Chapter 11 of [DDIA] & Streaming 101

Streaming Processing

Outline

- Background
  - Terminology
  - Capabilities
  - Time Domains
- The What, Where, When and How of Data Processing
  - *What* results are calculated
  - *Where* in event time are results calculated
  - *When* in processing time are results materialized
  - *How* do refinements of results relate
- Stream and Table
- Flink Applications in eBay

Background - terminology

- Bounded data
  - A type of dataset that is finite in size.
- Unbounded data
  - A type of dataset that is infinite in size (at least theoretically).
- Table
  - A holistic view of a dataset at a specific point in time. SQL systems have traditionally dealt in tables.
- Stream
  - An element-by-element view of the evolution of a dataset over time. The MapReduce lineage of data processing systems have traditionally dealt in streams.

Background - terminology

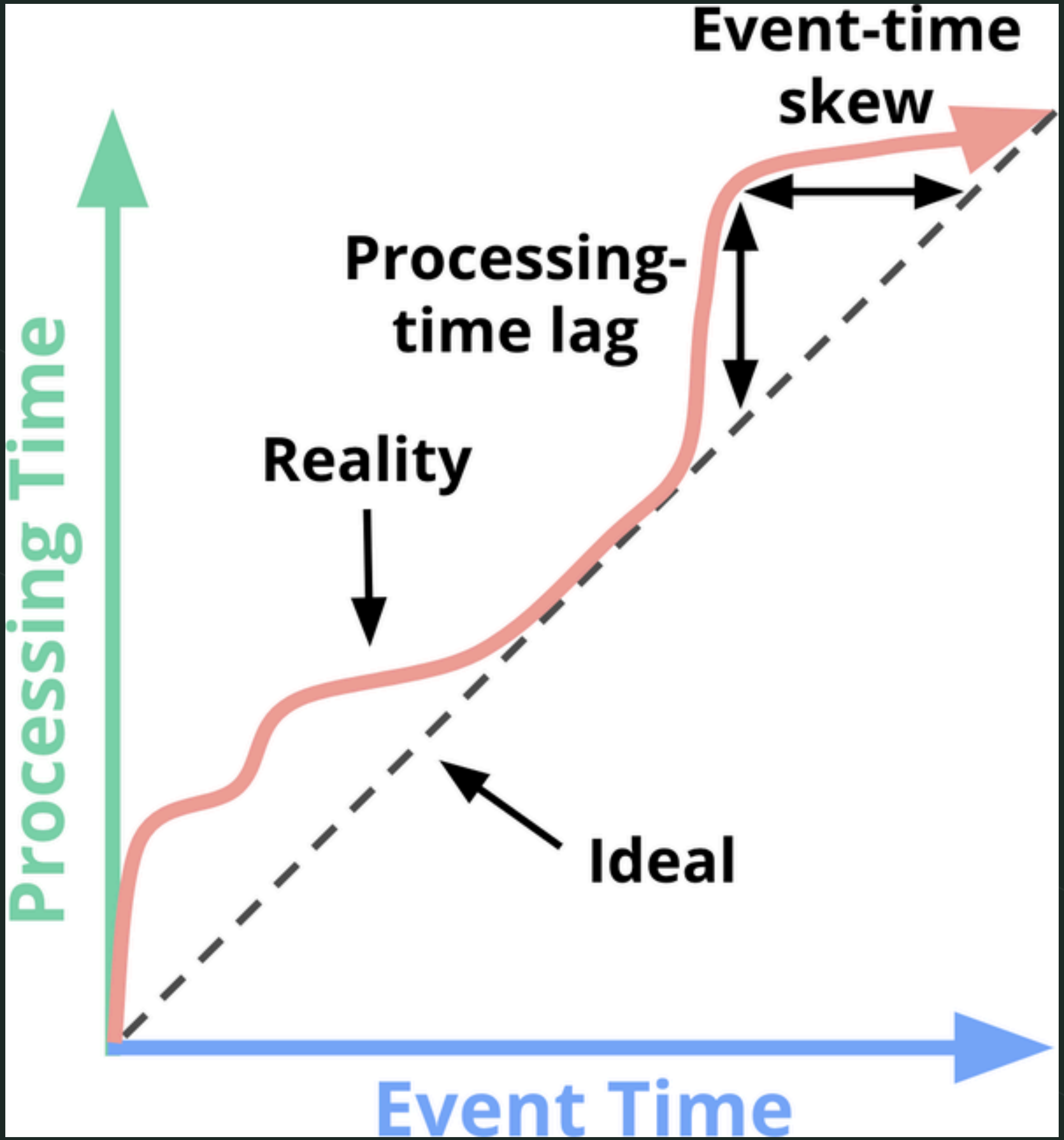
- Streaming system
    - A type of data processing engine that is designed with infinite datasets in mind.

Background - Capabilities

- Correctness
  - This gets you parity with batch.
- Tools for reasoning about time
  - This gets you beyond batch.

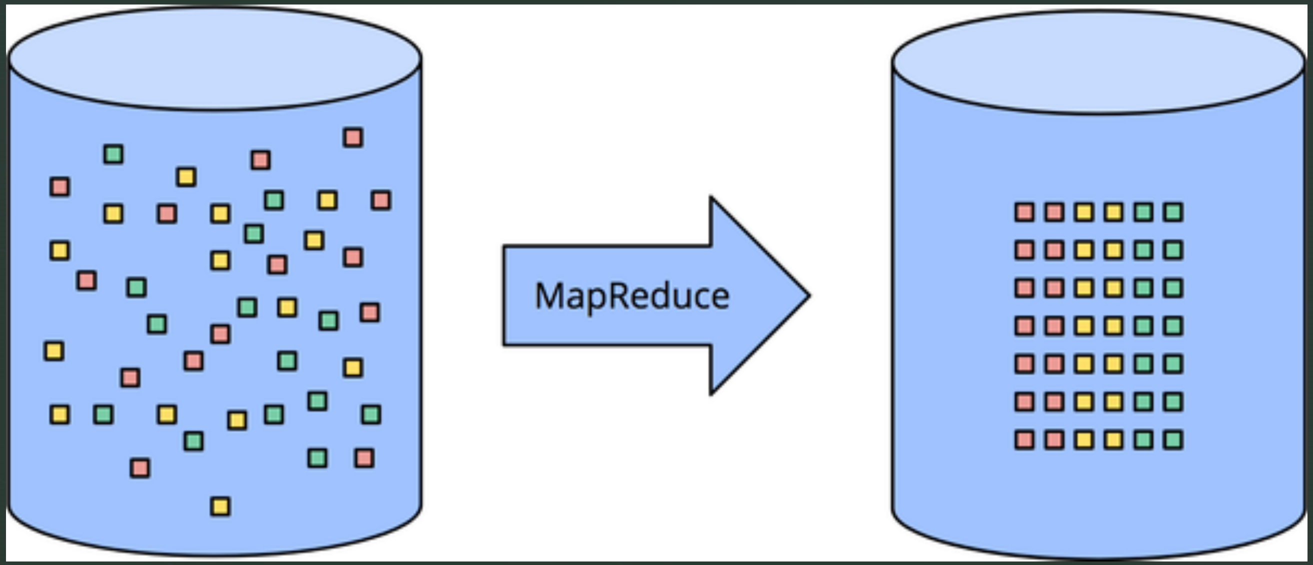
Background – time domains

- Event time
  - This is the time at which events actually occurred.
- Processing time
  - This is the time at which events are observed in the system.


Event-time skew
Processing-time lag
Reality
Ideal
Processing Time
Event Time

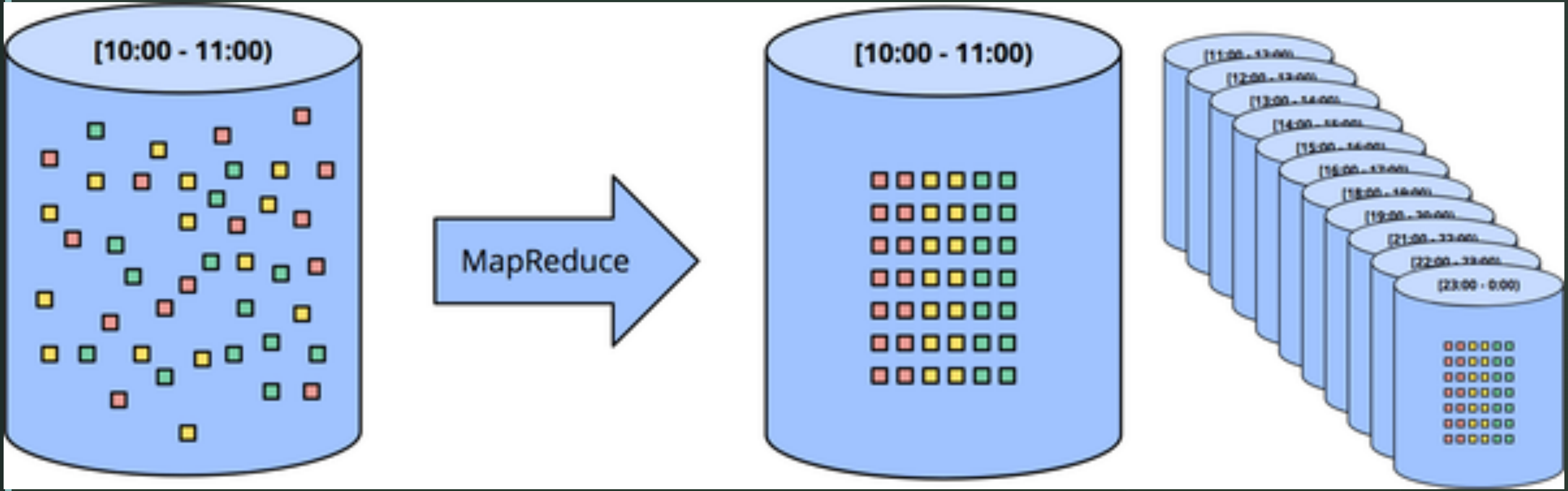
Data Processing Patterns

- Bounded data
- Unbounded Data: Batch
  - Fixed windows
  - Sessions
- Unbounded Data: Streaming



