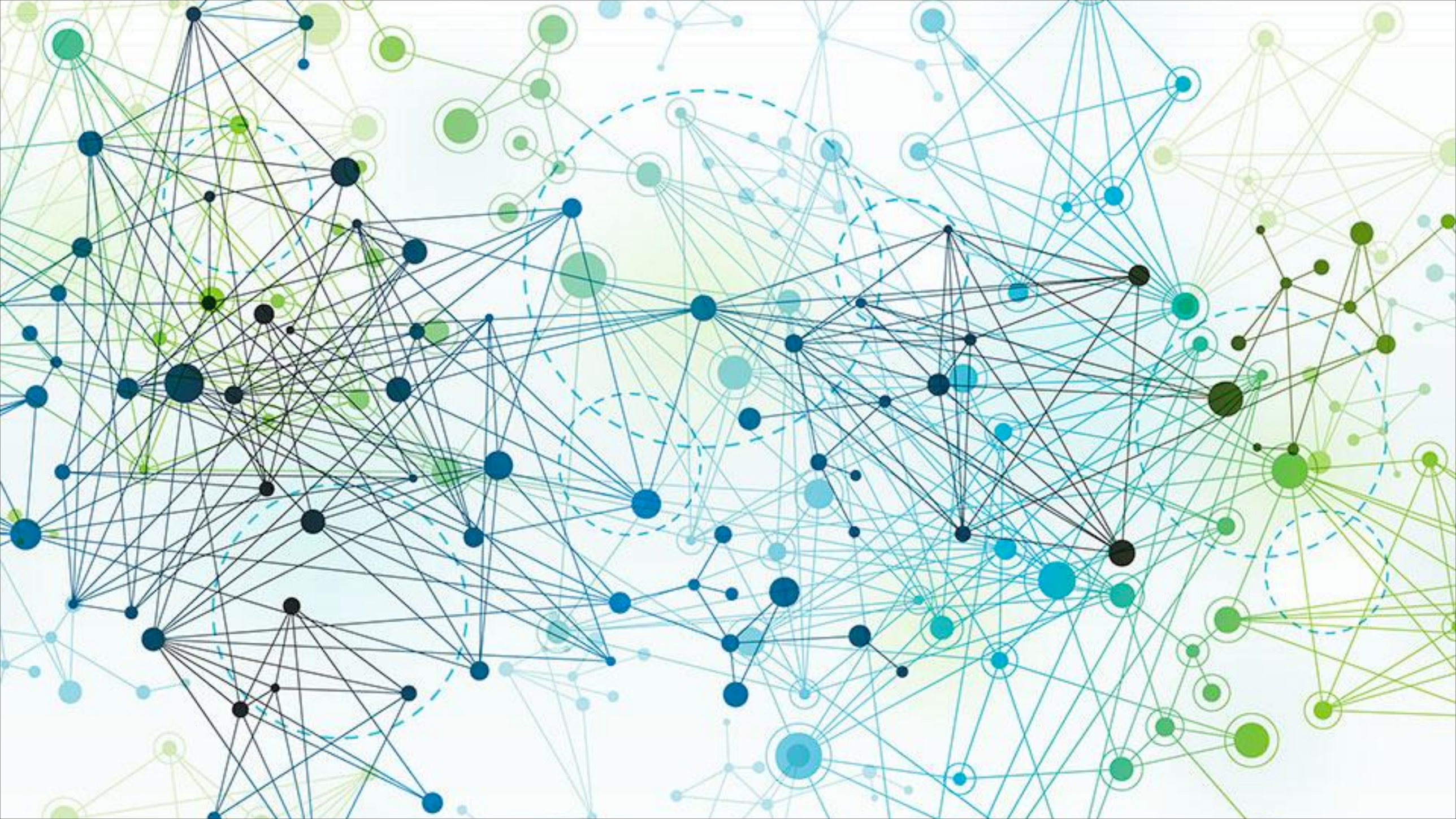
Crossroads of asynchrony and graceful degradation

Nitesh Kant,
Software Engineer,
Netflix Edge Engineering.

@NiteshKant




P4-GJC

Nitesh Kant *@NiteshKant*

Who Am I?

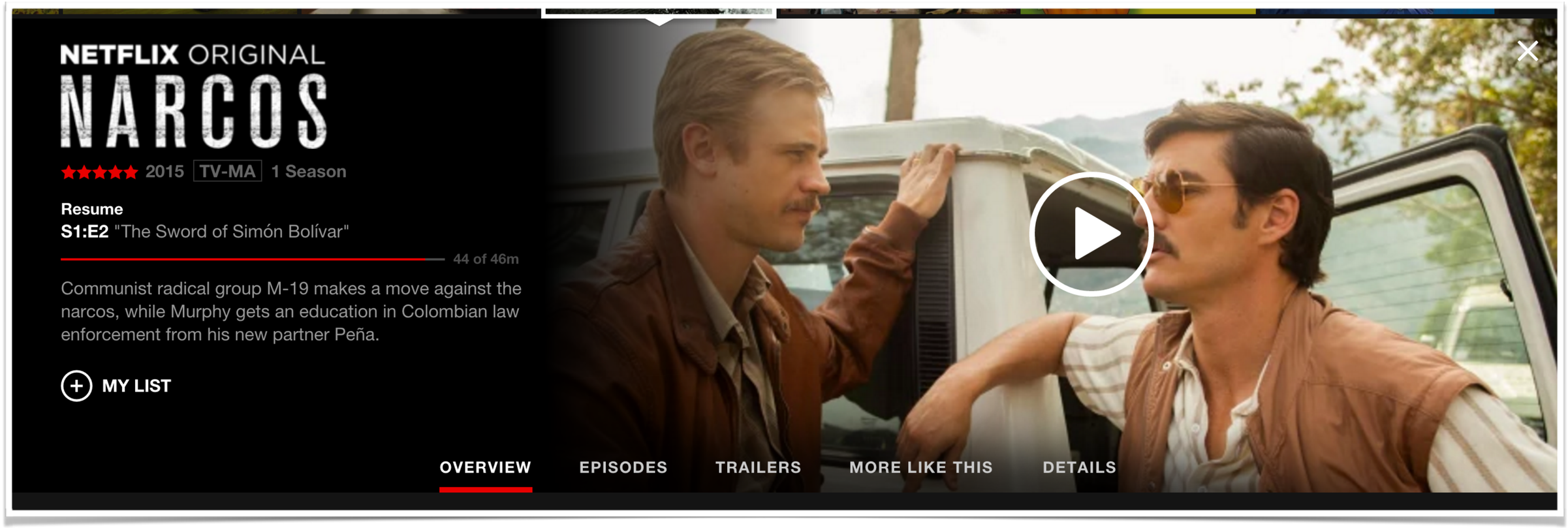
- Engineer, Edge Engineering, Netflix.
- Core contributor, RxNetty*
- Contributor, Zuul**

* https://github.com/ReactiveX/RxNetty

** https://github.com/Netflix/zuul

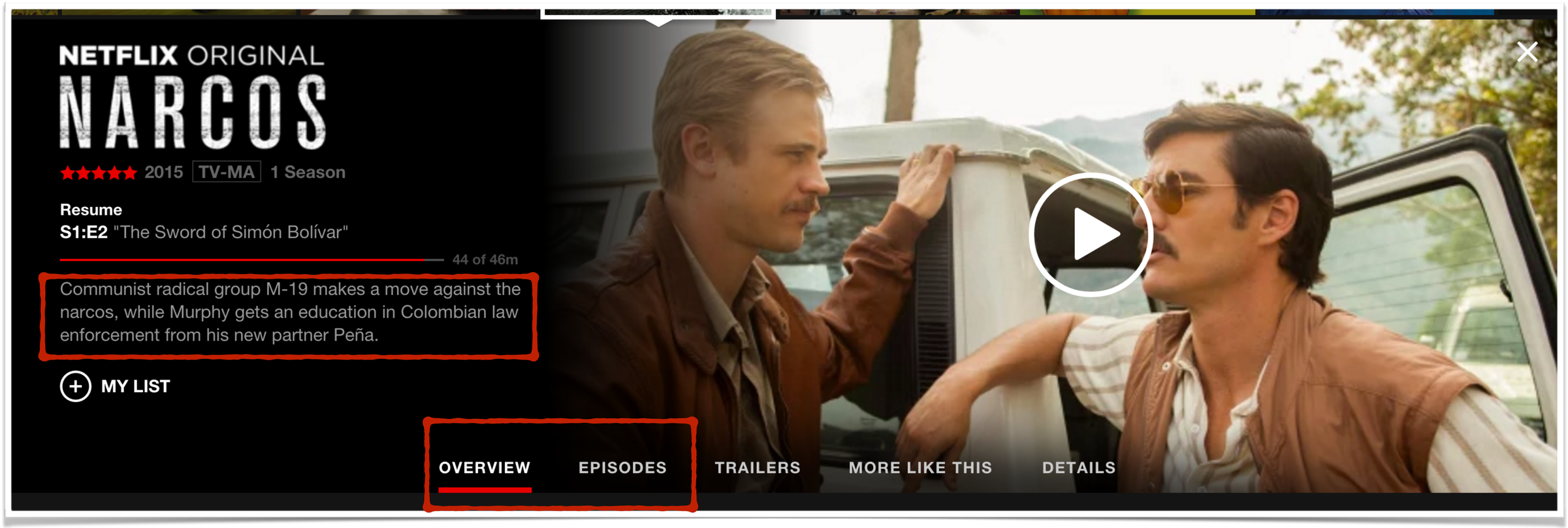
Graceful degradation is the ability of a computer, machine, electronic system or network to maintain limited functionality even when a large portion of it has been destroyed or rendered inoperative. The purpose of **graceful degradation** is to prevent catastrophic failure.

How do systems fail?

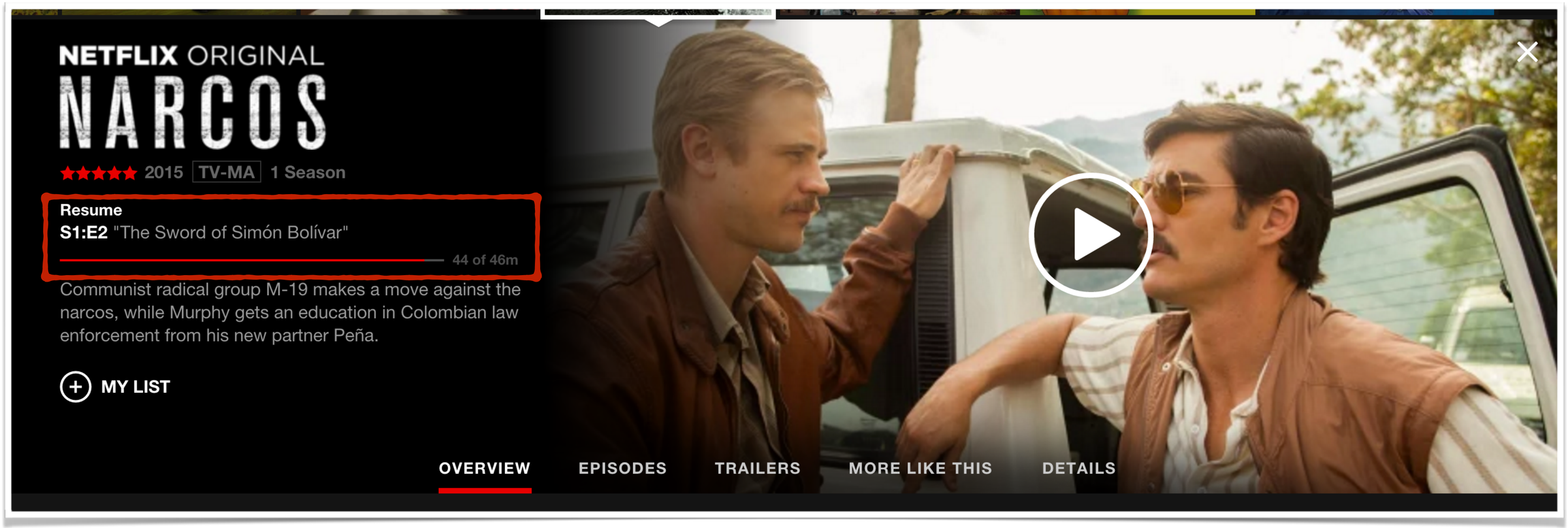


A simple example.

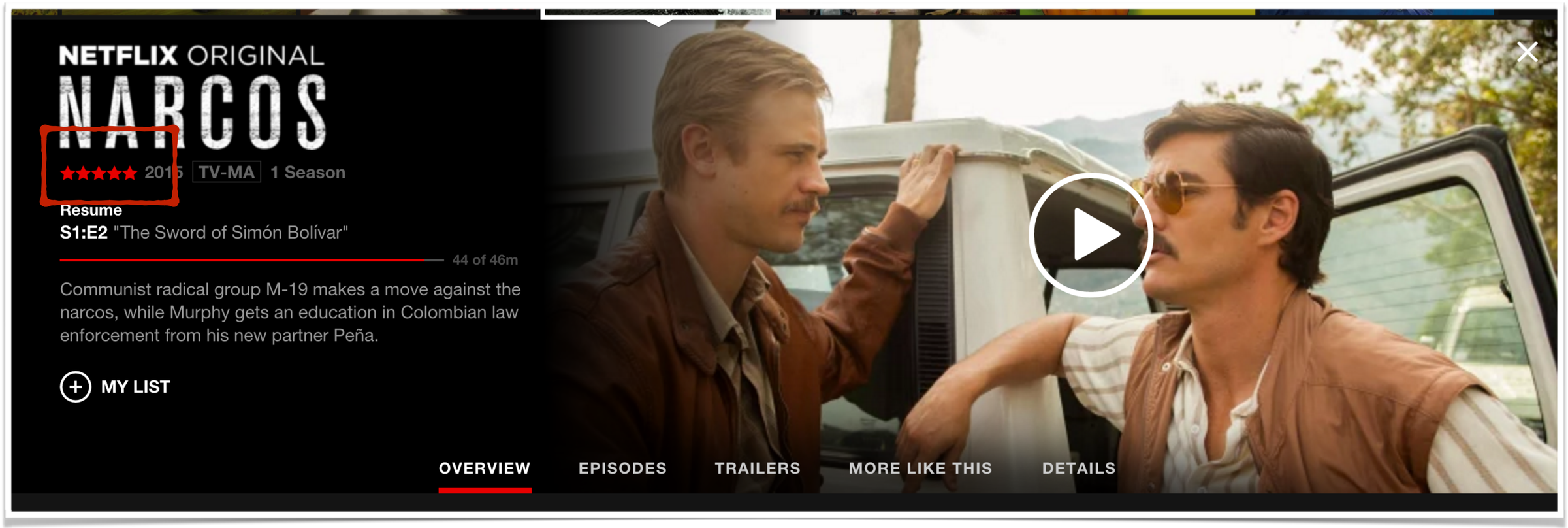
Showing a movie on Netflix.



Video Metadata



Video Bookmark



Video Rating

```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

Disclaimer: This is an example and not an exact representation of the processing

Synchronicity

