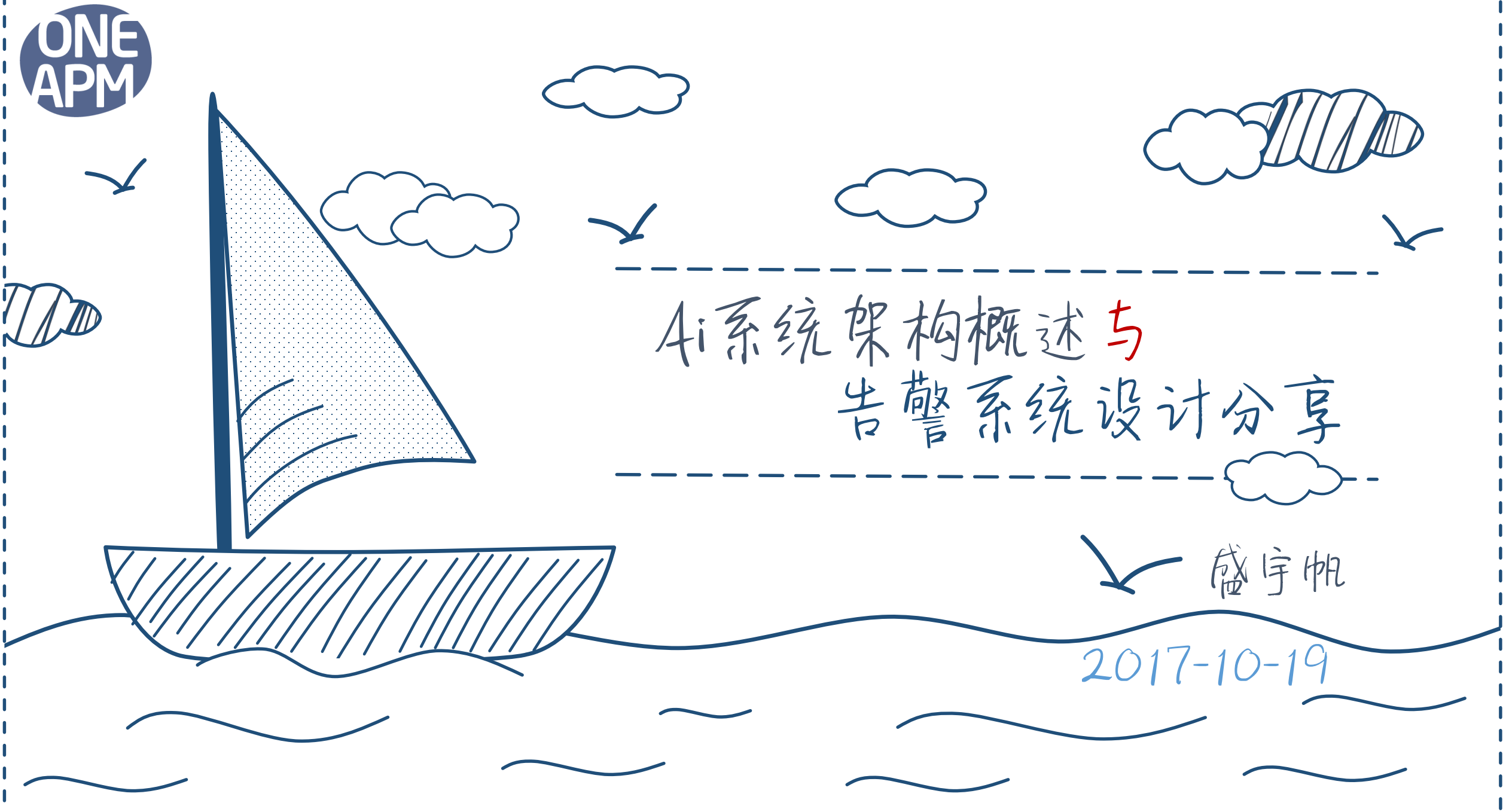


Ai系统架构概述与告警系统设计分享

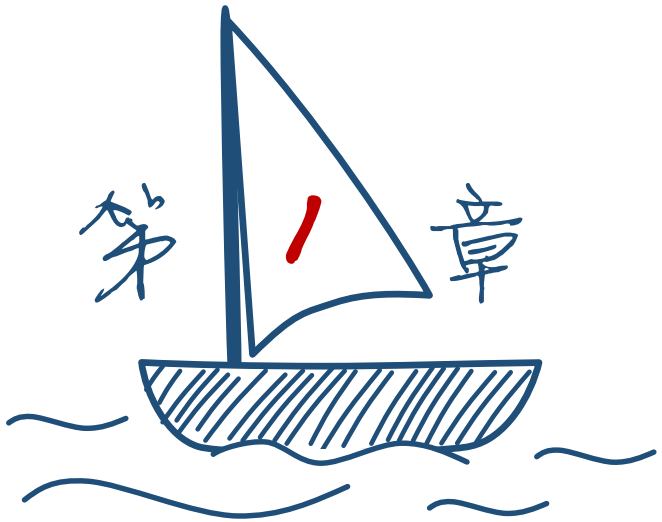
盛宇帆

2017-10-19

目录

content

1. AI设计与架构演进
2. 告警系统设计与架构演进
3. 总结与思考
4. Q&A



Ai设计与架构演进

Ai的整体架构是在不断选型试错和部署需求中迭代至今天的样子，作为一个APM系统，我们走了很多的弯路，但是我们也实现了一些别人没想到的东西。

1.1 Ai 设计变更

Ai企业版里程碑版本

探索时期
Si

2013

3.0

2014

第一个对外版本
存储使用HBase

代码拆分企业版和Saas版
企业版存储改为Druid

2015

4.0

2016

4.1

开发金字塔模块ClickHouse
抽象存储查询部分

SaaS与企业版代码合并
系统服务化，打通各个组件

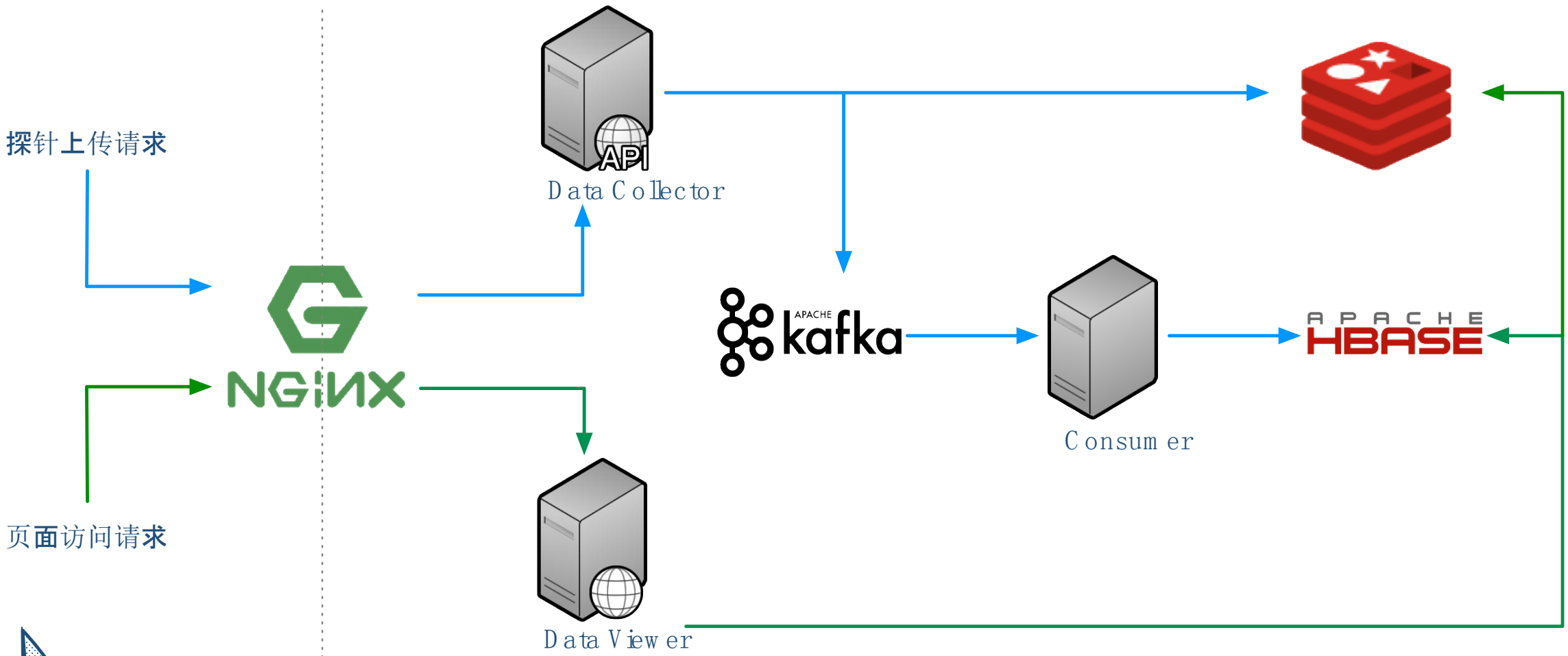
2017

4.2

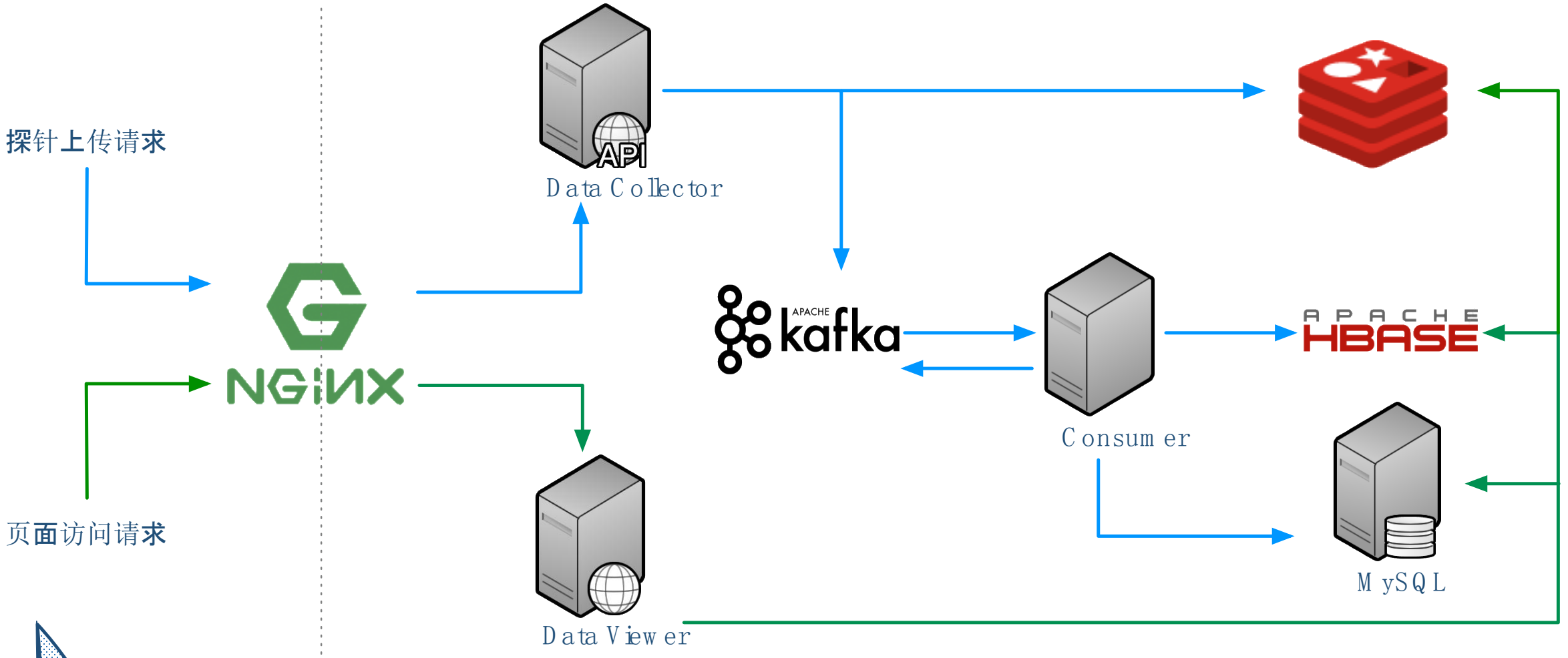
5.0 ?

对于任何一个OLAP系统，存储是其中重中之重，HBase 初期的开发的确十分便捷，但是却无法承担日益增长的数据量。对于 Druid，我们基于它的查询方式进行分析我们的业务系统，发现常见的查询最终都是那么几大类查询方式的组合，在次我们统一了我们存储的数据Schema。为了简化部署，增加架构的可扩展性，同时承担更海量的数据，我们抽象出金字塔模块，并在这个基础上确定不同聚合粒度的数据表，将存储这一层做了深度抽象。

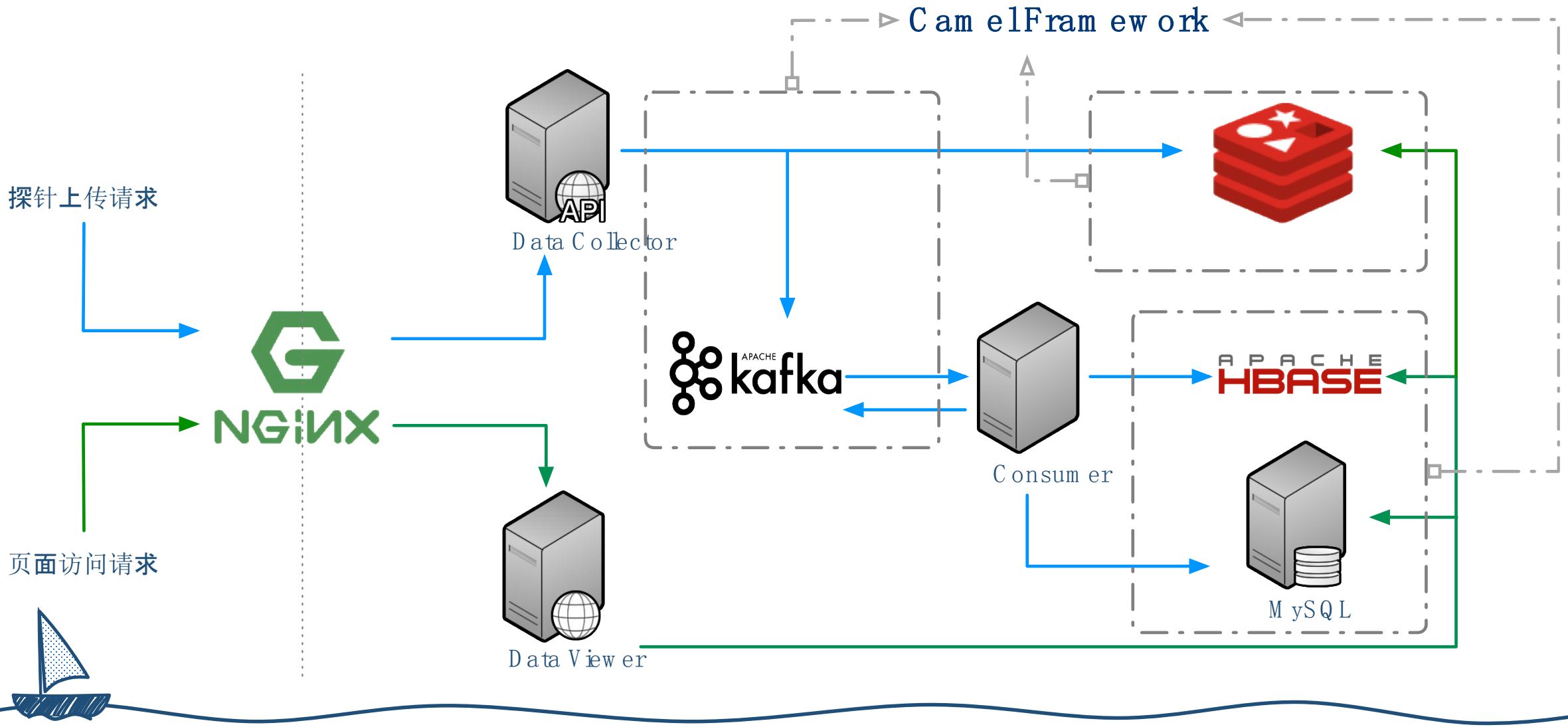
3.0 架构设计


探针上传请求
页面访问请求
NGINX
API
Data Collector
Data Viewer
APACHE kafka
Consumer
APACHE HBASE

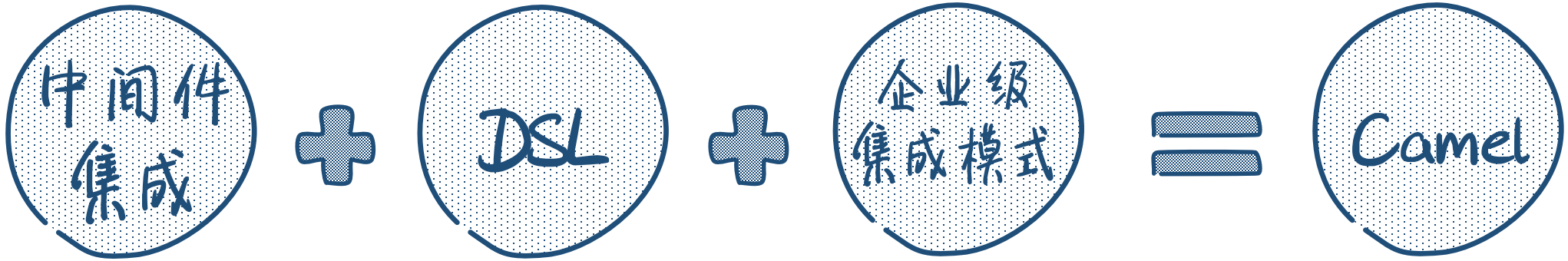
3.0 架构设计


探针上传请求
API
Data Collector
NGINX
APACHE kafka
Consumer
APACHE HBASE
页面访问请求
Data Viewer
MySQL

Camel 中间层


CamelFramework
探针上传请求
API
Data Collector
NGINX
APACHE kafka
Consumer
APACHE HBASE
MySQL
页面访问请求
Data Viewer

Camel 介绍



Apache Camel is a powerful Open Source Integration Framework based on Enterprise Integration Patterns

```
<camelContext xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="direct:foo"/>
    <to uri="log:input"/>
    <f
  </
</ca
/bean
```

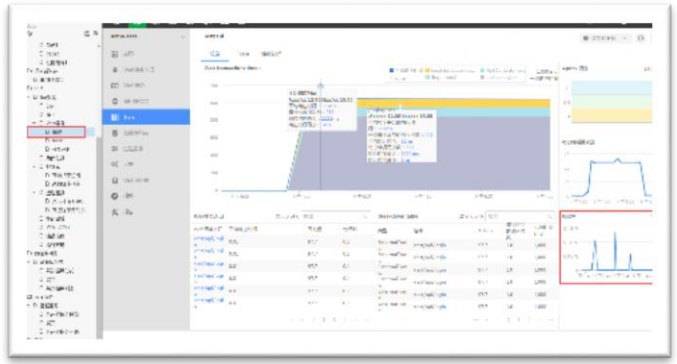
filter	http://camel.apache.org/schema/spring
from	http://camel.apache.org/schema/spring
doFinally	http://camel.apache.org/schema/spring
interceptFrom	http://camel.apache.org/schema/spring
setFaultBody	http://camel.apache.org/schema/spring
transform	http://camel.apache.org/schema/spring

Press ^Space to view tags from other namespaces

Mock Agent 开发

```
// 模拟一个Application mock_demo,别名叫做mock_instance
declare App mock_demo {
    // 在该Application下,声明一个负载为10%的Agent
    declare Agent mock_agent_1 10%
} as mock_instance

declare WebTransaction login duration 3 in mock_demo {
    // 包含另一个已经存在的WebTransaction dispatch
    dispatch
    // 声明一个叫做Java/checkLogin,持续3ms的子WebTransaction
    Java/checkLogin    duration 3
    // 声明一个叫做Java/register,持续1min的子WebTransaction
    Java/register    duration 1min
    // 声明一个叫做Java/Other,持续5ms的子WebTransaction 测试中文乱码
    Java/Other    duration 5s
} as login
```



