Fundamentals of Stream Processing with Apache Beam (incubating)

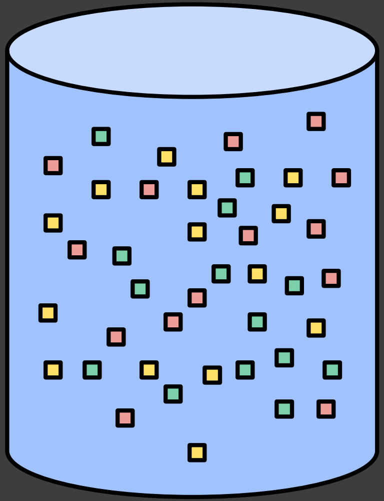
Frances Perry & Tyler Akidau
@francesjperry, @ takidau
Apache Beam Committers & Google Engineers

Agenda

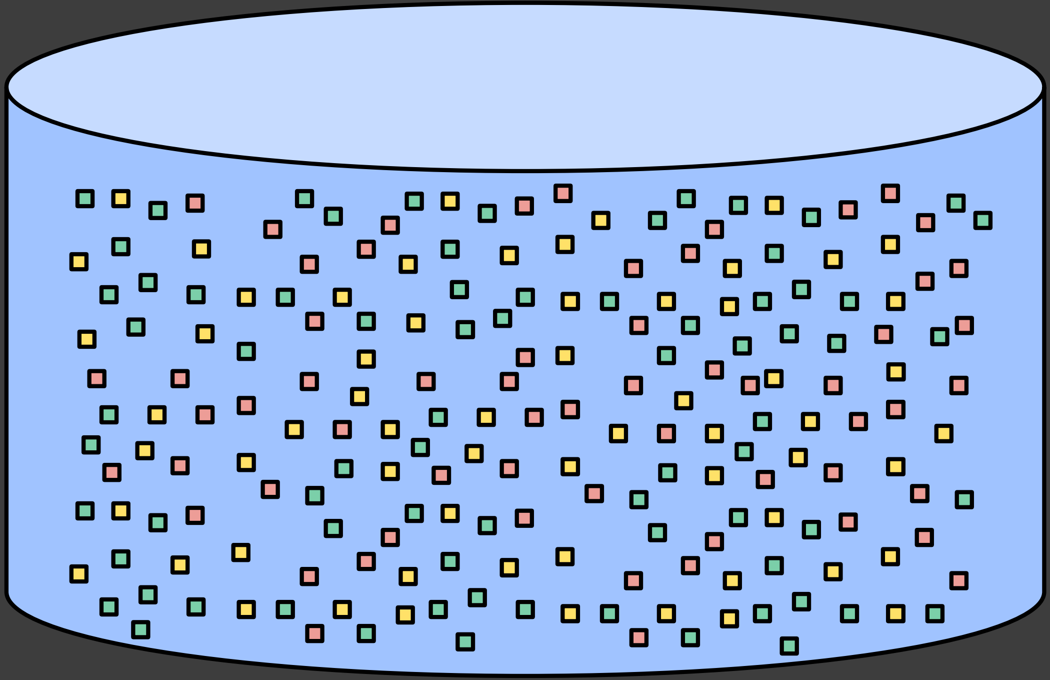
1. Infinite, Out-of-Order Data Sets
2. What, Where, When, How
3. Reasons This is Awesome
4. Apache Beam (incubating)

1 Infinite, Out-of-Order Data Sets

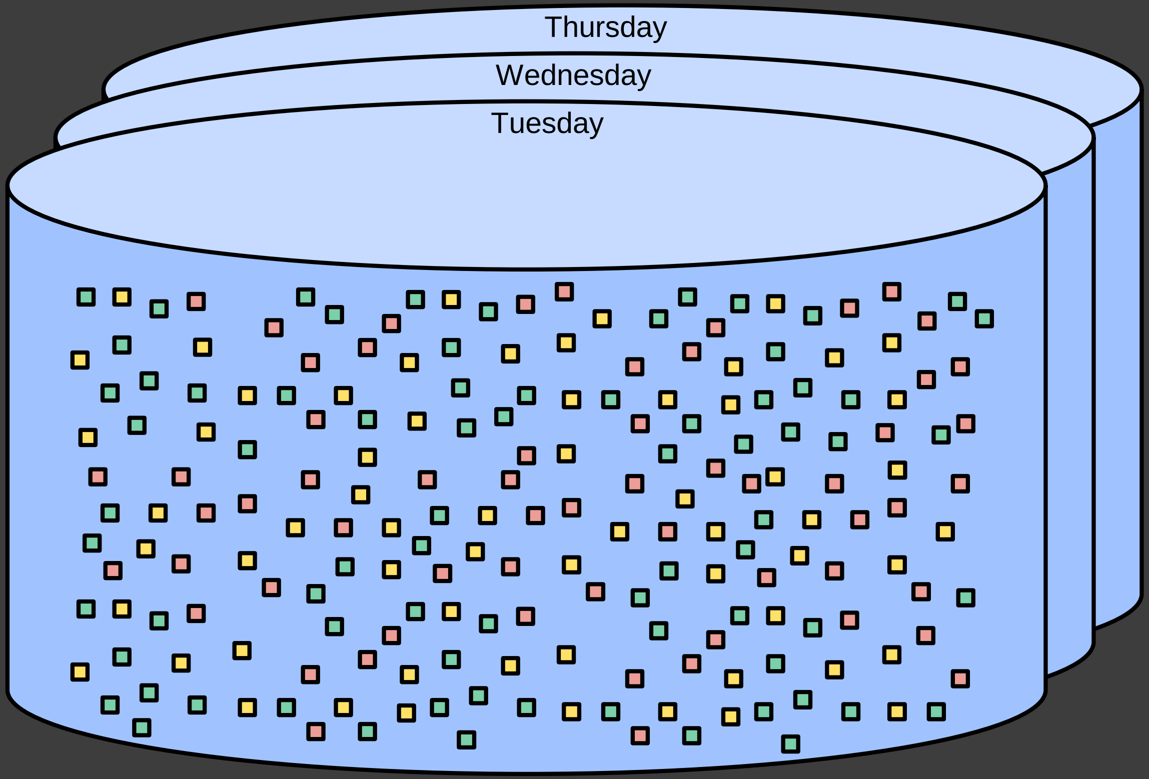
Data...



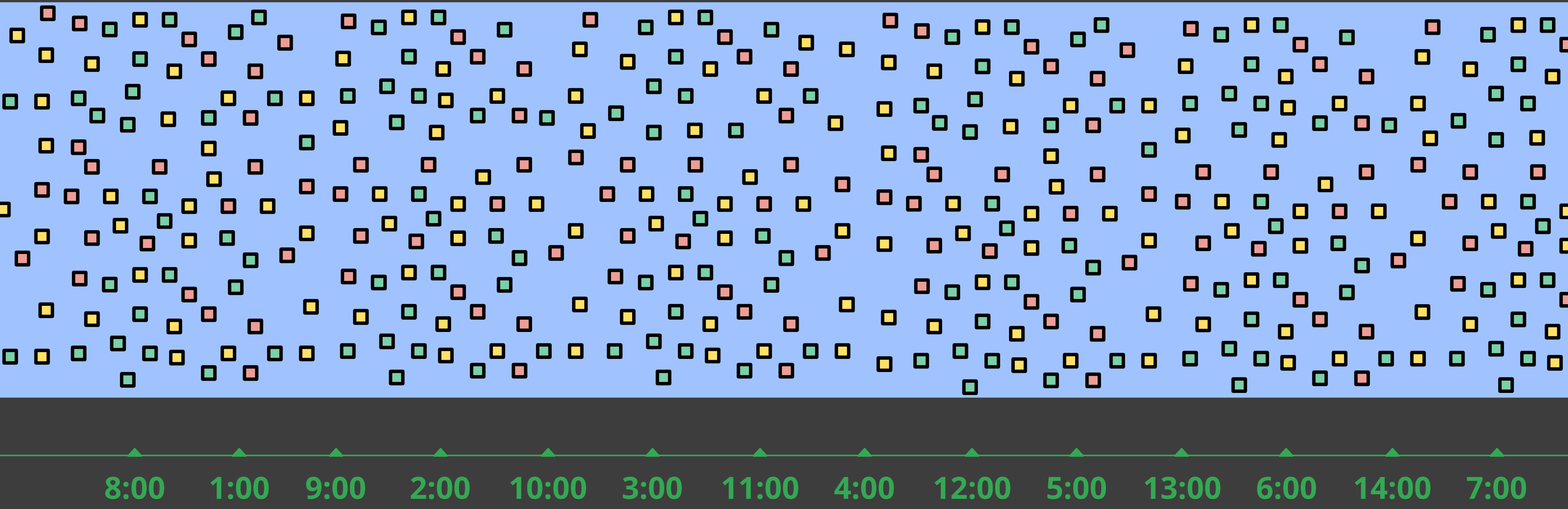
…can be big…



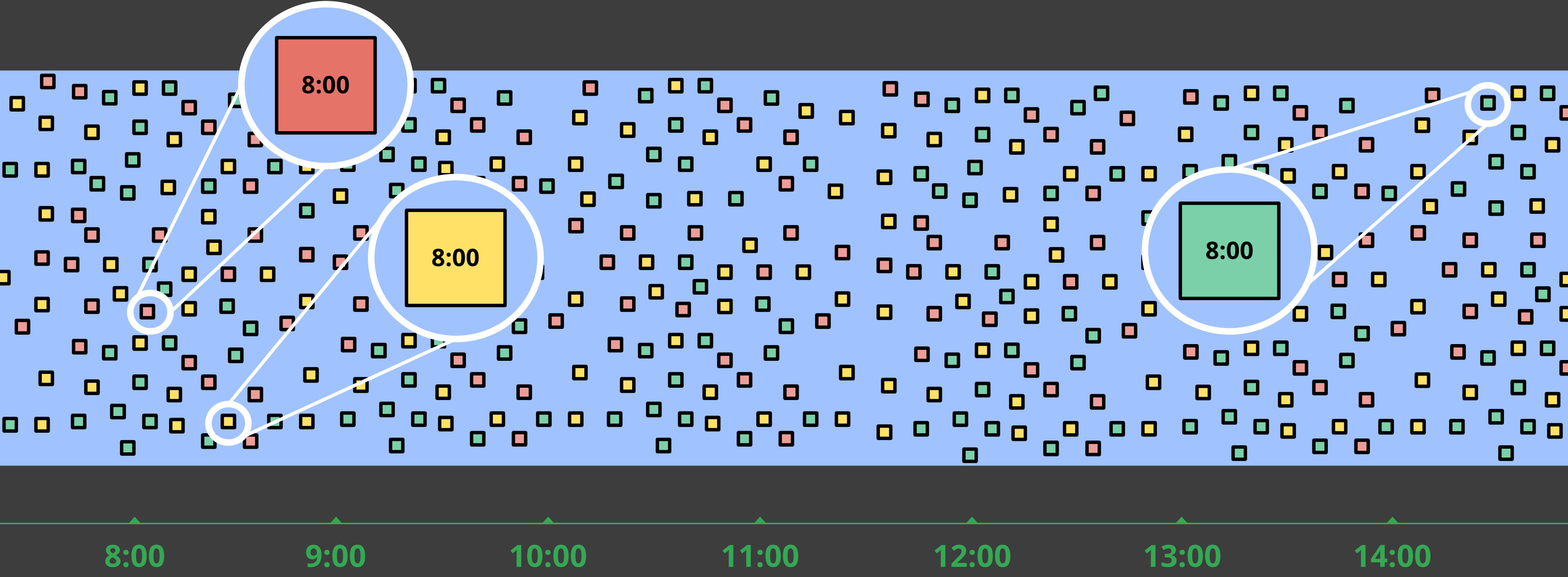
...really, really big...



