China · Shenzhen

全链路稳定性背后的数字化支撑
阿里巴巴鹰眼技术解密

周小帆

阿里巴巴 — 中间件技术部

北京　伦敦　纽约　旧金山　圣保罗　上海　东京

QCon 全球软件开发大会

[上海站]

信息安全　机器学习　人工智能　黑产　互联网金融（FinTech）　团队管理　基础设施　云计算　软件性能　硅谷　微服务　互联网架构

主办方 Geekbang 极客邦科技　InfoQ

2017年10月17-19日
上海·宝华万豪酒店

扫描二维码
开启软件开发新思路

SPEAKER

INTRODUCE

周小帆

- 就职于阿里巴巴中间件&稳定性平台。
- 参与了阿里近五年来监控体系的建设及演进。中间件“鹰眼”监控系统服务端技术负责人。
- 阿里云产品“业务实时监控(ARMS)”技术负责人。

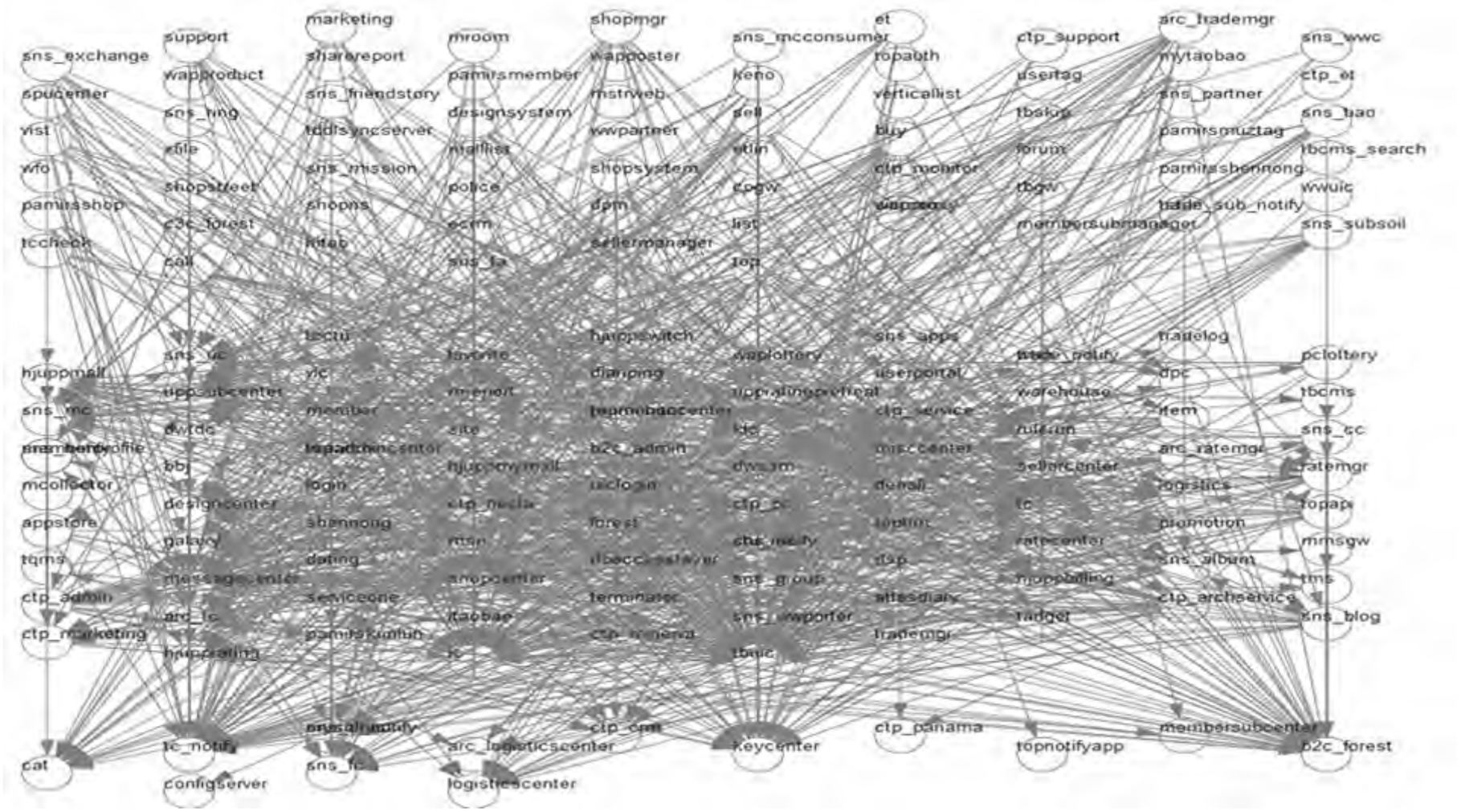
Agenda

- EagleEye@Alibaba
  - 分布式链路追踪技术原理、基础功能与使用场景

- EagleEye架构演进
  - 流计算、存储技术演进，模块化

- 被动 -> 主动
  - 识别、关联、定位

微服务之“熵”



