THE LINUX FOUNDATION
OPEN SOURCE SUMMIT

Apache Beam: portable and evolutive data-intensive applications

Ismaël Mejía - @iemejia

Talend

THE LINUX FOUNDATION

Who am I?

@iemejia

Software Engineer

Apache Beam PMC / Committer

ASF member

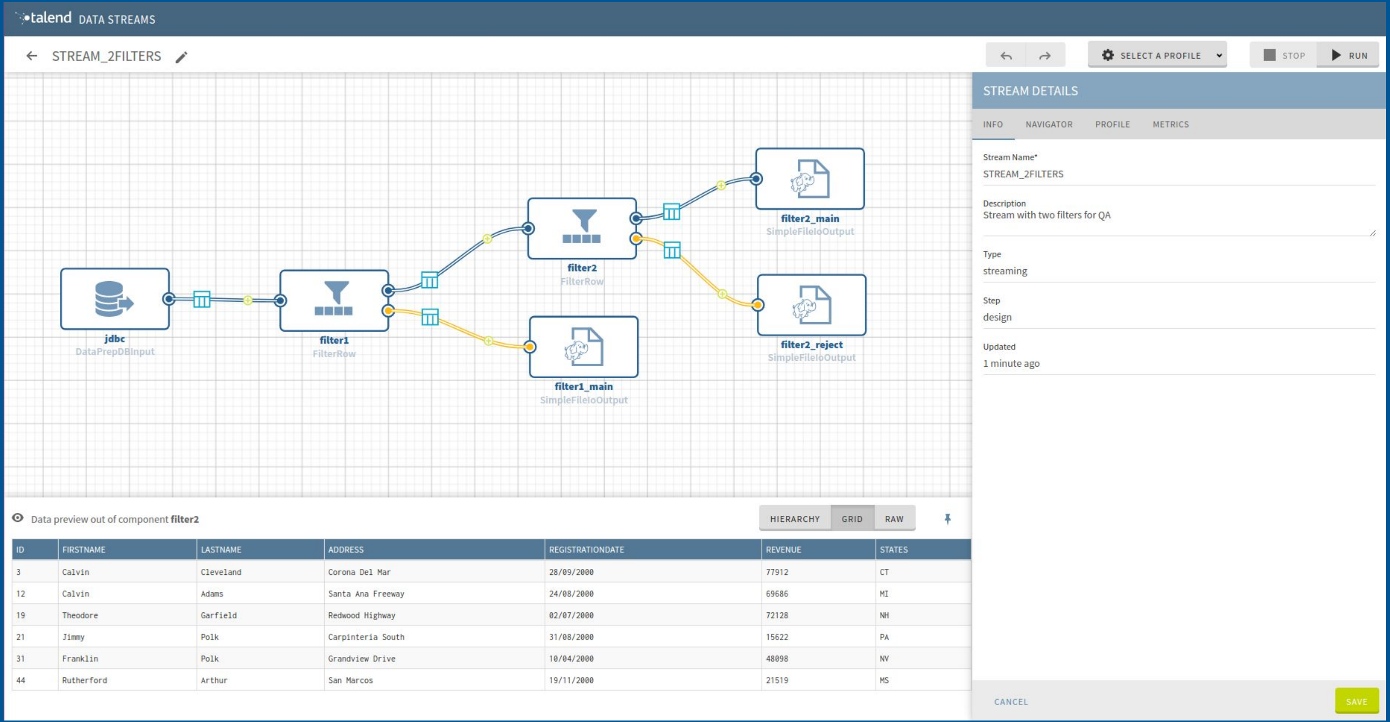
talend

Integration Software

Big Data / Real-Time

Open Source / Enterprise

New products



Stream Name*
STREAM_2FILTERS

Description
Stream with two filters for QA

Type
streaming

Step
design

Updated
1 minute ago

Data preview out of component **filter2**

ID	FIRSTNAME	LASTNAME	ADDRESS	REGISTRATIONDATE	REVENUE	STATES
3	Calvin	Cleveland	Corona Del Mar	28/09/2000	77912	CT
12	Calvin	Adams	Santa Ana Freeway	24/08/2000	69686	MI
19	Theodore	Garfield	Redwood Highway	02/07/2000	72128	NH
21	Jimmy	Polk	Carpinteria South	31/08/2000	15622	PA
31	Franklin	Polk	Grandview Drive	10/04/2000	48098	NV
44	Rutherford	Arthur	San Marcos	19/11/2000	21519	MS

We are hiring !

Introduction: Big data state of affairs

Before Big Data (early 2000s)

The **web pushed data** analysis / infrastructure **boundaries**

- Huge data analysis needs (Google, Yahoo, etc)
- Scaling DBs for the web (most companies)

DBs (and in particular RDBMS) had too many constraints and it was hard to operate at scale.

Solution: We need to go back to basics but in a distributed fashion

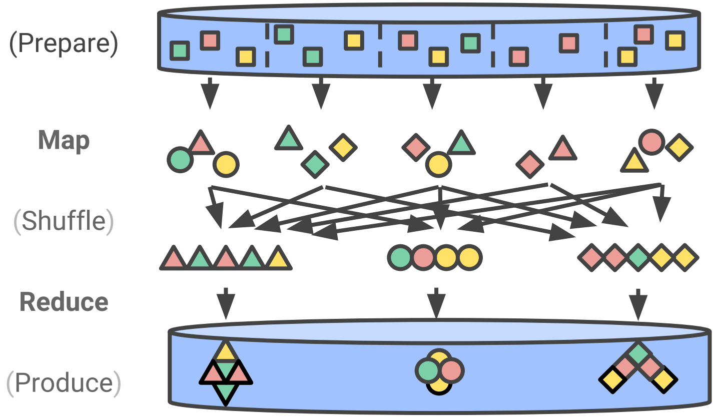
MapReduce, Distributed Filesystems and Hadoop

- Use distributed file systems (HDFS) to scale data storage horizontally
- Use Map Reduce to execute tasks in parallel (performance)
- Ignore strict model (let representation loose to ease scaling e.g. KV stores).

Great for huge dataset analysis / transformation

but…

- Too low-level for many tasks (early frameworks)
- Not suited for latency dependant analysis

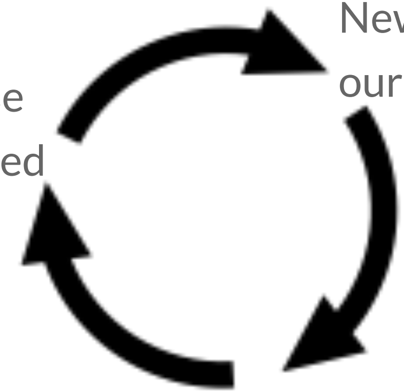


The distributed database Cambrian explosion

cassandra
Solr
CouchDB
relax
mongoDB
elasticsearch
MEMCACHED
redis
neo4j
APACHE
HBASE

… and **MANY** others, all of them with different properties, utilities and APIs

Distributed databases API cycle



NoSQL, because
SQL is too limited

NewSQL let's reinvent
our own thing

SQL is back,
because it is awesome

(yes it is an over-simplification but you get it)

The fundamental problems are **still** the same

or **worse** (because of heterogeneity) …

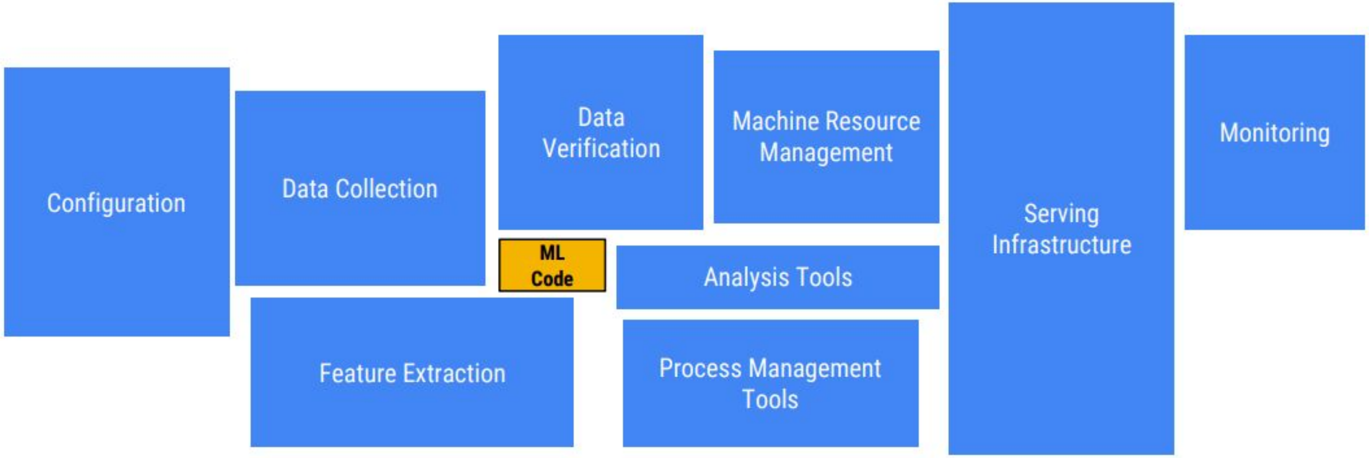
- Data analysis / processing from systems with different semantics
- Data integration from heterogeneous sources
- Data infrastructure operational issues

