Eötvös Loránd University (ELTE)
Faculty of Informatics (IK)
Pázmány Péter sétány 1/c
1117 Budapest, Hungary

WATERMARKS AND WINDOWING (W2)

Stream mining (SM)

Imre Lendák, PhD, Associate Professor
Péter Kiss, PhD candidate

2020
Budapest, Hungary

Outline

- Advanced watermarking
- Advanced windowing

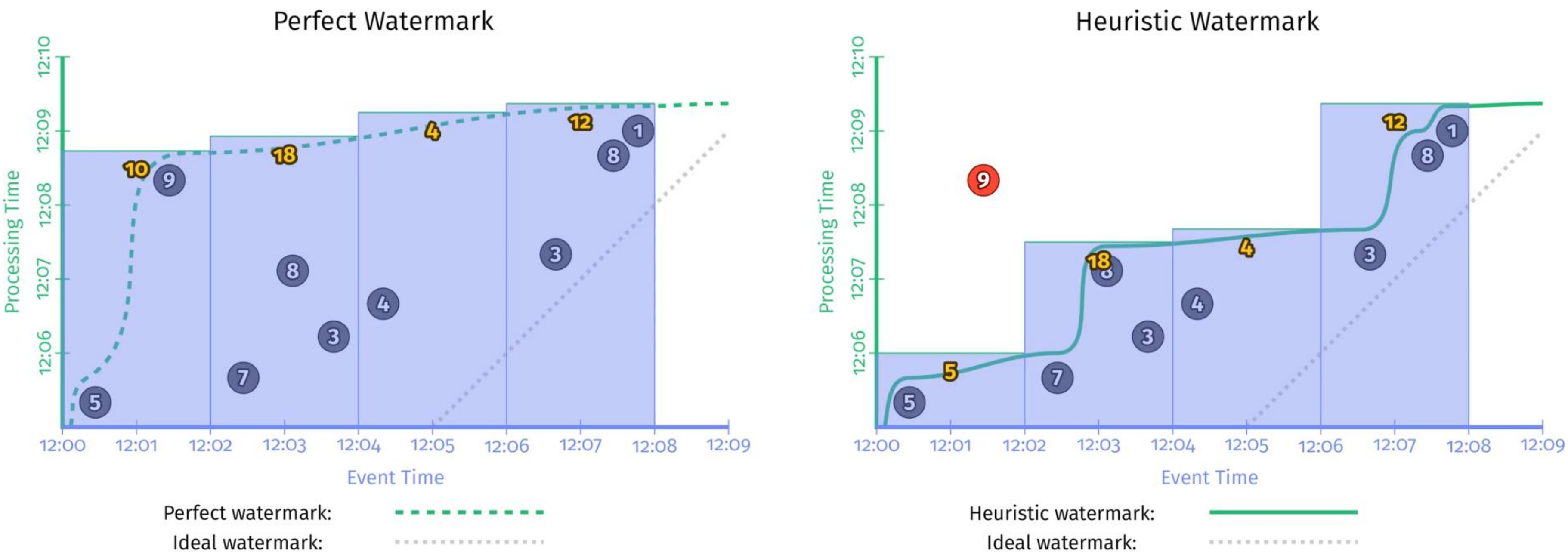
Stream mining

ADVANCED WATERMARKING

Watermarks

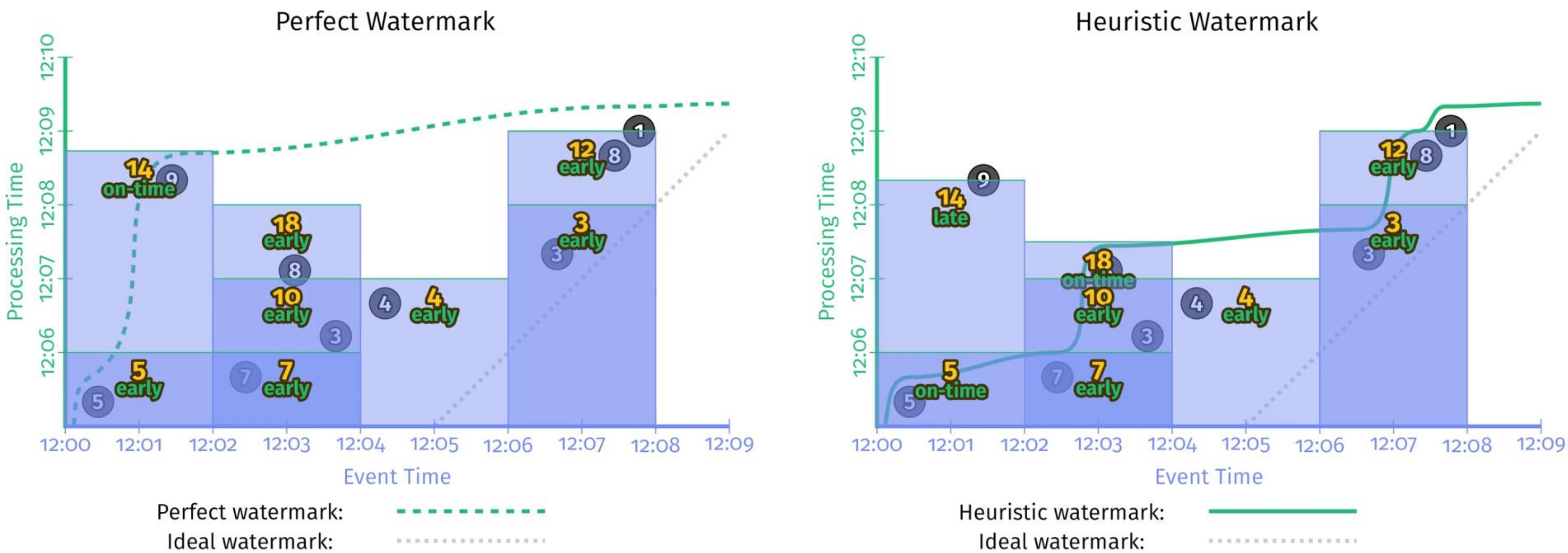
- **DEF #1:** Watermarks are temporal notions of input completeness in the event time domain
  - 'Guesses' about data completeness, i.e. all relevant data received
- **DEF #2:** Watermarks allow streaming systems to measure progress and completeness relative to event times of the records being processed
- Watermark varieties:
  - Perfect watermark = usable when we have perfect knowledge about the input data → all data on time (i.e. no late data
  - Heuristic watermark = provide an estimate of (data) progress as good as it gets → usually based on domain knowledge
- **NOTE:** Additional watermark management challenges in data processing pipelines with multiple processing steps