2012 淘宝核心链路应用拓扑图

故障定位难

容量预估难

资源浪费多

链路梳理难

EagleEye是什么

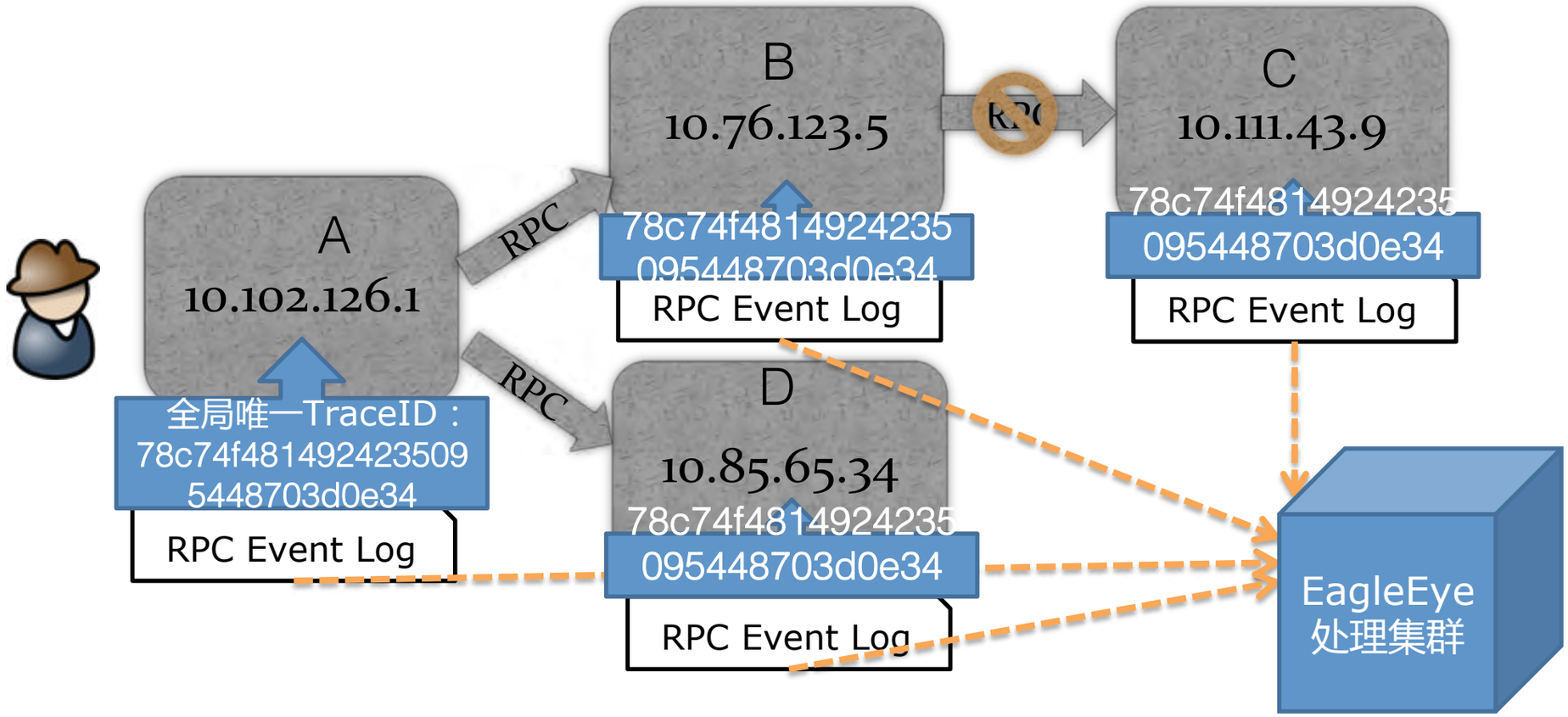
EagleEye是一个以链路追踪技术为核心的监控系统。

EagleEye通过收集、存储、分析分布式系统中的调用事件数据，协助开发和运维人员进行故障诊断、容量预估、性能瓶颈定位、调用链路梳理。

EagleEye的灵感来自于Google的Dapper论文。
（https://research.google.com/pubs/pub36356.html）

一次分布式调用

问题：在每天10000+亿次的分布式调用中，
如何定位这一次分布式调用某一环节出现的问题？



一次分布式调用

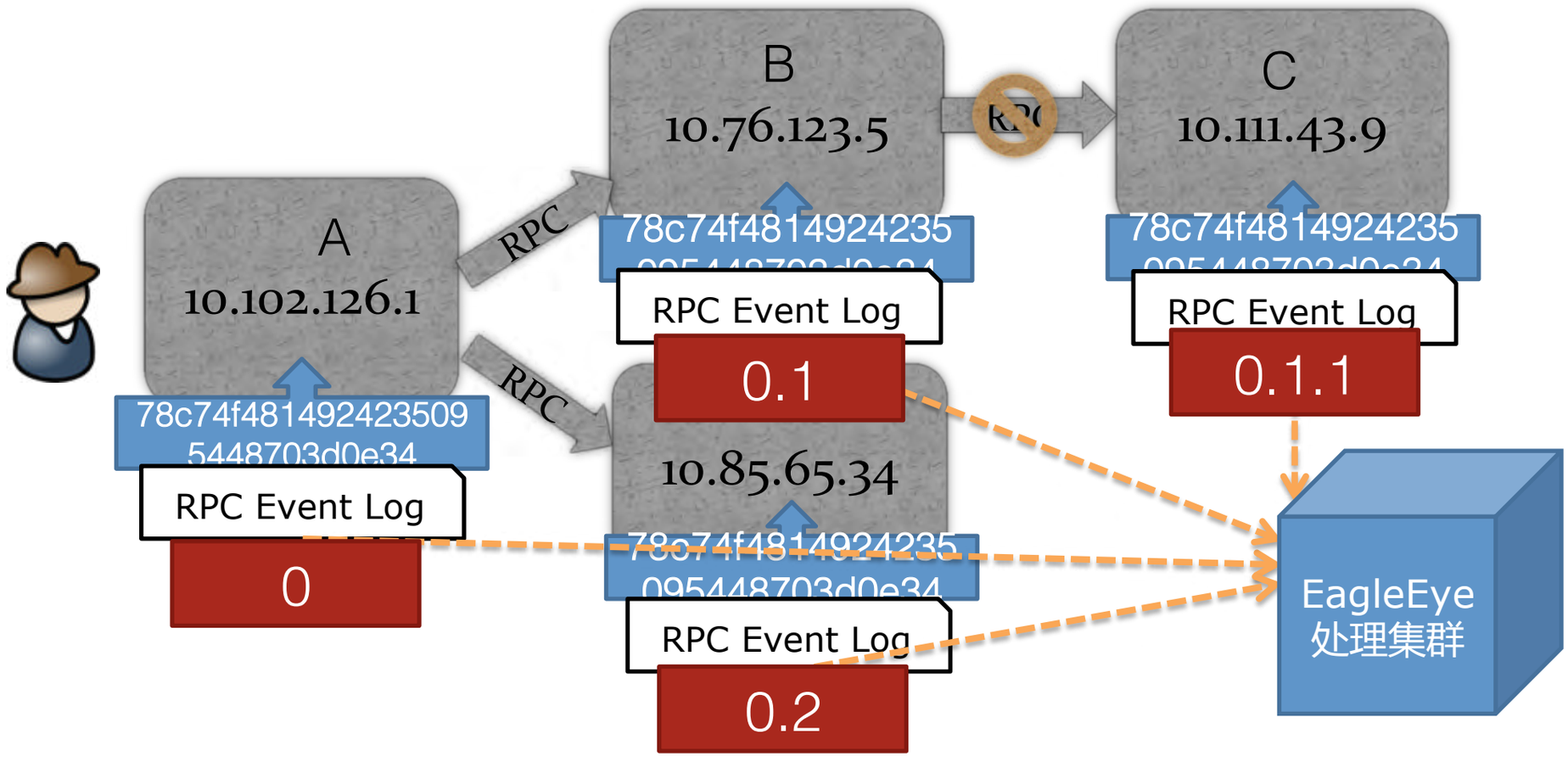
从倒排索引中查找TraceID: 78c74f4814924235095448703d0e34

时间	调用
2014-11-11 00:00:00,123	10.102.126.1 | HTTP Call | OK
2014-11-11 00:00:00,127	10.76.123.5 | Service Call | OK
2014-11-11 00:00:00,128	10.111.43.9 | DB Call | TIMEOUT
…	…

问题：如何还原实际调用“栈”?

RPCID

一次分布式调用
TraceID +
RPCID(SpanID)

