

# 效果展示 – 基于接口调用统计的后向流控

## 流控效果精准

通过流控增强对业务系统的把控

- 压测确定容量上限和消费速率
- 常态配置流控策略



设置某业务的调用上限为2100w/min

# 目录

- 背景
- 设计
- 实现
- 最佳实践
- 效果展示
- 总结



# 总结

- 引入Paxos能让事情变简单
  - 保证数据可靠性
  - 杜绝乱序、重复消费
- 良好设计解决其它核心问题
  - 高性能：Plog as queue、Group Commit
  - 高可用：Store接入均衡、轻量级租约维护、Consumer负载均衡
- 实现投产
  - 业务配合、热点处理、屏蔽策略、流控策略.....
- 展望：Paxos将会成为一种开发模式

# PhxPaxos组件介绍

PhxPaxos是微信团队开源的Paxos类库

- 高性能
- 功能完善
- 接口方便易用

项目地址

- <https://github.com/tencent-wechat/phxpaxos>

# 腾讯云商用队列介绍

## CKafka

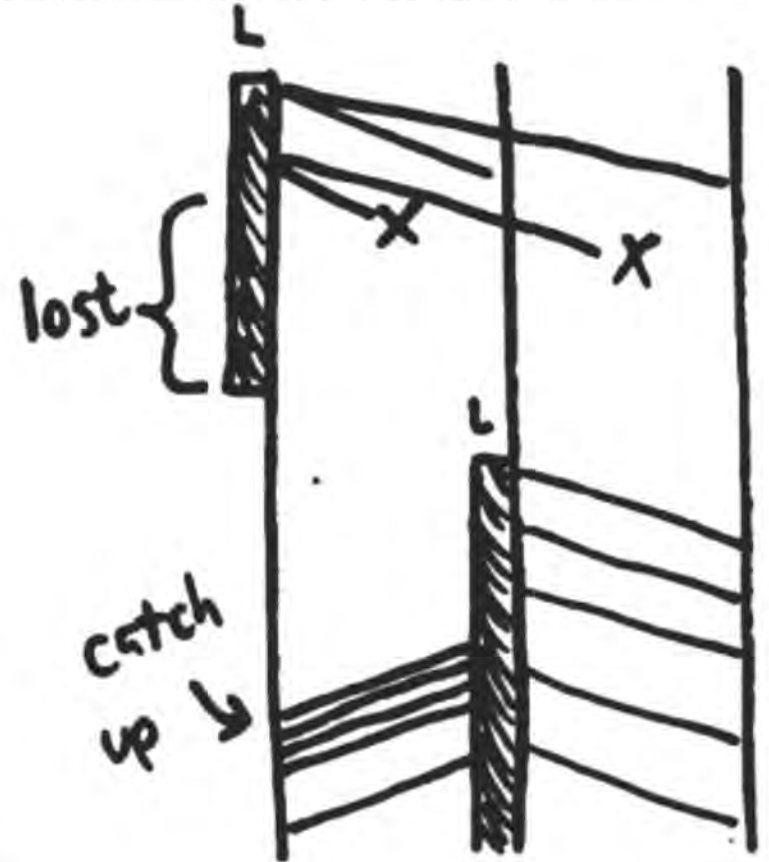
- 场景：大数据分析，日志采集
- 一致性协议：ISR

## CMQ


- 场景：金融级高可靠强一致
- 一致性协议：Raft

# About Kafka tolerate $f-1$ failures with $f$ nodes

- ISR为空可能导致数据丢失
  - <https://aphyr.com/posts/293-jepsen-kafka>
- 进一步说明经逻辑证明的一致性模型才是可靠的



# AGENDA



PayPal & PayPal Risk (Platform)



Risk DAL Service Challenge



Async Solution




Async Future Plan







# AGENDA



PayPal & PayPal Risk (Platform)



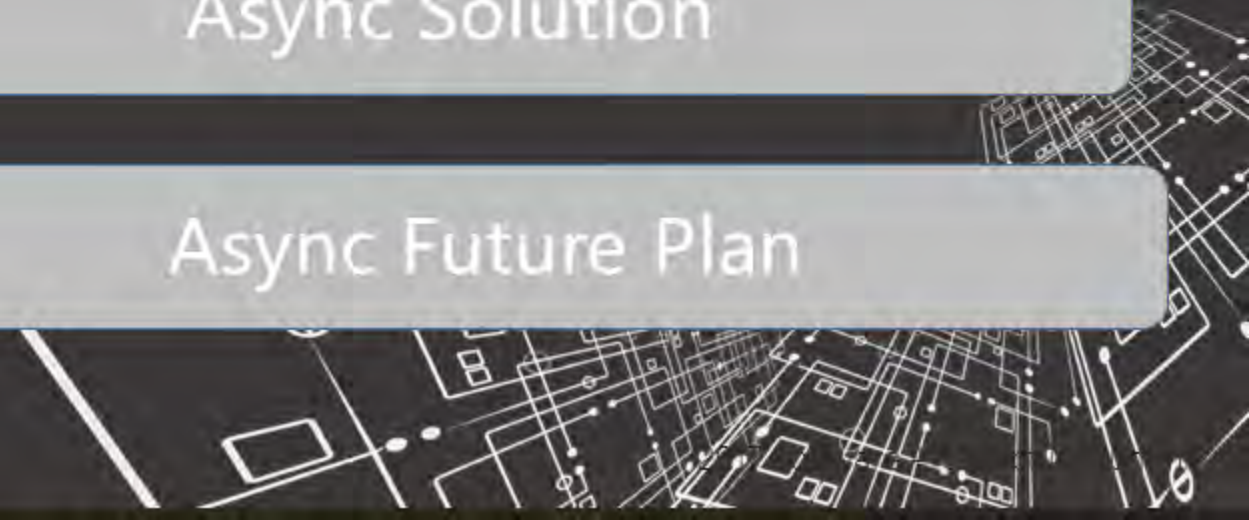
Risk DAL Service Challenge



Async Solution



Async Future Plan

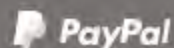




# PayPal Overview



The power of  
our platform



Our technology transformation enables us to:

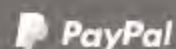
- Process payments at tremendous scale (**200**+ countries & **25** currencies supported)
- Accelerate the innovation of new products
- Engage world-class developers & technologists



# PayPal Risk KPI



The power of  
our platform



Our technology transformation enables us to:

- Process payments at tremendous scale (**200**+ countries & **25** currencies supported)
- Accelerate the innovation of new products
- Engage world-class developers & technologists

# Requirement for Risk Platform

## Accuracy vs Latency

Need more Variables to Make Accurate Decision...

... But that impact decision Speed

Minimize Data Load Latency ...

... But that impact decision Accuracy

## Low Latency + Hardware Investment Vs Large Throughput

Less Operation Cost & Better Latency Under low Traffic ...

