Runtime Improvements in Blink for Large Scale Streaming at Alibaba

Feng Wang

Zhijiang Wang

April, 2017

Outline

1 Blink Introduction

2 Improvements to Flink Runtime

3 Future Plans

Blink Introduction

Section 1

Blink – Alibaba's version of Flink

✓Looked into Flink two since 2 years ago

- best choice of unified computing engine
- a few of issues in flink that can be problems for large scale applications

✓Started Blink project

- aimed to make Flink work reliably and efficiently at the very large scale at Alibaba

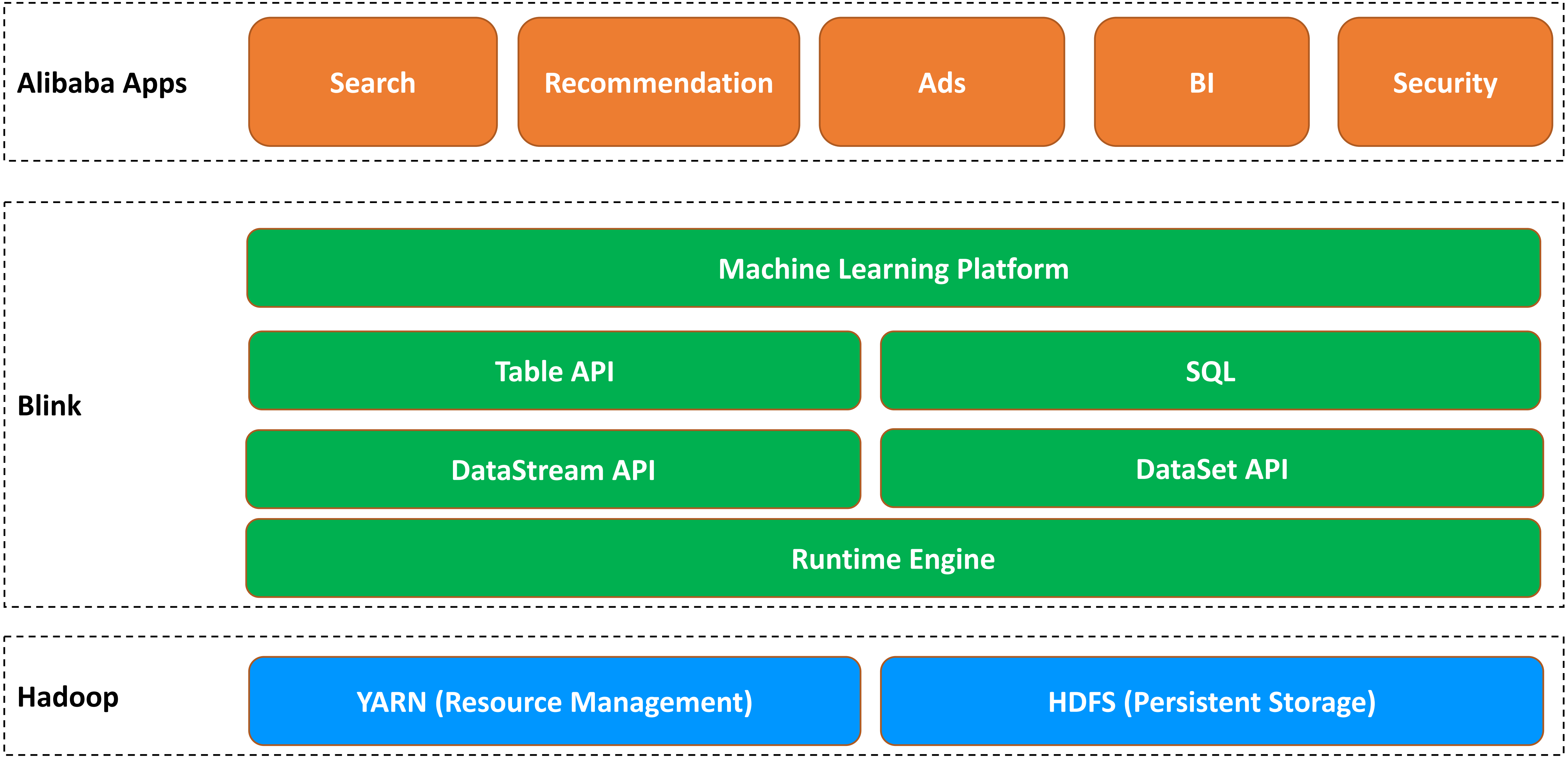
✓Made various improvements in Flink runtime

- Runs natively on yarn cluster
- failover optimizations for fast recovery
- incremental checkpoint for large states
- async operator for high throughputs

