Google Cloud Dataflow

A Unified Model for Batch and Streaming Data Processing

Jelena Pjesivac-Grbovic

STREAM 2015

Google Cloud Platform

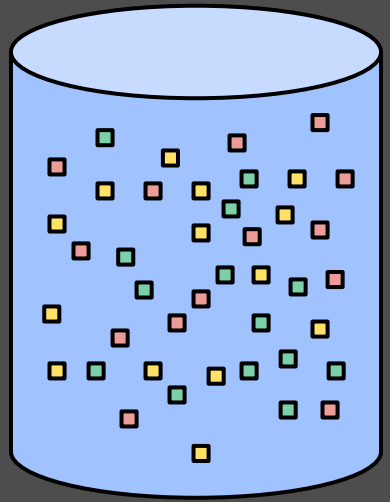
Agenda

1. Data Shapes
2. Data Processing Tradeoffs
3. Google's Data Processing Story
4. Google Cloud Dataflow

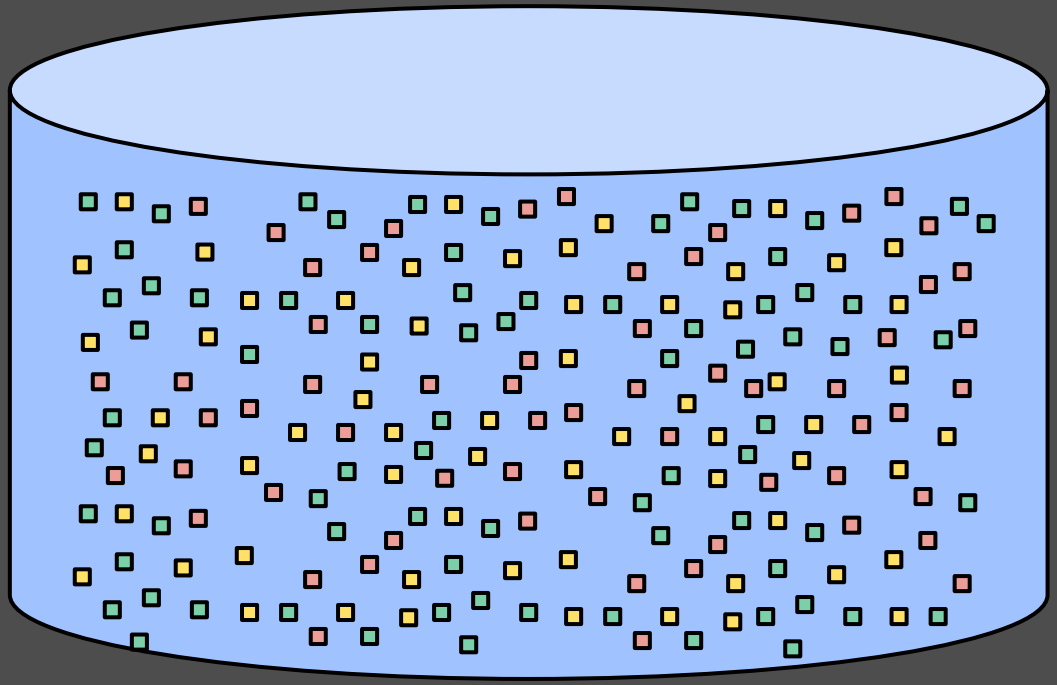
https://commons.wikimedia.org/wiki/File:Globe_centered_in_the_Atlantic_Ocean_(green_and_grey_globe_scheme).svg

Data Shapes

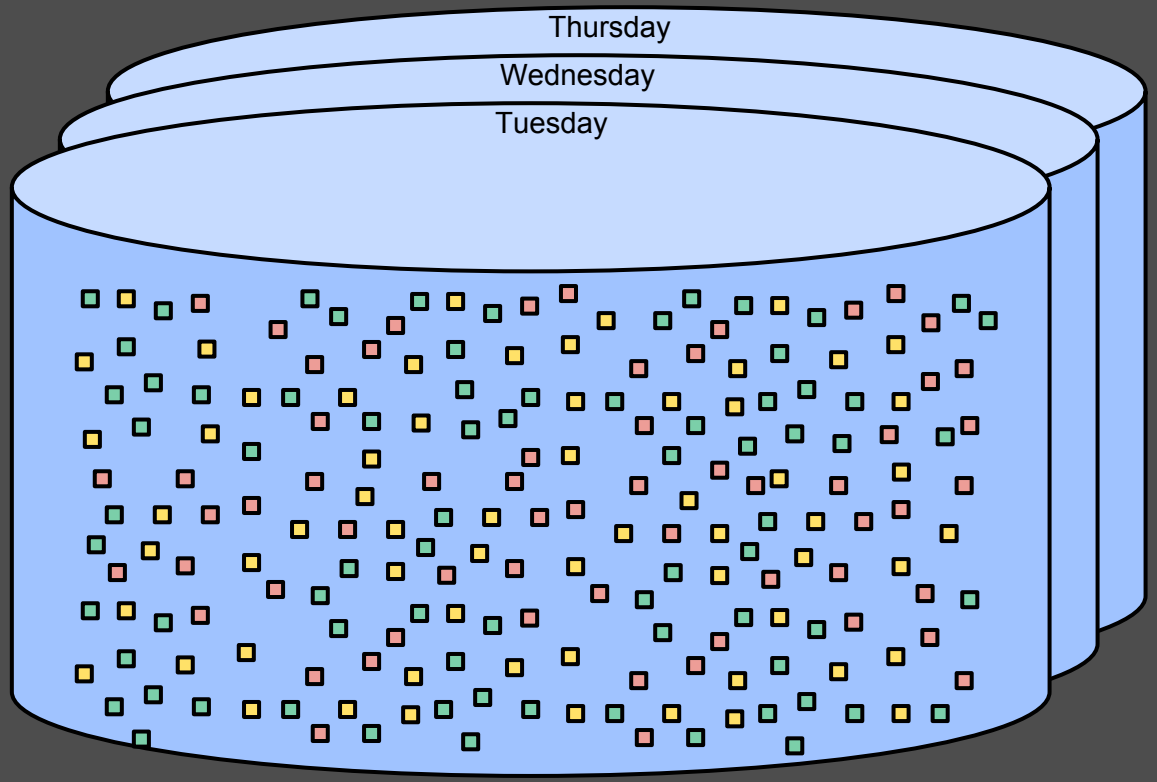
Data...



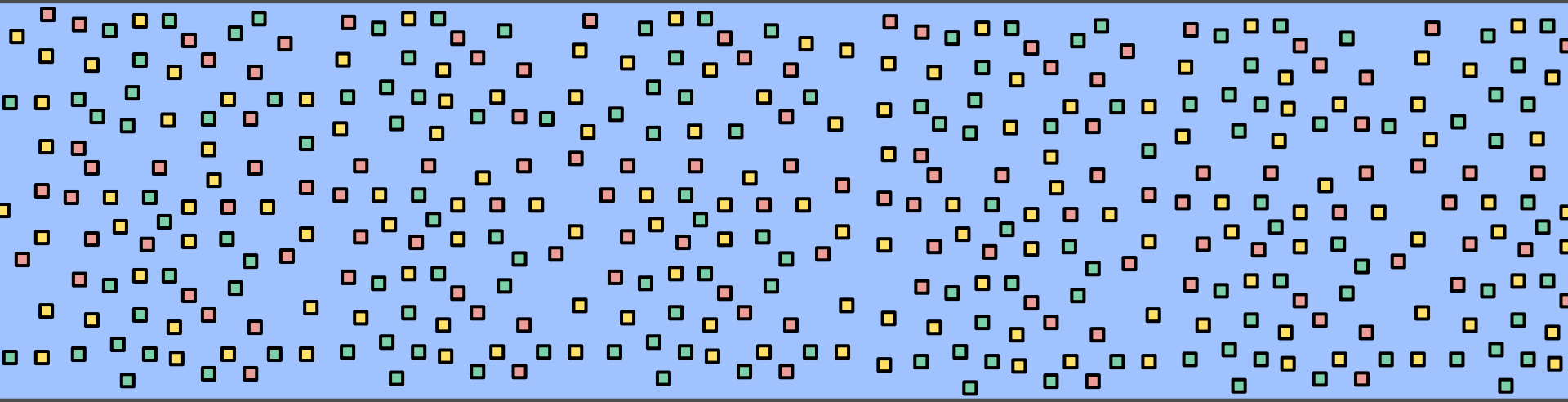
...can be big...



...really, really big...

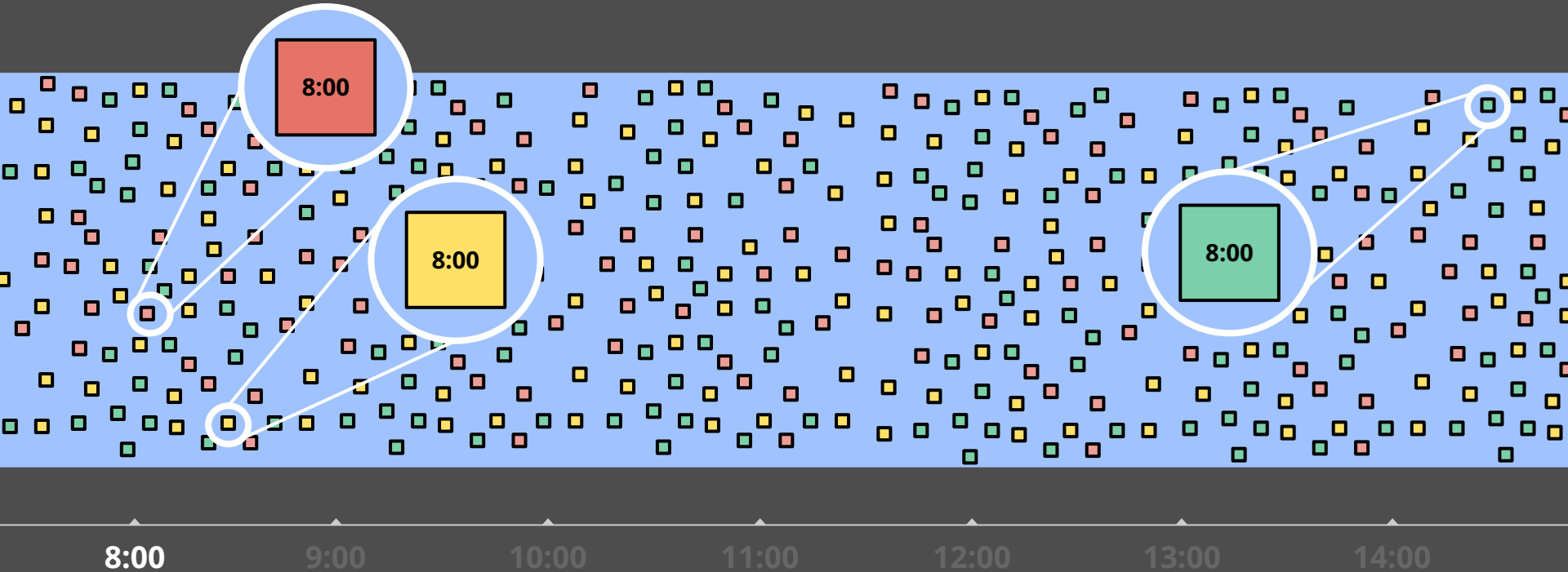


...maybe even infinitely big...