Perfect vs heuristic watermarks



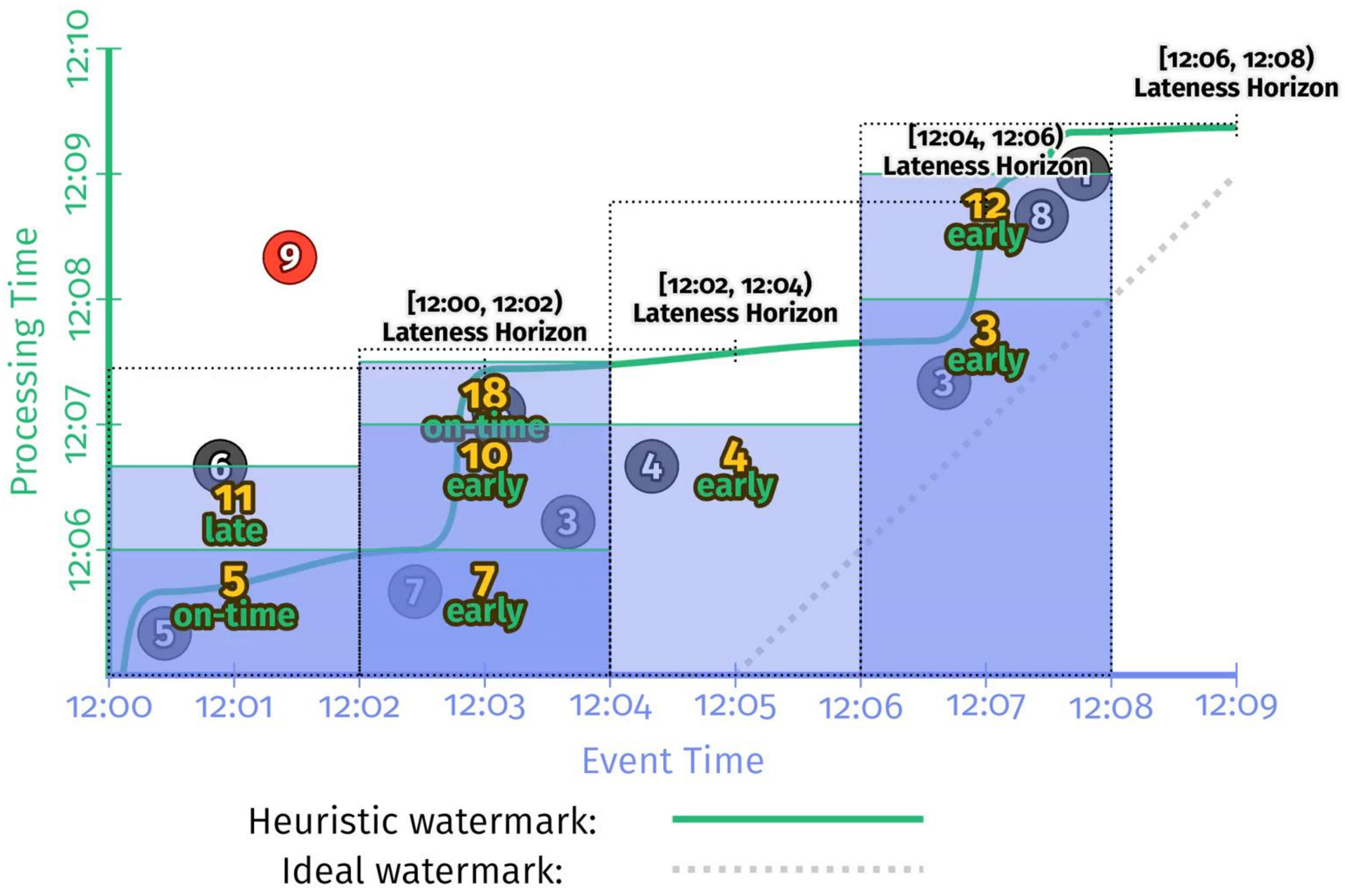
http://streamingsystems.net/fig/2-10

Watermarks meet triggers



- Early/on-time/late triggers
  - Zero or more early triggers **periodically** firing
  - Single **on-time** trigger on completeness/watermark
  - Zero or more triggers for **late data** unaccounted for by the watermark

