Pluggable Shuffle Service and Unaligned Checkpoint

王治江
阿里巴巴计算平台高级技术专家
Apache Flink Committer

FLINK FORWARD # ASIA

实时即未来 # Real-time Is The Future

FLINK FORWARD

Contents
目录

01 Shuffle Service 插件化架构

The Architecture of Pluggable Shuffle Service

02 Unaligned Checkpoint 机制

The Mechanism of Unaligned Checkpoint

Shuffle Service 插件化架构

The Architecture of Pluggable Shuffle Service

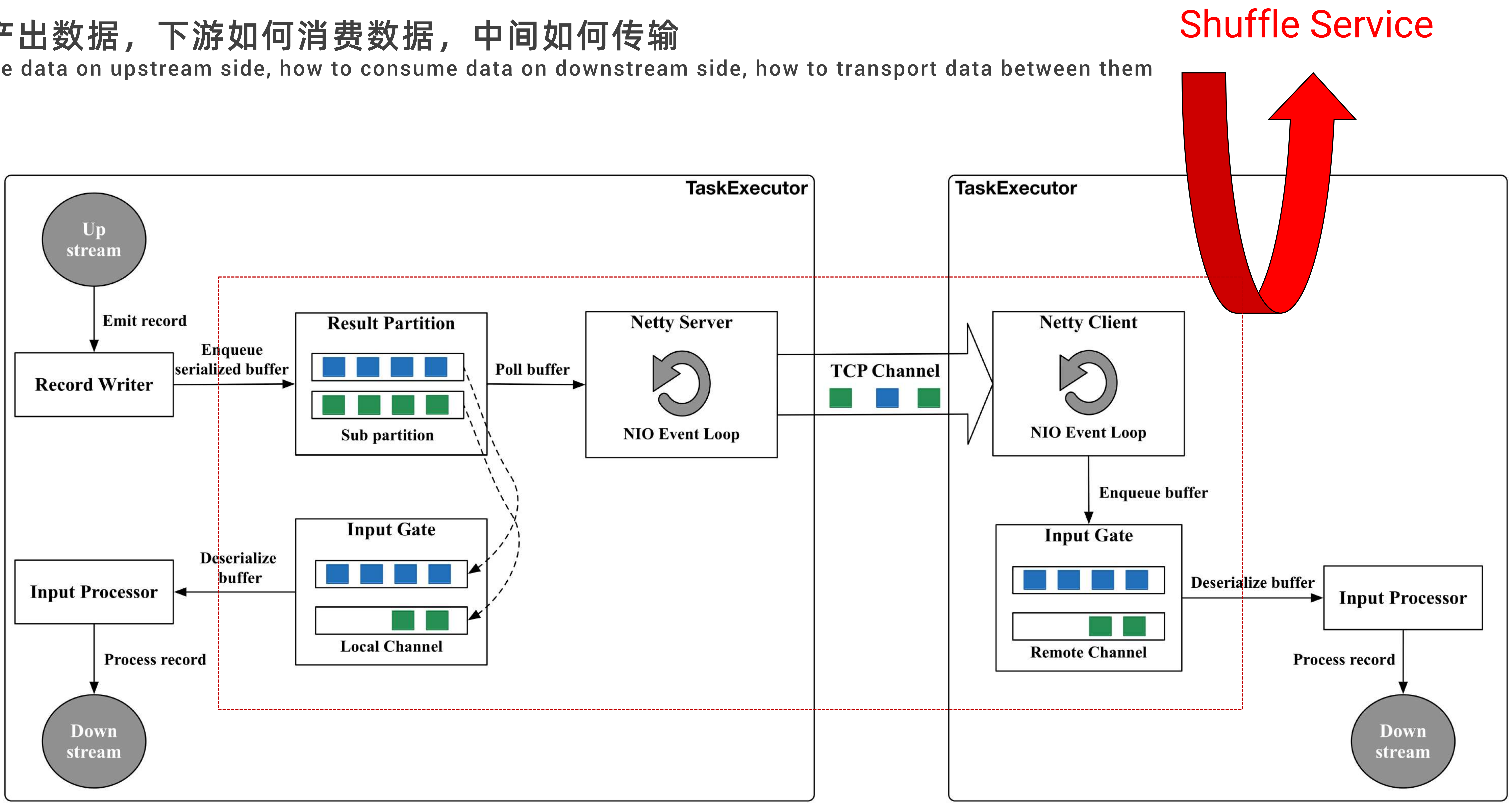
01

数据生产消费流程

The Process of Data Production and Consumption

上游如何产出数据，下游如何消费数据，中间如何传输

How to produce data on upstream side, how to consume data on downstream side, how to transport data between them



问题

The Problems

Partition / Task / Task Executor 生命周期和资源管理

Life cycles among partition / task / Task Executor and resource management

- Task 结束 -> Task Executor 被回收 -> Partition 无法传输 -> 作业失败重启
  Task finishes -> Task Executor is released -> Partition fails to transport -> Job fails and restarts
- Task 结束 -> Task Executor 服务数据传输 -> 浪费集群资源
  Task finishes -> Task Executor serves partition shuffle -> Waste cluster resources
- Partition 传输结束 -> Partition 释放 -> 下游消费失败 -> 重启上游产出 partition
  Partition transport finishes -> partition is released -> downstream fails during consuming-> restart upstream to re-produce partition

架构上很难扩展满足不同的需求 (batch job)

Hard to extend new implementations to satisfy other requirements based on current architecture

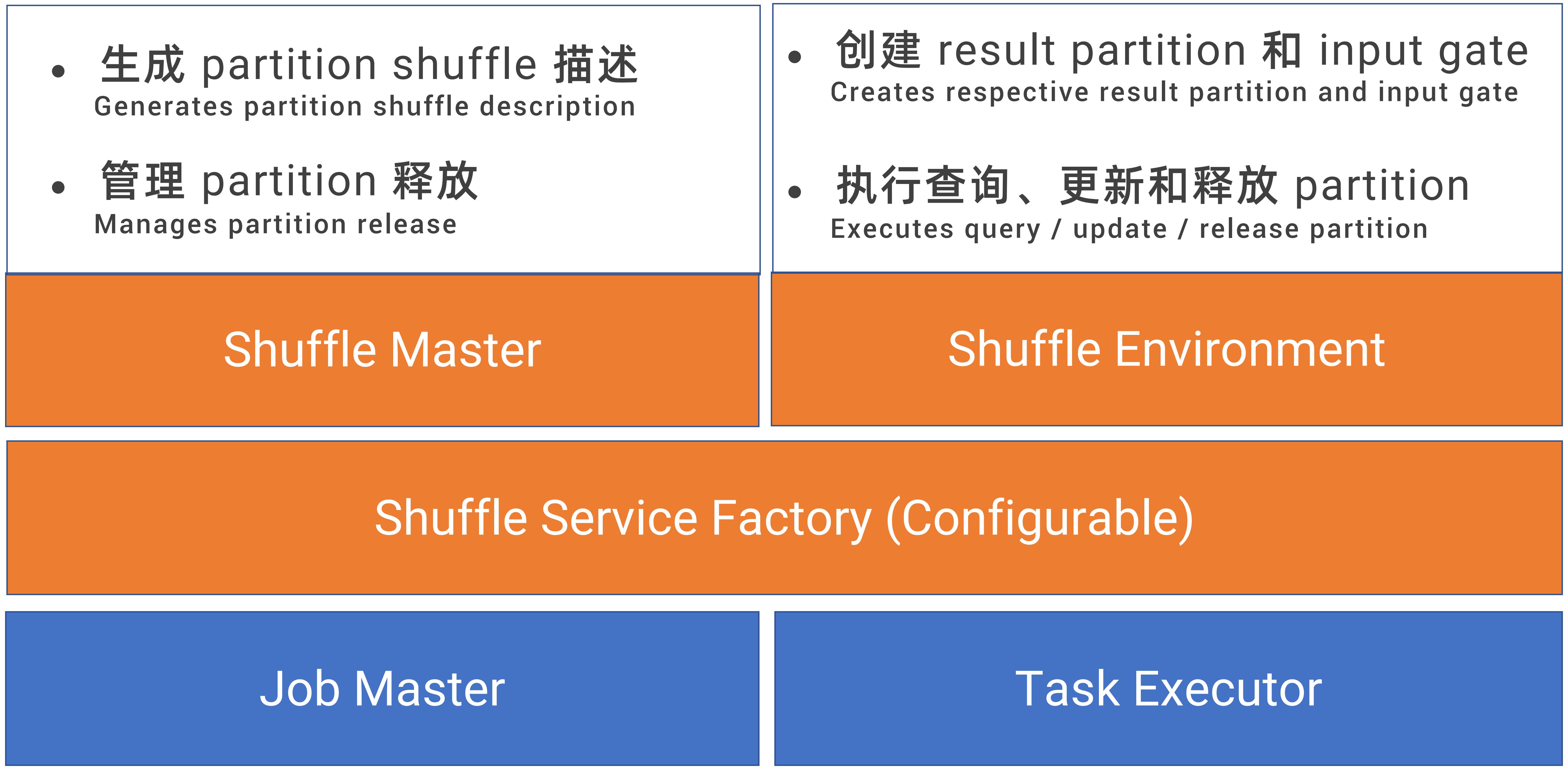
- Sort & merge partition
- 计算存储分离架构
  The architecture of separating computation and storage
- **External shuffle service**

