NIKOS FERTAKIS

PAPERS WE LOVE - ATHENS

THE DATAFLOW MODEL

THE DATAFLOW MODEL

- A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing
- Akidau et al. (Google) - 2015

TERMINOLOGY: UNBOUNDED/BOUNDED VS STREAMING/BATCH

- Data sets are **bounded/unbounded**.
- Execution engines are **streaming/batch**.
- **Streaming** systems have been designed for **unbounded** datasets.
- Unbounded datasets have been processed using repeated runs of batch systems since their conception.

STREAMING VS. BATCH

- Well-designed streaming systems
  - Are perfectly capable of processing bounded data
  - Provide a strict superset of batch functionality
- To beat batch systems you need
  - *correctness*
  - *tools for reasoning about time.*

INTRODUCTION

- Unbounded, unordered, global-scale datasets are increasingly common in day-to-day business.
- Web logs, mobile usage statistics, sensor networks, etc.
- Consumers of these datasets have evolved sophisticated requirements.
- Practicality dictates that one can never fully optimise along all dimensions of correctness, latency, and cost for these types of input.

MOTIVATION

- A streaming video provider displaying video ads.
- Billing advertisers for the amount of advertising watched.
- Video provider wants to know:
  - How much to bill each advertiser each day
  - Aggregate statistics about videos and ads

MOTIVATION (CONT)

- Advertisers/content providers want to know:
  - How often and for how long their videos are being watched
  - With which content/ads
  - By which demographic groups
  - How much they are being charged/paid
  - They want this information as quickly as possible

MOTIVATION (CONT)

- The video provider wants a programming model that is simple and flexible.
- Can handle global scale data.
- Money is involved: correctness is paramount.
- Information needs to be quick so budgets can be adjusted.
- Information that must be calculated: time and length of each video viewing, who viewed it, and with which ad or content it was paired (i.e. per-user, per-video viewing *sessions*).
- Existing models and systems all fall short of meeting the stated requirements.

MOTIVATION (CONT)

- Batch systems suffer from latency problems inherent with collecting all input data into a batch before processing it.
- Many streaming systems aren't fault-tolerant at scale.
- Others fail to provide exactly-once semantics, impacting correctness.
- Others lack temporal primitives necessary for windowing or provide limited windowing semantics (tuple- or processing-time-based windows).
- MillWheel and Spark Streaming lack high-level programming models that make calculating event-time sessions straightforward.
- Finally, Lambda Architecture systems fail on the simplicity axis - two systems to build and maintain.

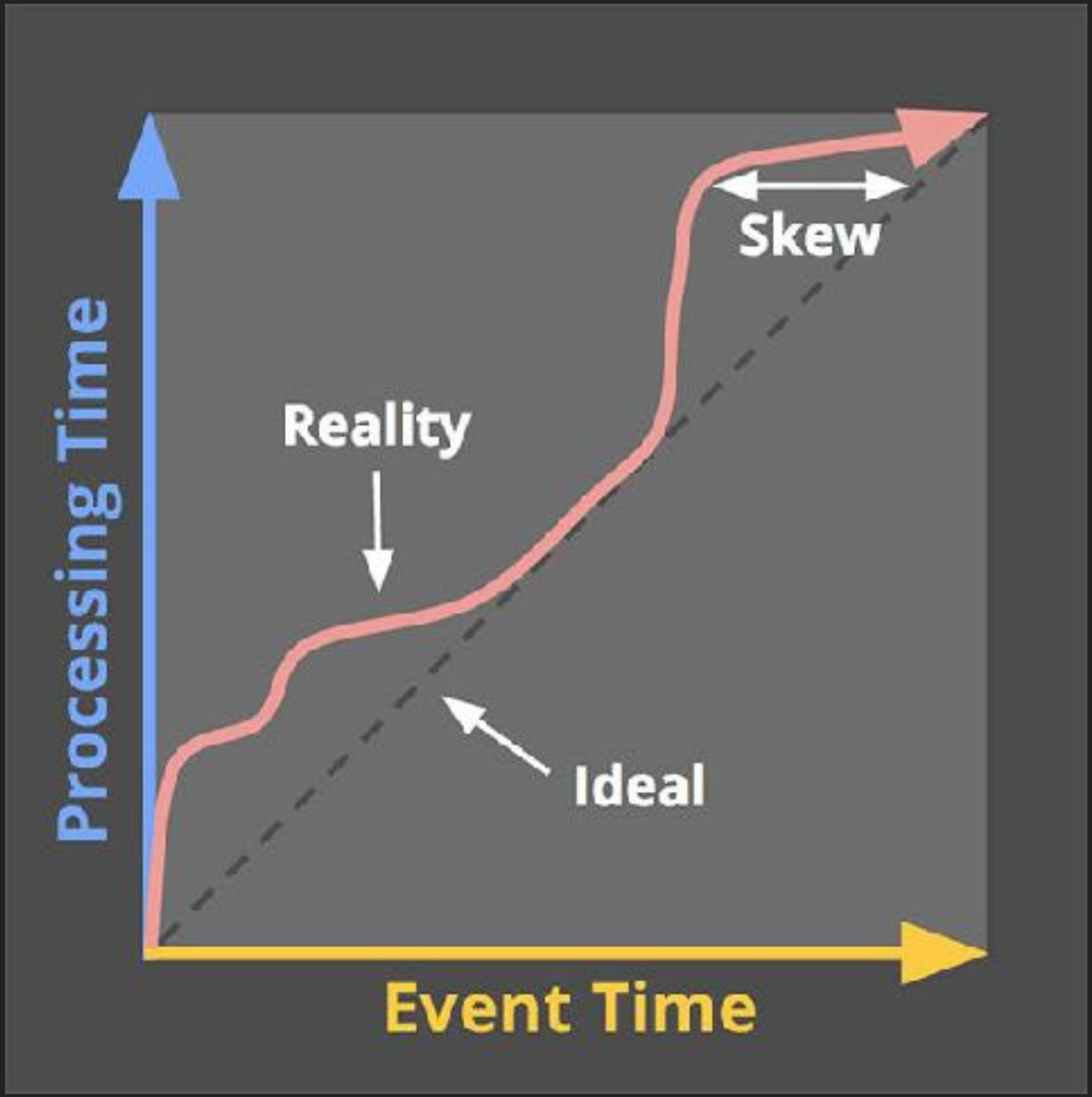
MANIFESTO

- A fundamental shift of approach is necessary.
- We must stop trying to groom unbounded datasets into finite pools of information that eventually become complete.
- We must live and breathe under the assumption that we will never know if or when we have seen all of our data.
- New data will arrive, old data may be retracted.
- The only way to make this problem tractable is via principled abstractions that allow the practitioner the choice of appropriate tradeoffs along the axes of interest.
- These are: **correctness**, **latency**, and **cost**.

