DataStream API

Windows & Time

Apache Flink® Training

dataArtisans

Flink v1.2.0 – 22.04.2017

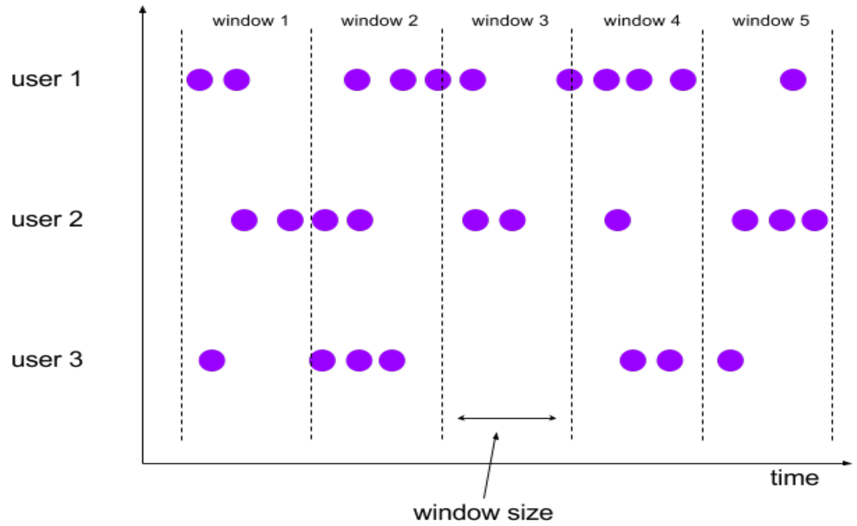
Windows and Aggregates

Windows

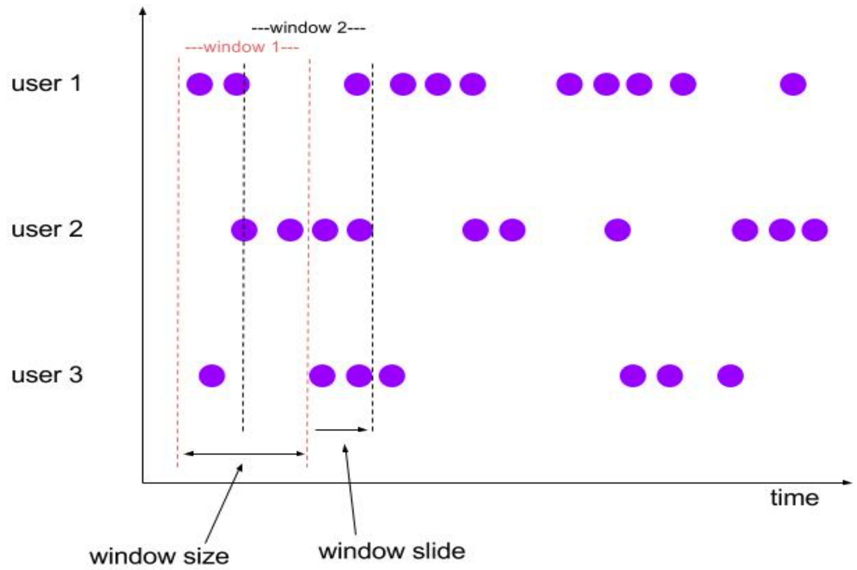
- Aggregations on DataStreams are different from aggregations on DataSets
  - You cannot count all records of an unbounded stream

- Aggregations make sense on windowed streams
  - A window is a finite subset of stream elements

