Talk Python to Me

Stream Processing in Your Favourite Language with Beam on Flink

Apache Beam

Apache Flink

Based on work and slides by Frances Perry, Tyler Akidau, Kenneth Knowles & Sourabh Bajaj

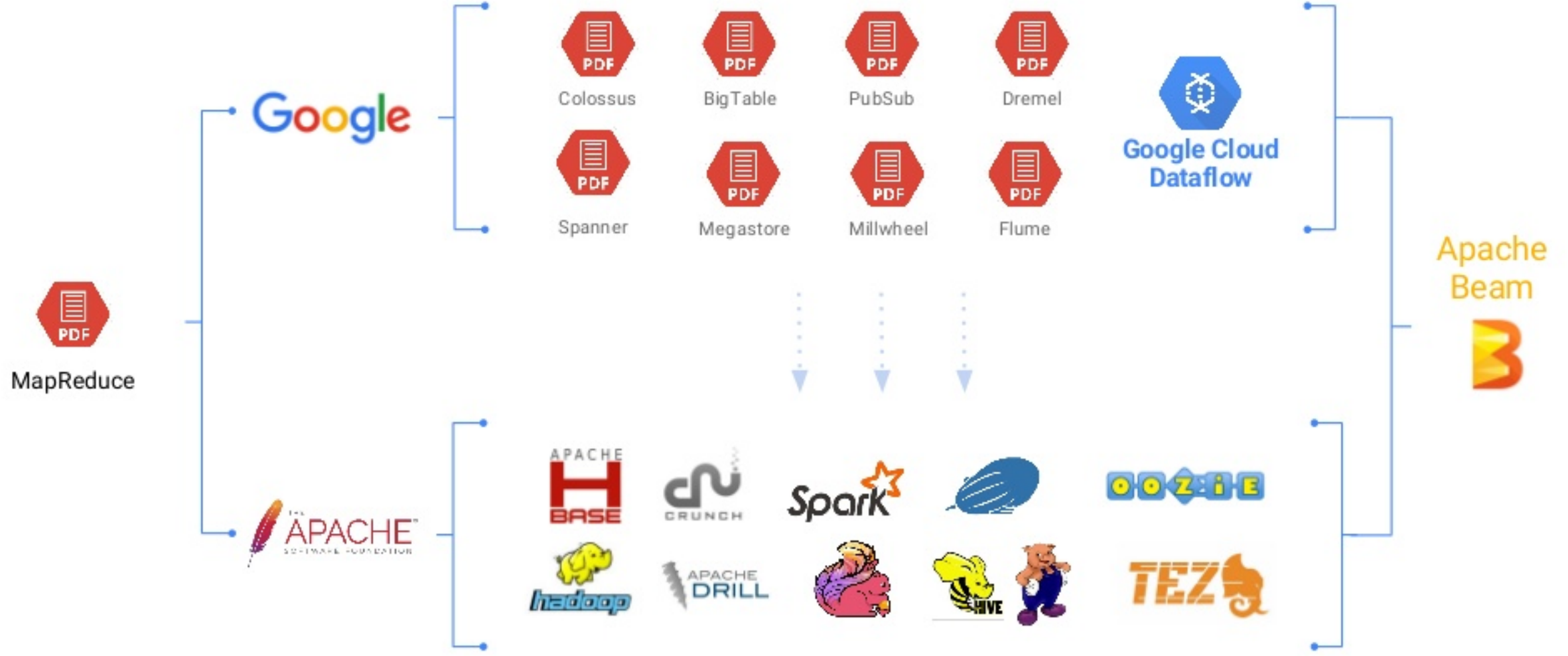
Slides by Aljoscha Krettek, September 2017, Flink Forward 2017

Agenda

1. What is Beam?
2. The Beam Portability APIs (Fn / Pipeline)
3. Executing Pythonic Beam Jobs on Flink
4. The Future

What is Beam?

