# CS 591 K1: Data Stream Processing and Analytics

**Spring 2020**

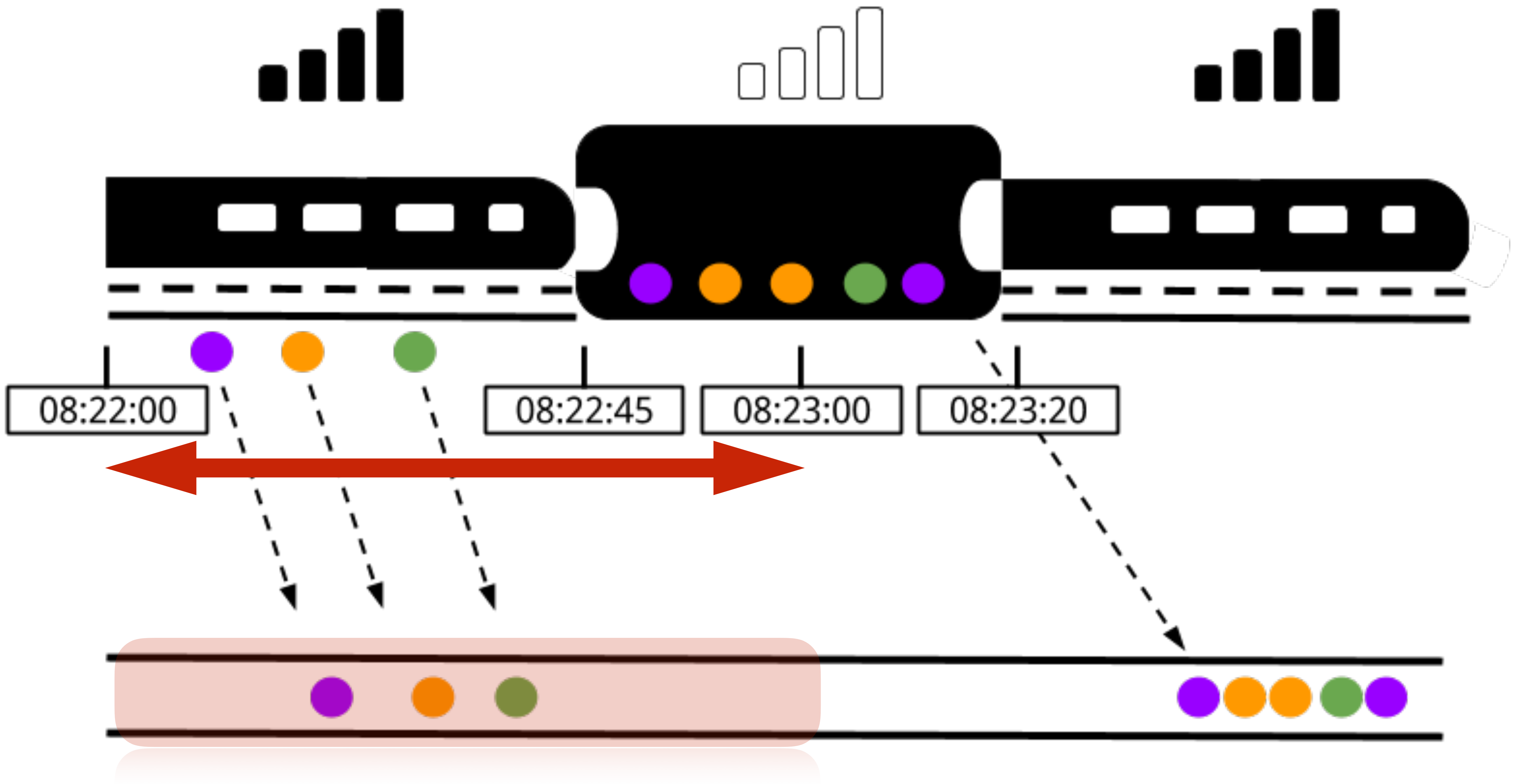
2/06: Notions of time and progress

**Vasiliki (Vasia) Kalavri**
**vkalavri@bu.edu**

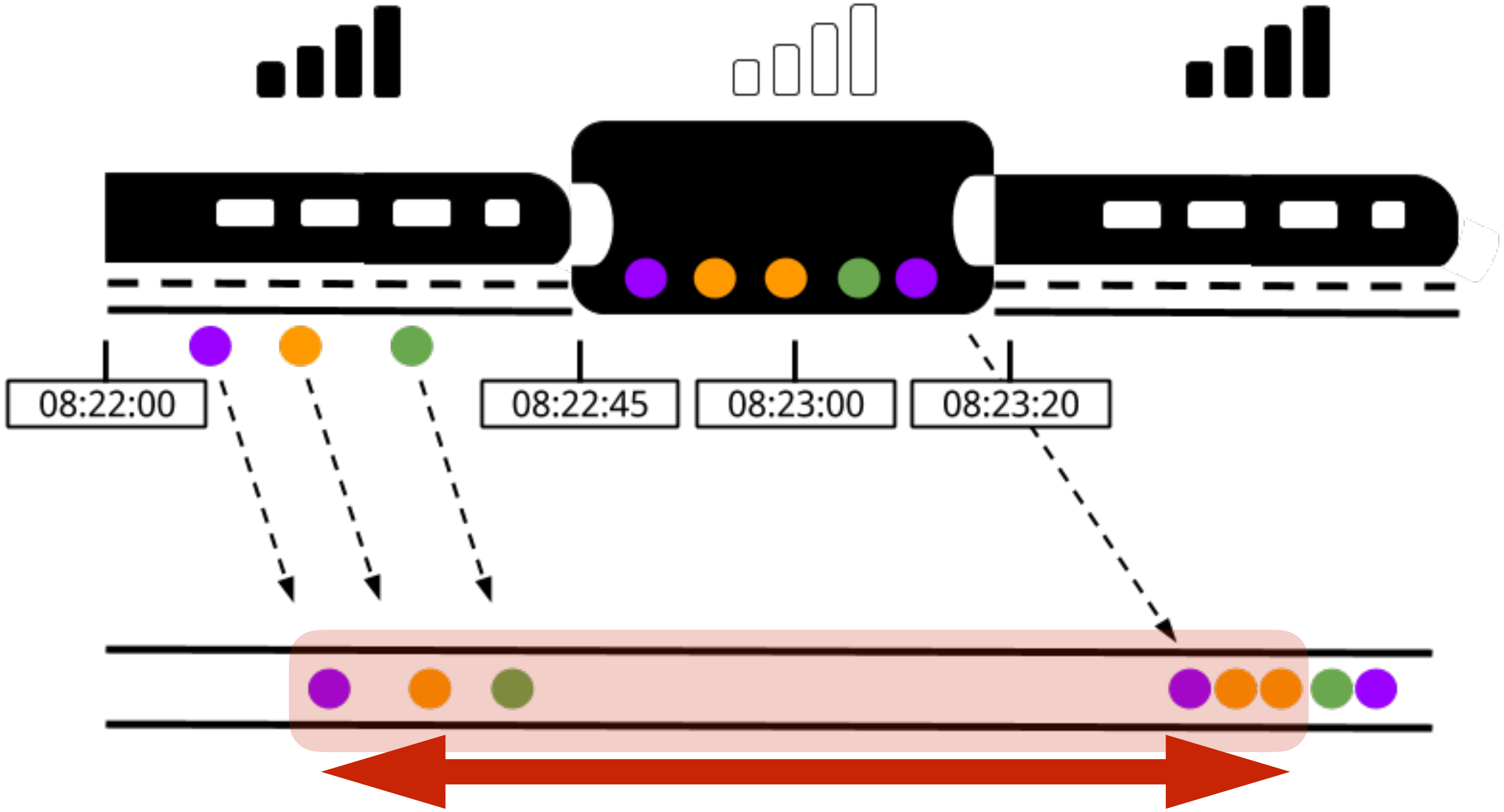
Mobile game application

