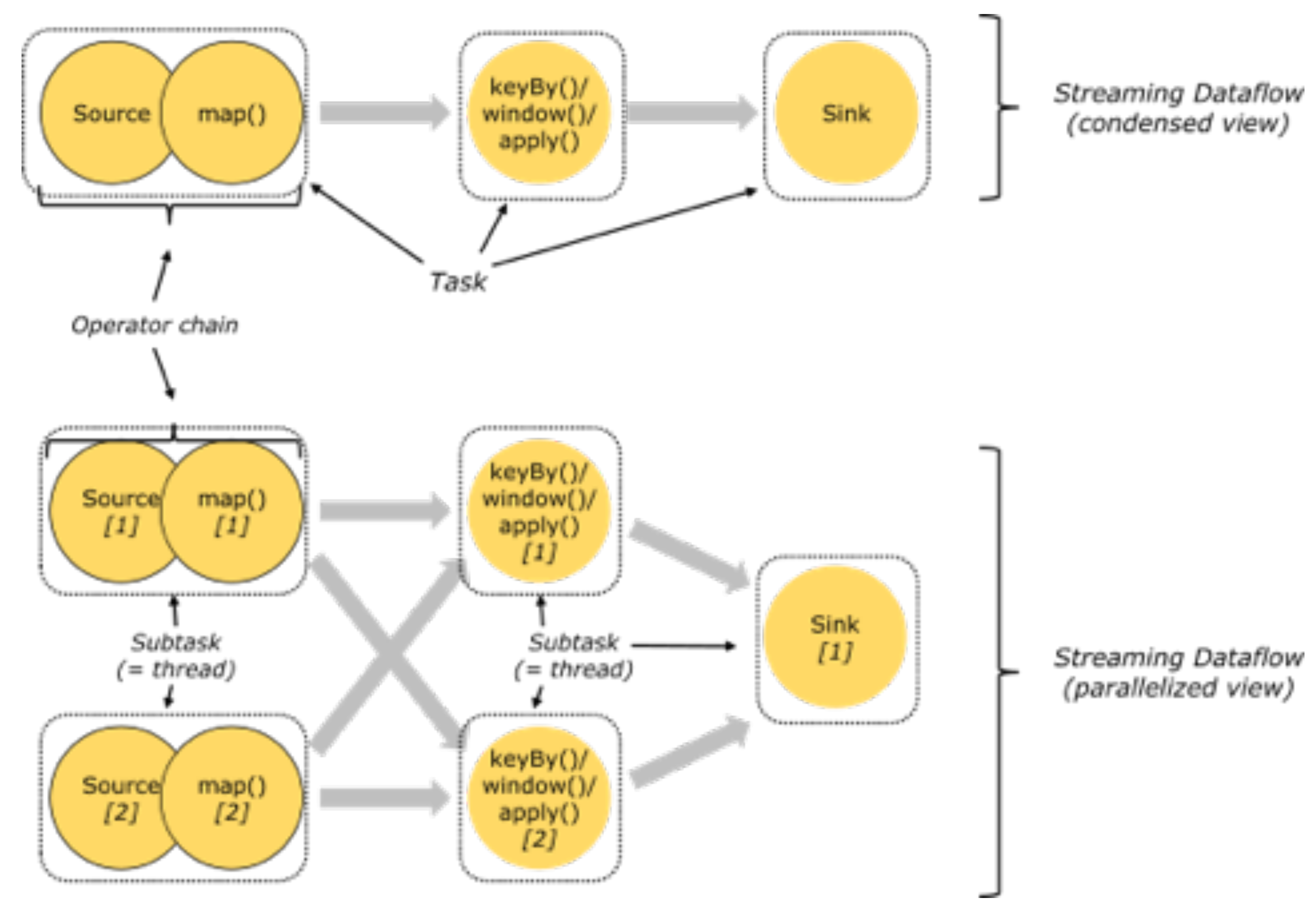
Flink’s Distributed Environment

Tasks and Operator Chains

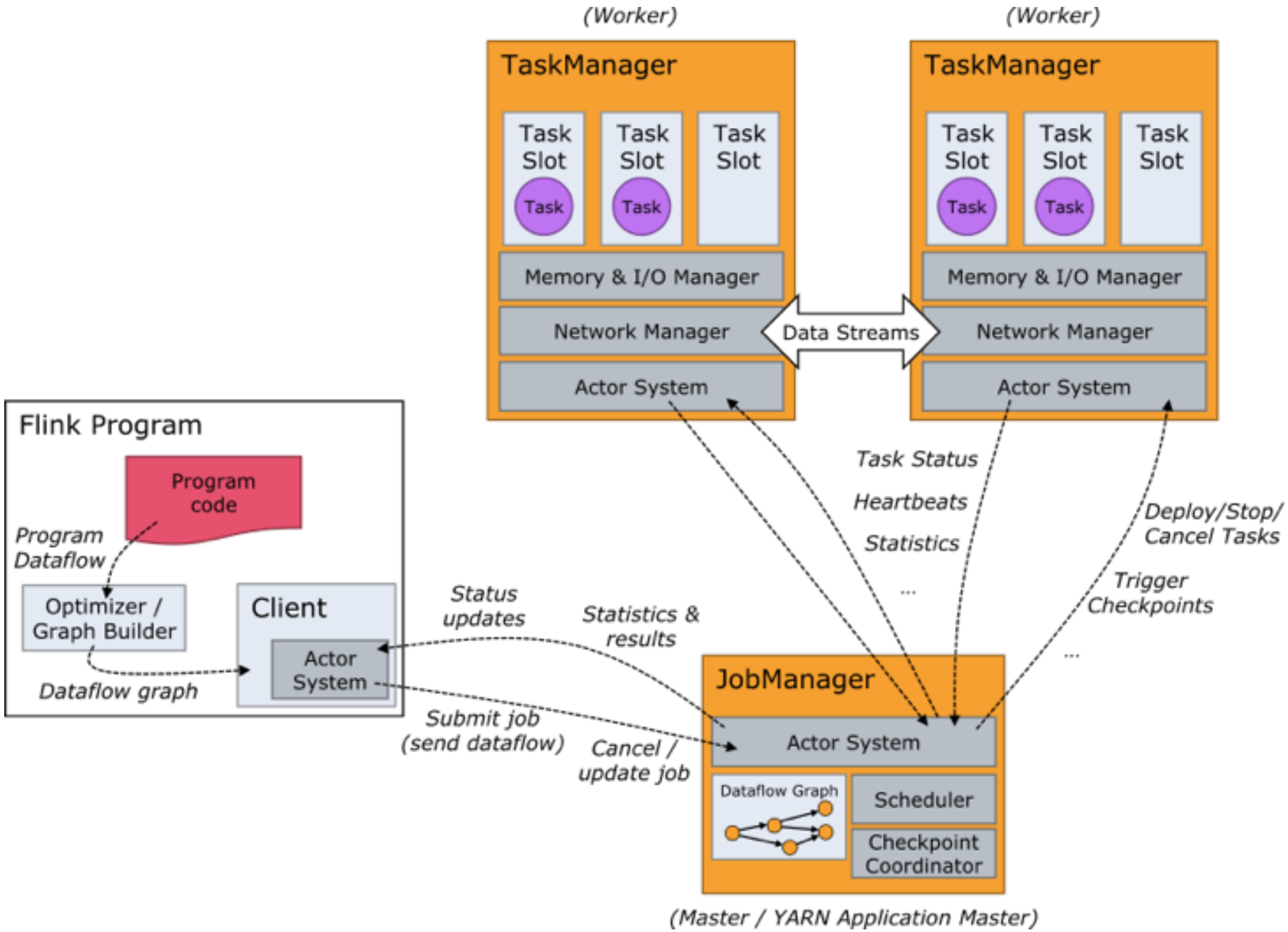
- If possible, Flink chains operators together into tasks;
- Each task is executed by one thread;
- Chaining reduces overhead of communications, increases throughput, and reduces latency.
- Chaining is only possible when a transformation *forwards* records downstream — i.e., two transformation has the same parallelism and no data partitioning strategy specified



JM, TM, Client

- The JobManager(*s — in HA mode*) coordinates the distributed execution:
  - Schedules tasks to TaskManagers;
  - Detects failure using heartbeats;
  - Coordinates checkpoints and recovery upon failure;
  - others…
- The TaskManagers execute tasks and exchange the data streams.

