ArchSummit
全球架构师峰会 2017

主办方 Geekbang 极客邦科技 InfoQ

musical.ly 基于社交关系的Smart Feed架构

杜鹏
musical.ly 高级Java架构师

主办方 Geekbang 极客邦科技 InfoQ

AiCon

全球人工智能与机器学习技术大会

助力人工智能落地

2018.1.13 - 1.14　北京国际会议中心

扫描关注大会官网

极客时间

重拾极客精神·提升技术认知

下载极客时间App

获取有声IT新闻、技术产品专栏，每日更新

扫一扫下载极客时间App

SPEAKER INTRODUCE

杜鹏 | Cooper Du

musical.ly 高级Java架构师

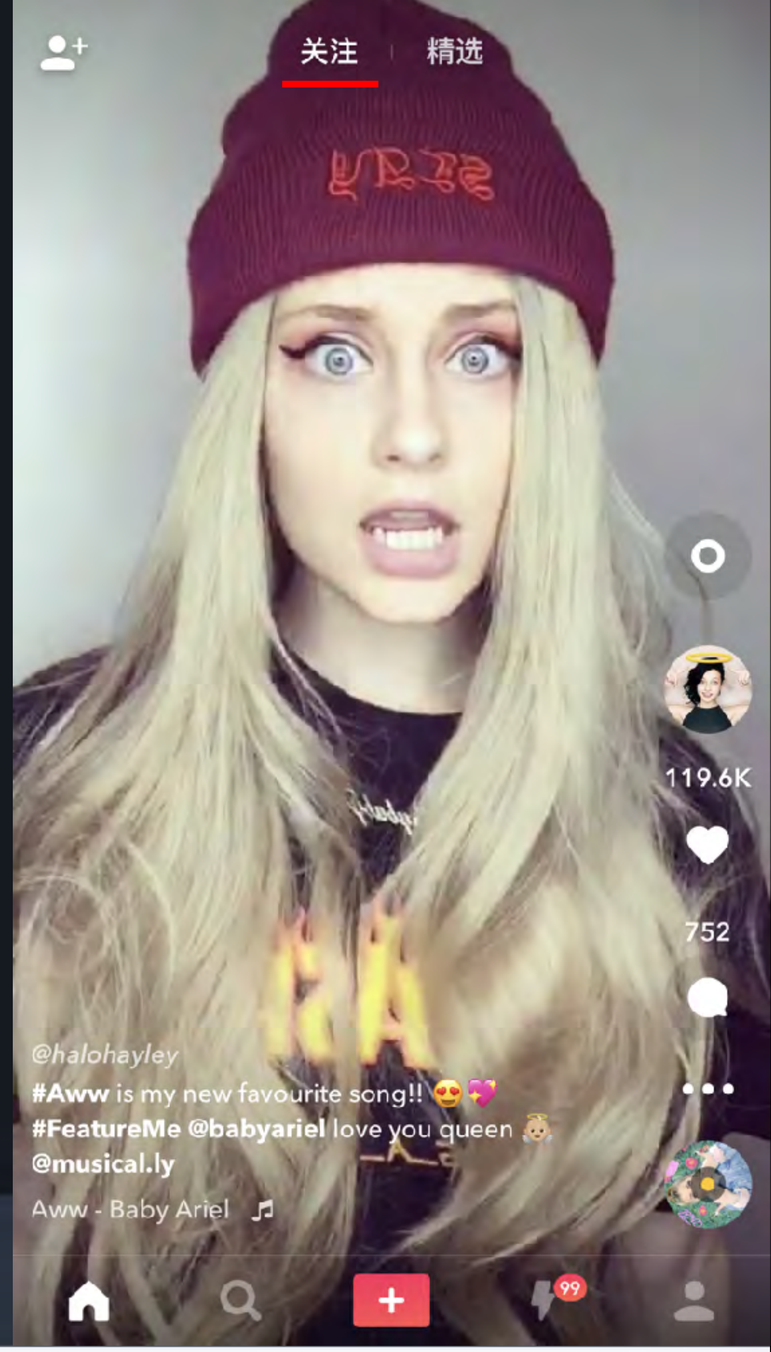
2016年初加入musical.ly，在musical.ly担任高级Java架构师，负责musical.ly的业务架构和基础架构工作。

TABLE OF
CONTENTS 大纲

- 业务背景介绍
- 产品技术背景介绍
- Smart Feed设计思路和技术架构
- 实践中的重难点
- 总结

关于musical.ly

- musical.ly是一个短视频社交应用，视频的生产和消费是驱动社区发展的主干功能
- 用户的主要内容消费场景之一是关注栏目
- musical.ly的关注栏目是基于时间线的feed流



