Distributed Stream Processing: Substream Management and Fault Tolerance

Artem Trofimov, ML infrastructure lead @ Nebius

What is this talk about?

1. Stream processing basics
2. Unbounded stream problems
3. Failure-recovery problems
4. How to choose the right failure-recovery model?

Stream processing applications

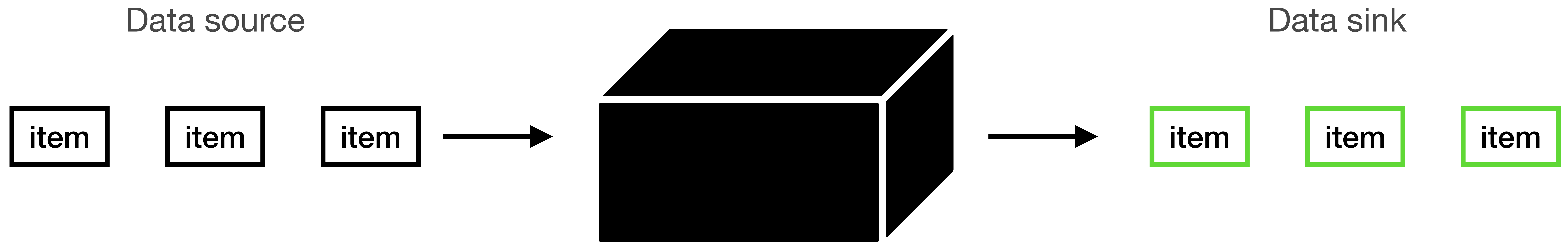
- Online-analytics
- Short-term personalization
- Online ML (training & inference)

Stream vs batch processing

- (Potentially) unbounded data
- (Potentially) unbounded computations
- Strong latency requirements*

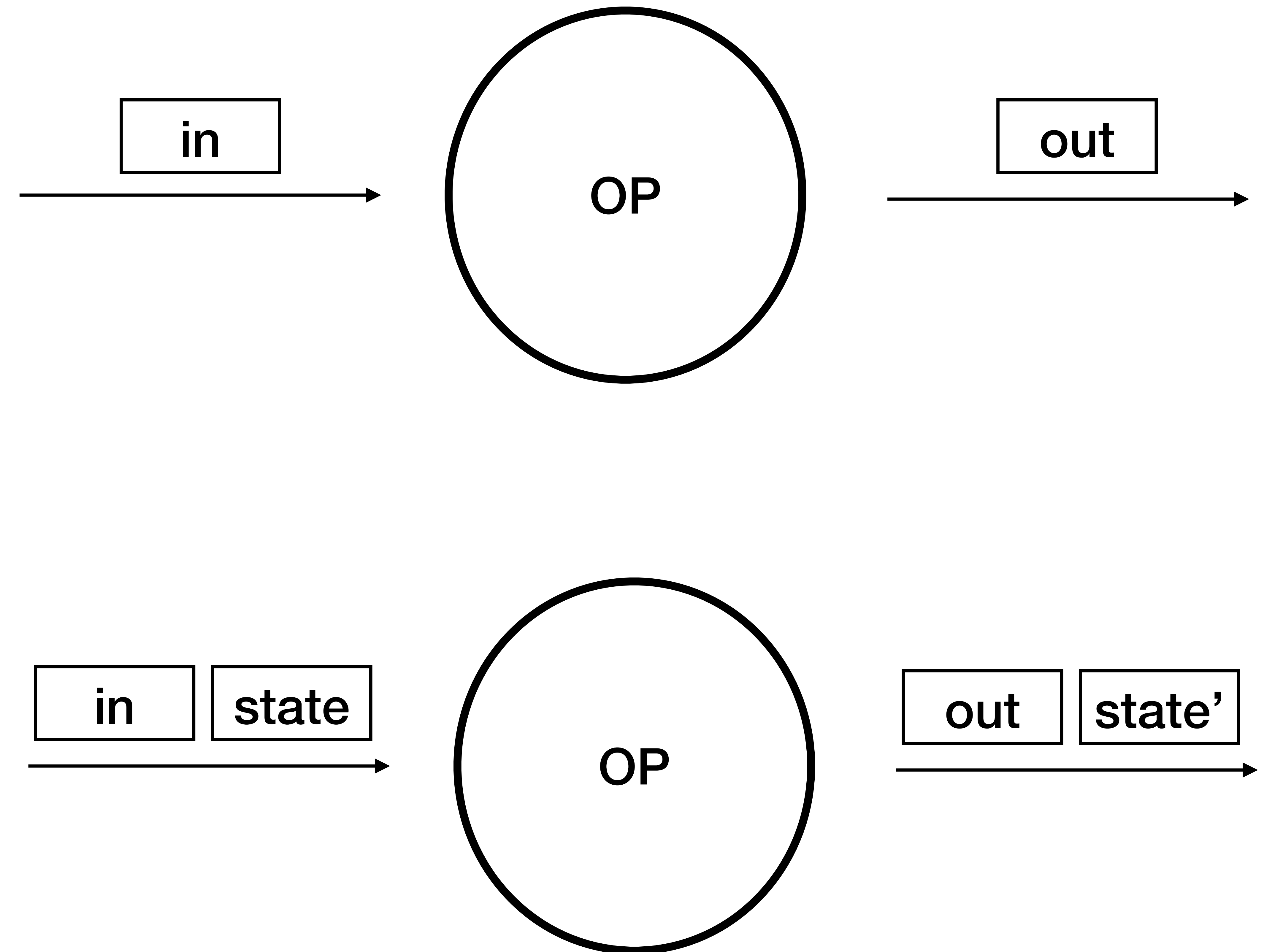
*throughput is important as well

Stream processing model



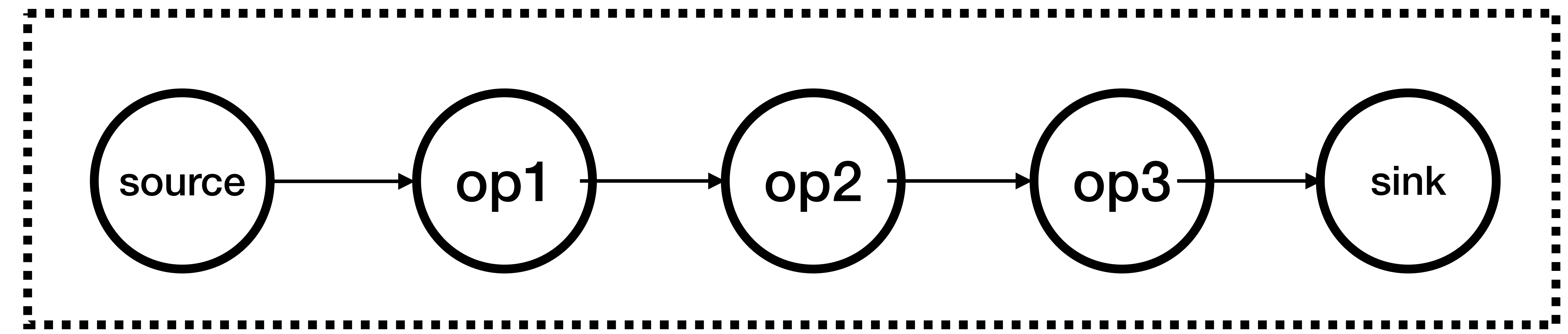
Streaming operations

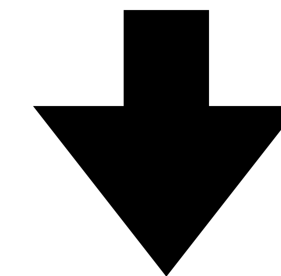
- Stateless
- Stateful



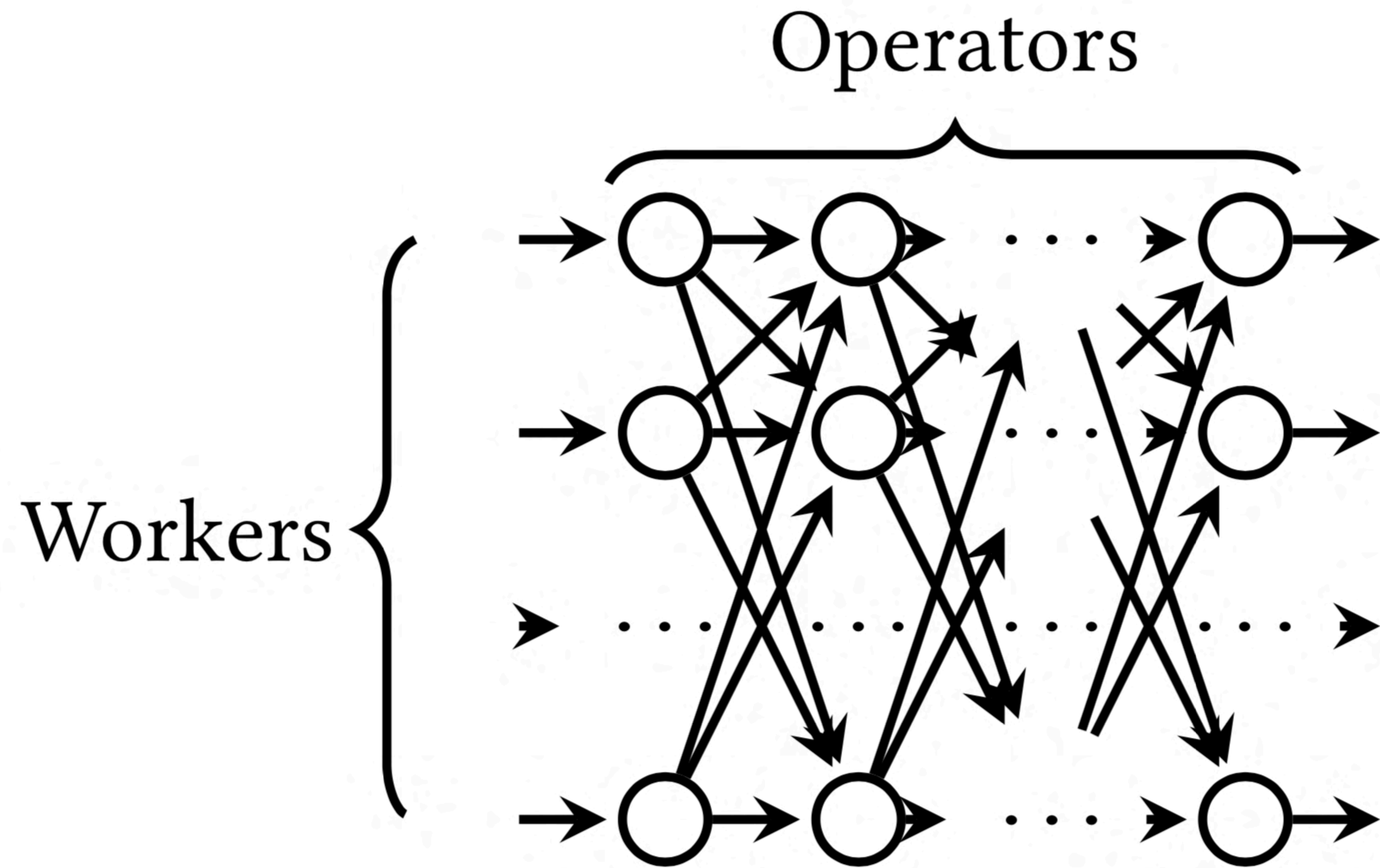
Logical execution graph

- Nodes are operations
- Vertices are connections between operations
- User can define data partitioning scheme before each operation



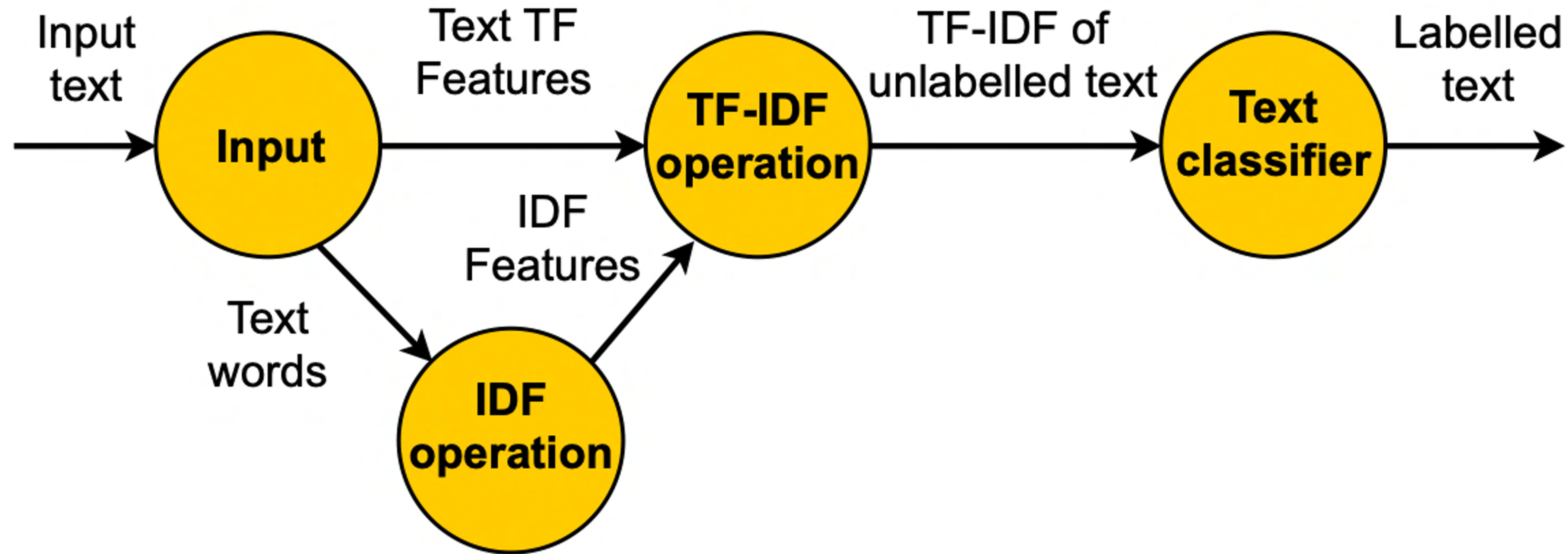


Physical execution graph



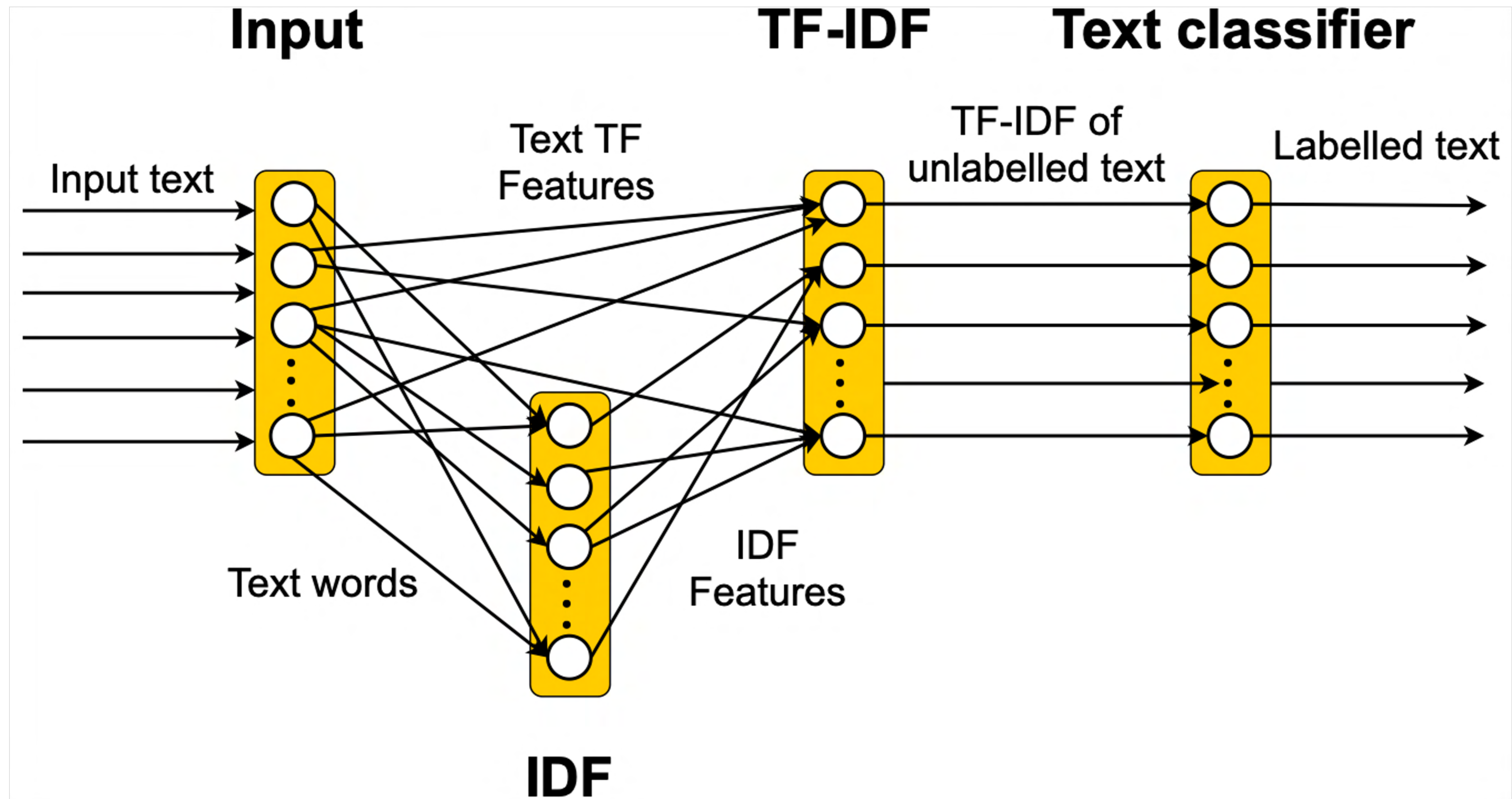
Example: text classification

Logical graph



Example: text classification

Physical graph



Difficulties

- Unbounded input - unbounded output
  - How to prune state?
  - When to release aggregation results?
- Computational nodes may fail
  - How to recover state?
  - How to ensure consistent results?