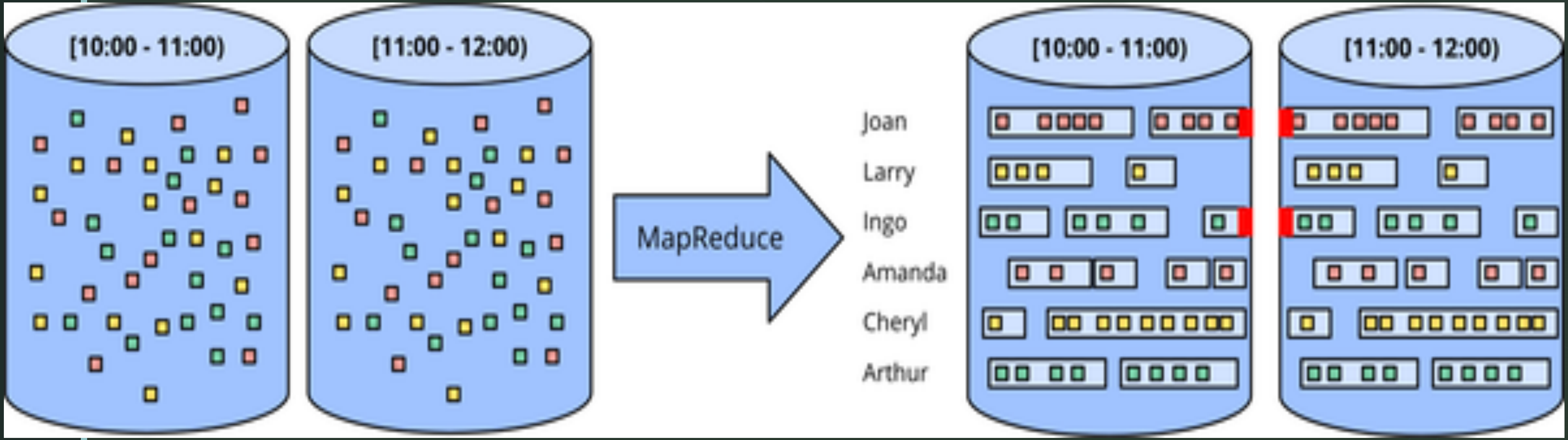
Bounded Data

Unbounded Data: Batch
- Fixed Window



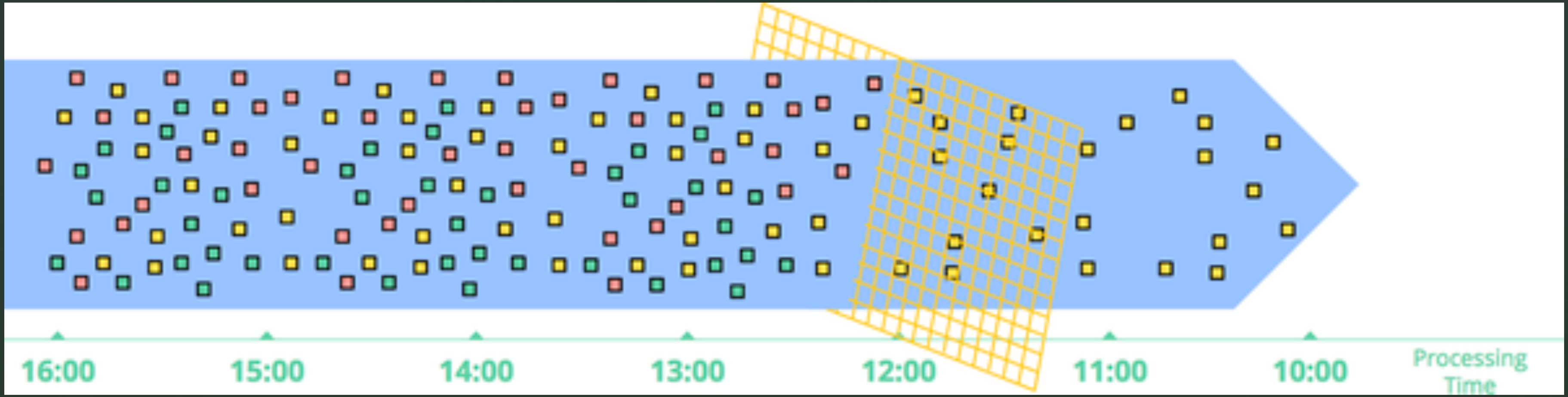
Unbounded Data: Batch
- Session

- An unbounded dataset is collected up front into finite, fixed-size windows of bounded data that are then subdivided into dynamic session windows via successive runs a of classic batch engine.



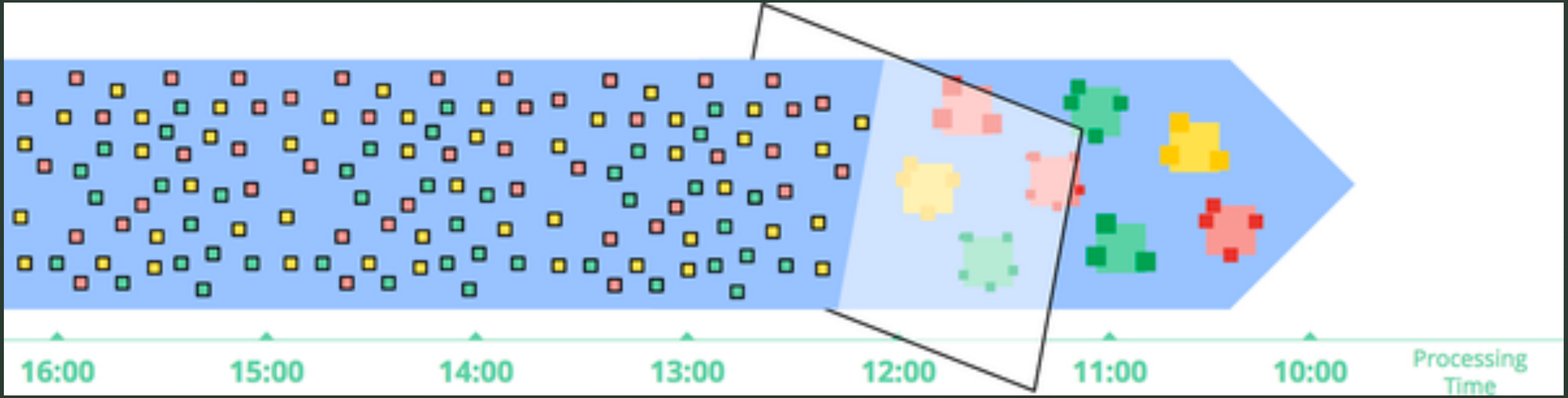
Unbounded Data: Streaming
- time-agnostic

- Filtering a certain domain traffic
- A collection of data (flowing left to right) of varying types is filtered into a homogeneous collection containing a single type.



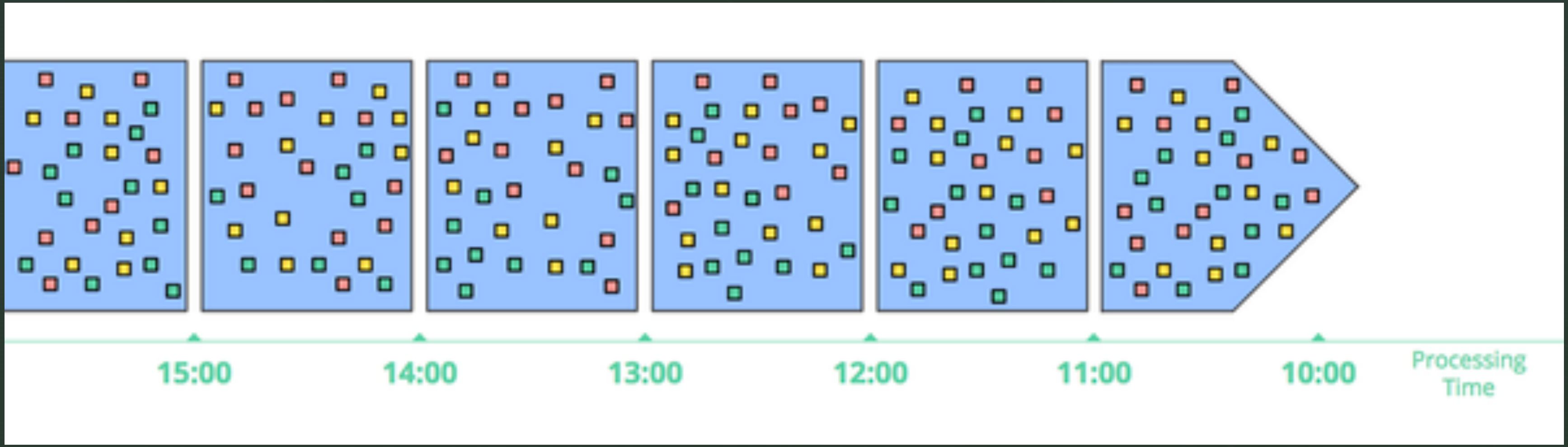
Unbounded Data: Streaming
- approximation algorithms

- Data are run through a complex algorithm, yielding output data that look more or less like the desired result on the other side.