业务背景介绍

2017年2月

1.5亿用户

1500万
DAU

每日
1200万短视频

用户基数和内容生产数量已经达到一定规模
提高内容消费即分发效率成为公司的核心目标之一

TABLE OF
CONTENTS 大纲

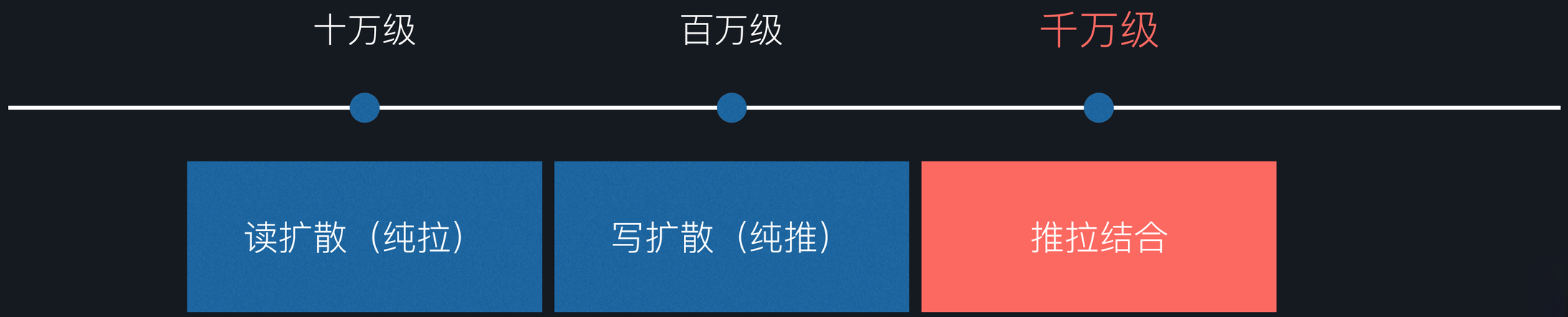
- 业务背景介绍
- 产品技术背景介绍
- Smart Feed设计思路和技术架构
- 实践中的重难点
- 总结

Feed流简介

musical.ly 关注栏目

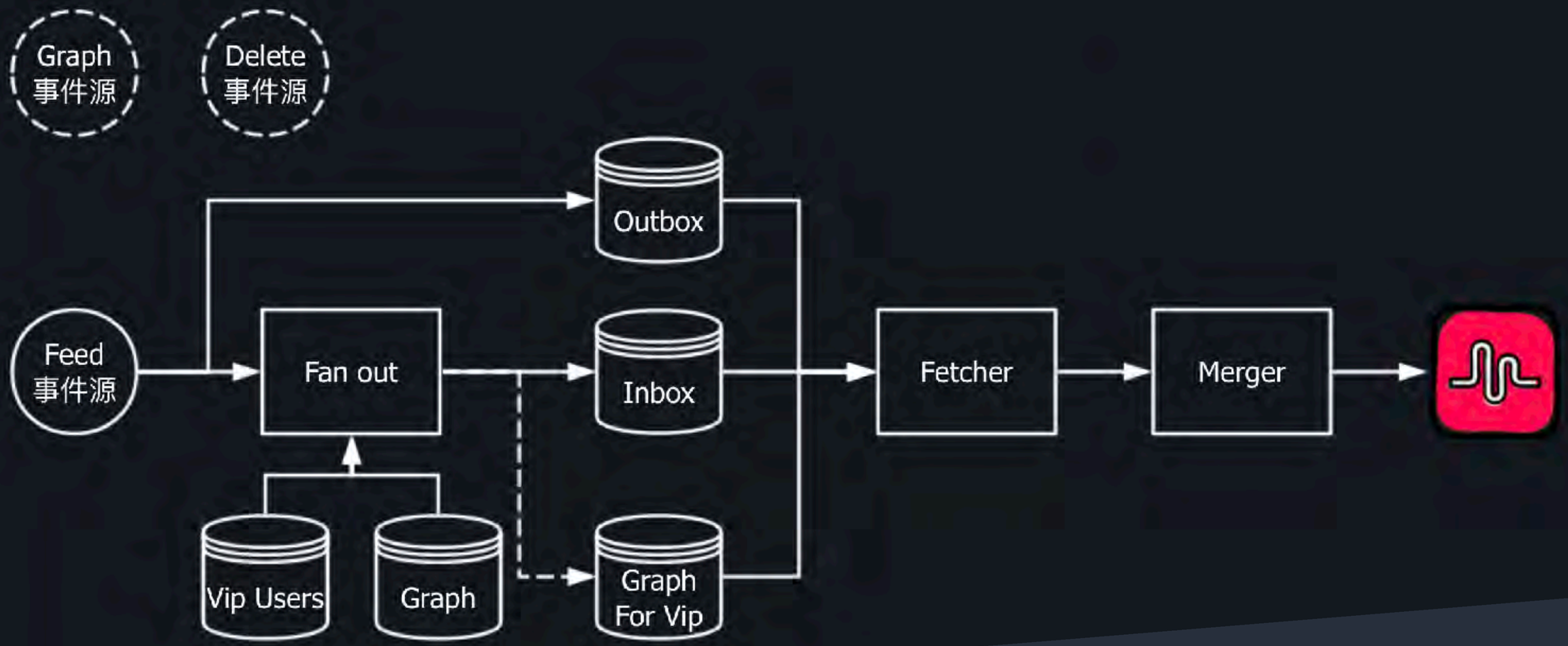
内容来源	社交关系	兴趣	其他
产品形态	时间线	瀑布流	其他

时间线产品的常见架构



注：纯推的方案还是推拉结合，另外一个重要的因素是社交关系结构--即大V的粉丝数量。

最简单的推拉结合模型


Graph
事件源
Delete
事件源
Outbox
Feed
事件源
Fan out
Inbox
Fetcher
Merger
Vip Users
Graph
Graph
For Vip