定义DSL，编写应用数据模板

解析模板自动生成
数据发送至 DC

Ai 处理后展示结果

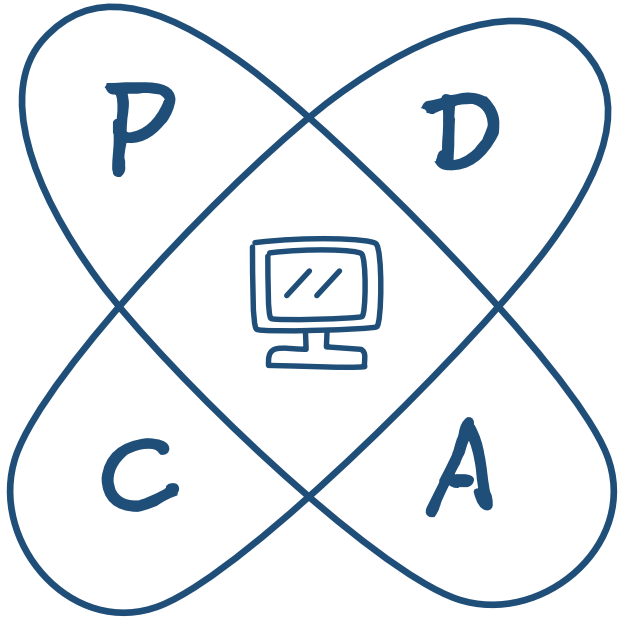
HBase版问题

performance

在大数据量写入查询时，有比较高的时延，性能表现不好，易OOM。

complex

需要使用 Redis 来缓存查询结果，为何保证数据时效性，缓存失败的缓存风暴等问题，写了很多不必要的非业务代码。



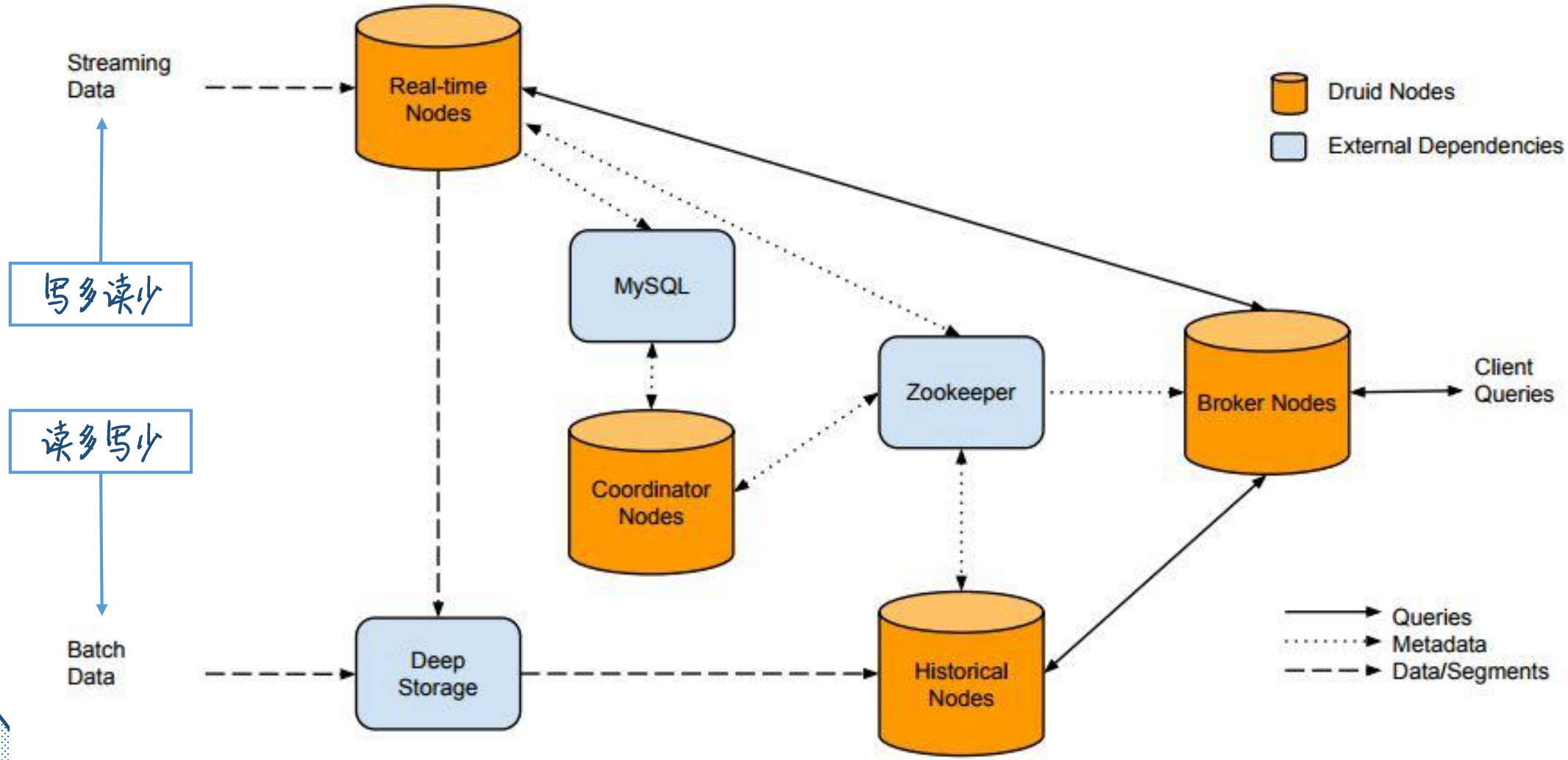
distribution

即使使用Camel，也只是某种程度上简化了配置问题，但并未简化部署。

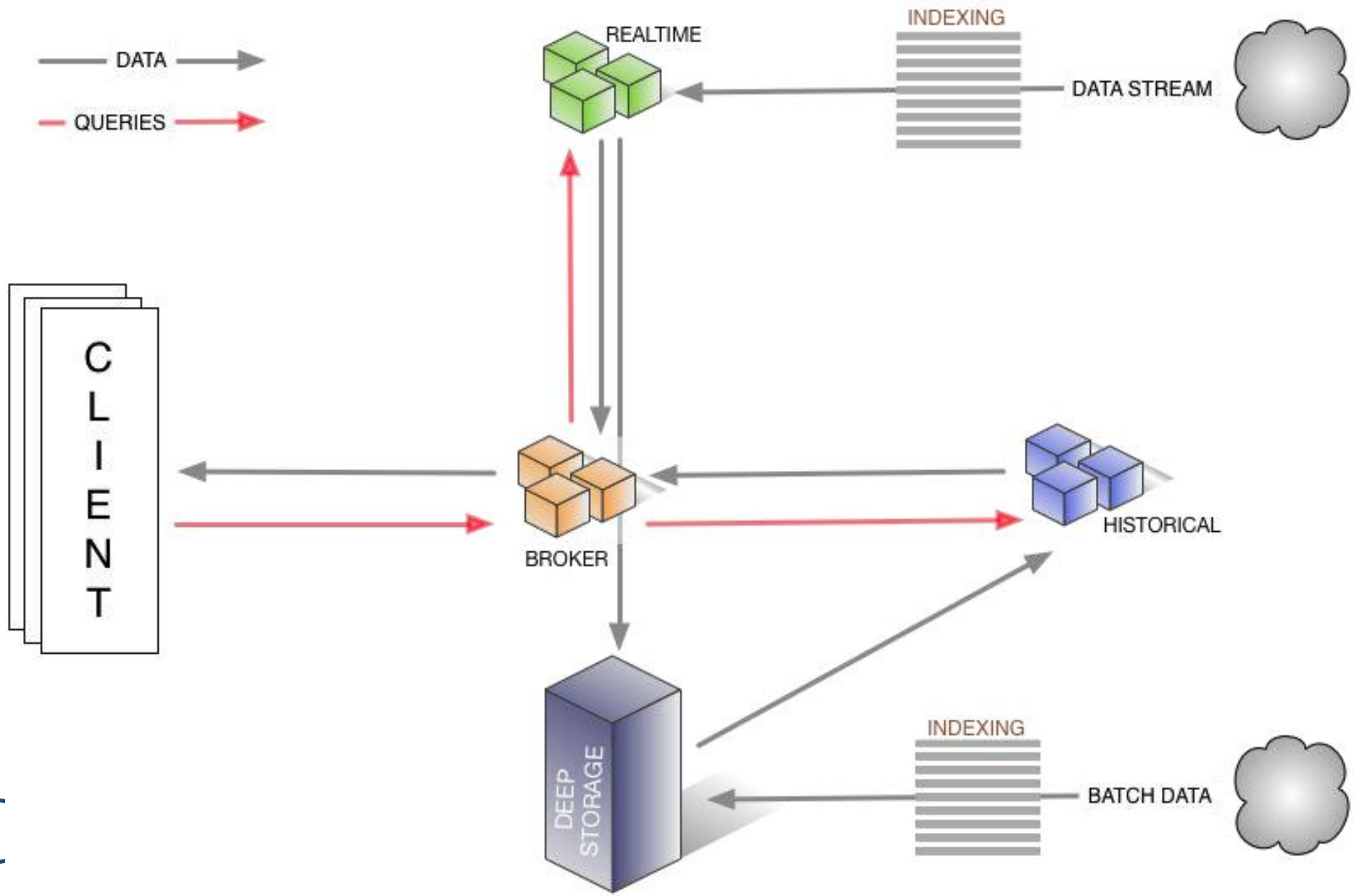
Availability

因为对 Camel 的使用不当，部分组件还是单节点。

Druid 架构


Streaming Data
Real-time Nodes
MySQL
Coordinator Nodes
Zookeeper
Broker Nodes
Client Queries
Druid Nodes
External Dependencies
写多读少
读多写少
Batch Data
Deep Storage
Historical Nodes
Queries
Metadata
Data/Segments

Druid 数据存储


DATA
QUERIES
REALTIME
INDEXING
DATA STREAM
CLIENT
BROKER
HISTORICAL
DEEP STORAGE
INDEXING
BATCH DATA

OLAP的查询方式

Timeseries

基于时间维度的范围查询。没有其他特定维度，返回该时间范围内的全部数据集。Druid针对此场景做了特殊优化，提升计算时间。

Time Boundary

查询最新和最旧的数据的边界。

TopN

针对某一维度的头多少条的查询，只支持单一维度。

GroupBy

Druid的主要查询方式，最全面的参数选择，包含维度选择、统计指标选择、过滤条件、时间范围及时区、排序方式、结果集返回大小，数据集选择。支持 count, longsum, doublesum, min, max, js 等。

Druid 使用与优化

问题

我们使用 Druid 的时候，查询只支持 HTTP JSON，报文也是 JSON 格式。这样在查询条件构造和结果解析上，会有过多的额外工作。

Druid SQL

使用 antlr4 定义和实现了前面提到的 4大类我们需要的查询的 SQL 语句。并基于对应的语法解析，编写了对应的 jdbc 驱动，能将 SQL 查询转为 HTTP 请求，并将结果转换为对应的数据实体。

Druid 使用与优化

问题

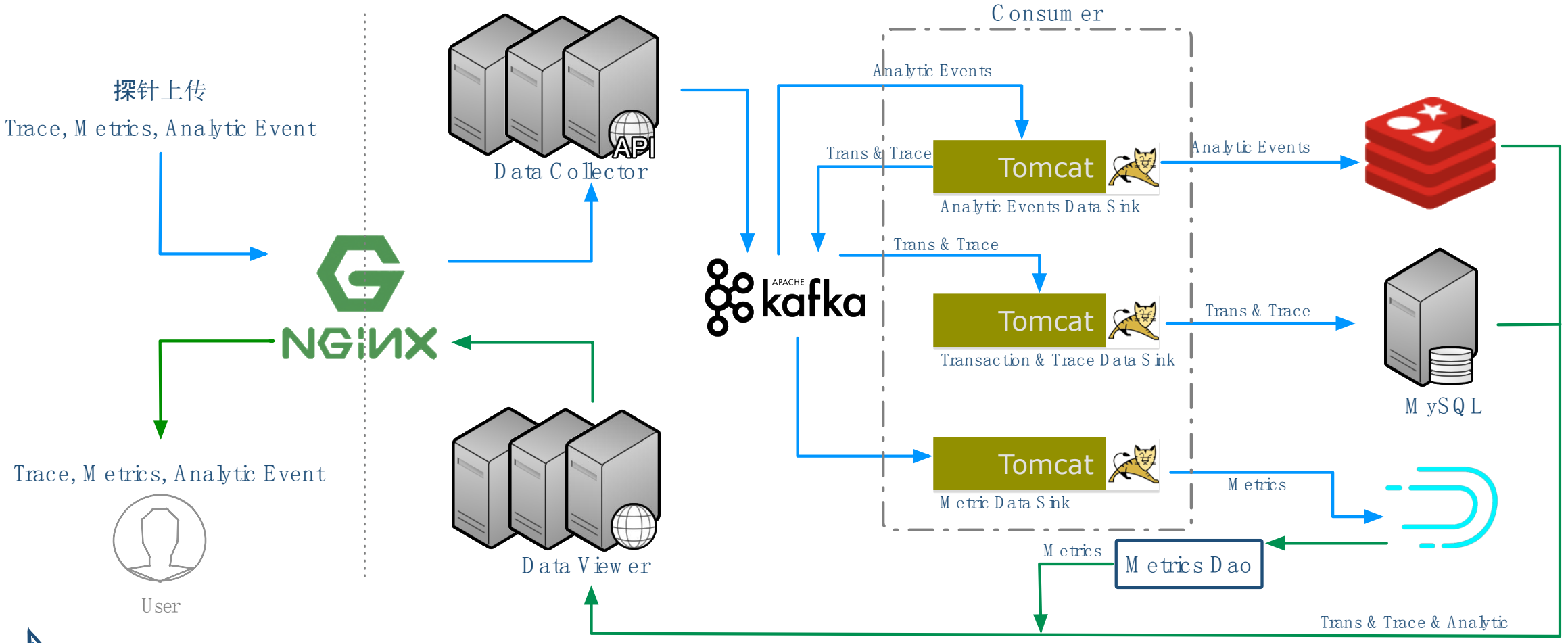
Druid对硬件资源要求较高，硬盘至少SSD。但部分交付用户只能提供普通硬盘，部分云主机上即使是 SSD，IO 也一般。

按照时间粒度进行聚合，减少数据量

时间窗口	折线上点的间隔	折线上点的个数	数据粒度
30min	1min	30	1min
1h	2min	30	1min
3h	5min	36	1min
6h	10min	36	10min
12h	20min	36	10min
1day	1h	24	1h
3day	2h	36	1h
7day	6h	28	1h
15day	12h	30	1h
1month	1day	30	1day
6month	6day	30	1day
12month	10day	36	1day

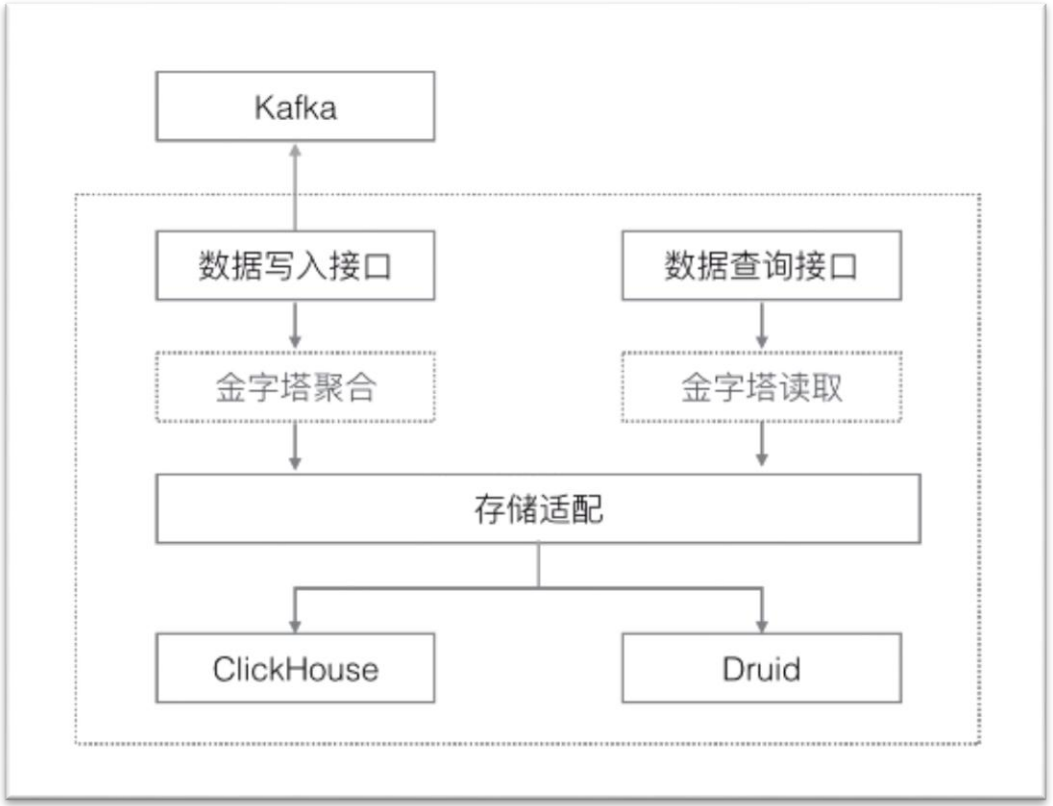
1.3 Ai 4.0 Druid 版

4.0 架构图



金字塔模块

Pyramid 设计上是为了兼容多种不同类型的数据库(如 ClickHouse、MySQL 等)，允许通过配置指定数据的来源和存储位置，实现时序数据的读写。同时提供了一个抽象的查询适配层，支持从不同类型的数据库中以一致的方式查询时序数据，并进行多维组合分析。

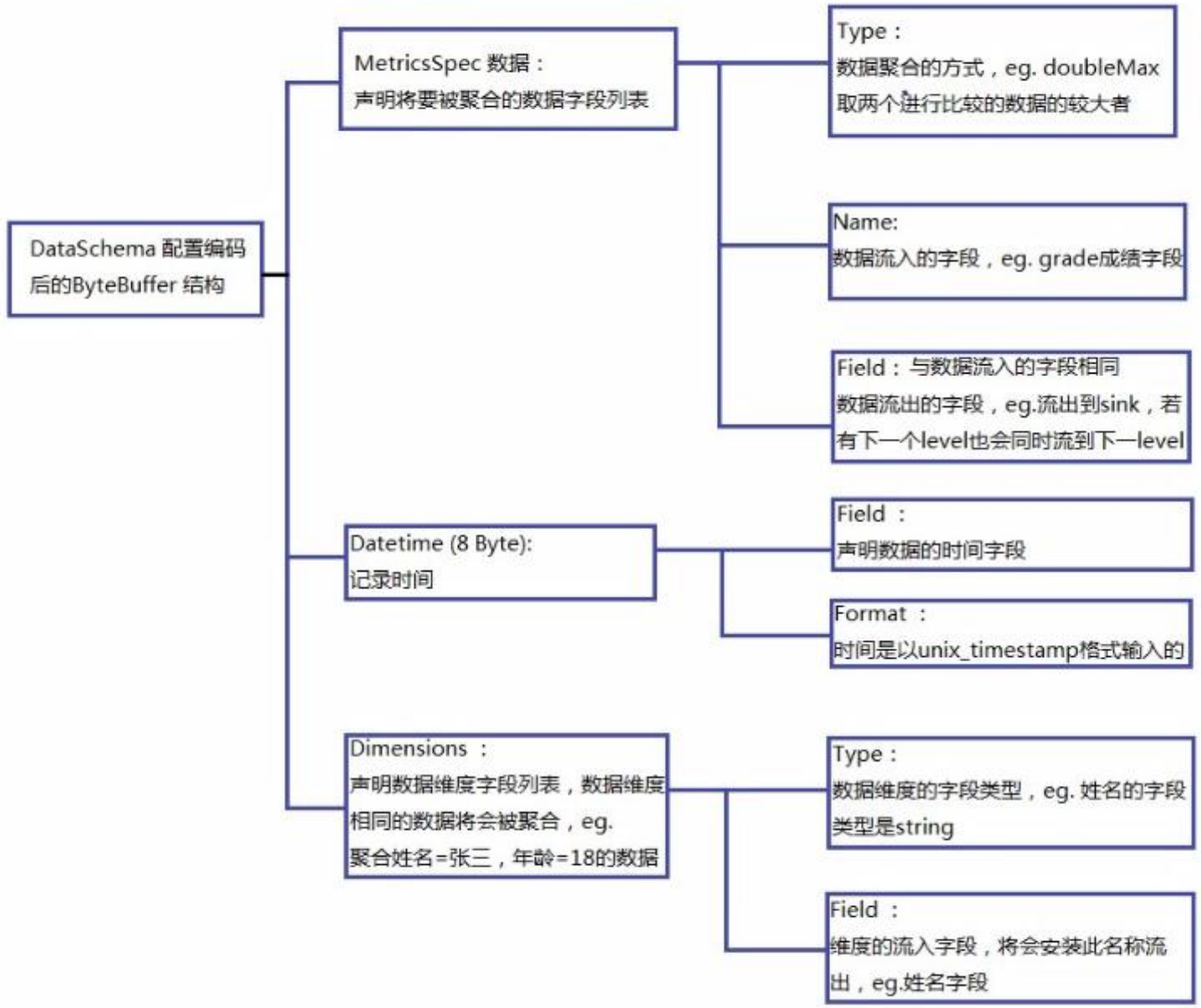


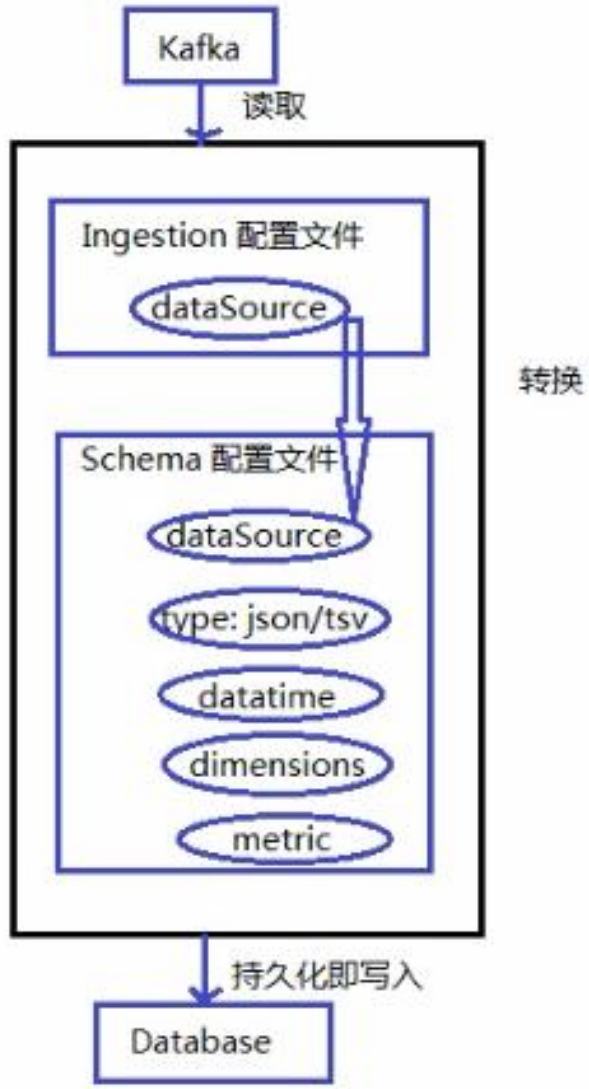
主要功能如下：

1. 时序数据的持久化（即写入功能）： 将指定的数据源（如 Kafka）中的时序数据读取并写入到不同类型的数据库中（如 ClickHouse、MySQL 等）。
2. 查询时序数据： 通过抽象的查询适配层，实现对不同类型数据库中的时序数据的统一方式查询，主要支持三类查询：时序查询 (timeseries query)、topN 查询和 groupBy 查询。

