Nexmark: Using Apache Beam to create a unified benchmarking suite

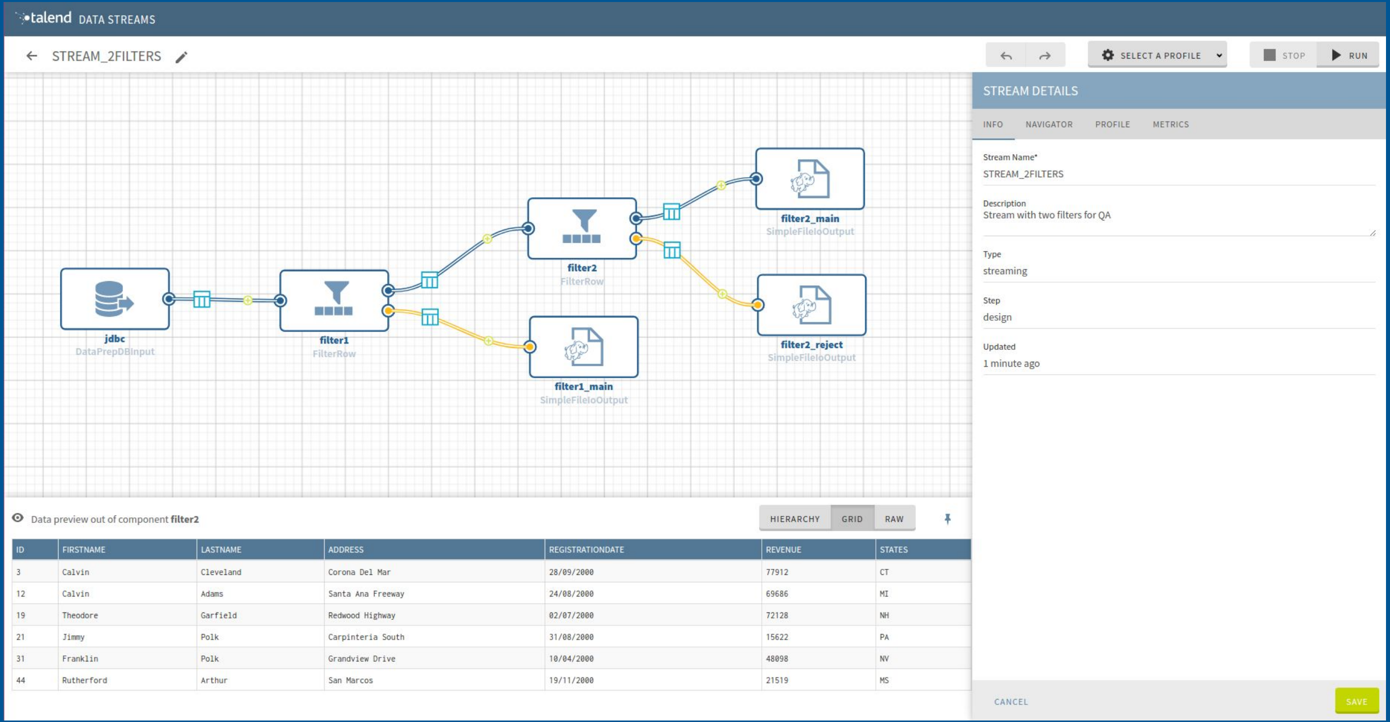
Ismaël Mejía
@iemejia

Who are we?

talend

Integration Software
Big Data / Real-Time
Open Source
Enterprise

New products



STREAM DETAILS

INFO NAVIGATOR PROFILE METRICS

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 !

Agenda

1. **Big Data Benchmarking**
   a. State of the art
   b. NEXMark: A benchmark over continuous data streams
2. **Apache Beam and Nexmark**
   a. Introducing Beam
   b. Advantages of using Beam for benchmarking
   c. Implementation
   d. Nexmark + Beam: a win-win story
3. **Using Nexmark**
   a. Neutral benchmarking: a difficult issue
   b. Example: Running Nexmark on Apache Spark
4. **Current status and future work**

Big Data Benchmarking

Benchmarking

Why do we benchmark?

1. Performance
2. Correctness

Benchmark suites steps:

1. Generate data
2. Compute data
3. Measure performance
4. Validate results

Types of benchmarks

- Microbenchmarks
- Functional
- Business case
- Data Mining / Machine Learning

Issues of Benchmarking Suites for Big Data

- **No de-facto** suite: Terasort, TPCx-HS (Hadoop), HiBench, ...
- No common model/API: Strongly tied to each processing engine or SQL
- Too focused on **Hadoop** infrastructure
- Mixed benchmarks for storage/processing
- Few benchmarking suites focus on **streaming** semantics

State of the art

Batch

- Terasoft: Sort random data
- TPCx-HS: Sort to measure Hadoop compatible distributions
- TPC-DS on Spark: TPC-DS business case with Spark SQL
- Berkeley Big Data Benchmark: SQL-like queries on Hive, Redshift, Impala
- HiBench* and BigBench

Streaming

- Yahoo Streaming Benchmark

* HiBench includes also some streaming / windowing benchmarks

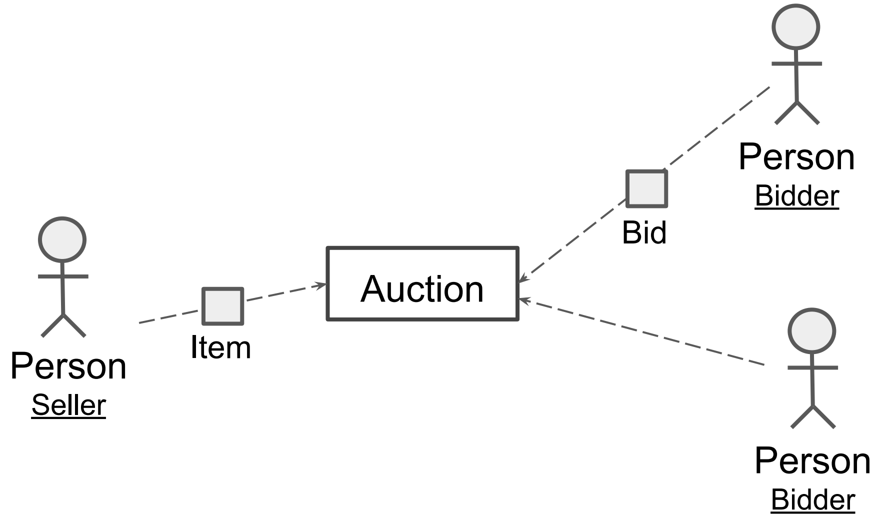
NEXMark

Benchmark for queries over data streams

Online Auction System

Research paper draft 2004

8 CQL-like queries



Example:

Query 4: What is the average selling price for each auction category?

