Apache Flink Streaming

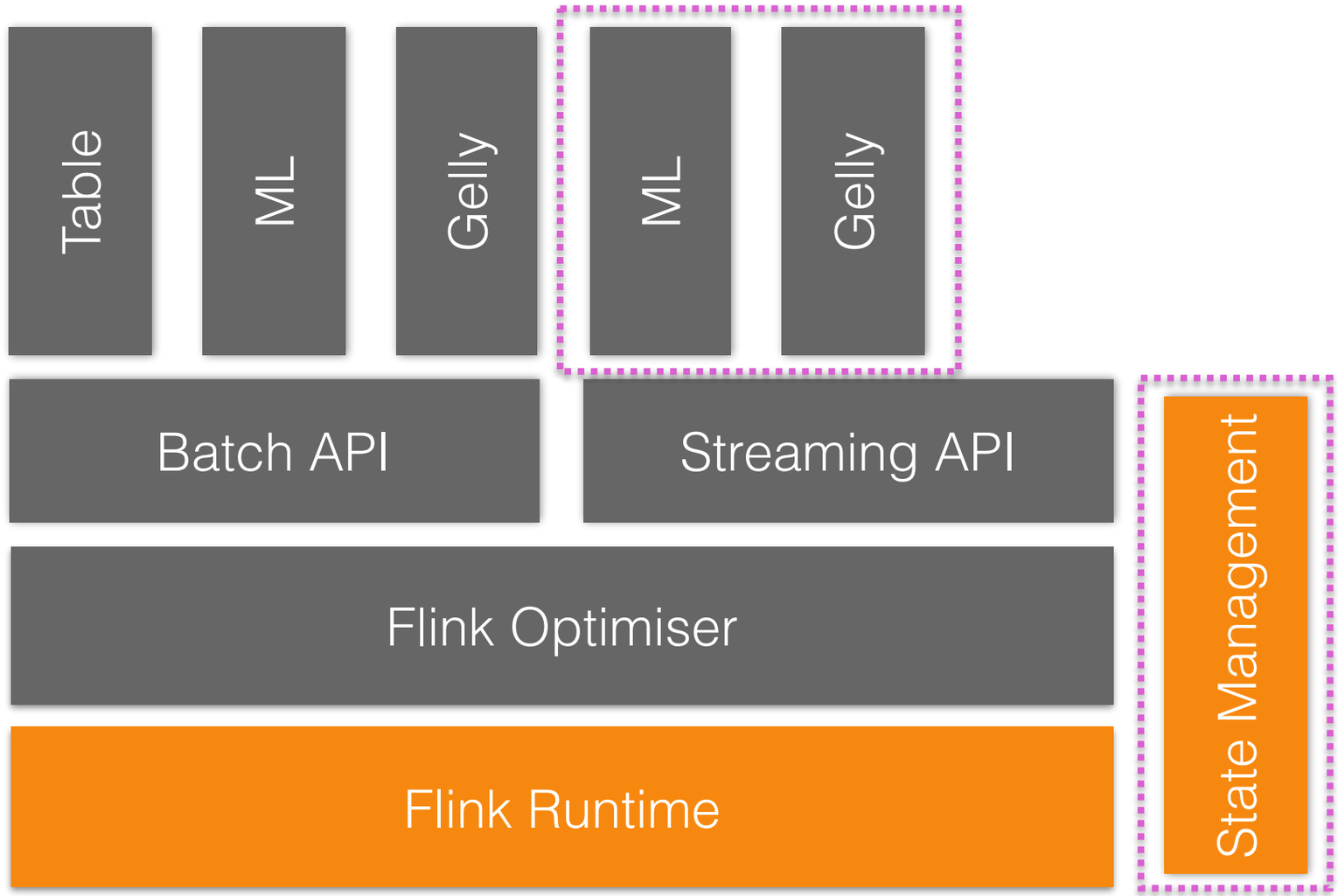
Resiliency and Consistency

Paris Carbone - PhD Candidate
KTH Royal Institute of Technology
<parisc@kth.se, senorcarbone@apache.org>

Overview

- The Flink Execution Engine Architecture
- Exactly-once-processing challenges
- Using Snapshots for Recovery
- The ABS Algorithm for DAGs and Cyclic Dataflows
- Recovery, Cost and Performance
- Job Manager High Availability

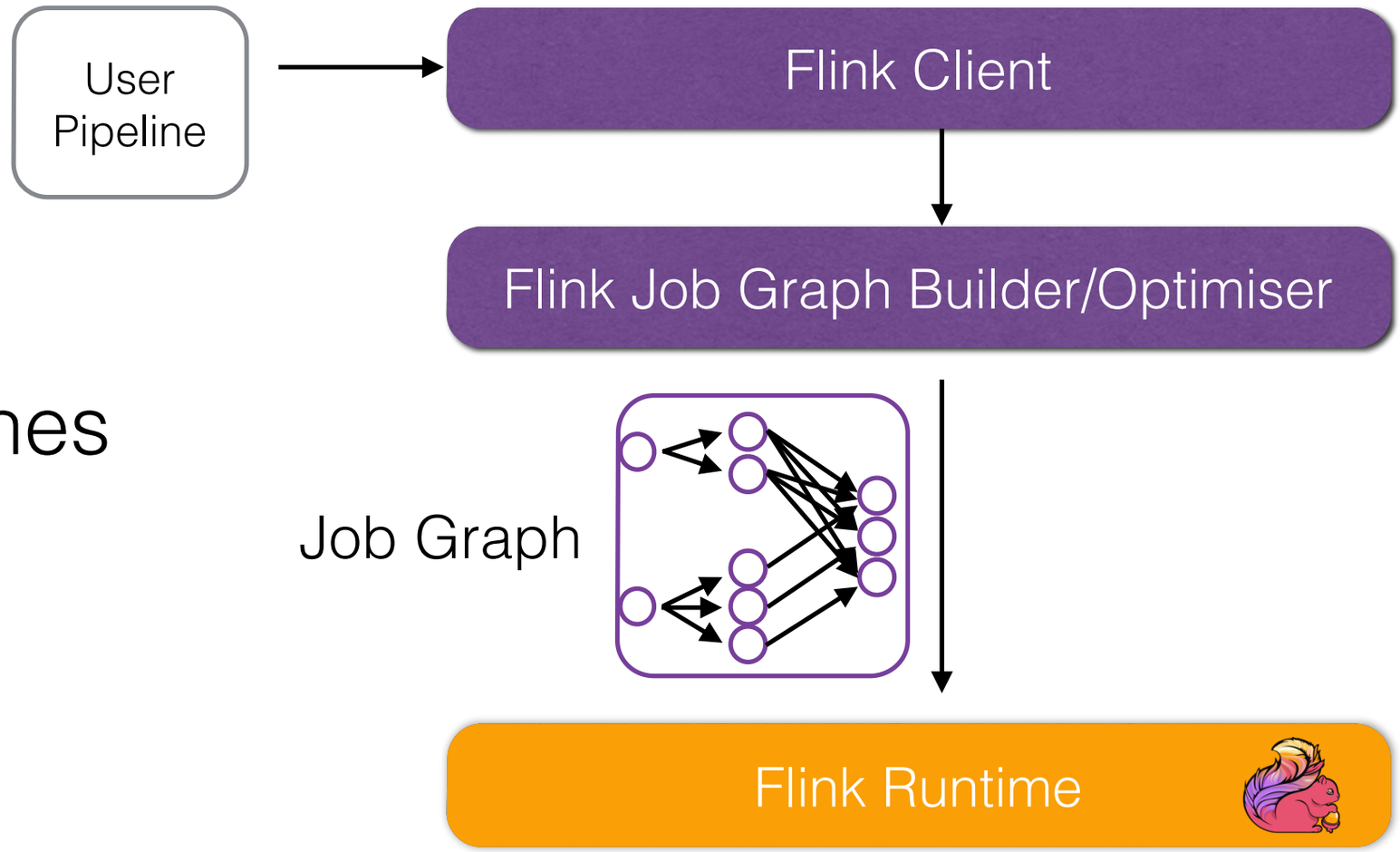
Current Focus



Executing Pipelines

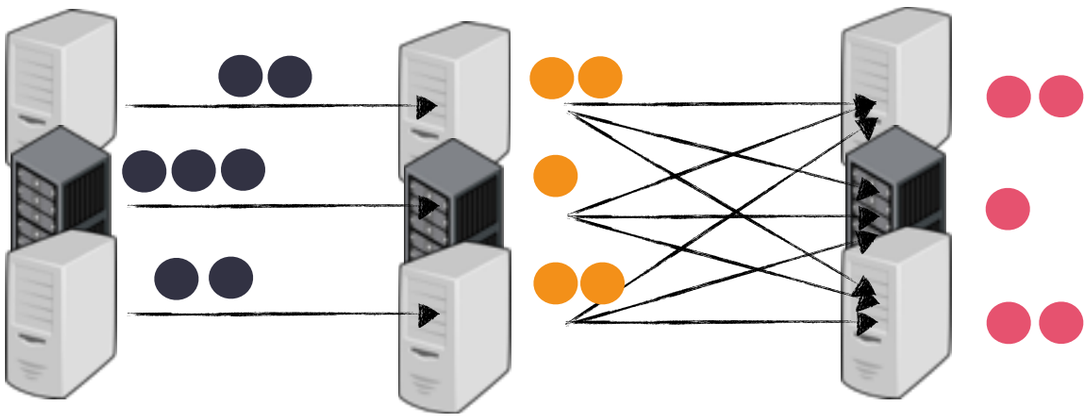
```
val lines: DataStream[String] = env.fromSocketStream(...)
lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
     .groupBy("word").sum("frequency")
     .print()
```

- Streaming pipelines translate into **job graphs**

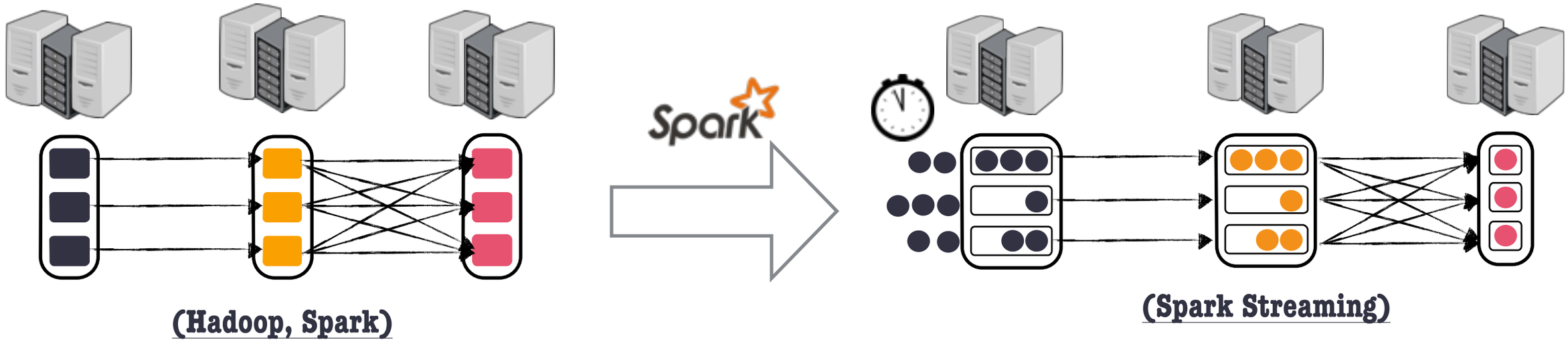


Unbounded Data Processing Architectures

1) Streaming (Distributed Data Flow)



2) Micro-Batch

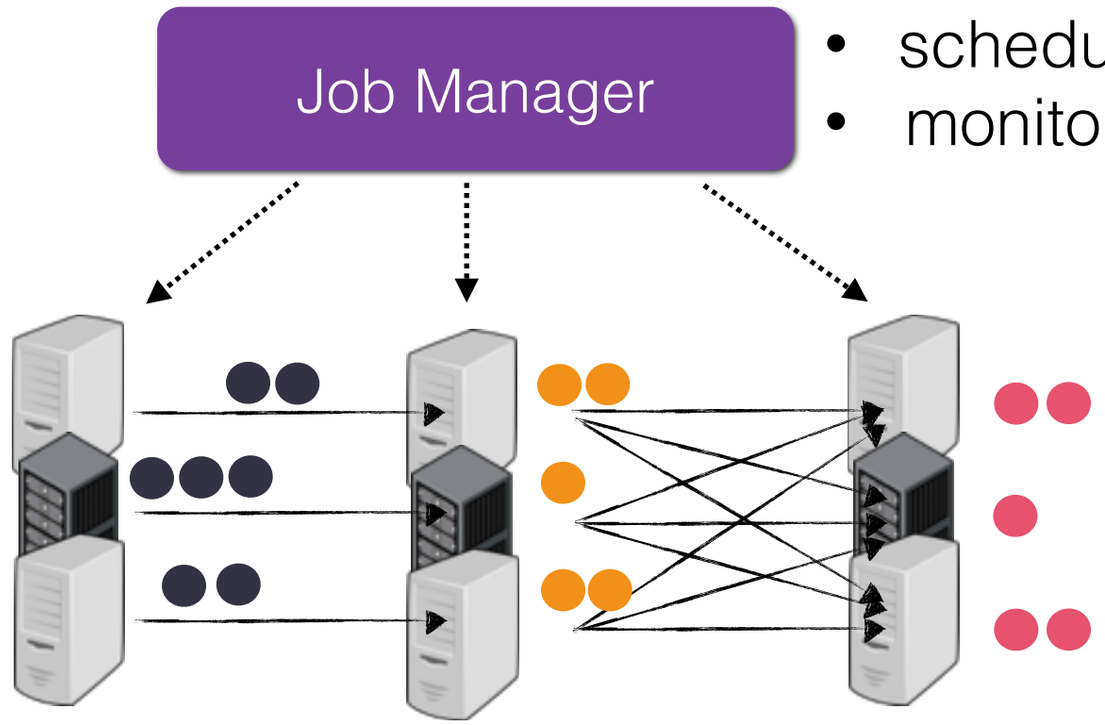


The Flink Runtime Engine

- **Tasks** run operator logic in a pipelined fashion

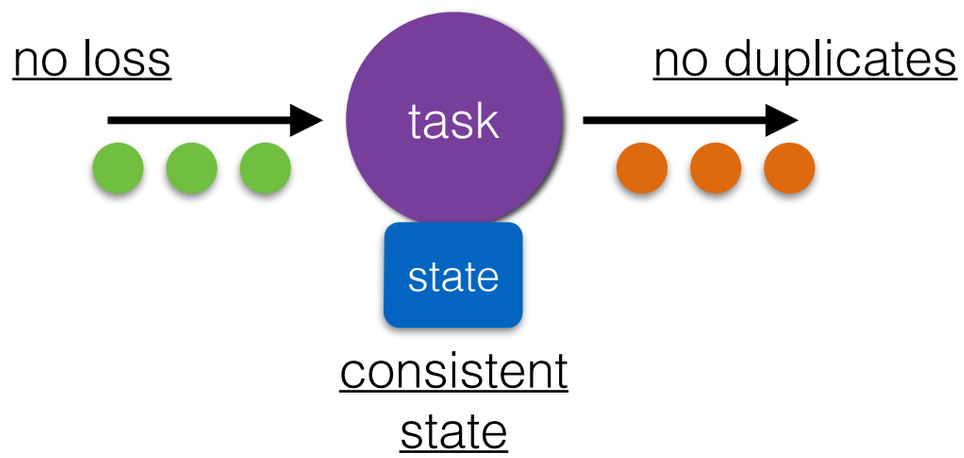
- They are scheduled among workers

- **State** is kept within tasks



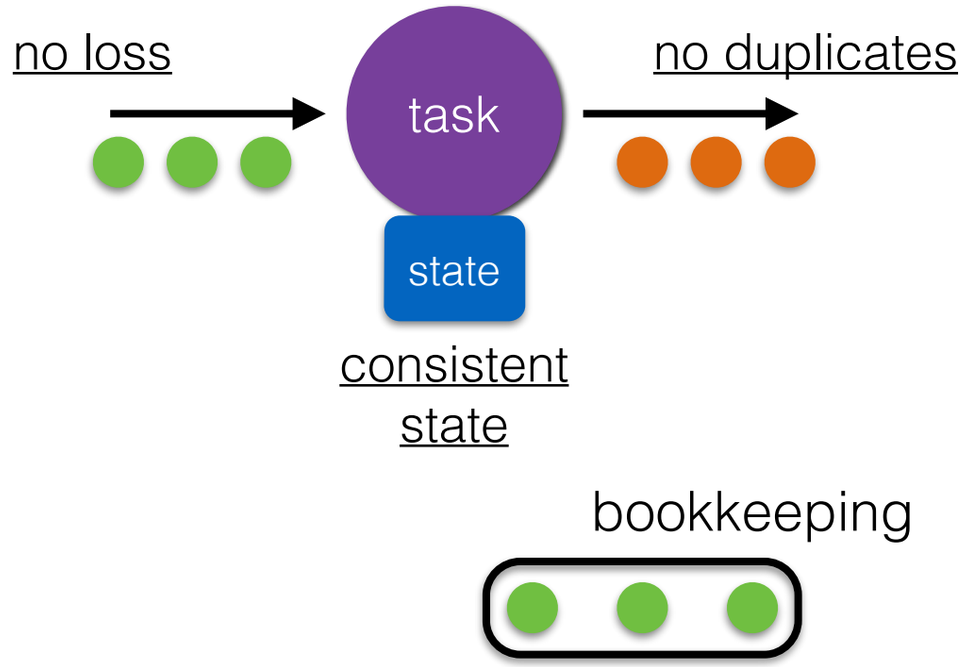
Task Failures

- Task failures are **guaranteed** to occur
- We need to make them **transparent**
- Can we simply recover a task from scratch?



Record Acknowledgements

- We can monitor the consumption of every event



- It guarantees no loss

- Duplicates and inconsistent state are not handled

Atomic transactions per update

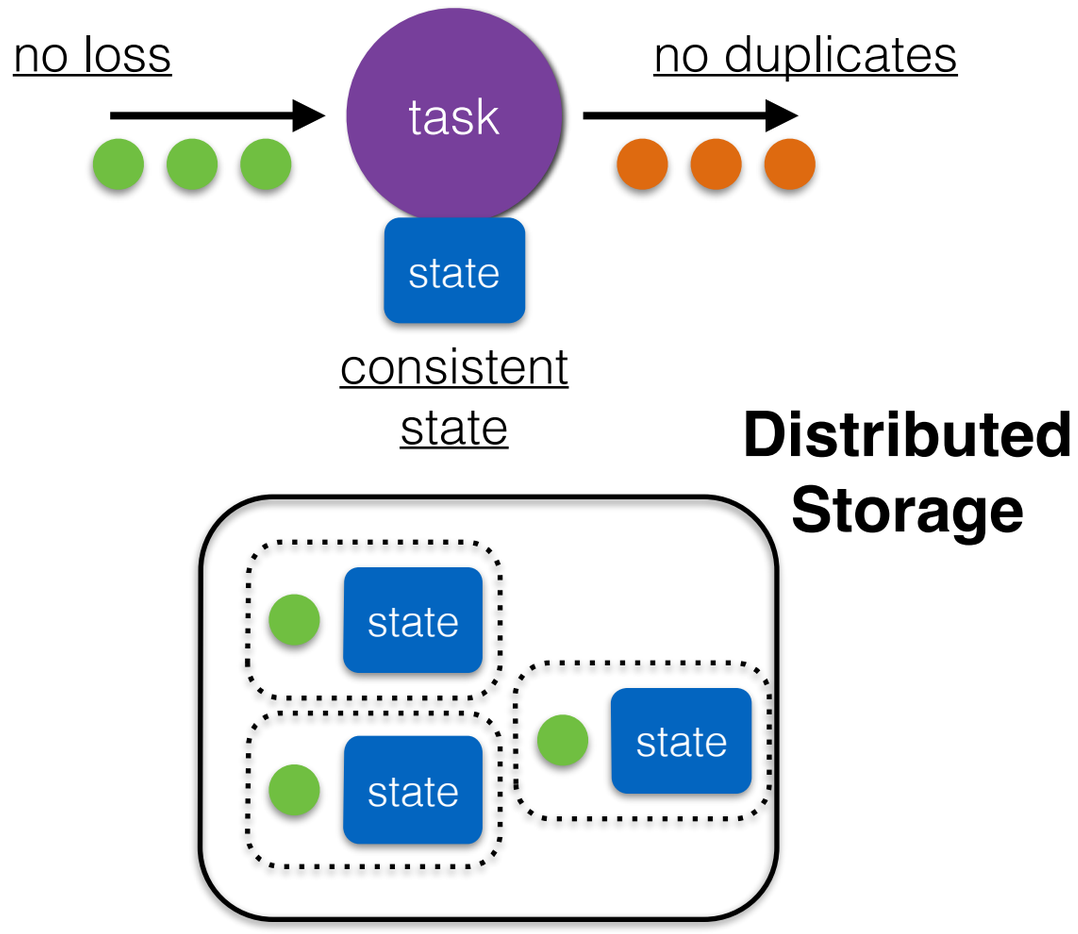
- It guarantees <u>no loss and consistent state.</u>

- No duplication can be achieved e.g. with bloom filters or caching

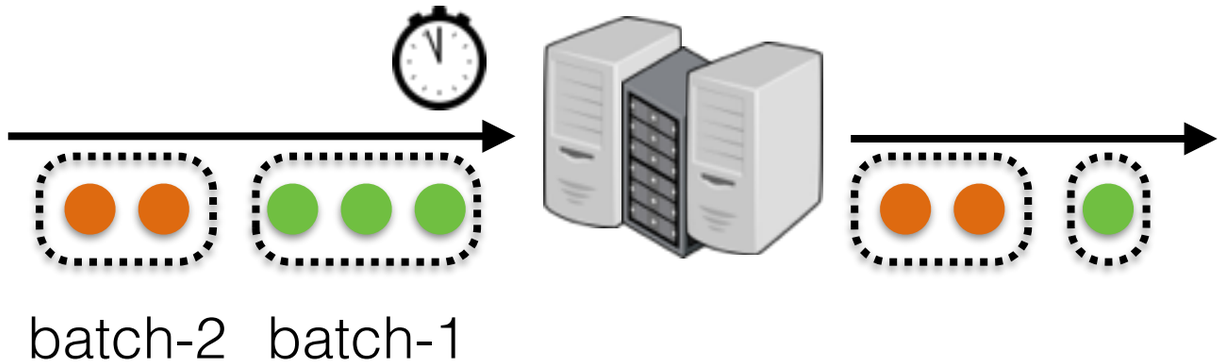
- Fine grained recovery

- Non-constant load in external storage can lead to unsustainable execution

Trident



Lessons Learned from Batch



- If a batch computation fails, simply repeat computation as a transaction
- Transaction rate is **constant**
- Can we apply these principles to a true streaming execution?

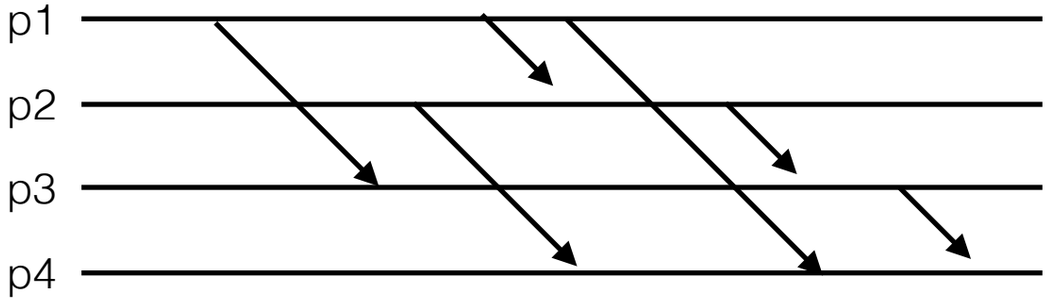
Distributed Snapshots

Distributed Snapshots

“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”

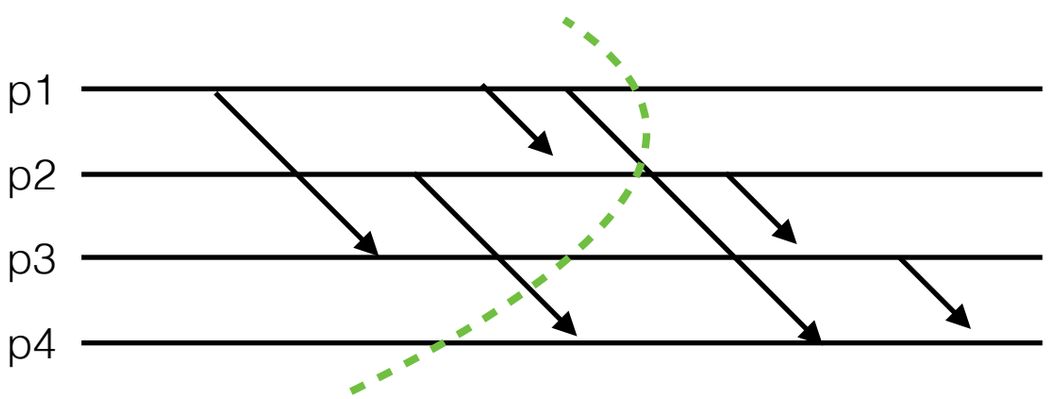
Distributed Snapshots

“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”



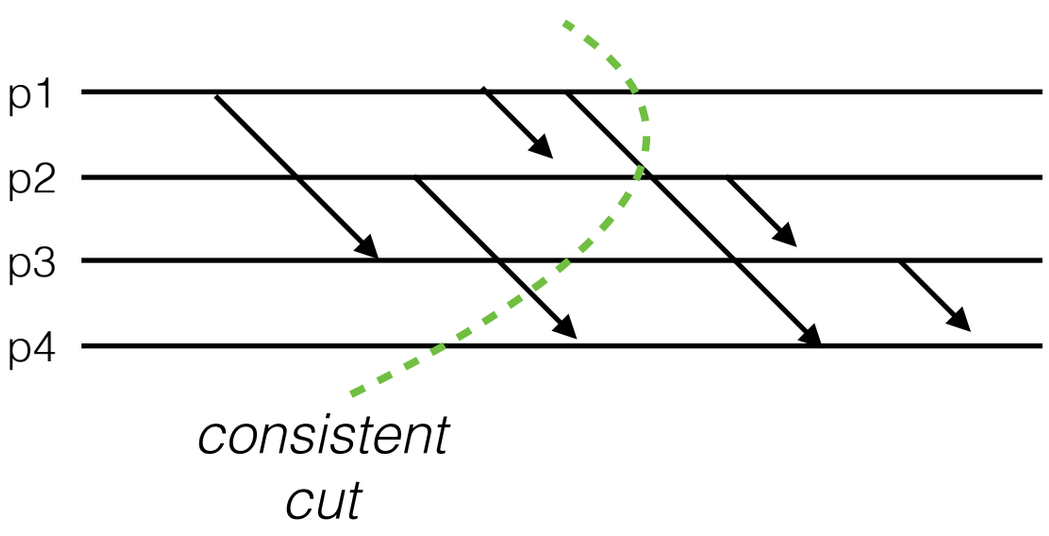
Distributed Snapshots

“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”



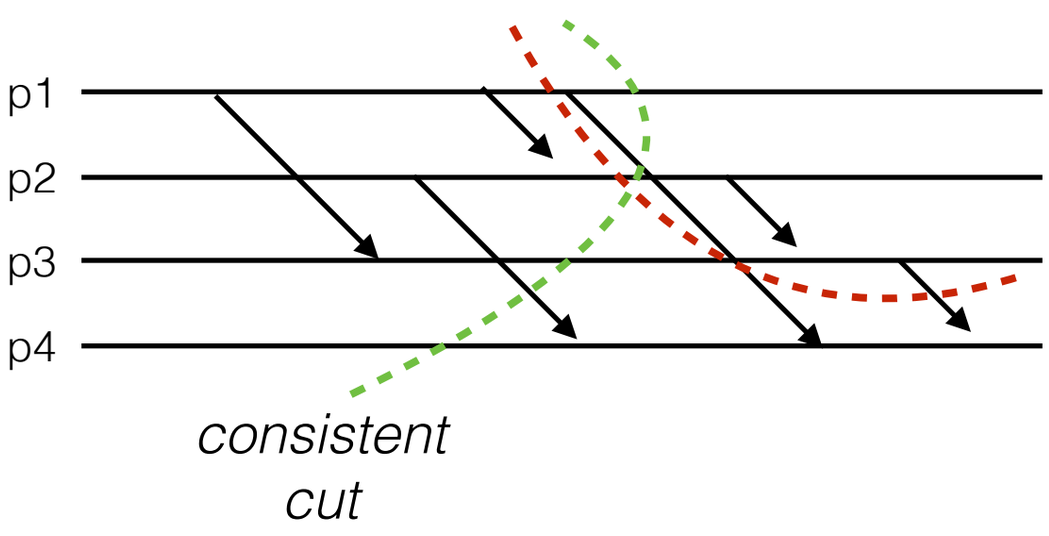
Distributed Snapshots

"A collection of operator states and records in transit (channels) that reflects a moment at a valid execution"



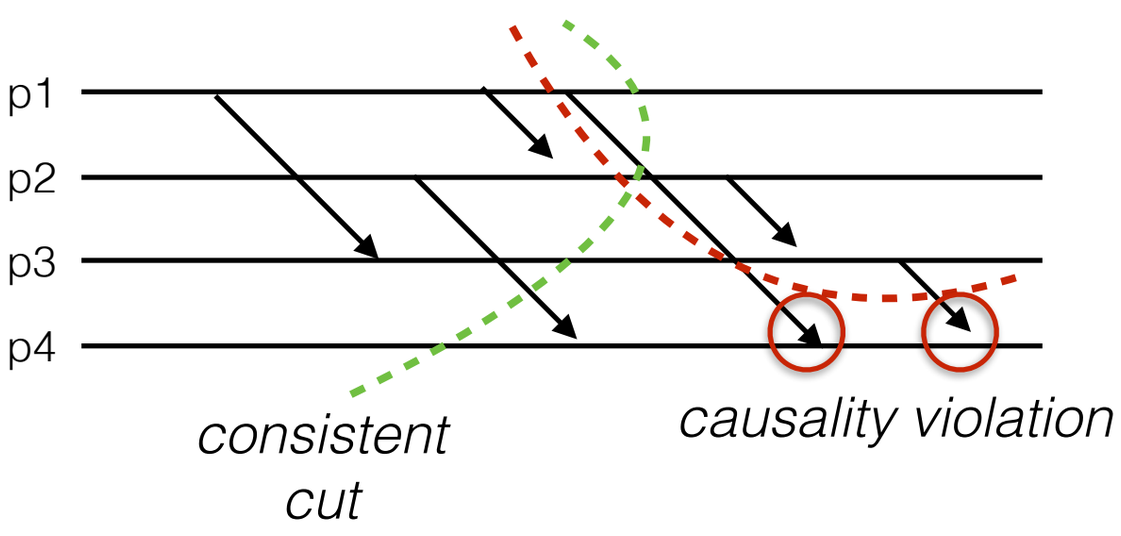
Distributed Snapshots

“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”