✓Working with Flink community to contribute changes back since last August

- several key improvements
- hundreds of patches

Blink Ecosystem in Alibaba



Blink in Alibaba Production

✓In production for almost one year

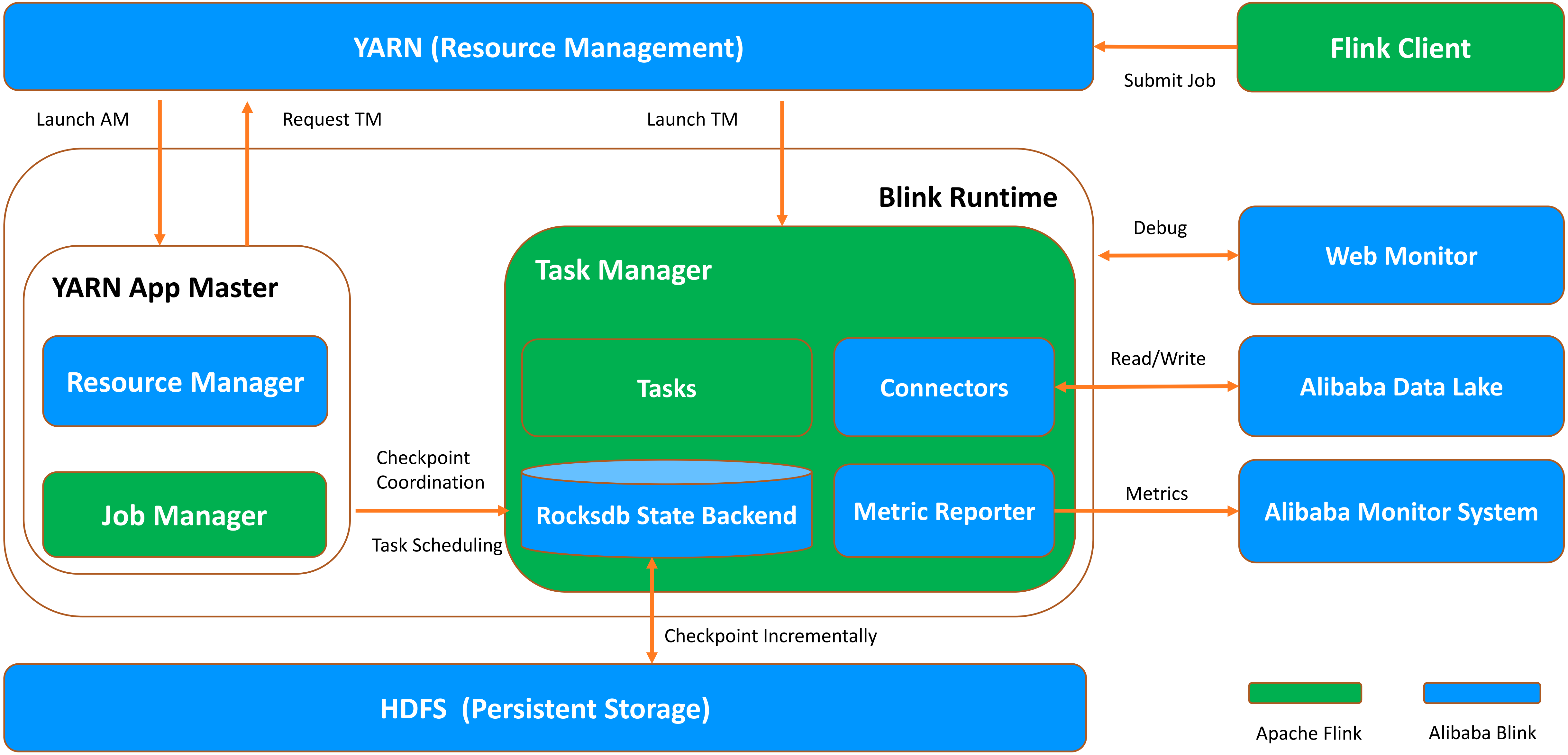
✓Run on thousands of nodes

- hundreds of jobs
- The biggest cluster is more than 1000 nodes
- the biggest job has 10s TB states and thousands of subtasks

✓Supported key production services on last Nov 11th, China Single's Day

- China Single's Day is by far the biggest shopping holiday in China, similar to Black Friday in US
- Last year it recorded $17.8 billion worth of gross merchandise volumes in one day
- Blink is used to do real time machine learning and increased conversion by around 30%

Blink Architecture



Improvements to Flink Runtime

Section 2

Improvements to Flink Runtime

✓Native integration with Resource Management

- Take YARN for an Example

✓Performance Improvements

- Incremental Checkpoint
- Asynchronous Operator

✓Failover Optimization

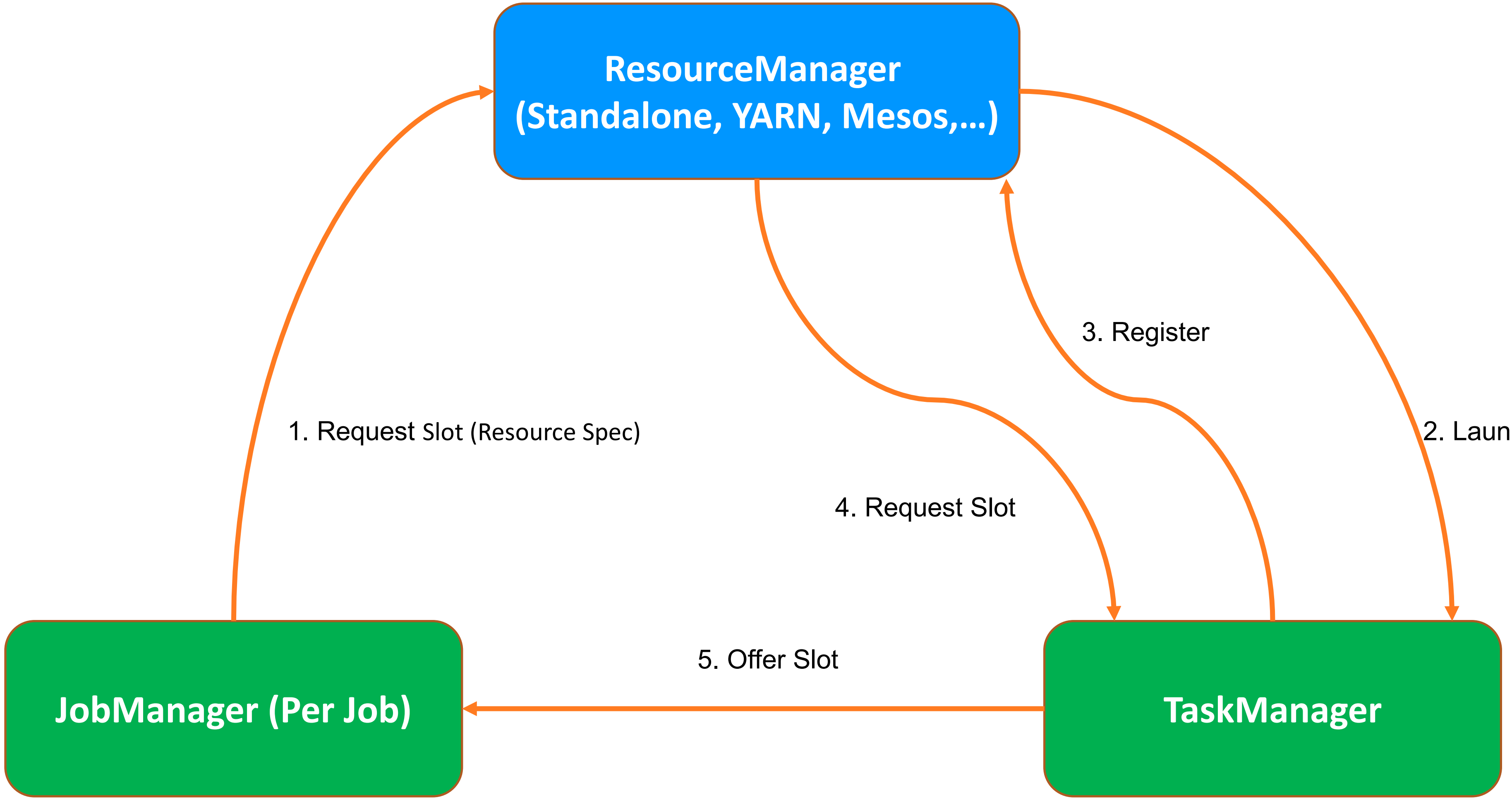
- Fine-grained Recovery for Task Failures
- Allocation Reuse for Task Recovery
- Non-disruptive JobManager failure recovery