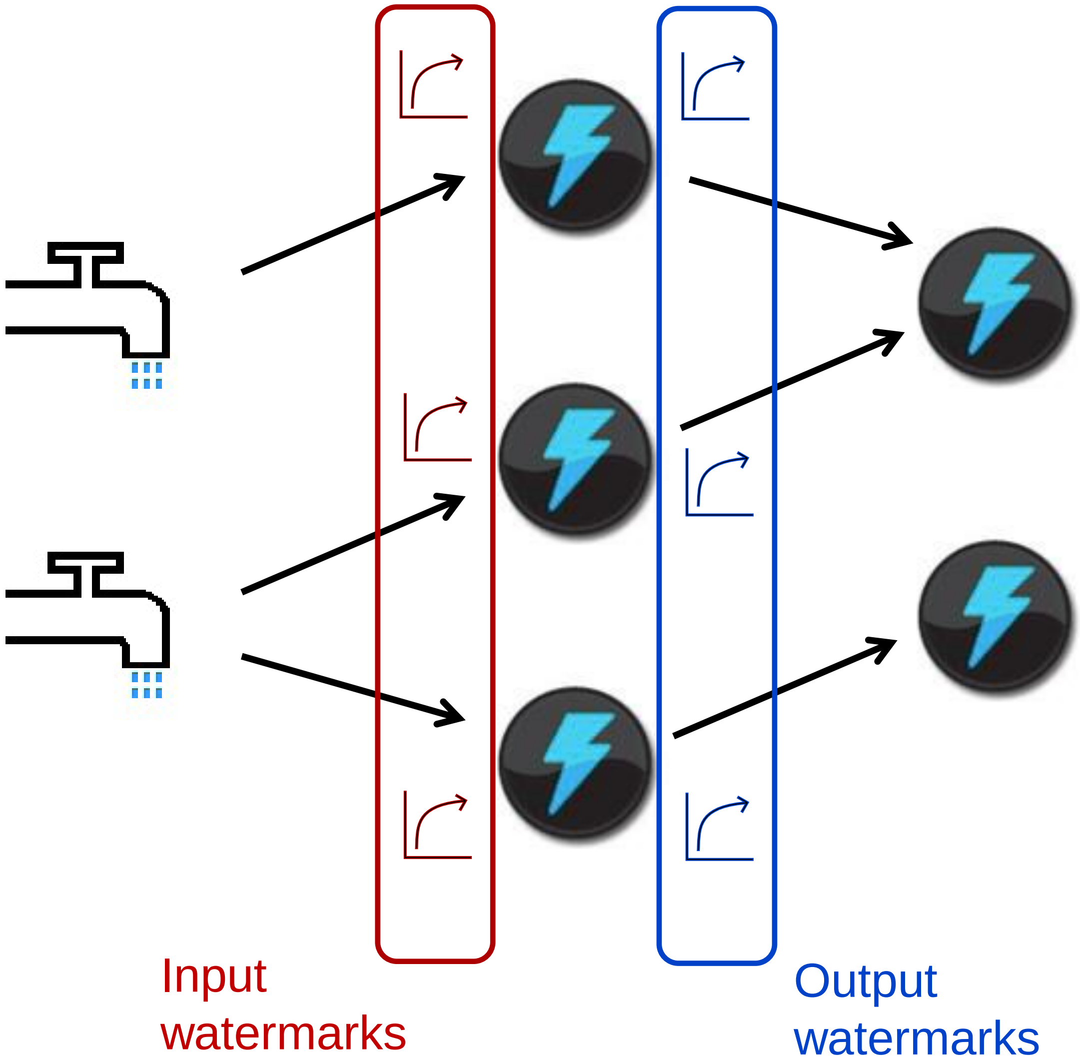
Allowed lateness

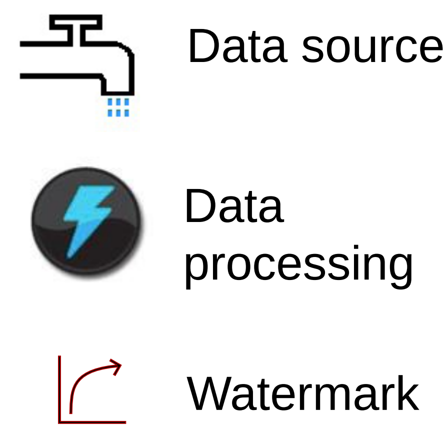


http://streamingsystems.net/fig/2-12

Watermark @ boundaries

* Apache Storm logic/icons





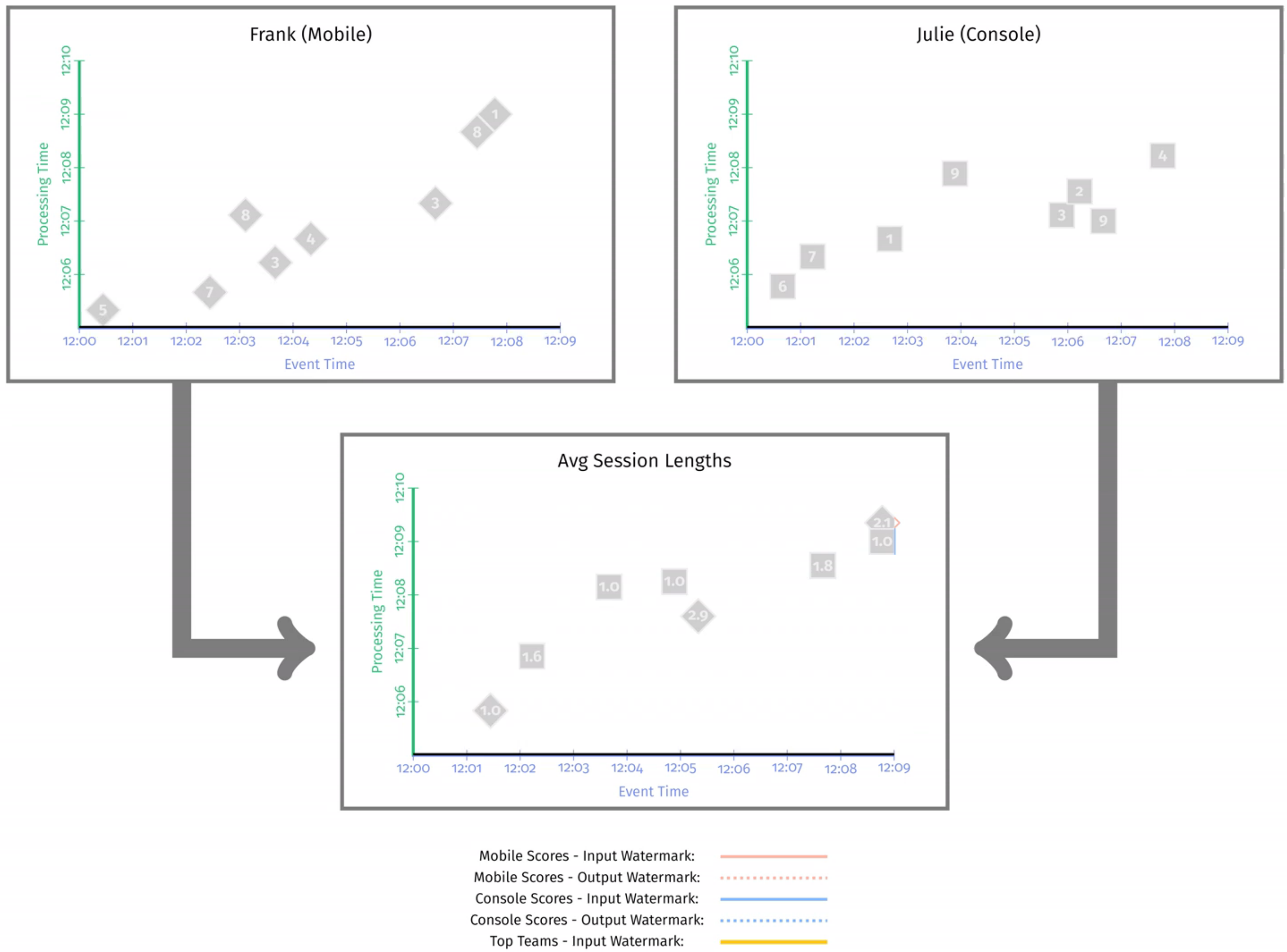
Input watermarks

- **Input watermark** = calculated based on the data processing pipeline’s progress upstream (‘to the left’) of the input of the current processing stage
- For (data) sources there is a specific function generating the watermark. It is created based on our understanding of the input data source
- For non-sources it is the **minimum** of all dependent output watermarks upstream
  - Why the minimum?

Output watermarks

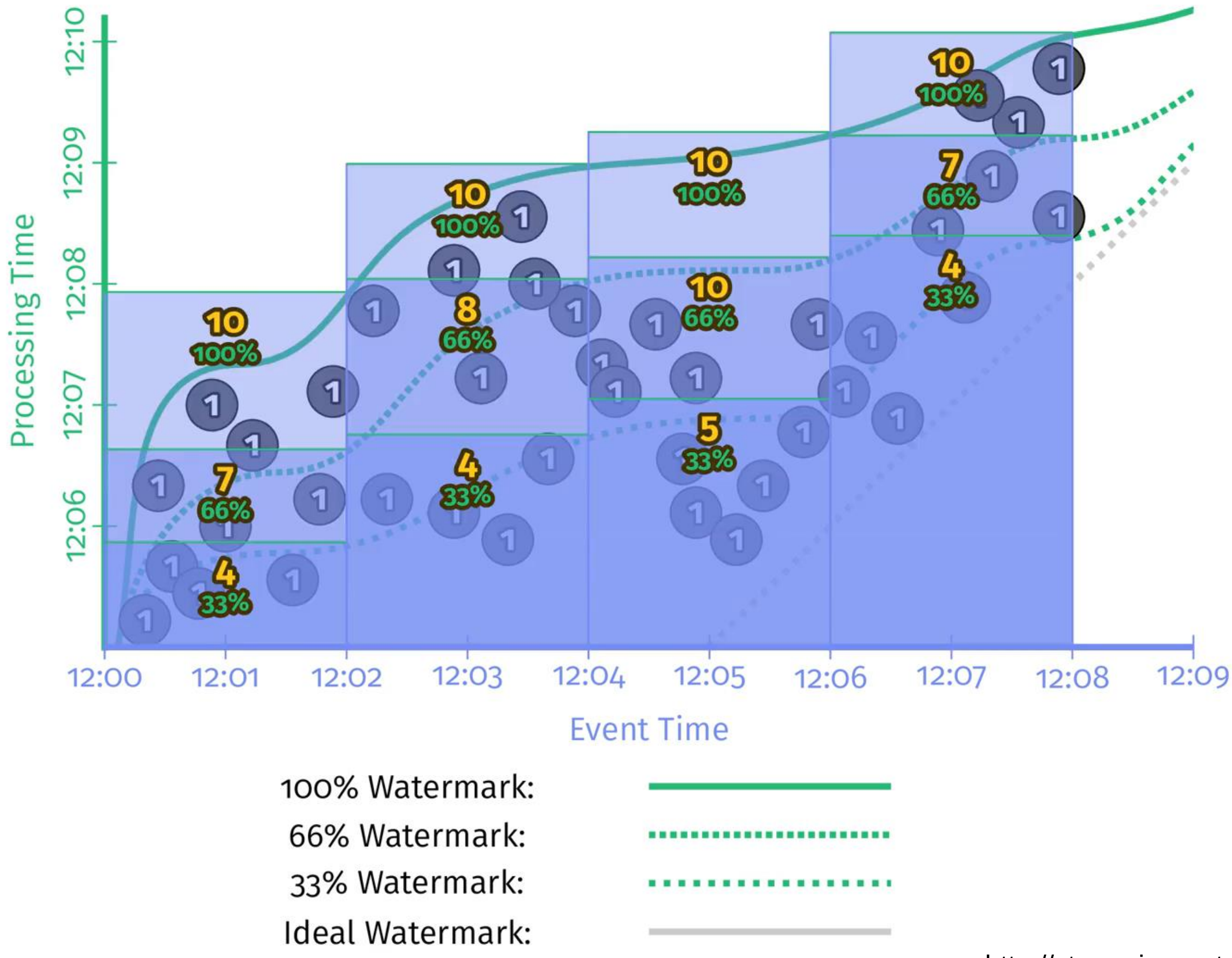
- Each processing node can **buffer** active messages until some operation is completed, e.g. aggregation in a previous stage.
  - Each buffer can be tracked with its own watermark
- **DEF:** Output watermarks are calculated at the output(s) of processing nodes and incorporate/observe input + processing (time) delay
- Output watermarks are (usually) based on a combination of the following watermarks
  - Data source → source watermarks
  - External input → other external sources, e.g. upstream processing
  - (Internal) state → there is an internal state-machine inside the processing node with clearly defined rules of progress
  - Per-output buffer → times of (data) processing results written to output buffers

