DataStream API

Windows & Time

Apache Flink® Training

dataArtisans

Flink v1.1.2 – 14.09.2016

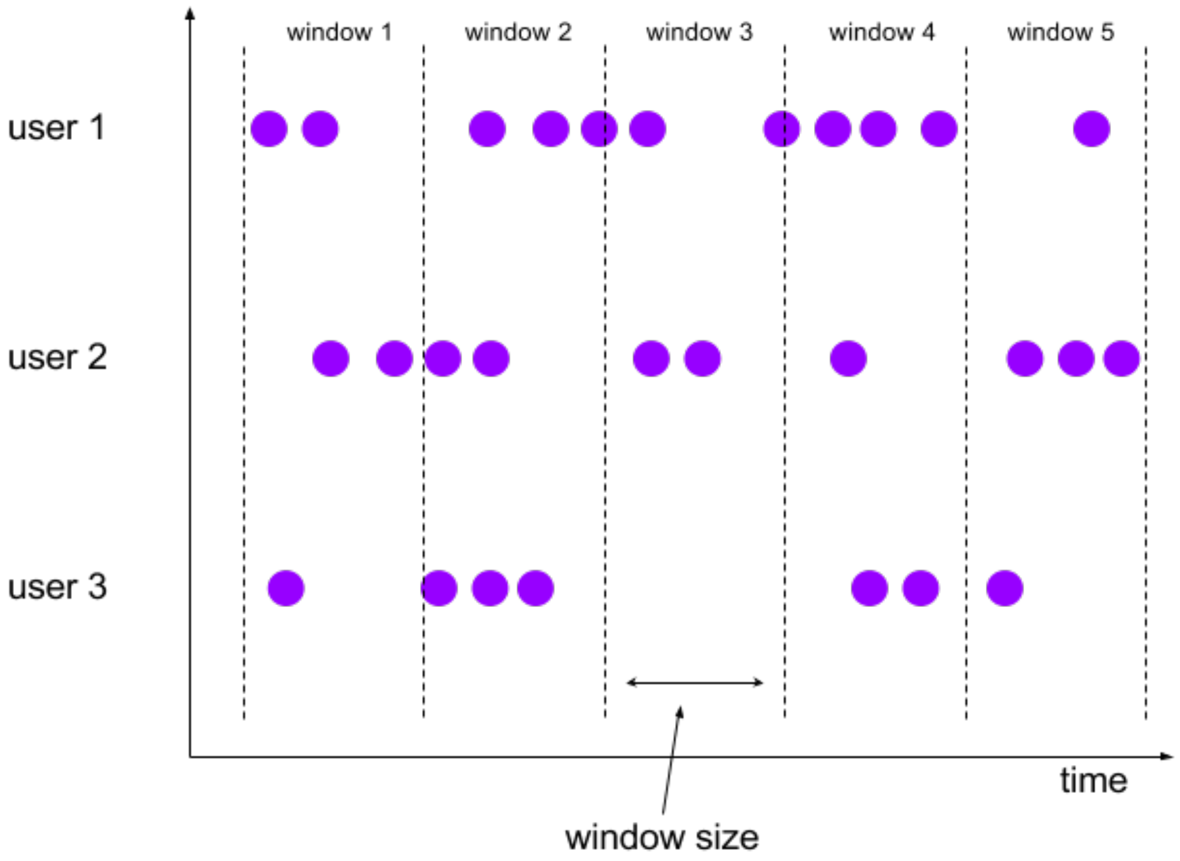
Windows and Aggregates

