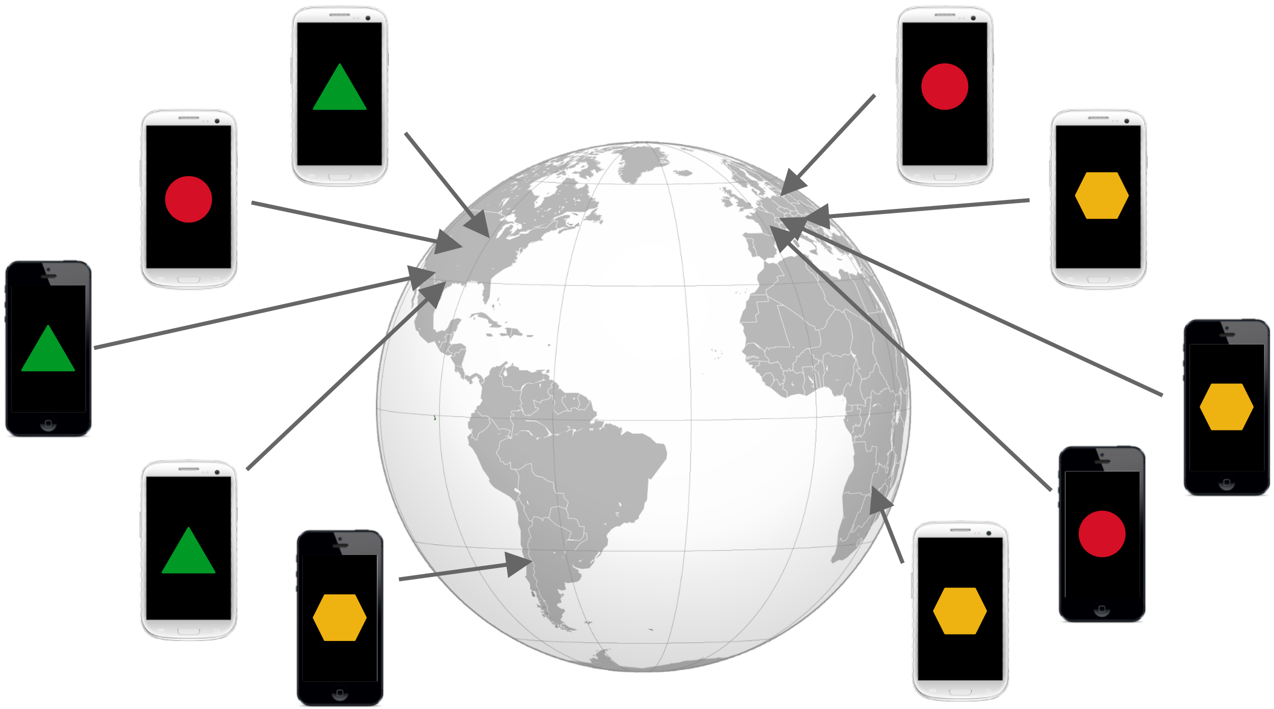
A Generalized Model for Stream Processing
and
Apache Beam

Acknowledgements

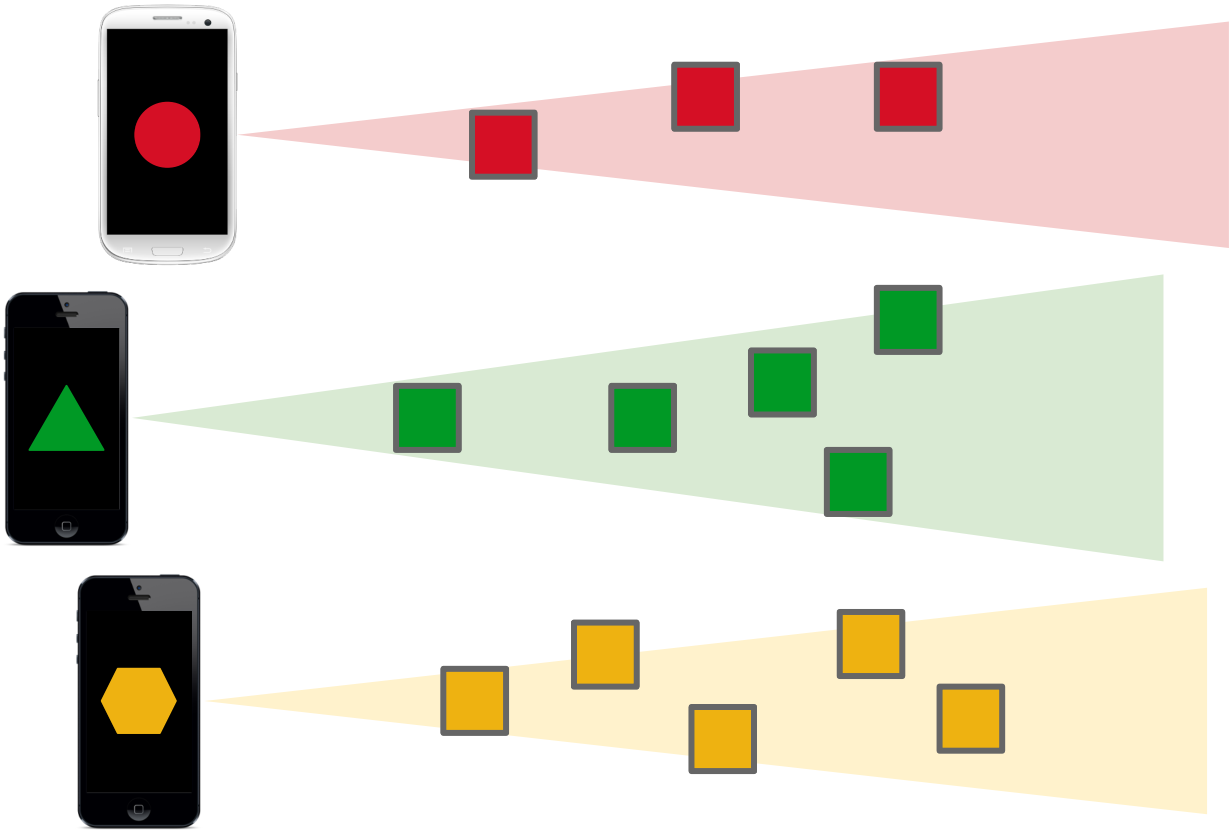
Most of the Slides in this talk have been adapted from the following sources:

- Frances Perry, Tyler Akidau, of Google, Apache Beam Committers, "Fundamentals of Stream Processing with Apache Beam", QCon, San Francisco, Nov. 2016, https://goo.gl/yzvLXe
- Kenneth Knowles of Google, Apache Beam PMC, "Unified, Portable, Efficient Batch and Stream Processing with Apache Beam," Strata San Jose, CA, 2017, https://goo.gl/sRxNxF
- https://2021.beamsummit.org/sessions/state-apache-beam/

How to deal with Infinite, Delayed, Out-of-Order Data Streams (e.g. from Global, Distributed Sources?)

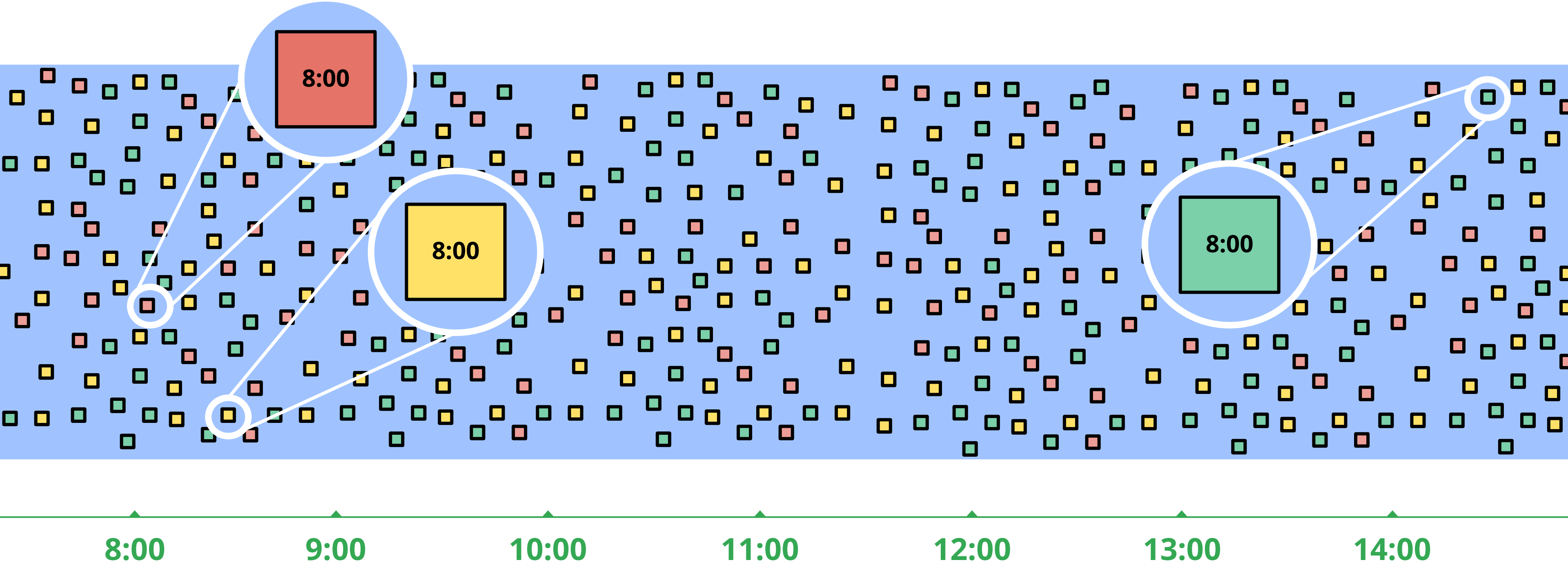


Incoming!

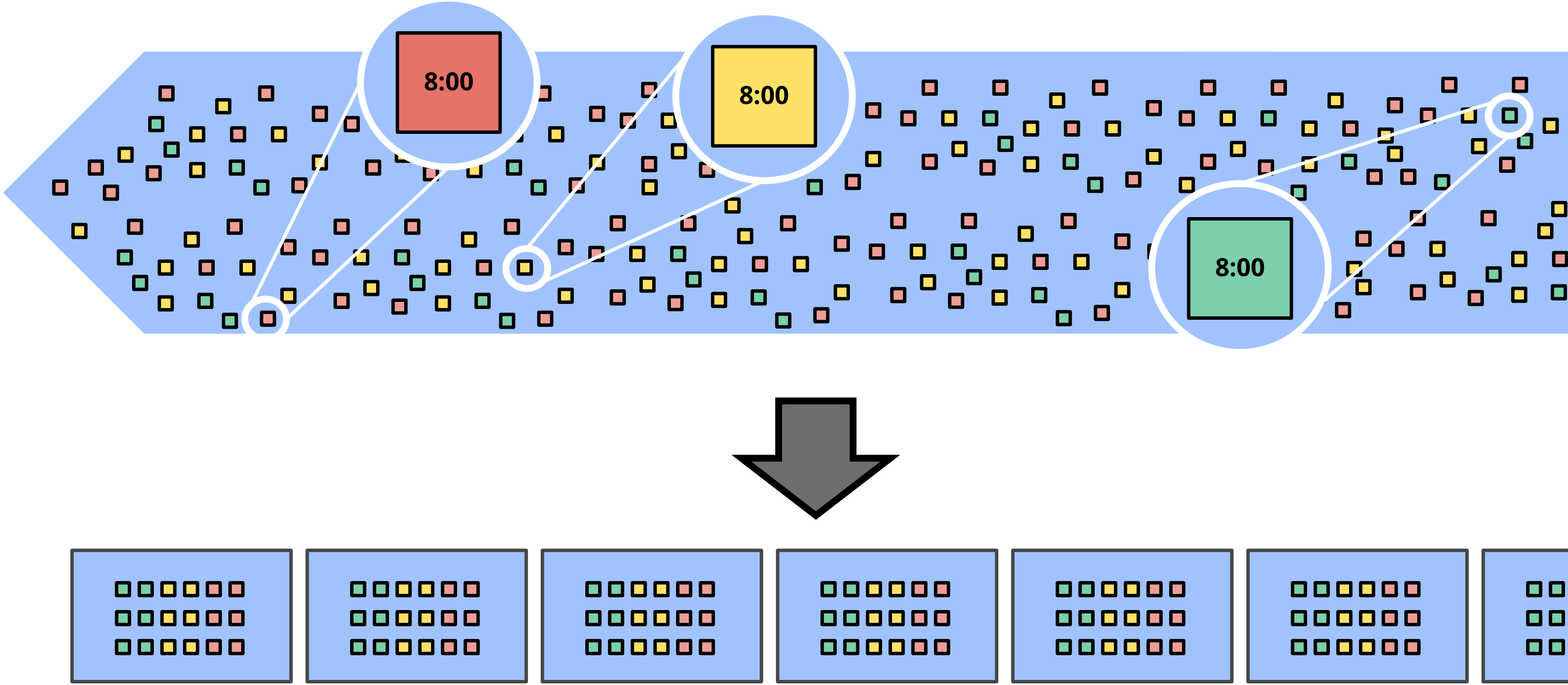


Score per user?

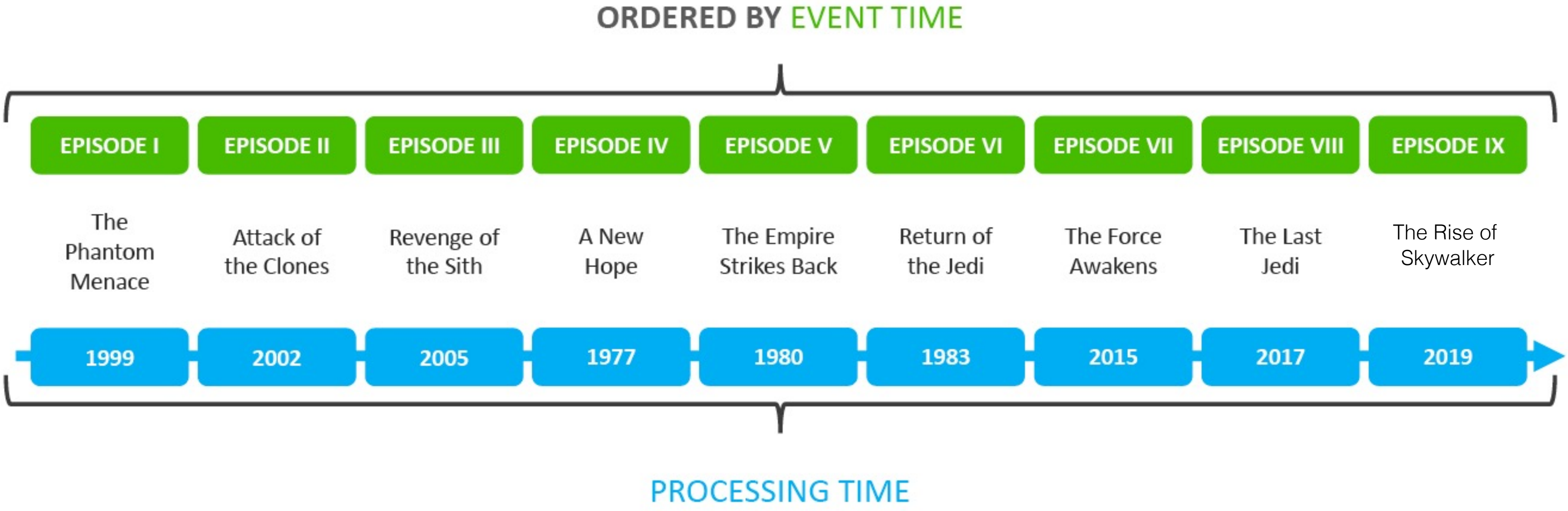
Data Can be Unbounded, Delayed, Out of Order...



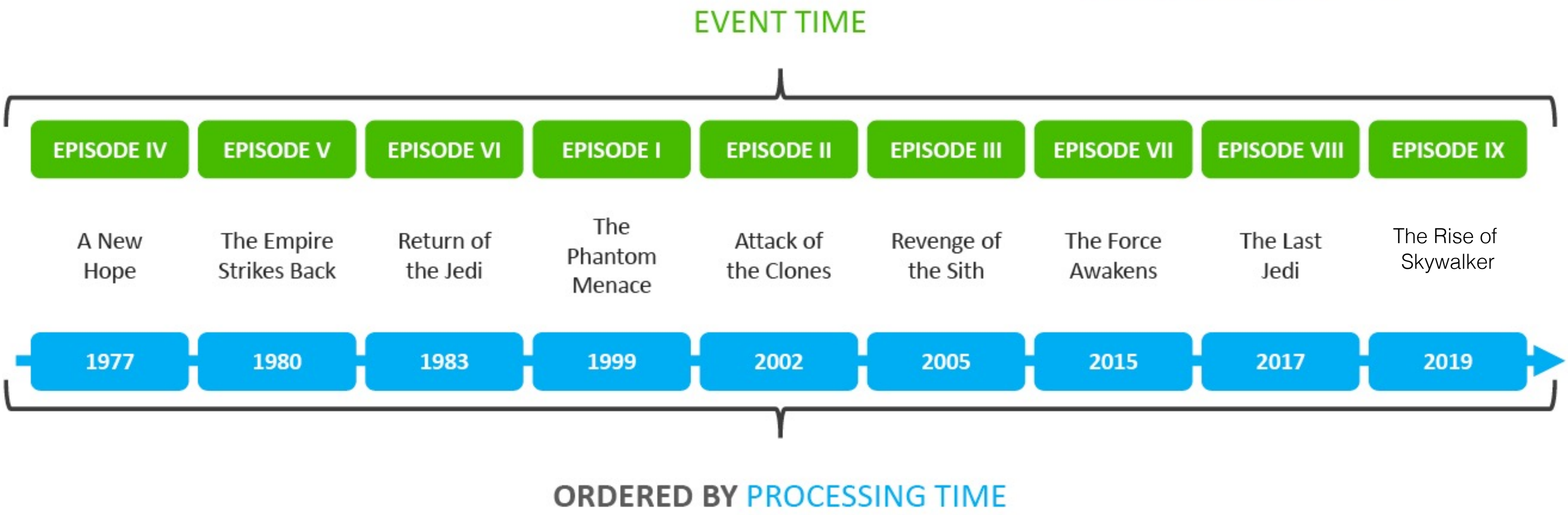
Different Ways to Organize the Data Streams