Watermark propagation



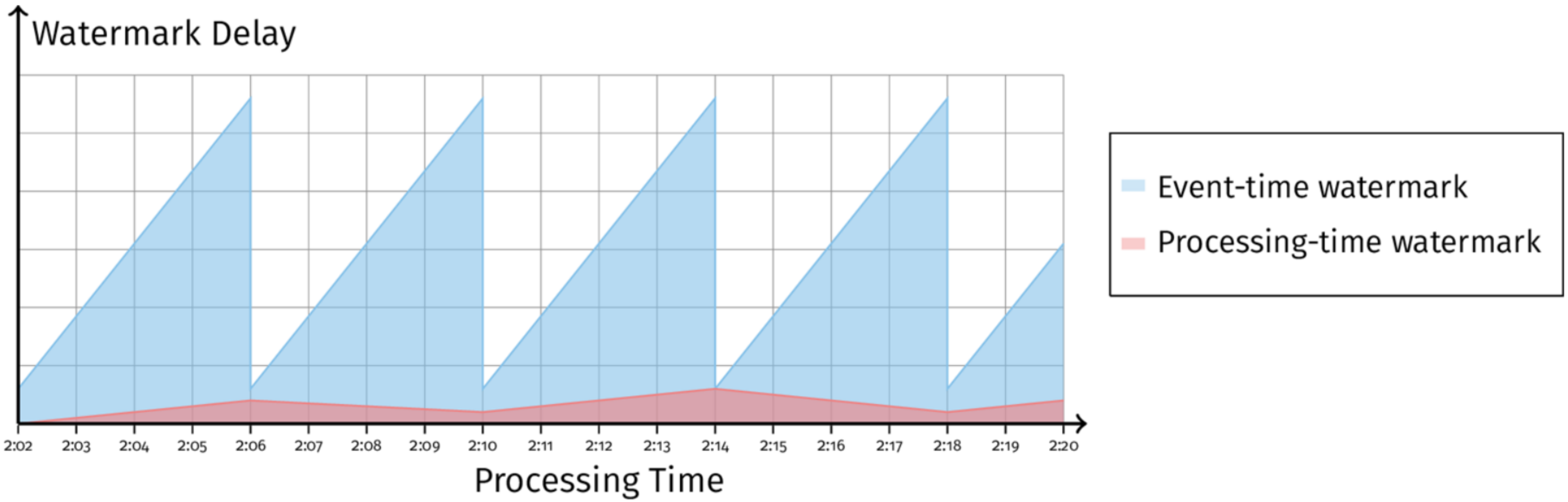
http://streamingsystems.net/fig/3-5

Percentile watermarks



http://streamingsystems.net/fig/3-11

Processing time watermarks



- The processing time watermark is constructed based on the timestamp of the oldest (processing) operation not yet completed
  - Allows insight into processing delay separate from data delay
  - Growing processing watermark → faults preventing (process) operations to complete → user/administrator action needed
- Useful to distinguish (streaming) system latency from data (source) latency

http://streamingsystems.net/fig/3-15

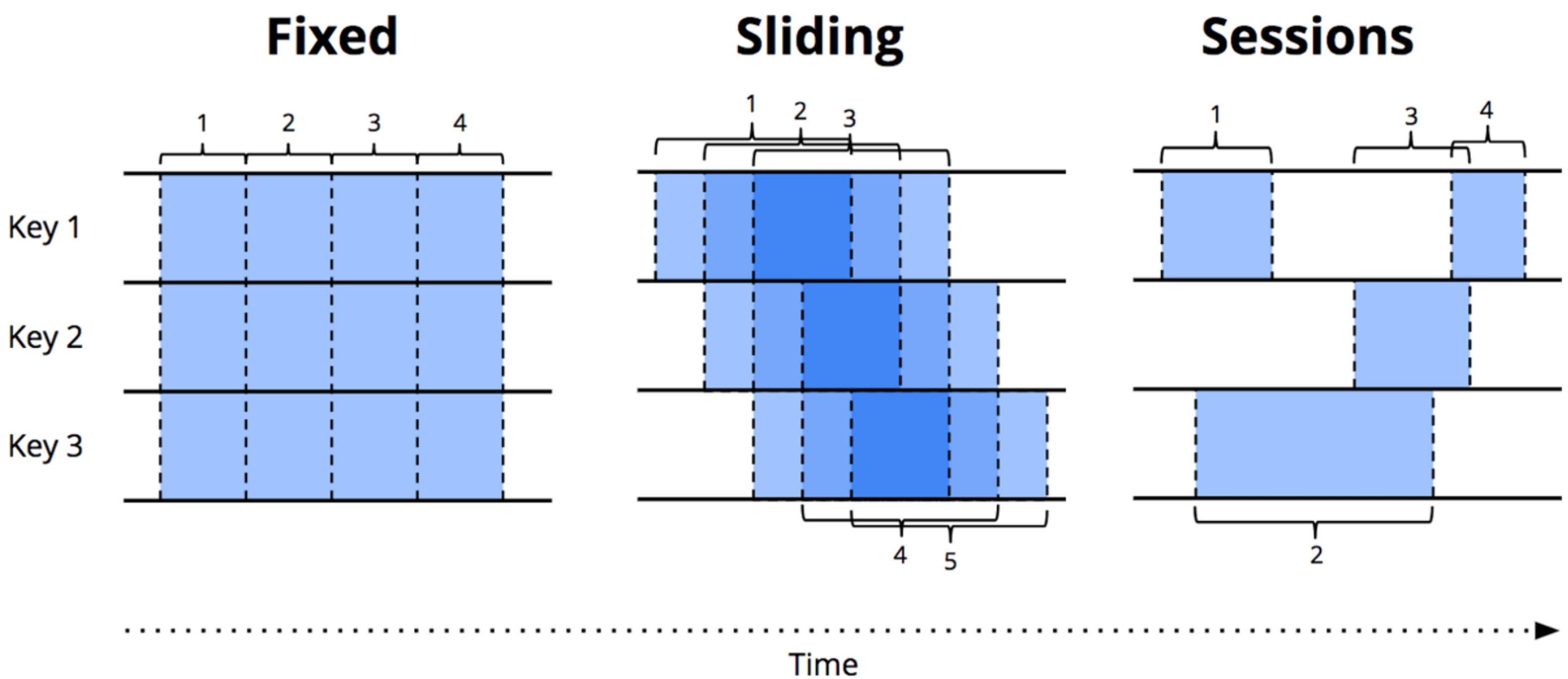
Stream mining

WINDOWING

Advanced windowing intro

- Processing-time windowing is useful for use cases in which data is observed as received, e.g. web server traffic
  - Implementation via triggers or ingress time
  - Time-sensitive → use only when data is received in perfect order or when we do not care about when events actually happenned
- Event-time windowing is used in use cases in which the exact sequence of events is relevant, e.g. billing, user behavior
  - Order agnostic, i.e. does not care about the order in which the data is received as these solutions are able to re-order pieces of data received from different sources in jumbled order

General windowing strategies



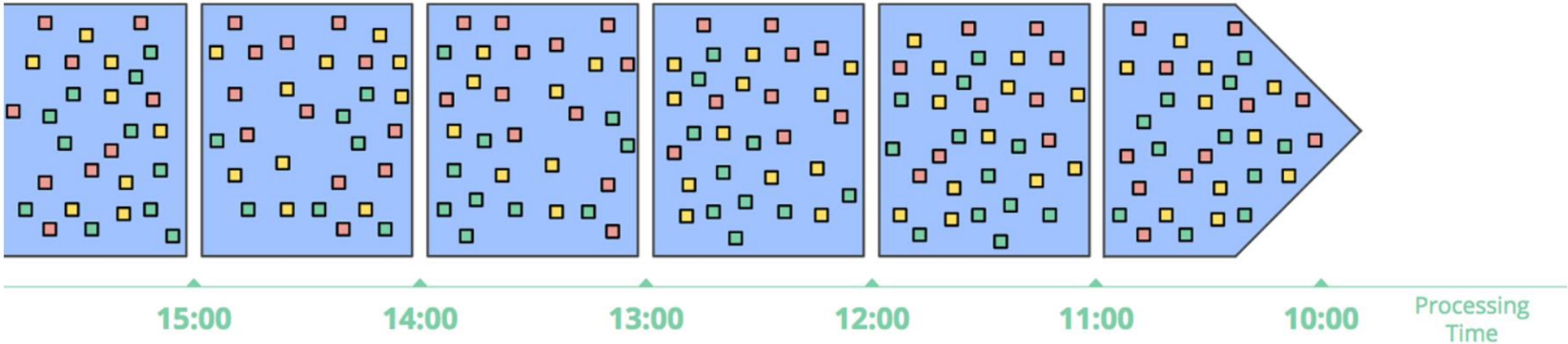
http://streamingsystems.net/fig/1-8

- Fixed windows → Slice time into fixed-sized temporal length, usually across all keys/variables.
- Sliding windows → Defined by fixed length & fixed period.
- Sessions → Sequences of events terminated by a gap.

