03

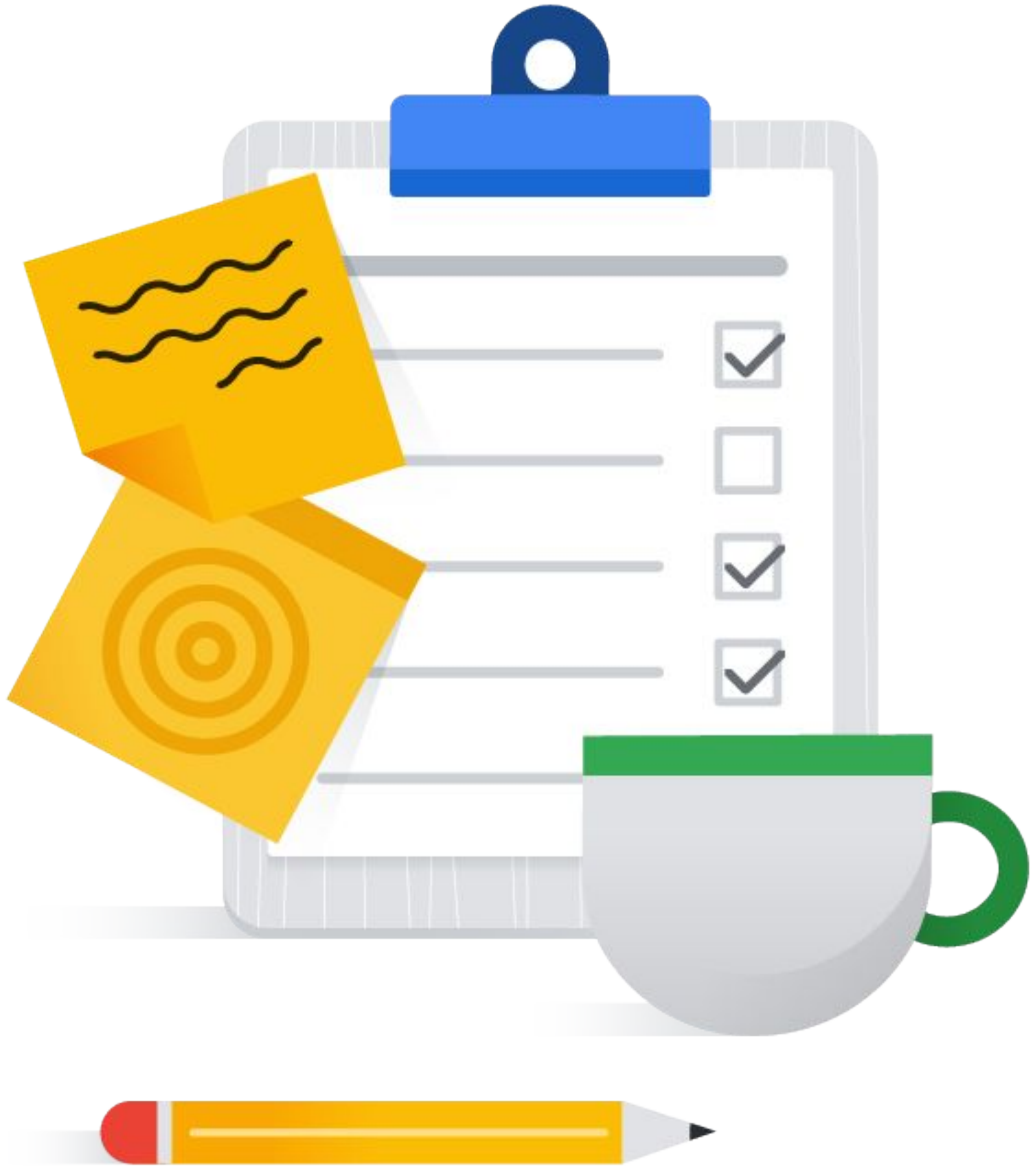
Dataflow Streaming Features

Dataflow Streaming Features

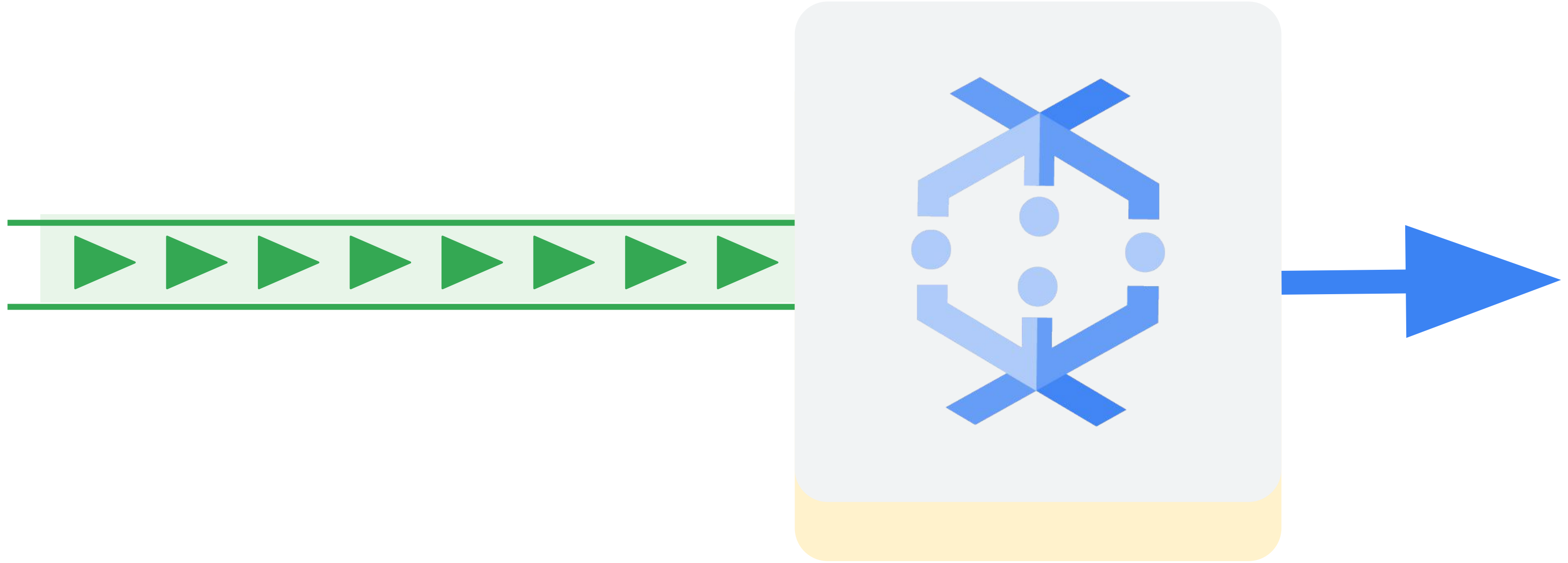
01	Streaming data challenges
02	Dataflow windowing

