Microservices

AGENDA

- Netflix – background and evolution
- Monolithic Apps
  - Characteristics
- What are Microservices?
- Microservices
  - Why?
  - Challenges
  - Best practices
  - Tools of the trade
- InterProcess Communication
- Takeaways

Netflix - Evolution

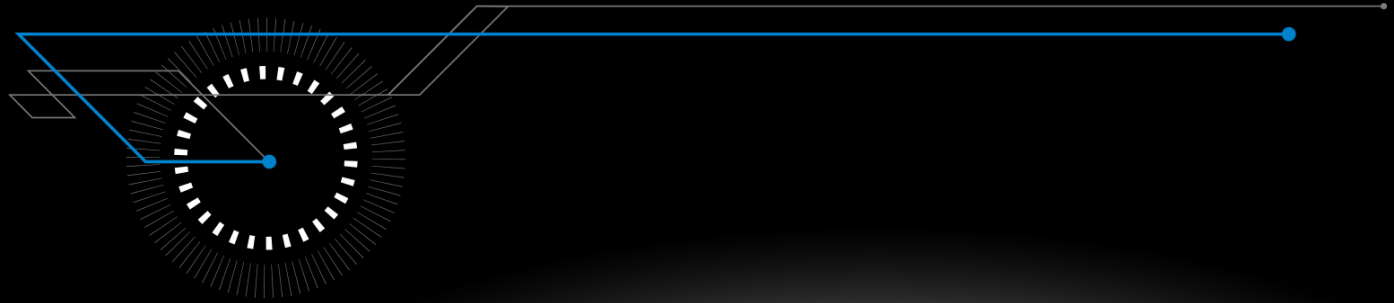
- Old DataCenter (2008)
- Everything in one WebApp (.war)

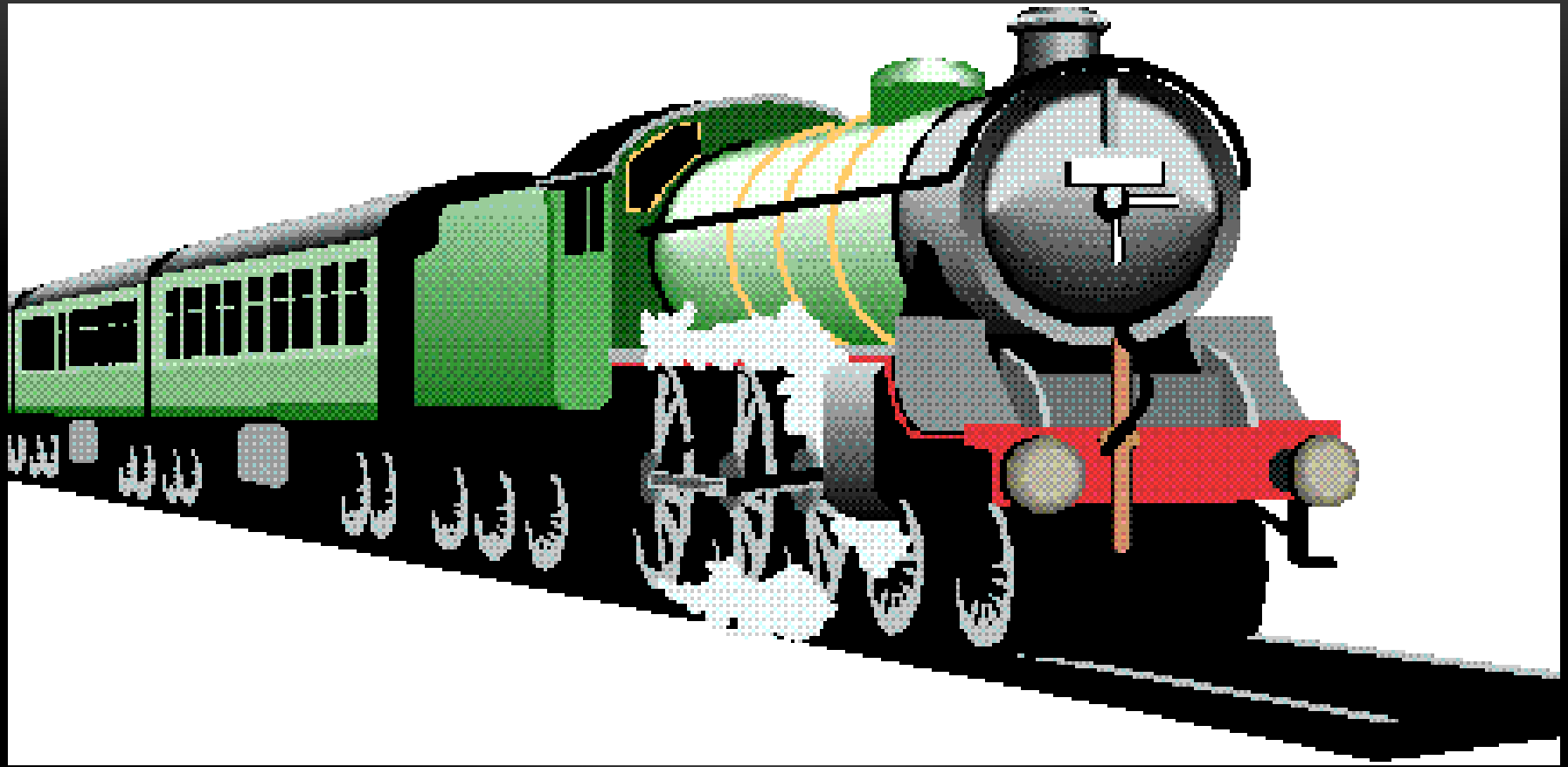
- AWS Cloud (~2010)
- 100s of Fine Grained Services

Netflix Scale

- ~ 1/3 of the peak Internet traffic a day
- ~50M subscribers
- ~2 Billion Edge API Requests/Day
- >500 MicroServices
- ~30 Engineering Teams (owning many microservices)

Monolithic Apps





MONOLITHIC APP

Monolithic Architecture

Load Balancer

Monolithic App

Account Component

Catalog Component

Recommendation Component

Customer Service Component

Database

Characteristics

- Large Codebase
- Many Components, no clear ownership
- Long deployment cycles

Pros

- Single codebase
  - Easy to develop/debug/deploy
  - Good IDE support
- Easy to scale horizontally (but can only scale in an “un-differentiated” manner)
- A Central Ops team can efficiently handle

Monolithic App – Evolution

- As codebase increases …
  - Tends to increase “tight coupling” between components
    - Just like the cars of a train
  - All components have to be coded in the same language

User Accounts

Customer Service

Shopping Cart

Product Catalog

Evolution of a Monolithic App

Monolithic App - Scaling

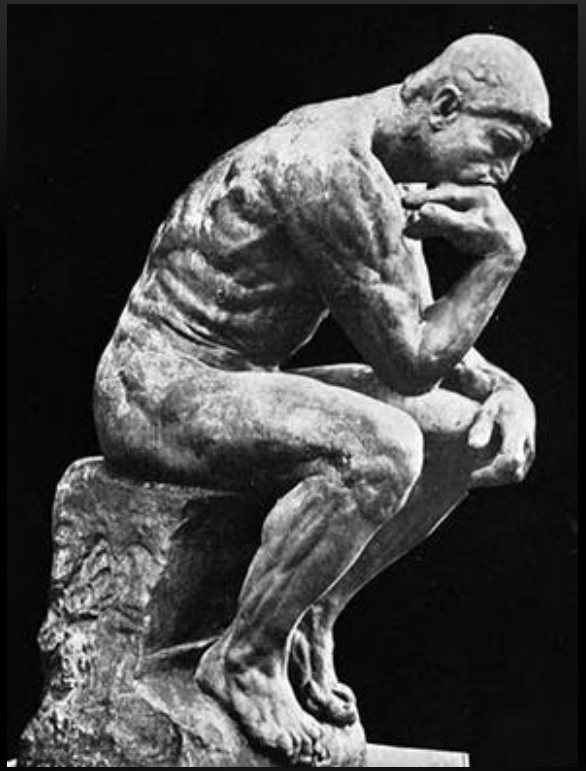
- Scaling is “undifferentiated”
  - Cant scale “Product Catalog” differently from “Customer Service”

AVAILABILITY

Availability

- A single missing “;” brought down the Netflix website for many hours (~2008)

MONOLITHIC APPS – FAILURE & AVAILABILITY



MicroServices

You Think??

TIPPING POINT

Organizational Growth

&

Disverse Functionality

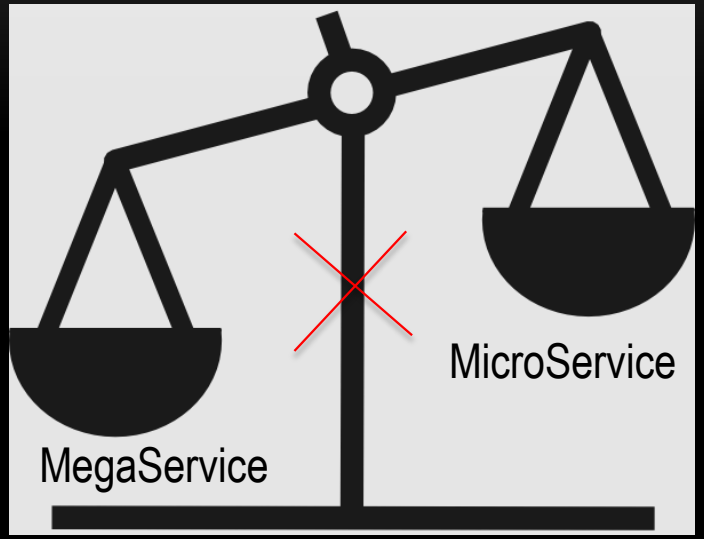
&

Bottleneck in Monolithic stack

What are MicroServices?

NOT ABOUT …

- Team size
- Lines of code
- Number of API/EndPoints



CHARACTERISTICS

- Many smaller (fine grained), clearly scoped services
  - Single Responsibility Principle
  - Domain Driven Development
  - Bounded Context
  - Independently Managed
- Clear ownership for each service
  - Typically need/adopt the “DevOps” model

Attribution: Adrian Cockroft, Martin Fowler …

Composability– unix philosophy

- Write programs that do one thing and do it well.
- Write programs to work together

```
tr 'A-Z' 'a-z' < doc.txt | tr -cs 'a-z' '\n' | sort | uniq | comm -23 - /usr/share/dict/words
```

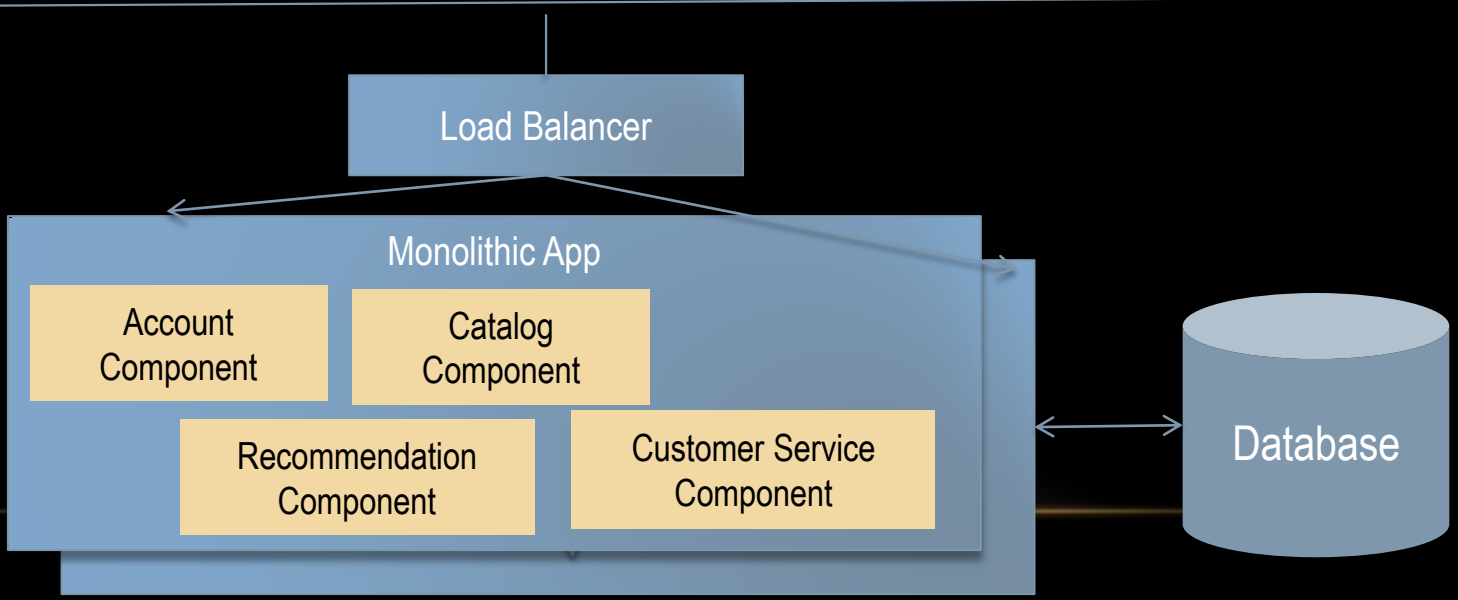
Program to print misspelt words in doc.txt

