Flink 在数据分析中的应用

肖 强

TalkingData 架构师

Agenda

- TalkingData流处理的背景与痛点
- Flink在TalkingData SaaS分析中的演进路线
- Flink实践中的重点问题解决方案
- 总结与展望

1. TalkingData流处理的背景与痛点

TalkingData SaaS的流处理服务演进:

Jetty 服务

- 不易扩展与维护
- 性能问题

Before 2014

自研etl-framework

- 无法完整表达DAG
- 容错机制不足
- 性能问题

2014~2016

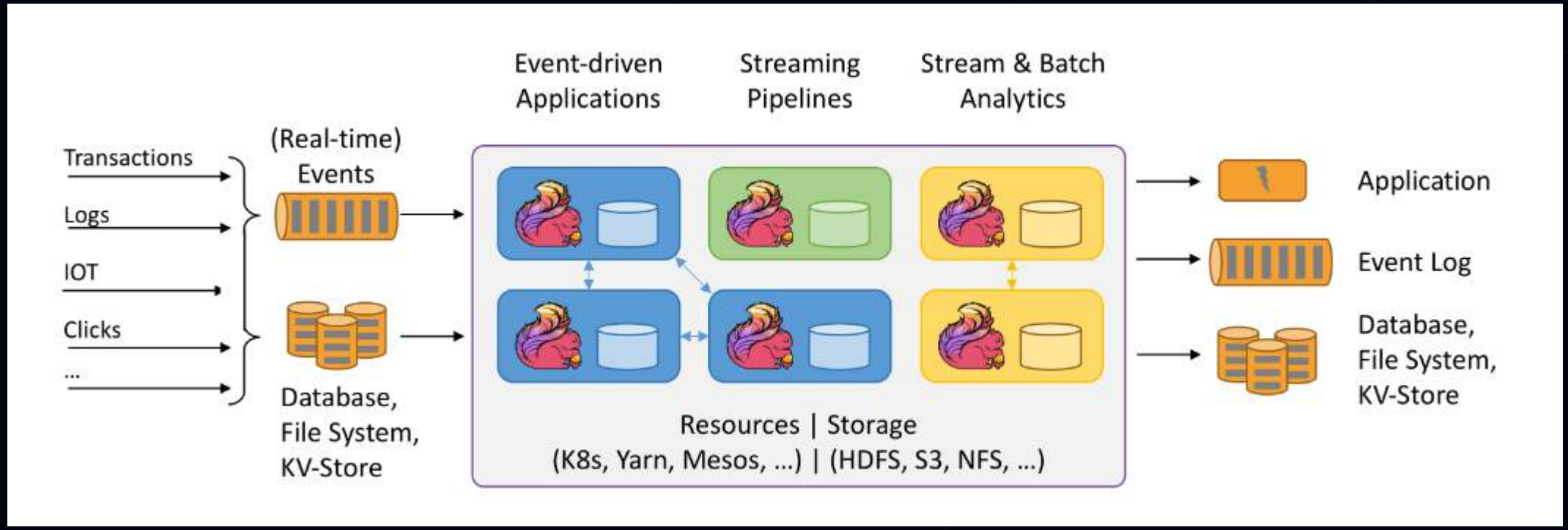
新的流处理系统

- 满足更大的业务量
- 满足更复杂的业务场景

2017~now

1. TalkingData流处理的背景与痛点

Flink是什么



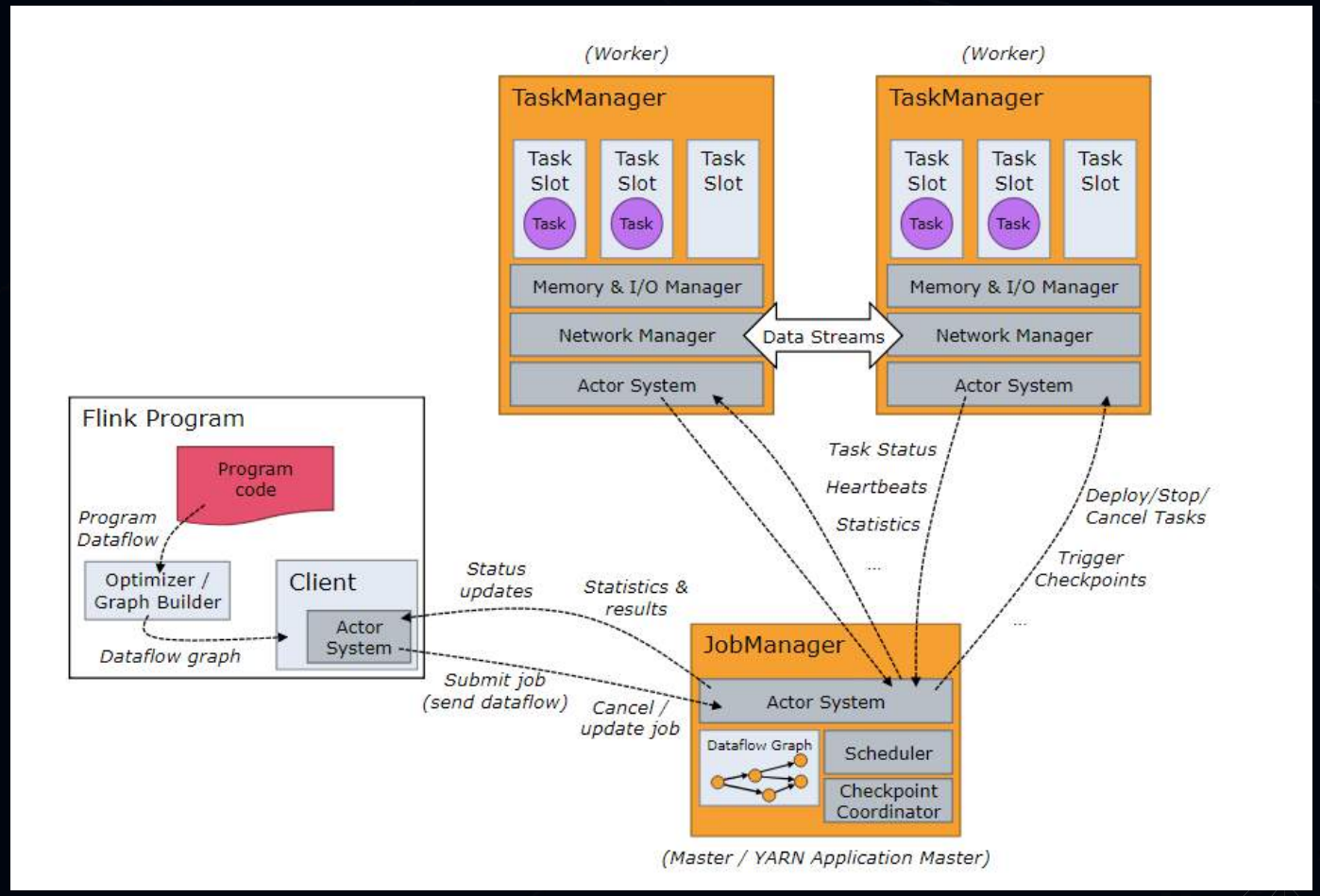
1. TalkingData流处理的背景与痛点

技术选型

注:对比为16年的特性	Flink	Heron
性能	优于storm，基于流	优于storm，基于流
语义	exactly-once/at-least-once	at-least-once
自主内存管理	是	否
Operator支持	较丰富	较丰富
SQL支持	是	否
Batch支持	Batch只是stream的特例	否
监控	不完善	较完善
使用者	阿里, 华为	Twitter