8:00 1:00 9:00 2:00 10:00 3:00 11:00 4:00 12:00 5:00 13:00 6:00 14:00 7:00

... with unknown delays.



Data Processing Tradeoffs

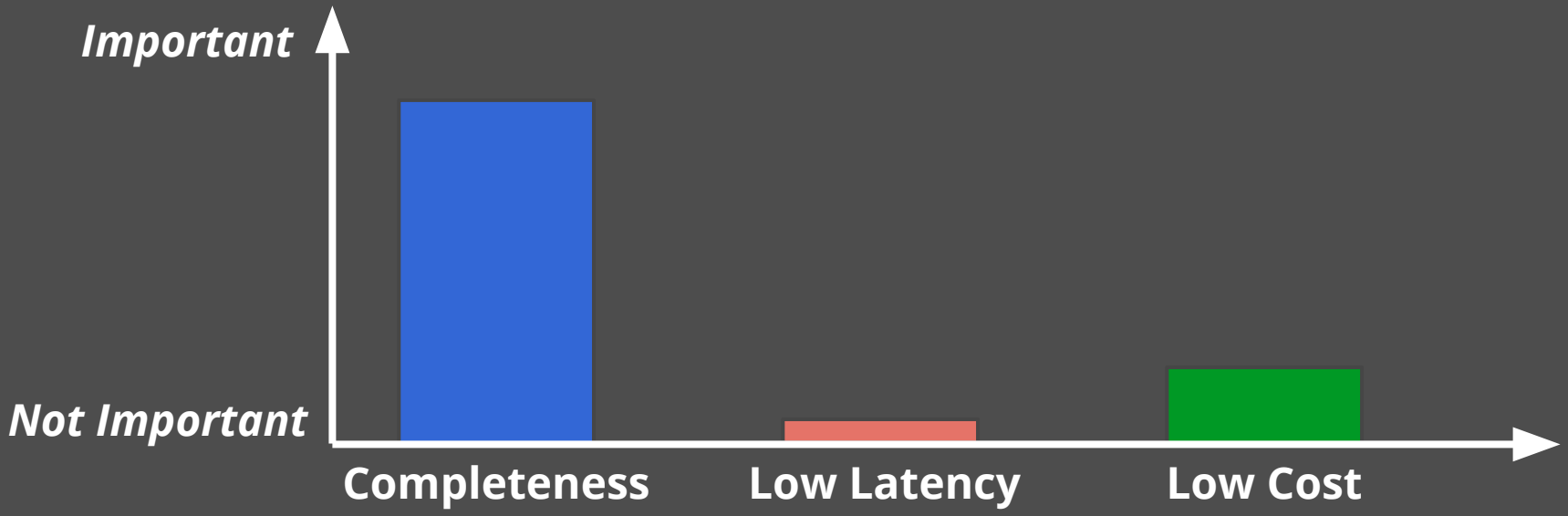
Data Processing Tradeoffs

1+1=2

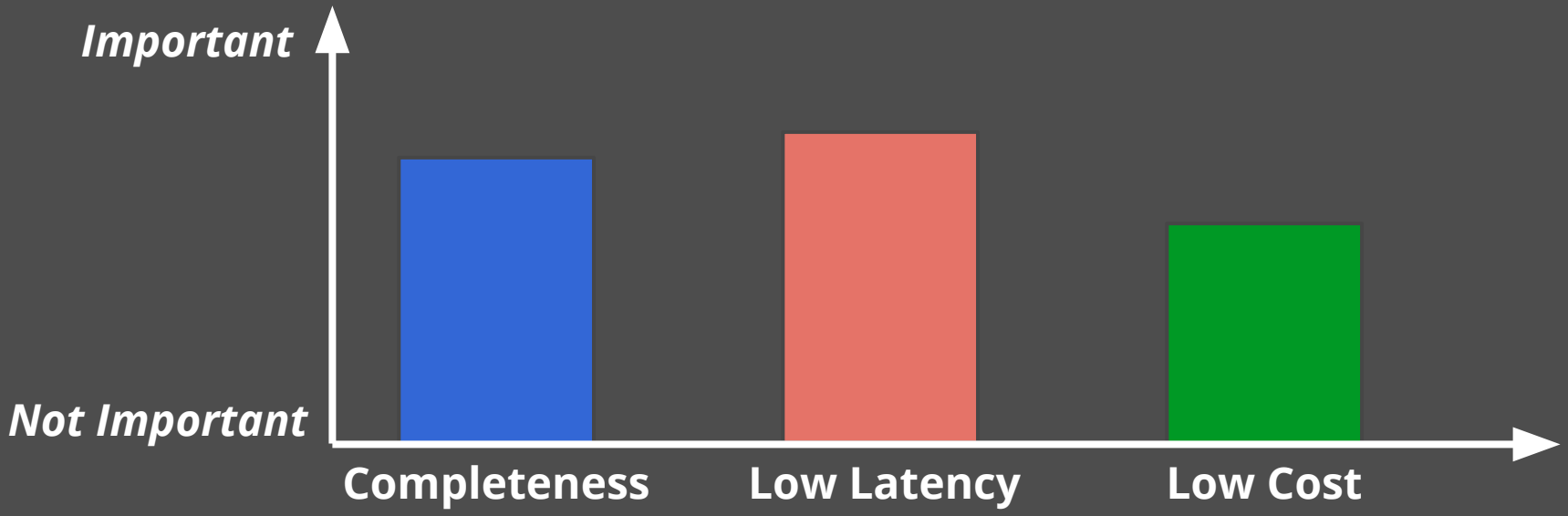
$$$

Completeness **Latency** **Cost**

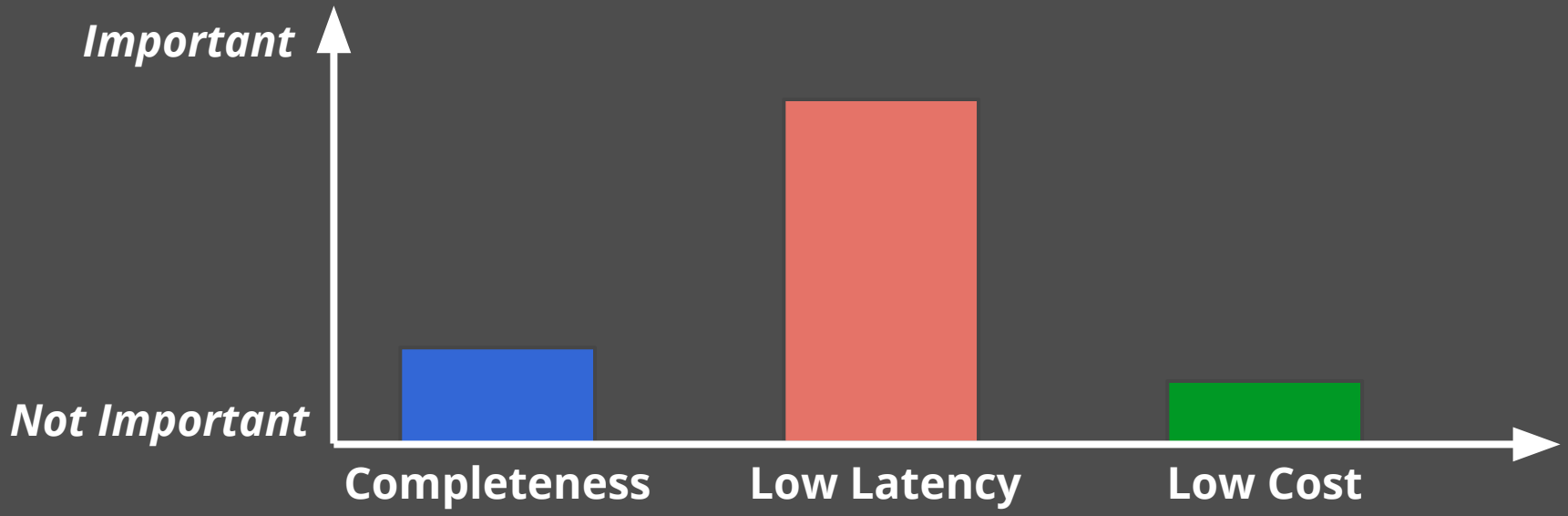
Requirements: Billing Pipeline



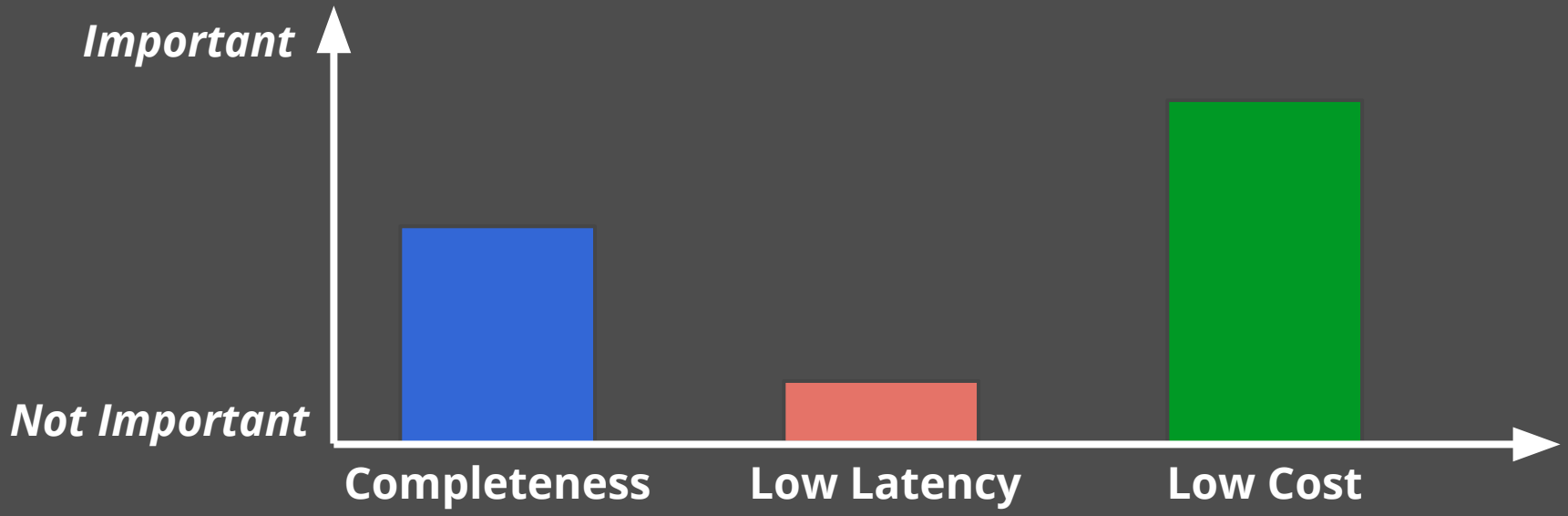
Requirements: Live Cost Estimate Pipeline



