Be Cynical

Cynical software expects bad things to happen and is never surprised when they do. Cynical software doesn't even trust itself, so it puts up internal barriers to protect itself from failures. It refuses to get too intimate with other systems, because it could get hurt.

- Michael Nygard

Jimmi Kristensen
Software Engineering Manager
Works @ TV 2 PLAY

@jimmibk

https://github.com/jimmikristensen/talk-be-cynical

GR8 Conf
EUROPE 2019

The Law of Large Systems

A large system exists in a state of continuous partial failure

- Dealing with a large number of machines and distributed software, like micro services
- Some piece is being redeployed
- Some piece is under a refresh or restart
- Some piece actually has failed

The Law of Large Systems

“Everything is working” is the anomaly

- The probability that everything is working becomes smaller and smaller as the number of components go up
- Everything is working, is the weird state

What can we do to stop the propagation and avoid amplifying those failures?

Stability

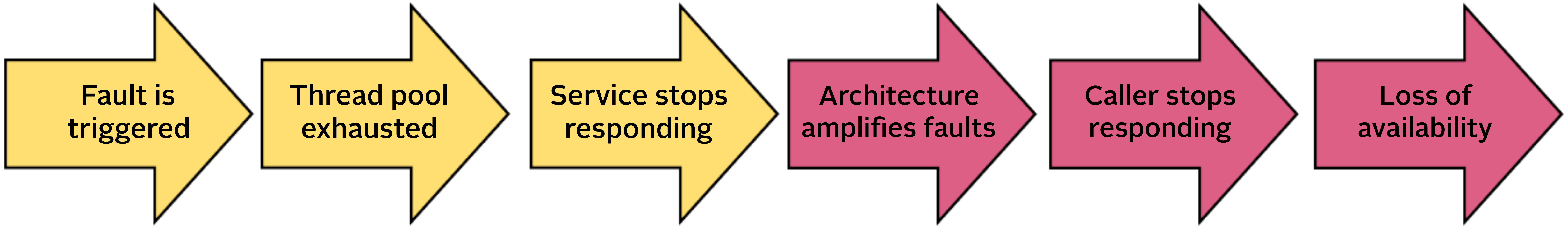
Architectural characteristics that produces availability despite faults and errors

- **Observed Availability:** the system is responding to requests within an given time
- **Fault:** an incorrect state is introduced into the system which may or may not produce visibly incorrect behavior
- **Failure:** inability of the system to perform required functionality within specified requirements - *observable by the user*

Components failure will happen

No matter how robust we make them

- Systems are going to have faults and errors
- Faults will creep in especially around the edges of a service - integrations points
- Even if you have component-level stability, it does not guarantee system- level stability



