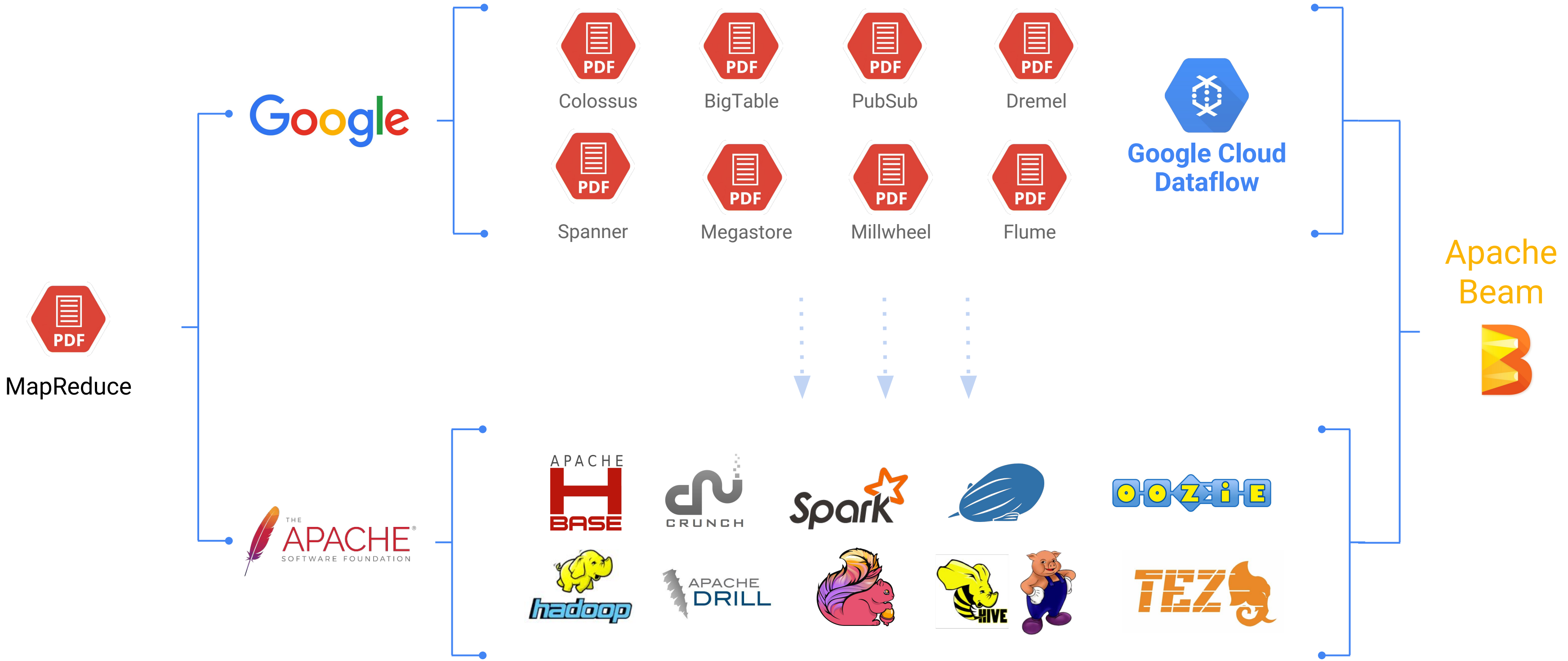
Efficient and portable data processing with Apache Beam and HBase

Eugene Kirpichov, Google

Agenda

1. History of Beam
2. Philosophy of the Beam programming model
3. Apache Beam project
4. Beam and HBase

The Evolution of Apache Beam



MapReduce

Google

Colossus

BigTable

PubSub

Dremel

Spanner

Megastore

Millwheel

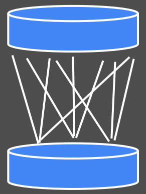
Flume

Google Cloud Dataflow

Apache Beam

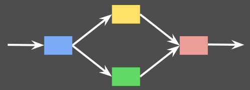
(2004) MapReduce

SELECT + GROUPBY



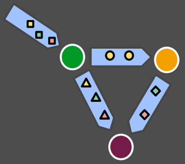
(2008) FlumeJava

High-level API



(2013) Millwheel

Deterministic
Streaming



(2014) Dataflow

Batch/streaming agnostic,
Infinite out-of-order data,
Portable

(2016) Apache Beam

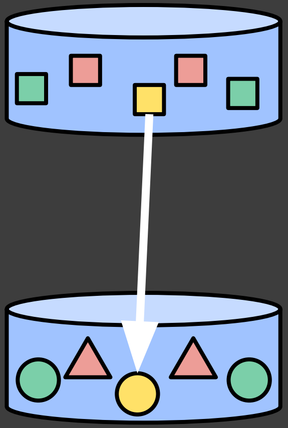
Open ecosystem,
Community-driven
Vendor-independent