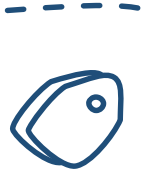
1.4 Ai 4.1 金字塔存储

金字塔模块



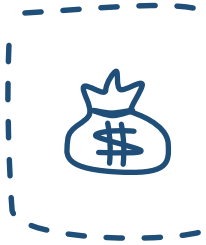


Druid shortcoming



丢数据

过时数据直接丢弃，无法存储。

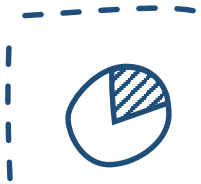


无法更新

Druid 无法删除和更新数据，遇到脏数据就会很麻烦。

部署问题

架构复杂，部署不简单。对于内存和磁盘等硬件要求高。

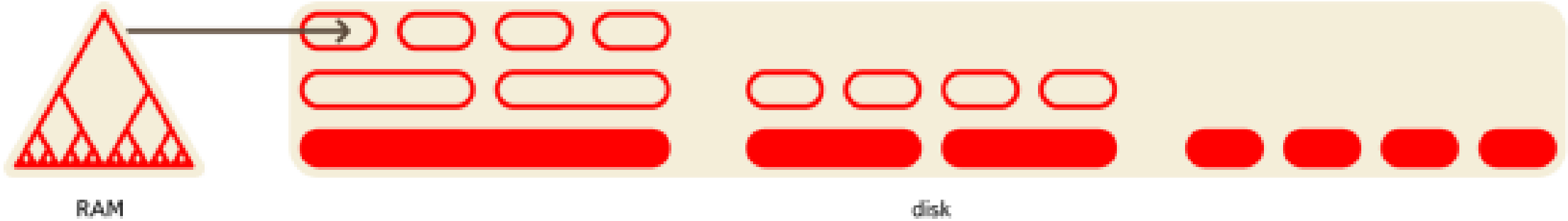


空数据处理

无法判断查询出来的聚合指标为0时的实际结果。

B+ Tree to LSM-Tree

B+ Tree 写入的数据在磁盘中的存储图示



HBase Cassandra LevelDB Druid 等使用的 LSM-Tree

1.5 新的存储 ClickHouse

No-aggregated ClickHouse

Replace/update records

- ReplacingMergeTree
- CollapsingMergeTree

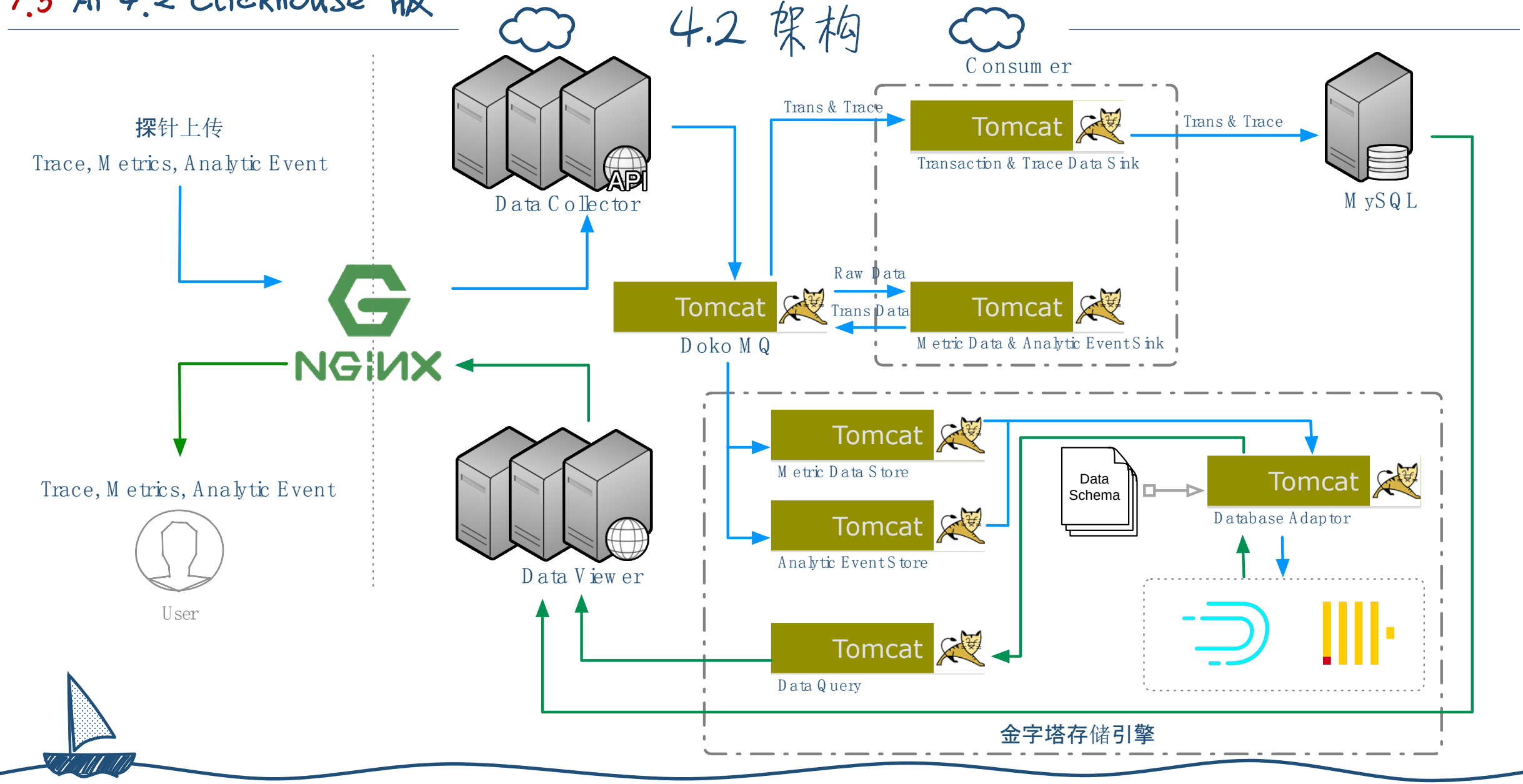
Pre-aggregate data

- AggregatingMergeTree

Metrics rollup

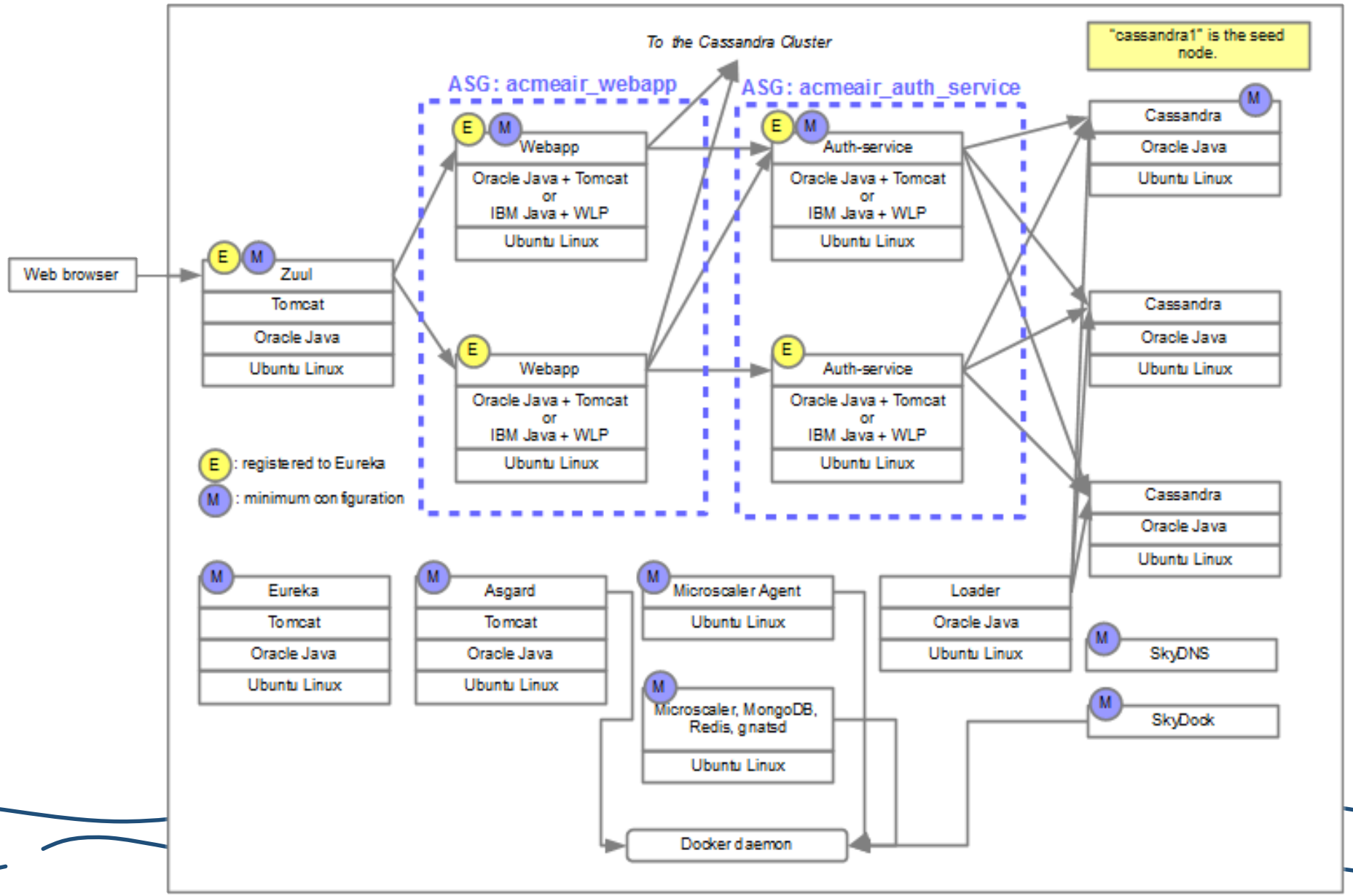
- GraphiteMergeTree

1.5 Ai 4.2 Clickhouse 版

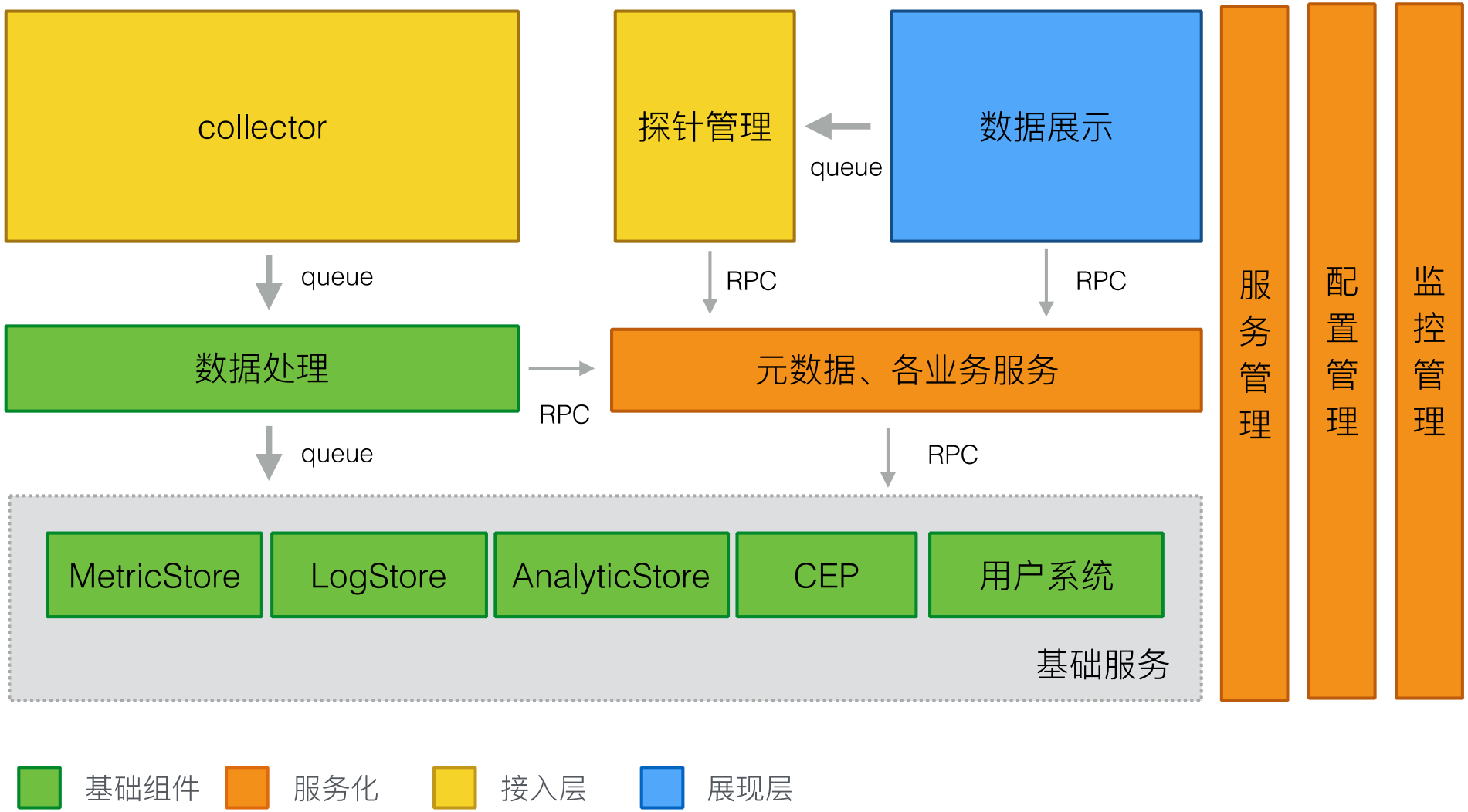


应用部署方式变迁

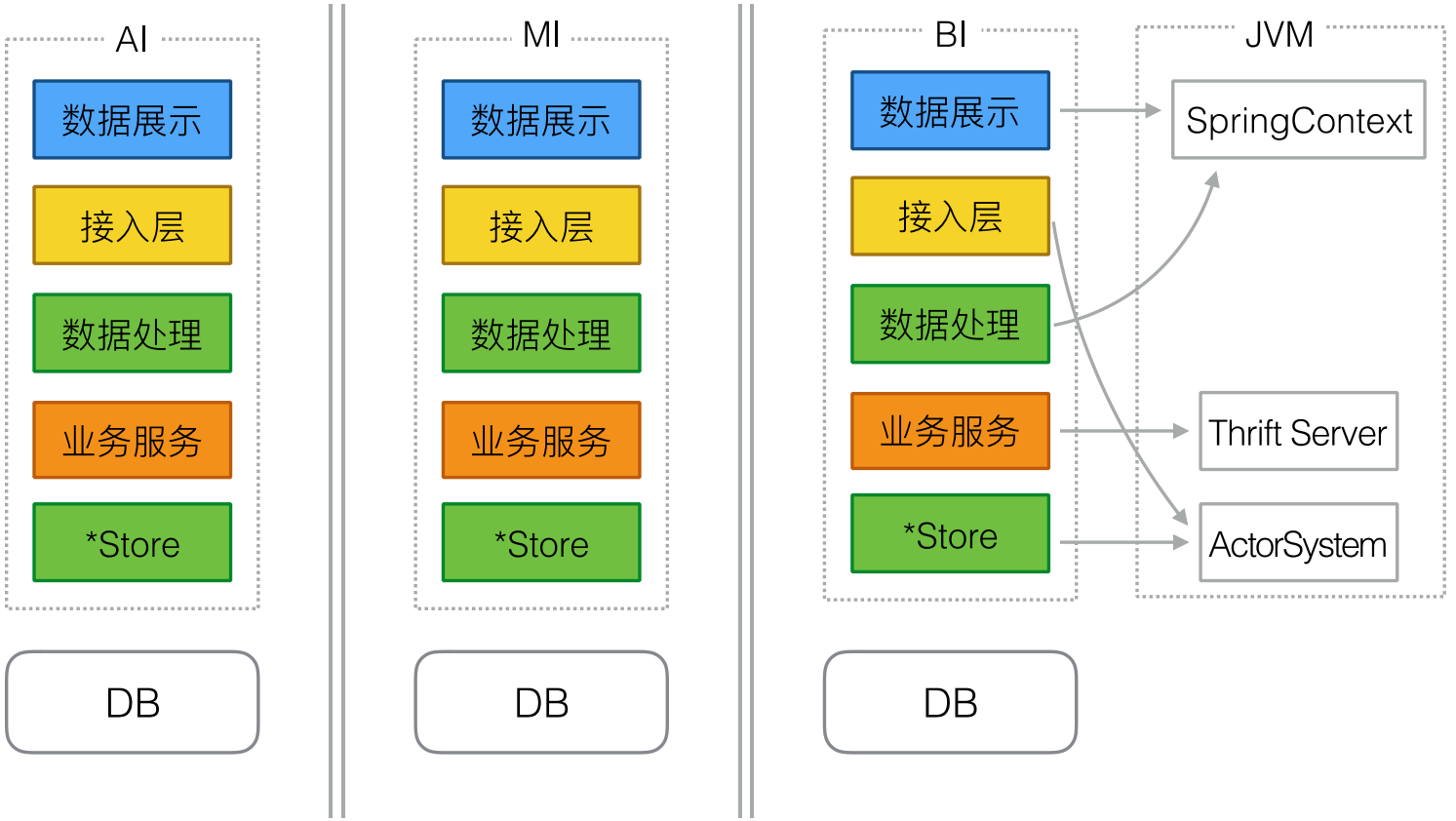
容器化演进


To the Cassandra Cluster
"cassandra1" is the seed node.
ASG: acmeair_webapp
ASG: acmeair_auth_service
Web browser
Zuul
Tomcat
Oracle Java
Ubuntu Linux
Webapp
Oracle Java + Tomcat
or
IBM Java + WLP
Ubuntu Linux
Auth-service
Cassandra
E: registered to Eureka
M: minimum configuration
Eureka
Asgard
Microscaler Agent
Loader
SkyDNS
SkyDock
Microscaler, MongoDB, Redis, gnatsd
Docker daemon

应用服务化


collector
探针管理
queue
数据展示
queue
RPC
RPC
数据处理
RPC
元数据、各业务服务
queue
RPC
MetricStore
LogStore
AnalyticStore
CEP
用户系统
基础服务
服务管理
配置管理
监控管理
基础组件
服务化
接入层
展现层