- input stream: user activity
- output: rewards based on how fast the user meets goals
- e.g. pop 500 bubbles within 1 minute and get extra life

# What’s the meaning of one minute?



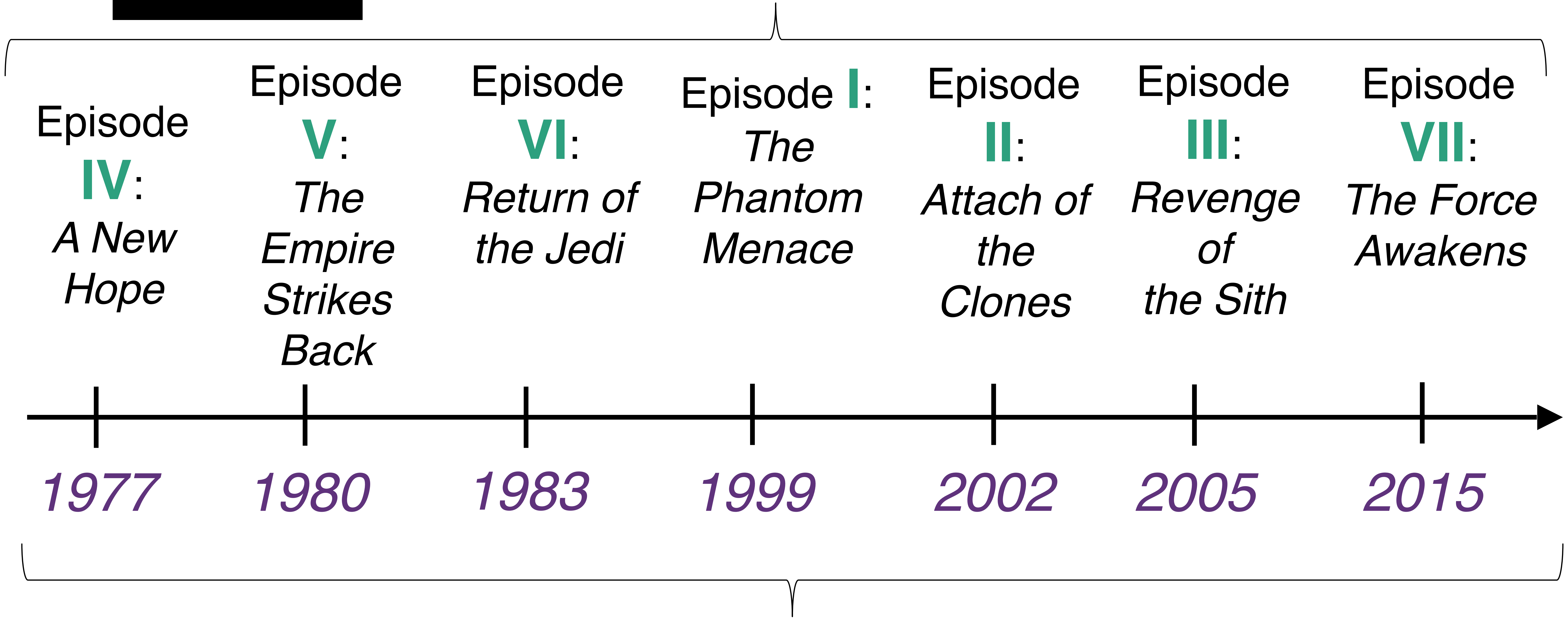
# What’s the meaning of one minute?