Beam model: Unbounded, temporal, out-of-order data

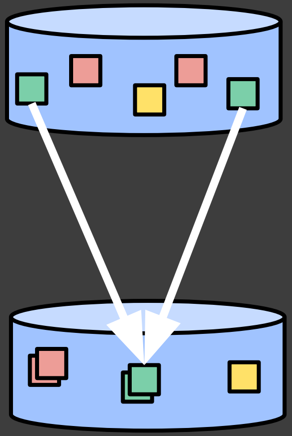
Unified	No concept of "batch" / "streaming" at all
Time	Event time (*when it happened*, not *when we saw it*)
Windowing	Aggregation within time windows
Keys	Windows scoped to a key (e.g. user sessions)
Triggers	When is a window "complete enough" What to do when late data arrives

What are you computing?	Transforms
Where in event time?	Windowing
When in processing time?	Triggers
How do refinements relate?	

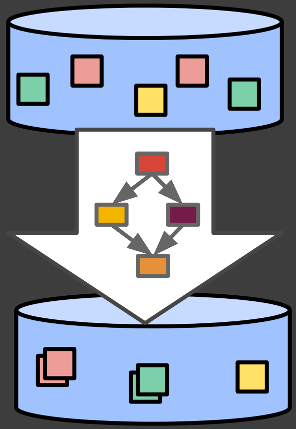
What - transforms



Element-Wise



Aggregating

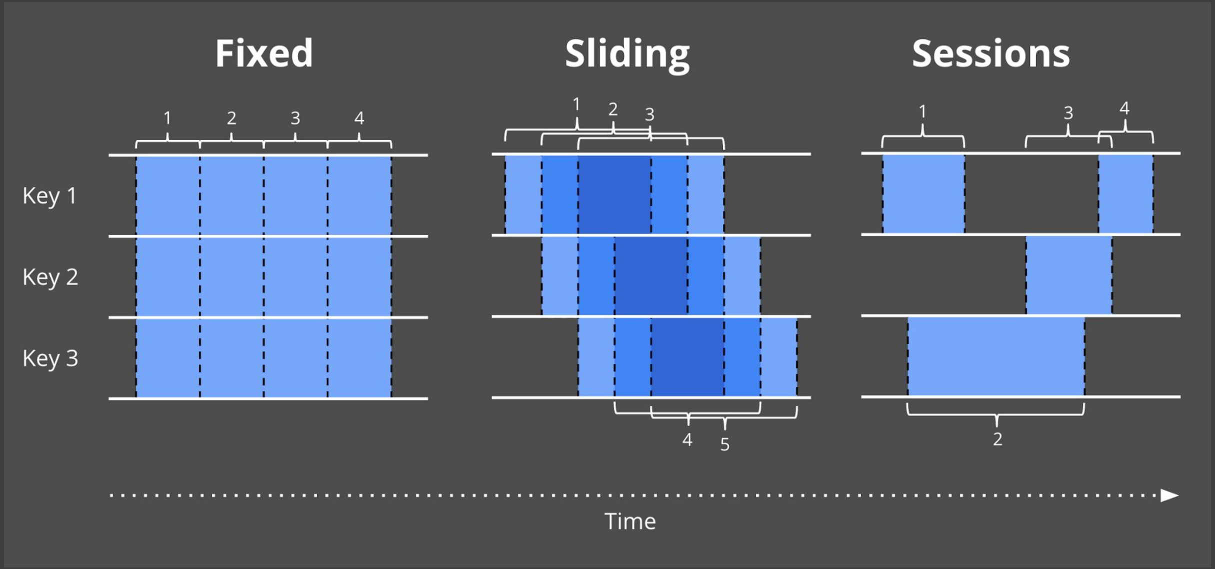


Composite

```
Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
 .apply(FlatMapElements.via(
     word → Arrays.asList(word.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();
```