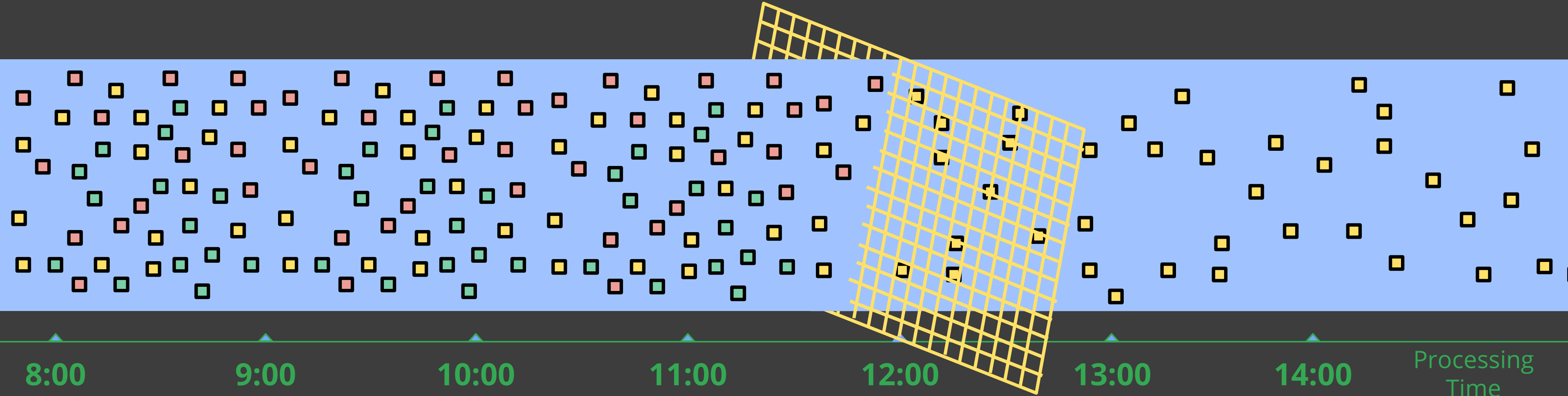
… maybe infinitely big…



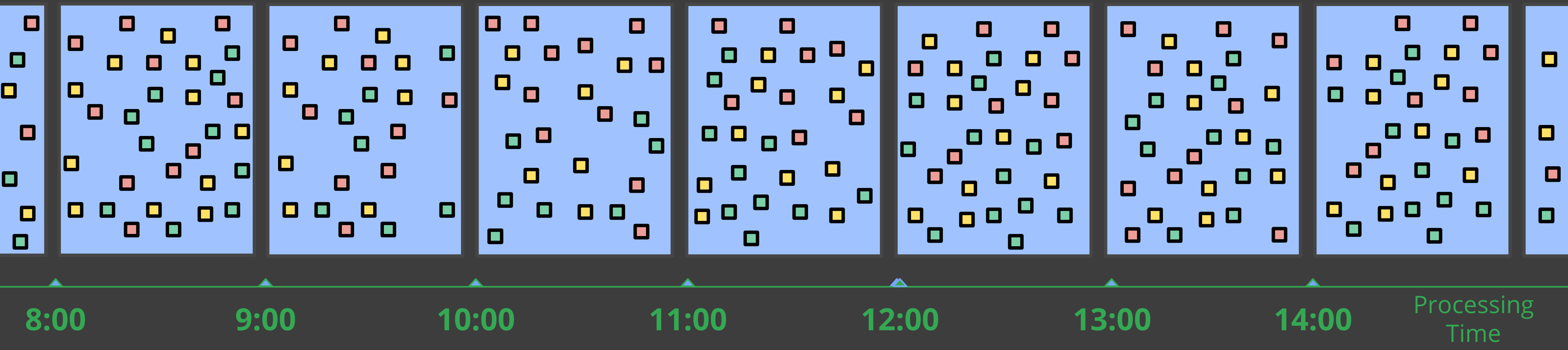
… with unknown delays.



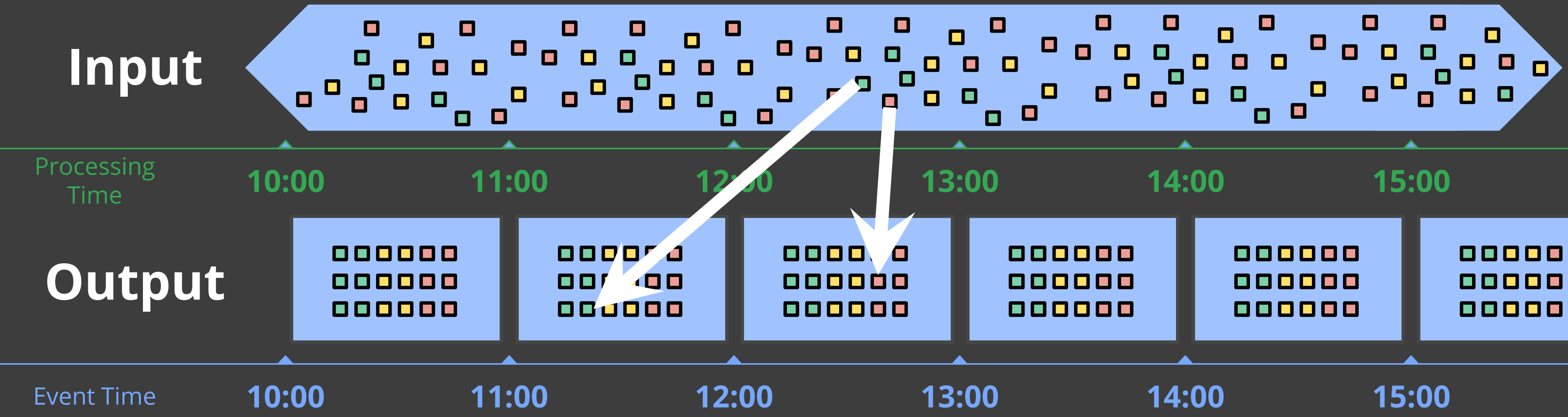
Element-wise transformations



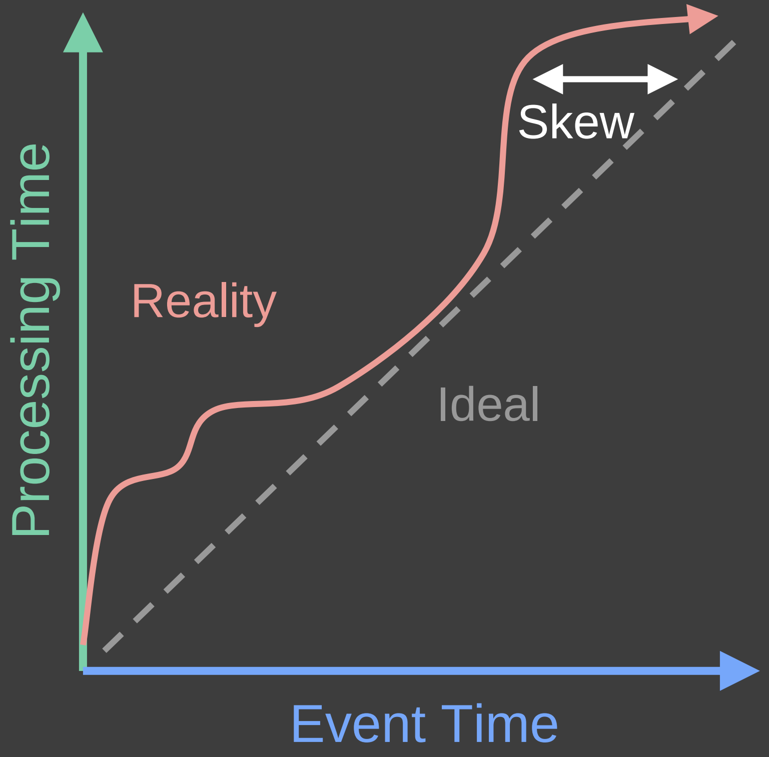
Aggregating via Processing-Time Windows



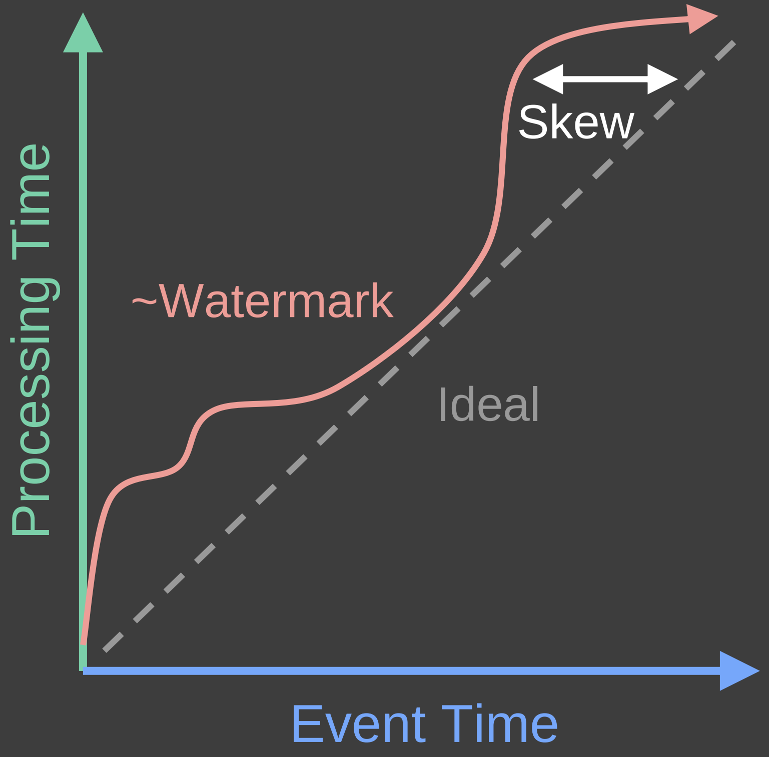
Aggregating via Event-Time Windows



Formalizing Event-Time Skew



Formalizing Event-Time Skew



Watermarks describe event time progress.