Dataflow Streaming Features

01	Streaming data challenges
02	Dataflow windowing



Streaming features of Dataflow

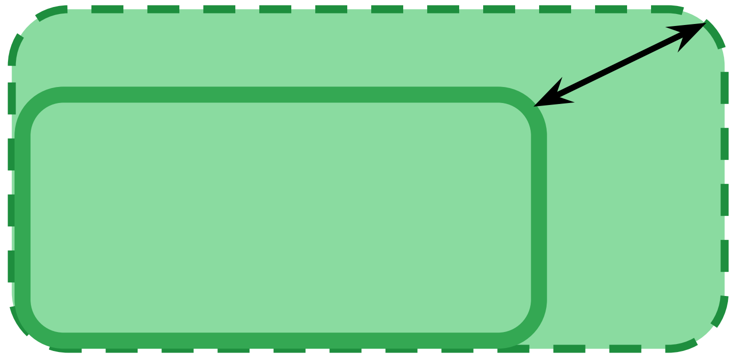


Qualities that Dataflow contributes to data engineering solutions:

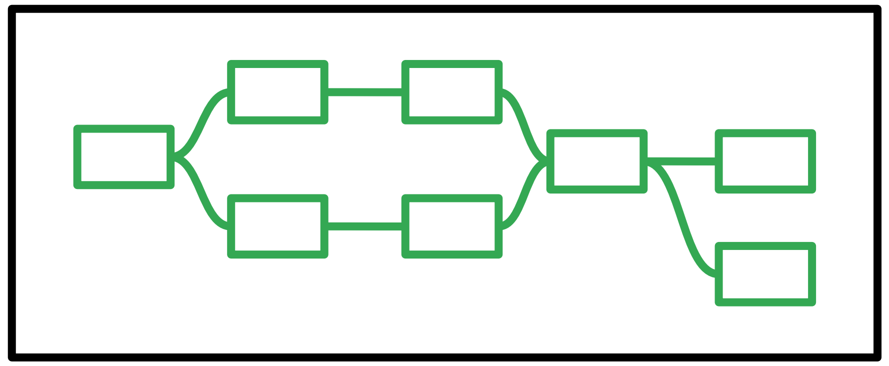
- Scalability
- Low latency

Continuing from the Data Processing course

Unbounded PCollection

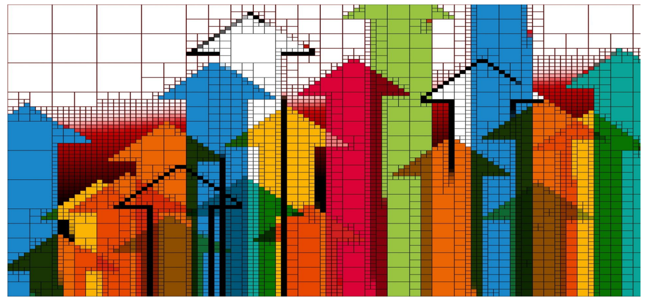


Pipeline



Streaming Jobs

There are challenges with processing streaming data



Scalability

Streaming data generally only grows larger and more frequent

Fault Tolerance

Maintain fault tolerance despite increasing volumes of data

Model

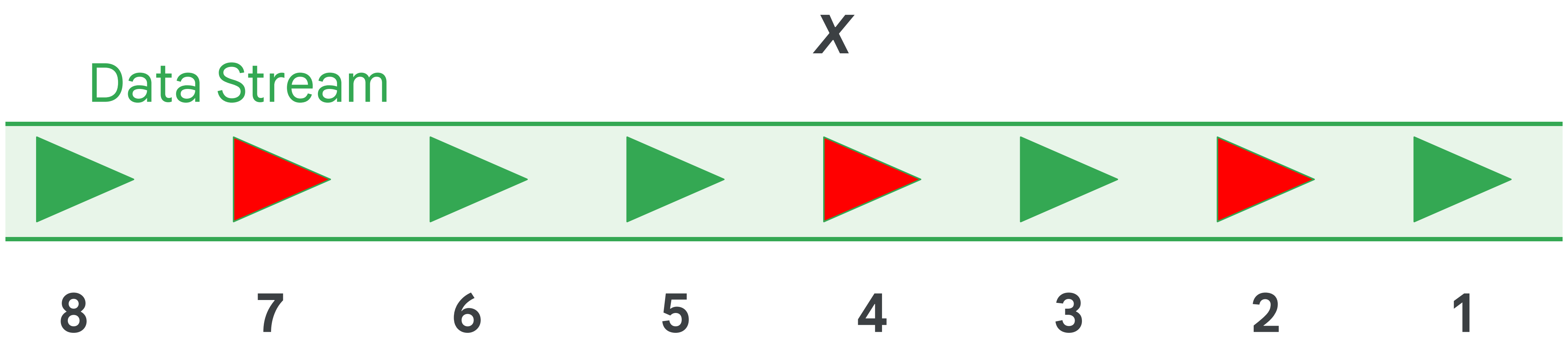
Is it streaming or repeated batch?

Timing

What if data arrives late?

How do you aggregate an unbounded set?

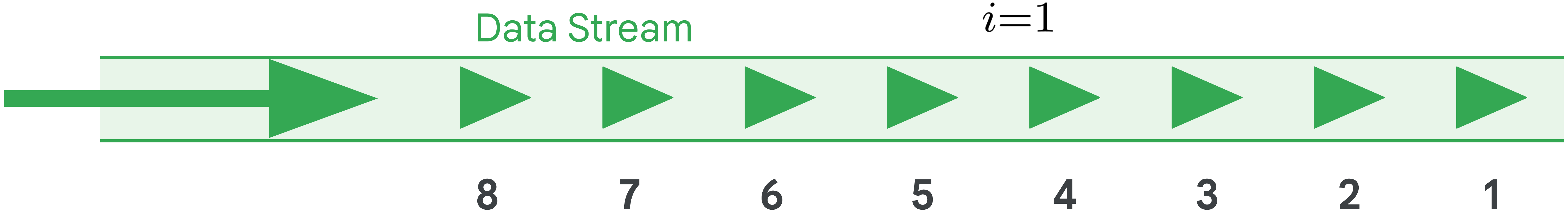
Non-aggregating operations such as filtering are straightforward