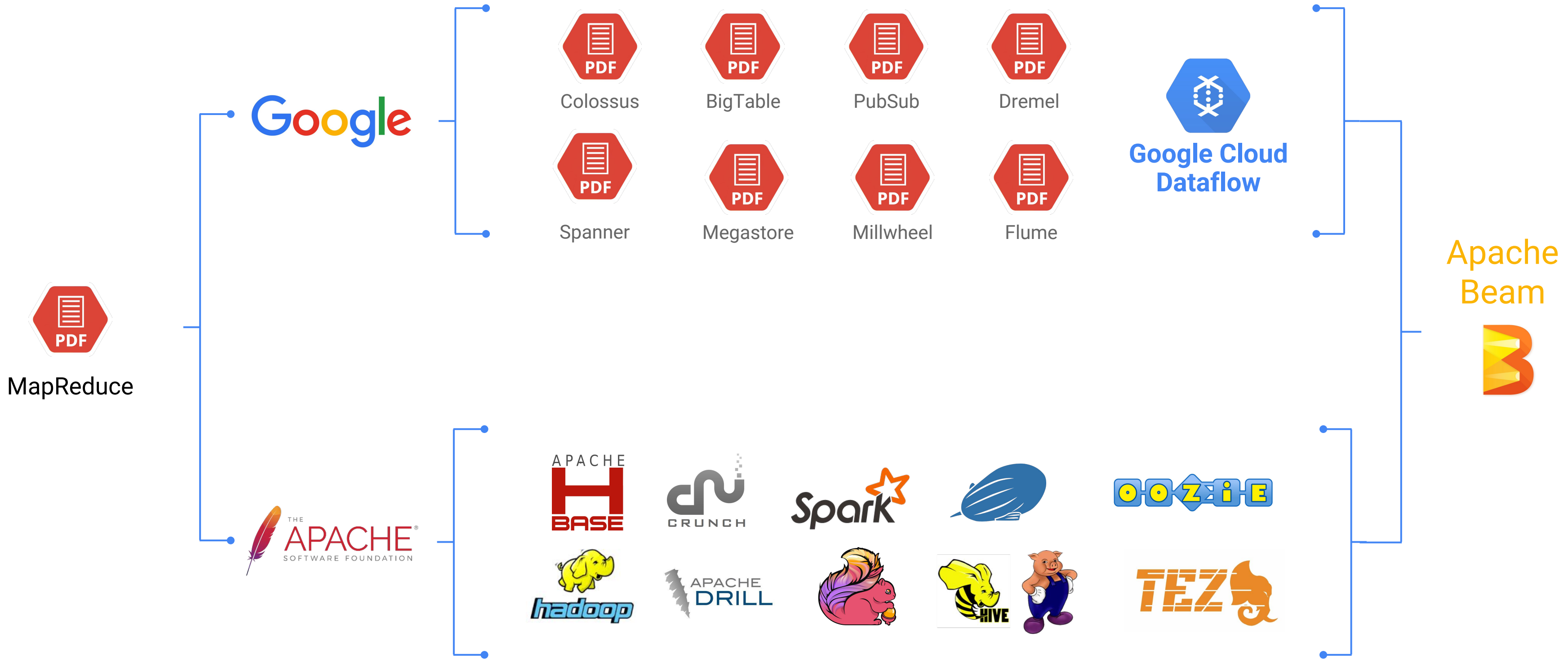
Query 8: Who has entered the system and created an auction in the last period?

Nexmark on Google Dataflow

- Port of the queries from the NEXMark research paper
- Enriched suite with client use cases
- Used as a rich integration test scenario

Apache Beam and Nexmark

Apache Beam origin



What is Apache Beam?

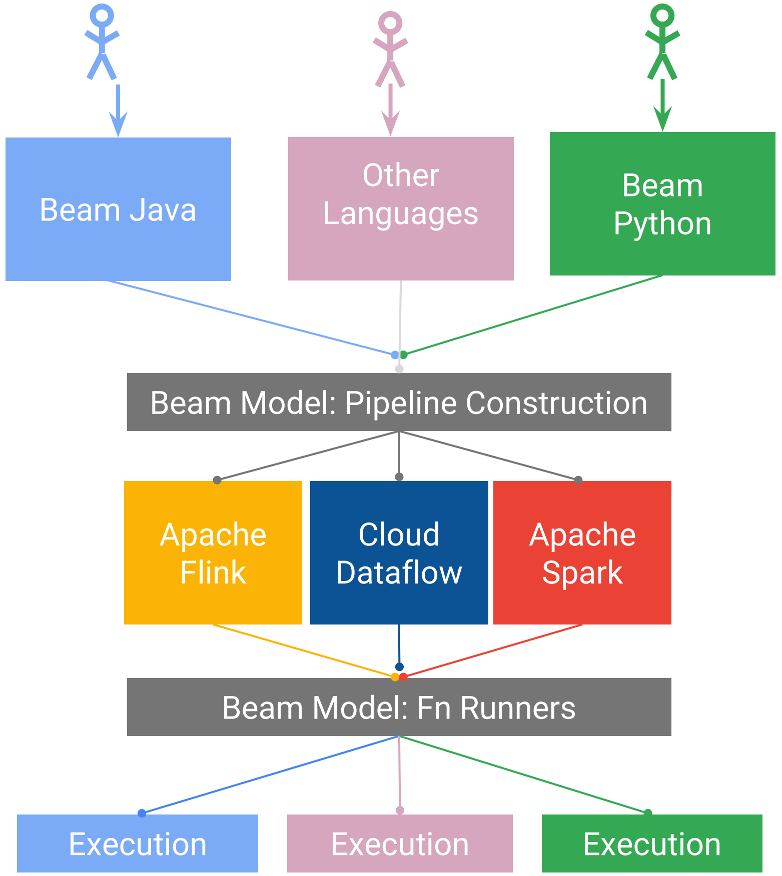
Apache Beam is a **unified** programming model designed to provide **efficient** and **portable** data processing pipelines

Apache Beam vision

Batch + str**EAM** Unified model

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

1. **SDKs**: Java, Python, Go (WIP), etc
2. **DSLs & Libraries:** Scio (Scala), SQL (WIP)
3. **IOs**: Data store Sources / Sinks
4. **Runners** for existing Distributed Processing Engines



Runners

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

Apache Beam Direct Runner

Google Cloud Dataflow

Apache Spark

Apache Flink

Apache Apex

WIP

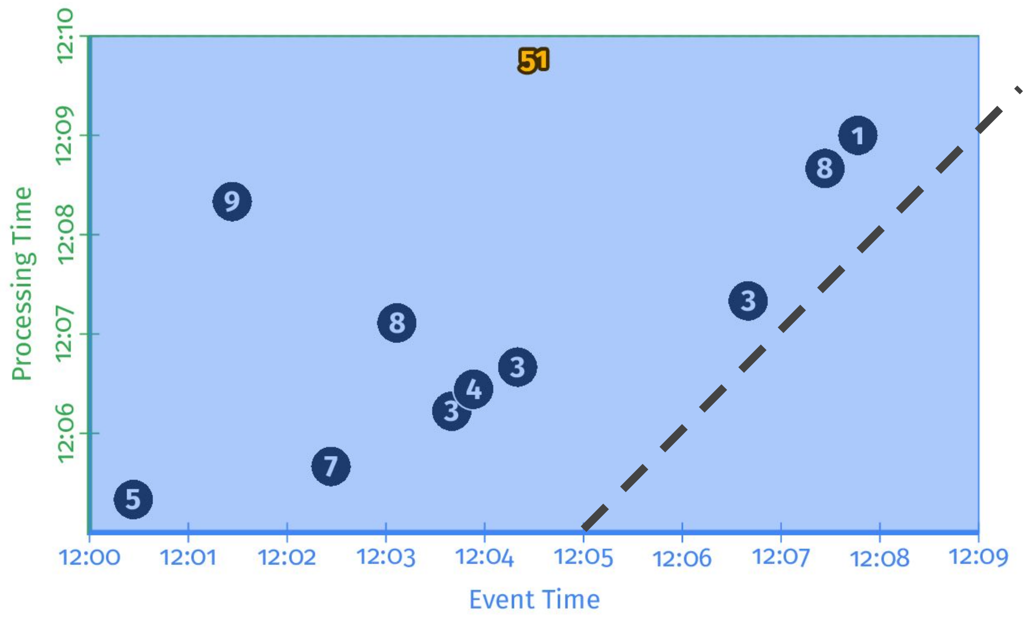
Apache Storm

Apache Gearpump

Ali Baba JStorm

* Same code, different runners & runtimes

The Beam Model: What is Being Computed?