解决方案：Shuffle Service 插件化架构

The Solution : Pluggable Shuffle Service Architecture

基本思路（FLIP-31）

Basic ideas



- 生成 partition shuffle 描述
  Generates partition shuffle description
- 管理 partition 释放
  Manages partition release

- 创建 result partition 和 input gate
  Creates respective result partition and input gate
- 执行查询、更新和释放 partition
  Executes query / update / release partition

Shuffle Master

Shuffle Environment

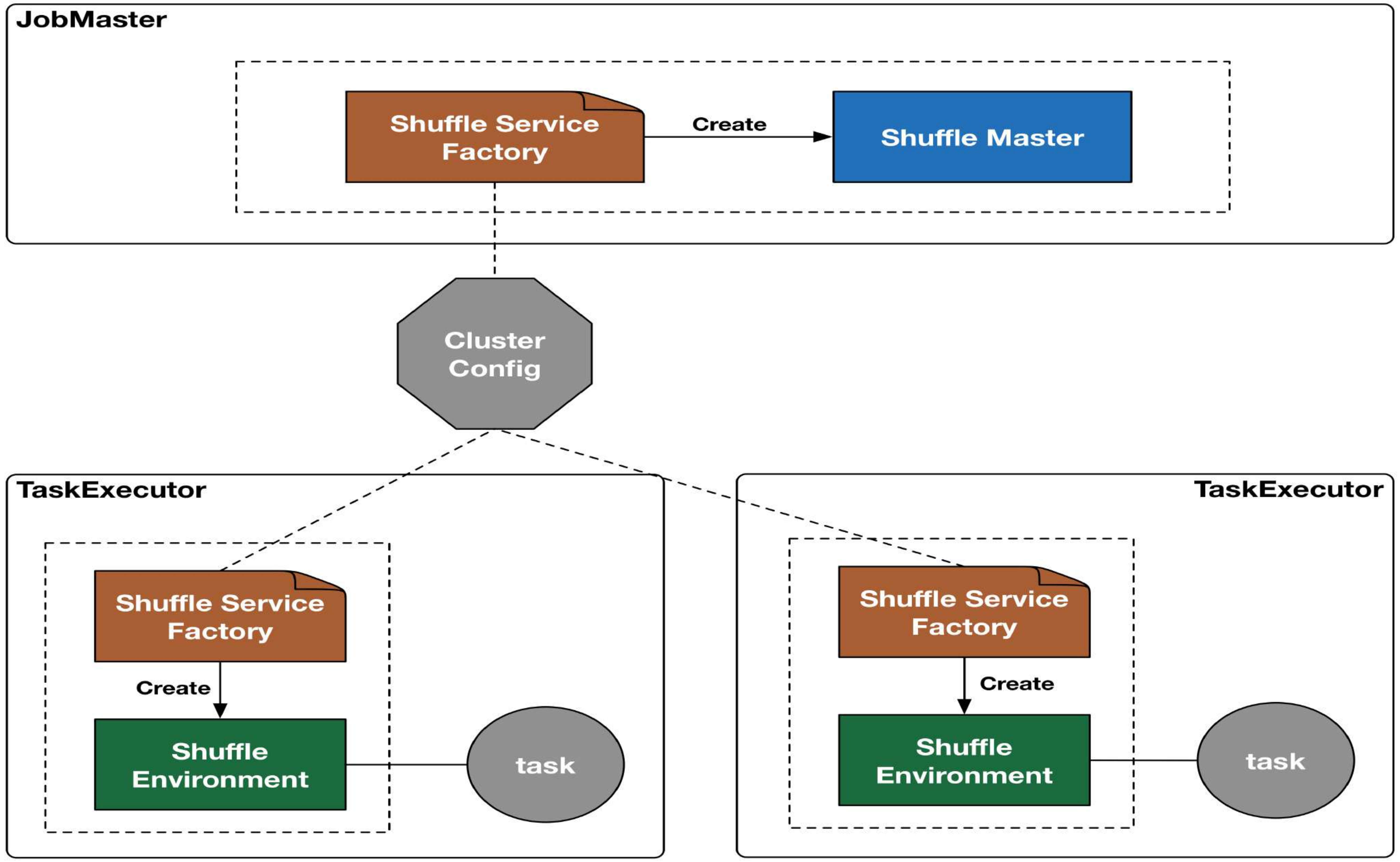
Shuffle Service Factory (Configurable)

Job Master

Task Executor

Shuffle Service 整体架构

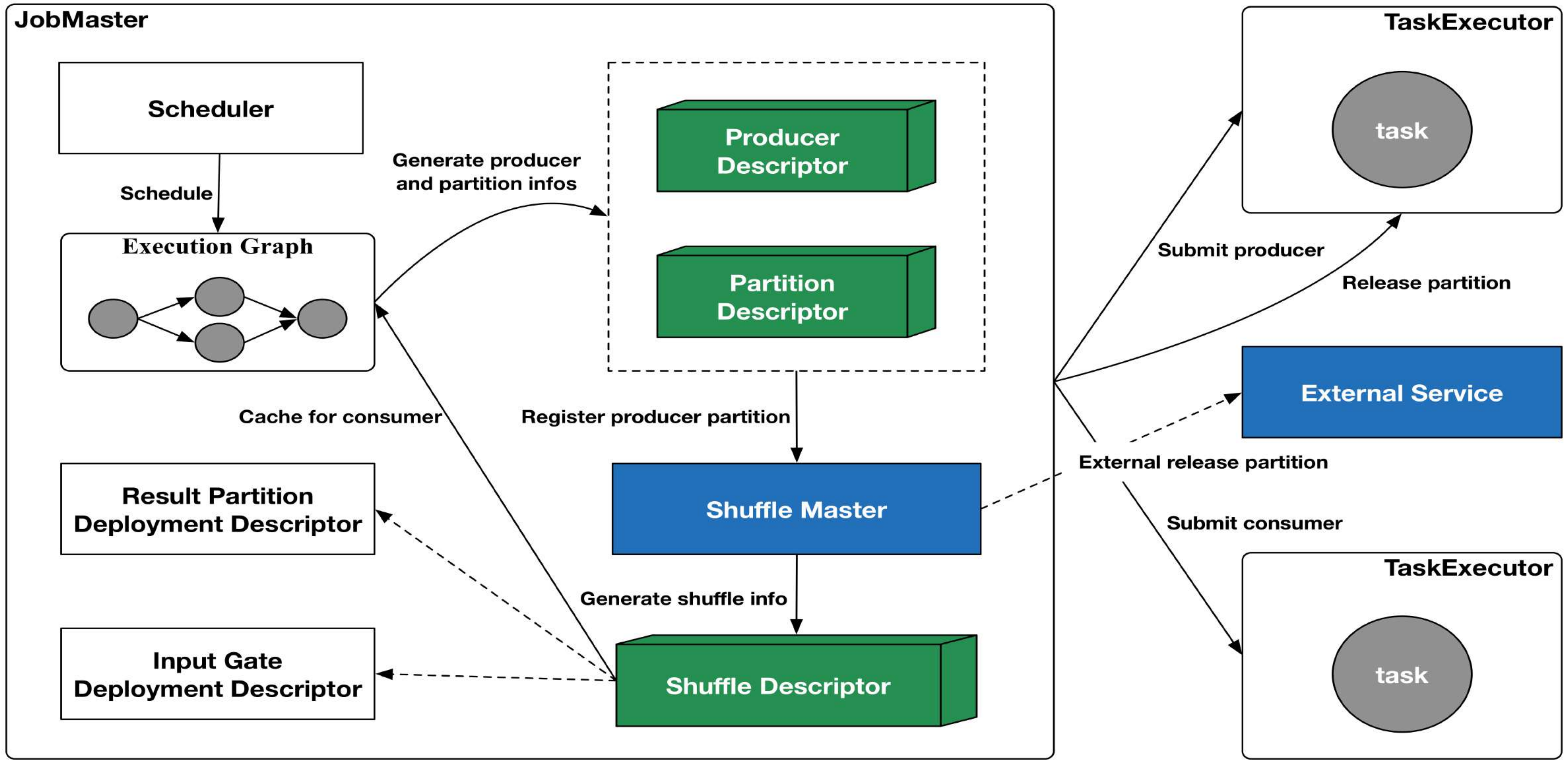
The Architecture of Shuffle Service



```
interface ShuffleServiceFactory<SD extends ShuffleDescriptor, P extends ResultPartitionWriter, G extends InputGate>

{
    ShuffleMaster<SD>  createShuffleMaster(Configuration configuration);

    ShuffleEnvironment<P, G>  createShuffleEnvironment(ShuffleEnvironmentContext shuffleEnvironmentContext);

}
```