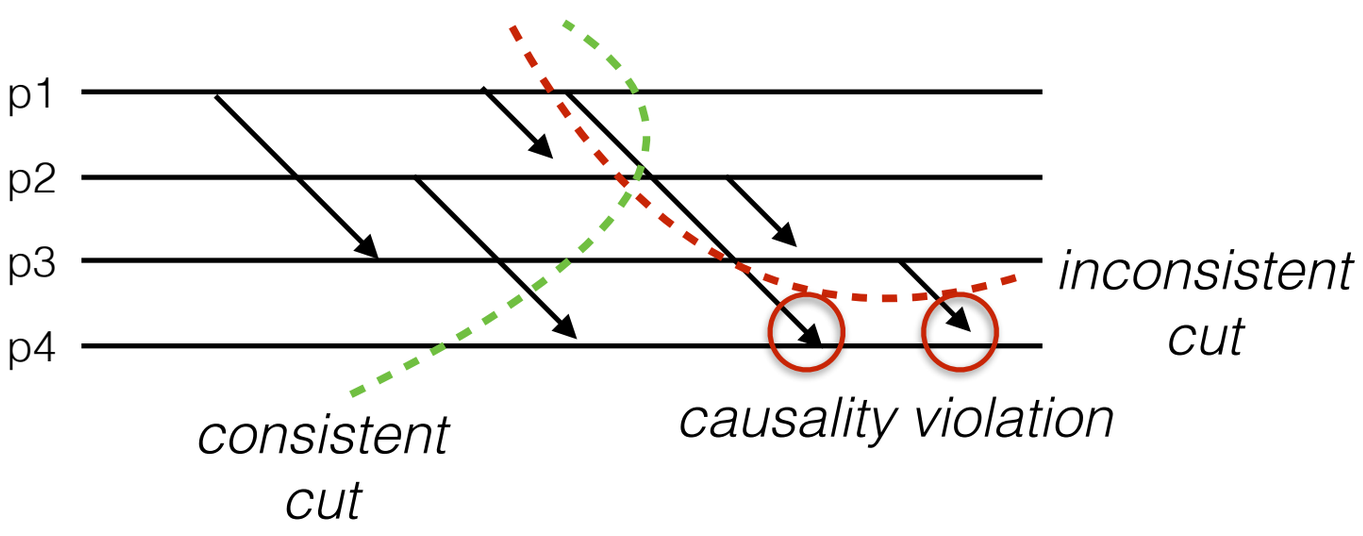
Distributed Snapshots

"A collection of operator states and records in transit (channels) that reflects a moment at a valid execution"



Distributed Snapshots

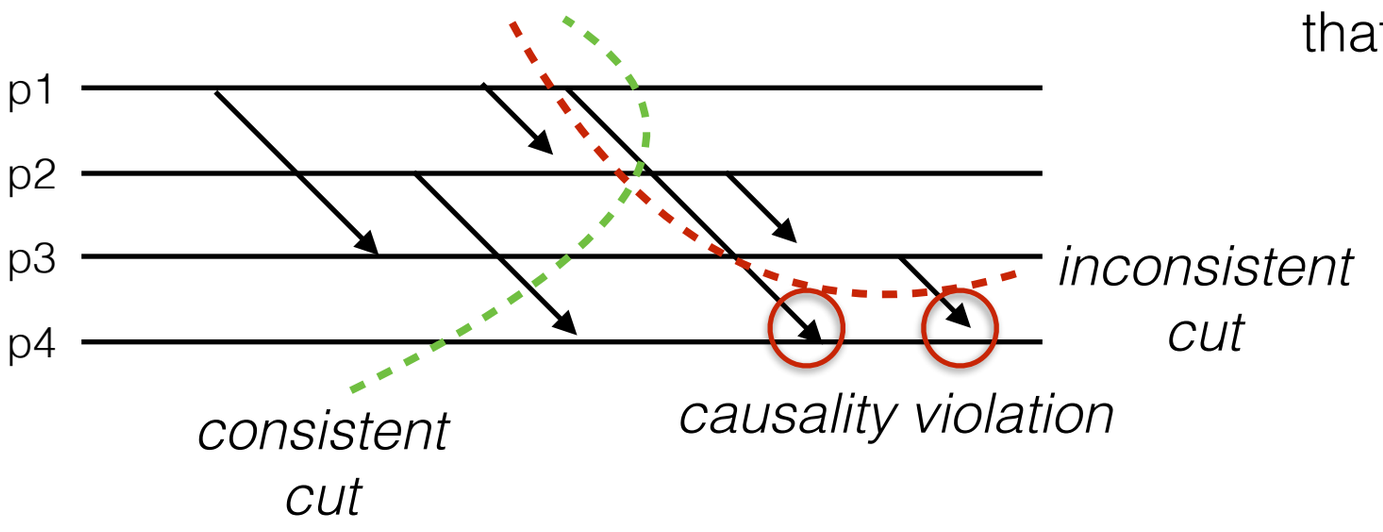
"A collection of operator states and records in transit (channels) that reflects a moment at a valid execution"



Distributed Snapshots

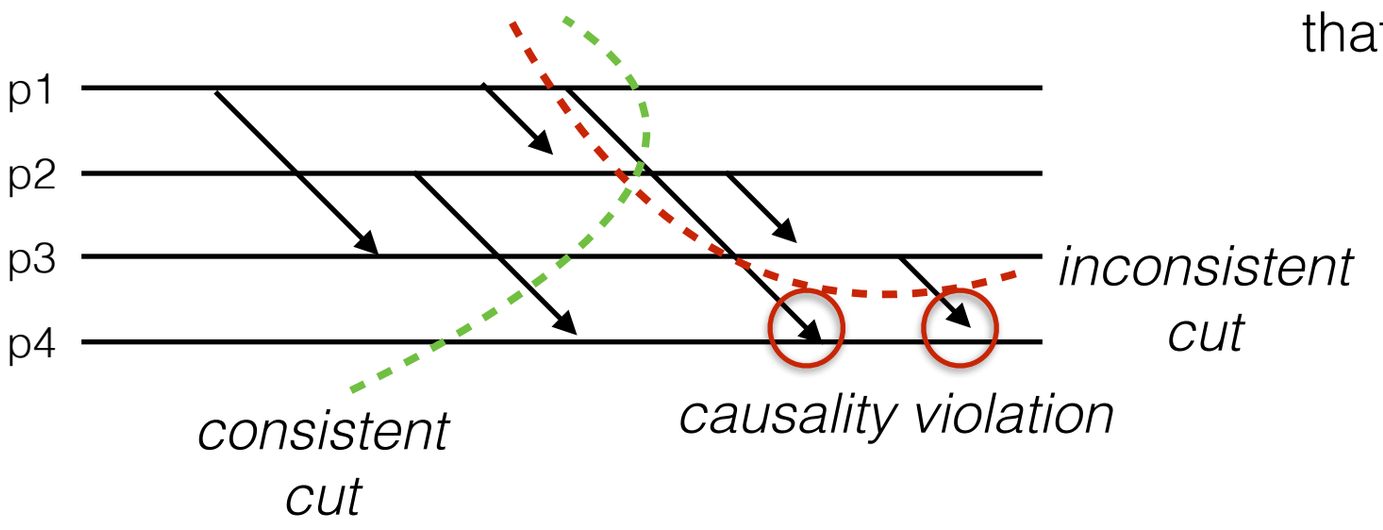
“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”

Idea: We can resume from a snapshot that defines a consistent cut



Distributed Snapshots

“A collection of operator states and records in transit (channels) that reflects a moment at a valid execution”



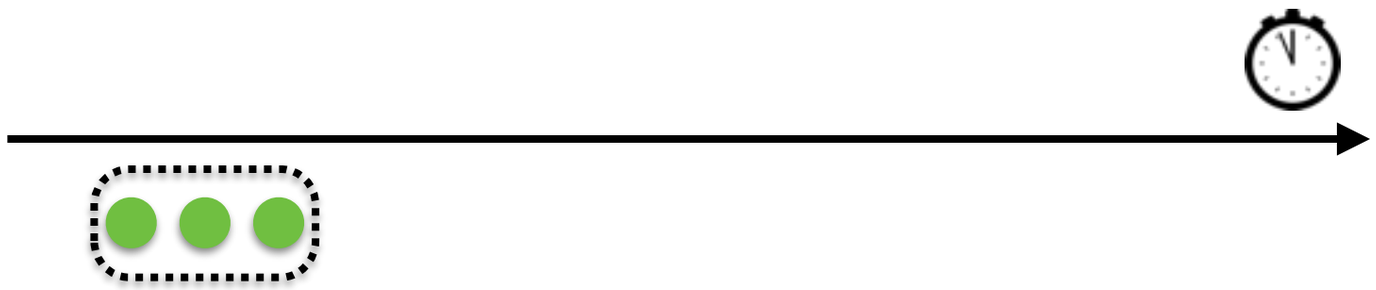
Idea: We can resume from a snapshot that defines a consistent cut

Assumptions

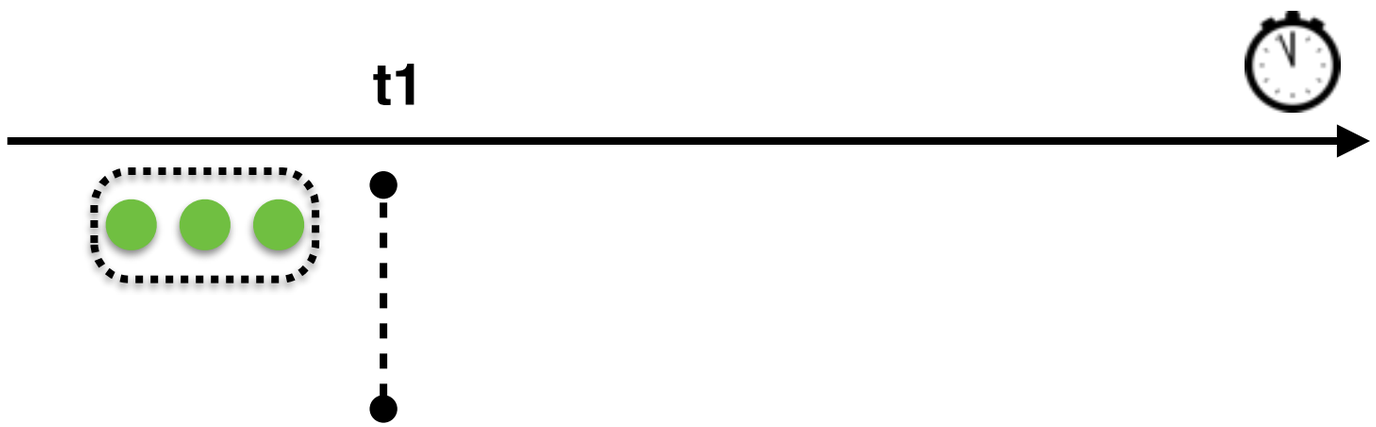
- repeatable sources
- reliable FIFO channels

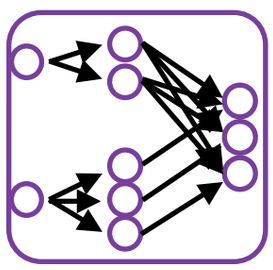
Distributed Snapshots

Distributed Snapshots



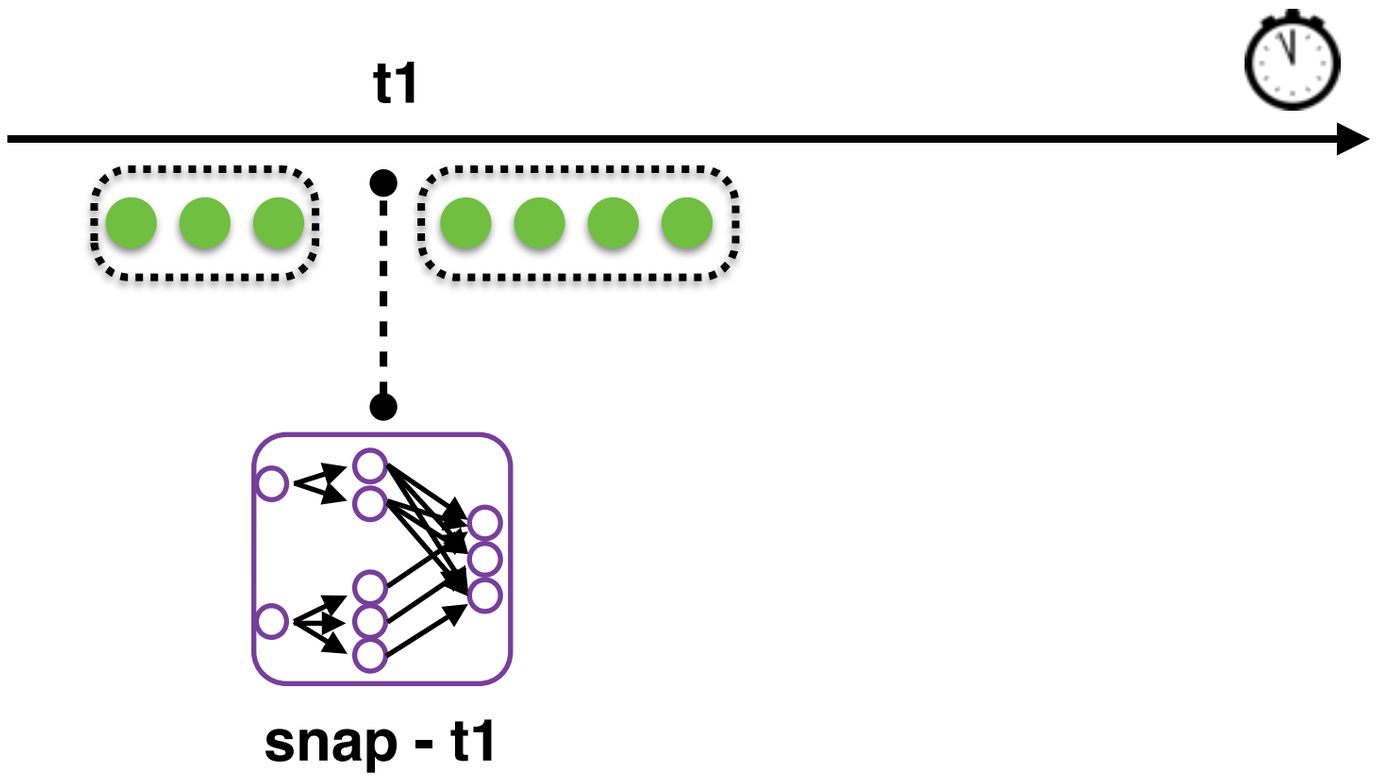
Distributed Snapshots



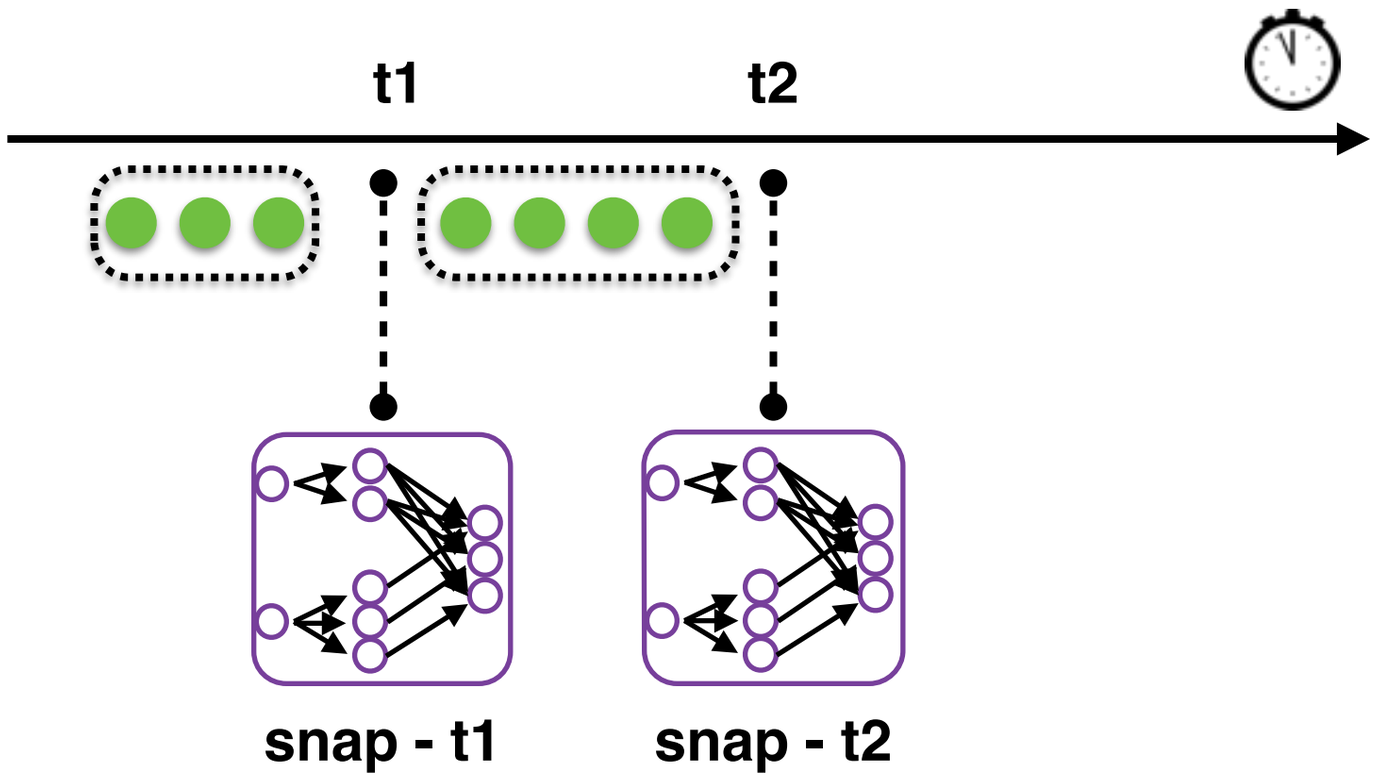


snap - t1

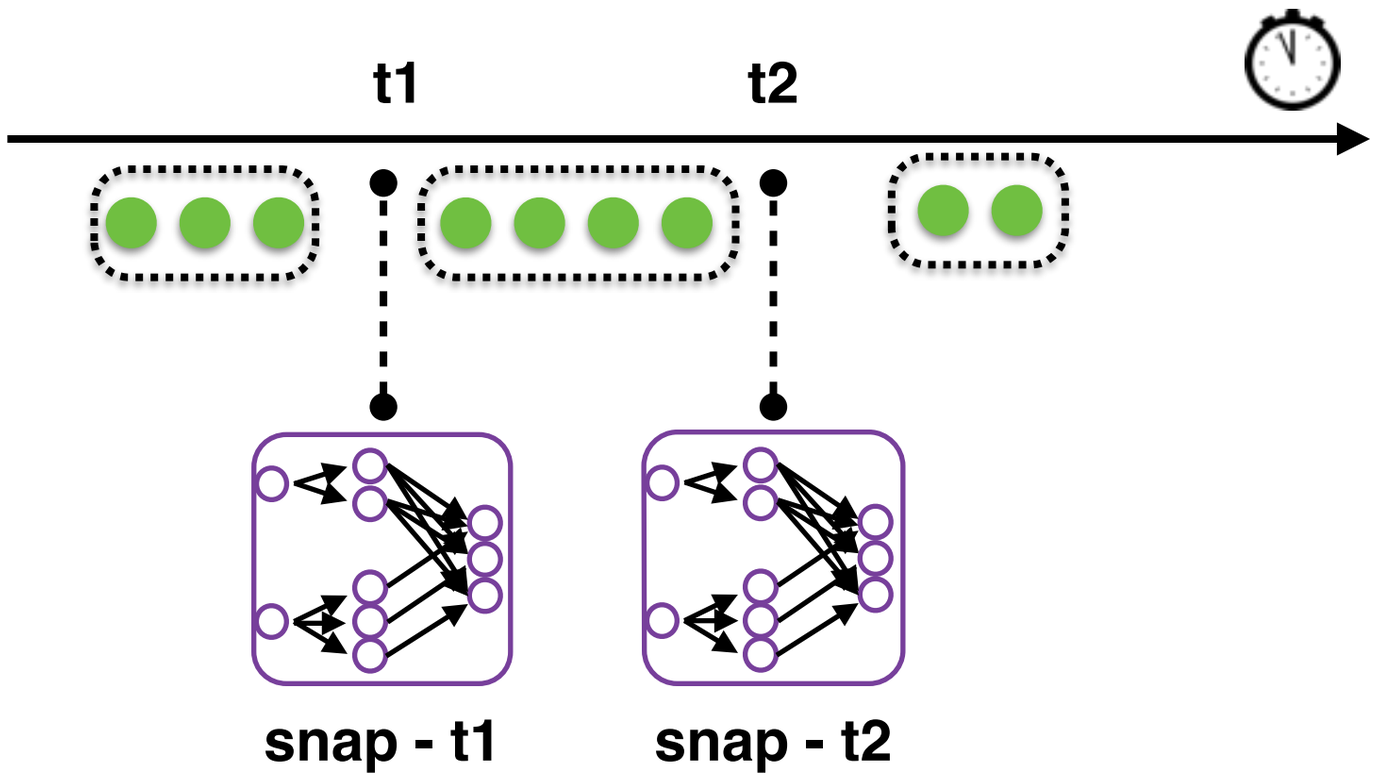
Distributed Snapshots



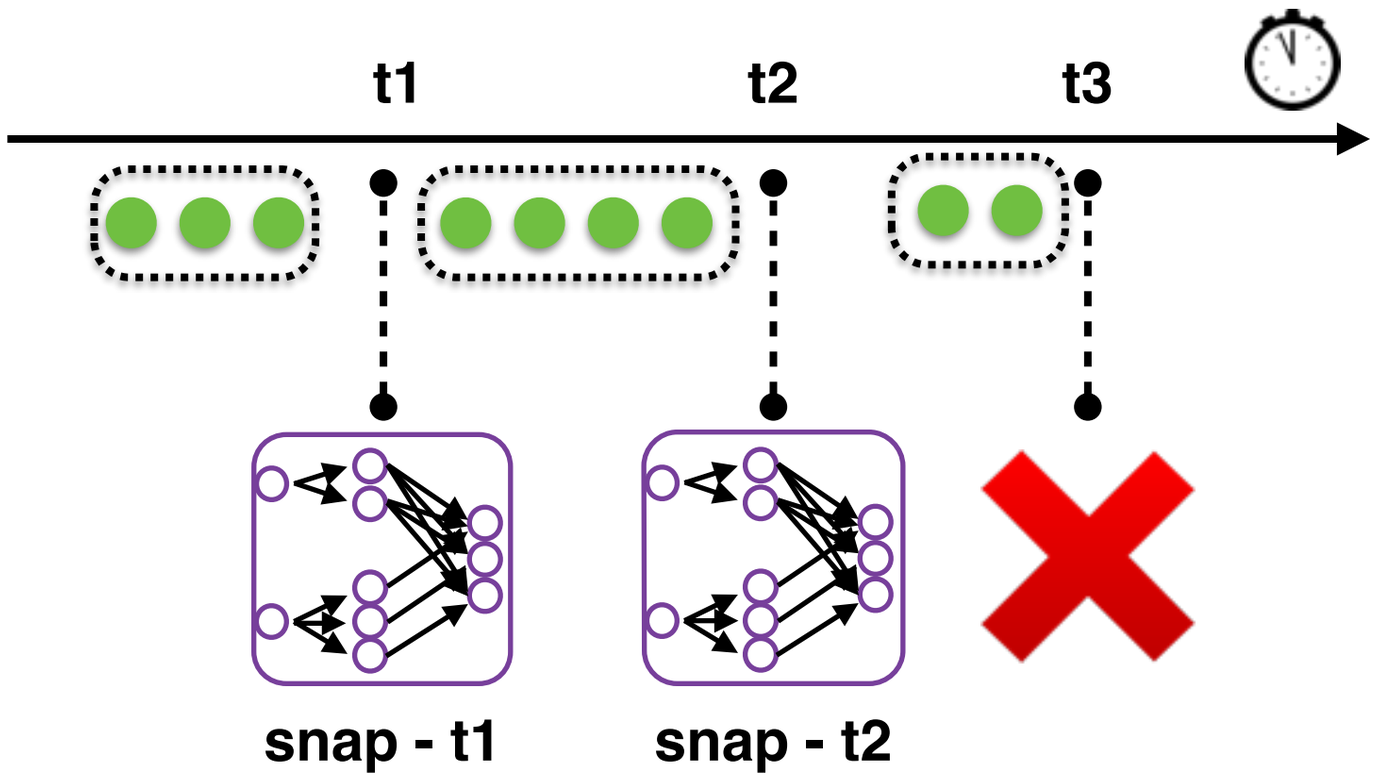
Distributed Snapshots



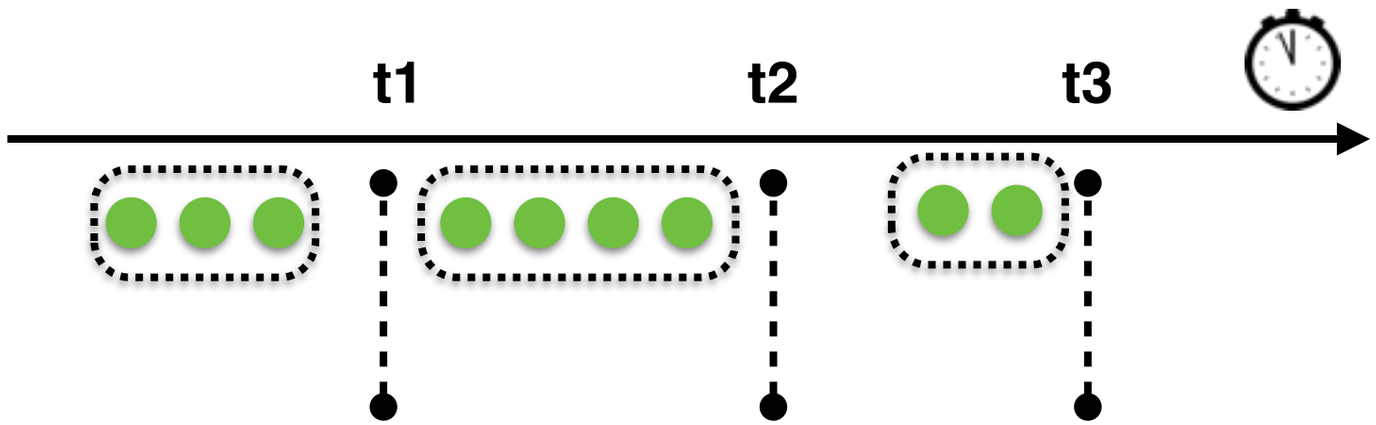
Distributed Snapshots

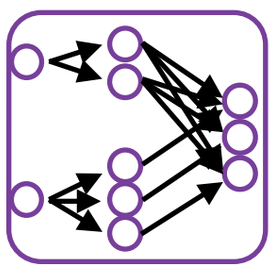


Distributed Snapshots

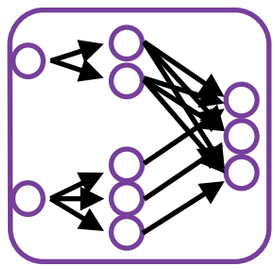


Distributed Snapshots





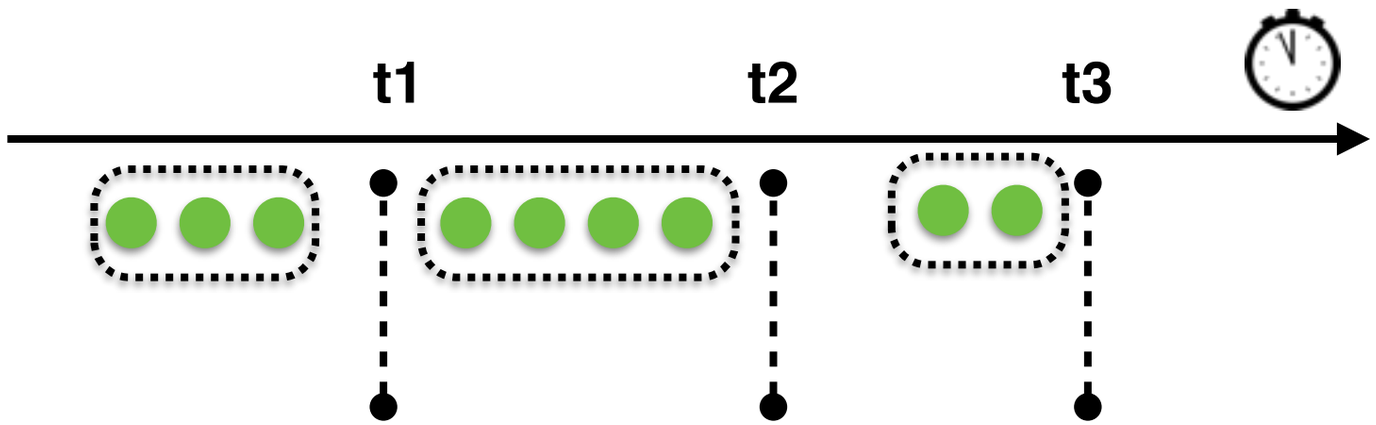
snap - t1

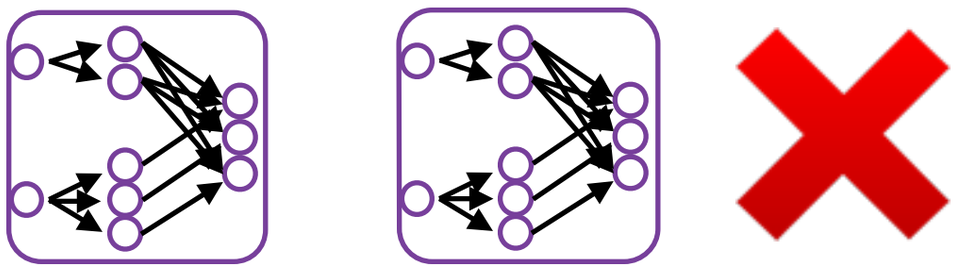


snap - t2

reset from snap t2

Distributed Snapshots





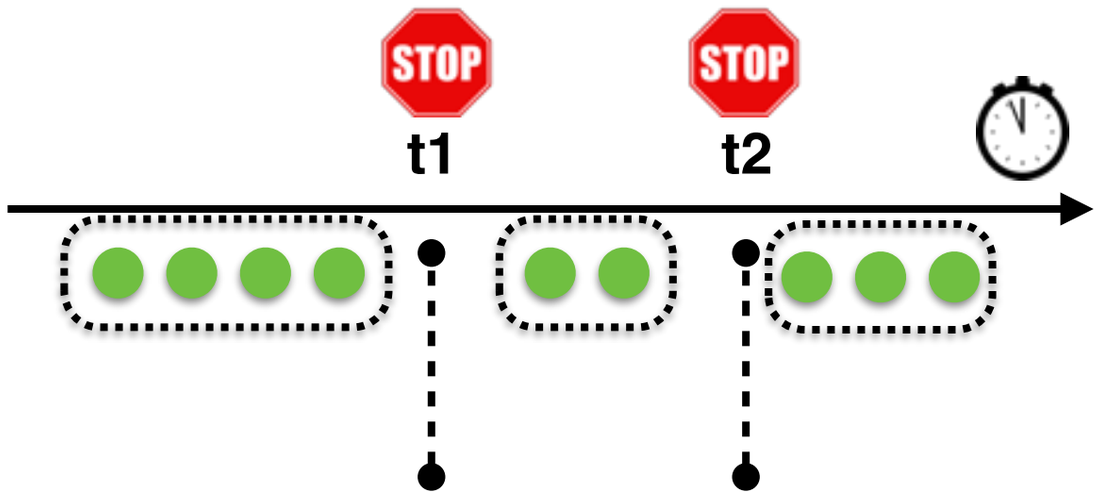
snap - t1 snap - t2

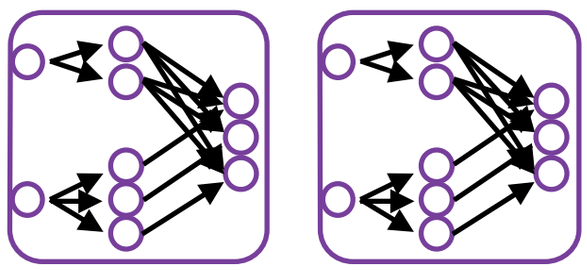
reset from snap t2

Assumptions

- repeatable sources
- reliable FIFO channels

Taking Snapshots





execution snapshots

Initial approach (see Naiad)