... But it fails to scale & support business

Huge Traffic Growth & Convergence

... But more hardware & Higher Latency



# PayPal Risk Platform Architecture

Online

Read Path

Gateway Service

Decision Service

Model + Variable Computation Service

DAL Service

Real-time Compute Data

Write Path

Variable Rollup Service

Offline Generated Data

Logging System/ ETL

Offline

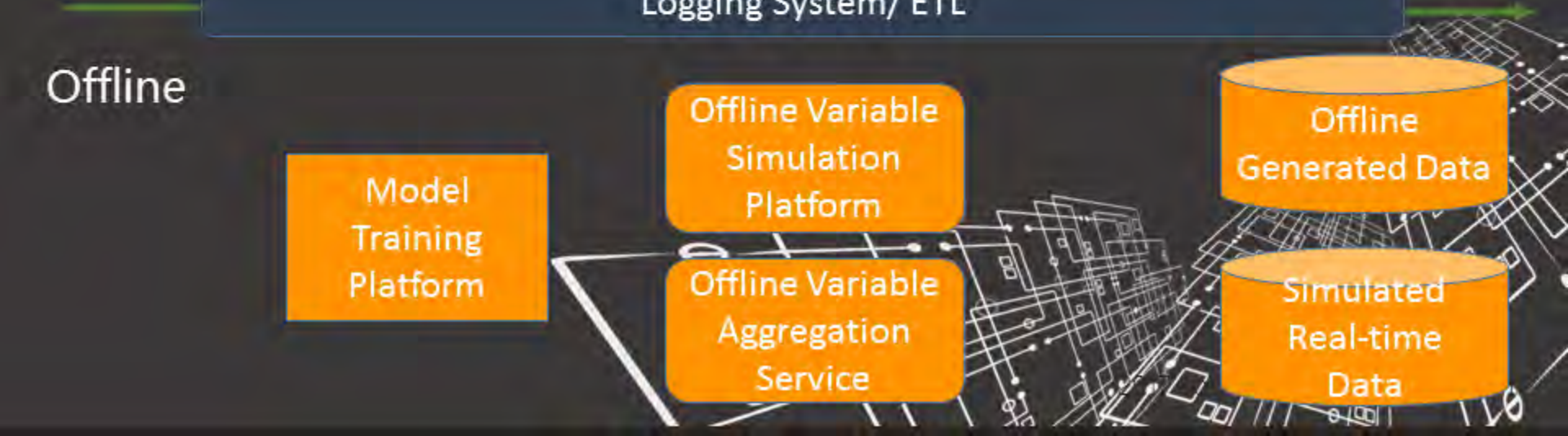
Model Training Platform

Offline Variable Simulation Platform

Offline Variable Aggregation Service

Offline Generated Data

Simulated Real-time Data



# PayPal Risk Platform Architecture

Online

Read Path

Gateway Service

Decision Service

Model + Variable Computation Service

DAL Service

Offline Generated Data

Real-time Compute Data

Write Path

Variable Aggregation Service

Logging System/ ETL

Offline

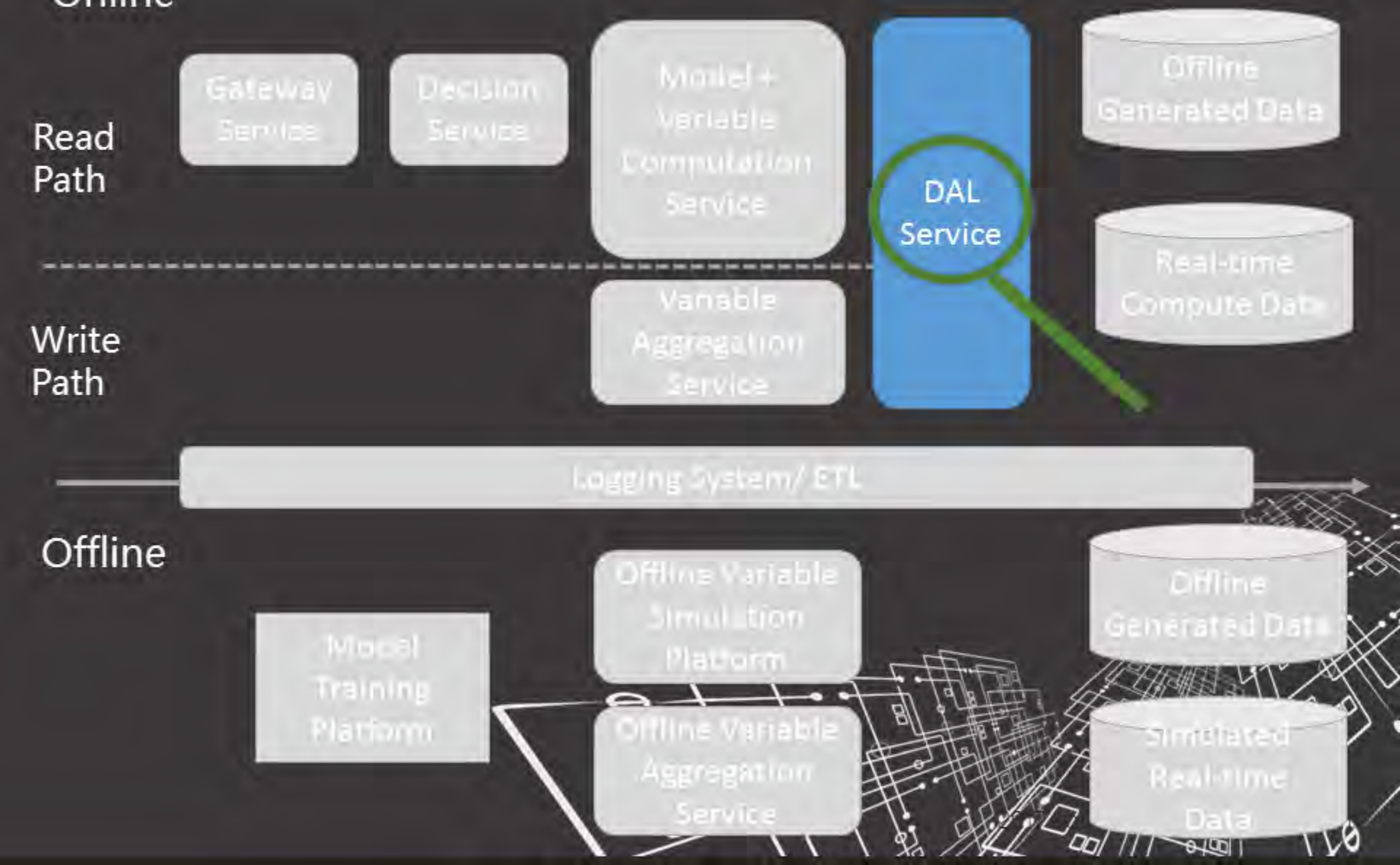
Model Training Platform

Offline Variable Simulation Platform

Offline Variable Aggregation Service

Offline Generated Data

Simulated Real-time Data




# AGENDA




PayPal & PayPal Risk (Platform)



Risk DAL Service Challenge



Async Solution



Async Future Plan





# DAL Service Ultimate Questions



<100ms P99.99 Latency ??



For single instance, 20k-30k Peak TPS ??



99.99% Availability-To-Business??

## JVM-Based High Performance & ATB DAL Service

# DAL Service Technical Challenges

Req

## Customer Requirement

- Adopt New Use Case
- Access behavior Differentiate per Colo
- Flexibility & Fast-evolving Use Case
  - Replication
  - Traffic Strategy

## Budget Cost

- Align with traffic, Hardware investment Exponential Increase

Value

Cost

## Performance Issue


- P99 Latency Significantly differentiate Avg latency
- Too Many Latency Spike under Traffic
- Storage Cluster Unavailability Impact Latency

## Operational Cost


- Maintain too many Client with multiple versions
- Too Frequent Release tie to Biz Case
- Standby Storage Cluster switch-over

Tech


# AGENDA




PayPal & PayPal Risk (Platform)