Shuffle Master 详细设计

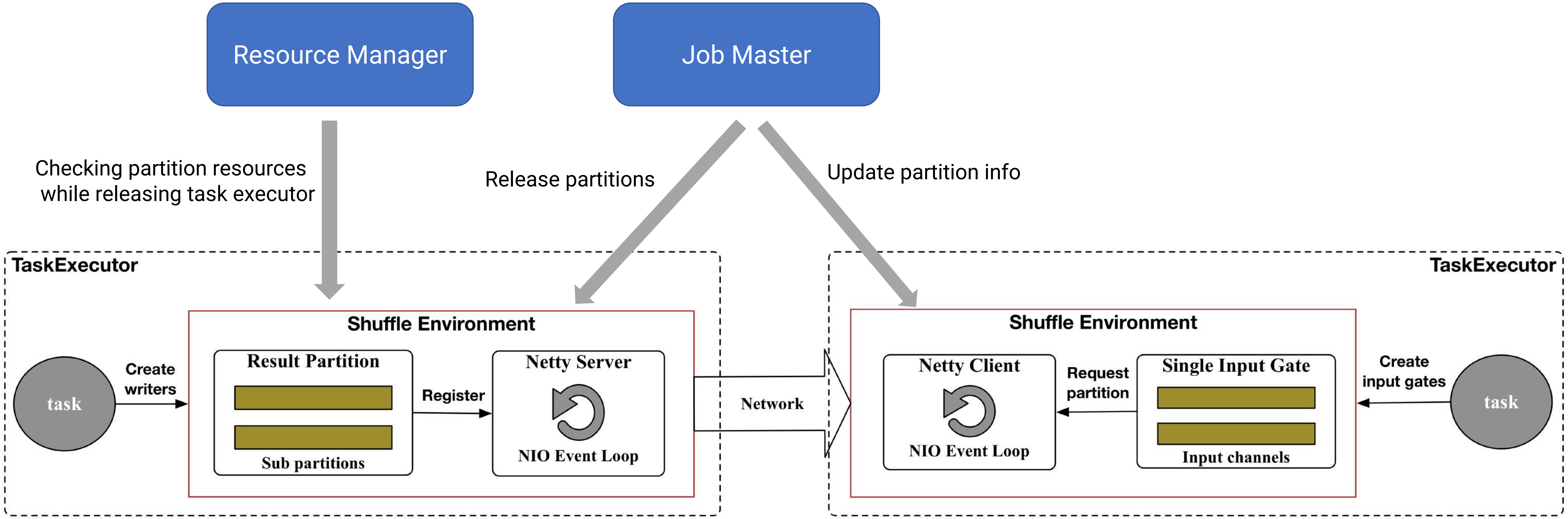
The Detail Design for Shuffle Master



```
interface ShuffleMaster<T extends ShuffleDescriptor>
{
        CompletableFuture<T> registerPartitionWithProducer(PartitionDescriptor partitionDescriptor, ProducerDescriptor producerDescriptor);

         void releasePartitionExternally(ShuffleDescriptor shuffleDescriptor);
}
```

Shuffle Environment 详细设计

The Detail Design for Shuffle Environment



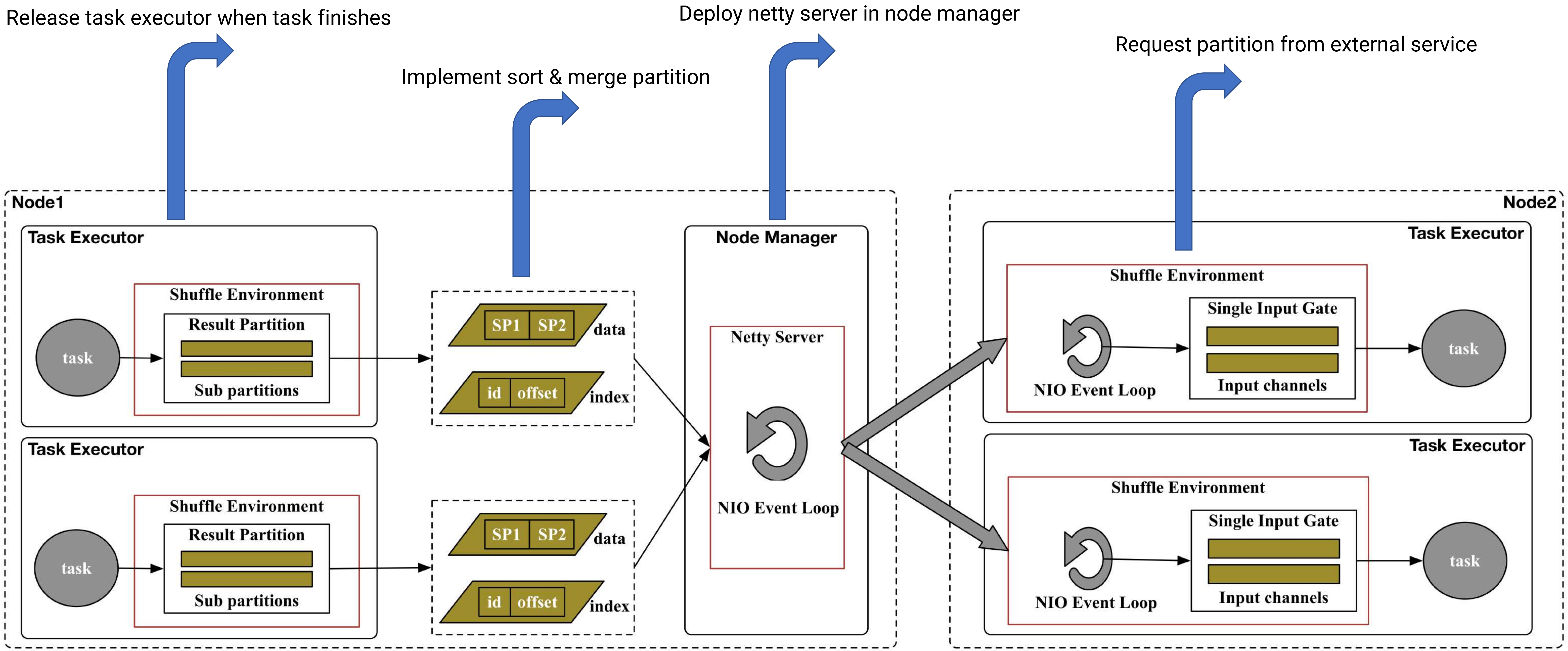
```
interface ShuffleEnvironment<P extends ResultPartitionWriter, G extends InputGate> extends AutoCloseable
{
    Collection<P> createResultPartitionWriters();

    void releasePartitionsLocally(Collection<ResultPartitionID> partitionIds);

    Collection<ResultPartitionID> getPartitionsOccupyingLocalResources();

    Collection<G> createInputGates();

    boolean updatePartitionInfo(ExecutionAttemptID consumerID,  PartitionInfo partitionInfo);
}
```

Blink 实现 Yarn Shuffle Service

External Yarn Shuffle Service in Alibaba Blink Branch

应用在搜索离线场景

The external shuffle service is used in the search offline scenarios currently



总结

The Conclusions

已完成工作 (release-1.9)

Work already done

- **定义抽象了** shuffle service **相关的接口 API**

  Defines and abstracts the related interface APIs for shuffle service architecture

- **基于新的架构重构了** task executor **内部的实现**

  Refactors the relevant implementations of existing internal shuffle service in task executor based on the new architecture

后续工作 (release-1.*)

