DataStream API

Windows & Time

Apache Flink® Training

dataArtisans

Flink v1.2.0 – 27.02.2017

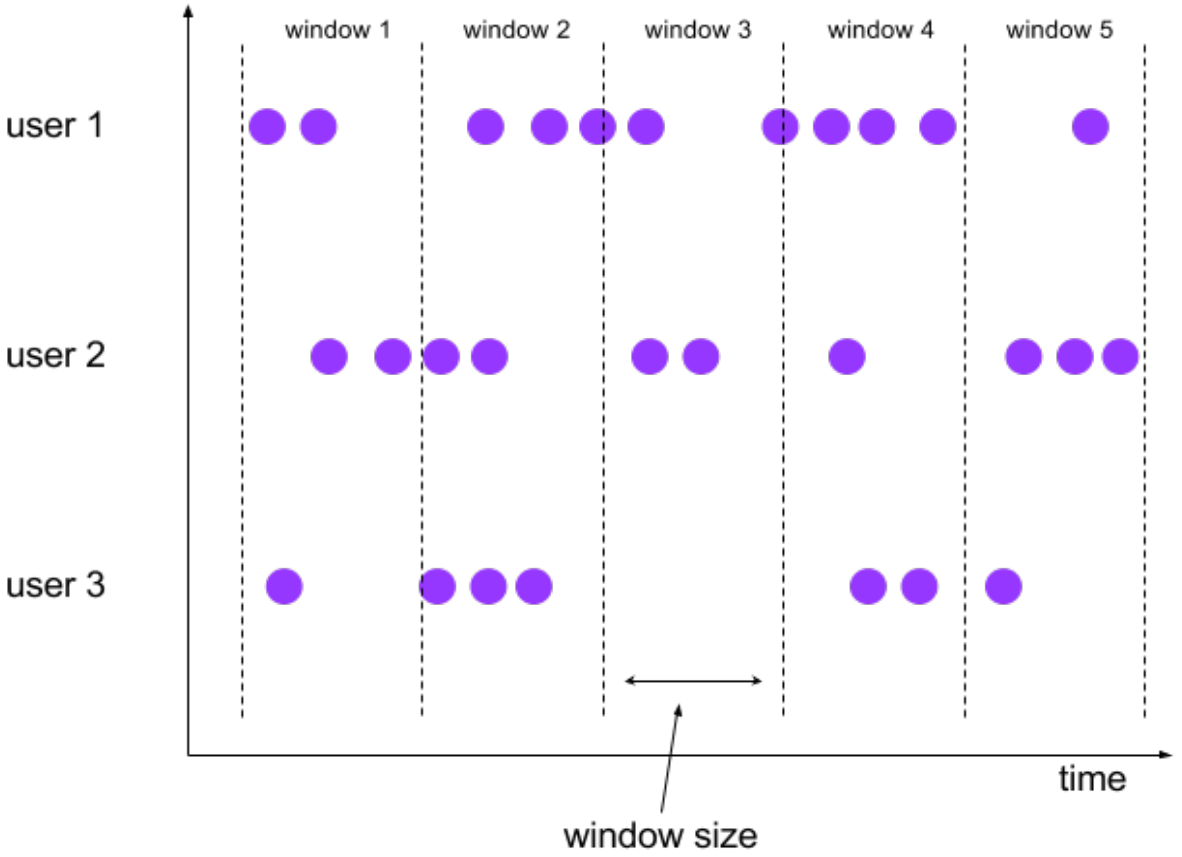
Windows and Aggregates

Windows

- Aggregations on DataStreams are different from aggregations on DataSets
  - You cannot count all records of an infinite stream
- DataStream aggregations make sense on windowed streams
  - A finite subset of stream elements