Comparing Monolithic to MicroServices

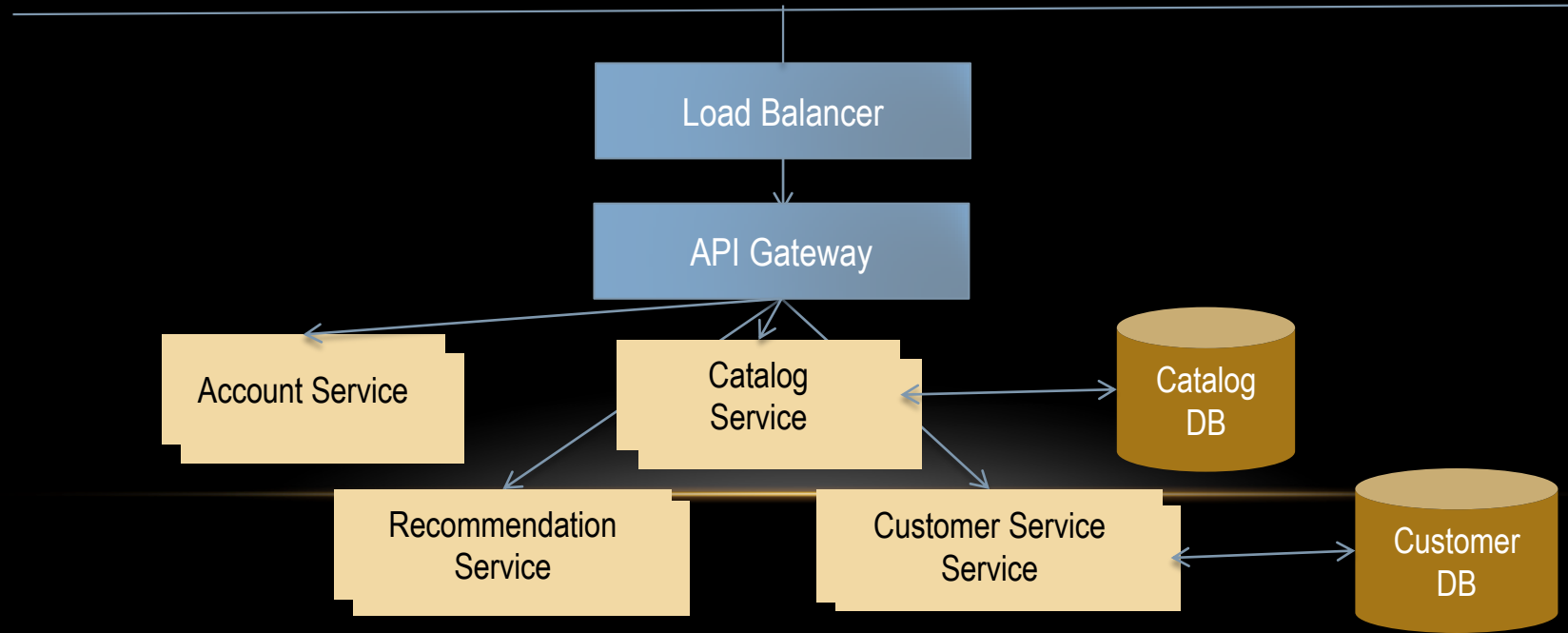
MONOLITHIC APP (VARIOUS COMPONENTS LINKED TOGETHER)

MICROSERVICES – SEPARATE SINGLE PURPOSE SERVICES

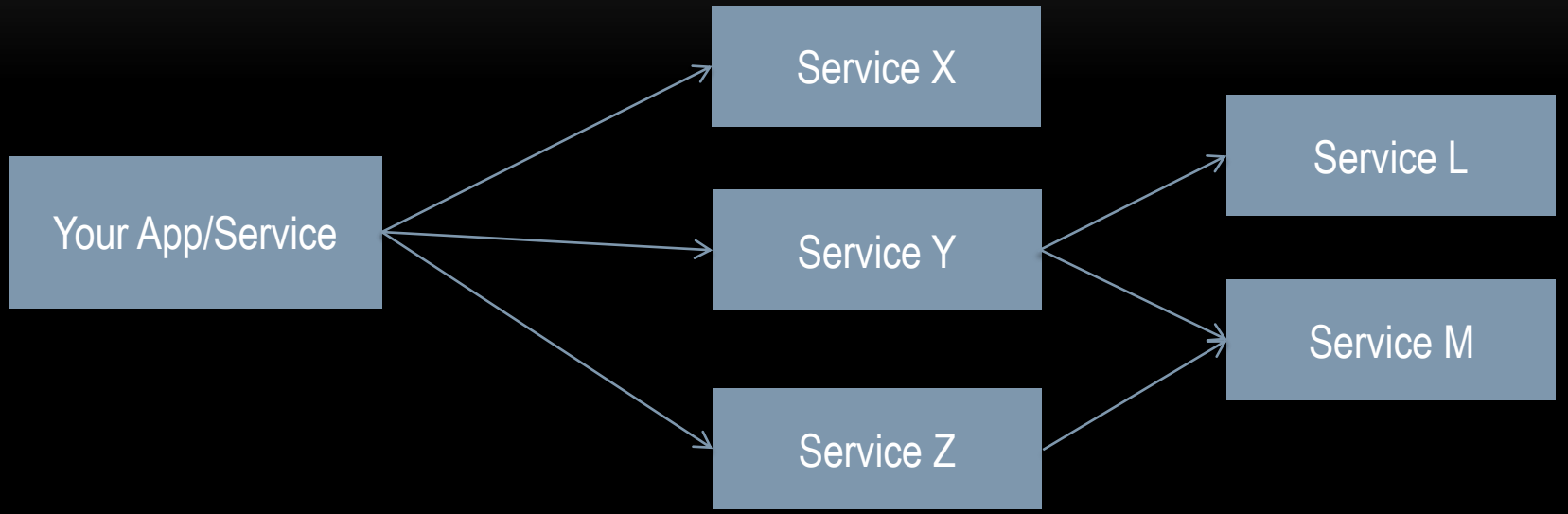
Monolithic Architecture (Revisiting)



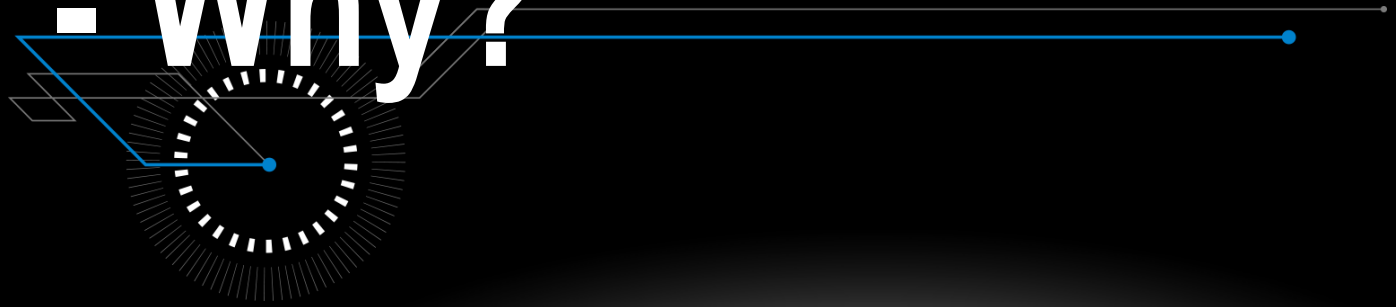
Microservices Architecture



Concept -> Service Dependency Graph



MicroServices
- Why?



WHY?

- Faster and simpler deployments and rollbacks
  - Independent Speed of Delivery (by different teams)
- Right framework/tool/language for each domain
  - Recommendation component using Python?, Catalog Service in Java ..
- Greater Resiliency
  - Fault Isolation
- Better Availability
  - If architected right ☺

MicroServices - Challenges

CHALLENGES

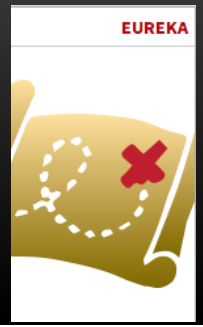
Can lead to chaos if not designed right …

OVERALL COMPLEXITY

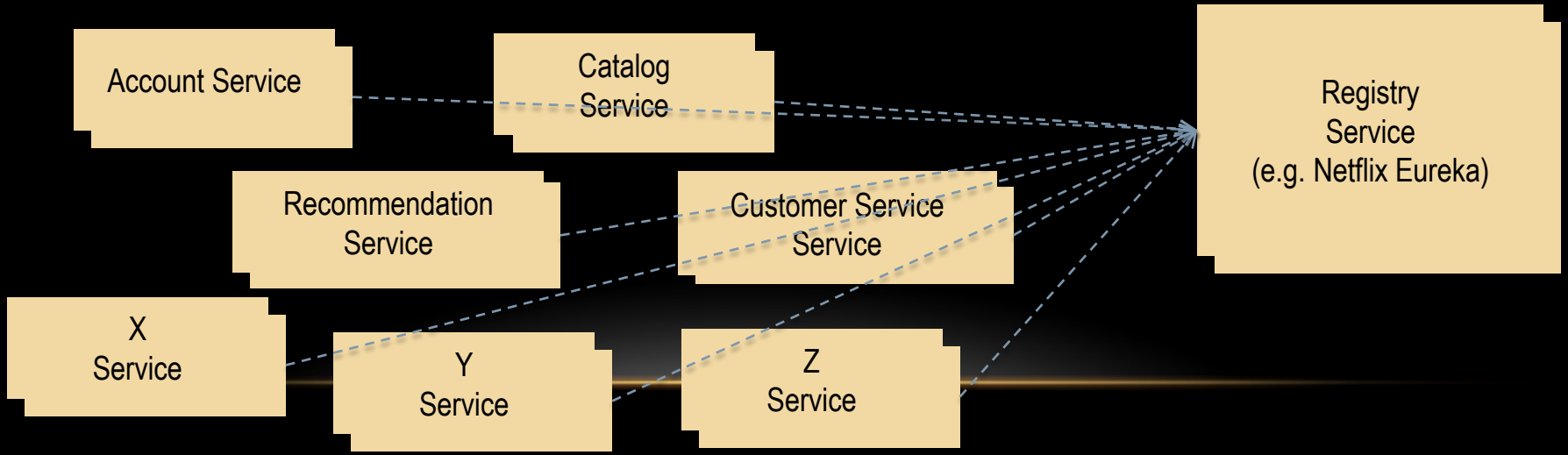
- Distributed Systems are inherently Complex
  - N/W Latency, Fault Tolerance, Retry storms ..
- Operational Overhead
  - TIP: Embrace DevOps Model

SERVICE DISCOVERY

- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)