TIME DOMAINS

- Two inherent domains of time to consider:
  - **Event Time**: the time at which the event itself actually *occurred*.
  - **Processing Time**: the time at which an event is observed at any given point *during processing*.
- Event time for a given event never changes.
- Processing time changes constantly.
- During processing, the realities of systems in use result in an inherent and dynamically changing amount of skew between the two domains.

TIME DOMAINS VISUALISED



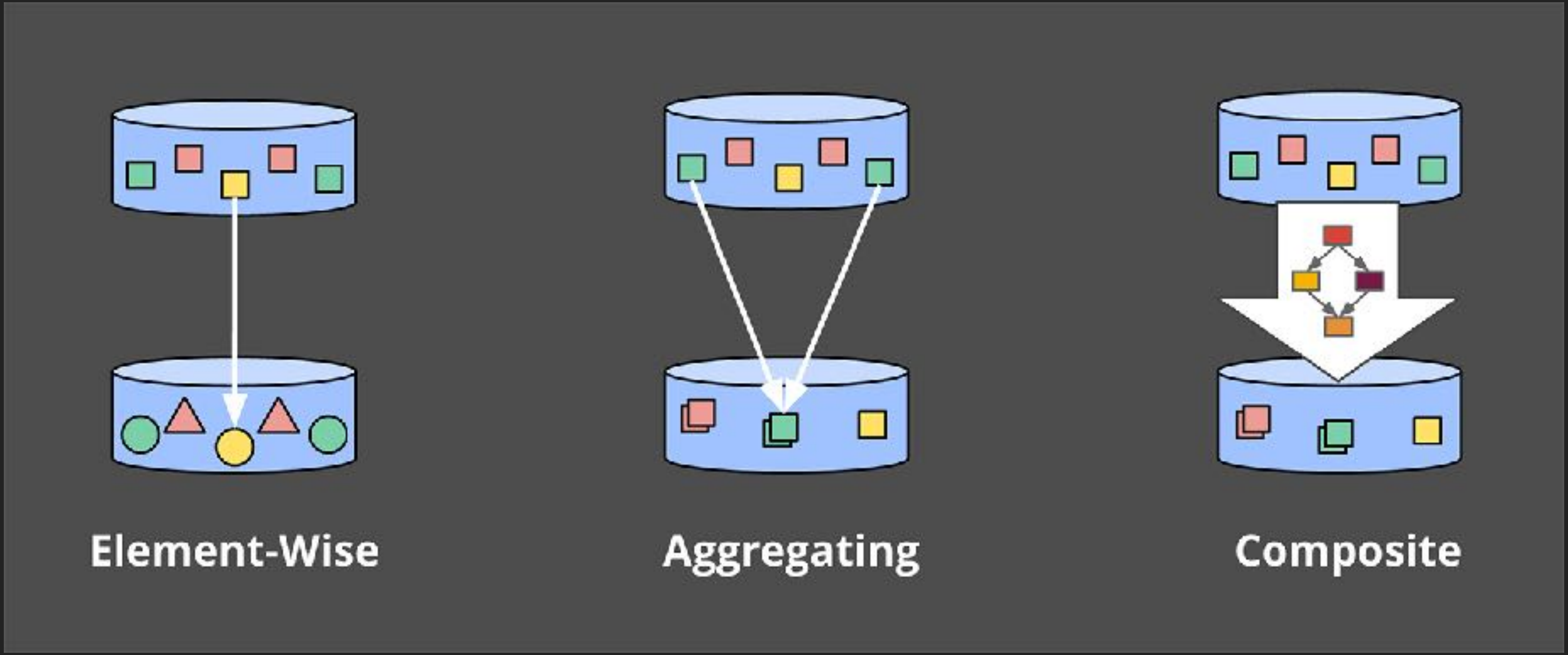
FOUR QUESTIONS

- The Dataflow model decomposes pipeline implementation across four related dimensions:
  - **What** results are being computed? Answered by the types of transformations within the pipeline.
  - **Where** in event time they are being computed? Answered by the use of event-time windowing.
  - **When** in processing time they are materialised? Answered by the use of watermarks and triggers.
  - **How** earlier results relate to later refinements. Answered by the type of accumulation used (discarding/accumulating/accumulating and retracting).

WHAT: TRANSFORMATIONS

- The transformations applied in classic batch processing answer the question *what results are calculated?*
- Two basic primitives in Dataflow:
  - `PCollections`: represent data sets across which parallel transformations may be performed.
  - `PTransforms`: applied to `PCollections`, to create new `PCollections`. May perform element-wise transformations, aggregate multiple elements, or be a composite combination of other `PTransforms`.