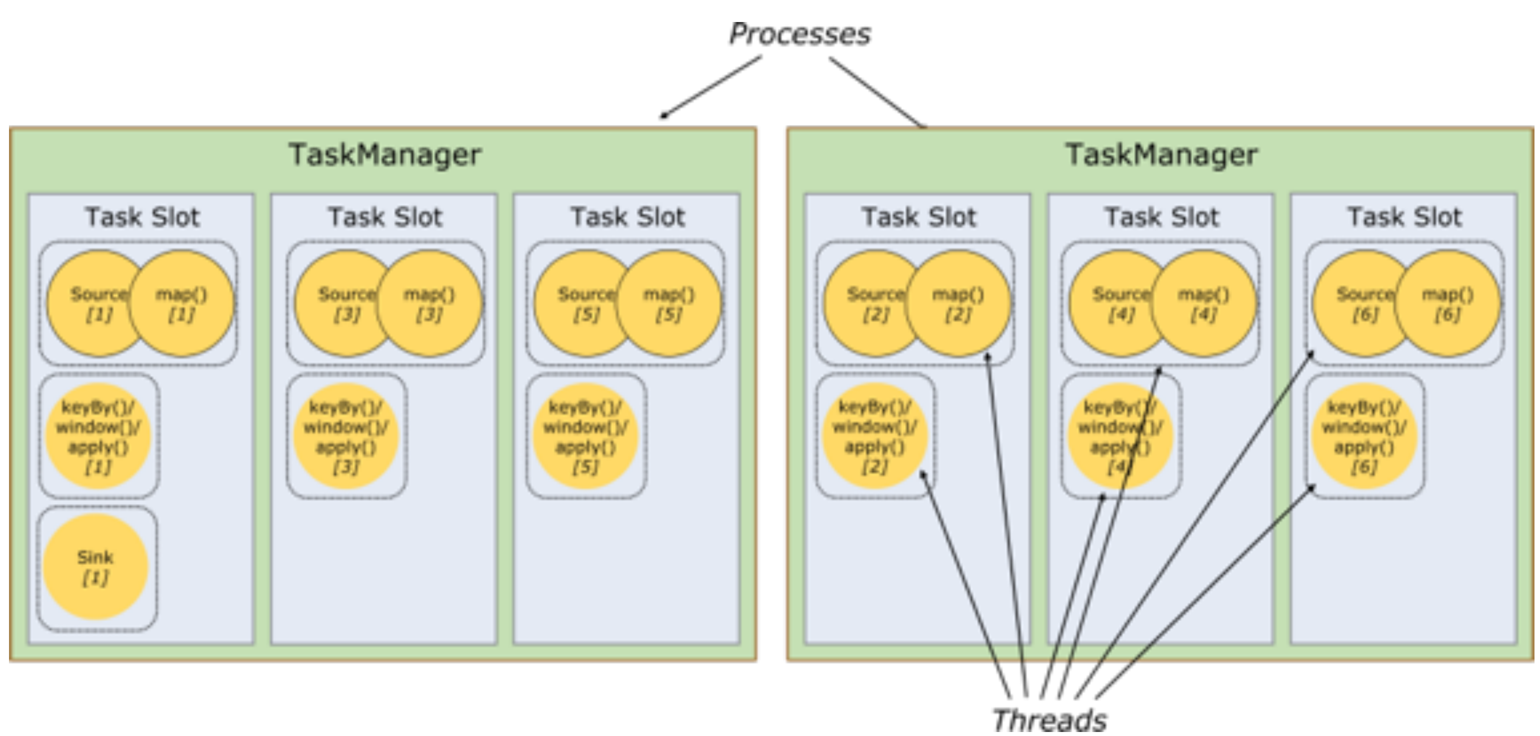
JM, TM, Client



(Worker)
TaskManager
Task Slot
Task
Task Slot
Task
Task Slot
Memory & I/O Manager
Network Manager
Actor System
(Worker)
TaskManager
Task Slot
Task
Task Slot
Task
Task Slot
Memory & I/O Manager
Network Manager
Actor System
Data Streams
Flink Program
Program code
Program Dataflow
Optimizer / Graph Builder
Client
Actor System
Dataflow graph
Status updates
Statistics & results
Submit job (send dataflow)
Cancel / update job
Task Status
Heartbeats
Statistics
...
Deploy/Stop/ Cancel Tasks
Trigger Checkpoints
...
JobManager
Actor System
Dataflow Graph
Scheduler
Checkpoint Coordinator
(Master / YARN Application Master)

Task Slots

- Each TM is a JVM process that executes subtasks with separate threads;
- The resources used by a TM are distributed over *task slots*. (3 task slots —> 1/3 of managed memory per slot);
- Slots are *job private*, however, only subtasks of different tasks can share the same task slot;
- The result is that one slot may hold an entire pipeline of the job (see below);
- Tasks in the same JVM share TCP connections (via multiplexing) and heartbeat messages (and others);
- A Flink cluster needs exactly as many task slots as the highest parallelism used in the job.



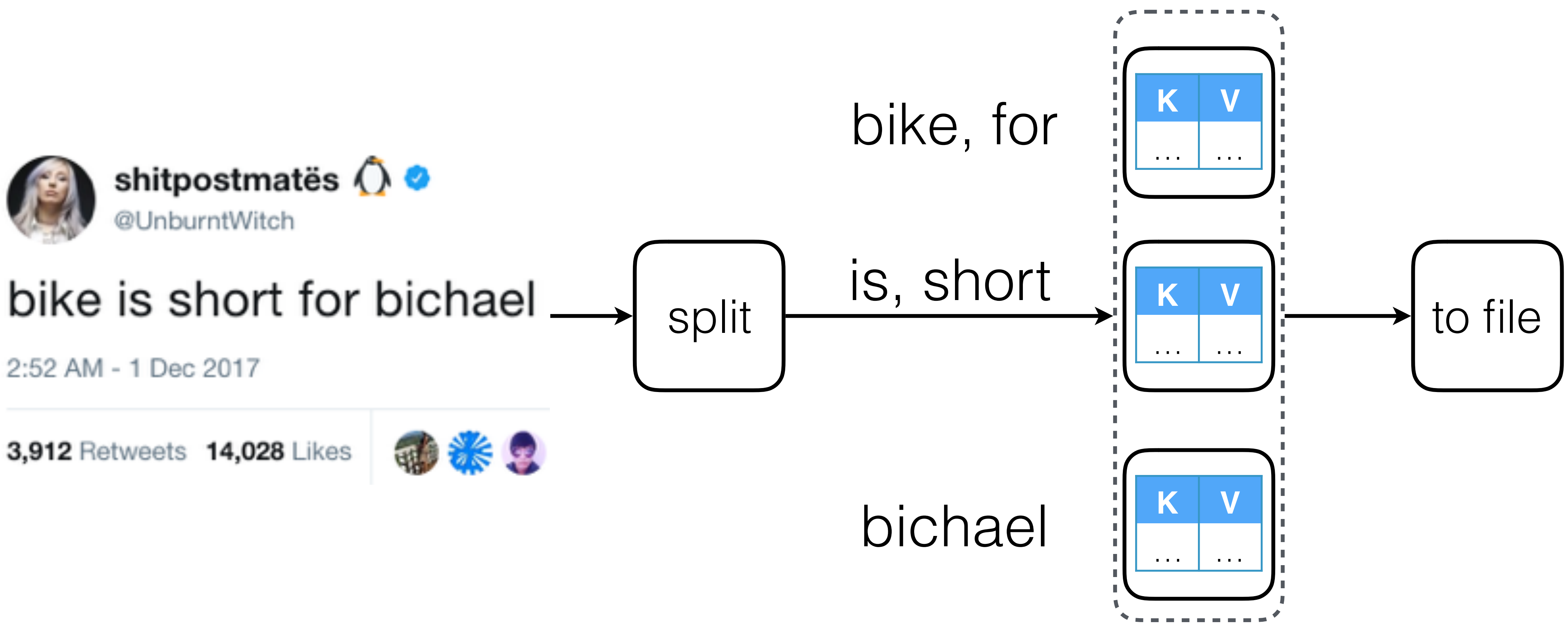
Fault Tolerance

The Problem

- Operators carry an internal state.
- What happens upon failure?
- Flink offers a fault tolerance mechanism to consistently recover the state of data streaming applications;
- The mechanism ensures that even in the presence of failures, the program's state will eventually process every record from the data stream **exactly once**

