On-the-fly Reconfiguration of Query Plans for Stateful Stream Processing Engines

BTW 2019 - 7. March 2019

Adrian Bartnik

Bonaventura Del Monte

Dr. Tilmann Rabl

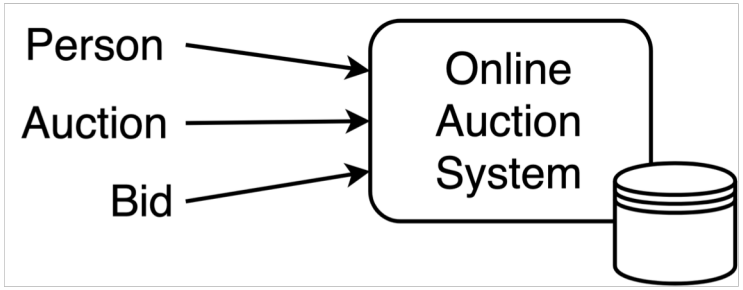
Dr. Volker Markl

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

Motivation

- Stream processing engines emphasize **processing velocity**
- SPEs can handle **unbounded data sets**
- Many streaming workloads **vary over time**



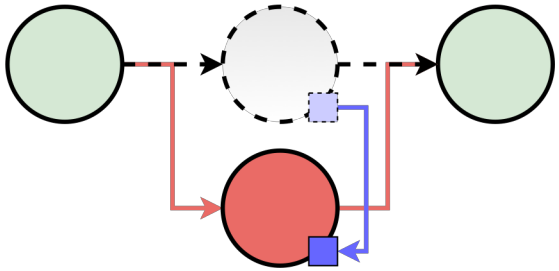
⇨ Adapt streaming job to incoming workload

Problem Statement

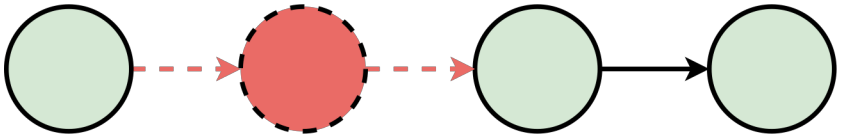
- Restarting job only way to modify running streaming job
  - May involve distribution of large state
  - Require external system to guarantee correct processing semantics
  - Not possible if other systems rely on streaming output

Proposed Solution

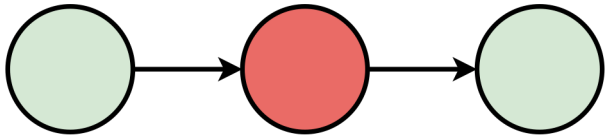
- Modify a running streaming job in Apache Flink
  - Stateful operator migration



  - Introduction of new operators

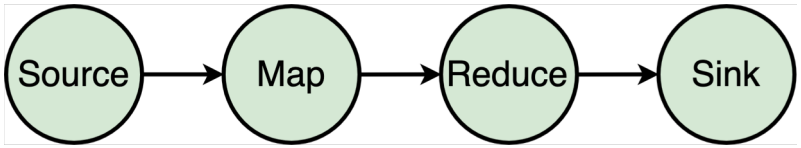


  - Optimize or replace existing operators

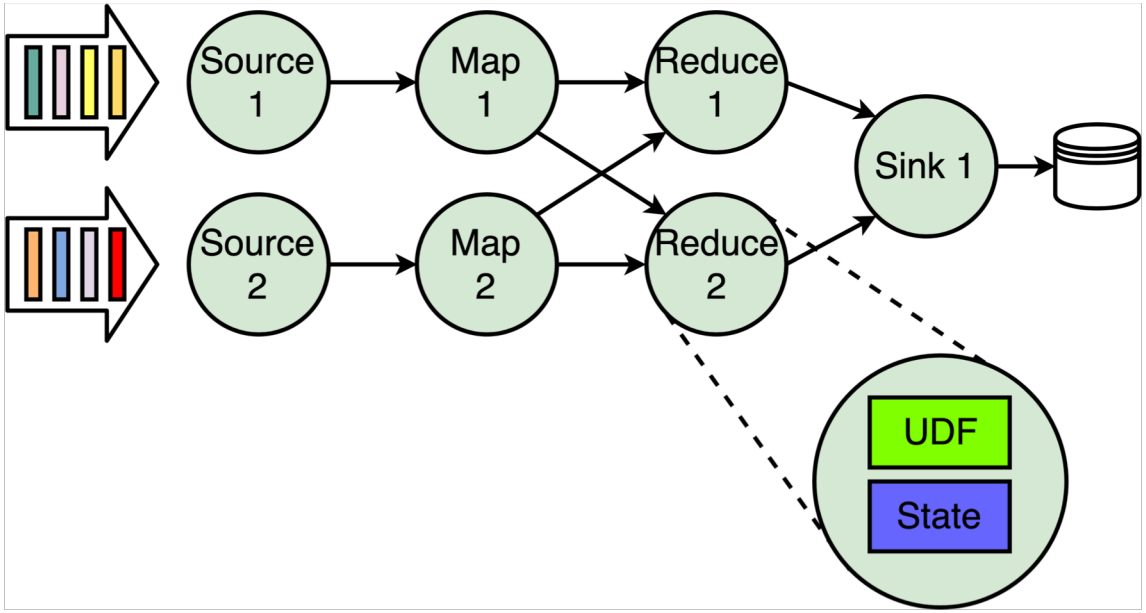


Apache Flink – Streaming Job

- Dataflow of streams and operators form DAG

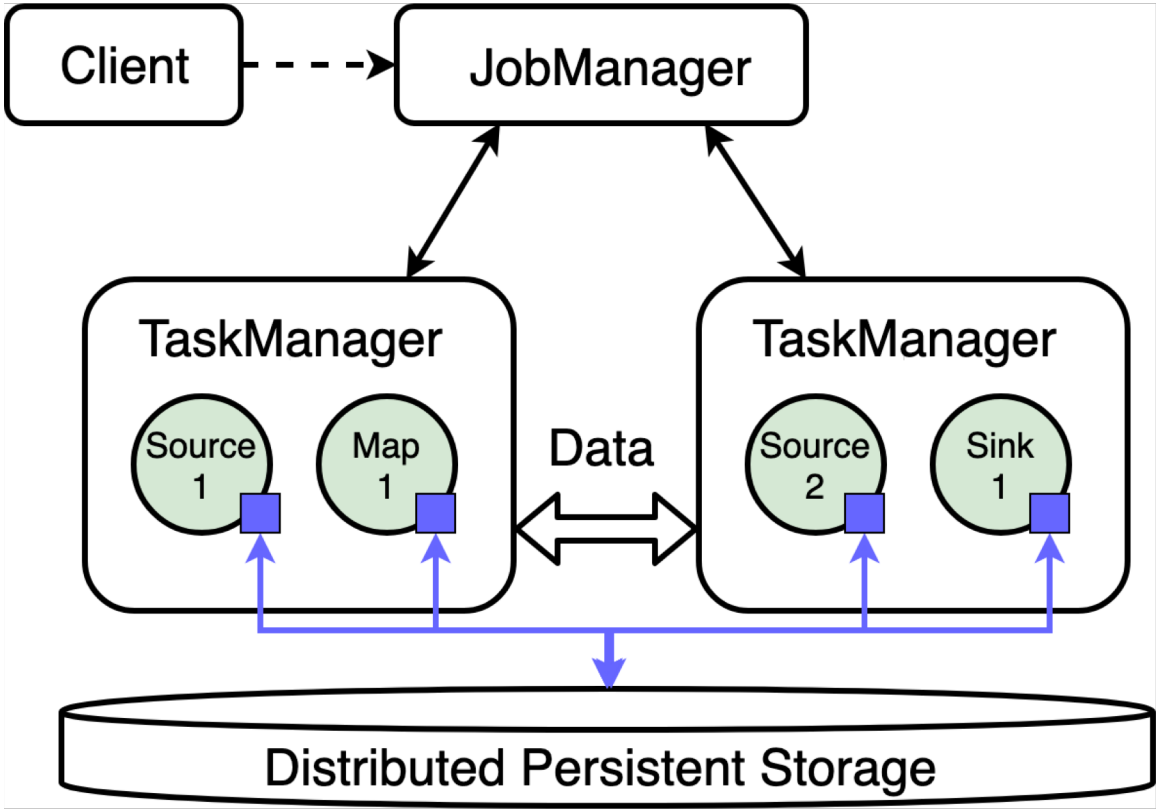


Flink deployment



Apache Flink – Architecture

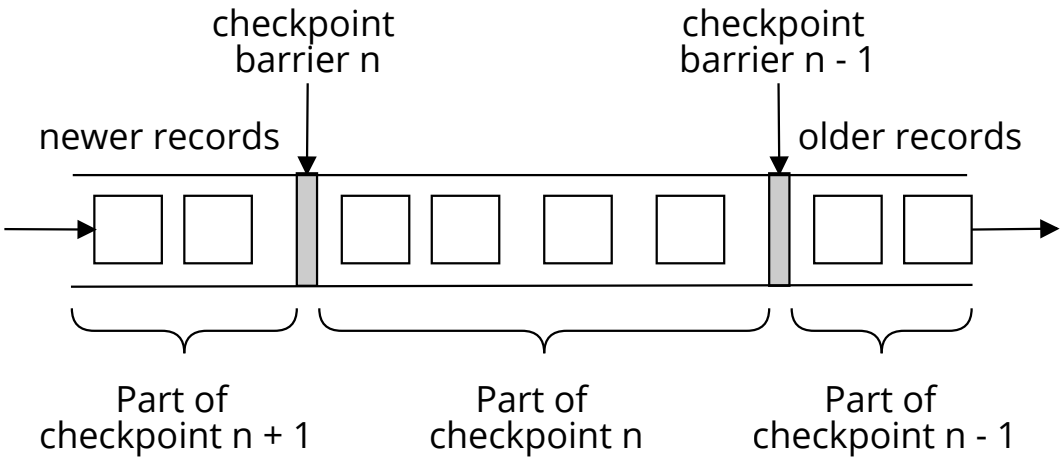
- Dataflow of streams and transformations form DAG



Apache Flink – Checkpointing Mechanism

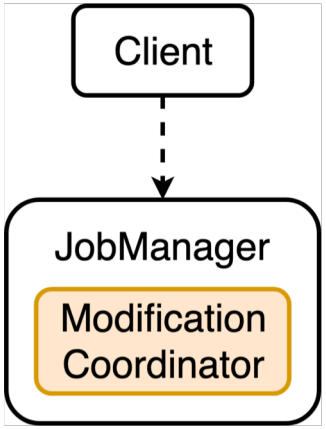
- Checkpoint mechanism ensures **fault tolerance** and allows jobs to be restarted with previous state

- Checkpoint barriers divide stream into indepentent and **disjoint subsets**

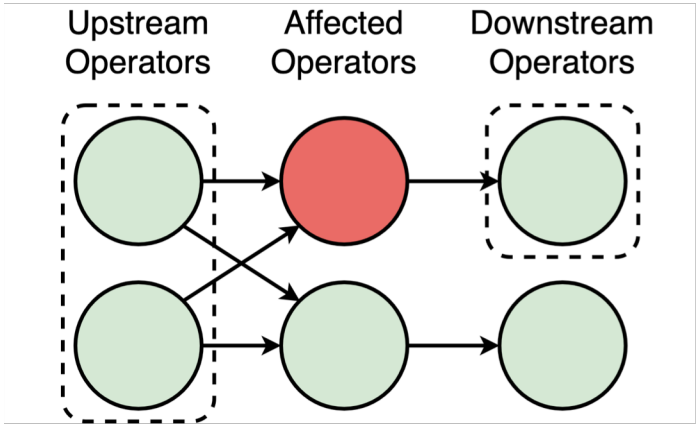


Modification Mechanism

- Integration with checkpoint mechanism
  - Ensures exactly-once processing semantics
  - Reuse of Flink state backend

