Beyond Analytics

The Evolution of Stream Processing Systems

Time, order, and progress

Paris Carbone, Marios Fragkoulis, Vasiliki Kalavri, Asterios Katsifodimos

Slides: **streaming-research.github.io/Tutorial-SIGMOD-2020**

Tutorial overview

- Part I: Introduction & Fundamentals (Vasia)
- Part II: Time, Order, & Progress (Marios)
- Part III: State Management (Paris)
- Part IV: Fault Recovery & High Availability (Marios)
- Part V: Load Management & Elasticity (Vasia)
- Part VI: Prospects (All)

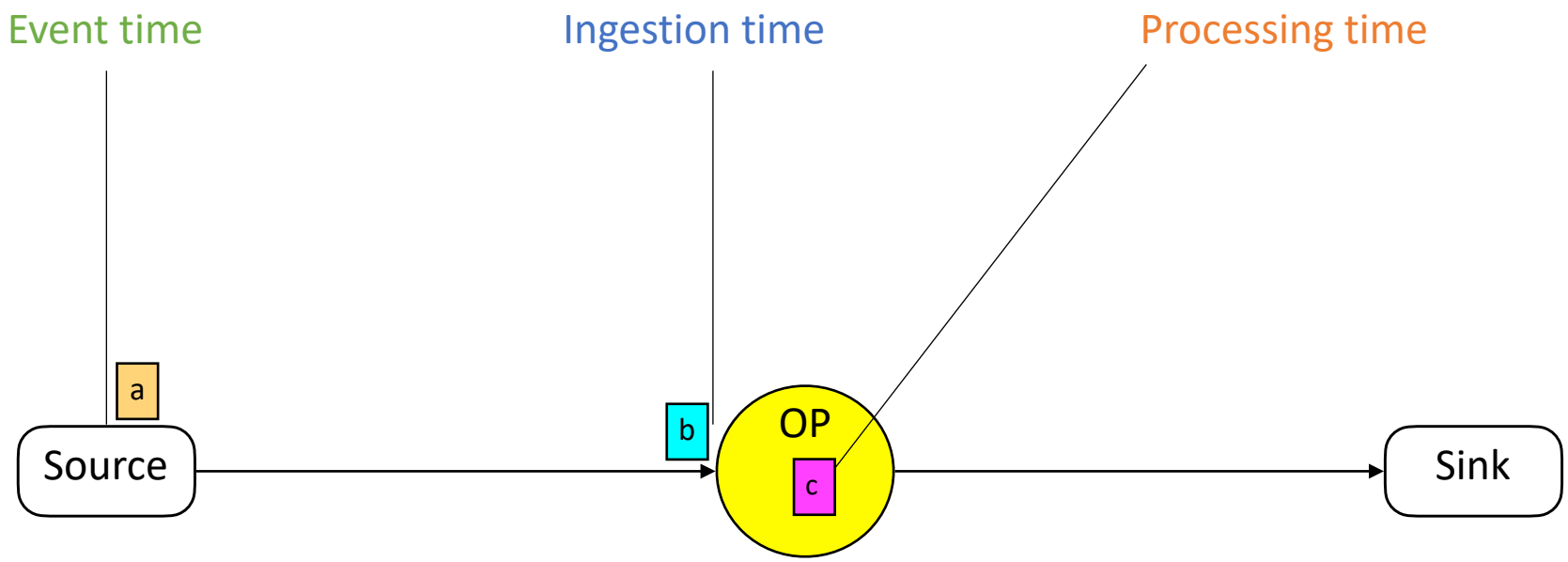
Overview

- Notions of time in stream processing
- Processing progress
- Out-of-order data management
- Putting it all together
- Open problems

Notions of time

- Event time
  - is the time tuples are generated at the sources. Also called application time.
- Processing time
  - is the time tuples are processed in a streaming system.
- Ingestion time
  - is the time tuples arrive in a streaming system.

Notions of time by example



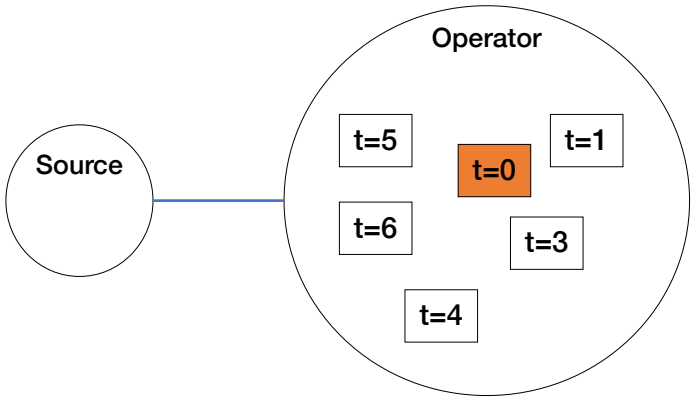
Flexible time management (PODS '04)
Google Dataflow (VLDB '15)