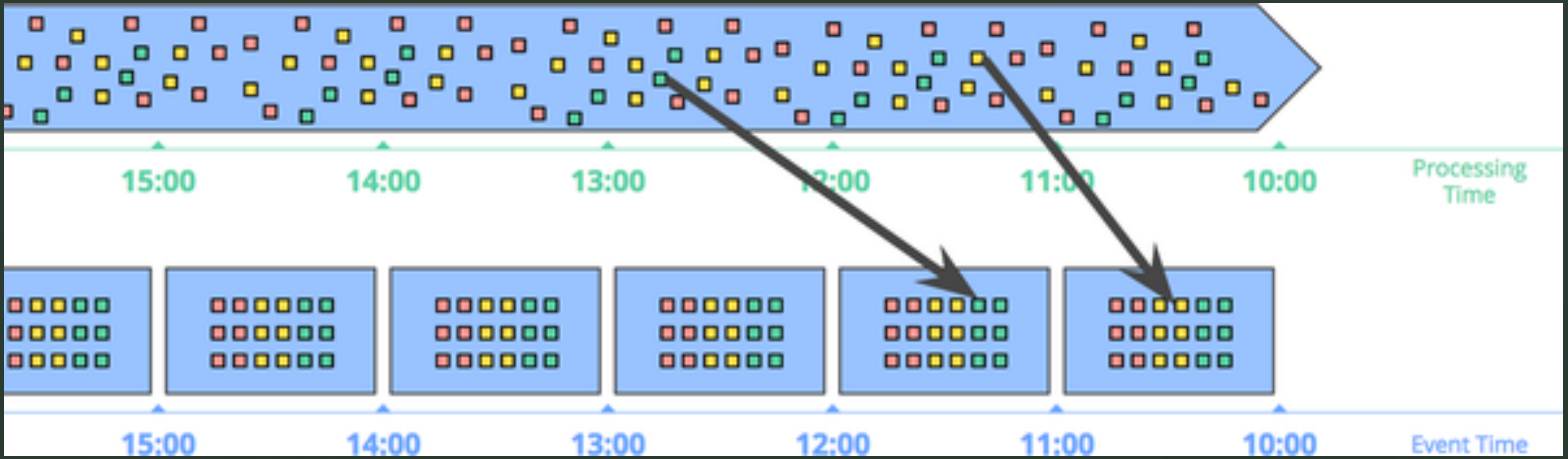
Unbounded Data: Streaming
- windowing by processing time

- Windowing into fixed windows by processing time. Data are collected into windows based on the order they arrive in the pipeline.



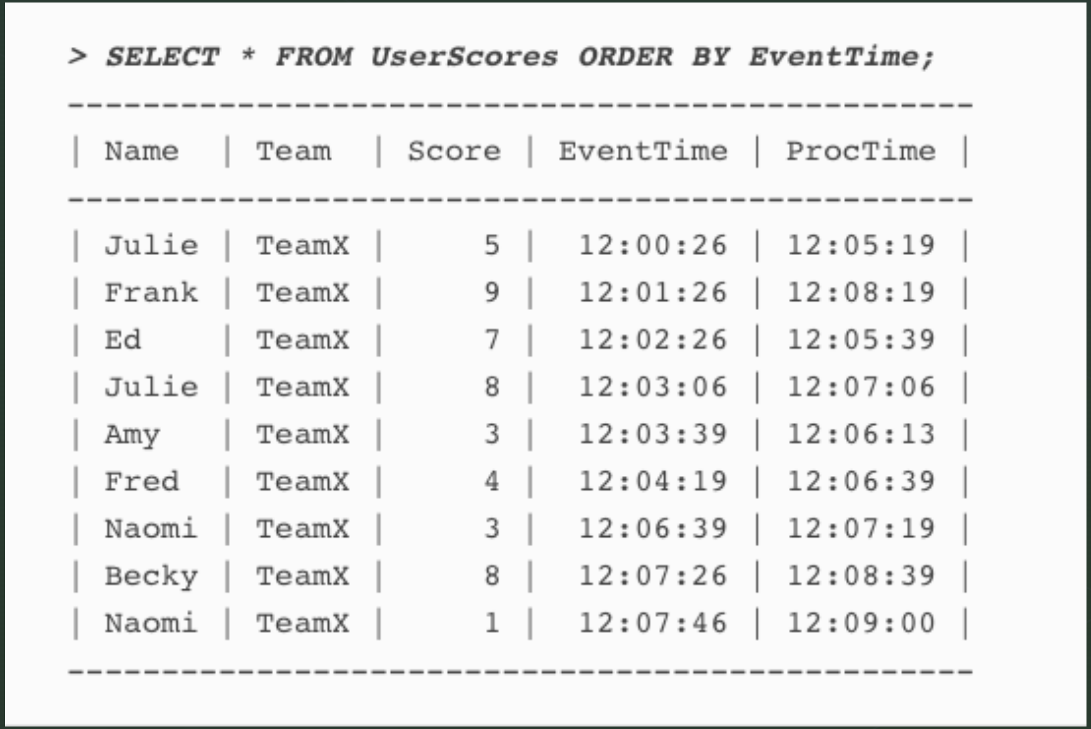
Unbounded Data: Streaming
- windowing by event time

- Windowing into fixed windows by event time. Data are collected into windows based on the times at which they occurred. The black arrows call out example data that arrived in processing-time windows that differed from the event-time windows to which they belonged.

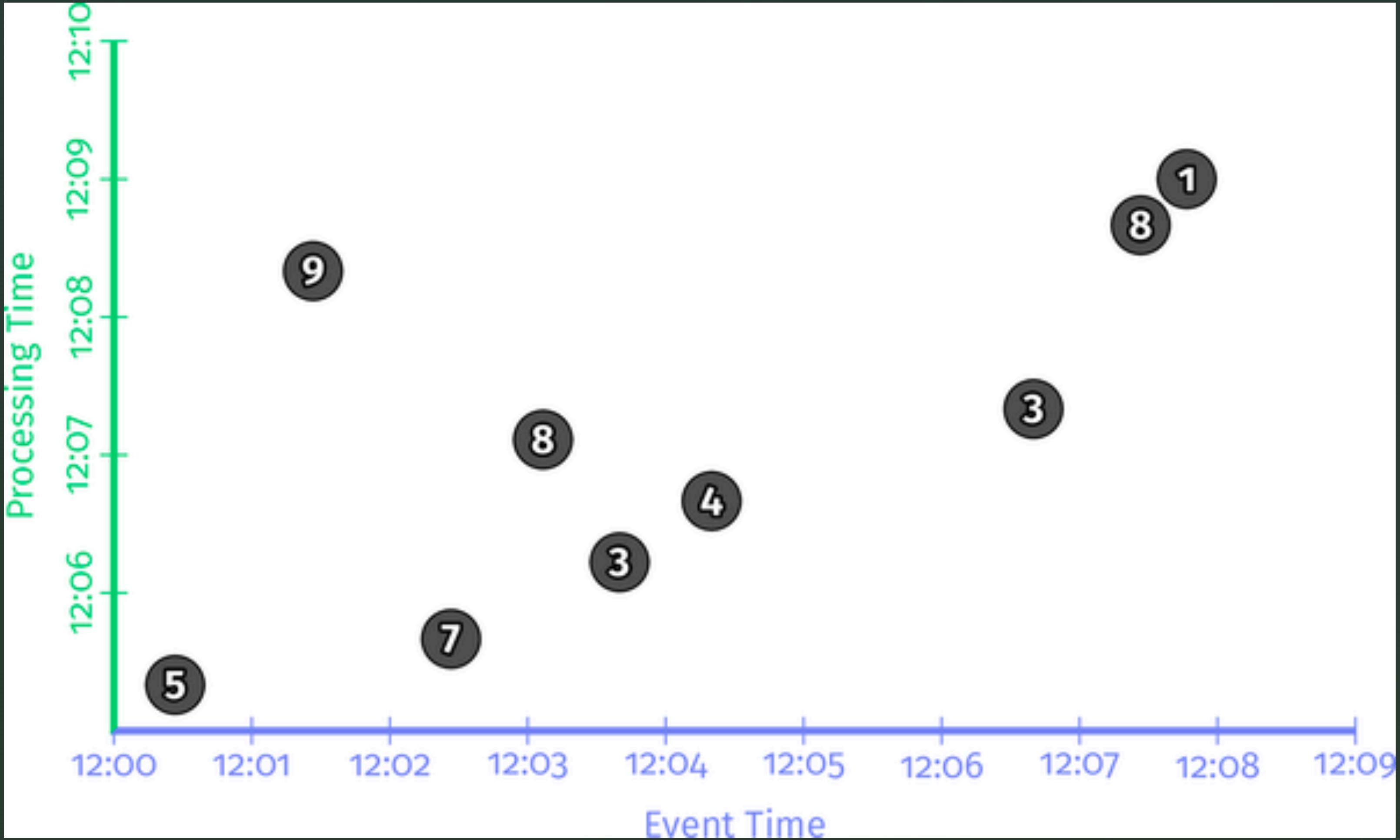


What results are calculated: Transformations

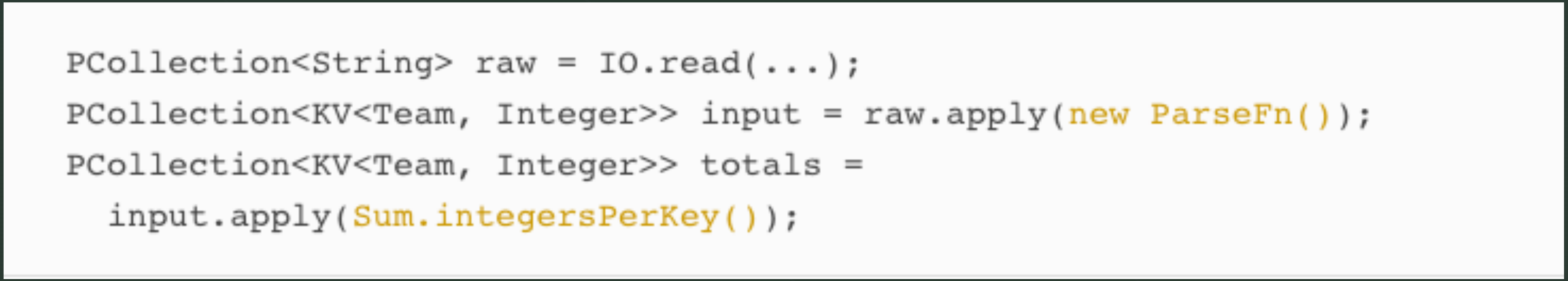
- Let's imagine that we've written a team-based mobile game and we want to build a pipeline that calculates team scores by summing up the individual scores reported by users' phones. If we were to capture our nine example scores in a SQL table named "UserScores," it might look something like this:



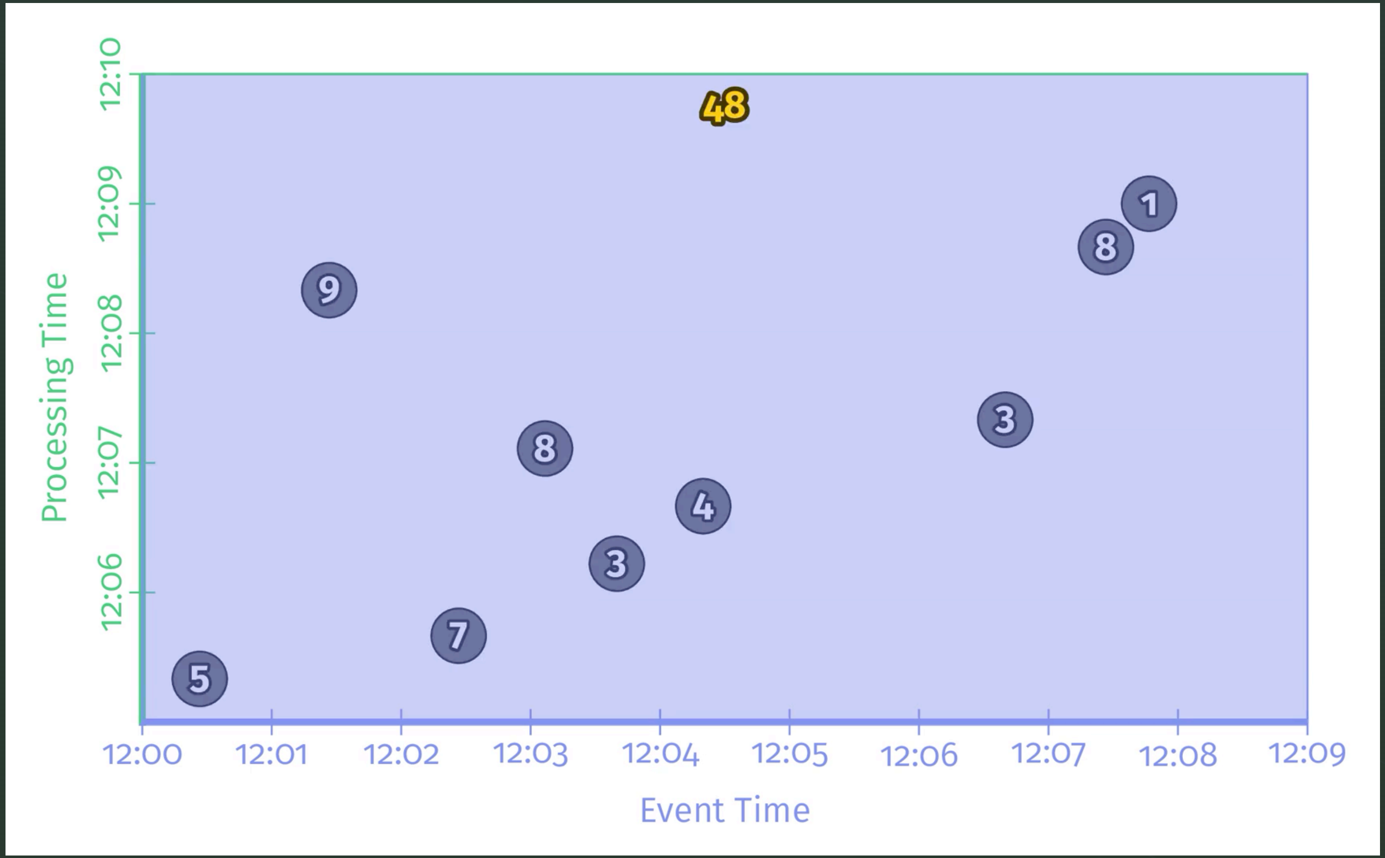
```
> SELECT * FROM UserScores ORDER BY EventTime;
--------------------------------------------------
| Name  | Team  | Score | EventTime | ProcTime |
--------------------------------------------------
| Julie | TeamX |     5 |  12:00:26 | 12:05:19 |
| Frank | TeamX |     9 |  12:01:26 | 12:08:19 |
| Ed    | TeamX |     7 |  12:02:26 | 12:05:39 |
| Julie | TeamX |     8 |  12:03:06 | 12:07:06 |
| Amy   | TeamX |     3 |  12:03:39 | 12:06:13 |
| Fred  | TeamX |     4 |  12:04:19 | 12:06:39 |
| Naomi | TeamX |     3 |  12:06:39 | 12:07:19 |
| Becky | TeamX |     8 |  12:07:26 | 12:08:39 |
| Naomi | TeamX |     1 |  12:07:46 | 12:09:00 |
--------------------------------------------------
```


Processing Time
12:10
12:09
12:08
12:07
12:06
Event Time
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
5
7
9
8
3
4
3
8
1

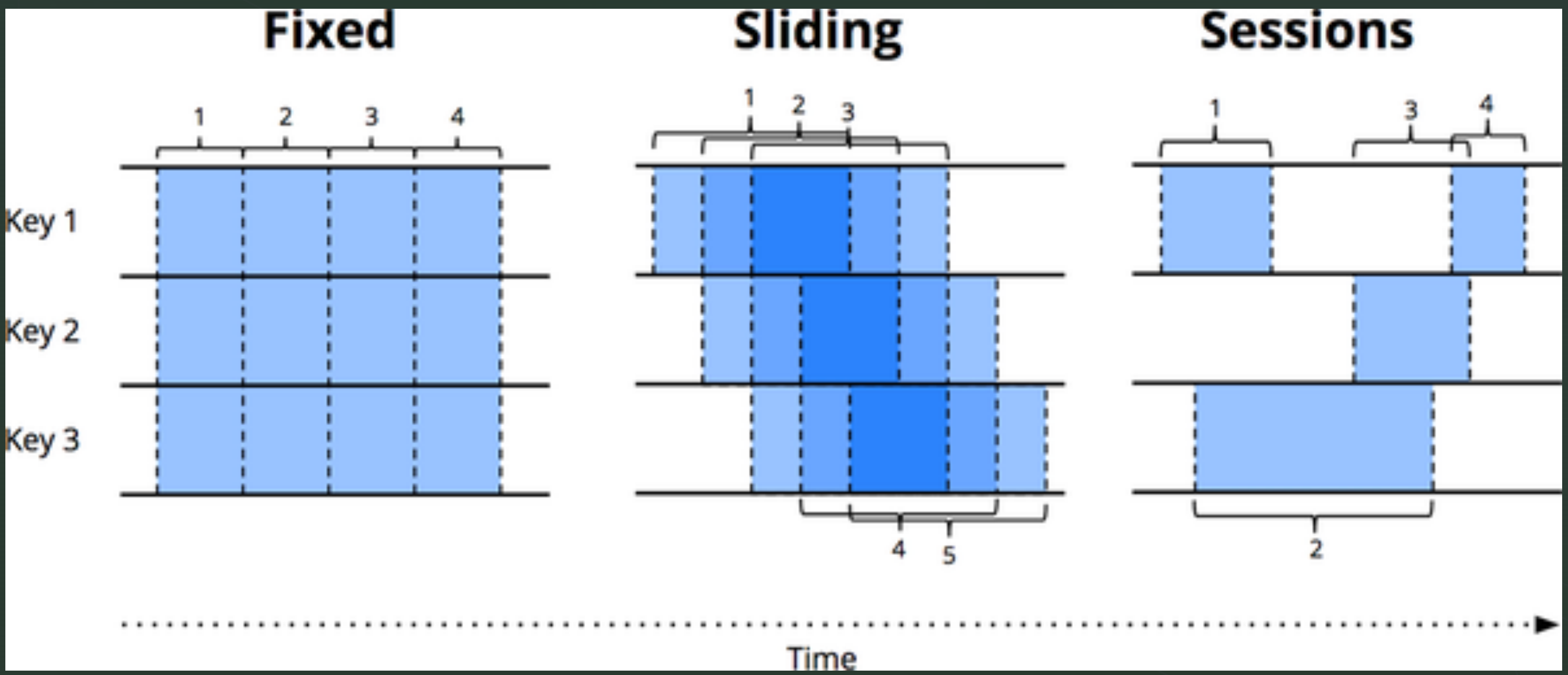
- The example pseudocode is written in Beam



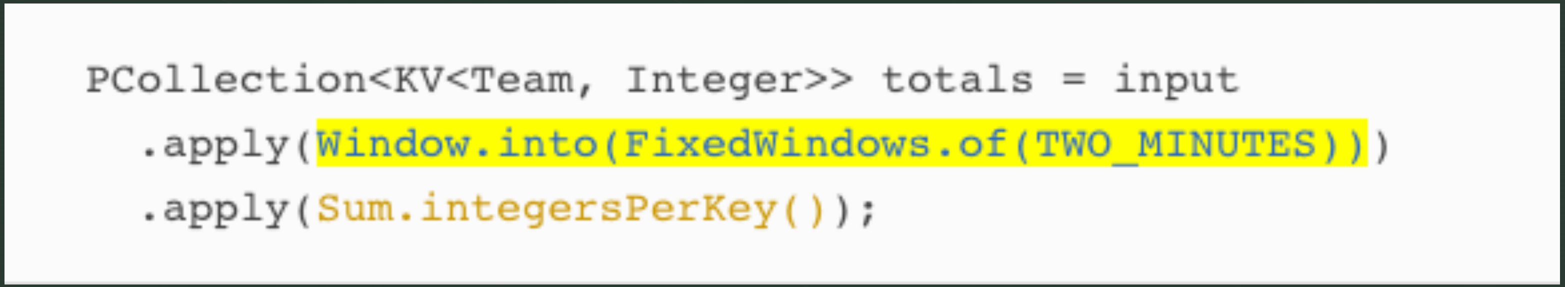
```
PCollection<String> raw = IO.read(...);
PCollection<KV<Team, Integer>> input = raw.apply(new ParseFn());
PCollection<KV<Team, Integer>> totals =
  input.apply(Sum.integersPerKey());
```