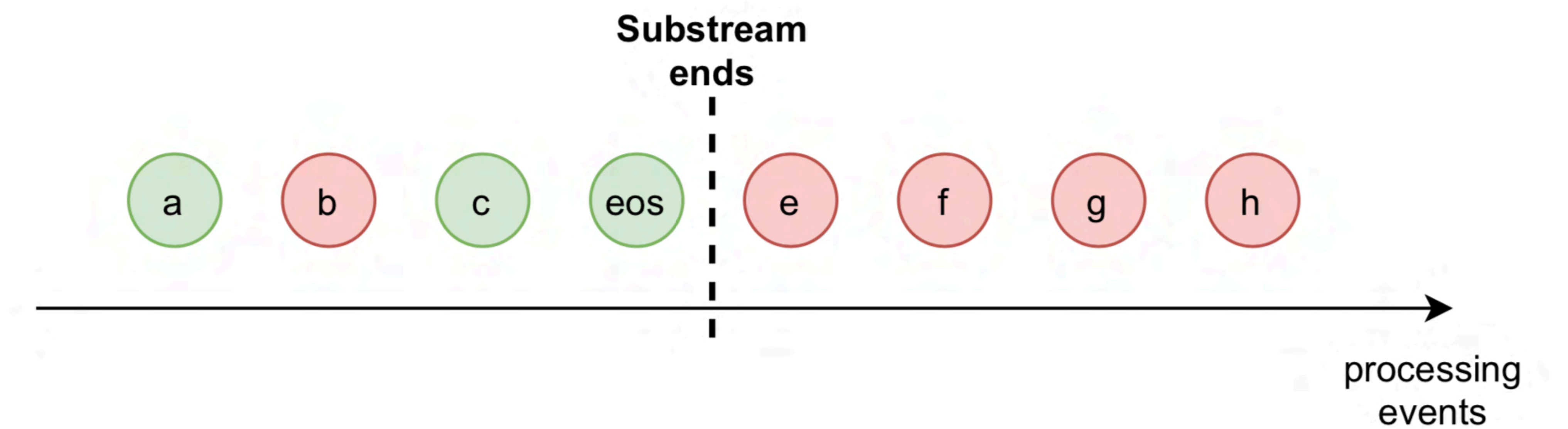
Part 1: unbounded stream problems

- If we do HashJoin of two streams, the state grows on each new element
- At which moment we can release the results?

https://www.reddit.com/r/itookapicture/comments/7r9nqc/itap_of_two_streams_joining_together/

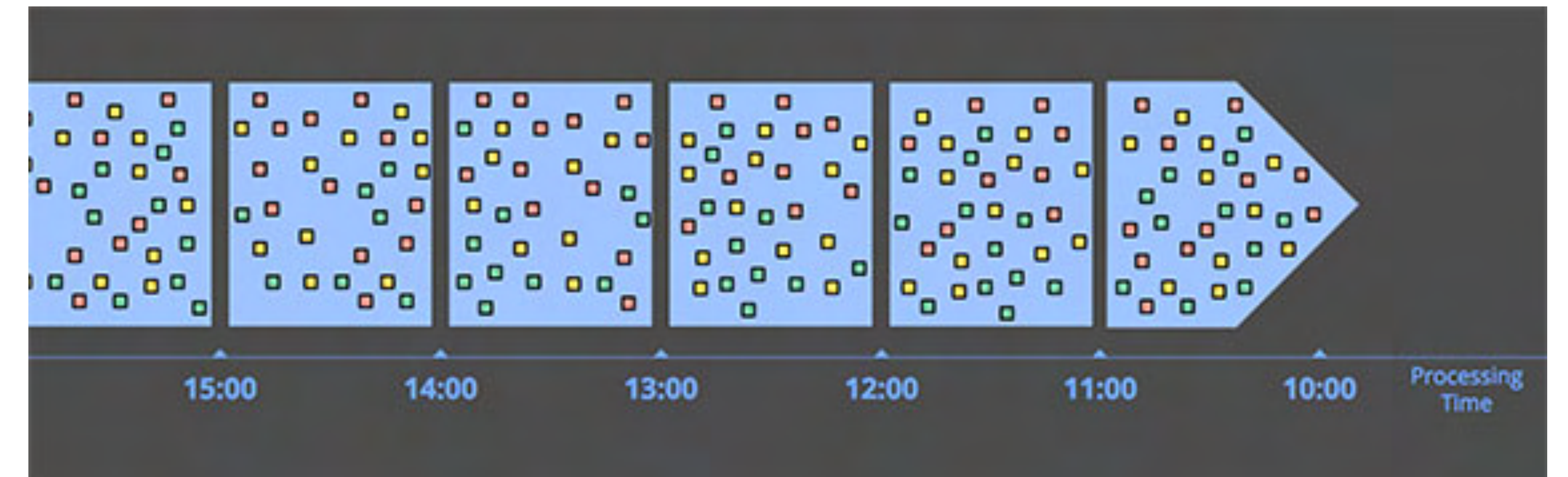
Substreams

- Let p(x) be a predicate defined on stream elements
- All elements satisfying p(x) form a substream
- We are especially interested in discovering substream end
- Multiple substreams can simultaneously coexist



Windows: time-defined substreams

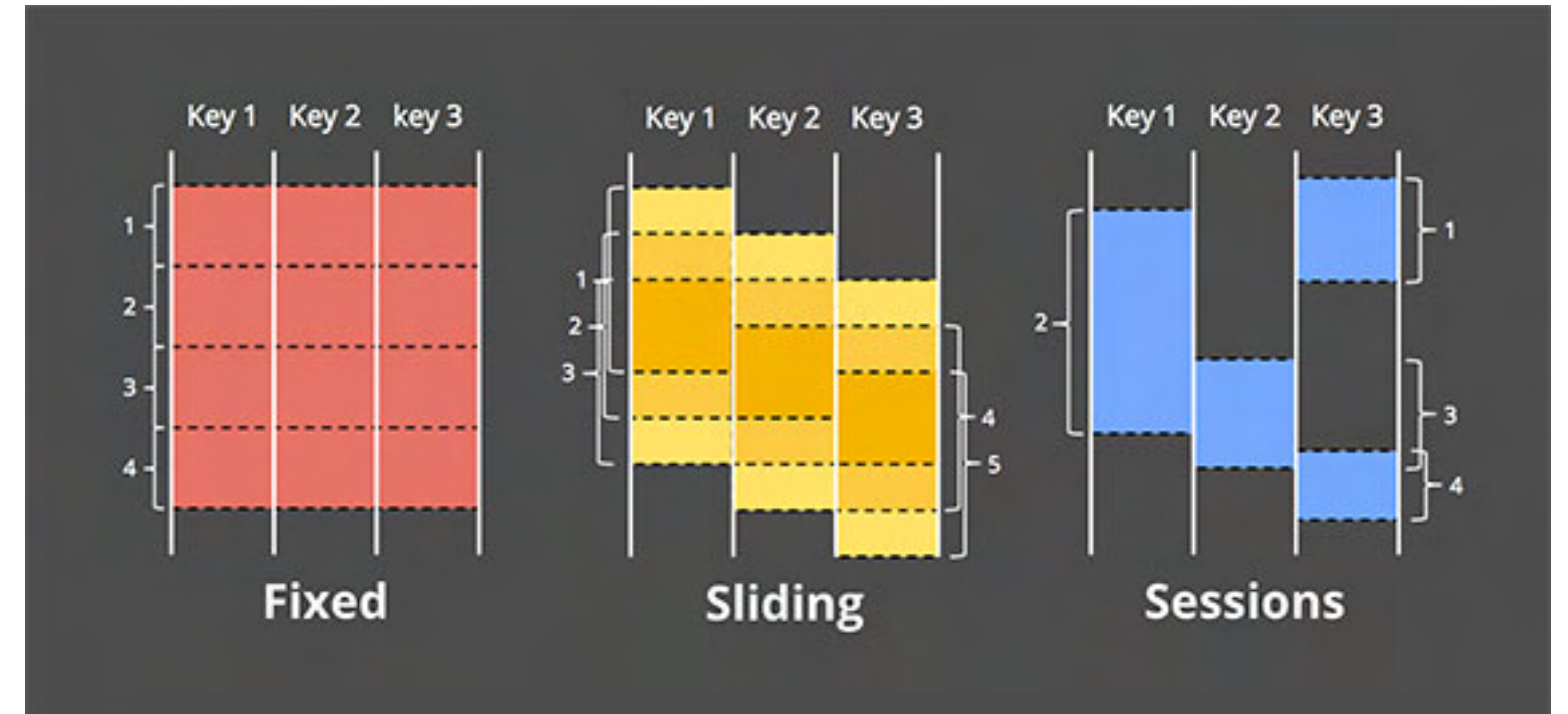
- We can divide stream by timestamps assigned to data elements
- Timestamps can be user or system defined
- For each window we can compute an aggregation and release results