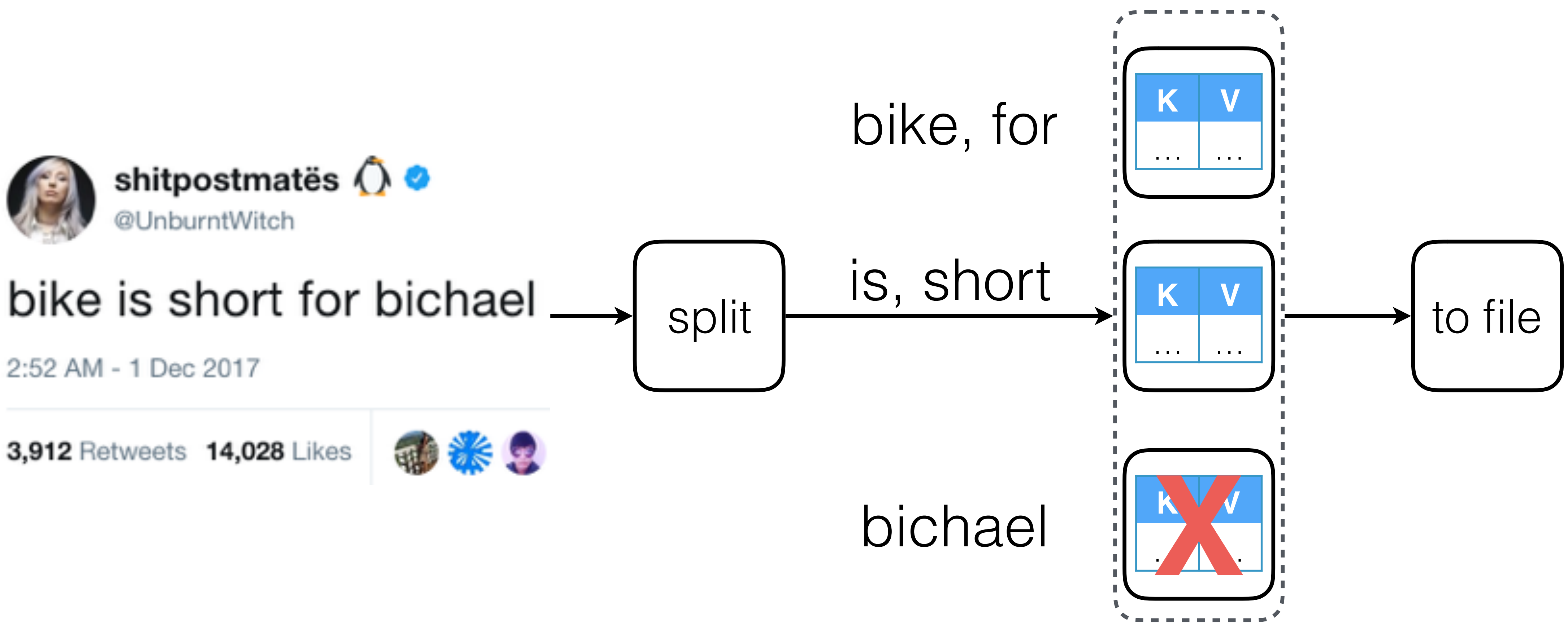
Failure & State problem

Suppose you count word occurrences from tweets



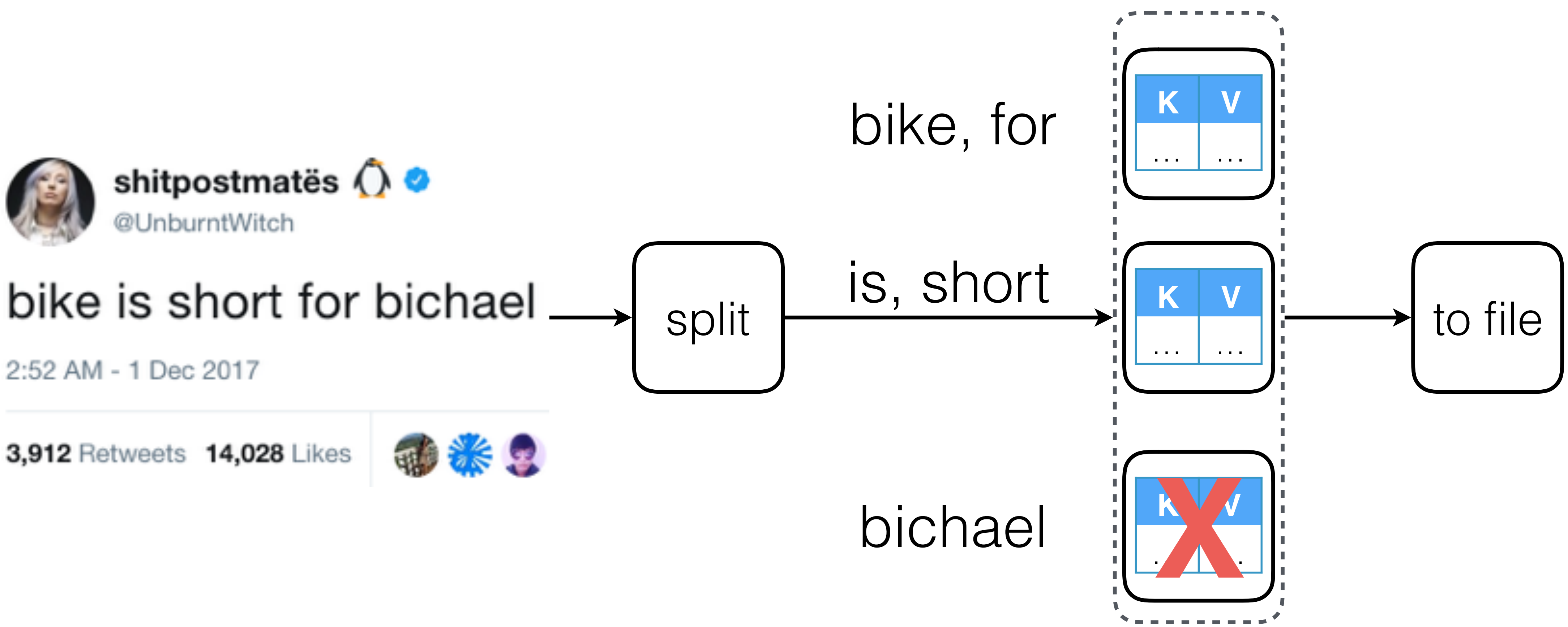
Failure & State problem

Suppose you count word occurrences
from tweets



Failure & State problem

- What now? If we replay the tweet, “bike” will be counted **twice**
- If we don’t replay it, “bichael” will be **lost**!



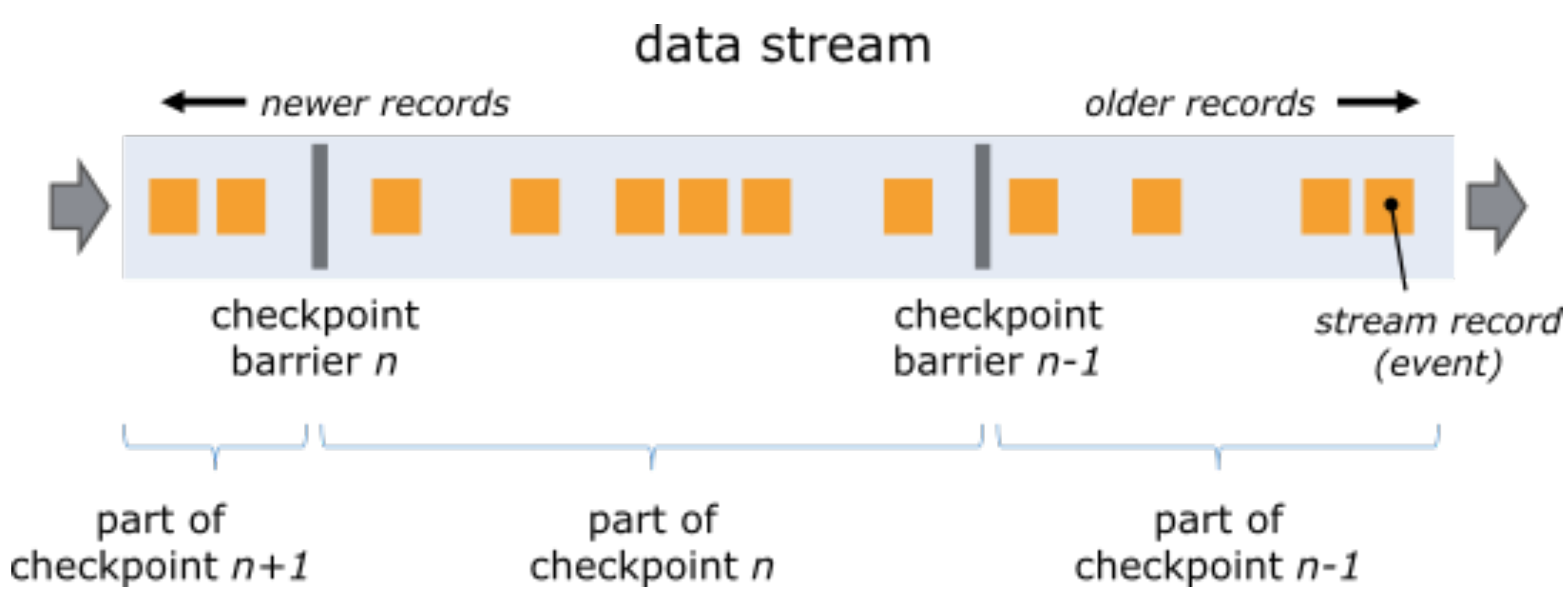
The Solution: Global Snapshot

> "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

Snapshotting[1]

- The idea is that input records are divided into *epochs*;
- For every epoch, the system saves the internal state so that it has been affected by every record of previous epochs **and nothing else**;
- The division in epochs is represented by special markers that flow with records called *barriers*.



[1] Lightweight Asynchronous Snapshots for Distributed Dataflows - Carbone et al.

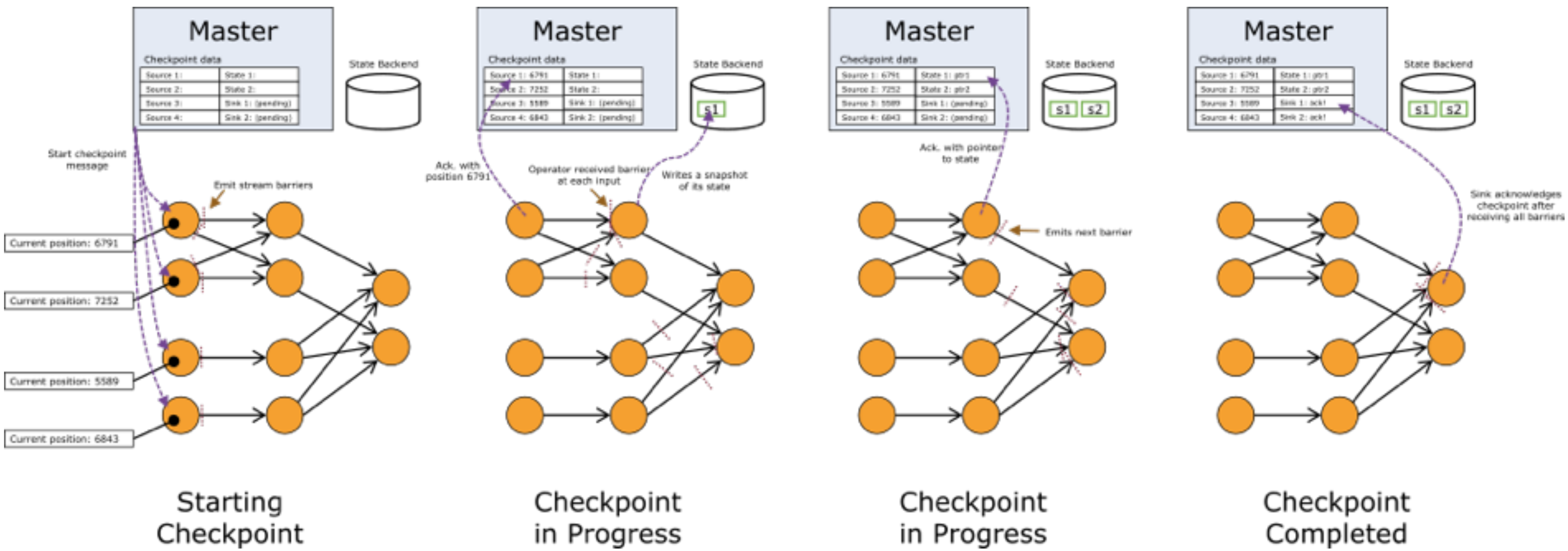
Snapshotting in a Nutshell

- The state saved at every epoch is called *snapshot*;
- In case of a program **failure**, the system:
  - **stops** the distributed streaming job;
  - **restarts** (—> different strategies) it;
  - **restores** the latest successful snapshot;
  - **replays** the input streams from the offset at which the snapshot was saved (—> *reliable sources*);
  - Any record that is processed as part of the restarted parallel dataflow is guaranteed to not have been part of the previous snapshot.