48
9
8
7
5
3
4
3
8
1
Processing Time
12:06
12:07
12:08
12:09
12:10
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
Event Time

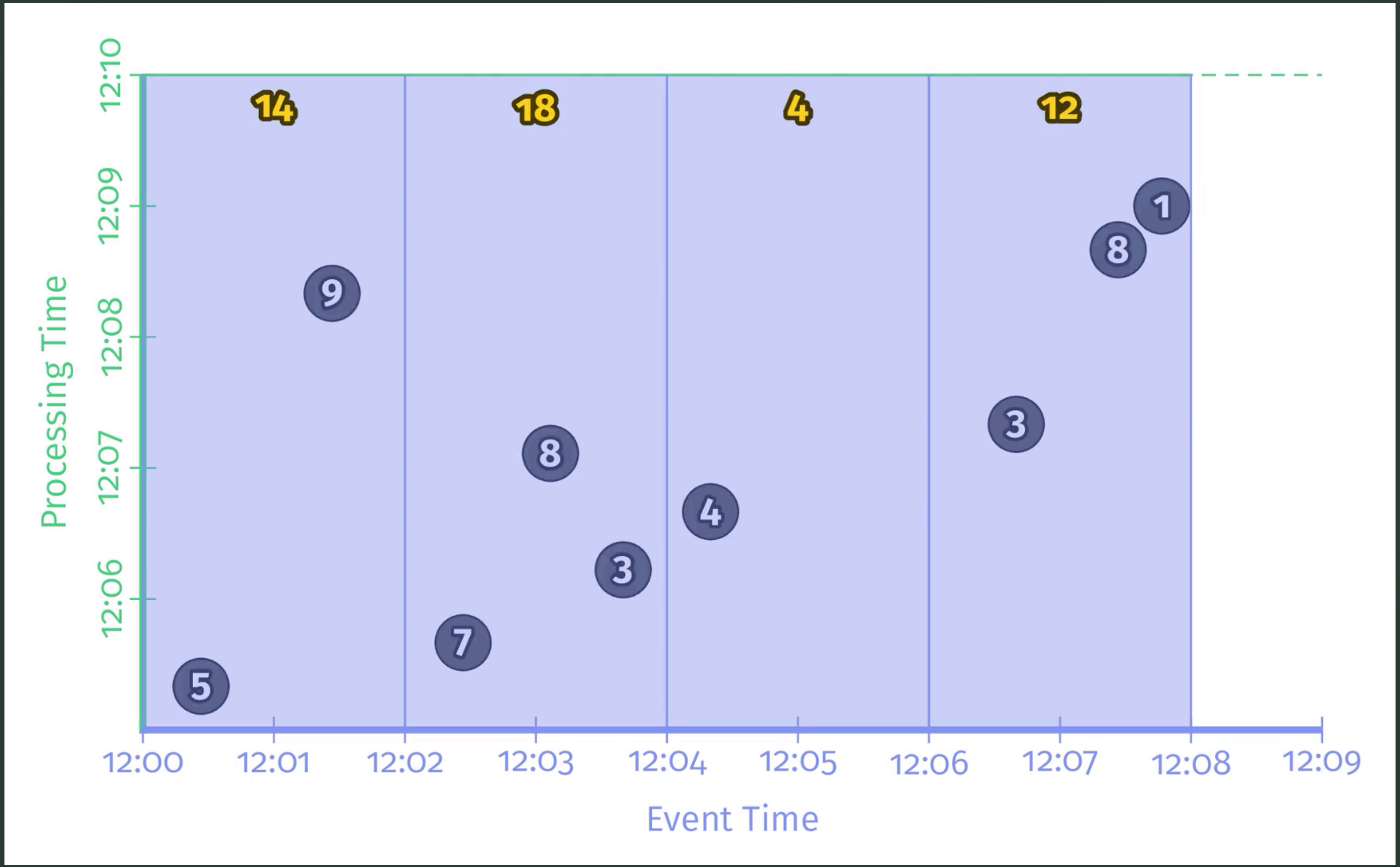
Where in event time are results calculated: Windowing



- Adding 2 minutes FixedWindow (aka TumblingWindow)



```
PCollection<KV<Team, Integer>> totals = input
  .apply(Window.into(FixedWindows.of(TWO_MINUTES)))
  .apply(Sum.integersPerKey());
```

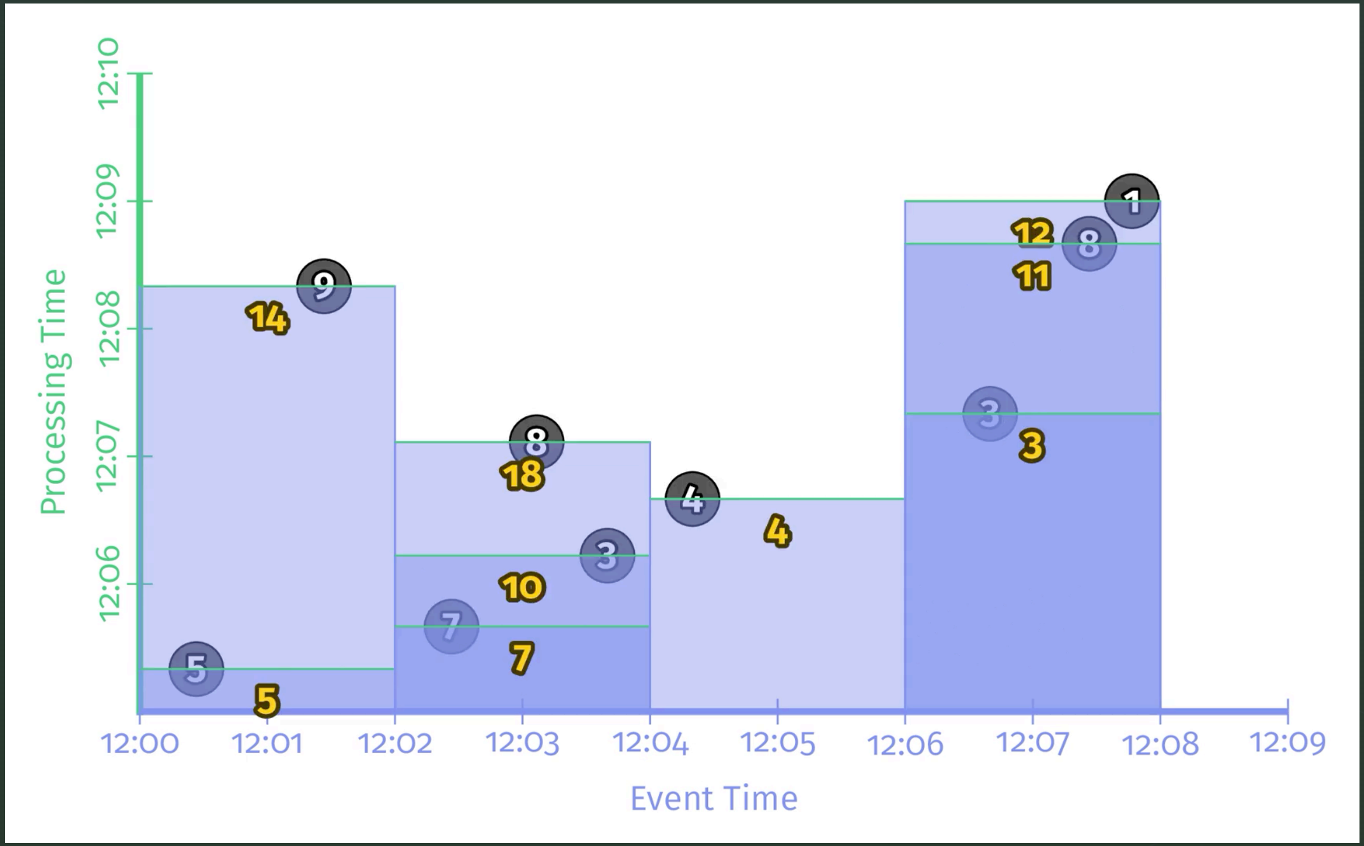


14
18
4
12
9
5
8
3
7
4
3
8
1
12:10
12:09
12:08
12:07
12:06
Processing Time
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
Event Time