Tumbling and Sliding Windows



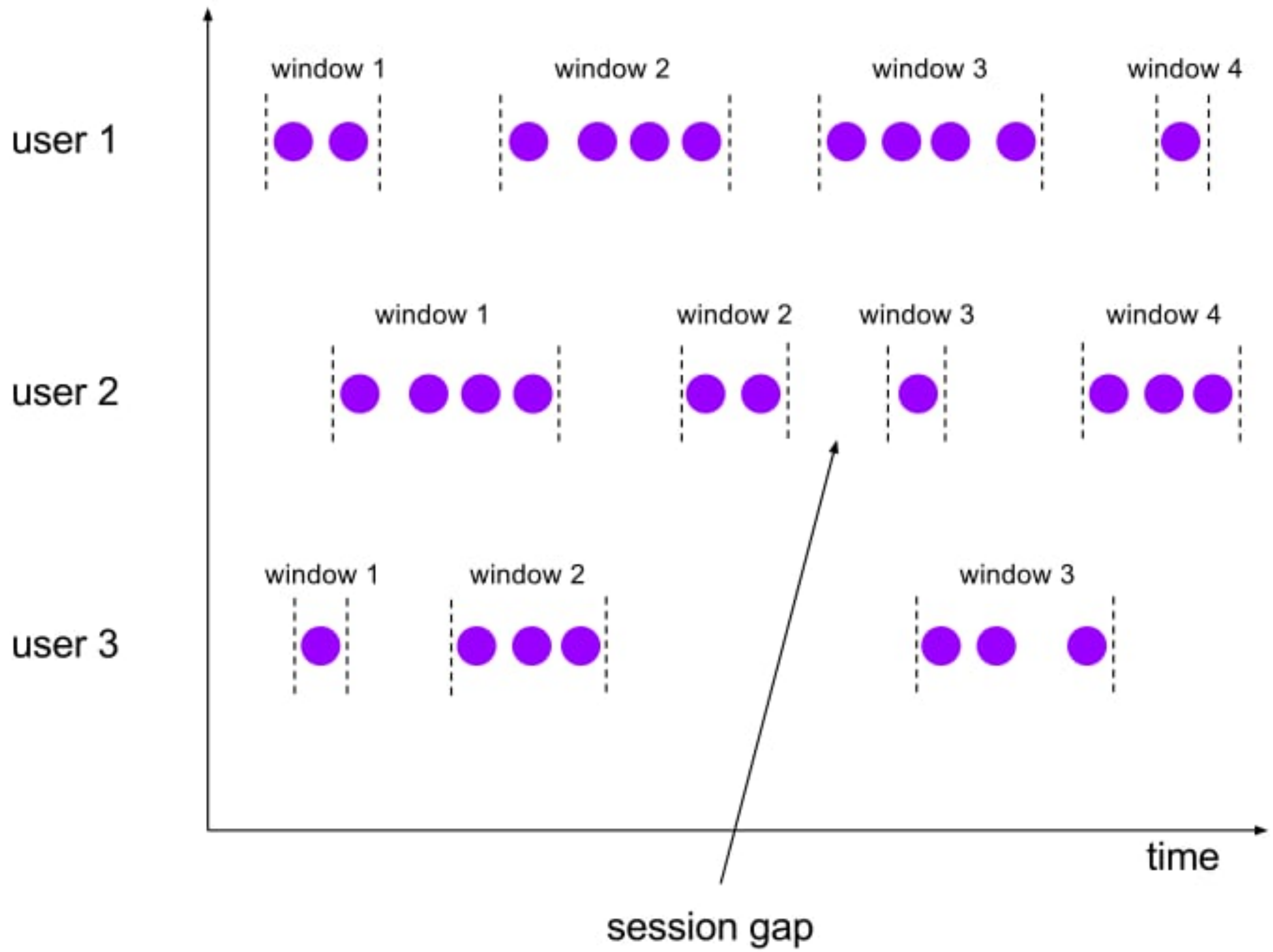
Tumbling:
aligned, fixed length,
non-overlapping windows



Sliding:
aligned, fixed length,
overlapping windows

Session Windows

Non-aligned, variable length windows.



Specifying Windowing

```
stream
    .keyBy(…)              / keyed vs non-keyed windows
    .window(…)             / "Assigner"
    .trigger(…)            / each Assigner has a default Trigger
    .evictor(…)            / default: no Evictor
    .allowedLateness()     / default: zero
    .reduce/apply()        / window function
```

Predefined Keyed Windows

- Tumbling time window
  ```
  .timeWindow(Time.minutes(1))
  ```
- Sliding time window
  ```
  .timeWindow(Time.minutes(1), Time.seconds(10))
  ```
- Tumbling count window
  ```
  .countWindow(100)
  ```
- Sliding count window
  ```
  .countWindow(100, 10)
  ```
- Session window
  ```
  .window(SessionWindows.withGap(Time.minutes(30)))
  ```

Non-keyed Windows

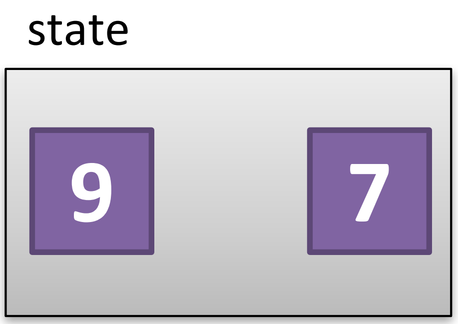
- Windows on non-keyed streams are not processed in parallel!
  - `stream.windowAll(…)…`
  - `stream.timeWindowAll(Time.seconds(10))…`
  - `stream.countWindowAll(20, 10)…`

Aggregations on Windowed Streams

```
DataStream<SensorReading> input = …

input
  .keyBy("key")
  .timeWindow(Time.minutes(1))
  .apply(new MyWastefulMax());

public static class MyWastefulMax implements WindowFunction<
    SensorReading,                    // input type
    Tuple3<String, Long, Integer>,    // output type
    Tuple,                            // key type
    TimeWindow> {                     // window type

    @Override
    public void apply(
        Tuple key,
        TimeWindow window,
        Iterable<SensorReading> events,
        Collector<Tuple3<String, Long, Integer>> out) {

        int max = 0;
        for (SensorReading e : events) {
            if (e.f1 > max) max = e.f1;
        }
        out.collect(new Tuple3<>(Tuple1<String>key).f0, window.getEnd(), max));
    }
}
```

