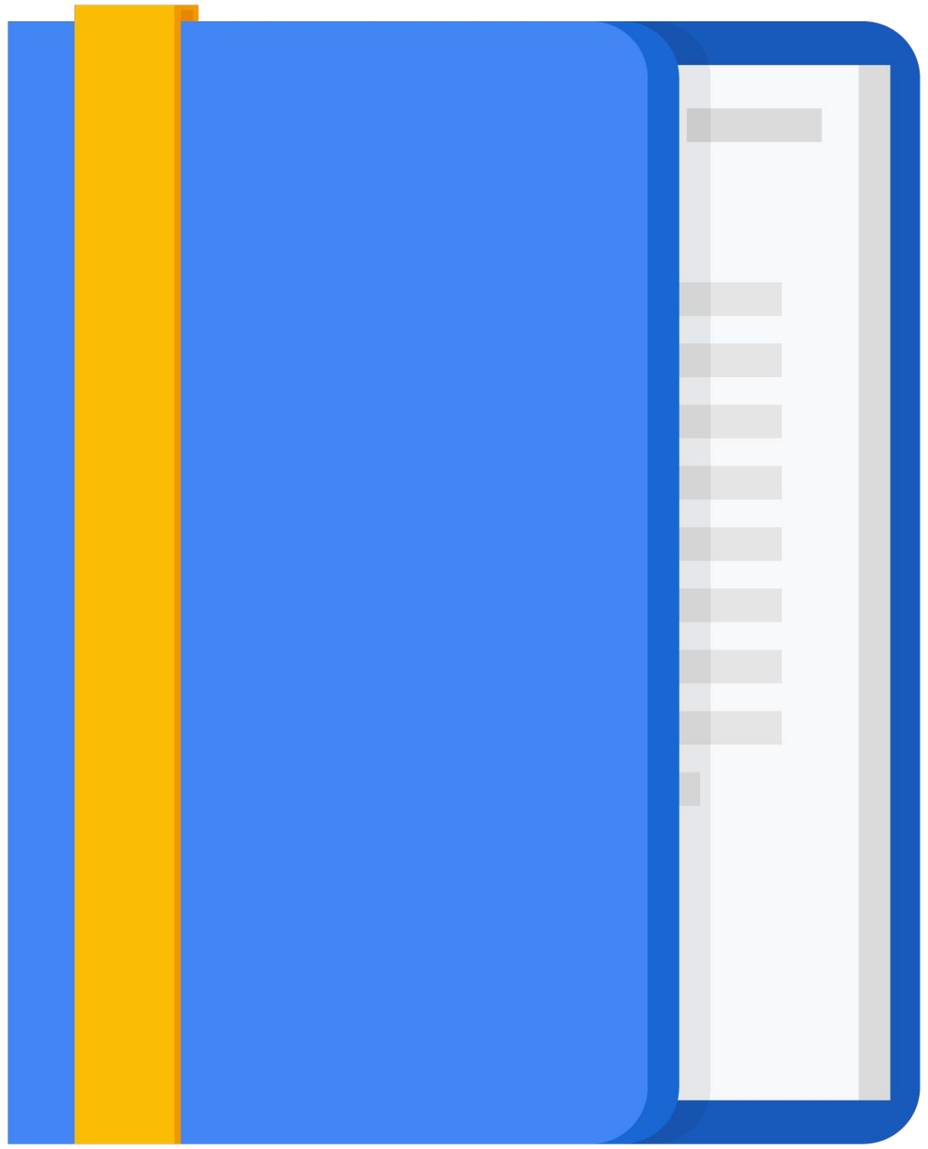
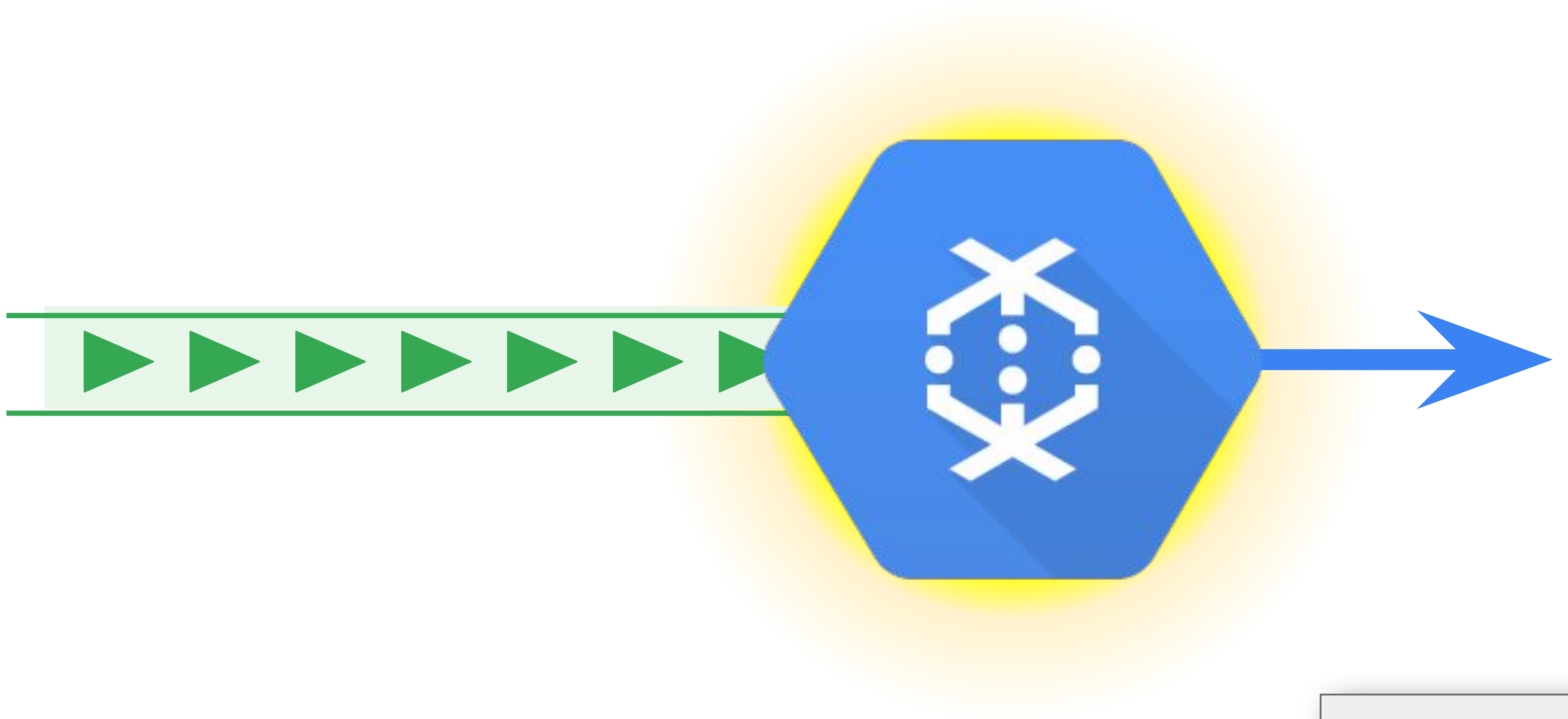
Dataflow Streaming Features