Future work

- **扩展支持** job **或** edge **级别的** shuffle service **配置**

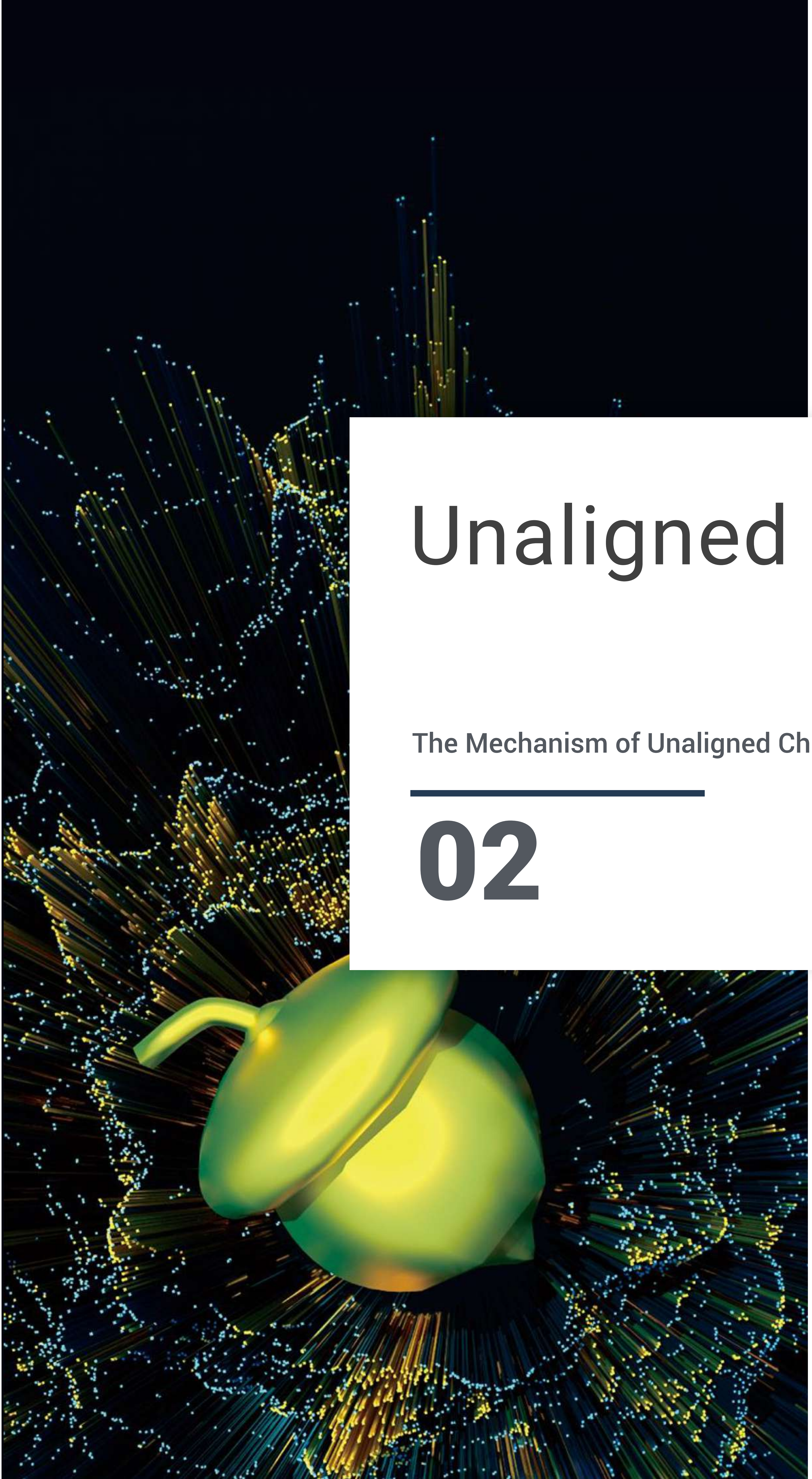
  Extends to support the config of shuffle service for job or edge level

- ResultPartitionWriter & InputGate **接口读写** record **代替** buffer

  Supports to read & write record instead of buffer in the interfaces of ResultPartitionWriter & InputGate

- **贡献** external shuffle service **实现** (YARN / K8S / RDMA)

  Contributes other external shuffle service implementations, such as YARN/K8S/RDMA



Unaligned Checkpoint 机制

The Mechanism of Unaligned Checkpoint

02

Flink Checkpoint 机制

The Mechanism of Checkpoint in Flink

基于 Chandy – Lamport 算法的分布式一致性快照

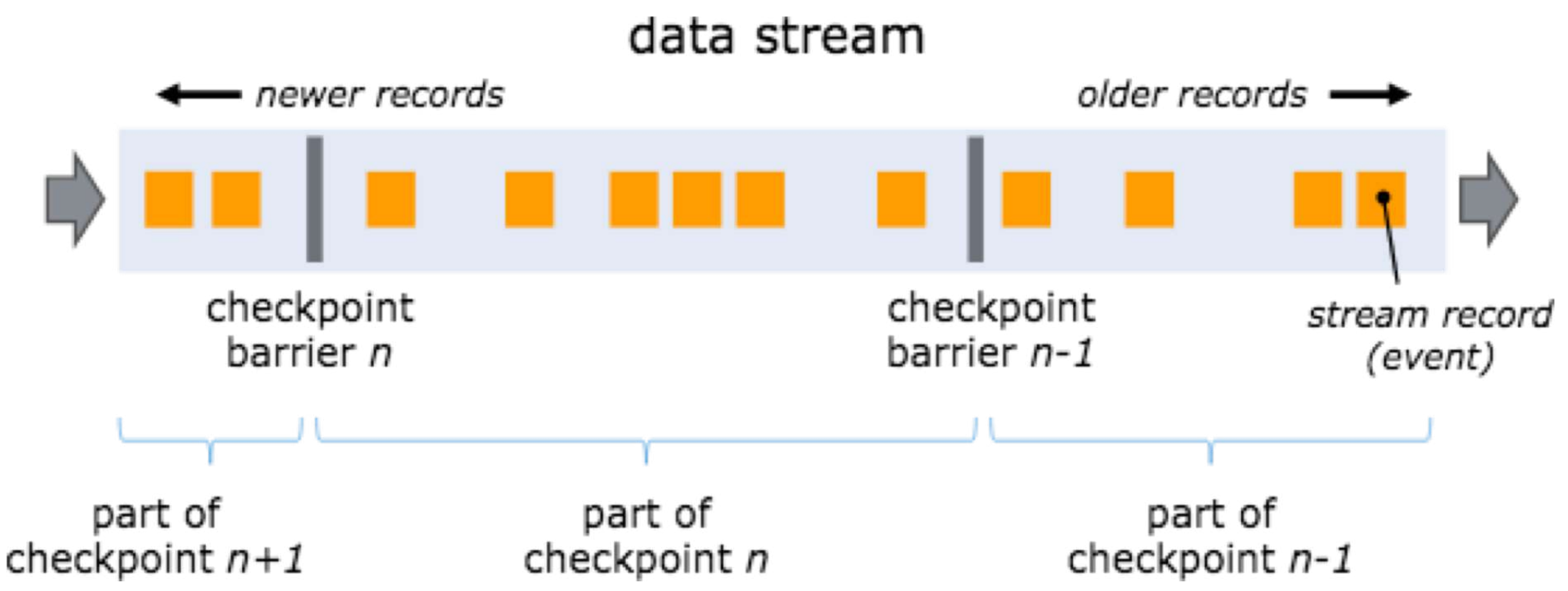
Distributed consistent snapshot based on Chandy – Lamport algorithm

定期从 source 向数据流插入 barrier 流经 sink，不跨越数据，轻量化

Barrier is injected into the data stream from source and flows with the records until sink. It never overtakes records, light-weight

Barrier 前面的数据处理后对应的状态在读到 barrier 时快照

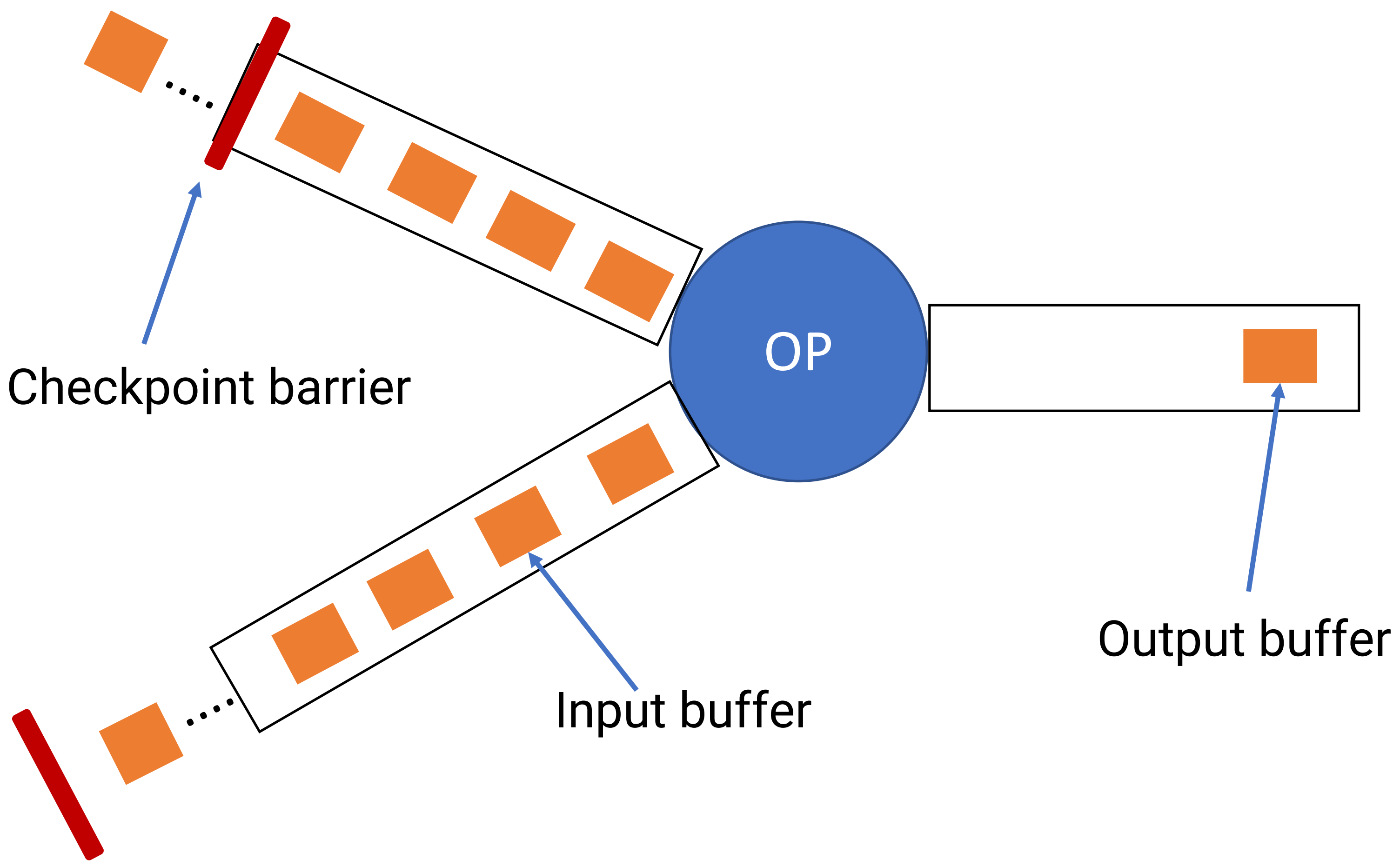
Records before barrier are processed and reflected on the respective checkpoint state via snapshotting