Processing progress in stream processing

- Assuming that a stream is ordered by one of its attributes *A* in increasing order, then the processing of the stream progresses when the smallest value of A among the unprocessed tuples increases over time
- *A* is called a *progressing attribute*
- A popular progressing attribute is event time timestamp

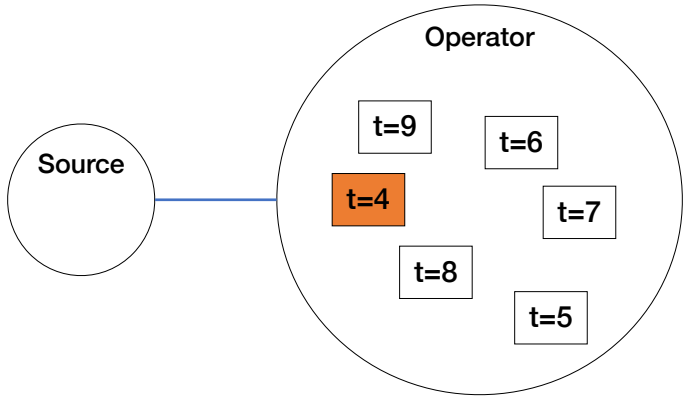
Processing progress by example

Progressing attribute:
t=0



later

Progressing attribute:
t=4



Out-of-order data management

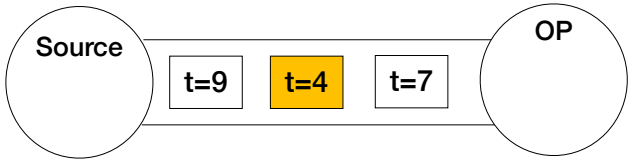
Overview

- Definition
- Causes of disorder
- System architectures
- Effects of disorder
- Progress-tracking mechanisms
- Revision processing

Out-of-order data

Definition

Out-of-order data tuples arrive in a streaming system after tuples with later event time timestamps.