Good old **Extract-Transform-Load (ETL)** is still an important need

The fundamental problems are **still** the same

"Data preparation accounts for about 80% of the work of data scientists" [1]

[2]



[1] Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task

[2] Sculley et al.: Hidden Technical Debt in Machine Learning Systems

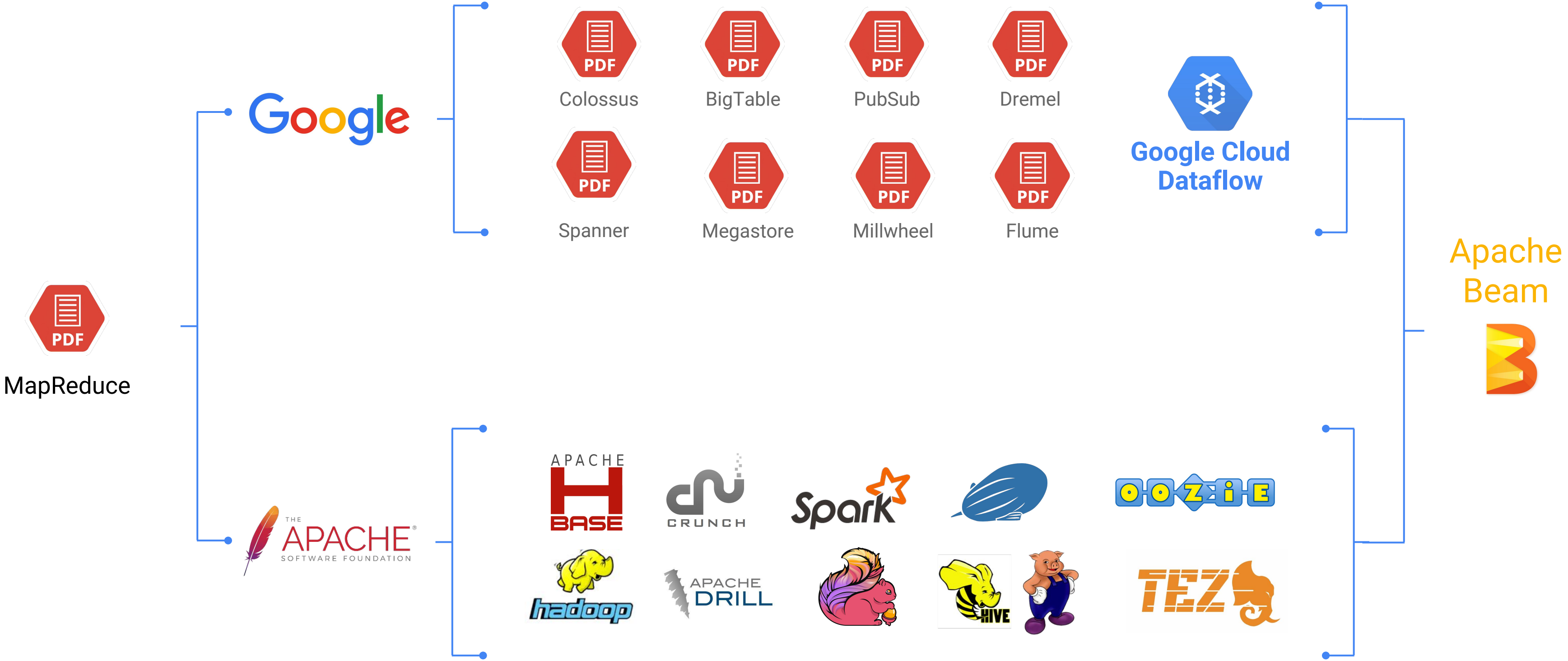
and evolution continues ...

- **Latency needs:** Pseudo real-time needs, distributed logs.
- **Multiple platforms:** On-premise, cloud, cloud-native (also multi-cloud).
- **Multiple languages and ecosystems:** To integrate with ML tools

Software issues: **New APIs, new clusters, different semantics,**
... and of course MORE data stores !

Apache Beam

Apache Beam origin



MapReduce

Google

Colossus

BigTable

PubSub

Dremel

Spanner

Megastore

Millwheel

Flume

Google Cloud Dataflow

Apache Beam

What is Apache Beam?

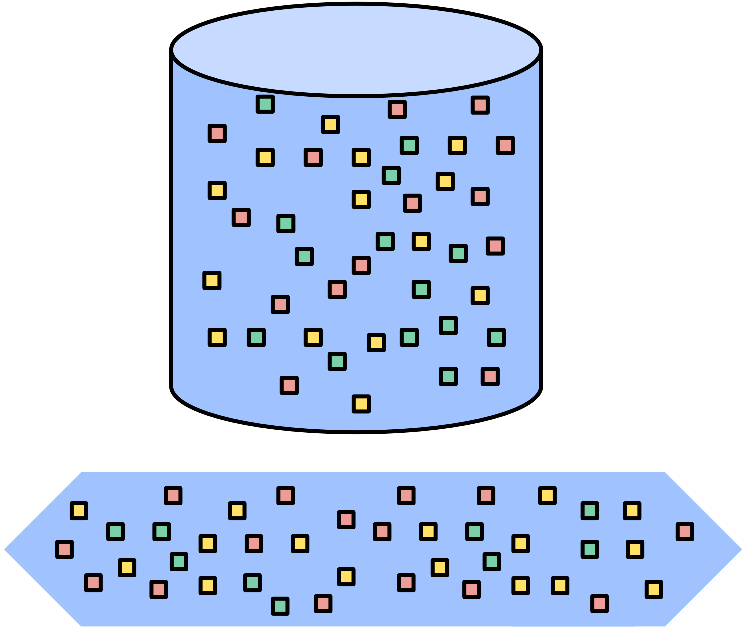
Apache Beam is a unified programming model designed to provide efficient and portable data processing pipelines

Beam Model: Generations Beyond MapReduce

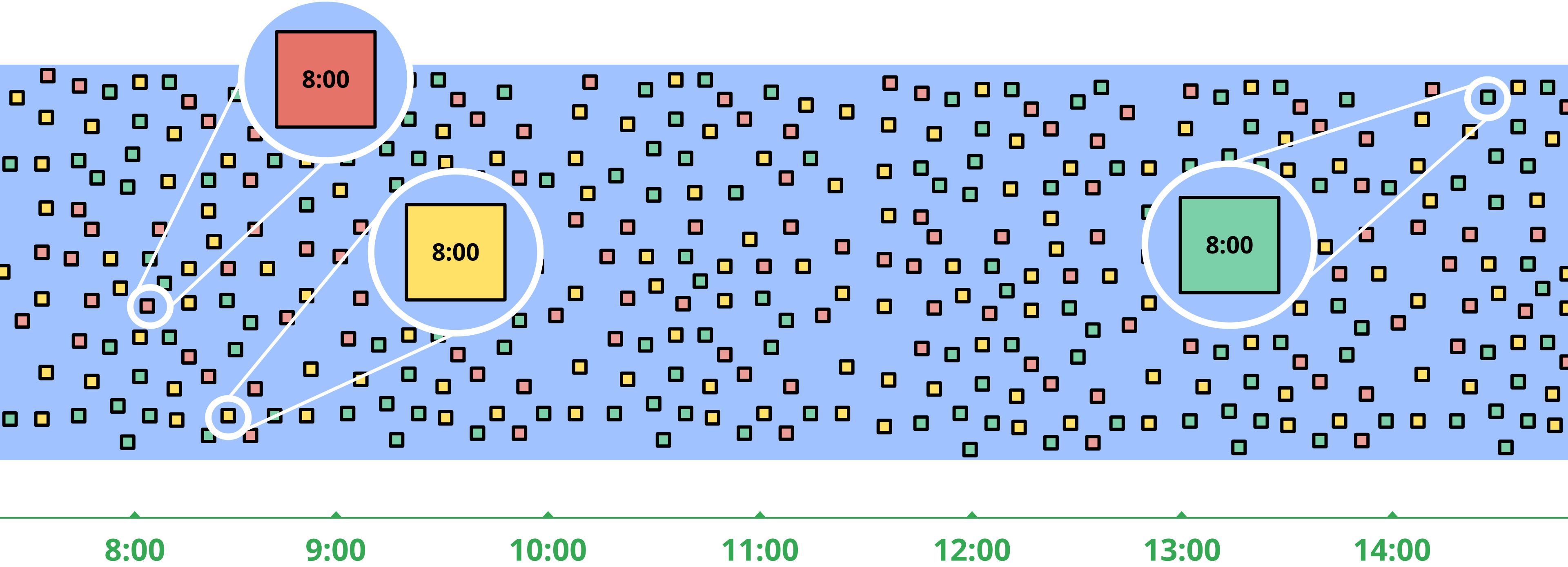
Improved abstractions let you focus on your application logic