```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

Disclaimer: This is an example and not an exact representation of the processing

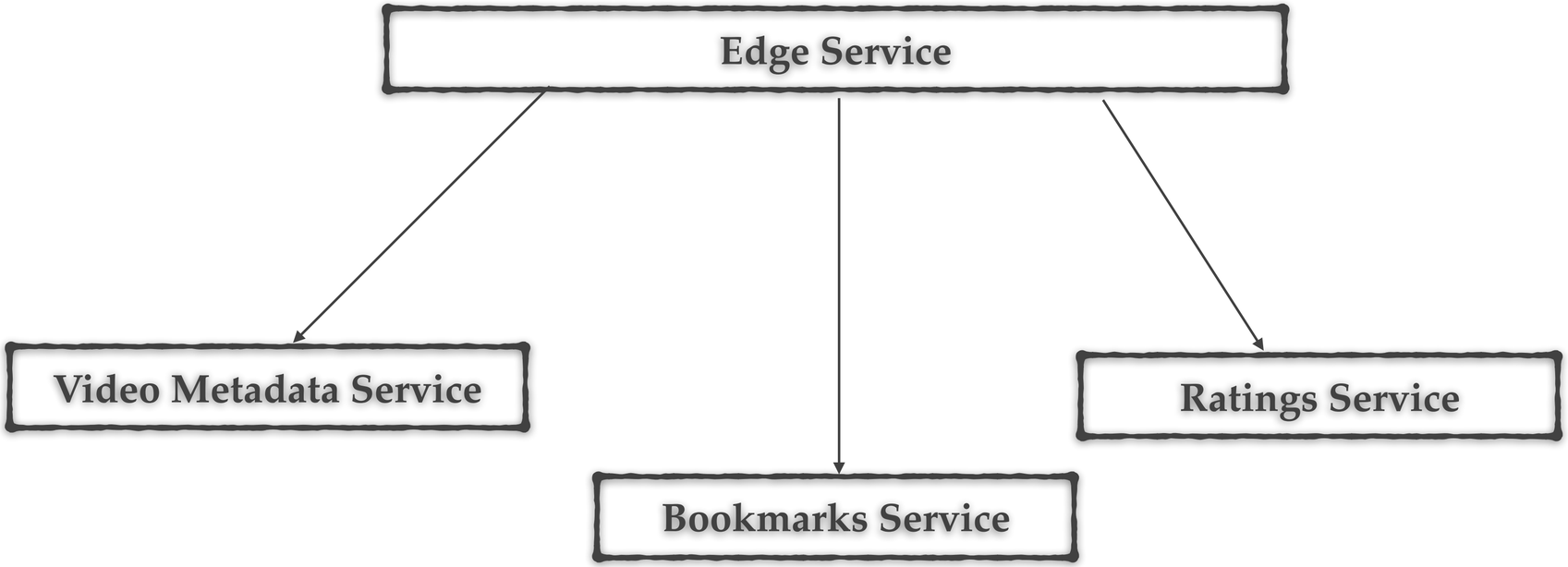
```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

The bigger picture

Price of being synchronous?

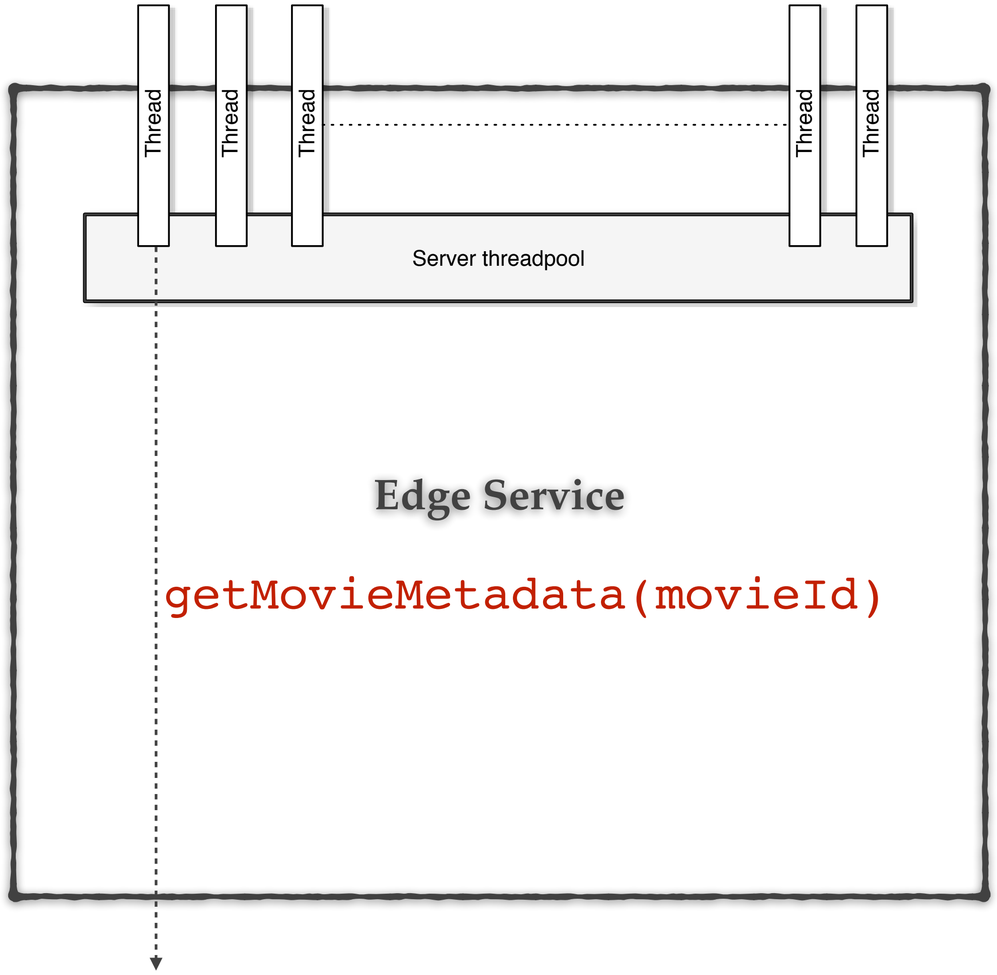
Disclaimer: This is an example and not an exact representation of the processing

In a microservices world



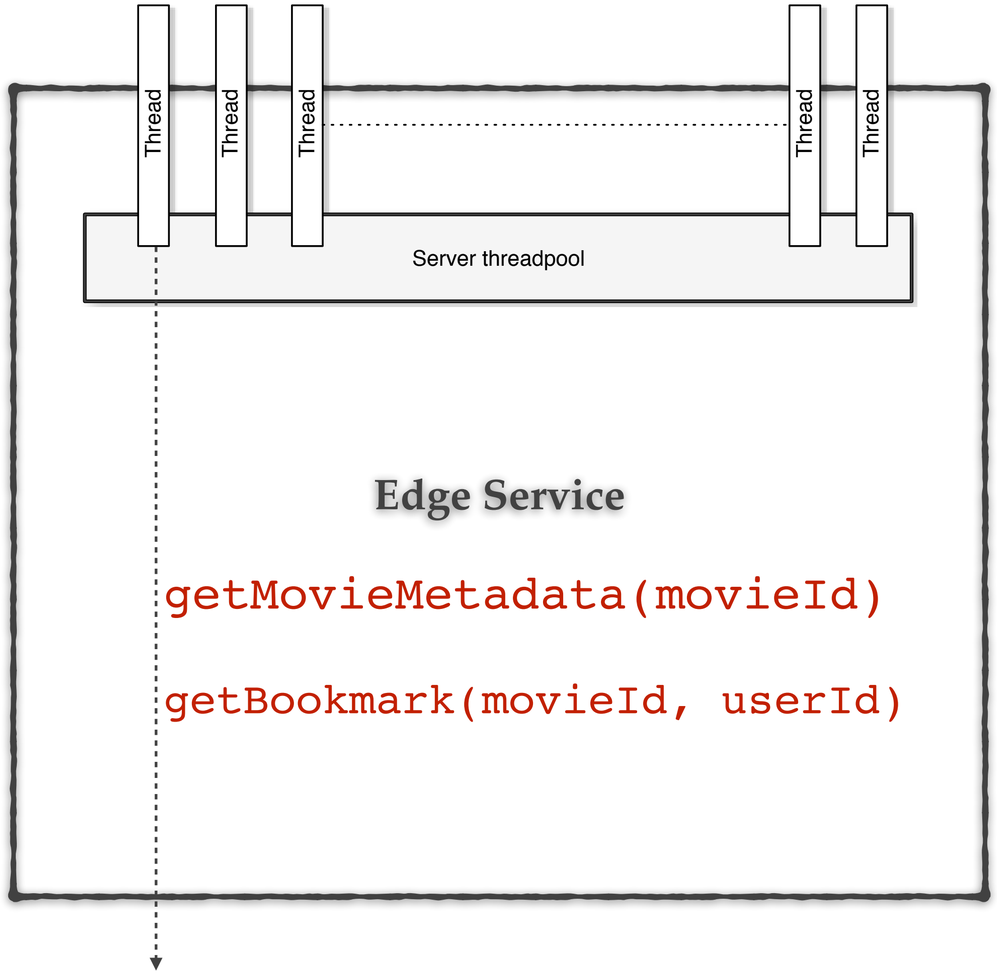
Disclaimer: This is an example and not an exact representation of the processing

In a microservices world



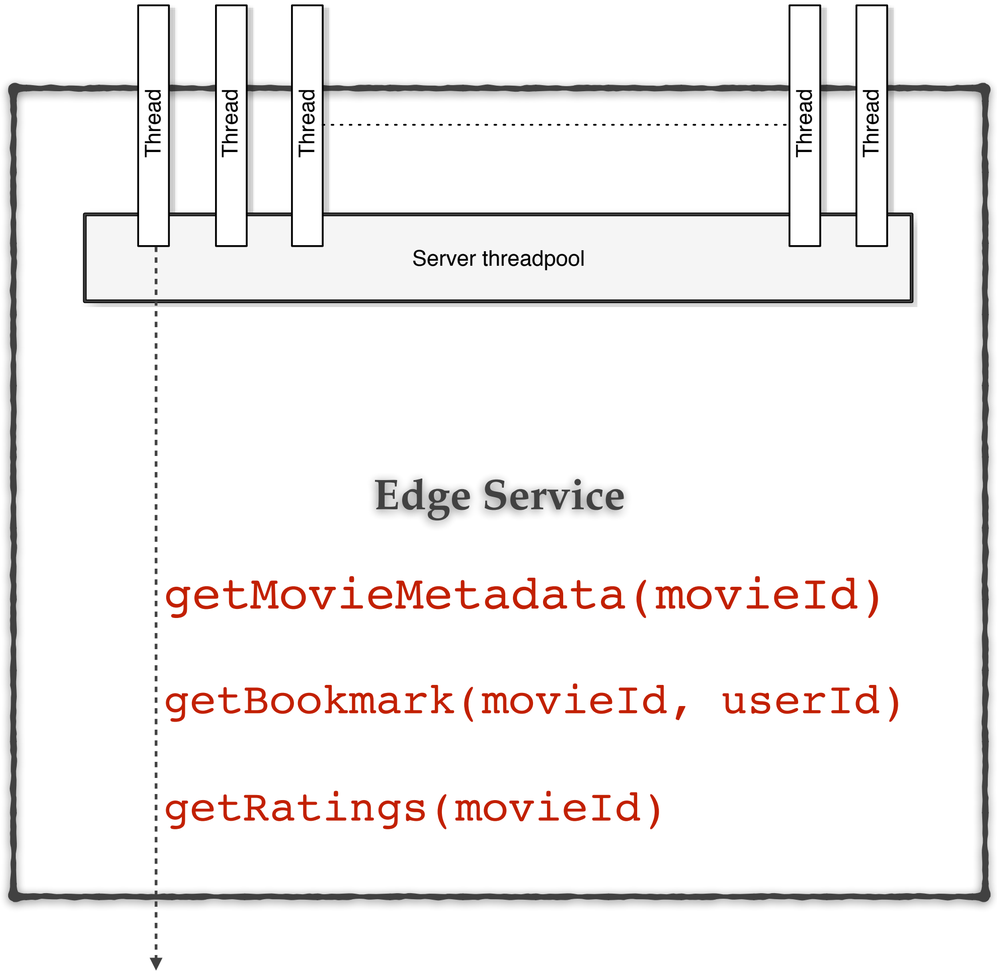
Disclaimer: This is an example and not an exact representation of the processing

In a microservices world



Disclaimer: This is an example and not an exact representation of the processing

In a microservices world



Disclaimer: This is an example and not an exact representation of the processing

Busy thread time

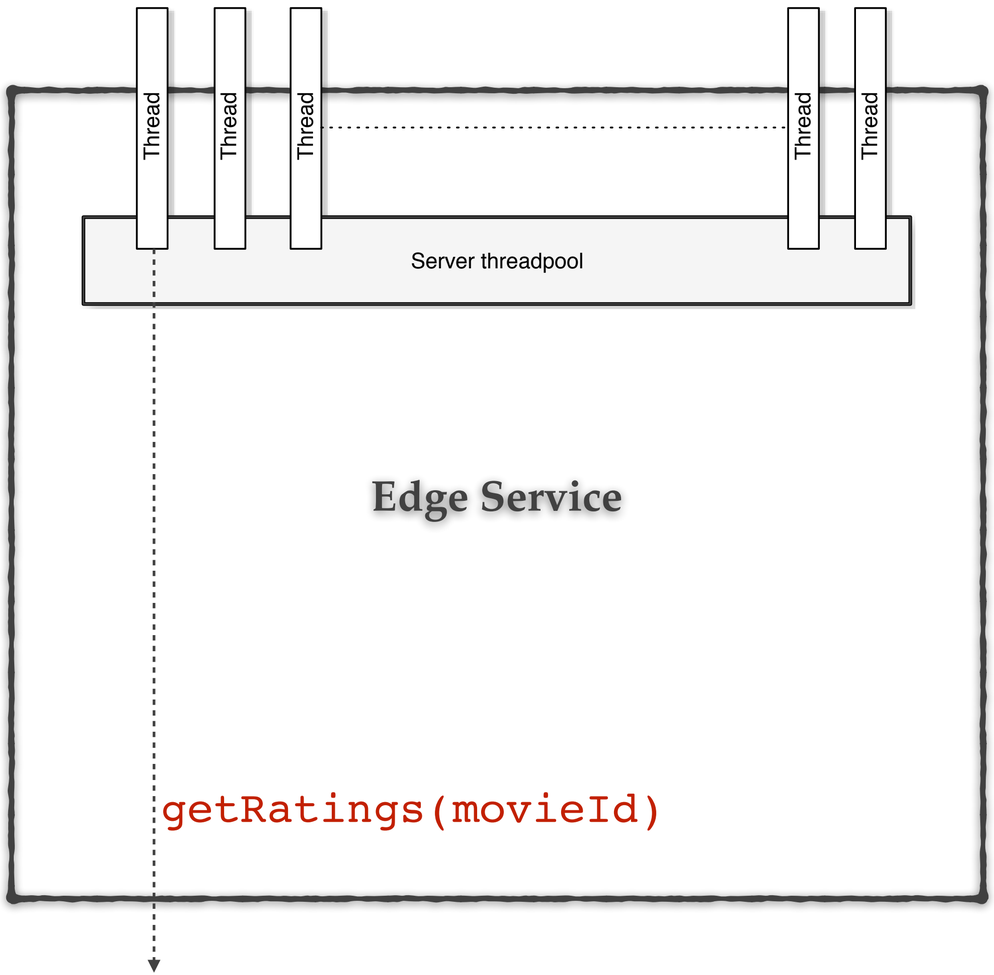
=

Sum of the time taken to
make all 3 service calls

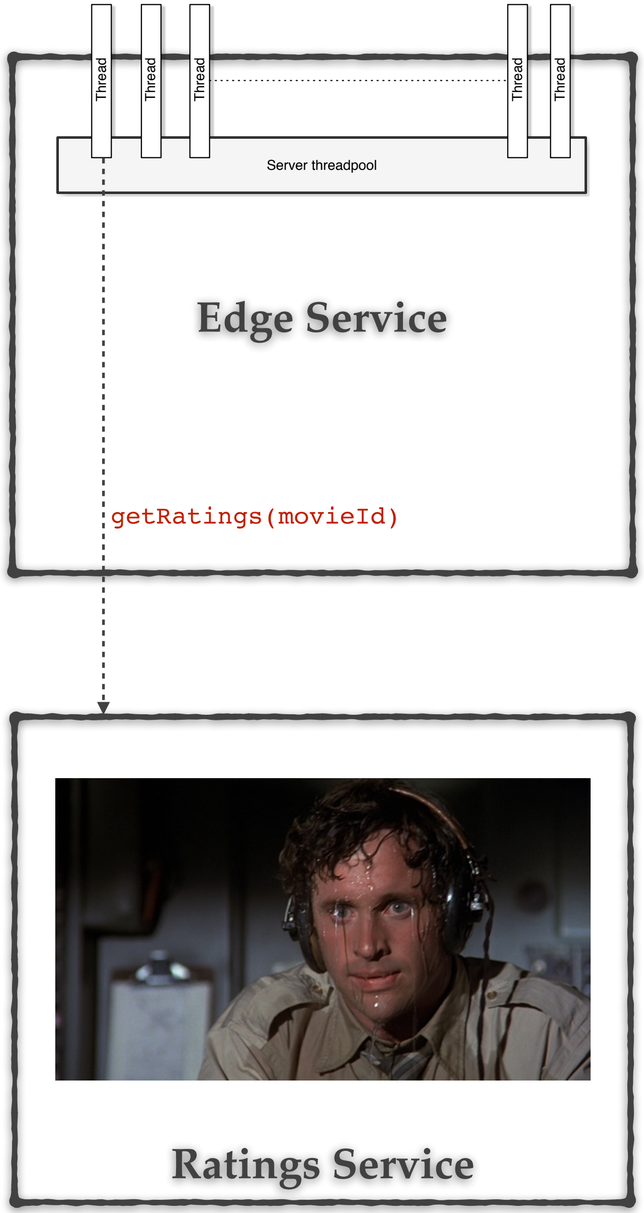
How do systems fail?

1. Latency

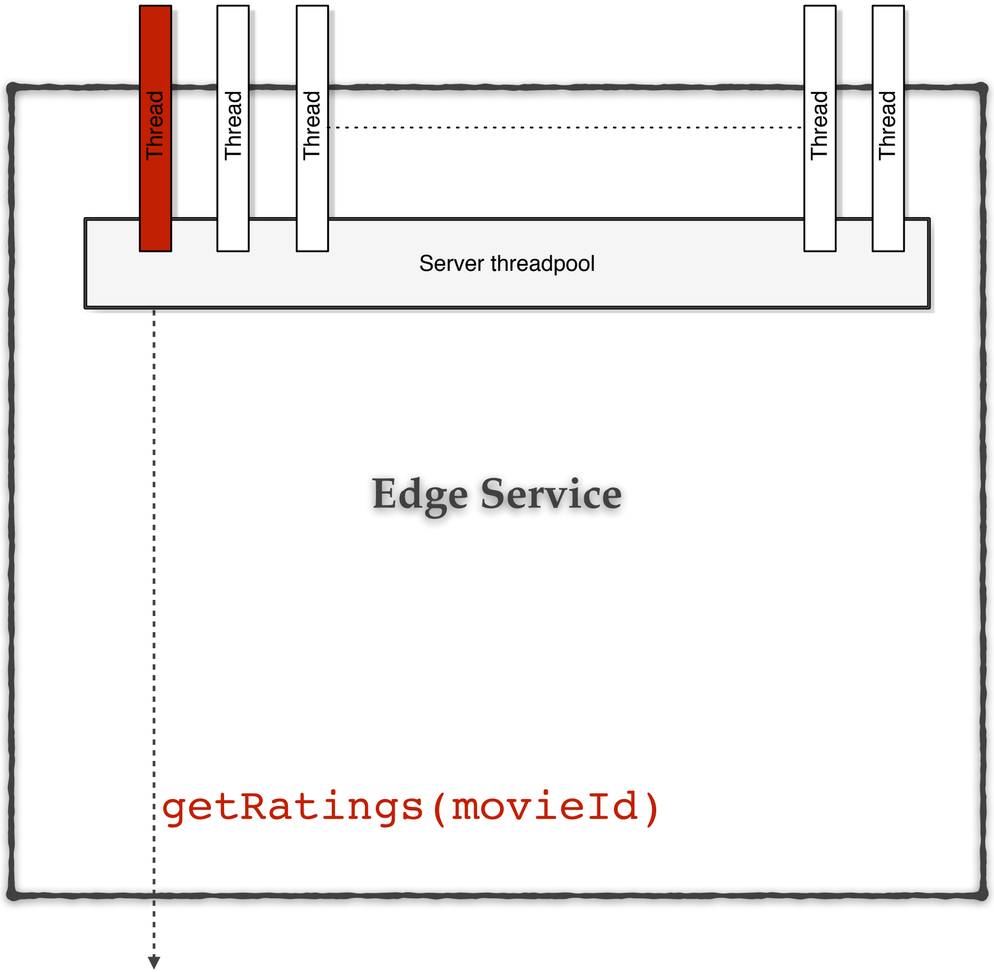
Latency is your worst enemy in a synchronous world.



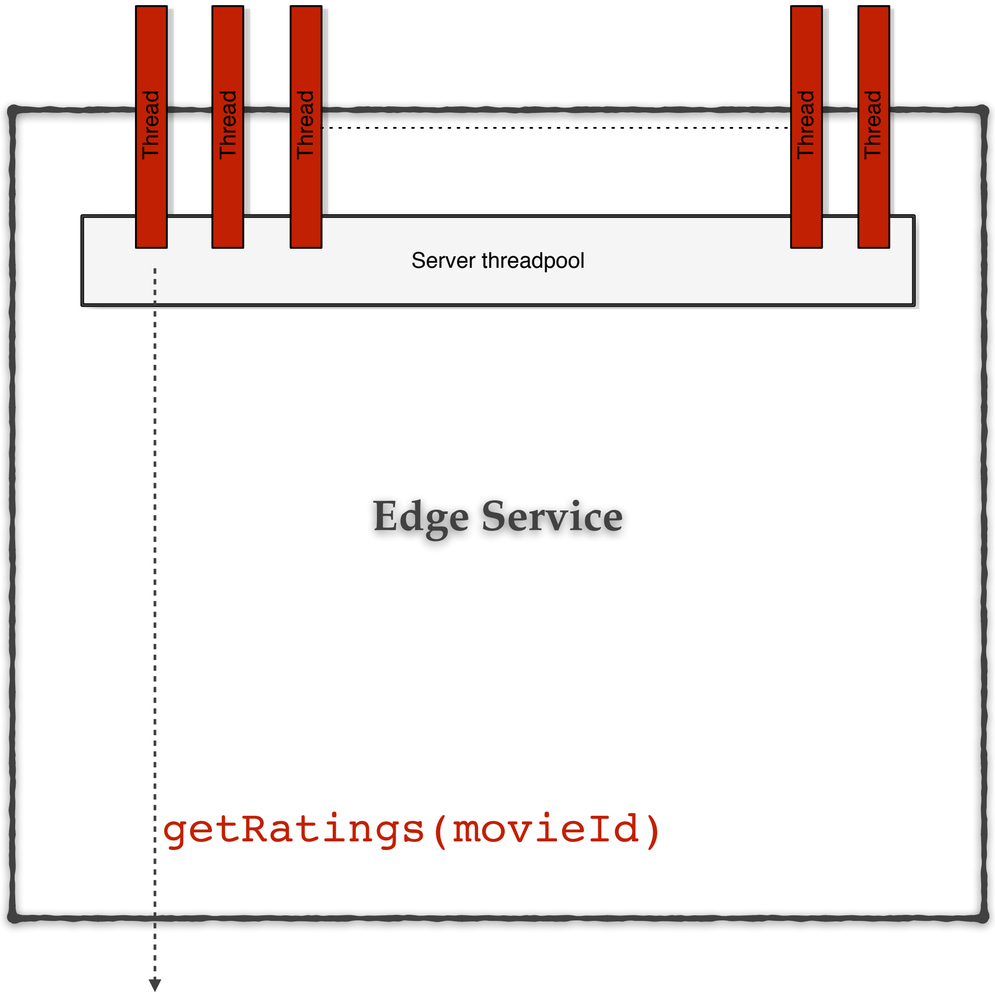
Disclaimer: This is an example and not an exact representation of the processing



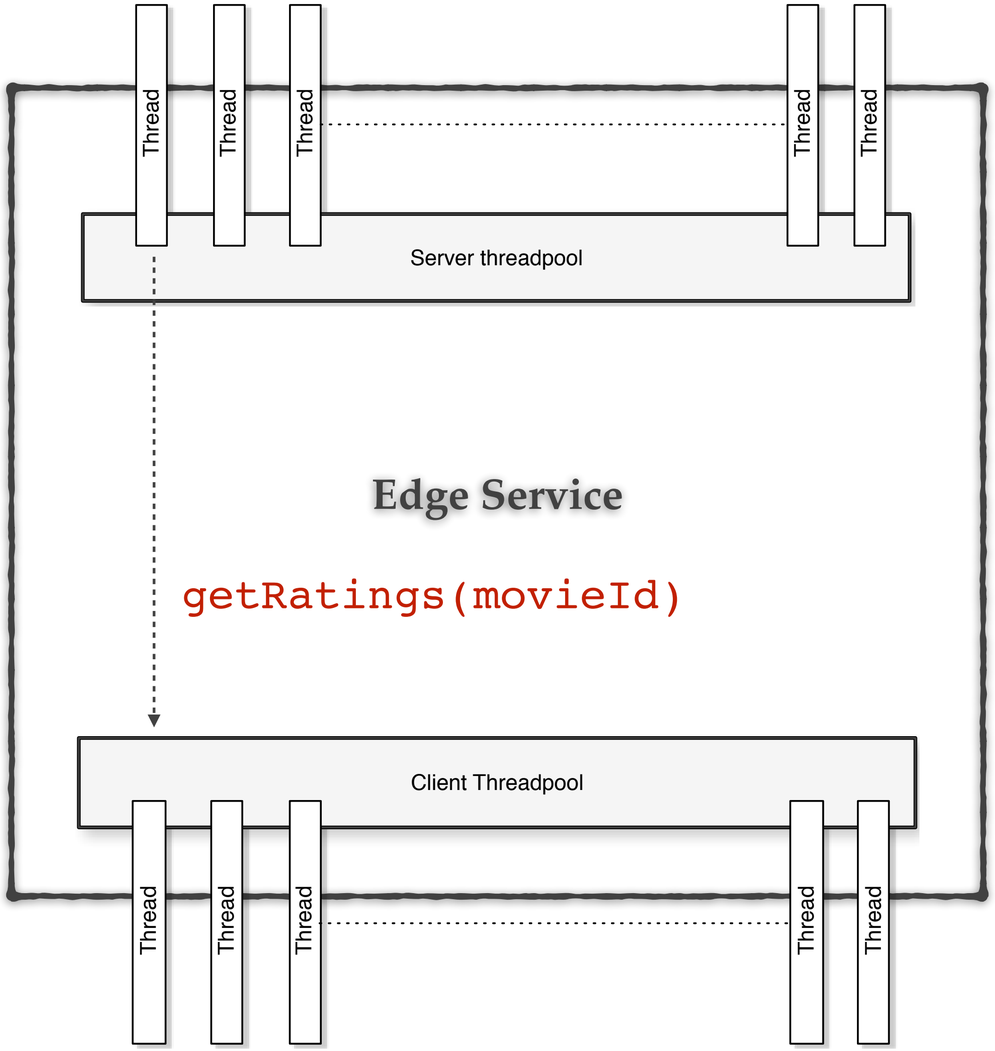
Disclaimer: This is an example and not an exact representation of the processing



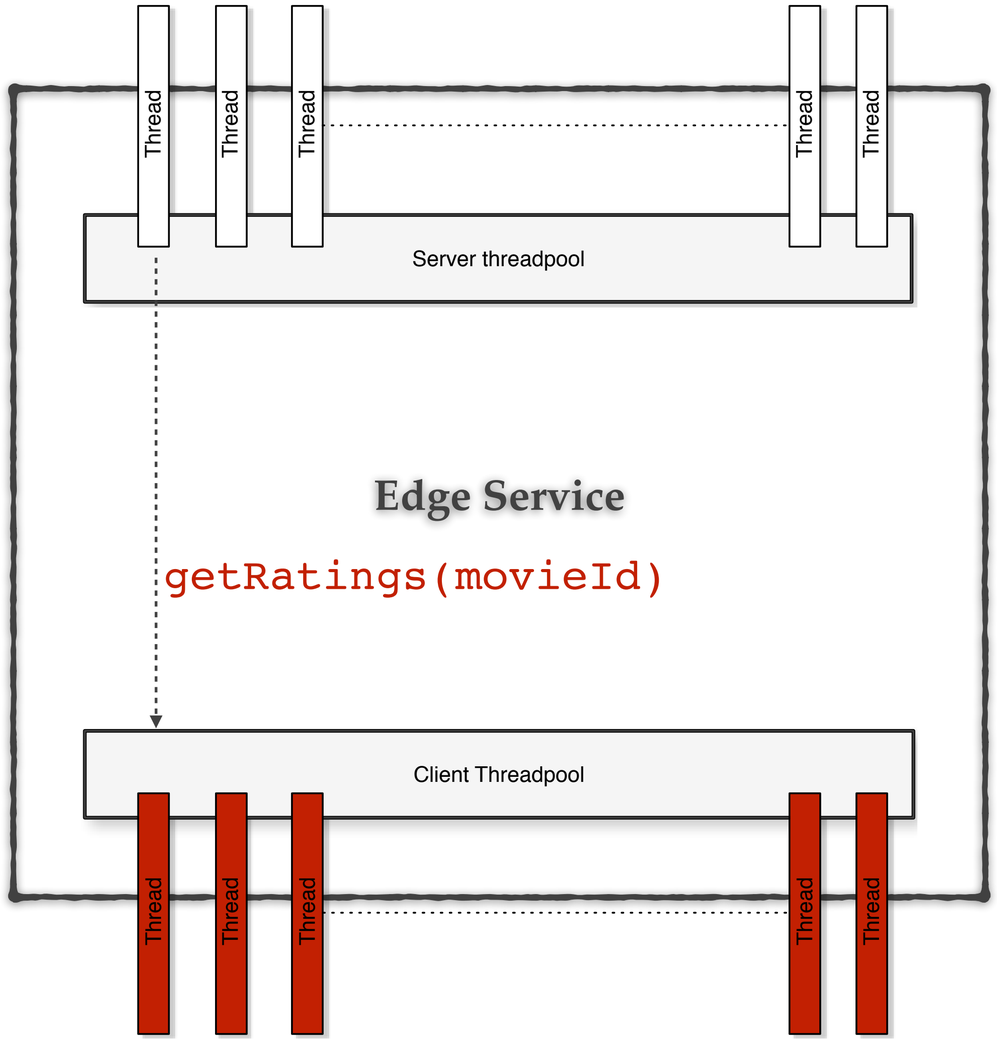
Disclaimer: This is an example and not an exact representation of the processing



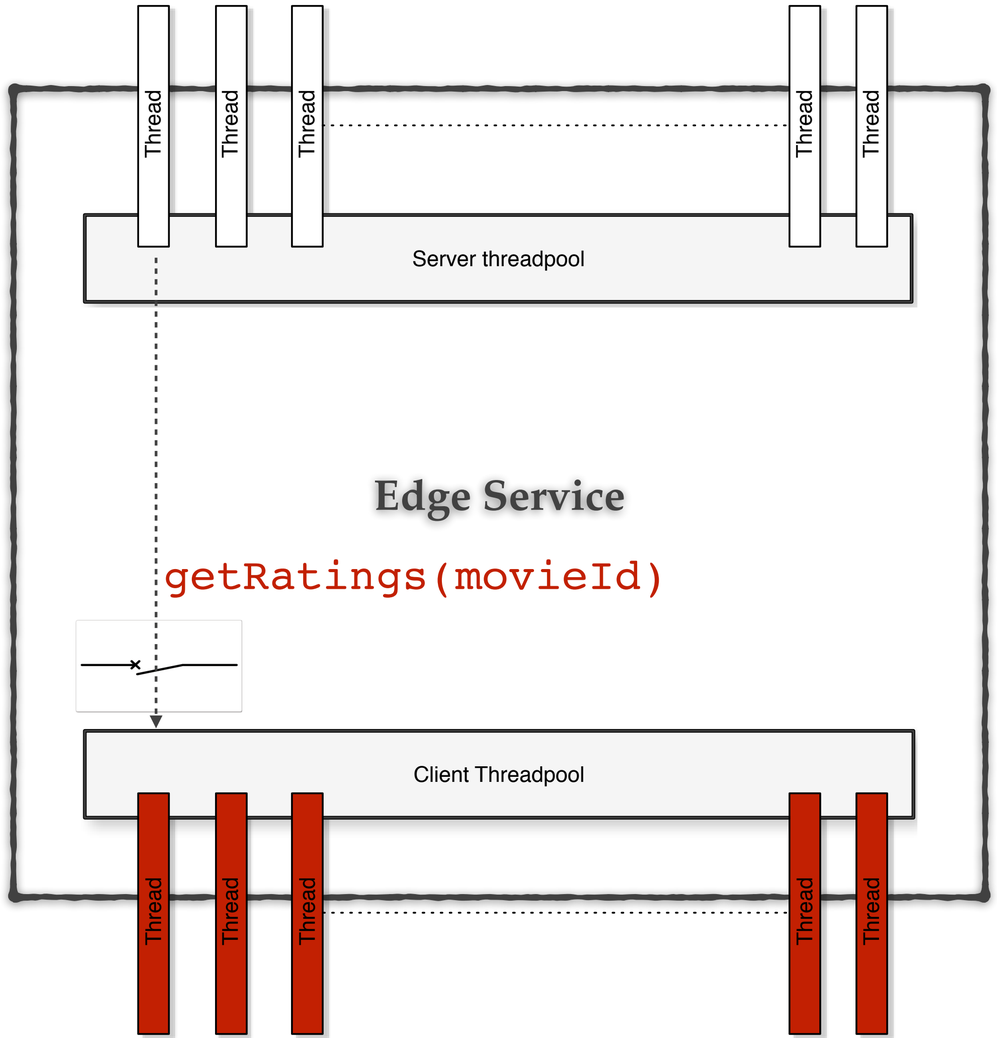
Disclaimer: This is an example and not an exact representation of the processing



Disclaimer: This is an example and not an exact representation of the processing

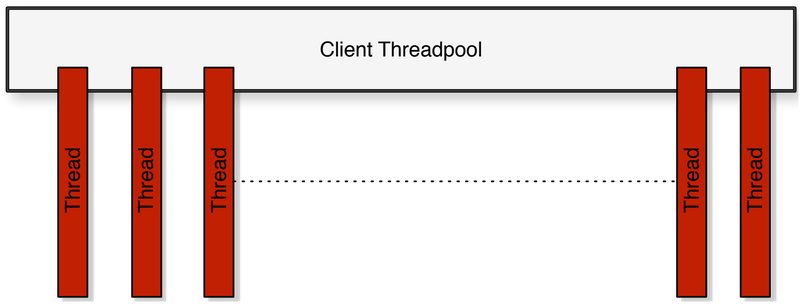


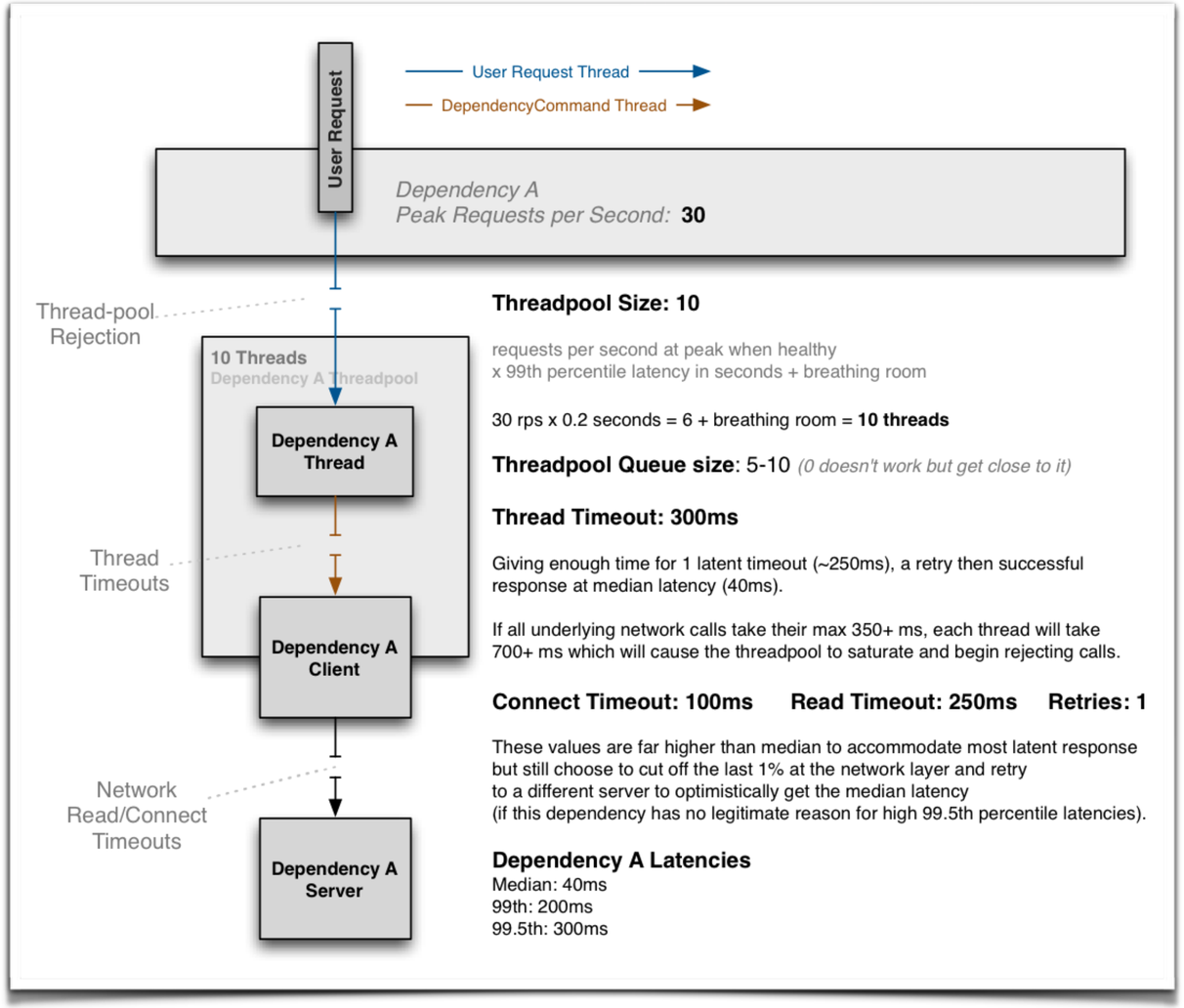
Disclaimer: This is an example and not an exact representation of the processing



Disclaimer: This is an example and not an exact representation of the processing

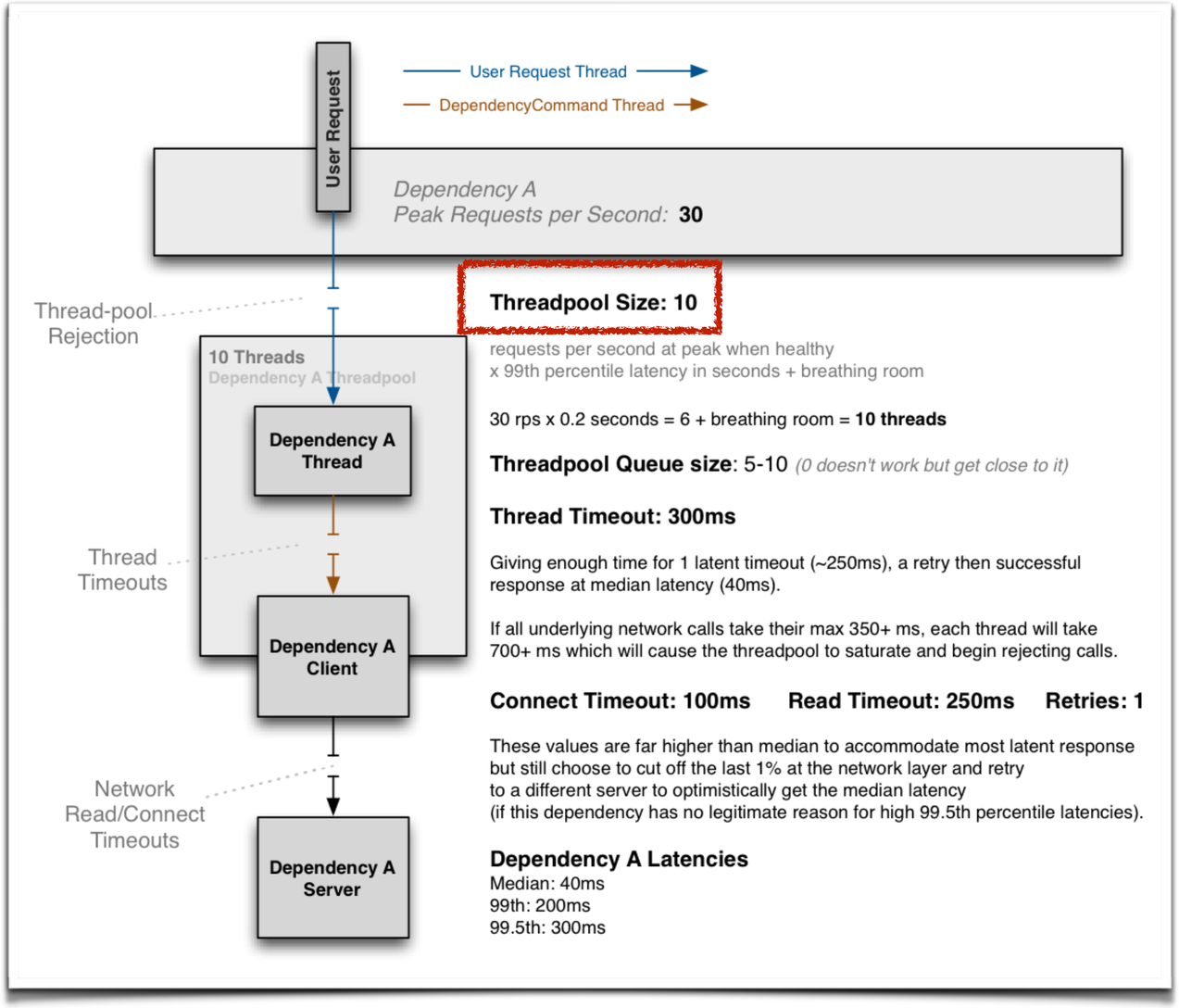
Managing client thread pools



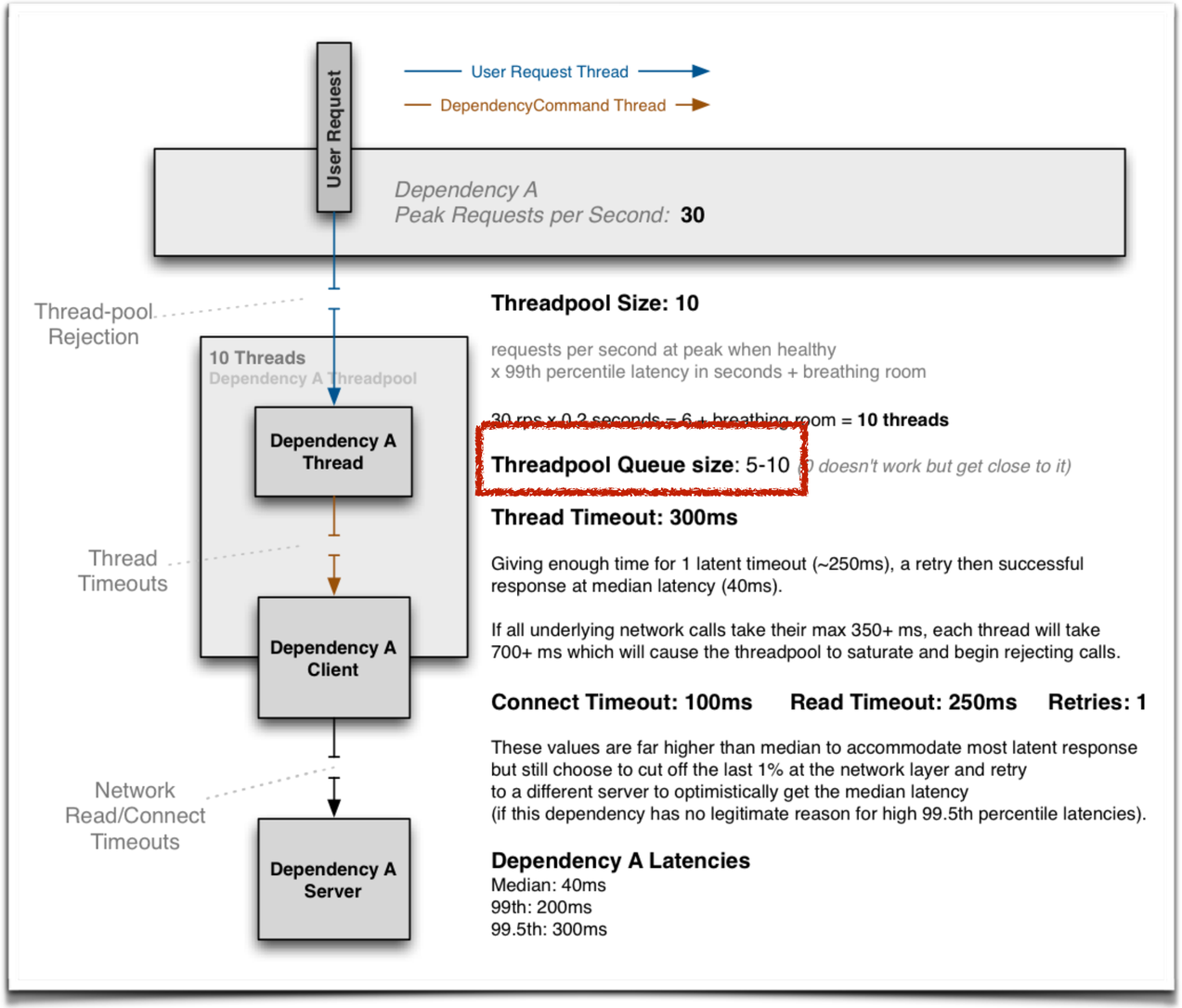


Disclaimer: This is an example and not an exact representation of the processing

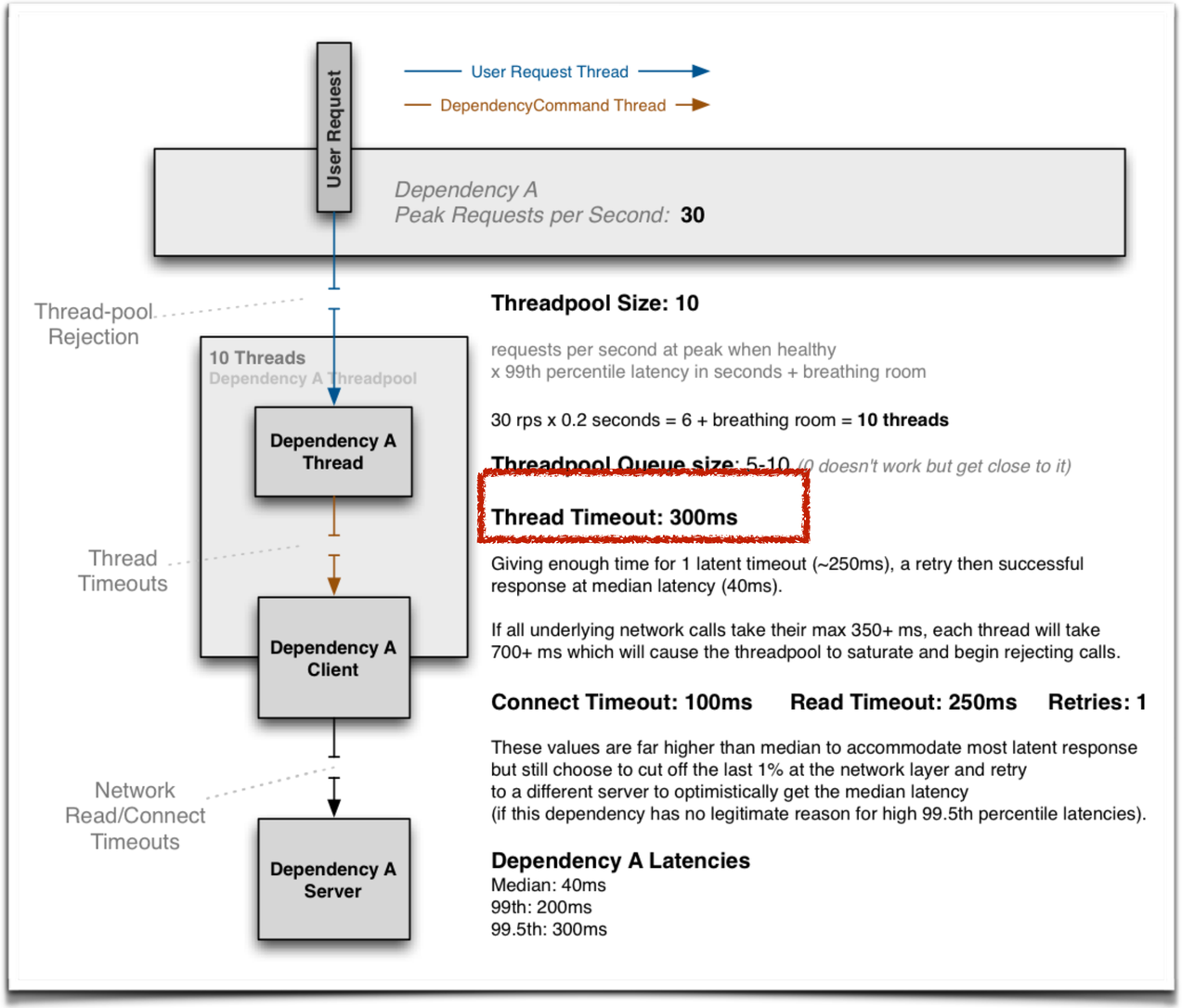
Managing client thread pools



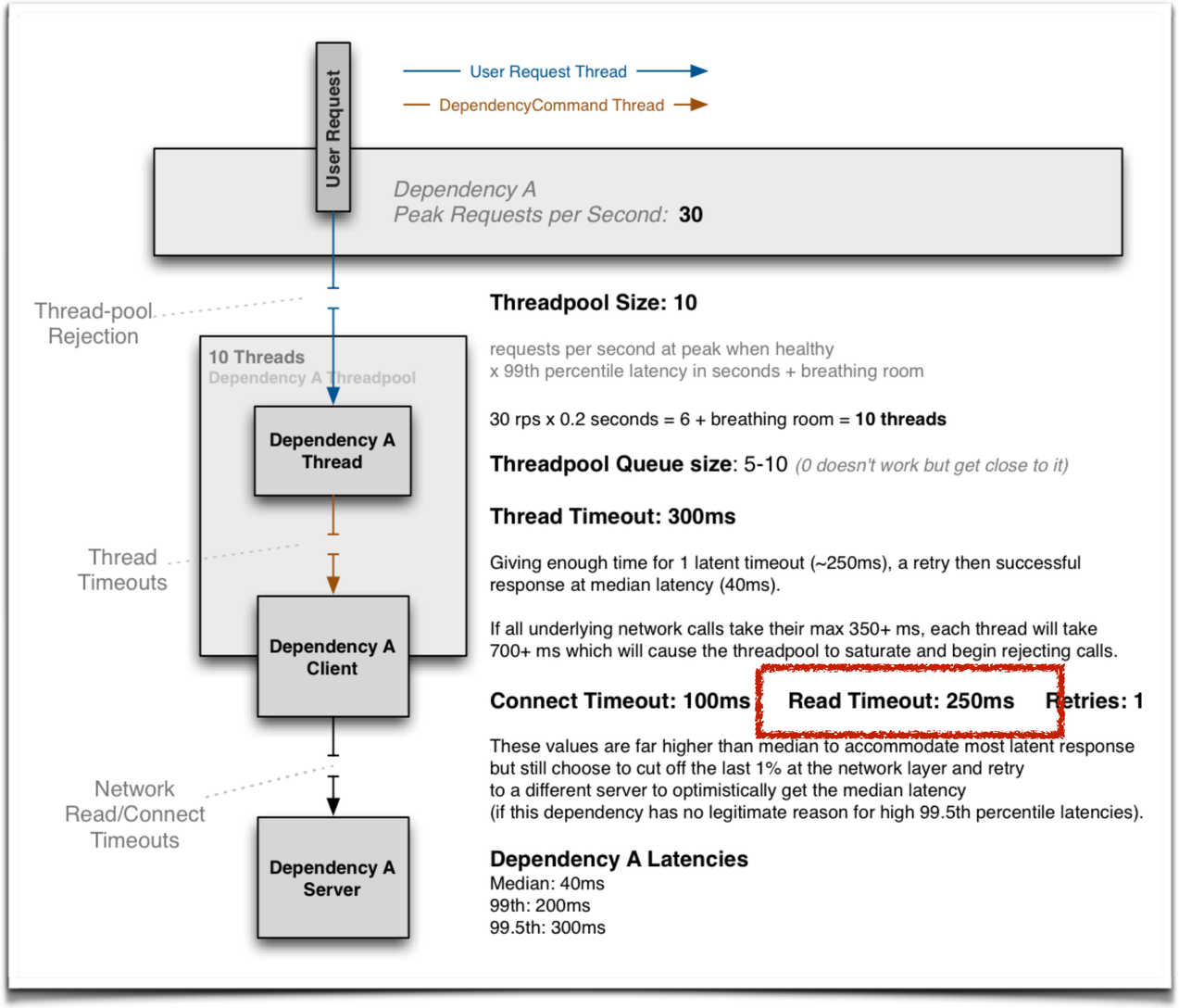
Managing client thread pools



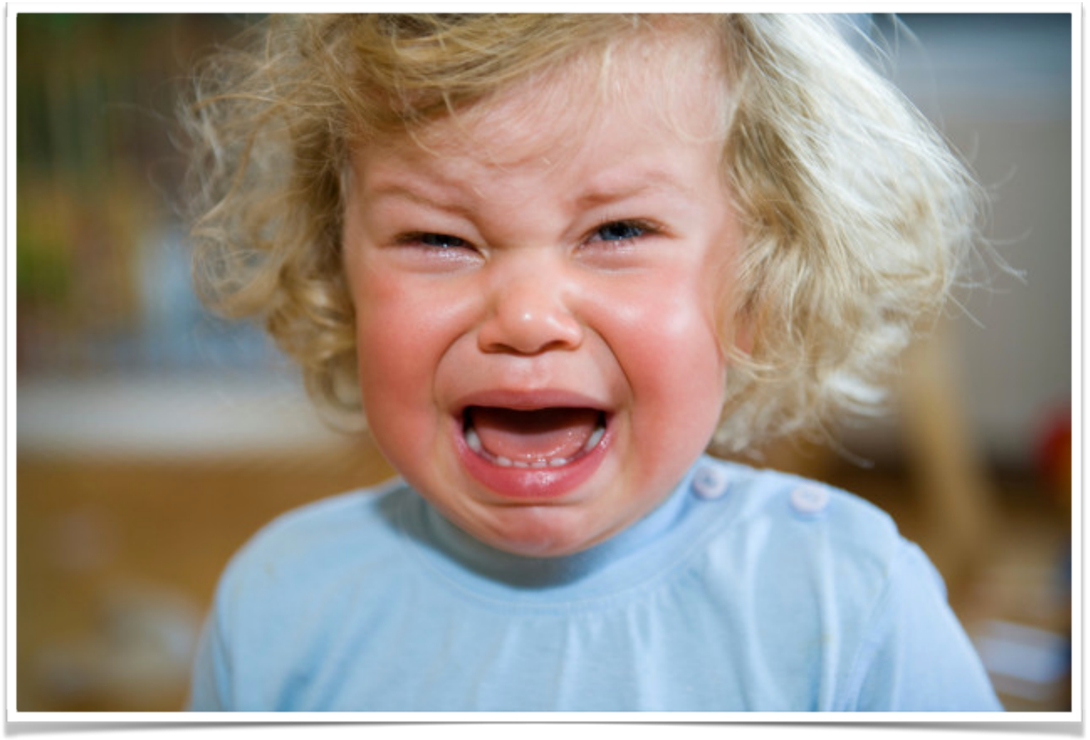
Managing client thread pools



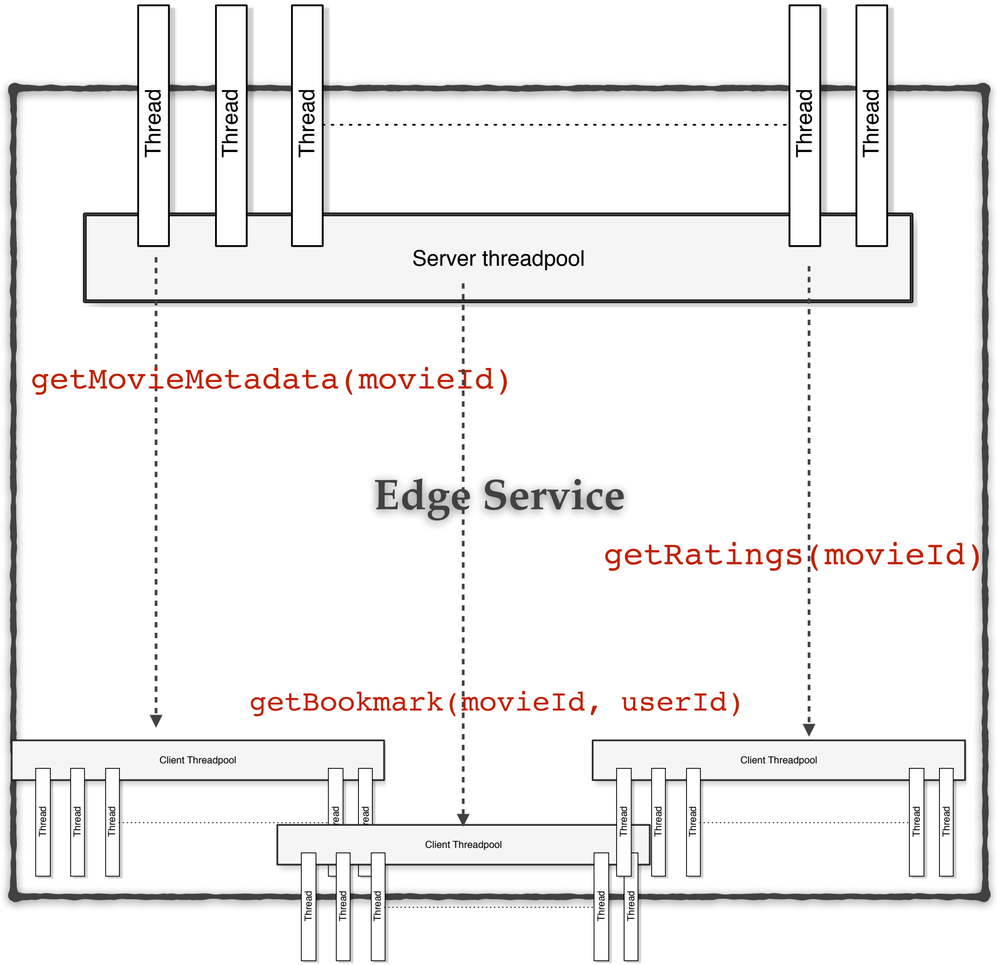
Managing client thread pools



Clients have become our babies



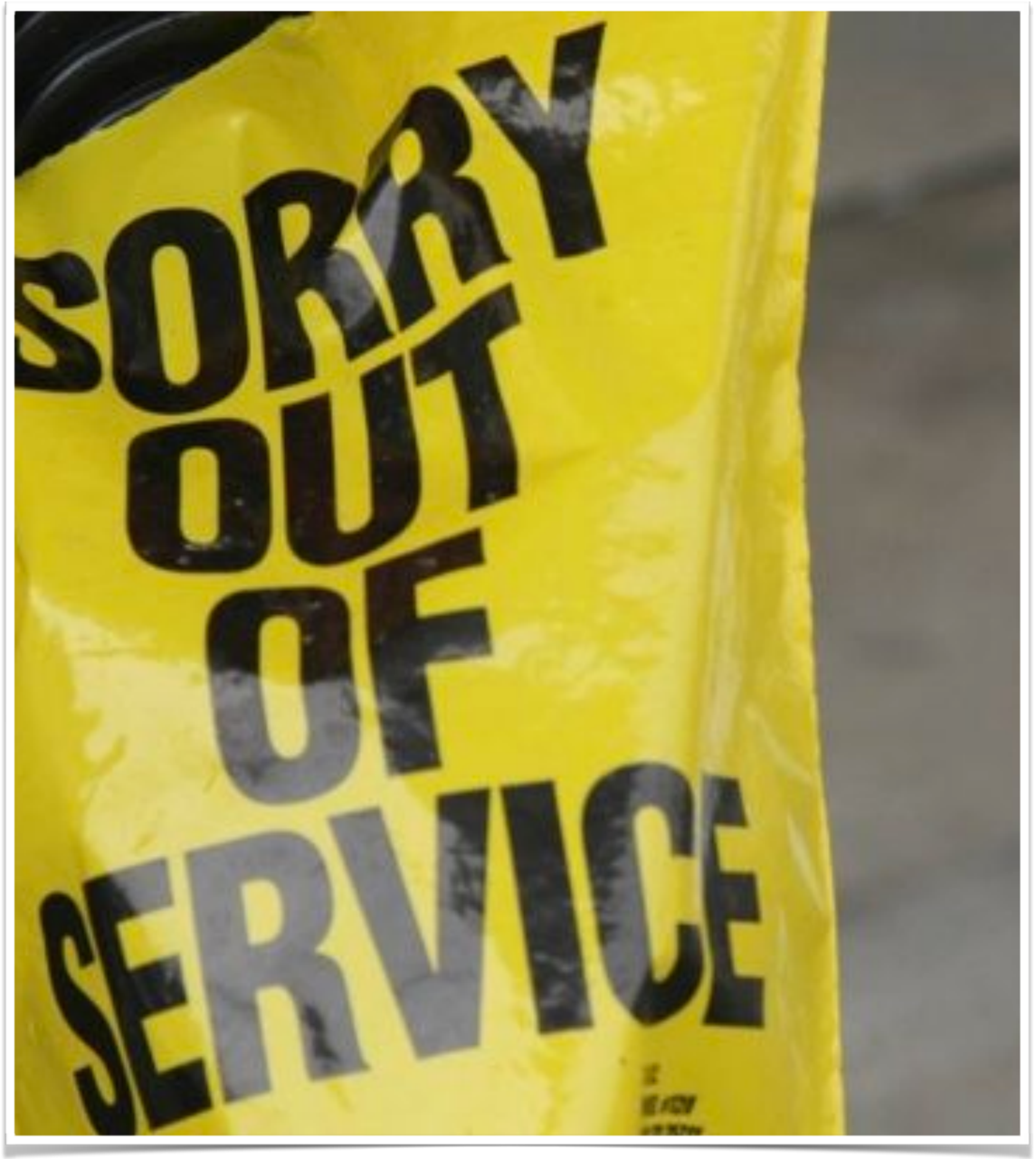
Clients have become our babies



Disclaimer: This is an example and not an exact representation of the processing

Clients have become our babies

Untuned/Wrongly tuned clients cause many outages.

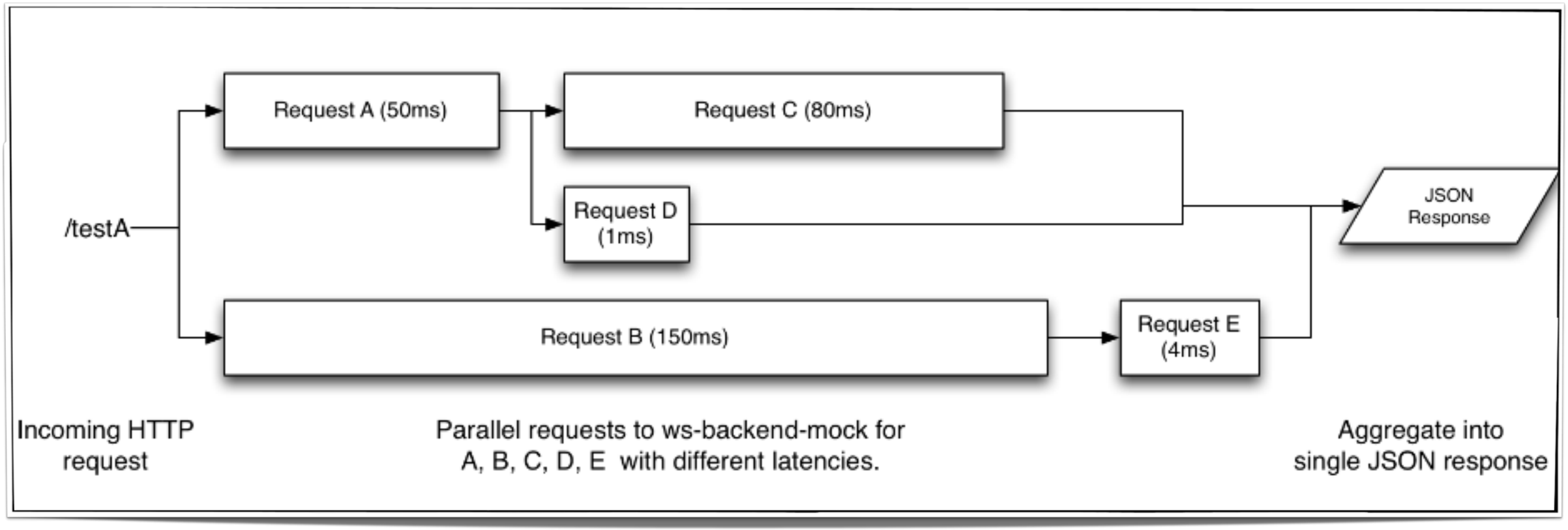


Have we exchanged a bigger problem with a smaller one?

How do systems fail?

2. Overload

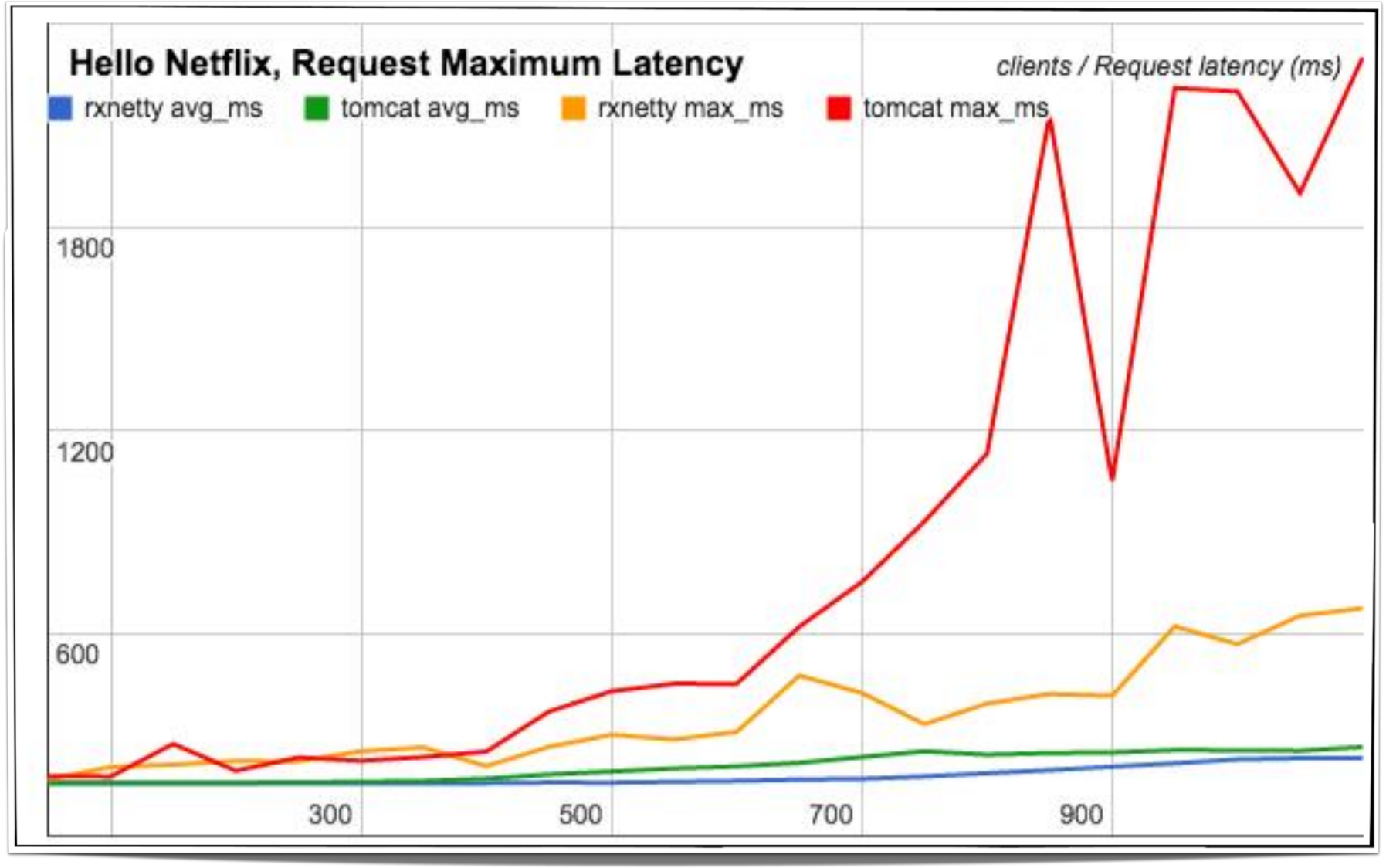
Abusive clients, recovery spikes, special events ….



Hello Netflix!

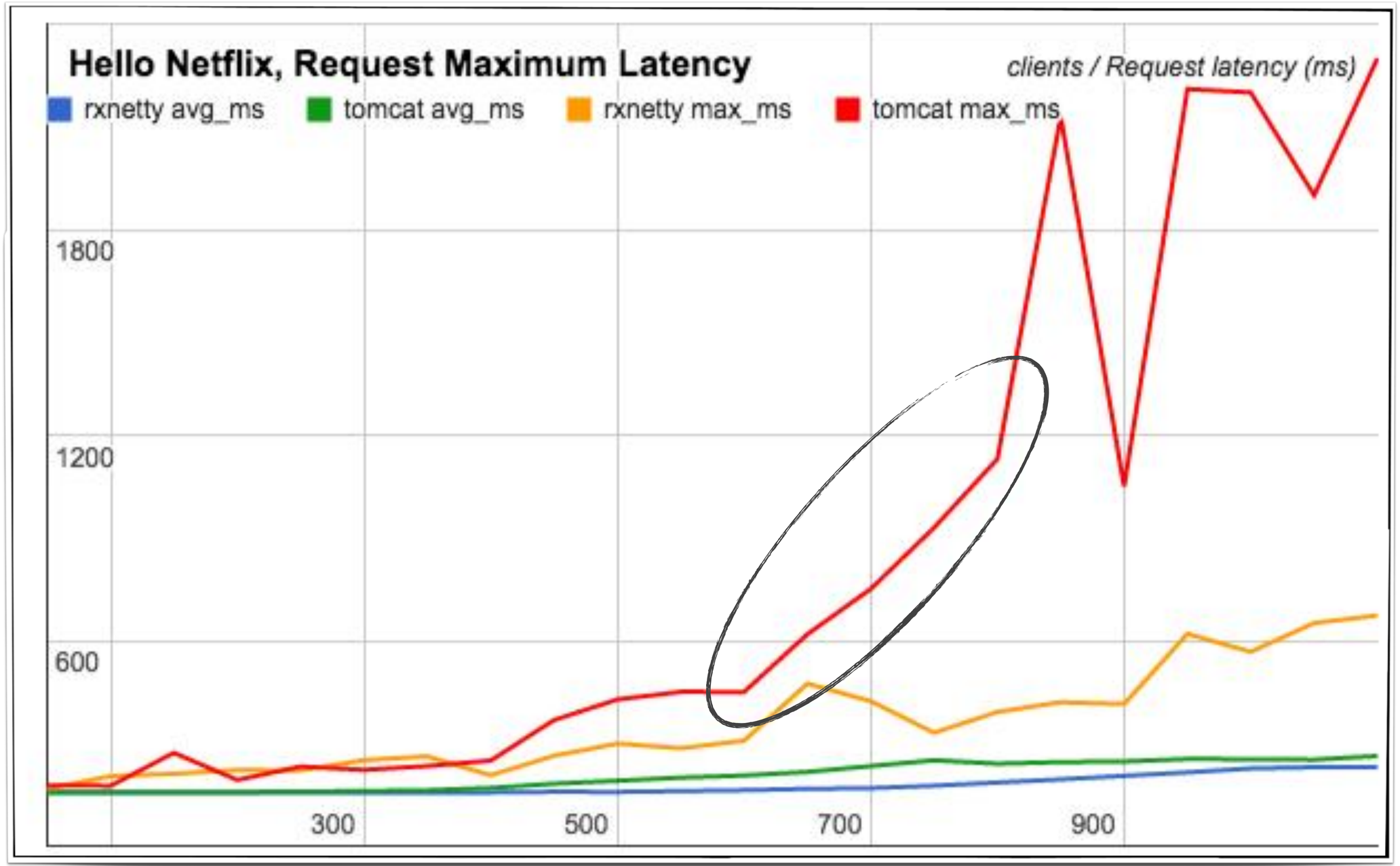
We did a load test…

https://github.com/Netflix-Skunkworks/WSPerfLab



Detailed analysis available online:

https://github.com/Netflix-Skunkworks/WSPerfLab/blob/master/test-results/RxNetty_vs_Tomcat_April2015.pdf


Hello Netflix, Request Maximum Latency
clients / Request latency (ms)
rxnetty avg_ms
tomcat avg_ms
rxnetty max_ms
tomcat max_ms
1800
1200
600
300
500
700
900

! Graceful

This isn’t graceful degradation!

This happens at high CPU usage.

This happens at high CPU usage.

So, don’t let the system reach that limit…

This happens at high CPU usage.

So, don’t let the system reach that limit…

a.k.a Throttling.

Fairness?

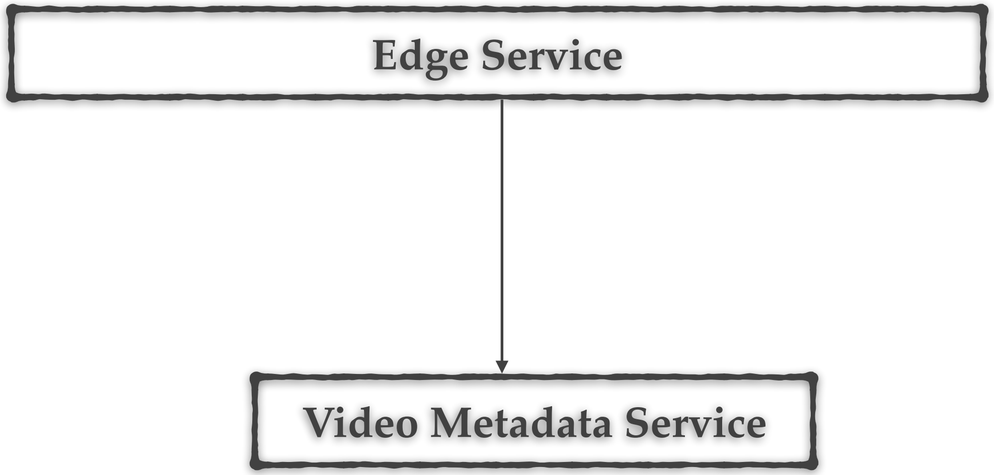
One abusive request type can penalize other request paths.

How do systems fail?

3. Thundering herds

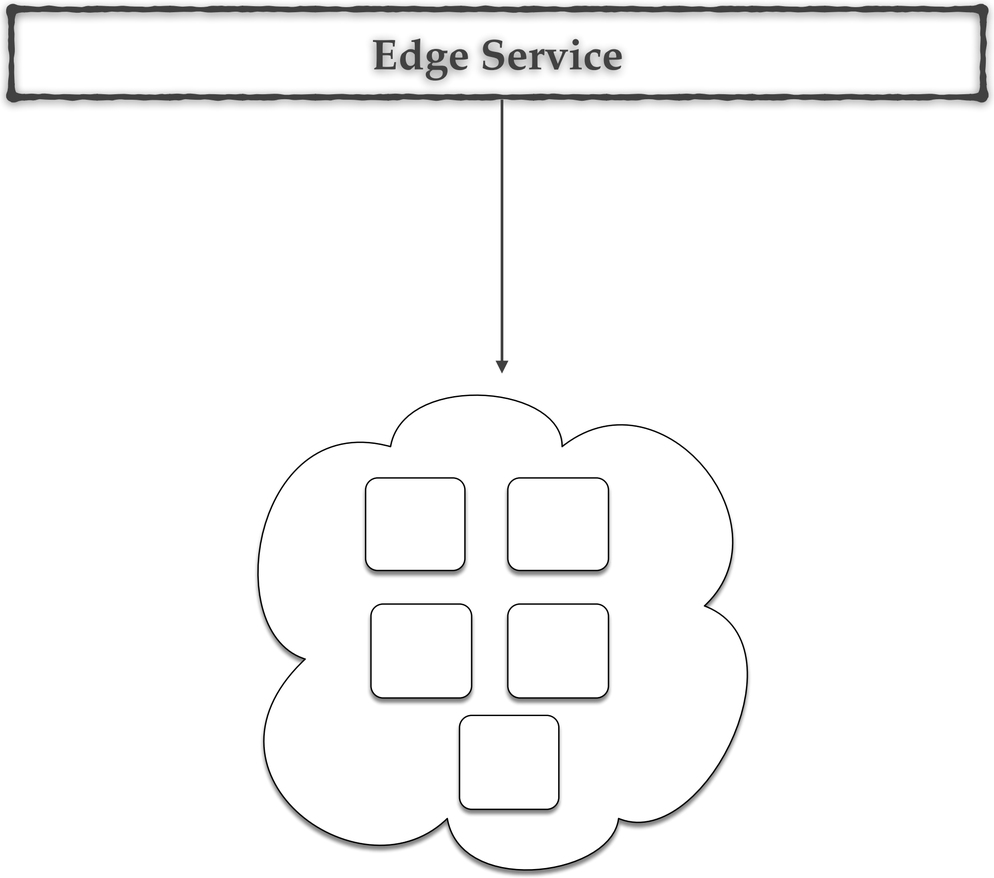
The failure after recovery….

Retries



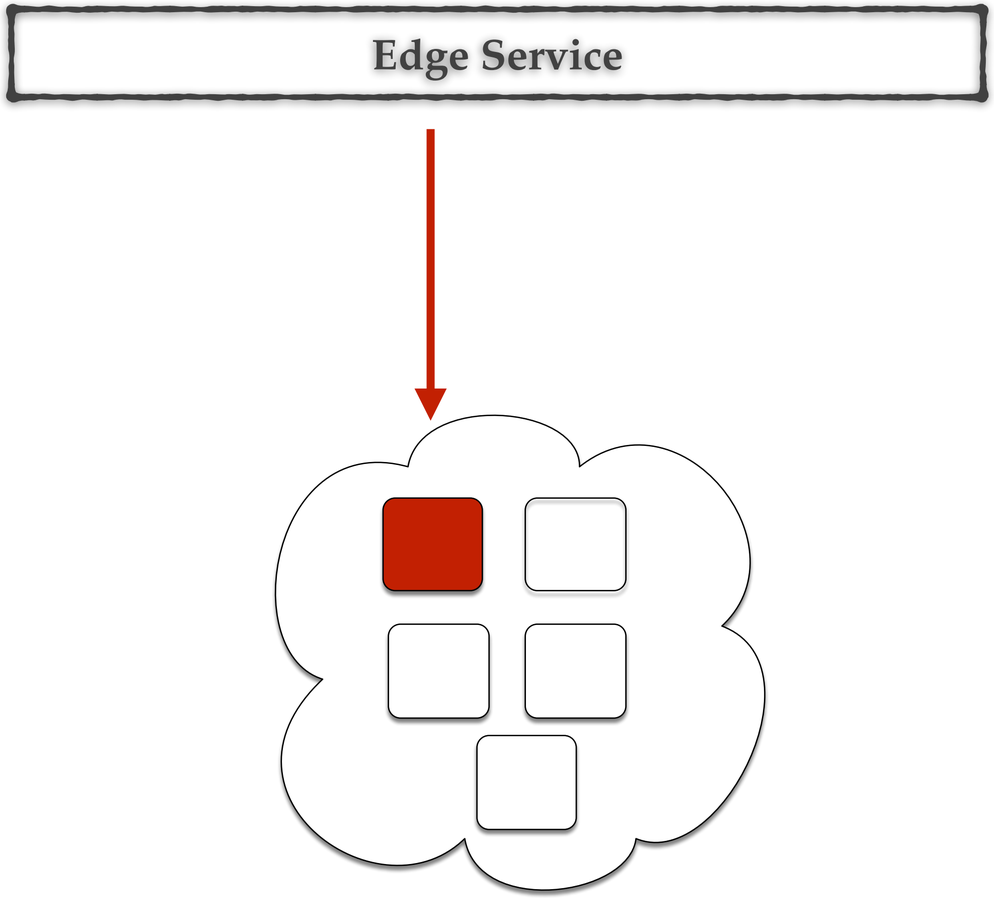
Disclaimer: This is an example and not an exact representation of the processing

Retries



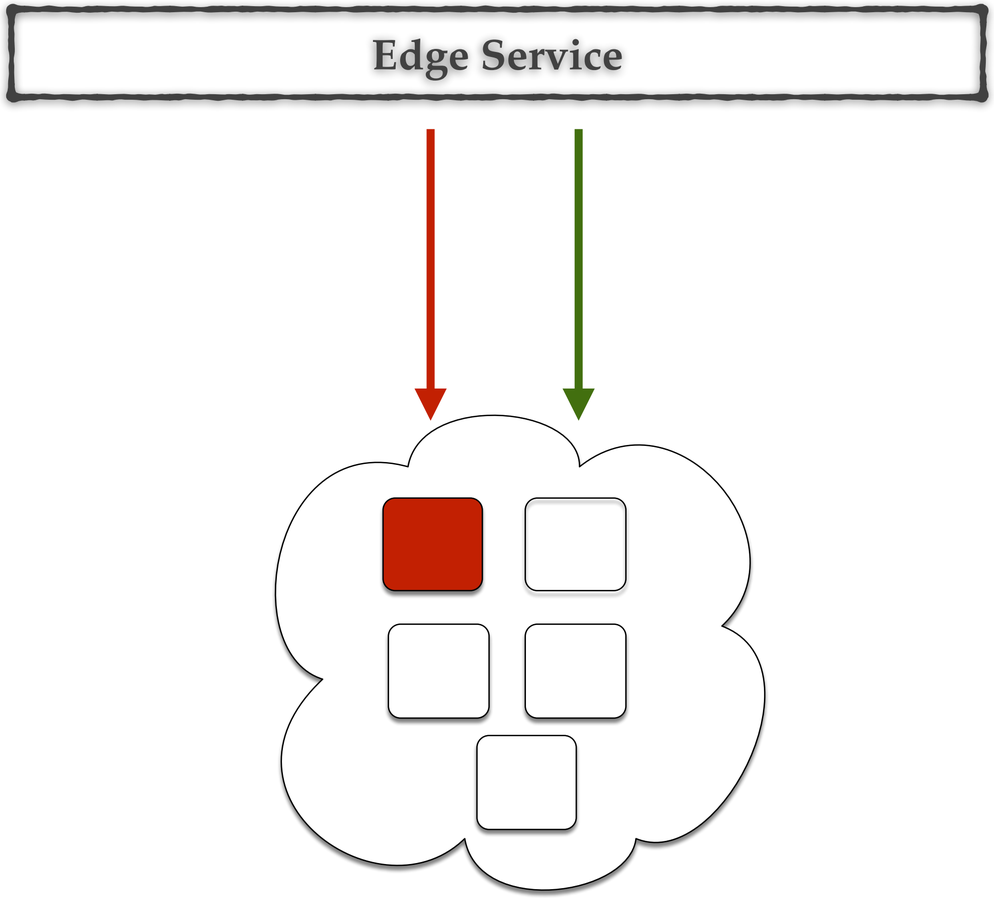
Disclaimer: This is an example and not an exact representation of the processing

Retries



Disclaimer: This is an example and not an exact representation of the processing

Retries

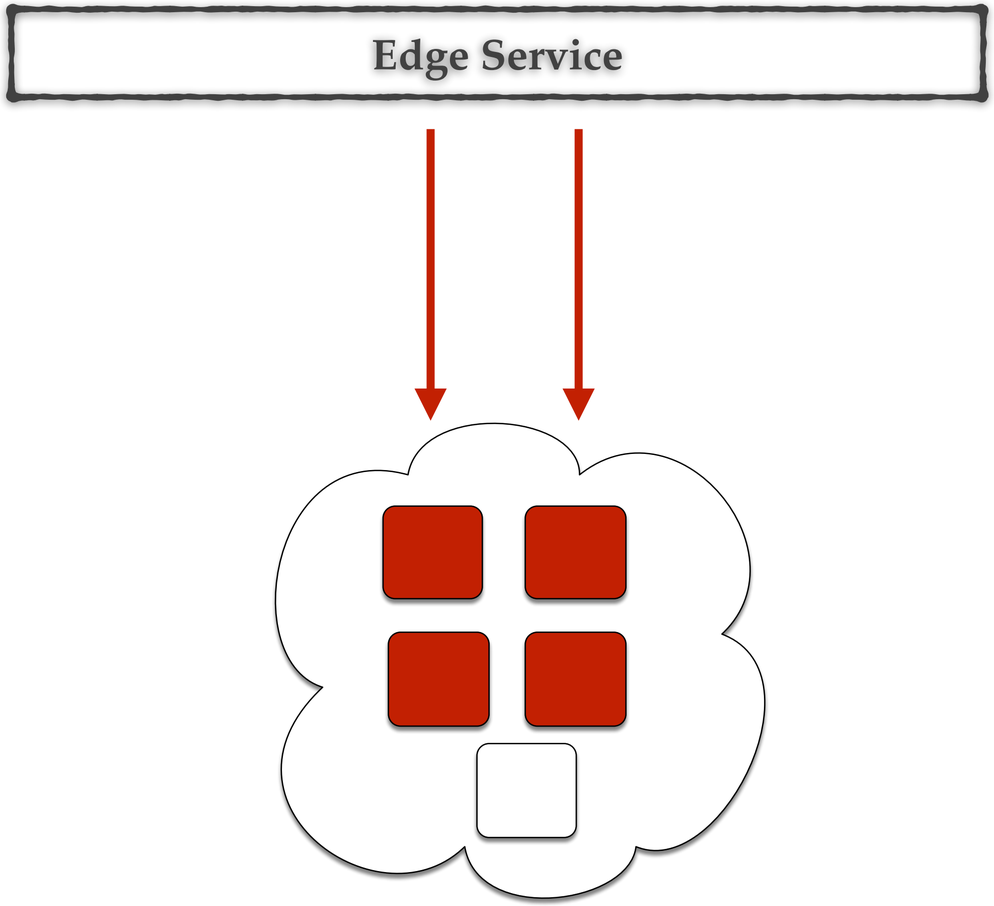


Disclaimer: This is an example and not an exact representation of the processing

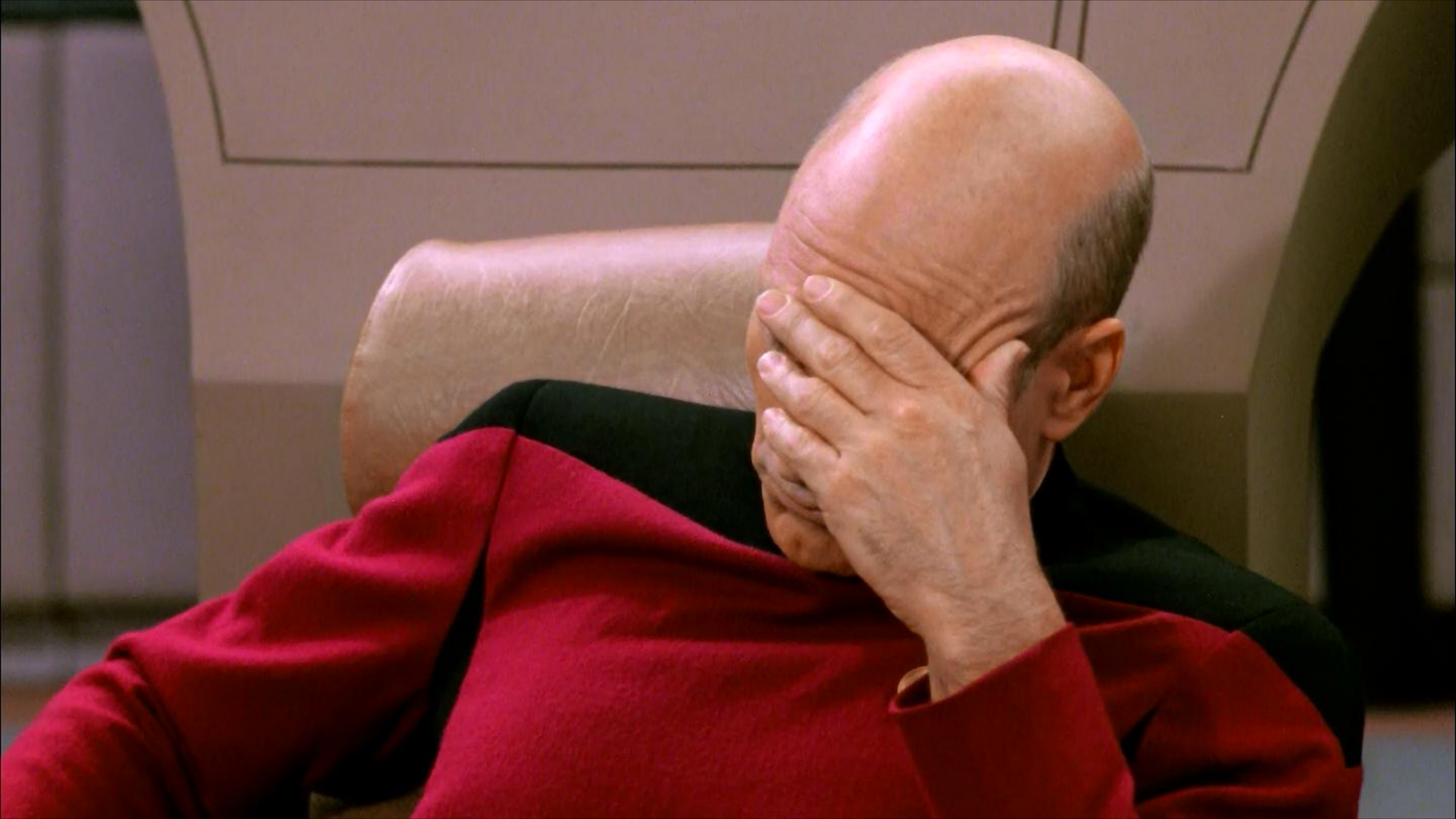
Retries are useful in steady state....

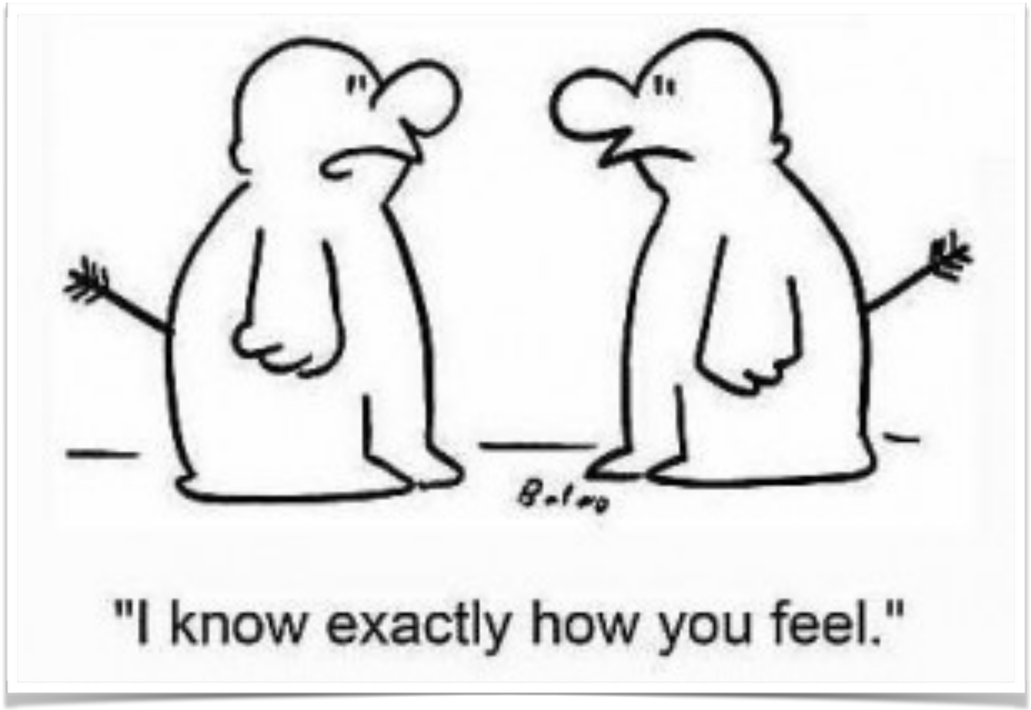
...but...

Retries



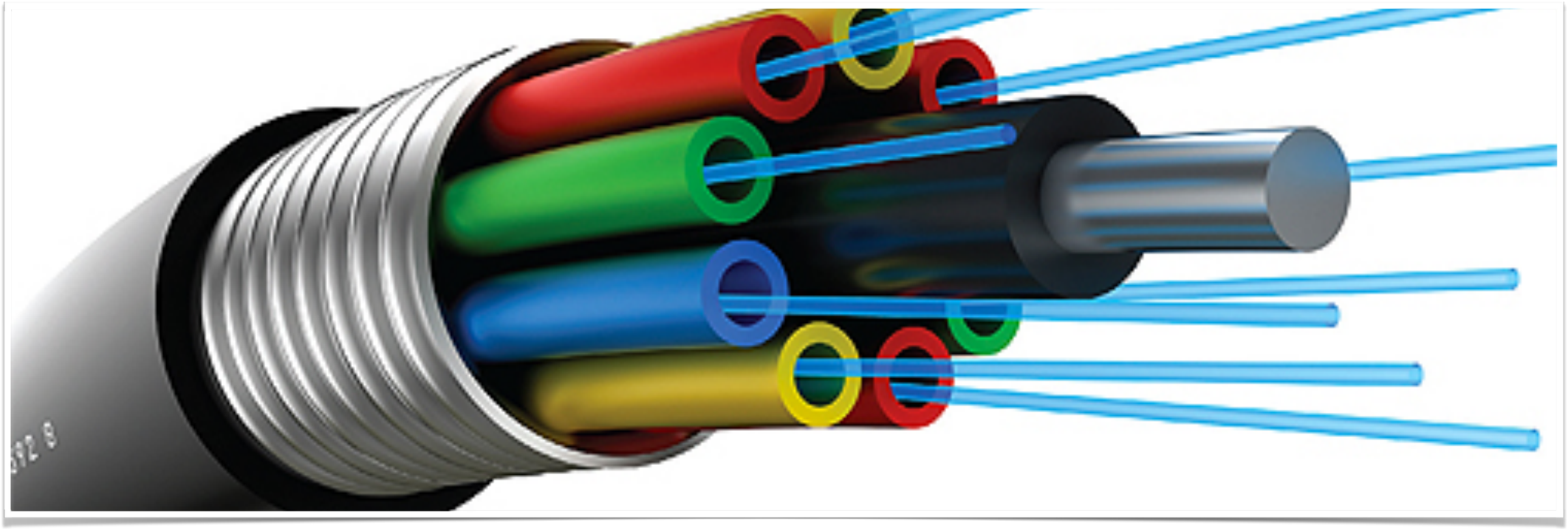
Disclaimer: This is an example and not an exact representation of the processing





Our systems are missing empathy.

Because they lack knowledge about the peers.



Knowledge comes from various signals..

Ability to adapt to those signals is important.

This can not adapt...