Risk DAL Service Challenge



Async Solution

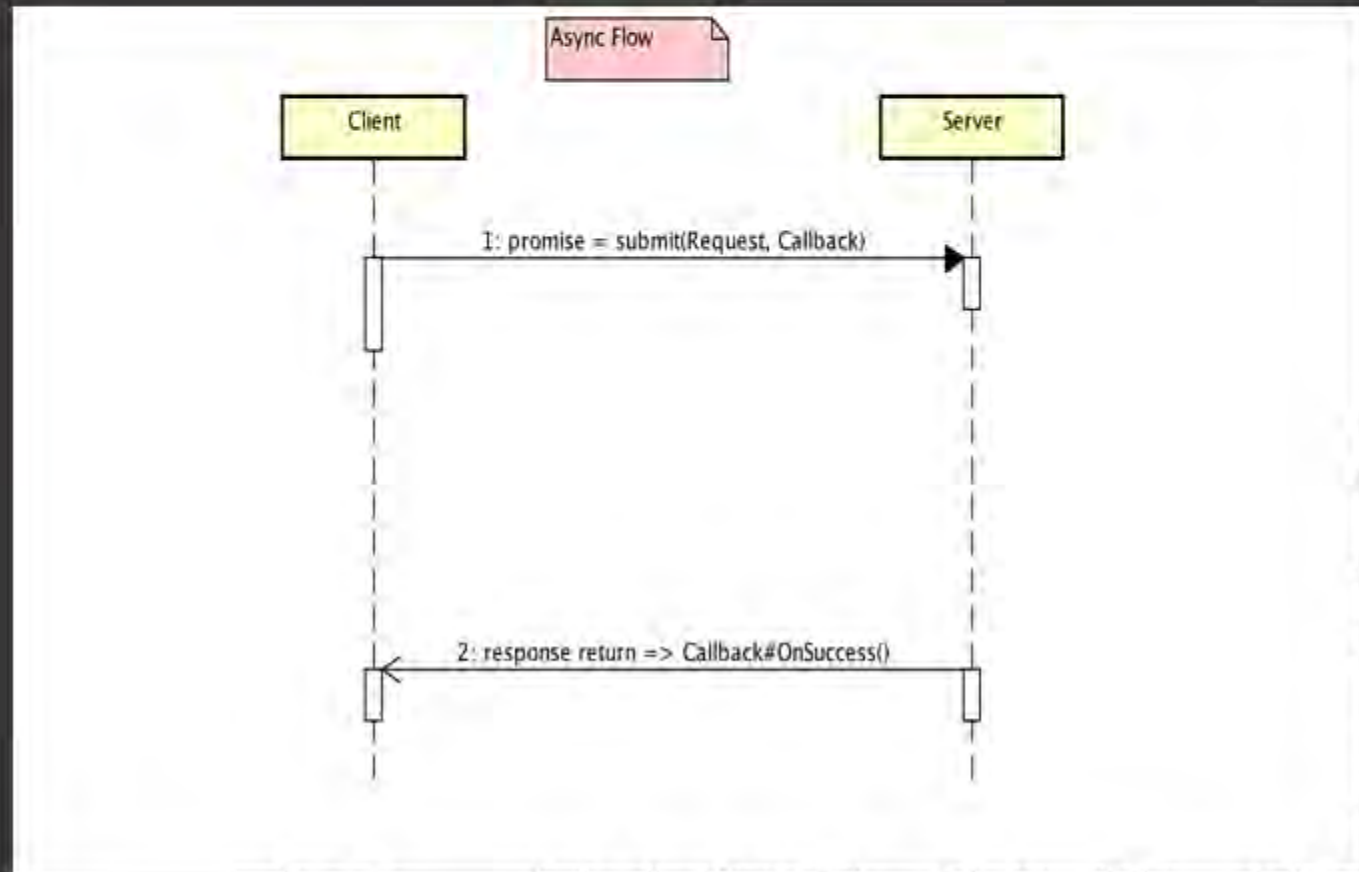


Async Future Plan



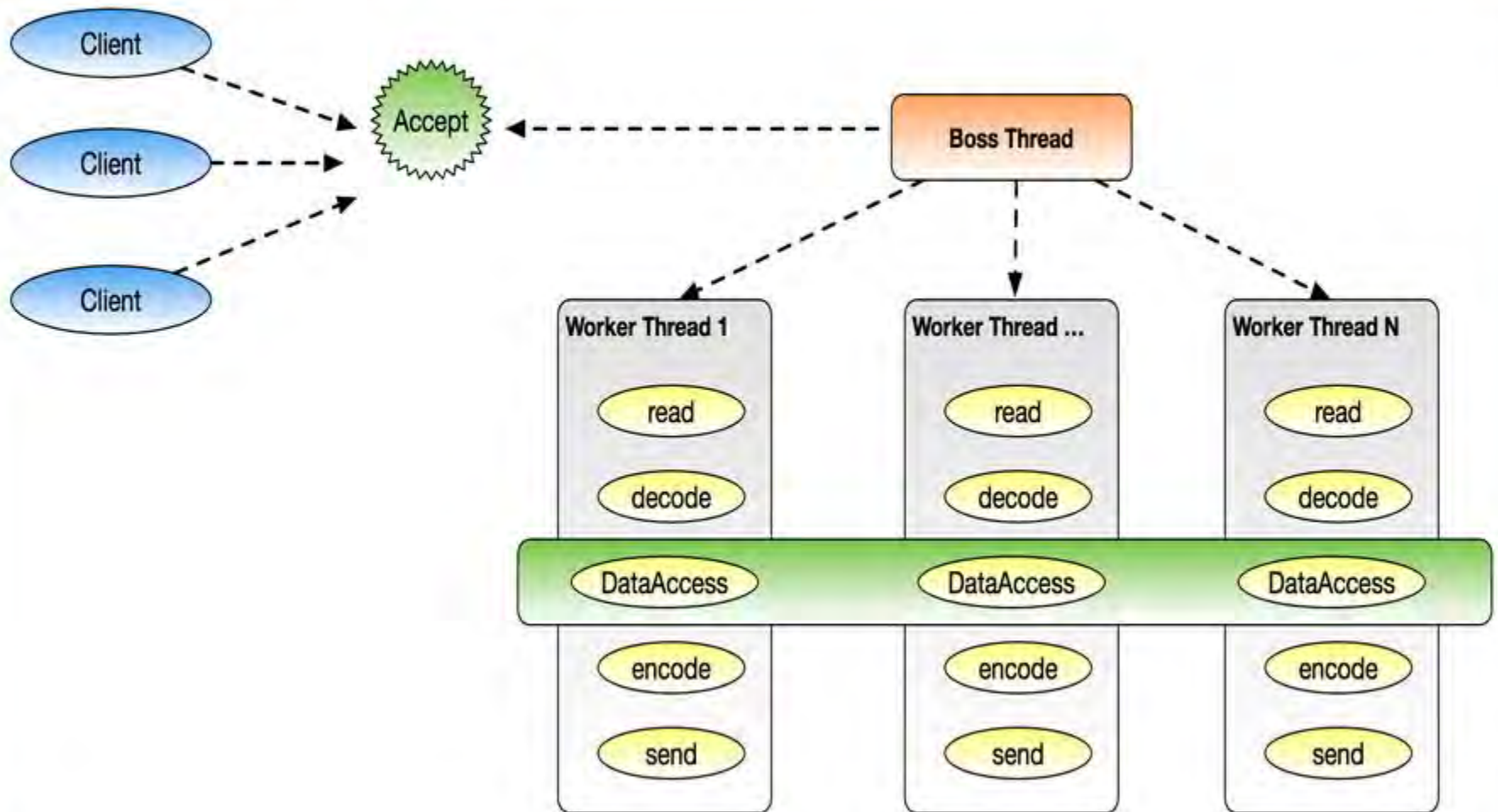
# Async Original Benefit

- More Efficient Thread Scheduling
  - Non-blocking Call
  - Event-Driven Callback
- Less Context Switch
- Fault Isolation





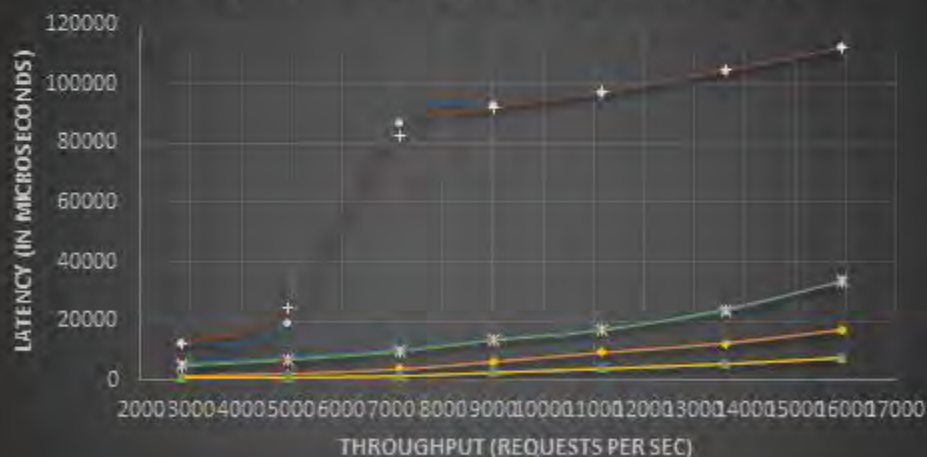
# Reactor Pattern Threading Model



# Async DAL Service KPI Comparison

- Low Latency
  - ~10-35% Reduction (Average/P99)

