
BEAM
SUMMIT

Dealing with order in streams

Israel Herraiz

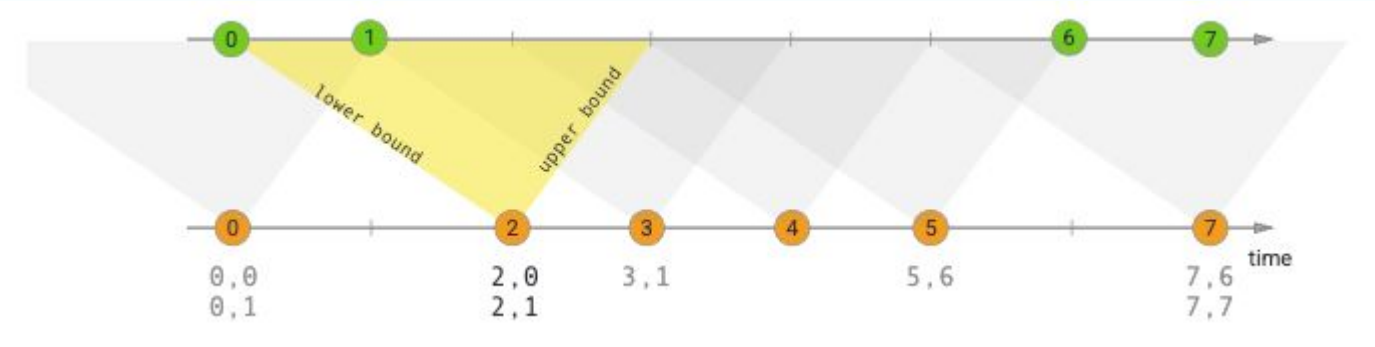
Strategic Cloud Engineer
Google Cloud

BEAM SUMMIT
NYC 2023

Why this talk?

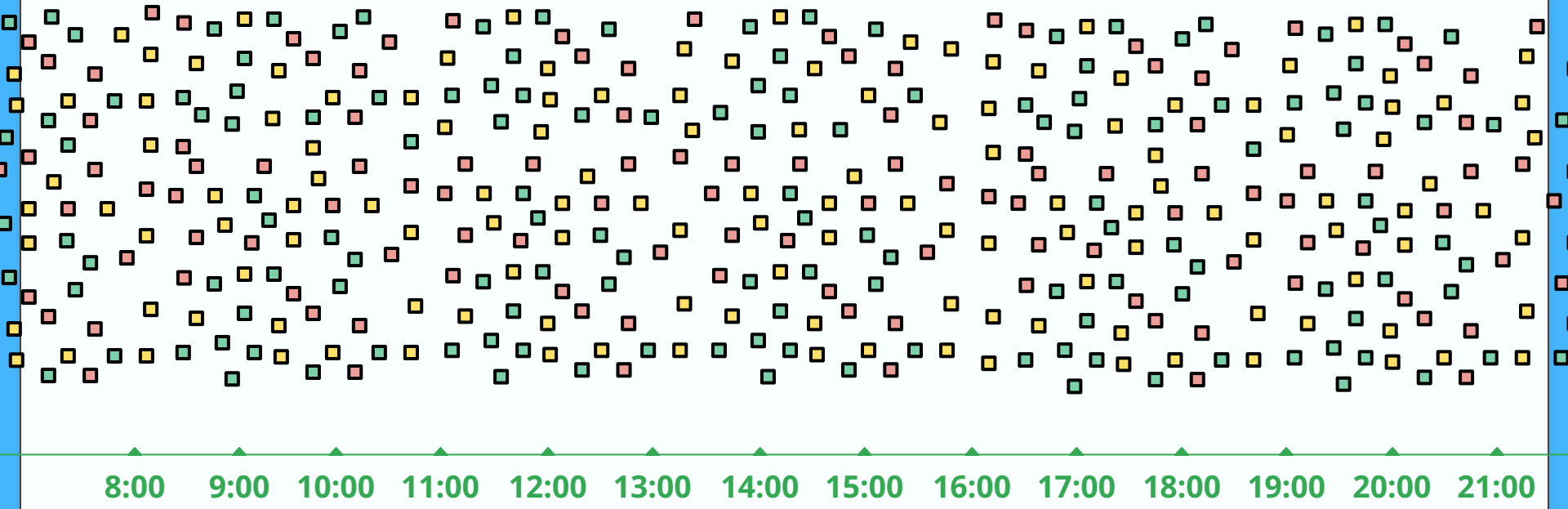
Simple ask, difficult implementation:

- I want to do interval joins between streams in Apache Beam
  - And joins between session-windowed streams too

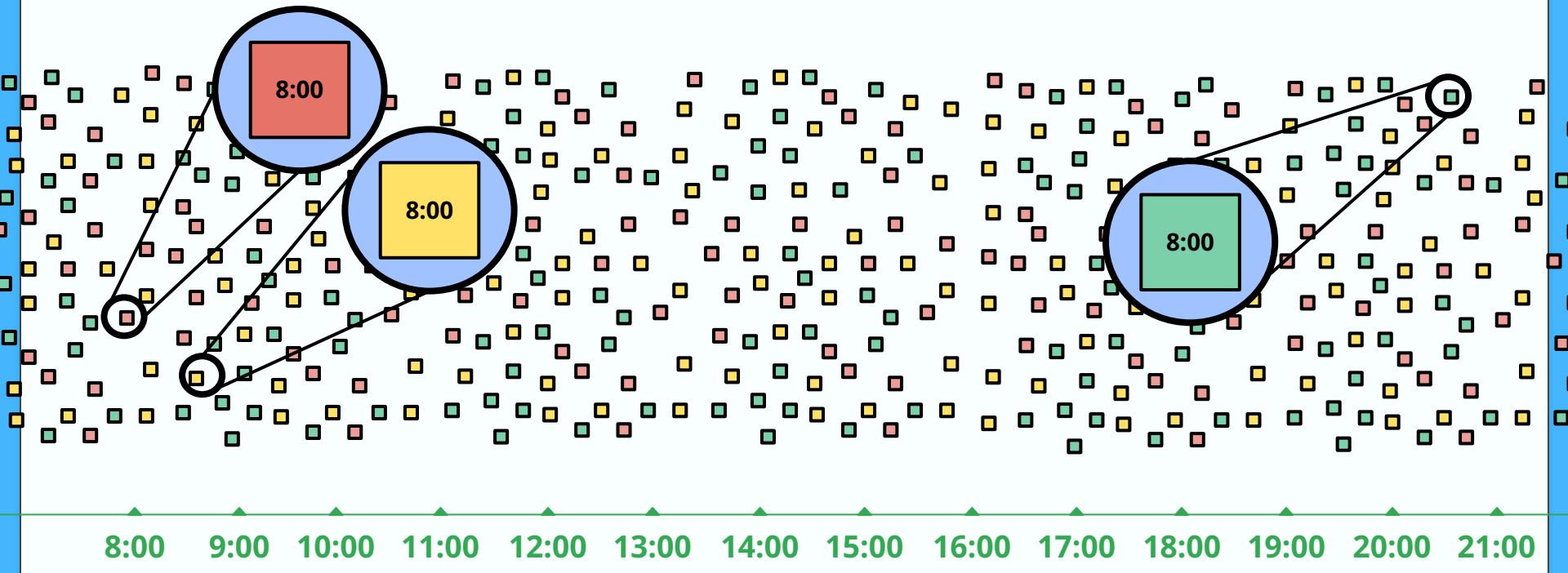


*https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/operators/joining/

What is a data stream?



What is a data stream?



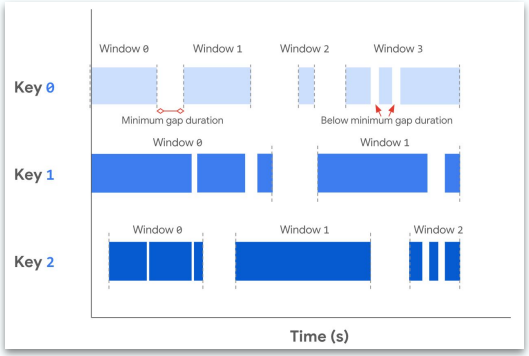
Impossible to guarantee order, but order can be recovered