- Pause execution on t1,t2,..
- Collect state
- Restore execution

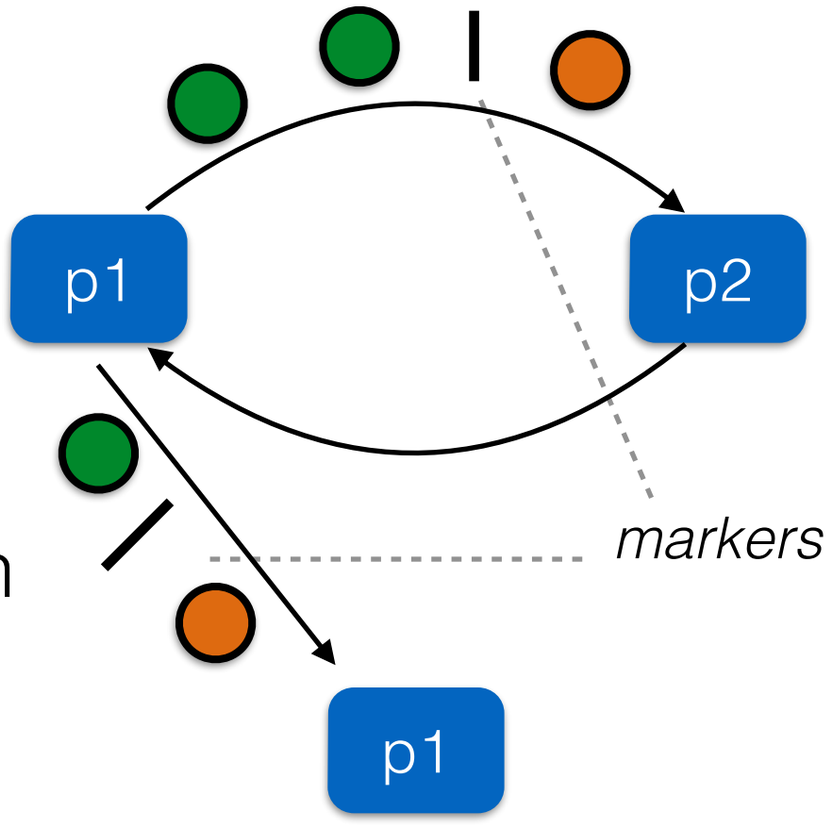
Lamport On the Rescue

“The global-state-detection algorithm is to be superimposed on the underlying computation: it must run concurrently with, but not alter, this underlying computation”

Chandy-Lamport Snapshots

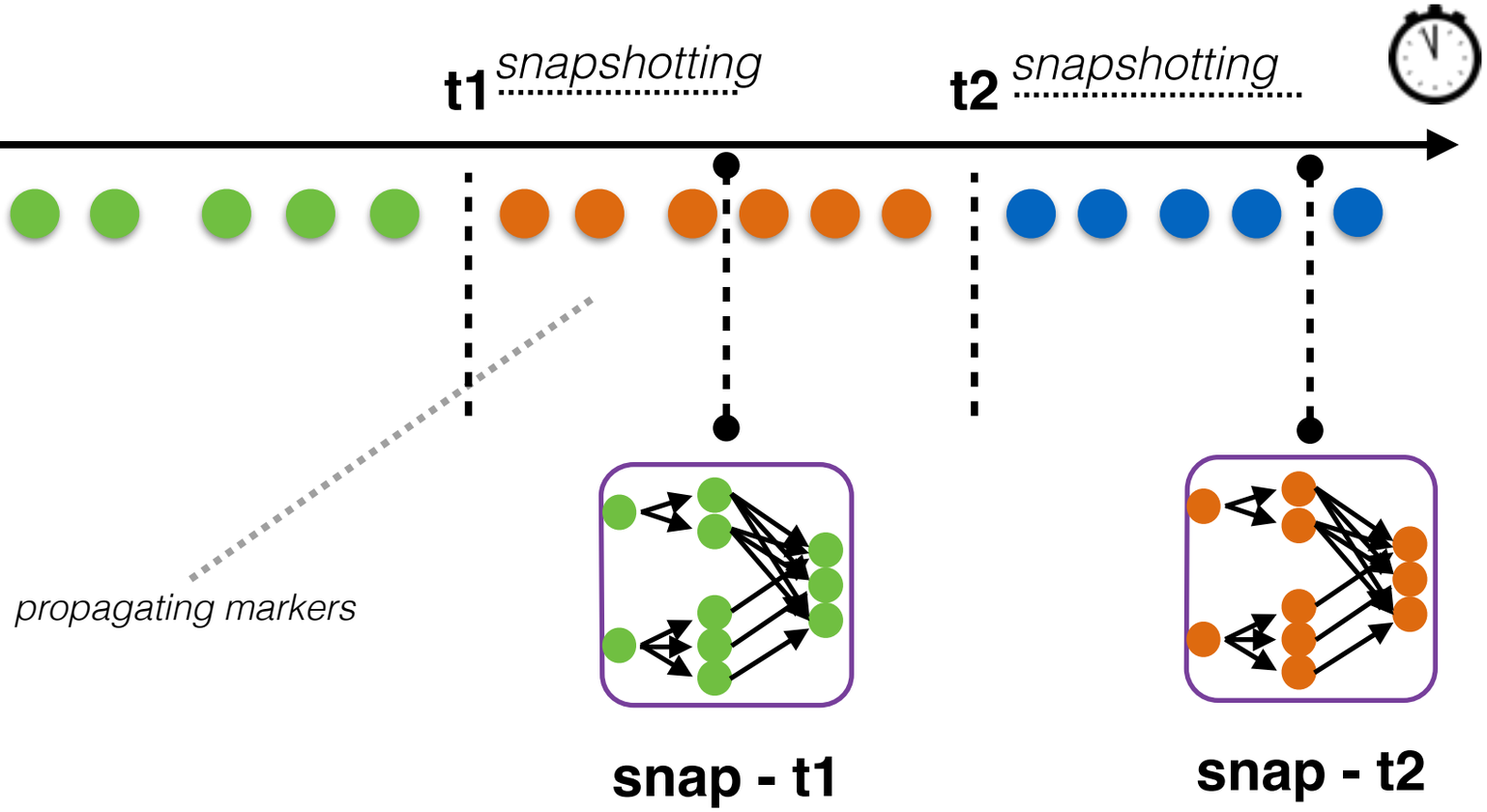
Using Markers/Barriers

- Triggers Snapshots
- Separates preshot-postshot events
- Leads to a consistent execution cut