"No timestamp earlier than the watermark will be seen"

Often heuristic-based.

Too Slow? Results are *delayed*.
Too Fast? Some data is *late*.

2

What, Where, When, How

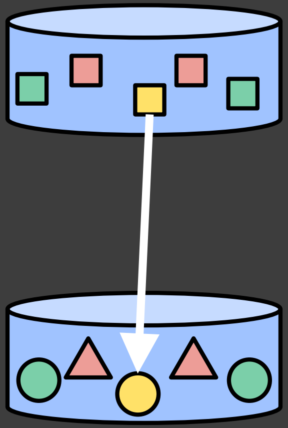
What are you computing?

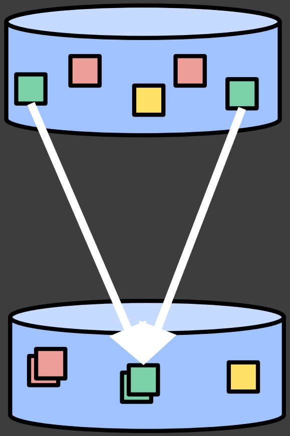
Where in event time?

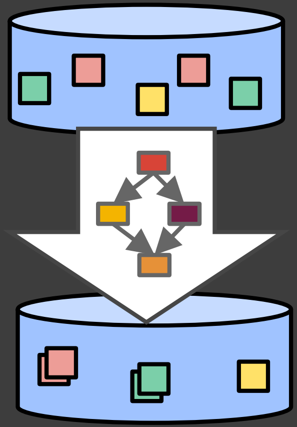
When in processing time?

How do refinements relate?

What are you computing?


Element-Wise

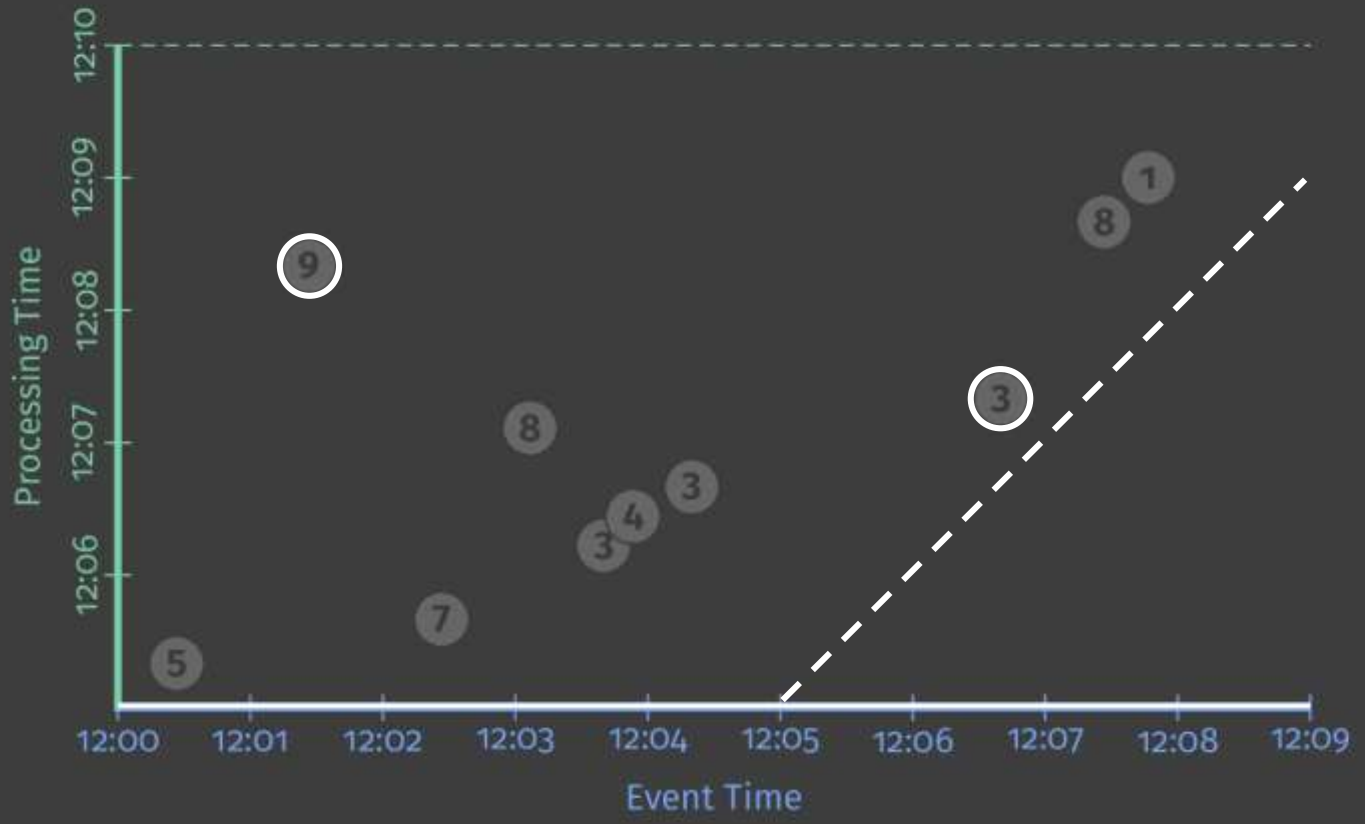

Aggregating


Composite

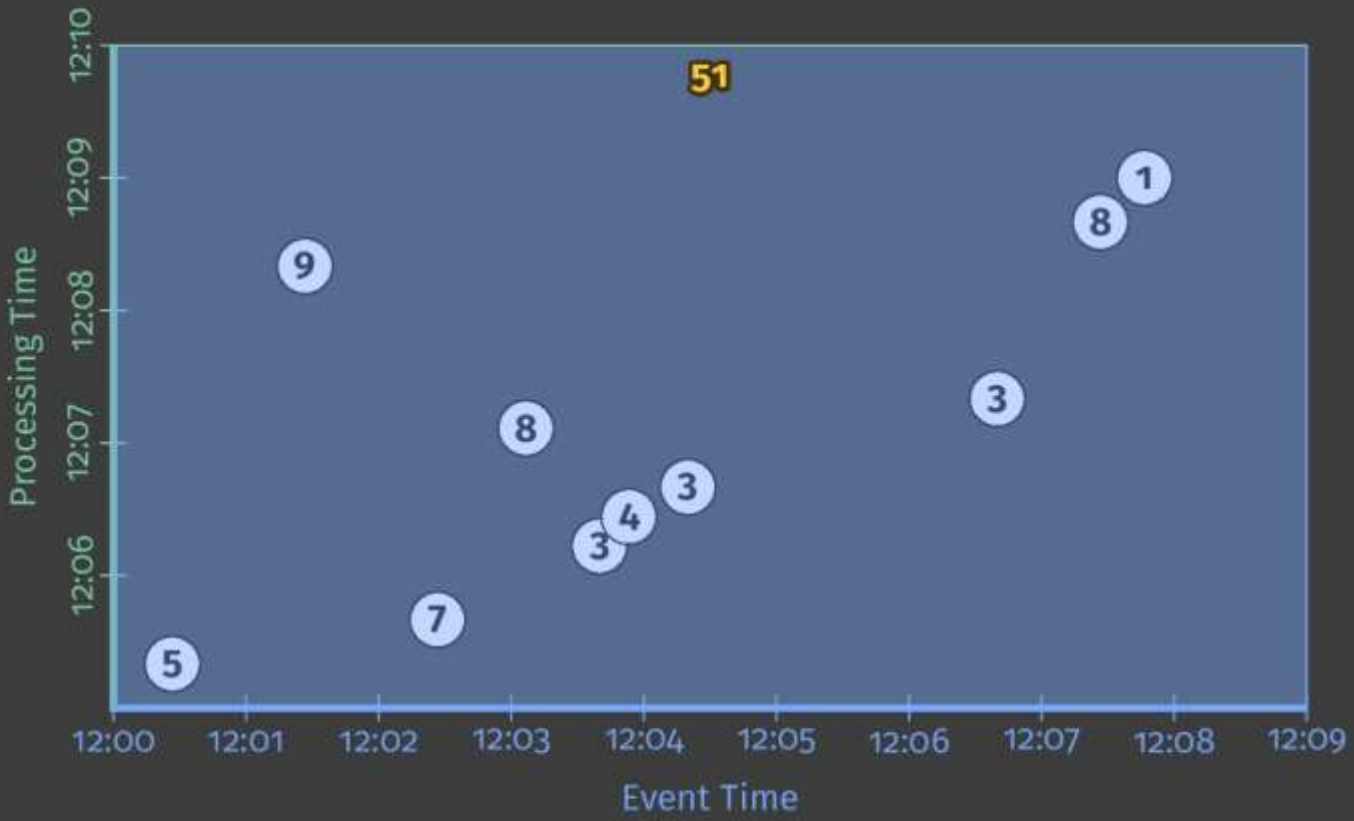
What: Computing Integer Sums

```
// Collection of raw log lines
PCollection<String> raw = IO.read(...);

// Element-wise transformation into team/score pairs
PCollection<KV<String, Integer>> input =
     raw.apply(ParDo.of(new ParseFn()));

// Composite transformation containing an aggregation
PCollection<KV<String, Integer>> scores =
     input.apply(Sum.integersPerKey());
```