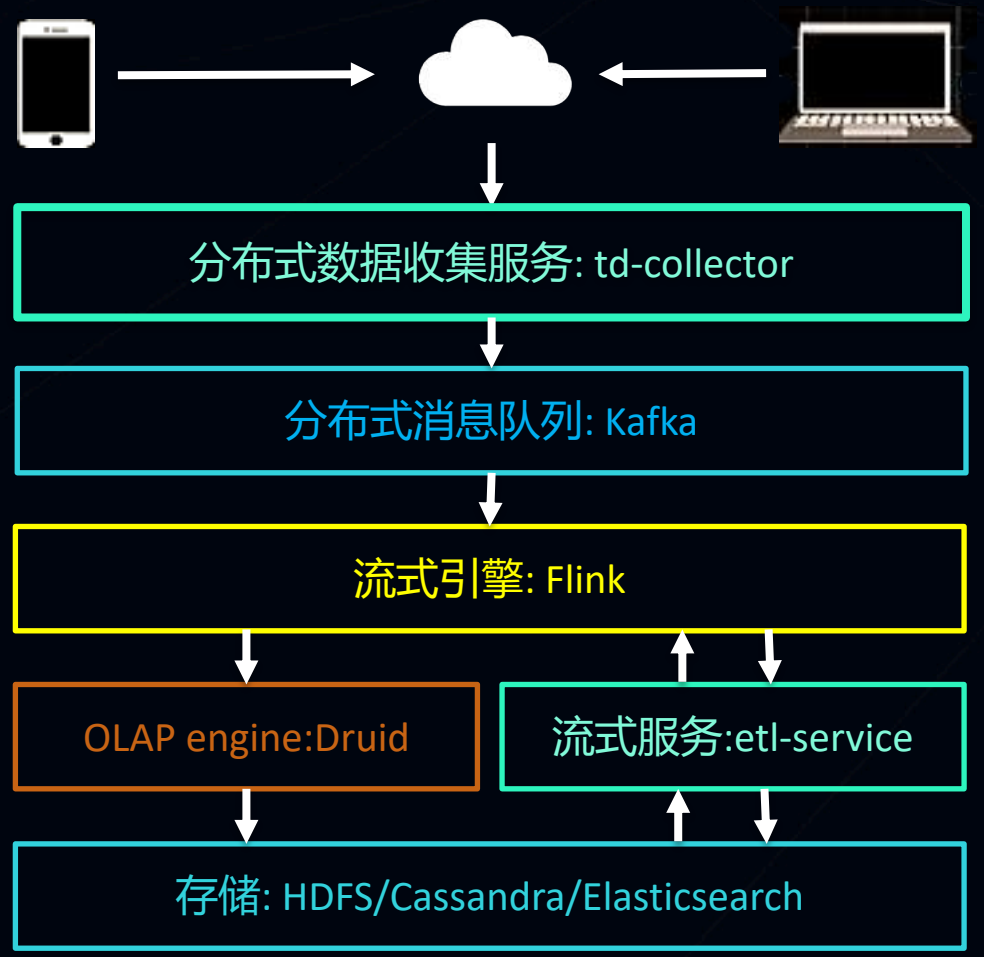
2. Flink在TalkingData SaaS分析中的演进路线

2.1 standalone cluster



2. Flink在TalkingData SaaS分析中的演进路线

2.1 standalone cluster



2. Flink在TalkingData SaaS分析中的演进路线

2.1 standalone cluster

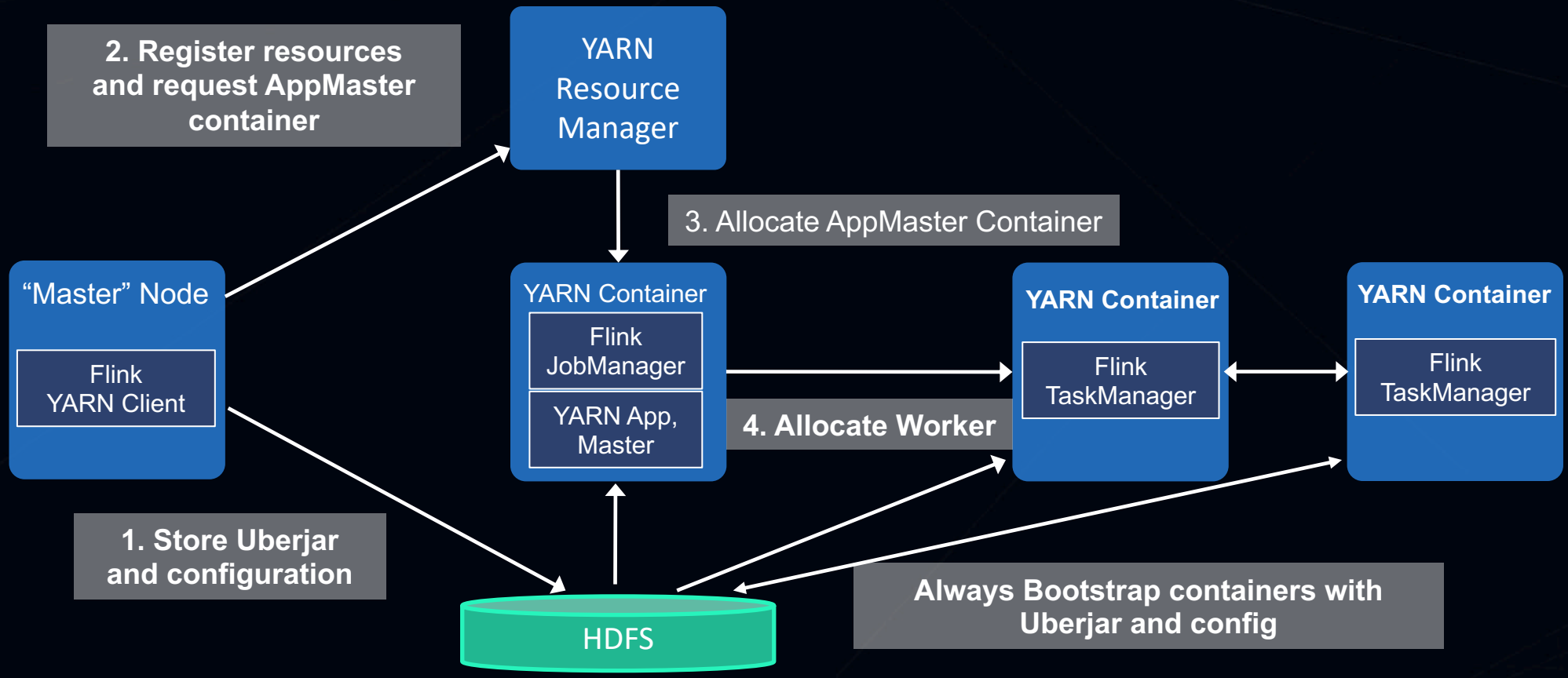
时间点	数据量	问题	部署规模
2017.4~ 2017.6	Game: 12亿 SDK packages/day, Game: 峰值 1.8w SDK packages/s	• 无	单集群，48 cores
2017.7~ 2017.8	Game: 18亿 SDK packages/day, Game: 峰值 3w SDK packages/s	• Job deployed不均匀，导致消费不均匀	单集群, 120cores
2017.9~ 2017.12	Game: 16亿SDK packages/day, Game: 峰值2.7w SDK package/s, App: 19亿 SDK packages/day, App: 峰值2.5w SDK packages/s	• 随着Job部署增多，相互干扰抢占资源； • 阻塞在requestBufferBlocking,导致整个Job假死	单集群，264cores
2018.1~ 2018.12	Game: 16亿 SDK packages/day, Game: 峰值2.8w SDK package/s, App: 28亿 SDK packages/day, App: 峰值 4.6w SDK packages/day, Other-jobs: 20w events/s	• 资源分配不均，job相互干扰	拆分成2个集群，456cores