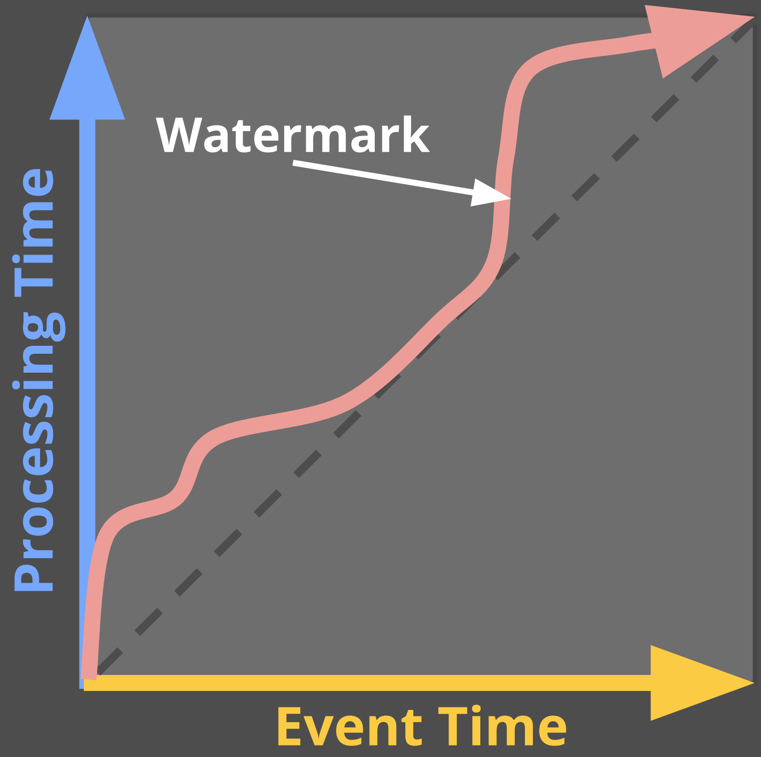
Where - windowing

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



- Required when doing aggregations over unbounded data.

When - triggers



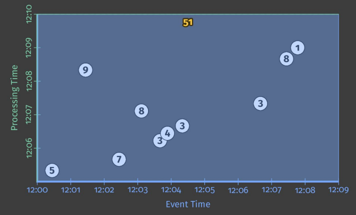
Control when a window emits results of aggregation

Often relative to the **watermark** *(promise about lateness of a source)*

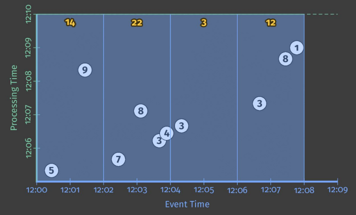
How do refinements relate?

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

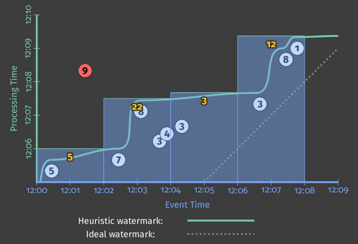
Customizing What Where When How



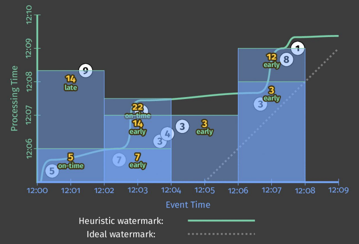
1.Classic Batch



2. Batch with Fixed Windows



3. Streaming



4. Streaming with Speculative + Late Data

3

Apache Beam Project

What is Apache Beam?

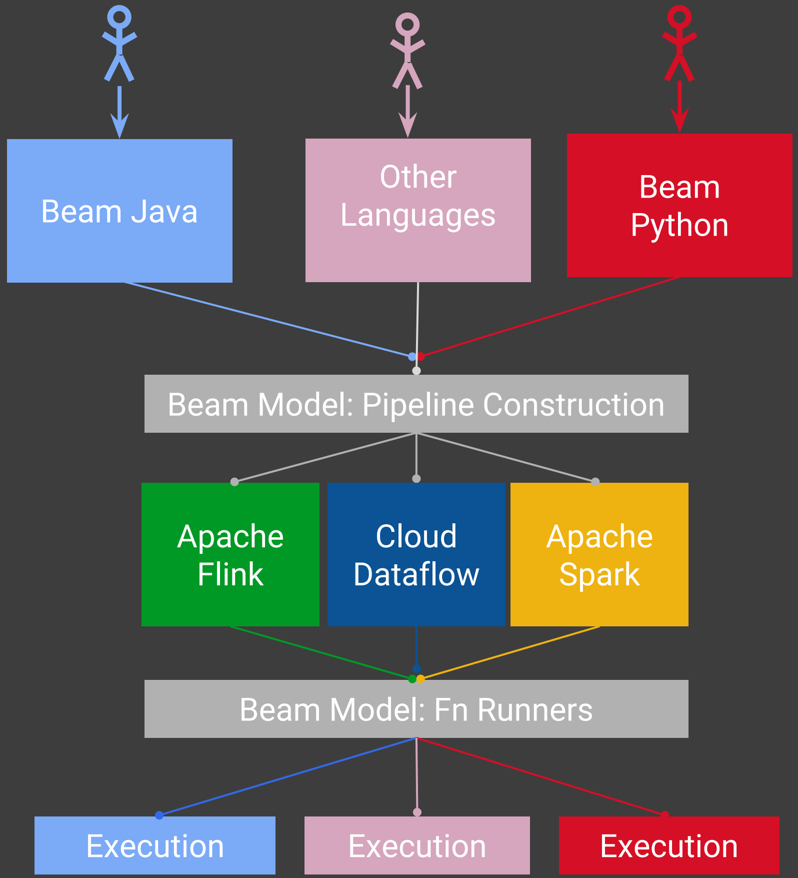
1. The Beam Model: **What** / **Where** / **When** / **How**
2. SDKs for writing Beam pipelines -- Java, Python
3. Runners for Existing Distributed Processing Backends
   - Apache Apex
   - Apache Flink
   - Apache Spark
   - Google Cloud Dataflow
   - *(WIP)* Gearpump and others
   - Local (in-process) runner for testing