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

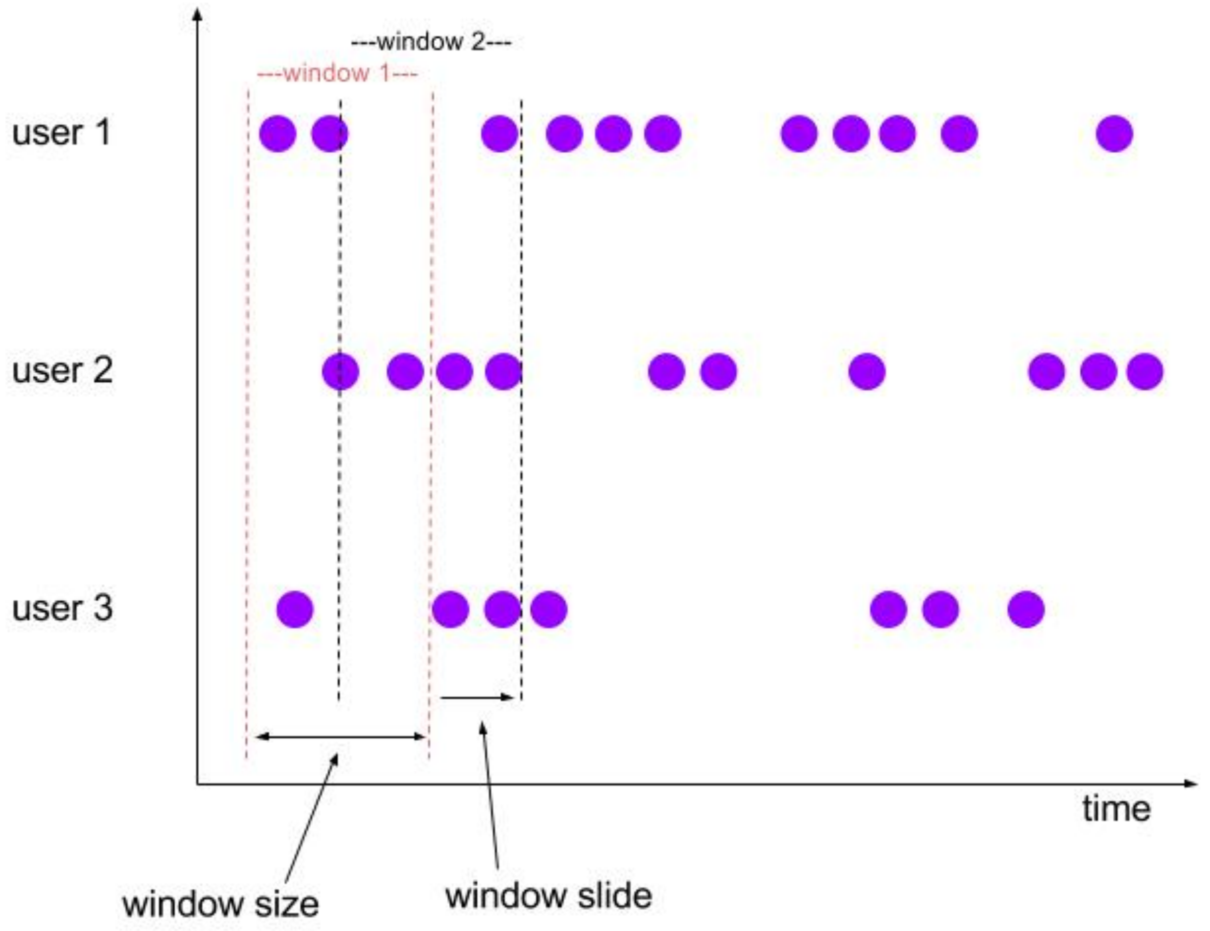
Tumbling Windows

Aligned, fixed length, non-overlapping windows.