# Notions of time

- **Processing** time
  - the time of the local clock where an event is being processed
  - a processing-time window wouldn’t account for game activity while the train is in the tunnel
  - results depend on the processing speed and aren’t deterministic
- **Event** time
  - the time when an event actually happened
  - an event-time window would give you the extra life
  - results are deterministic and independent of the processing speed

This is called **event time**



Episode **IV**: *A New Hope*

Episode **V**: *The Empire Strikes Back*

Episode **VI**: *Return of the Jedi*

Episode **I**: *The Phantom Menace*

Episode **II**: *Attach of the Clones*

Episode **III**: *Revenge of the Sith*

Episode **VII**: *The Force Awakens*

*1977* *1980* *1983* *1999* *2002* *2005* *2015*

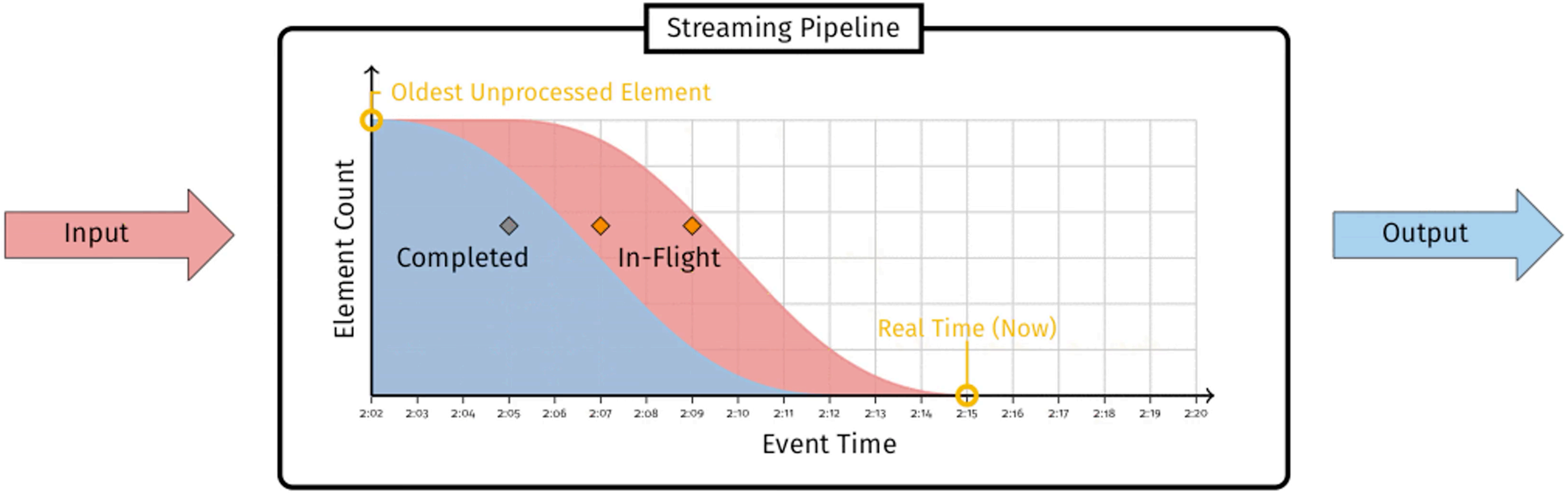
This is called ***processing time***

## How do we know what *event* time it is?

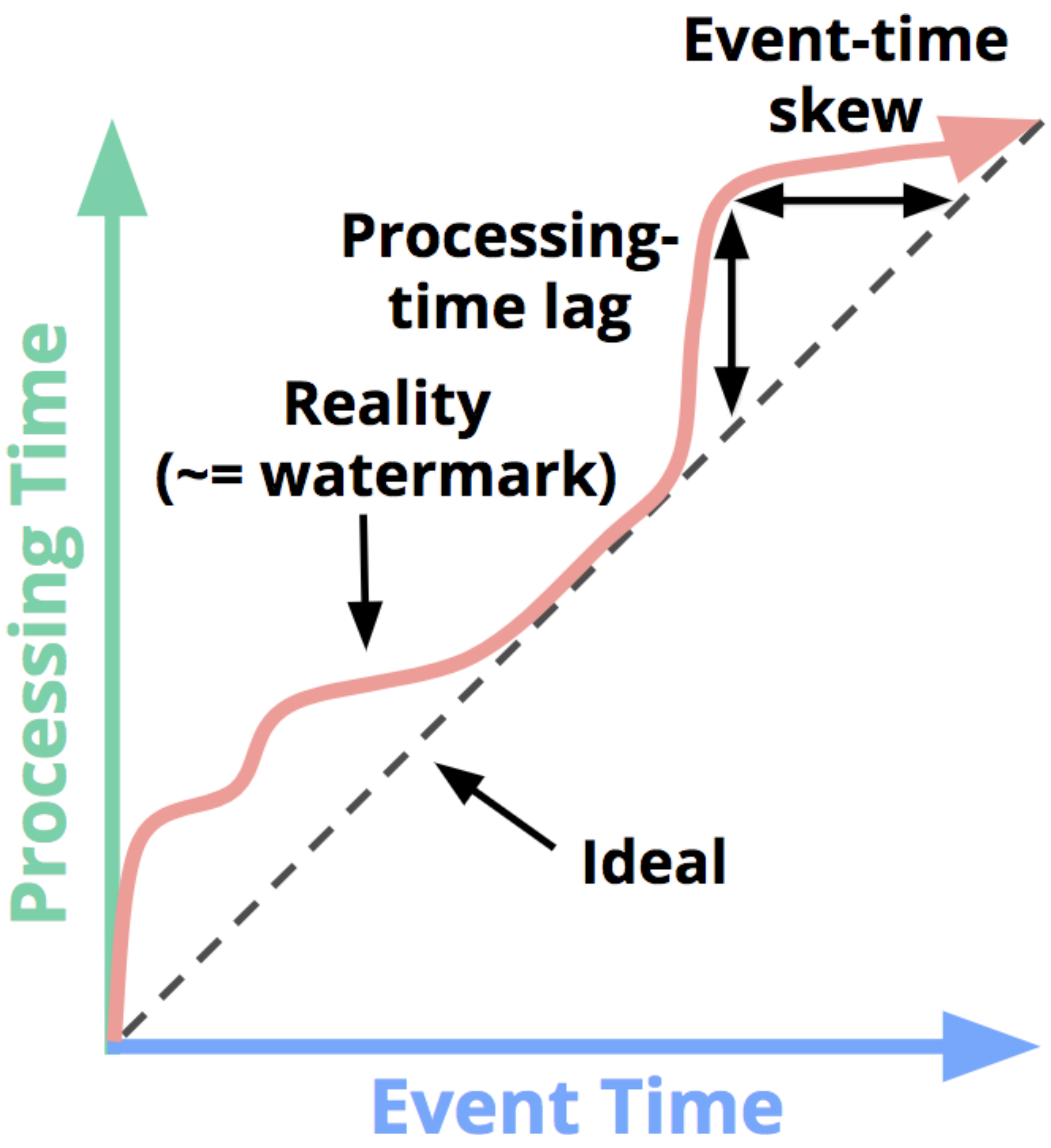
- What if you were in a plane and not on a train?
- What if you never came back online?
- How long do we have to wait before we decide that we have seen all events?