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

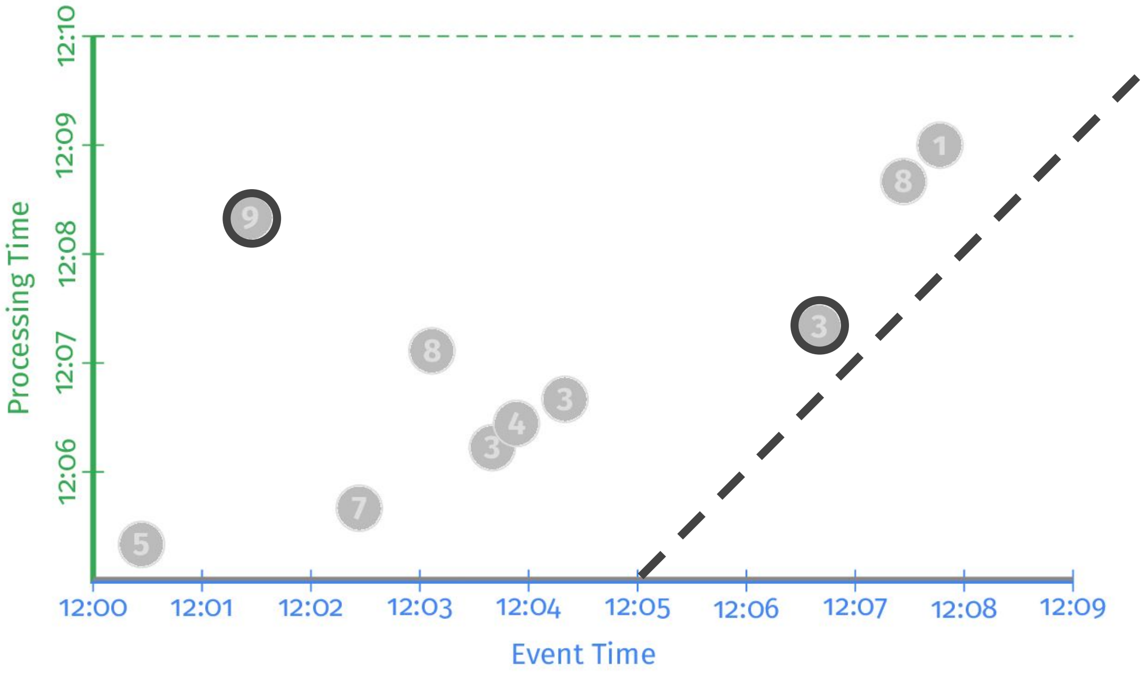
Clearly separates event time from processing time.



Streaming - late data



Processing Time vs. Event Time



Beam Model: Asking the Right Questions

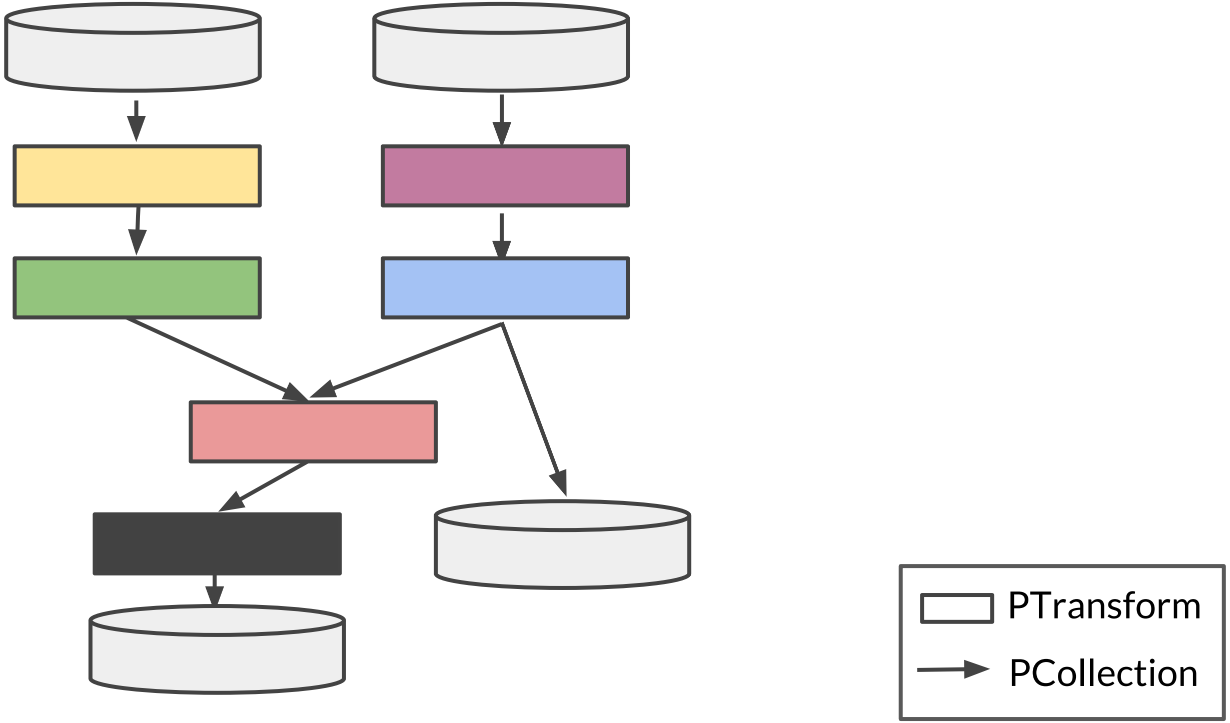
What results are calculated?

Where in event time are results calculated?

When in processing time are results materialized?

How do refinements of results relate?

Beam Pipelines



PTransform

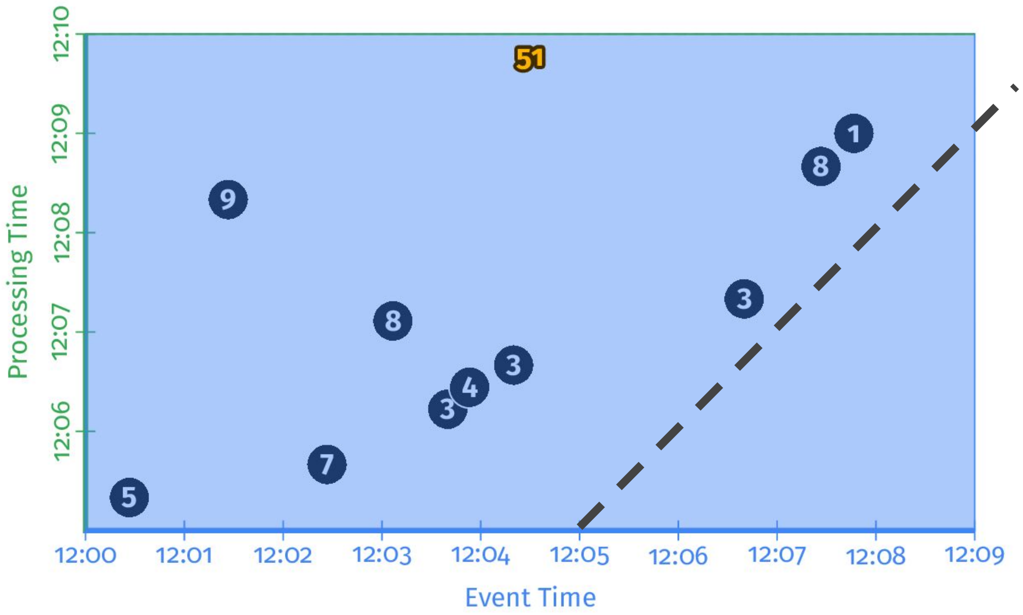
PCollection

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?