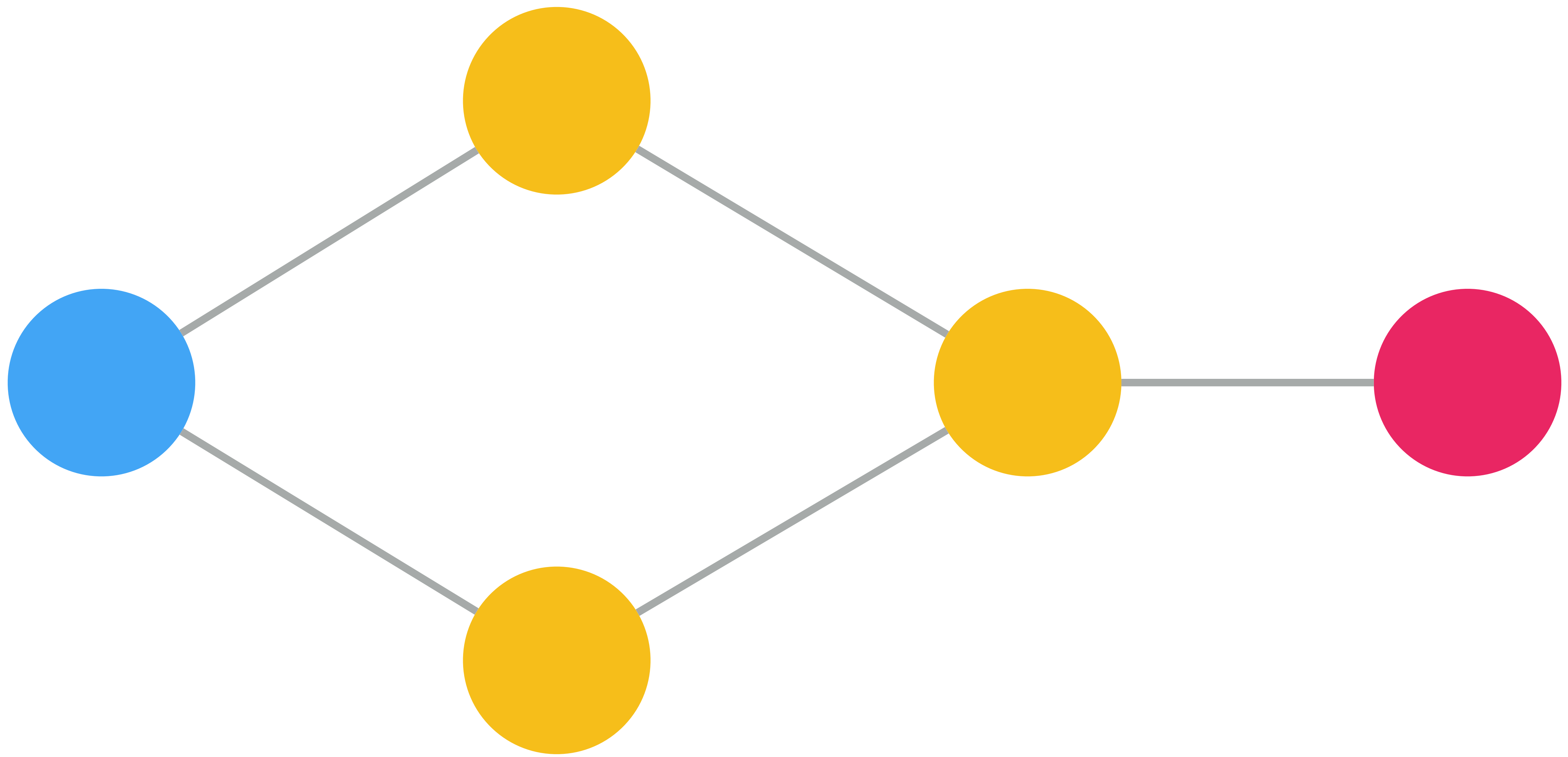
Snapshotting in Detail

- Upon snapshot, the **source operator** injects the barrier and reports the offset of the source stream to the the checkpoint coordinator (the JM).
- The barriers flow with the records as part of the data stream (we'll see later how…), they do not interrupt the flow.
- Upon checkpoint, a **stateful operator** stores its internal state; ACKs the checkpoint to the JM (report the pointer to the state); emits the barrier; and proceeds.
- Once a **sink operator** has received the barrier it ACKs that the snapshot is completed to the JM.
- After all sinks have acknowledged a snapshot, it is considered completed.