2. Flink在TalkingData SaaS分析中的演进路线

2.2 flink on yarn

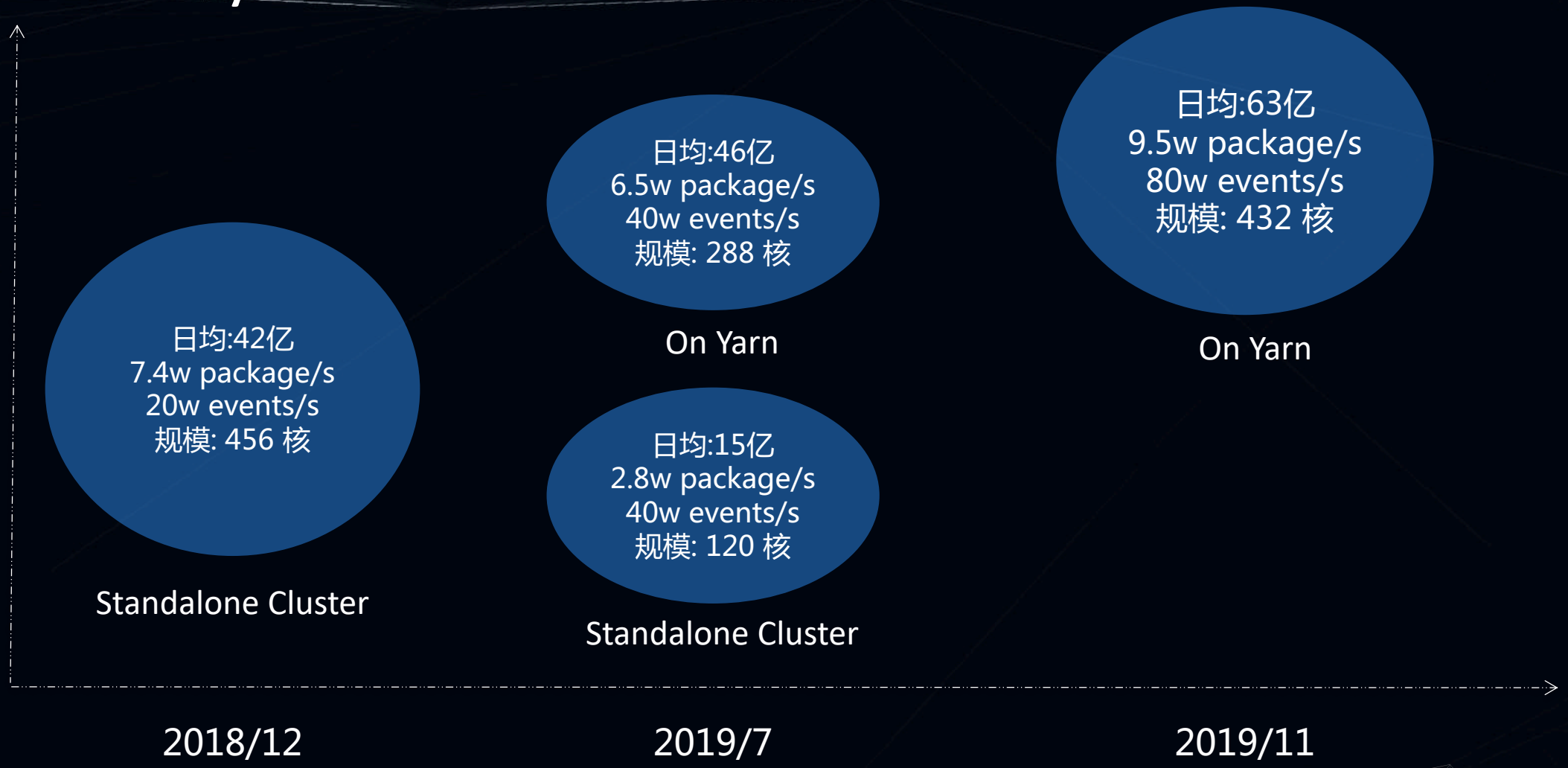
2. Flink在TalkingData SaaS分析中的演进路线

2.2 flink on yarn



2. Flink在TalkingData SaaS分析中的演进路线