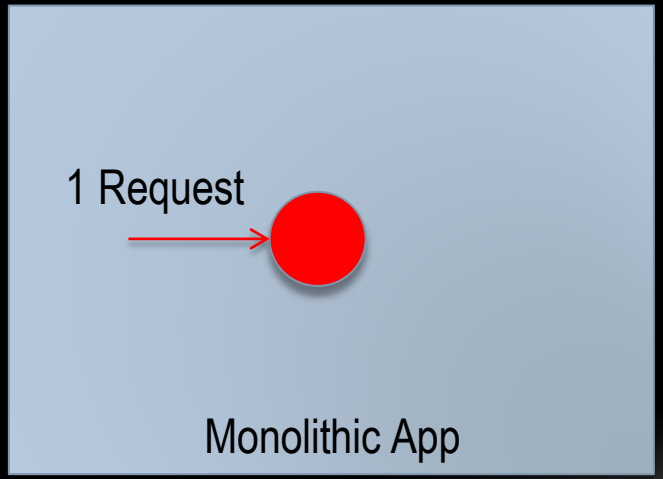
NETFLIX | OSS

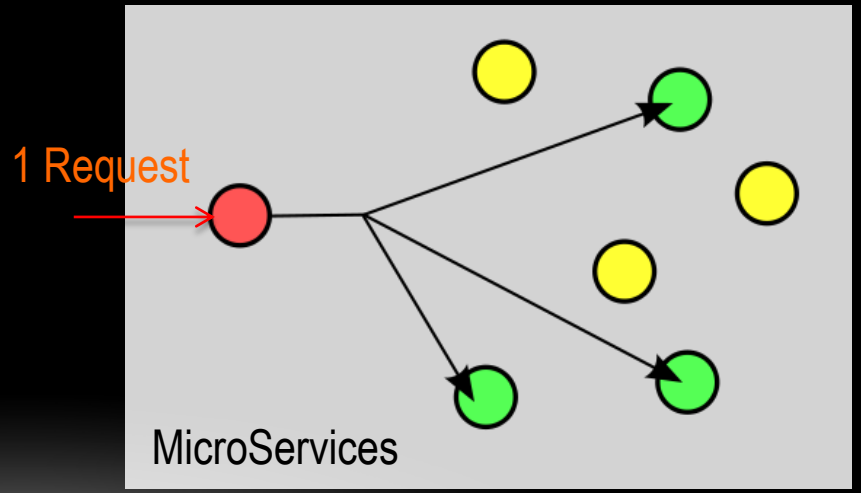


CHATTINESS (AND FAN OUT)

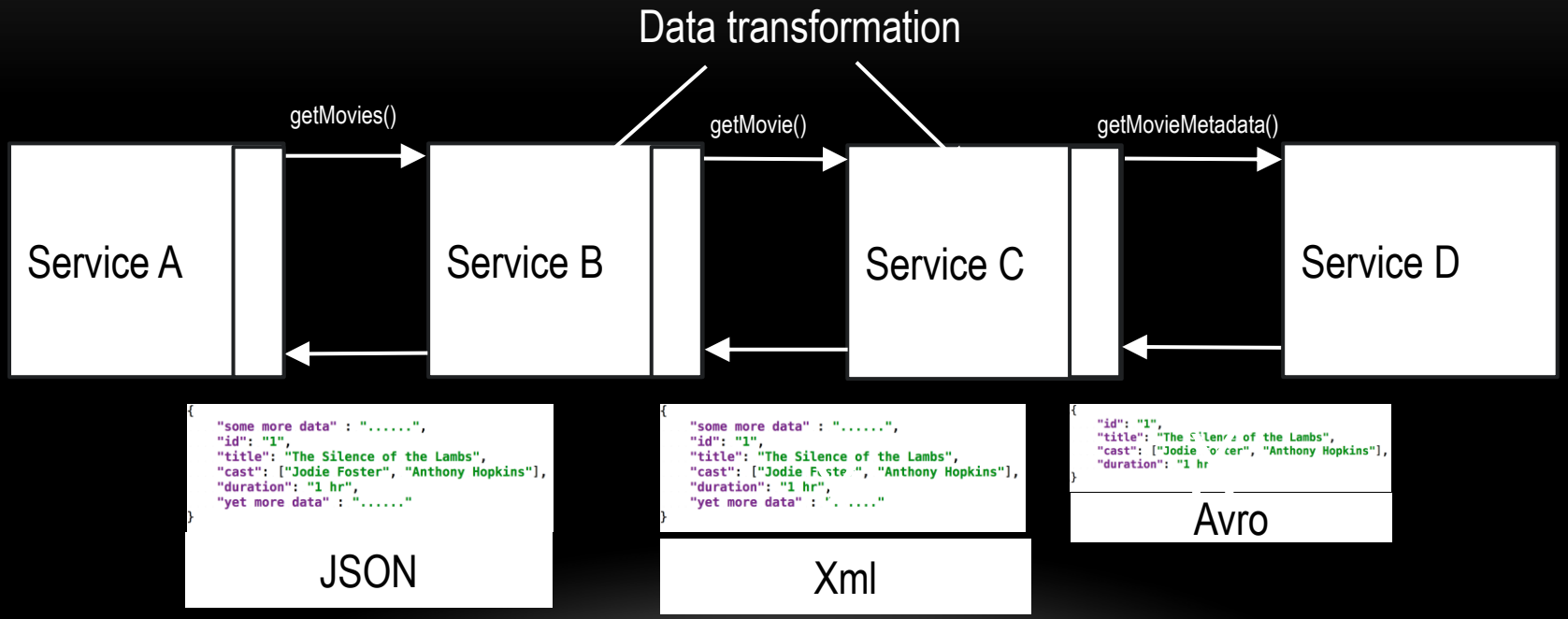
~2 Billion Requests per day on Edge Service

Results in ~20 Billion Fan out requests in ~100 MicroServices





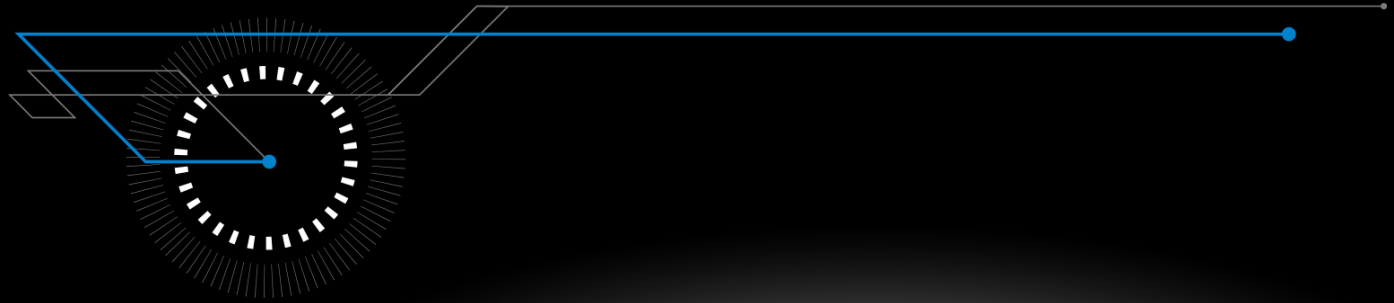
DATA SERIALIZATION OVERHEAD



CHALLENGES - SUMMARY

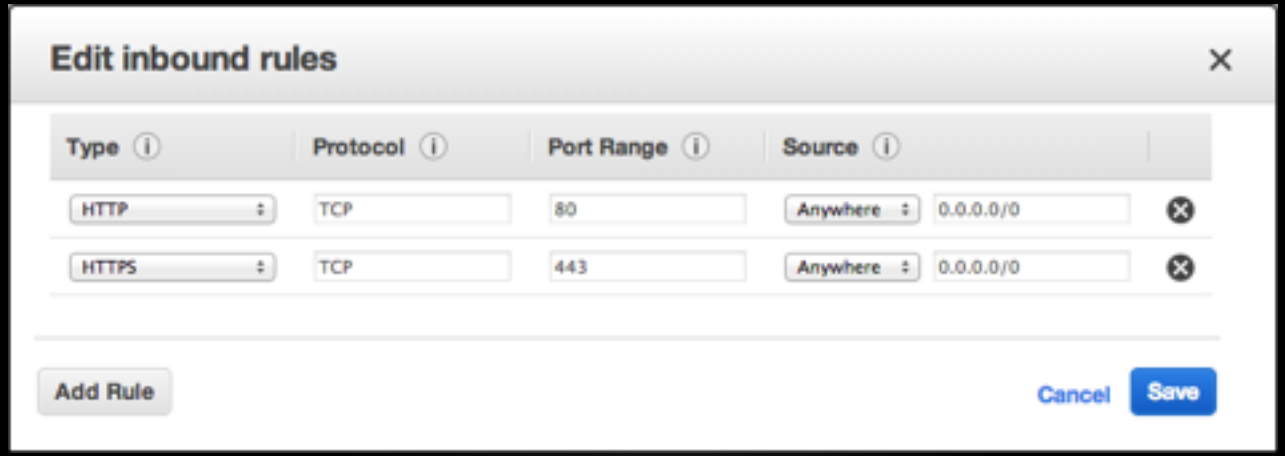
- Service Discovery
- Operational Overhead (100s of services; DevOps model absolutely required)
- Distributed Systems are inherently Complex
  - N/W Latency, Fault Tolerance, Serialization overhead ..
- Service Interface Versioning, Mismatches?
- Testing (Need the entire ecosystem to test)
- Fan out of Requests -> Increases n/w traffic

Best Practices/Tips



Best Practice -> Isolation/Access

- TIP: In AWS, use Security Groups to isolate/restrict access to your MicroServices