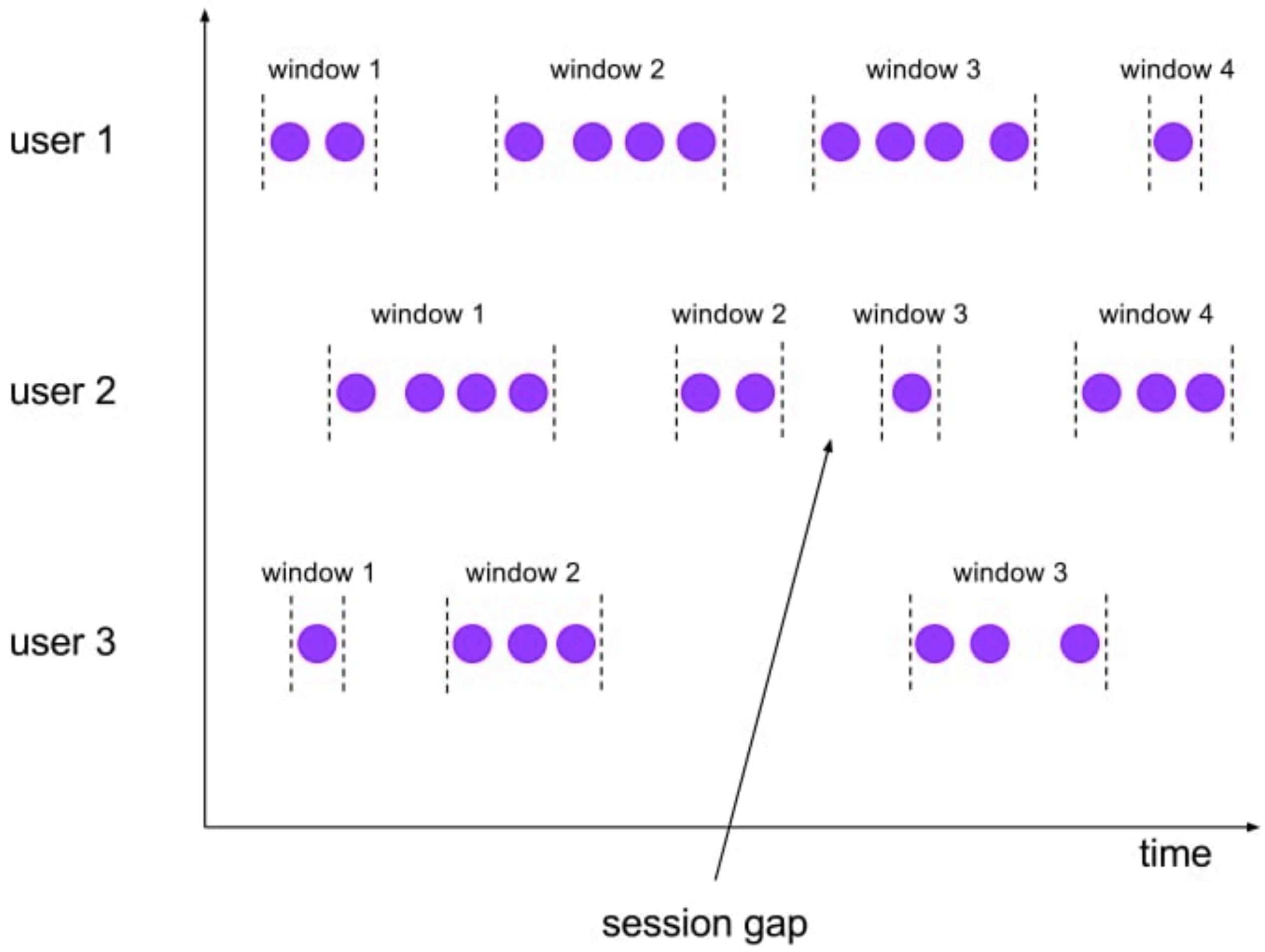
Sliding Windows

Aligned, fixed length, overlapping windows.



Session Windows

Non-aligned, variable length windows.



Specifying Windowing

```
// (name, age) of passengers
DataStream<Tuple2<String, Integer>> passengers = …

passengers
    // group by second field (age)
    .keyBy(1)
    // window definition: tumbling window of 1 minute
    .timeWindow(Time.minutes(1))
```

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)))
```

Predefined Non-keyed Windows

- Windows on non-keyed streams are not processed in parallel!

- TimeWindow (tumbling, 10 seconds)

```
.timeWindowAll(Time.seconds(10))
```

- CountWindow (sliding, 20/10)