The Evolution of Apache Beam

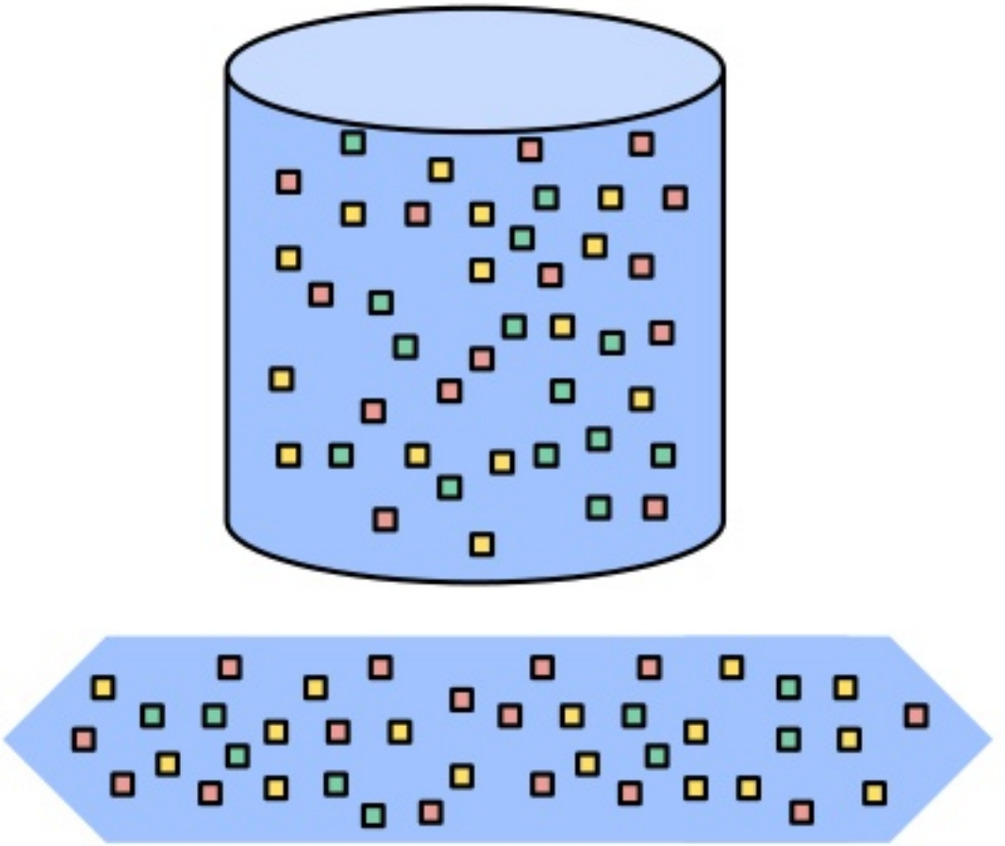


Beam Model: Generations Beyond MapReduce

Improved abstractions let you focus on your application logic

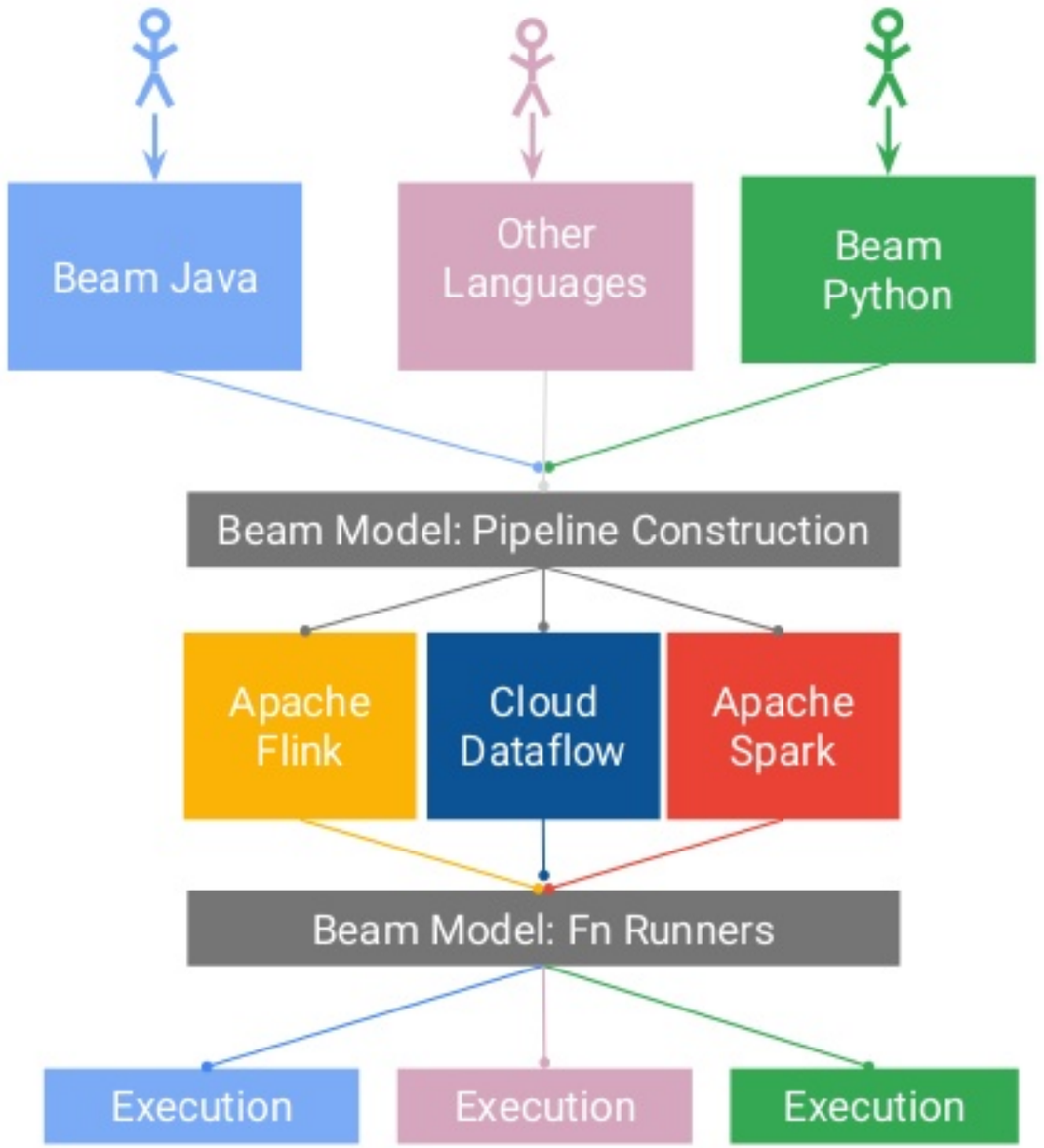
Batch and stream processing are *both* first-class citizens -- no need to choose.

Clearly separates event time from processing time.

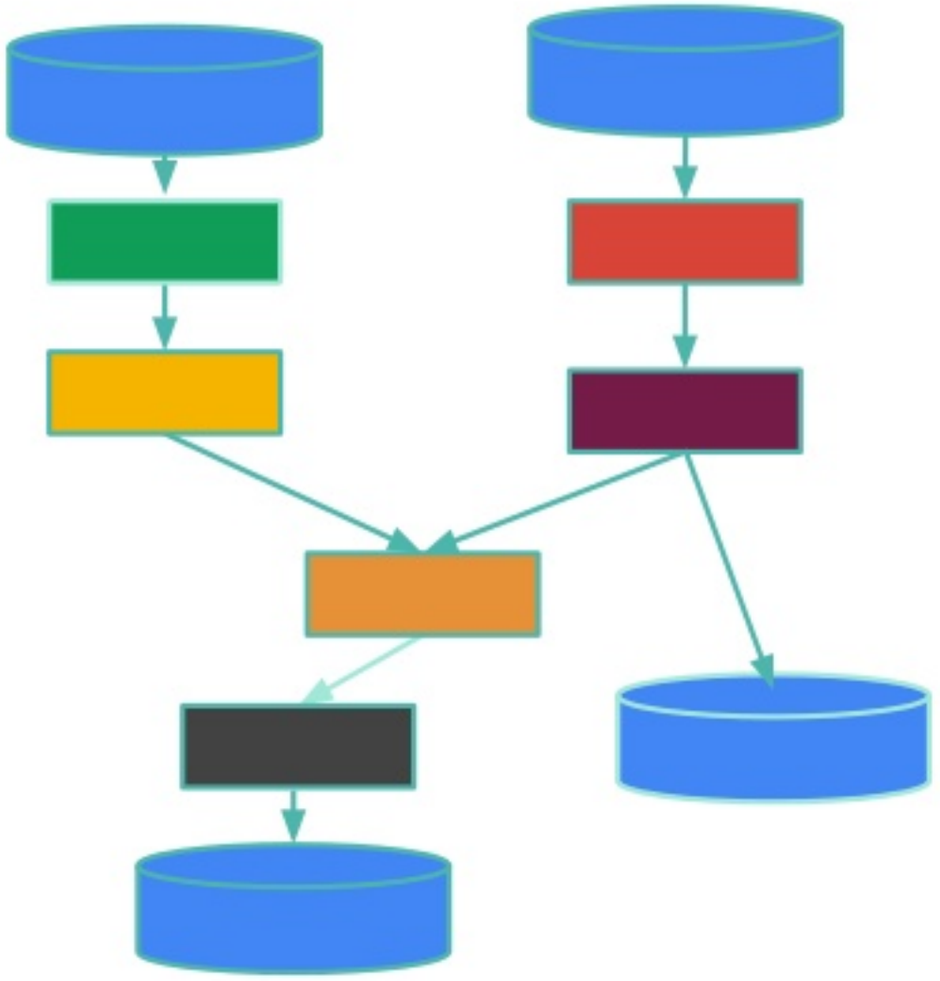


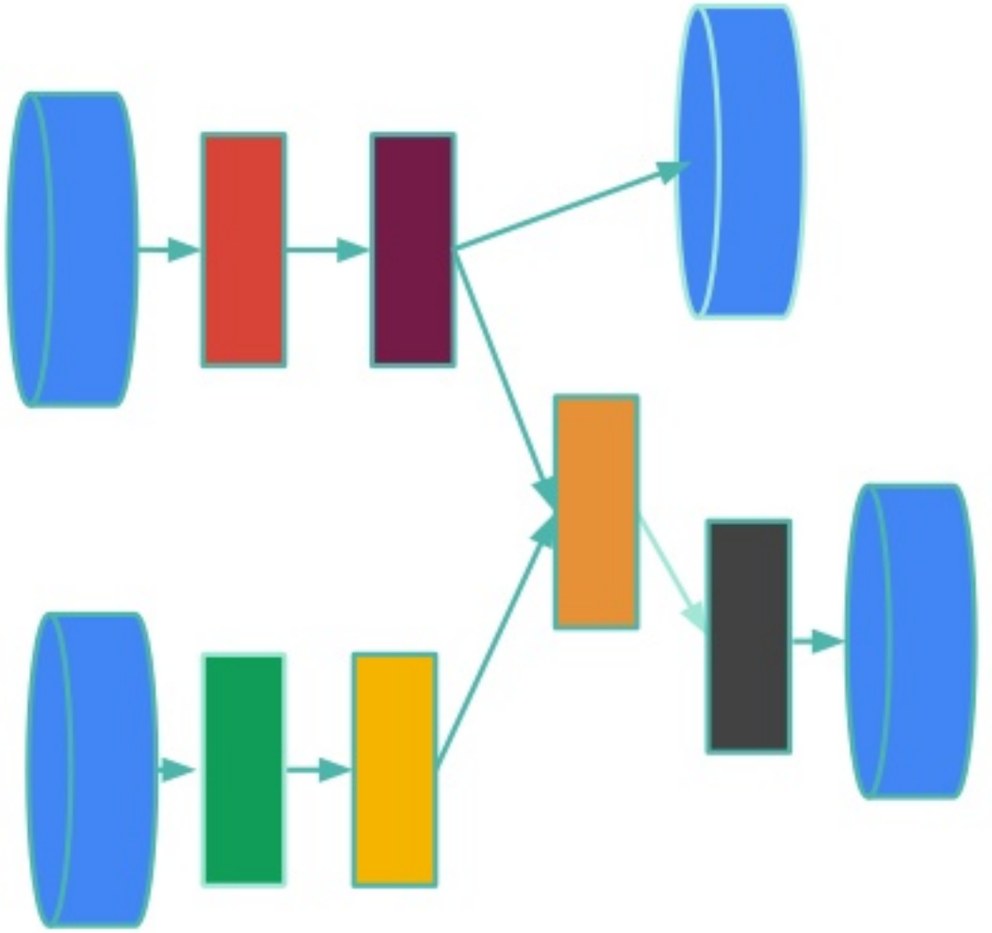
The Apache Beam Vision

1. **End users:** who want to write pipelines in a language that's familiar.
2. **SDK writers:** who want to make Beam concepts available in new languages.
3. **Runner writers:** who have a distributed processing environment and want to support Beam pipelines