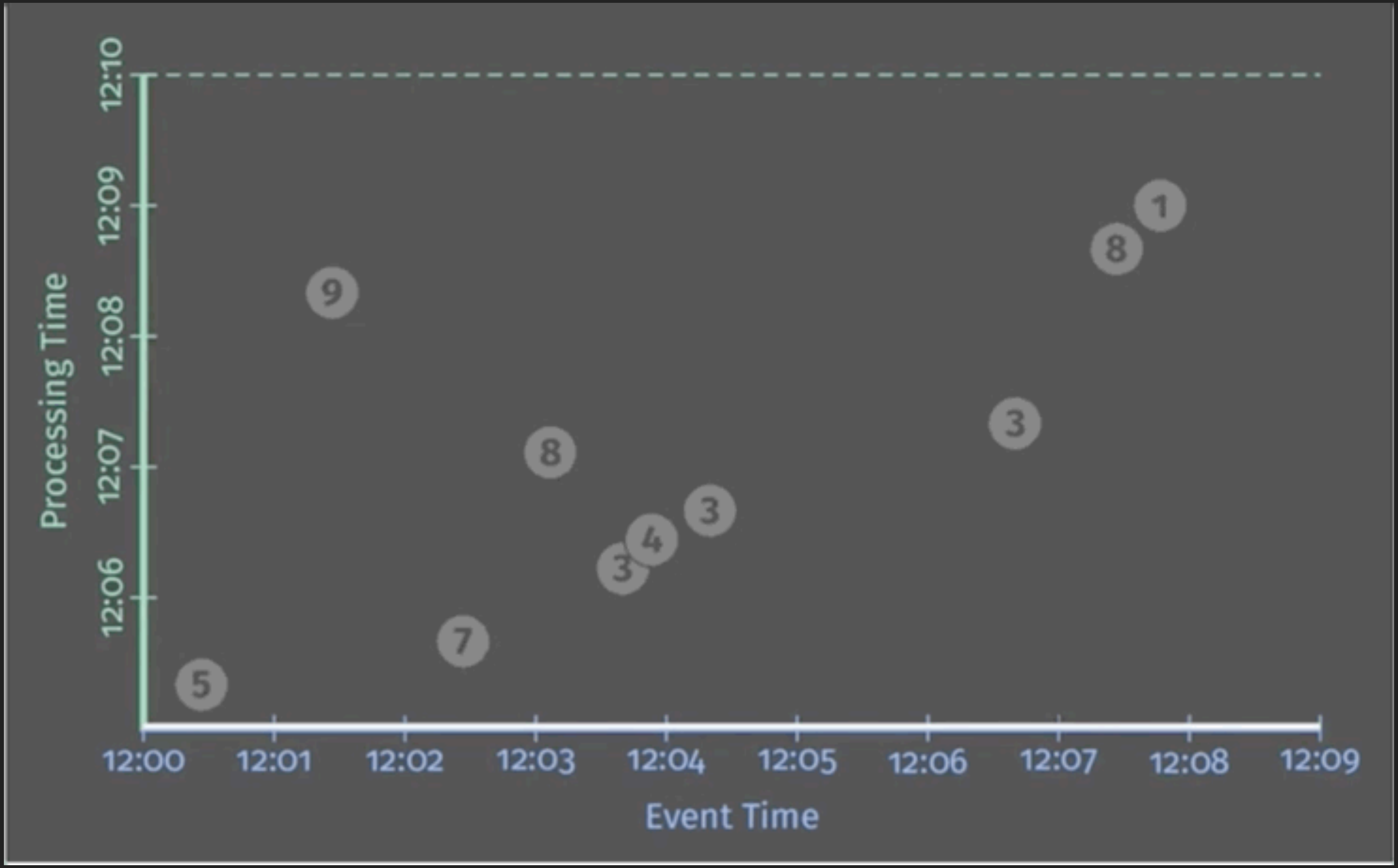
TRANSFORMATIONS VISUALISED



TRANSFORMATIONS (CONT)

- Two core transforms that operate on the *(key, value)* pairs flowing through the system:
  - `ParDo` for generic parallel processing. Each input element to be processed is provided to a user-defined function, which can yield zero or more output elements per input.
  - `GroupByKey` for key-grouping.