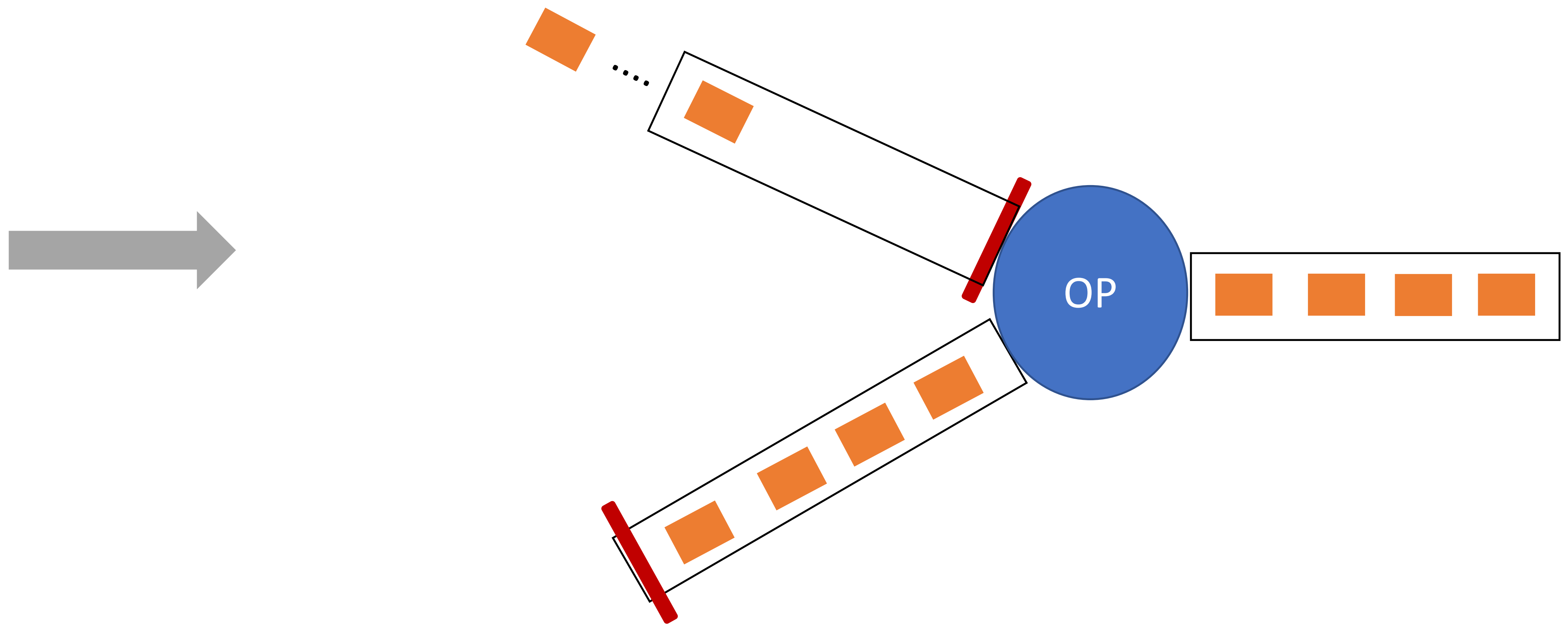
Aligned Checkpoint 原理（Exactly-once）

The Mechanism of Aligned Checkpoint for exactly-once

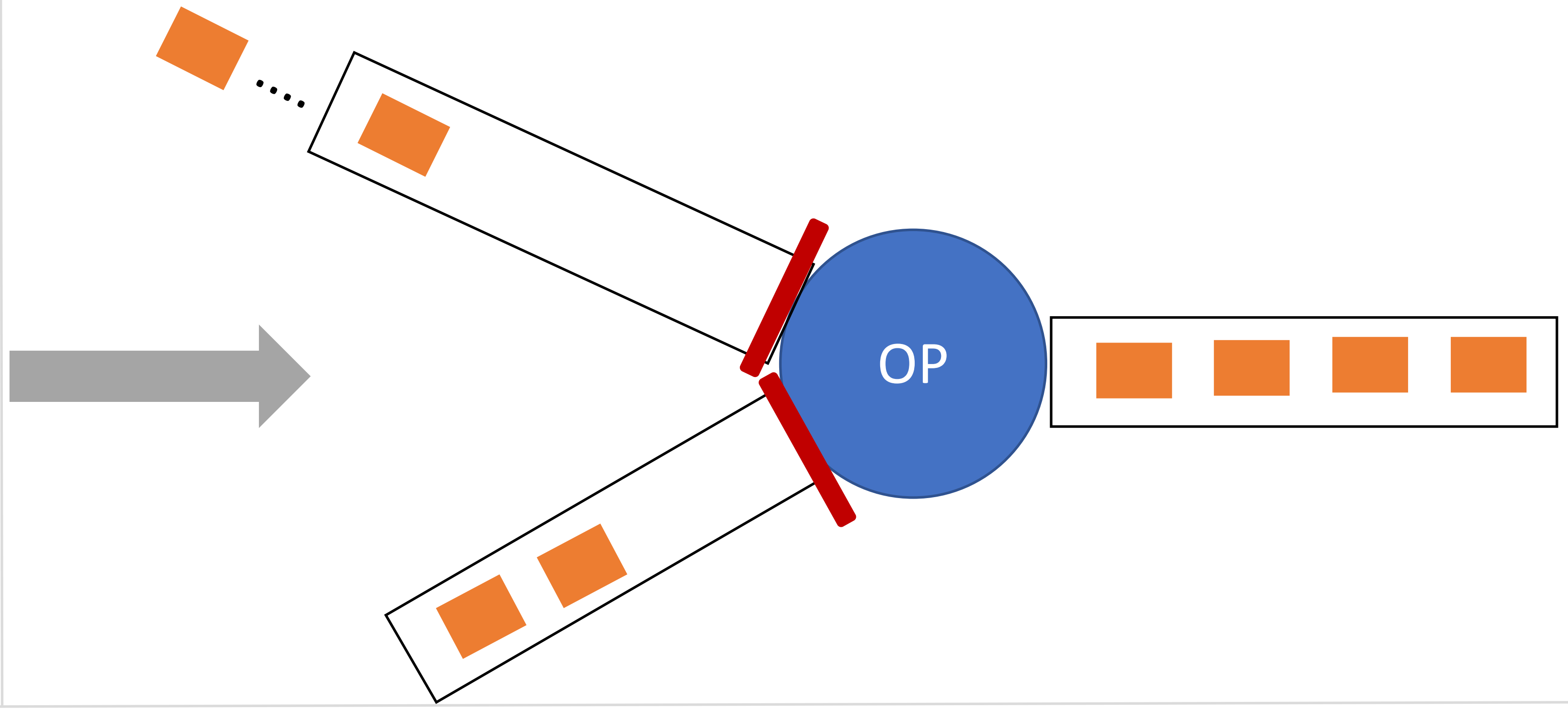
- Channel-1：收到 barrier 插入队尾



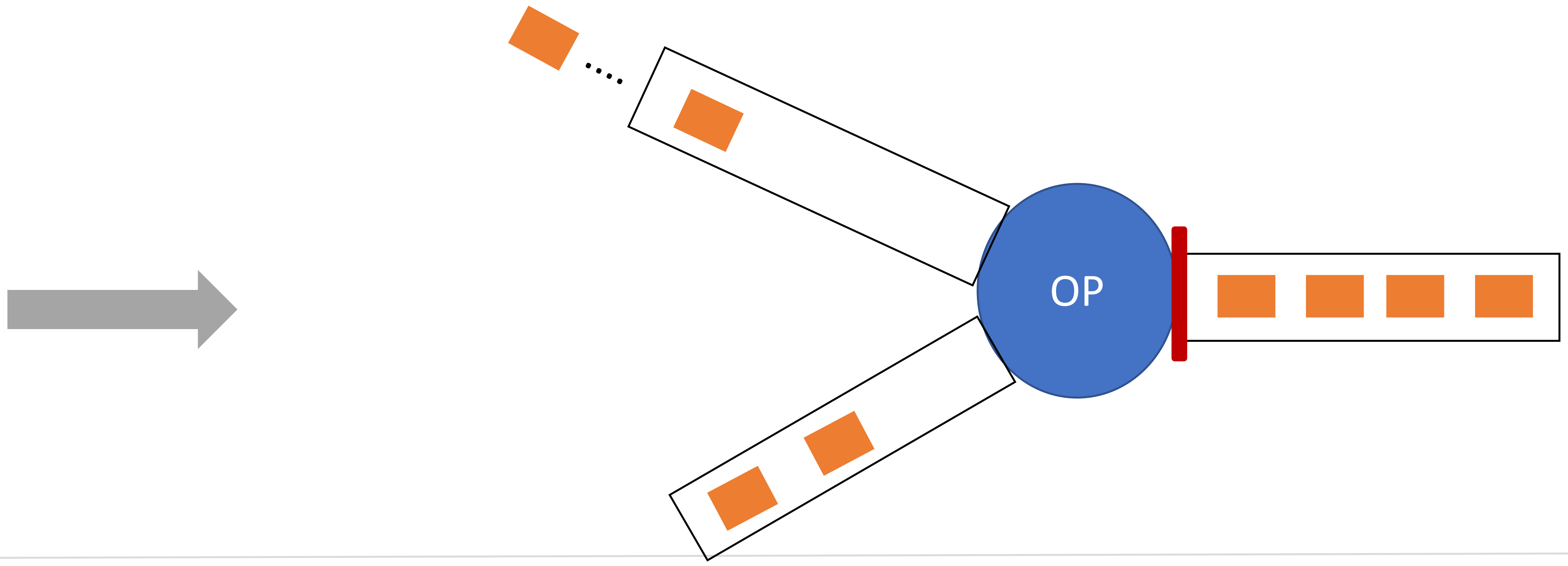
- Channel-1：Barrier 前面的 buffer 被处理，后面的 buffer 被 block
- Channel-2：收到相同 barrier 插入队尾