Recovering order: assumptions

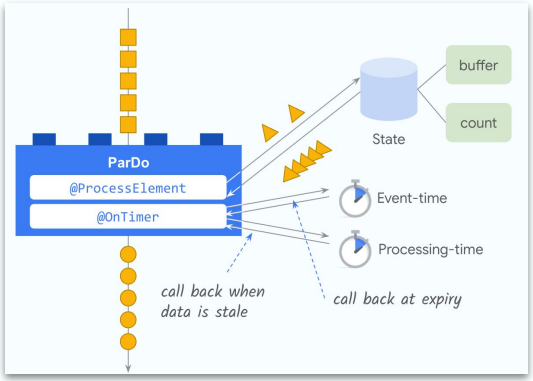
- Keyed stream
  - KV in Java
  - 2-elements tuple in Python
- Size of group per key "small"
  - Sorting per key. No global sorting.
  - Groups should fit in memory.
  - Not strict, can be relaxed in some cases (but performance would suffer)
- Sorting by timestamp
  - But in some cases, we could sort by any other criteria
    - Data or metadata
- Streaming :)
  - In batch, sorting is less challenging
- Java and runner-dependent examples/features

Three approaches in this session

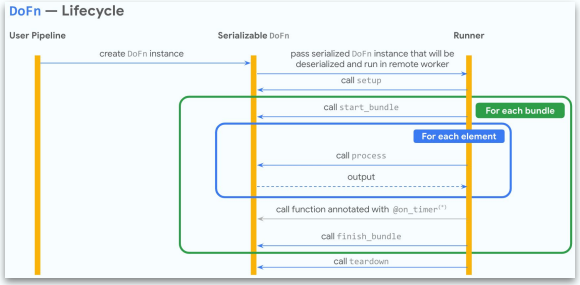
Windows



State & timers



DoFn annotations



Our data

Dummy data class

- Key
- Some value (*ignored*)
- Timestamp in data
- Property to close session

```
syntax = "proto3";

package dev.herraiz.protos;

message MyDummyEvent {
  string msg_key = 1;
  int32 value = 2;
  int64 event_timestamp = 3;
  optional bool is_last_msg = 4;
}
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/proto/events.proto

Generat

Elements are

```
public static List<TimestampedValue<MyDummyEvent>> generateData(
        int numEvents, String msgKey, Instant testEpoch) {
    Random r = new Random();
    List<TimestampedValue<MyDummyEvent>> events = new ArrayList<>();

    for (int k = 0; k < numEvents; k++) {
        // Generate events shifted ~1 sec from each other
        Long shift = r.longs(100, 2999).findFirst().getAsLong();
        Long ts = testEpoch.plus(shift).plus(1000 * (k + 1)).getMillis();

        MyDummyEvent.Builder builder =
                MyDummyEvent.newBuilder()
                        .setMsgKey(msgKey)
                        .setValue(k) // Sequential for easier debugging of order/lack of order
                        .setEventTimestamp(ts);

        if (k == numEvents - 1) {
            builder.setIsLastMsg(true);
        } else {
            builder.setIsLastMsg(false);
        }

        MyDummyEvent event = builder.build();

        TimestampedValue<MyDummyEvent> tsval =
                TimestampedValue.of(event, new org.joda.time.Instant(ts));

        events.add(tsval);
    }

    return events;
}
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/utils/Events.java#L36-L68

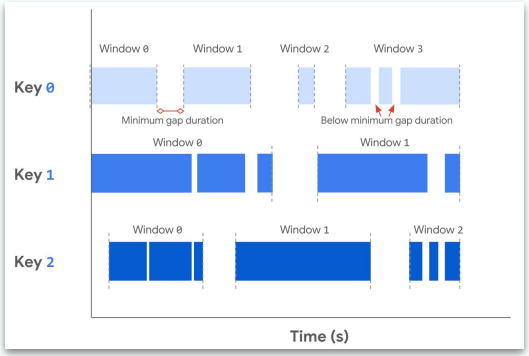
Data is shuffled before used

```
public static List<TimestampedValue<MyDummyEvent>> getTimestampedValues(int numMessages) {
    List<TimestampedValue<MyDummyEvent>> events =
            generateData(numMessages, MSG_KEY, TEST_EPOCH);
    Collections.shuffle(events); // Disorder data
    Collections.shuffle(events); // Disorder data
    Collections.shuffle(events); // Disorder data
    Collections.shuffle(events); // Disorder data
    return events;
}
```