Native integration with Resource Management

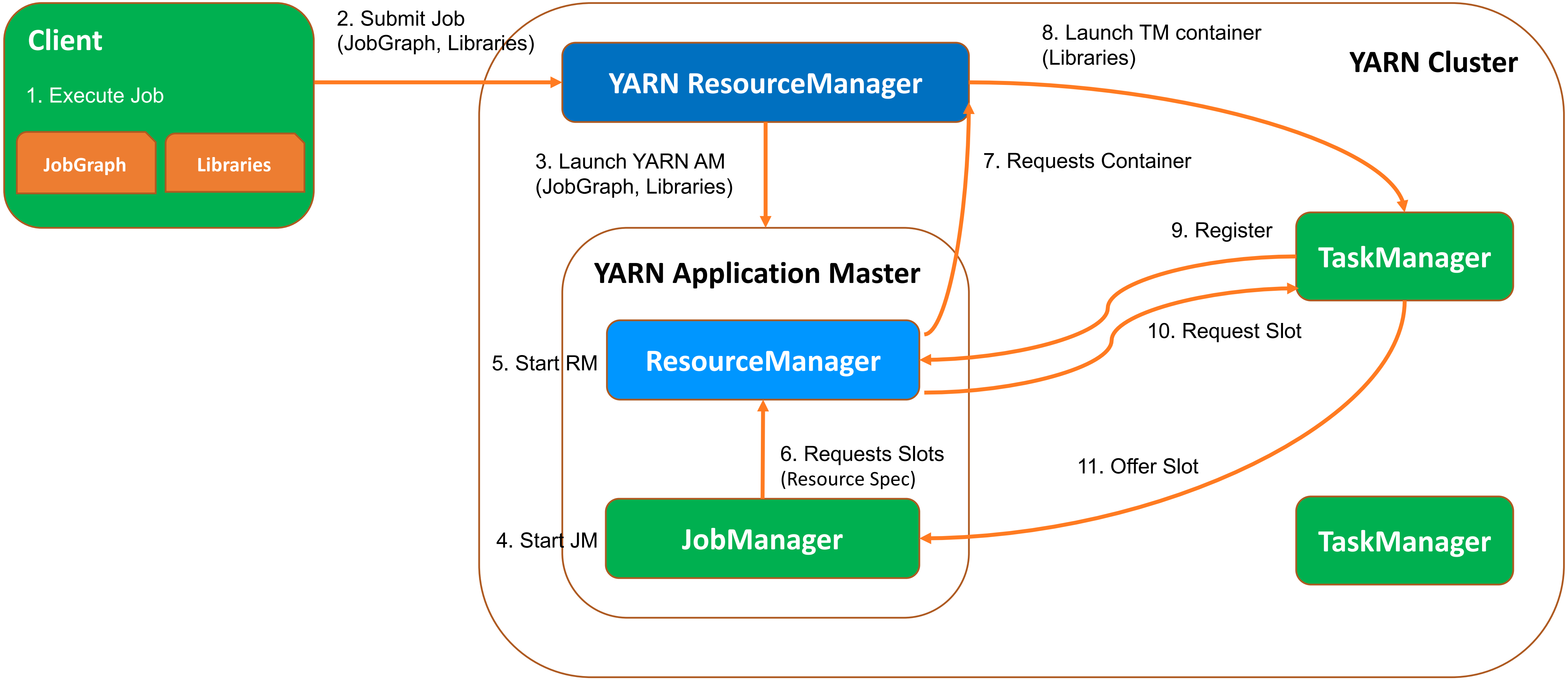
✓Background

- Cluster resource is allocated upfront. The resource utilization can be not efficient
- A single JobManager handles all the jobs, which limits the scale of the cluster

Native integration with Resource Management



Native integration with YARN



Incremental Checkpoint

✓Background

- The job state could be very large (many TBs)
- The state size of individual task can be many GBs