Stream mining

PROCESSING-TIME WINDOWING

Windowing by processing time



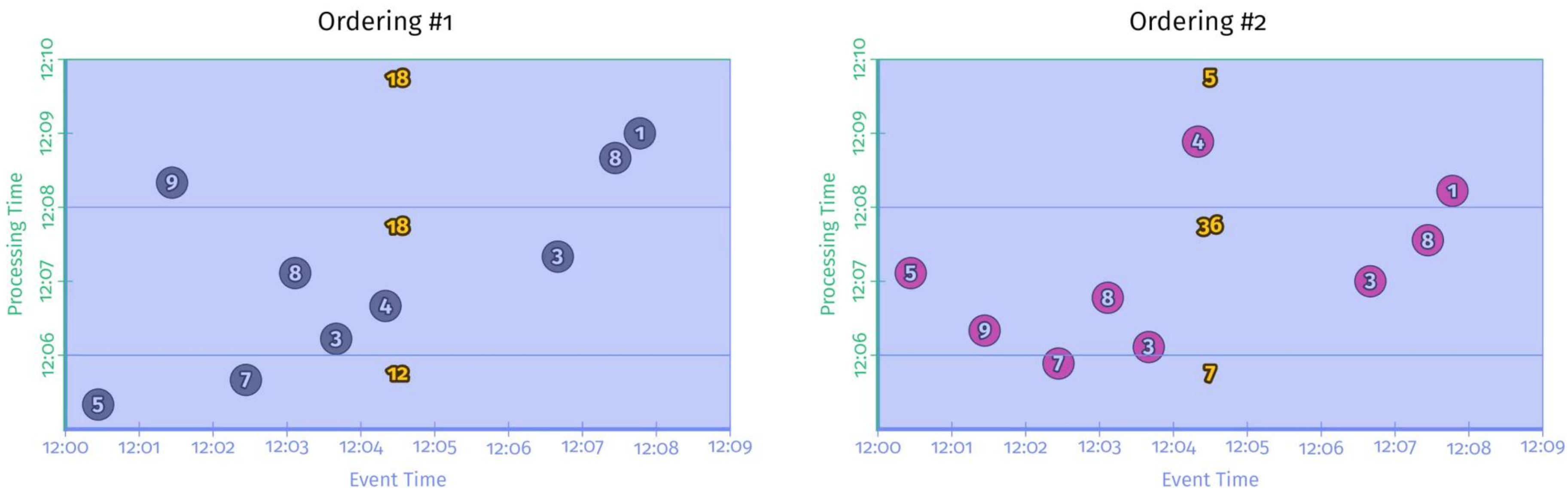
http://streamingsystems.net/fig/1-9

- Windows are created based on absolute (processing) time, i.e. data is put into windows based on the order they arrive in

Processing-time windowing & triggers

- Triggers fire periodically in the processing-time domain and initiate data processing, e.g. every 2 minutes
- Accumulating or discarding depending on the use case
- Most traditional streaming systems worked like this

PT with triggers in action

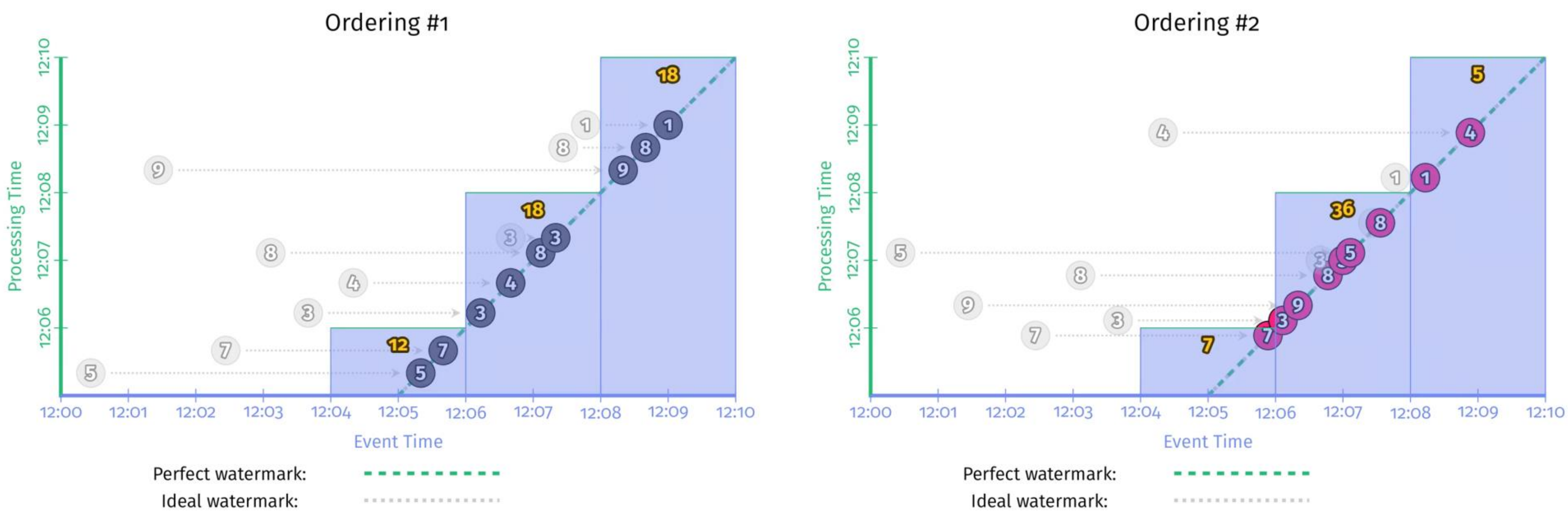


- Processing-time windowing via triggers can create different results for the same input data set with different processing-time ordering

Processing-time windowing & ingress time

- Ingress time windowing basics
  - Event times are replaced with ingress times, i.e. the times when they were received by the streaming system
  - Utilize perfect watermarks made possible via ingress time → processing is triggered when the watermark is reached
  - Accumulation mode is based on the use case
- Outputs can be different for different ordering of the same set of input data