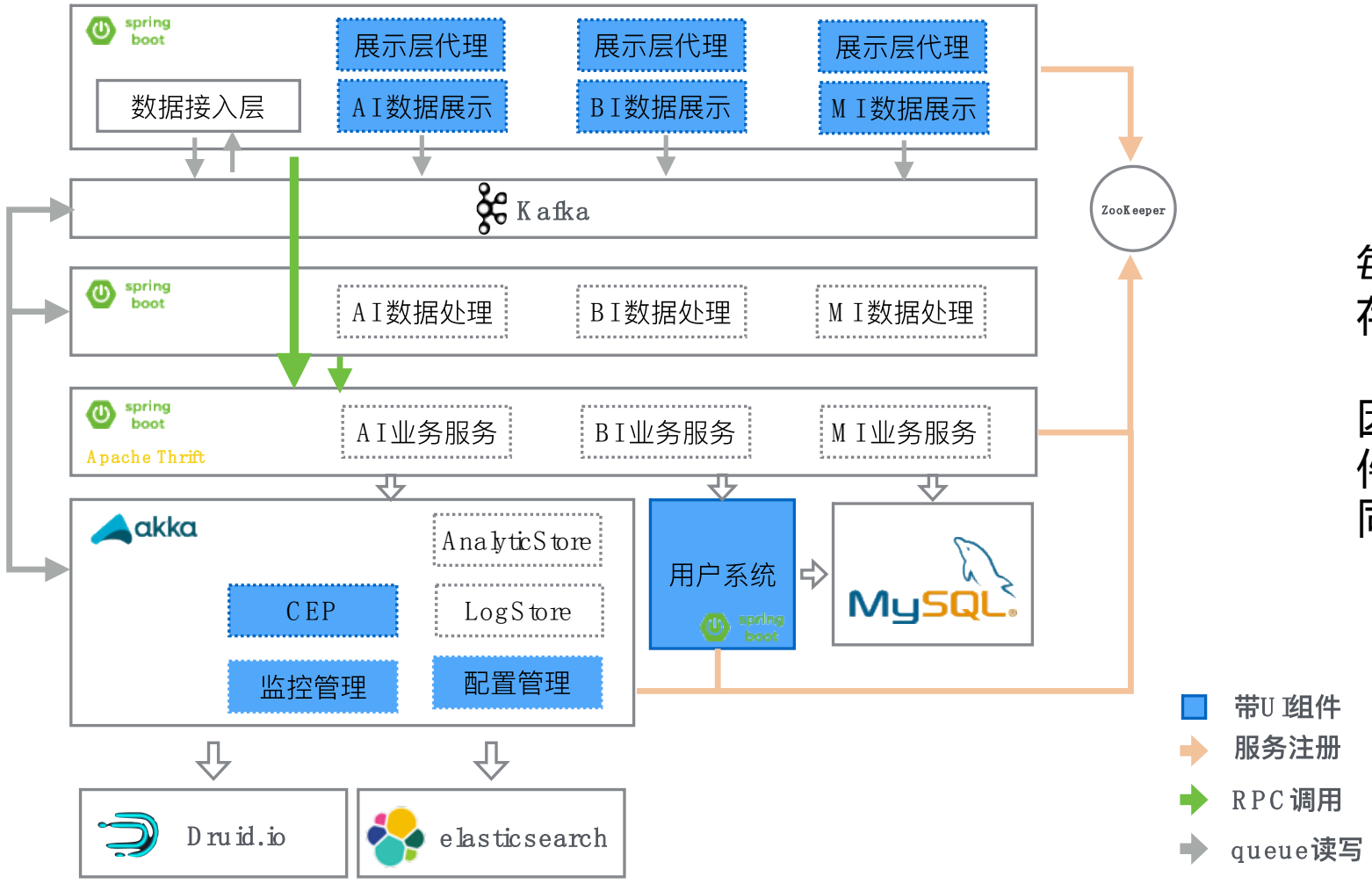
最小化部署



最小化部署，就是所有的服务实例在一起部署，不分组，每个产品单独部署所需服务。

1.6 部署与交付

弹性部署



每个业务按照相同程序模块分组，存储、消息队列等均共享。

因为这些组件均已实现自有中间件，可以按照数据量不同选用不同的存储。

经验与教训

部署环境不同但代码需唯一

企业级和SaaS存在不同的部署需求、数据量，但不能人为地分成多个分支维护。

中间件不一定是解耦的银弹

Camel 的引入并没有减少开发量，简化部署压力，在改变架构选型的时候，甚至是畔脚石。

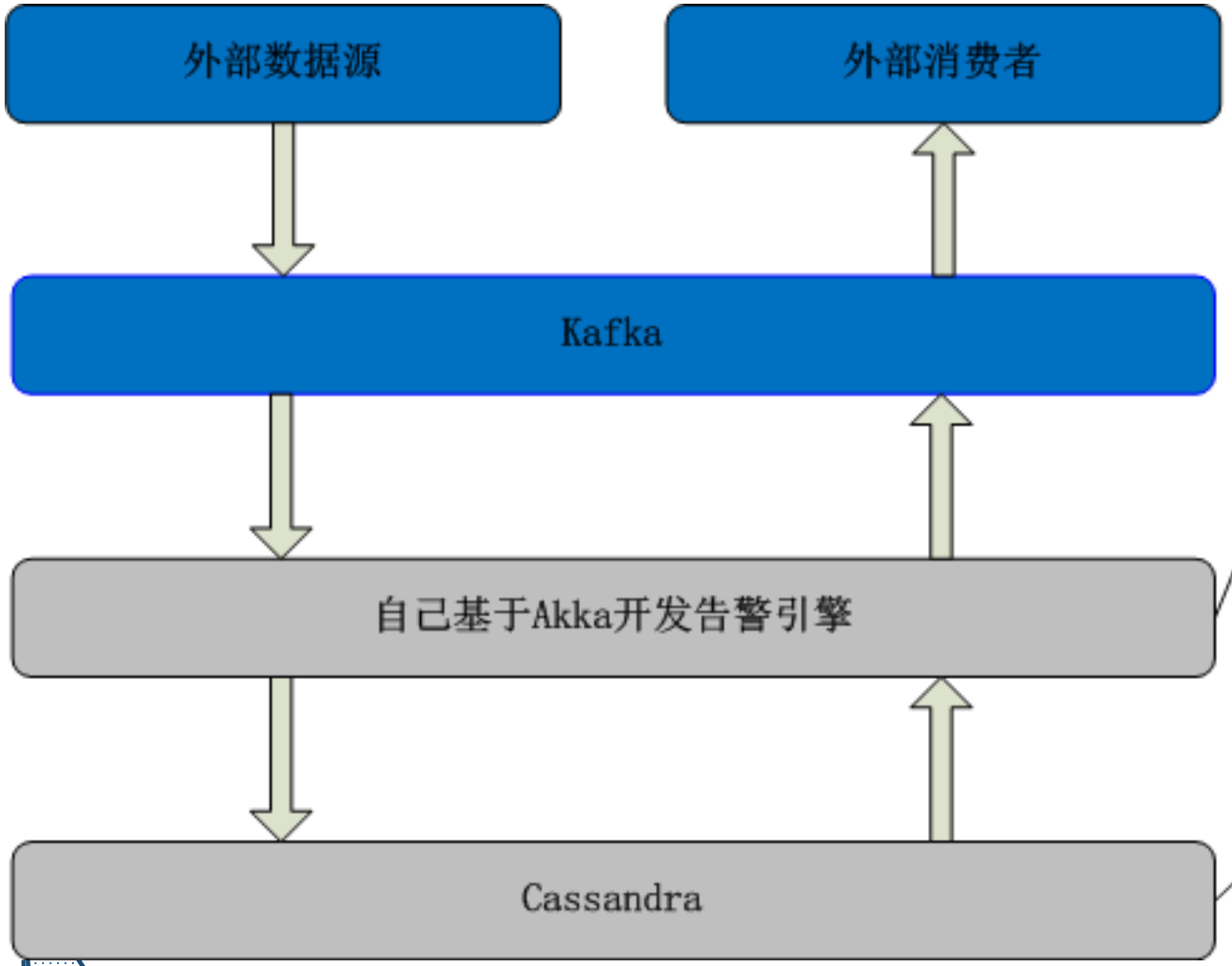
DSL一定程度上可以简化开发

MockAgent、Druid SQL 等 DSL 的引入，减少了大量开发和测试中的重复劳动。虽然性能下降。

告警系统设计与架构演进

由复杂到简单，从最基础的规则抽象到实现灵活计算需求，我们实现了最基础的计算引擎，并基于它演变出现在的告警系统。

2.0 告警系统立项与调研 初版告警服务架构



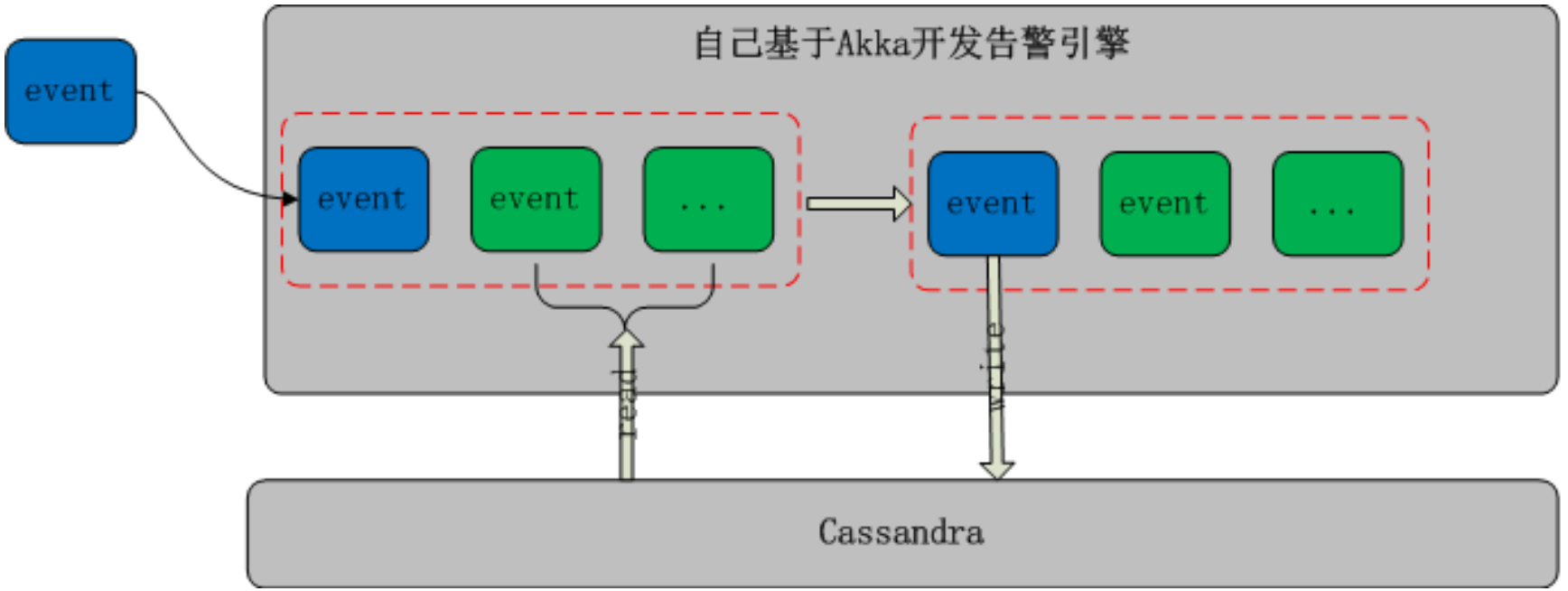
存在的问题：

- 分布式部署困难(并不是在多节点运行就是分布式)
- 没有分布式路由功能(同一个告警策略的数据不能保证在一起，完全依赖外部存储)
- 对存储造成巨大压力(来1条数据，查1批数据，写1次数据库，窗口数据依赖外部存储)

存在的问题：

- 读写压力剧增(由于引擎功能缺失而转移的压力)

2.0 告警系统立项与调研 初版告警读写细节



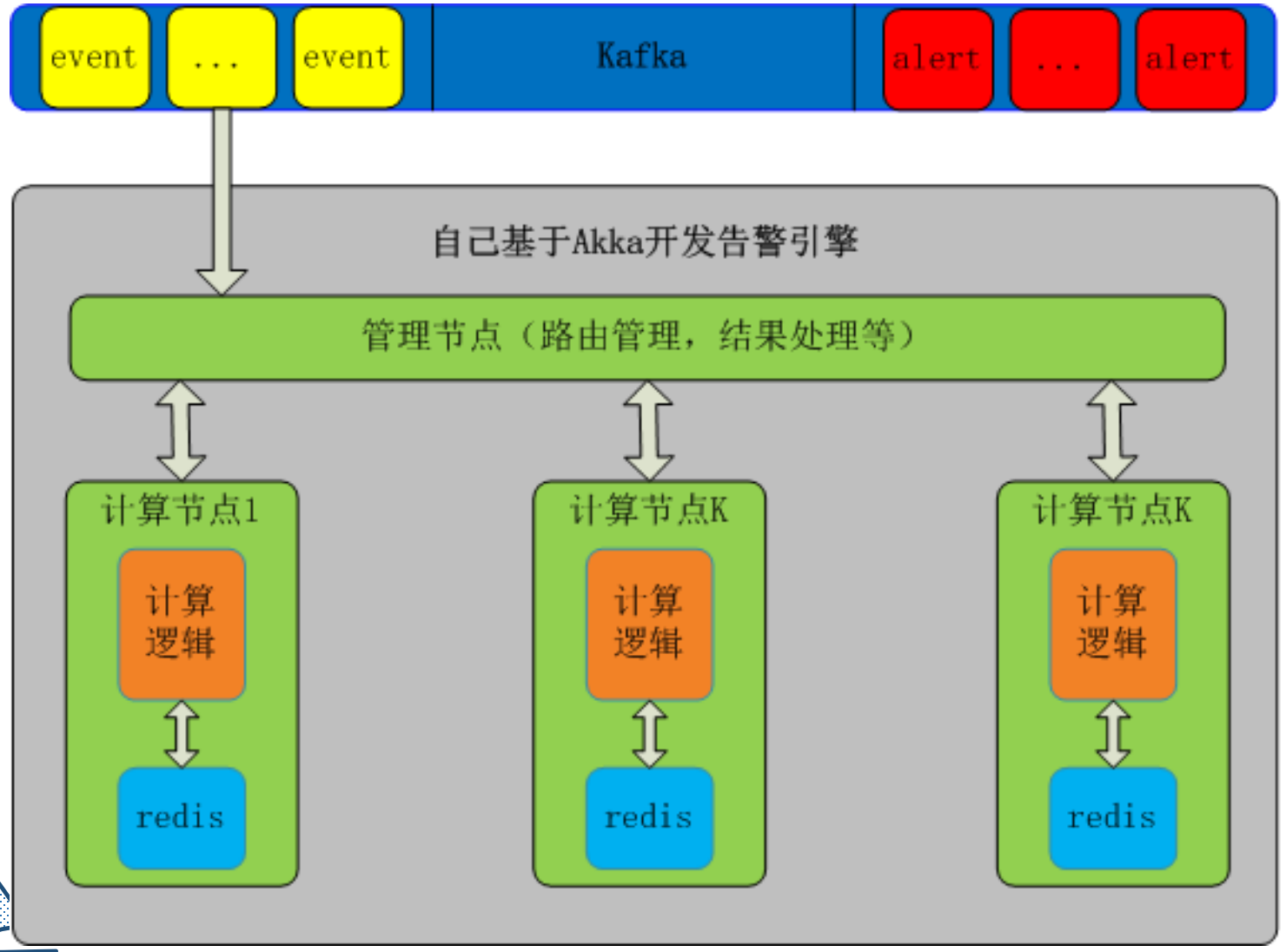
- 引擎靠外部存储实现窗口数据存取
- 数据写入没有优化（无写缓存，类比HBase那个表，读写压力大得多）

2.0 告警系统立项与调研 对比 Ai 3.0 存储

比较	场景	写	读
Ai HBase	写多读少	准实时写，写优化(批量写入，write buffer)	用户不可能在同一时刻都来查，且每个用户每天也查不了几次。好多用户都不带查的，咱们网站上的pv也才5千多
本解决方案	写多读多	实时写，单条写入	实时查，每来一个event都要查询一个窗口的数据

每个event对应一个查询和写入操作，读操作比Ai HBase多了可不是一星半点。

2.0 告警系统立项与调研 初版告警架构优化



改造内容：

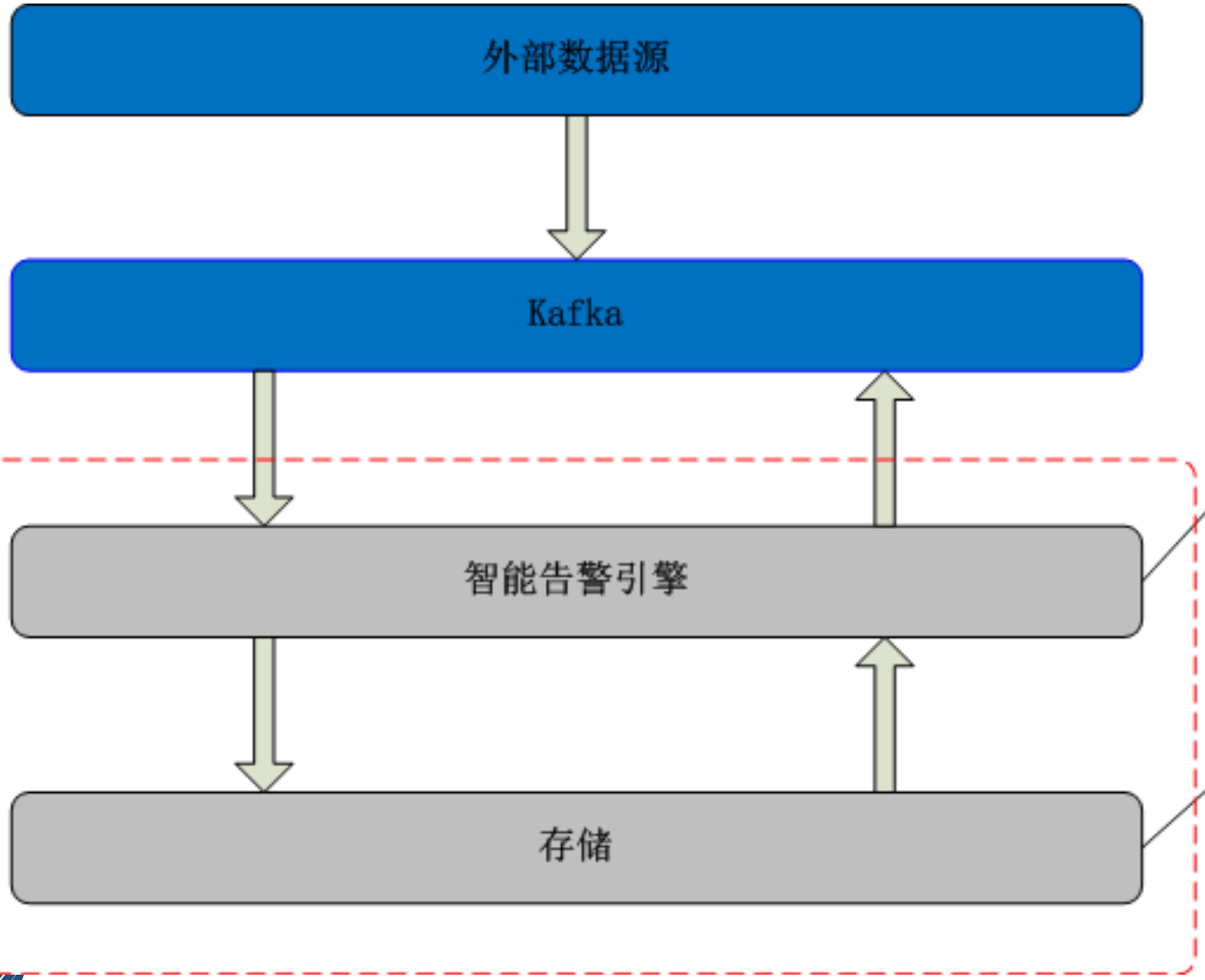
- 分布式改造(管理节点，计算节点)
- Redis实现高速读写

反思：

- 单点问题(管理节点、计算节点、redis)
- 高可用问题
- 重复造轮子(自己开发流式计算框架)

2.0 告警系统立项与调研

告警技术架构调研

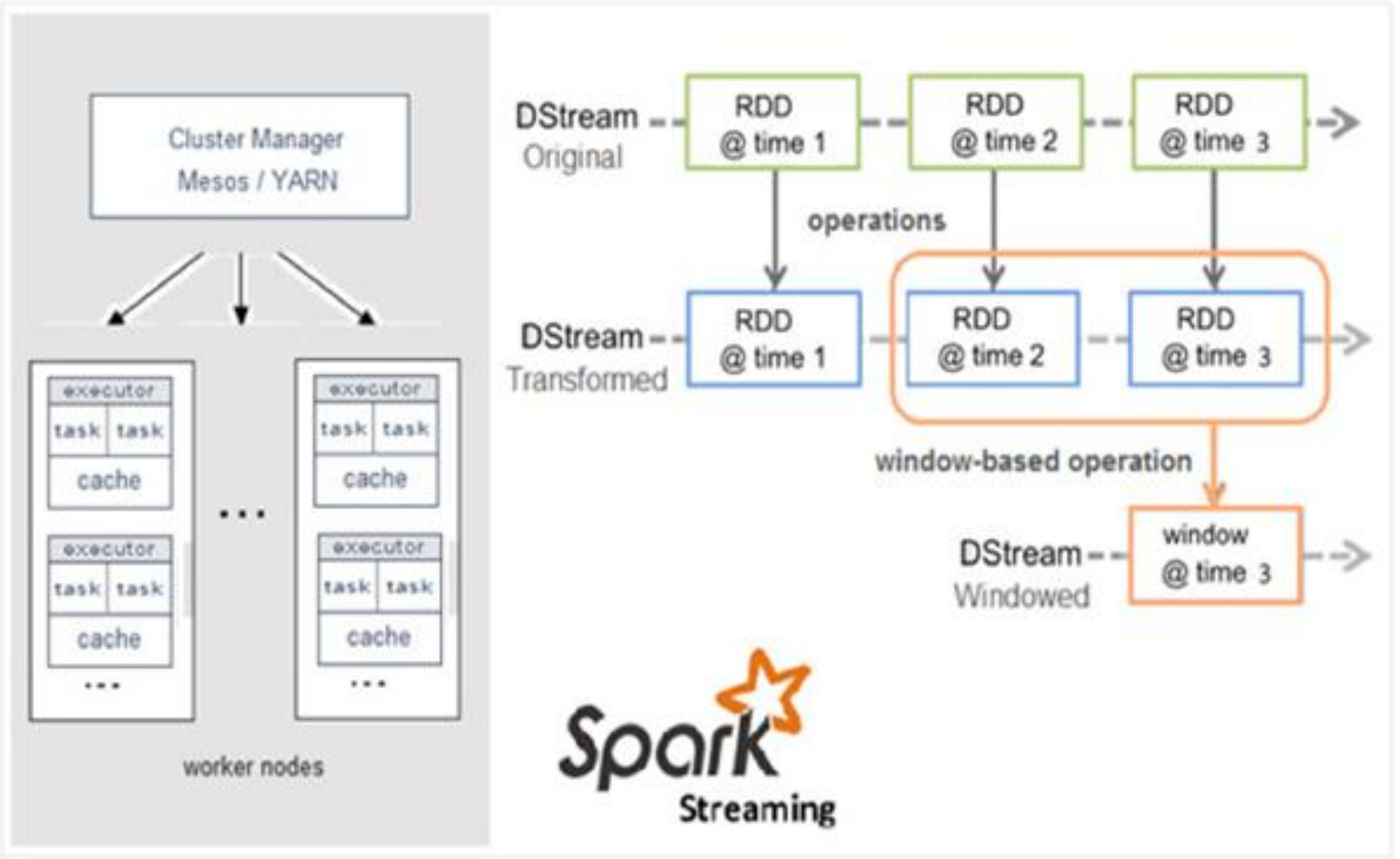


关注点：
- 实时流式计算(单条)
- 分布式(横向线性可扩展)
- 分组路由
- 简单强大的编程模型

关注点：
- 实时高并发存取
- 分布式

2.0 告警系统立项与调研

基于Spark Streaming的解决方案



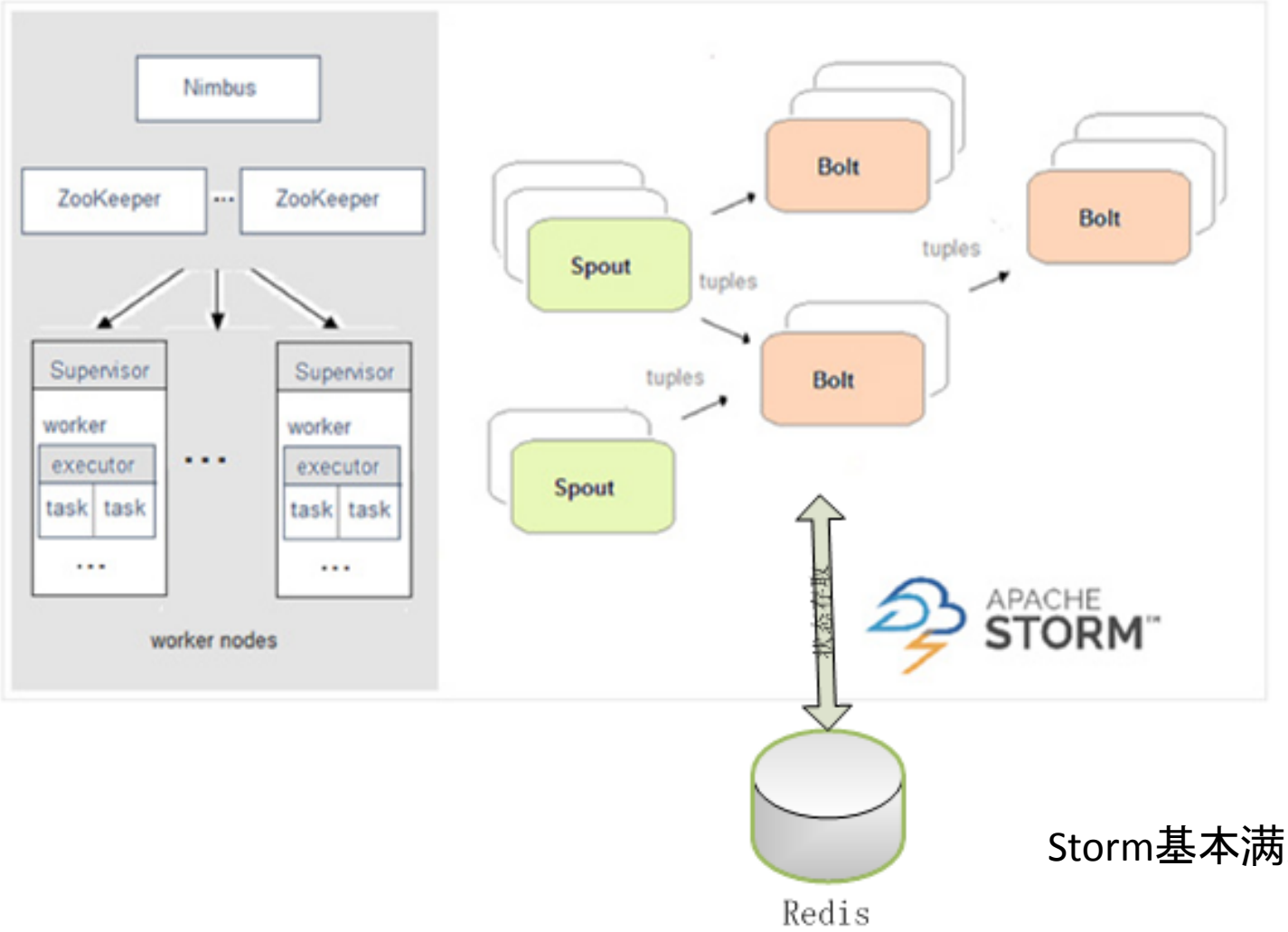
无法解决的问题：

- 无法处理真正的流失计算(微批量处理)