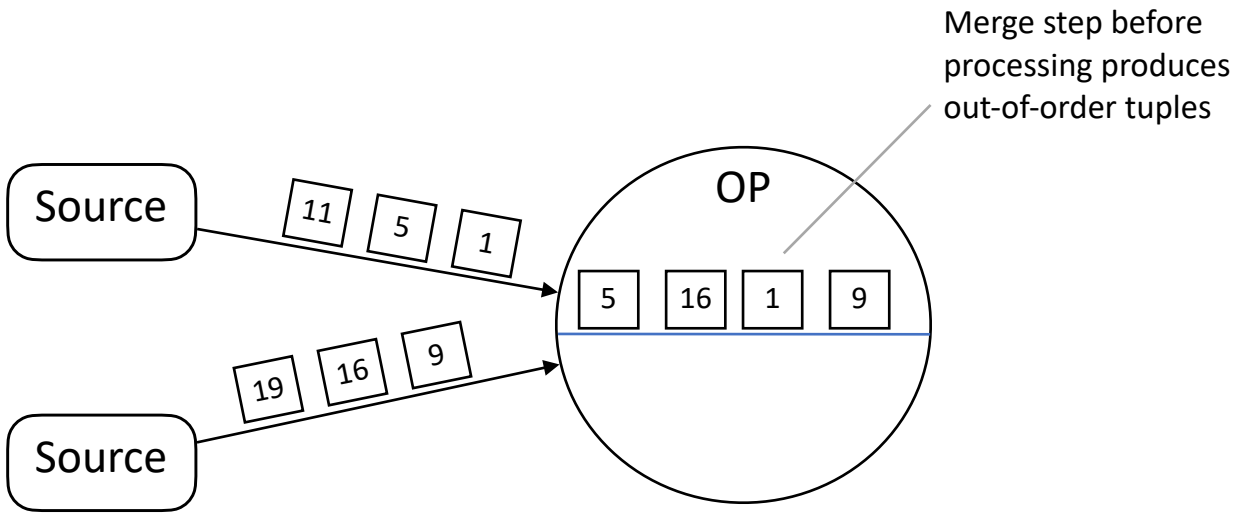
Causes of disorder

- External stochastic factors
- System operations

Causes of disorder

External factors

- External stochastic factors
  - Network routing
  - Multiple input sources

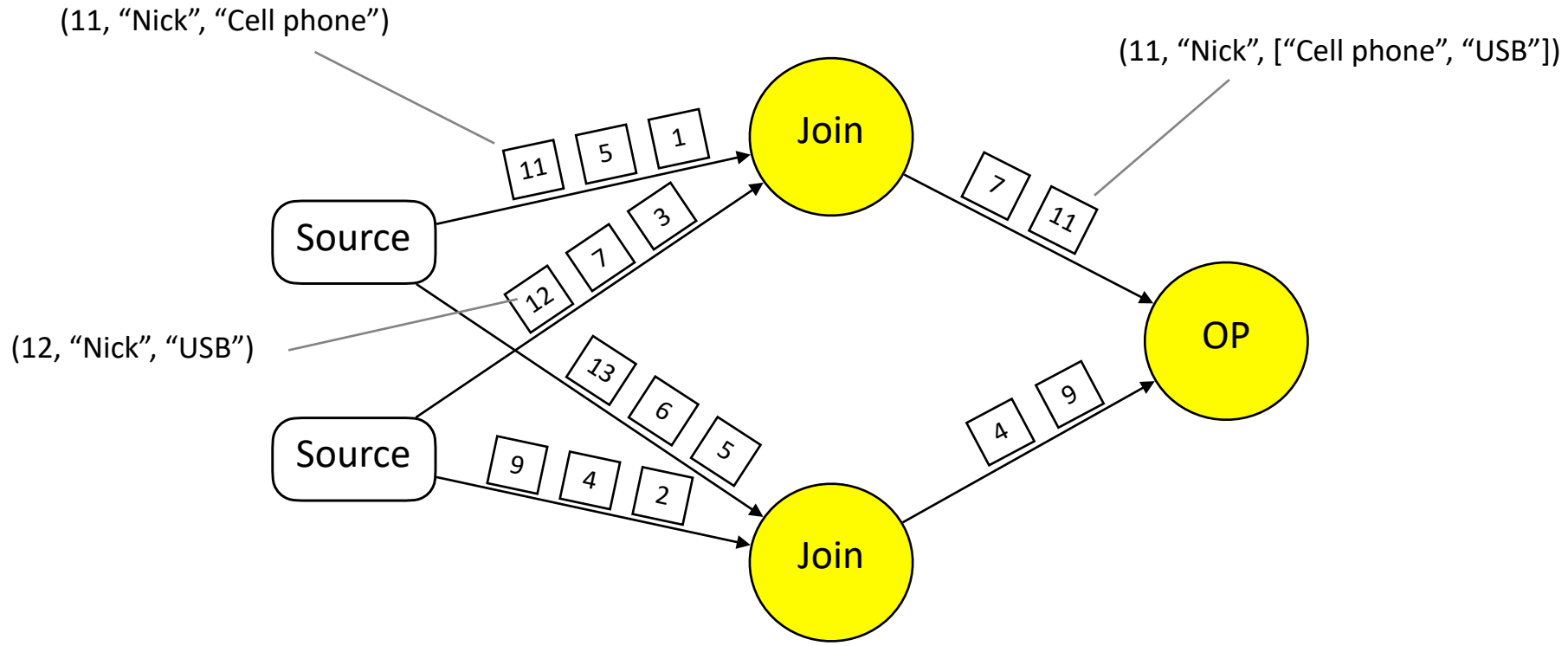


Causes of disorder

System operations

- System operations
  - Parallel join operator produces a shuffled combination of the two joined streams
  - Union operator on two unsynchronized streams yields a stream with all tuples of the two input streams interleaving each other in random order
  - Windowing based on an attribute that is different to the ordering attribute reorders the stream
  - Data prioritization using an attribute different to the ordering one changes the order

Disorder caused by system operation



System architectures

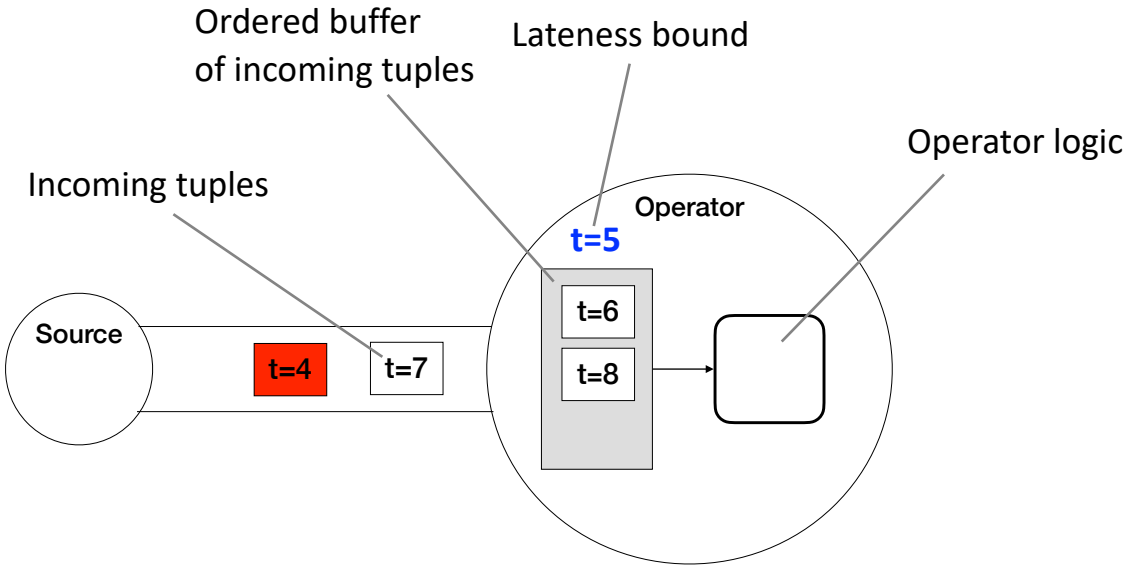
- In-order architecture
- Out-of-order architecture