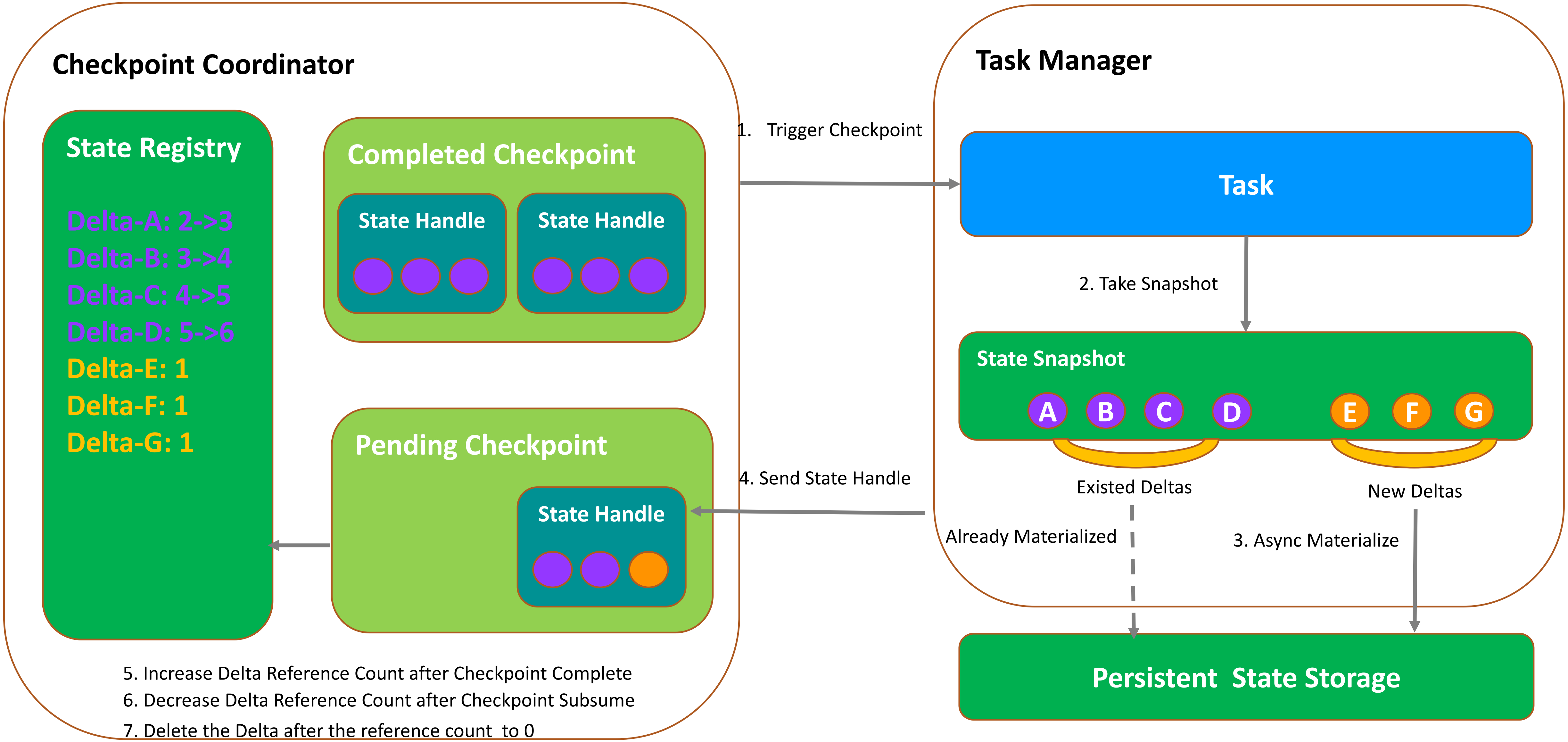
✓The problems of Full Checkpoint

- Materialize all the states to persistent store at each checkpoint
- As the states get bigger, materialization may take too much time to finish
- One of the biggest blocking issues in large scale production

✓Benefits of Incremental Checkpoint

- Only the modified states since last checkpoint need to be materialized
- The checkpoint will be faster and more efficient

Incremental Checkpoint – How It Works



Checkpoint Coordinator

State Registry

Delta-A: 2->3

Delta-B: 3->4

Delta-C: 4->5

Delta-D: 5->6

Delta-E: 1

Delta-F: 1

Delta-G: 1

Completed Checkpoint

State Handle

State Handle

Pending Checkpoint

State Handle

1. Trigger Checkpoint

4. Send State Handle

Task Manager

Task

2. Take Snapshot

State Snapshot

A B C D E F G

Existed Deltas

New Deltas

Already Materialized

3. Async Materialize

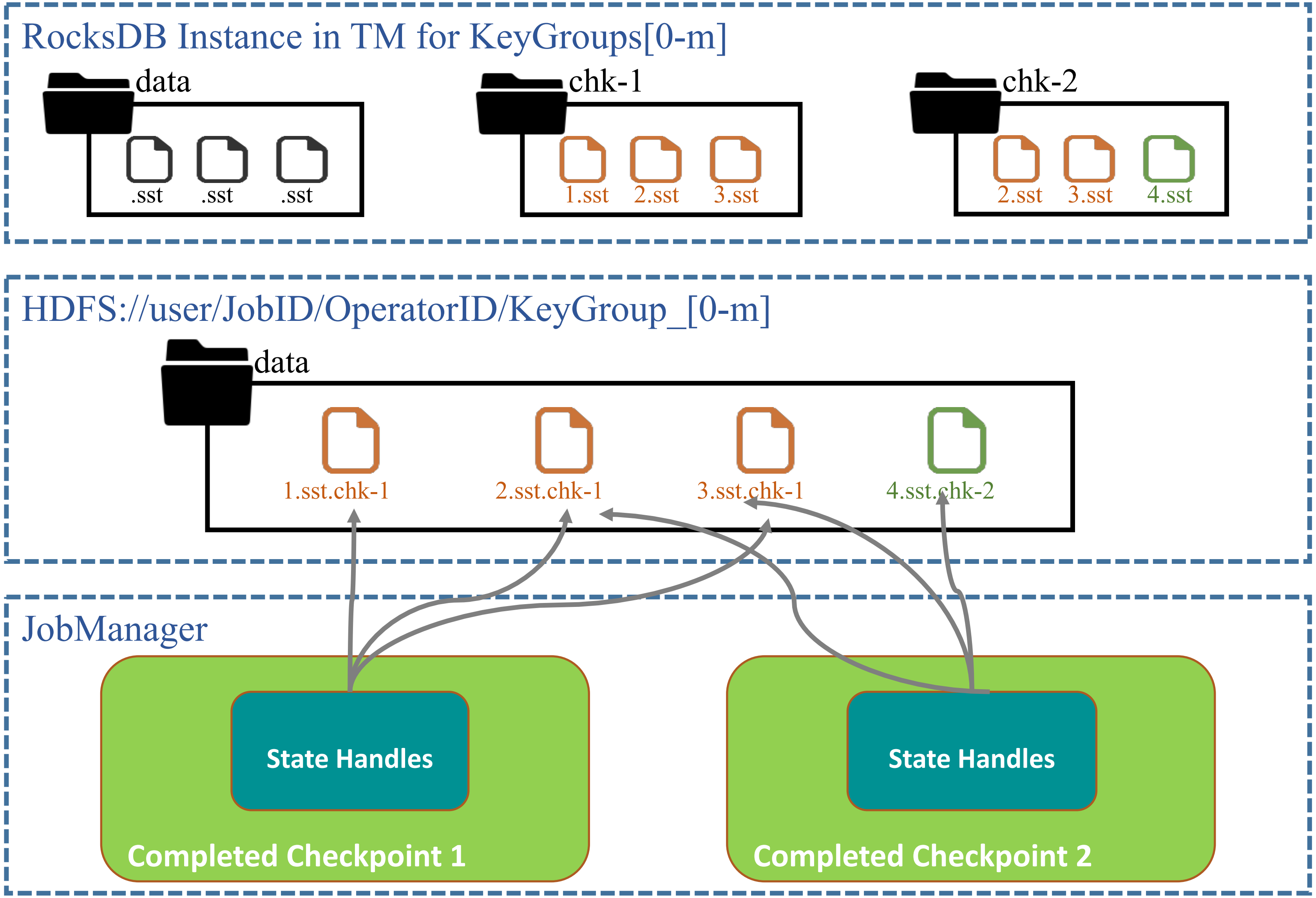
Persistent State Storage

5. Increase Delta Reference Count after Checkpoint Complete

6. Decrease Delta Reference Count after Checkpoint Subsume

7. Delete the Delta after the reference count to 0

Incremental Checkpoint - RocksDB State Backend Implementation



Asynchronous Operator

✓Background

- Flink task use a single thread to process events
- Flink task sometimes need to access external services (hbase, redis,…)