Requirements: Abuse Detection Pipeline

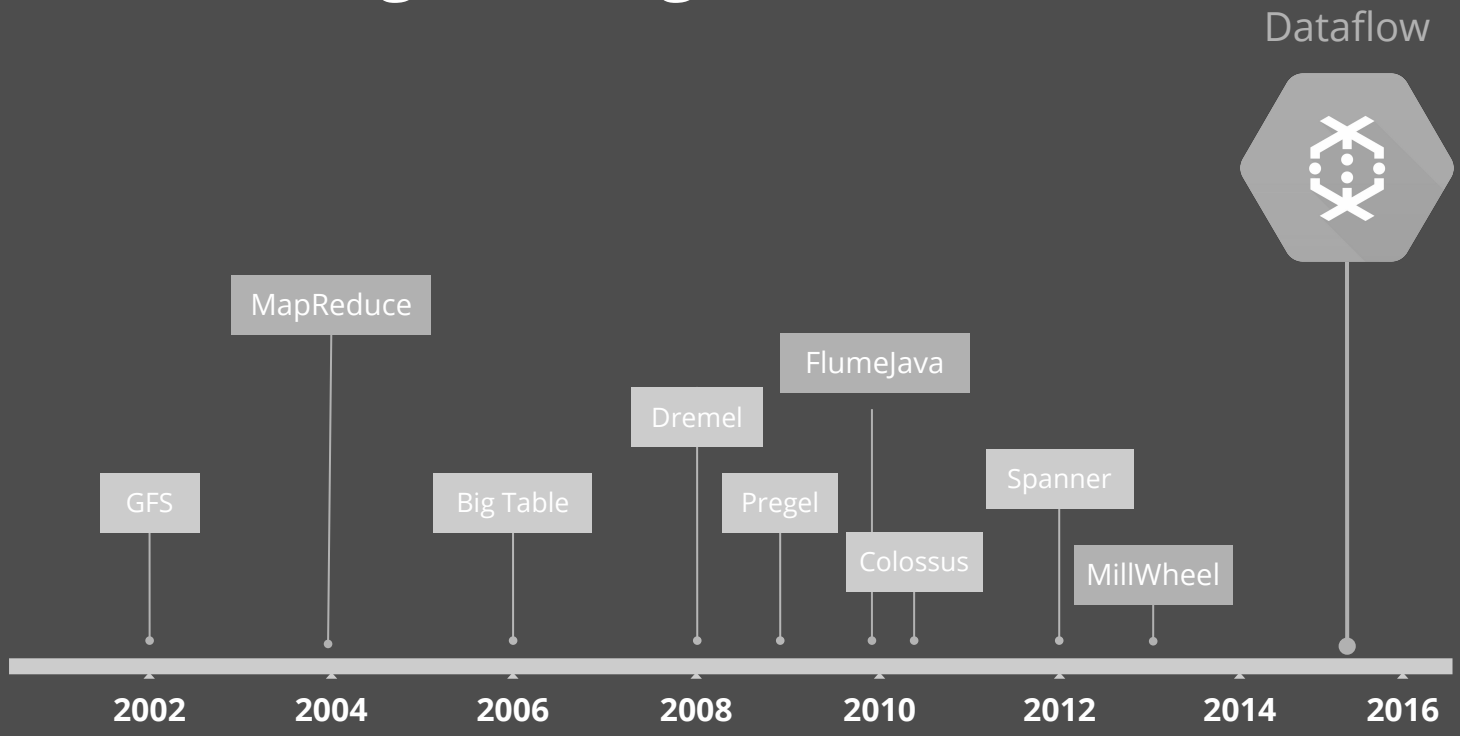


Requirements: Abuse Detection Backfill Pipeline

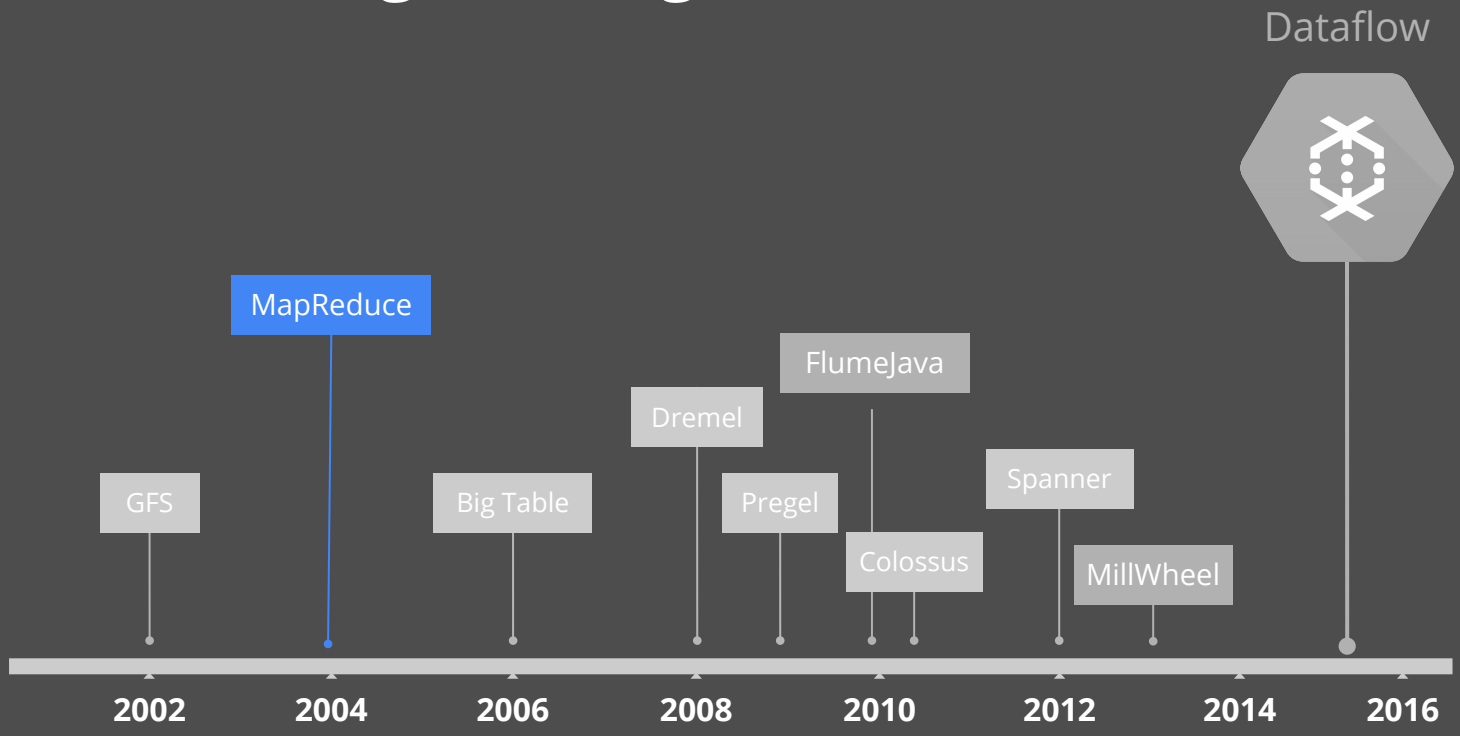


Google's Data Processing Story

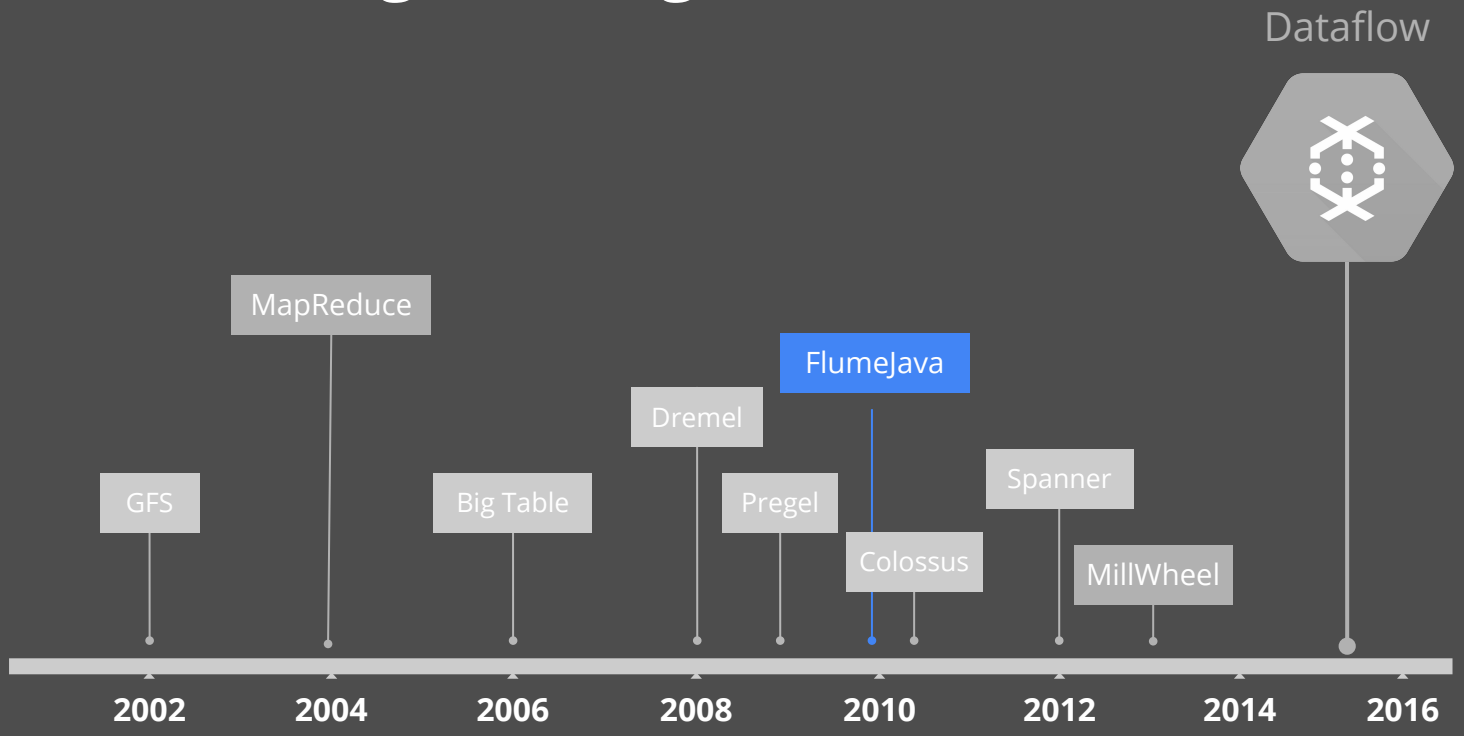
Data Processing @ Google



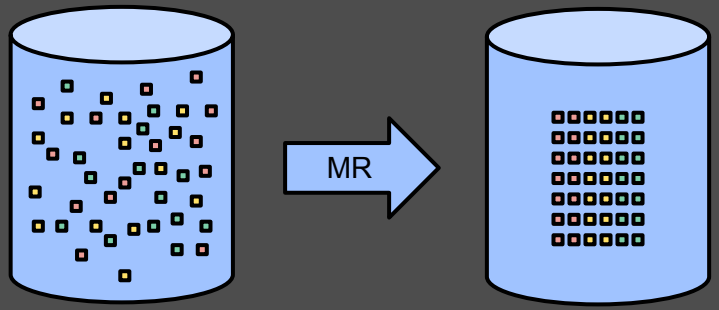
Data Processing @ Google



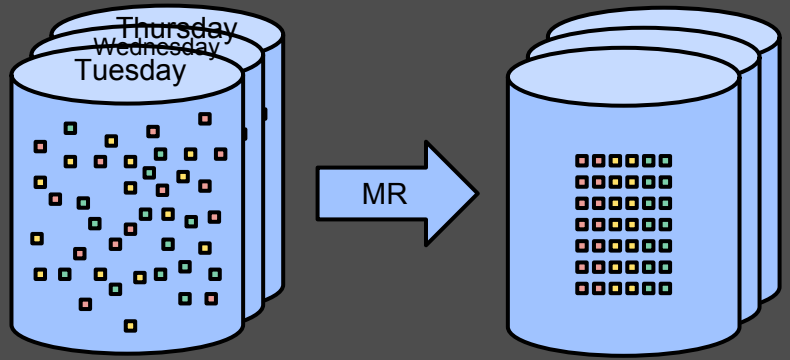
Data Processing @ Google



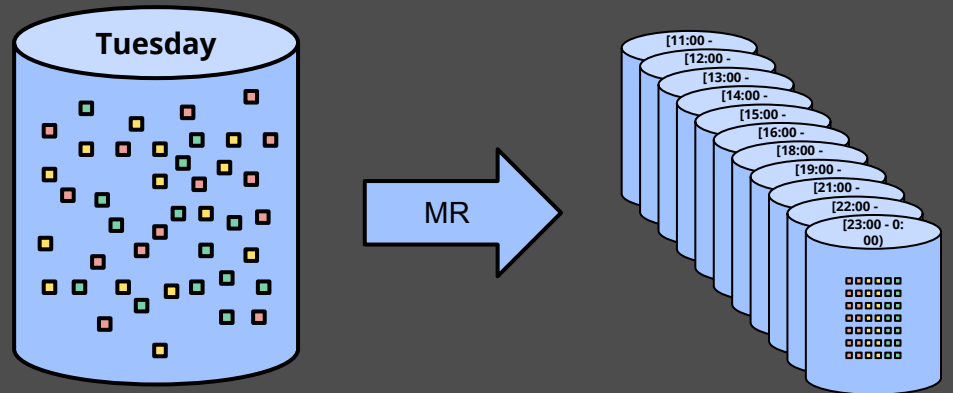
Batch Patterns: Transform, Filter, and Aggregate