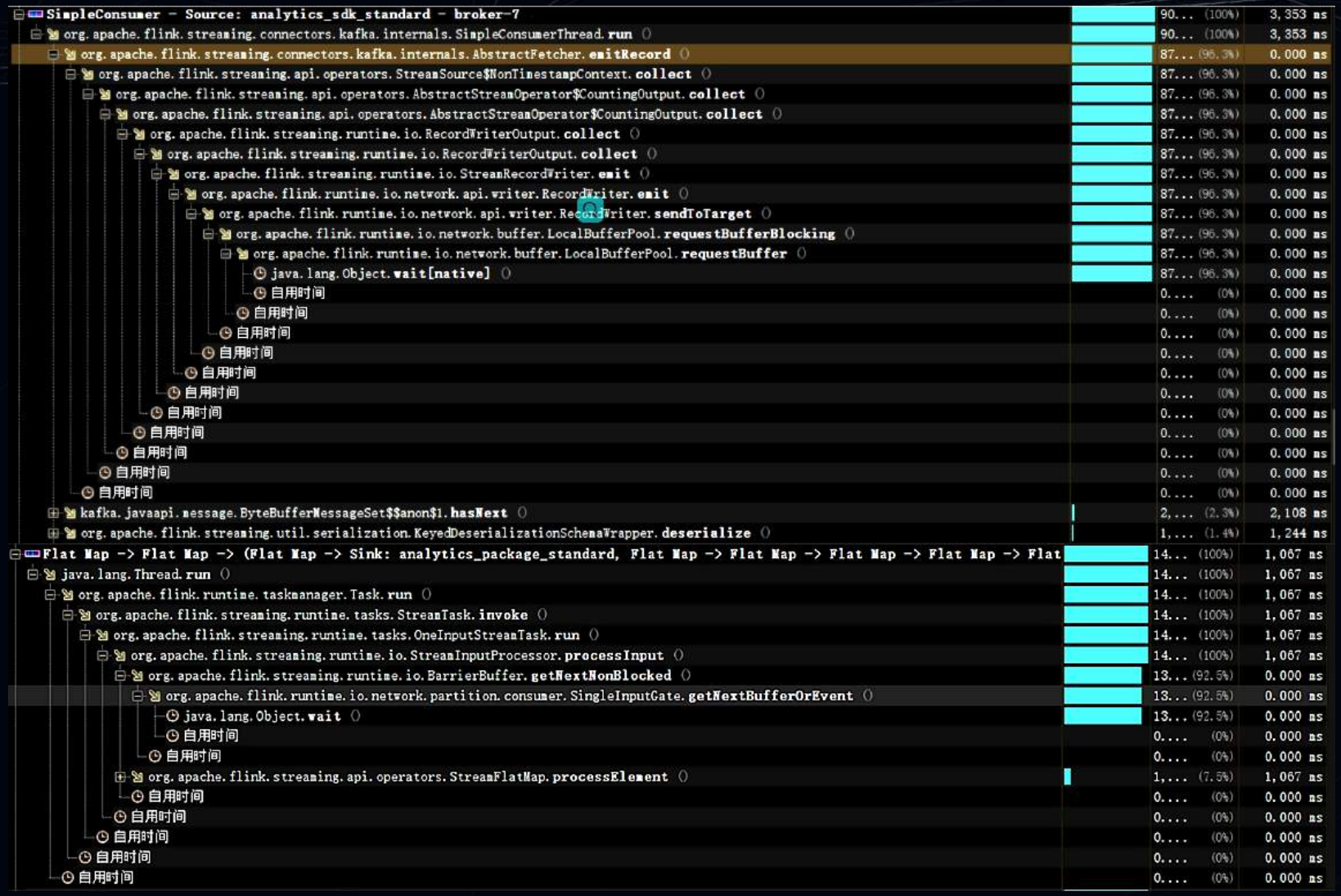
2.2 flink on yarn



3. 实践经验

3.1 Job阻塞与网络栈的优化

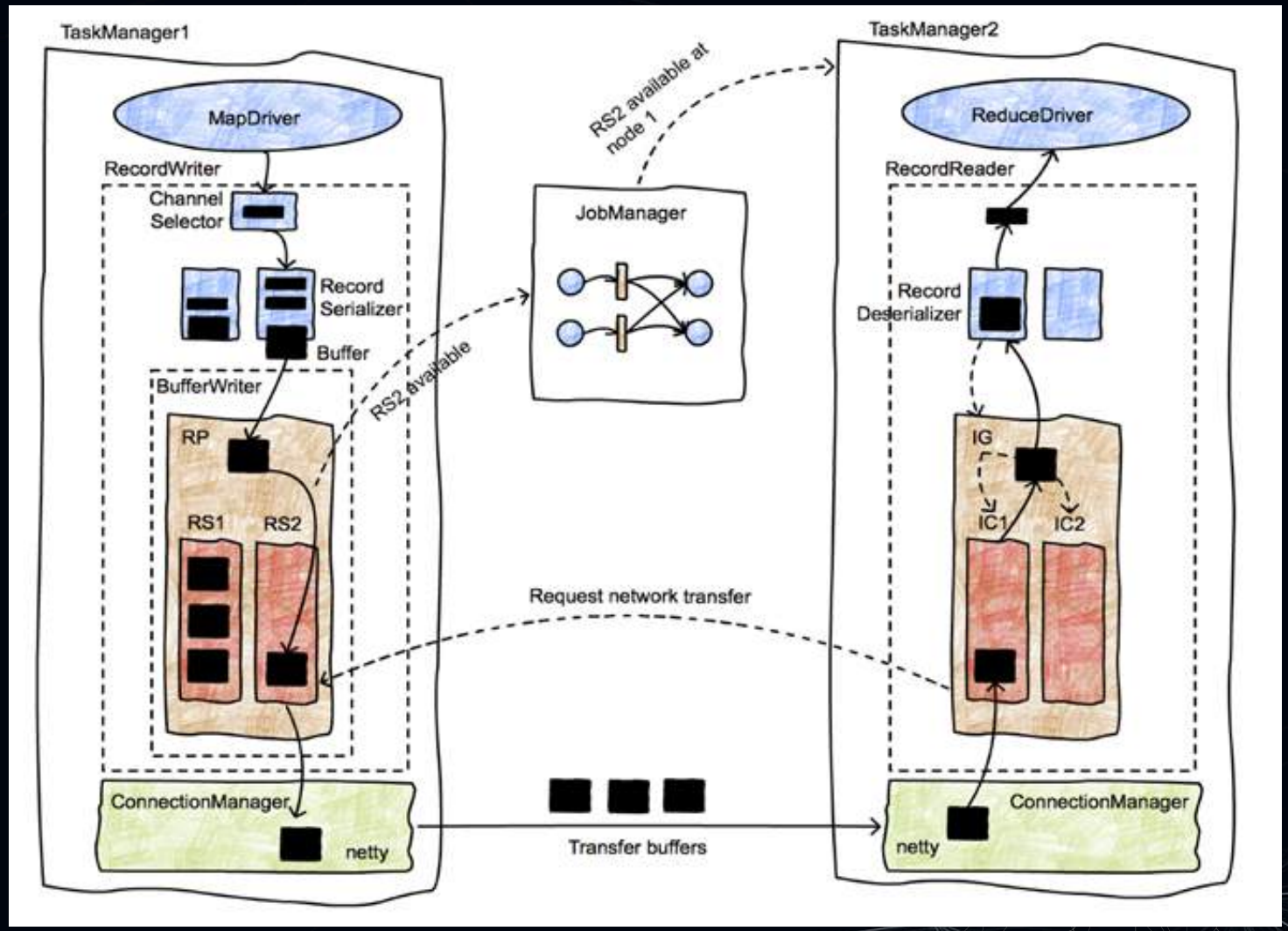
问题：Job Blocked,
吞吐量急剧下降



3. 实践经验