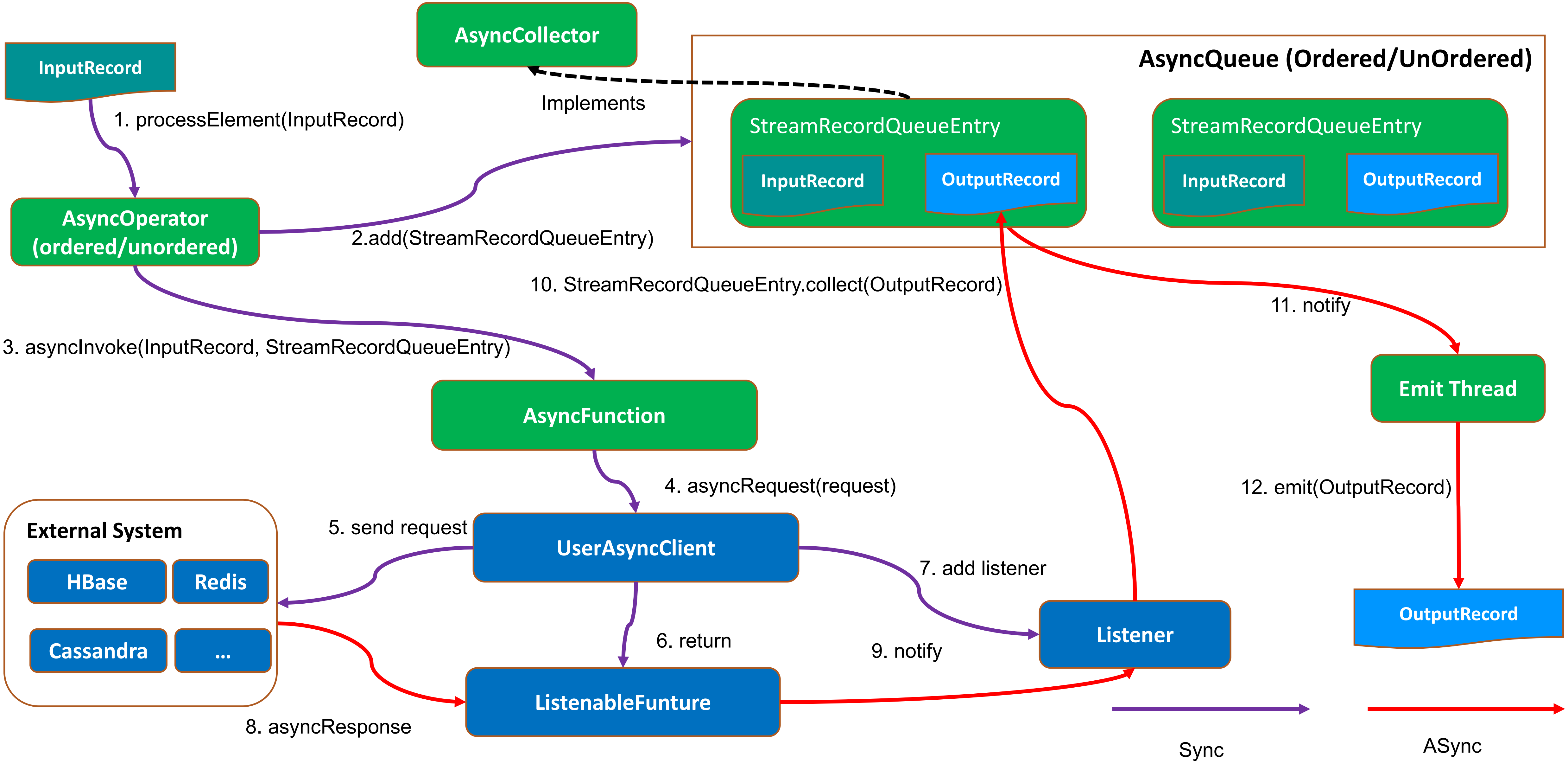
✓Problem

- The high latency may block the event processing
- Throughput can be limited by latency