PT with ingress time in action

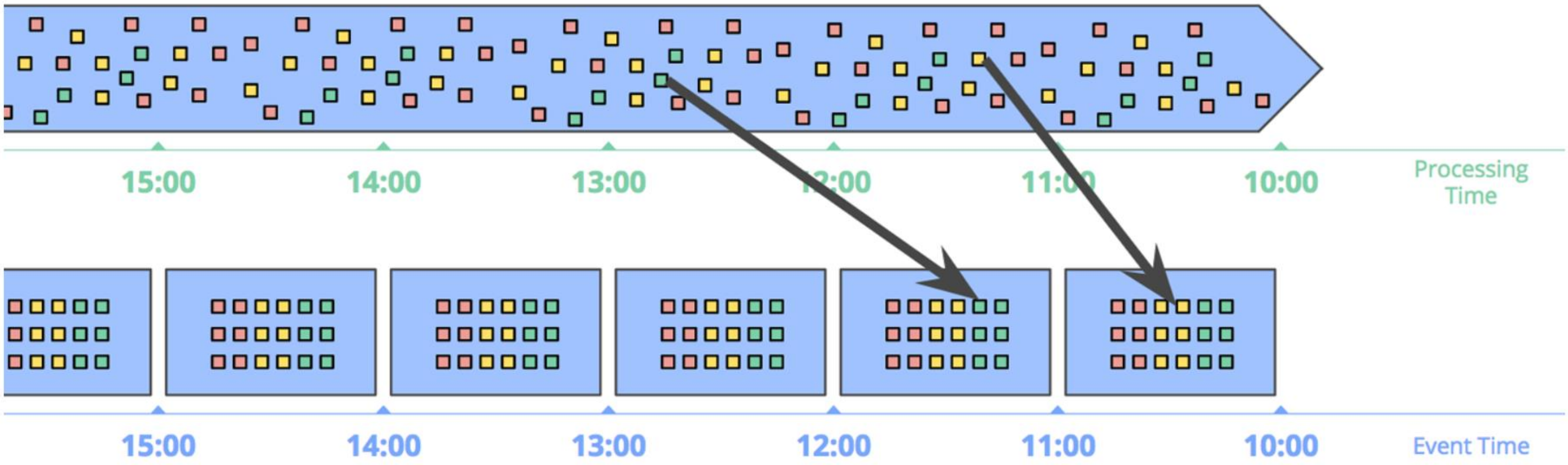


- Note: a perfect watermark is made possible via the use of ingress time

Stream mining

EVENT-TIME WINDOWING

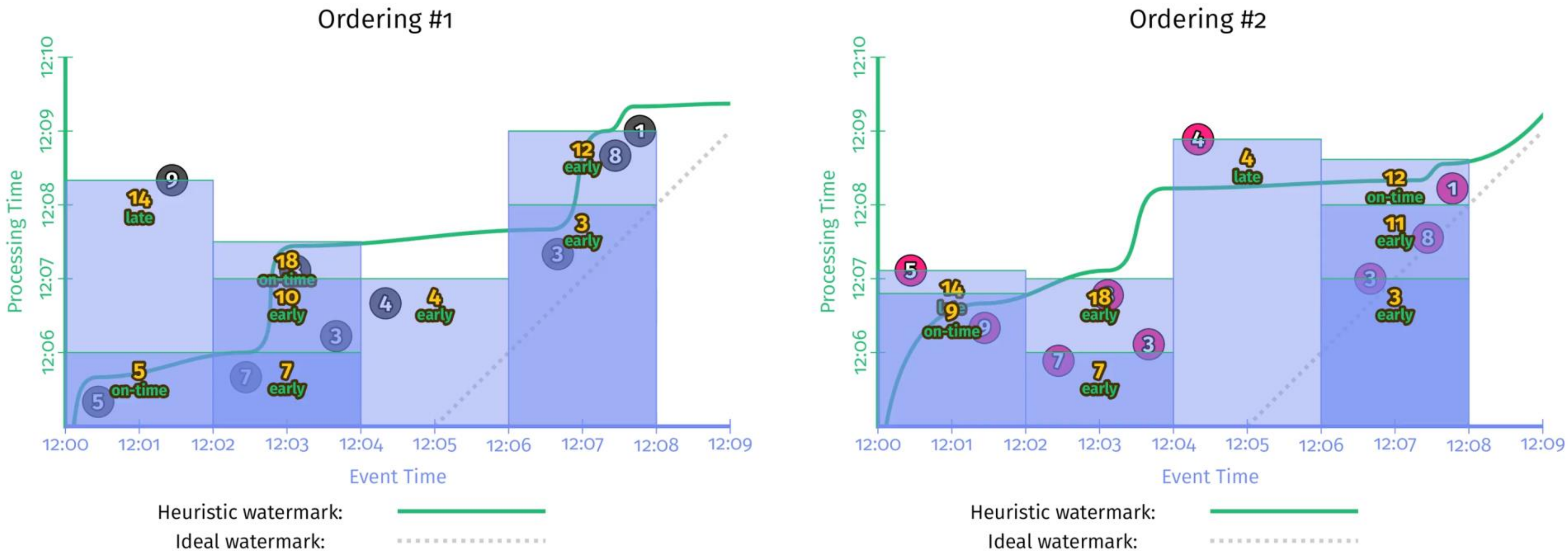
Windowing by event time – Fixed



http://streamingsystems.net/fig/1-10

- Data is collected and windowed based on times at which it occurred, i.e. event time.

Event-time windowing in action



- Sum calculus in event-time windows over the same pieces of data in two different processing time orderings.

http://streamingsystems.net/fig

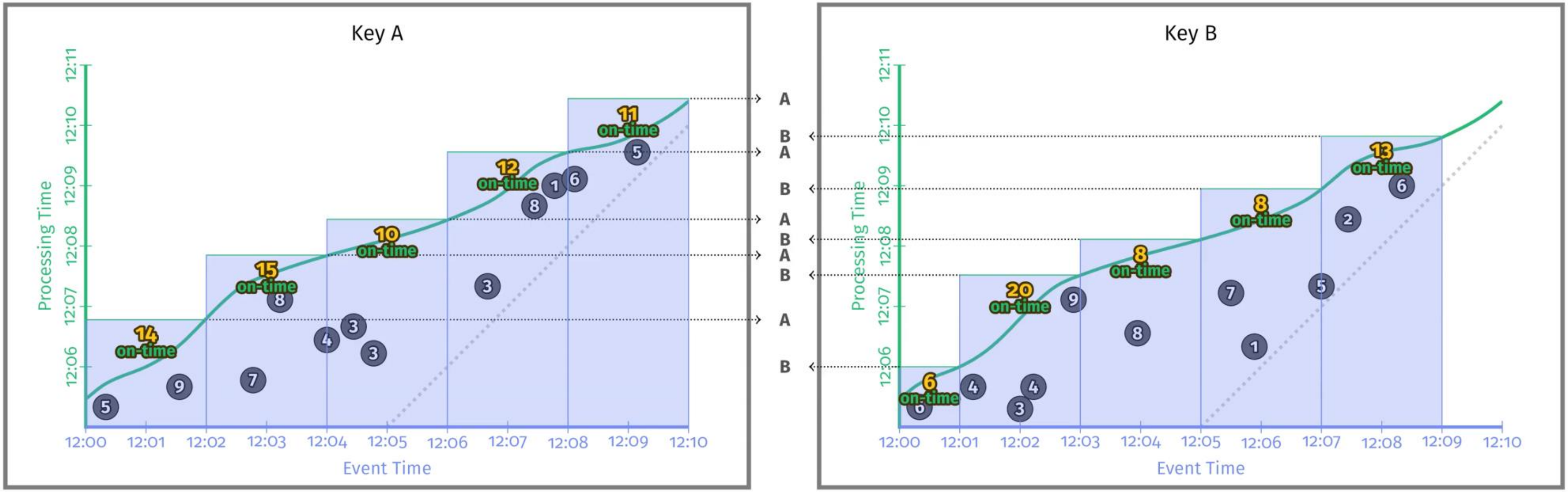
Variation #1: Aligned fixed windows

- Aligned fixed windows start and end at the same time for all keys/observations
  - Example: thousands of measurement points on a factory floor with advanced motor monitoring with high frequency data sampling for early fault detection, which are processed every minute (in absolute, processing time)
- Pro:
  - Implementing fixed windows is relatively easy
- Contra:
  - The data is seldom occurring in a windowed fashion in real-life streaming systems
  - There is a high processing (CPU, RAM) load at the end of the windows when all windows are processed at the same time

Variation #2: Unaligned fixed windows

- Unaligned fixed windows do not start at the same time for all keys/observations
- The length of these windows is still pre-defined
- Pro:
  - More evenly spread processing loads
- Contra:
  - The data is seldom occurring in a windowed fashion in real-life streaming systems
  - Not as easy to implement

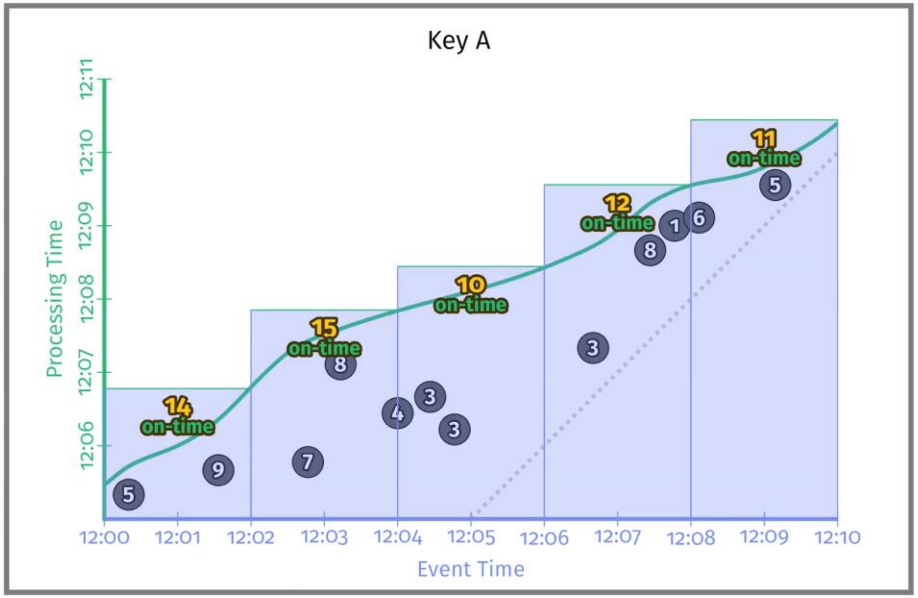
Unaligned fixed windows in action

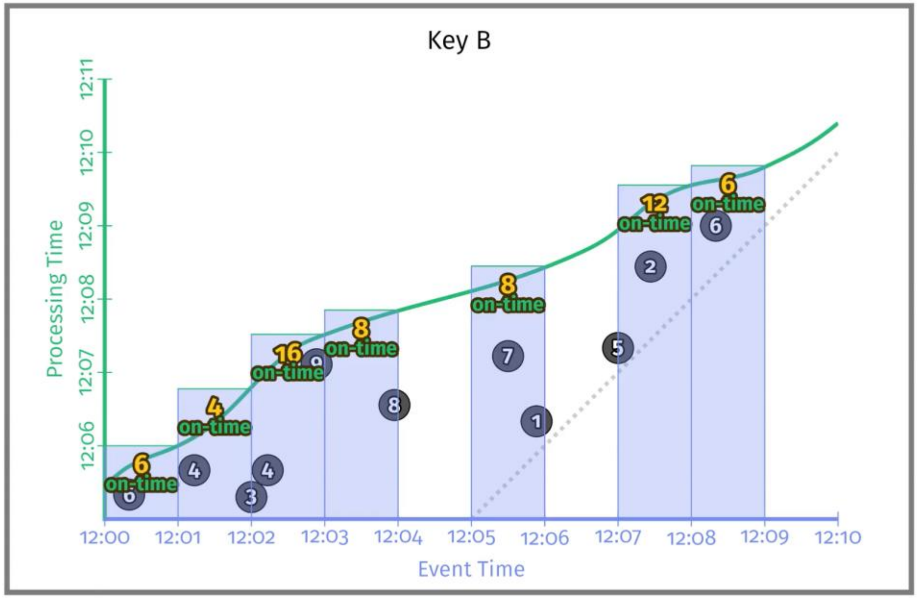


- The horizontal lines mark the ‘ends’ of the windows → that’s when processing is triggered