https://www.oreilly.com/radar/the-world-beyond-batch-streaming-101/

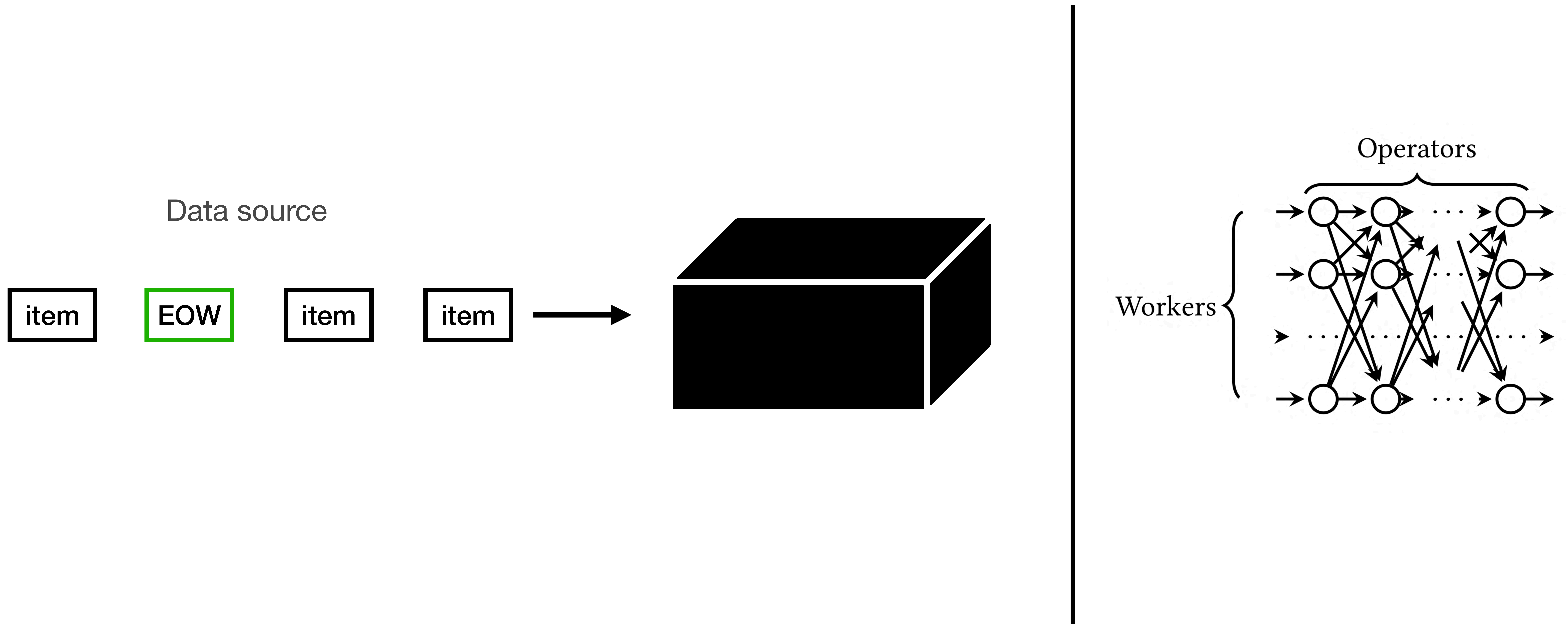
Window types

- Tumbling or fixed
- Sliding
- Session

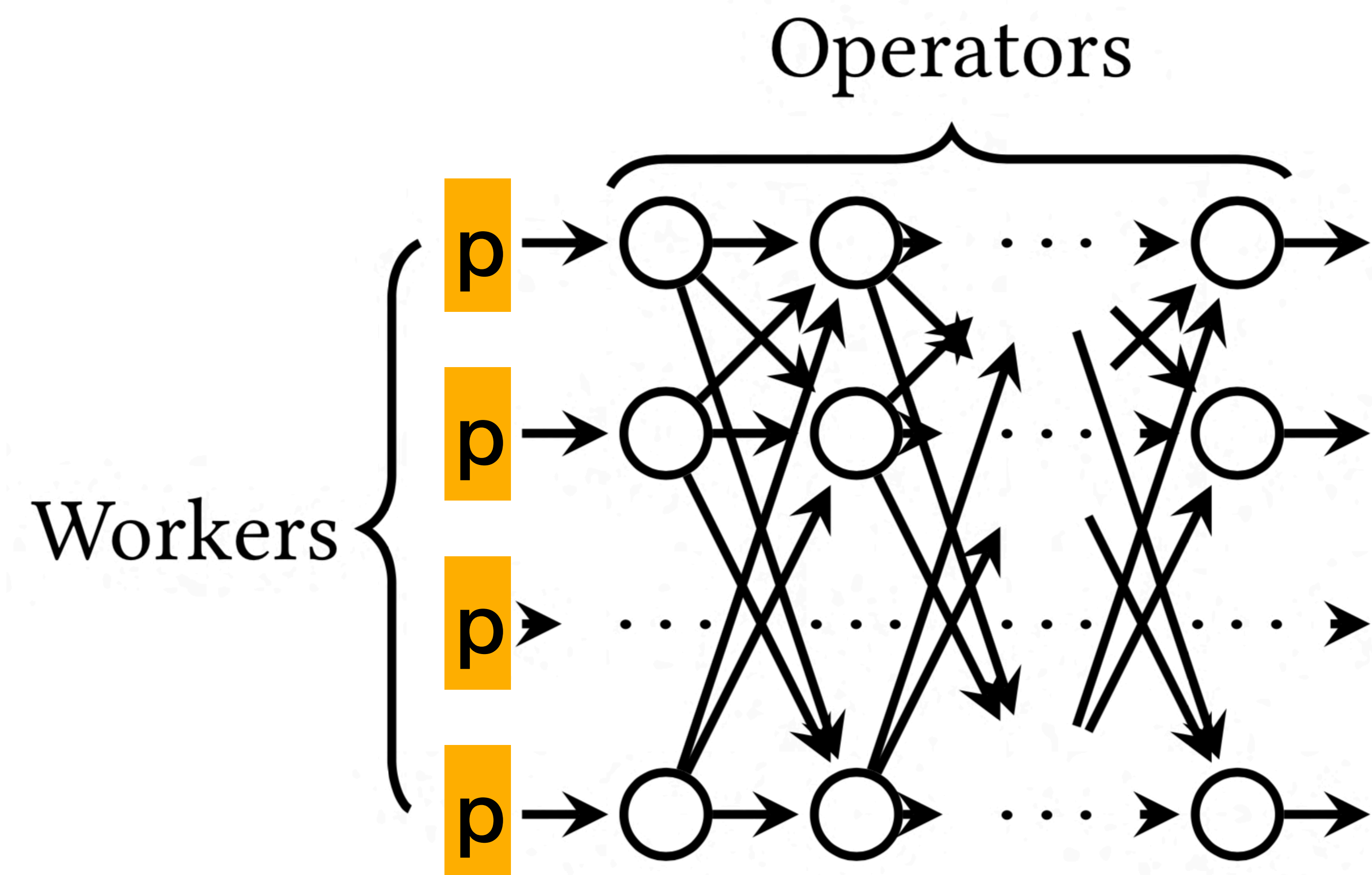


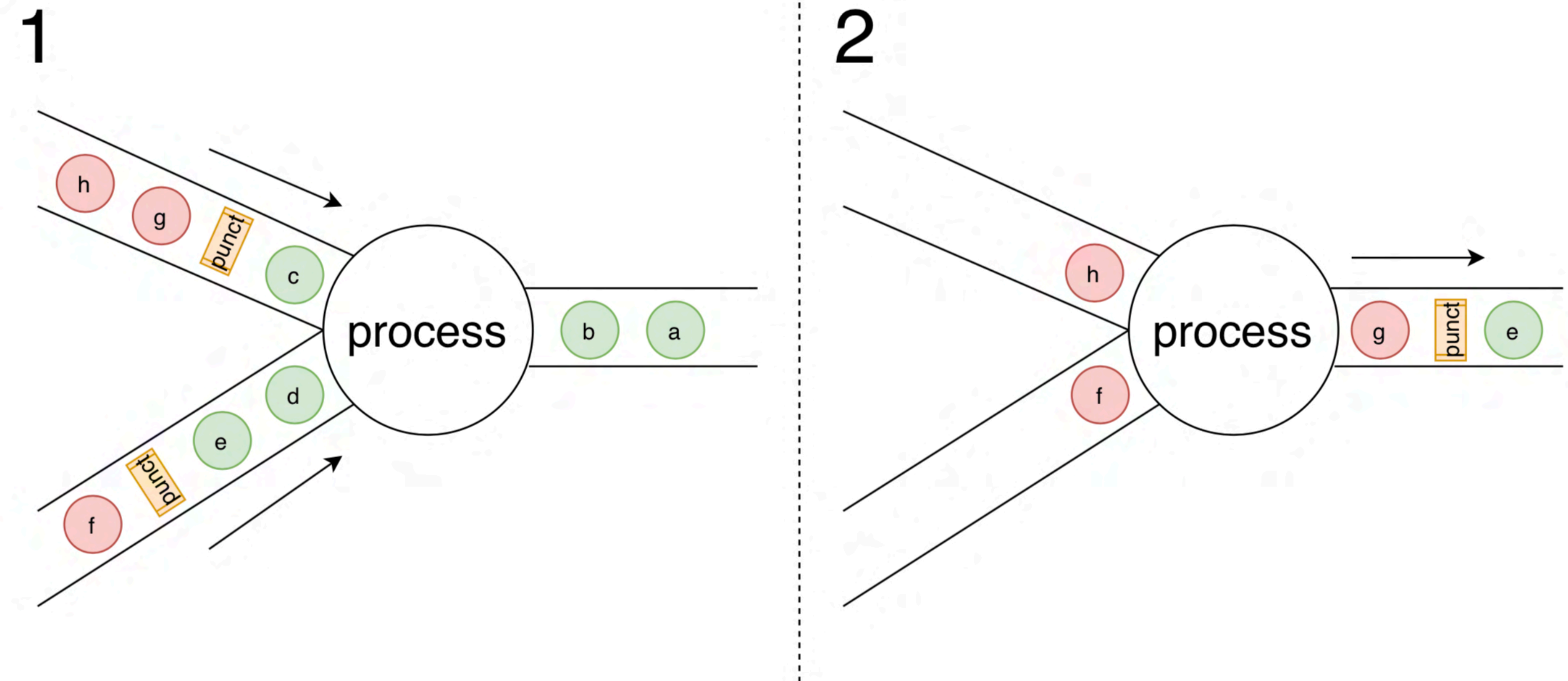
https://www.oreilly.com/radar/the-world-beyond-batch-streaming-101/

How to determine window/substream end?



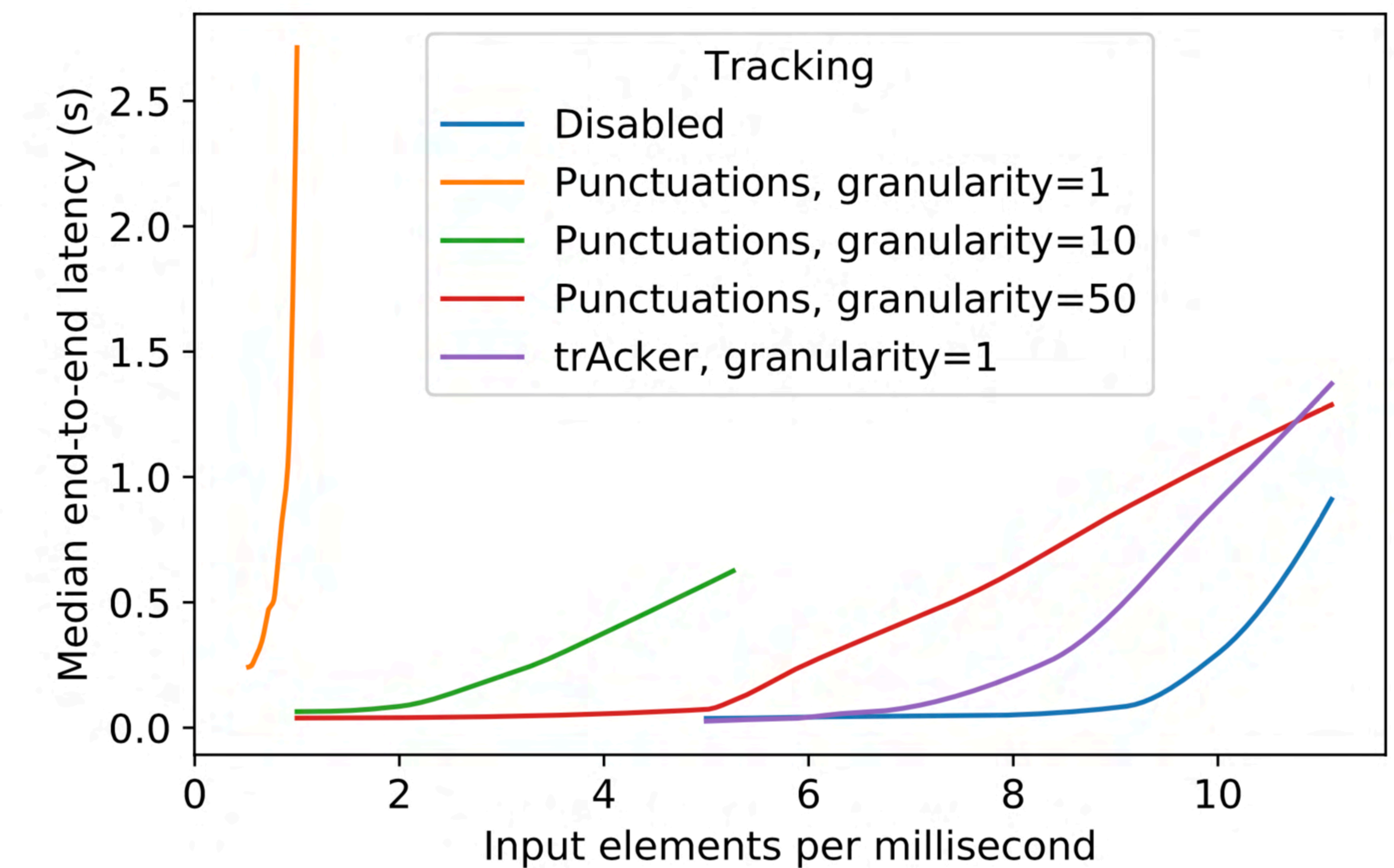
Punctuations: delivering substream end signals to nodes

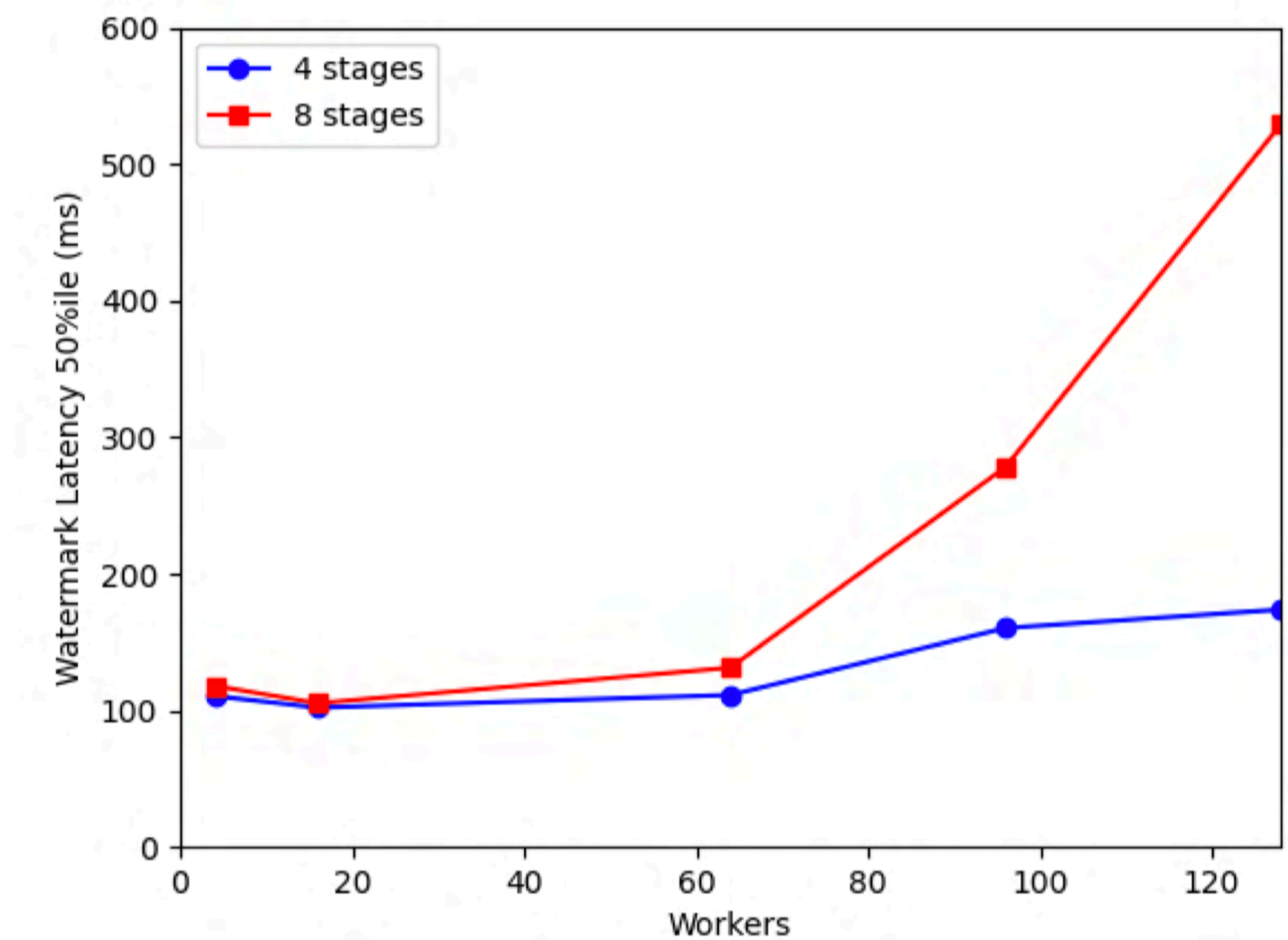




Punctuations properties

- Are easy to implement: no need to add any special agents
- Do not support cyclic execution graphs
- Have O(K||P^2||) network traffic complexity, K - substreams number, P - computational nodes number
- Can limit processing throughput

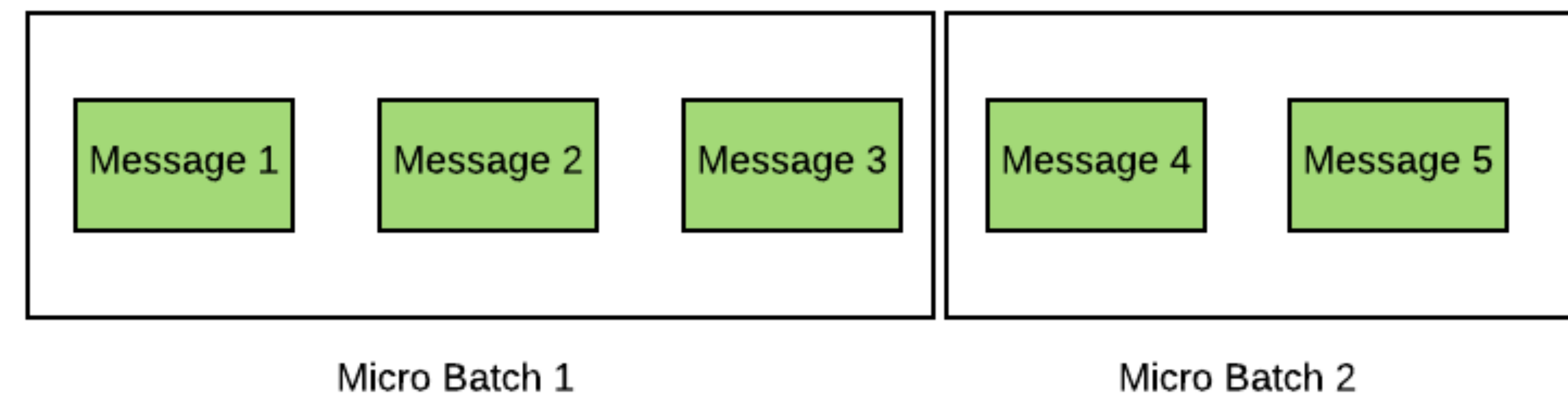




Akidau T. et al. Watermarks in Stream Processing Systems: Semantics and Comparative Analysis of Apache Flink and Google Cloud Dataflow. VLDB 2021

What is the difference from micro-batching?