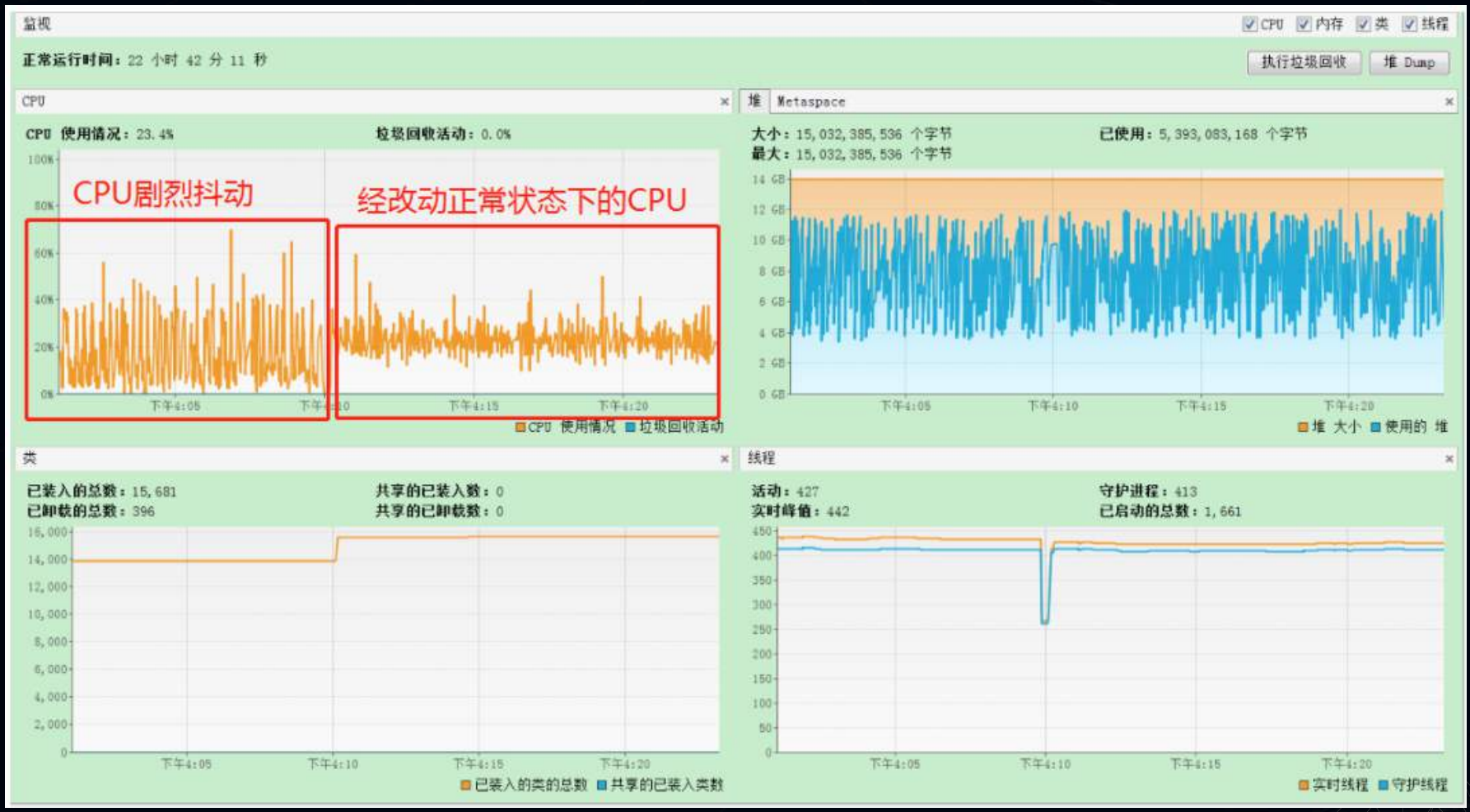
3.1 Job阻塞与网络栈的优化

- 分布在不同TM的operator之间的数据传输需要Buffer
- Buffer即MemorySegment，有效减少GC
- 数据传输与Buffer申请释放的过程
- BufferPool中Buffer不足时会阻塞