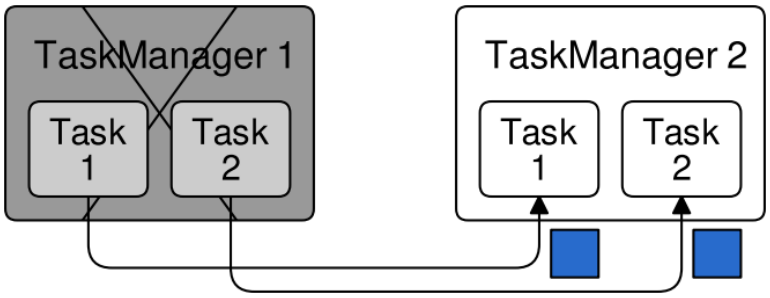


- ModificationCoordinator
  - checks validity of modification
  - Ingests trigger messages
  - starts and supervises modification execution

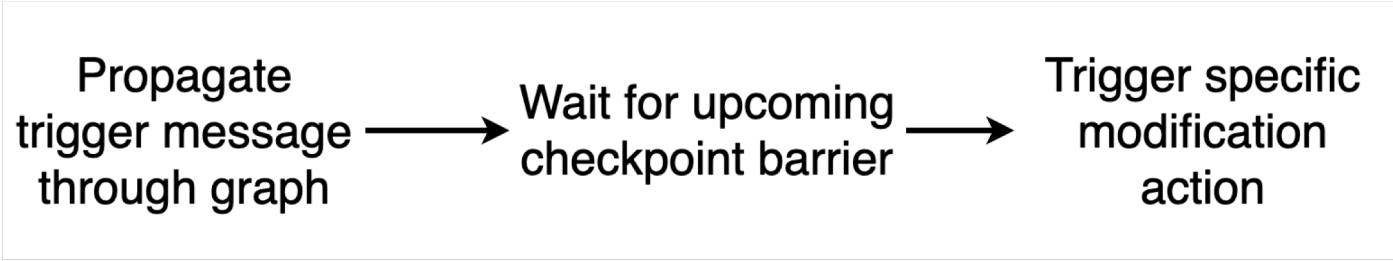


Migrating Operators

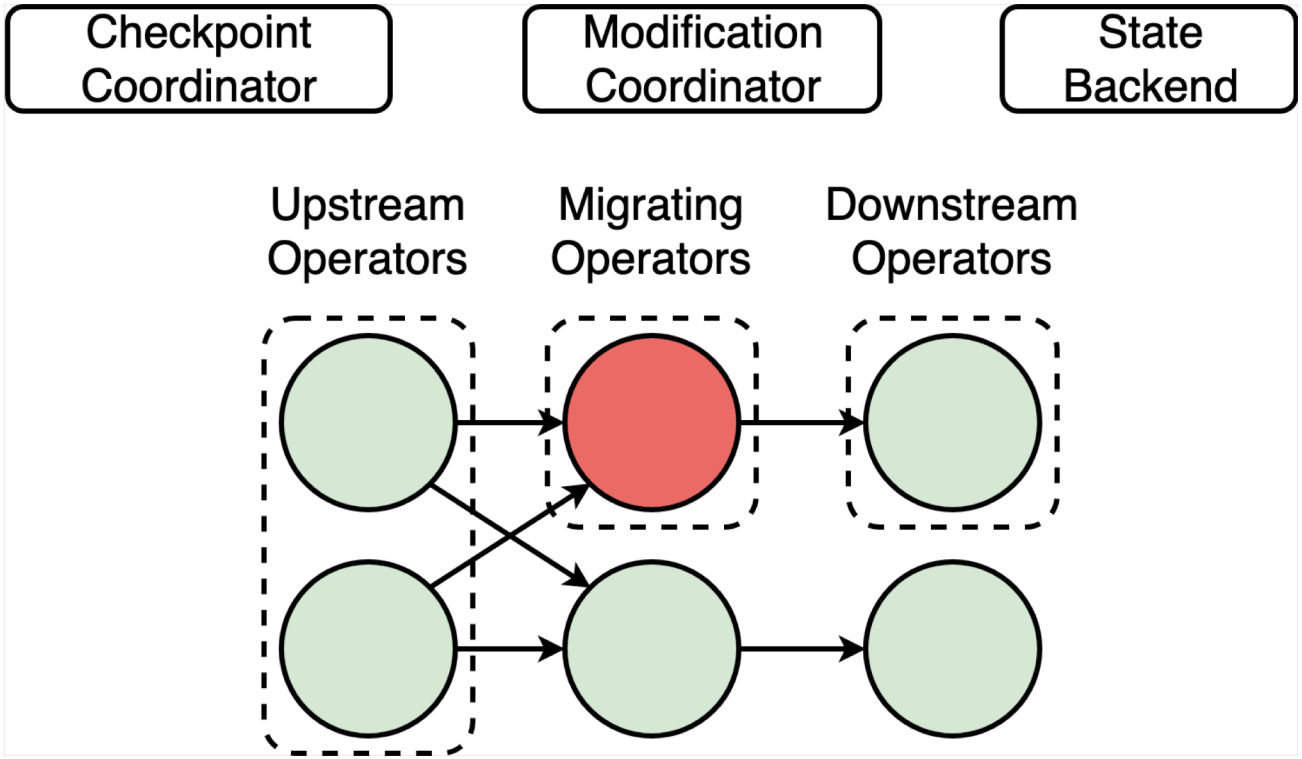
- Motivation: TaskManager metrics indicate an upcoming hardware failure but job should not stop



1. Preparing Migration – 2. Waiting - 3. Performing Migration

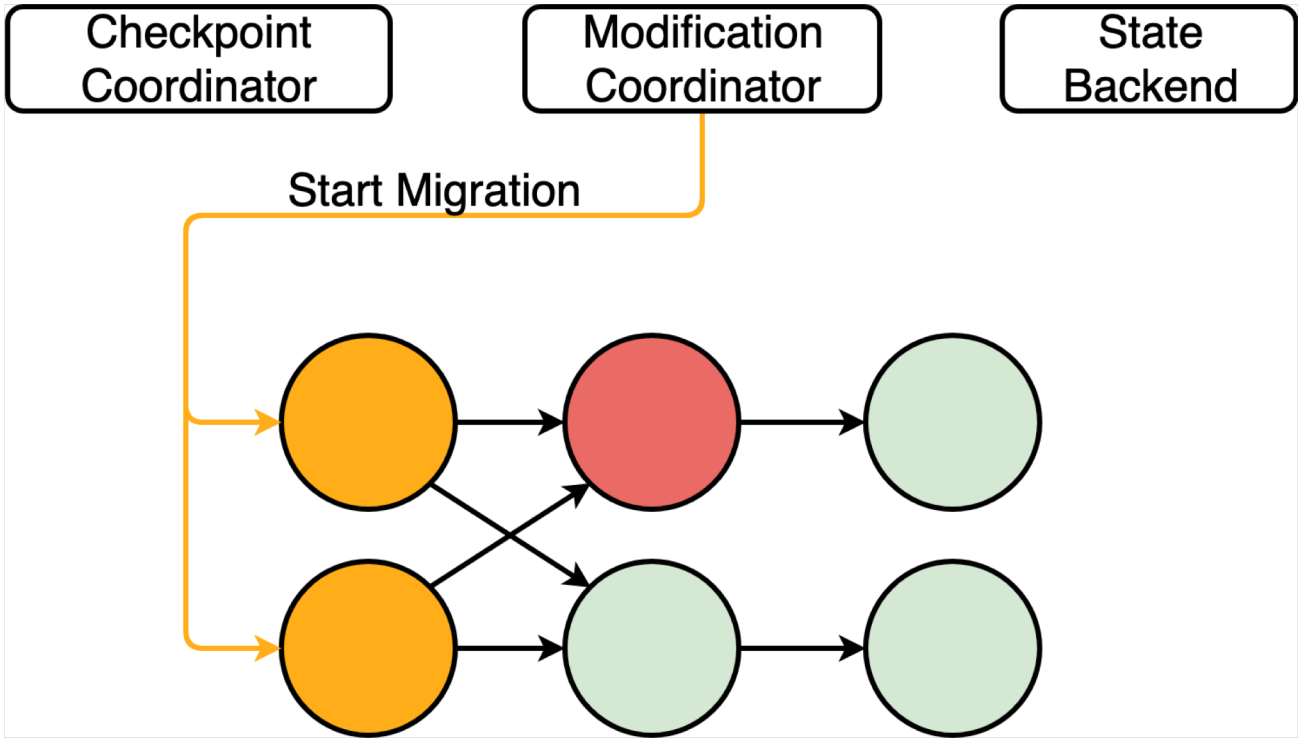


Migrating Operators



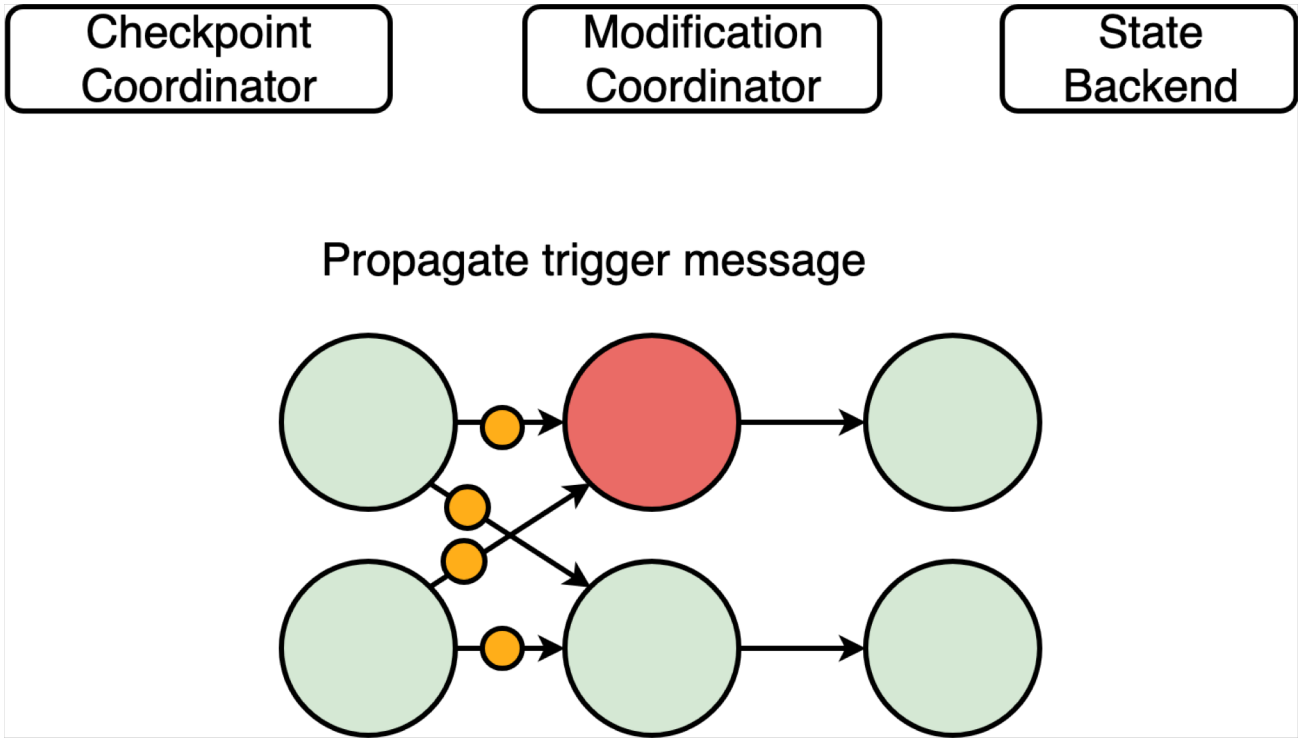
Preparing Migration – Waiting - Performing Migration