**E2E Client-Service-Aerospike  
Benchmark: Read 50% Write 50%  
Latency vs. Throughput (4-core VM)**



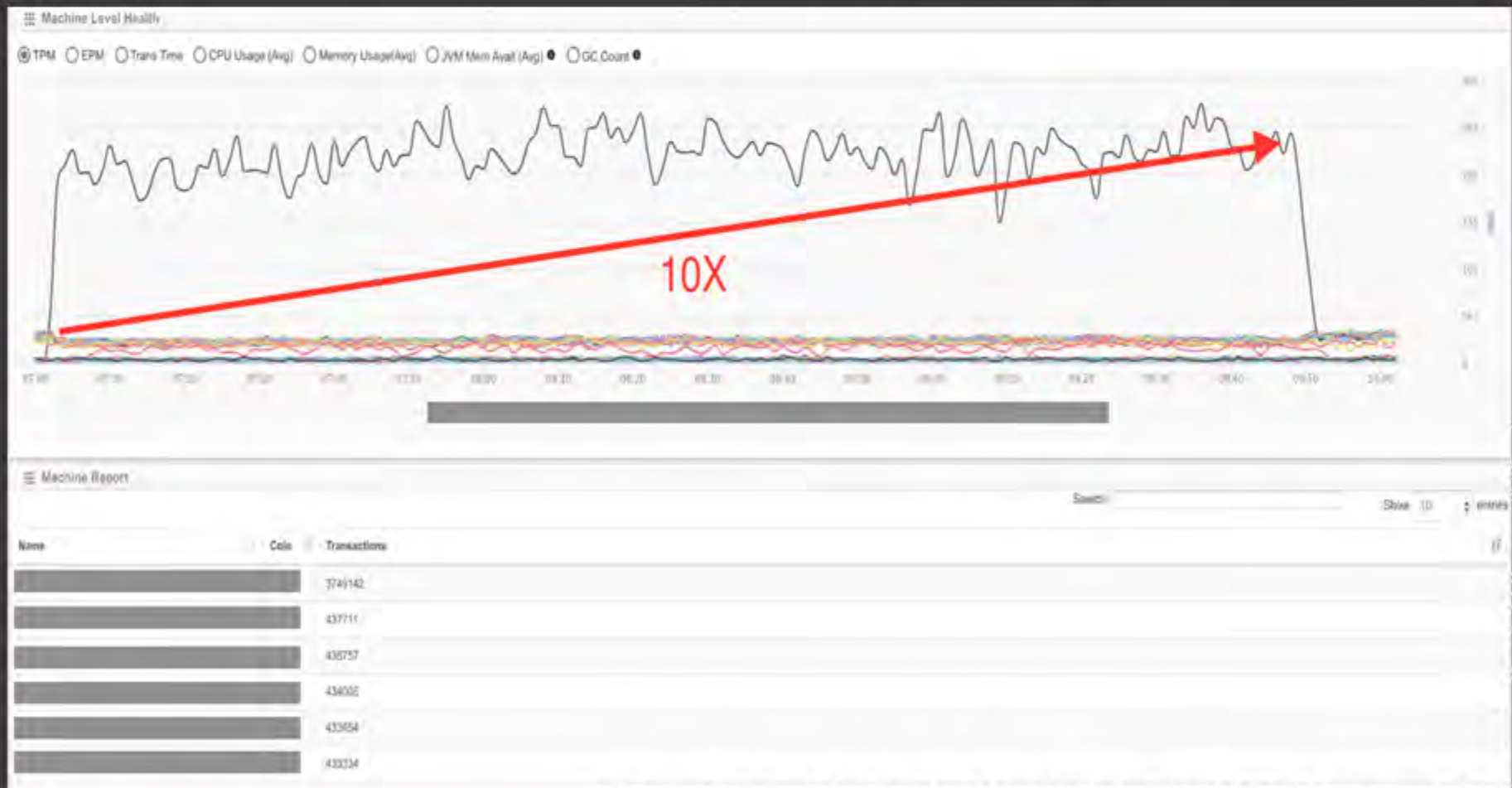
99thPercentileLatency\_update 99thPercentileLatency\_read  
AvgLatency\_read AvgLatency\_update  
99.9thPercentileLatency\_read 99.9thPercentileLatency\_update  
99.99thPercentileLatency\_read 99.99thPercentileLatency\_update





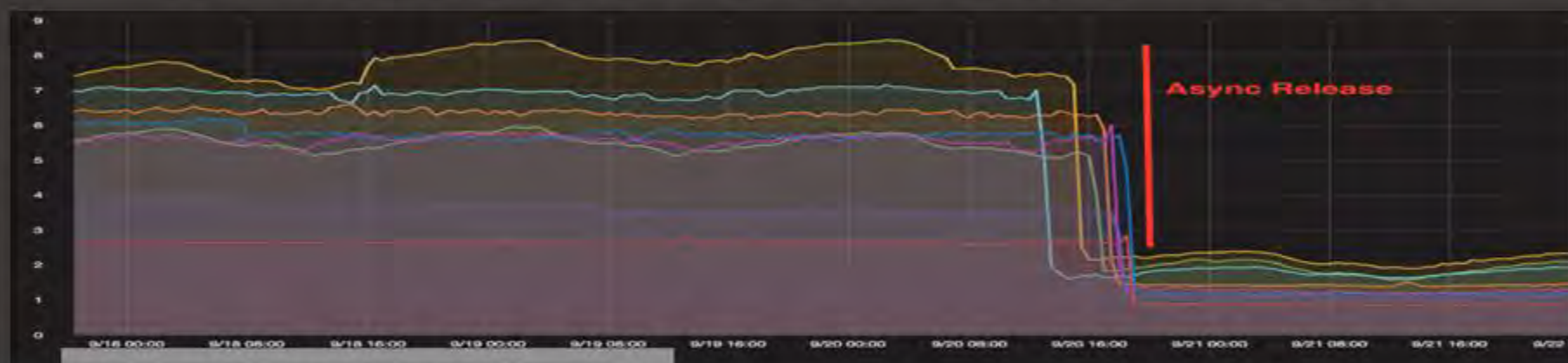
# Async DAL Service KPI Comparison – Cont.

- High Throughput
  - 3-10X Increase (Single Instance Comparison)



# Async DAL Service KPI Comparison – Cont.

- Less CPU Usage
  - 50% CPU Usage Reduction
  - 66%+ Reduction for Context Switch & System Interrupts



Async vs Sync System Interruption Comparison

Async vs Sync Context Switch Comparison

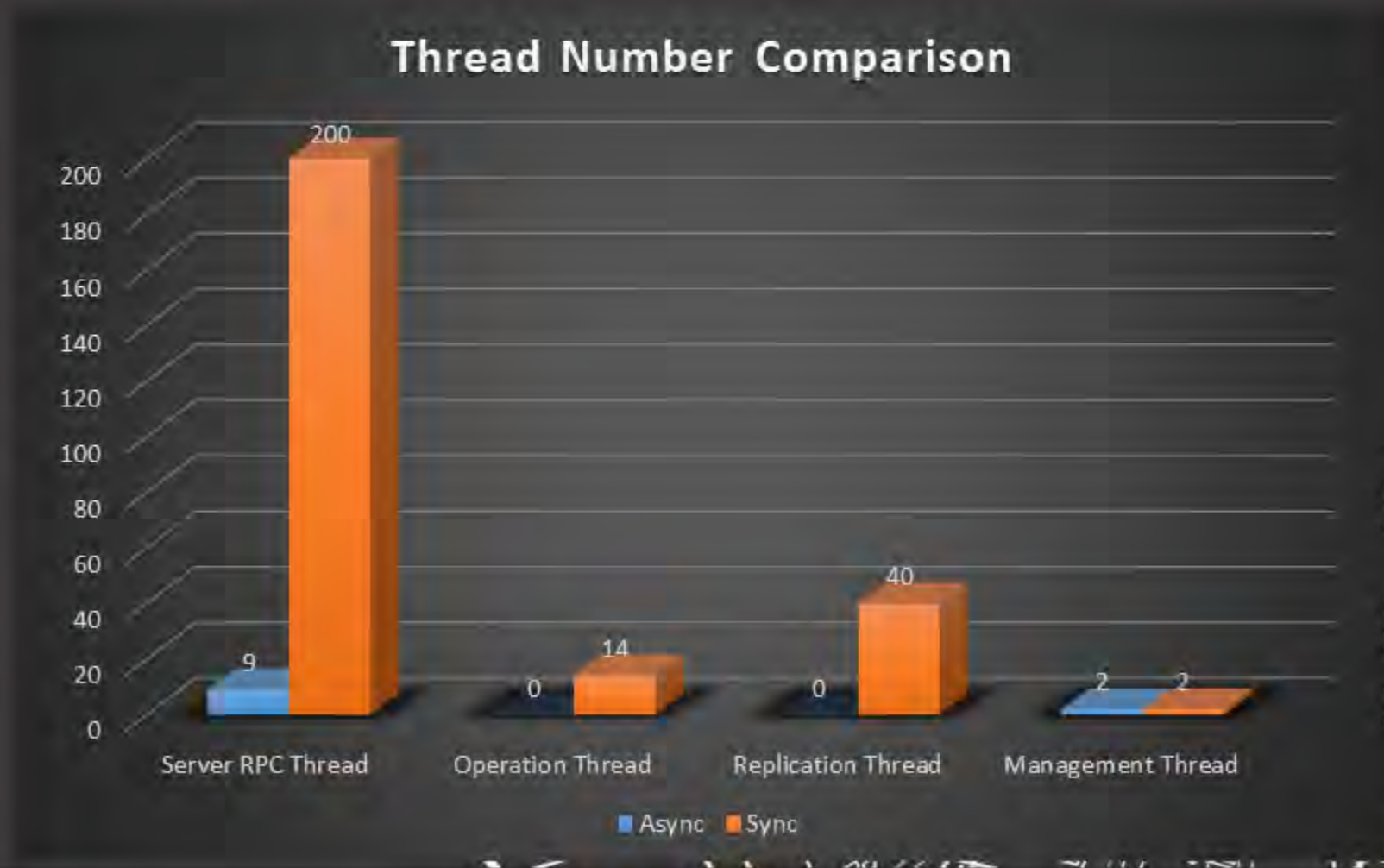
— Sync Interruption — Async Interruption

