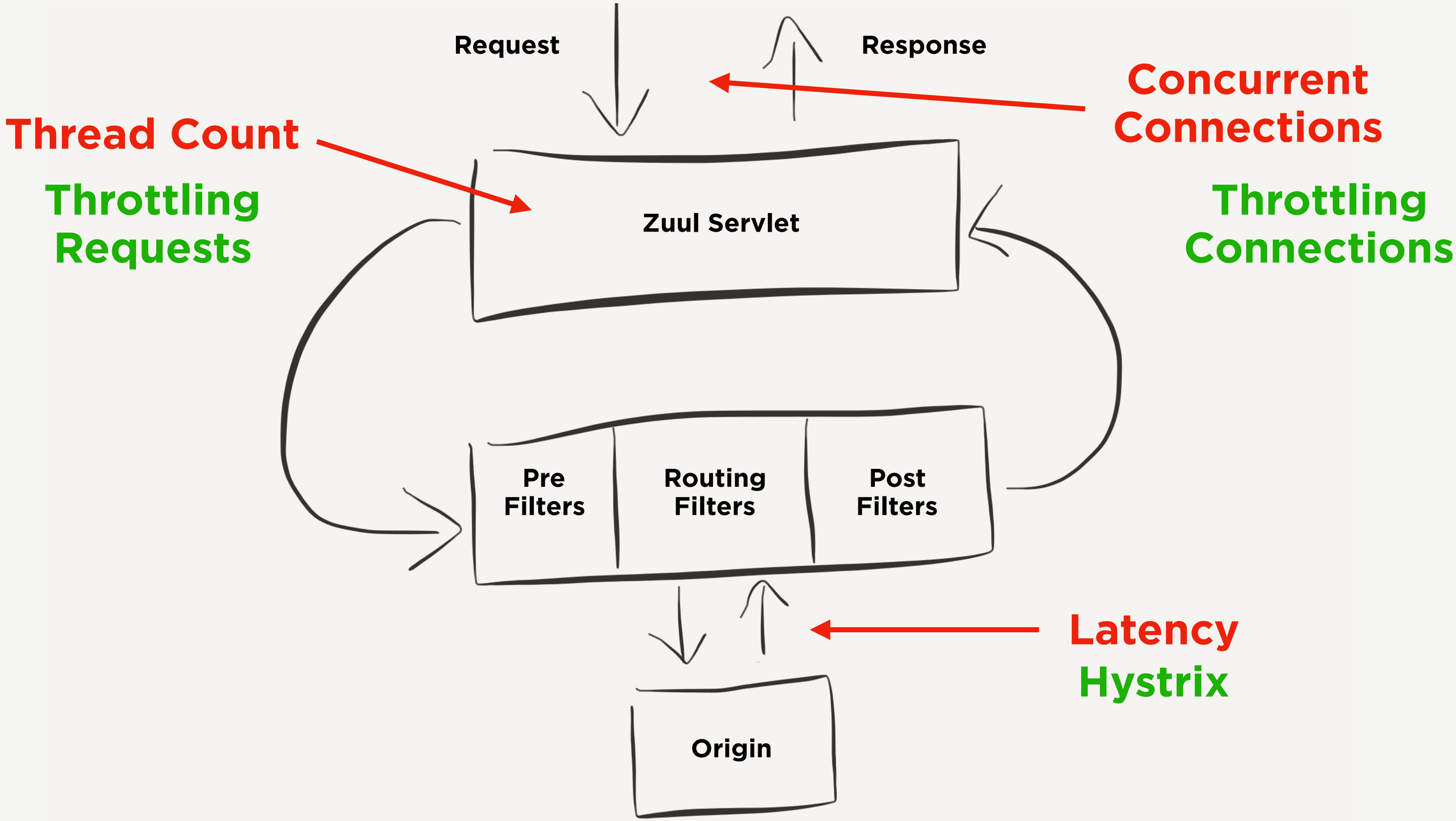
Tens of billions of requests per day

Hundreds of terabytes transferred per day

Zuul 1.0


Request
Response
Zuul Servlet
Pre Filters
Routing Filters
Post Filters
Origin


Encryption
Geo Data Lookup
Authentication
Static Response Handling
Decryption
Connection Pooling
Hashing
Routing
DDoS Protection
Sticky Canari
Security
Service Discovery
Request Context Population
Load Balanc
Load Shedding
Stress Testing
Atlas Metrics
Canary Testing
Insights
Dynamic Configuration Management
LOTS OF STUFF


Request
Response
Concurrent Connections
Thread Count
Throttling Requests
Zuul Servlet
Throttling Connections
Pre Filters
Routing Filters
Post Filters
Latency
Hystrix
Origin

Noel
@noellinnane

Follow

Netflix is down! Panic!!