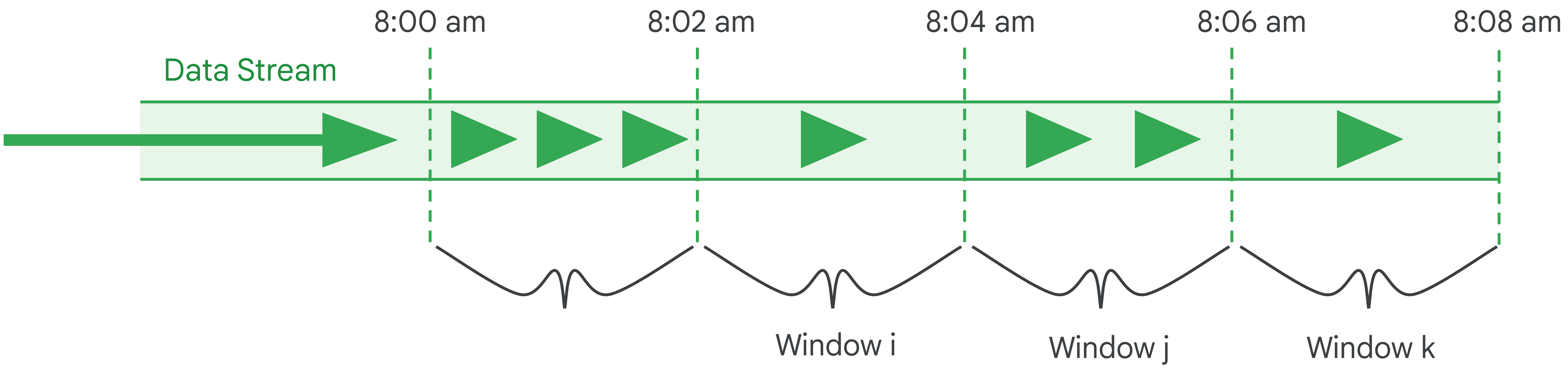
But how do you take an average on an unbounded stream of data?

$$\sum_{i=1}^{n} x_i$$

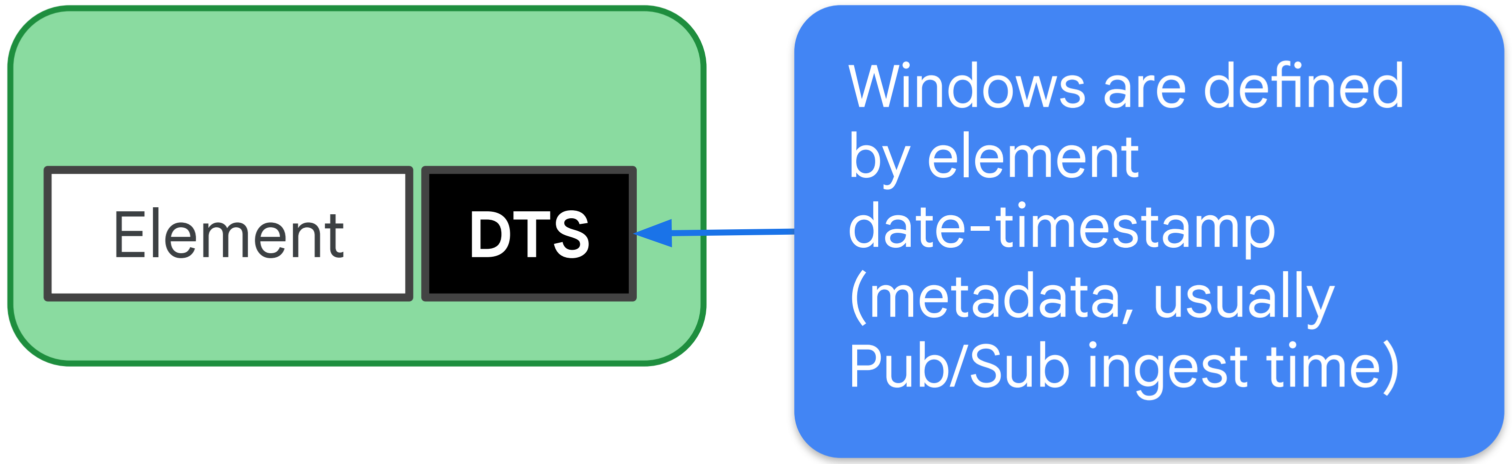
What is the stopping limit?



Divide the stream into a series of finite windows



$$\sum_{i}^{n} x_i \qquad \sum_{j}^{n} x_j \qquad \sum_{k}^{n} x_k$$



X_i Represents data in window i

n Represents number of data in each window

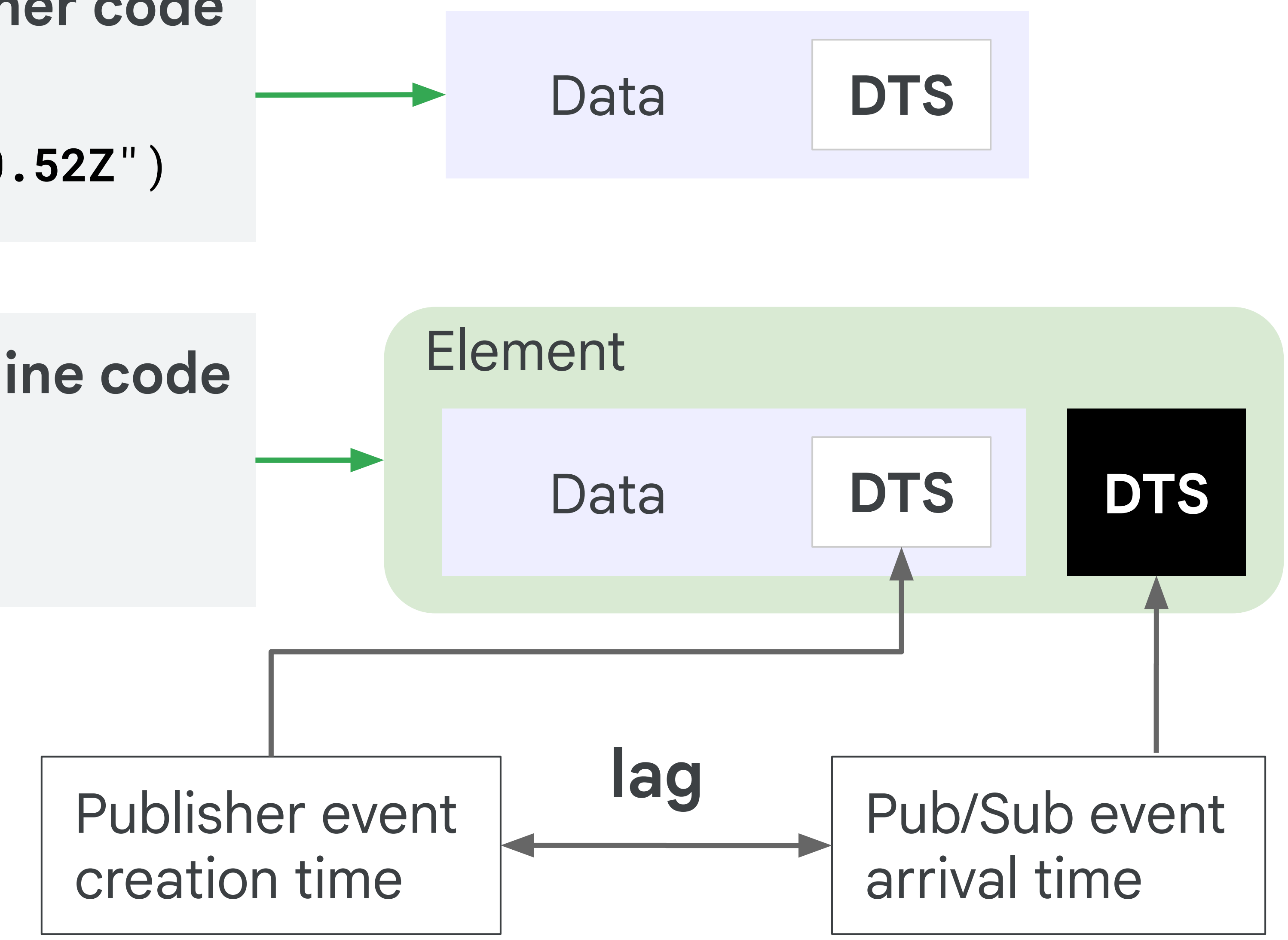
Message ordering and late data: The timestamp matters ... and windowing

