

**微服务容错限流**

**Hystrix架构和实践**

**讲师：杨波**

研发总监/资深架构师

第

1

部分

课程概述

# 课程大纲

01

课程概述

02

容错限流需求

03

容错限流原理

04

Netflix Hystrix背景介绍

05

Hystrix设计原理

06

Hystrix主要概念

07

信号量vs线程池隔离

08

Hystrix主要配置

09

Hystrix基础实验(Lab01)

# 课程大纲

- 10 Hystrix模拟案例分析(Code Review )
- 11 Hystrix+ Dashboard实验(Lab02)
- 12 网关集成Hystrix(Code Review)
- 13 Spring Cloud Hystrix实验(Lab03)
- 14 Netflix Turbine简介
- 15 Hystrix生产最佳实践
- 16 参考资料和后续课程预览

# 课程概述和亮点

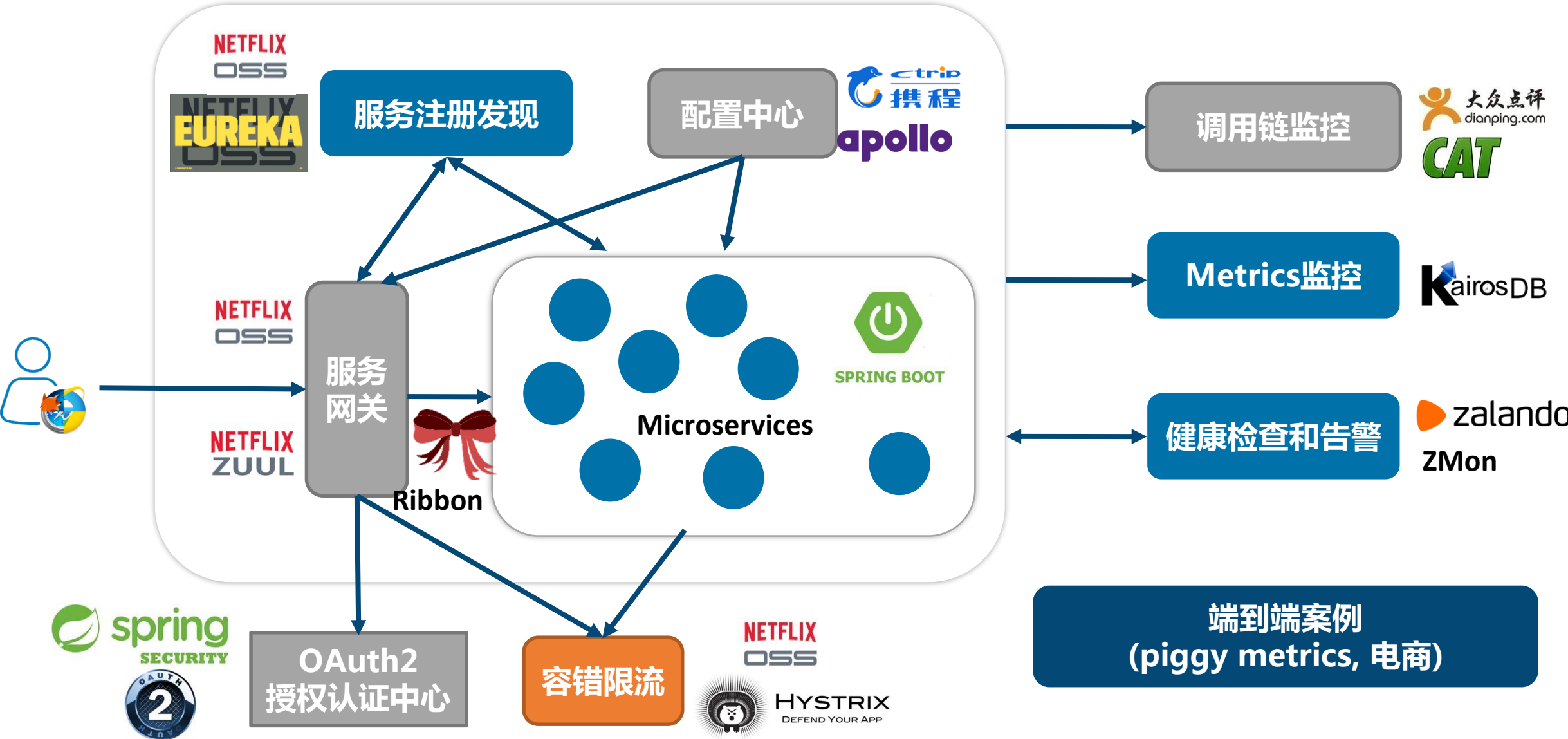


- 1 杨波的微服务基础架构体系的**第五个**模块
- 2 容错限流**原理模式**剖析
- 3 Netflix开源容错限流组件**Hystrix**深度剖析
- 4 **Hystrix**案例和实验
- 5 **Zuul**网关和**Hystrix**集成
- 6 **Hystrix**生产最佳实践
- 7 **Spring Cloud Hystrix**简介