1:06 PM - 1 Oct 2016

Bongo
@congobongo93

Follow

#Netflixdown per tre minuti e parte il panico

Translate from Italian

12:57 PM - 15 Jun 2017

ᑭᕃᓐ - Kevin
@MurmeltierS

Follow

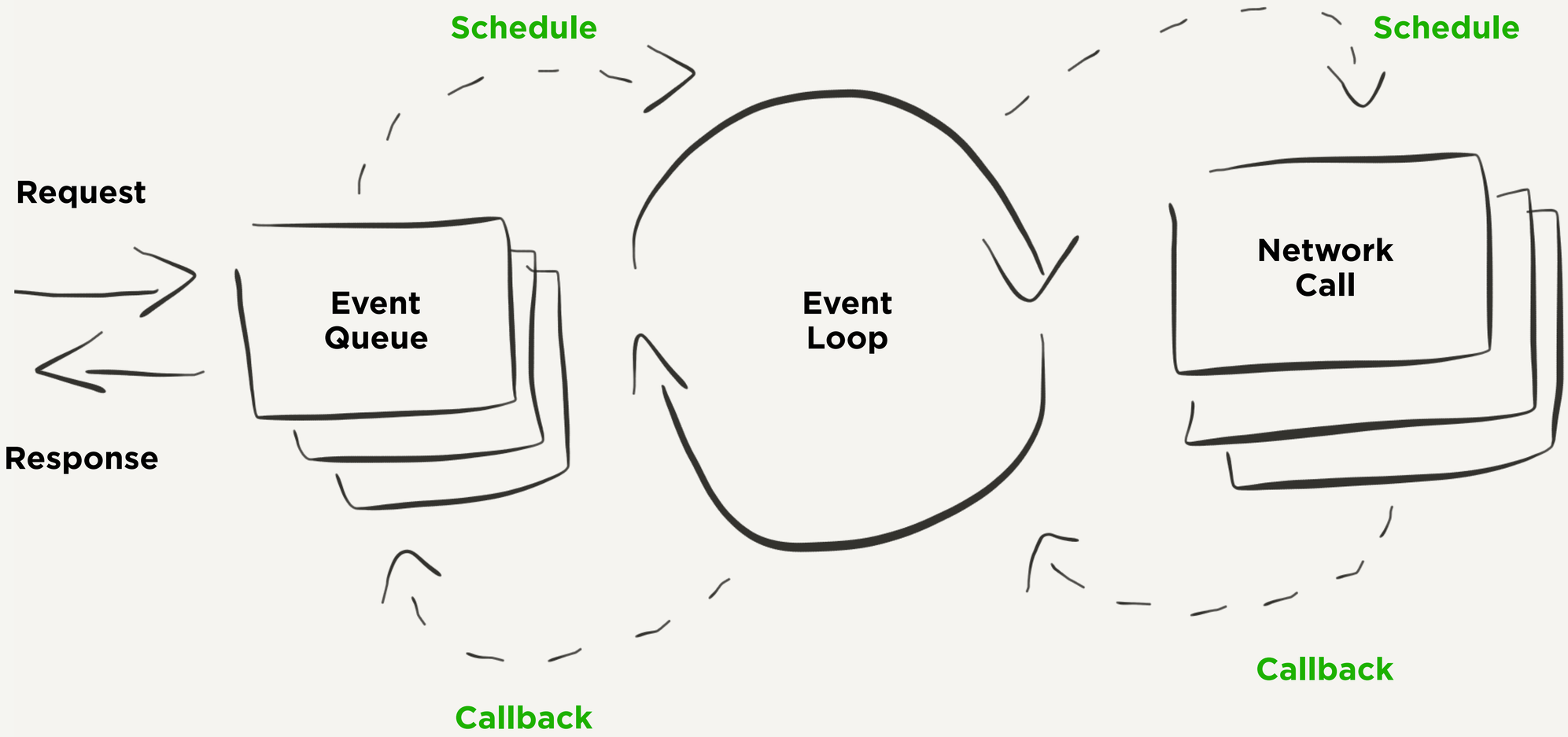
Panik, Panik #netflixdown

1:58 PM - 15 Oct 2015

What is Non-Blocking?

Non-Blocking I/O Primitives
Asynchronous Programming
Event-loop Pattern

Event Loop


Schedule
Schedule
Request
Event Queue
Event Loop
Network Call
Response
Callback
Callback
Single Thread

Why Non-Blocking?

Performance → CPU Utilization

Resiliency → Connection Scaling

Zuul 1.5

APR connector
Decouple connections and requests
New filter interfaces
Run the same filter logic on both systems

New Filter Interface

Sync:

```
O apply(I message);
```

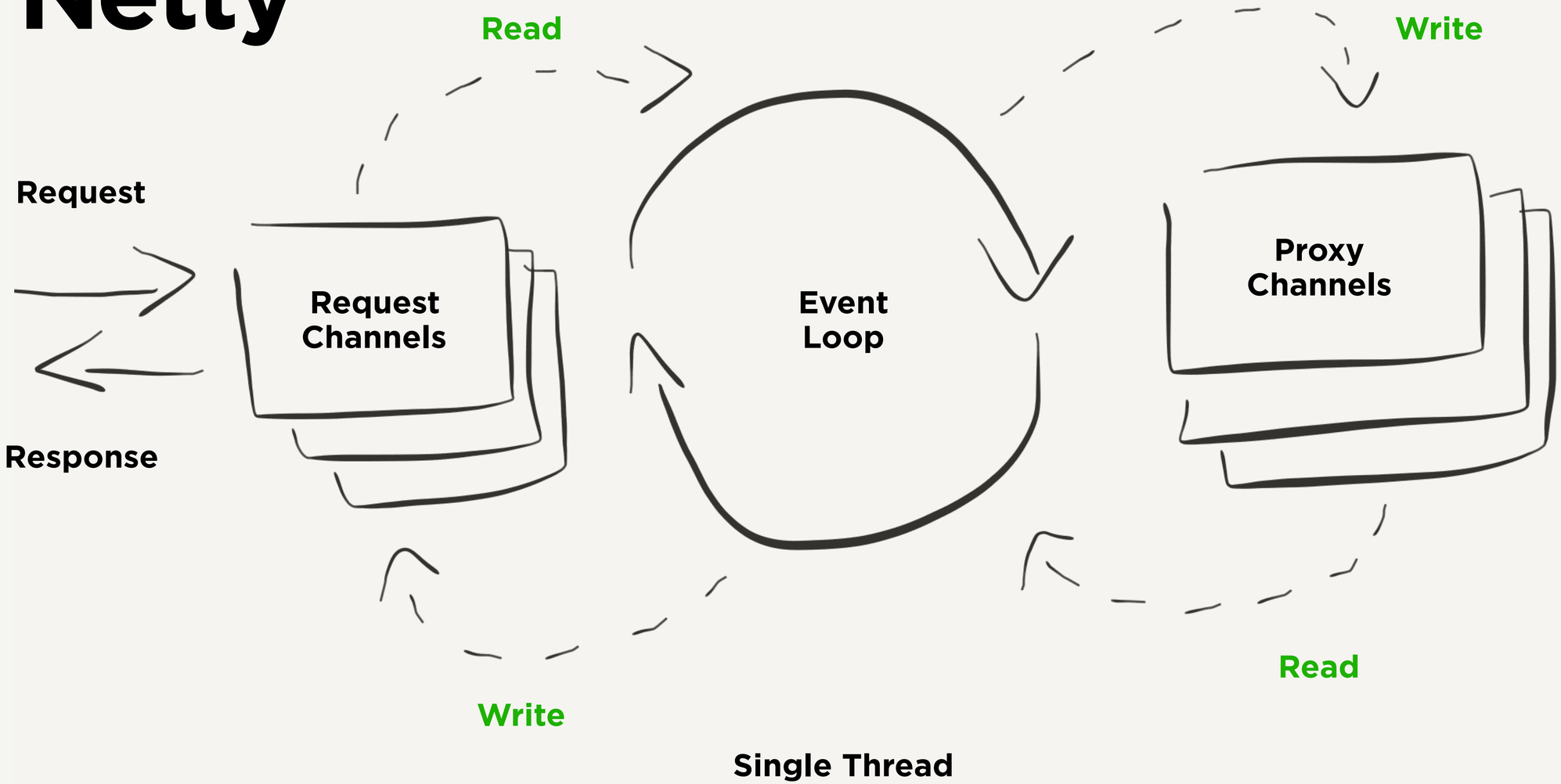
Async:

```
Observable<O> applyAsync(I message);
```