In-order architecture

- In-order architecture
  - Buffer and reorder tuples
  - Forward the reordered tuples for processing
  - Clear the corresponding buffer slots

In-order architecture

- Buffer incoming tuples
- Reorder incoming tuples
- Proceed tuples to processing according to a lateness bound and ignore tuples that arrive to the operator after that

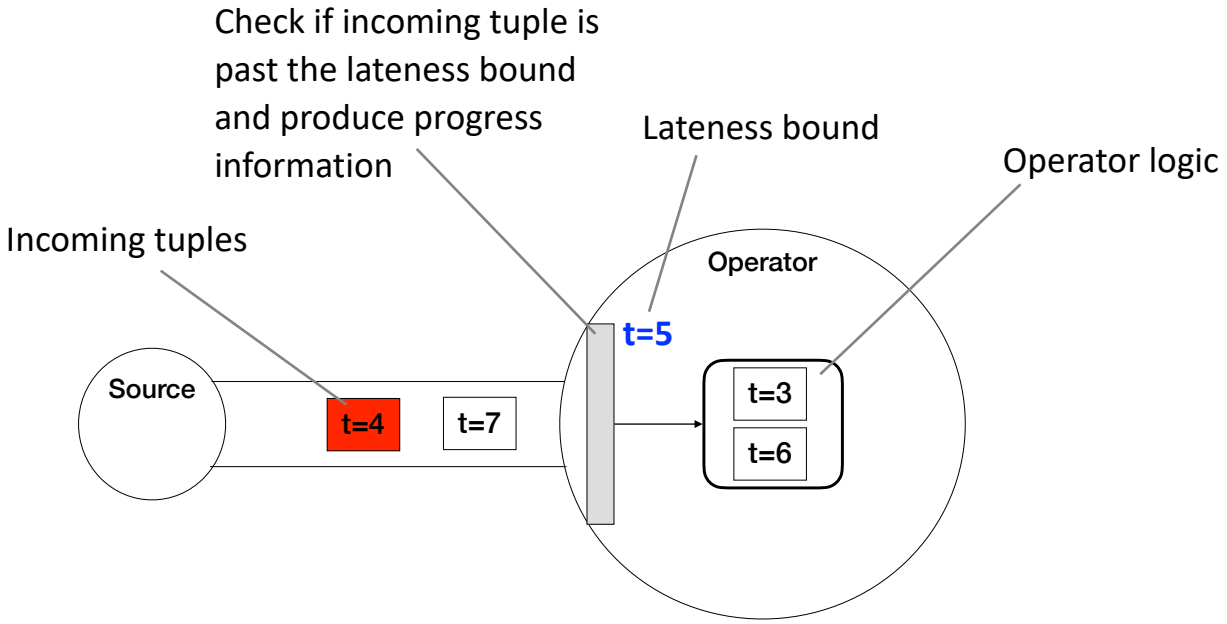


Out-of-order architecture

- Out-of-order architecture
  - Operators or a global authority produce progress information using some progress metrics and propagate it to the dataflow graph
  - The progress information typically reflects the oldest unprocessed tuple in the system and establishes a lateness bound for admitting out-of-order tuples
  - In contrast to in-order systems, tuples are processed in the order of their arrival up to the lateness bound

Out-of-order architecture

- Admit incoming tuples that are not past the lateness bound and ignore the rest
- The lateness bound typically reflects the oldest pending work
- Update progress information
- Propagate progress information to the data flow graph



Effects of disorder

- Leads to wrong results if ignored
  - Dropping a tuple that arrived after its time will make a join computation incorrect
- Impedes processing progress for order-sensitive operators (join, aggregate)
  - In-order architecture systems
    - Buffer and reorder data as they come
    - Add processing overhead, memory space overhead, and latency
  - Out-of-order architecture systems
    - Establish bound based on processing progress and process tuples since that point without reordering
    - Stock processing state
    - Add implementation complexity
- Except for order-agnostic operators
  - project, filter, map, dupelim, and union

