Relational Beam: Process columns, not rows!

By Andrew Pilloud, Brian Hulette
https://s.apache.org/beam-relational-2022

BEAM SUMMIT

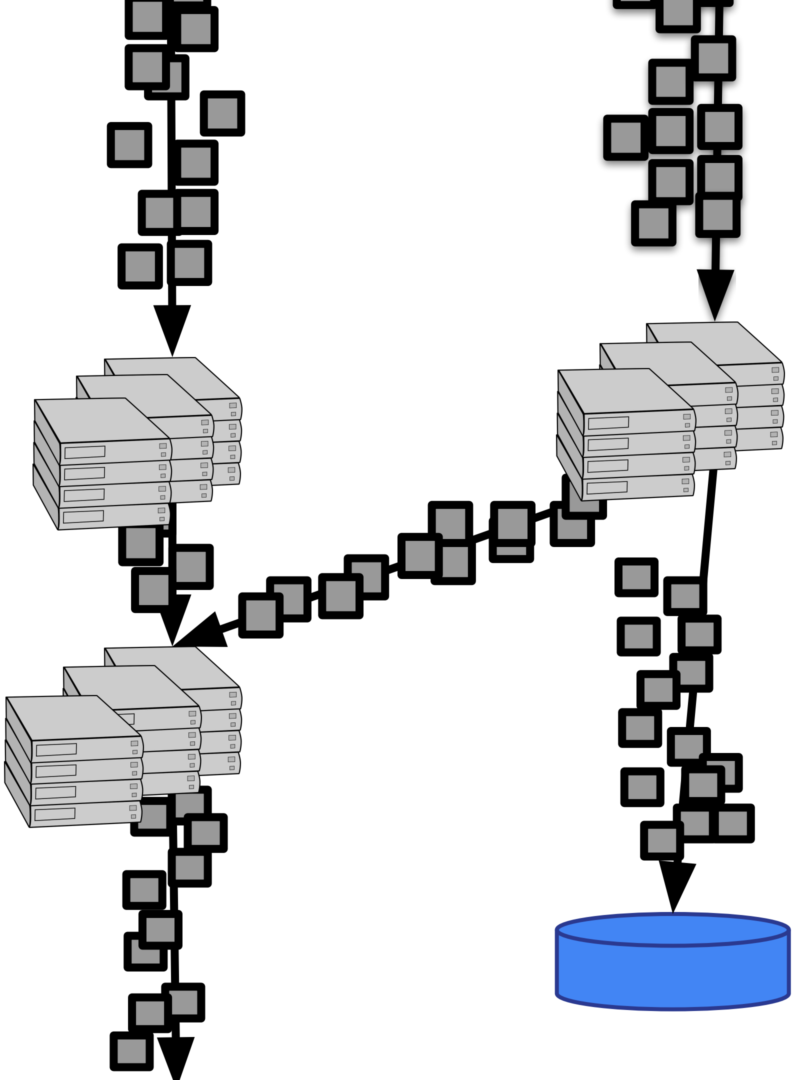
Austin, 2022

Agenda

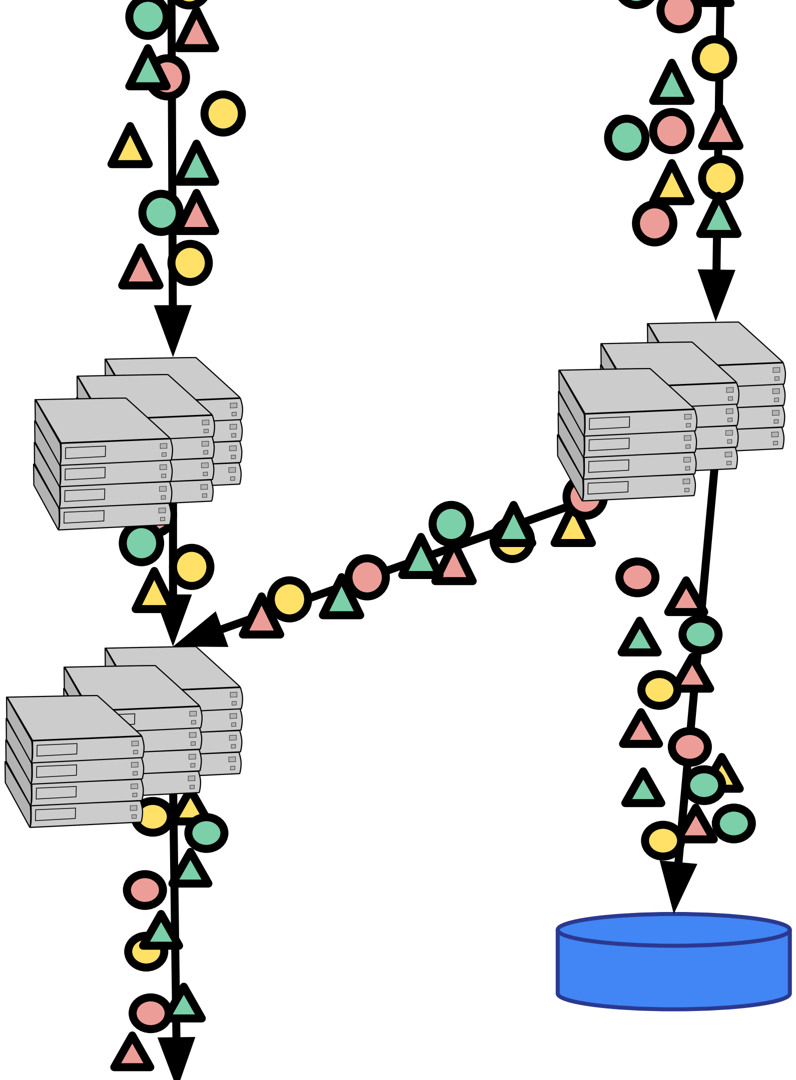
- Relational?
- Practical Relational Beam
    - Towards Columnar and Vectorization in the Python SDK
    - Demo! Java Projection Pushdown
- Best Practices

Relational?

Beam is not Relational



Your data is Relational



Why should we make Beam Relational?

- It’s good for Beam developers
  - Improved runner and language interoperability
  - Allows for new classes of optimizations
- It’s good for Beam users
  - Simpler APIs more accurately capturing user intent
  - Better performance

What do we need?

- Beam has Structured Coders, but they aren’t enough.
  - We need metadata about your data!

Beam Schema and Row enables Relational

- Beam Schemas expose the structure of your data

```
Schema.builder()
    .addInt64Field("foo").addInt32Field("baz").build();
```

- Beam Row provides an abstraction for programmatic data access

```
public abstract class Row {
  <T> @Nullable T getValue(int fieldIdx);
  <T> @Nullable T getValue(String fieldName);
}
```

What else do we need?

- Beam has a graph of PCollections, but that won’t do.
    - We need metadata about your computations!

Beam needs a Row Expression

- Calcite calls this a RexNode
  - `SELECT <row>` and `WHERE <bool>`from SQL
- Three Required Operators
  - Field Access (FieldAccessDescriptor)
  - Constant (Schema Value)
  - Call (Arbitrary function call, the difficult one)

DoFns can provide Relational metadata

- Basic Relational DoFns use Row (or a Schema type)