Observable Filter Chain

```
Observable.of(request)
          .flapMap(msg -> applyFilterChain(msg))
          .flatMap(msg -> writeResponse(msg))
          .doOnError(e -> writeErrorResponse(e))
          .doOnNext(msg -> postRequestMetrics(msg));
```

Netty


Read
Write
Request
Request Channels
Event Loop
Proxy Channels
Response
Write
Read
Single Thread

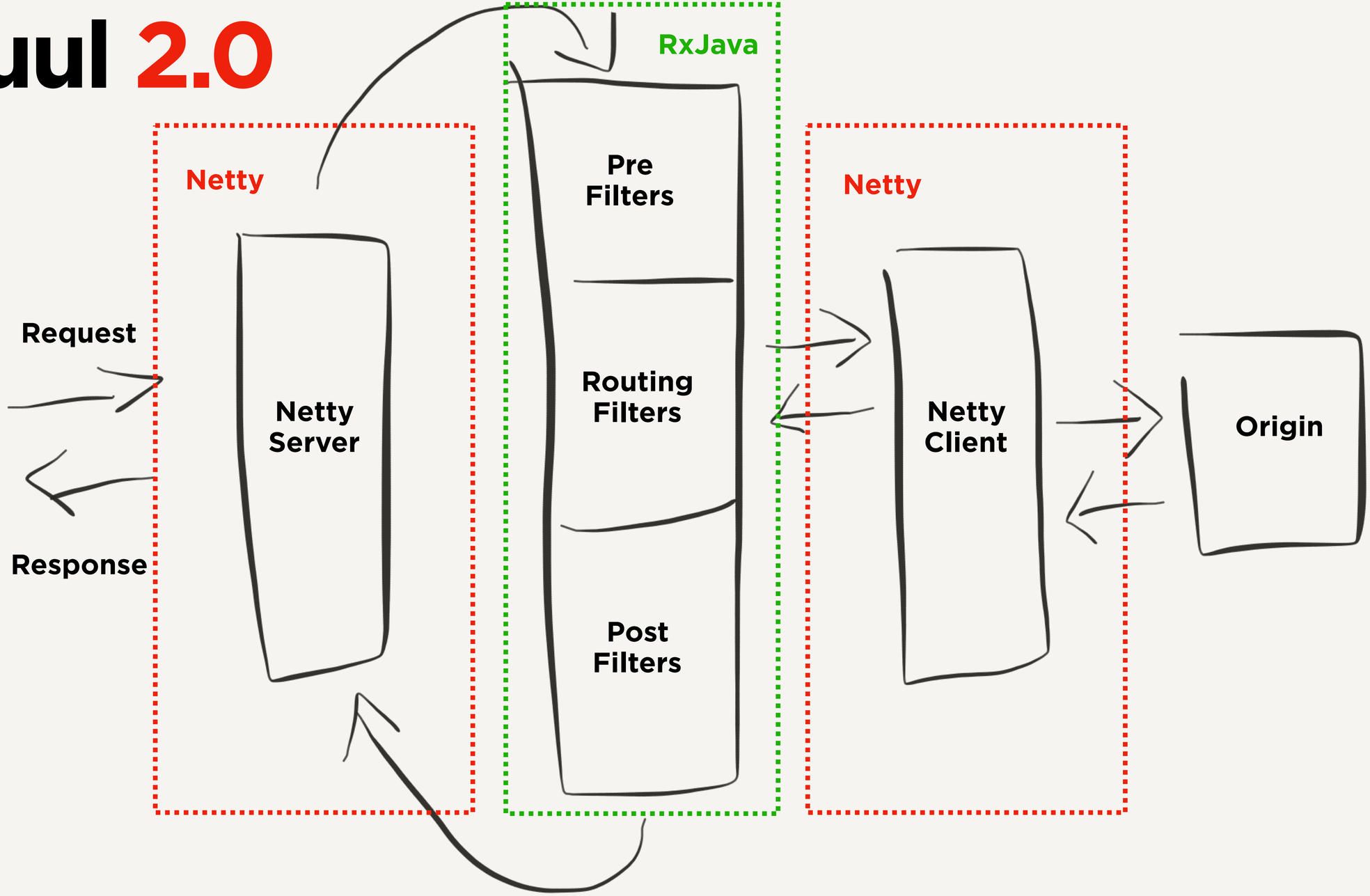
Building the Core

Adapters between Netty and RxJava
Adapters to wrap ByteBuf to Observable
Maintain the same functionality and metrics

Netty Client

Remove blocking client
Write new Netty-based client
Maintain existing load balancing algorithms

Zuul 2.0


RxJava
Netty
Pre
Filters
Netty
Request
Netty
Server
Routing
Filters
Netty
Client
Origin
Response
Post
Filters

Initial Hurdles

Operating Zuul 2.0 in the Real World

Underlying Assumptions

Unexpected blocking

Libraries built with assumption of blocking

Reactive Audit to the rescue!

Reactive Audit Example

```
HIGH   : Call method void java.lang.Thread.sleep(long)
com.netflix.reactive.audit.lib.CPUReactiveAuditException:
Call method void java.lang.Thread.sleep(long)
        at thread "Salamander-ClientToZuulWorker-3"
        at com.netflix.numerus.NumerusRollingNumber.getCurrentBucket(NumerusRollingNumber.java:330)
        at com.netflix.numerus.NumerusRollingNumber.getRollingSum(NumerusRollingNumber.java:161)
        at com.netflix.zuul.origins.InstrumentedOrigin.getErrorPercentage(InstrumentedOrigin.java:190)
        at com.netflix.zuul.origins.OriginDeviceRetryThrottle.shouldThrottle(OriginDeviceRetryThrottle.java:45)
        at com.netflix.zuul.origins.InstrumentedOrigin.request(InstrumentedOrigin.java:229)
        at com.netflix.zuul.filters.endpoints.NfProxyEndpoint.lambda$applyAsync$1(NfProxyEndpoint.java:46)
        at rx.internal.operators.OnSubscribeMap$MapSubscriber.onNext(OnSubscribeMap.java:69)
        at rx.internal.operators.OnSubscribeMap$MapSubscriber.onNext(OnSubscribeMap.java:77)
        at rx.internal.util.ScalarSynchronousObservable$WeakSingleProducer.request(ScalarSynchronousObservable.java:276)
        at rx.Subscriber.setProducer(Subscriber.java:211)
        at rx.internal.operators.OnSubscribeMap$MapSubscriber.setProducer(OnSubscribeMap.java:102)
        at rx.internal.operators.OnSubscribeMap$MapSubscriber.setProducer(OnSubscribeMap.java:102)
        at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:138)
        at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:129)
        at rx.Observable.unsafeSubscribe(Observable.java:10200)
        at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:48)
        at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:33)
        at rx.Observable.unsafeSubscribe(Observable.java:10200)
        at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:48)
```