Progress-tracking mechanisms

Overview

- Slack
- Heartbeat
- Low-watermark
- Pointstamp and frontier

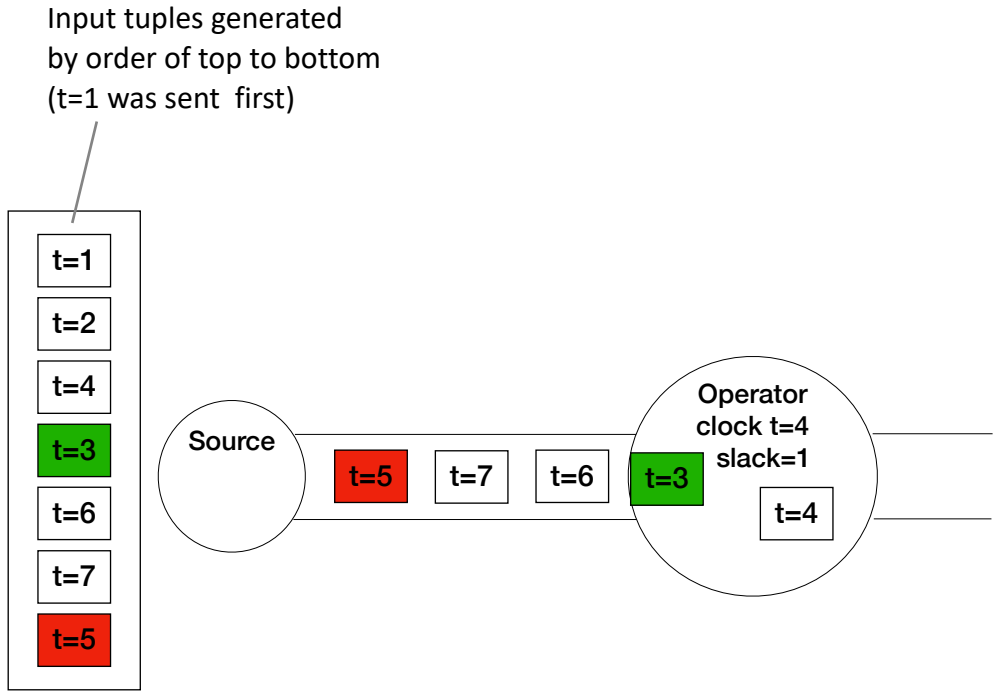
Slack

- Involves waiting for out-of-order data for a fixed amount of a certain metric
- Originally denoted the number of tuples intervening between the actual occurence of an out-of-order tuple and the position it would have in the input stream if it arrived on time
- Can also be quantified in terms of time
- Slack marks a fixed grace period for late tuples

Slack in action

4-second event time tumbling window count aggregation

- Close first window [0,4) when t=3 arrives
- Normally window would close when t=4 arrives, but because of slack=1 window closing awaits the next tuple that will make the slack expire
- Because t=3 arrives it is included in the window
- The window will output C=3 for t=1, t=2, and t=3
- Admit t=3 because of slack=1



