

# CAT性能优化的实践和思考

梁锦华

携程 高级技术专家



关注 QCon 公众号

# 收获国内外一线大厂实践 与技术大咖同行成长

✓ 演讲视频 ✓ 干货整理 ✓ 大咖采访 ✓ 行业趋势



# 自我介绍

- 曾就职阿里、百度、大众点评
- 10+ 年框架和中间件研发
- 携程
  - 消息中间件
  - 应用监控

# 目录

- CAT在携程
- CAT性能优化案例
- 总结和思考

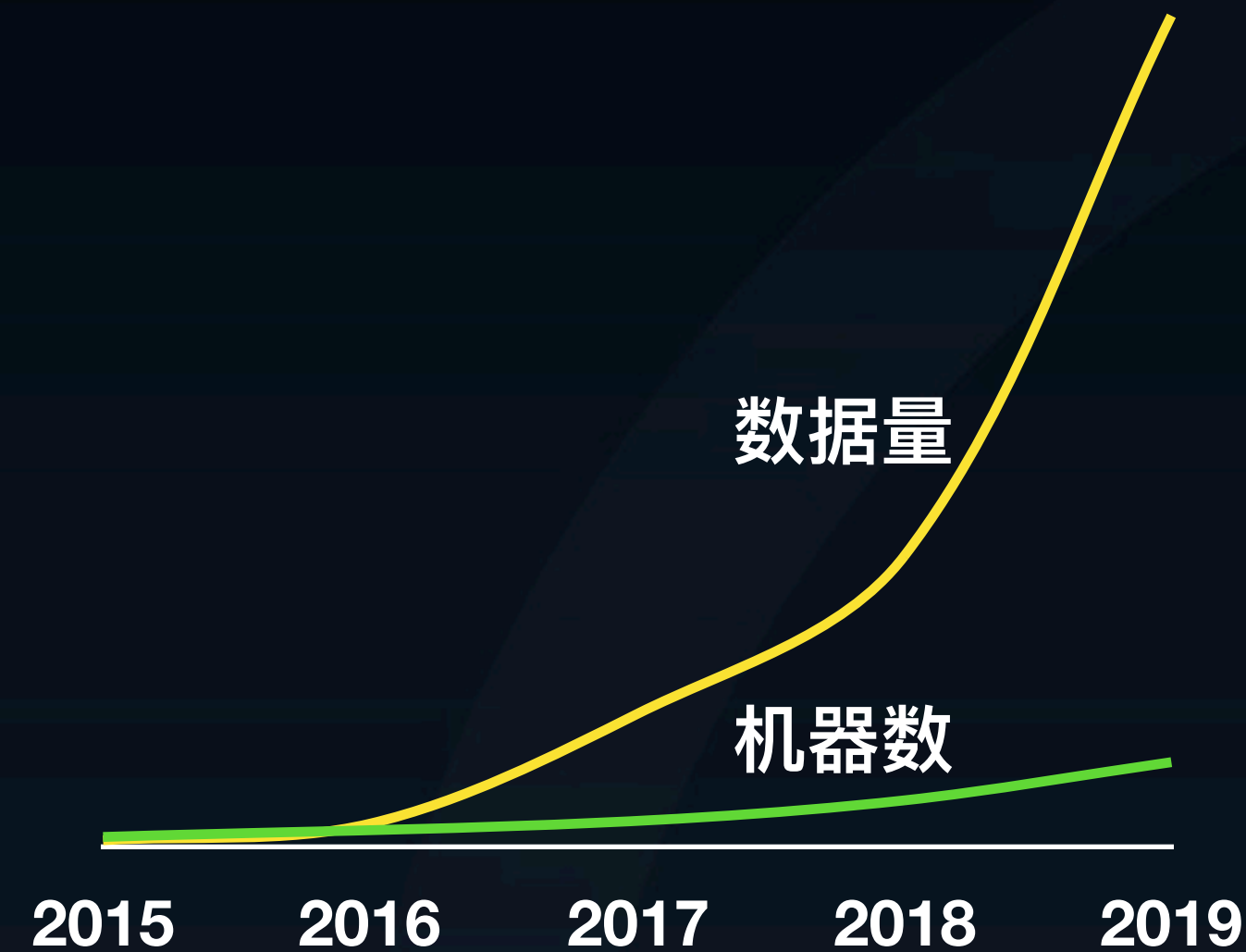
# 目录

- CAT在携程
- CAT性能优化案例
- 总结和思考



# CAT在携程

- 2014年底落地
- 监控数据指数级增长
- 2019年



Watch 1,086 Star 11,539 Fork 3,865

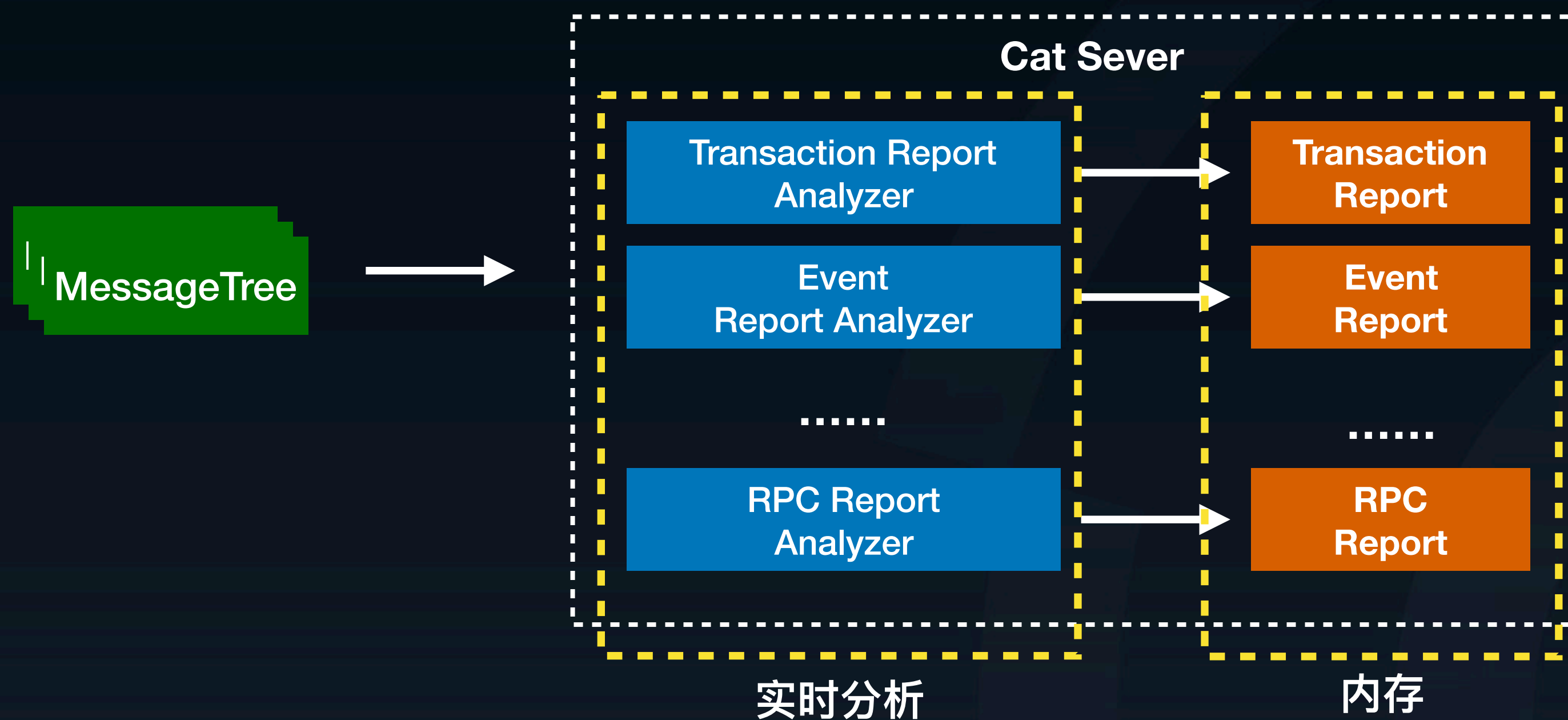
<https://github.com/dianping/cat>

- 7W+ 客户端
- 消息树：8,000亿消息/天，900TB/天，峰值流量 3,000万消息/秒
- 日志：40,000亿行/天，900TB/天，峰值流量 1.5亿行/秒