More Underlying Assumptions

Libraries rely on thread locals

Shared state between libs in the thread

Time-consuming, manual inspection

Debugging Async Requests

Tracing request failures
Comprehending stack traces
Timing asynchronous requests

Request Passport

Thanks Zach Tellman and Strange Loop!
Maintain "passport" on the channel
Add entries with state and timestamp
Make passports available on failure

Passport Example

+0=IN_REQ_HEADERS_RECEIVED,
+705017=FILTERS_INBOUND_START,
+1949958=MISC_IO_START,
+2027915=IN_REQ_LAST_CONTENT_RECEIVED,
+3391350=MISC_IO_STOP,
+4767202=ORIGIN_CONN_ACQUIRE_START,
+4784096=CLIENT_CH_CONNECTING,
+4891510=FILTERS_INBOUND_END,
+5748769=CLIENT_CH_CONNECTED,
+5773298=ORIGIN_CONN_ACQUIRE_END,
+6336479=OUT_REQ_HEADERS_SENDING,
+6388730=OUT_REQ_HEADERS_SENT,
+6402264=OUT_REQ_LAST_CONTENT_SENDING,
+6408016=OUT_REQ_LAST_CONTENT_SENT,
+6416189=CLIENT_CH_ACTIVE,
+45006120586=CLIENT_CH_CLOSE,
+45006221594=CLIENT_CH_READ_TIMEOUT,
+45006359072=ORIGIN_CONN_ACQUIRE_START,
+45006385636=CLIENT_CH_CONNECTING,
+45006492694=CLIENT_CH_CLOSE,
+45006907204=CLIENT_CH_CONNECTED,
+45006928564=ORIGIN_CONN_ACQUIRE_END,
+45007638281=OUT_REQ_HEADERS_SENDING,
+45007690499=OUT_REQ_HEADERS_SENT,
+45007708575=OUT_REQ_LAST_CONTENT_SENDING,
+45007716697=OUT_REQ_LAST_CONTENT_SENT,
+45007725840=CLIENT_CH_ACTIVE,
+59920094115=SERVER_CH_INACTIVE,
+59920182824=IN_REQ_CANCELLED,
+90007646473=CLIENT_CH_CLOSE,
+90007777157=CLIENT_CH_READ_TIMEOUT,
+90007965586=FILTERS_OUTBOUND_START,
+90008909963=FILTERS_OUTBOUND_END,
+90009003762=NOW

Long Tail Problems

Operating Zuul 2.0 in the Real World

Direct Memory Leaks

Servers slowly use more memory
Eventually RPS drops to zero
Extremely difficult to debug

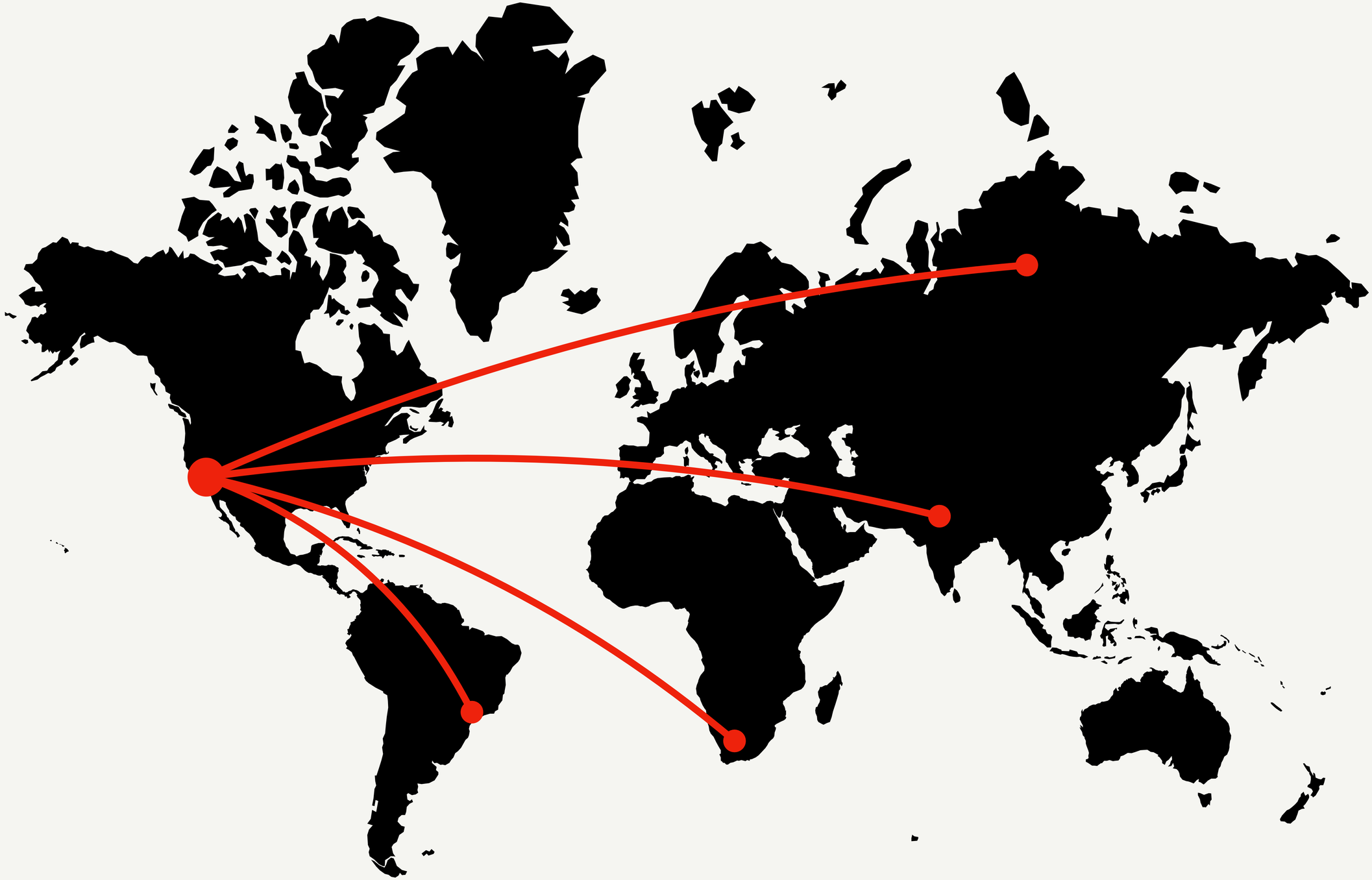
What's going on here?