Event Time: Timestamp when the event happened

Processing Time: Absolute program time (wall clock)

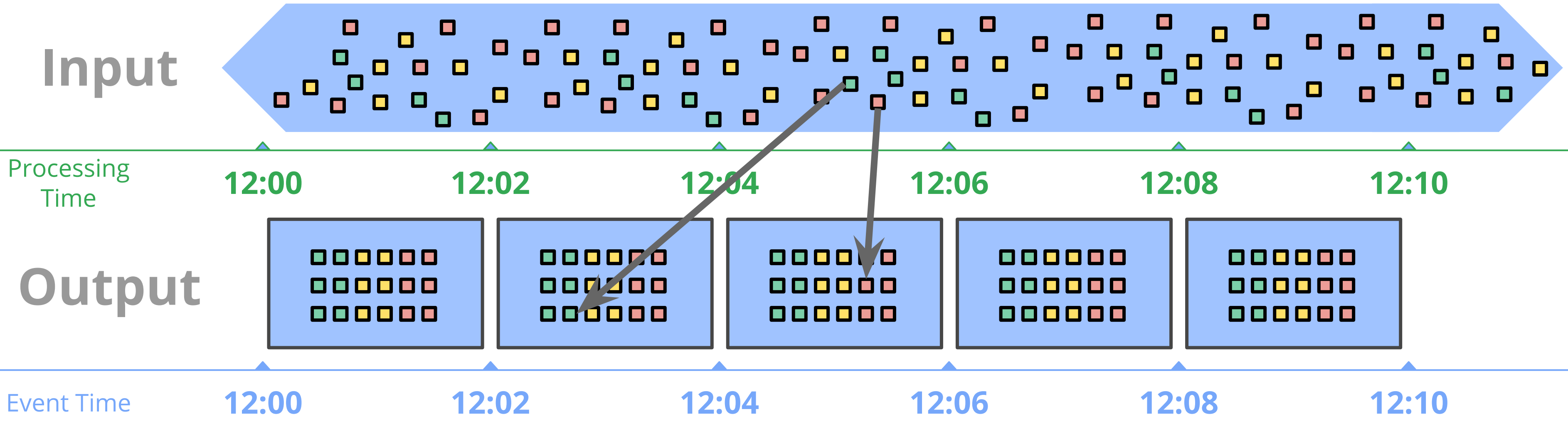
The Beam Model: Where in Event Time?

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

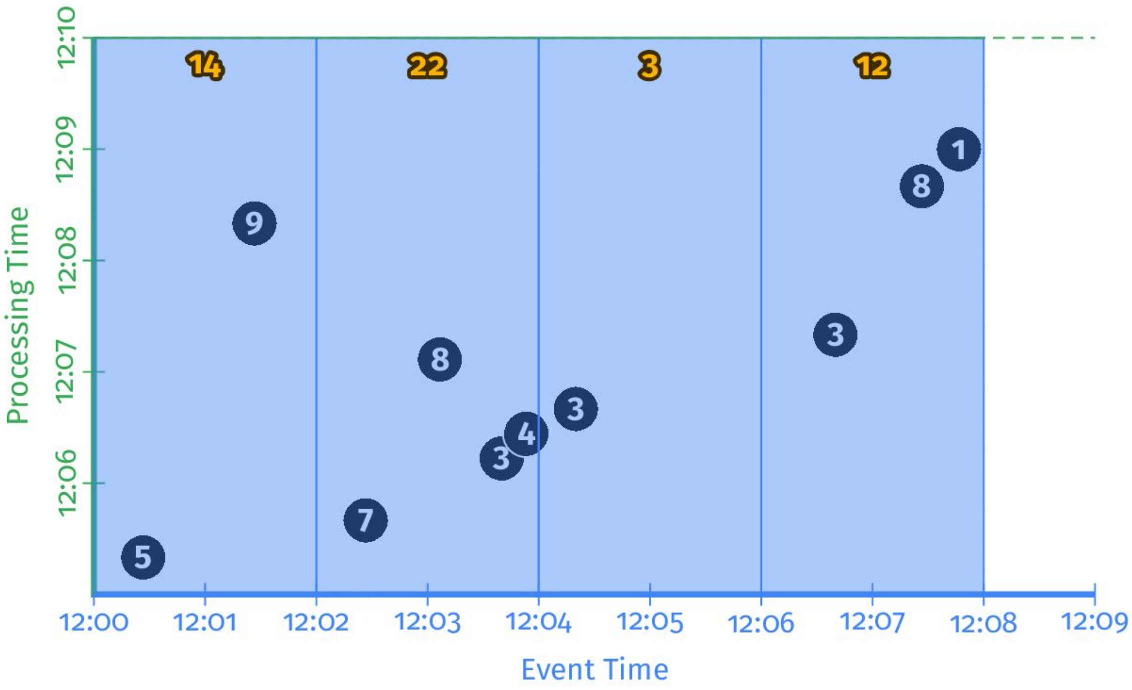
```
scores = (input
    | beam.WindowInto(FixedWindows(2 * 60))
    | Sum.integersPerKey())
```

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

- Split infinite data into finite chunks



The Beam Model: Where in Event Time?