— Sync Context Switch — Async Context Switch



# Async DAL Service KPI Comparison – Cont.

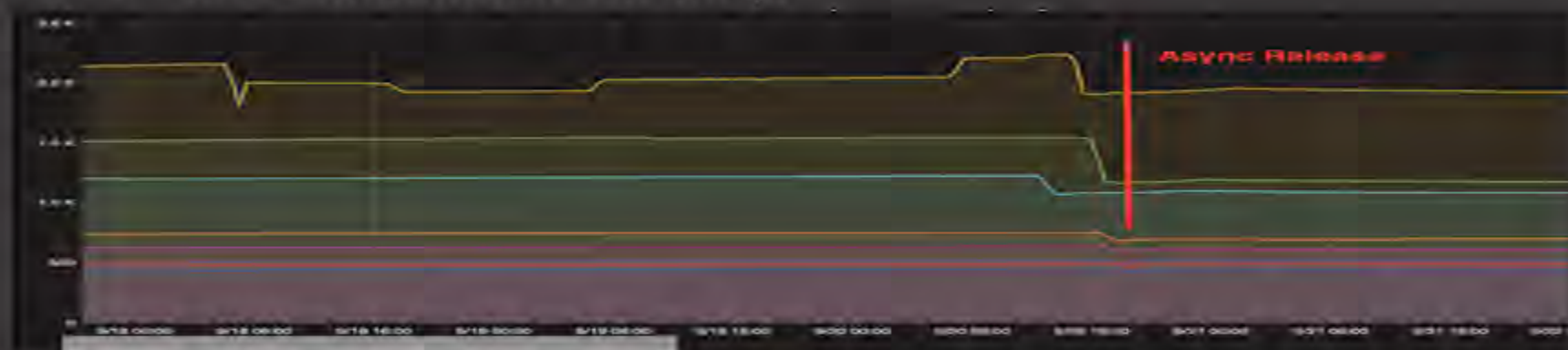
- Less Thread Pool
  - 90% Reduction for Thread pool number



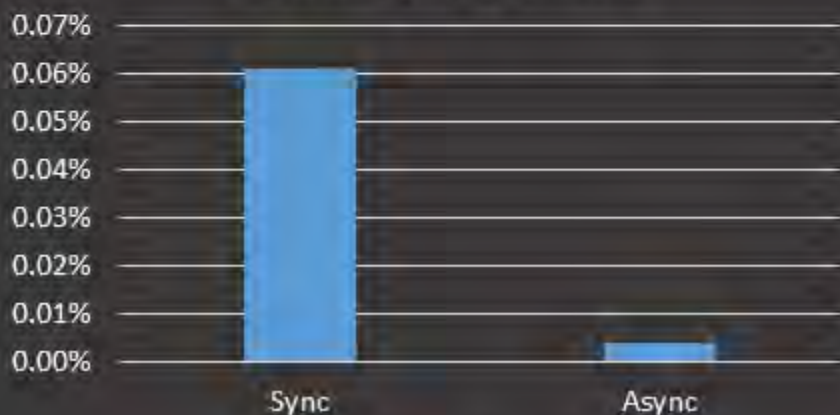


# Async DAL Service KPI Comparison – Cont.

- Memory Friendly
  - 20% Reduction for Memory Allocation
  - 100+MB Young Generation after Young GC
  - 130+MB Pooled Off-heap

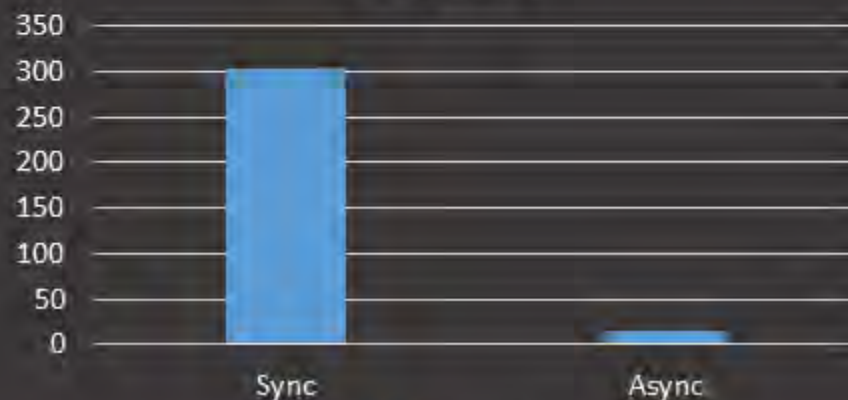


GC Time / Total Time



■ GC Time / Total Time

GC Count



■ GC Count

# **We Have ONE Async Dream**

- **Reform Application Charter from CPU-bound Charter to IO-bound**
- **Traffic Throughput (non-)linear growth with CPU Usage**
- **By guarantee Low Latency, Taking 20-30K TPS with 500MB JVM Heap (After young GC)**
- **Cloud Friendly Application**
  - **Less Hardware Investment**
  - **Low Operational Cost**
  - **Easy Capacity Estimation**



# High Performance Design

## E2E Async

- Non-blocking Pipeline: Async RPC + Async DataAccess

---

## Less is More

- **Shared** ThreadPool OVER Separate ThreadPool
- **Inline** Execution over Execution cross Multiple Thread Pool

---

## Autonomous Memory Management

- Use Off-Heap as much as possible  
( inbound/outbound & [de]serialization )
- Release Inbound Memory At earlier stage (submitRequest)



# High Performance Good Practice

## Inbound/Outbound Management

- Batch Consolidation
  - Order Management
  - Timeout Management
  - Retry Only Happen in Client Side
- 

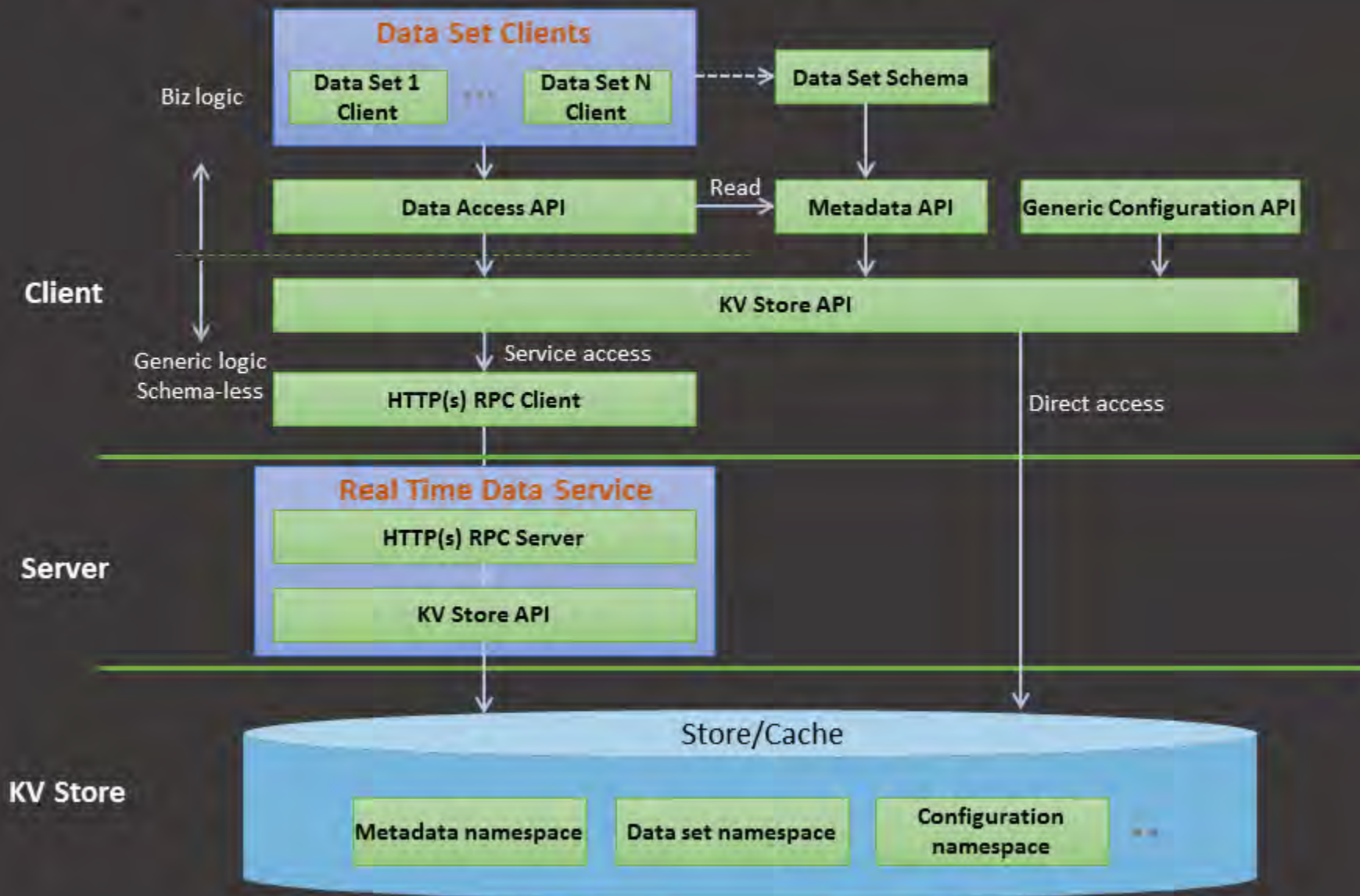
## Programming Habit

- Fast Fail over Exception Thrown Cascading
  - Logging & Monitoring Matters
  - Thread-safe Write Operation In Control Plan while Exception-safe Read Operation In Data Plane
- 