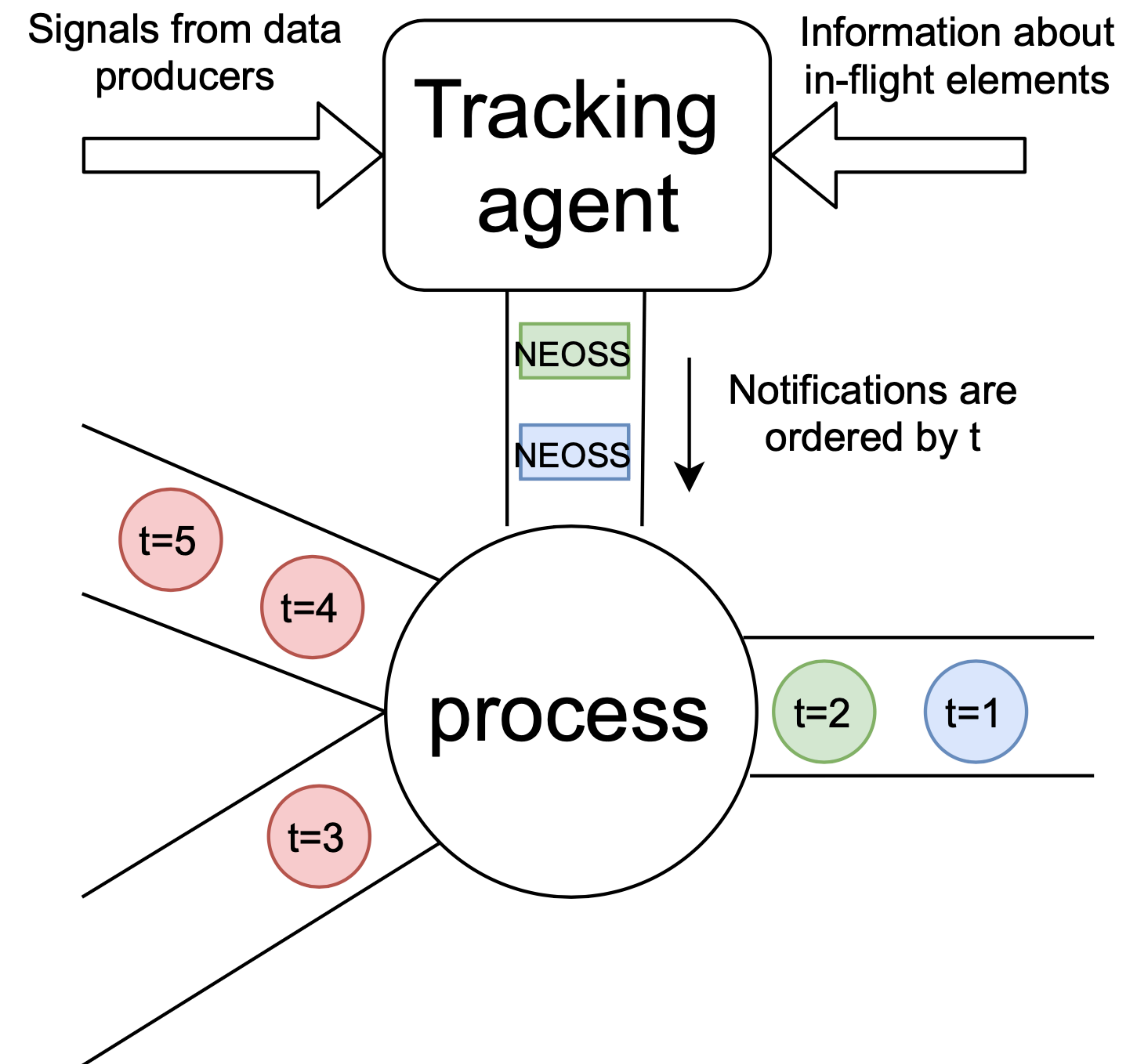
- In micro-batching the next stage does not start before all data from the previous one is processed
- In punctuated stream all computations can be already done, we can just wait for a substream end event to release output



https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781787126497/9/ch09lvl1sec58/understanding-micro-batching

Tracker: a novel approach to deliver substream end signal

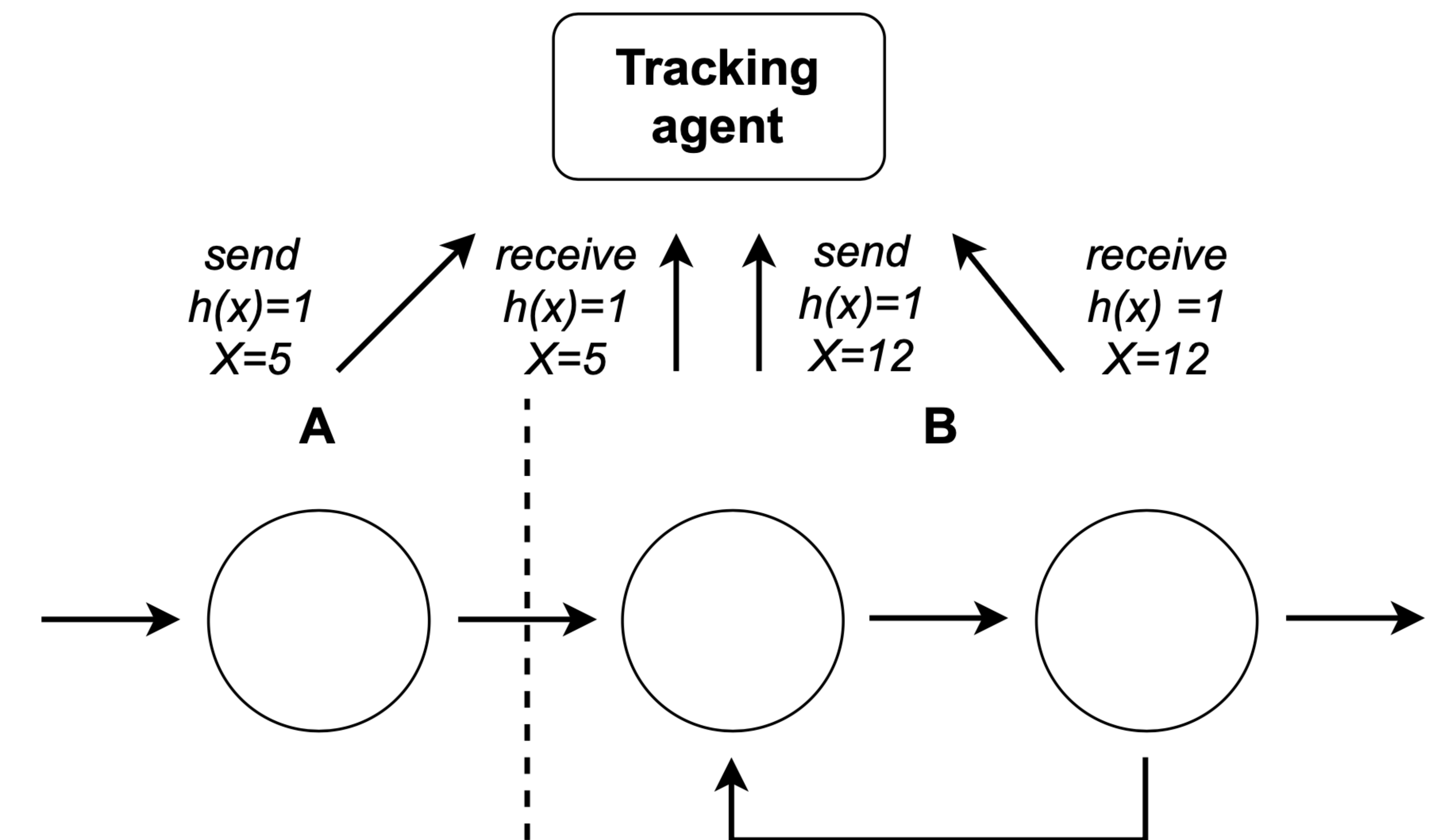
- Signals from data source are sent to external tracking agent
- This agent aggregates information about all in-flight data elements
- Tracking agent determines when a substream ends and notifies all nodes



Trofimov, A., Sokolov, N., Marshalkin, N., Kuralenok, I., & Novikov, B. (2022, June). Substream management in distributed streaming dataflows. In Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems (pp. 55-66).

Tracker: implementation

- Each process sends to the tracking agent reports about every data element
- Each report is labeled by a random number X that appears twice: on send and on receive
- XOR operation for all numbers received from such chain turns into 0
- Tracking agent groups the reports by the predicates and sends termination events
- We call this approach a trAcker framework

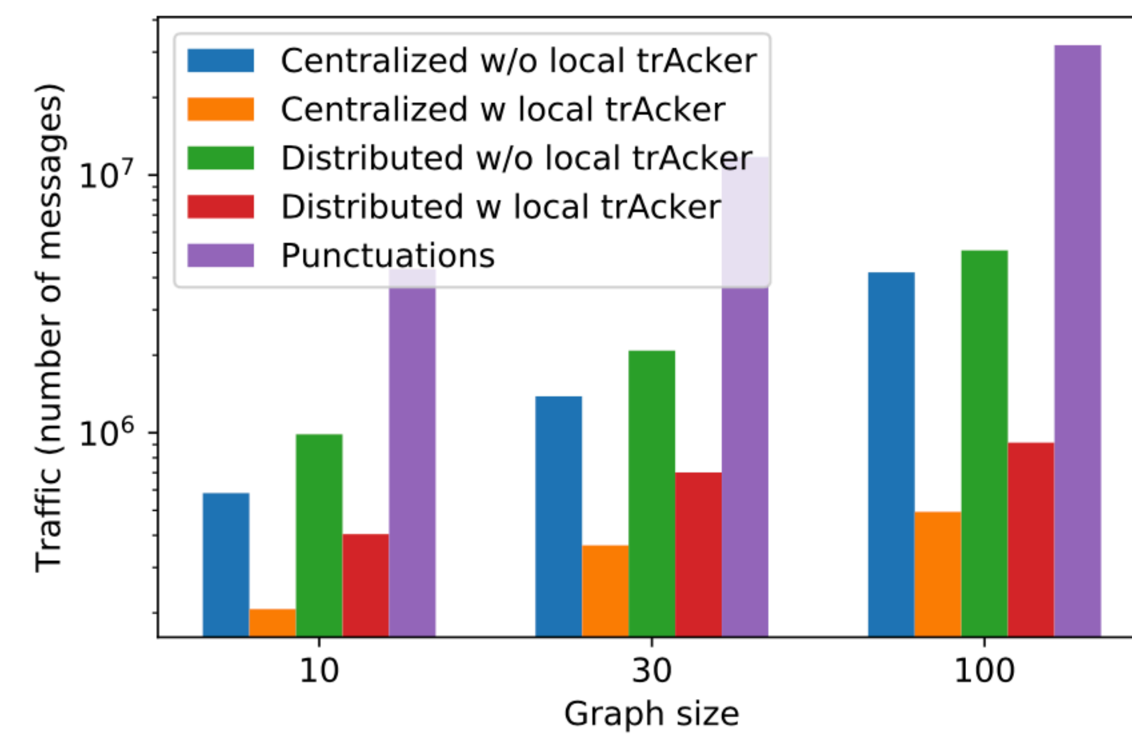


Notified	Predicate	Segment	**Segment XOR**	**XOR**
✓	h(x)	A	**000**	**000**
		B	**000**	
	q(x)	A	**000**	**110**
		B	**110**	
✓	z(x)	A	**000**	**000**
		B	**000**	

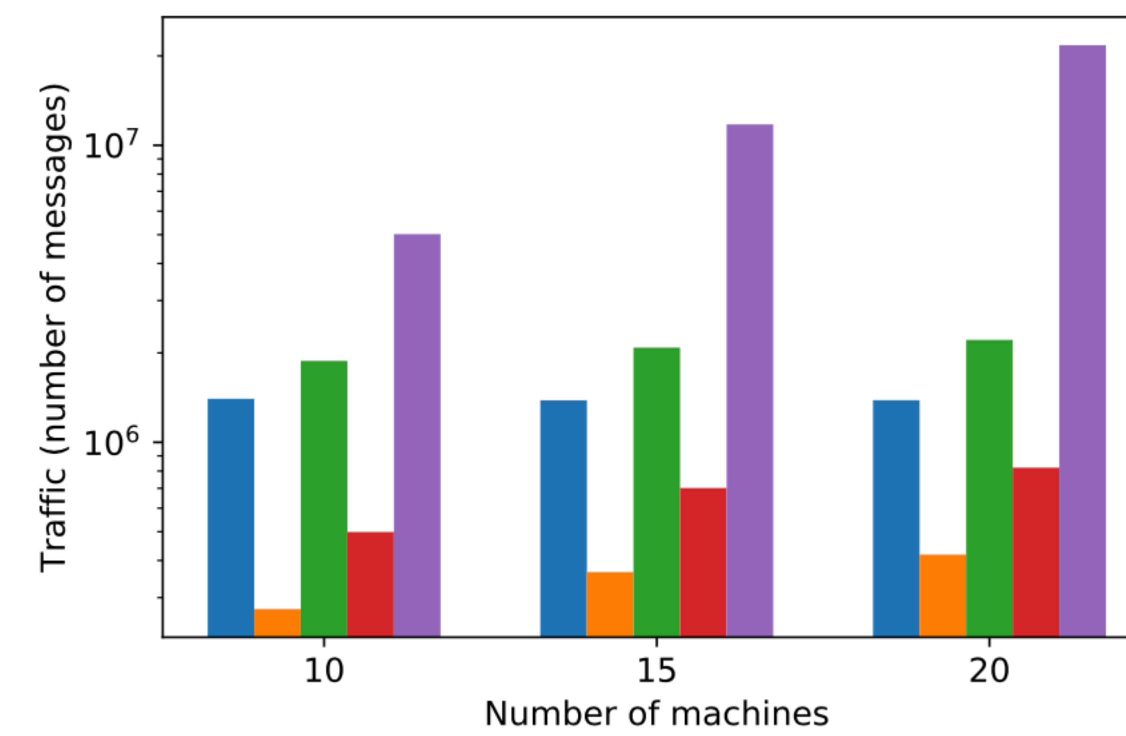
Tracker: properties

- Traffic complexity is linear from the nodes number - O(K||P||)
- XOR is a commutative operation, so we can do pre-aggregation on nodes
- Cyclic graphs are supported
- It is pretty easy to design distributed implementation of the tracking agent

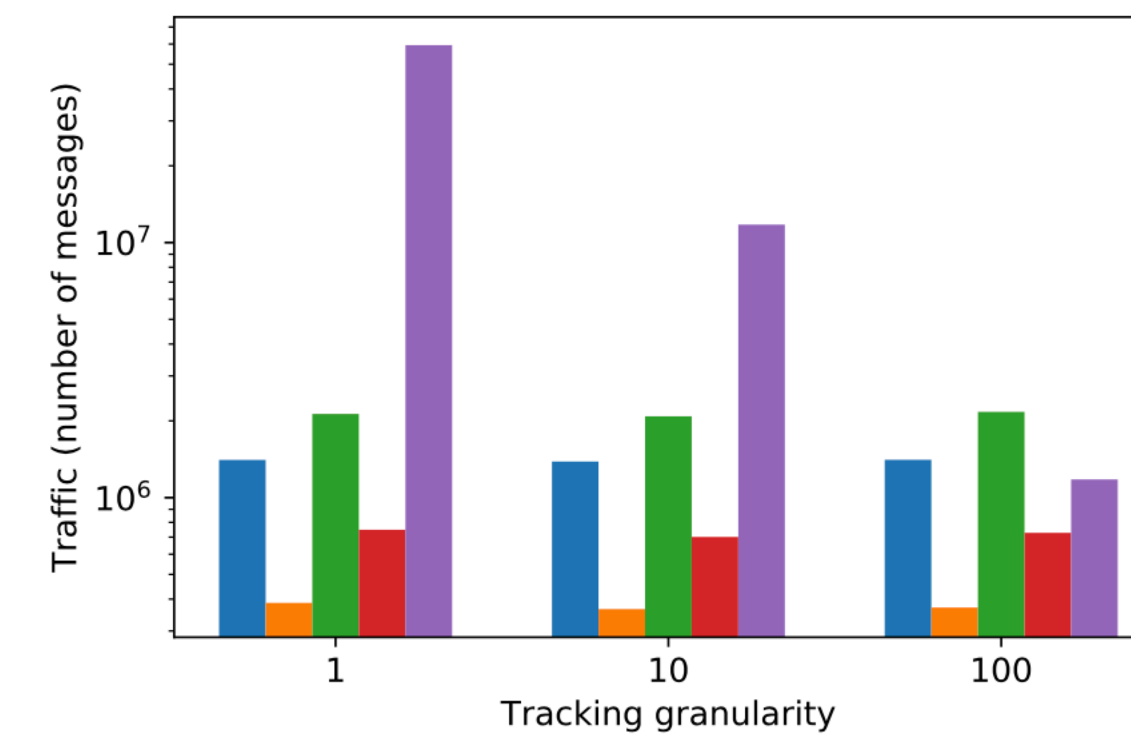
Tracker: overhead on a stream processing engine