# 目录

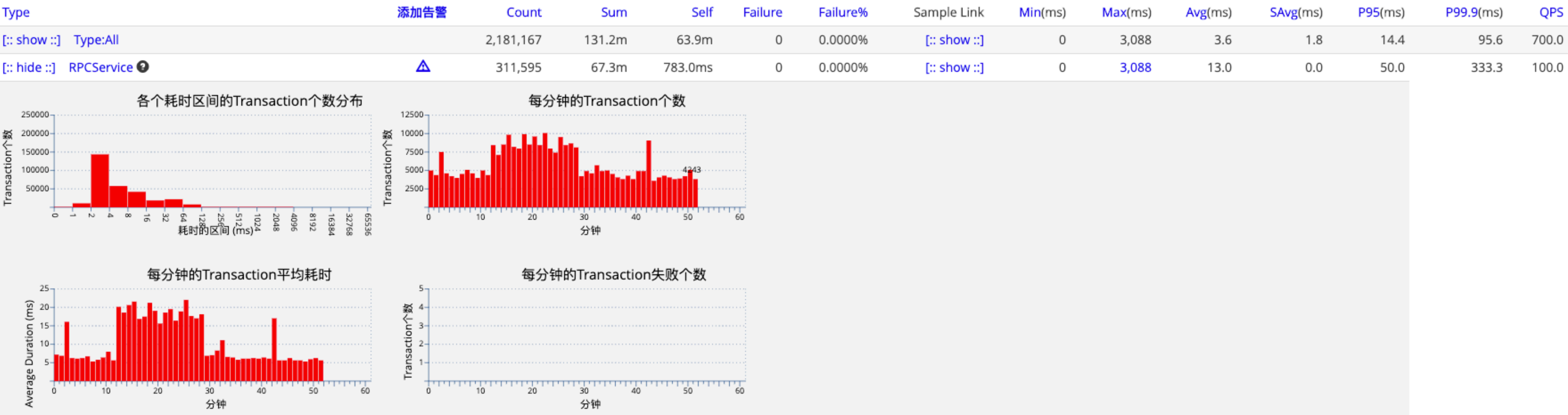
- CAT在携程
- **CAT性能优化案例**
- 总结和思考

# CAT的计算模型





# Report示例

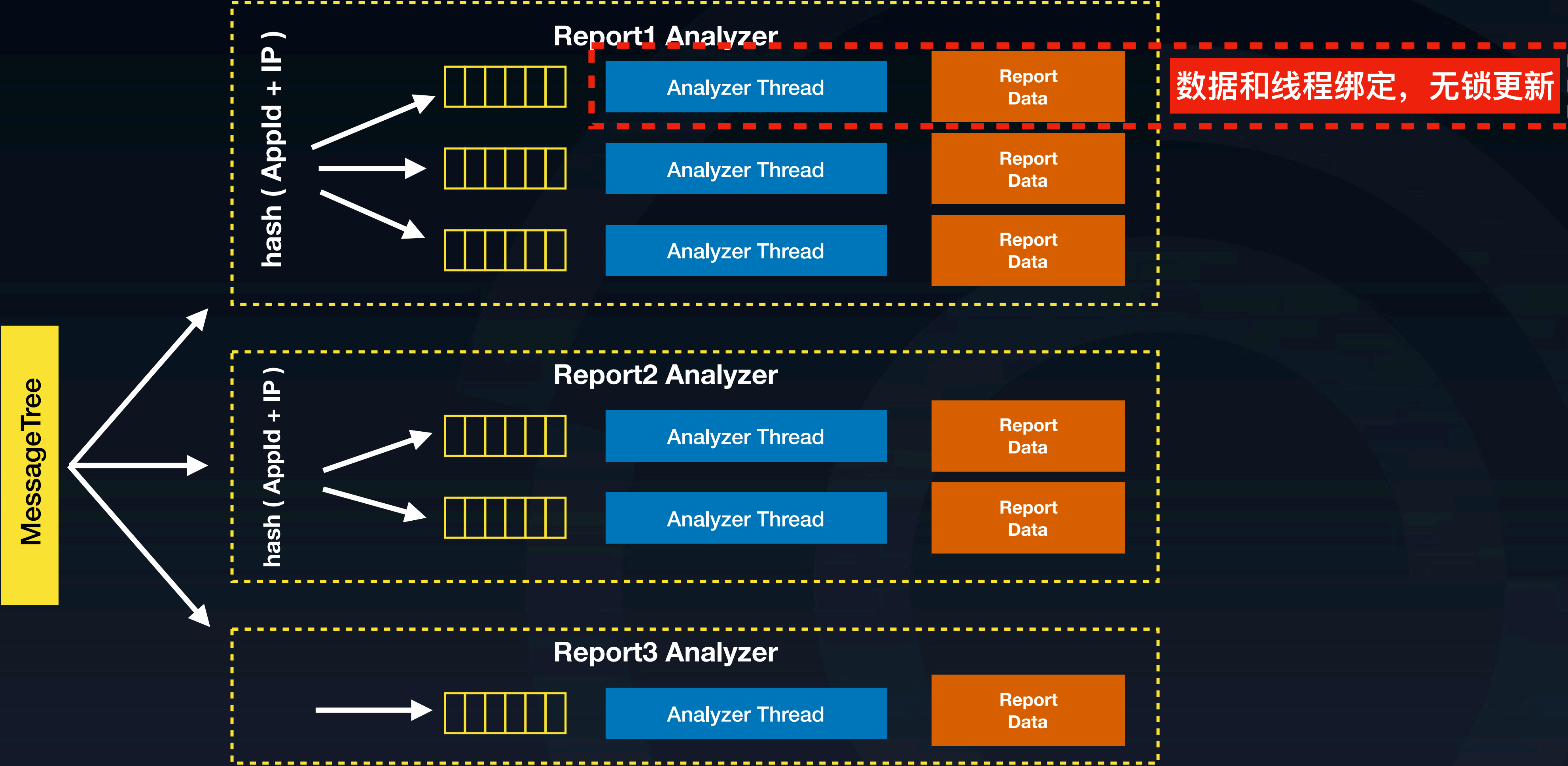


# CAT服务端常见问题

- CPU满
- GC频繁

# #1 线程模型优化

# CAT线程模型

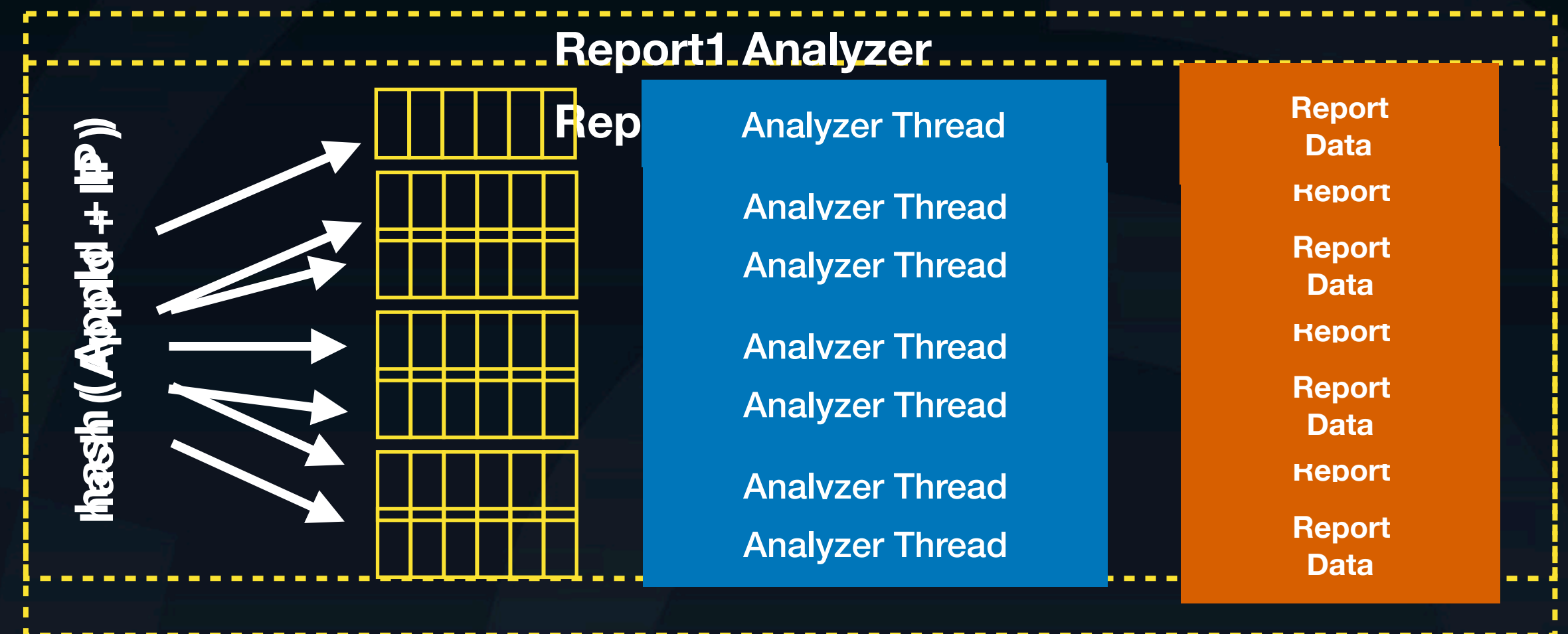


# 遇到的问题

- 应用流量不均导致部分队列堆积

- 打散数据：增加队列和线程

多次调整后，处理能力反而下降

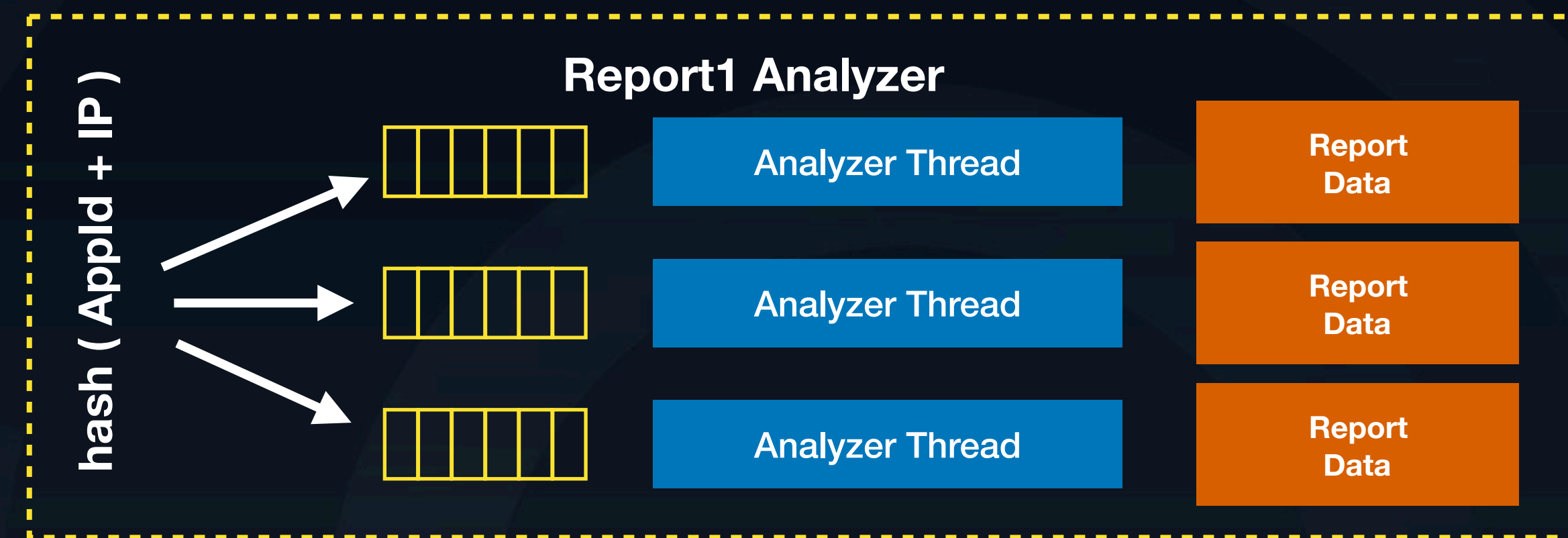




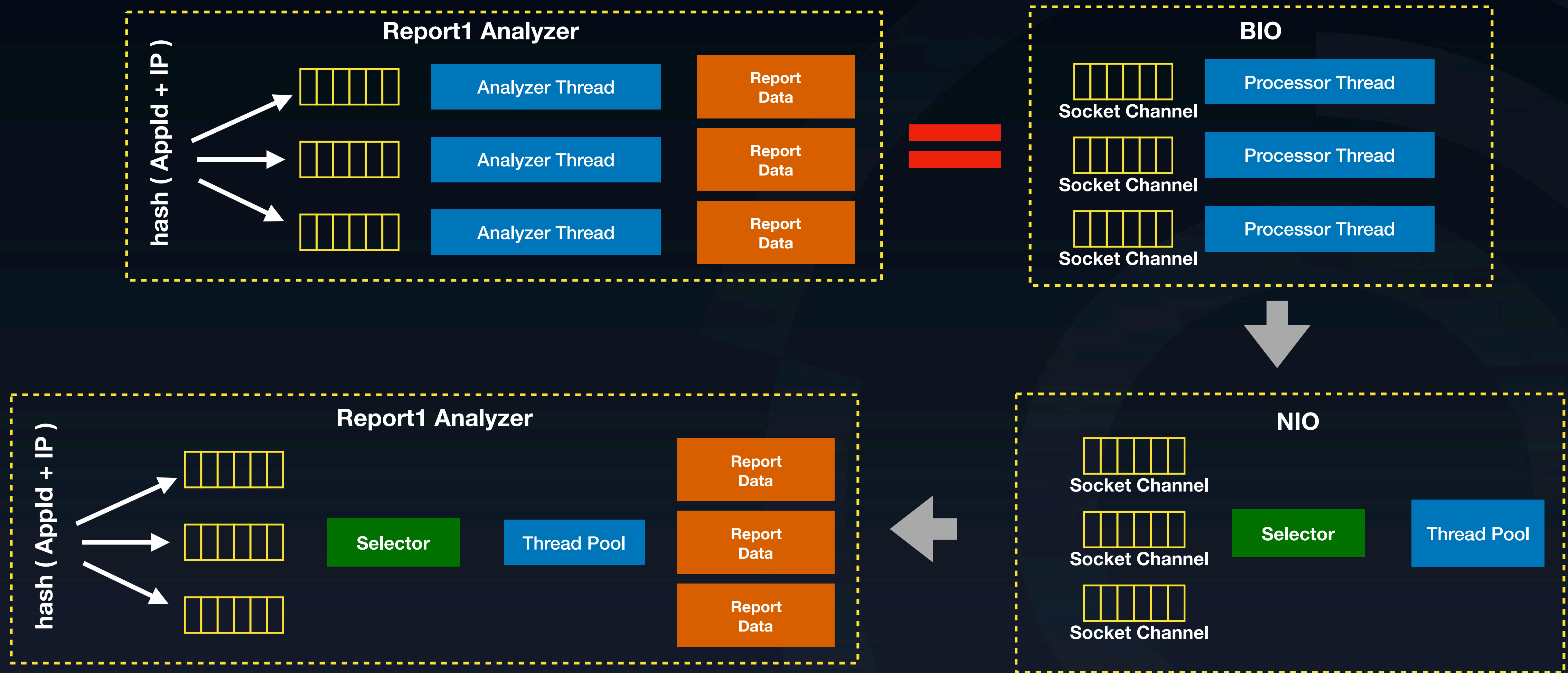
# 分析问题

- 处理能力下降
- 上下文切换频繁
- 线程过多

## 队列和线程解耦



# 分析问题



# 新线程模型

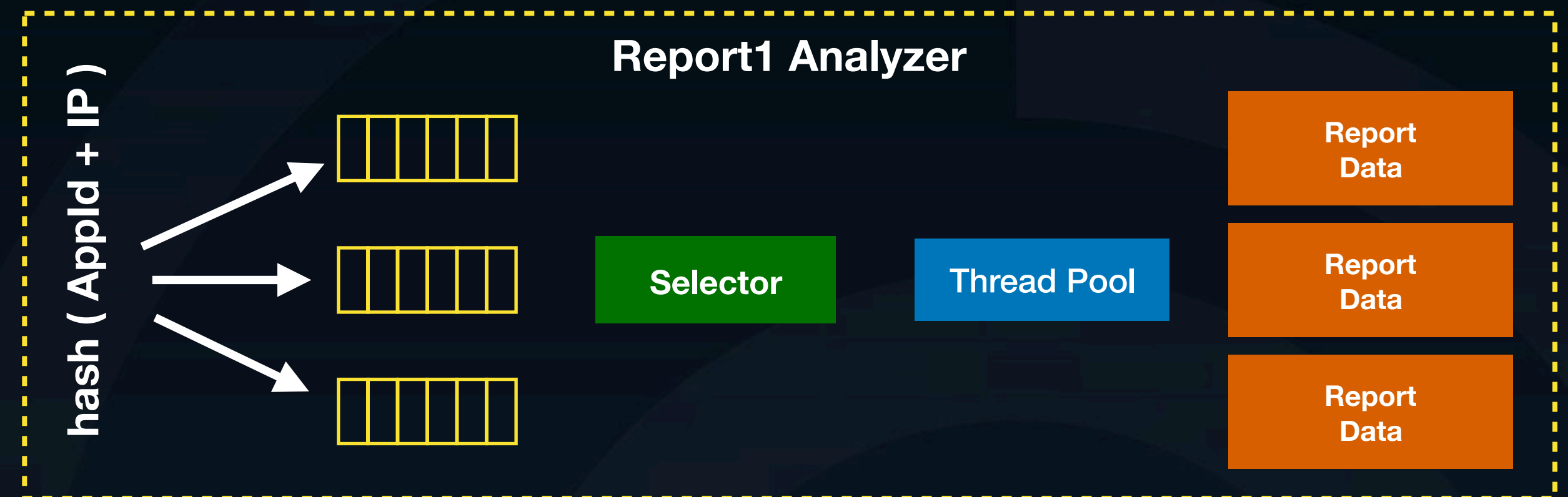
- Selector

- 监听队列数据

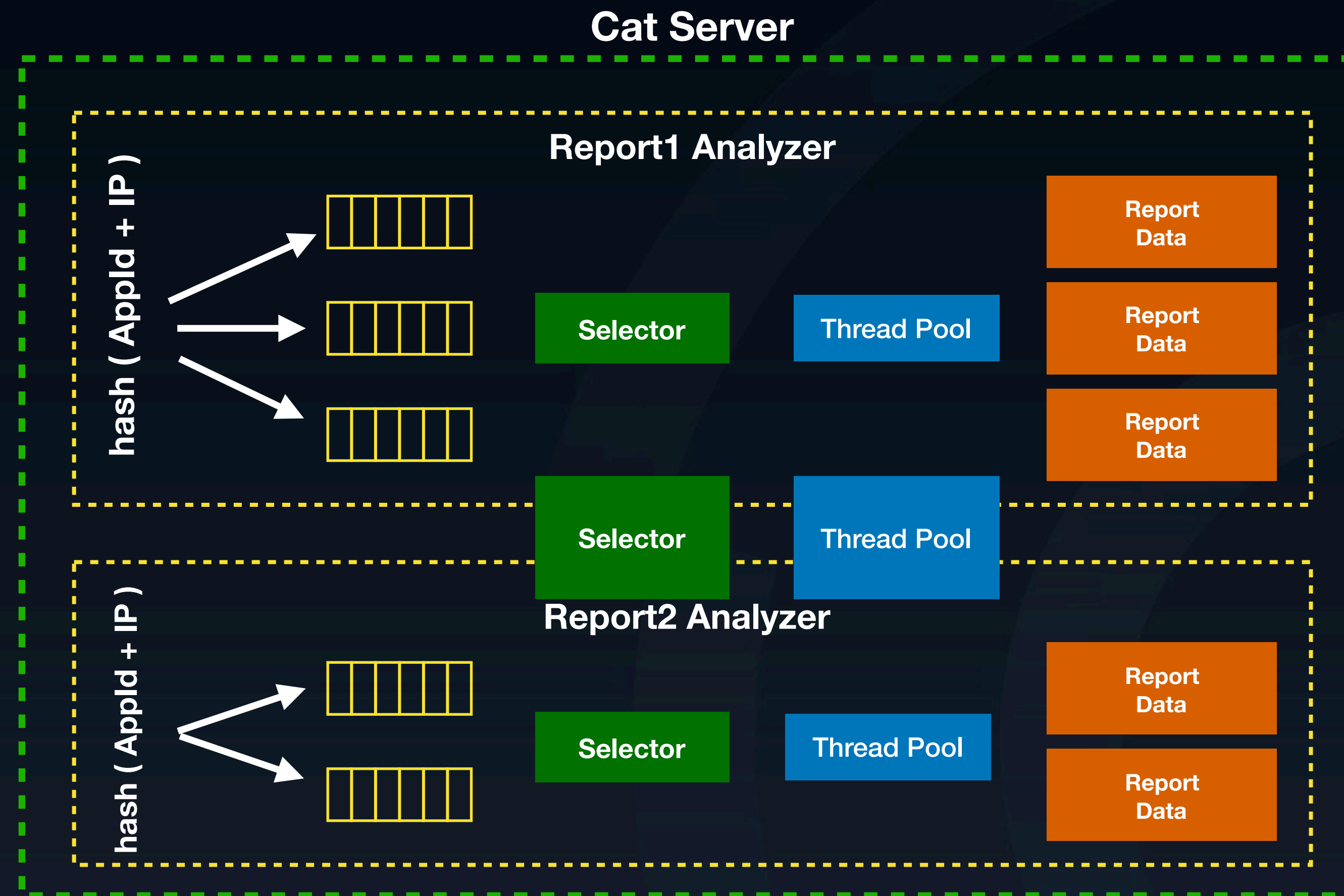
- 调度策略

- 充分利用ThreadPool

- 避免同一份队列数据的并发更新

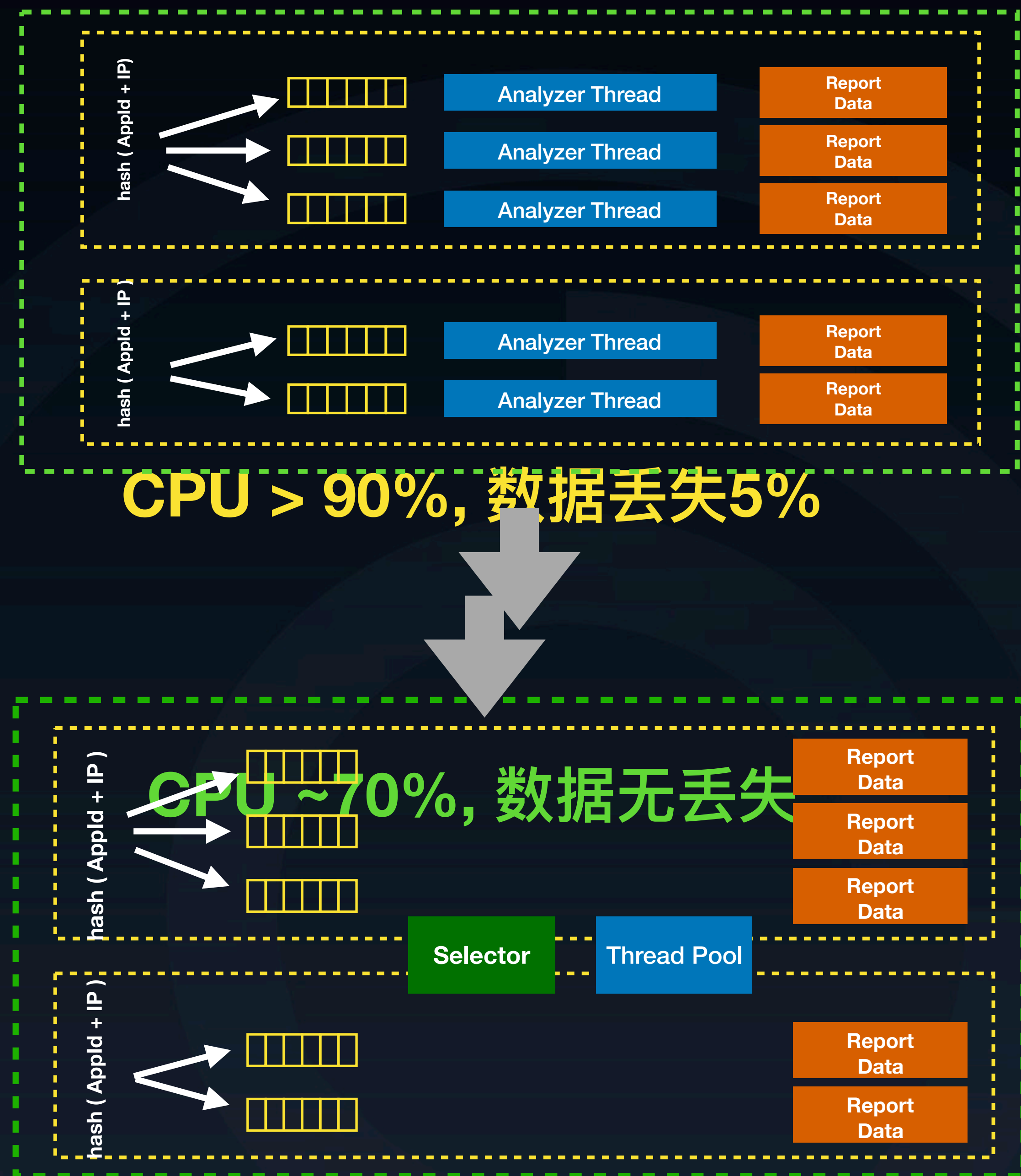


# 新线程模型



# 小结

- 队列和线程解耦
  - 多队列
    - 均衡数据
    - 减少单个队列的锁竞争
  - CPU核数个线程
    - 减少上下文切换
- 提供更灵活的调度策略