推拉结合改进：Vip发件箱数据存取

- 问题：拉取时需要循环访问Vip的发件箱
- 解决方案：把Vip用户的发件箱独立出来，把循环逻辑向底层推。
- 效果：平均每个Vip用户的发件箱访问耗时从之前的约2ms，降低到0.2ms

推拉结合改进：异步拉取Vip数据

- 问题：少量用户关注了数百个Vip用户，导致拉取Feed时出现超时
- 解决方案：拉取Feed时，限制同步拉取Vip Feed的数量，新增异步拉取的流程
- 效果：线上接口访问超时减少90%
- 劣势：部分Vip的Feed是异步完成分发的，用户可能会错过一些精彩内容

推拉结合改进：推-拉-推结合

- 问题：异步拉取Vip数据导致的Vip Feed内容丢失
- 解决方案：把 “推拉结合” 的方案进一步改进为 “推-拉-推结合”。
  - 定义关注超过500个Vip的用户为 “Vip Zealot”
  - Vip发布内容后，向所有的“Vip Zealot”进行分发
  - “Vip Zealot”拉取Feed时，不再拉取Vip的Outbox
- 效果：解决Vip Feed内容丢失问题，同时不降低在线接口访问性能

Vip用户的社交关系数据的演进

- 接口需求：对于给定用户，返回其关注的所有Vip User
- 实现：一开始，我们使用“大表分区、小表复制”的方式实现
- 新的问题：随着流量增加、数据量增大，上述实现逐渐成为系统瓶颈。最后在实现“Vip Zealot”时，对这部分数据做了冗余
- 效果：99线从超过50ms，下降到约12ms

纯时间线产品的不足

- 错过精彩内容：用户消费信息的顺序是固定的。存在错过精彩内容、感兴趣的内容、强社交关系内容的情况。
- 抑制消费兴趣：部分低质量的内容抑制了用户继续消费的兴趣。
- 个性化单一：用户通过“关注好友”或者“订阅”和信息来源建立关系，满足用户的个性化定制需求，但这种方式是一元的个性化。