(a) Traffic by graph size

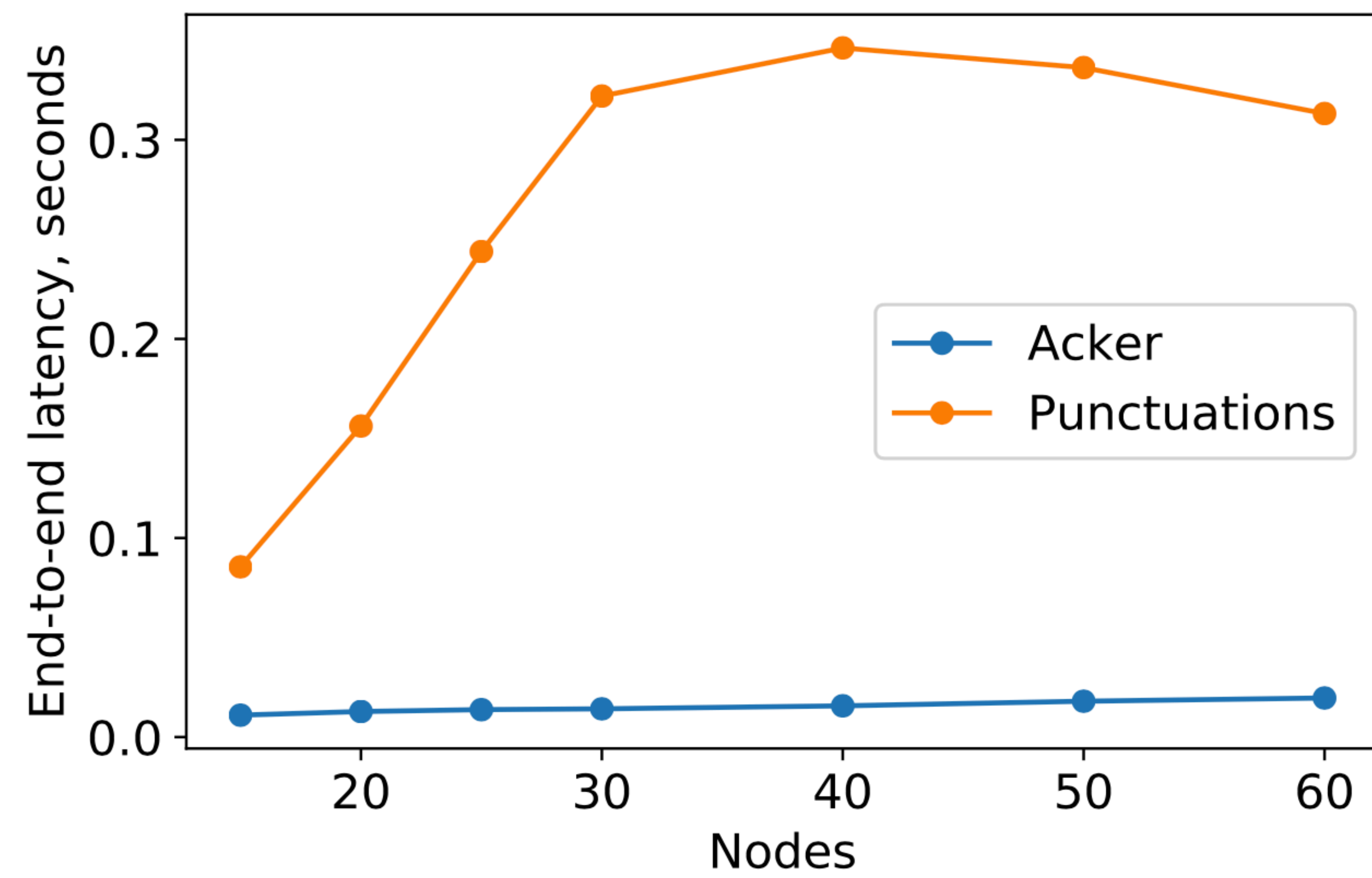


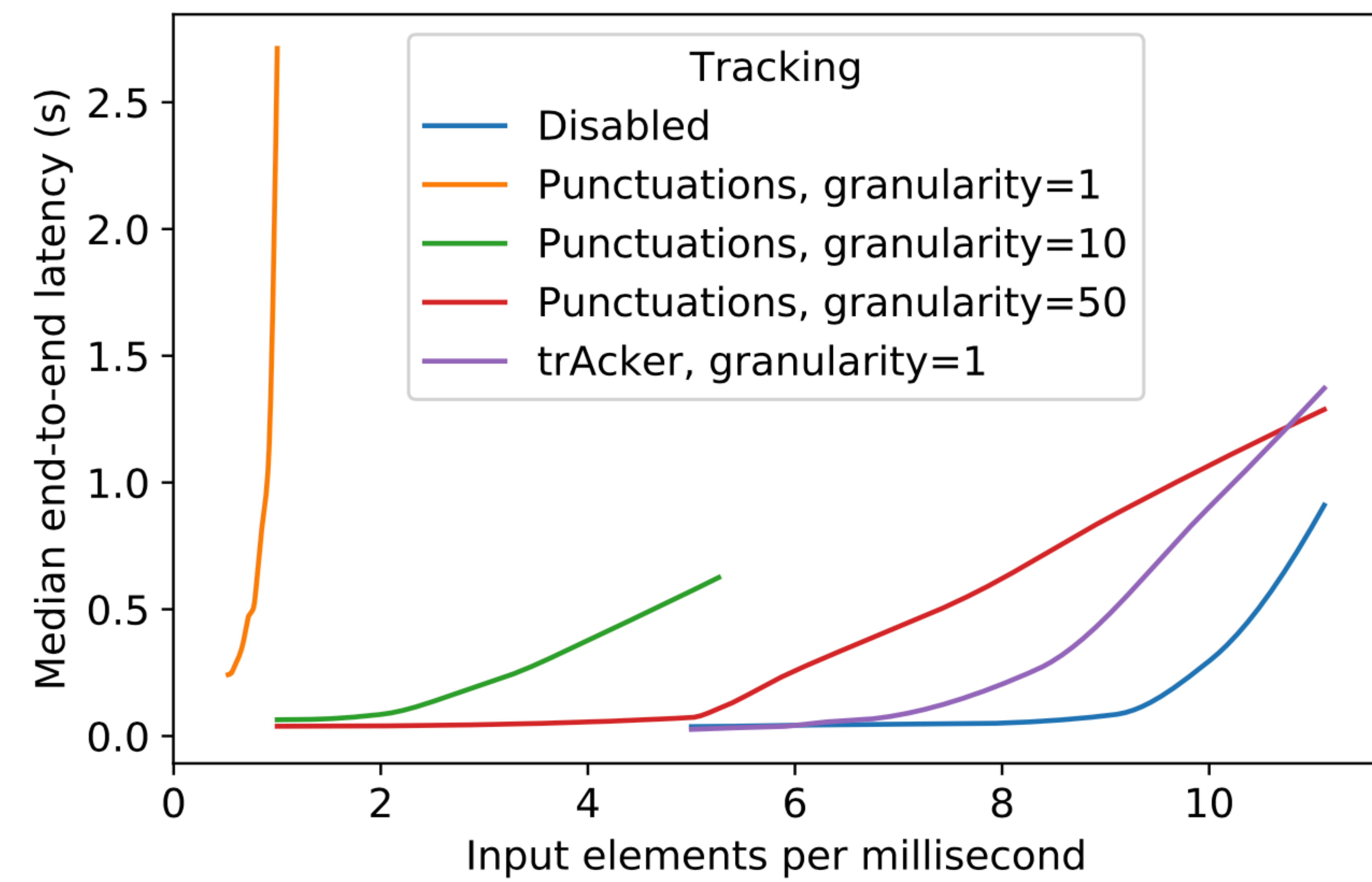
(b) Traffic by number of VMs



(c) Traffic by tracking frequency

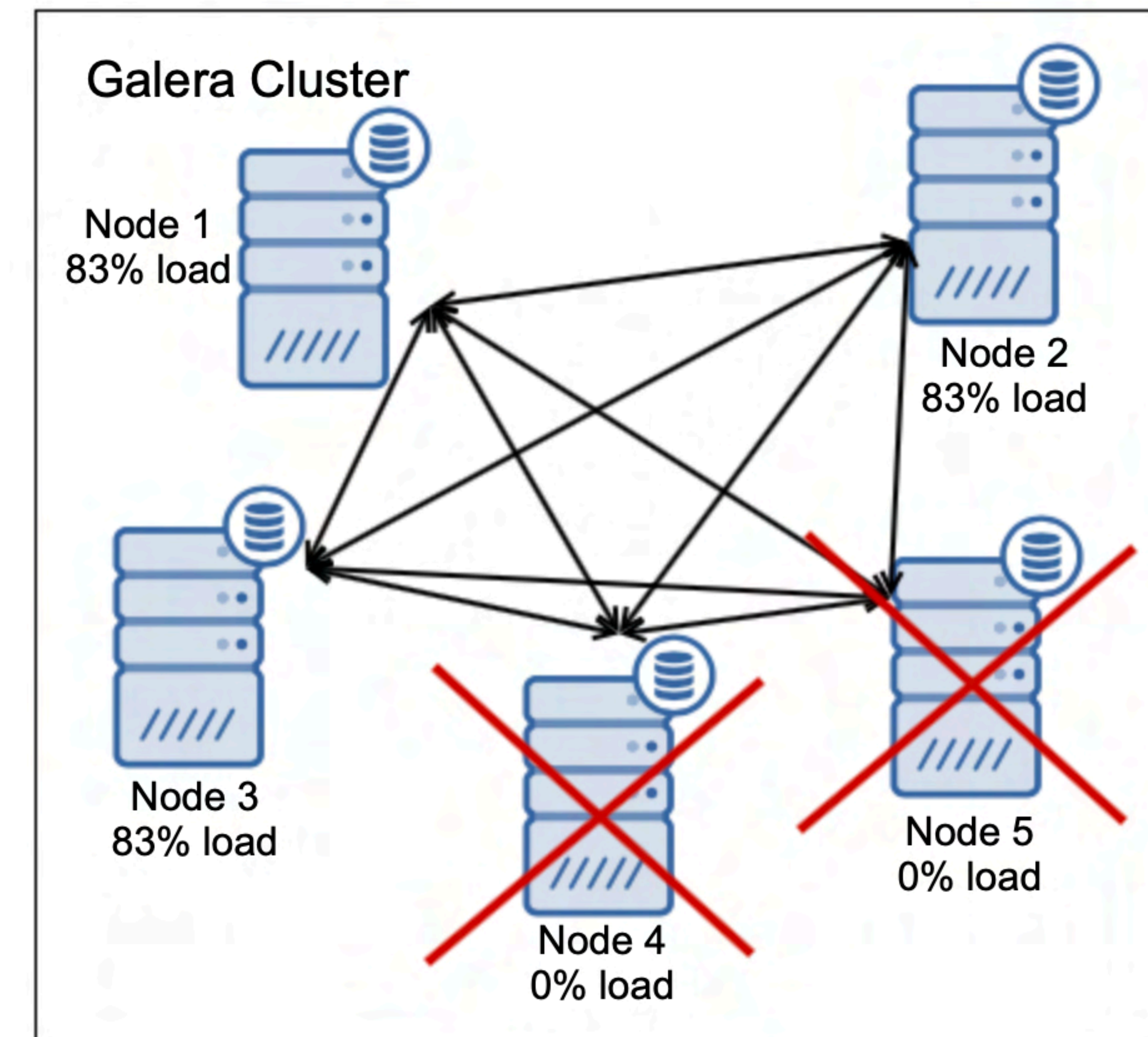
Tracker: end-to-end experiments





Part 2: fault tolerance and consistency problems

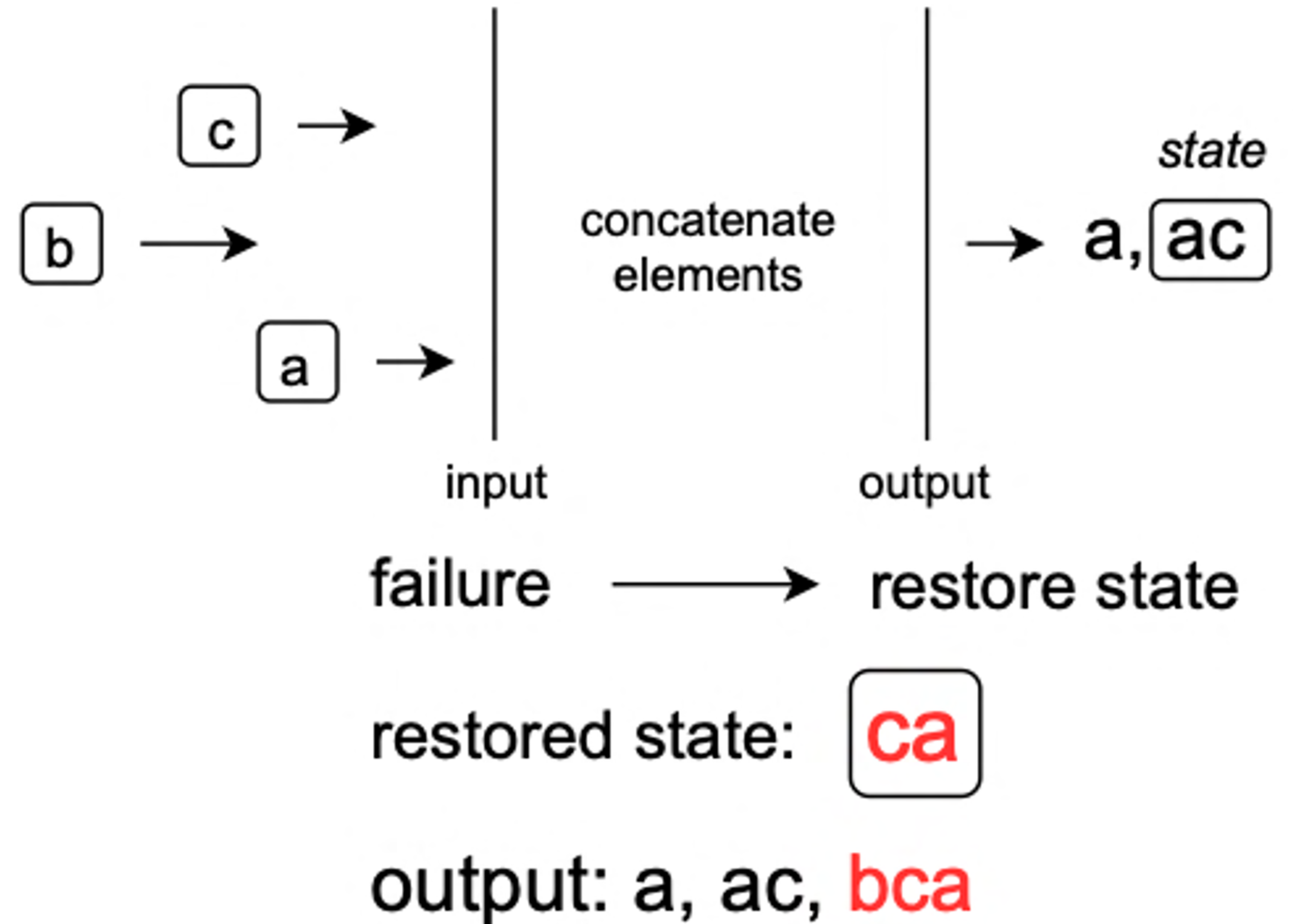
- Computational nodes may fail
- Users should NOT observe failures



https://severalnines.com/database-blog/clustered-database-node-failure-and-its-impact-high-availability