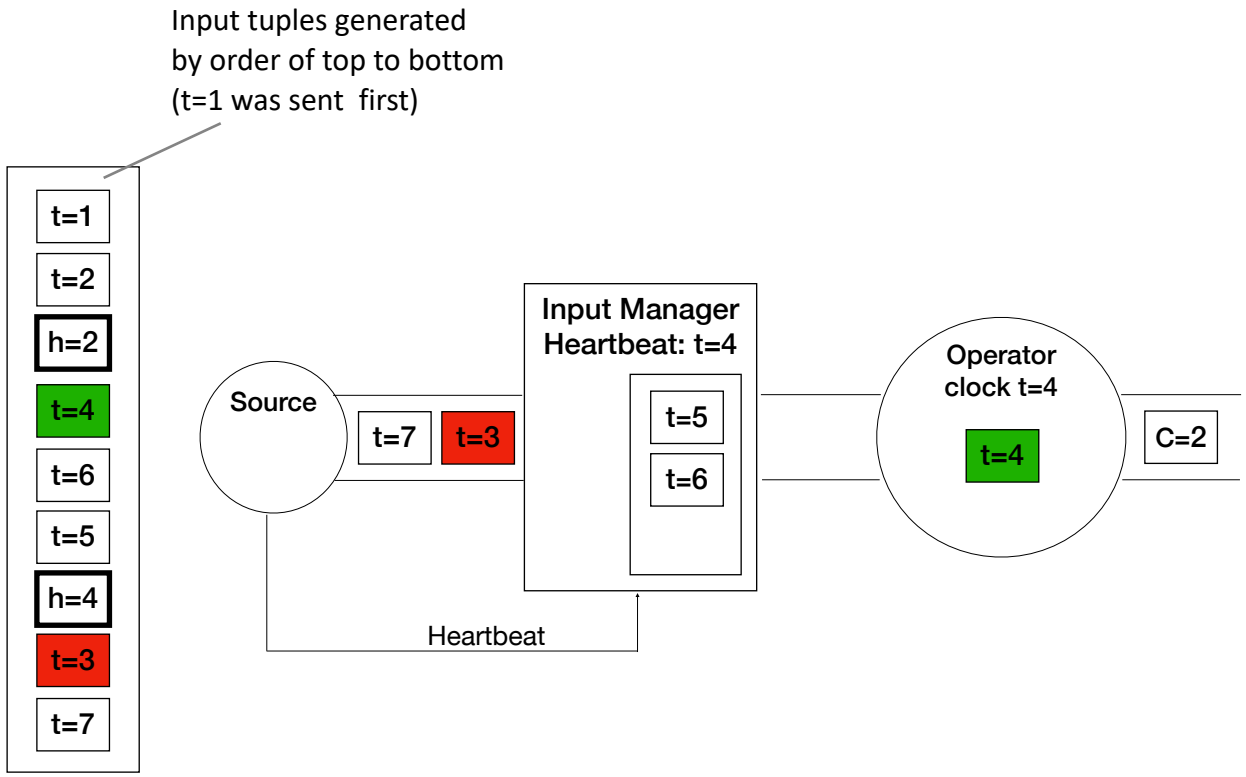
Heartbeat

- External signal carrying progress information about a data stream
- Contains a timestamp indicating that all succeeding stream tuples will have a more recent timestamp than the heartbeat's timestamp
- Otherwise they are dropped.
- Heartbeats can be generated by an input source
- A streaming system can deduce heartbeats by observing environment parameters

Heartbeat in action

4-second event time tumbling window count aggregation

- First heartbeat t=2 not shown here allows t=1 and t=2 to proceed from the input manager to the operator
- Second heartbeat t=4 allows t=4 to proceed from the input manager to the operator
- Close first window [0,4) when t=4 arrives and output C=2 for t=1 and t=2.



Low-watermark

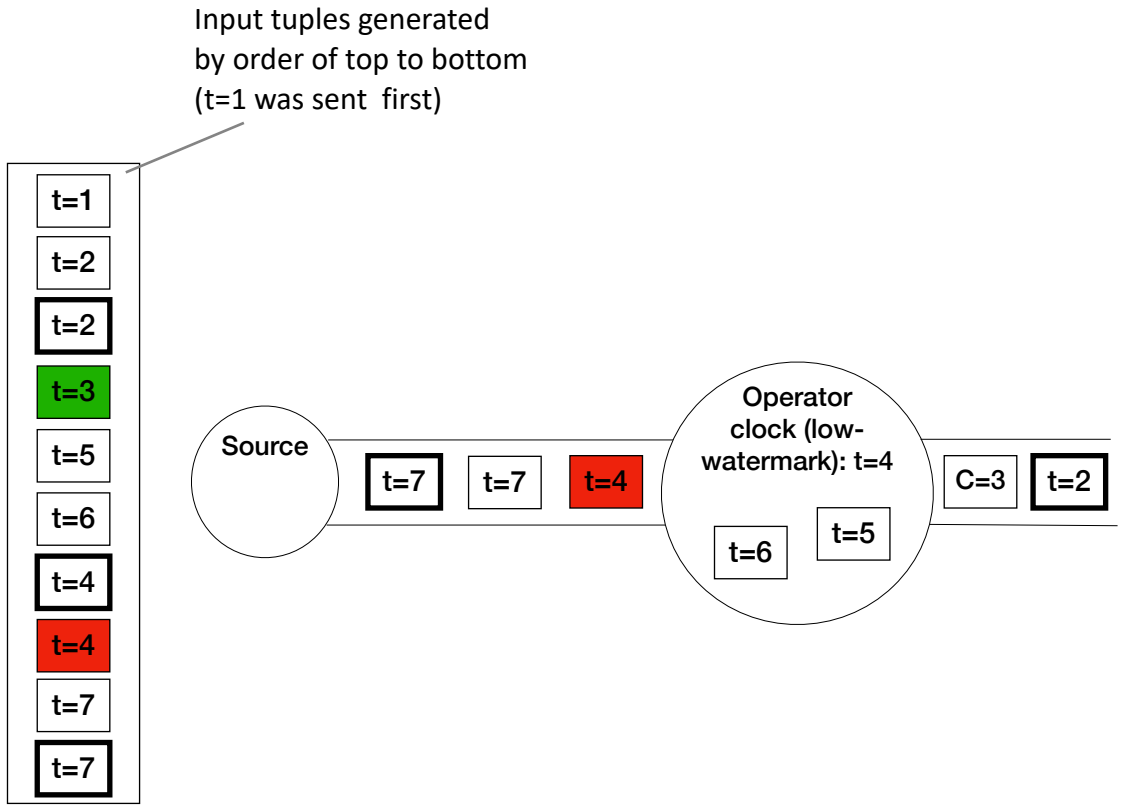
- The low-watermark for an attribute of a stream is the lowest value of that attribute within a certain subset of the stream
- Future tuples will probabilistically bear a higher value than the current low-watermark for the same attribute
- The mechanism is used by a streaming system to process data past the low-water mark for an attribute, e.g. an aggregate grouped by the attribute, or to remove state that is maintained for the attribute, for instance, the corresponding hash table entries of a hash join computation

Out-of-order processing (VLDB ’08)
Google Dataflow (VLDB ’15)
Flink (IEEE. Eng. Bull. ’15)