Migrating Operators



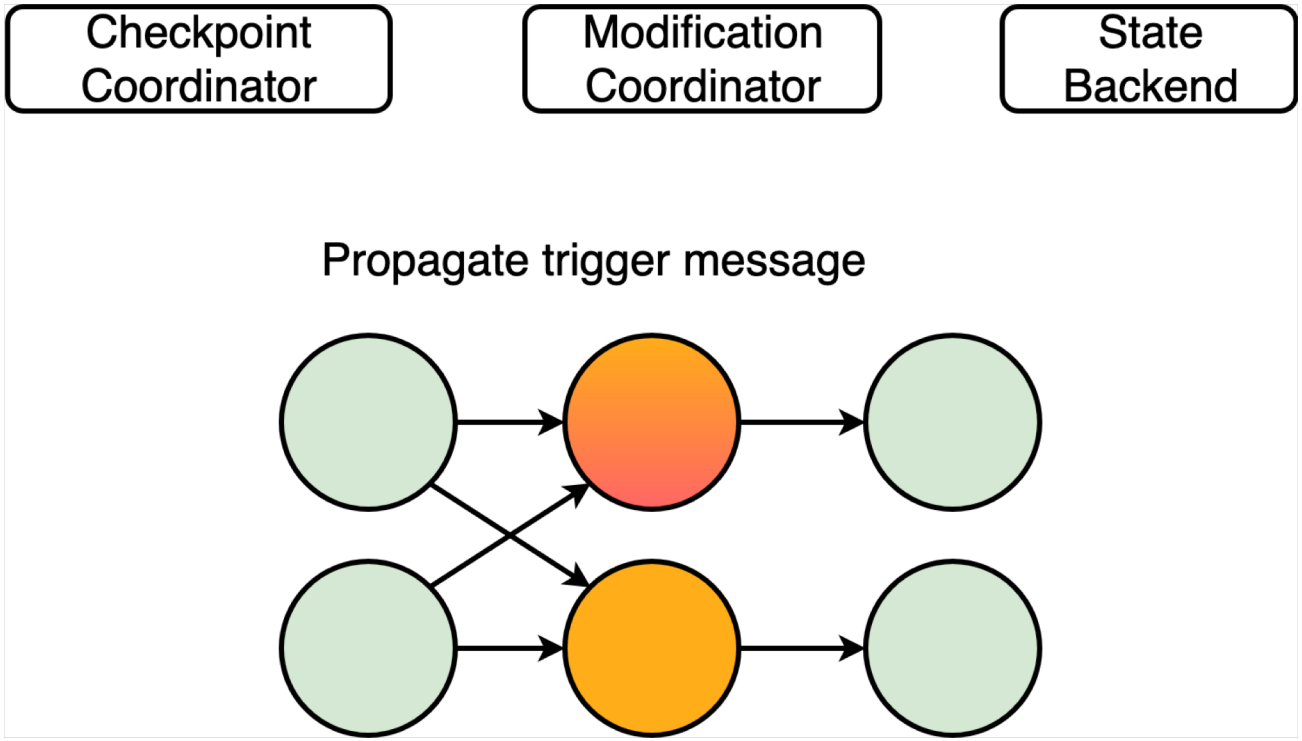
Preparing Migration – Waiting - Performing Migration

Migrating Operators



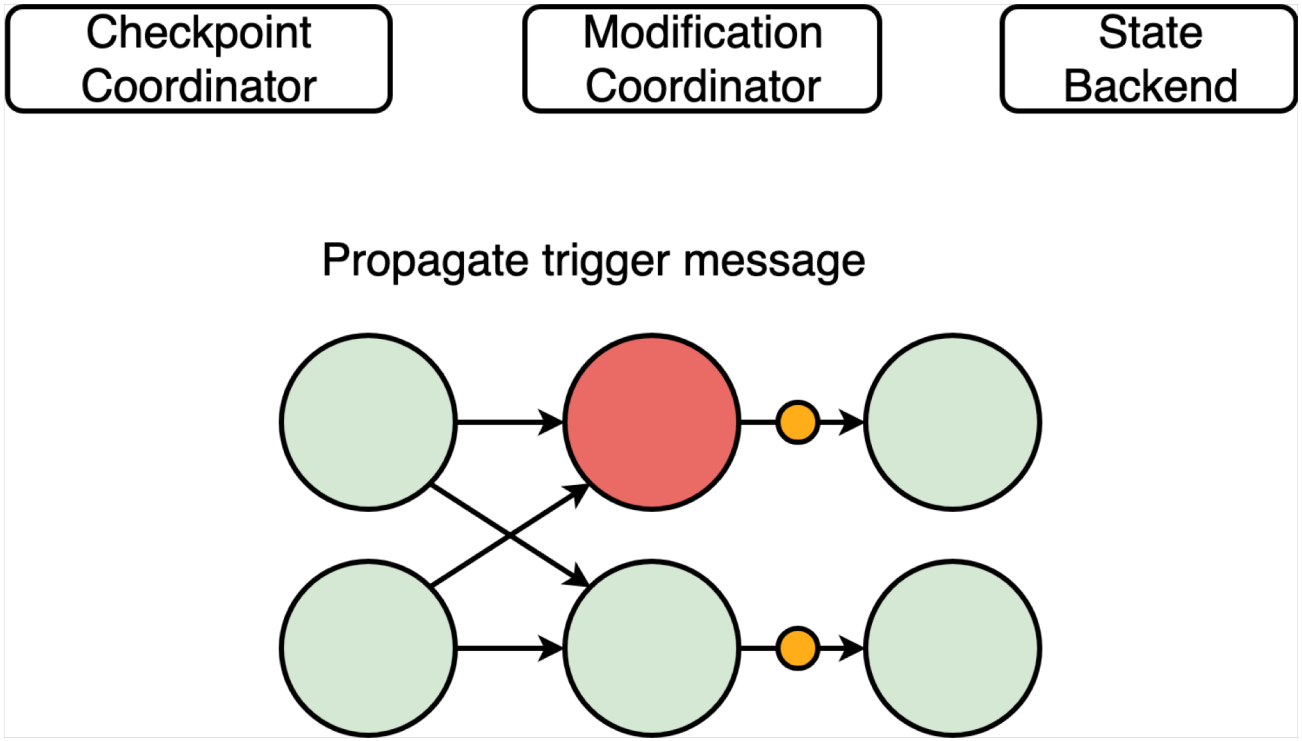
Preparing Migration – Waiting - Performing Migration

Migrating Operators



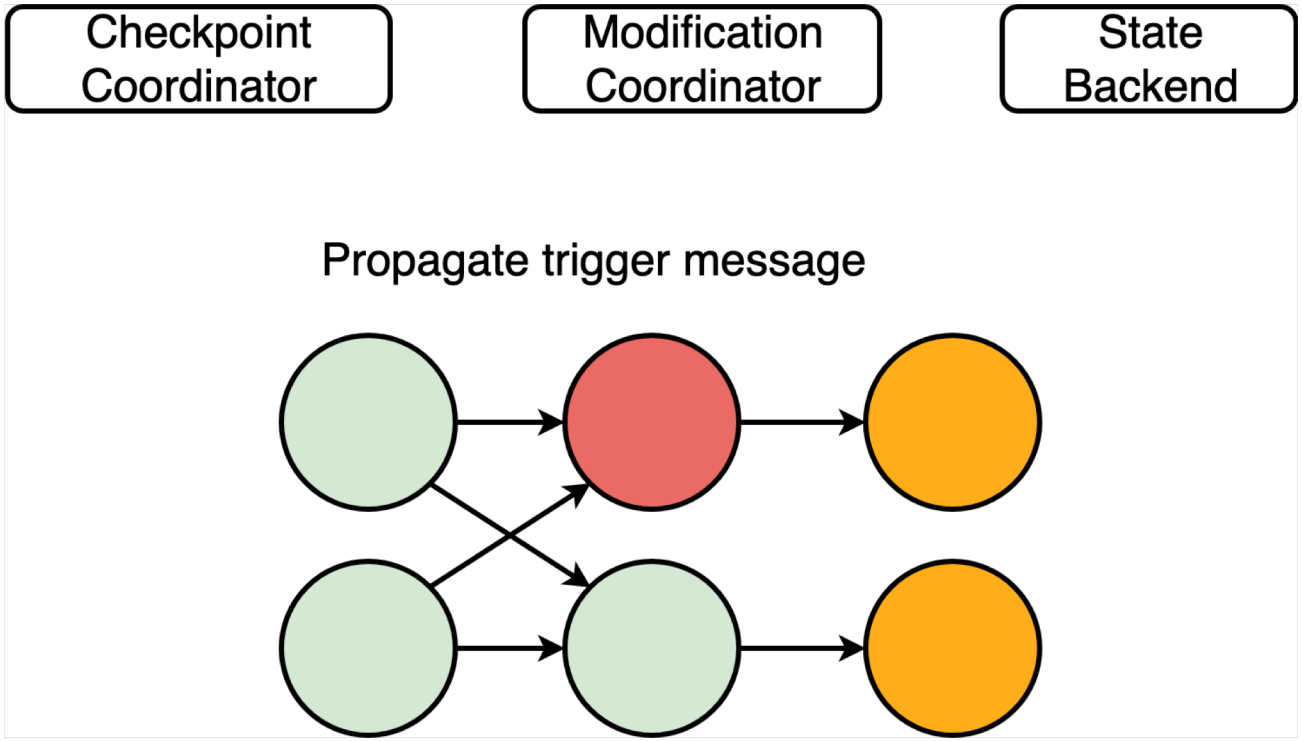
Preparing Migration – Waiting - Performing Migration

Migrating Operators



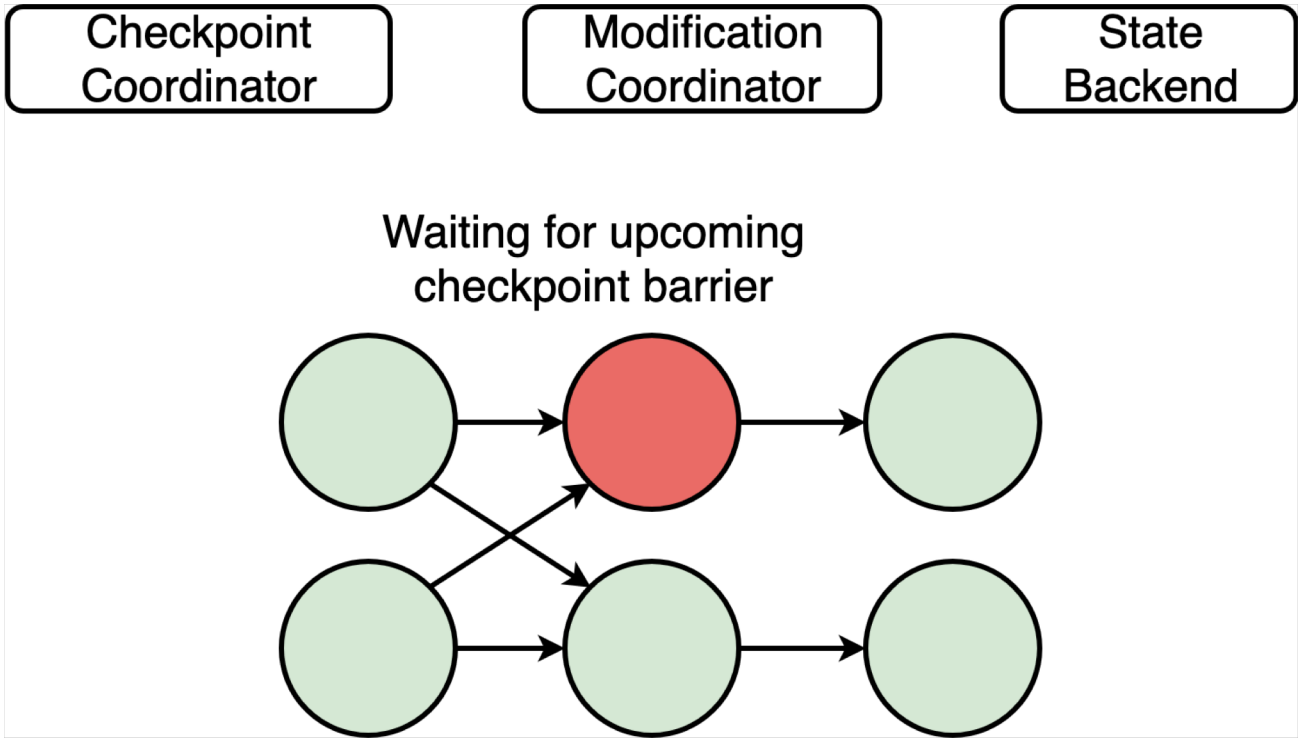
Preparing Migration – Waiting - Performing Migration