# Watermarks

# Stream progress



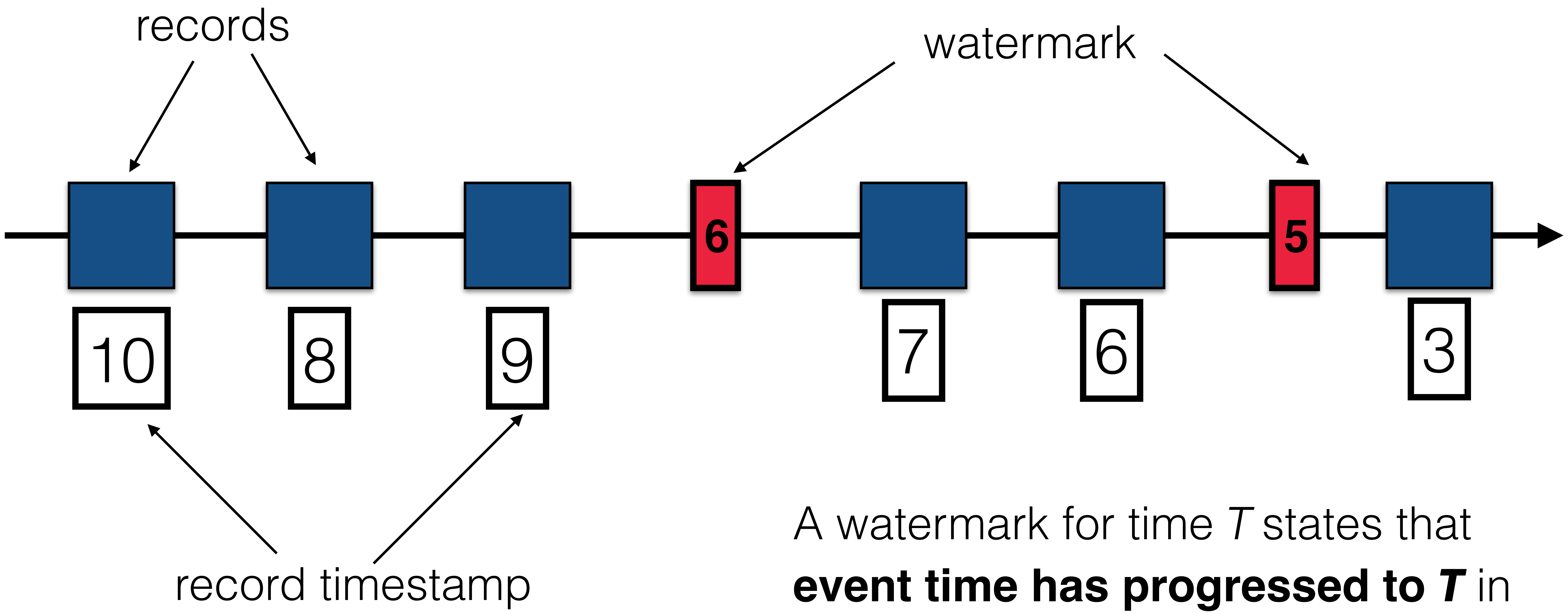
http://streamingbook.net/fig/3-1



http://streamingbook.net/fig/2-9

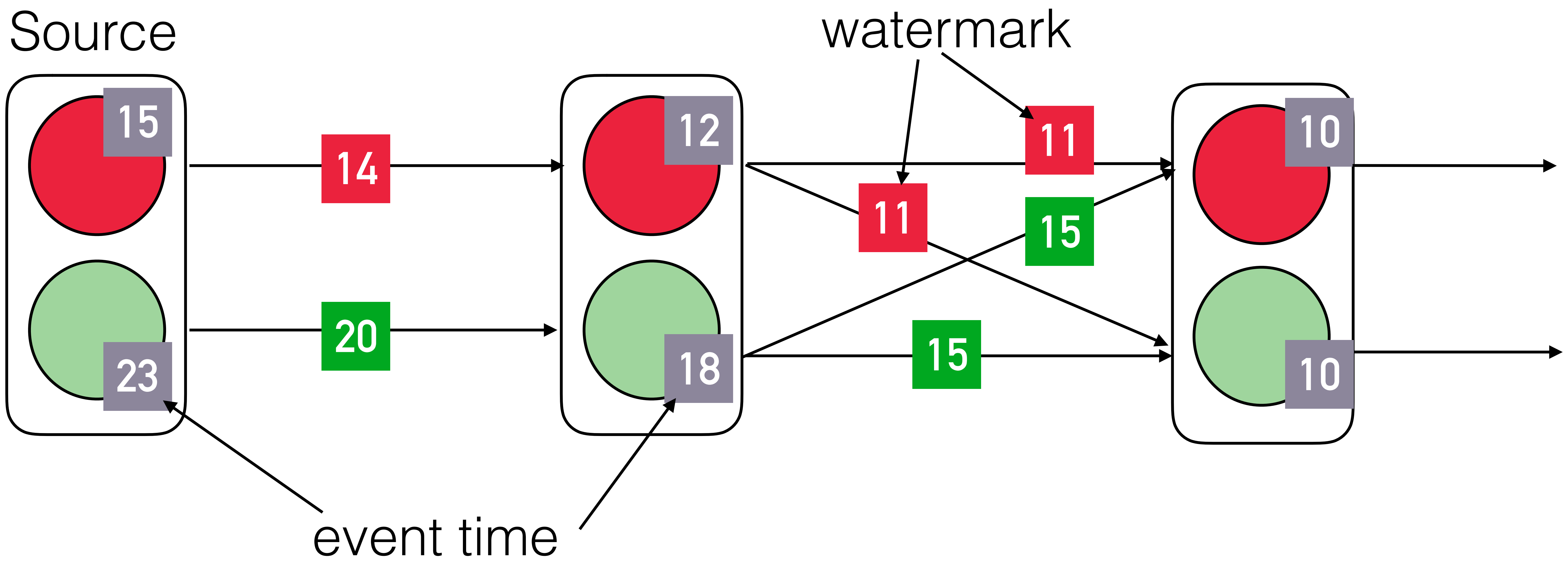
- A watermark is a **global progress metric** that indicates a certain point in time when we are confident that no more delayed events will arrive.
- Watermarks provide a **logical clock** which informs the system about the current event time.

Watermarks (in Flink) flow along dataflow edges. They are **special records** generated by the sources or assigned by the application.