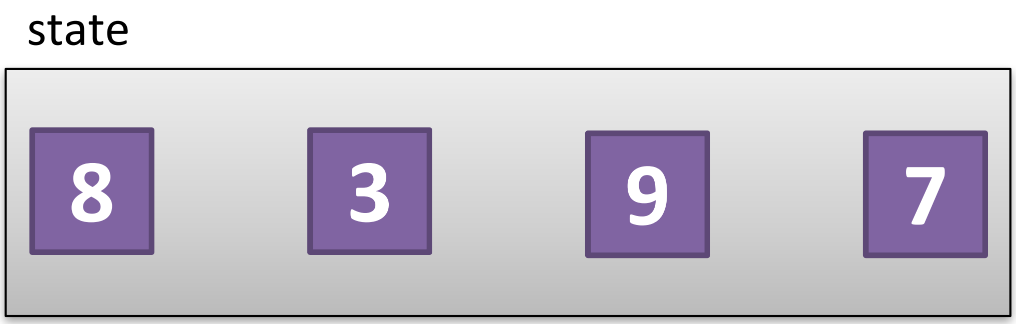
Window State during Aggregation

Window State during Aggregation

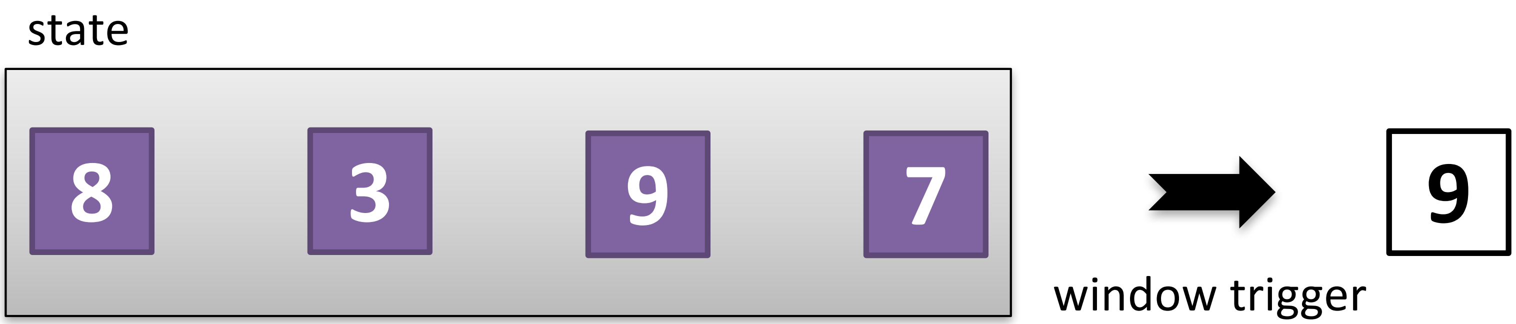


Window State during Aggregation

Window State during Aggregation



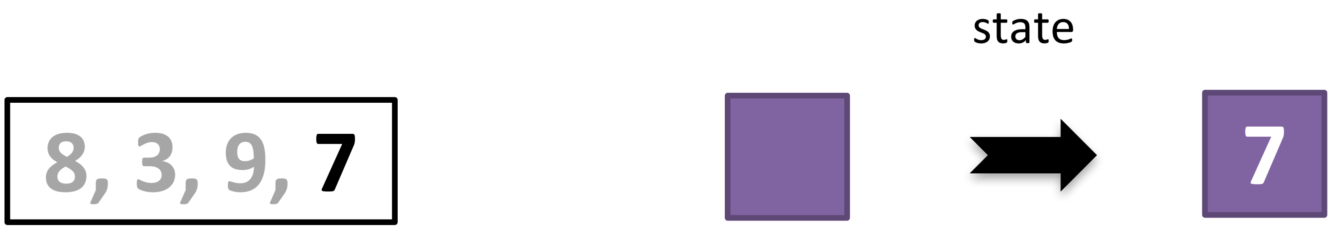
Window State during Aggregation



Incremental Window Aggregation

```
DataStream<SensorReading> input = …

input
  .keyBy(“key”)
  .timeWindow(Time.minutes(1))
  .reduce(new MyReducingMax(), new MyWindowFunction());

private static class MyReducingMax implements ReduceFunction<SensorReading> {
  public SensorReading reduce(SensorReading r1, SensorReading r2) {
      return r1.value() > r2.value() ? r1 : r2;
  }
}

private static class MyWindowFunction implements WindowFunction<
  SensorReading, Tuple2<Long, SensorReading>, String, TimeWindow> {
      public void apply(String key,
                        TimeWindow window,
                        Iterable<SensorReading> maxReadings,
                        Collector<Tuple2<Long, SensorReading>> out) {
          SensorReading max= maxReadings.iterator().next();
          out.collect(new Tuple2<Long, SensorReading>(window.getStart(), max));
      }
}
```