Pub/Sub publisher code

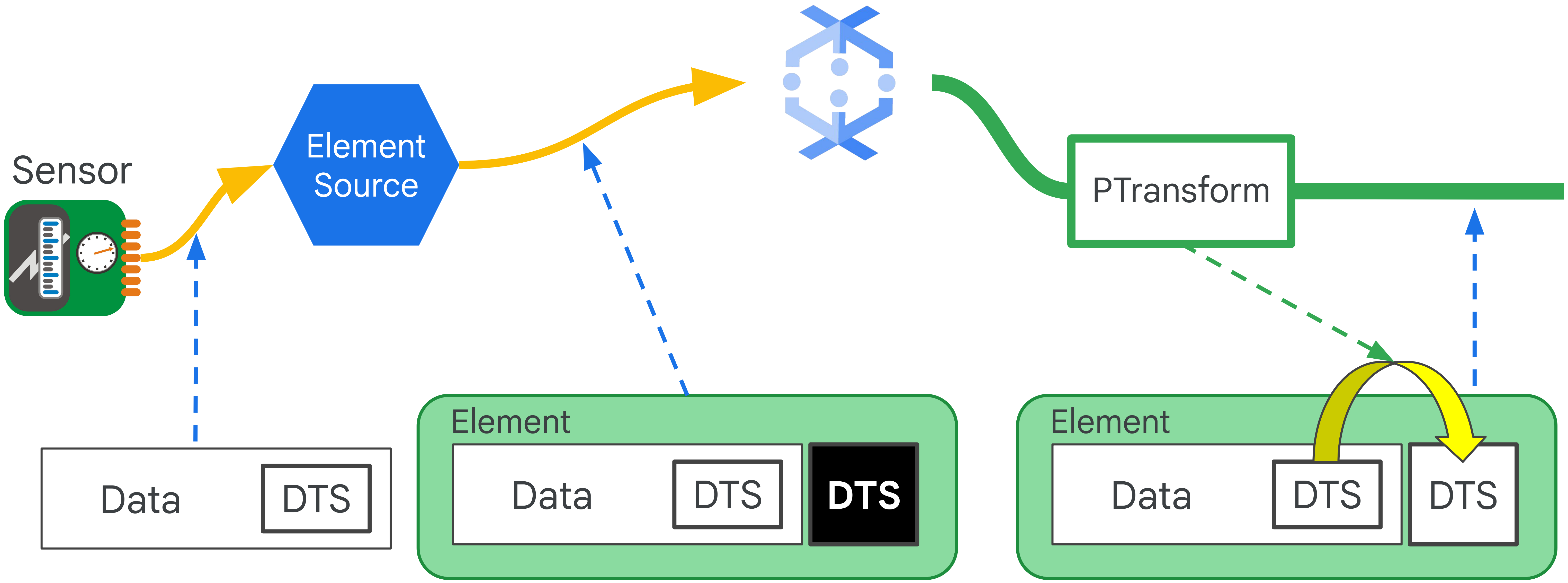
```
msg.publish(event_data, mytime="2020-04-12T23:20:50.52Z")
```

Dataflow pipeline code

```
p.apply( PubsubIO.readStrings()
   .fromTopic(t).withTimestampAttribute("mytime") )
```



Modify the date-timestamp with a PTransform if needed



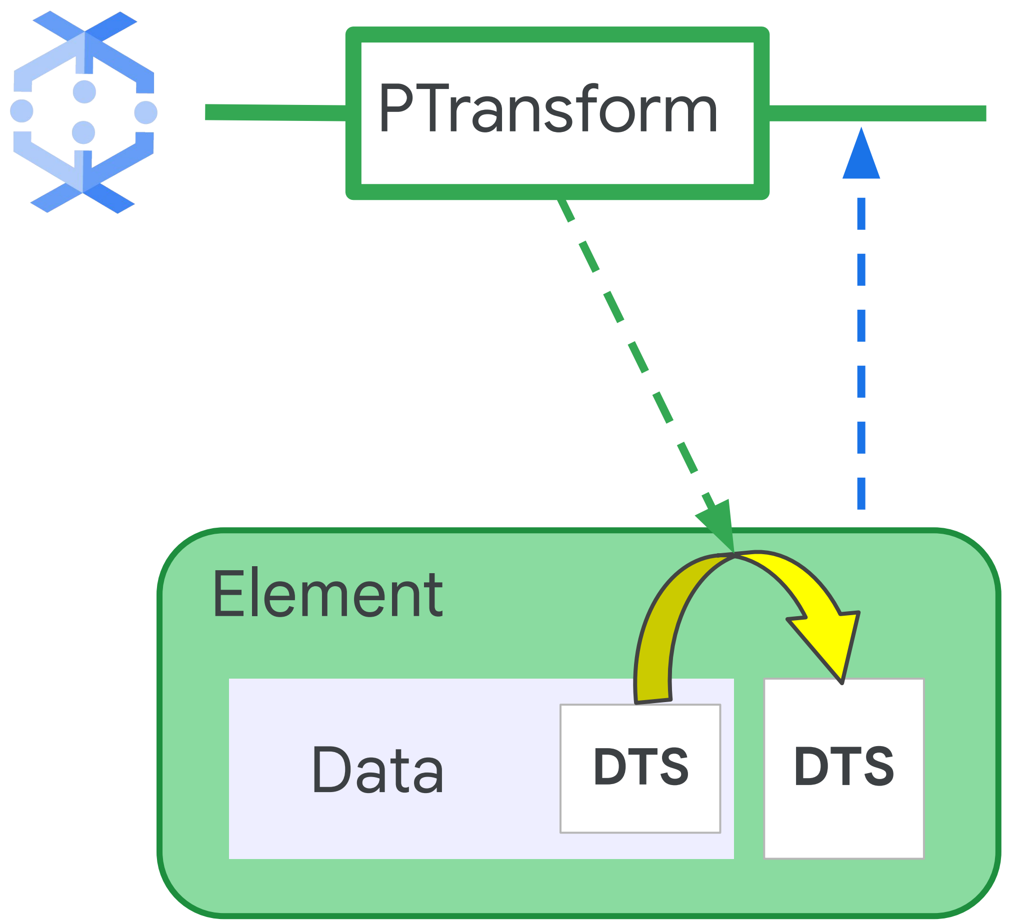
Code to modify date-timestamp

Python

```
unix_timestamp = extract_timestamp_from_log_entry(element)
    # Wrap and emit the current entry and new timestamp in a
TimestampedValue.
     yield beam.window.TimestampedValue(element, unix_timestamp)
```