# 杨波的微服务基础架构体系2018预览(draft)



# 架构和技术栈预览



第

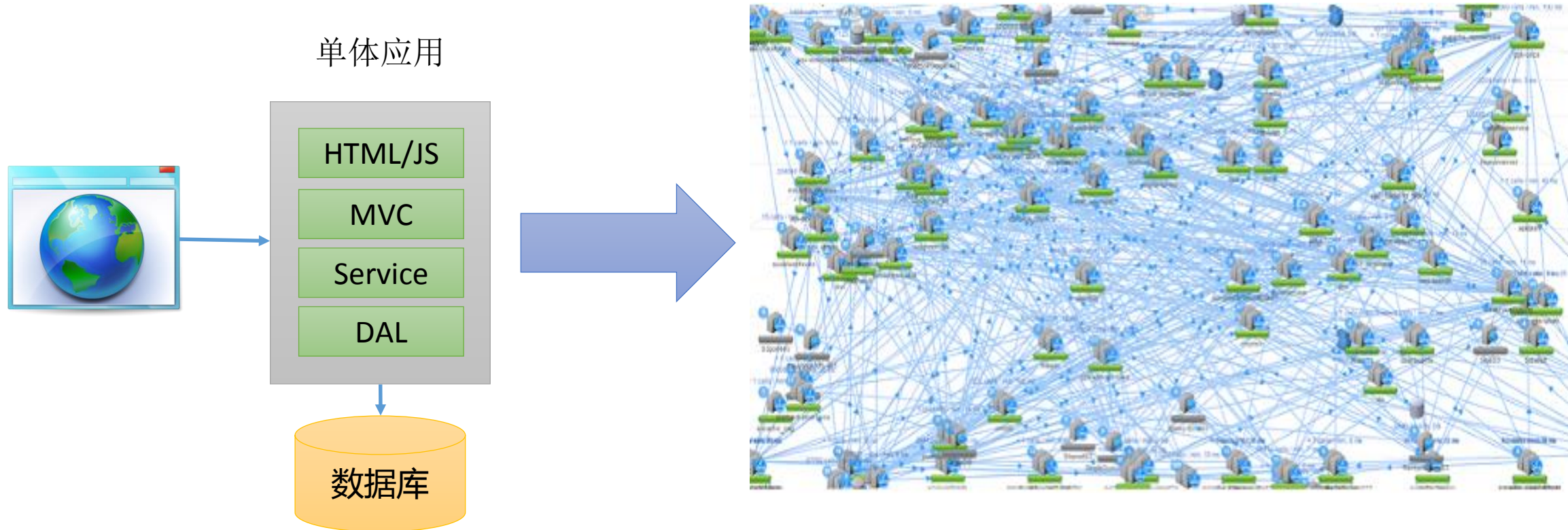
2

部分

容错限流需求



# 从单块到微服务



# 微服务可用性

## ■ 问题

假定一个**单块**服务的可用性是**99.99%**，如果我们有~**30个微服务**，每个的可用性都是99.99%，那么总体可用性是多少呢？

## ■ 总体可用性

**99.7%** uptime!  
每月**2小时**宕机时间  
实际情况往往更糟糕

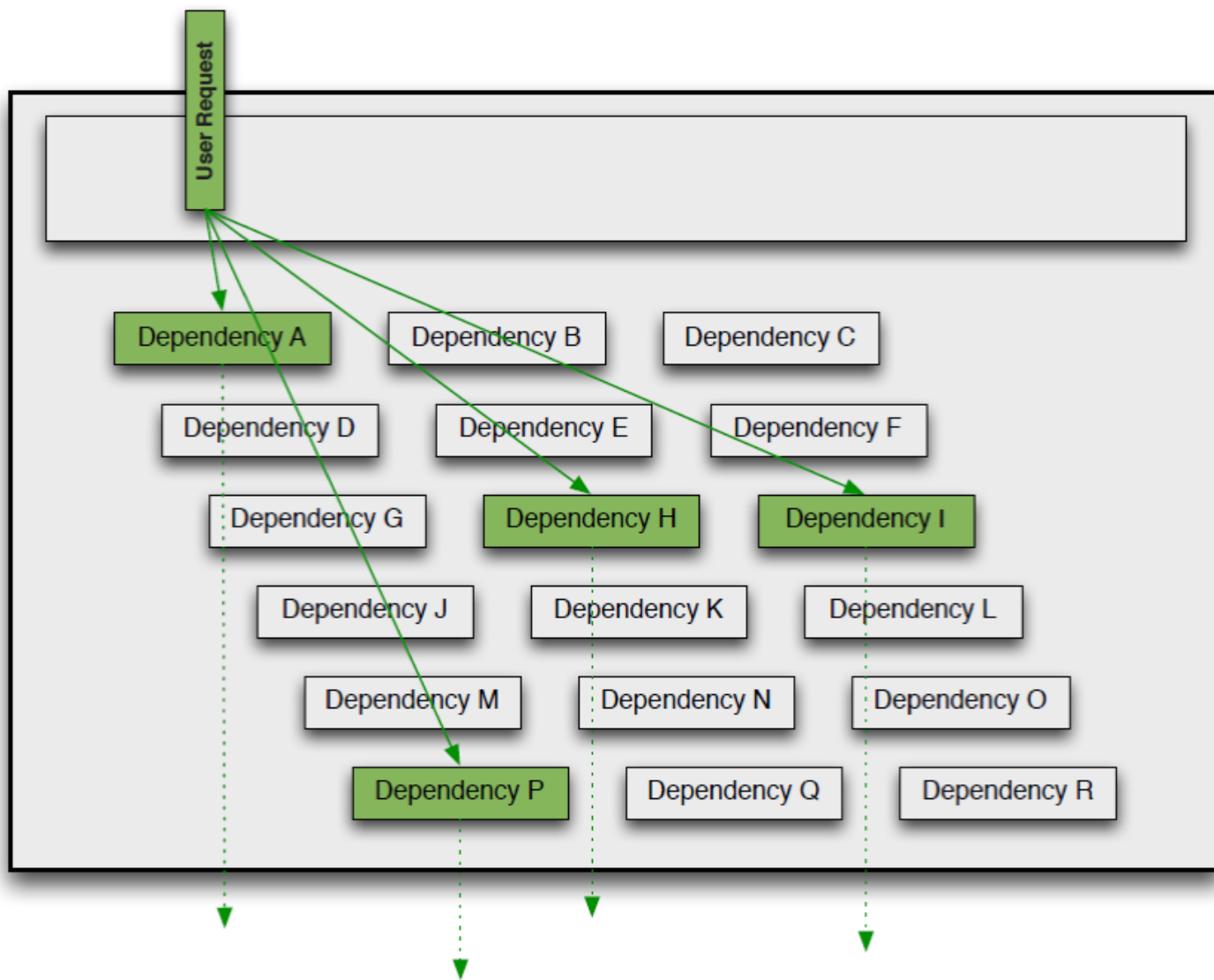


# 容错限流缺失的坑

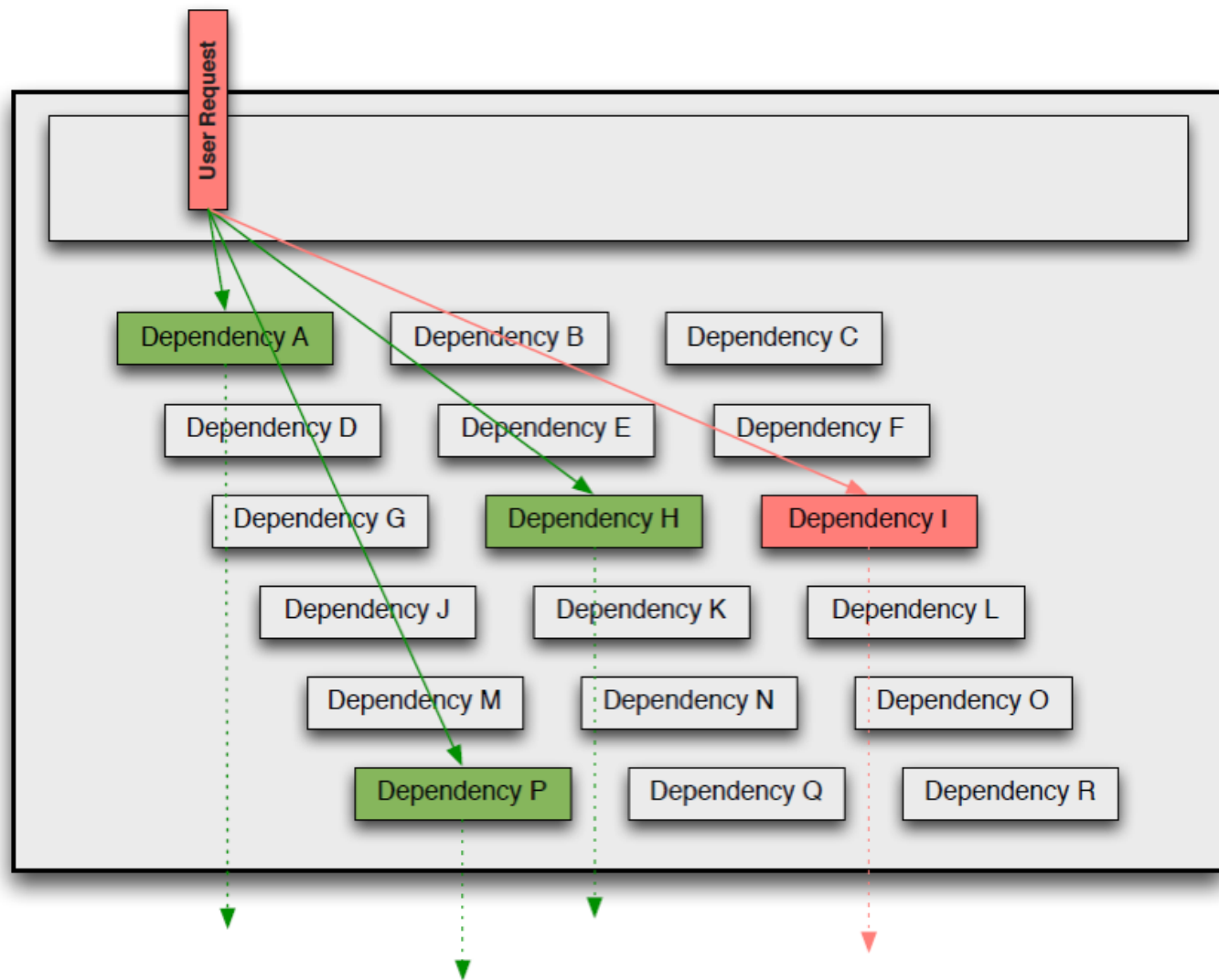
- 复杂分布式系统通常有很多依赖，如果一个应用不能对来自依赖故障进行隔离，那么应用本身就处在被拖垮的风险中。在一个高流量的网站中，某个单一后端一旦发生延迟，将会在数秒内导致所有应用资源被耗尽
- 一个臭鸡蛋影响一篮筐
- 微服务需要容错限流！！！！



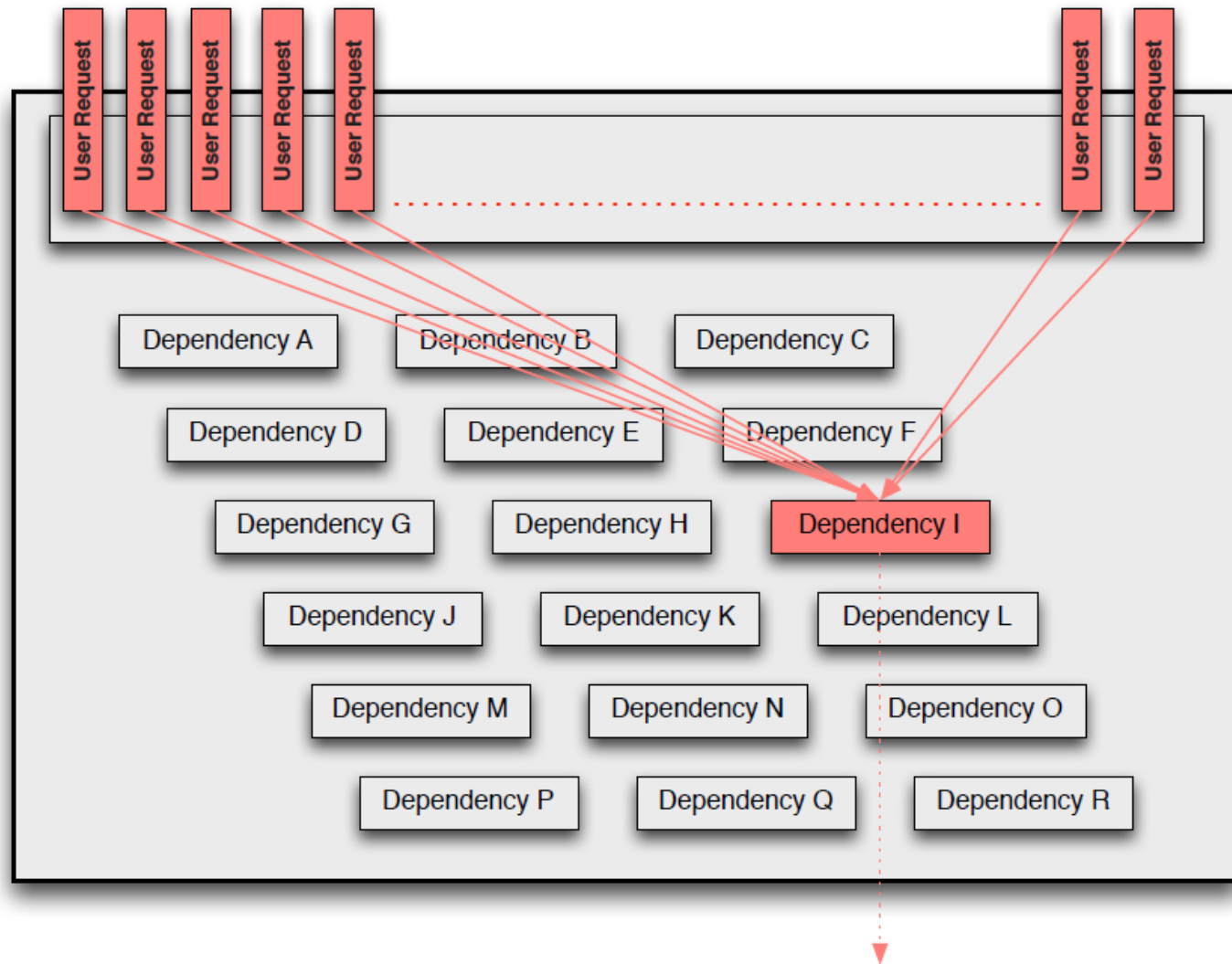
# 服务依赖



# 单个服务延迟



# 高峰期致雪崩效应



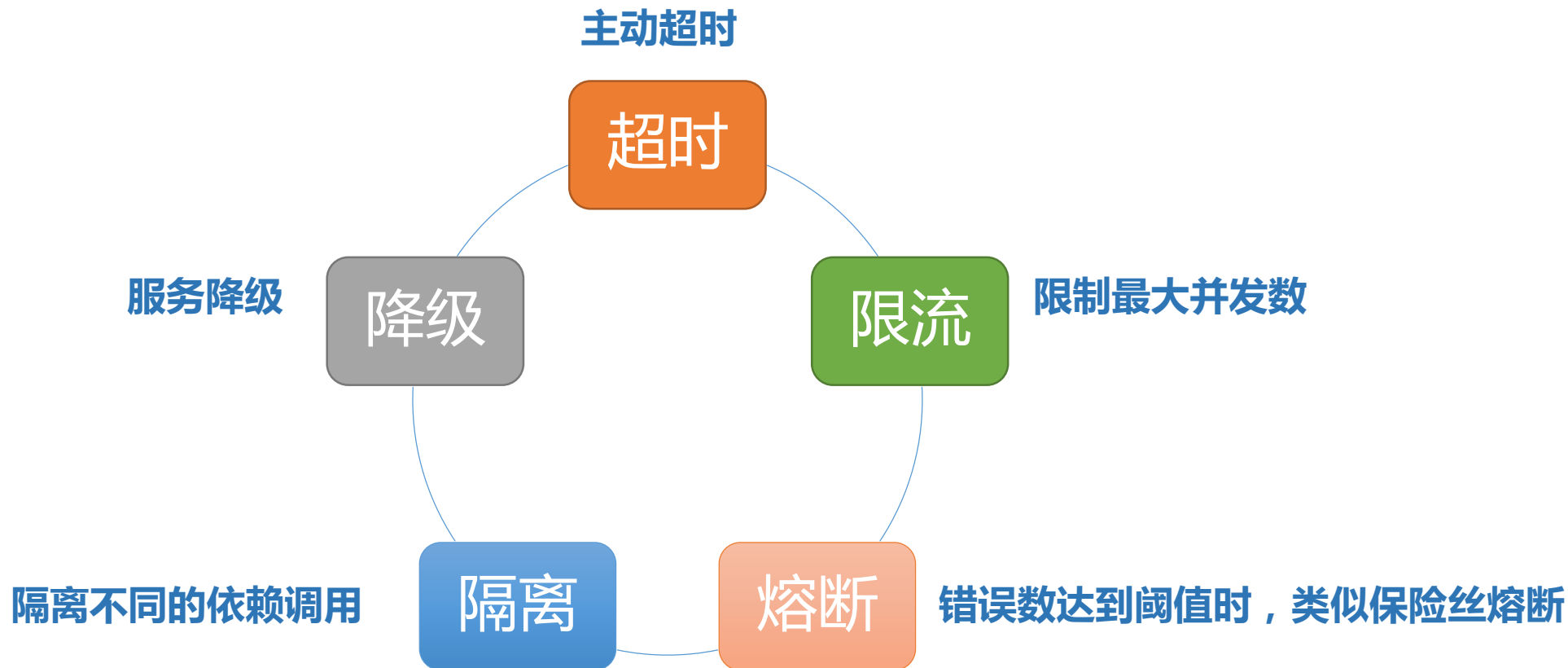
第

3

部分

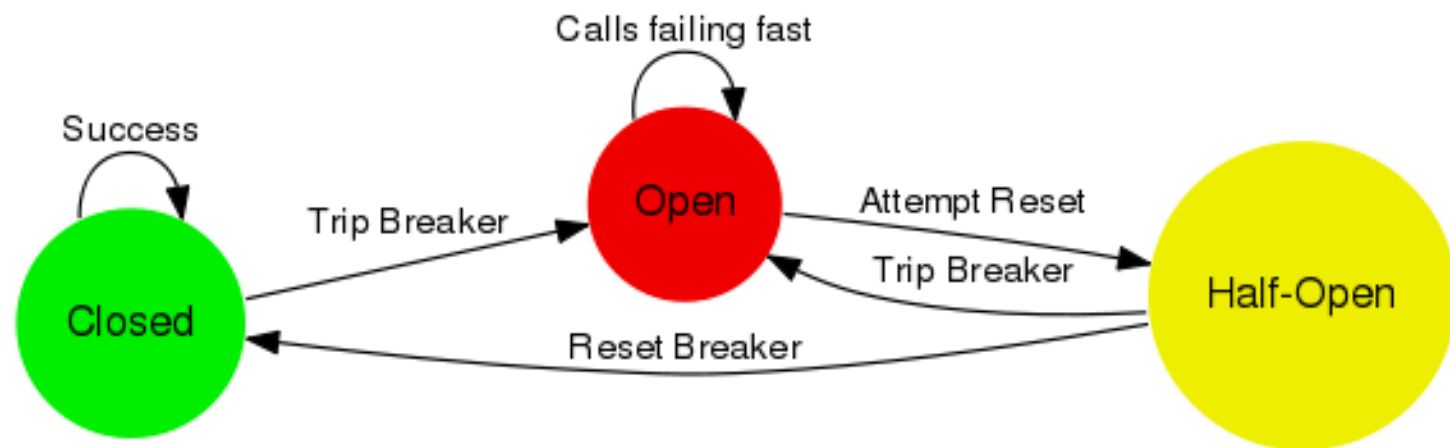
## 容错限流原理

# 基本的容错模式





# 断路器模式



The Pragmatic Programmers

## Release It!

Design and Deploy  
Production-Ready Software



Michael T. Nygard



# 舱壁隔离模式



# 容错理念

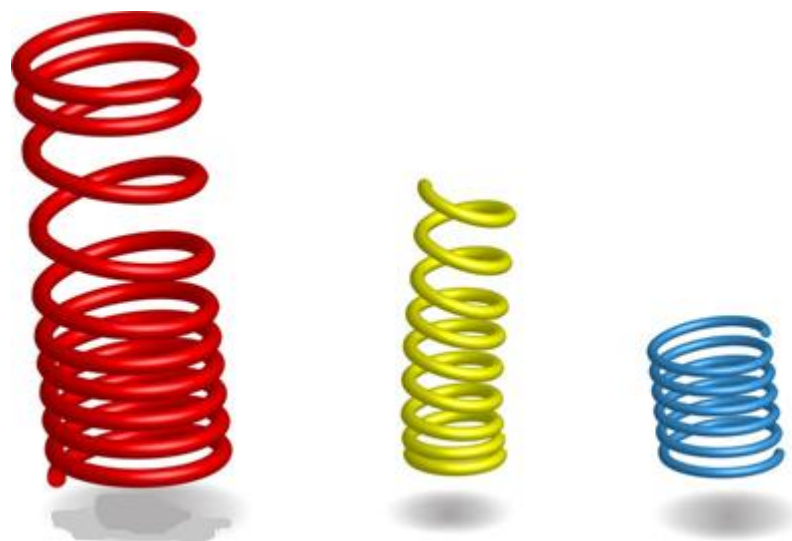
- 凡是依赖都可能会失败
- 凡是资源都有限制
  - CPU/Memory/Threads/Queue
- 网络并不可靠
- 延迟是应用稳定性杀手