The Apache Beam Vision

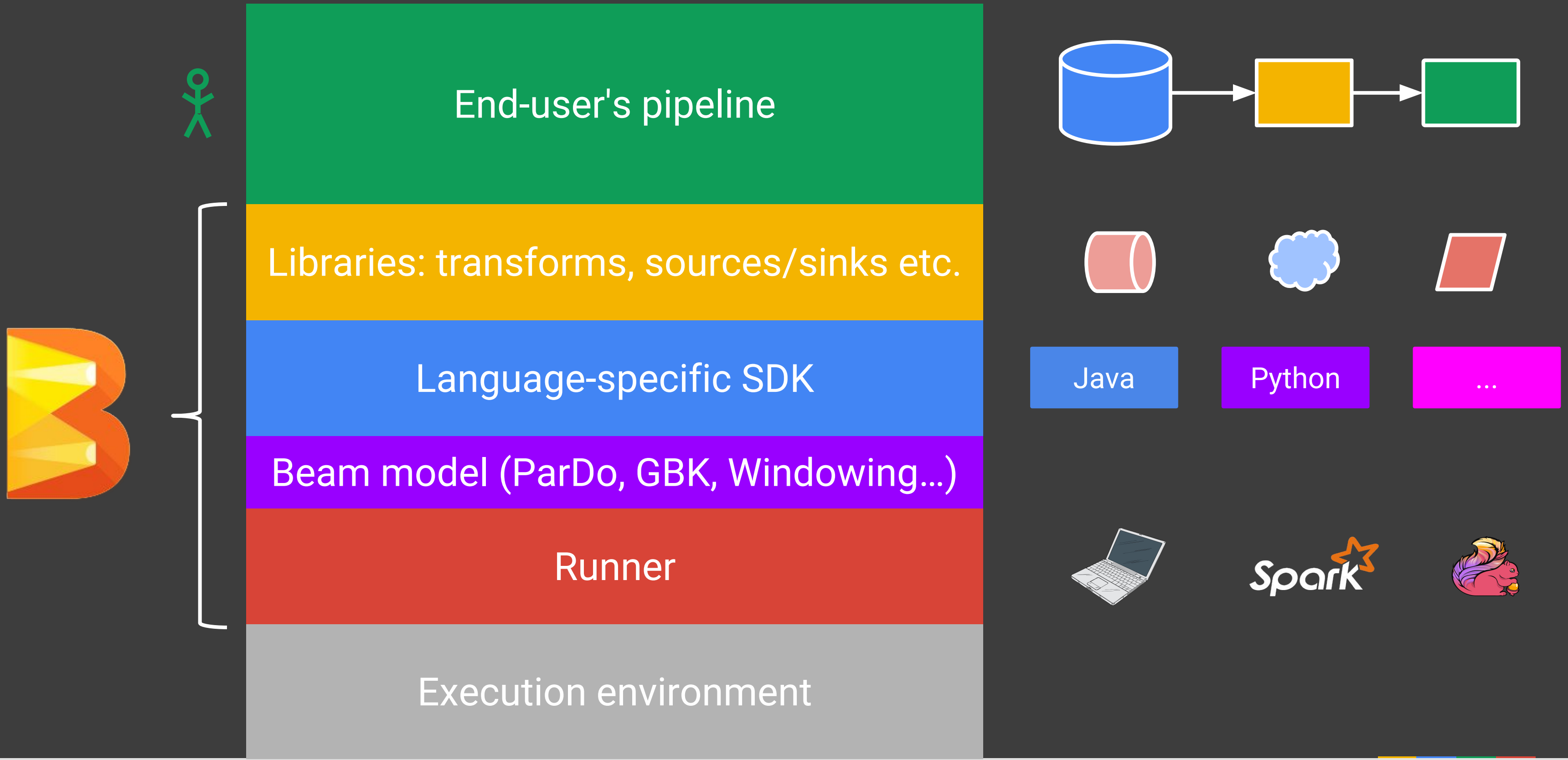
1. **End users:** who want to write pipelines in a language that's familiar.