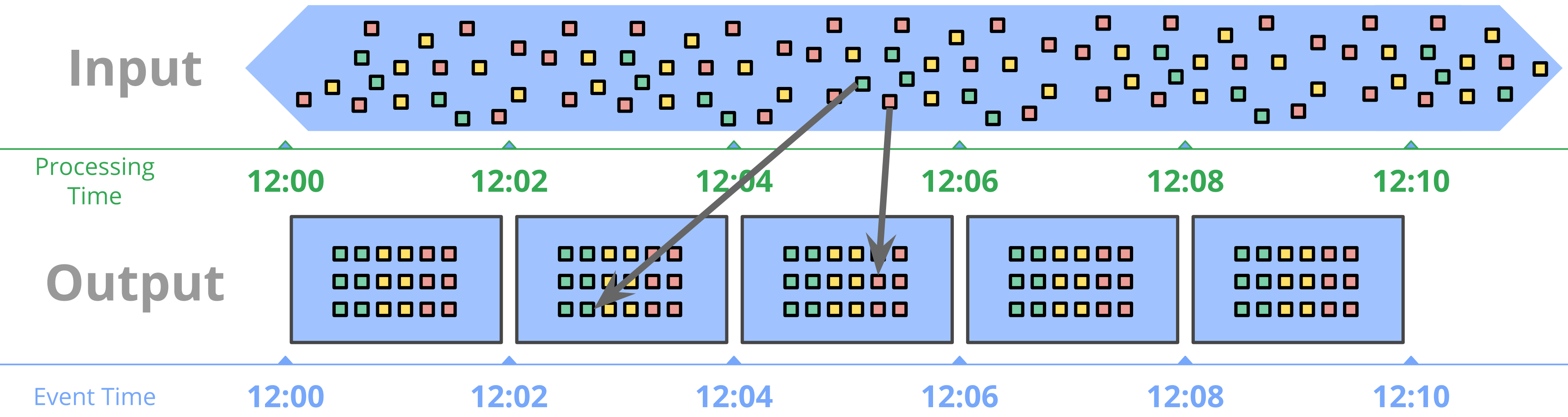


Event Time: Timestamp when the event happened

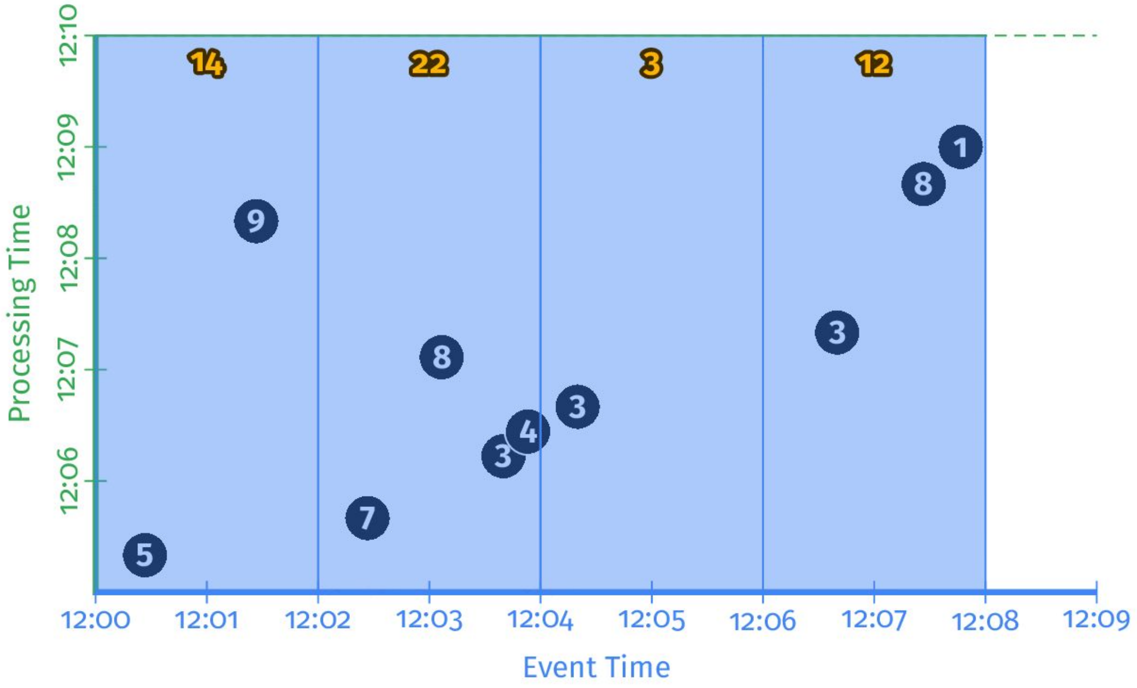
Processing Time: Absolute program time (wall clock)

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

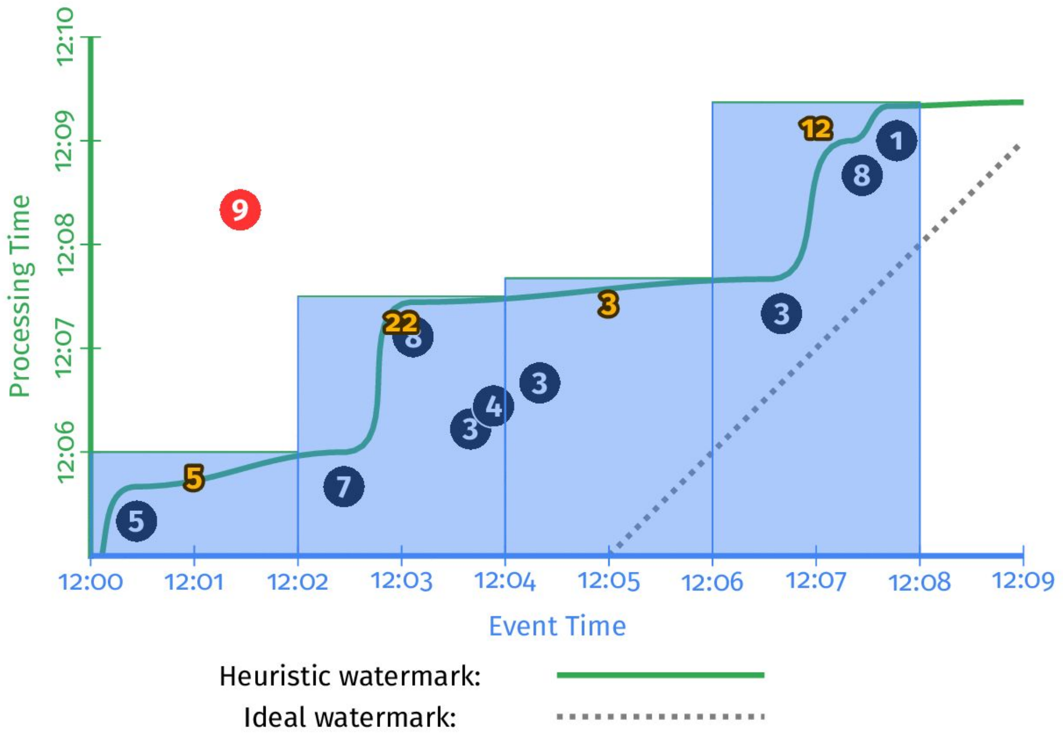
- Split infinite data into finite chunks