Agenda

- Processing Streaming Data
- Cloud Pub/Sub
- **Cloud Dataflow Streaming Features**
- BigQuery and Bigtable Streaming Features
- Advanced BigQuery Functionality



Streaming features of Cloud Dataflow



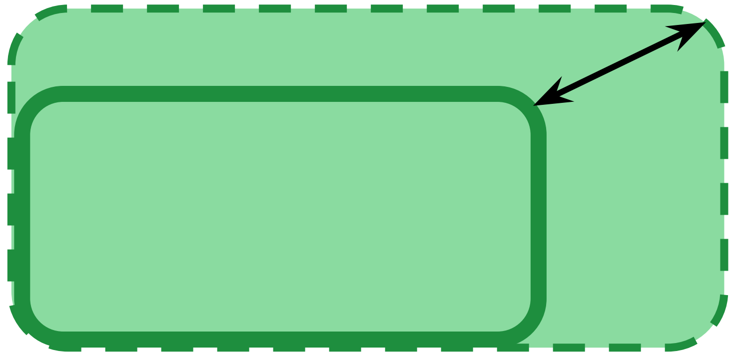
Cloud
Dataflow

Qualities that Cloud Dataflow contributes to Data Engineering solutions:

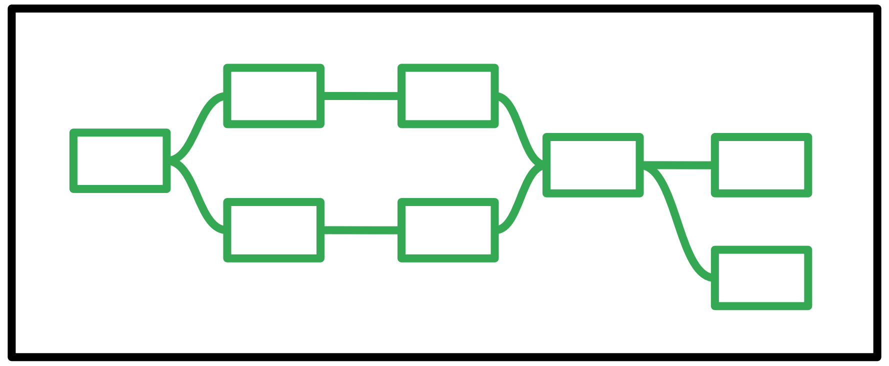
Scalability
Low latency

Continuing from the Data Processing course

Unbounded PCollection

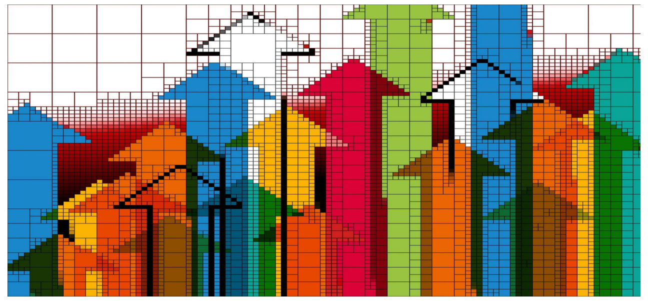


Pipeline



Streaming Jobs

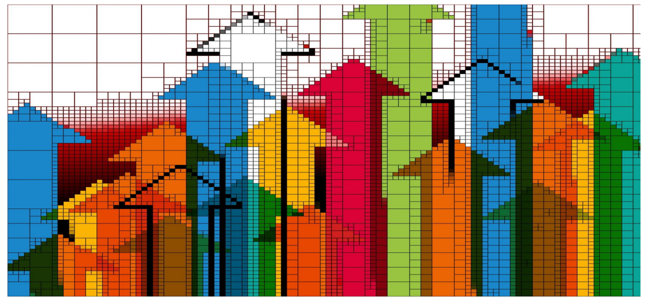
There are challenges with processing streaming data



Scalability

Streaming data generally only grows larger and more frequent

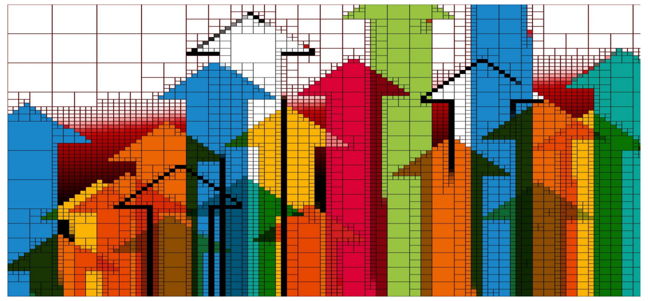
There are challenges with processing streaming data



Scalability
Streaming data generally only grows larger and more frequent

Fault Tolerance
Maintain fault tolerance despite increasing volumes of data

There are challenges with processing streaming data

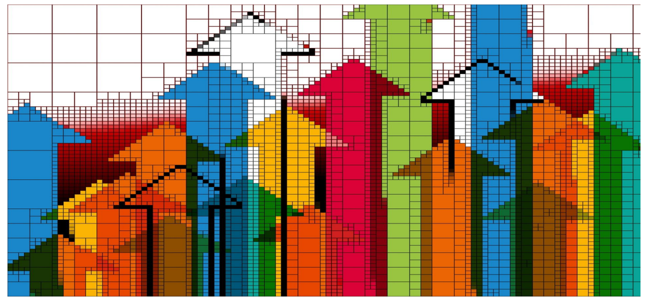


Scalability
Streaming data generally only grows larger and more frequent

Fault Tolerance
Maintain fault tolerance despite increasing volumes of data

Model
Is it streaming or repeated batch?