Analyzing async code is hard

Throw in RxJava for even more fun

Stack traces become useless

```
java.lang.IllegalStateException: Demo exception.
  at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
  at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
  at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
  at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
  at org.codehaus.groovy.reflection.CachedConstructor.invoke(CachedConstructor.java:80)
  at org.codehaus.groovy.runtime.callsite.ConstructorSite$ConstructorSiteNoUnwrapNoCoerce.callConstructor(ConstructorSite.java:105)
  at org.codehaus.groovy.runtime.callsite.CallSiteArray.defaultCallConstructor(CallSiteArray.java:60)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:235)
  at org.codehaus.groovy.runtime.callsite.AbstractCallSite.callConstructor(AbstractCallSite.java:247)
  at global.ThreadChannelMismatchTrackerOutbound.apply(ThreadChannelMismatchTrackerOutbound.groovy:23)
  at global.ThreadChannelMismatchTrackerOutbound.apply(ThreadChannelMismatchTrackerOutbound.groovy)
  at com.netflix.zuul.FilterProcessorImpl.processSyncFilter(FilterProcessorImpl.java:388)
  at com.netflix.zuul.FilterProcessorImpl.chainFilters(FilterProcessorImpl.java:122)
  at com.netflix.zuul.FilterProcessorImpl.lambda$chainFilters$1(FilterProcessorImpl.java:133)
  at rx.internal.operators.OnSubscribeMap$MapSubscriber.onNext(OnSubscribeMap.java:69)
  at rx.internal.operators.OnSubscribeDoOnEach$DoOnEachSubscriber.onNext(OnSubscribeDoOnEach.java:101)
  at rx.internal.operators.OnSubscribeMap$MapSubscriber.onNext(OnSubscribeMap.java:77)
  at rx.internal.operators.OperatorOnErrorResumeNextViaFunction$4.onNext(OperatorOnErrorResumeNextViaFunction.java:154)
  at rx.internal.util.ScalarSynchronousObservable$WeakSingleProducer.request(ScalarSynchronousObservable.java:276)
  at rx.internal.producers.ProducerArbiter.setProducer(ProducerArbiter.java:126)
  at rx.internal.operators.OperatorOnErrorResumeNextViaFunction$4.setProducer(OperatorOnErrorResumeNextViaFunction.java:159)
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:138)
  at rx.internal.util.ScalarSynchronousObservable$JustOnSubscribe.call(ScalarSynchronousObservable.java:129)
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48)
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:30)
  at rx.Observable.unsafeSubscribe(Observable.java:10256)
  at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:48)
  at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:33)
  at rx.Observable.unsafeSubscribe(Observable.java:10256)
  at rx.internal.operators.OnSubscribeDoOnEach.call(OnSubscribeDoOnEach.java:41)
  at rx.internal.operators.OnSubscribeDoOnEach.call(OnSubscribeDoOnEach.java:30)
  at rx.Observable.unsafeSubscribe(Observable.java:10256)
  at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:48)
  at rx.internal.operators.OnSubscribeMap.call(OnSubscribeMap.java:33)
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:48)
  at rx.internal.operators.OnSubscribeLift.call(OnSubscribeLift.java:30)
```