Migrating Operators



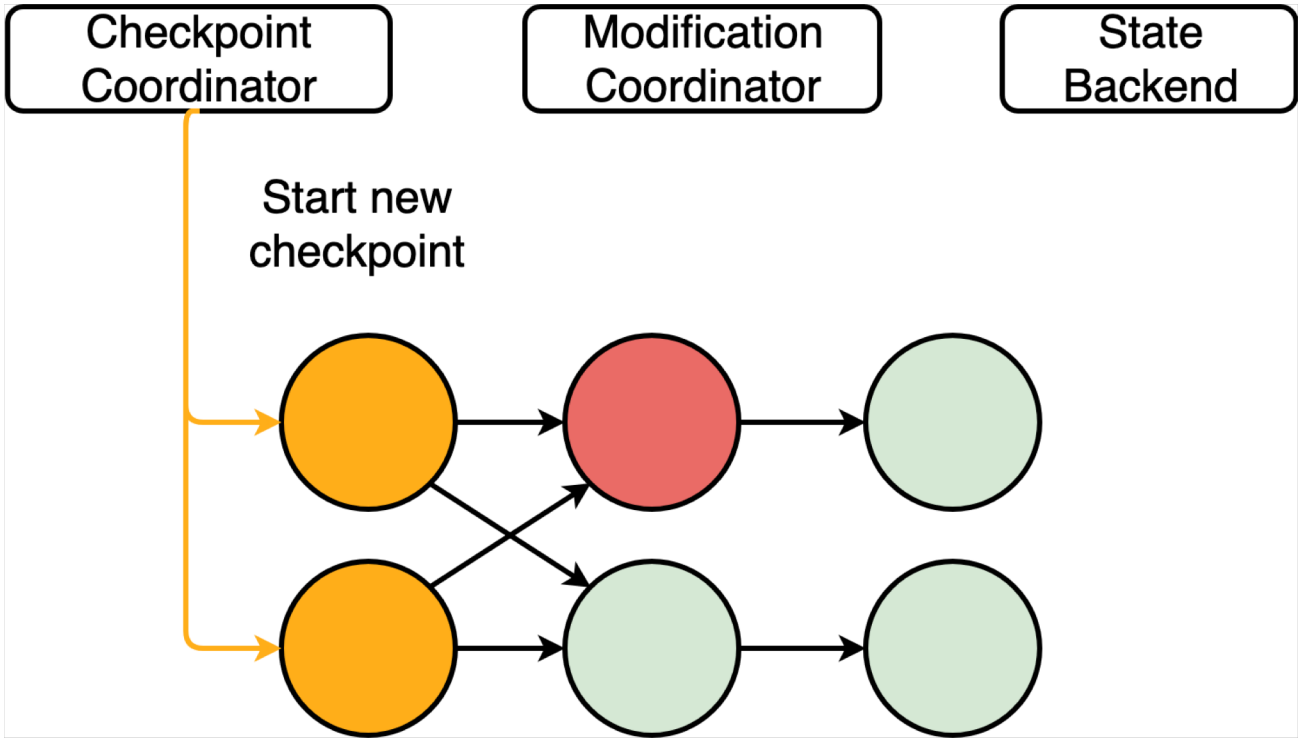
Preparing Migration – Waiting - Performing Migration

Migrating Operators



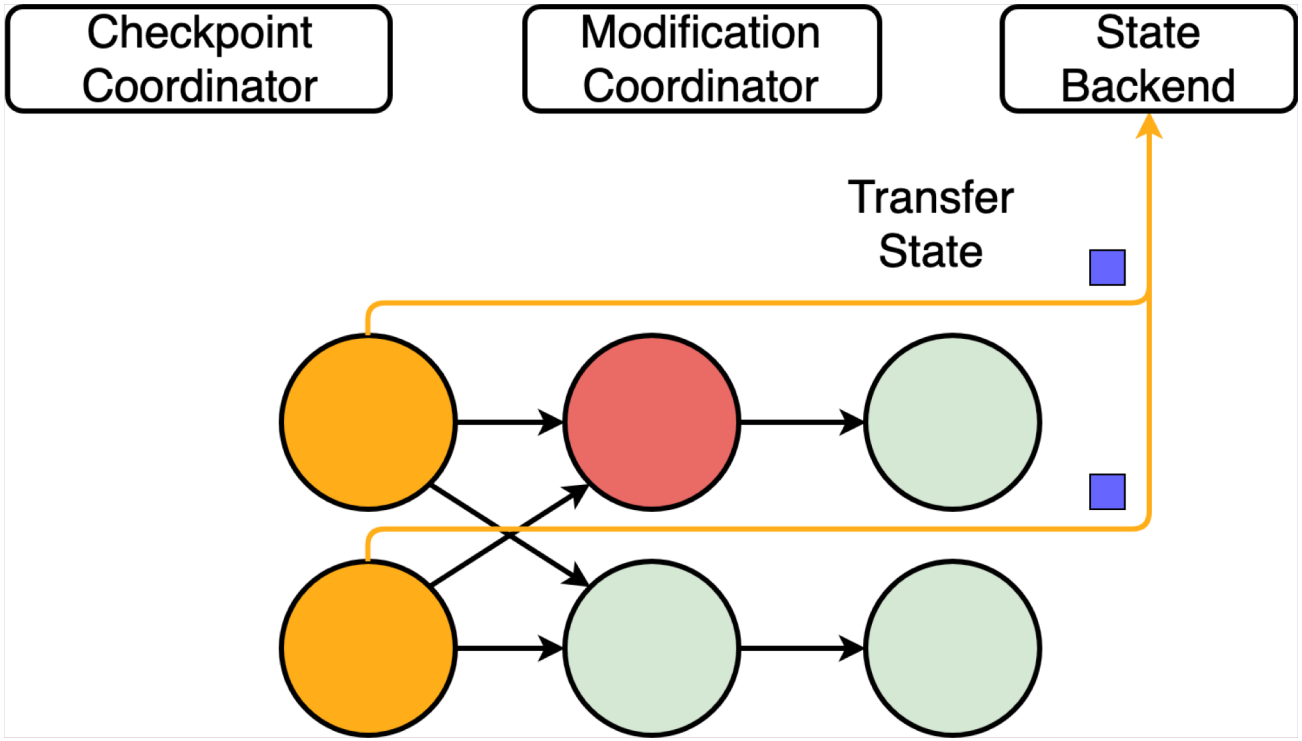
Preparing Migration – Waiting - Performing Migration

Migrating Operators



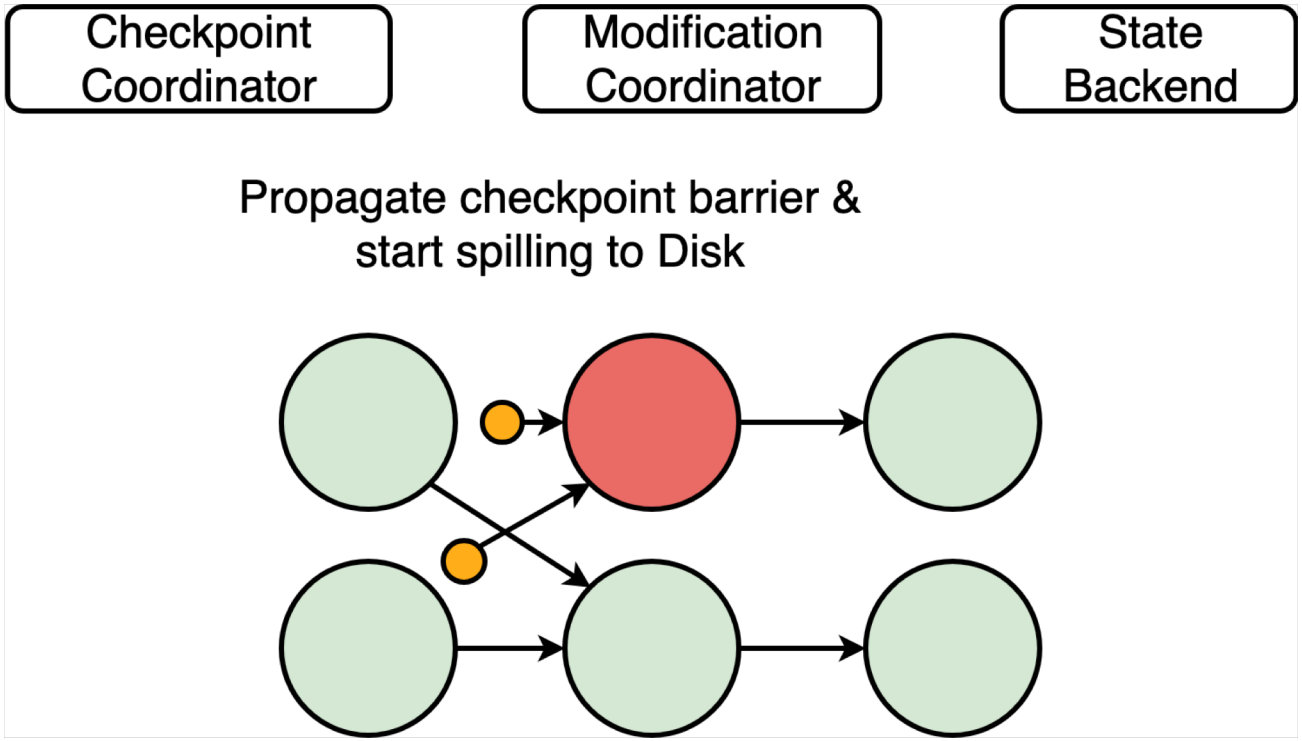
Preparing Migration – Waiting - Performing Migration