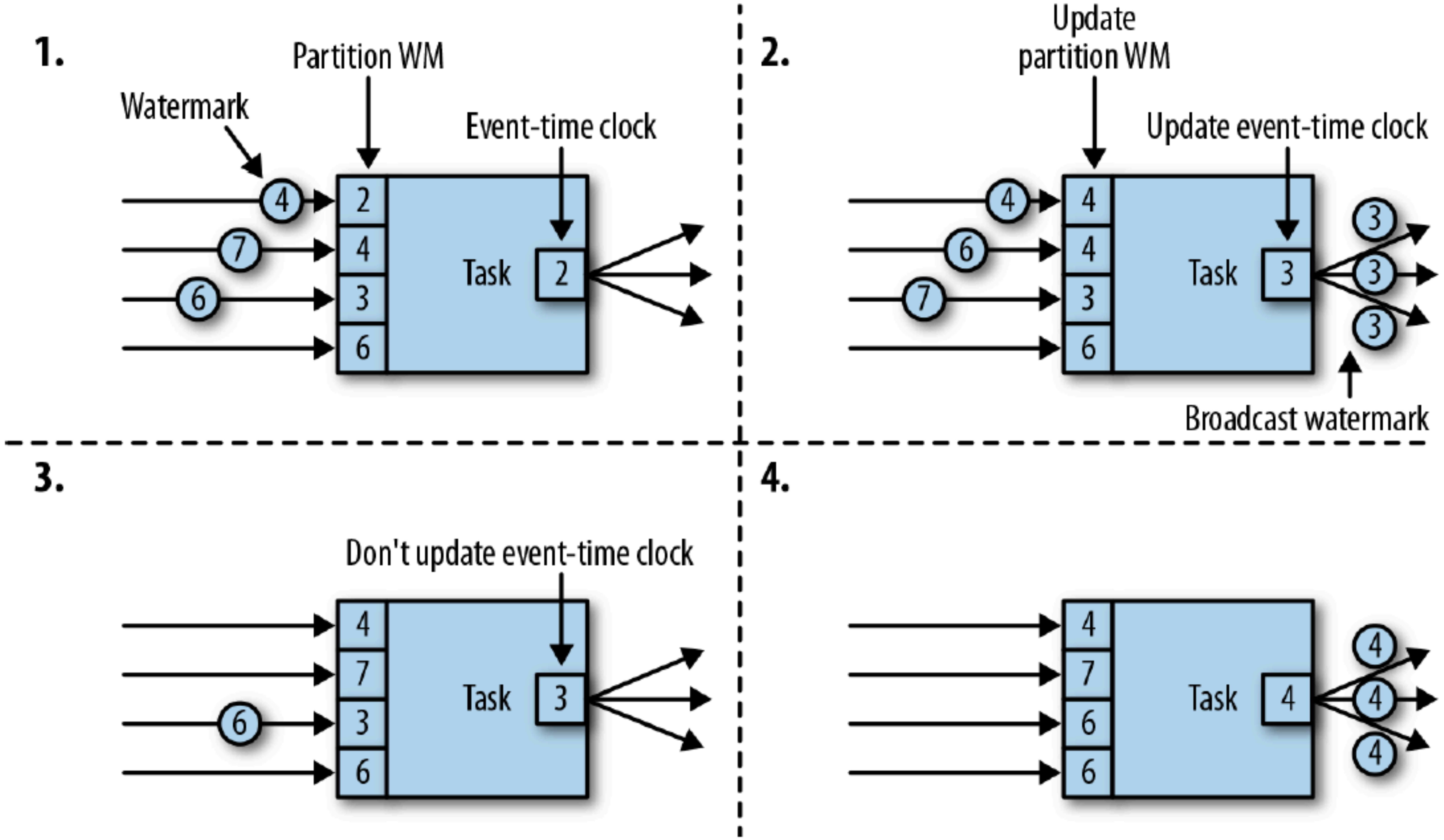
A watermark for time *T* states that **event time has progressed to *T*** in that particular stream (or partition).

# Watermark propagation



- The *input* watermark captures the progress of upstream stages
    - minimum of output watermarks of all upstream tasks
- The *output* watermark captures the progress of the stage itself
    - minimum of input watermarks and event-times of non-late data

# Event-time update



# Watermark properties

1. Watermarks must be **monotonically increasing** in order to ensure that the event time clocks of tasks are progressing and not going backwards.