http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html

Best Practice -> Loadbalancers

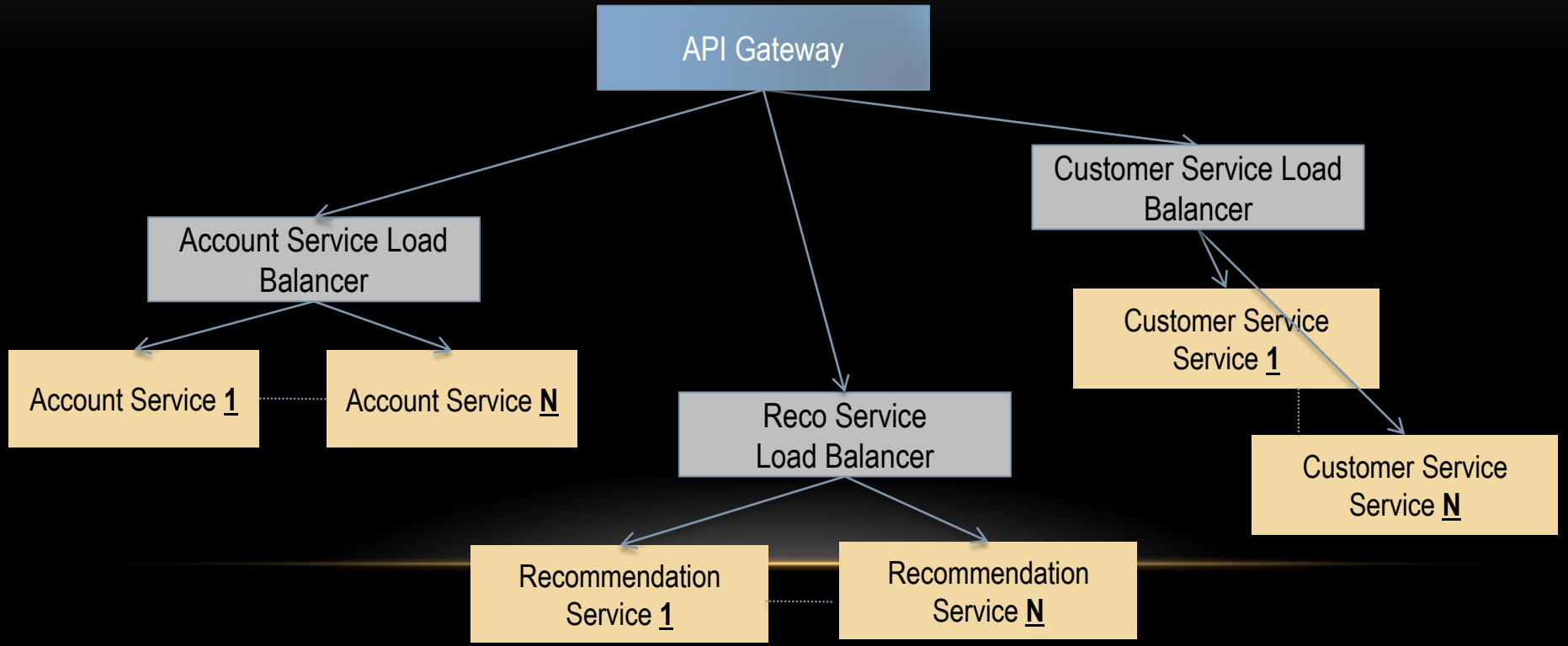
Choice

1. Central Loadbalancer? (H/W or S/W)

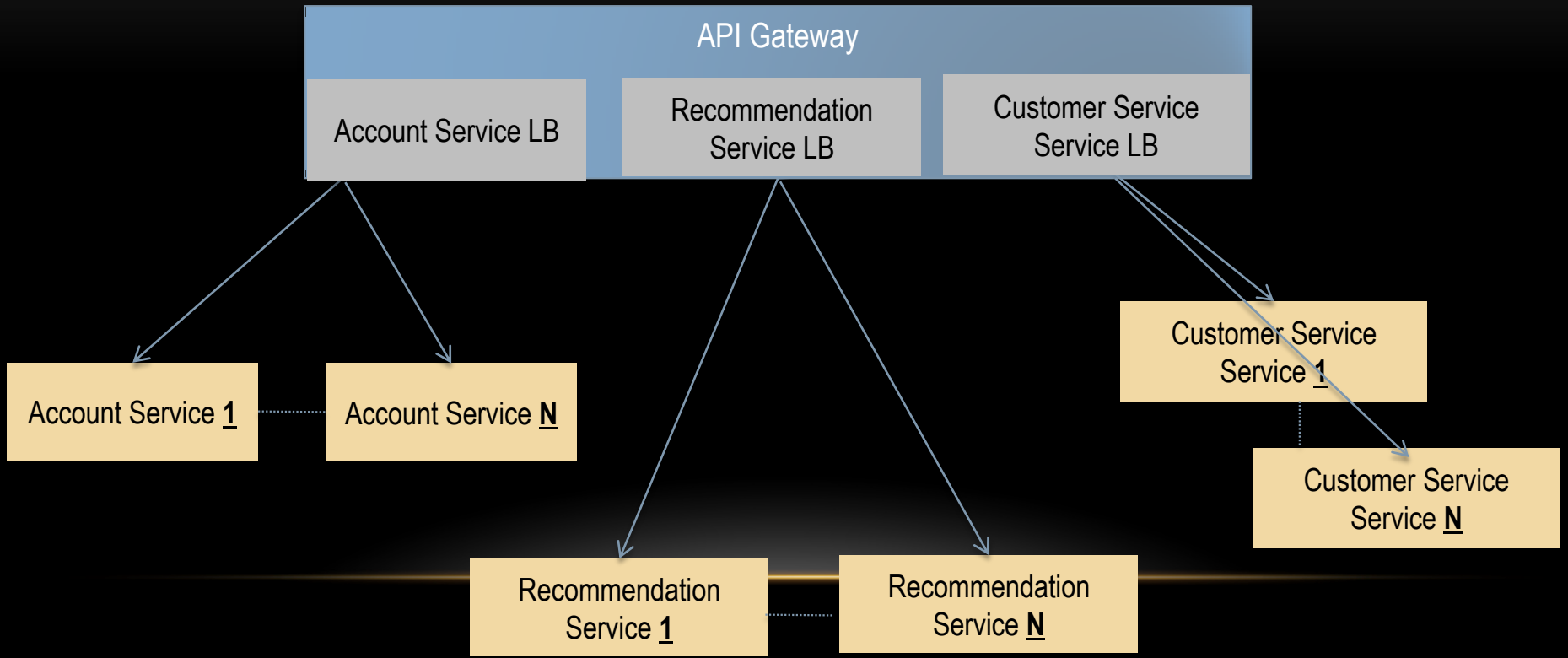
OR

2. Client based S/W Loadbalancer?

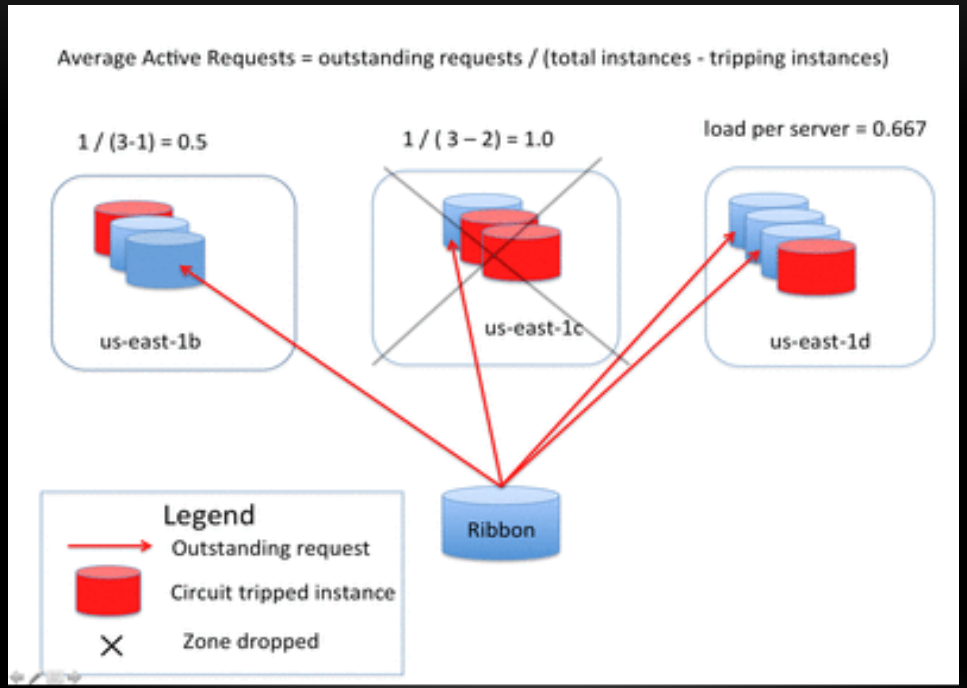
Central (Proxy) Loadbalancer



Client Loadbalancer



Client based Smart Loadbalancer



Use Ribbon (http://github.com/netflix/ribbon)

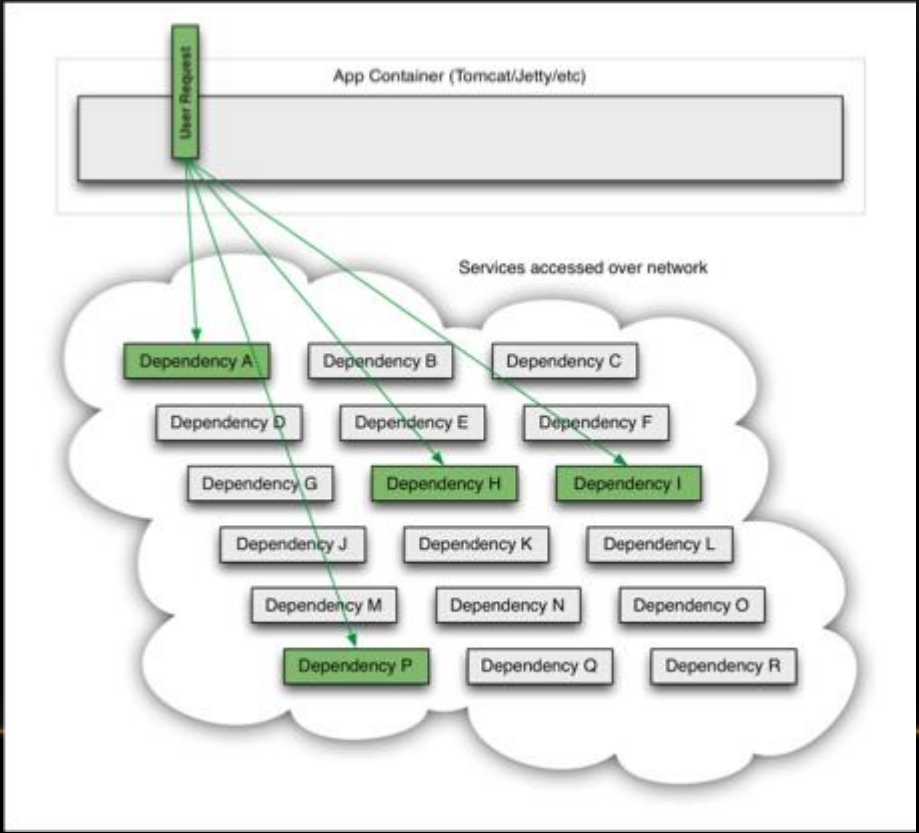
Best Practice -> LoadBalancers

- TIP: Use Client Side Smart LoadBalancers

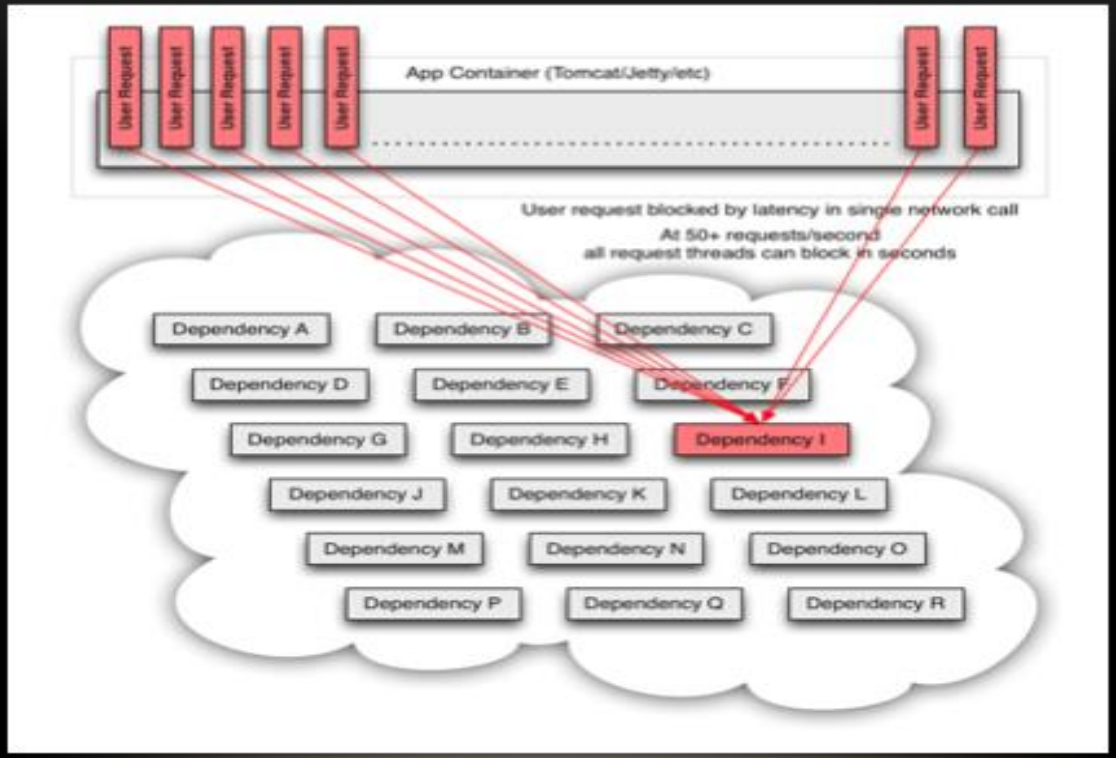
BEST PRACTICES CONTD..

- Dependency Calls
  - Guard your dependency calls
  - Cache your dependency call results
  - Consider Batching your dependency calls
  - Increase throughput via Async/ReactiveX patterns

Dependency Resiliency



Service Hosed!!