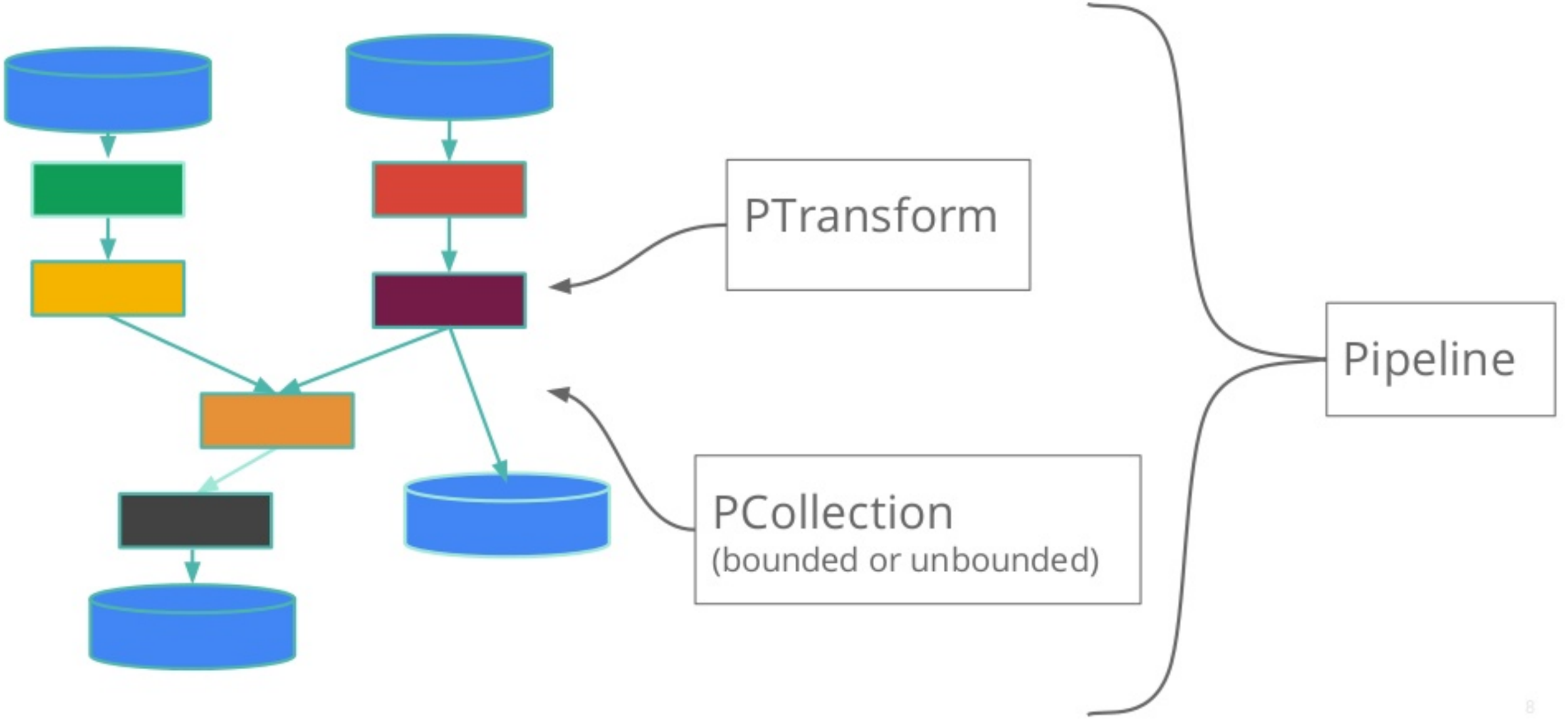
The Beam Model





(Flink draws it more like this)

The Beam Model



Beam Model: Asking the Right Questions

What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

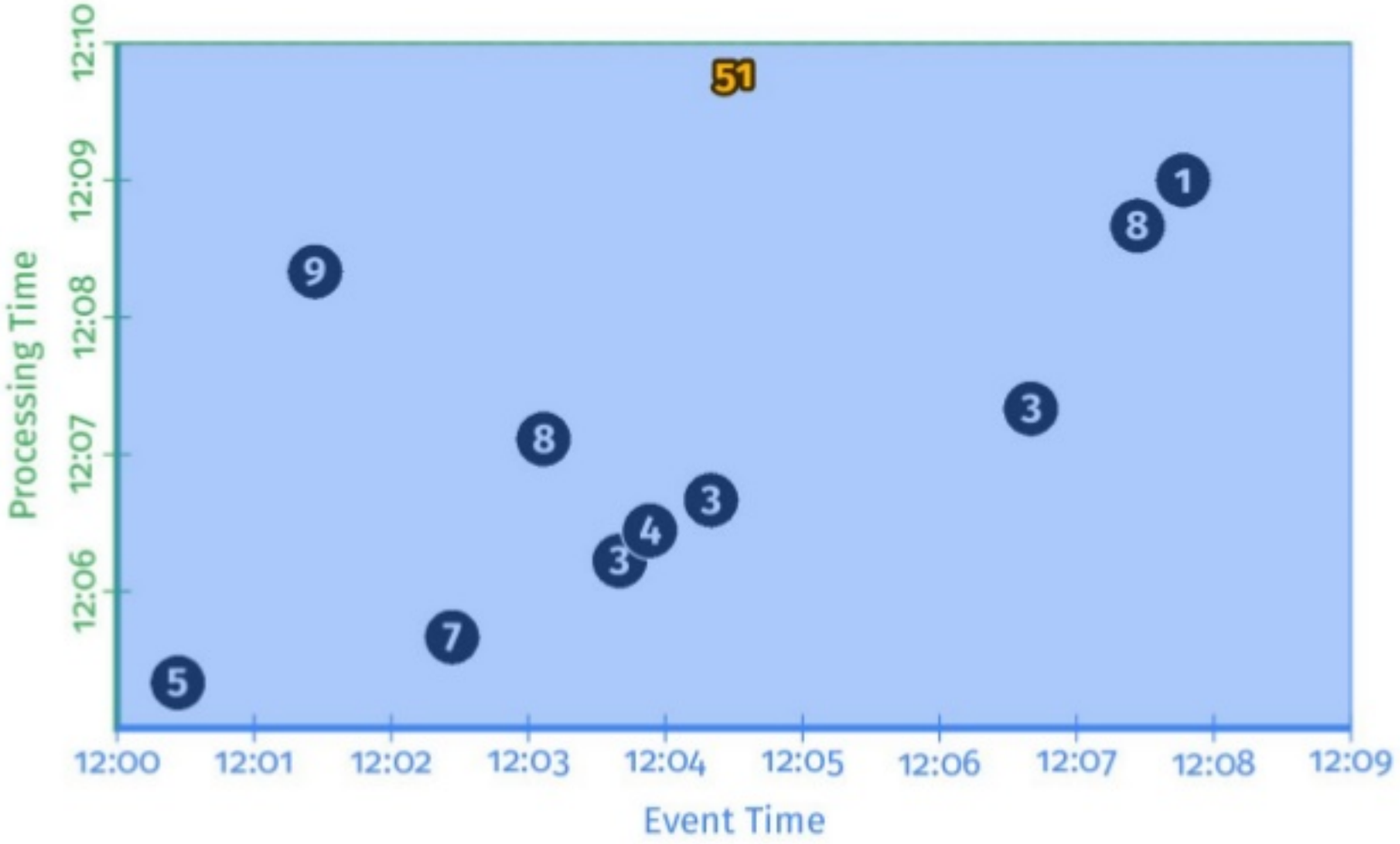
How do refinements of results relate?

The Beam Model: **What** is Being Computed?

```
PCollection<KV<String, Integer>> scores = input
    .apply(Sum.integersPerKey());
```

```
scores= (input
    | Sum.integersPerKey())
```

The Beam Model: **What** is Being Computed?