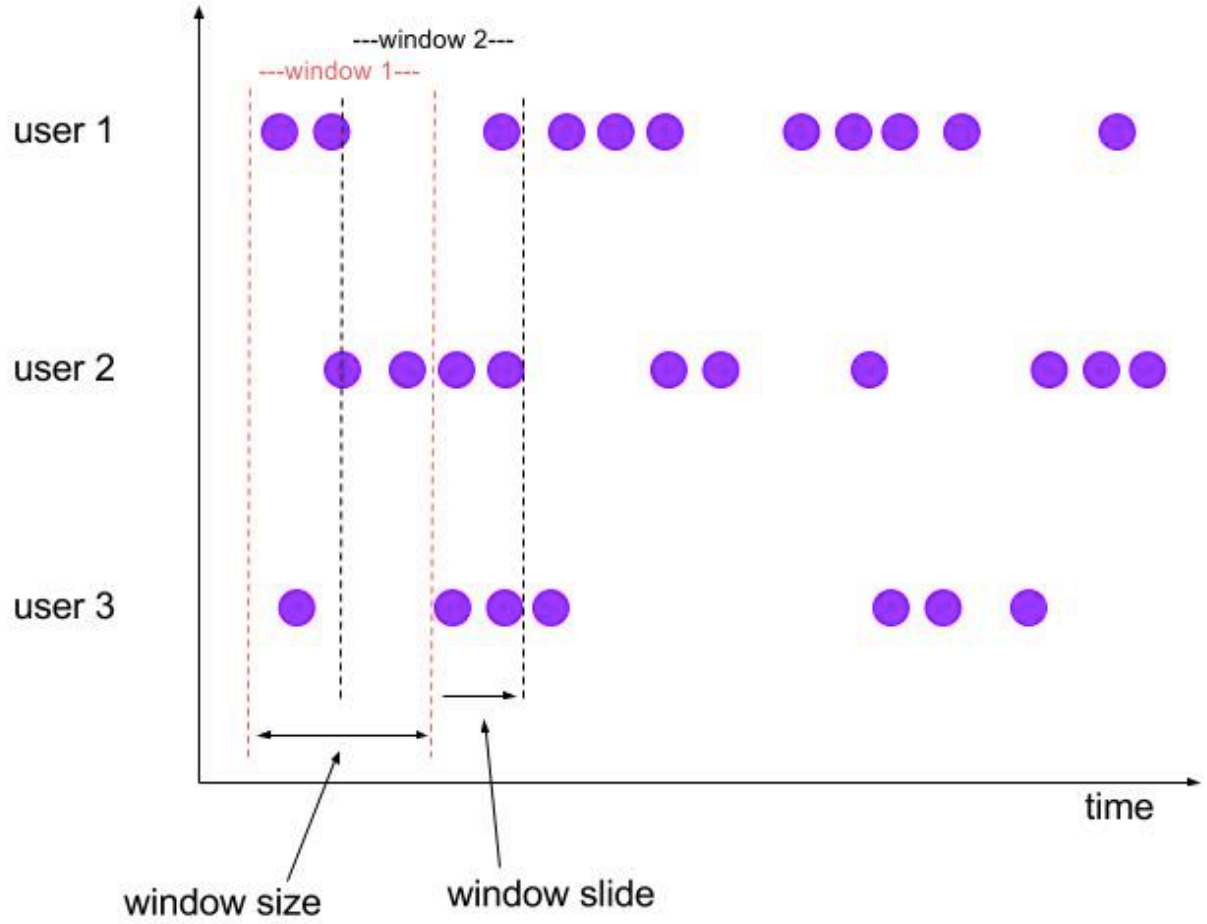
Tumbling Windows

Aligned, fixed length, non-overlapping windows.