```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

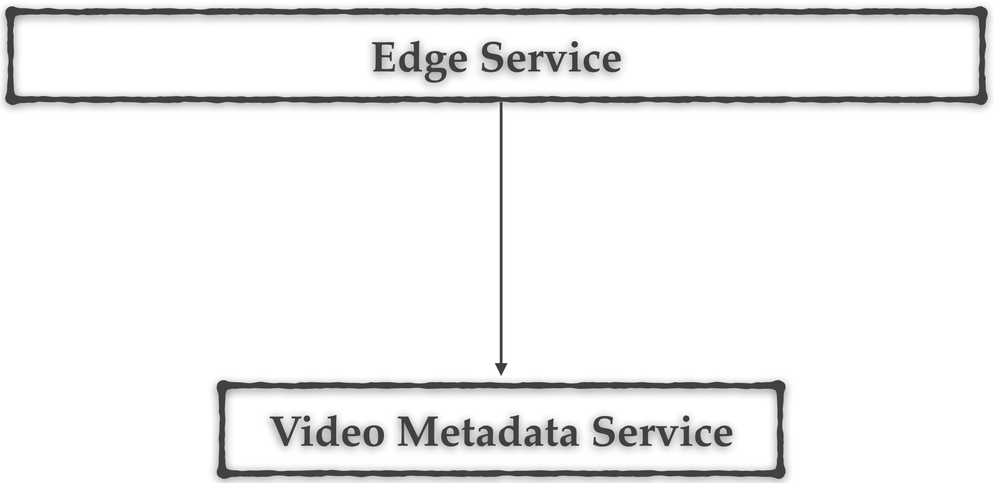
Disclaimer: This is an example and not an exact representation of the processing