Spark Streaming基本不能满足我们的场景。

2.0 告警系统立项与调研

基于Storm的解决方案



相对于Spark Streaming的优势：

- 真正的实时流计算

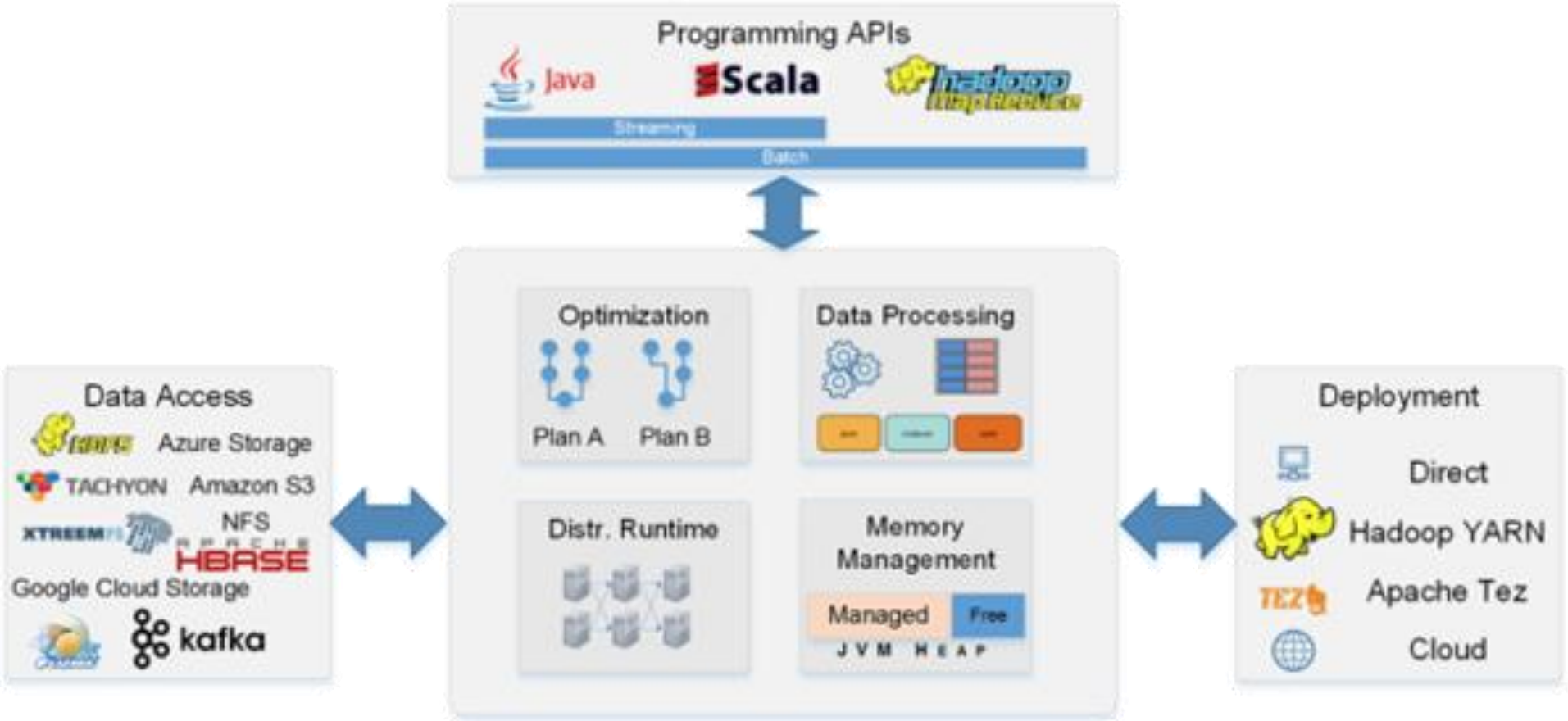
缺点：

- 无状态，窗口数据还是依赖外部存储

Storm基本满足使用场景，但是不是最优的。

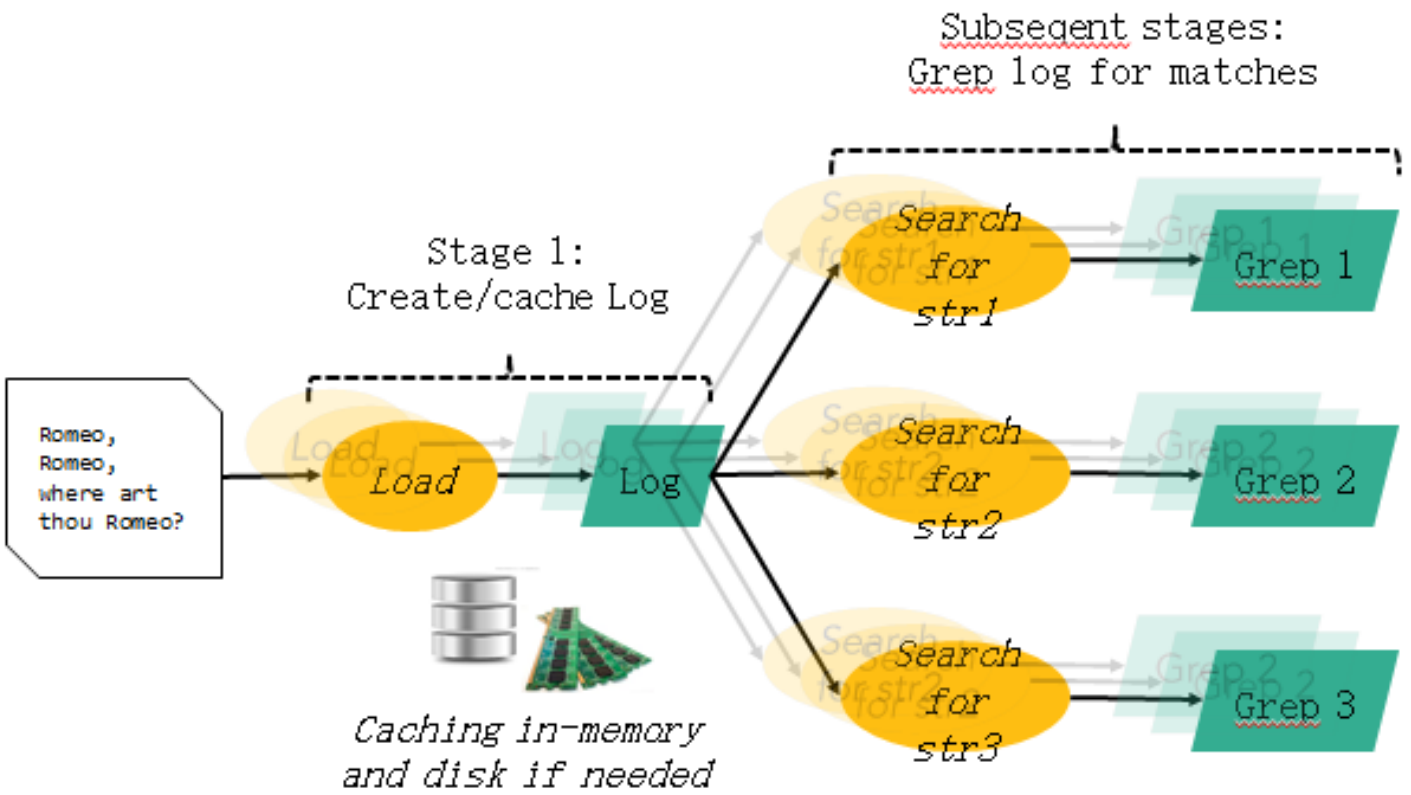
2.0 告警系统立项与调研

基于Flink的解决方案

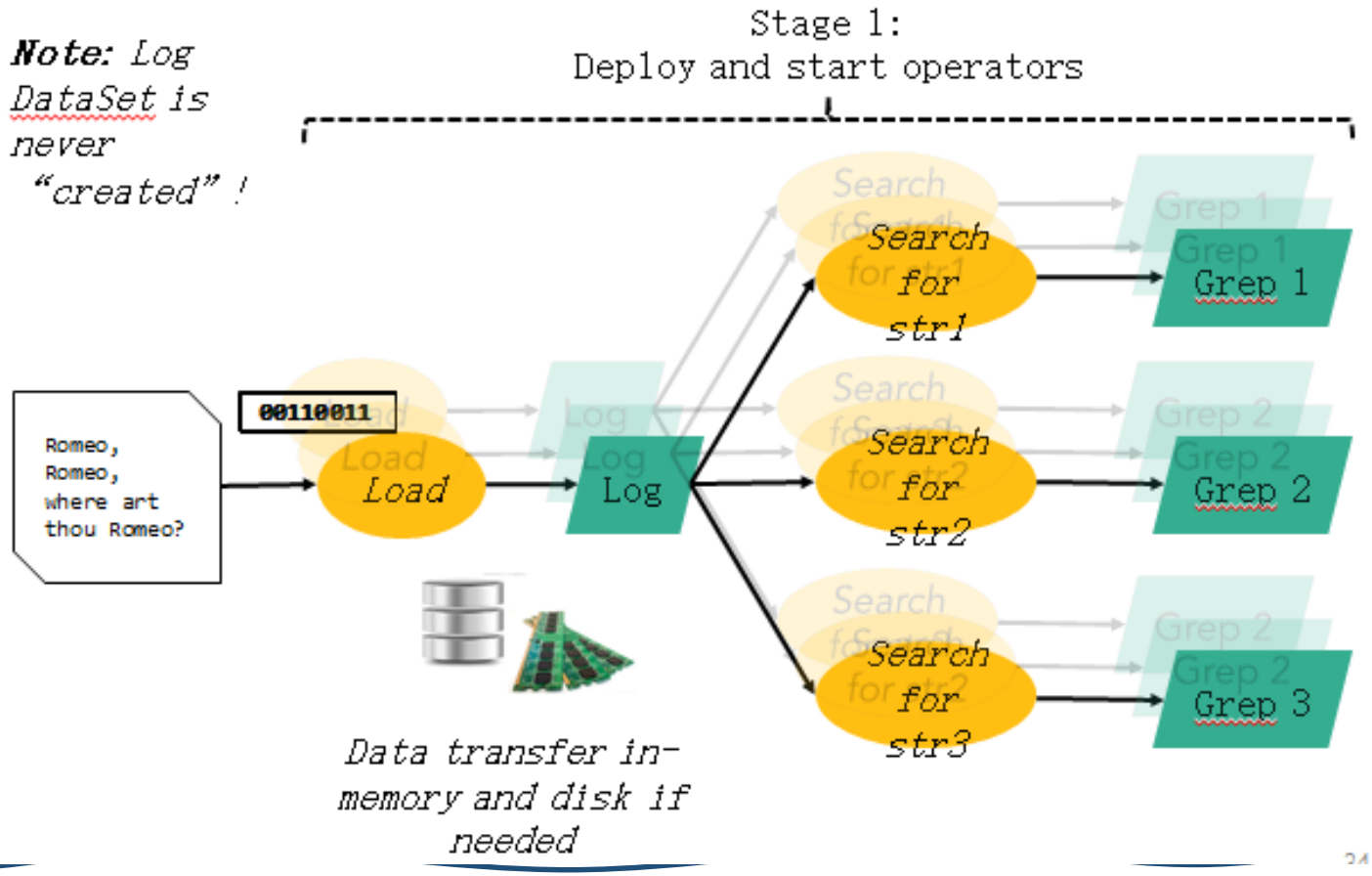


Stage 模式

Staged (batch) execution



Pipelined execution

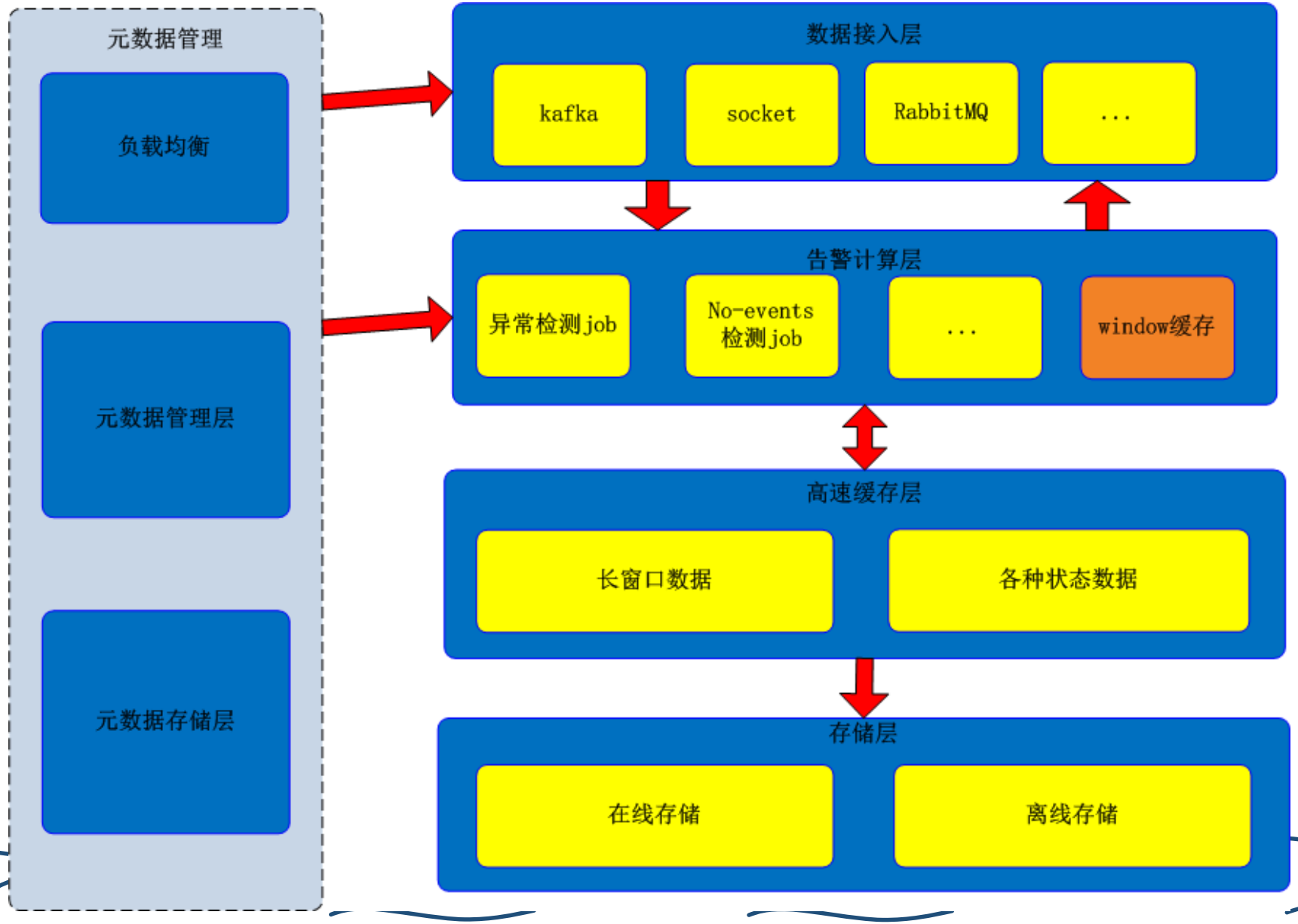


2.0 告警系统立项与调研

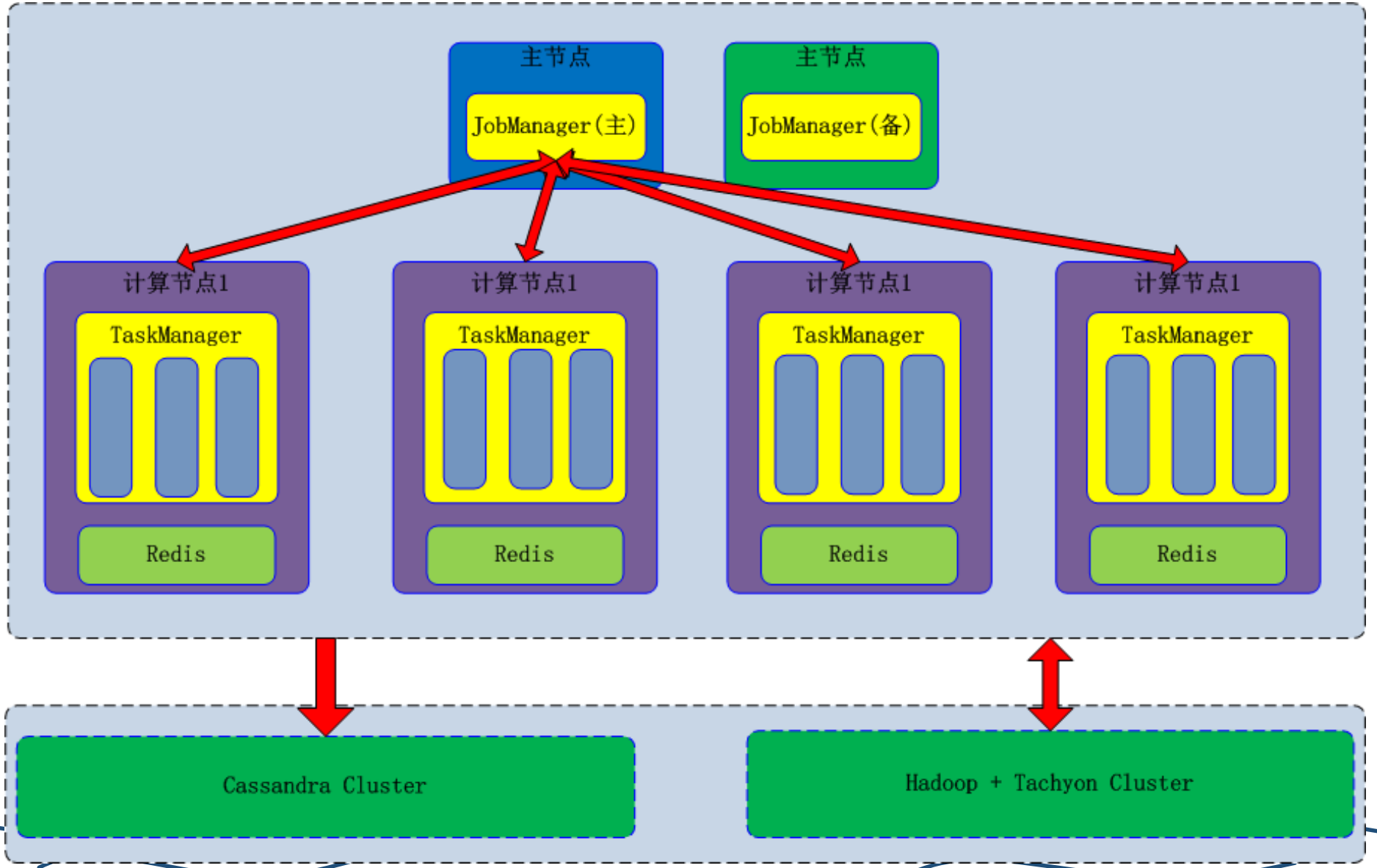
各类告警架构对比

方案	简述	实时流式计算(单条)	分布式	状态管理(中间数据等)	延迟	语言支持	hadoop整合	执行方式
现有方案(Akka+Cassandra)	状态依赖于外部存储(cassandra不一定抗的住)	支持	不支持	不支持，依赖外部存储	不好说	java	没有整合	
现有方案改造(Akka+Redis)	状态依赖于外部存储(Redis)，衍生出其他不可控的单点故障，而且开发工作量和难度大	支持	支持	不支持，依赖外部存储	不好说	java	没有整合	
Spark Streaming	Spark Streaming哪里都好，就是不支持真正意义上的流式计算（单条）	不支持(小批量)	支持	有状态(RDD)	秒级	java scala python	整合的比较好	Stage execution
Storm	Storm为流式计算而生，但是无法满足我们需要状态管理的场景，需要引入外部存储。 另外。	支持	支持	无状态	毫秒级	jvm系 Ruby python perl javascript	整合一般	
Flink	流计算方面综合了上面两者的优点，且基于pipeline模式要优于Spark的Stage模式	支持	支持	有状态，自己管理内存	毫秒级	java scala python	整合非常好，	Pipelined execution

