An Introduction to

The Beam Model

Apache Beam (incubating)

Agenda

1. Infinite, Out-of-order Data Sets
2. The Evolution of the Beam Model
3. What, Where, When, How
4. Apache Beam (incubating)

1 Infinite Out-of-Order Data Sets

Goals:

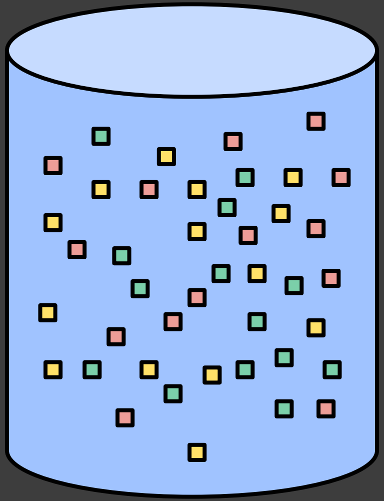
Write interesting computations

Run in both batch & streaming

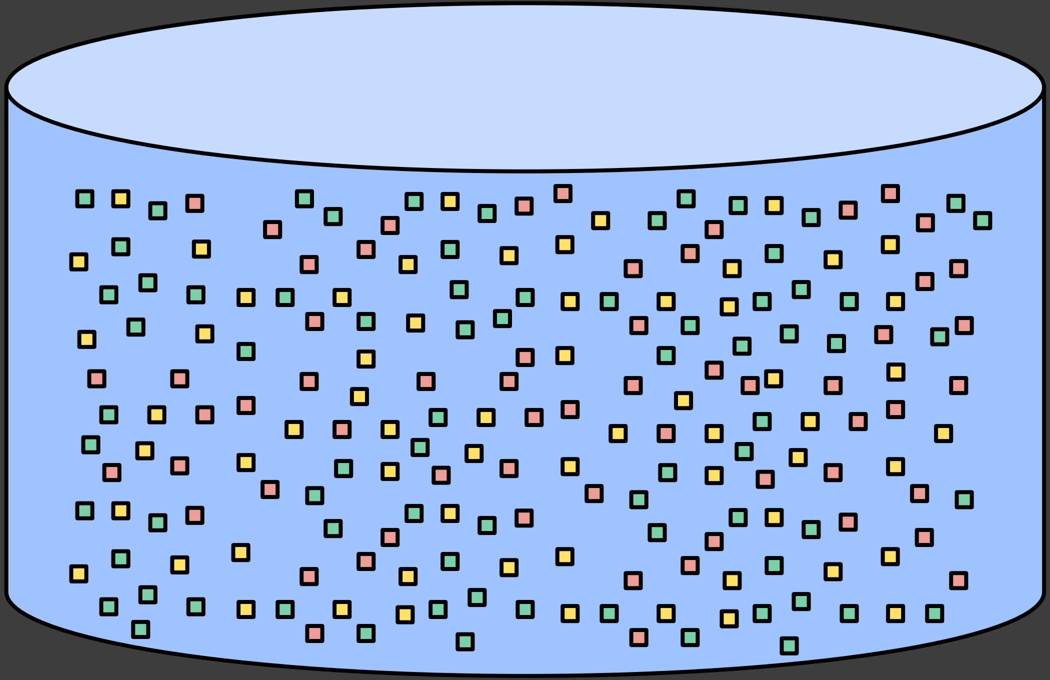
Use custom timestamps

Handle late data

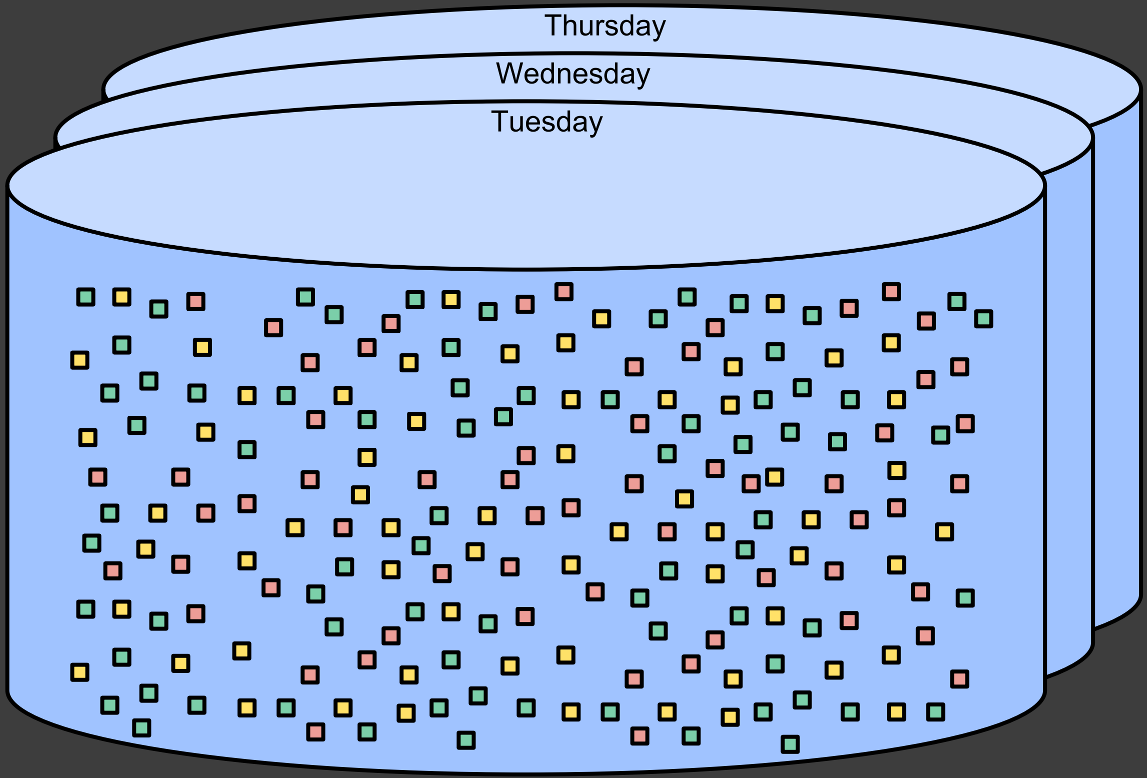
Data...



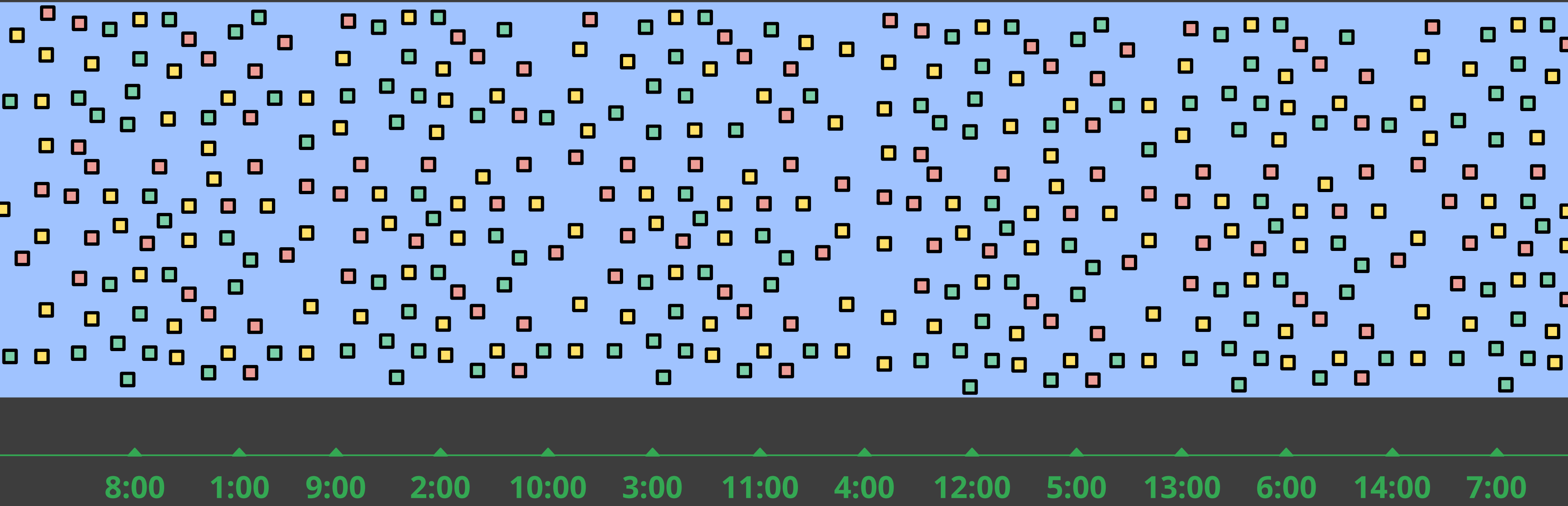
...can be big...



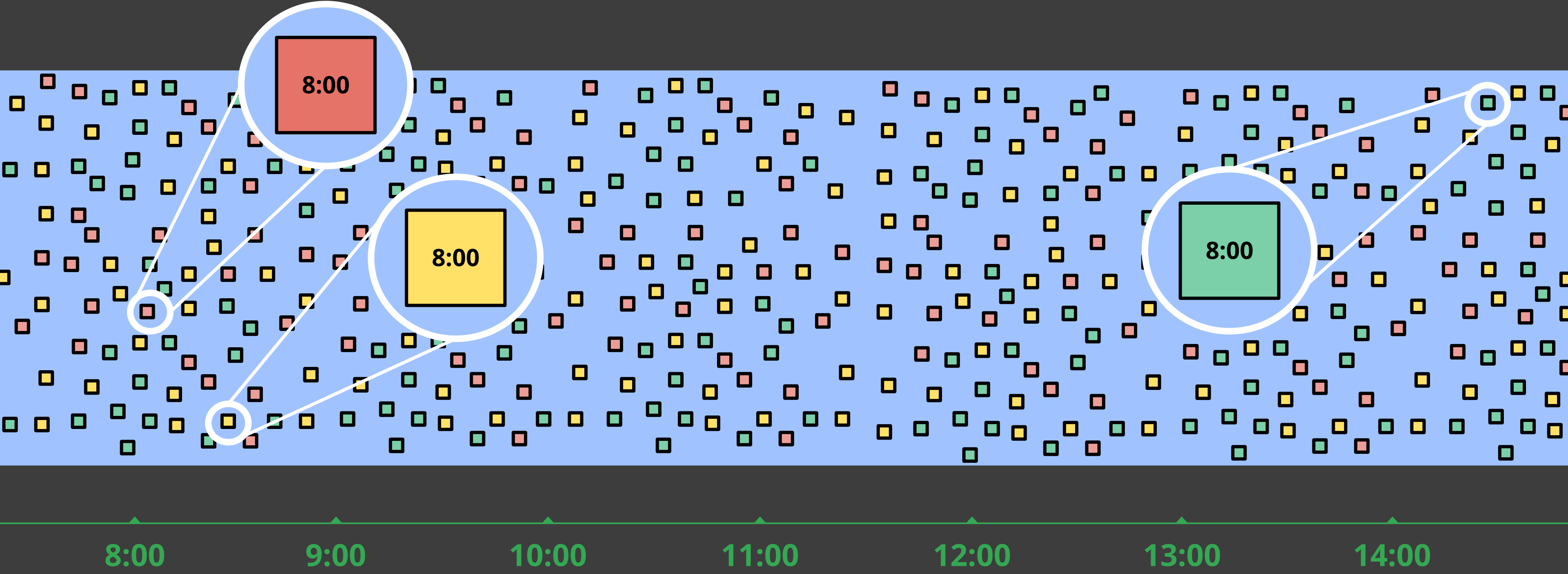
...really, really big...