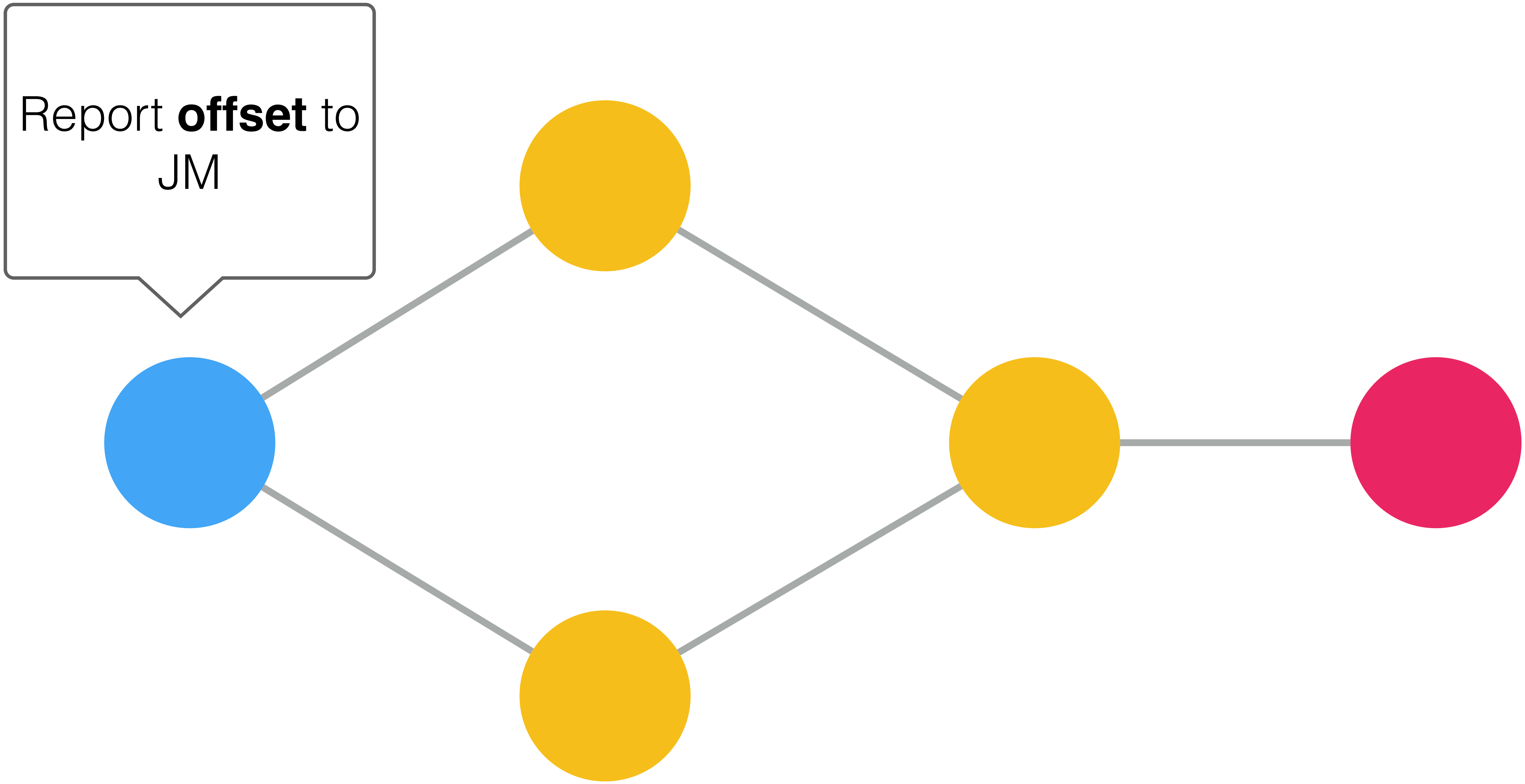
Snapshotting Visualization



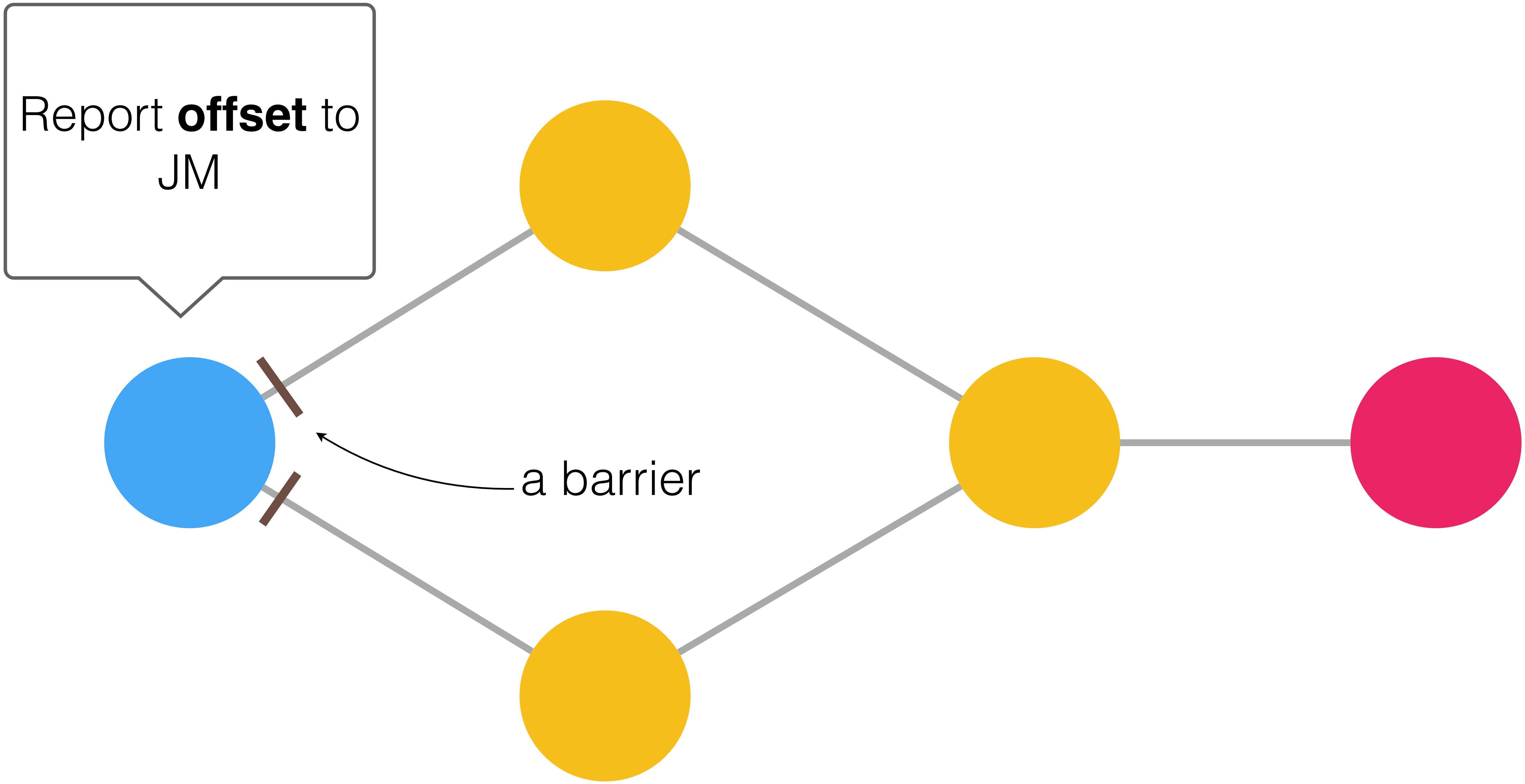
Snapshotting - Start



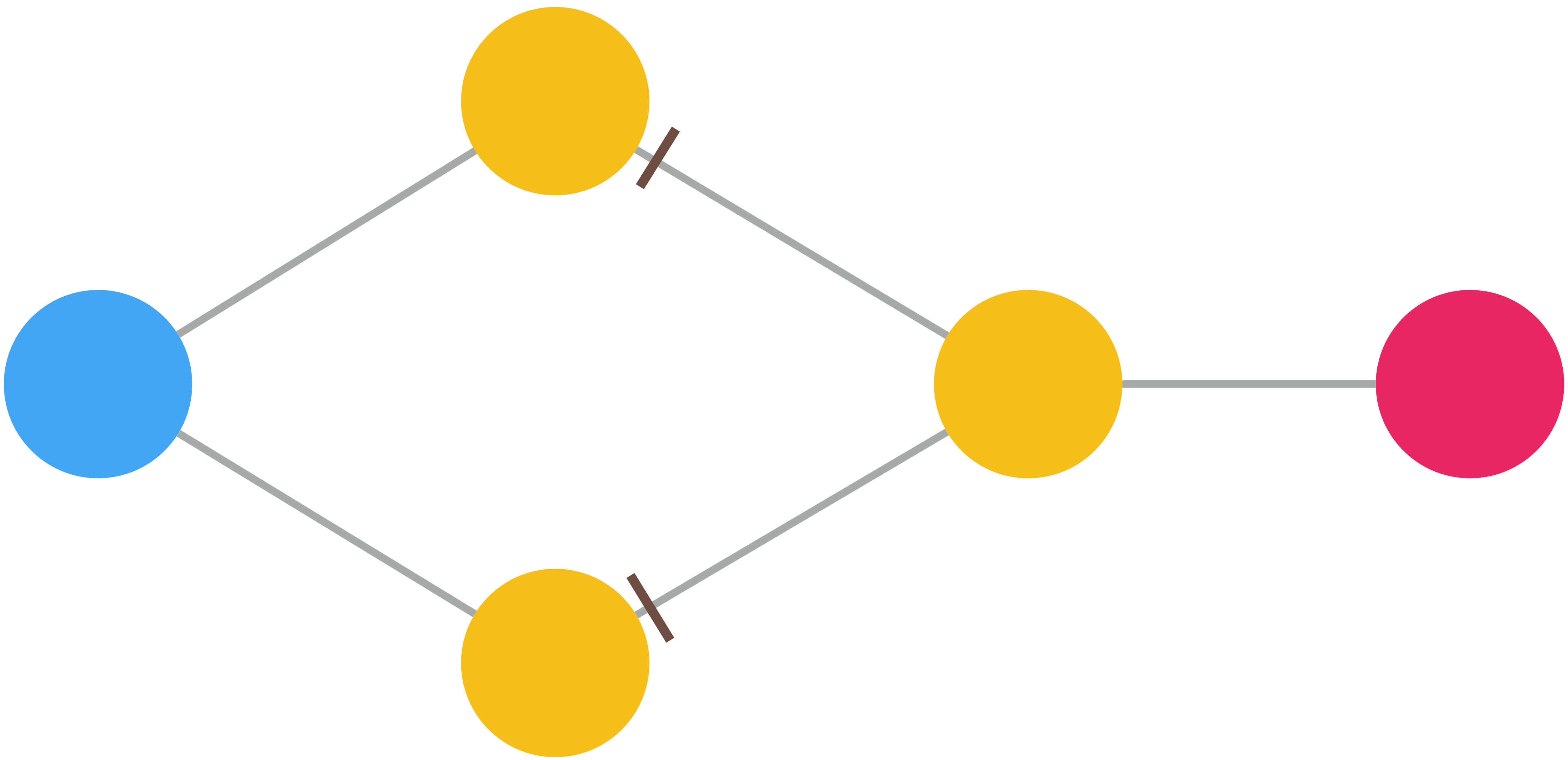
Snapshotting - Start



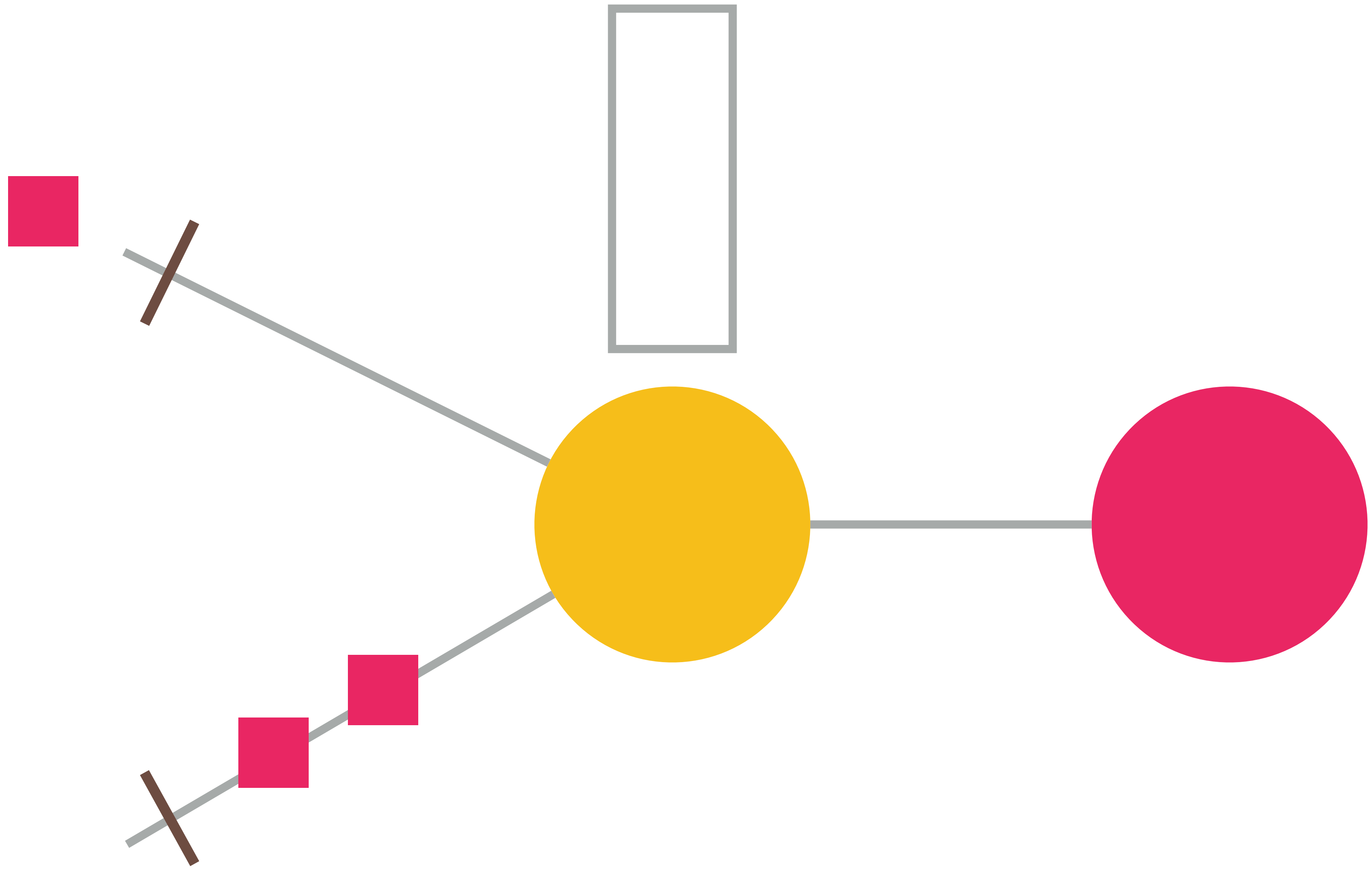
Snapshotting - Start



