Distributed Systems Reading Group - Session 2 - Klas Segeljakt
Snapshotting Protocols in Distributed Stream Processing Systems

Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark

Michael Armbrust[†], Tathagata Das[†], Joseph Torres[†], Burak Yavuz[†], Shixiong Zhu[†], Reynold Xin[†], Ali Ghodsi[†], Ion Stoica[†], Matei Zaharia[†‡]
[†]Databricks Inc., [‡]Stanford University

Consistent Regions: Guaranteed Tuple Processing in IBM Streams

Gabriela Jacques-Silva[♠], Fang Zheng[♠], Daniel Debrunner[‡], Kun-Lung Wu[♠], Victor Dogaru[‡], Eric Johnson[‡], Michael Spicer[‡], Ahmet Erdem Sarıyüce[◇†]
[♠]IBM T. J. Watson Research Center, [‡]IBM Analytics Platform, [◇]Sandia National Labs

State Management in Apache Flink®

Consistent Stateful Distributed Stream Processing

Paris Carbone[†] Stephan Ewen[‡] Gyula Fóra[⋆]
Seif Haridi[†] Stefan Richter[‡] Kostas Tzoumas[‡]

[†]KTH Royal Institute of Technology
{parisc,haridi}@kth.se

[⋆]King Digital Entertainment Limited
gyula.fora@king.com

[‡]data Artisans
{stephan,s.richter,kostas}
@data-artisans.com

Outline

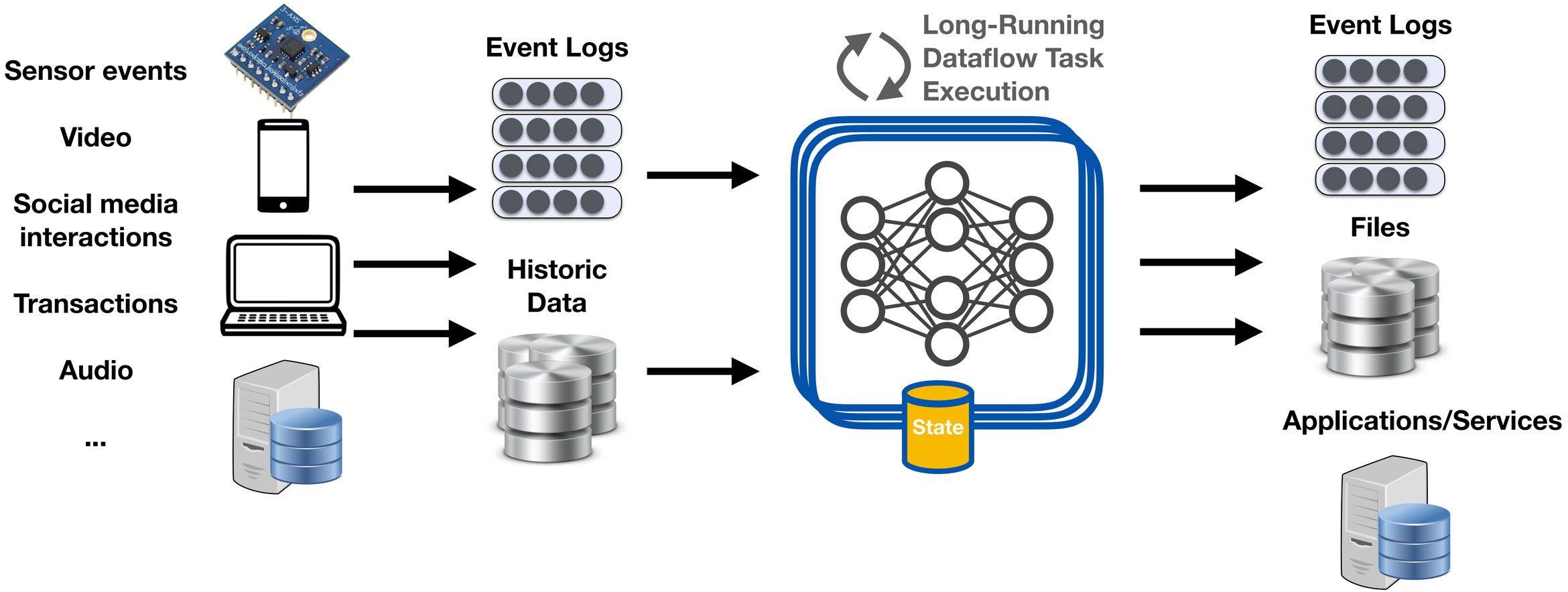
- Introduction
- Flink ➡ IBM Streams ➡ Spark Structured Streaming
  - Problem statement
  - System overview
  - Snapshotting protocol
  - Discussion (based on results)
- Comparison & Conclusion

Introduction

Introduction - Data Stream Processing

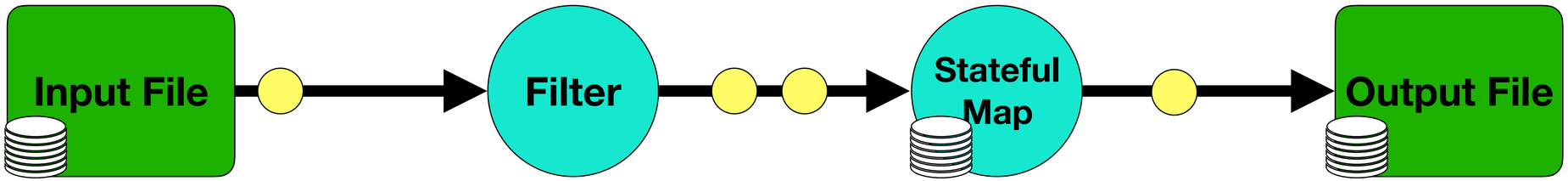
Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Introduction - Data Stream Processing

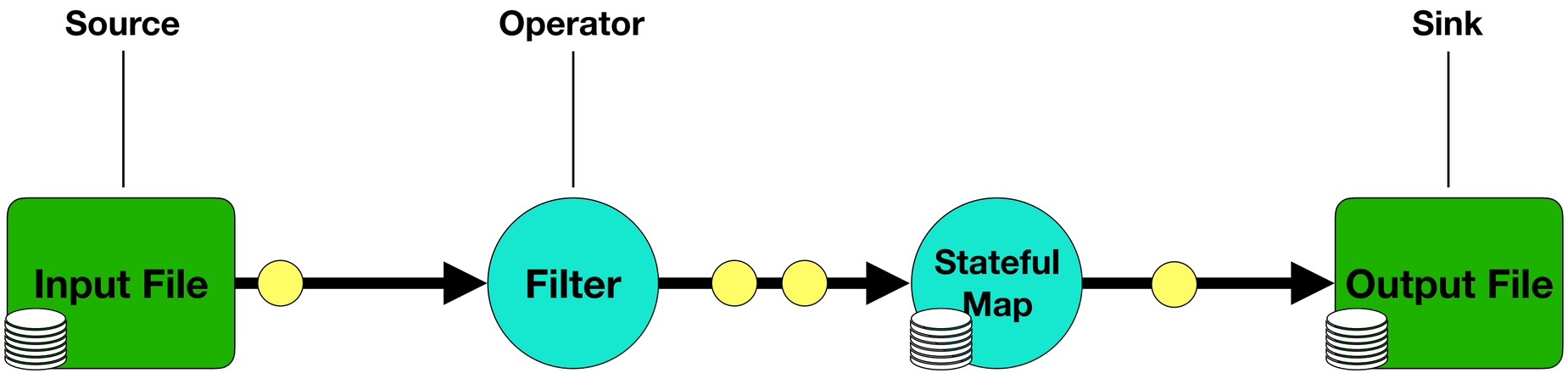


Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

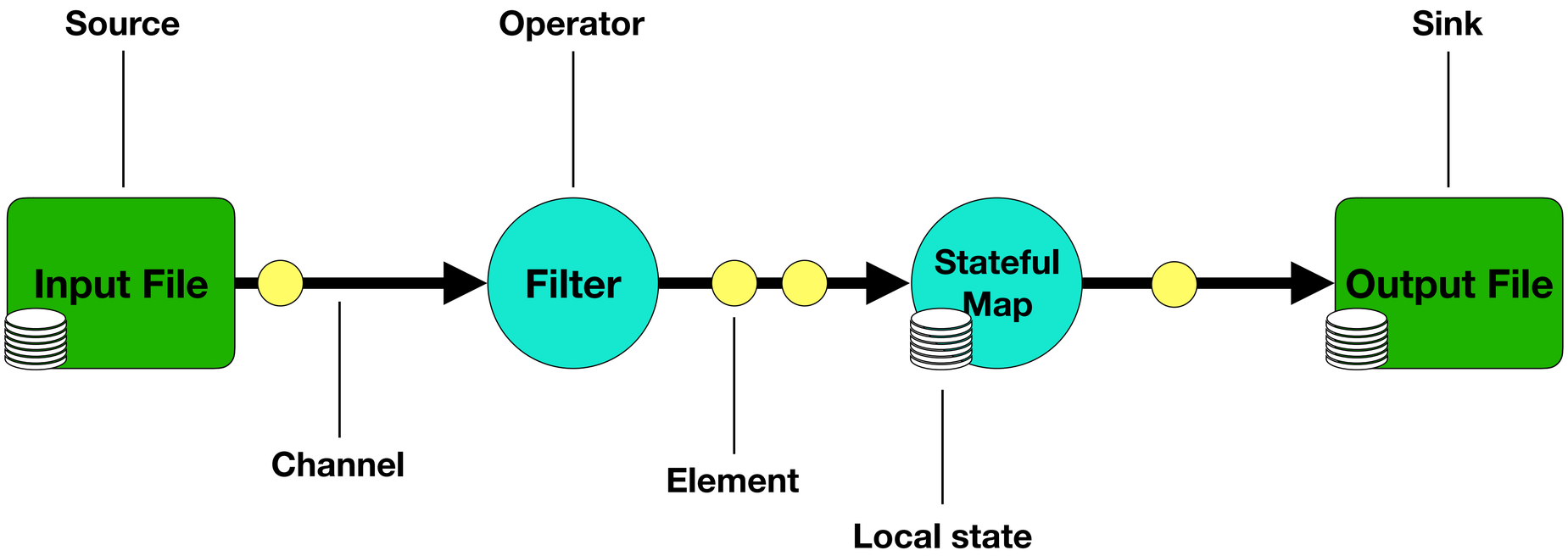
Introduction - Dataflow Graphs



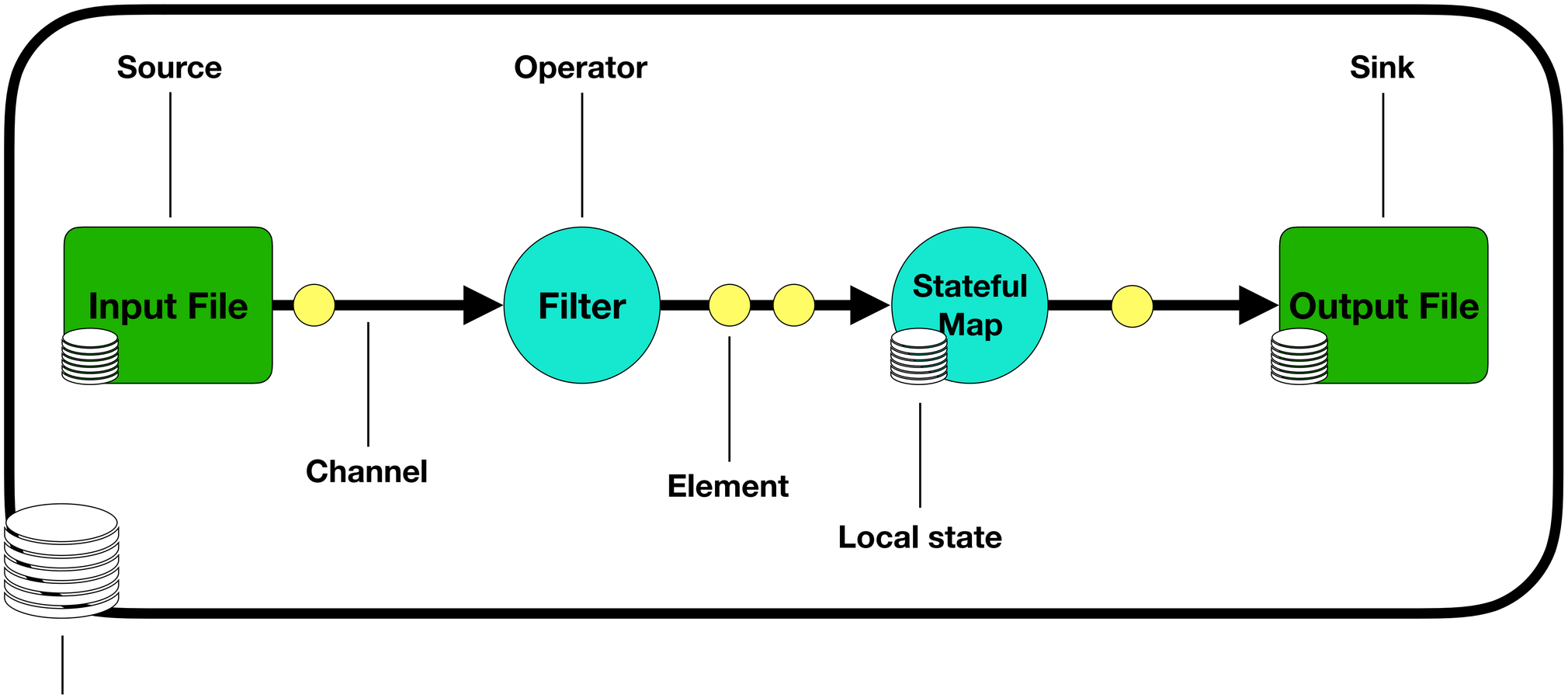
Introduction - Dataflow Graphs



Introduction - Dataflow Graphs

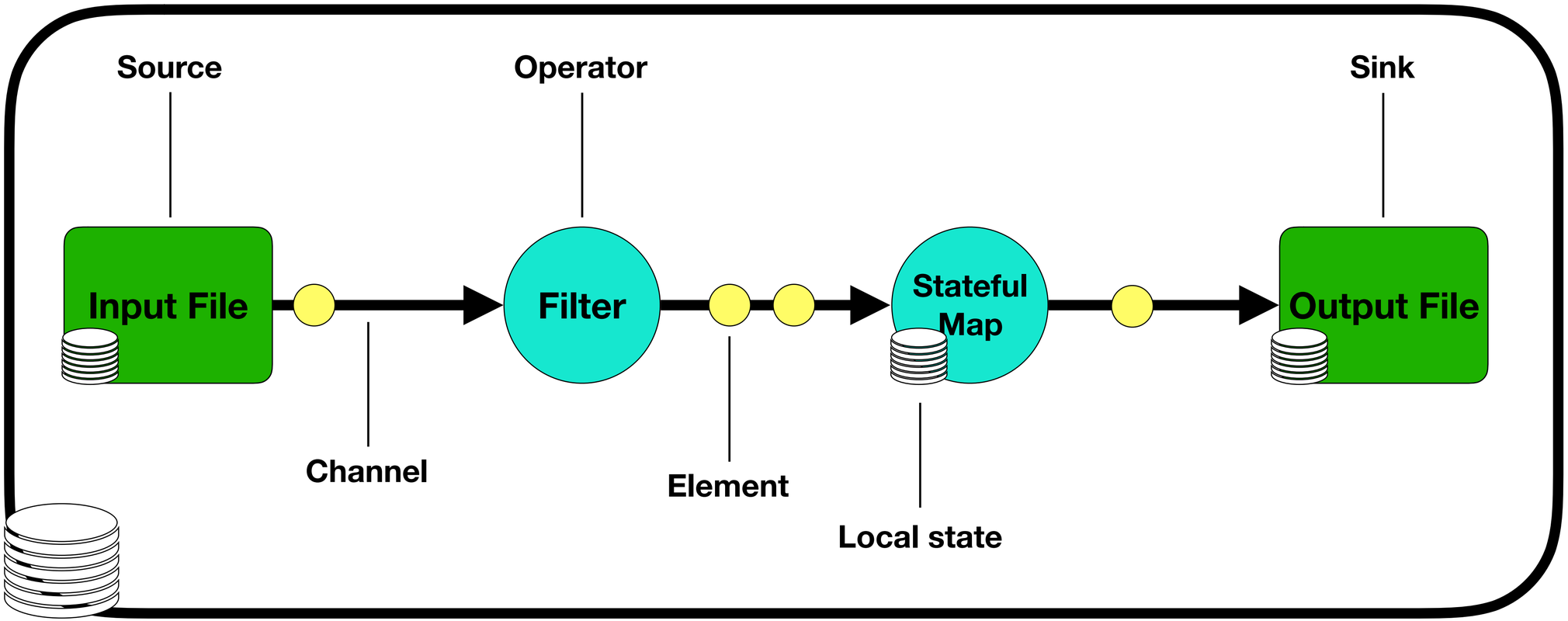


Introduction - Dataflow Graphs



Global state =
in-transit elements
+ local state of operators

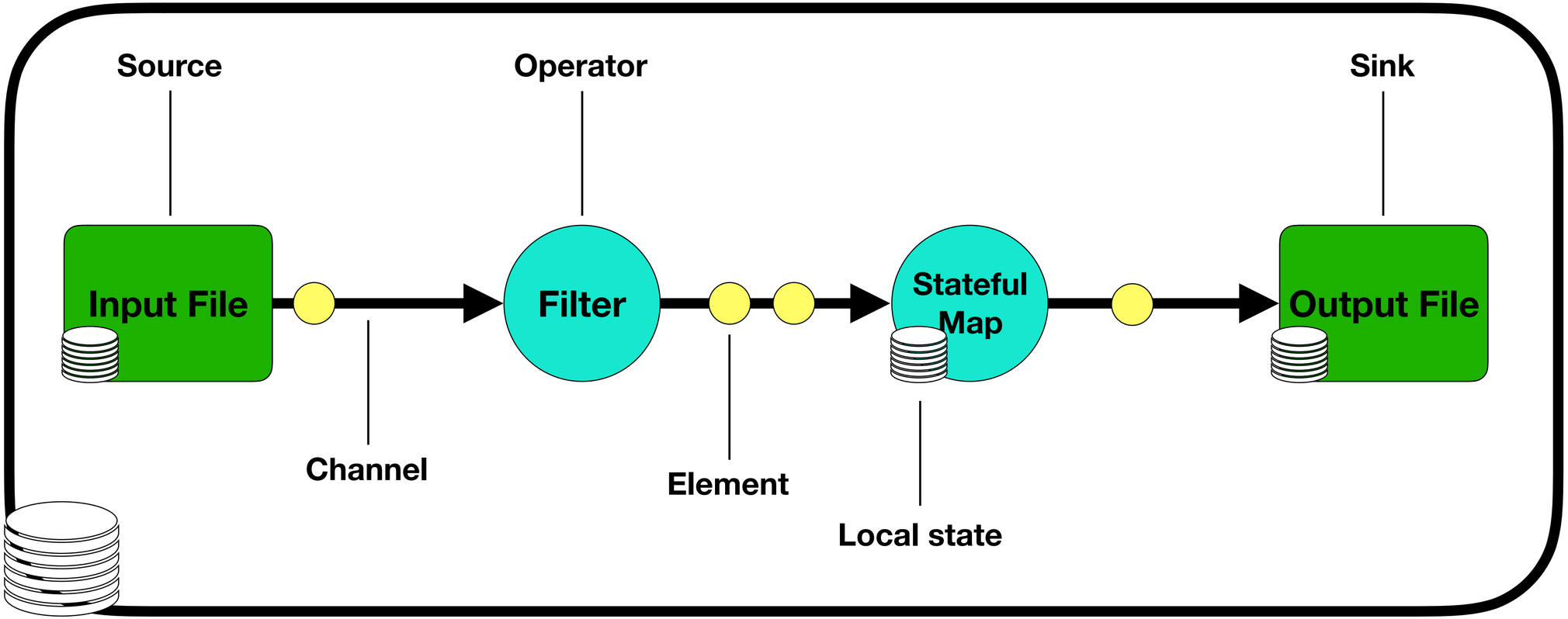
Introduction - Dataflow Graphs



Global state =
in-transit elements
+ local state of operators

➡ Reconfiguration
➡ Interactive queries

Introduction - Dataflow Graphs



Global state =
in-transit elements
+ local state of operators

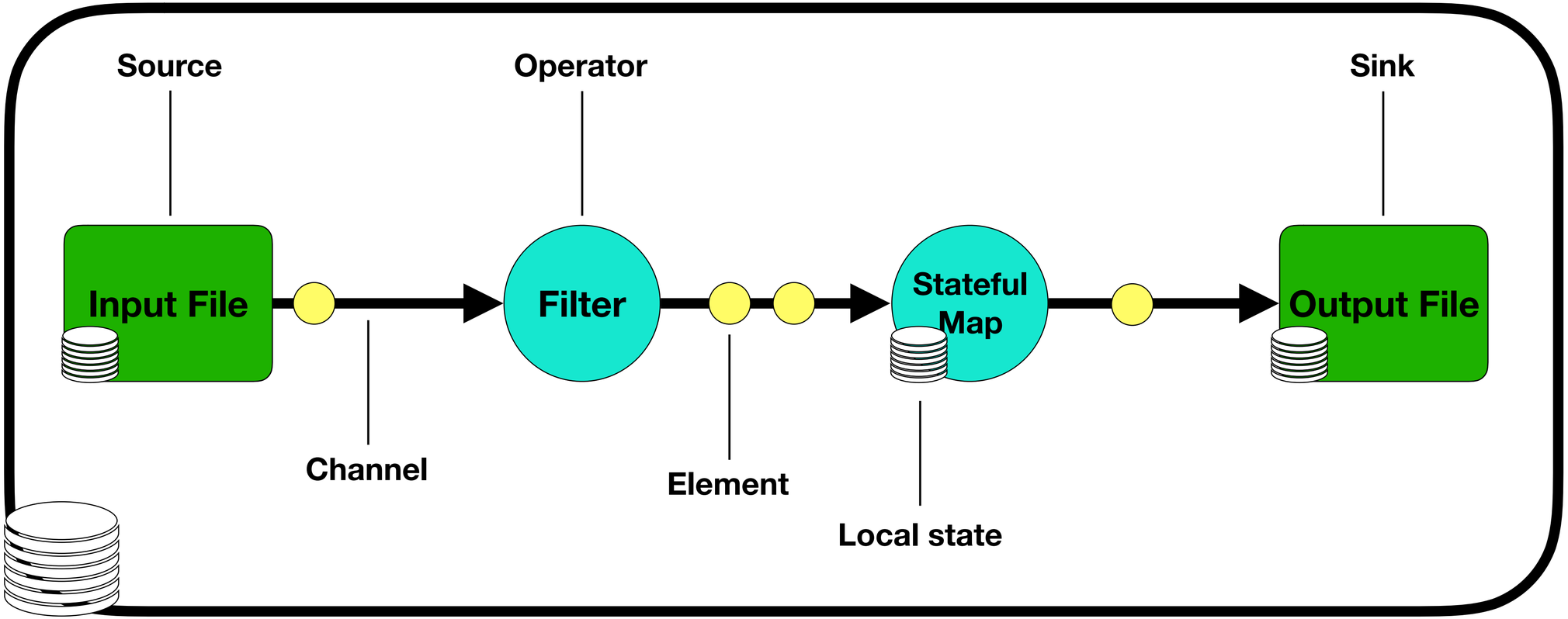
➡ Reconfiguration
➡ Interactive queries

Consistency Guarantees

Exactly-once / At-least-once Processing

Exactly-once / At-least-once Delivery

Introduction - Dataflow Graphs



Global state =
in-transit elements
+ local state of operators

➡ Reconfiguration
➡ Interactive queries

?

Consistency Guarantees

Exactly-once / At-least-once Processing

Exactly-once / At-least-once Delivery

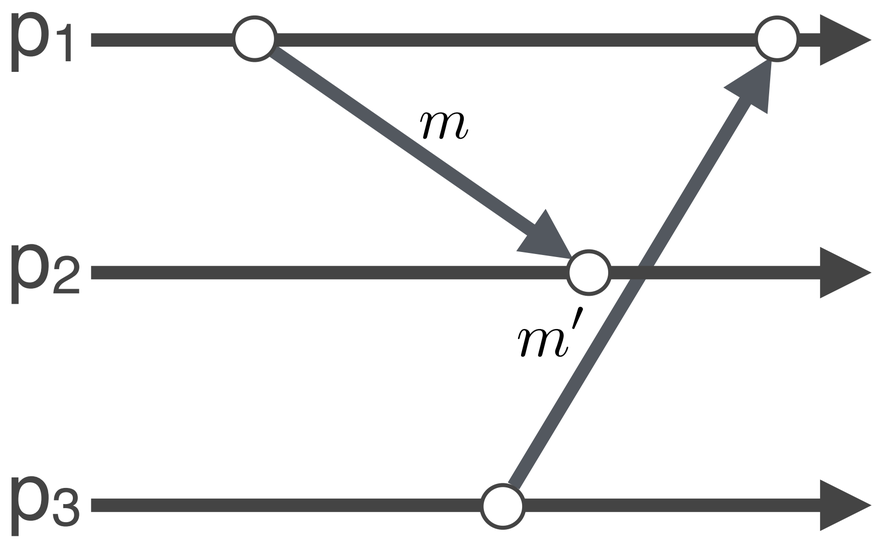
Introduction - Snapshotting Protocols

Snapshotting Protocols: Distributed Algorithms that implement **distributed cuts** in a system execution to yield a global state

Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Introduction - Snapshotting Protocols

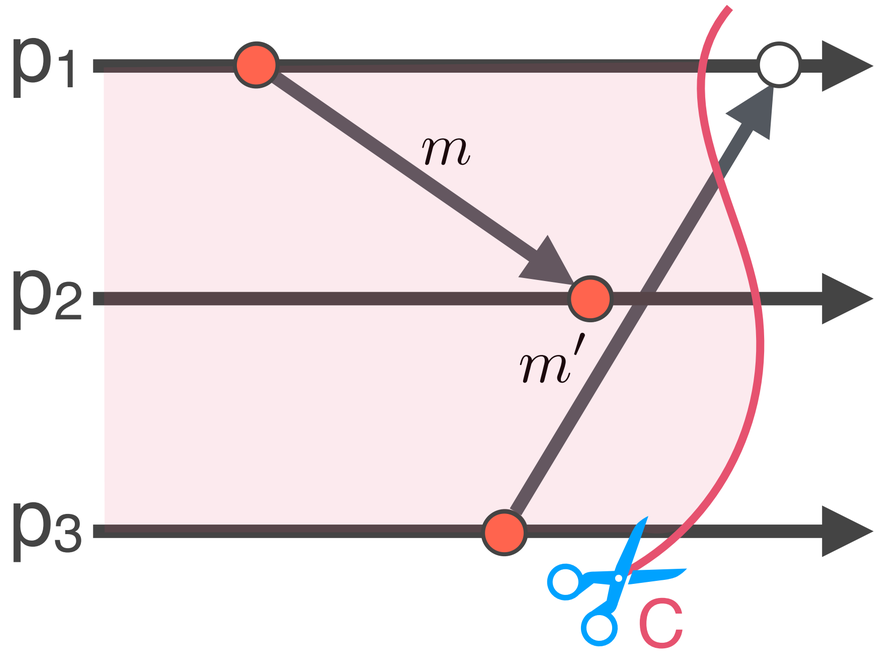
Snapshotting Protocols: Distributed Algorithms that implement **distributed cuts** in a system execution to yield a global state



Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Introduction - Snapshotting Protocols

Snapshotting Protocols: Distributed Algorithms that implement **distributed cuts** in a system execution to yield a global state

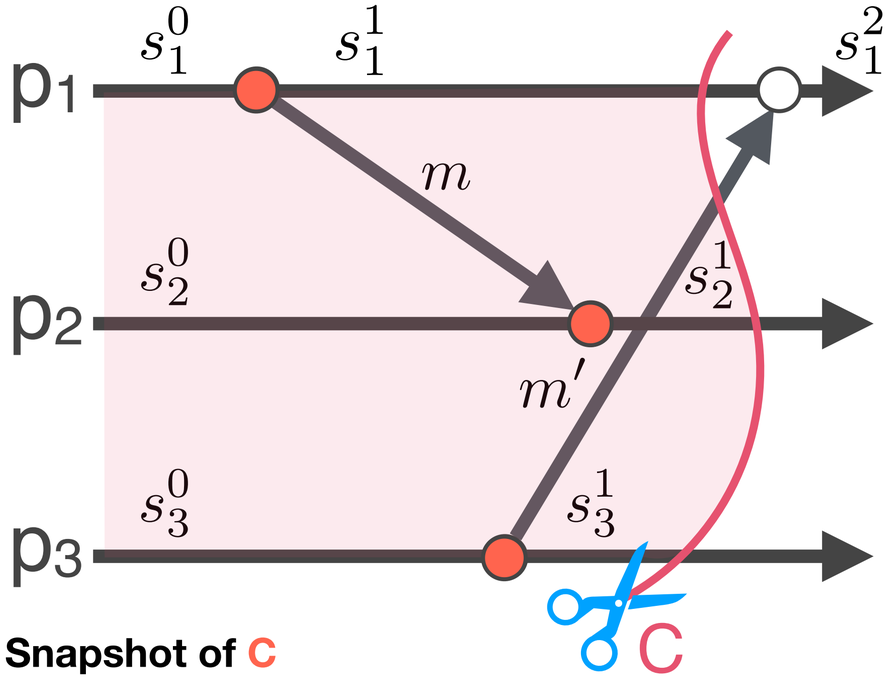


Consistent cut
(Satisfies causality)

Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Introduction - Snapshotting Protocols

Snapshotting Protocols: Distributed Algorithms that implement **distributed cuts** in a system execution to yield a global state



Snapshot of C

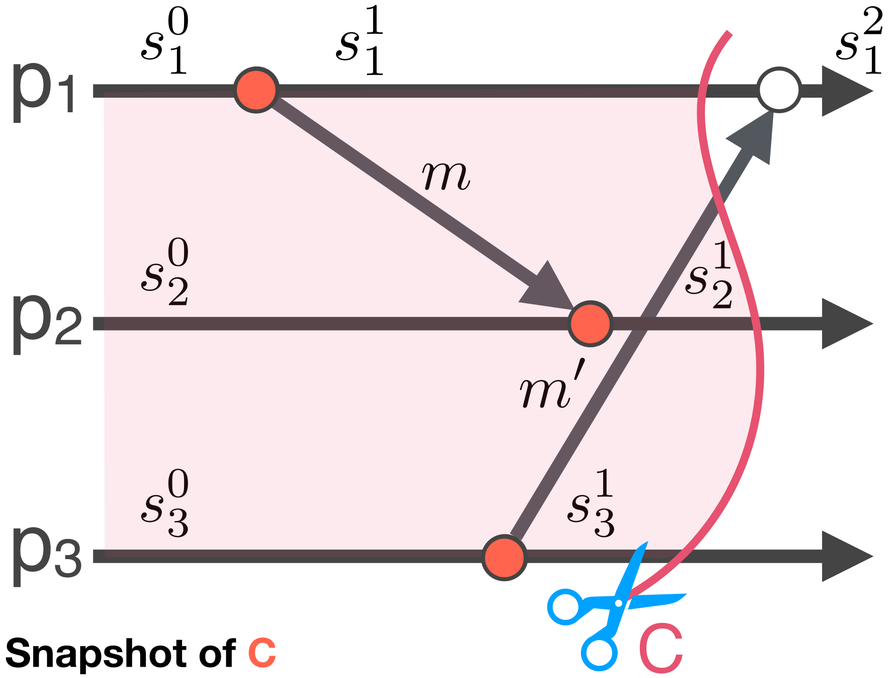
$\{s_1^1, s_2^1, s_3^1\}$
$\{m'\}$

Consistent cut (Satisfies causality)

Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Introduction - Snapshotting Protocols

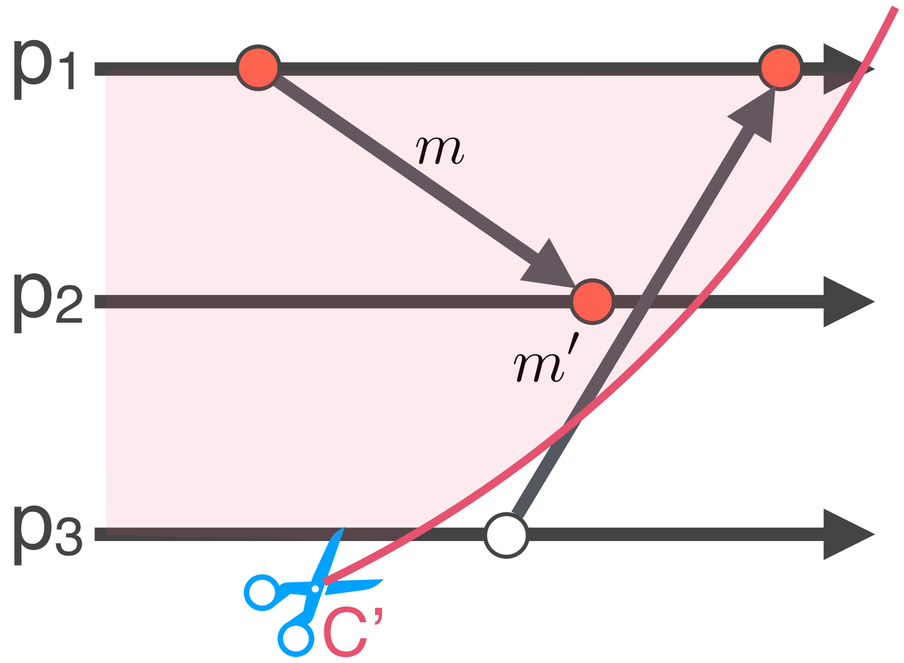
Snapshotting Protocols: Distributed Algorithms that implement **distributed cuts** in a system execution to yield a global state



Snapshot of C

$\{s_1^1, s_2^1, s_3^1\}$
$\{m'\}$

Consistent cut (Satisfies causality)



Inconsistent cut (Violates causality)

Slide borrowed from Paris Carbone, LADIS 2018, Asynchronous Epoch Commits

Apache Flink

Flink - Problem statement

Stream processing systems lack **state abstractions**

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

➡ **Maintenance & Transactional costs**

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

➡ **Maintenance & Transactional costs**

➡ **Operational challenges** (scaling, failure-recovery, patches)

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

➡ **Maintenance & Transactional costs**

➡ **Operational challenges** (scaling, failure-recovery, patches)

Or handled by **micro-batching ➡ Processing latency**

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

➡ **Maintenance & Transactional costs**

➡ **Operational challenges** (scaling, failure-recovery, patches)

Or handled by **micro-batching ➡ Processing latency**

Question: Can state management be **interleaved** with processing?

Flink - Problem statement

Stream processing systems lack **state abstractions**

State is either handled **externally** by the user (e.g. DBMS) ...

➡ **Maintenance & Transactional costs**

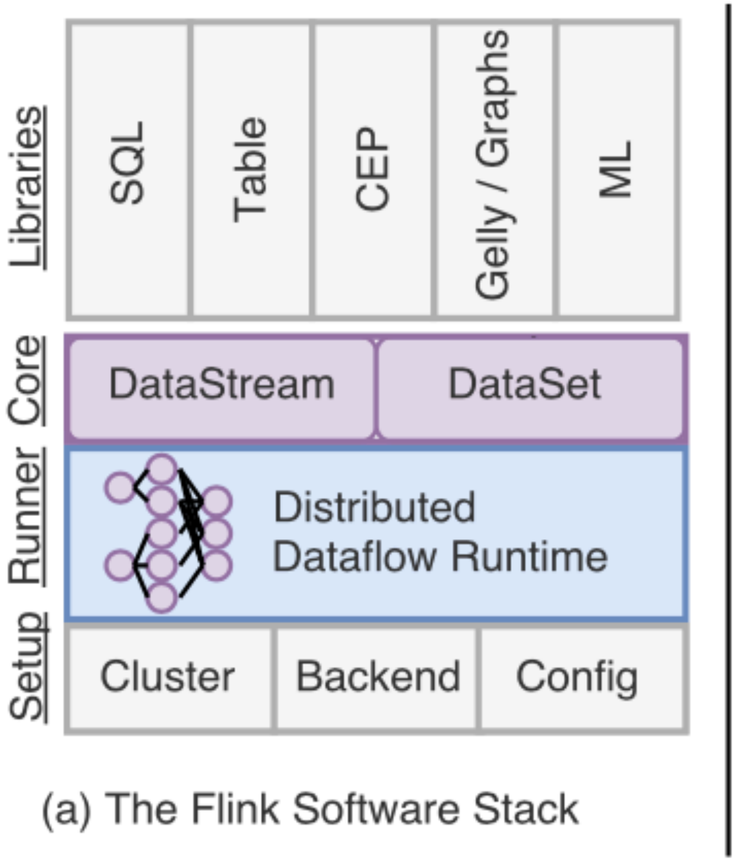
➡ **Operational challenges** (scaling, failure-recovery, patches)

Or handled by **micro-batching ➡ Processing latency**

Question: Can state management be **interleaved** with processing?