Create structured data

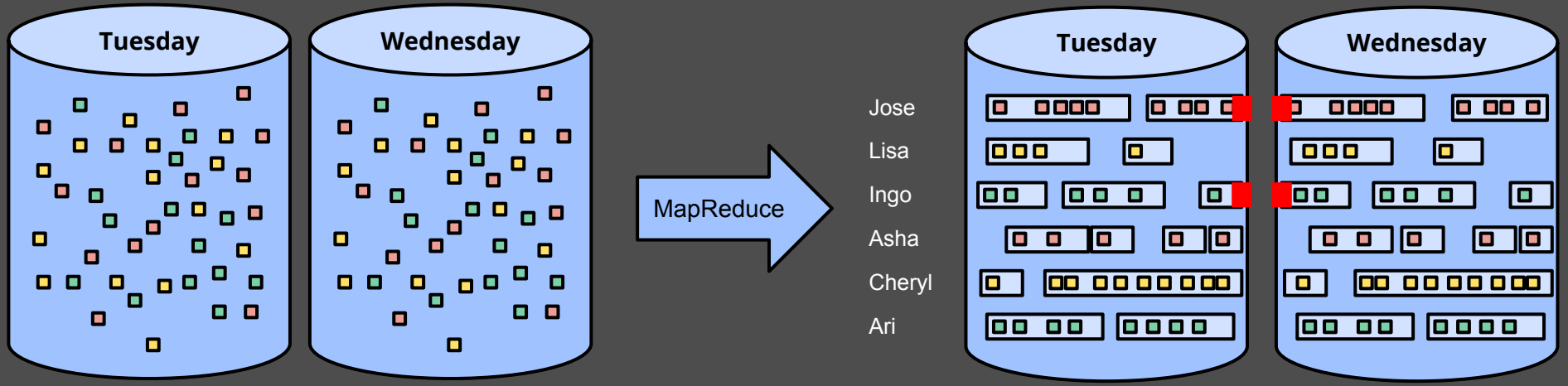


Create structured data, repeatedly

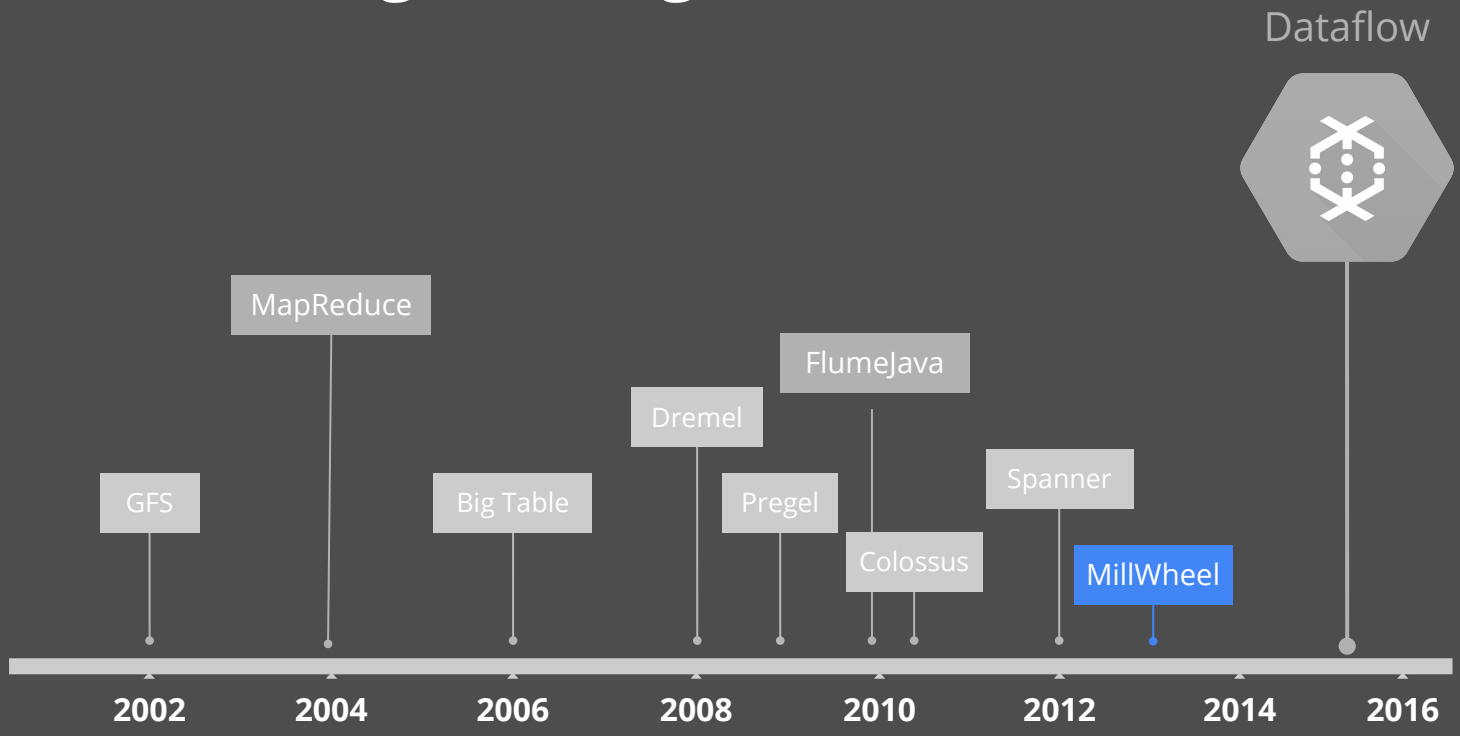


Generate time-based windows

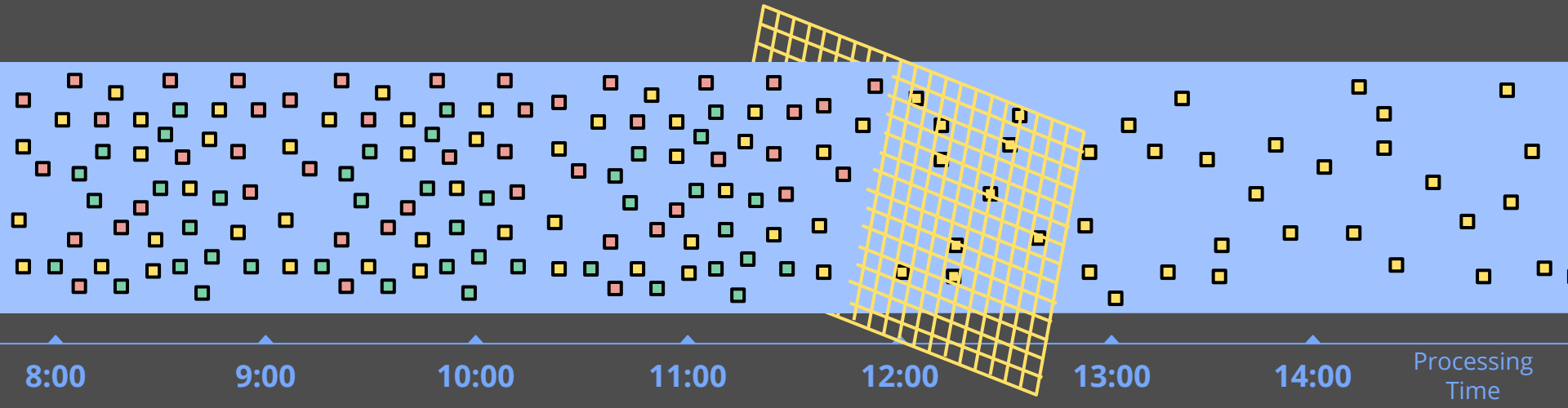
Batch Patterns: Sessions



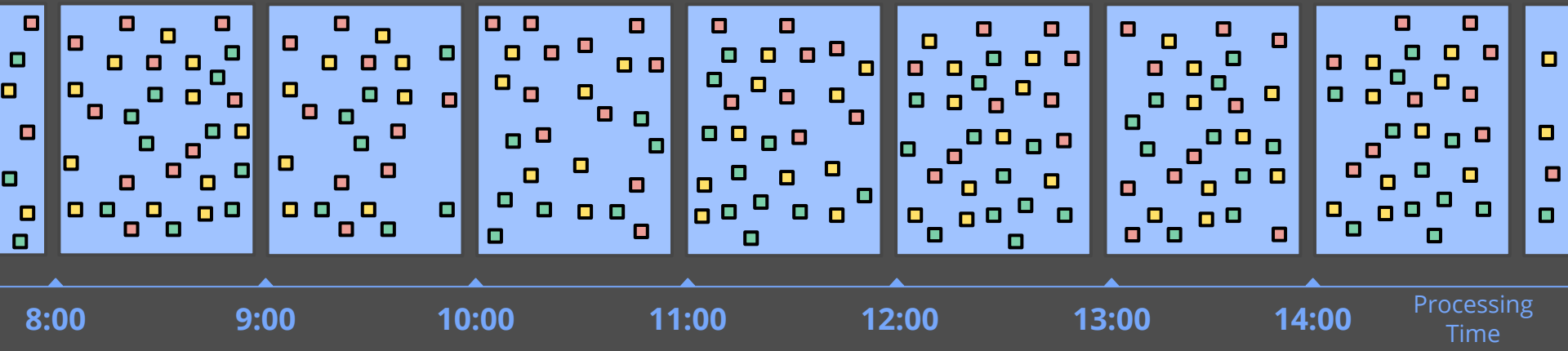
Data Processing @ Google