Low-watermark in action

4-second event time tumbling window count aggregation

- Close first window [0,4) when low-watermark t=4 arrives
- Normally window would close when t=5 arrives, but because the low-watermark reflects the oldest pending work in the system it is the low-watermark that closes windows to cater for late data
- The window will output C=3 for t=1, t=2, and t=3
- Drop t=4 because it is not greater (more recent) than the low-watermark



Slack vs heartbeats

- Heartbeats and slack are both external to a data stream
- Heartbeats are signals communicated from an input source to a streaming system's ingestion point.
- Differently to heartbeats, which is an internal mechanism of a streaming system hidden from users, slack is part of the query specification provided by users

Heartbeats vs low-watermark

- Heartbeats and low-watermarks are similar in terms of progress-tracking logic
- While heartbeats address the progress of stream tuple generation at the input sources, the low-watermark extends this to the processing progress of computations in the streaming system by reflecting their oldest pending work
- The low-watermark generalizes the concept of the oldest value, which signifies the current progress point, to any progressing attribute of a stream tuple besides timestamps

Punctuations

- Metadata annotations embedded in data streams
- A punctuation is itself a stream tuple, which consists of a set of patterns each identifying an attribute of a stream data tuple
- For example: (*, *, *, *, *, 12:00:00AM)
- A punctuation is a generic mechanism that communicates information across the dataflow graph
- Regarding progress tracking, it provides a channel for communicating progress information such as a tuple attribute's low-watermark produced by an operator, event time skew, or slack
- Punctuations are useful in multiple other functional areas of a streaming system, such as state management, monitoring, and load management.

Punctuation semantics (IEEE TKDE ’03)

Pointstamp and frontier

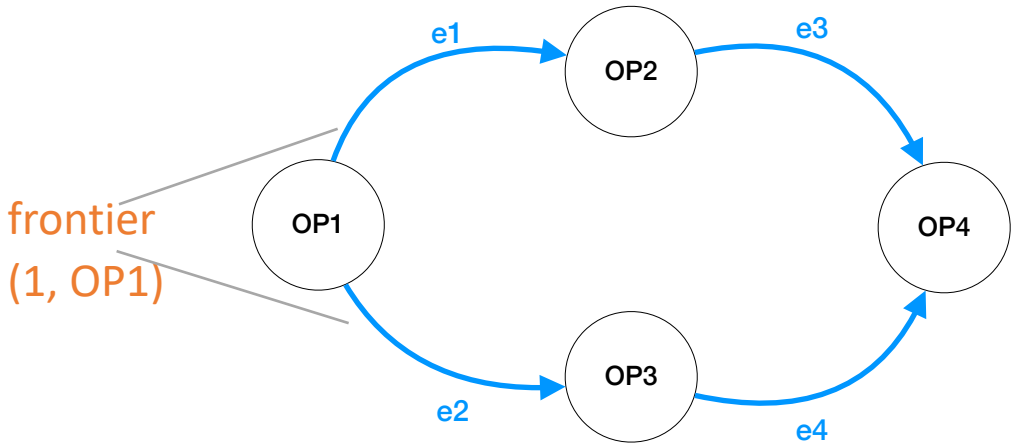
- Pointstamps are embedded in data streams, but a pointstamp is attached to each stream data tuple as opposed to a punctuation, which forms a separate tuple
- Pointstamps are pairs of timestamp and location that position data tuples on a vertex or edge of the dataflow graph at a specific point in time
- An unprocessed tuple *p* at a specific location *could-result-in* another unprocessed tuple *p`* with timestamp *t`* at another location when *p* can arrive at *p`* before or at timestamp *t`*.
- Unprocessed tuples *p* with timestamp *t* are in the *frontier* of processing progress when no other unprocessed tuples *could-result-in p.*
- Tuples bearing t or an earlier timestamp are processed and the frontier moves on
- This modeling of processing progress traces the course of data tuples on the dataflow graph with timestamps and tracks the dependencies between unprocessed events in order to compute the current frontier.
- The system enforces that future tuples will bear a greater timestamp than the tuples that generated them
- The concept of a frontier is similar to a low-water mark

Naiad (SOSP ’13)

Pointstamp and frontier in action

- The example includes three active pointstamps. Poinstamps are active when they correspond to one or more unprocessed events.
- Pointstamp (1, OP1) is in the frontier of active pointstamps because its precursor count is 0. The precursor count specifies the number of active pointstamps that *could-result-in* a following pointstamp. In the frontier, notifications for unprocessed events can be delivered. Thus, unprocessed events e1 and e2 can be delivered to OP2 and OP3 respectively. The occurrence count is 2 because both events e1 and e2 bear the same pointstamp.
- Looking at this snapshot of the data flow graph it is easy to see that pointstamp (1, OP1) *could-result-in* pointstamps (2, OP2) and (2, OP3). Therefore, the precursor count of the latter is 1.