Java

```
c.outputWithTimestamp (element, timestamp);
```



Duplication will happen: Exactly-once processing with Pub/Sub and Dataflow

Pub/Sub publisher code

```
msg.publish(event_data, myid="34xwy57223cdg")
```

Dataflow pipeline code

```
p.apply(
   PubsubIO.readStrings().fromTopic(t).idLabel("myid") )
```

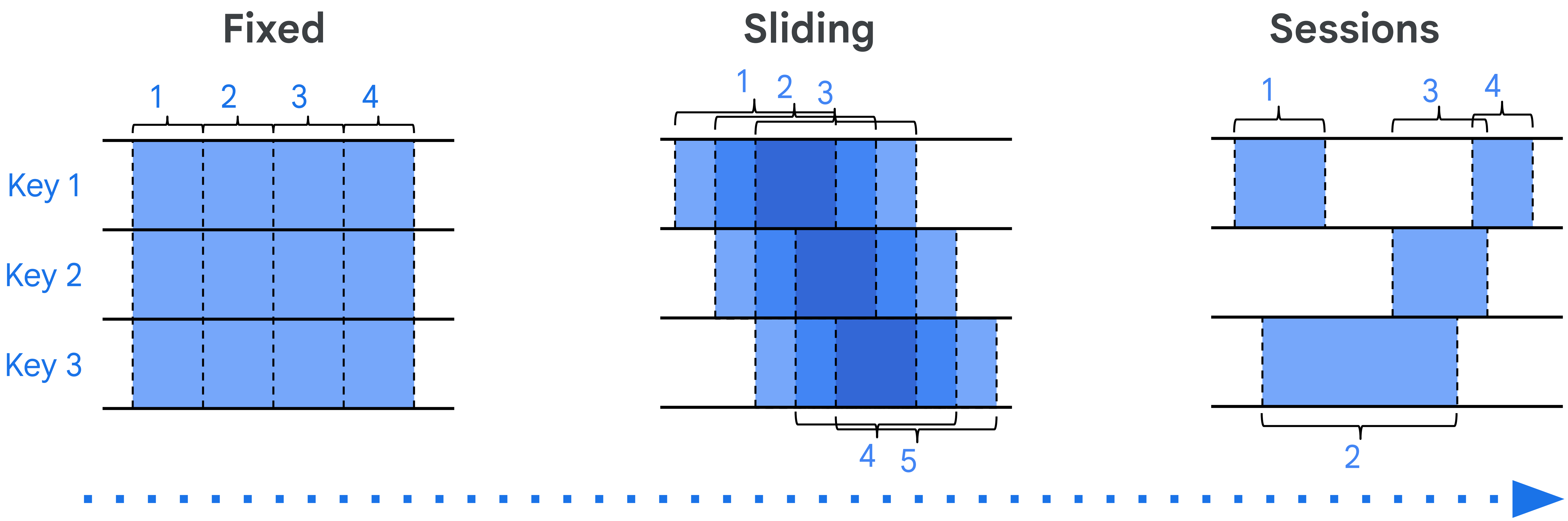
Dataflow Streaming Features

01	Streaming data challenges
02	Dataflow windowing

Three kinds of windows fit most circumstances

- Fixed
- Sliding
- Sessions

Three kinds of windows fit most circumstances



Windowing divides data into time-based finite chunks

Often required when doing aggregations over unbounded data

Setting time windows

Fixed-time windows

Python

```
from apache_beam import window
fixed_windowed_items = (
    items | 'window' >> beam.WindowInto(window.FixedWindows(60)))
```

Sliding time windows

Python

```
from apache_beam import window
sliding_windowed_items = (
    items | 'window' >> beam.WindowInto(window.SlidingWindows(30, 5)))
```

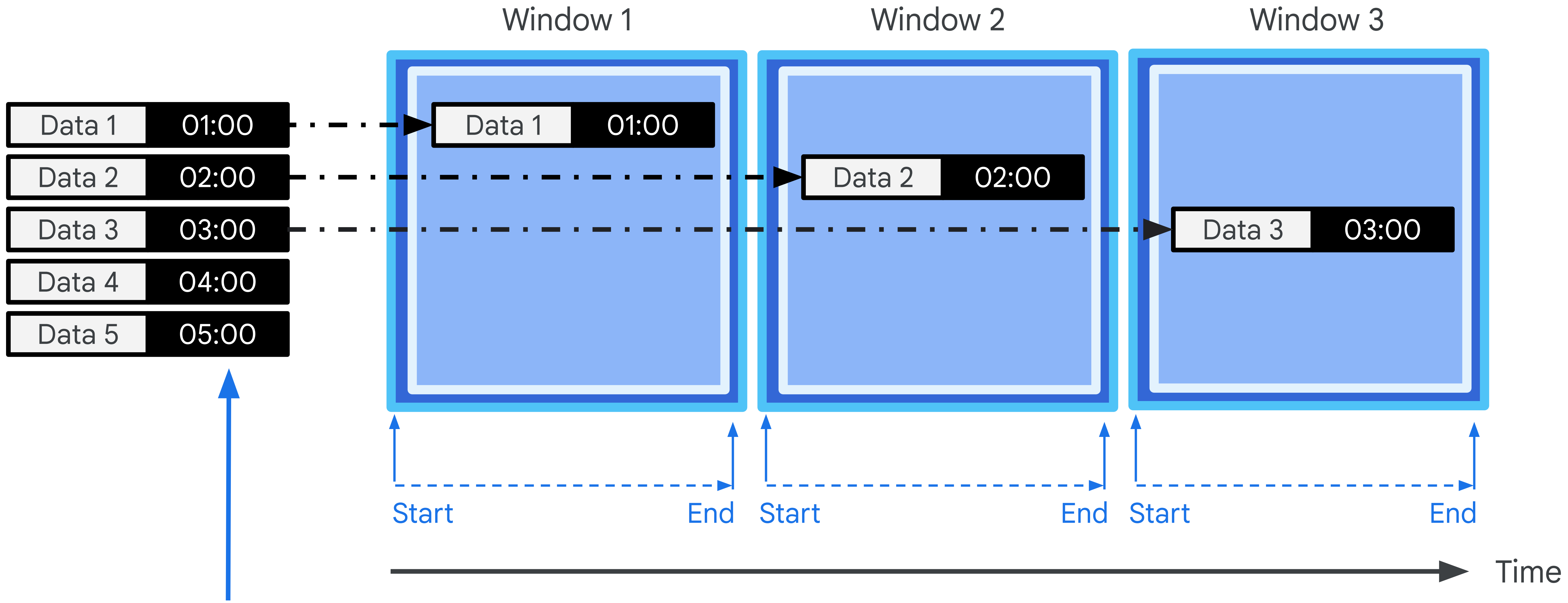
Session windows

Python

```
from apache_beam import window
session_windowed_items = (
    items | 'window' >> beam.WindowInto(window.Sessions(10 * 60)))
```

Remember:
you can apply windows to batch data, although you may need to generate the metadata date-timestamp on which windows operate.