A single “bad” service can still bring your service down

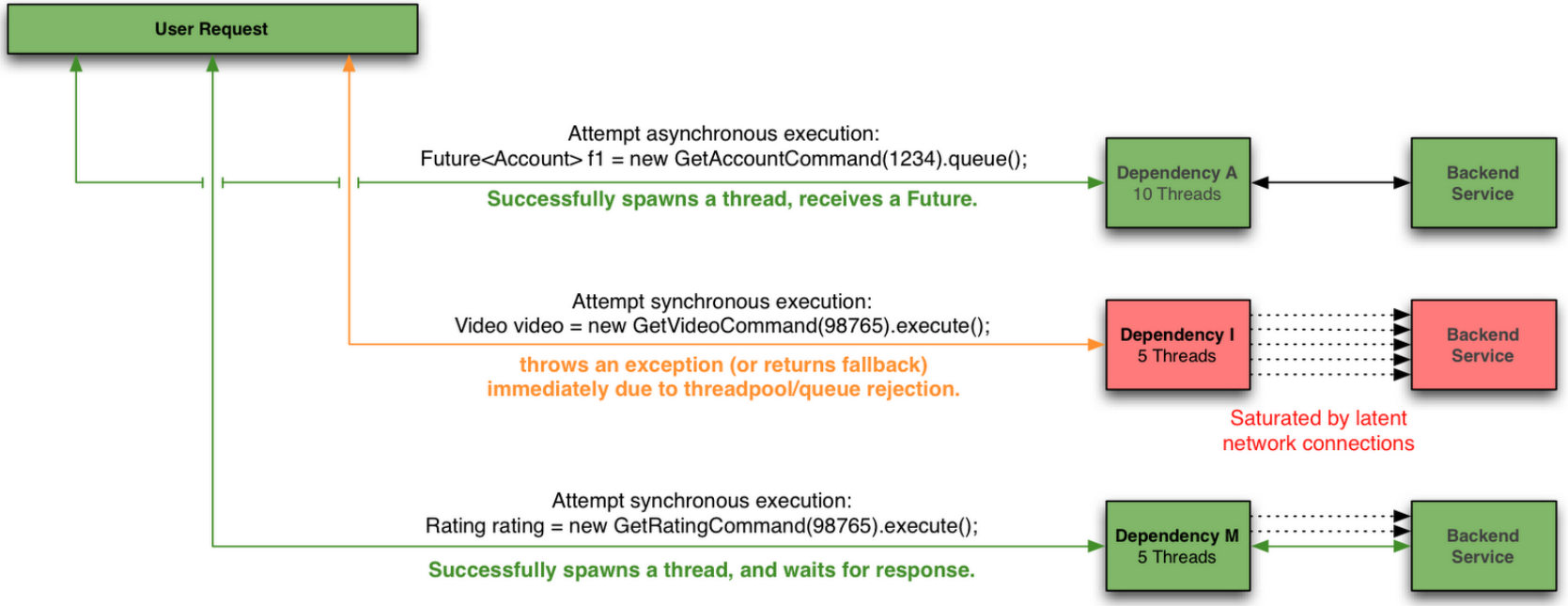
AVAILABILITY

MicroServices does not automatically mean better Availability

- Unless you have Fault Tolerant Architecture

Resiliency/Availability

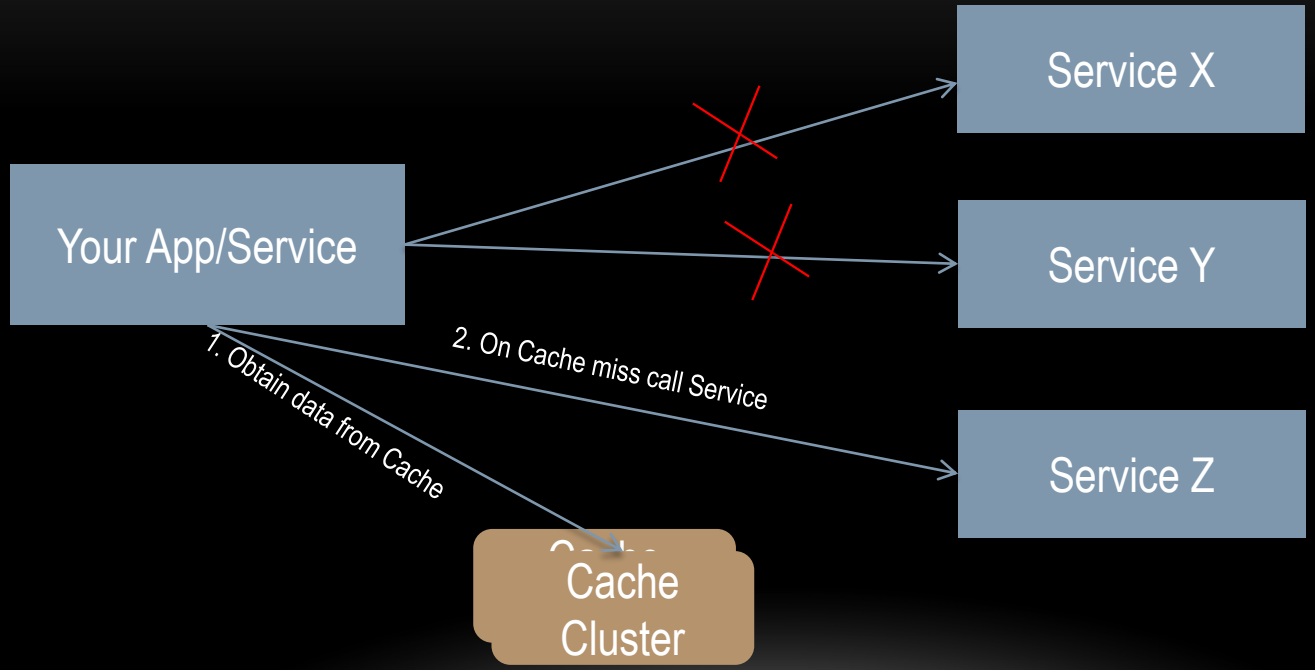
HYSTRIX
DEFEND YOUR APP
NETFLIX | OSS



Circuit Breaker, Retries, Bulk Heading and Fallbacks

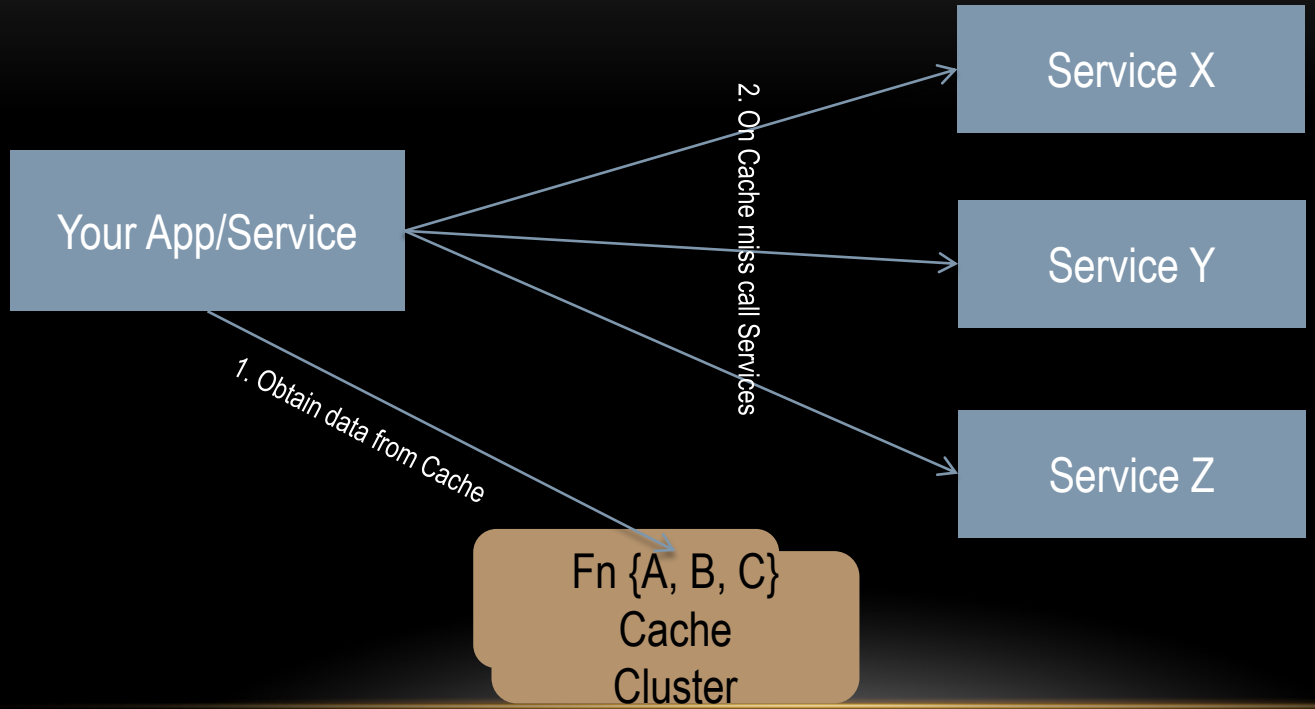
HANDLING FAN OUTS

SERVER CACHING

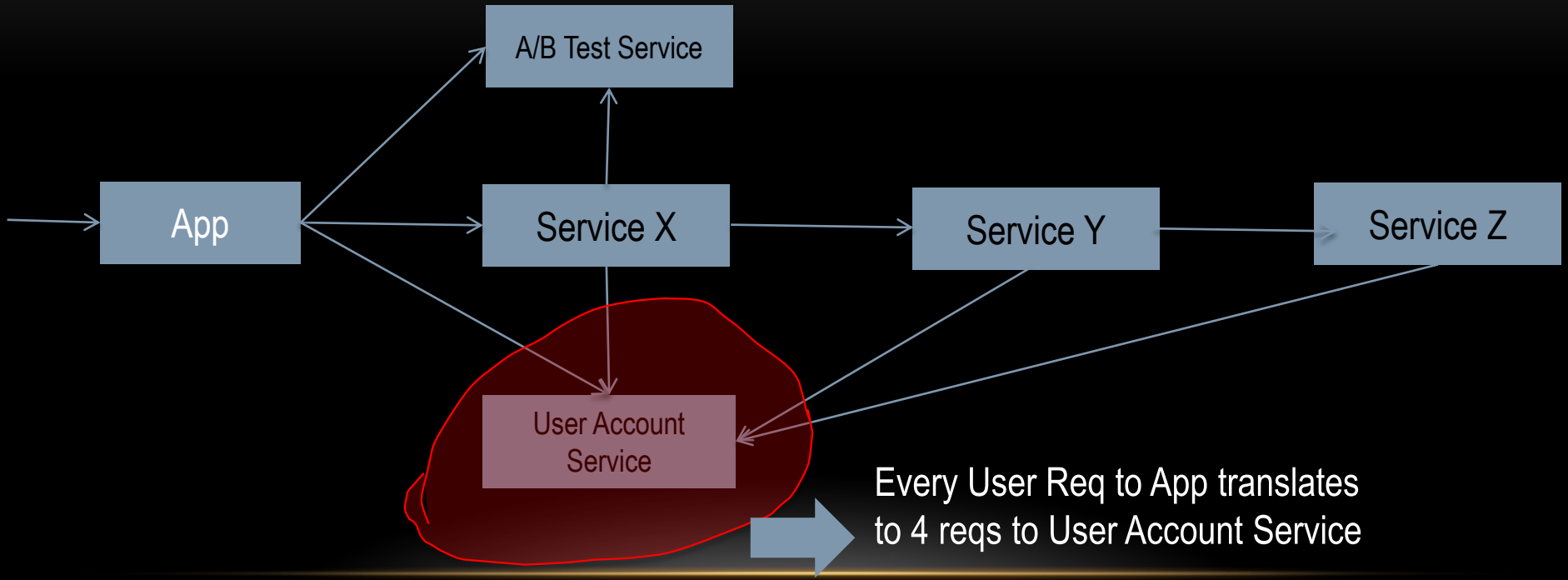


Tip: Config your TTL based on flexibility with data staleness!