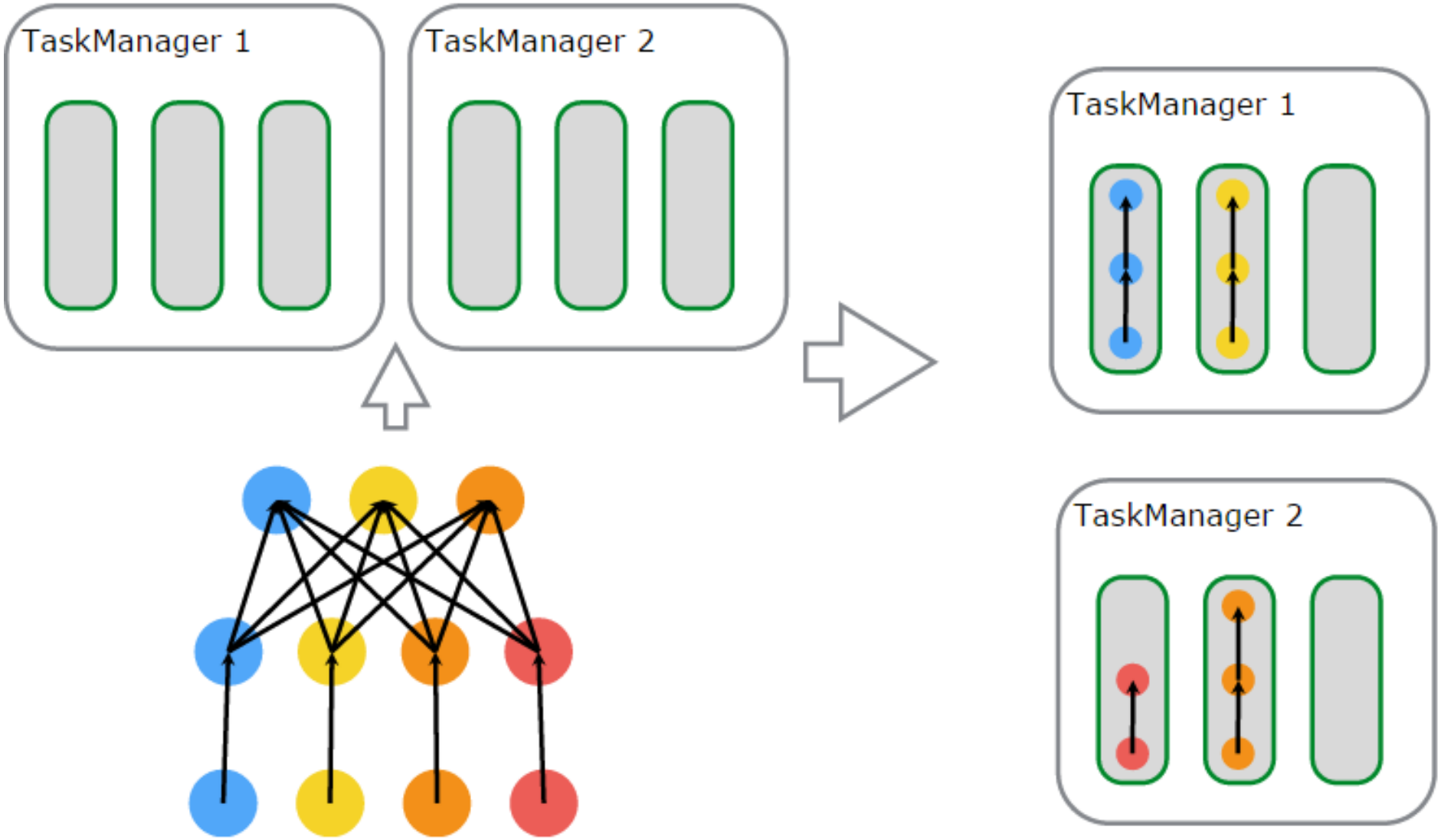
Flink逻辑架构


元数据管理
负载均衡
元数据管理层
元数据存储层
数据接入层
kafka
socket
RabbitMQ
...
告警计算层
异常检测job
No-events
检测job
...
window缓存
高速缓存层
长窗口数据
各种状态数据
存储层
在线存储
离线存储

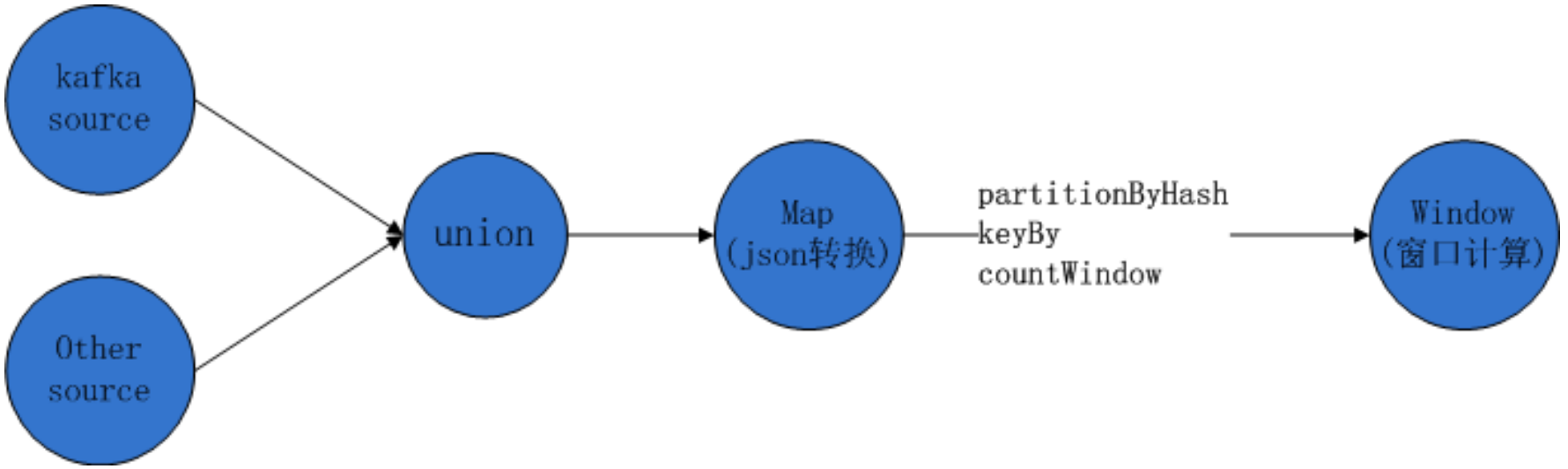
Flink物理架构


主节点
JobManager(主)
主节点
JobManager(备)
计算节点1
TaskManager
Redis
计算节点1
TaskManager
Redis
计算节点1
TaskManager
Redis
计算节点1
TaskManager
Redis
Cassandra Cluster
Hadoop + Tachyon Cluster

Flink任务调度



Flink 问题：多数据源问题



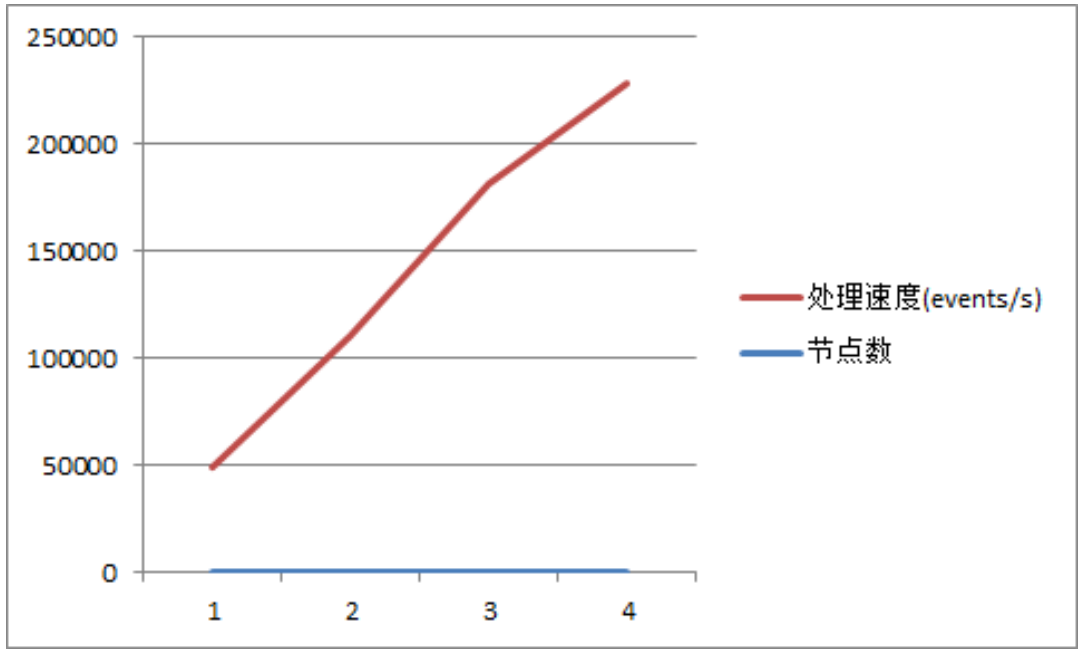
多数据源问题很好解决，简单改改代码即可，甚至可以实现可配置化，如图所示使用 union 可轻松实现，无需自己写额外的代码

Flink 问题：Rule更新问题 CRUD

- Json转换
  json→(key, event)
- 把event的到达时间存到Redis
  为no-events检测提供便利
- Event往redis存一份
  为跨events计算提供可能

- 根据event keys查询对应的Rule
  (key, event)—(key, event, rule)
- 算子内部缓存一份Rule
  避免每次都查数据库
- 收到预定义的Event就更新对应的rule
- 预定义的Rule只能发送到算子，所以当前算子只能有一个实例，会出现性能瓶颈(我仍然跑4节点跑出了57000 events/s的成绩)

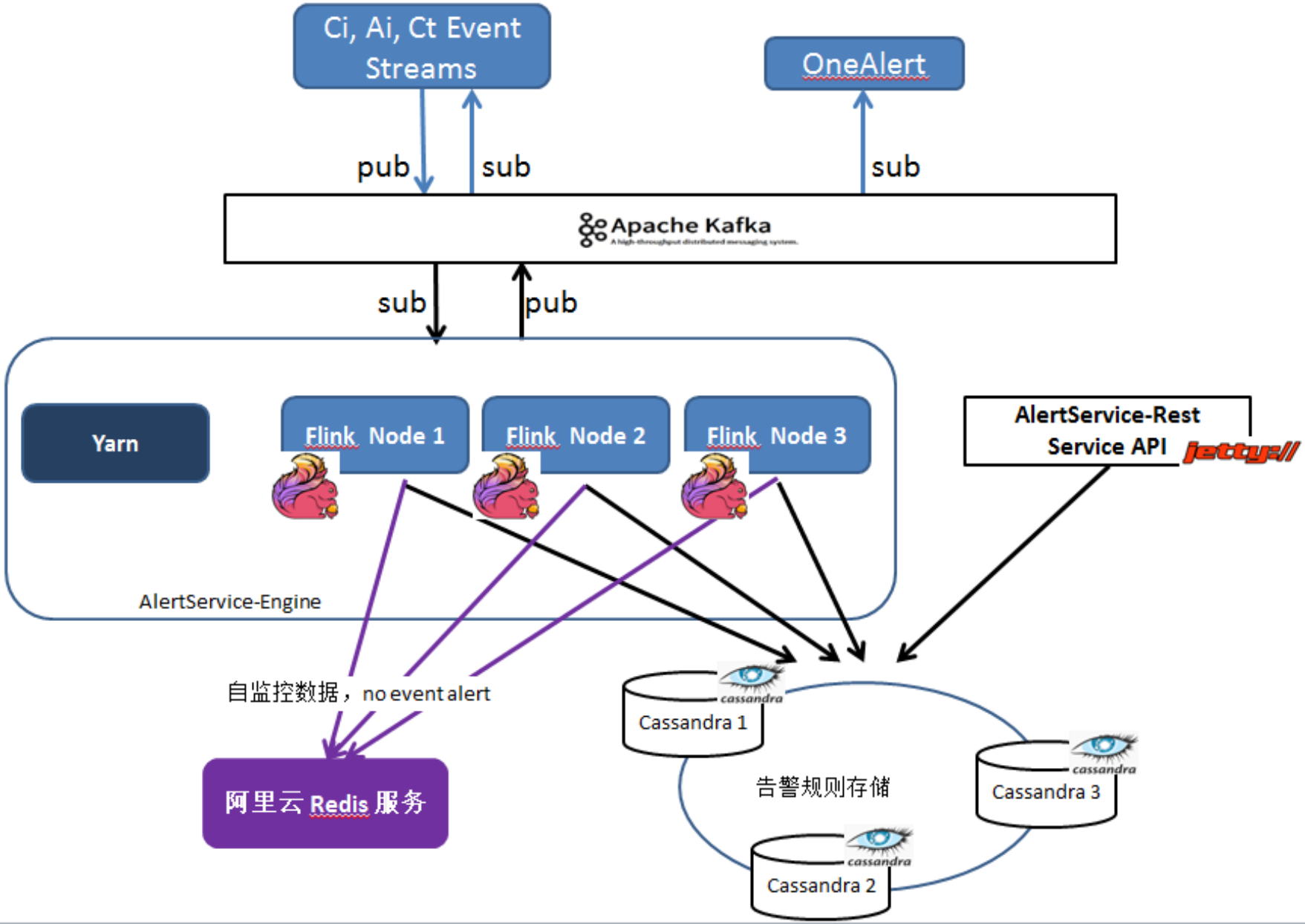
Flink 问题：无事件监测



节点数	处理速度(events/s)
1	48883
2	110813
3	181168
4	228202

2.1 CEP 1.0 Flink版

告警引擎Flink版 阿里云拓扑



问题与最终废弃原因

太吃硬件，同类框架同样的计算量可使用更少的机器

Yarn 部署的 Flink 集群，3台机器，单内存就有 180G。

交付问题，无法在客户环境部署

客户无法提供同样规则的机器，部署麻烦。

告警规则的需求无法得到满足

Flink 版本支持的规则过于草率，无法实现复杂粒度的逻辑运算。

告警规则抽象设计

SQL 是最好的告警规则抽象

SELECT avg(jvm_usage) AS jvm_usage_avg FROM some_agent WHERE avg(jvm_usage) > 200 FOR LAST 2 MINUTES

过去 2 分钟内，JVM 平均使用率大于200M则触发告警。

SELECT max(disk_usage) AS disk_usage_min FROM some_agent WHERE max(disk_usage) > 1000 FOR LAST 1

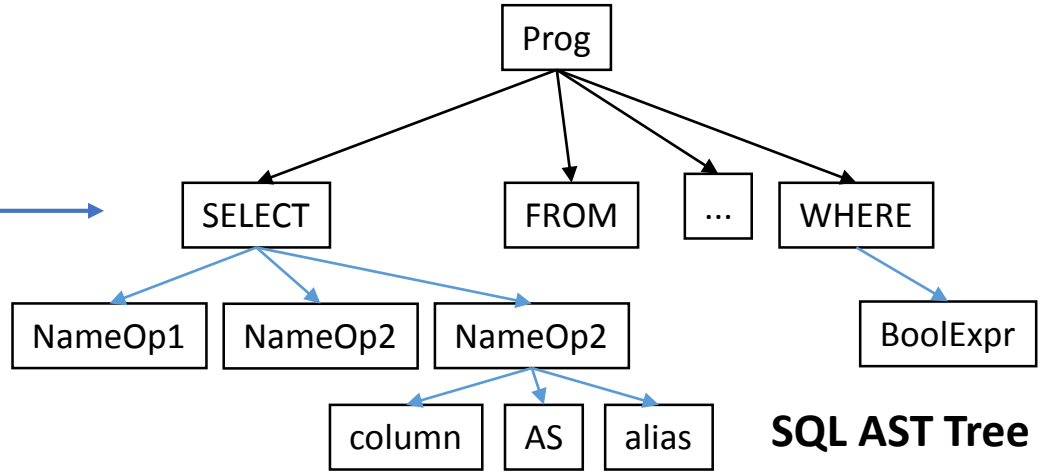
当磁盘使用量大于 1000G 的时候触发告警。

1. SELECT 部分相当于指标（计算、实时）结果集
2. FROM 相当于数据源
3. WHERE 相当于触发告警条件
4. FOR LAST 相当于计算队列

系统模块 - Engine

```
SELECT avg(jvm_usage) AS
jvm_usage_avg FROM some_agent
WHERE avg(jvm_usage) > 200 FOR
LAST 2 MINUTES
```

Raw SQL DSL



SQL AST Tree

```
private final List<String> streams;
private final IBooleanExpression whereClause;
private final List<Operand> columns;
private final List<NameOperand> filterKeys;

private final BoundingBox boundingBox;
```

Process Model（RuleTemplate）

Buffered Event Queue

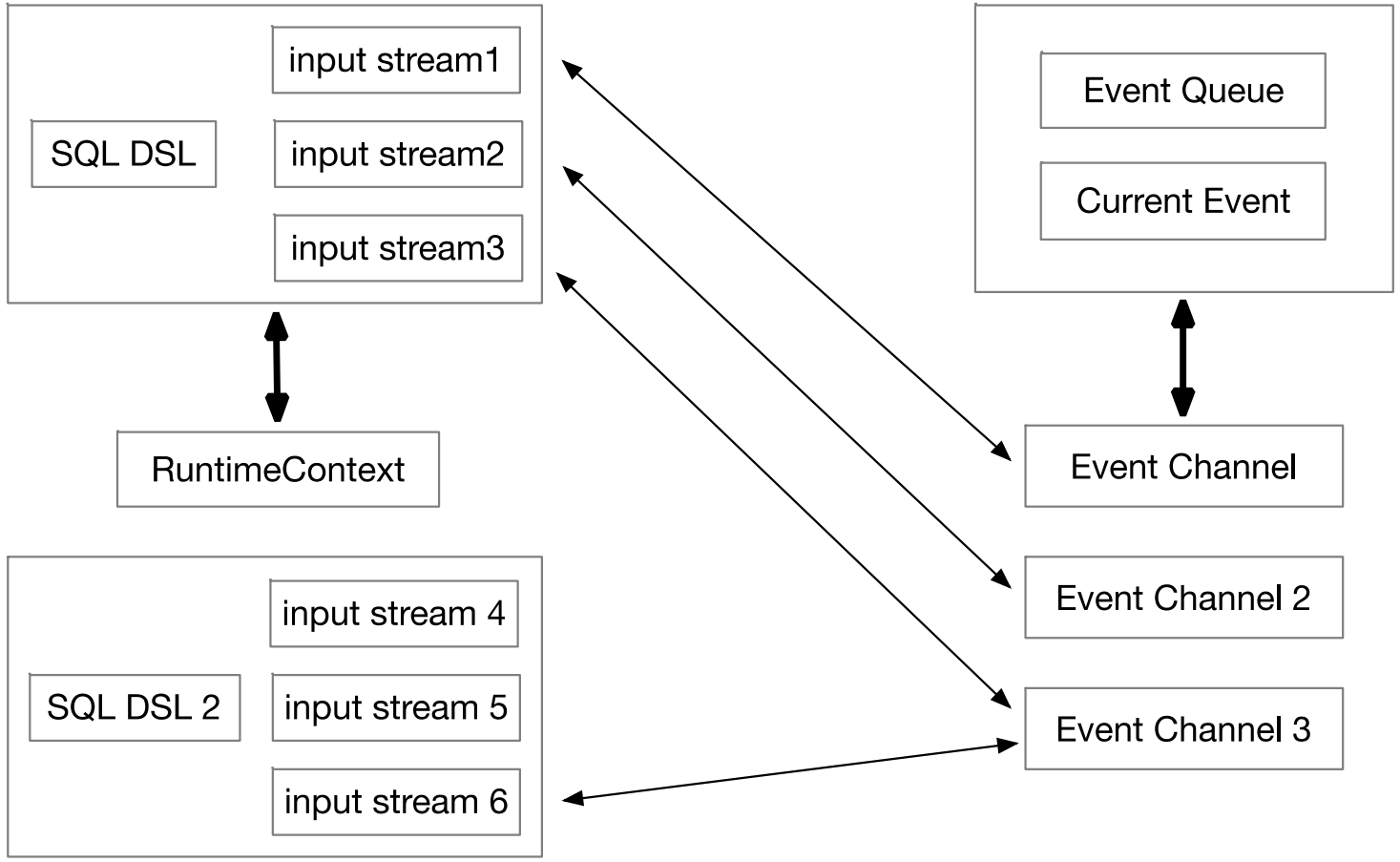
Current Event

Is Alarm (Boolean)

Metrics (Map)