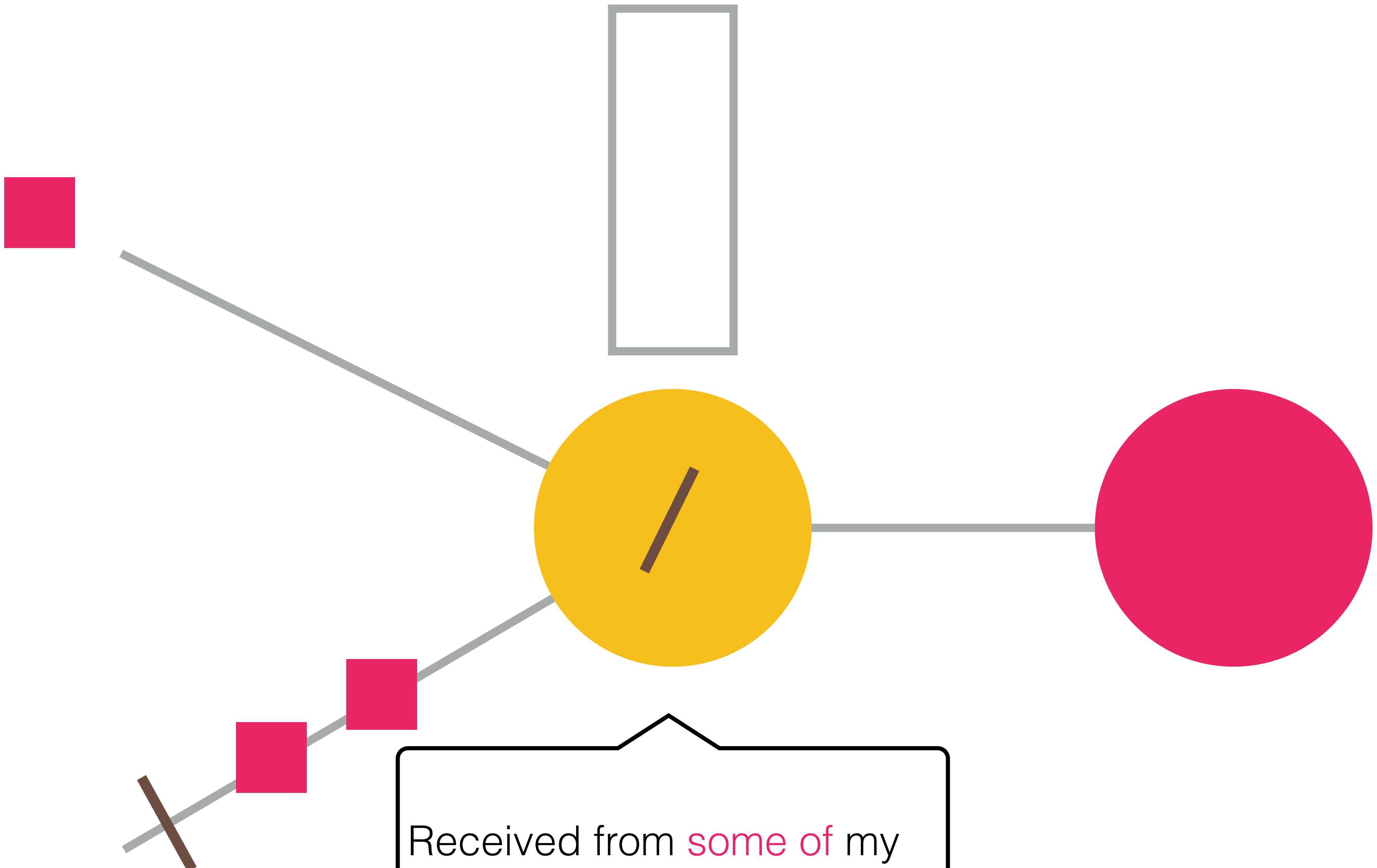
Snapshotting - Aligning



Snapshotting - Aligning

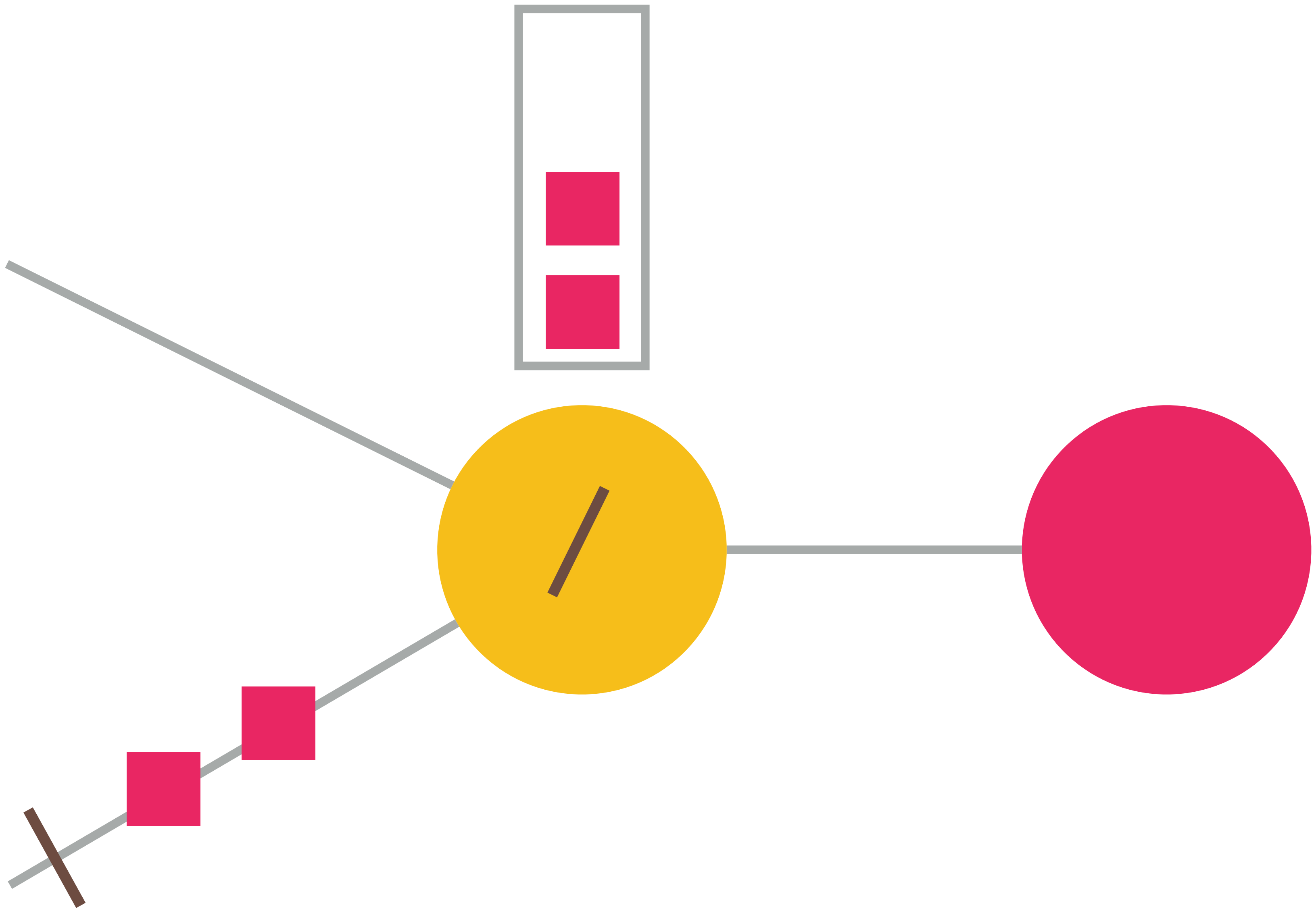


Snapshotting - Aligning

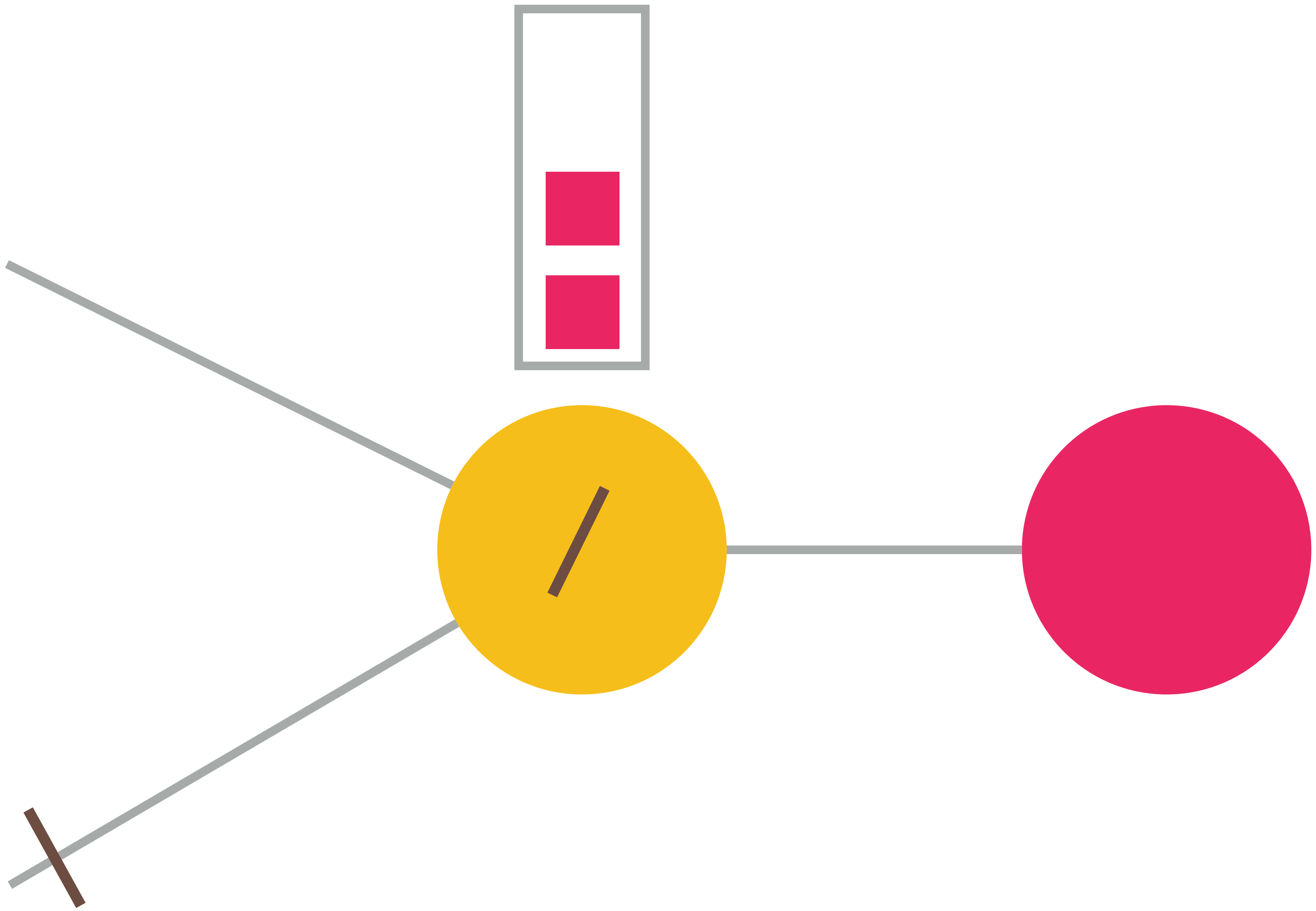


Received from some of my input streams.
Buffer messages and wait for the other barriers.