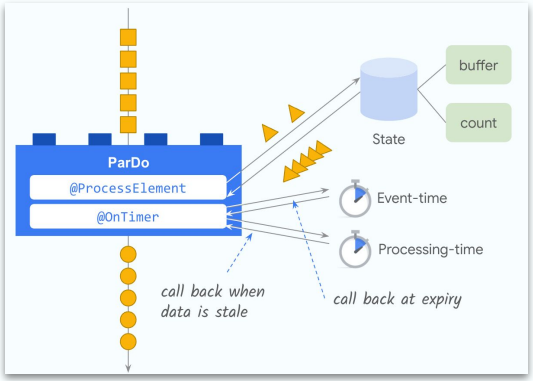
github.com/iht/beam-keyed-stream-sorting/blob/main/src/test/java/dev/herraiz/beam/transform/CommonTestConfig.java#L116-L124

First approach: windows

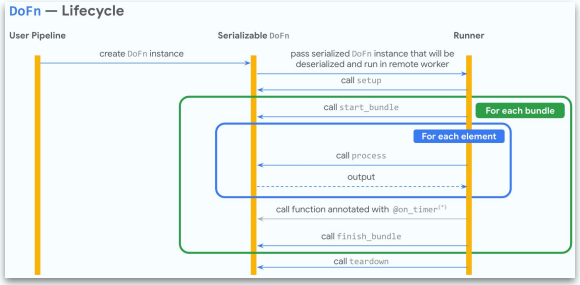
Windows



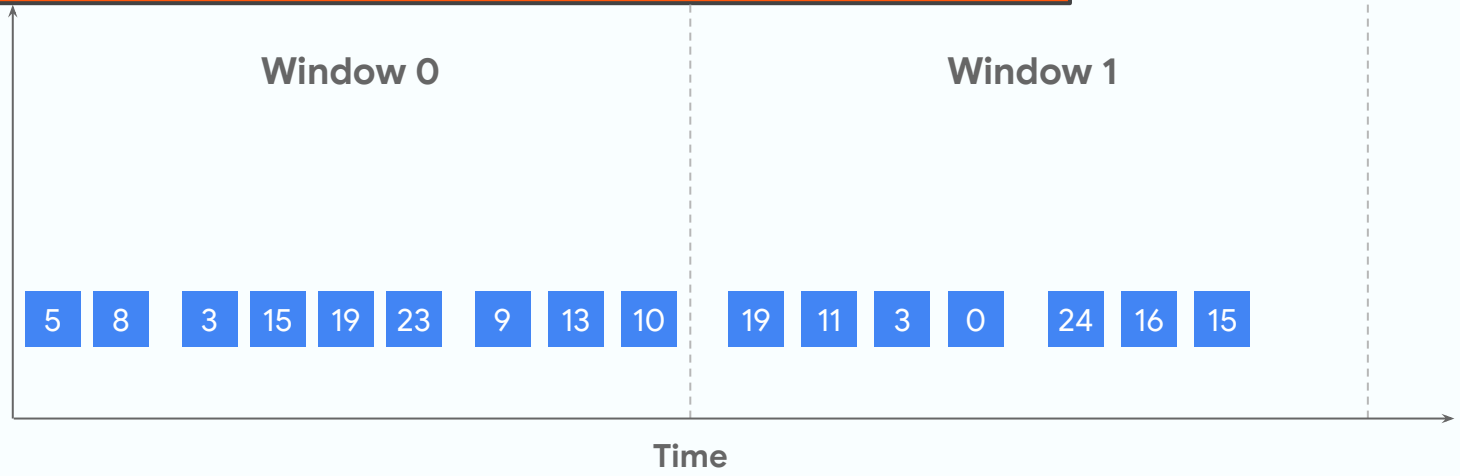
State & timers



DoFn annotations



Using windows



Group by key and sort

```
Window 0    [3, 5, 8, 9, 10, 13, 15, 19, 23]
Window 1    [0, 3, 11, 15, 16, 19, 24]
```