… maybe infinitely big…



... with unknown delays.



Data Processing Tradeoffs

1+1=2

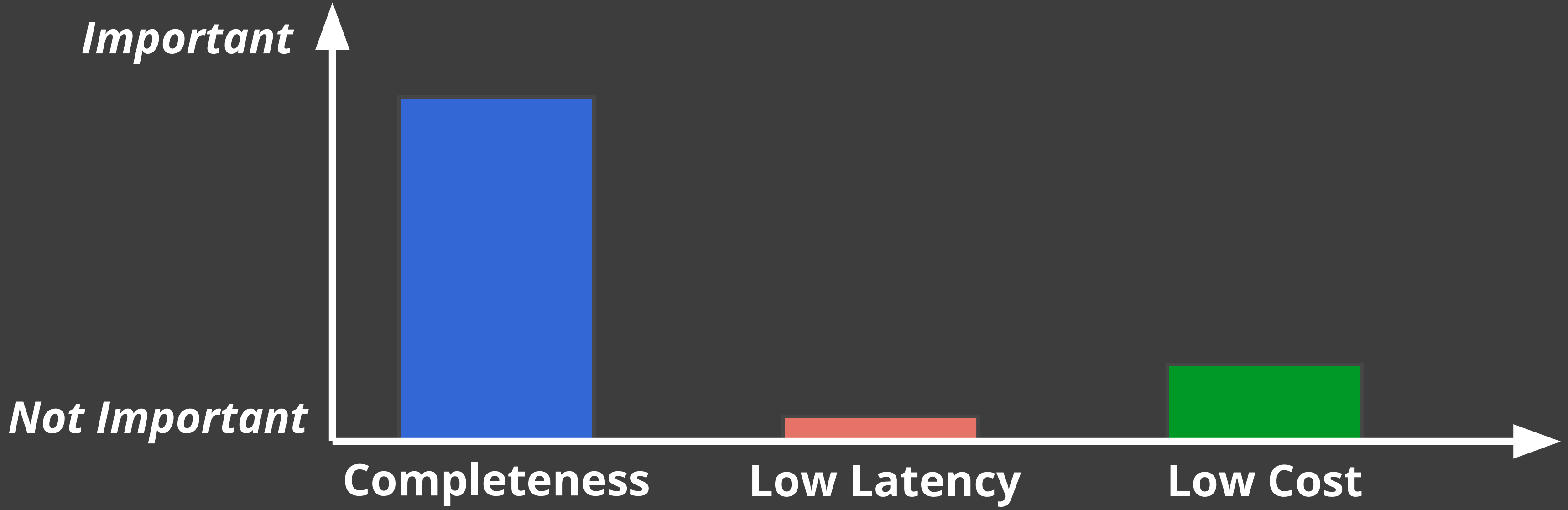
Completeness

Latency

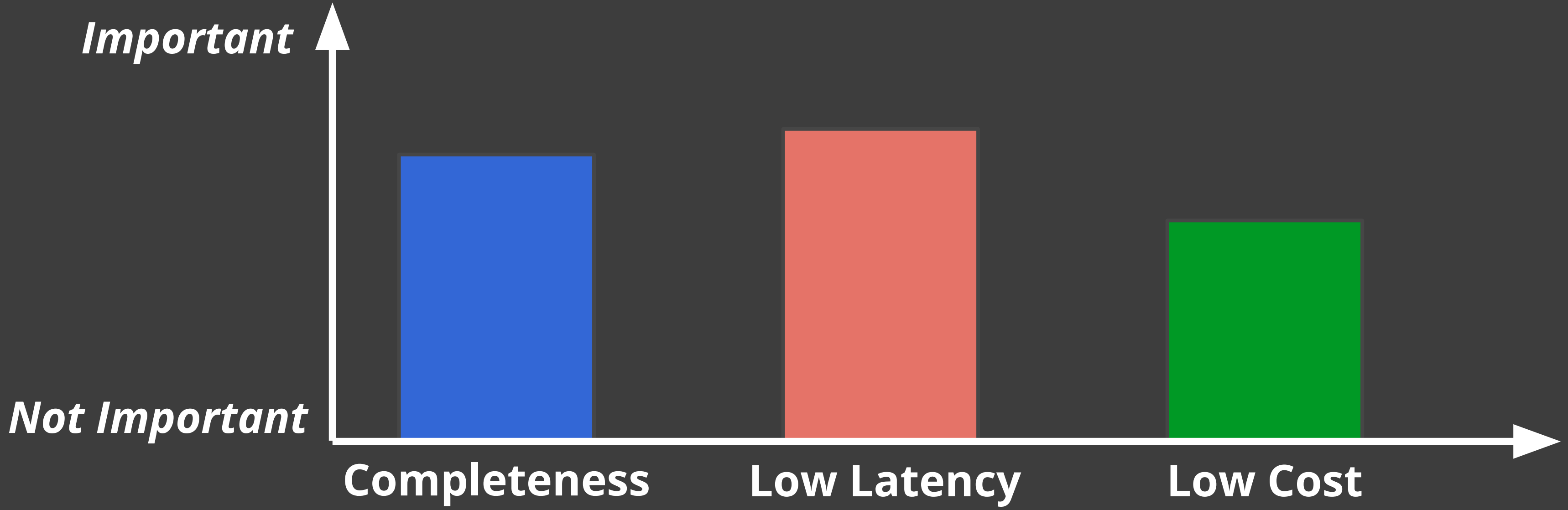
$$$

Cost

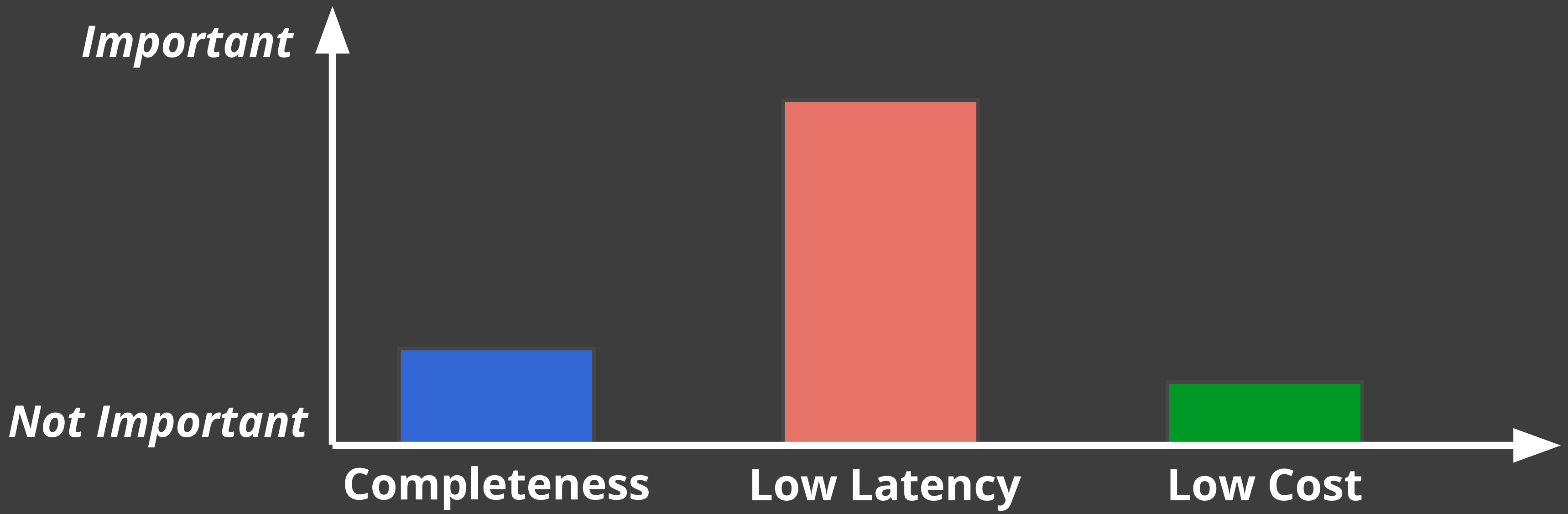
Requirements: Billing Pipeline



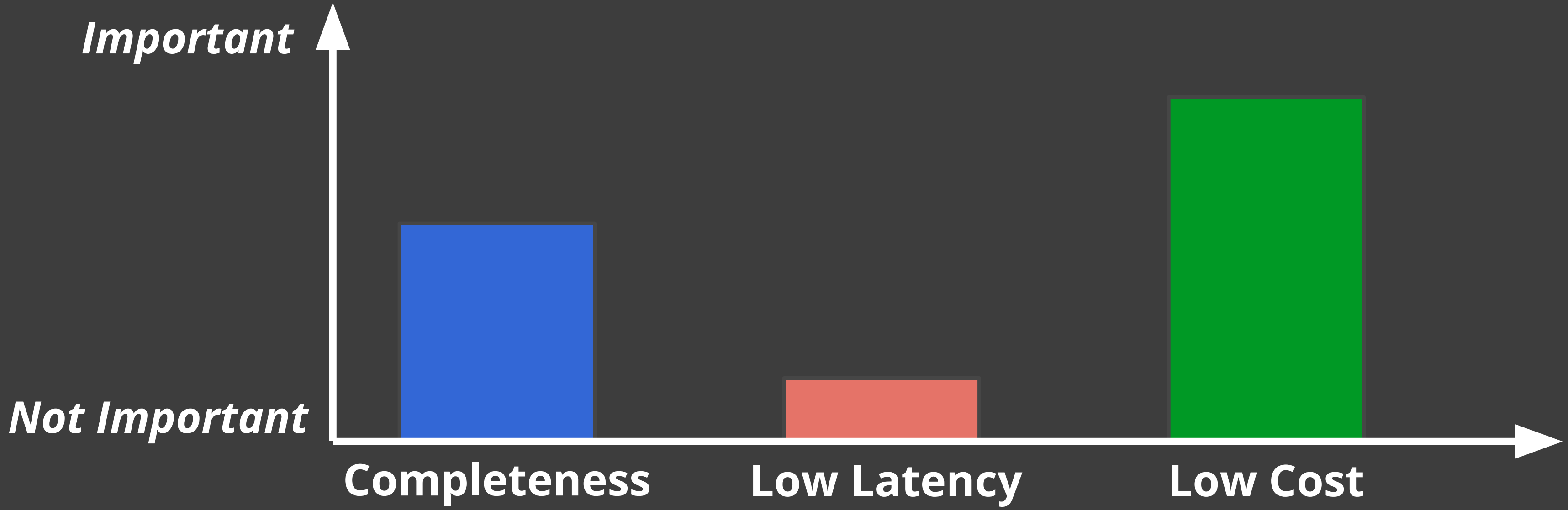
Requirements: Live Cost Estimate Pipeline



