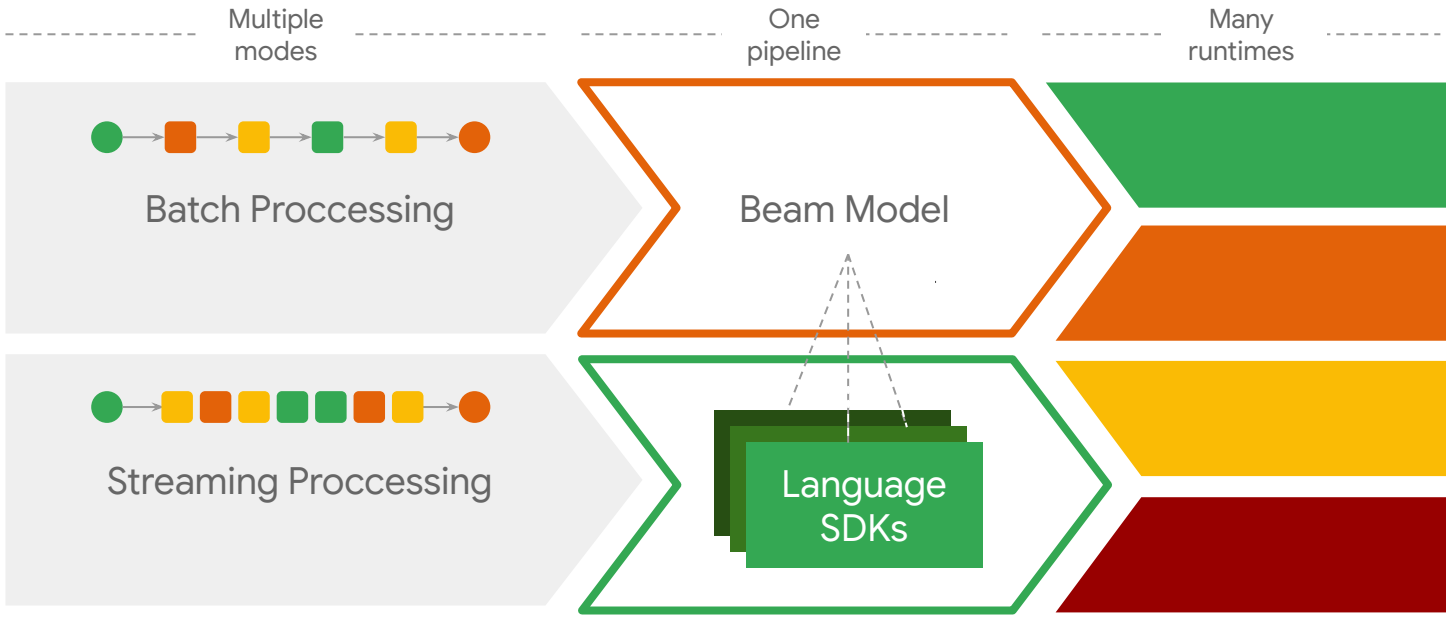
Apache Beam Overview

Miren Esnaola

What is Apache Beam?

Apache Beam is an open source, **unified model** for defining both **batch** and **streaming** data-parallel processing pipelines. Using the open source Beam SDKs, you build a program defining the pipeline, that then is executed by one of Beam's supported distributed processing backends.


Multiple modes
One pipeline
Many runtimes
Batch Proccessing
Streaming Proccessing
Beam Model
Language SDKs

SDKs

Pipeline Runners

Dataflow

Flink

Pipeline

A **`Pipeline`** is a Directed Acyclic Graph (DAG) of data transformations applied to one or more collections of data. It might include multiple input sources and output sinks and its operations (`PTransform`s) can both read and output `PCollection`s.