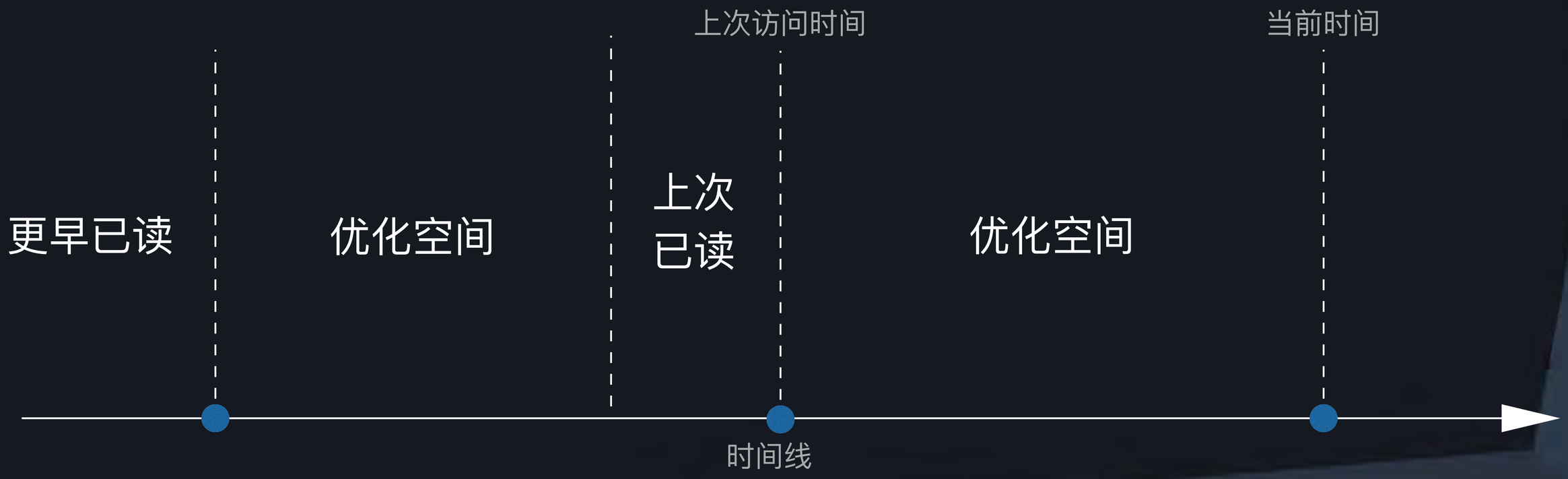
TABLE OF
CONTENTS 大纲

- 业务背景介绍
- 产品技术背景介绍
- Smart Feed设计思路和架构
- 实践中的重难点
- 总结

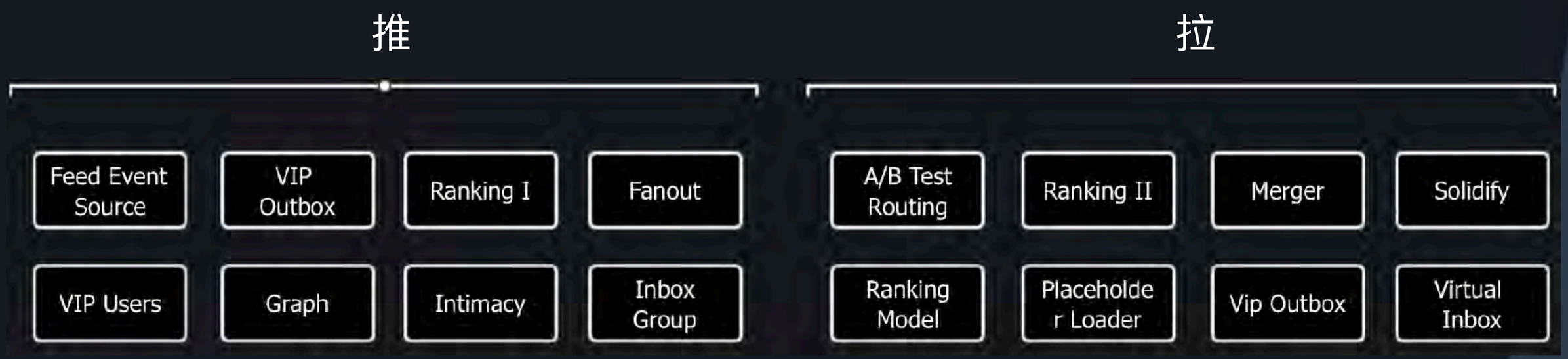
时间线产品进行Smart Feed优化

时间线产品进行Smart Feed优化

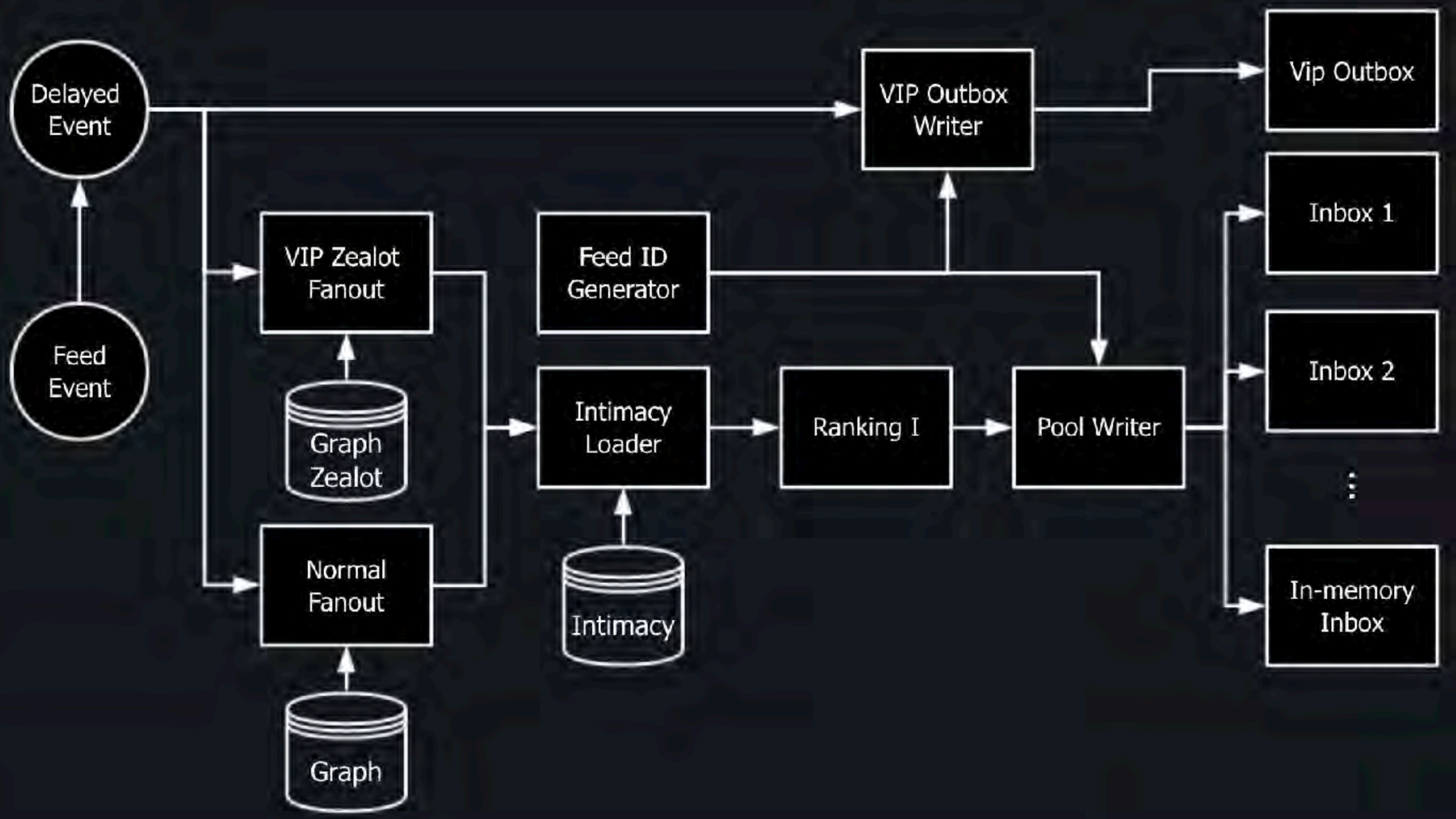
时间线产品进行Smart Feed优化



Smart Feed的系统组成



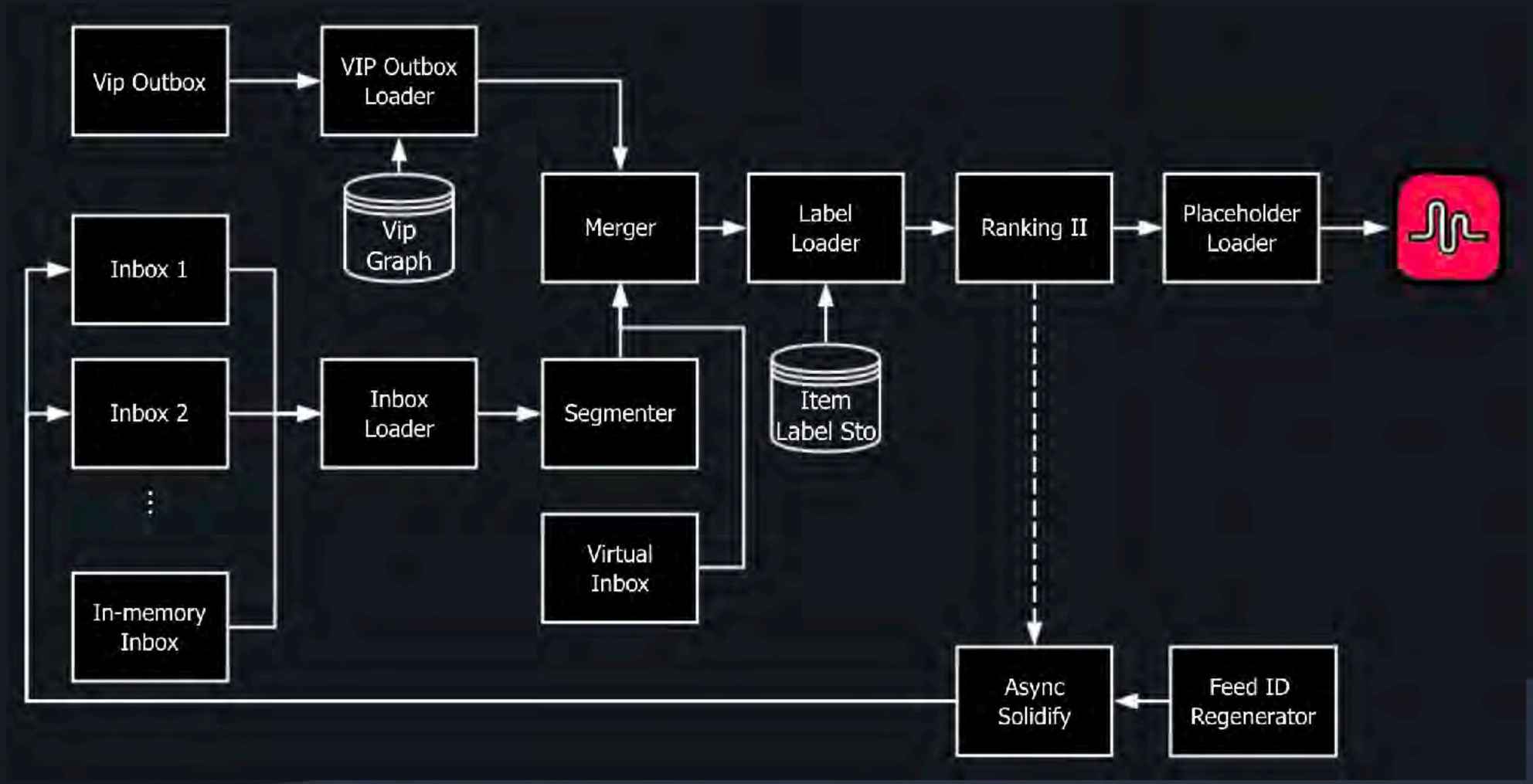
Smart Feed分发流程



Smart Feed系统和Feed系统分发流程上的区别

- 事件源需要增加延迟，留出时间对视频进行内容理解
- 根据社交网络的特点，增加了用户间的亲密度数据源
- 对同一个用户存在多个Inbox，在分发阶段进行打分，根据打分结果将数据分发到不同的Inbox中（Inbox、Priority Inbox）