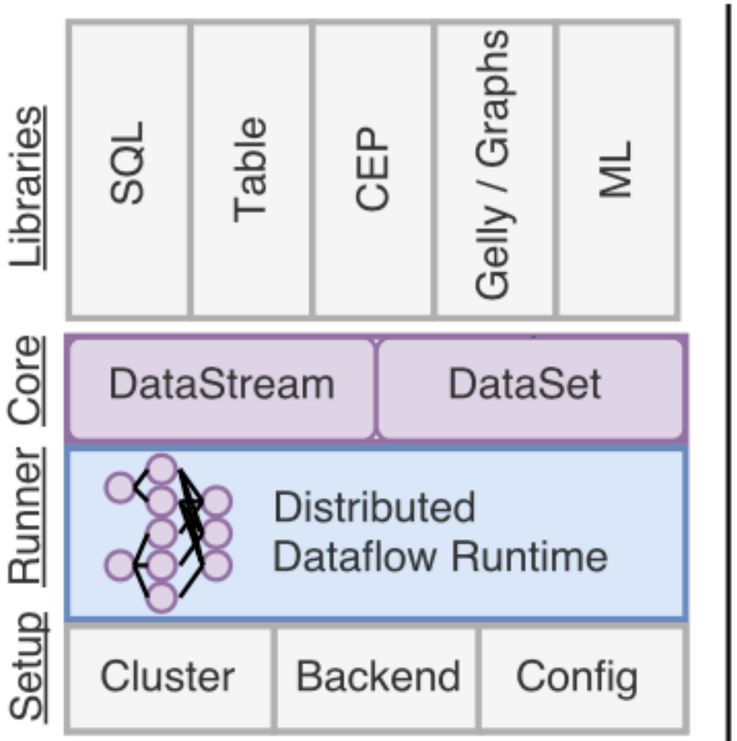
Solution: Distributed Snapshots

Flink - System Overview



(a) The Flink Software Stack

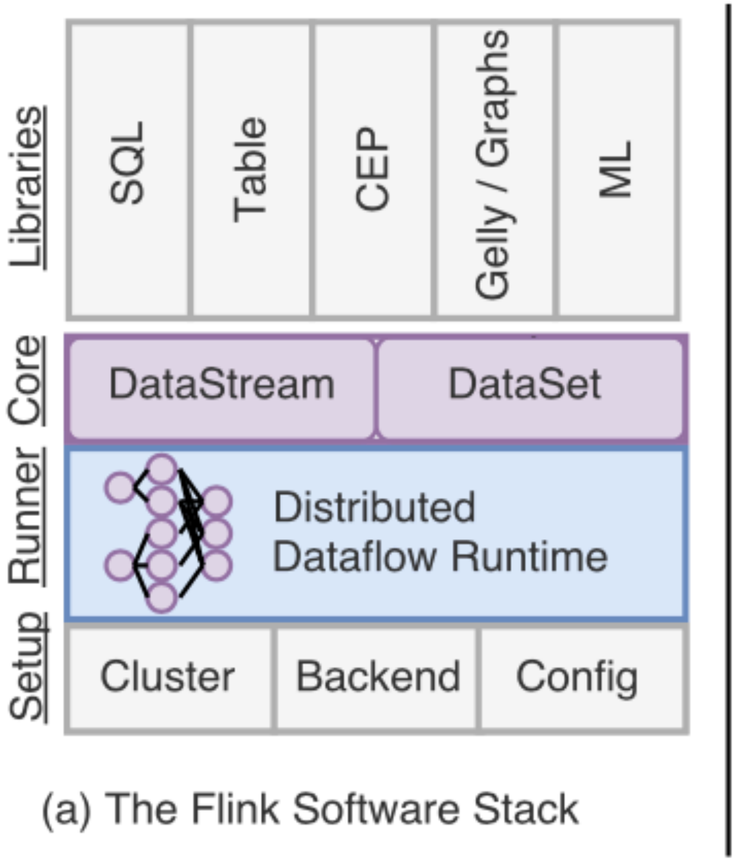
Flink - System Overview



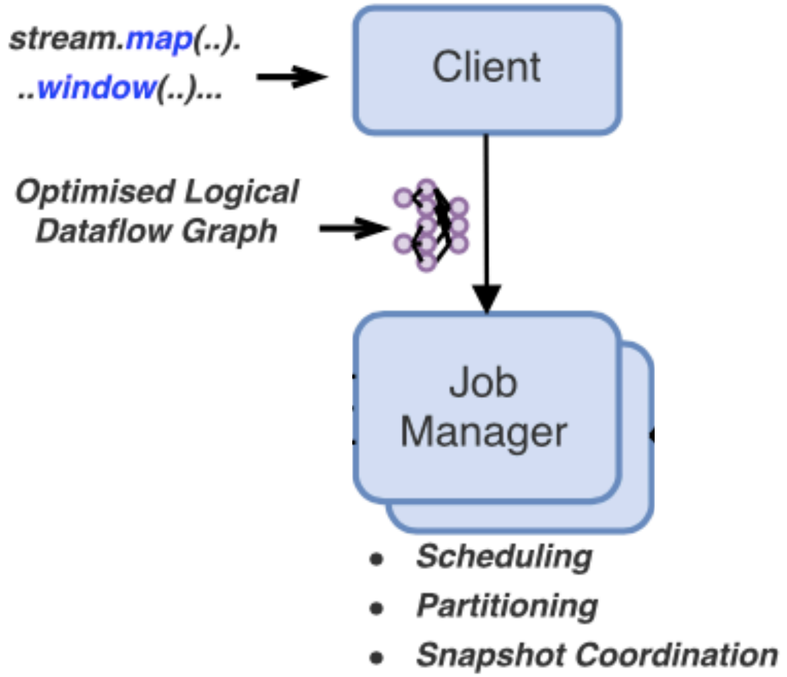
(a) The Flink Software Stack

(b) The Flink System Architecture

Flink - System Overview

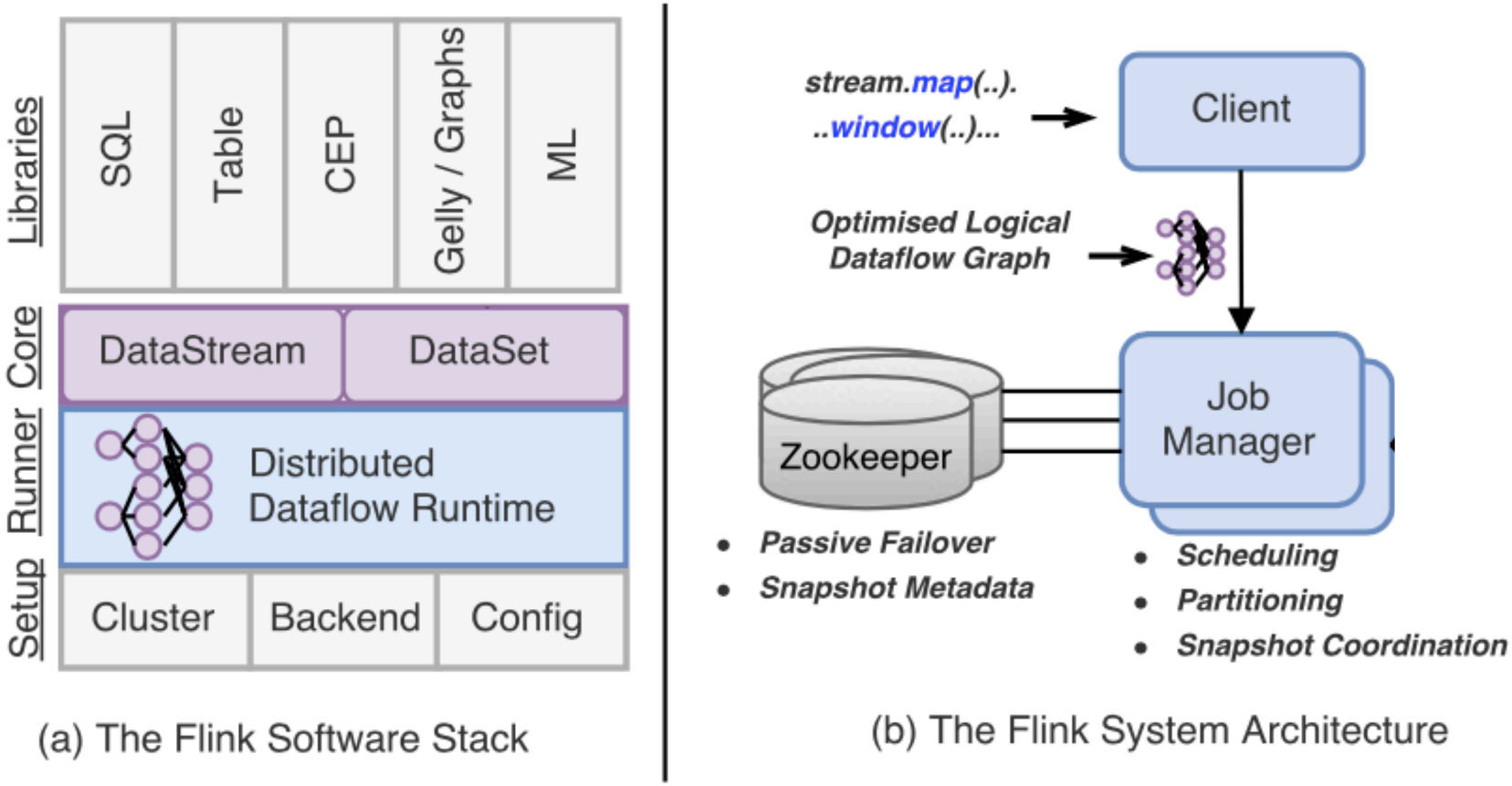


(a) The Flink Software Stack



(b) The Flink System Architecture

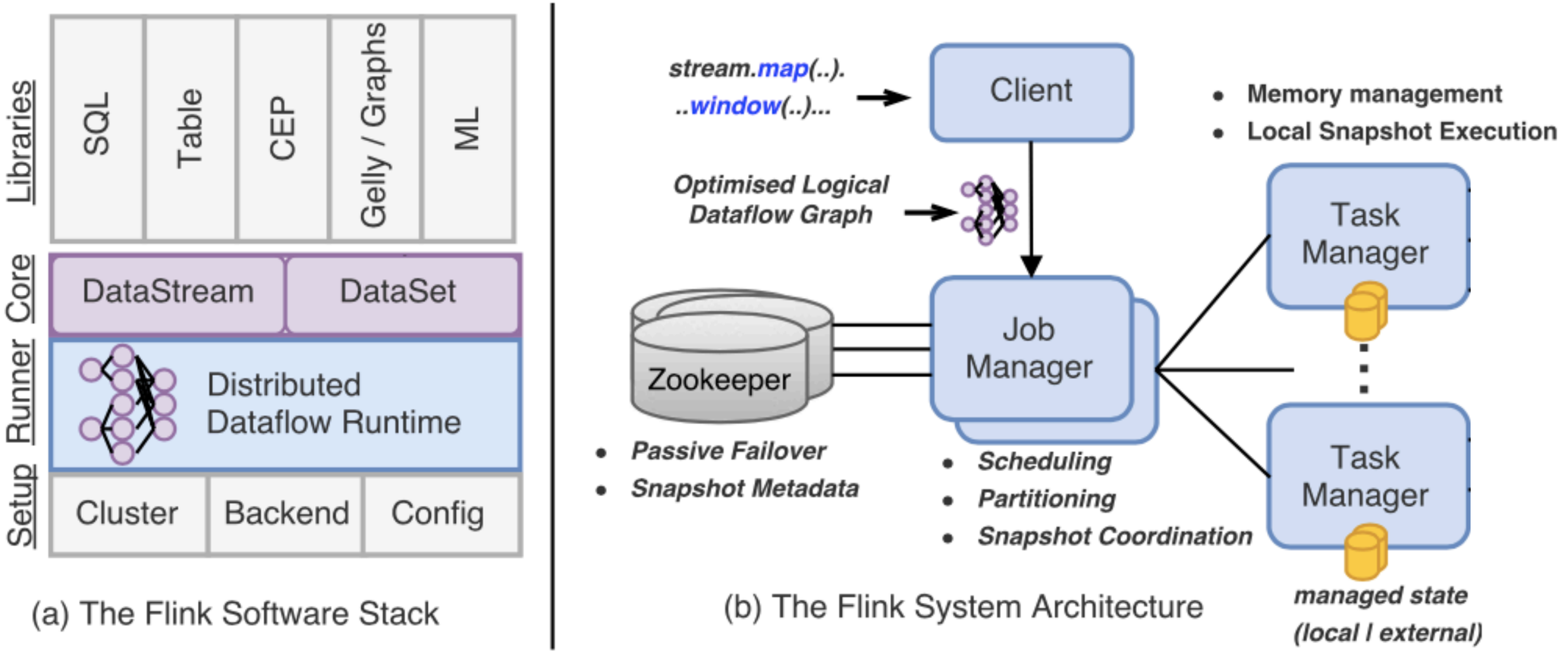
Flink - System Overview



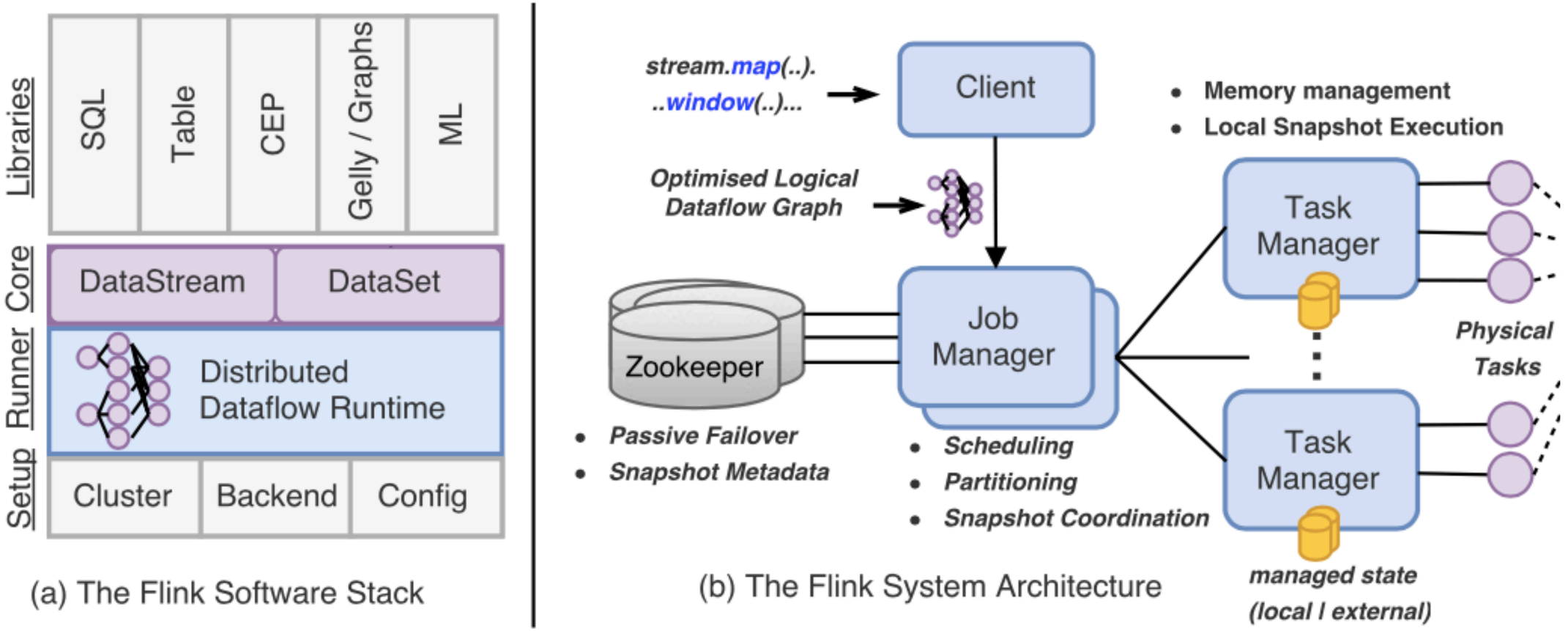
(a) The Flink Software Stack

(b) The Flink System Architecture

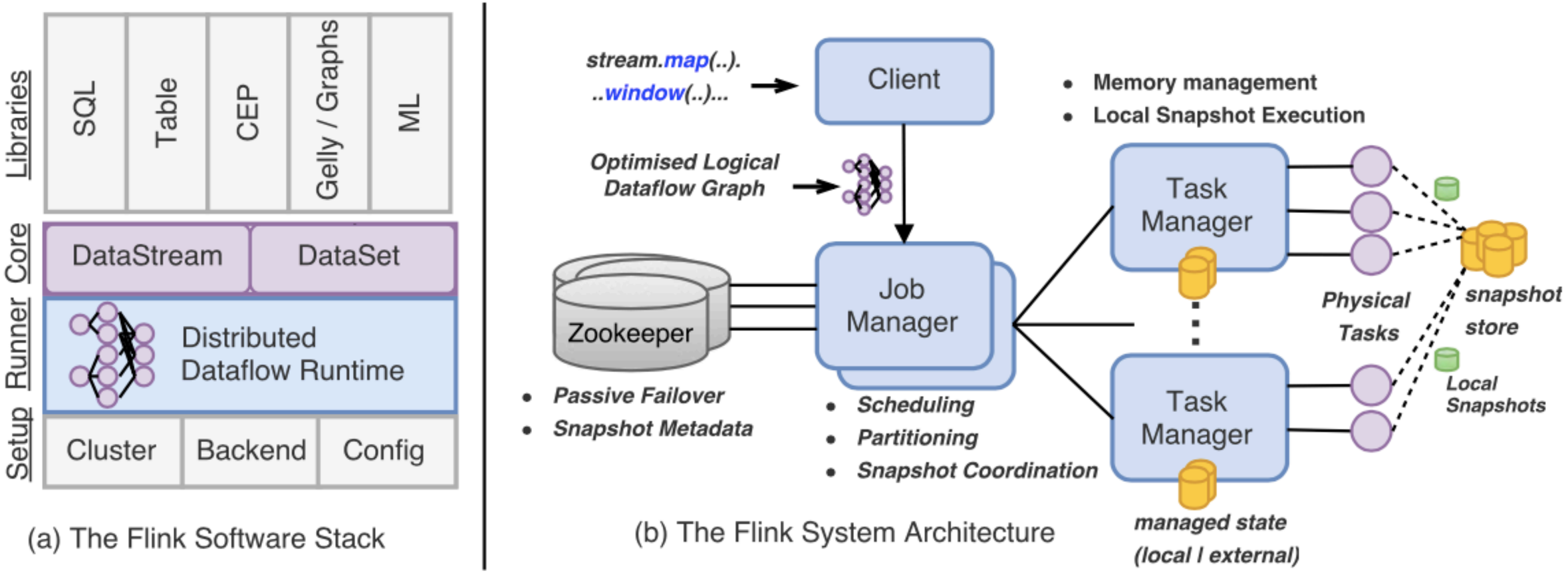
Flink - System Overview



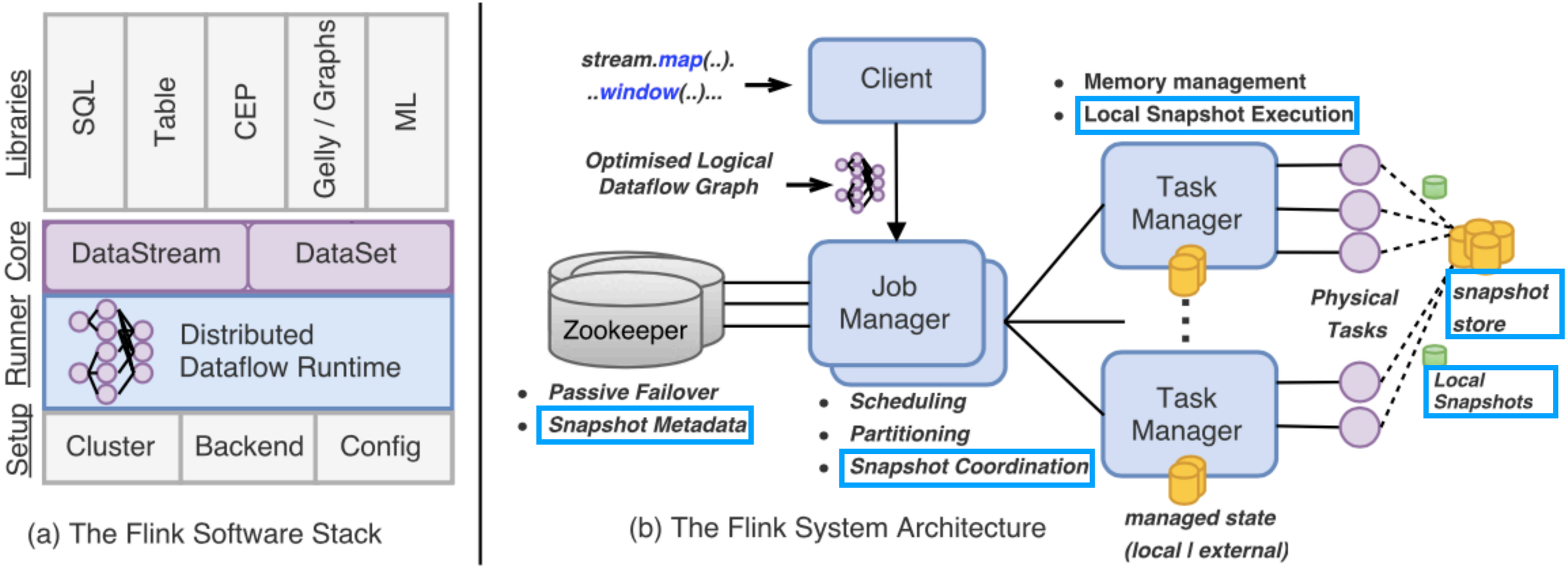
Flink - System Overview



Flink - System Overview



Flink - System Overview



How do snapshots work?

Recap: Chandy-Lamport - Consistent Snapshots

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels

Recap: Chandy-Lamport - Consistent Snapshots

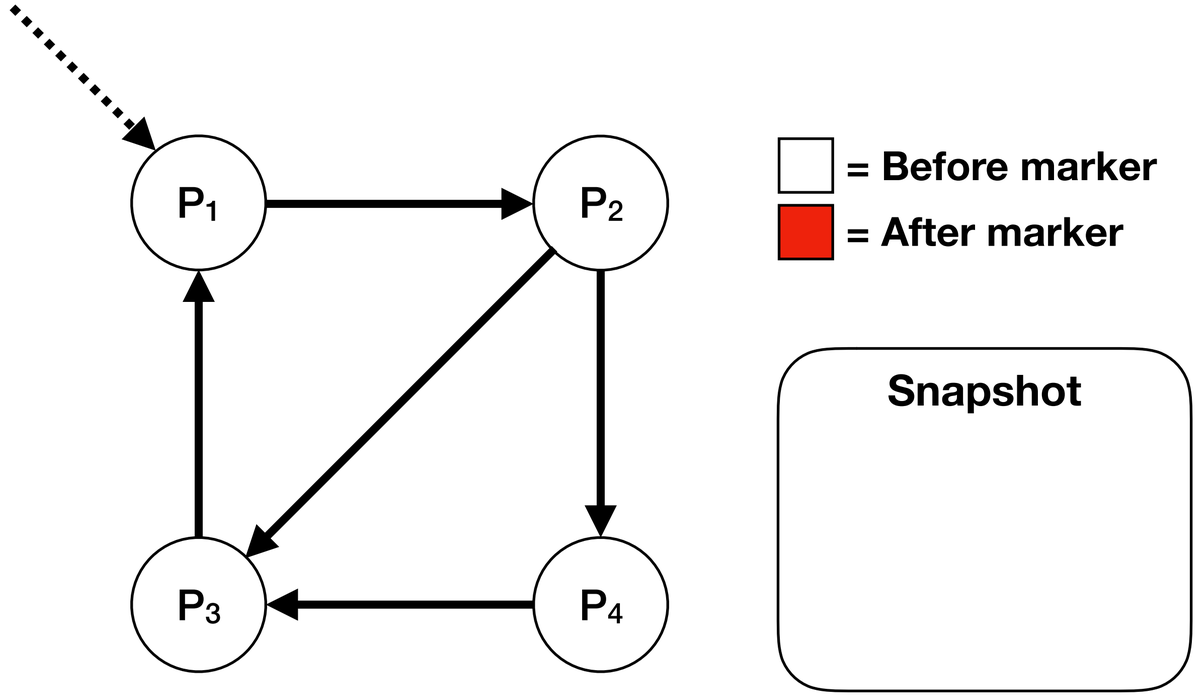
- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

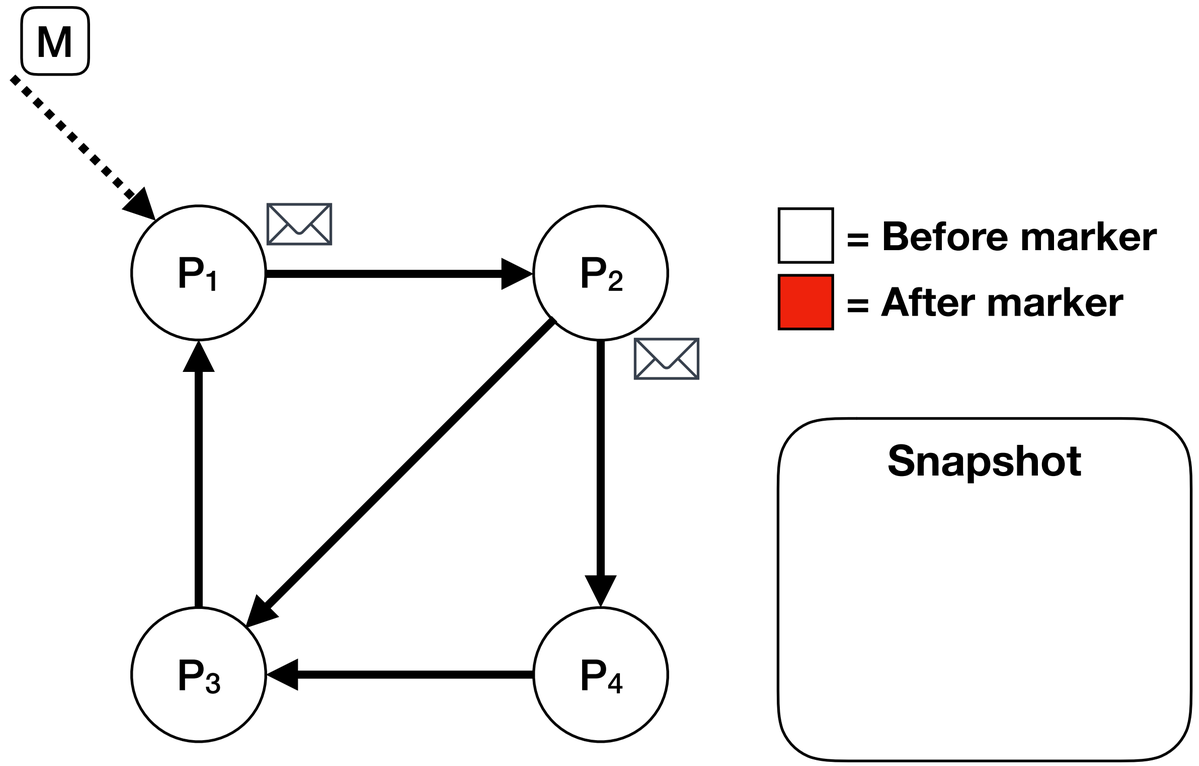
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



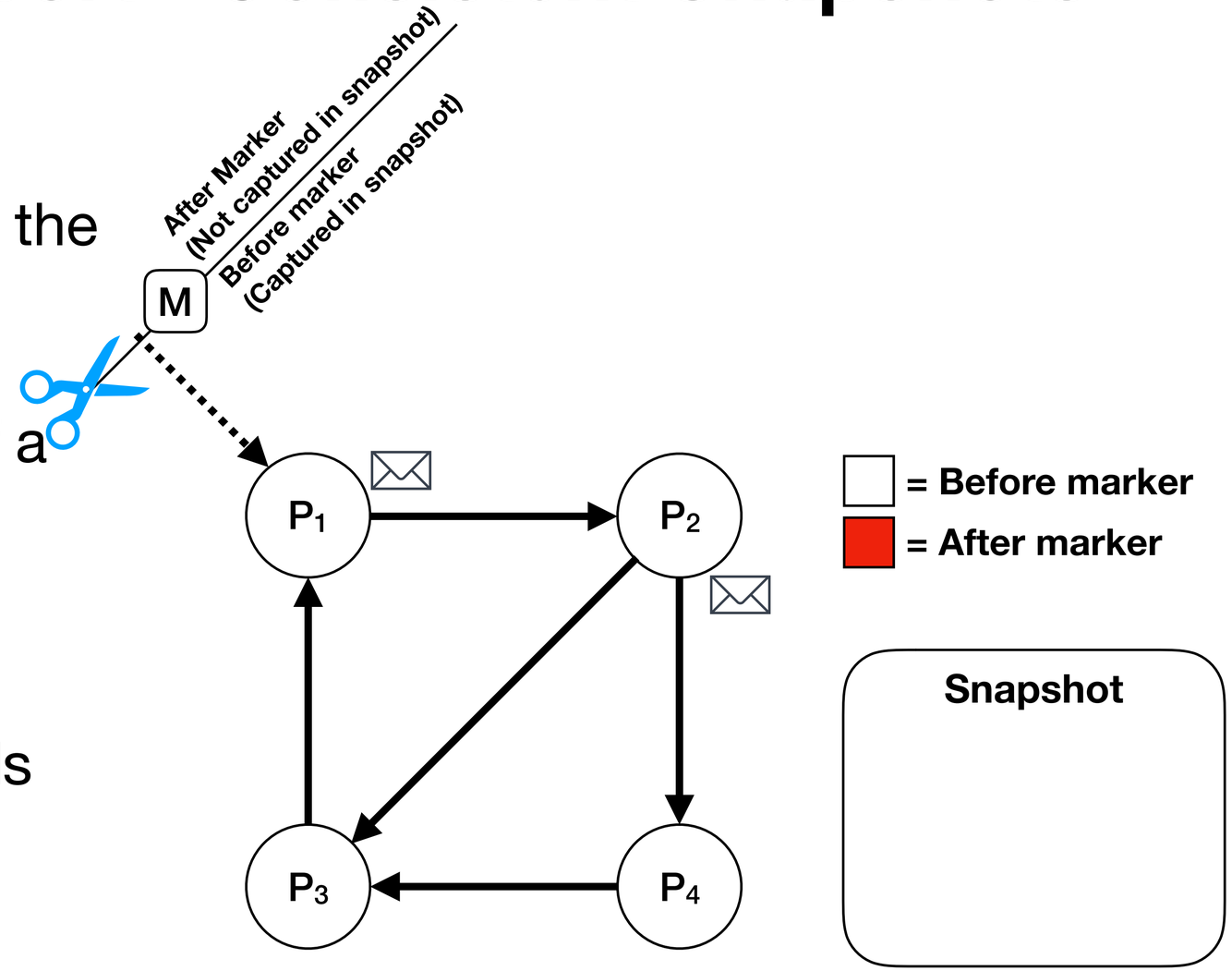
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



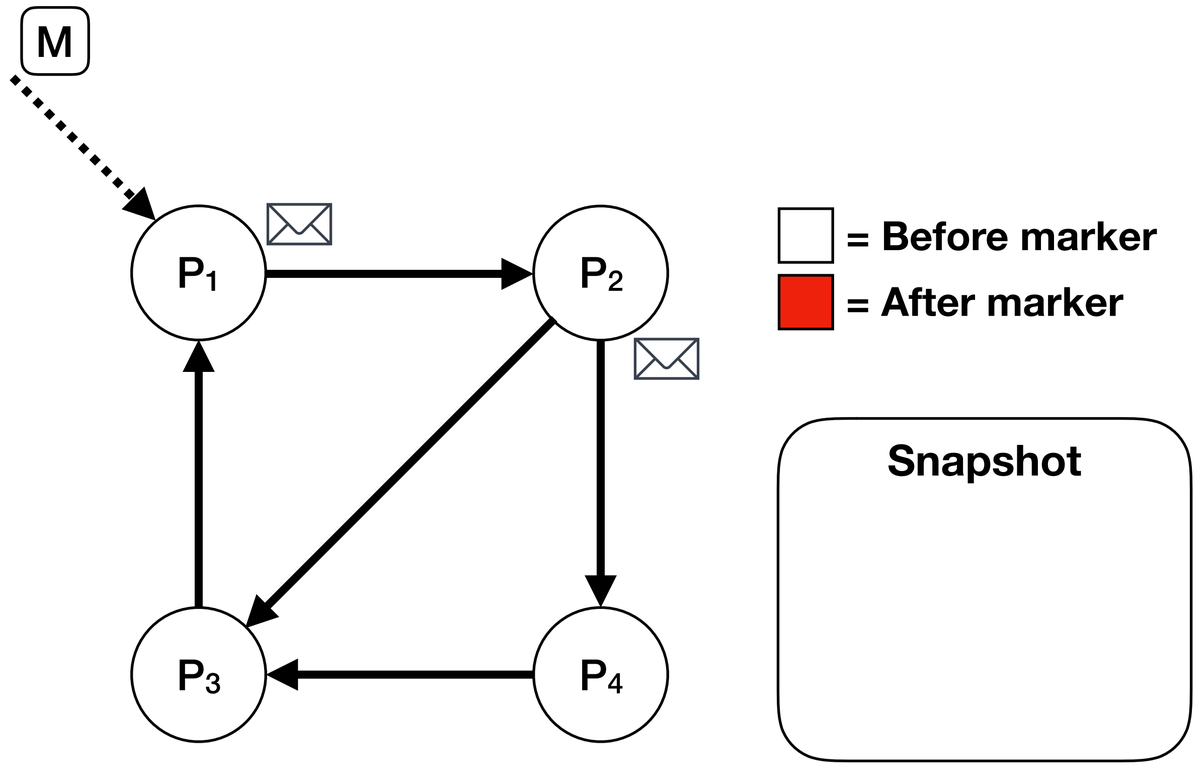
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



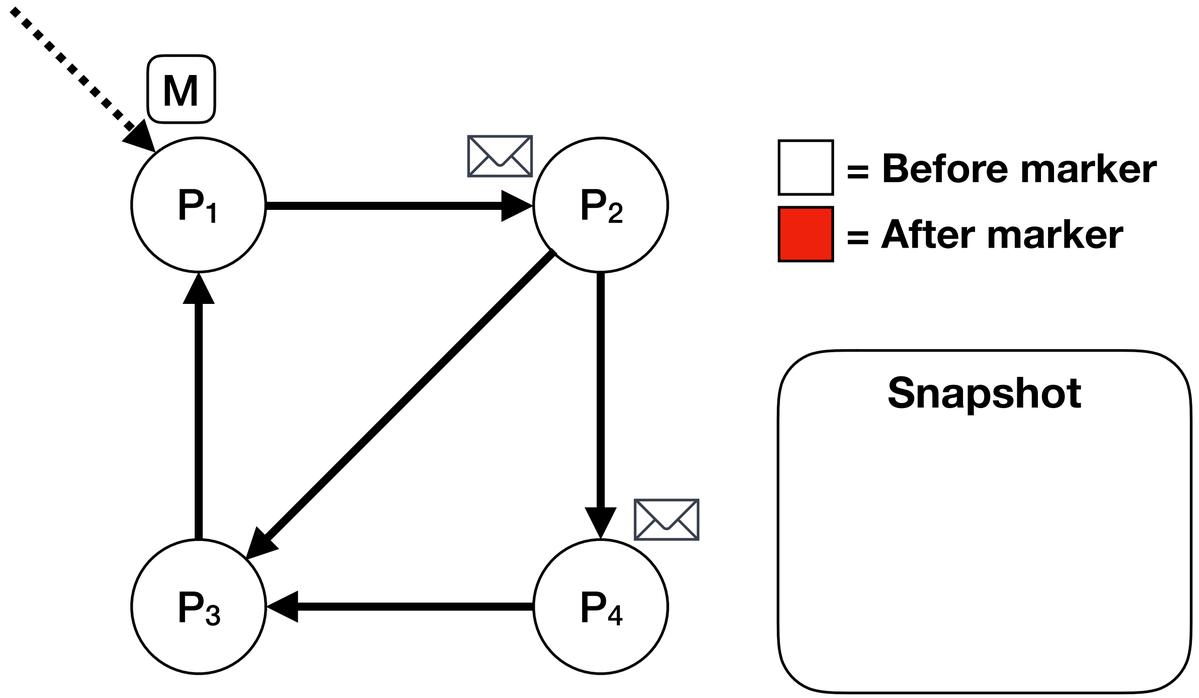
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



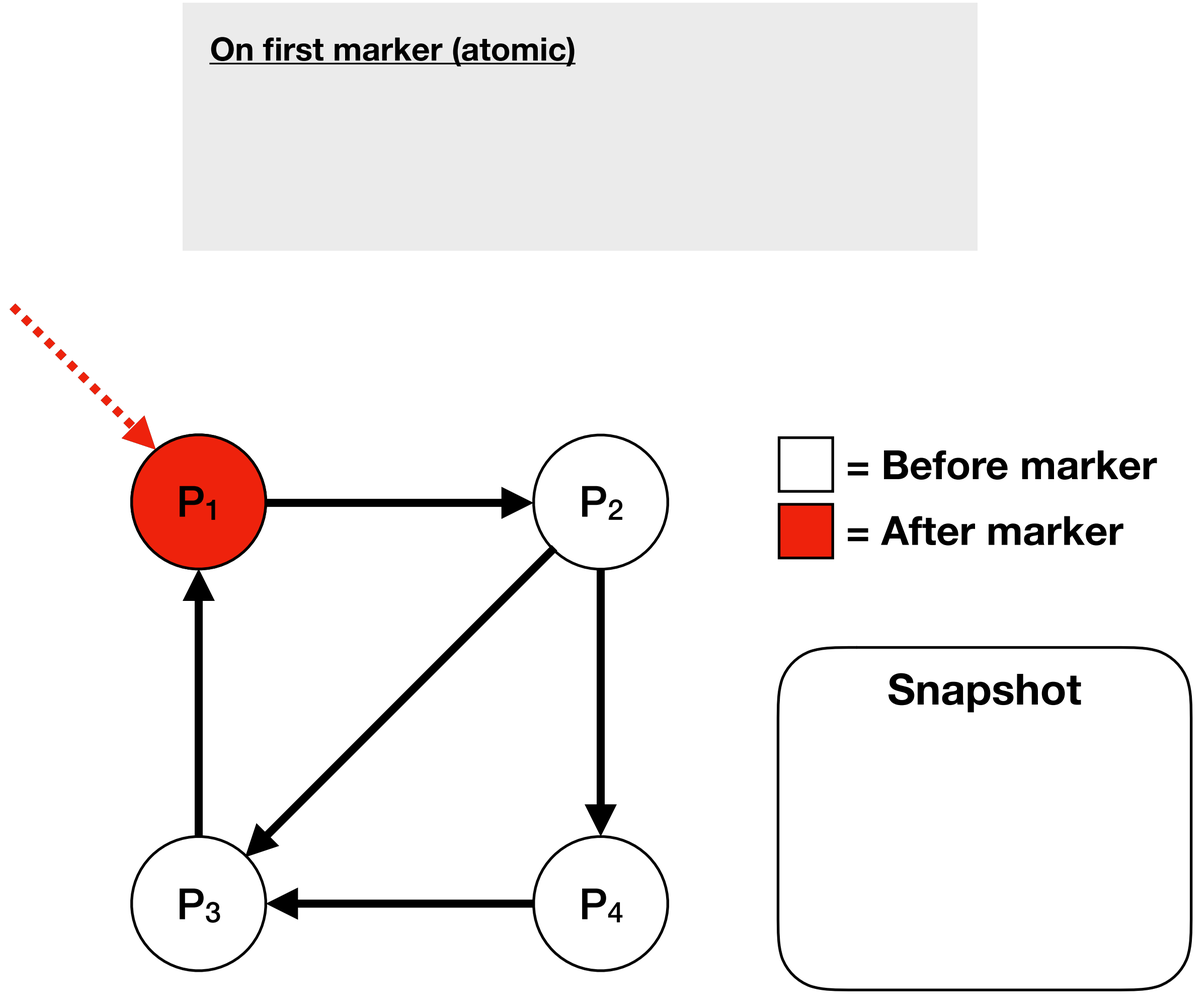
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



Recap: Chandy-Lamport - Consistent Snapshots

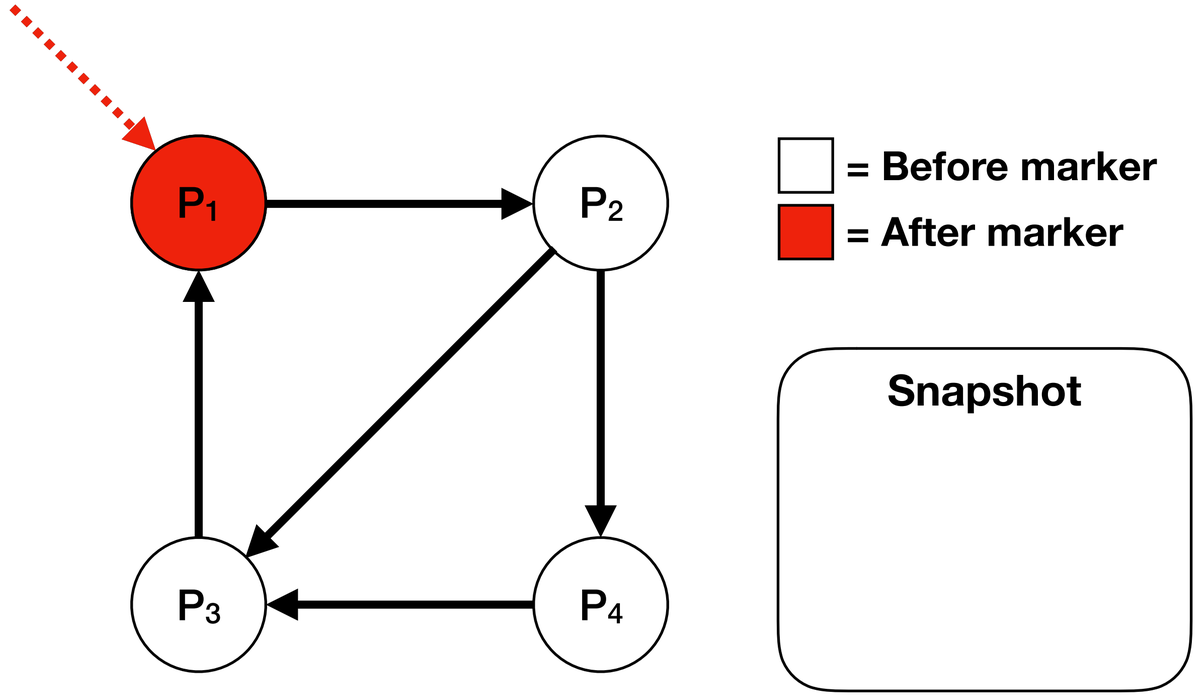
- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

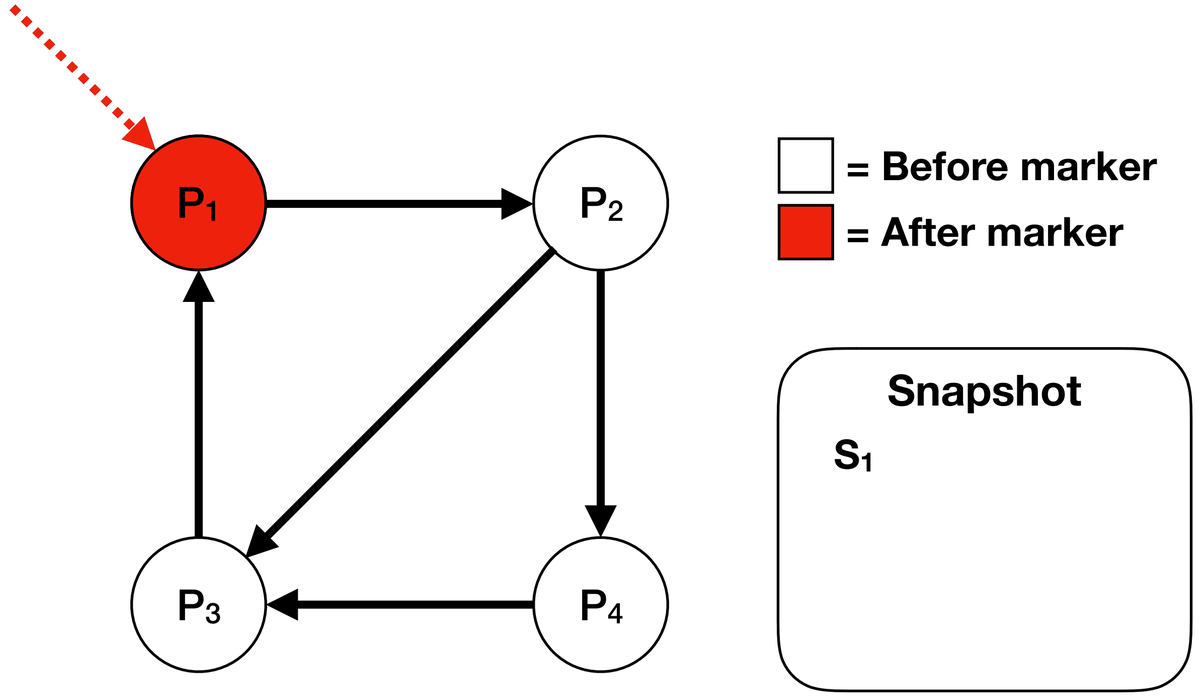
On first marker (atomic)
1. **Capture** internal state



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

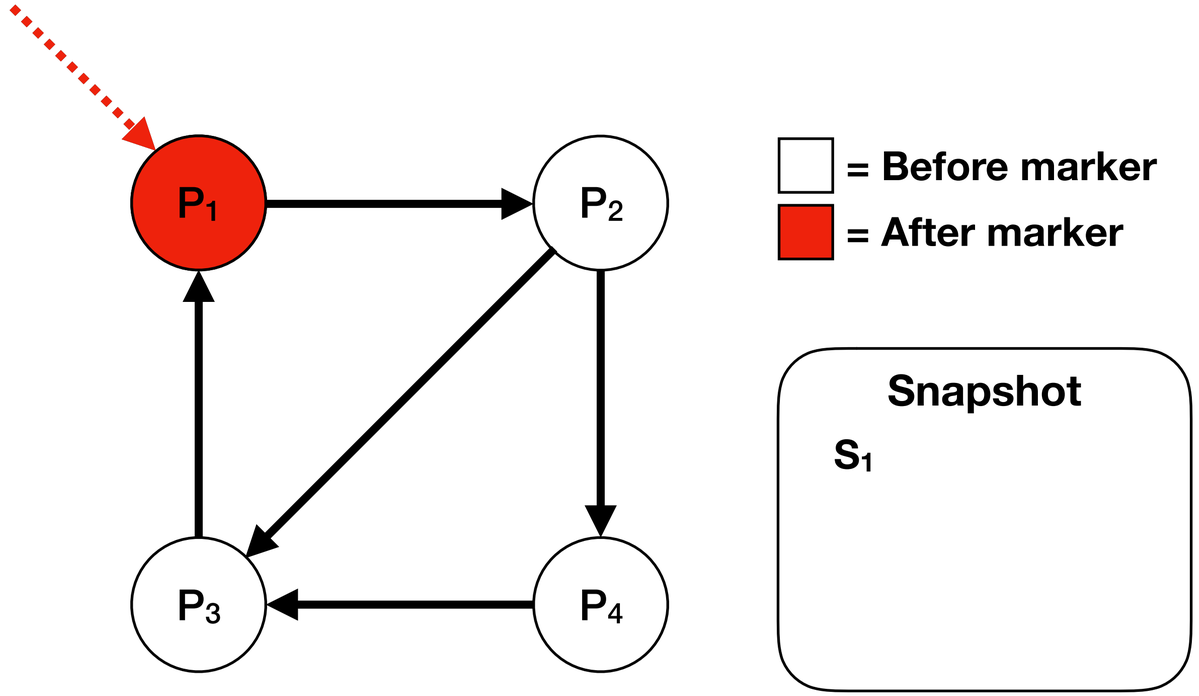
On first marker (atomic)
1. **Capture** internal state



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

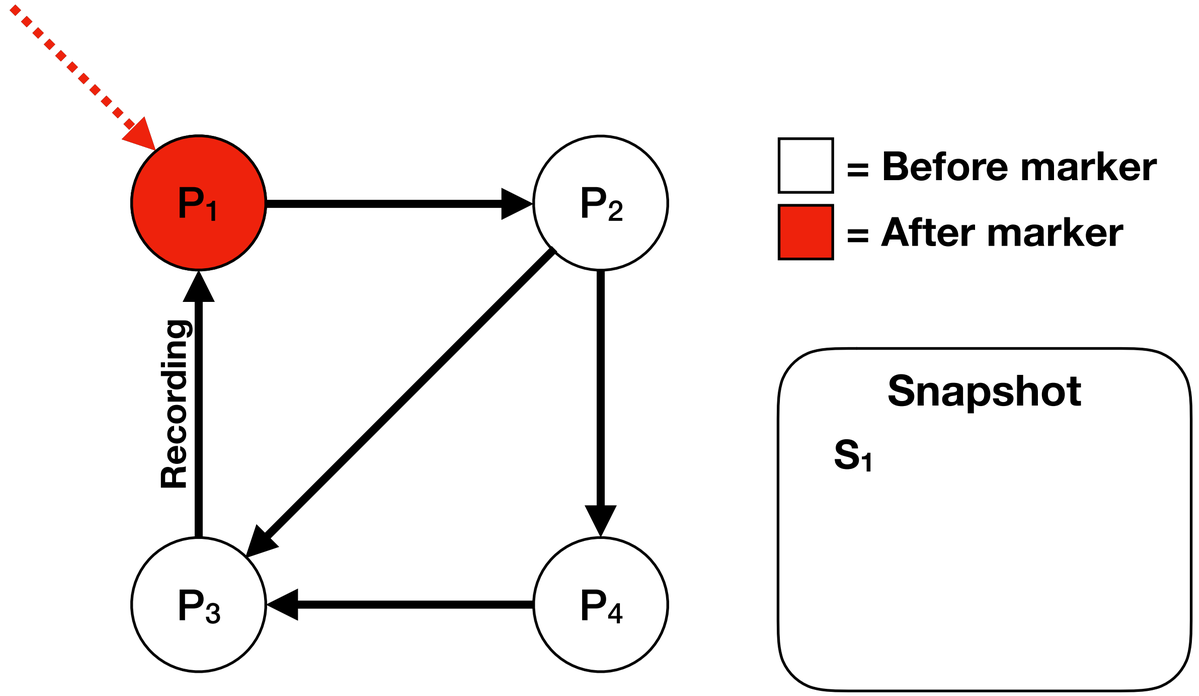
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

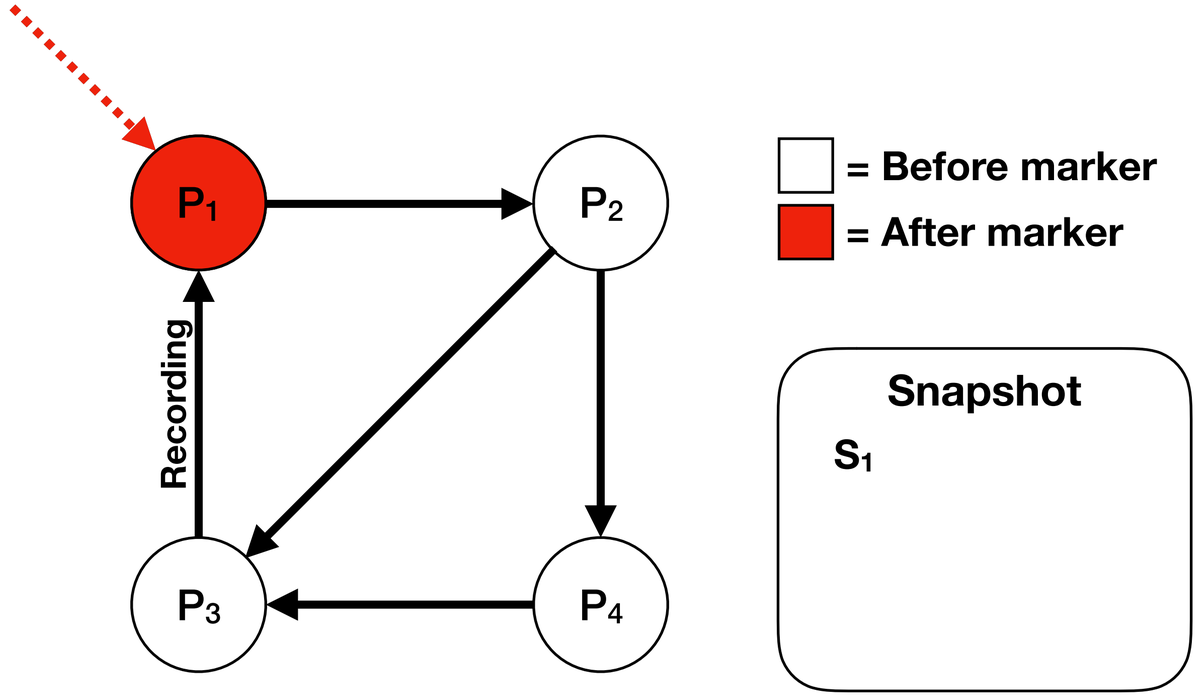
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

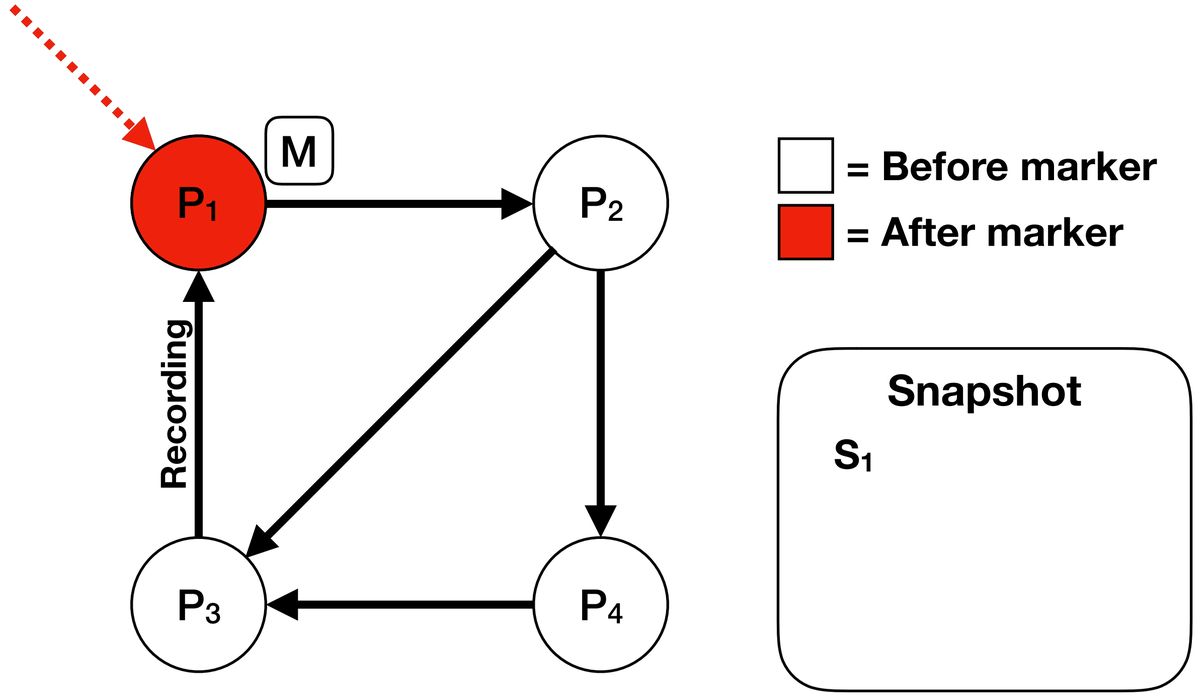
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker

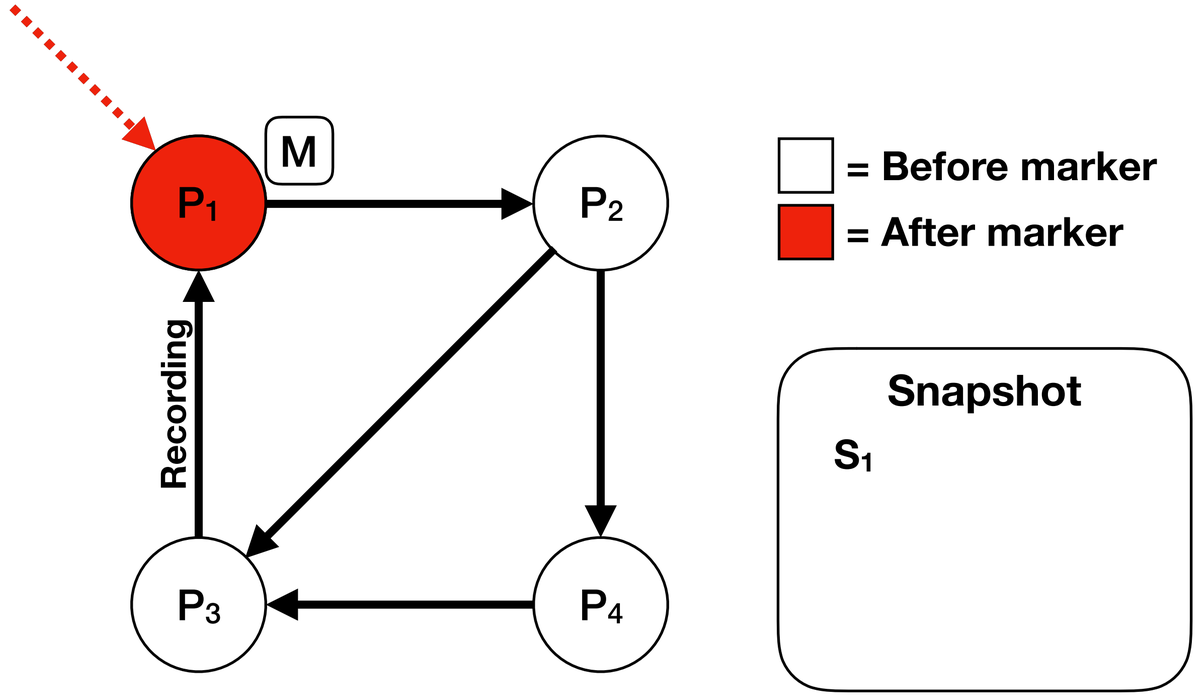


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)

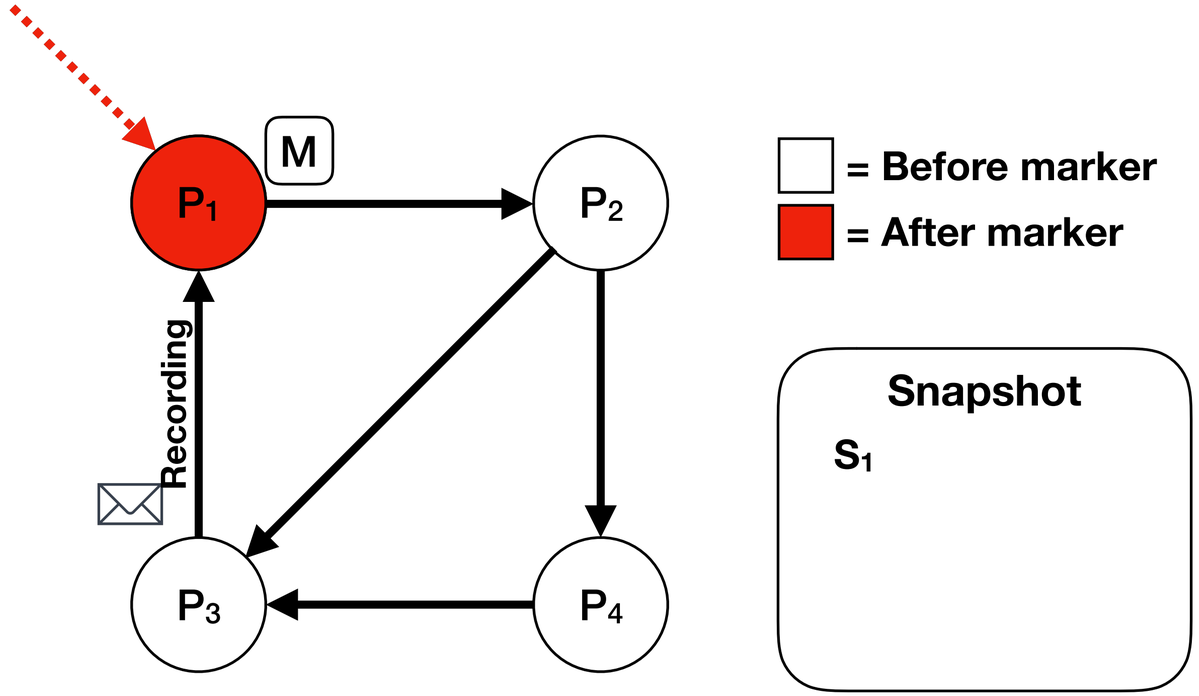
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

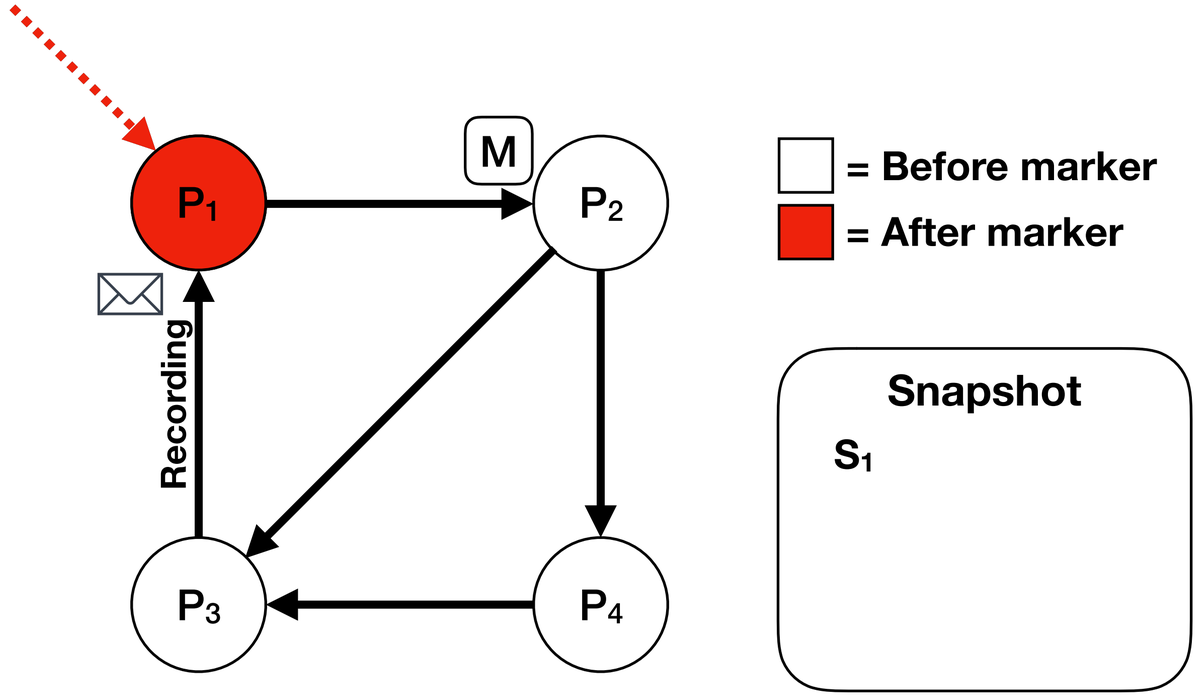
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