Using windows

- Apply session window
- Group by key
- Sort with DoFn

```java
@Override
public PCollection<KV<String, Iterable<MyDummyEvent>>> expand(
        PCollection<KV<String, MyDummyEvent>> input) {

    PCollection<KV<String, MyDummyEvent>> windowed =
            input.apply(
                    "Session windowing",
                    Window.<KV<String, MyDummyEvent>>into(
                                    Sessions.withGapDuration(Duration.standardSeconds(30)))
                            .triggering(AfterWatermark.pastEndOfWindow())
                            .discardingFiredPanes()
                            .withAllowedLateness(Duration.ZERO));

    PCollection<KV<String, Iterable<MyDummyEvent>>> grouped =
            windowed.apply("Group by key", GroupByKey.create());

    return grouped.apply("Sort", ParDo.of(new SortWithWindowsDoFn()));
}
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithWindows.java#L54-L71

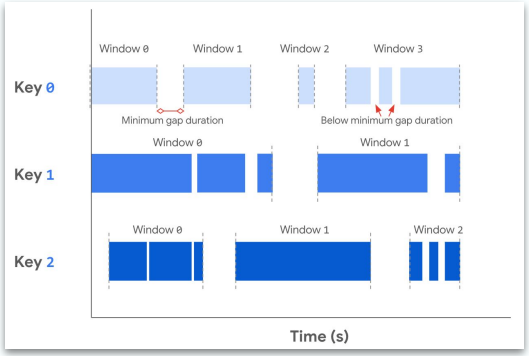
Using windows: sorting DoFn is quite simple

```
@ProcessElement
public void processElement(
        @Element KV<String, Iterable<MyDummyEvent>> element,
        OutputReceiver<KV<String, Iterable<MyDummyEvent>>> receiver) {

    List<MyDummyEvent> events = Lists.newArrayList(element.getValue());
    events.sort(new Events.MyDummyEventComparator());

    receiver.output(KV.of(element.getKey(), events));
}
```

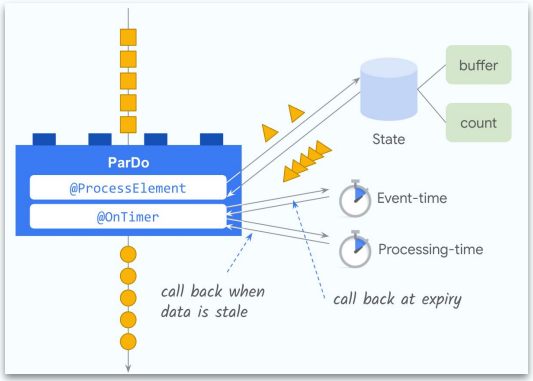
- Iterable loaded into memory (list) for sorting
  - Just because I am lazy, but not strictly necessary
  - https://beam.apache.org/documentation/sdks/java-extensions/#sorter

Second approach: state & timers

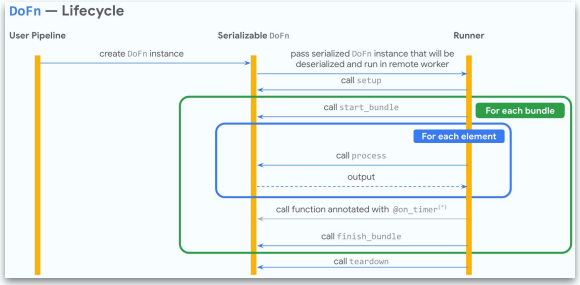
Windows



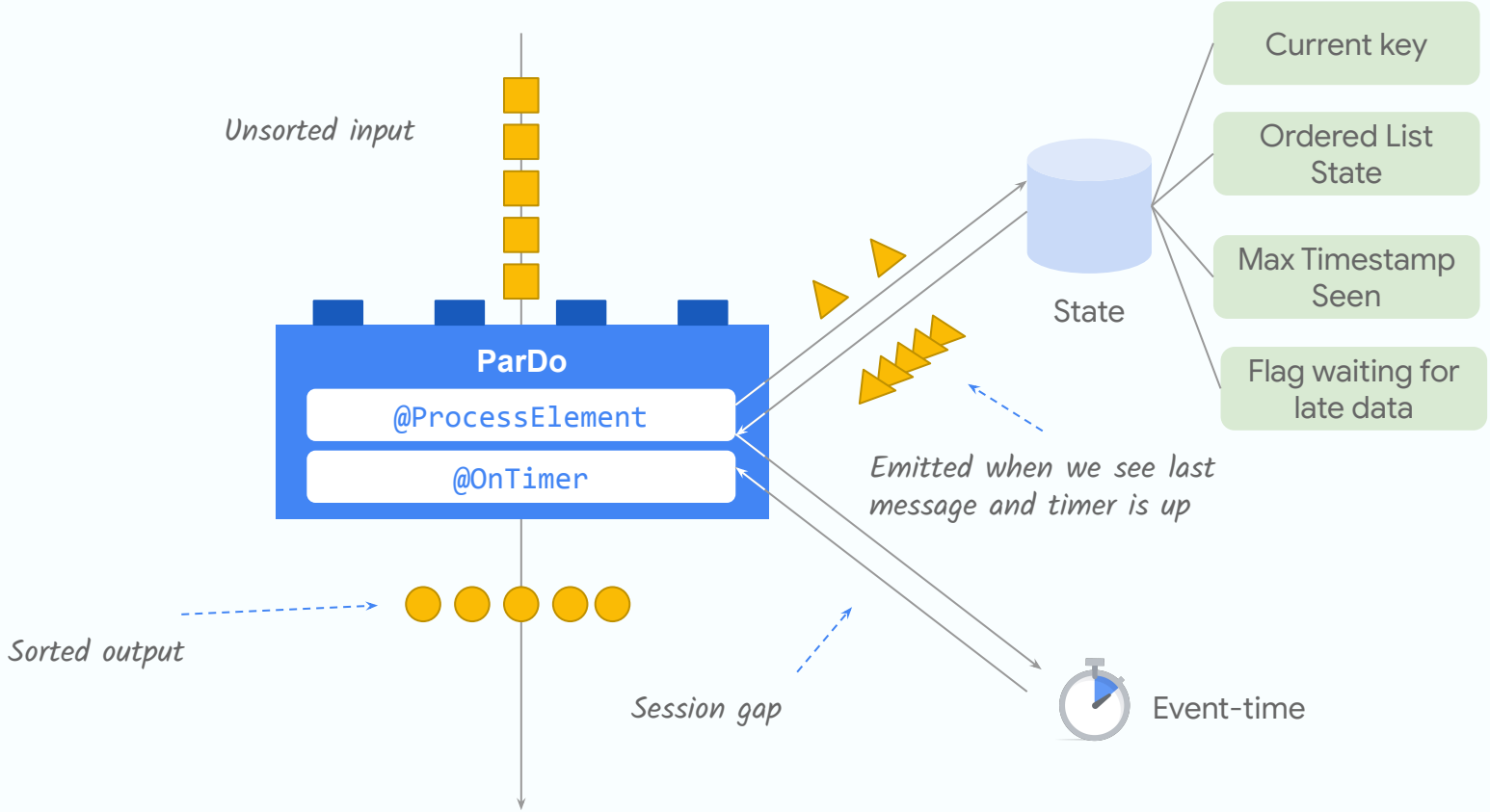
State & timers



DoFn annotations



State & timers



State & timers: variables in the DoFn