State recovery problem

- Elements processing order can be non-deterministic
- Operations can be non-commutative
- Our goal is to ensure that user does not observe failures



Delivery guarantees

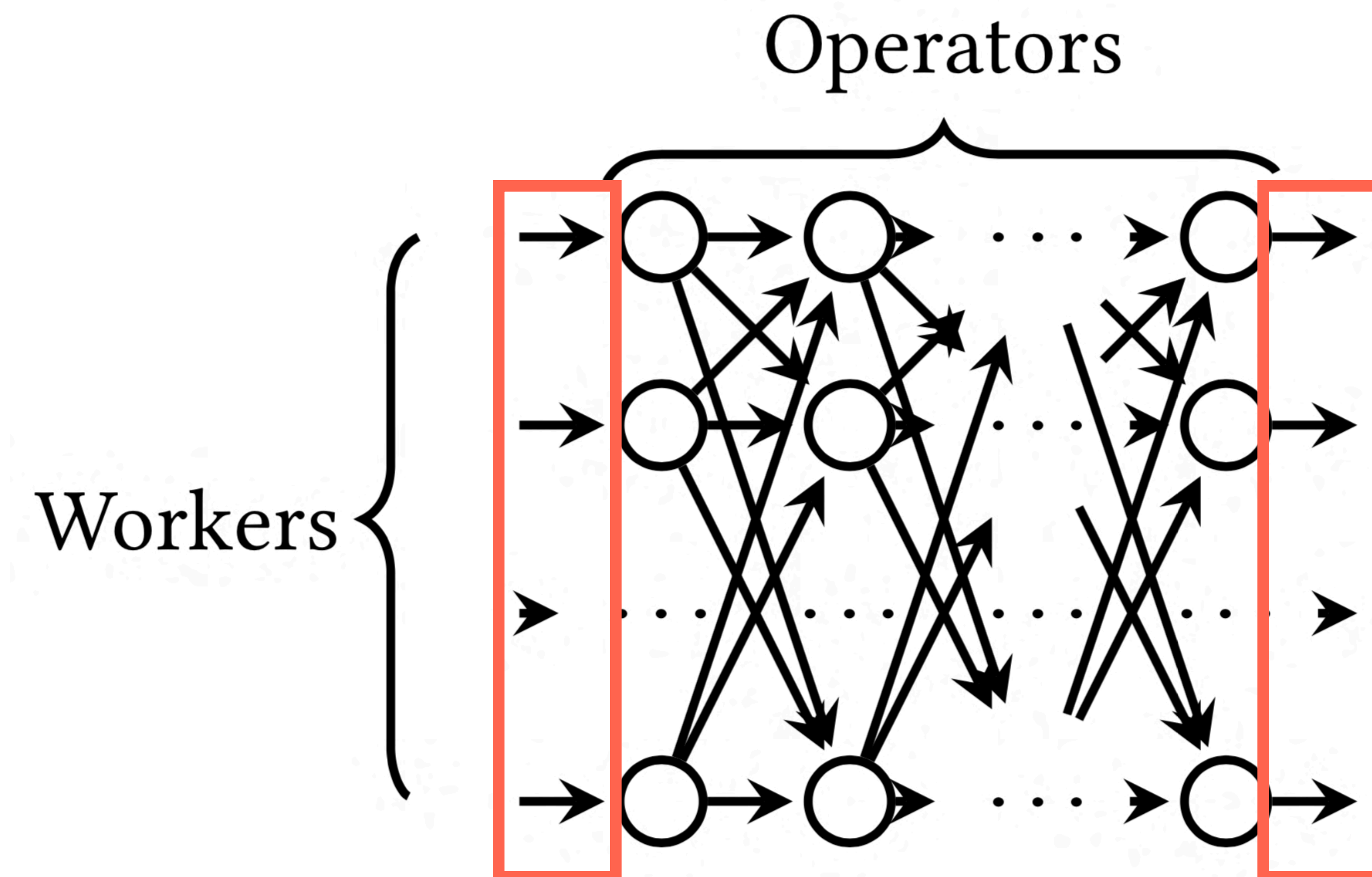
- At-most-once
- **At-least-once**
- **Exactly-once**

Delivery guarantees are actually about consistency

- Suppose that we have a model recovery mechanism that ensures recovery of operations states all in-flight exactly as they were before the failure
- Let B be a set of output elements released by a system with the model recovery mechanism
- **At most once** guarantees that output consists of a subset of B
- **At least once** guarantees that output consists of a superset of B
- **Exactly once** guarantees that output consists of exactly B

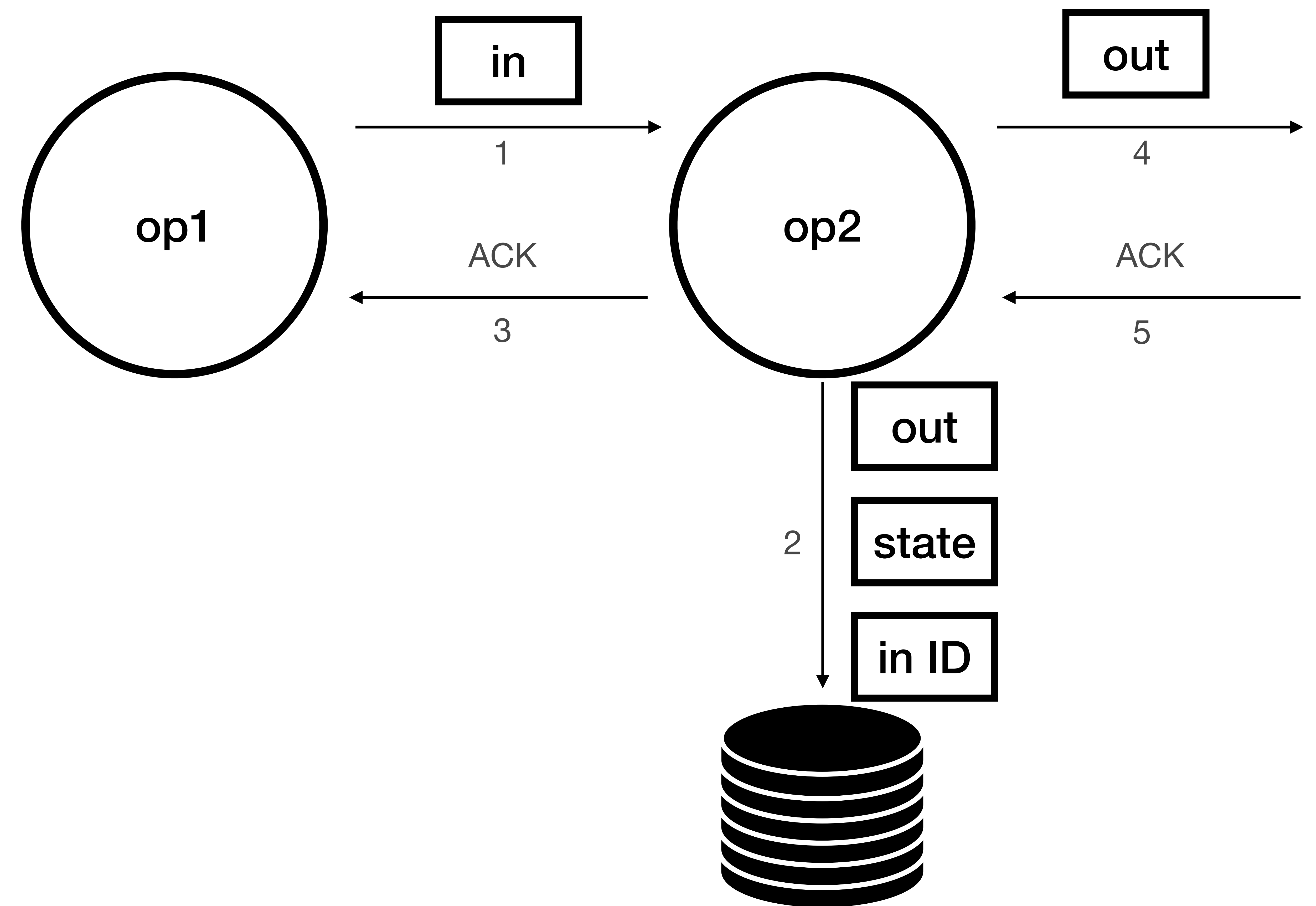
Two recovery models

- Model 1: internal network channels are controlled
- Model 2: input and output channels only are controlled



Model 1 by MillWheel example: state saving process

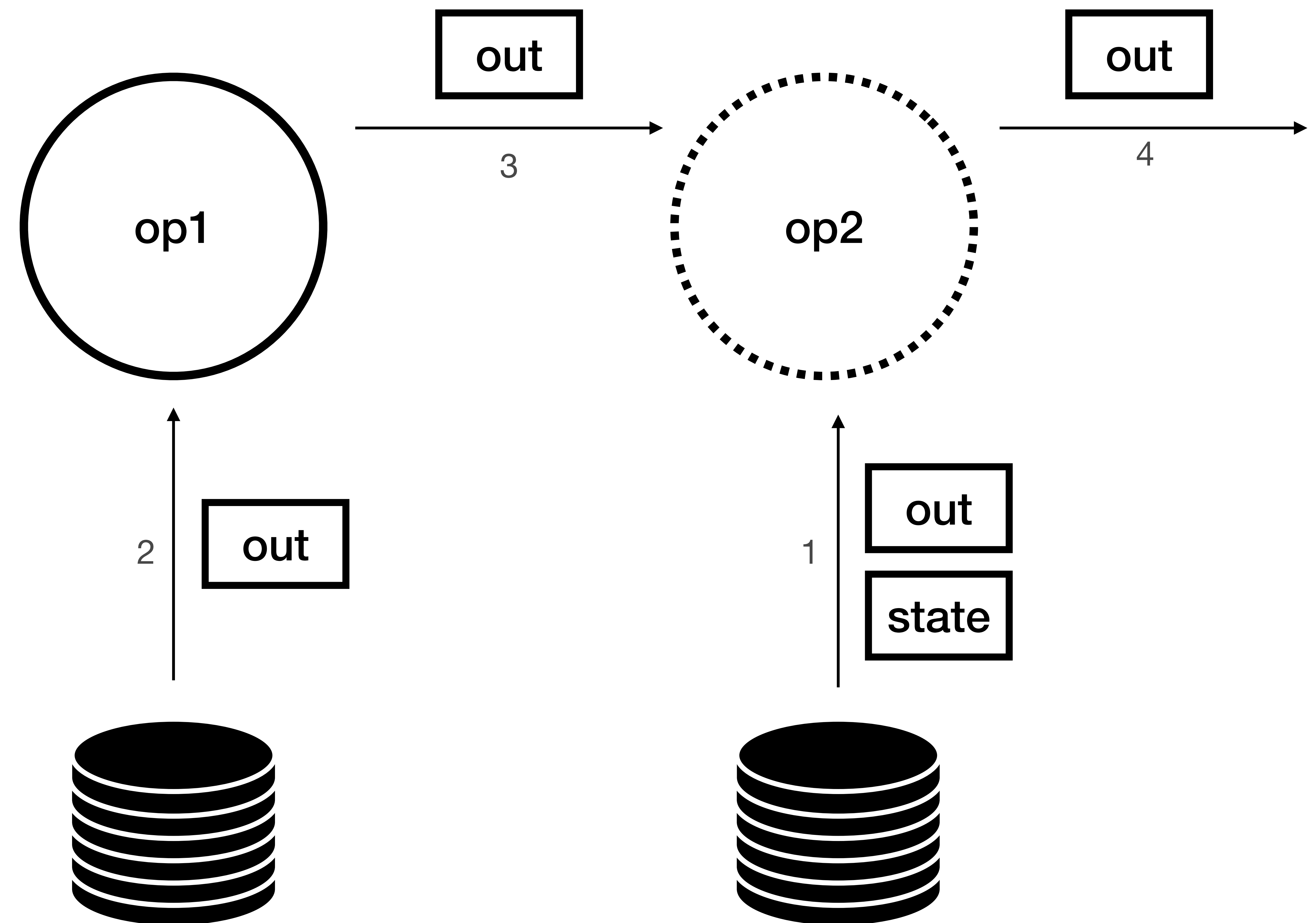
- Each element has a unique ID
- Node filters out an element if it has been already processed
- In a single transaction system saves: input element ID, new state (or diff), output elements
- Node sends ACK for the input element to the previous node



Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., ... & Whittle, S. (2013). Millwheel: Fault-tolerant stream processing at internet scale. Proceedings of the VLDB Endowment, 6(11), 1033-1044.

Model 1 by MillWheel example: state recovery

- Nodes re-send all output elements without ACKs
- If failed node has processed an element but did not send ACK, then repeated input will be filtered out by the deduplicator

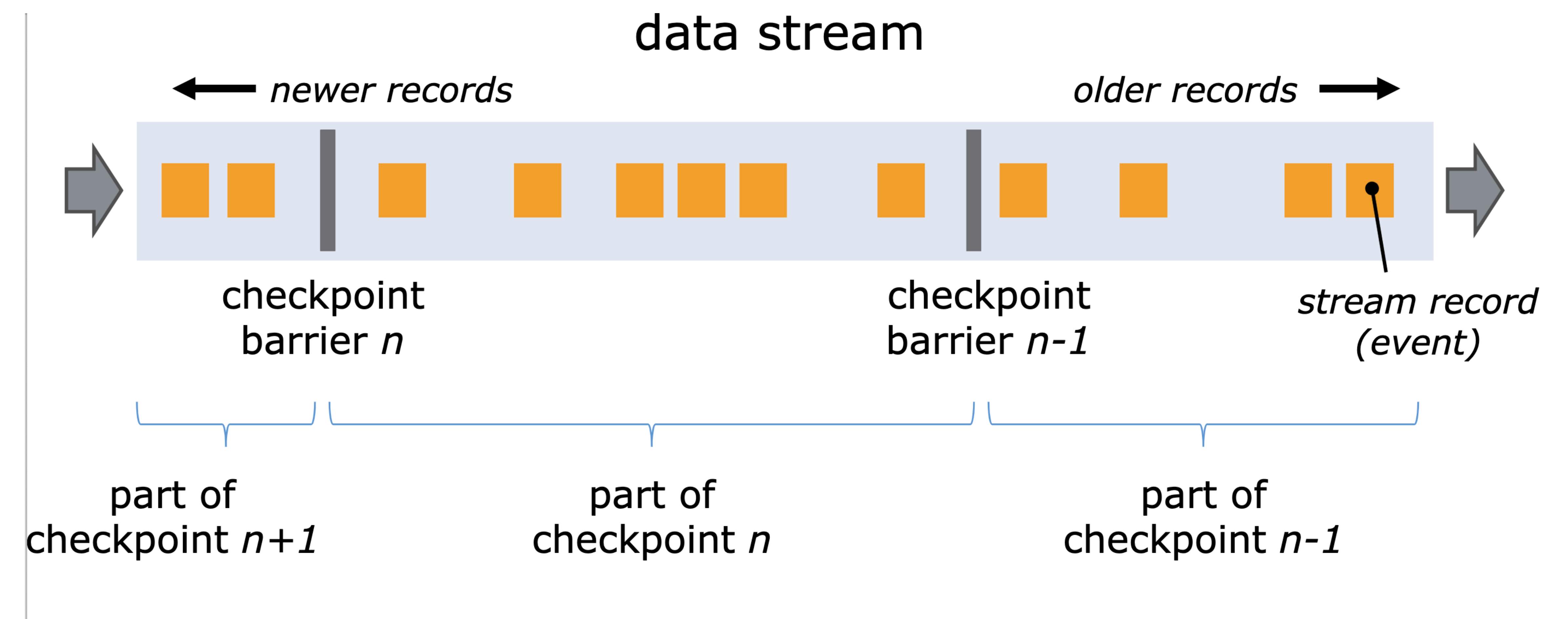


Model 1 properties

- Recovery does not require stop-the-world
- Overhead is spreaded among the operations: there is no overhead on stateless operations
- There is a need for a very efficient transactional storage for the state