- 分发过程增加Feed ID，由ID Generator生成。保证顺序性、唯一性、稀疏性。

Smart Feed拉取流程



Smart Feed系统和Feed系统拉取流程上的区别

- 需要从当前用户的多个Inbox拉取数据
- 数据合并后根据是否已被固化分成多个Segment
- 从Vip Outbox中拉取Feed需要根据Segment的结果进行
- 在精排后，向用户展示的内容需要在Inbox中进行固化，固化时需要重写Feed ID

Smart Feed存储数据结构

	Feed	Smart Feed
User ID	✓	✓
Item Type	✓	✓
Item ID	排序列	✓
Feed ID	-	排序列，固化时会批量重设
Bottom	-	固化Segment信息
Item Features	-	分发时冗余的Feed特征，包括视频tag、cv label、作者国家地区等信息

Feed ID生成逻辑

- 使用Redis Lua Script实现，每个用户用Hash数据结构维护状态
- 目的是为了使生成的Feed ID是稀疏的，后期可以在任意两个ID之间插入内容

Timestamp	Timestamp_Ms << 17
Status (3 bits)	Feed状态，初始值为0，固化后为1，其余值用作一些特殊情况
Sequence (10 bits)	固化顺序，分发时统一填充为0，固化时根据Ranking II的结果依次填充排序值
VIP (1 bit)	
Preserved_Sequence (3 bits)	预留，初始值为0，用于特殊情况下，在已经完成固化后继续插入内容

TABLE OF

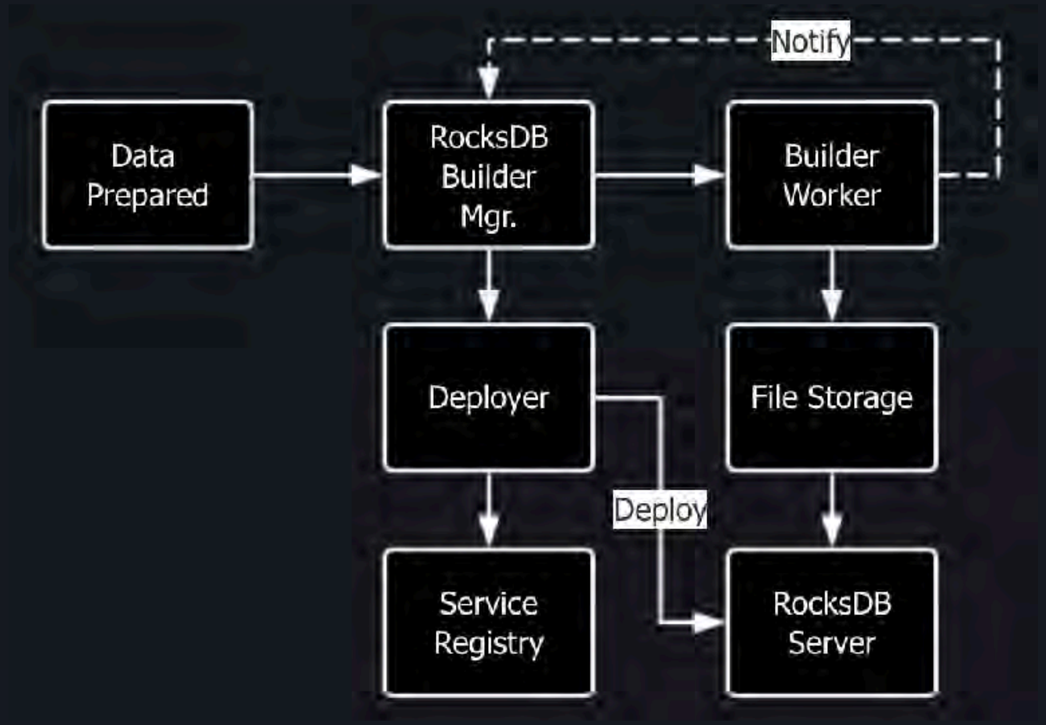
CONTENTS 大纲

- 回顾推拉结合模式下的Feed流架构
- Smart Feed的主要需求和挑战
- Smart Feed的设计思路和架构
- 实践中的重难点
- 总结

好友亲密度数据的挑战

- 数据量大：百亿级，每条记录需要保存10~20个浮点数，数据规模约1TB左右
- 变化频繁：在离散化（牺牲精度）之后，每日仍有超过一半数据需要更新
- 更新慢：一次全量更新需要超过2周时间，只能保证日活用户先更新
- 扩容困难：MySQL增加Slave之后 “追不上”