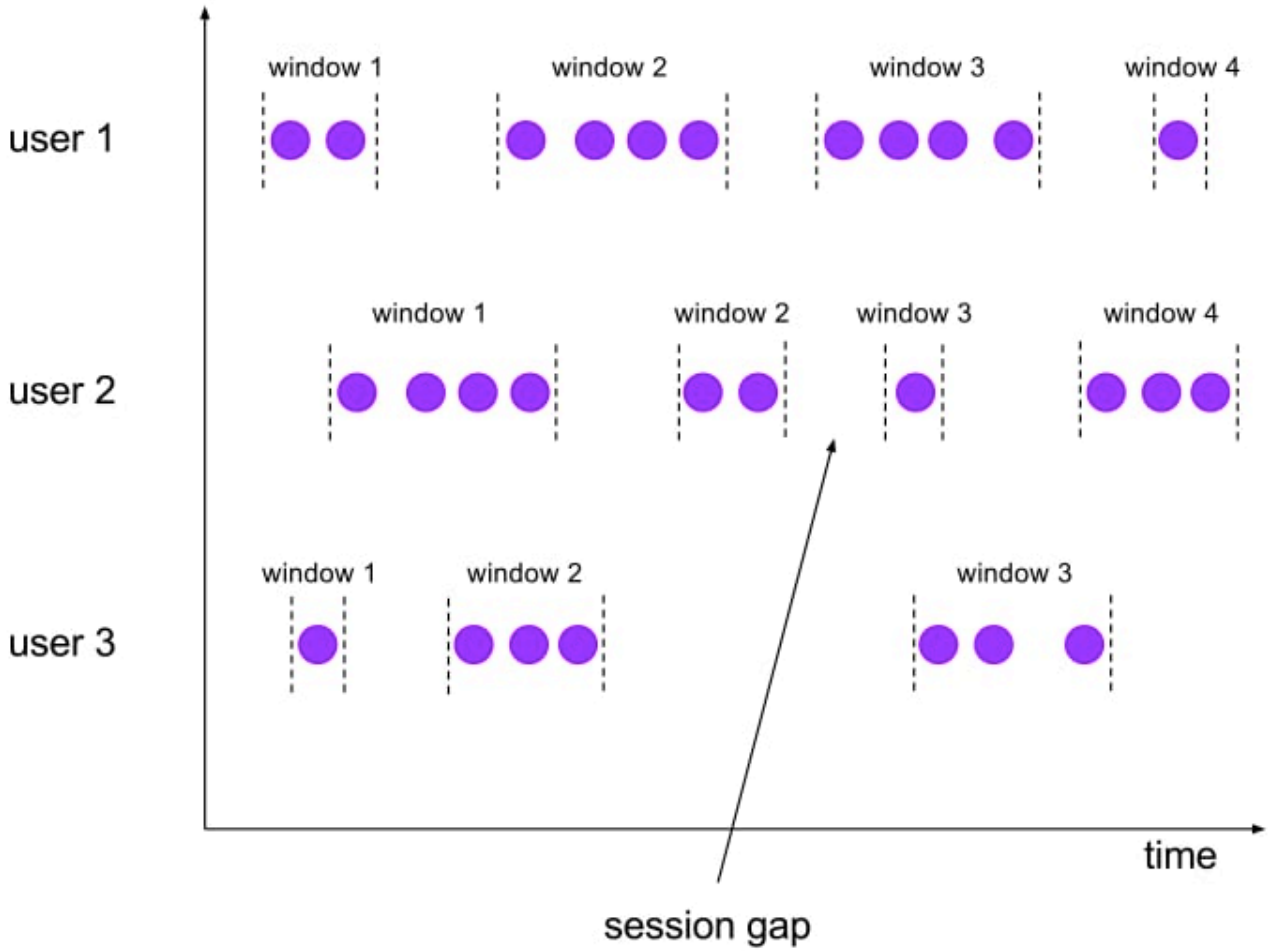
Sliding Windows

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()        / the 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

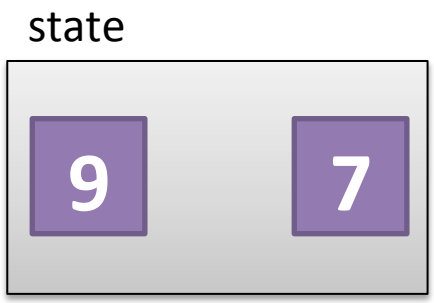
```
DataStream<SensorReading> input = …

input
  .keyBy(“key”)
  .timeWindow(Time.minutes(1))
  .apply(new MyWastefulFunction());
```

Aggregation with a WindowFunction

```
public static class MyWastefulFunction 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));
    }
}
```