What: Computing Integer Sums

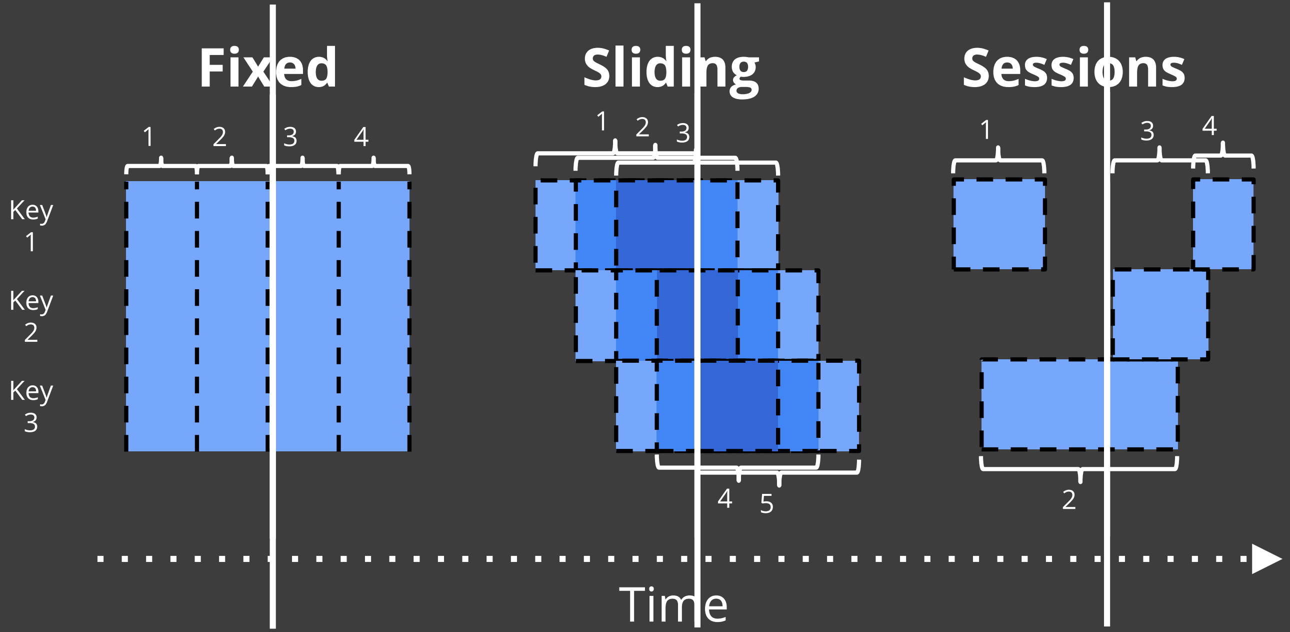


What: Computing Integer Sums



Where in event time?

Windowing divides data into event-time-based finite chunks.

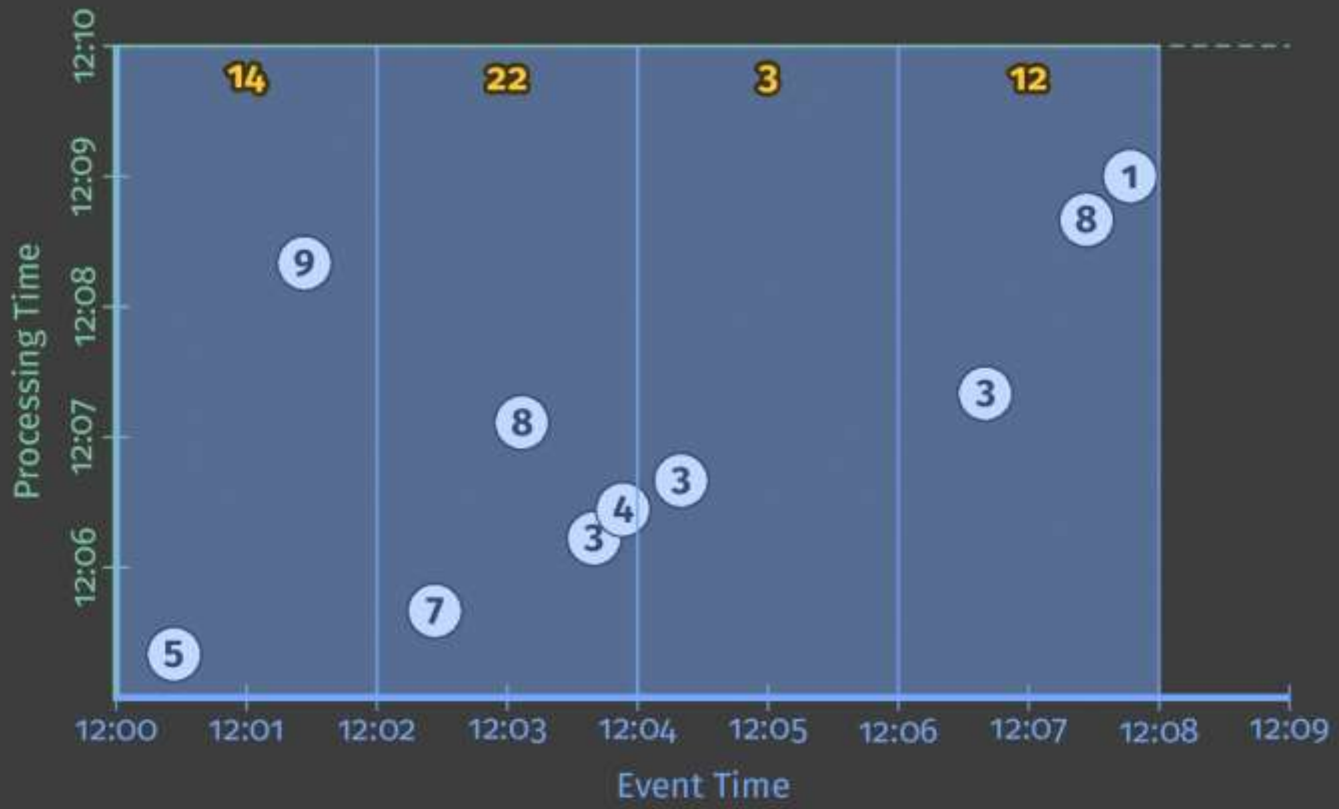


Often required when doing aggregations over unbounded data.

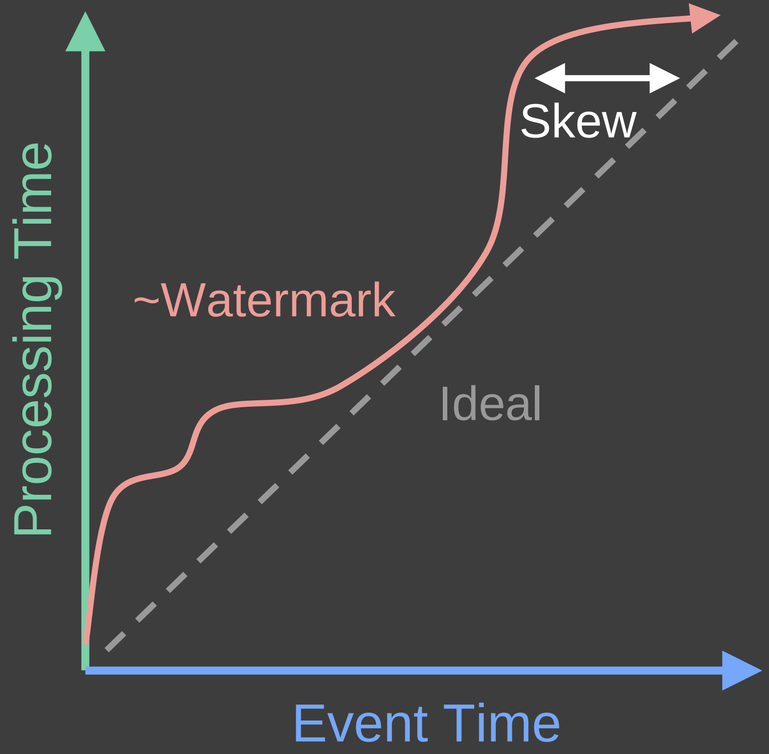
Where: Fixed 2-minute Windows

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

Where: Fixed 2-minute Windows



When in processing time?

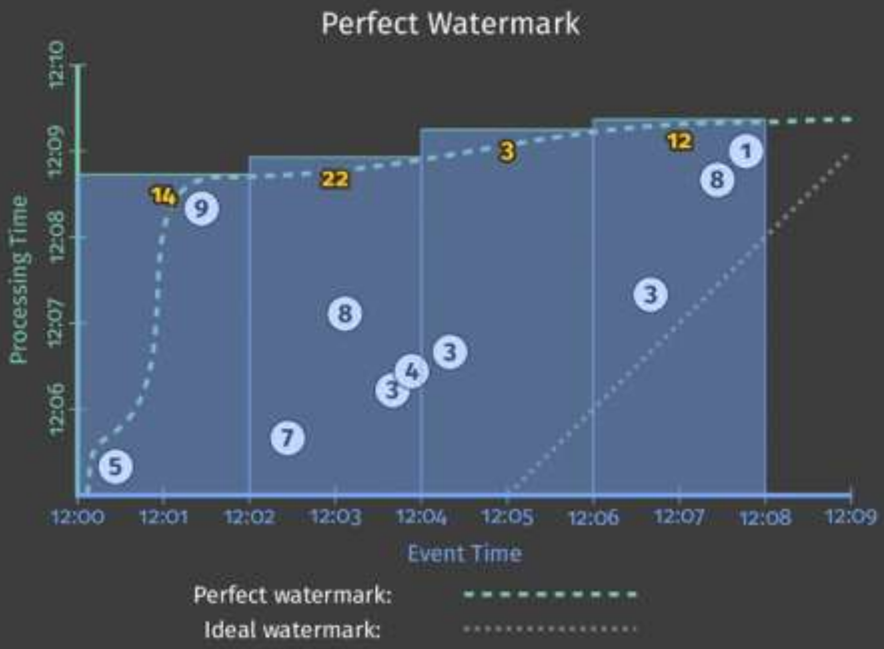


- Triggers control when results are emitted.
- Triggers are often relative to the watermark.