CLASSIC BATCH PROCESSING



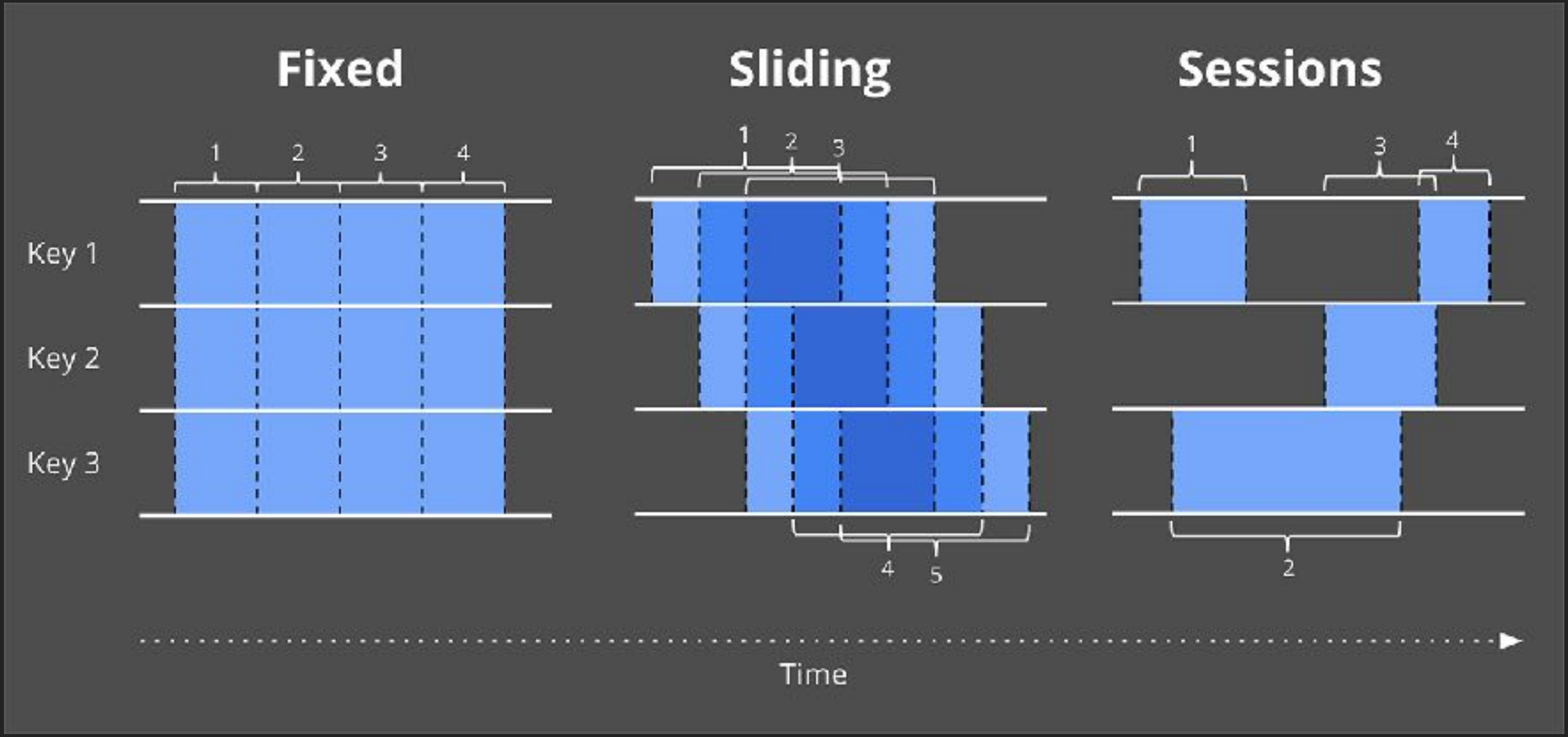
WHERE: WINDOWING

- If we want to process an unbounded data source, classic batch processing won't be sufficient; we can't wait for the input to end.
- **Windowing**: chopping up the data set into finite pieces along temporal boundaries.
- If you care about correctness, you cannot define these boundaries using processing time.
- Lacking a predictable mapping between processing time and event time *you can't determine when you've observed all the data for a given event time X*.
- **Aligned**: apply to all keys.
- **Unaligned**: apply to some keys.

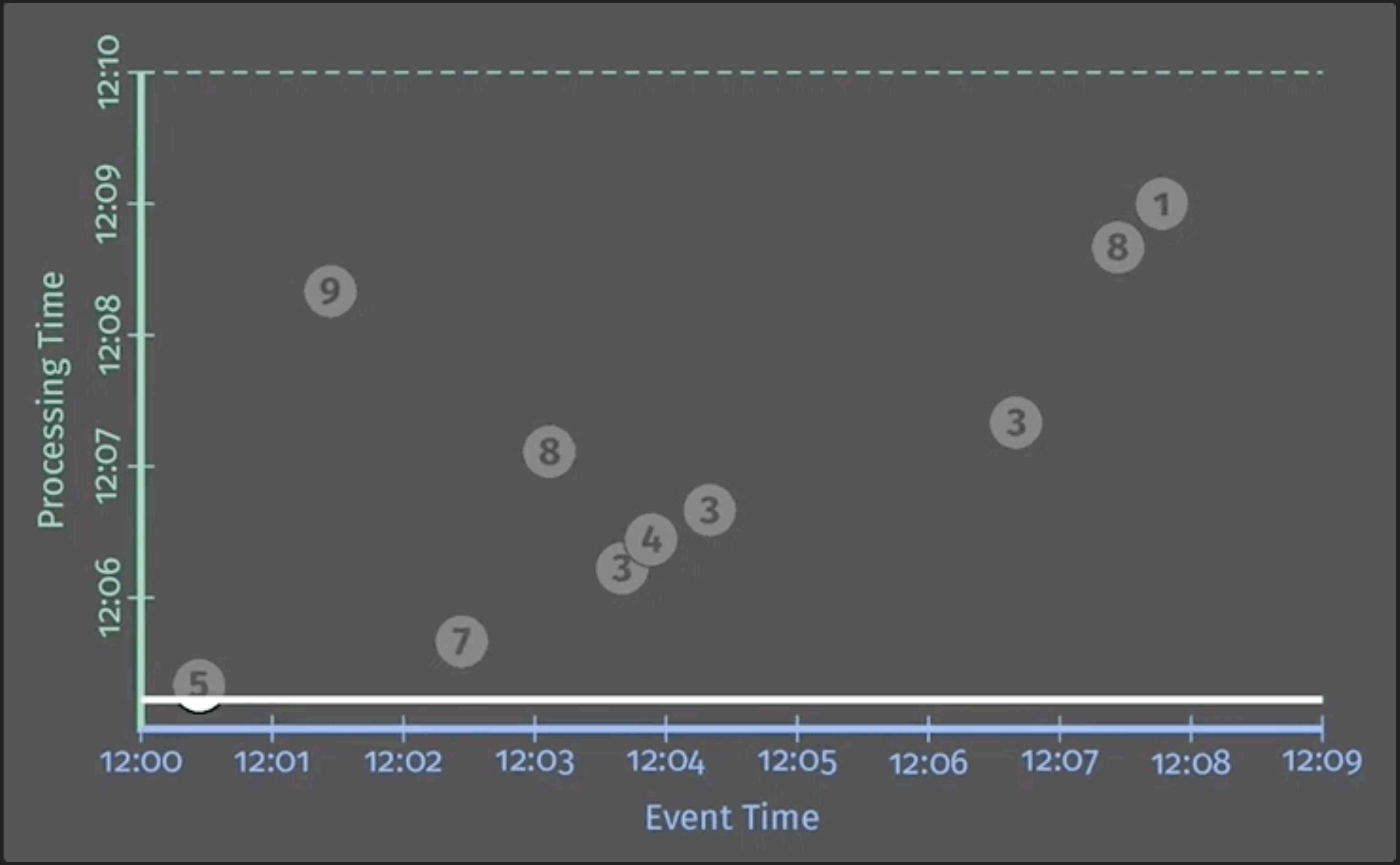
MAJOR TYPES OF WINDOWS

- **Fixed windows**: defined by a static window size, e.g. hourly or daily. Typically aligned.
- **Sliding windows**: defined by a window size and a slide period, e.g. hourly windows starting every minute. Fixed windows are sliding windows where size equals period. Typically aligned.
- **Sessions**: windows that capture some period of activity over a subset of the data, e.g. per key. Events that occur within a span of time less that the timeout are grouped together as a session. Unaligned.

WINDOW TYPES VISUALISED



WINDOWED SUMMATION ON A BATCH ENGINE



WHEN: WATERMARKS

- Watermarks are the first half of the answer to the question *"When in processing time are results materialised?"*
- They are the way the system measures *progress* and *completeness* relative to the event times of the records being processed in a stream of events.
- `F(P) -> E`, takes a point in processing time and returns a point in event time.
- That point in event time is the point up to which the system believes all inputs with event times less than E have been observed.