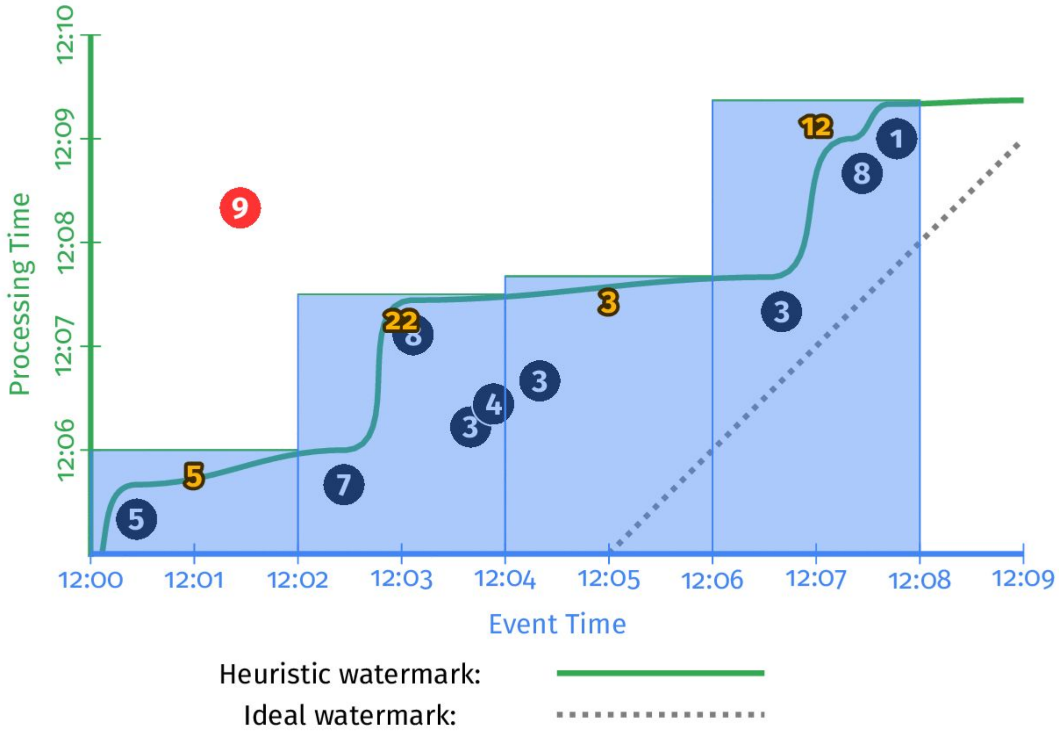


The Beam Model: **When** in Processing Time?

```
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?

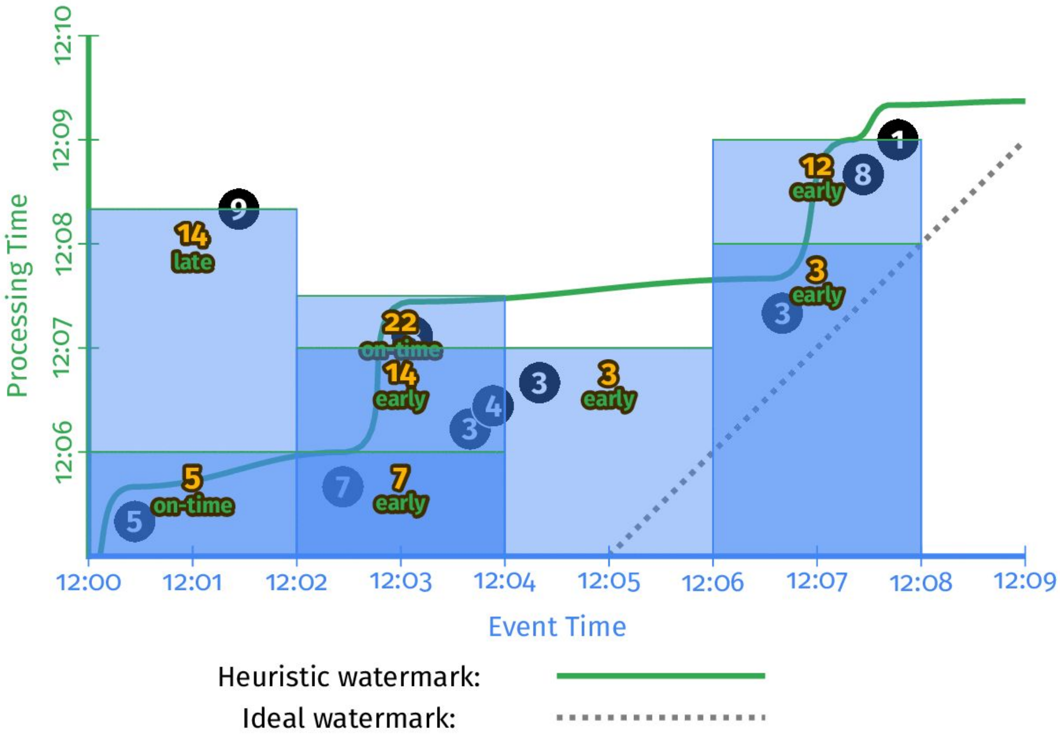


The Beam Model: How Do Refinements Relate?

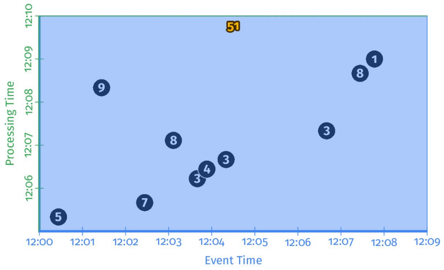
```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Duration.standardMinutes(2))
           .triggering(AtWatermark()
              .withEarlyFirings(AtPeriod(Duration.standardMinutes(1)))
              .withLateFirings(AtCount(1)))
           .accumulatingFiredPanes())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

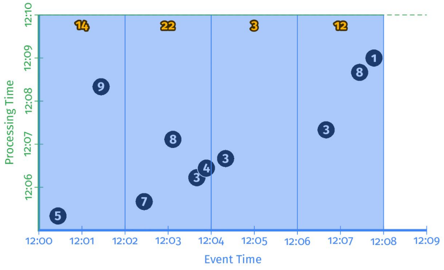


Customizing What Where When How



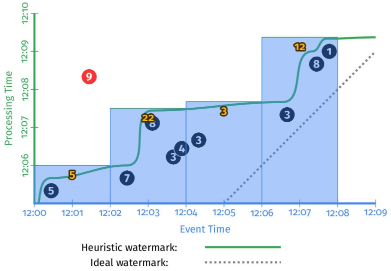
1

Classic Batch



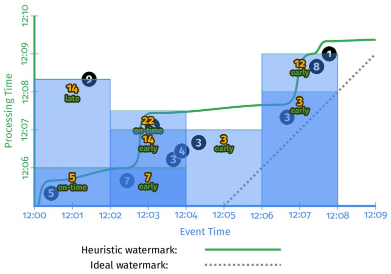
2

Windowed Batch



3

Streaming

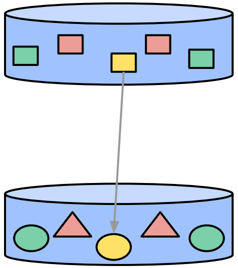


4

Streaming + Accumulation

Apache Beam - Programming Model

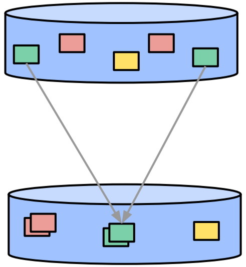
Element-wise



ParDo -> DoFn
MapElements
FlatMapElements
Filter

WithKeys
Keys
Values

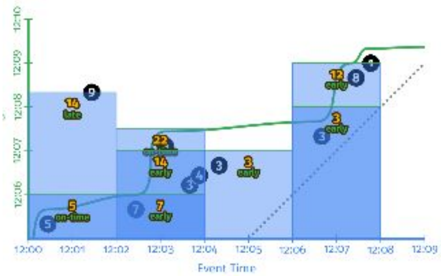
Grouping



GroupByKey
CoGroupByKey

Combine -> Reduce
Sum
Count
Min / Max
Mean
...

Windowing/Triggers

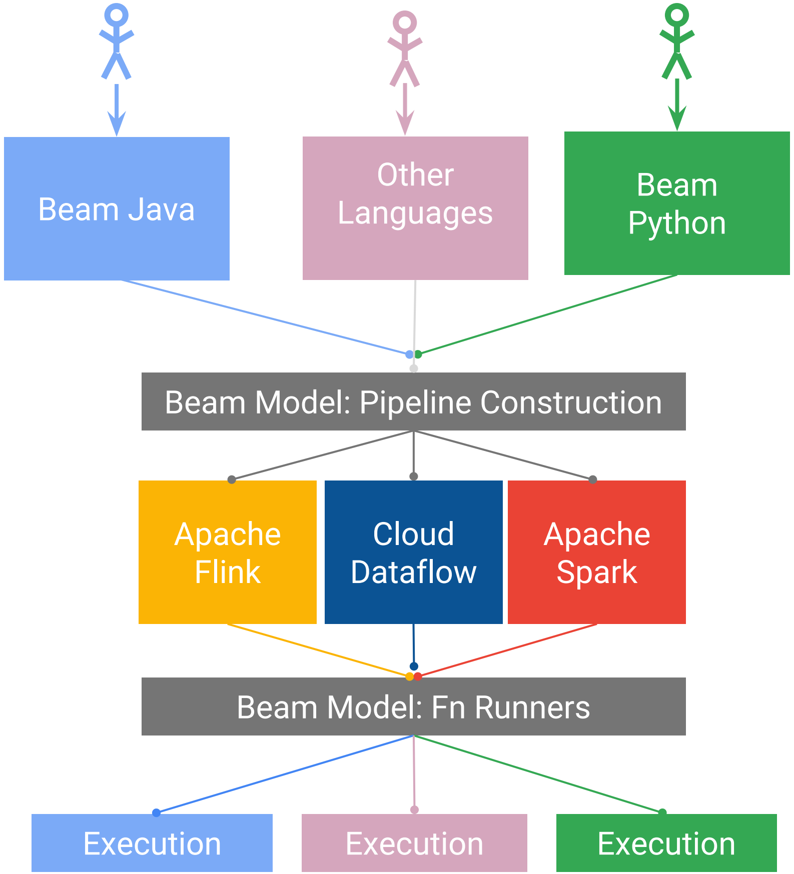


Windows
FixedWindows
GlobalWindows
SlidingWindows
Sessions

Triggers
AfterWatermark
AfterProcessingTime
Repeatedly

The Apache Beam Vision

1. **End users:** who want to write pipelines in a language that’s familiar
   .
2. **Library / IO connectors:** Who want to create generic transforms.
3. **SDK writers:** who want to make Beam concepts available in new languages.
4. **Runner writers:** who have a distributed processing environment and want to support Beam pipelines