Geo Shuffle

CDN team gets alerted to location issues

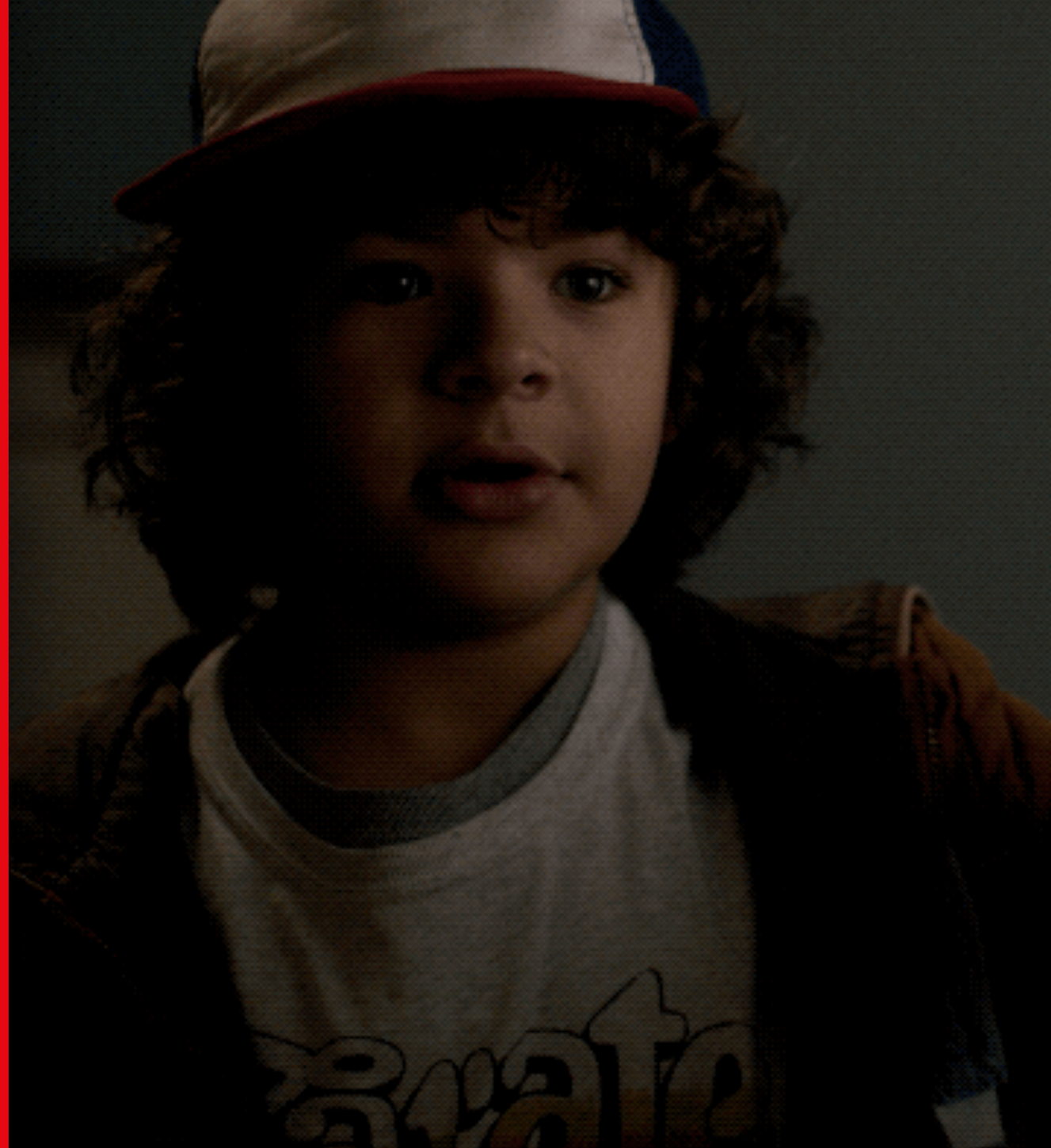
We're sending people across the globe

Thread Locals! Argh!