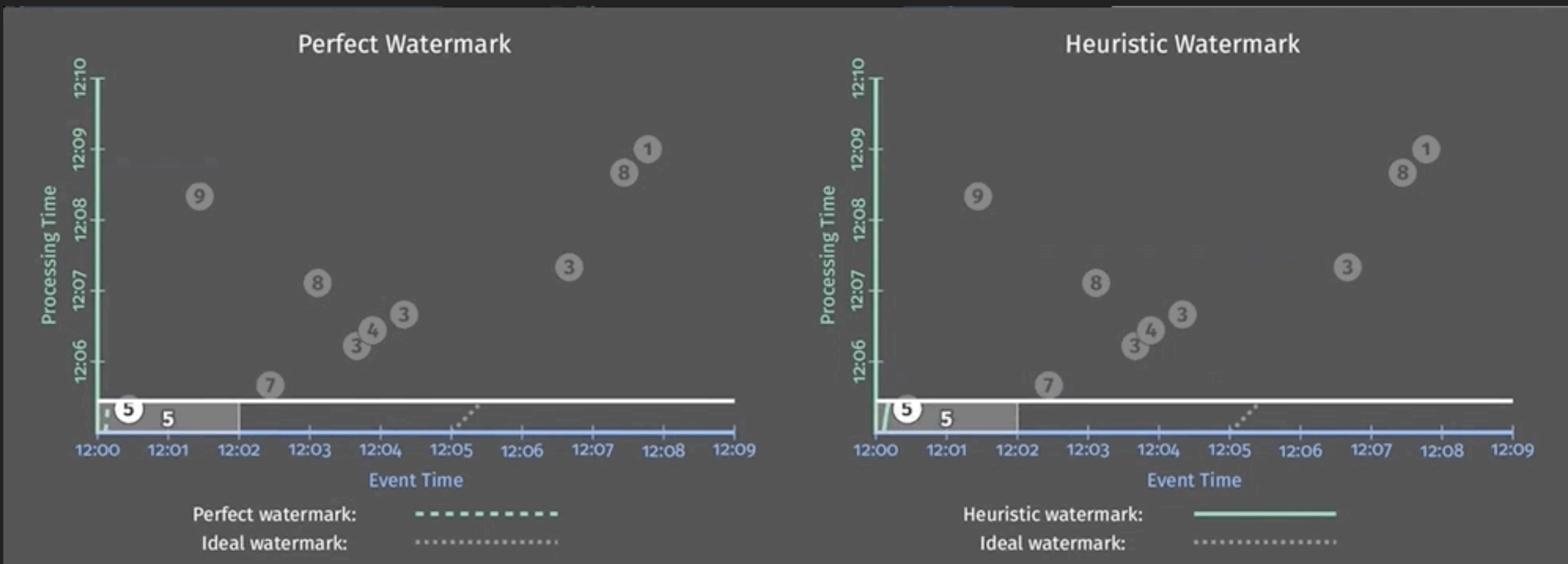
WATERMARK TYPES

- **Perfect watermarks**: if we have *perfect knowledge* of all the input data, it is possible to construct a perfect watermark. No chance of late data.
- **Heuristic watermarks**: use whatever information is available about the inputs (e.g. partitions, ordering within partitions, growth rates of files, etc.) to provide an *estimate of progress*. Can be remarkably accurate in their predictions.

WATERMARK SHORTCOMINGS

- They are sometimes **too fast**: there may be late data that arrives behind the watermark. For many distributed data sources, it is intractable to derive a completely perfect time watermark. Impossible to rely on it solely if we want 100% correctness.
- They are sometimes **too slow**: the watermark can be held back for the entire pipeline by a single slow datum. Likely to yield higher latency of overall results than, e.g., a comparable Lambda Architecture pipeline.

WINDOWED SUMMATION ON A STREAMING ENGINE



WHEN: TRIGGERS

- Triggers are the second half of the answer to the question *"When in processing time are results materialised?"*
- Triggering determines *when* in processing time should the output for a window happen.
- *Pane* of the window: each specific output for a window.

EXAMPLE SIGNALS USED FOR TRIGGERING

- **Watermark progress (i.e. event time progress)**: outputs are materialised when the watermark passes the end of a window.
- **Processing time progress**: useful for providing regular, periodic updates.
- **Element counts**: useful for triggering after some finite amount of elements have been observed in a window.
- **Punctuations**: some record or feature of record indicates output should be emitted.
- Triggers can also be composed into logical combinations (and, or, etc.), loops, sequences, etc.
- Users may define their own triggers.

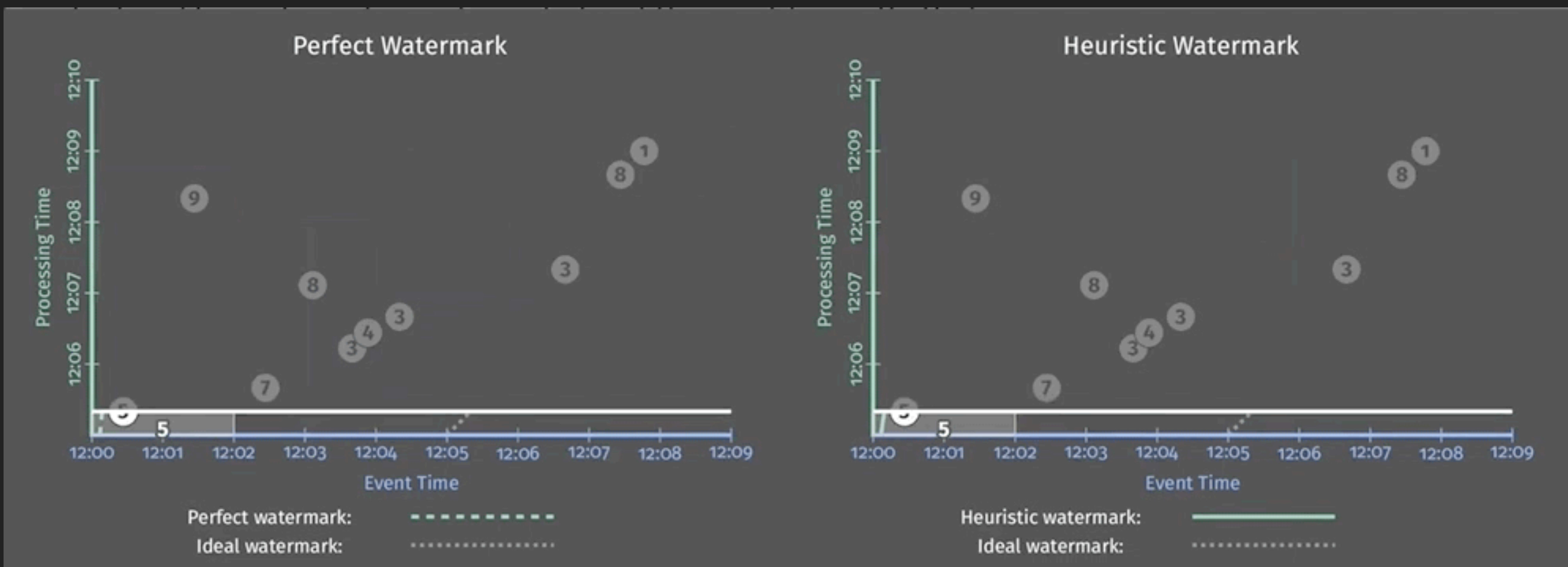
TRIGGERS (CONT)