Requirements: Abuse Detection Pipeline



Requirements: Abuse Detection Backfill Pipeline



2 The Evolution of the Beam Model

MapReduce: Batch Processing

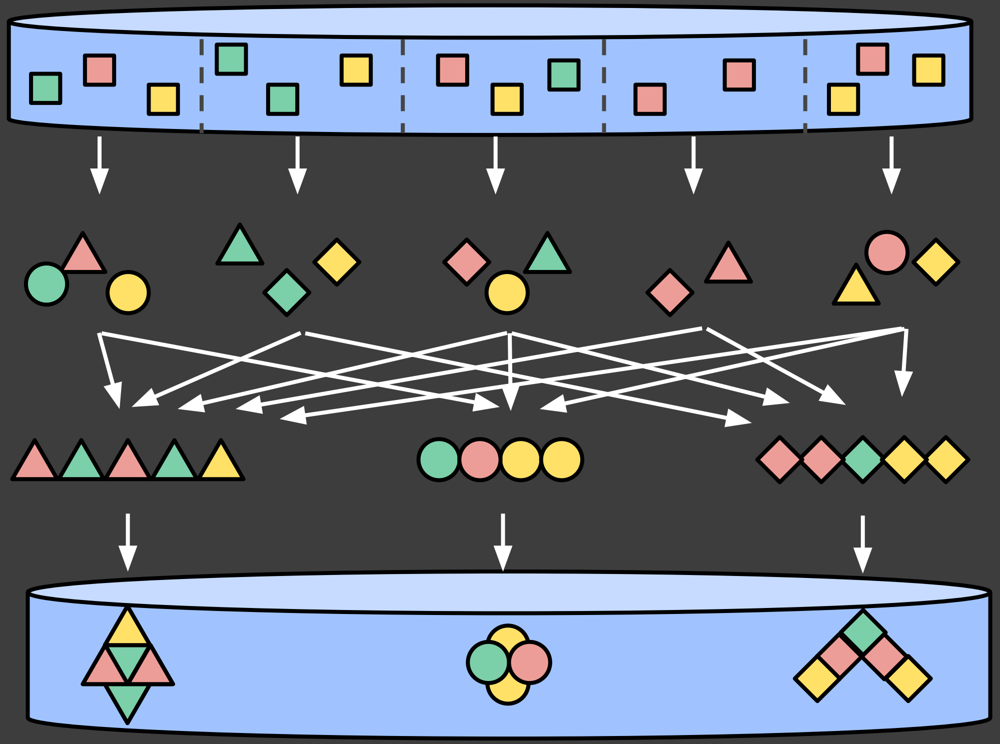
(Prepare)

Map

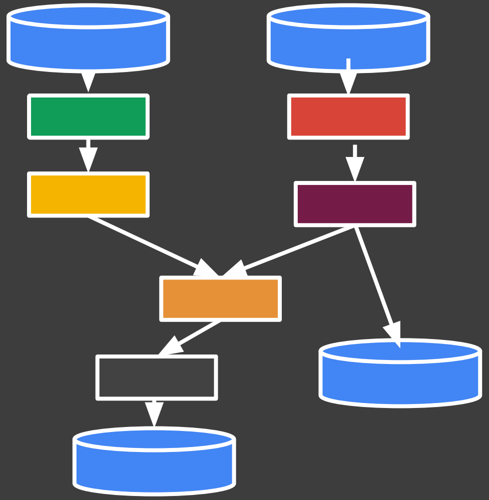
(Shuffle)

Reduce

(Produce)

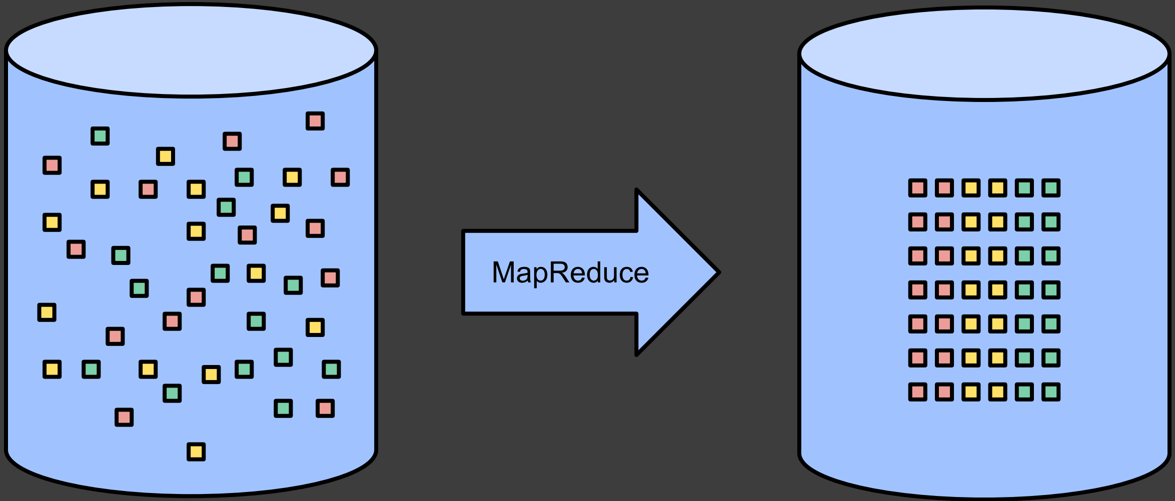


FlumeJava: Easy and Efficient MapReduce Pipelines

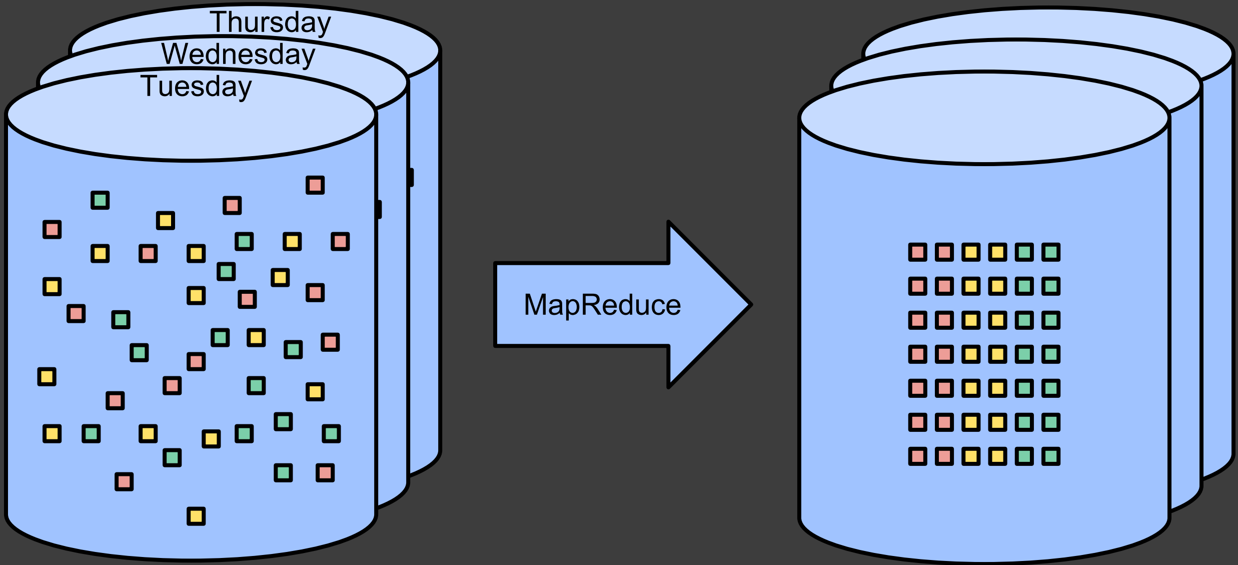


- Higher-level API with simple data processing abstractions.
  - Focus on what you want to do to your data, not what the underlying system supports.
- A graph of transformations is automatically transformed into an optimized series of MapReduces.

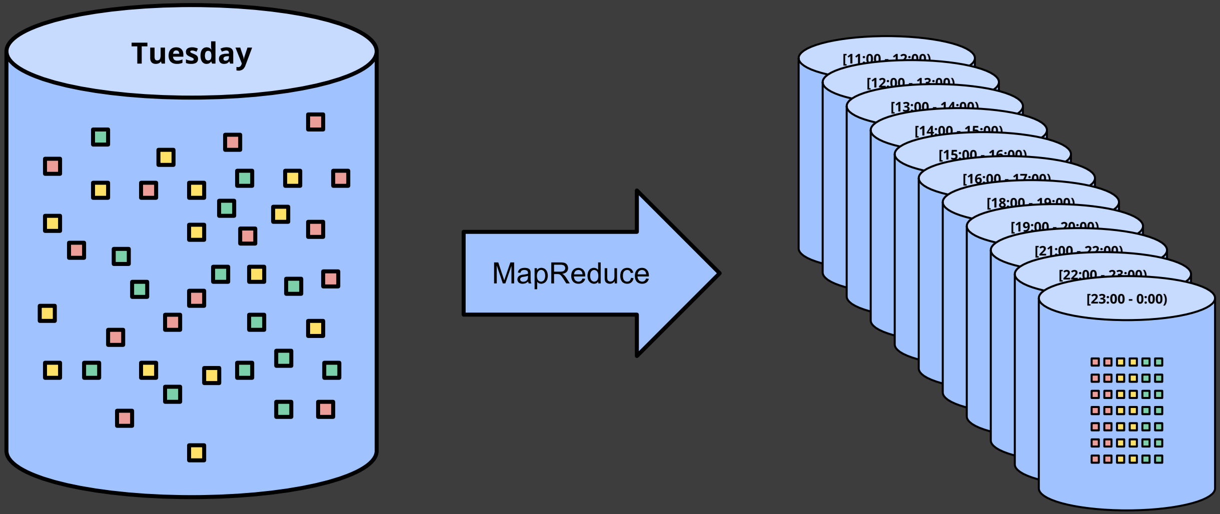
Batch Patterns: Creating Structured Data