Incremental Aggregation

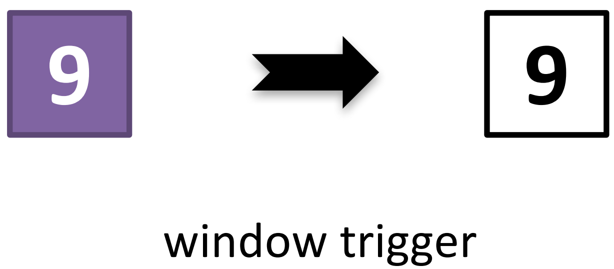


Incremental Aggregation

Incremental Aggregation

Incremental Aggregation

Incremental Aggregation



Operations on Windowed Streams

- `reduce(reduceFunction)`
  - Apply a functional reduce function to the window

- ~~`fold(initialVal, foldFunction)`~~
  - ~~Apply a functional fold function with a specified initial value to the window~~

- Aggregation functions
  - `sum()`, `min()`, `max()`, and others

Custom window logic

- The DataStream API allows you to define very custom window logic

- GlobalWindows
  - a flexible, low-level window assignment scheme that can be used to implement custom windowing behaviors
  - only useful if you explicitly specify triggering, otherwise nothing will happen

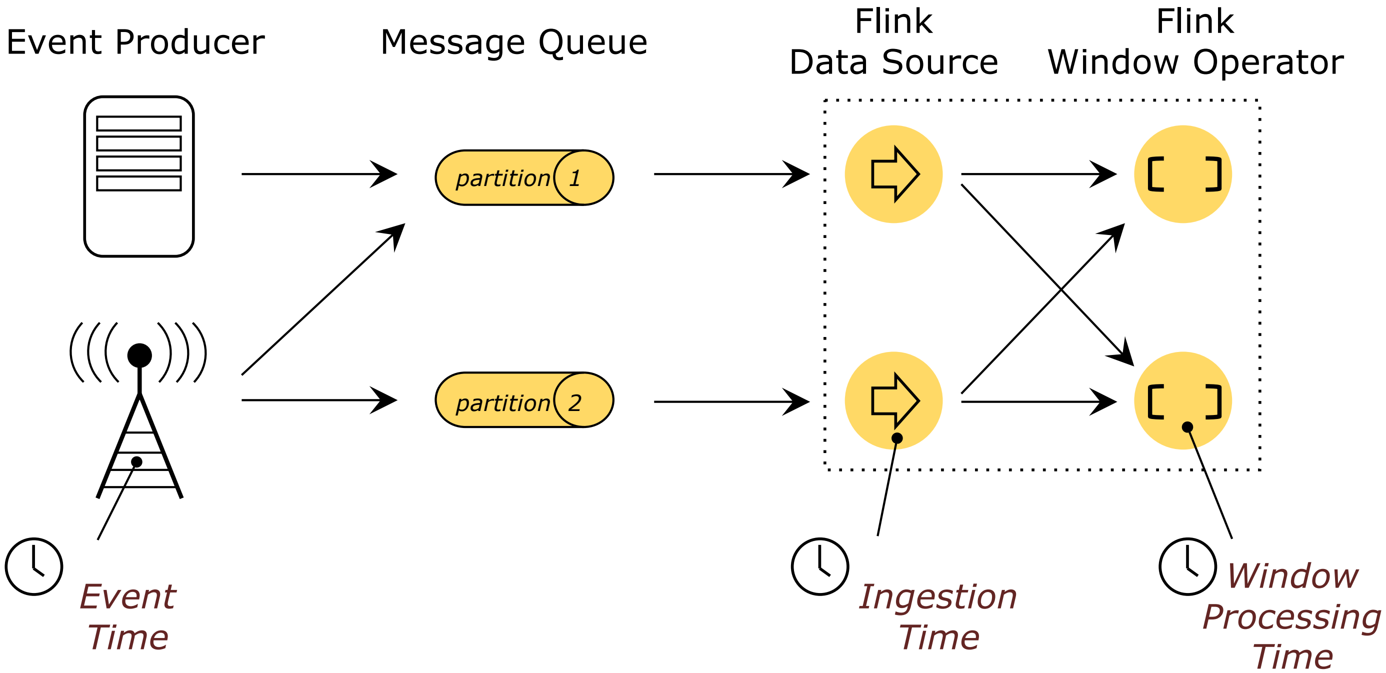
- Trigger
  - defines when to evaluate a window
  - whether to purge the window or not

- *Careful!* This part of the API requires a good understanding of the windowing mechanism!

Handling Time Explicitly

The **biggest change** in moving from batch to streaming is **handling time explicitly**