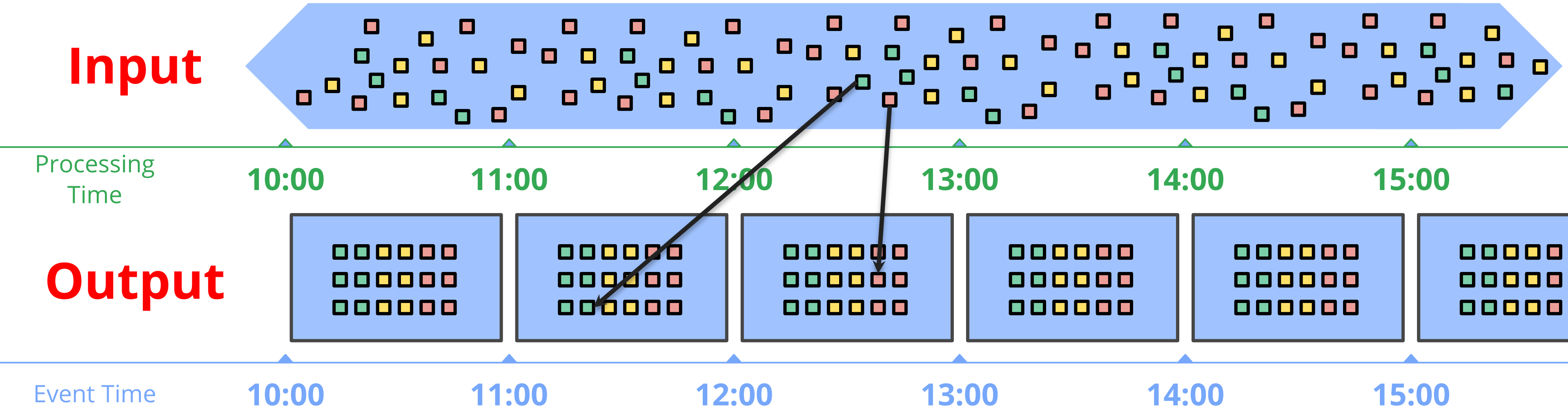
Event Time vs. Processing Time



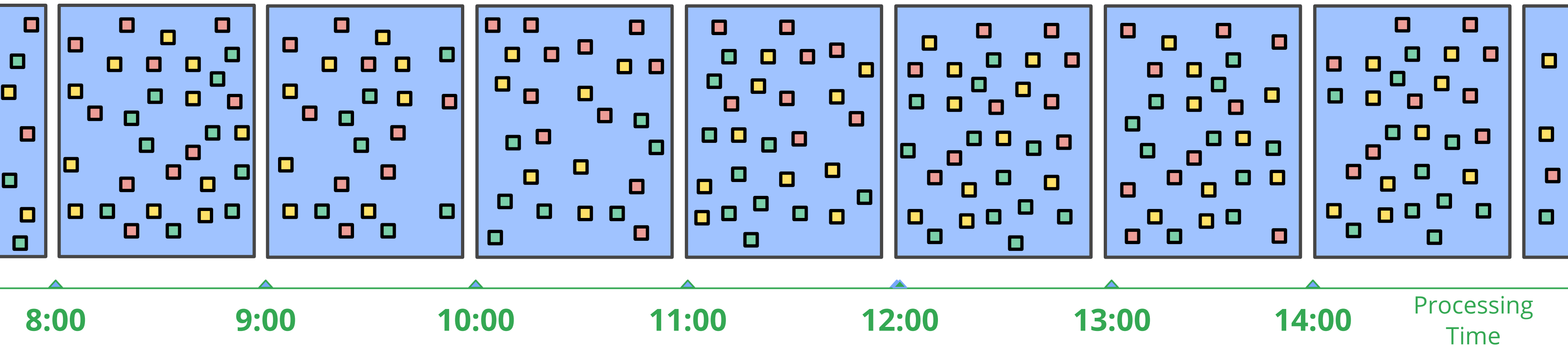
Event Time vs. Processing Time



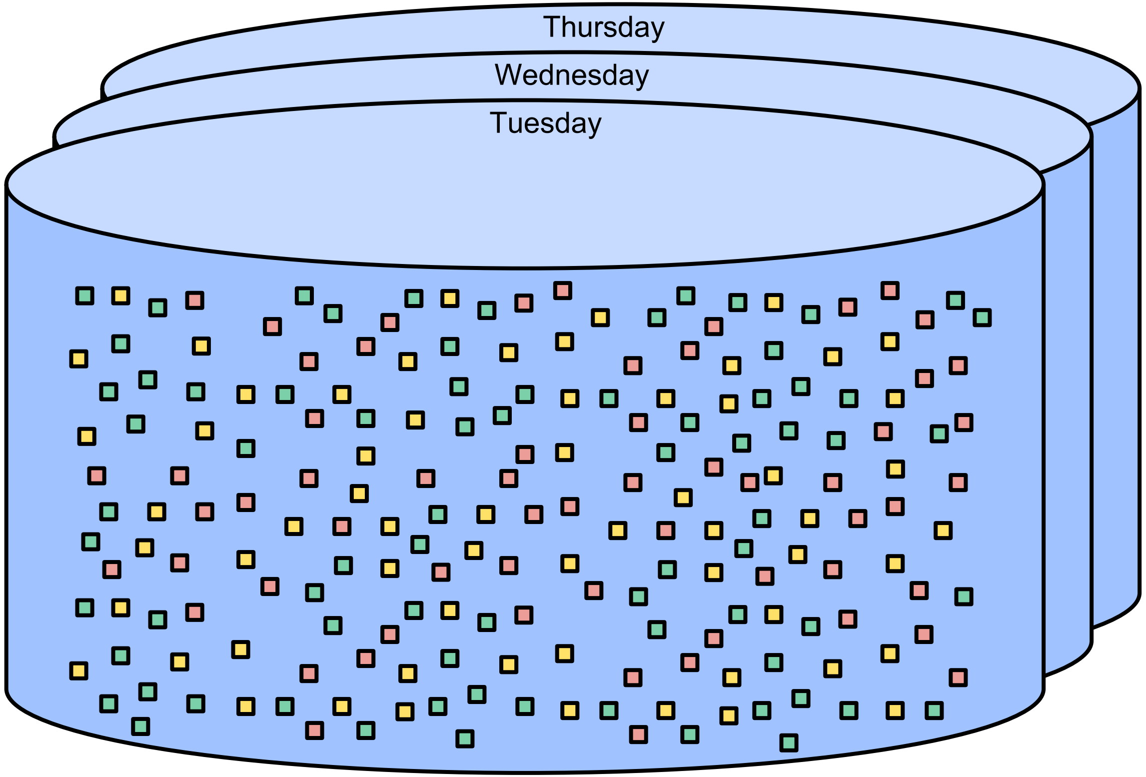
Aggregating via Event-Time Windows



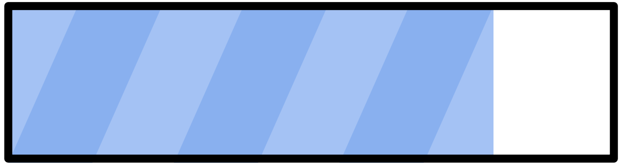
Aggregating via Processing-Time Windows

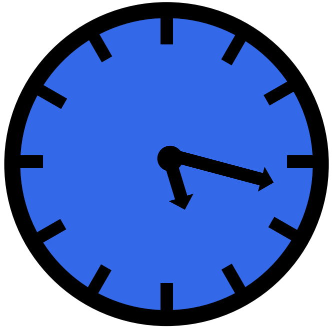


Historical analysis



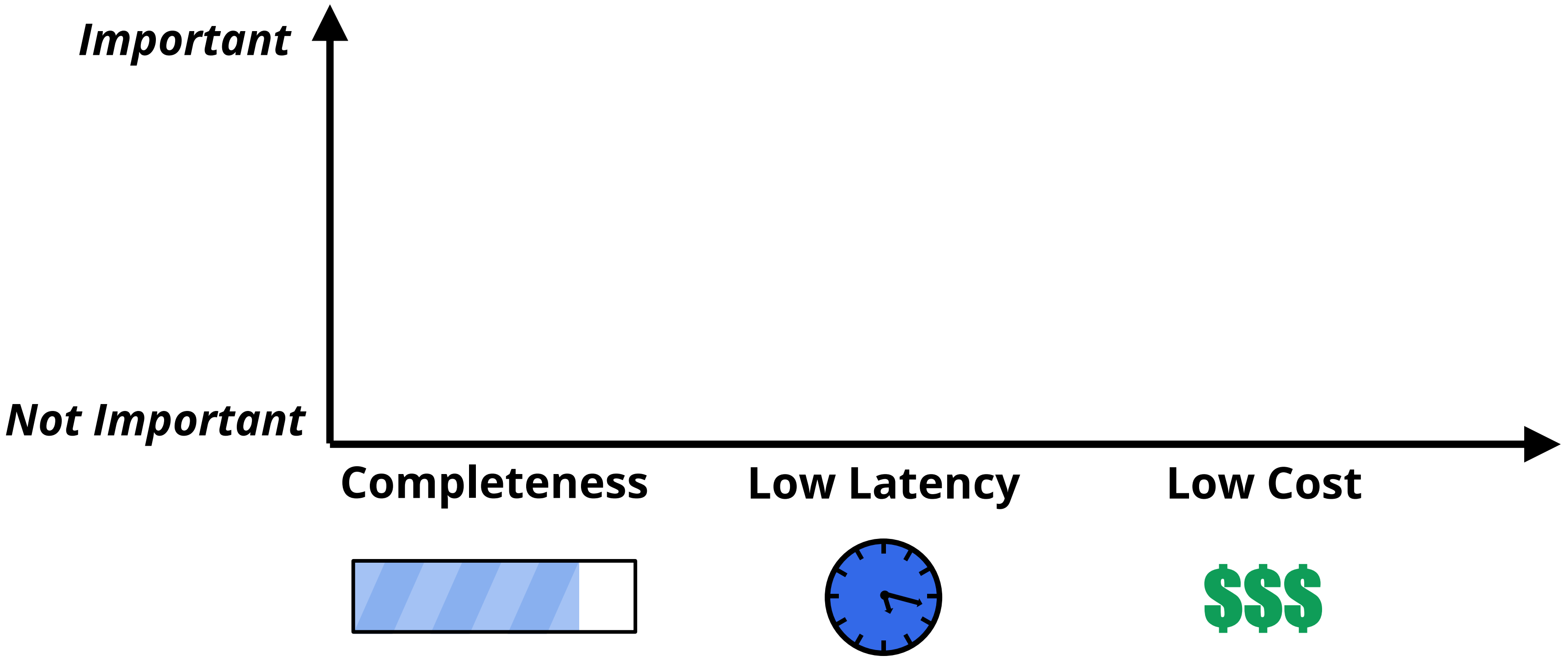
Data Processing Tradeoffs



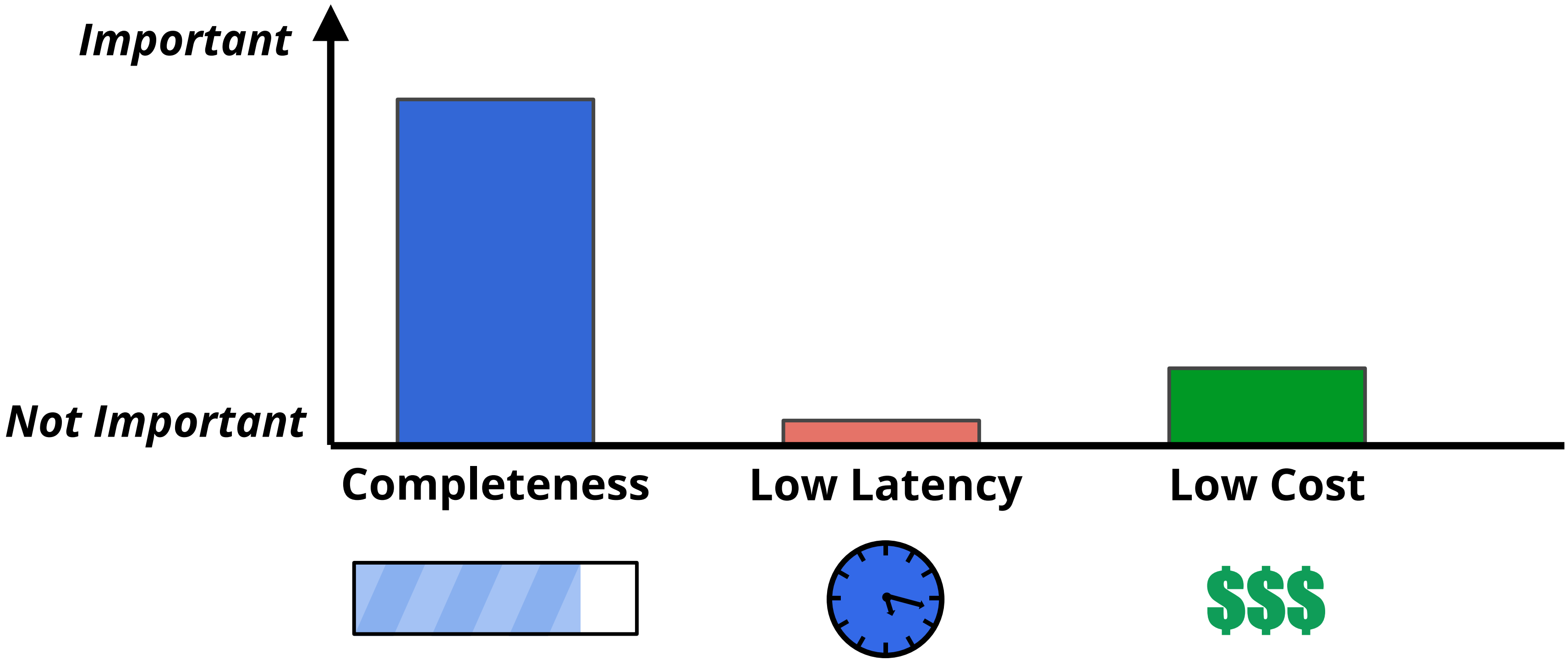


Completeness **Latency** **Cost**

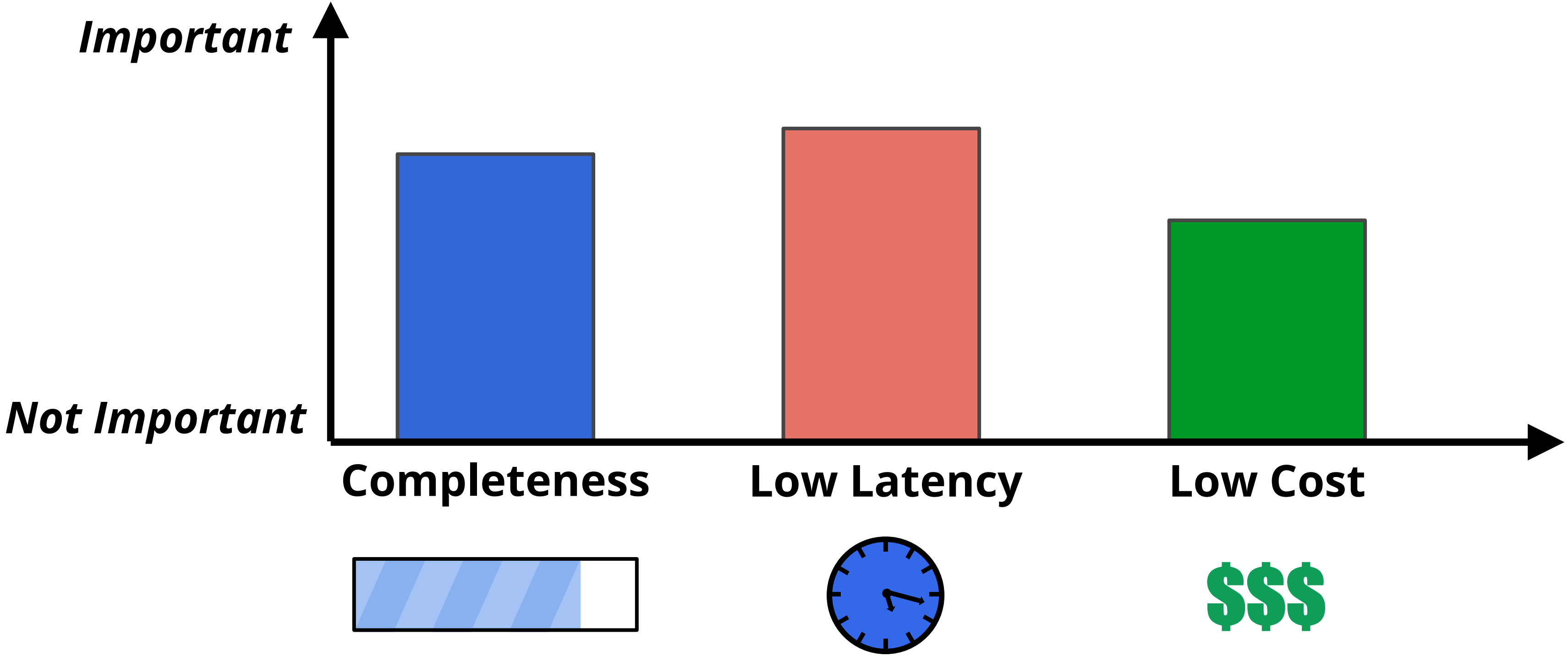
What is important for your application?



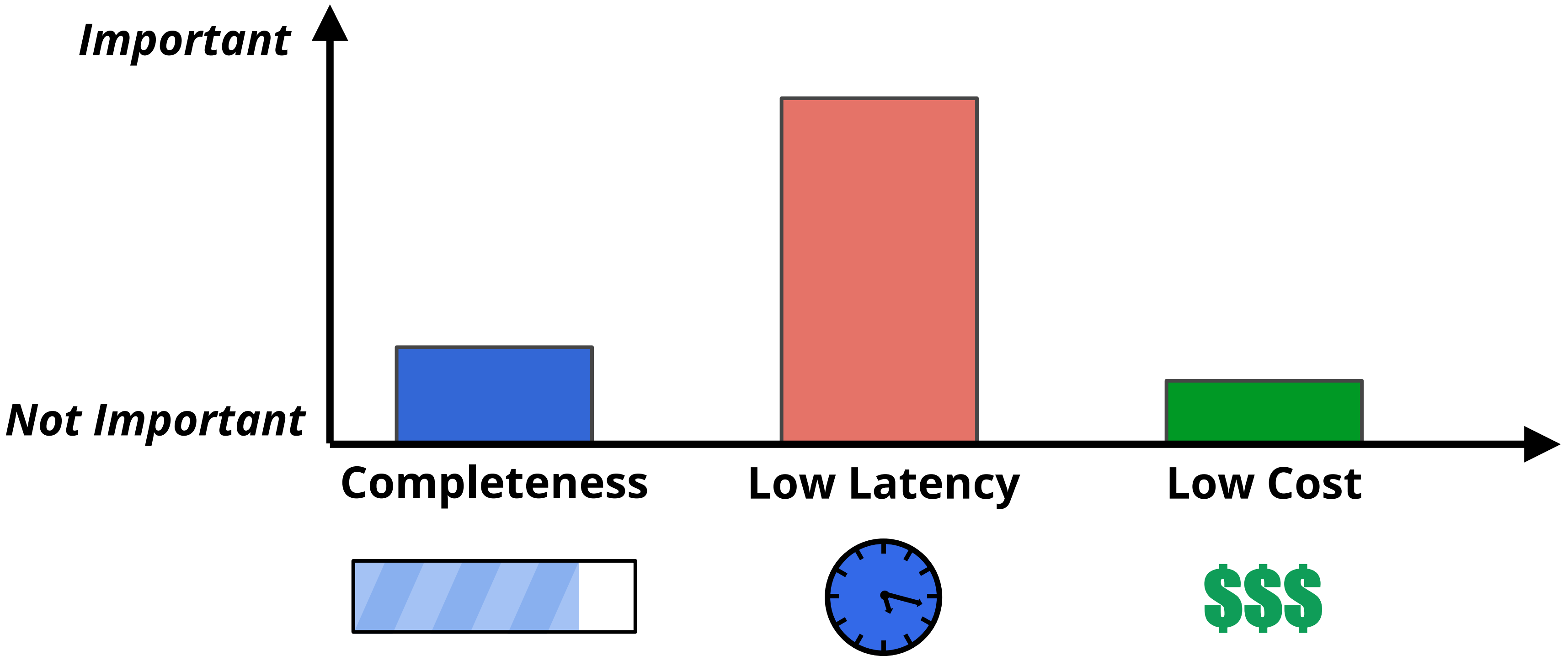
Monthly Billing



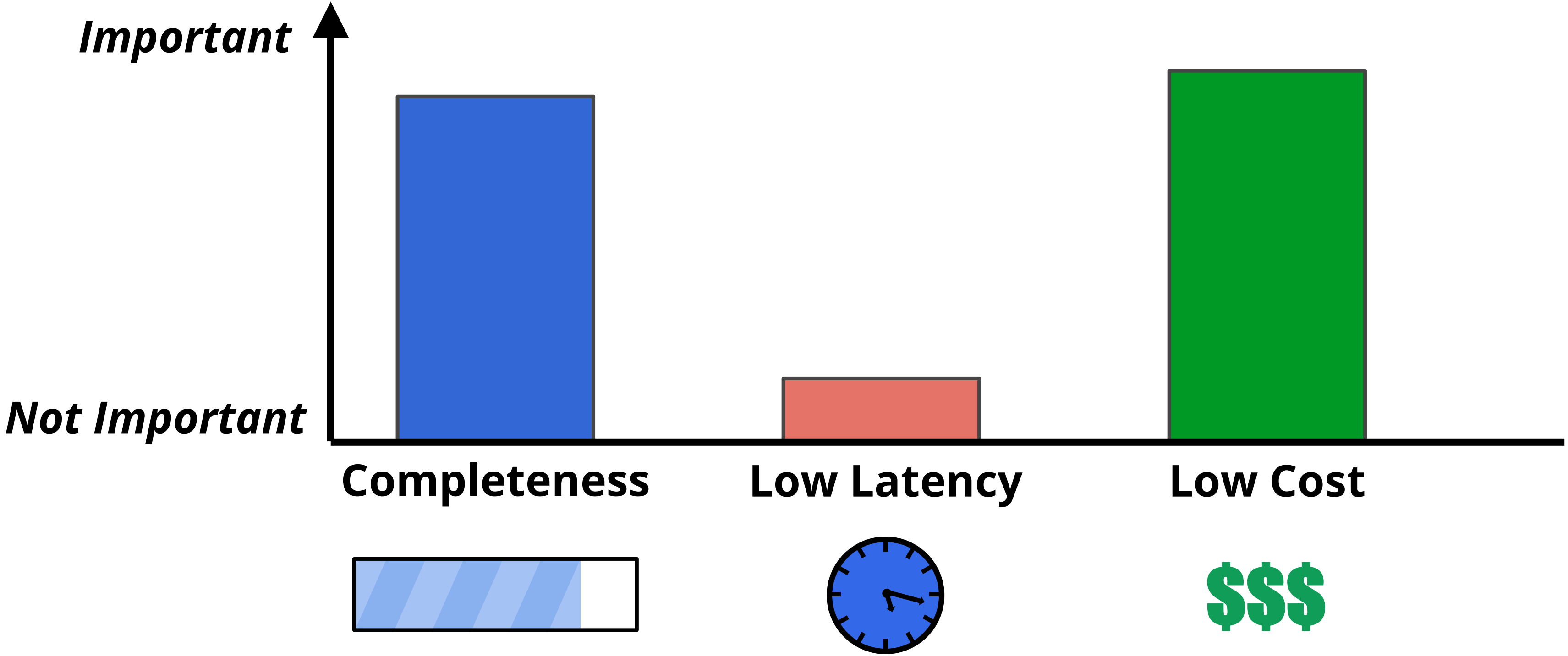
Billing estimate



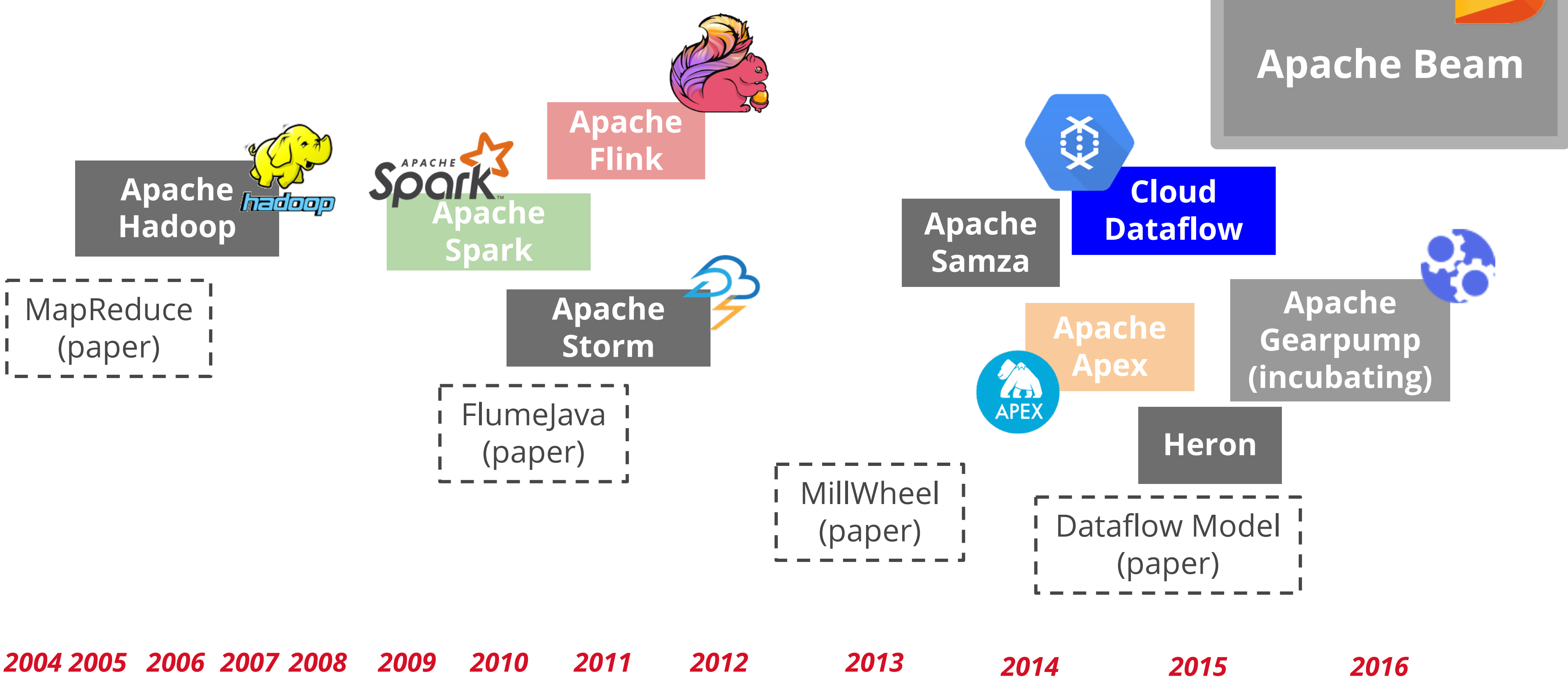
Abuse Detection



Historical Analysis



Choices abound



See also: Tyler Akidau's talk on *Evolution of Massive-Scale Data Processing*

The Generalized Streaming Model (aka the Dataflow/ Beam model)

What are you computing?

Where in event time?

When in processing time are results produced?

How do refinements relate?

The Beam Model: Asking the Right Questions

What are you computing?

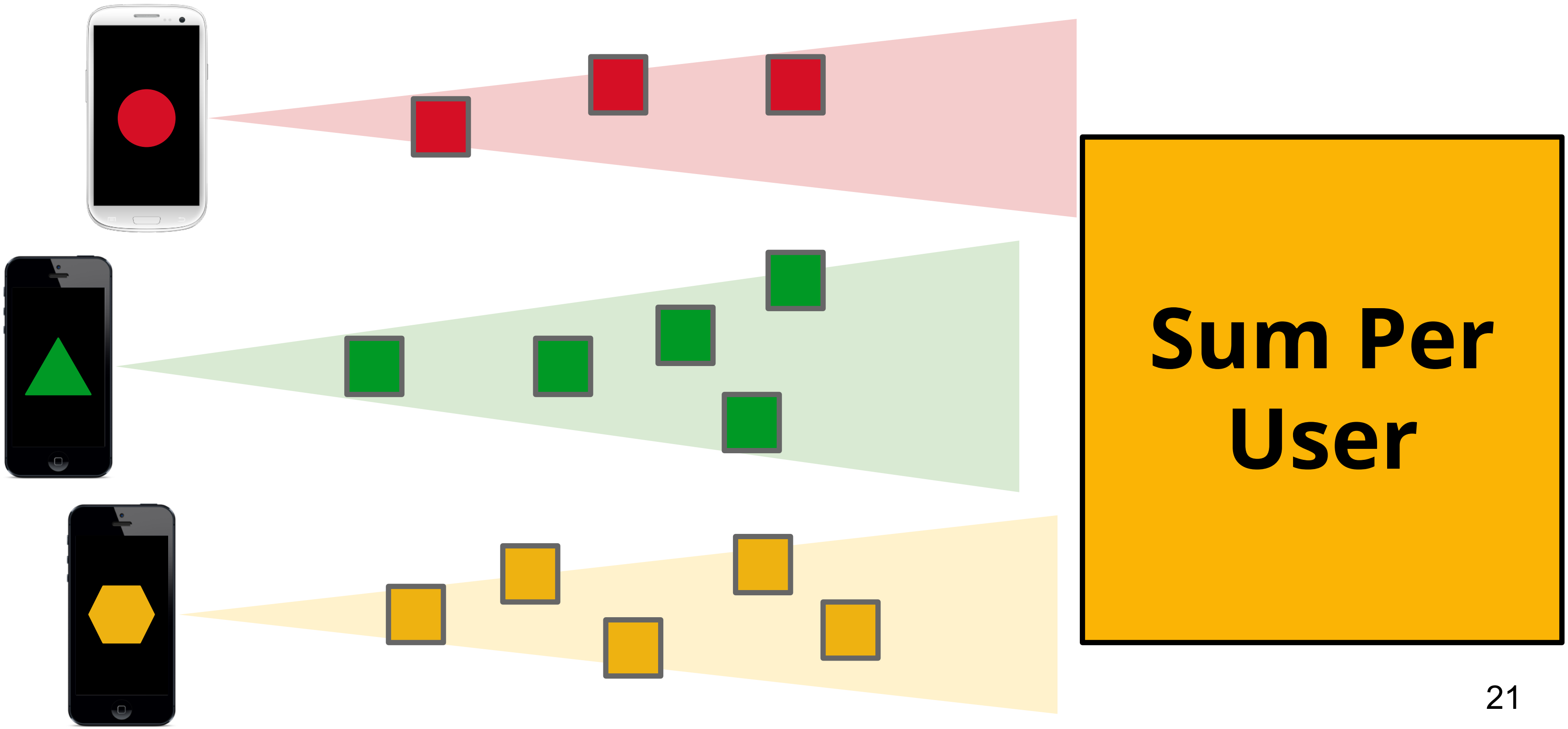
Aggregations, transformations, …

Where in event time?

When in processing time are results produced?

How do refinements relate?

The Beam Model: What are you computing?



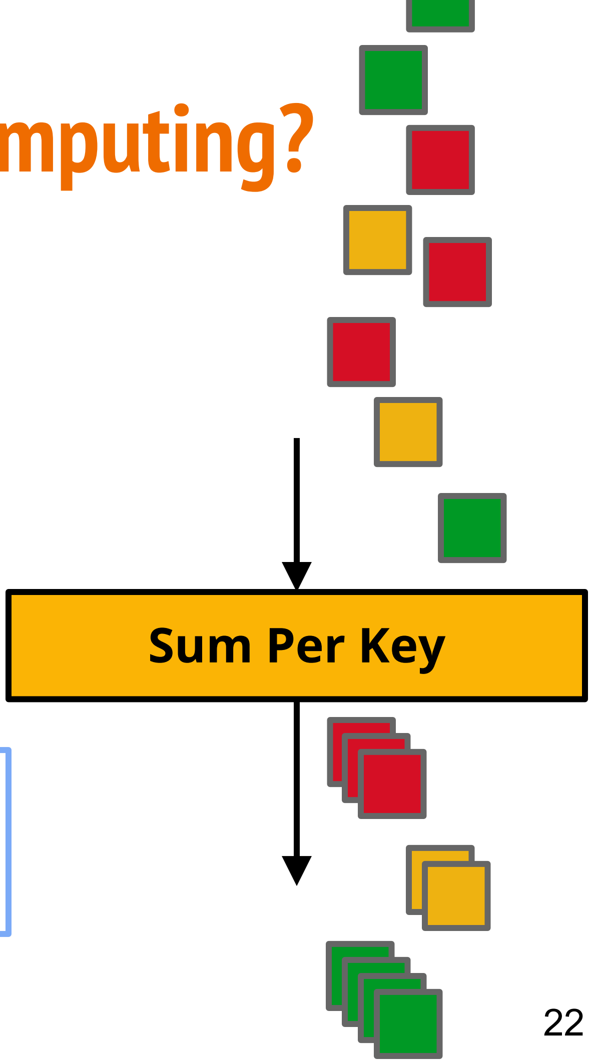
The Beam Model: What are you computing?

Python

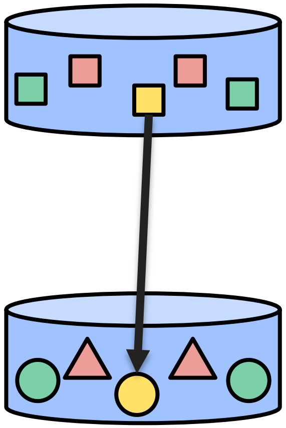
```
input | Sum.PerKey()
      | Write(BigQuerySink(...))
```

Java

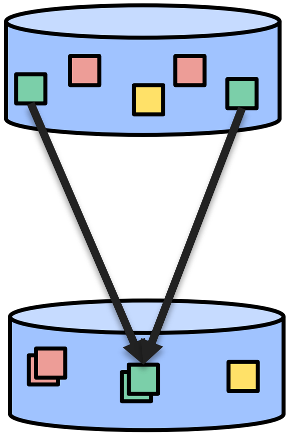
```
input.apply(Sum.integersPerKey())
     .apply(BigQueryIO.Write.to(...));
```



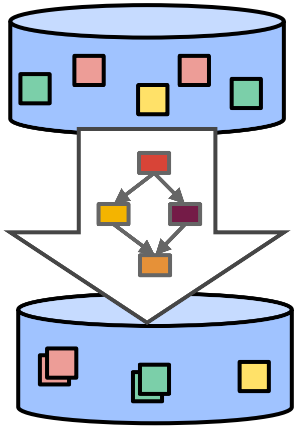
What are you computing?



Element-Wise

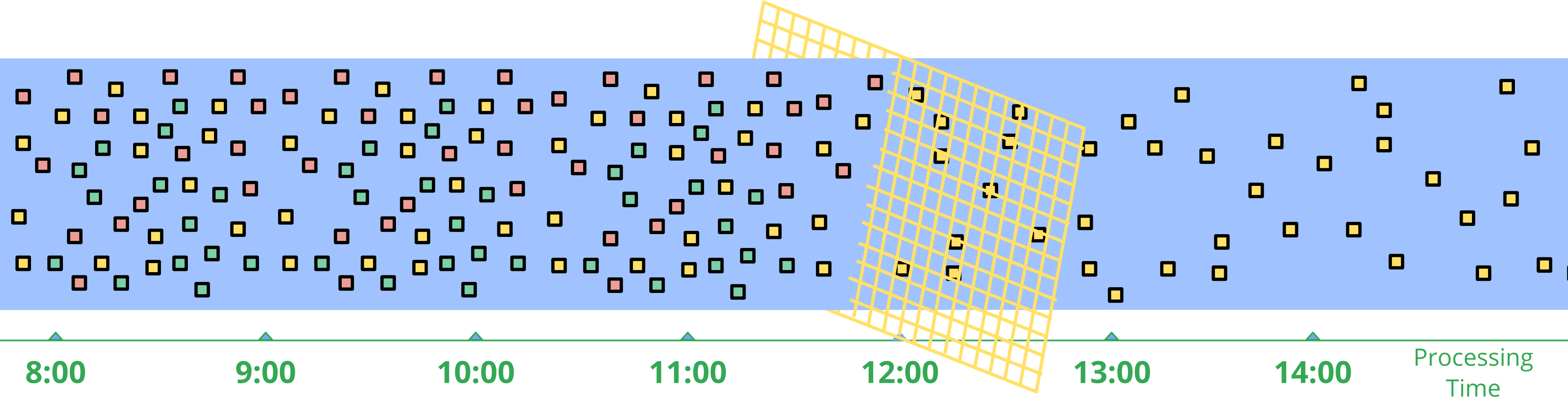


Aggregating



Composite

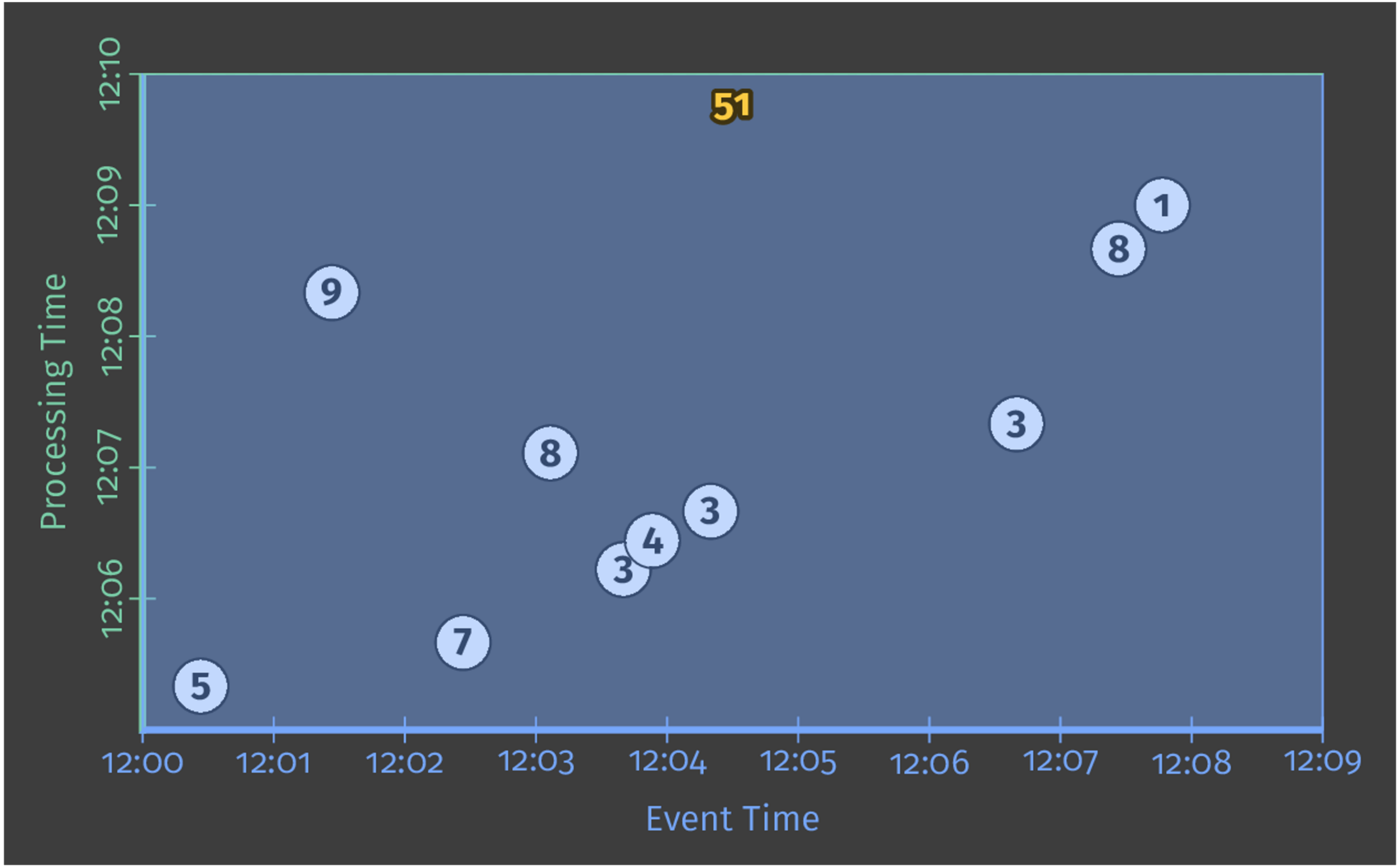
An example: Element-wise transformations



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



The Beam Model: Asking the Right Questions

What are you computing?