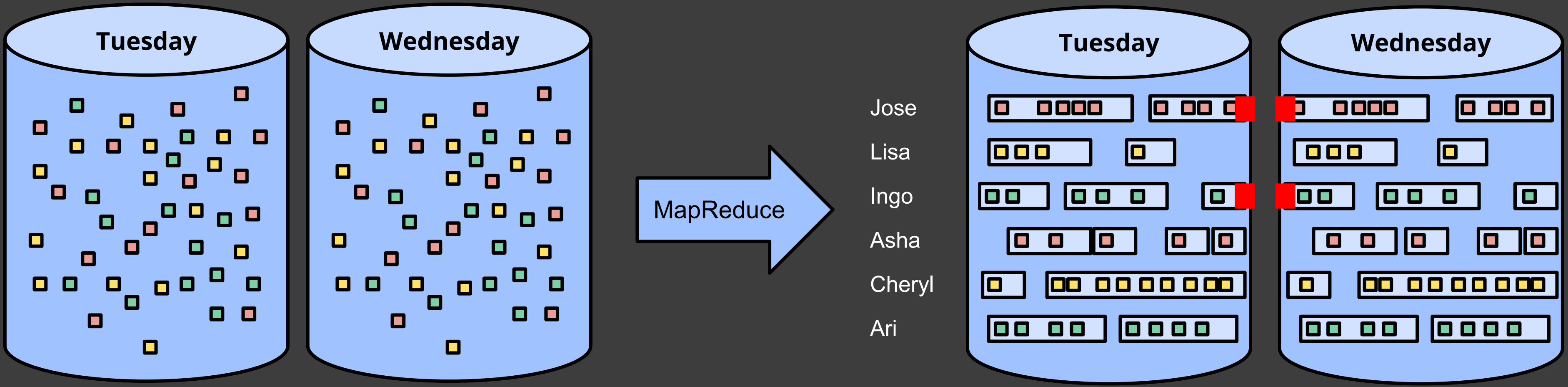
Batch Patterns: Repetitive Runs



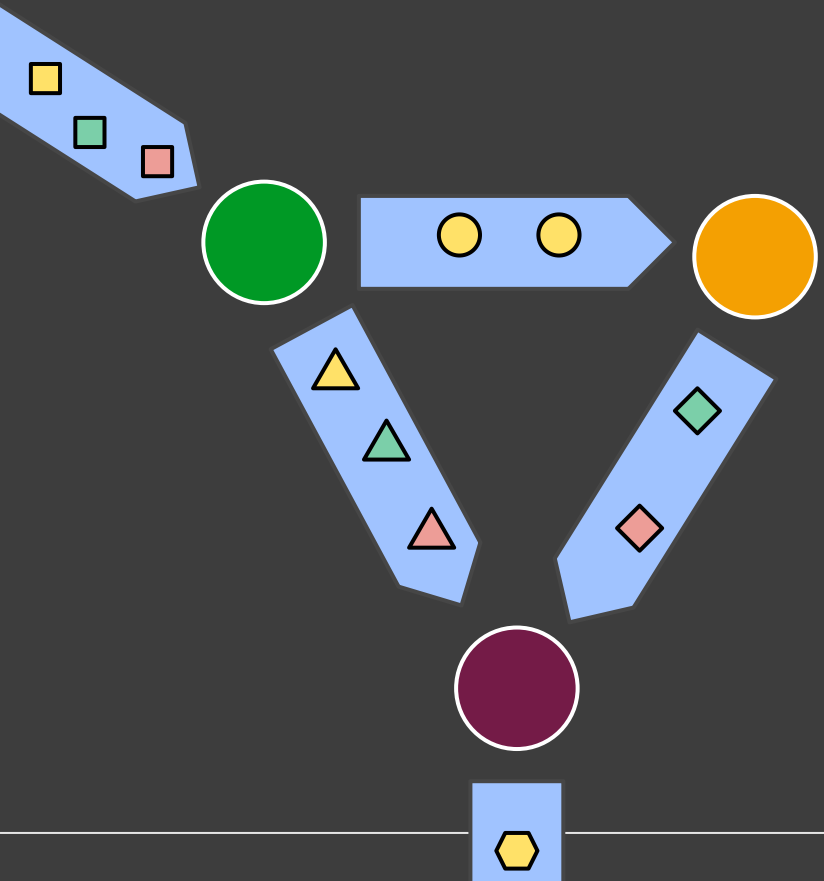
Batch Patterns: Time Based Windows



Batch Patterns: Sessions

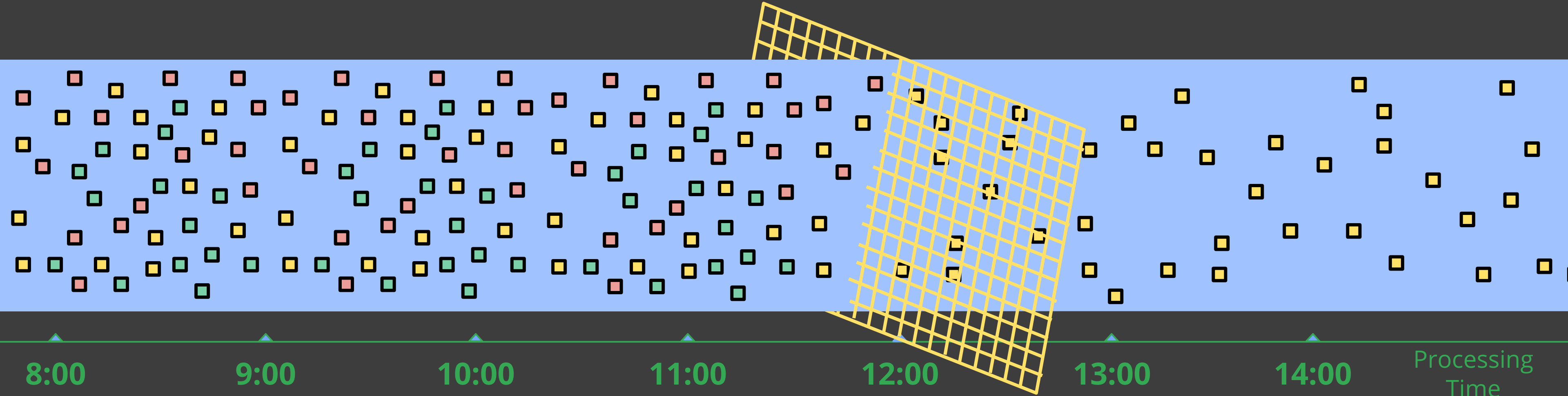


MillWheel: Streaming Computations

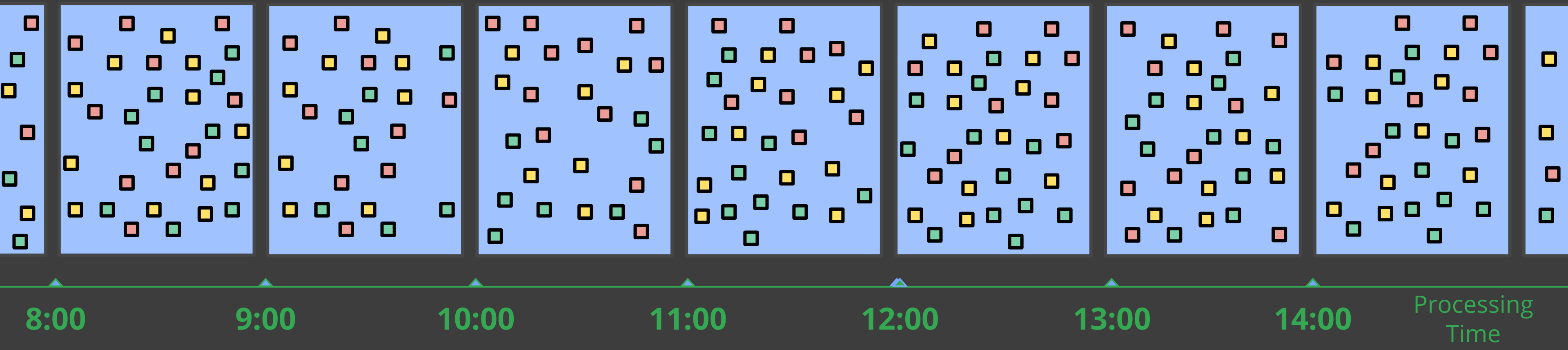


- Framework for building low-latency data-processing applications
- User provides a DAG of computations to be performed
- System manages state and persistent flow of elements

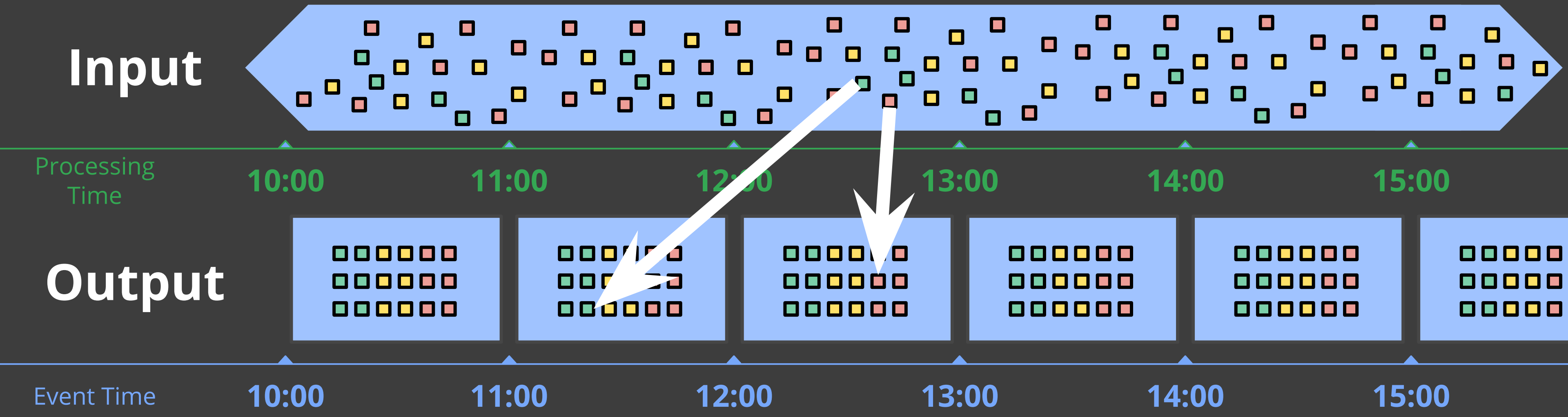
Streaming Patterns: Element-wise transformations