```
@ProcessElement
public void processElement(
        @Element KV<String, MyDummyEvent> element,
        @Timestamp Instant elementTimestamp,
        @StateId("currentKey") ValueState<String> currentKeyState,
        @AlwaysFetched @StateId("holdingUpAfterLastMsg")
                ValueState<Boolean> currentlyHoldingUpState,
        @StateId("elementsOrderedList") OrderedListState<MyDummyEvent> eventsListState,
        @StateId("maxTimestampSeen") CombiningState<Long, long[], Long> maxTimestampState,
        @TimerId("gapTimer") Timer gapTimer,
        OutputReceiver<KV<String, Iterable<MyDummyEvent>>> receiver) {
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithState.java#L93-L102

State & timers: update state, check status

Sorted automatically by timestamp

```
// Update state
currentKeyState.write(element.getKey());
eventsListState.add(TimestampedValue.of(element.getValue(), elementTimestamp));
maxTimestampState.add(elementTimestamp.getMillis());
// Check if we have met the conditions to close the session
boolean isLastMsg = element.getValue().getIsLastMsg();
// Messages coming out of order after the last msg will not verify the condition,
// so we need a state variable to remember that we have seen the last msg
boolean currentlyHoldingUp =
        Optional.ofNullable(currentlyHoldingUpState.read()).orElse(false);
boolean sessionEndFound = currentlyHoldingUp || isLastMsg;
```

```
if (sessionEndFound) {
    currentlyHoldingUpState.write(true);
    gapTimer.withOutputTimestamp(Instant.ofEpochMilli(maxTimestampState.read()))
            .offset(Duration.standardSeconds(sessionGap))
            .setRelative();
}
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithState.java#L103-L113
github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithState.java#L115-L120

State & timers: no need for explicit sorting