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



Apache Beam ecosystem



End-user's pipeline

Libraries: transforms, sources/sinks etc.

Language-specific SDK

Beam model (ParDo, GBK, Windowing…)

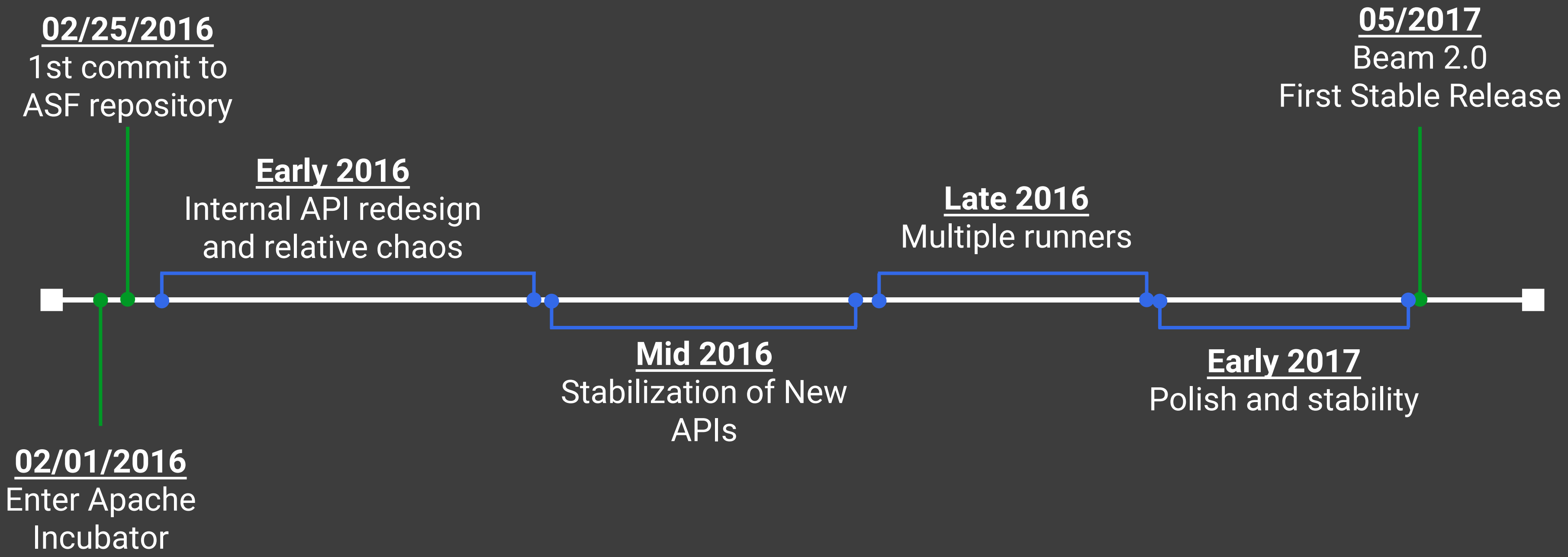
Runner

Execution environment

Java

Python

...



02/01/2016
Enter Apache
Incubator

02/25/2016
1st commit to
ASF repository

Early 2016
Internal API redesign
and relative chaos

Mid 2016
Stabilization of New
APIs

Late 2016
Multiple runners

Early 2017
Polish and stability

05/2017
Beam 2.0
First Stable Release

Apache Beam Community

178 contributors

24 committers from **8 orgs** (none >50%)

>3300 PRs, **>8600** commits, **27** releases

>20 IO (storage system) connectors

5 runners

4

Beam and HBase

Beam IO connector ecosystem

Many uses of Beam = importing data from one place to another

Files	Text, Avro, XML, TFRecord *(pluggable FS - local, HDFS, GCS)*
Hadoop ecosystem	**HBase**, HadoopInputFormat, Hive (HCatalog)
Streaming systems	Kafka, Kinesis, MQTT, JMS, *(WIP)* AMQP
Google Cloud	Pubsub, BigQuery, Datastore, **Bigtable**, Spanner
Other	JDBC, Cassandra, Elasticsearch, MongoDB, GridFS

HBaseIO

```
PCollection<Result> data = p.apply(
    HBaseIO.read()
           .withConfiguration(conf)
           .withTableId(table)
           … withScan, withFilter …)


PCollection<KV<byte[], Iterable<Mutation>>> mutations = …;
mutations.apply(
    HBaseIO.write()
           .withConfiguration(conf))
           .withTableId(table)
```