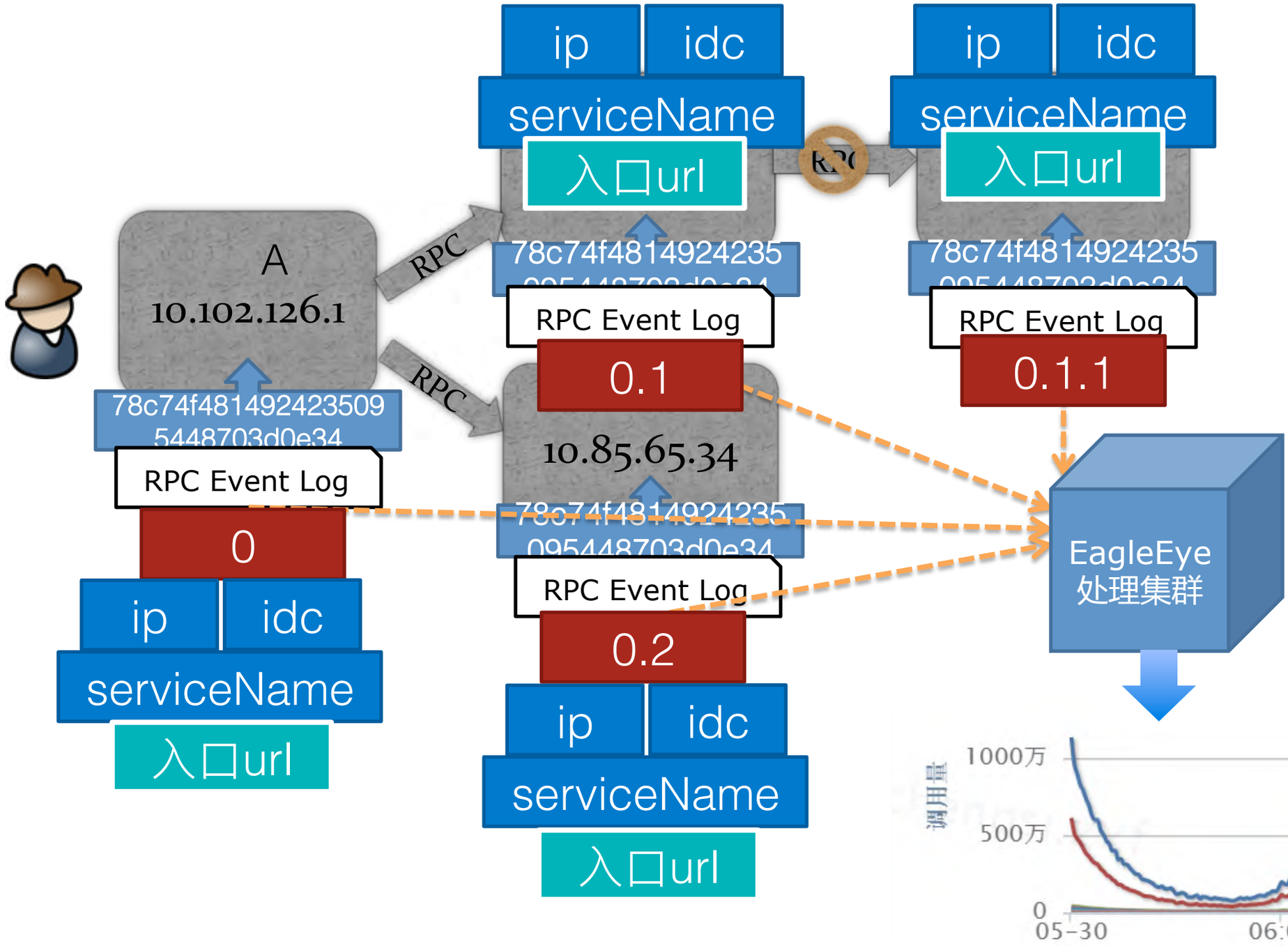


一次分布式调用
TraceID + RPCID

在异常日志的错误信息中找到 TraceId

TraceId=ac1828791374269125174692 3

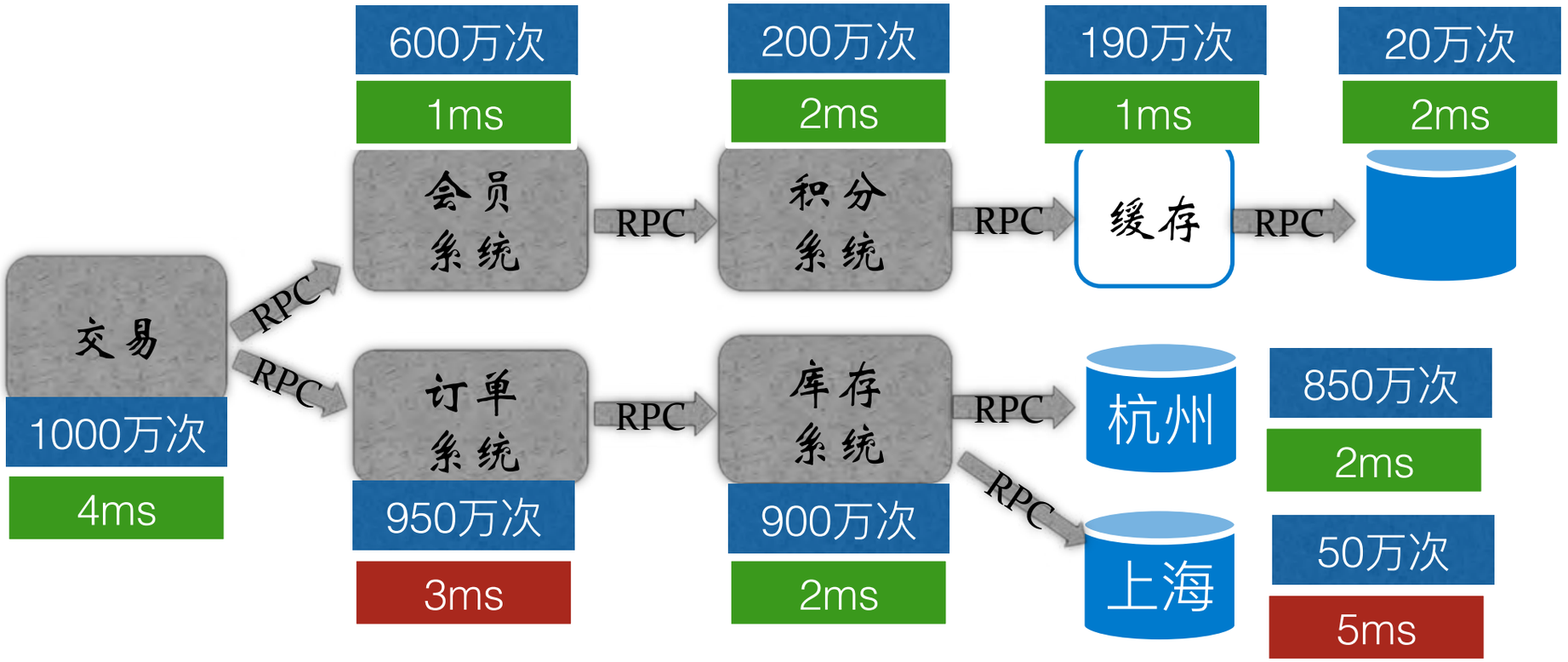
将万亿级别调用链数据进行聚合



按照指定Tag统计调用次数

将万亿级别调用链数据进行聚合
（根据业务入口进行聚合）

交易
下单
入口



价值：
1. 发现热点，发现瓶颈，发现“非法流量”。
2. 容量预估。（例：按照双十一各入口流量预估值，按照链路下游各系统的调用比例，计算出双十一各子系统所需承担的流量。）

调用链的聚合结果 - 链路分析

下单链路

层次	名称	QPS	峰值QPS	调用比例	被调用均值	平均耗时	耗时比例	出错率	同机房	标记
0	▾ http://	82.66	*8700*	1.0000	1.0	312ms	16.52%	0.00%	0.0%	瓶颈
1		1.83	*34450*	0.0221	11.07	0ms	0.01%	0.00%	100.0%	
1		439.22	*22200*	5.3138	5.45	0ms	0.86%	0.02%	99.98%	
1	▾	0.21	*3660*	0.0025	1.02	2ms	0.00%	0.00%	100.0%	
2		0.21	*3660*	0.0025	1.02	0ms	0.00%	0.00%	100.0%	
2		0.13	*20*	0.0016	1.0	1ms	0.00%	0.00%	100.0%	
1	▾	80.61	*8480*	0.9752	1.0	5ms	1.49%	1.56%	100.0%	
2	▸	0.0	*130*	0.0000	1.0	10ms	0.00%	0.00%	98.0%	
2	▸	0.01	*190*	0.0001	2.13	10ms	0.00%	0.00%	81.25%	
2		0.01	*130*	0.0001	2.27	0ms	0.00%	0.00%	100.0%	
2		0.01	*190*	0.0001	2.13	0ms	0.00%	0.00%	100.0%	
1		79.45	*8440*	0.9612	1.0	0ms	0.08%	0.00%	100.0%	
1	▸	0.85	*60*	0.0103	1.09	5ms	0.01%	0.01%	100.0%	强依赖 错误阻塞
1	▸	0.15	*520*	0.0018	1.0	107ms	34.29%	0.00%	100.0%	瓶颈
1		0.08	*30*	0.0010	1.0	2ms	0.00%	0.00%	100.0%	
1		0.15	*520*	0.0018	1.0	0ms	0.00%	0.00%	100.0%	

依赖压力点

易故障点

潜在易故障点

耗时瓶颈点

EagleEye基础功能小结

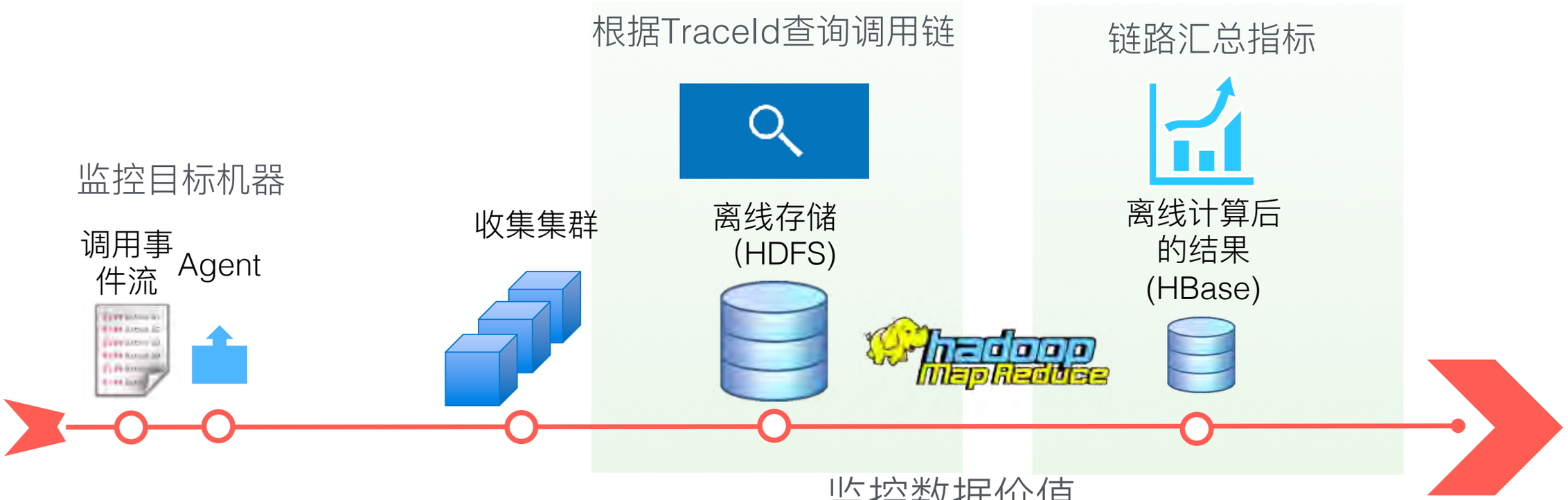
EagleEye通过集中收集与存储分布式系统中的调用数据，并按照TraceId进行索引，按照RPCId进行重组，将分布式调用链堆栈进行还原。从而实现故障定位的功能。

通过对调用链数据的分析，按照业务入口、链路特征、应用、机房等一系列Tag进行聚合统计。
为容量预估、性能瓶颈定位、调用链路梳理等稳定性保障工作提供报表。

Agenda

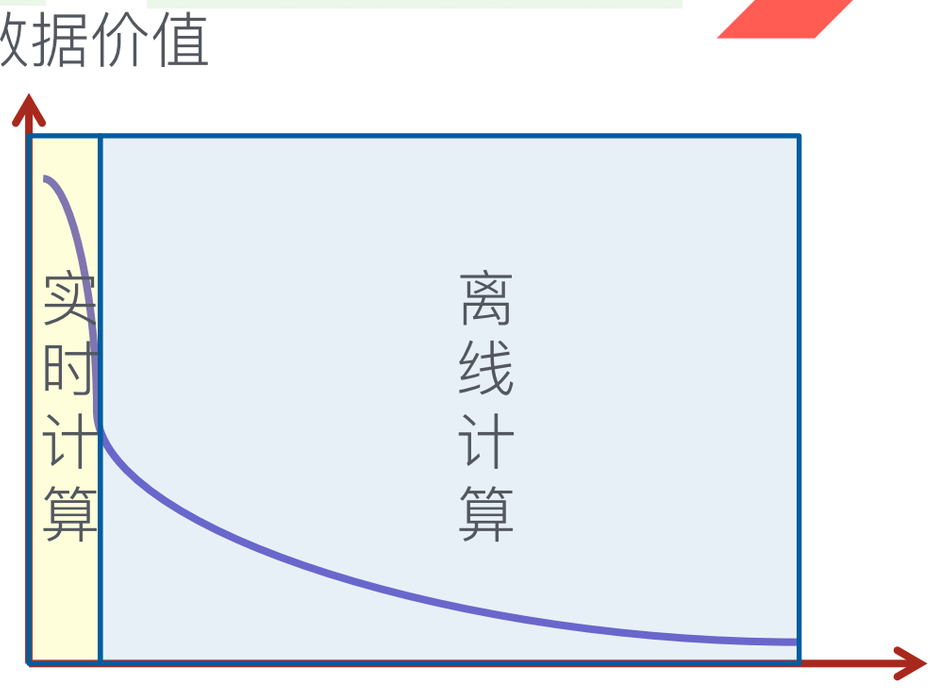
- EagleEye@Alibaba
  - 分布式链路追踪技术原理、基础功能与使用场景
- EagleEye架构演进
  - 流计算、存储技术演进，模块化
- 被动 -> 主动
  - 识别、关联、定位

鹰眼整体架构 (2012)



主要问题

1. 不够实时
2. 不够轻量



鹰眼整体架构 – 实时化 (2014)

监控目标机器

调用事件流

Agent

流计算引擎

根据TraceId查询调用链

离线存储 (HDFS)

hadoop MapReduce

链路汇总指标 (离线)

离线计算后的结果 (HBase)

实时计算后的结果(HBase)

链路汇总指标 (实时)

实时拓扑大屏

实时场景

全链路压测
实时流量调度与限流
异常链路主动发现

鹰眼整体架构 – 轻量化 (2016)

监控目标机器

调用事件流

Agent

流计算引擎

根据TraceId、耗时、错误、时间段等各种条件查询调用链

MPP列式数据库

实时计算后的结果(HBase)