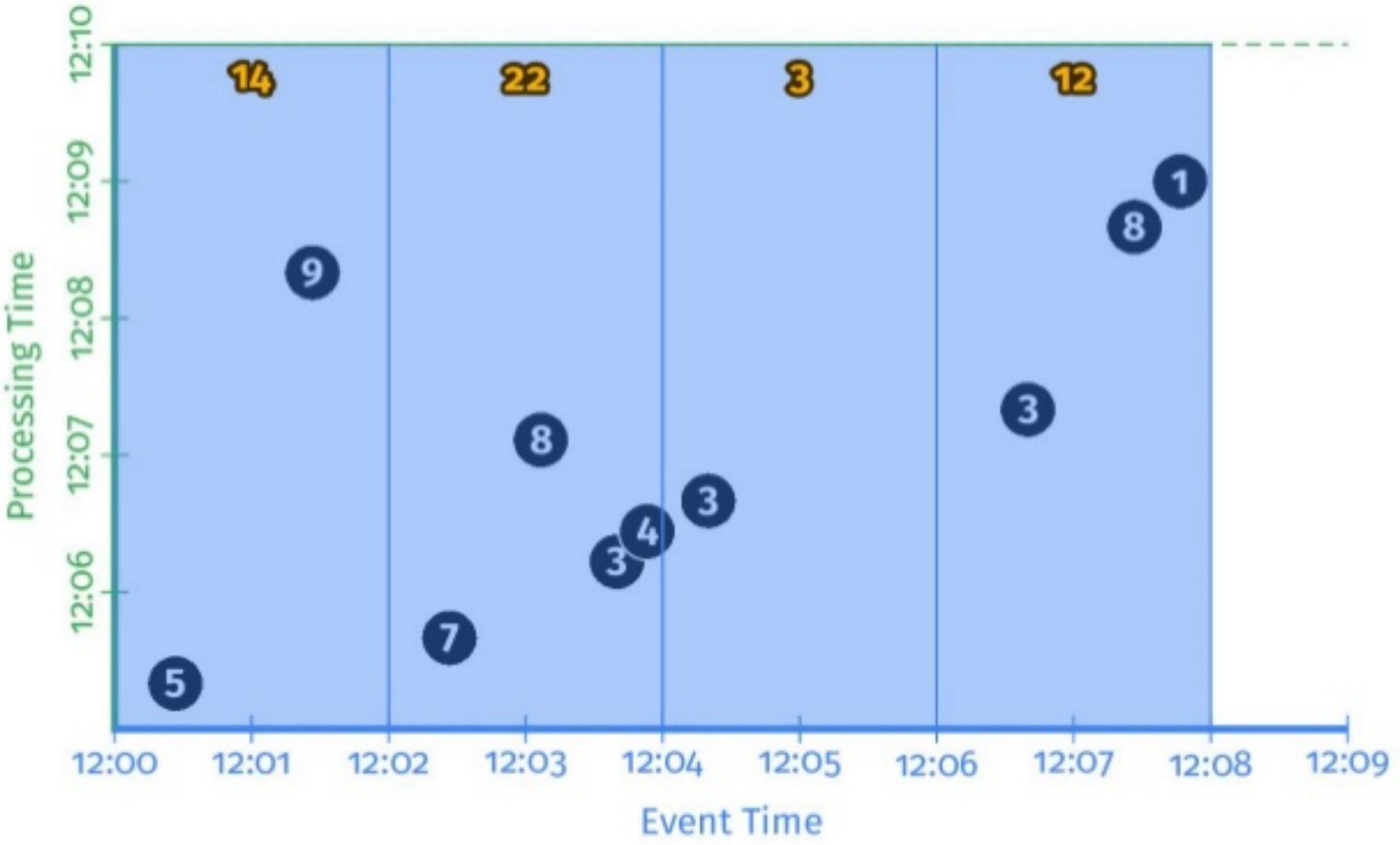


The Beam Model: **Where** in Event Time?

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))))
    .apply(Sum.integersPerKey());
```

```
scores= (input
    | beam.WindowInto(FixedWindows(2 * 60))
    | Sum.integersPerKey())
```

The Beam Model: **Where** in Event Time?

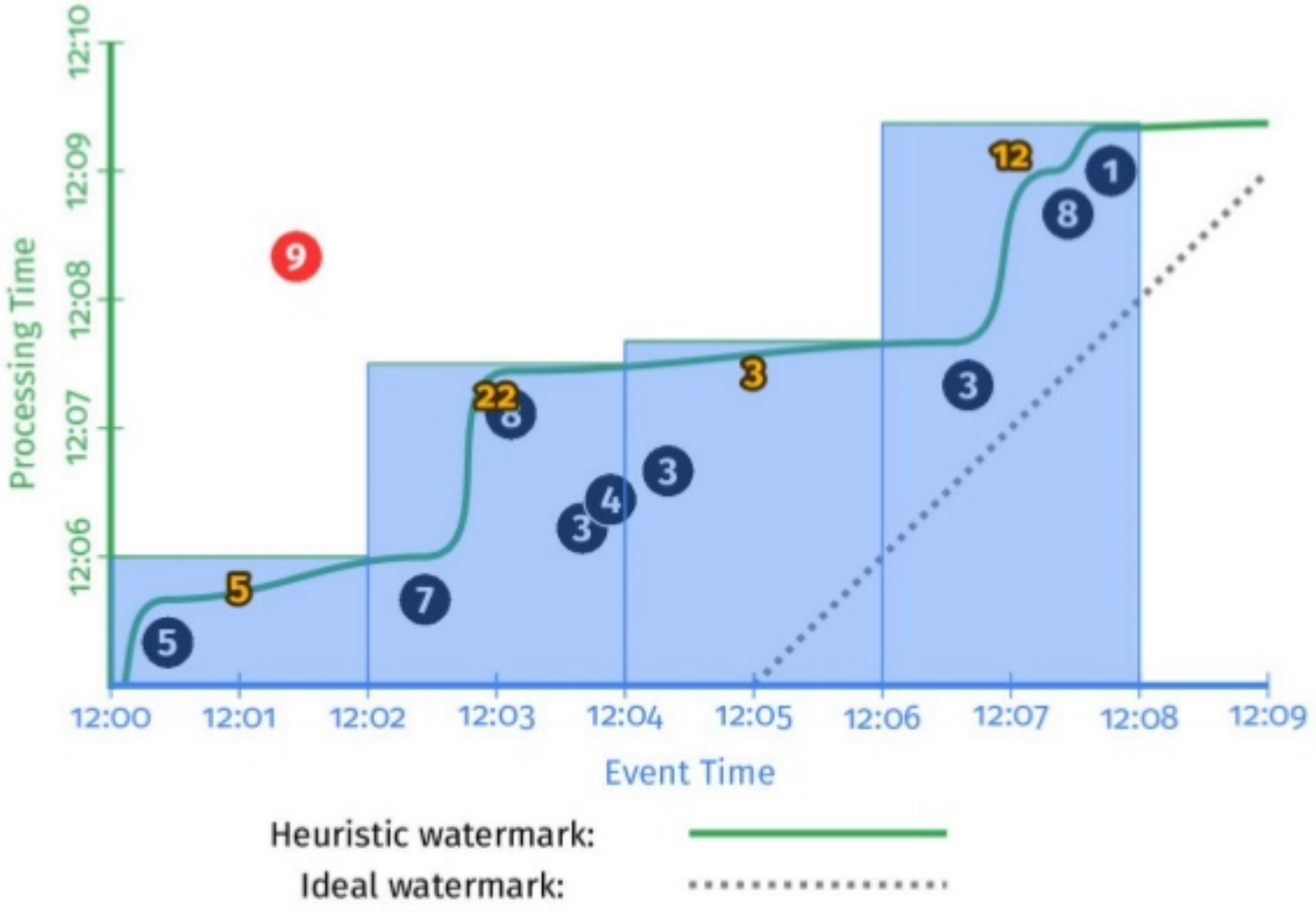


The Beam Model: **When** in Processing Time?

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark()))
    .apply(Sum.integersPerKey());
```

```
scores = (input
    | beam.WindowInto(FixedWindows(2 * 60)
      .triggering(AtWatermark()))
    | Sum.integersPerKey())
```

The Beam Model: **When** in Processing Time?