IO Connectors = just Beam transforms

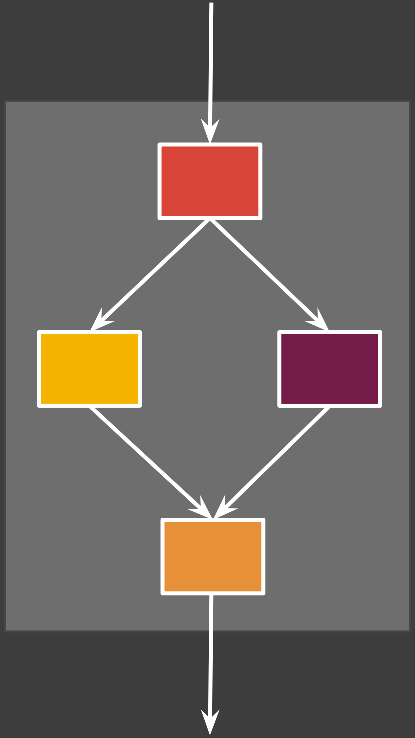
Made of Beam primitives

ParDo, GroupByKey, …

Write = often a simple ParDo

Read = a couple of ParDo,
“Source API” for power users

⇒ straightforward to develop, clean API, very flexible,
batch/streaming agnostic



Beam Write with HBase

A bundle is a group of elements processed and committed together.

APIs (ParDo/DoFn):

```
                setup()            -> Creates Connection
                startBundle()      -> Gets BufferedMutator
                processElement()   -> Applies Mutation(s)
Transaction     finishBundle()     -> BufferedMutator flush
                tearDown()         -> Connection close
```

Mutations must be idempotent, e.g. Put or Delete.
Increment and Append should not be used.

Beam Source API

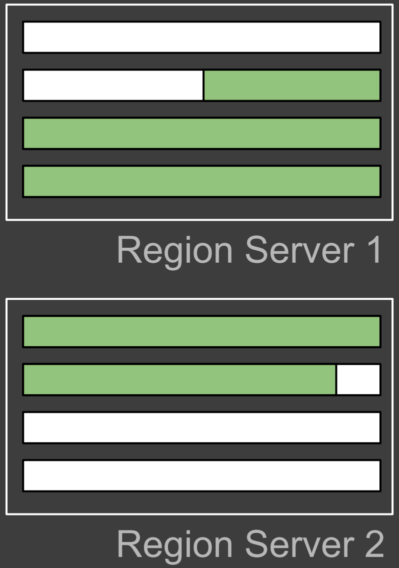
(similar to Hadoop InputFormat, but cleaner / more general)

- **Estimate** size
- **Split** into sub-sources *(of ~given size)*
- **Read**
  - Iterate
  - Get progress
  - Dynamic split