# 弹性(Resilience)理念

在被弯曲，压缩或者拉伸之后，能够恢复原状的能力。  
从疾病，抑郁和困境等类似情况中恢复出来的能力。

**工程师需要弹性思维！！！！**



第

4

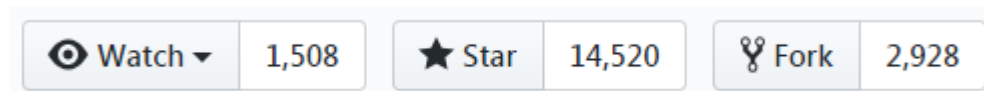
部分

# Netflix Hystrix背景介绍

# 起源和现状(2013)



- Hystrix源于Netflix API团队在2011年启动的弹性工程项目，目前(2013)它在Netflix每天处理**数百亿**的线程隔离以及**数千亿**的信号量隔离调用
  - <http://www.infoq.com/cn/news/2013/01/netflix-hystrix-fault-tolerance>
- Hystrix是基于Apache License 2.0协议的开源库，目前托管在github上，当前超过**1.4万**颗星
  - <https://github.com/Netflix/Hystrix>
- Netflix云端开源工具Hystrix曾助奥巴马竞选
  - <http://it.sohu.com/20121129/n358943361.shtml>



# Hystrix主要作者



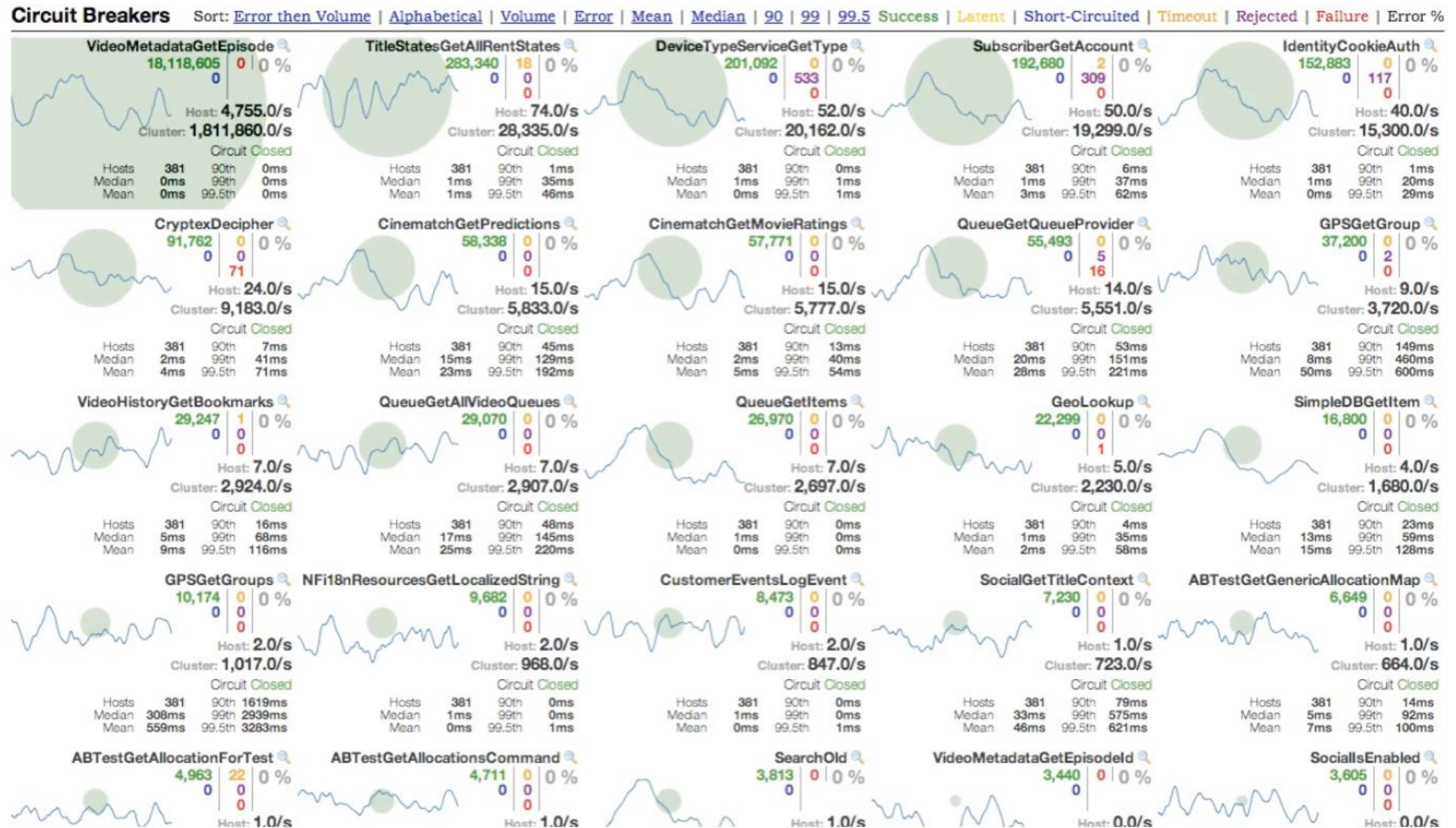
**Ben Christensen**

<https://www.linkedin.com/in/benjchristensen>



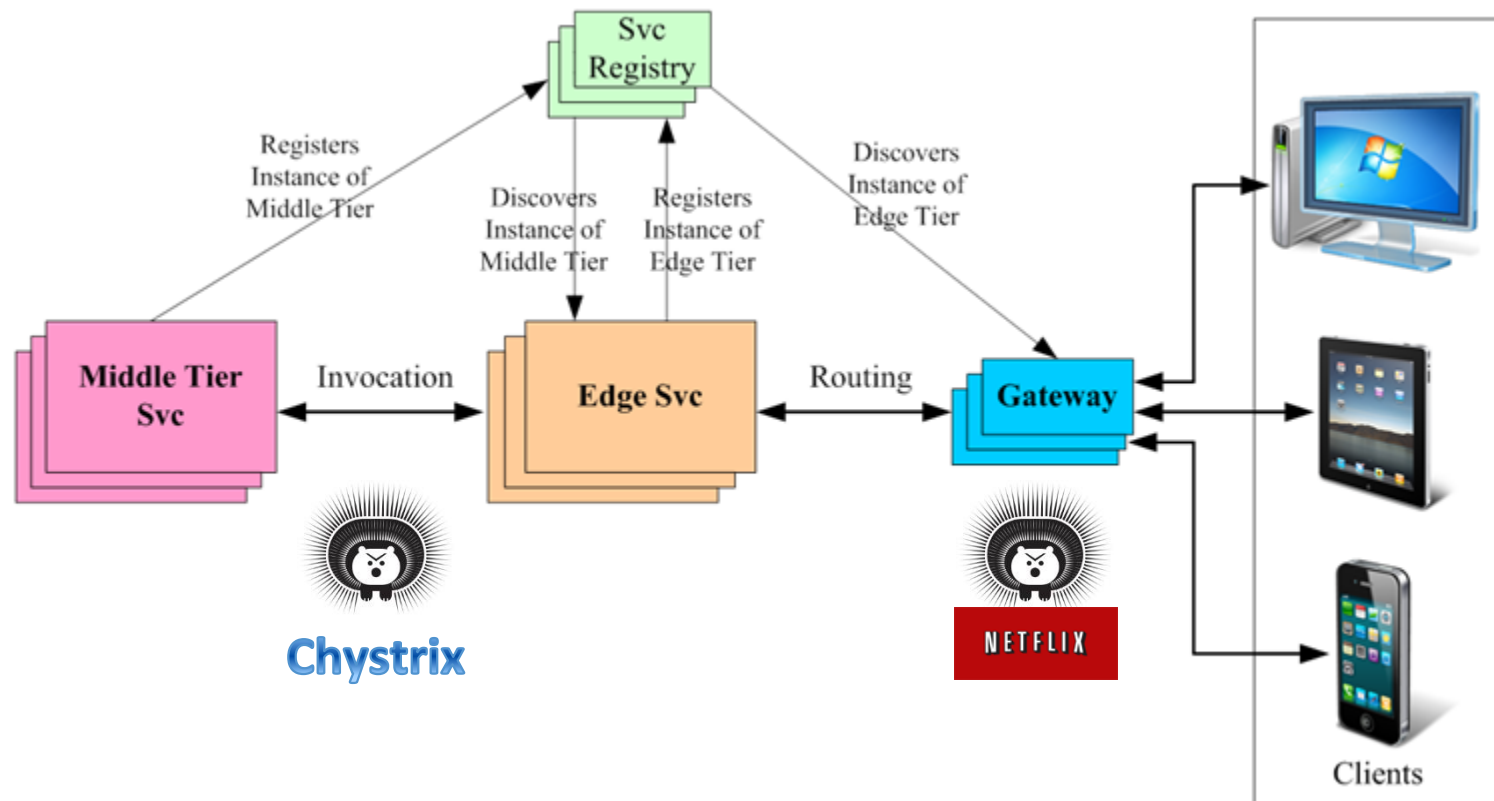


# Netflix Hystrix Dashboard





# 携程案例(2015)



# CTrip Hystrix Dashboard

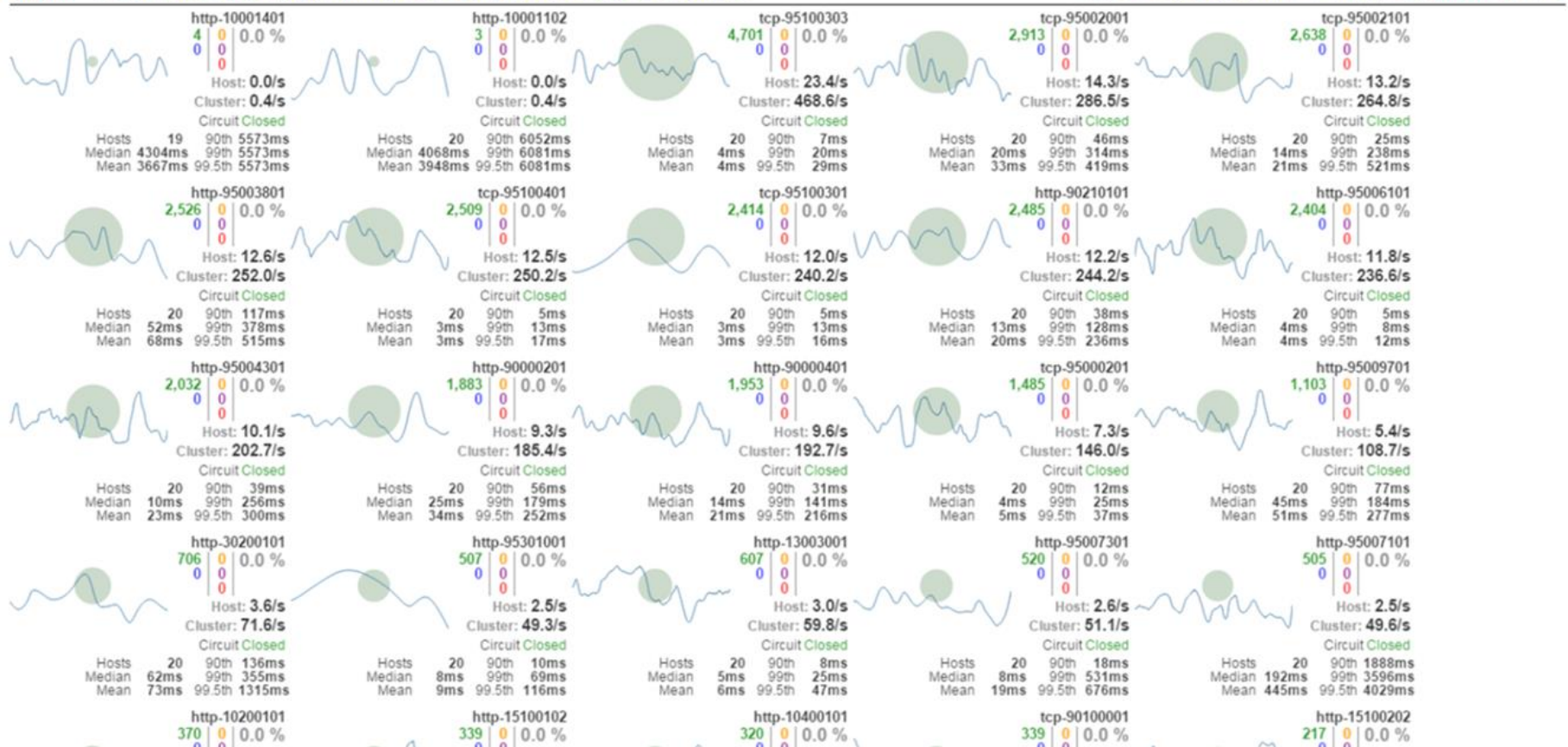
Hystrix Stream: <http://turbine.soa.ctripcorp.com/turbine/turbine.stream?cluster=tcp-gatekeeper>