Runners

Runners “**translate**” the code into the target runtime

Apache Beam Direct Runner

Apache Apex

Apache Spark

Apache Flink

Apache Gearpump

Google Cloud Dataflow

IBM Streams

Apache Storm

Ali Baba JStorm

Apache Samza

Hadoop MapReduce

WIP

* Same code, different runners & runtimes

Beam IO (Data store connectors)

Filesystems: Google Cloud Storage, Hadoop FileSystem, AWS S3, Azure Storage (in progress)

File support: Text, Avro, Parquet, Tensorflow

Cloud databases: Google BigQuery, BigTable, DataStore, Spanner, AWS Redshift (in progress)

Messaging: Google Pubsub, Kafka, JMS, AMQP, MQTT, AWS Kinesis, AWS SNS, AWS SQS

Cache: Redis, Memcached (in progress)

Databases: Apache HBase, Cassandra, Hive (HCatalog), Mongo, JDBC

Indexing: Apache Solr, Elasticsearch

And other nice ecosystem tools / libraries:

Scio: Scala API by Spotify

Euphoria: Alternative Java API closer to Java 8 collections

Extensions: joins, sorting, probabilistic data structures, etc.

A simple evolution example

A log analysis simple example

Logs rotated and stored in HDFS and analyzed daily to measure user engagement.
Running on-premise Hadoop cluster with Spark

Data:

```
64.242.88.10   user01   07/Mar/2018:16:05:49  /news/abfg6f
64.242.88.10   user01   07/Mar/2018:16:05:49  /news/de0aff
...
```

Output:

```
user01, 32 urls, 2018/03/07
```

A log analysis simple example

```
PCollection<KV<User, Long>> numVisits =
   pipeline
       .apply(TextIO.read().from("hdfs://..."))
       .apply(MapElements.via(new ParseLog()))
       .apply(Count.perKey());
```

```
$ mvn exec:java -Dexec.mainClass=beam.example.loganalysis.Main -Pspark-runner
-Dexec.args="--runner=SparkRunner --master=tbd-bench"
```

A log analysis simple example

Remember the software engineering maxima:

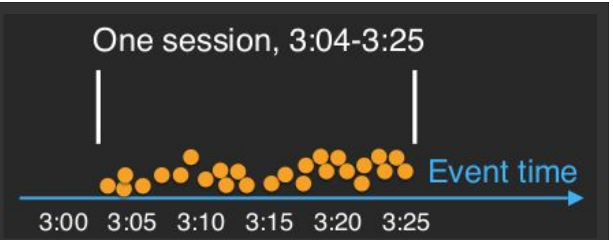
Requirements always change

We want to identify user sessions and calculate the number of URL visits per session
and we need quicker updates from a different source, a Kafka topic
and we will run this in a new Flink cluster

* Session = a sustained burst of activity

A log analysis simple example

```
PCollection<KV<User, Long>> numVisitsPerSession =
pipeline
    .apply(
        KafkaIO.<Long, String>read()
            .withBootstrapServers("hostname")
            .withTopic("visits"))
    .apply(Values.create())
    .apply(MapElements.via(new ParseLog()))
    .apply(Window.into(Sessions.withGapDuration(Duration.standardMinutes(10))))
    .apply(Count.perKey());
```



```
$ mvn exec:java -Dexec.mainClass=beam.example.loganalysis.Main -Pflink-runner
-Dexec.args="--runner=FlinkRunner --master=realtime-cluster-master"
```

Apache Beam Summary

Expresses data-parallel **batch and streaming** algorithms with one **unified** API.

Cleanly **separates** data processing **logic** from **runtime requirements**.

Supports execution on **multiple distributed processing runtime** environments.

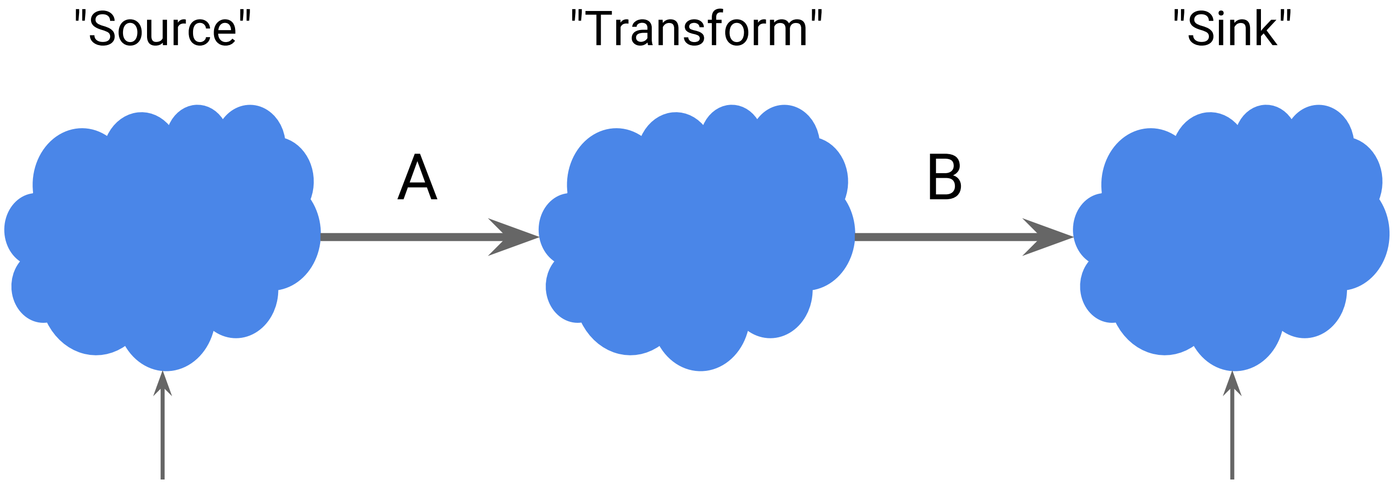
Integrates with the larger data processing **ecosystem**.

Current status and upcoming features

Beam is evolving too...

- Streaming SQL support via Apache Calcite
- Schema-aware PCollections friendlier APIs
- Composable IO Connectors: Splittable DoFn (SDF) (New API)
- Portability: Open source runners support for language portability
- Go SDK finally gophers become first class citizens on Big Data

IO connectors APIs are too strict



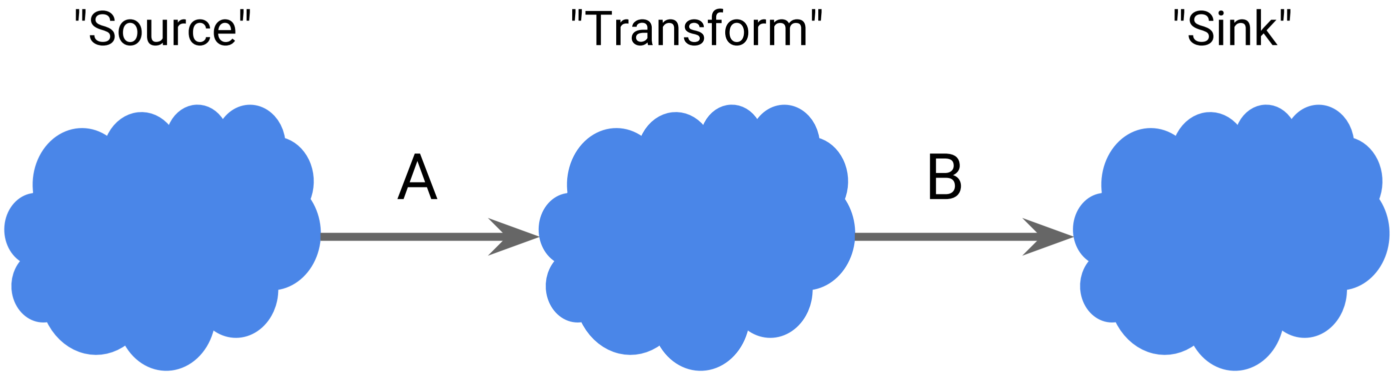
InputFormat / Receiver / SourceFunction / ...

Configuration:
Filepattern
Query string
Topic name
...

OutputFormat / Sink / SinkFunction / ...

Configuration:
Directory
Table name
Topic name
...