Fault is triggered

Thread pool exhausted

Service stops responding

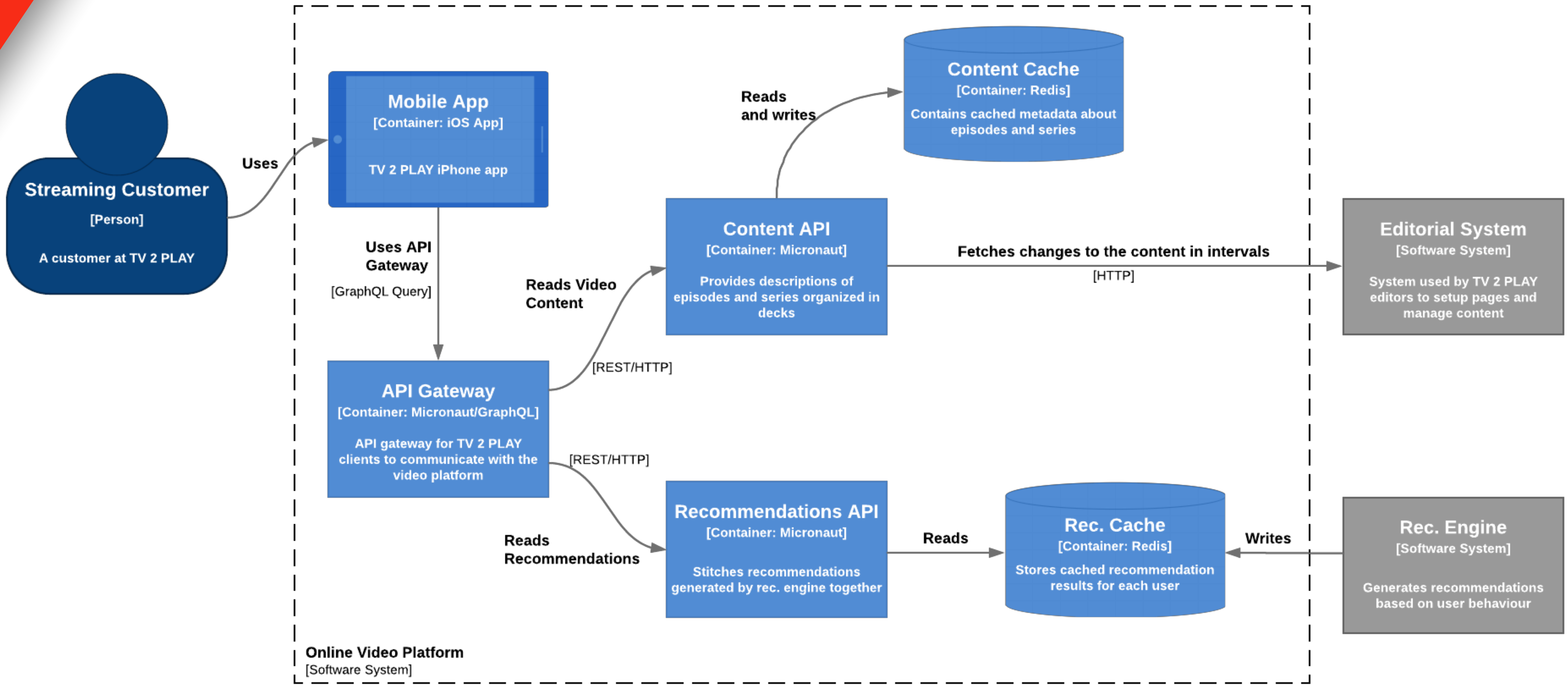
Architecture amplifies faults

Caller stops responding

Loss of availability

Let's observe this behavior through an example

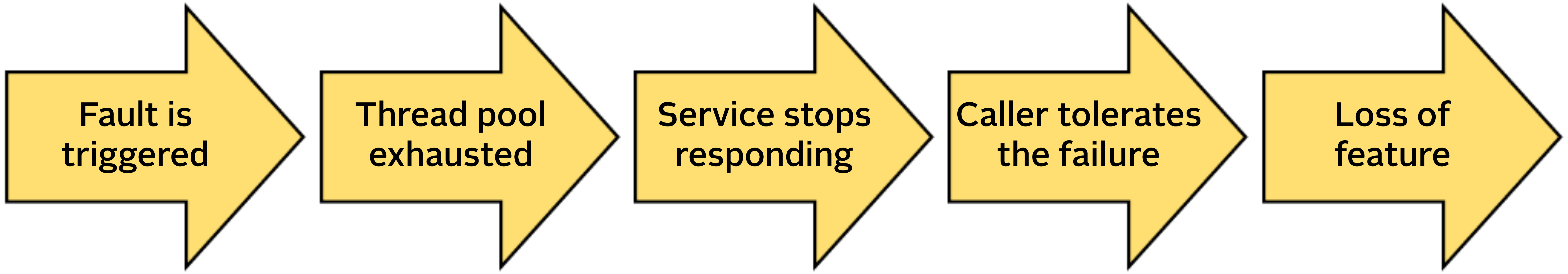
Online Video Platform Example


Streaming Customer
[Person]
A customer at TV 2 PLAY
Uses
Mobile App
[Container: iOS App]
TV 2 PLAY iPhone app
Uses API Gateway
[GraphQL Query]
Reads and writes
Content Cache
[Container: Redis]
Contains cached metadata about episodes and series
Content API
[Container: Micronaut]
Provides descriptions of episodes and series organized in decks
Fetches changes to the content in intervals
[HTTP]
Editorial System
[Software System]
System used by TV 2 PLAY editors to setup pages and manage content
Reads Video Content
[REST/HTTP]
API Gateway
[Container: Micronaut/GraphQL]