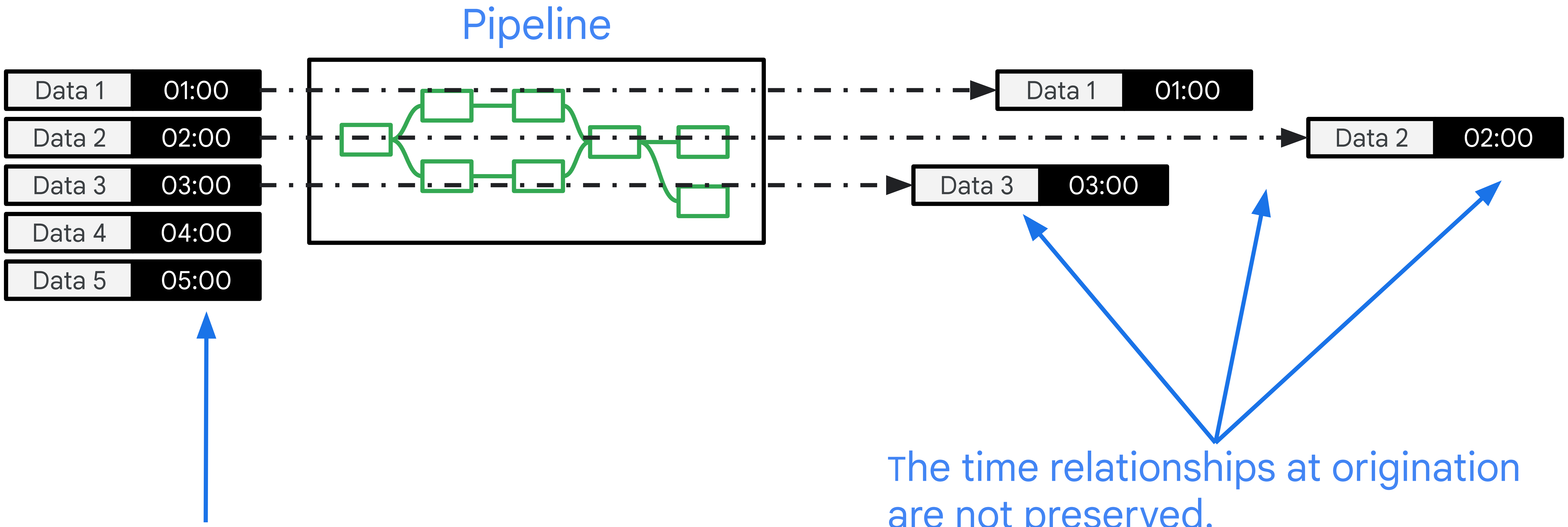
Windowing by time if there is no latency



This is the event time when the event originated.

These windows are defined to be one minute long.

Pipeline processing can introduce latency



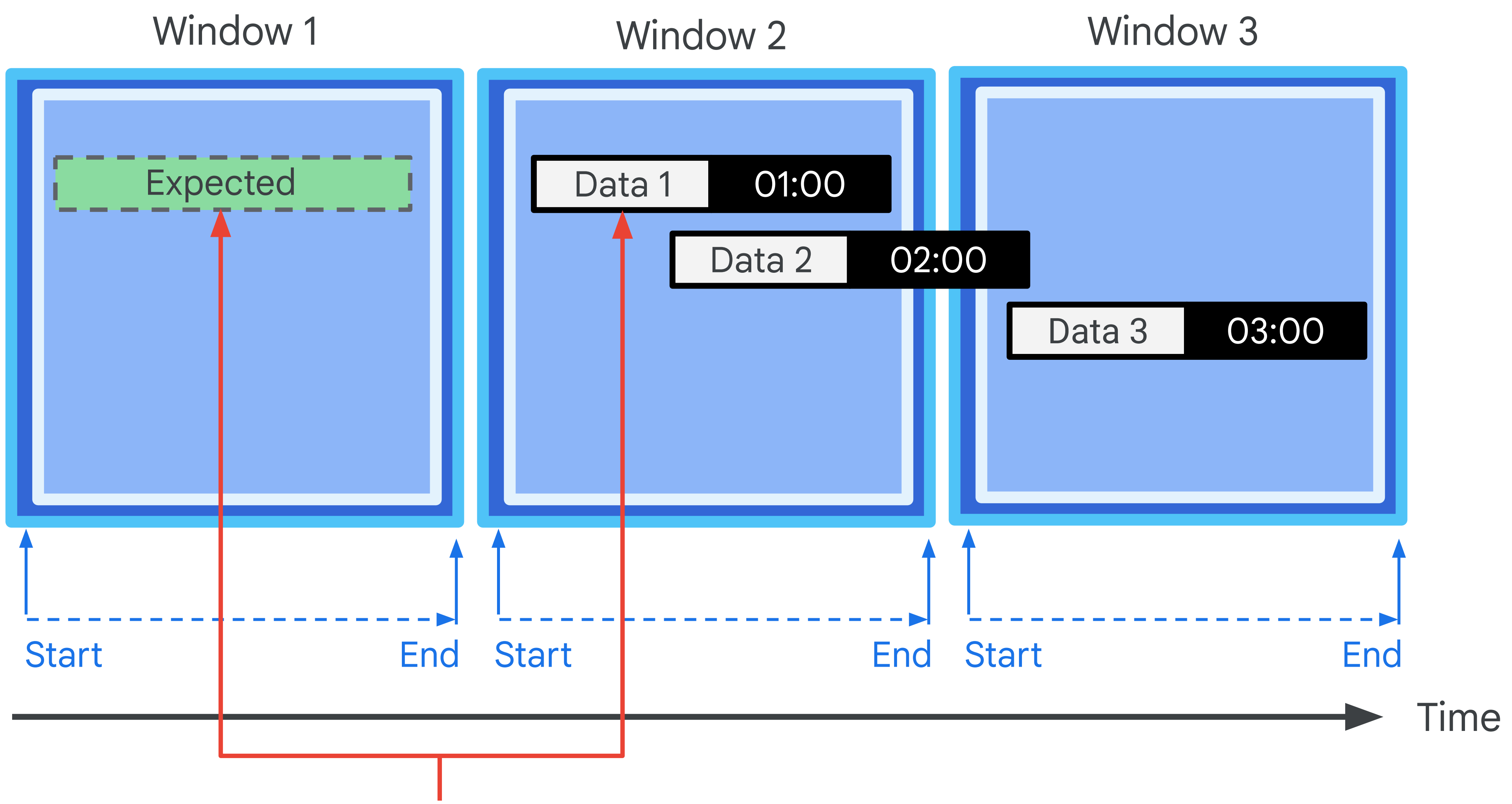
This is the event time when the event originated.

The time relationships at origination are not preserved.

They are arriving later than anticipated. And some of them are outside the original 1 minute window.