Asynchrony

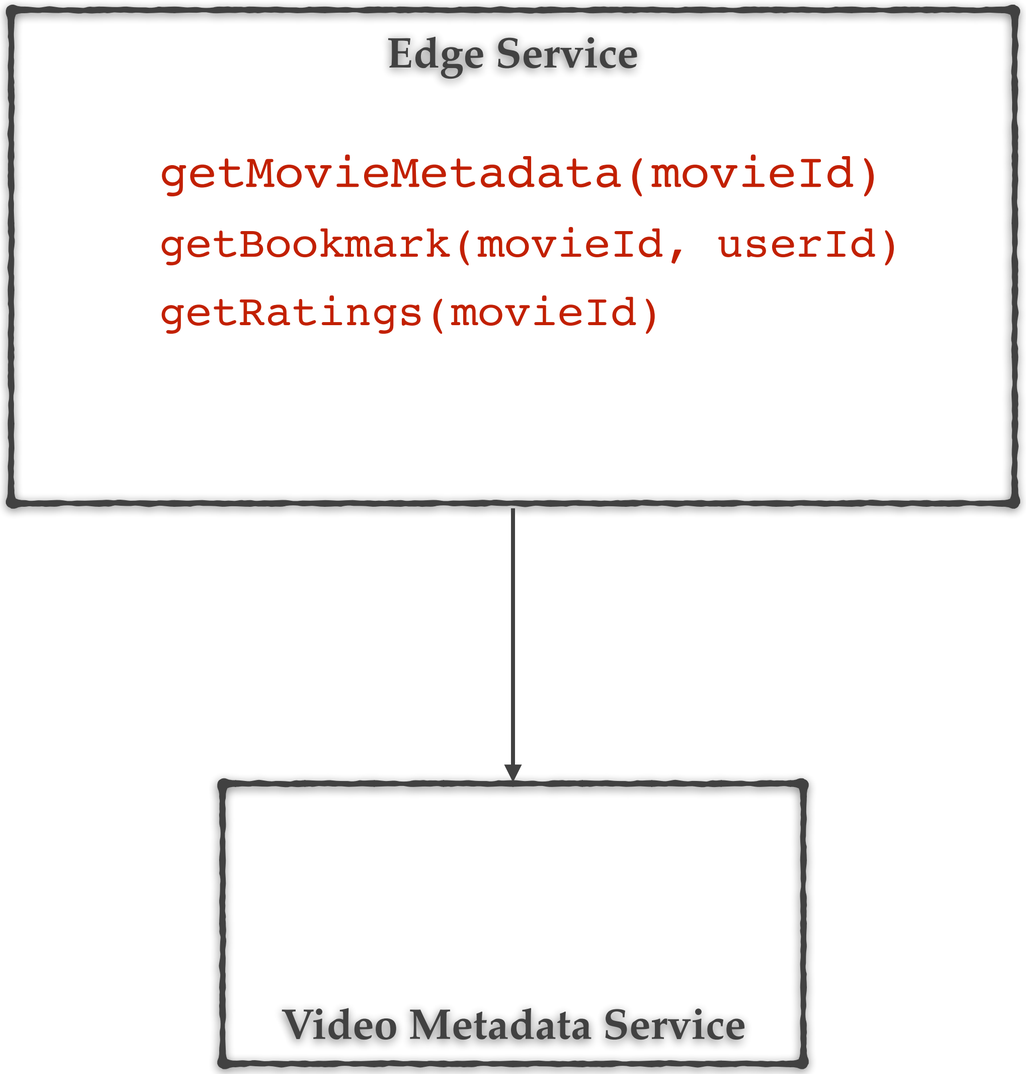
It is the key to success.

What should be async?

What should be async?

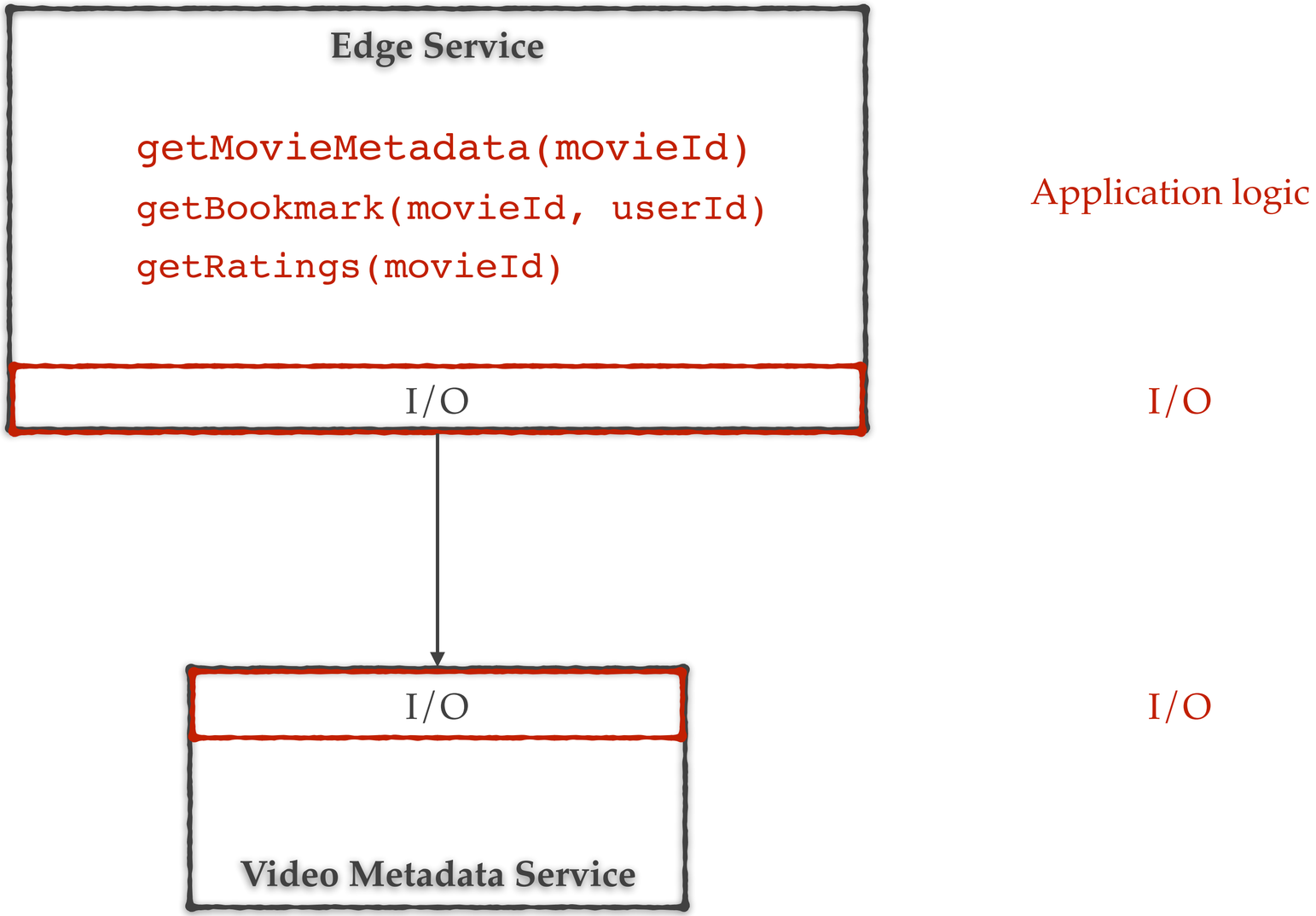


What should be async?

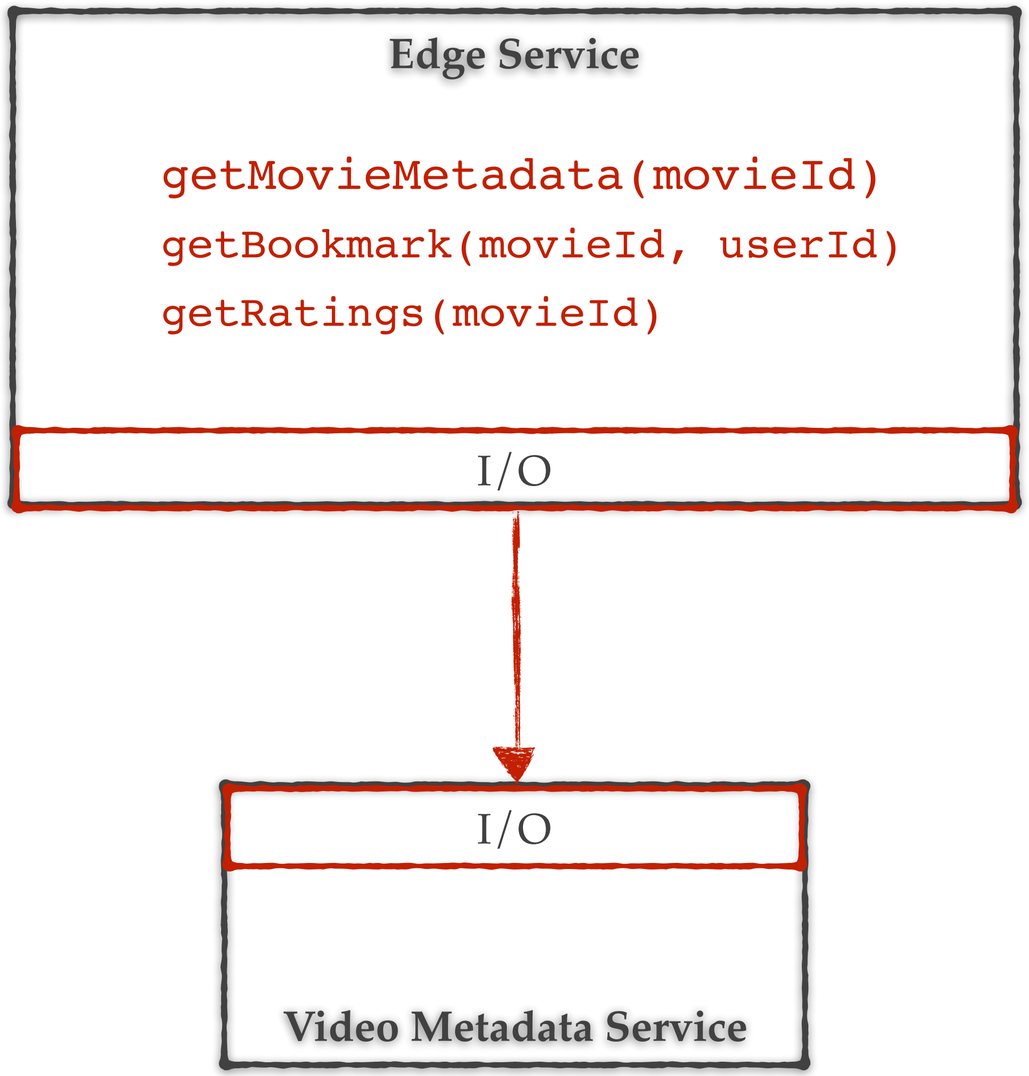


Application logic

What should be async?



What should be async?



Application logic

I/O

Network protocol

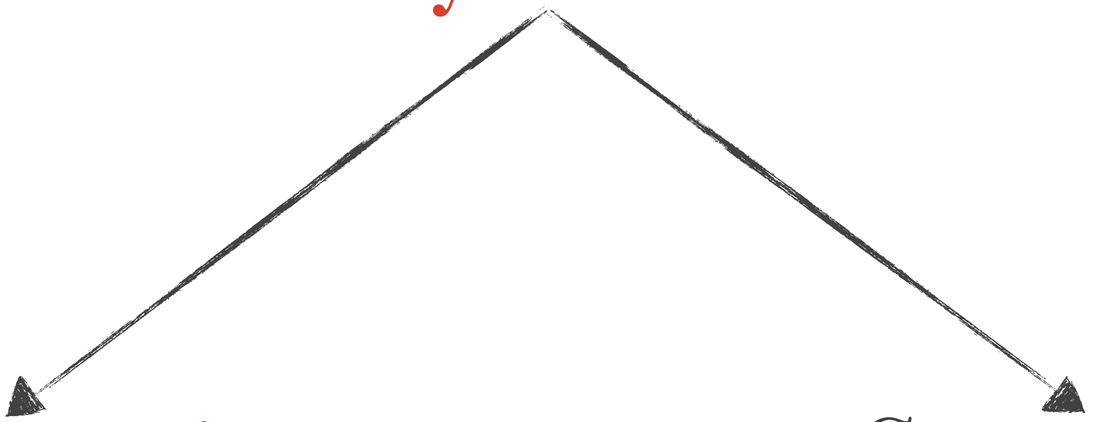
I/O

Key aspects of being async.

Key aspects of being async.

1. Lifecycle control

Lifecycle control

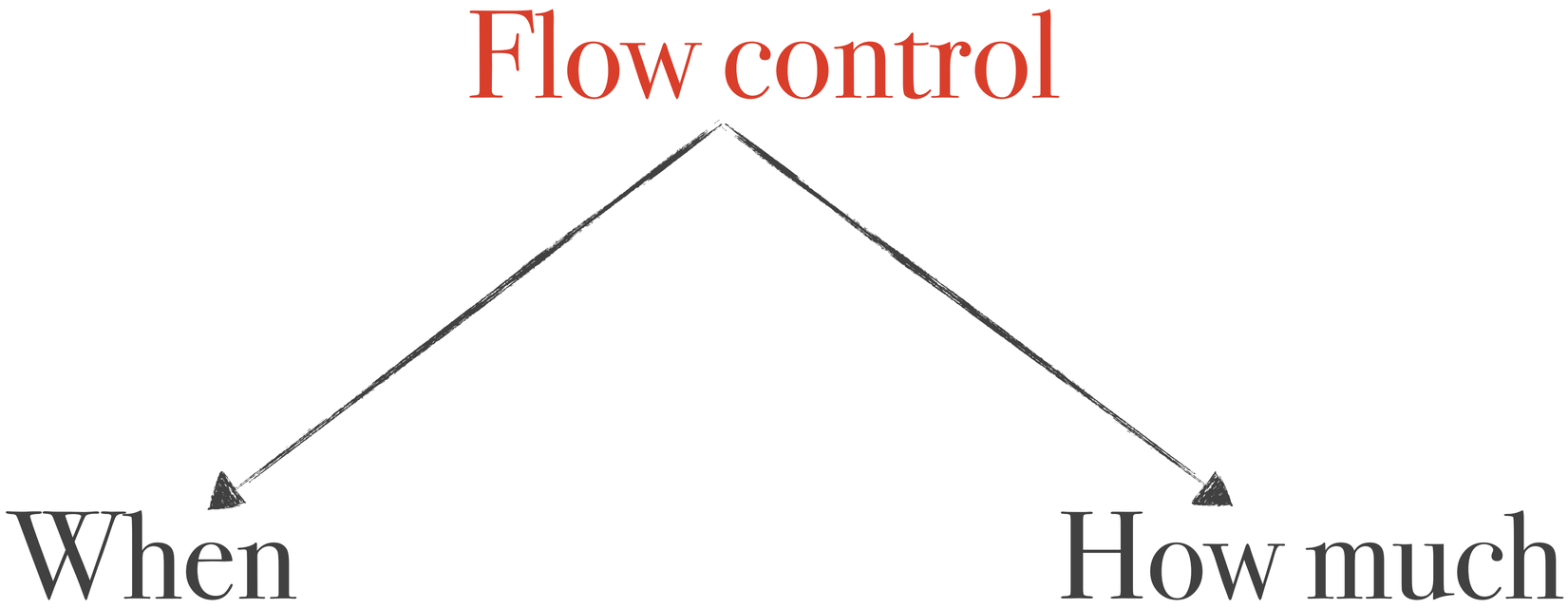


Start processing

Stop processing

Key aspects of being async.

2. Flow control


Flow control
When
How much

Key aspects of being async.

3. Function composition

Function composition

```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

Function composition