There are challenges with processing streaming data



Scalability
Streaming data generally only grows larger and more frequent

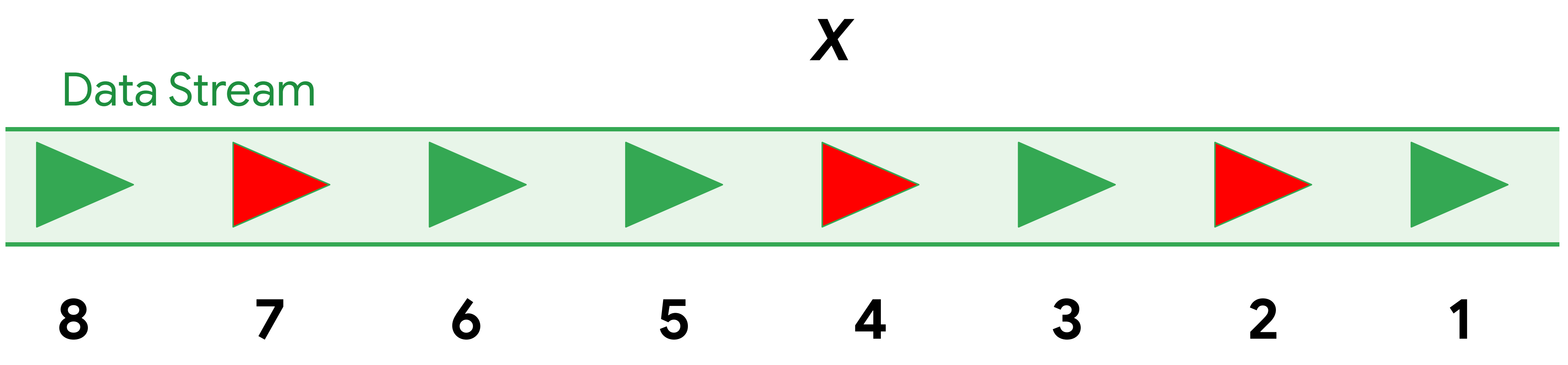
Fault Tolerance
Maintain fault tolerance despite increasing volumes of data

Model
Is it streaming or repeated batch?

Timing
What if data arrives late?

How do you aggregate an unbounded set?

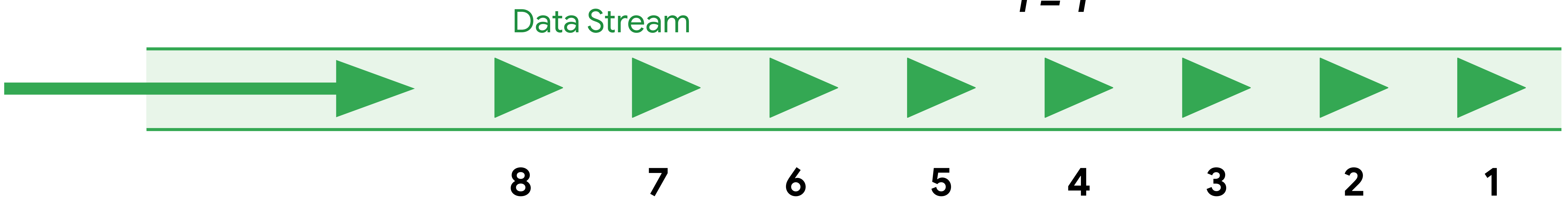
Filtering is straightforward



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

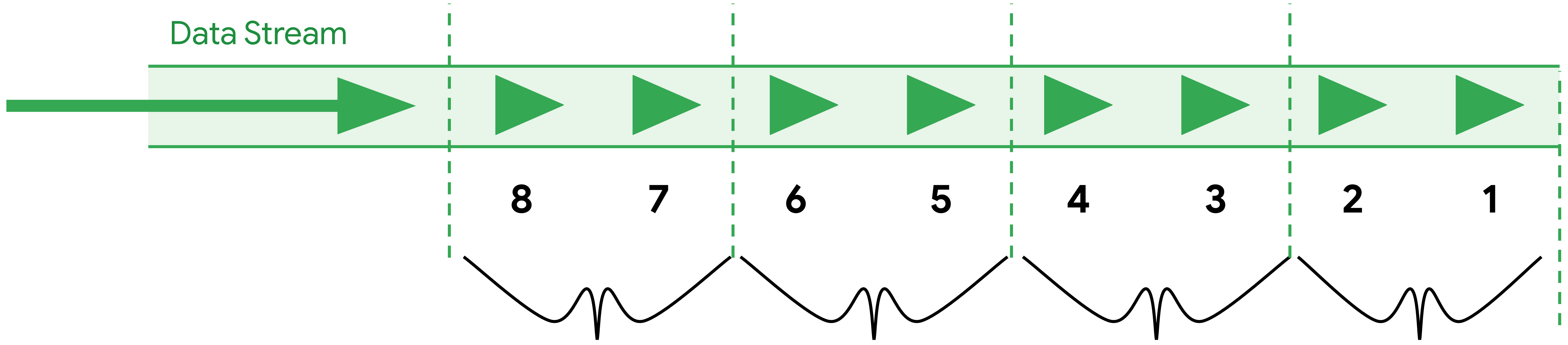
$$\sum_{i=1}^{n} X_i$$

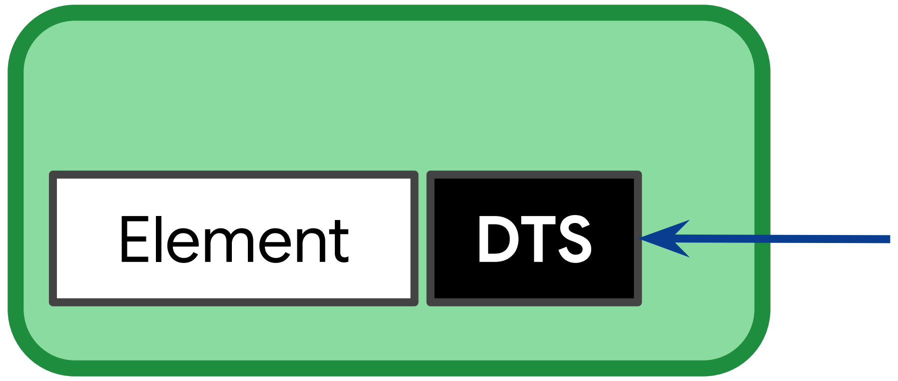
What is the stopping limit?