**HYSTRIX**  
DEFEND YOUR APP

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



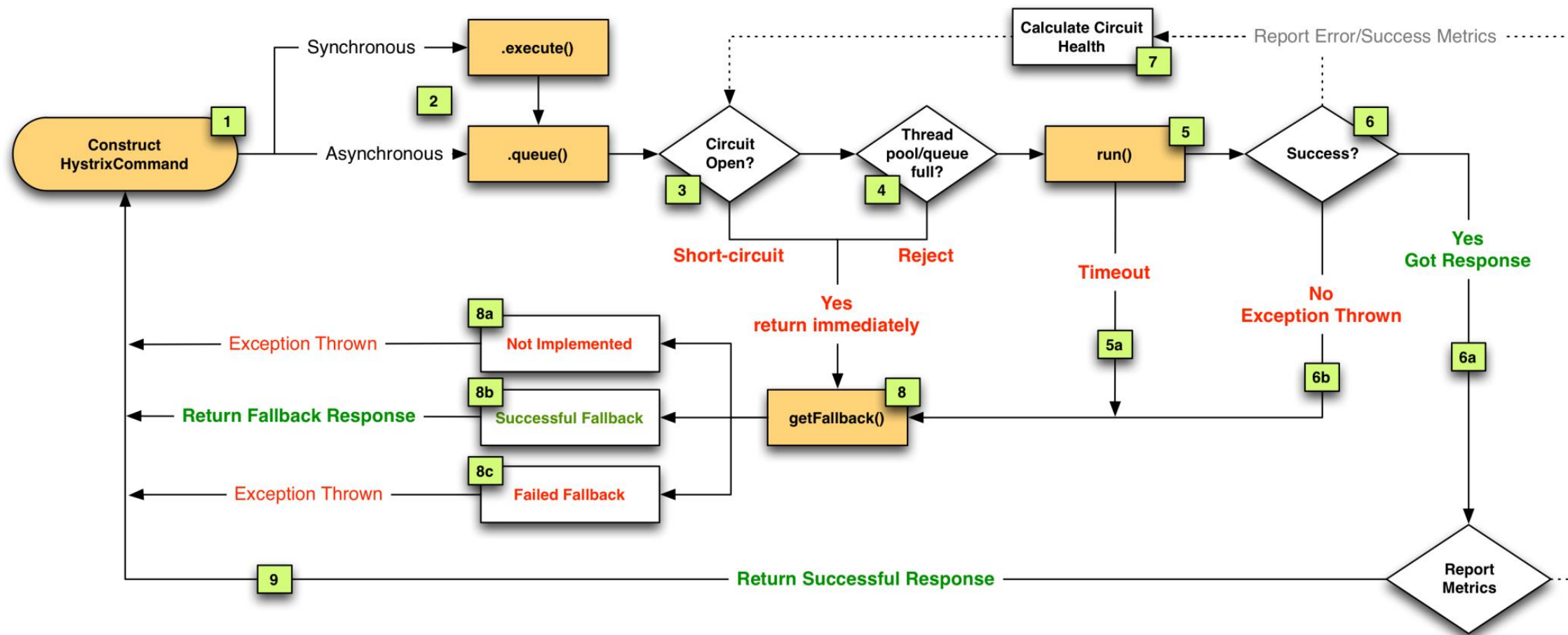
第

5

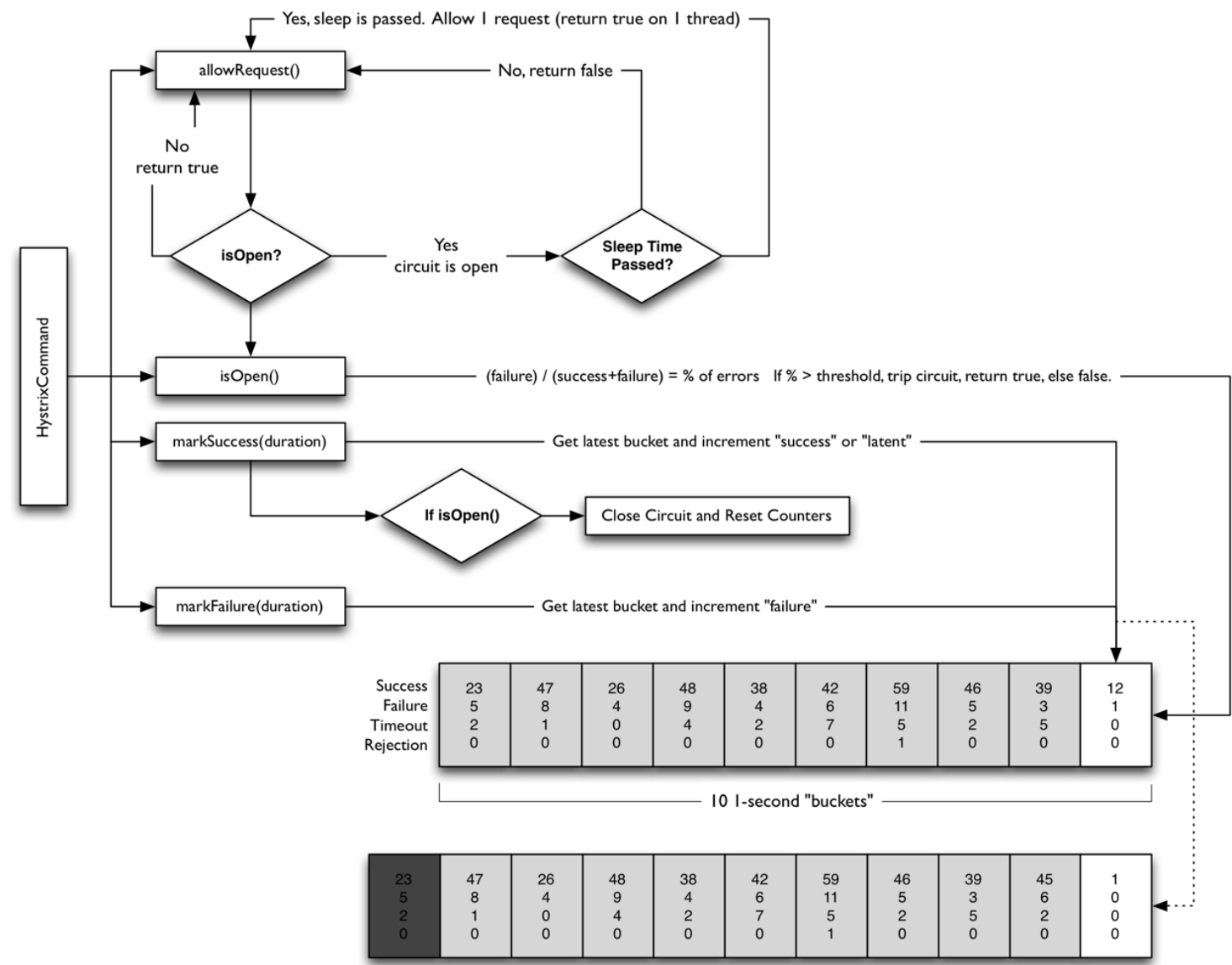
部分

# Hystrix设计原理

# Hystrix工作流程(自适应反馈机)



# 断路器内核



On "getLatestBucket" if the 1-second window is passed a new bucket is created, the rest slid over and the oldest one dropped.