```
public Observable<Movie> getMovie(String movieId) {
        return Observable.zip(getMovieMetadata(movieId),
                              getBookmark(movieId, userId),
                              getRatings(movieId),
                              (meta,bmark,rating)->new Movie(meta,bmark,rating));
}
```

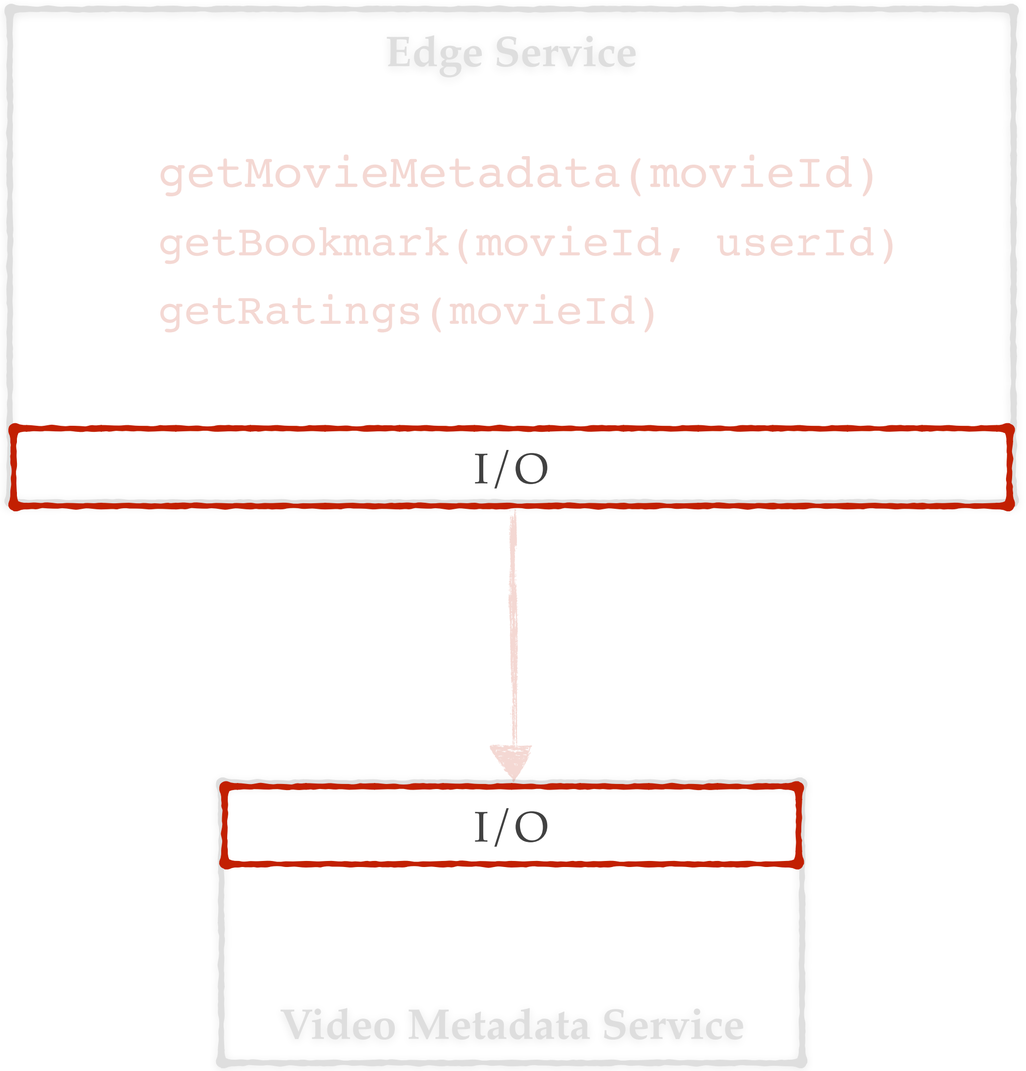
Composing the processing
of a method into a single control point.

Composing the processing
of **a method** into a single control point.

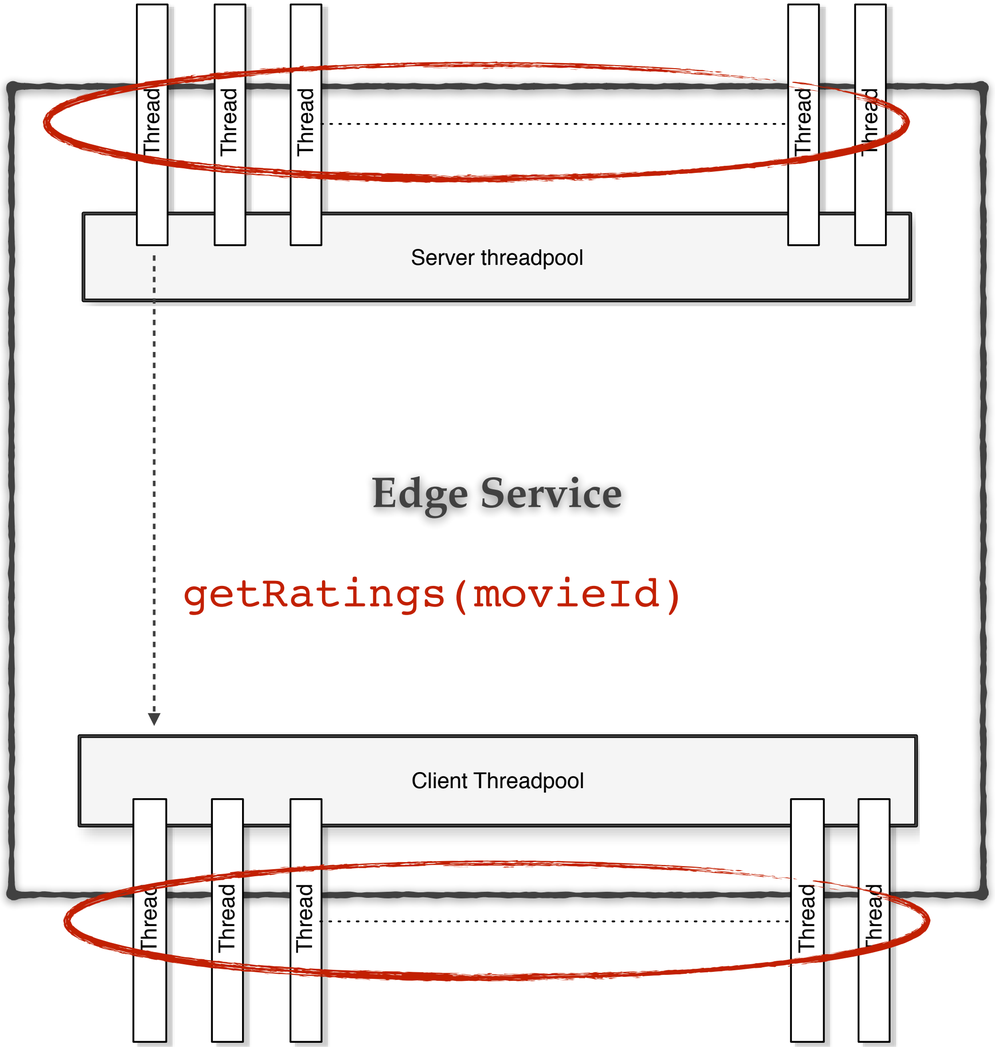
with

Flow & Lifecycle Control

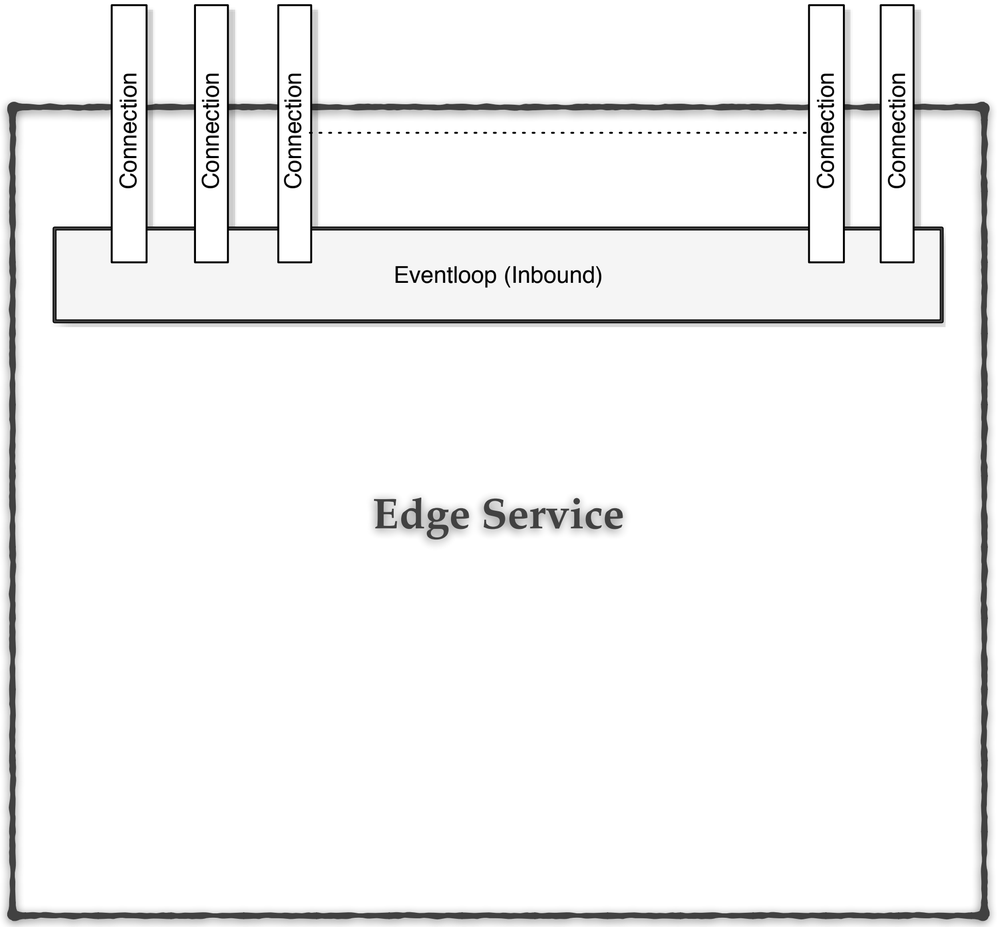
What should be async?



I/O


Thread
Thread
Thread
Thread
Thread
Server threadpool
Edge Service
getRatings(movieId)
Client Threadpool
Thread
Thread
Thread
Thread
Thread

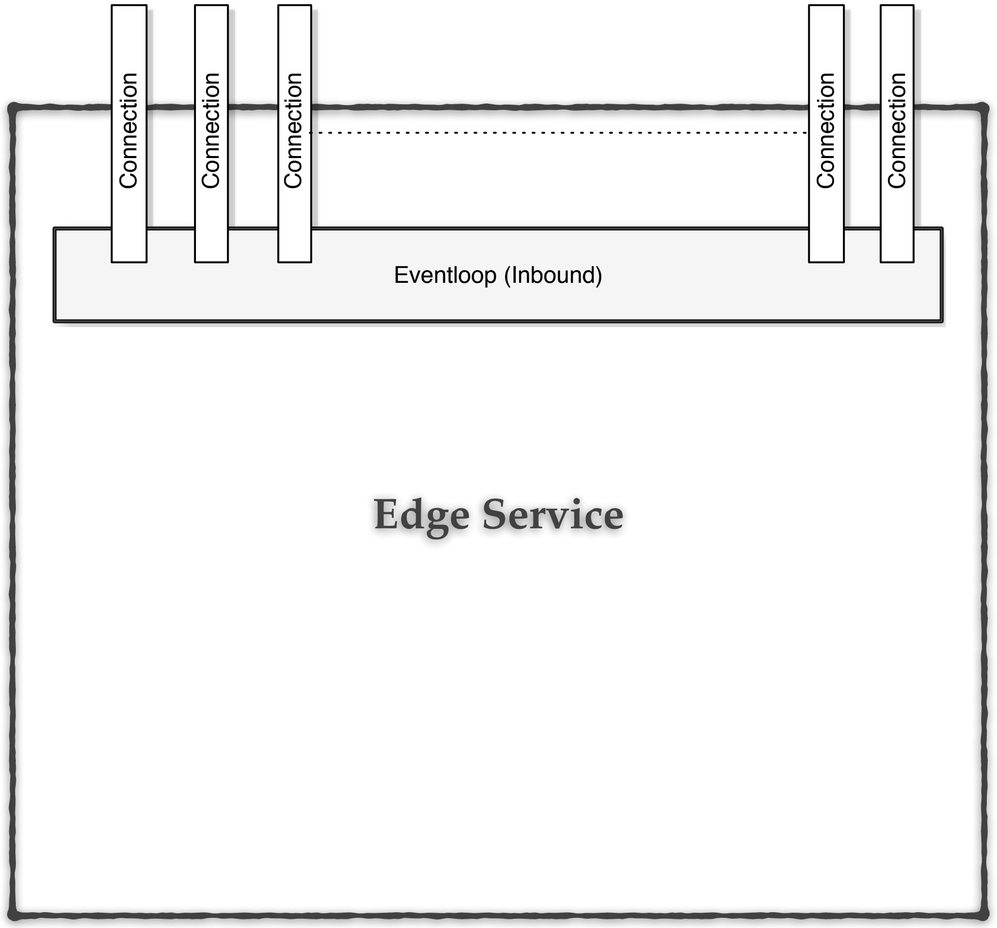
I/O



Eventloops = f (Number of cores)

Disclaimer: This is an example and not an exact representation of the processing

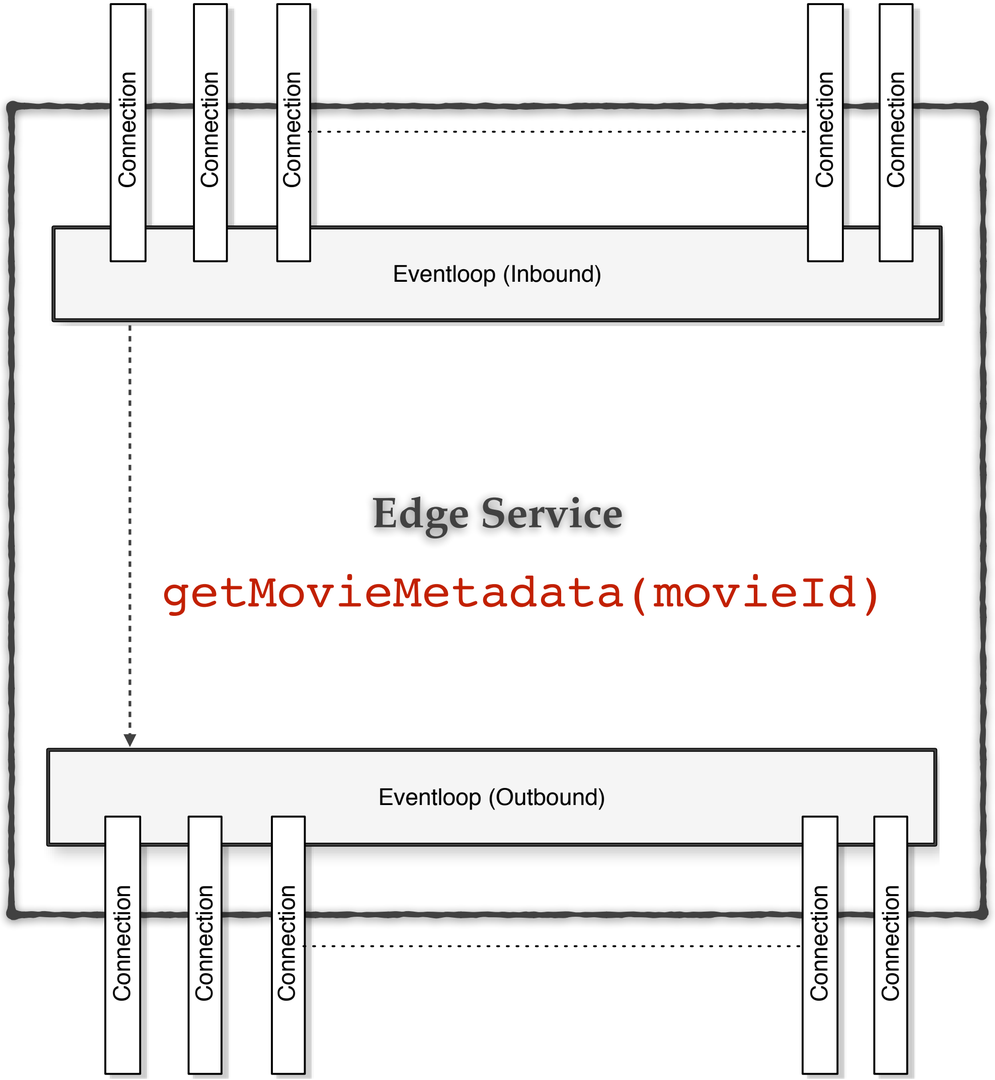
I/O



Connections multiplexed
on a
single eventloop.

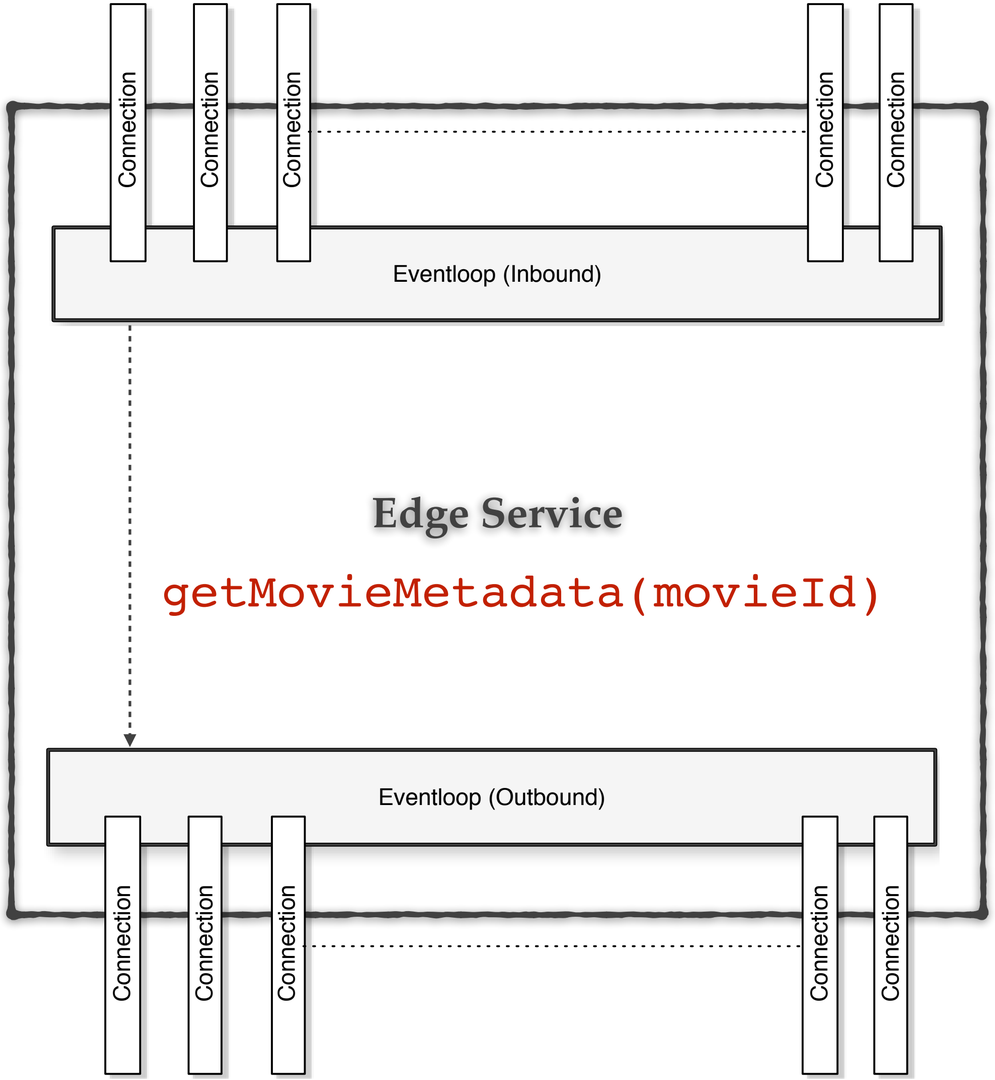
Disclaimer: This is an example and not an exact representation of the processing

I/O



Disclaimer: This is an example and not an exact representation of the processing

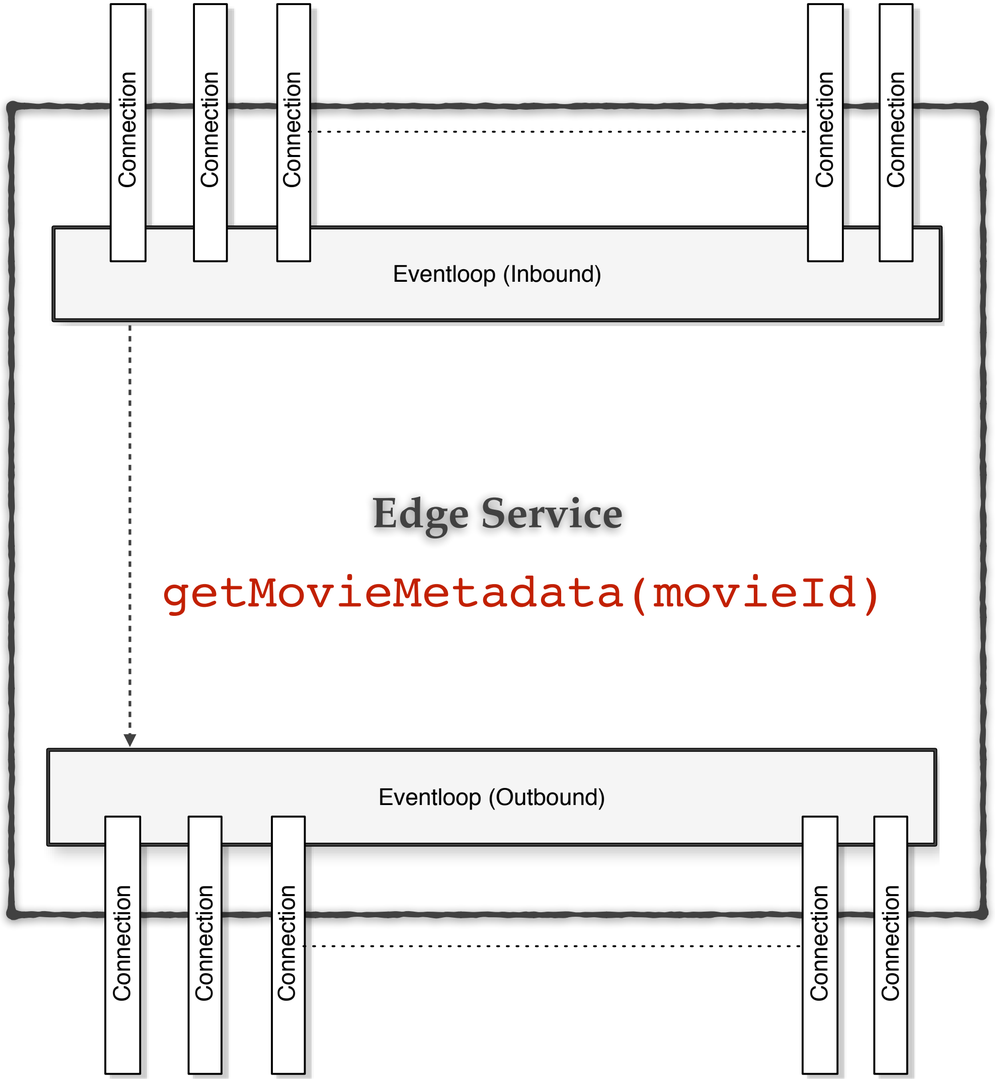
I/O



Clients share the eventloops
with
the server.

Disclaimer: This is an example and not an exact representation of the processing

I/O



All clients share
the
same eventloop

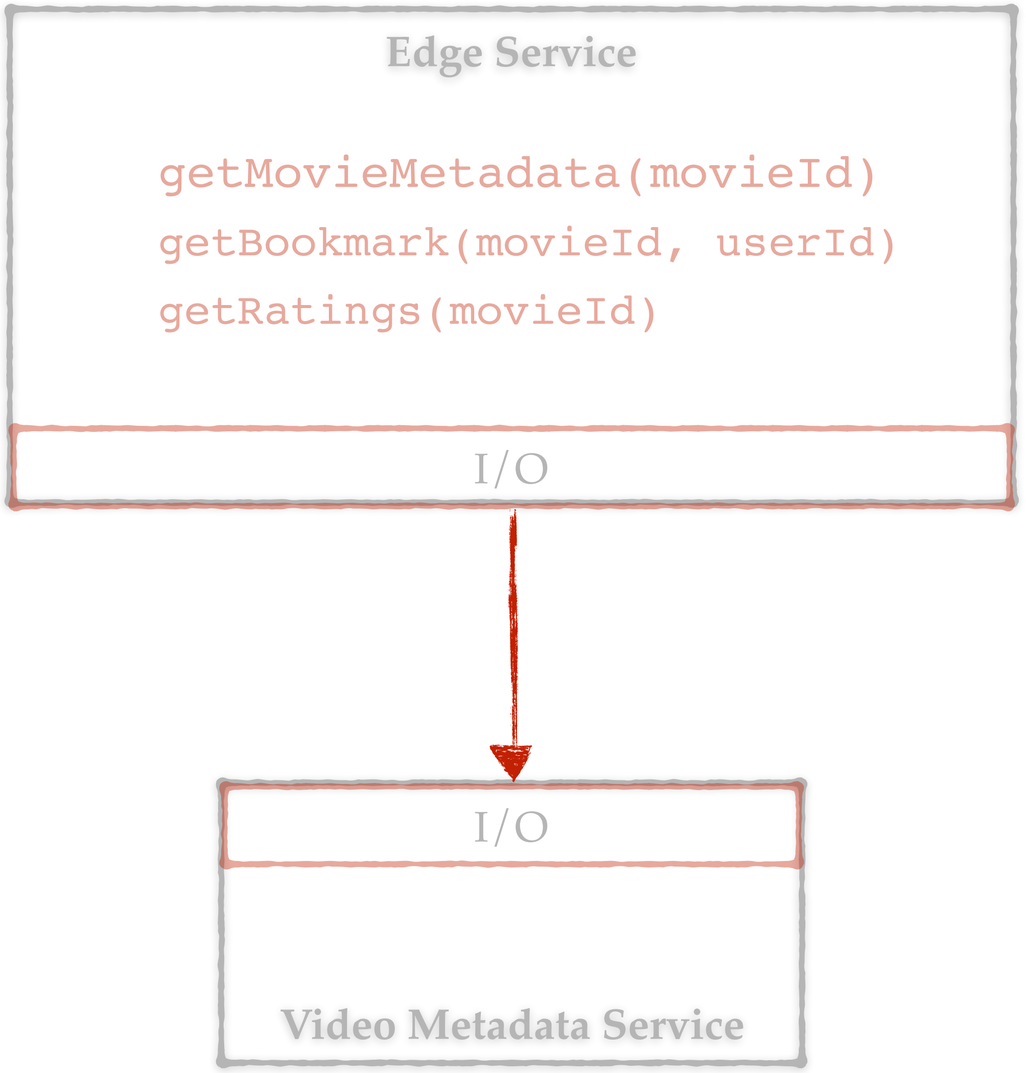
Disclaimer: This is an example and not an exact representation of the processing

Composing the processing
of **a service** into a single control point.

with

Flow & Lifecycle Control

What should be async?



Network Protocol

HTTP/1.1?

HTTP/1.1

GET /movie?id=1 HTTP/1.1

HTTP/1.1

GET /movie?id=2 HTTP/1.1

GET /movie?id=1 HTTP/1.1

HTTP/1.1

GET /movie?id=3 HTTP/1.1

GET /movie?id=2 HTTP/1.1

GET /movie?id=1 HTTP/1.1

HTTP/1.1

GET /movie?id=3 HTTP/1.1

GET /movie?id=2 HTTP/1.1

GET /movie?id=1 HTTP/1.1

HTTP/1.1 200 OK
ID: 1
…

HTTP/1.1

GET /movie?id=3 HTTP/1.1

GET /movie?id=2 HTTP/1.1

GET /movie?id=1 HTTP/1.1

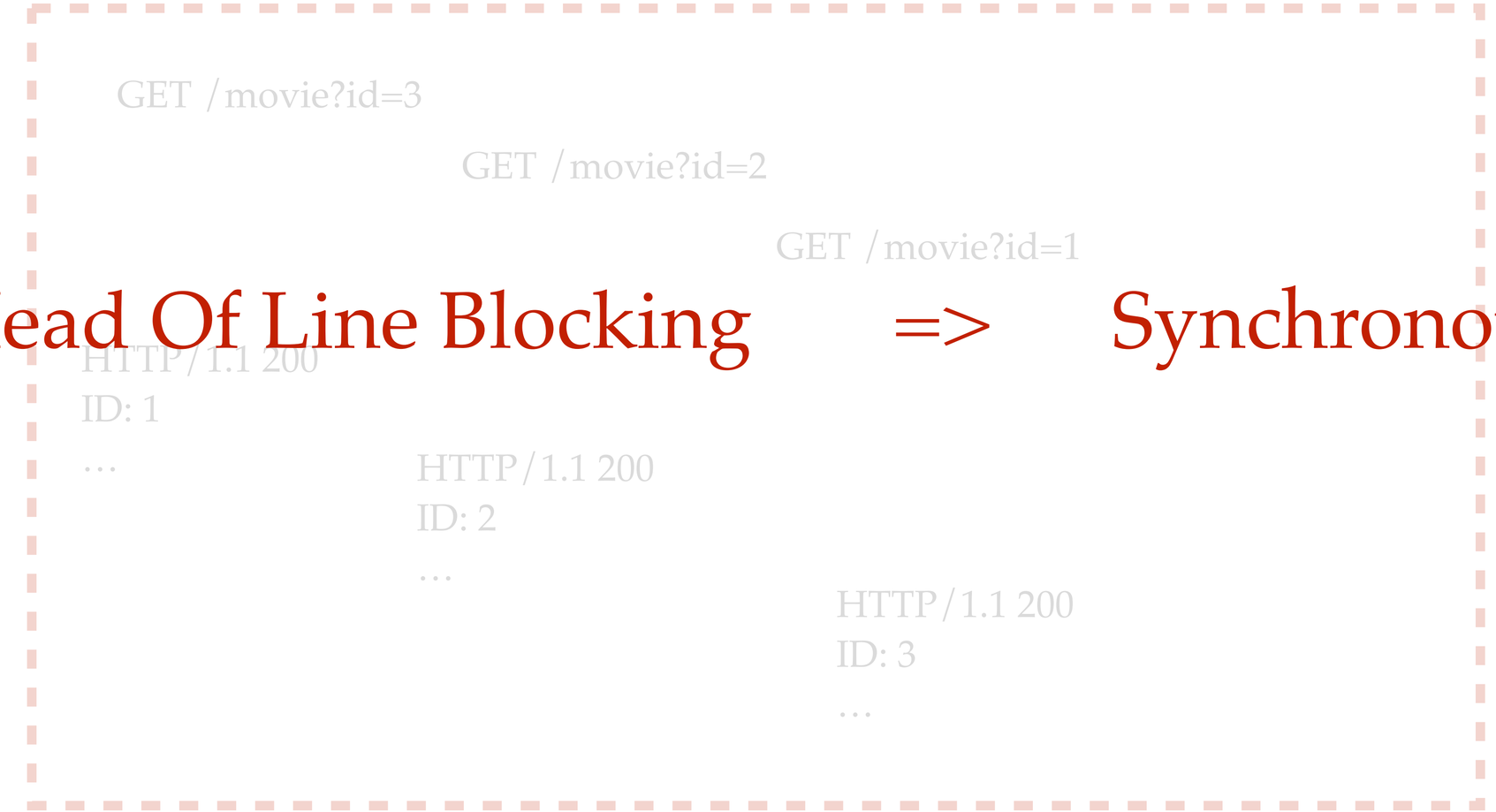
HTTP/1.1 200 OK
ID: 1
…

HTTP/1.1 200 OK
ID: 2
…

HTTP/1.1


GET /movie?id=3 HTTP/1.1
GET /movie?id=2 HTTP/1.1
GET /movie?id=1 HTTP/1.1
HTTP/1.1 200 OK
ID: 1
…
HTTP/1.1 200 OK
ID: 2
…
HTTP/1.1 200 OK
ID: 3
…

HTTP/1.1



Head Of Line Blocking => Synchronous

Network Protocol

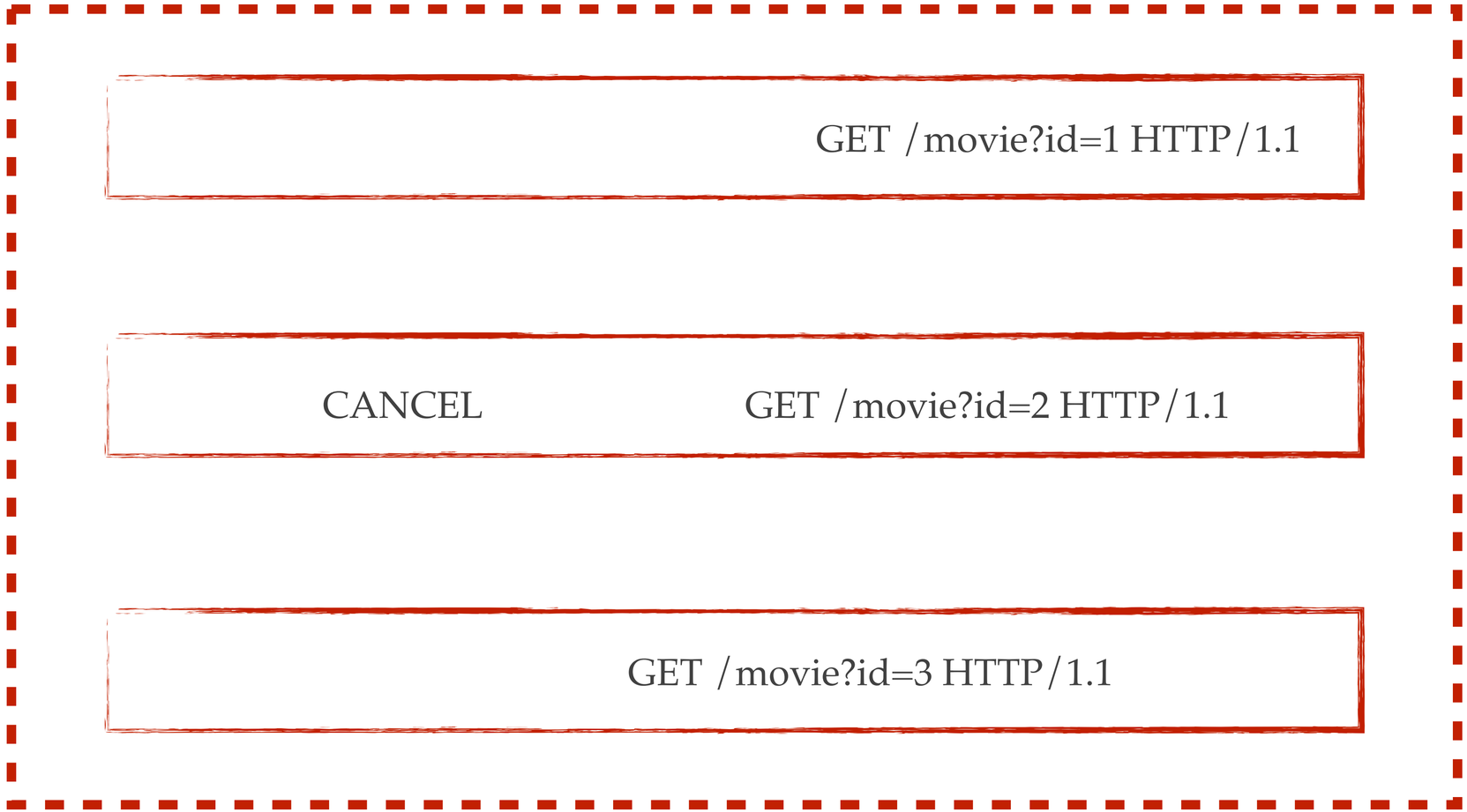
~~HTTP/1.1?~~

Network Protocol

We need a multiplexed bi-directional protocol

Multiplexed

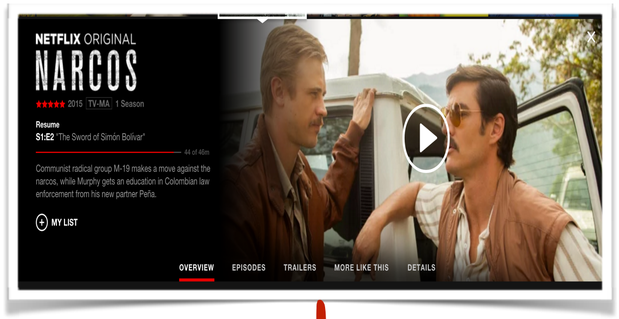
Bi-directional



Composing the processing
of **the entire application** into a single control point.

with

Flow & Lifecycle Control



/movie?id=123

Edge Service

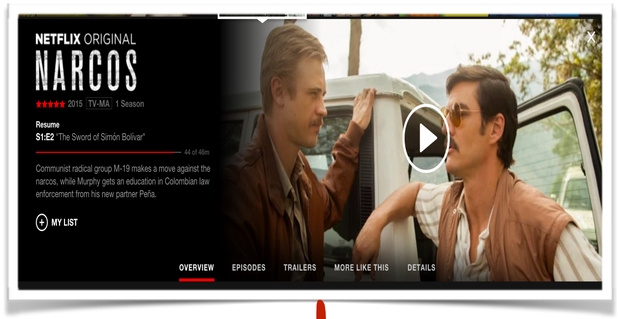
Video Metadata Service

Rating service

C* store

C* store

Disclaimer: This is an example and not an exact representation of the processing



~~/movie?id=123~~ Observable<Movie>

Edge Service

Video Metadata Service

Rating service

C* store

C* store

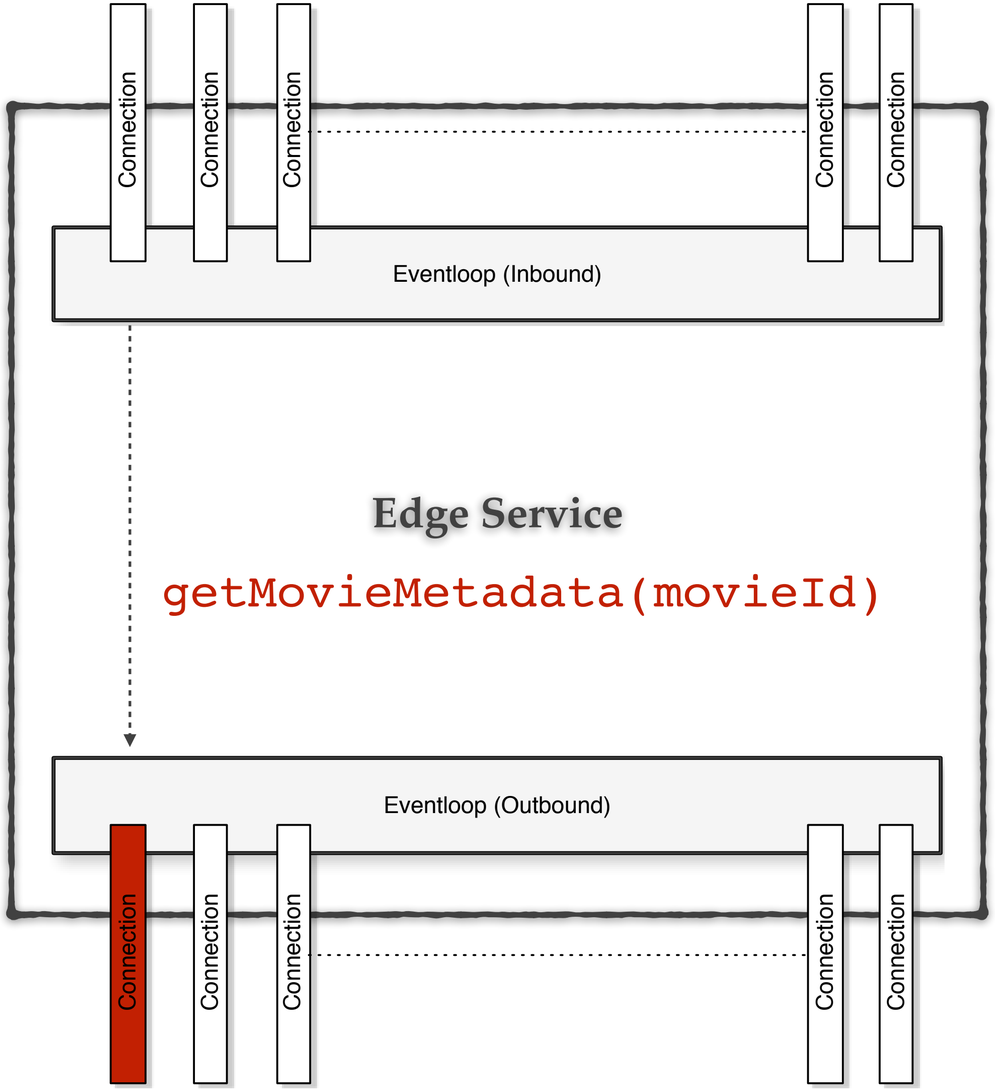
Disclaimer: This is an example and not an exact representation of the processing

`Observable<Movie>`

Composing the processing
of **the entire application** into a single control point.

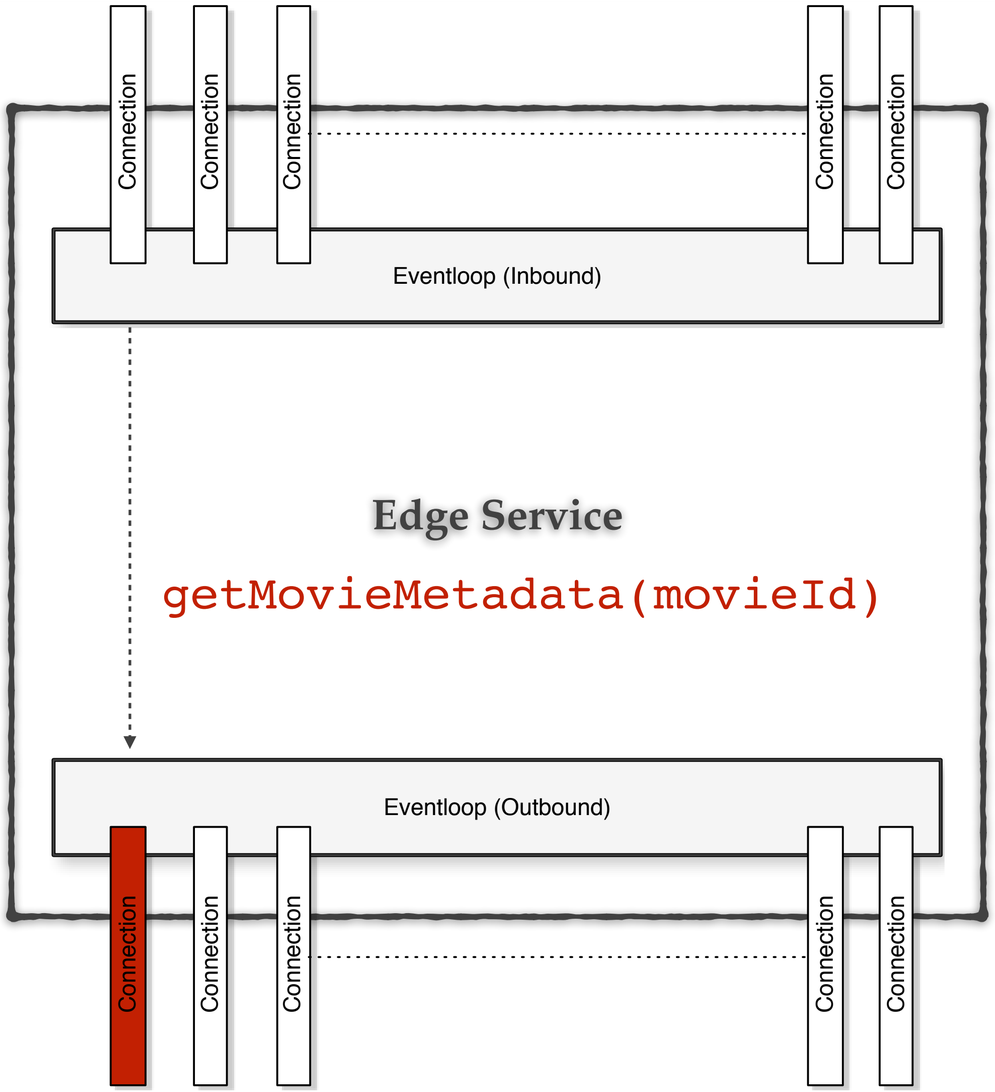
Revisiting the failure modes

Latency



Disclaimer: This is an example and not an exact representation of the processing

Latency



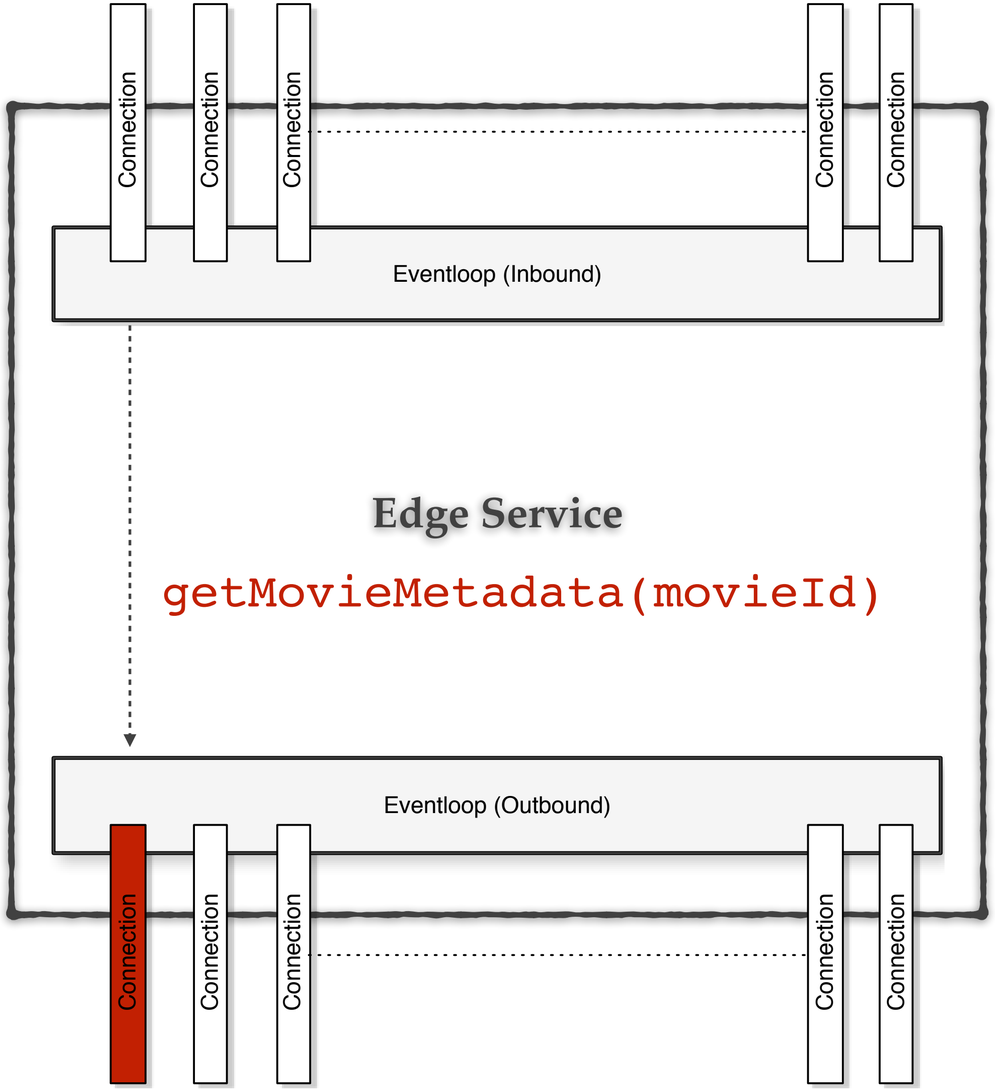
Impact is localized to the connection.

Disclaimer: This is an example and not an exact representation of the processing

Latency

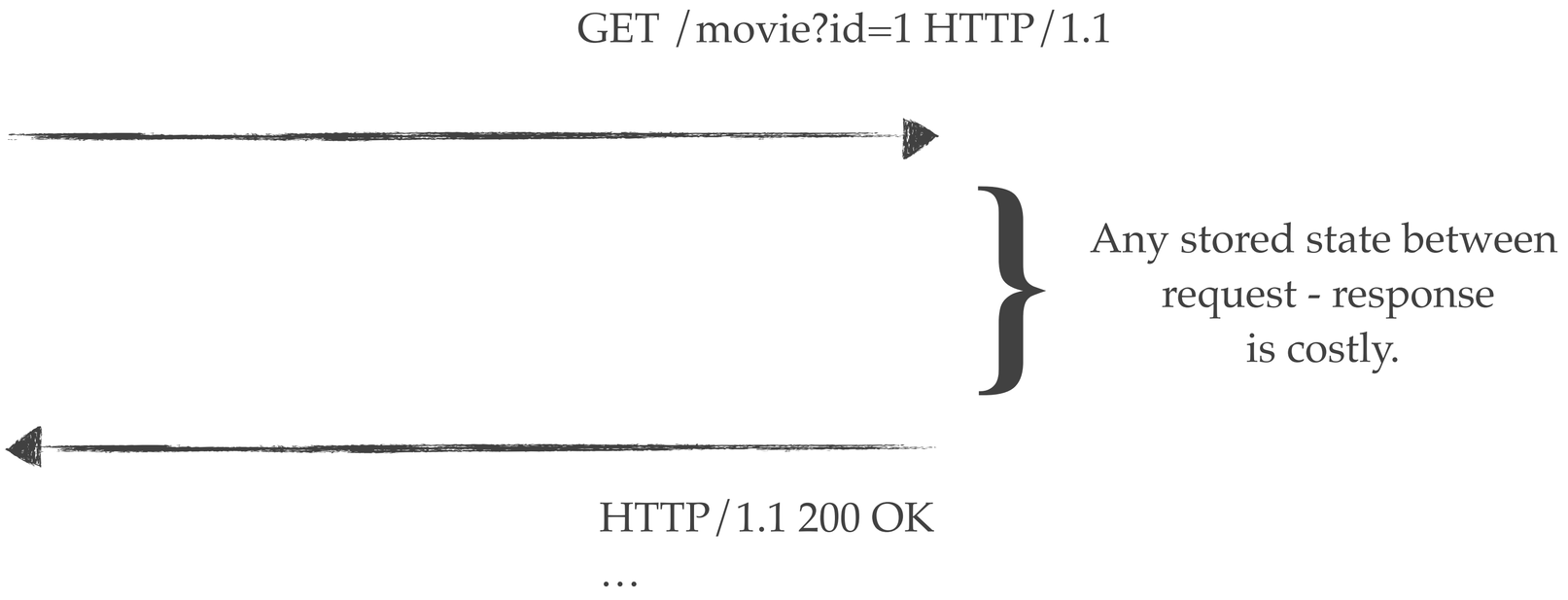
Impact is localized to the connection.

An outstanding request has little cost.



Disclaimer: This is an example and not an exact representation of the processing

An outstanding request has little cost.



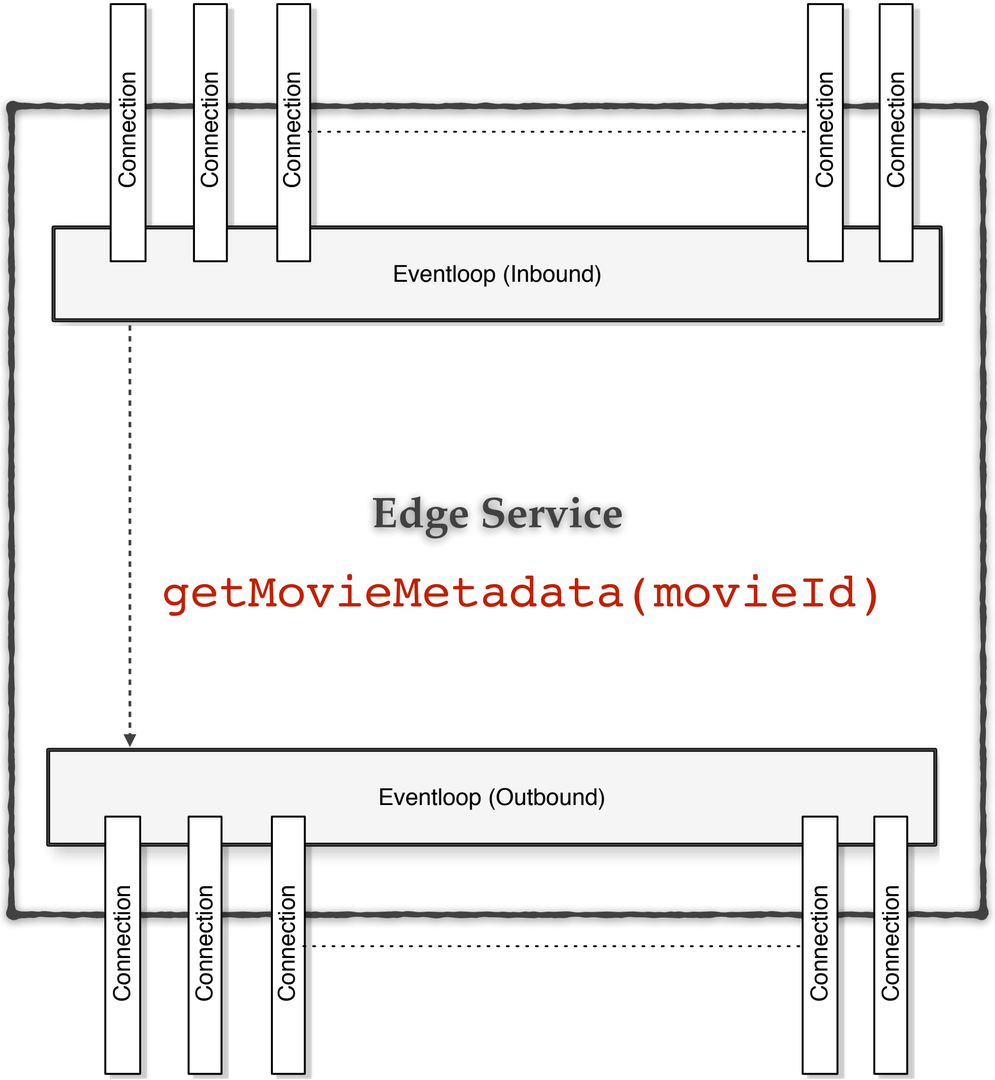
Outstanding requests have low cost

so

Latency is a lesser evil in asynchronous systems.

Overload & Thundering Herds

Reduce work done when overloaded

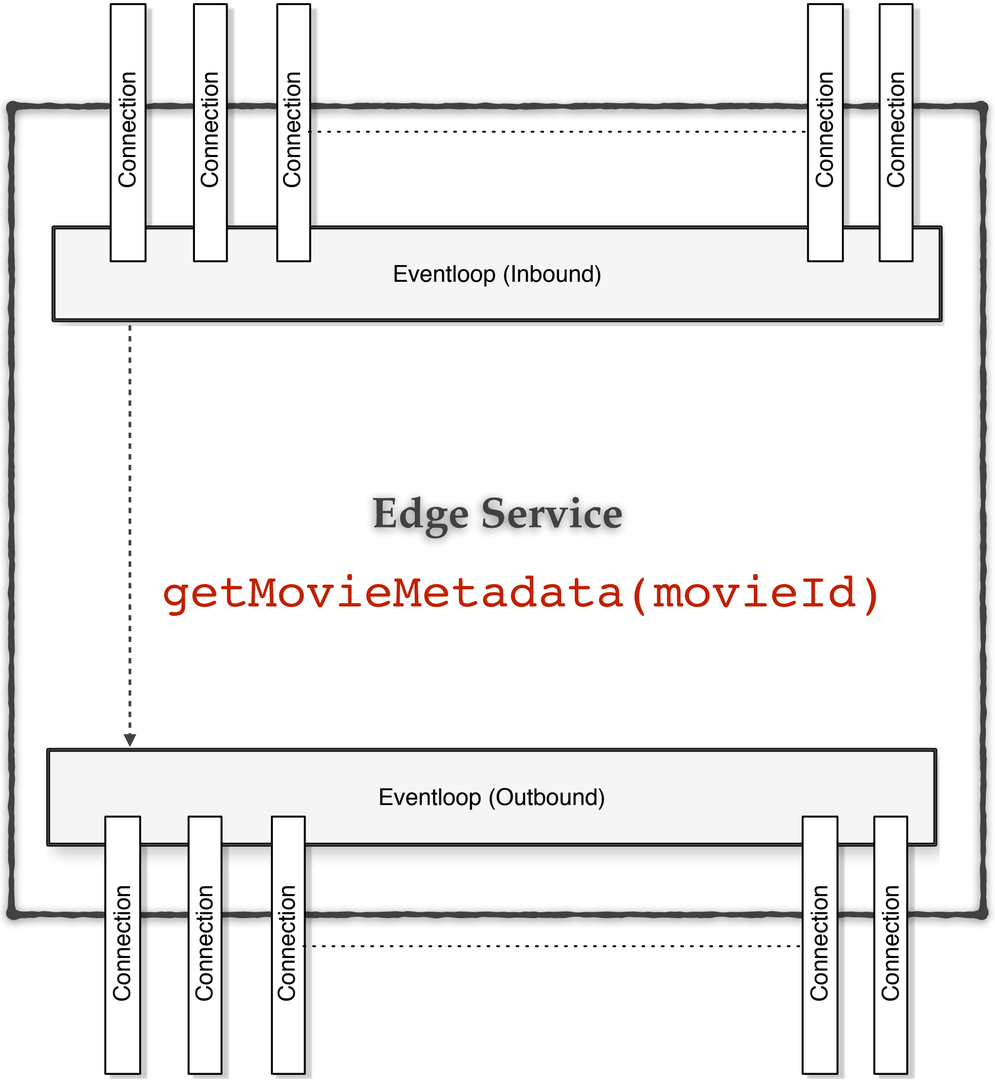


Disclaimer: This is an example and not an exact representation of the processing

Overload & Thundering Herds

Stop accepting new requests.

Reduce work done when overloaded



Disclaimer: This is an example and not an exact representation of the processing

Stop accepting new requests

Non-blocking I/O gives better control

Stop accepting new requests

But … we are still “throttling”

Stop accepting new requests

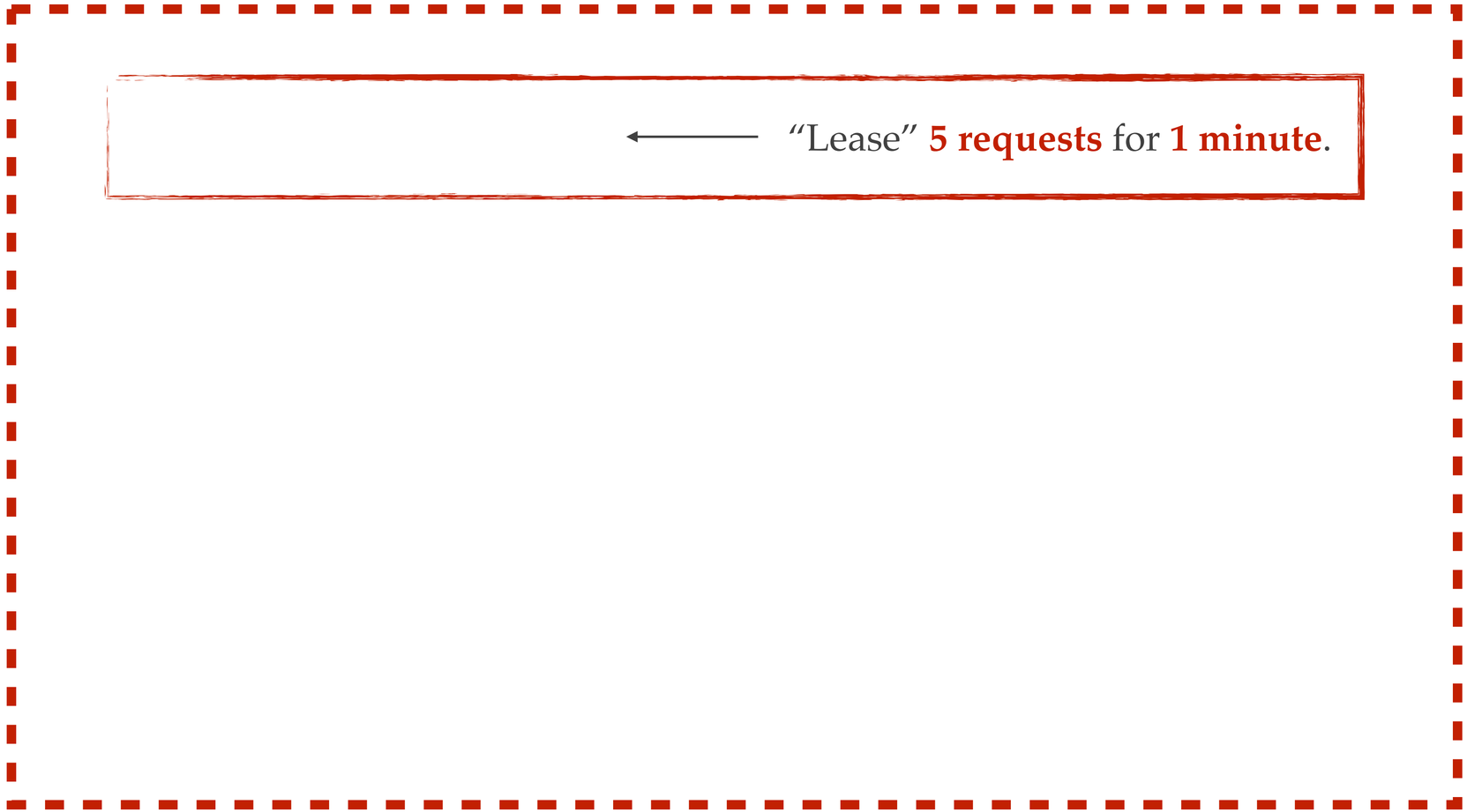
Are we being empathetic?

Request-leasing

http://reactivesocket.io/

Request-leasing

Request-leasing

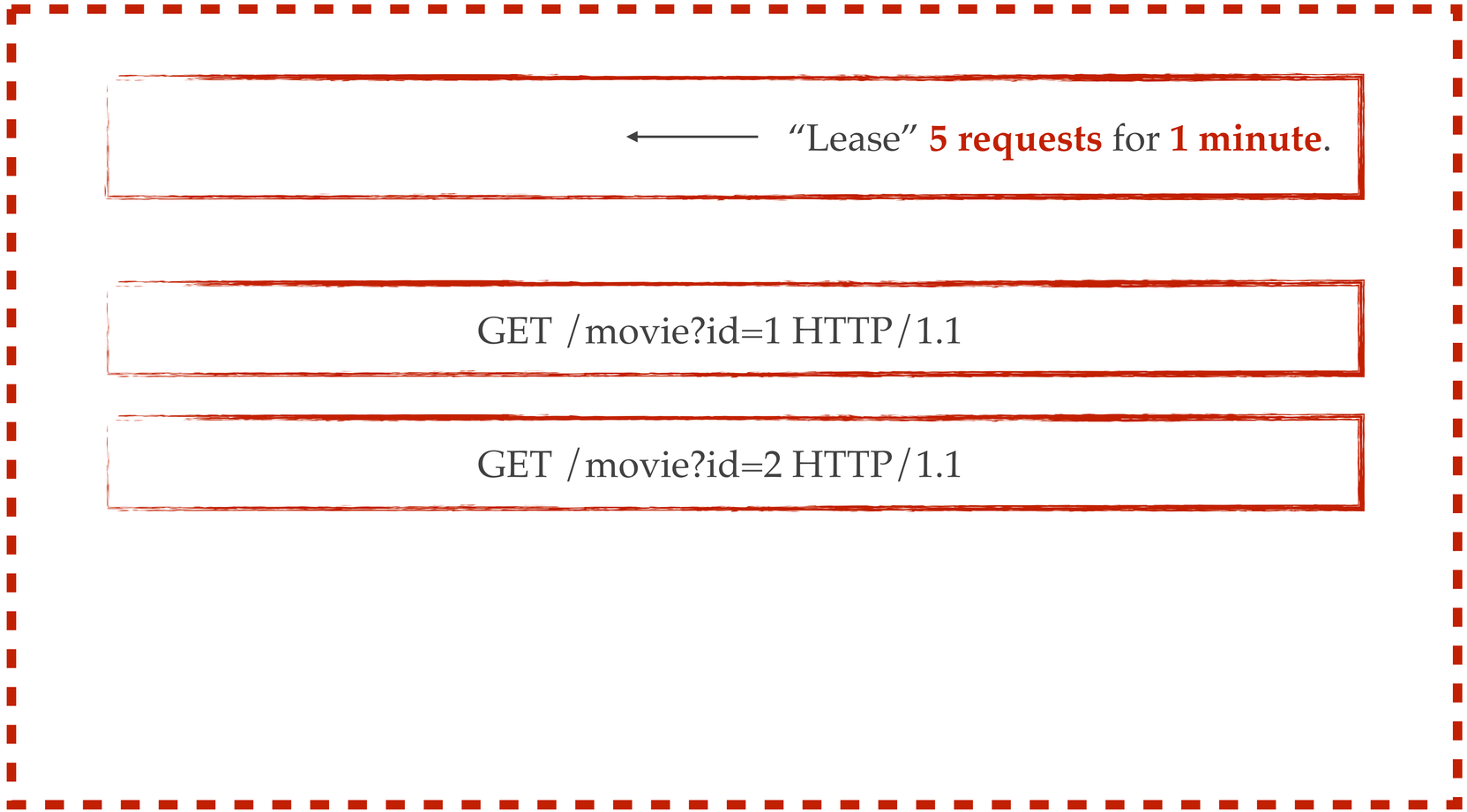


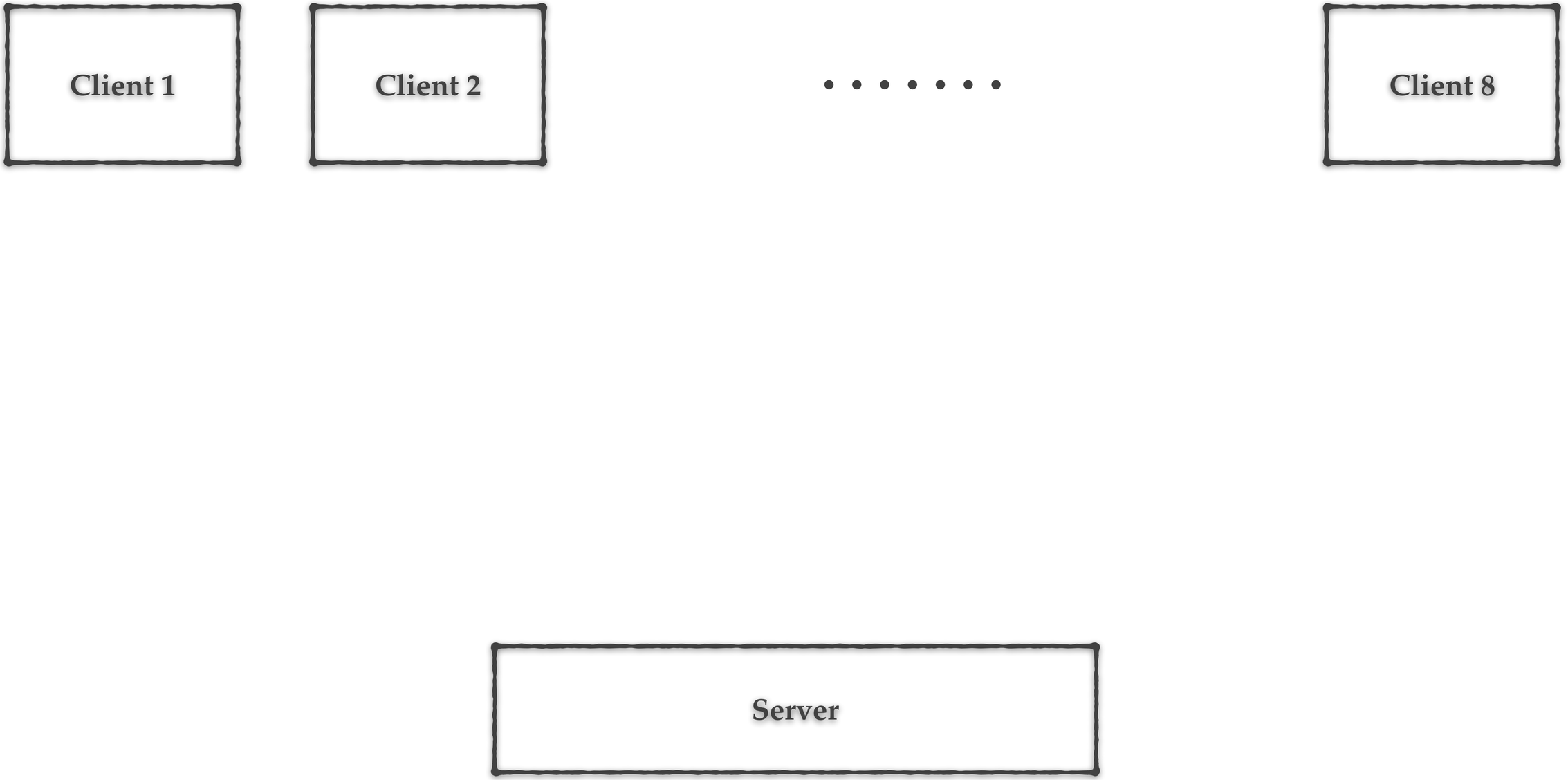
Peer 2

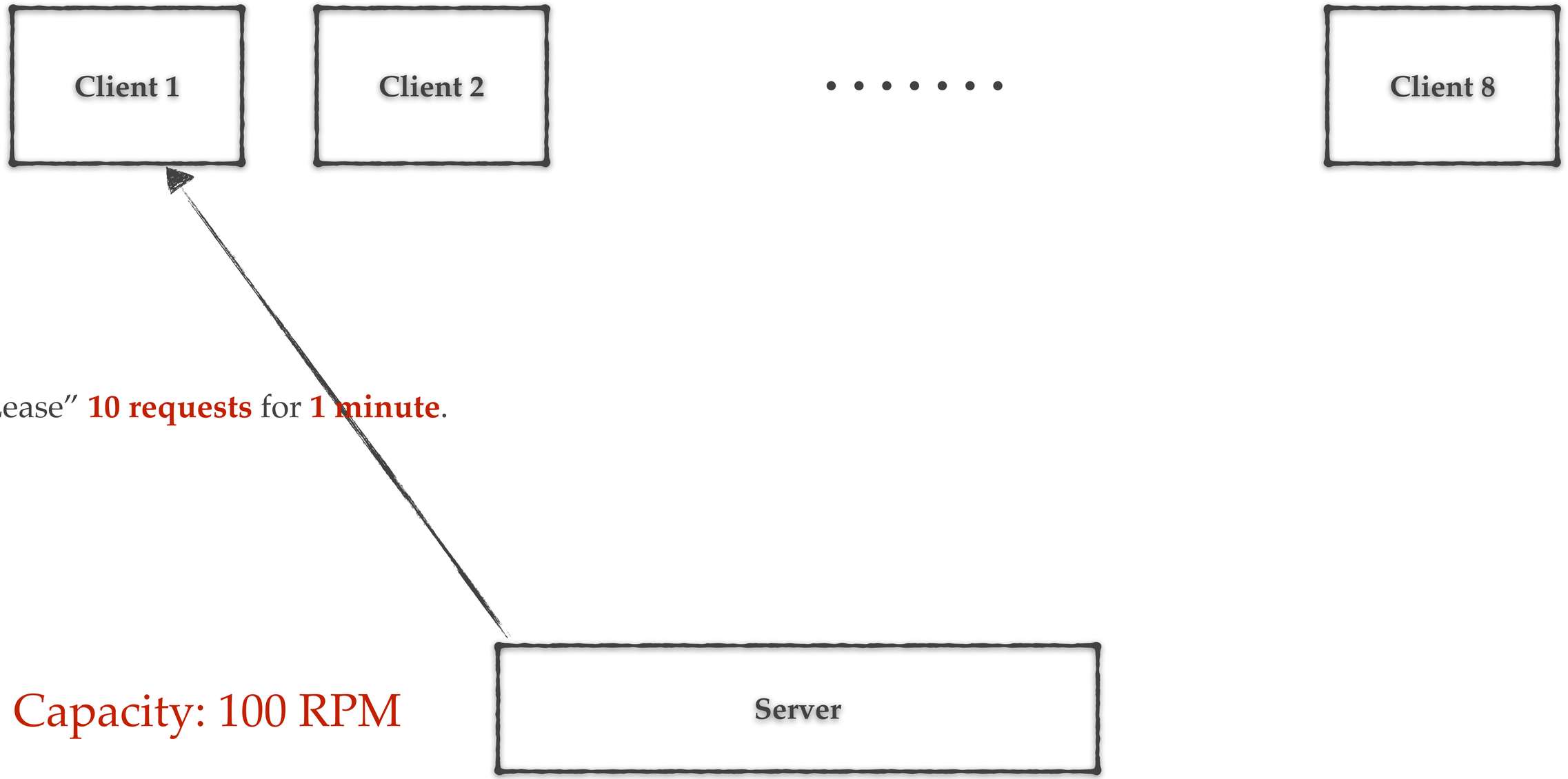
Peer 1

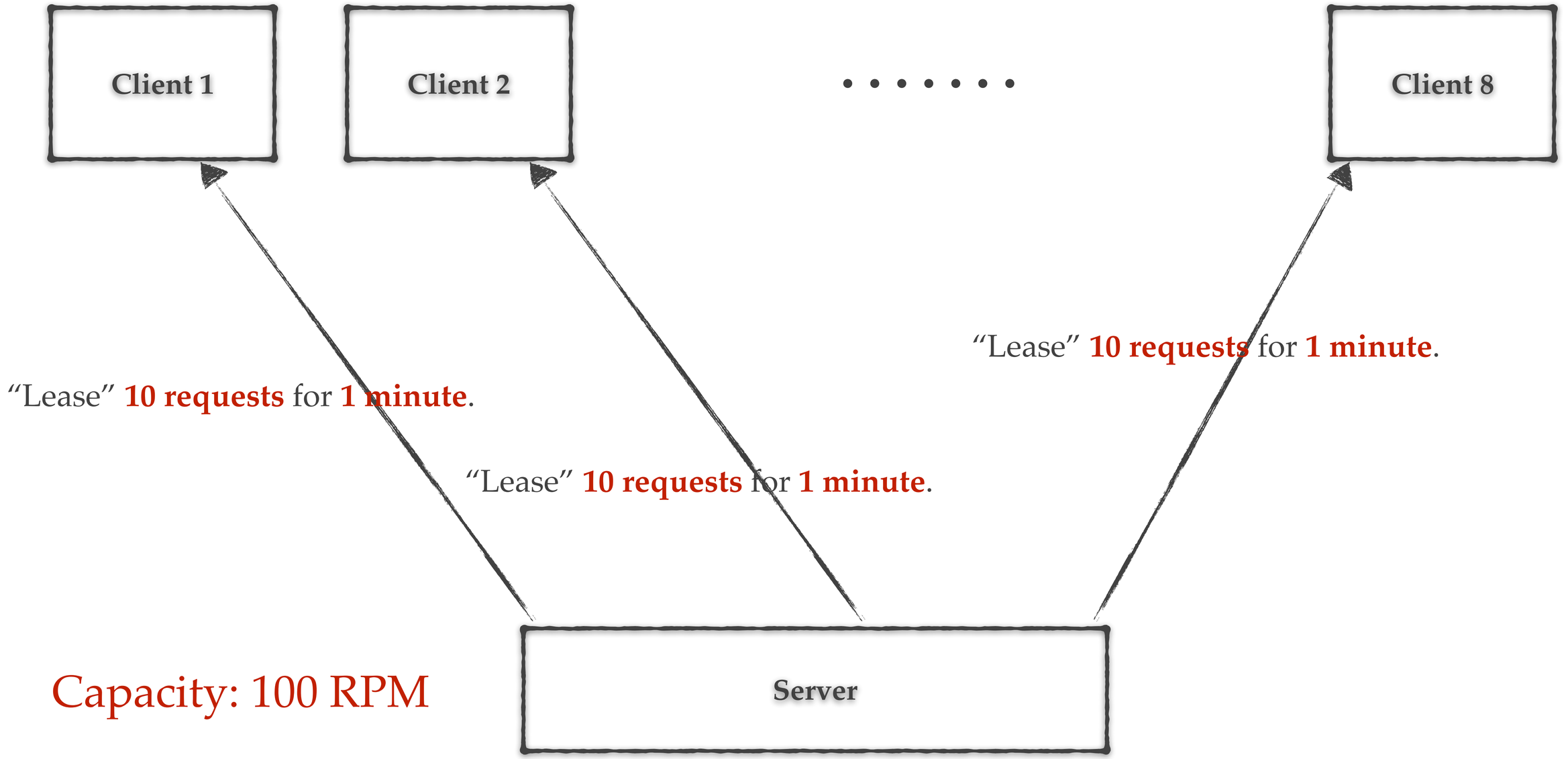
Network connection