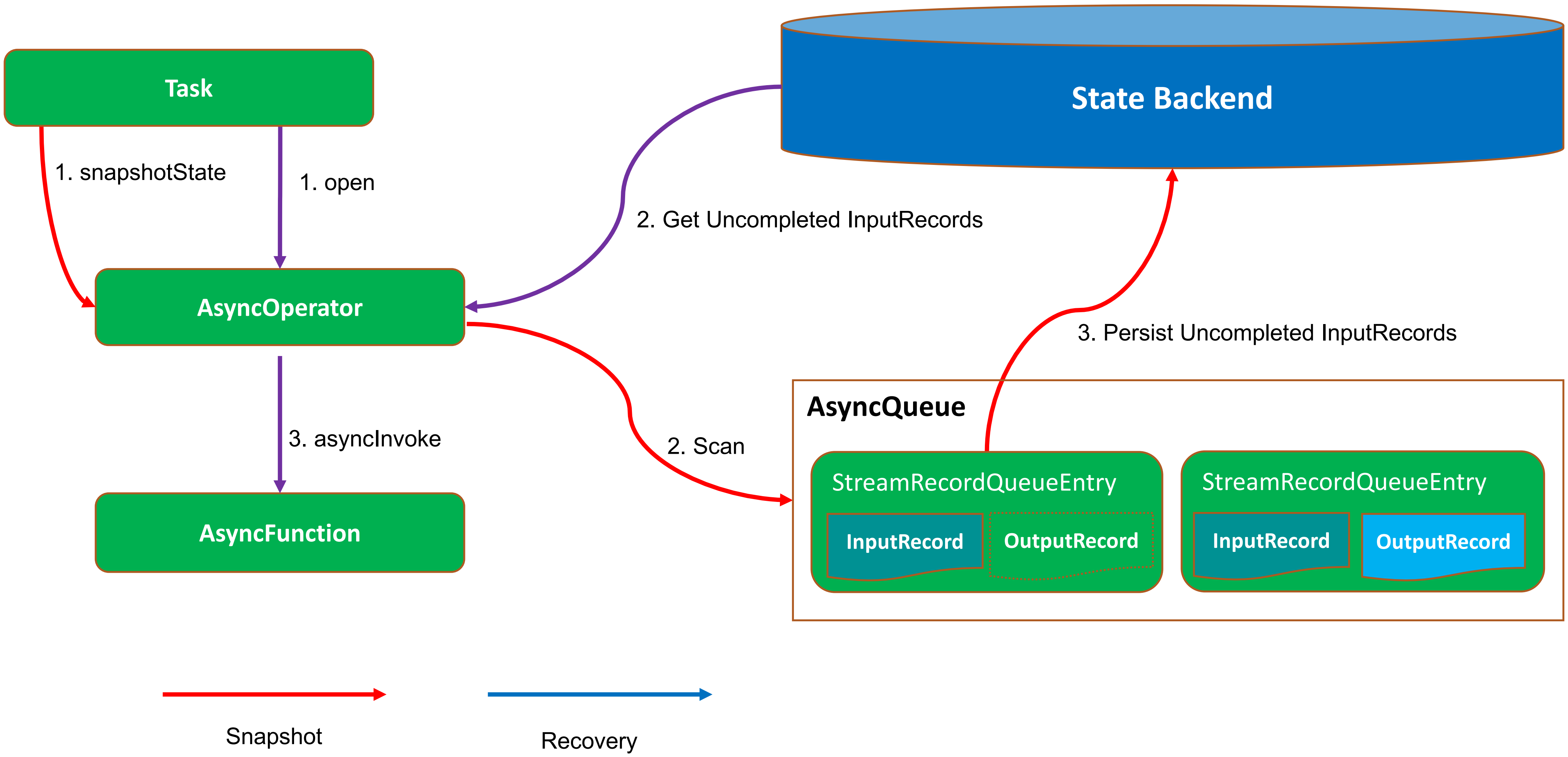
✓Benefits of Asynchronous Operator

- Decouple the throughput of task from the latency of external system
- Improve the CPU utilization of the Flink tasks
- Simplify resource specification for Flink jobs

Asynchronous Operator – How It Works



Asynchronous Operator – How It Manages State



Fine-grained Recovery from Task Failures

✓Status of batch job

- Large scale to thousands of nodes
- Node failures are common in large clusters
- Prefer non-pipelined mode due to limited resource

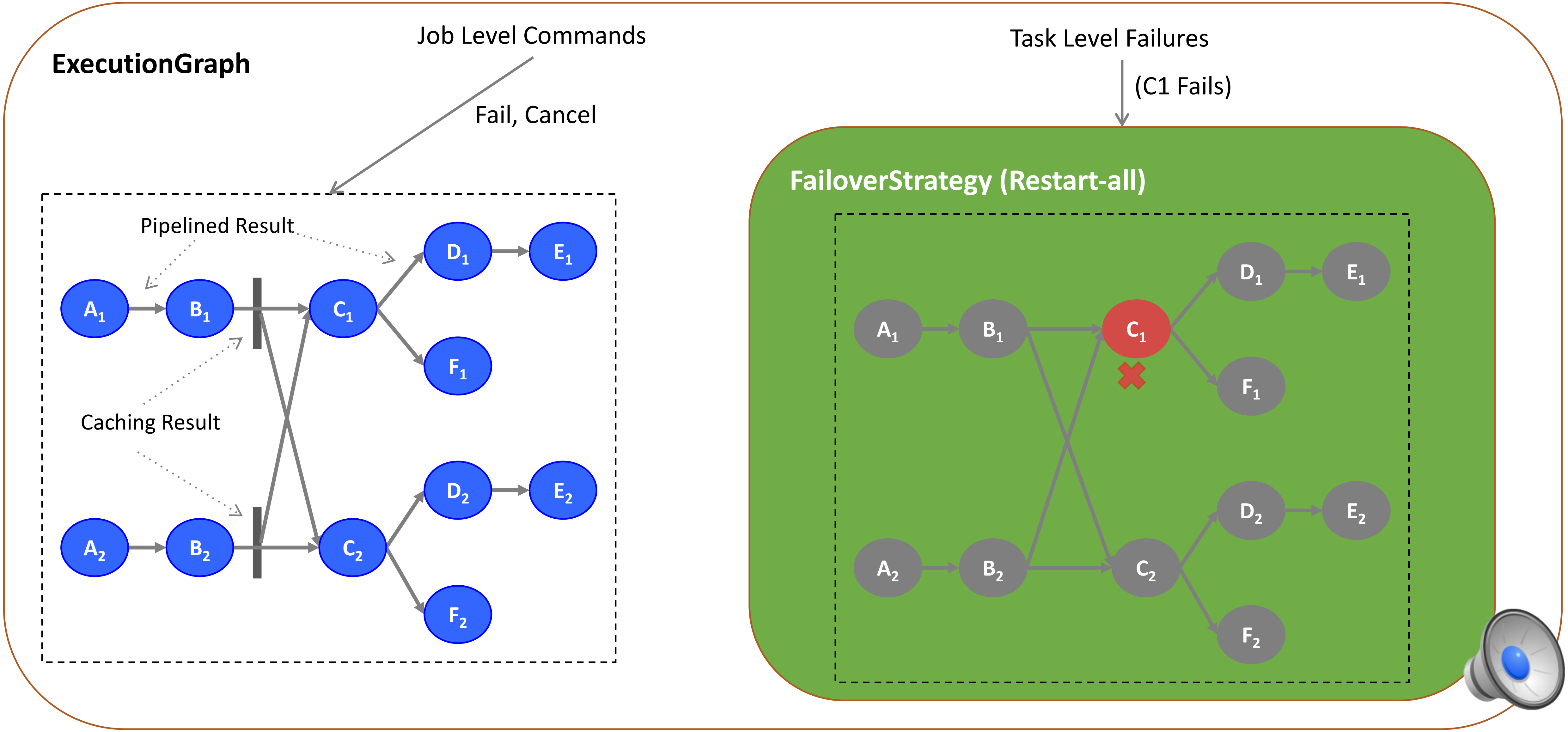
✓Problems

- One task failure needs to restart the entire execution graph
- It is especially critical for batch jobs