```
.countWindowAll(20, 10)
```

Aggregations on Windowed Streams

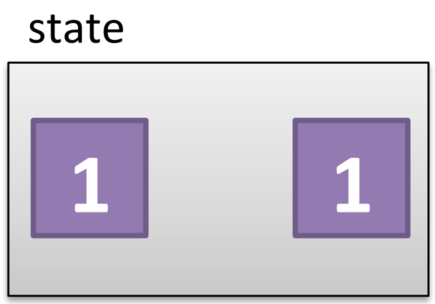
```
// (name, age) of passengers
DataStream<Tuple2<String, Integer>> passengers = …

passengers
    // group by second field (age)
    .keyBy(1)
    // windows that are 1 minute long
    .timeWindow(Time.minutes(1))
    // apply a custom window function on window data
    .apply(new CountByAge());
```

Aggregation with a WindowFunction

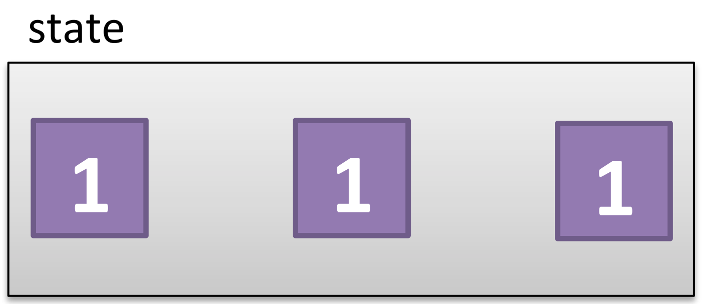
```
public static class CountByAge implements WindowFunction<
     Tuple2<String, Integer>,         // input type
     Tuple3<Integer, Long, Integer>, // output type
     Tuple,                           // key type
     TimeWindow> {                    // window type

     @Override
     public void apply(
          Tuple key,
          TimeWindow window,
          Iterable<Tuple2<String, Integer>> persons,
          Collector<Tuple3<Integer, Long, Integer>> out) {

          int age = ((Tuple1<Integer>)key).f0;
          int cnt = 0;

          for (Tuple2<String, Integer> p : persons) {
               cnt++;
          }
          out.collect(new Tuple3<>(age, window.getEnd(), cnt));
     }
}
```

Window State during Aggregation

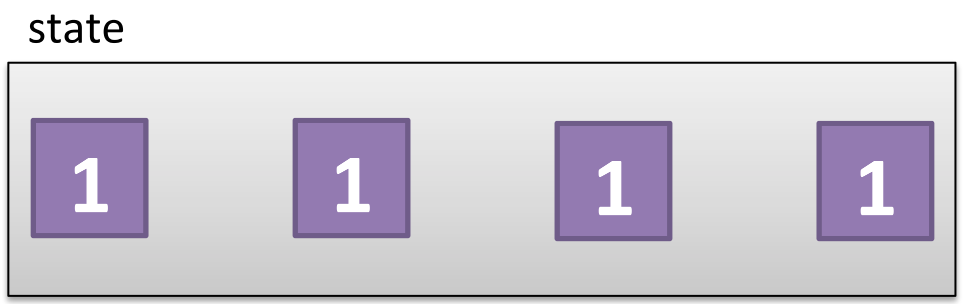
Window State during Aggregation



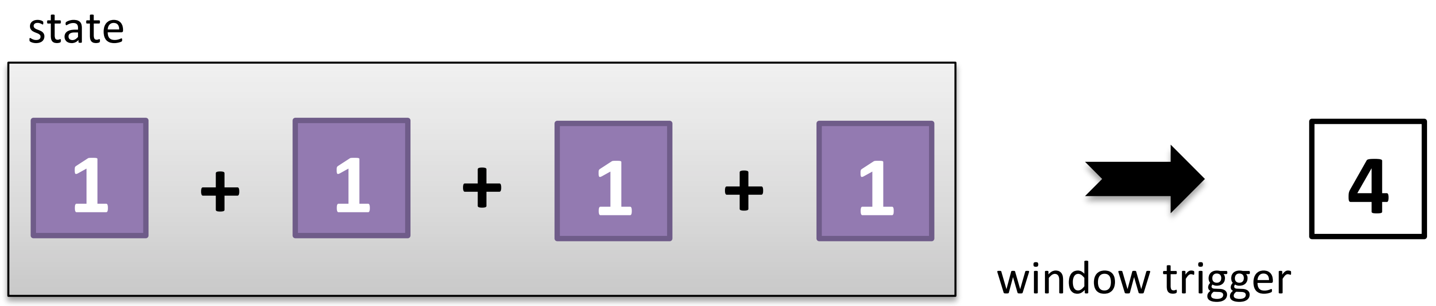
Window State during Aggregation



Window State during Aggregation



Window State during Aggregation

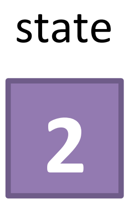


Operations on Windowed Streams

- `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

Incremental Aggregation

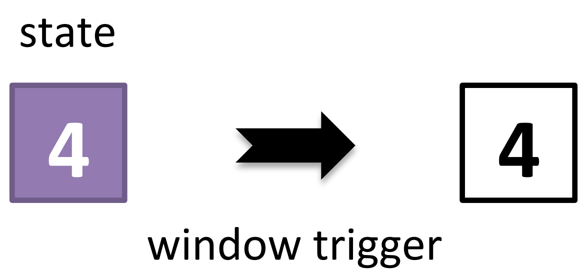
Incremental Aggregation



Incremental Aggregation

Incremental Aggregation

Incremental Aggregation



Incremental Window Aggregation