Different Notions of Time



Event Time vs Processing Time

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*

Setting the StreamTimeCharacteristic

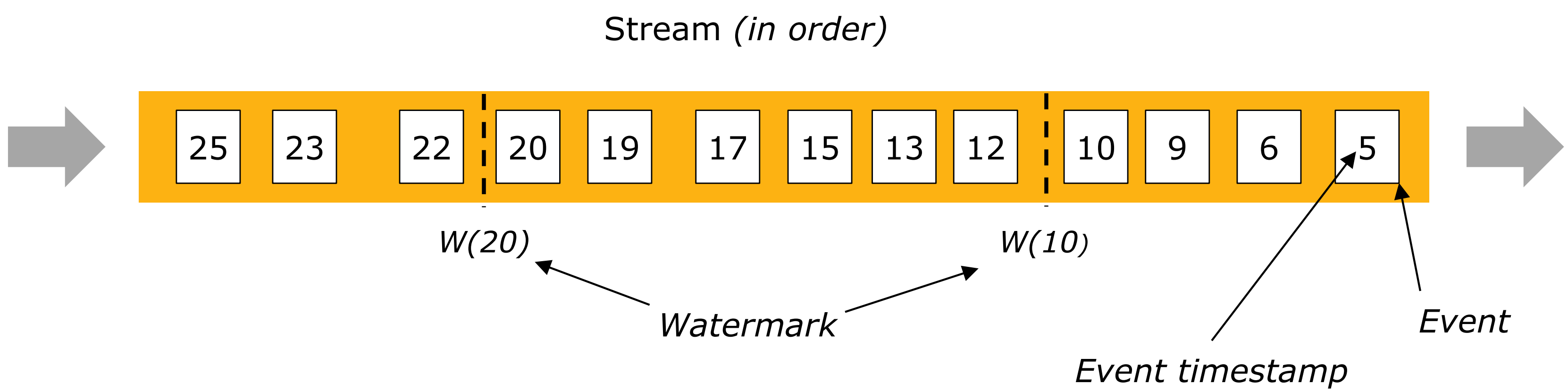
```java
final StreamExecutionEnvironment env =
  StreamExecutionEnvironment.getExecutionEnvironment();

env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

// alternatively:
// env.setStreamTimeCharacteristic(TimeCharacteristic.IngestionTime);
// env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
```

Choosing Event Time has Consequences

- With event time, Flink needs to know
  - how to extract timestamps from stream elements
  - when enough event time has elapsed that a time window should be triggered

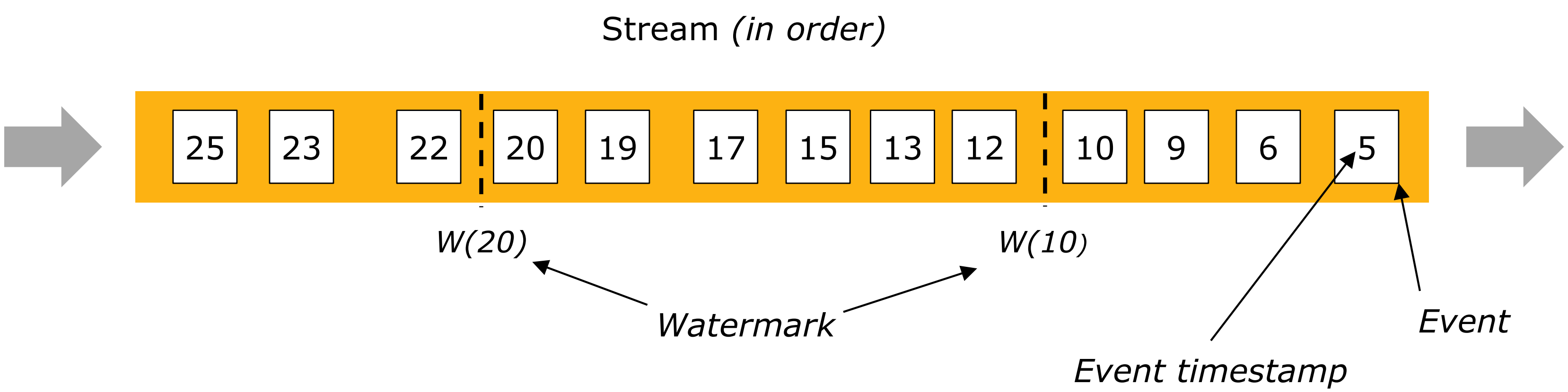
Watermarks

- Watermarks mark the progress of event time
- They flow with the data stream and carry a timestamp
- *Watermarks assert that all earlier events have (probably) arrived*



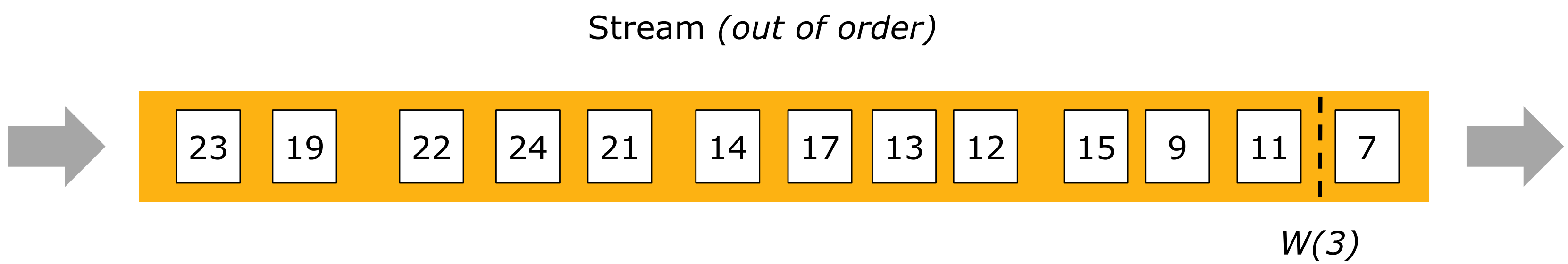
Perfect Watermarks

- When stream elements are in order (or in order by key), we can achieve perfect watermarking