Pipeline

Creating a `Pipeline`

```
import apache_beam as beam
from apache_beam.options.pipeline_options import PipelineOptions

with beam.Pipeline(options=PipelineOptions()) as p:

   pass  # build your pipeline here
```

Pipeline

Setting `PipelineOptions` from command-line arguments

```
from apache_beam.options.pipeline_options import PipelineOptions
options = PipelineOptions(flags=argv)
```

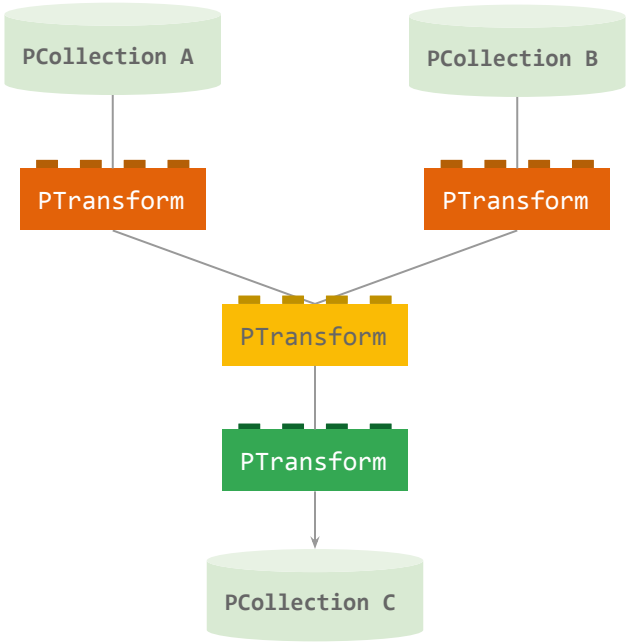
This interprets command-line arguments that follow the format: `--<option>=<value>`

Pipeline

Creating custom options

```
from apache_beam.options.pipeline_options import PipelineOptions

class MyOptions(PipelineOptions):

    @classmethod
    def _add_argparse_args(cls, parser):
        parser.add_argument(
            '--input',
            help='Input for the pipeline',
            default='gs://my-bucket/input')
        parser.add_argument(
            '--output',
            help='Output for the pipeline',
            default='gs://my-bucket/output')

with beam.Pipeline(options=MyOptions()) as p:

    pass
```

Pipeline


PCollection A
PCollection B
PTransform
PTransform
PTransform
PTransform
PCollection C

PCollection

A **`PCollection`** is an immutable collection of values. It can contain a bounded or unbounded number of elements. Beam transforms use `PCollection` objects as **inputs** and **outputs**.

PCollection

A `PCollection` is owned by the specific `Pipeline` object for which it was created. Multiple pipelines cannot share one.

Elements in a `PCollection` may be of **any type**, but all must be of the **same type**.

A `PCollection` is **immutable**. A transform might process each element in it and generate a new `PCollection`, but it does modify the original one.

`PCollections` can be **bounded** (batch) or **unbounded** (streaming) in size.

`PCollections` **do not support random access** to elements.

Each element in a `PCollection` has an associated **intrinsic timestamp**. Timestamps can be manually assigned to the elements if the source doesn't do it for you.

Beam needs to encode elements as a byte strings to support distributed processing. The SDKs include **built-in encoding mechanisms** for common types as well as support for specifying **custom encodings**.

In many cases, the element type in a `PCollection` has a an associated **schema** (e.g., JSON, Protocol Buffer, Avro, database records)

PTransform

A `PTransform` is an **operation** in a pipeline. Developers provide processing logic as a function object applied to each element of one or more input `PCollection`(s). Depending on the runner, multiple workers across a cluster may execute the code in parallel and generate the output elements ultimately added to the final `PCollection` produced by the transform.

PTransform

To invoke a transform, you must **apply** it to the input `PCollection`.

Transforms can be **nested** to form composite transforms.

Transforms can be **chained** to be applied sequentially to an input `PCollection`.

Beam SDKs contain generic **core transforms** and **pre-written composite transforms** combining one or more of the core ones.

A transform does **not alter** the input collection.

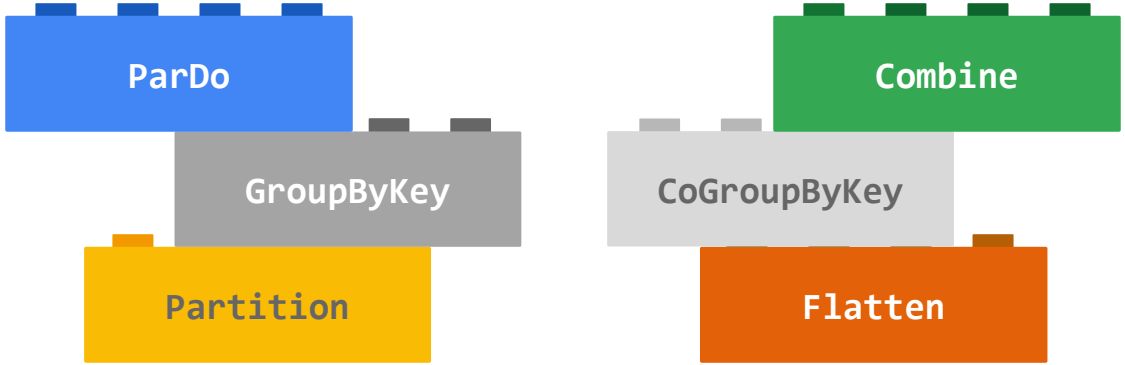
PTransform

Applying a transform

```
[Output PCollection] = [Input PCollection] | [Transform]
```