Migrating Operators



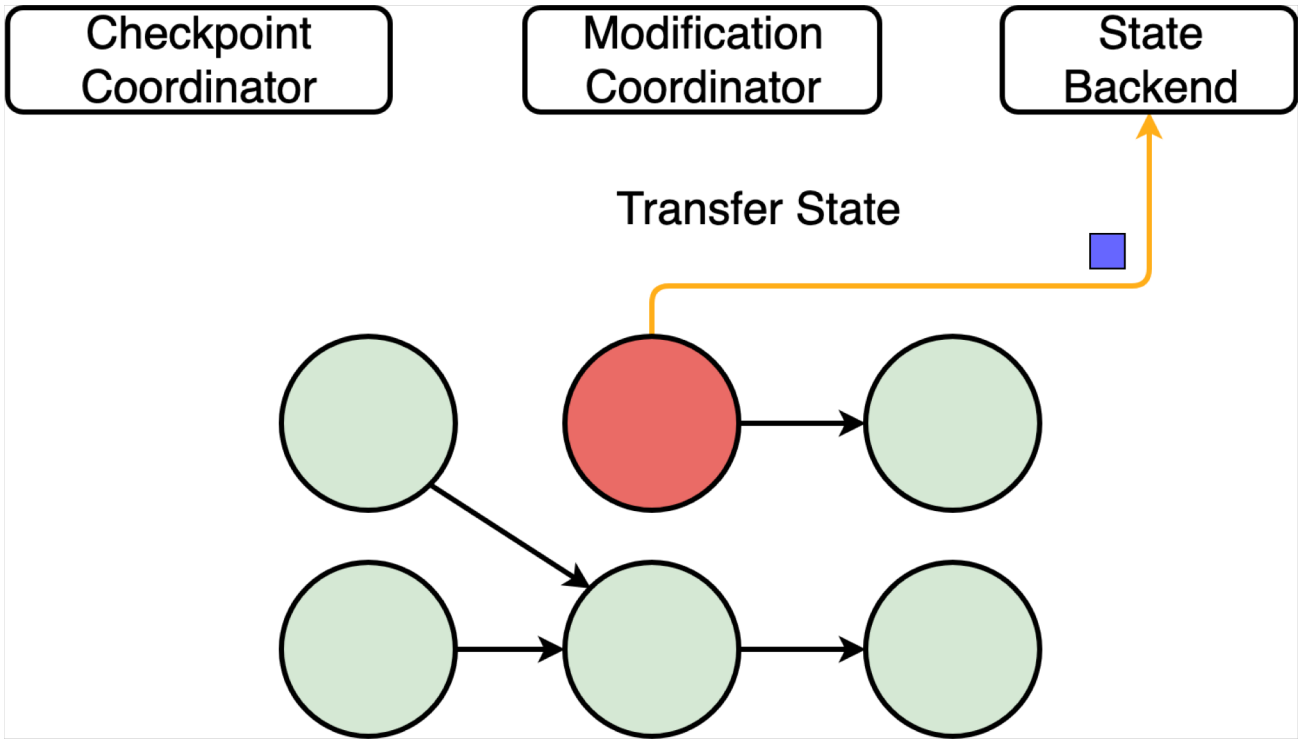
Preparing Migration – Waiting - Performing Migration

Migrating Operators



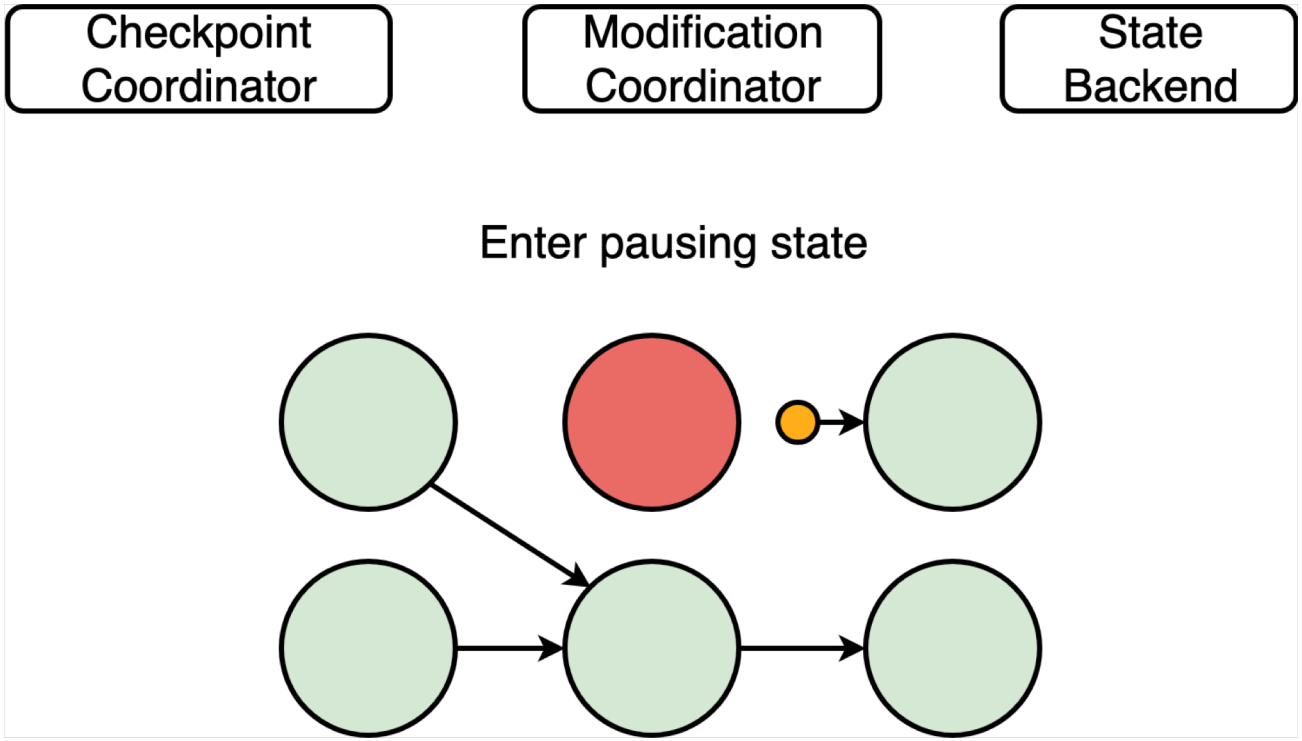
Preparing Migration – Waiting - Performing Migration

Migrating Operators



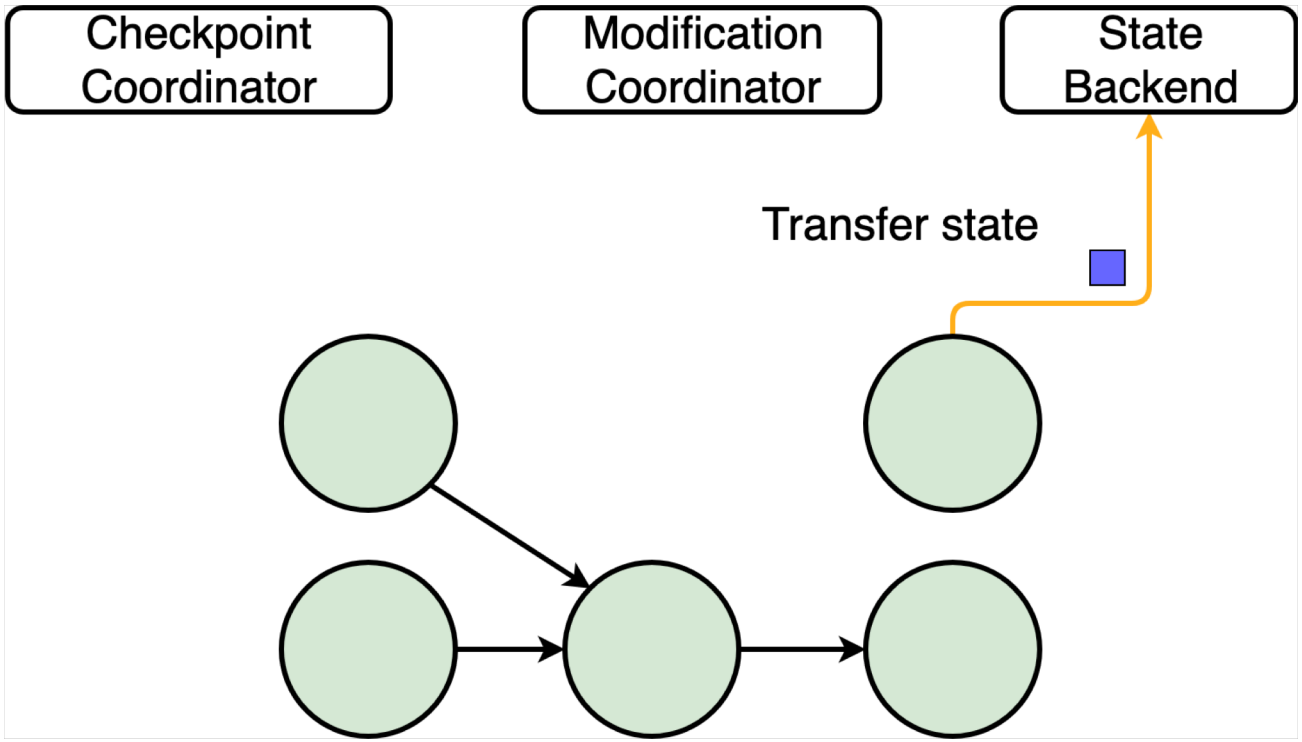
Preparing Migration – Waiting - Performing Migration

Migrating Operators



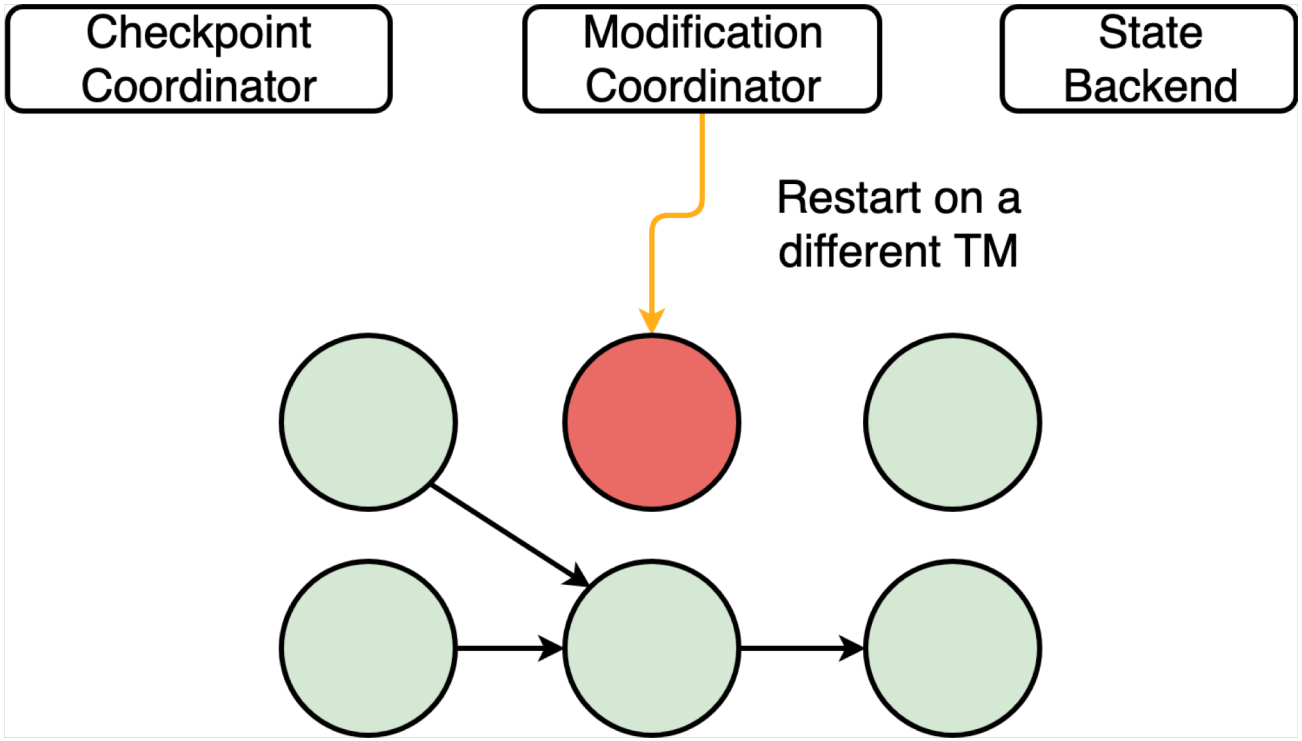
Preparing Migration – Waiting - Performing Migration