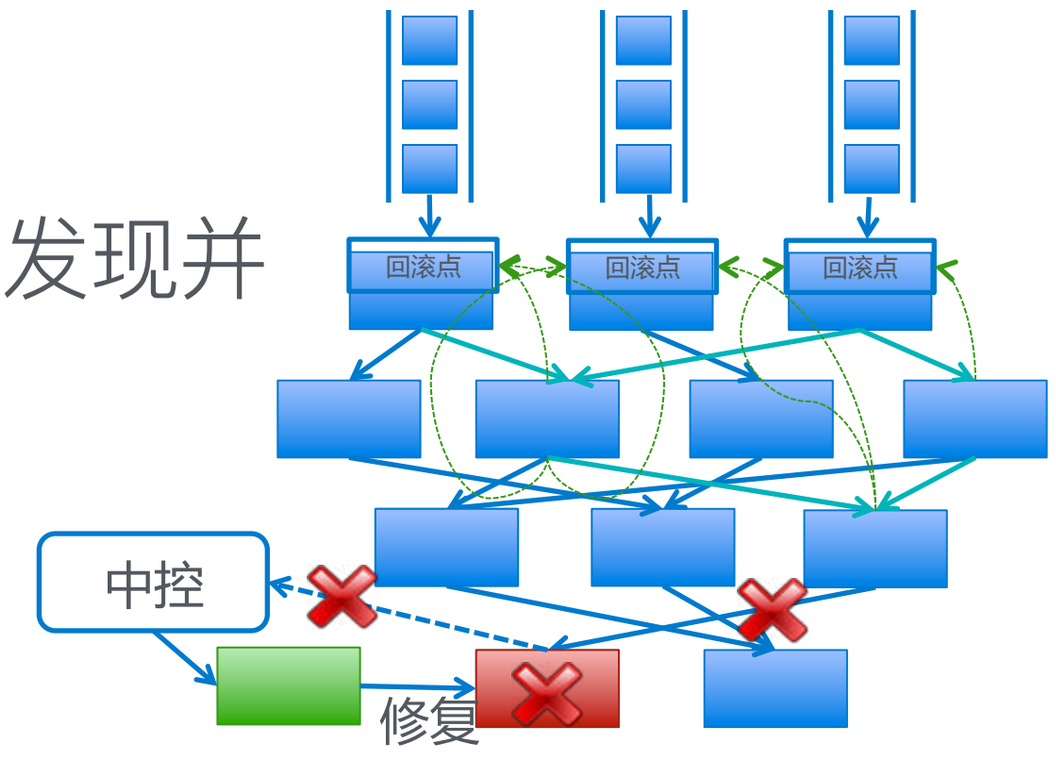
链路汇总指标(实时)

报警

监控系统 - 流计算的挑战

- 持续计算稳定性
  24x7持续计算，无法停机维护。
  宕机自愈能力。

- 解决方案
  具备高可用中控节点，自动发现并重启计算节点的流计算引擎(JStorm)。
  自带Exactly-Once语义。



监控系统 - 流计算的挑战

- 计算过程不落地
  有限内存 vs 海量数据
  热点问题

- 解决方案

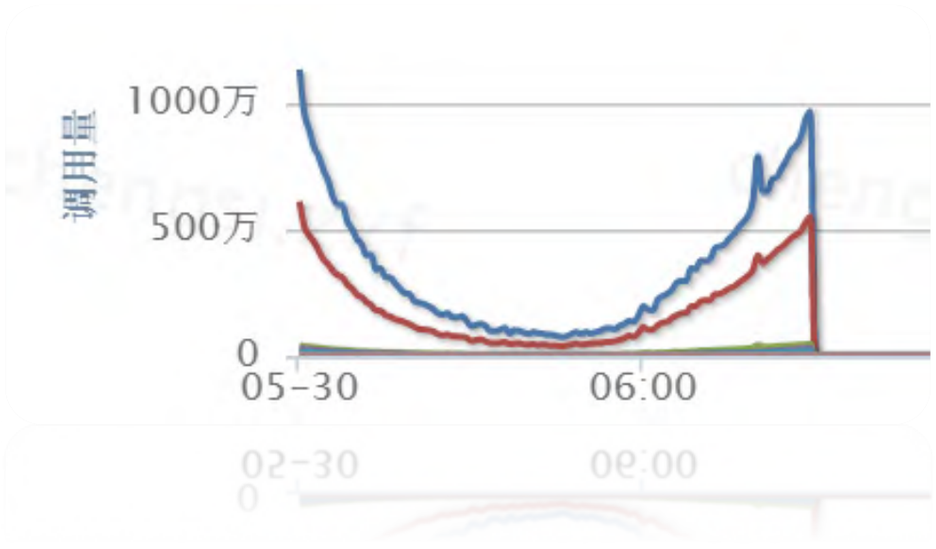
  全量计算->流式增量计算（例：全量数据计算入口页面uv vs. 增量计算入口页面uv ）

  One-Pass & Mergable 推荐StreamLib（https://github.com/addthis/stream-lib）

  流式聚合将聚合拆分为LocalReduce+GlobalReduce以避免热点。

监控系统 – 确定性保证

延迟与准确性
Tradeoff



监控对象出现问题？

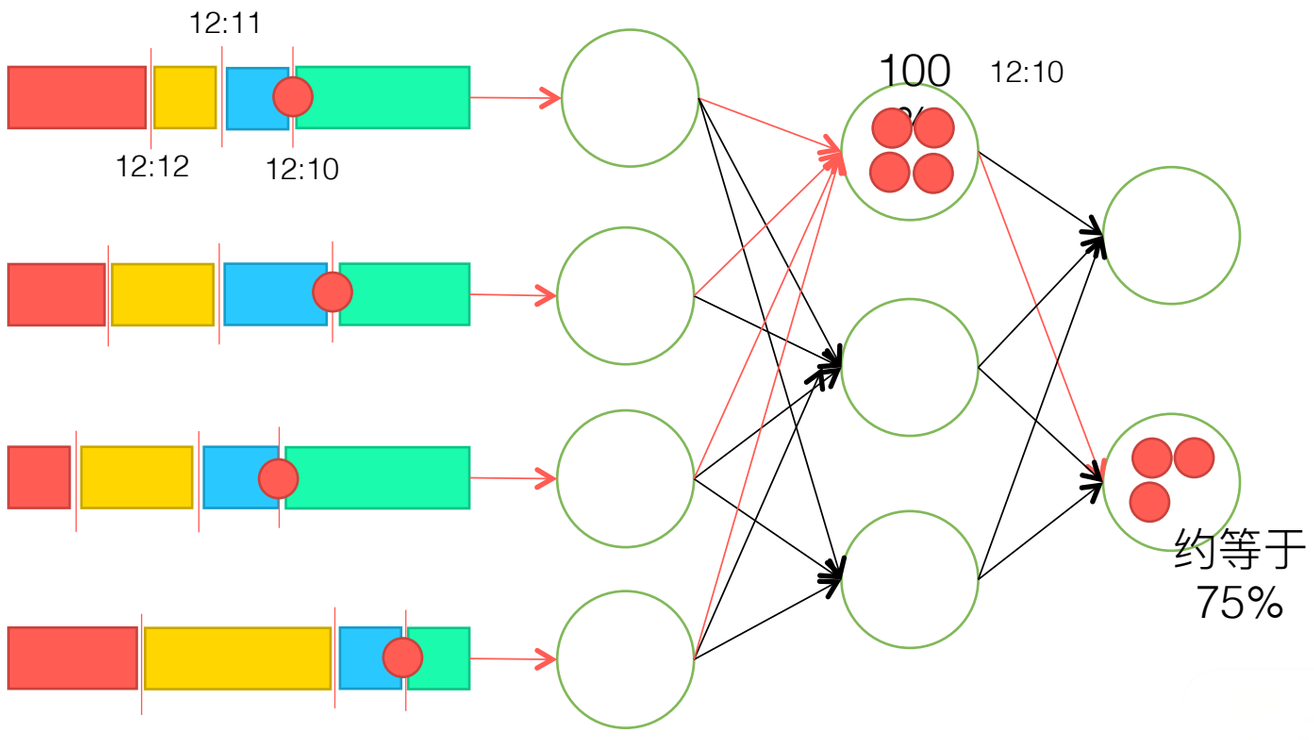
还是

监控系统处理延迟？

监控确定性 > 最终一致性

相关优化 – 齐全度优化

延迟与准确性SLA



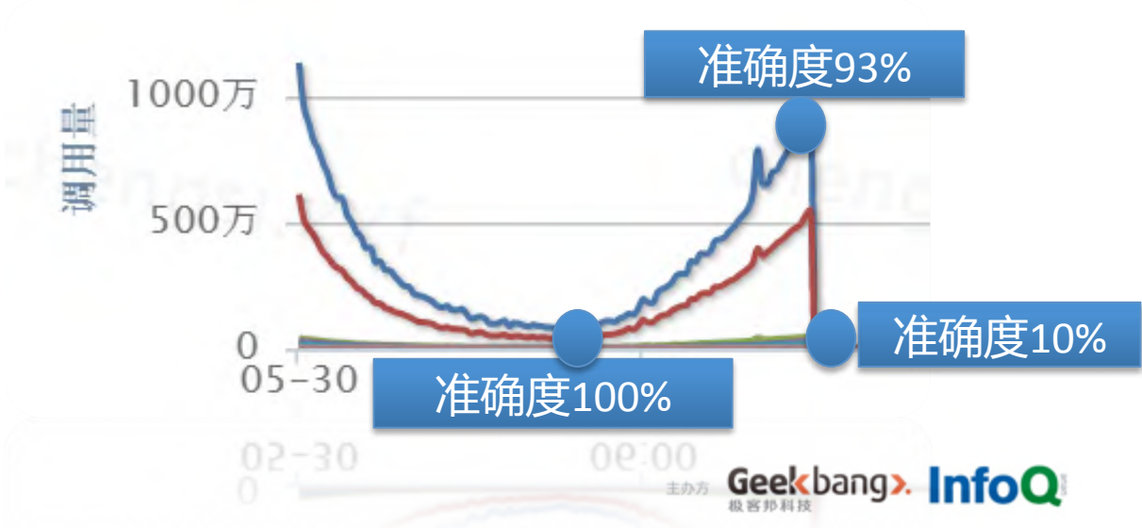
Partition越大，Partition数据越均匀，
到达Barrier/全部Barrier越接近于真实齐全度。