PTransform

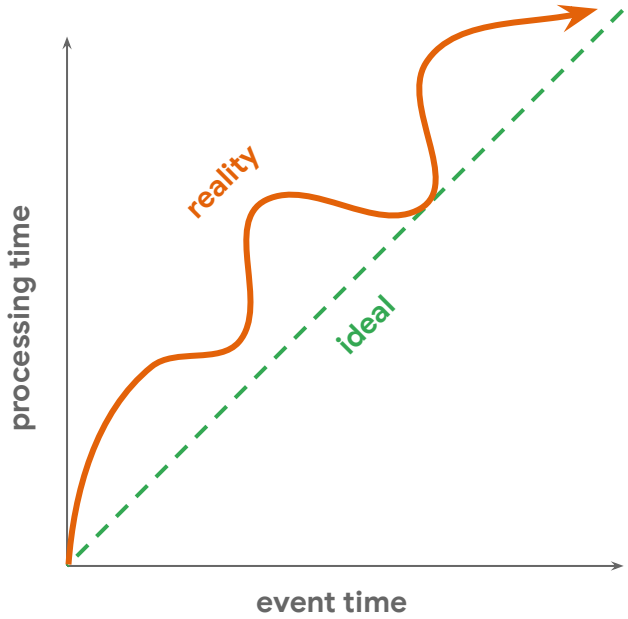
Core transforms



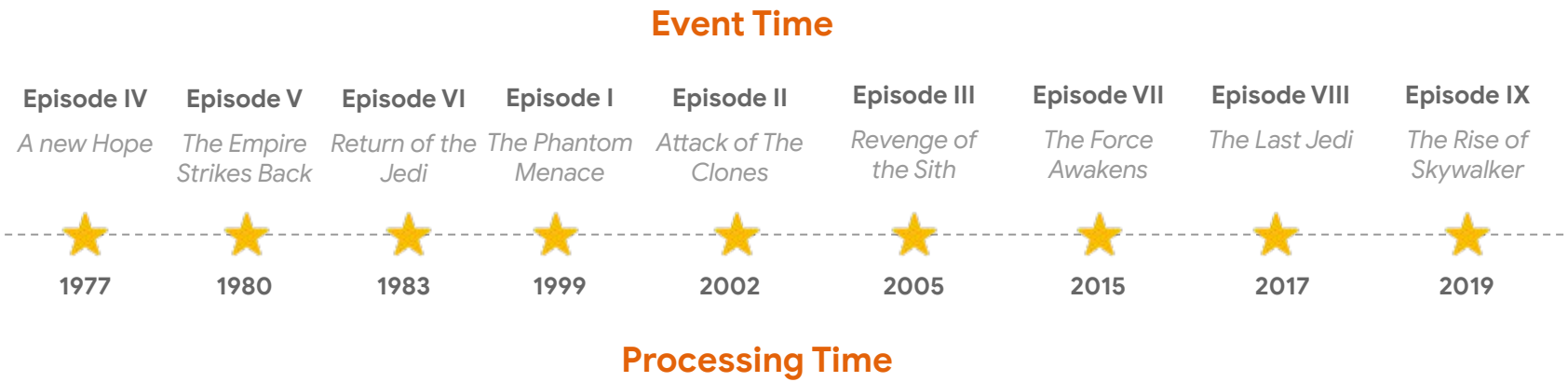
Event time vs. processing time

In any streaming data processing system:

- There is a certain amount of lag between:
  - The **event time**, when a data event occurs (determined by the timestamp on the data element itself).
  - The **processing time**, when a data element gets processed at any stage in a pipeline (determined by the clock on the processing system).
- There are no guarantees that data events will appear in a pipeline in the same order that they were generated.