When: Triggering at the Watermark

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

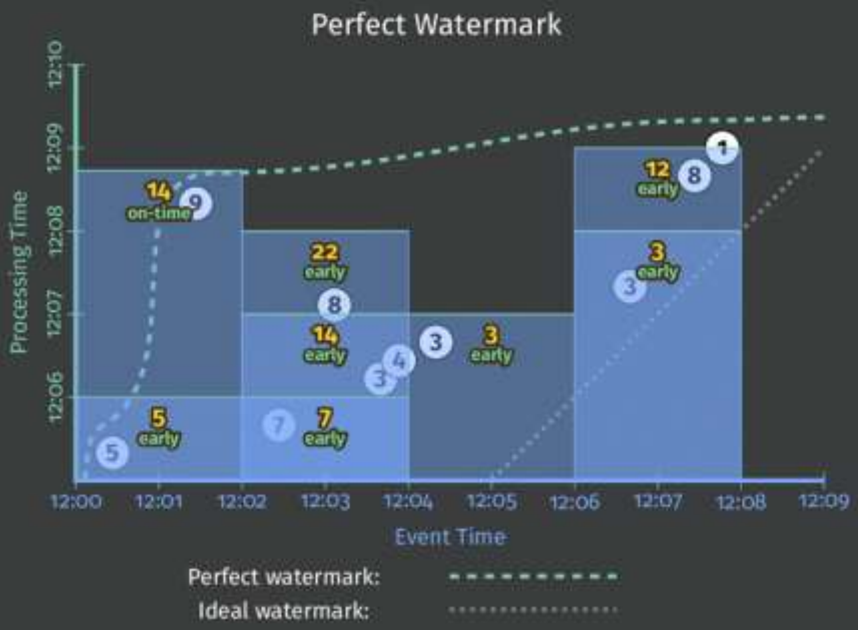
When: Triggering at the Watermark

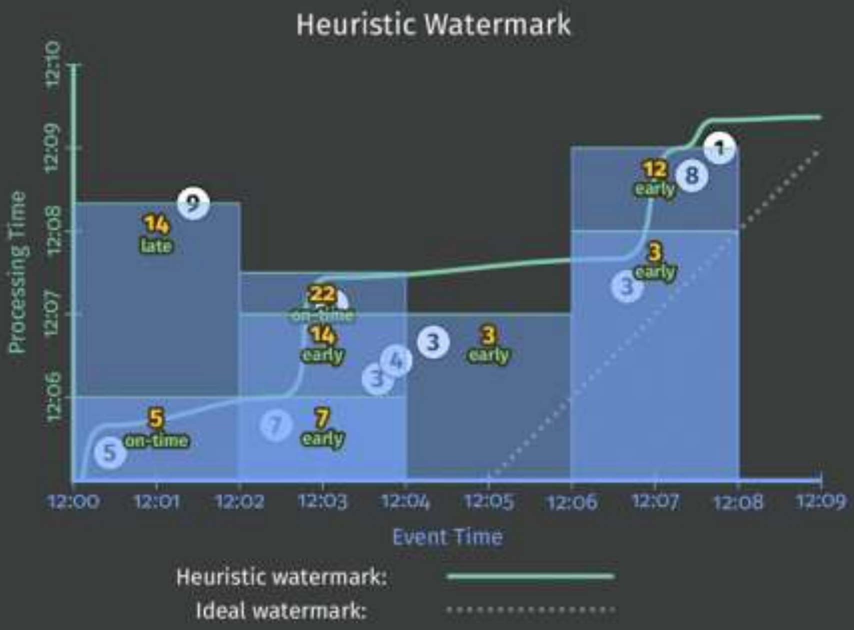


When: Early and Late Firings

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

When: Early and Late Firings





How do refinements relate?

- How should multiple outputs per window accumulate?
- Appropriate choice depends on consumer.

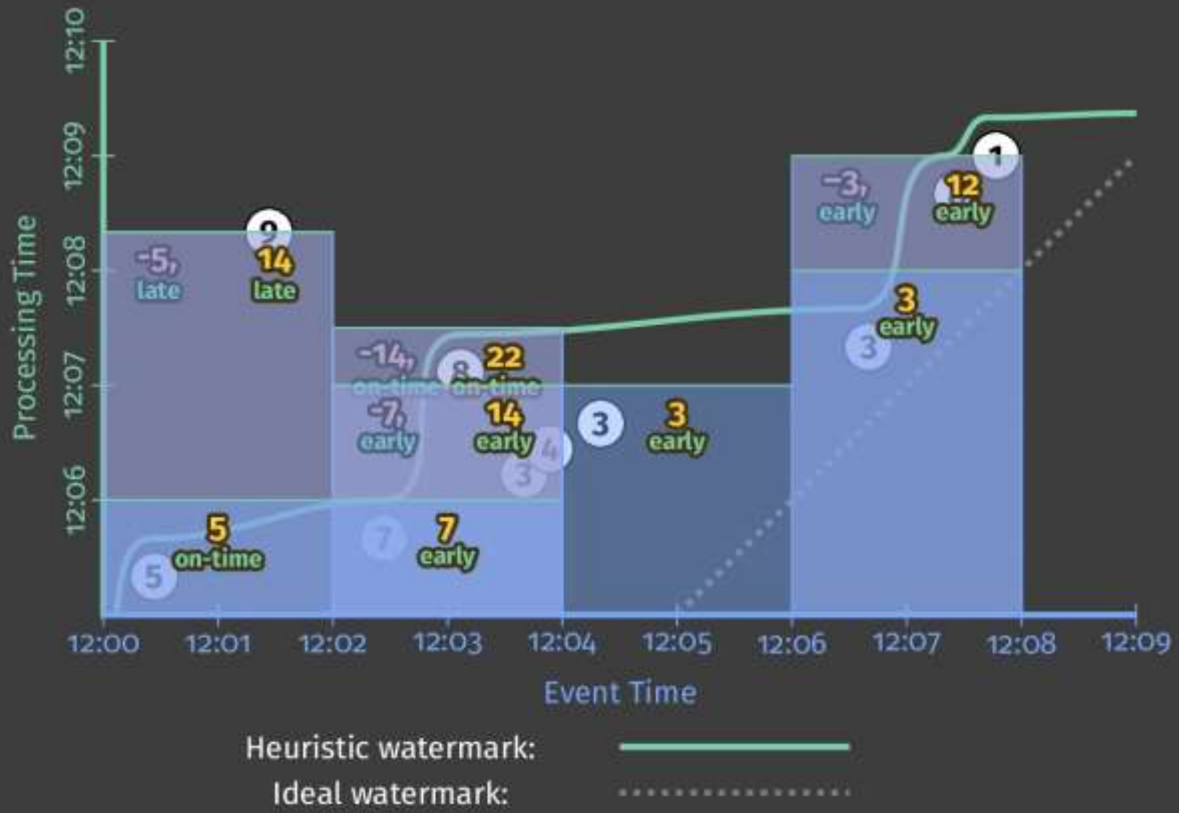
Firing	Elements	Discarding	Accumulating	Acc. & Retracting
Speculative	[3]	3	3	3
Watermark	[5, 1]	6	9	9, -3
Late	[2]	2	11	11, -9
Last Observed		***2***	***11***	***11***
Total Observed		***11***	***23***	***11***

(Accumulating & Retracting not yet implemented.)