Model 2 by Flink example: state saving process

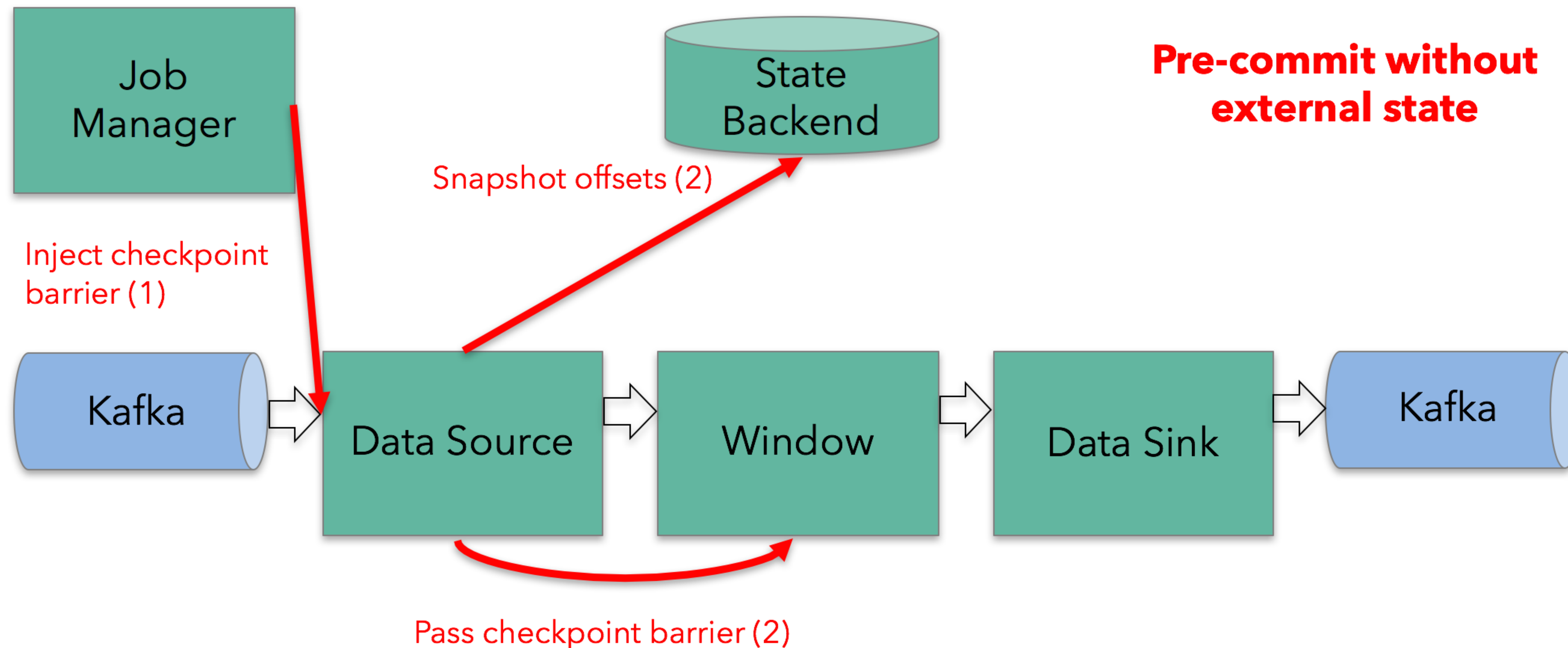
- The main idea is to divide stream into epochs and create a snapshot for each epoch
- Periodically special elements called “barriers” are injected into a stream
- When a barrier arrives to a node, the corresponding input network channel is blocked
- When barriers arrive from all input channels, node saves its state snapshot (a local snapshot) to an external storage
- When all barriers arrive to data sinks, the set of all local snapshots is labeled as a global snapshot



https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/concepts/stateful-stream-processing/

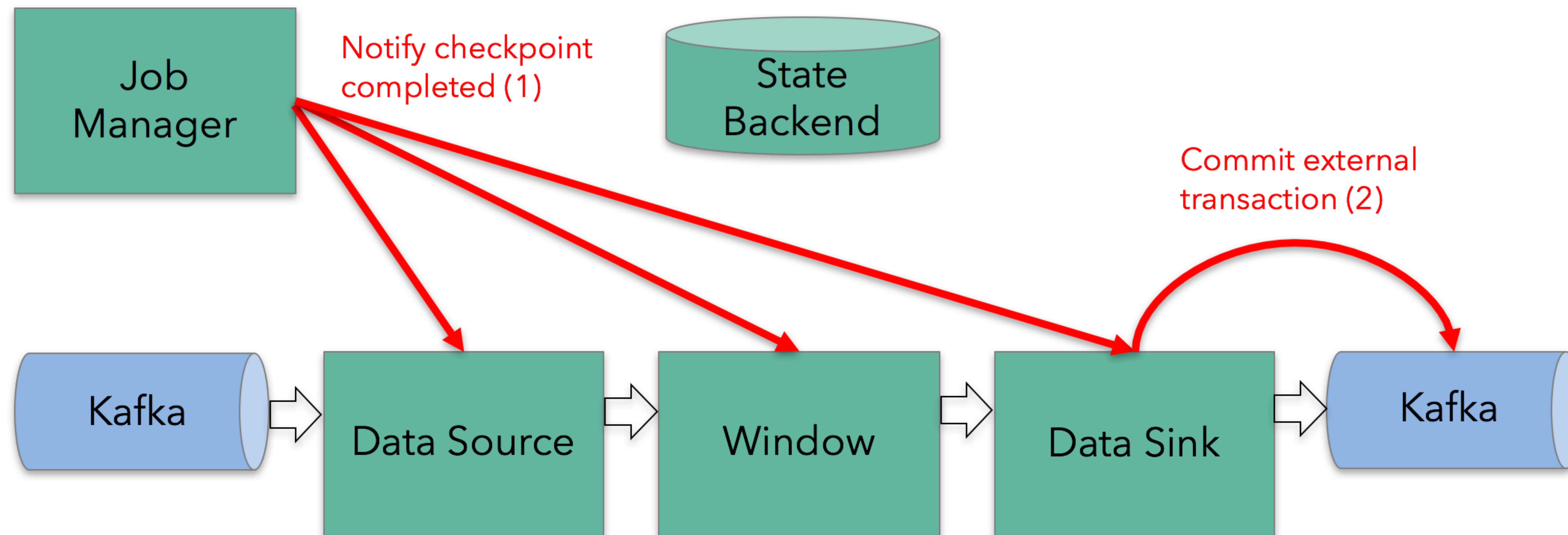
Carbone P. et al. Lightweight asynchronous snapshots for distributed dataflows //arXiv preprint arXiv:1506.08603. – 2015.

Model 2 by Flink example: state saving process phase 1



https://flink.apache.org/features/2018/03/01/end-to-end-exactly-once-apache-flink.html

Model 2 by Flink example: state saving process phase 2



https://flink.apache.org/features/2018/03/01/end-to-end-exactly-once-apache-flink.html

Barrier alignment

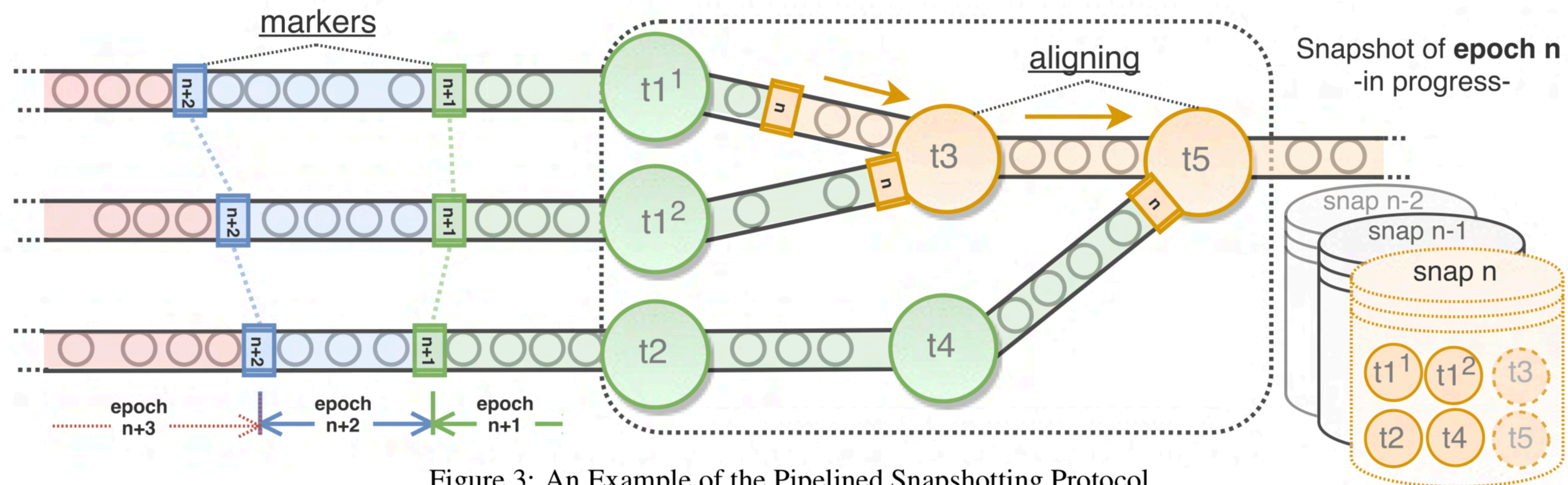


Figure 3: An Example of the Pipelined Snapshotting Protocol.

Carbone P. et al. State management in Apache Flink®: consistent stateful distributed stream processing //Proceedings of the VLDB Endowment. – 2017. – T. 10. – №. 12. – C. 1718-1729.

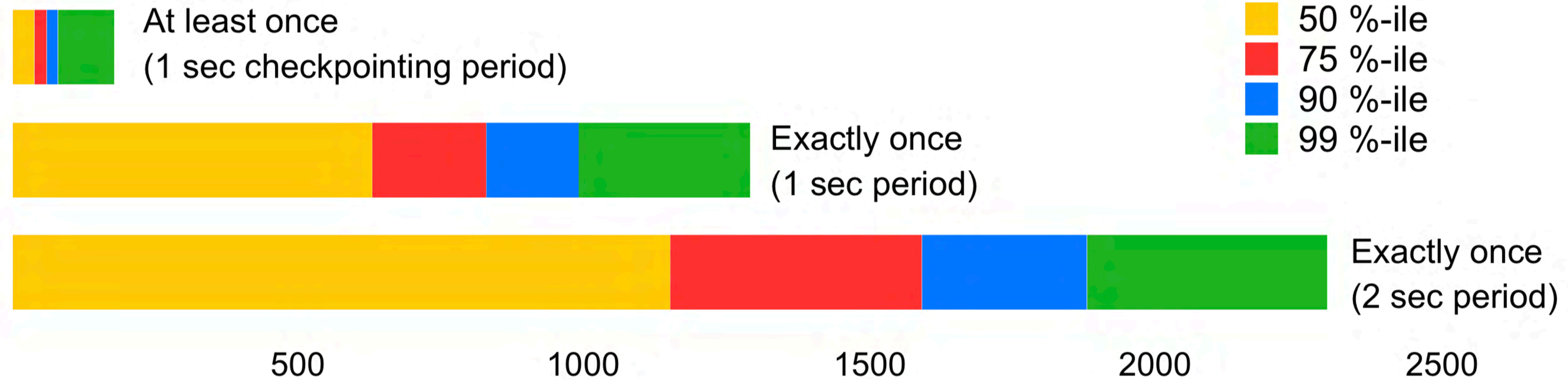
Model 2 by Flink example: state recovery

- Nodes load the last saved state
- Some amount of input elements are reprocessed

Model 2 properties

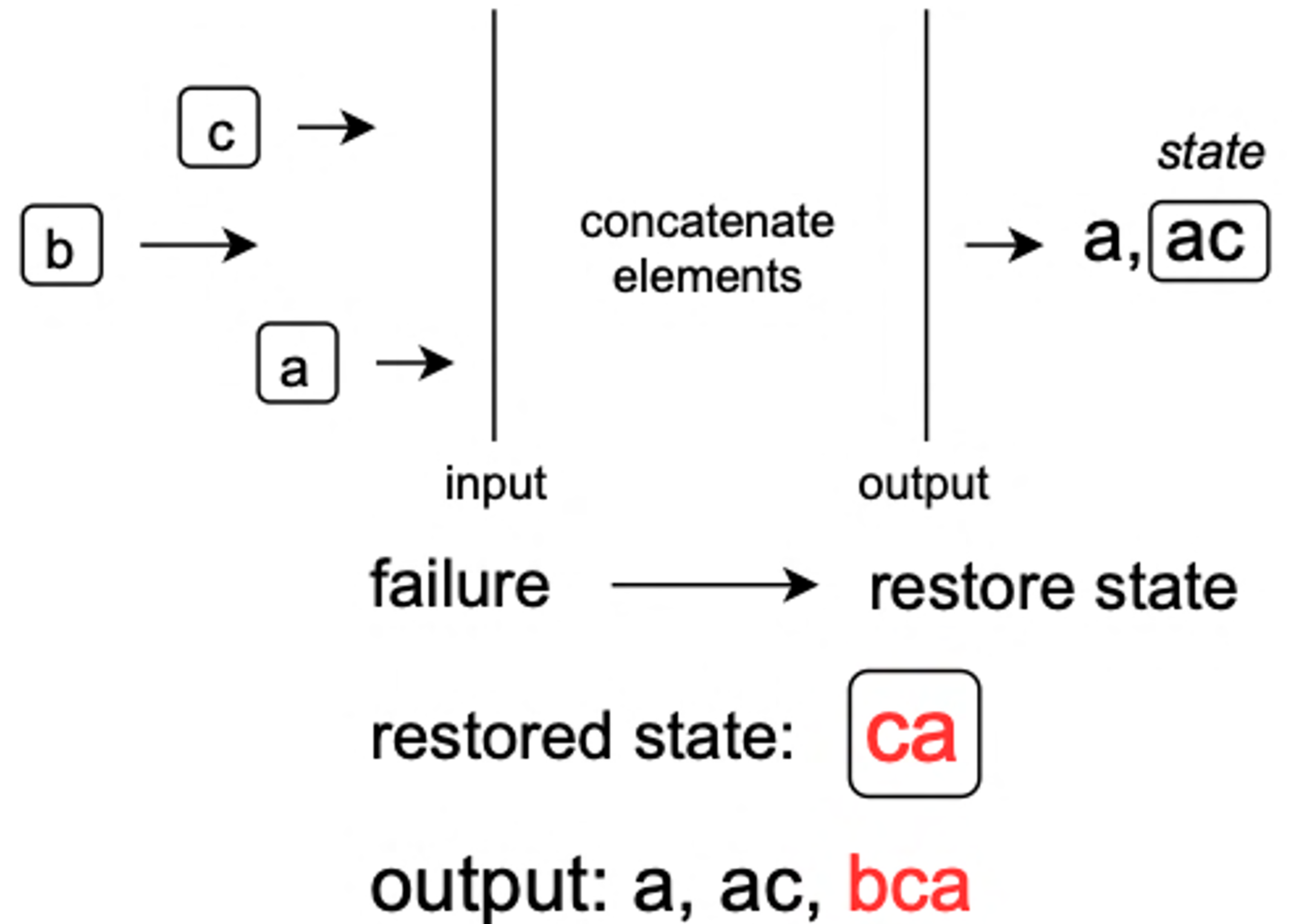
- Recovery process requires stop-the-world
- It is supposed that data source can re-send some input elements
- Latency directly depends on the snapshotting period for exactly-once
- At-least-once is efficient (but it has anomalies)

Model 2 properties: latency for exactly-once



Model 2 properties: at-least-once anomaly

- Output elements do not wait for commit in at-least-once guarantee
- If graph has a non-commutative operation, output elements can be inconsistent



Model 2 modification: deterministic processing

- At-least-once anomaly is caused by a non-deterministic order of elements processing
- If we could ensure deterministic processing, there will be no need to wait for commit in exactly-once
- How can we make processing deterministic?