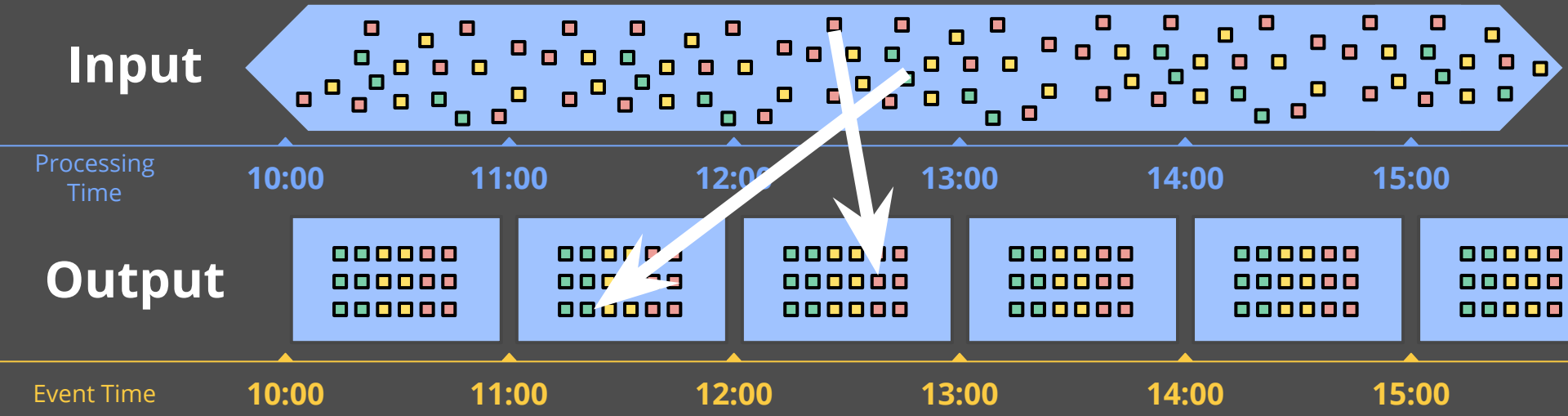
Streaming Patterns: Element-wise transformations



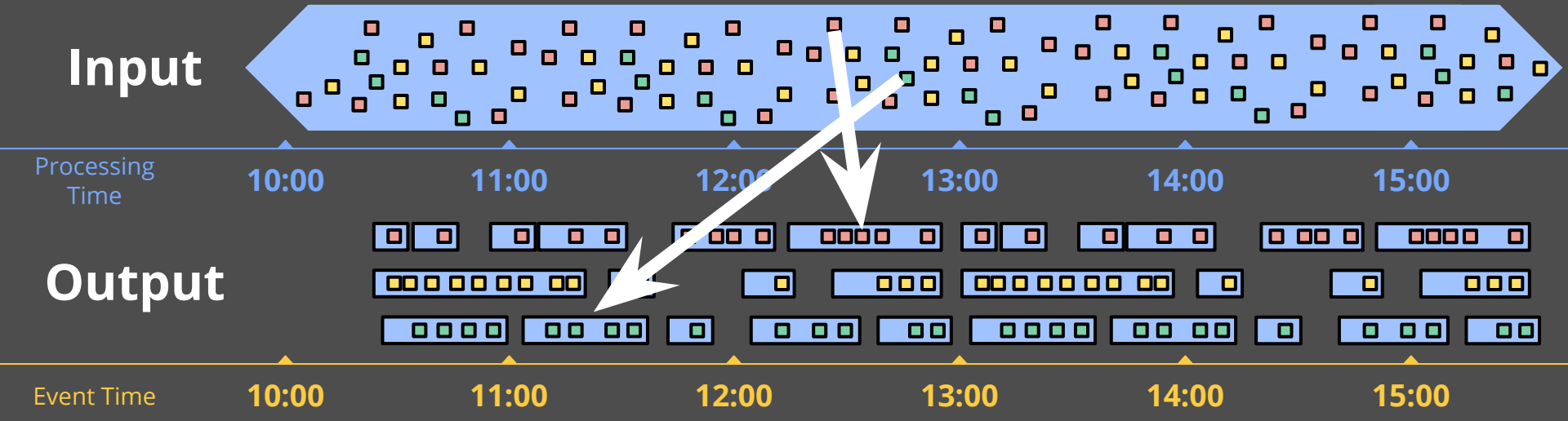
Streaming Patterns: Aggregating Time Based Windows



Streaming Patterns: Event Time Based Windows



Streaming Patterns: Session Windows



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?