- With triggers we can tackle problems where watermarks are too slow or too fast. For example:
  - **Too slow**: trigger periodically while processing the window.
  - **Too fast**: trigger after observing element count of 1 - will catch any remaining data after the end of the window.

TRIGGERS VISUALISED

- Window: fixed duration of 2 minutes.
- Triggers set:
  - Early: emit every one minute
  - On-time: emit when watermark passes the end of a window
  - Late: emit every time you see late data
- You can define how late individual data may be (or else you will eventually run out of disk!)

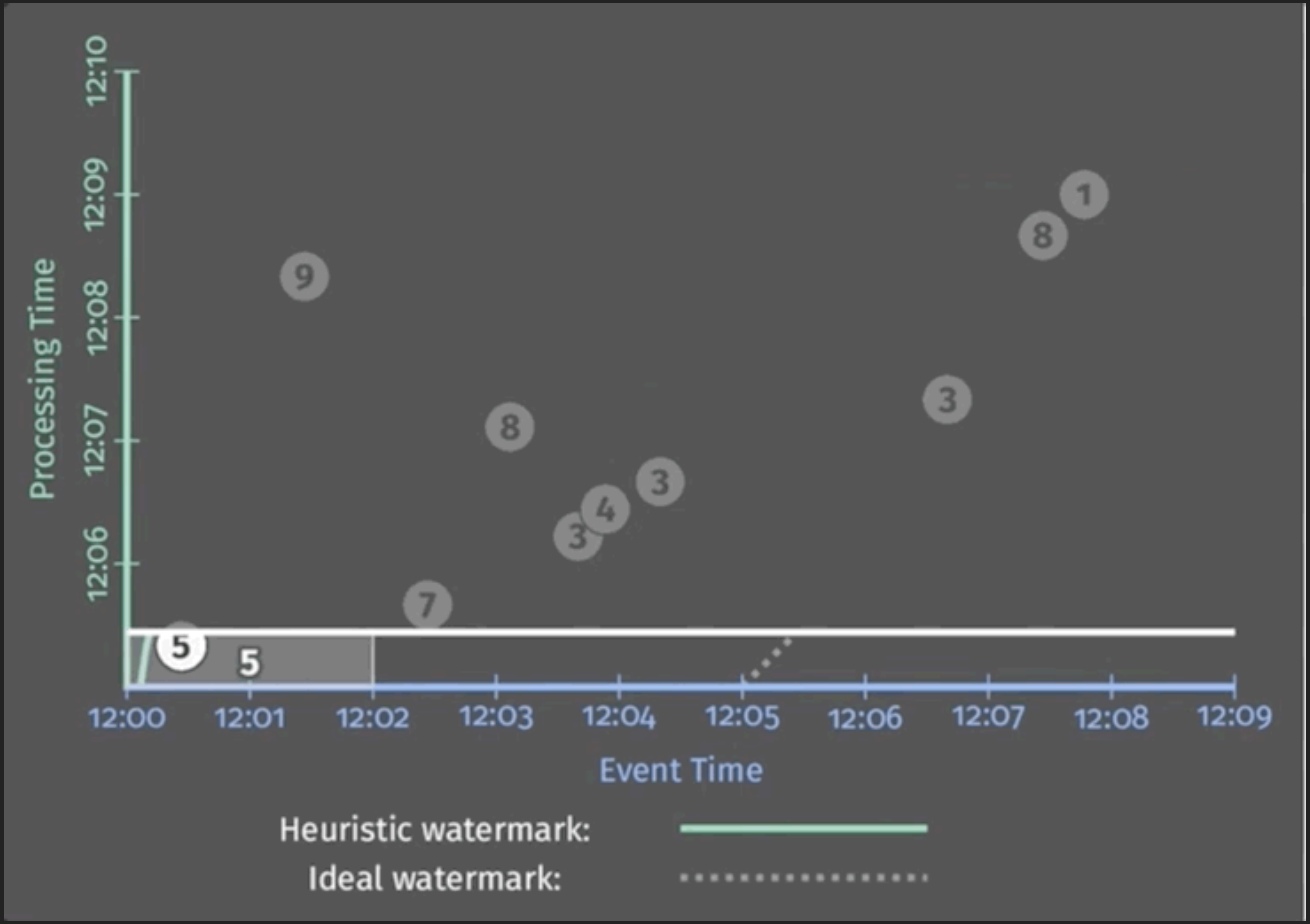
TRIGGERS VISUALISED



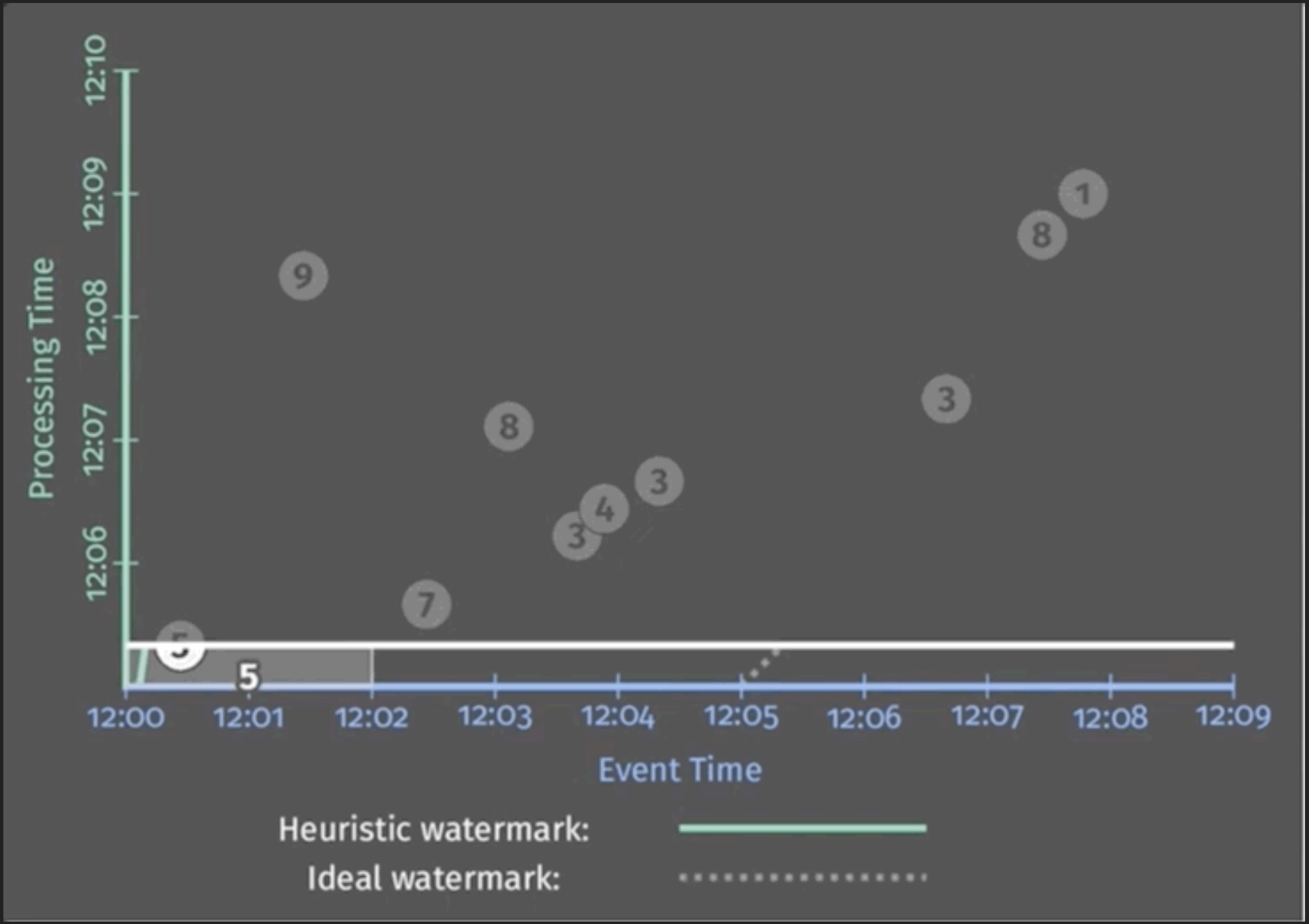
HOW: ACCUMULATION

- How do multiple panes for the same window relate two each other?
- Three different refinement modes:
  - **Discarding**: Upon triggering, window contents are discarded, and later results bear no relation to previous results. The client must keep track and combine old and new results.
  - **Accumulating**: Upon triggering, window contents are left intact in persistent state, and later results become a refinement of previous results. The client can just keep the latest results.
  - **Accumulating & Retracting**: A retraction for the previous value will be emitted first, followed by the new, accumulated value. Useful for complex pipelines where windows are merged or data get regrouped in different dimensions.

DISCARDING VISUALISED



ACCUMULATING & RETRACTING VISUALISED



EXAMPLE: SESSIONS

- From a windowing perspective, sessions are interesting in two ways:
  - They are an example of a **data-driven** window: the location and sizes of the windows are a direct consequence of the input data themselves, rather a predefined time pattern.