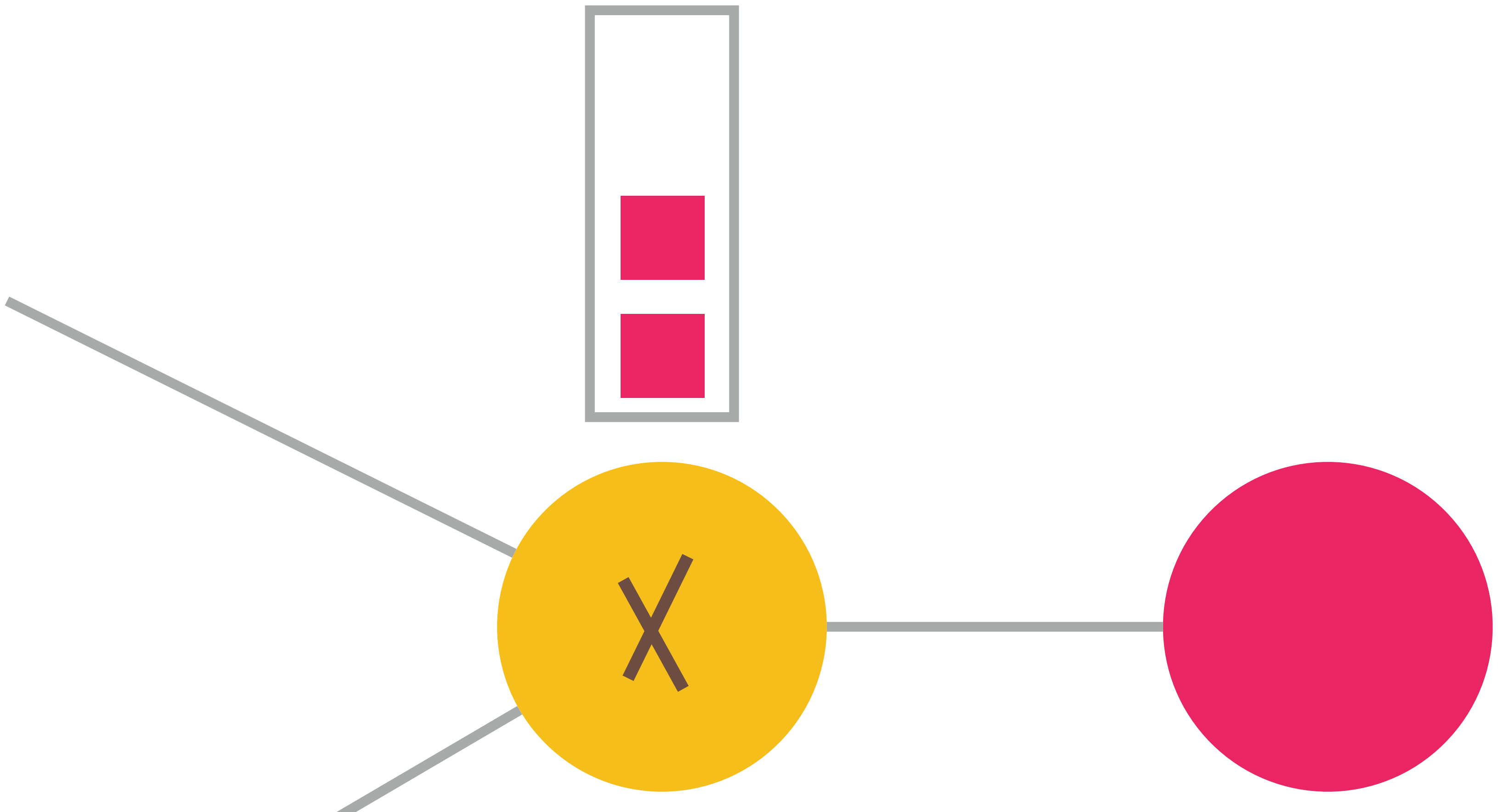
Snapshotting - Aligning



Snapshotting - Aligning

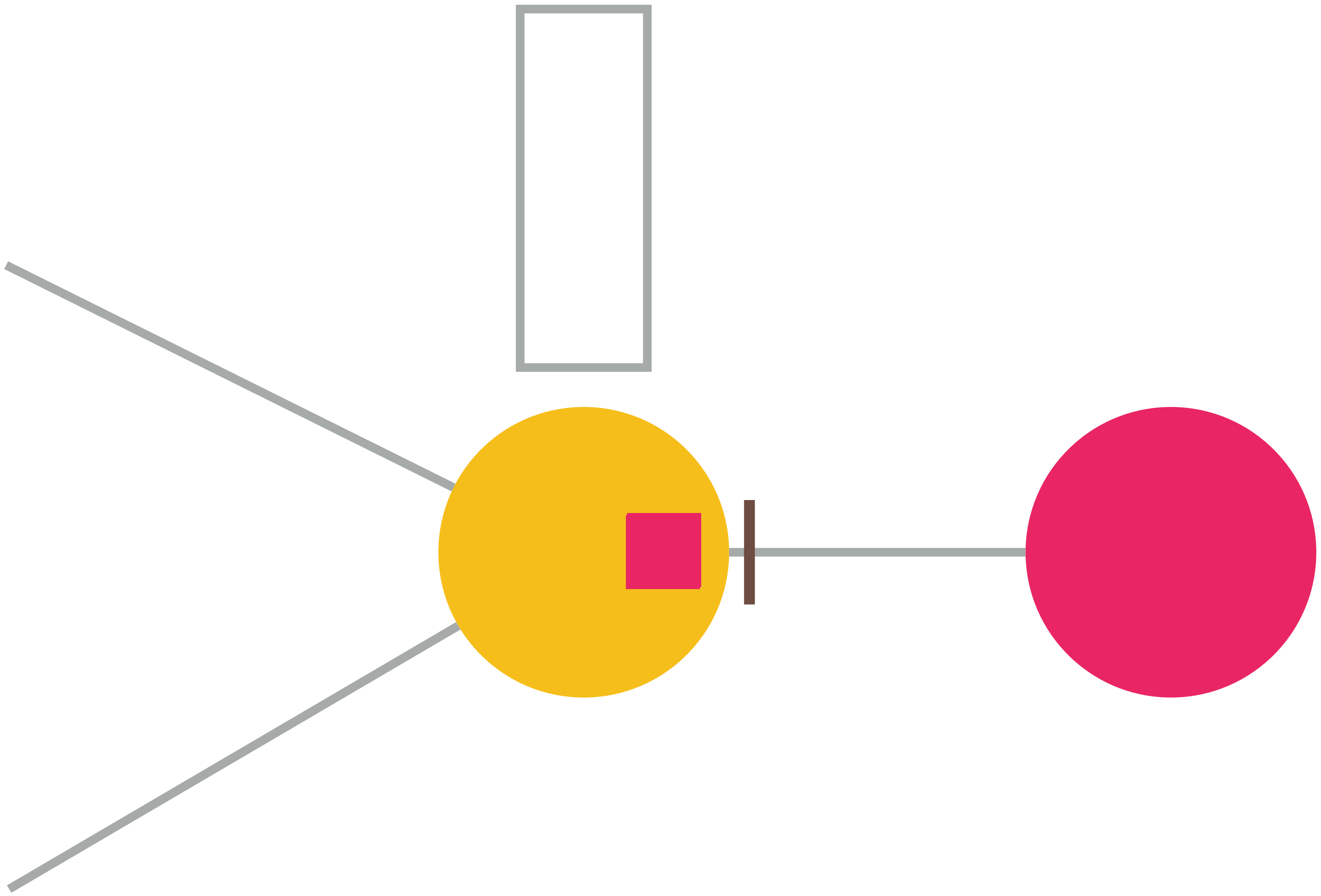


Snapshotting - Aligning

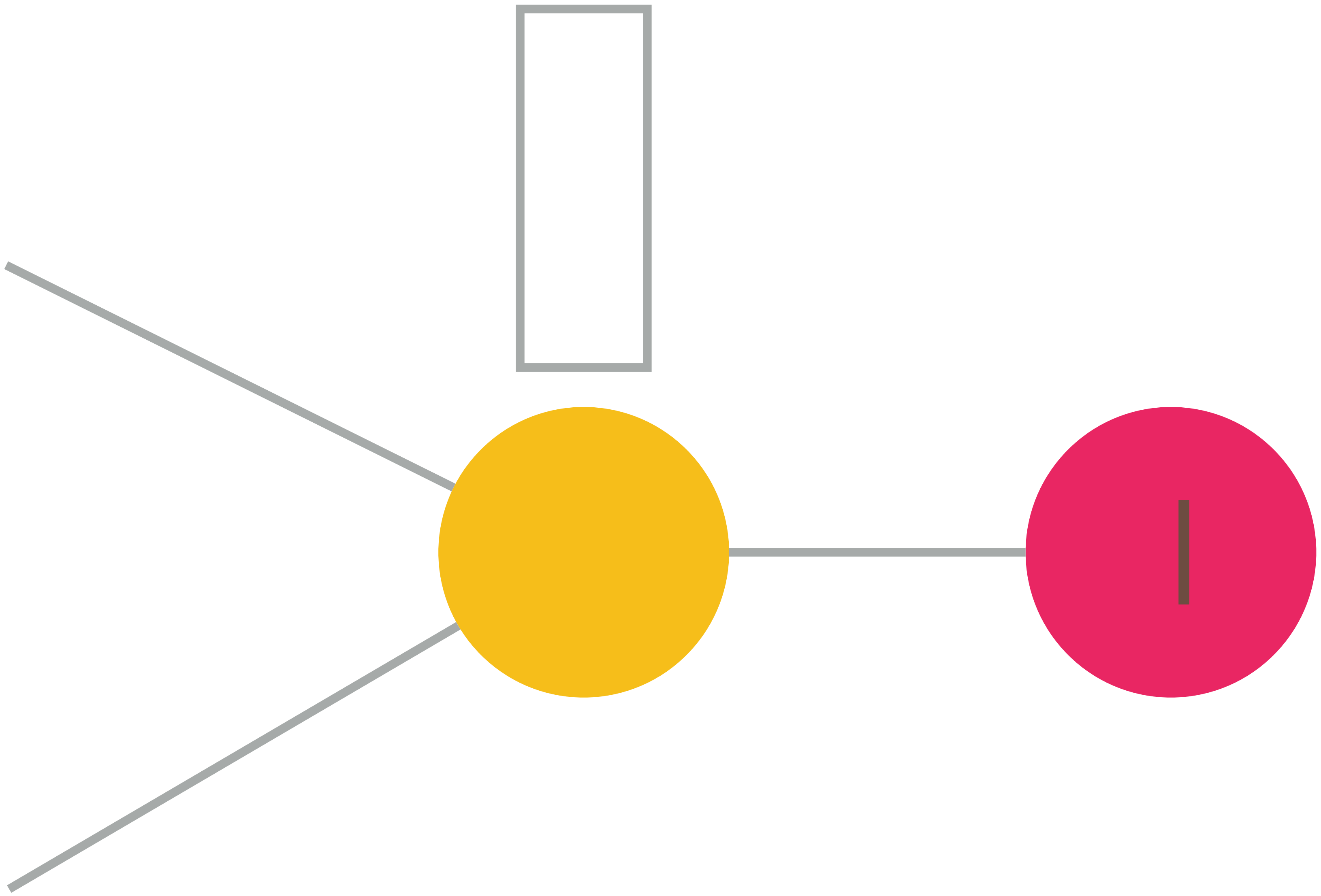


Received from every input stream.
Store state (if any);
and emit barrier.