How: Add Newest, Remove Previous

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

How: Add Newest, Remove Previous



3

Reasons This is Awesome

What / Where / When / How

Correctness

Power

Composability

Flexibility

Modularity

What / Where / When / How

Correctness

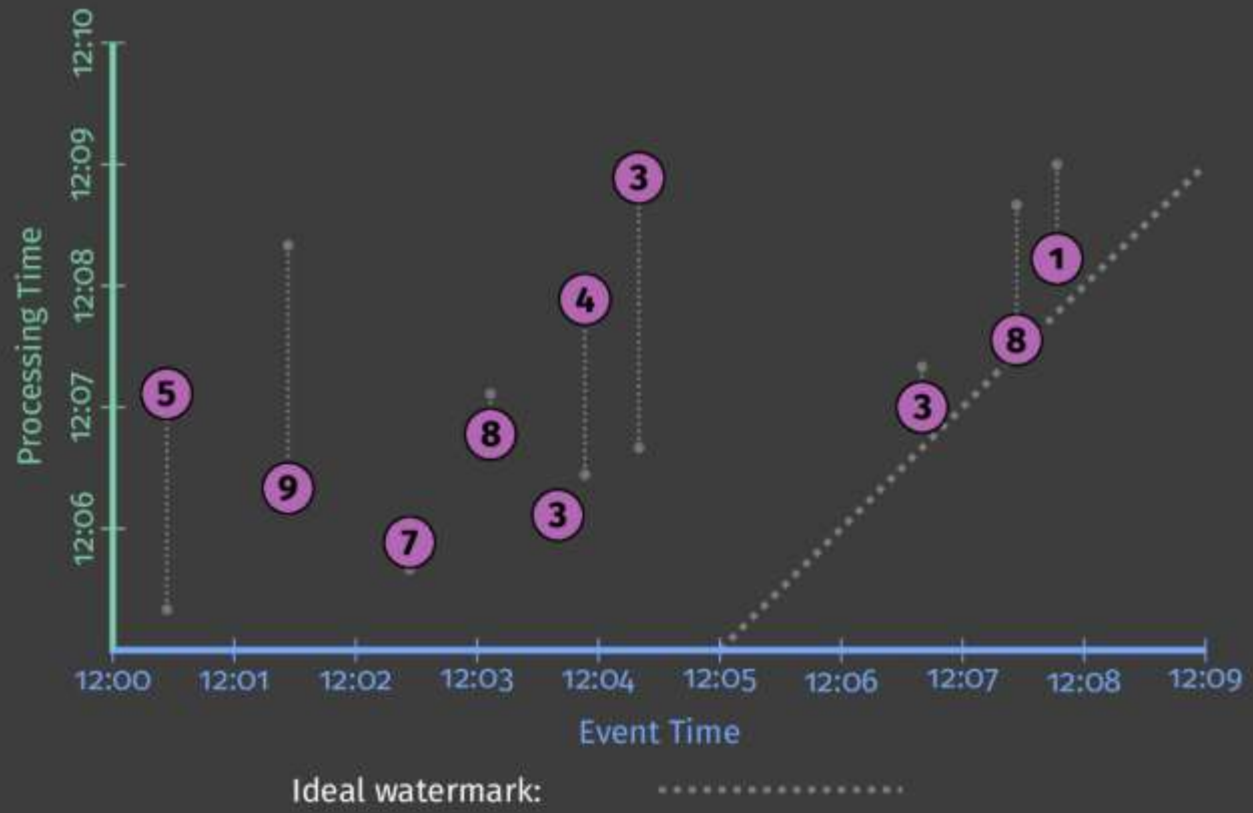
Power

Composability

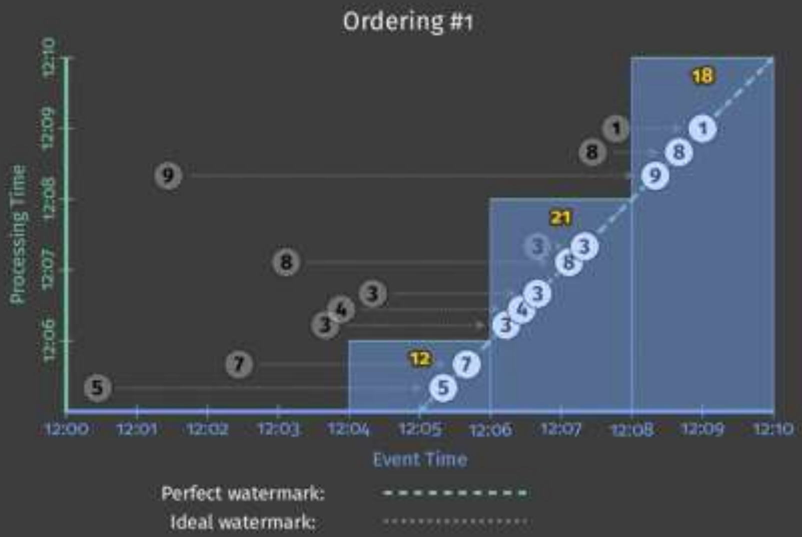
Flexibility

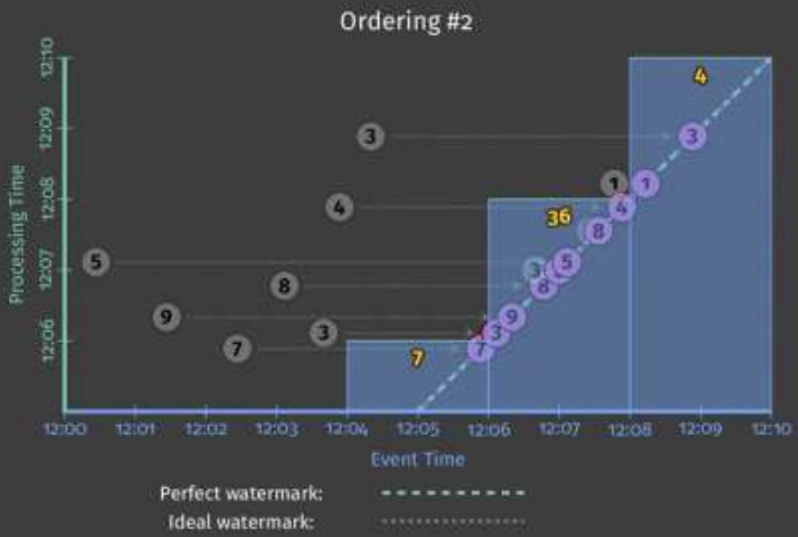
Modularity

Distributed Systems are Distributed

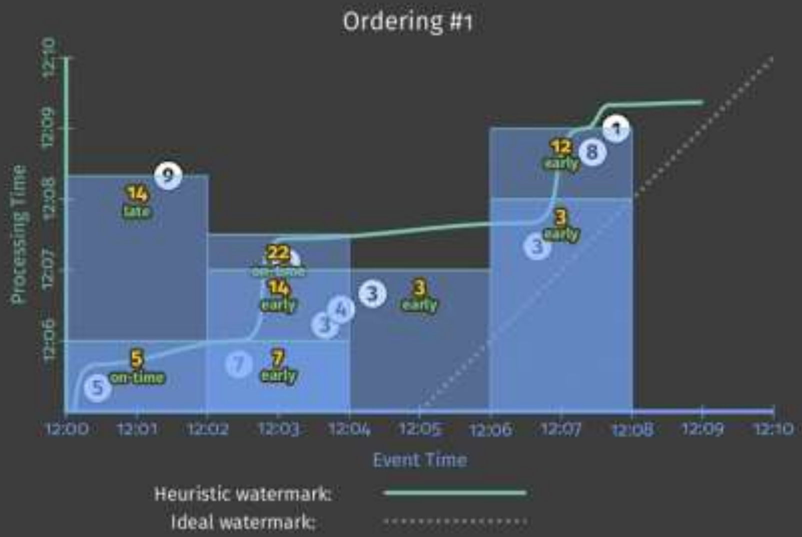


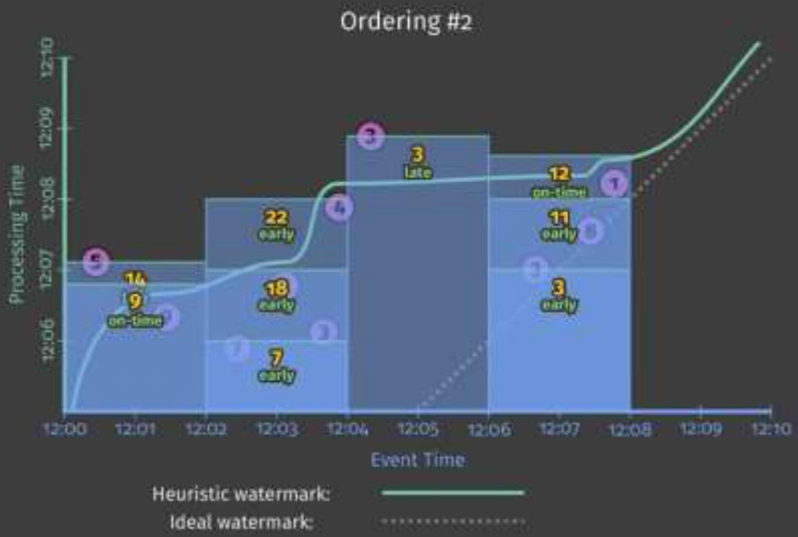
Processing Time Results Differ





Event Time Results are Stable





What / Where / When / How

Correctness

Power

Composability

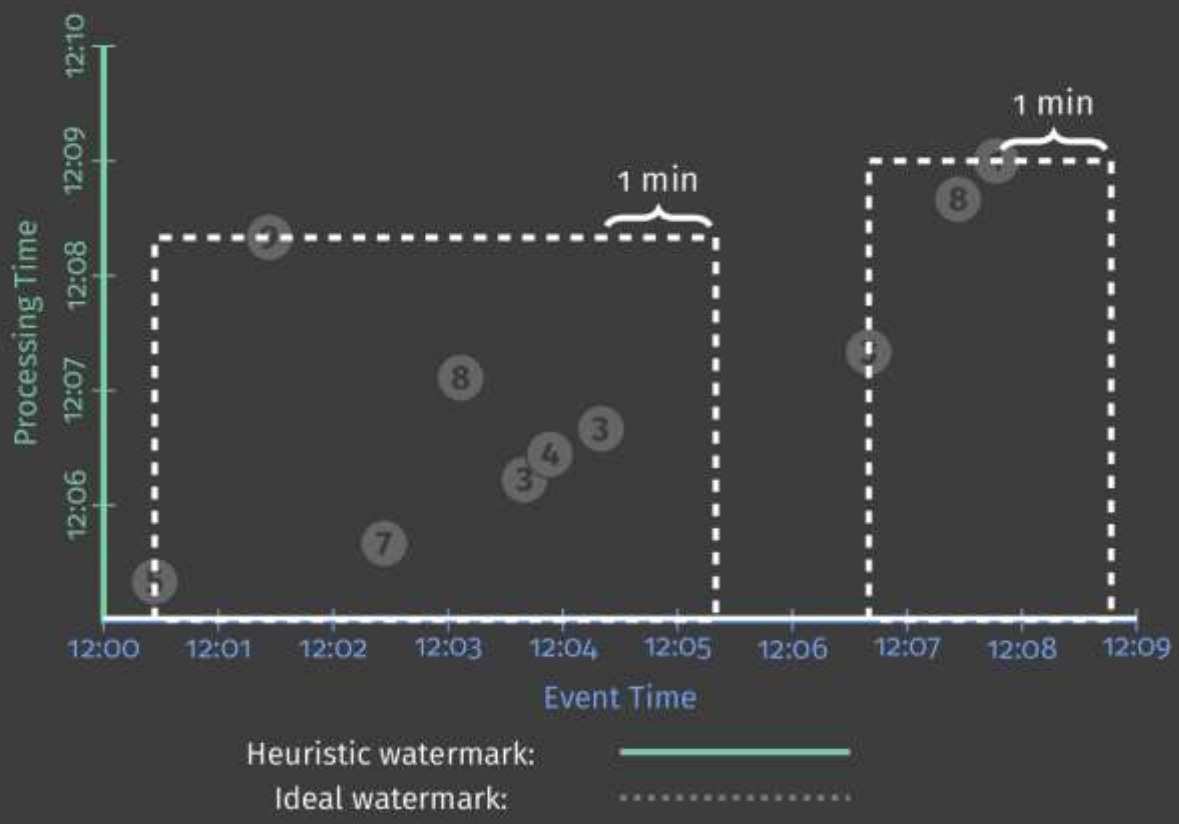
Flexibility

Modularity

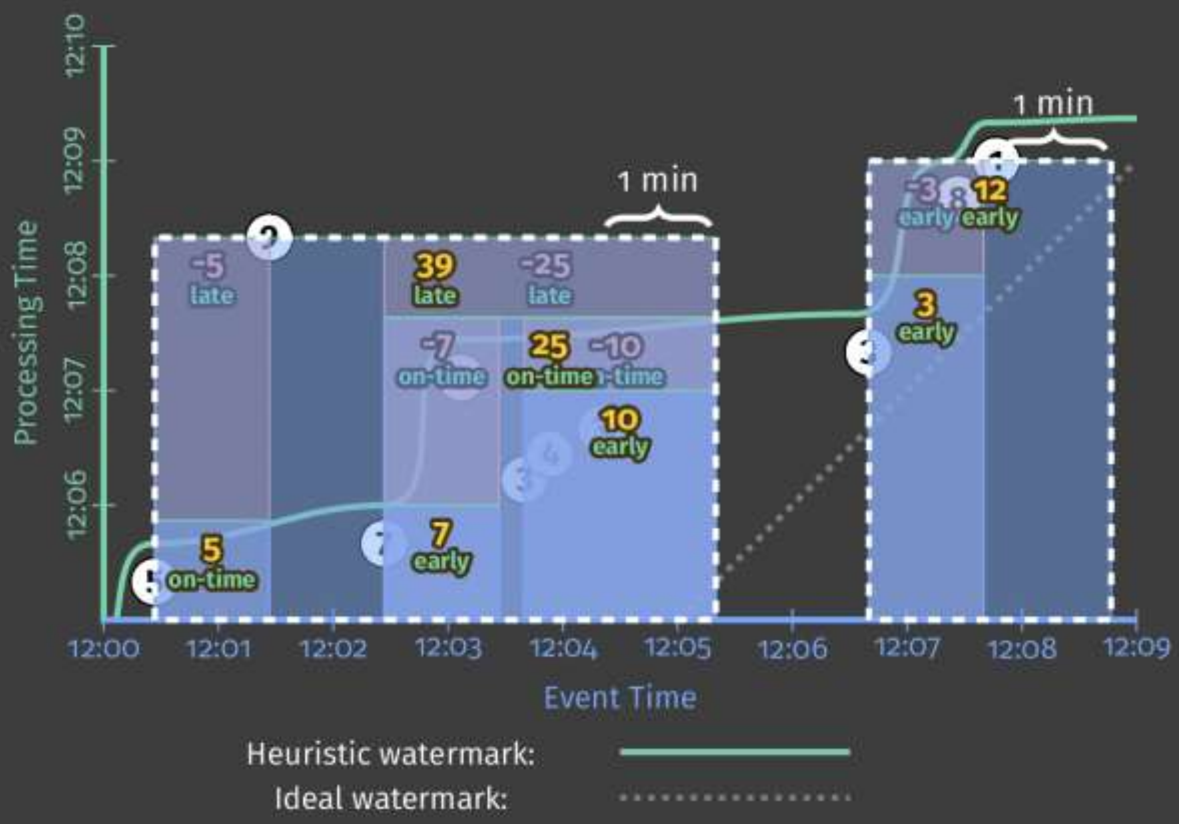
Sessions

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(Sessions.withGapDuration(Minutes(1))
                .triggering(AtWatermark()
                  .withEarlyFirings(AtPeriod(Minutes(1)))
                  .withLateFirings(AtCount(1)))
              .accumulatingAndRetractingFiredPanes())
    .apply(Sum.integersPerKey());
```

Identifying Bursts of User Activity



Identifying Bursts of User Activity



What / Where / When / How

Correctness

Power

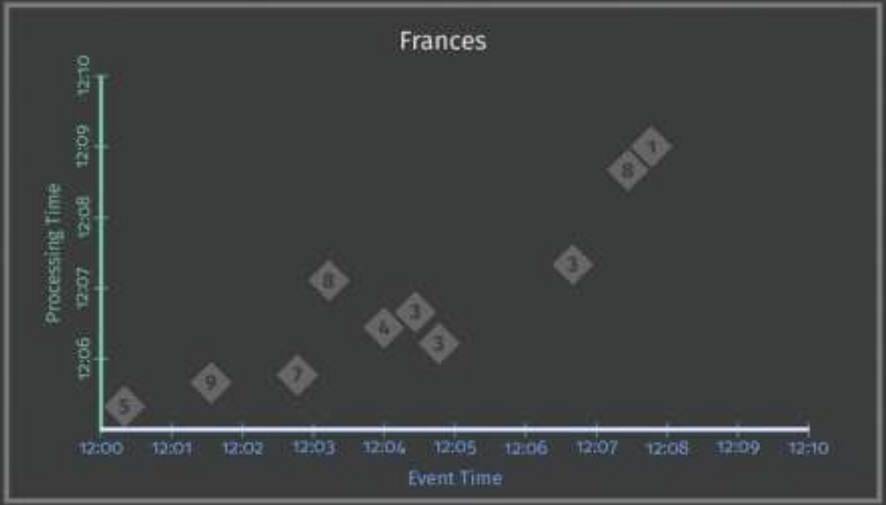
Composability

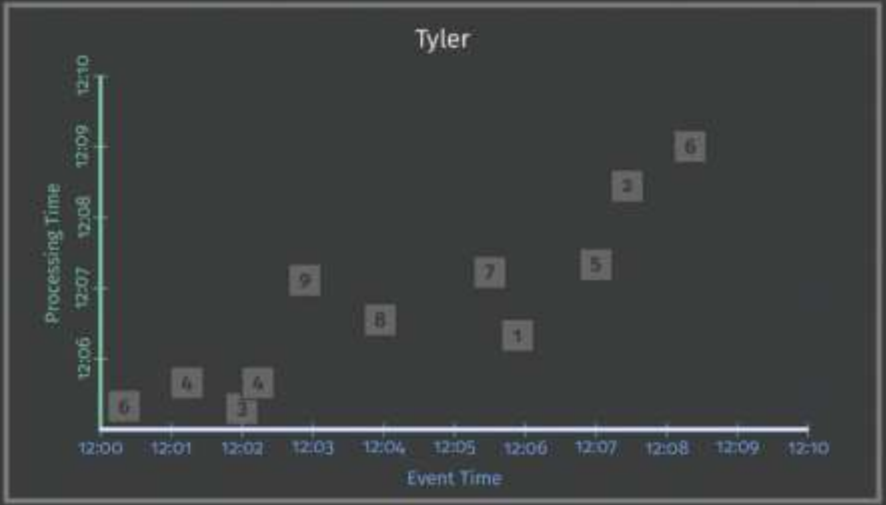
Flexibility

Modularity

Calculating Session Lengths

```
input
  .apply(Window.into(Sessions.withGapDuration(Minutes(1)))
               .trigger(AtWatermark())
               .discardingFiredPanes())
  .apply(CalculateWindowLength()));
```


Frances
Processing Time
Event Time