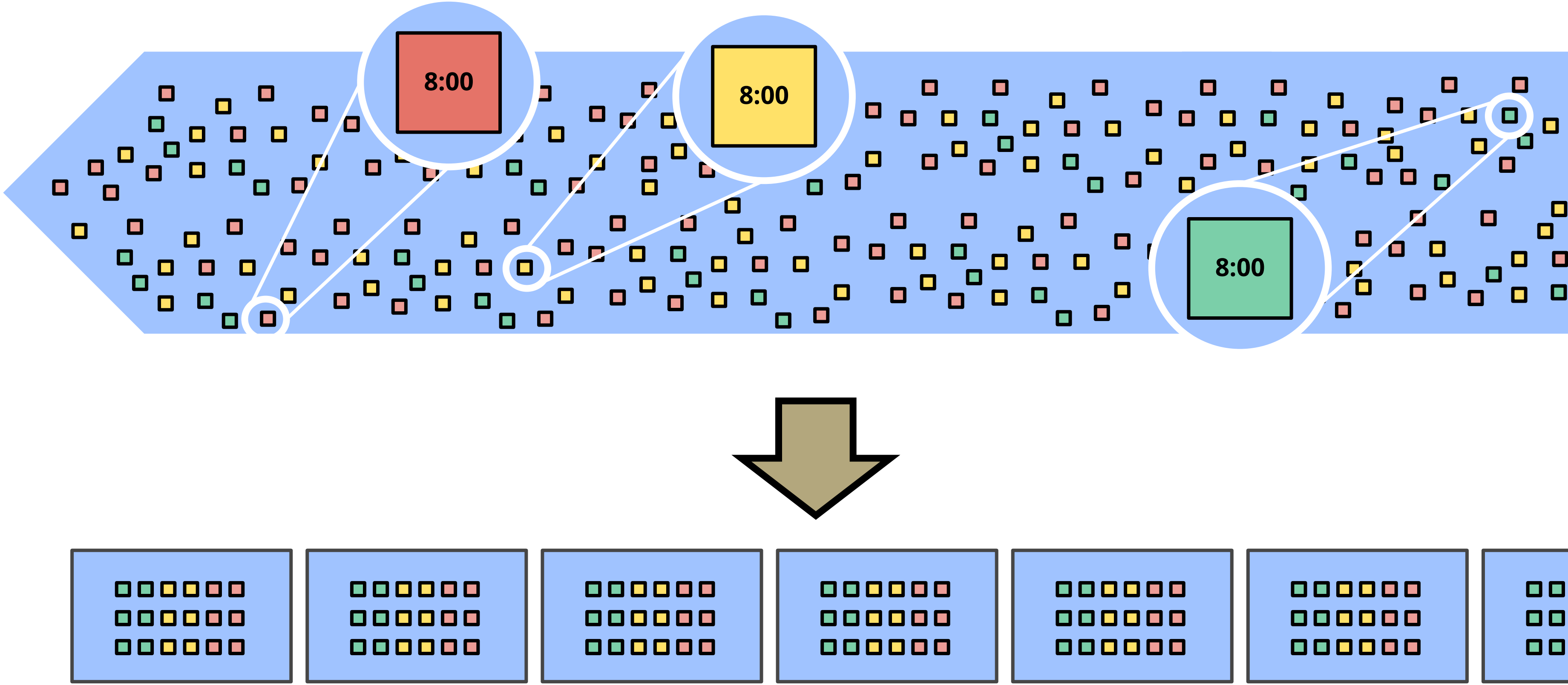
Where in event time?

Event time windowing

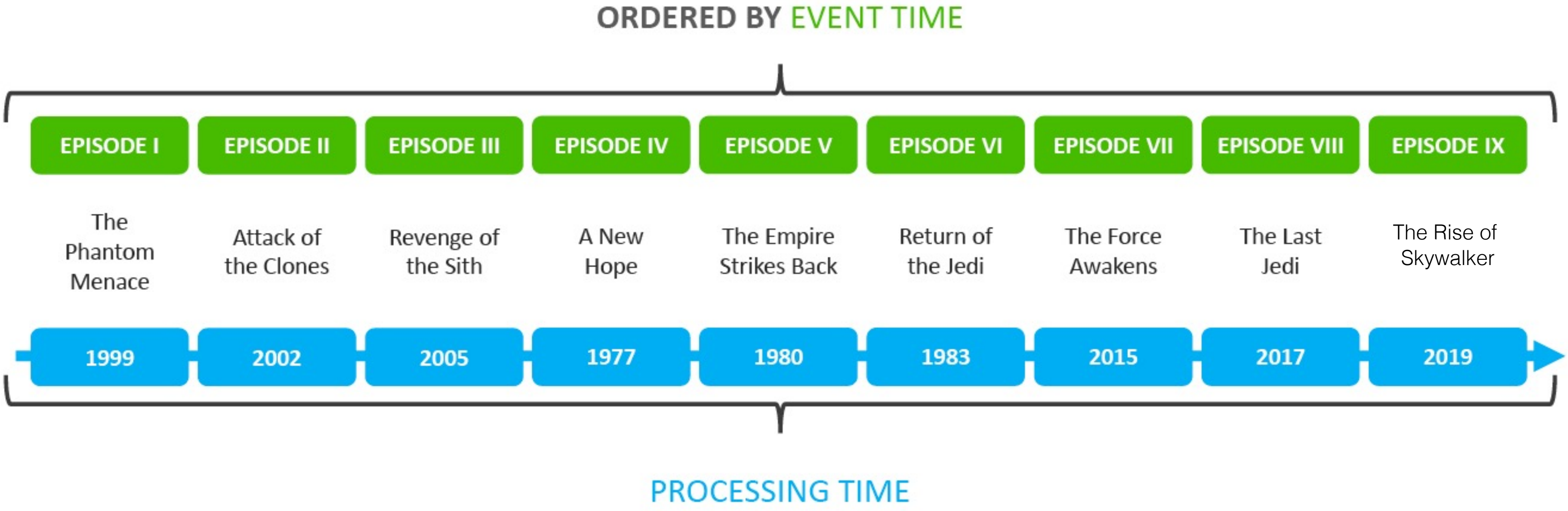
When in processing time are results produced?

How do refinements relate?

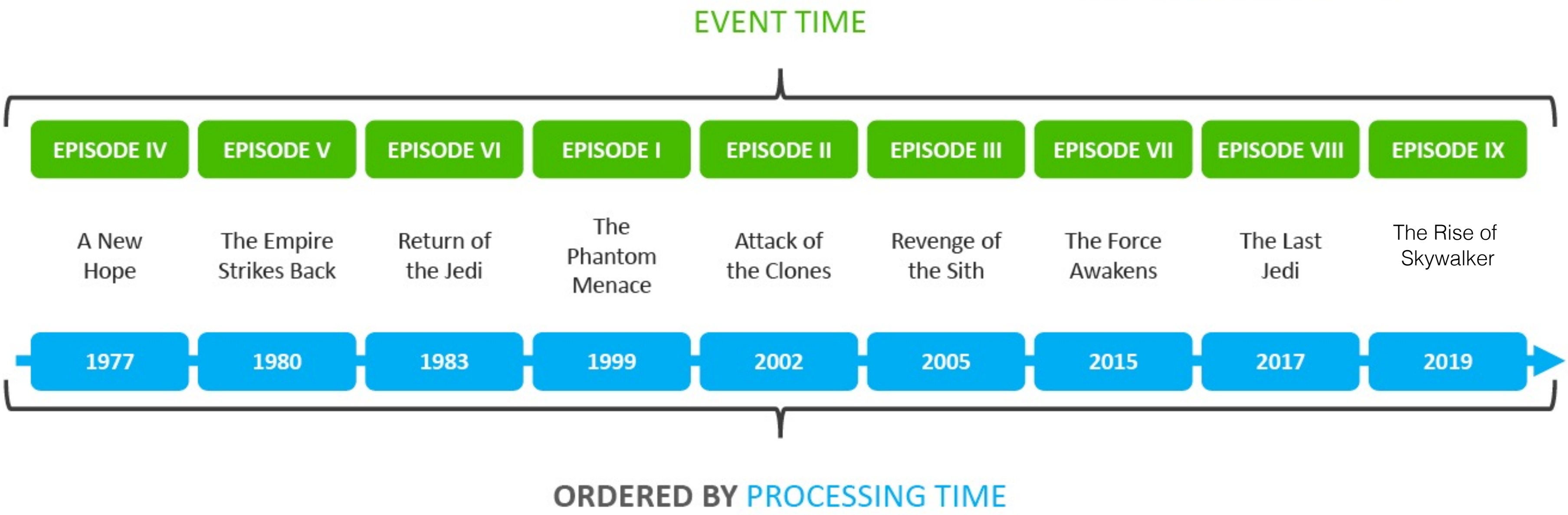
The Beam Model: Where in Event Time?



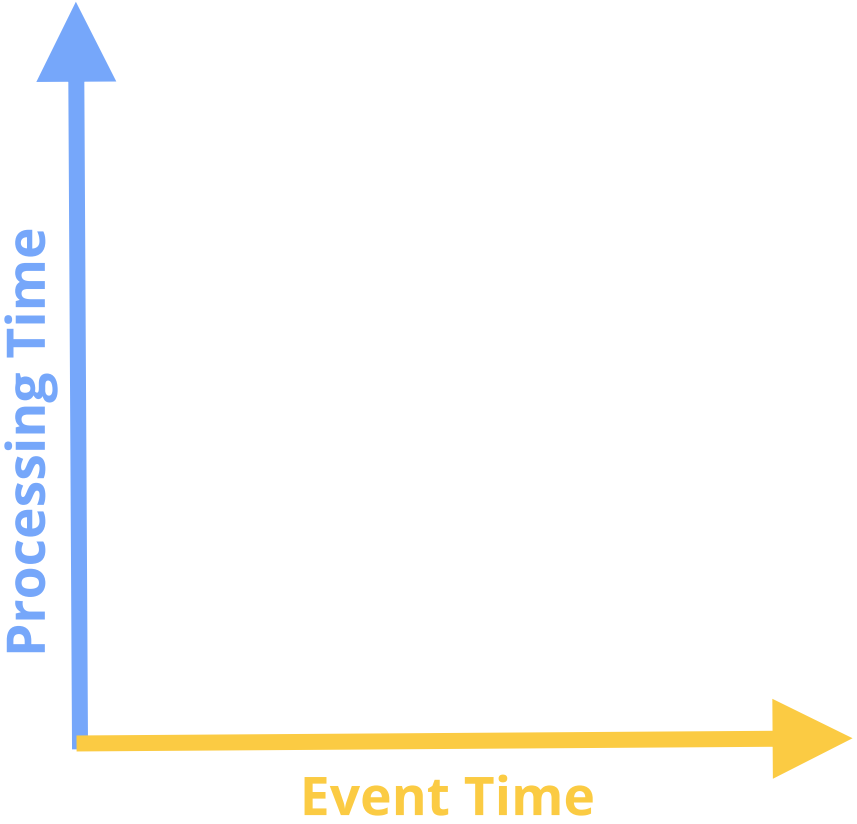
Event Time vs. Processing Time



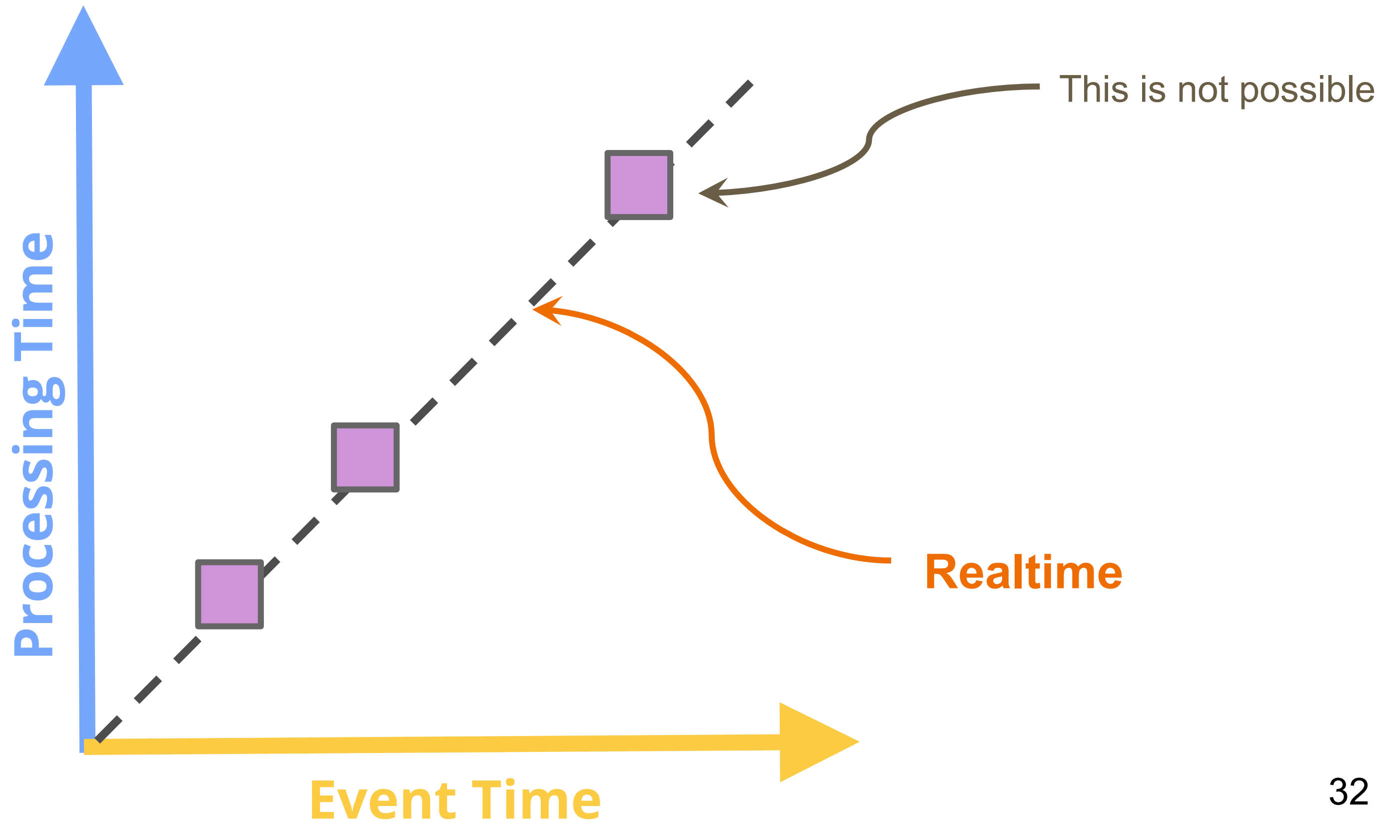
Event Time vs. Processing Time



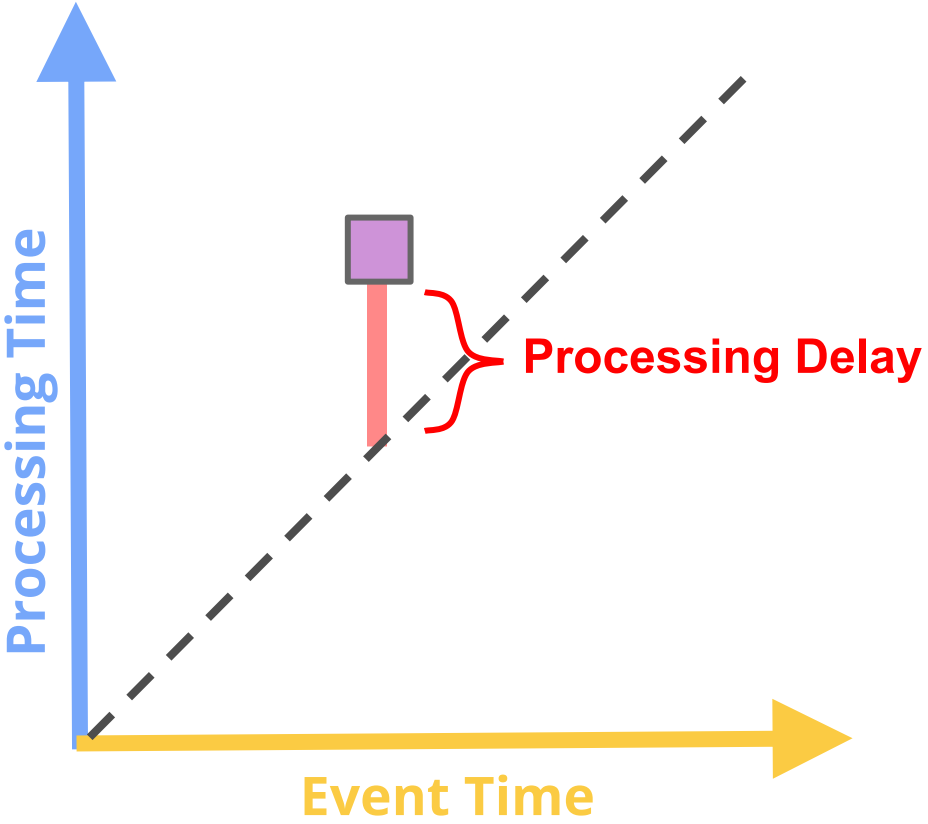
Processing Time vs Event Time



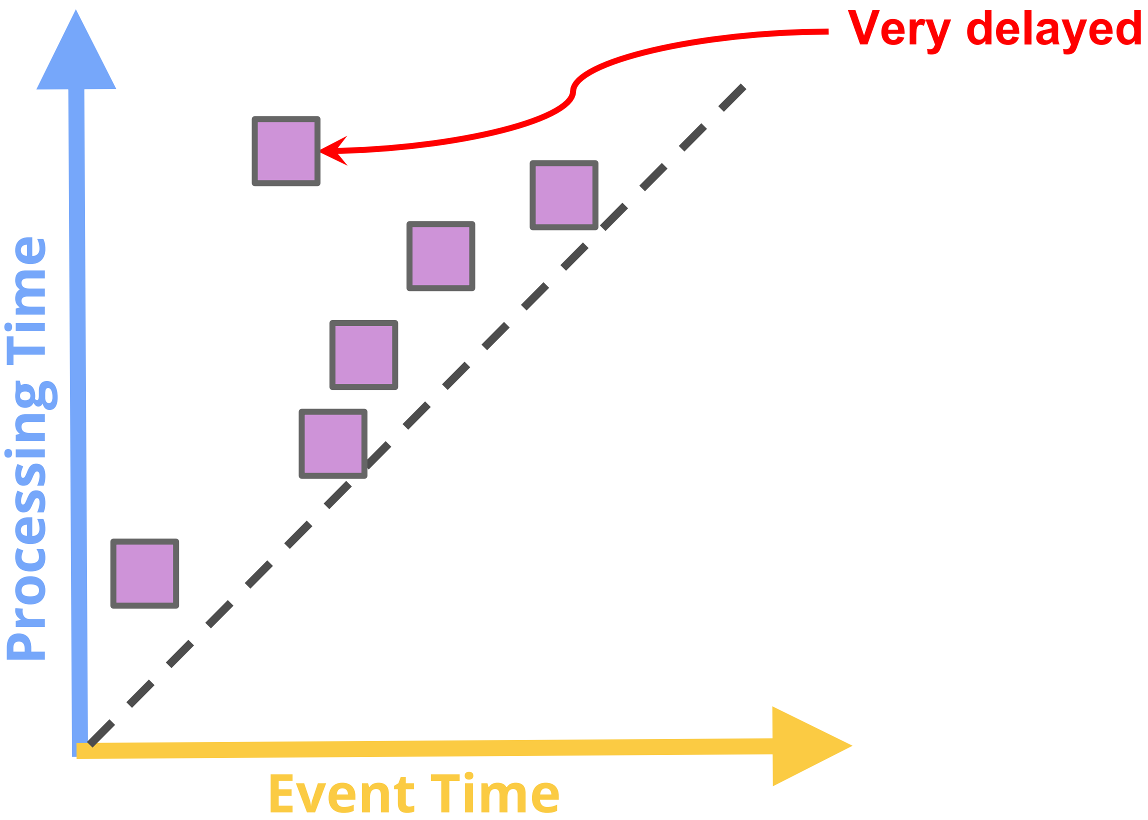
Processing Time vs Event Time



Processing Time vs Event Time

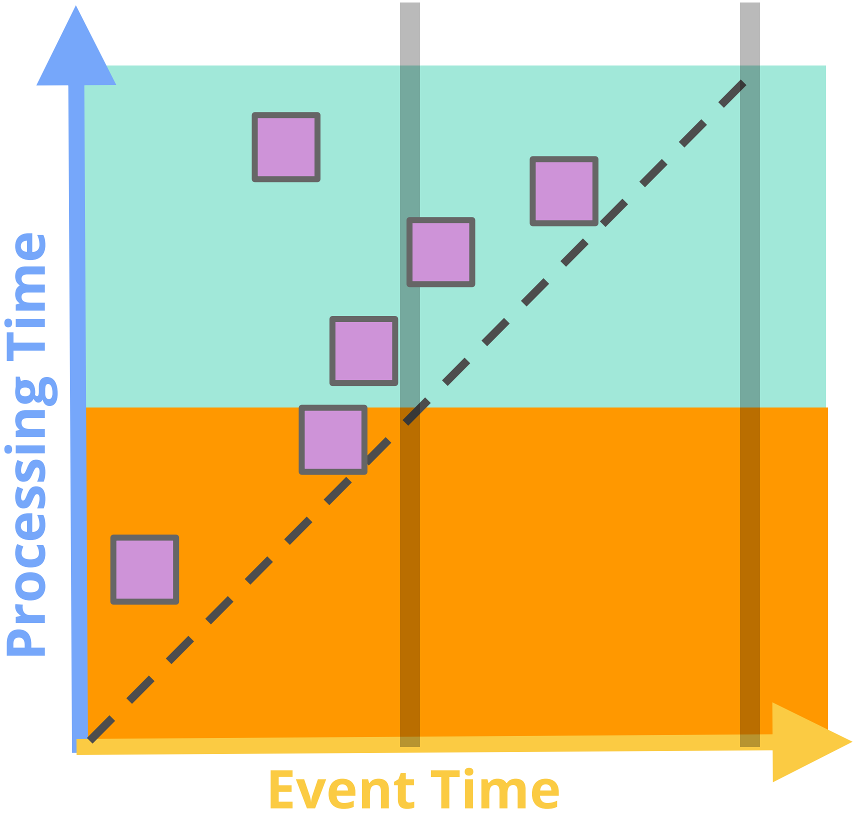


Processing Time vs Event Time

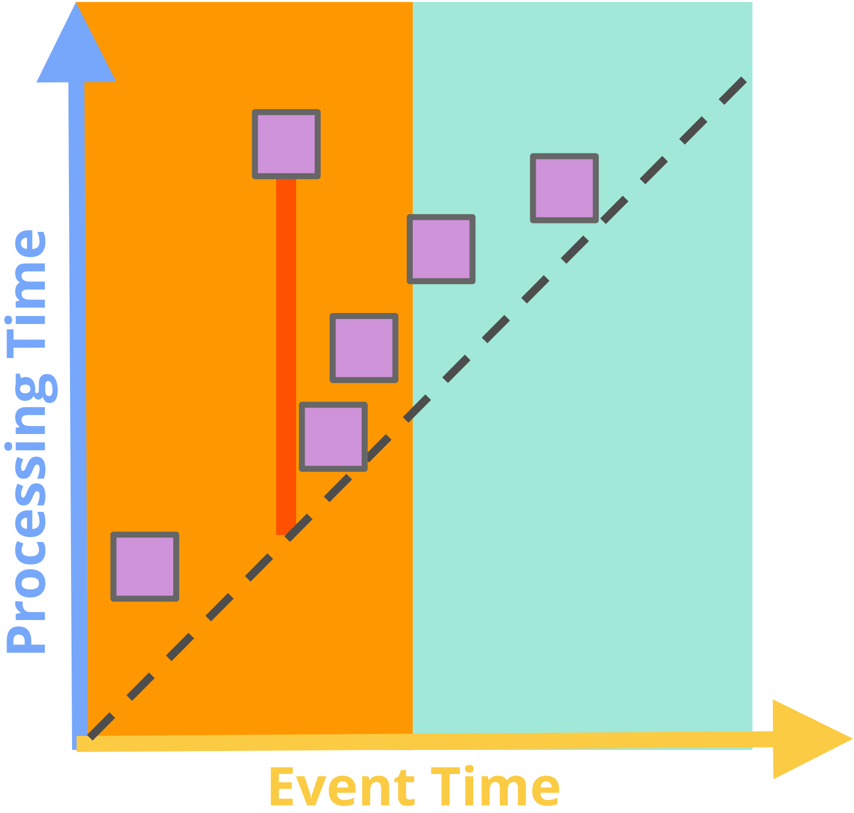


Processing Time windows

(probably are not what you want)



Event Time Windows



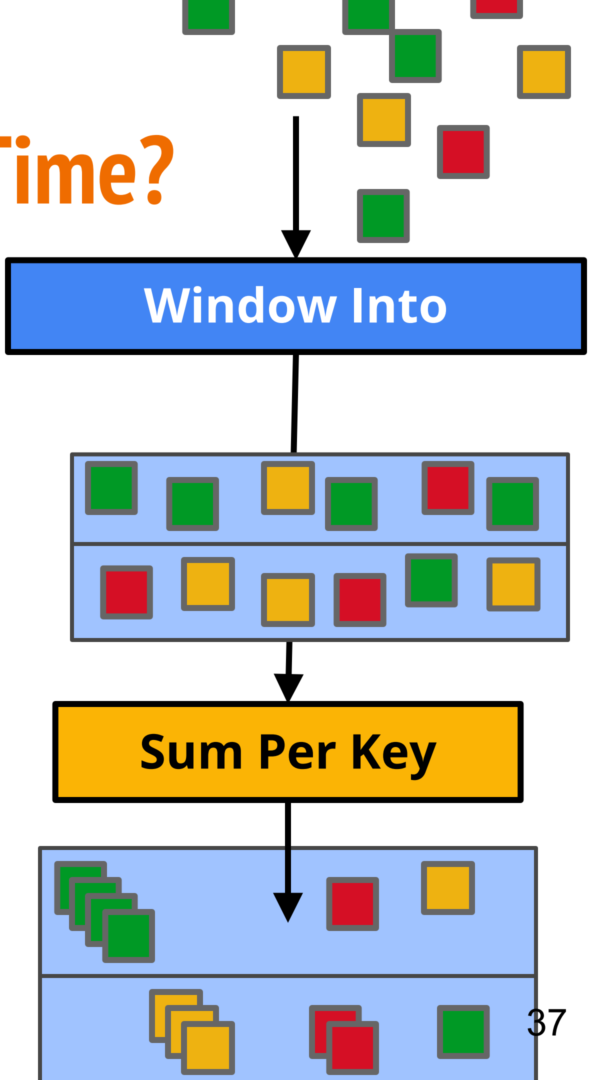
The Beam Model: Where in Event Time?

Python

```
input | WindowInto(FixedWindows(3600)
      | Sum.PerKey()
      | Write(BigQuerySink(...))
```

Java

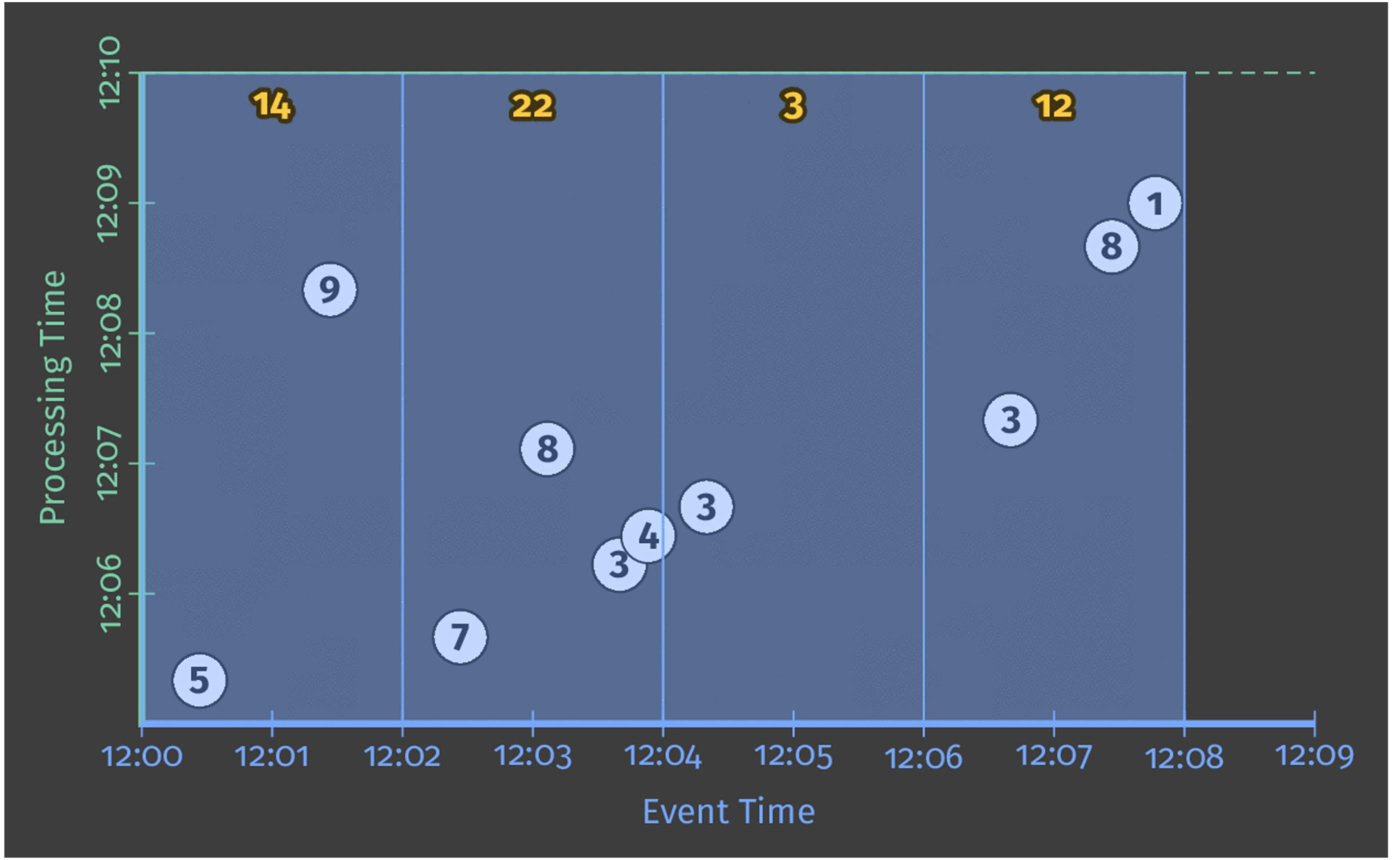
```
input.apply(
  Window.into(
    FixedWindows.of(
      Duration.standardHours(1)))
   .apply(Sum.integersPerKey())
   .apply(BigQueryIO.Write.to(...))
```



Where: Fixed 2-minute Windows

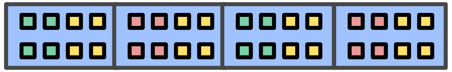
```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Minutes(2)))
    .apply(Sum.integersPerKey());
```

Where: Fixed 2-minute Windows

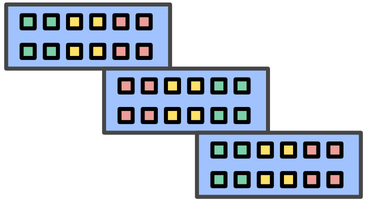


The Beam Model: Where in Event Time?