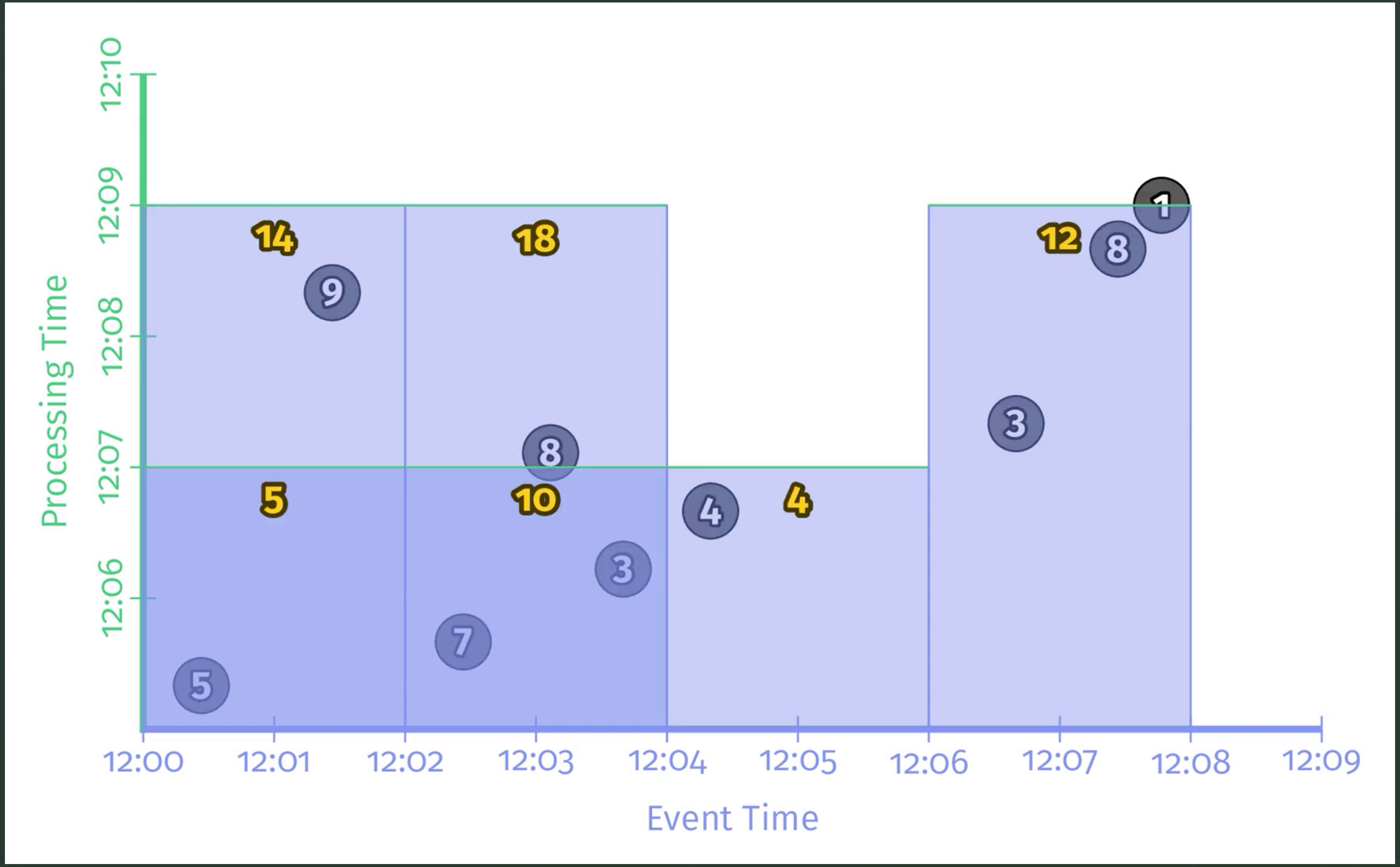
When in processing time are results materialized: Trigger

- A trigger is a mechanism for declaring when the output for a window should be materialized relative to some external signal
- Two types:
  - Repeated update
  - Completeness

```
PCollection<KV<Team, Integer>> totals = input
  .apply(Window.into(FixedWindows.of(TWO_MINUTES))
               .triggering(Repeatedly(AfterCount(1))));
  .apply(Sum.integersPerKey());
```

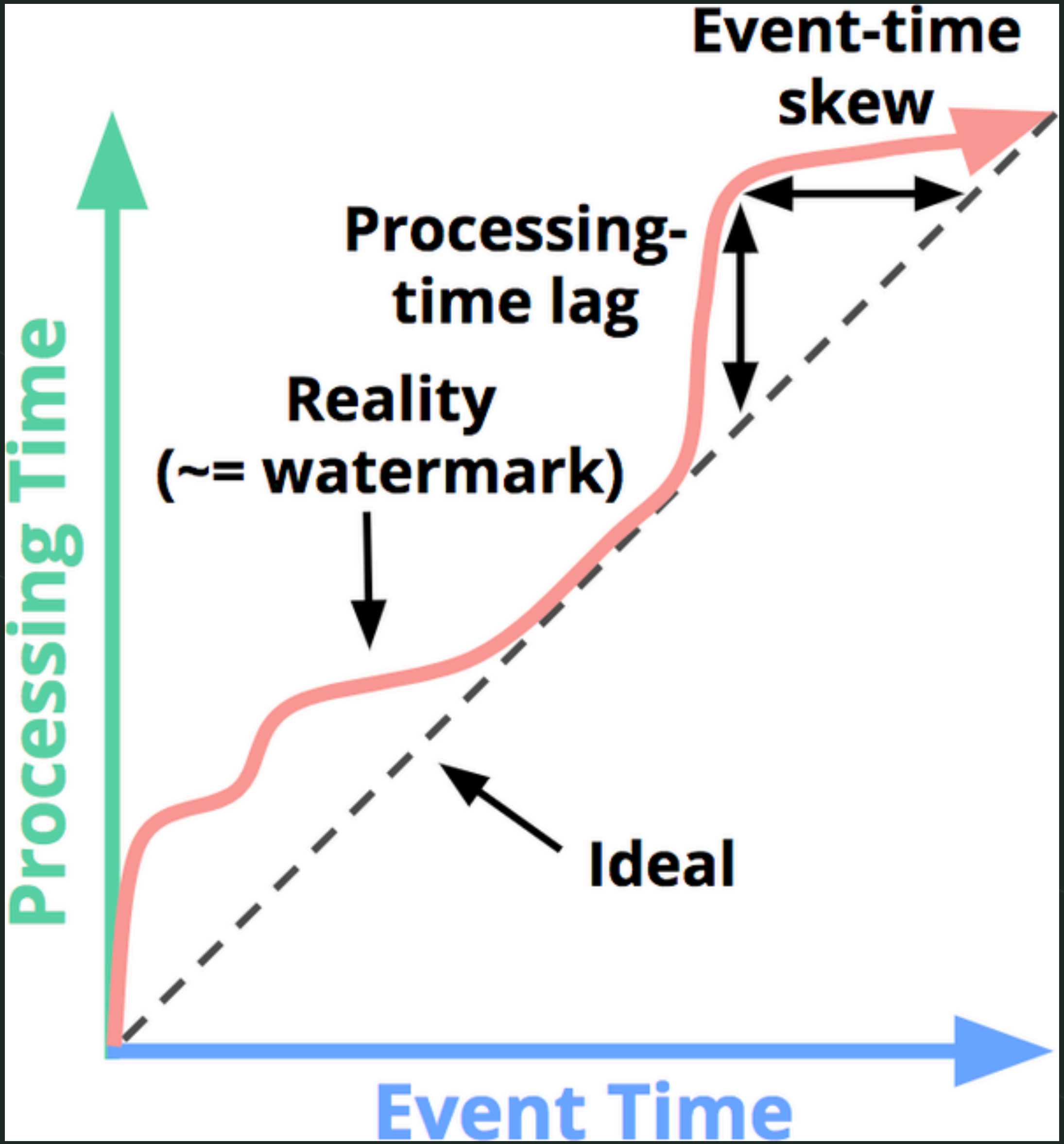


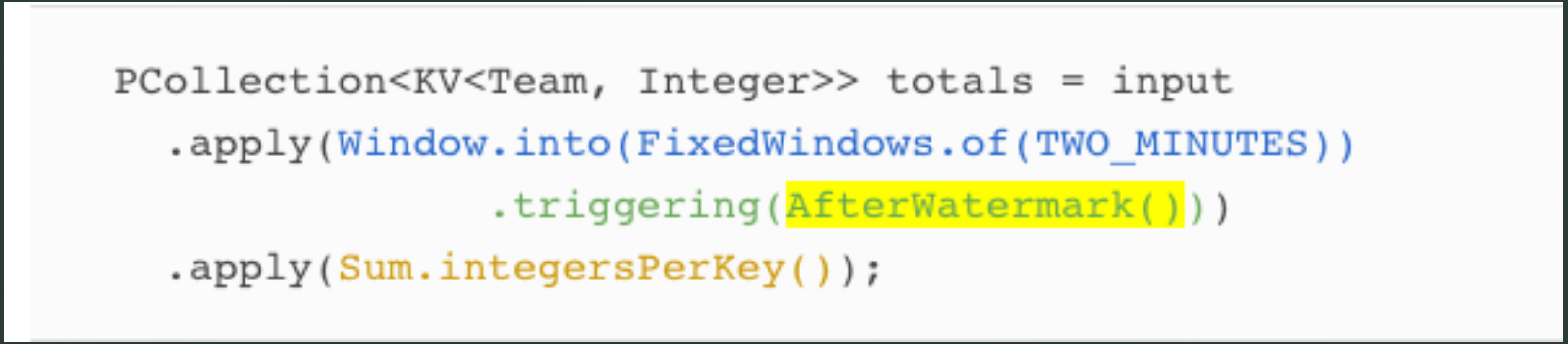
```
PCollection<KV<Team, Integer>> totals = input
  .apply(Window.into(FixedWindows.of(TWO_MINUTES))
               .triggering(Repeatedly(AlignedDelay(TWO_MINUTES)))
  .apply(Sum.integersPerKey());
```



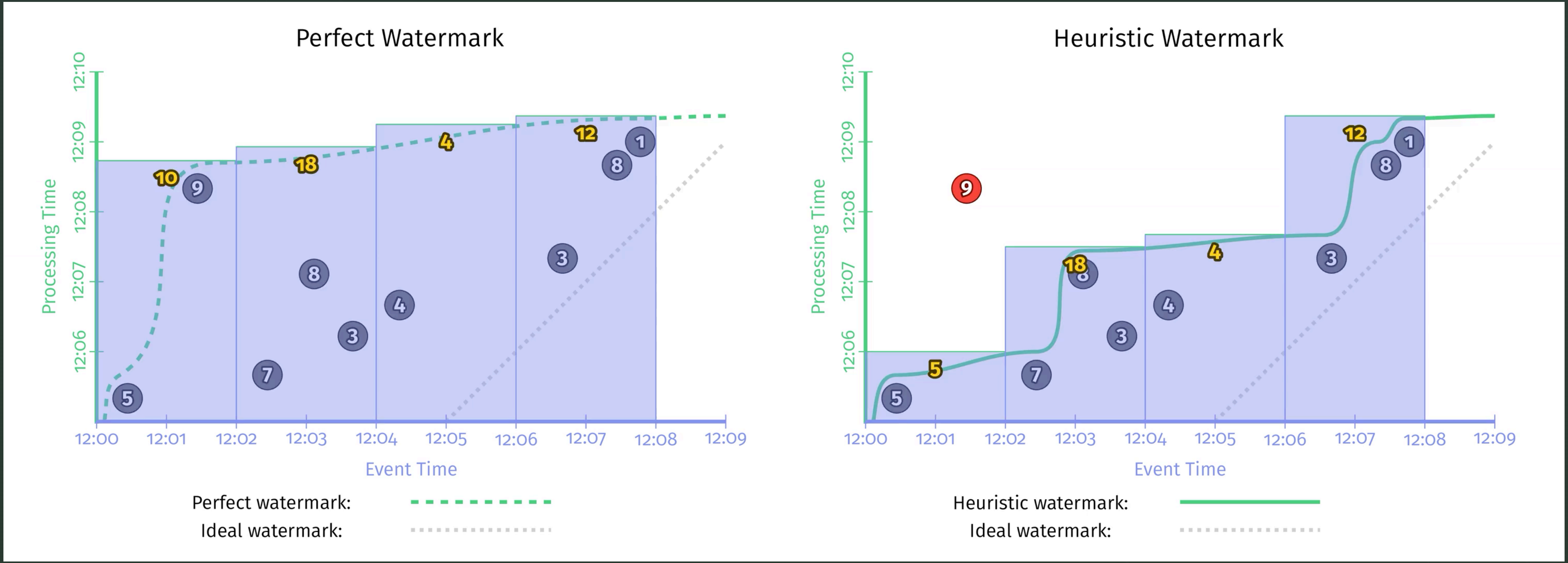
When in processing time are results materialized: Watermarks