Bounded out-of-orderness

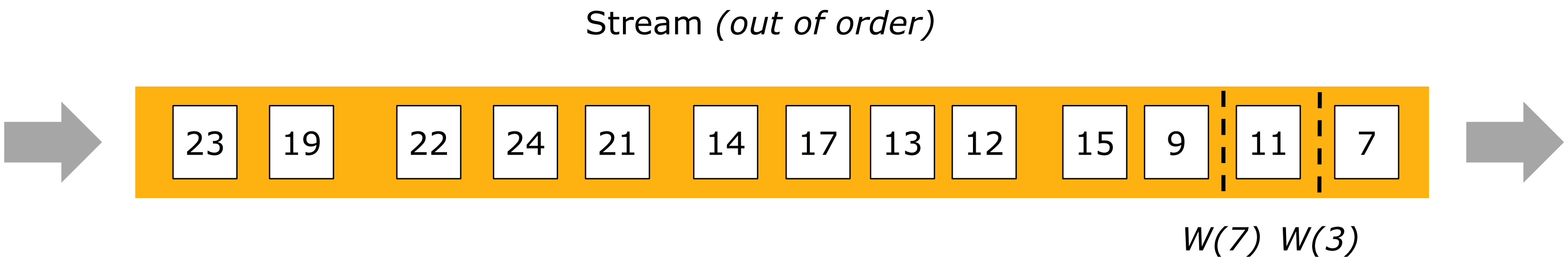
- When events are out-of-order, we often assume there is some bound to how out-of-order they can be



maxOutOfOrderness = 4

Bounded out-of-orderness

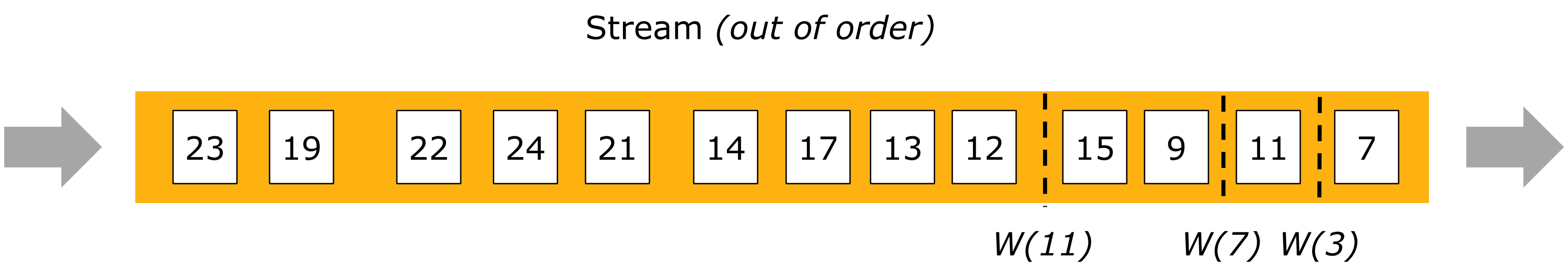
- Each time a new maximum timestamp arrives, we have enough info to emit a new Watermark



maxOutOfOrderness = 4

Bounded out-of-orderness

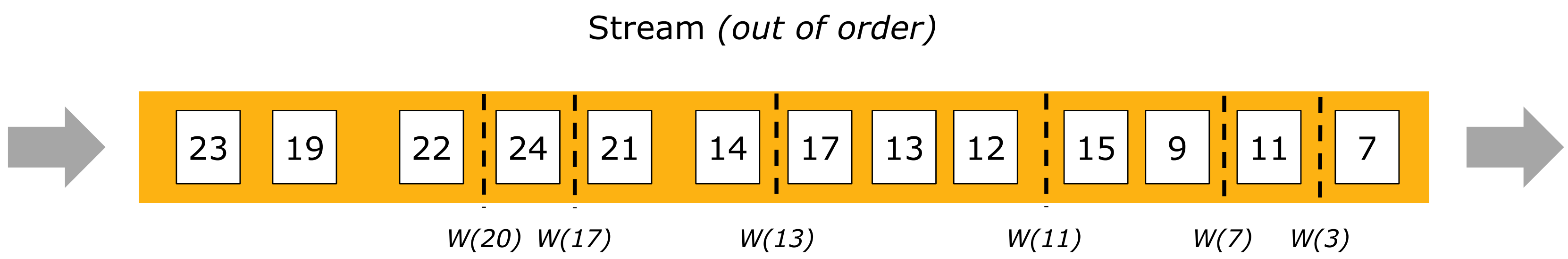
- Each time a new maximum timestamp arrives, we have enough info to emit a new Watermark