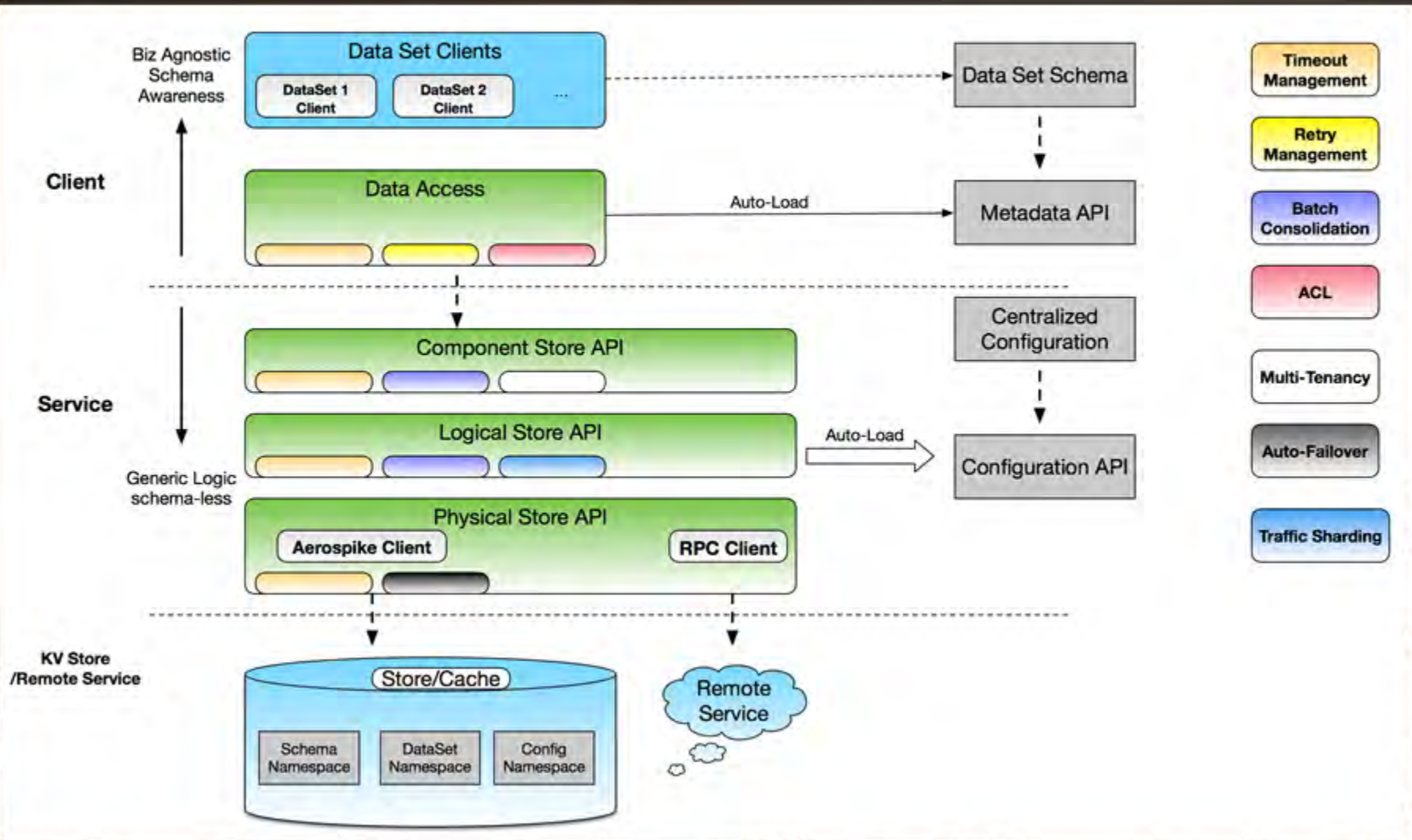
## KPI Sign-Off

- Performance Test as Critical Path for Each Commit
- [Mandatory] Continuous Performance Test for Each Commit

# Async High Level Architecture



# Async DAL Service Hierarchy



# Async Data Access Maturity

## DAL Service Feature

- Client& Server RoR Identification
  - biz-schema aware on Client Side
  - Schema-less on Sever Side
- Traffic Sharding & Routing
  - Active-Active/Active-Standby
- Auto-Failover
- Multi-Tenancy
  - ACL
- Direct/Service-To-Service Replication

... ....

---

## Data Access Mapping

DataSet => KV Mapping  
Logical => Physical DataSet Mapping

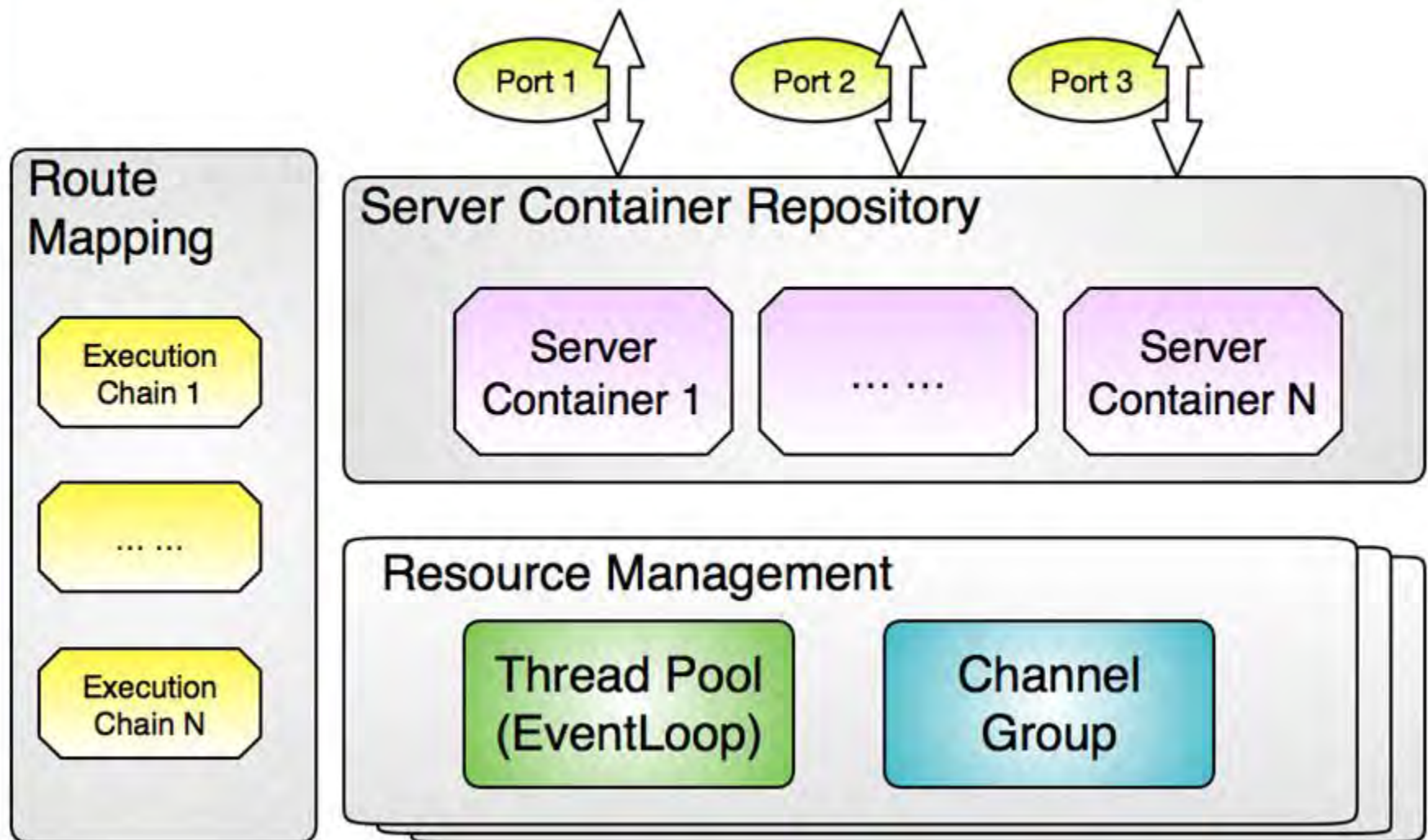
---

## Metadata Driven

- Source-of-Truth for Online Guideline & Offline Inventory
- Centralized Configuration
- Zero Restart/Auto-Fresh