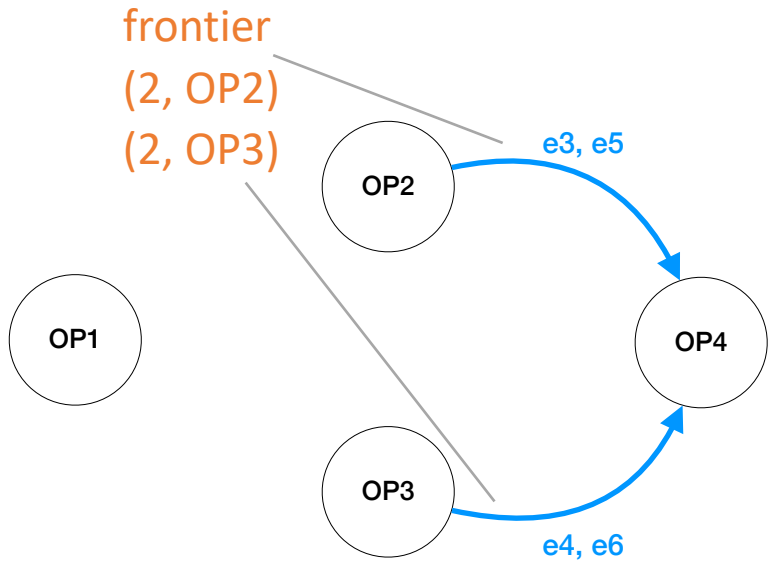
Active pointstamp	Unprocessed event(s)	Occurence count	Precursor count
(1, OP1)	e1, e2	2	0
(2, OP2)	e3	1	1 (1, OP1)
(2, OP3)	e4	1	1 (1, OP1)



Pointstamp and frontier in action

- After events e1 and e2 are delivered to OP2 and OP3 respectively, their processing results in the generation of new events e5 and e6, which bear the same pointstamp as unprocessed events e3 and e4 respectively.
- Since there are no more unprocessed events with timestamp 1 and the precursor count of pointstamps (2, OP2) and (2, OP3) is 0, then the frontier moves on to these active pointstamps. Consequently, all four event notifications can be delivered.
- Obsolete pointstamps (1, OP1), (2, OP2), and (2, OP3) are removed from their location.

Active pointstamp	Unprocessed event(s)	Occurence count	Precursor count
(2, OP2)	e3, e5	1	0
(2, OP3)	e4, e6	1	0



Revision processing

Overview

- Definition
- Mechanisms
  - Triggers
- Strategies
  - Store and revise
  - Replay and revise
  - Partition and consolidate
- Models
  - The Google Dataflow model (VLDB ’15)
- Both in-order and out-of-order systems can support revision processing

Putting it all together: Out-of-order data management in streaming systems

System	Architecture	Mechanism	Revision processing
Aurora*	In-order	Slack	
STREAMS	In-order	Heartbeat	
Borealis	Out-of-order	History bound	✓
Millwheel	Out-of-order	Low-watermark	
Naiad	Out-of-order	Pointstamp	✓
Trill	In-order	Low-watermark	
Flink	Out-of-order	Low-watermark	✓
Dataflow	Out-of-order	Low-watermark	✓
Spark	Out-of-order	Slack	✓

Vintage vs modern

- The problem of managing order in unbounded data became obvious very early
- Most of the solution aspects (architectures and mechanisms presented were also developed early
- From then on, they have mainly been combined with each other
- Some like out-of-order processing, low-water marks, and triggers received traction many years later by MillWheel (VLDB ’13), Google Dataflow Model (VLDB ’15), Apache Flink (IEEE Eng. Bull. ’15), etc.

Open problems

- In-order vs out-of-order architecture
  - Cost vs implementation complexity
- Streaming applications may receive unbounded data from multiple sources that include different notions of event time
  - Streaming systems do not currently support multiple time domains
- Data sources with disparate latency characteristics result in unaligned watermarks that challenge progress tracking

References

- D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDBJ*, 2003.
- P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE TKDE*, 2003.
- U. Srivastava and J. Widom. Flexible time management in data stream systems. In *PODS*. ACM, 2004.
- J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, and D. Maier. Out-of-order processing: A new architecture for high-performance stream systems. *In VLDB*, 2008.
- T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle. MillWheel: Fault-tolerant stream processing at internet scale. In *VLDB*, 2013.
- D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A timely dataflow system. In *SOSP*, 2013.
- T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, un-bounded, out-of-order data processing. *In VLDB*, 2015.
- P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38:28–38, 2015.

Thanks for viewing! And don't forget:

1. Streaming systems need to support computations based on event time and processing time and expose that option to the API

2. Out-of-order data is common occurrence. Dealing with them is a foundational aspect in streaming systems and an architectural concern.

3. There are four main progress tracking mechanisms for measuring processing progress: slack, heartbeat, low-watermark, and pointstamps. Low-watermarks are the most popular.