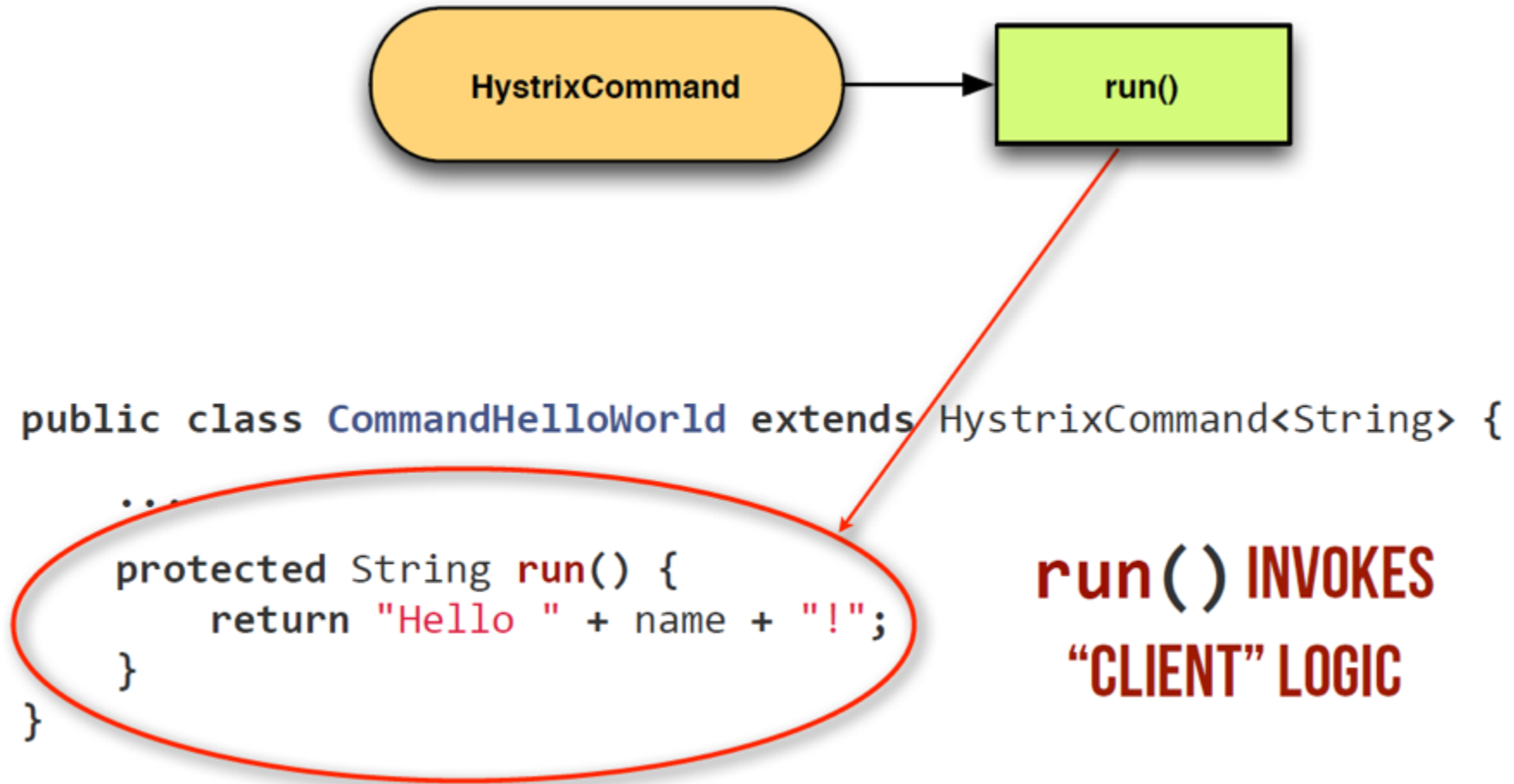
第

6

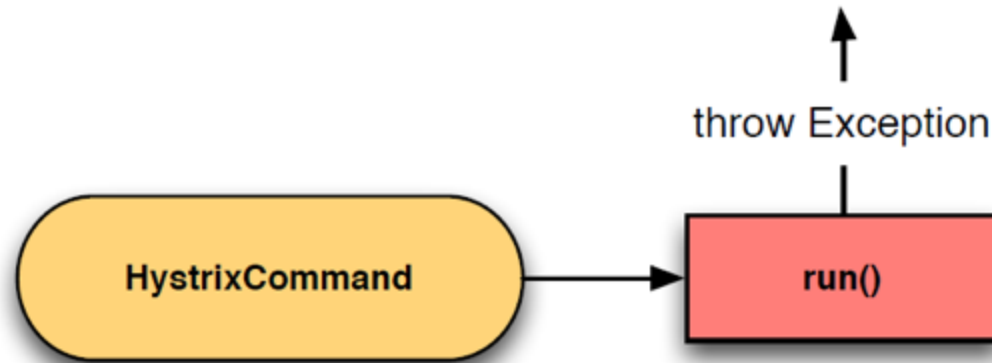
部分

# Hystrix主要概念

# Hystrix Command

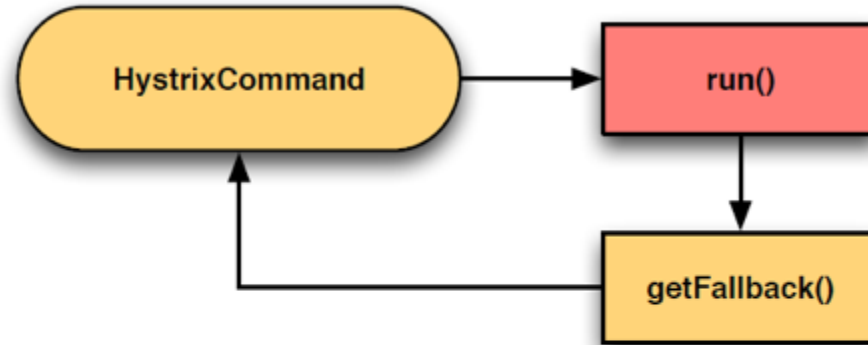


# Fail Fast



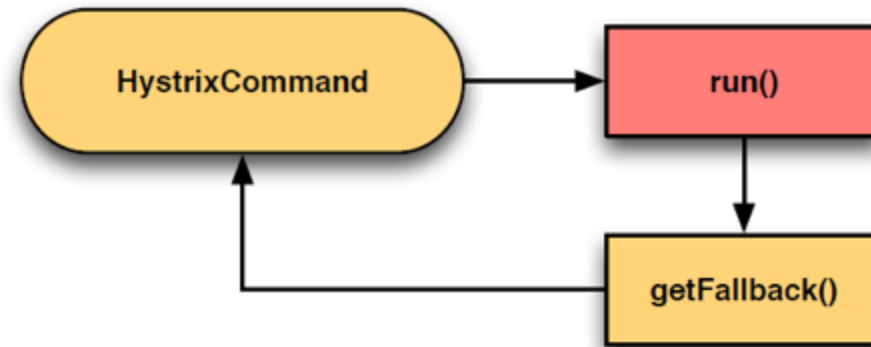


# Fail Silent



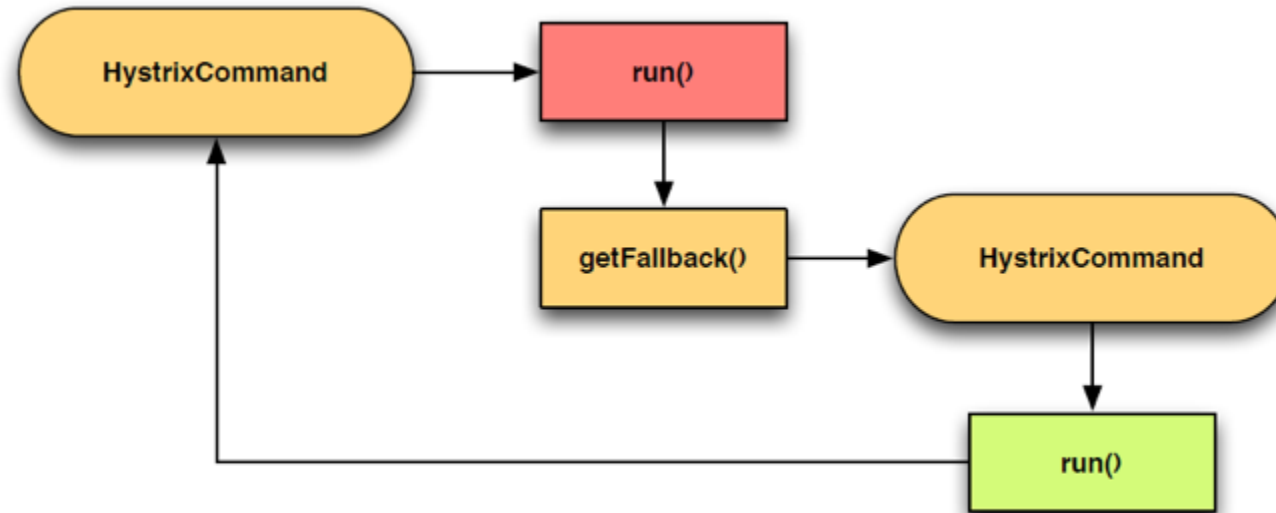
```
return null;  
return new Option<T>();  
return Collections.emptyList();  
return Collections.emptyMap();
```

# Static Fallback

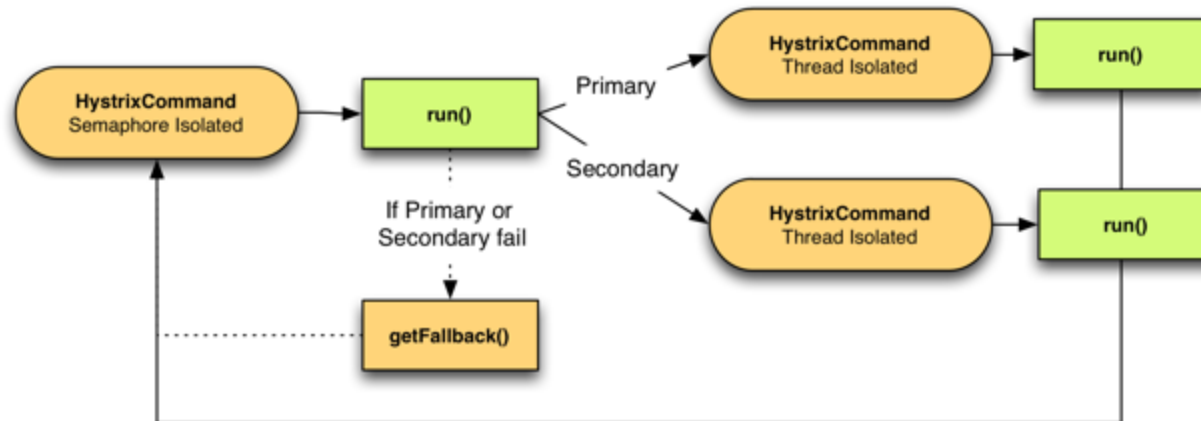


```
return true;  
return DEFAULT_OBJECT;
```