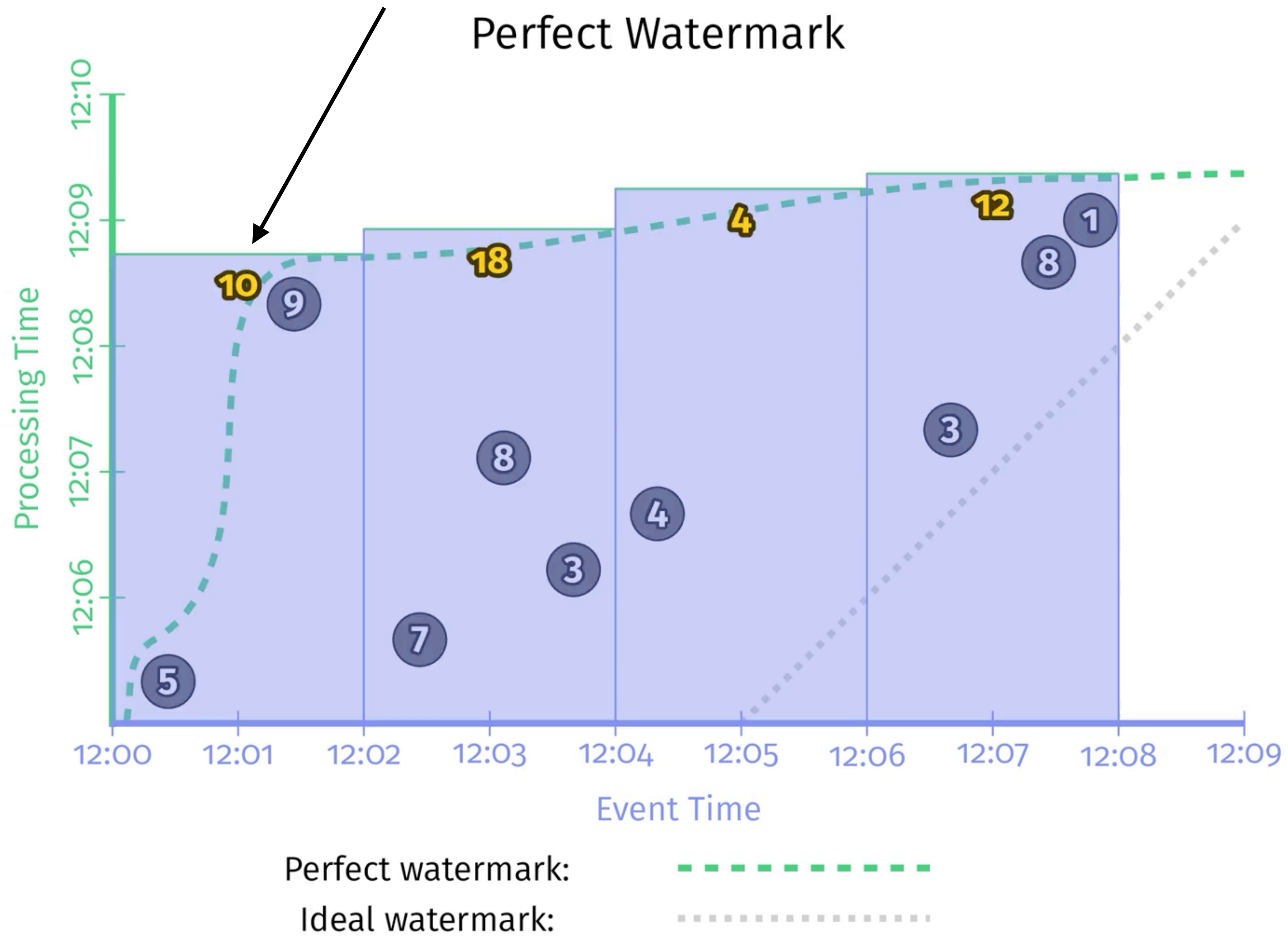
2. A watermark with a timestamp *T* indicates that all subsequent records should have timestamps > *T*.