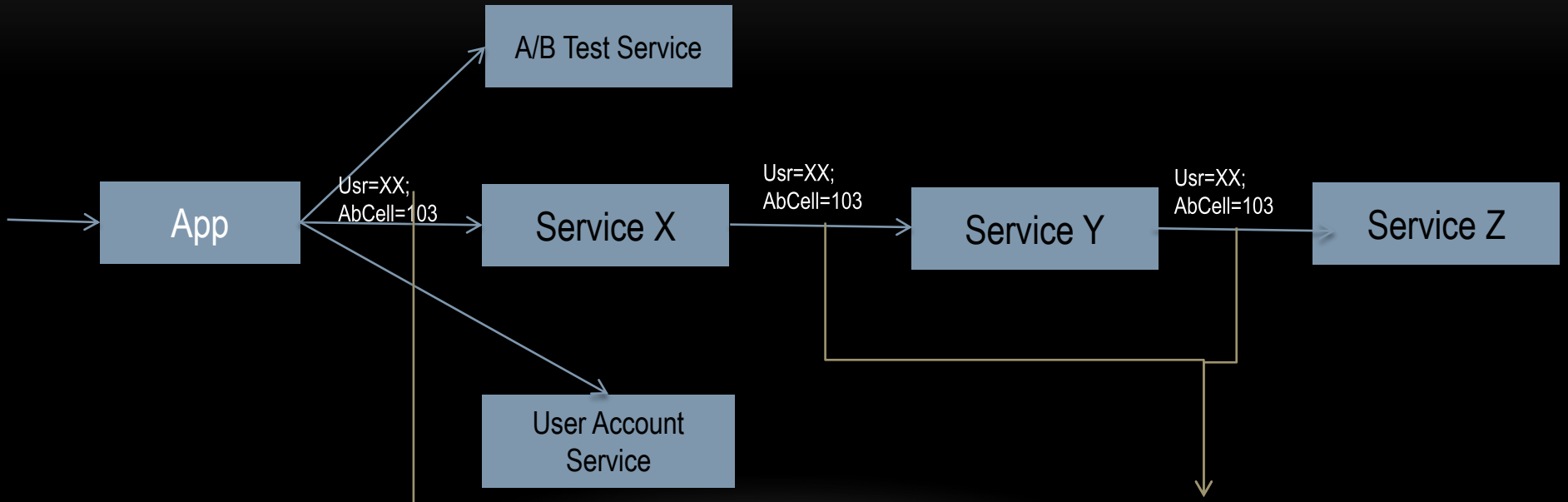
Composite (Materialized View) Caching



BottleNecks/HotSpots



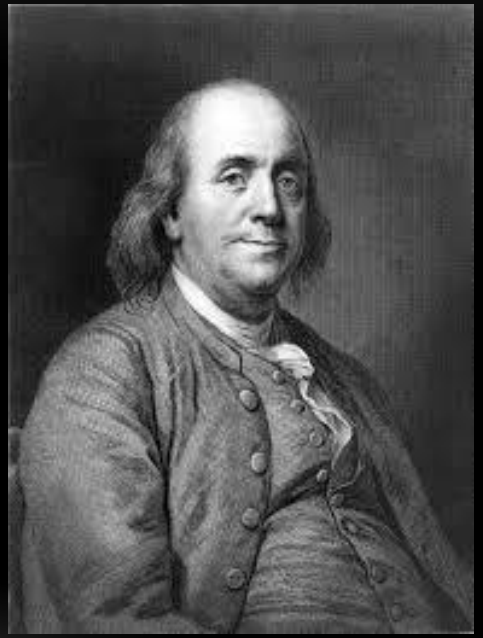
Tip: Pass data via Headers



Passing data (e.g. user and test cell) via HTTP Headers reduces dependency and load on User Account Service

TEST RESILIENCY (of Overall MicroServices)

- There are only two things certain in life*
  - Death
  - Taxes



* Benjamin Franklin

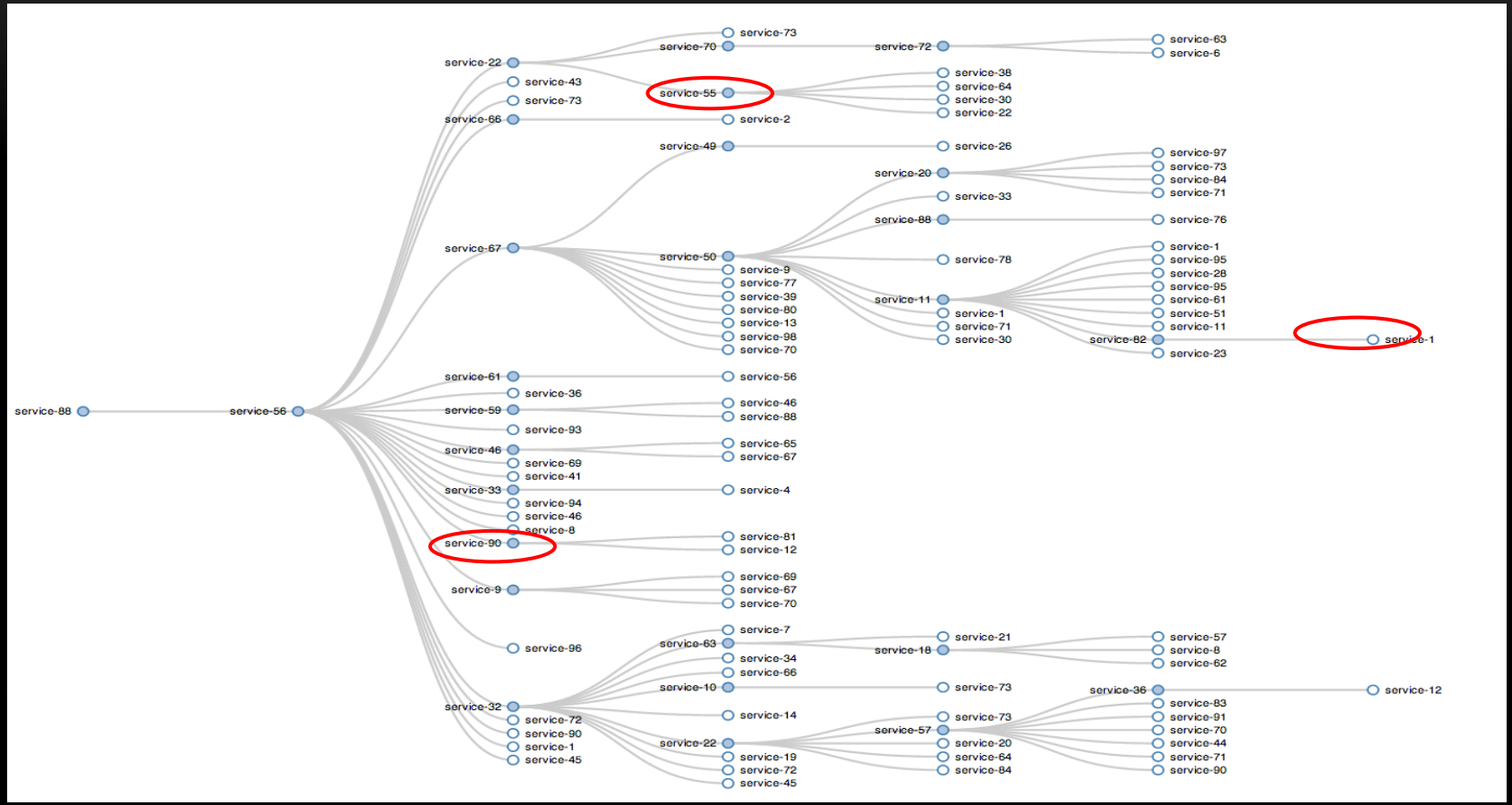
- There are only **three** things certain in life*
  - Death
  - Taxes
  - **Outages in Production**

* Inspired by Benjamin Franklin

Best Practices contd..

- Test Services for Resiliency
  - Latency/Error tests (via Simian Army)
  - Dependency Service Unavailability
  - Network Errors

Test Resiliency – to dependencies



TEST RESILIENCY

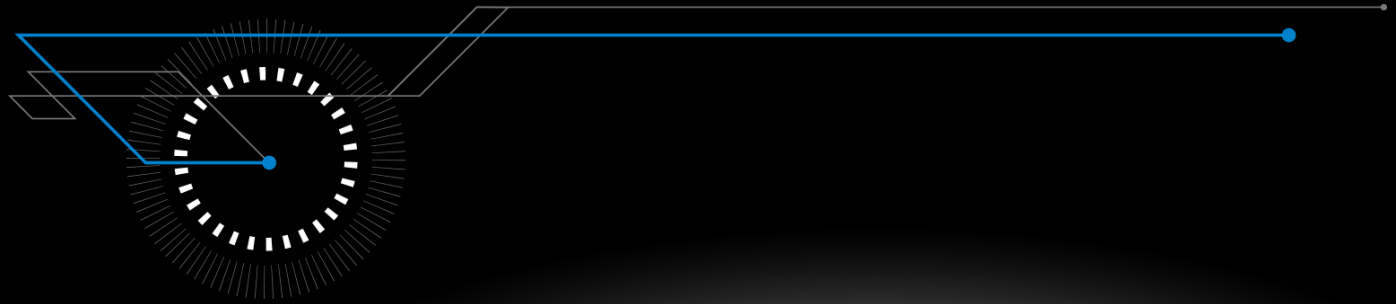
Use Simian Army https://github.com/Netflix/SimianArmy

NETFLIX | OSS

BEST PRACTICES - SUMMARY

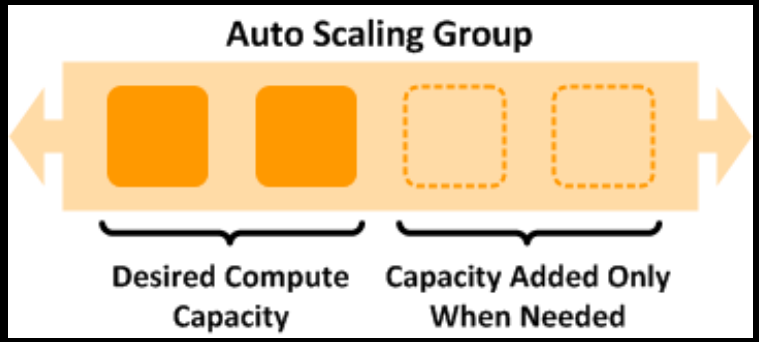
- Isolate your services (Loosely Coupled)
- Use Client Side Smart LoadBalancers
- Dependency Calls
  - Guard your dependency calls
  - Cache your dependency call results
  - Consider Batching your dependency calls
  - Increase throughput via Async/ReactiveX patterns
- Test Services for Resiliency
  - Latency/Error tests (via Simian Army)
  - Dependency Service Unavailability
  - Network Errors

Tools of the Trade



AUTO SCALING

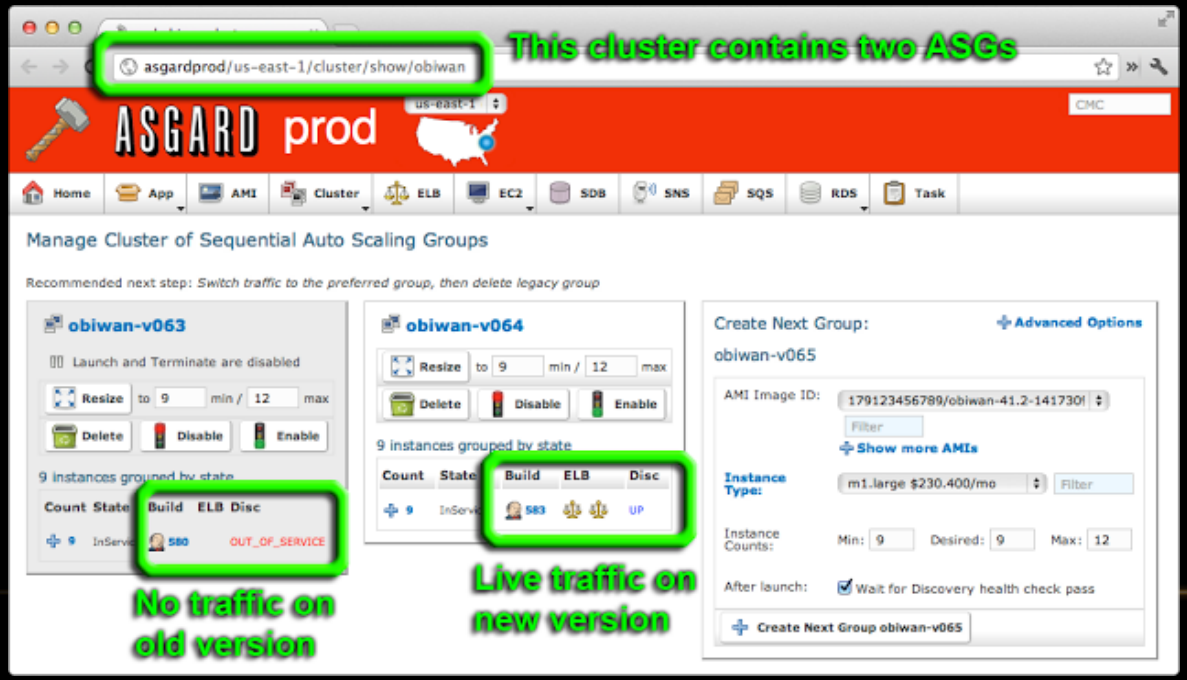
- Use AWS Auto Scaling Groups to automatically scale your microservices
  - RPS or CPU/LoadAverage via **CloudWatch** are typical metrics used to scale



http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/WhatIsAutoScaling.html

USE CANARY, RED/BLACK PUSHES

- NetflixOSS Asgard helps manage deployments



Service Dependency Visualization

MicroServices at Netflix

SERVICE DEPENDENCY GRAPH

How many dependencies does **my service** have?