# Async RPC Control Plane Abstraction



# Async RPC Maturity

## High Flexibility Configuration

- Configurable Execution Chain per URL
    - Customize protobuf / json encoder
    - Inject Monitoring Module
  - Execution Resource Configuration
    - Threadpool size / netty option (tcp\_nodelay)
    - Sharable or not
  - Service Listener Registry
- 

## RPC Resource Management

- Server Container Life Cycle Management
  - Graceful Shutdown
  - Partial Shutdown Given Container
- Auto Rebuild RPC Client Channel





# Async RPC Embrace Async DataAccess

RPC Client

RPC Client1

RPC Client2

RPC Client N

Accept Inbound

SSL Handler

Http Object Aggregator

Flush Outbound

RPC Server

Outbound  
Encoding  
Worker

Execution Chain Worker

URL /rds/get

Inbound  
Decoding  
Worker

CAL  
PreFilter

Protobuf  
PreFilter

DataAccess Async Handler

Logical Store API

Physical Store API

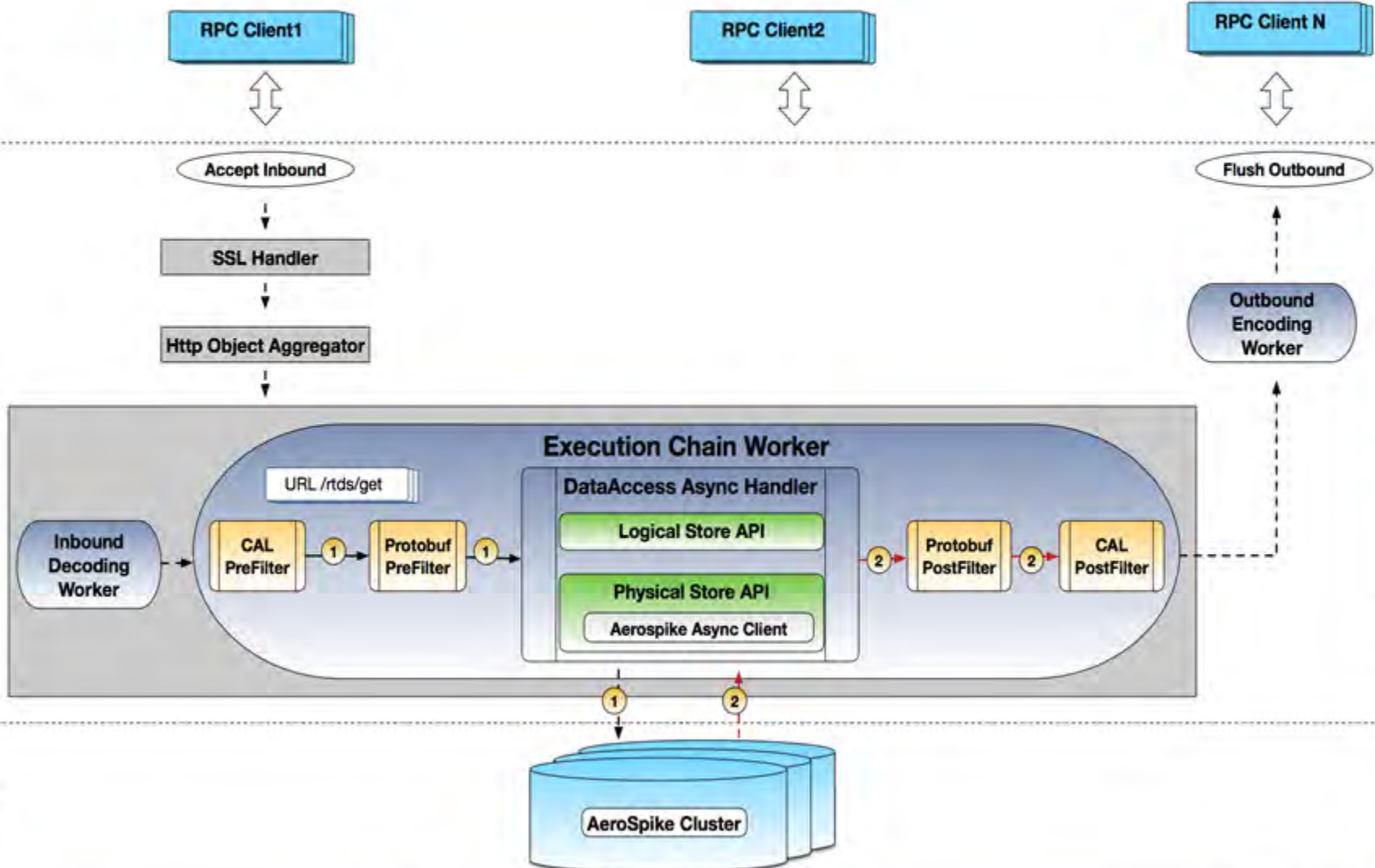
Aerospike Async Client

Protobuf  
PostFilter

CAL  
PostFilter

KV Store

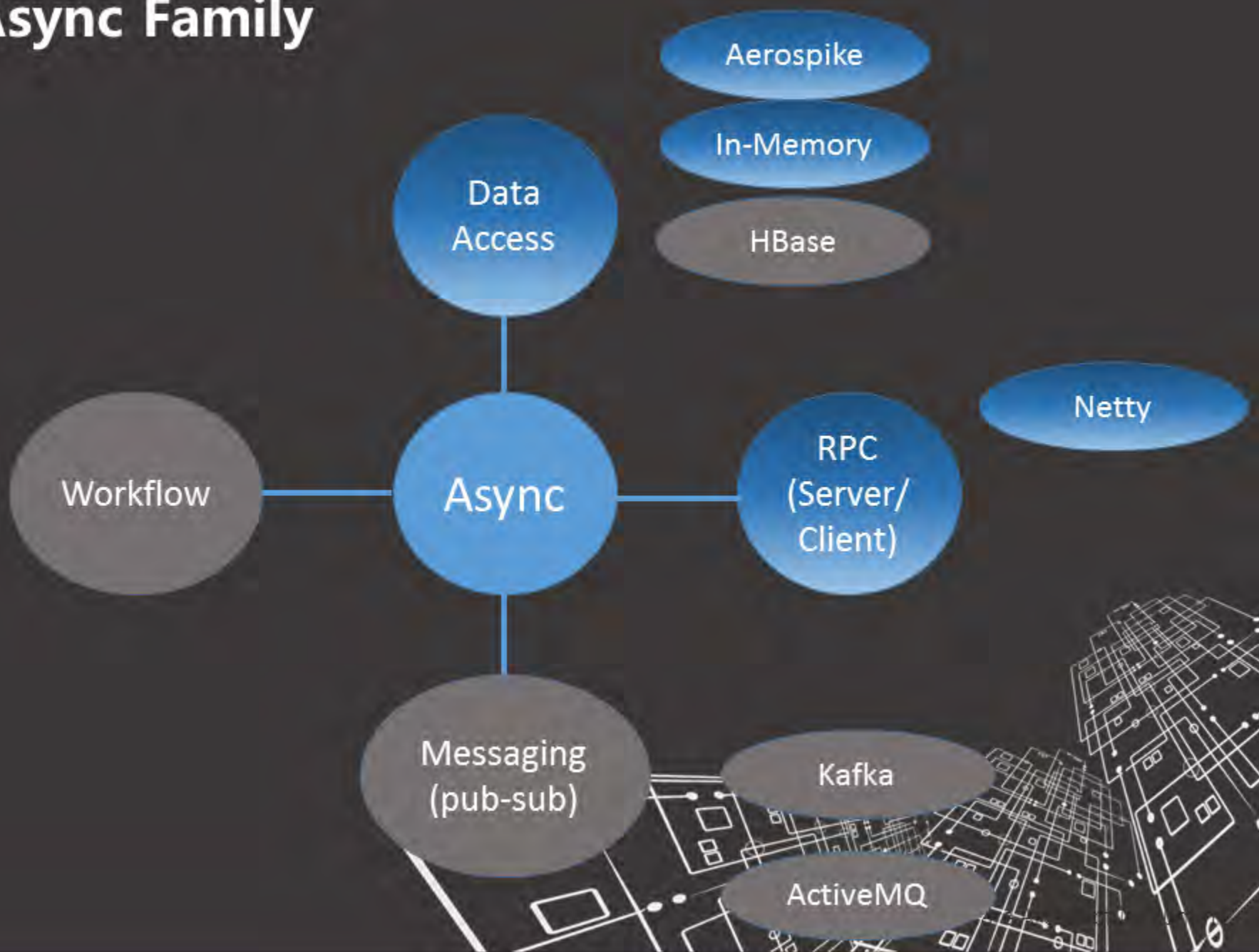
AeroSpike Cluster



# Async Core Value

|                                       |   |
|---------------------------------------|---|
| <b>High Performance</b>               | <ul style="list-style-type: none"><li>• Low Latency + High Throughput</li><li>• Low System Load</li><li>• SLA Isolation</li><li>• Understand Performance Contribution More</li></ul>  |
| <b>Cost Saving</b>                    | <ul style="list-style-type: none"><li>• Less Hardware Investment</li><li>• Loose Constraint for Hardware/VM SKU</li></ul>   |