# Evaluation of event-time windows

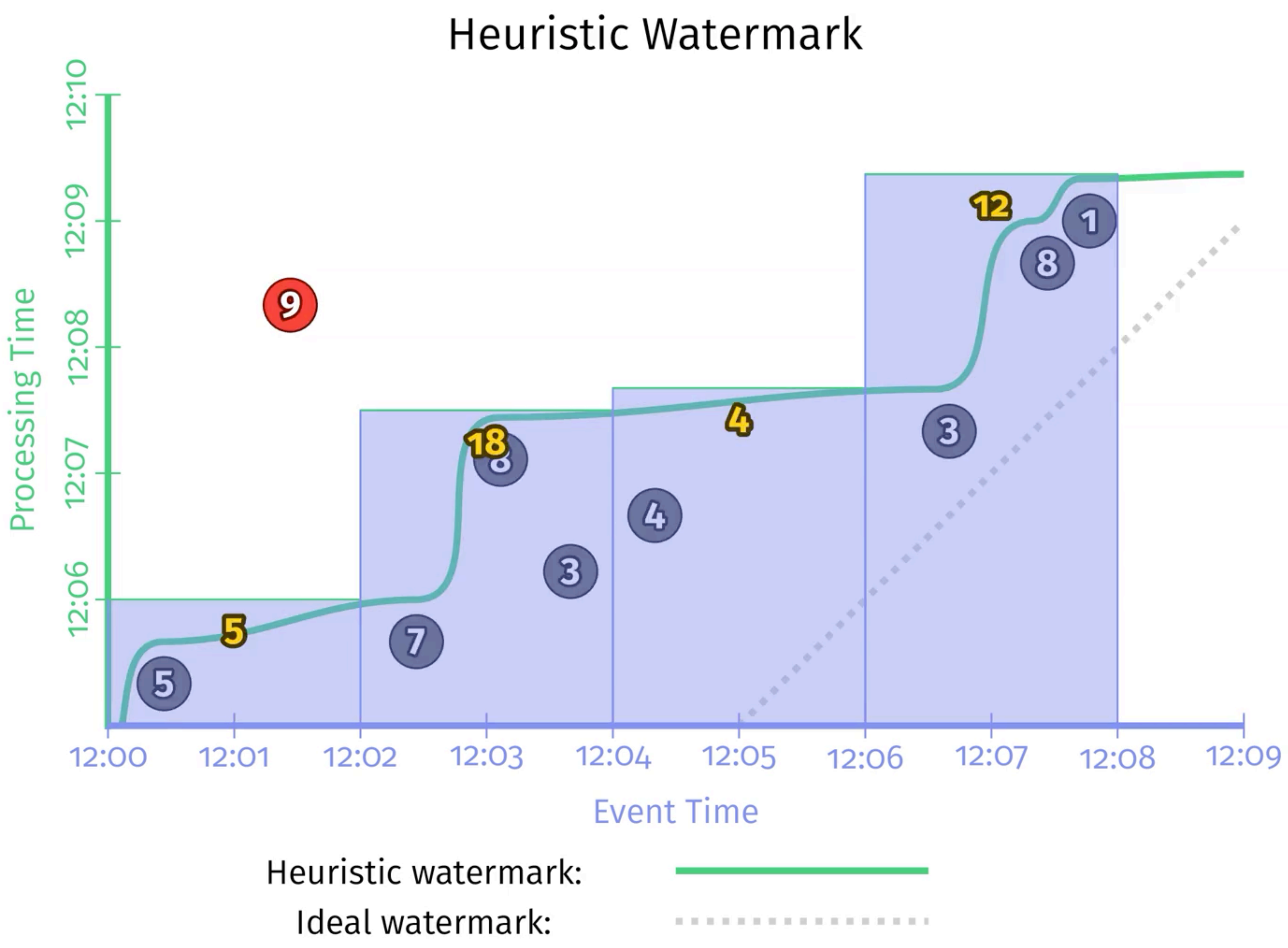
Watermarks are essential to both event-time windows and operators handling out-of-order events:

- When an operator receives a watermark with time *T*, it can assume that no further events with timestamp less than *T* will be received.
- It can then either trigger computation or order received events.

14



Perfect Watermark



Heuristic Watermark

http://streamingbook.net/fig/3-2

Vasiliki Kalavri | Boston University 2020

# Trade-offs

Watermarks provide a configurable trade-off between **results confidence** and **latency**:

- *Eager* watermarks ensure low latency but provide lower confidence
  - Late events might arrive after the watermark
- *Slow* watermarks increase confidence but they might lead to higher processing latency.

# Watermarks in Flink

**Periodic**: periodically ask the user-defined function for the current watermark timestamp.

**Punctuated**: check for a watermark in each passing record, e.g. if the stream contains special records that encode watermark information.

```
val env = StreamExecutionEnvironment.getExecutionEnvironment
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)
// generate watermarks every 5 seconds
env.getConfig.setAutoWatermarkInterval(5000)
```

```
class PeriodicAssigner
  extends AssignerWithPeriodicWatermarks[Reading] {

 val bound: Long = 60 * 1000 // 1 min in ms
 var maxTs: Long = Long.MinValue // the max observed timestamp

 override def getCurrentWatermark: Watermark = {
 // generated watermark with 1 min tolerance
 new Watermark(maxTs - bound)
 }

 override def extractTimestamp(r: Reading, prevTS: Long): Long = {
  // update maximum timestamp
  maxTs = maxTs.max(r.timestamp)
  // return record timestamp
  r.timestamp
 }
}
```

```
class PunctuatedAssigner
  extends AssignerWithPunctuatedWatermarks[Reading] {

 val bound: Long = 60 * 1000 // 1 min in ms

 override def checkAndGetNextWatermark(
   r: Reading,
   extractedTS: Long): Watermark = {

  if (r.id == "sensor_1") {
   // emit watermark if reading is from sensor_1
   new Watermark(extractedTS - bound)
  }
  else {
   // do not emit a watermark
   null
  }
 }

 override def extractTimestamp(r: Reading, prevTS: Long): Long = {
  // assign record timestamp
  r.timestamp
 }
}
```

# Using a watermark assigner

```
val env = StreamExecutionEnvironment.getExecutionEnvironment

// set the event time characteristic
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime)

// ingest sensor stream
val readings: DataStream[Reading] = env.addSource(new SensorSource)

// assign timestamps and generate watermarks
.assignTimestampsAndWatermarks(new MyAssigner())
```

# Further reading

- Streaming 102: The world beyond batch: https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102
- Watermarks, Tables, Event Time, and the Dataflow Model: https://www.confluent.jp/blog/watermarks-tables-event-time-dataflow-model/