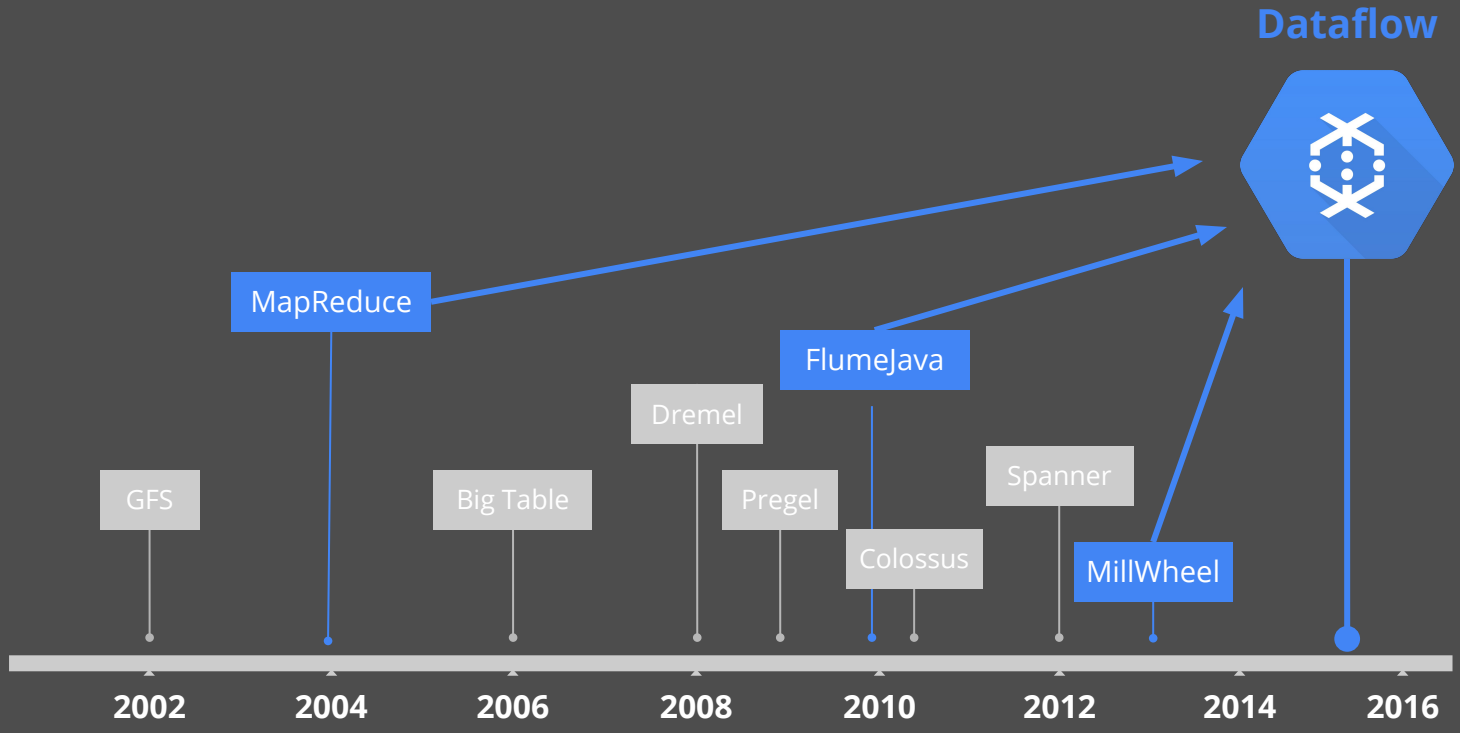
1+1=2

$$$

Completeness **Latency** **Cost**

Why not both?

Data Processing @ Google



Google Cloud Dataflow

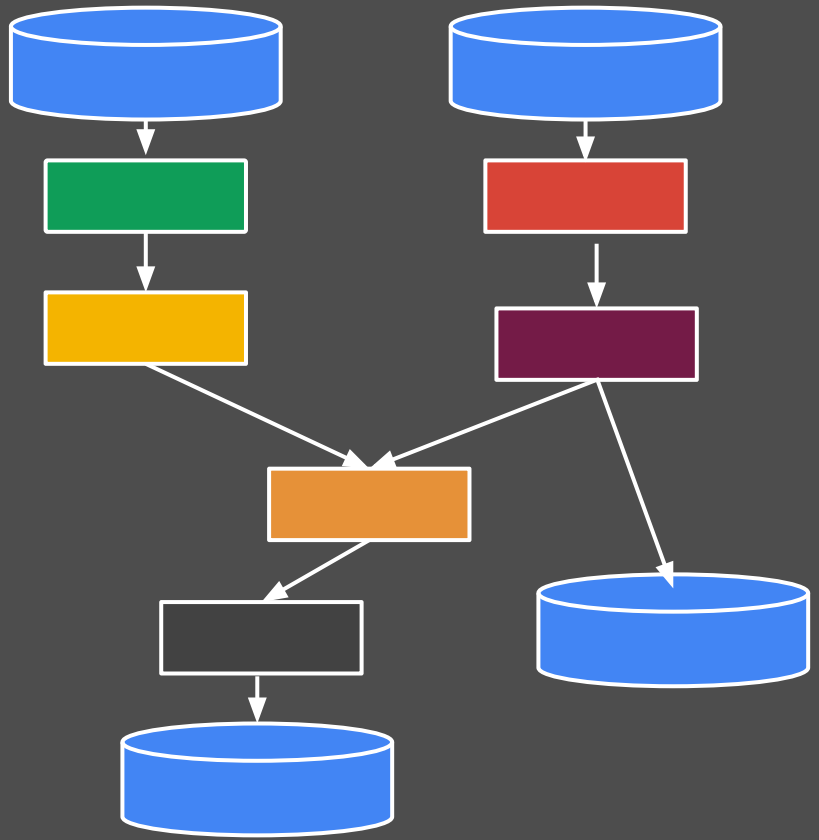
What are you computing?

How does the event time affect computation?

How does the processing time affect latency?

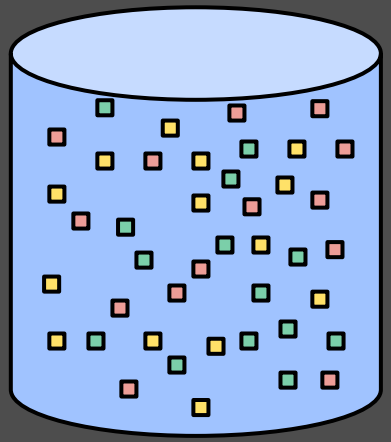
How do results get refined?

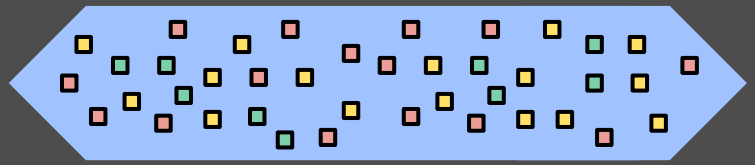
What are you computing?



- A Pipeline represents a graph of data processing transformations
- PCollections flow through the pipeline
- Optimized and executed as a unit for efficiency

Dataflow Model: Data

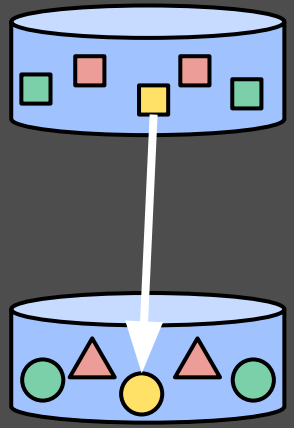




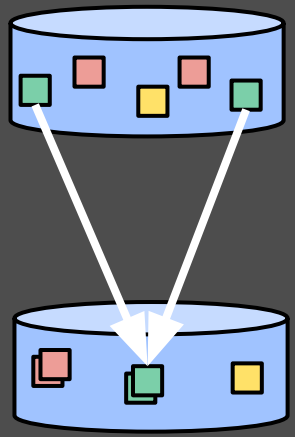
- A PCollection<T> is a collection of data of type T
- Maybe be bounded or unbounded in size
- Each element has an implicit timestamp
- Initially created from backing data stores

Dataflow Model: Transformations

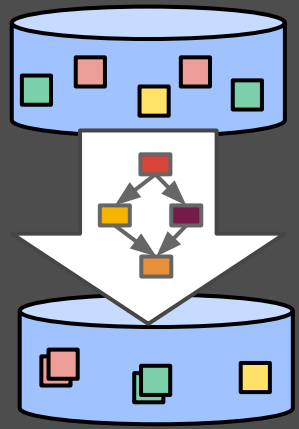
PTransforms transform PCollections into other PCollections.