API gateway for TV 2 PLAY clients to communicate with the video platform
[REST/HTTP]
Reads Recommendations
Recommendations API
[Container: Micronaut]
Stitches recommendations generated by rec. engine together
Reads
Rec. Cache
[Container: Redis]
Stores cached recommendation results for each user
Writes
Rec. Engine
[Software System]
Generates recommendations based on user behaviour
Online Video Platform
[Software System]

Design Shock Absorbers

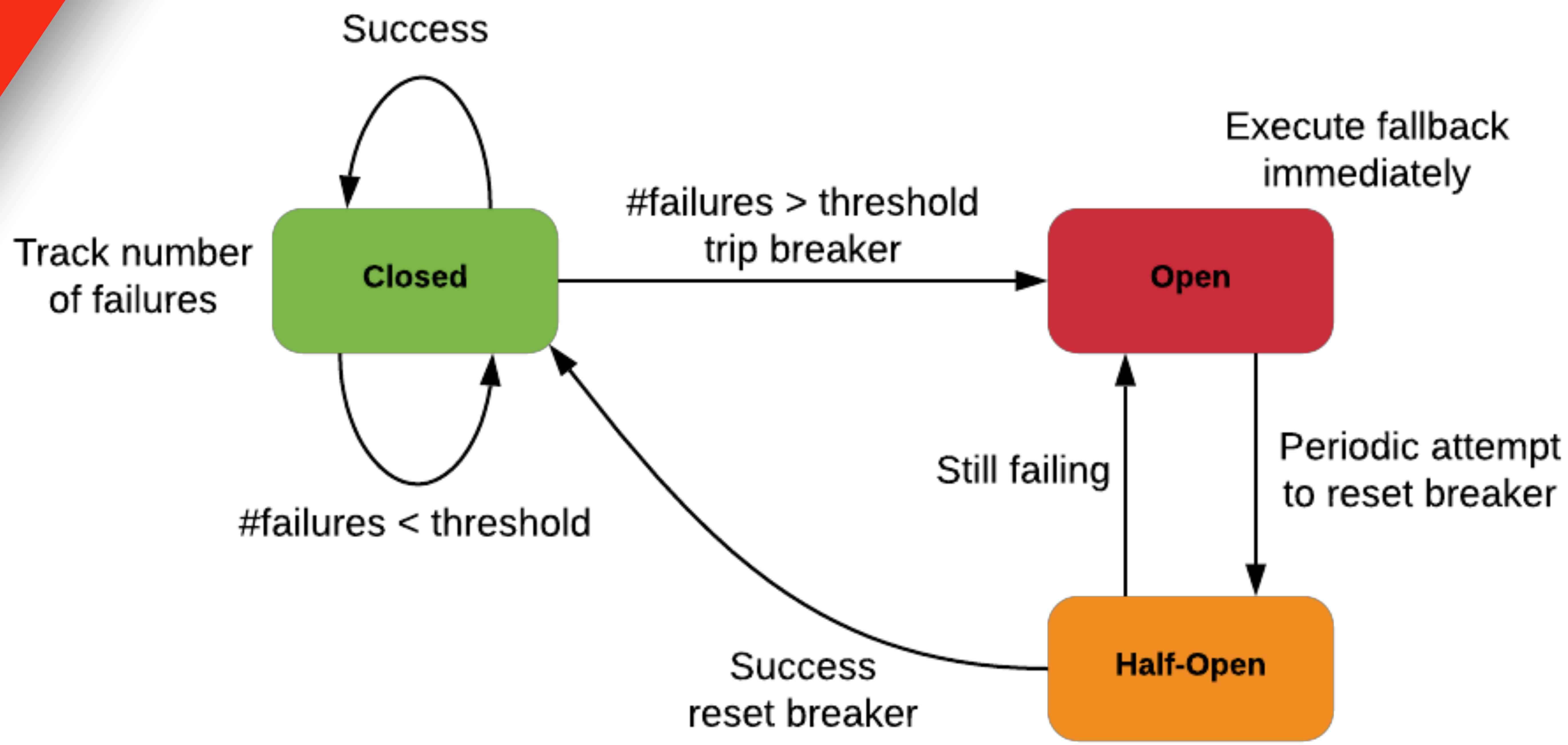
- **Fail fast**
  - Inform the caller as fast as possible, that you are unable to process the workload
- **Bulkheads**
  - Split capacity such that if one is overloaded, the other can continue
- **Circuit breakers**
  - More on that later
- **Load shedding**
  - Only allow a defined number of active connections and start to shed load - similar to fail fast
- **Caches**