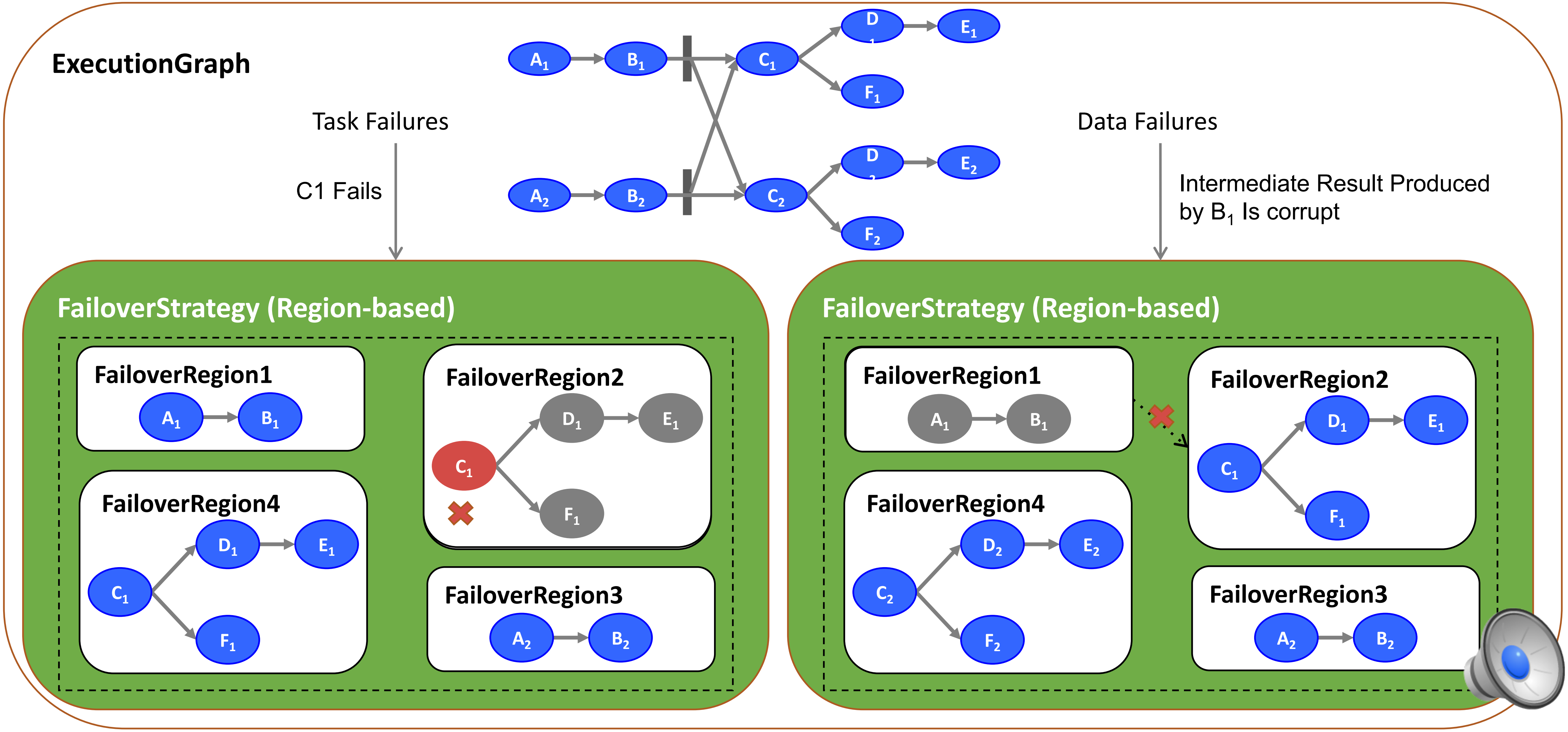
✓Benefit

- Make recovery more efficient by restarting only what needs to be restarted

Fine-grained Recovery from Task Failures – Restart-all Strategy



Fine-grained Recovery from Task Failures – Region-based Strategy



ExecutionGraph

Task Failures

C1 Fails

Data Failures

Intermediate Result Produced by B_1 Is corrupt

FailoverStrategy (Region-based)

FailoverRegion1

FailoverRegion2

FailoverRegion4

FailoverRegion3

FailoverStrategy (Region-based)

FailoverRegion1

FailoverRegion2

FailoverRegion4

FailoverRegion3

Allocation Reuse for Task Recovery

✓Background

- The job state can be very big in Alibaba, hence use RocksDB as state backend
- State restore by RocksDB backend involves in copying data from HDFS