Google Cloud

Divide the stream into a series of finite windows

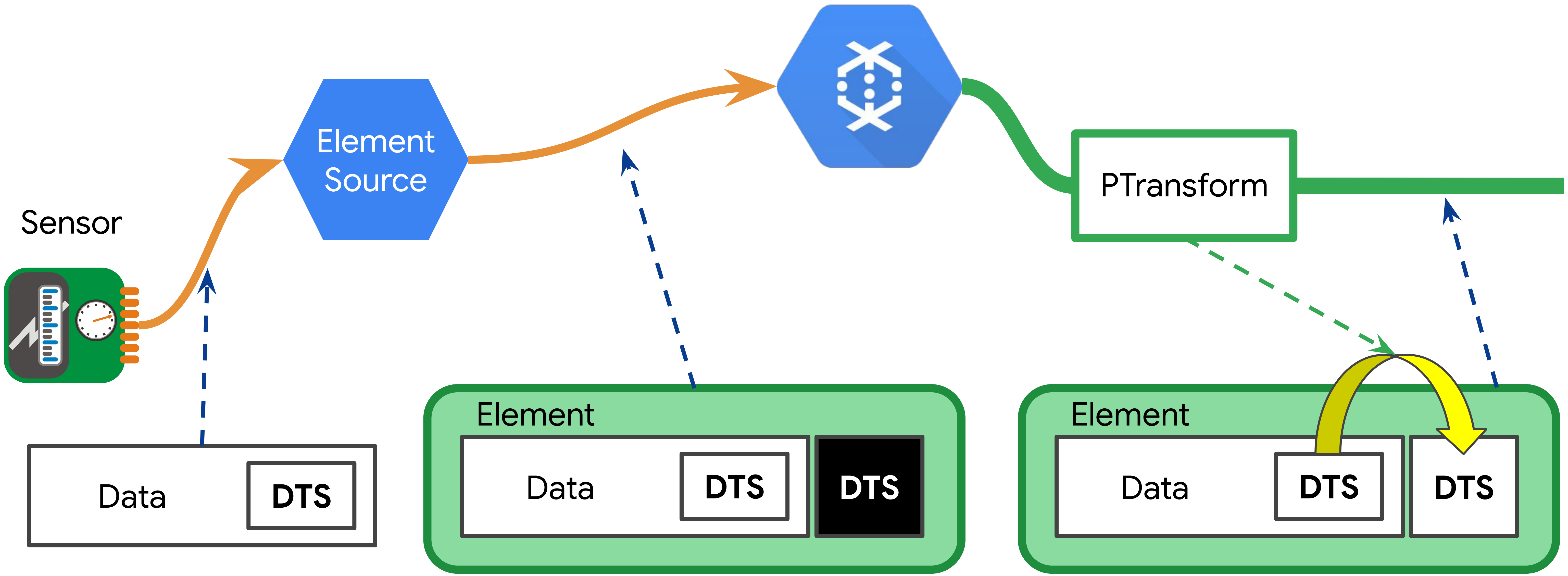




Windows are defined by element date-timestamp (metadata, usually Pub/Sub ingest time)

$$\sum_{i=w}^{n} x_i \qquad \sum_{i=w}^{n} x_i \qquad \sum_{i=w}^{n} x_i$$

Modify the date-timestamp with a PTransform if needed



Code to modify date-timestamp

Python

```
yield beam.window.TimestampedValue(element, unix_timestamp)
```

Java

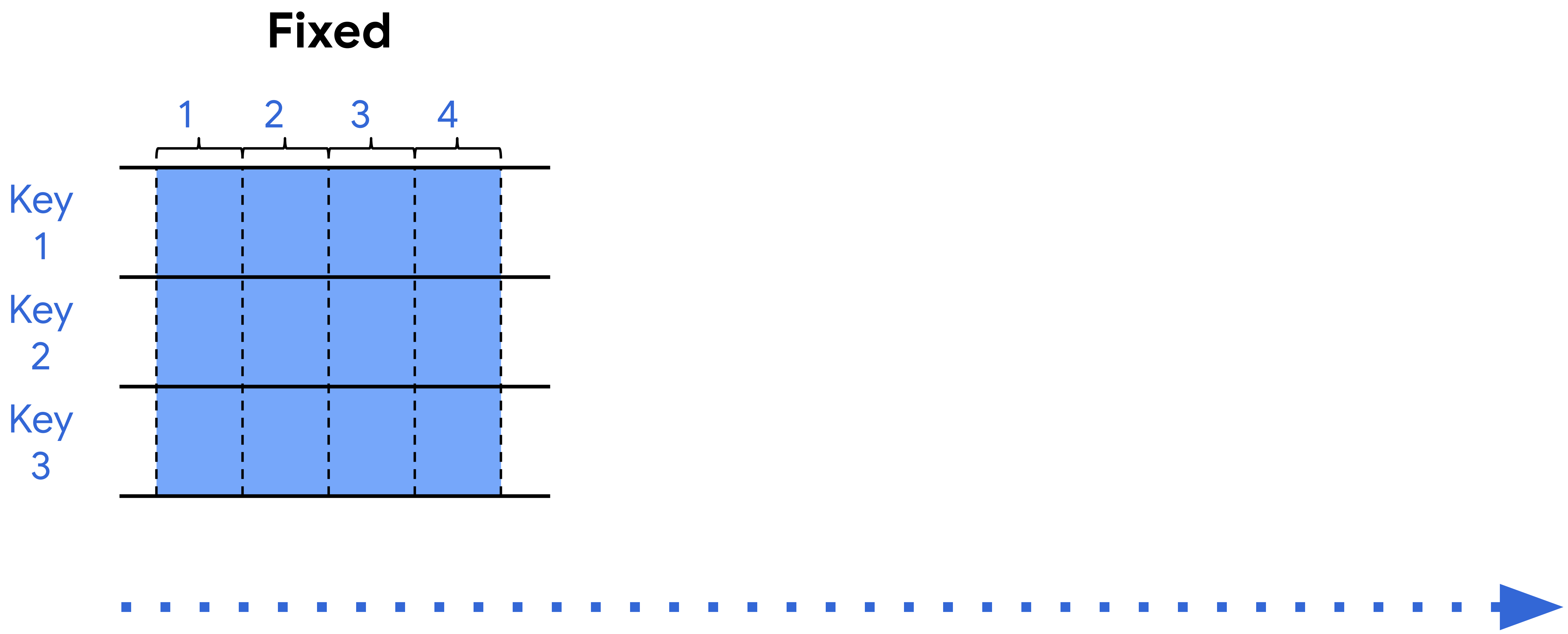
```
c.outputWithTimestamp (element, timestamp);
```