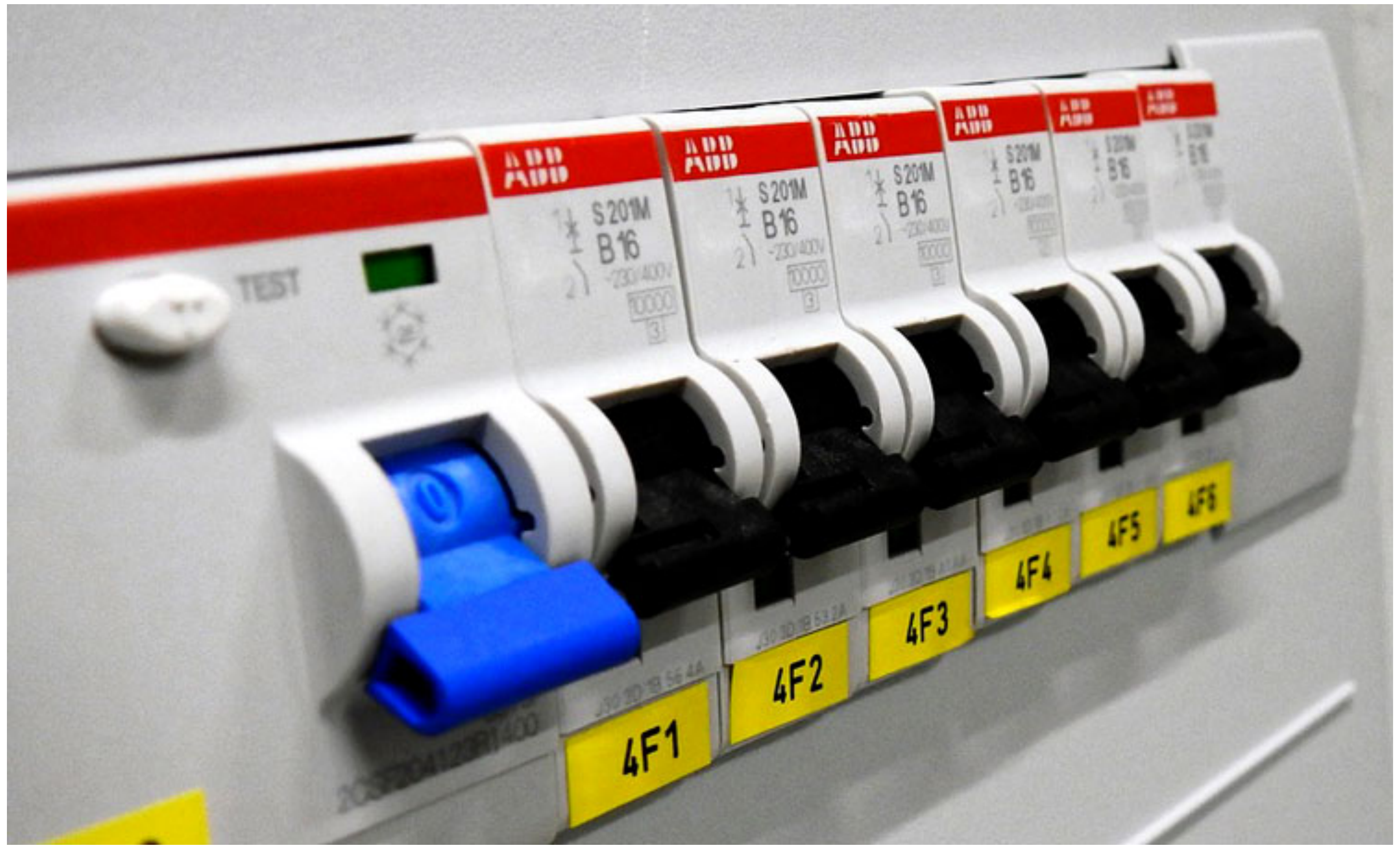
Stop the Propagation



- The fault still happens and it still affects the system, we can’t prevent that completely
- Instead of loss of availability, we have loss of feature
- The service is in a degraded state, but still serves requests