Event time vs. processing time



Source: *Introduction to Apache Flink by Ellen Friedman, Kostas Tzoumas*

Event time vs. processing time

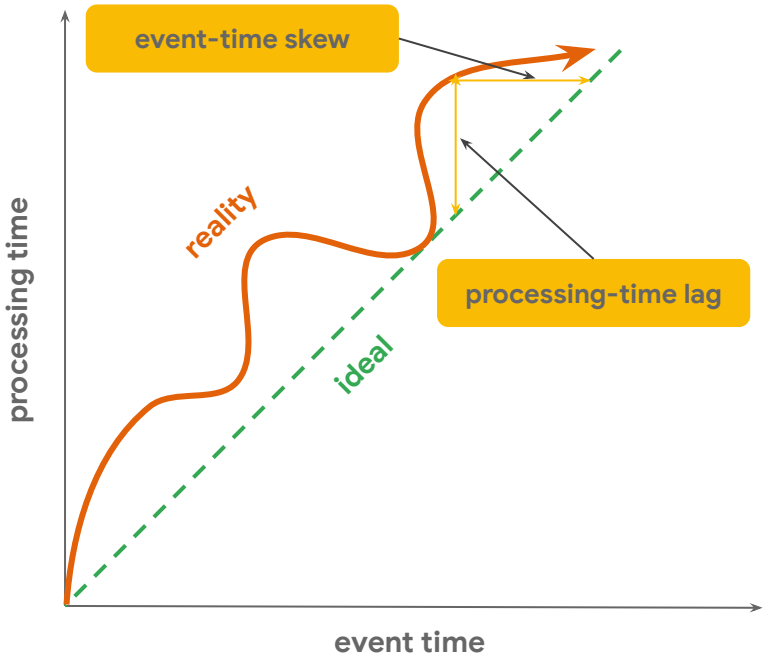
Processing-time lag

It measures the delay observed between the time when the event occurred and the time when it was processed

Event-time skew

It measures how far behind the ideal in event time the pipeline currently is.

processing-time lag = event-time skew



Window

A **`Window`** subdivides a `PCollection` according to the timestamps of its individual elements, which is especially useful for unbounded `PCollection`s, as it allows operating on sub-groups of elements.

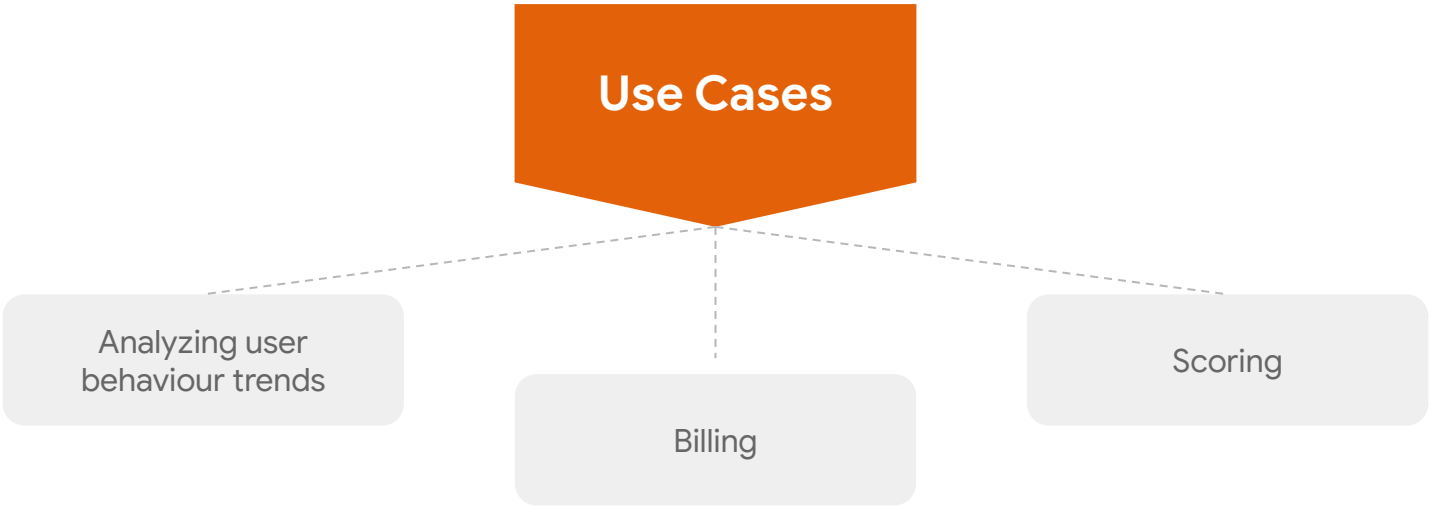
Processing-time windowing

Inferring information about a data source ***as it is observed***

Processing-time windowing

- No need to worry about shuffling data within time, just buffer elements as they arrive until the window closes.
- Easy to judge window completeness.