Migrating Operators



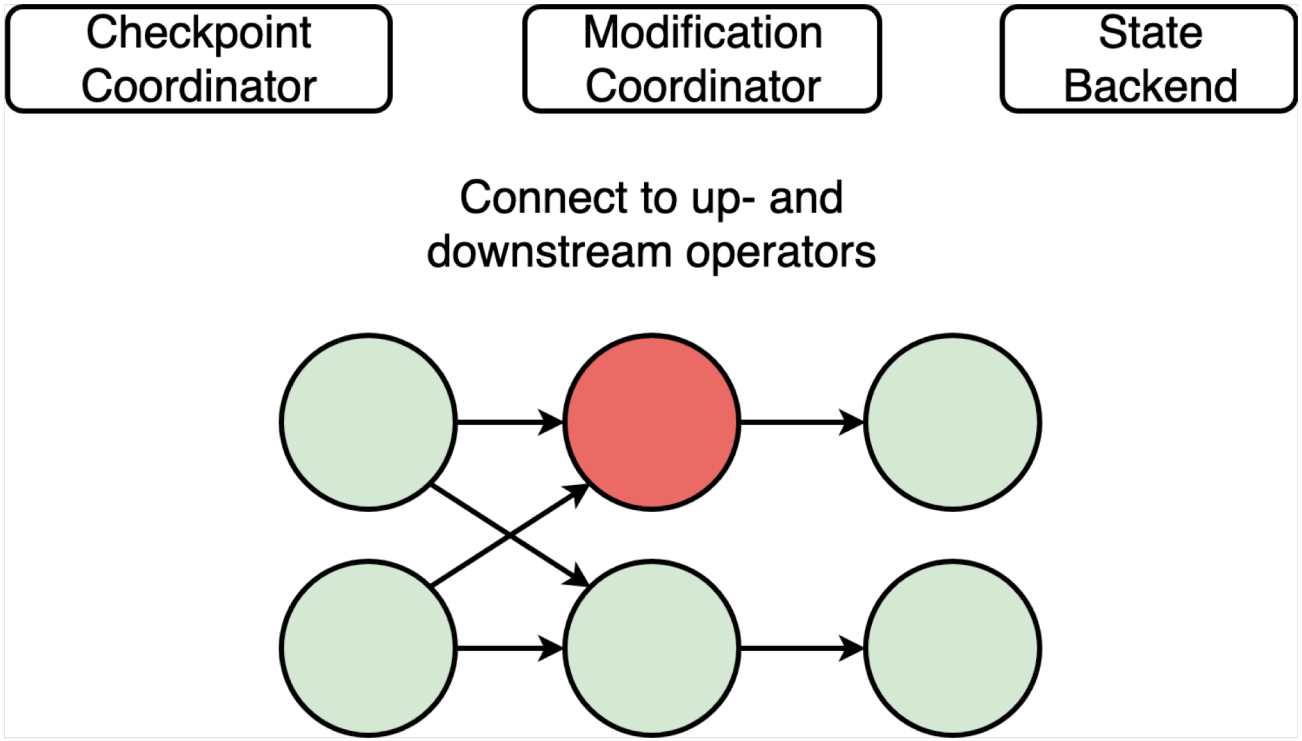
Preparing Migration – Waiting - Performing Migration

Migrating Operators



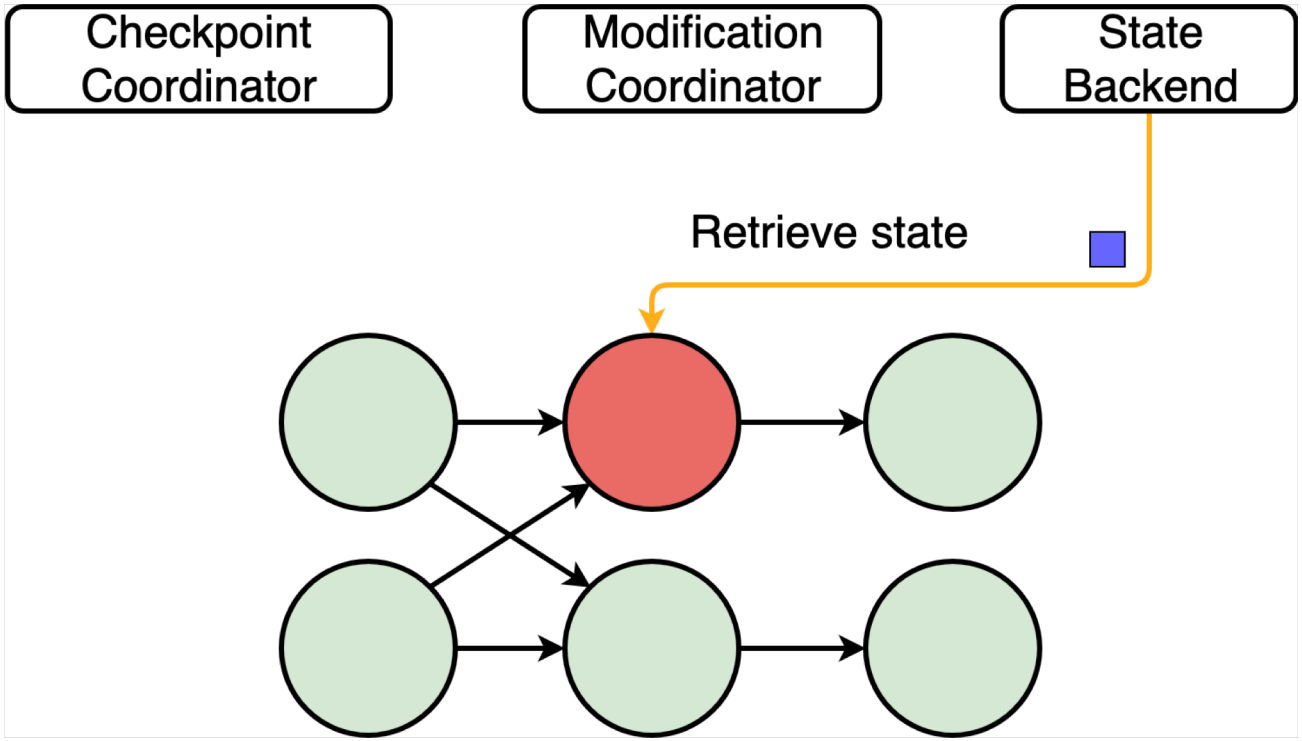
Preparing Migration – Waiting - Performing Migration

Migrating Operators



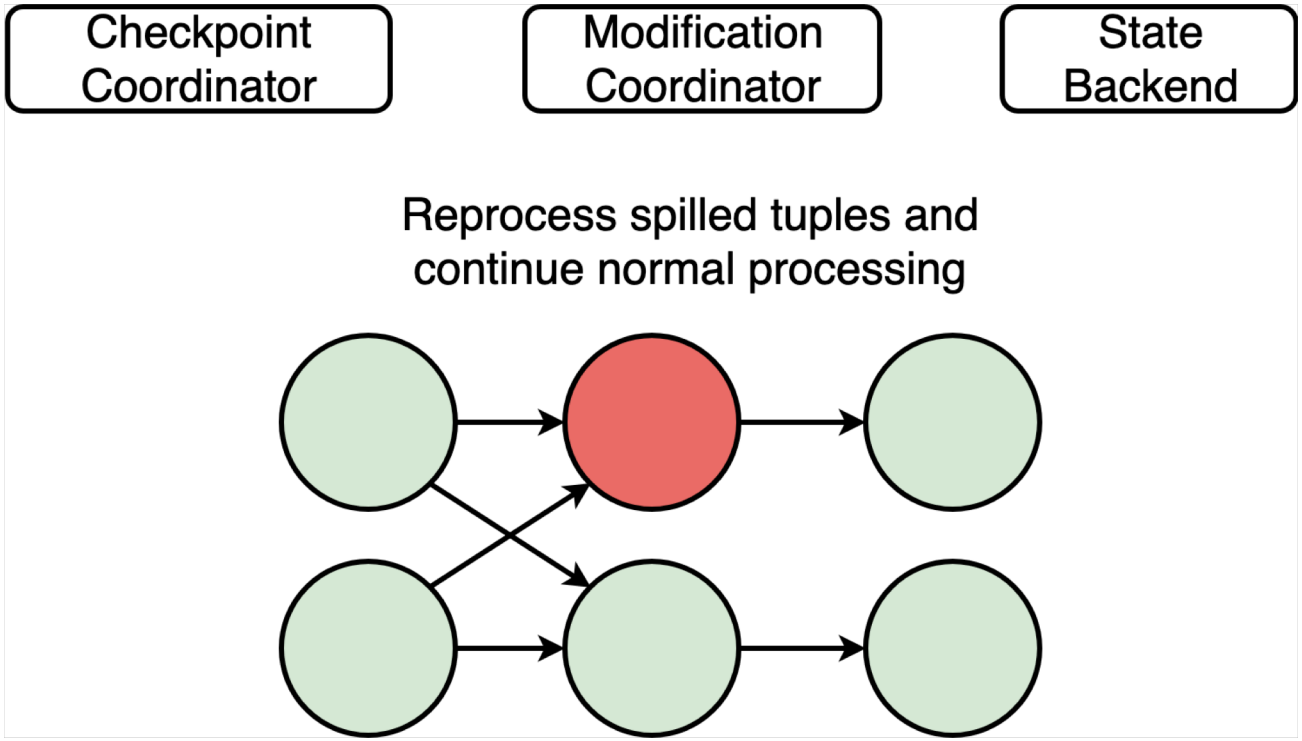
Preparing Migration – Waiting - Performing Migration

Migrating Operators



Preparing Migration – Waiting - Performing Migration

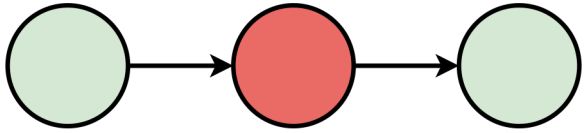
Migrating Operators



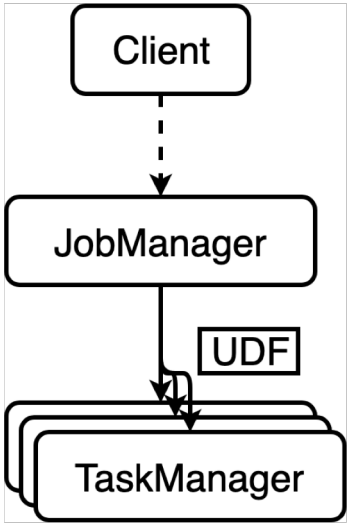
Preparing Migration – Waiting - Performing Migration

Changing Operator Function

- Motivation: Optimize UDF at runtime, e.g. based on runtime metrics

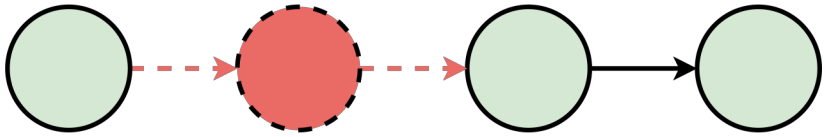


- New UDF distributed via JobManager to all TaskManagers
- Synchronization upcoming checkpoint barrier
  - All operators will replace their UDF