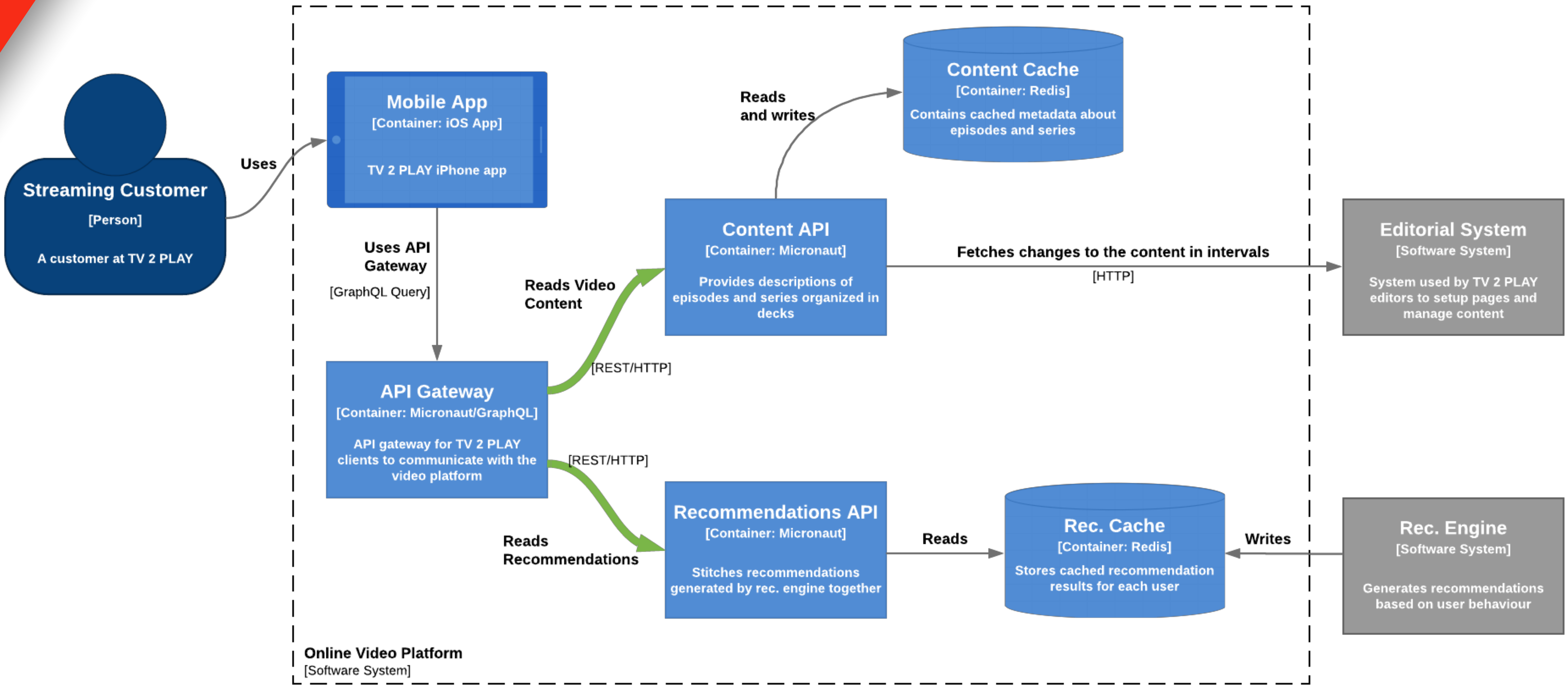
Circuit Breakers





Let's observe another example this time with circuit breakers

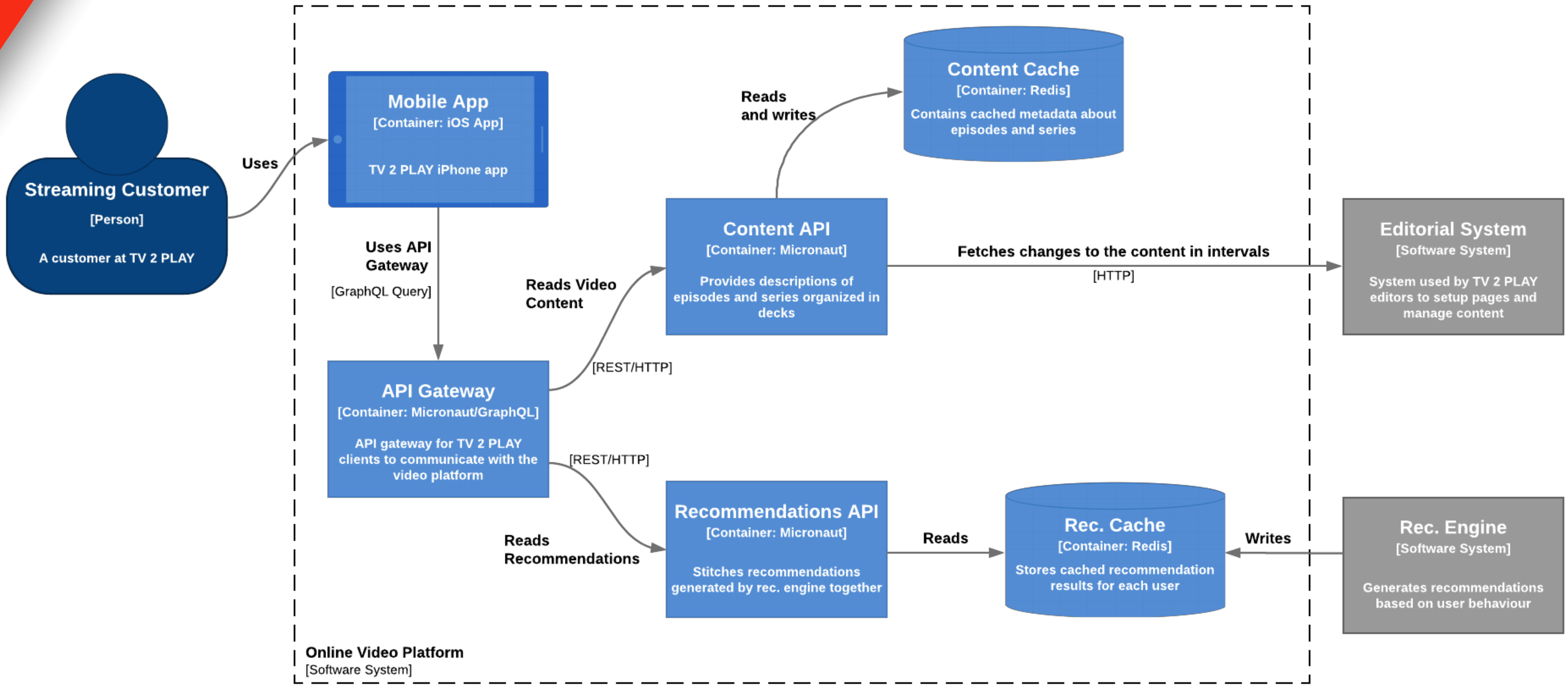
Online Video Platform Example


Streaming Customer
[Person]
A customer at TV 2 PLAY
Uses
Mobile App
[Container: iOS App]
TV 2 PLAY iPhone app
Uses API Gateway
[GraphQL Query]
API Gateway
[Container: Micronaut/GraphQL]
API gateway for TV 2 PLAY clients to communicate with the video platform
Reads Video Content
[REST/HTTP]
Content API
[Container: Micronaut]
Provides descriptions of episodes and series organized in decks
Reads and writes
Content Cache
[Container: Redis]
Contains cached metadata about episodes and series
Fetches changes to the content in intervals
[HTTP]
Editorial System
[Software System]
System used by TV 2 PLAY editors to setup pages and manage content
[REST/HTTP]
Reads Recommendations
Recommendations API
[Container: Micronaut]
Stitches recommendations generated by rec. engine together
Reads
Rec. Cache
[Container: Redis]
Stores cached recommendation results for each user
Writes
Rec. Engine
[Software System]
Generates recommendations based on user behaviour
Online Video Platform
[Software System]