1. Assign each timestamped event to one or more windows

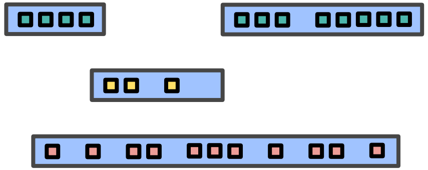


Fixed Windows
(also called Tumbling)



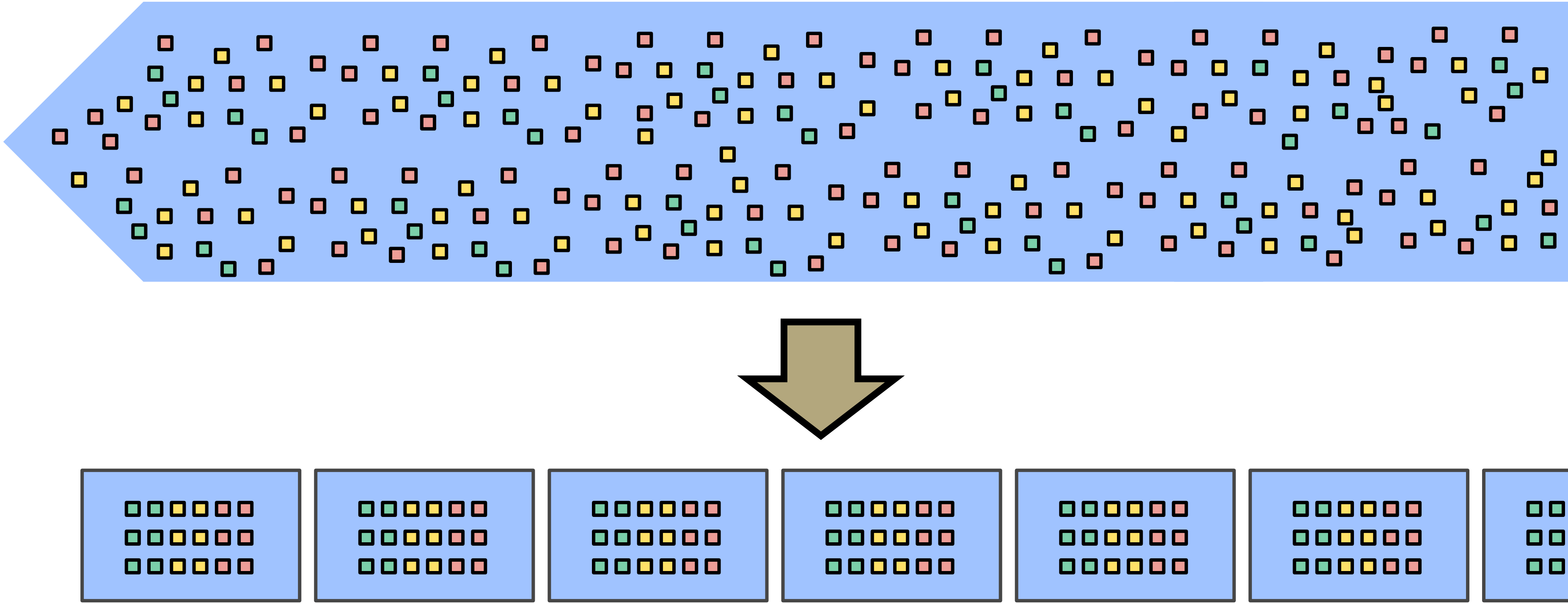
Sliding Windows

1. Merge those windows according to custom logic



User Sessions

So that's what and where...



Beam Model: Asking the Right Questions

What are you computing?

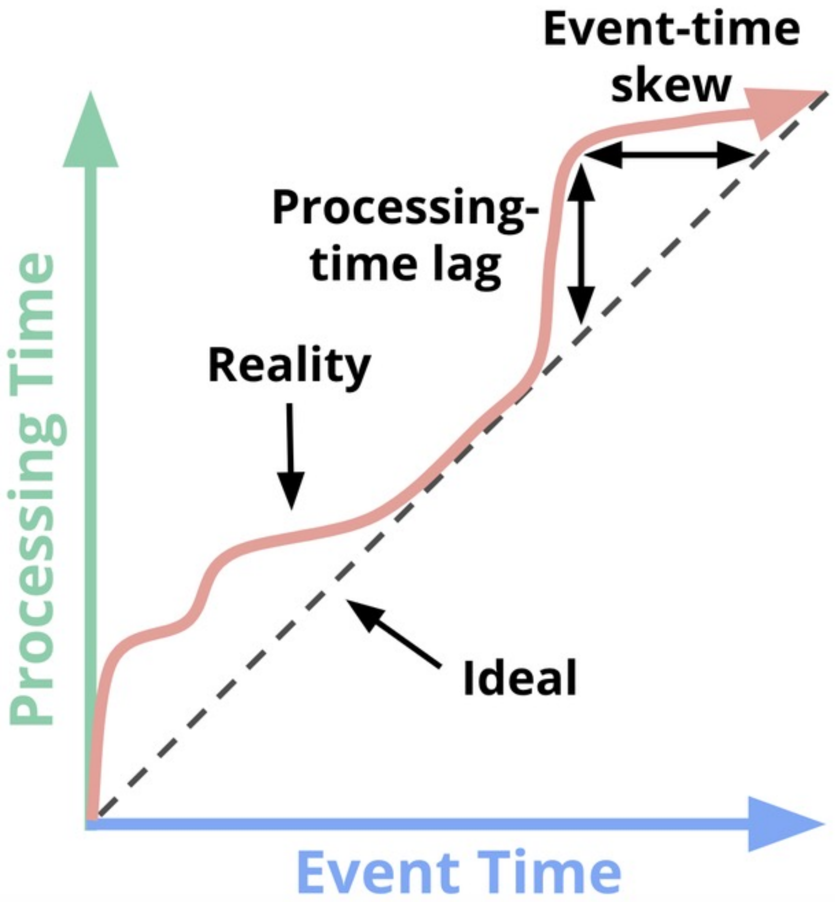
Triggers

Where in event time?

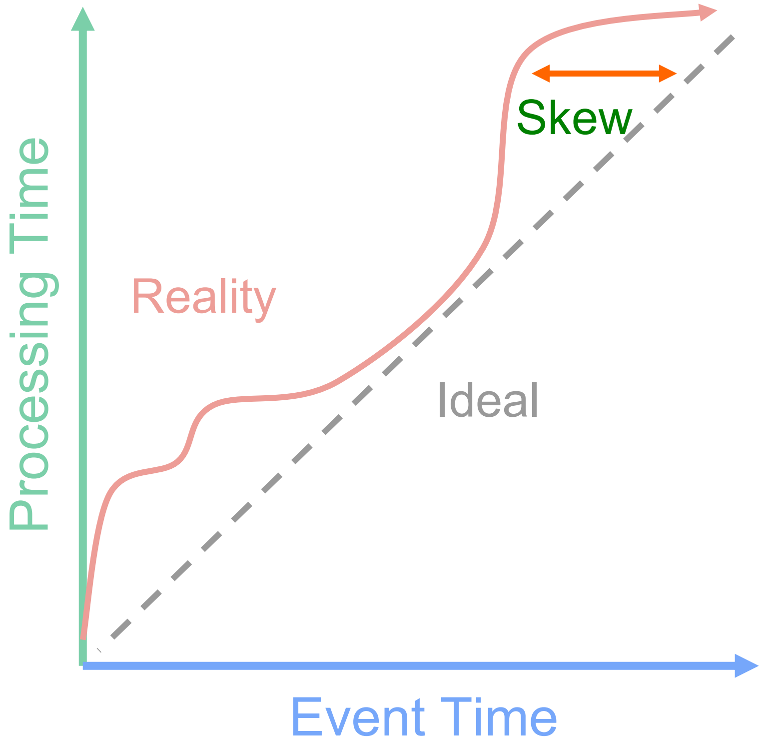
When in processing time are results produced?

How do refinements relate?

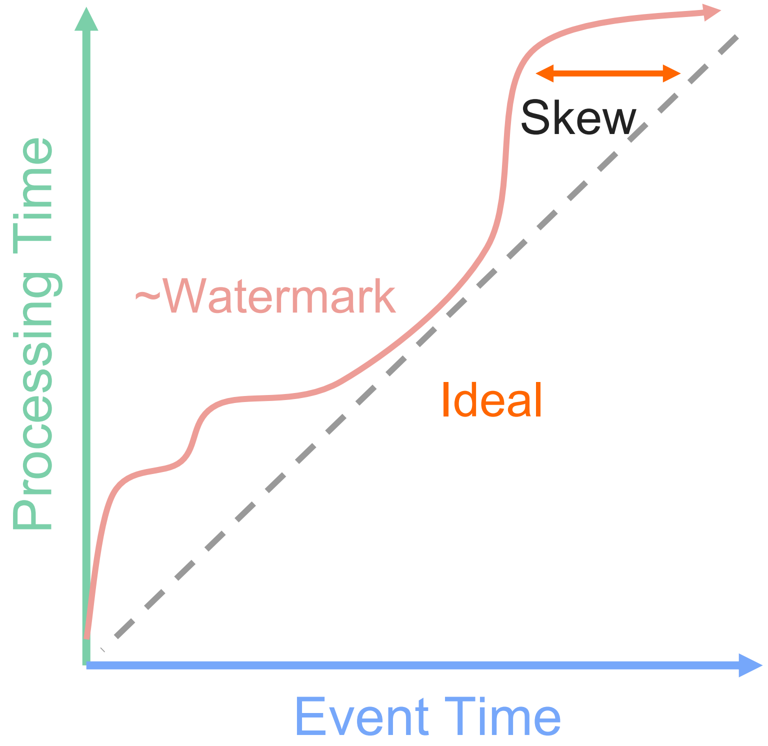
Formalizing Event-Time Skew



Formalizing Event-Time Skew



Formalizing Event-Time Skew



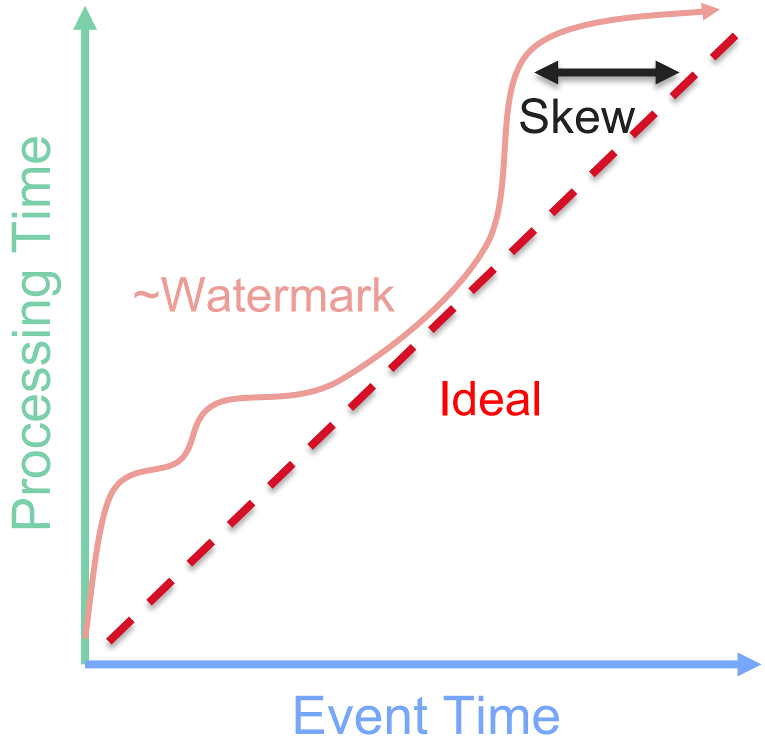
Watermarks describe event time progress.

"No (event-time) timestamp earlier than the watermark will be seen

Often heuristic-based.

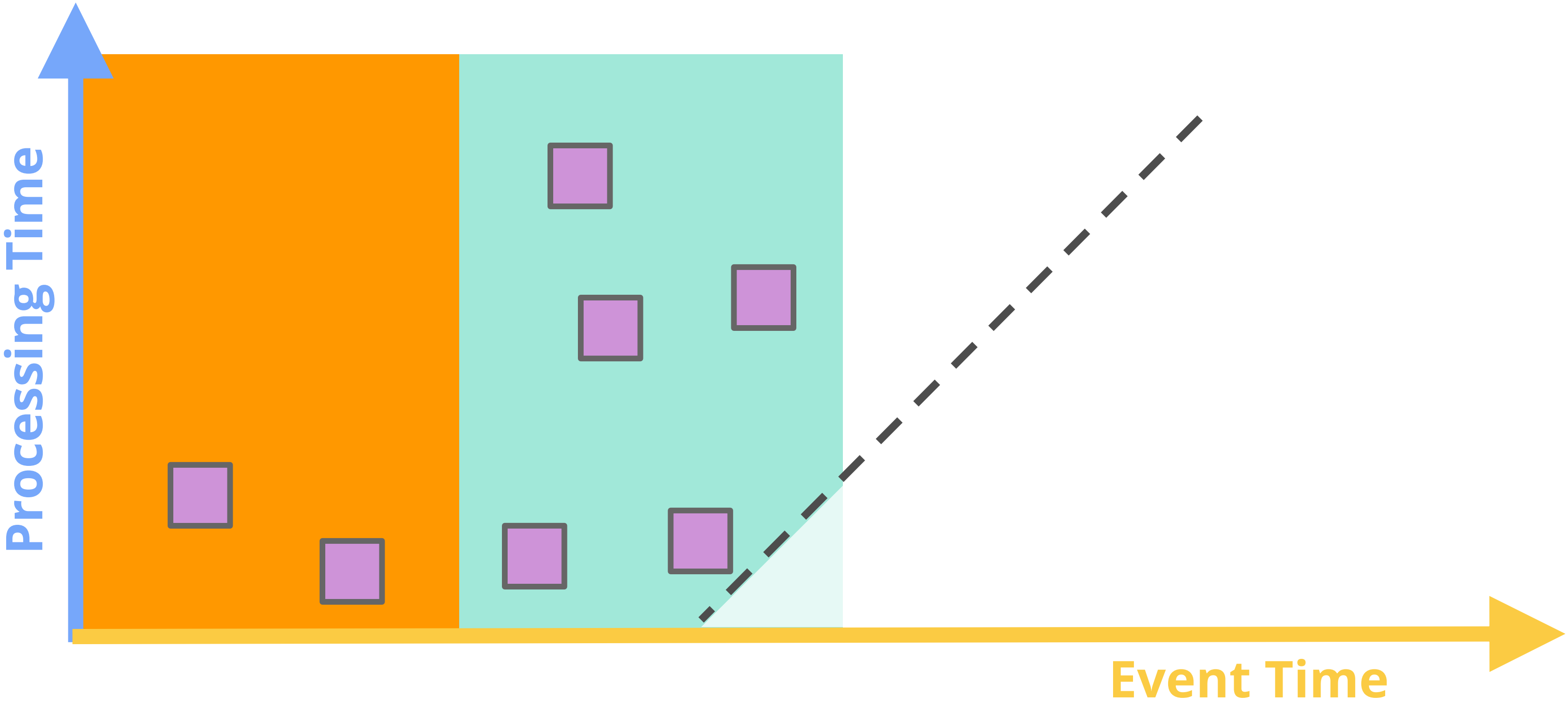
Too Slow? Results are *delayed*.
Too Fast? Some data is *late*.

When in processing time?

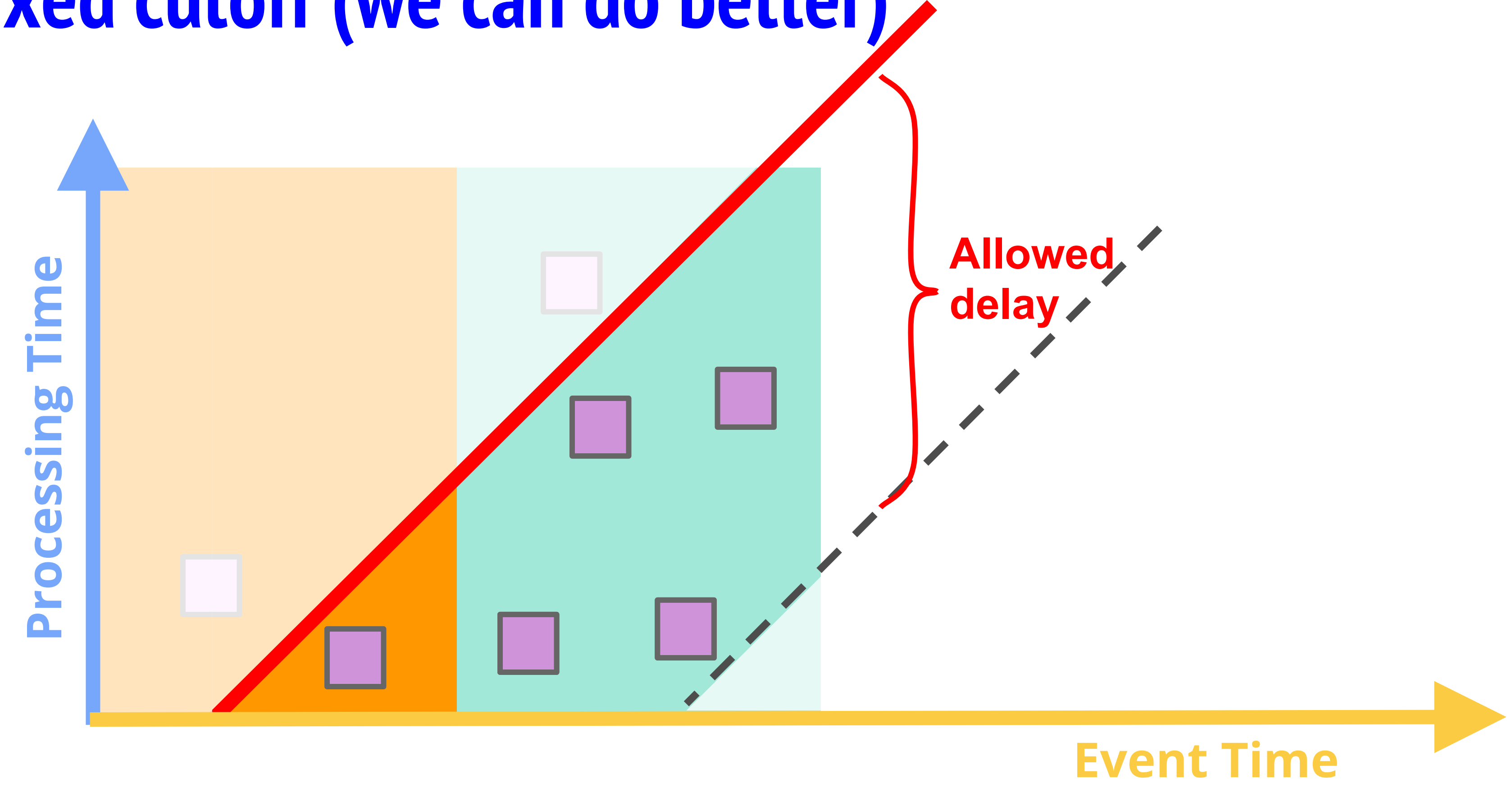


- Triggers control when results are emitted.
- Triggers are often relative to the watermark.

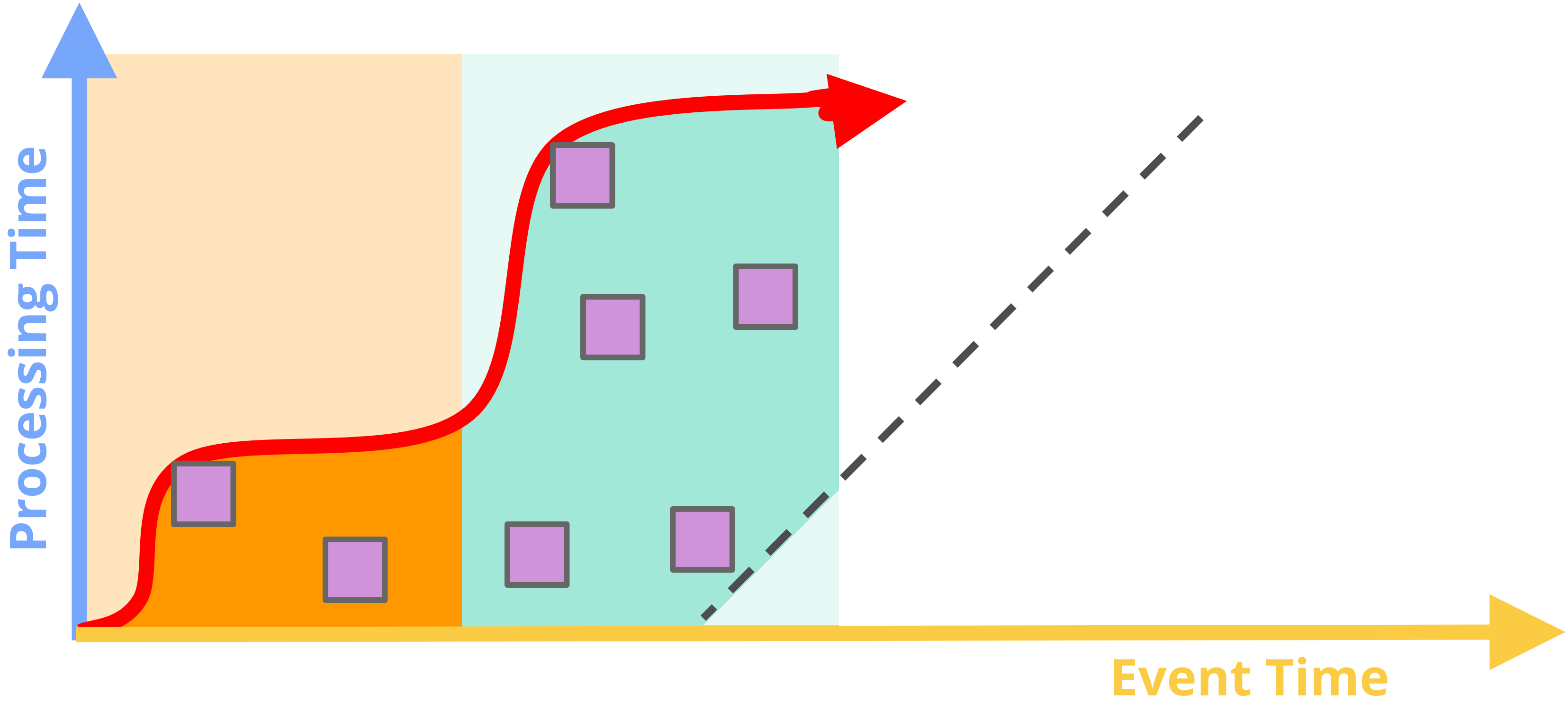
Event time windows



Fixed cutoff (we can do better)



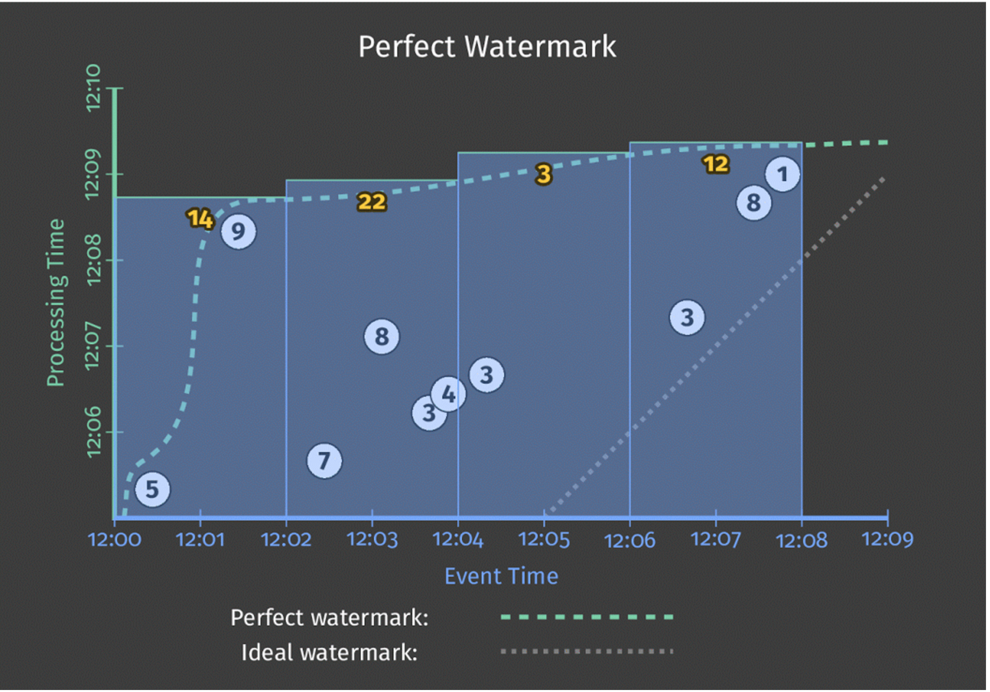
Perfect watermark



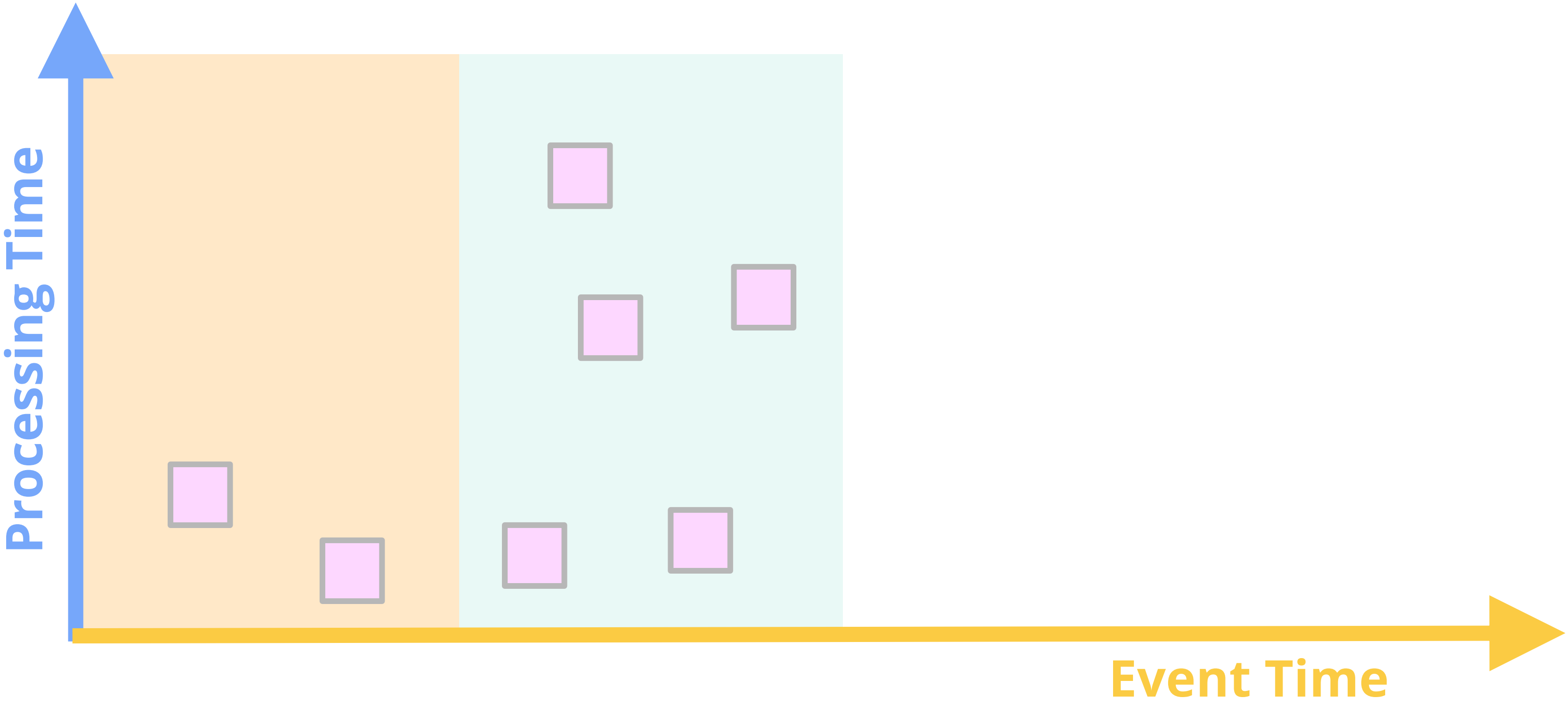
When:

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

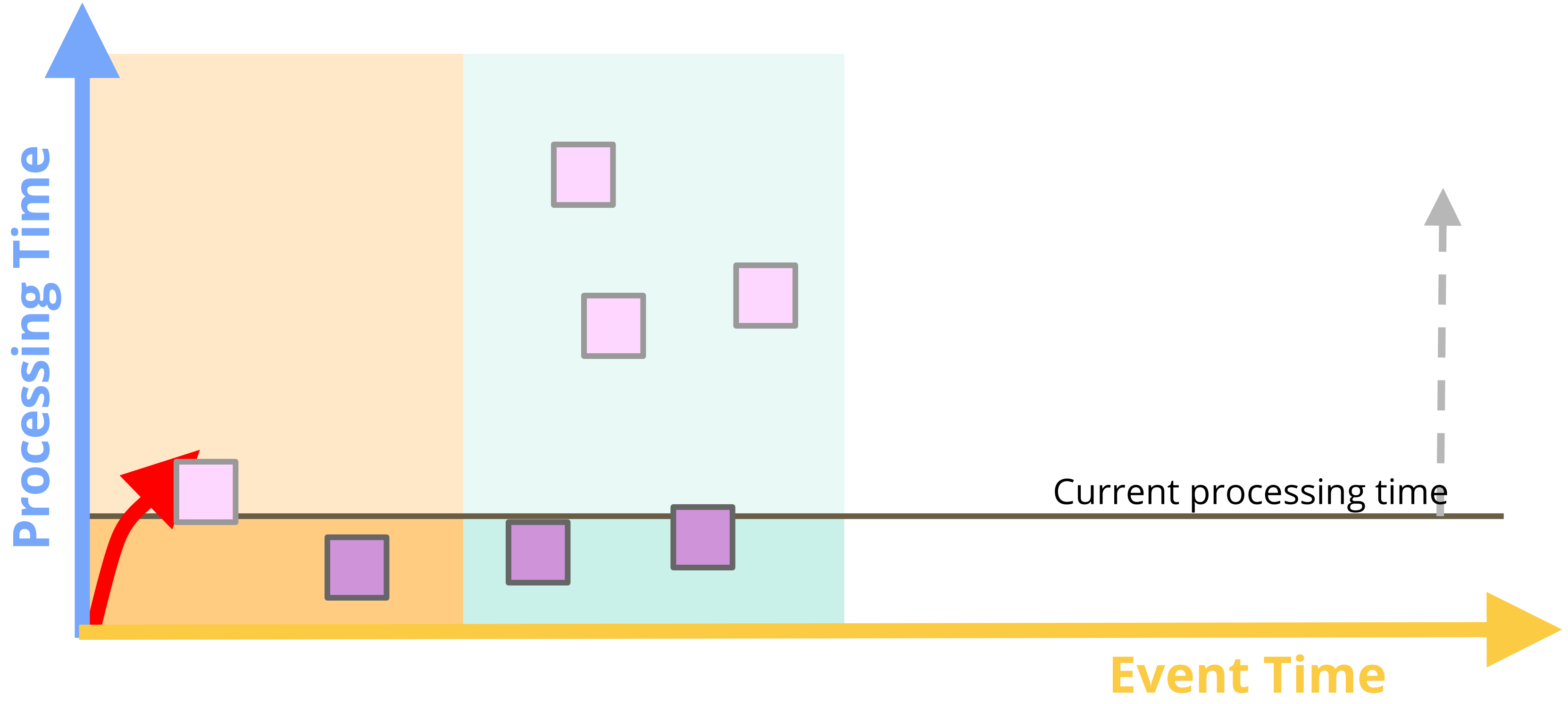
When: Triggering at the Watermark



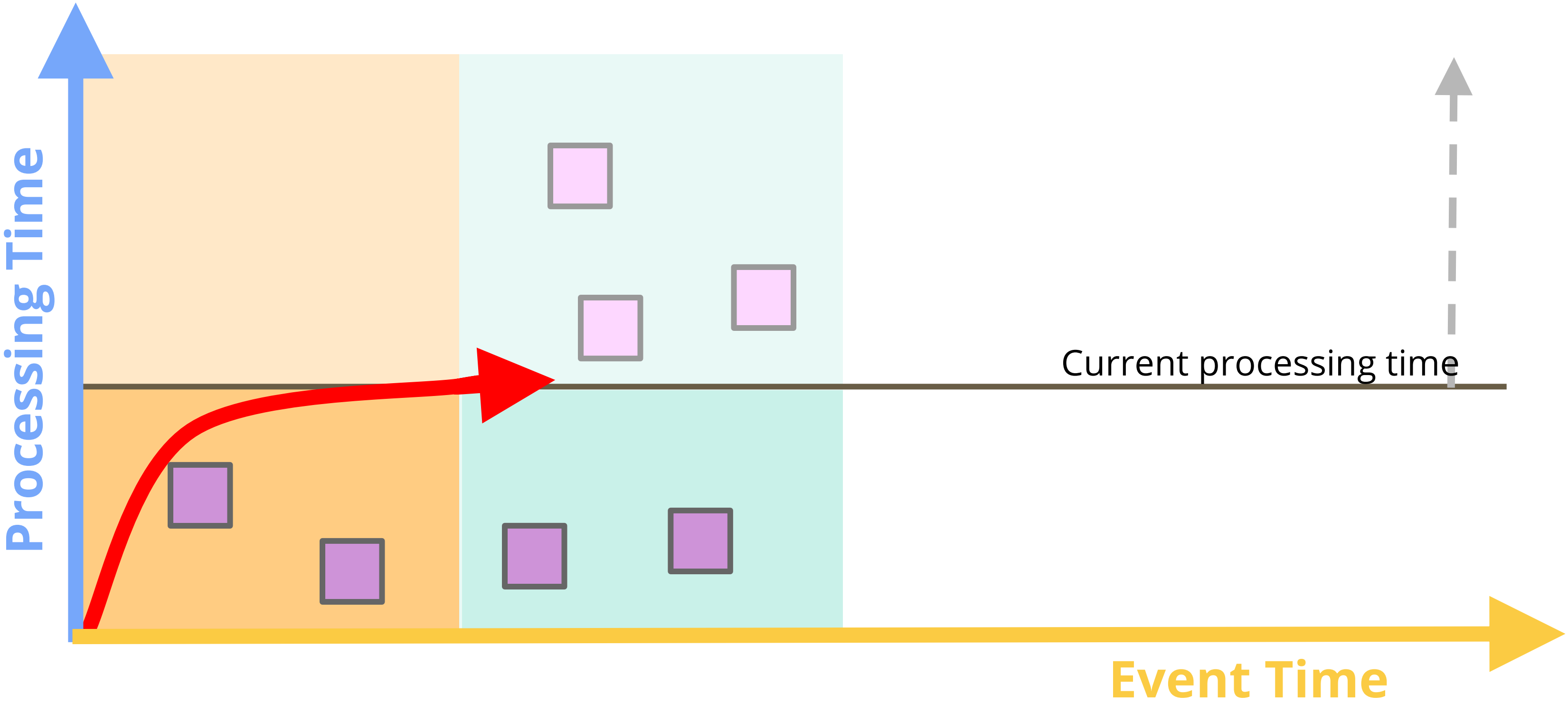
Heuristic Watermark



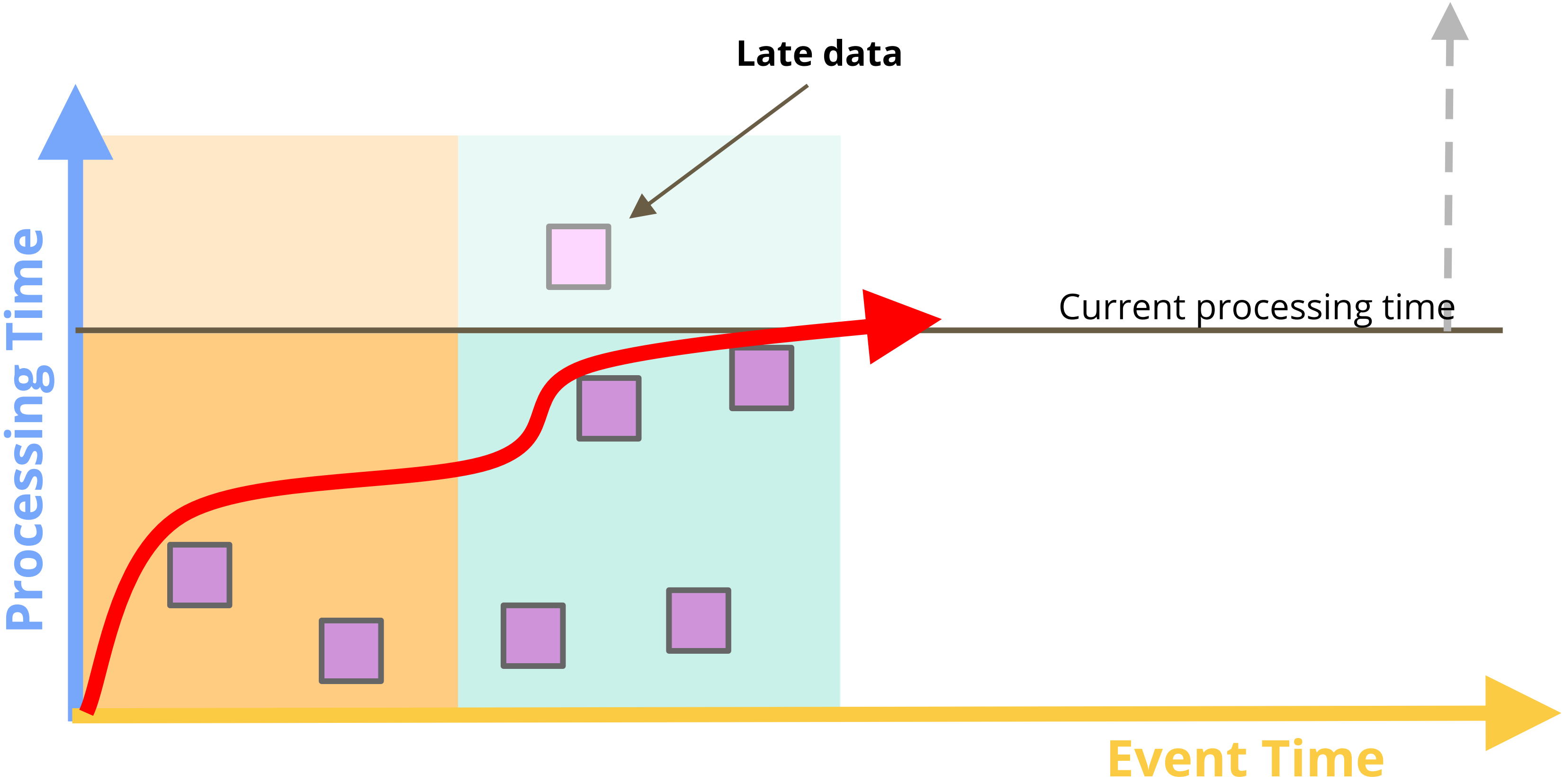
Heuristic Watermark



Heuristic Watermark

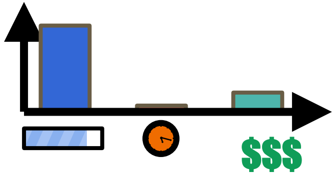


Heuristic Watermark

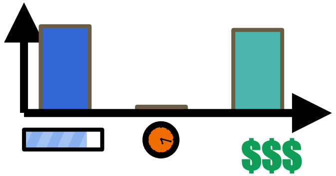


Watermarks measure completeness

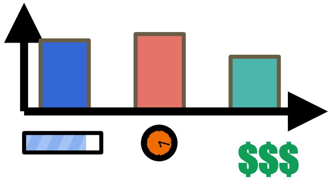
✓ Monthly billing



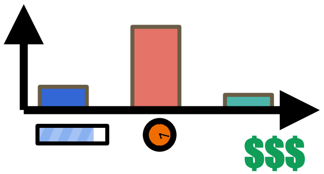
✓ Historical Analysis



? Running Total



? Abuse Detection



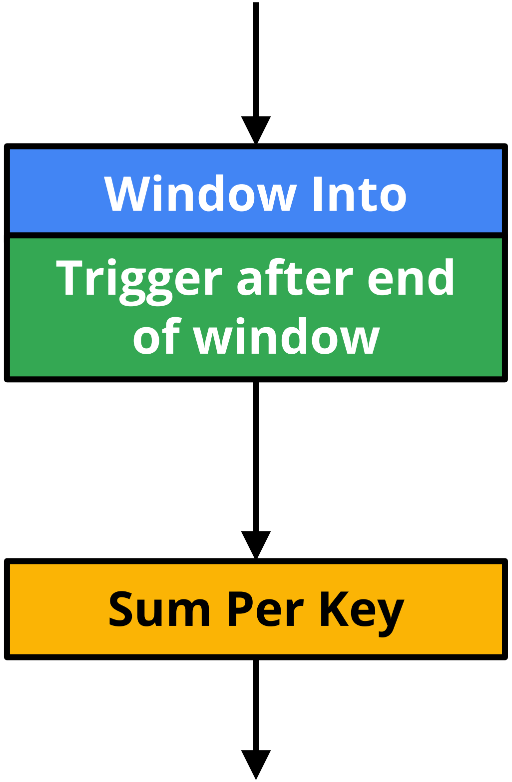
The Beam Model: When in Processing Time?

Python

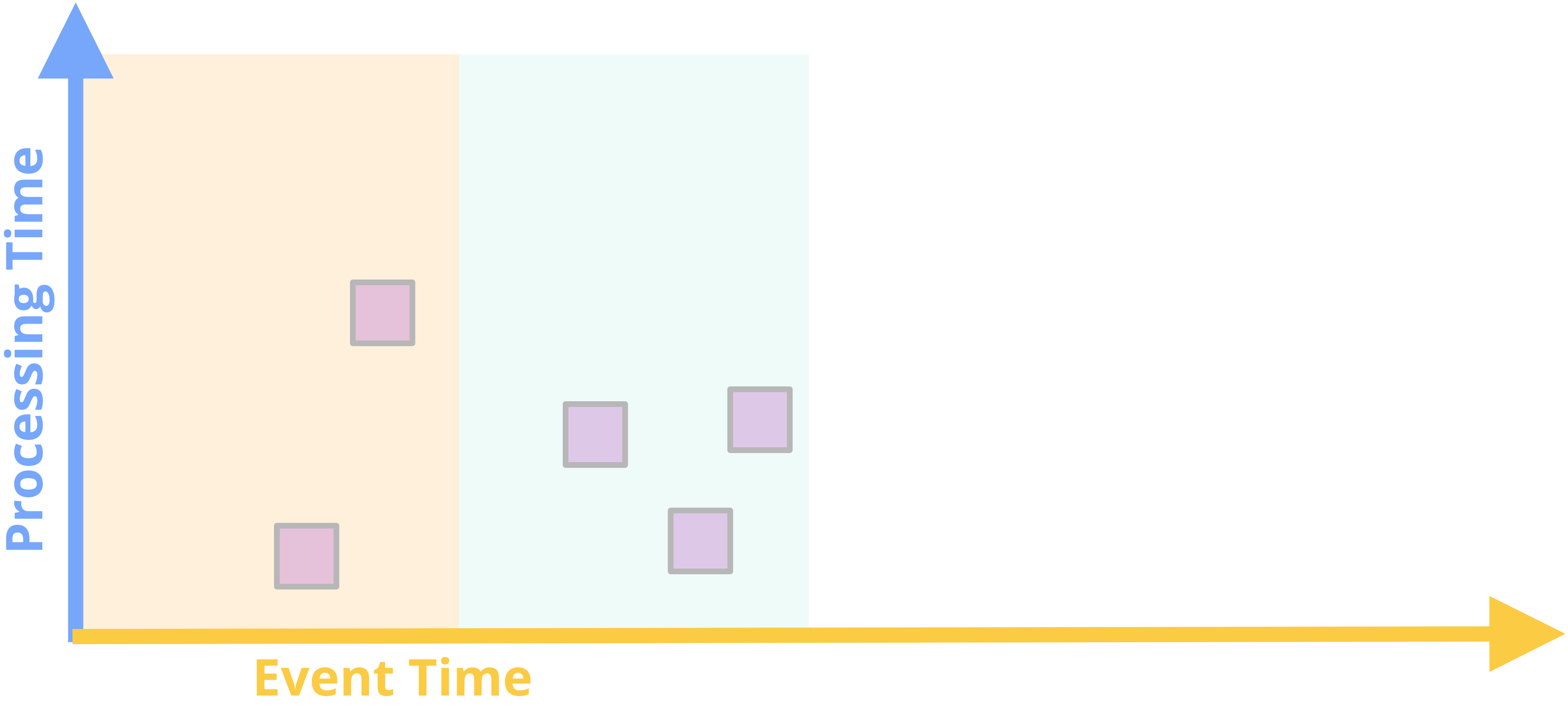
```
input | WindowInto(FixedWindows(3600),
                   trigger=AfterWatermark())
      | Sum.PerKey()
      | Write(BigQuerySink(...))
```

Java

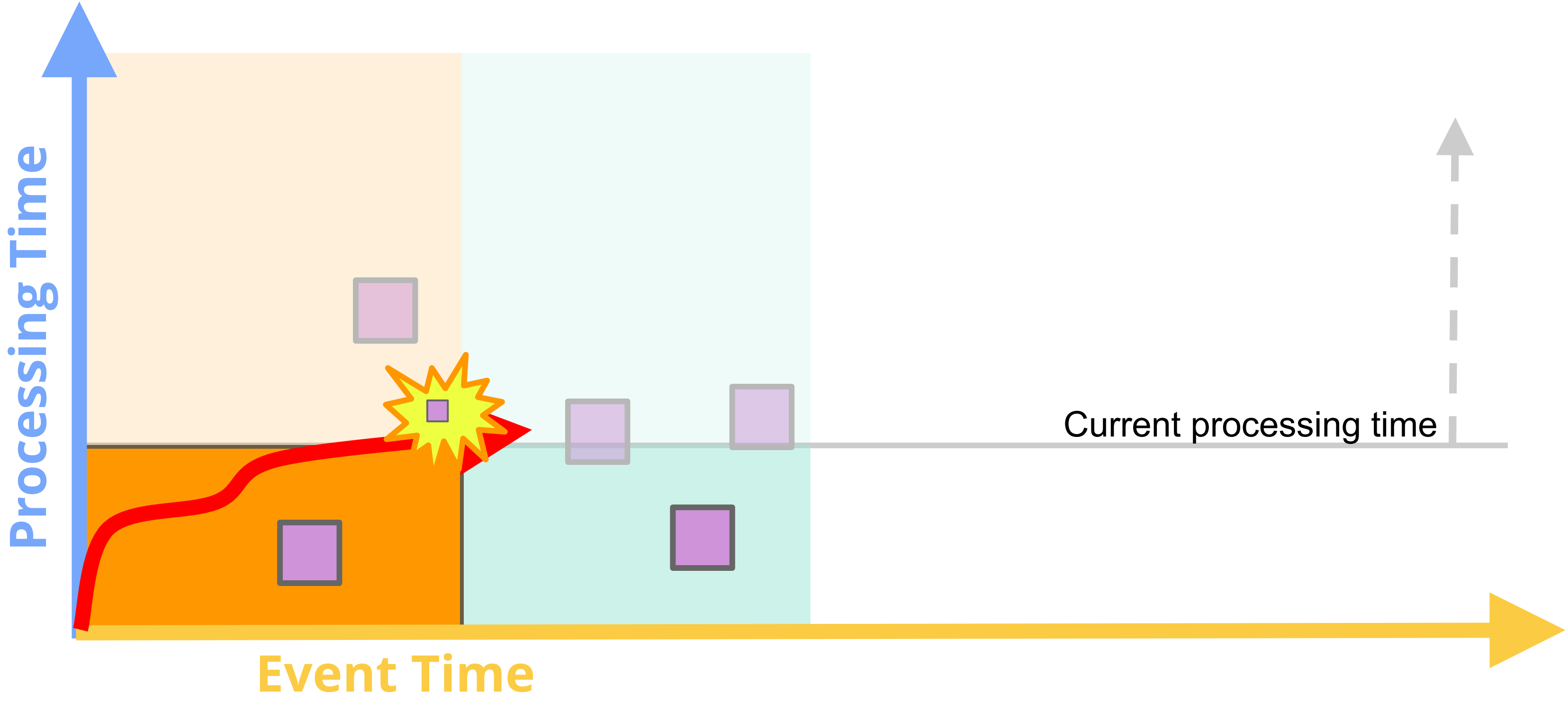
```
input
  .apply(Window.into(FixedWindows.of(...))
    .triggering(
      AfterWatermark.pastEndOfWindow()))
  .apply(Sum.integersPerKey())
  .apply(BigQueryIO.Write.to(...))
```



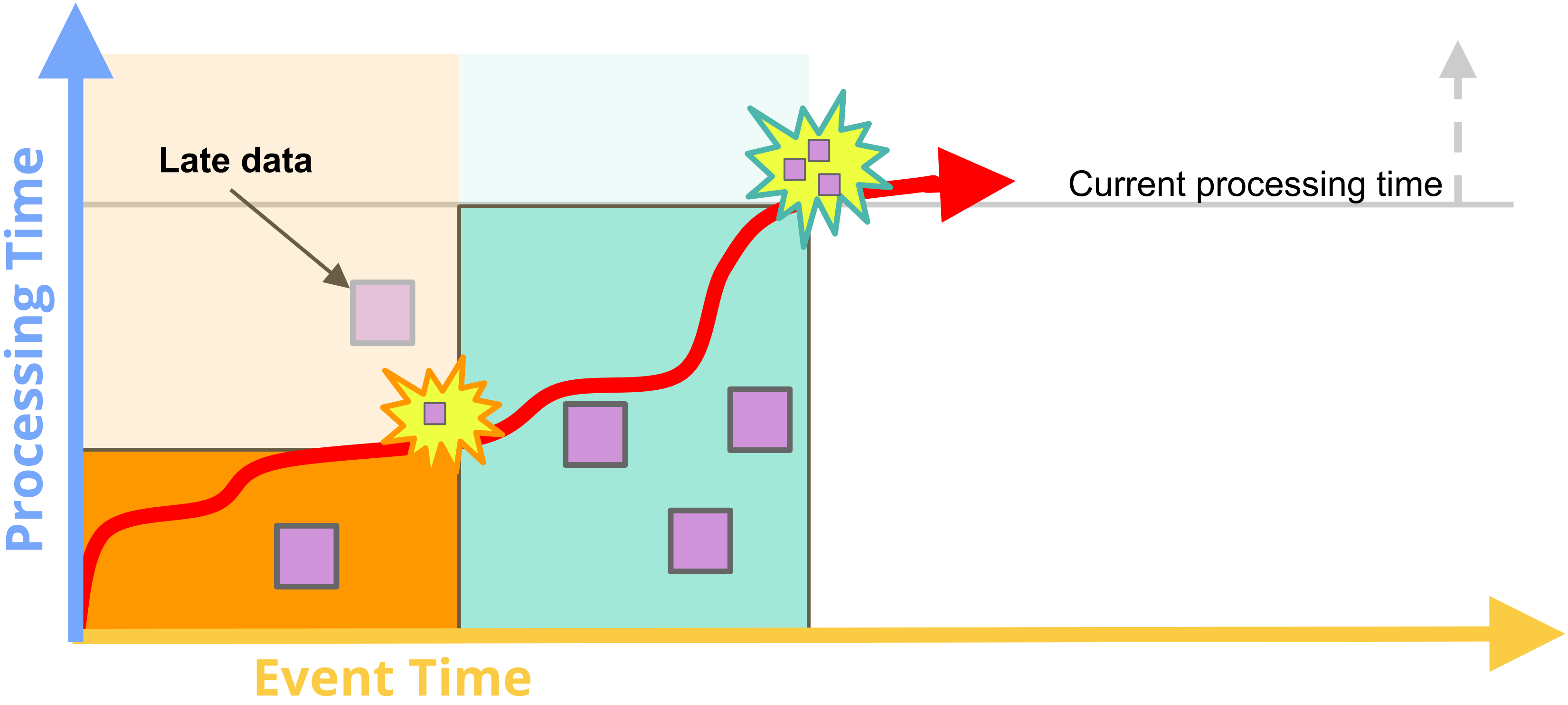
AfterWatermark.pastEndOfWindow()



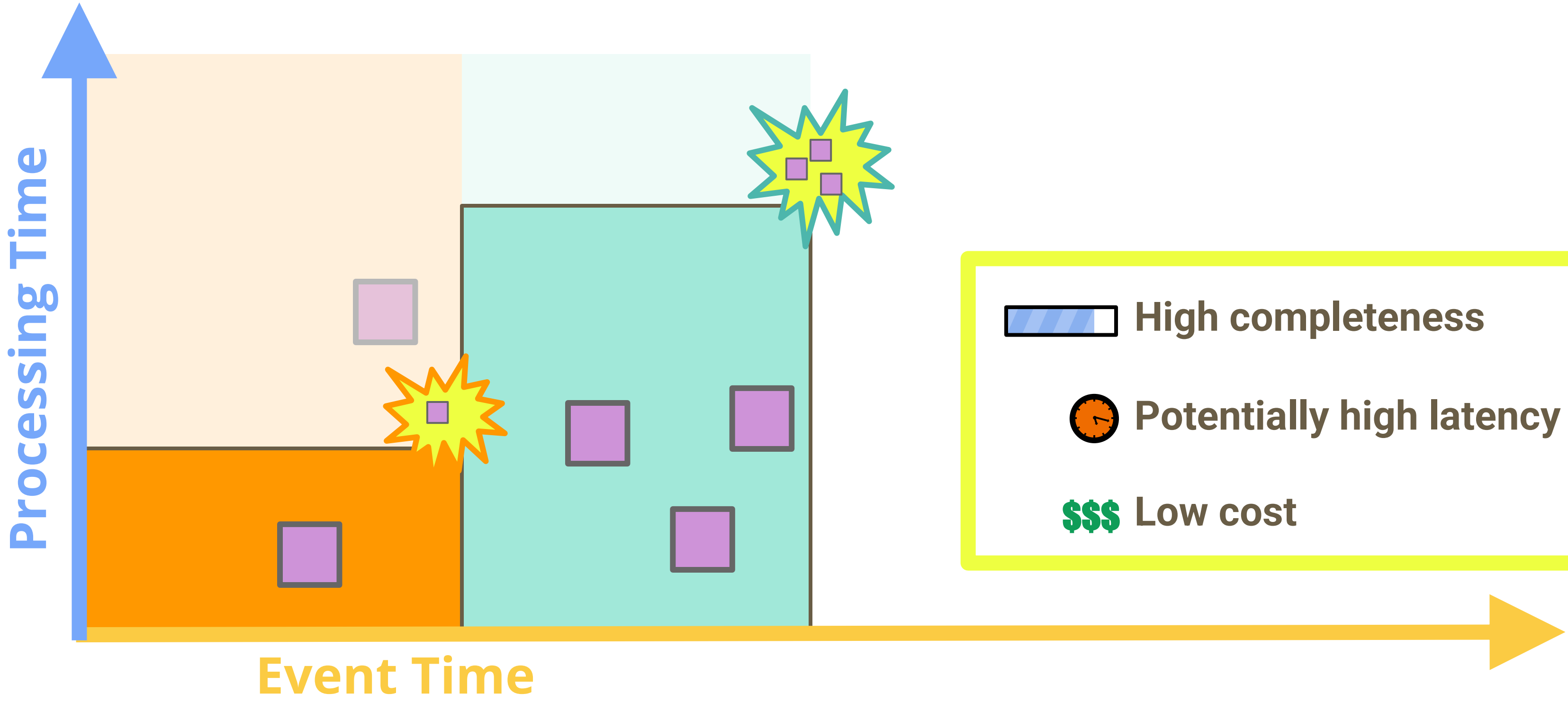
AfterWatermark.pastEndOfWindow()



AfterWatermark.pastEndOfWindow()

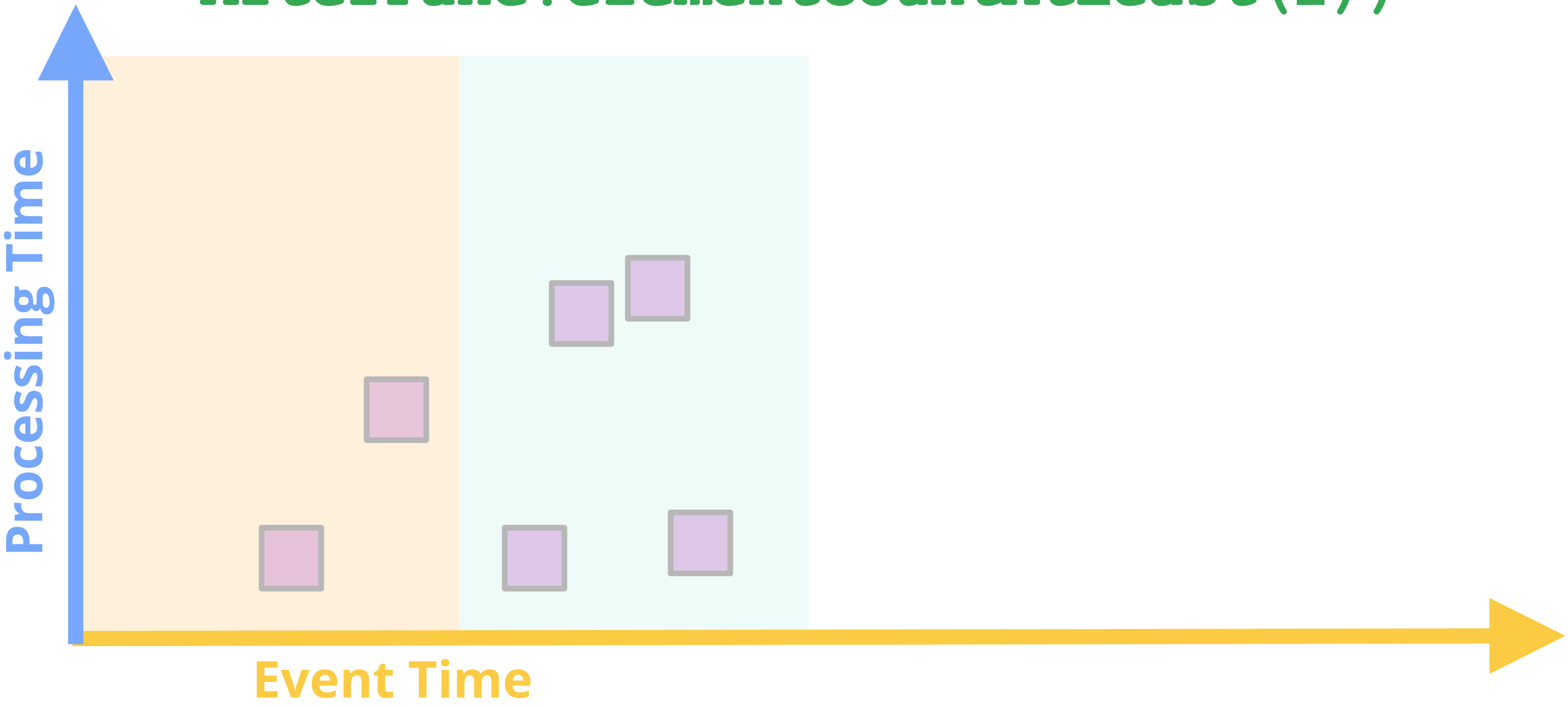


AfterWatermark.pastEndOfWindow()