Element-Wise



Aggregating

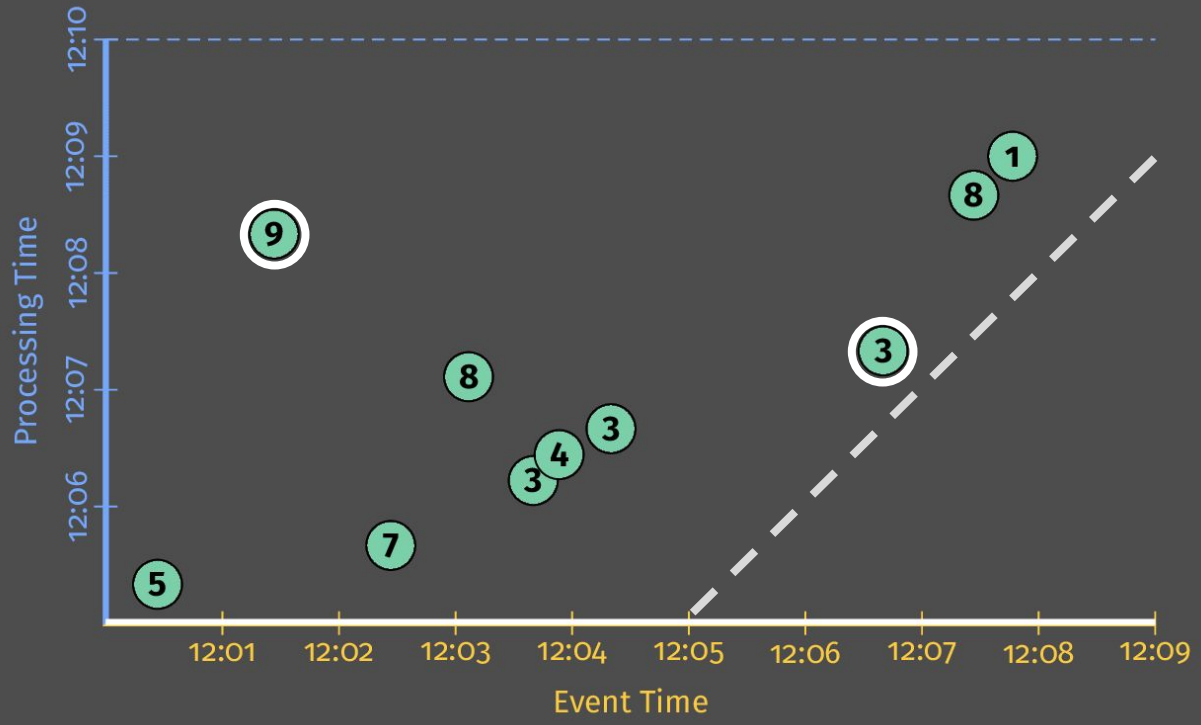


Composite

Example: Computing Integer Sums

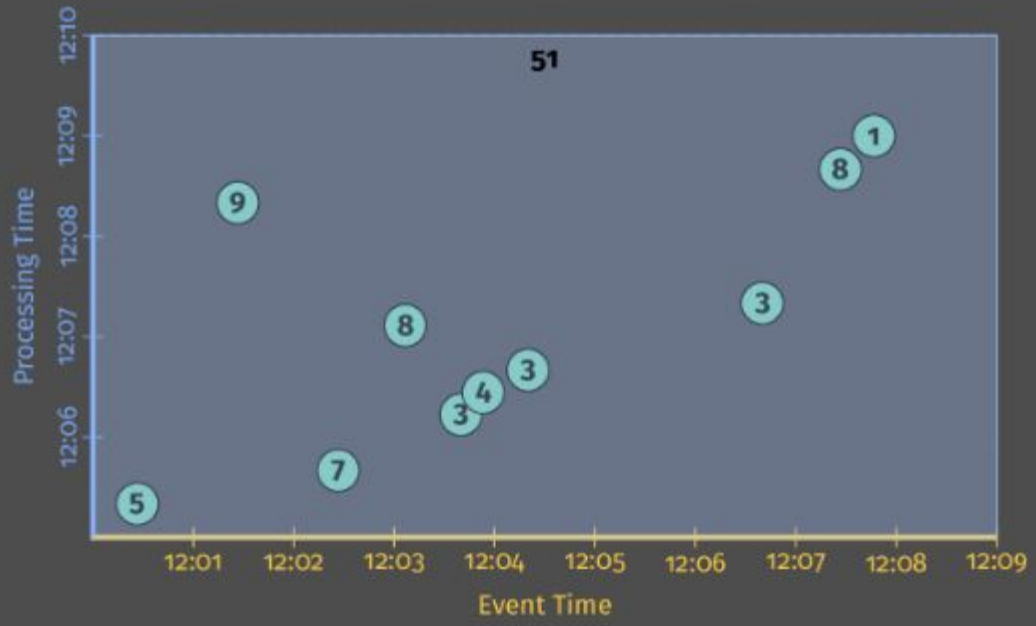
```
// Collection of raw log lines
PCollection<String> raw = ...;

// 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>> output = input
  .apply(Sum.integersPerKey());
```

Example: Computing Integer Sums

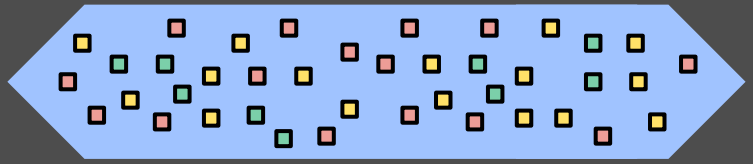


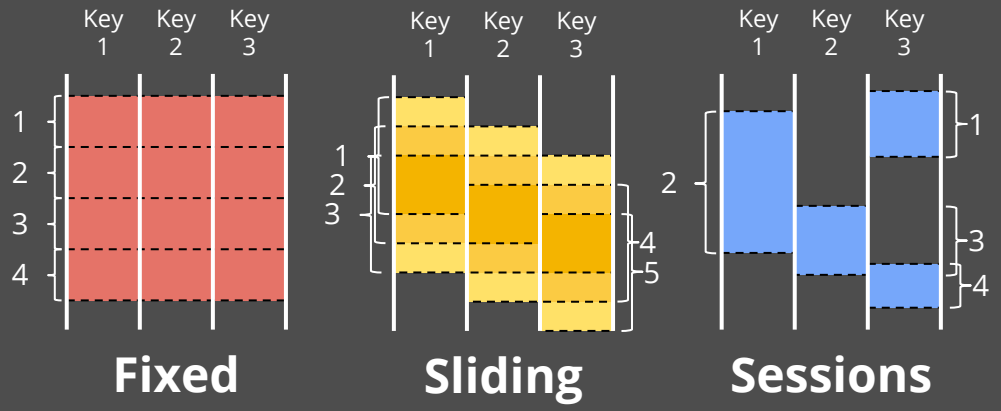
What Where When How

Example: Computing Integer Sums



Aggregating According to Event Time



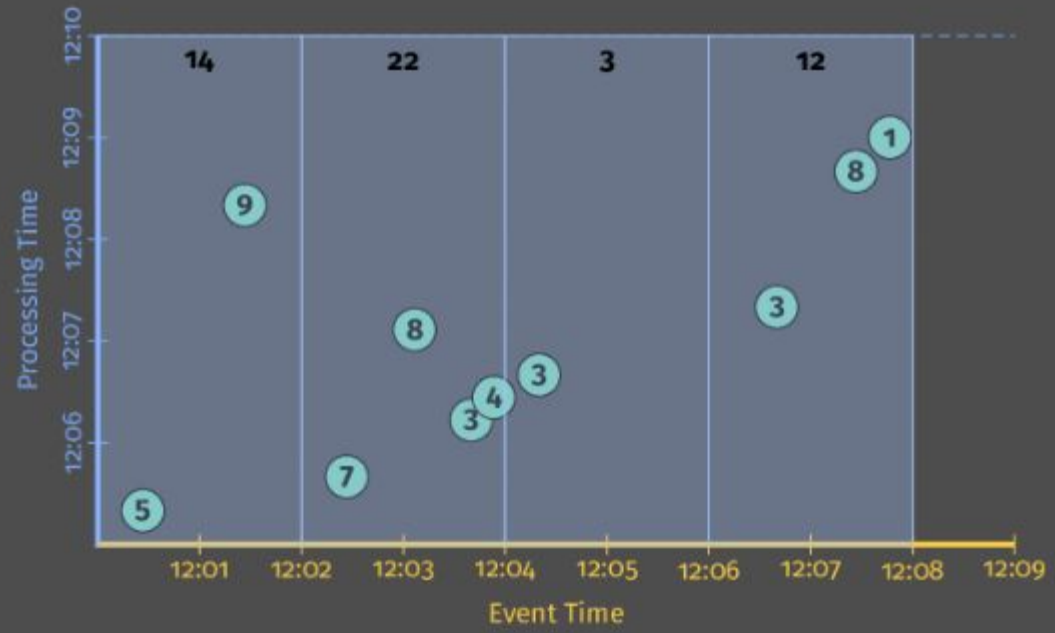


- Windowing divides data into event-time-based finite chunks.
- Required when doing aggregations over unbounded data.

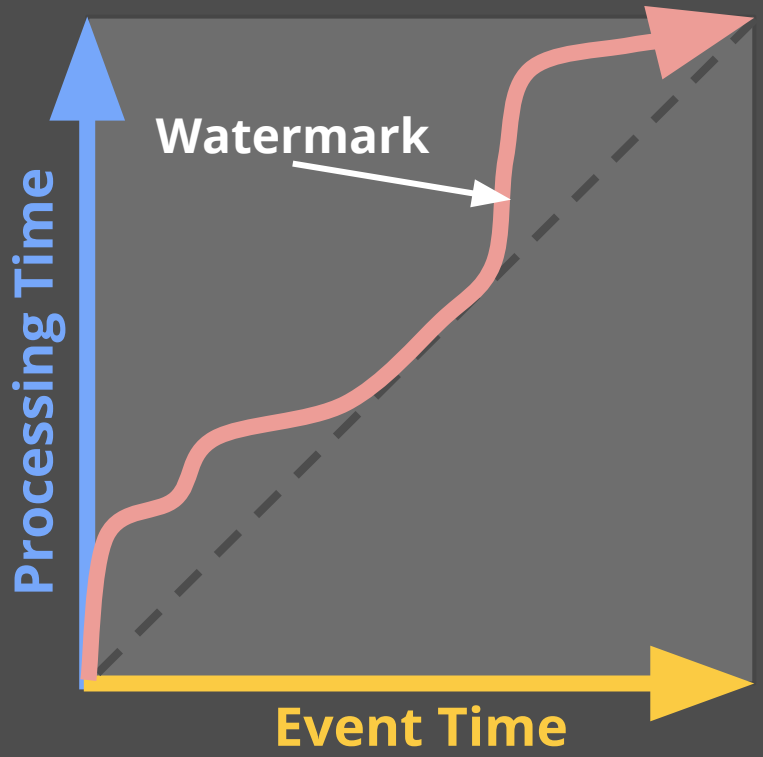
Example: Fixed 2-minute Windows

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

Example: Fixed 2-minute Windows



When in Processing Time?

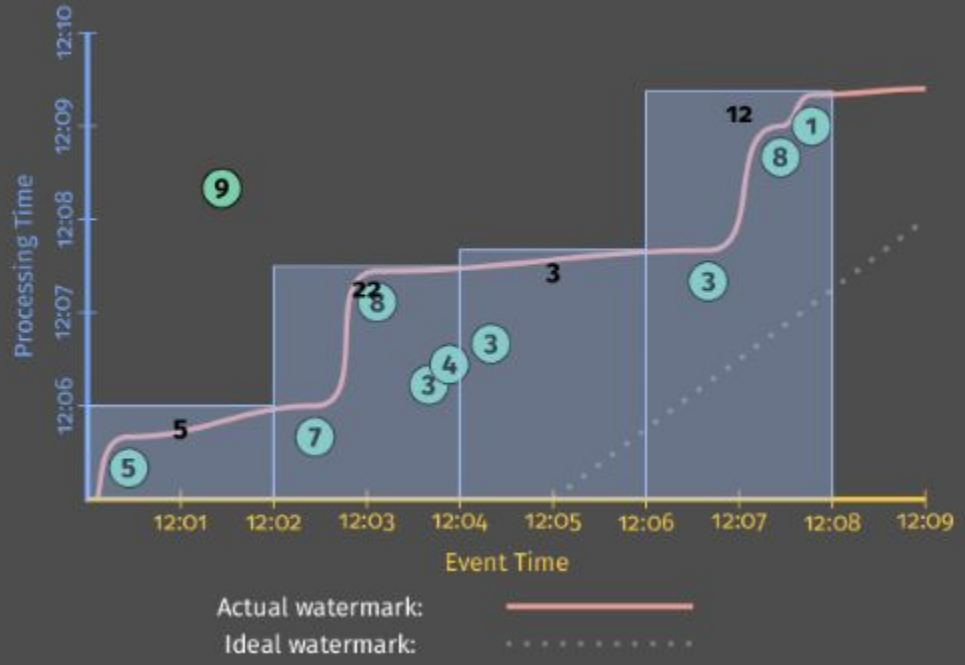


- Triggers control when results are emitted.
- Triggers are often relative to the watermark.