好友亲密度数据的挑战：RocksDB实现数据回流



- builder-mgr: 拆分和管理构建任务
- worker: 解析数据文件构建索引
- server: 提供在线查询
- smart-client: 实现sharding、新旧集群切换

使用RocksDB的优势

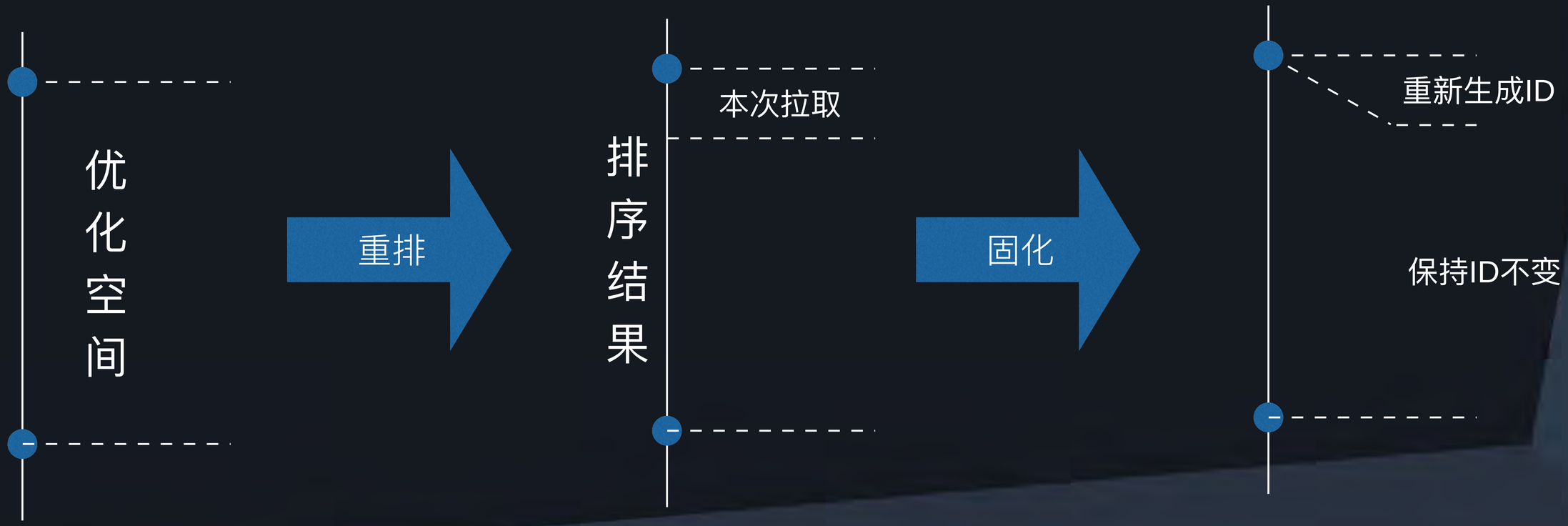
- 数据构建过程非常快，瓶颈基本只存在于数据解析
- 支持全量和增量数据更新
- 研发成本非常低，只需要开发4个功能非常单一的组件
- 数据更新过程对在线系统完全没有影响，节省了使用两个集群切换的成本
- 容易Scale：增加分片、增加实例
- Schema Free
- 一开始可以复用现有基础设施；后期仍有改进空间：直接使用SstWriter、重新实现服务端