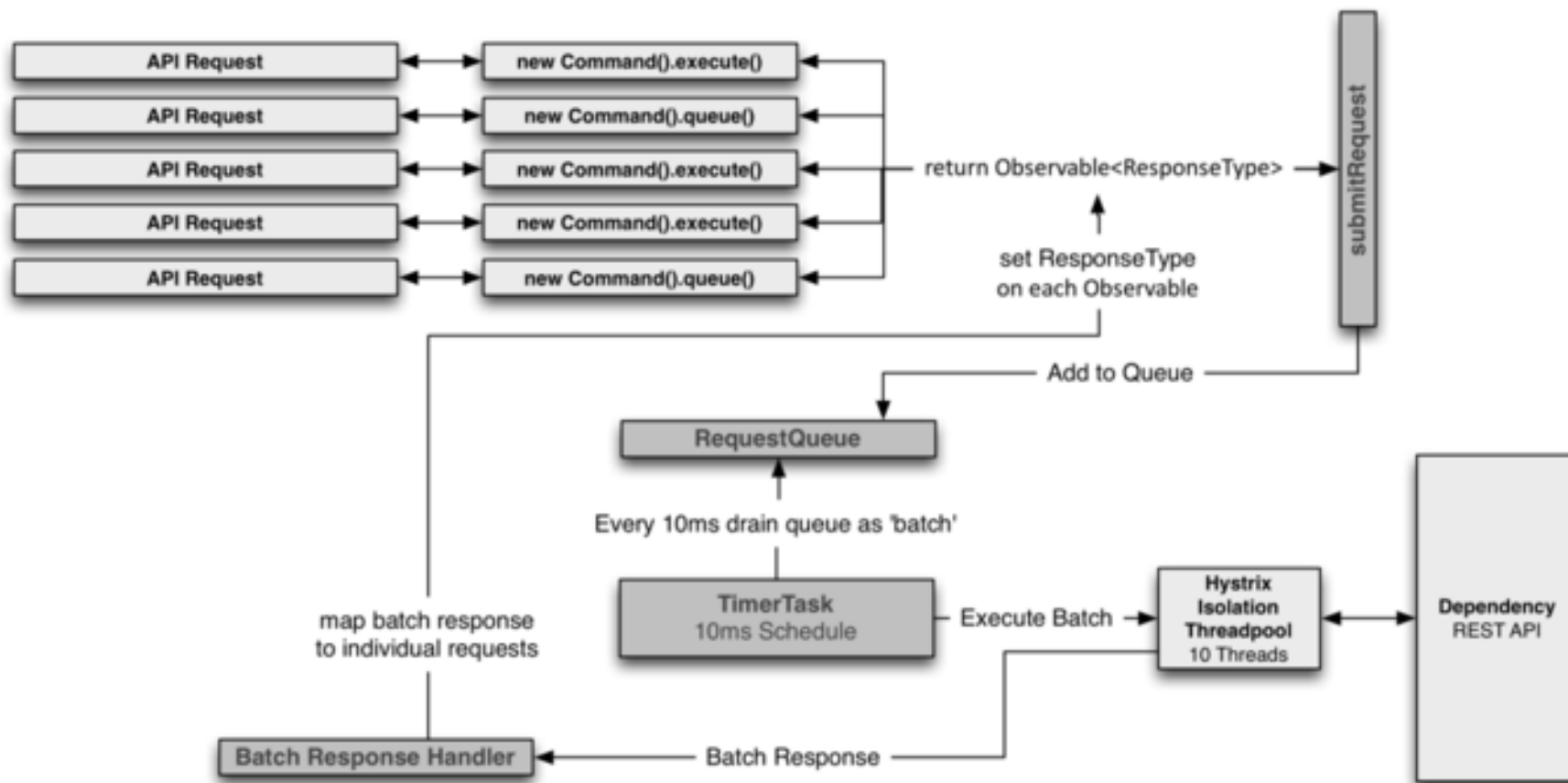
# Fallback via Network



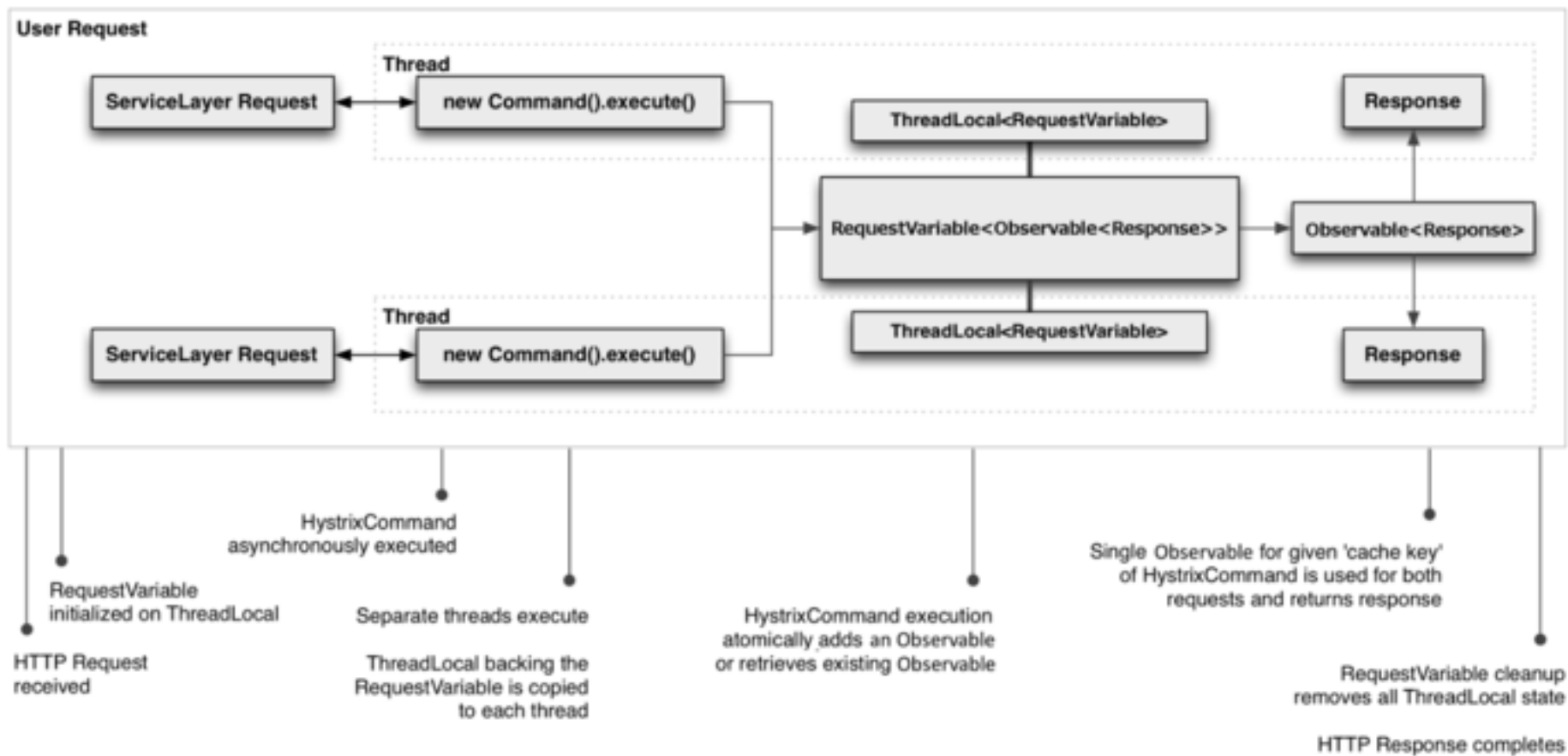
# Primary + Secondary with Fallback



# 请求合并



# 请求缓存



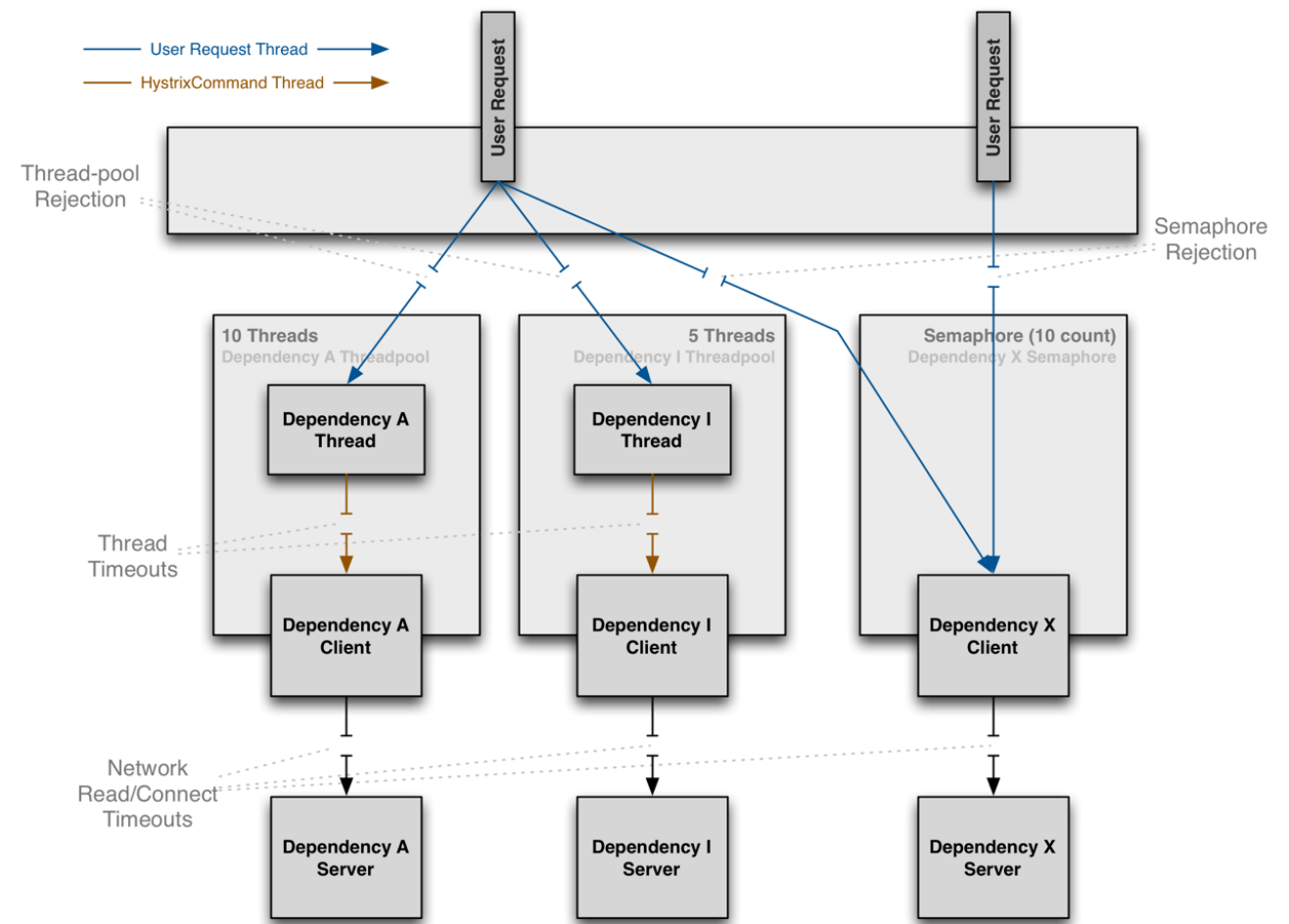
第

7

部分

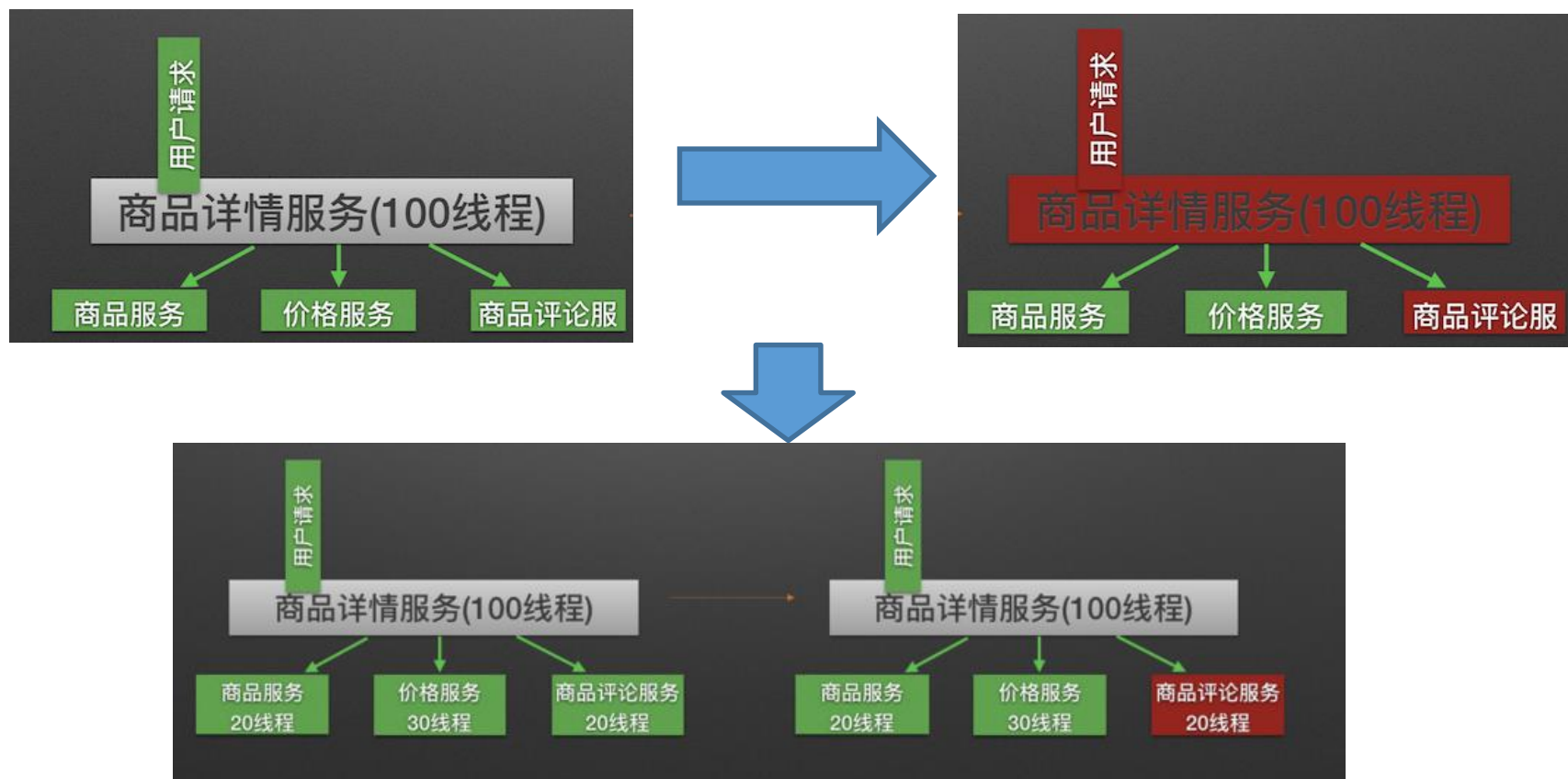
## 信号量vs线程池隔离

# 线程和信号量隔离





# 线程隔离案例



防雪崩利器：熔断器Hystrix的原理与使用

<https://segmentfault.com/a/1190000005988895>

# 线程 vs 信号量隔离

- 信号量隔离
- 优点
  - 轻量，无额外开销
- 不足
  - 不支持任务排队和主动超时
  - 不支持异步调用
- 适用
  - 受信客户
  - 高扇出(网关)
  - 高频高速调用(cache)



- 线程池隔离
- 优点
  - 支持排队和超时
  - 支持异步调用
- 不足
  - 线程调用会产生额外的开销
- 适用
  - 不受信客户
  - 有限扇出

第

8

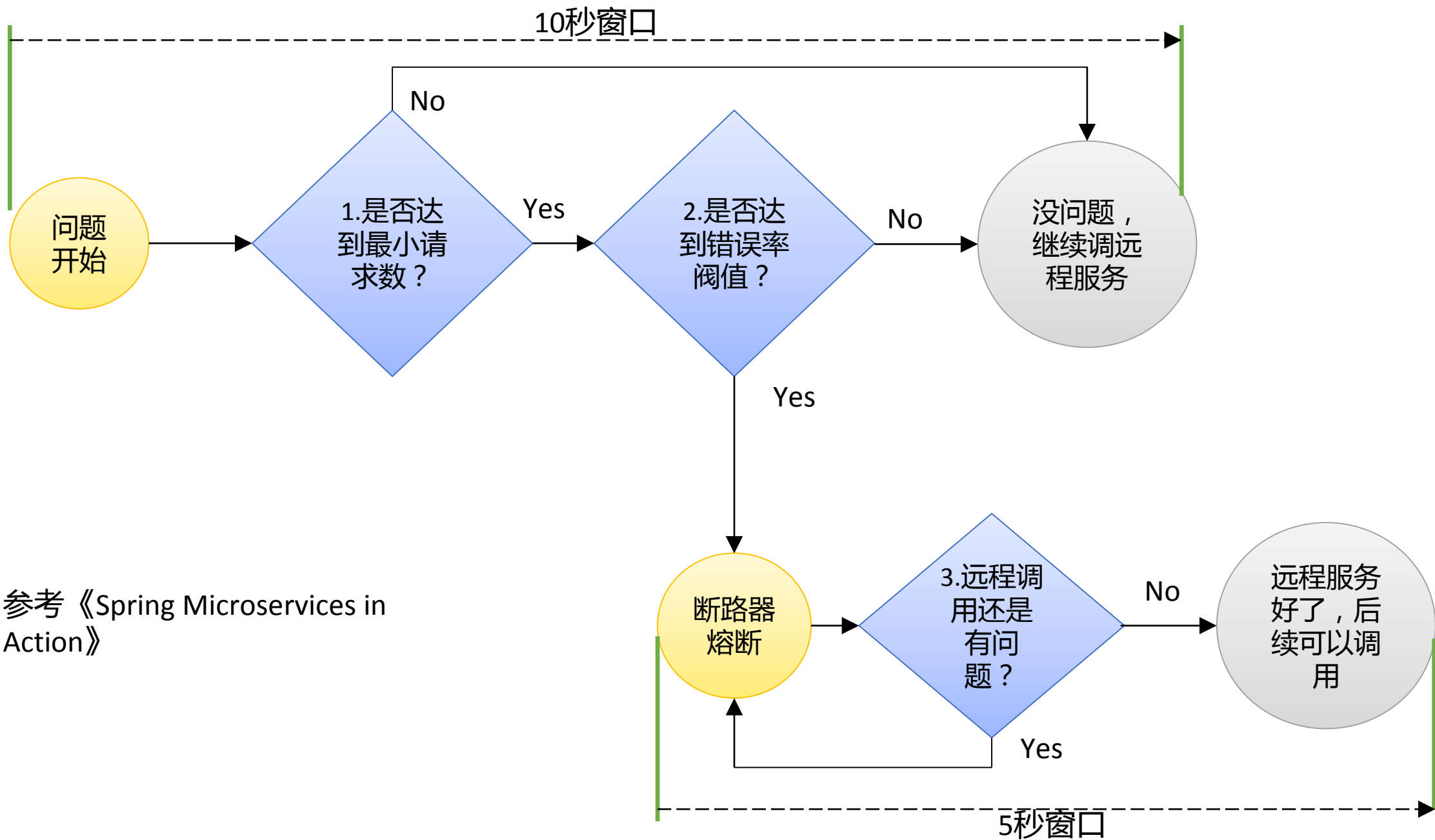
部分

## Hystrix主要配置项

# 主要配置项

配置项(前缀hystrix.command.*.)	含义
execution.isolation.strategy	线程“THREAD”或信号量“SEMAPHORE”隔离(Default: THREAD)
execution.isolation.thread.timeoutInMilliseconds	run()方法执行超时时间(Default: 1000)
execution.isolation.semaphore.maxConcurrentRequests	信号量隔离最大并发数(Default:10)
circuitBreaker.errorThresholdPercentage	熔断的错误百分比阈值(Default:50)
circuitBreaker.requestVolumeThreshold	断路器生效必须满足的流量阈值(Default:20)
circuitBreaker.sleepWindowInMilliseconds	熔断后重置断路器的时间间隔(Default:5000)
circuitBreaker.forceOpen	设true表示强制熔断器进入打开状态(Default: false)
circuitBreaker.forceClosed	设true表示强制熔断器进入关闭状态(Default: false)