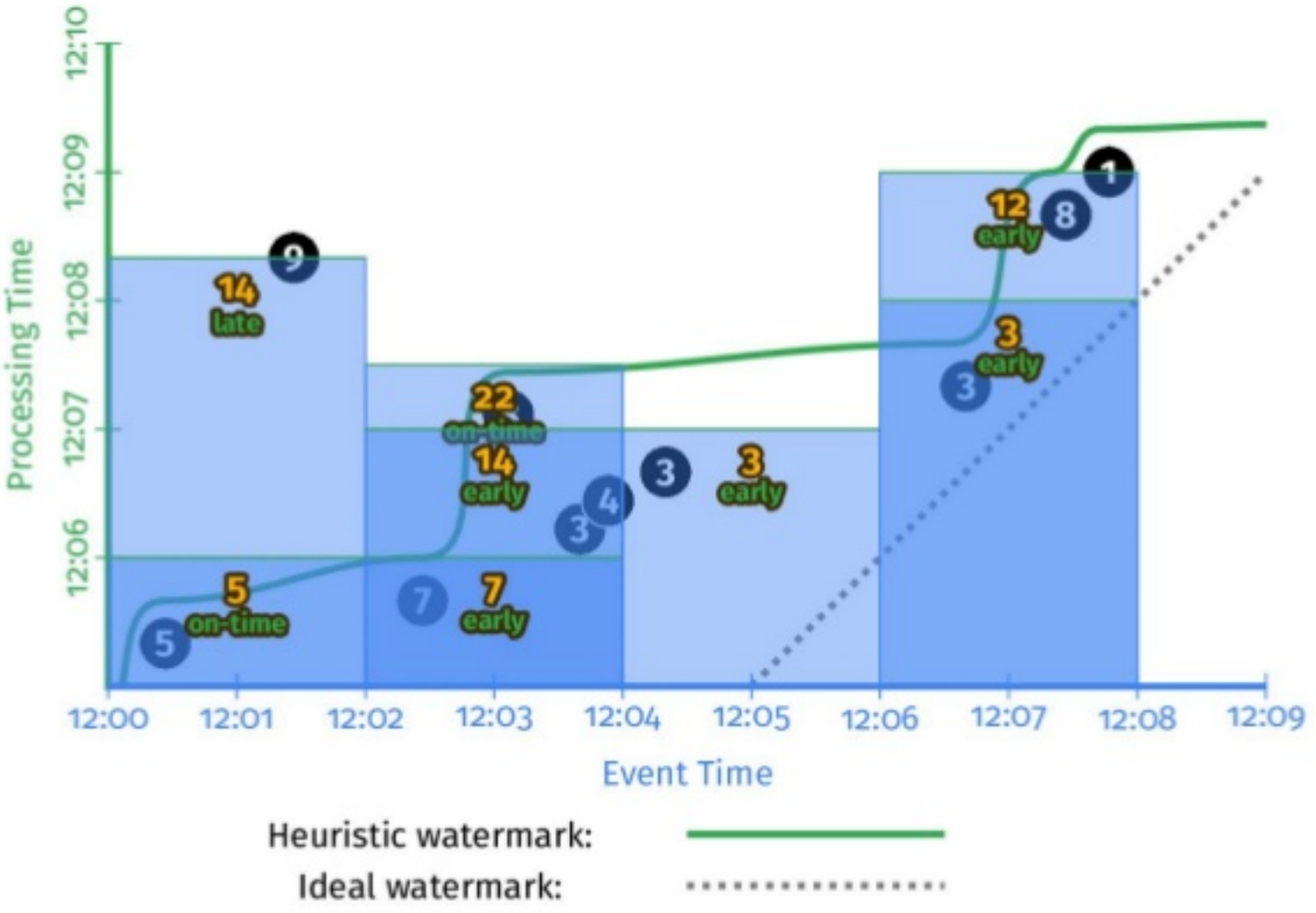


The Beam Model: **How** Do Refinements Relate?

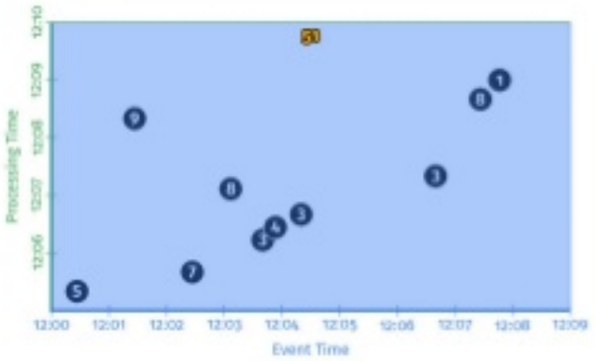
```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
            .triggering(AtWatermark()
                .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
                .withLateFirings(AtCount(1)))
            .accumulatingFiredPanes())
    .apply(Sum.integersPerKey());
```

```
scores = (input
    | beam.WindowInto(FixedWindows(2 * 60)
        .triggering(AtWatermark()
            .withEarlyFirings(AtPeriod(1 * 60))
            .withLateFirings(AtCount(1)))
        .accumulatingFiredPanes())
    | Sum.integersPerKey())
```

The Beam Model: **How** Do Refinements Relate?

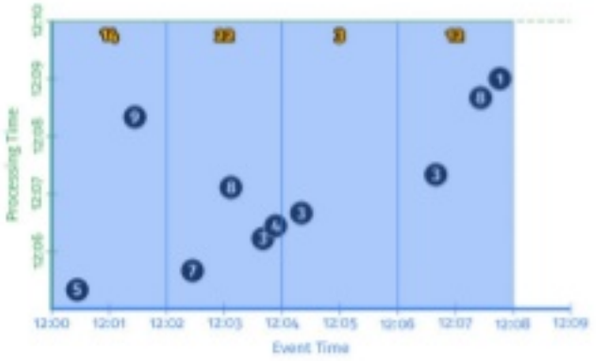


Customizing What Where When How



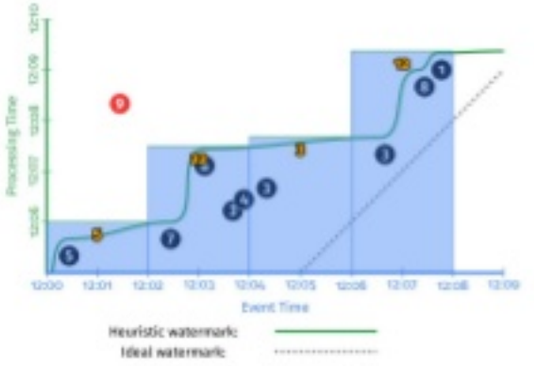
1

Classic Batch



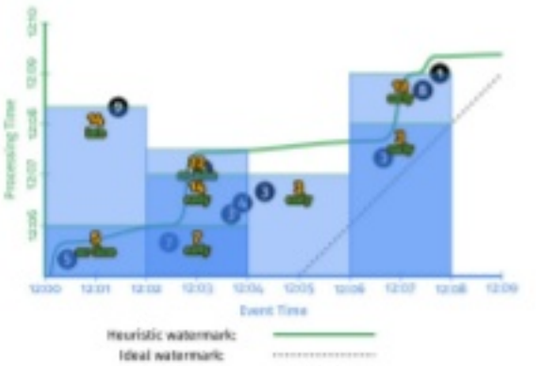
2

Windowed Batch



3

Streaming



4

Streaming + Accumulation

For more information see https://cloud.google.com/dataflow/examples/gaming-example

A Complete Example of Pythonic Beam Code

```
import apache_beam as beam
with beam.Pipeline() as p:
    (p
        | beam.io.ReadStringsFromPubSub("twitter_topic")
        | beam.WindowInto(SlidingWindows(5*60, 1*60))
        | beam.ParDo(ParseHashTagDoFn())
        | beam.combiners.Count.PerElement()
        | beam.ParDo(BigQueryOutputFormatDoFn())
        | beam.io.WriteToBigQuery("trends_table"))
```

What is Apache Beam?