  - They are an example of an **unaligned** window: a window that does not apply uniformly across the data.

SUPPORT FOR UNALIGNED WINDOWS

- Windowing can be broken apart into two related operations:
  - `AssignWindows`: assigns the element to zero or more windows.
  - `MergeWindows`: merges windows at grouping time. Allows data-driven windows to be constructed over time as data arrive and are grouped together.
- To support event-time windowing natively, we pass *(key, value, event_time, window)* 4-tuples.
- Elements are initially assigned to a default global window, covering all of event time [0, ∞).

WINDOW ASSIGNMENT FOR SESSIONS

- The sessions implementation of `AssignWindows` puts each element into a single window that extends 30 minutes beyond its own timestamp.
- This window denotes the range of time into which later events can fall if they are to be considered part of the same session.
- We then begin the `GroupByKeyAndWindow` operation.

ASSIGN WINDOWS

- Input:
  (k1, v1, 13:02, [0, ∞)),
  (k2, v2, 13:14, [0, ∞)),
  (k1, v3, 13:57, [0, ∞)),
  (k1, v4, 13:20, [0, ∞))

- `AssignWindows(Sessions(30m))`
  (k1, v1, 13:02, [13:02, 13:32)),
  (k2, v2, 13:14, [13:14, 13:44)),
  (k1, v3, 13:57, [13:57, 14:27)),
  (k1, v4, 13:20, [13:20, 13:50))

GROUP BY KEY AND WINDOW OPERATION

- Five-part composite operation:
  - `DropTimestamps`: drops element timestamps, as only the window is relevant from now on.
  - `GroupByKey`: groups *(value, window)* tuples by key.
  - `MergeWindows`: merges the set of currently buffered windows by key. The merge logic is defined by the windowing strategy.
  - `GroupAlsoByWindow`: for each key, groups values by window.
  - `ExpandToElements`: expands per-key, per-window groups of values into *(key, value, event_time, window)* tuples, with new per-window timestamps.