Note: Separate API for unbounded sources + (WIP) a new unified API

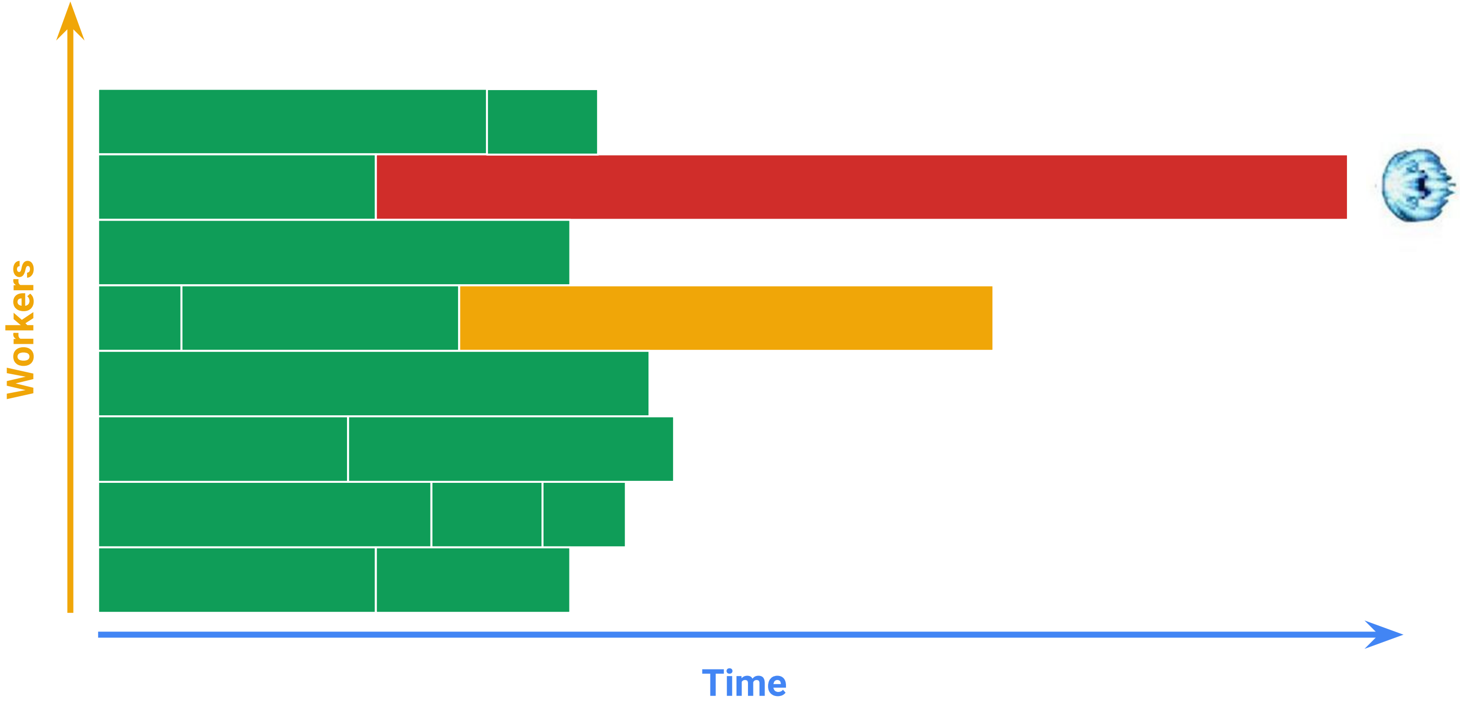
HBase on Beam Source API

HBaseSource	Scan
Estimate	RegionSizeCalculator
Split	RegionLocation
Read	
Iterate	ResultScanner
Get progress	Key interpolation
Dynamic split*	RangeTracker