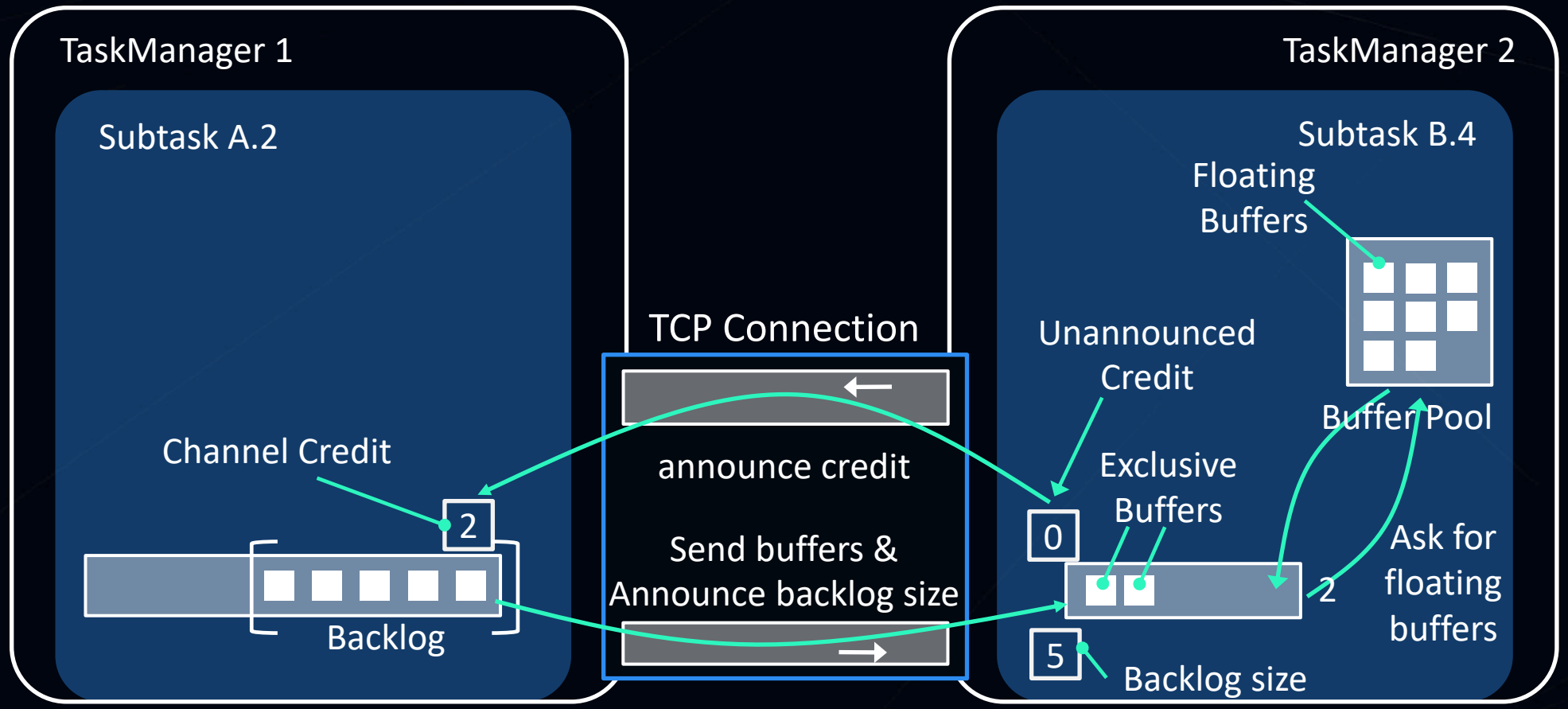
3. 实践经验

3.1 Job阻塞与网络栈的优化



3. 实践经验

3.1 Job阻塞与网络栈的优化

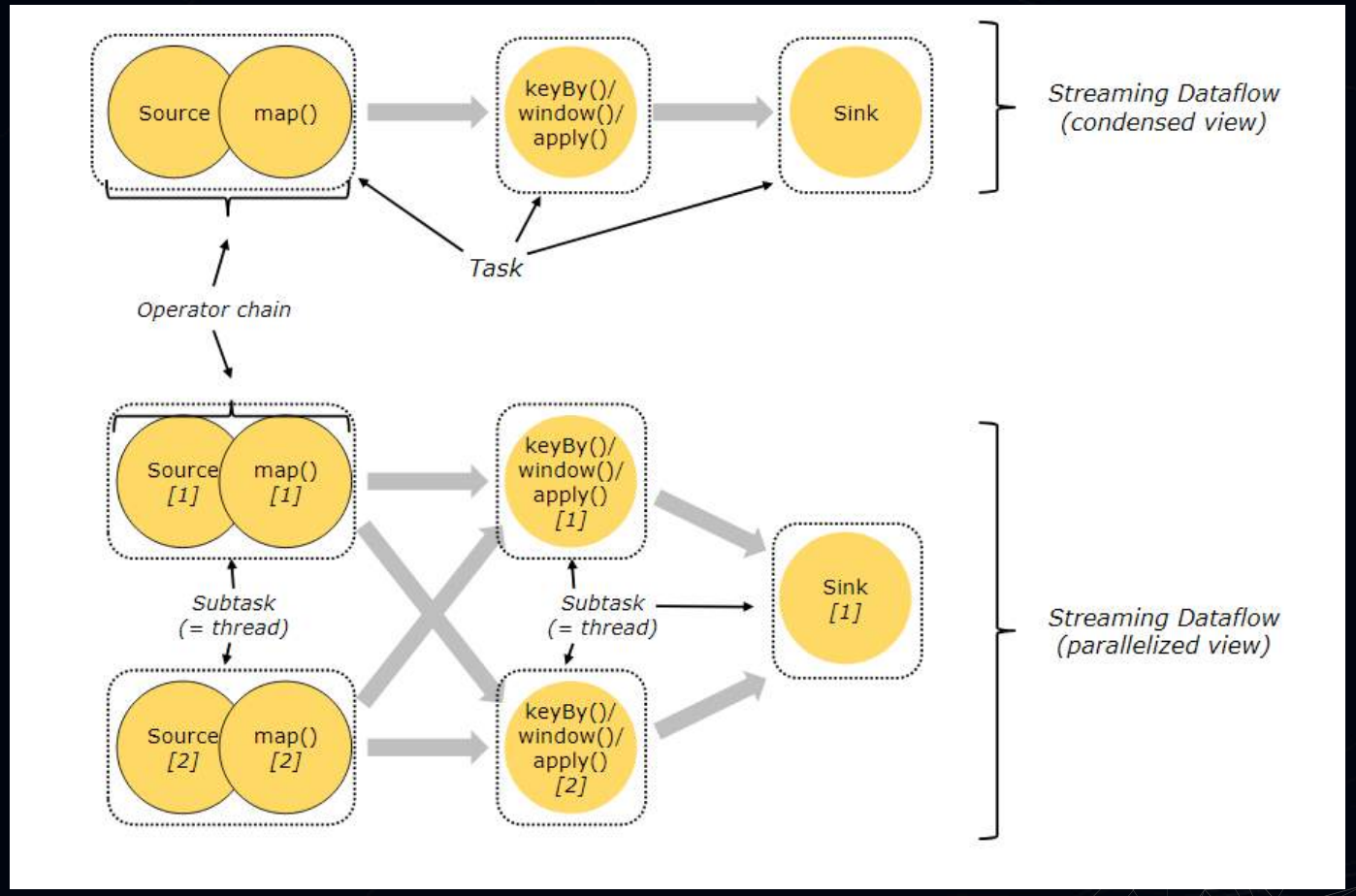


3. 实践经验

3.1 Job阻塞与网络栈的优化

- 尽可能的将operator chain在一起，避免网络传输和序列化反序列化的开销

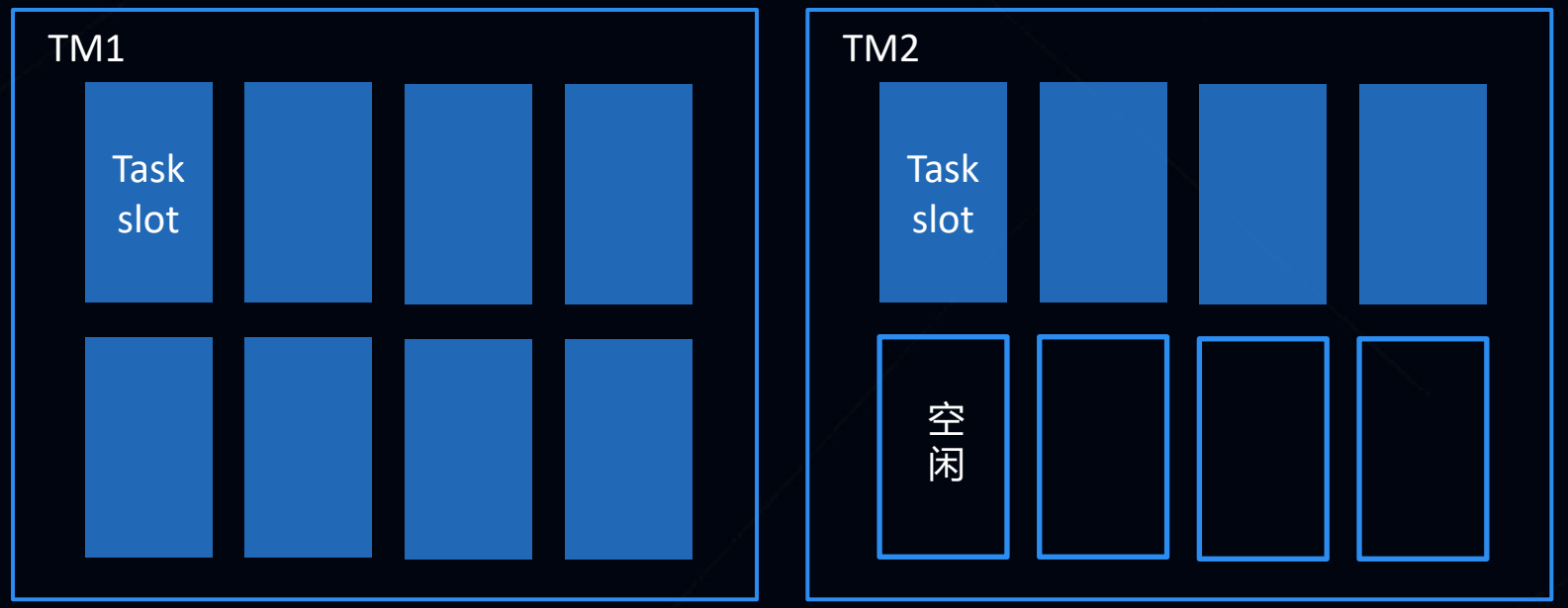
- 使用Flink 1.5之后的版本



3. 实践经验

3.2 资源的balance与isolation