配置项(前缀hystrix.threadpool.*.)	含义
coreSize	使用线程池时的最大并发请求(Default: 10)
maxQueueSize	最大LinkedBlockingQueue大小，-1表示用SynchronousQueue(Default:-1)
default.queueSizeRejectionThreshold	队列大小阈值，超过则拒绝(Default:5)



参考《Spring Microservices in Action》

第

9

部分

## Hystrix基础实验 ( Lab01 )

# 第10部分

## Hystrix模拟案例分析(Code Review)

# 模拟交易

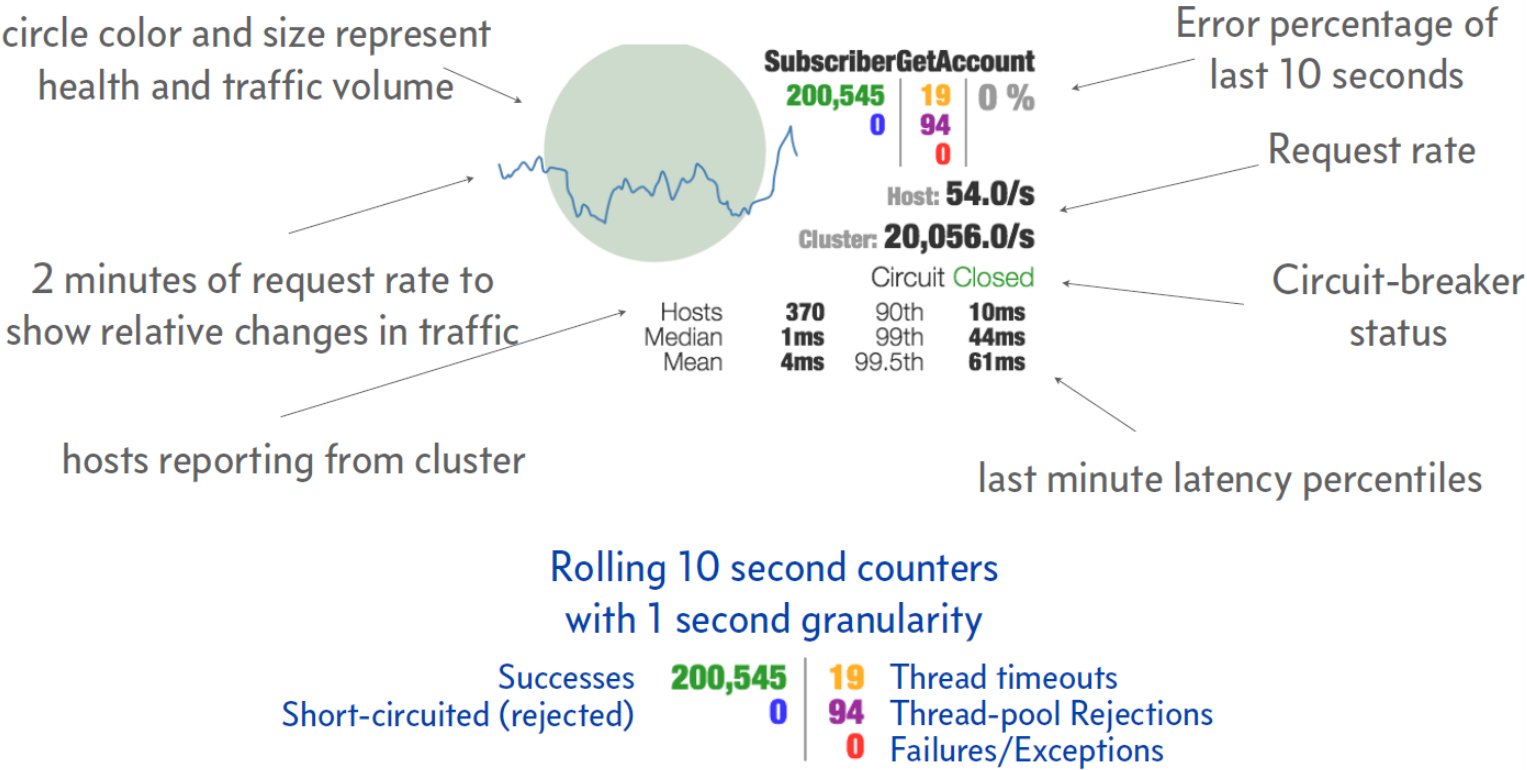




# 第11部分

## Hystrix + Dashboard实验 ( Lab02 )

# Hystrix Dashboard



标识	含义
绿色计数	成功请求数
蓝色计数	短路请求数
黄色计数	超时请求数
紫色计数	线程池满拒绝请求数
红色计数	失败异常请求数
灰色百分比	10秒内错误百分比
Host	单机请求率
Cluster	集群请求率
Circuit	断路器状态
Hosts	主机实例数量
Median	请求耗时中位数
Mean	请求耗时平均值
90/99/99.5 <sup>th</sup>	90/99/99.5百分位耗时
Rolling 10 sec counters	10秒钟滚动窗口统计
1 sec granularity	精度1秒一个桶

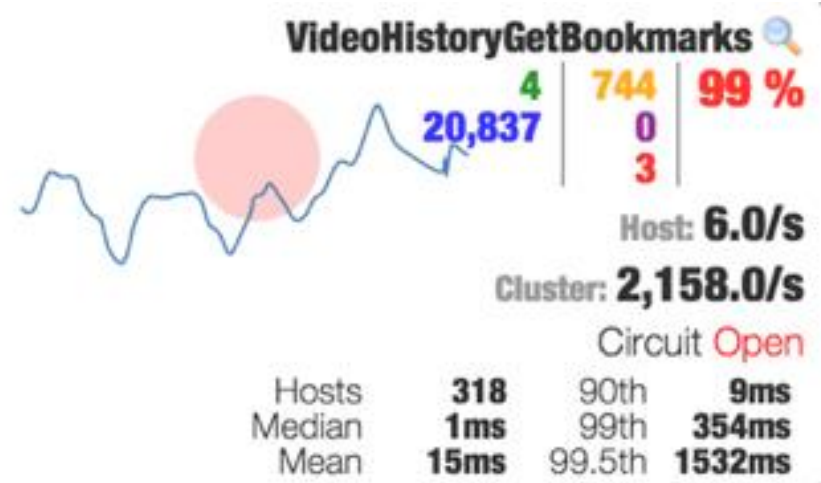
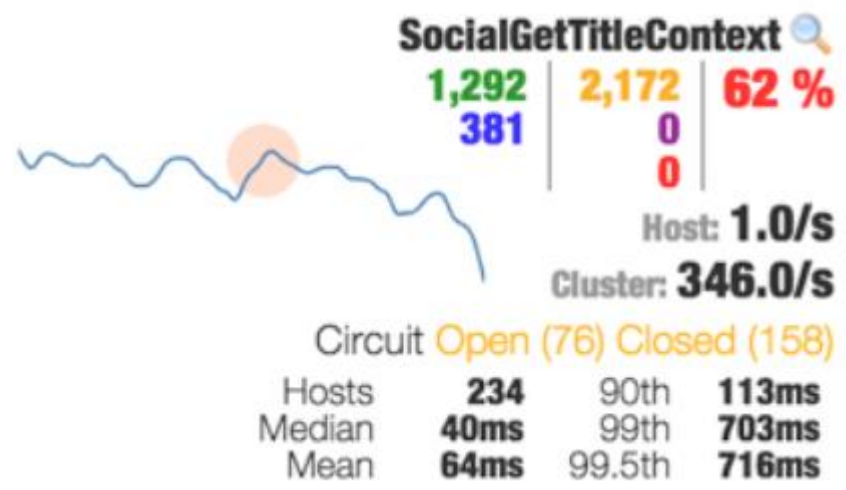
# 滚桶式统计

Success	23	47	26	48	38	42	59	46	39	12
Timeout	5	8	4	9	4	6	11	5	3	1
Failure	2	1	0	4	2	7	5	2	5	0
Rejection	0	0	0	0	0	0	1	0	0	0

23	47	26	48	38	42	59	46	39	45	1
5	8	4	9	4	6	11	5	3	6	0
2	1	0	4	2	7	5	2	5	2	0
0	0	0	0	0	0	1	0	0	0	0

10秒钟滚动窗口，精度1秒钟一个桶

# 熔断



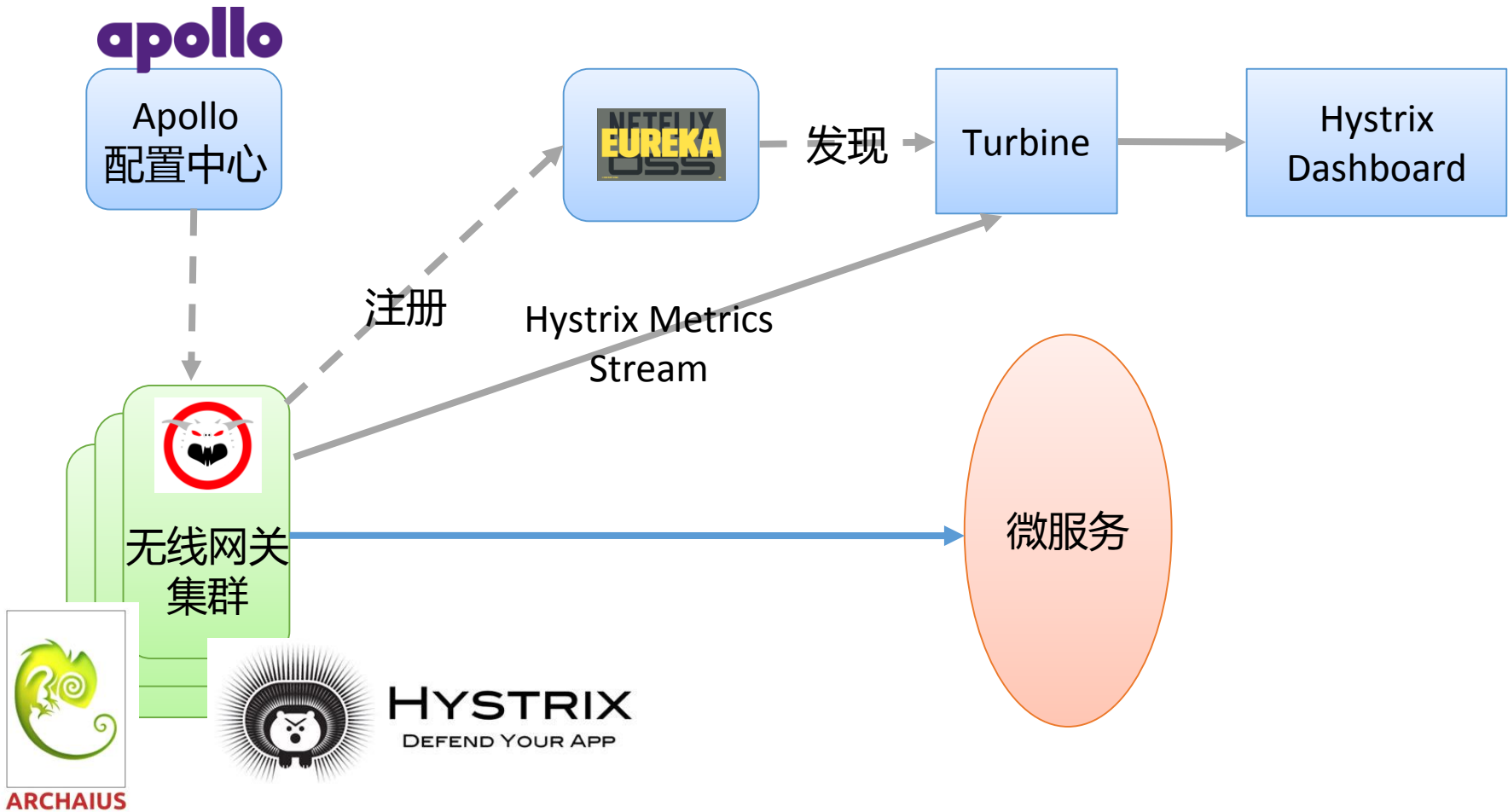
第

12

部分

网关集成Hystrix(Code Review)

# 网关和Hystrix参考部署



第

13

部分

Spring Cloud Hystrix实验(Lab03)