产生机制：

基于Snapshot算法。
利用单一数据源局部有序特性。
在业务时间窗口交界处产生。
(Watermark)

用途：

依据任意时间点该Watermark百分比，估算出该时间点的齐全度。



存储层优化

调用链存储

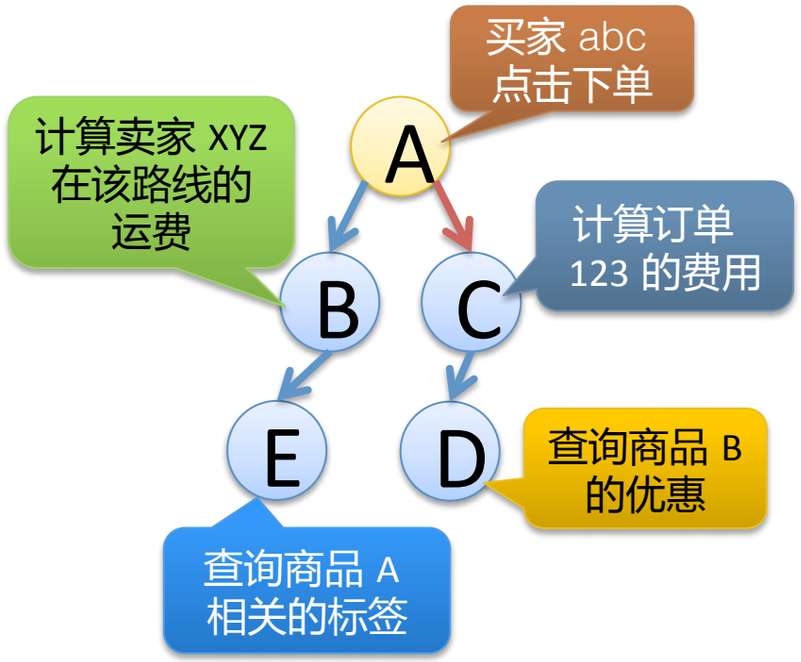
- 阶段一：使用 HBase 存储，TraceId 做 Rowkey
- 阶段二：使用 Hadoop/ODPS 存储
  1. 按 TraceId 的时间戳和哈希值进行分片，分片内按 TraceId 排序
  2. 针对调用中记录的每个列进行针对性压缩，节省存储
- 阶段三：使用分库分表的列式存储HiStore，按 TraceId 的时间戳和哈希值进行分库分表 HiStore 支持列式高压缩比存储，兼容 MySQL 生态，非常适合写多读少的场景

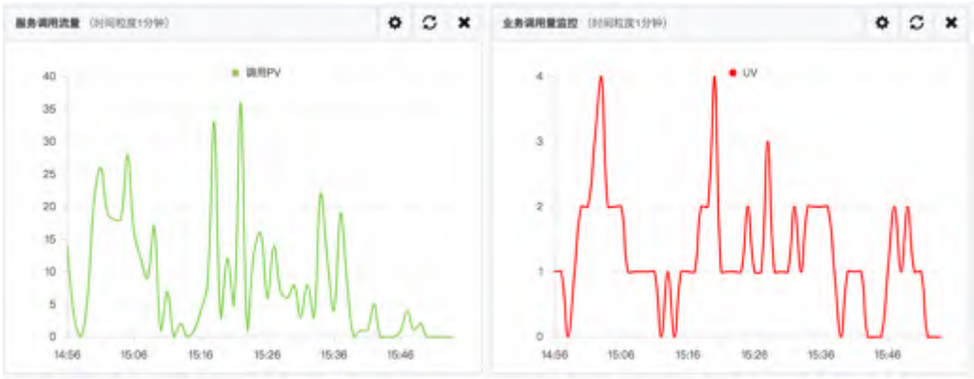
指标存储

- 基于OpenTSDB的Rowkey Schema
- 移除OpenTSDB的Proxy层
- 流计算中完成降精度
- Co-Processor和预计算（流计算）互补。解决合并时间窗、TTL等多种后计算需求。
- 同样引入StreamLib完成基于海量维度的估算

模块化改造

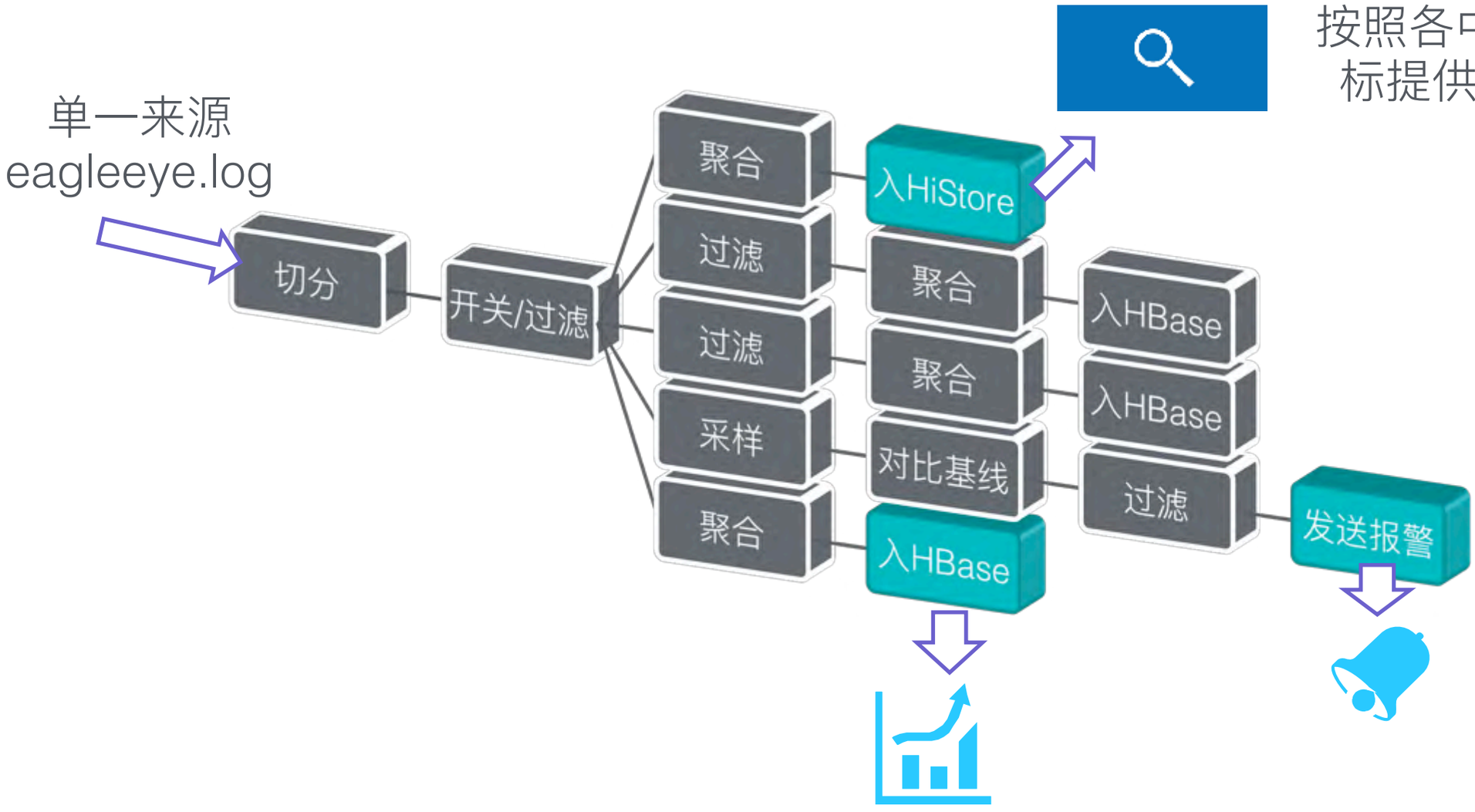
- 实际排查中遇到的案例
  - 线上的某次系统调用异常，是由哪笔订单的什么操作引起的？
  - 这笔异常订单，是否由于卖家对应关联商品的某些异常操作导致？
  - 系统QPS出现波动，影响了上游哪些服务？