## #2 客户端计算

# 遇到的问题

**CPU再次用满，数据丢失**

# 分析问题

- 容量不够，加机器
- 优化，节省CPU
  - 部分服务端计算移到客户端





# 分析问题

- 哪些计算适合放在客户端
  - 不变的逻辑 Transaction和Event Report
- 服务端CPU使用比较多

# Transaction/Event Report的CPU使用

NAME	STATE	CPU Usage Per Core(%) ↓
AnalyzerThreadFor-transaction-1	 RUNNABLE	90.9
AnalyzerThreadFor-transaction-7	 RUNNABLE	85.3
AnalyzerThreadFor-transaction-2	 RUNNABLE	89.93
AnalyzerThreadFor-transaction-3	 RUNNABLE	89.3
AnalyzerThreadFor-transaction-6	 RUNNABLE	89.11
AnalyzerThreadFor-transaction-5	 RUNNABLE	87.05
AnalyzerThreadFor-transaction-4	 RUNNABLE	88.37

5.3个核

NAME	STATE	CPU Usage Per Core(%)
AnalyzerThreadFor-event-4	 RUNNABLE	50.08
AnalyzerThreadFor-event-1	 RUNNABLE	47.96
AnalyzerThreadFor-event-3	 RUNNABLE	57.96
AnalyzerThreadFor-event-2	 RUNNABLE	67.88