Example: Triggering at the Watermark

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

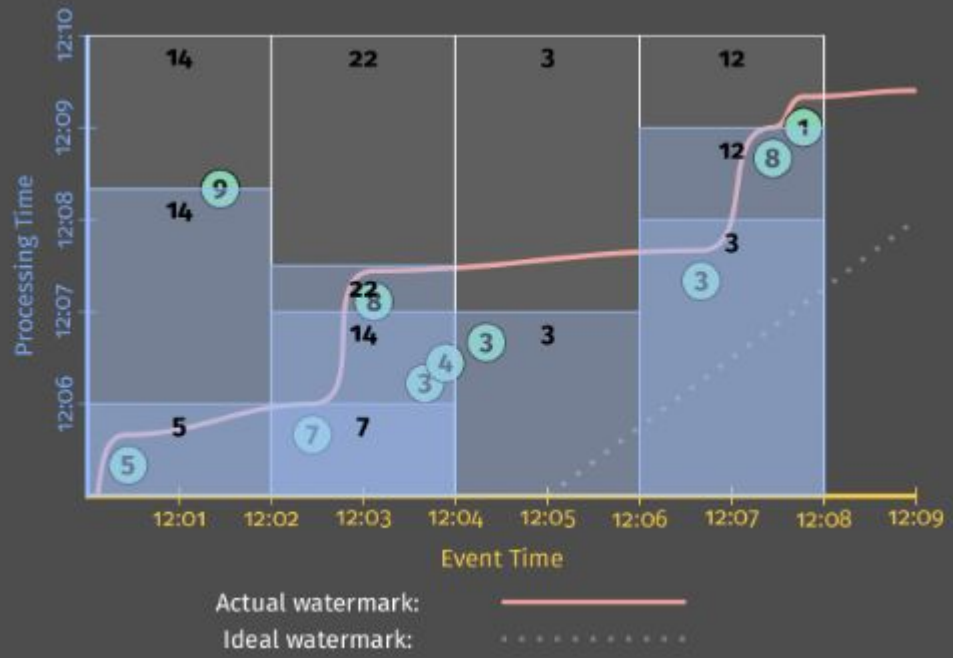
Example: Triggering at the Watermark



Example: Triggering for Speculative & Late Data

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

Example: Triggering for Speculative & Late Data



How are Results refined?

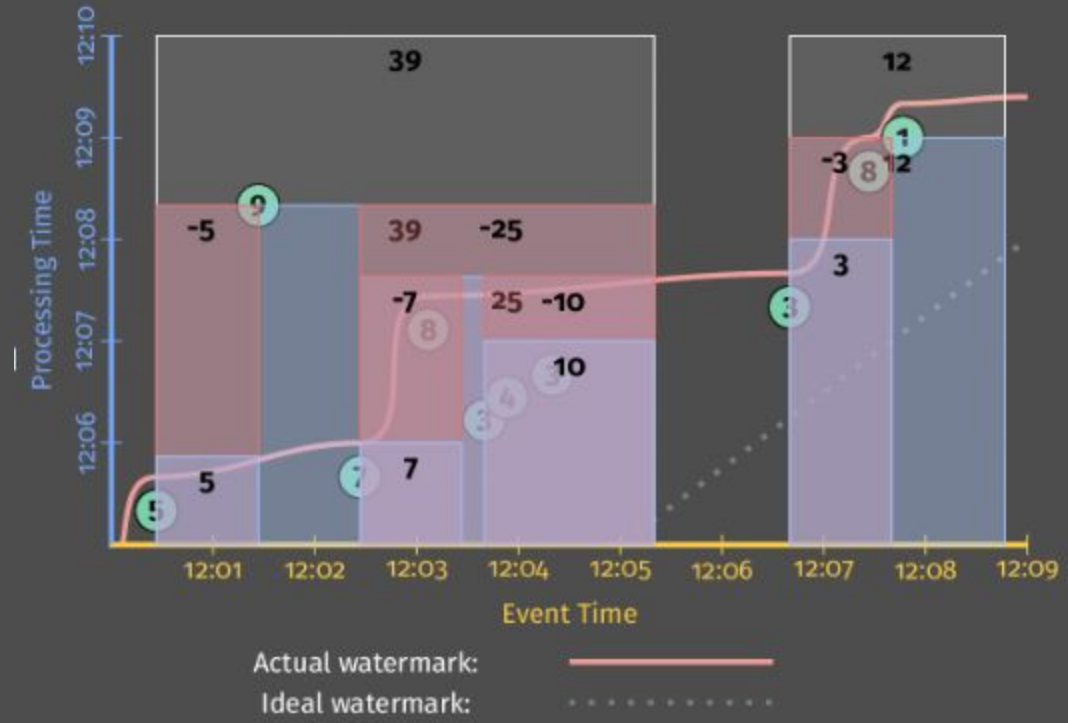
- How should multiple outputs per window accumulate?
- Appropriate choice depends on consumer.

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***

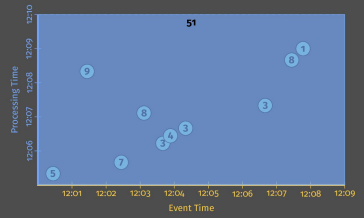
Example: Add Newest, Remove Previous

```
PCollection<KV<String, Integer>> output = input
  .apply(Window.into(Sessions.withGapDuration(Minutes(1)))
                .trigger(AtWatermark()
                  .withEarlyFirings(AtPeriod(Minutes(1)))
                  .withLateFirings(AtCount(1)))
                .accumulatingAndRetracting())
  .apply(new Sum());
```

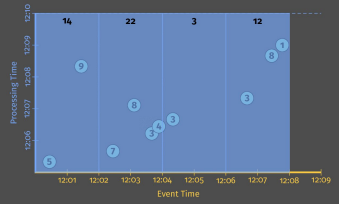
Example: Add Newest, Remove Previous



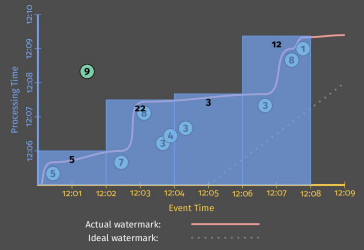
Customizing What Where When How



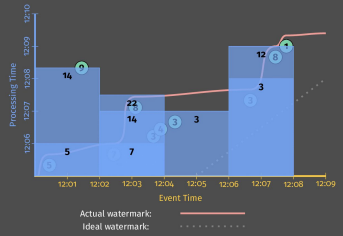
1. Classic Batch



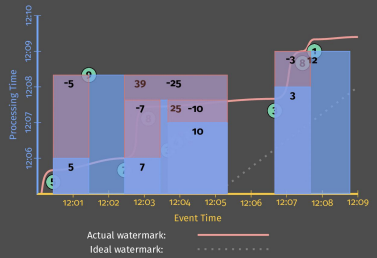
2. Batch with Fixed Windows



3. Streaming



4. Streaming with Speculative + Late Data



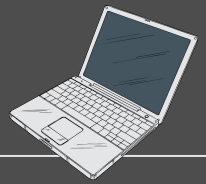
5. Streaming with Retractions

Google Cloud Dataflow

A fully-managed cloud service and programming model for batch and streaming big data processing.

Open Source SDKs

- Used to construct a Dataflow pipeline.
  - Custom IO: both bounded and unbounded
- Available in Java. Python in the works.
- Pipelines can run…
  - On your development machine
  - On the **Dataflow Service** on **Google Cloud Platform**
  - On third party environments like Spark or Flink.

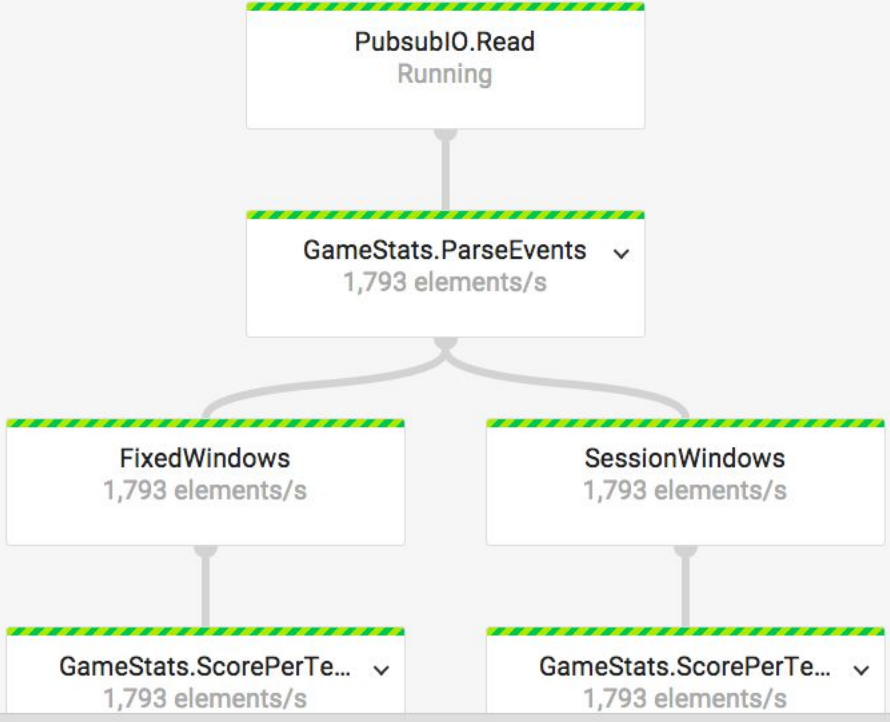


Fully Managed Dataflow Service

Runs the pipeline on Google Cloud Platform. Includes:

- *Graph optimization:* Modular code, efficient execution
- *Smart Workers:*
  - **Full Lifecycle management**,
  - **Autoscaling** (up and down)
  - **Dynamic task rebalancing**
- *Easy Monitoring:* Dataflow UI, Restful API and CLI, Integration with Cloud Logging, etc.

Cloud Dataflow Jobs / 2015-09-14_14_05_56-16113162258983054276



Summary | Job Log | Step

Cancel job | View

Job Name	livegamestats-fjp-0914210548
Job ID	2015-09-14_14_05_56-16113162258983054276
Job Status	Running
Job Type	Streaming
Start Time	Sep 14, 2015, 2:05:56 PM
Elapsed Time	7 hr 23 min
Errors	0
Warnings	0
Reserved CPU Time	29 min 30 sec

Custom counters

Filter

/DroppedDueToLateness	80

Learn More!

- The Dataflow Model @VLDB 2015
  http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf
- Dataflow SDK for Java
  https://github.com/GoogleCloudPlatform/DataflowJavaSDK
- Google Cloud Dataflow on Google Cloud Platform
  http://cloud.google.com/dataflow (Free Trial!)

Thank you

Google Cloud Platform