1. The Beam Model: **What** / **Where** / **When** / **How**
2. SDKs for writing Beam pipelines
3. Runners for Existing Distributed Processing Backends
    - Apache Apex
    - Apache Flink
    - Apache Spark
    - Google Cloud Dataflow
    - Local (in-process) runner for testing

Beam Portability APIs (Pipeline / Job / Fn)

What are we trying to solve?

- Executing user code written in an arbitrary language (Python) on a Runner written in a different language (Java)
- Mixing user functions written in different languages (Connectors, Sources, Sinks, …)

Terminology

Beam Model
Describes the API concepts and the possible operations on PCollections.

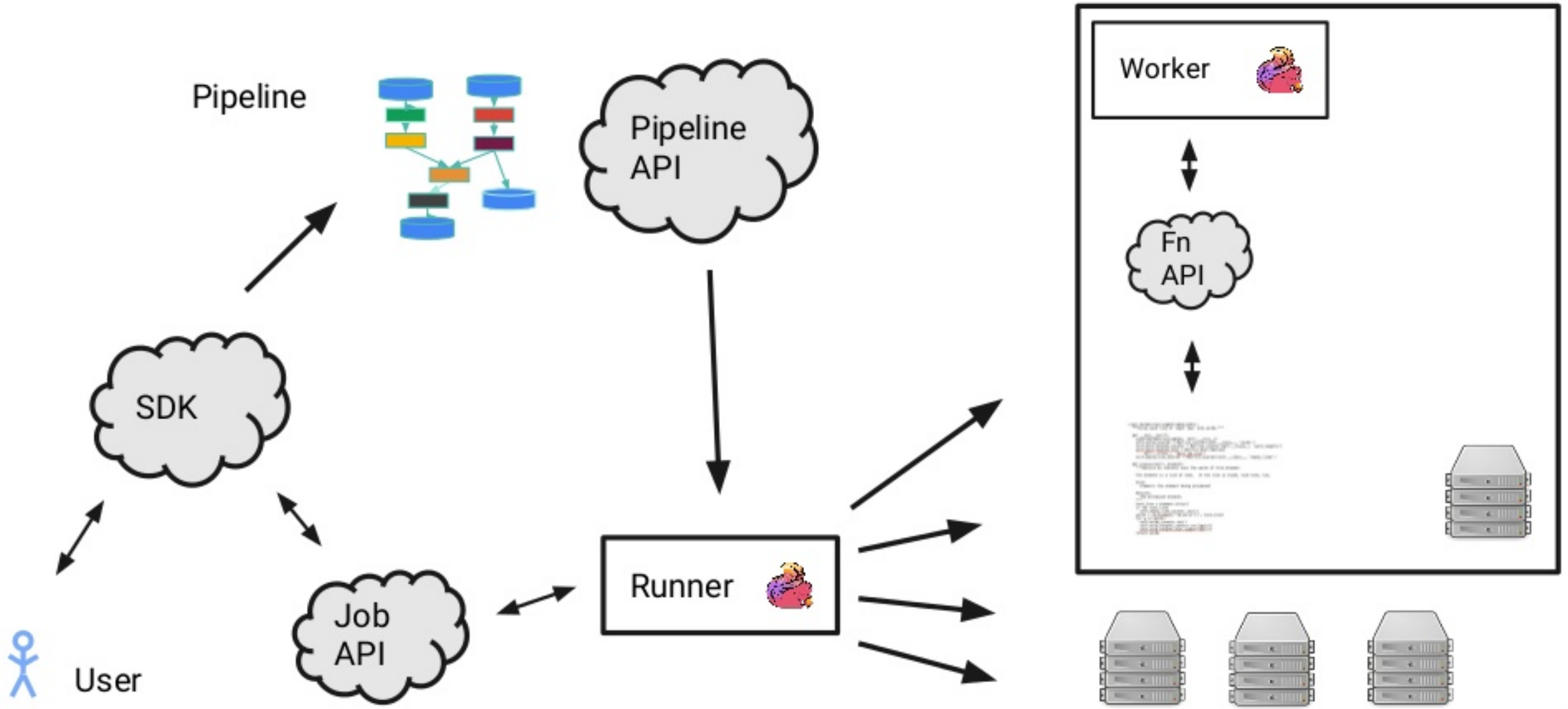
Pipeline
User-defined graph of transformations on PCollections. This is constructed using a Beam SDK. The transformations can contain UDFs.

Runner
Executes a Pipeline. For example: *FlinkRunner*.

Beam SDK
Language specific library/framework for creating programs that use the Beam Model. Allows defining Pipelines and UDFs and provides APIs for executing them.

User-defined function (UDF)
Code in Java, Python, … that specifies how data is transformed. For example *DoFn* or *CombineFn*.

Executing a Beam Pipeline - The Big Picture



APIs for Different Pipeline Lifecycle Stages

Pipeline API

- Used by the SDK to construct SDK-agnostic Pipeline representation
- Used by the Runner to translate a Pipeline to runner-specific operations

Job API

- (API for interacting with a running Pipeline)

Fn API

- Used by an SDK harness for communication with a Runner
- User by the Runner to push work into an SDK harness

Pipeline API (simplified)

- Definition of common primitive transformations (Read, ParDo, Flatten, Window.into, GroupByKey)
- Definition of serialized Pipeline (protobuf)

```
Pipeline = {PCollection*, PTransform*, WindowingStrategy*, Coder*}

PTransform = {Inputs*, Outputs*, FunctionSpec}

FunctionSpec = {URN, payload}
```

https://s.apache.org/beam-runner-api

Job API

```
public interface JobApi {
  State getState(); // RUNNING, DONE, CANCELED, FAILED ...
  State cancel() throws IOException;
  State waitUntilFinish(Duration duration);
  State waitUntilFinish();
  MetricResults metrics();
}
```

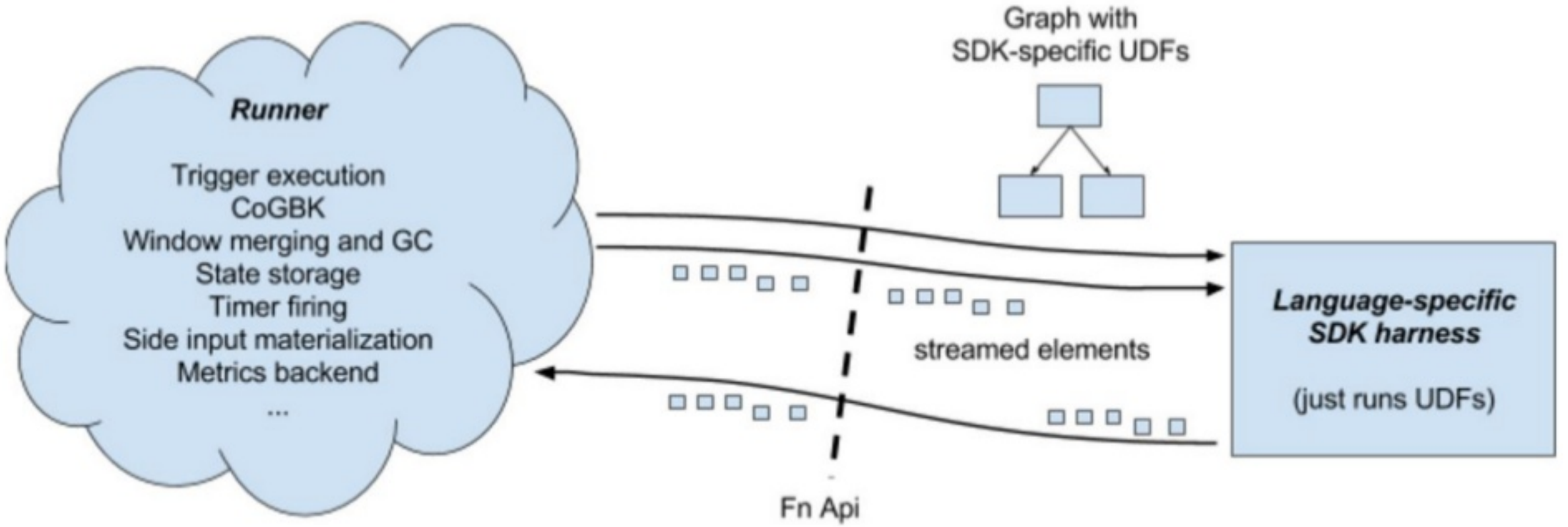
Fn API

- gRPC interface definitions for communication between an SDK harness and a Runner

- **Control**: Used to tell the SDK which UDFs to execute and when to execute them.
- **Data**: Used to move data between the language specific SDK harness and the runner.
- **State**: Used to support user state, side inputs, and group by key reiteration.
- **Logging**: Used to aggregate logging information from the language specific SDK harness.