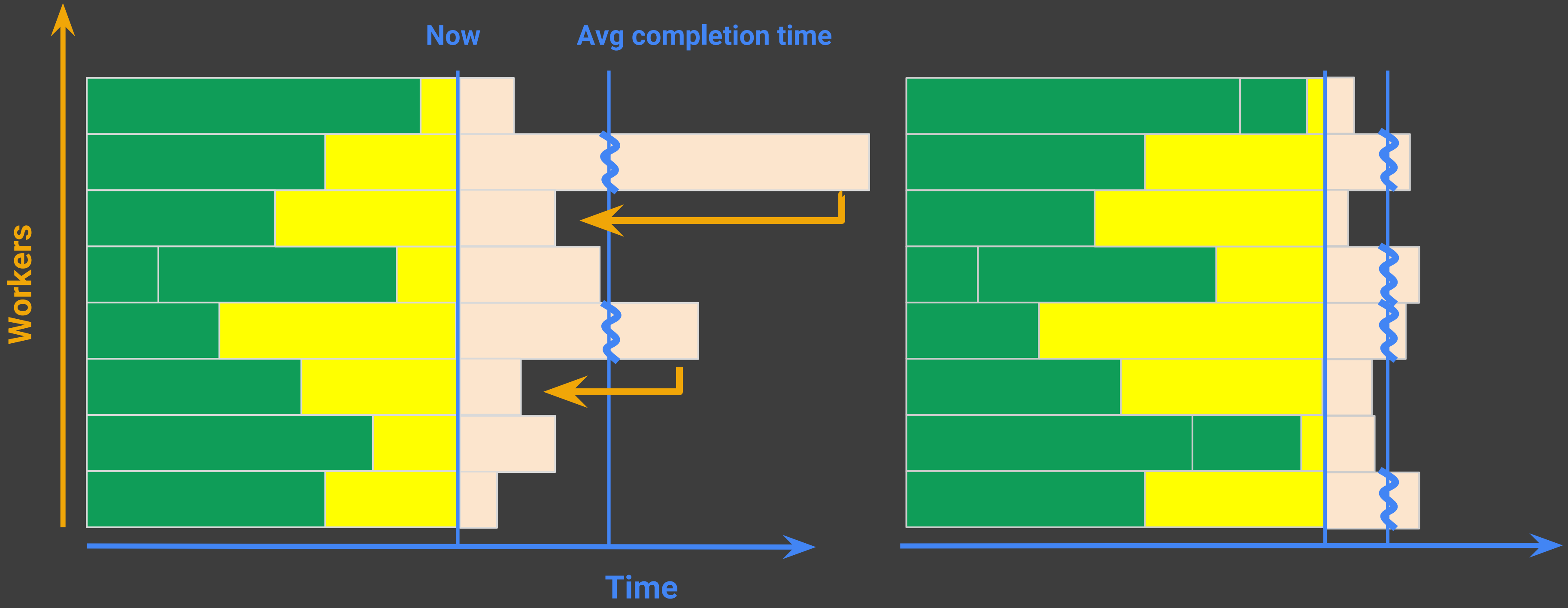


* Dynamic Split for HBaseIO PR in progress

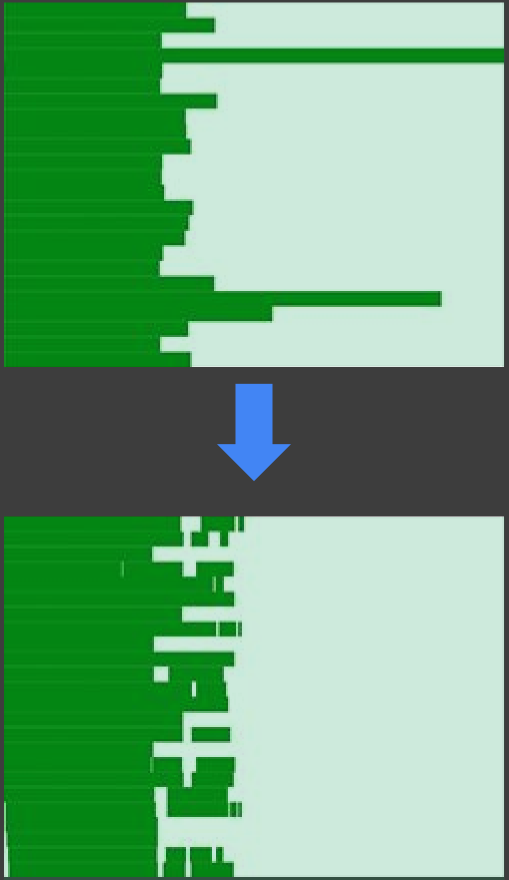
Digression: Stragglers

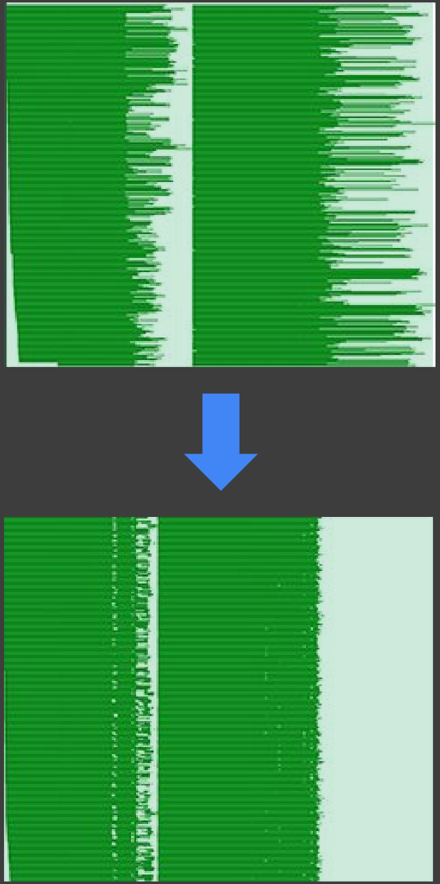


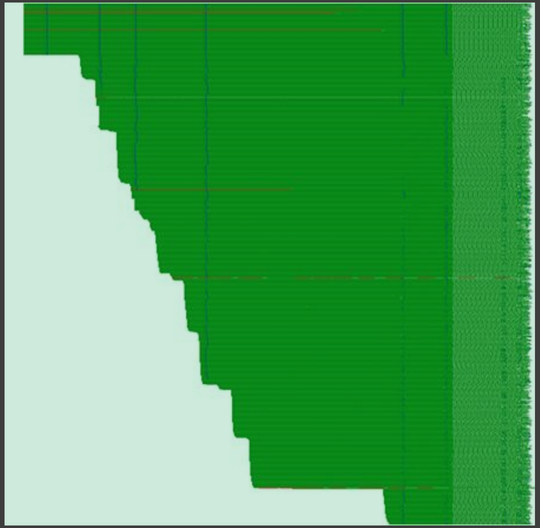
Beam approach: Dynamic splitting*



Currently implemented only by Dataflow







Autoscaling

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

No Shard Left Behind

Straggler Free Data Processing in Cloud Dataflow

Join the mailing lists!

user-subscribe@beam.apache.org
dev-subscribe@beam.apache.org

Follow @ApacheBeam on Twitter

Thank you