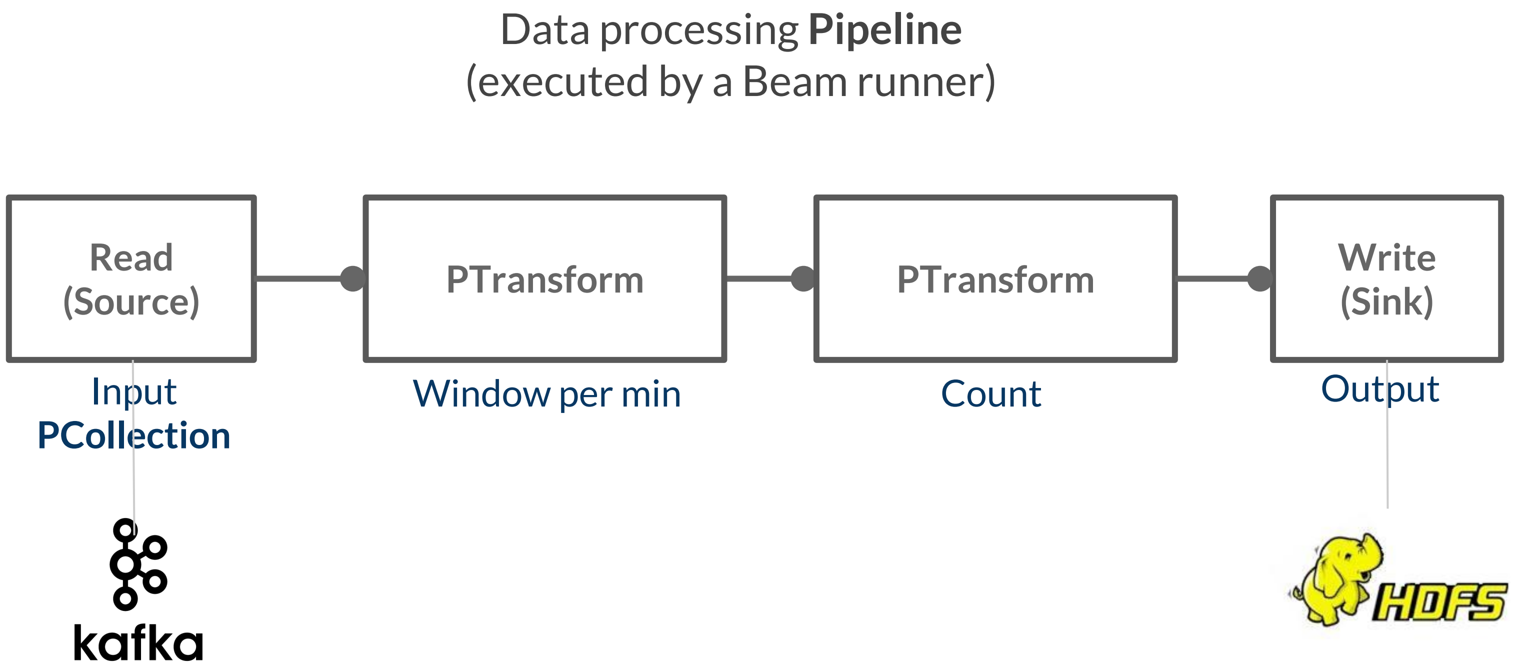
The Beam Model: **Where** in Event Time?



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



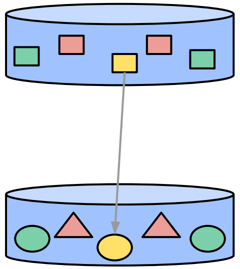
Apache Beam Pipeline concepts



* Don't think it is only a straight pipeline any directed acyclic graph (DAG) is valid.

Apache Beam - Programming Model

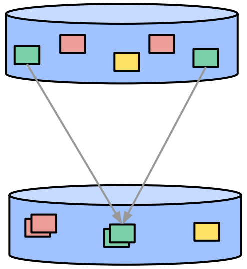
Element-wise



ParDo -> DoFn
MapElements
FlatMapElements
Filter

WithKeys
Keys
Values

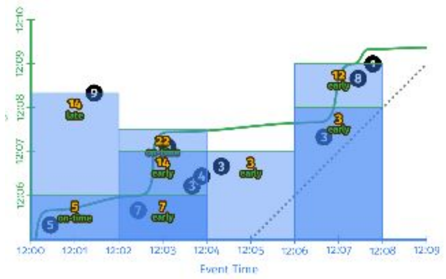
Grouping



GroupByKey
CoGroupByKey

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

Windowing/Triggers



Windows
FixedWindows
GlobalWindows
SlidingWindows
Sessions

Triggers
AfterWatermark
AfterProcessingTime
Repeatedly

Nexmark on Apache Beam

- Nexmark was ported from Dataflow to Beam 0.2.0 as an integration test case
- Refactored to the just released **stable** version of Beam **2.0.0**
- Made code generic to support all the Beam runners
- Changed some queries to use new APIs
- Validated queries in all the runners to test their support of the Beam model

Advantages of using Beam for benchmarking

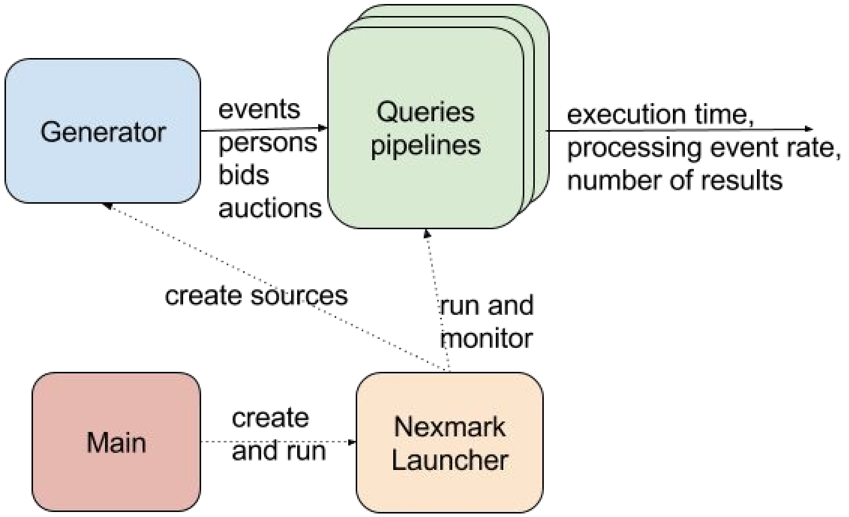
- **Rich model**: all use cases that we had could be expressed using Beam API
- Can test both **batch and streaming** modes with exactly the **same code**
- **Multiple runners**: queries can be executed on Beam supported runners*
- **Metrics**

* Runners must provide the specific capabilities (features) used by the query

Implementation

Components of Nexmark

- **NexmarkLauncher**:
  Start sources to generate Events
  Run and monitor the queries (pipelines)
- **Generator**:
  Timestamped and correlated events:
  Auction, Bid, Person
- **Metrics**:
  Each query includes ParDos to update metrics:
  execution time, processing event rate, number of results,
  but also invalid auctions/bids, …
- **Configuration***:
  Batch: test data is finite and uses a BoundedSource
  Streaming: test data is finite but uses an UnboundedSource



* Configuration details discussed later

Interesting Queries

Query	Description	Beam concepts
3	Who is selling in particular US states?	Join, State, Timer
5	Which auctions have seen the most bids in the last period?	Sliding Window, Combiners
6	What is the average selling price per seller for their last 10 closed auctions?	Global Window, Custom Combiner
7	What are the highest bids per period?	Fixed Windows, Side Input
9 *	What are the winning bids for each closed auction?	Custom Window
11 *	How many bids did a user make in each session he was active?	Session Window, Triggering
12 *	How many bids does a user make within a fixed processing time limit?	Global Window in Processing Time