```
DataStream<Tuple2<String, Integer>> passengers = …

passengers
  .keyBy(1)
  .timeWindow(Time.minutes(1), Time.seconds(10))
  .apply(new Tuple3<Integer, Long, Integer>(0, 0L, 0), new MyFoldFunction(), new MyWindowFunction())

private static class MyFoldFunction
    implements FoldFunction<Tuple2<String, Integer>, Tuple3<Integer, Long, Integer>> {

  public Tuple3<Integer,Long,Integer> fold(Tuple3<Integer, Long, Integer> acc, Tuple2<String, Integer> p) {
      Integer count = acc.getField(2);
      acc.setField(2, count + 1);
      return acc;
  }
}

private static class MyWindowFunction
    implements WindowFunction<Tuple3<Integer,Long,Integer>, Tuple3<Integer,Long,Integer>, Integer, TimeWindow> {

  public void apply(Integer age_key,
                    TimeWindow window,
                    Iterable<Tuple3<Integer,Long,Integer>> counts,
                    Collector<Tuple3<Integer,Long,Integer>> out) {
    Integer count = counts.iterator().next().getField(2);
    out.collect(new Tuple3<Integer,Long,Integer>(age_key, window.getEnd(), count));
  }
}
```

Incremental Window Aggregation

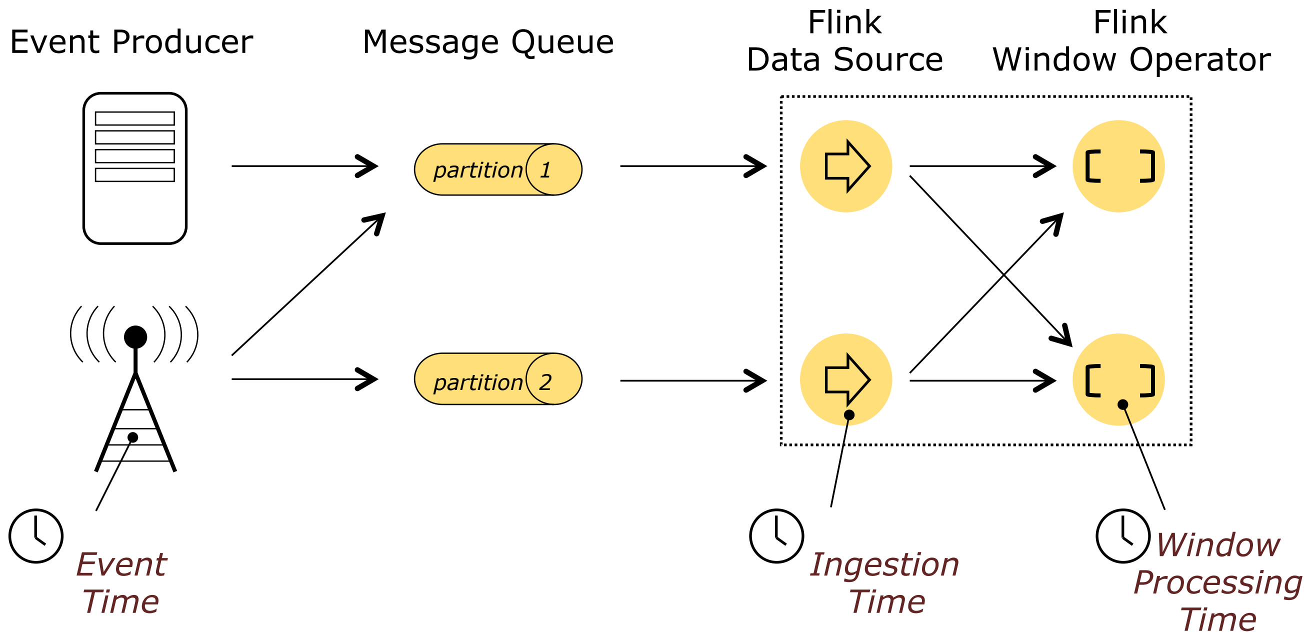
```
DataStream<Tuple2<String, Integer>> passengers = …