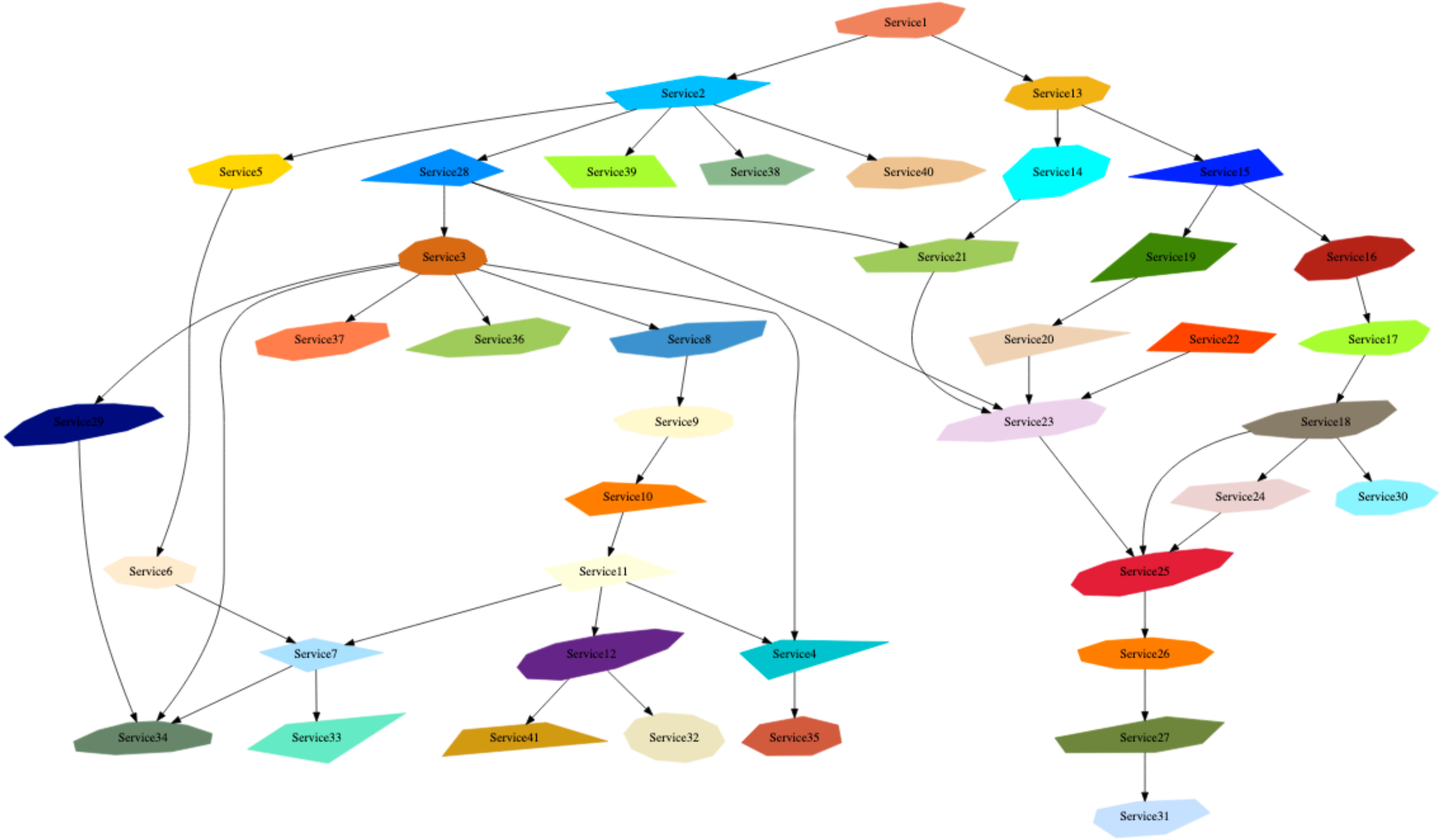
How do I test if my service is being cynical?

- I could set up Quality Attribute Scenarios for Availability and test them
- I could do load tests
- I could do manual tests
- I want something that can run automatically in my CI pipeline

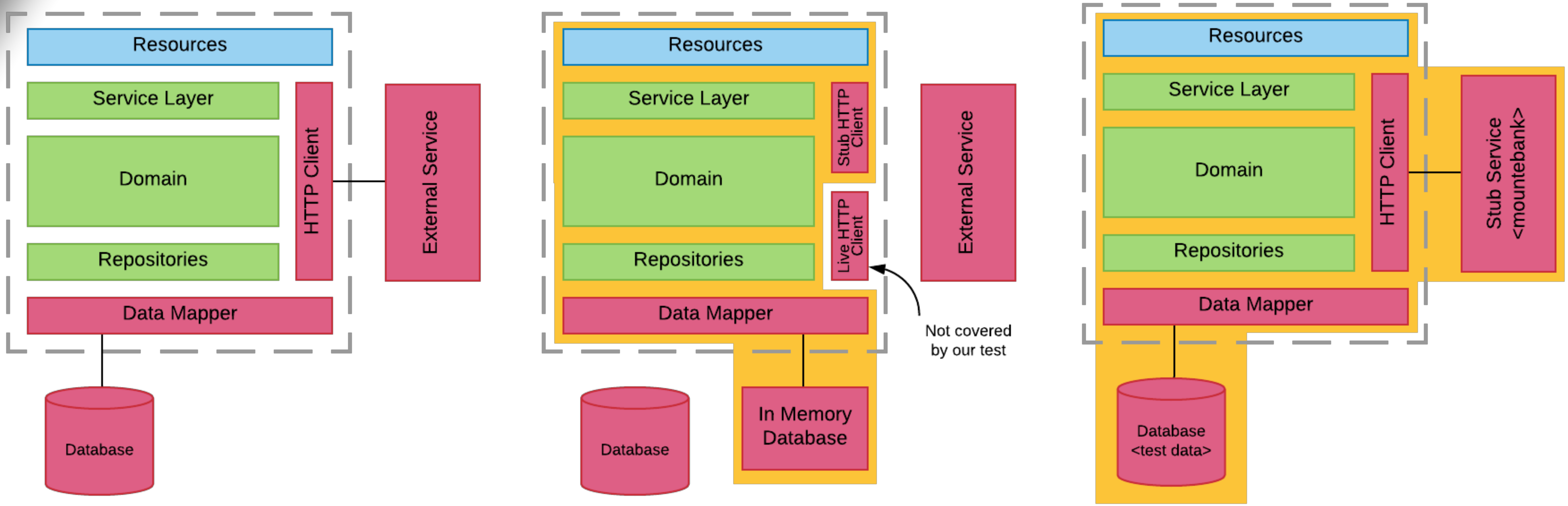
What about depended-on components like downstream services?