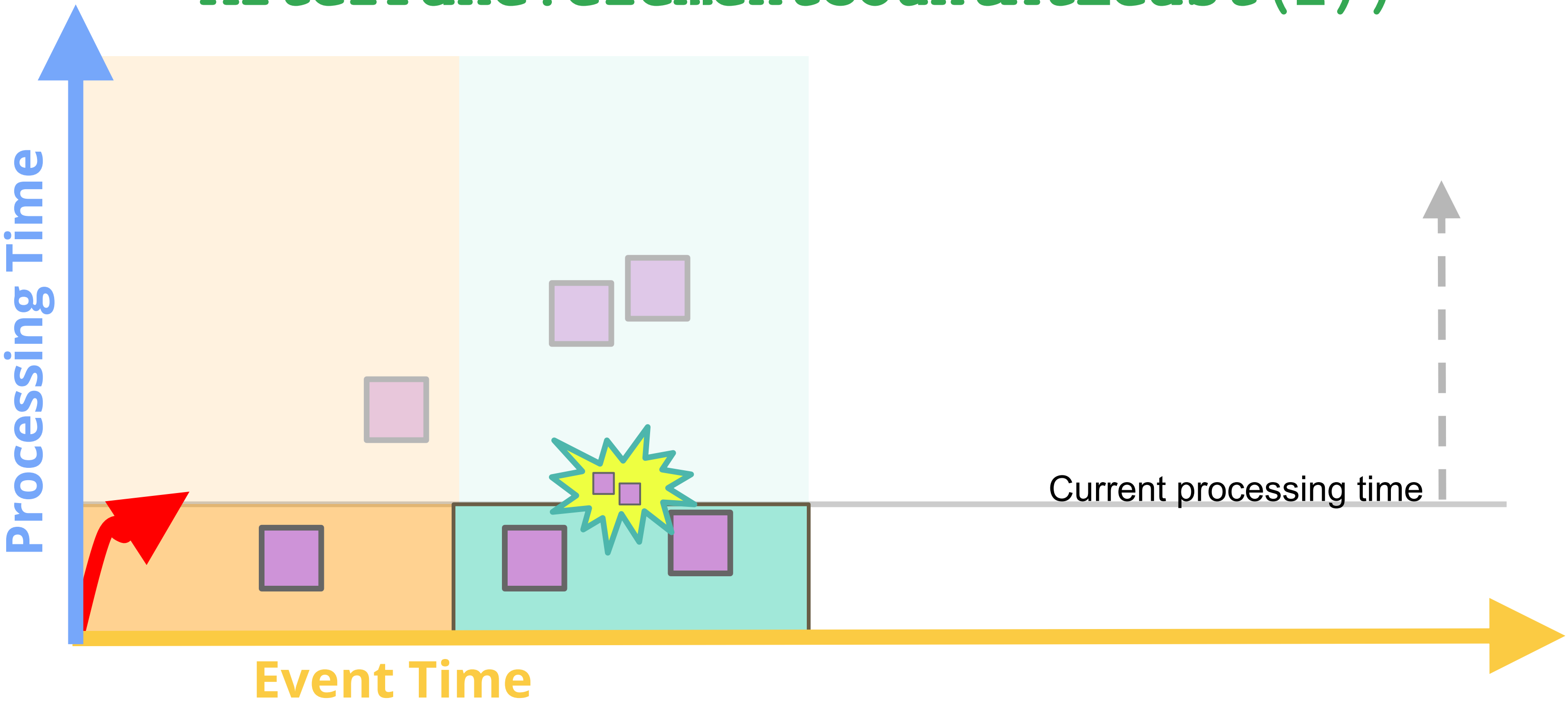


- High completeness
- Potentially high latency
- $$$ Low cost

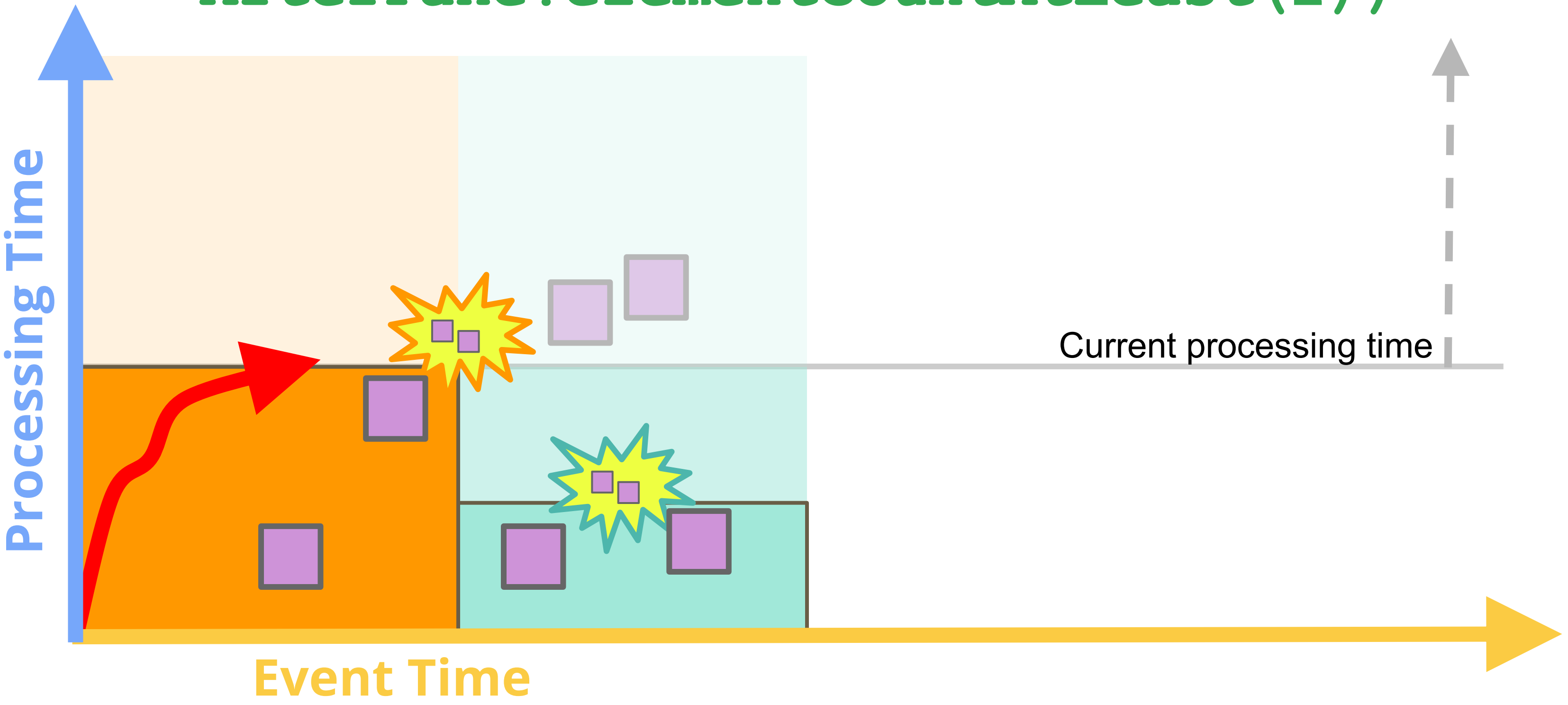
```
Repeatedly.forever(
    AfterPane.elementCountAtLeast(2))
```



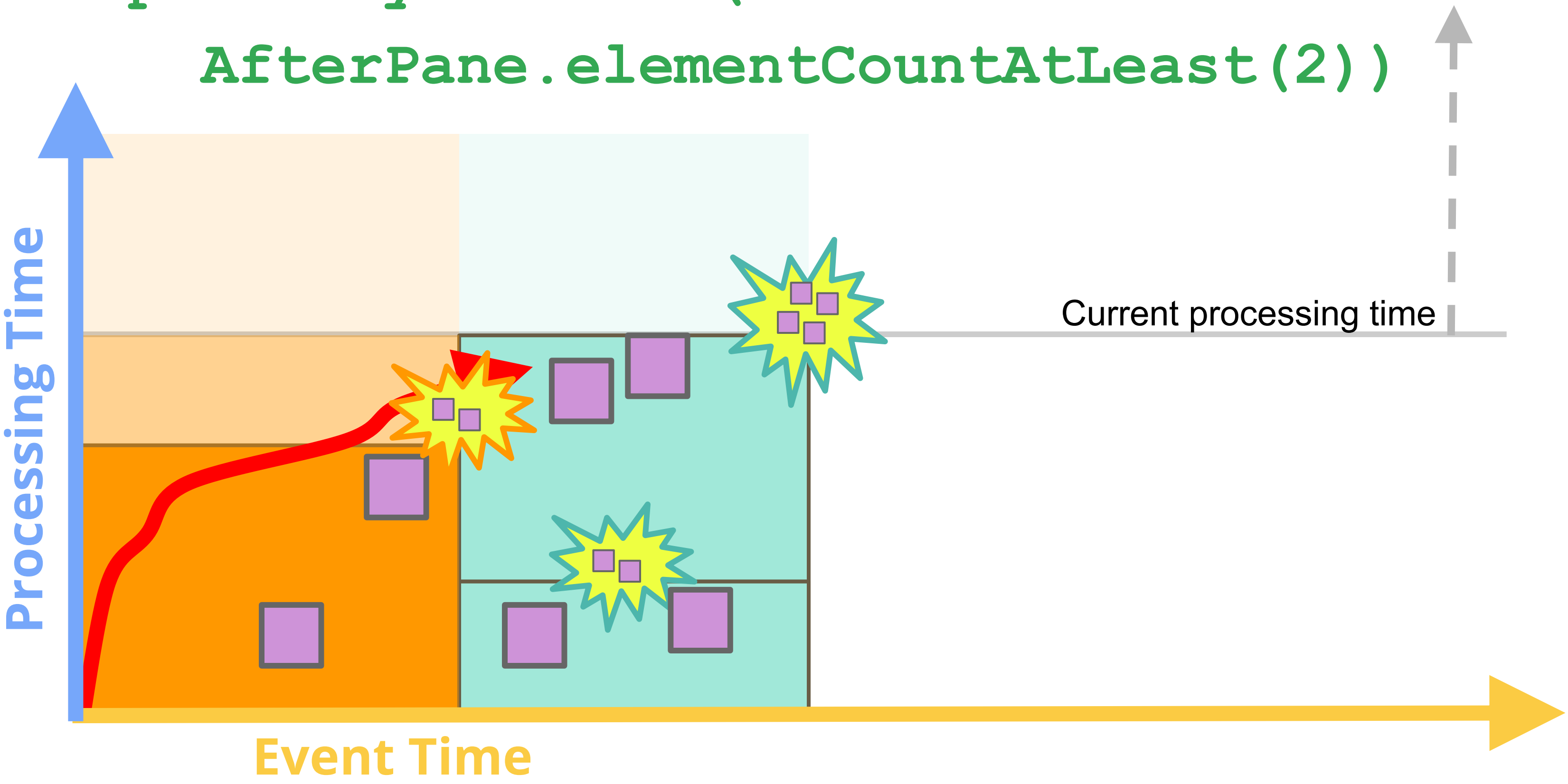
```
Repeatedly.forever(
    AfterPane.elementCountAtLeast(2))
```



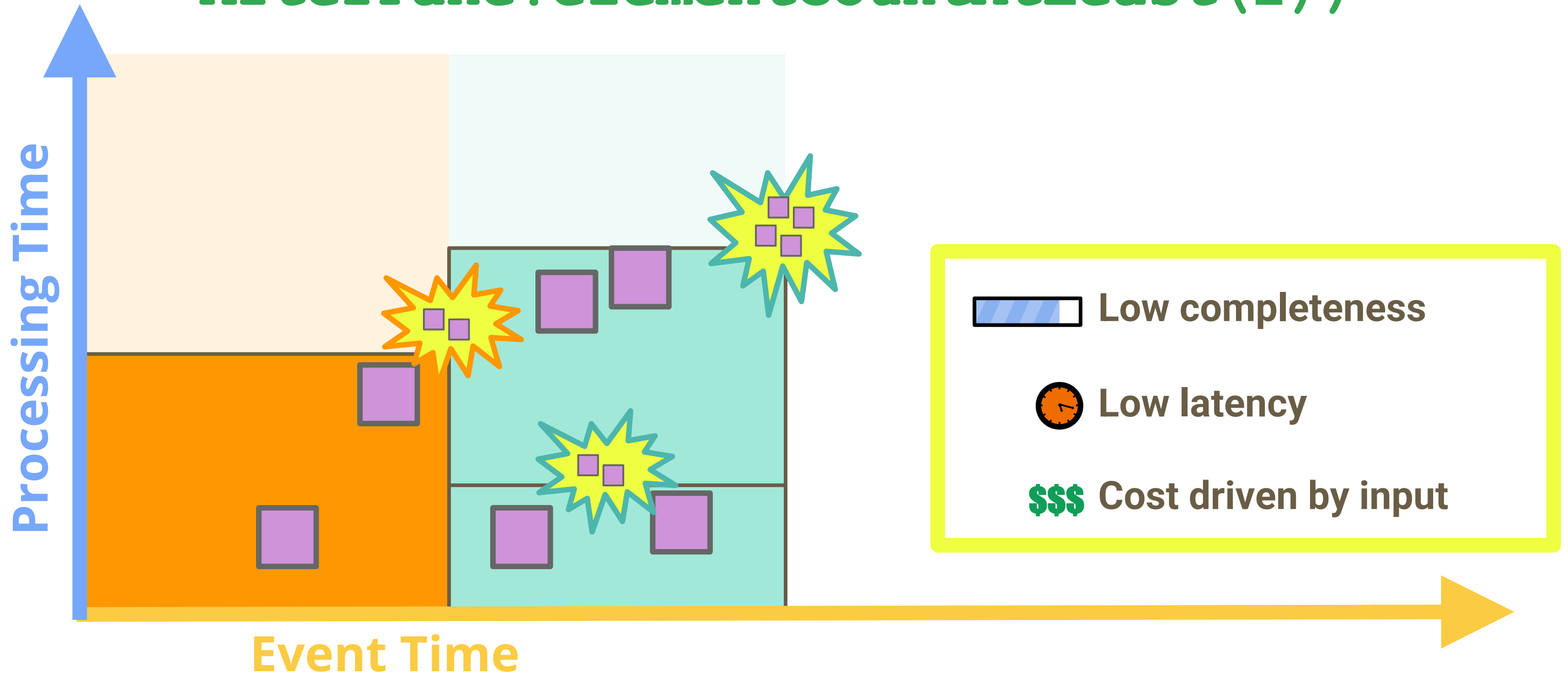
```
Repeatedly.forever(
    AfterPane.elementCountAtLeast(2))
```



```
Repeatedly.forever(
    AfterPane.elementCountAtLeast(2))
```



```
Repeatedly.forever(
    AfterPane.elementCountAtLeast(2))
```



Build a finely tuned trigger for your use case

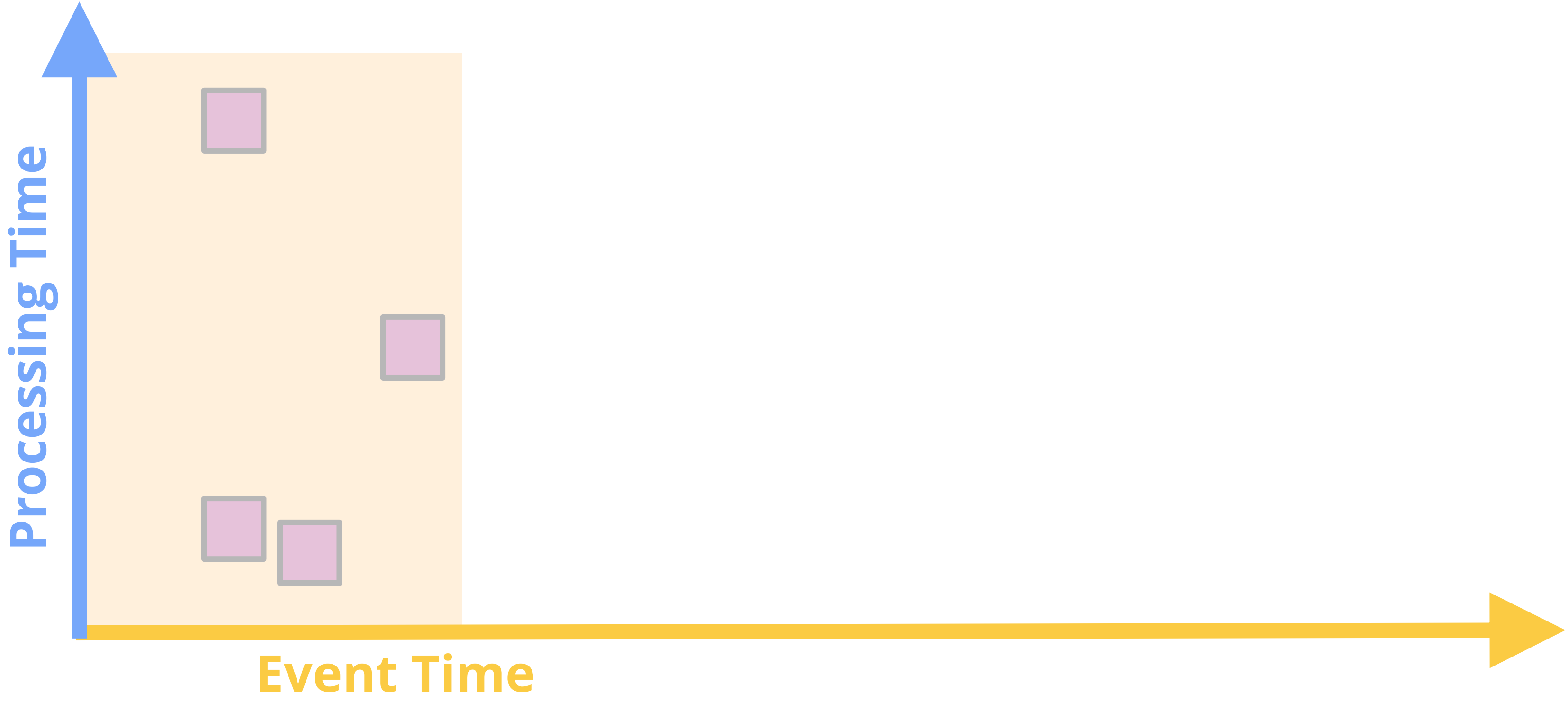
```
AfterWatermark.pastEndOfWindow()

    .withEarlyFirings(
        AfterProcessingTime
            .pastFirstElementInPane()
            .plusDuration(Duration.standardMinutes(1)))

    .withLateFirings(AfterPane.elementCountAtLeast(1))
```

Bill at end of month

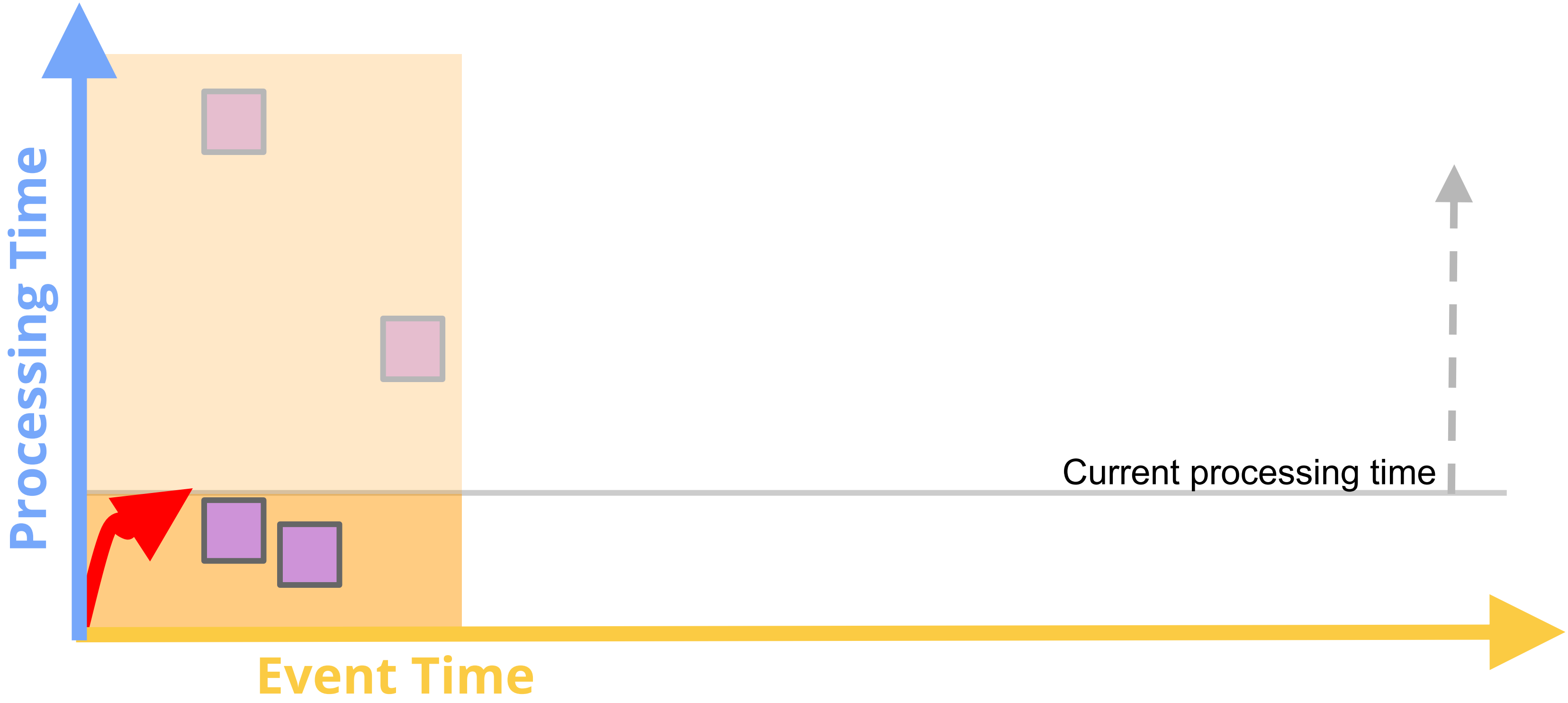
Near real-time estimates

Immediate corrections

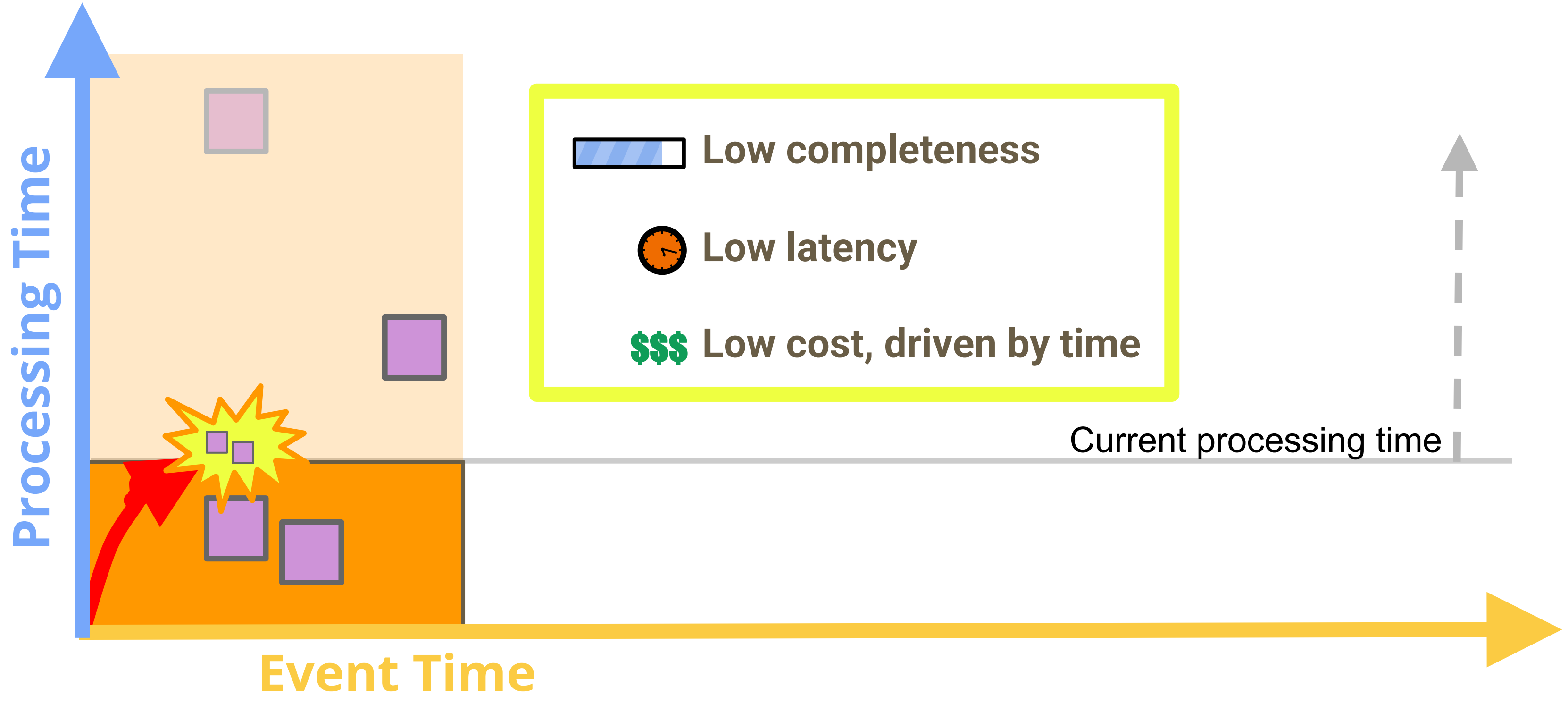
```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



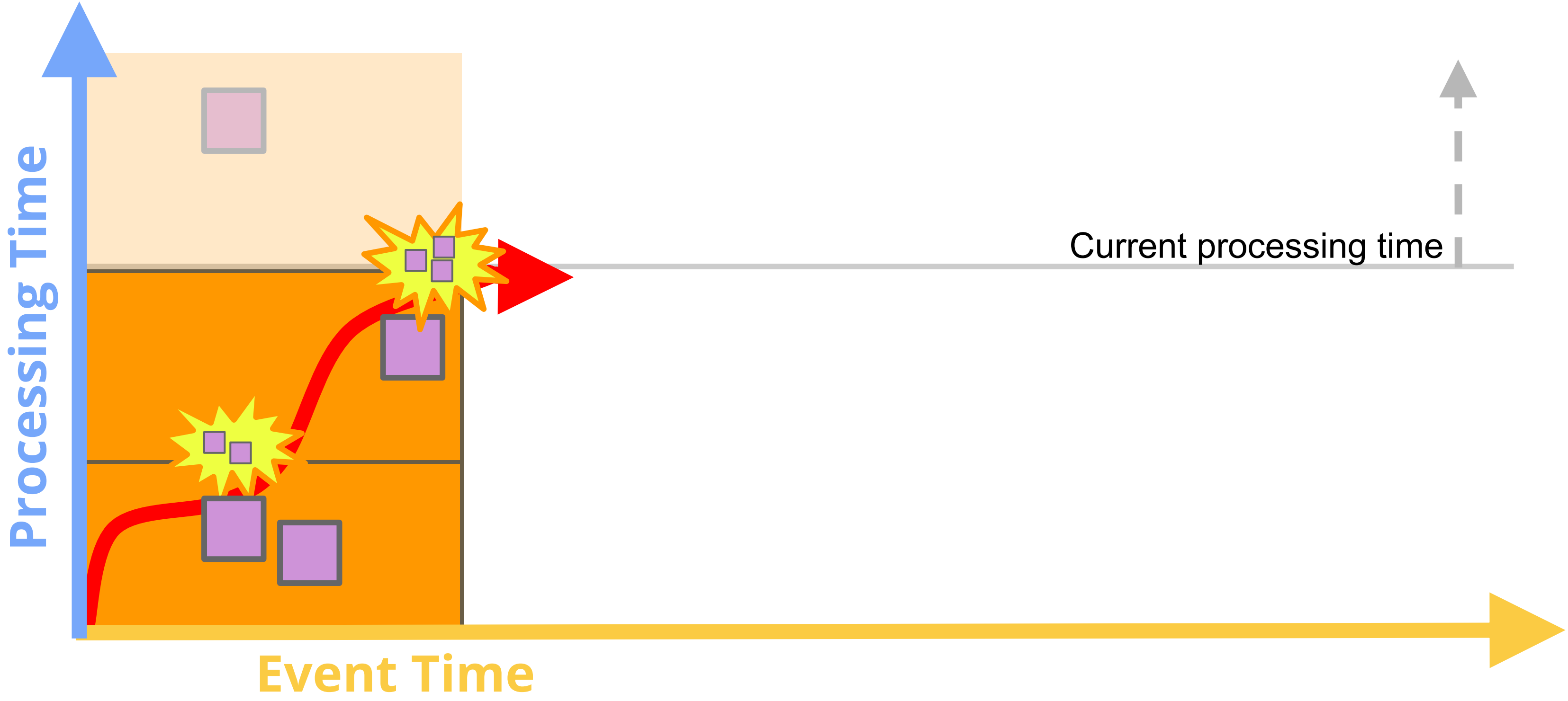
```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