maxOutOfOrderness = 4

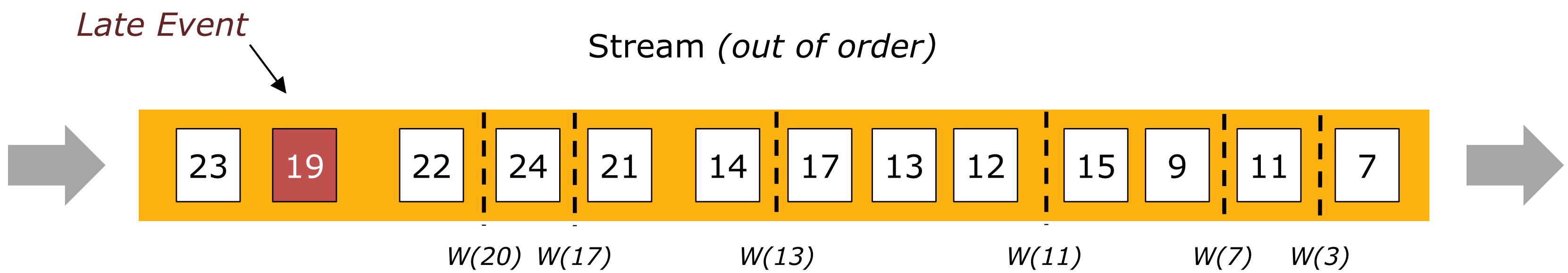
Bounded out-of-orderness

- Each time a new maximum timestamp arrives, we have enough info to emit a new Watermark