Streaming Customer
[Person]
A customer at TV 2 PLAY
Uses
Mobile App
[Container: iOS App]
TV 2 PLAY iPhone app
Uses API Gateway
[GraphQL Query]
API Gateway
[Container: Micronaut/GraphQL]
API gateway for TV 2 PLAY clients to communicate with the video platform
Reads Video Content
[REST/HTTP]
Content API
[Container: Micronaut]
Provides descriptions of episodes and series organized in decks
Reads and writes
Content Cache
[Container: Redis]
Contains cached metadata about episodes and series
Fetches changes to the content in intervals
[HTTP]
Editorial System
[Software System]
System used by TV 2 PLAY editors to setup pages and manage content
[REST/HTTP]
Reads Recommendations
Recommendations API
[Container: Micronaut]
Stitches recommendations generated by rec. engine together
Reads
Rec. Cache
[Container: Redis]
Stores cached recommendation results for each user
Writes
Rec. Engine
[Software System]
Generates recommendations based on user behaviour
Online Video Platform
[Software System]

What about depended-on components like downstream services?



Let's explore some possibilities



Resources

Service Layer

Domain

Repositories

HTTP Client

Data Mapper

External Service

Database

Resources

Service Layer

Domain

Repositories

Stub HTTP Client

Live HTTP Client

Data Mapper

External Service

In Memory Database

Database

Not covered by our test

Resources

Service Layer

Domain

Repositories

HTTP Client

Data Mapper

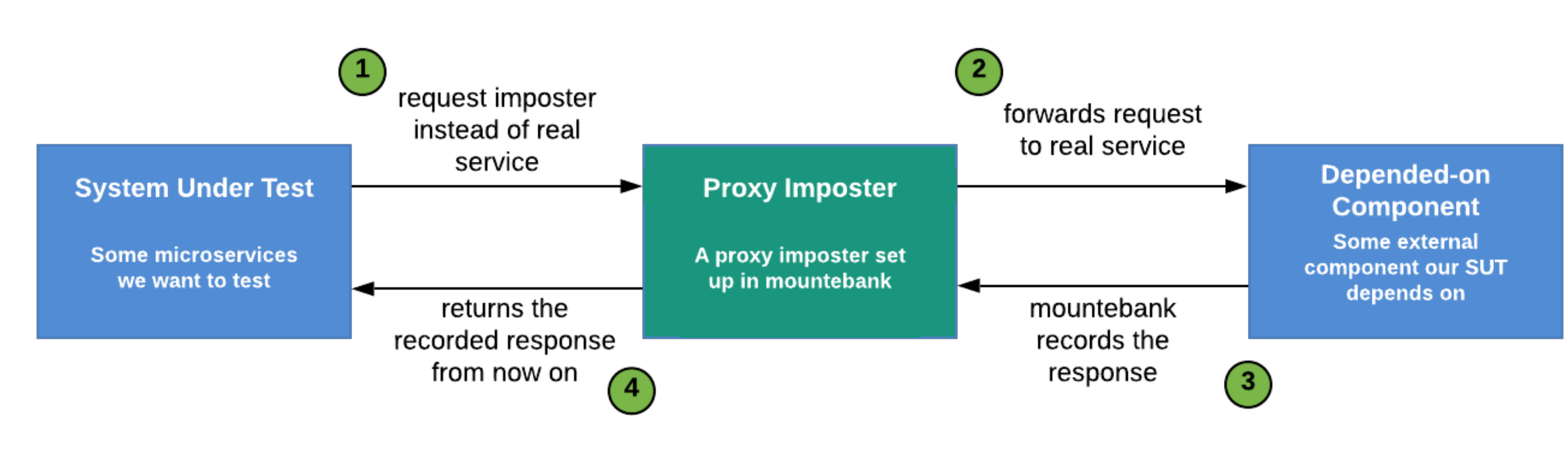
Stub Service <mountebank>

Database <test data>

Mountebank

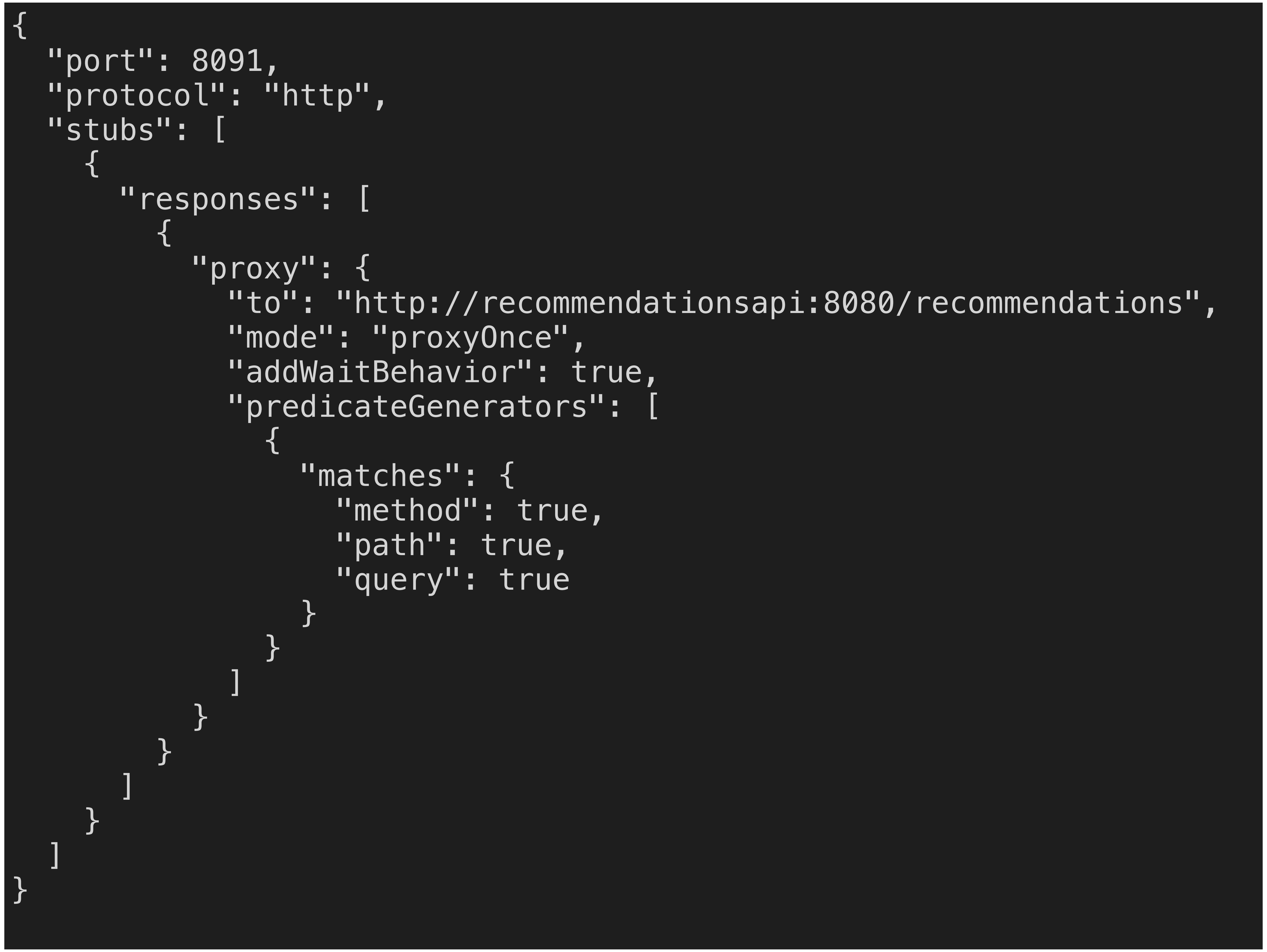
mountebank is the first open source tool to provide cross-platform, multi-protocol test doubles over the wire.

http://www.mbtest.org/



1
request imposter instead of real service
2
forwards request to real service
System Under Test
Some microservices we want to test
Proxy Imposter
A proxy imposter set up in mountebank
Depended-on Component
Some external component our SUT depends on
returns the recorded response from now on
4
mountebank records the response
3