http://streamingsystems.net/fig

Variation #3: Per-element/key fixed windows





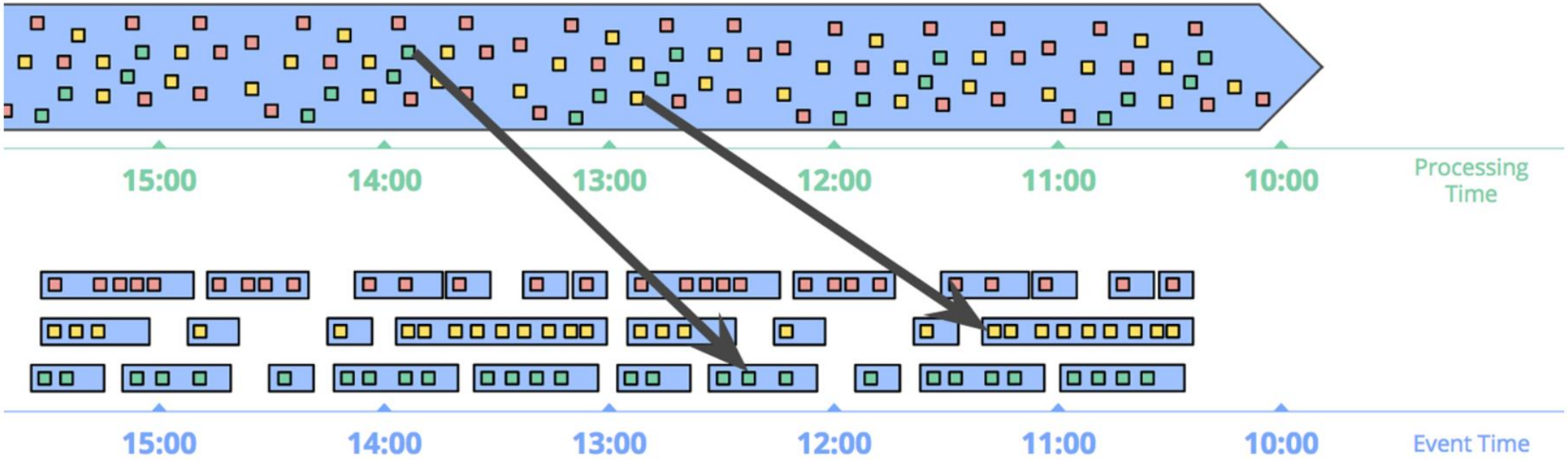
- Fixed windows with different key sizes:
  - Key A with fixed window size = 2 minute(s)
  - Key B with fixed window size = 1 minute(s)
- Useful when different stream analysis requirements exist

http://streamingsystems.net/fig

Stream mining

Sessions in windowing

Windowing by event time – Sessions



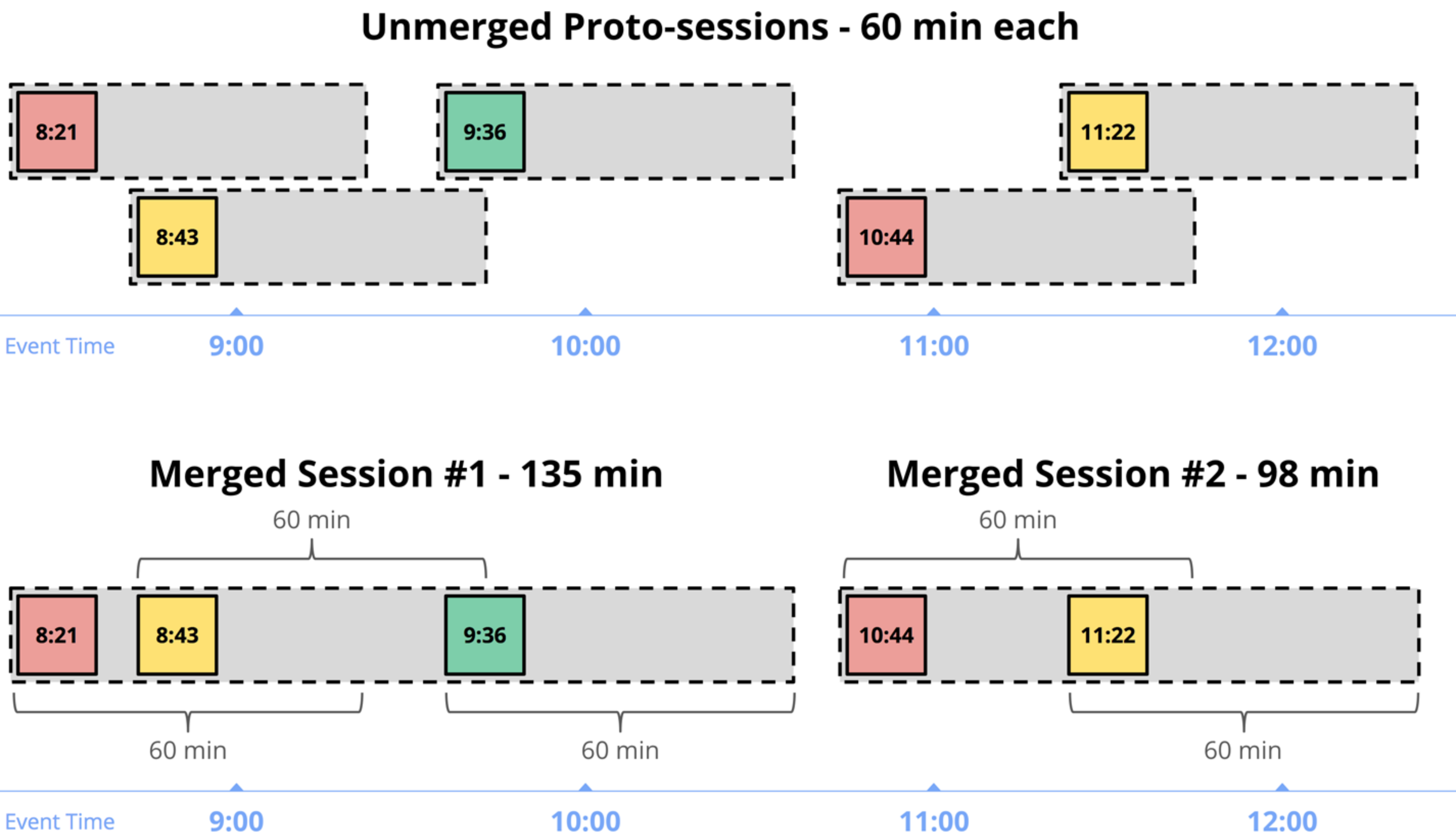
http://streamingsystems.net/fig/1-11

- Data is collected into sessions based on temporal proximity and key/user.