建立业务id与TraceId的双向绑定
业务指标与系统指标双向关联
用户存在“自定义”链路的需求

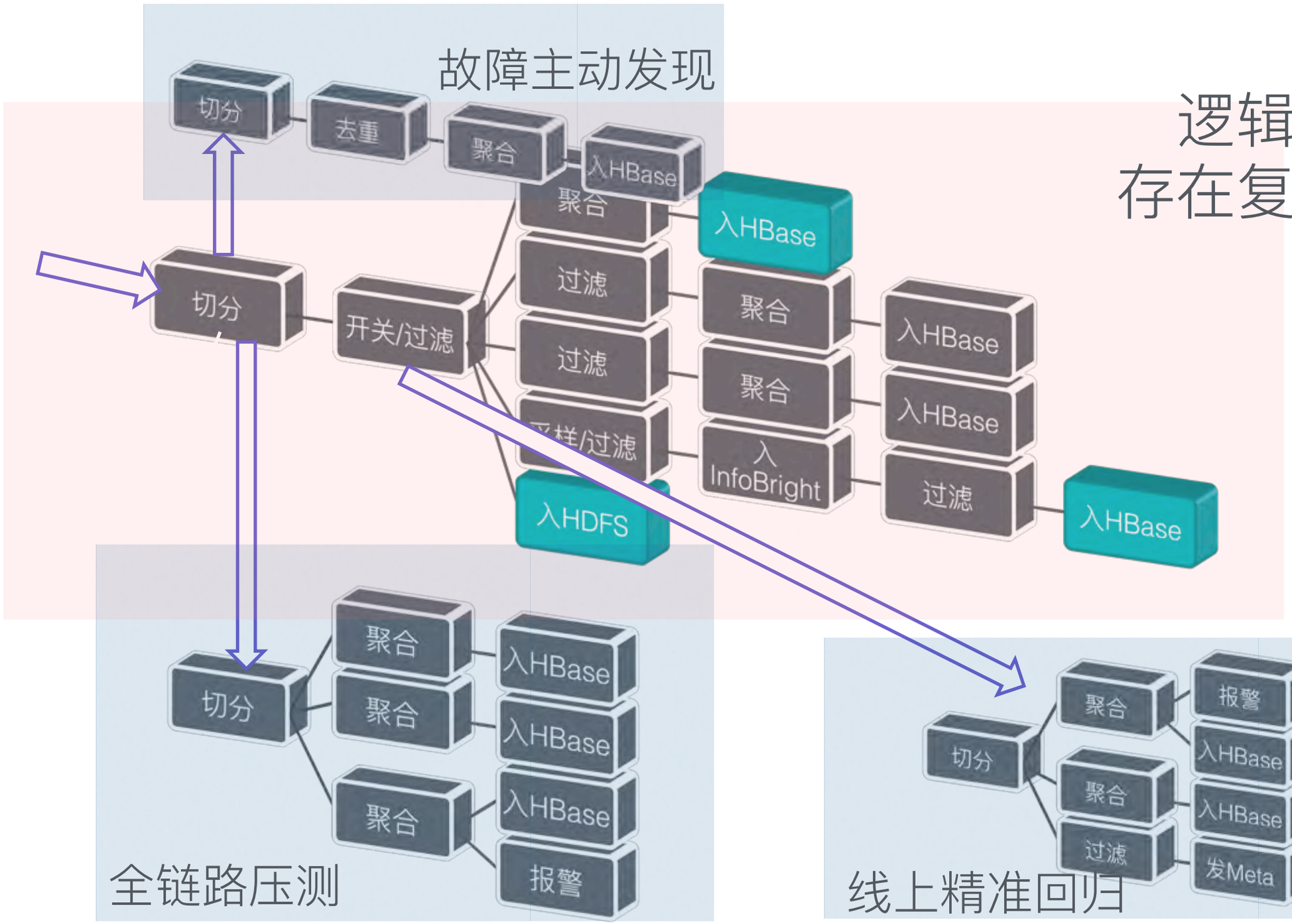
EagleEye集群Pipeline



“自定义链路”

- 数据来源
  日志、消息队列、binlog、巡检、拨测
- 数据过滤方式
  按服务过滤、按TraceID采样…
- 聚合维度&精度
  维度：按容器、服务器、机房、服务、库、队列Topic、入口、业务类型、用户区域等
  统计时间粒度：按秒、分钟、小时、天
- 持久化方式
  HDFS/HBase/MySQL/列式存储/分布式缓存/搜索…
- 部署方式
  跨机房部署、独立部署、逻辑热拔插

满足个性化需求的EagleEye Pipeline



逻辑爆炸
存在复用可能

用“积木块”搭建数据处理流程 (ARMS，内部名TLog)

分词、切分

逻辑判断

聚合计算

持久化

告警、消息

Still in Lab

Powered by Google Blockly

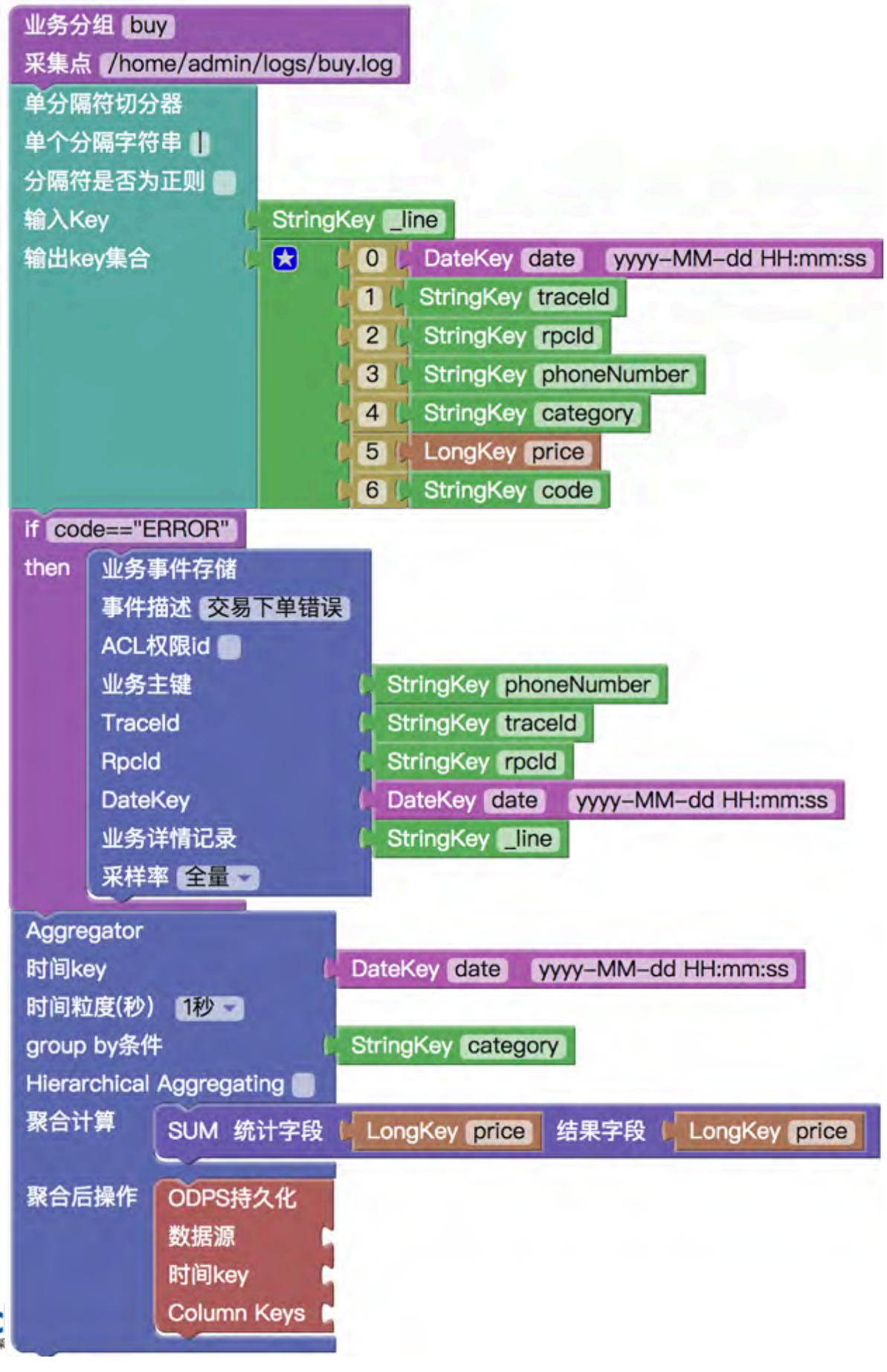
日志来源: buy

日志路径: /home/admin/logs/buy.log

2016-07-25 17:25:00|0a4851441424944934716233 9e|0.1|18661234567|天猫|300|ERROR
2014-07-25 17:25:00|0a4851441424944934716233 9e|0.3|18661234567|淘宝|400|SUCCESSFUL
2014-07-25 17:25:00|0a4851441424944934716233 9e|0.15|18662233445|聚划算|500|53160|SUCCESSFUL
…
…

竖线分割，分别表示:
日期|TraceId|RPCId|电话|来源|价格|操作结果

目标: 1. 如果操作出现失败，将该业务事件与鹰眼调用链产生关联。
2. 分别统计各来源的交易额度



采集来源

Alibaba Group 阿里巴巴集团

数据清洗，切分字段

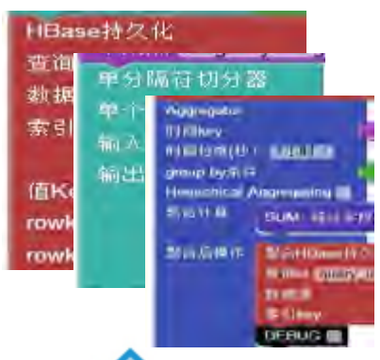
2016-07-25 17:25:00|
0a485144142494493471623 39e|0.1|
18661234567|天猫|300|ERROR

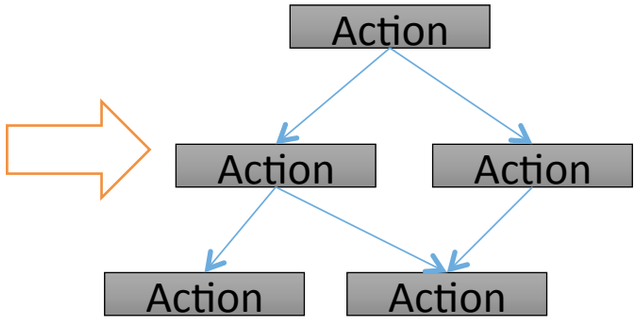
如果返回码错误，将本行日志关联至链路

按照入口进行Group By，Sum(Price)

存入离线数据分析平台，后续分析

将“积木块”转化为流计算(ARMS)