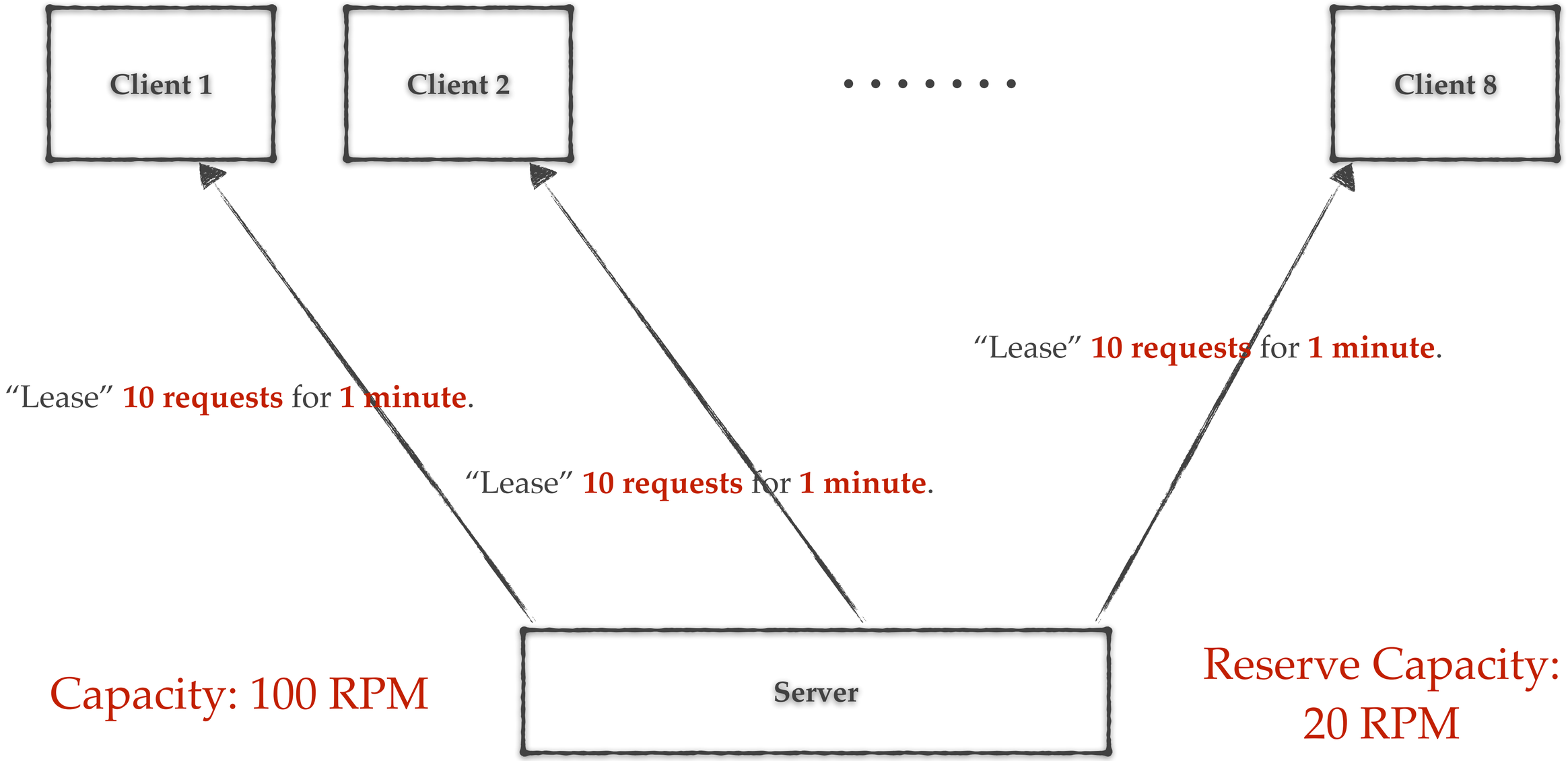
Request-leasing




Client 1
Client 2
Client 8
Server


Client 1
Client 2
Client 8
"Lease" 10 requests for 1 minute.
Server
Capacity: 100 RPM


Client 1
Client 2
Client 8
"Lease" 10 requests for 1 minute.
"Lease" 10 requests for 1 minute.
"Lease" 10 requests for 1 minute.
Server
Capacity: 100 RPM


Client 1
Client 2
Client 8
"Lease" 10 requests for 1 minute.
"Lease" 10 requests for 1 minute.
"Lease" 10 requests for 1 minute.
Server
Capacity: 100 RPM
Reserve Capacity:
20 RPM

“Lease” **10 requests** for **1 minute**.

Time bound lease.

“Lease” **10 requests** for **1 minute**.

Time bound lease.

No extra work for cancelling leases.

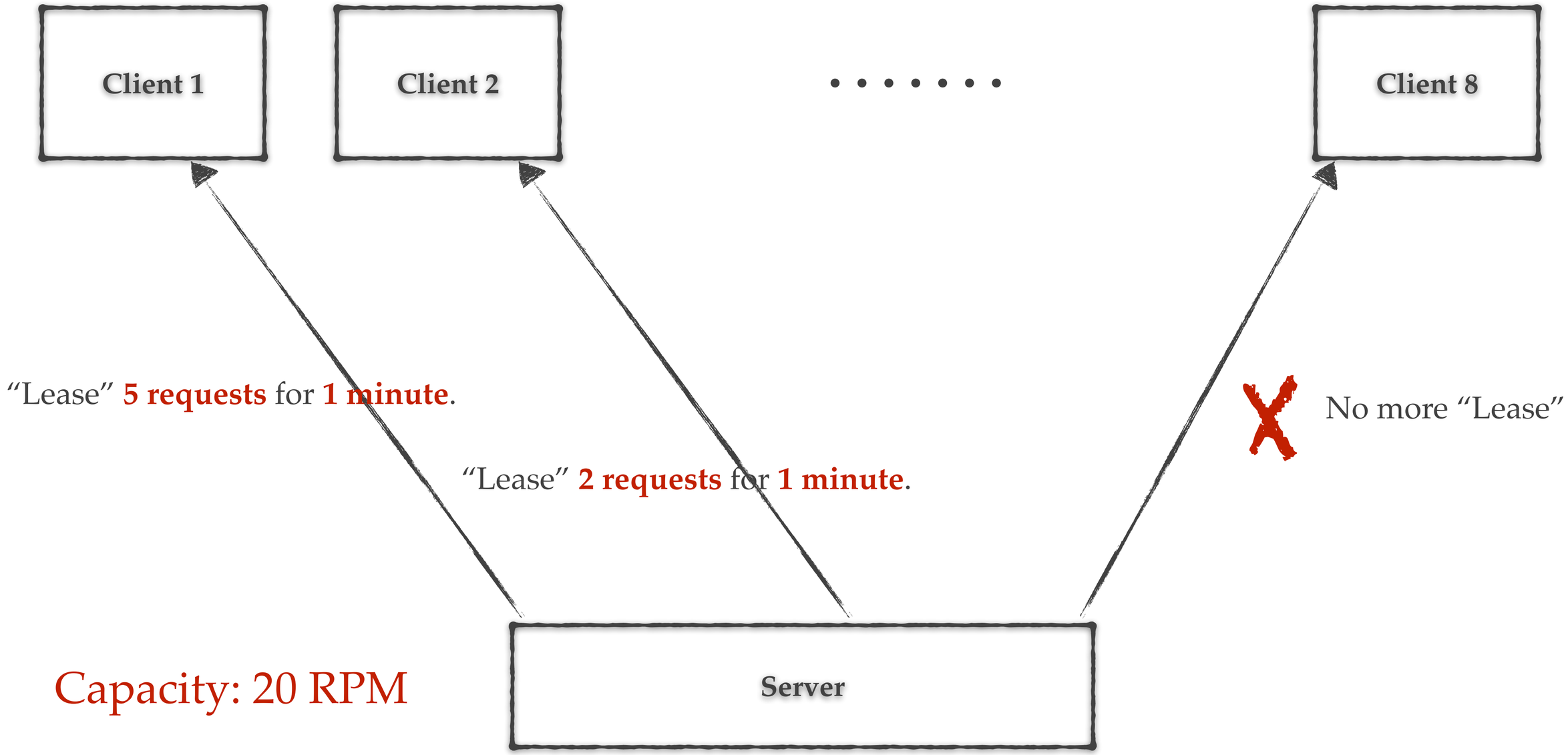
“Lease” **10 requests** for **1 minute**.

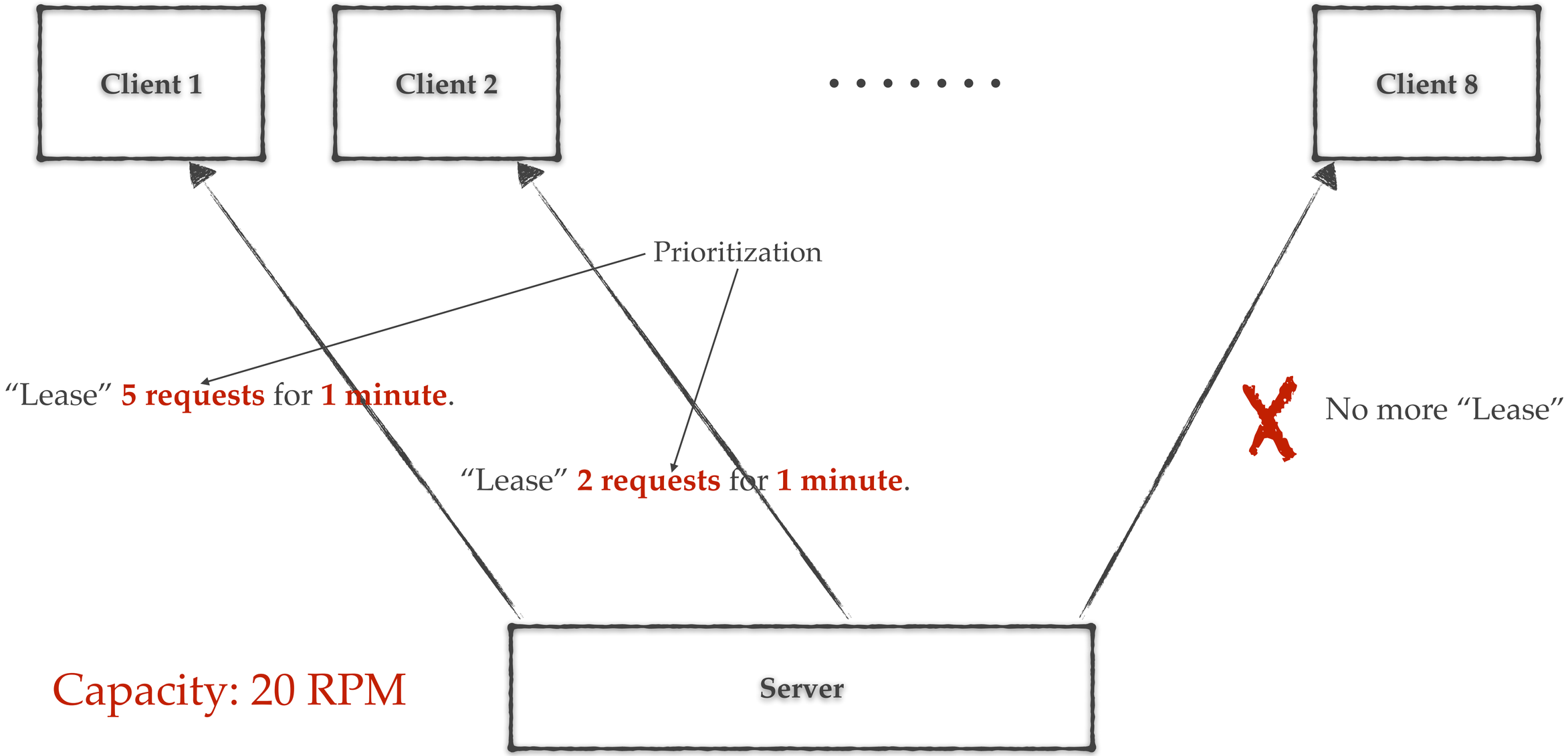
Time bound lease.

No extra work for cancelling leases.

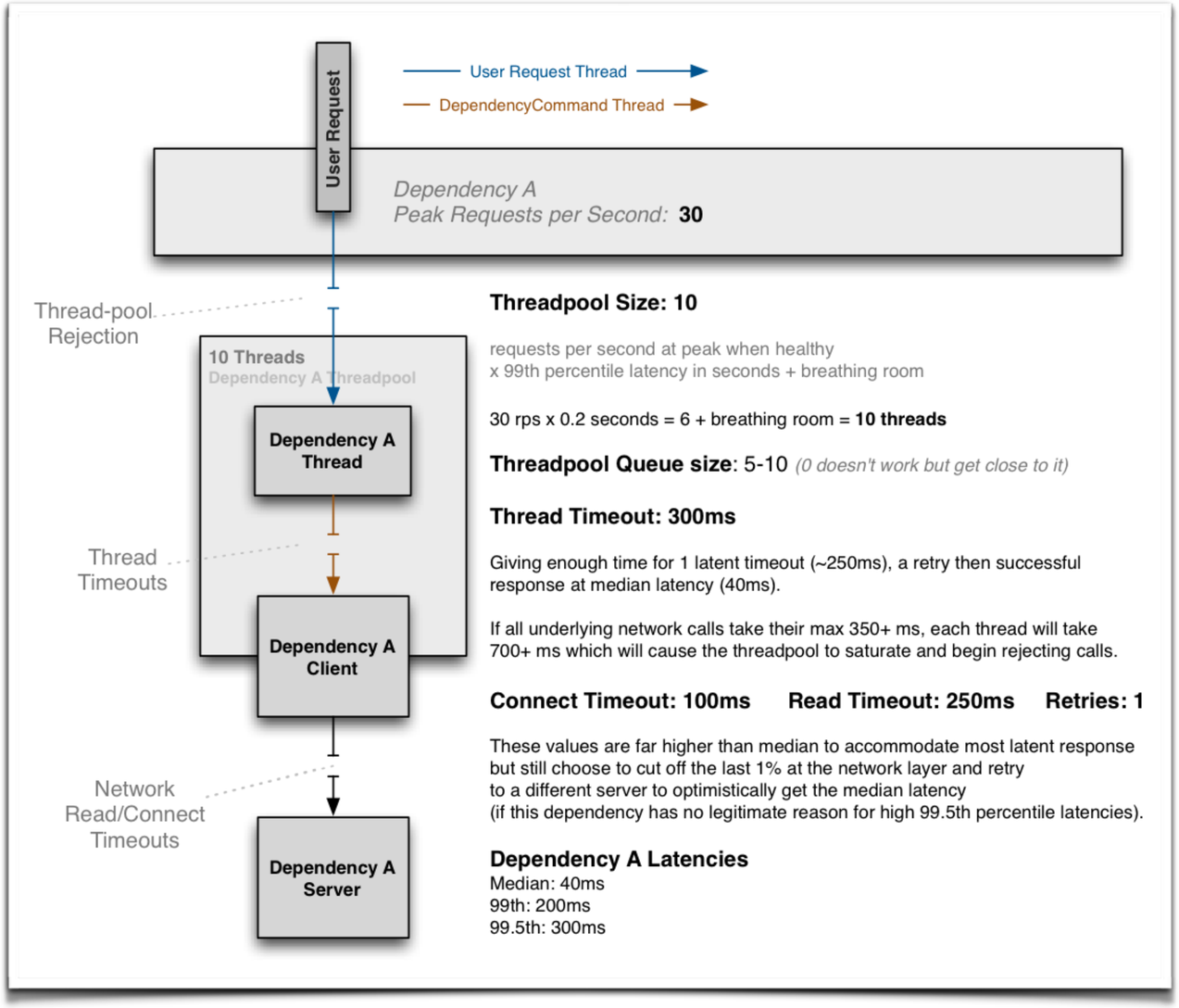
Receiver controls the flow of requests

When things go south


Client 1
Client 2
Client 8
"Lease" 5 requests for 1 minute.
"Lease" 2 requests for 1 minute.
No more "Lease"
Server
Capacity: 20 RPM

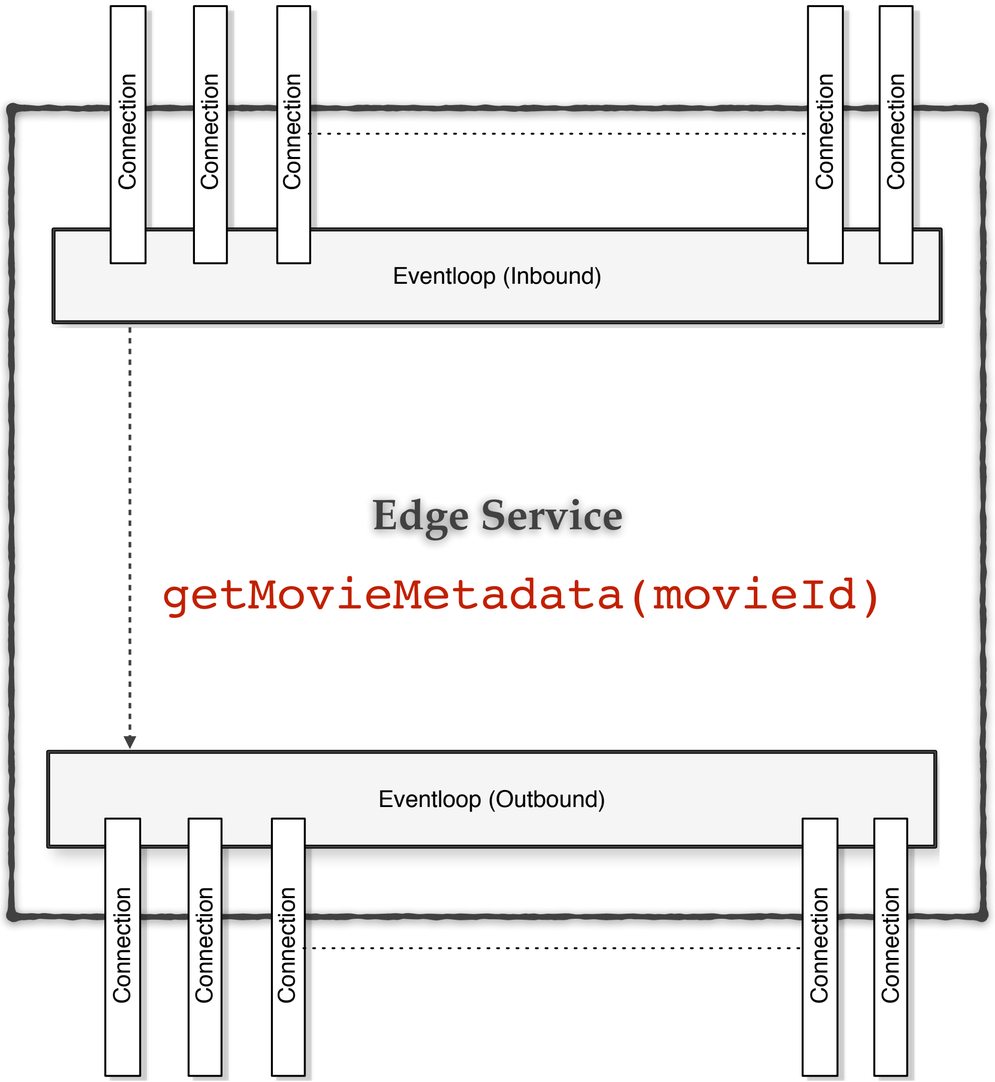

Client 1
Client 2
Client 8
Prioritization
"Lease" 5 requests for 1 minute.
"Lease" 2 requests for 1 minute.
No more "Lease"
Server
Capacity: 20 RPM

Managing client configs?



Threadpools?

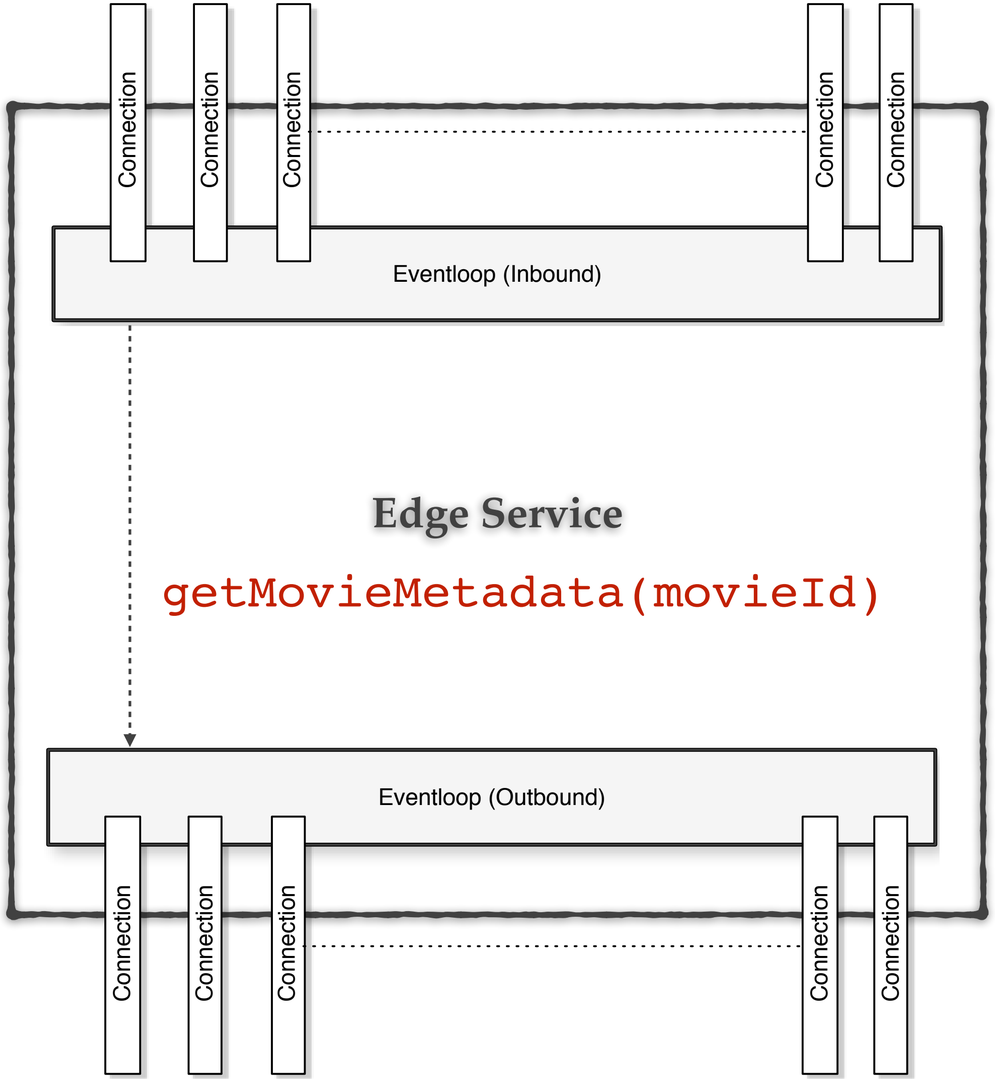
I/O
is
non-blocking.



Disclaimer: This is an example and not an exact representation of the processing

Threadpools?

Application code
is
non-blocking.



Disclaimer: This is an example and not an exact representation of the processing

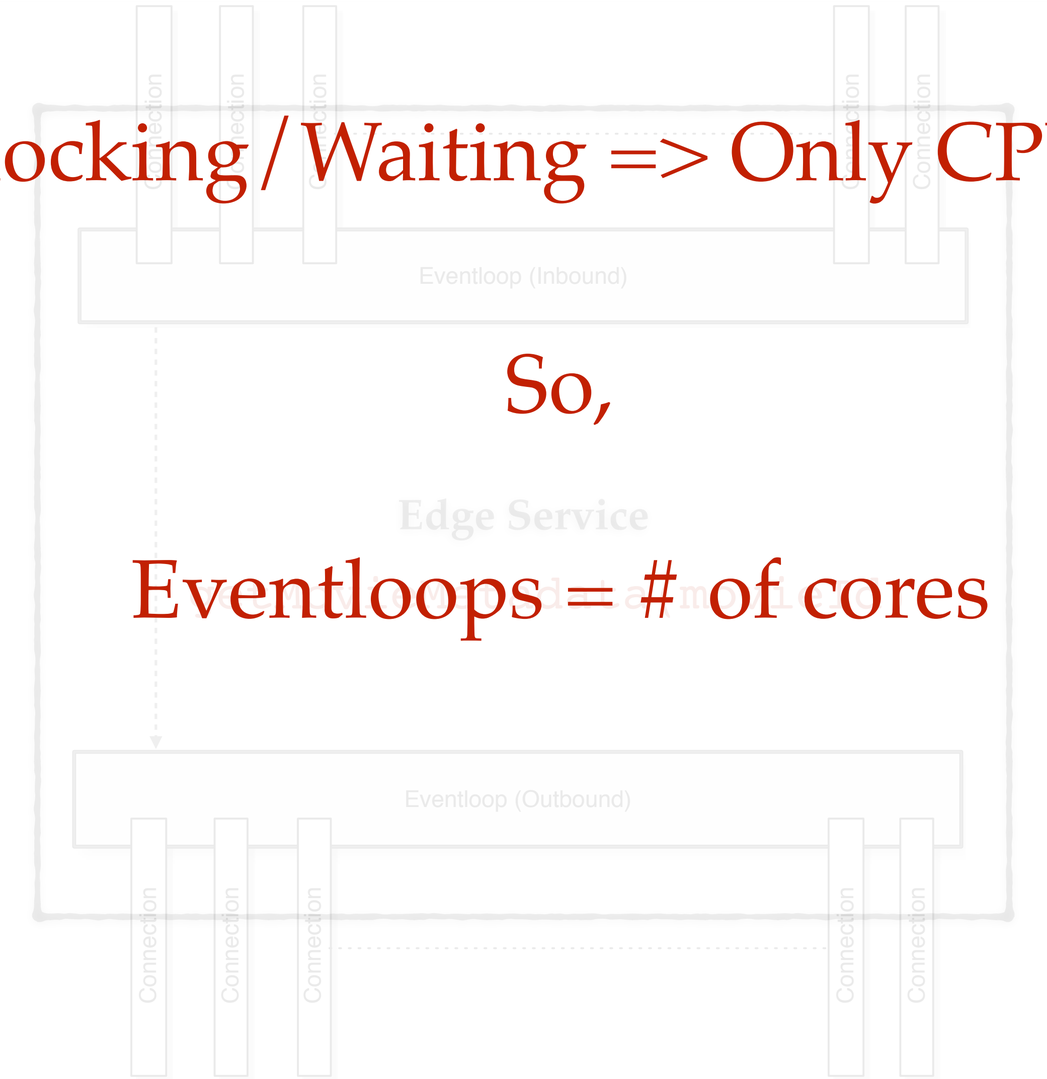
Threadpools?

No blocking / Waiting => Only CPU work

Disclaimer: This is an example and not an exact representation of the processing

Threadpools?

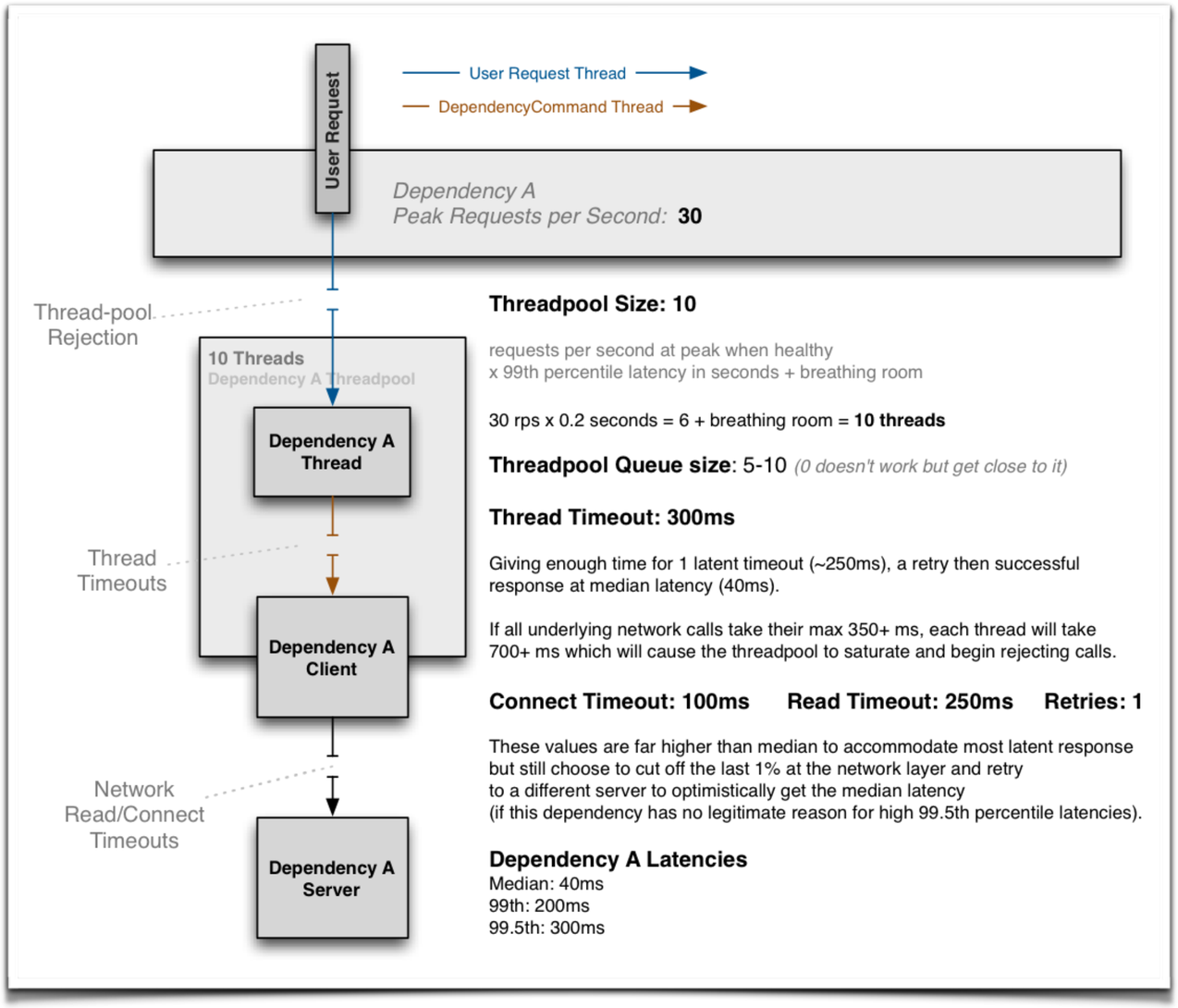
No blocking / Waiting => Only CPU work



So,

Eventloops = # of cores

Disclaimer: This is an example and not an exact representation of the processing



Threadpool Size: 10

requests per second at peak when healthy
x 99th percentile latency in seconds + breathing room

30 rps x 0.2 seconds = 6 + breathing room = **10 threads**

Threadpool Queue size: 5-10 *(0 doesn't work but get close to it)*

Thread Timeout: 300ms

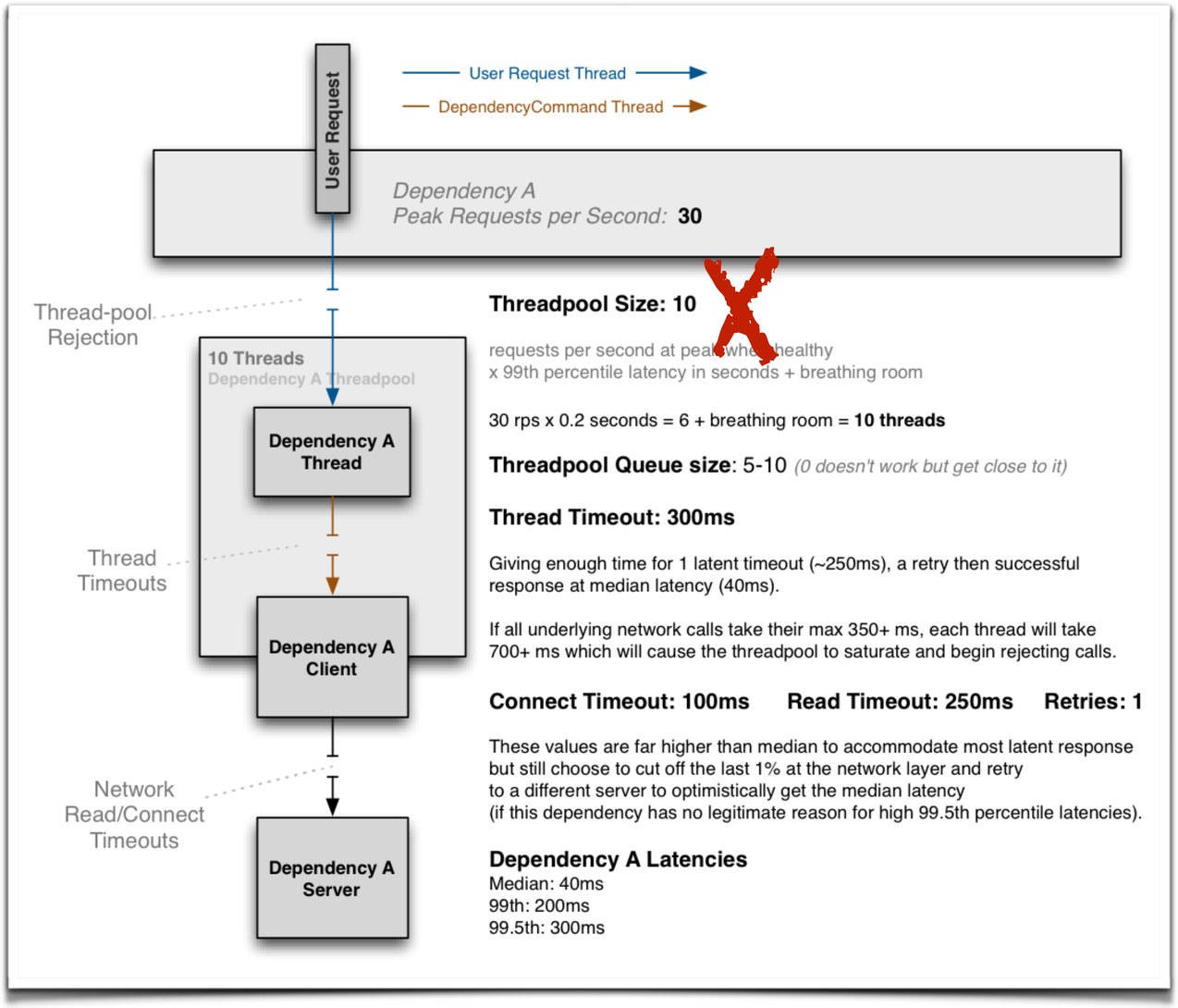
Giving enough time for 1 latent timeout (~250ms), a retry then successful response at median latency (40ms).

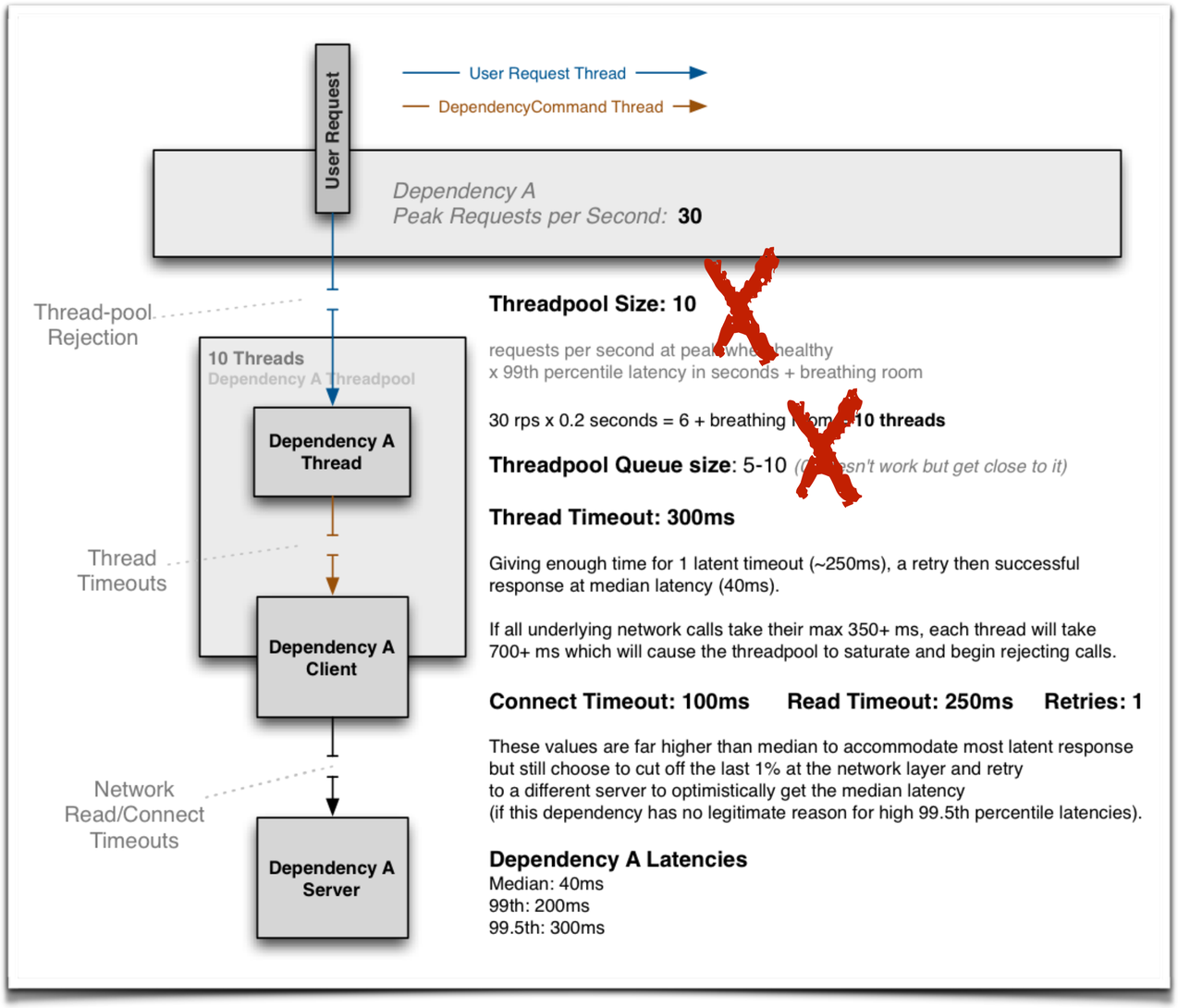
If all underlying network calls take their max 350+ ms, each thread will take 700+ ms which will cause the threadpool to saturate and begin rejecting calls.

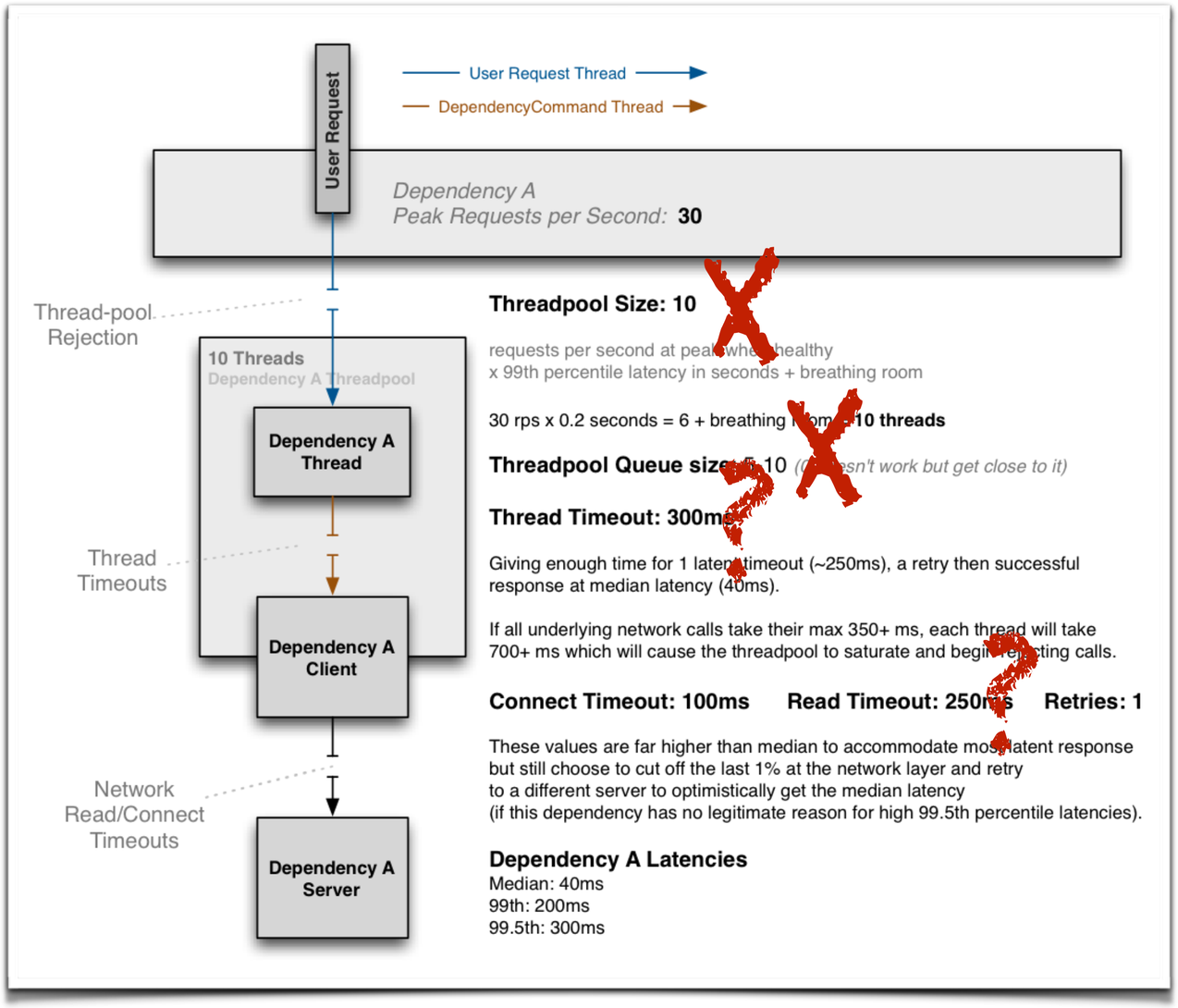
Connect Timeout: 100ms **Read Timeout: 250ms** **Retries: 1**

These values are far higher than median to accommodate most latent response but still choose to cut off the last 1% at the network layer and retry to a different server to optimistically get the median latency (if this dependency has no legitimate reason for high 99.5th percentile latencies).

Dependency A Latencies
Median: 40ms
99th: 200ms
99.5th: 300ms


User Request Thread
DependencyCommand Thread
User Request
Dependency A
Peak Requests per Second: 30
Thread-pool Rejection
10 Threads
Dependency A Threadpool
Dependency A Thread
Thread Timeouts
Dependency A Client
Network Read/Connect Timeouts
Dependency A Server
Threadpool Size: 10
requests per second at peak when healthy
x 99th percentile latency in seconds + breathing room
30 rps x 0.2 seconds = 6 + breathing room = 10 threads
Threadpool Queue size: 5-10 (0 doesn't work but get close to it)
Thread Timeout: 300ms
Giving enough time for 1 latent timeout (~250ms), a retry then successful response at median latency (40ms).
If all underlying network calls take their max 350+ ms, each thread will take 700+ ms which will cause the threadpool to saturate and begin rejecting calls.
Connect Timeout: 100ms Read Timeout: 250ms Retries: 1
These values are far higher than median to accommodate most latent response but still choose to cut off the last 1% at the network layer and retry to a different server to optimistically get the median latency (if this dependency has no legitimate reason for high 99.5th percentile latencies).
Dependency A Latencies
Median: 40ms
99th: 200ms
99.5th: 300ms


User Request Thread
DependencyCommand Thread
User Request
Dependency A
Peak Requests per Second: 30
Thread-pool Rejection
10 Threads
Dependency A Threadpool
Dependency A Thread
Thread Timeouts
Dependency A Client
Network Read/Connect Timeouts
Dependency A Server
Threadpool Size: 10
requests per second at peak when healthy
x 99th percentile latency in seconds + breathing room
30 rps x 0.2 seconds = 6 + breathing room = 10 threads
Threadpool Queue size: 5-10 (doesn't work but get close to it)
Thread Timeout: 300ms
Giving enough time for 1 latent timeout (~250ms), a retry then successful response at median latency (40ms).