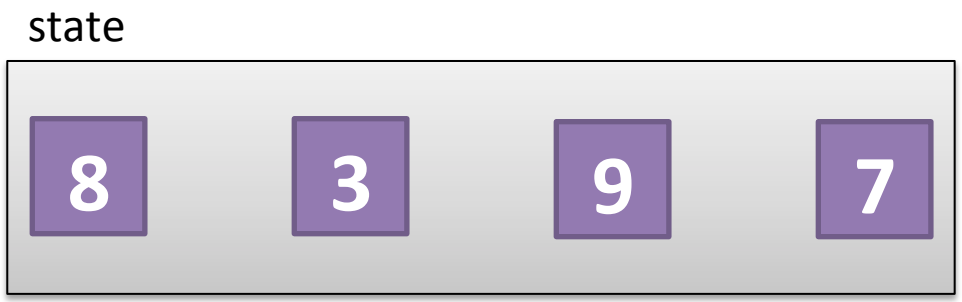
Window State during Aggregation

Window State during Aggregation

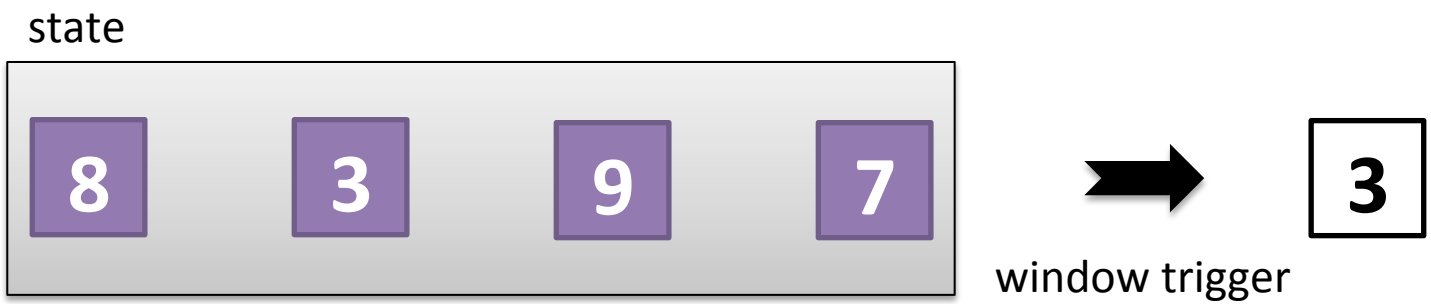


Window State during Aggregation

Window State during Aggregation



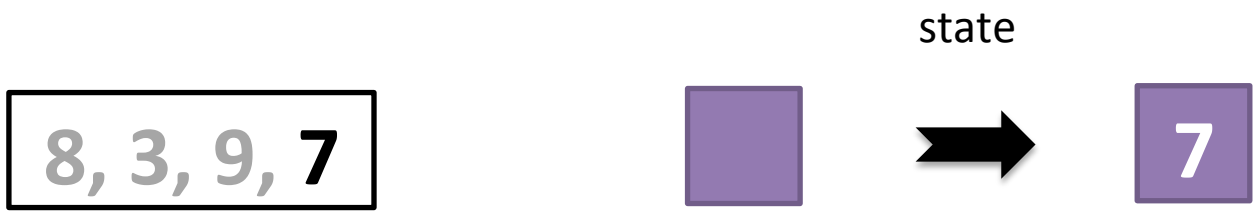
Window State during Aggregation



Incremental Window Aggregation

```
DataStream<SensorReading> input = …

Input
  .keyBy(<key selector>)
  .timeWindow(<window assigner>)
  .reduce(new MyReduceFunction(), new MyWindowFunction());

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

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

Incremental Aggregation

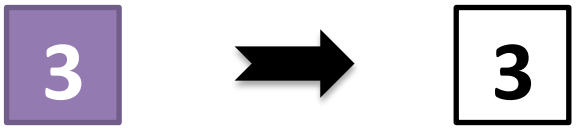


Incremental Aggregation

Incremental Aggregation

Incremental Aggregation

Incremental Aggregation



window trigger

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~~

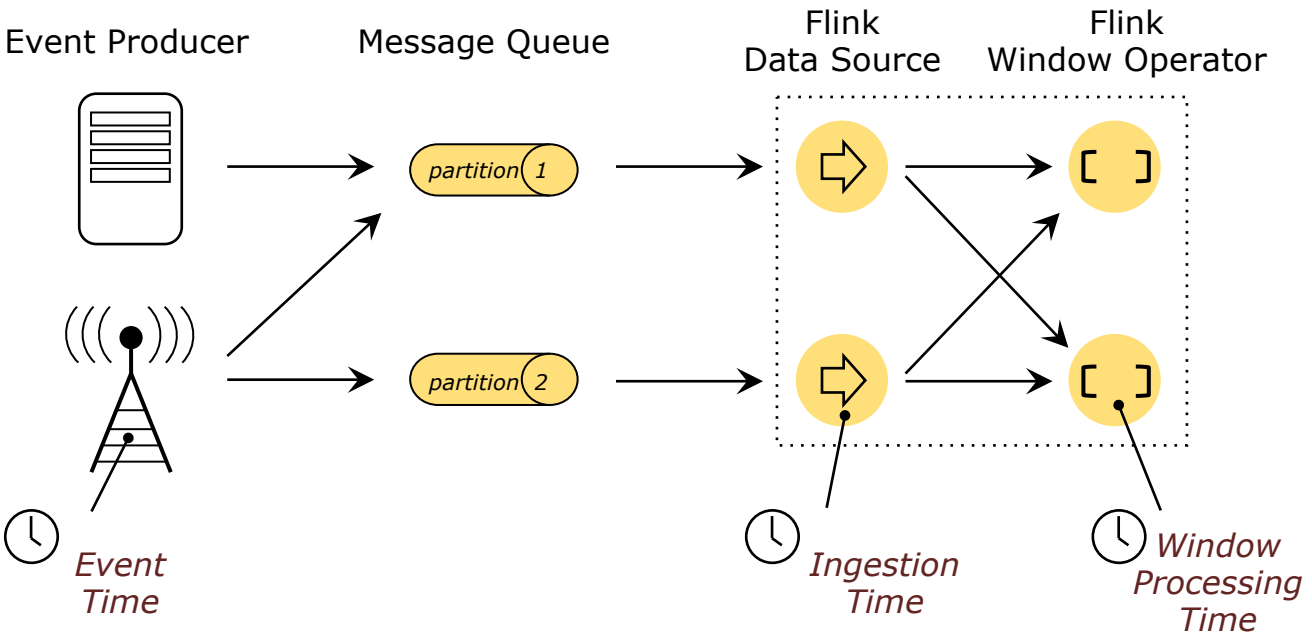
- Aggregation functions
  - `sum()`, `min()`, `max()`, and others