Cloud Dataflow Windowing

Three kinds of windows fit most circumstances

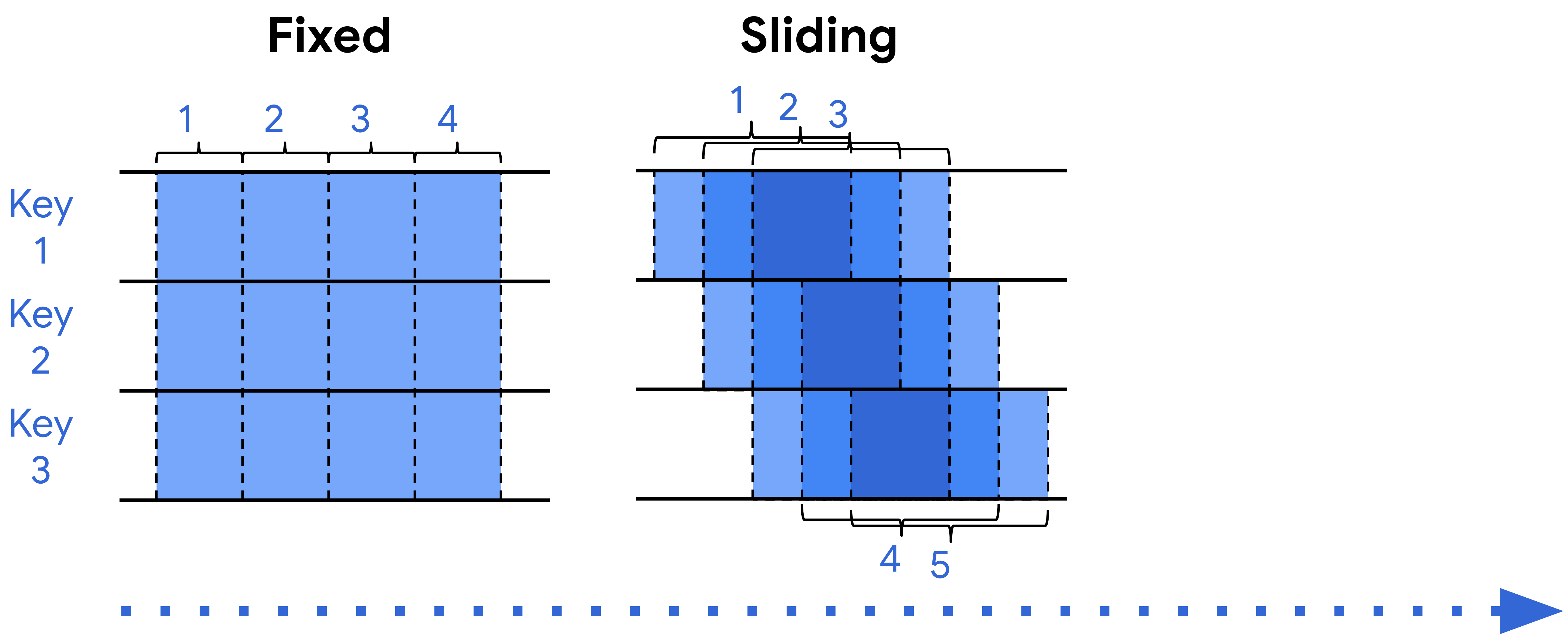
- Fixed
- Sliding
- Sessions

Three kinds of windows fit most circumstances



Windowing divides data into time-based finite chunks

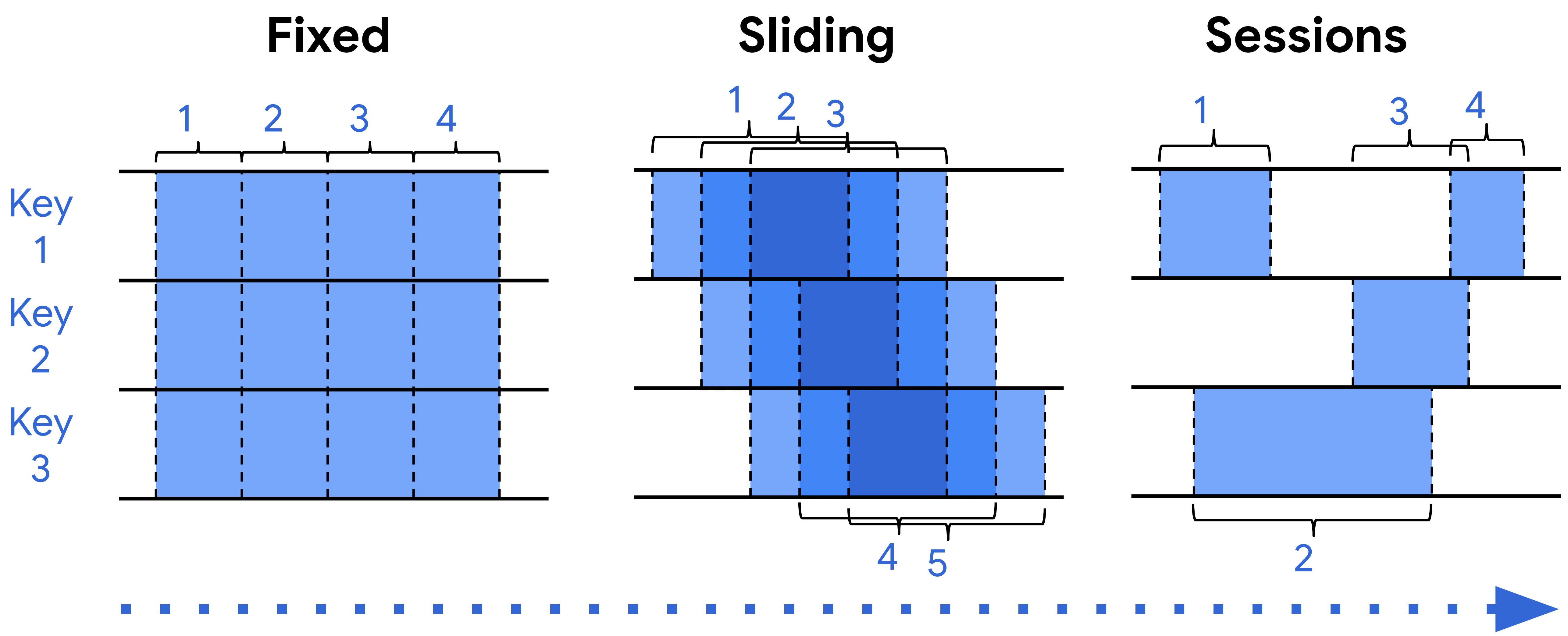
Three kinds of windows fit most circumstances



Windowing divides data into time-based finite chunks

Often required when doing aggregations over unbounded data

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

Remember:

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

Fixed-time windows

Python

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

Sliding time windows

Python

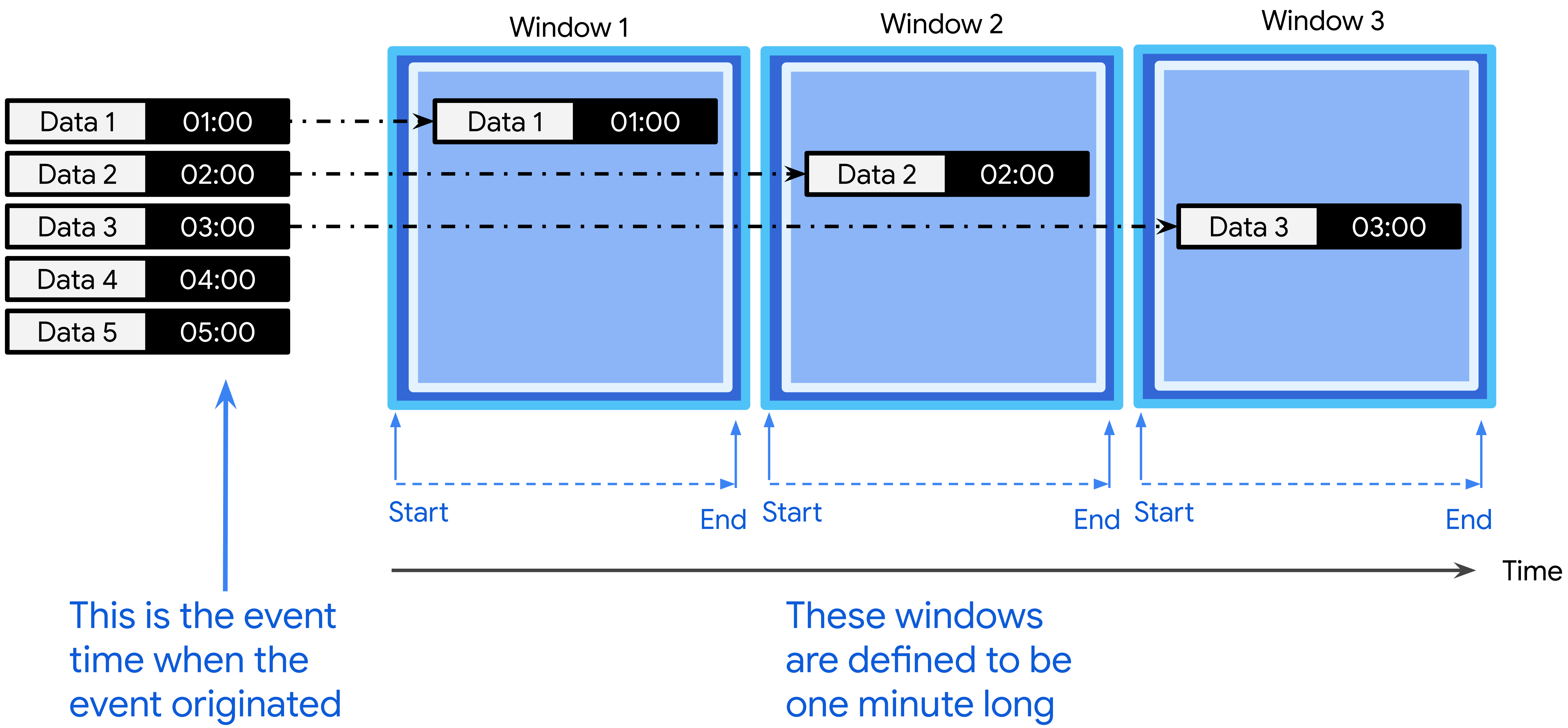
```
from apache_beam import window
sliding_windowed_items = (
    items | 'window' >> beam.WindowInto(window.SlidingWindows(30, 5)))
```

Session windows

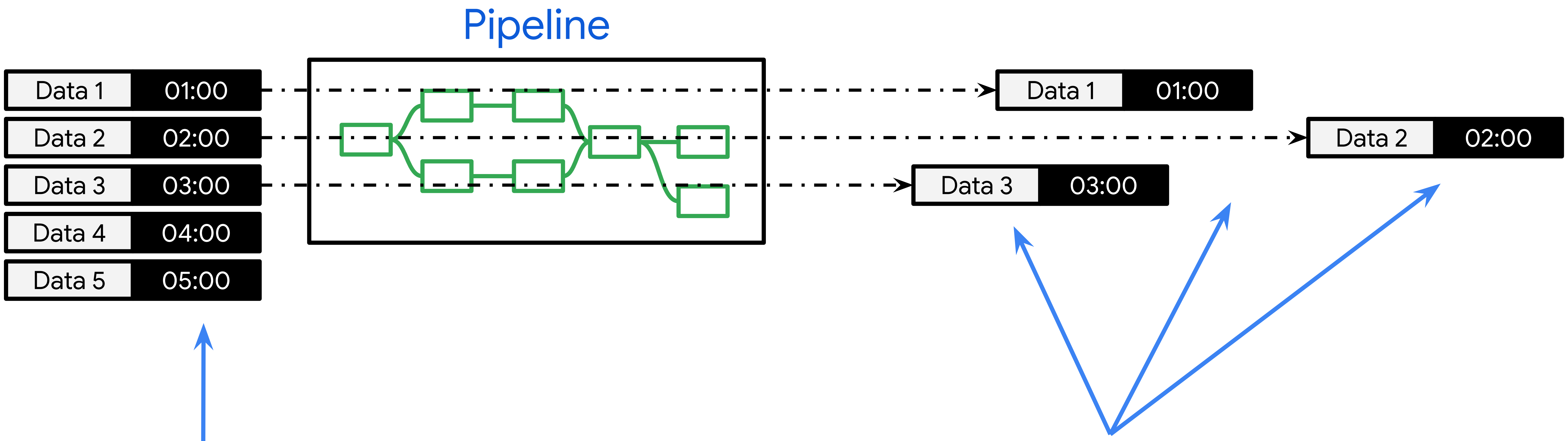
Python

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

Windowing by time if there is no latency



Pipeline processing can introduce latency



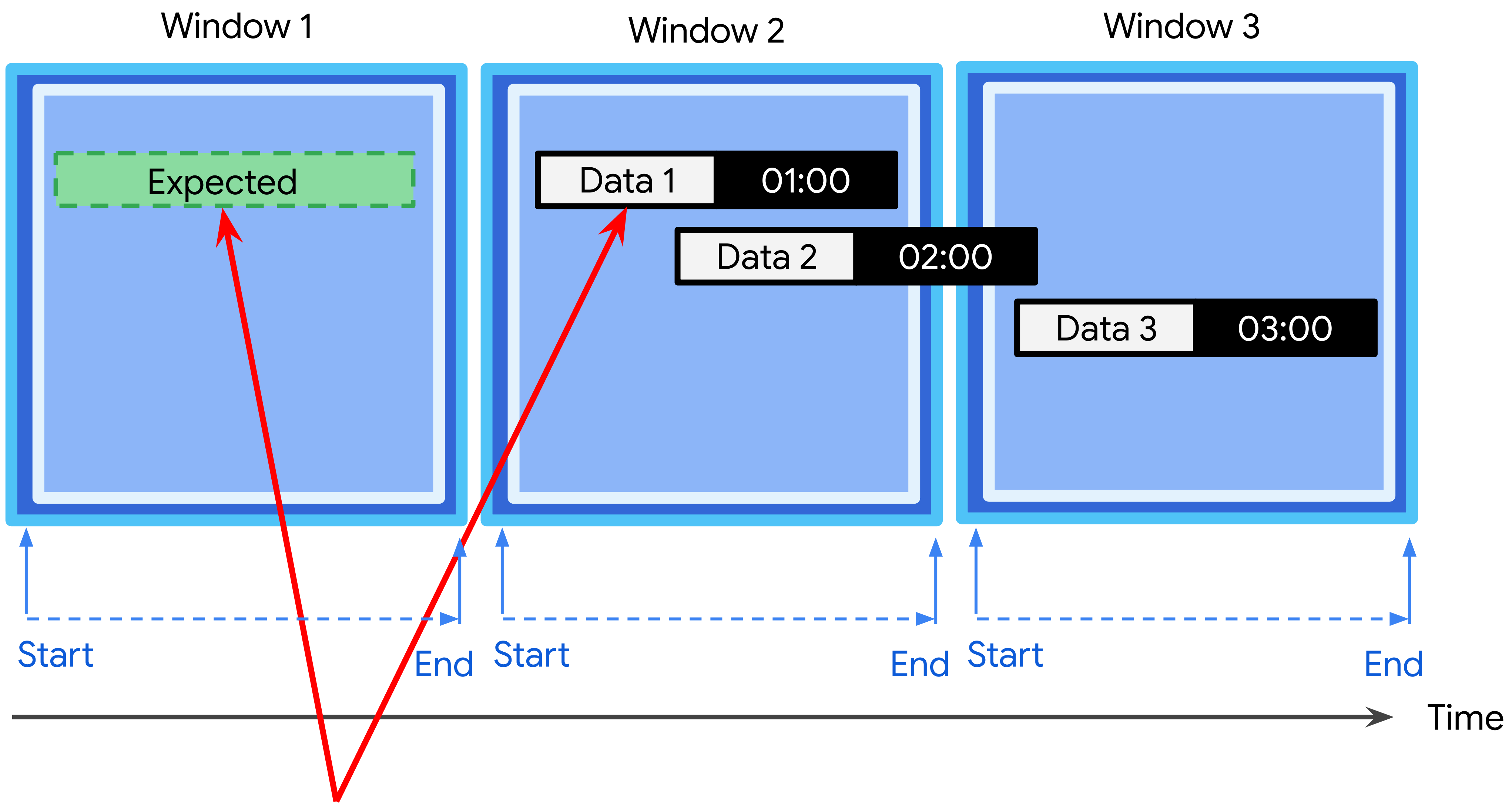
This is the event time when the event originated