| <b>Easy Adoption</b>                  | <ul style="list-style-type: none"><li>• Zero Code Change + Zero Release (new case on-board)</li><li>• Minimize new DB Storage Integration Effort</li><li>• Lego-Style Customization</li><li>• Highly Reusable Functionality</li></ul> |
| <b>High Flexibility Configuration</b> | <ul style="list-style-type: none"><li>• Execution Chain per URL (RPC)</li><li>• DataAccess Storage &amp; Option [consistency &amp; ttl]</li><li>• Traffic Routing Strategy</li><li>• Replication Strategy</li></ul>                   |

# Async Family



# AGENDA



PayPal & PayPal Risk (Platform)

Risk DAL Service Challenge

Async Solution

Async Future Plan





# Future Plan

## Open Source in Year 2019

### Async+Sync Hybrid Workflow Execution

---

#### Async DataAccess

- Compute Operation Support
    - DB Server-side UDF Adoption
  - Smart Client for Direct & Service Access
  - Async HBase Integration
- 

#### Async RPC

- Finer Granularity Monitoring & Throttling
  - Error Handling Injection
  - Client Side Multiplexing
  - Server Push Partial Response + RPC Client Consolidate Response
- 

#### Continuous Performance Tuning Deep Dive

- Shared Eventloop
- Netty Option (IO Ratio)
- NIO vs Epoll SocketChannel
- JDK SSL vs OpenSSL
- Protobuf vs Msgpack
- Sync Client vs Async Client
- W/- Monitoring/Replication features

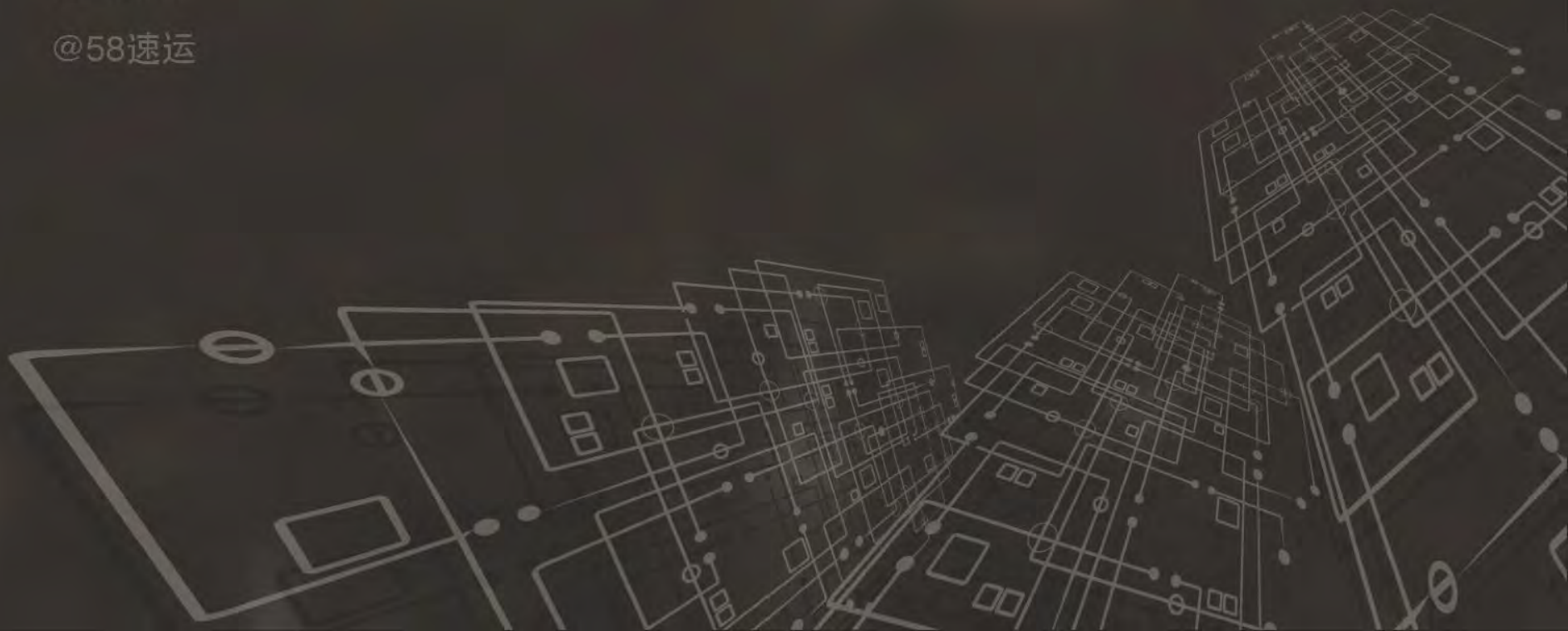


2017 Software Architecture Summit

# 打造58速运 高性能、高可用、实时消息平台

任桃术

@58速运



# I 目录

- 业务背景
- 早期架构
- 58速运消息平台实践
  - 高性能、高可用、实时、高到达率、高扩展性
- 核心业务流程
- 总结

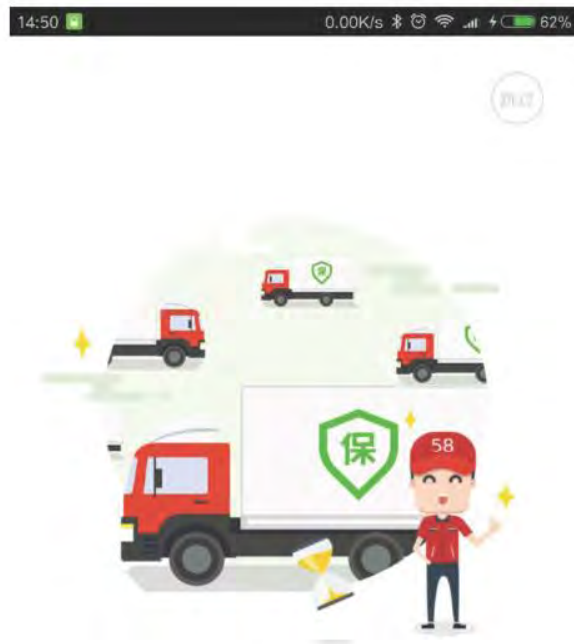


# 业务背景

- 58速运司机GPS位置实时上报



- 用户订单实时推送给司机

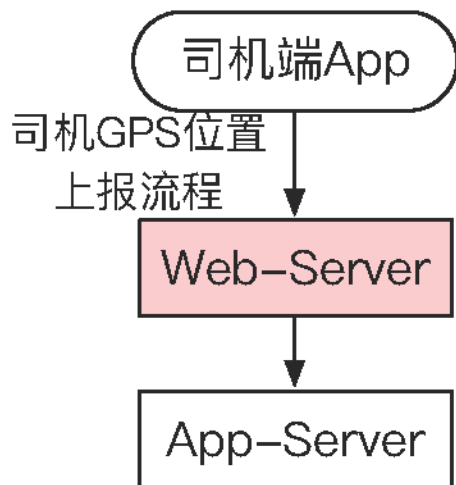


安全快捷

15秒快速响应 安全省心服务

# I 早期架构

- Web-Server性能问题



- 单点问题
- 三方推送  
及时性、限速、可达性