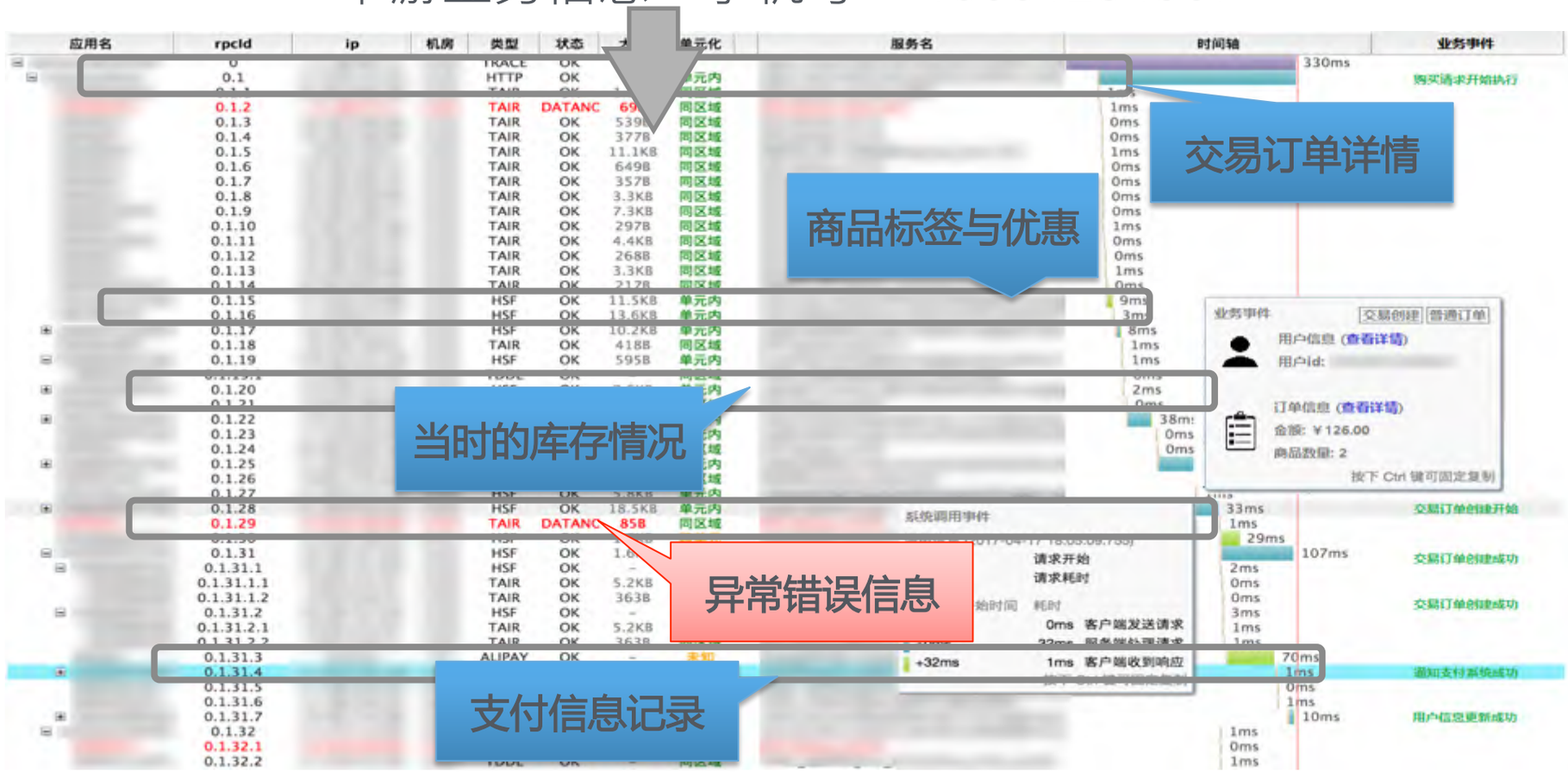
优化DAG

例：
聚合逻辑前置
自动合并开销较小的算子
自动删除不必要的数据传递

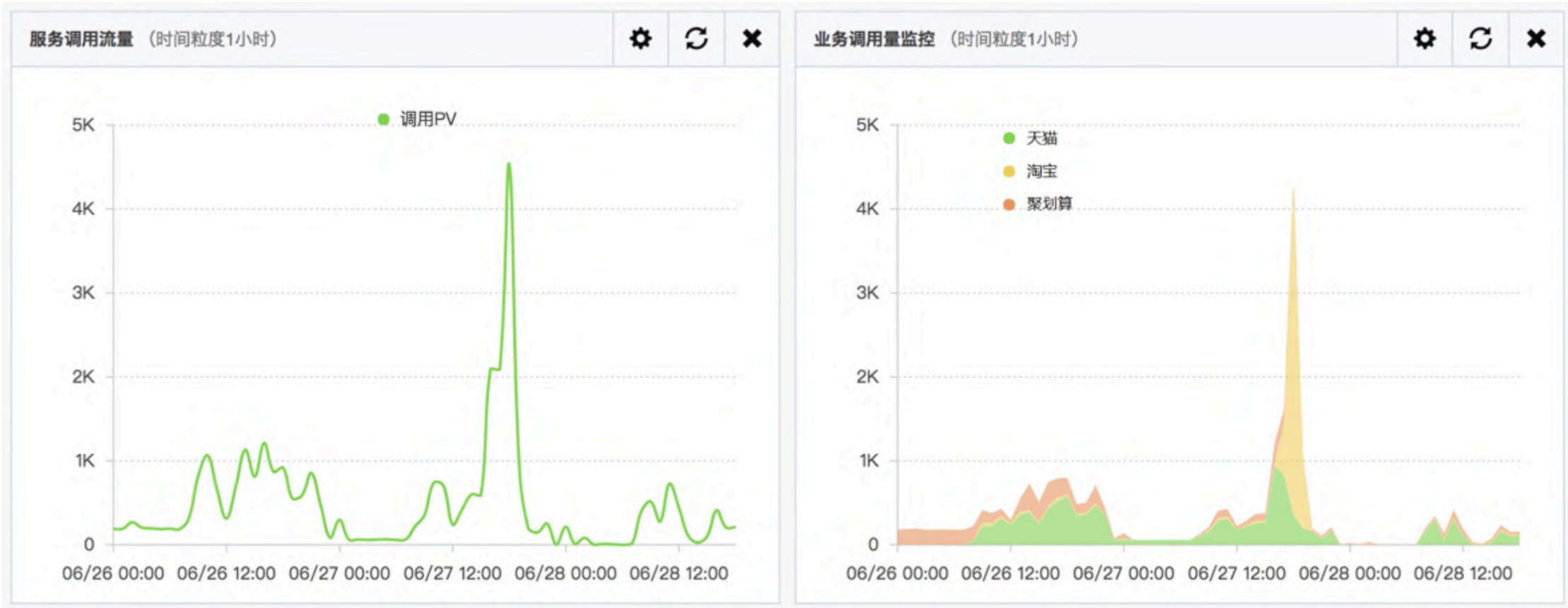
分发至流计算引擎上进行执行。(JStorm or SparkStreaming)

将业务与系统链路双向绑定

根据业务事件 id 反查调用链，从而顺藤摸瓜找到更多的上下游业务信息：手机号 = 18661234567



将业务与系统指标双向关联



模块化改造的结果

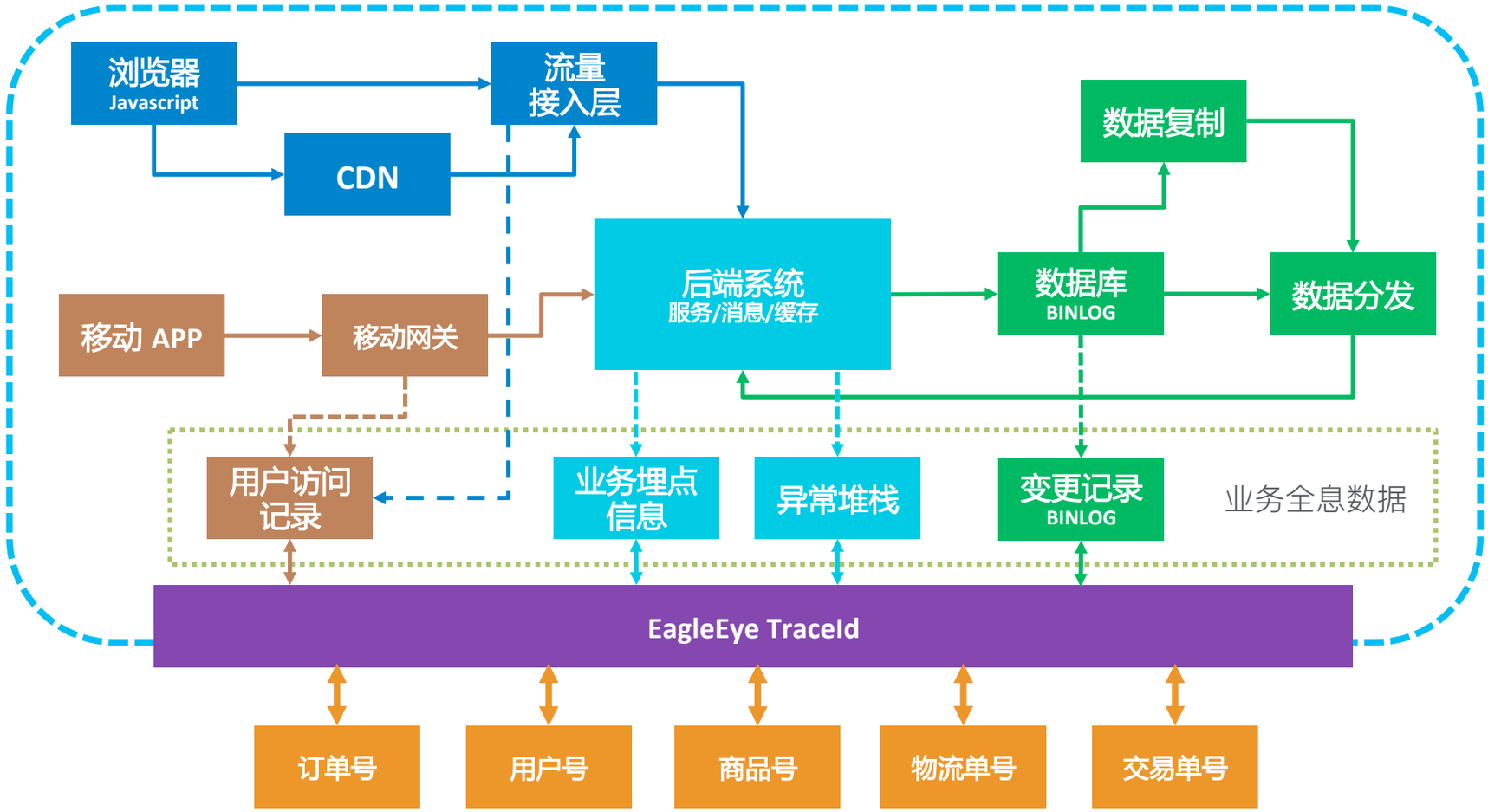
2014年Q1以前

EagleEye是一个只能接受固定来源与格式、按固定方式清洗、聚合、持久化的的监控系统。

2014年Q1以后

EagleEye演进成为了一个为用户提供自定义流式数据采集、清洗、计算、持久化能力，能够处理自定义指标与事件，并与原有的EagleEye链路产生双向关联。

全息排查全景图



线上作业数	日均处理量	秒级处理峰值
1780+	1PB+	7000W/s+

截止2017年7月

鹰眼整体架构 – 模块化 (2016)