How to ensure determinism?

- One way is to log all non-deterministic actions and to replay them (elements order, random generators, etc)

- If operations are pure, one can buffer input elements and sort them when the order is ensured

Research Data Management Track Paper SIGMOD '21, June 20–25, 2021, Virtual Event, China

Clonos: Consistent Causal Recovery for Highly-Available Streaming Dataflows

Pedro F. Silvestre Marios Fragkoulis Diomidis Spinellis Asterios Katsifodimos
Delft University of Technology
{P.F.Silvestre,M.Fragkoulis,D.Spinellis,A.Katsifodimos}@tudelft.nl

StreamBox: Modern Stream Processing on a Multicore Machine

Hongyu Miao[1], Heejin Park[1], Myeongjae Jeon[2],
Gennady Pekhimenko[2], Kathryn S. McKinley[3], and Felix Xiaozhu Lin[1]
[1]*Purdue ECE* [2]*Microsoft Research* [3]*Google*

Optimistic approach to ensure determinism

- Suppose that all operations in graph are pure
- Let us define a total order on data elements t(x)
- If elements are arrived properly ordered, worker processes them as usual
- If an element is out-of-order, worker invalidates state and previous output elements affected by this element and re-computes new state and new output elements

An optimistic approach to handle out-of-order events within analytical stream processing

Igor E. Kuralenok #1, Nikita Marshalkin #2, Artem Trofimov #3, Boris Novikov #4
JetBrains Research
Saint Petersburg, Russia
[1] ikuralenok@gmail.com [2] marnikitta@gmail.com [3] trofimov9artem@gmail.com [4] borisnov@acm.org

Part 3: How to choose guarantee?

- Bank transactions (maybe) require exactly-once
- For ML training at-least-once is sufficient in many cases
- For ML inference the choice of a guarantee highly depends on a specific problem

Distributed Classification of Text Streams: Limitations, Challenges, and Solutions

Artem Trofimov
Saint Petersburg State University / JetBrains Research
Saint Petersburg, Russia
trofimov9artem@gmail.com

Nikita Sokolov
ITMO university
Saint Petersburg, Russia
faucct@gmail.com

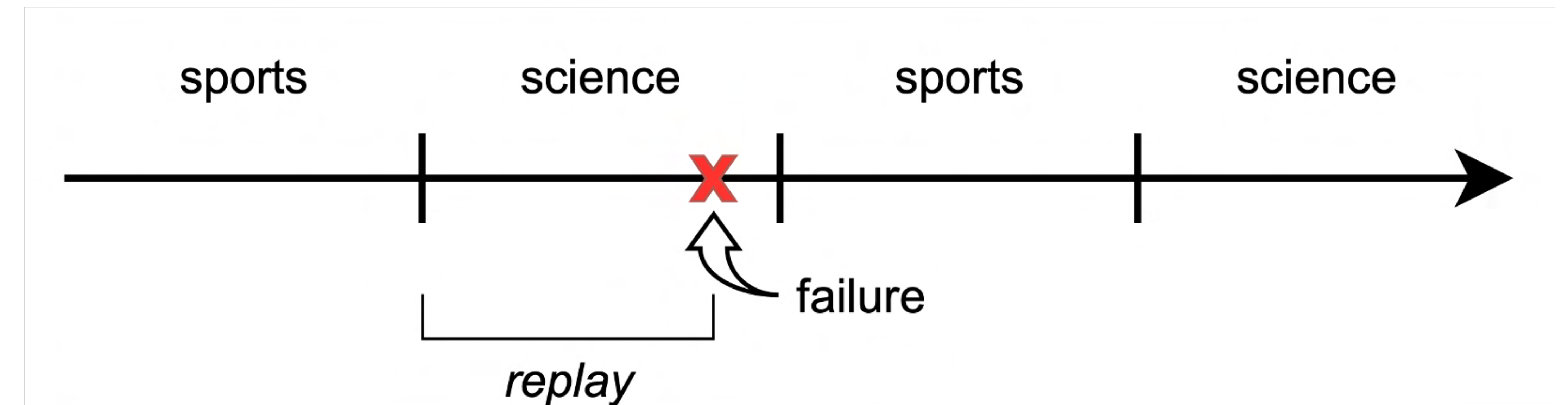
Mikhail Shavkunov
National Research University Higher School of Economics
Saint Petersburg, Russia
mv.shavkunov@gmail.com

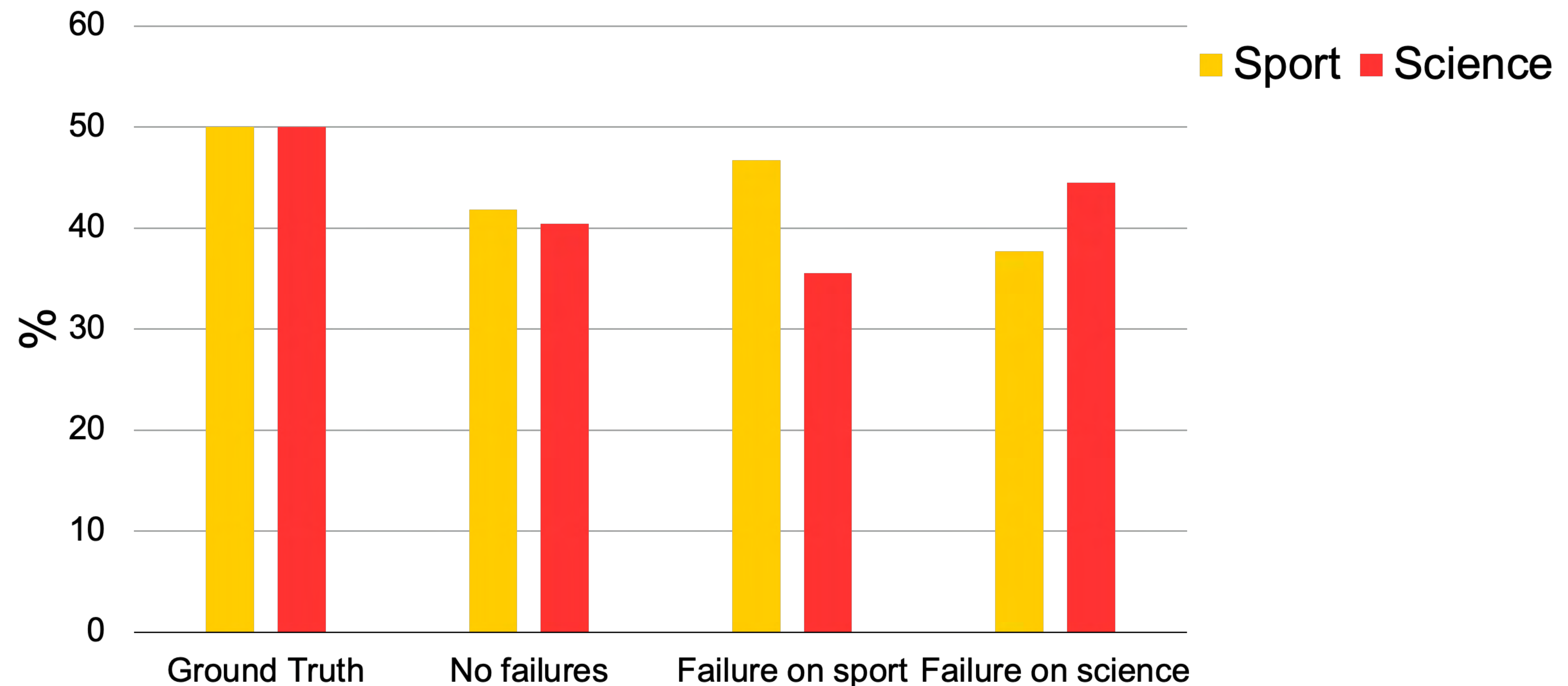
Igor Kuralenok
Yandex
Saint Petersburg, Russia
solar@yandex-team.ru

Boris Novikov
National Research University Higher School of Economics
Saint Petersburg, Russia
borisnov@acm.org

How to choose guarantee? At least once: example 1

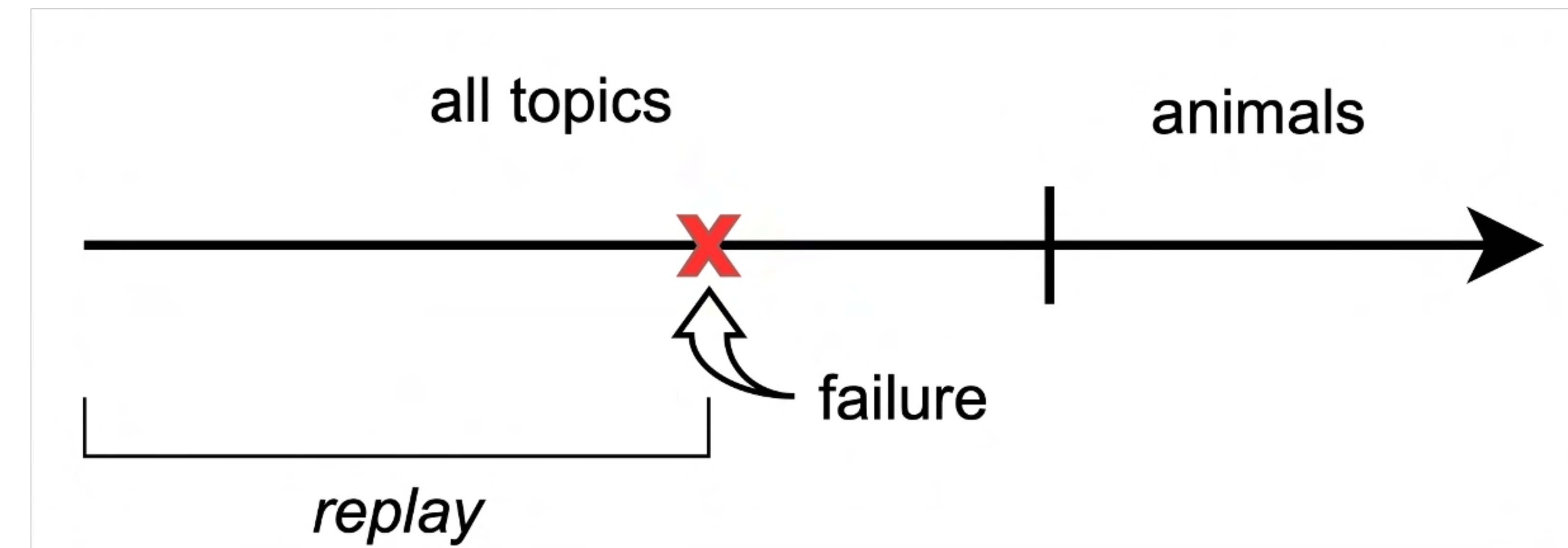
- 4000 articles (window)
- Sport and science topics
- 2 Amazon EC2 small instances

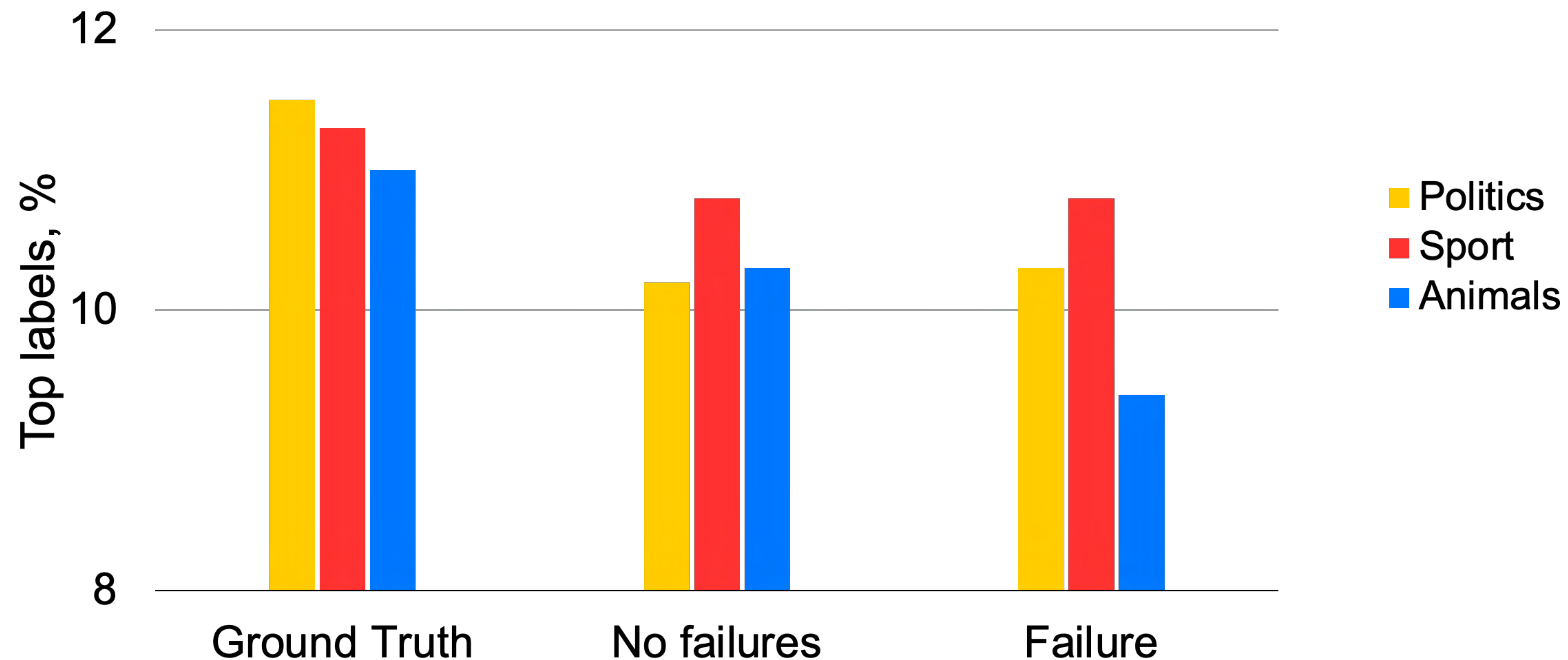




How to choose guarantee? At least once: example 2

- 5000 articles (window)
- Looking for “popular” topics
- 2 Amazon EC2 small instances





Conclusion

- It is hard to work with unbounded streams but we can divide them into substreams
- Punctuations is the standard technique for substreams management but it is inefficient in case of a large number of nodes or substreams
- Tracker is more suitable for large substreams number but its implementation is more complex
- Exactly-once guarantee affects latency if stream processing system applies global checkpointing model (e.g., Flink)
- Exactly-once = at-least-once + deduplication, if a system is deterministic
- There are multiple ways to ensure determinism but they can affect throughput