Event-time windowing



Inferring information about a data source ***as it occurs***

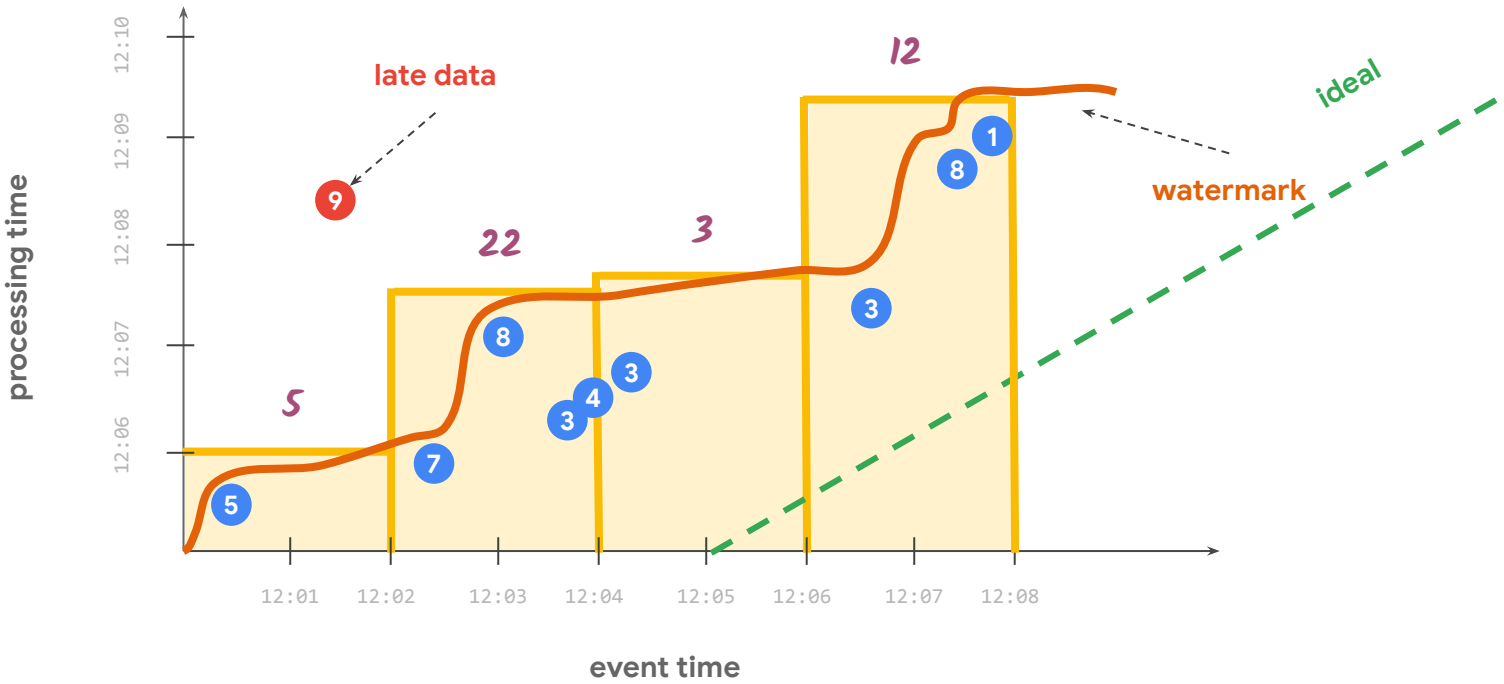
Event-time windowing

- More buffering of data occurs, as windows must often live longer in processing time than their actual length.
- Hard to judge window completeness, we can only estimate.

Watermark and late data

A **watermark** is the system's notion of when all data in a certain window can be expected to have arrived in the pipeline. Once the watermark progresses past the end of a window, any further element arriving with a timestamp in that window is considered **late data**.