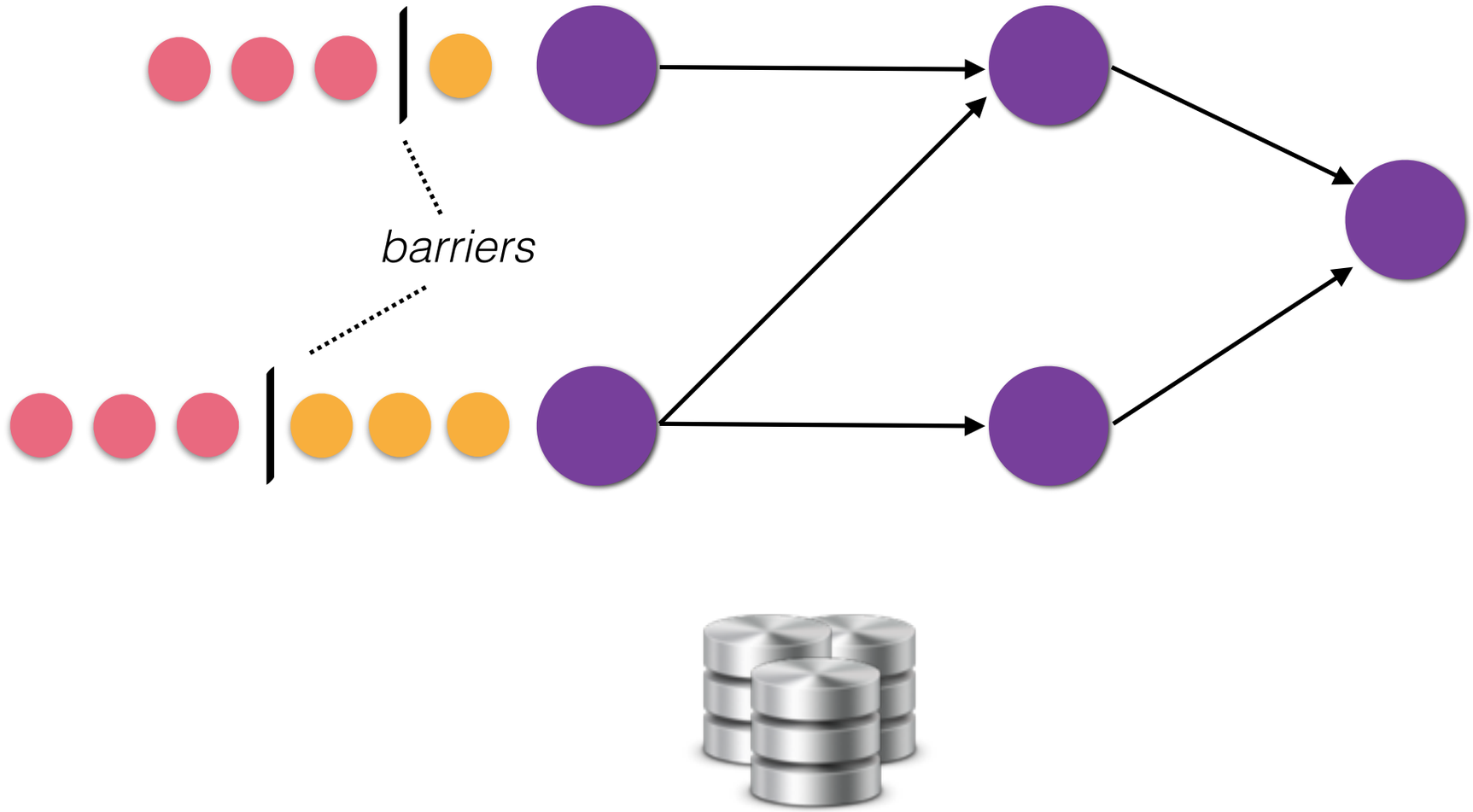


markers propagate the snapshot execution under continuous ingestion

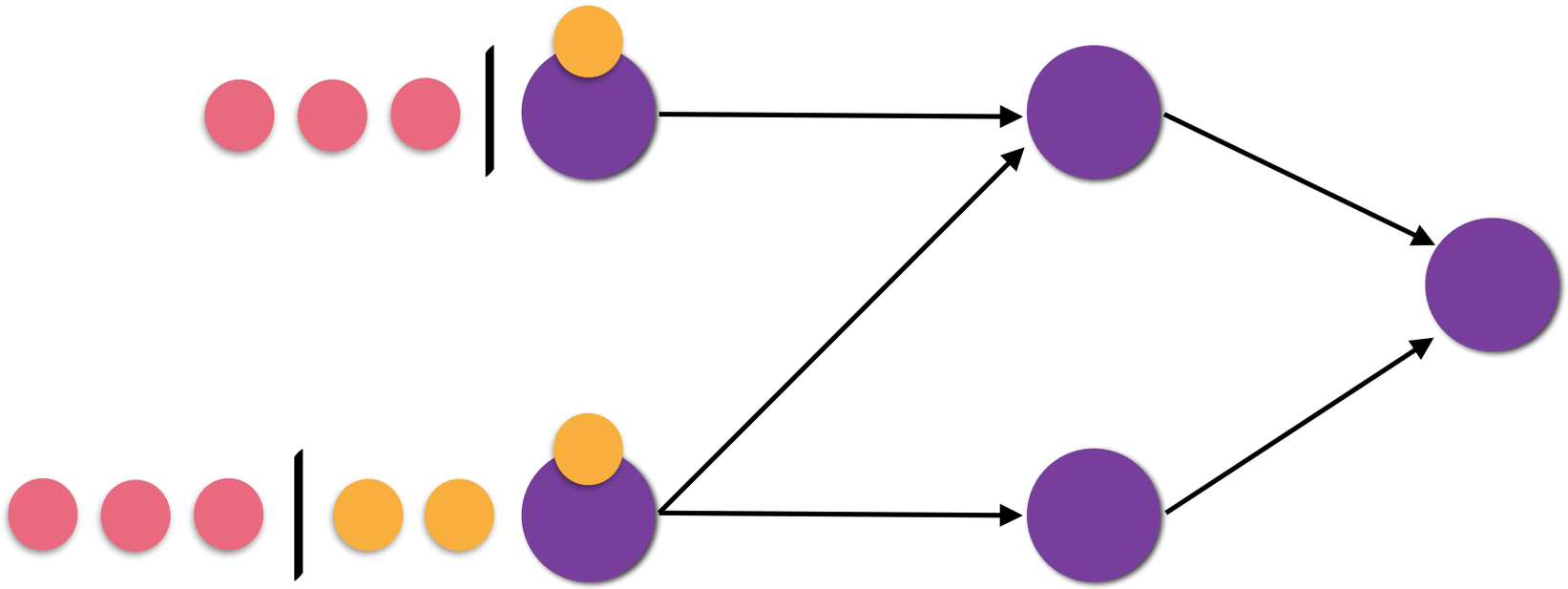
Asynchronous Snapshots



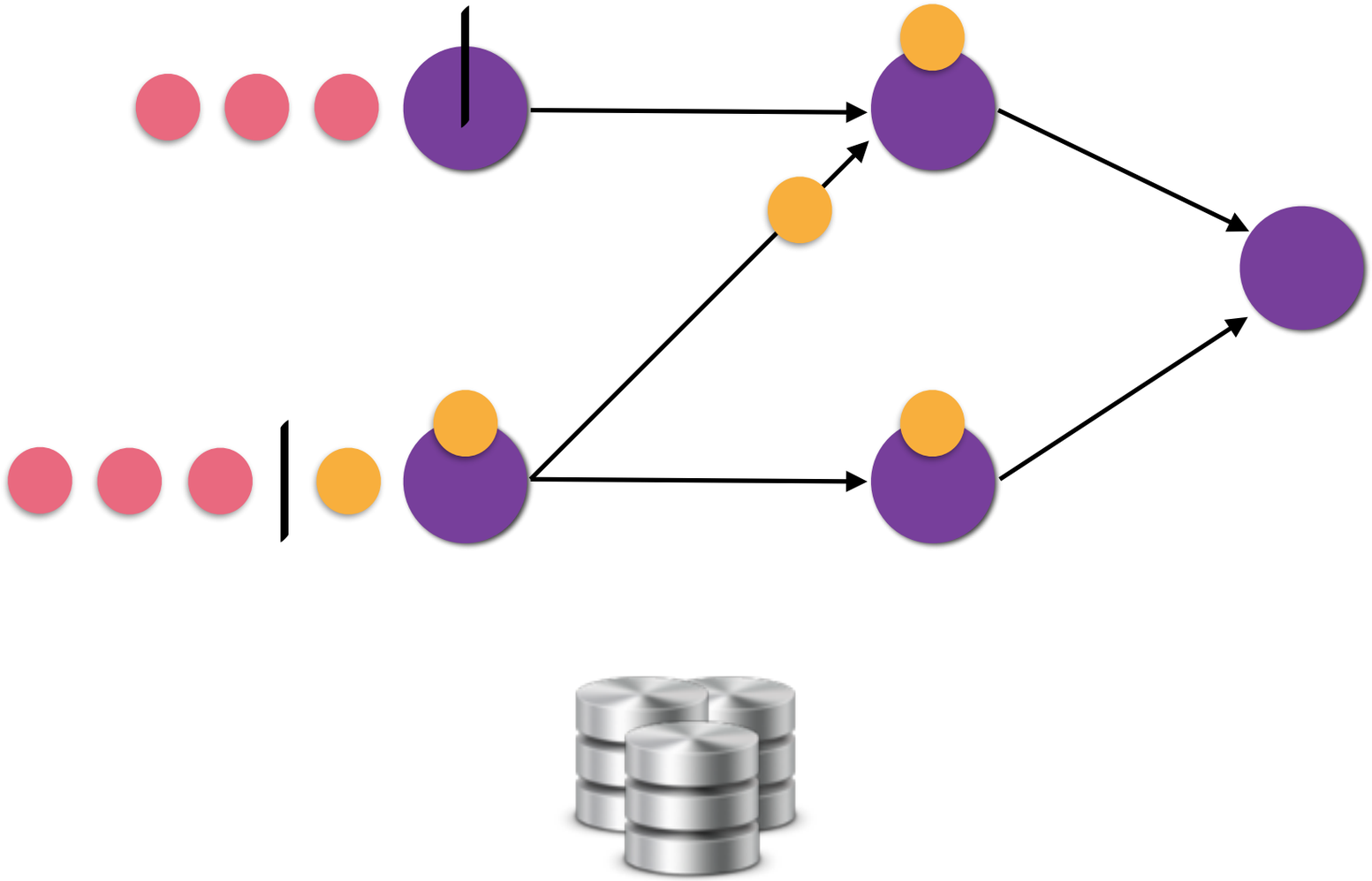
Asynchronous Barrier Snapshotting



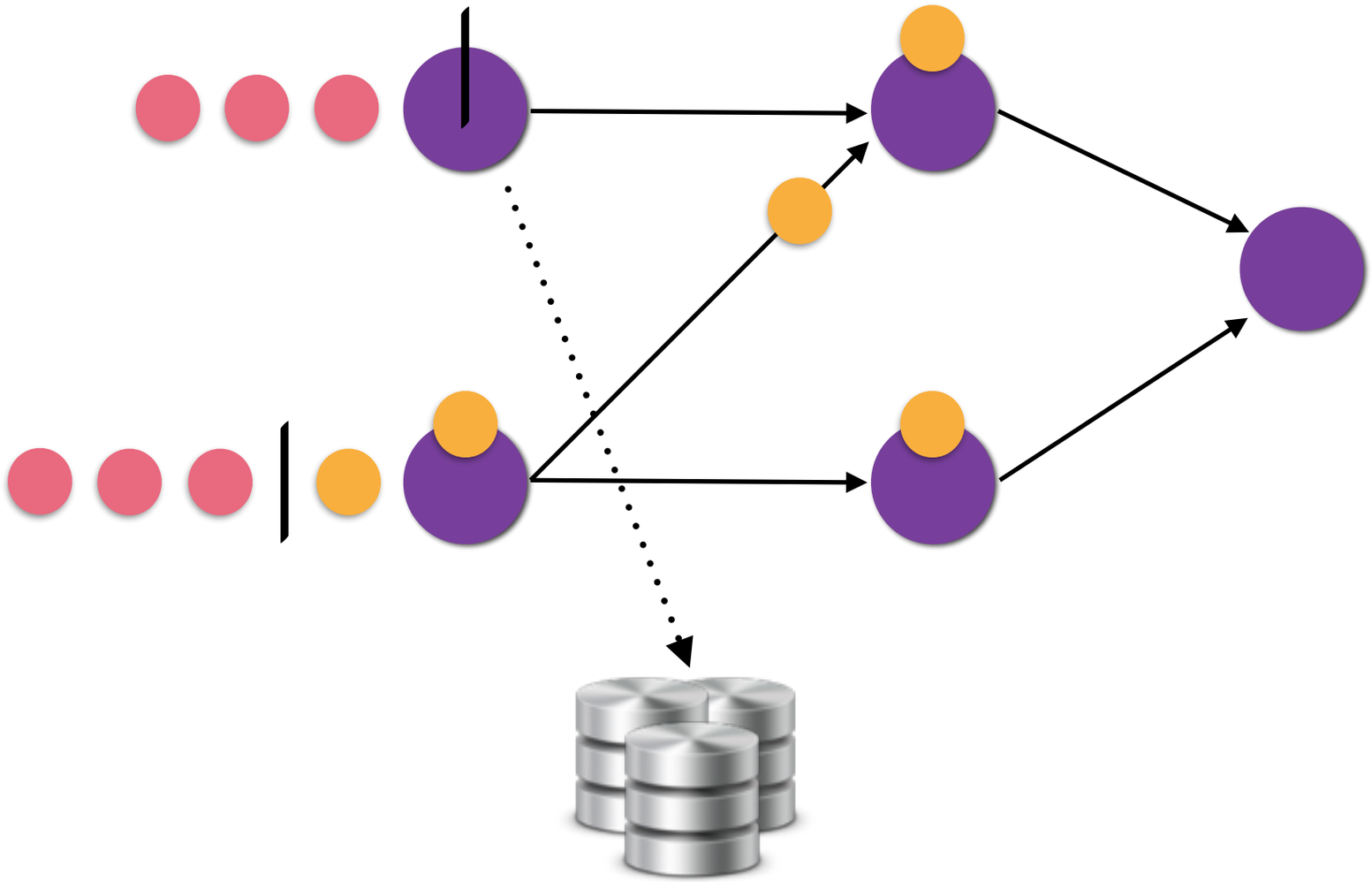
Asynchronous Barrier Snapshotting



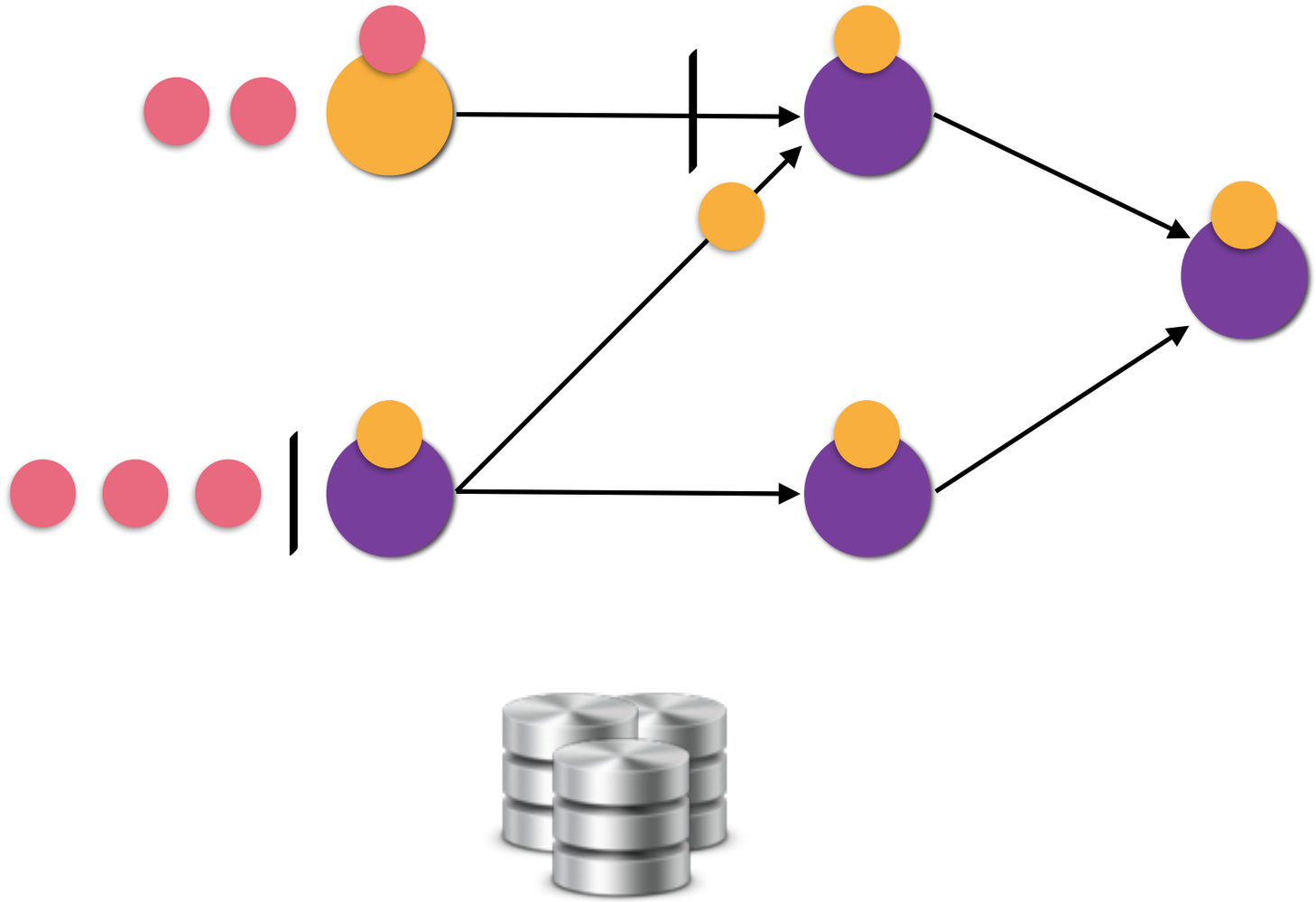
Asynchronous Barrier Snapshotting



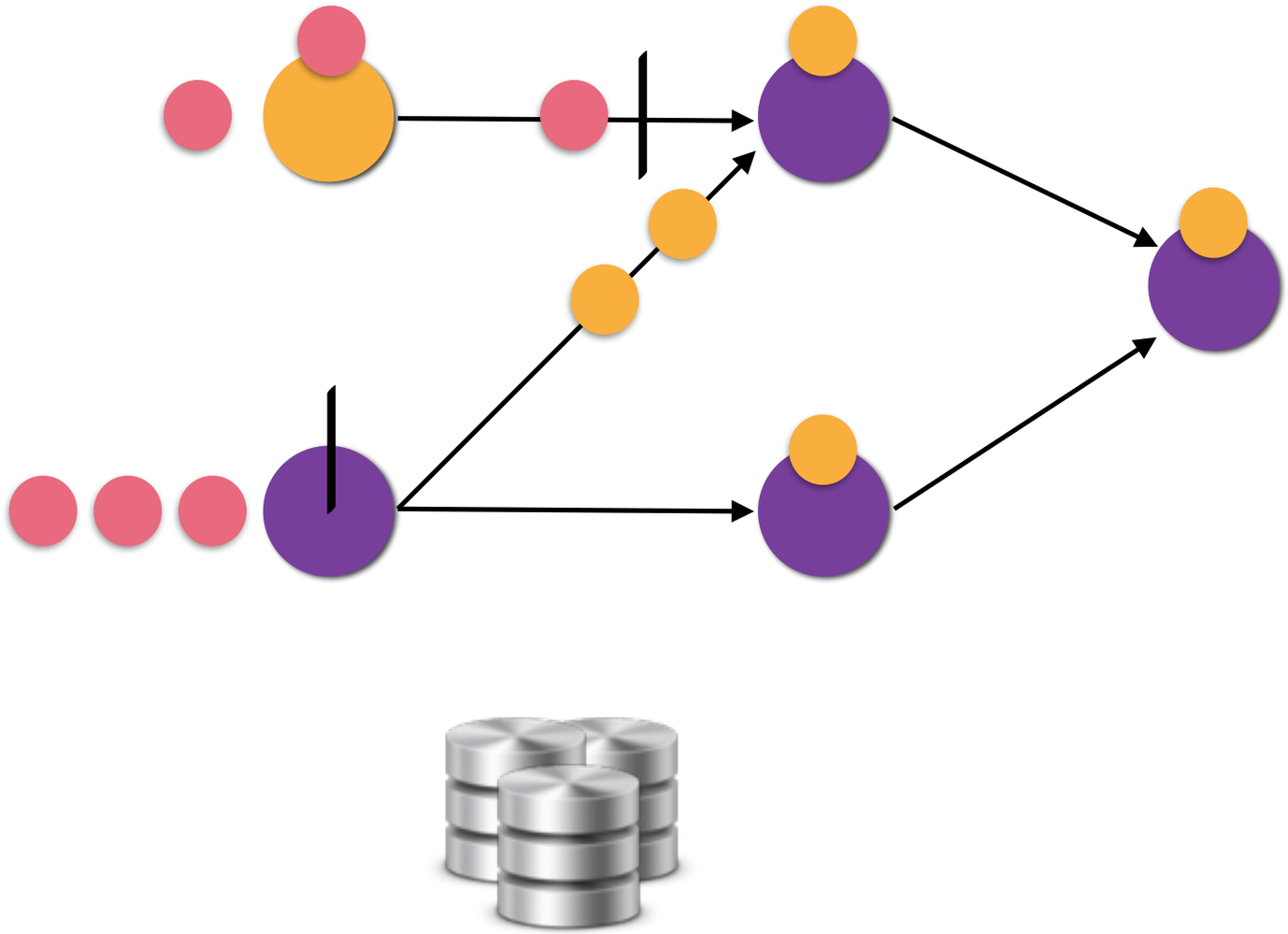
Asynchronous Barrier Snapshotting



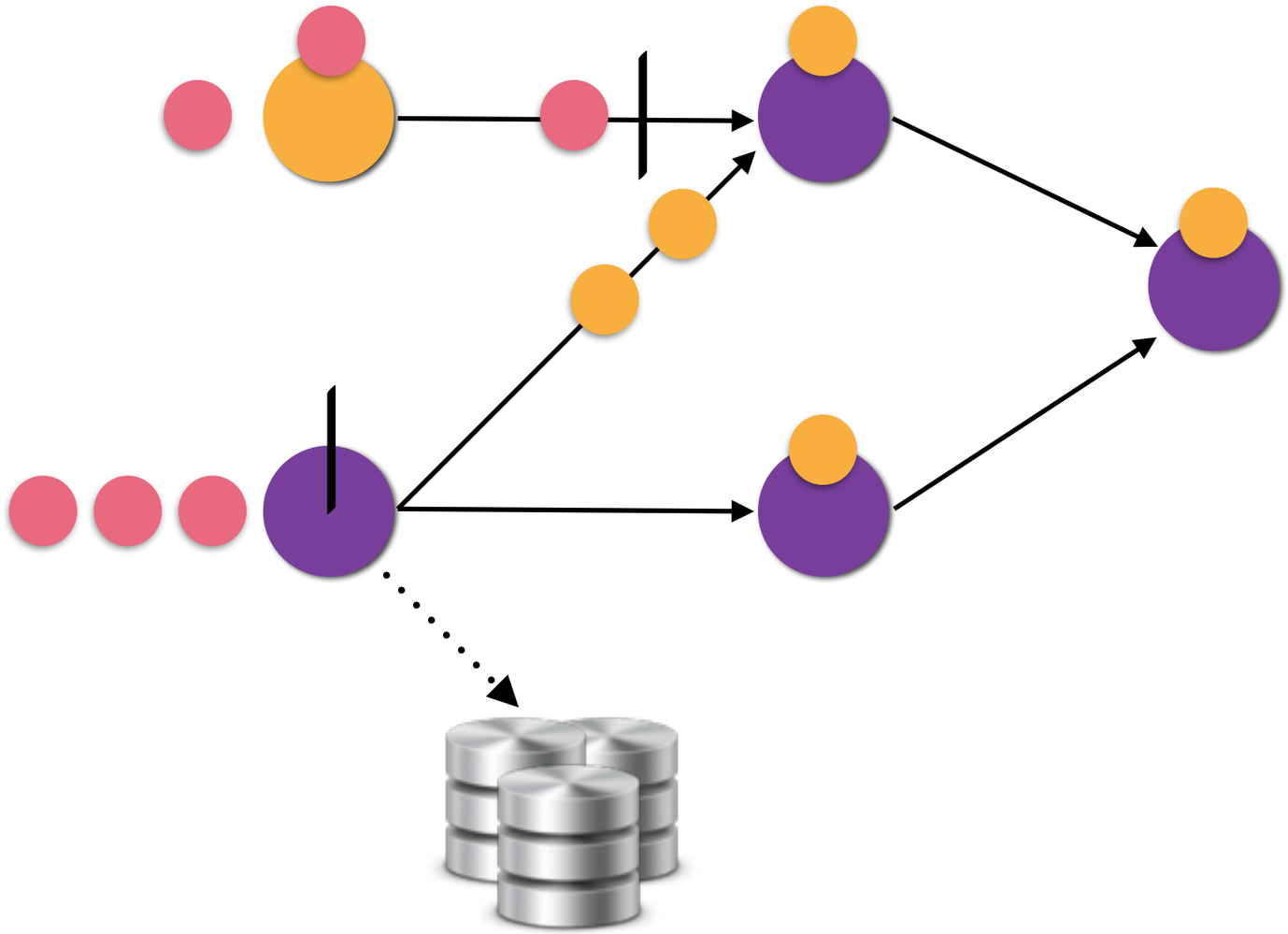
Asynchronous Barrier Snapshotting



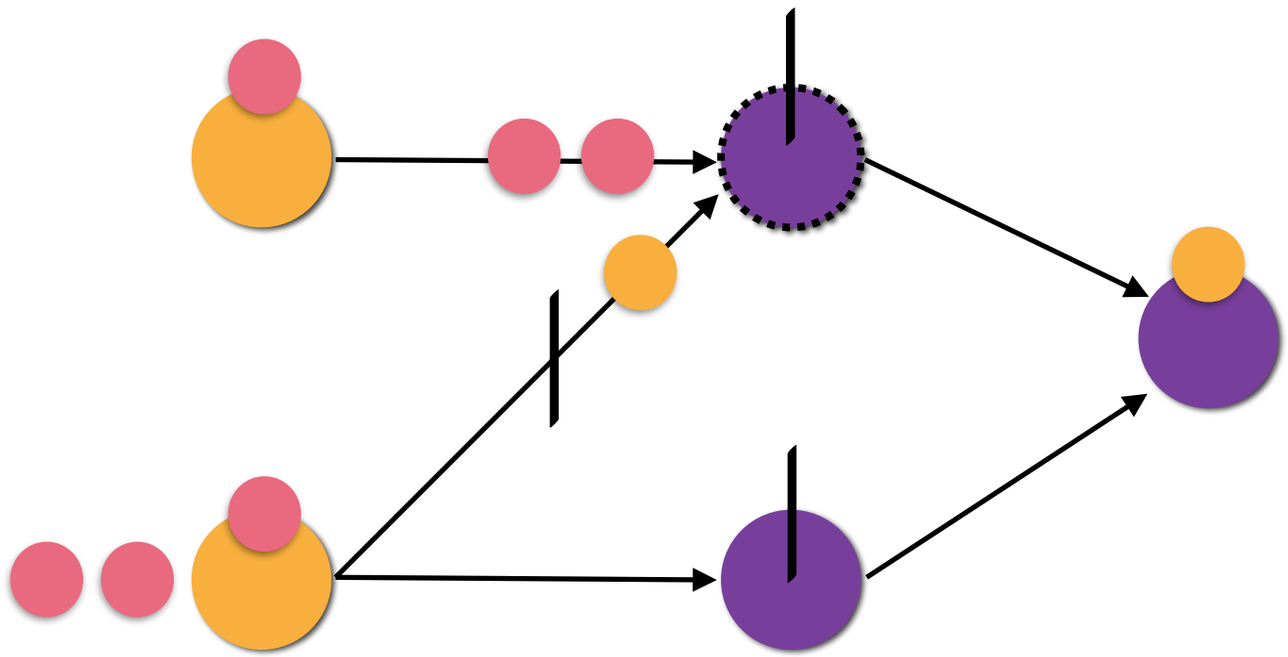
Asynchronous Barrier Snapshotting



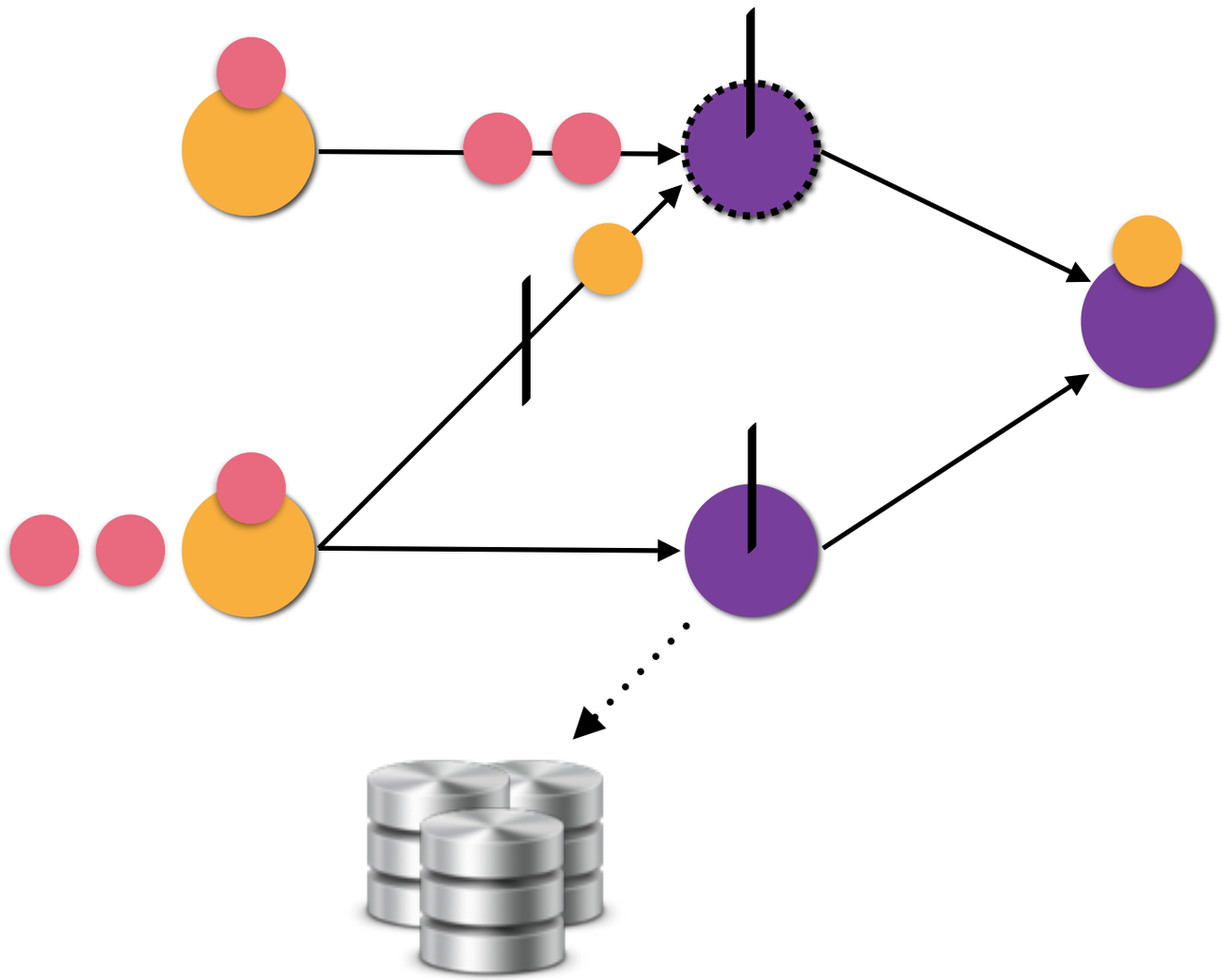
Asynchronous Barrier Snapshotting



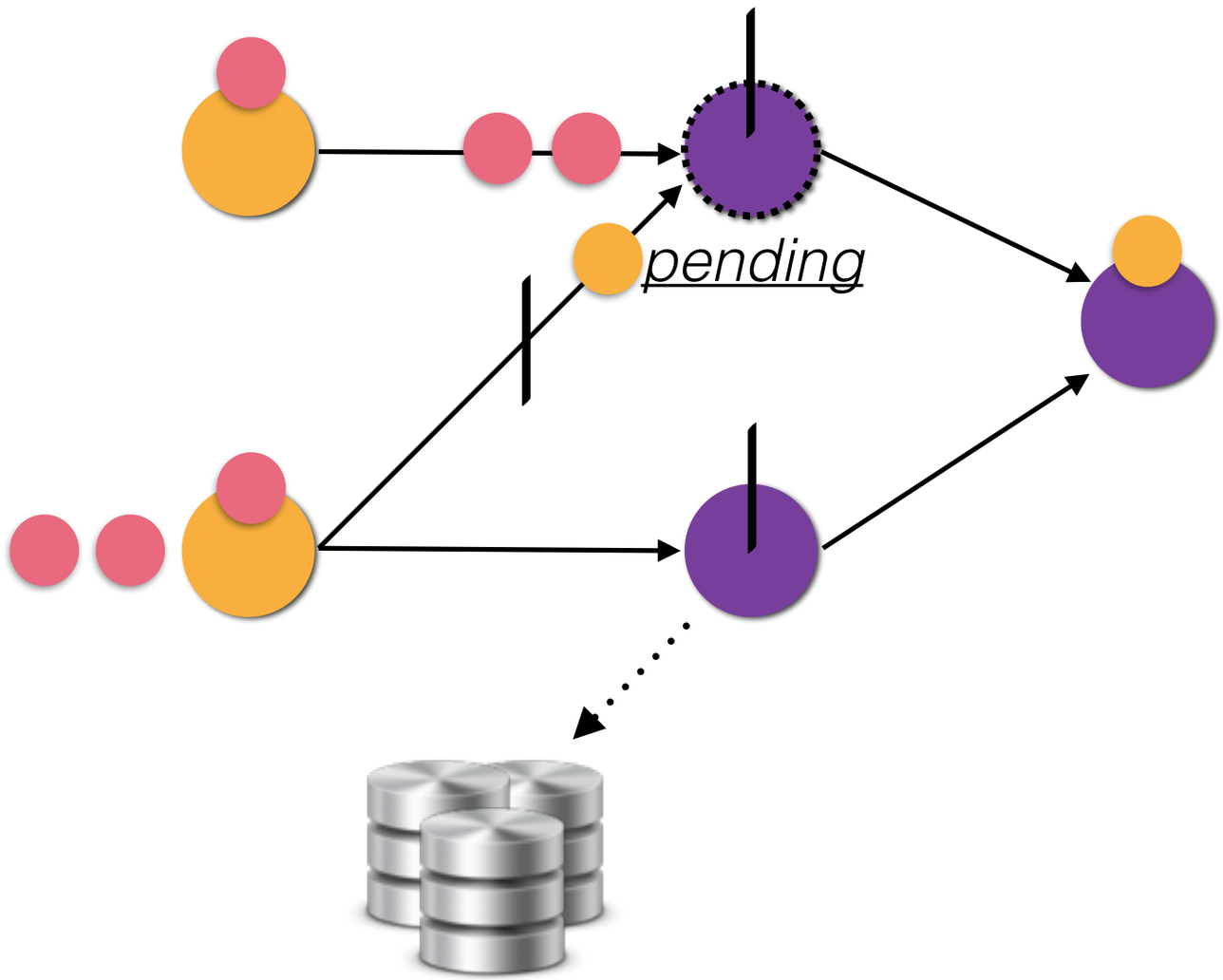
Asynchronous Barrier Snapshotting



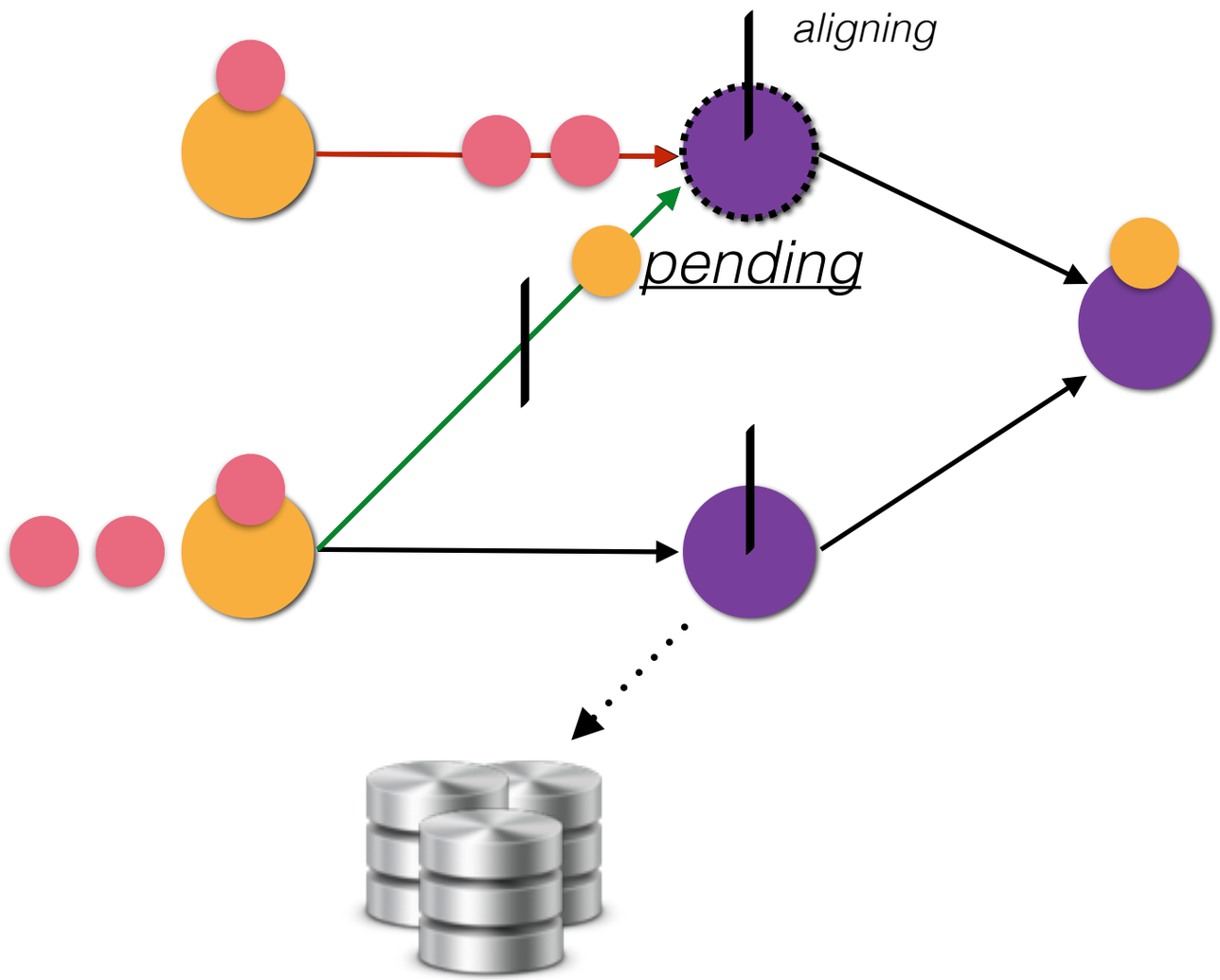
Asynchronous Barrier Snapshotting



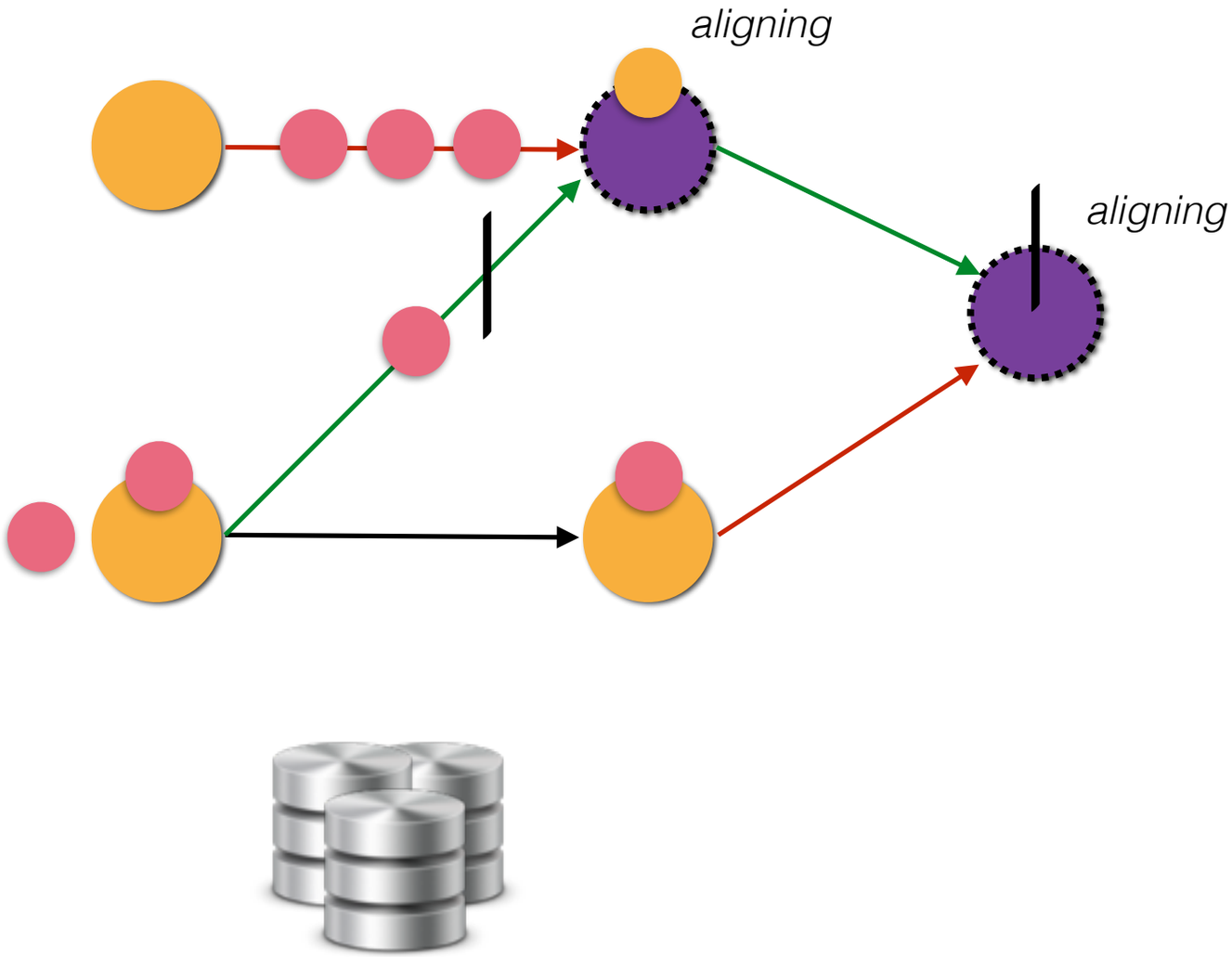
Asynchronous Barrier Snapshotting



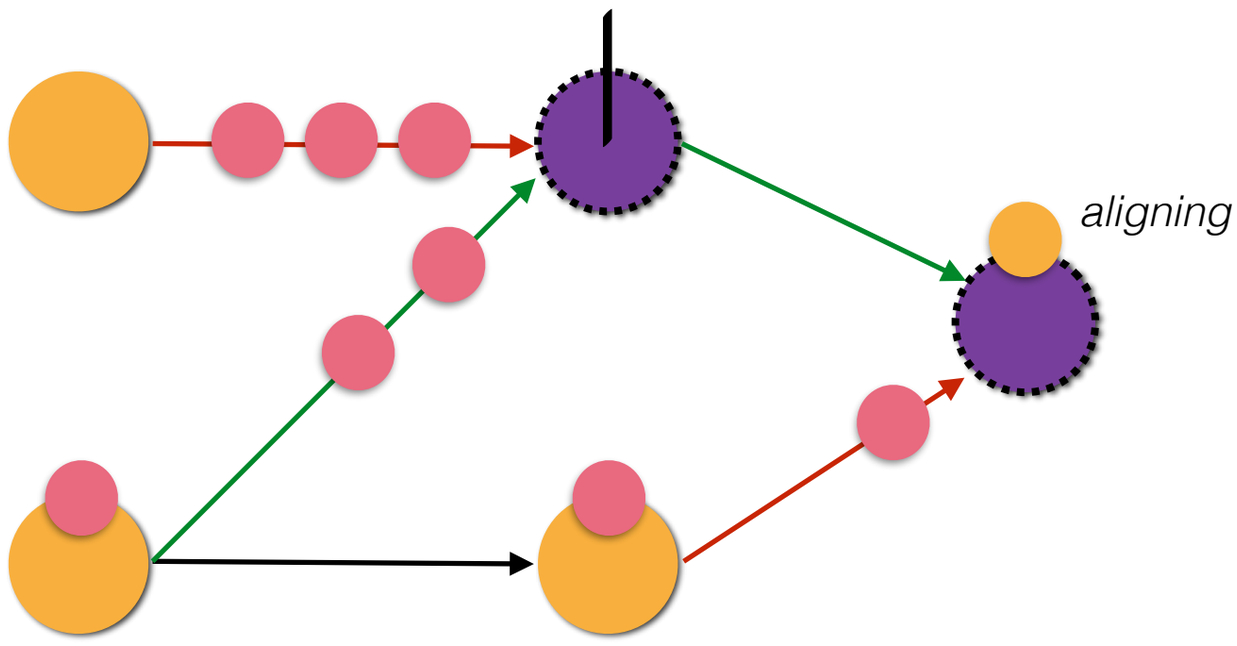
Asynchronous Barrier Snapshotting



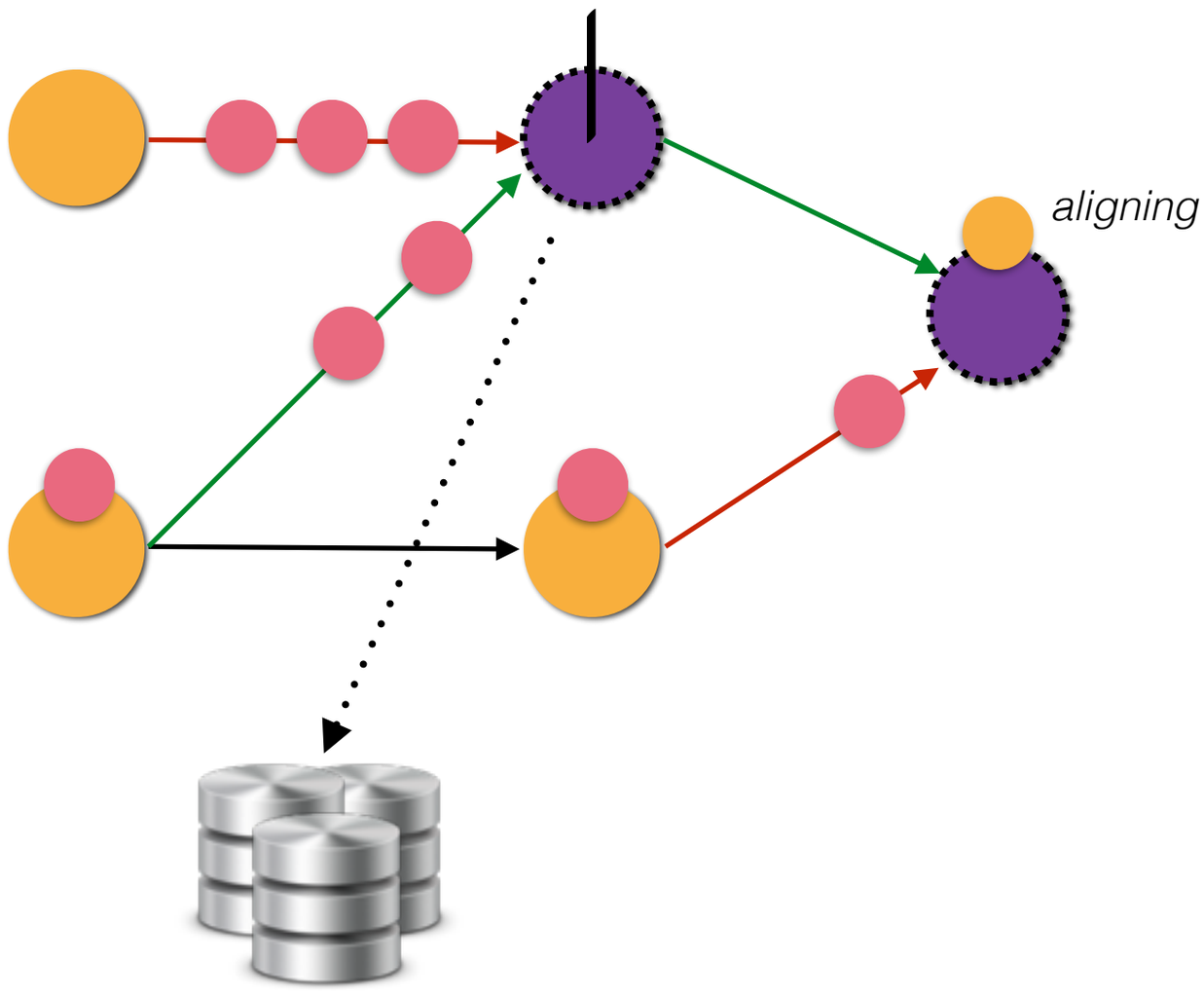
Asynchronous Barrier Snapshotting



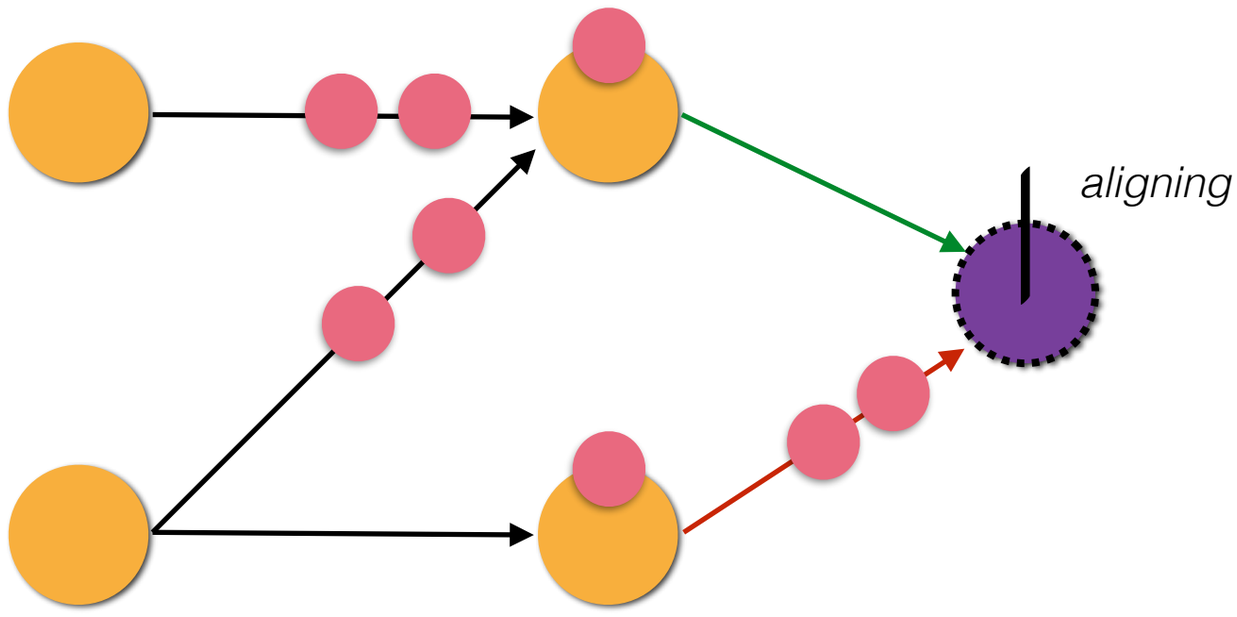
Asynchronous Barrier Snapshotting



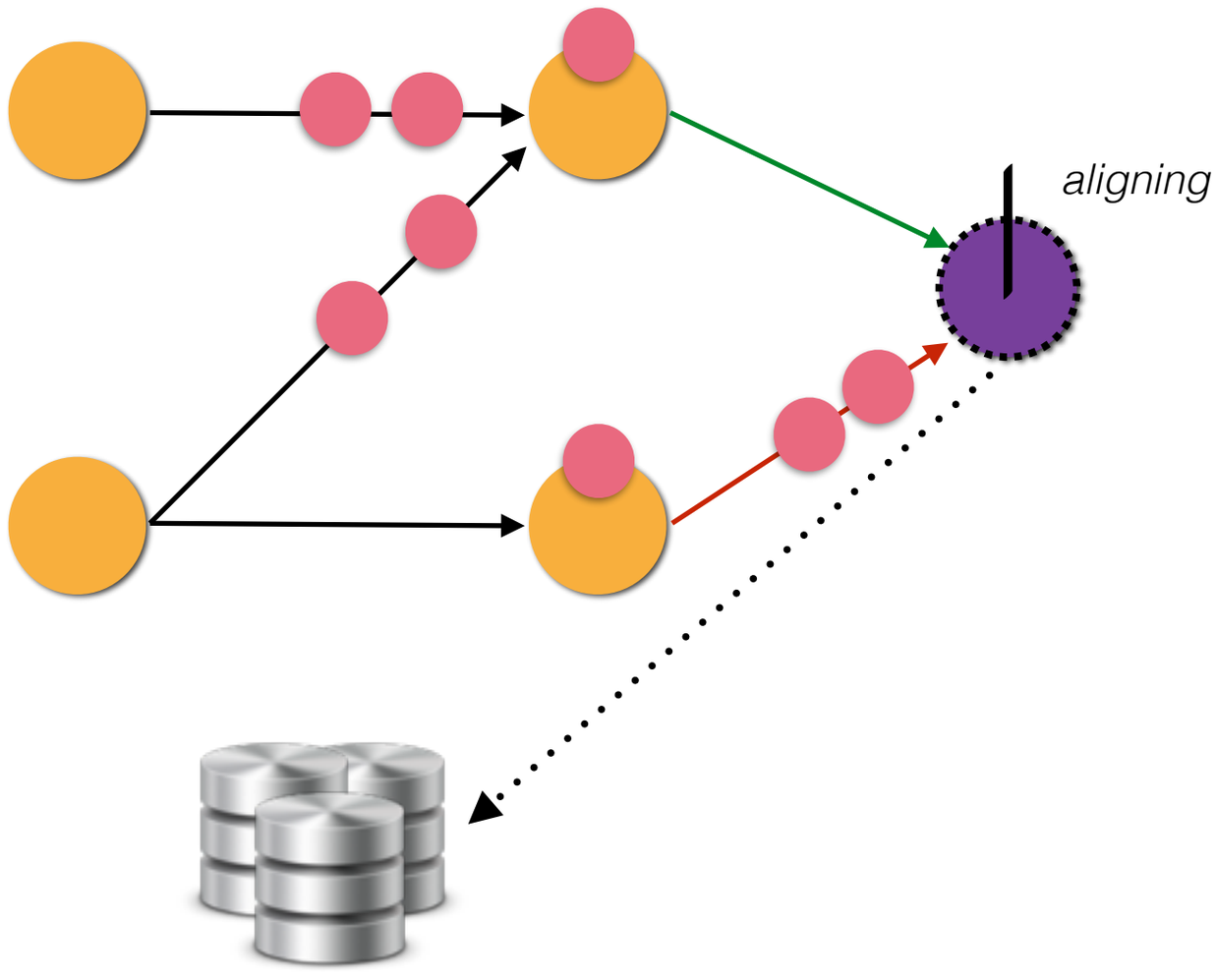
Asynchronous Barrier Snapshotting



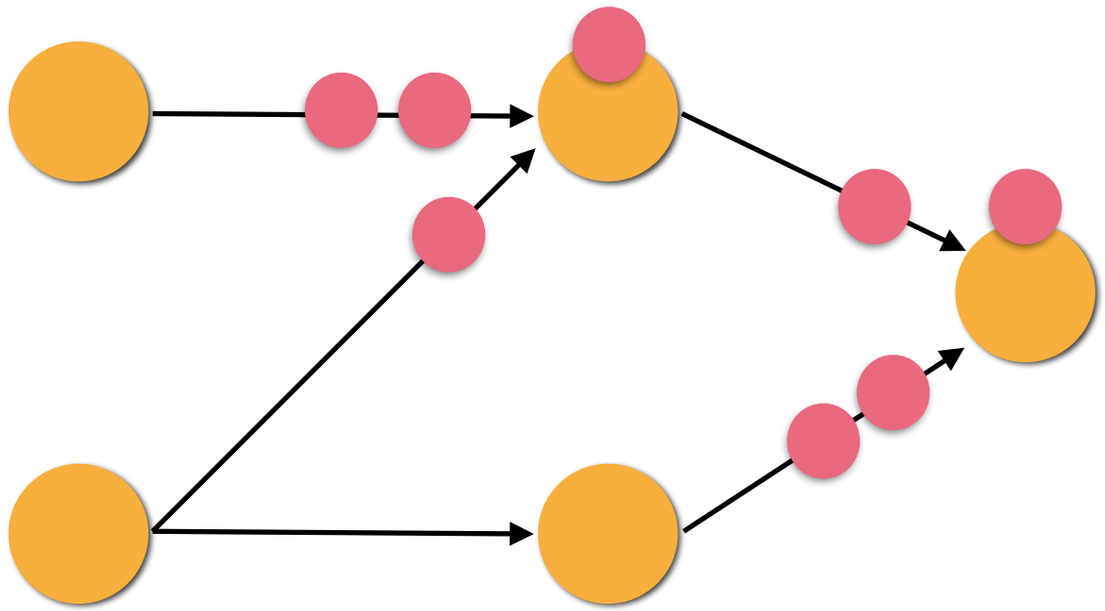
Asynchronous Barrier Snapshotting



Asynchronous Barrier Snapshotting



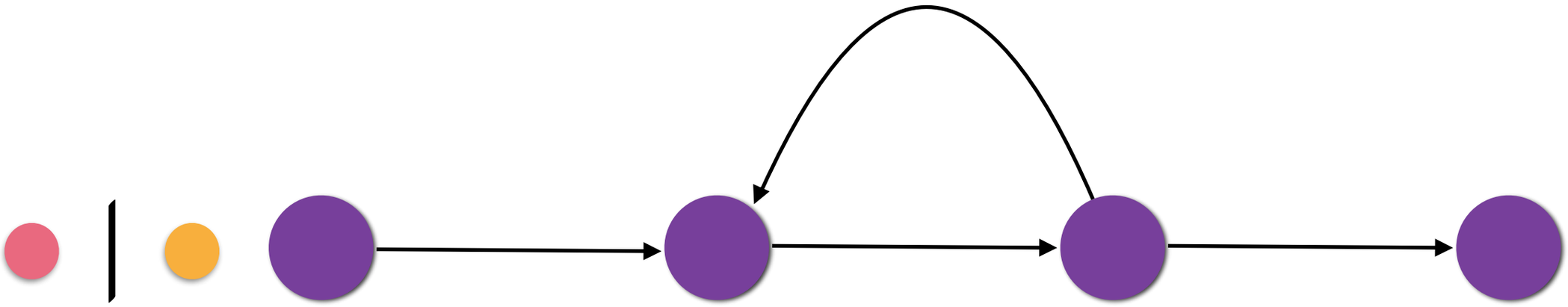
Asynchronous Barrier Snapshotting



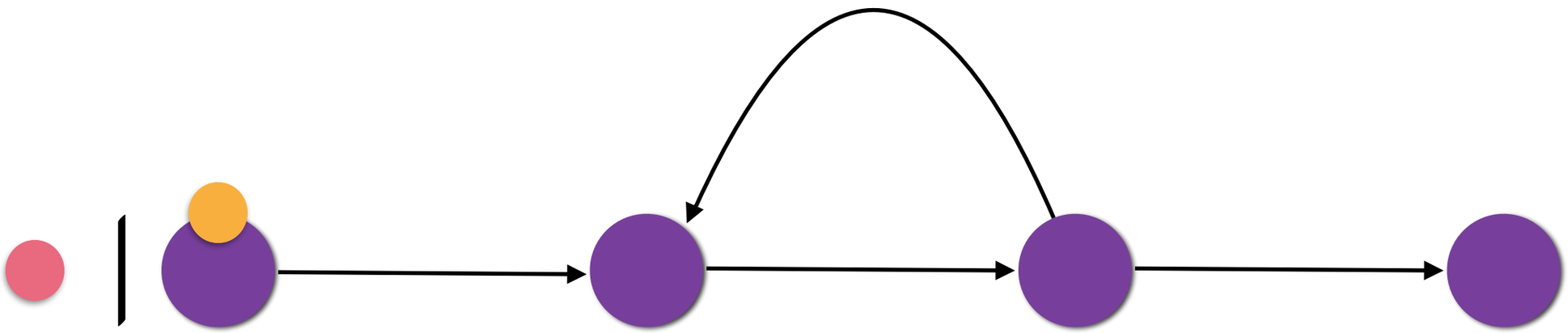
Asynchronous Barrier Snapshotting Benefits