Custom window logic

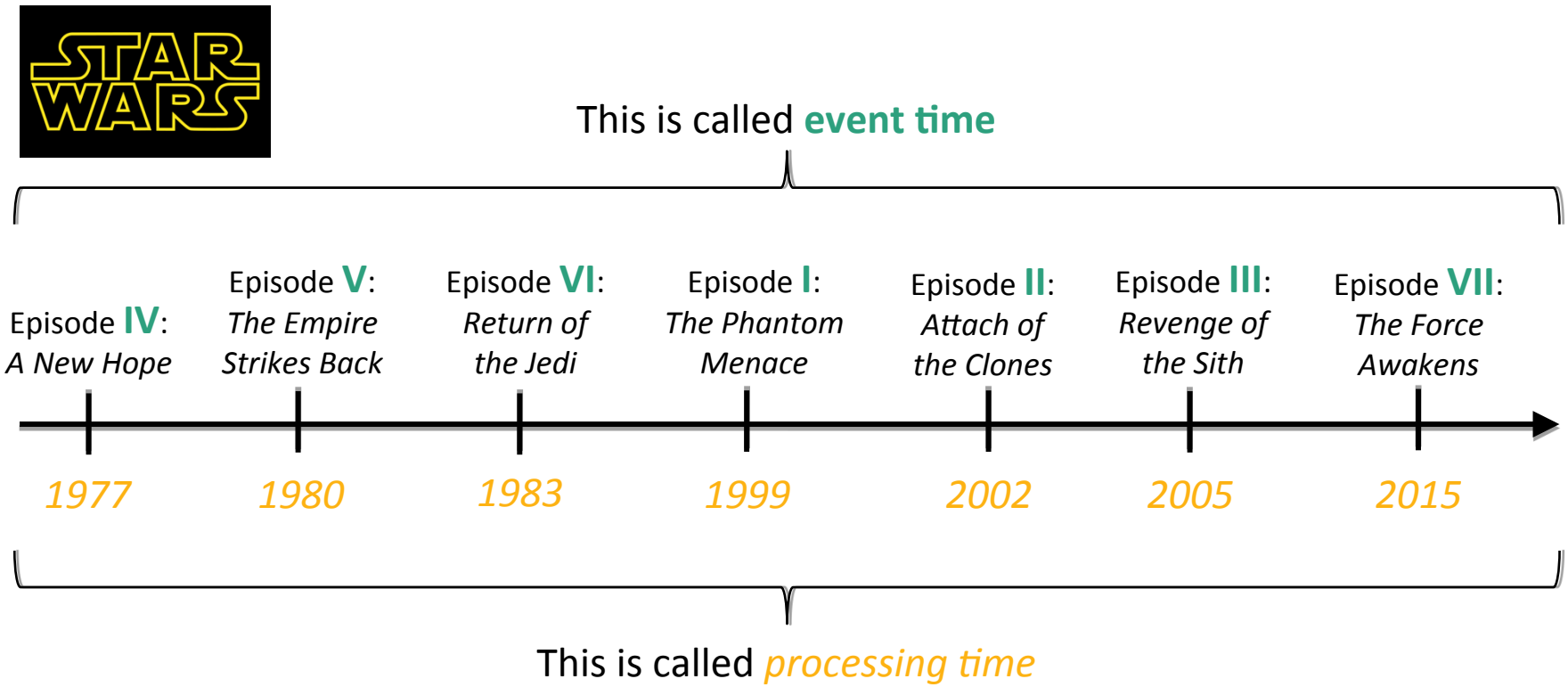
- The DataStream API allows you to define very custom window logic
- GlobalWindows
  - a flexible, low-level window assignment scheme that can be used to implement custom windowing behaviors
  - only useful if you explicitly specify triggering, otherwise nothing will happen
- Trigger
  - defines when 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

Different Notions of Time



Event Time vs Processing Time



Setting the StreamTimeCharacteristic

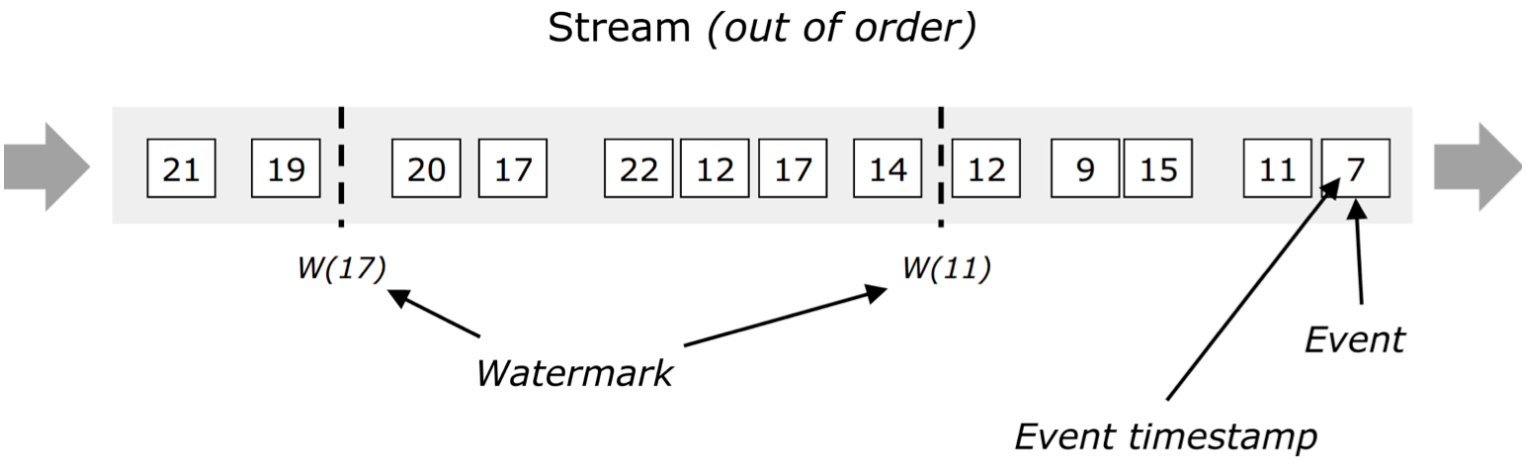
```
final StreamExecutionEnvironment env =
  StreamExecutionEnvironment.getExecutionEnvironment();

env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

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

Choosing Event Time has Consequences

- When working with event time, Flink needs to know
  - how to extract the timestamp from a stream element
  - 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; they are crucial for handling out-of-order events
- A *Watermark(t)* is a declaration that all events with a *timestamp < t* have occurred



Timestamp Assigners / Watermark Generators

```
DataStream<MyEvent> stream = …

DataStream<MyEvent> withTimestampsAndWatermarks = stream
    .assignTimestampsAndWatermarks(new MyTSExtractor());

withTimestampsAndWatermarks
    .keyBy(...)
    .timeWindow(...)
    .addSink(...);
```

Timestamp Assigners / Watermark Generators

- There are different types of timestamp extractors
- BoundedOutOfOrdernessTimestampExtractor
  - Periodically emits watermarks that lag a fixed amount of time behind the max timestamp seen so far
  - To use, subclass and implement

```
public abstract long extractTimestamp(T element)
```

  - Constructor

```
public BoundedOutOfOrdernessTimestampExtractor(
   Time maxOutOfOrderness)
```

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/