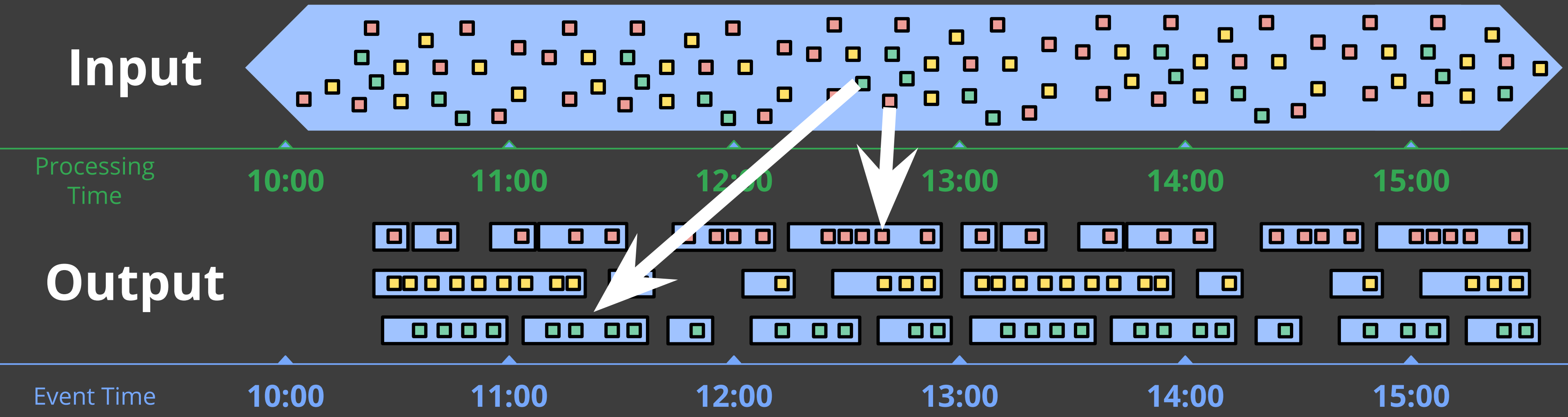
Streaming Patterns: Aggregating Time Based Windows



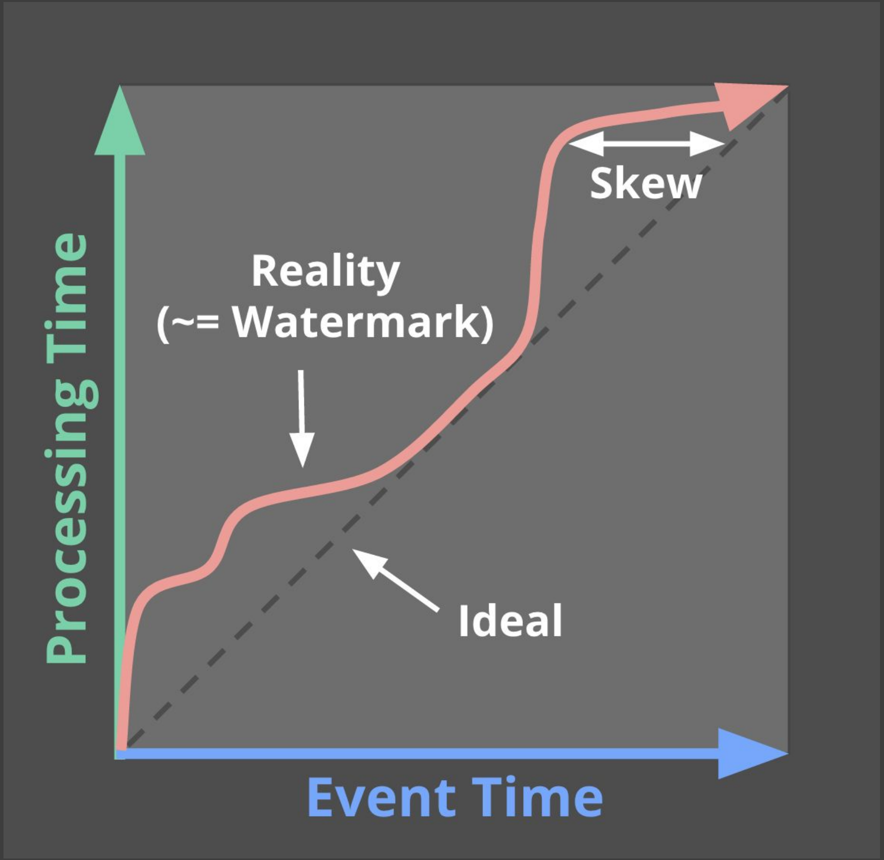
Streaming Patterns: Event-Time Based Windows



Streaming Patterns: Session Windows



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*.

Streaming or Batch?

1+1=2

Completeness

Latency

$$$

Cost

Why not both?

3 What, Where, When, and How

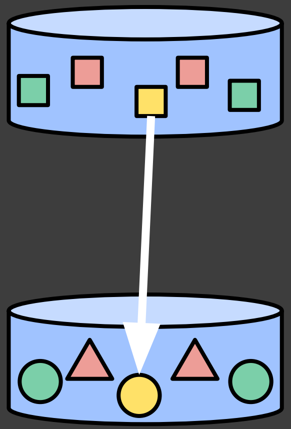
What are you computing?

Where in event time?

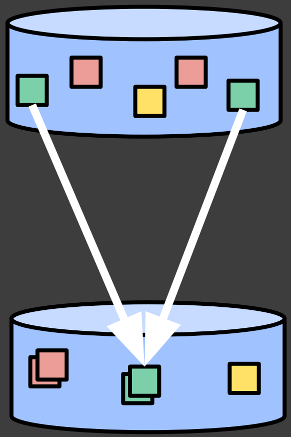
When in processing time?

How do refinements relate?

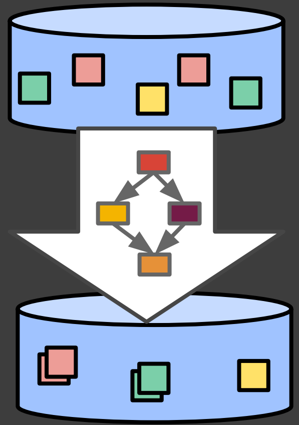
What are you computing?