SDF - Enable composable IO APIs



"Source"

A

"Transform"

B

"Sink"

My filenames come on a Kafka topic.

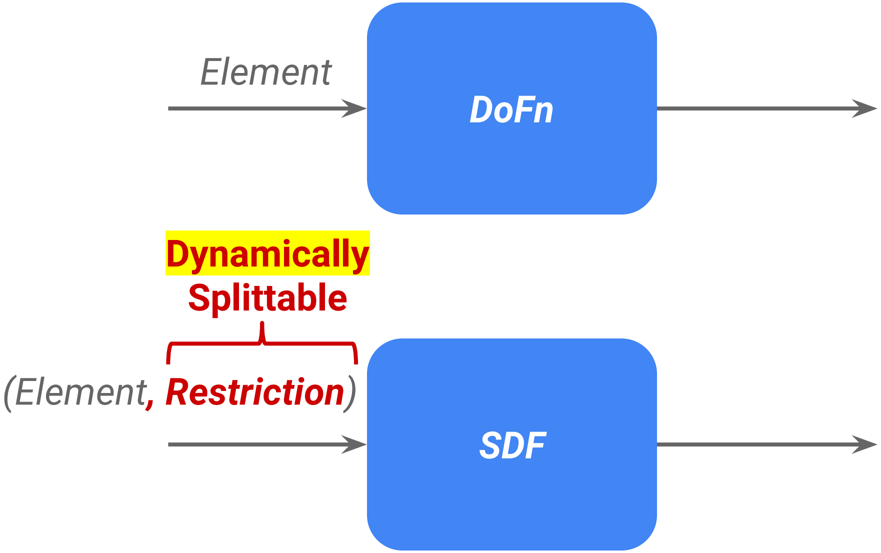
I have a table per client + table of clients

Narrow APIs are not **hackable**

I want to know which records failed to write

I want to kick off another transform after writing

Splittable DoFn (SDF): Partial work via restrictions



Element: what work

Restriction: what **part** of the work

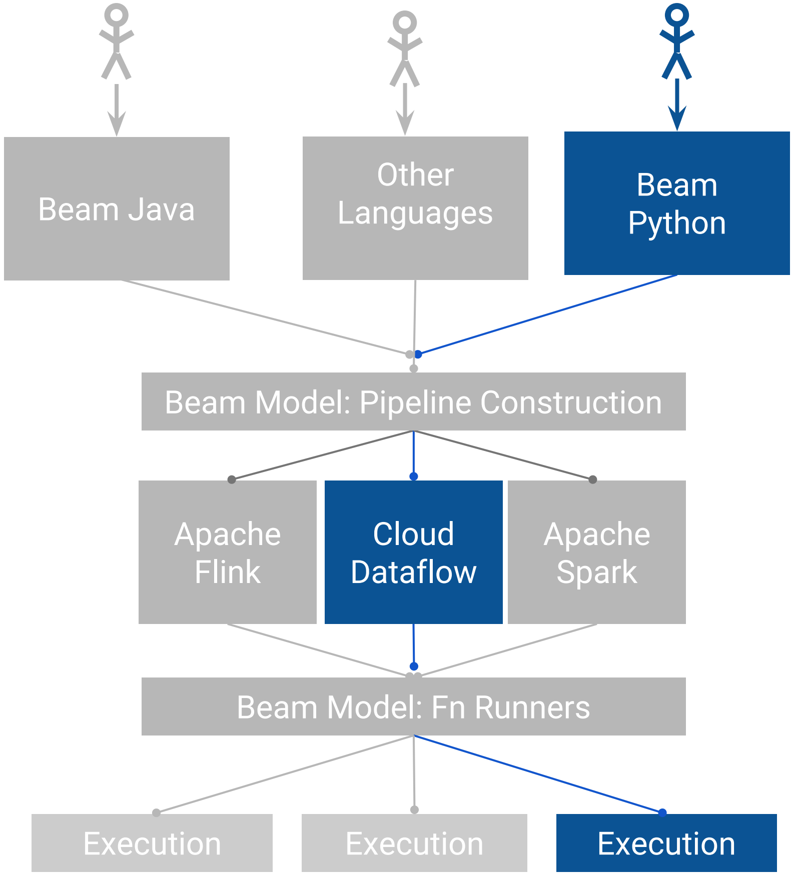
Design: s.apache.org/splittable-do-fn

* More details in this video by Eugene Kirpichov

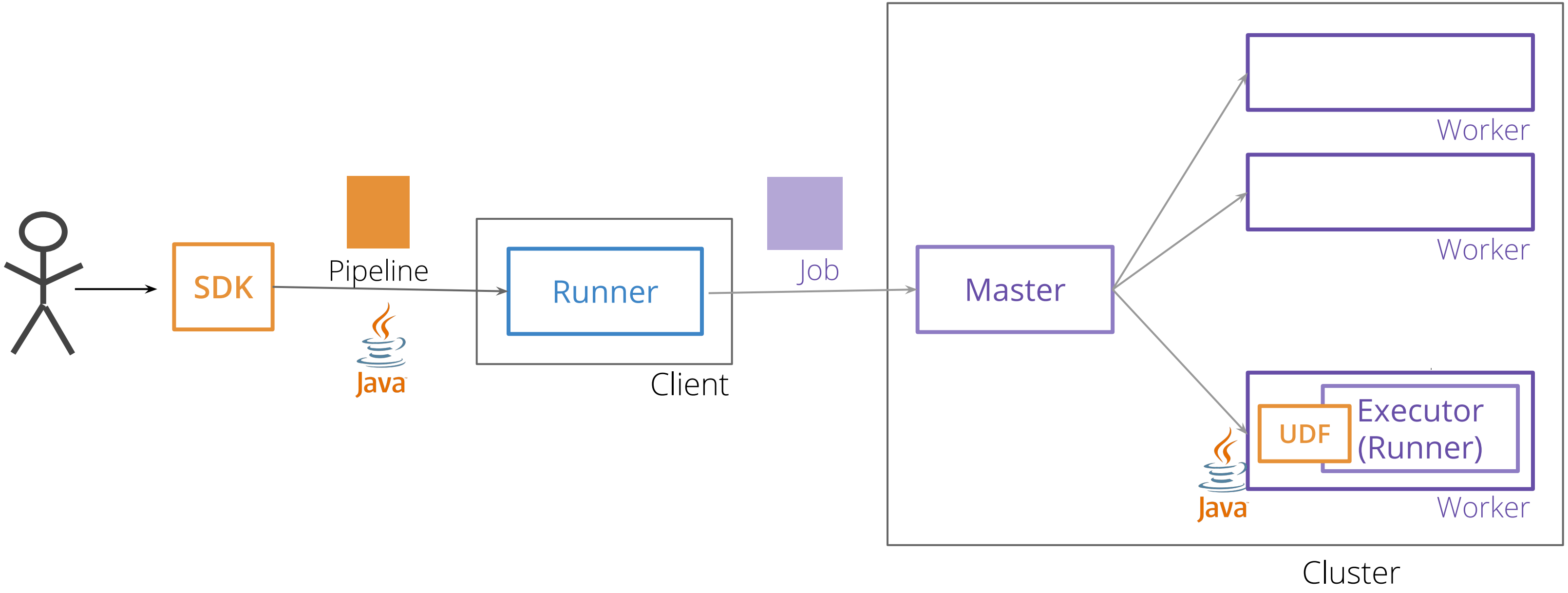
Language portability

- If I run a Beam python pipeline on the Spark runner, is it translated to PySpark?
- Wait, can I execute python on a Java based runner?
- Can I use the python Tensorflow transform from a Java pipeline?
- I want to connect to Kafka from Python but there is not a connector can I use the Java one?