2.2个核

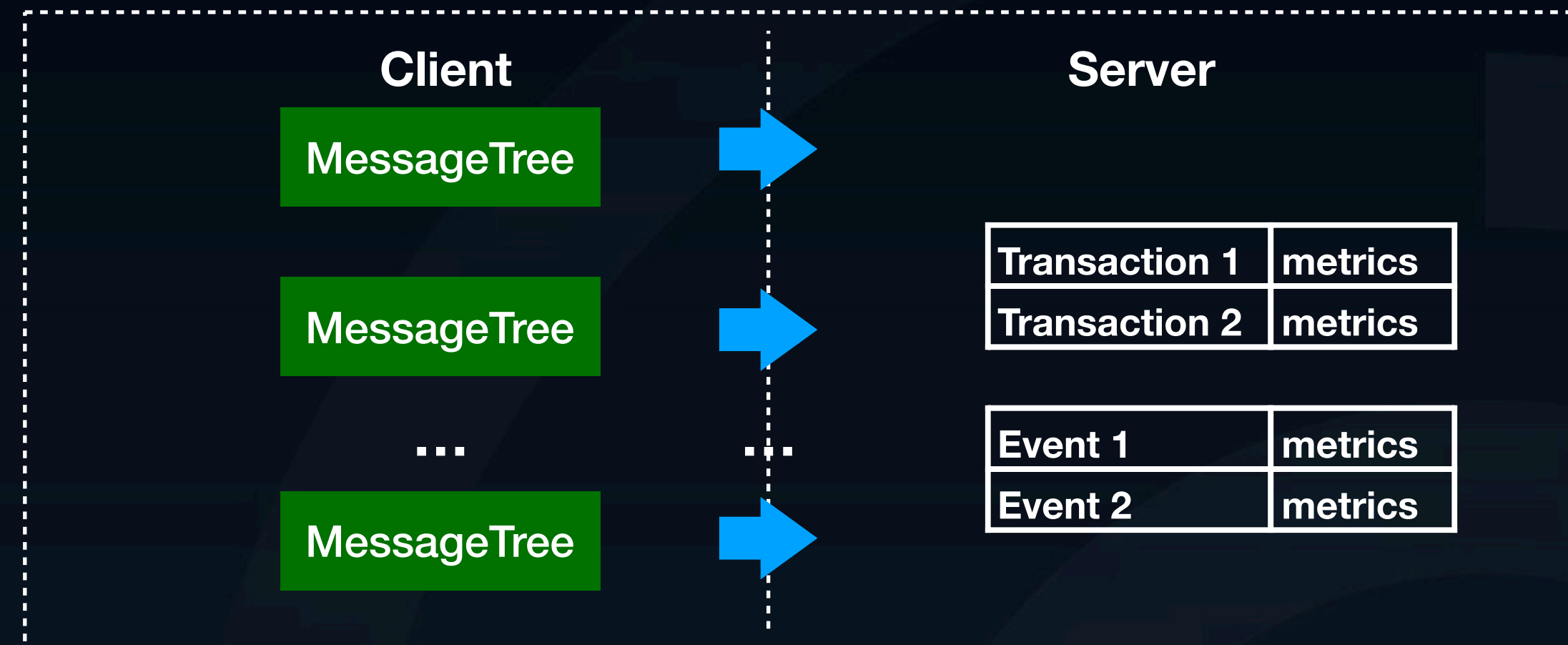
两个Report共占用7.5个核, 23% CPU资源!



# Transaction/Event Report计算

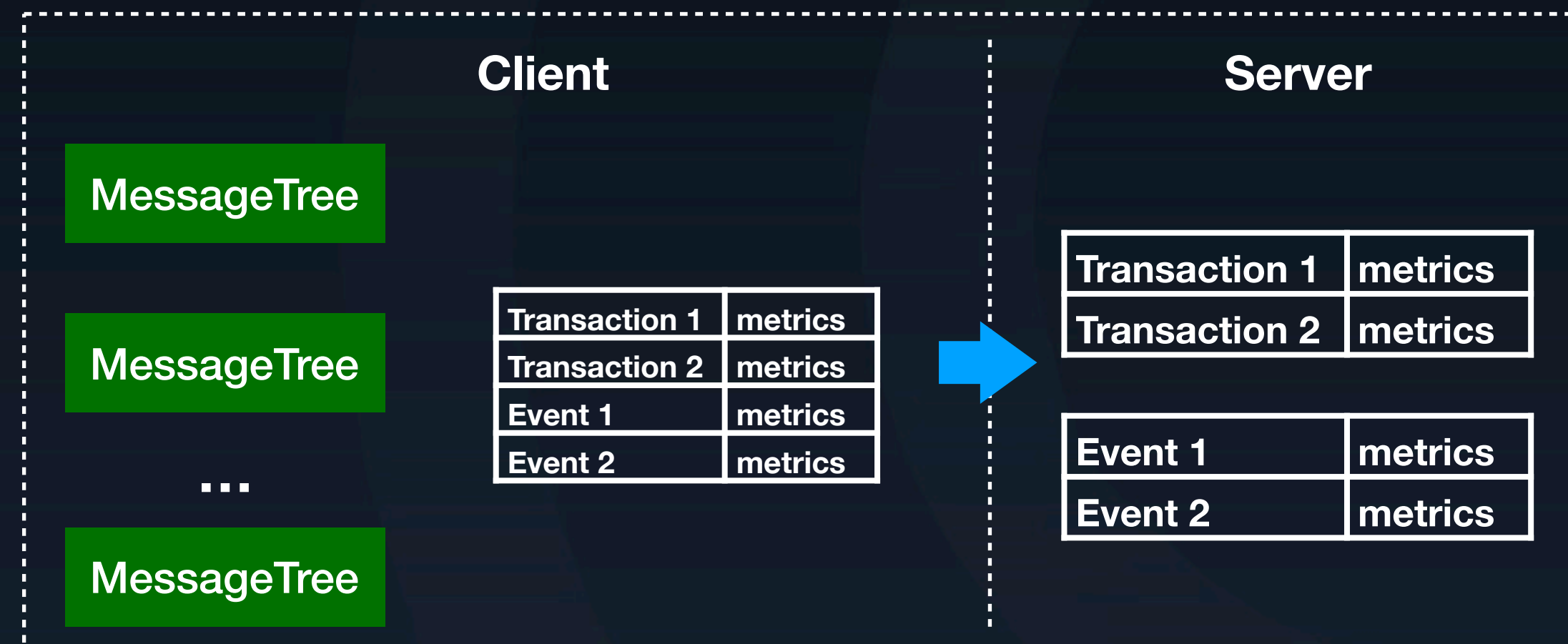
## 服务端计算

1. 遍历每棵MessageTree
2. 更新Transaction/Event metric

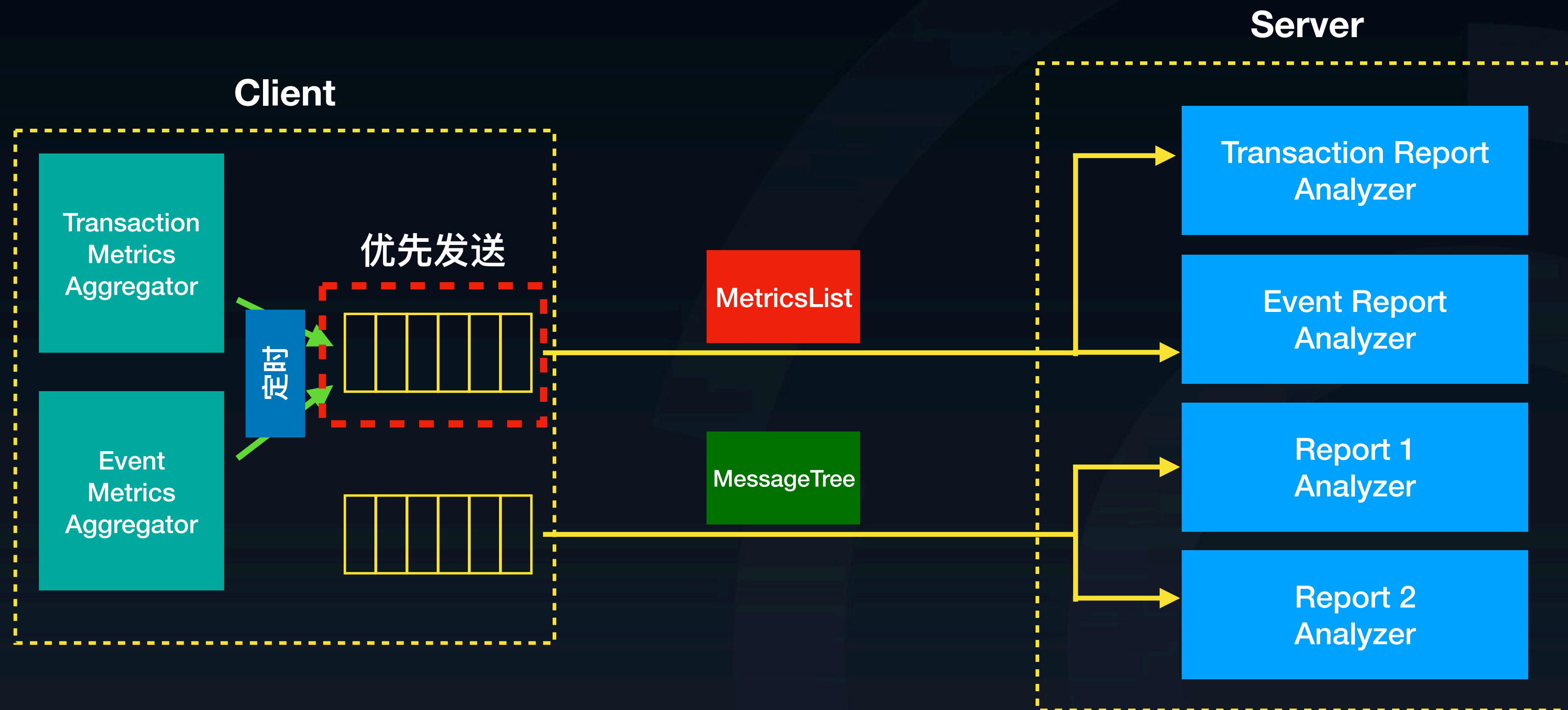


## 客户端计算

1. 合并多个MessageTree的统计
2. 一次发送





# Transaction/Event报表客户端计算



# 效果

NAME	STATE	CPU Usage Per Core(%) ↓
AnalyzerThreadFor-transaction-1	 RUNNABLE	90.9
AnalyzerThreadFor-transaction-7	 RUNNABLE	85.3
AnalyzerThreadFor-transaction-2	 RUNNABLE	89.93
AnalyzerThreadFor-transaction-3	 RUNNABLE	89.3
AnalyzerThreadFor-transaction-6	 RUNNABLE	89.11
AnalyzerThreadFor-transaction-5	 RUNNABLE	87.05
AnalyzerThreadFor-transaction-4	 RUNNABLE	88.37



NAME	STATE	CPU Usage Per Core(%)
AnalyzerThreadFor-transaction-6	 WAITING	2.38
AnalyzerThreadFor-transaction-5	 WAITING	2.27
AnalyzerThreadFor-transaction-2	 WAITING	2.24
AnalyzerThreadFor-transaction-7	 WAITING	2.24
AnalyzerThreadFor-transaction-1	 WAITING	2.19
AnalyzerThreadFor-transaction-3	 WAITING	2.19
AnalyzerThreadFor-transaction-4	 WAITING	2.14

# 效果

NAME	STATE	CPU Usage Per Core(%)
AnalyzerThreadFor-event-4	 RUNNABLE	50.08
AnalyzerThreadFor-event-1	 RUNNABLE	47.96
AnalyzerThreadFor-event-3	 RUNNABLE	57.96
AnalyzerThreadFor-event-2	 RUNNABLE	67.88



NAME	STATE	CPU Usage Per Core(%)
AnalyzerThreadFor-event-4	 RUNNABLE	1.08
AnalyzerThreadFor-event-1	 RUNNABLE	0.96
AnalyzerThreadFor-event-3	 RUNNABLE	1.03
AnalyzerThreadFor-event-2	 RUNNABLE	1.35