- Temporal notions of input completeness in the event-time domain


Event-time skew
Processing-time lag
Reality (~= watermark)
Ideal
Processing Time
Event Time



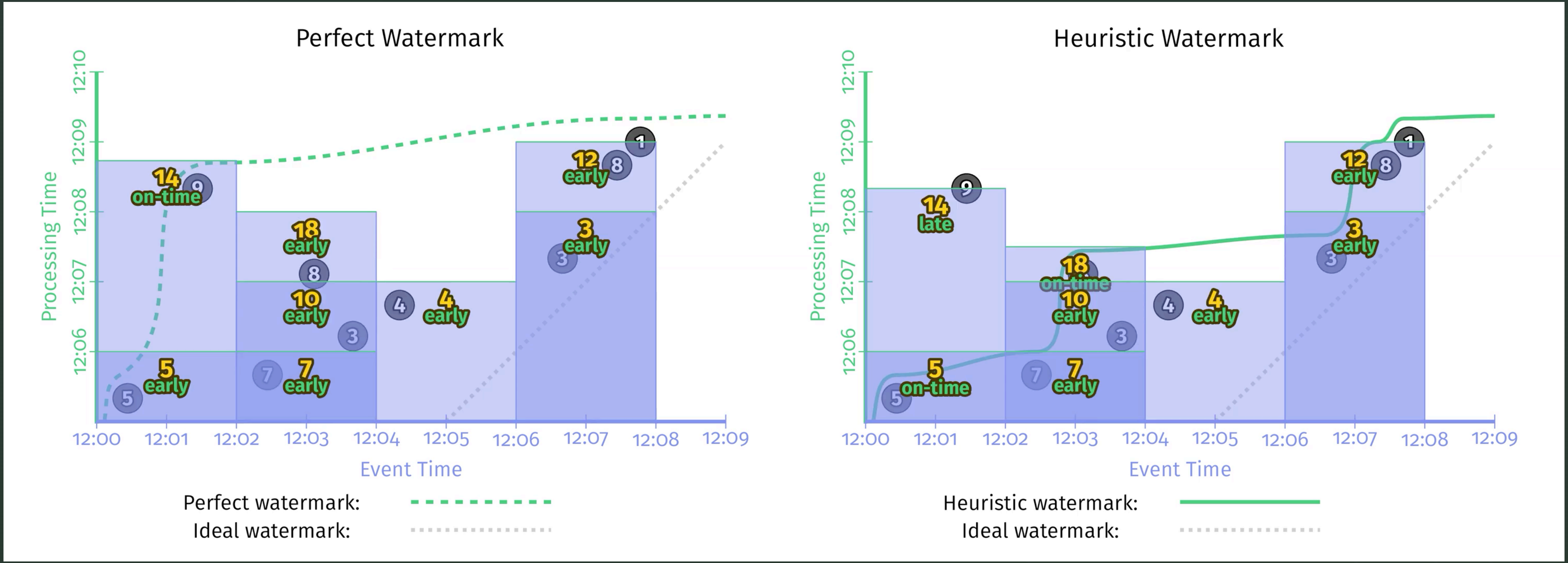
```
PCollection<KV<Team, Integer>> totals = input
   .apply(Window.into(FixedWindows.of(TWO_MINUTES))
                .triggering(AfterWatermark()))
   .apply(Sum.integersPerKey());
```

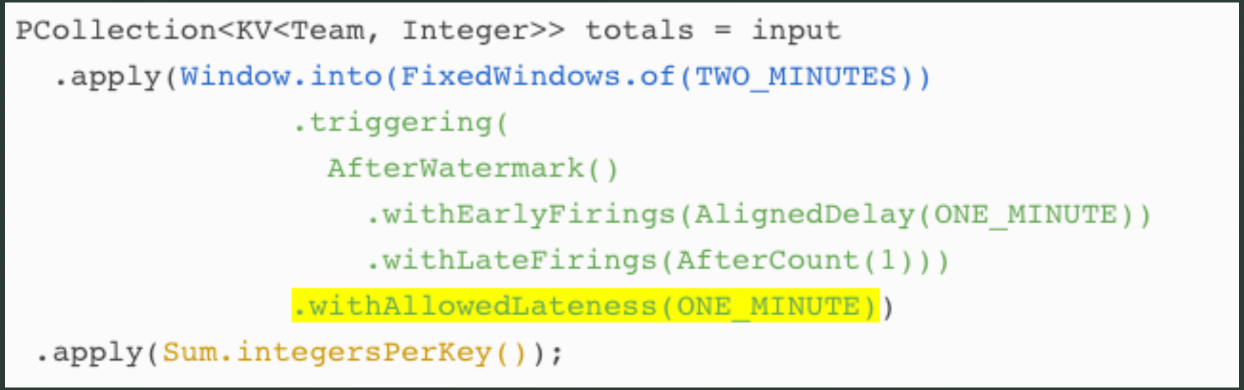


Too Slow

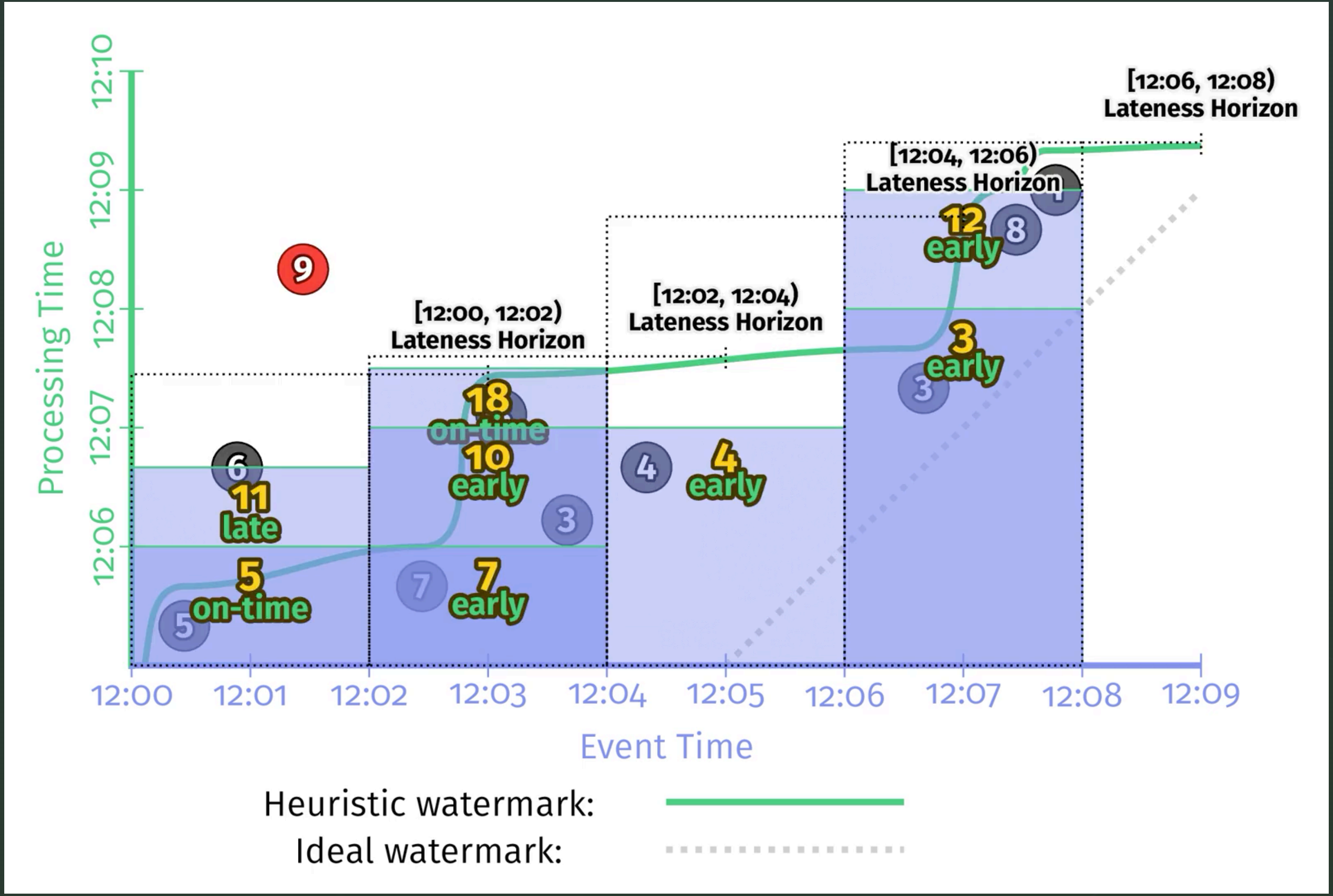
Too Fast

```
PCollection<KV<Team, Integer>> totals = input
  .apply(Window.into(FixedWindows.of(TWO_MINUTES))
               .triggering(AfterWatermark()
                            .withEarlyFirings(AlignedDelay(ONE_MINUTE))
                            .withLateFirings(AfterCount(1))))
  .apply(Sum.integersPerKey());
```