Element-Wise



Aggregating



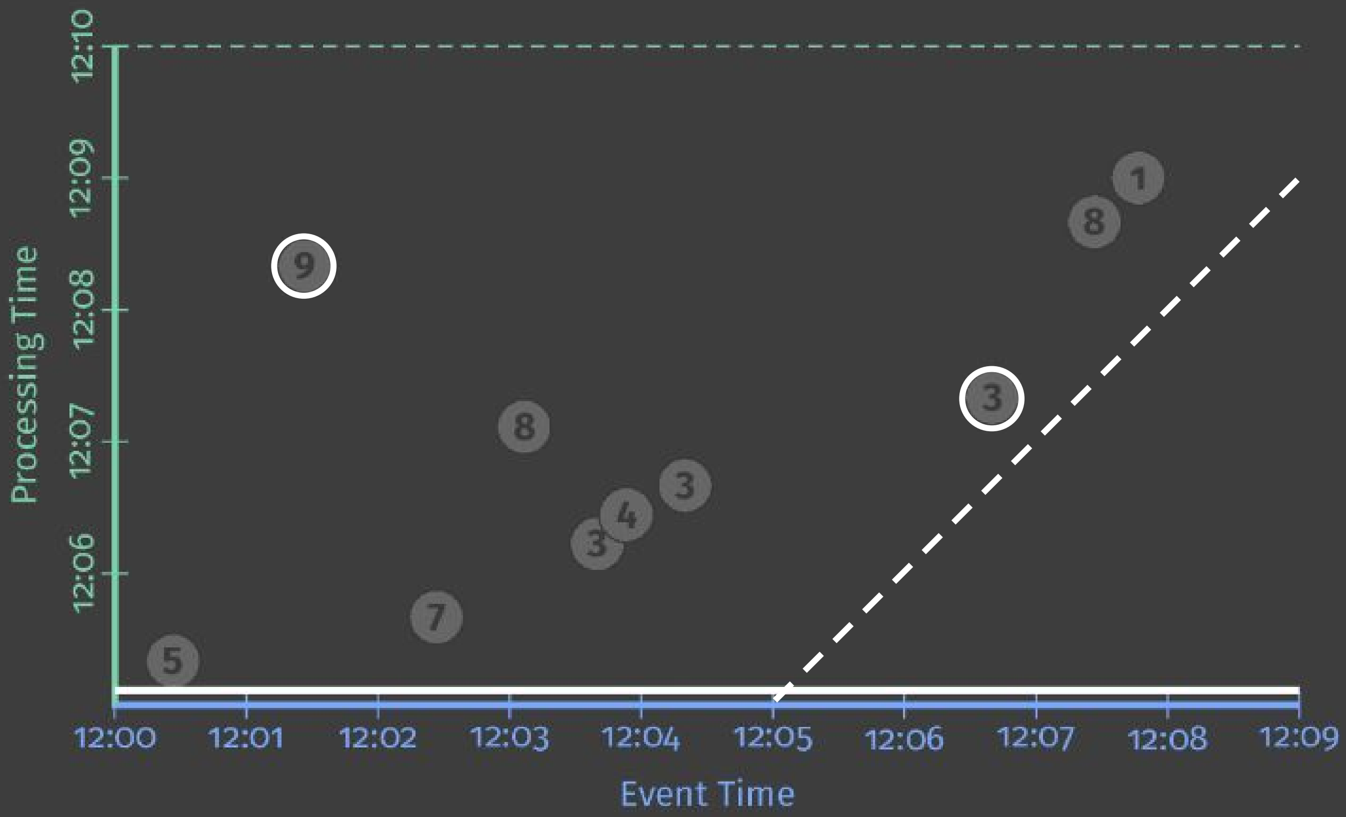
Composite

What: Computing Integer Sums

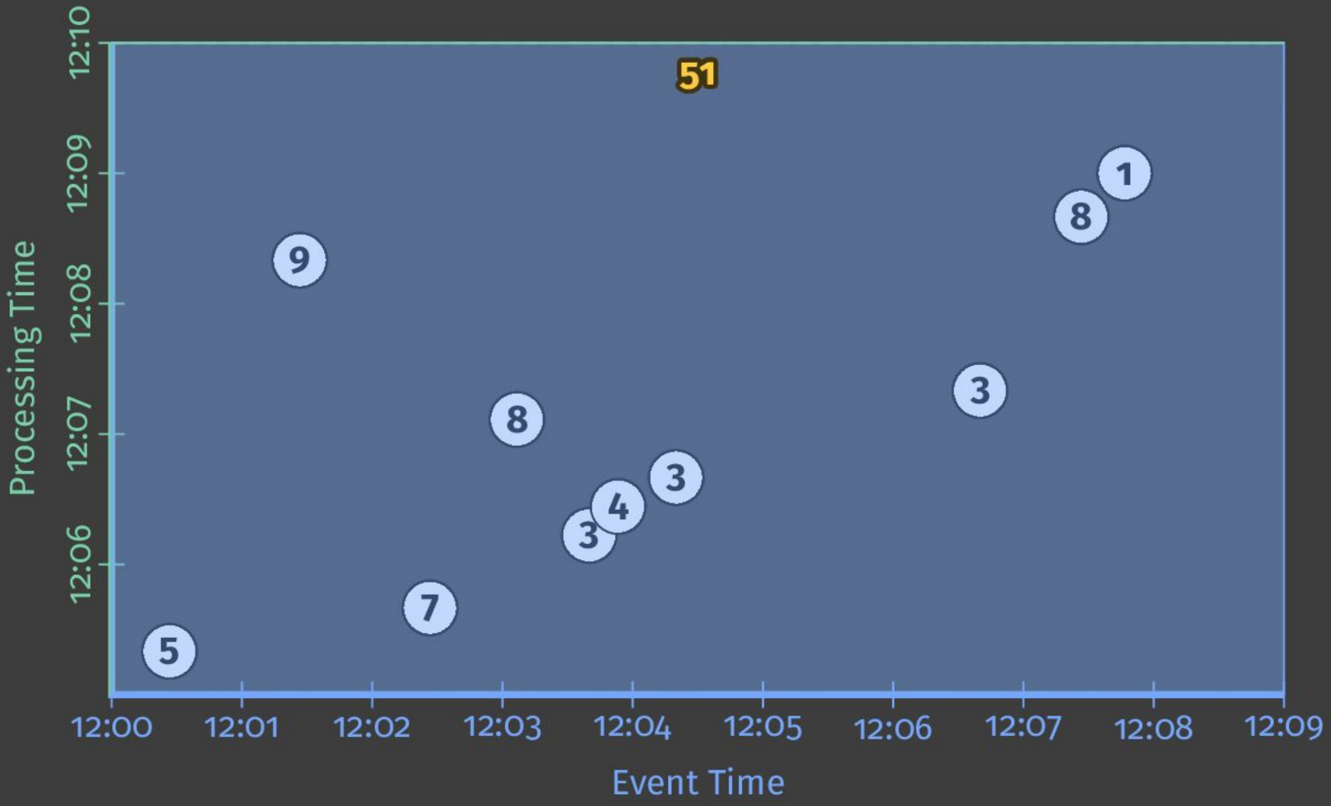
```
// Collection of raw log lines
PCollection<String> raw = IO.read(...);

// 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());
```

*All code snippets are pseudo-java -- details shortened or elided for clarity.

What: Computing Integer Sums

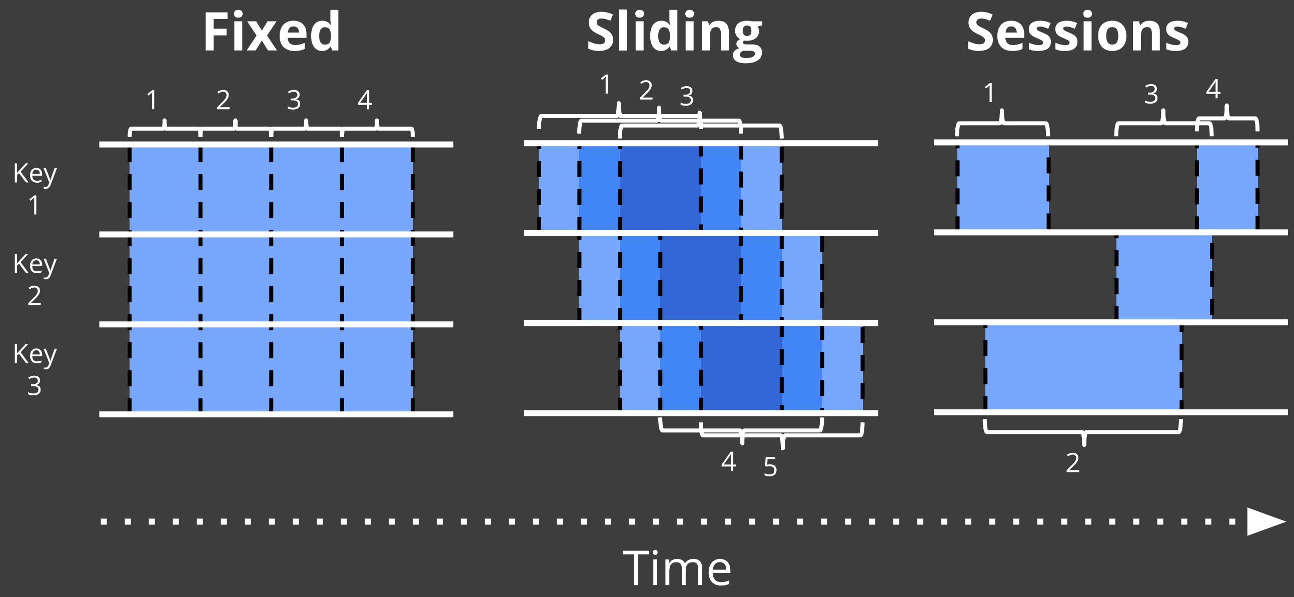


What: Computing Integer Sums



Where in event time?

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

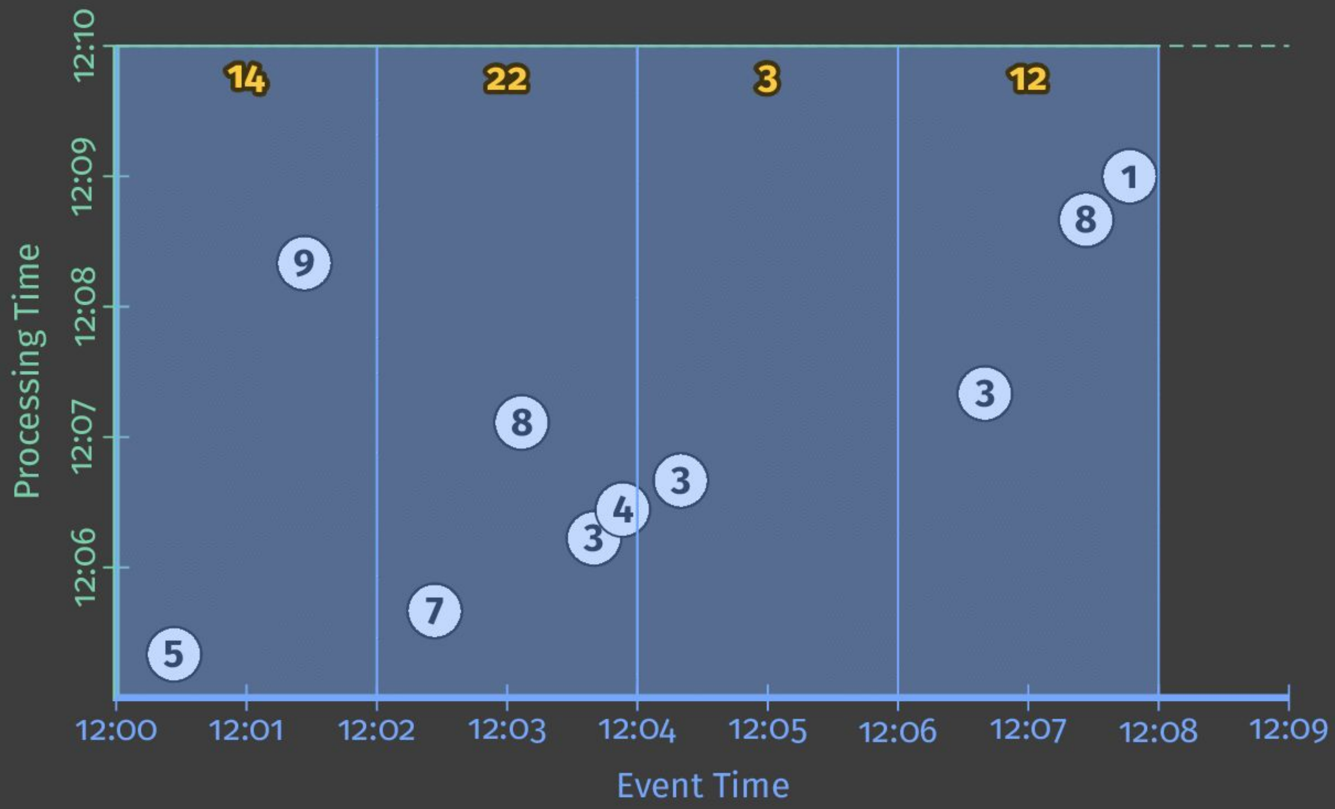


Often required when doing aggregations over unbounded data.

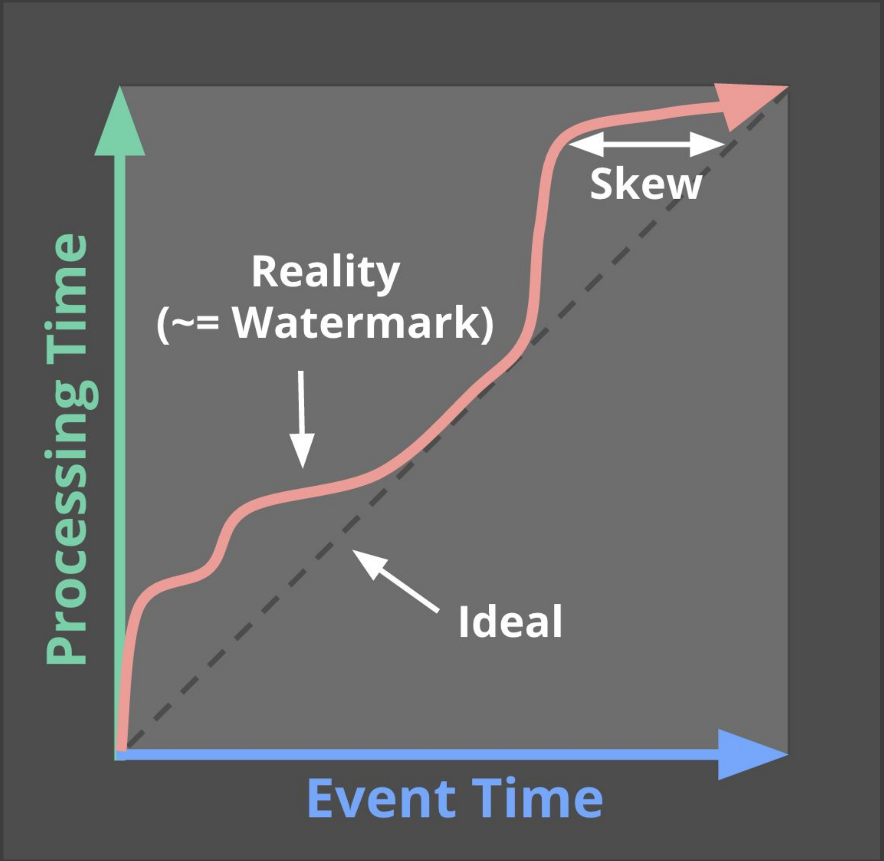
Where: Fixed 2-minute Windows

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

Where: Fixed 2-minute Windows



When in processing time?