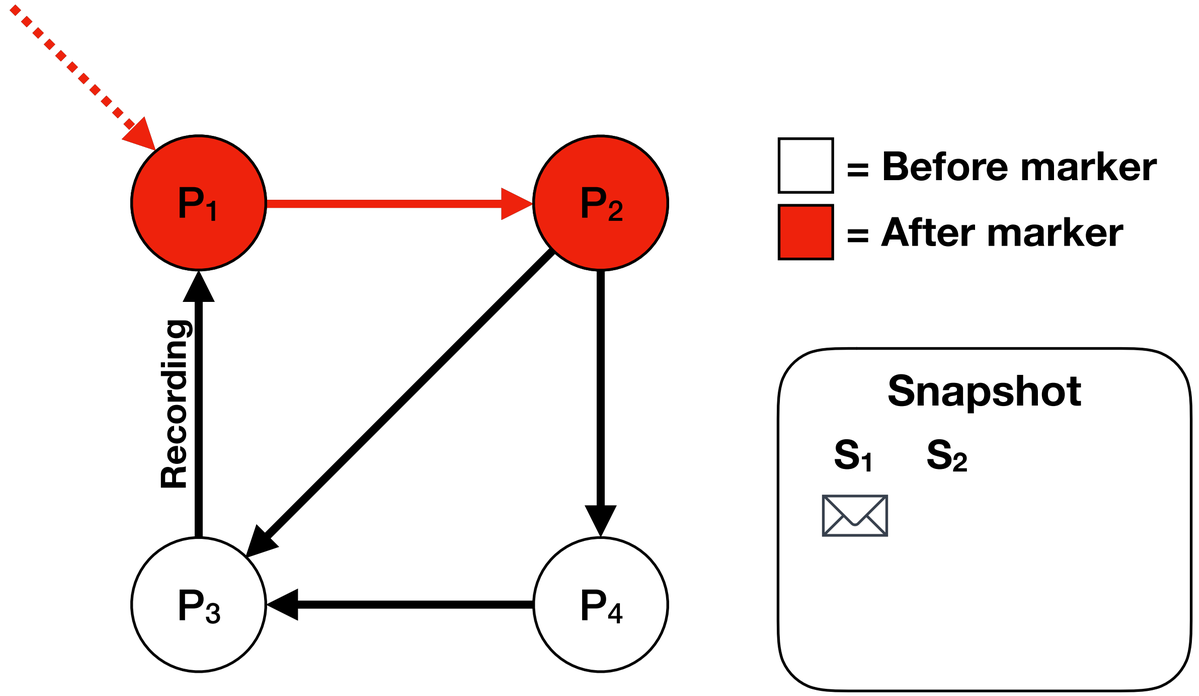


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)

1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

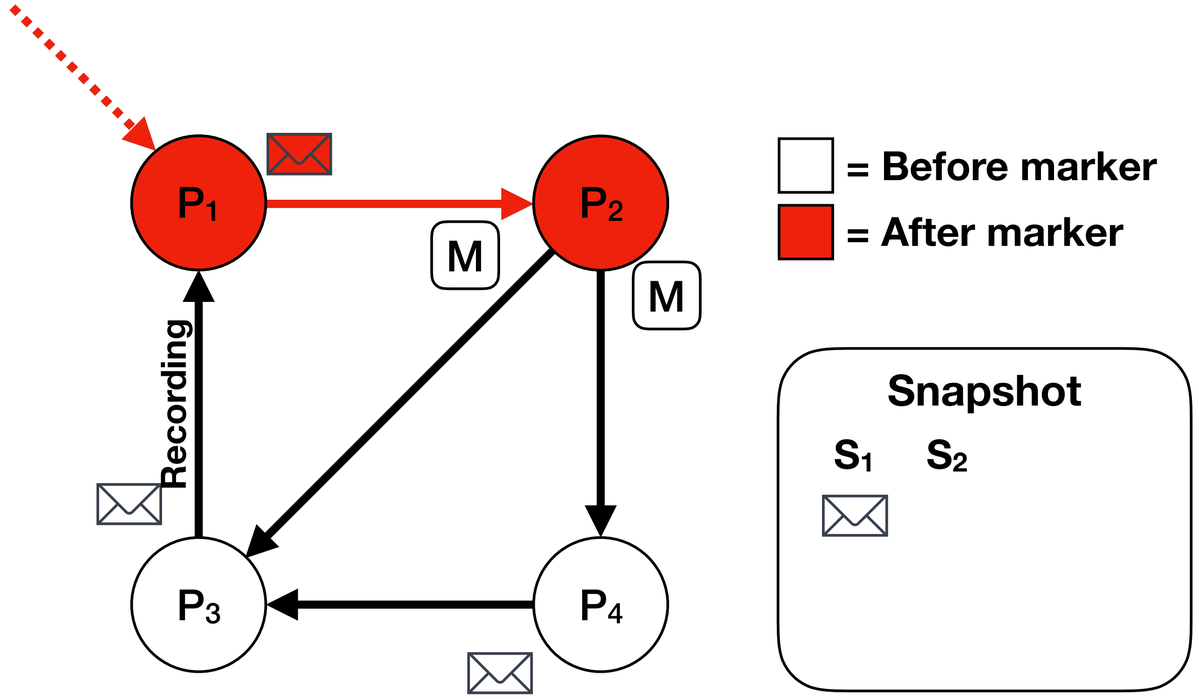


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)

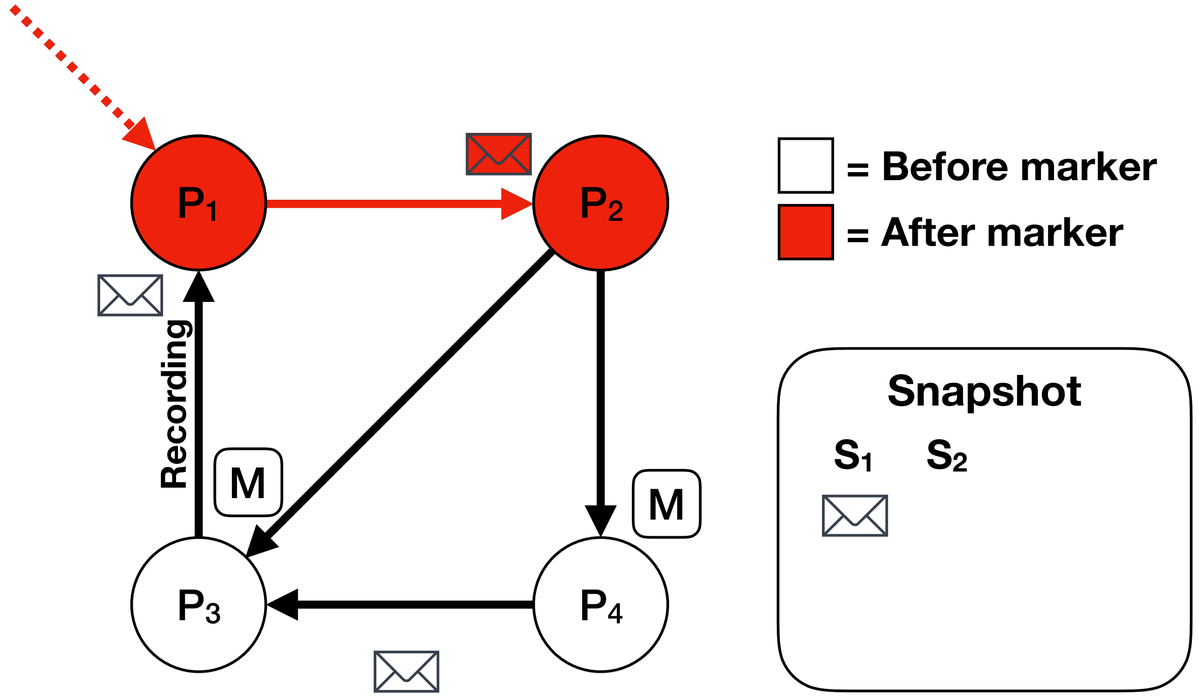
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

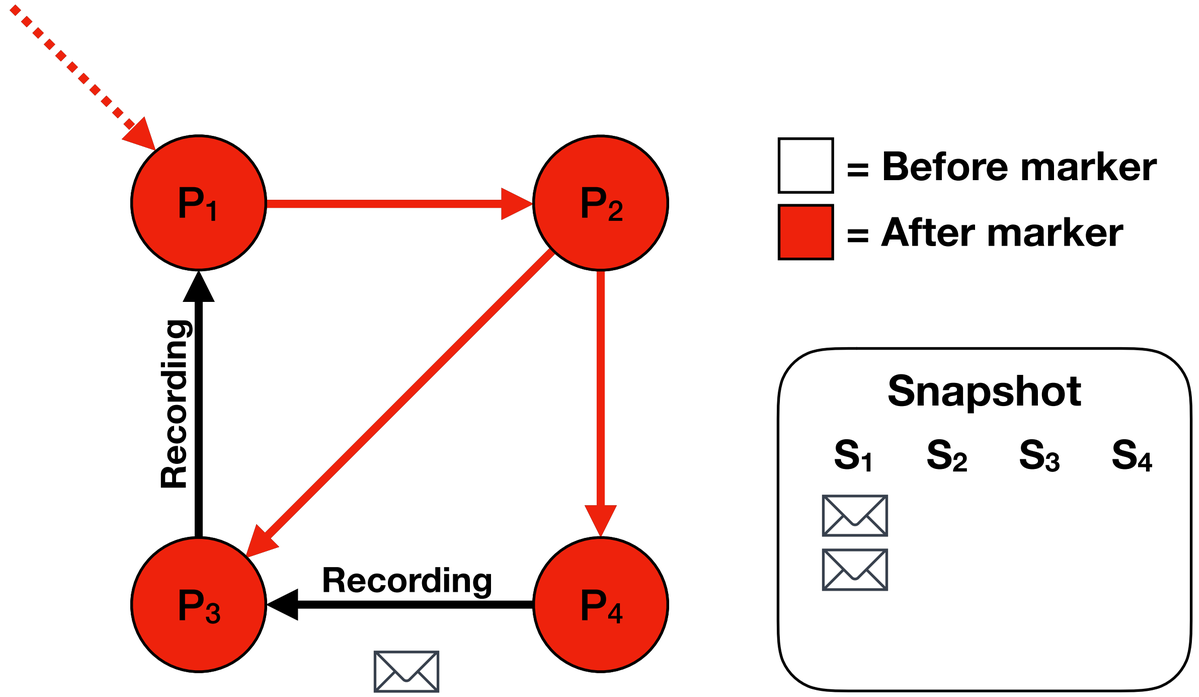
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

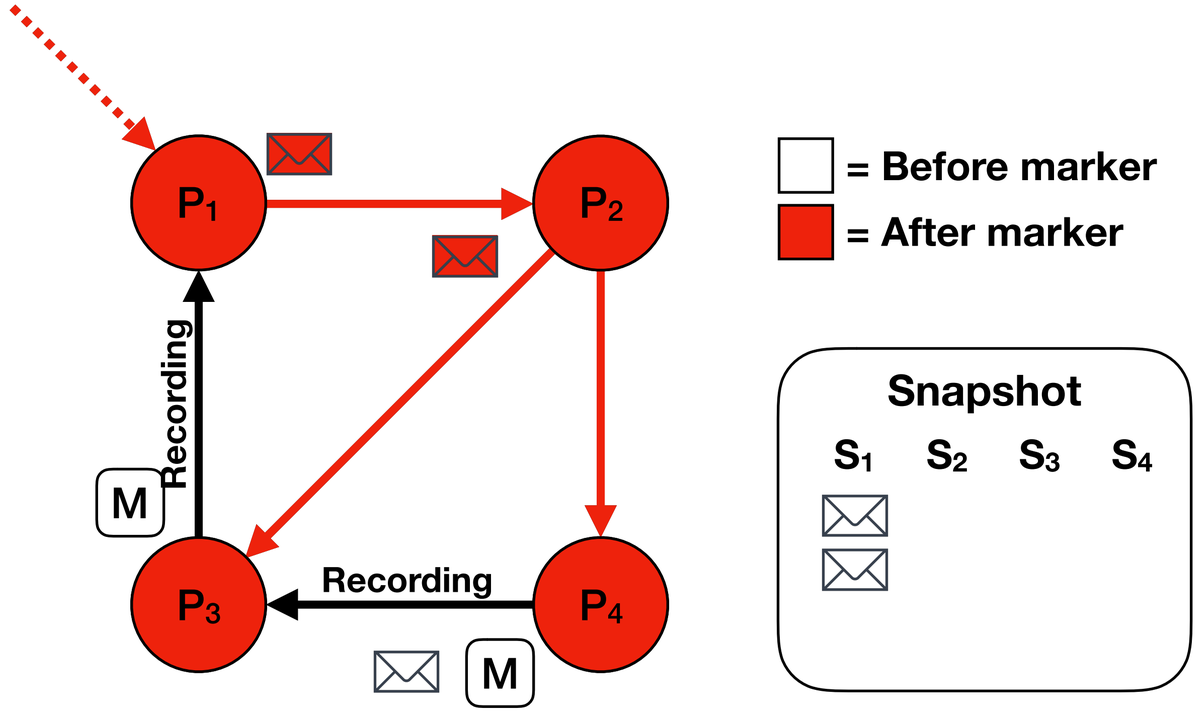
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

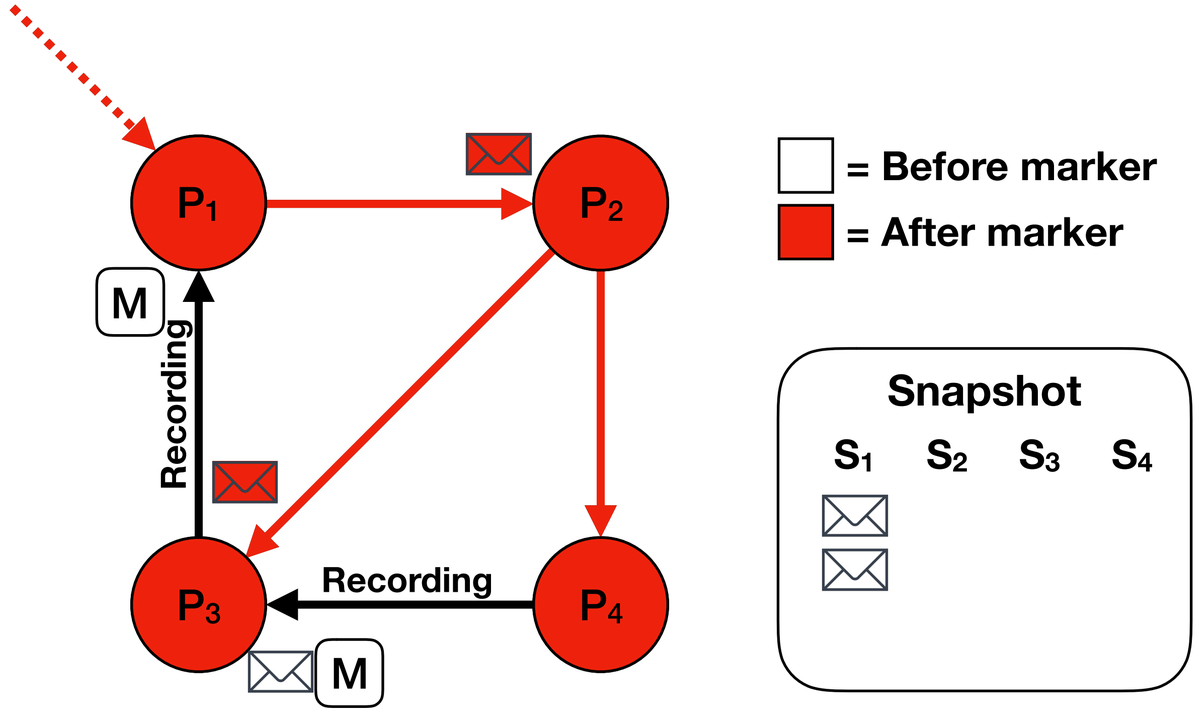
On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded



Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

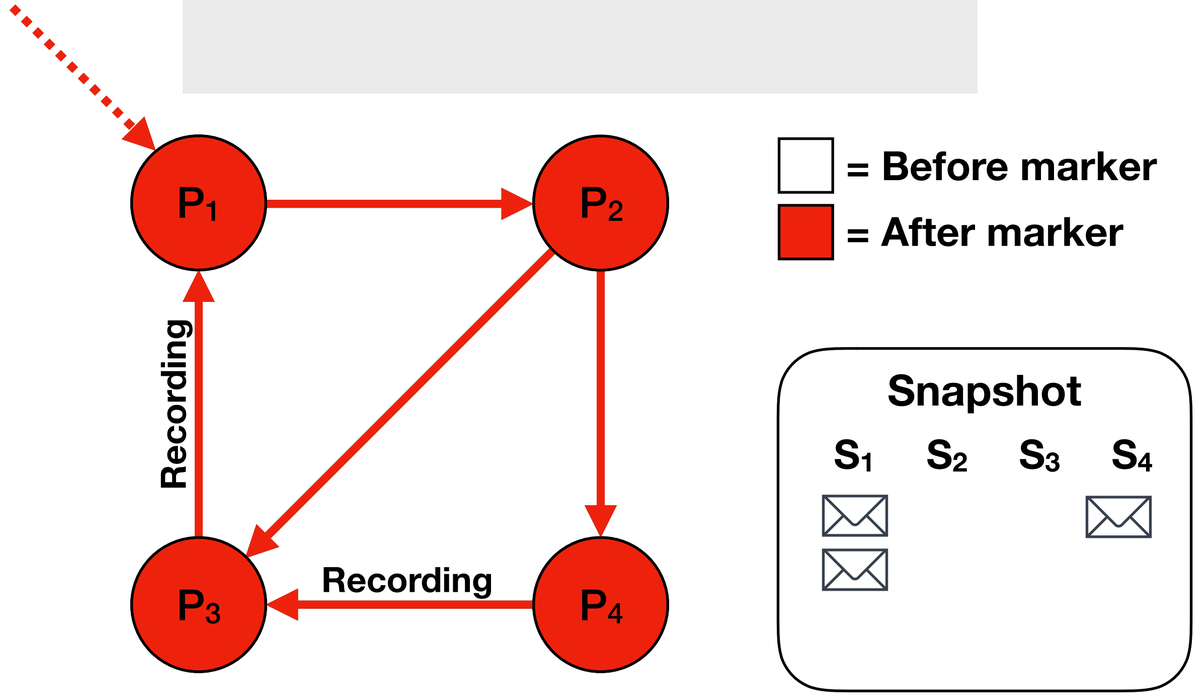


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)

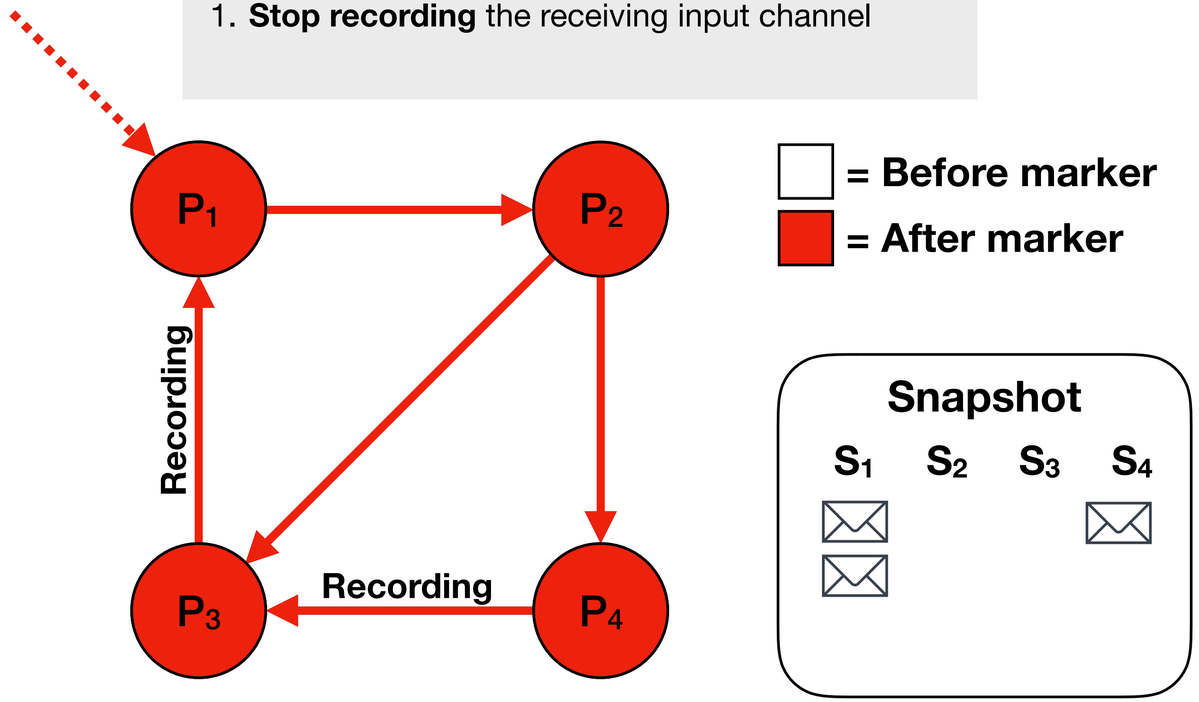


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel

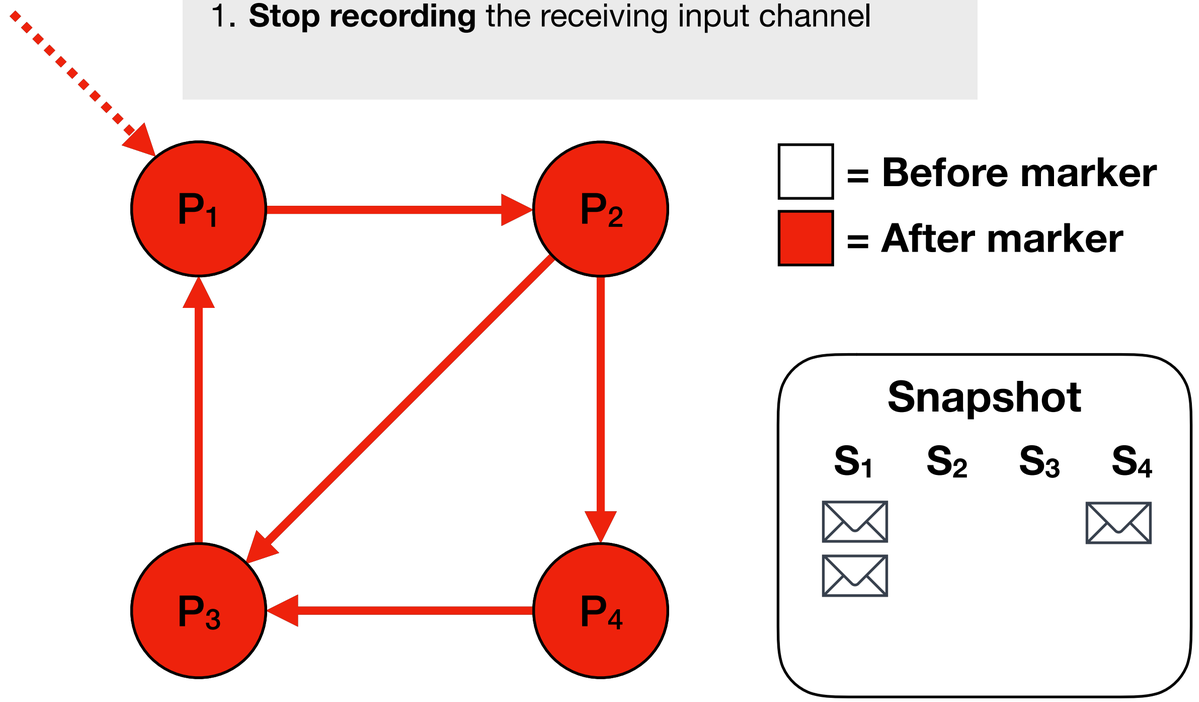


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel

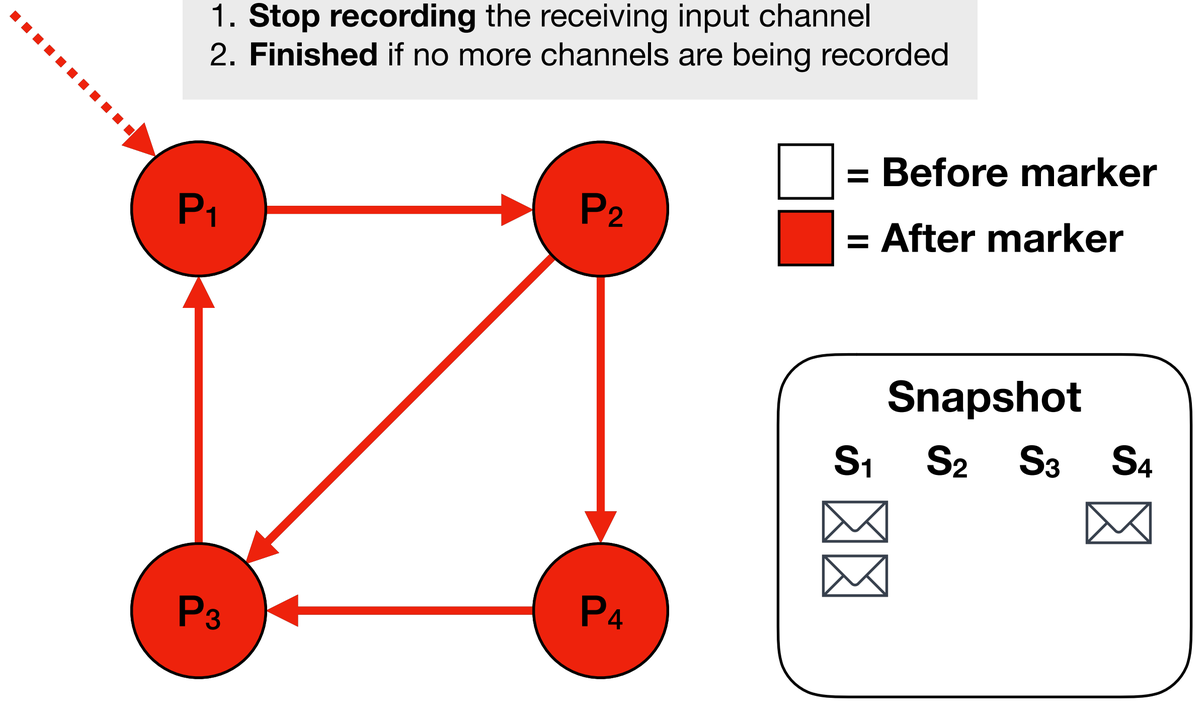


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded

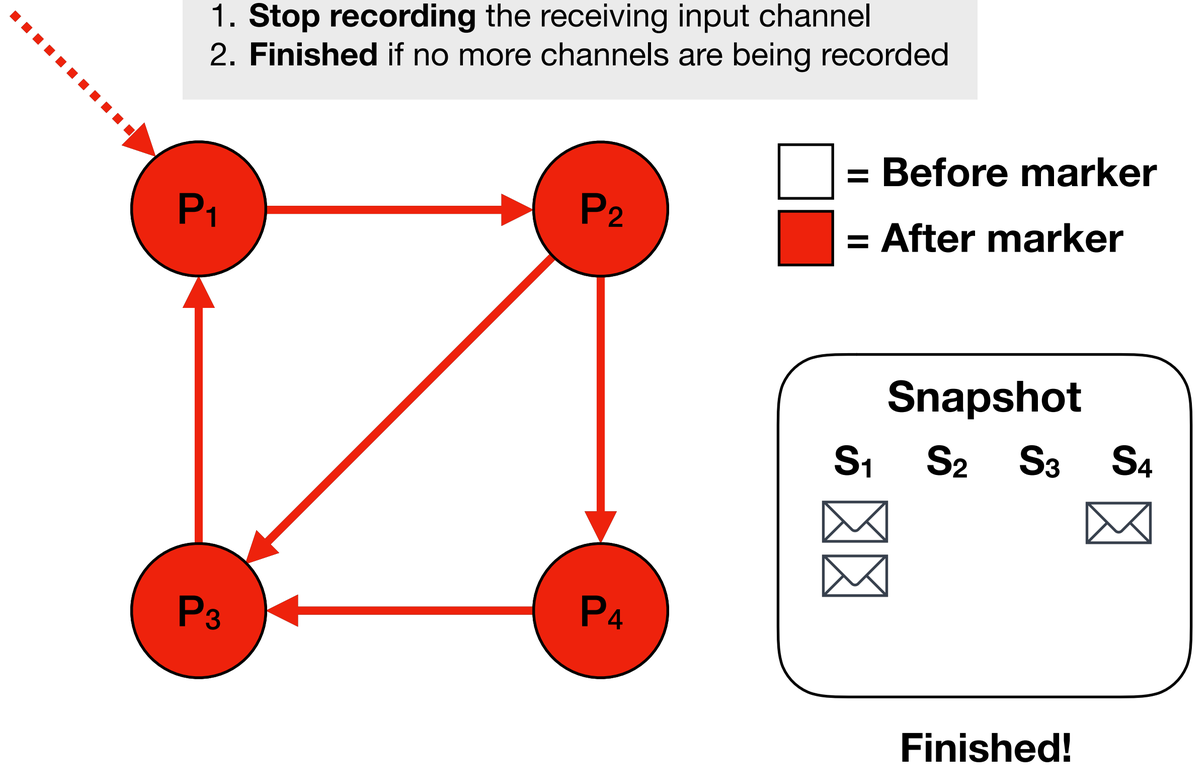


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded

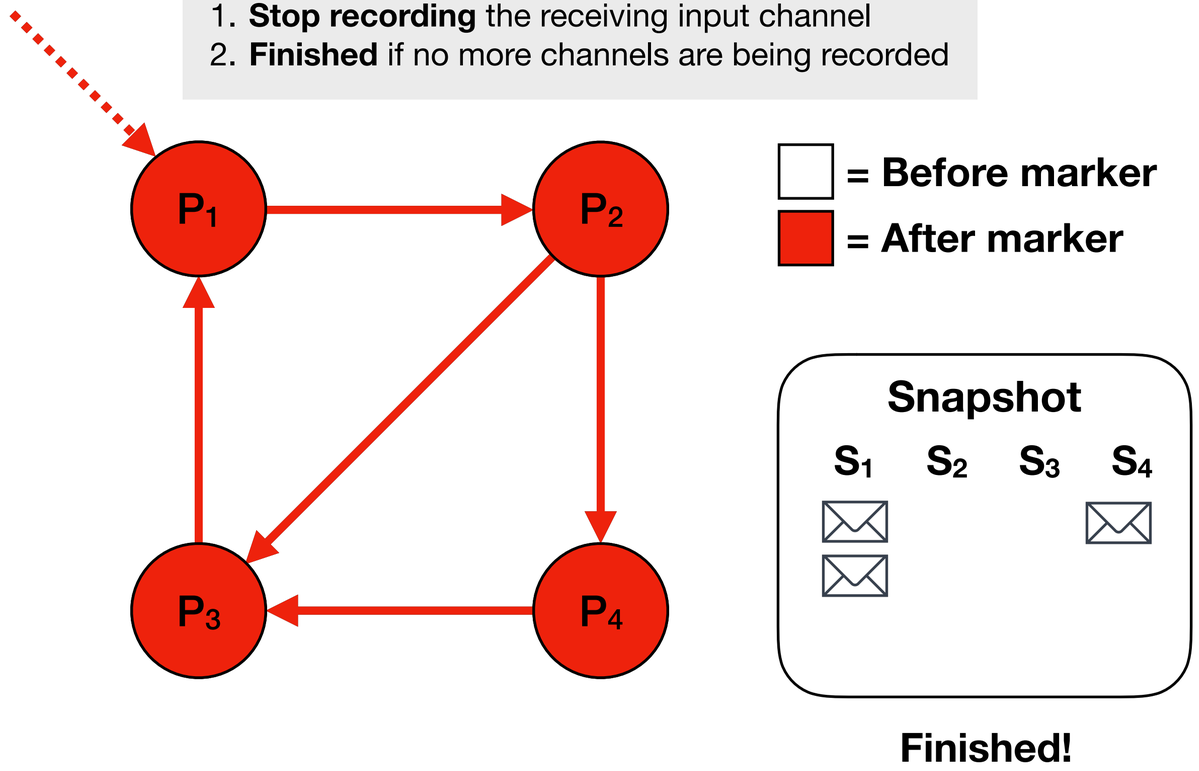


Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded



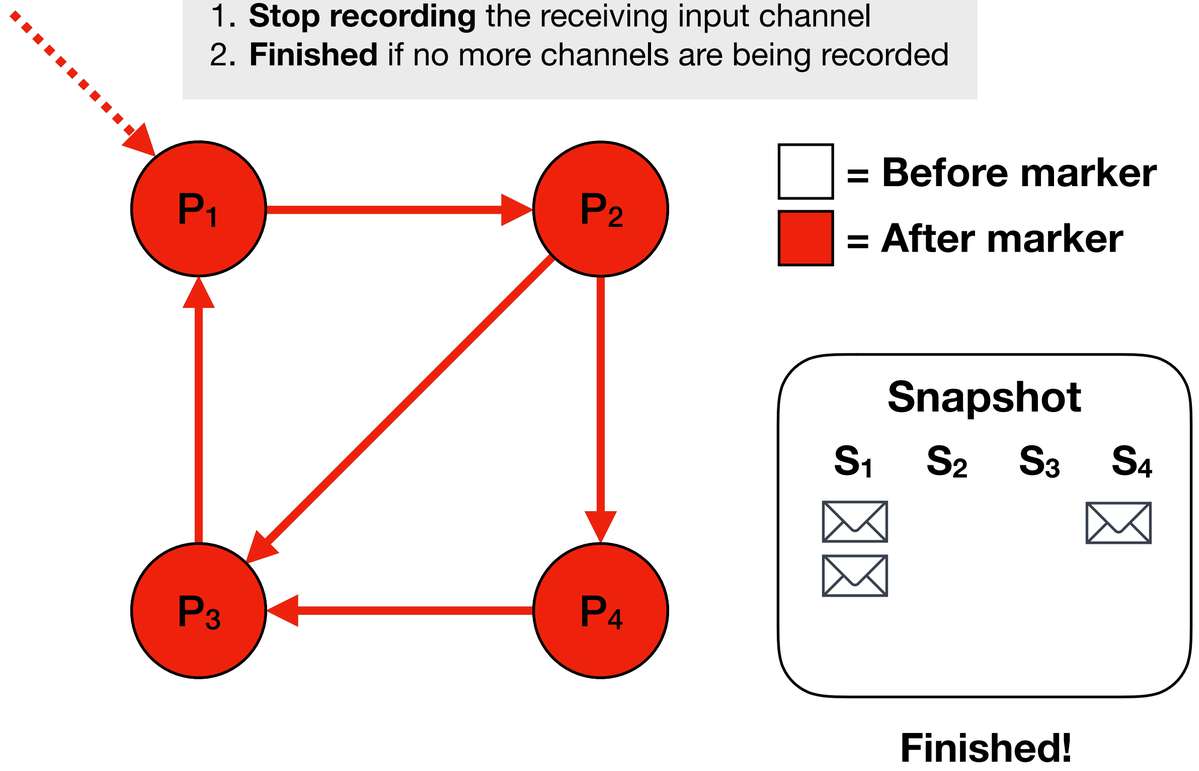
Intuition:
Liveness (Termination)?
Safety (Causal validity)?

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded



Intuition:

Liveness (Termination)? Markers eventually reach **all processes**, from **all inputs**.

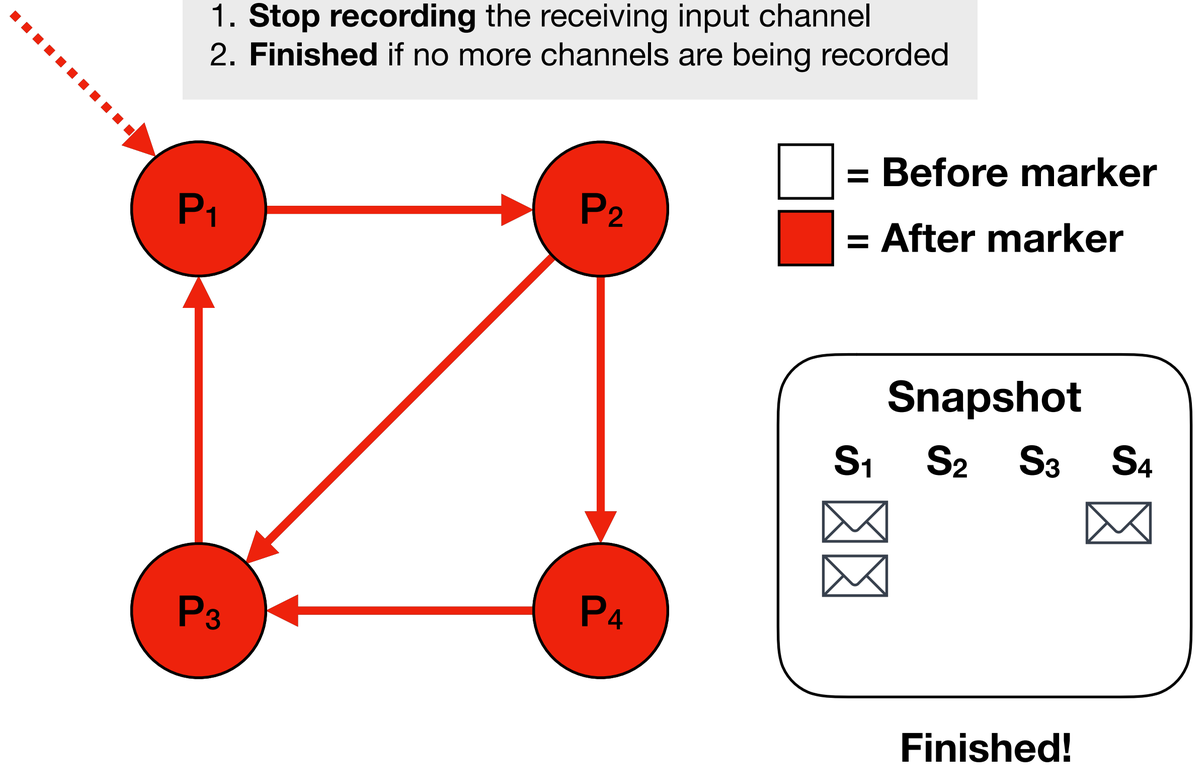
Safety (Causal validity)?

Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

On first marker (atomic)
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

On secondary markers (atomic)
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded



Intuition:

Liveness (Termination)? Markers eventually reach **all processes**, from **all inputs**.

Safety (Causal validity)? State is persisted **before** elements are recorded.

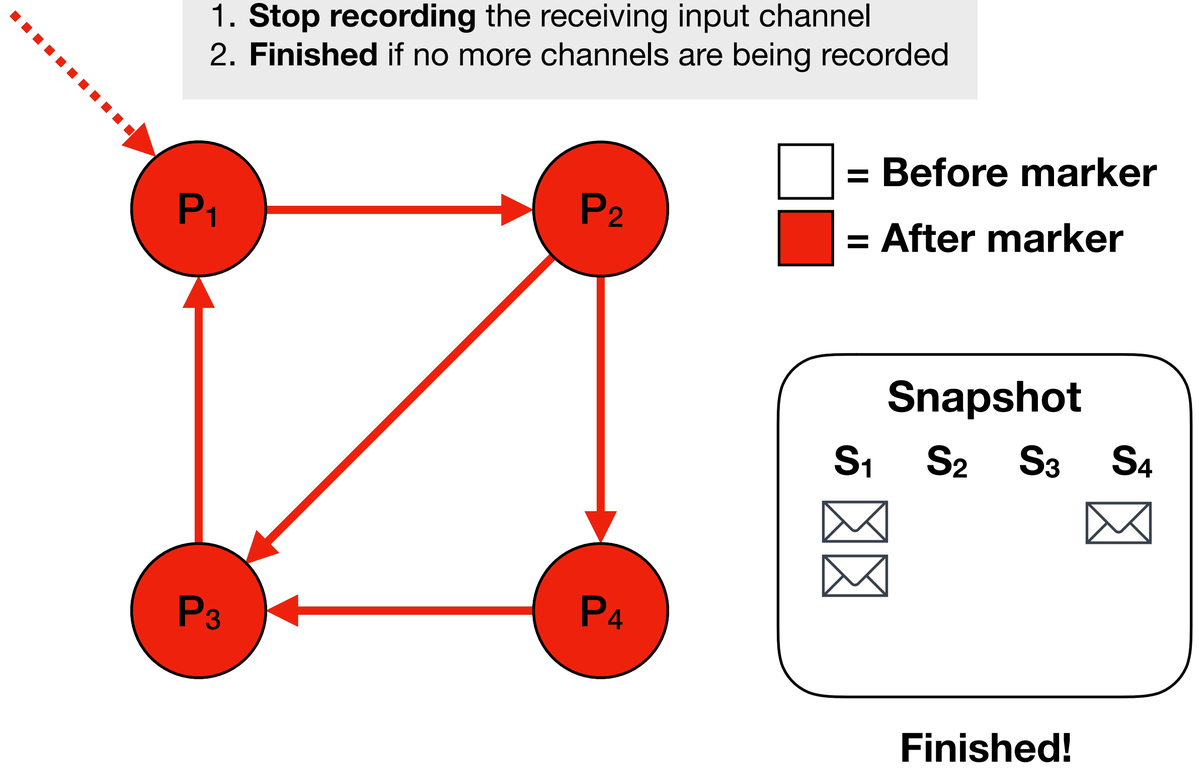
Recap: Chandy-Lamport - Consistent Snapshots

- **Flink's snapshots** are related to the **Chandy-Lamport** protocol
- Both capture the **global state** of a distributed system
- Assumptions
  - Reliable FIFO ordered channels
  - Strongly connected graph
  - Single initiating process

Can the assumptions be modified to fit the stream-processing use-case?

<u>On first marker (atomic)</u>
1. **Capture** internal state
2. **Start recording** other input channels
3. **Forward** marker
4. **Finished** if no more channels are being recorded

<u>On secondary markers (atomic)</u>
1. **Stop recording** the receiving input channel
2. **Finished** if no more channels are being recorded



Intuition:

Liveness (Termination)? Markers eventually reach **all processes**, from **all inputs**.

Safety (Causal validity)? State is persisted **before** elements are recorded.

Flink - Distributed Snapshots

Chandy-Lamport - Assumptions

- Reliable FIFO ordered channels
- Strongly connected graph
- Single initiating process

Apache Flink - Assumptions

- Reliable FIFO ordered channels

Flink - Distributed Snapshots

Chandy-Lamport - Assumptions

- Reliable FIFO ordered channels
- Strongly connected graph
- Single initiating process

Apache Flink - Assumptions

- Reliable FIFO ordered channels
- **Weakly** connected graph

Flink - Distributed Snapshots

Chandy-Lamport - Assumptions

- Reliable FIFO ordered channels
- Strongly connected graph
- Single initiating process

Apache Flink - Assumptions

- Reliable FIFO ordered channels
- **Weakly** connected graph
- Channels can be **blocked**/ **unblocked** by tasks

Flink - Distributed Snapshots

Chandy-Lamport - Assumptions

- Reliable FIFO ordered channels
- Strongly connected graph
- Single initiating process

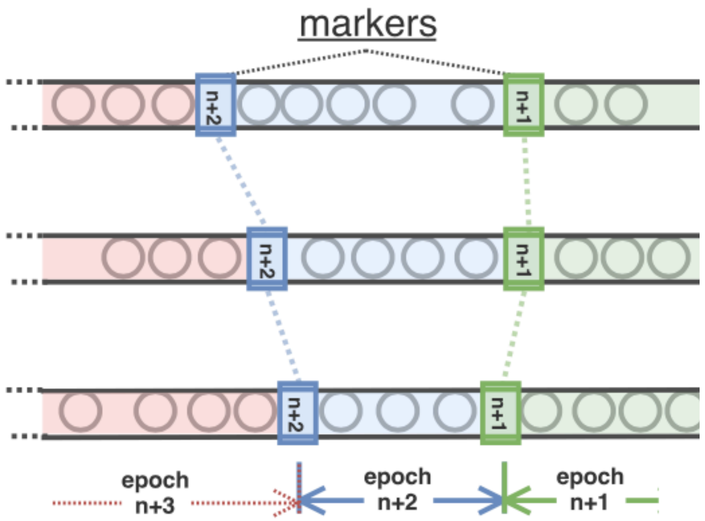
Apache Flink - Assumptions

- Reliable FIFO ordered channels
- **Weakly** connected graph
- Channels can be **blocked**/ **unblocked** by tasks
- Sources are **replayable**

Flink - Distributed Snapshots

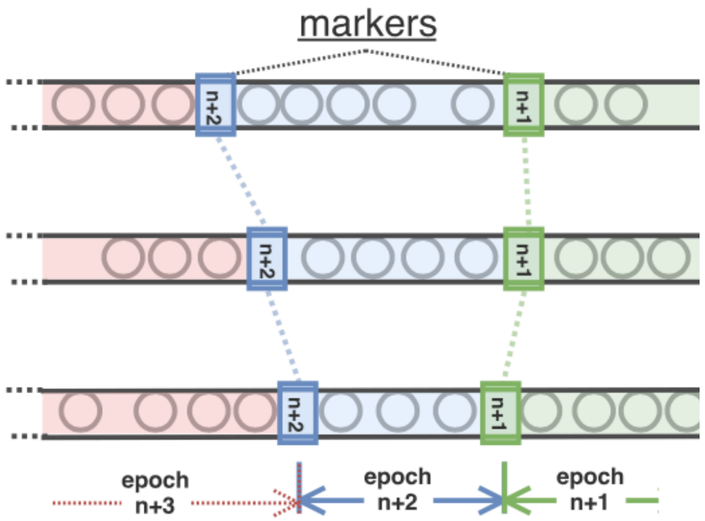
Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.



Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

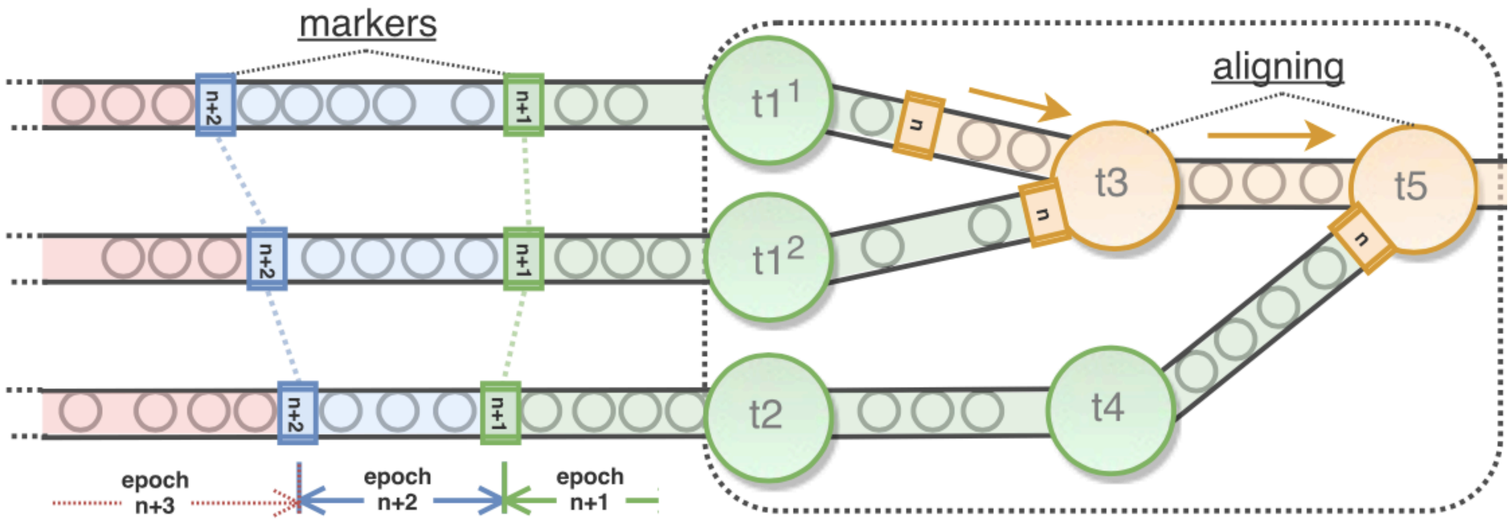


Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.



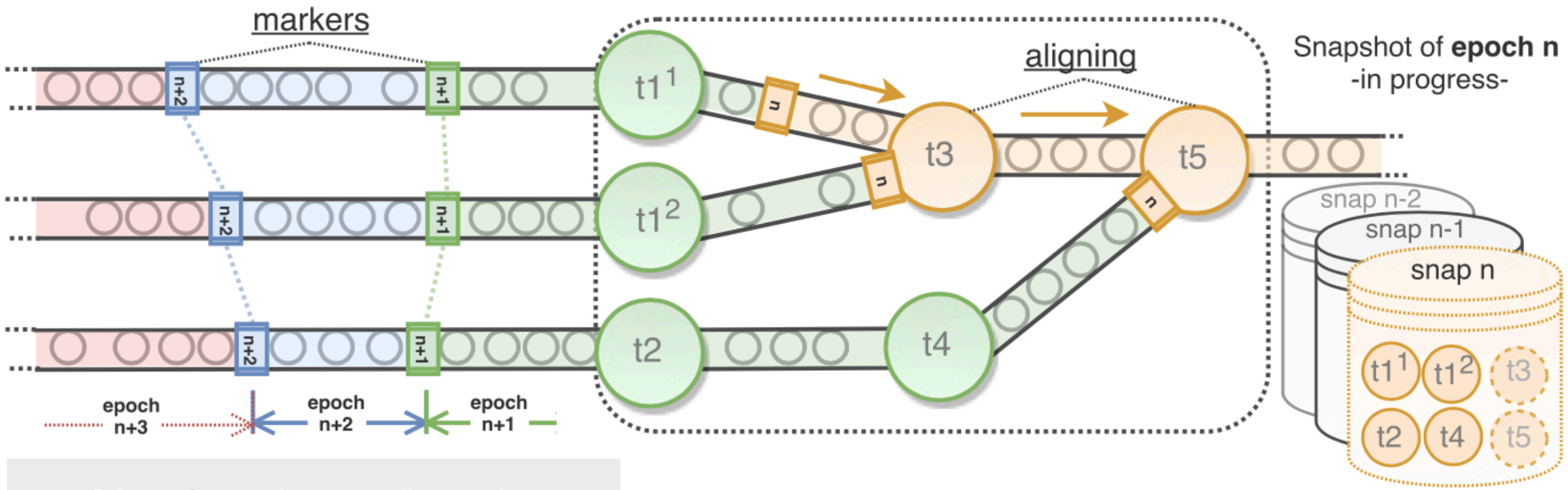
Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.