# 客户端影响

- 内存 10M以下
- CPU 0.1%以下影响



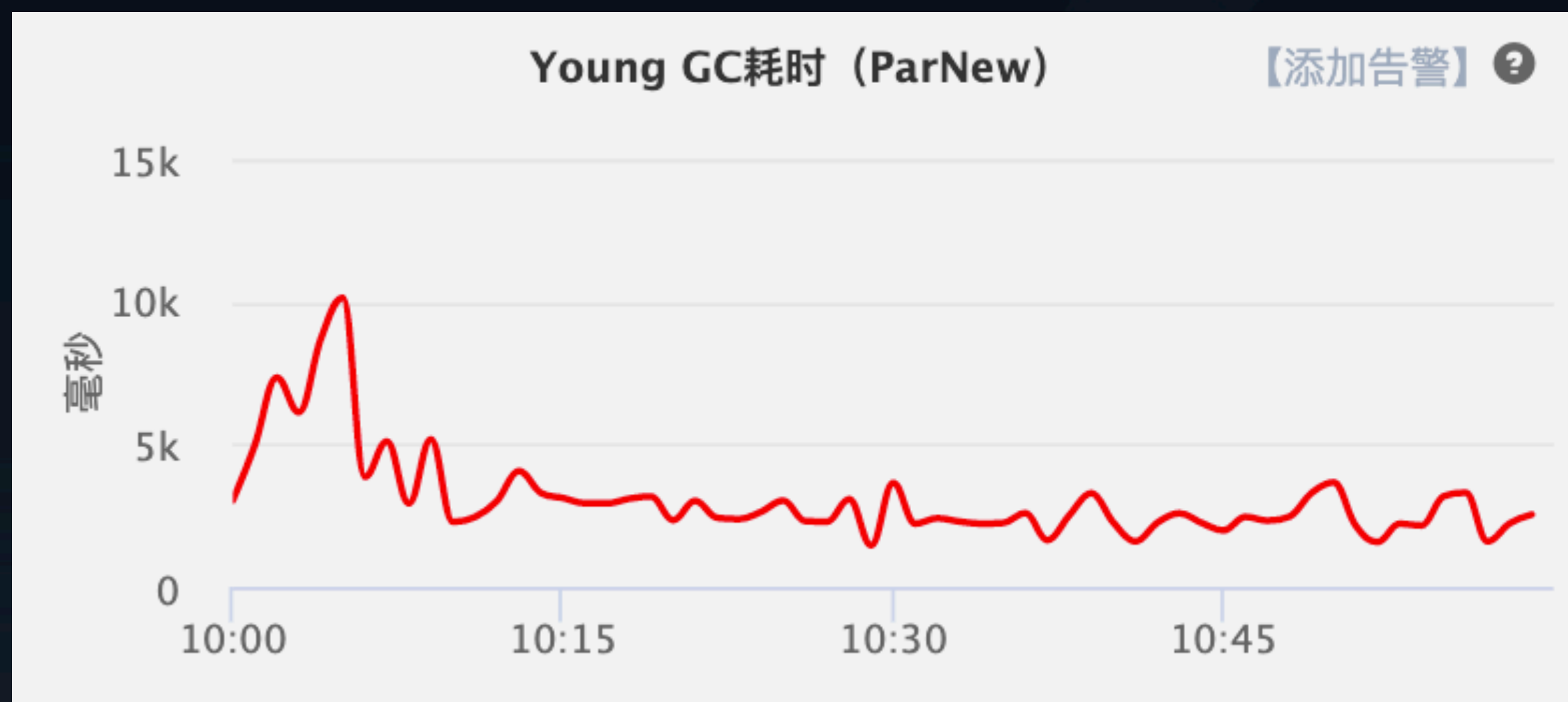
# 小结

- 对于简单、变更少的逻辑可以考虑客户端计算
  - 客户端计算复杂度比服务端计算低
  - 服务端计算量只和客户端数量及间隔时间有关

## #3 Report双缓冲

# 遇到的问题

- 每小时前几分钟发生数据丢失



# Cat服务端内存使用

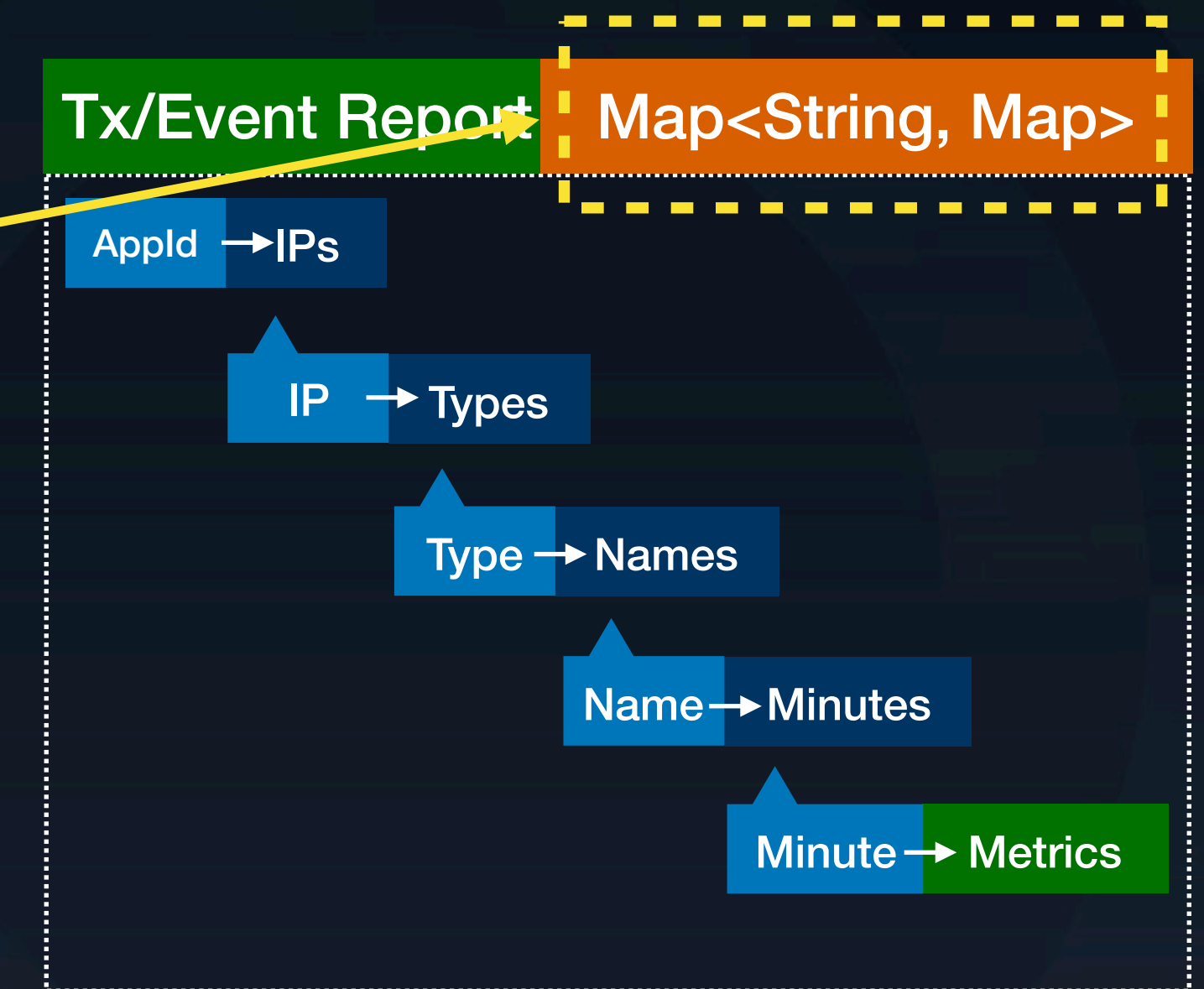
- 网络过来的数据      流量平稳
- 当前小时Report

# Cat Report的生命周期

- 当前小时Report常驻内存
- 跨小时会创建一份新的Report, 并把上小时Report持久化到存储

不断创建下层Map, 每个Map不断resize

从Young区移到Old区, 很多没有用的Young GC, Old区不断增加

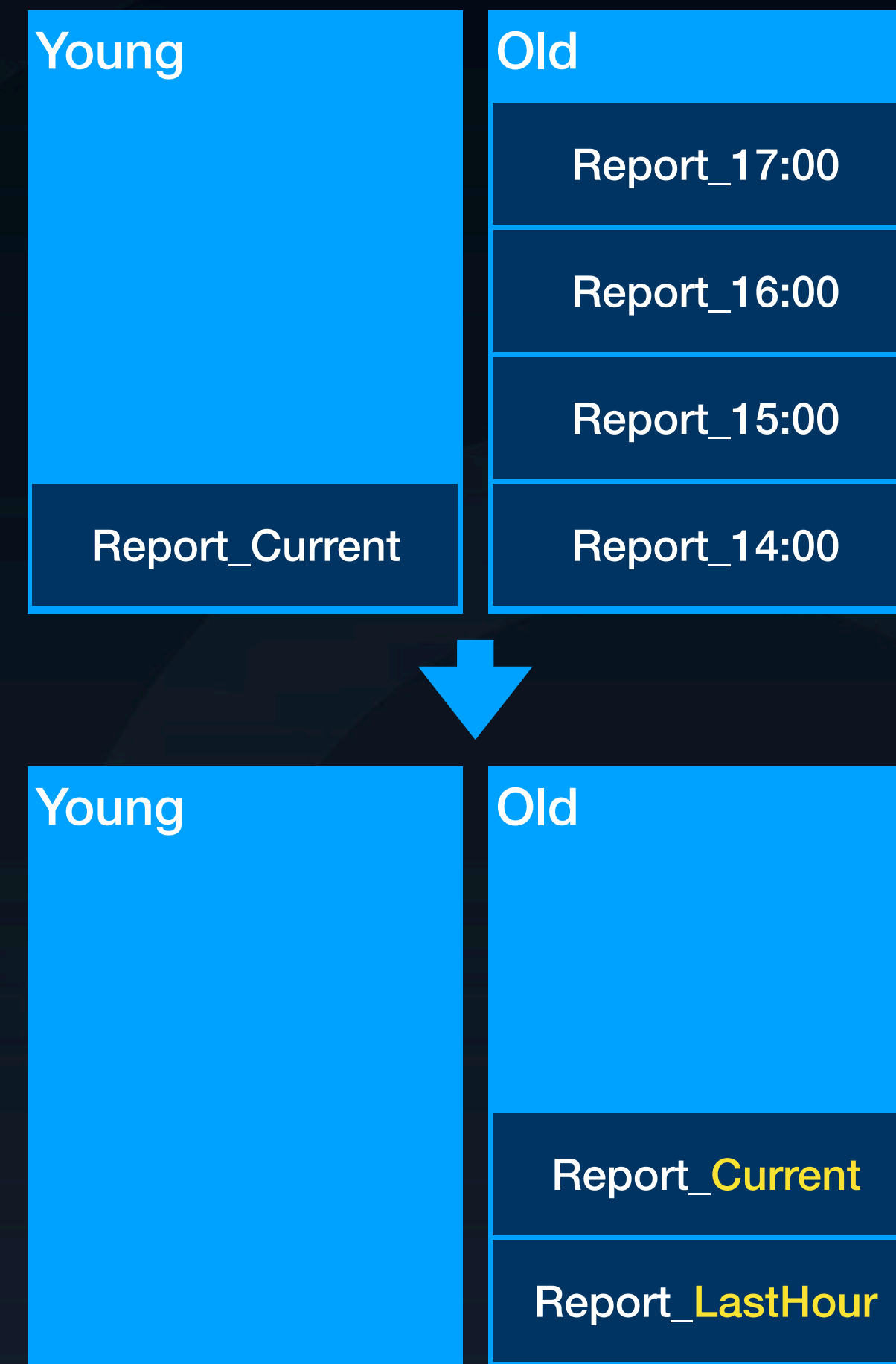




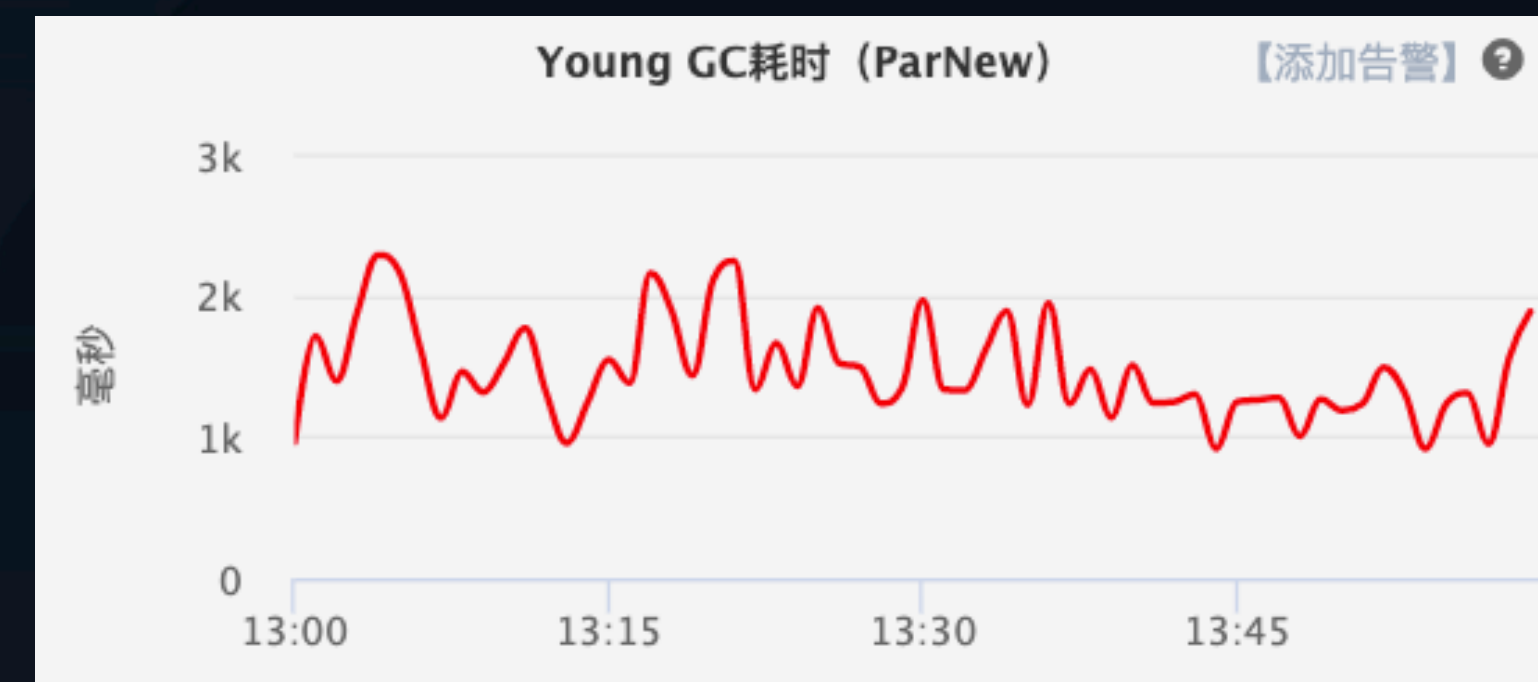
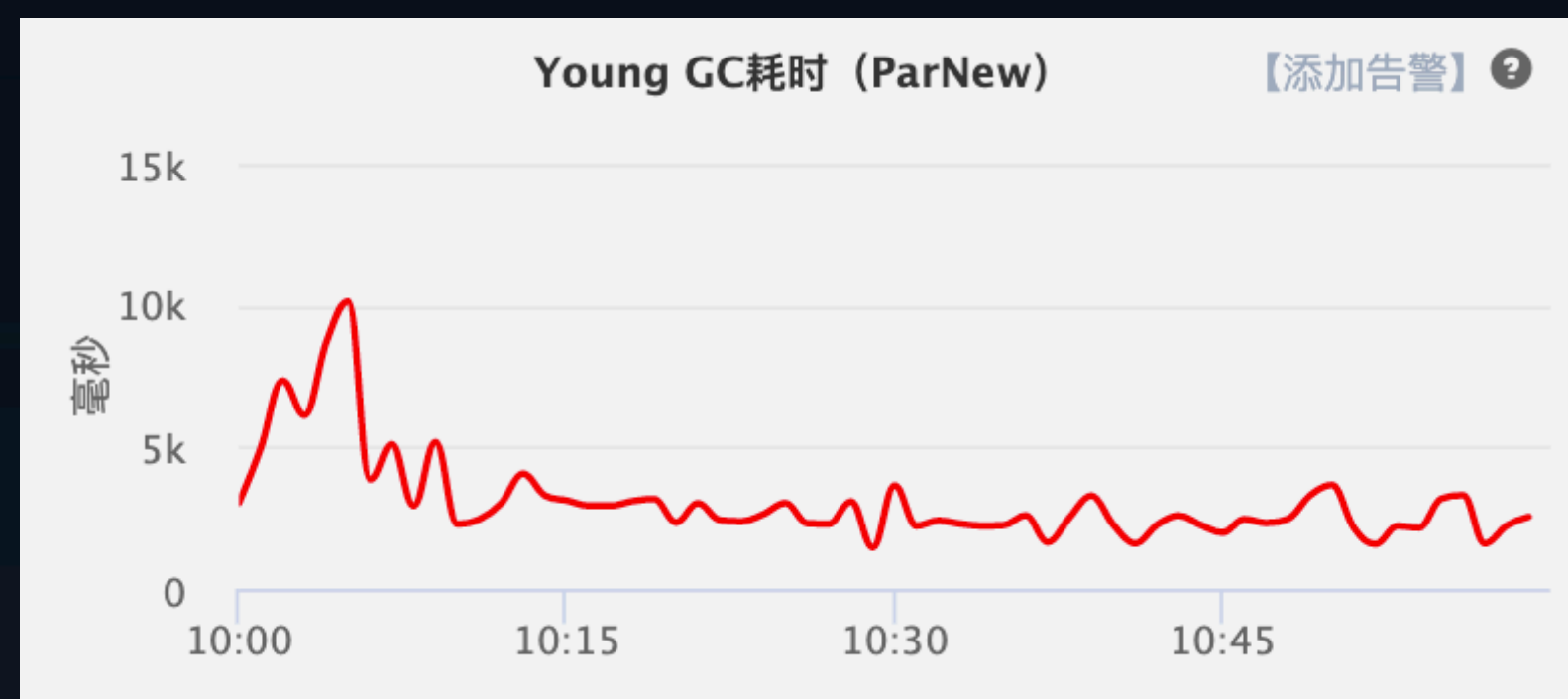
# 分析问题

- 不断resize Clone Report 供下个小时使用
- 不断创建下层Map
- 无用Young GC
- Old区不断增加

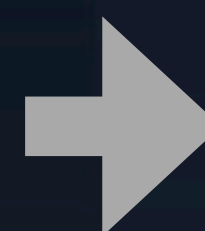
内存中保留两份 Report 轮换使用



# 效果



**Full GC 每天20次**



**Full GC 每天3次**

# 小结

- GC问题