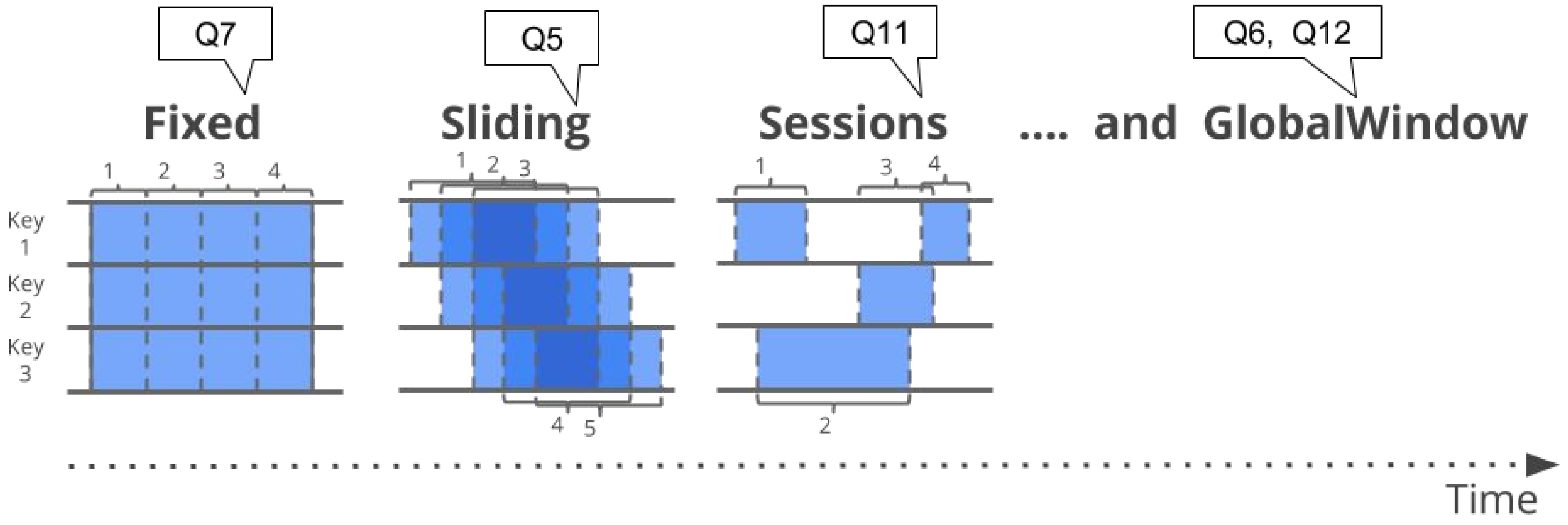
*: not in the original NEXMark paper

Query Structure

1. Get **PCollection<Event>** as input
2. Apply **ParDo** + **Filter** to extract object of interest: Bids, Auction, Person
3. Apply transforms: **Filter, Count, GroupByKey, Window,** etc.
4. Apply **ParDo** to output the final PCollection: collection of AuctionPrice, AuctionCount ...

Key point: Where in time to compute data?

- **Windows**: divide data into event-time-based finite chunks.
  - Often required when doing aggregations over unbounded data

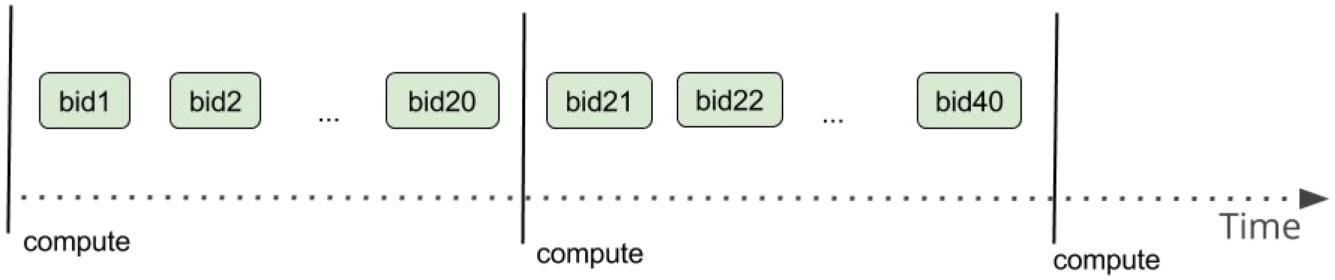


Key point: **When** to compute data?

Triggers: Condition to emit the results of aggregation

Deal with producing early results or including late-arriving data

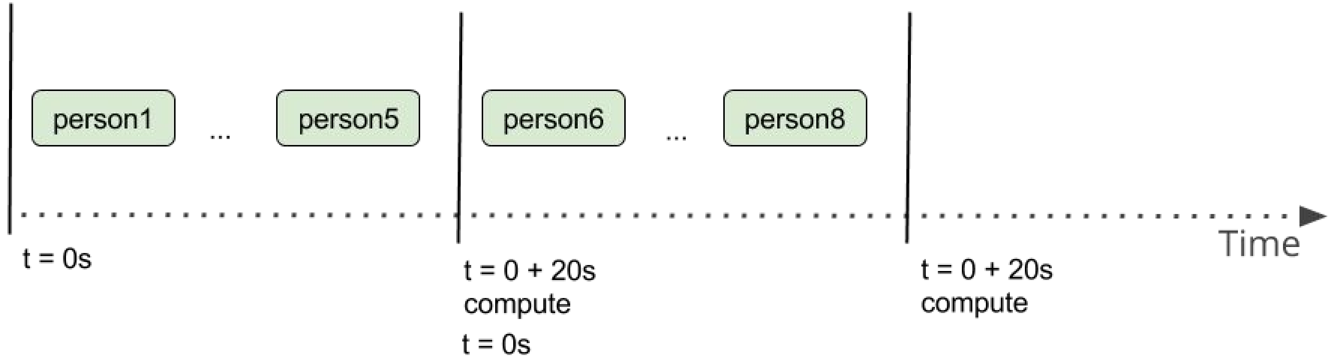
- Q11: uses a data-driven trigger fires when 20 elements were received



* Triggers can be Event-time, Processing-Time, Data-driven or Composite

Key point: When to compute data?

- Q12: Processing-time trigger fired when first element is received + delay (works in processing in global window time to create a duration)