3. Commit
Once an operator has received markers on all channels, it **forwards** the marker, **persists** its local state**,** and **unblocks**.



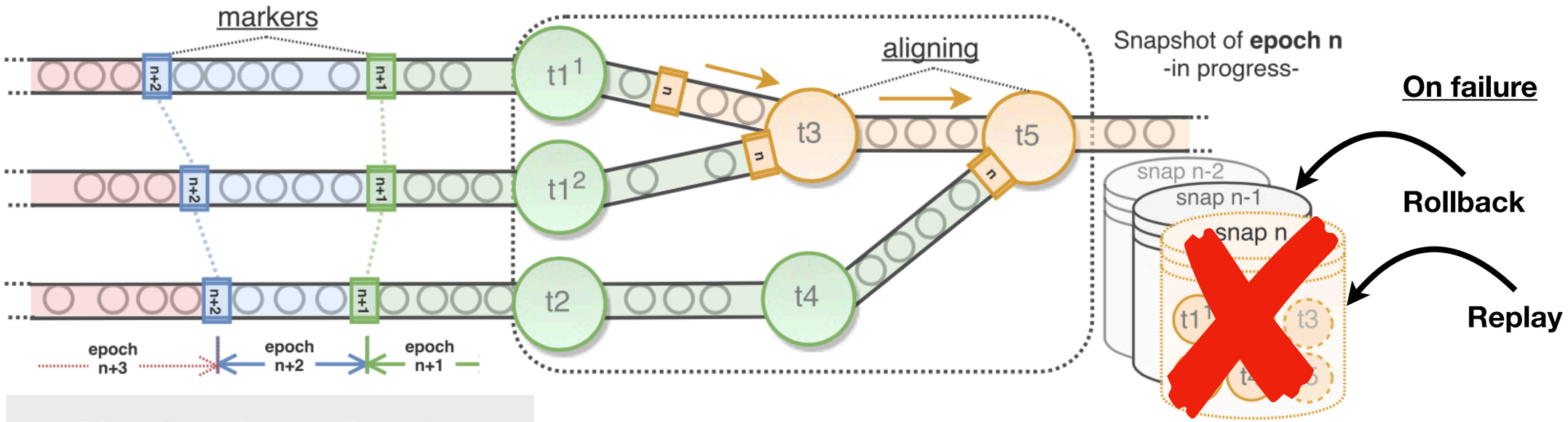
Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.

3. Commit
Once an operator has received markers on all channels, it **forwards** the marker, **persists** its local state**,** and **unblocks**.



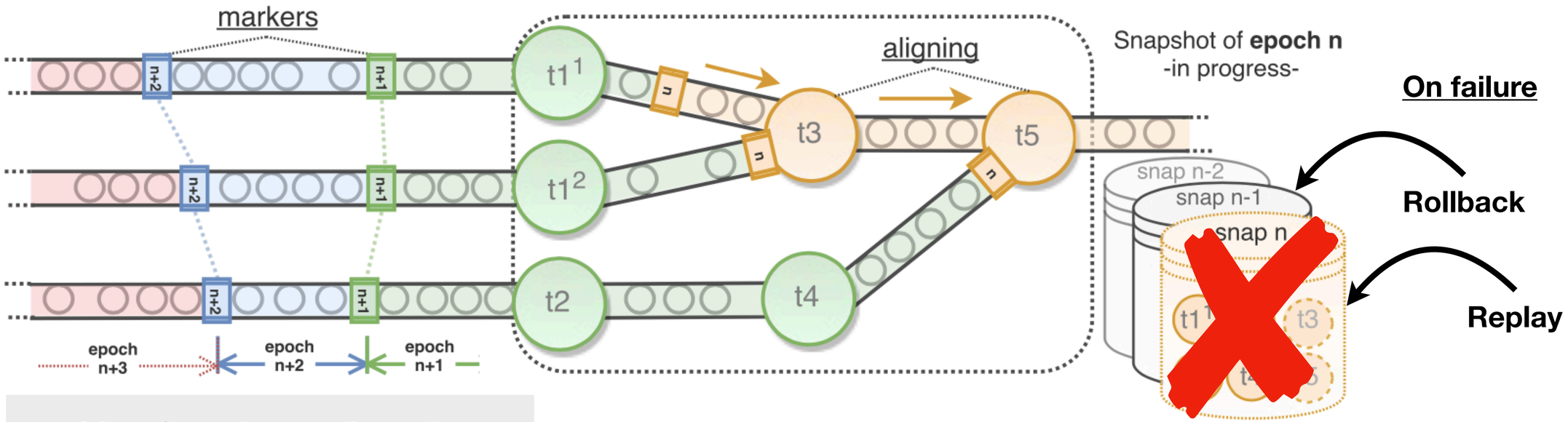
Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.

3. Commit
Once an operator has received markers on all channels, it **forwards** the marker, **persists** its local state**,** and **unblocks**.



Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

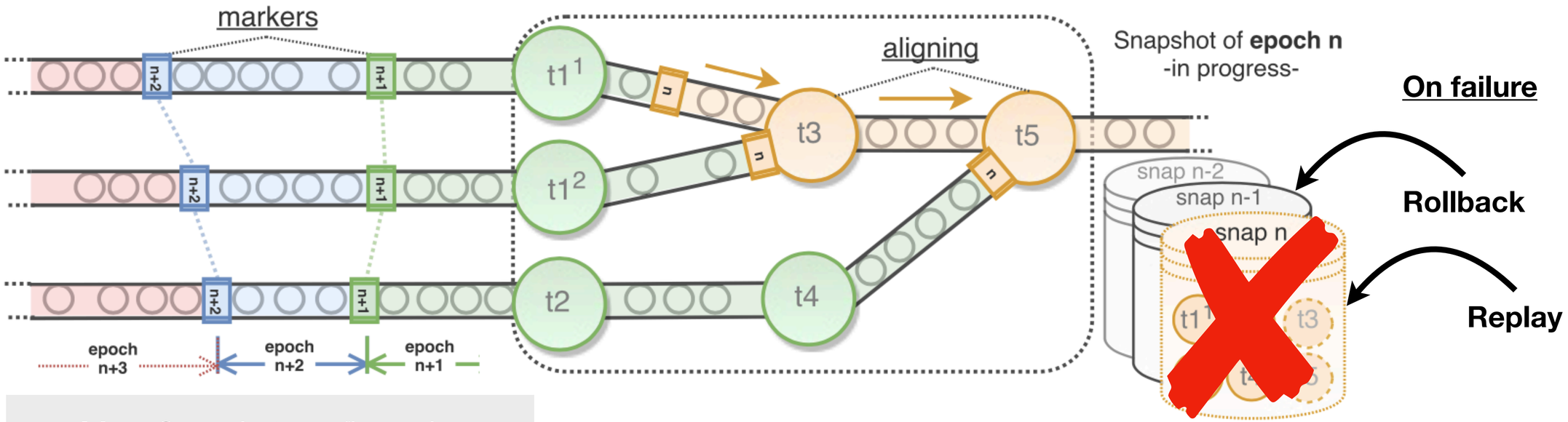
Optimizations:
- **Incremental snapshots**
- **Asynchronous compactions**

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.

3. Commit
Once an operator has received markers on all channels, it **forwards** the marker, **persists** its local state**,** and **unblocks**.



Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

Optimizations:
- **Incremental snapshots**
- **Asynchronous compactions**

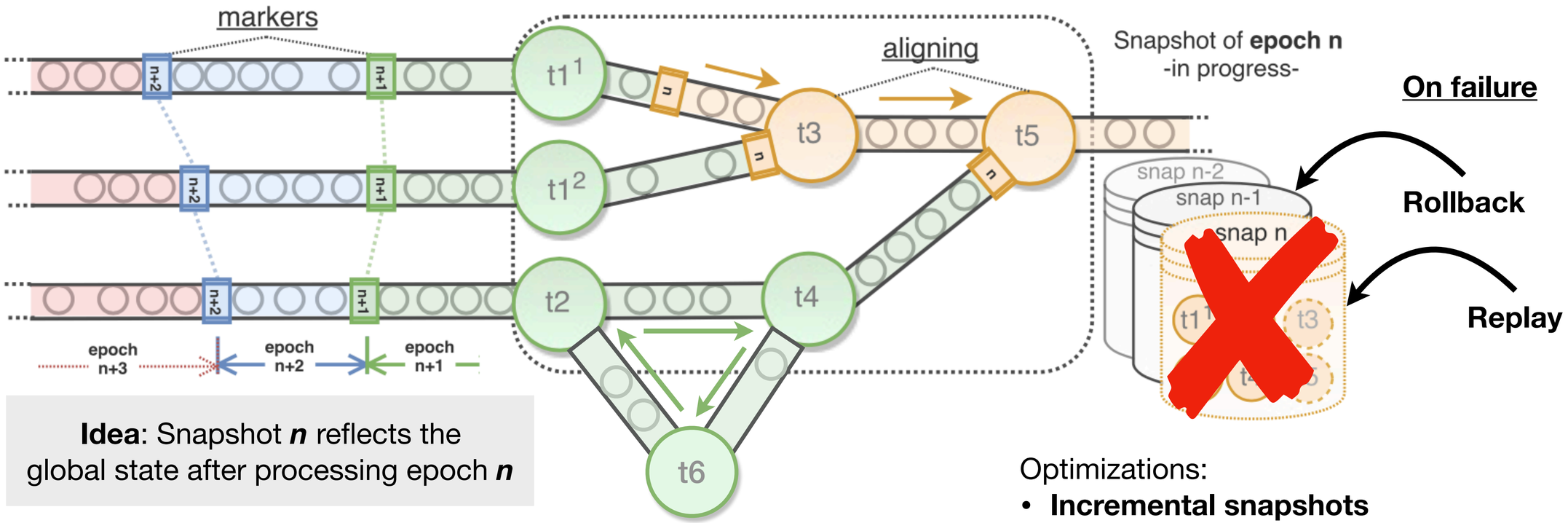
Exactly-once delivery:
- **Idempotent sinks**
- **Transactional sinks**

Flink - Distributed Snapshots

1. Inject markers
Begin snapshots by injecting **markers** at the **sources,** that cut the stream into **epochs**.

2. Alignment
When an operator receives a marker, the receiving channel is **blocked**.

3. Commit
Once an operator has received markers on all channels, it **forwards** the marker, **persists** its local state**,** and **unblocks**.



Idea: Snapshot ***n*** reflects the global state after processing epoch ***n***

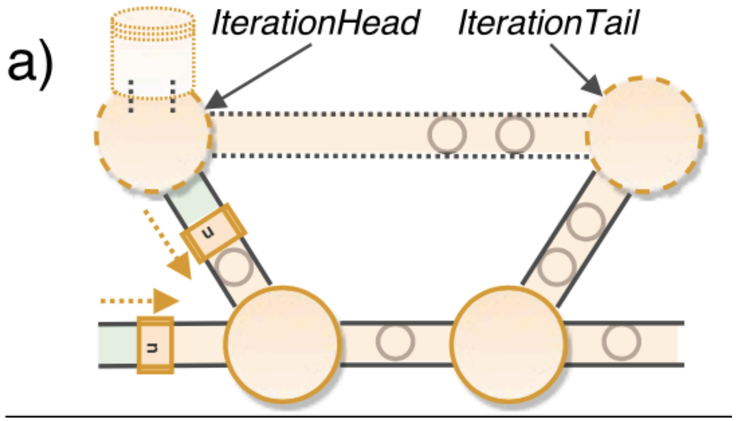
What about cyclic graphs?

Optimizations:
- **Incremental snapshots**
- **Asynchronous compactions**

Exactly-once delivery:
- **Idempotent sinks**
- **Transactional sinks**

Flink - Distributed Snapshots

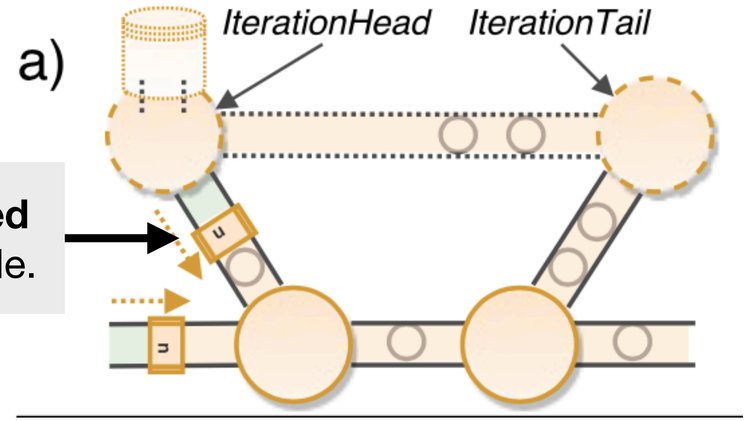
1. When cycles are present,
Flink falls back to **Chandy Lamport**



Flink - Distributed Snapshots

1. When cycles are present,
Flink falls back to **Chandy Lamport**

2. Markers are **injected** at the **head** of the cycle.

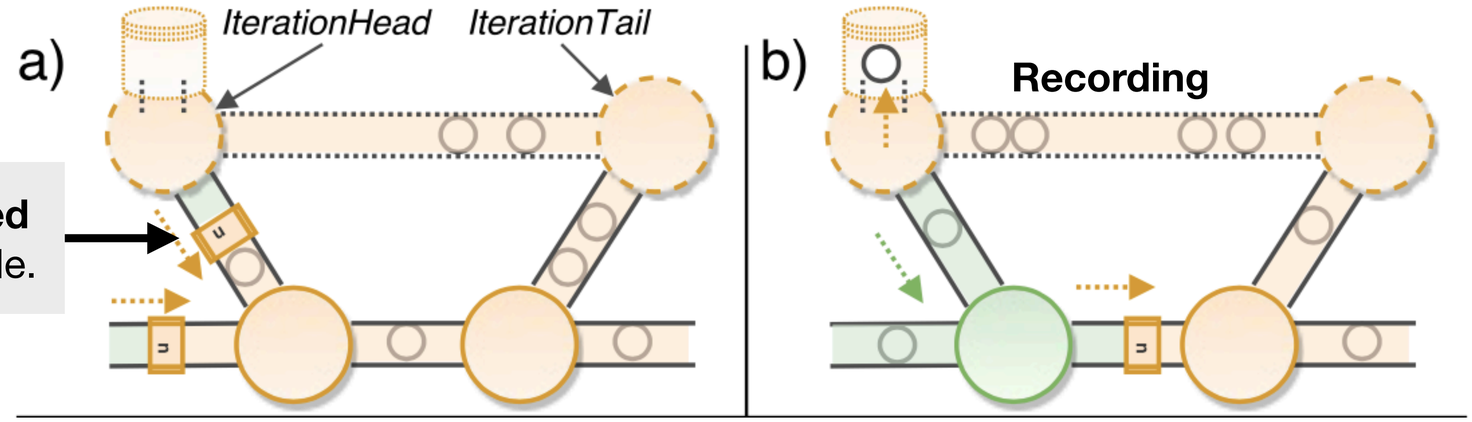


Flink - Distributed Snapshots

1. When cycles are present, Flink falls back to **Chandy Lamport**

3. Elements from the **tail** of the cycle are **recorded**.

2. Markers are **injected** at the **head** of the cycle.

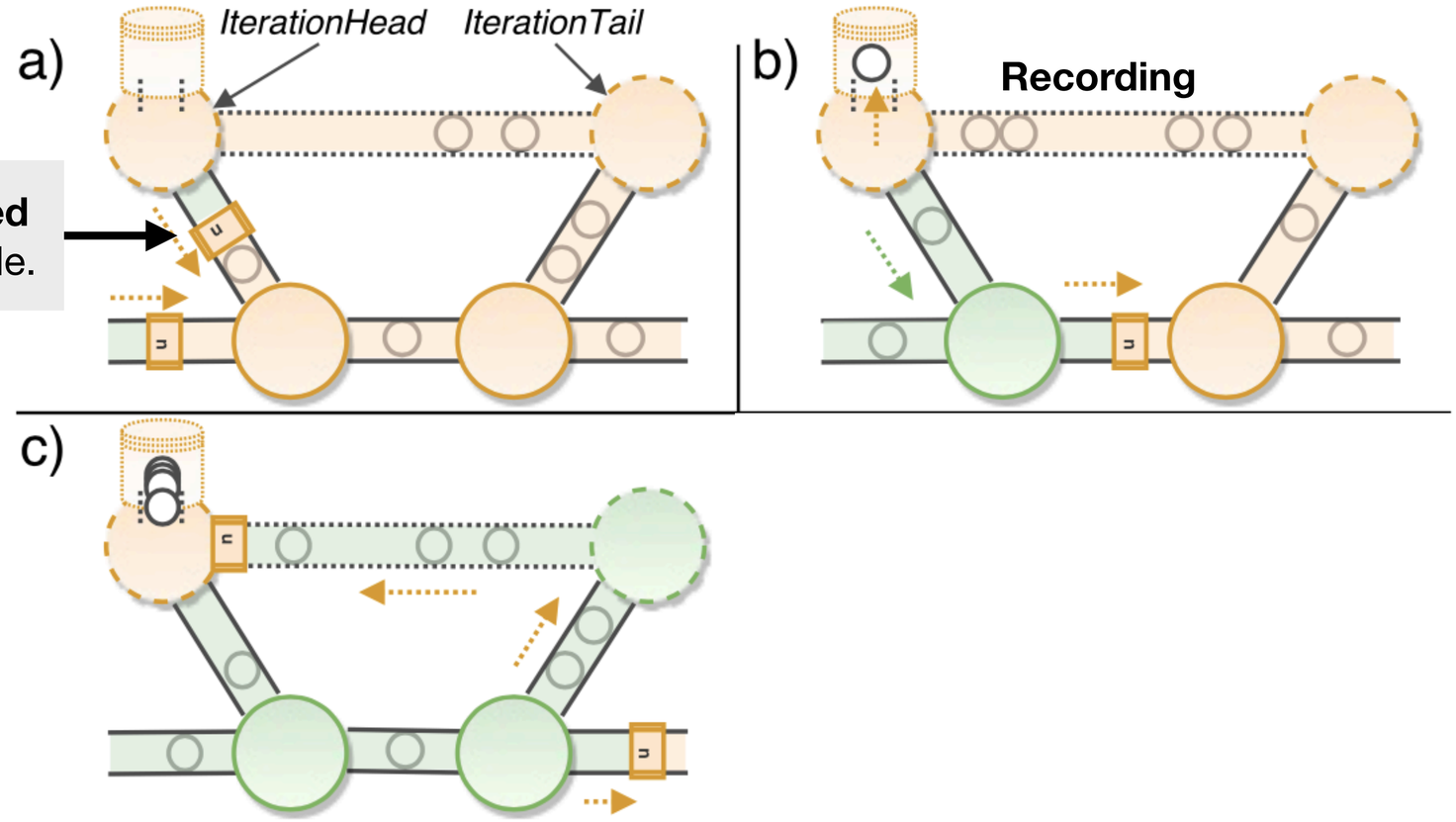


Flink - Distributed Snapshots

1. When cycles are present, Flink falls back to **Chandy Lamport**

3. Elements from the **tail** of the cycle are **recorded**.

2. Markers are **injected** at the **head** of the cycle.

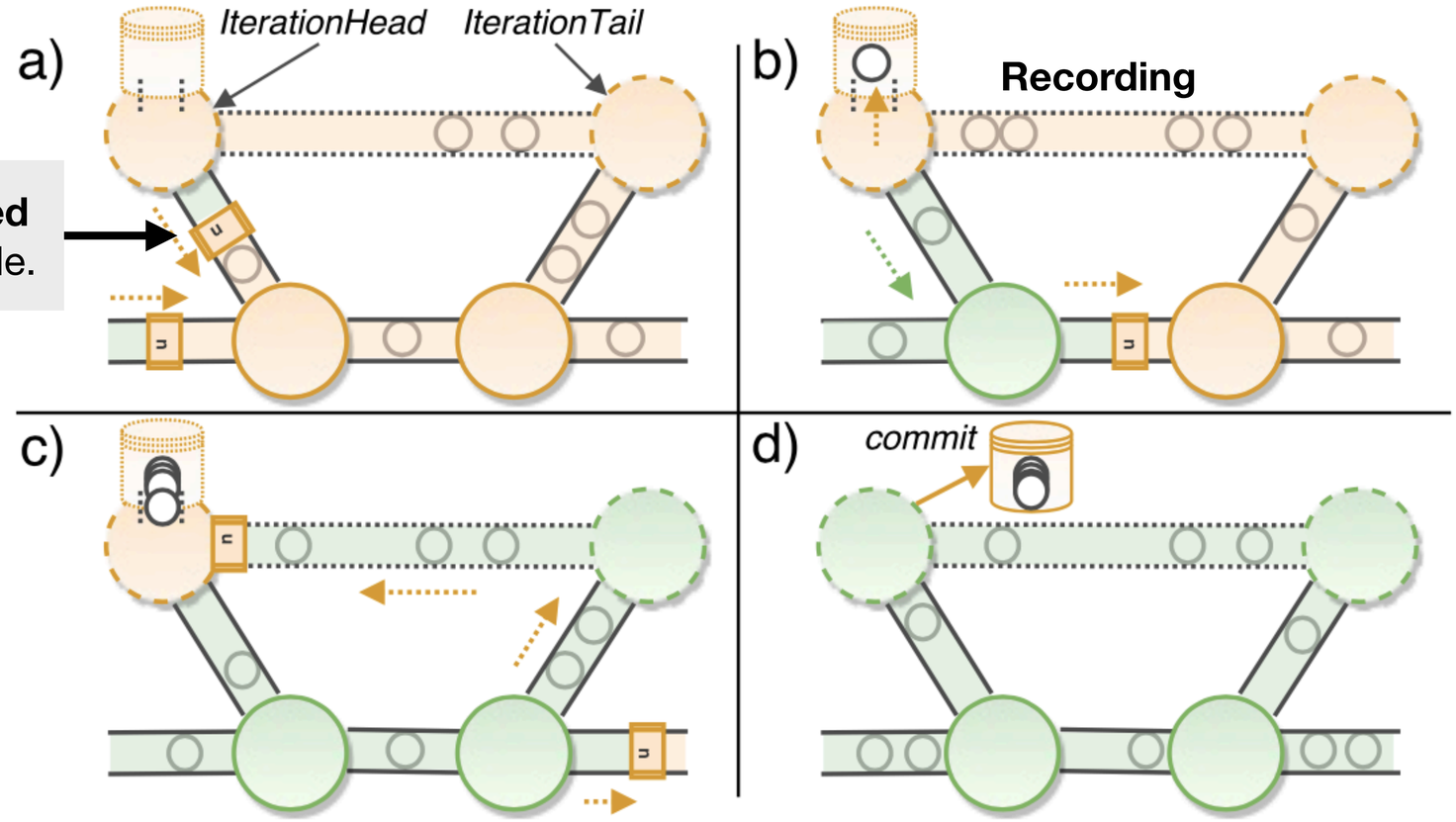


Flink - Distributed Snapshots

1. When cycles are present, Flink falls back to **Chandy Lamport**

3. Elements from the **tail** of the cycle are **recorded**.

2. Markers are **injected** at the **head** of the cycle.



Flink - Discussion

Flink - Discussion

Q: What affects **snapshotting latency**?

Flink - Discussion

Q: What affects **snapshotting latency**?

A: Alignment introduces latency. Global state size is not a factor.

Flink - Discussion

Q: What affects **snapshotting latency**?

A: Alignment introduces latency. Global state size is not a factor.

Q: What affects the **alignment time**?

Flink - Discussion

Q: What affects **snapshotting latency**?

A: Alignment introduces latency. Global state size is not a factor.

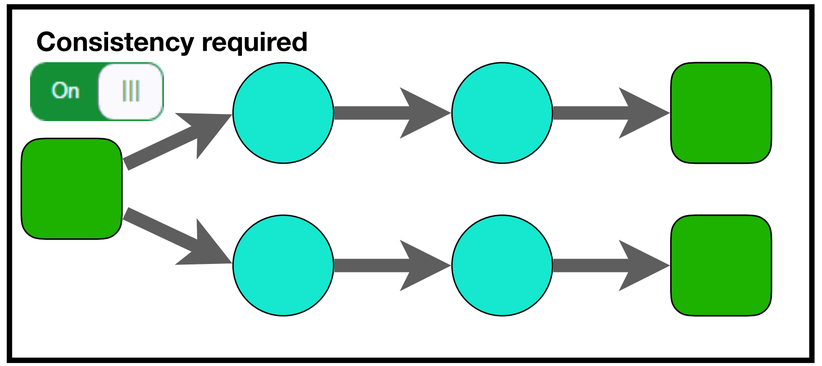
Q: What affects the **alignment time**?

A: The **number of input channels** (determined by shuffles & data parallelism).

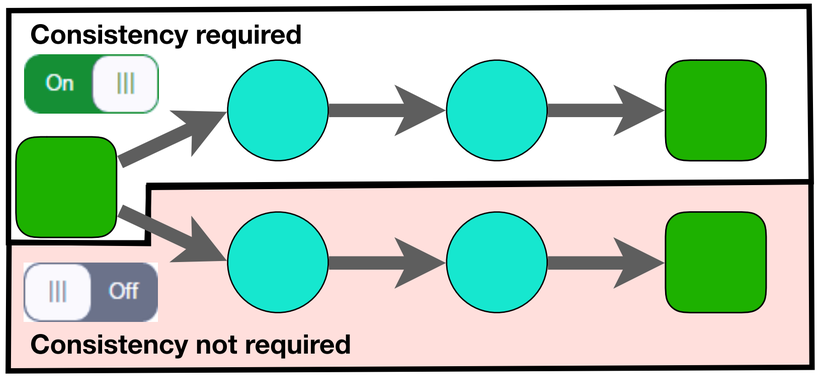
IBM-Streams

IBM-Streams - Problem statement

Apache Flink



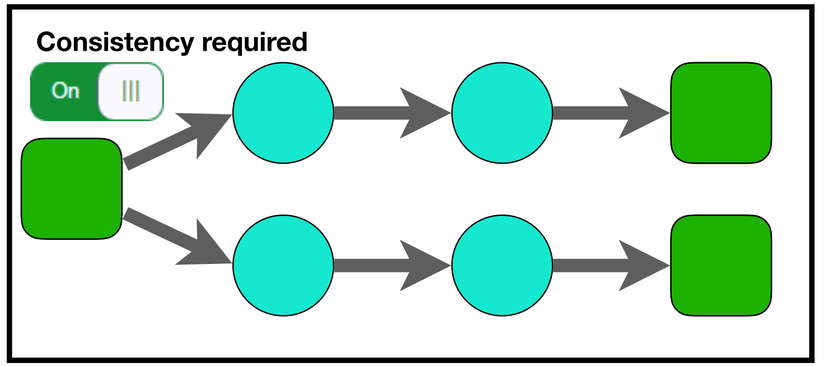
IBM Streams



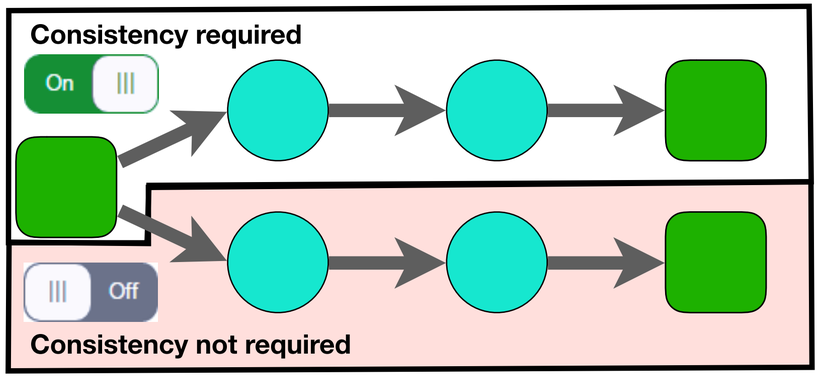
IBM-Streams - Problem statement

- Snapshots impose overhead

Apache Flink



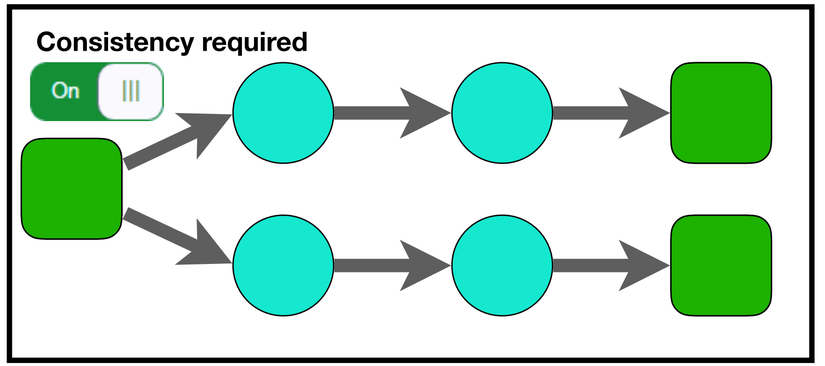
IBM Streams



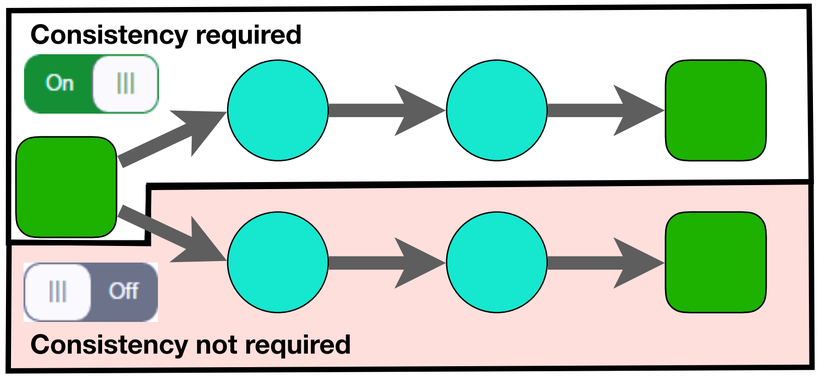
IBM-Streams - Problem statement

- Snapshots impose overhead
- What if processing guarantees are only necessary for some operators?

Apache Flink



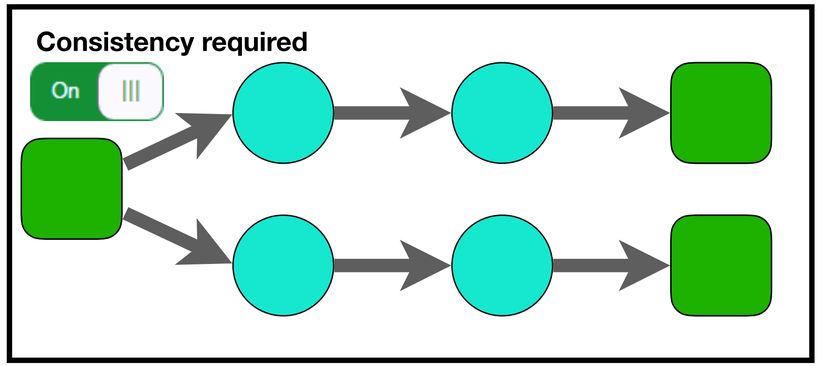
IBM Streams



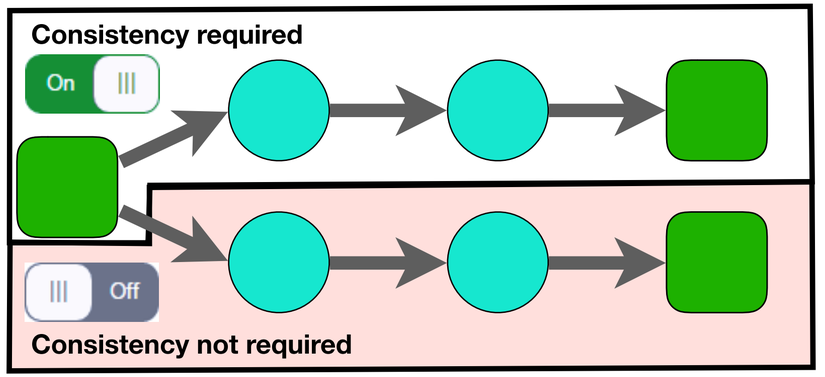
IBM-Streams - Problem statement

- Snapshots impose overhead
- What if processing guarantees are only necessary for some operators?
- **Solution?** Create a stream processing language for more fine-grained control

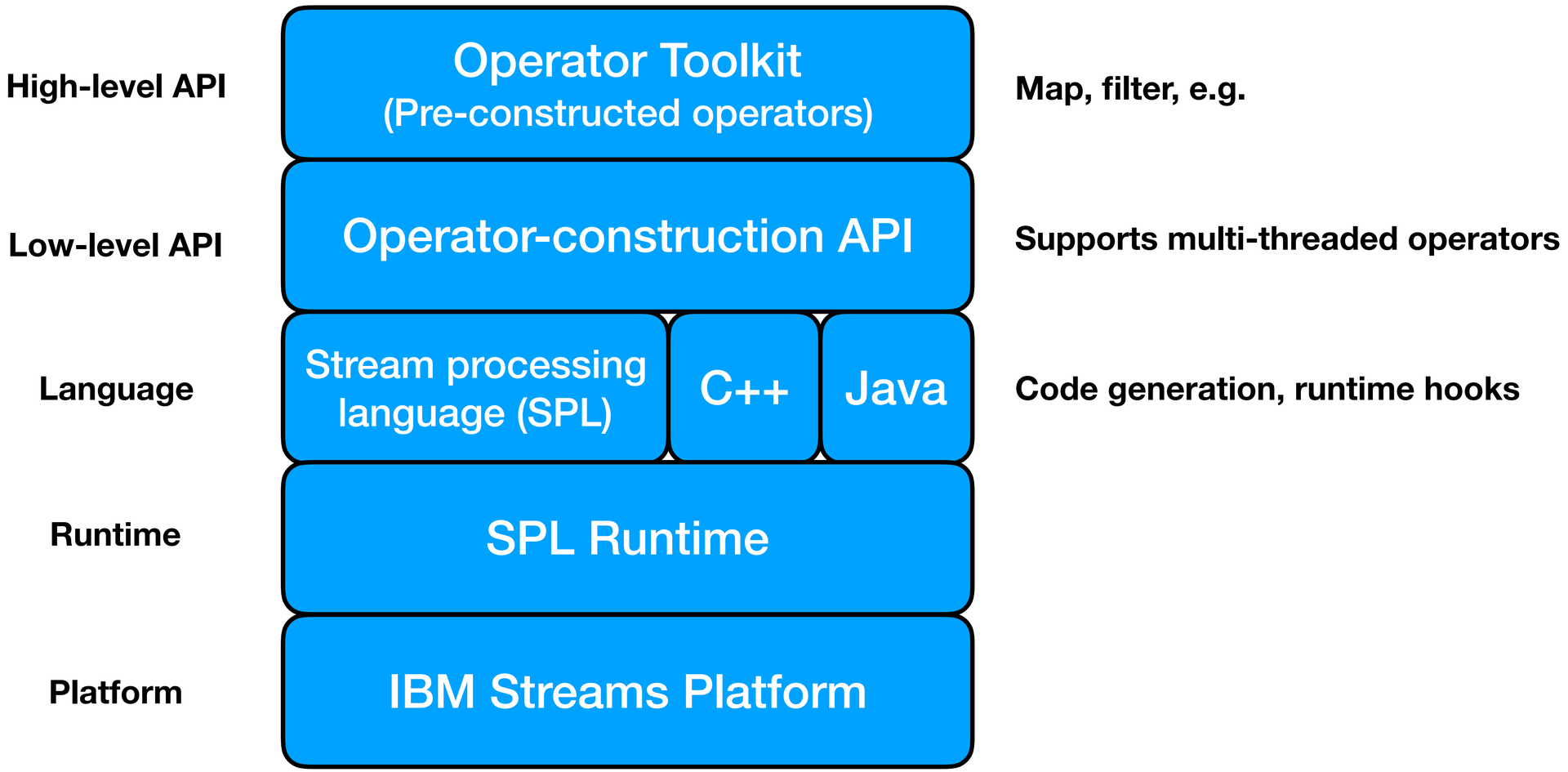
Apache Flink



IBM Streams



IBM Streams - System Overview



IBM Streams - Consistent Regions

IBM Streams - Consistent Regions

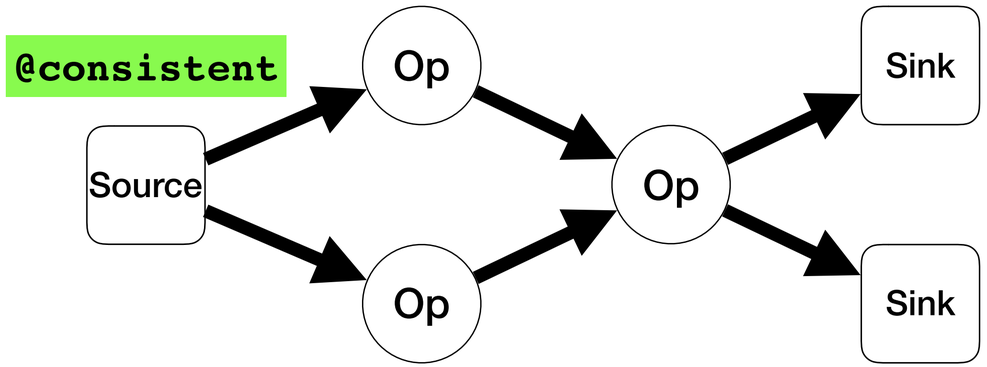
Snapshots only occur inside **consistent regions**

IBM Streams - Consistent Regions

Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region

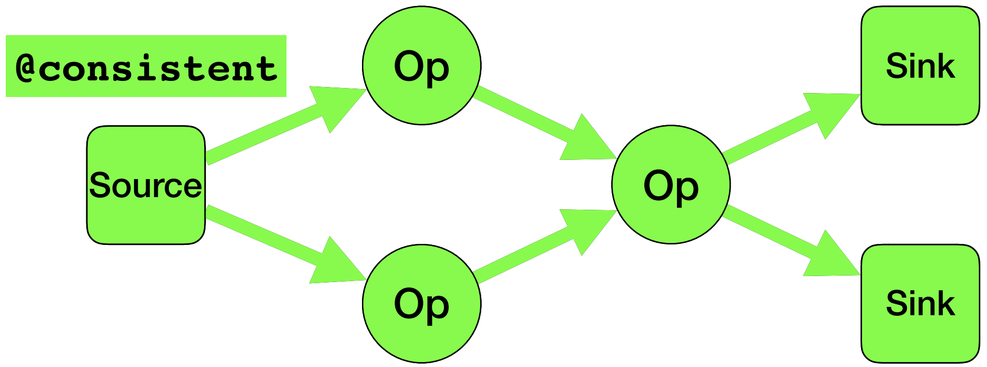
IBM Streams - Consistent Regions

Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region



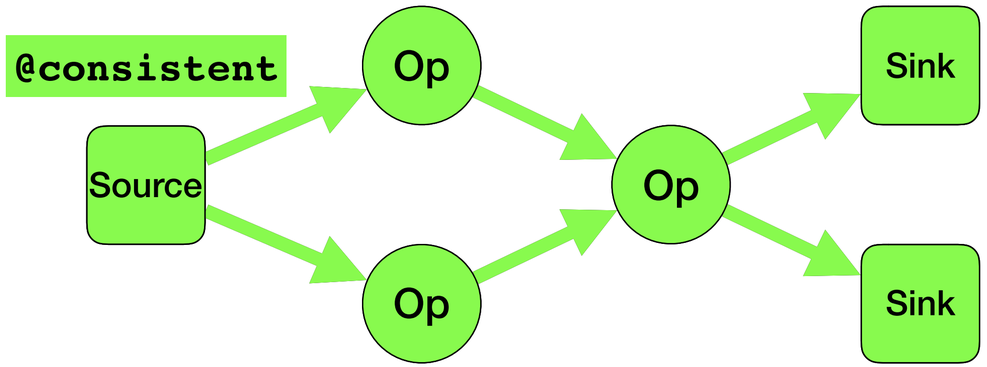
IBM Streams - Consistent Regions

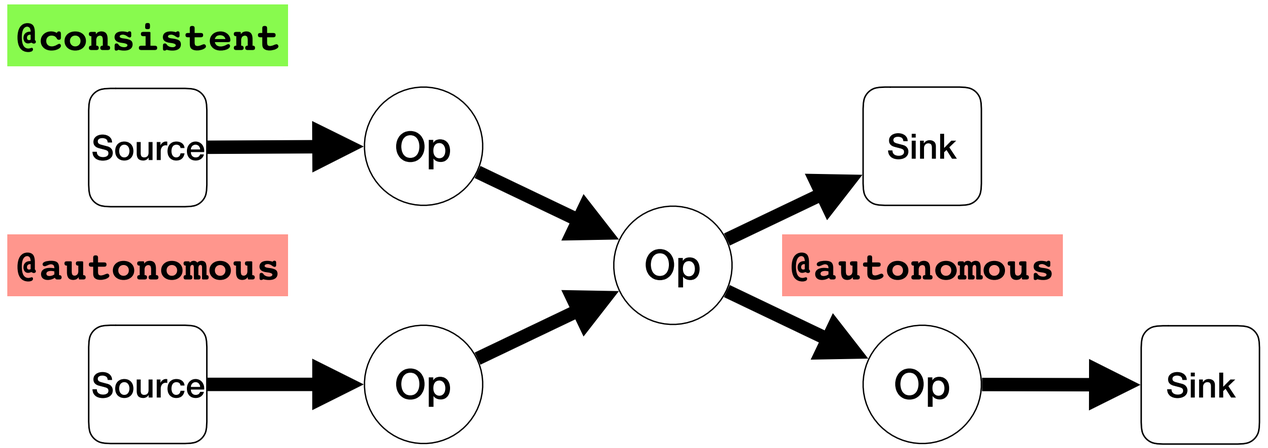
Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region



IBM Streams - Consistent Regions

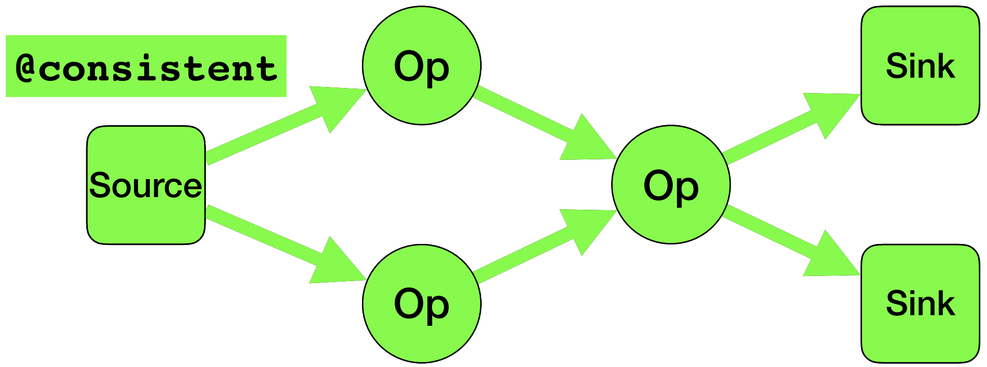
Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region

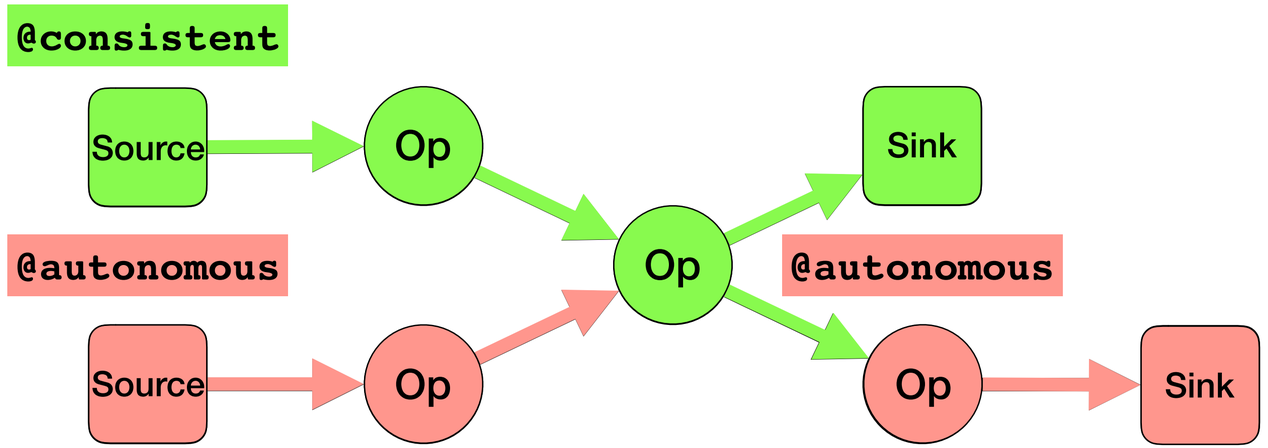




IBM Streams - Consistent Regions

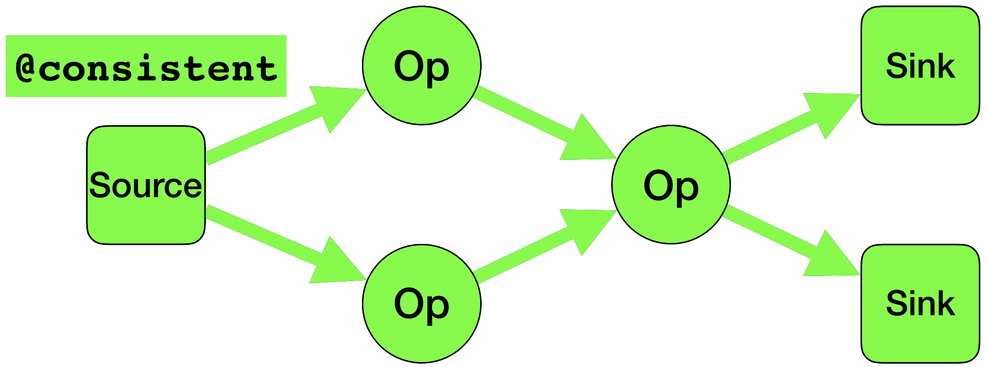
Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region