  - 尽量少分配内存
  - 是否可以复用内存

# #4 字符串

1分钟内内存分配：String对象

堆栈跟踪	总 TLAB 大小
▼ com.dianping.cat.message.spi.codec.CodecHelper.readString(ByteBuf, byte[])	12.05 GB
▼ com.dianping.cat.message.codec.BinaryMessageDecoder6\$Context.readString(ByteBuf)	11.59 GB

1分钟内内存分配：char[]

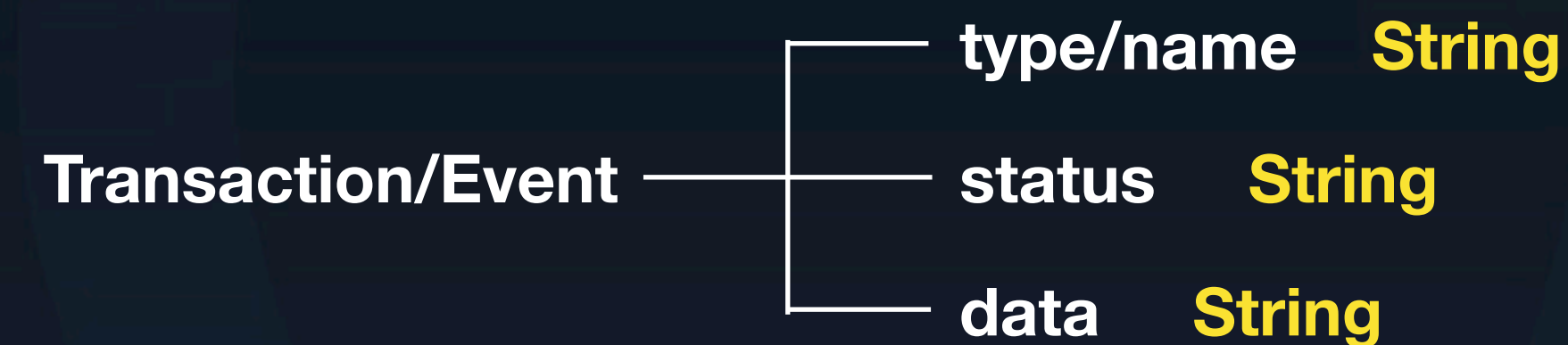
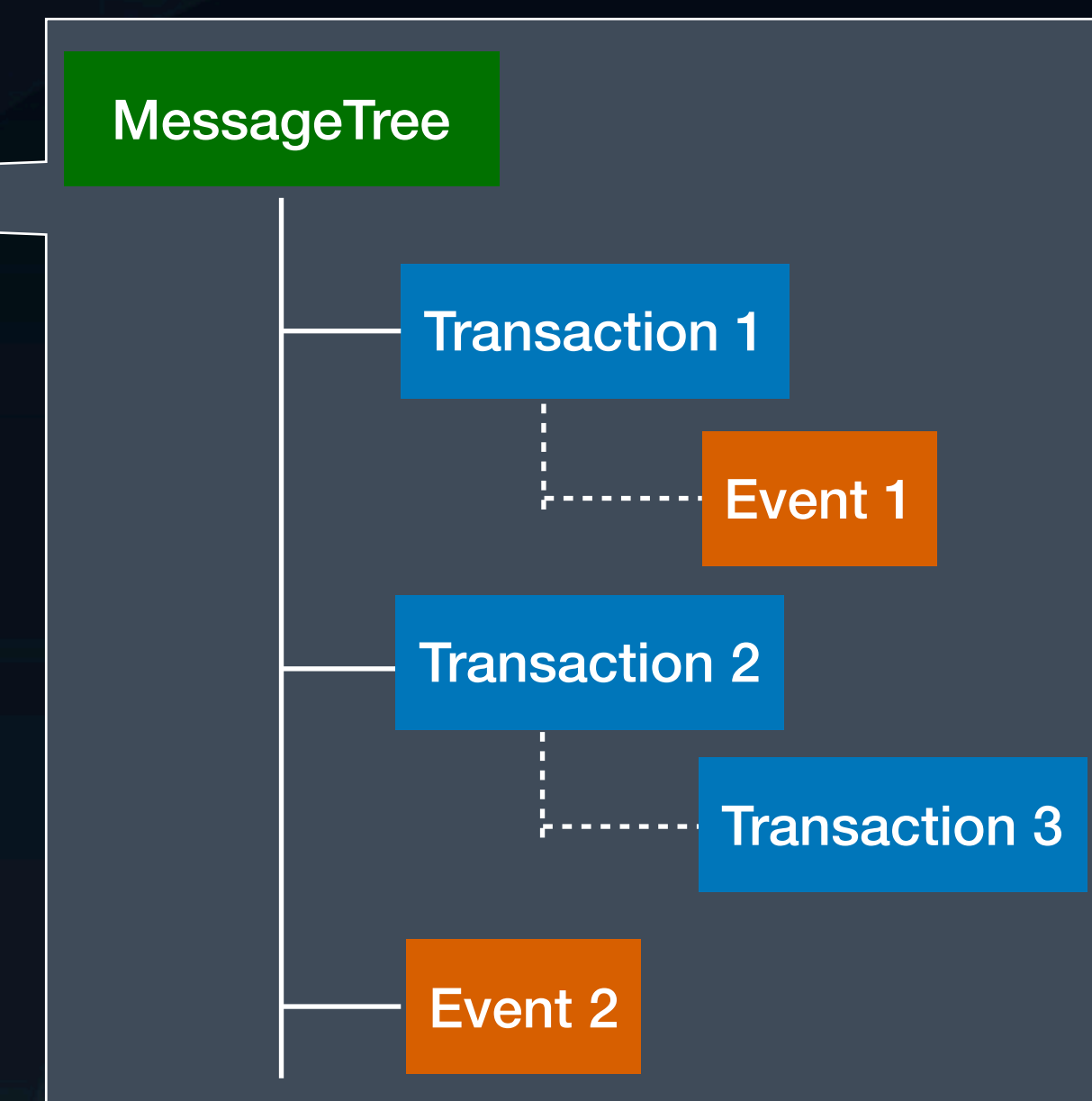
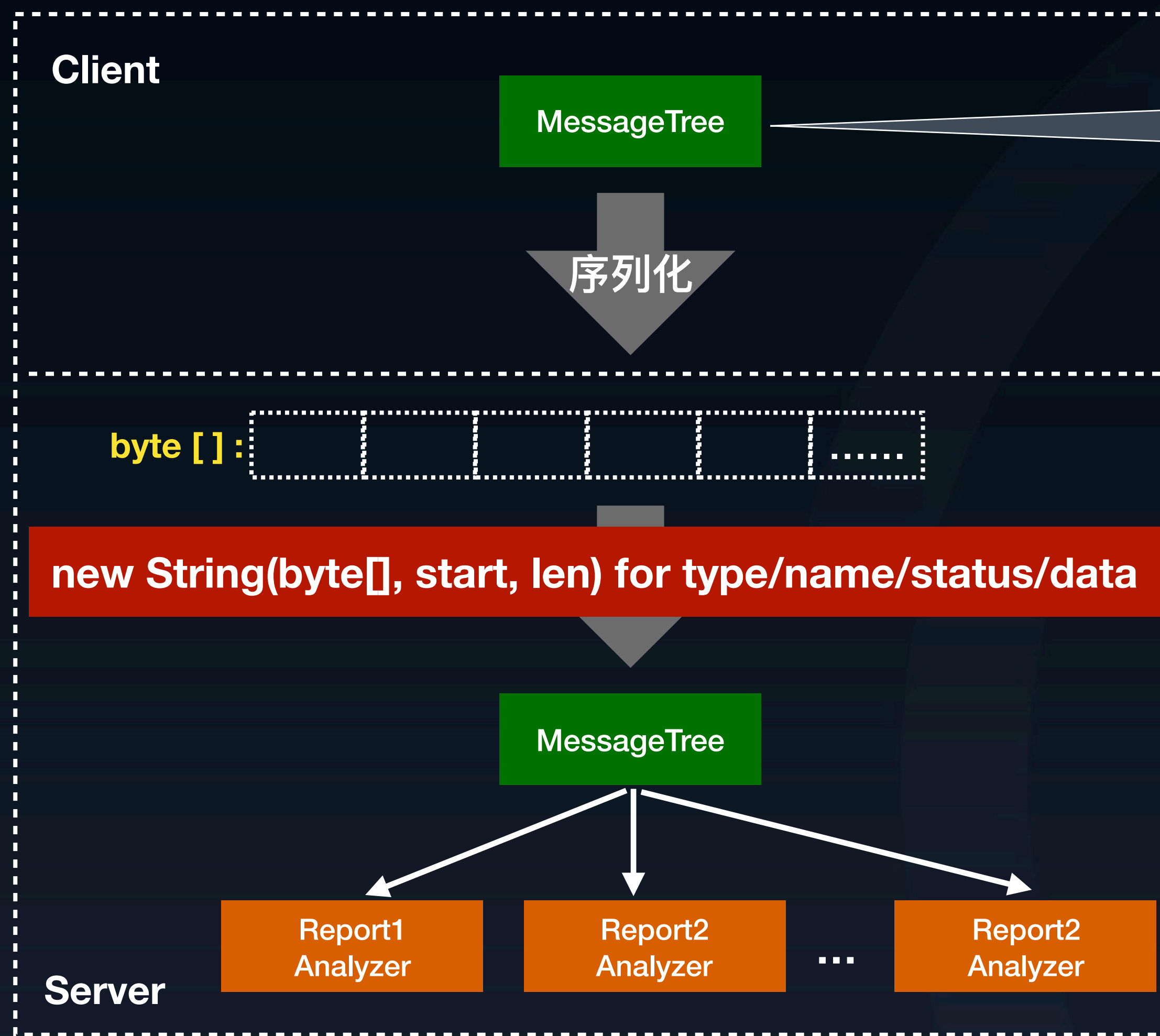
堆栈跟踪	总 TLAB 大小
▼ java.lang.StringCoding.decode(Charset, byte[], int, int)	27.10 GB
▼ java.lang.String.<init>(byte[], int, int, Charset)	27.10 GB
▼ com.dianping.cat.message.spi.codec.CodecHelper.readString(ByteBuf, byte[])	26.83 GB
▶ com.dianping.cat.message.codec.BinaryMessageDecoder6\$Context.readString(ByteBuf)	26.07 GB