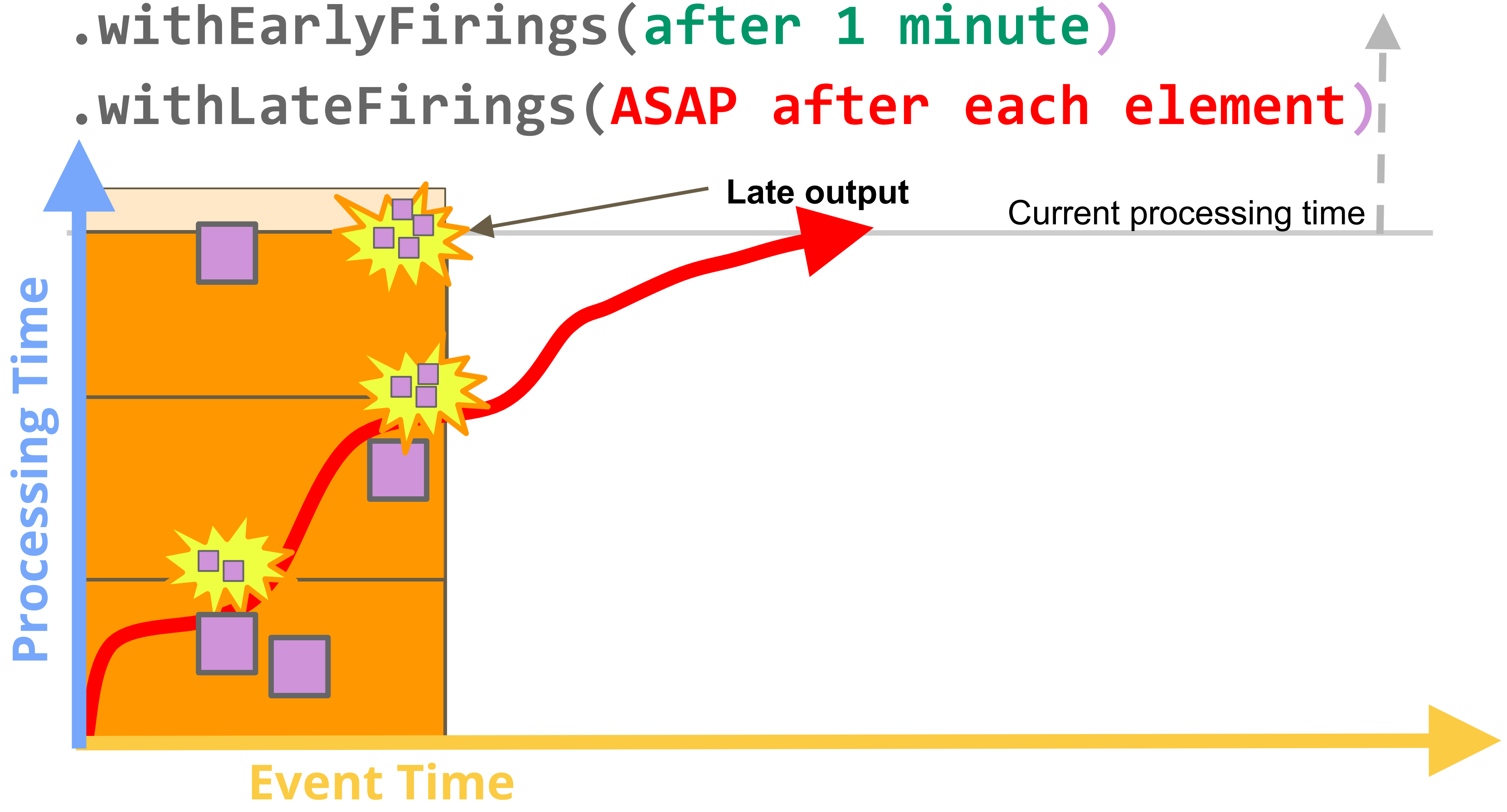
```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



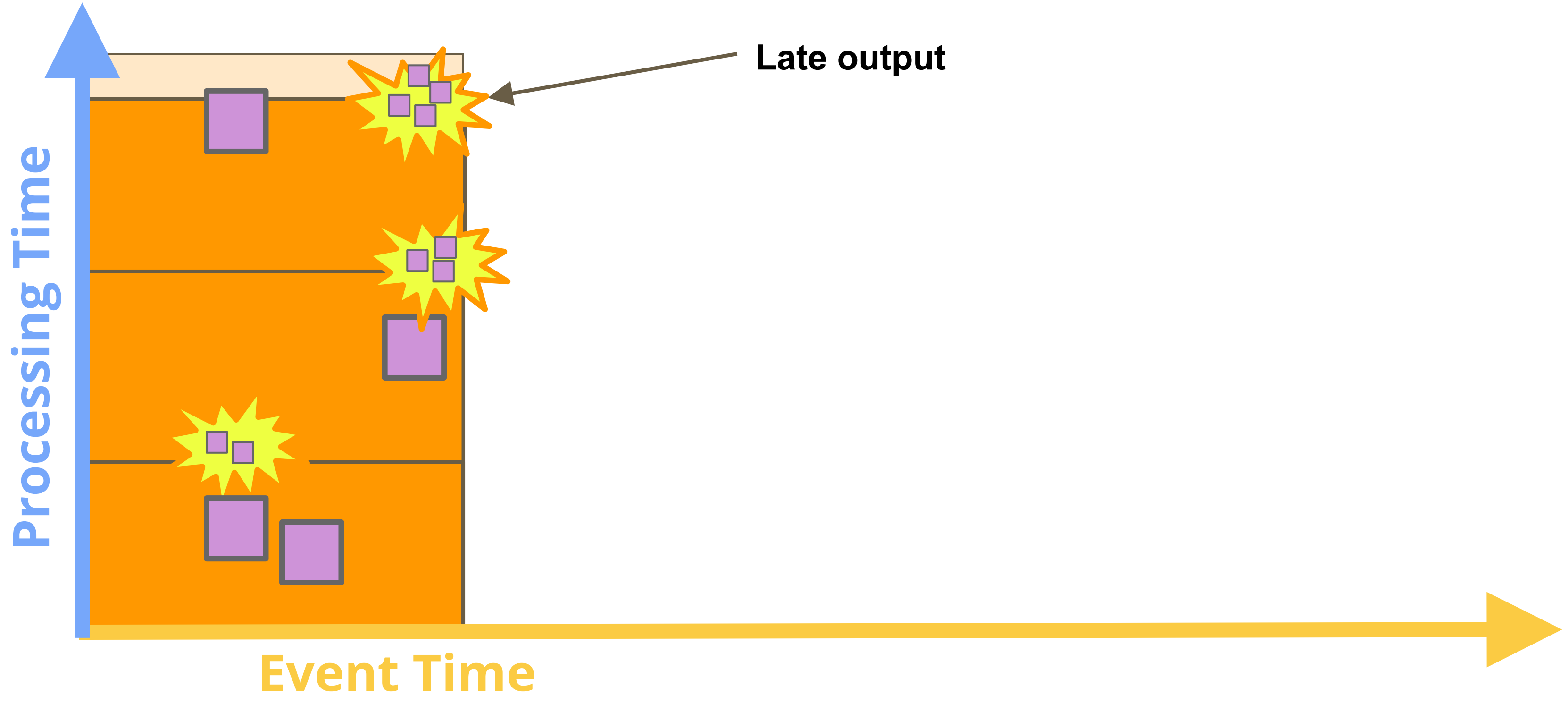
```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



```
.withEarlyFirings(after 1 minute)
.withLateFirings(ASAP after each element)
```



The Beam Model: Asking the Right Questions

What are you computing?

Where in event time?

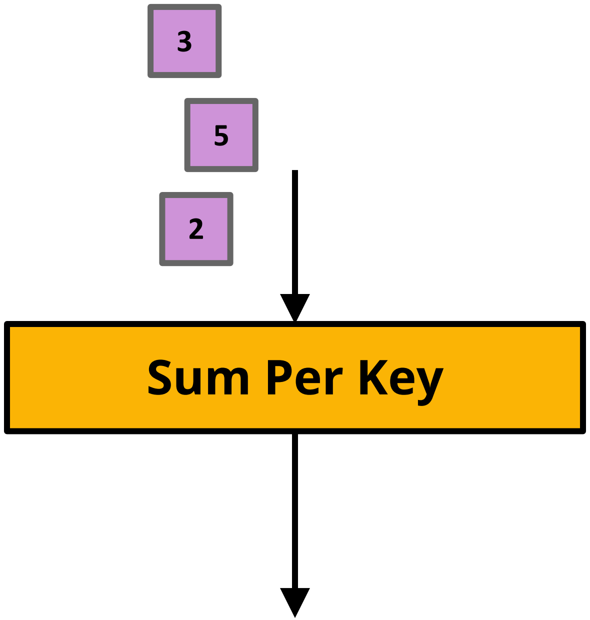
When in processing time are results produced?

How do refinements relate?

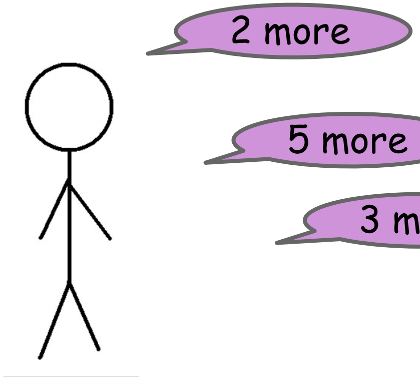
Accumulation Mode

How do refinements relate?

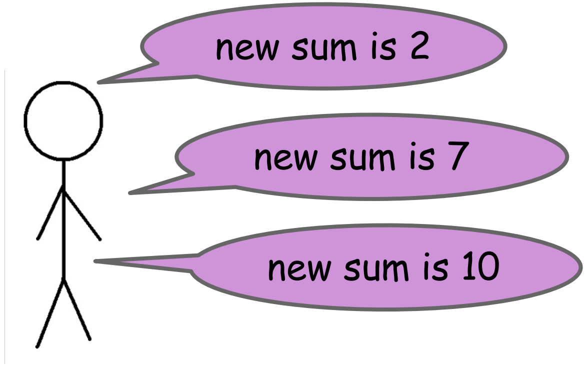
How should multiple outputs per window accumulate?
Appropriate choice depends on consumer.




```
Window.into(...)
  .triggering(...)
  .discardingFiredPanes()
```




```
Window.into(...)
  .triggering(...)
  .accumulatingFiredPanes()
```



How do refinements relate? A more detail Example

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

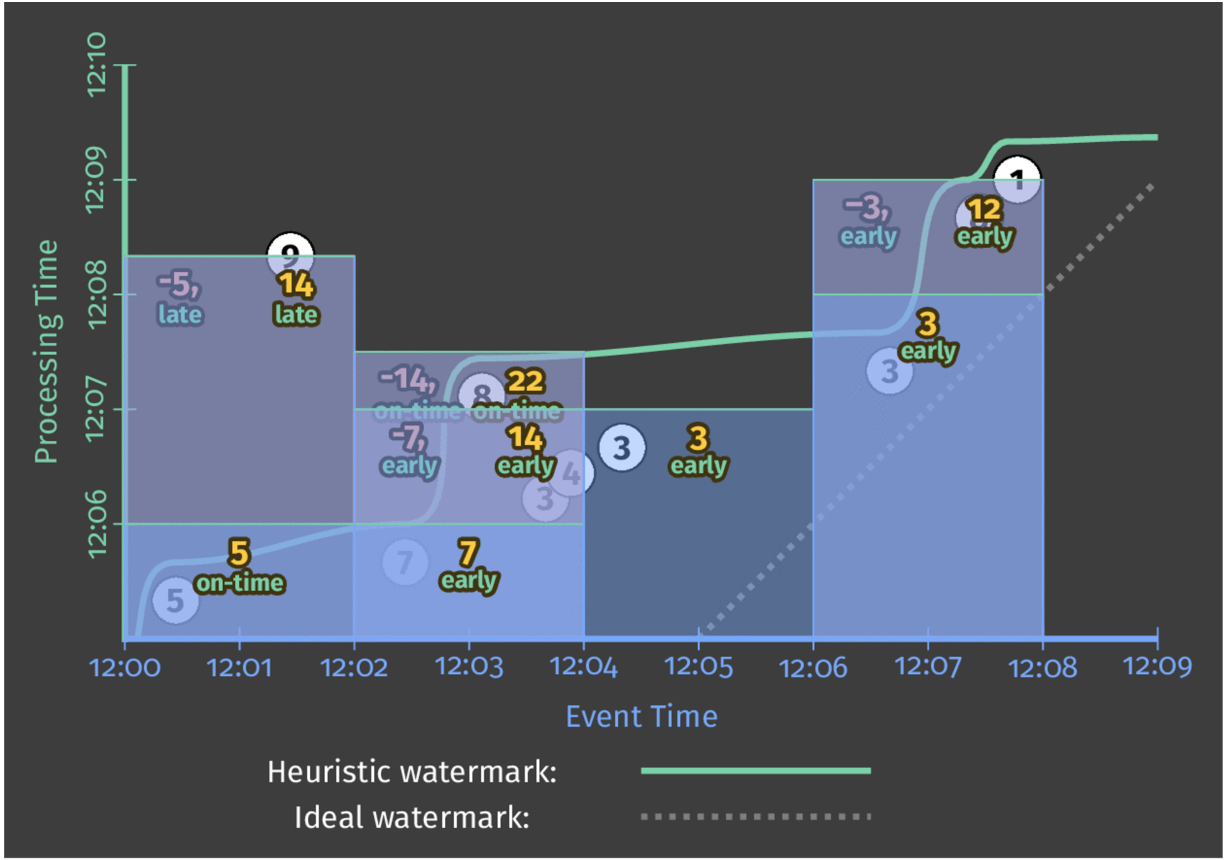
Firing	Elements	Discarding	Accumulating	Acc. & Retracting
Speculative	[3]	3	3	3
Watermark	[5, 1]	6	9	9, -3
Late	[2]	2	11	11, -9
Last Observed		***2***	***11***	***11***
Total Observed		***11***	***23***	***11***

(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



What can this
Generalized Stream Processing model
(aka the Beam model)
offer ?