- Channel-2： barrier 前的 buffer 处理完
- 所有 channel barrier 收齐，触发 checkpoint

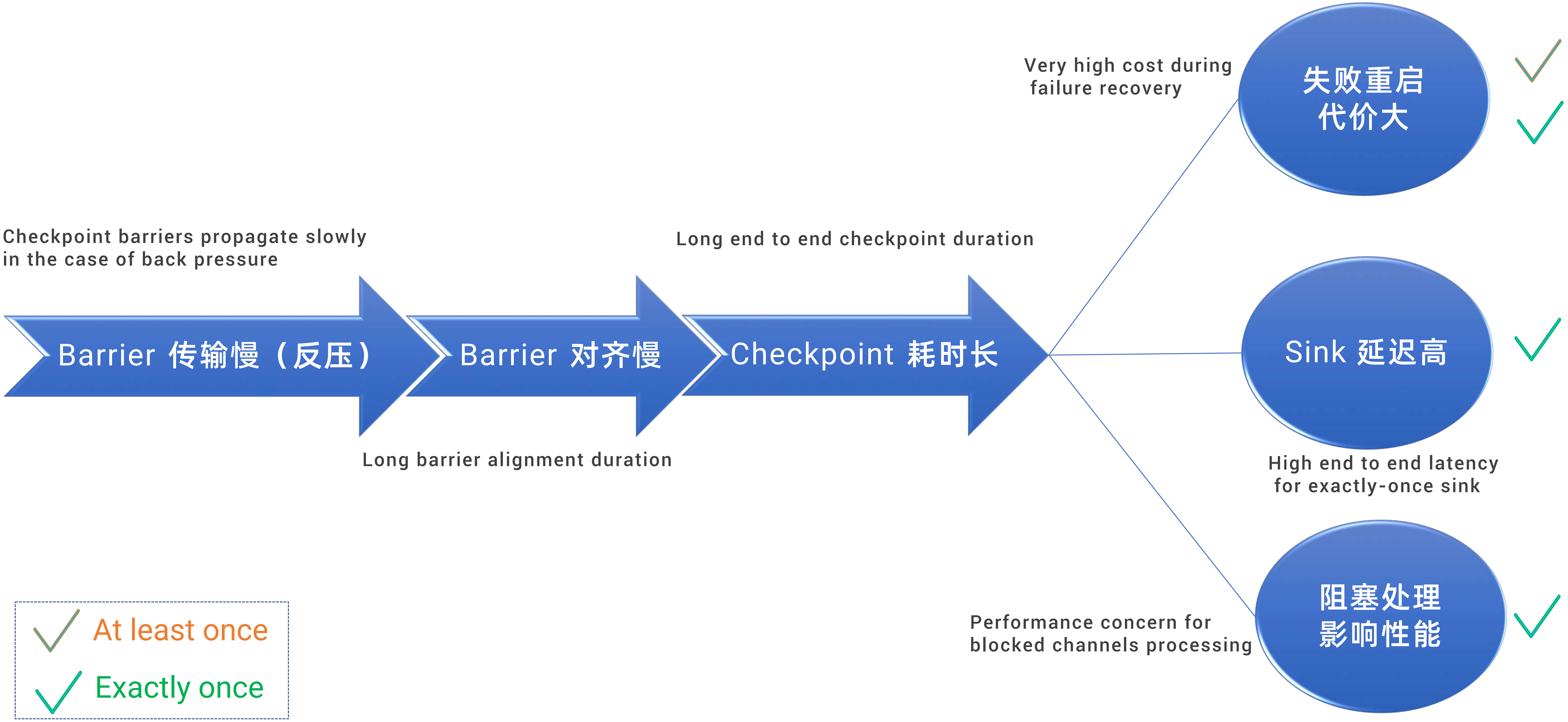


- 广播 barrier 到输出队列中
- Operator 快照 state



Aligned Checkpoint 问题

The Problems of Aligned Checkpoint

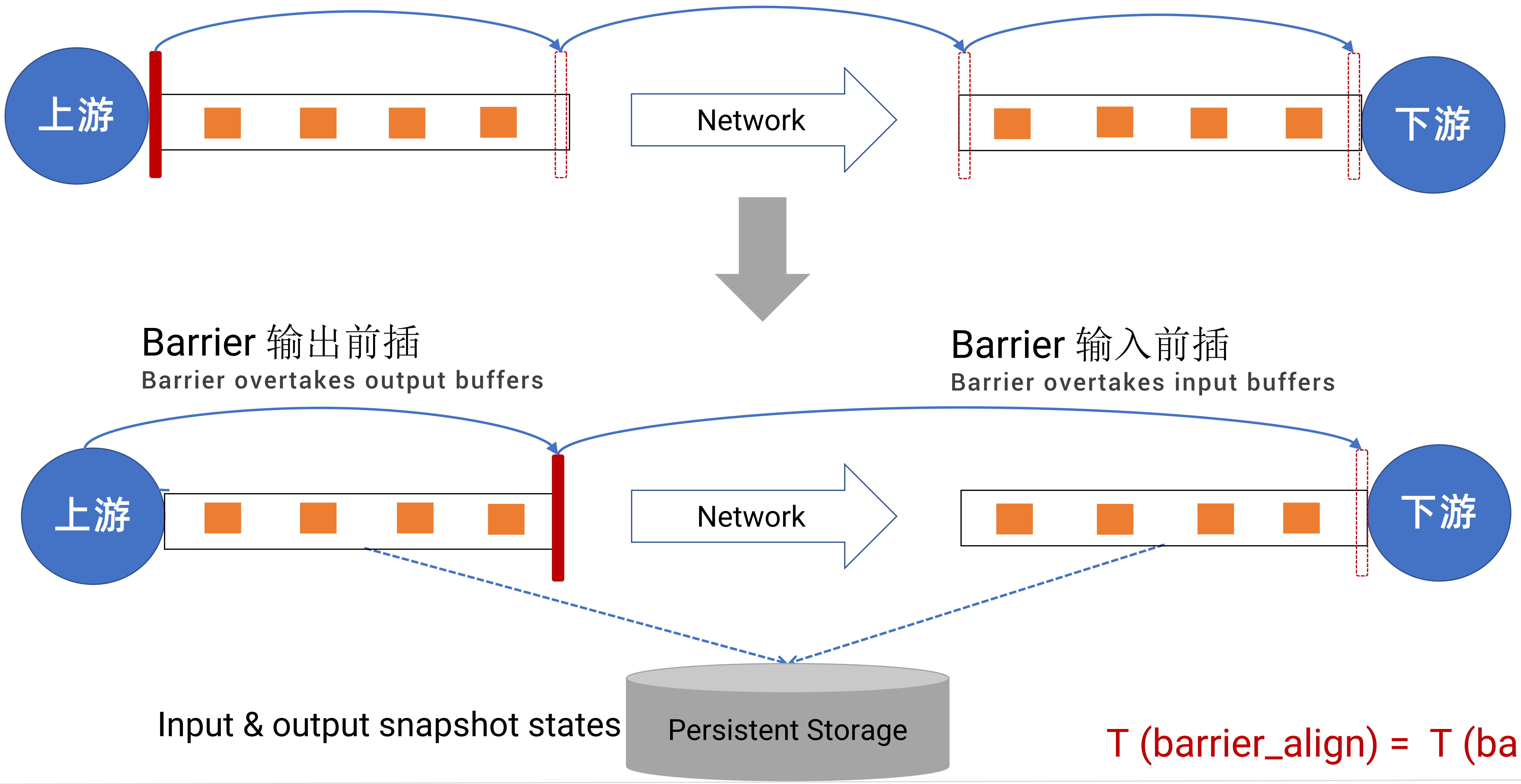

Checkpoint barriers propagate slowly in the case of back pressure
Barrier 传输慢（反压）
Barrier 对齐慢
Long barrier alignment duration
Long end to end checkpoint duration
Checkpoint 耗时长
Very high cost during failure recovery
失败重启代价大
✓
✓
Sink 延迟高
✓
High end to end latency for exactly-once sink
Performance concern for blocked channels processing
阻塞处理影响性能
✓
✓ At least once
✓ Exactly once