1分钟内内存分配：UTF\_8\$Decoder

堆栈跟踪	总 TLAB 大小
▼ sun.nio.cs.UTF_8.newDecoder()	24.74 GB
▼ java.lang.StringCoding.decode(Charset, byte[], int, int)	24.74 GB
▼ java.lang.String.<init>(byte[], int, int, Charset)	24.74 GB
▼ com.dianping.cat.message.spi.codec.CodecHelper.readString(ByteBuf, byte[])	24.45 GB
▶ com.dianping.cat.message.codec.BinaryMessageDecoder6\$Context.readString(ByteBuf)	23.71 GB



# MessageTree的传输



# byte[] —> String

- **1次** new String(byte[], start, len, "UTF-8" )
  - **2次**创建char[]
  - **1次**字符集解码

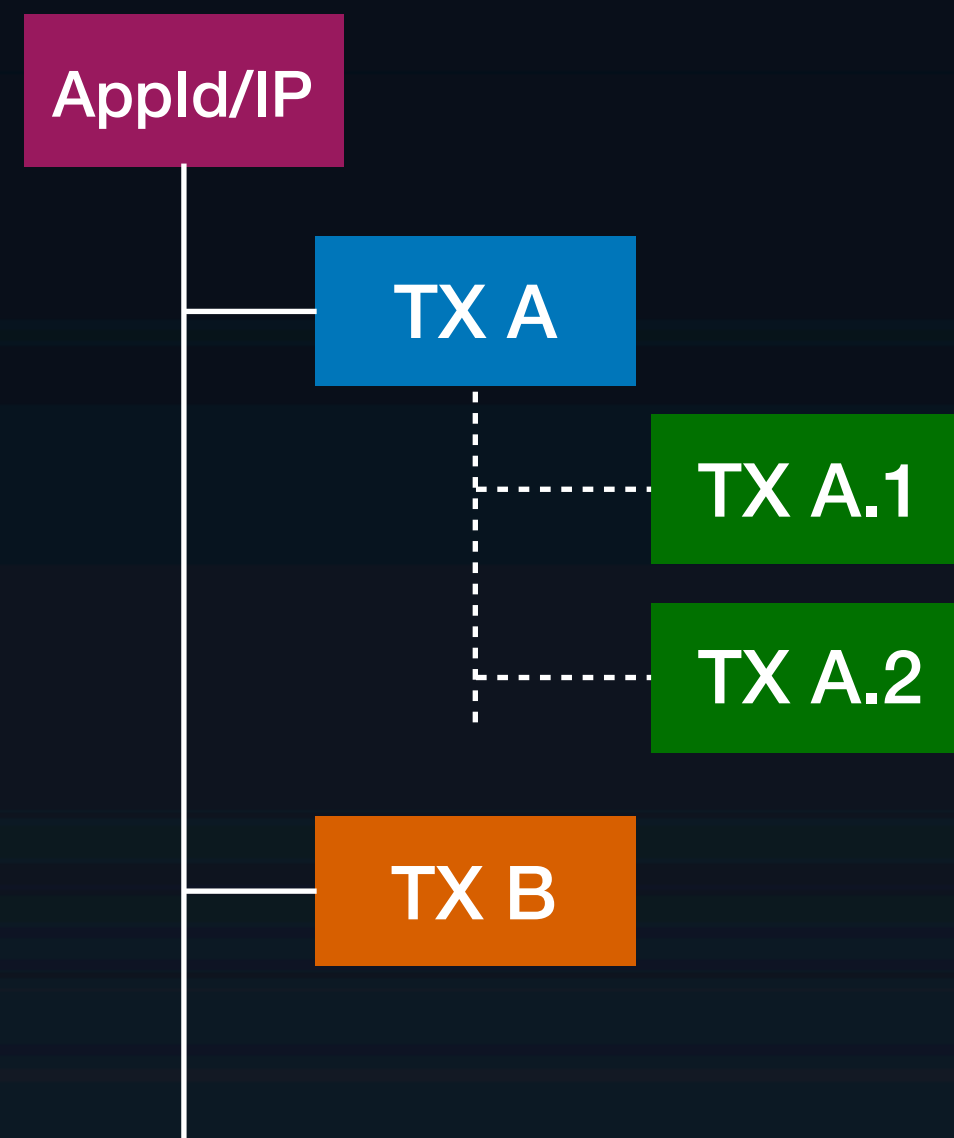
既消耗内存也消耗CPU

# byte[] —> String必须?

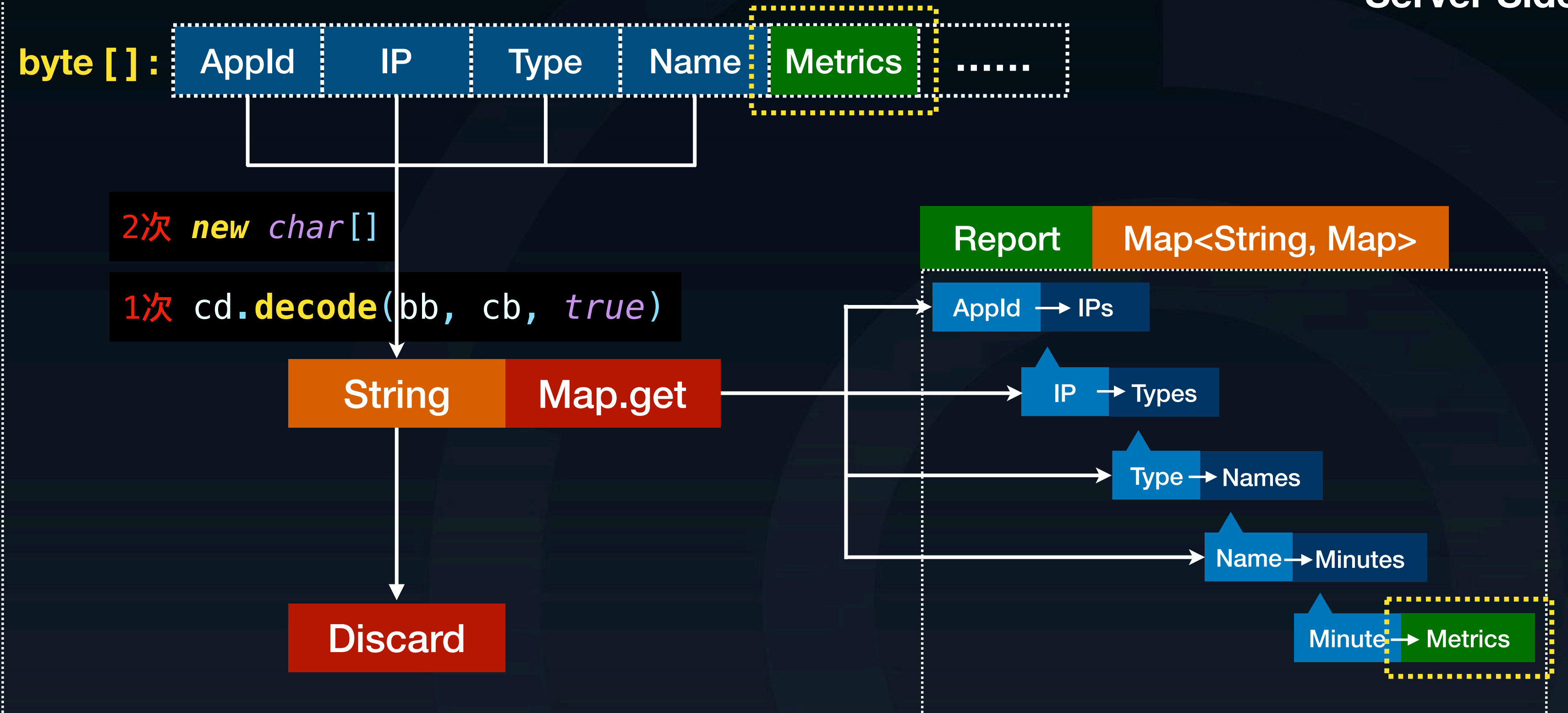
- **type/name**
- **status**      大部分Report只关心是否成功      特殊对待成功，其他状态按需lazy做即可
- **data**      不是所有Report都需要      按需lazy做即可