```
String key = currentKeyState.read();
List<MyDummyEvent> events =
        StreamSupport.stream(orderedListState.read().spliterator(), false)
                .map(ts -> ts.getValue())
                .collect(Collectors.toList());

receiver.outputWithTimestamp(
        KV.of(key, events), Instant.ofEpochMilli(maxTimestampState.read()));
```

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithState.java#L133-L140

State and timers

- Not all runners support *OrderedListState*
- Can be emulated with a *MapState*
- Awful performance using a *ValueState* with a *List*

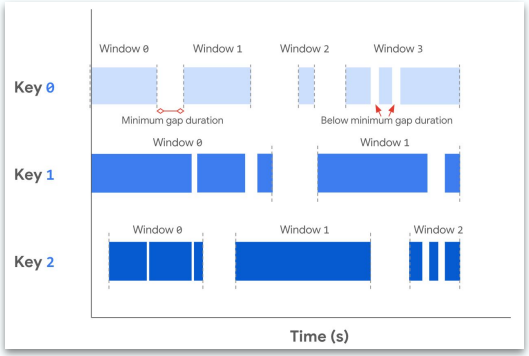
```
// Update state
currentKeyState.write(element.getKey());
Integer currentIndex = indexState.read();
if (currentIndex == null) {
    currentIndex = 0;
}
eventsMapState.put(currentIndex, element.getValue());
indexState.write(currentIndex + 1);

maxTimestampState.add(elementTimestamp.getMillis());
```

```
List<MyDummyEvent> events = new ArrayList<>();

for (int k = 0; k < indexState.read(); k++) {
    events.add(eventsMapState.get(k).read());
}

// sort(new MyDummyEventComparator()); <-- NO NEED TO SORT

receiver.outputWithTimestamp(
        KV.of(key, events), Instant.ofEpochMilli(maxTimestampState.read()));
```

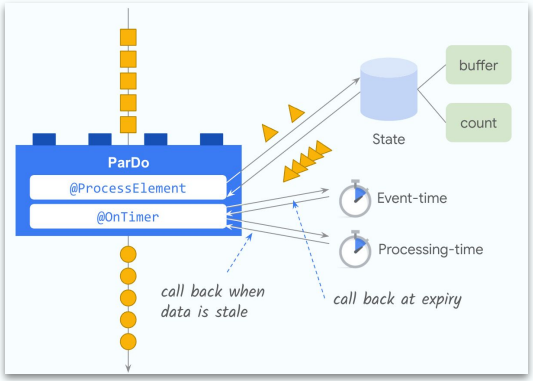
github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithMapState.java#L104-L113
github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithMapState.java#L142-L151

Third approach: DoFn annotations

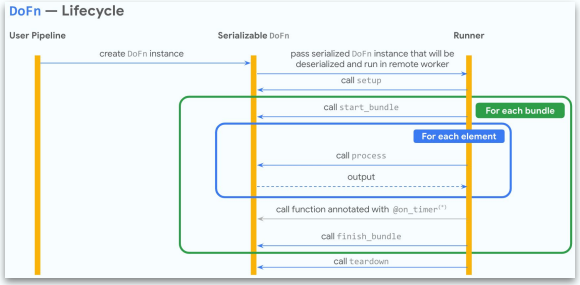
Windows



State & timers



DoFn annotations



DoFn annotations: list and time sorted input

Magic happens here

```
@RequiresTimeSortedInput
@ProcessElement
public void processElement(
        @Element KV<String, MyDummyEvent> element,
        @Timestamp Instant elementTimestamp,
        @StateId("currentKey") ValueState<String> currentKeyState,
        @AlwaysFetched @StateId("holdingUpAfterLastMsg")
                ValueState<Boolean> currentlyHoldingUpState,
        @AlwaysFetched @StateId("elementsList")
                ValueState<List<MyDummyEvent>> eventsListState,
        @StateId("maxTimestampSeen") CombiningState<Long, long[], Long> maxTimestampState,
        @TimerId("gapTimer") Timer gapTimer,
        OutputReceiver<KV<String, Iterable<MyDummyEvent>>> receiver) {
    // Update state
    currentKeyState.write(element.getKey());