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.

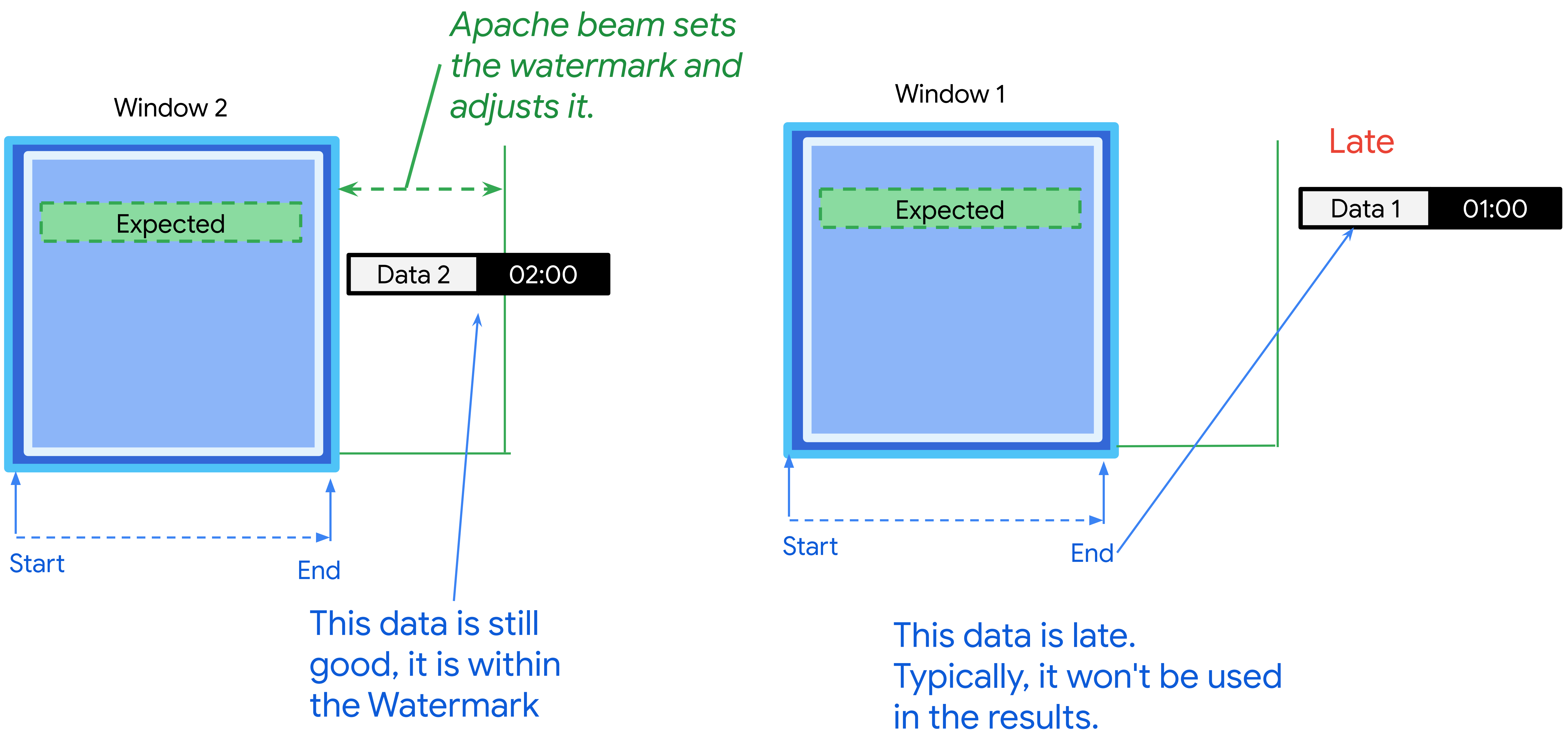
How should Cloud Dataflow deal with this situation?

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.