Reset thread locals when jumping threads

Long Term Benefits

Operating Zuul 2.0
in the Real World

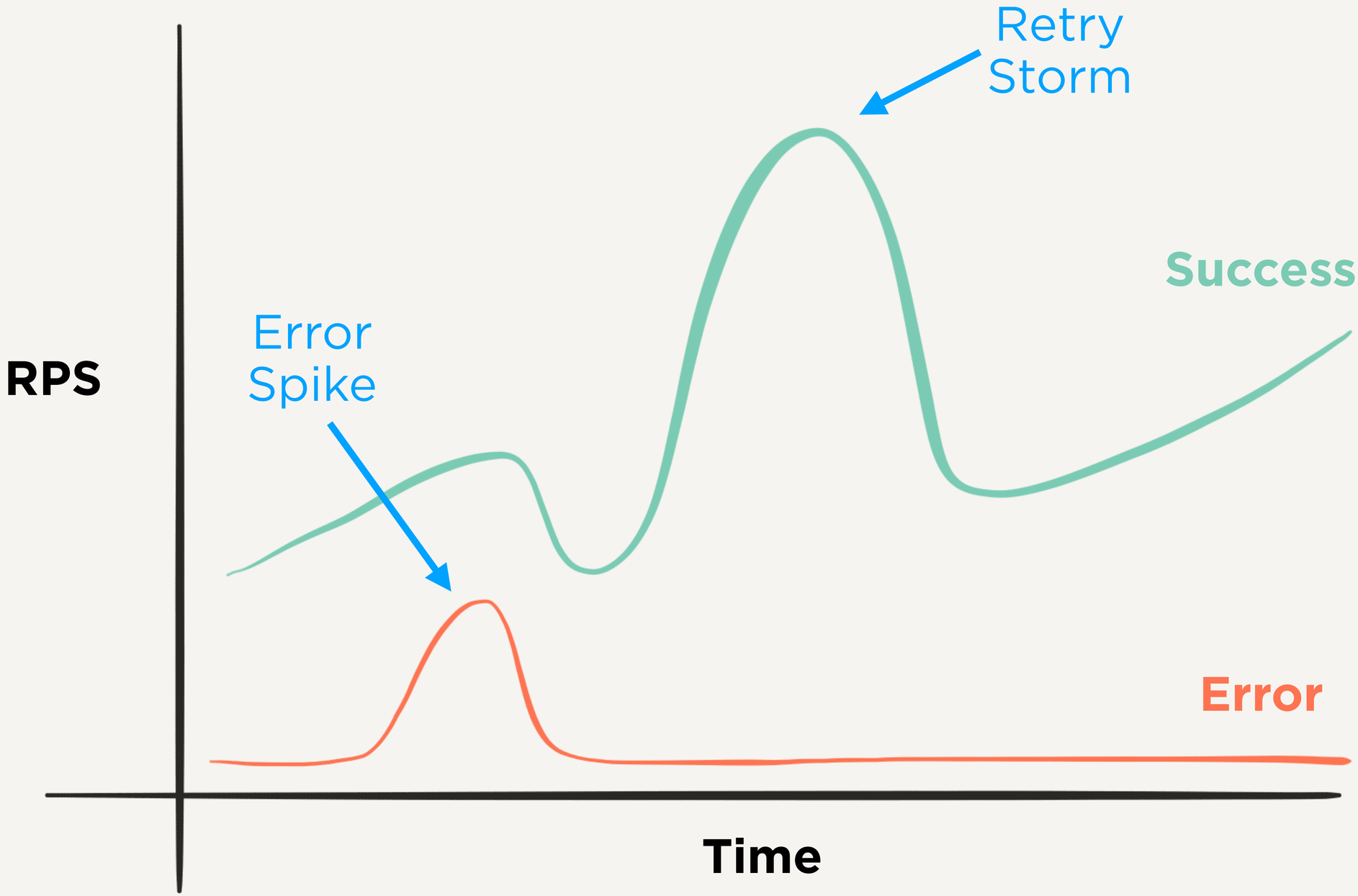


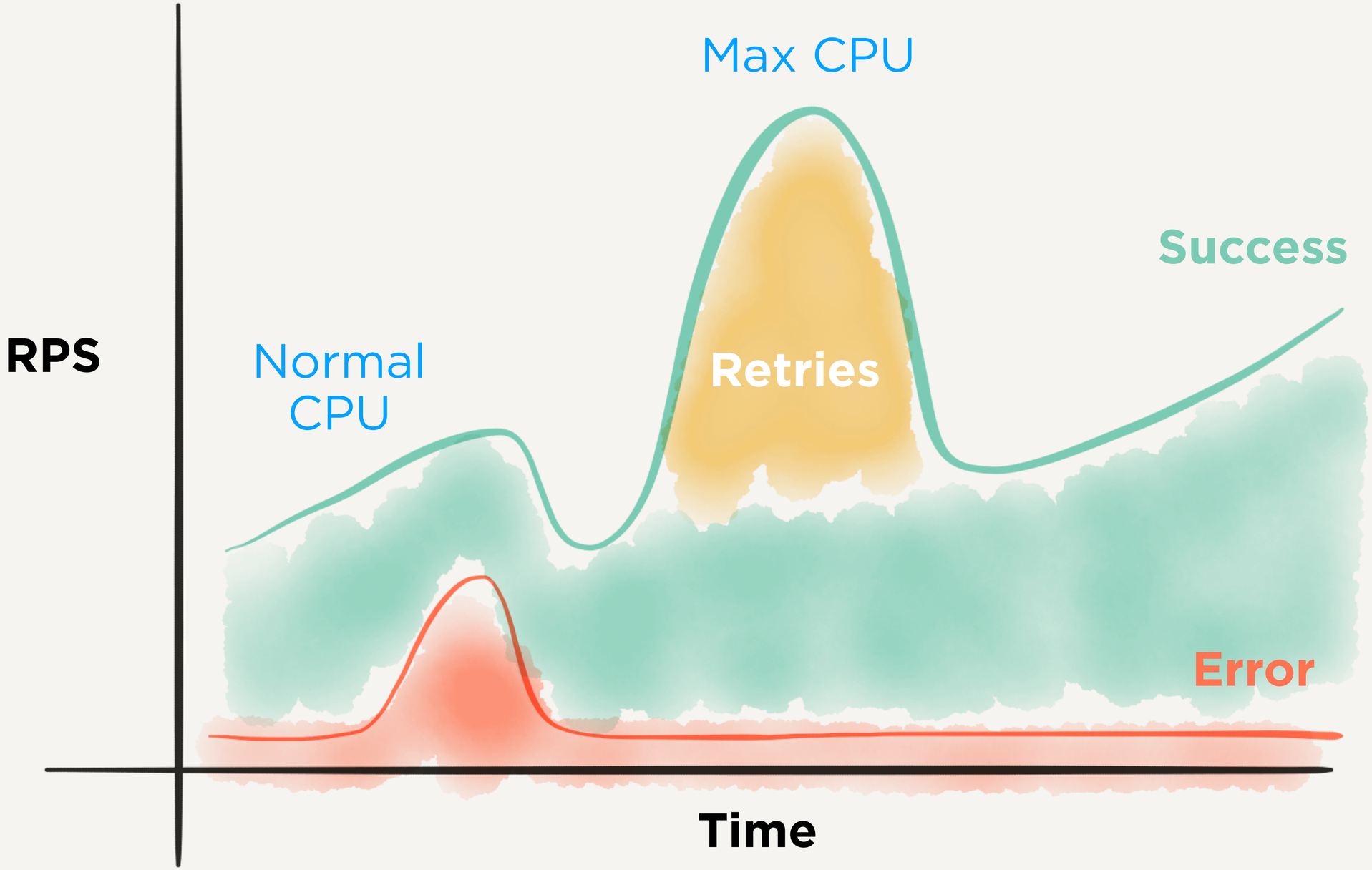
Weathering the Storm

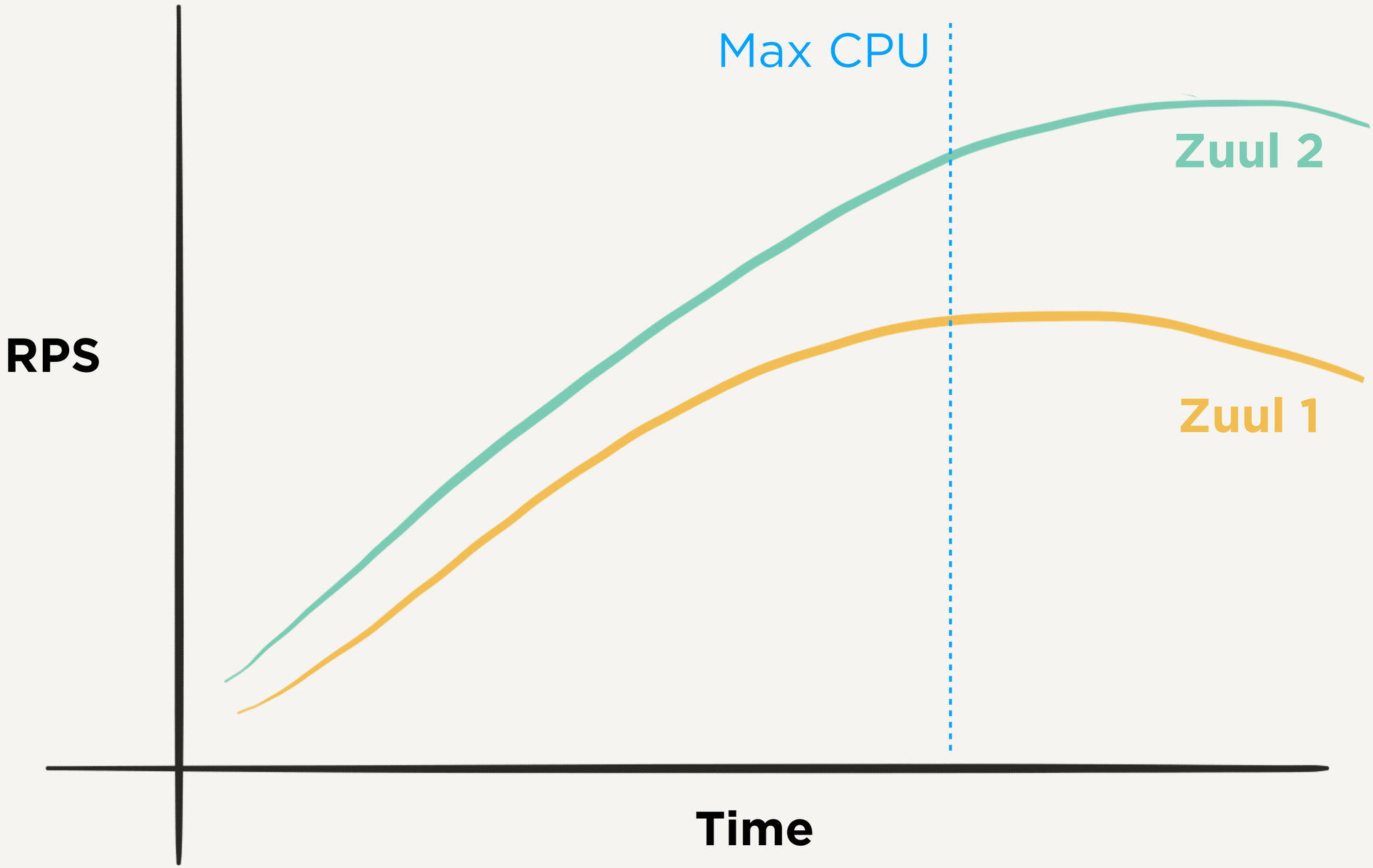
Fully up during massive retry storm

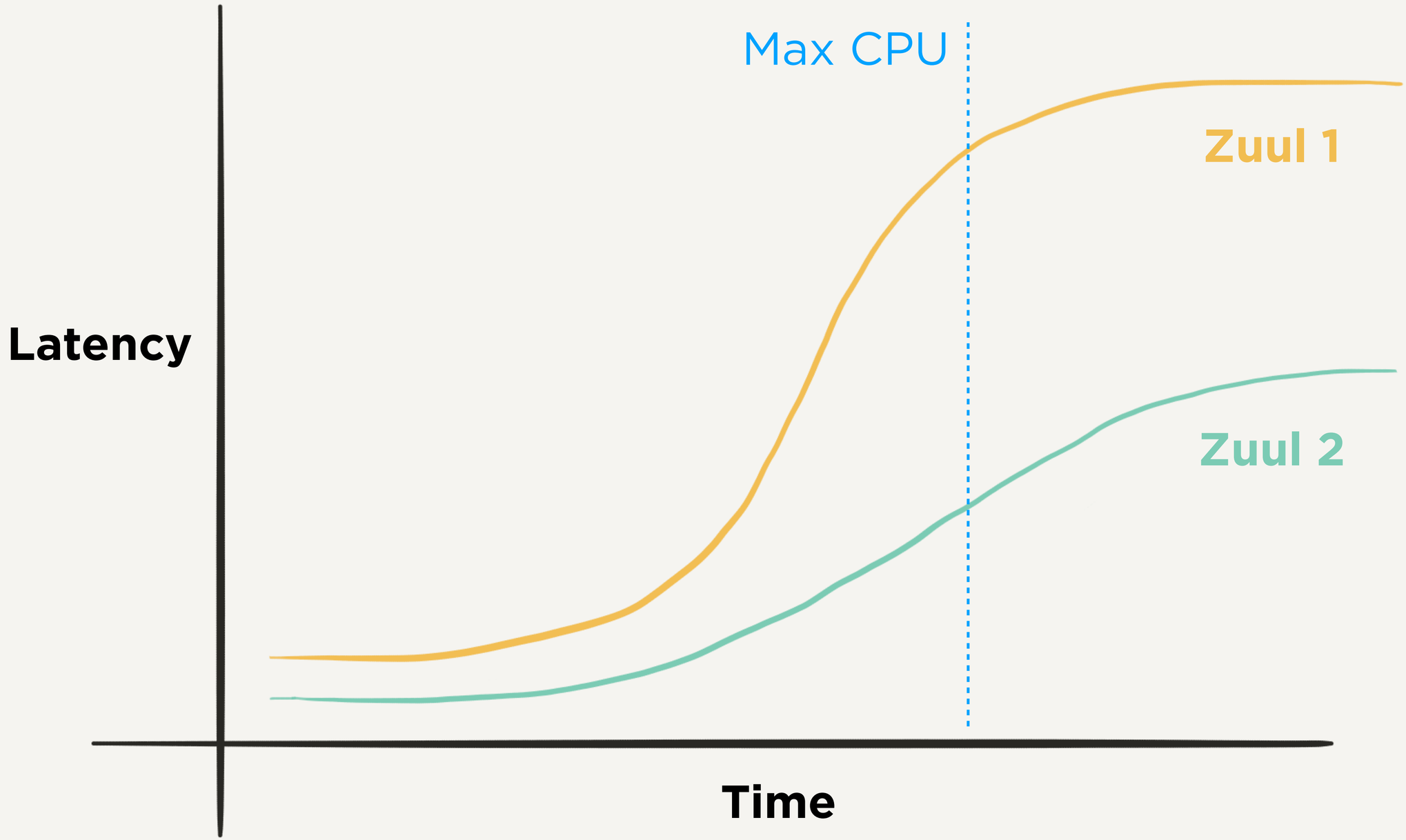
Continued serving 200s at max CPU

What is a retry storm?


Retry
Storm
Success
Error
Spike
RPS
Error
Time


Max CPU
Success
RPS
Normal
CPU
Retries
Error
Time


Max CPU
Zuul 2
Zuul 1
RPS
Time


Max CPU
Zuul 1
Zuul 2
Latency
Time

Resiliency Benefits

Allows Zuul to withstand huge spikes

Shields origins from retry storms

Prevents throttling origins from going underwater

Long Lived Connections

Tens of thousands of connections per node
Serving tens of millions of concurrent clients
Websocket support for push notifications
Allows for new features for partner teams

Relaxed Latency Contraints

Timeouts are no longer strictly guarded
Can be configured by origins or devices
Hystrix circuit breaking not necessary
Zuul will operate consistently regardless

Blocking or Non-Blocking

Which one is right for you?

Real World Performance

Request per second per node

Responsibilities

Encryption

Geo Data Lookup

Authentication

Static Response Handling

Decryption

Spam Blocking

Connection Pooling

Hashing

Routing

Multi-Region Resiliency

DDoS Protection

Sticky Canaries

Security

Service Discovery

Request Context Population

Load Balancing

Mantis Real Time Events

Load Shedding

Stress Testing

Logging

Atlas Metrics

Canary Testing

Insights

Hive Long Term Storage

Dynamic Configuration Management

When to Block

Highly CPU-bound work loads
Desire operational simplicity
Desire development simplicity
Run legacy systems that are blocking

When Not to Block

Highly I/O-bound workloads
Long requests and/or large files
Streaming data from queues
Massive amounts of connections

Benchmark Accuracy

Important to set up realistic benchmarks
Simulate real-world workloads
Verify assumptions and trendy dogma

Long Term Costs

Account for lack of debuggability
Ongoing maintenance cost of async code
Support and operations burden

Critical Features

What features do you really need?
Are those features worth the costs?
What's your answer to "it depends"?

Was it worth it?

Thank you.

Office Hours: Netflix Booth 10am-12pm tomorrow

NETFLIX