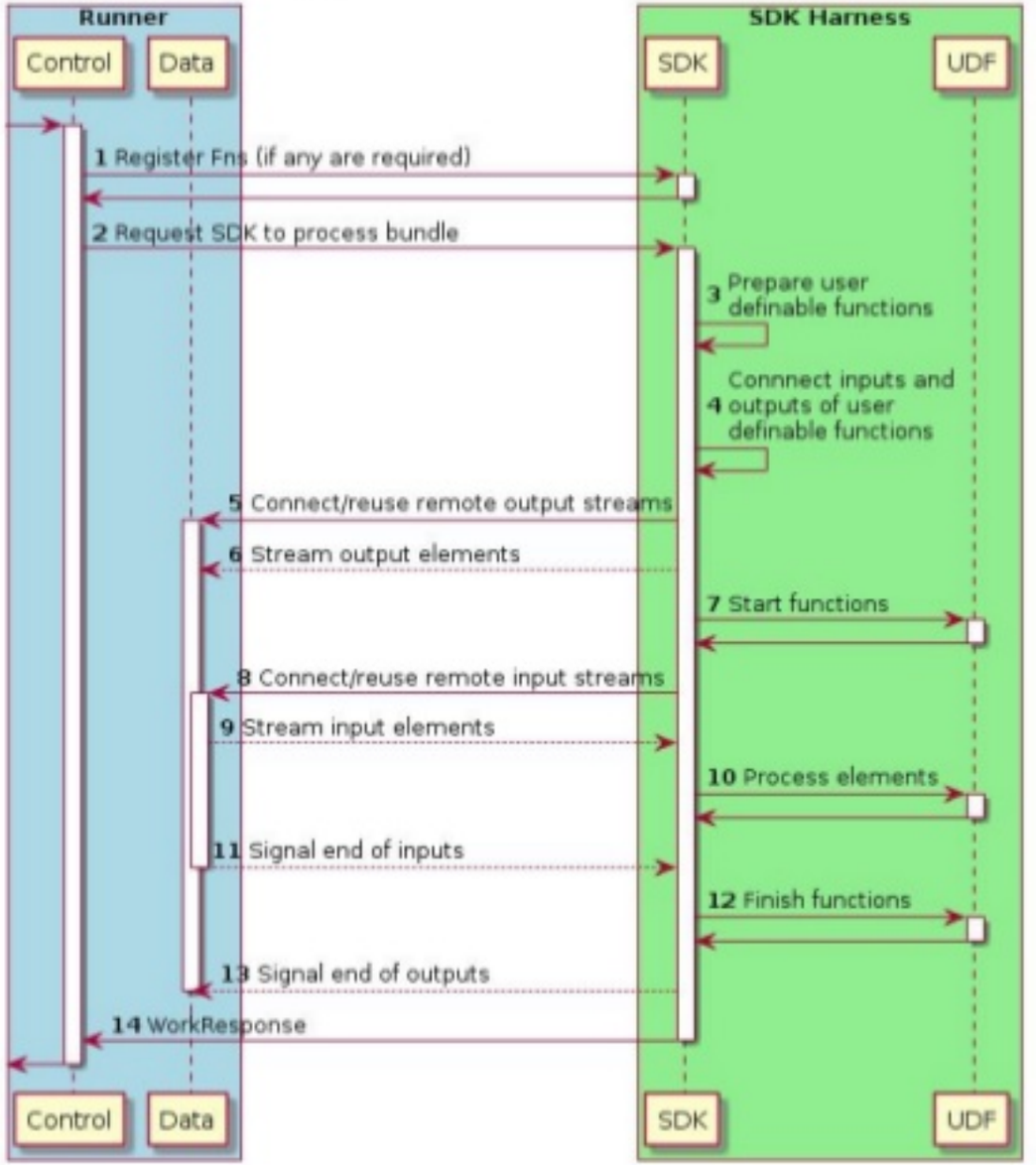
https://s.apache.org/beam-fn-api

Fn API (continued)