原生鹰眼数据

监控目标机器

中间件调用事件流

Agent

鹰眼流计算逻辑

MPP列式数据库

实时计算后的结(HBase)

报警

自定义计算逻辑

业务日志事件流

数据库事件流

终端（无线、PC)事件流

自定义鹰眼数据

Agenda

- EagleEye@Alibaba
  - 分布式链路追踪技术原理、基础功能与使用场景
- EagleEye架构演进
  - 流计算、存储技术演进，模块化
- 被动 -> 主动
  - 识别、关联、定位

被动->主动

我们真的需要存储每一条链路吗?

我们的用户真的理解每一张报表的含义吗?

我们能否比用户更早地发现问题、定位问题?

被动->主动解决思路

识别

- 只存储“可疑的”数据
  - 调用返回错误、超时
  - 指标离群点、链路离群点、时序异常点

关联

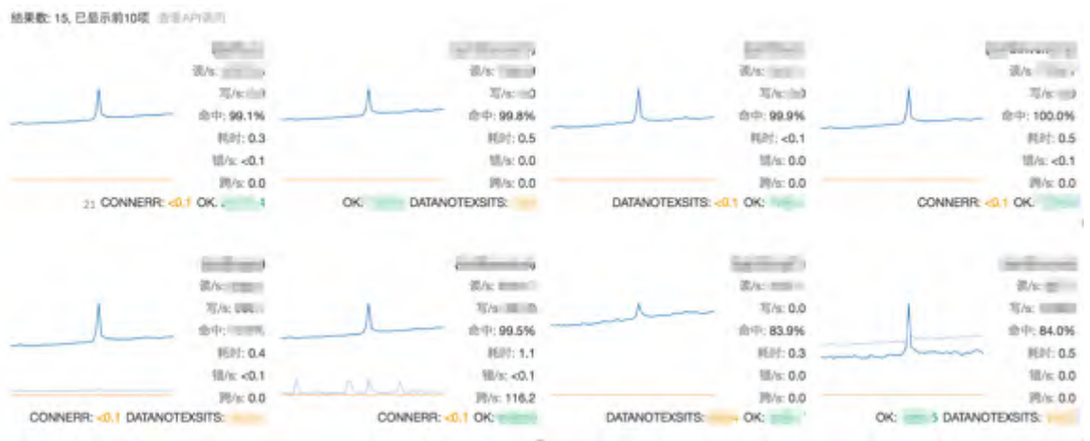
- 只展示与可疑数据有关联的报表
  - 点(事件)与面（指标）
  - 上游与下游

定位

- 根据关联关系，将“果”关联到“因”

识别

调用链中的异常



时序指标的异常点

指标中的离群点

识别

非结构化日志中的离群点与异常

识别+关联+定位

发布事件 配置变更
网络故障 业务事件

关联

全量调用链数据流

识别

调用链异常
时序指标异常点
指标离群点
离群点与异常
链路形态离群点
…

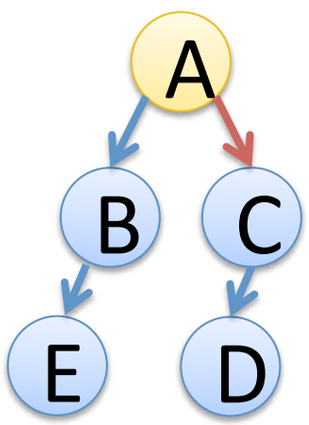
聚合

按点关联：同一部署单元上的问题
按线关联：时间点上临近的变更
按链路关联：链路上下游的问题

定位

现象 -> 原因

关联



案例

波动前1分钟
分库分表规则配置变更

关联

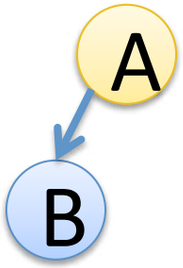
识别 某核心入口应用A业务指标出现小幅波动

识别 应用B卖家中心日志出现新增异常堆栈

聚合

按点关联：B应用接收到了配置变更推送
按时间线关联：配置变更发生在指标波动前1分钟。
按链路关联：A应用强依赖B应用。

关联



定位

配置变更为引发指标波动的高嫌疑原因。

人工验证：
卖家中心新增异常堆栈为：跨库事务异常嫌疑确认

阿里巴巴鹰眼系统演进史

2012

1.0版本 初出茅庐

- 链路跟踪能力(TraceId, RPCId)
- 离线、实时统计能力
- 数百个集团内应用接入

2014

2.0版本 平台化

- 支持全链路压测
- 流程插件化、配置可视化
- 对接1000+集团内个性化需求

2016

3.0版本 系统链路+业务链路

- 业务全息排查
- 业务实时监控
- 强化流计算/时序与事件存储

2017

云化 创新与探索

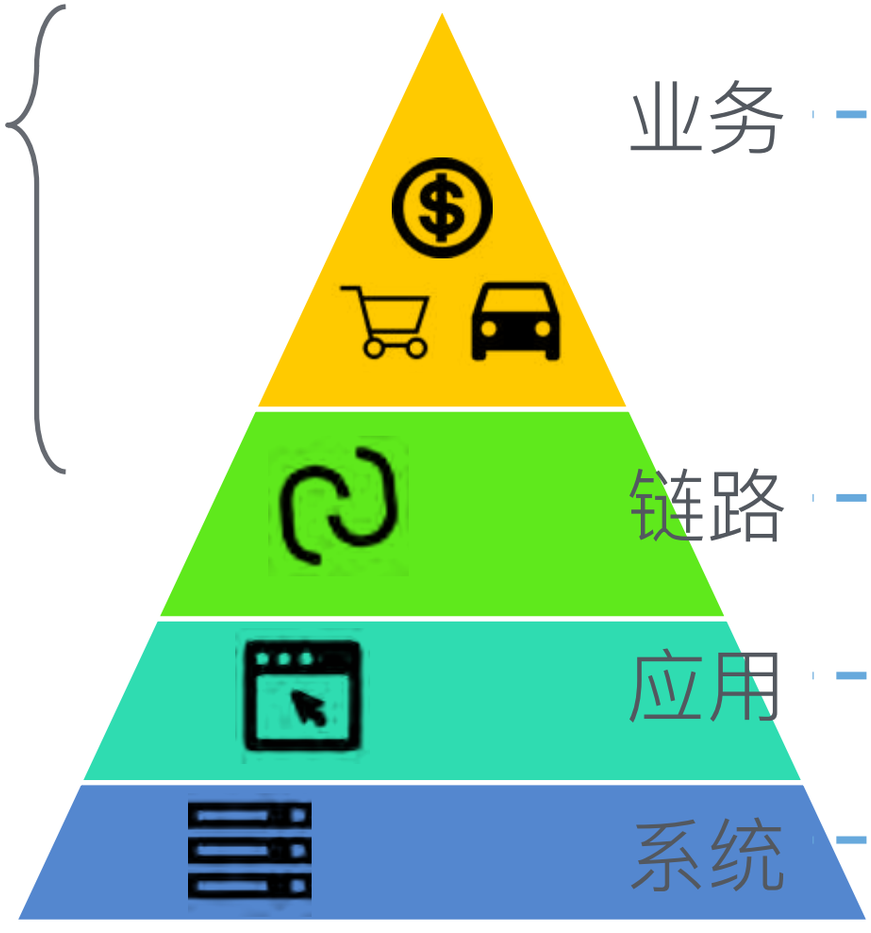
- 云产品EDAS/ARMS
- 全链路根因分析
- SmartLog
- 业务链路自动梳理

鹰眼3.0 – 一个端到端的APM实现

业务运营层面
(ARMS)

应用管控层面
(EDAS)

基础运维层面
(云监控)



电商：
- 店铺销售额变化趋势
- 线上最畅销热门商品

车联网：
- 车辆同时在线率
- 车辆行驶最拥堵区域

零售：
- 各大区域销售额实时统计
- 库存动态变化率

- 调用链全息排查
- 全局调用拓扑
- 链路依赖项的分析梳理

- 应用调用次数，错误占比
- 异常堆栈日志、慢 SQL

- 操作系统：CPU, 内存, IO, 网络
- 组件服务：容器, 数据库, 缓存

Alibaba Group 阿里巴巴集团

THANKS!

阿里技术微信公众号

让创新技术推动社会进步

HELP TO BUILD A BETTER SOCIETY WITH
INNOVATIVE TECHNOLOGIES

Geekbang.
极客邦科技

InfoQ

专注中高端技术人员的技术媒体

EGO EXTRA GEEKS' ORGANIZATION NETWORKS

高端技术人员学习型社交平台

StuQ 斯达克学院

实践驱动的 IT 教育平台

地址：北京市朝阳区洛娃大厦C座8层1801室　　网址：www.geekbang.org