passengers
  .keyBy(1)
  .timeWindow(Time.minutes(1), Time.seconds(10))
  .apply(new Tuple3<Integer, Long, Integer>(0, 0L, 0), new MyFoldFunction(), new MyWindowFunction())

private static class MyFoldFunction
    implements FoldFunction<Tuple2<String, Integer>, Tuple3<Integer, Long, Integer>> {

  public Tuple3<Integer,Long,Integer> fold(Tuple3<Integer, Long, Integer> acc, Tuple2<String, Integer> p) {
      Integer count = acc.getField(2);
      acc.setField(2, count + 1);
      return acc;
  }
}

private static class MyWindowFunction
    implements WindowFunction<Tuple3<Integer,Long,Integer>, Tuple3<Integer,Long,Integer>, Integer, TimeWindow> {

  public void apply(Integer age_key,
                    TimeWindow window,
                    Iterable<Tuple3<Integer,Long,Integer>> counts,
                    Collector<Tuple3<Integer,Long,Integer>> out) {
    Integer count = counts.iterator().next().getField(2);
    out.collect(new Tuple3<Integer,Long,Integer>(age_key, window.getEnd(), count));
  }
}
```

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!

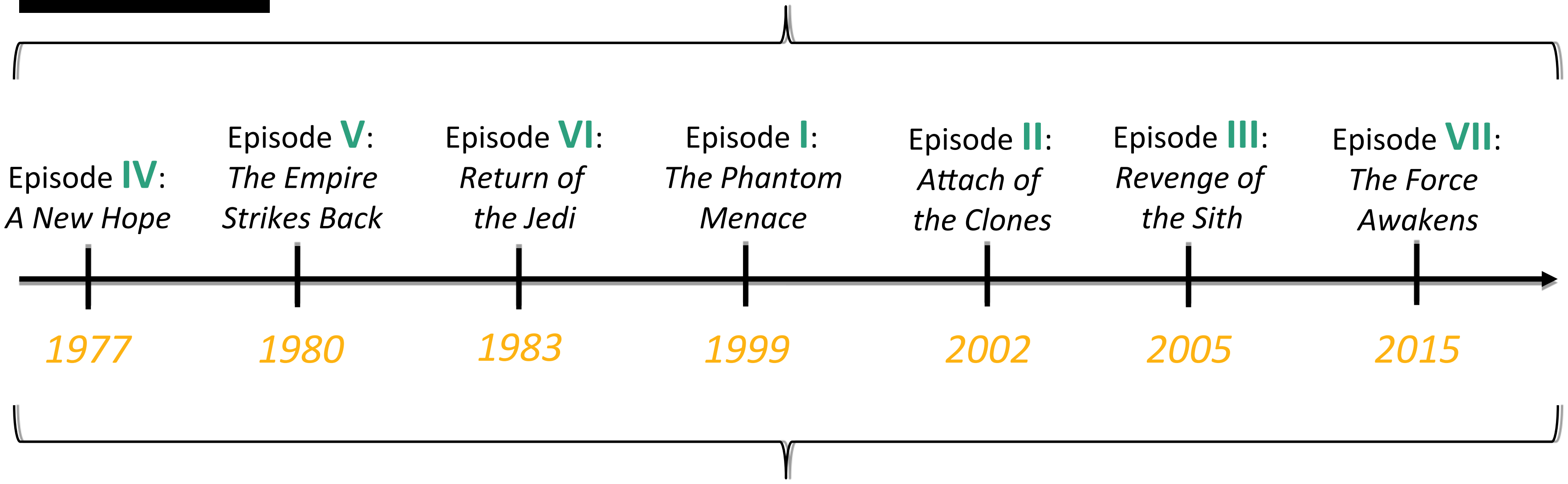
Handling Time Explicitly

Different Notions of Time



Event Time vs Processing Time

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

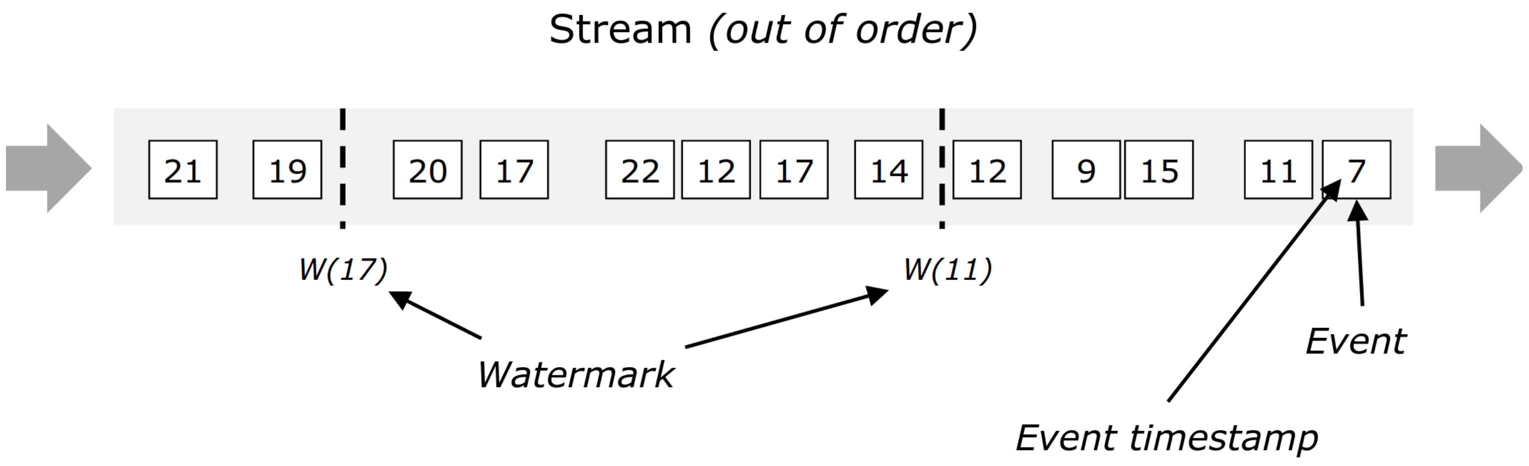
```
final StreamExecutionEnvironment env =
  StreamExecutionEnvironment.getExecutionEnvironment();

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 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

- Documentation
  - https://ci.apache.org/projects/flink/flink-docs-release-1.1/apis/streaming/event_time.html
  - https://ci.apache.org/projects/flink/flink-docs-release-1.1/apis/streaming/event_timestamps_watermarks.html
  - https://ci.apache.org/projects/flink/flink-docs-release-1.1/apis/streaming/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/