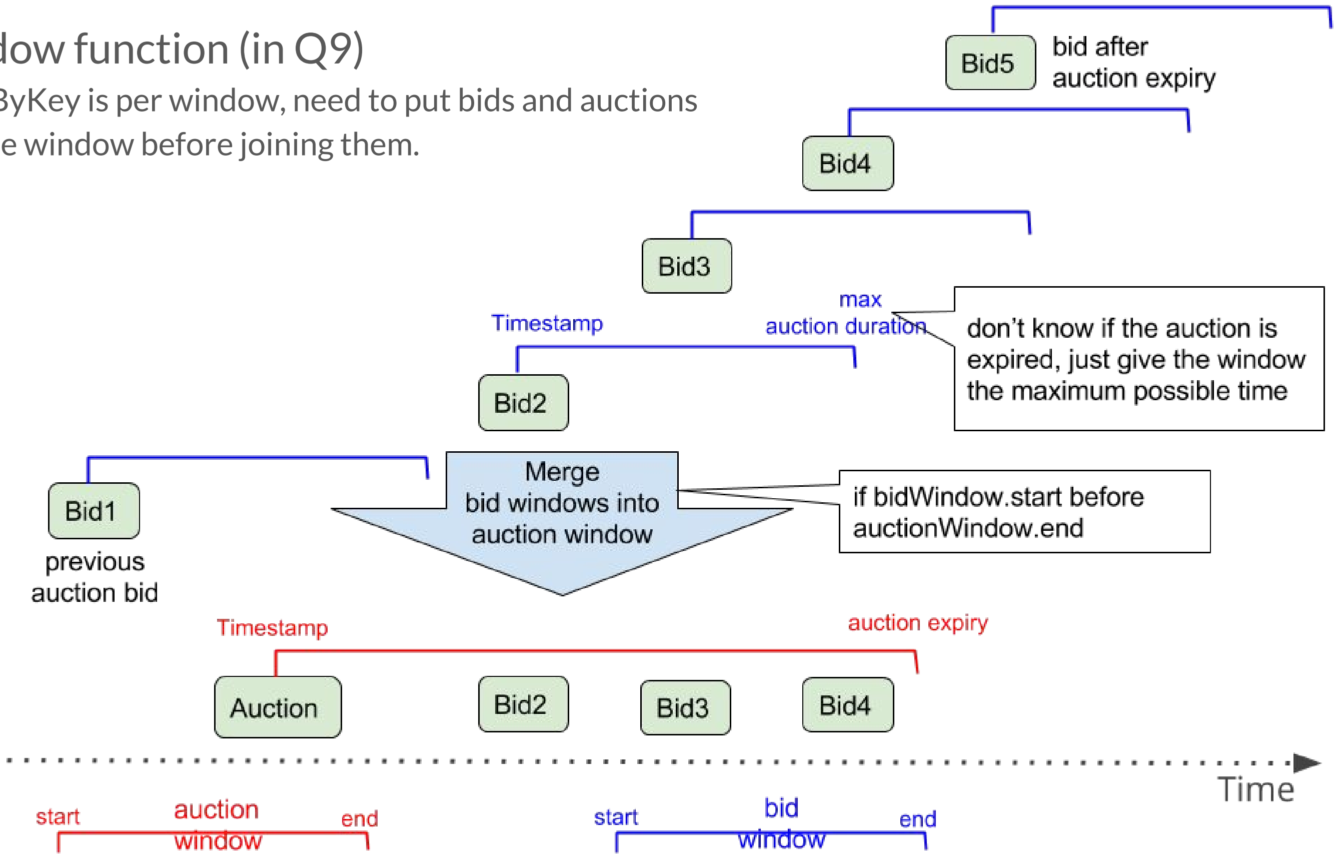


- **Processing time**: wall clock absolute program time
- **Event time**: timestamp in which the event occurred

Default trigger: at the end of the window (Event-time)

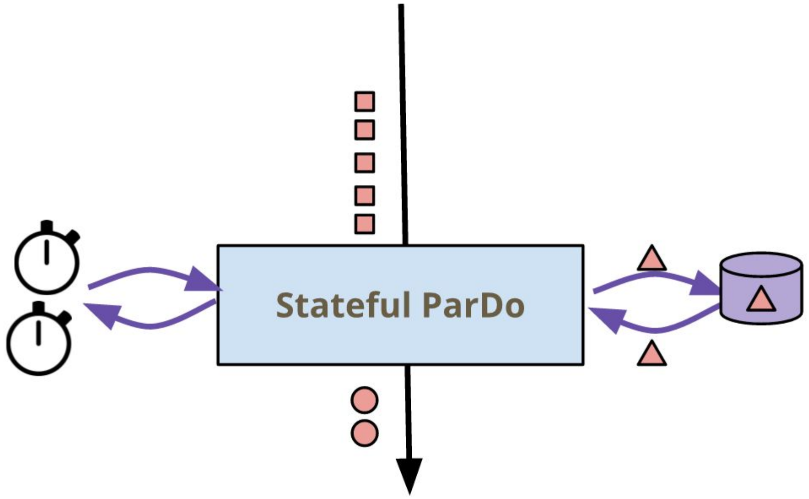
Key point: How to **temporarily group** events?

- Custom window function (in Q9)
  - CoGroupByKey is per window, need to put bids and auctions in the same window before joining them.



Key point: How to deal with **out of order** events?

- State and Timer APIs in an incremental join (Q3):
    - Memorize person event waiting for corresponding auctions and clear at timer
    - Memorize auction events waiting for corresponding person event



Conclusion on queries

- Wide coverage of the Beam API
  - Most of the API
  - Illustrates also working in processing time
- Realistic
  - Real use cases, valid queries for an end user auction system
- Complex queries
  - Leverage all the runners capabilities

Why Nexmark on Beam? A win-win story

- Advanced streaming semantics
- A/B testing of execution engines (e.g. regression and performance comparison between 2 versions of the same engine or of the same runner, ...)
- Integration tests (SDK with runners, runners with engines, …)
- Validate Beam runners capability matrix

Using Nexmark

Neutral Benchmarking: A difficult issue

- **Different levels of support** of capabilities of the Beam model among runners
- All execution systems have **different strengths**: we would end up comparing things that are not always comparable
    - Some runners were designed to be batch oriented, others stream oriented
    - Some are designed towards sub-second latency, others prioritize auto-scaling
- Runners / Systems can have multiple **knobs to tweak the options**
- Benchmarking on a **distributed environment** can be inconsistent.
  Even worse if you benchmark on the cloud (e.g. Noisy neighbors)

Nexmark - How to run

```
$ mvn exec:java -Dexec.mainClass=org.apache.beam.integration.nexmark.Main -Pflink-runner
-Dexec.args="--runner=FlinkRunner --suite=SMOKE --streaming=true --manageResources=false
--monitorJobs=true --flinkMaster=tbd-bench"
```

```
$ mvn exec:java -Dexec.mainClass=org.apache.beam.integration.nexmark.Main -Pspark-runner
-Dexec.args="--runner=SparkRunner --suite=SMOKE --streaming=false
--manageResources=false --monitorJobs=true --sparkMaster=local"
```

```
$ spark-submit --master yarn-client --class org.apache.beam.integration.nexmark.Main
--driver-memory 512m --executor-memory 512m --executor-cores 1
/home/imejia/beam-integration-java-nexmark-bundled-2.1.0-SNAPSHOT.jar
--runner=SparkRunner --query=5 --streamTimeout=60 --streaming=true
```

Benchmark workload configuration

Events generation
smoke config defaults

- 100 000 events generated
- 100 generator threads
- Event rate in SIN curve
- Initial event rate of 10 000
- Event rate step of 10 000
- 100 concurrent auctions
- 1000 concurrent persons bidding / creating auctions

Windows

- size 10s
- sliding period 5s
- watermark hold for 0s

Proportions:

- Hot Auctions = ½
- Hot Bidders =¼
- Hot Sellers=¼

Technical

- Artificial CPU load
- Artificial IO load

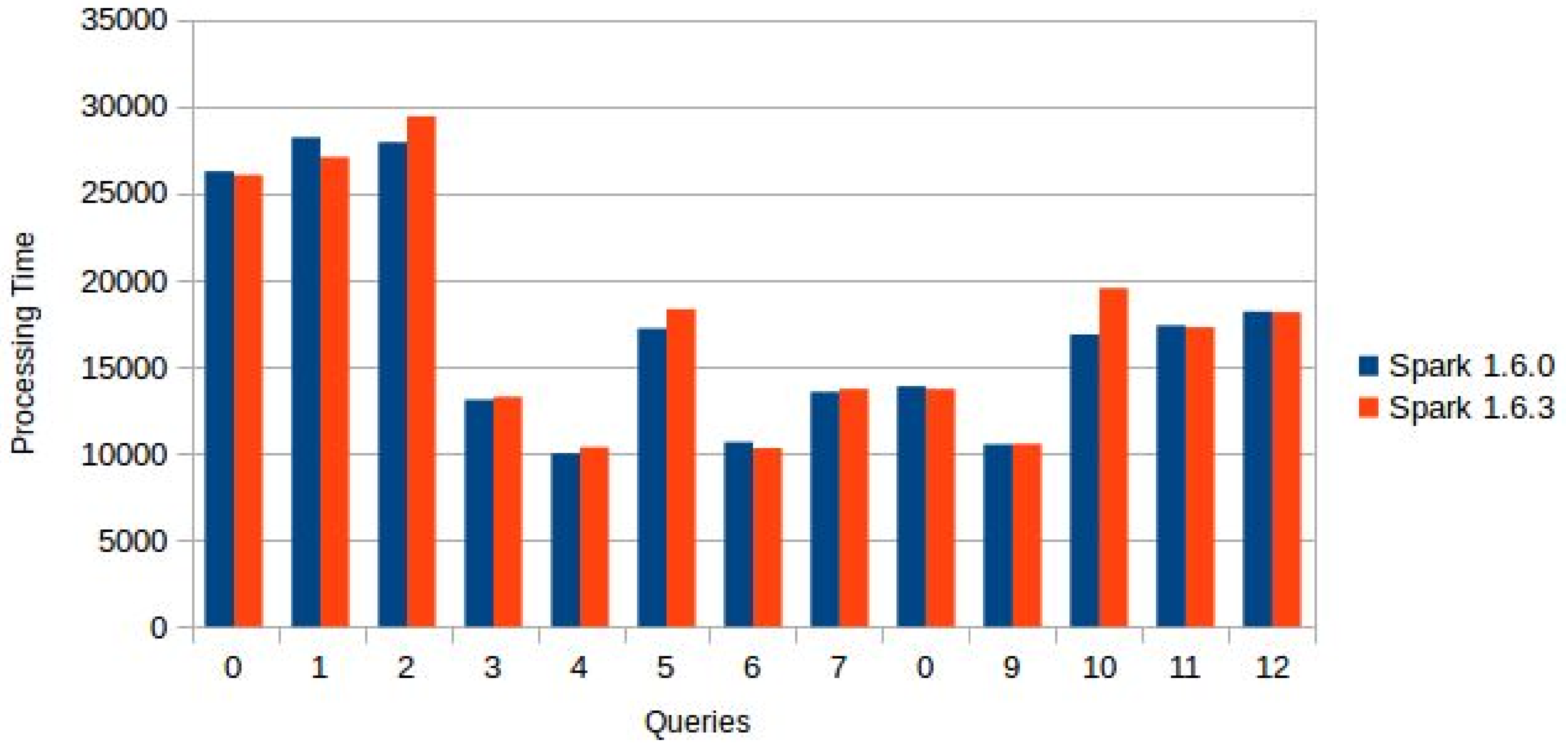
Nexmark Output - Spark Runner (Batch)

```
Conf  Runtime(sec)                Events(/sec)                      Results
0000           3.8                     26267.4                       100000
0001           3.5                     28232.6                        92000
0002           3.6                     27964.2                          713
0003           7.5                     13253.8                          580
0004          10.0                     10006.0                           50
0005           5.8                     17214.7                            3
0006           9.4                     10642.8                         1631
0007           7.4                     13539.1                            1
0000           7.2                     13861.9                         6000
0009           9.5                     10517.5                         5243
0010           5.9                     16877.6                            1
0011           5.8                     17388.3                         1992
0012           5.5                     18181.8                         1992
```

Nexmark Output - Spark Runner (Streaming)

```
Conf  Runtime(sec)                Events(/sec)                 Results
0000           1.0                     10256.1                  100000
0001           1.3                      7722.1                   92000
0002           0.7                     14705.8                     713
0003             0                         0.0                       0
0004          17.3                      5779.7                      50
0005          16.6                      6020.8                       3
0006          26.5                      3773.4                    1631
0007             0                         0.0                       0
0008          12.3                      8142.0                    6000
0009          17.7                      5650.0                    5243
0010          13.1                       768.8                       1
0011          10.0                      9962.1                    1992
0012          10.2                      9783.8                    1992
```

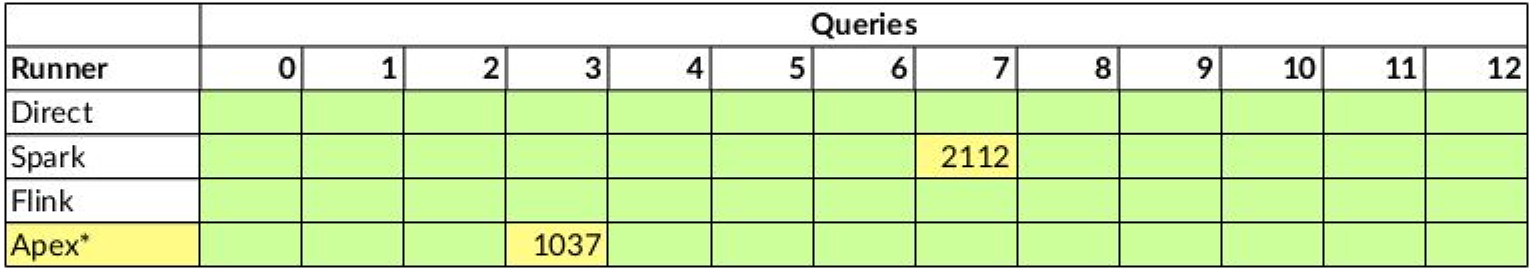
Comparing different versions of the Spark engine



Current status and future work

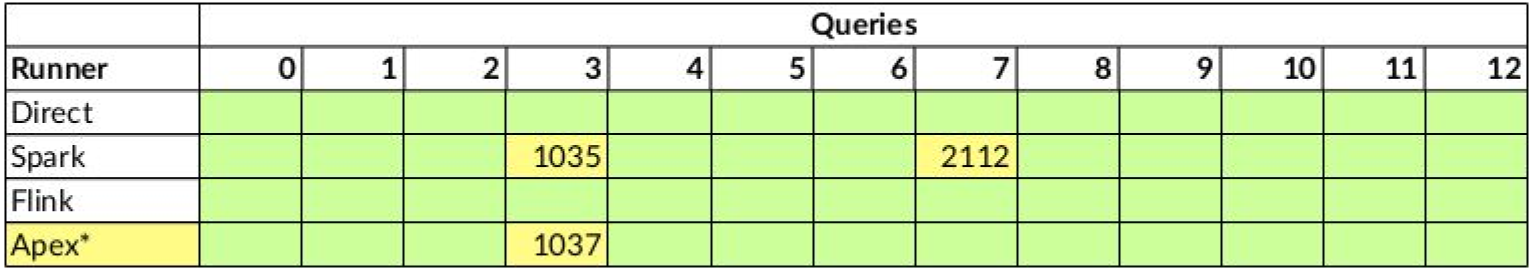
Execution Matrix

Batch



	Queries												
Runner	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Direct	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Spark	🟩	🟩	🟩	🟩	🟩	🟩	🟩	2112	🟩	🟩	🟩	🟩	🟩
Flink	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Apex*	🟩	🟩	🟩	1037	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