No

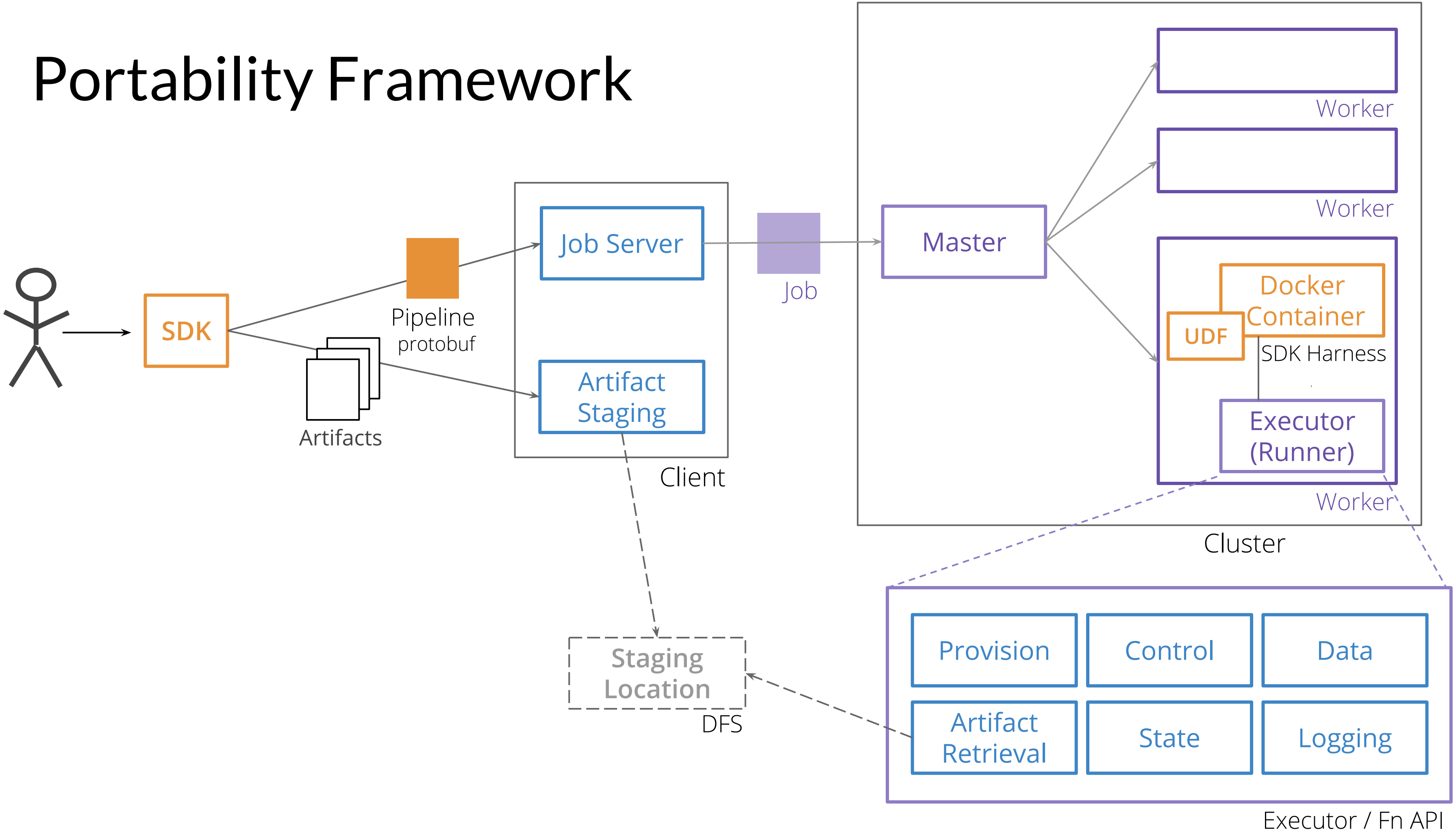


How do Java-based runners do work today?



Executor / Fn API

Portability Framework



Language portability advantages

Isolation of user code

Isolated configuration of user environment

Multiple language execution

Mix user code in different languages

Makes creating new SDK easier (homogeneous)

Issues

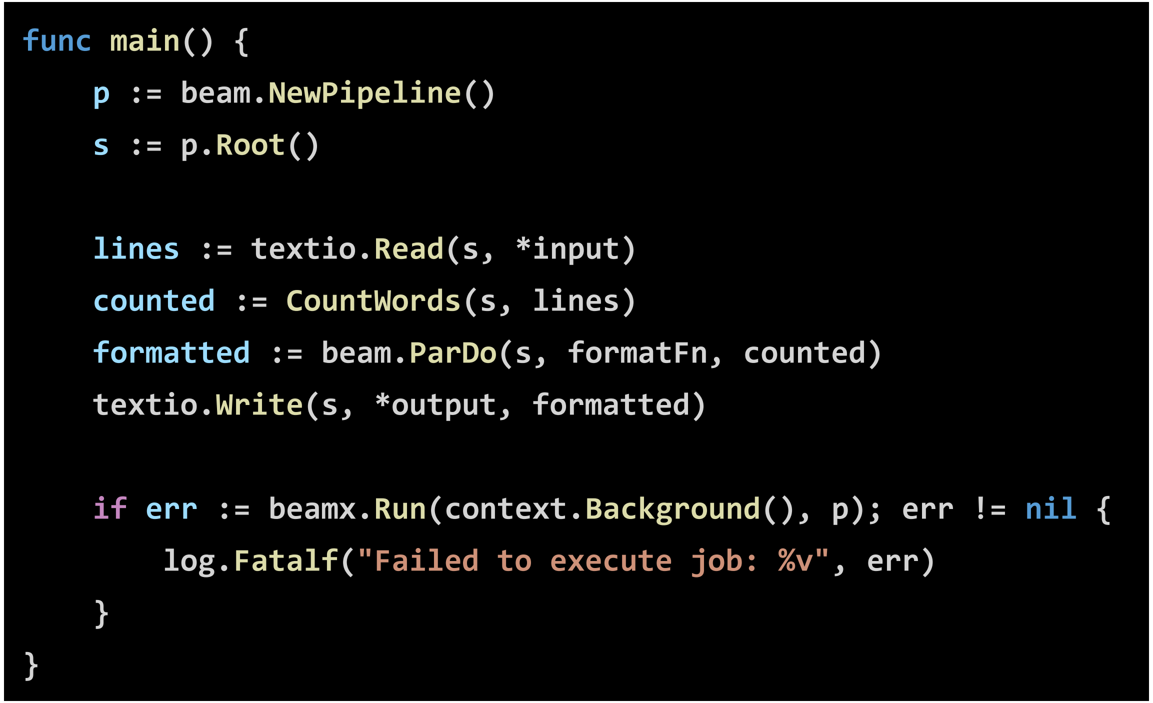
Performance overhead (15% in early evaluation). via extra RPC + container

Extra component (docker)

A bit more complex but it is the price of reuse and consistent environments

Go SDK

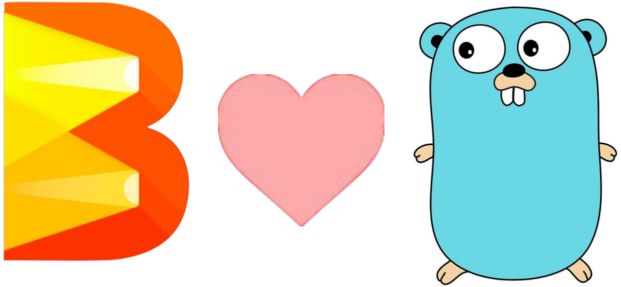
First user SDK completely based on Portability API.



```go
func main() {
    p := beam.NewPipeline()
    s := p.Root()

    lines := textio.Read(s, *input)
    counted := CountWords(s, lines)
    formatted := beam.ParDo(s, formatFn, counted)
    textio.Write(s, *output, formatted)

    if err := beamx.Run(context.Background(), p); err != nil {
        log.Fatalf("Failed to execute job: %v", err)
    }
}
```



Contribute

A vibrant community of contributors + companies:

Google, data Artisans, Lyft, Talend, Yours?

- Try it and help us report (and fix) issues.
- Multiple Jiras that need to be taken care of.
- New feature requests, new ideas, more documentation.
- More SDKs (more languages) .net anyone please, etc
- More runners, improve existing, a native go one maybe?

Beam is in a perfect shape to jump in.

First Stable Release. 2.0.0 API stability contract (May 2017)
Current: 2.6.0

Learn More!

Apache Beam
https://beam.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 mailing lists!
user-subscribe@beam.apache.org
dev-subscribe@beam.apache.org

Follow @ApacheBeam on Twitter

* The nice slides with animations were created by Tyler Akidau and Frances Perry and used with authorization.
Special thanks too to Eugene Kirpichov, Dan Halperin and Alexey Romanenko for ideas for this presentation.

Thanks

THE LINUX FOUNDATION

OPEN SOURCE SUMMIT

THE LINUX FOUNDATION