- Taking advantage of the execution graph structure
- No records in transit included in the snapshot
- Aligning has lower impact than halting

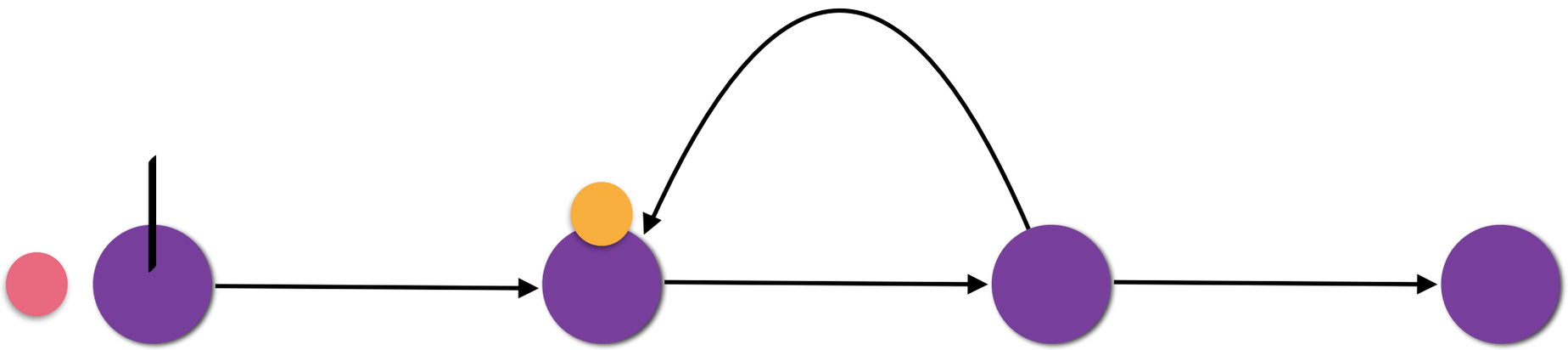
Snapshots on Cyclic Dataflows



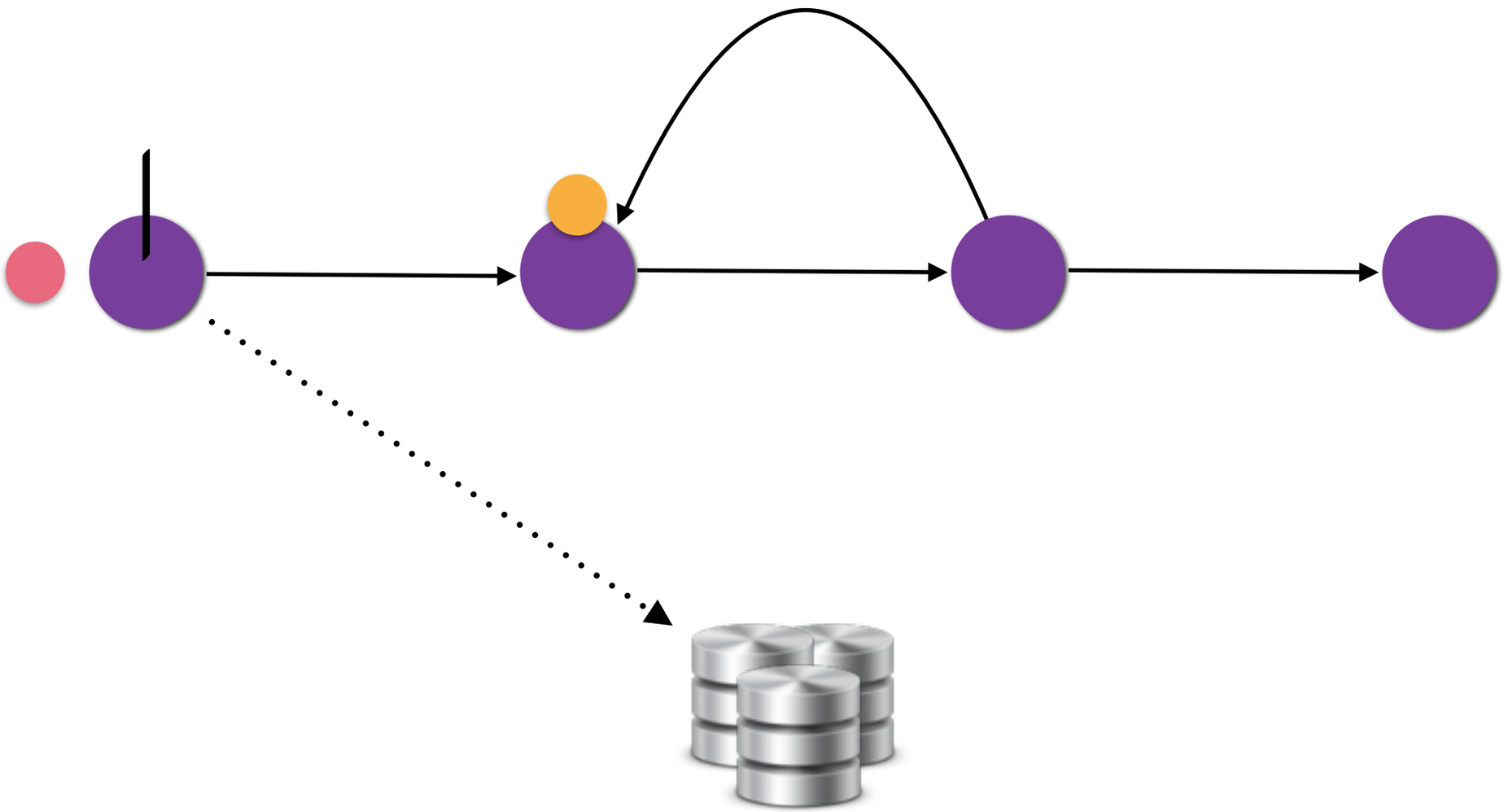
Snapshots on Cyclic Dataflows



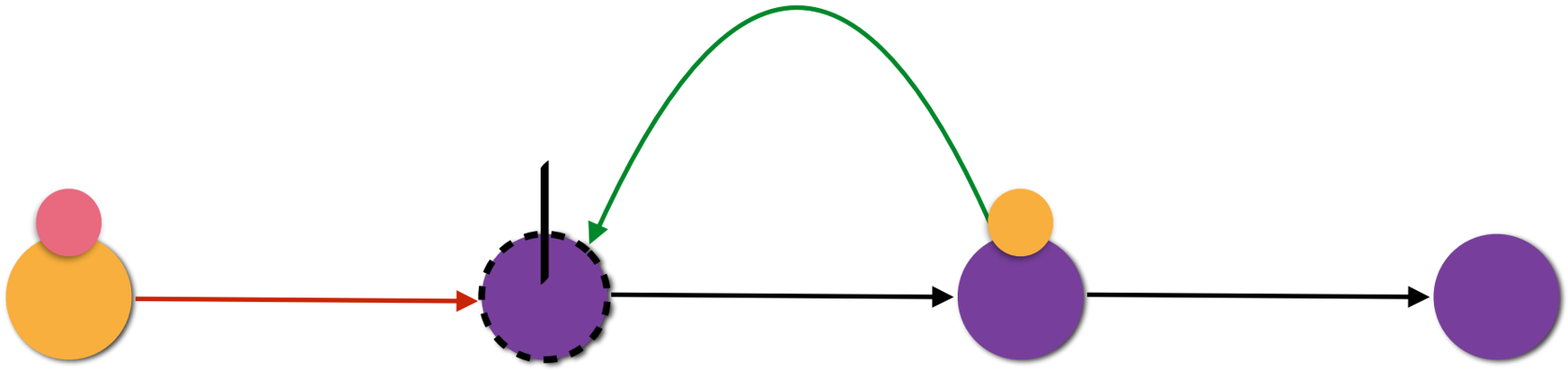
Snapshots on Cyclic Dataflows



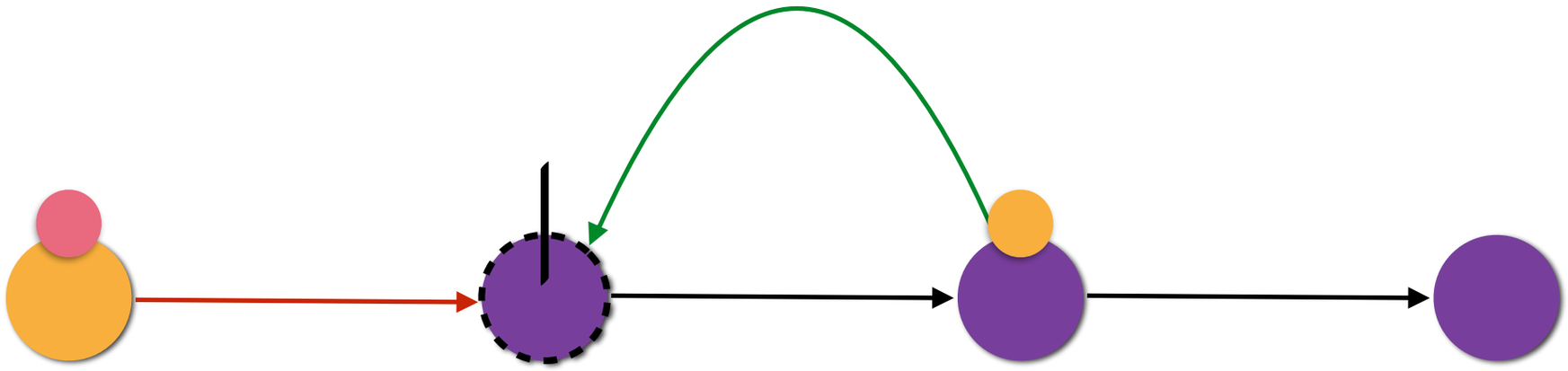
Snapshots on Cyclic Dataflows



Snapshots on Cyclic Dataflows



Snapshots on Cyclic Dataflows



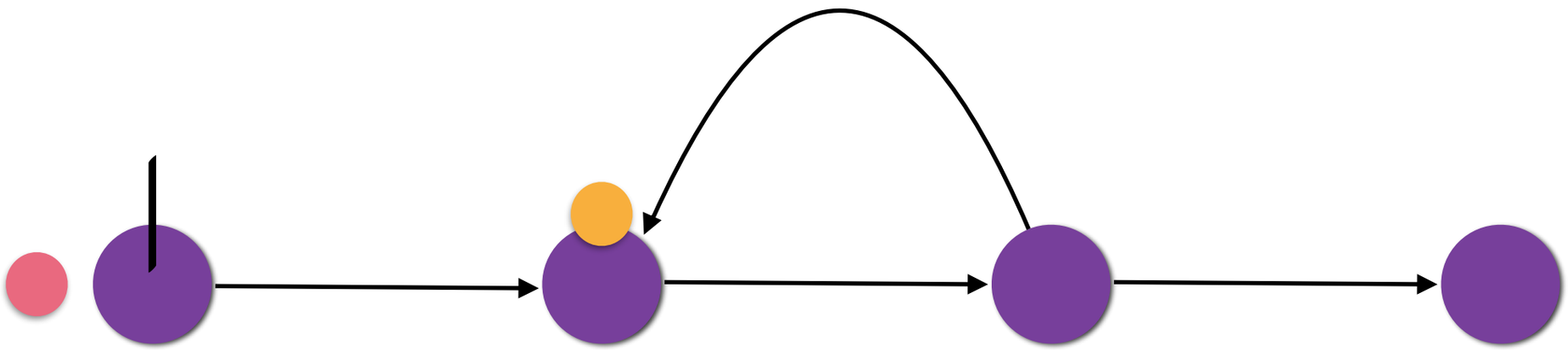
deadlock

Snapshots on Cyclic Dataflows

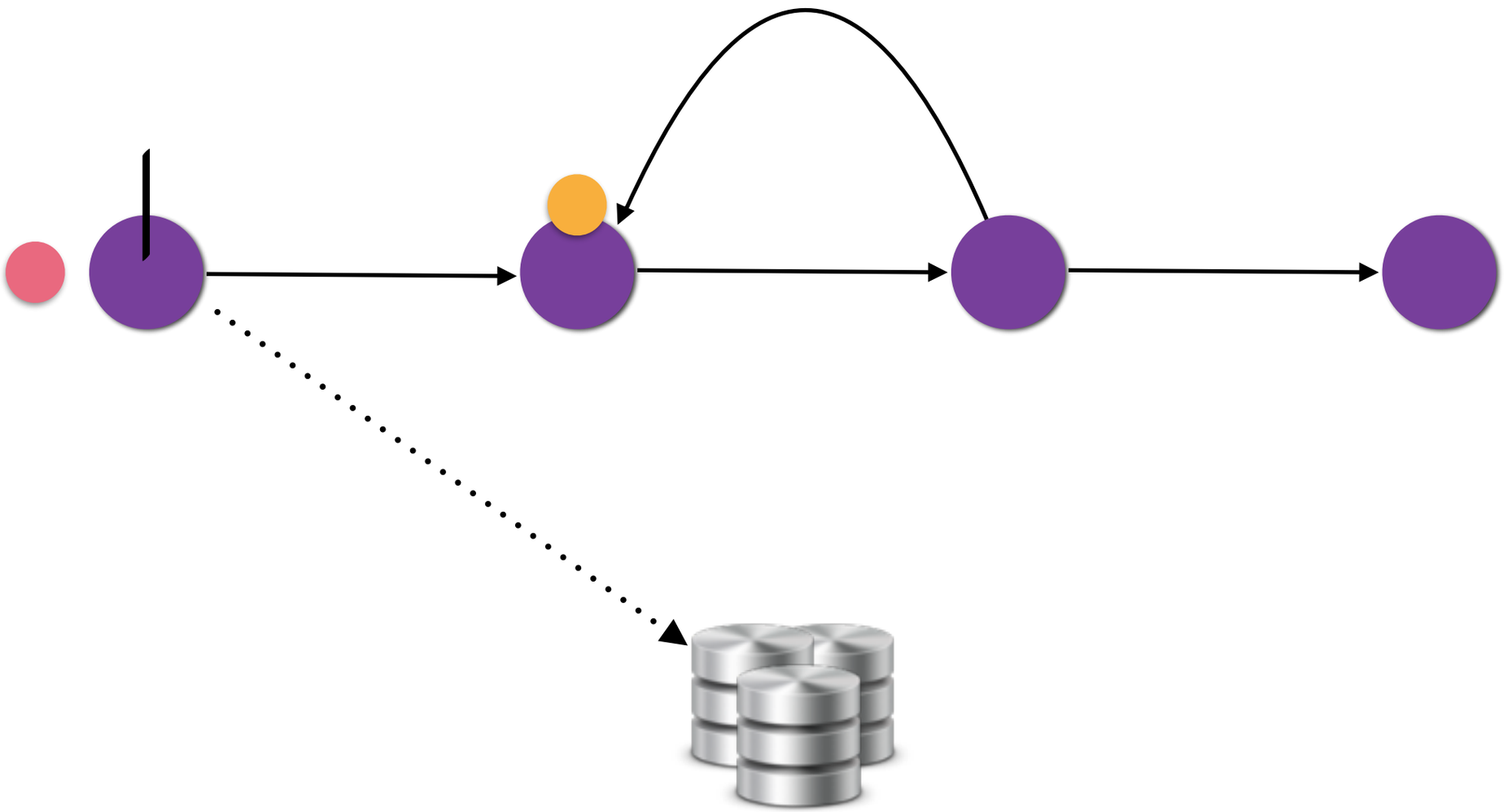
- Checkpointing should eventually terminate