Streaming

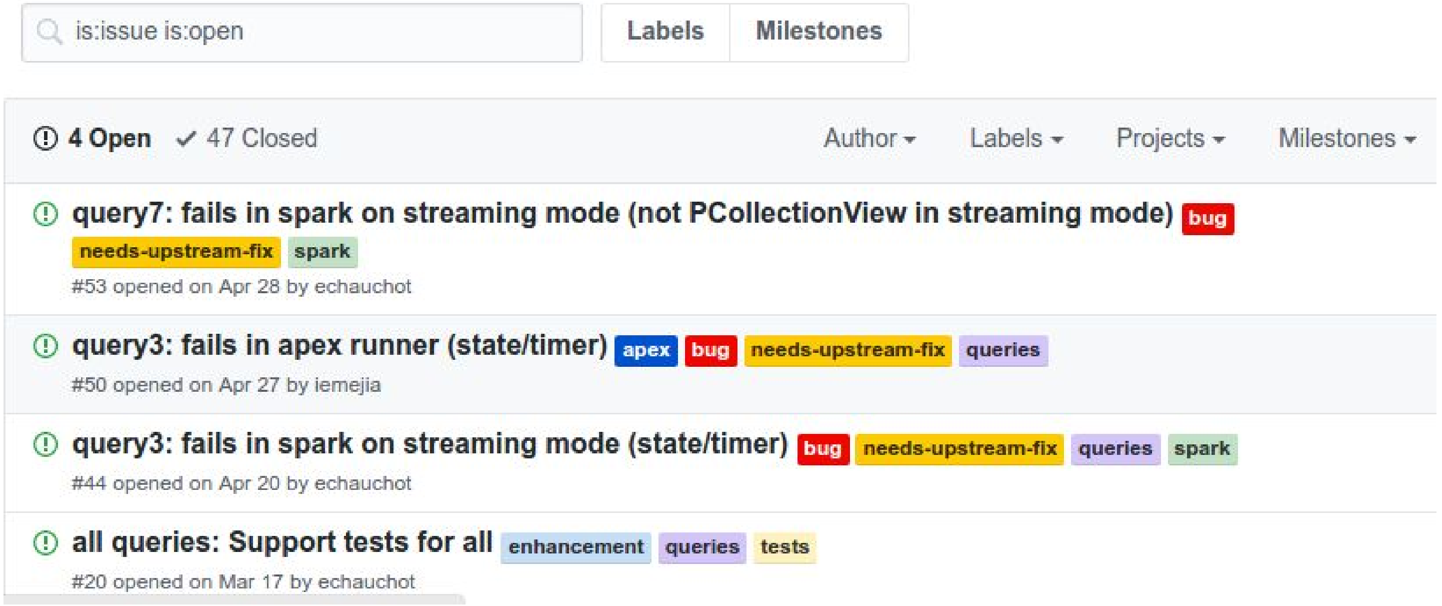


	Queries												
Runner	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Direct	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Spark	🟩	🟩	🟩	1035	🟩	🟩	🟩	2112	🟩	🟩	🟩	🟩	🟩
Flink	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Apex*	🟩	🟩	🟩	1037	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩

* Apex runner lacks support for metrics

· · We have not tested yet on Google Dataflow

Current status



4 Open ✓ 47 Closed

query7: fails in spark on streaming mode (not PCollectionView in streaming mode) bug needs-upstream-fix spark
#53 opened on Apr 28 by echauchot

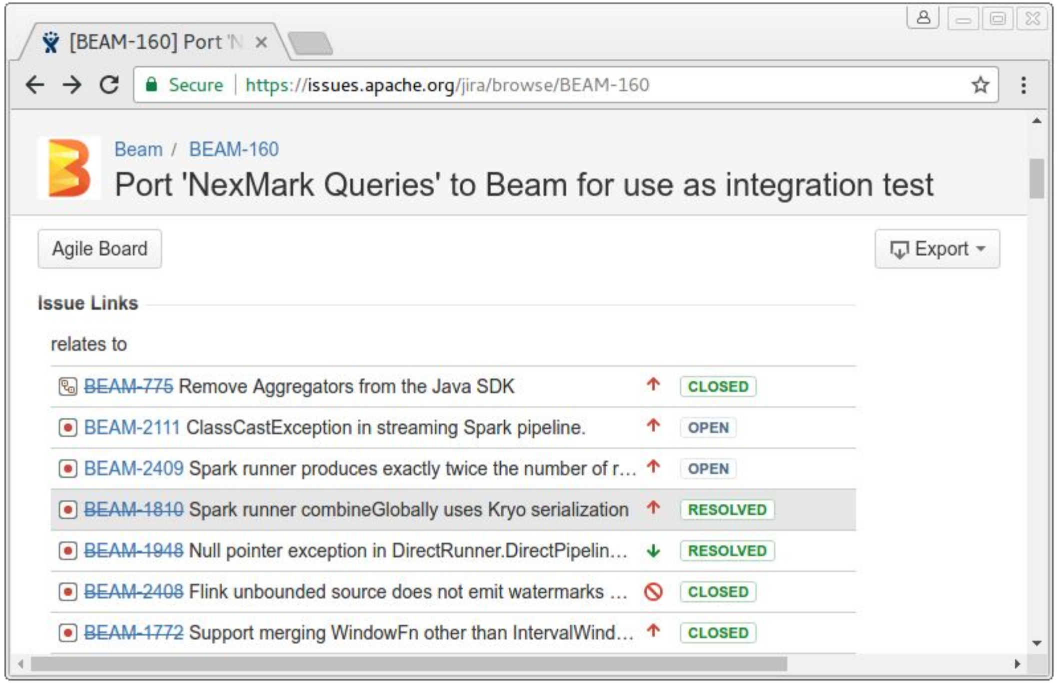
query3: fails in apex runner (state/timer) apex bug needs-upstream-fix queries
#50 opened on Apr 27 by iemejia

query3: fails in spark on streaming mode (state/timer) bug needs-upstream-fix queries spark
#44 opened on Apr 20 by echauchot

all queries: Support tests for all enhancement queries tests
#20 opened on Mar 17 by echauchot

- Manage Nexmark issues in a dedicated place.
- Pending issues will be migrated to upstream

Current status


[BEAM-160] Port 'N ×

Secure | https://issues.apache.org/jira/browse/BEAM-160

Beam / BEAM-160

Port 'NexMark Queries' to Beam for use as integration test

Agile Board

Export

Issue Links

relates to

~~BEAM-775~~ Remove Aggregators from the Java SDK	↑	CLOSED
BEAM-2111 ClassCastException in streaming Spark pipeline.	↑	OPEN
BEAM-2409 Spark runner produces exactly twice the number of r…	↑	OPEN
~~BEAM-1810~~ Spark runner combineGlobally uses Kryo serialization	↑	RESOLVED
~~BEAM-1948~~ Null pointer exception in DirectRunner.DirectPipelin…	↓	RESOLVED
~~BEAM-2408~~ Flink unbounded source does not emit watermarks …	🚫	CLOSED
~~BEAM-1772~~ Support merging WindowFn other than IntervalWind…	↑	CLOSED

- Nexmark helped discover bugs and missing features in Beam
- 10 open issues / 7 closed issues on Beam upstream. BEAM-160
- Nexmark PR is reviewed, and LGTM It must be merged into master for **Beam 2.1.0**

Future work

- Resolve open Nexmark and Beam issues
- Integrate Nexmark into the Integration tests of Beam
- Add more queries to evaluate corner cases
- Validate new runners: Gearpump, Storm, JStorm
- Streaming SQL-based queries (using the ongoing work on Calcite DSL)

Contribute

You are welcome to contribute!

- 5 open Github issues and 9 Beam Jiras that need to be taken care of
- Improve documentation + more refactoring
- New ideas, more queries, support for IOs, etc

Not only for Nexmark, **Beam** is in a perfect shape to jump in.

Greetings

- Mark Shields (Google): Contributing Nexmark + answering our questions
- Etienne Chauchot (Talend): Co-maintainer of Nexmark
- Thomas Groh, Kenneth Knowles (Google): Direct runner + State/Timer API
- Amit Sela, Aviem Zur (Paypal): Spark Runner + Metrics
- Aljoscha Krettek (data Artisans), Jinsong Lee (Ali Baba): Flink Runner
- Jean-Baptiste Onofre, Abbass Marouni (Talend): comments and help to run Nexmark in our YARN cluster
- The rest of the Beam community in general for being awesome.

* The nice slides with animations were created by Tyler Akidau and Frances Perry and used with authorization.

References

Apache Beam
NEXMark
BEAM-160
Nexmark on Beam Issues
Big Data Benchmarks

Thanks