GROUP BY KEY AND WINDOW OPERATION (CONT)

- `DropTimestamps`

  (k1, v1, [13:02, 13:32)),
  (k2, v2, [13:14, 13:44)),
  (k1, v3, [13:57, 14:27)),
  (k1, v4, [13:20, 13:50))

- `GroupByKey`

  (k1, [(v1, [13:02, 13:32)),
  (v3, [13:57, 14:27)),
  (v4, [13:20, 13:50))]),
  (k2, [(v2, [13:14, 13:44))])

GROUP BY KEY AND WINDOW OPERATION (CONT)

- `MergeWindows(Sessions(30m))`

  (k1, [(v1, [**13:02**, **13:50**)),
  (v3, [13:57, 14:27)),
  (v4, [**13:02**, **13:50**))]),
  (k2, [(v2, [13:14, 14:44))])

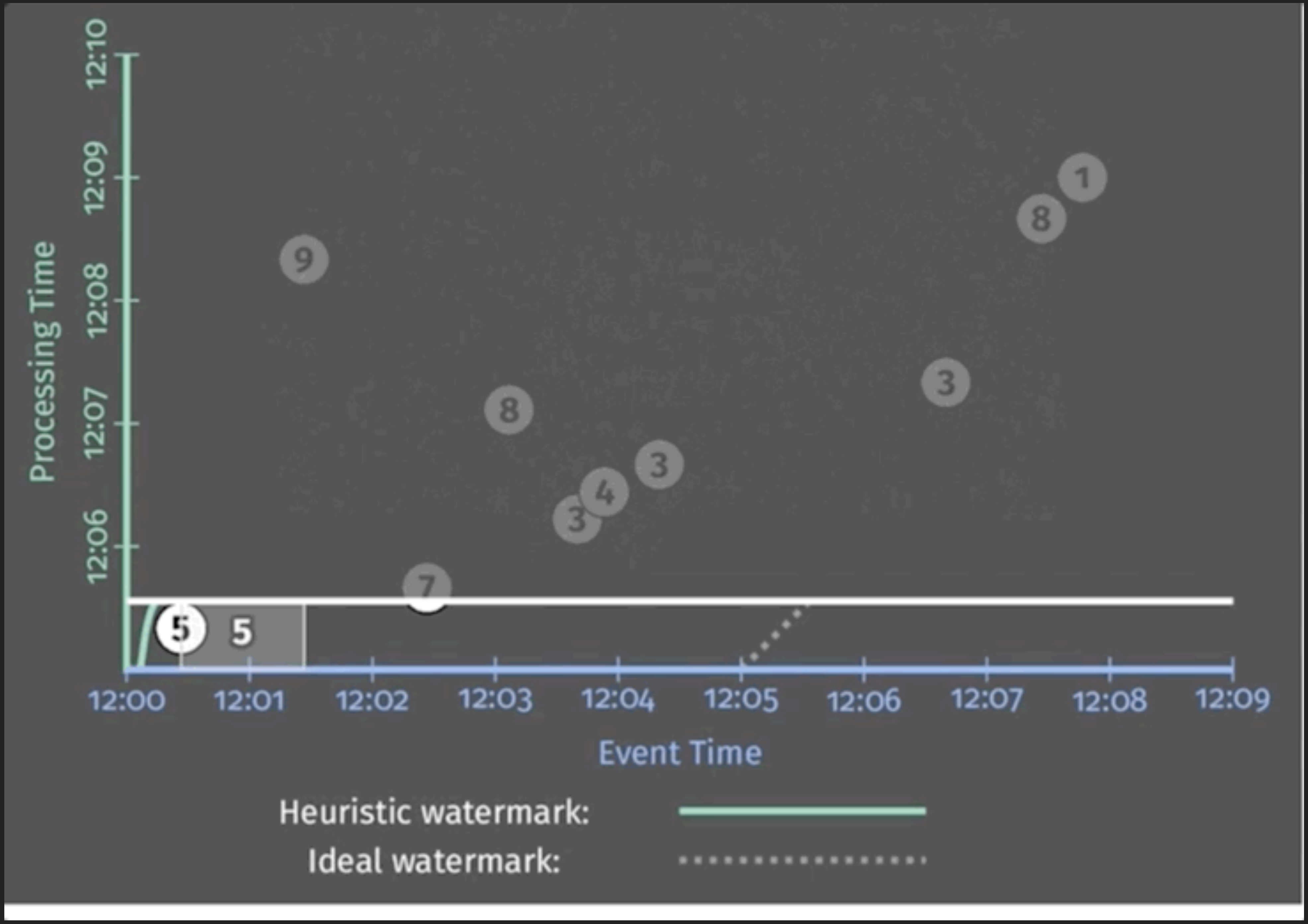
- `GroupAlsoByWindow`

  (k1, [([**v1**, **v4**], [13:02, 13:50)),
  ([**v3**], [13:57, 14:27))]),
  (k2, [([**v2**], [13:14, 14:44))])

GROUP BY KEY AND WINDOW OPERATION (CONT)

- `ExpandToElements`
  (k1, [v1, v4], **13:50**, [13:02, 13:50)),
  (k1, [v3], **14:27**, [13:57, 14:27)),
  (k2, [v2], **14:44**, [13:14, 14:44))
- Any event timestamp greater than or equal to the timestamp of the earliest event in the window is valid with respect to watermark correctness.

SESSION WINDOWS VISUALISED



MOTIVATING EXPERIENCES

- Large Scale Backfills & The Lambda Architecture: Unified Model.
- Unaligned Windows: Sessions
- Billing: Triggers, Accumulation, & Retraction
- Statistics Calculation: Watermark Triggers
- Recommendations: Processing Time Triggers
- Anomaly Detection: Data-Driven & Composite Triggers

DATAFLOW SOFTWARE

- Apache Beam: Dataflow SDK (https://beam.apache.org)
- Works with various runtimes:
  - Apache Apex
  - Apache Flink
  - Apache Spark
  - Google Cloud Dataflow
  - Apache Gearpump
- Different levels of compatibility, see https://beam.apache.org/documentation/runners/capability-matrix/

RESOURCES

- Tyler Akidau - Streaming 101: https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
- Tyler Akidau - Streaming 102: https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102
- Martin Kleppmann - Designing Data-Driven Applications

THANK YOU ALL!

- Thanks for your attention!
- Questions?
- Nikos Fertakis
  - twitter: @nikosfertakis
  - github: @greenonion
  - email: nikos.fertakis@gmail.com