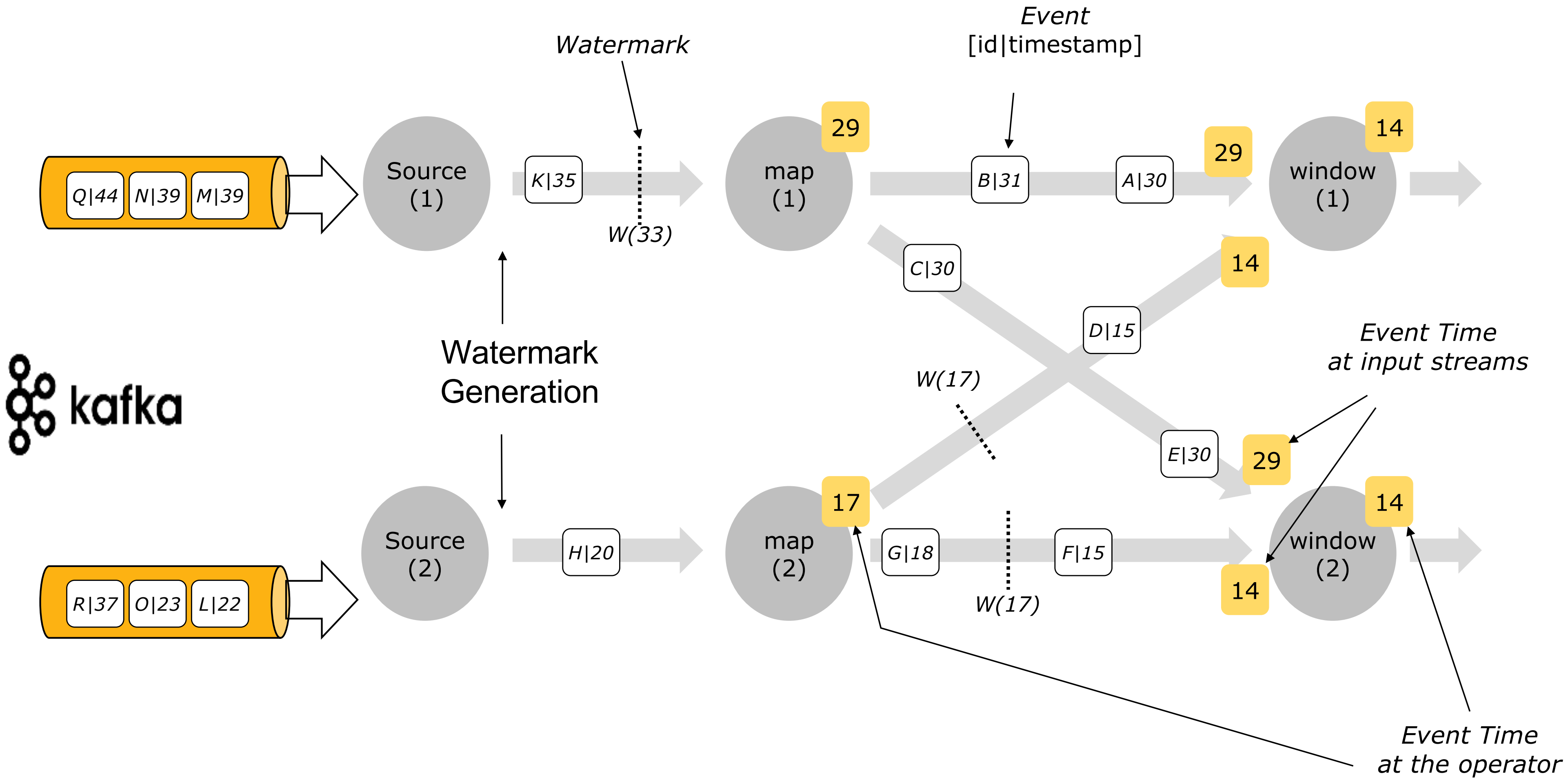


Watermarks define Lateness

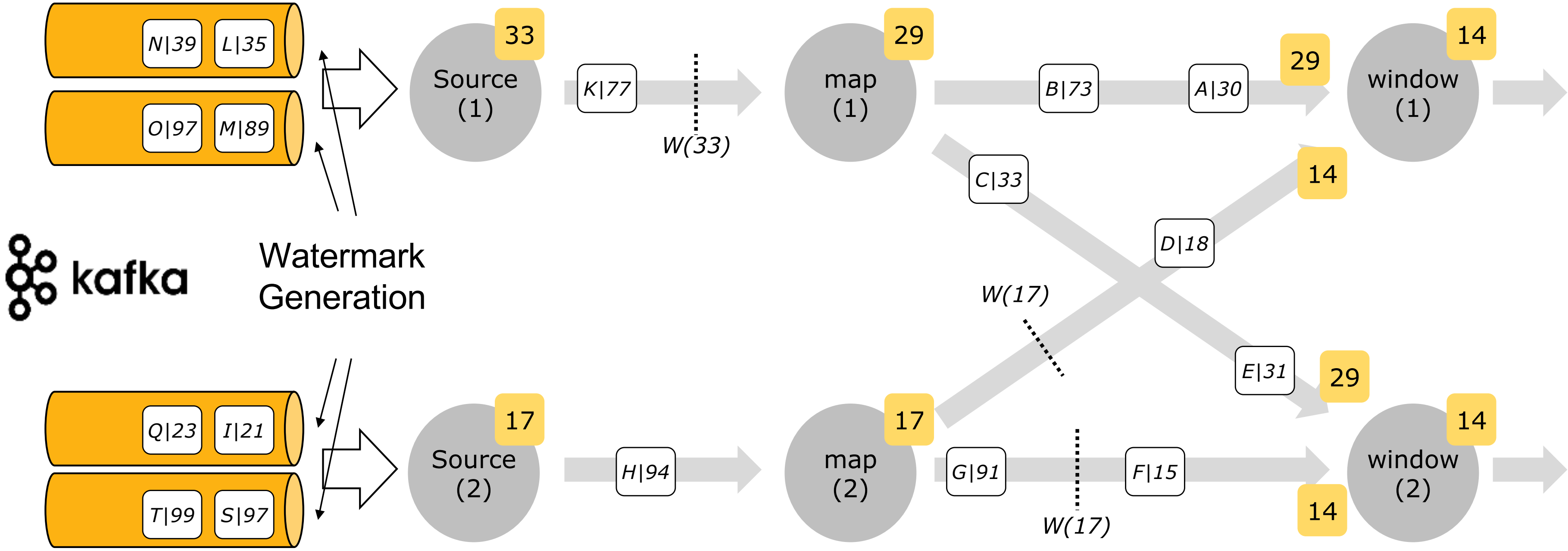
- Elements where *timestamp* < *currentWatermark* are late



Watermarks in Parallel



Per-Kafka-Partition Watermarks



Watermarking

- Perfect
- (Un)comfortably bounded by fixed delay
  - too slow: results are delayed
  - too fast: some data is late
- Heuristic
  - allow windows to produce results as soon as meaningfully possible, and then continue with updates during the allowed lateness interval

Pre-defined timestamp extractors / watermark emitters

- AscendingTimestampExtractor
  - For special case when timestamps are in ascending order
- BoundedOutOfOrdernessTimestampExtractor
  - Periodically emits watermarks that lag a fixed amount of time behind the max timestamp seen so far

Example

```
stream
    .assignTimestampsAndWatermarks(new MyTSExtractor())
    .keyBy(...)
    .timeWindow(...)
    .addSink(...);

public static class MyTSExtractor extends
  BoundedOutOfOrdernessTimestampExtractor<TaxiRide> {

    public TaxiRideTSExtractor() {
        super(Time.seconds(MAX_EVENT_DELAY));
    }

    @Override
    public long extractTimestamp(TaxiRide ride) {
        return ride.startTime.getMillis();
    }
}
```

References

- *The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing*
  https://research.google.com/pubs/pub43864.html

- Documentation
  - https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/event_time.html
  - https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/event_timestamps_watermarks.html
  - https://ci.apache.org/projects/flink/flink-docs-release-1.2/dev/windows.html

- Blog posts
  - http://flink.apache.org/news/2015/12/04/Introducing-windows.html
  - http://data-artisans.com/how-apache-flink-enables-new-streaming-applications-part-1/
  - https://www.mapr.com/blog/essential-guide-streaming-first-processing-apache-flink
  - http://data-artisans.com/session-windowing-in-flink/