第

14

部分

## Netflix Turbine简介



# 流聚合

<http://<host:port>/turbine.stream>



Hystrix  
Dashboard

Turbine Server



Hystrix应用

Hystrix应用

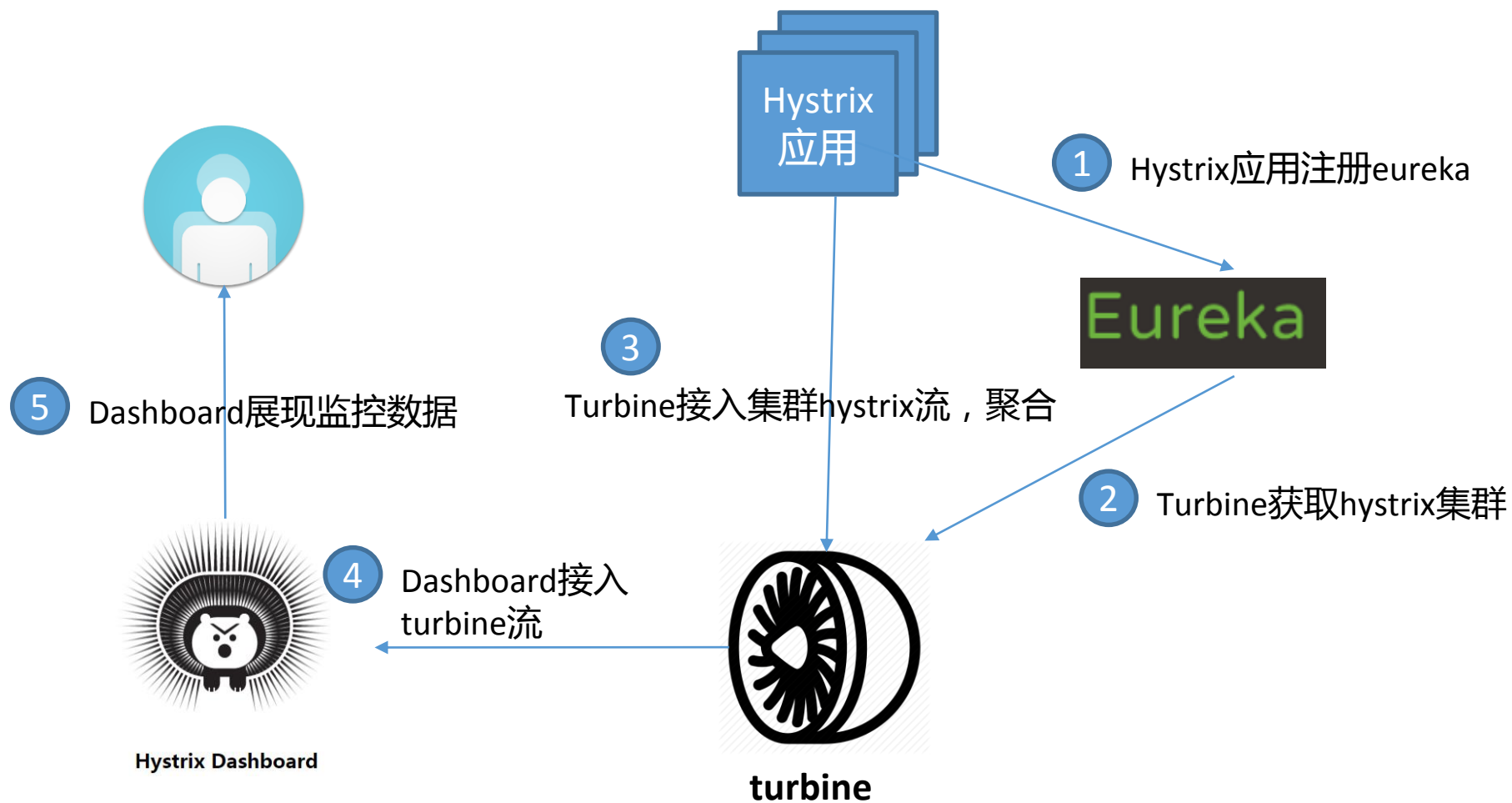
Hystrix应用

Hystrix应用

Hystrix应用

<http://<host:port>/hystrix.stream>

# 对接Eureka



第

15

部分

**Hystrix生产最佳实践**

# 最佳实践

- 网关集中埋点，覆盖大部分场景
- 尽量框架集中埋点，客户端为主
- 配置对接Apollo，根据实际使用调整阈值
- 信号量vs线程池场景
  - 信号量：网关，缓存
  - 线程池场景：服务间调用客户端，数据库访问，第三方访问
- 线程池大小经验值
  - $30 \text{ rps} \times 0.2 \text{ sec} = 6 + \text{breathing room} = 10 \text{ threads}$
  - Thread-pool Queue size : 5 ~ 10
- 部署
  - Hystrix Dashboard大盘（无线/H5/第三方网关）
  - 共享Hystrix Dashboard/Turbine服务器
  - 熔断告警
- Spring Cloud Hystrix标注



第

16

部分

参考资源和后续课程预览

# 参考文章和ppt

- Netflix Hystrix
  - <https://github.com/Netflix/Hystrix>
- 防雪崩利器：熔断器Hystrix的原理与使用
  - <https://segmentfault.com/a/1190000005988895>
- Application Resilience Engineering and Operations at Netflix with Hystrix
  - <https://speakerdeck.com/benjchristensen/application-resilience-engineering-and-operations-at-netflix-with-hystrix-javaone-2013>

# 其它开源容错限流产品

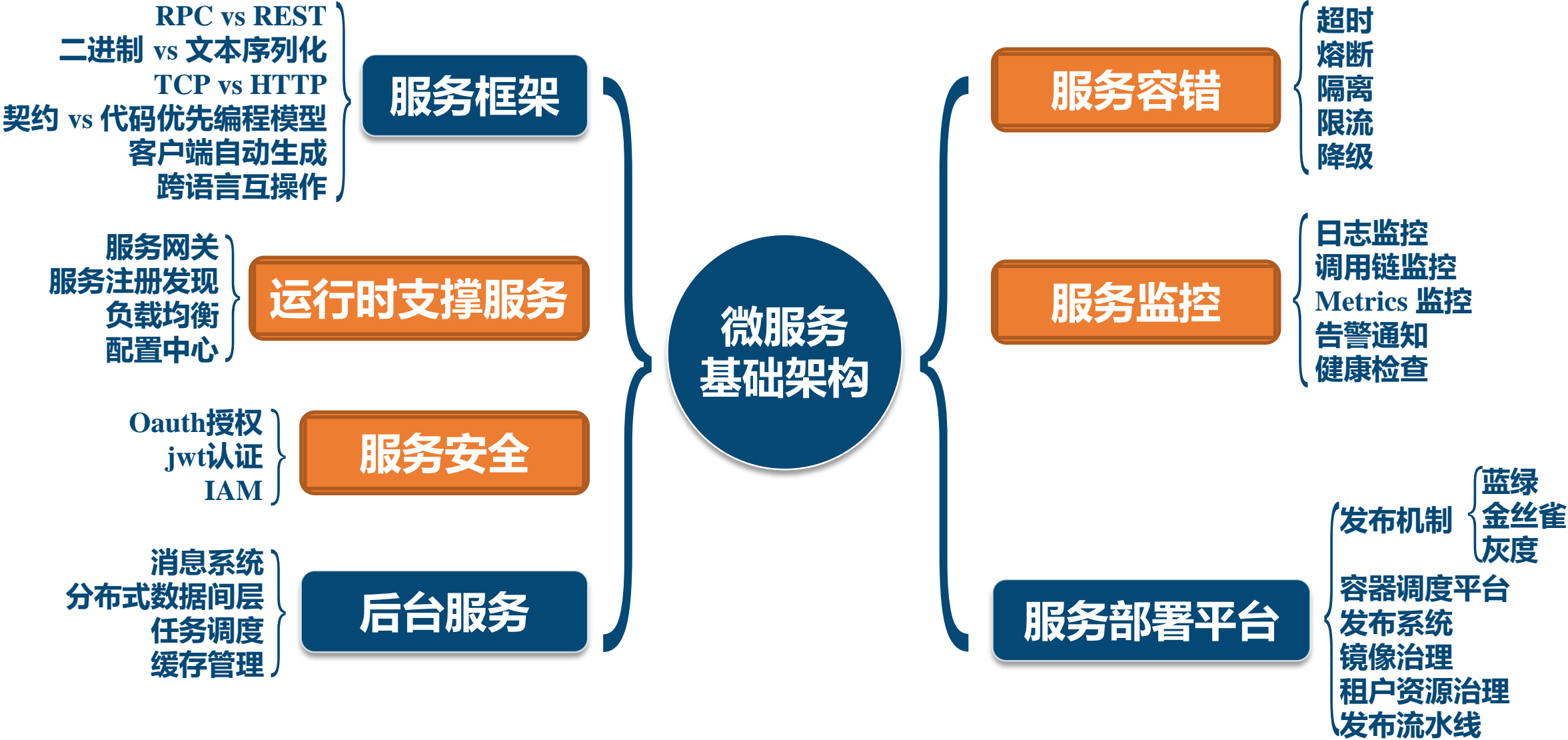
- Alibaba Sentinel(Java , 1.4k stars)
  - <https://github.com/alibaba/Sentinel>
- Polly(C# , 4.6k stars)
  - <https://github.com/App-vNext/Polly>
- Hystrix-go(golang, 1.2k stars)
  - <https://github.com/afex/hystrix-go>
- Failsafe(Java , 2.2k stars)
  - <https://github.com/jhalterman/failsafe>
- Resilience4j(Java , 947k stars)
  - <https://github.com/resilience4j/resilience4j>



Sentinel

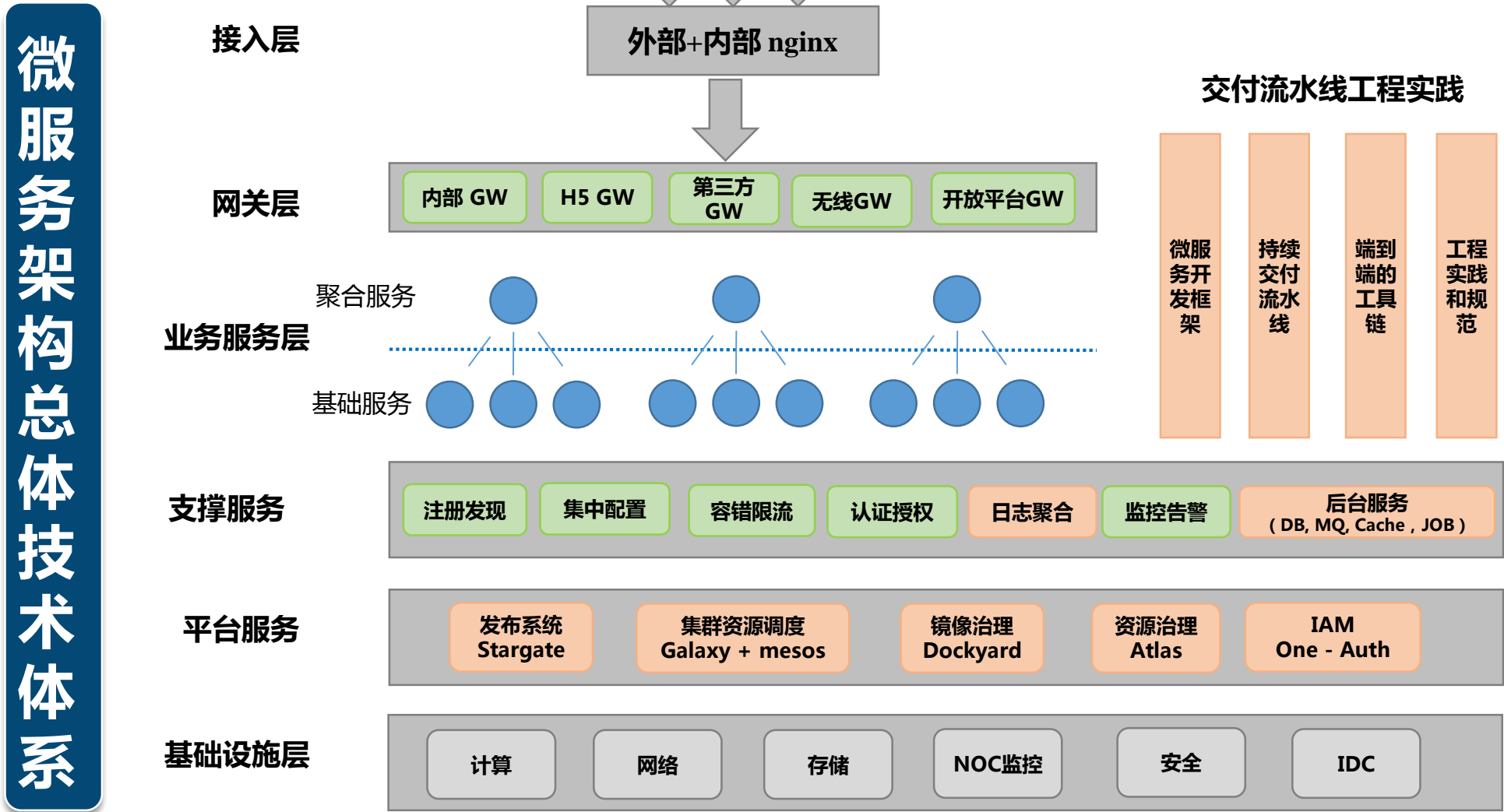


# 后续课程预览~2018课程模块





# 后续课程预览~技术体系



# 架构和技术栈预览