- Records in transit within loops should be included in the checkpoint

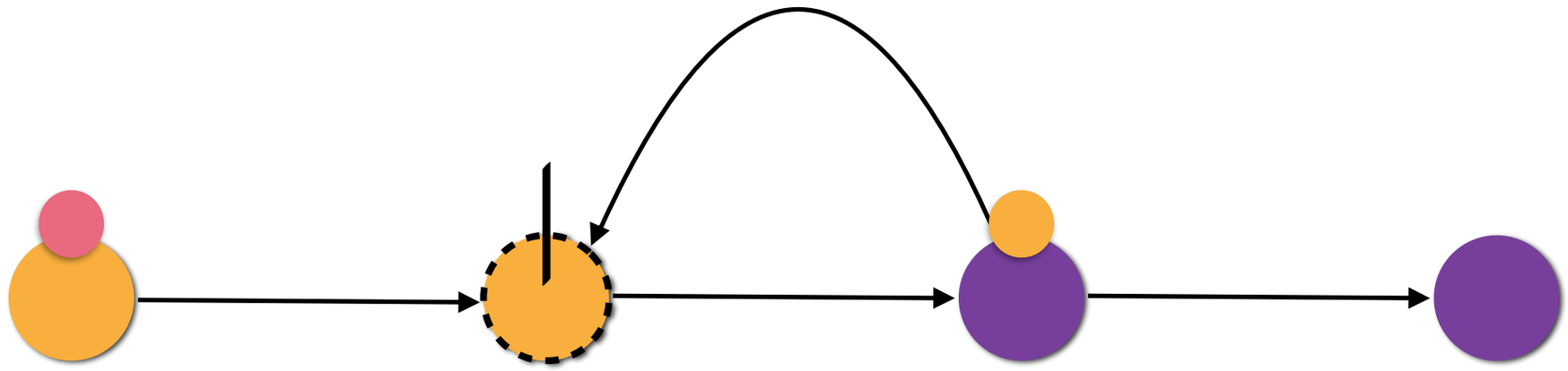
Snapshots on Cyclic Dataflows



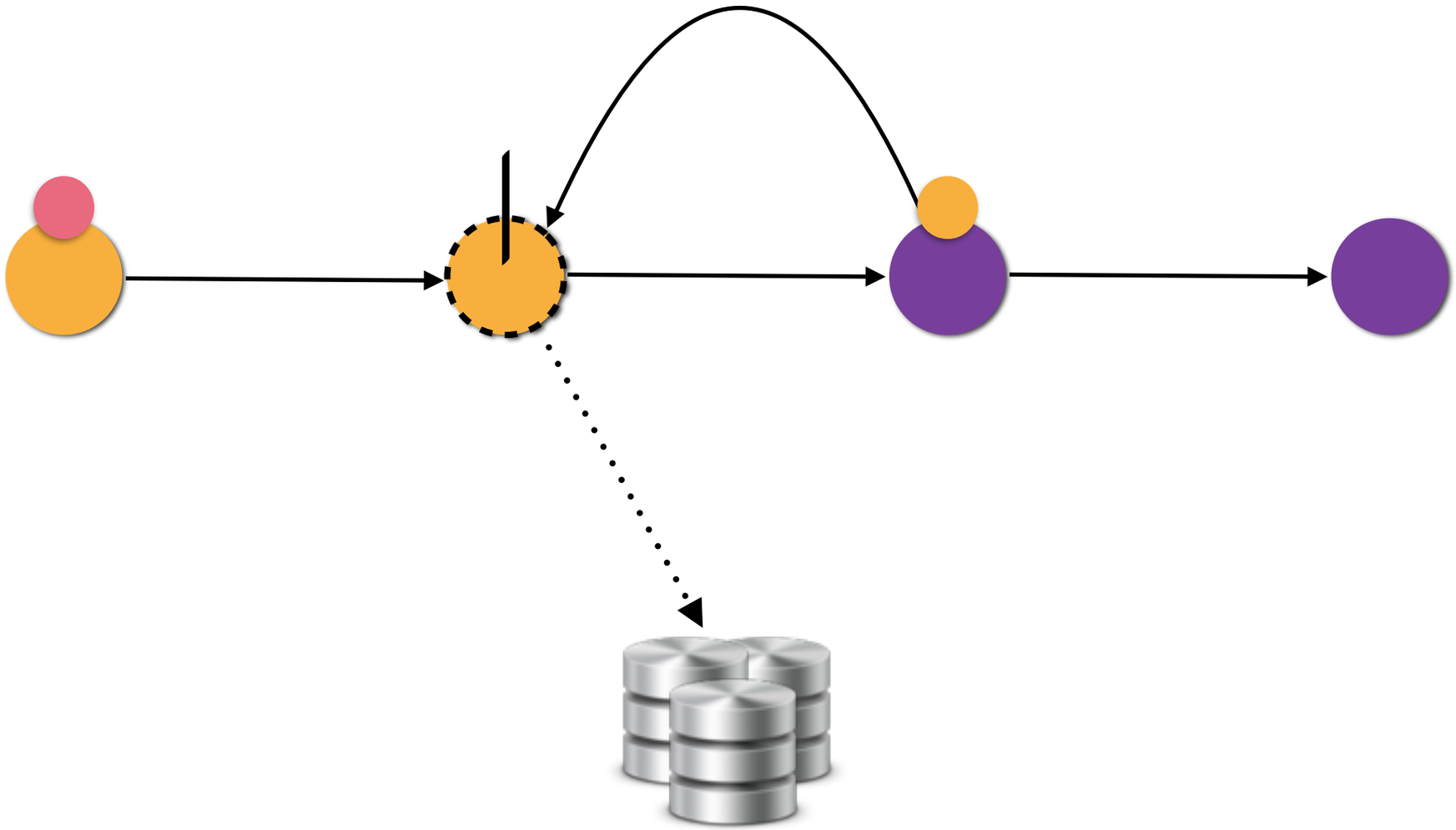
Snapshots on Cyclic Dataflows



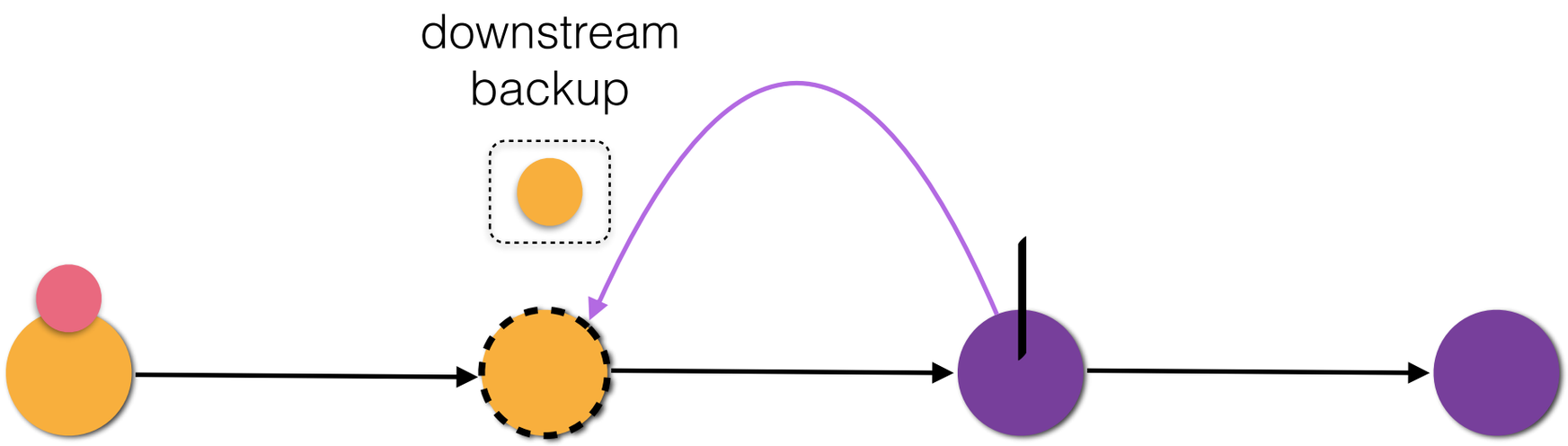
Snapshots on Cyclic Dataflows



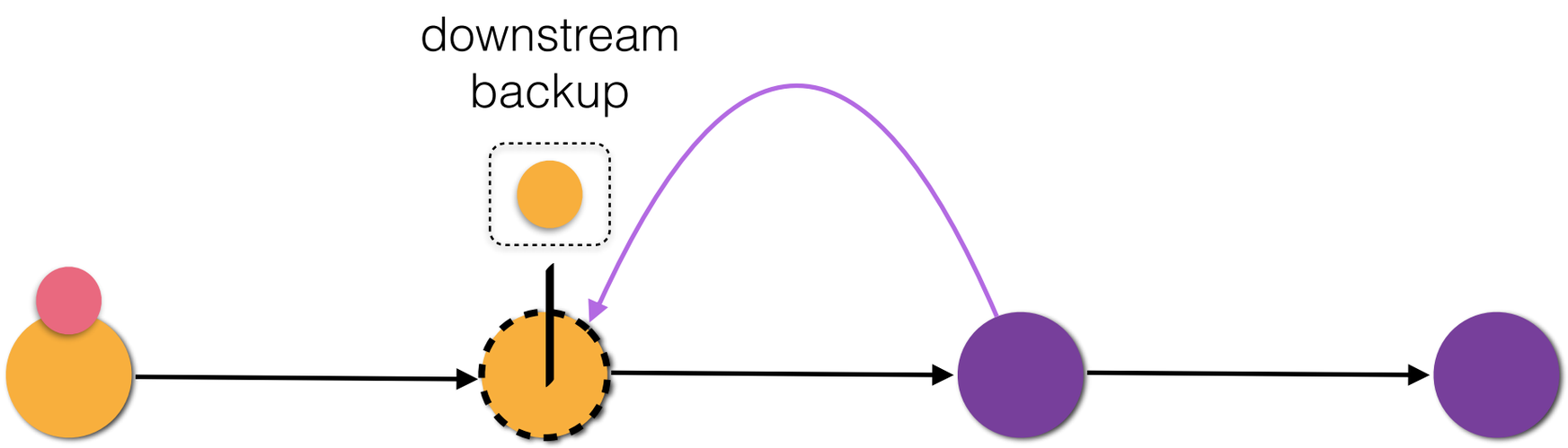
Snapshots on Cyclic Dataflows



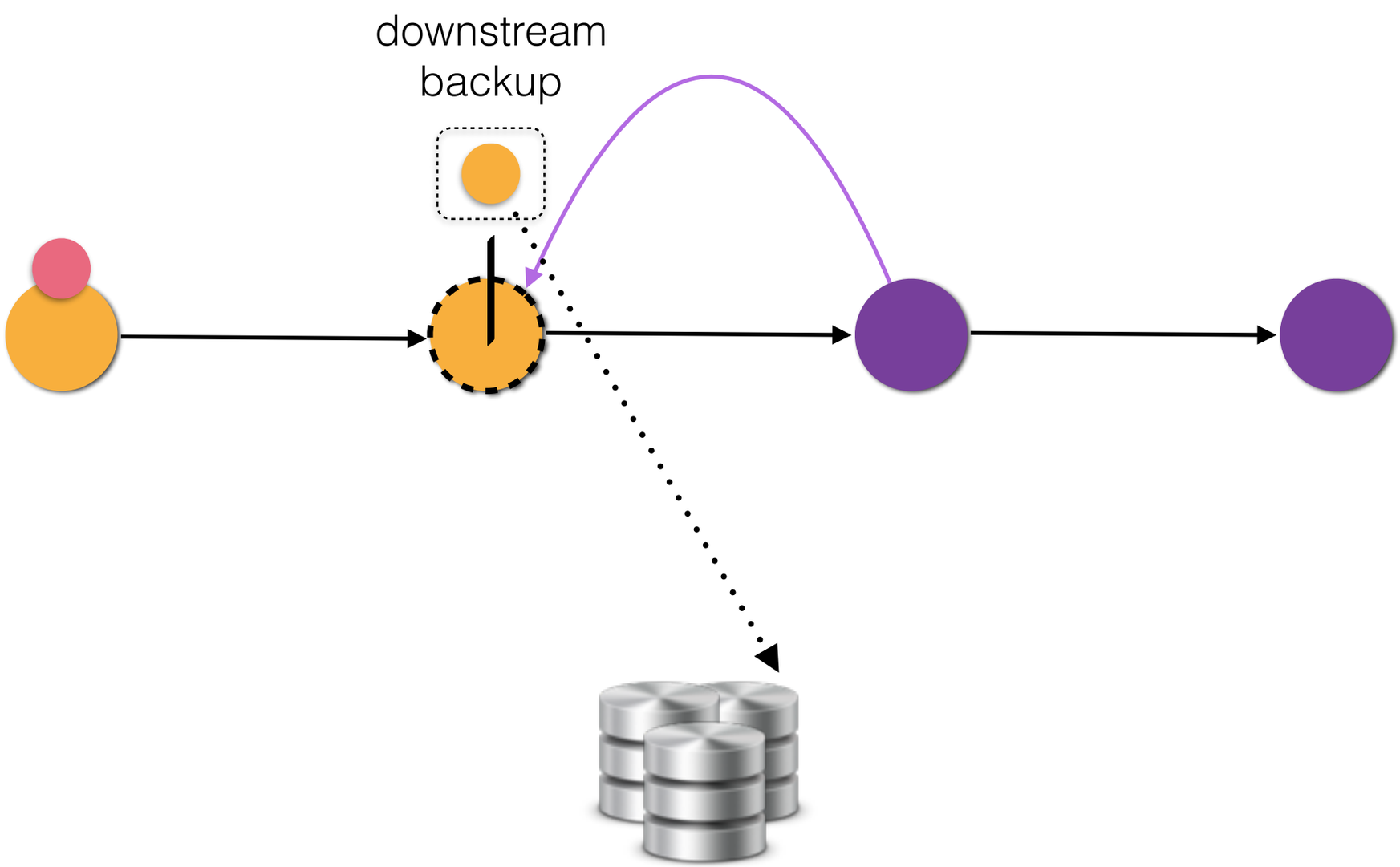
Snapshots on Cyclic Dataflows



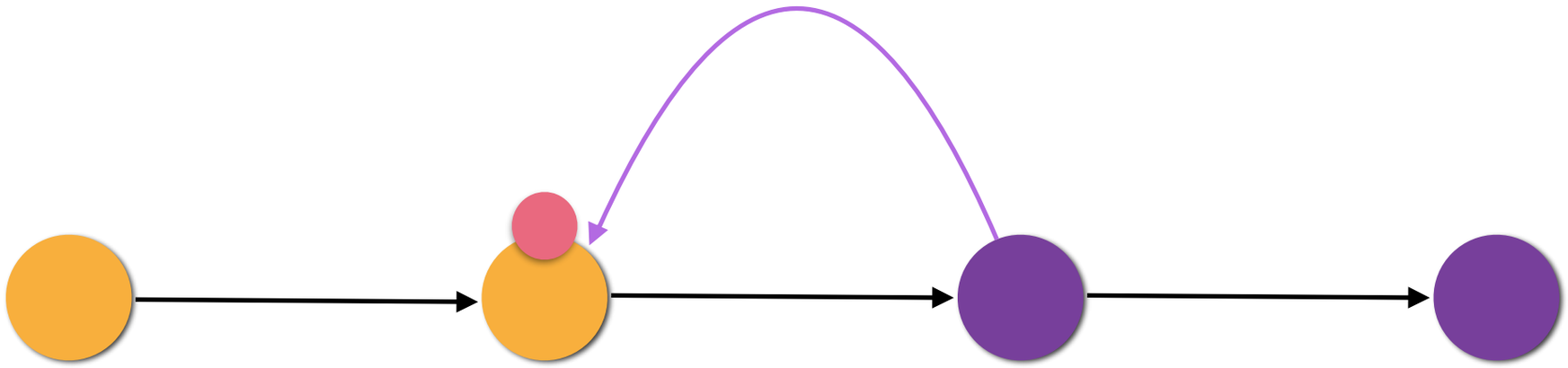
Snapshots on Cyclic Dataflows



Snapshots on Cyclic Dataflows



Snapshots on Cyclic Dataflows



Implementation in Flink

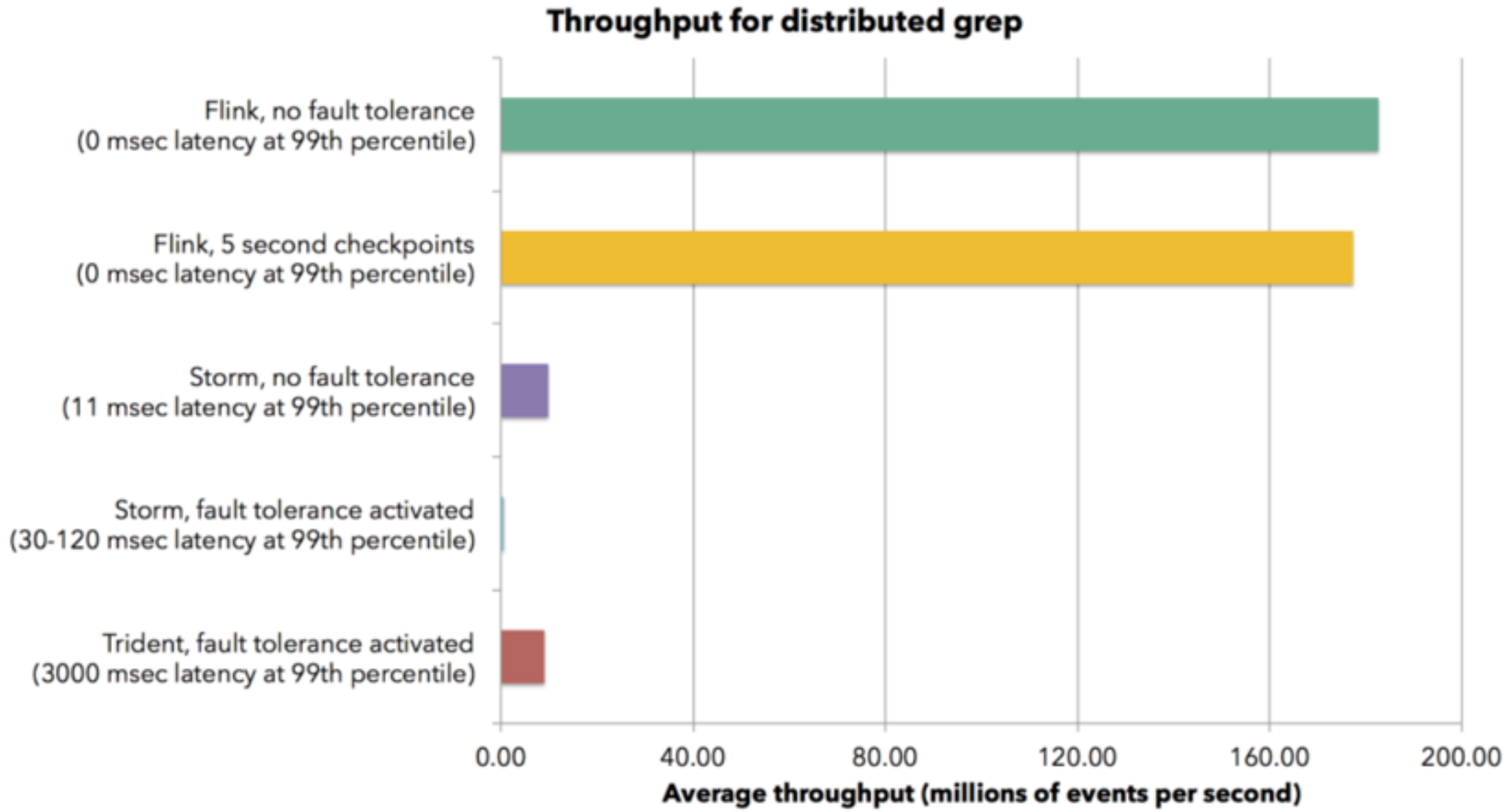
- **Coordinator**/Driver Actor per job in JM
  - sends periodic markers to sources
  - collects acknowledgements with state handles and registers complete snapshots
  - injects references to latest consistent state handles and back-edge records in pending execution graph tasks
- **Tasks**
  - snapshot state transparently in given backend upon receiving a barrier and send ack to coordinator
  - propagate barriers further like normal records

Recovery

- **Full** rescheduling of the execution graph from the latest checkpoint - simple, no further modifications

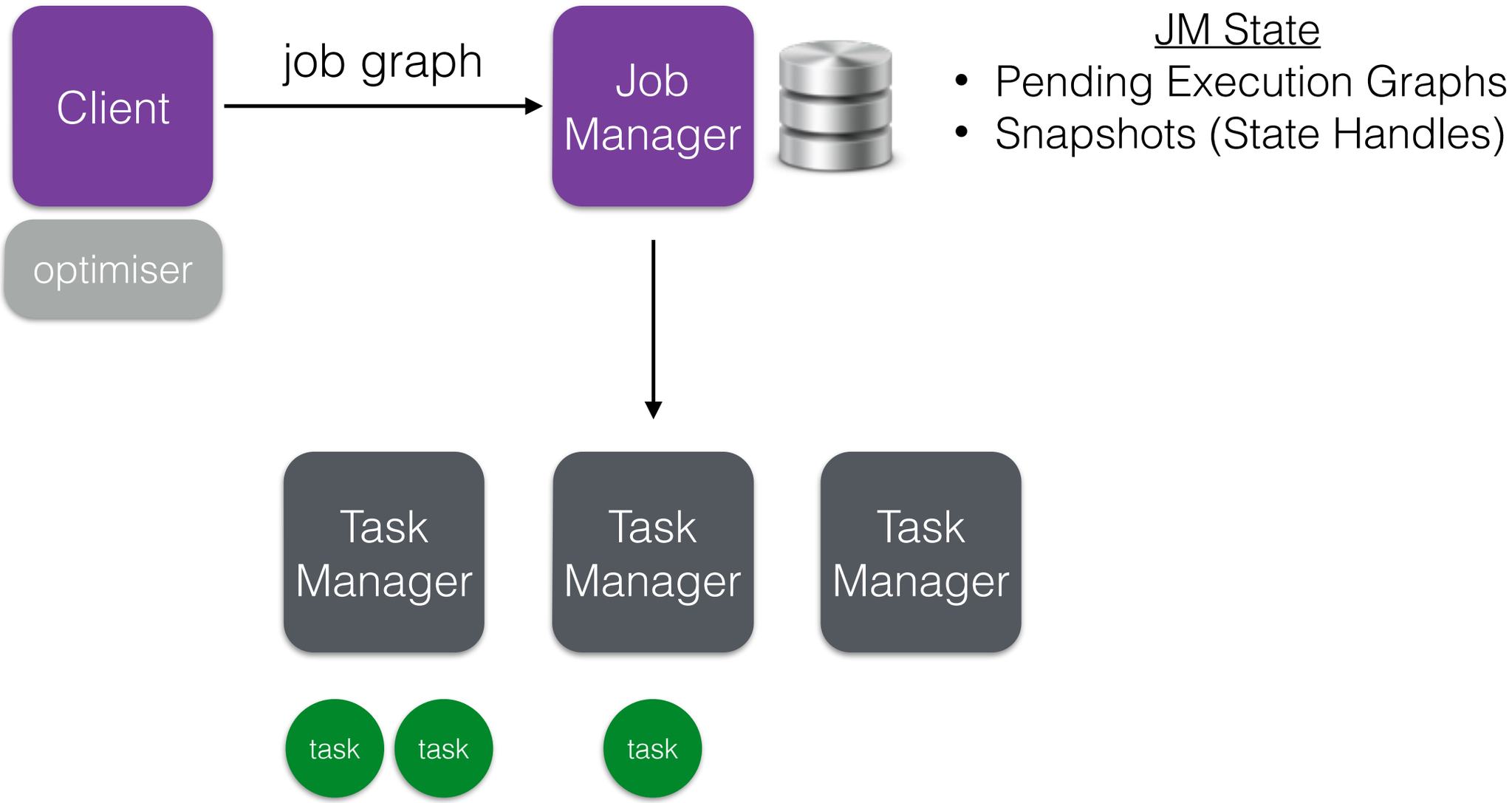
- **Partial** rescheduling of the execution graph for all upstream dependencies - trickier, duplicate elimination should be guaranteed

Performance Impact



http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/

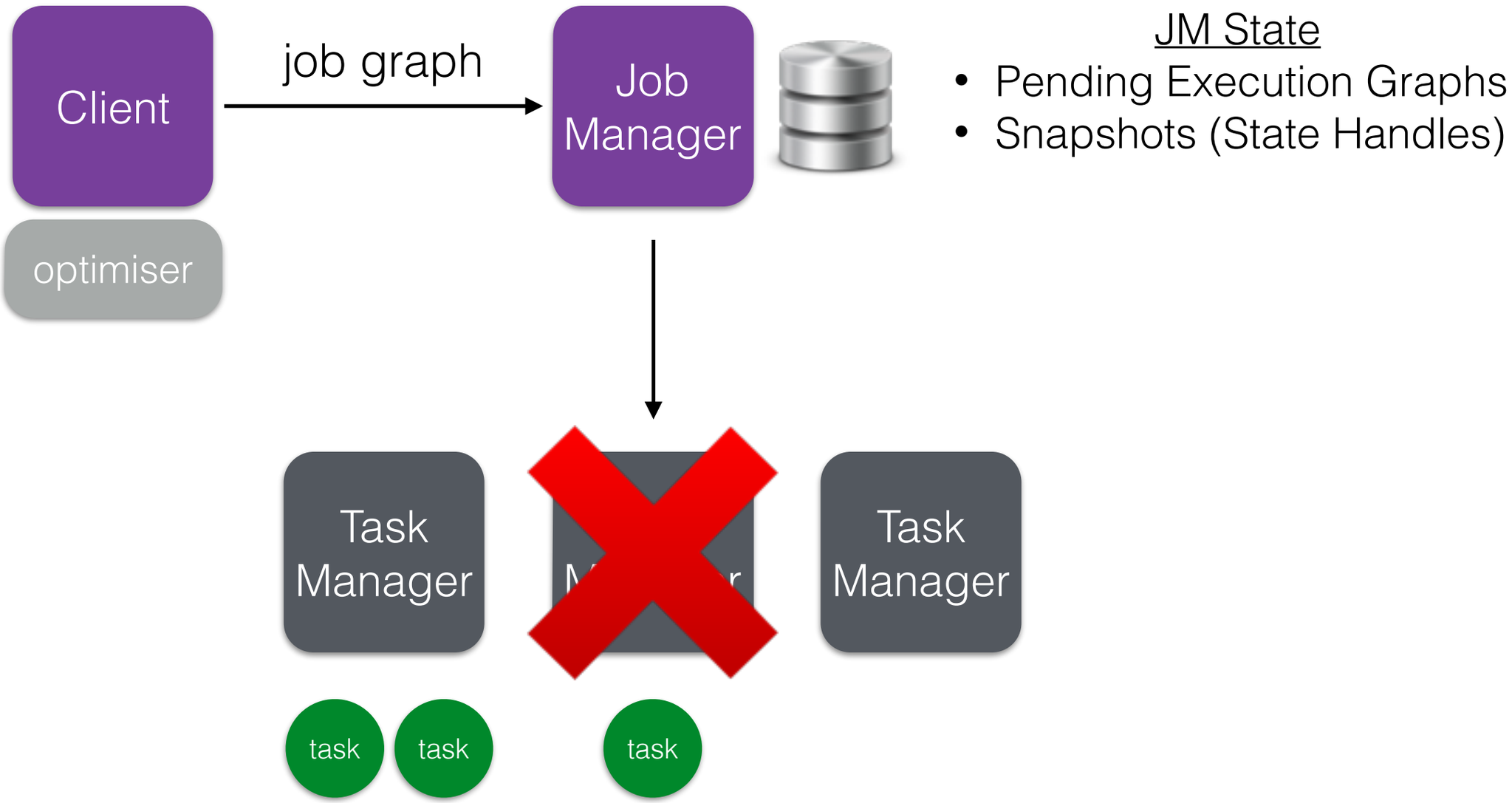
Node Failures



JM State

- Pending Execution Graphs
- Snapshots (State Handles)

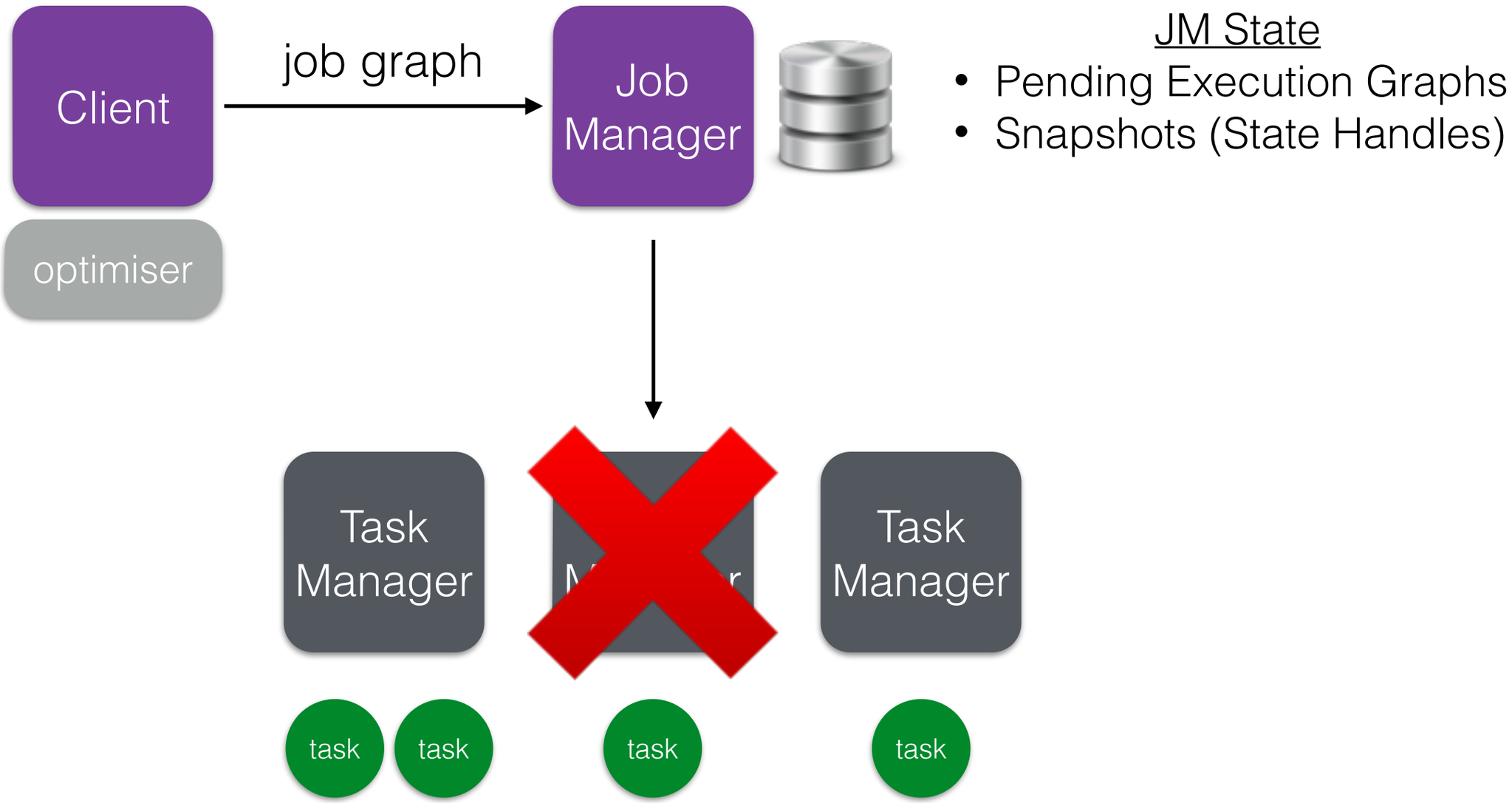
Node Failures



JM State

- Pending Execution Graphs
- Snapshots (State Handles)

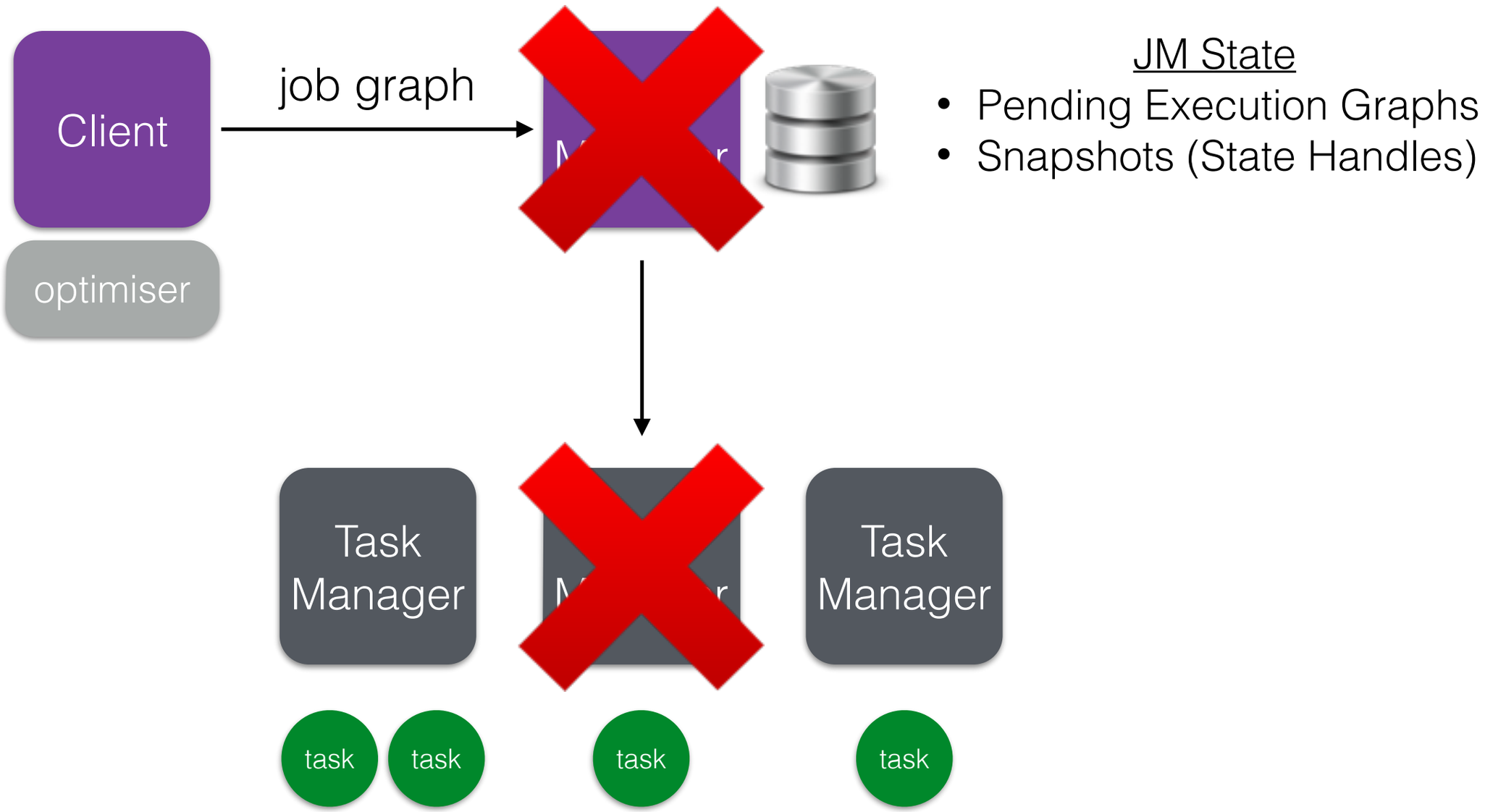
Node Failures



JM State

- Pending Execution Graphs
- Snapshots (State Handles)

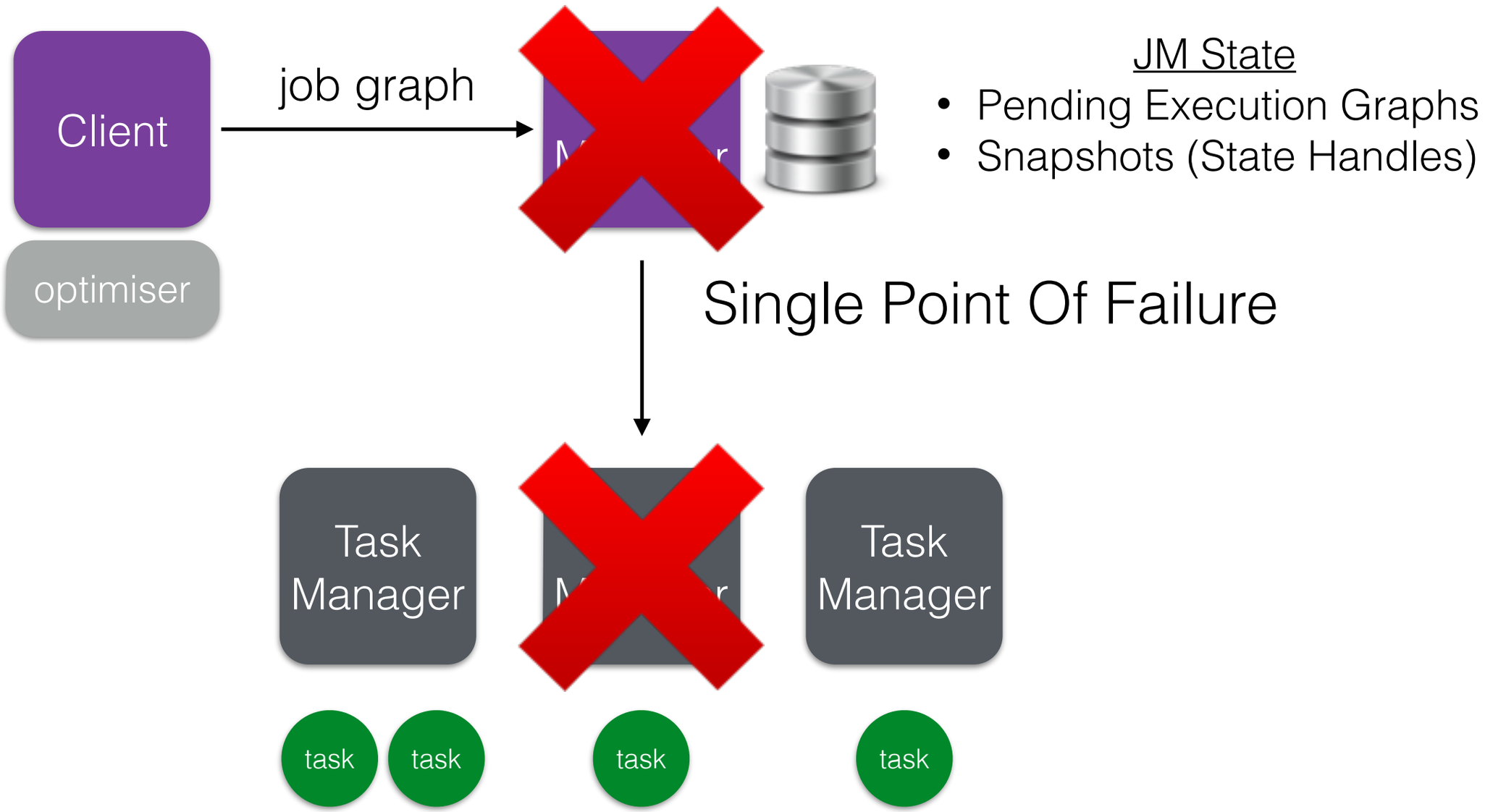
Node Failures



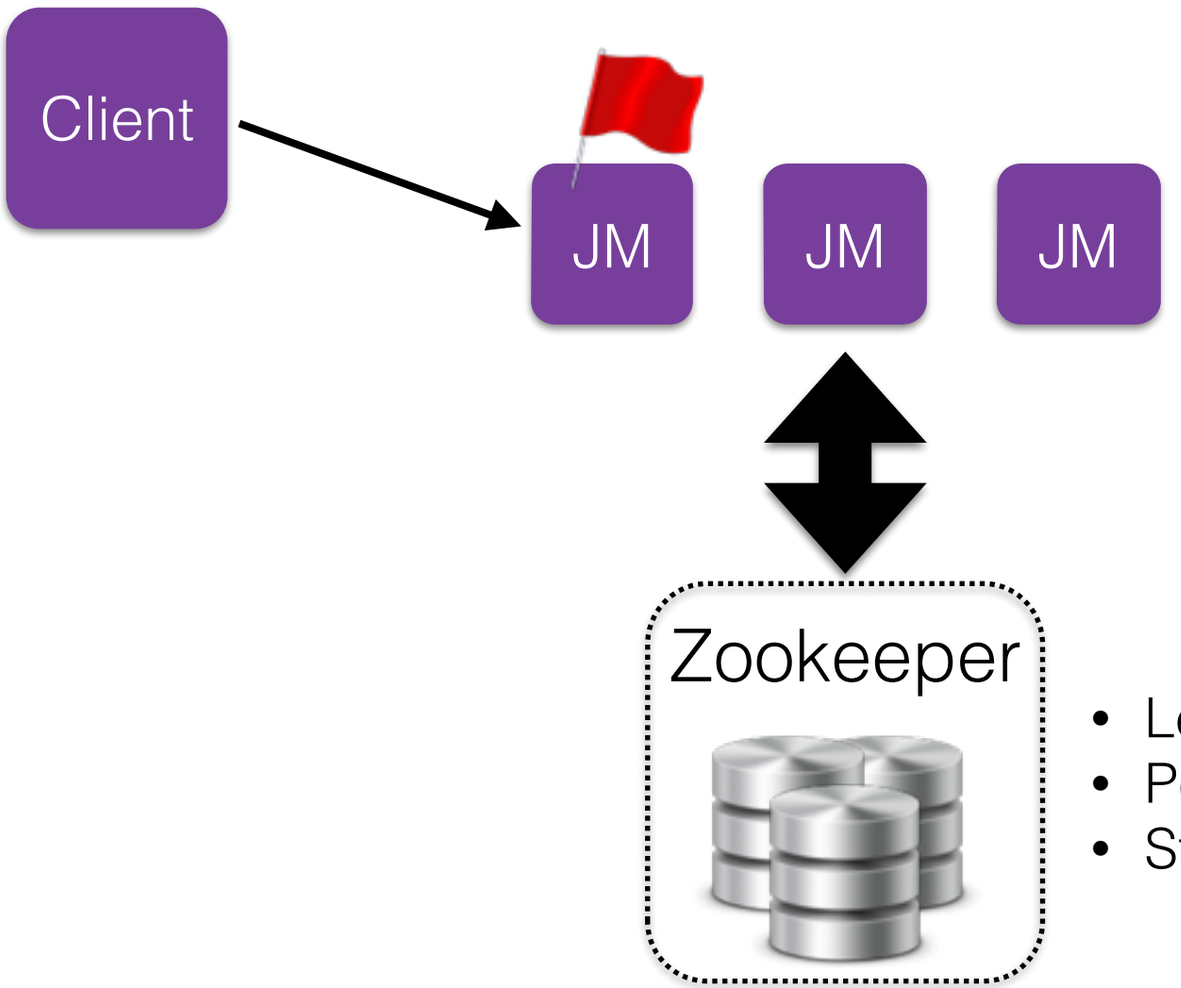
JM State

- Pending Execution Graphs
- Snapshots (State Handles)