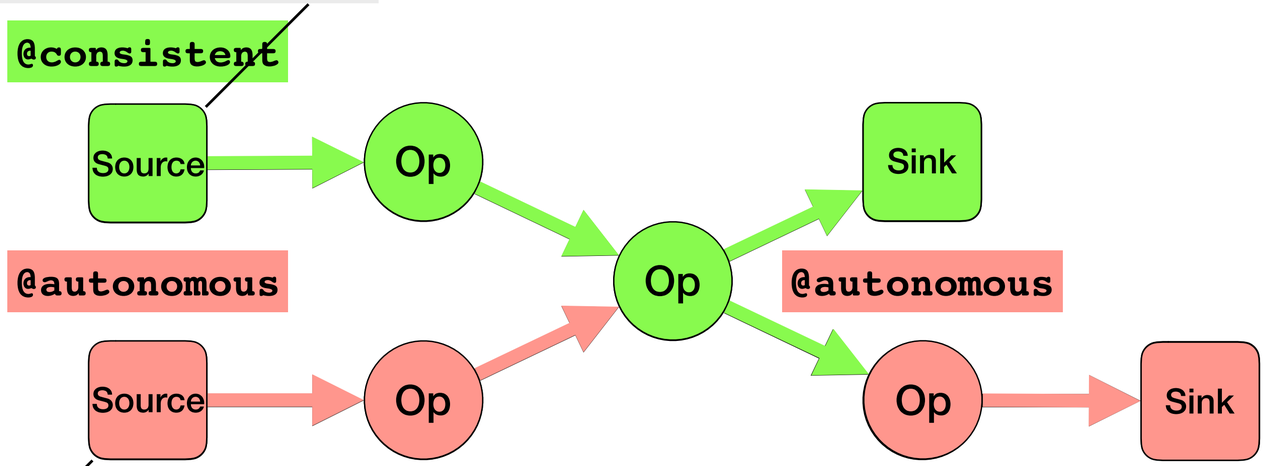


IBM Streams - Consistent Regions

Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region



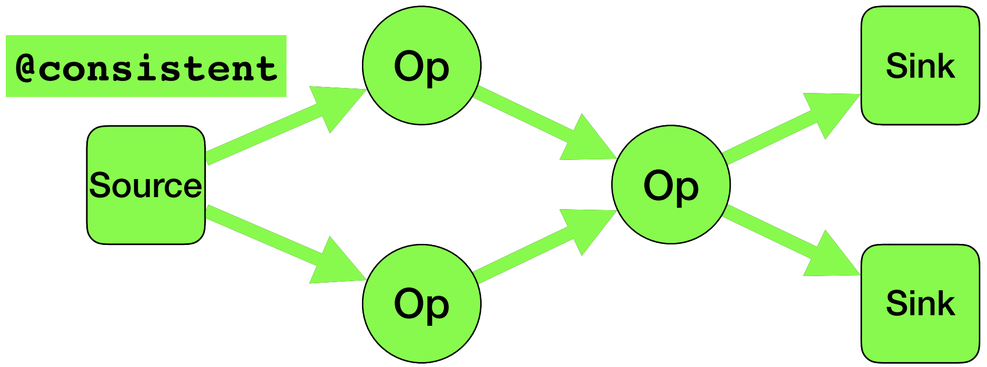
Consistent sources replay tuples

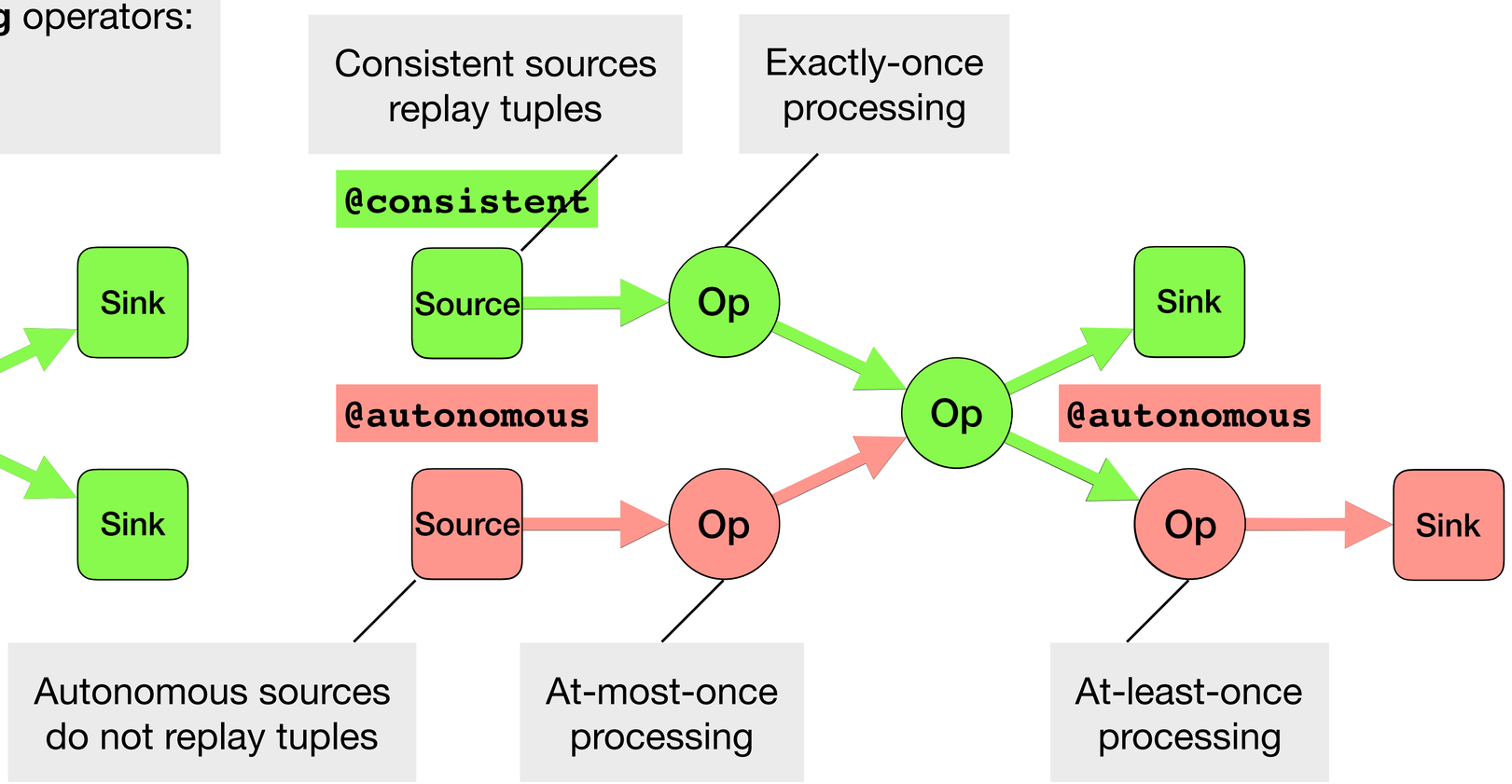


Autonomous sources do not replay tuples

IBM Streams - Consistent Regions

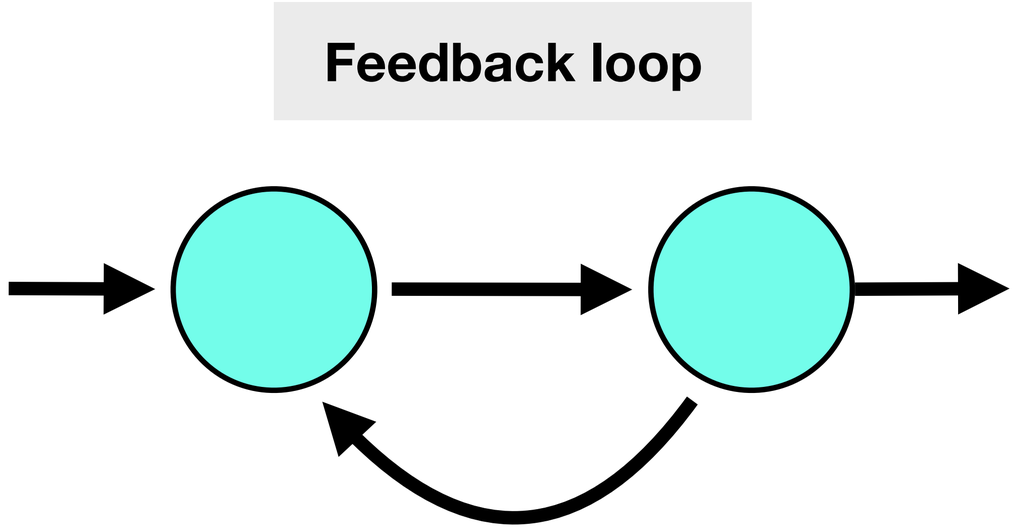
Snapshots only occur inside **consistent regions**
Consistent regions are formed by **annotating** operators:
- **`@consistent`** starts a consistent region
- **`@autonomous`** ends a consistent region





IBM Streams - Input ports

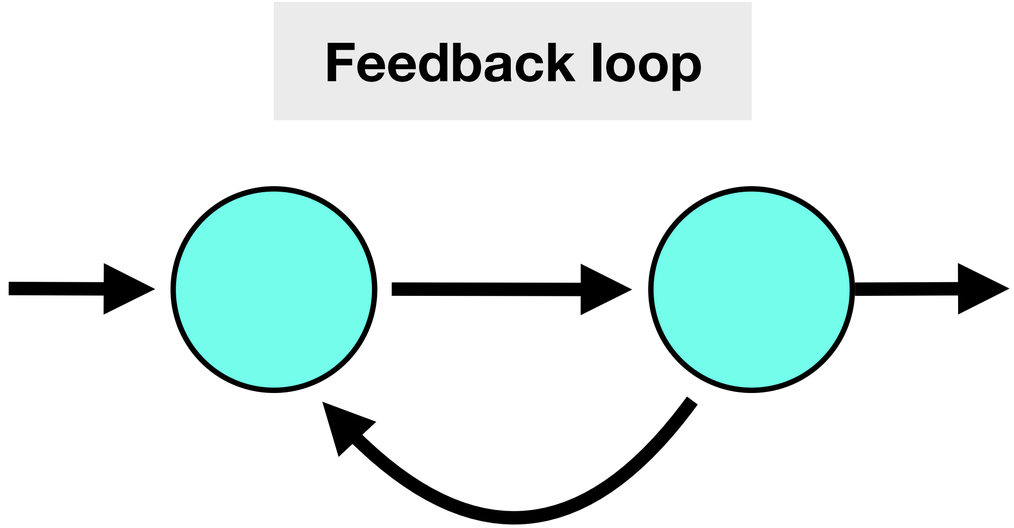
Problem: As in Flink, **feedback loops** must be considered when snapshotting



IBM Streams - Input ports

Problem: As in Flink, **feedback loops** must be considered when snapshotting

Solution: Model feedback loops explicitly

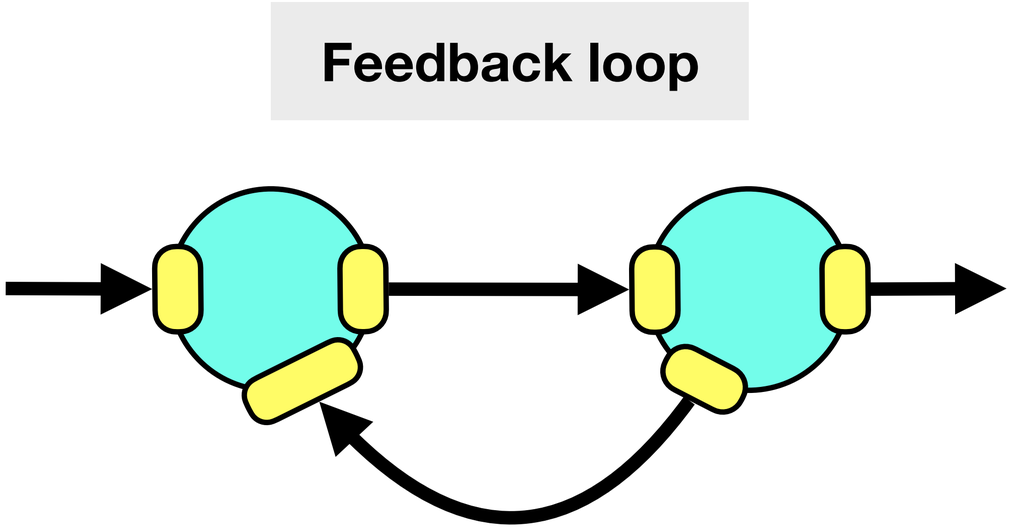


IBM Streams - Input ports

Problem: As in Flink, **feedback loops** must be considered when snapshotting

Solution: Model feedback loops explicitly

- Channels connect with operators through **input ports** and **output ports**

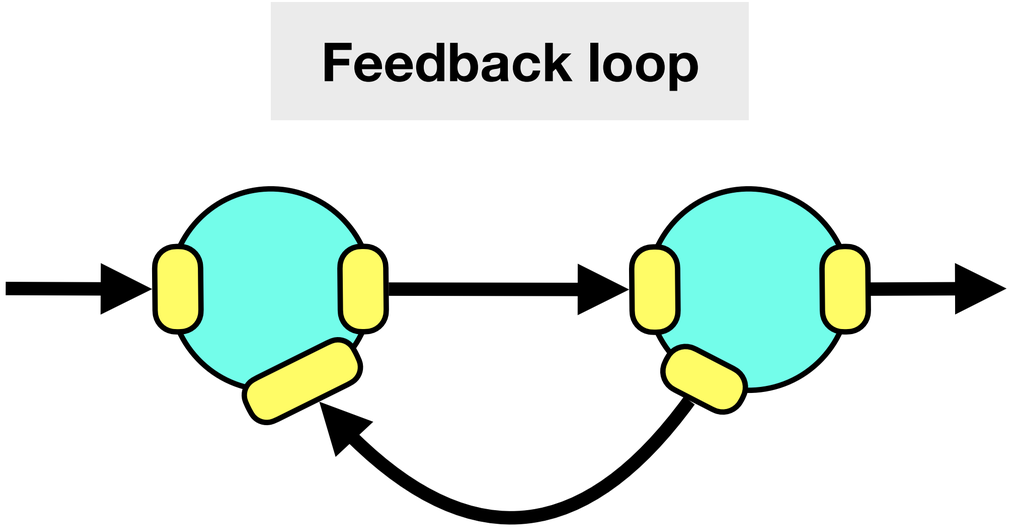


IBM Streams - Input ports

Problem: As in Flink, **feedback loops** must be considered when snapshotting

Solution: Model feedback loops explicitly

- Channels connect with operators through **input ports** and **output ports**
- **Input ports** are either ...

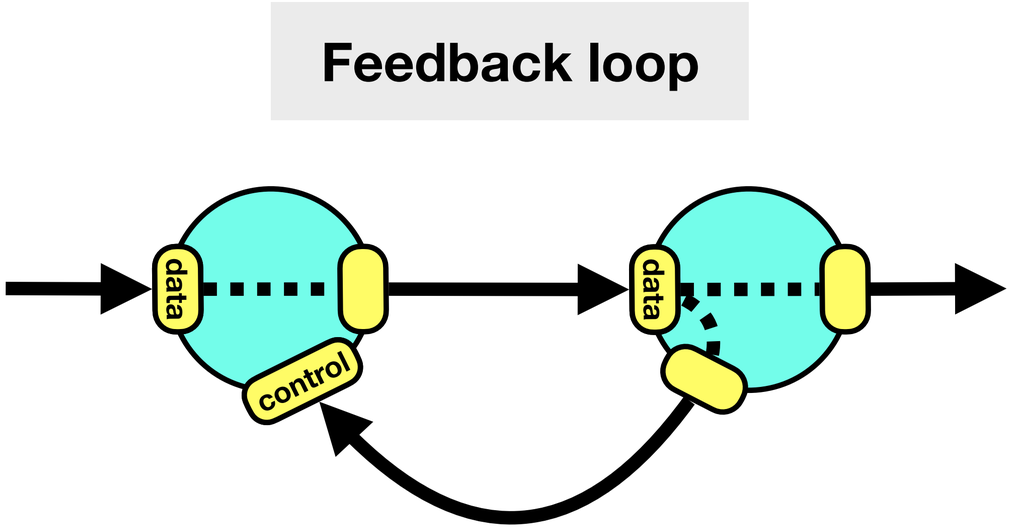


IBM Streams - Input ports

Problem: As in Flink, **feedback loops** must be considered when snapshotting

Solution: Model feedback loops explicitly

- Channels connect with operators through **input ports** and **output ports**
- **Input ports** are either ...
  - **Data ports**, that may **both** mutate state and output new elements
  - **Control ports**, that may **only** mutate state

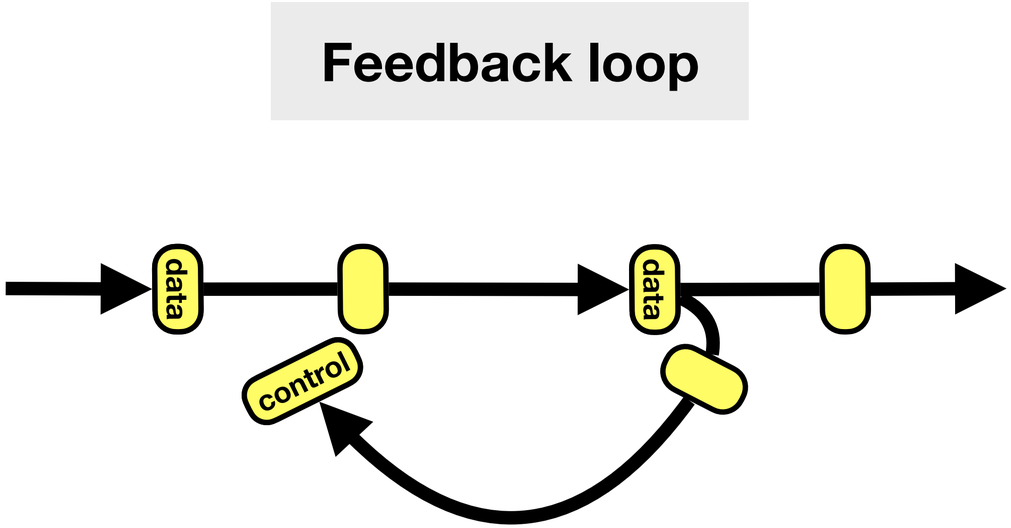


IBM Streams - Input ports

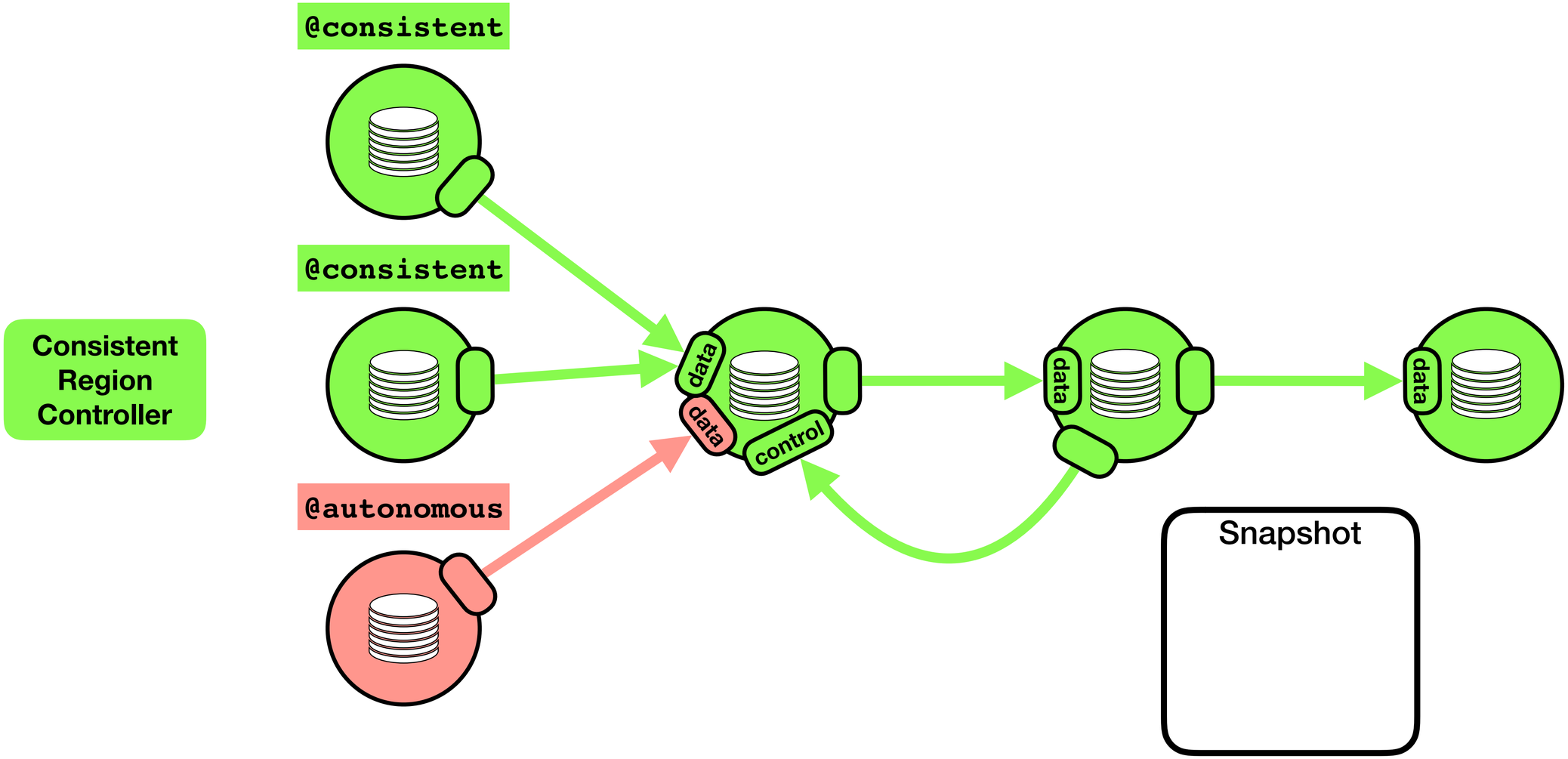
Problem: As in Flink, **feedback loops** must be considered when snapshotting

Solution: Model feedback loops explicitly

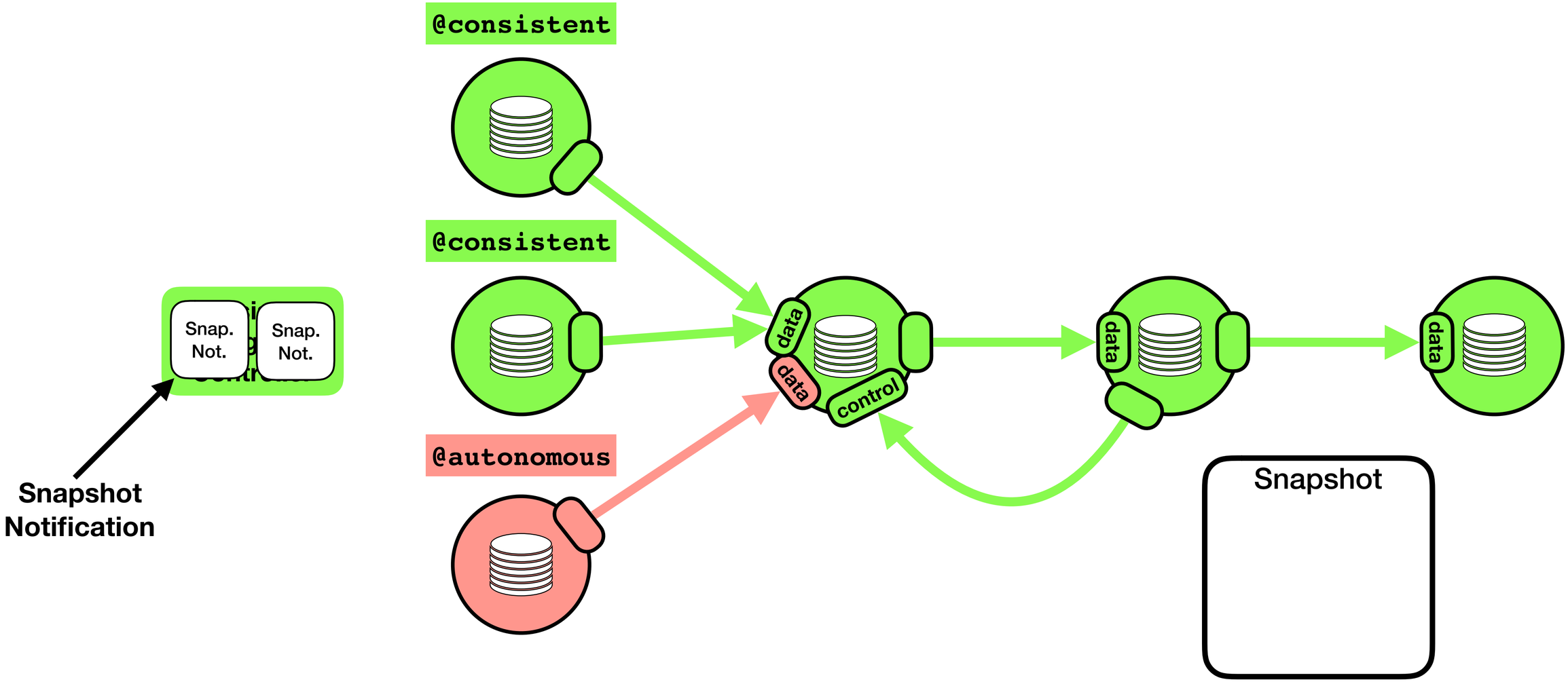
- Channels connect with operators through **input ports** and **output ports**
- **Input ports** are either ...
  - **Data ports**, that may **both** mutate state and output new elements
  - **Control ports**, that may **only** mutate state



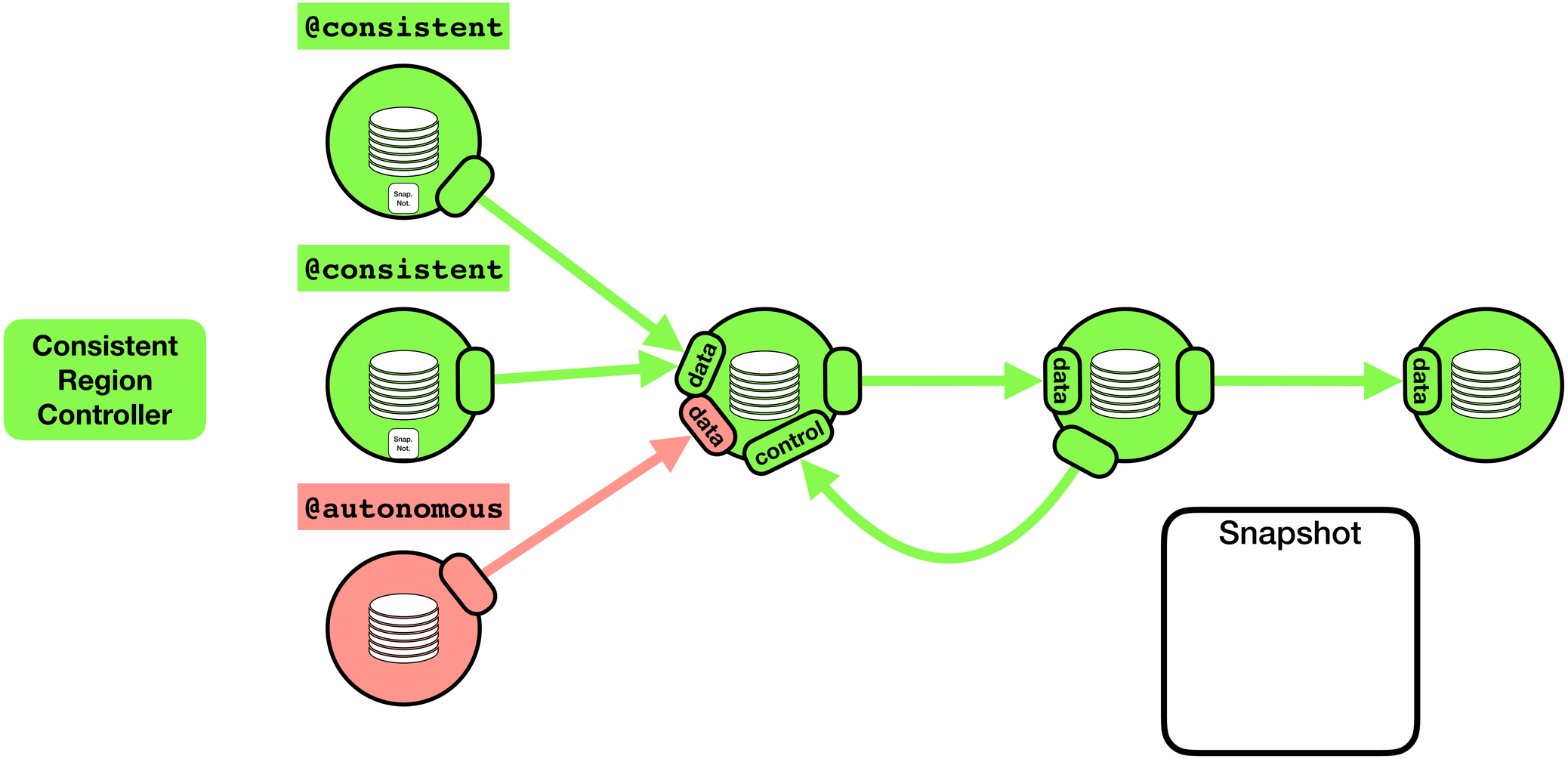
IBM Streams - Snapshotting Protocol



IBM Streams - Snapshotting Protocol

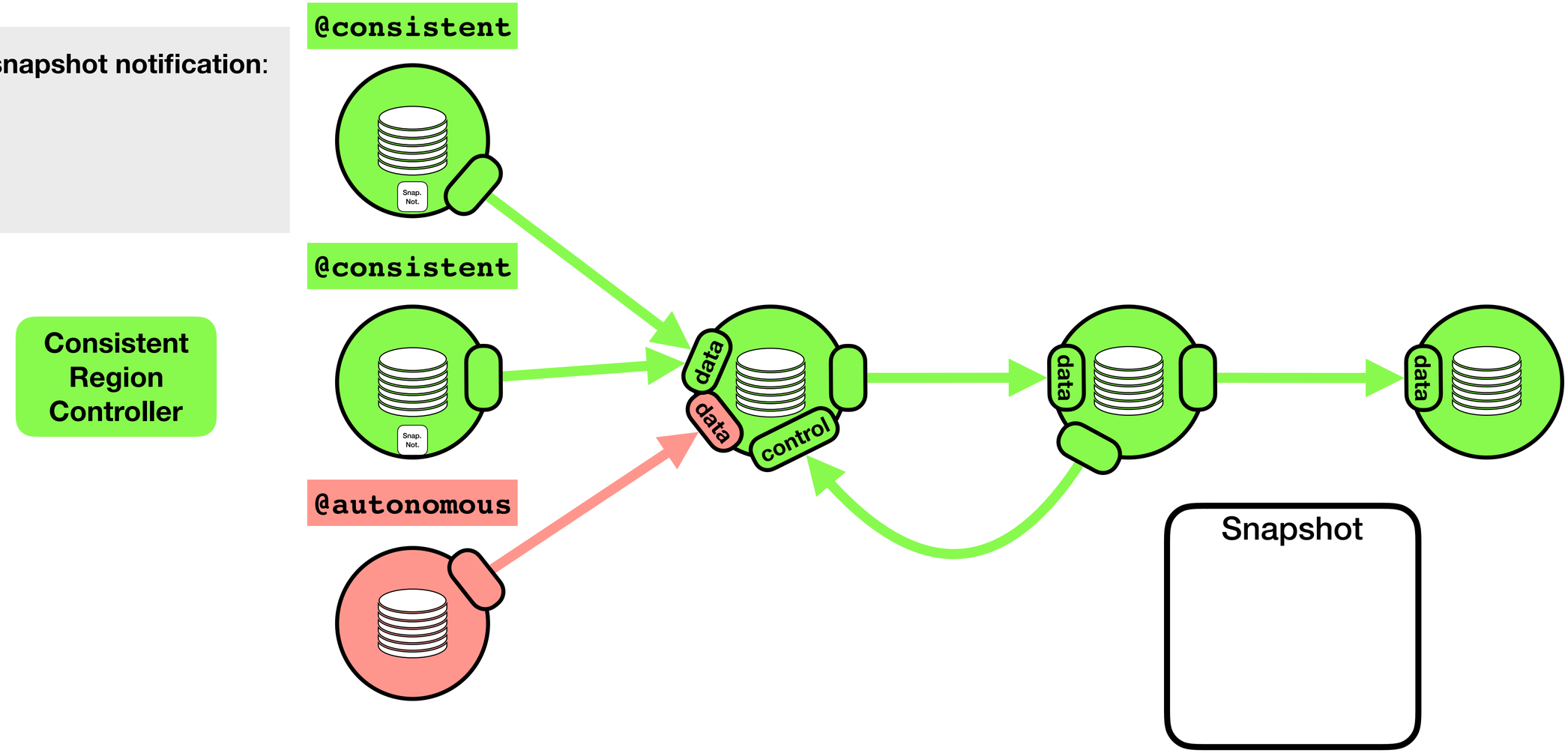


IBM Streams - Snapshotting Protocol



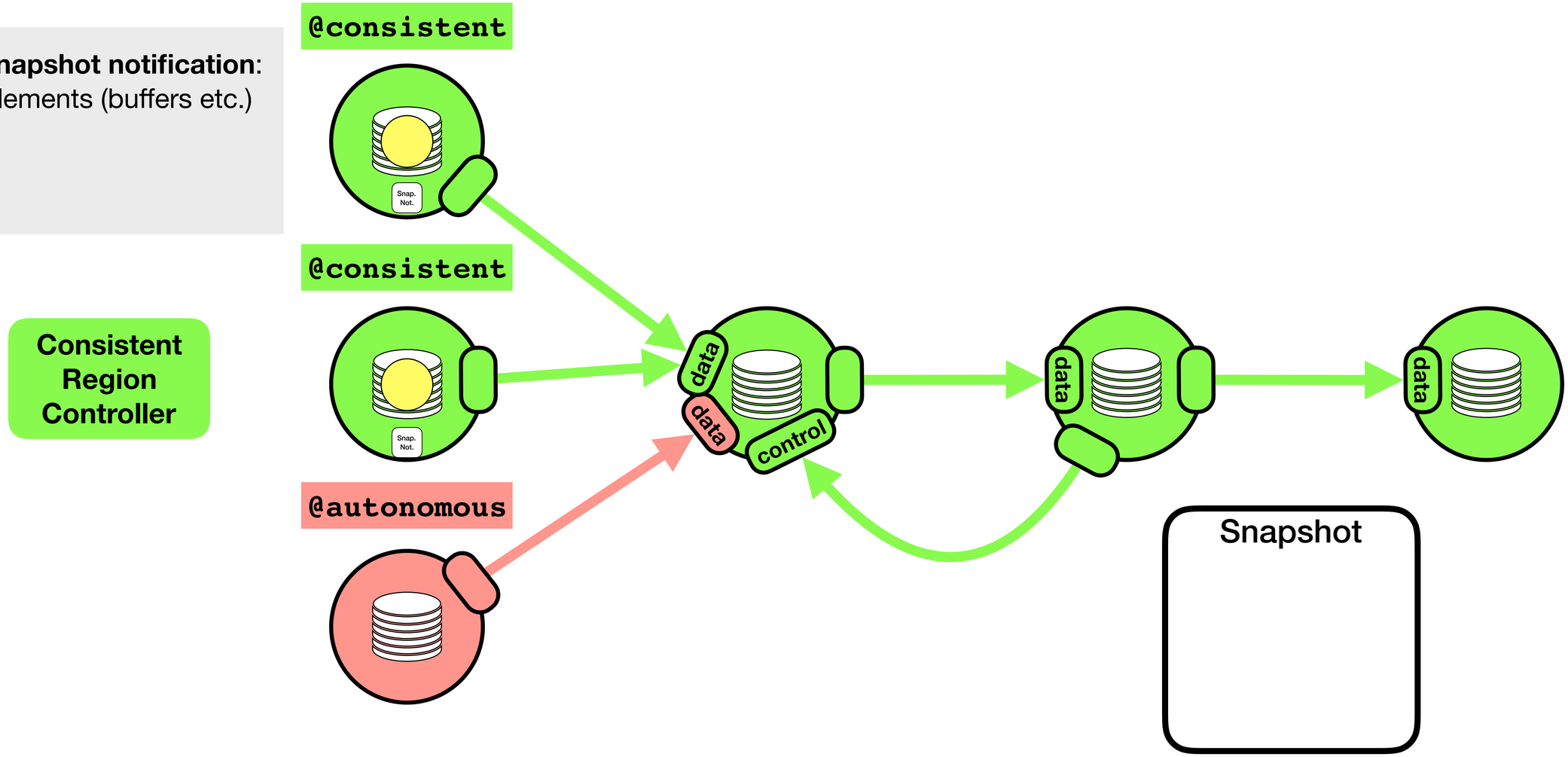
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:



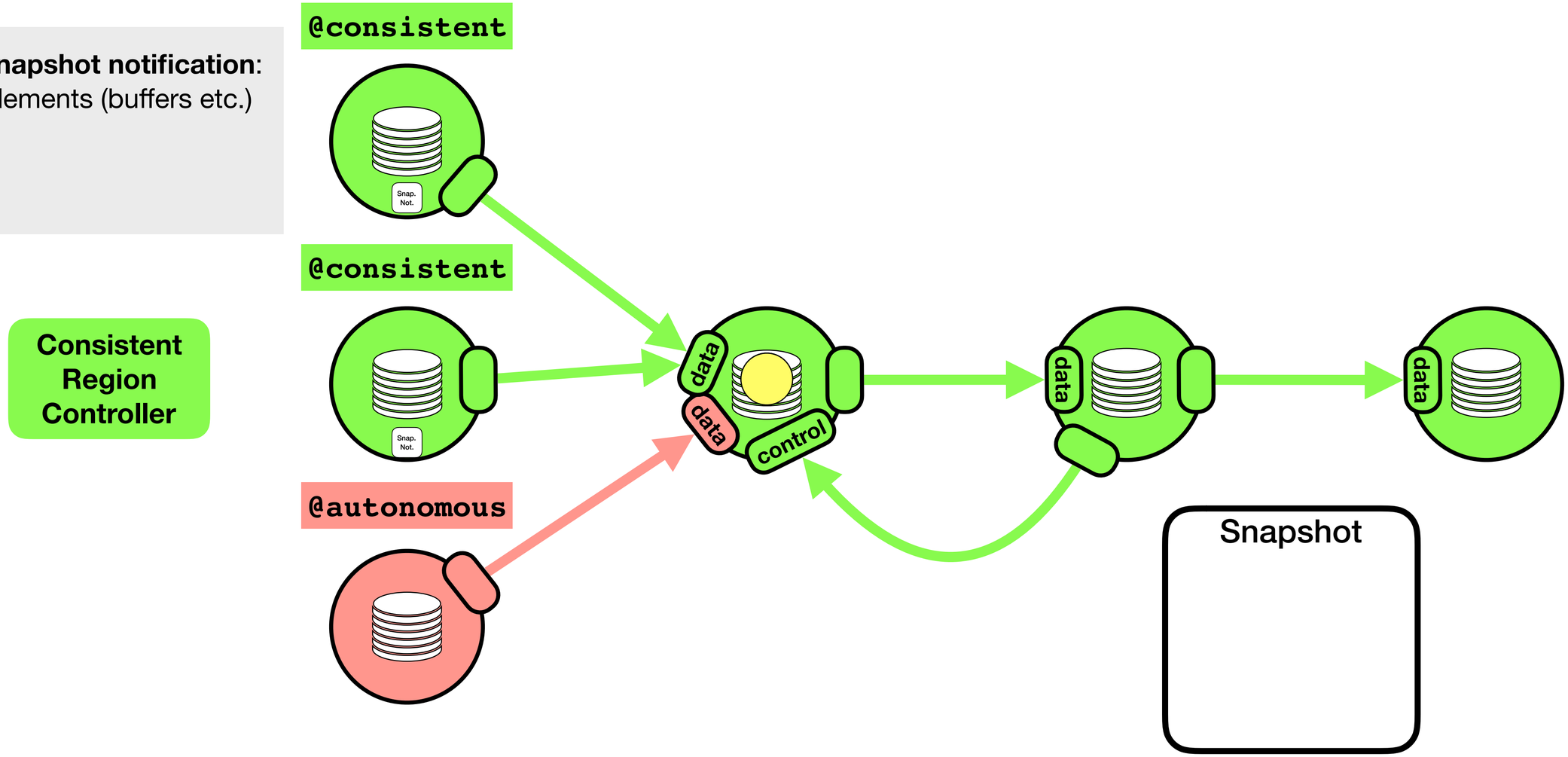
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)



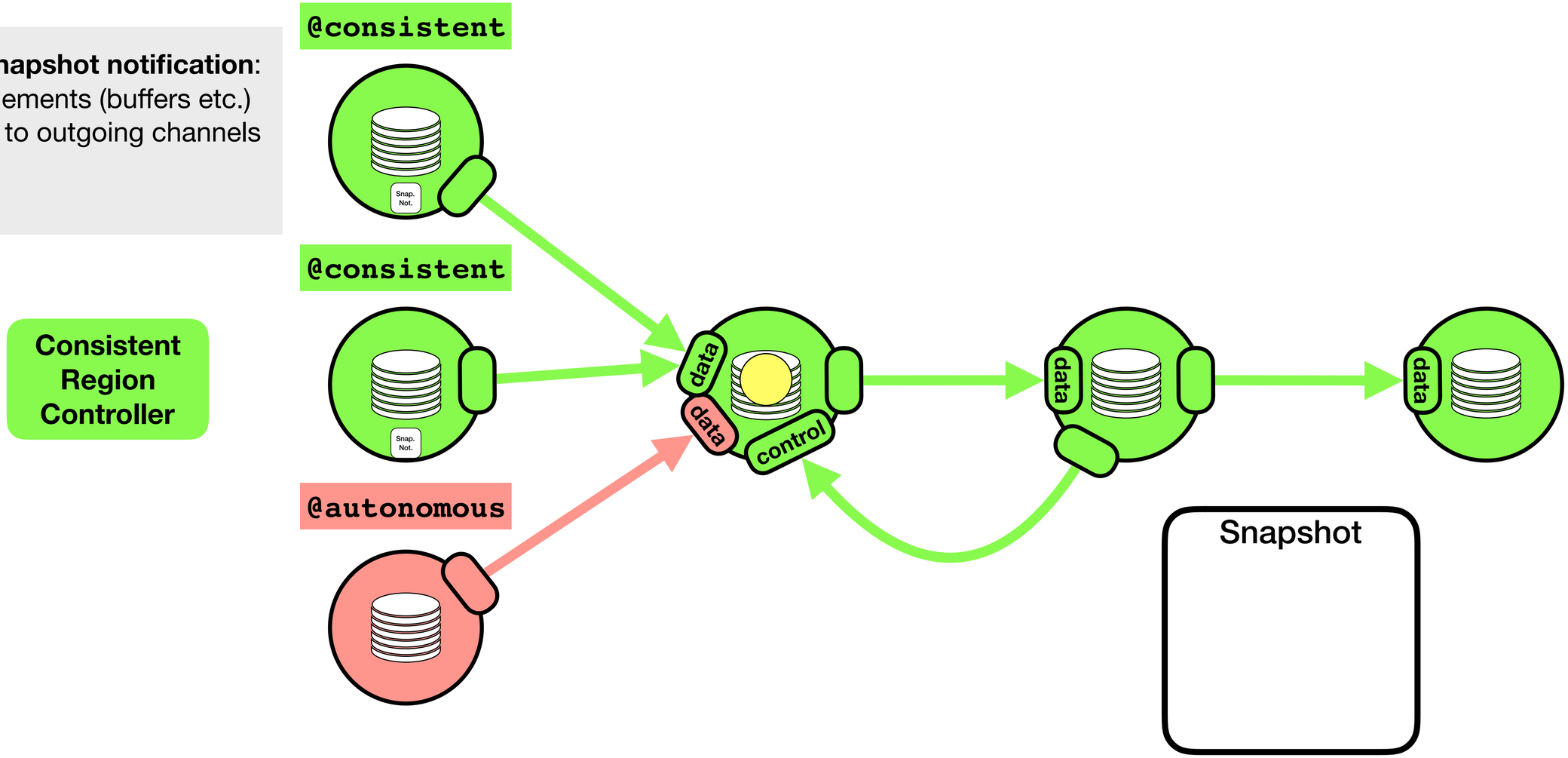
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)



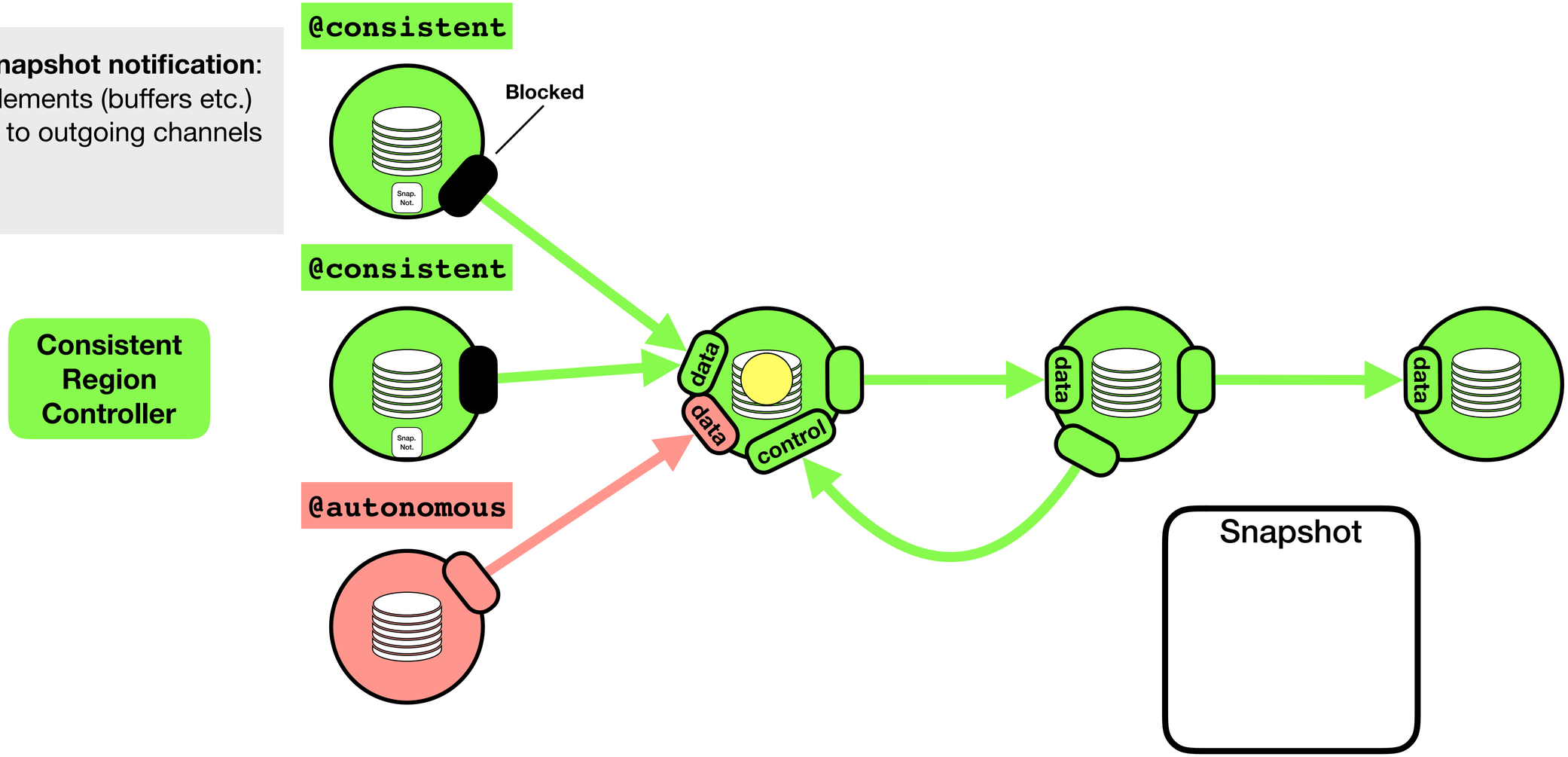
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels



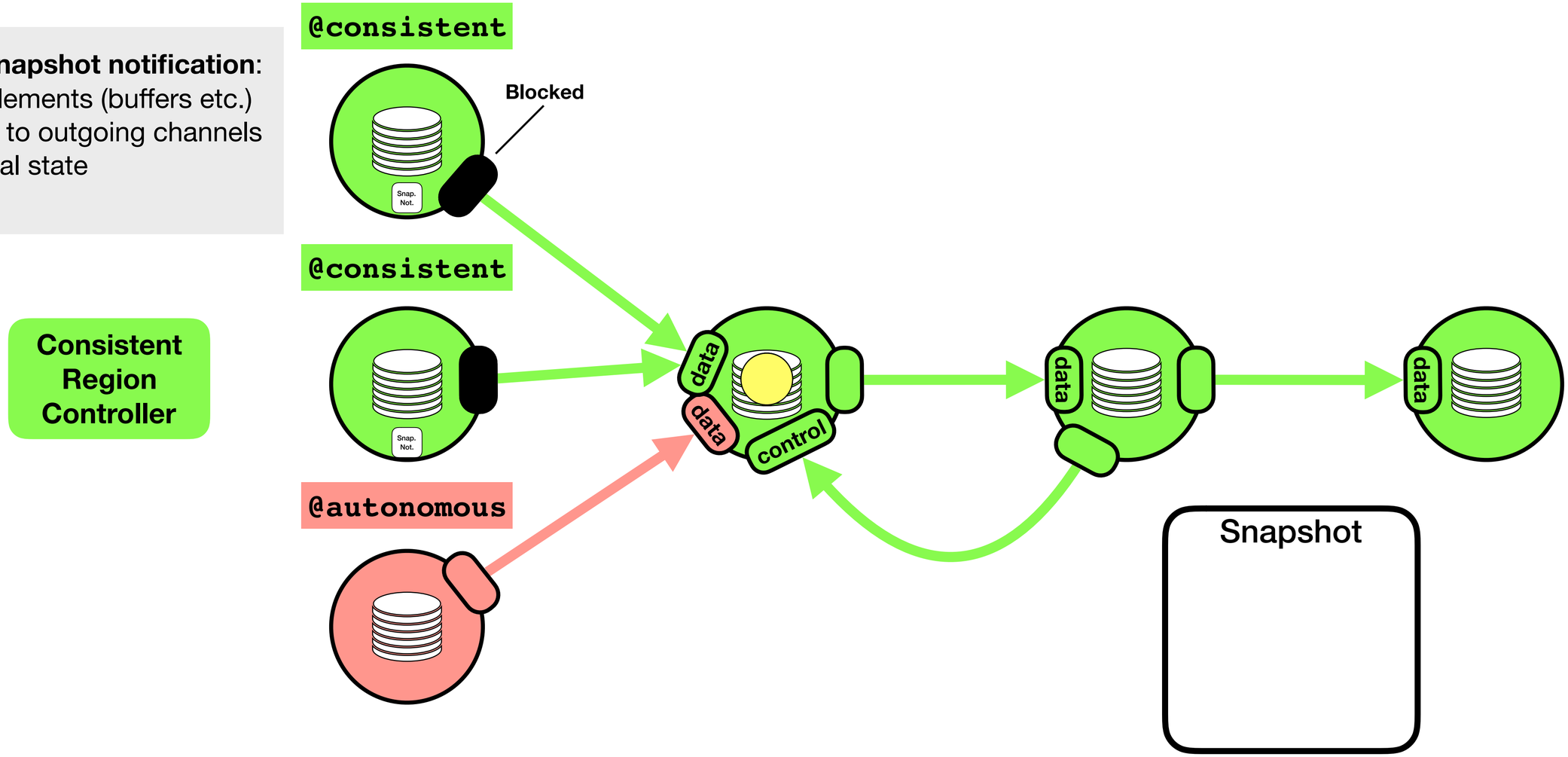
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels



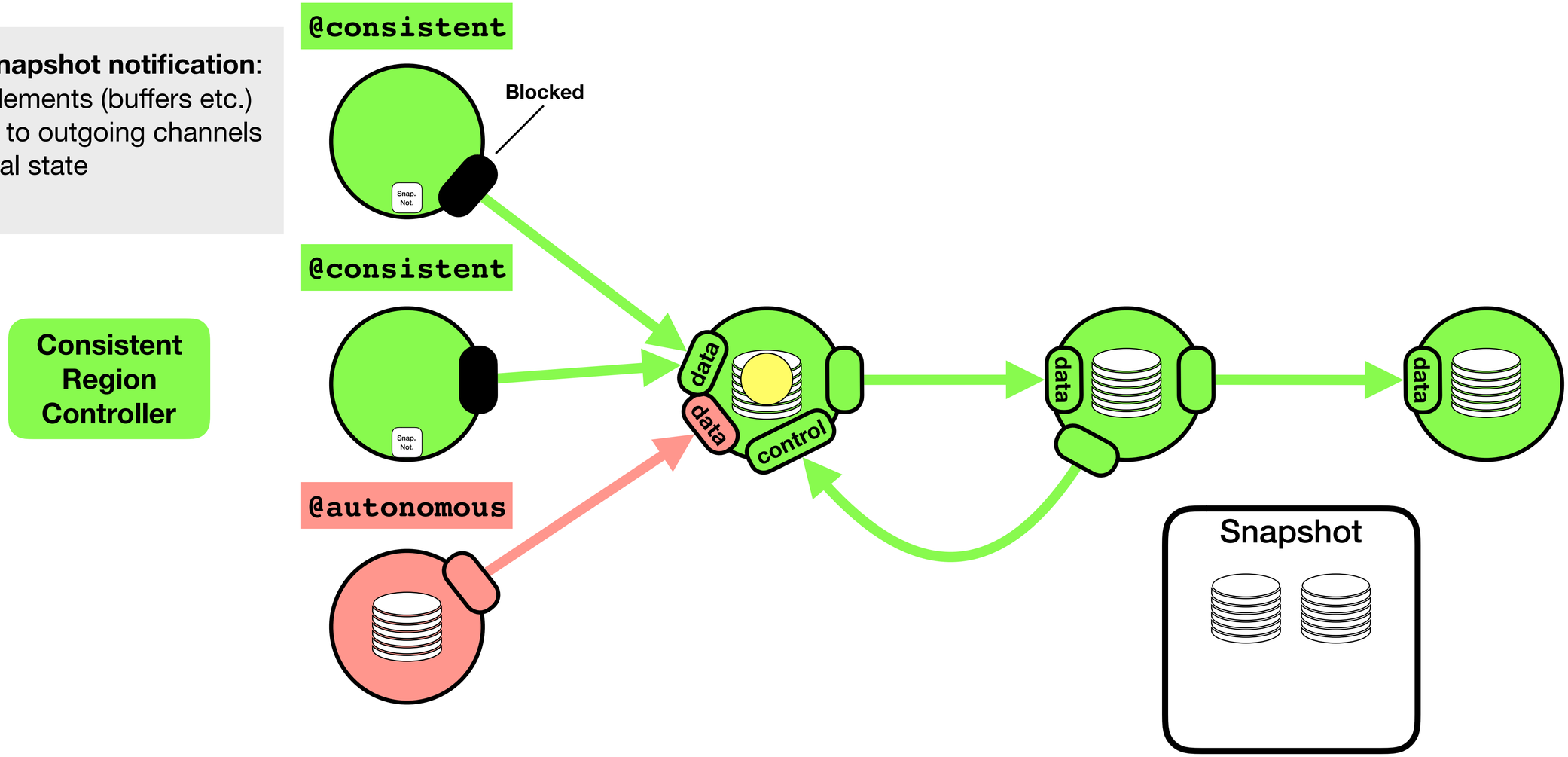
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state