解决思路（barrier 跨数据流）

The Solution Idea (Barrier Overtakes Stream Records)

Barrier 跨数据流，加速对齐，未处理数据快照状态

Barriers overtake records which would be snapshotted as state to speed up alignment



Barrier 输出前插

Barrier overtakes output buffers

Barrier 输入前插

Barrier overtakes input buffers

Input & output snapshot states

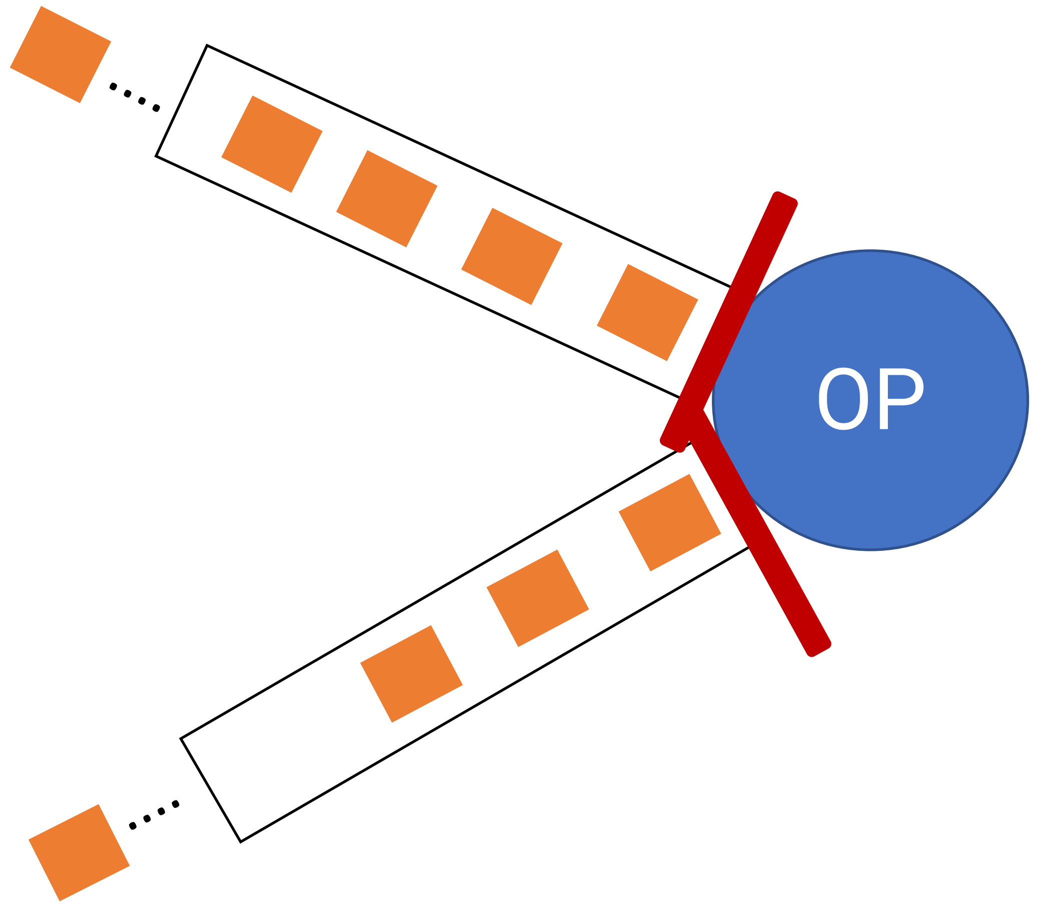
T (barrier_align) = T (barrier_network_transport)

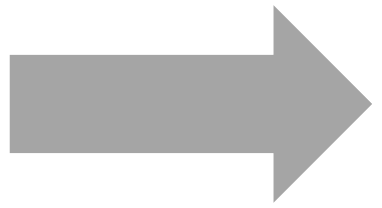
解决思路（Aligned -> Unaligned）

The Solution Idea (Aligned -> Unaligned)

收齐所有 barriers 触发 checkpoint (aligned)

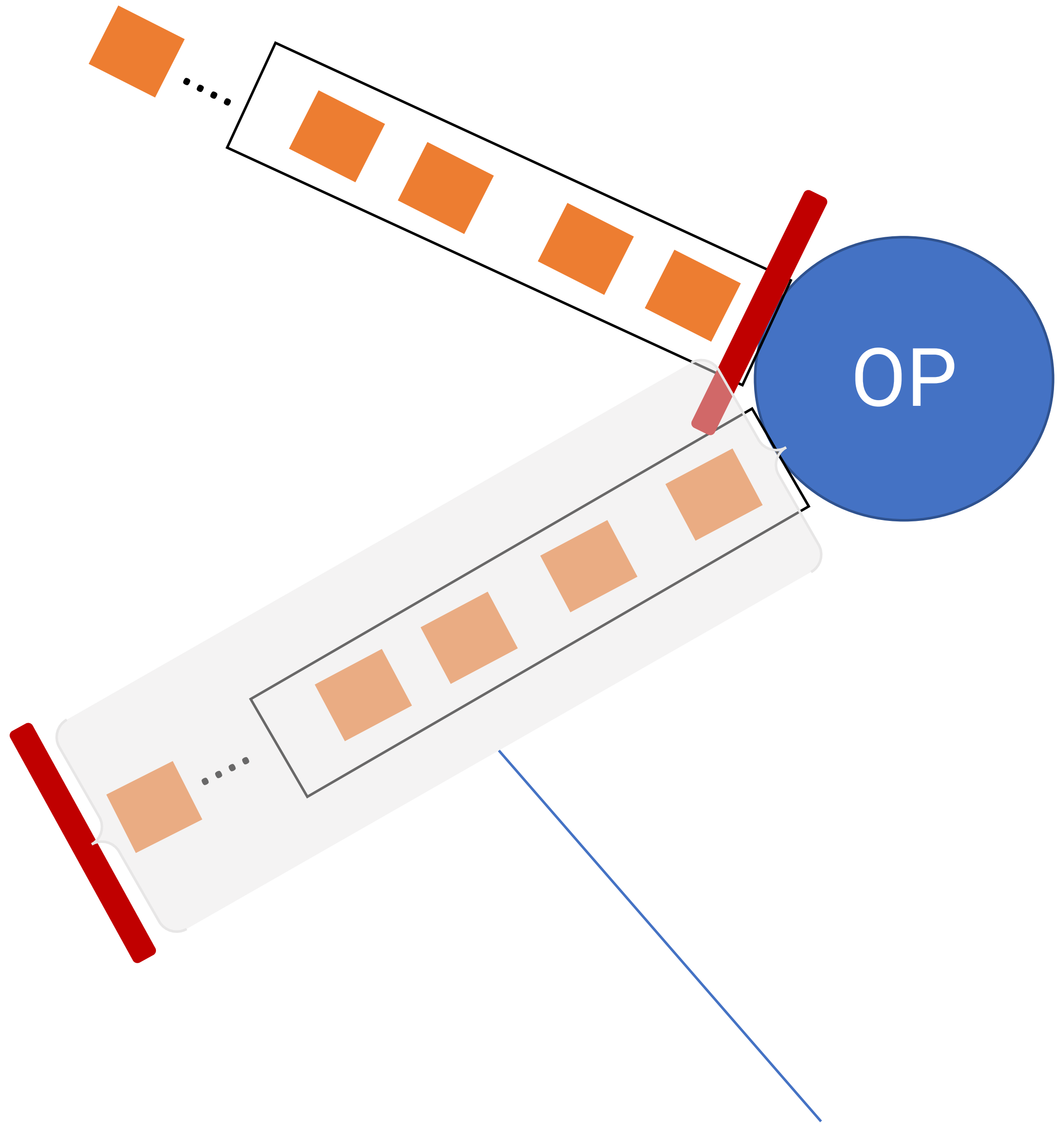
Triggers checkpoint when barrier alignment





收到第一个 barrier 触发 checkpoint (unaligned)