What is the Call Volume on my Service?

Are any Dependency Services running **Hot**?

What are the **Top N Slowest** "Business Transactions"?

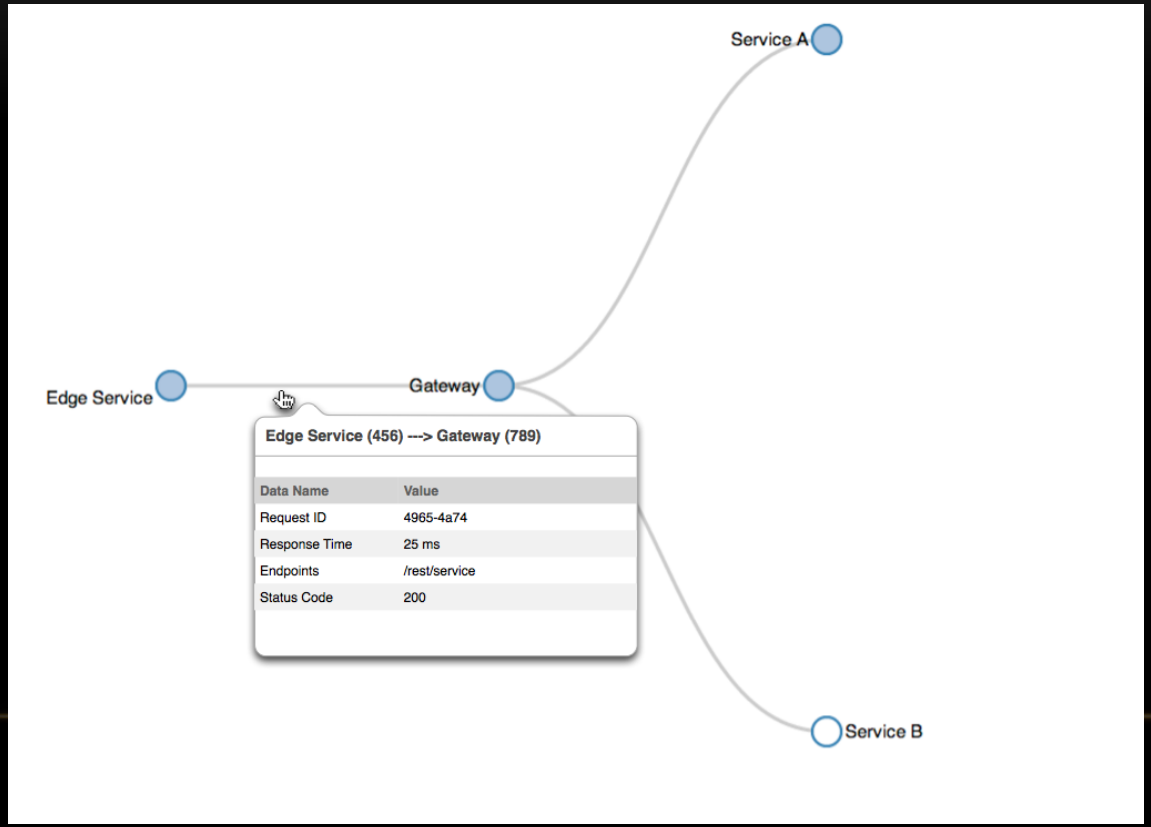
What are the **sample HTTP Requests/Responses** that had a 500 Error Code in the last 30 minutes?

SERVICE DEPENDENCY VISUALIZATION

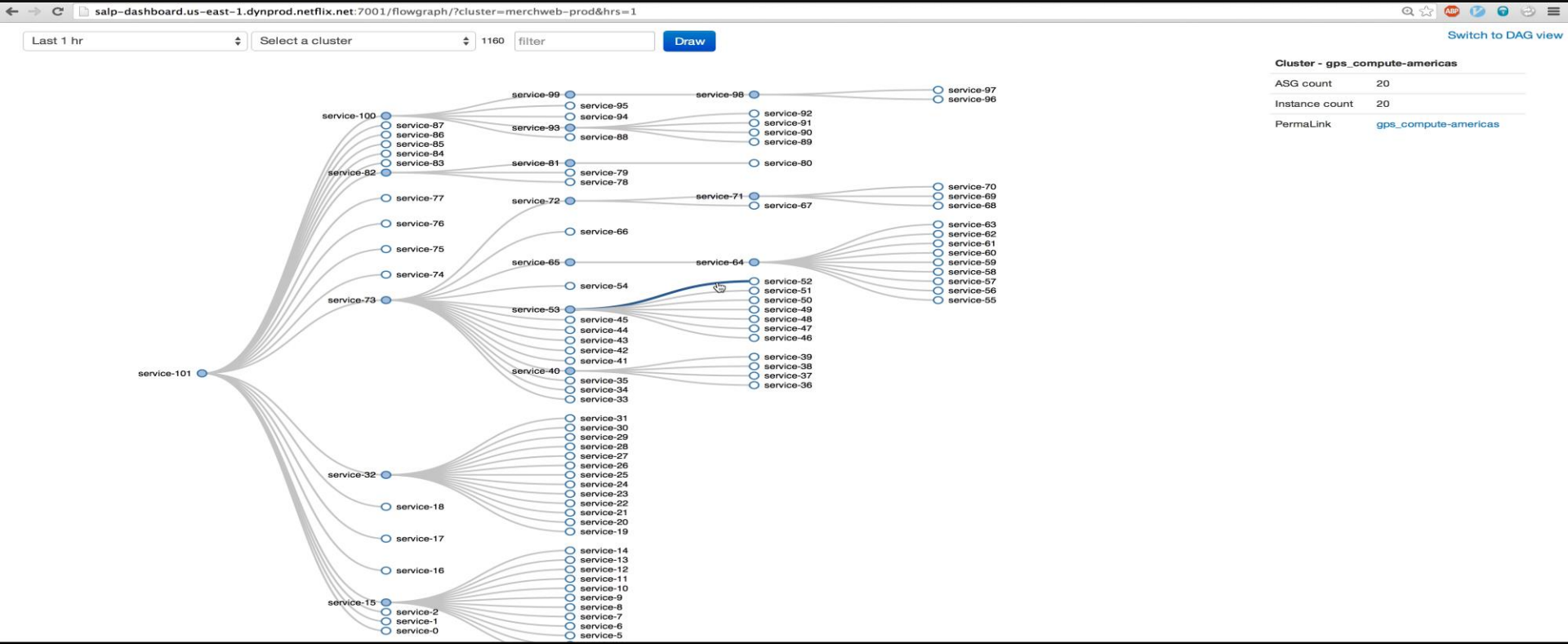
You

Your Service Dependency Graph

Service Dependency Visualization

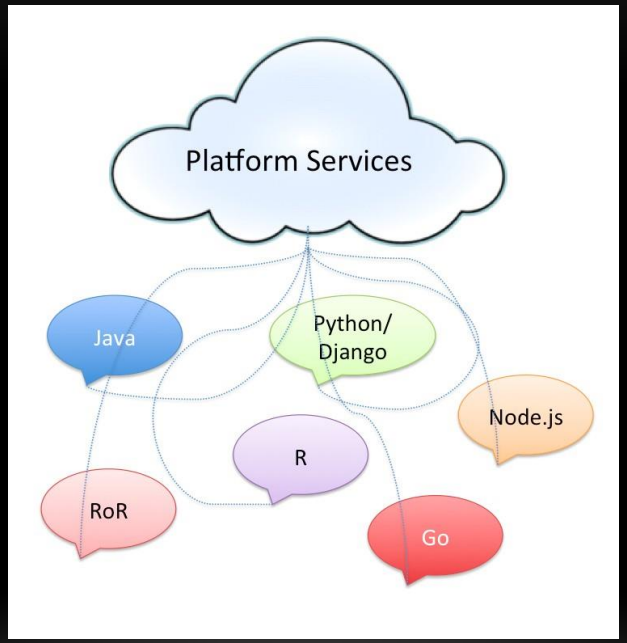


Dependency Visualization



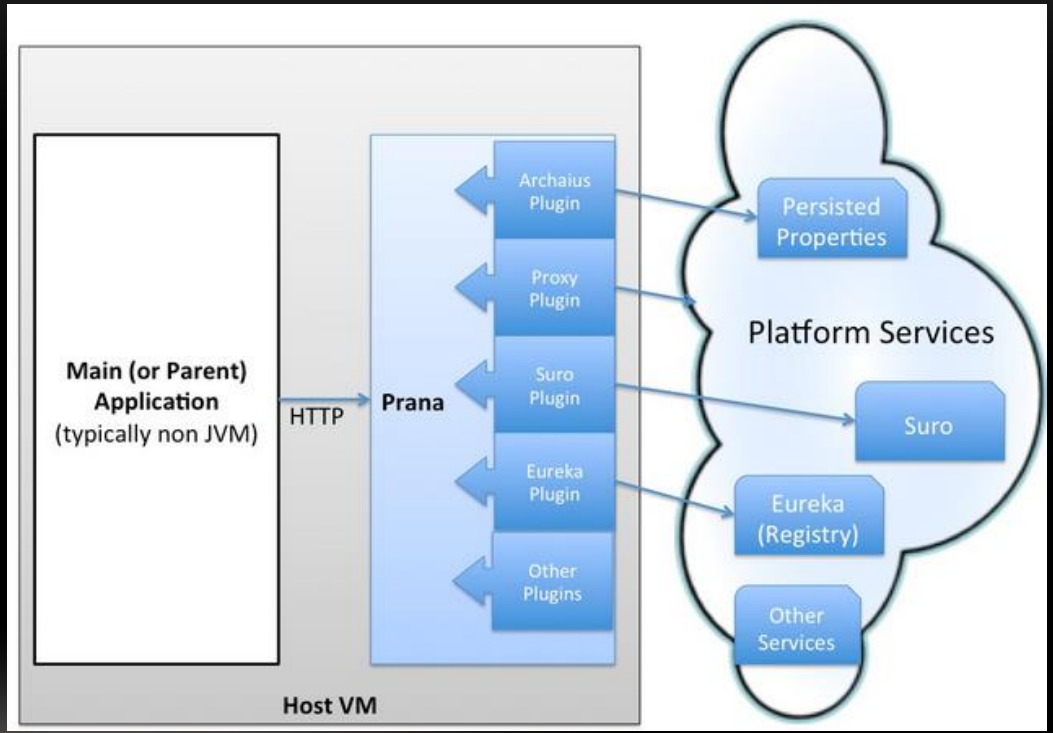
Polyglot Ecosystem

Homogeneity in A Polyglot Ecosystem



TIP: USE A SIDECAR

- Provides a common homogenous Operational/Infrastructural component for all your non-JVM based MicroServices