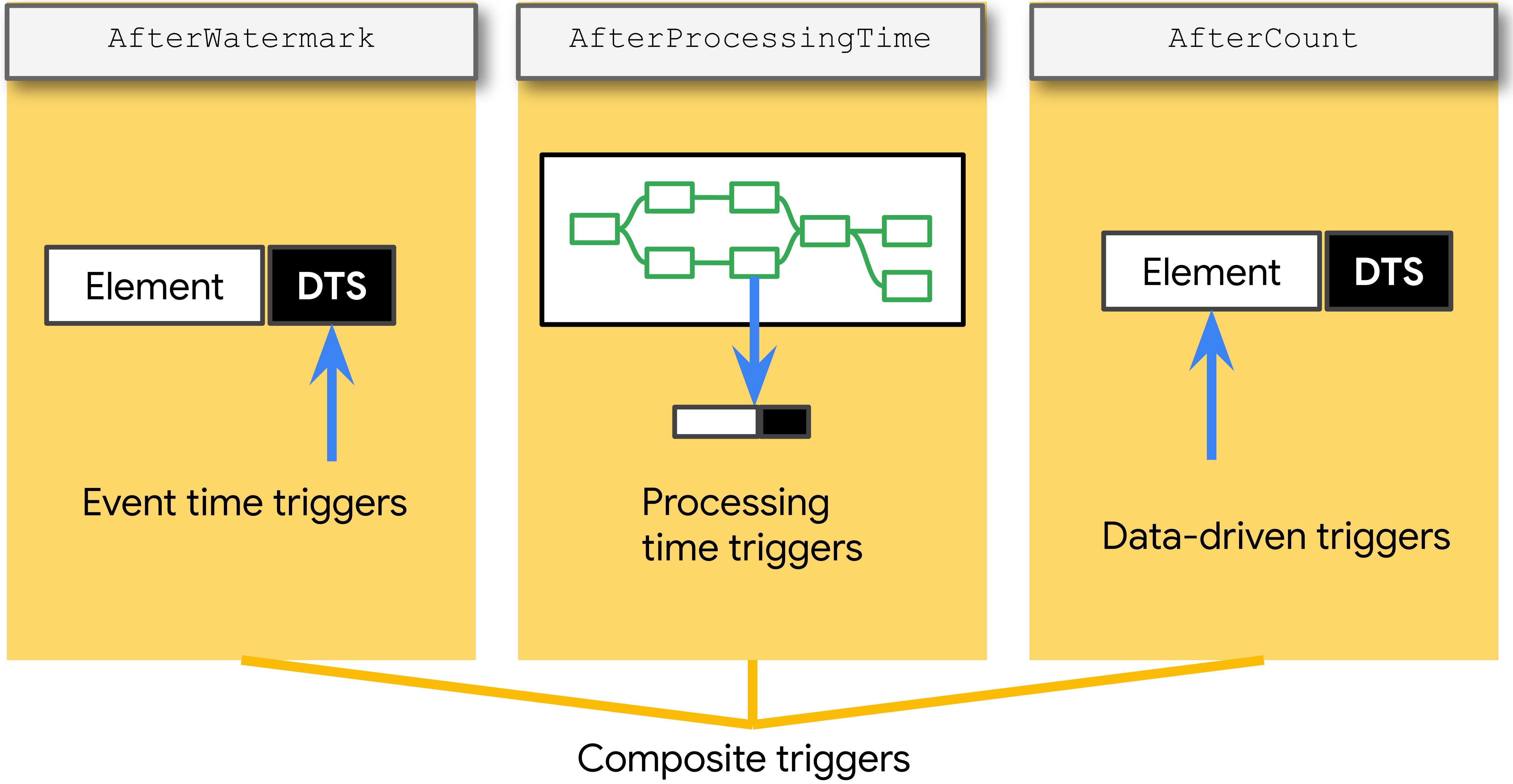
The difference in time from when data was expected to when it actually arrived is called the **lag time**.

Watermarks provide flexibility for a little lag time



Google Cloud

The default is to trigger at the watermark, but we can also add custom trigger(s)



Google Cloud

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
```

```
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
```

You can allow late data past the watermark

Allowing Late Data

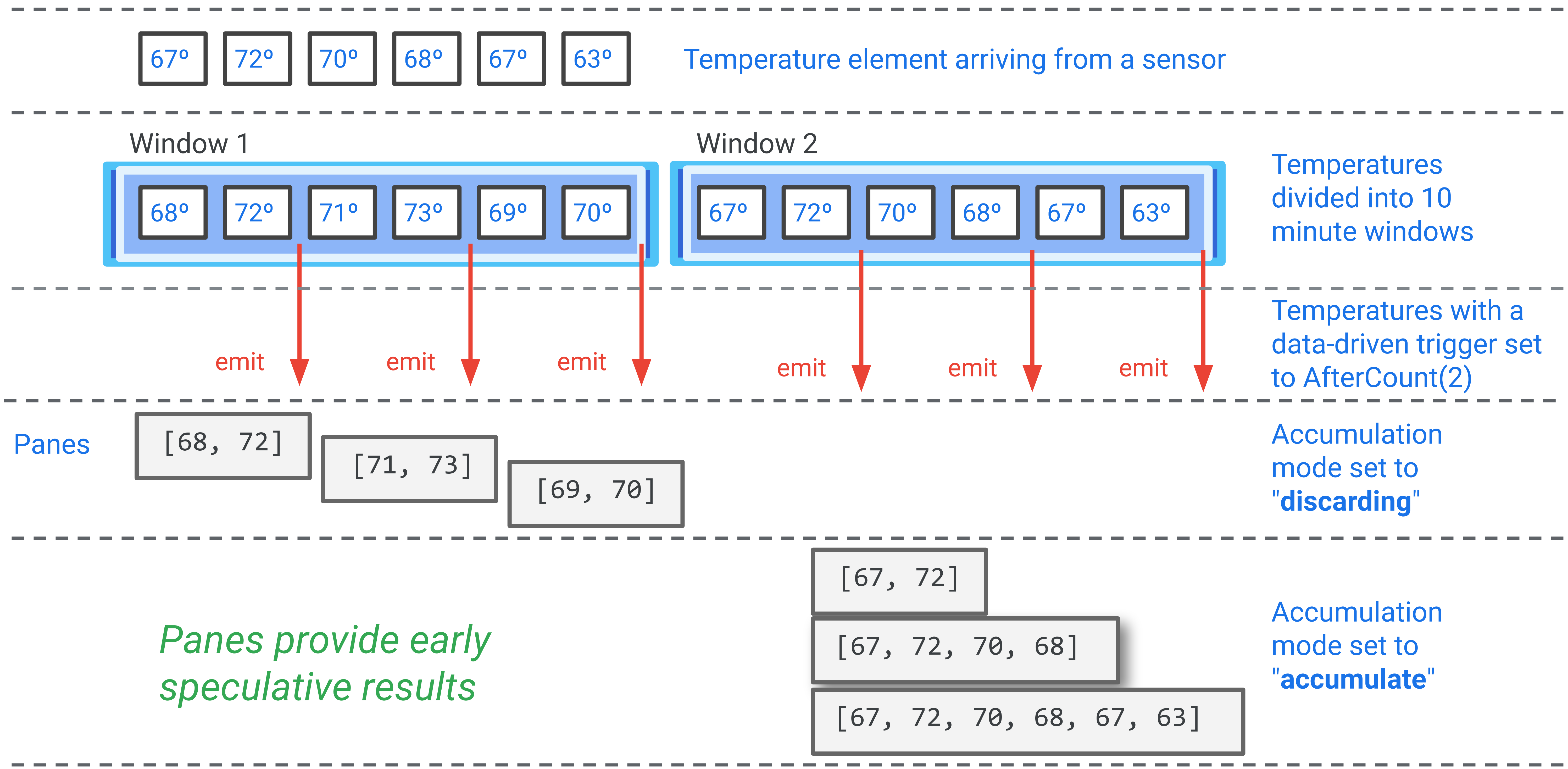
Java

```
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



Google Cloud

Lab

Streaming Data Pipelines

Objectives

- Launch Dataflow and run a Dataflow job
- Understand how data elements flow through the transformations of a Dataflow pipeline
- Connect Dataflow to Pub/Sub and BigQuery
- Observe and understand how Dataflow autoscaling adjusts compute resources to process input data optimally
- Learn where to find logging information created by Dataflow
- Explore metrics and create alerts and dashboards with Cloud Monitoring