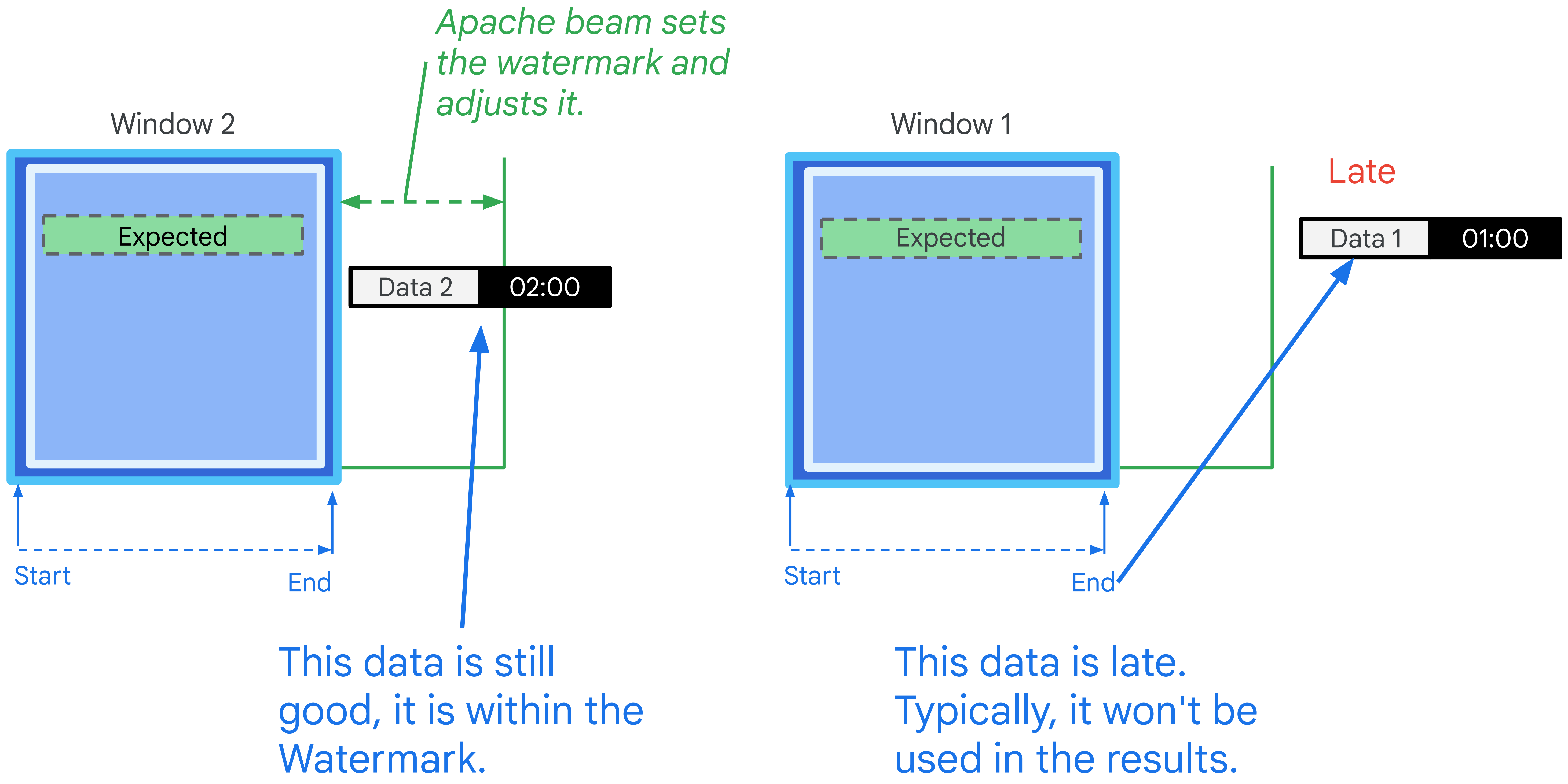
How should Dataflow deal with this situation?

The data could be a little past the window or a lot. Data 2 is a little outside of Window 2. Data 1 is completely outside of Window 1.

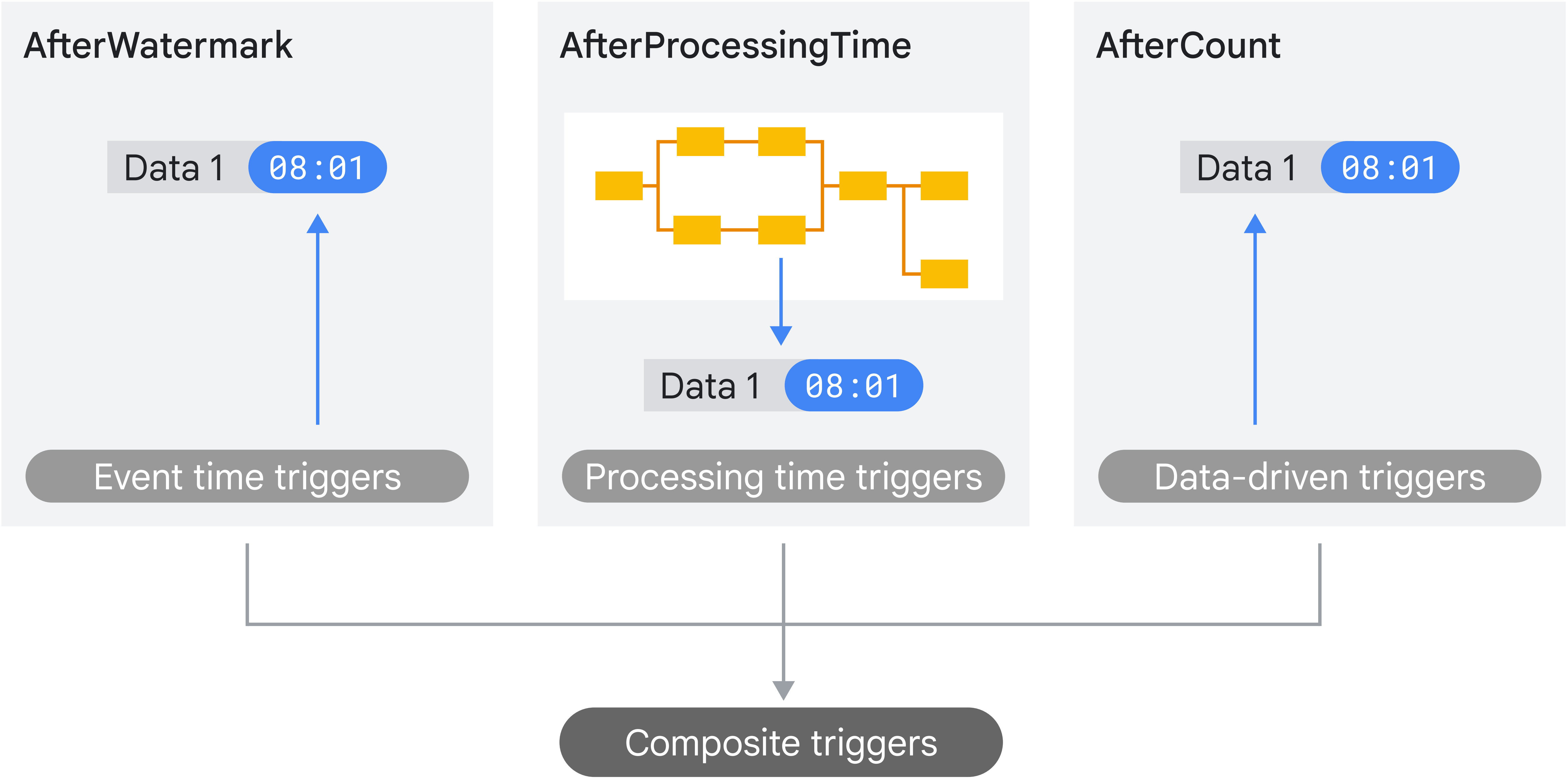


The difference in time from when data was expected to when it actually arrived is called the **lag time.**