Node Failures



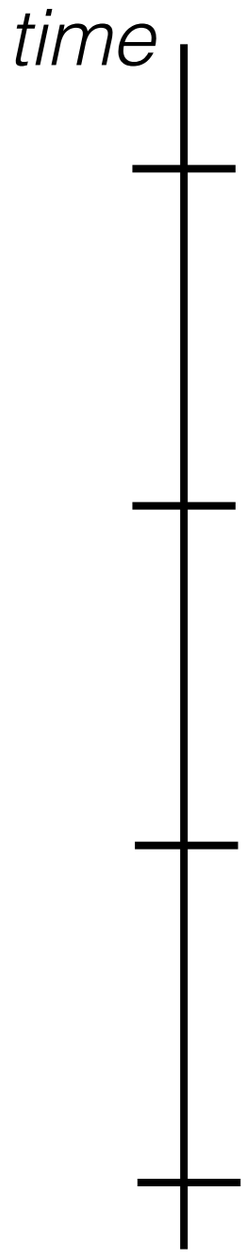
Passive Standby JM



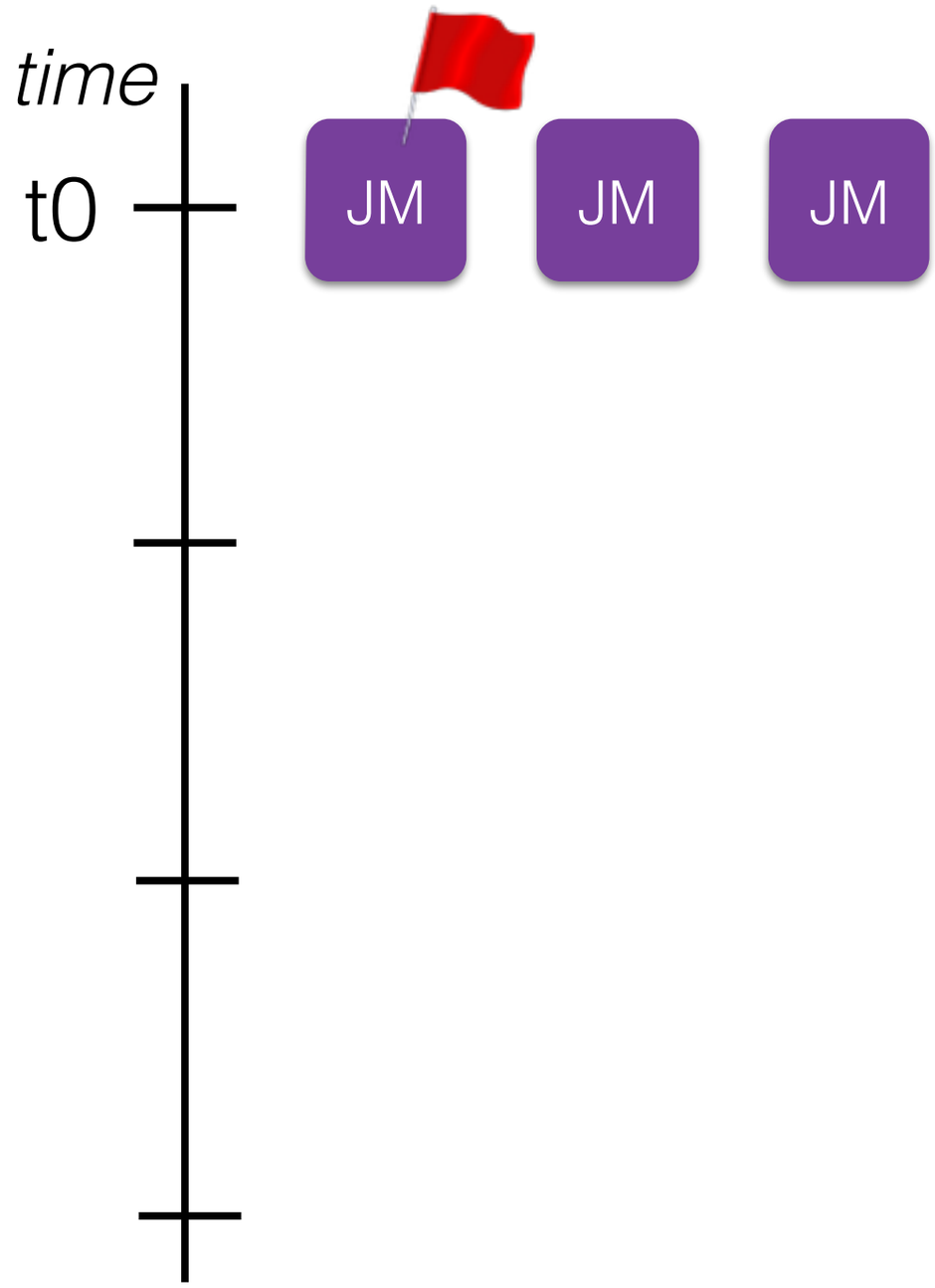
State

- Leader JM Address
- Pending Execution Graphs
- State Snapshots

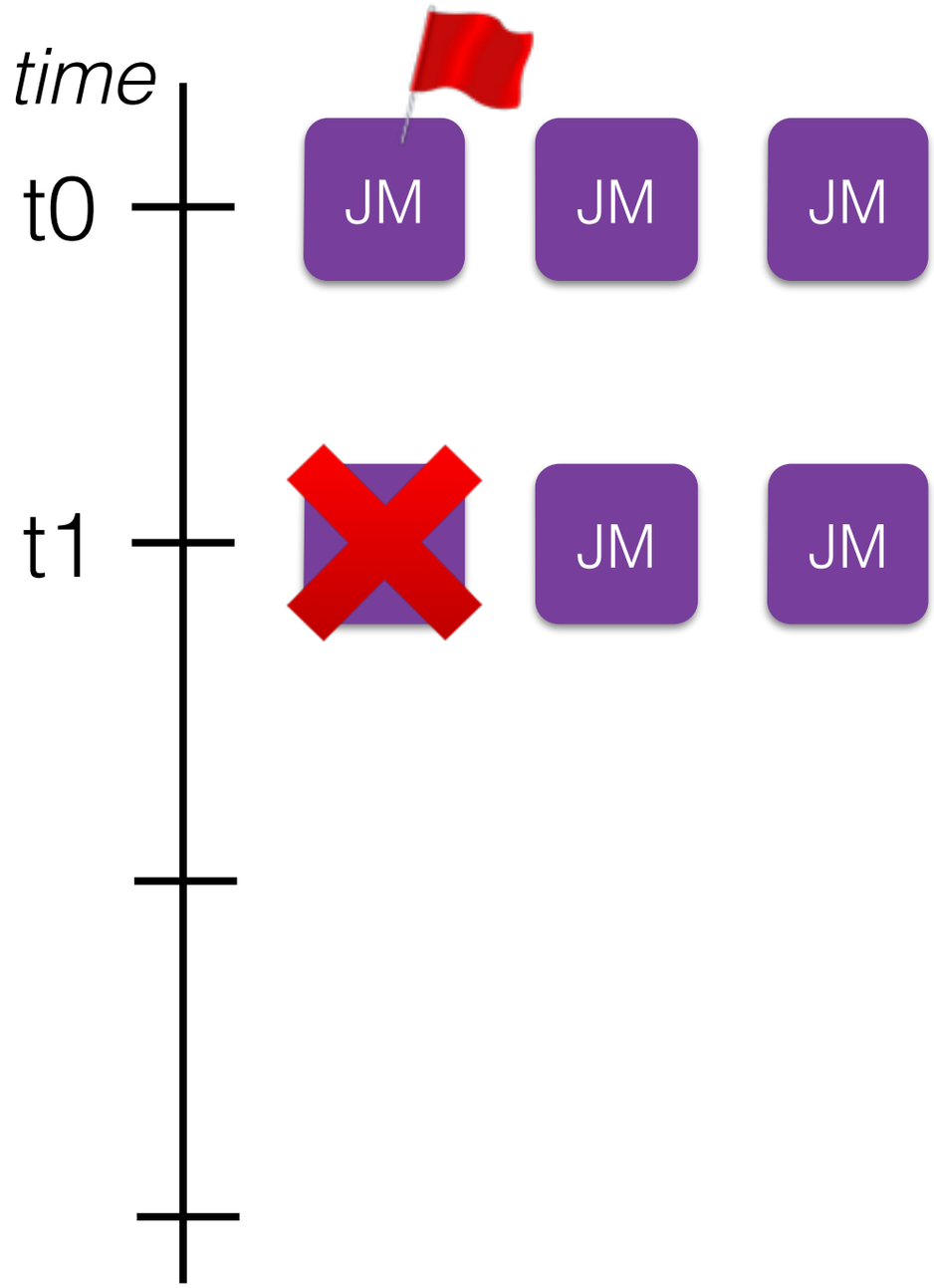
Eventual Leader Election



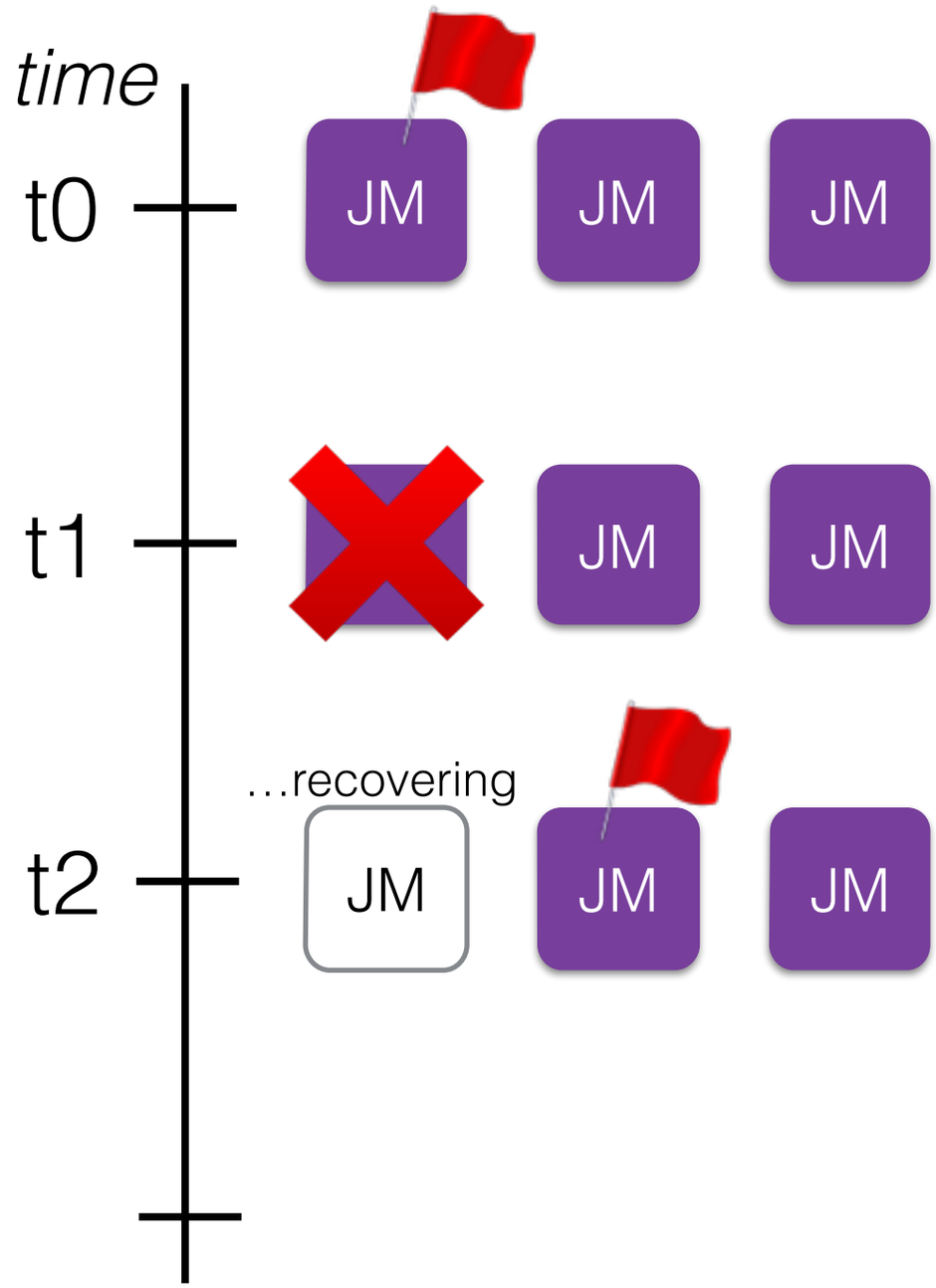
Eventual Leader Election



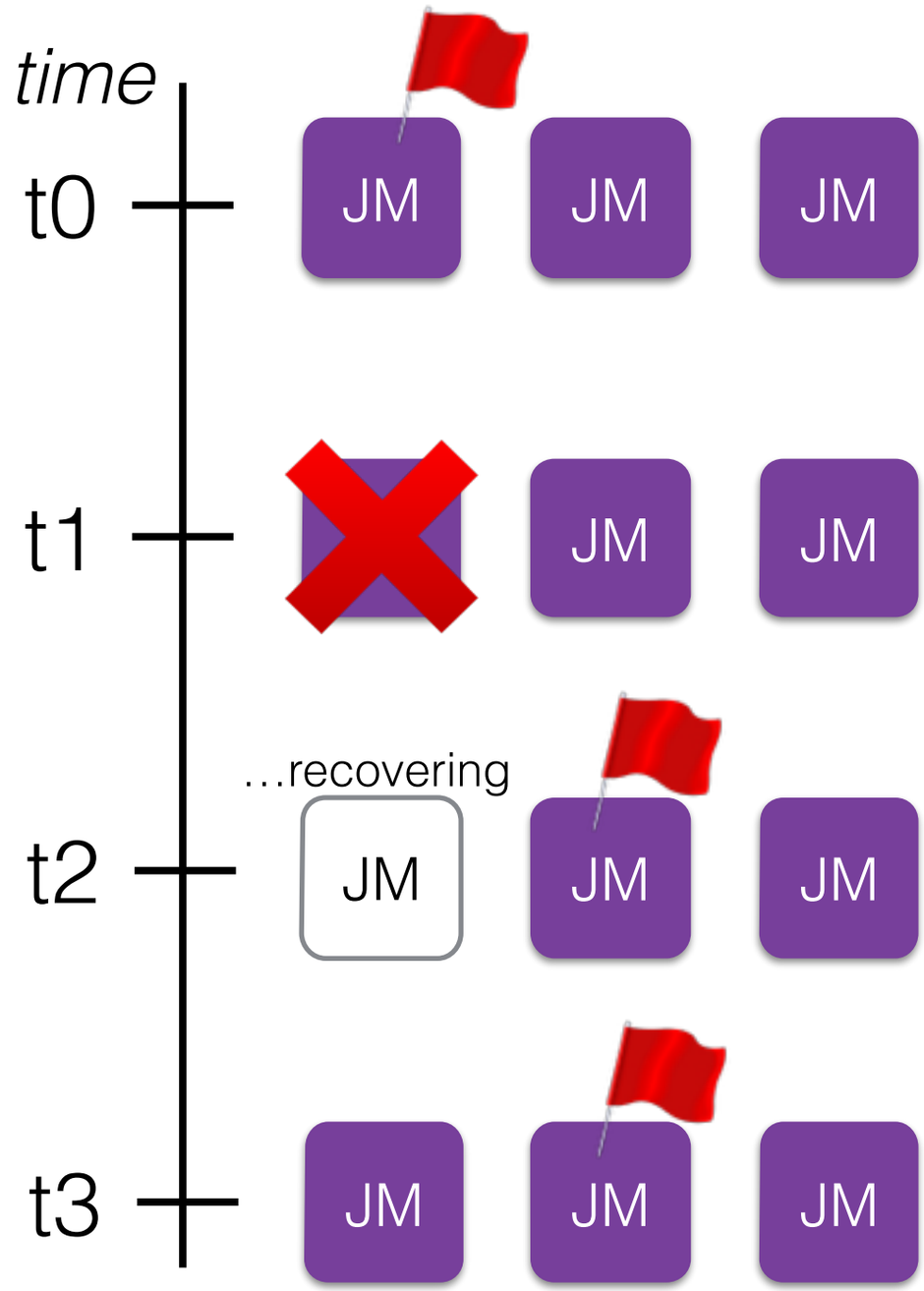
Eventual Leader Election



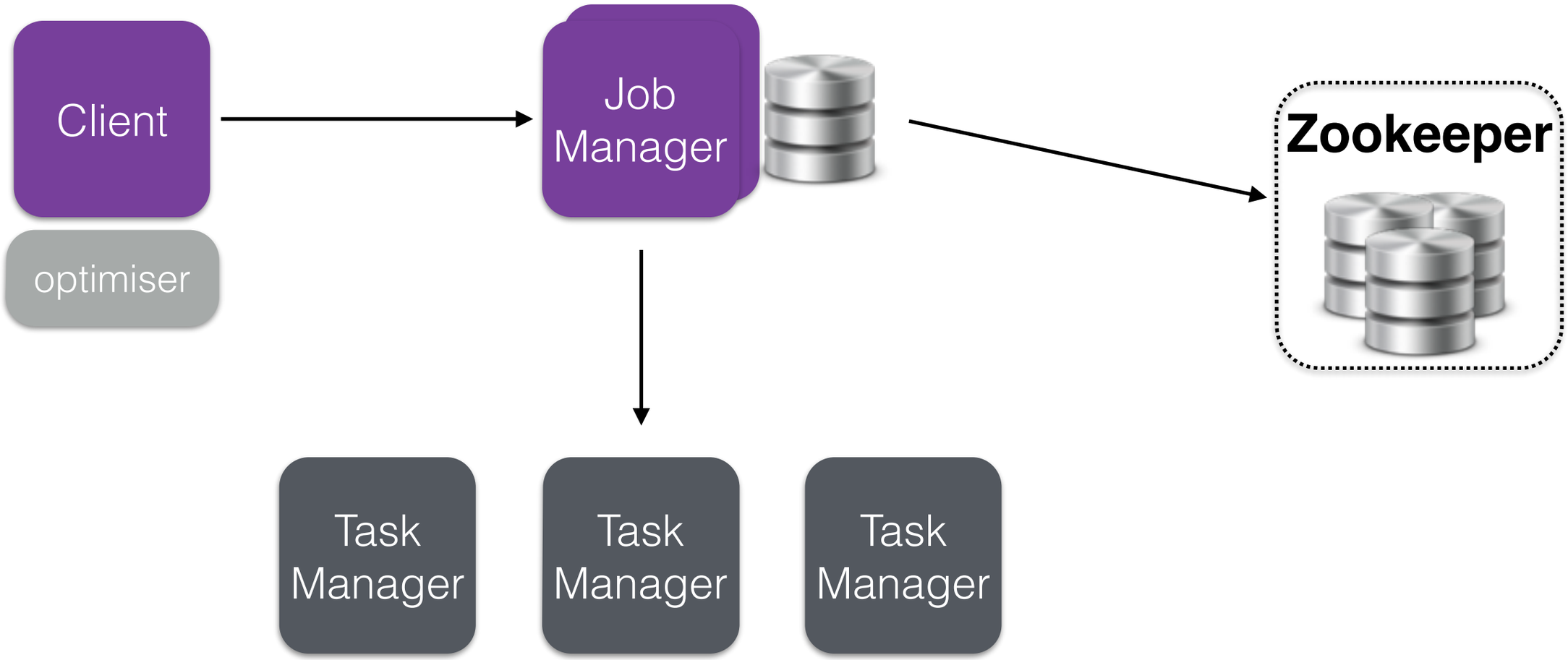
Eventual Leader Election



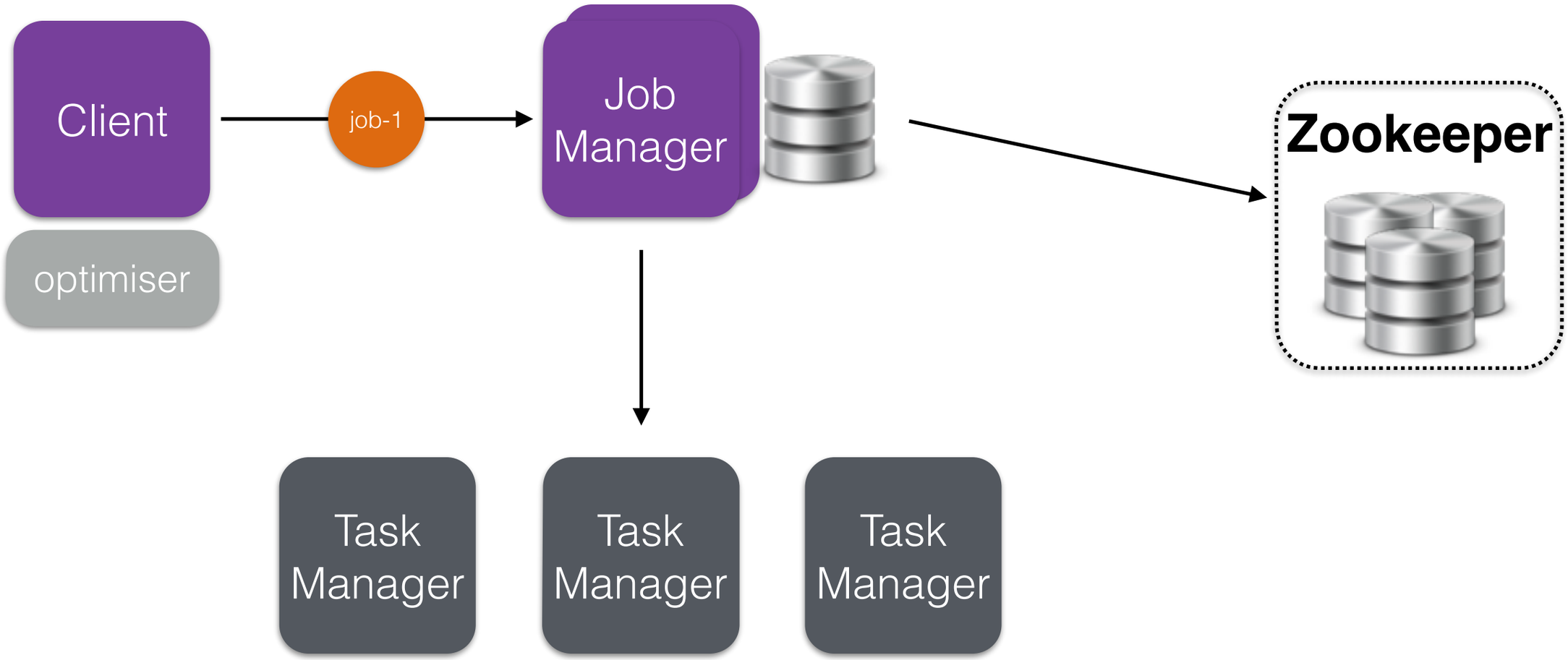
Eventual Leader Election



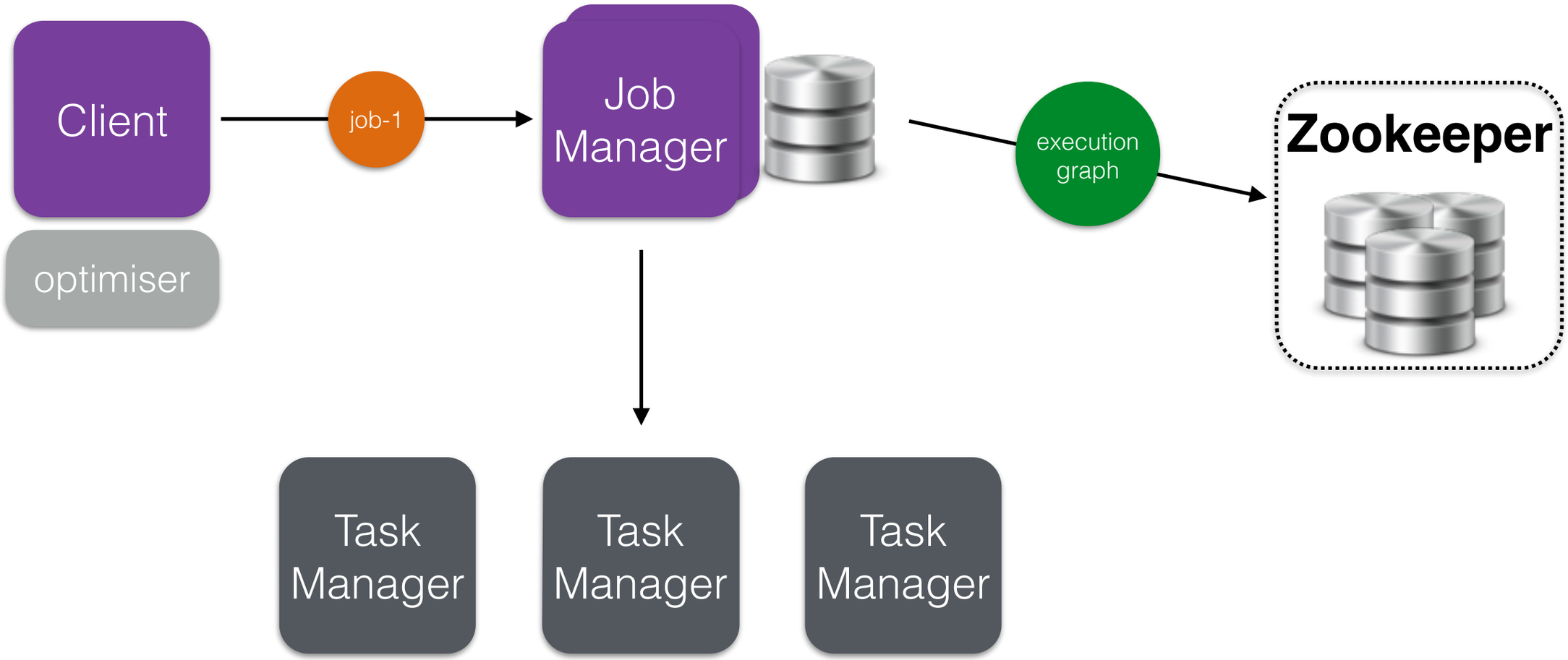
Scheduling Jobs



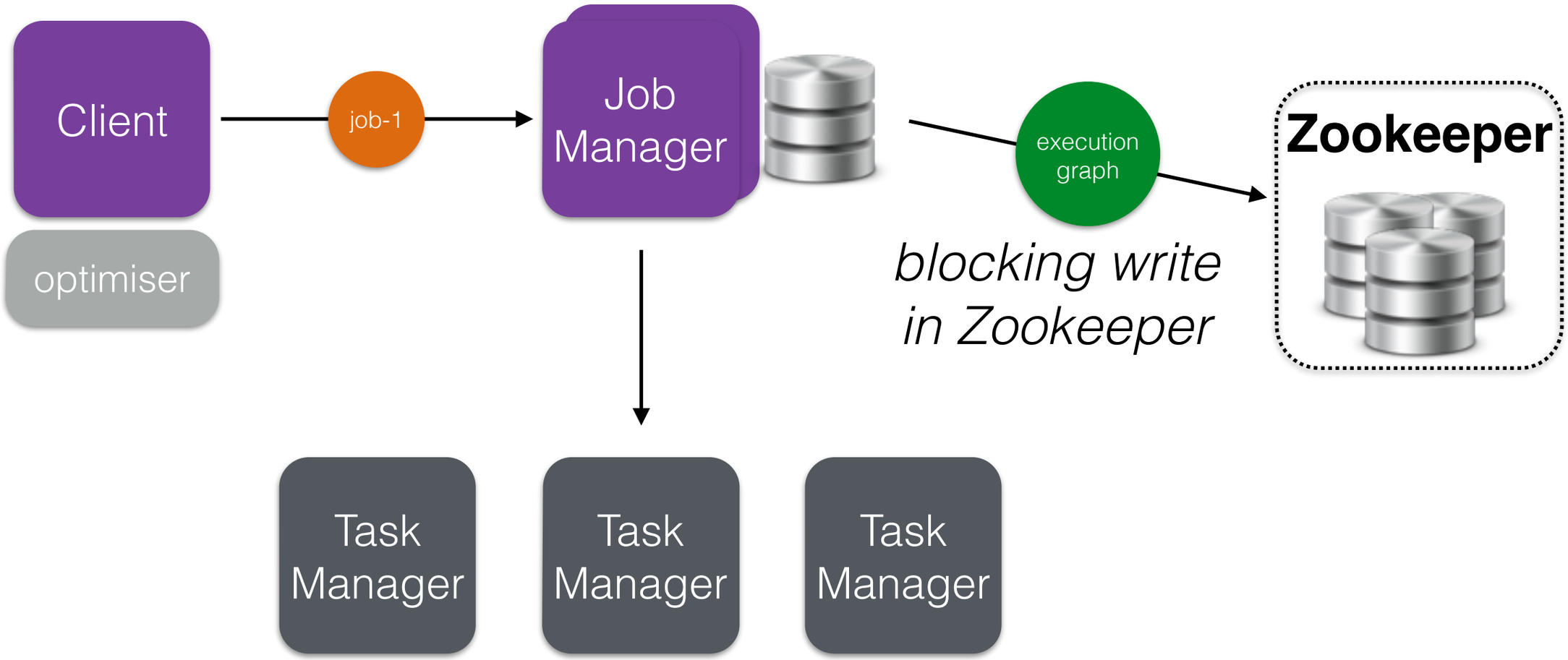
Scheduling Jobs



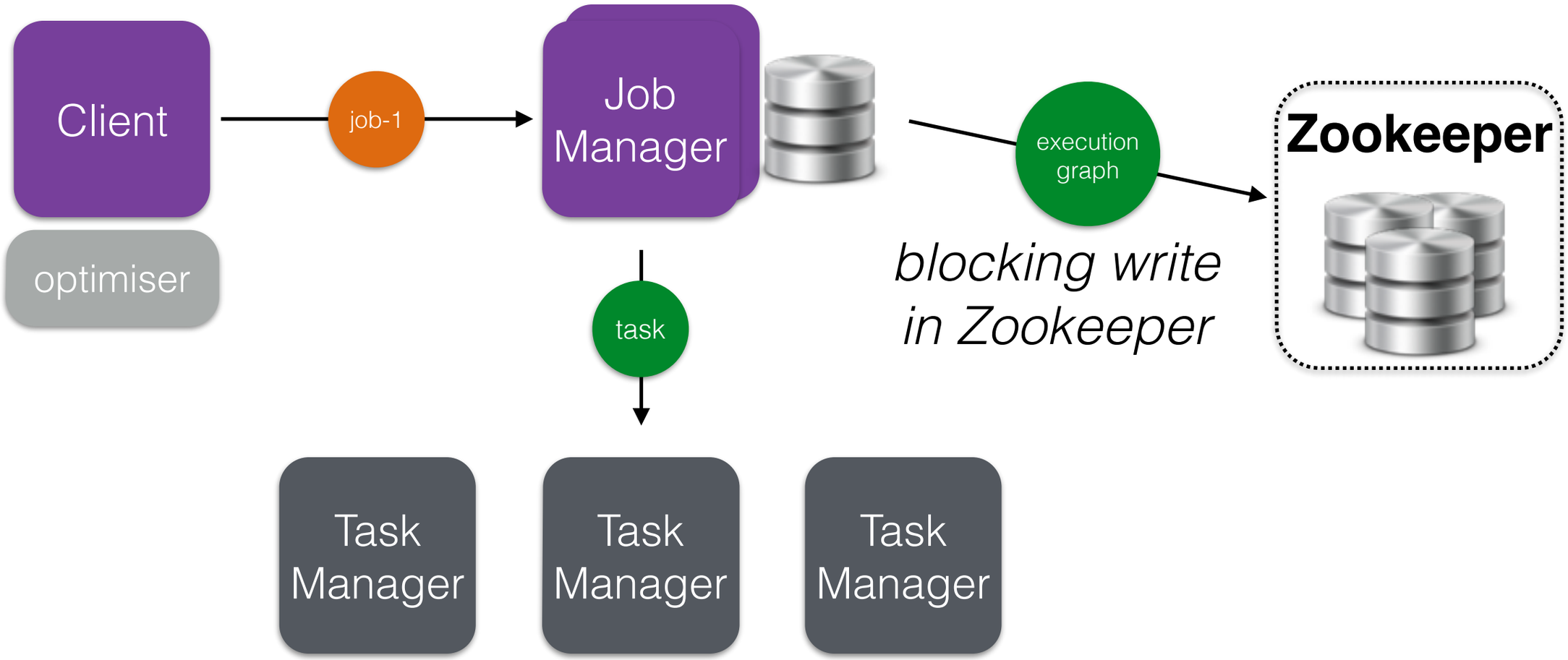
Scheduling Jobs



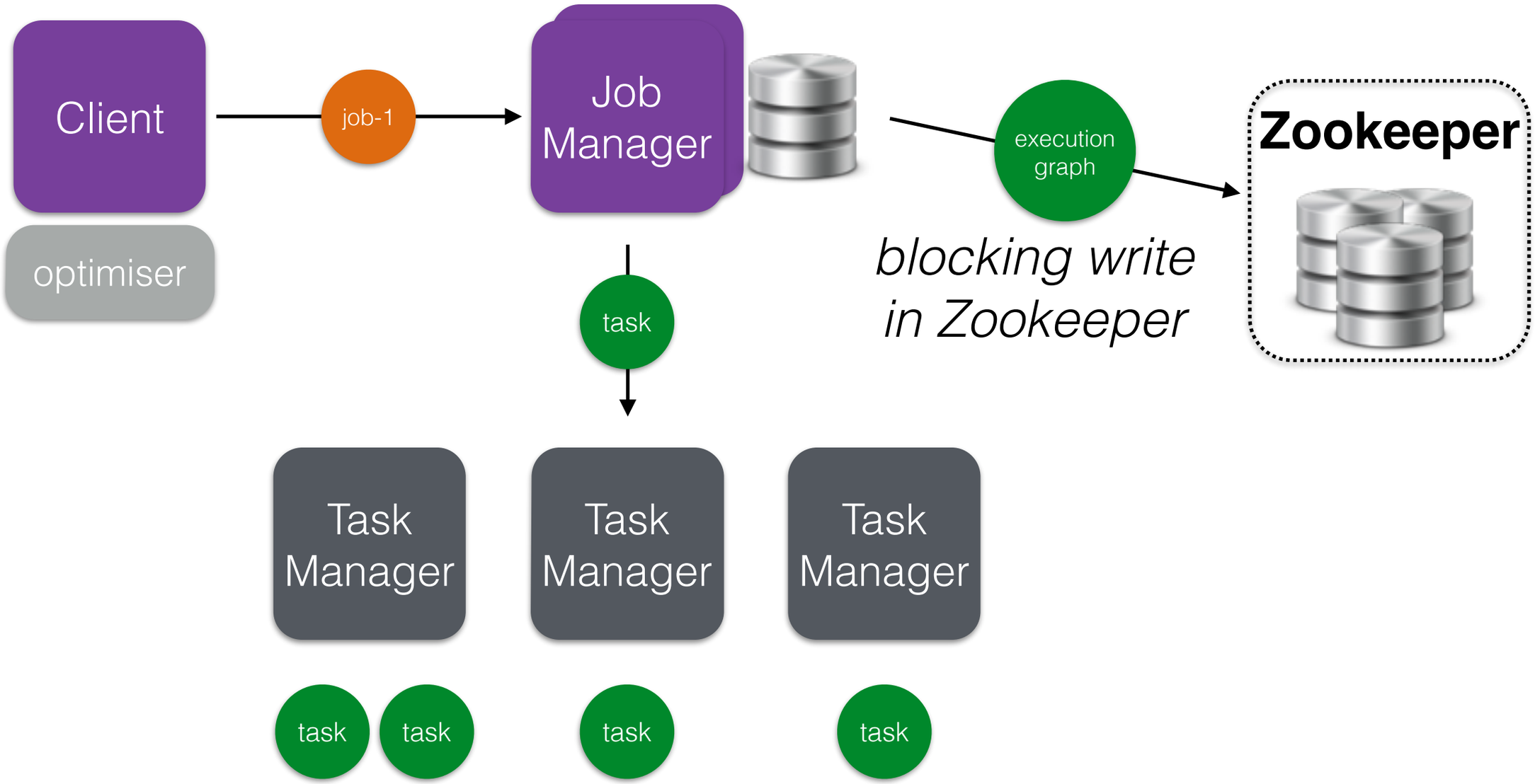
Scheduling Jobs



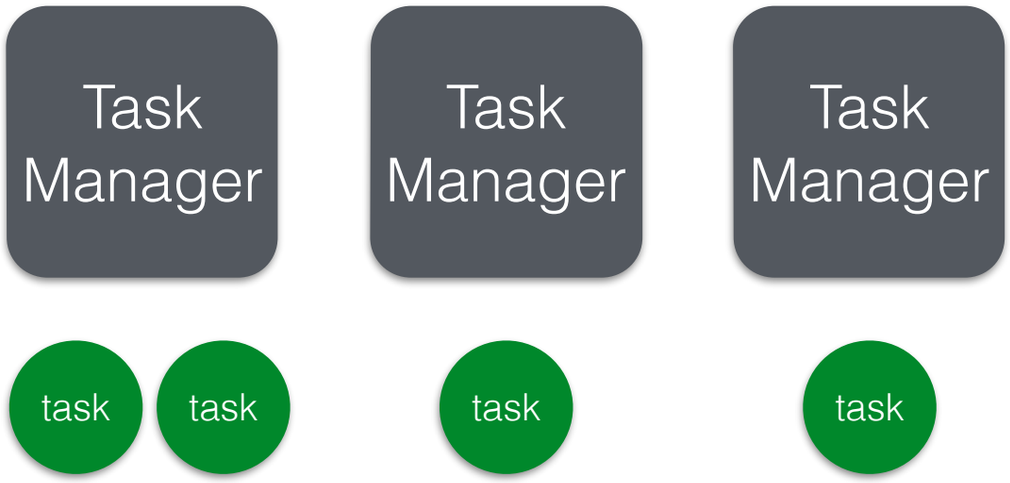
Scheduling Jobs



Scheduling Jobs



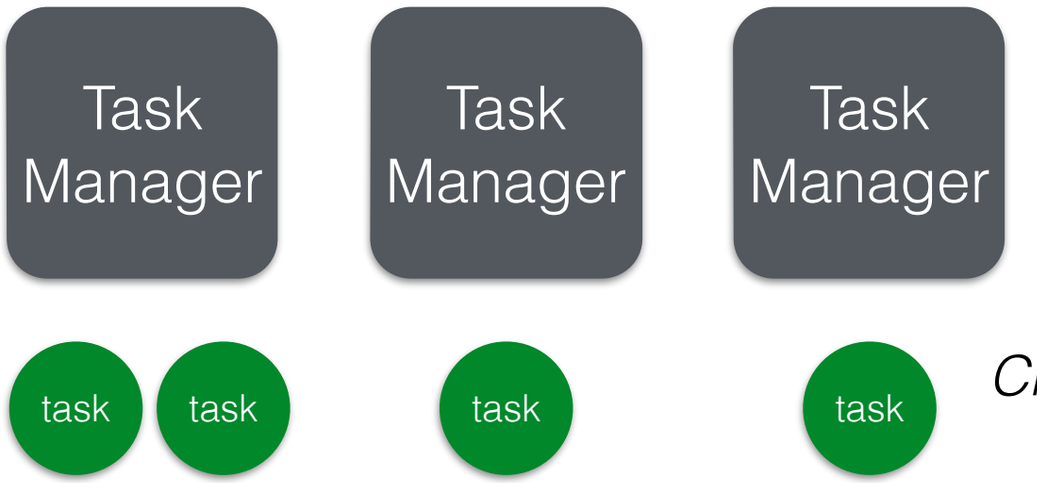
Logging Snapshots



asynchronous writes in Zookeeper

State Backend

Logging Snapshots



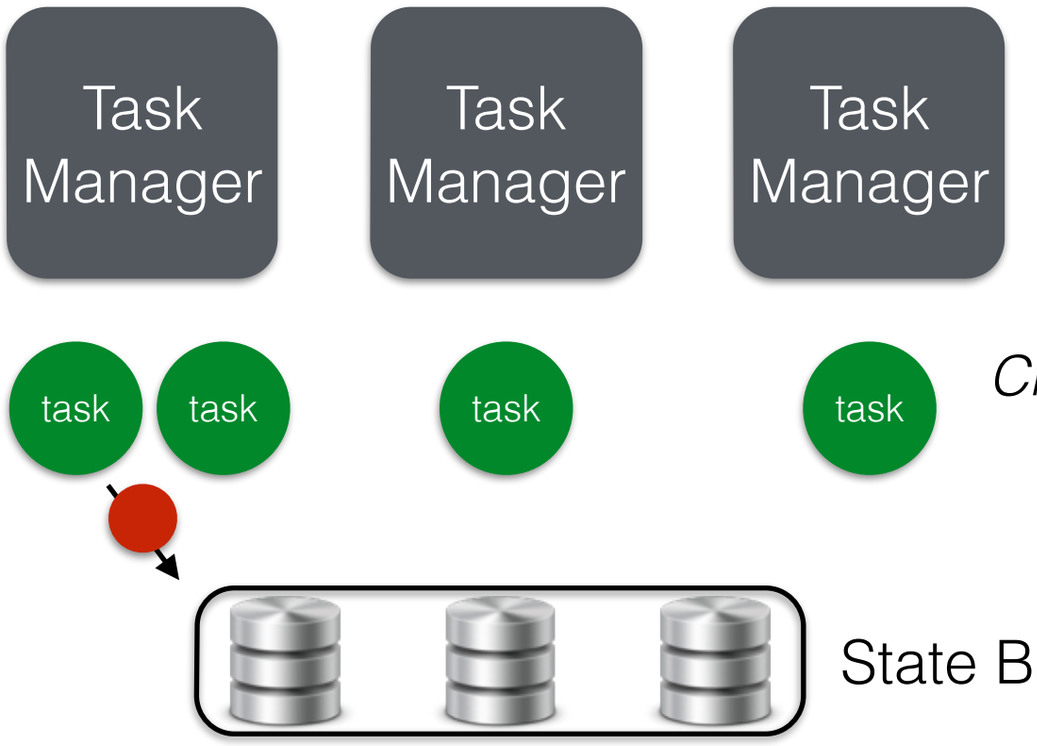
asynchronous writes in Zookeeper

Checkpointing states

State Backend

Logging Snapshots

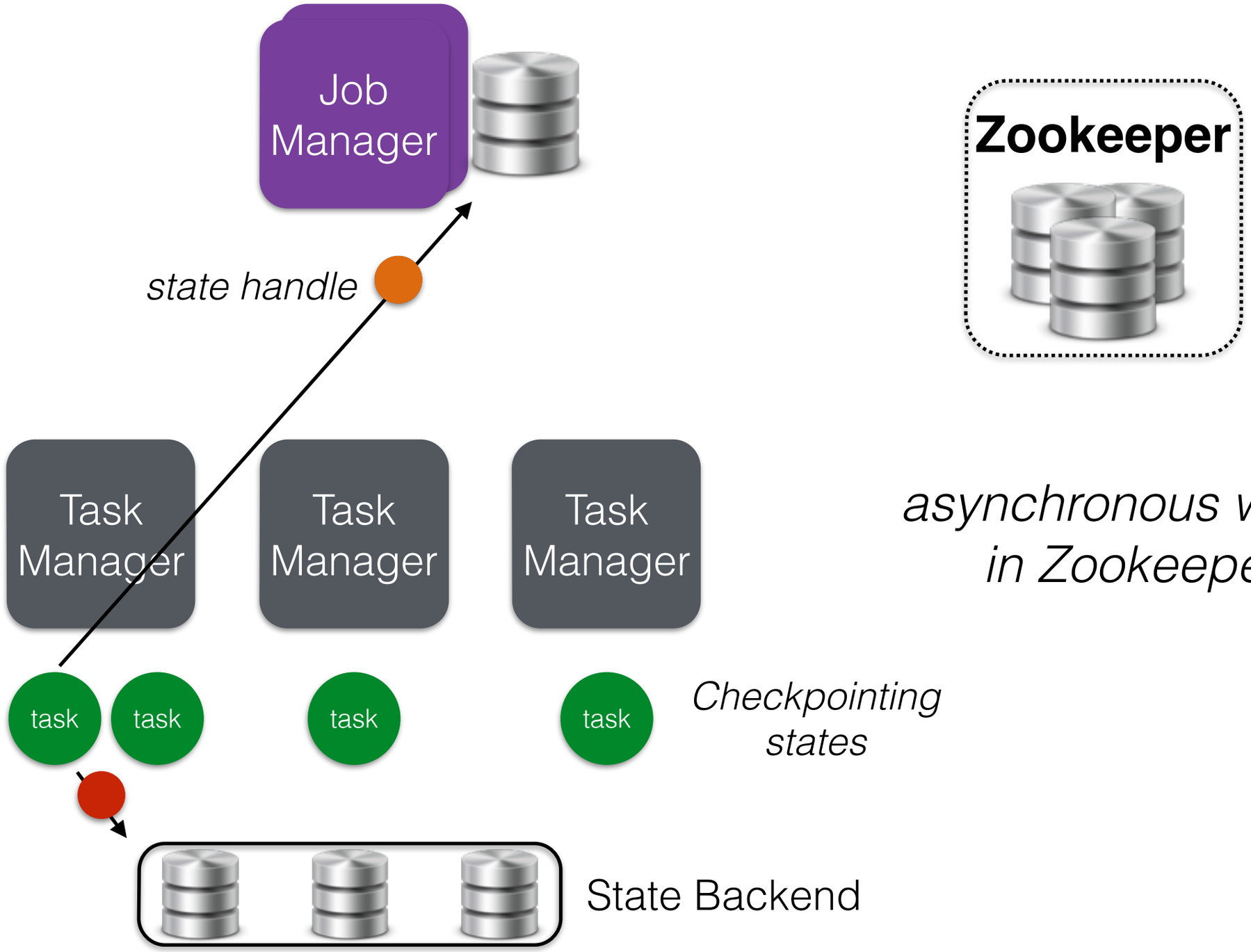