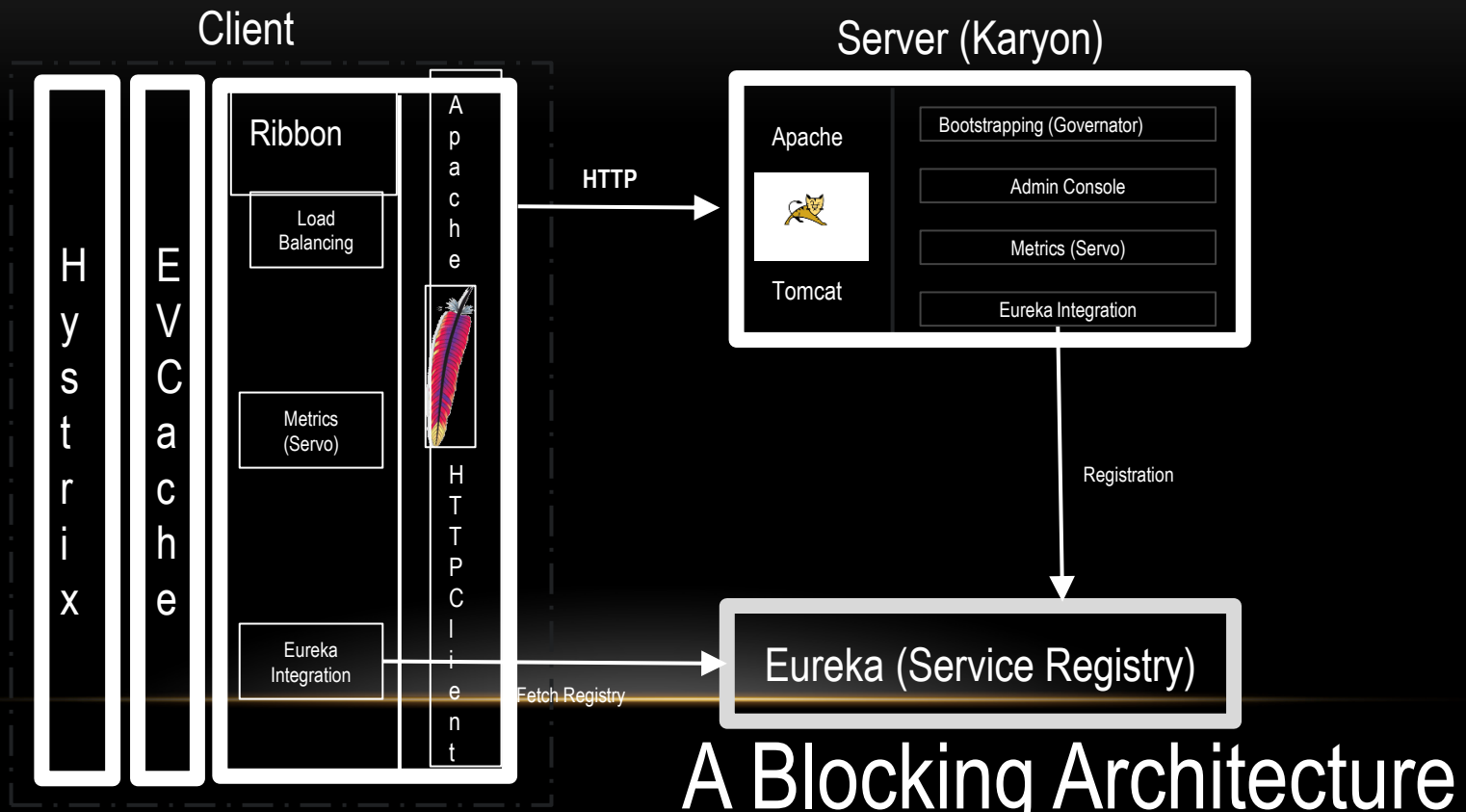


Prana Open Sourced!

- Just this morning!
- http://github.com/netflix/Prana

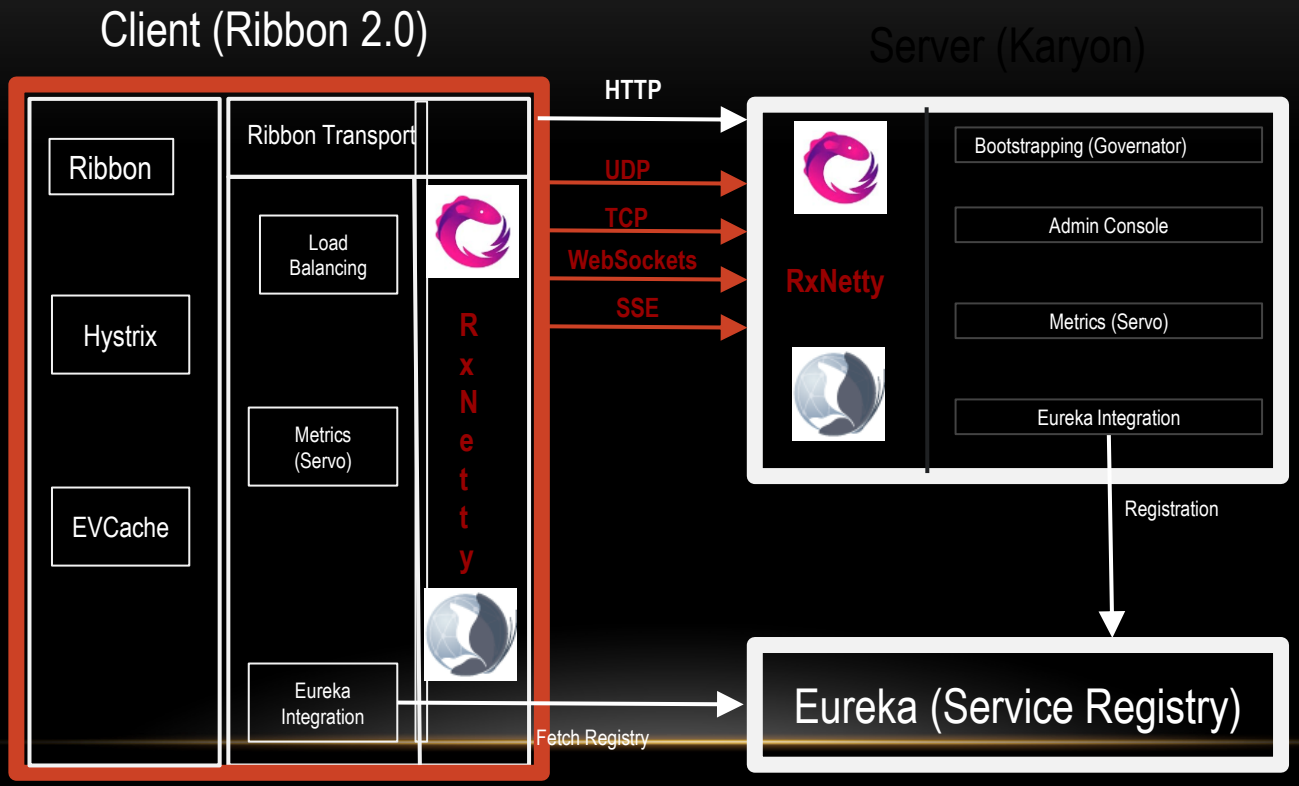
Inter Process Communication

Netflix IPC Stack (1.0)



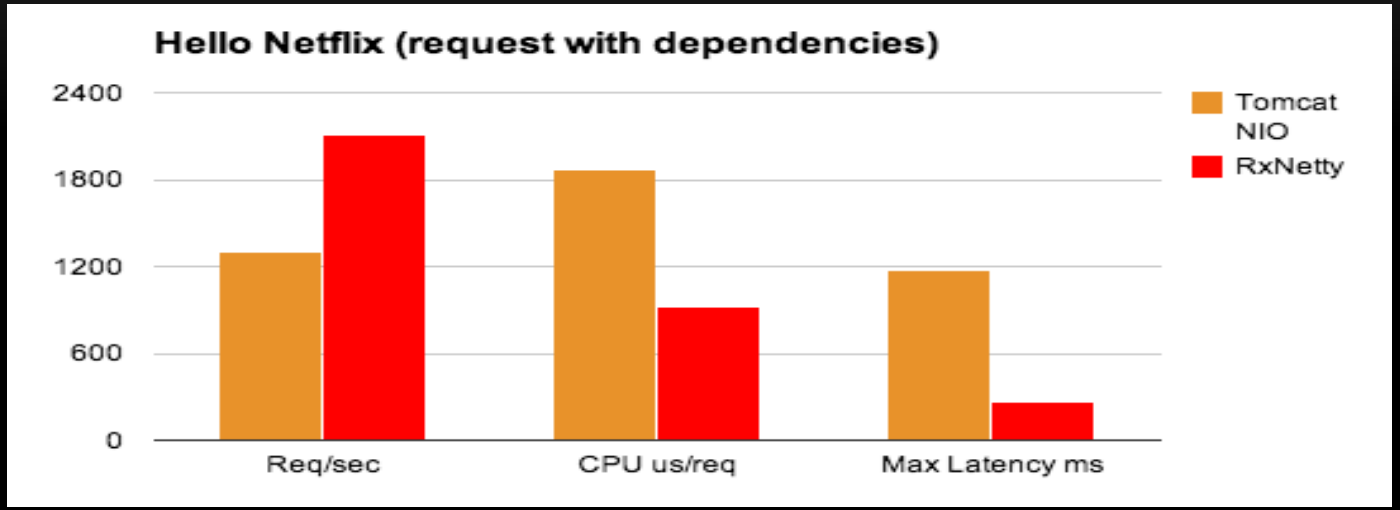
A Blocking Architecture

Netflix IPC Stack (2.0)



A Completely Reactive Architecture

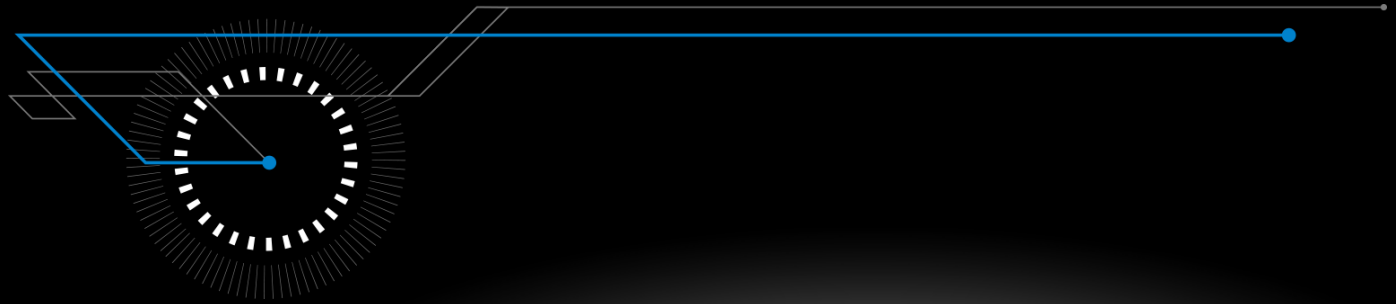
Performance – Throughput



Bounded Thread model (Tomcat) vs Reactive Async (RxNetty)

Details: http://www.meetup.com/Netflix-Open-Source-Platform/events/184153592/

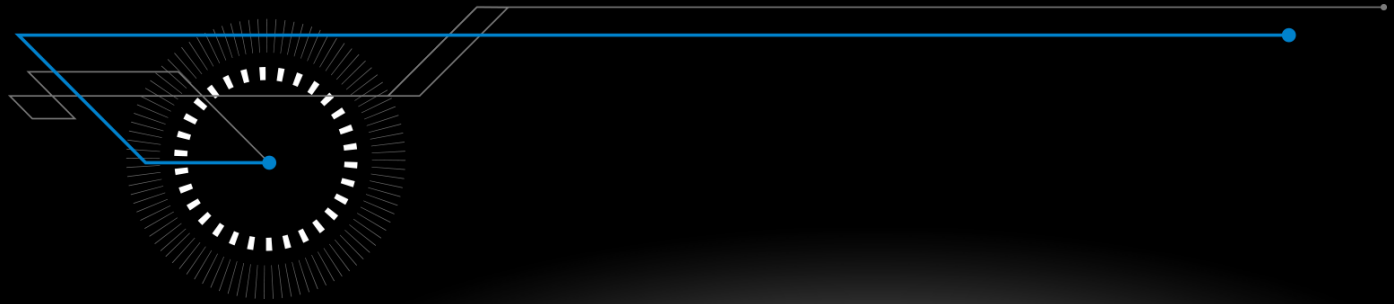
NetflixOSS



NETFLIX | OSS

- Eureka – for Service Registry/Discovery
- Karyon – for Server (Reactive or threaded/servlet container based)
- Ribbon – for IPC Client
  - And Fault Tolerant Smart LoadBalancer
- Hystrix – for Fault Tolerance and Resiliency
- Archaius – for distributed/dynamic Properties
- Servo – unified Feature rich Metrics/Insight
- EVCache – for distributed cache
- Curator/Exhibitor – for zookeeper based operations
- …

Takeaways



Takeaways

- Monolithic apps – good for small organizations
- MicroServices – have its challenges, but the benefits are many
  - Consider adopting when your **organization scales**
  - Leverage Best Practices
    - An Elastic Cloud provides the **ideal** environment (Auto Scaling etc.)
    - NetflixOSS has many libraries/samples to aid you