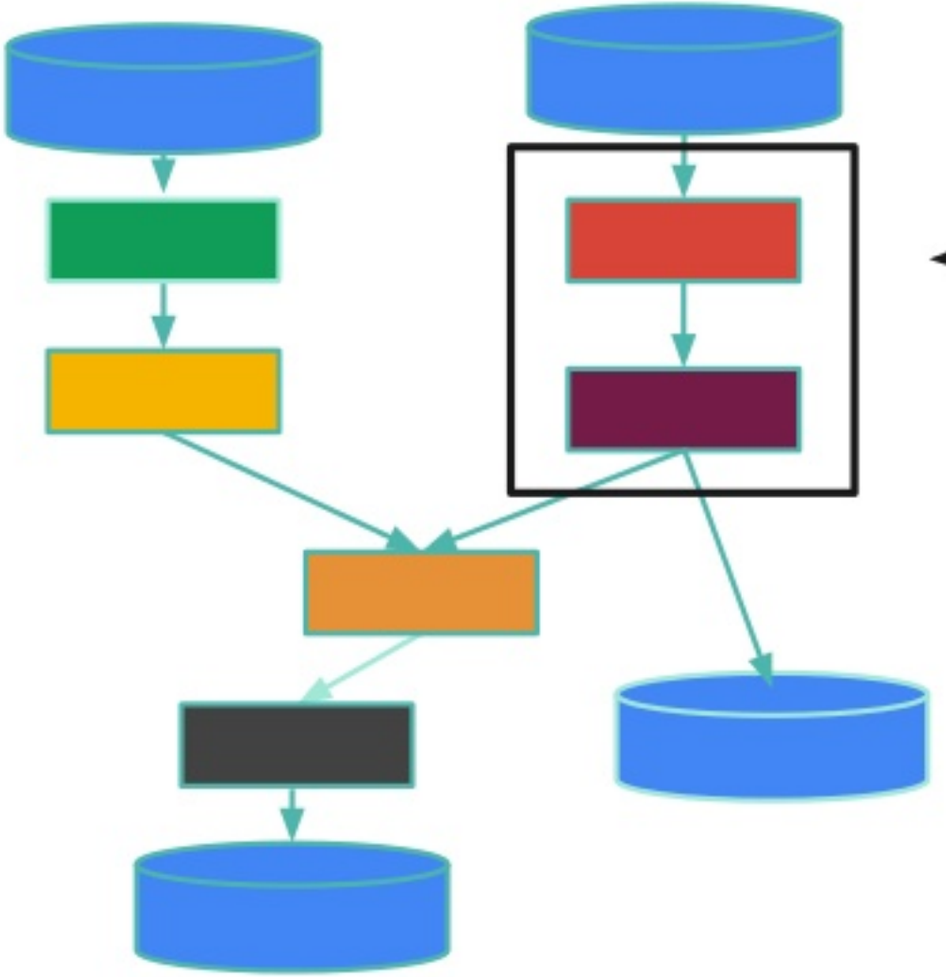
https://s.apache.org/beam-fn-api

Fn API - Bundle Processing



https://s.apache.org/beam-fn-api-processing-a-bundle

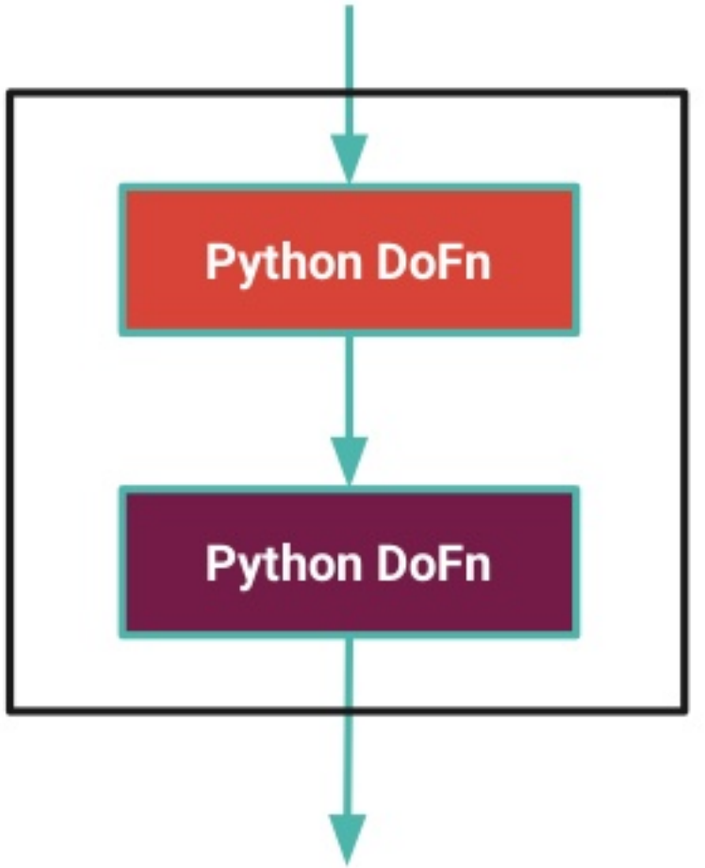
Fn API - Processing DoFns



Say we need to execute this part

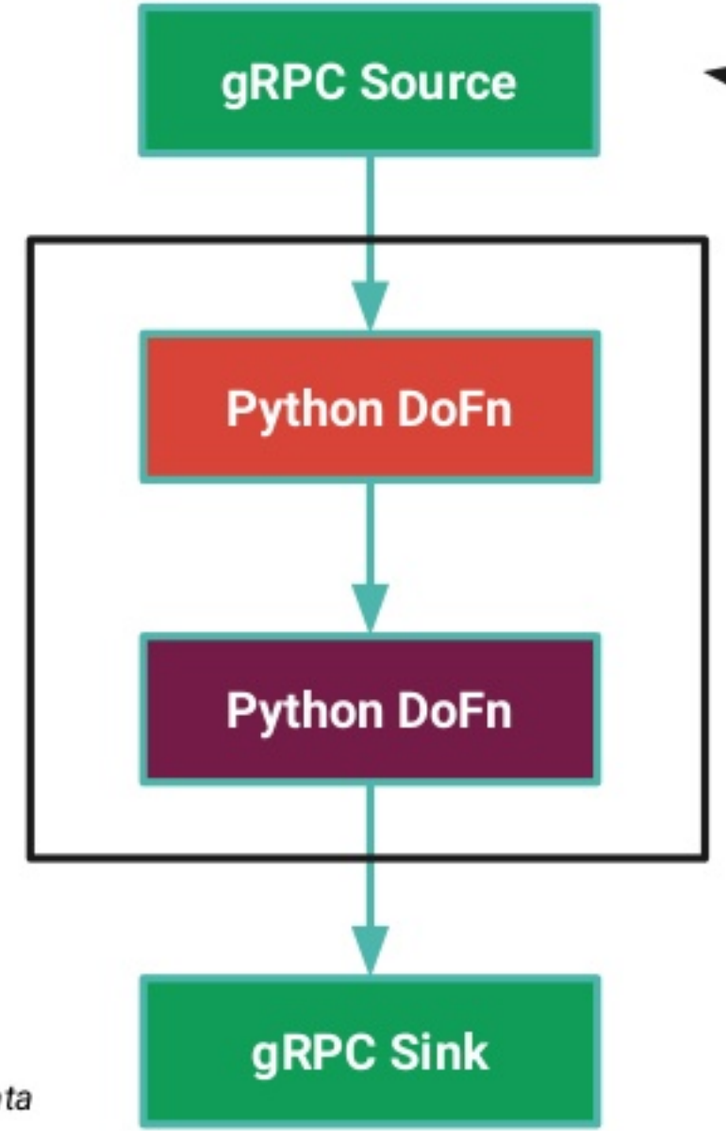
https://s.apache.org/beam-fn-api-send-and-receive-data

Fn API - Processing DoFns



https://s.apache.org/beam-fn-api-send-and-receive-data

Fn API - Processing DoFns (Pipeline manipulation)



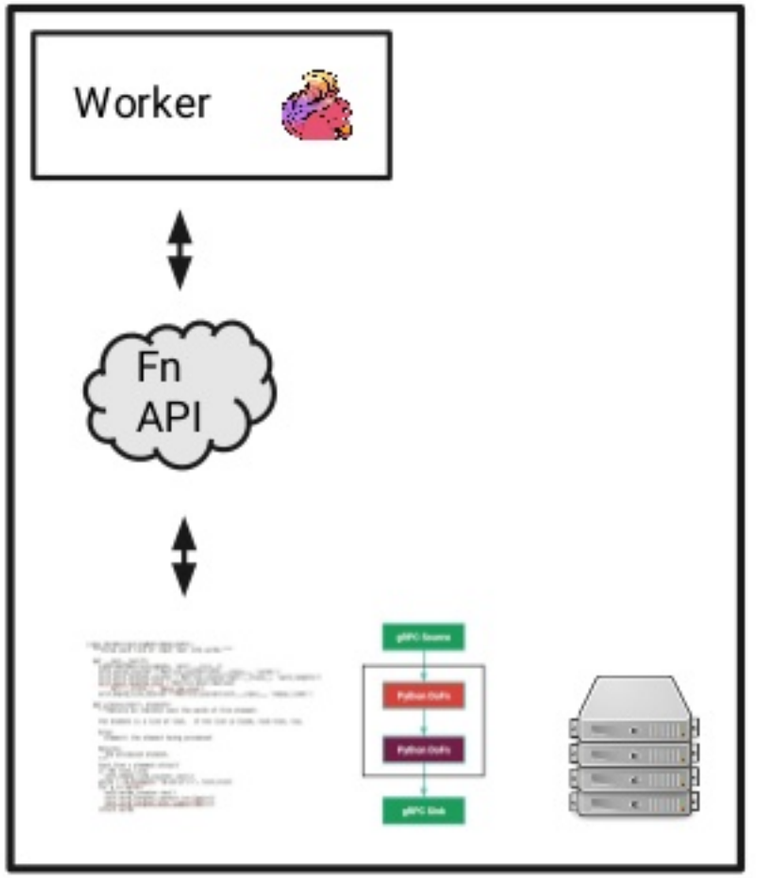
The Runner inserts these

https://s.apache.org/beam-fn-api-send-and-receive-data

Fn API - Executing the user Fn using a SDK Harness

- We can execute as a separate process
- We can execute in a Docker container

- Repository of containers for different SDKs
- We inject the user code into the container when starting
- Container is user-configurable



https://s.apache.org/beam-fn-api-container-contract

Executing Pythonic* Beam Jobs on Fink

*or other languages

What is the (Flink) Runner/Flink doing in all this?

- Analyze/transform the Pipeline (Pipeline API)
- Create a Flink Job (DataSet/DataStream API)
- Ship the user code/docker container description
- In an operator: Open gRPC services for control/data/logging/state plane
- Execute arbitrary user code using the Fn API

Easy, because Flink state/timers map well to Beam concepts!

Advantages/Disadvantages

- Complete isolation of user code
- Complete configurability of execution environment (with Docker)
- We can support code written in arbitrary languages
- We can mix user code written in different languages

- Slower (RPC overhead)
- Using Docker requires docker 

The Future

Future work

- Finish what I just talked about
- Finalize the different APIs (not Flink-specific)
- Mixing and matching connectors written in different languages
- Wait for new SDKs in other languages, they will just work

Learn More!

Apache Beam/Apache Flink

https://flink.apache.org / https://beam.apache.org

Beam Fn API design documents

https://s.apache.org/beam-runner-api
https://s.apache.org/beam-fn-api
https://s.apache.org/beam-fn-api-processing-a-bundle
https://s.apache.org/beam-fn-state-api-and-bundle-processing
https://s.apache.org/beam-fn-api-send-and-receive-data
https://s.apache.org/beam-fn-api-container-contract

Join the mailing lists!

user-subscribe@flink.apache.org / dev-subscribe@flink.apache.org
user-subscribe@beam.apache.org / dev-subscribe@beam.apache.org

Follow @ApacheFlink / @ApacheBeam on Twitter

Thank you!

Backup Slides

Processing Time vs. Event Time