使用RocksDB的优势

	上线前	上线后
写入速度	<1万	平均每个节点>5万
全量更新周期	大于2周	每日
扩容速度	大于1周	次日

数据固化的实现

Feed ID是排序列。利用Feed ID生成的稀疏性，通过重新生成Feed ID，把固化的Feed放在剩余数据的前面。



Vip Feed和普通Feed的合并

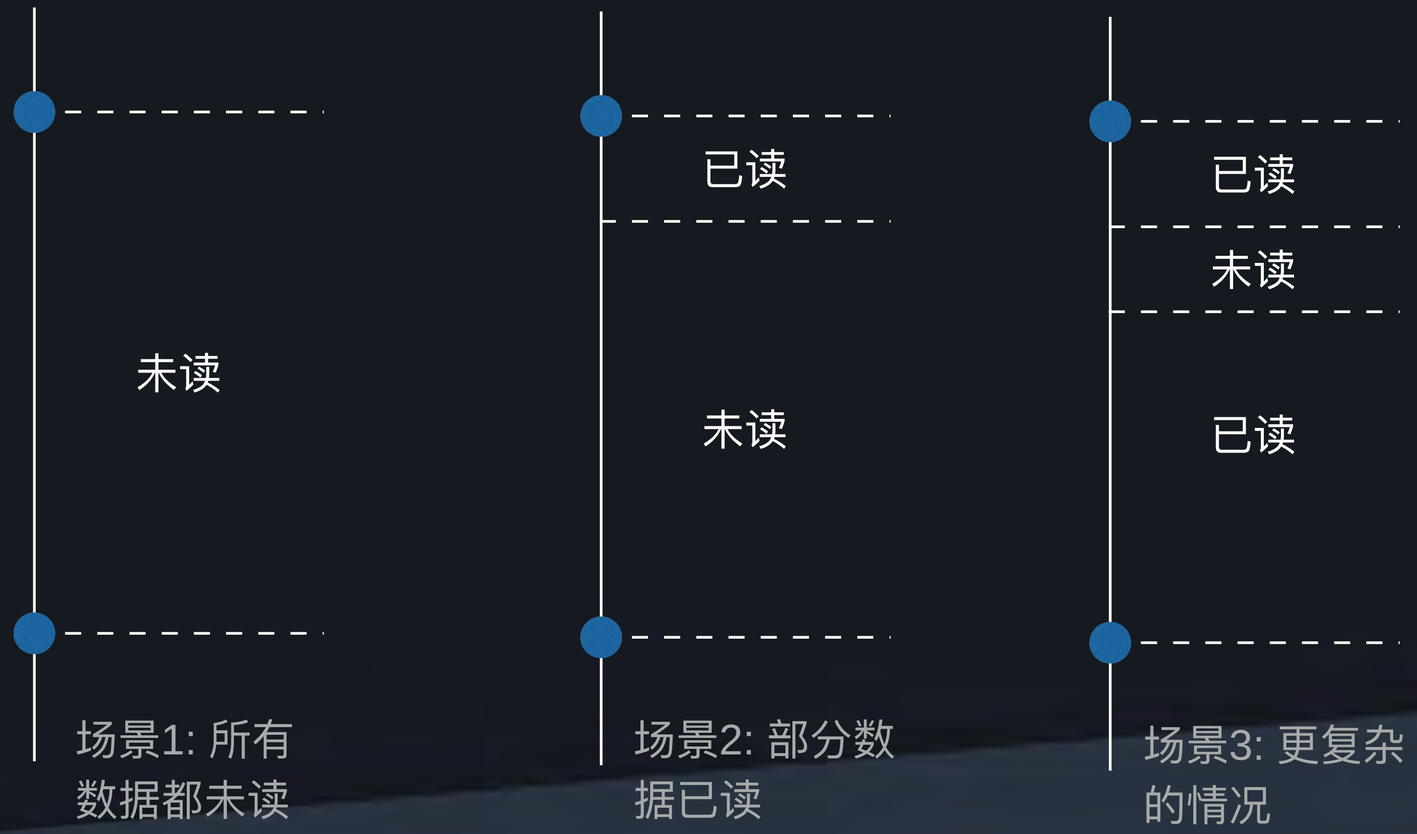
- 和普通Feed的推拉结合方案类似，Vip的数据会在拉取后异步的合并回用户的Inbox中。
- 主要有两方面的原因：
  - Vip发件箱的访问相对耗时较多，回写之后下次访问时不需要重复拉取。
  - 业务上需要保证Vip的内容顺序稳定
- Vip数据回写Inbox之后，需要在逻辑上保证不要重复拉取Vip的内容

Vip Feed和普通Feed的合并

从Inbox拉取数据后，对数据按是否固化，进行分段。

已固化的数据段对应的Vip Feed已经合并回Inbox。

未固化段需要根据相邻区间的边界，从Vip Outbox中拉取Vip Feed。



Vip Outbox的实现优化

- 使用Redis ZSET实现，以Feed ID作为Score
- 使用ZRANGEBYSCORE指令进行查询，使用Redis Script减少网络交互
- 为每个Vip准备多个副本，让压力更均匀的分担在每个分片上
- 拉取Vip Feed内容耗时下降到：平均 6ms，99线 9ms；未读数量查询耗时从4ms下降到2.36ms

TABLE OF

CONTENTS 大纲

- 业务背景介绍
- 产品技术背景介绍
- 设计思路和技术架构
- 实践中的重难点
- 总结

Smart Feed优化效果

UV

消费时长

Smart Feed技术指标

	feed	smart feed
拉取数量	20	200
平均响应时间	79 ms	118 ms

总结

- 时间线产品常见的推拉结合的架构及数据量增长之后的优化
- 时间线产品进行Smart Feed改进的产品需求
- musical.ly在Smart Feed上的技术探索，其系统架构和主要流程
- RocksDB，一种低实现成本的海量数据回流的方案

SPEAKER INTRODUCE

杜鹏 Cooper Du

musical.ly 高级Java架构师

2016年初加入musical.ly，在musical.ly担任高级Java架构师，负责musical.ly的业务架构和基础架构工作。

微信：Cooper_Du

THANK YOU

如有需求，欢迎至 [讲师交流会议室] 与我们的讲师进一步交流

ArchSummit
全球架构师峰会 2017