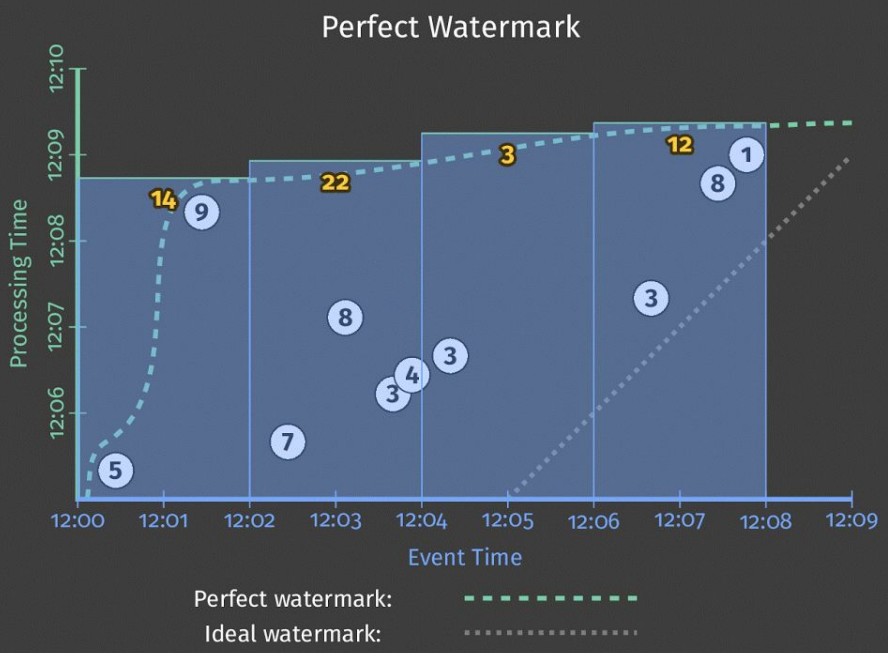


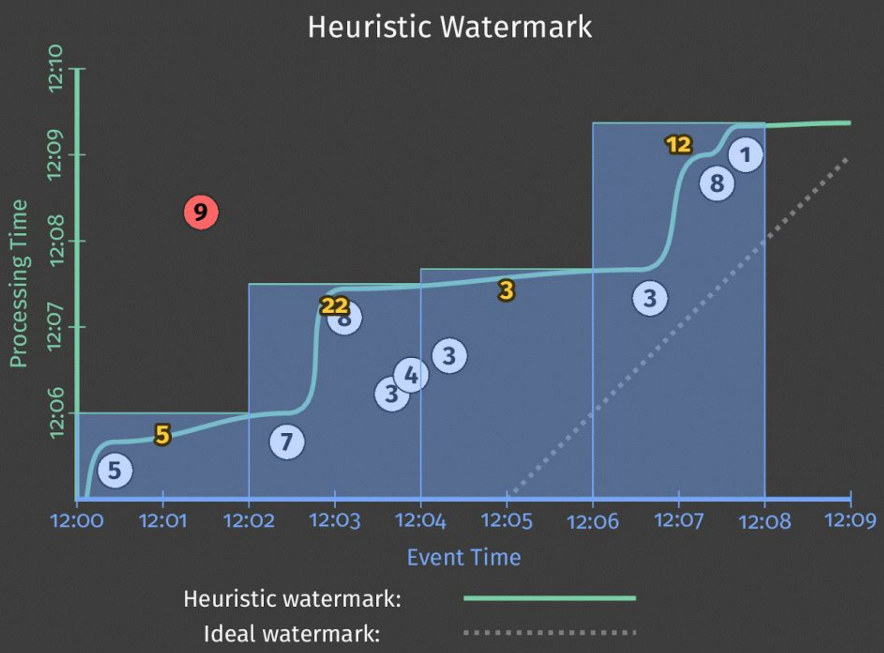
- Triggers control when results are emitted.
- Triggers are often relative to the watermark.

When: Triggering at the Watermark

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

When: Triggering at the Watermark

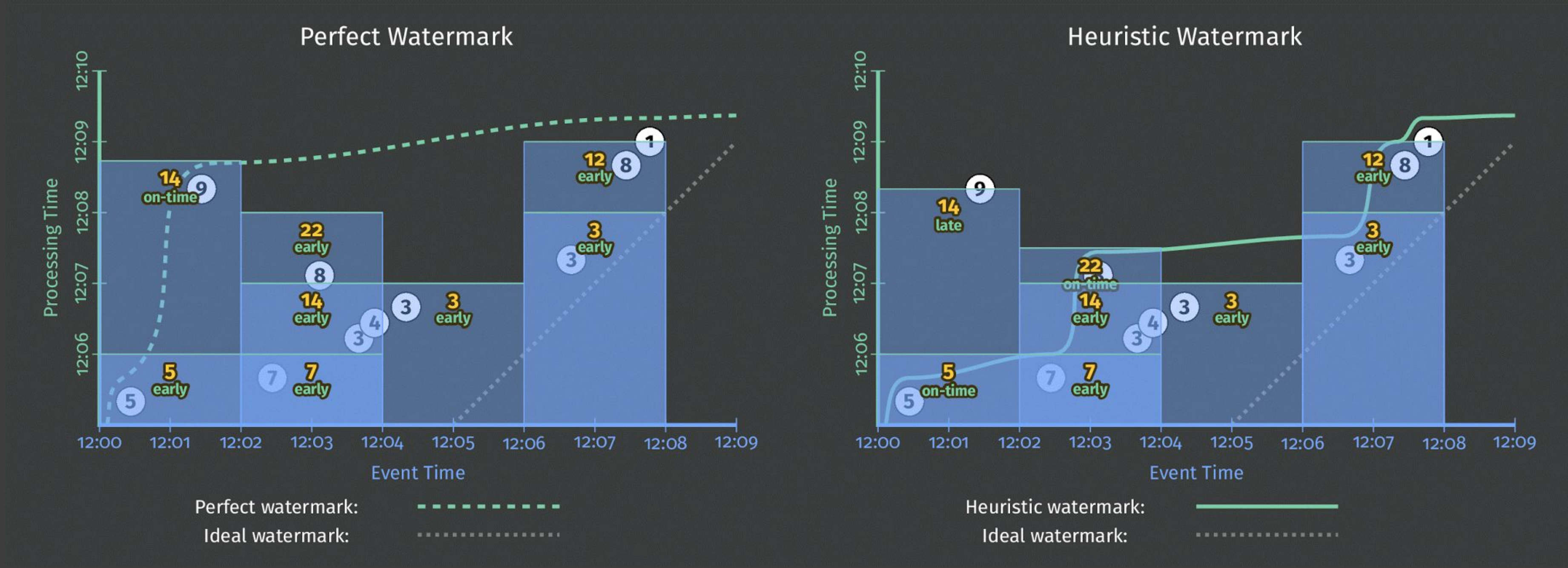




When: Early and Late Firings

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

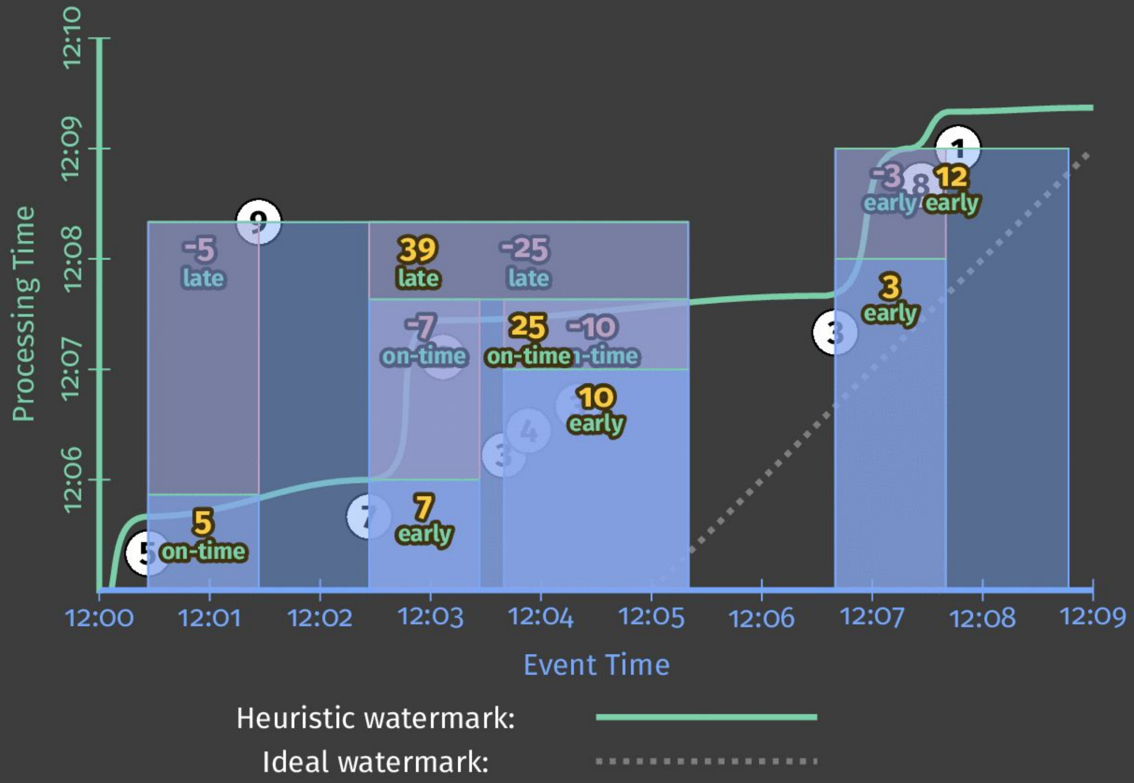
Firing	**Elements**	**Discarding**	**Accumulating**	**Acc. & Retracting***
Speculative	3	3	3	3
Watermark	5, 1	6	9	9, -3
Late	2	2	11	11, -9
Total Observ	***11***	***11***	***23***	***11***

*Accumulating & Retracting not yet implemented in Apache Beam.