asynchronous writes in Zookeeper



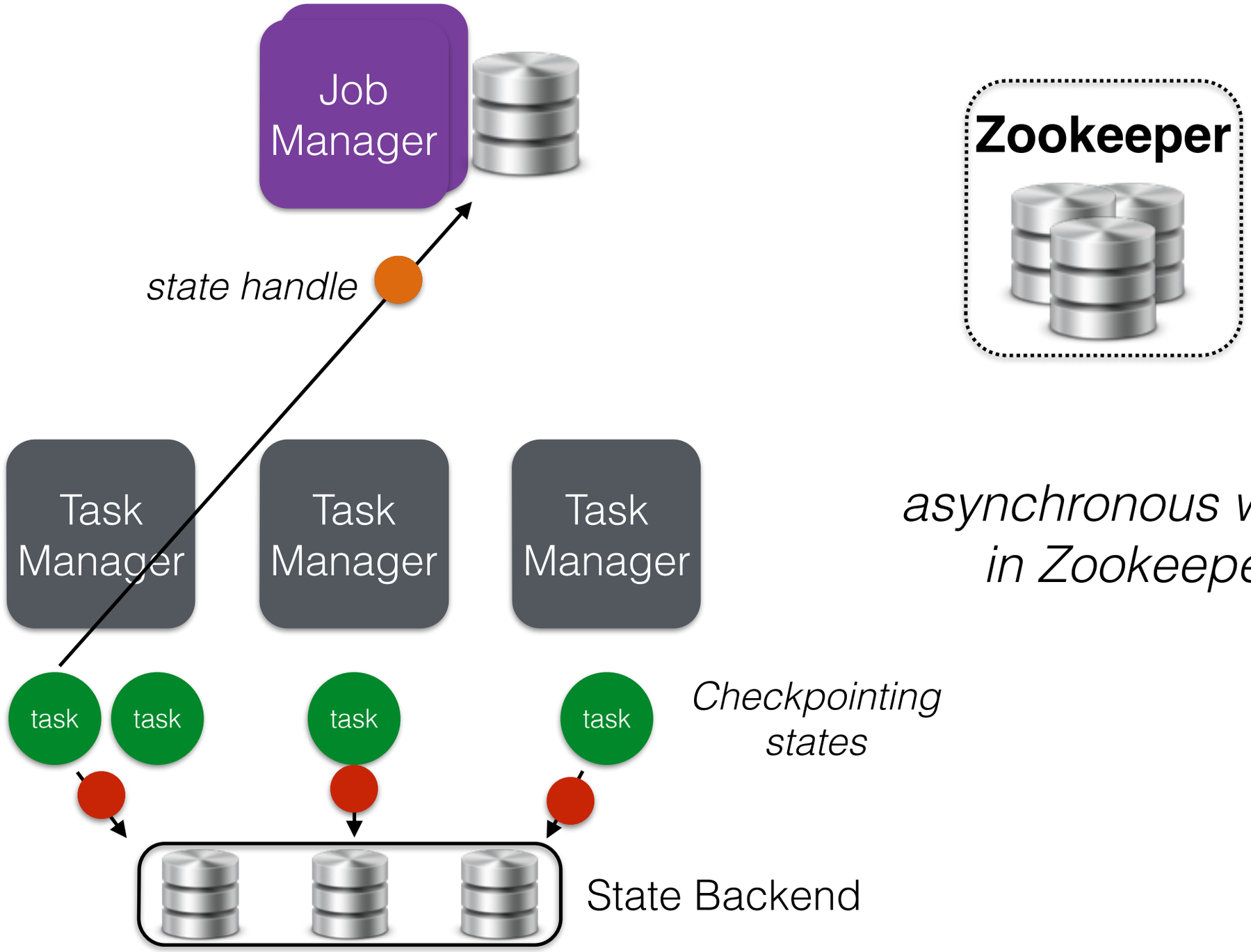
Checkpointing states

State Backend

Logging Snapshots



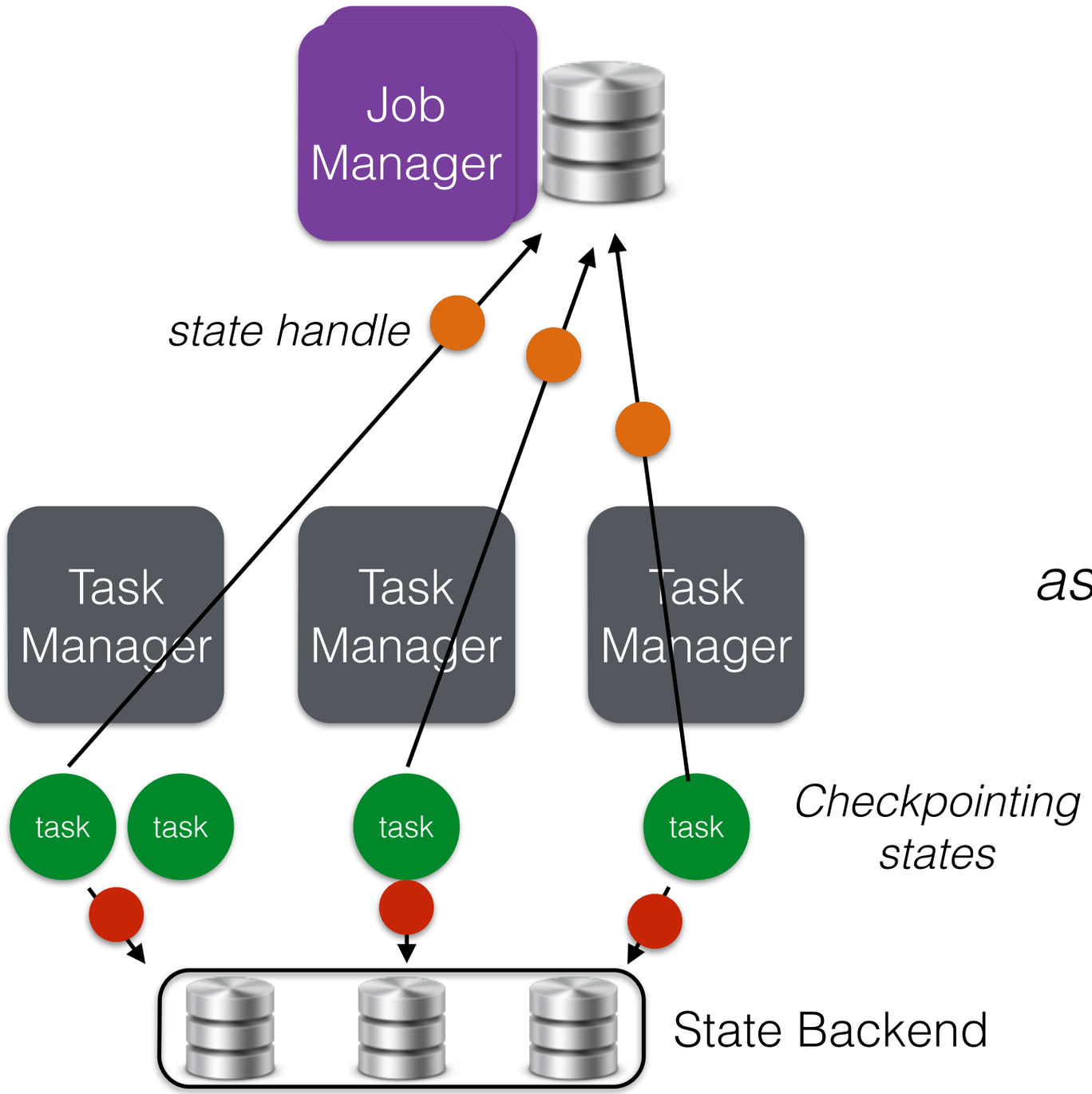
Logging Snapshots



asynchronous writes in Zookeeper

Checkpointing states

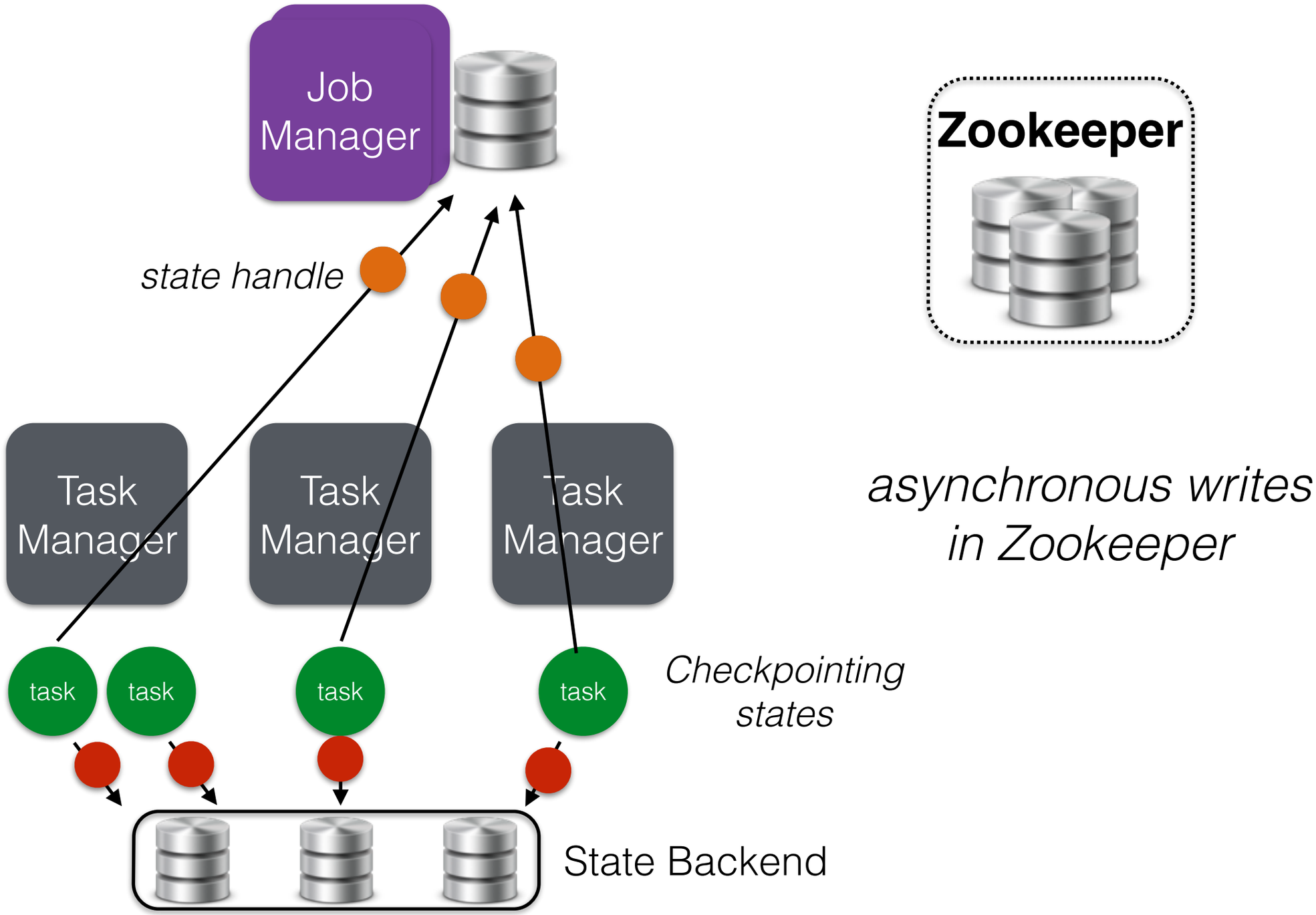
Logging Snapshots



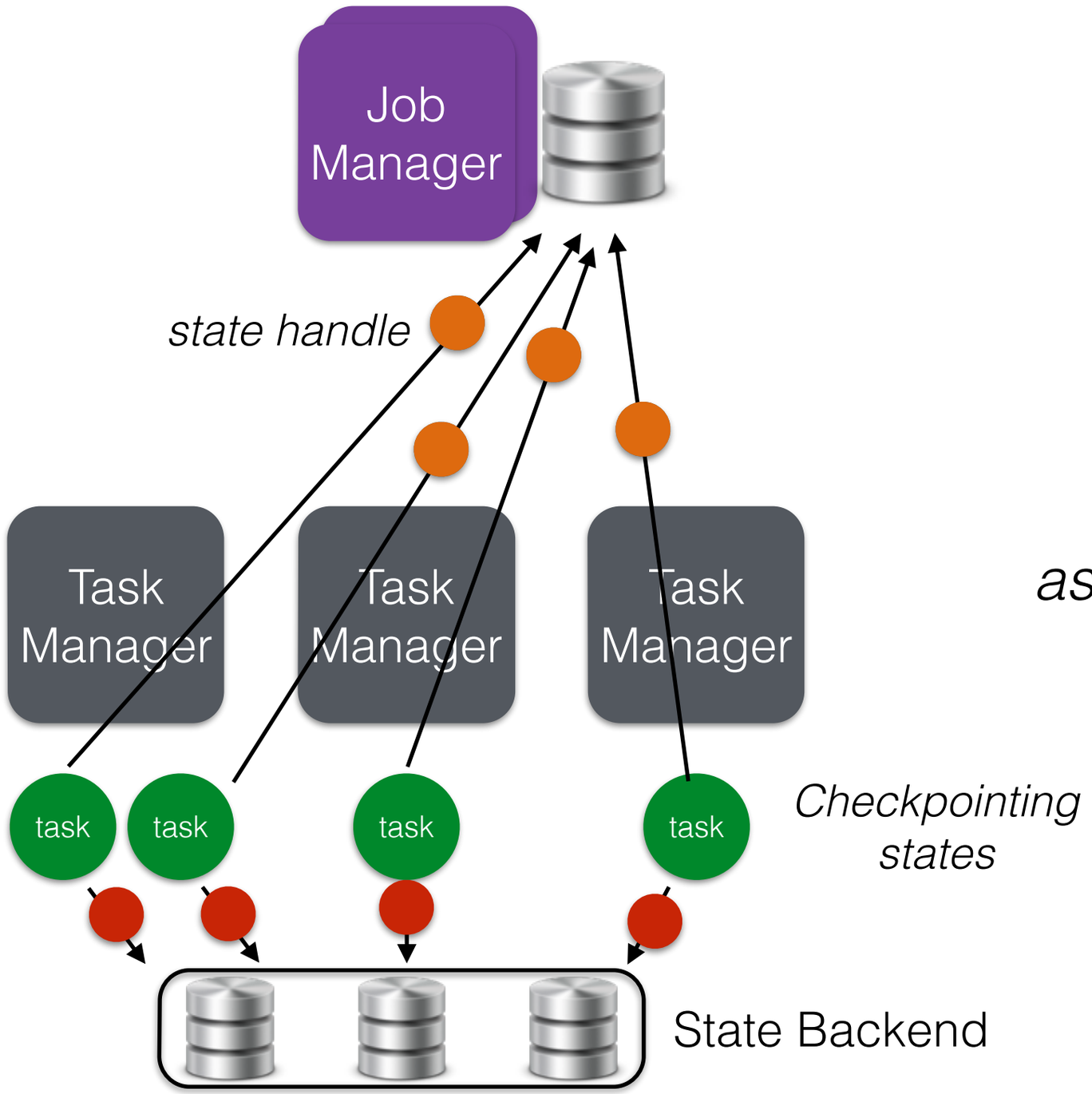
asynchronous writes in Zookeeper

Checkpointing states

Logging Snapshots



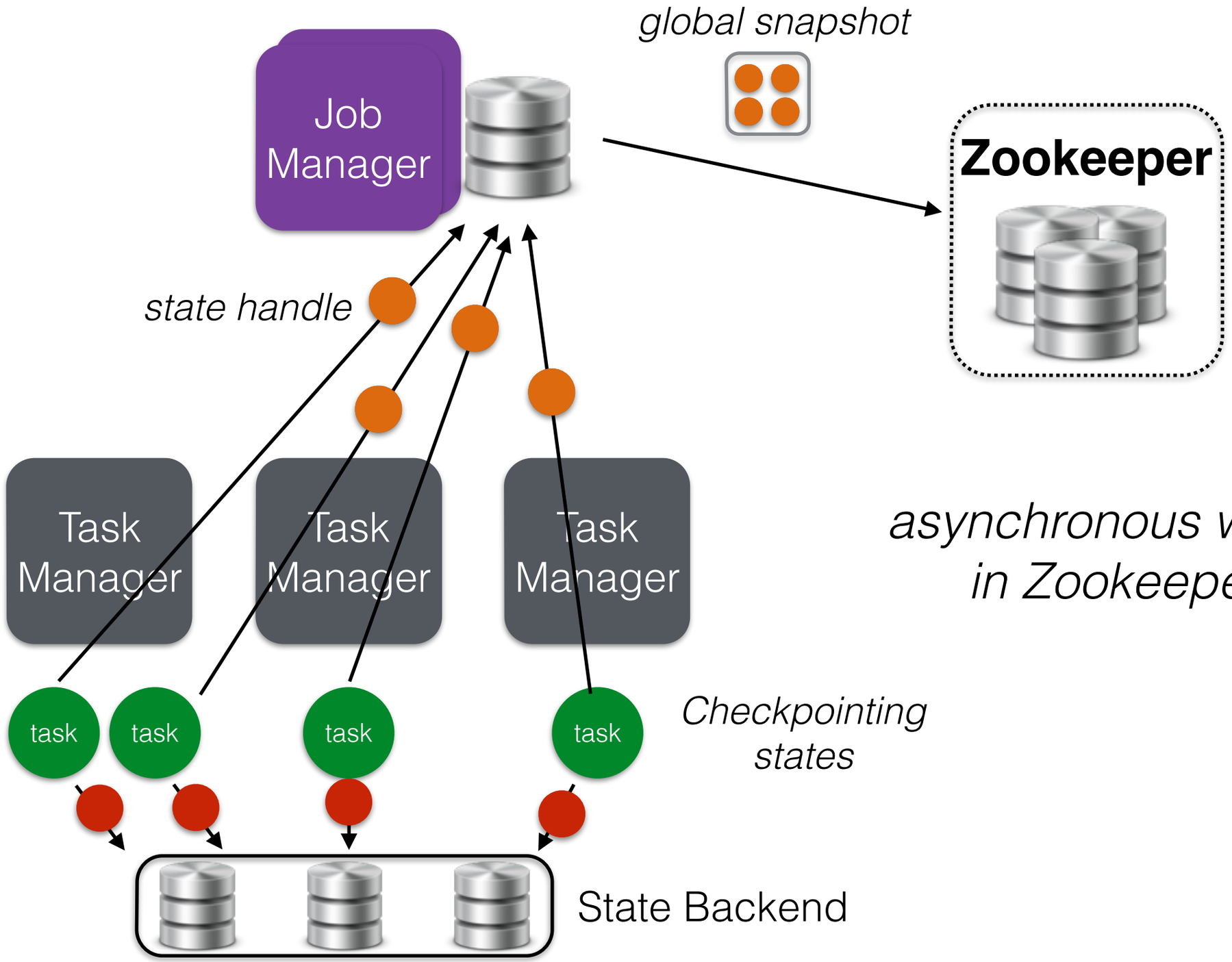
Logging Snapshots



asynchronous writes in Zookeeper

Checkpointing states

Logging Snapshots



asynchronous writes in Zookeeper

Checkpointing states

HA in Zookeeper

- Task Managers query ZK for the current leader before connecting (with lease)
- Upon standby failover **restart** pending execution graphs
- **Inject** state handles from the last global snapshot

Summary

- The Flink execution monitors long running tasks
- Establishing exactly-once-processing guarantees with snapshots can be achieved without halting the execution
- The ABS algorithm persists the minimal state at a low cost
- Master HA is achieved through Zookeeper and passive failover