Watermarks provide flexibility for a little lag time



Custom triggers



Some example triggers

```
pcollection | WindowInto(
    SlidingWindows(60, 5),                          # Sliding window of 60 seconds, every 5 seconds
    trigger=AfterWatermark(                         # Relative to the watermark, trigger:
      early=AfterProcessingTime(delay=30),          # -- fires 30 seconds after pipeline commences
      late=AfterCount(1))                           # -- and for every late record (< allowedLateness)
    accumulation_mode=AccumulationMode.ACCUMULATING)  # the pane should have all the records
    allowed_lateness=Duration(seconds=2*24*60*60))  # 2 days
```

```
pcollection | WindowInto(
    FixedWindows(60),                              # Fixed window of 60 seconds
    trigger=Repeatedly(                            # Set up a composite trigger that triggers ...
        AfterAny(                                  # whenever either of these happens:
            AfterCount(100),                       # -- 100 elements accumulate
            AfterProcessingTime(1 * 60))),         # -- every 60 seconds (ignore watermark)
    accumulation_mode=AccumulationMode.DISCARDING)  # the trigger should be with only new records
```

https://beam.apache.org/documentation/programming-guide/#composite-triggers

You can allow late data past the watermark

Allowing Late Data

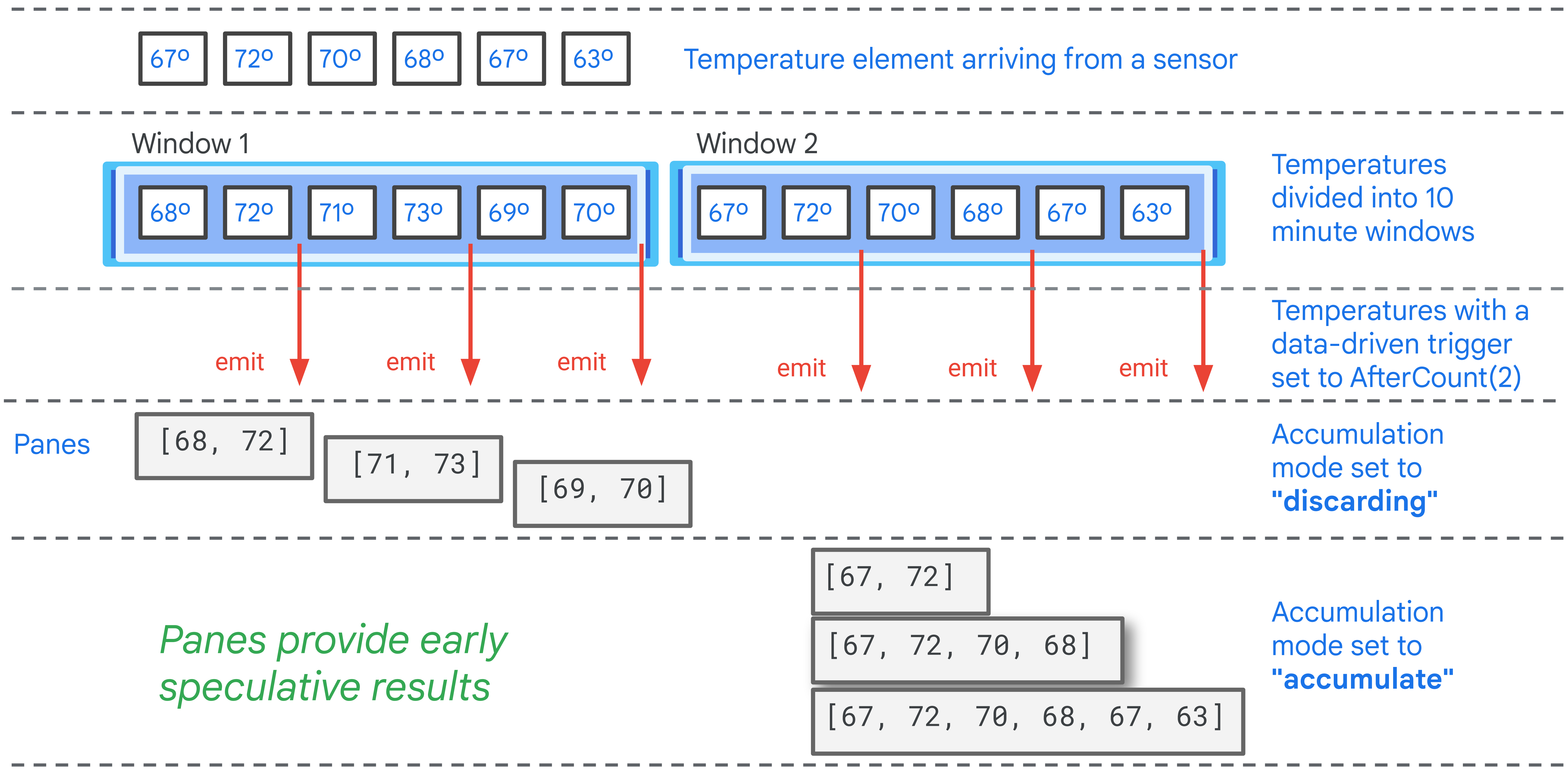
Java

```
PCollection<String> items = ...;

    PCollection<String> fixedWindowedItems = items.apply(

Window.<String>into(FixedWindows.of(Duration.standardMinutes(1)))
              .withAllowedLateness(Duration.standardDays(2)));
```

Python

```
pc = [Initial PCollection]
   pc | beam.WindowInto(
              FixedWindows(60),
              trigger=trigger_fn,
              accumulation_mode=accumulation_mode,
              timestamp_combiner=timestamp_combiner,
              allowed_lateness=Duration(seconds=2*24*60*60)) # 2 days
```

Accumulation modes: What to do with additional events



Lab Intro

Streaming Data Processing:
Streaming Data Pipelines

Lab objectives

01 Launch Dataflow and run a Dataflow job

02 Understand how data elements flow through the transformations of a Dataflow pipeline

03 Connect Dataflow to Pub/Sub and BigQuery

04 Observe and understand how Dataflow autoscaling adjusts compute resources to process input data optimally

05 Learn where to find logging information created by Dataflow

06 Explore metrics and create alerts and dashboards with Cloud Monitoring