Event Process Result

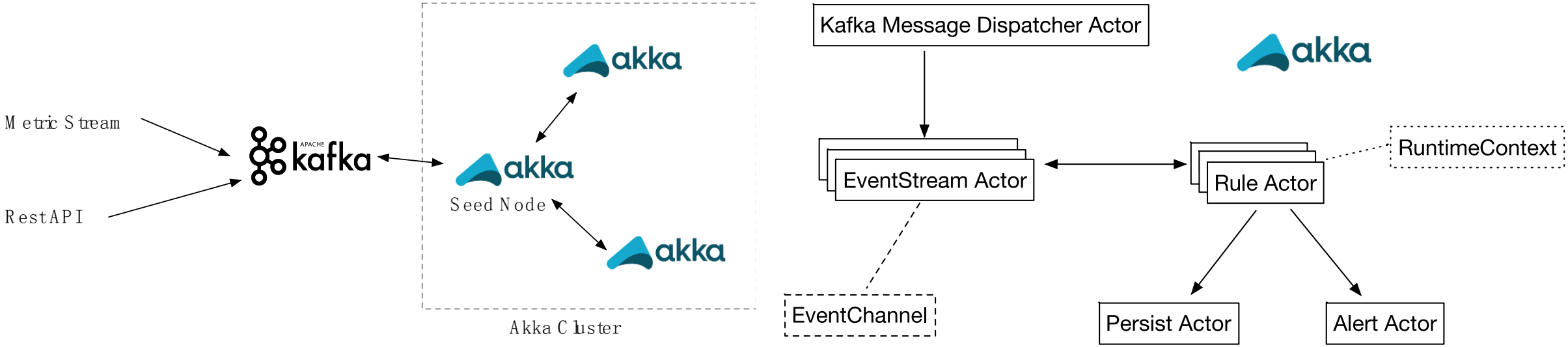
系统模块 - Runtime



RuntimeContext: Rule's CRUD

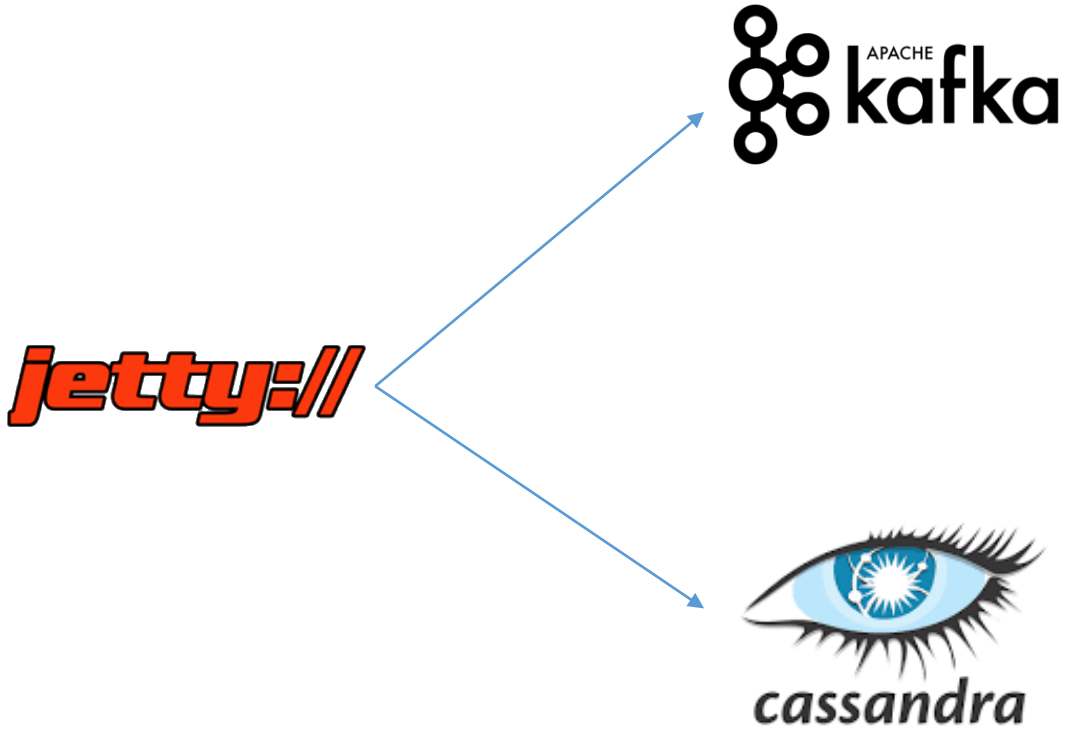
EventChannel: Event Buffer & Dispatcher

系统模块 - Akka



选出 Seed 节点和 Kafka 单线程消费分发，避免 Kafka repartition。

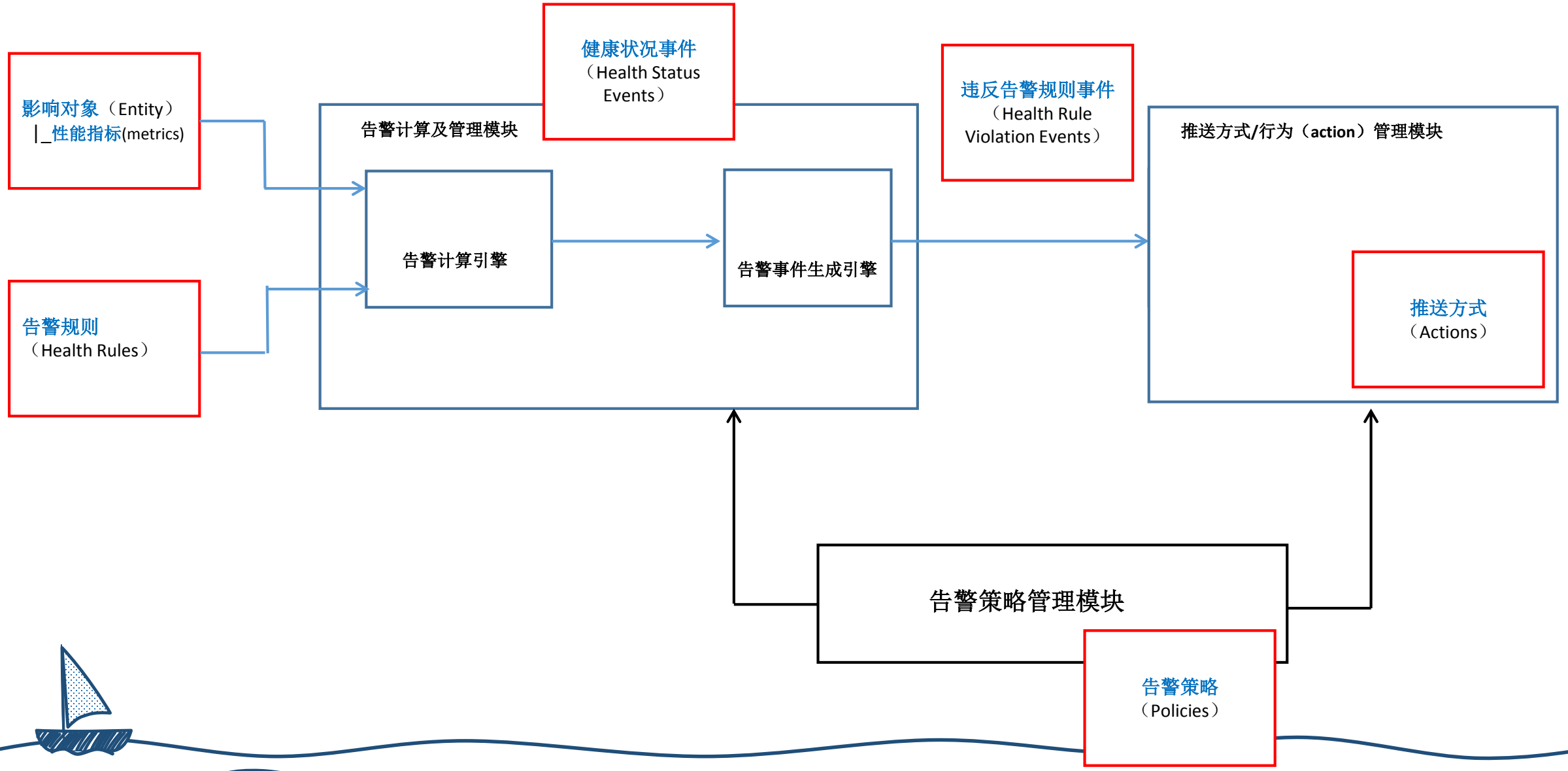
系统模块 - Jetty



纯 Restful API，以 Kafka 的形式通知 Akka 做变更。

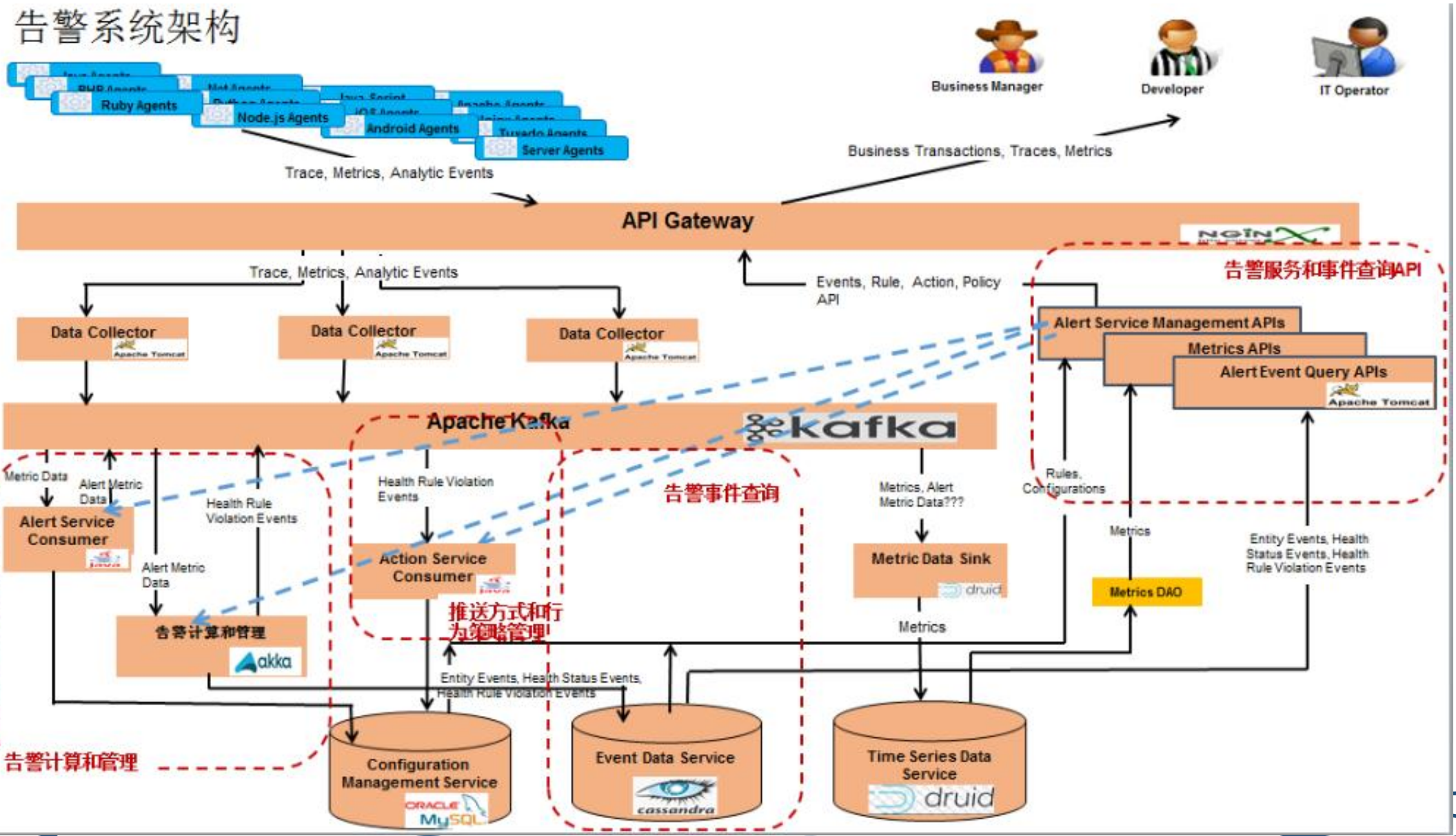
2.2 CEP 2.0 Akka版

逻辑模块与关联



2.2 CEP 2.0 Akka版

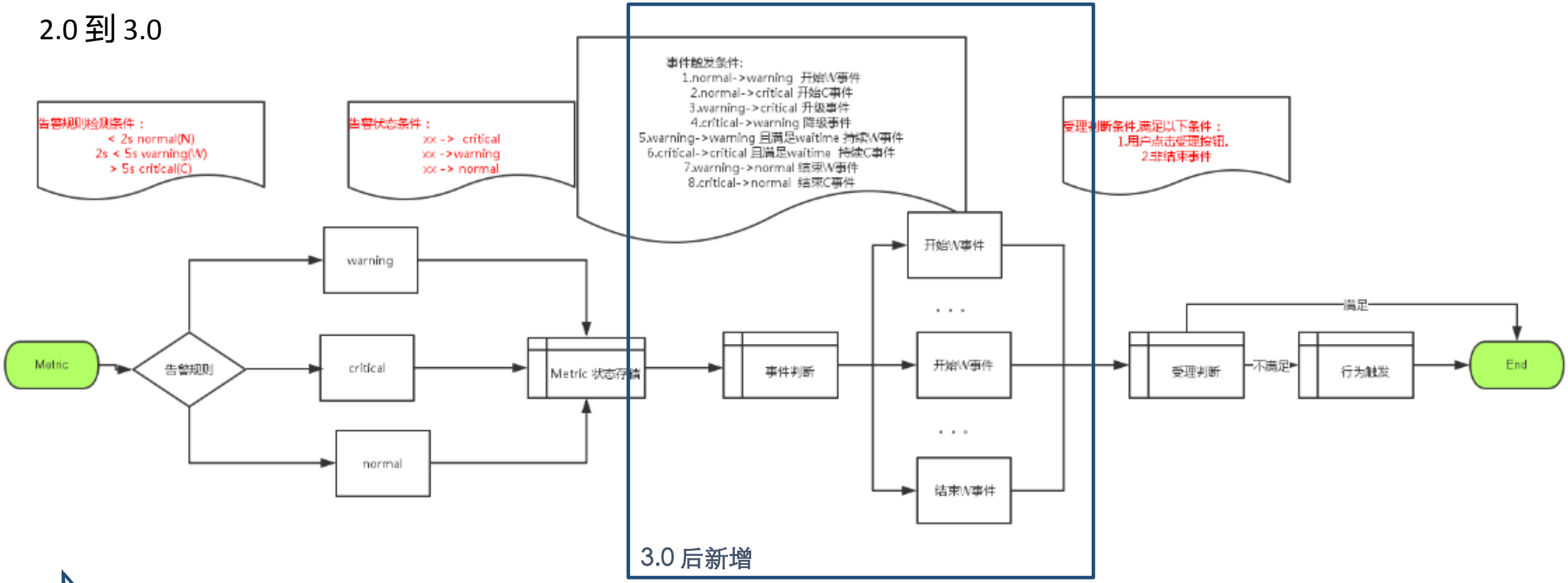
部署架构



2.3 CEP 3.0 动态告警

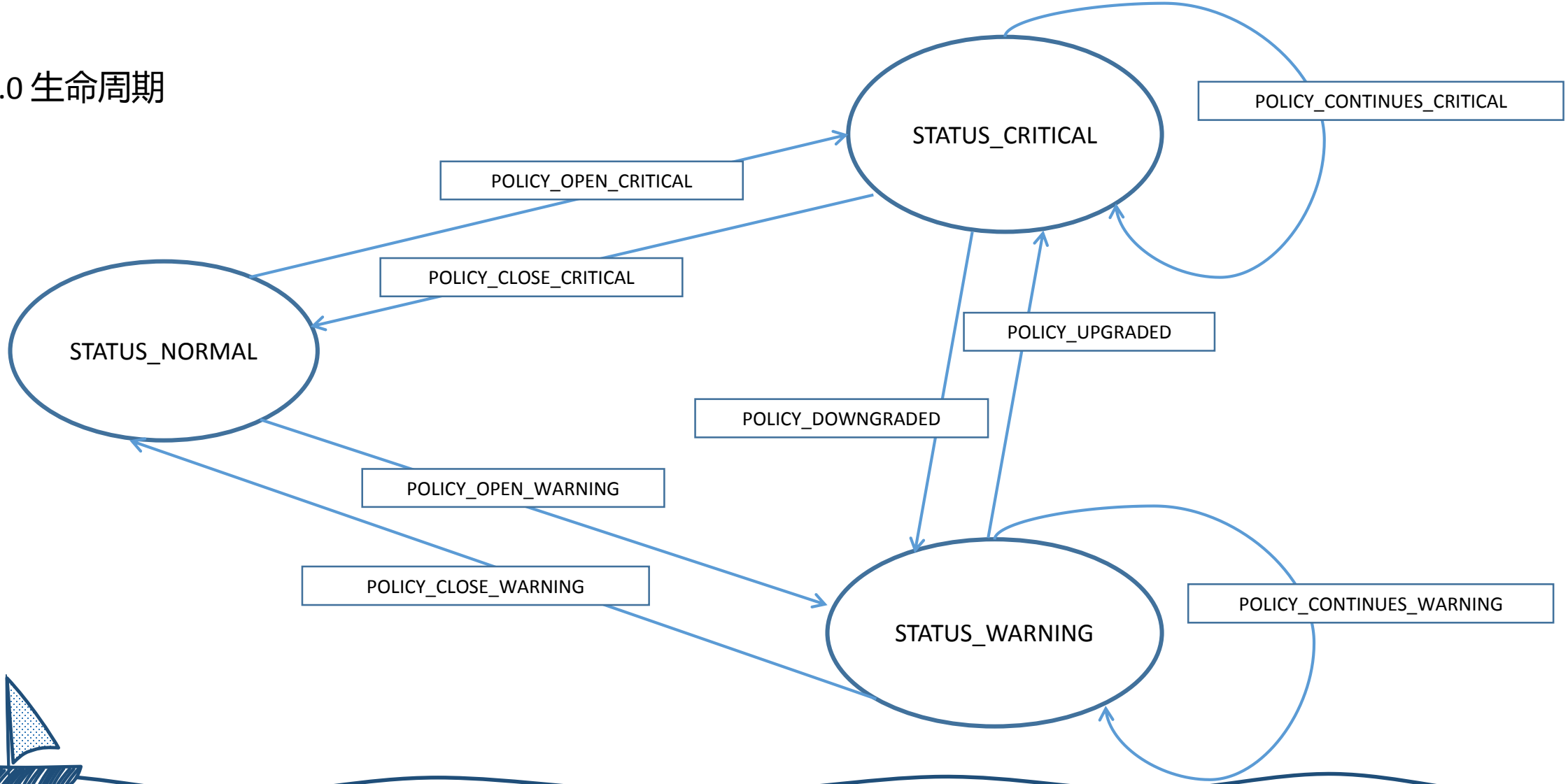
告警事件生命周期

2.0 到 3.0



告警事件生命周期

3.0 生命周期

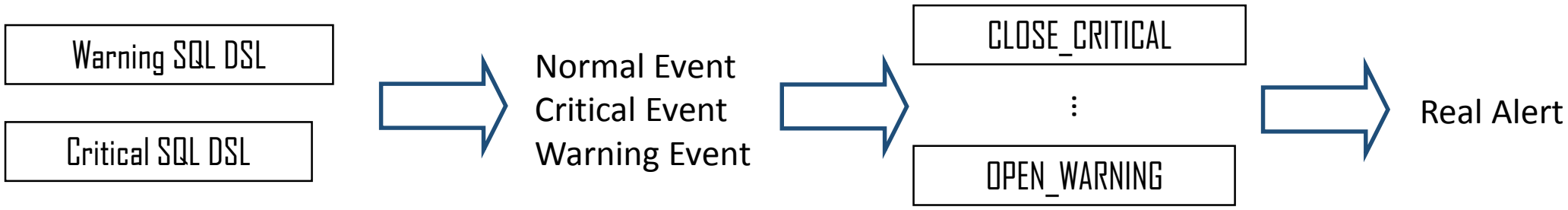


事件状态机设计

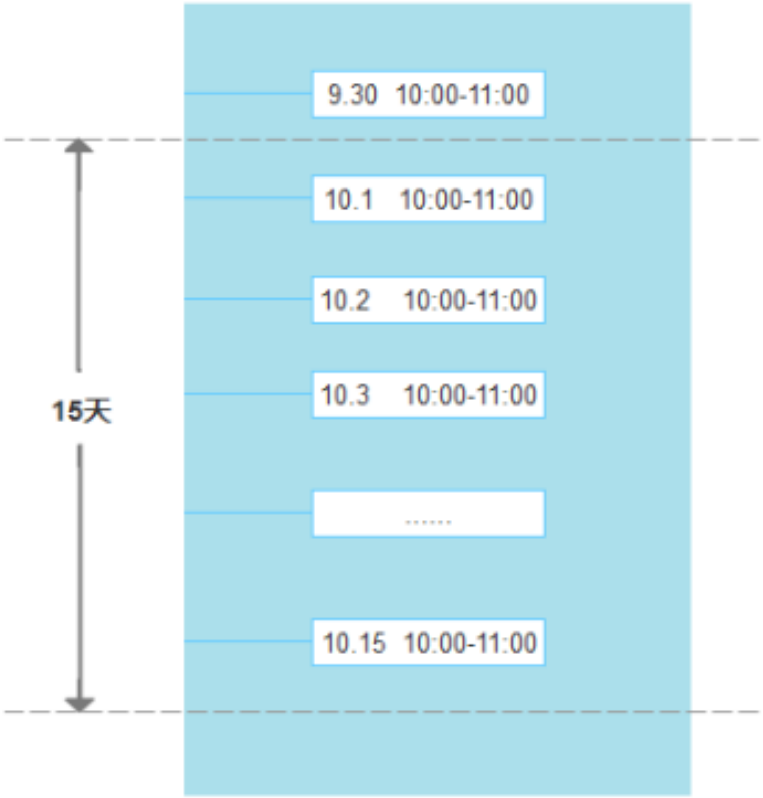
Use SQL calculate SQL

Open Warning

SELECT previous(severity) as previous_severity, severity as current_severity from eventStream
WHERE
(previous(severity) = '' OR previous(severity) = 'NORMAL')
AND
severity = WARNING' for last 3 min