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
12:10
5
9
7
8
4
3
3
3
8
1


Tyler
Processing Time
Event Time
12:00
12:01
12:02
12:03
12:04
12:05
12:06
12:07
12:08
12:09
12:10
6
4
3
4
9
8
7
1
5
2
6

Calculating the Average Session Length

```
input
  .apply(Window.into(Sessions.withGapDuration(Minutes(1)))
               .trigger(AtWatermark())
               .discardingFiredPanes())
  .apply(CalculateWindowLength()));
```

```
  .apply(Window.into(FixedWindows.of(Minutes(2)))
               .trigger(AtWatermark())
                 .withEarlyFirings(AtPeriod(Minutes(1)))
               .accumulatingFiredPanes())
  .apply(Mean.globally());
```

This image cannot currently be displayed.

What / Where / When / How

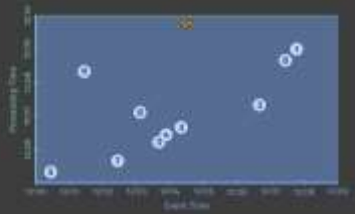
Correctness

Power

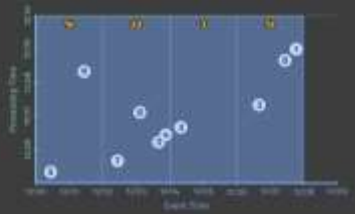
Composability

Flexibility

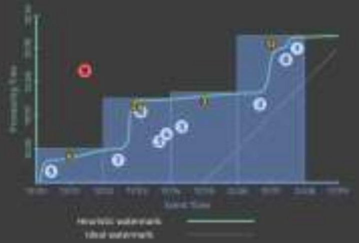
Modularity



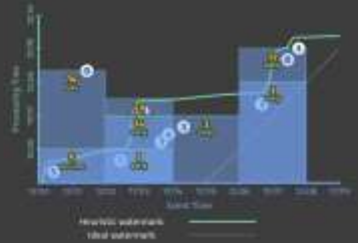
1.Classic Batch



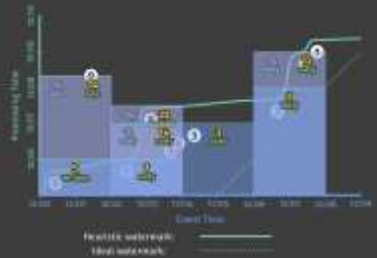
2. Batch with Fixed Windows



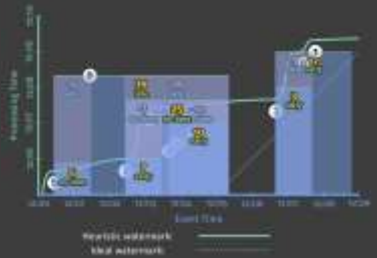
3. Streaming



4. Streaming with Speculative + Late Data



5. Streaming With Retractions



6. Sessions

What / Where / When / How

Correctness

Power

Composability

Flexibility

Modularity

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

1.Classic Batch

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

2. Batch with Fixed Windows

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

3. Streaming

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

4. Streaming with Speculative + Late Data

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

5. Streaming With Retractions