Triggers checkpoint while receiving the first barrier from one channel



快照更多未处理数据

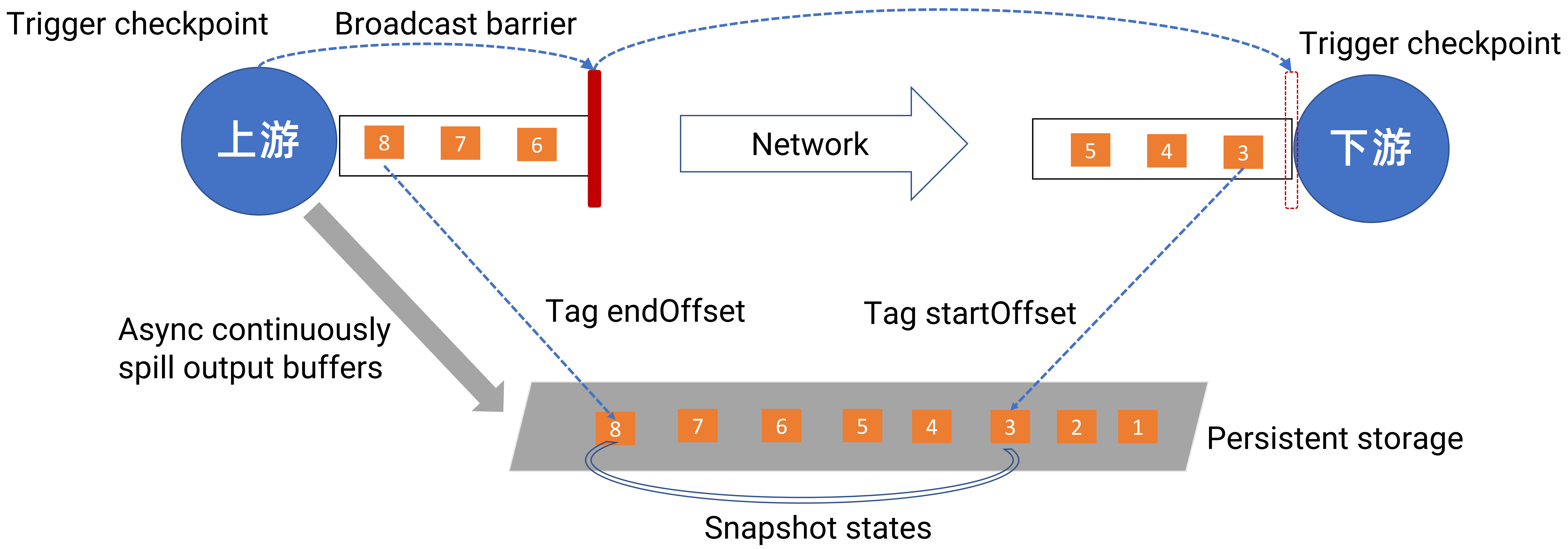
Snapshots more inflight buffers

T (barrier_align) = T (first_barrier_network_transport)

解决思路（如何持久化未处理数据）

The Solution Idea (How to Persist Inflight Buffers)

Option 1 ：Continuously persistent channel



Pros

- **提前持久化数据，缩短 checkpoint 完成时间**
  Persists data beforehand to shorten checkpoint completion time
- **有利于未来的失败容错优化**
  Benefits for the future feature of fine-grained failure recovery

Cons

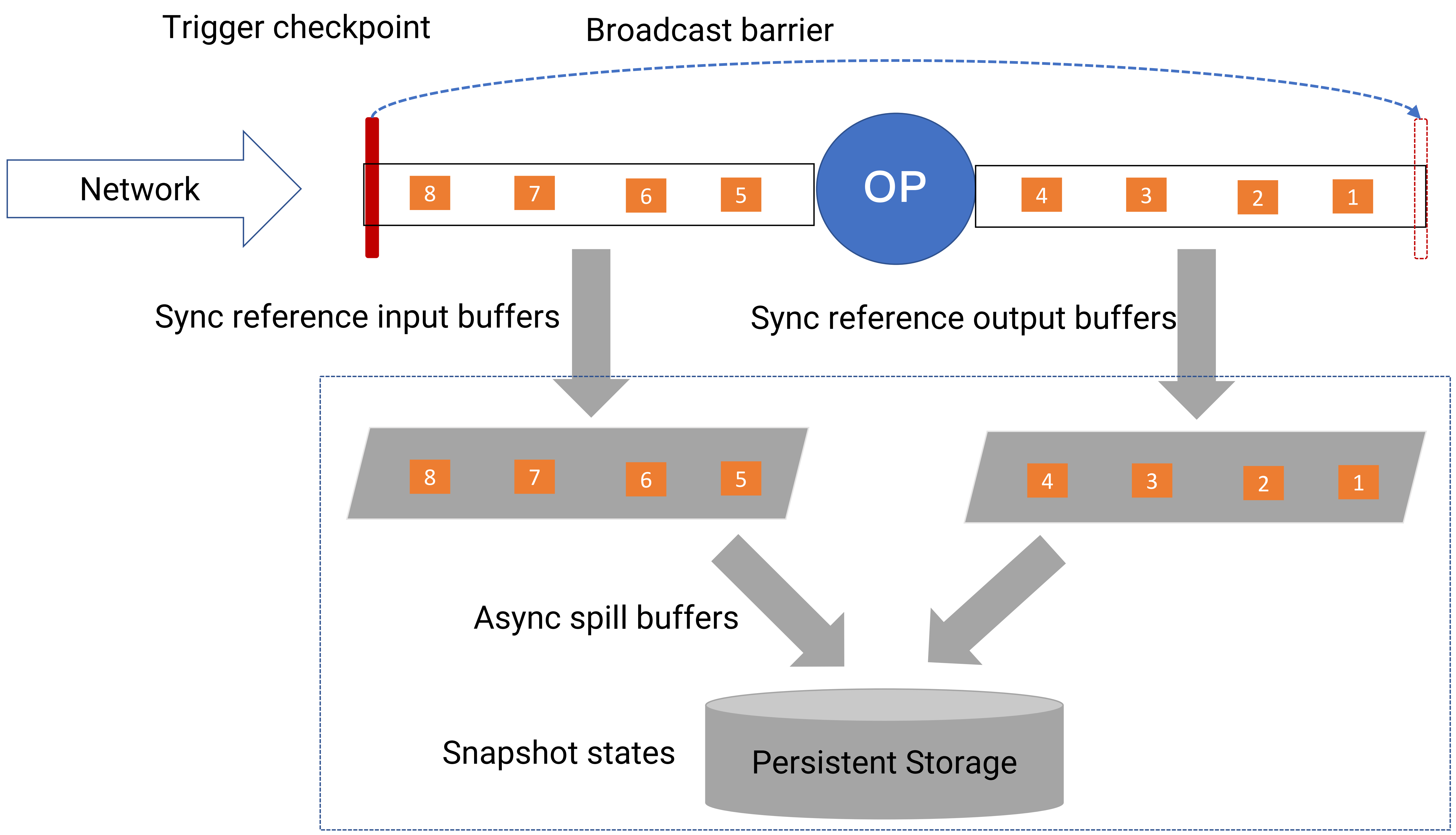
- **持久化数据量大**
  All the output data needs to be persisted

解决思路（如何持久化未处理数据）

The Solution Idea (How to Persist Inflight Buffers)

Option 2：Checkpoint 期间持久化未处理数据

Persists inflight buffers during checkpoint period



Pros：持久化数据总量小于等于网络buffer 总数量，通过配置可估算

The total persisted data is no more than the total network buffers, which can be estimated via configuration

Cons：Checkpoint 期间执行持久化数据，延迟完成时间

It may still need to persist large amount data during checkpoint to delay checkpoint completion

解决思路（失败重启恢复）

The Solution Idea (Failure Restart Recovery)

Rescaling

- Keyed：上游反序列化数据恢复，重新分发保证有序（不考虑网络 shuffle 乱序）

  Upstream can deserialize restored data and emit to downstream based on current key selector. It can guarantee data order without considering network shuffle issue

- Non-keyed：单独 FLIP-X 解决数据有序问题

  Proposes a separate FLIP future to solve the order concern for non-keyed data

进度保证

Guarantee progress

- 恢复期间处理 checkpoint barrier

  Checkpoint barriers should be handled properly while recovering from the previously spilled input and output data

- 恢复期间不重复持久化数据

  The previously spilled data should not be persisted again while processing checkpoint barriers during recovery

总结

The Conclusions

加速了对齐时间，增加了快照状态时间，在数据等待处理和持久化之间权衡

Speeds up the alignment time but increases the checkpoint state size during snapshot. Tradeoff between data processing and persisting

如何配置使用

How to config to use unaligned checkpoint

Benefits

- Checkpoint 更快可预测，尤其在反压场景

  Faster and predictable checkpoint, especially in back pressure case

- Checkpoint 间隔更短，频率更快，失败恢复更少的数据

  Shorter checkpoint internal and more frequent checkpoint resulting in less data to recover during failure

- 端到端 exactly-once 延迟更短

  Decreases end-to-end latency in exactly-once case

Limitations

- Inflight 数据持久化，IO 瓶颈场景不适合

  Not suitable for IO sensitive scenarios because of persisting inflight data

- 失败恢复 inflight 数据，时间变长

  More time to recover inflight data during failure

目前进度

The Current Progress

FLIP-76

实现非阻塞网络输出（FLINK-14551）

Finish non-blocking network output

- Mailbox 模型下不阻塞主线程处理 checkpoint barrier

  The main thread should not be blocked to handle checkpoint barrier based on the mailbox model

PoC

- 基于 Persistent Channel 一直持久化输出

  Continuously persists output data based on persistent channel

- Checkpoint 期间上下游各自持久化输入输出

  Persists both input and output data for upstream and downstream separately during checkpoint period

- Benchmark 验证

  Benchmark verifies the above PoC

预期完成： Release-1.11

Expected done

THANKS