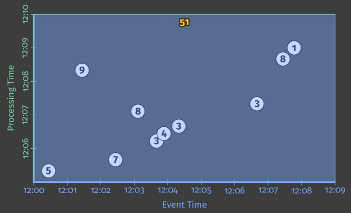
How: Add Newest, Remove Previous

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

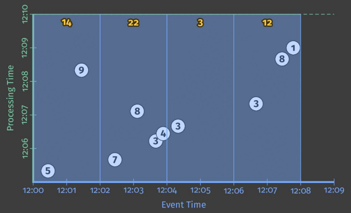
How: Add Newest, Remove Previous



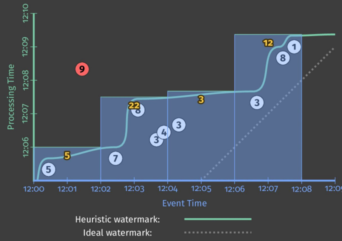
Customizing What When Where How



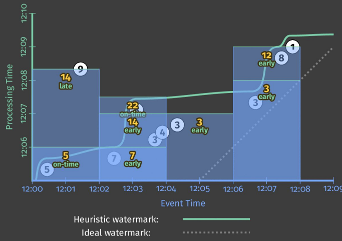
1.Classic Batch



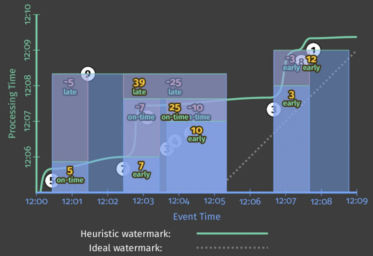
2. Batch with Fixed Windows



3. Streaming



4. Streaming with Speculative + Late Data



5. Streaming With Retractions

4 Apache Beam (incubating)

The Dataflow Model & Cloud Dataflow

Dataflow Model & SDKs

a unified model for
batch and stream processing

Google Cloud Dataflow

no-ops, fully managed service

The *Beam* Model & Cloud Dataflow

Apache Beam

a unified model for
batch and stream processing
supporting multiple runtimes

Google Cloud Dataflow

a great place to run Beam

What is Part of Apache Beam?

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

2. SDKs for writing Beam pipelines -- starting with Java

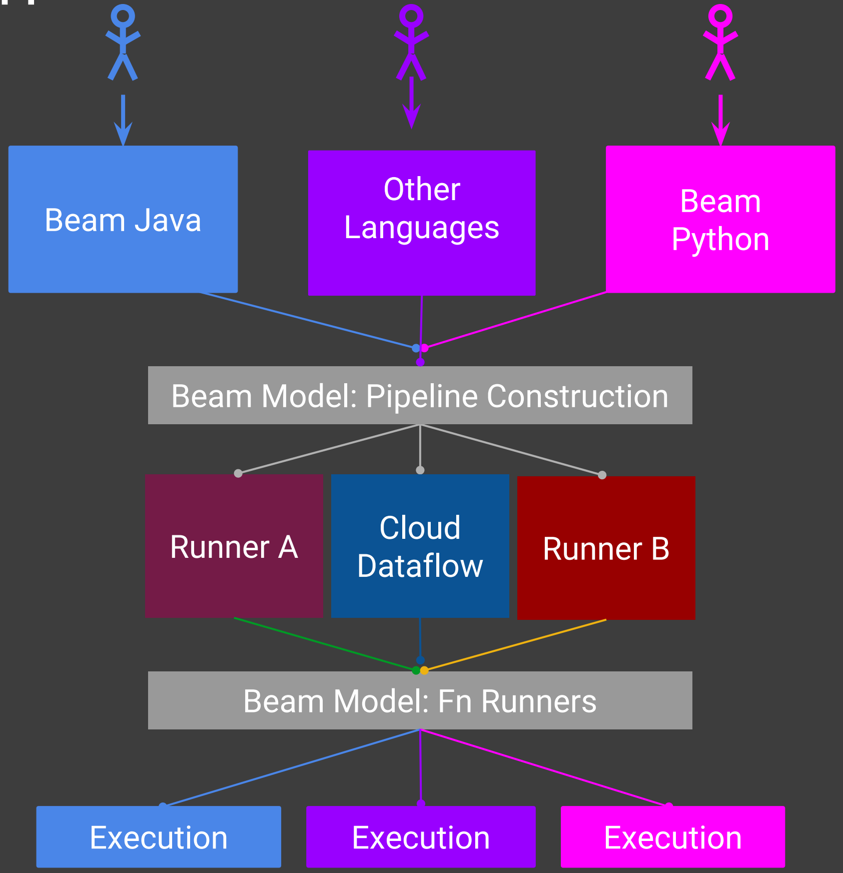
3. Runners for Existing Distributed Processing Backends
   - Apache Flink (thanks to data Artisans)
   - Apache Spark (thanks to Cloudera)
   - Google Cloud Dataflow (fully managed service)
   - Local (in-process) runner for testing

Apache Beam Technical Vision

1. **End users:** who want to write pipelines 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	✕	✕	✕	✕

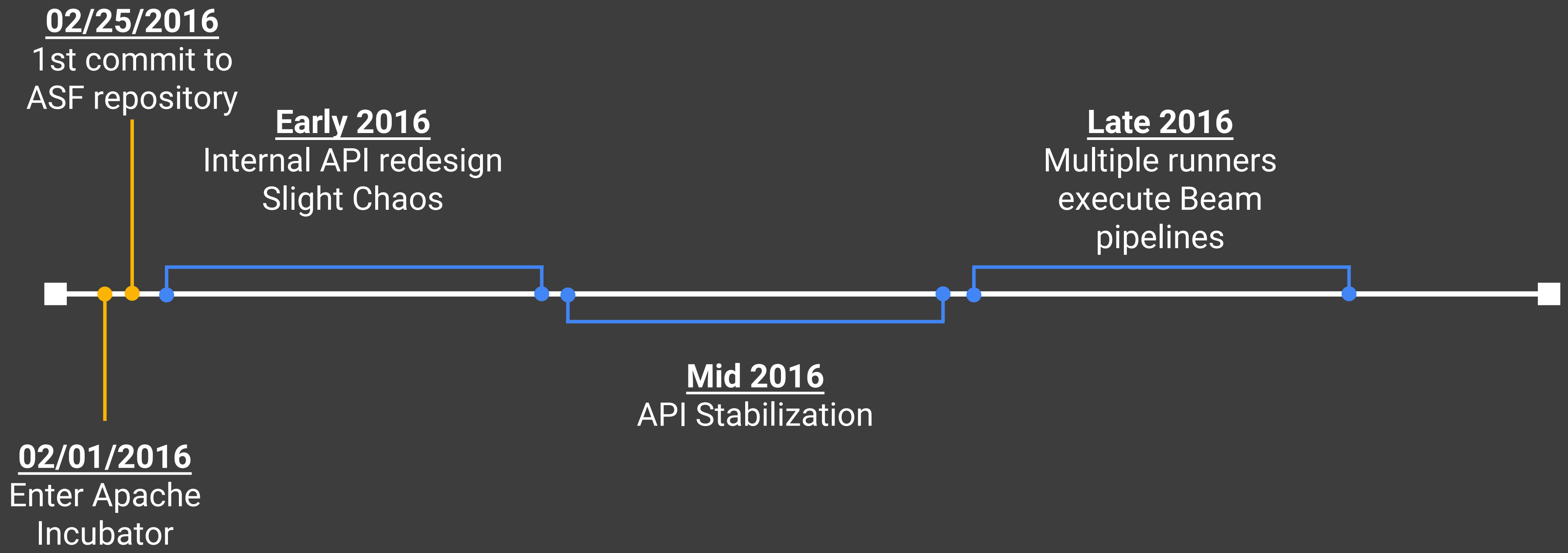
http://beam.incubator.apache.org/capability-matrix/

Growing the Beam Community

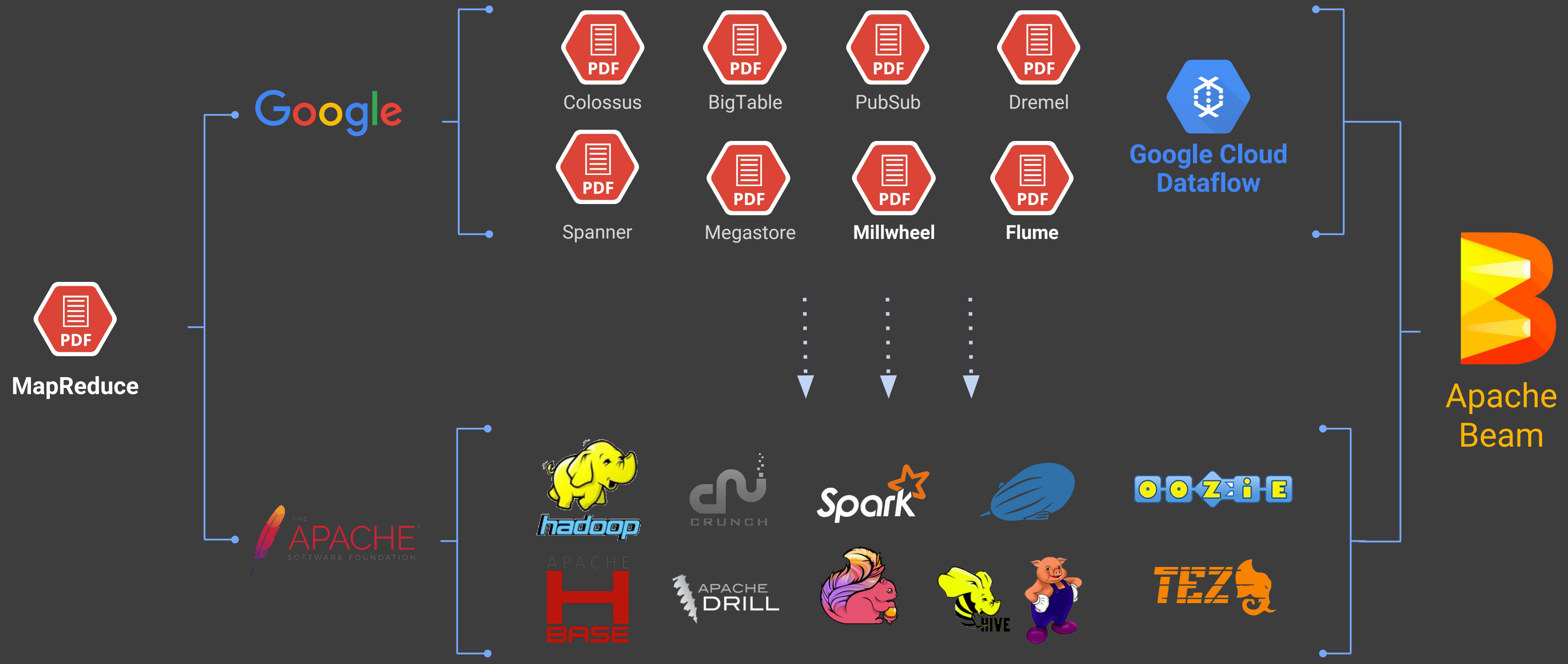
Collaborate - Beam is becoming a community-driven effort with participation from many organizations and contributors

Grow - We want to grow the Beam ecosystem and community with active, open involvement so Beam is a part of the larger OSS ecosystem

Apache Beam Roadmap



The Evolution of Apache Beam



Learn More!

Apache Beam (incubating)
http://beam.incubator.apache.org

The World Beyond Batch 101 & 102
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101
https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102

Join the Beam mailing lists!
user-subscribe@beam.incubator.apache.org
dev-subscribe@beam.incubator.apache.org

Follow @ApacheBeam on Twitter

Thank you!