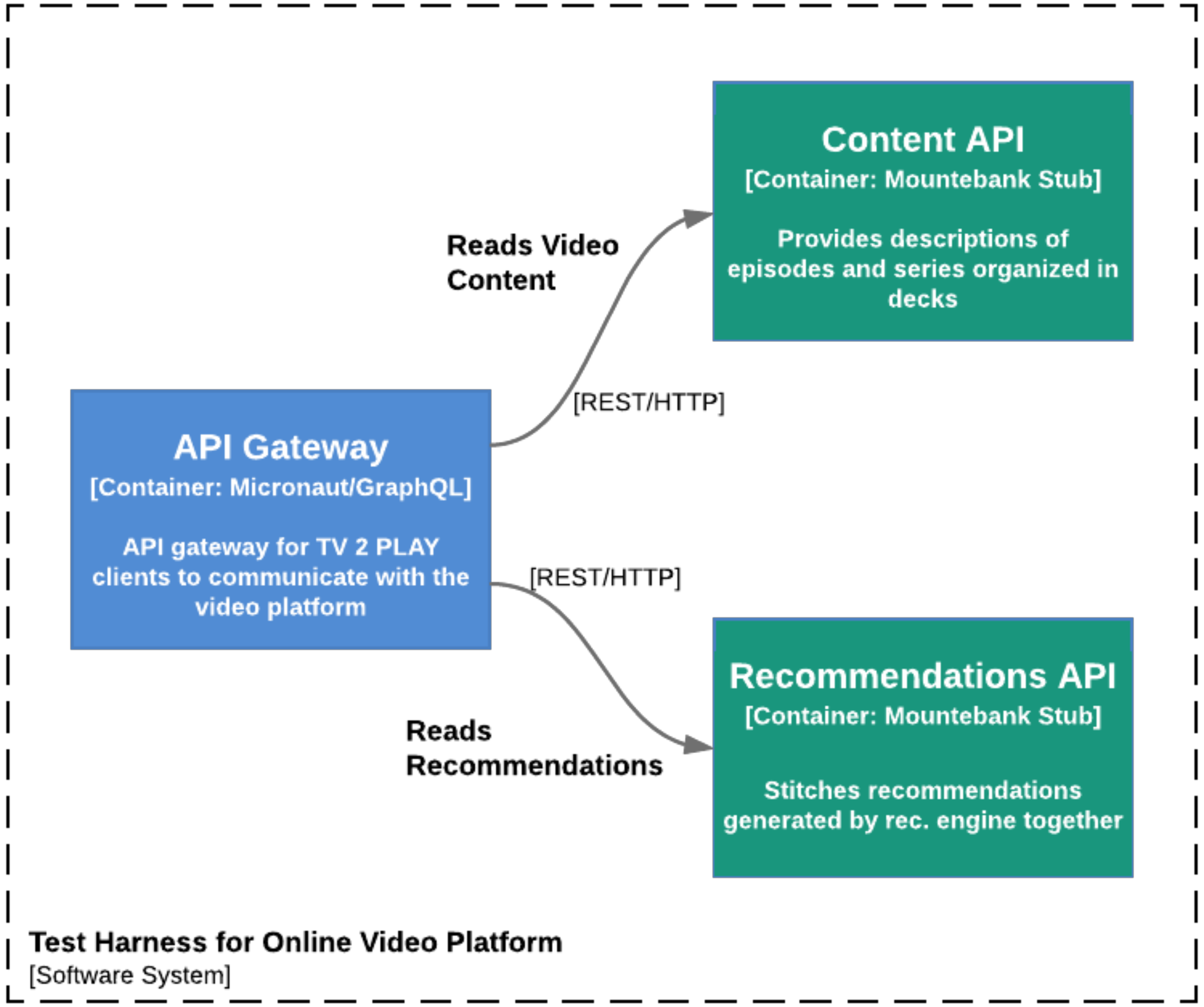
Mountebank

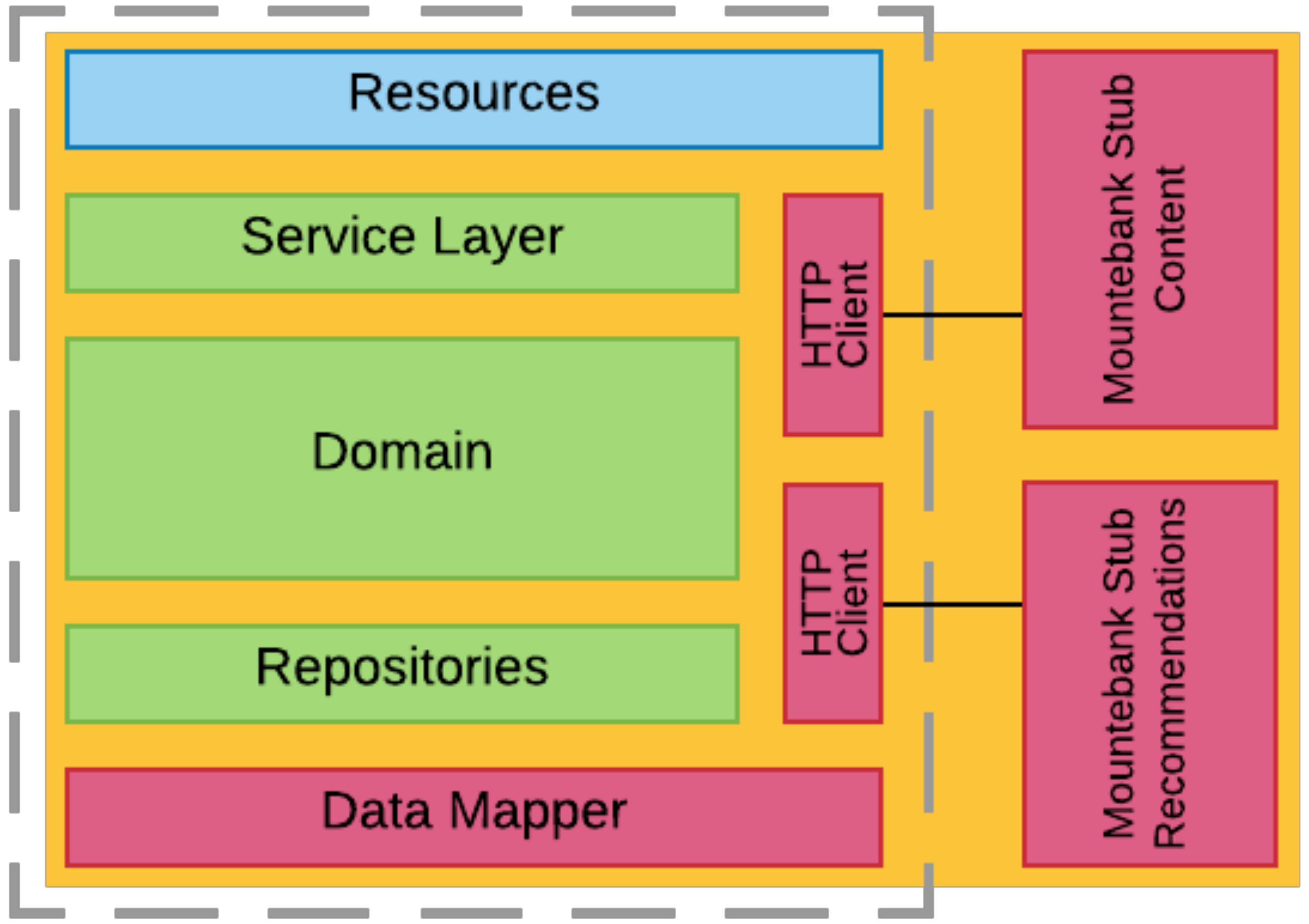


```
{
  "port": 8091,
  "protocol": "http",
  "stubs": [
    {
      "responses": [
        {
          "proxy": {
            "to": "http://recommendationsapi:8080/recommendations",
            "mode": "proxyOnce",
            "addWaitBehavior": true,
            "predicateGenerators": [
              {
                "matches": {
                  "method": true,
                  "path": true,
                  "query": true
                }
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Test Harness

We only need to stub immediate dependencies


Content API
[Container: Mountebank Stub]
Provides descriptions of episodes and series organized in decks
Reads Video Content
[REST/HTTP]
API Gateway
[Container: Micronaut/GraphQL]
API gateway for TV 2 PLAY clients to communicate with the video platform
[REST/HTTP]
Reads Recommendations
Recommendations API
[Container: Mountebank Stub]
Stitches recommendations generated by rec. engine together
Test Harness for Online Video Platform
[Software System]



Let's try out the mountebank stubs of the two dependend-on services

Test Harness

By stubbing the depended-on components, the downstream services, we are able to:

- make calls over the wire and thereby test the client integration
- create automated tests using the test-framework of your choice
- automate the process of starting the mountebank stubs
- run the entire test harness in a CI pipeline

Be Cynical

Make your microservice cynical against dependencies, so they don't get hurt

The Pragmatic Programmers

Release It!
Second Edition

Design and Deploy
Production-Ready Software

Michael T. Nygard
Edited by Katharine Dvorak

Excellent book by Michael Nygard

a must read for any developer or architect

- Think destructive, especially around integration points
- At every integration point, think "What can go wrong?"
- Use exception/error handlers
- Implement shock absorbers to prevent cracks in the system to propagate
- Mask faults in the system to prevent them becoming failures