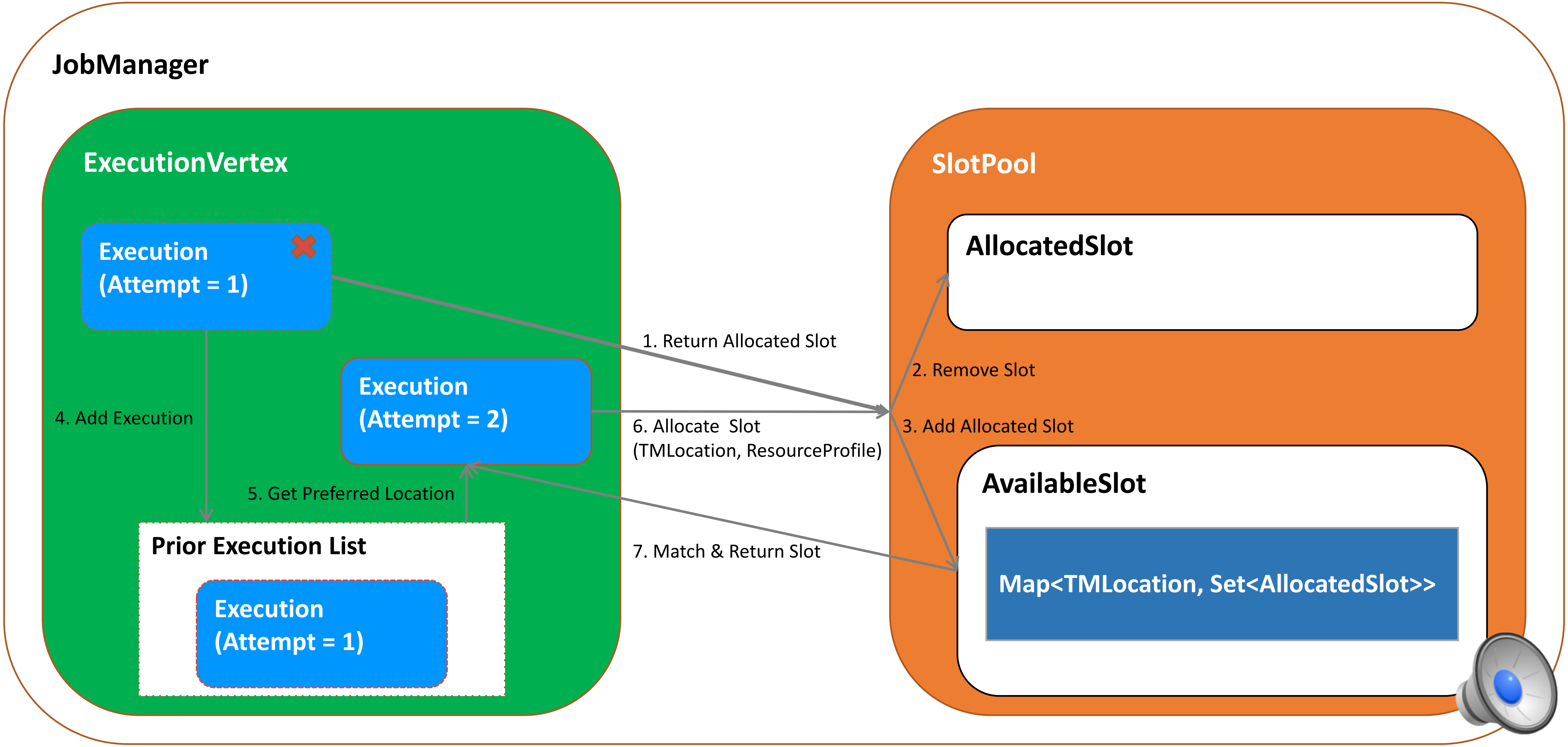
✓Problem

- It is expensive to restore state from HDFS during task recovery

✓Benefits of Allocation Reuse

- Deploy the restarted task in previous allocation to speed up recovery
- Restore state from local RocksDB to avoid copying data from HDFS

Allocation Reuse for Task Recovery – How It Works



Non-disruptive JobManager Failures via Reconciliation

✓Background

- The job is large scale to thousands of nodes
- The job state can be very big in TB level

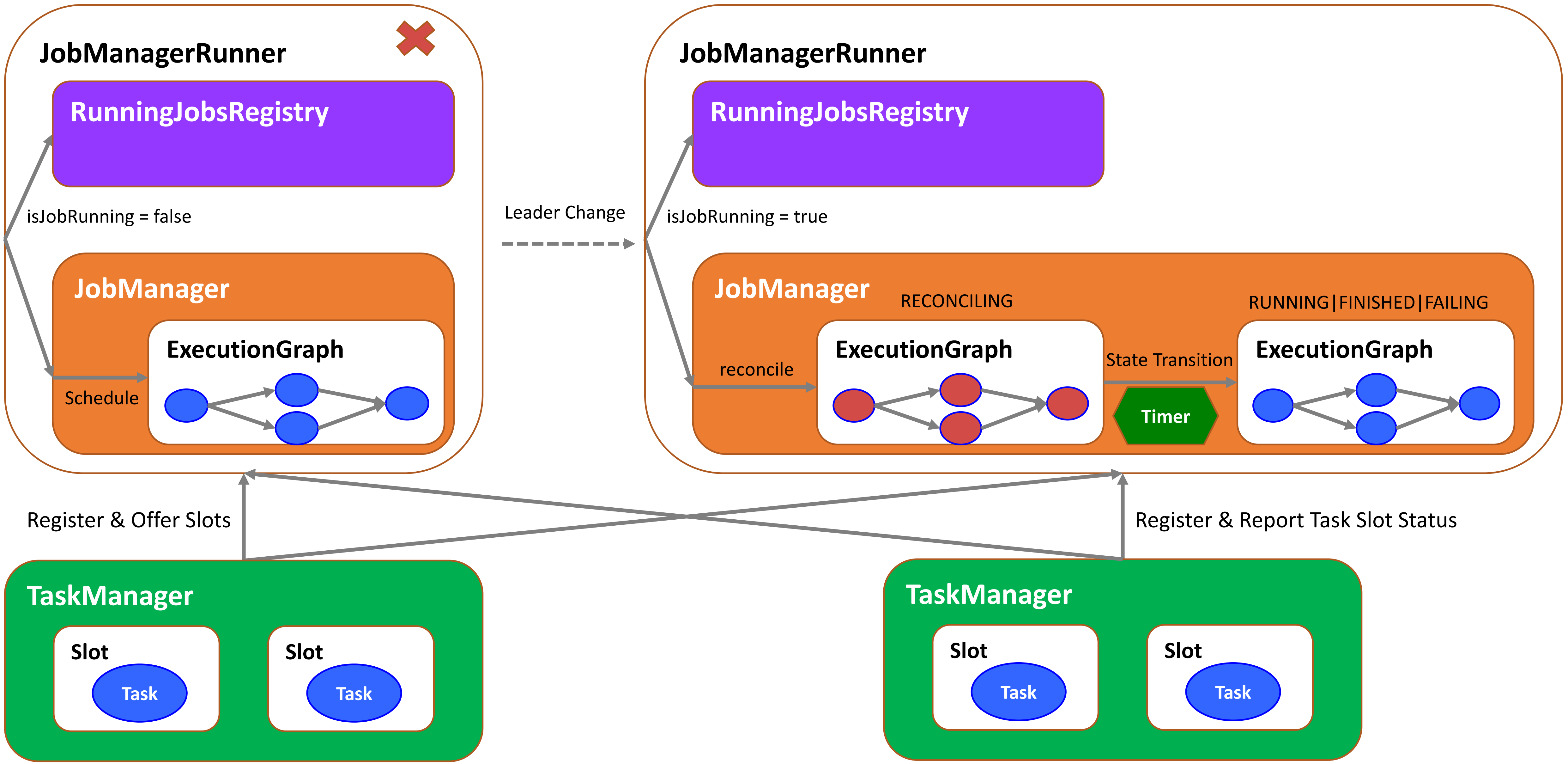
✓Problems

- All the tasks are restarted for job manager failures
- The cost is high for recovering states

✓Benefit

- Improve job stability by automatic reconciliation to avoid restarting tasks

Non-disruptive JobManager Failures via Reconciliation – How It works



Future Plans

Section 3

Future Plans

✓Blink is already popular in the streaming scenarios

- more and more streaming applications will run on blink

✓Make batch applications run on production

- increase the resource utilization of the clusters

✓Blink as Service

- Alibaba Group Wide

✓Cluster is growing very fast

- cluster size will double
- thousands of jobs run on production

Thanks

We are Hiring!

aliserach-recruiting@list.alibaba-inc.com