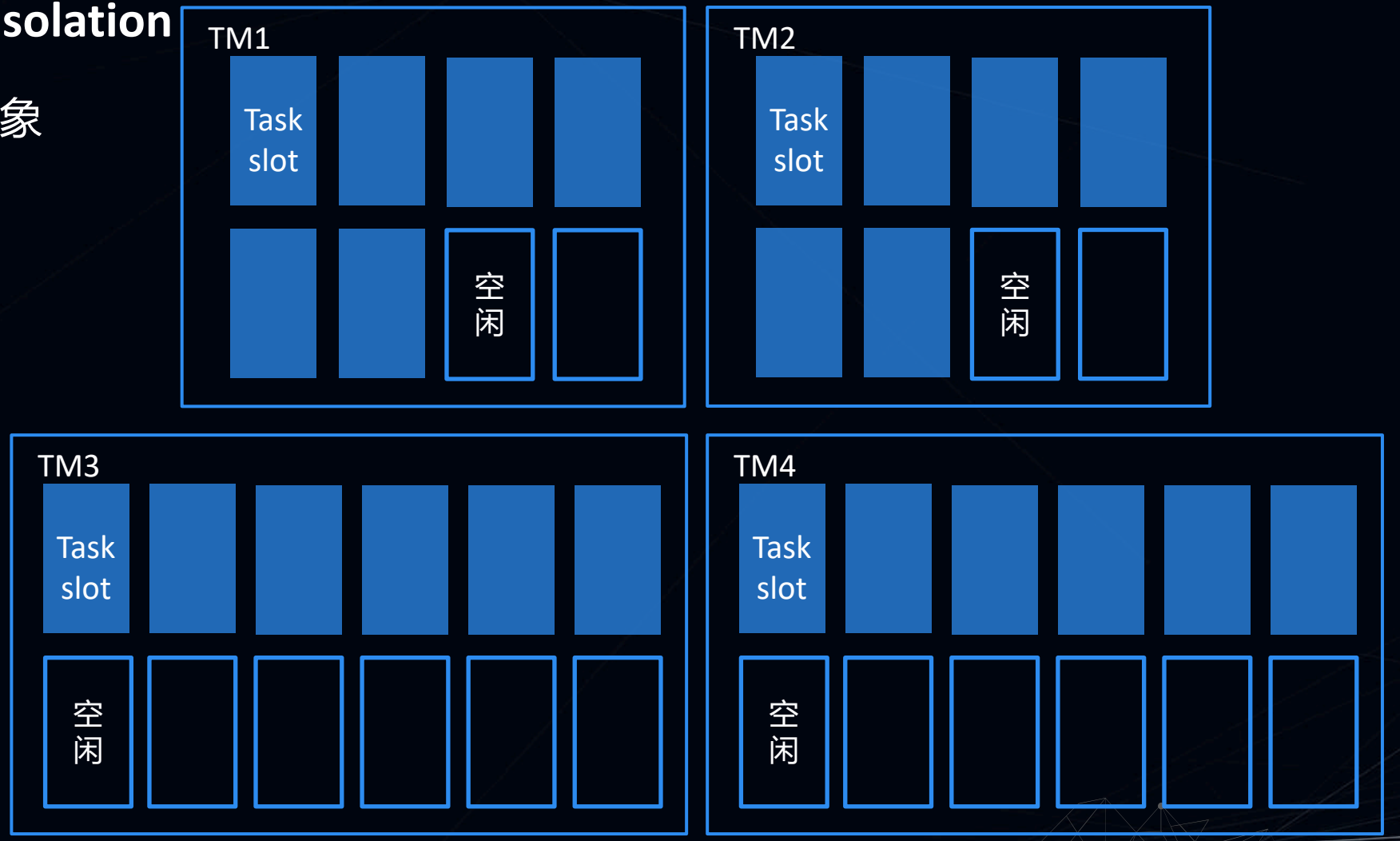
- Standalone cluster资源分配不均匀



3. 实践经验

3.2 资源的balance与isolation

- Standalone的不均匀现象



3. 实践经验

3.2 资源的balance与isolation

- 解决之道
  - 将TaskManager的粒度变小，即一台机器部署多个实例，每个实例持有的slot数较少；
  - 将大的业务job隔离到不同的集群上
  - Flink on Yarn
    - Container的拆解粒度不宜过小: 1core 2g vs 2core 4g