```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(Sessions.withGapDuration(Minutes(2))
             .triggering(AtWatermark()
               .withEarlyFirings(AtPeriod(Minutes(1)))
               .withLateFirings(AtCount(1)))
             .accumulatingAndRetractingFiredPanes())
    .apply(Sum.integersPerKey());
```

6. Sessions

What / Where / When / How

Correctness

Power

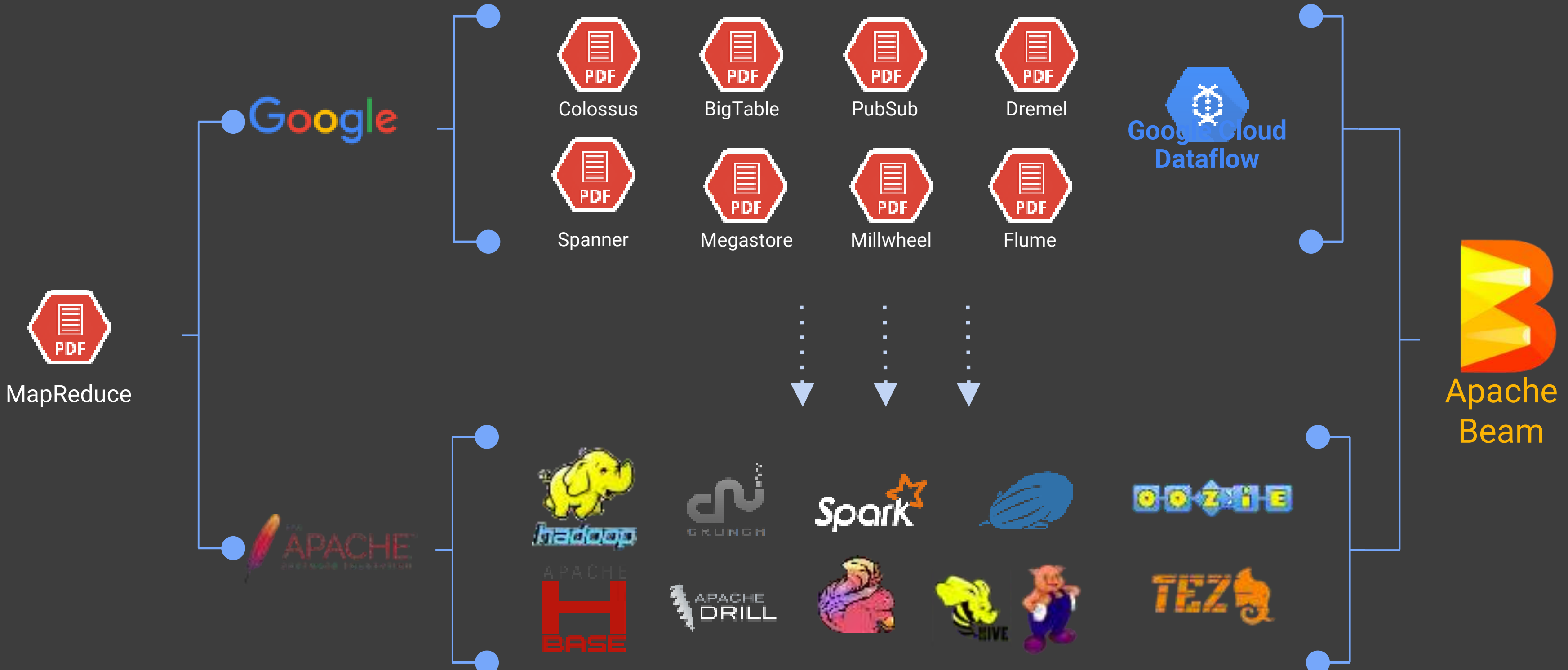
Composability

Flexibility

Modularity

4 Apache Beam (incubating)

The Evolution of Beam

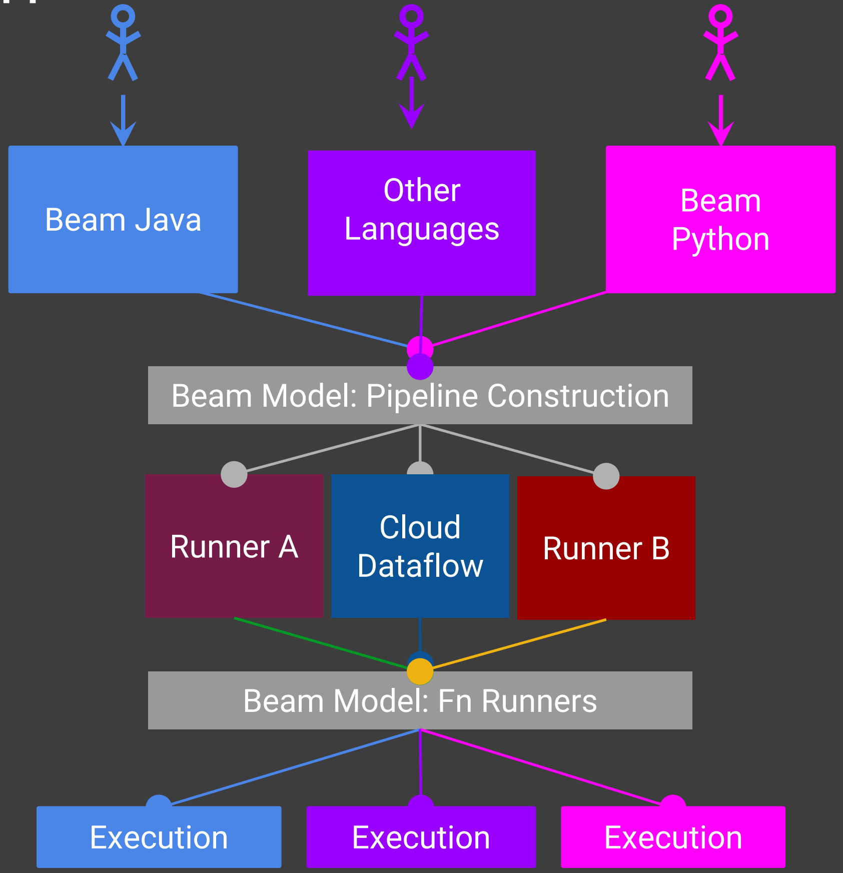


What is Part of Apache Beam?

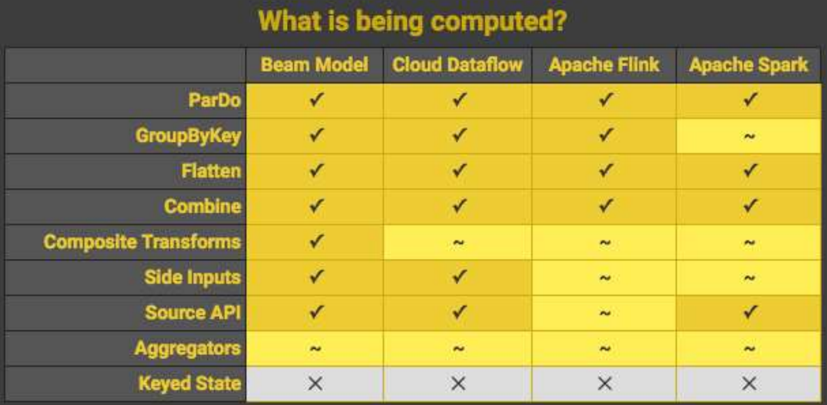
1. The Beam Model: **What** / **Where** / **When** / **How**
2. SDKs for writing Beam pipelines -- starting with Java
3. Runners for Existing Distributed Processing Backends
   - Apache Flink (thanks to data Artisans)
   - Apache Spark (thanks to Cloudera)
   - Google Cloud Dataflow (fully managed service)
   - Local (in-process) runner for testing

Apache Beam Technical Vision

1. **End users:** who want to write pipelines or transform libraries 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

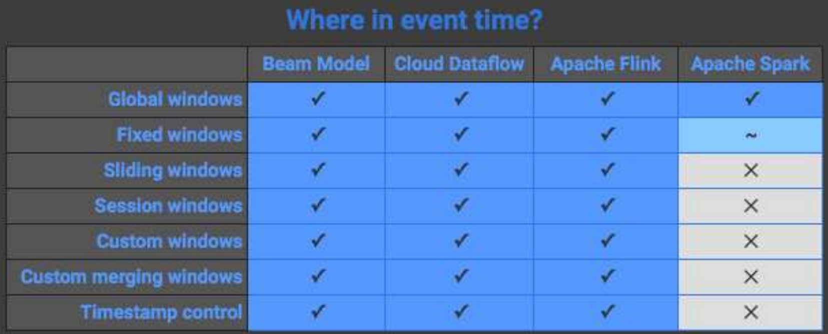


Categorizing Runner Capabilities



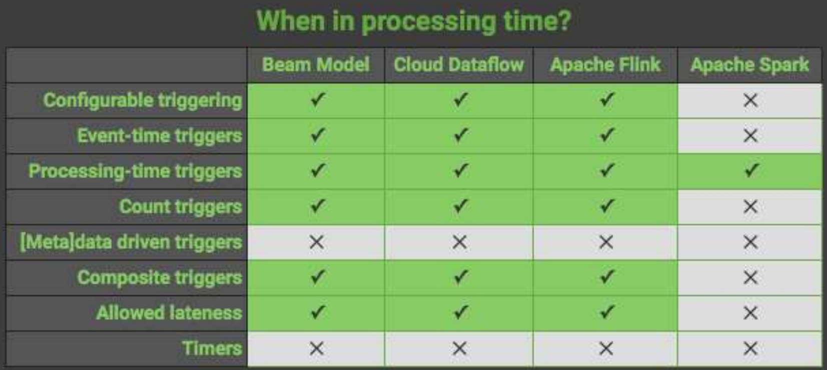
What is being computed?

	Beam Model	Cloud Dataflow	Apache Flink	Apache Spark
ParDo	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~
Flatten	✓	✓	✓	✓
Combine	✓	✓	✓	✓
Composite Transforms	✓	~	~	~
Side Inputs	✓	✓	~	~
Source API	✓	✓	~	✓
Aggregators	~	~	~	~
Keyed State	✕	✕	✕	✕



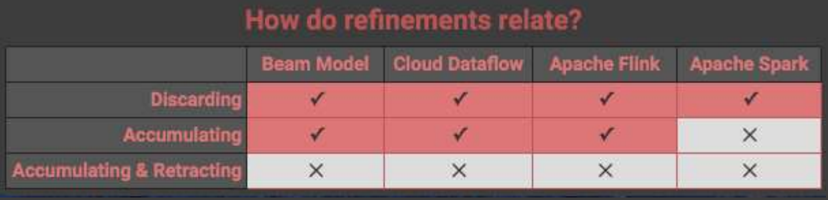
Where in event time?

	Beam Model	Cloud Dataflow	Apache Flink	Apache Spark
Global windows	✓	✓	✓	✓
Fixed windows	✓	✓	✓	~
Sliding windows	✓	✓	✓	✕
Session windows	✓	✓	✓	✕
Custom windows	✓	✓	✓	✕
Custom merging windows	✓	✓	✓	✕
Timestamp control	✓	✓	✓	✕



When in processing time?

	Beam Model	Cloud Dataflow	Apache Flink	Apache Spark
Configurable triggering	✓	✓	✓	✕
Event-time triggers	✓	✓	✓	✕
Processing-time triggers	✓	✓	✓	✓
Count triggers	✓	✓	✓	✕
[Meta]data driven triggers	✕	✕	✕	✕
Composite triggers	✓	✓	✓	✕
Allowed lateness	✓	✓	✓	✕
Timers	✕	✕	✕	✕



How do refinements relate?

	Beam Model	Cloud Dataflow	Apache Flink	Apache Spark
Discarding	✓	✓	✓	✓
Accumulating	✓	✓	✓	✕
Accumulating & Retracting	✕	✕	✕	✕

http://beam.incubator.apache.org/capability-matrix/