If all underlying network calls take their max 350+ ms, each thread will take 700+ ms which will cause the threadpool to saturate and begin rejecting calls.
Connect Timeout: 100ms Read Timeout: 250ms Retries: 1
These values are far higher than median to accommodate most latent response but still choose to cut off the last 1% at the network layer and retry to a different server to optimistically get the median latency (if this dependency has no legitimate reason for high 99.5th percentile latencies).
Dependency A Latencies
Median: 40ms
99th: 200ms
99.5th: 300ms


User Request Thread
DependencyCommand Thread
User Request
Dependency A
Peak Requests per Second: 30
Thread-pool Rejection
10 Threads
Dependency A Threadpool
Dependency A Thread
Thread Timeouts
Dependency A Client
Network Read/Connect Timeouts
Dependency A Server
Threadpool Size: 10
requests per second at peak when healthy
x 99th percentile latency in seconds + breathing room
30 rps x 0.2 seconds = 6 + breathing room = 10 threads
Threadpool Queue size: 5-10 (doesn't work but get close to it)
Thread Timeout: 300ms
Giving enough time for 1 latent timeout (~250ms), a retry then successful response at median latency (40ms).
If all underlying network calls take their max 350+ ms, each thread will take 700+ ms which will cause the threadpool to saturate and begin rejecting calls.
Connect Timeout: 100ms
Read Timeout: 250ms
Retries: 1
These values are far higher than median to accommodate most latent response but still choose to cut off the last 1% at the network layer and retry to a different server to optimistically get the median latency (if this dependency has no legitimate reason for high 99.5th percentile latencies).
Dependency A Latencies
Median: 40ms
99th: 200ms
99.5th: 300ms

Case for timeouts?

Case for timeouts?

Read Timeouts

✤ Useful in unblocking threads on socket reads.

Thread Timeouts

✤ Unblock the calling thread.

✤ Business level SLA.

Case for timeouts?

Read Timeouts

✤ Useful in unblocking threads on socket reads.

Thread Timeouts

✤ Unblock the calling thread.

✤ Business level SLA.

As there are no blocking calls.

Case for timeouts?

Read Timeouts

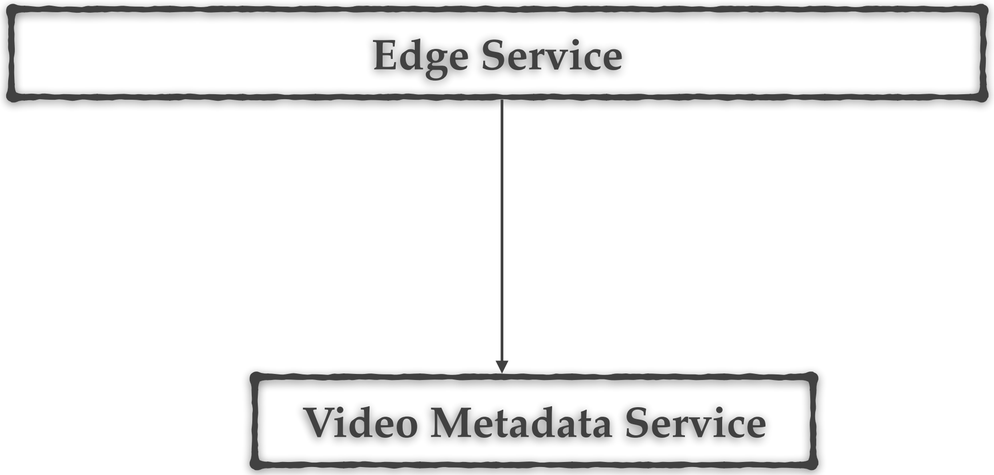
- ~~Useful in unblocking threads on socket reads.~~

Thread Timeouts

- ~~Unblock the calling thread.~~

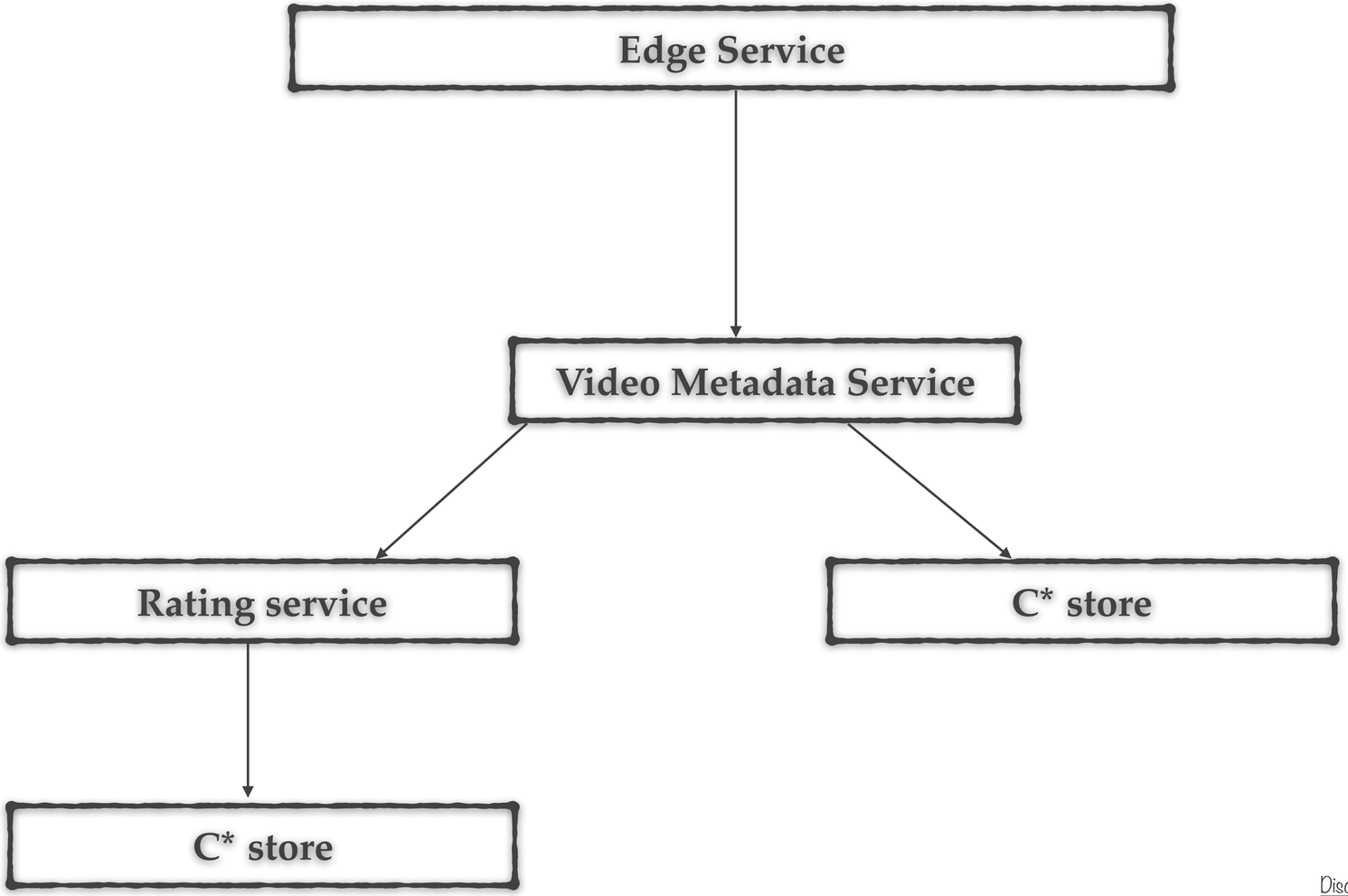
- Business level SLA.

Business level SLA



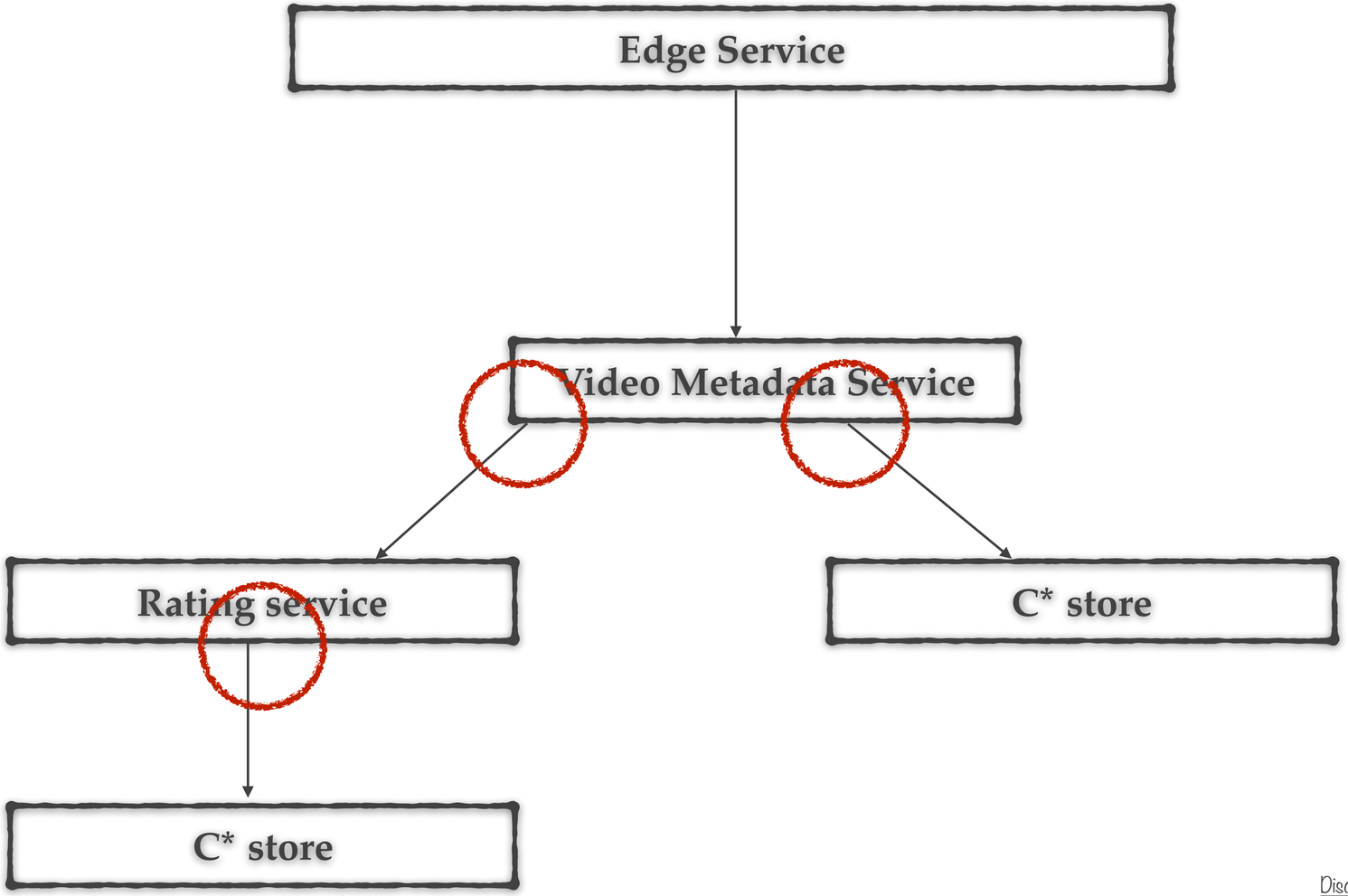
Disclaimer: This is an example and not an exact representation of the processing

Business level SLA



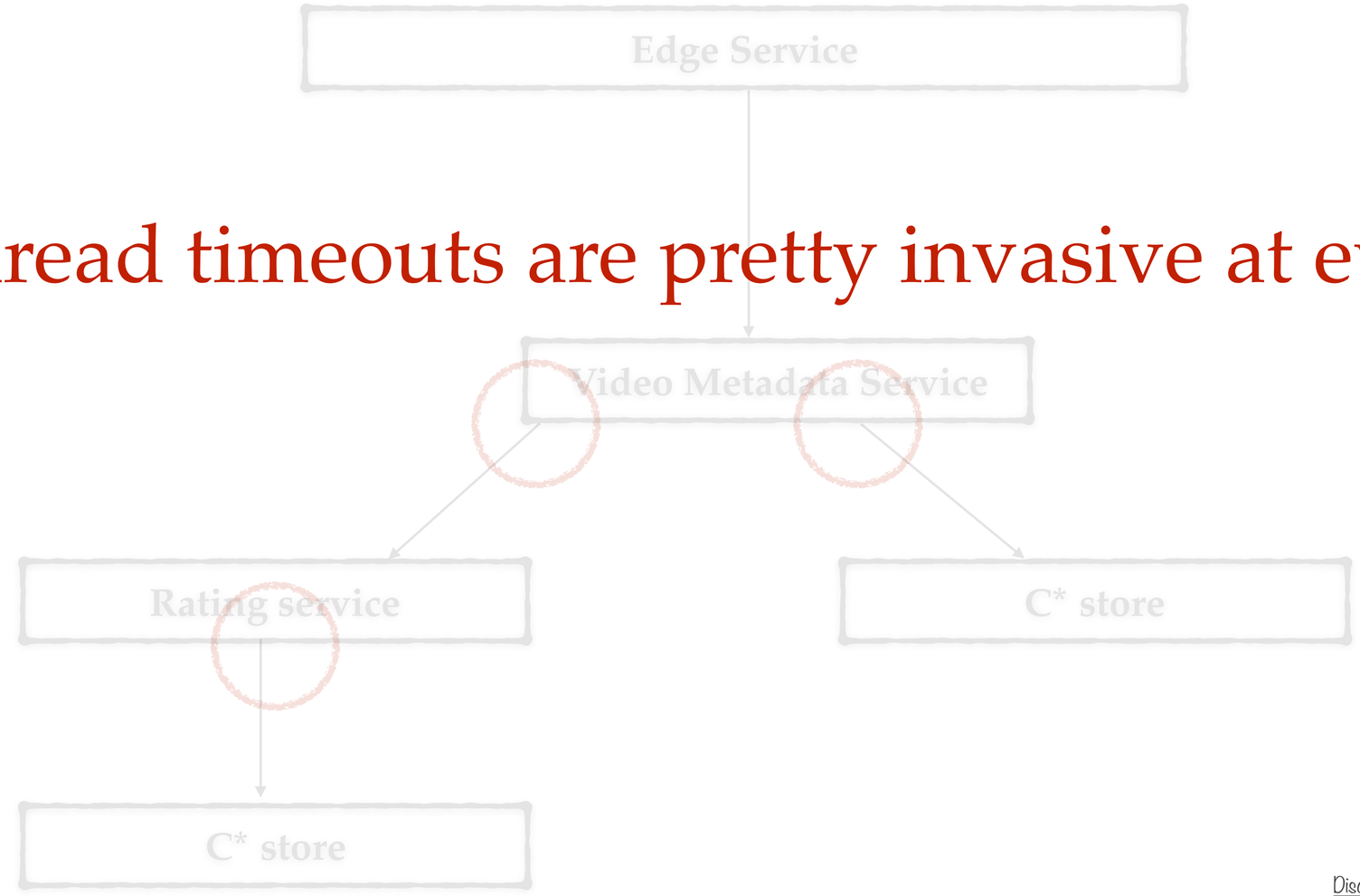
Disclaimer: This is an example and not an exact representation of the processing

Business level SLA



Disclaimer: This is an example and not an exact representation of the processing

Business level SLA



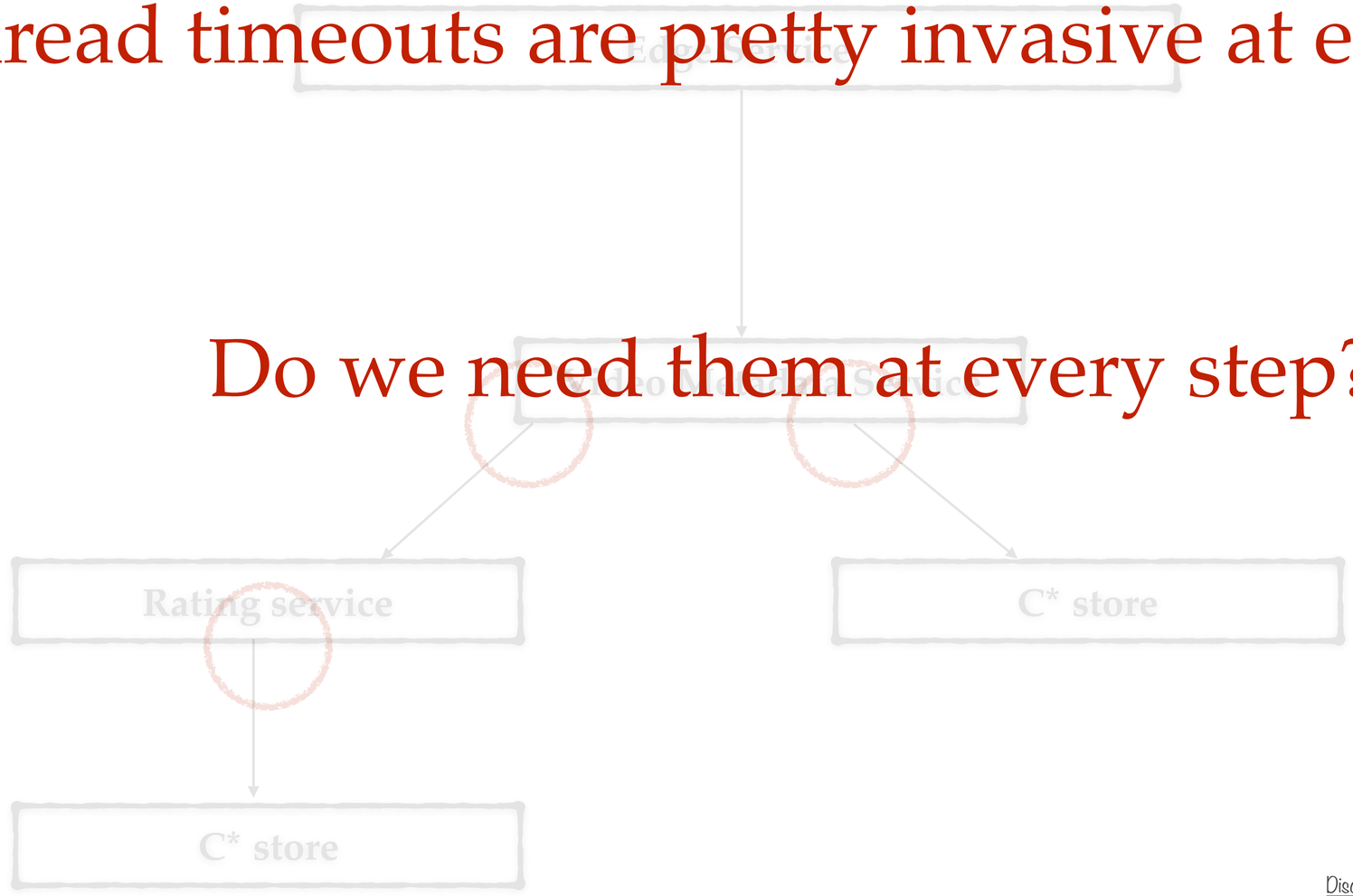
Thread timeouts are pretty invasive at every level

Disclaimer: This is an example and not an exact representation of the processing

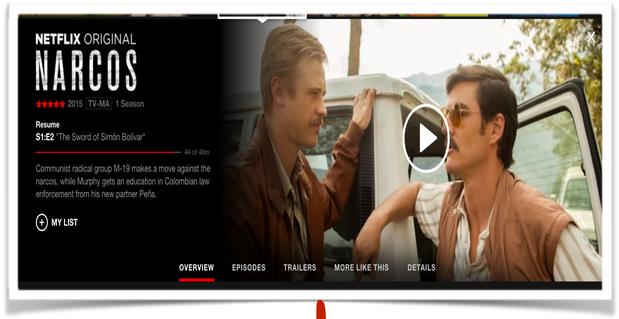
Business level SLA

Thread timeouts are pretty invasive at every level

Do we need them at every step?



Disclaimer: This is an example and not an exact representation of the processing



/movie?id=123

Edge Service

Video Metadata Service

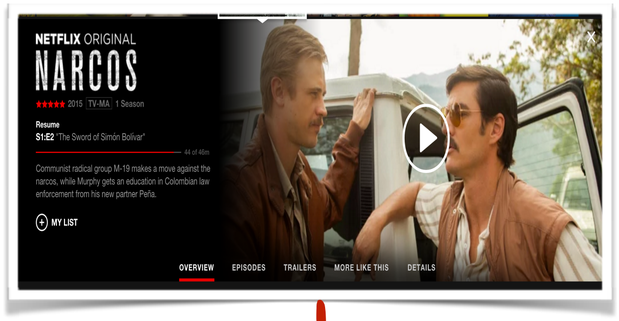
Rating service

C* store

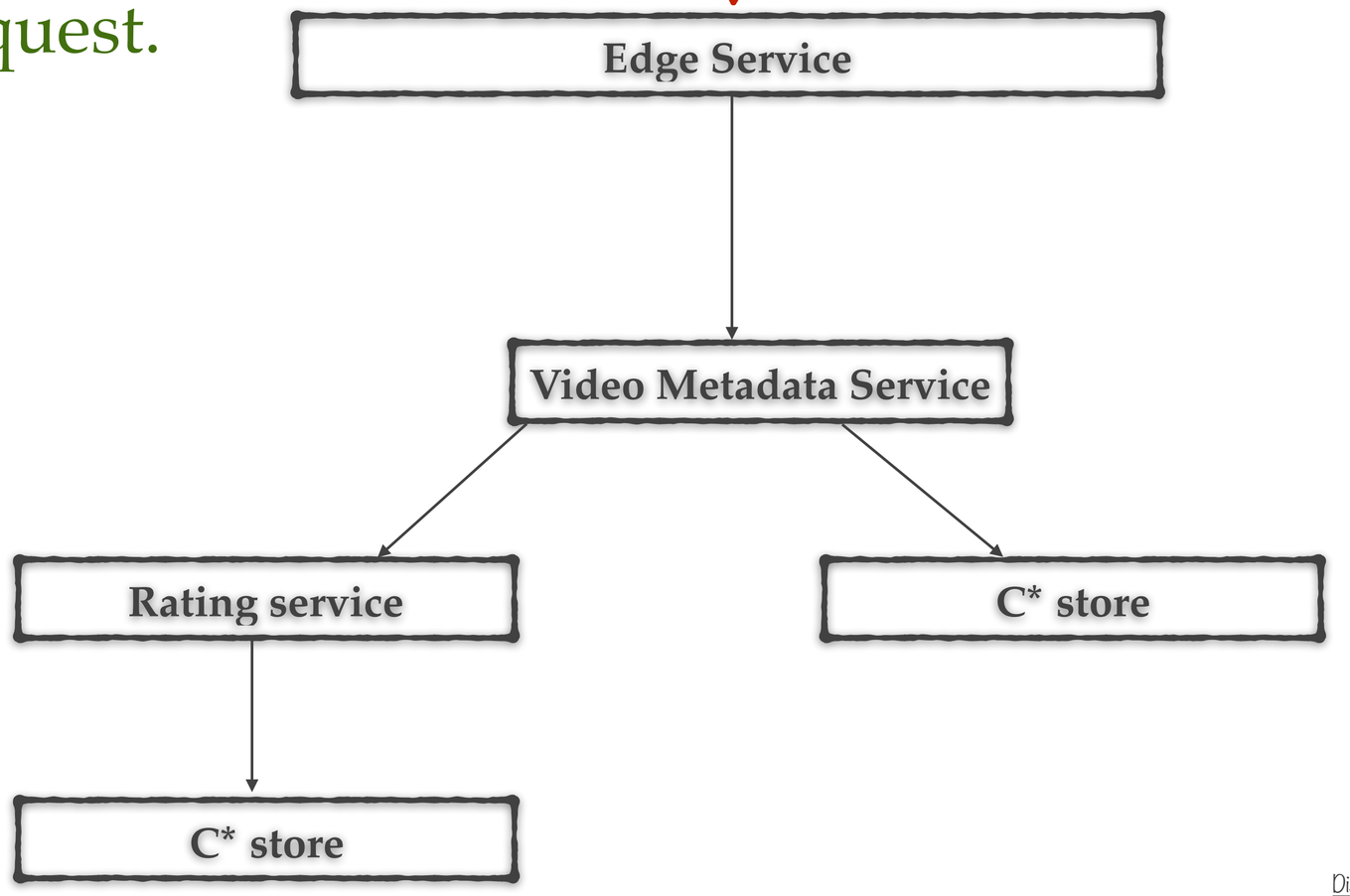
C* store

Disclaimer: This is an example and not an exact representation of the processing

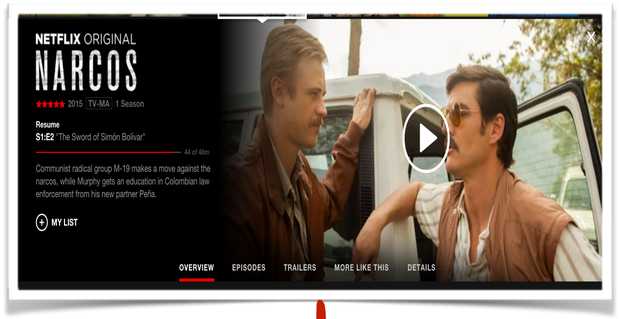
Business timeouts are
for
a client request.



/movie?id=123



Disclaimer: This is an example and not an exact representation of the processing



/movie?id=123

Edge Service

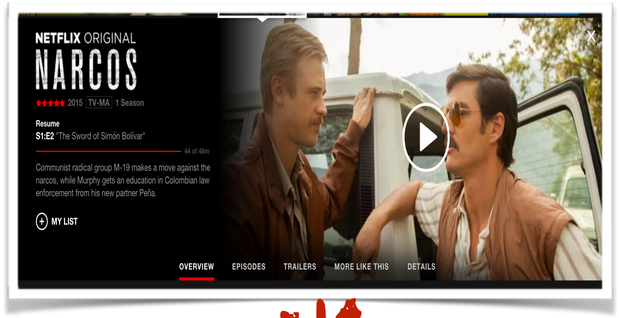
Video Metadata Service

Rating service

C* store

C* store

Disclaimer: This is an example and not an exact representation of the processing



/movie?id=123

Edge Service

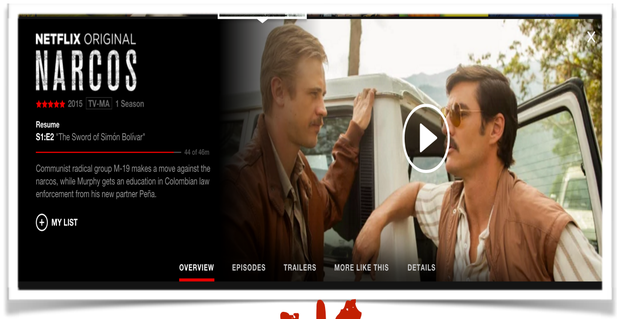
Video Metadata Service

Rating service

C* store

C* store

Disclaimer: This is an example and not an exact representation of the processing



/movie?id=123

Edge Service

Video Metadata Service

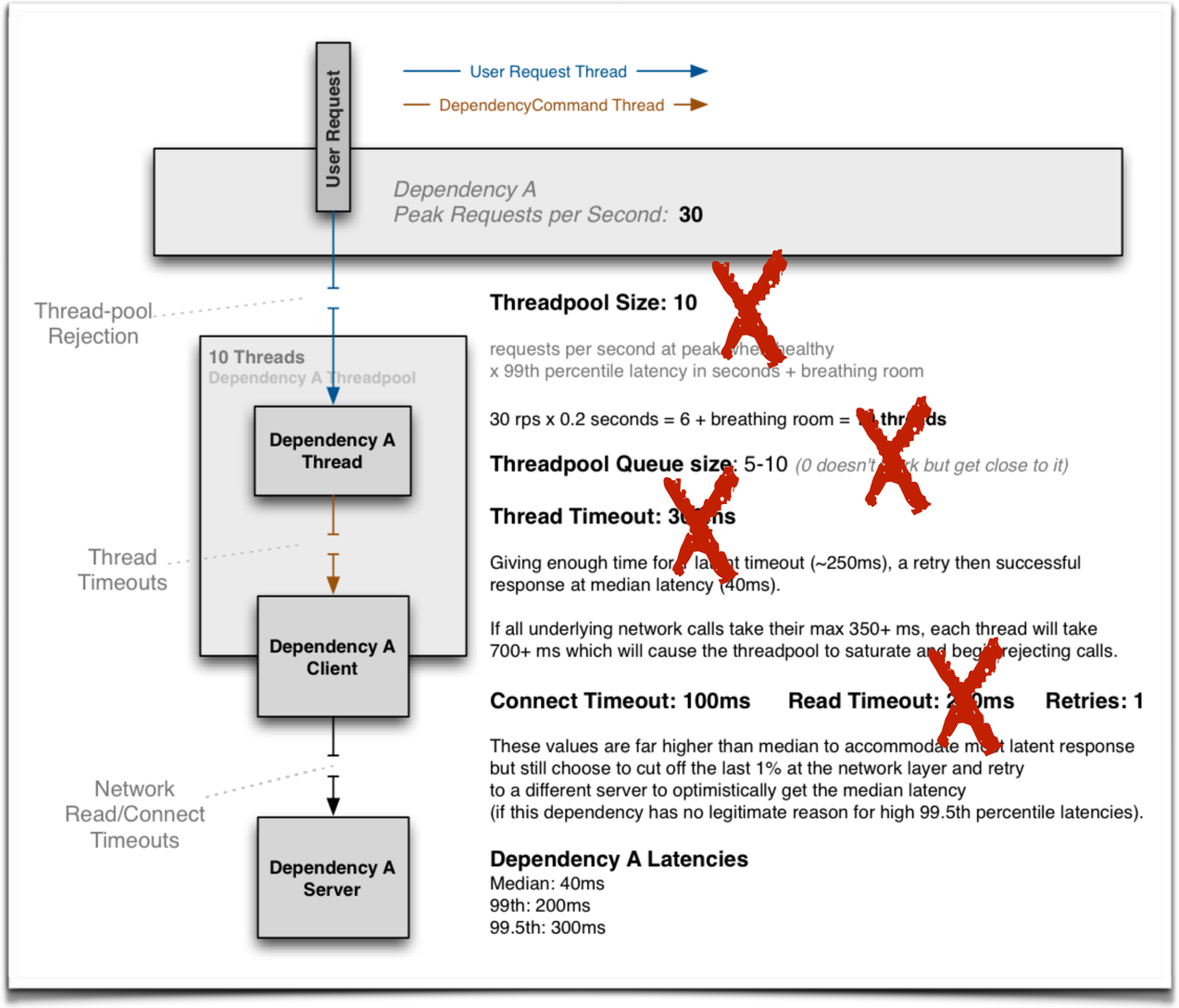
Rating service

C* store

C* store

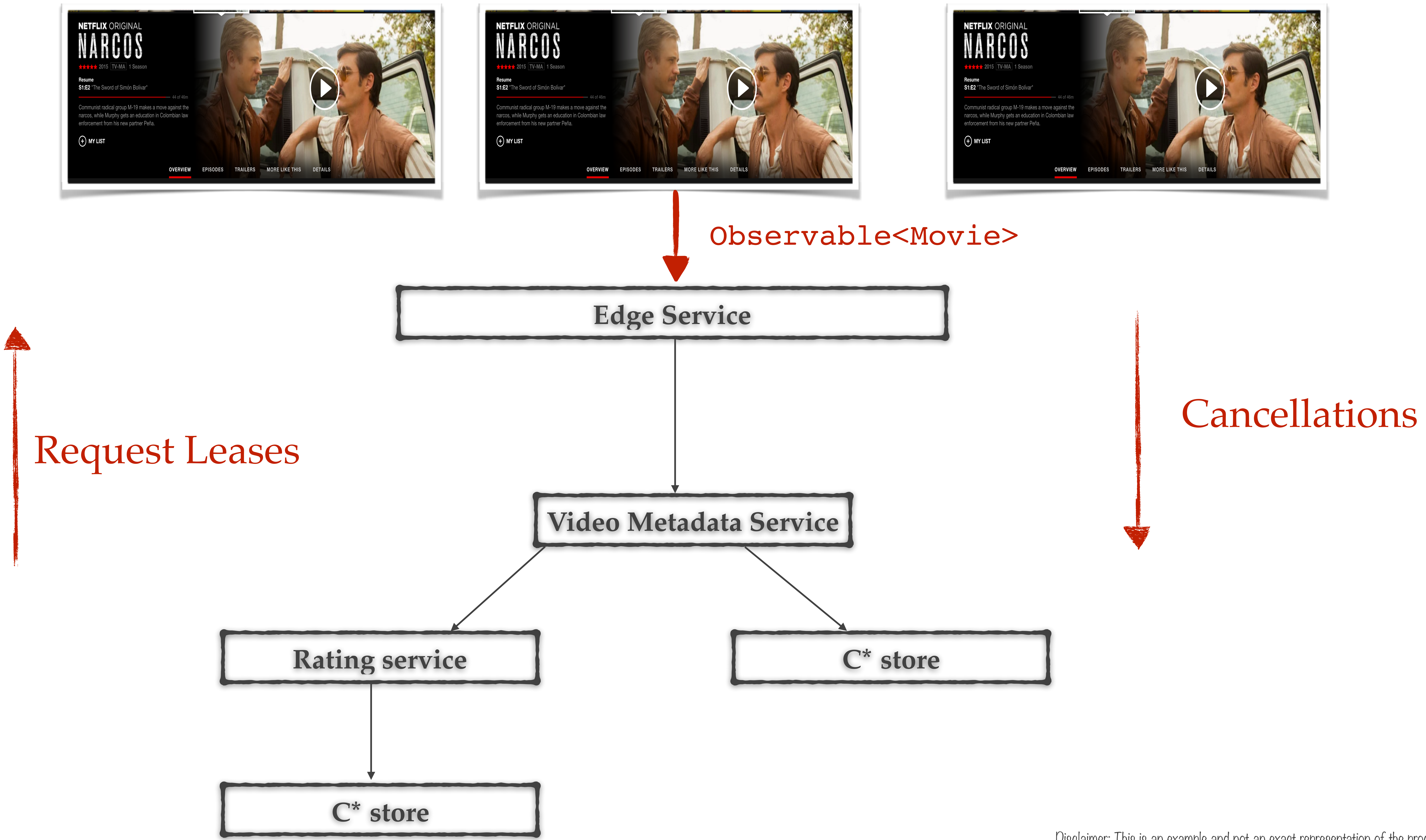
Disclaimer: This is an example and not an exact representation of the processing

Tuning parameters?

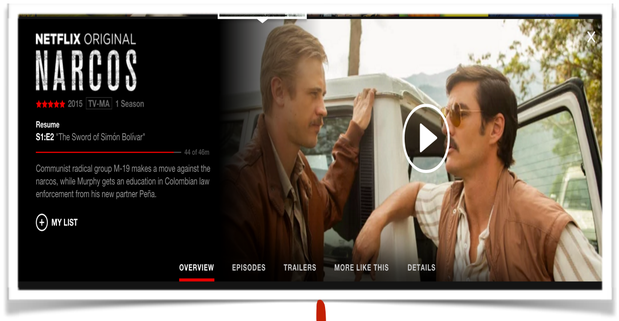


Less tuning


THE BIG
PICTURE
NEXT EXIT



Disclaimer: This is an example and not an exact representation of the processing



Observable<Movie>

Edge Service

Request Leases

Cancellations

Video Metadata Service

Rating service

C* store

C* store

Disclaimer: This is an example and not an exact representation of the processing

Graceful degradation is the ability of a computer, machine, electronic system or network to maintain limited functionality even when a large portion of it has been destroyed or rendered inoperative. The purpose of **graceful degradation** is to prevent catastrophic failure.