# type/name需要byte[] -> String?

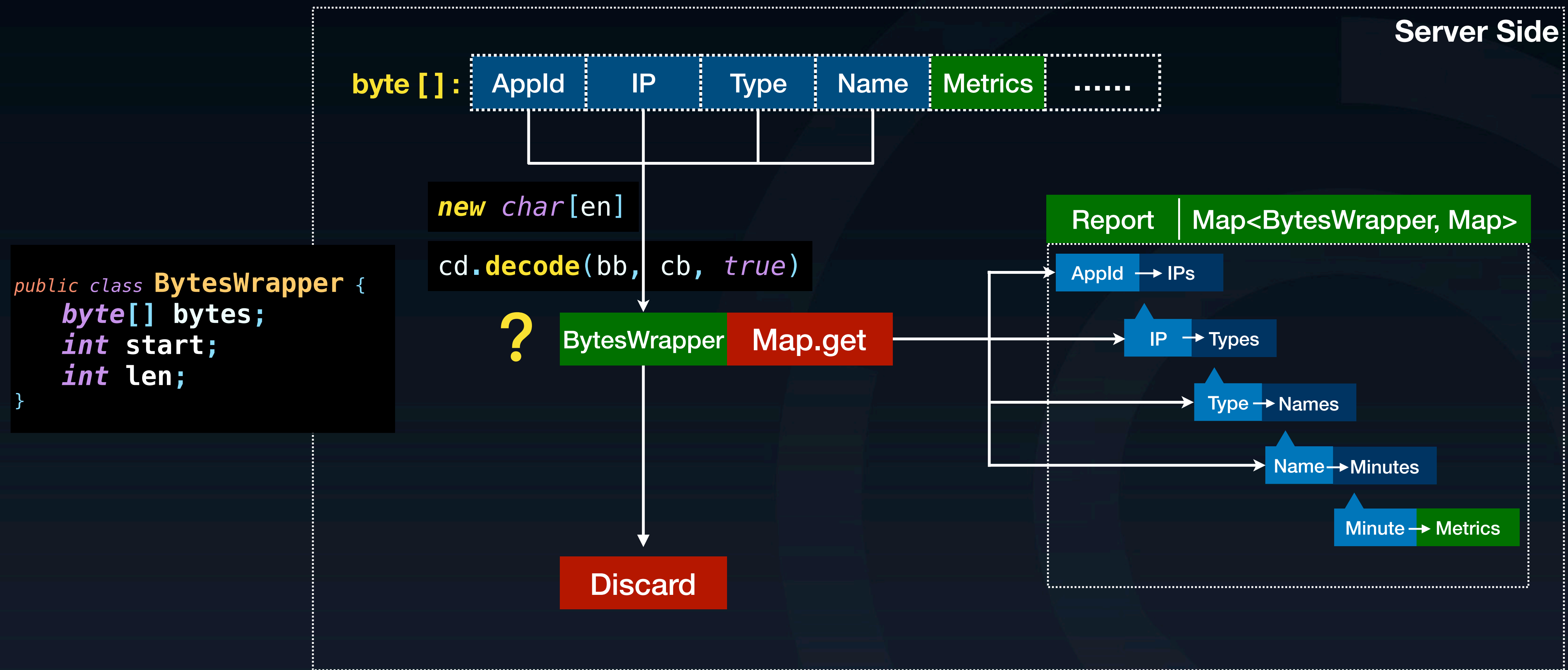
## Message Tree



## Server Side

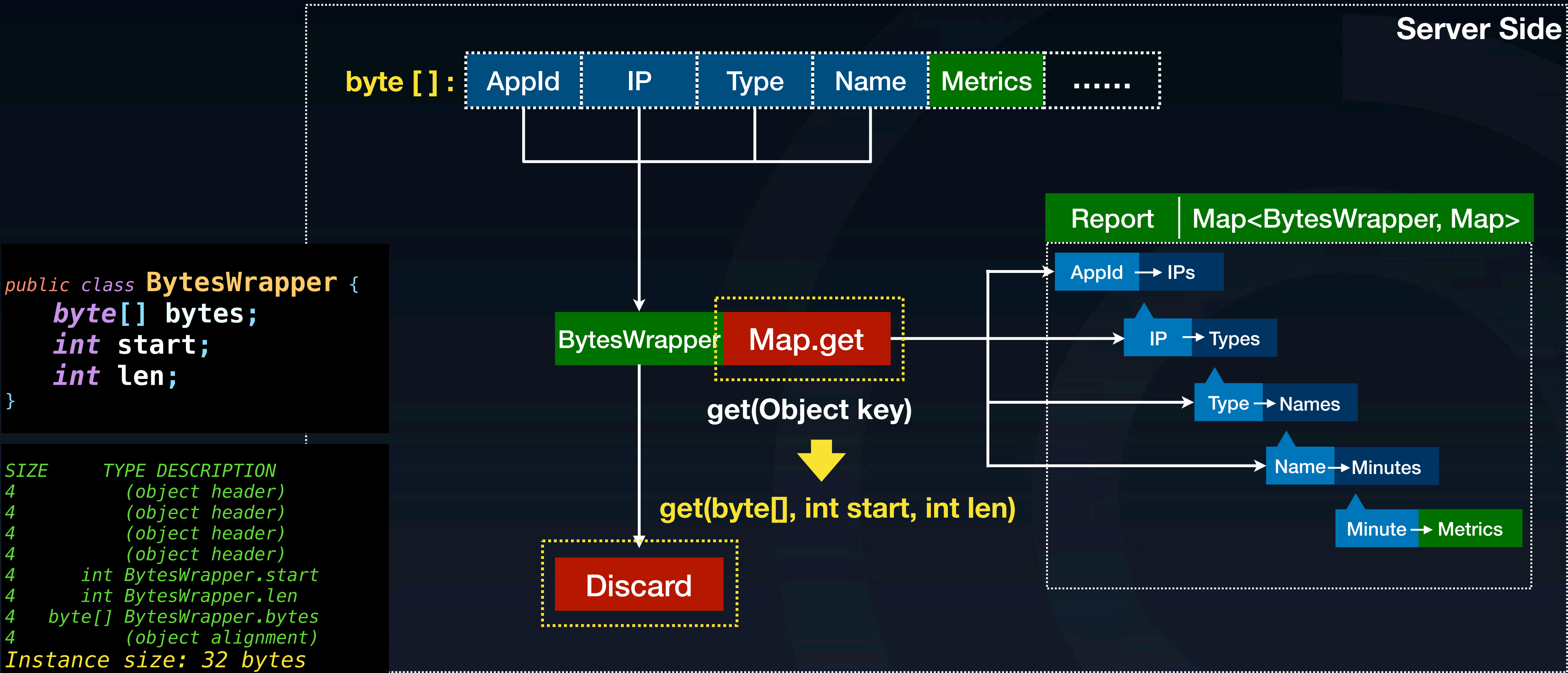


# type/name需要byte[] -> String?



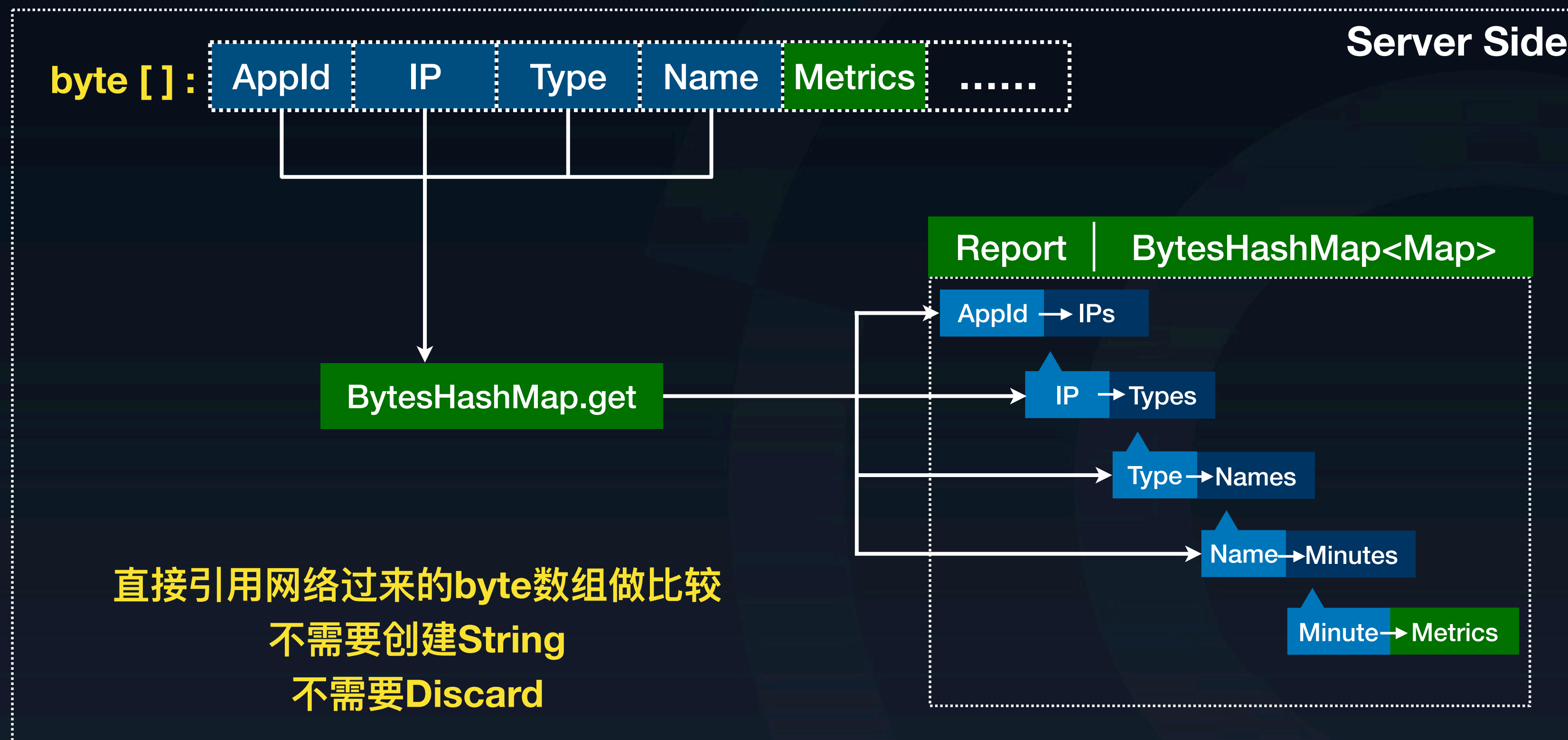


# 进一步思考



# 进一步思考

```
public class BytesHashMap<V> implements Map<byte[], V>{  
    public V get(byte[] bytes, int start, int len)  
}
```



# byte[] —> String必须?

- **type/name** Report改成BytesHashMap, 不需要
- **status**      大部分Report只关心是否成功      特殊对待成功, 其他状态按需lazy做即可
- **data**      不是所有Report都需要      按需lazy做即可

Young GC减少40%

# 小结

- 关注大量使用对象的创建
- `new String(byte[], start, len)`消耗大
  - 是否可以直接使用`byte[]`

# 目录

- CAT在携程
- CAT性能优化案例
- 总结和思考



# 优化思路

- CPU

- 减少额外损失

- 优化线程模型：上下文切换、锁竞争的消耗

- 减少不必要的操作

- 减少字符串构造：不必要的解码

- 服务端计算移到客户端

# 优化思路

- GC
  - 减少不必要的对象创建
    - 减少创建字符串，使用已有byte[]
    - 减少Report的重复创建、填充和resize
  - 内存复用
    - Report的内存双缓冲复用



# InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播



关注InfoQ网站  
第一时间浏览原创IT新闻资讯



免费下载迷你书  
阅读一线开发者的技术干货

THANKS! | QCon <sup>th</sup>