Session windows

- Session windows are data-driven → the location and length of windows is dictated by the input data itself
  - This is opposed to fixed windows, which are finalized after the expiration of a certain time period either in event or processing time
- Sessions are unaligned
  - There is absolutely no guarantee that data will occur at the exact same time for different keys
- Session creation can be based on
  - Predefined (time) gap which separates two sessions – this is done more often
  - Predefined tag which is assigned to each piece of data – this is a less frequent scenario

Proto sessions & merging

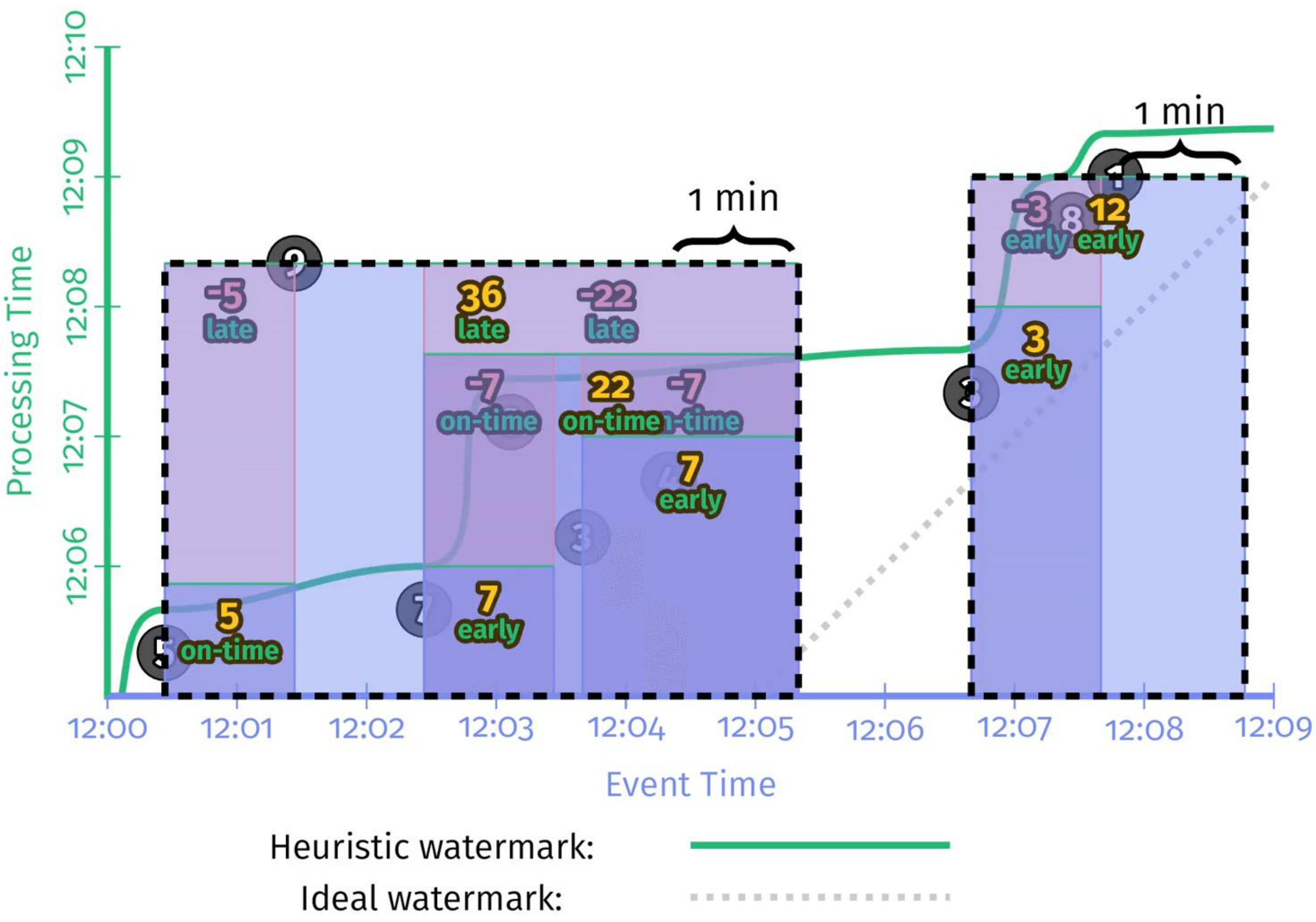


http://streamingsystems.net/fig

Session creation steps

- Phase 0: The gap duration is defined as the length of time between two sessions
- Phase I: Assignment
  - Each element is initially placed in a proto-session window
  - The proto-session starts with the occurrence of the event
  - The proto-session extends for the gap duration
- Phase II: Merging
  - A grouping strategy is defined
  - All eligible proto-sessions are sorted
  - All overlapping proto-sessions are grouped into sessions

Session creation in action

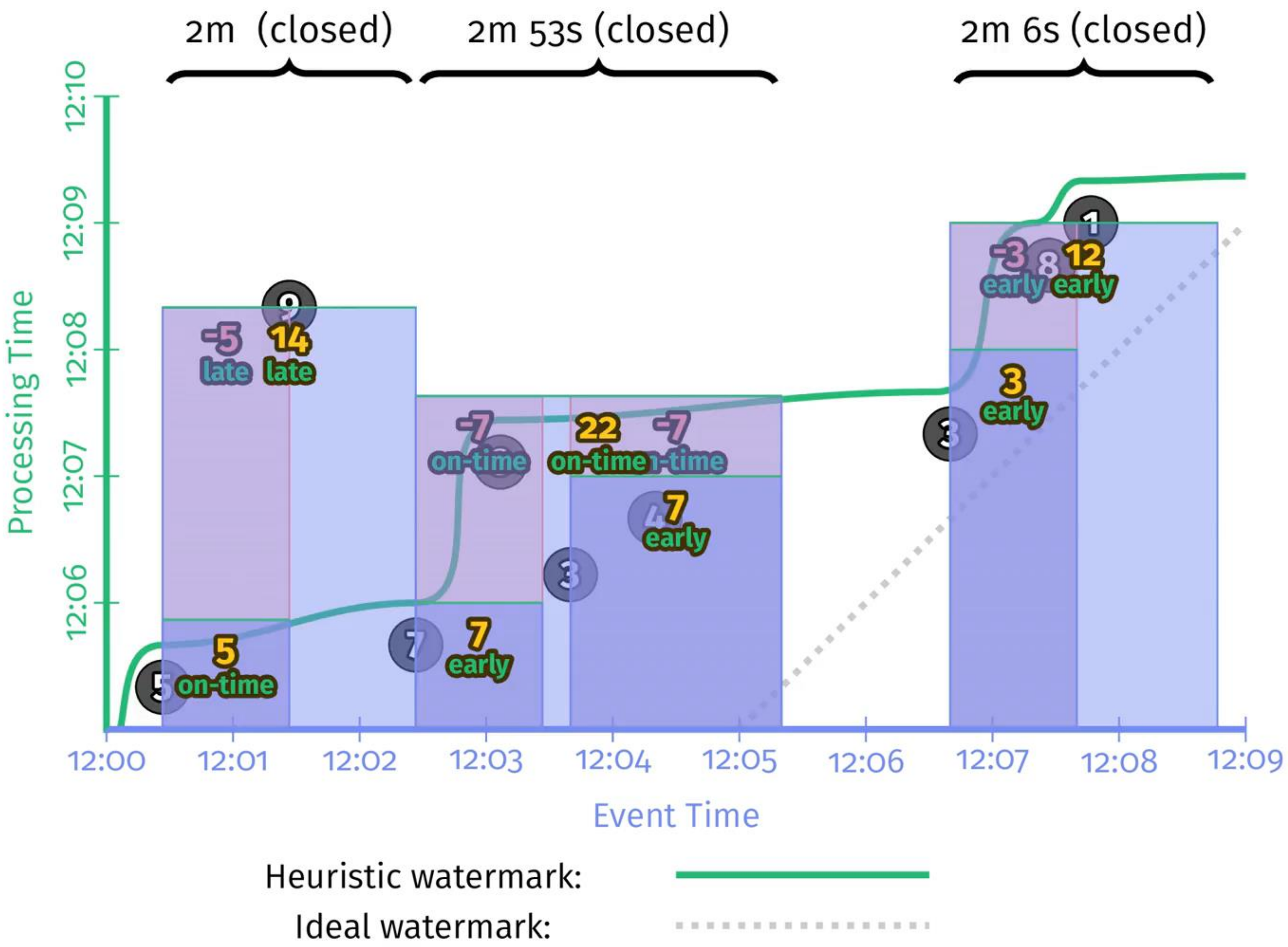


http://streamingsystems.net/fig

Variation #1: Bounded sessions

- Bounded sessions are not allowed to grow beyond a predefined size
- The ‘size’ can be defined in time, element count, aggregate value or other dimension
- Example: time-bounded sessions, e.g. a maximum session length of 3 minutes

Bounded sessions in action



http://streamingsystems.net/fig

Summary

- Watermarks
  - Perfect vs heuristic
  - Input & output
  - Percentile
  - Processing time
- Advanced windowing
  - Processing-time
  - Event-time
  - Sessions
  - No one-size-fits-all windowing strategy !

Thank you for your attention!