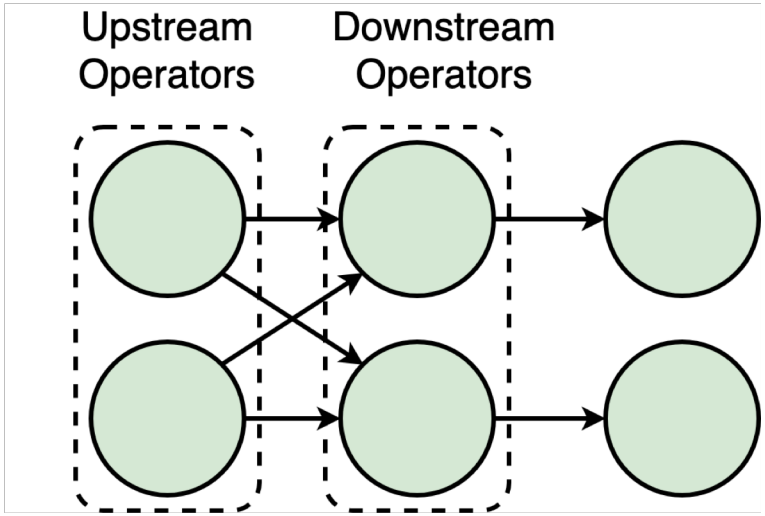


Introducing Operators

- Motivation: Modify data flow at runtime

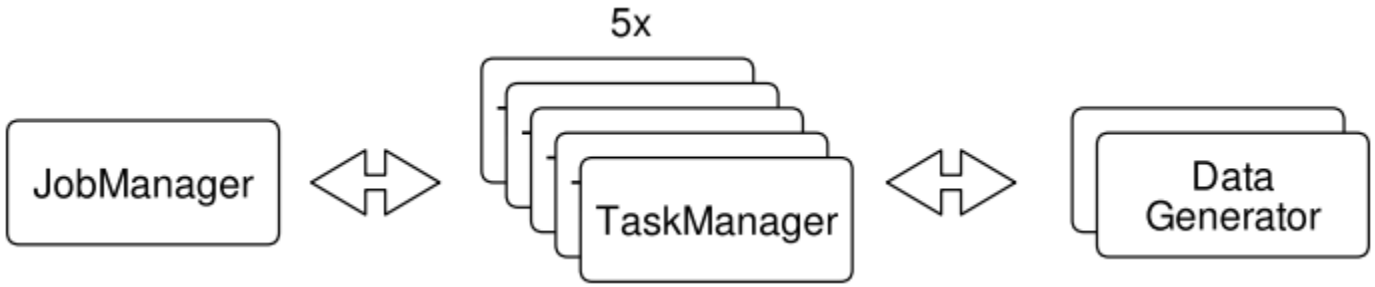


- Separation in up- and downstream operators



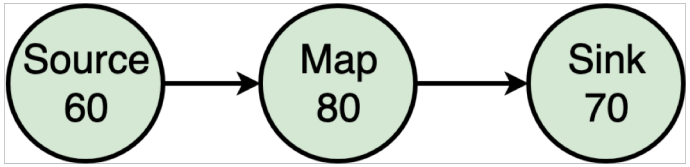
Benchmark

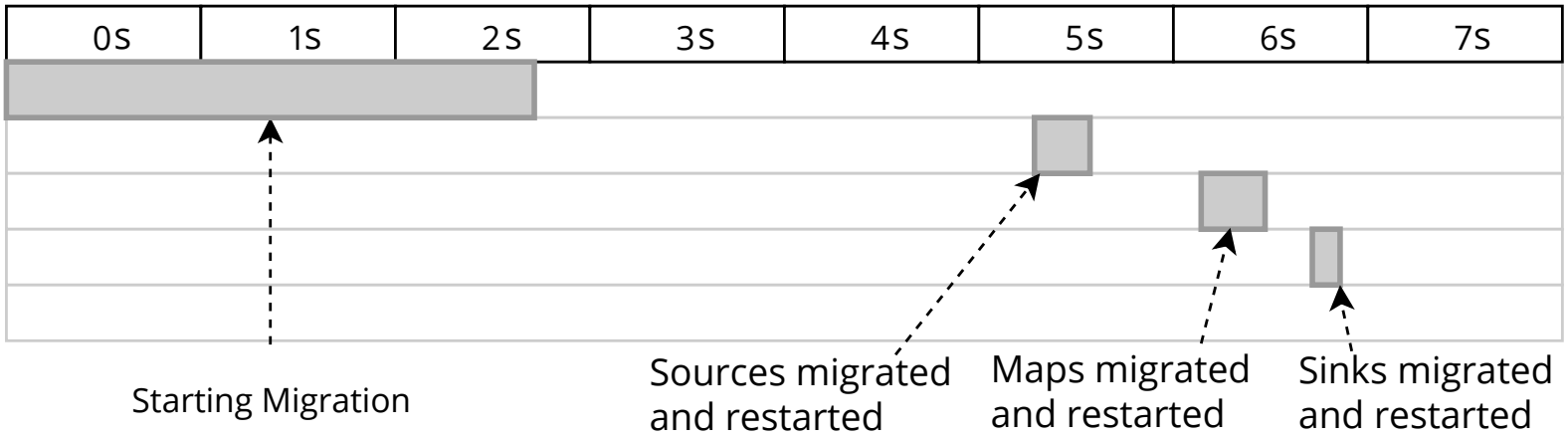
- All benchmarks performed on IBM cluster
  - 8 machines with 48 cores and 48 GB of memory each
  - 1 GB/s network between all machines
- Implemented custom data generator with constant load
- Warmup of 10 minutes for each benchmark



Migration – Small State - Stateful Map Query

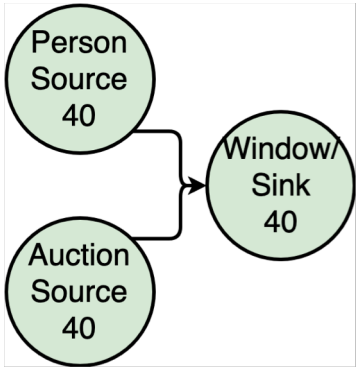
- Stateful Map Query covers operator migration with **small state**
  - 1 second checkpointing interval
  - Migrated 34 operator instances in less than 7.1 seconds with 17kB state size

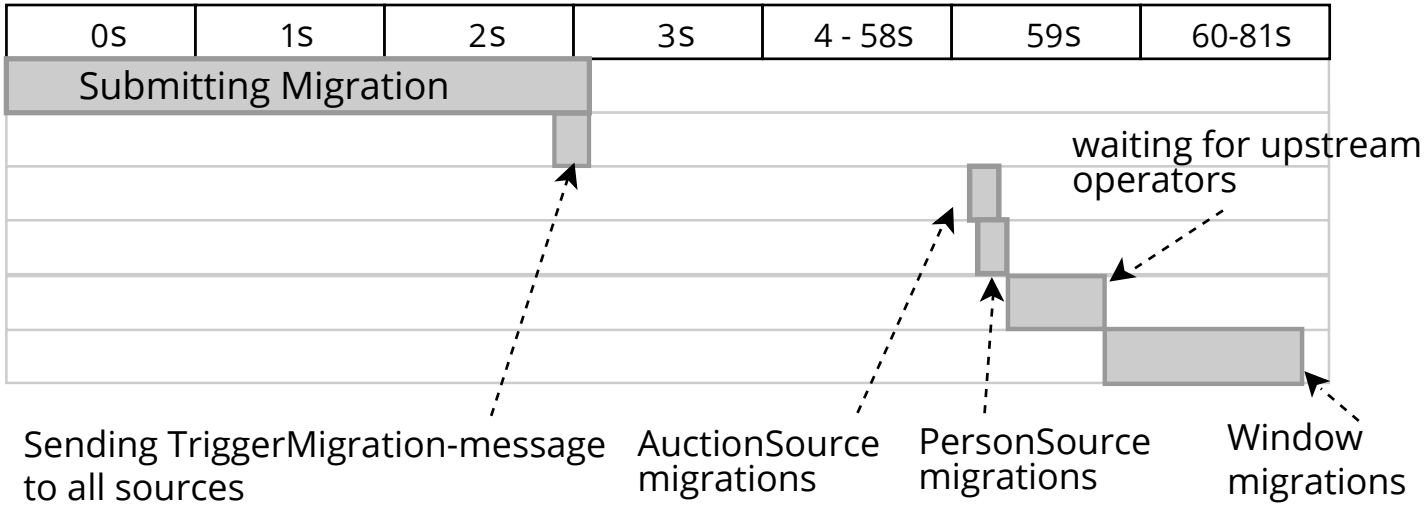




Migration – Large State - Nexmark Benchmark

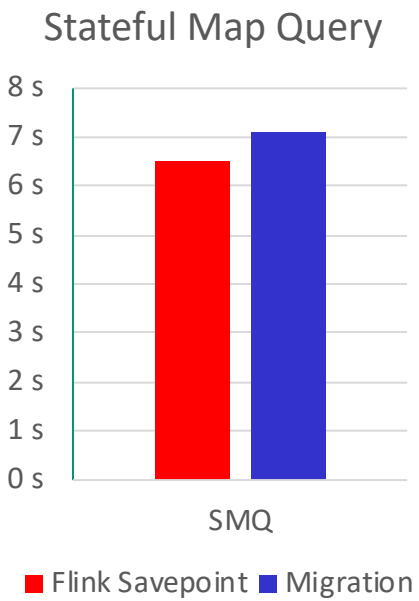
- Implemented Query 8 of the Nexmark Benchmark with **large operator state**
  - 30 second checkpointing interval
  - Migrated 2,7 GB operator state

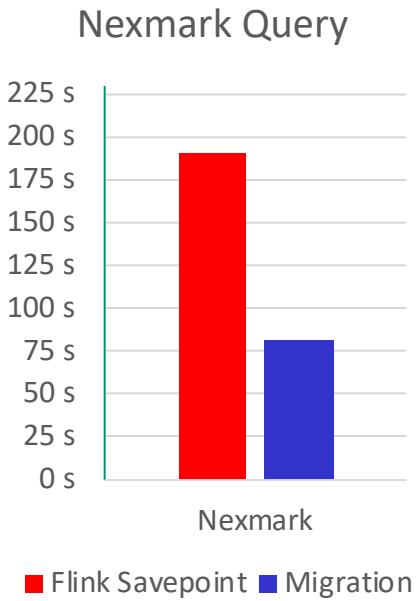




Migration - Benchmark Results

- Big performance improvements when migrating large state by a factor of more than 2.3
- No data loss between restarts with migration

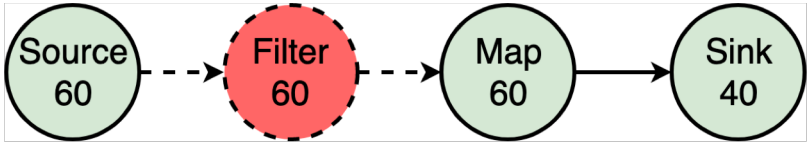


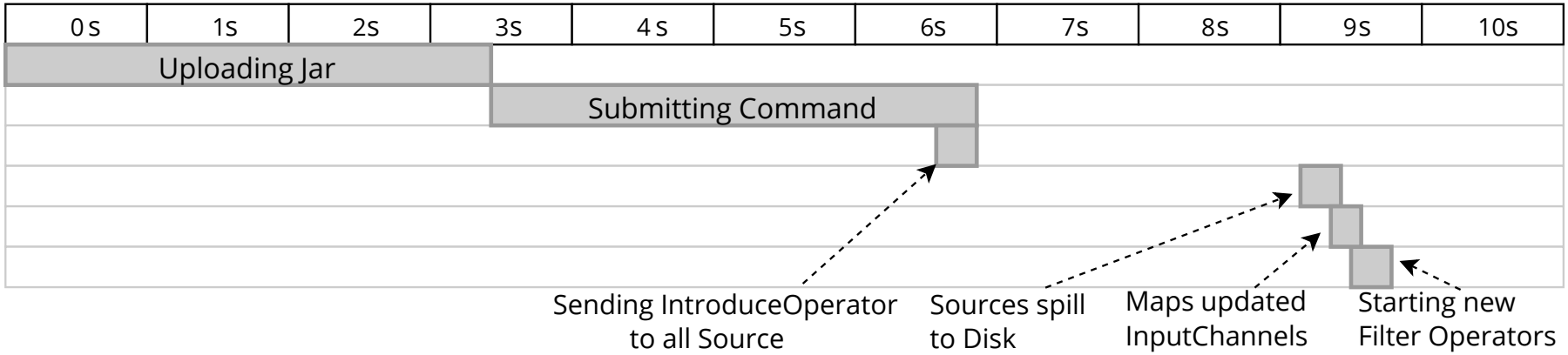


- Most time is not spent on actual migration
  - submitting migration
  - waiting for checkpoint barriers