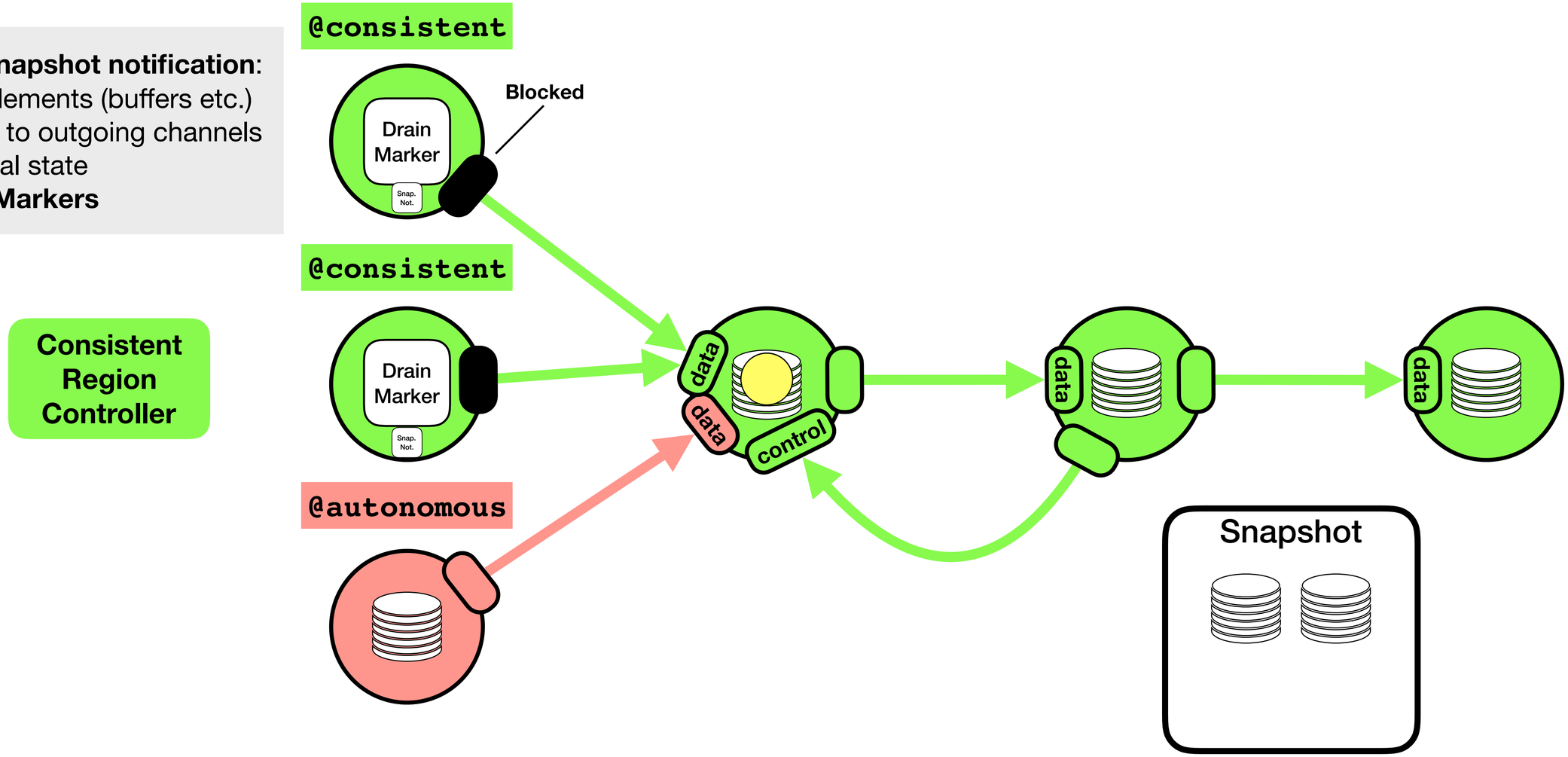
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state



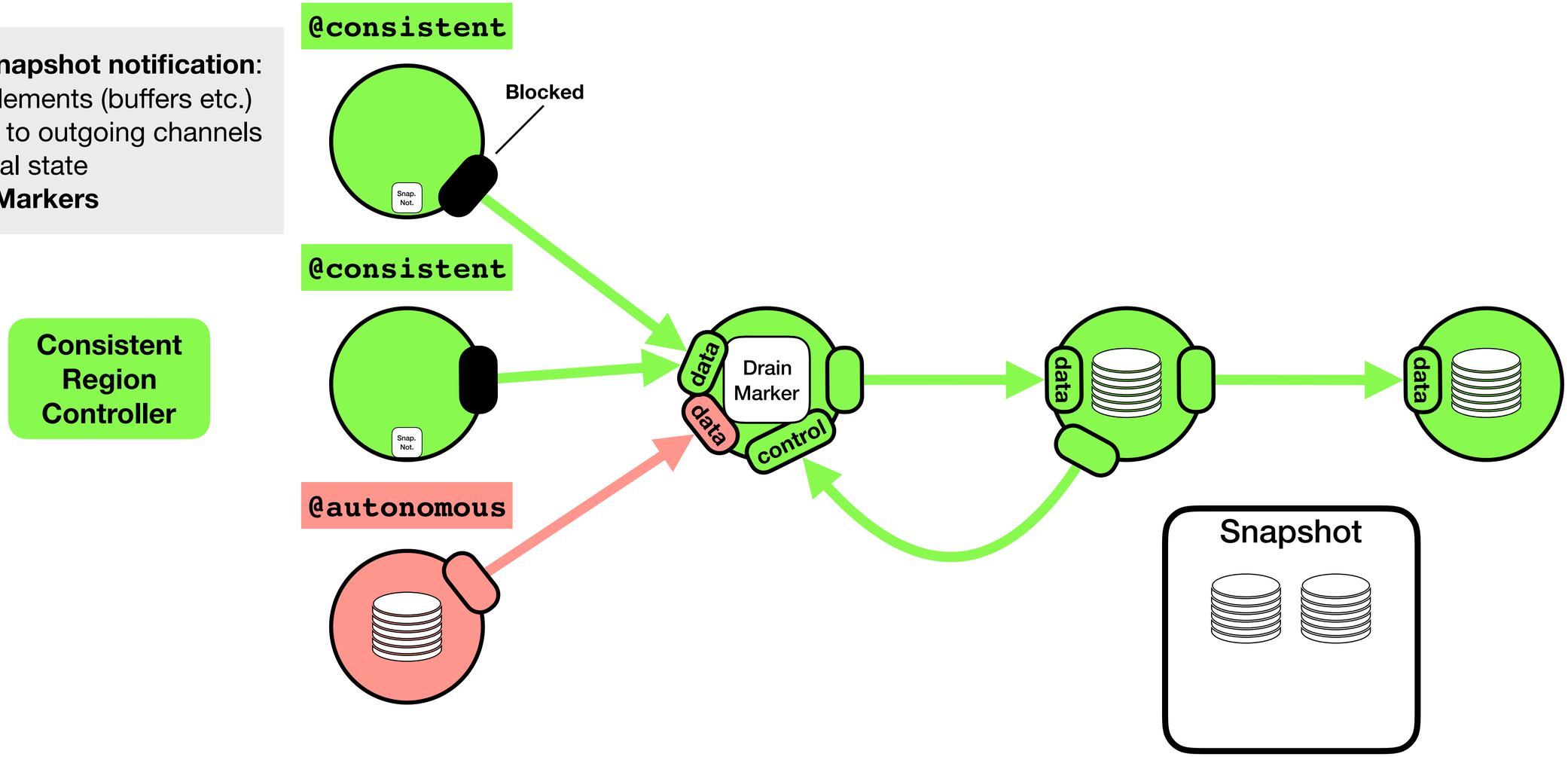
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**



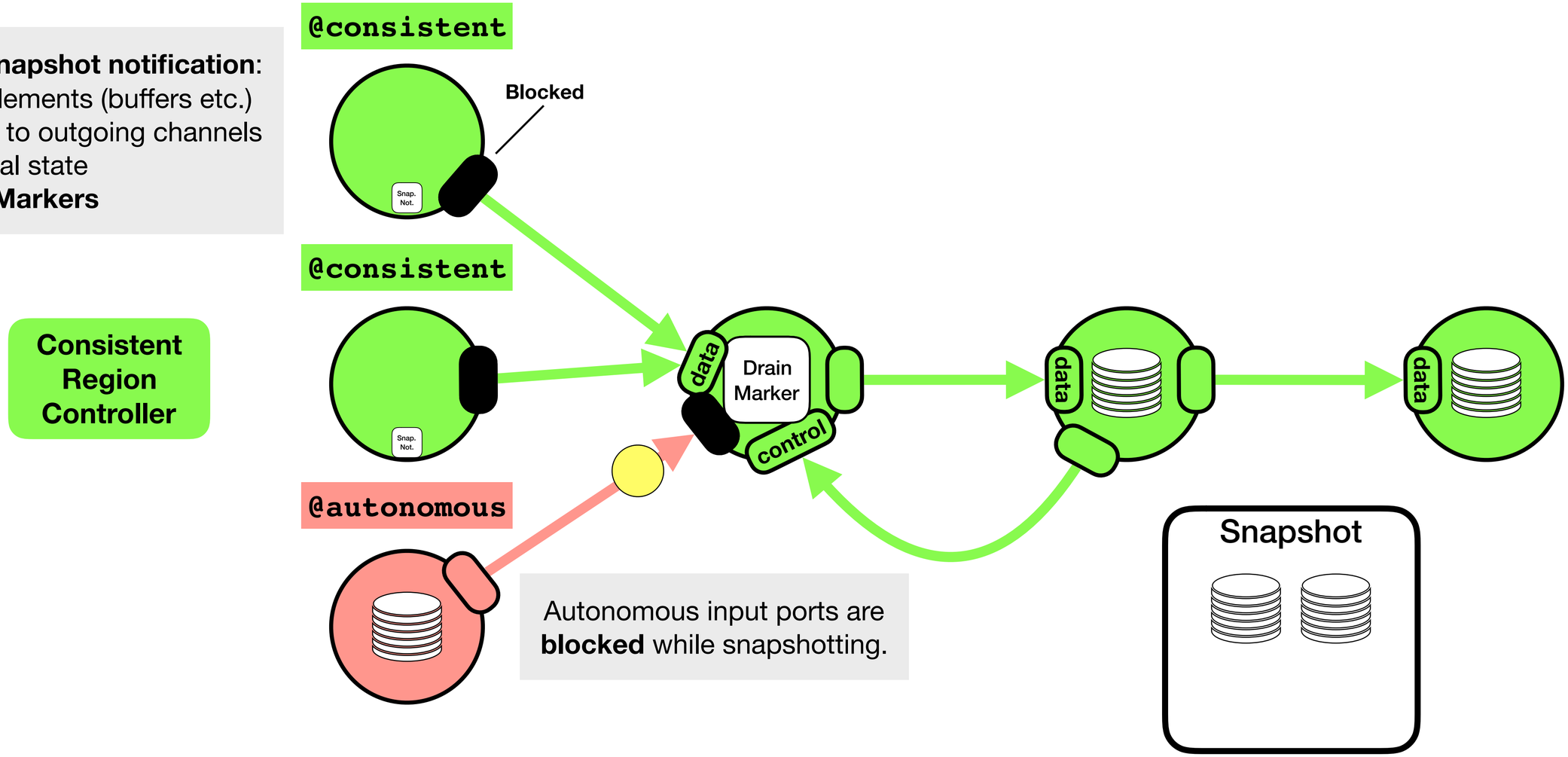
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**



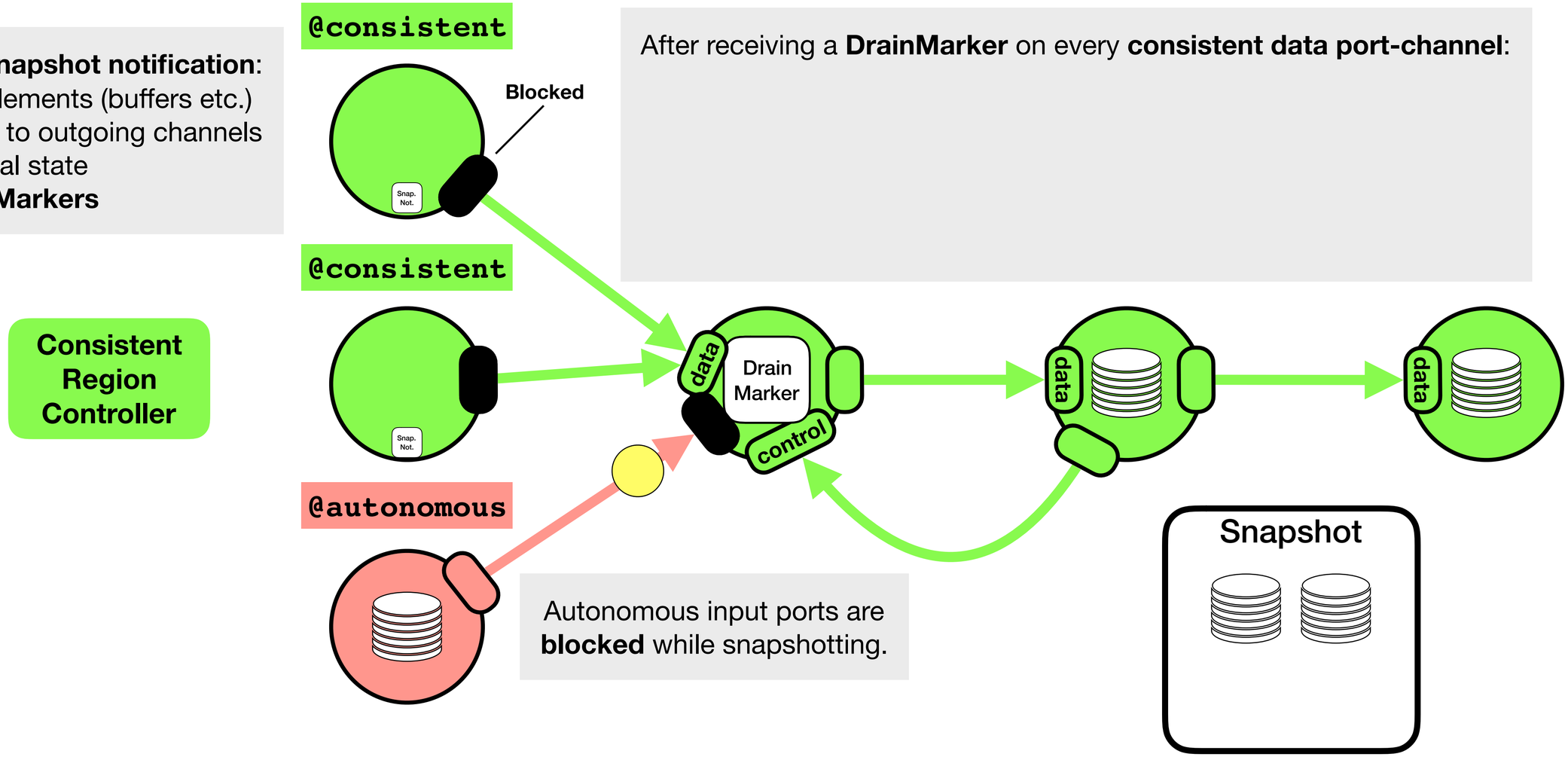
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**



IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

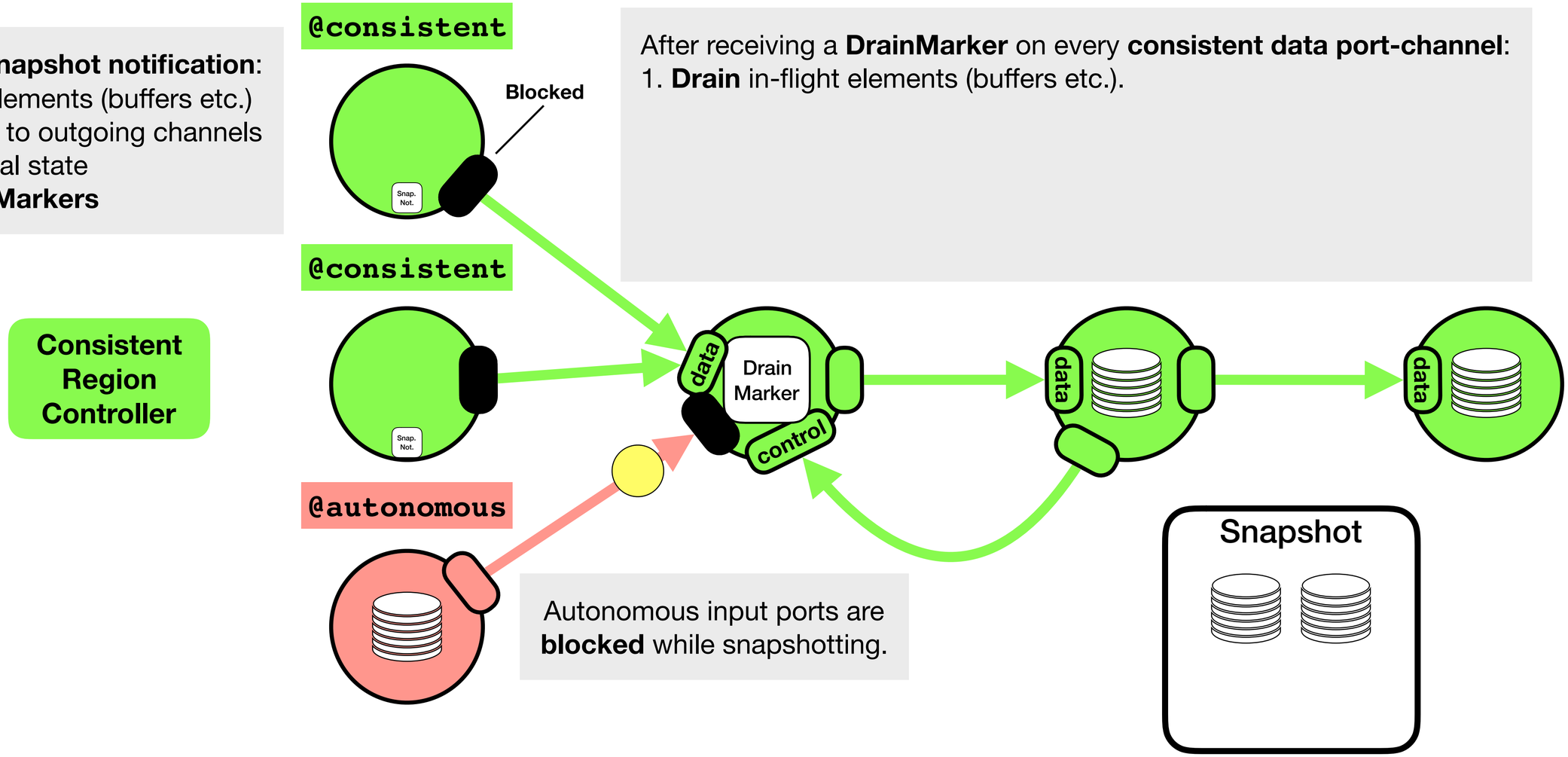


After receiving a **DrainMarker** on every **consistent data port-channel**:

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

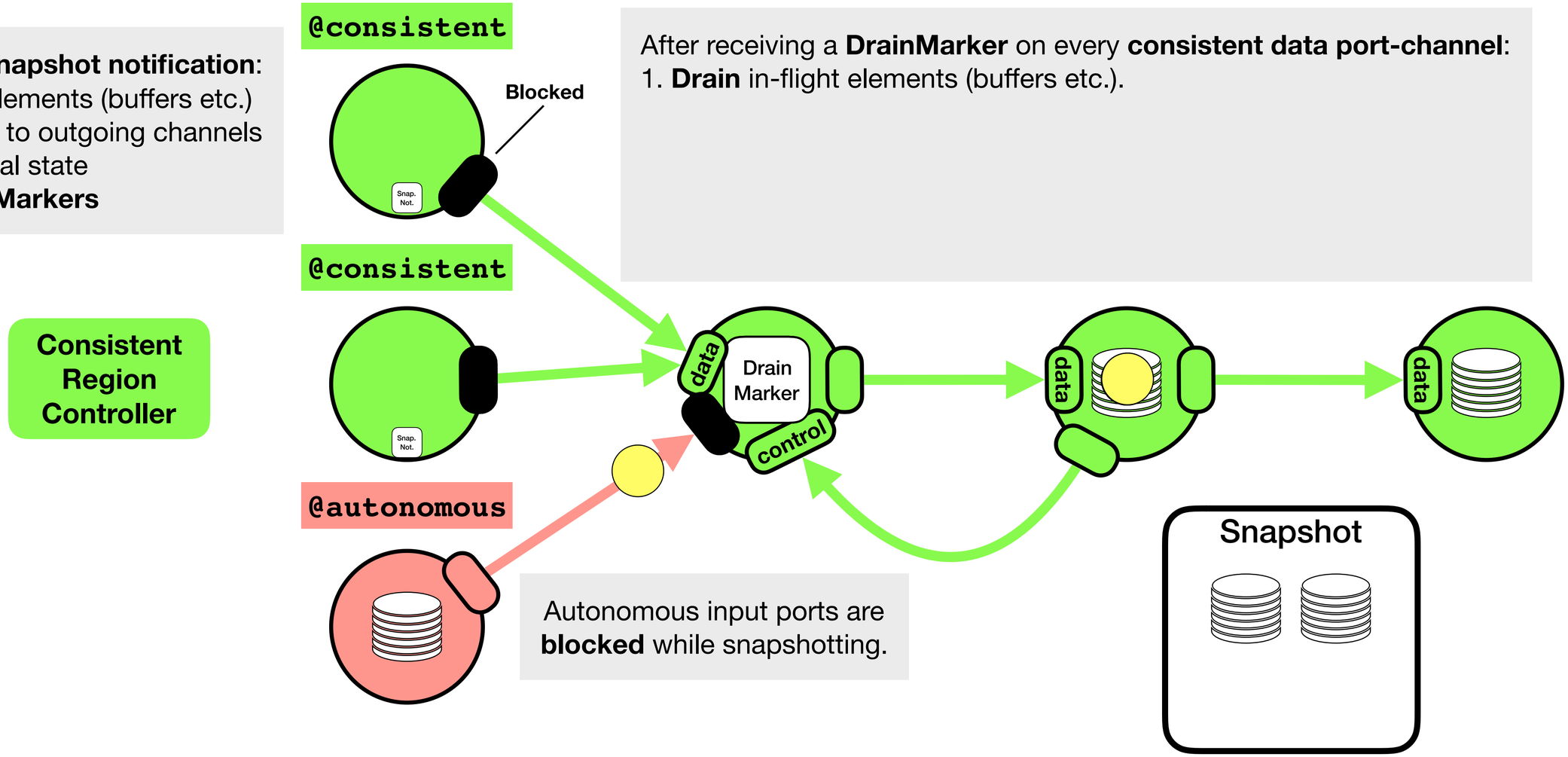


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

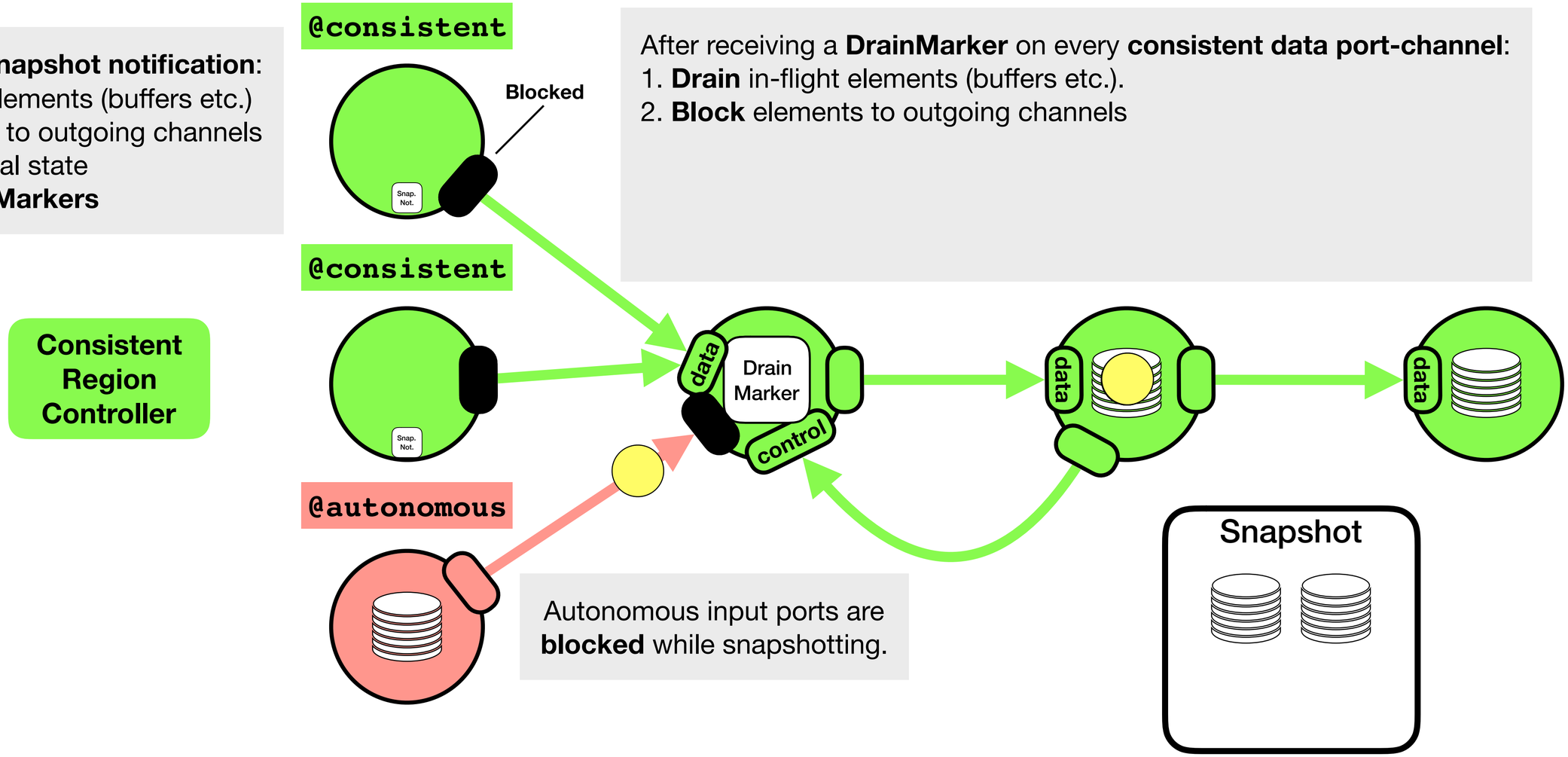


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

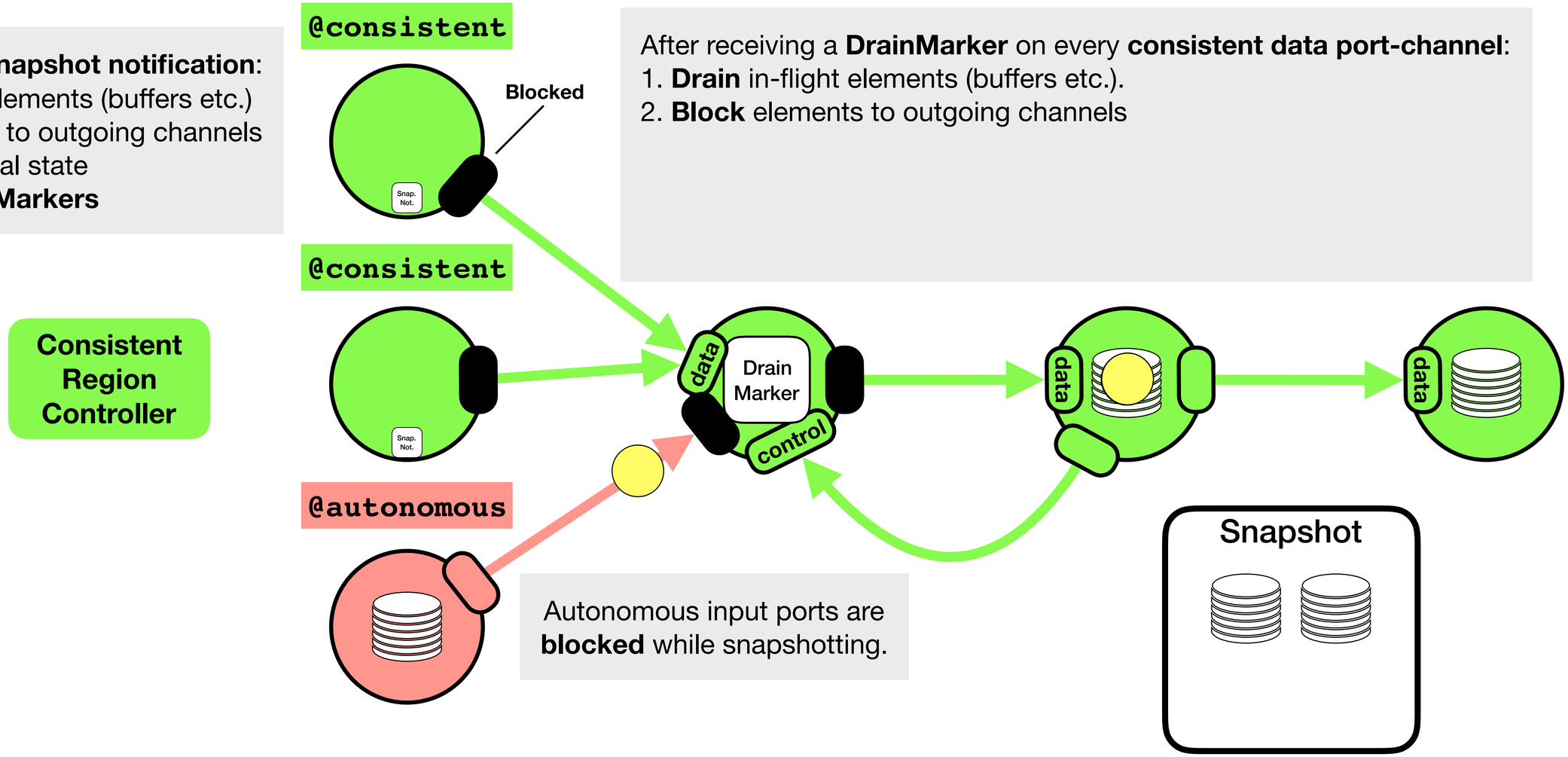


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

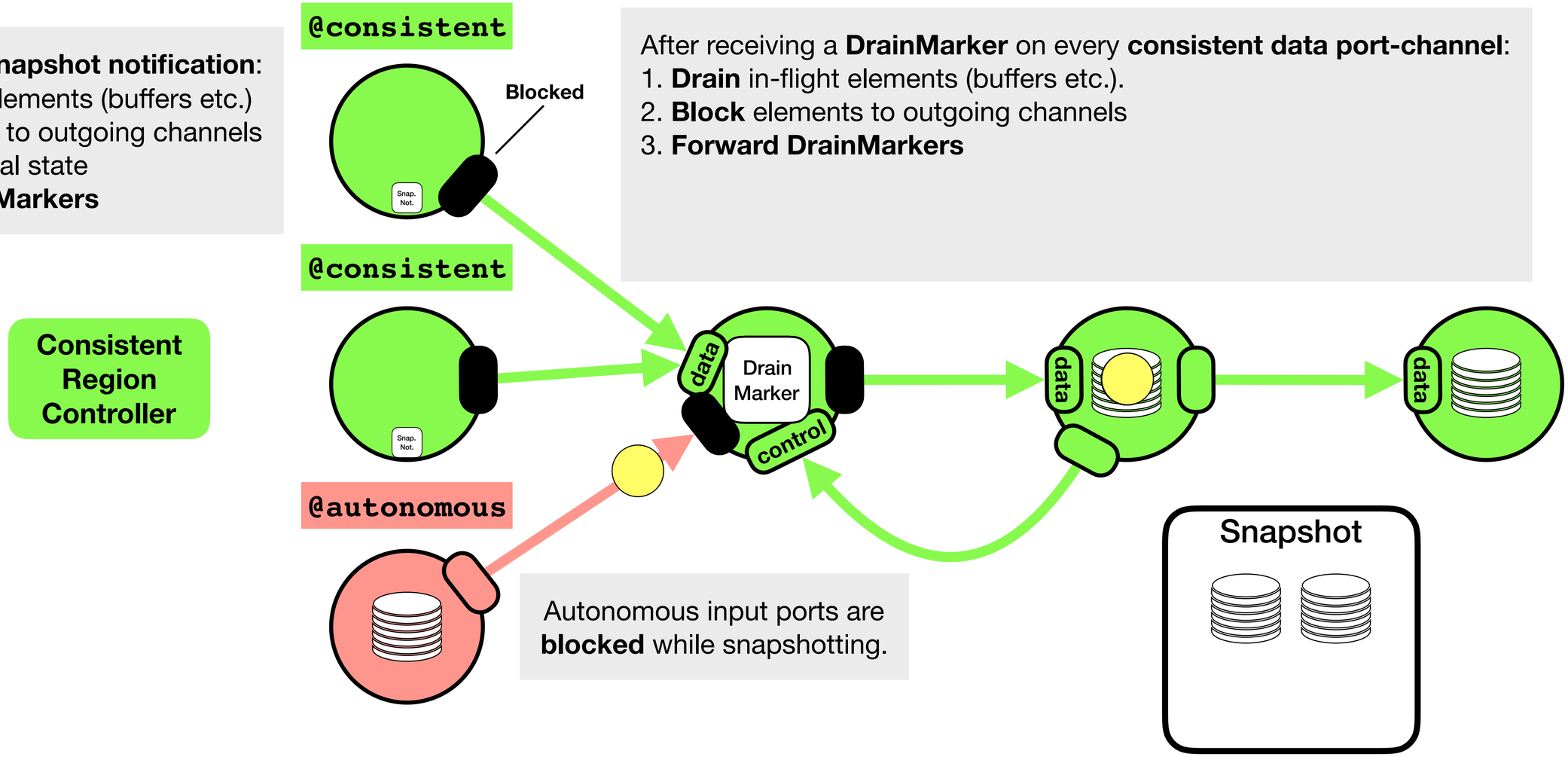


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

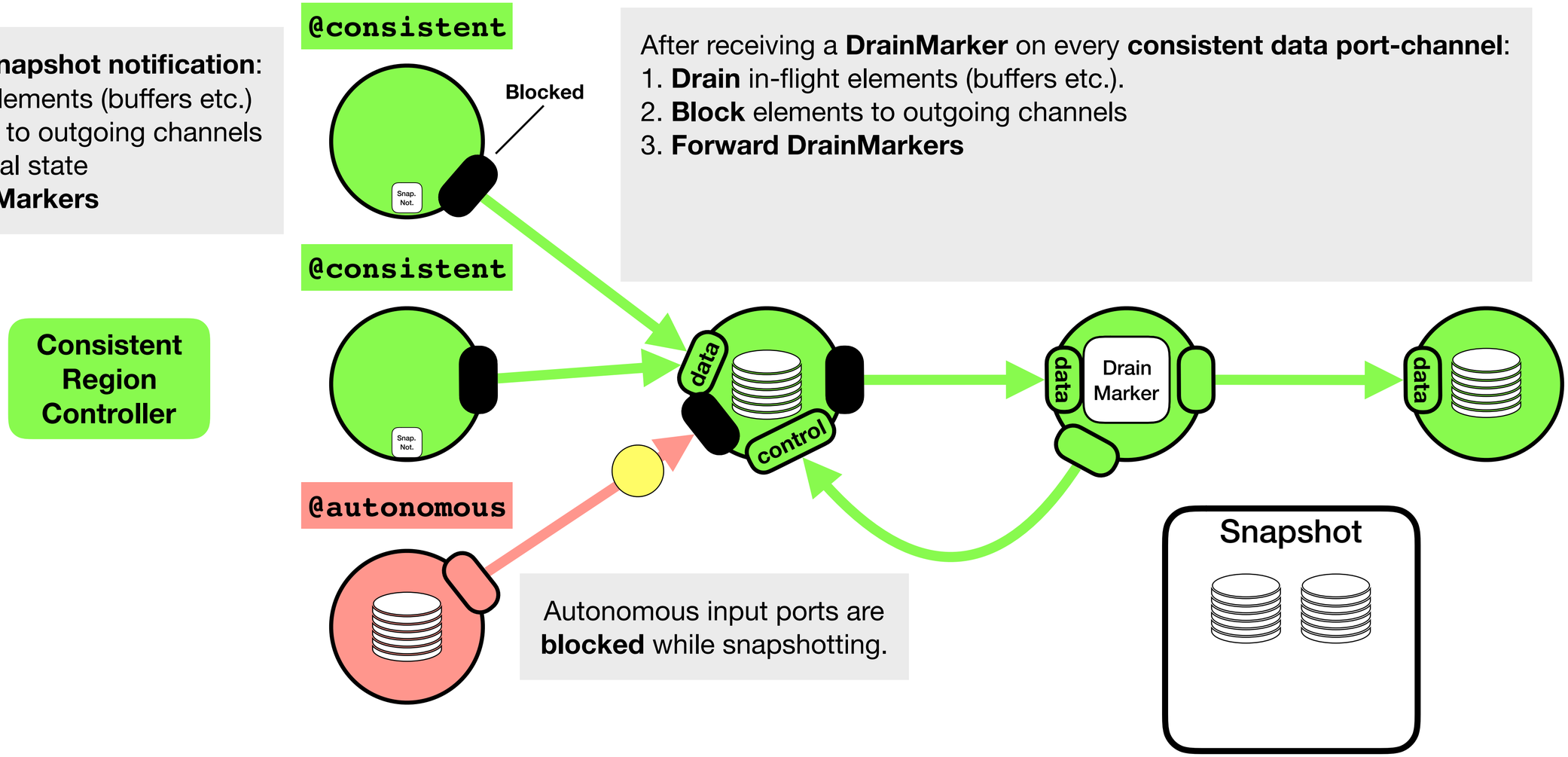


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

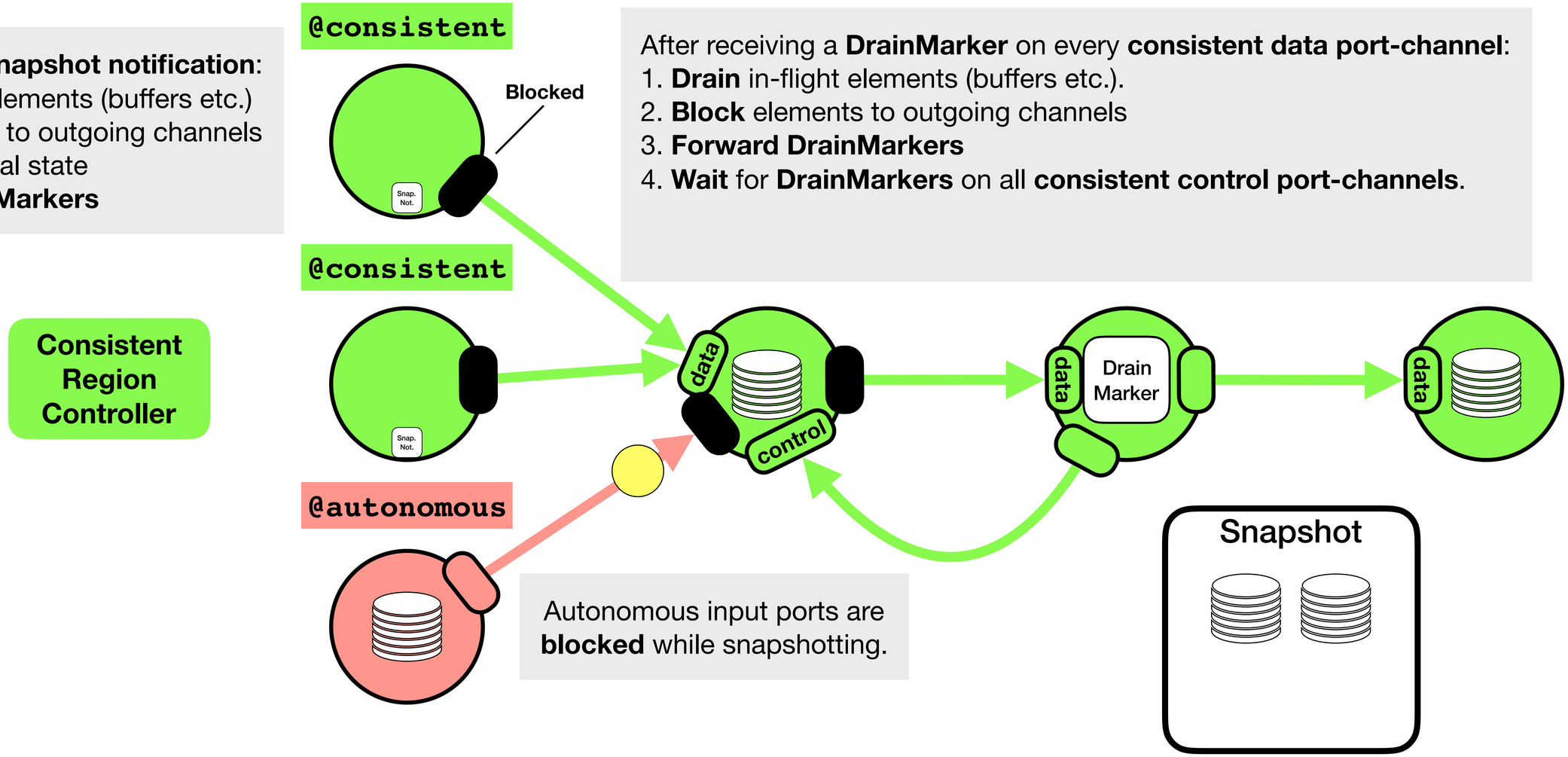


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

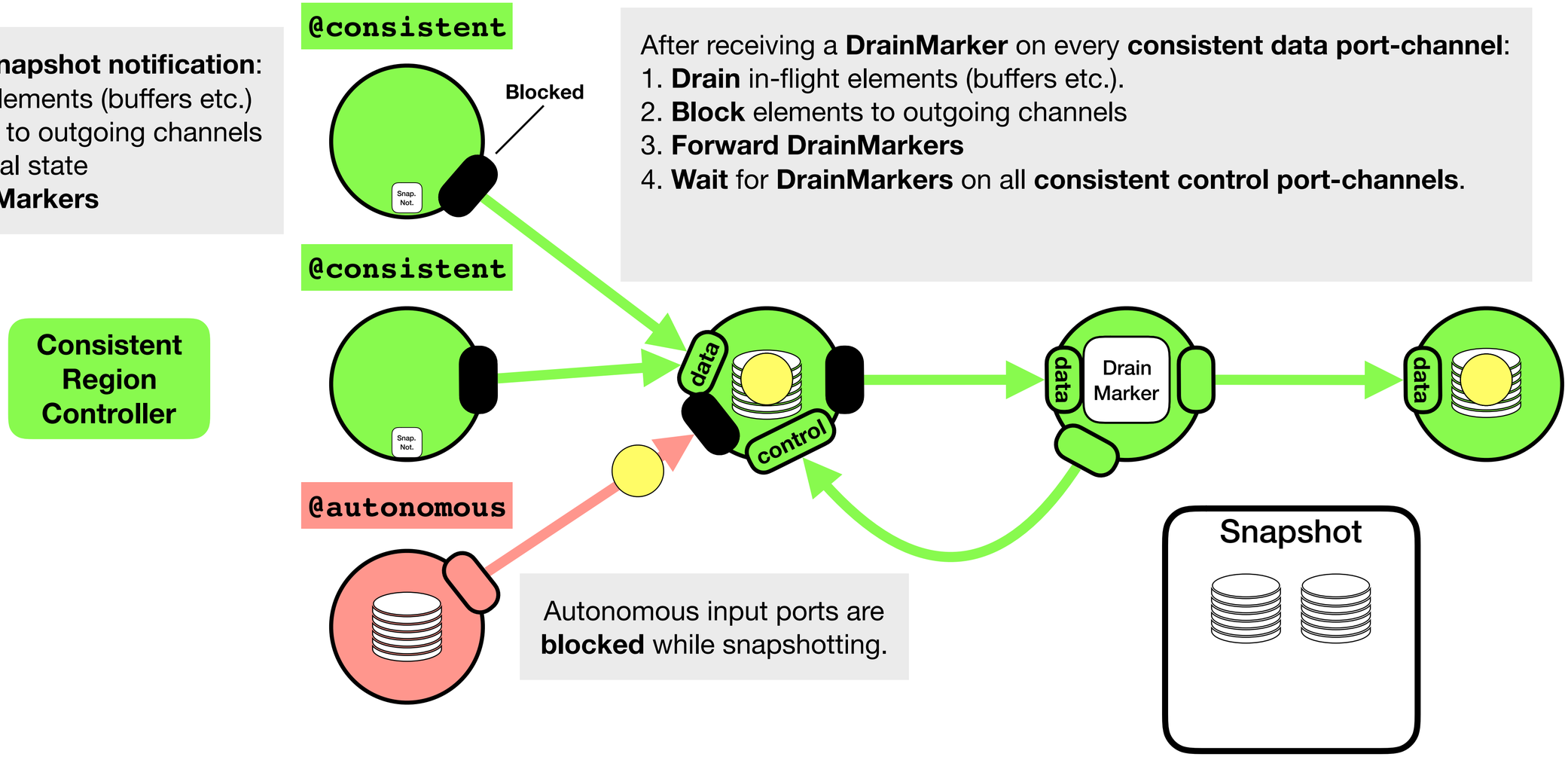


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

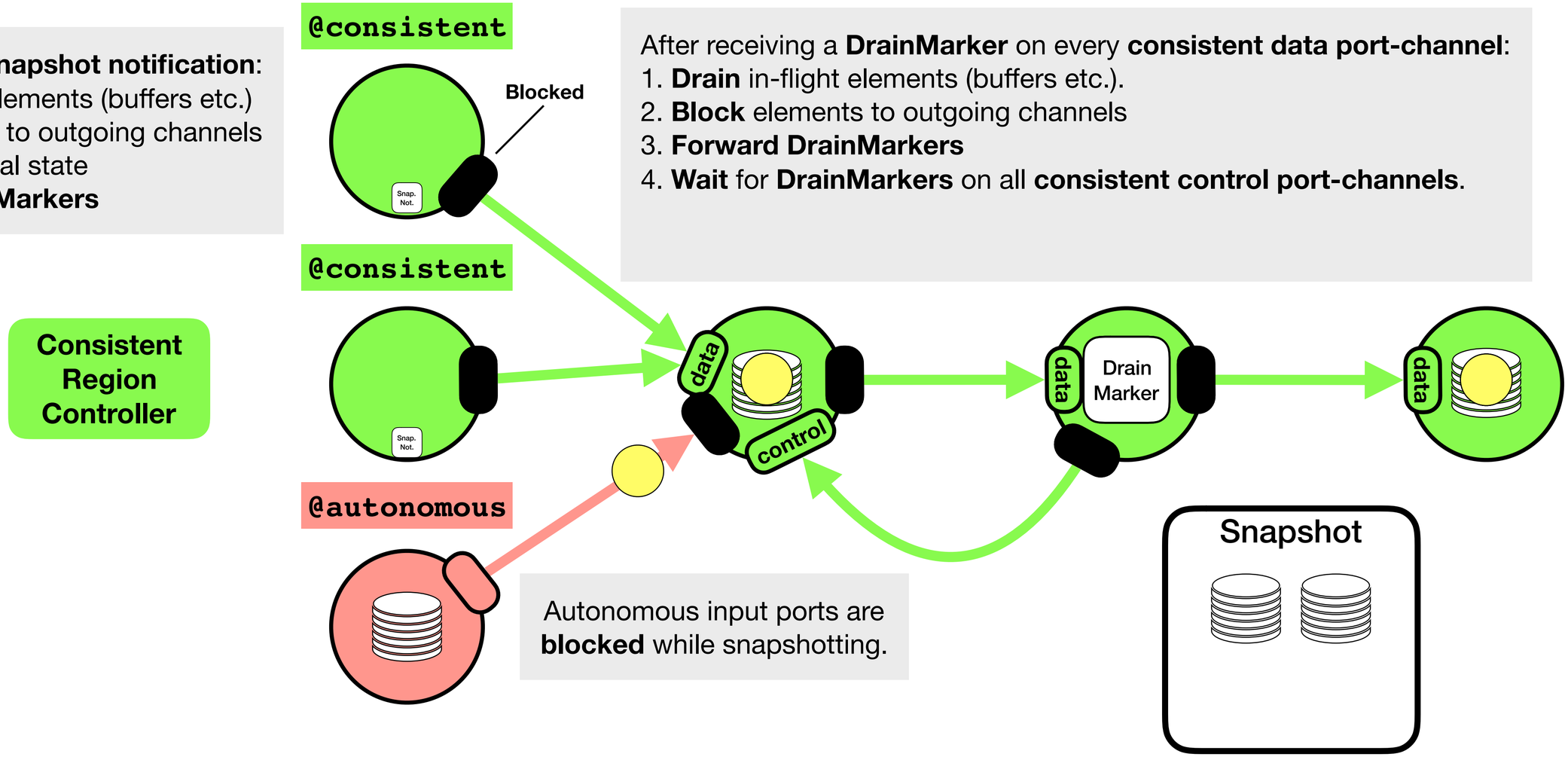


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

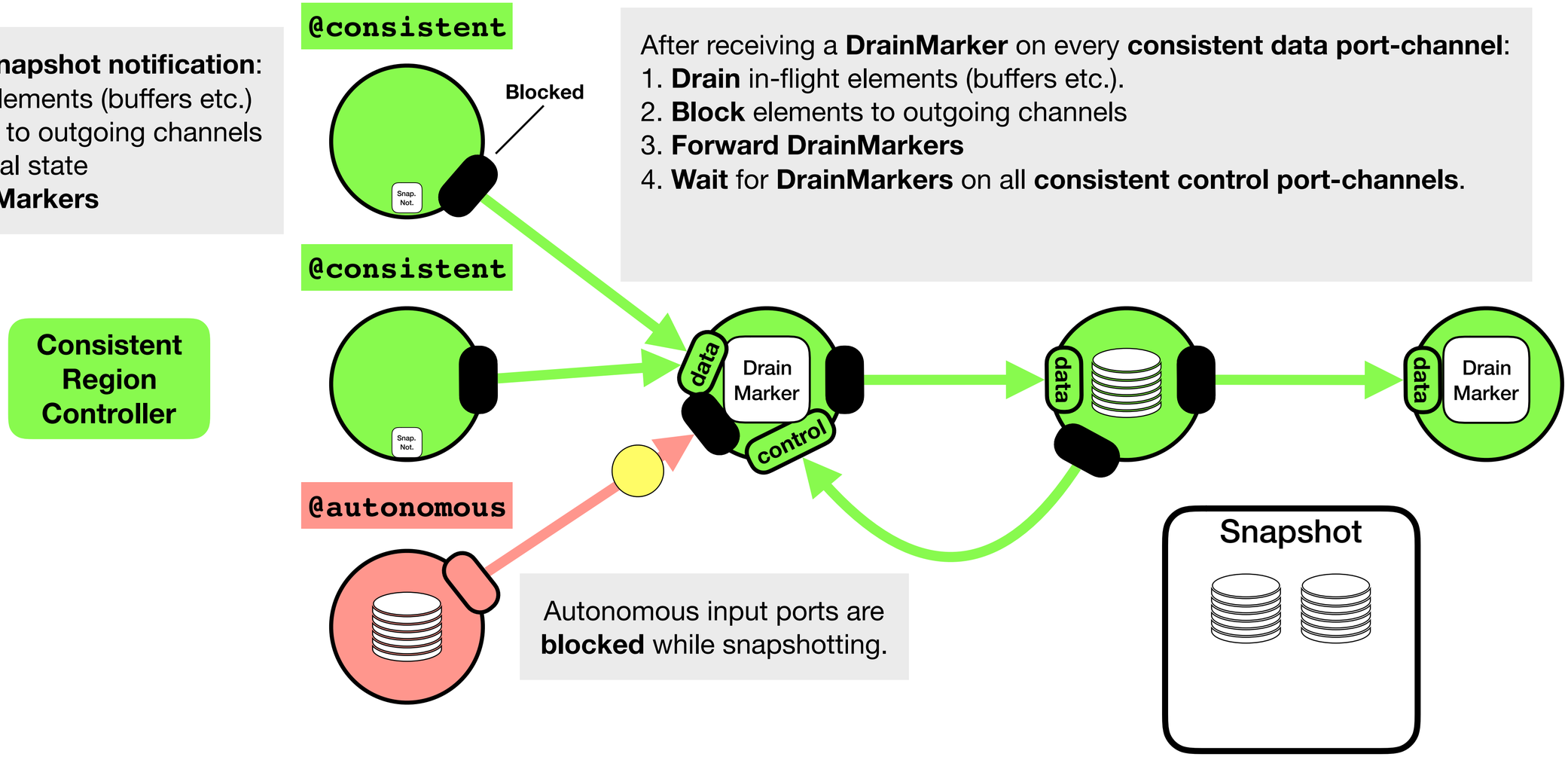


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

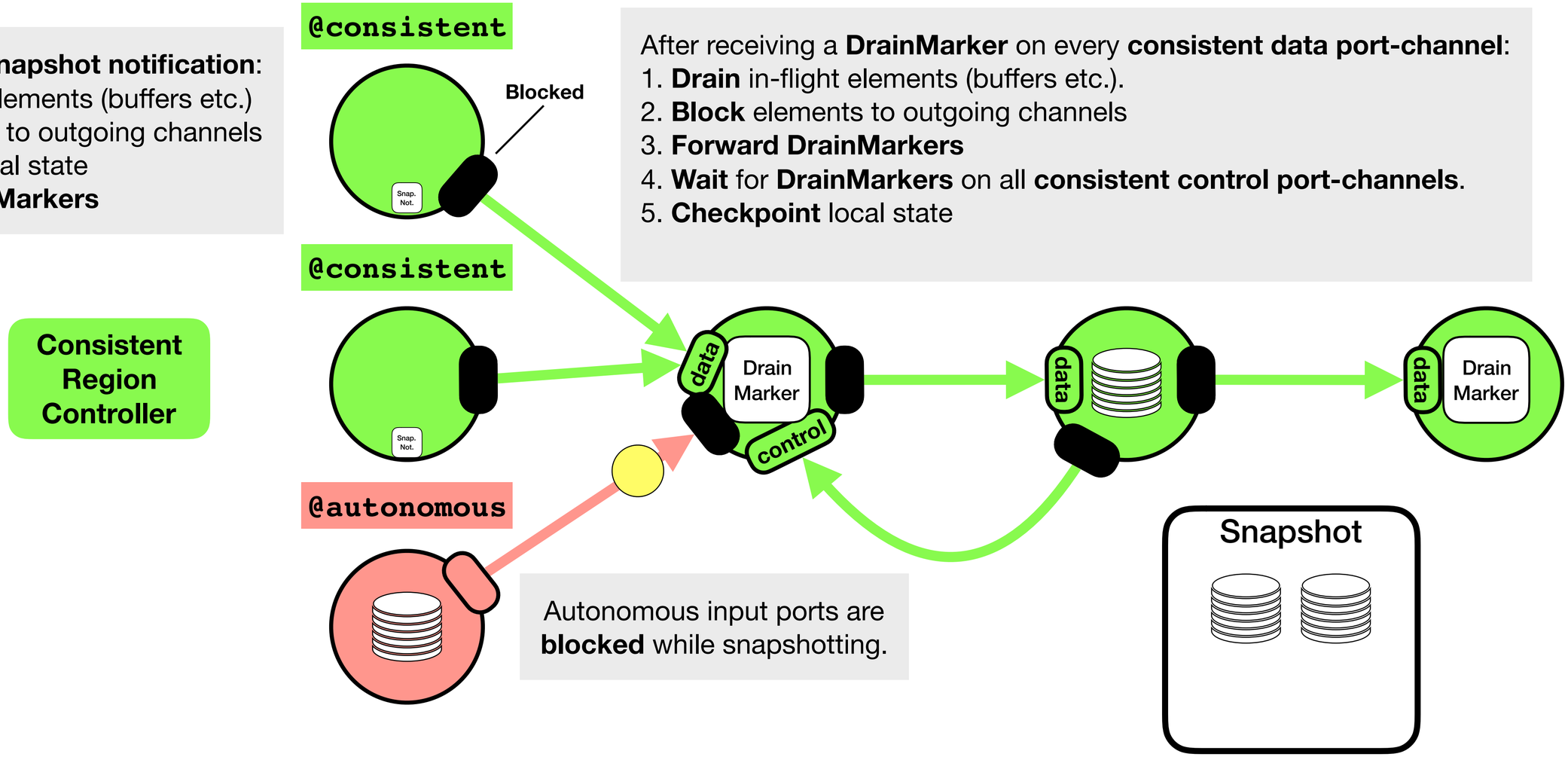


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

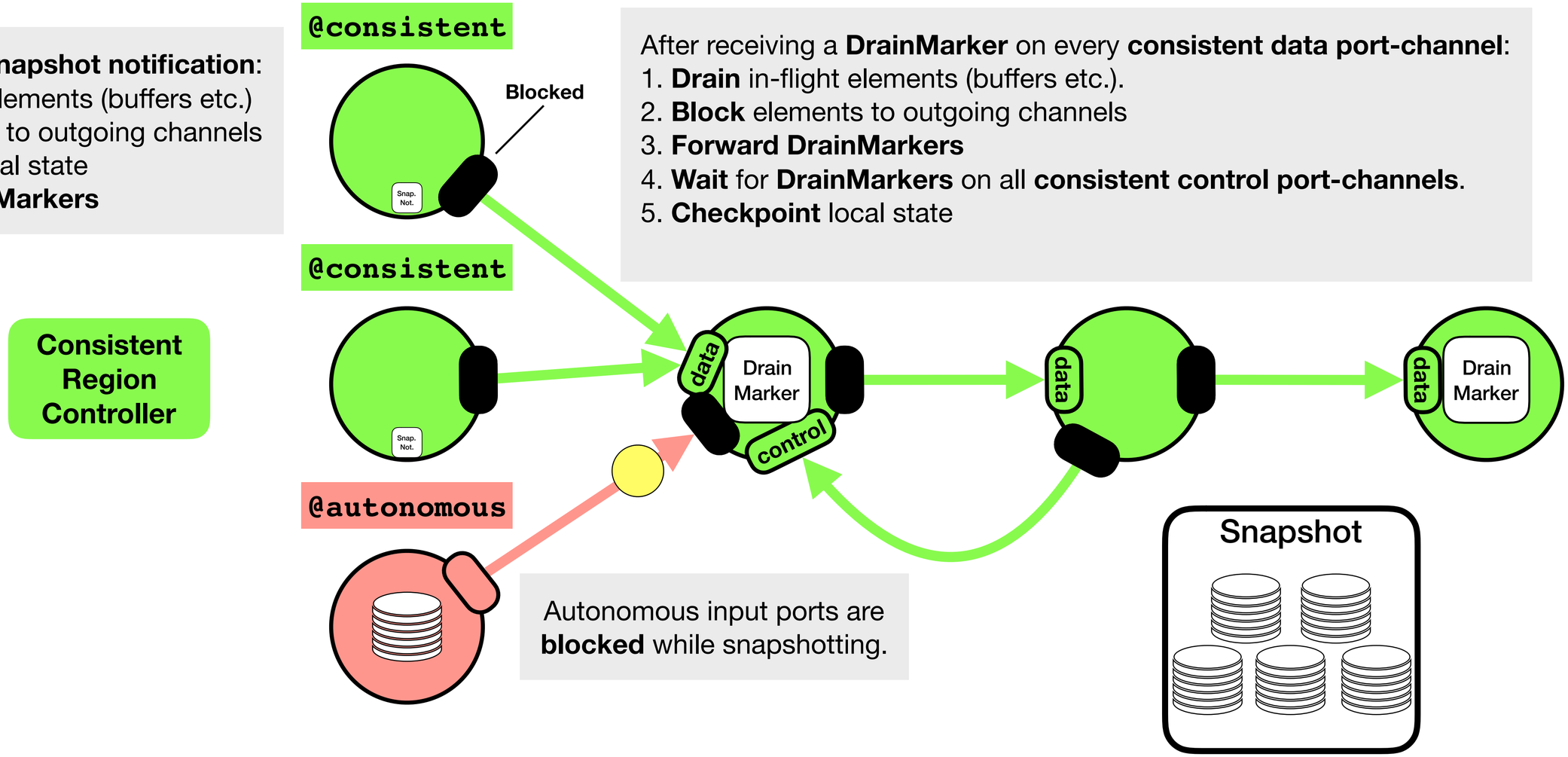


After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**



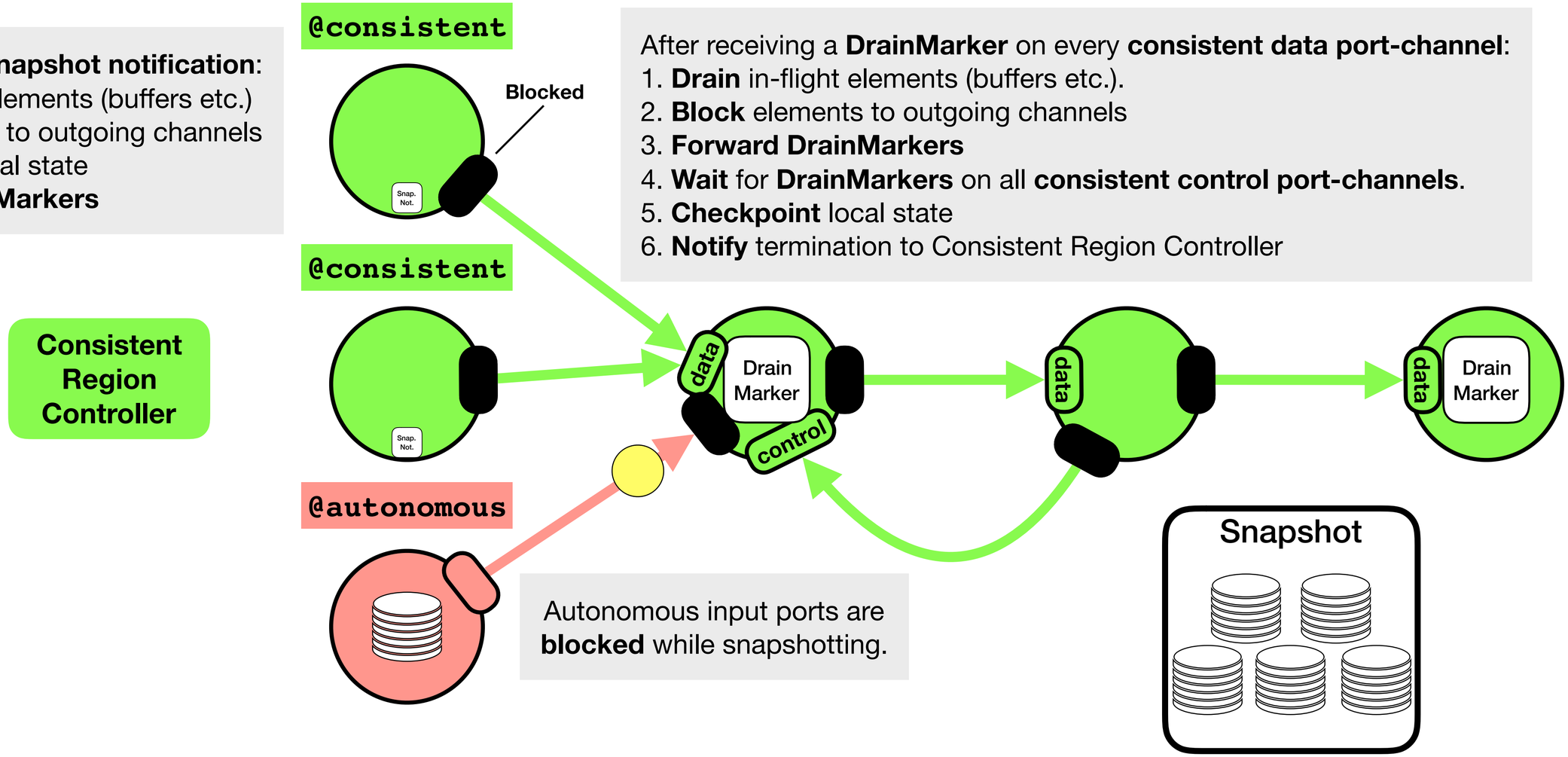
After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state

Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

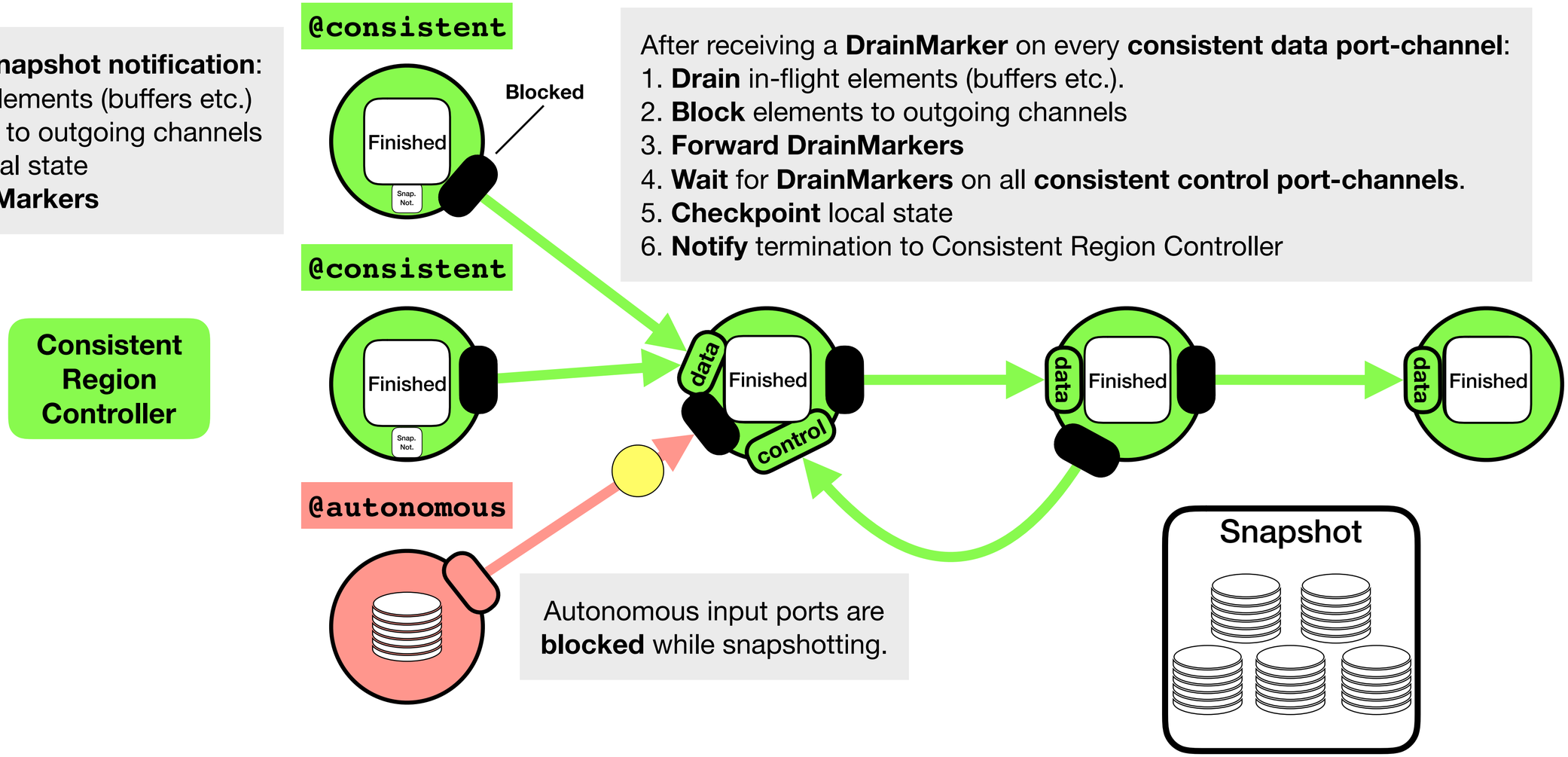
After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state
6. **Notify** termination to Consistent Region Controller



IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

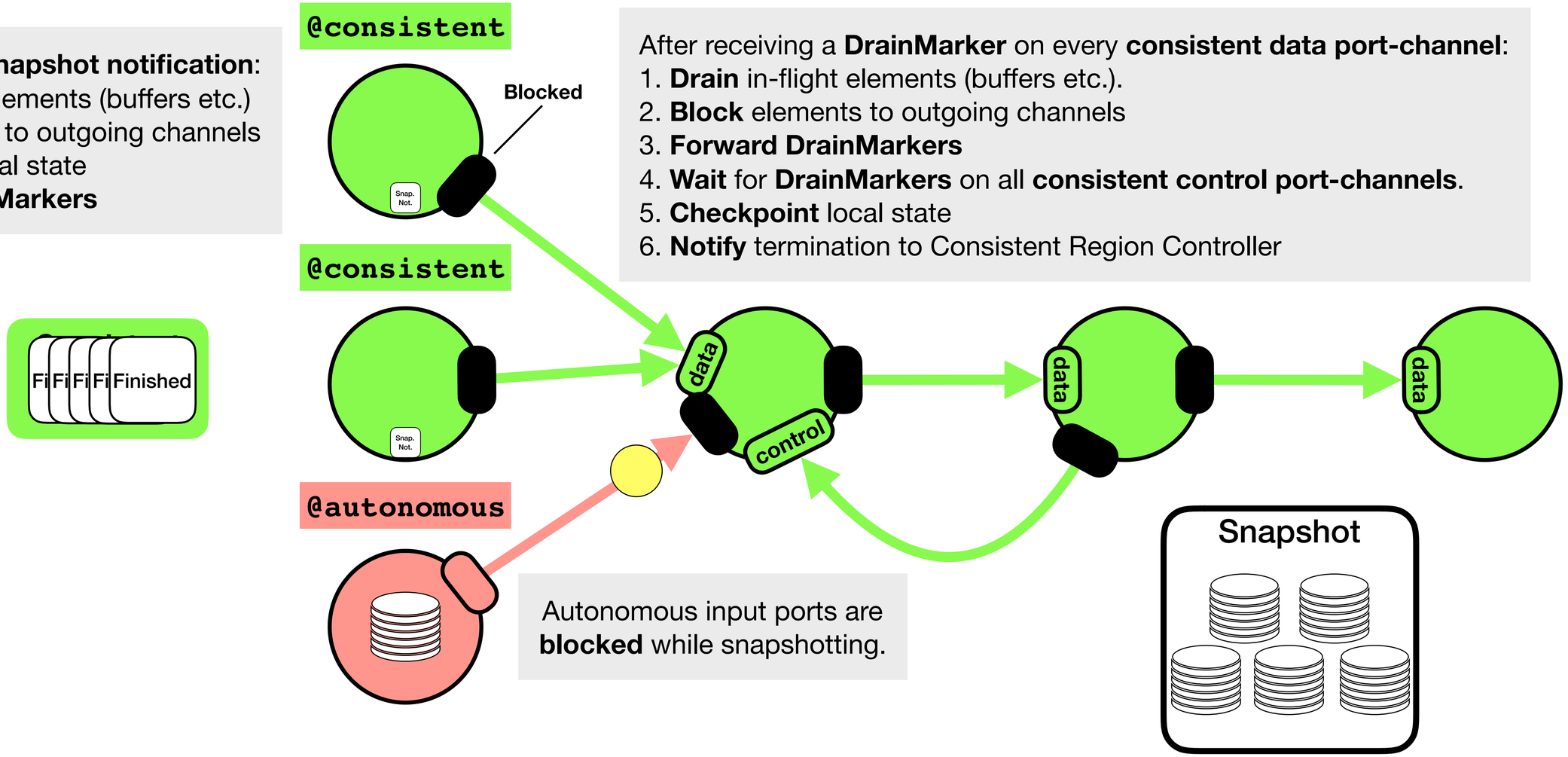
After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state
6. **Notify** termination to Consistent Region Controller



IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

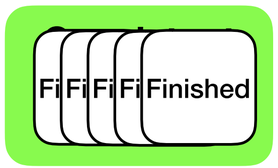
After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state
6. **Notify** termination to Consistent Region Controller



Autonomous input ports are **blocked** while snapshotting.

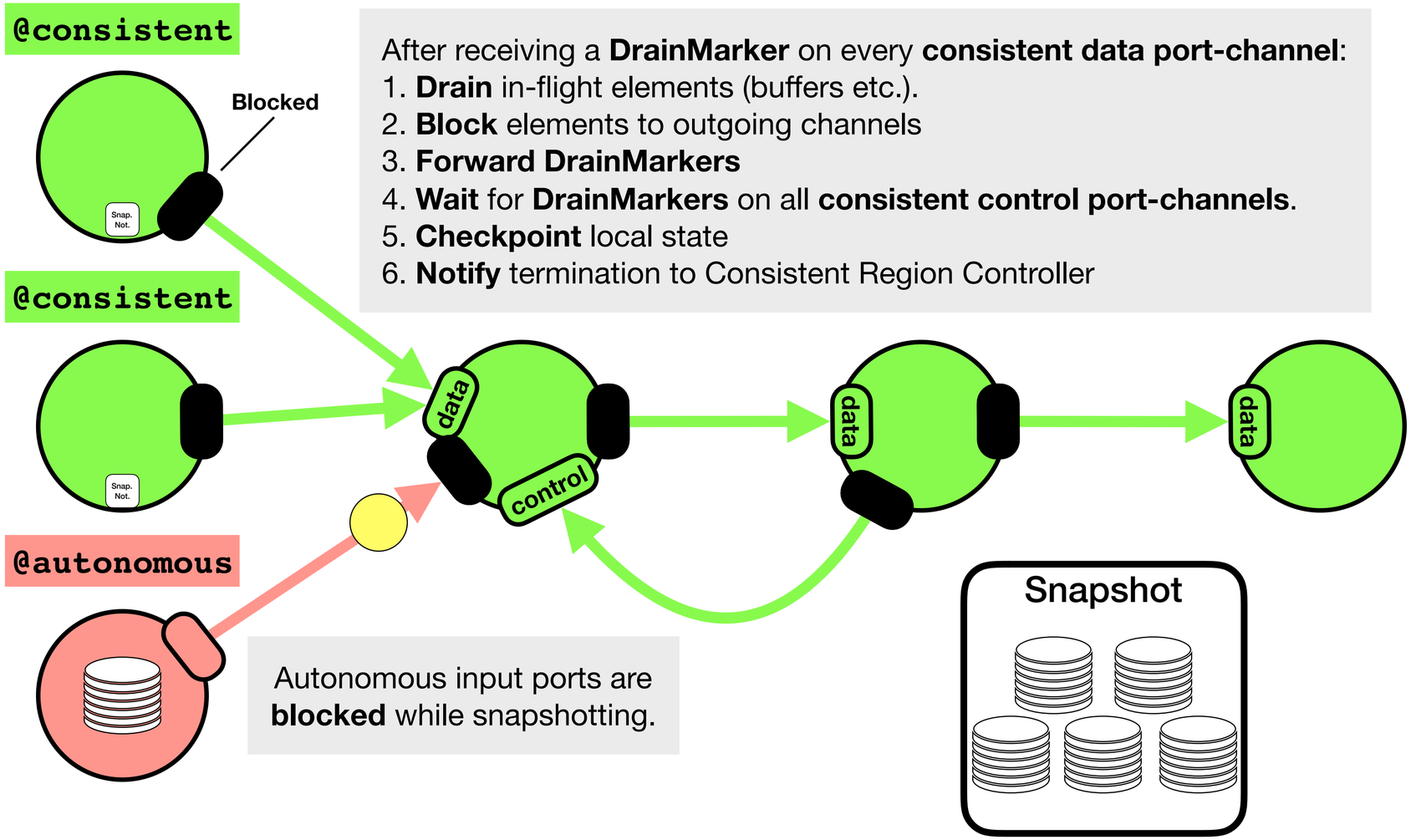
IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**



Finished

After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state
6. **Notify** termination to Consistent Region Controller

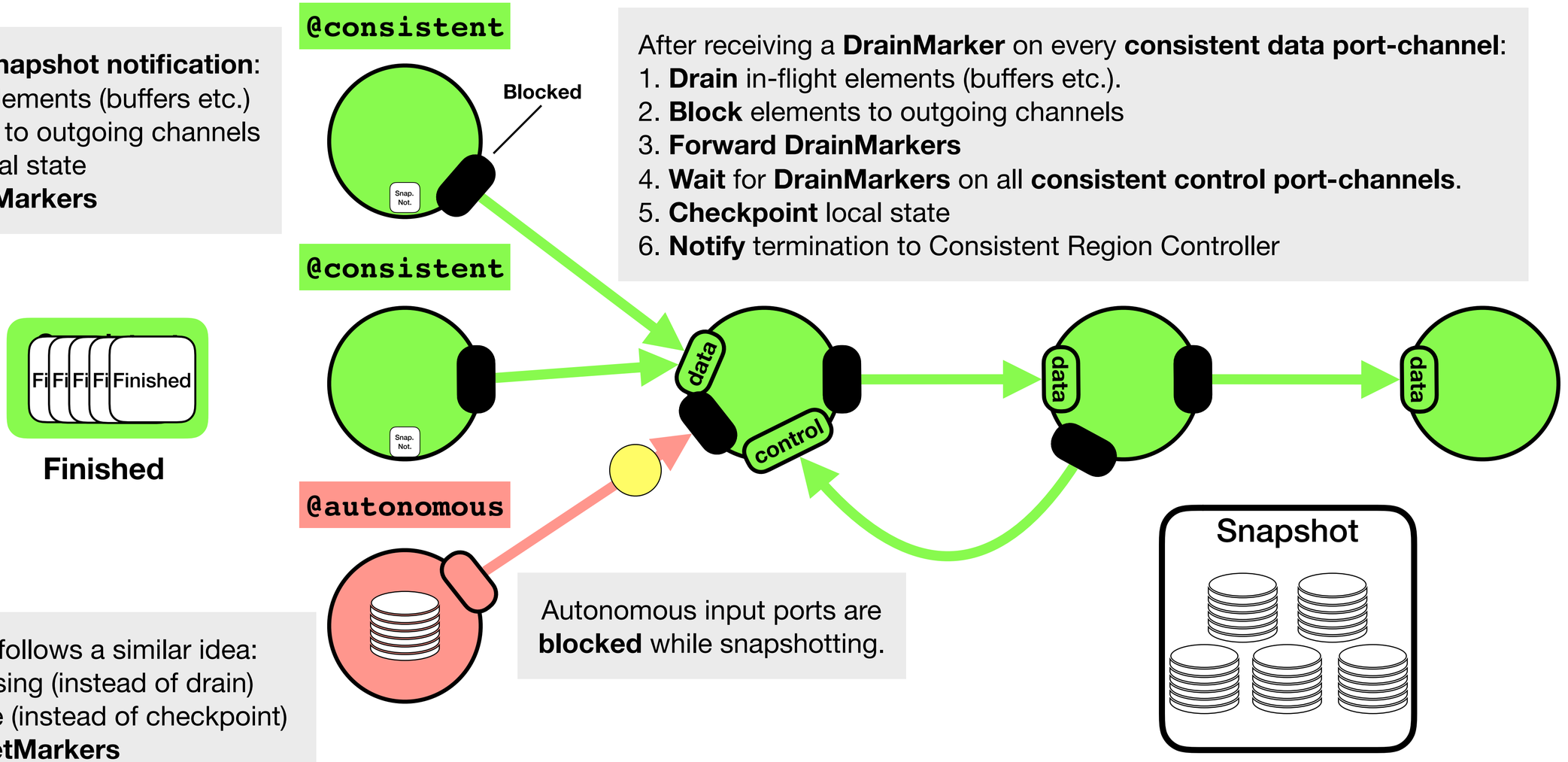


Autonomous input ports are **blocked** while snapshotting.

IBM Streams - Snapshotting Protocol

After receiving a **snapshot notification**:
1. **Drain** in-flight elements (buffers etc.)
2. **Block** elements to outgoing channels
3. **Checkpoint** local state
4. **Forward DrainMarkers**

After receiving a **DrainMarker** on every **consistent data port-channel**:
1. **Drain** in-flight elements (buffers etc.).
2. **Block** elements to outgoing channels
3. **Forward DrainMarkers**
4. **Wait** for **DrainMarkers** on all **consistent control port-channels**.
5. **Checkpoint** local state
6. **Notify** termination to Consistent Region Controller



Autonomous input ports are **blocked** while snapshotting.

Failure recovery follows a similar idea:
1. **Pause** processing (instead of drain)
2. **Rollback** state (instead of checkpoint)
3. **Forward ResetMarkers**

Finally, start **replaying** elements

IBM Streams - Discussion

IBM Streams - Discussion

Q: What are the **weaknesses** of the snapshotting protocol?

IBM Streams - Discussion

Q: What are the **weaknesses** of the snapshotting protocol?

A: Dataflow is **blocked** until sinks receive drain markers on all channels.

IBM Streams - Discussion

Q: What are the **weaknesses** of the snapshotting protocol?

A: Dataflow is **blocked** until sinks receive drain markers on all channels.

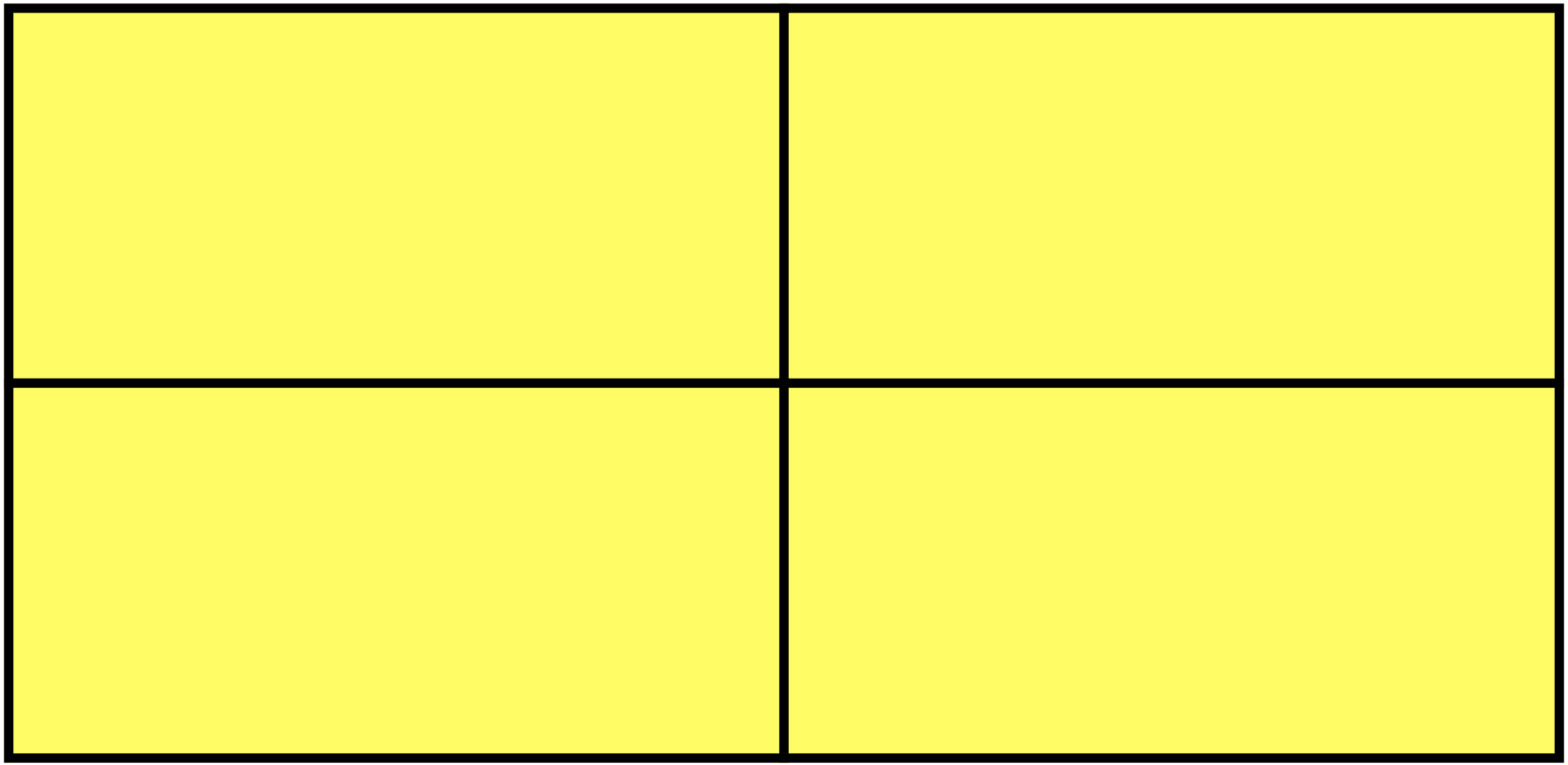
A: State checkpoints are currently **synchronous**.

Apache Spark

Structured Streaming

(Version 2.3.0)

Spark Structured Streaming - Problem Statement



Spark Structured Streaming - Problem Statement

API Challenges

Reusing programs for batch data is often not possible

Users must write **incrementally executing** code: **What** vs **How**

Spark Structured Streaming - Problem Statement

API Challenges

Reusing programs for batch data is often not possible

Users must write **incrementally executing** code: **What** vs **How**

Integration Challenges

Streaming programs **don't run in isolation**

Must support **interactive queries** & joins with **batch data**

Spark Structured Streaming - Problem Statement

API Challenges

Reusing programs for batch data is often not possible

Users must write **incrementally executing** code: **What** vs **How**

Integration Challenges

Streaming programs **don't run in isolation**

Must support **interactive queries** & joins with **batch data**

Operational Challenges

Streaming systems are **distributed**

Must handle **failures**, **migration**, **graceful shutdown**, **code updates**, **stragglers**, **monitoring,** ...

Spark Structured Streaming - Problem Statement

API Challenges

Reusing programs for batch data is often not possible

Users must write **incrementally executing** code: **What** vs **How**

Integration Challenges

Streaming programs **don't run in isolation**

Must support **interactive queries** & joins with **batch data**

Operational Challenges

Streaming systems are **distributed**

Must handle **failures**, **migration**, **graceful shutdown**, **code updates**, **stragglers**, **monitoring,** ...

Cost & Performance Challenges

Streaming systems run **24/7**

Must support **dynamic rescaling**

Favour **throughput** over latency

Spark Structured Streaming - Problem Statement

API Challenges

Reusing programs for batch data is often not possible

Users must write **incrementally executing** code: **What** vs **How**

Integration Challenges

Streaming programs **don't run in isolation**

Must support **interactive queries** & joins with **batch data**

Solution: Unified API & Runtime for Stream & Batch

Operational Challenges

Streaming systems are **distributed**

Must handle **failures**, **migration**, **graceful shutdown**, **code updates**, **stragglers**, **monitoring,** ...

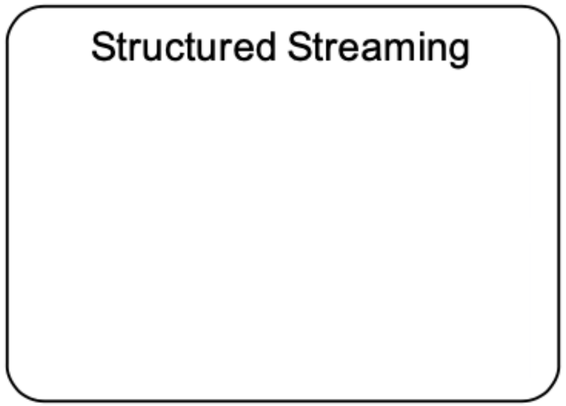
Cost & Performance Challenges

Streaming systems run **24/7**

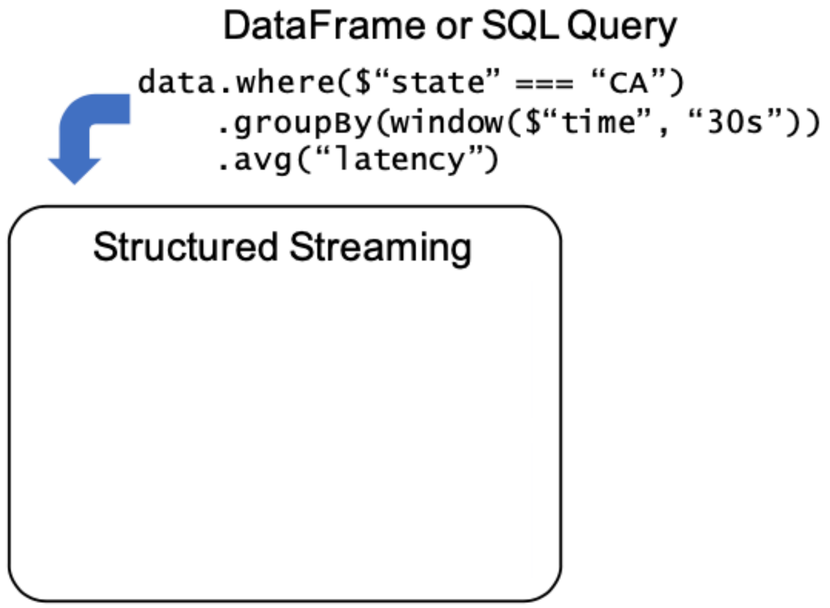
Must support **dynamic rescaling**

Favour **throughput** over latency

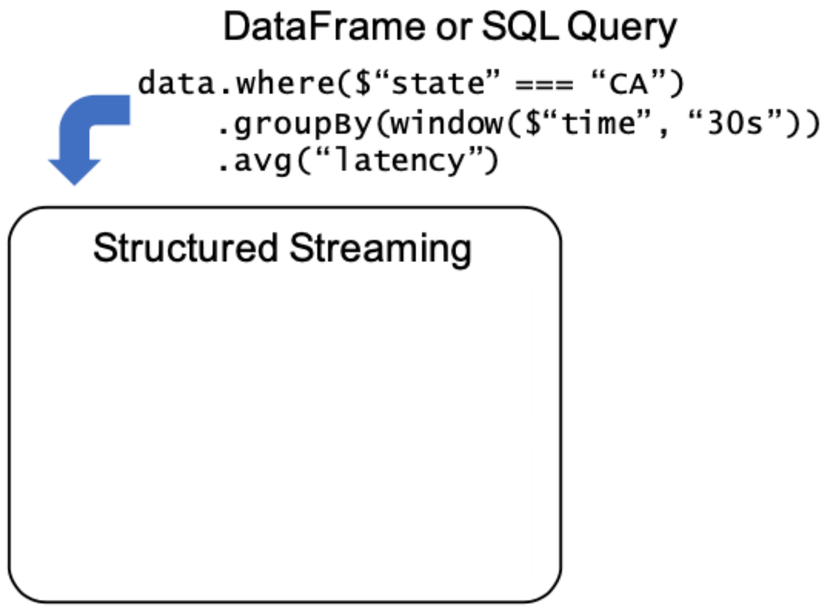
Spark Structured Streaming - System Overview



Spark Structured Streaming - System Overview

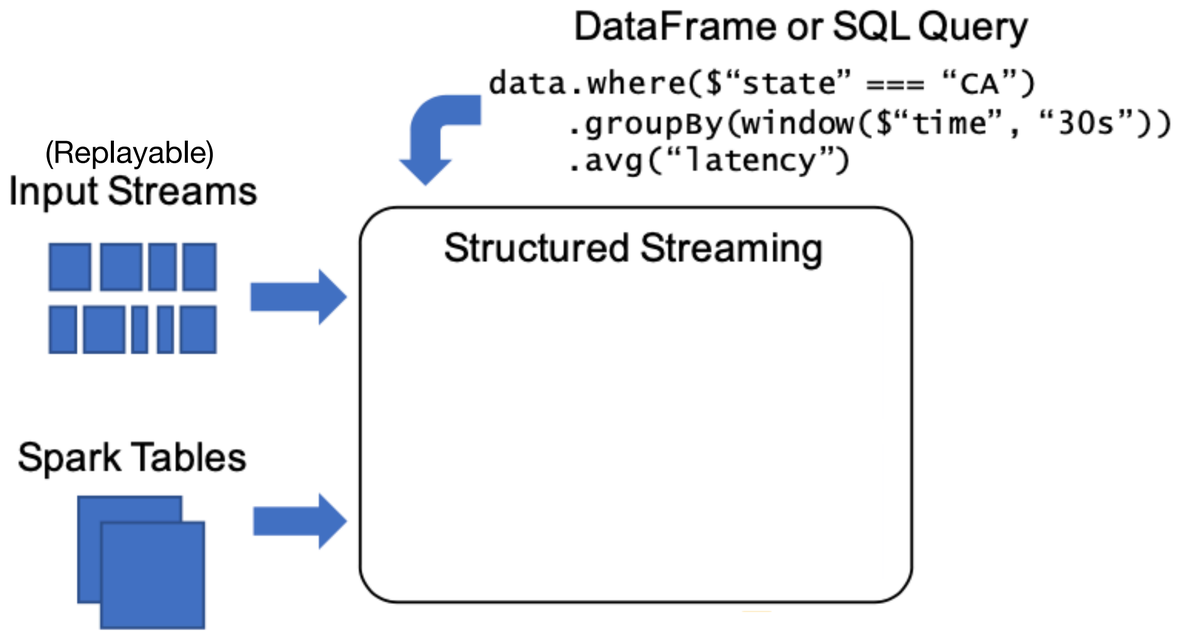


Spark Structured Streaming - System Overview



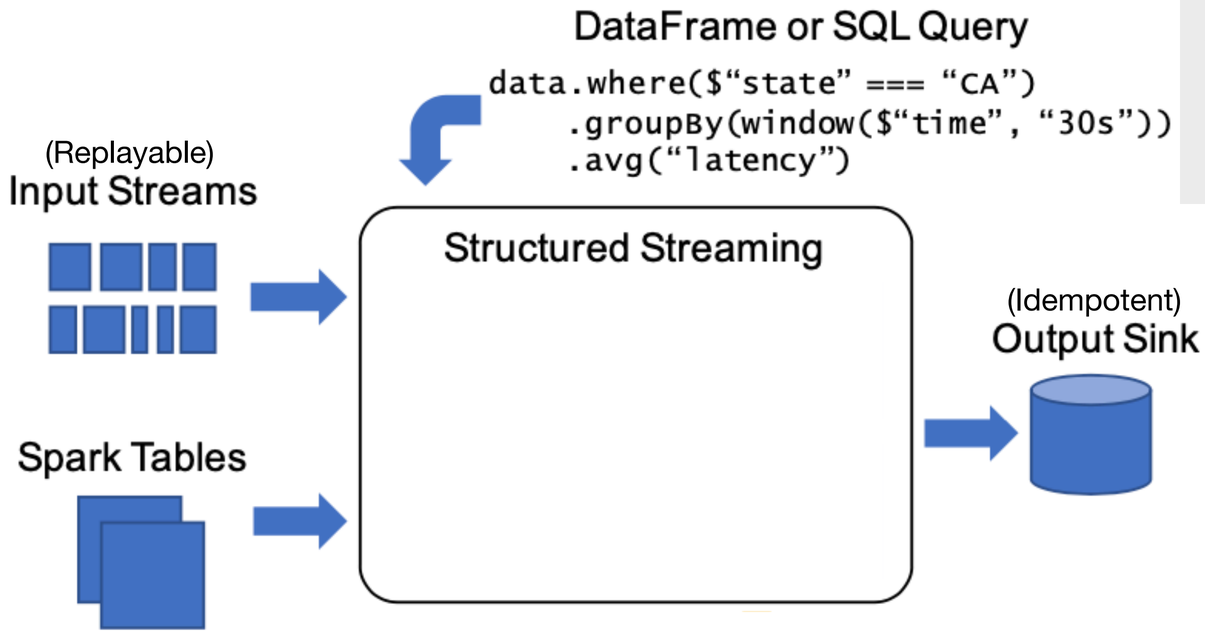
API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Spark Structured Streaming - System Overview



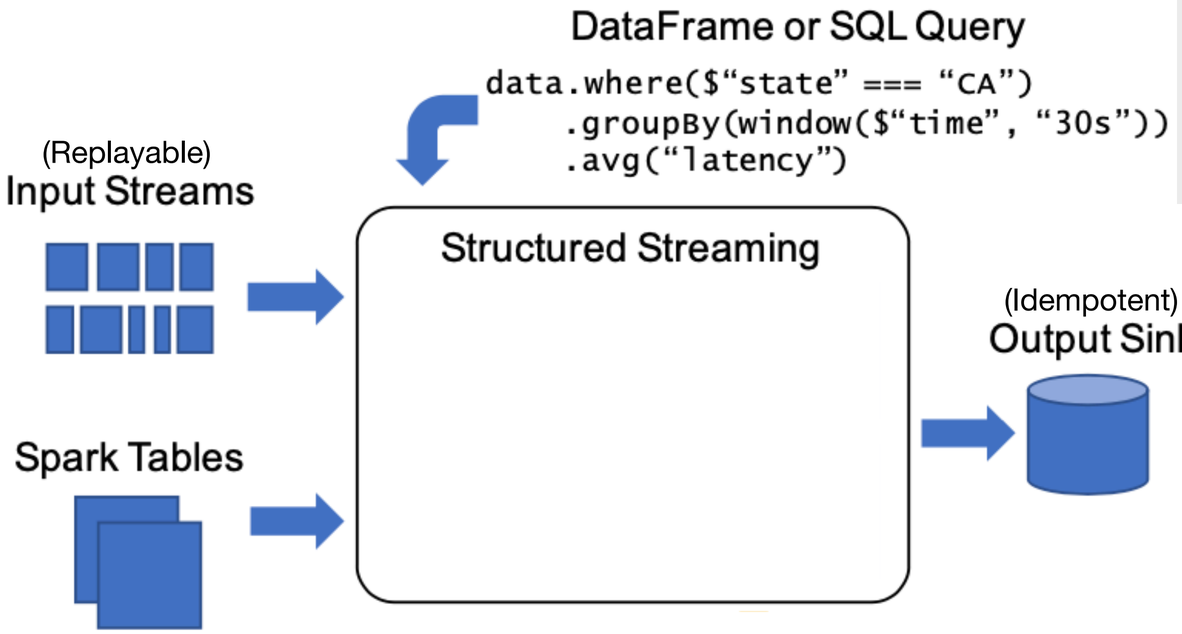
API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Spark Structured Streaming - System Overview



API additions
- Event-time
- Stateful operators
- Triggers (When to output)