同环比告警



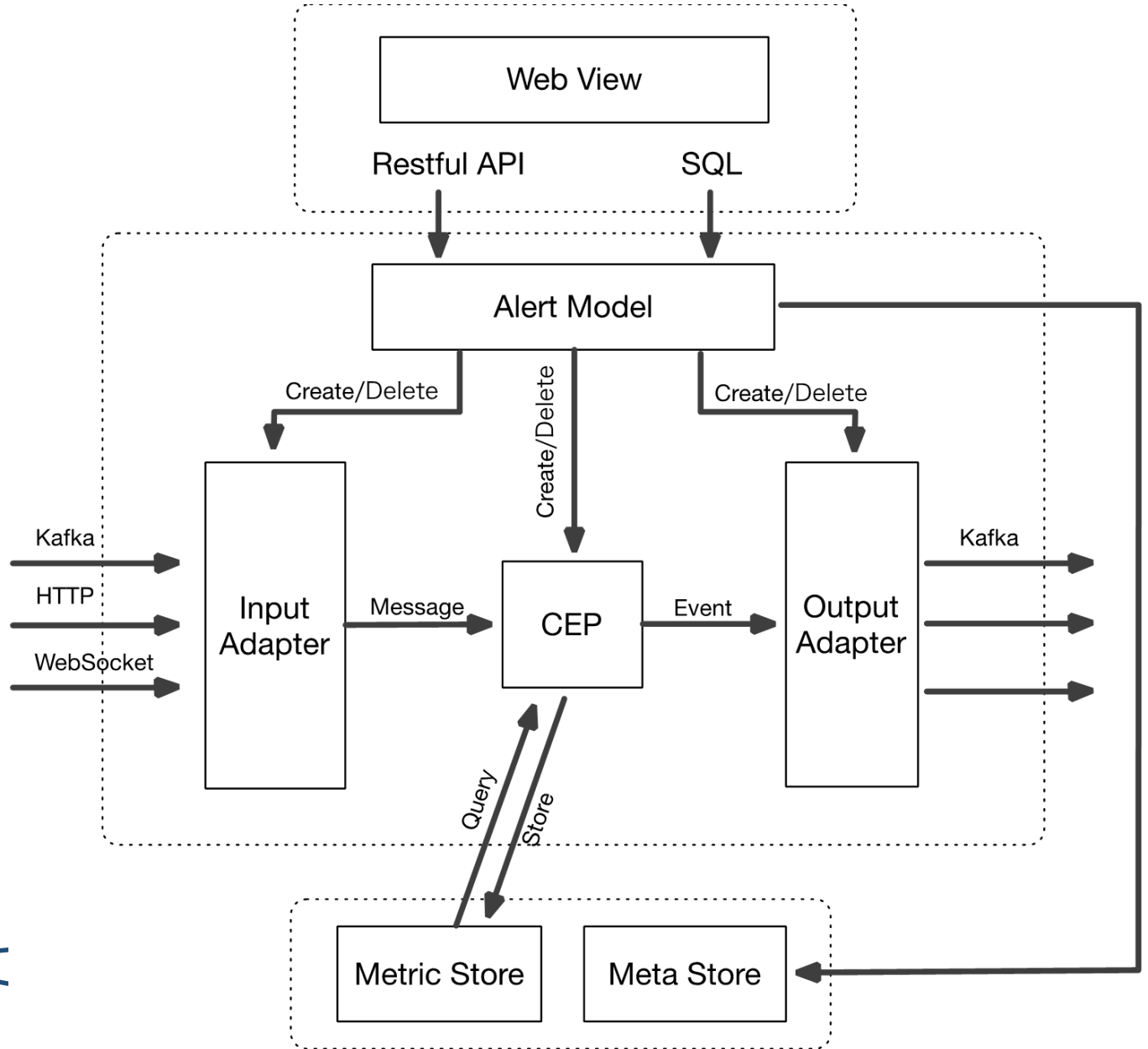

```
SELECT responseTime , avg(responseTime)
FROM tableA
WHERE (responseTime - avg(responseTime)) / avg(responseTime) > 0.05 *
period(now,  -1,  day, 5, min)
```

period(calTime, delayNum, delayUnit, durationNum, durationUnit)

calTime: 需要进行同比计算的时间节点描述；
delayNum: 同比周期时间；
delayUnit: 同比周期单位；
durationNum: 窗体时间；
durationUnit: 窗体单位

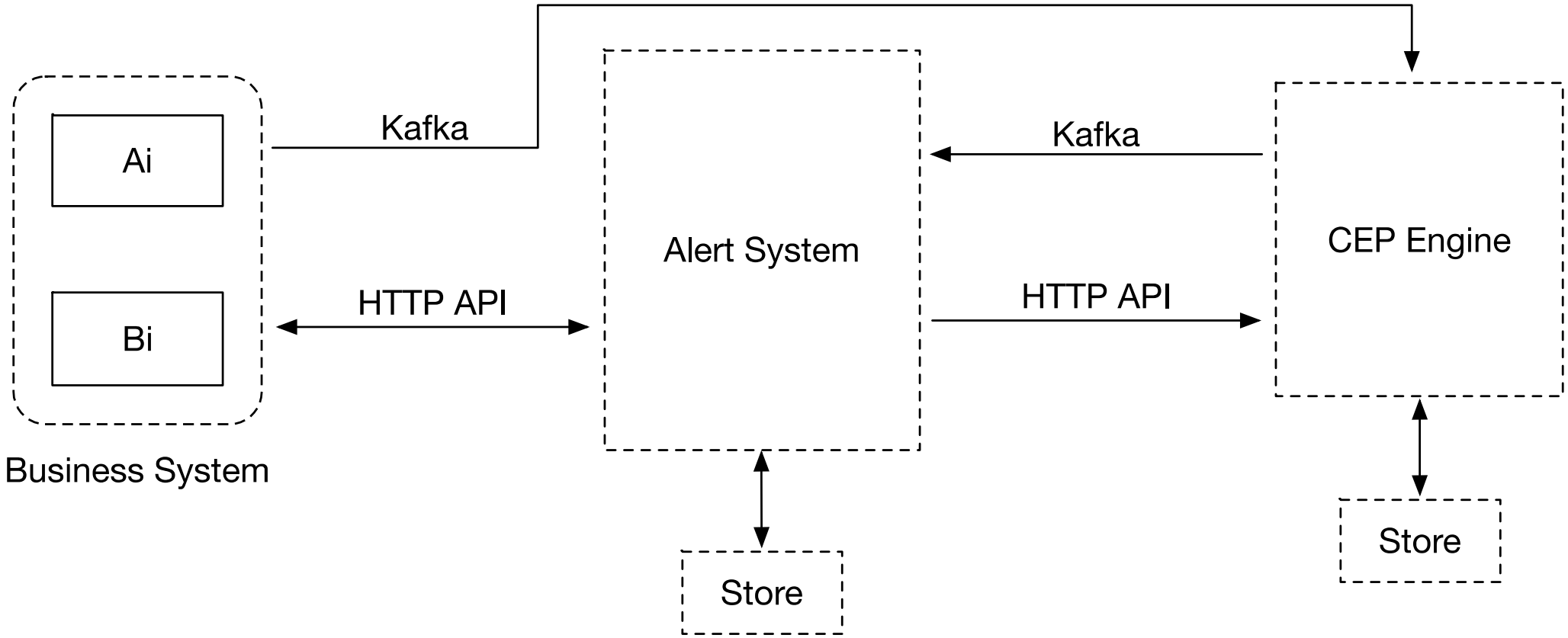
2.3 CEP 3.0 动态告警

存储变更

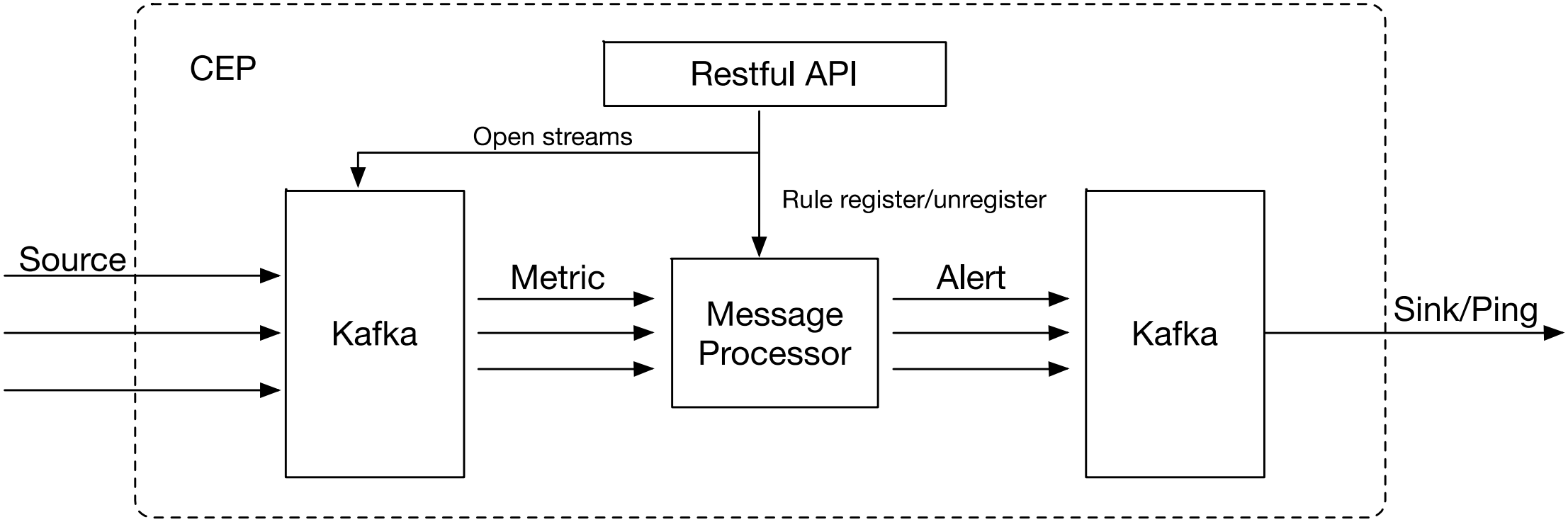


数据流图

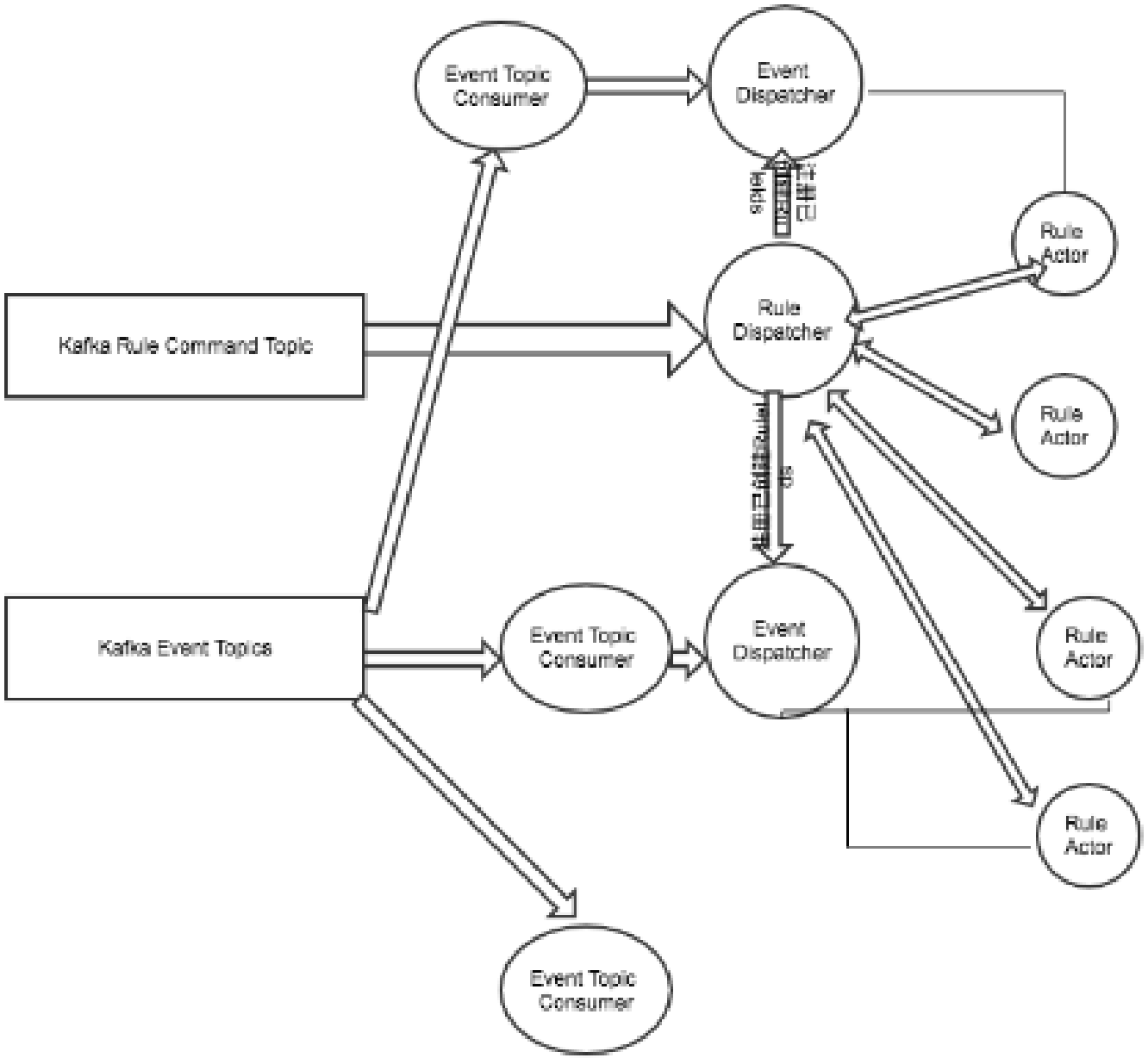
Data flow diagram for alert system



整体架构设计


CEP
Restful API
Open streams
Rule register/unregister
Source
Kafka
Metric
Message Processor
Alert
Kafka
Sink/Ping

2.4 CEP 4.0 流式计算 处理架构

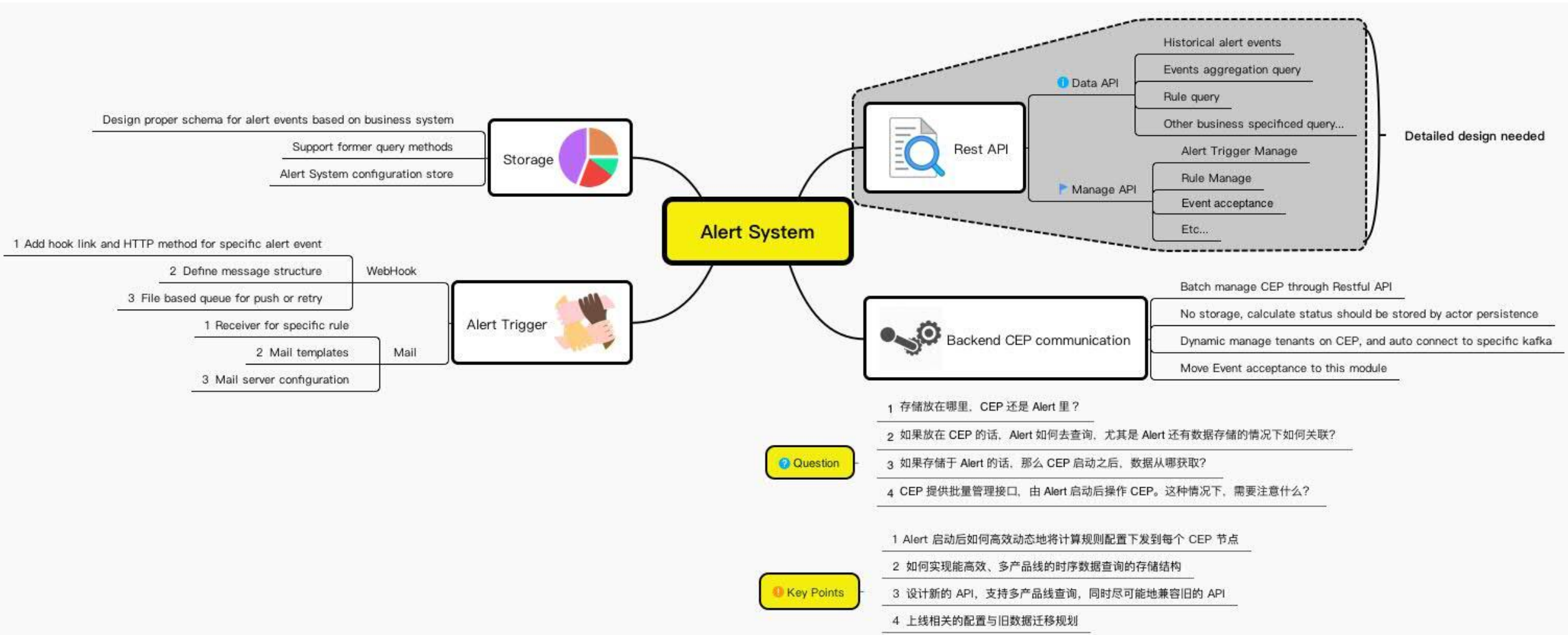


1. 增加CEP处理数据的伸缩性（scalability），水平伸缩以及垂直伸缩

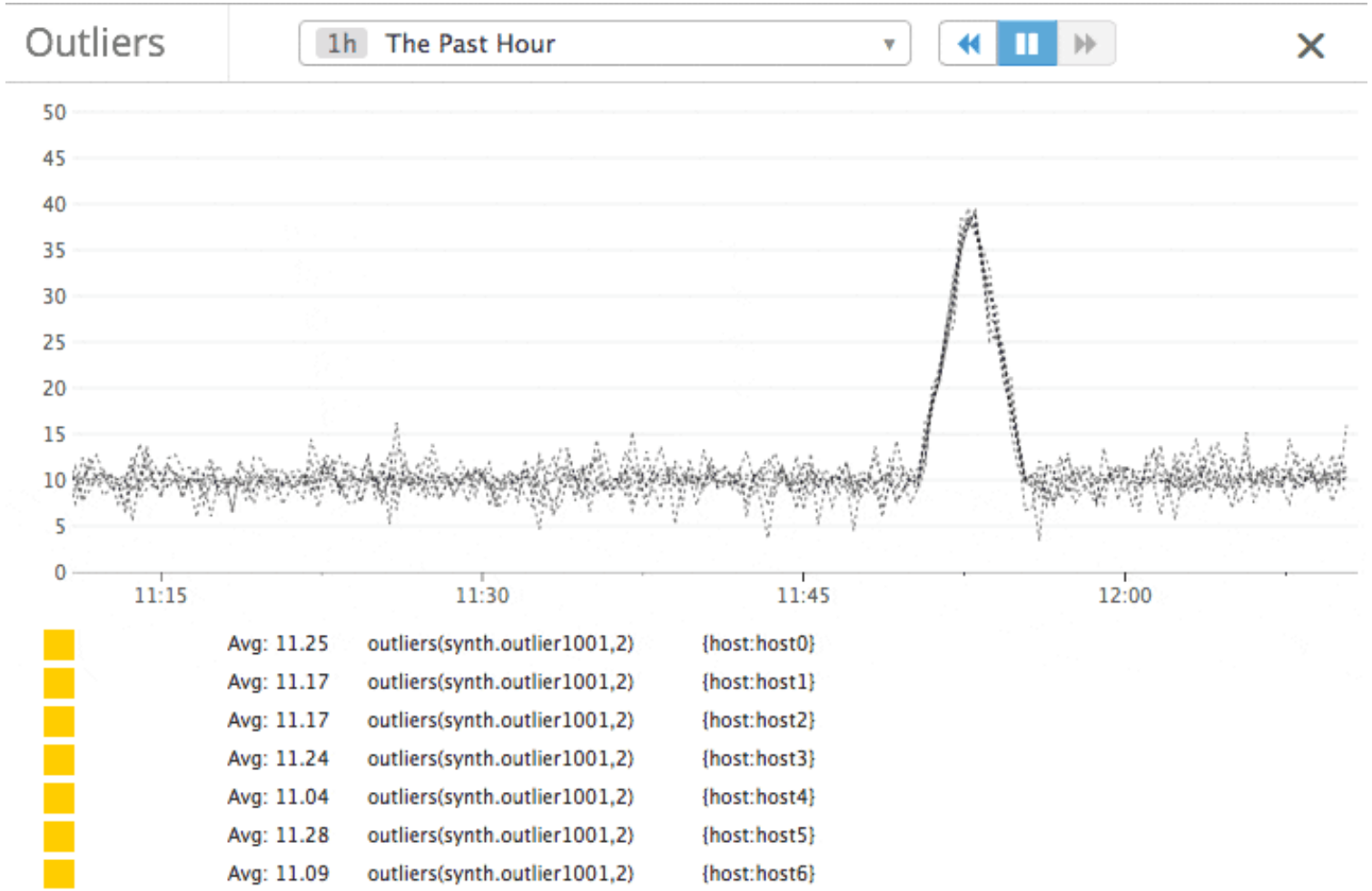
2. 提高CEP引擎的弹性（Resilience），也就是CEP处理引擎的容错能力

2.4 CEP 4.0 流式计算

系统模块



聚类告警



时序数据问题：

1. 数据维度高
2. 特征间的相关性大
3. 时序序列具有很大的噪声
4. 聚类算法的时间复杂度高

聚类算法：

1. 相似性度量
2. 数据降维
3. 聚类方法

3.2 CEP 开发总结

心得体会

DSL Driven Development

面向 DSL 的告警计算开发

Akka + Scala = Everything

函数式编程的强大表达能力，解决复杂处理逻辑。

Antlr4 解析与反解析（生成）

解析是处理 DSL，反解析是为了适配不同的旧模型。

提问环节

Q&A

THANK YOU