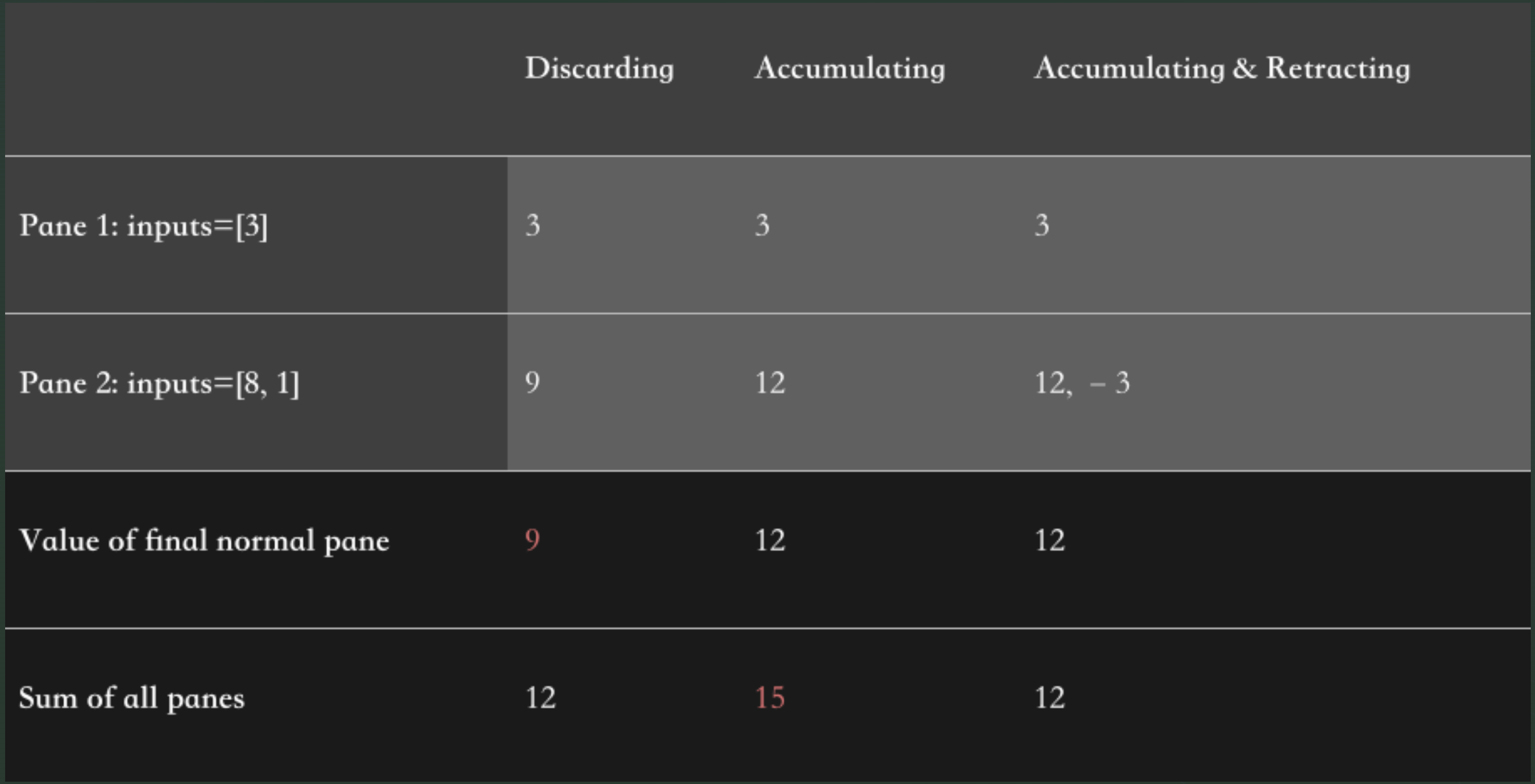
```
PCollection<KV<Team, Integer>> totals = input
  .apply(Window.into(FixedWindows.of(TWO_MINUTES))
               .triggering(
                 AfterWatermark()
                   .withEarlyFirings(AlignedDelay(ONE_MINUTE))
                   .withLateFirings(AfterCount(1)))
               .withAllowedLateness(ONE_MINUTE))
 .apply(Sum.integersPerKey());
```



How do refinements of results relate: **Accumulation**

- Discarding
- Accumulating
- Accumulating and retracting

The former example of event time 12:06 – 12:08



	Discarding	Accumulating	Accumulating & Retracting
Pane 1: inputs=[3]	3	3	3
Pane 2: inputs=[8, 1]	9	12	12, − 3
Value of final normal pane	9	12	12
Sum of all panes	12	15	12

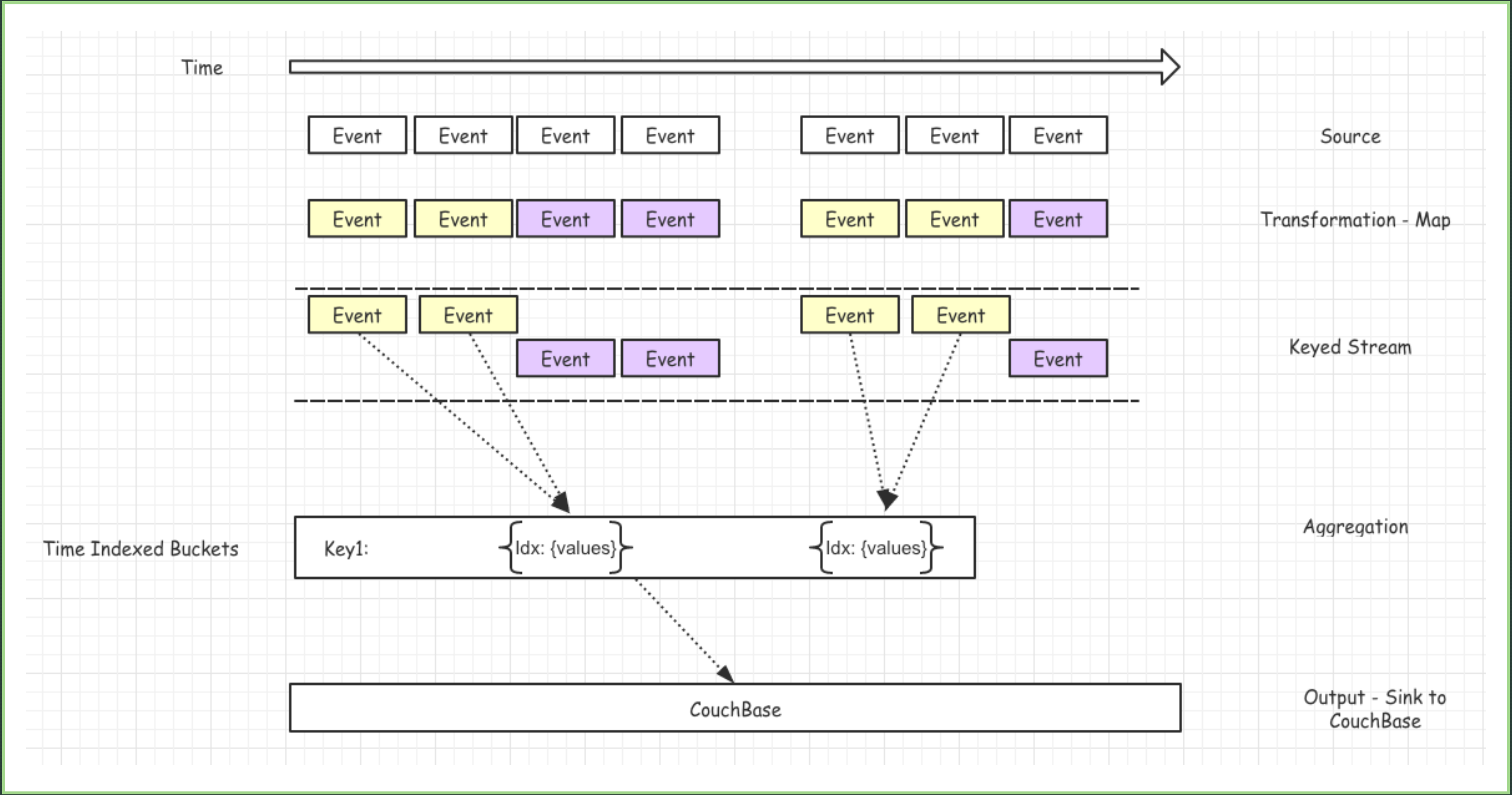
Stream and Table

- Streams → tables
  - The aggregation of a stream of updates over time yields a table.
- Tables → streams
  - The observation of changes to a table over time yields a stream.

- Tables are data *at rest*.
  - This isn’t to say tables are static in any way; nearly all useful tables are continuously changing over time in some way. But at any given time, a snapshot of the table provides some sort of picture of the dataset contained together as a whole.[2] In that way, tables act as a conceptual resting place for data to accumulate and be observed over time.
- Streams are data *in motion*.
  - Whereas tables capture a view of the dataset as a whole at a *specific point in time*, streams capture the evolution of that data *over time*. Julian Hyde is fond of saying streams are like the derivatives of tables, and tables the integrals of streams, which is a nice way of thinking about it for you math-minded individuals out there. Regardless, the important feature of streams is that they capture the inherent movement of data within a table as it changes.

Streaming Applications in eBay (Flink)

- RCDL
- ECDL
- RateLimiter


Time
Event
Event
Event
Event
Event
Event
Event
Source
Event
Event
Event
Event
Event
Event
Event
Transformation - Map
Event
Event
Event
Event
Event
Event
Event
Keyed Stream
Time Indexed Buckets
Key1:
Idx: {values}
Idx: {values}
Aggregation
CouchBase
Output - Sink to CouchBase

Thanks!