Snapshotting - Aligning



Snapshotting - Aligning



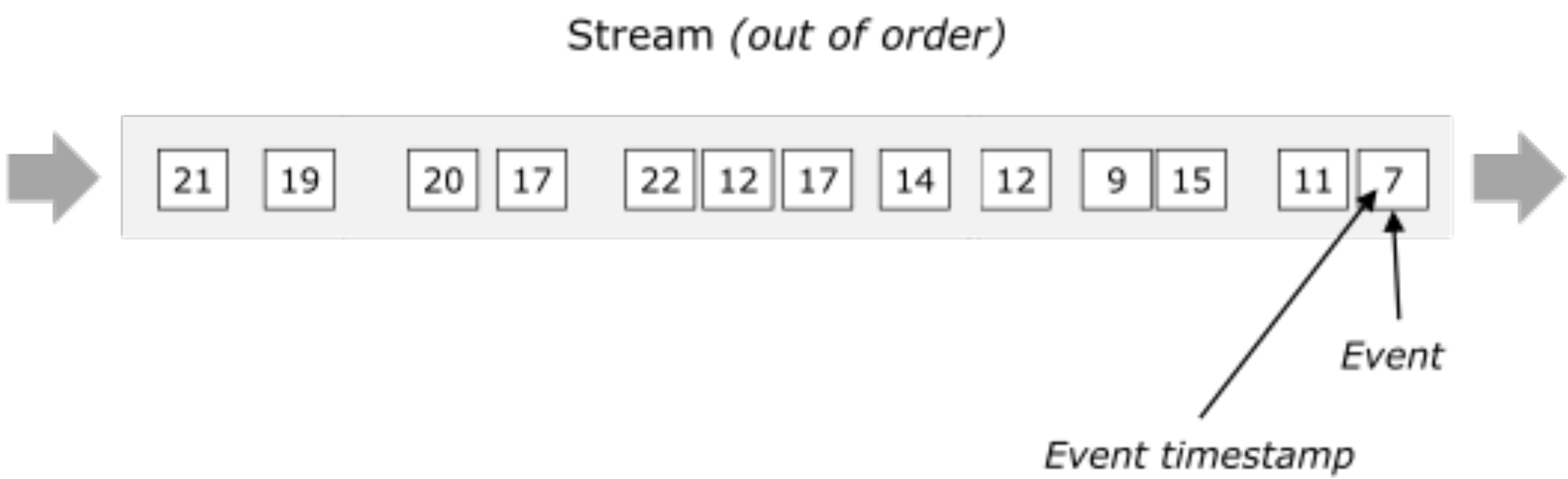
Exercise

- Implement a **fault-tolerant** word counter using Flink’s Keyed State;
- Simulate a **fault**;
- Check that everything goes at expected.

Event Time

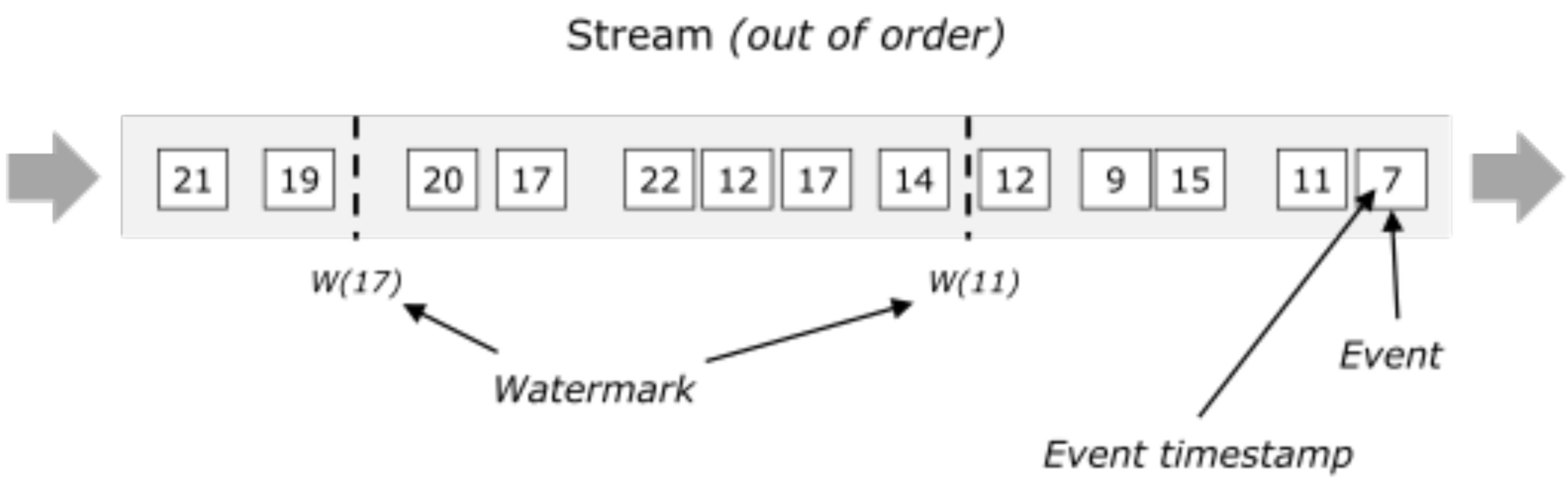
The Problem

How to keep the progress of event time?



The Problem

Watermarks

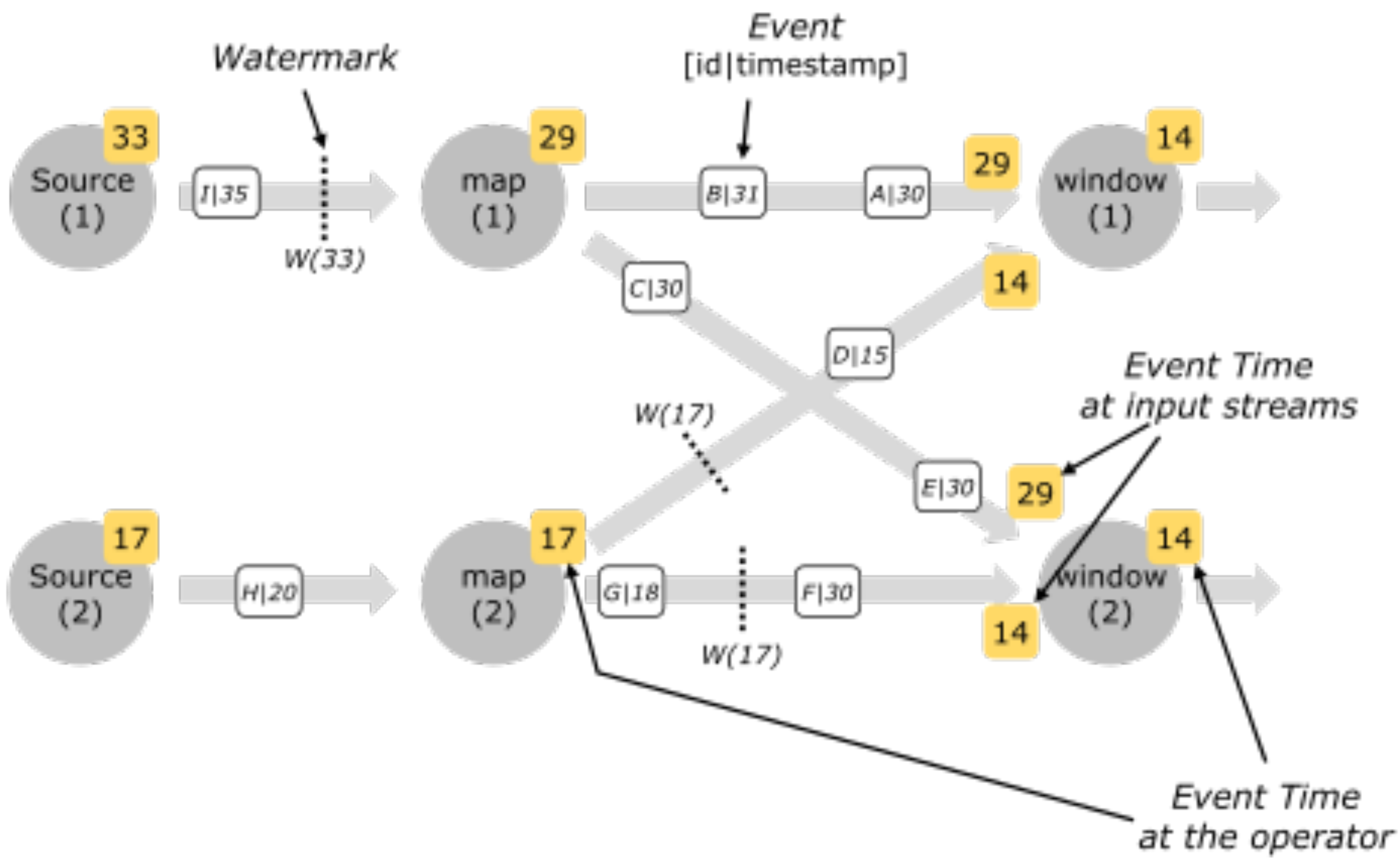


Watermarks

- Watermarks can be:
    - directly **injected** in the streams by the sources;
    - **extracted** from a timestamp field with various techniques[1];
- WM = *t* means that event time has reached *t*, thus that no record with a timestamp lower or equal to *t* will ever come;
- They become the clock of the system.

[1] more info in Flink's documentation

In a Distributed System?



References And Credits

- Data Artisans Blog
- Flink Documentation
- Lightweight Asynchronous Snapshots for Distributed Dataflows - Paris Carbone et al.
- Most images are Courtesy of the Flink Documentation