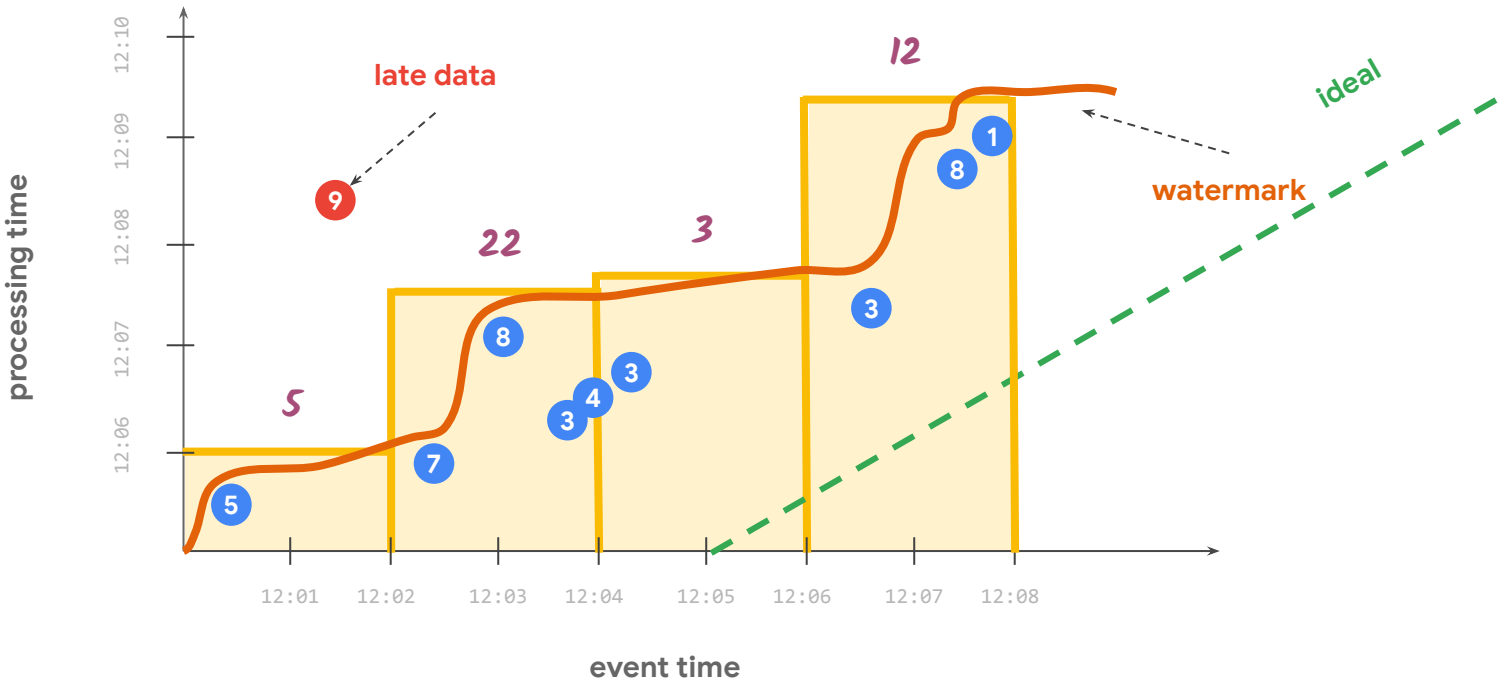
Watermark and late data



Trigger

A `Trigger` determines when to emit the aggregated results of each window.

Watermark and late data



Window

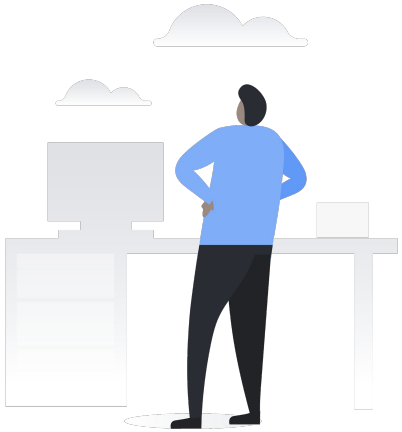
How is my data grouped?

Watermark

When is my data complete?

Trigger

When should I produce results?



Apache Beam defaults

- All elements of a `PCollection`, even in the case of unbounded ones, are assigned to a single global window.
- Aggregated results are emitted when it estimates that all the data has arrived.
- Late data is discarded.

Thank you!

Questions?