```
public Movie getMovie(String movieId) {
    Metadata metadata = getMovieMetadata(movieId);
    Bookmark bookmark = getBookmark(movieId, userId);
    Rating rating = getRatings(movieId);
    return new Movie(metadata, bookmark, rating);
}
```

```
public Observable<Movie> getMovie(String movieId) {
        return Observable.zip(getMovieMetadata(movieId),
                              getBookmark(movieId, userId),
                              getRatings(movieId),
                              (meta,bmark,rating)->new
Movie(meta,bmark,rating));
}
```

Resources

Asynchronous Function composition :	https://github.com/ReactiveX/RxJava
I/O :	https://github.com/ReactiveX/RxNetty
Network Protocol :	http://reactivesocket.io/


thank you
danke
謝謝
ngiyabonga
teşekkür ederim
gracias
merci
obrigado
dziękuję
bedankt
спасибо
sukriya
kop khun krap
go raibh maith agat
mochchakkeram
terima kasih
grazie
arigatô
dakujem
takk
tapadh leat
dank je
hvala
mauruuru
감사합니다
ありがとう
мерси
рахмат
Баярлалаа
faafetai lava
kiitos
dankie
dhanyavad
nandri
nanni
enkosi
sobodi
dèkuji
mèsi
didi madloba
sagolun
najis tuke
kam sah hamnida
rahmat
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
tanemirt
rahmet
diolch
dhanyavadagalu
shukriya
mercé
trugarez
mamnun
дякую
djiere dieuf
tau
obrigada
asante
manana
хвала
tenki
murakoze
chokrane
mahalo
paldies
grazzi
matondo
misaotra
blagodaram
спасибі
vinaka
mersi
kia ora
barka
welalin
tack
spas
köszönöm
akun
dankon
aciu
xièxie
ευχαριστώ