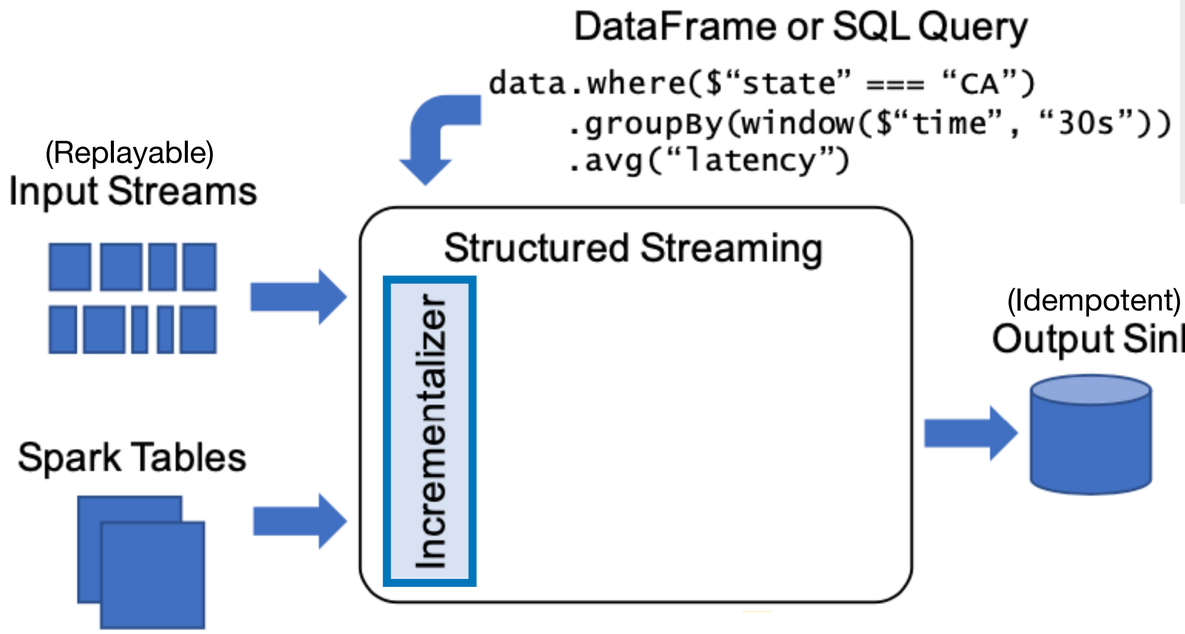
Spark Structured Streaming - System Overview



API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Output modes
(What to output)
- Complete
- Append
- Update

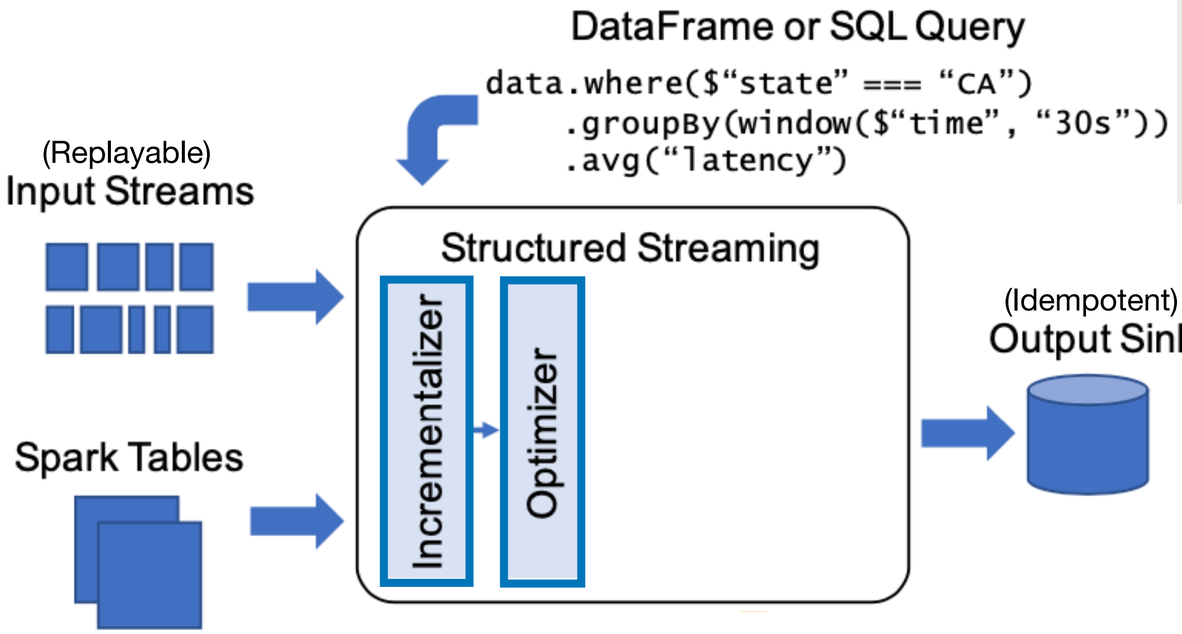
Spark Structured Streaming - System Overview



API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Output modes
(What to output)
- Complete
- Append
- Update

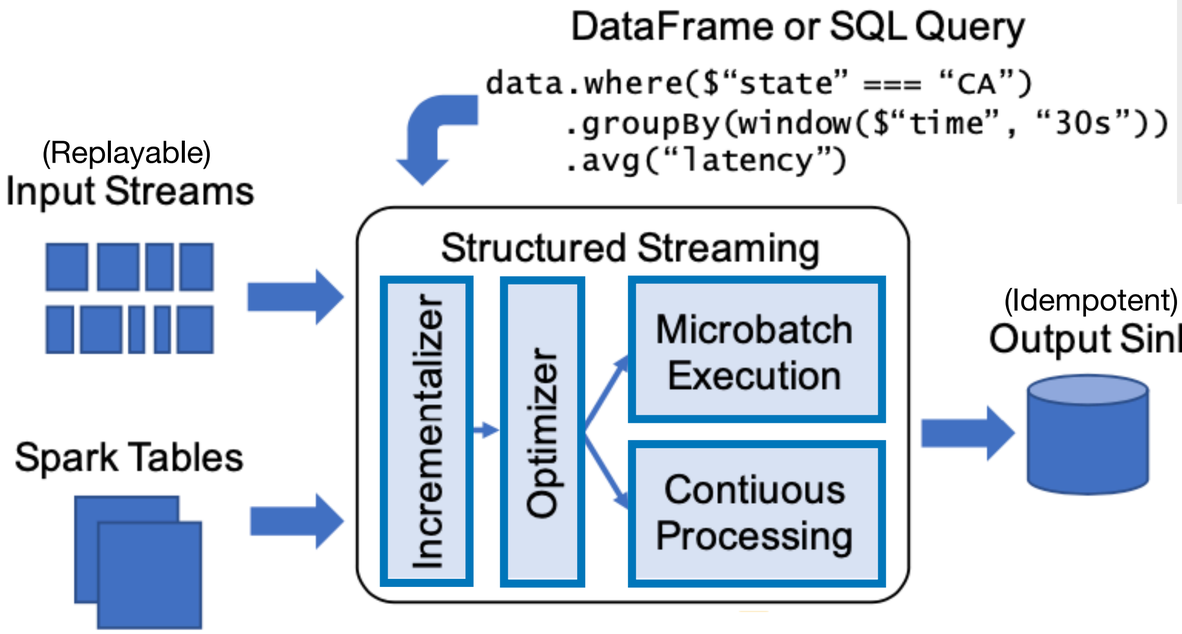
Spark Structured Streaming - System Overview



API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Output modes
(What to output)
- Complete
- Append
- Update

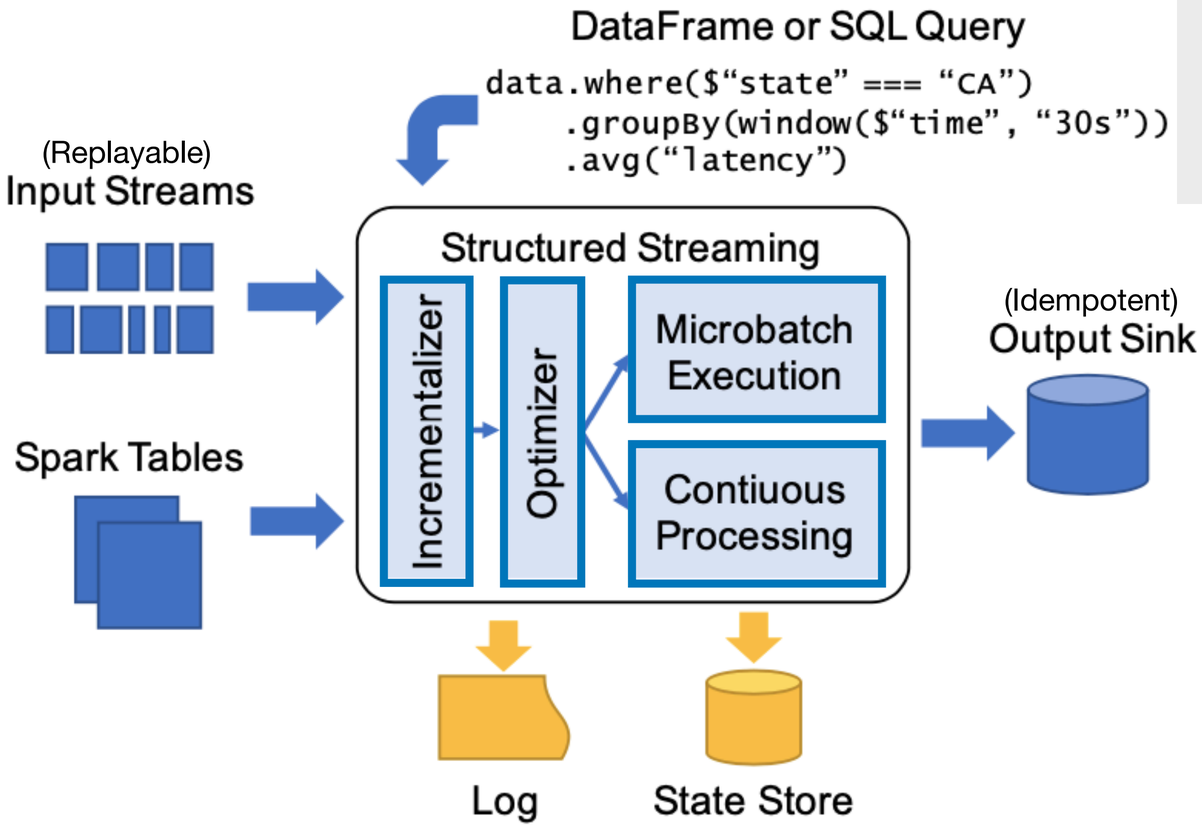
Spark Structured Streaming - System Overview



API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Output modes
(What to output)
- Complete
- Append
- Update

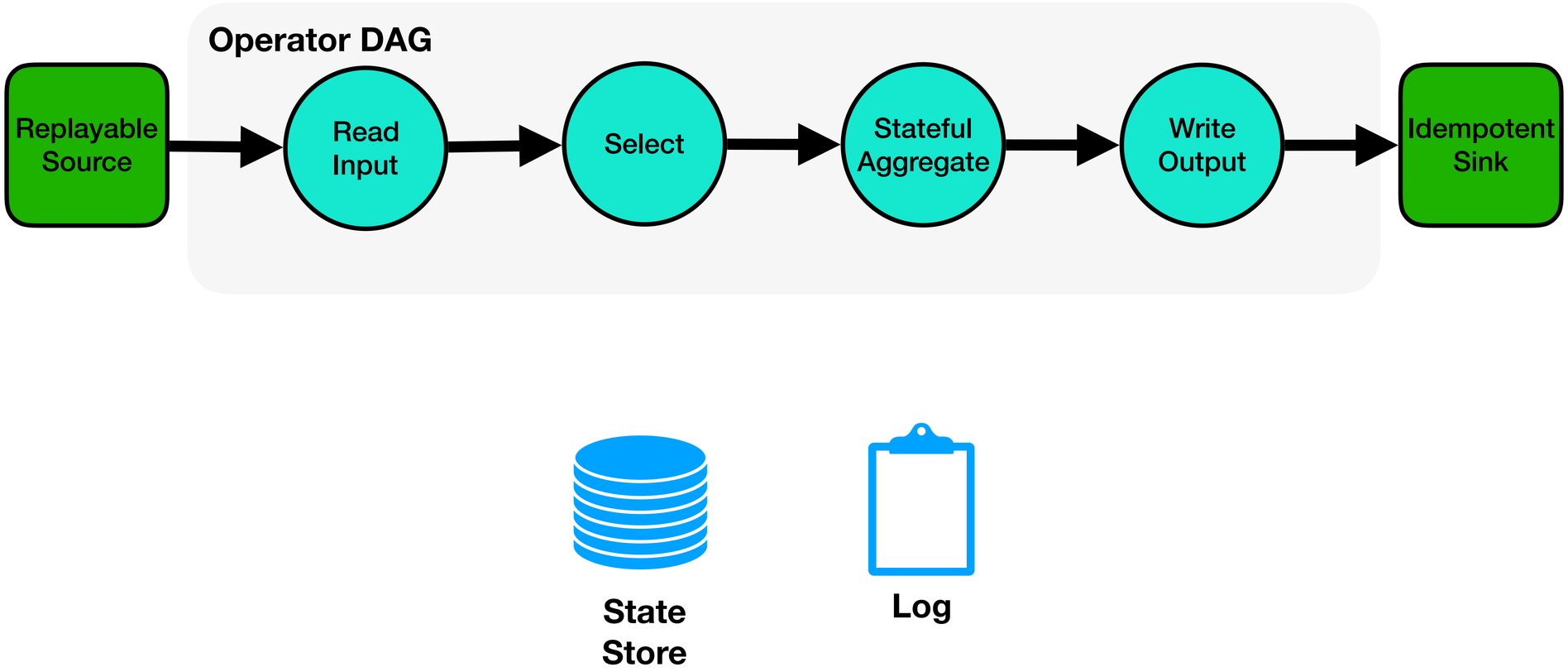
Spark Structured Streaming - System Overview



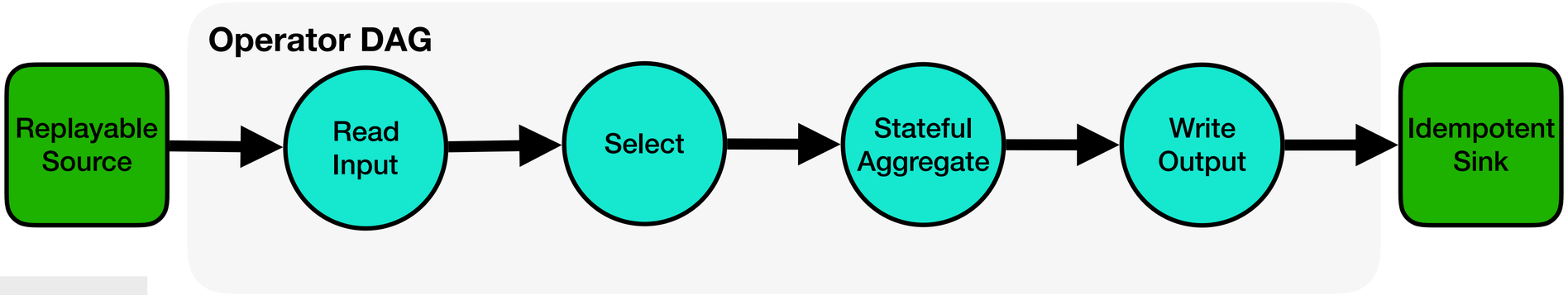
API additions
- Event-time
- Stateful operators
- Triggers (When to output)

Output modes
(What to output)
- Complete
- Append
- Update

Spark Structured Streaming - Checkpointing

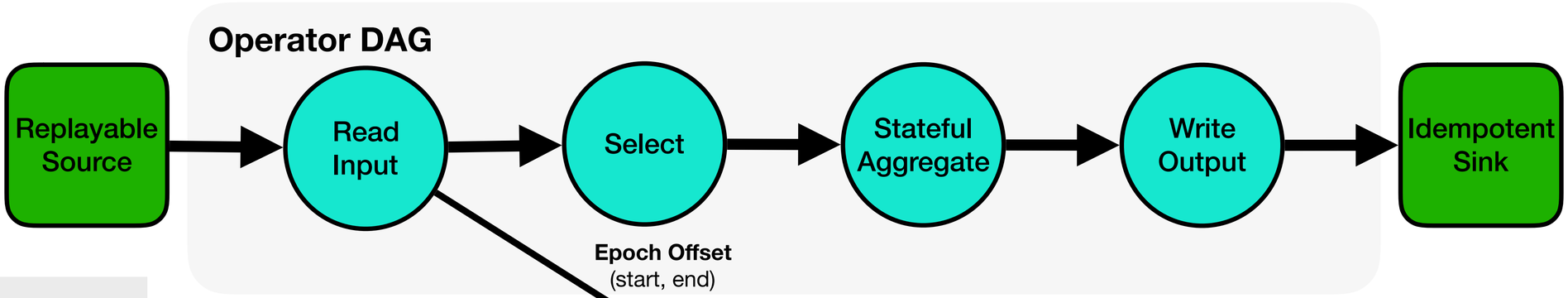


Spark Structured Streaming - Checkpointing

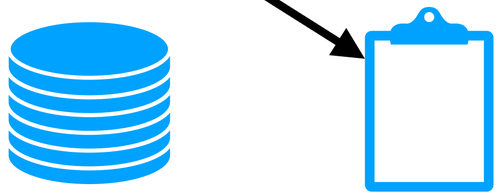


Spark's master node divides the input into **epochs**

Spark Structured Streaming - Checkpointing



Spark's master node divides the input into **epochs**

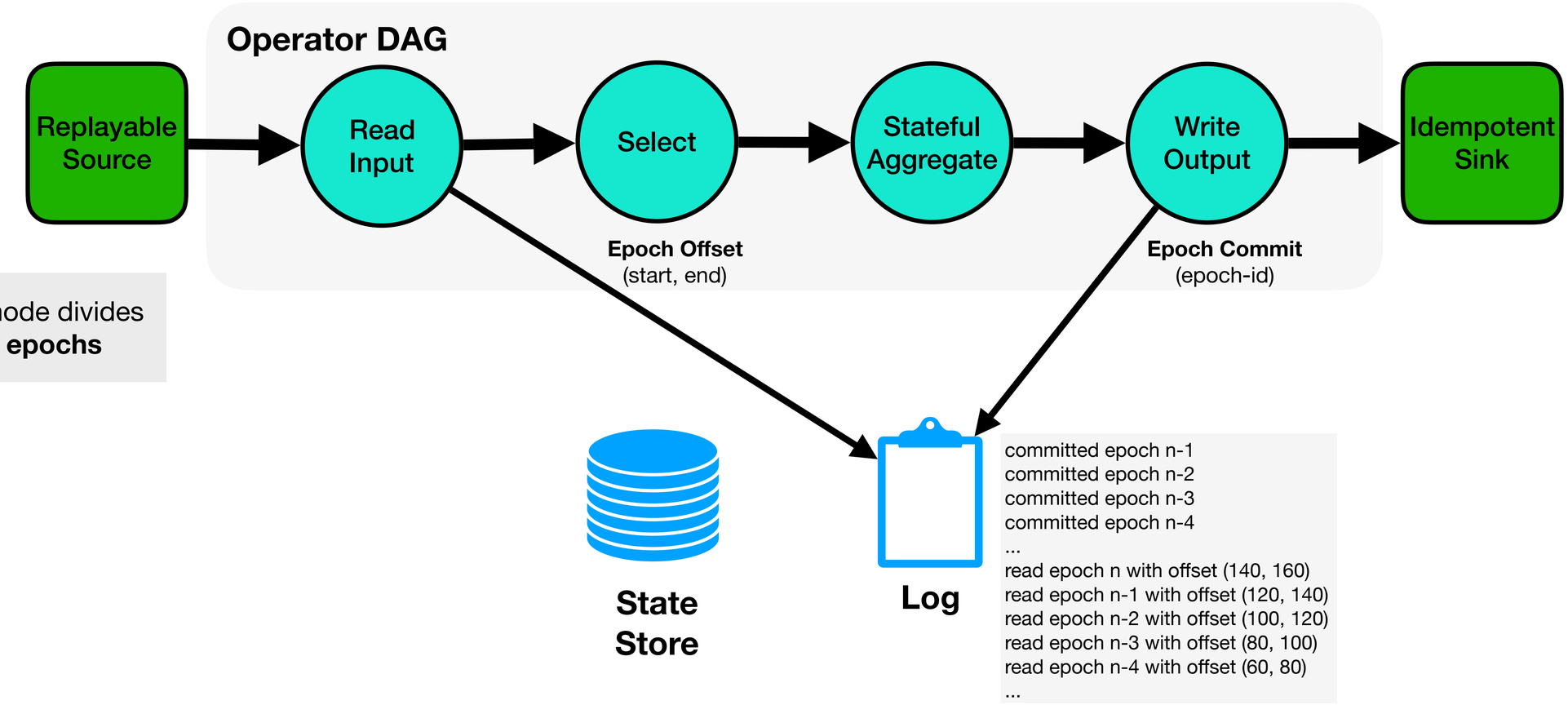


State Store

Log

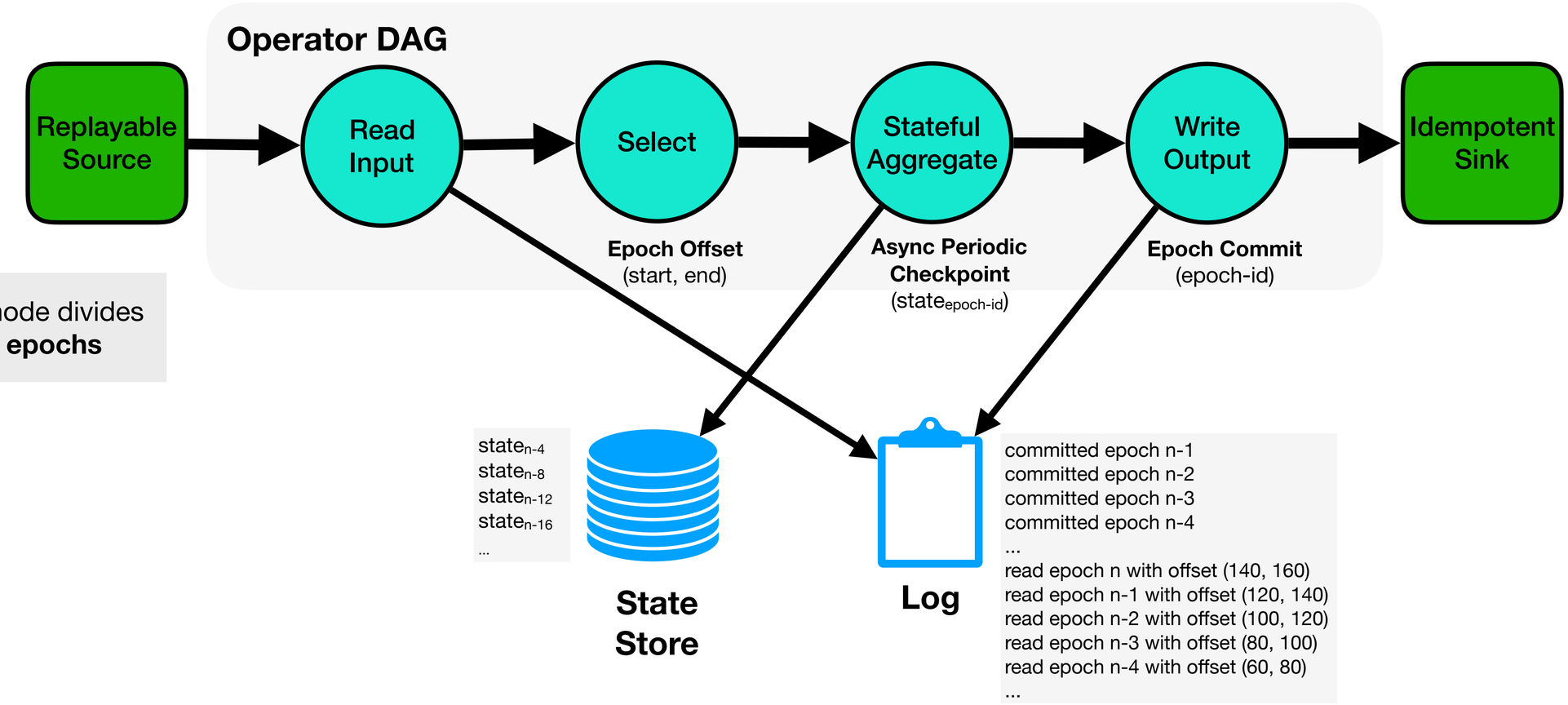
read epoch n with offset (140, 160)
read epoch n-1 with offset (120, 140)
read epoch n-2 with offset (100, 120)
read epoch n-3 with offset (80, 100)
read epoch n-4 with offset (60, 80)
...

Spark Structured Streaming - Checkpointing



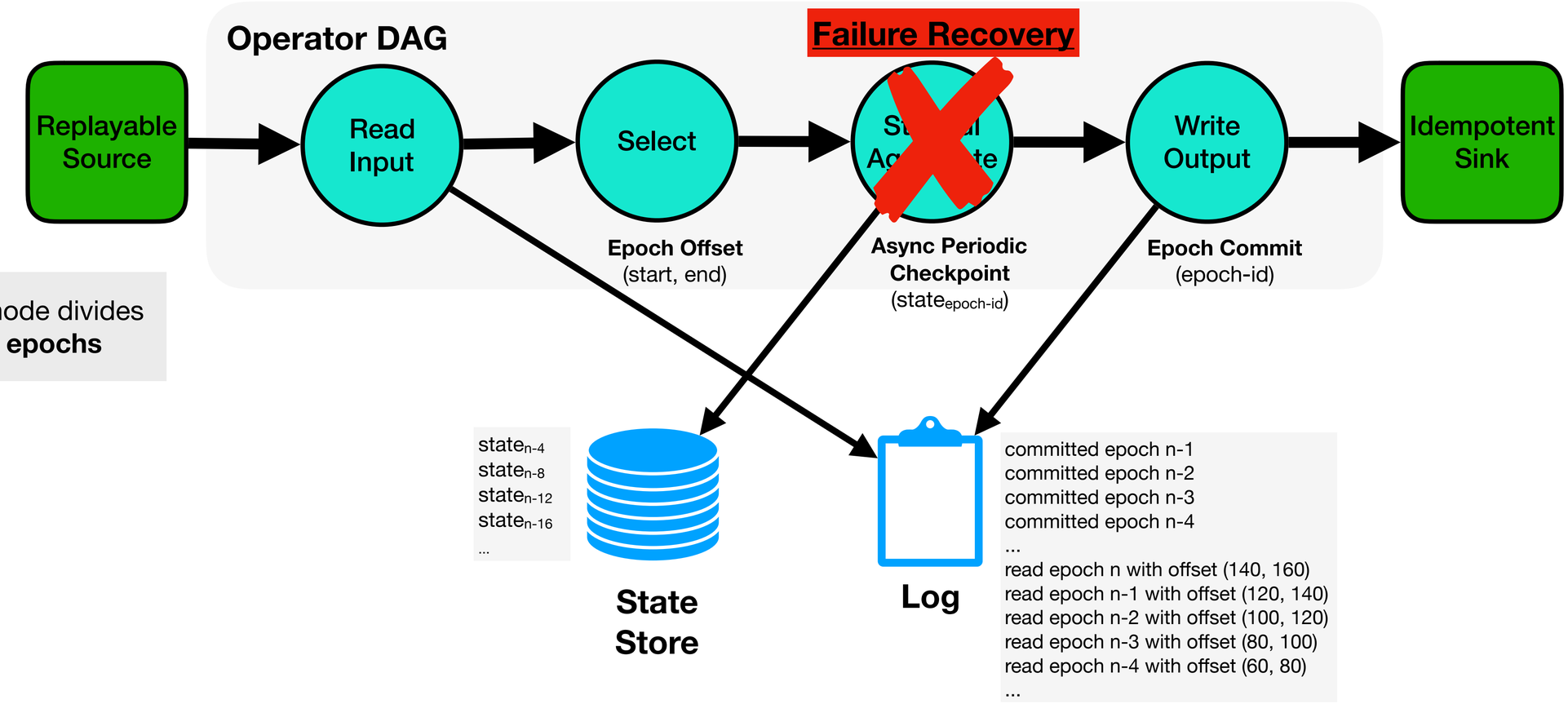
Spark's master node divides the input into **epochs**

Spark Structured Streaming - Checkpointing



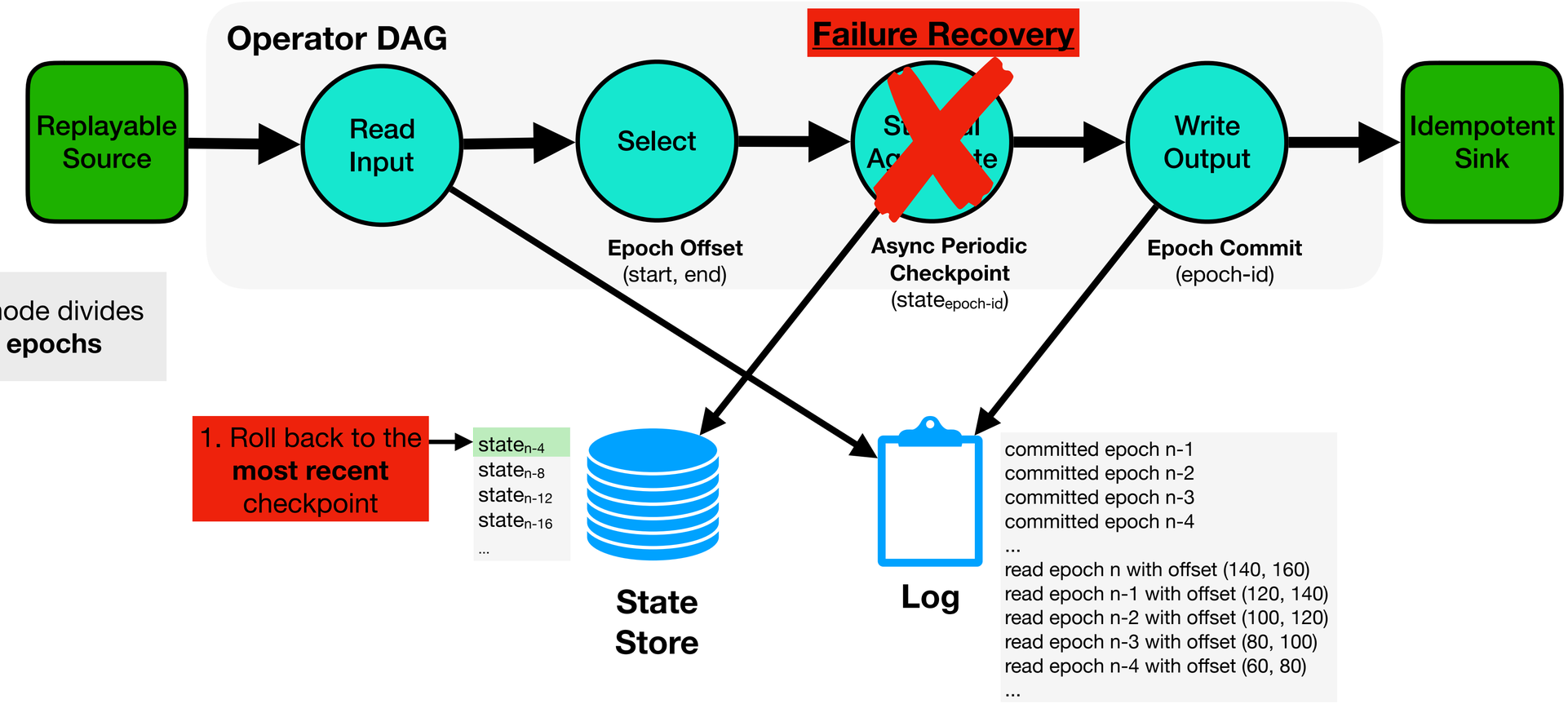
Spark's master node divides the input into **epochs**

Spark Structured Streaming - Checkpointing



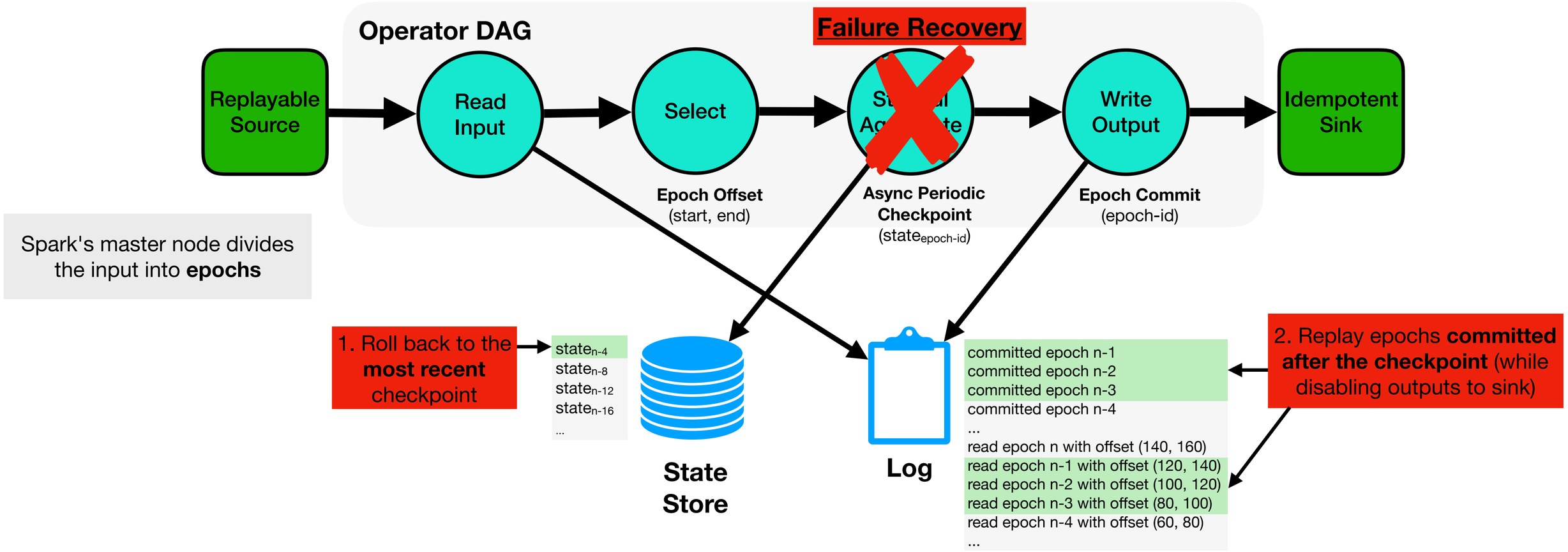
Spark's master node divides the input into **epochs**

Spark Structured Streaming - Checkpointing

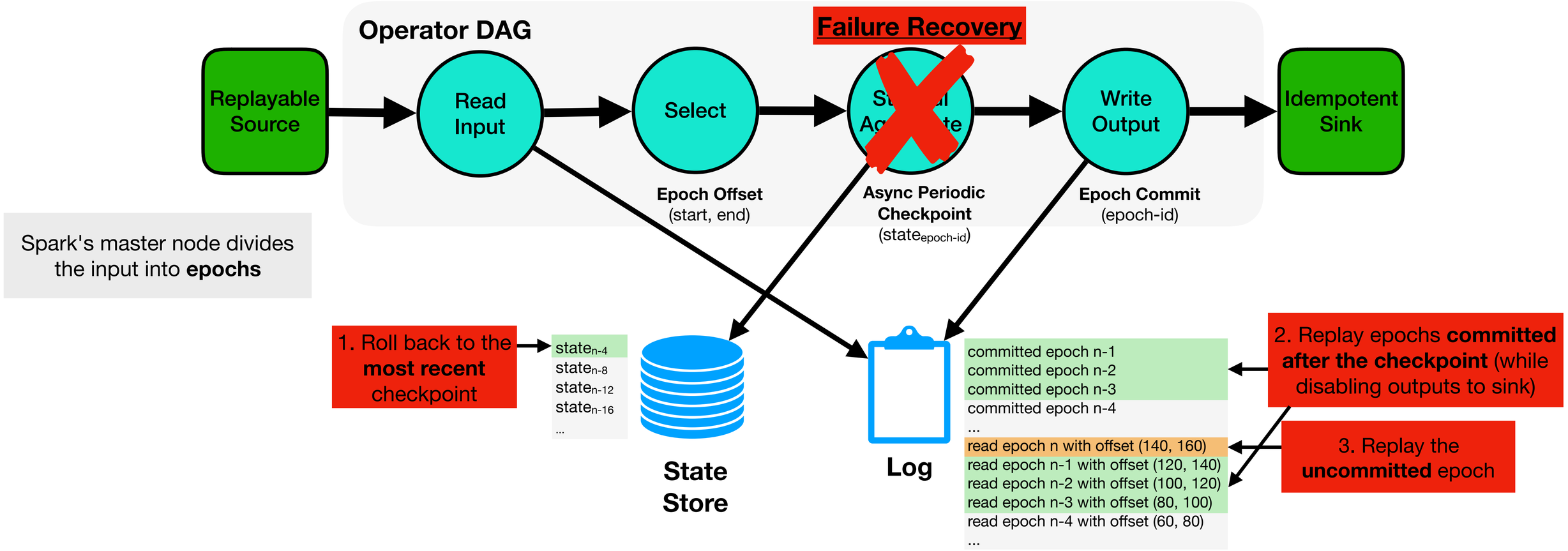


Spark's master node divides the input into **epochs**

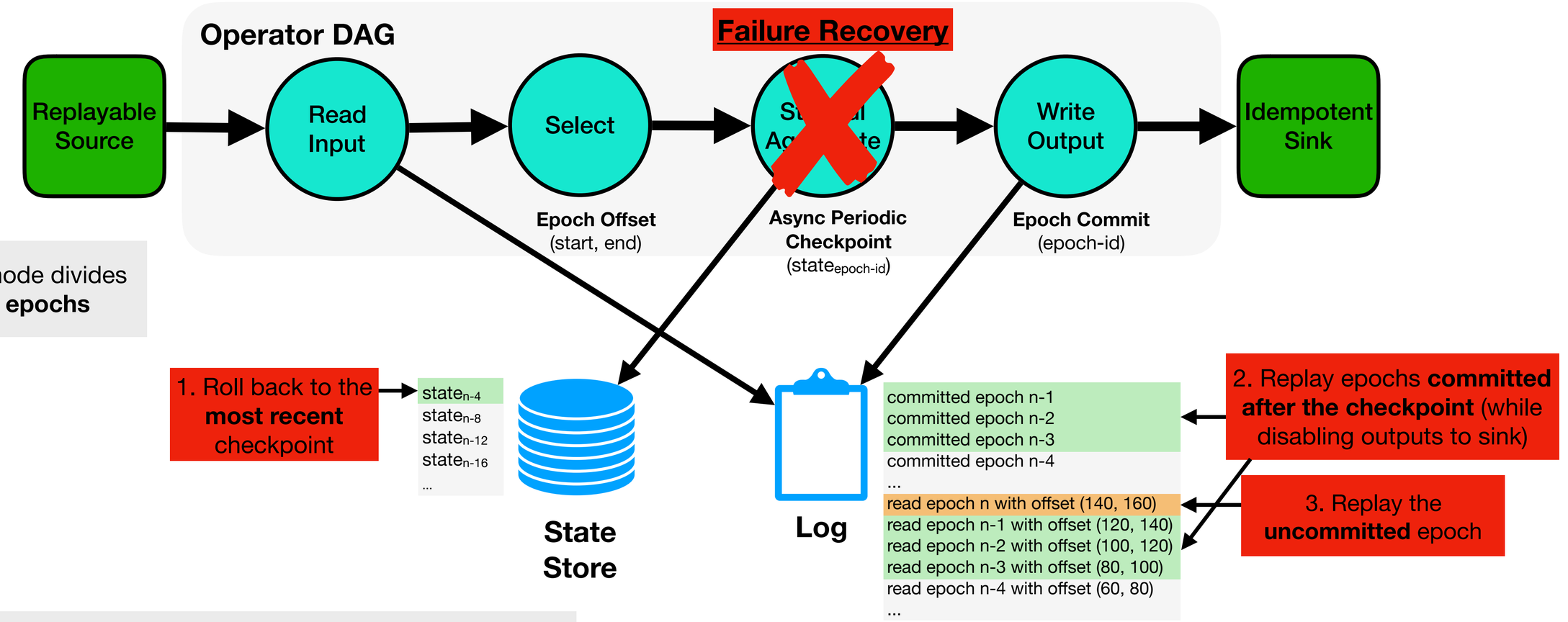
Spark Structured Streaming - Checkpointing



Spark Structured Streaming - Checkpointing



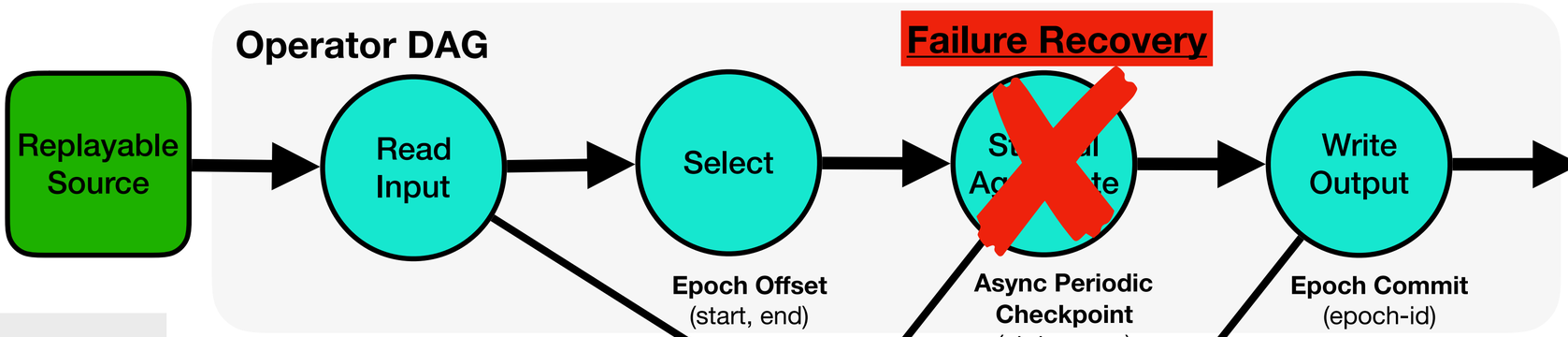
Spark Structured Streaming - Checkpointing



Spark's master node divides the input into **epochs**

Minibatch Execution:
Each **epoch** is executed by **multiple** (short-running) **tasks**

Spark Structured Streaming - Checkpointing



Spark's master node divides the input into **epochs**

1. Roll back to the **most recent** checkpoint

state$_{n-4}$
state$_{n-8}$
state$_{n-12}$
state$_{n-16}$
...

State Store

Log

committed epoch n-1
committed epoch n-2
committed epoch n-3
committed epoch n-4
...
read epoch n with offset (140, 160)
read epoch n-1 with offset (120, 140)
read epoch n-2 with offset (100, 120)
read epoch n-3 with offset (80, 100)
read epoch n-4 with offset (60, 80)
...

2. Replay epochs **committed after the checkpoint** (while disabling outputs to sink)

3. Replay the **uncommitted** epoch

Minibatch Execution:
Each **epoch** is executed by **multiple** (short-running) **tasks**

Continuous Processing:
Each (long-running) **task** reads from **one partition** and executes **multiple epochs** (but no shuffles are allowed)

Spark Structured Streaming - Discussion

Q: What are the **benefits** of each execution mode?

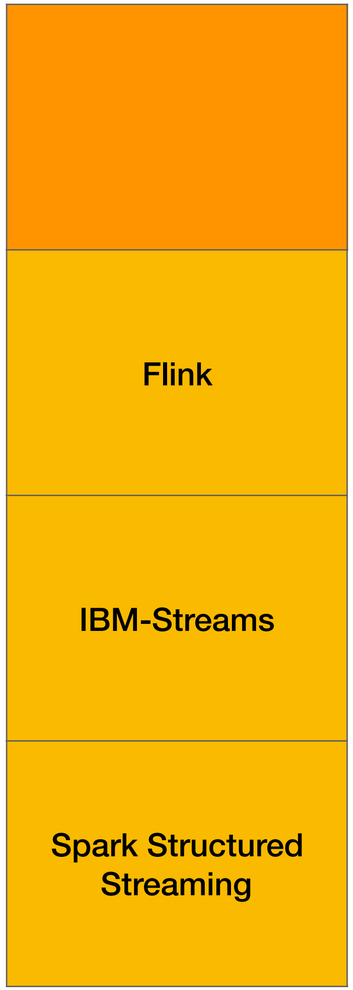
A: Microbatch Execution for **throughput** (load balancing).

A: Continuous Processing for **latency** (less scheduling).

Comparison & Conclusion

Comparison

Comparison



Comparison

	Execution Model?
Flink	Continuous
IBM-Streams	Continuous
Spark Structured Streaming	Continuous & Minibatch

Comparison

	Execution Model?	Queryable State?
Flink	Continuous	Yes
IBM-Streams	Continuous	N/A
Spark Structured Streaming	Continuous & Minibatch	Yes

Comparison

	Execution Model?	Queryable State?	Snapshot Granularity?
Flink	Continuous	Yes	Global
IBM-Streams	Continuous	N/A	Region
Spark Structured Streaming	Continuous & Minibatch	Yes	Global

Comparison

	Execution Model?	Queryable State?	Snapshot Granularity?	Cyclic Snapshots Method?
Flink	Continuous	Yes	Global	Chandy Lamport
IBM-Streams	Continuous	N/A	Region	Data & Control Ports (Only feedback loops)
Spark Structured Streaming	Continuous & Minibatch	Yes	Global	N/A

Comparison

	Execution Model?	Queryable State?	Snapshot Granularity?	Cyclic Snapshots Method?	Exactly-once Delivery Requirements?
Flink	Continuous	Yes	Global	Chandy Lamport	Idempotent or Transactional sinks
IBM-Streams	Continuous	N/A	Region	Data & Control Ports (Only feedback loops)	Retractable sinks
Spark Structured Streaming	Continuous & Minibatch	Yes	Global	N/A	Idempotent sinks

Comparison

	Execution Model?	Queryable State?	Snapshot Granularity?	Cyclic Snapshots Method?	Exactly-once Delivery Requirements?	Additional Properties?
Flink	Continuous	Yes	Global	Chandy Lamport	Idempotent or Transactional sinks	Incremental snapshot Async. compaction Channel blocking Channel recording
IBM-Streams	Continuous	N/A	Region	Data & Control Ports (Only feedback loops)	Retractable sinks	Incremental snapshot Channel blocking Sync. state transfer
Spark Structured Streaming	Continuous & Minibatch	Yes	Global	N/A	Idempotent sinks	Incremental snapshot Async. state transfer ???

Predictions / Future Work

Credits to Adam Hasselberg for predictions

Predictions / Future Work

Past

In-house Data center

Credits to Adam Hasselberg for predictions

Predictions / Future Work

Past	Present
In-house Data center 	Cloud platforms  

Credits to Adam Hasselberg for predictions

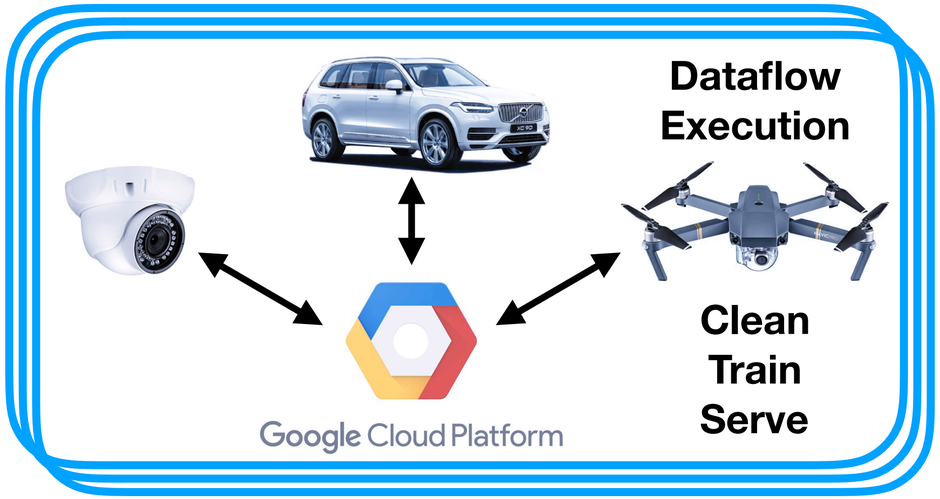
Predictions / Future Work

Past	Present	Future
In-house Data center 	Cloud platforms  	+ Edge devices   5G

Credits to Adam Hasselberg for predictions

Predictions / Future Work

Past	Present	Future
In-house Data center 	Cloud platforms  Google Cloud Platform 	+ Edge devices  kubernetes  5G



Credits to Adam Hasselberg for predictions

Predictions / Future Work

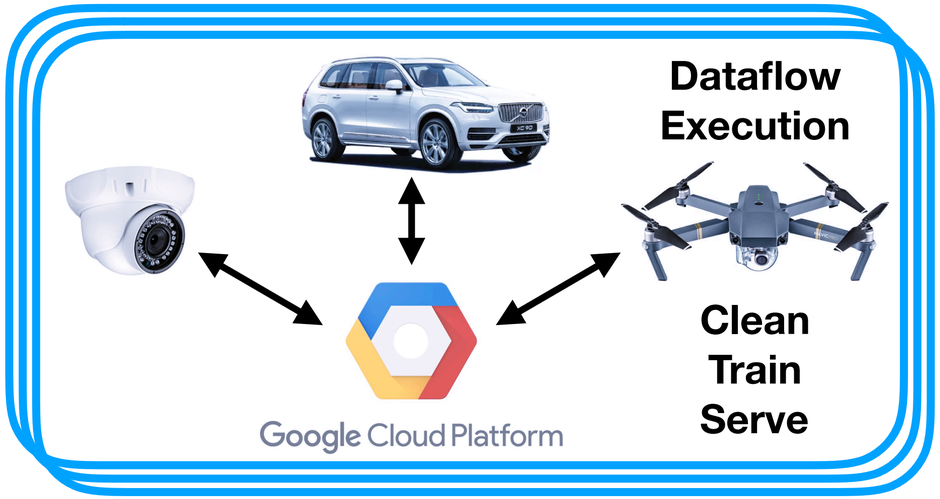
Past	Present	Future
In-house Data center	Cloud platforms	+ Edge devices

Google Cloud Platform

aws

kubernetes

5G



Future work for snapshotting protocols:

- Specialise towards throughput/latency/recovery time/space/...
- Take data into consideration
- Optimise code inside operators for snapshotting
- Hardware acceleration, e.g. Remote Direct Memory Access (RDMA)

Credits to Adam Hasselberg for predictions

END

BIN