What Where When How

Correctness

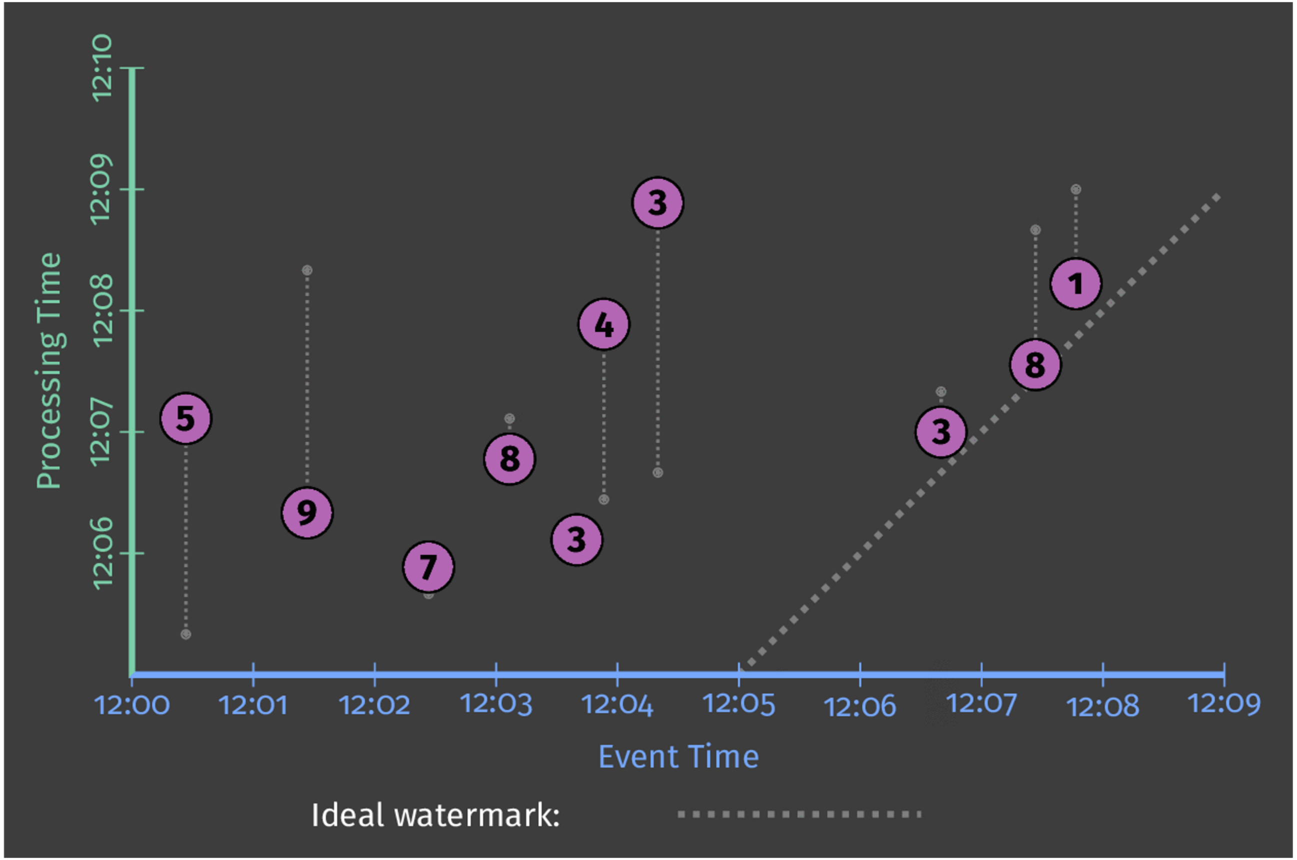
Power

Composability

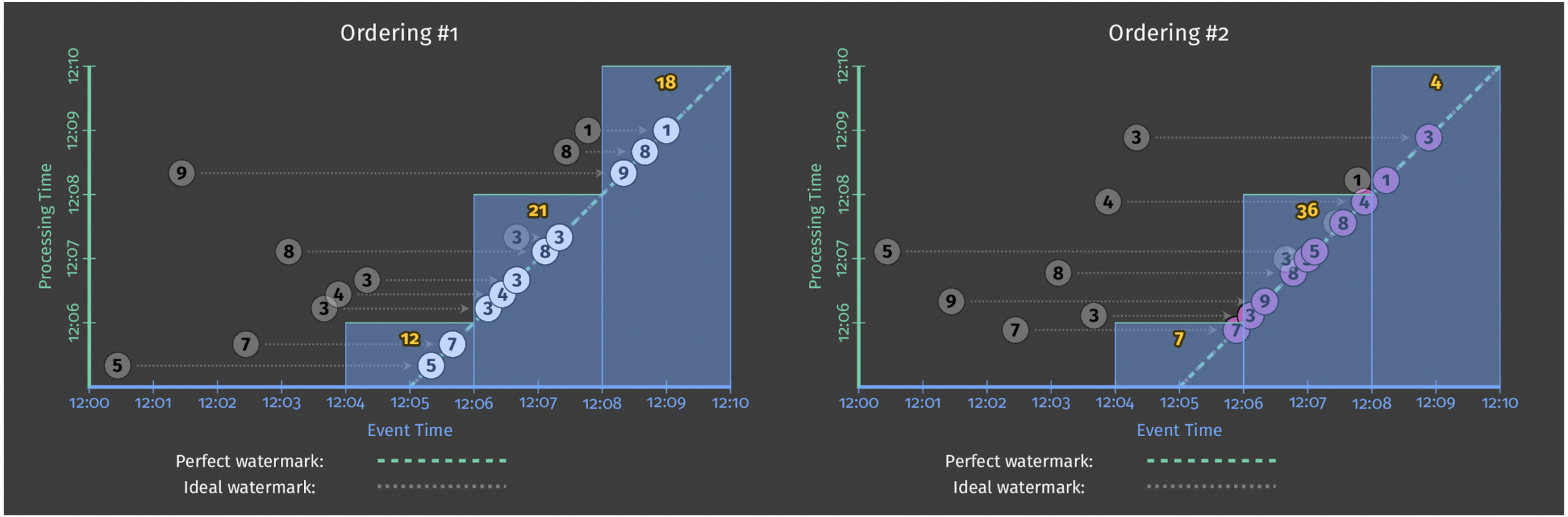
Flexibility

Modularity

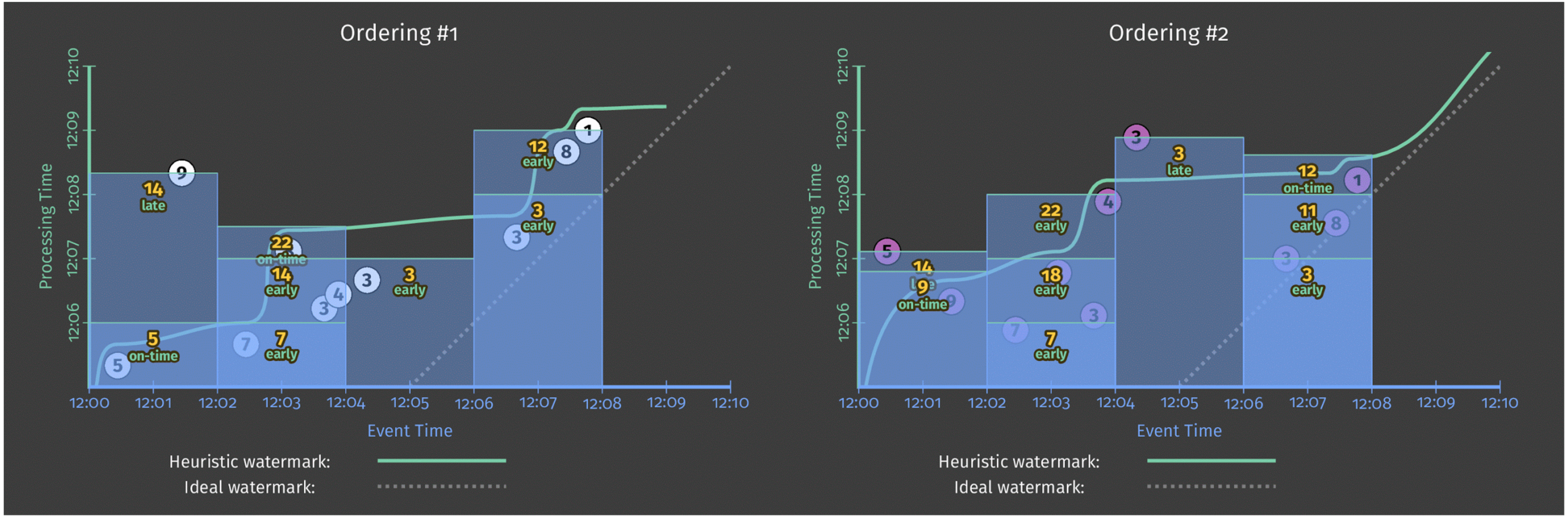
Distributed Systems are Distributed



Processing Time Results Differ



Event Time Results are Stable



What **Where** **When** **How**

Correctness

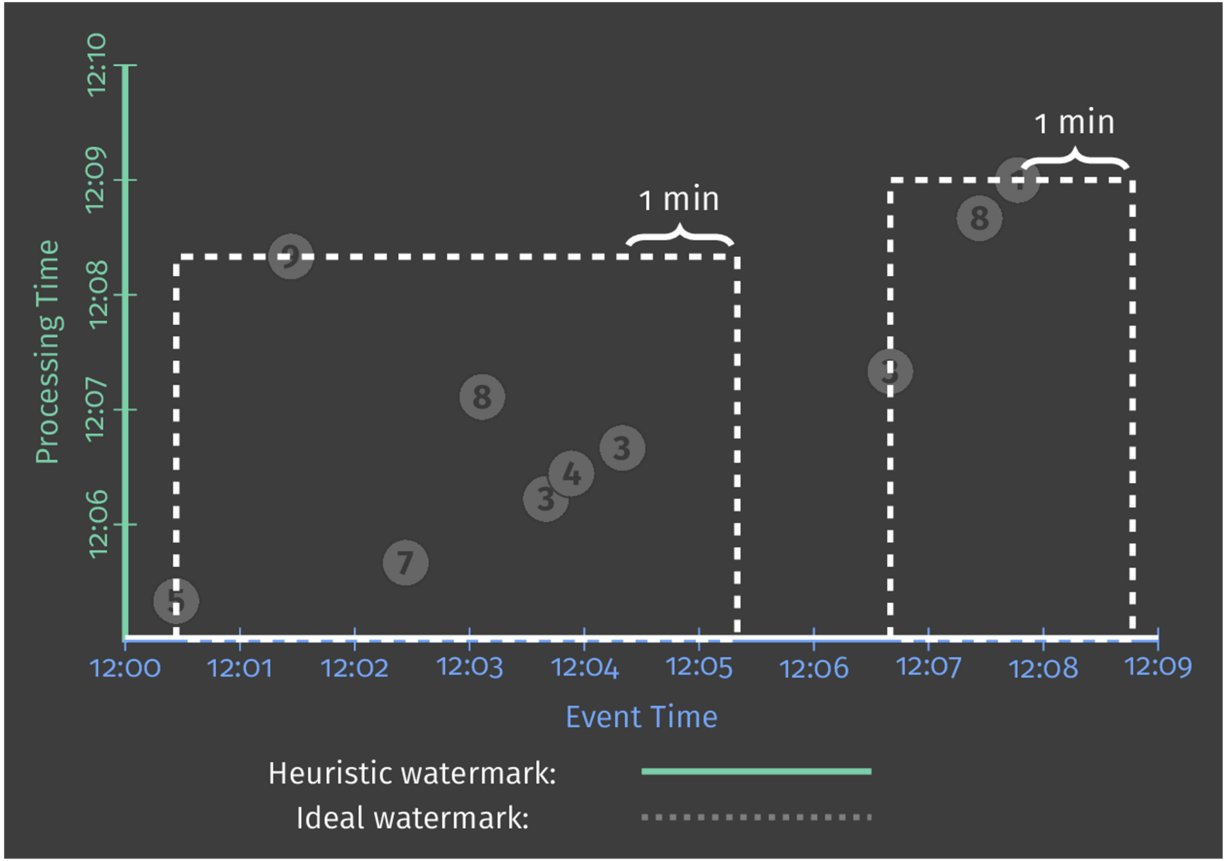
Power

Composability

Flexibility

Modularity

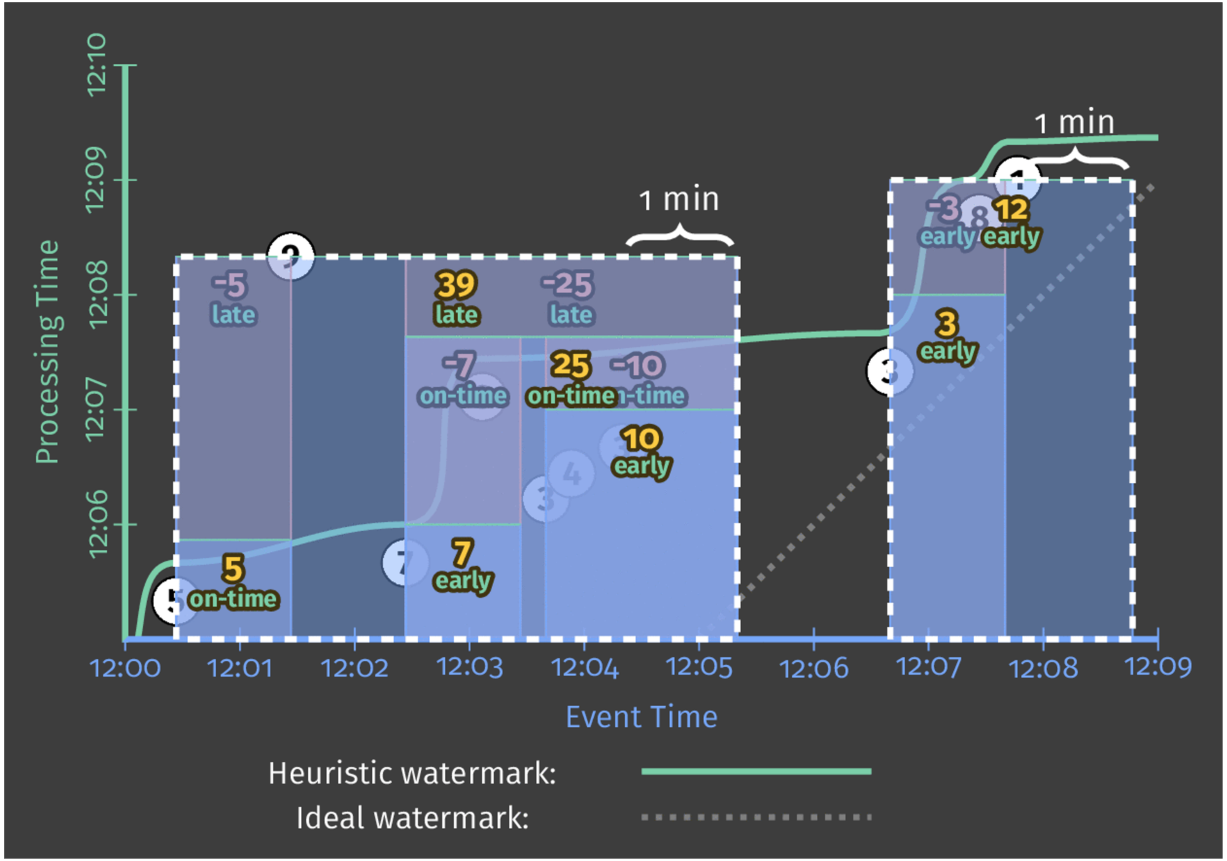
Identifying Bursts of User Activity



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



What **Where** **When** **How**

Correctness

Power

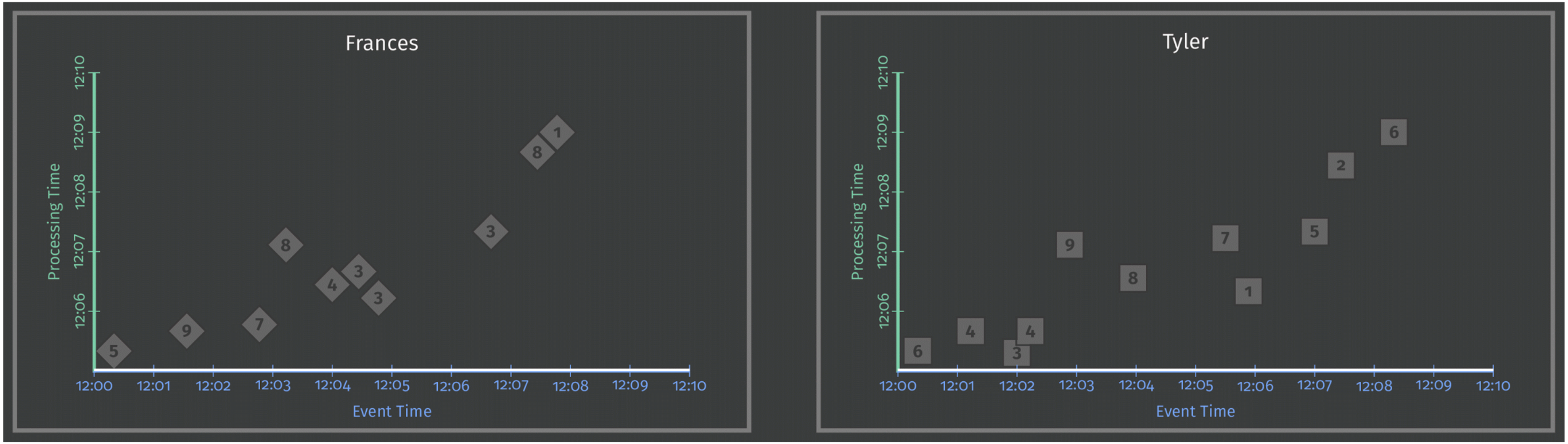
Composability

Flexibility

Modularity

Calculating Session Lengths

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


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

What Where When How

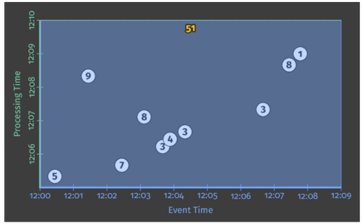
Correctness

Power

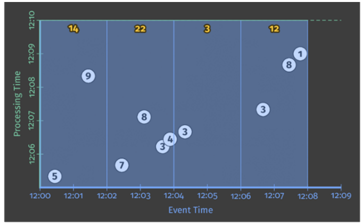
Composability

Flexibility

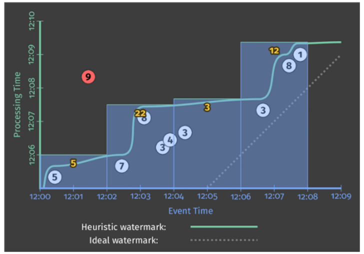
Modularity



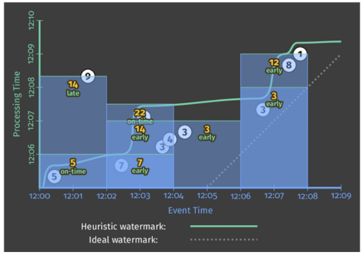
1.Classic Batch



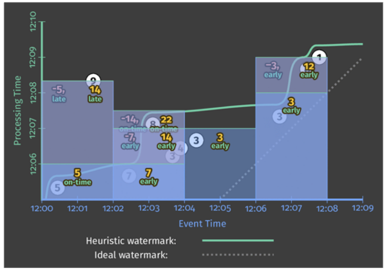
2. Batch with Fixed Windows



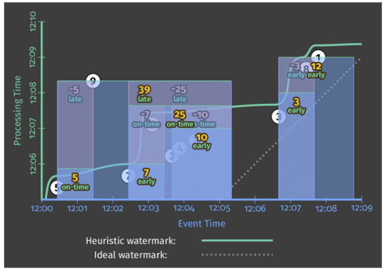
3. Streaming



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

```
tion<KV<String, Integer>> scores = input
ply(Window.into(FixedWindows.of(Minutes(2))
    .triggering(AtWatermark()))
pply(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

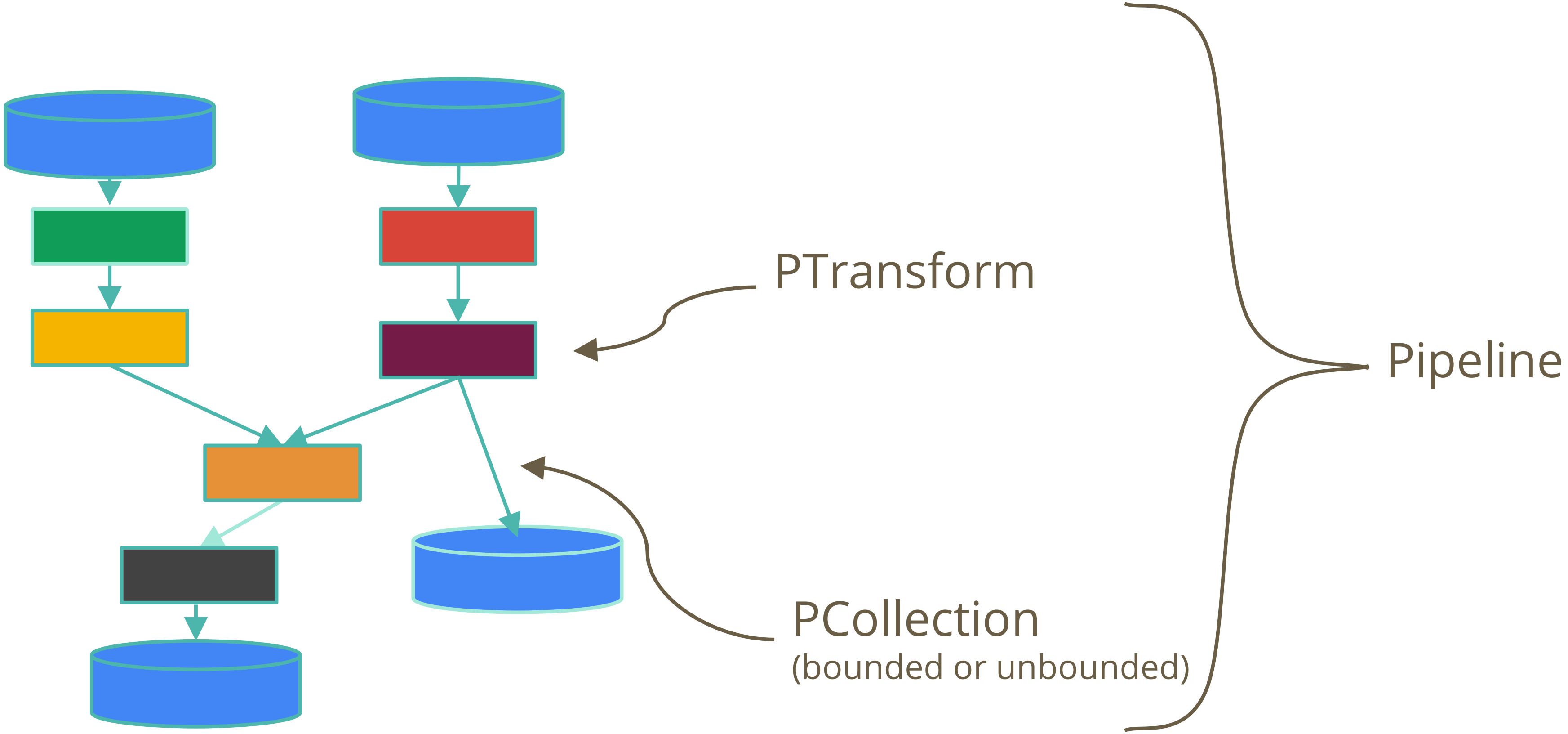
Composability

Flexibility

Modularity

Apache Beam

A Beam Computational Pipeline



The Beam Vision

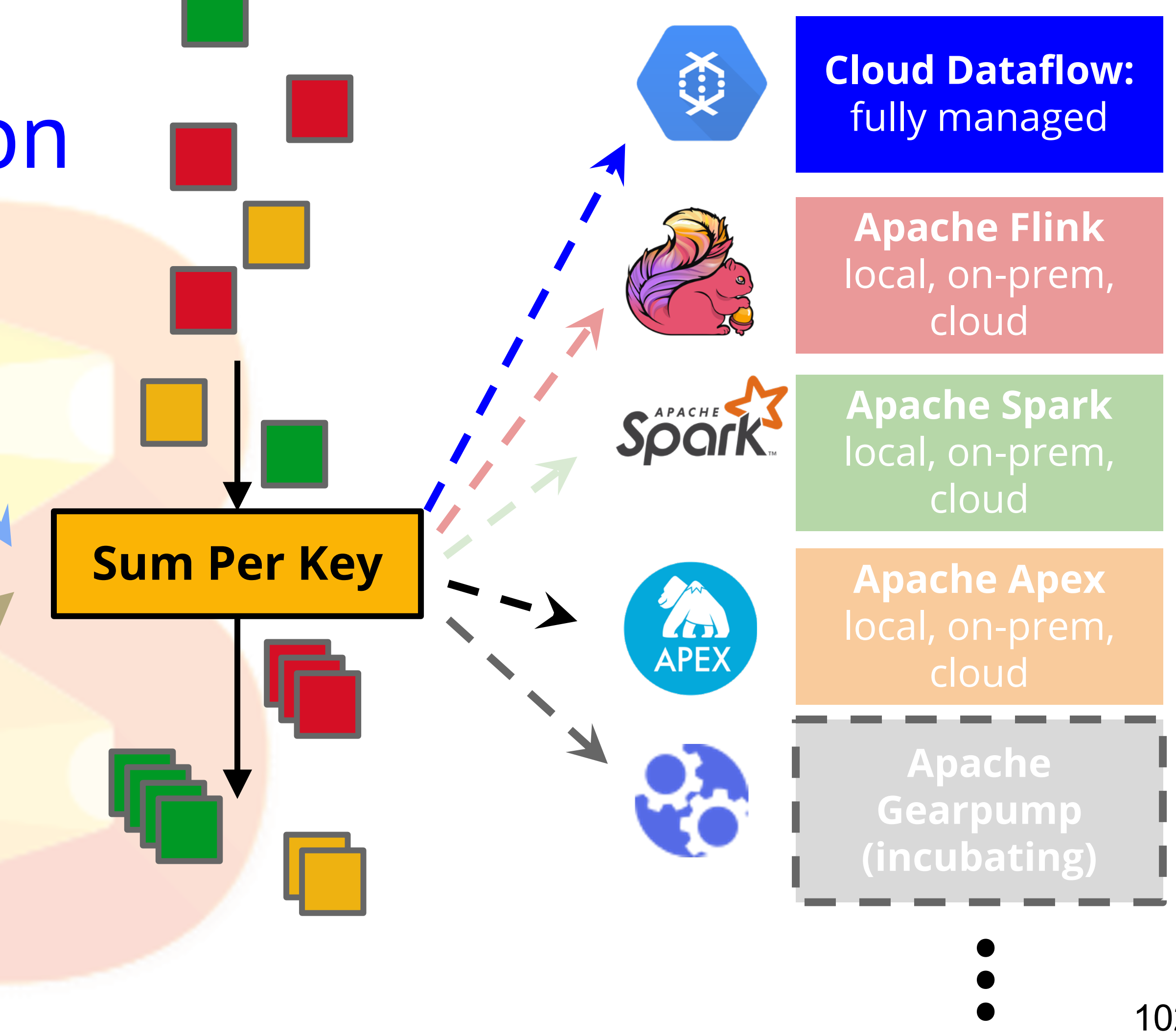
Java

```
input.apply(
  Sum.integersPerKey())
```

Python

```
input | Sum.PerKey()
```

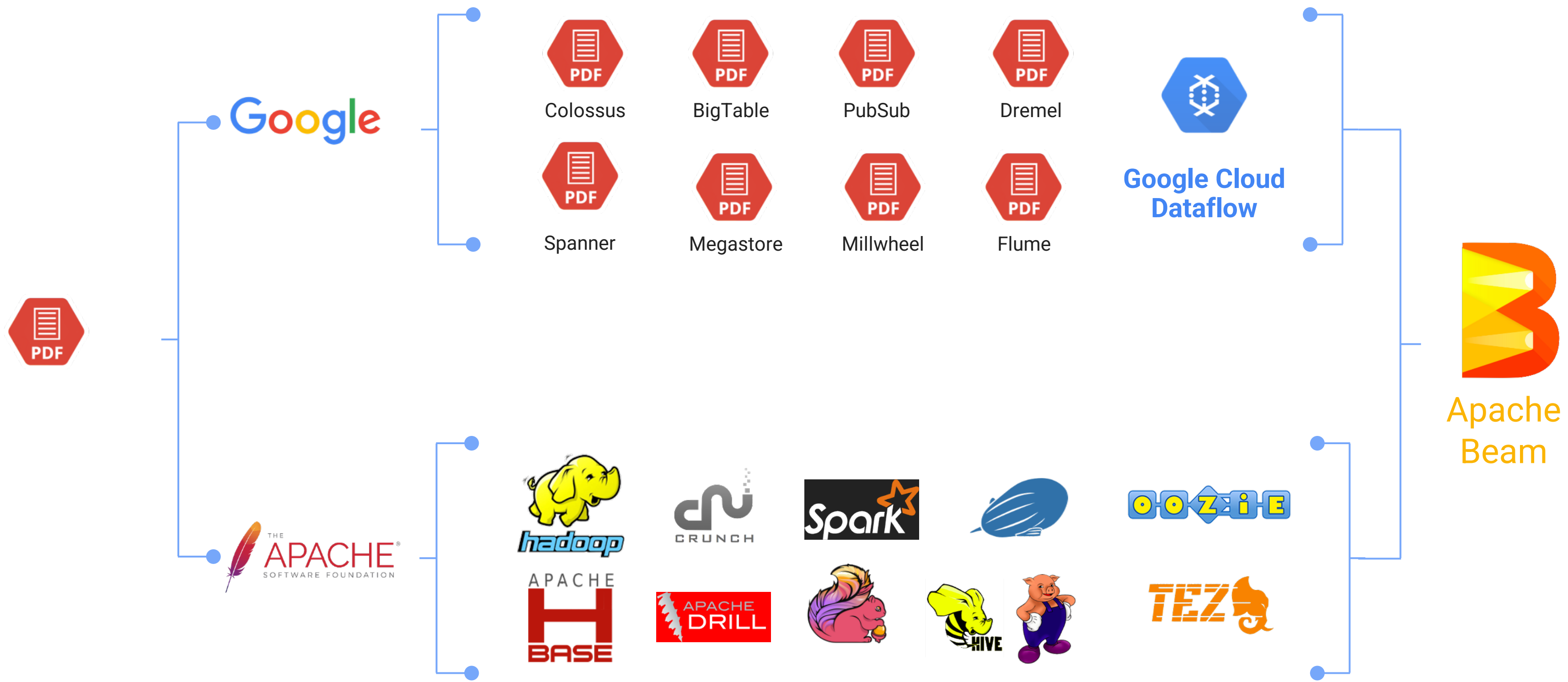
⋮



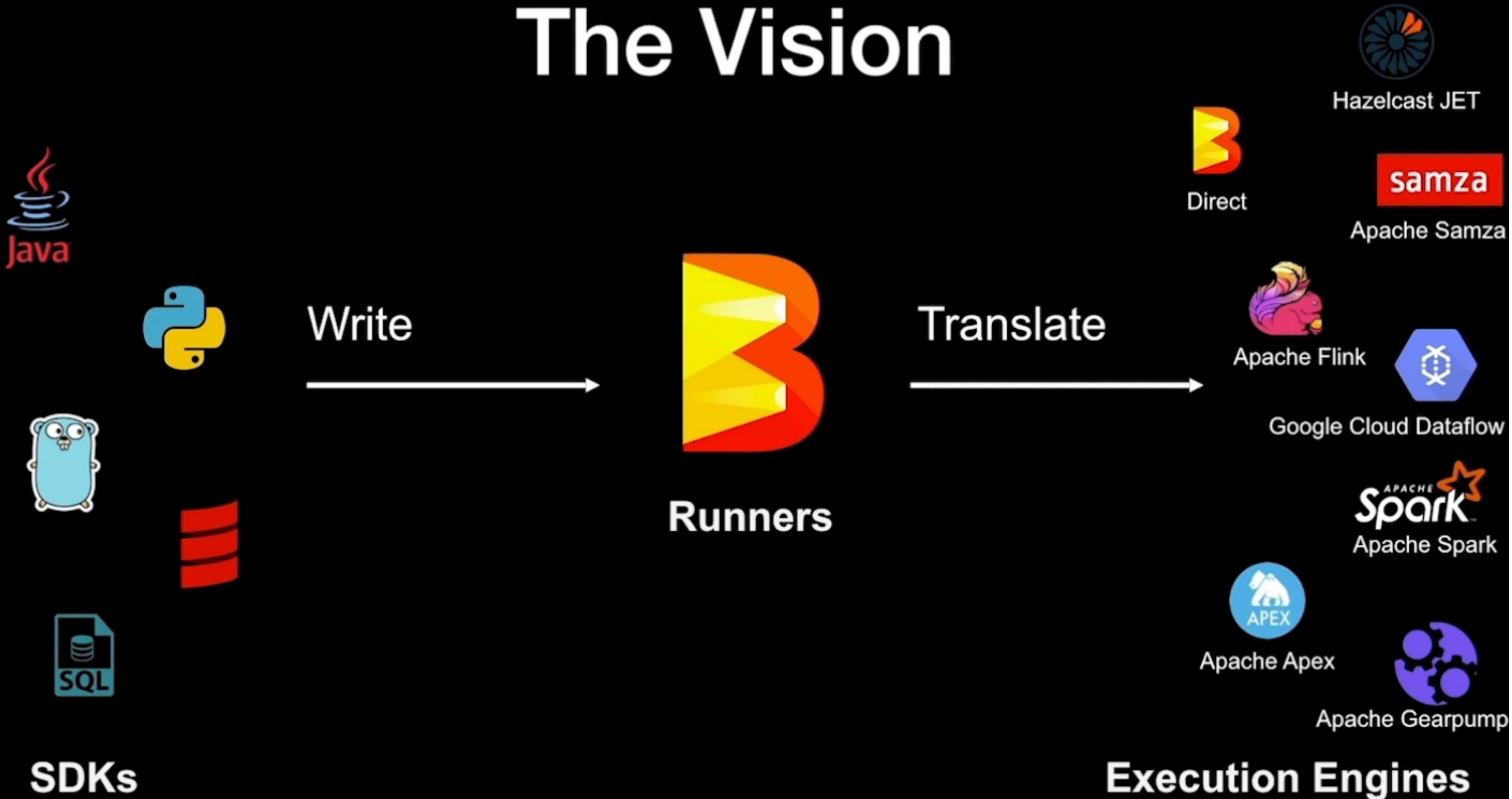
What your (Java) Beam code Looks Like

```java
Pipeline p = Pipeline.create(options);

p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))

 .apply(FlatMapElements.via(line -> Arrays.asList(line.split("[^a-zA-Z']+"))))

 .apply(Filter.byPredicate(word -> !word.isEmpty()))

 .apply(Count.perElement())

 .apply(MapElements.via(count -> count.getKey() + ": " + count.getValue())

 .apply(TextIO.Write.to("gs://..."));

p.run();
```

The Evolution of Beam



The Vision


Java
Write
Translate
Runners
SQL
SDKs
Hazelcast JET
Direct
samza
Apache Samza
Apache Flink
Google Cloud Dataflow
Spark
Apache Spark
APEX
Apache Apex
Apache Gearpump
Execution Engines

What is Part of Apache Beam?

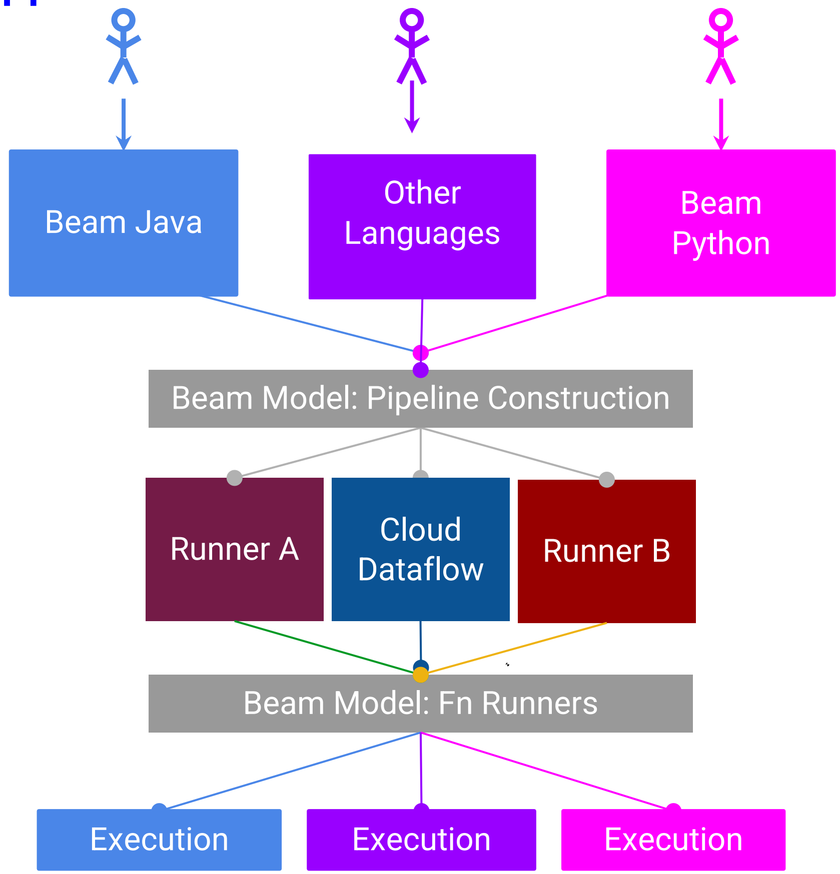
1. The Beam Model: **What** **Where** **When** **How**

2. SDKs for writing Beam pipelines -- Java and Python

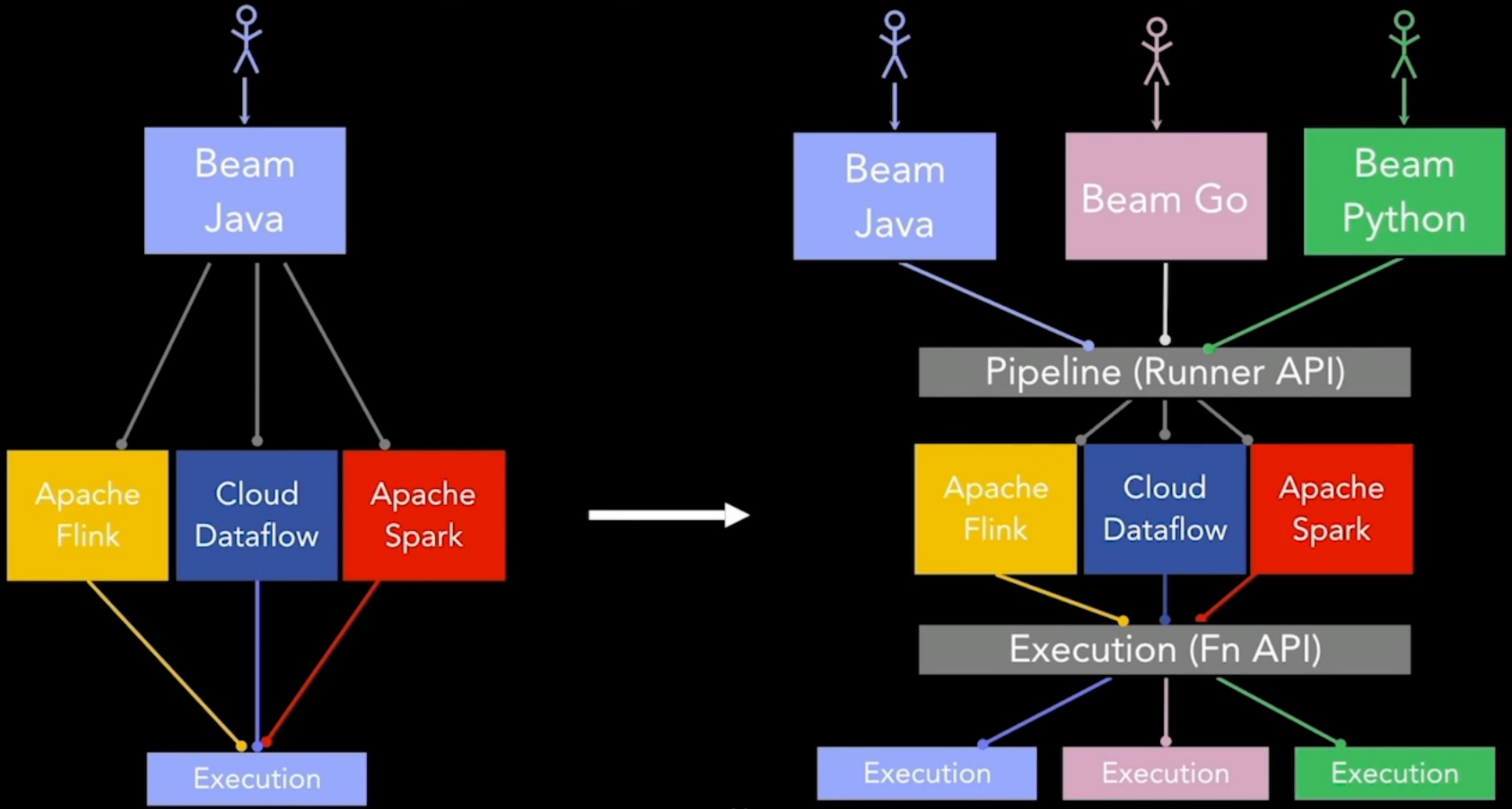
3. Runners for Existing Distributed Processing Backends
   - Apache Flink
   - Apache Spark
   - Google Cloud Dataflow
   - Direct runner for local development and testing
   - In development: Apache Gearpump and Apache Apex

Apache Beam Technical Vision

1. **The Beam Model:** the abstractions at the core of Apache Beam
2. **End users:** who want to write pipelines or transform libraries in a language that's familiar.
3. **SDK writers:** who want to make Beam concepts available in new languages.
4. **Runner writers:** who have a distributed processing environment (on-prem/ cloud, open-source/ closed-source) and want to support Beam pipelines
5. **A Runner platform (e.g. Flink)** may also make the power of the Beam model available to native users of the platform by extending the platform's native APIs.



Language-Portability



Example Beam Runners

Apache Spark

- Open-source cluster-computing framework
- Large ecosystem of APIs and tools
- Runs on premise or in the cloud

Apache Flink

- Open-source distributed data processing engine
- High-throughput and low-latency stream processing
- Runs on premise or in the cloud

Google Cloud Dataflow

- Fully-managed service for batch and stream data processing
- Provides dynamic auto-scaling, monitoring tools, and tight integration with Google Cloud Platform

Comparing Runner Capabilities

What is being computed?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo
ParDo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GroupByKey	✓	✓	✓	~	✓	✓	✓	✓	✓	✓	✓
Flatten	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Combine	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Composite Transforms	✓	~	~	~	~	~	✓	✓	~	~	✓
Side Inputs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Source API	✓	✓	✓	✓	✓	✓	~	✓	✓	✓	✓
Splittable DoFn (SDF)	~	✓	✓	~	~	~	✕	✕	✕	~	✕
Metrics	~	~	~	~	✕	✕	~	~	~	~	✕
Stateful Processing	✓	~	~	✕	~	✕	~	~	~	~	✕

Where in event time?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo
Global windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Fixed windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Sliding windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Session windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Custom windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Custom merging windows	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Timestamp control	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Latest version available at: http://beam.apache.org/ documentation/runners/ capability-matrix

Comparing Runner Capabilities

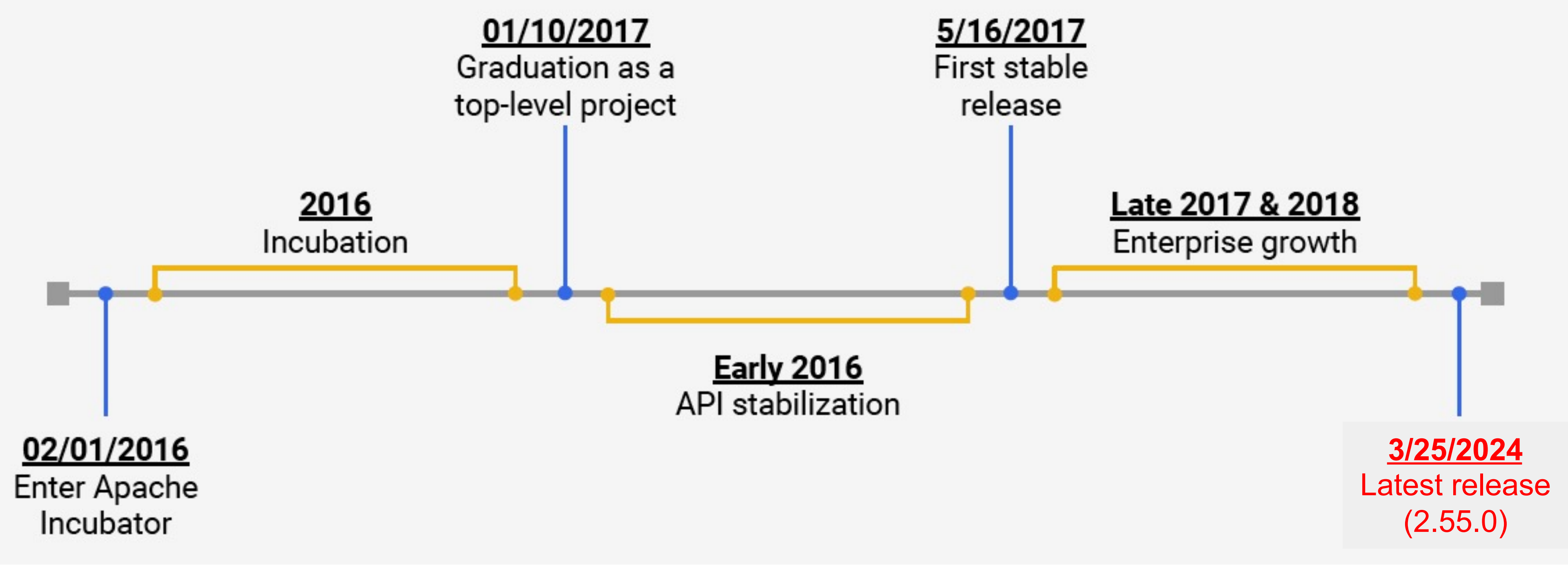
When in processing time?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo
Configurable triggering	✓	✓	✓	✕	✓	✕	✕	✓	✓	✓	✓
Event-time triggers	✓	✓	✓	✕	✓	✓	✕	✓	✓	✓	✓
Processing-time triggers	✓	✓	✓	✓	✓	✕	✕	✓	✓	✓	✓
Count triggers	✓	✓	✓	✕	✓	✕	✕	✓	✓	✓	✓
[Meta]data driven triggers	✕ (BEAM-101)	✕	✕	✕	✕	✕	✕	✕	✕	✕	✕
Composite triggers	✓	✓	✓	✕	✓	✕	✕	✓	✓	✓	✓
Allowed lateness	✓	✓	✓	✕	✓	✓	✕	✓	✓	✓	✓
Timers	✓	~	~	✕	✕	✕	✕	~	~	✕	✕

How do refinements relate?

	Beam Model	Google Cloud Dataflow	Apache Flink	Apache Spark	Apache Apex	Apache Gearpump	Apache Hadoop MapReduce	JStorm	IBM Streams	Apache Samza	Apache Nemo
Discarding	✓	✓	✓	✓	✓	✓	✕	✓	✓	✓	✓
Accumulating	✓	✓	✓	✕	✓	✕	✕	✓	✓	✓	✓
Accumulating & Retracting	✕ (BEAM-91)	✕	✕	✕	✕	✕	✕	✕	✕	✕	✕

Progress of Apache Beam



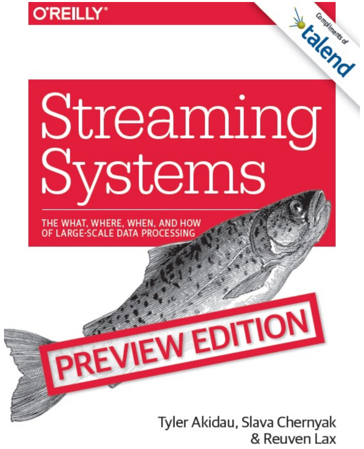
https://beam.apache.org/blog/beam-2.55.0/

Milestones of Apache Beam (circa: Aug 2021)

Learn More !

Free Book on Key Streaming Concepts and Apache Beam

http://asiandatascience.com/wp-content/uploads/2018/01/WP_EN_BD_OReilly_Streaming_Systems.pdf



Two Excellent Articles on Streaming Models and Beam
http://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
http://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

Apache Beam

http://beam.apache.org

Cloud Dataflow

http://cloud.google.com/dataflow/

Follow @ApacheBeam on Twitter

Beam Summit: https://2022.beamsummit.org