    List<MyDummyEvent> events = eventsListState.read();
    if (events == null) {
        events = new ArrayList<>();
    }
    events.add(element.getValue());
    eventsListState.write(events);
    maxTimestampState.add(elementTimestamp.getMillis());
```

Not so bad performance, since no sorting happening on appends

github.com/iht/beam-keyed-stream-sorting/blob/main/src/main/java/dev/herraiz/beam/transform/SortWithAnnotations.java#L89-L111

What's the best option? (streaming)



Windows	☑	☑	☑	35 secs
SortedListState	☑*	☑	☐	28 secs
MapState	☑*	☑	☐	45 secs
@RequireTimeSortedInput	☐	☑	☐	47 secs

Times running the test suite at
github.com/iht/beam-keyed-stream-sorting/
using the Direct Runner

* Runner V1

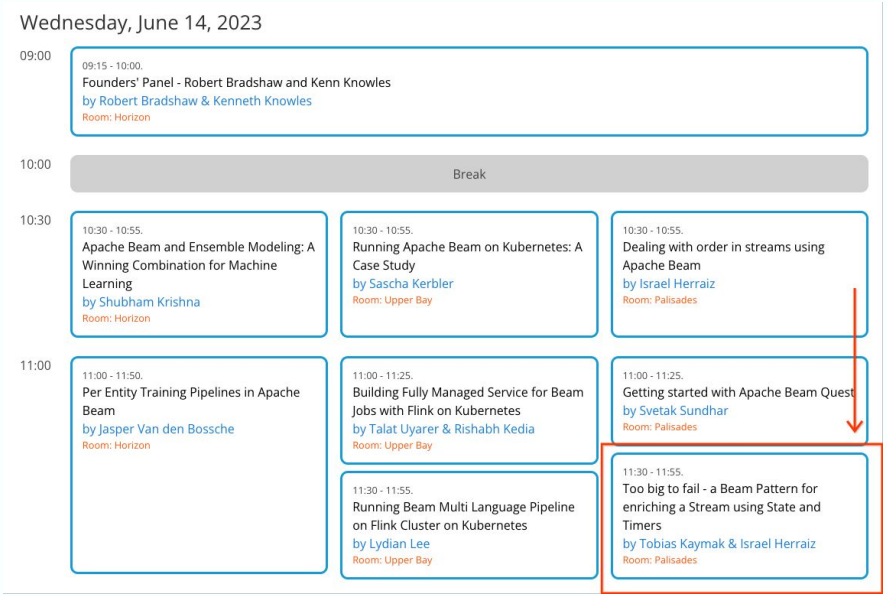
What's the best option? (streaming)

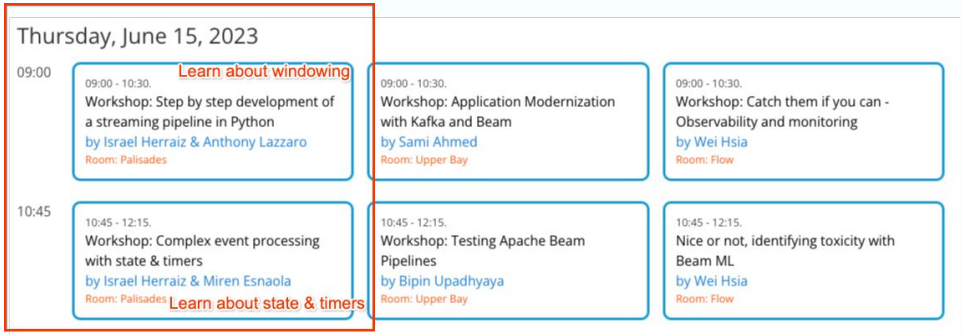
				
Windows	☑	☑	☑	30-35 secs
SortedListState	☑*	☑	☐	30-35 secs
MapState	☑*	☑	☐	45-50 secs
@RequireTimeSortedInput	☐	☑	☐	40-45 secs

Times running the test suite at
github.com/iht/beam-keyed-stream-sorting/
using the Direct Runner

* Runner V1

Don't miss the related talks and workshops!





Dealing with order in streams

Full example: github.com/iht/beam-keyed-stream-sorting

QUESTIONS?

Israel Herraiz
@herraiz
linkedin.com/in/herraiz
github.com/iht


BEAM
SUMMIT