3. 实践经验

3.3 Serialize/Deserialize

- 遇到问题:
  - 序列化成了CPU抽样的热点，对性能损耗较大
  - 更泛化的JsonNode vs Pojos
- TypeInformation:
  - 类型信息知道的越多，Flink可以选取更好的序列化方式，并使Flink对内存的使用更高效；
  - TypeInformation内部封装了自己的序列化器，可通过createSerializer()获取，这样可以让用户不再操心序列化框架的使用(例如如何将他们自定义的类型注册到序列化框架中，尽管用户的定制化和注册可以提高性能)
- POJOs:
  - 类似于Java-Bean
  - 成员类型是Primitive, 或者BasicType

3. 实践经验

3.3 Serialize/Deserialize

```
//org.apache.flink.api.java.typeutils.PojoTypeInfo

@Override
@PublicEvolving
@SuppressWarnings("unchecked")
public TypeSerializer<T> createSerializer(ExecutionConfig config) {
   if (config.isForceKryoEnabled()) {
      return new KryoSerializer<>(getTypeClass(), config);
   }

   if (config.isForceAvroEnabled()) {
      return AvroUtils.getAvroUtils().createAvroSerializer(getTypeClass());
   }

   return createPojoSerializer(config);
}
```

3. 实践经验

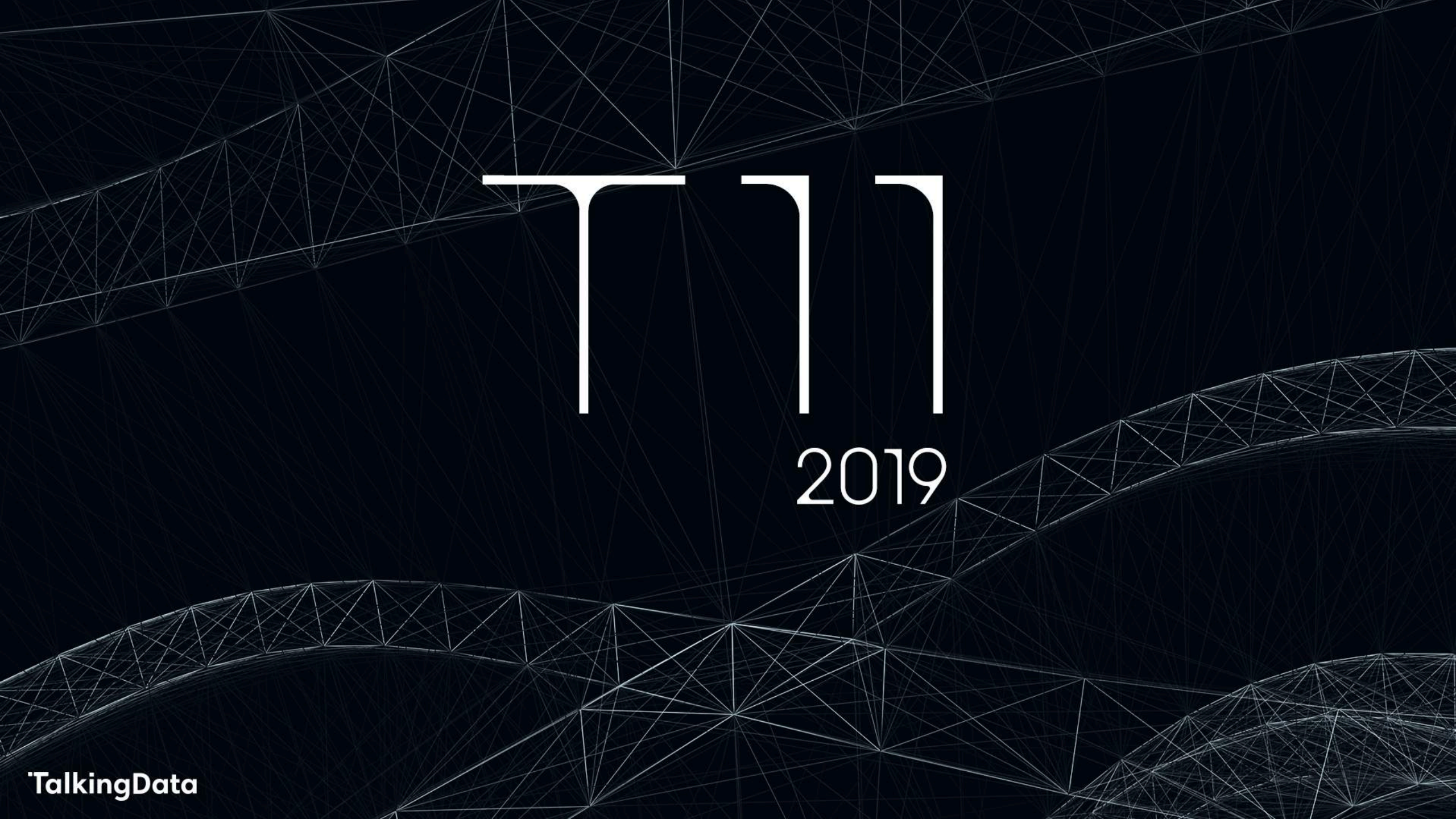
3.3 Serialize/Deserialize

- 显示调用returns方法，触发Flink的Type Hint:
- dataStream.flatMap(new MyOperator()).returns(MyClass.class)
- 注册subtypes：通过StreamExecutionEnvironment或ExecutionEnvironment的实例的registerType(clazz)方法注册我们的数据类及其子类、其字段的类型。如果Flink对类型知道的越多，性能会更好；
- 自己定制序列化器

4. 总结与展望

Flink已经稳定支持TalkingData分析线日均63亿package，峰值: 9.5w package/s 80w events/s的体量

未来可以探索将更复杂的业务迁移到Flink上，甚至Batch Job


2019
TalkingData