Benchmark – Introduce Operator

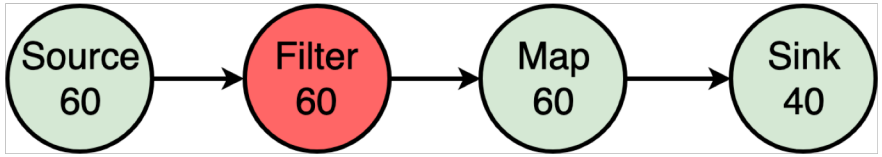
- Introduce new filter operator between source and map operators
  - 1 second checkpointing interval

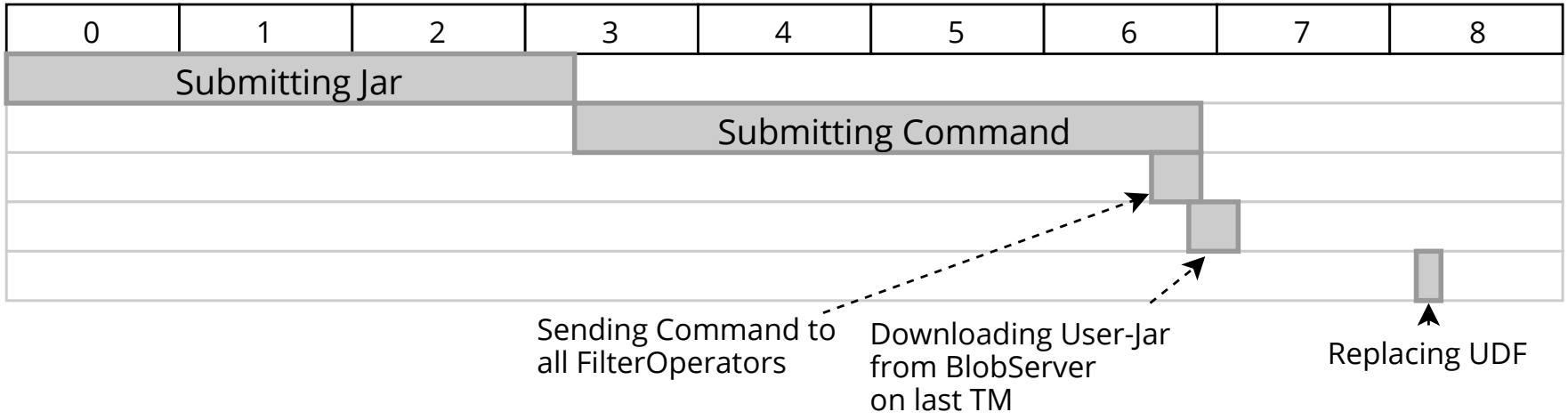




Benchmark – Replace UDF

- Change filter condition for an existing filter operator
  - 1 second checkpointing interval





Conclusion

- Mechanism to modify jobs at runtime
  - Modifications independent of actual operator
  - Prevents data loss during restart without persistent message queue
  - Implemented three modification use cases
    - Operator migration
    - Introduction of new operators
    - Replace UDF
- Benchmark results
  - Migrating operators is almost as fast as Flink's savepoints
  - For large state 2.3x faster
  - Actual Modifications take less than a few seconds

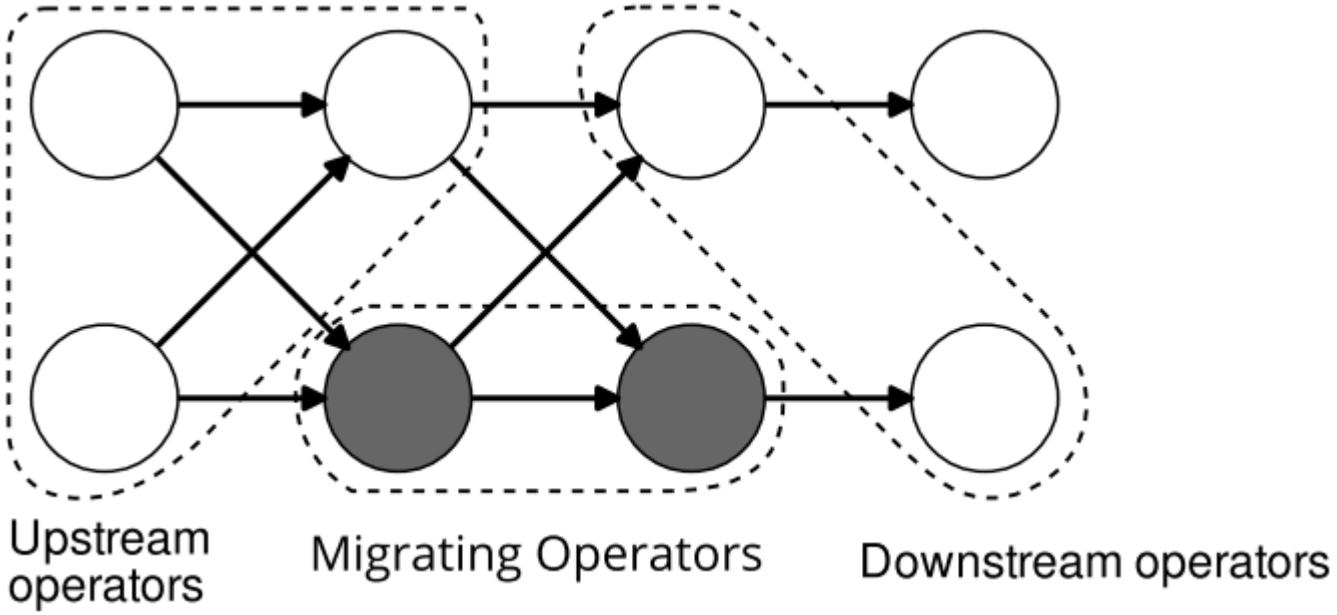
- Solid foundation for dynamic runtime optimizations

Backup Slides

Migrating Operators

- Real-life jobs require migration of multiple, successive operator instances

- Main insights
  - Upstream operator must still guarantee data safety
  - It does not matter if upstream operator is spilling to disk or migrating itself



Migrating Operators

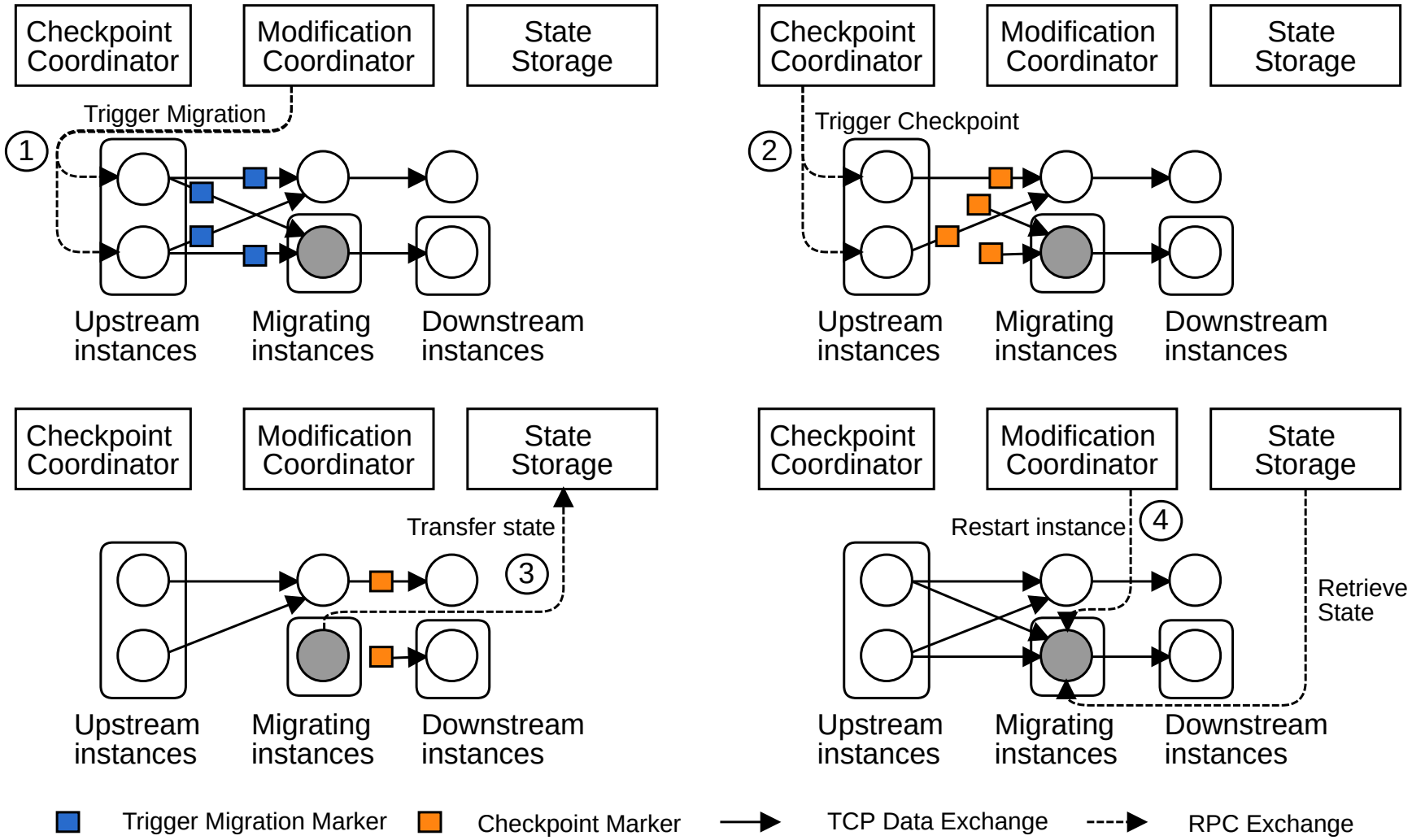
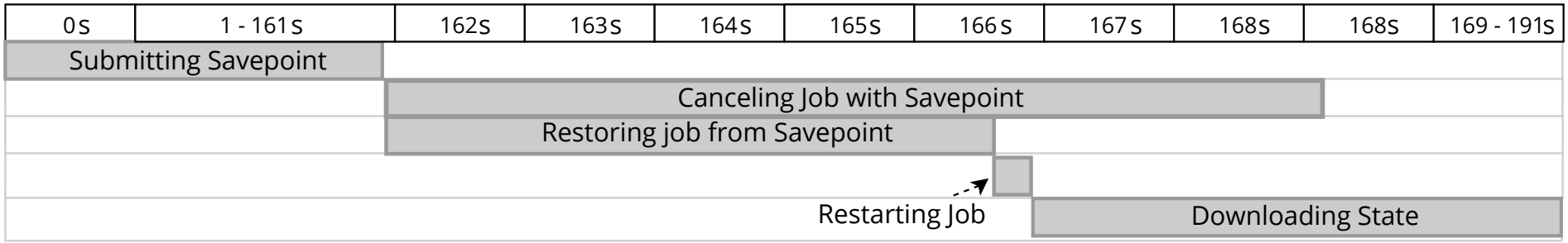


Image sources

- Big data: https://commons.wikimedia.org/wiki/File:BigData_2267x1146_white.png
- Flink & logos: https://flink.apache.org/poweredby.html
- Graph icon made by Smashicons from www.flaticon.com is licensed by CC 3.0 BY

Nexmark – Canceling with Savepoint



Benchmark Migration Results

Benchmark	Migrated state size	Waiting for Synchronization	Migration duration	Overall duration	Migrated Operators
Stateful Map Query	17 kB	2 s	2.4 s	7.1 s	34
Nexmark	2,7 GB	54.9 s	23.7 s	81.8 s	24

- Big performance improvements when migrating large state by a factor of more than 2.3
- No data loss between restarts with migration
- Most time is not spent on actual migration
  - submitting migration
  - waiting for checkpoint barriers

Flink Overview