  processElement(@Element Row row, …) {}

- More advanced DoFns provide metadata about access

  processElement(@FieldAccess(“col1”) int col1,

  @FieldAccess(“col2”) int col2, …) {}

- Or eventually vectorized execution

  int mapElement(@FieldAccess(“col1”) int[] col1, …) {…}

  processBatch(@FieldAccess(“col1”) int[] col1, …) {…}

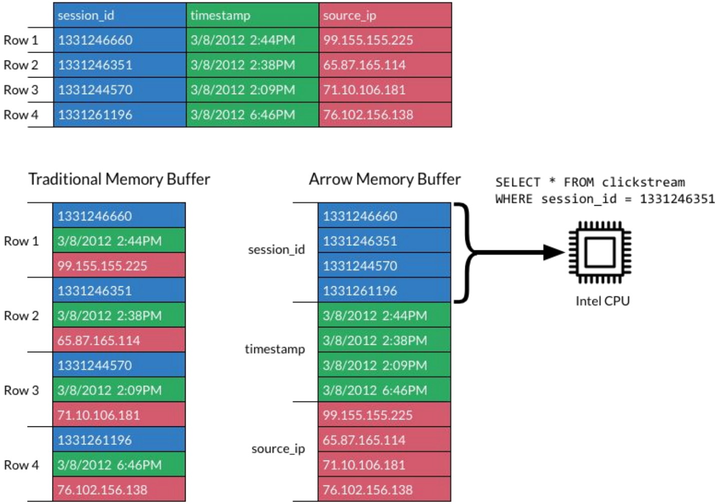
We need your help!

- Cross language? Relational for max interoperability!
- IOs? Relational to minimize copies and transforms!
- New SDK? Make it Relational by default!
- Python type troubles? Put Relational on it!
- Go KVs? Relational can make them disappear!
- Make it Relational with Schemas and RowCoder

Practical Relational Beam

Towards Columnar and Vectorization in the Python SDK

What is Columnar?



	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

Traditional Memory Buffer

Row 1	1331246660 3/8/2012 2:44PM 99.155.155.225
Row 2	1331246351 3/8/2012 2:38PM 65.87.165.114
Row 3	1331244570 3/8/2012 2:09PM 71.10.106.181
Row 4	1331261196 3/8/2012 6:46PM 76.102.156.138

Arrow Memory Buffer

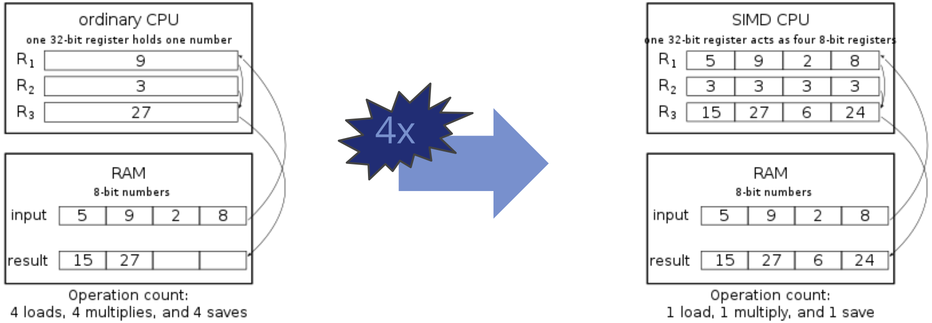
session_id	1331246660 1331246351 1331244570 1331261196
timestamp	3/8/2012 2:44PM 3/8/2012 2:38PM 3/8/2012 2:09PM 3/8/2012 6:46PM
source_ip	99.155.155.225 65.87.165.114 71.10.106.181 76.102.156.138

```
SELECT * FROM clickstream
WHERE session_id = 1331246351
```

(Image from https://arrow.apache.org/overview/)

That seems complicated, why bother?

Vectorization!



ordinary CPU

one 32-bit register holds one number

R_1 9

R_2 3

R_3 27

RAM

8-bit numbers

input 5 9 2 8

result 15 27

Operation count:
4 loads, 4 multiplies, and 4 saves

4x

SIMD CPU

one 32-bit register acts as four 8-bit registers

R_1 5 9 2 8

R_2 3 3 3 3

R_3 15 27 6 24

RAM

8-bit numbers

input 5 9 2 8

result 15 27 6 24

Operation count:
1 load, 1 multiply, and 1 save

(Images from https://en.wikipedia.org/wiki/Single_instruction,_multiple_data)

Many Python libraries are already vectorized!

pandas
TensorFlow

…but they require batches

```
   # Create batch
pc | beam.BatchElements(..)
   | beam.Map(lambda batch: np.array(batch))
   | beam.Map(lambda arr: arr*2)
   # Explode batch
   | beam.FlatMap(lambda arr: arr)
```

Enter Batched DoFns

```
class MyDoFn(DoFn):
  def process(self, element: np.int64) -> np.int64:
    yield element * 2
```

```
class MyVectorizedDoFn(DoFn):
  def process_batch(self, batch: NumpyArray[np.int64]) -> NumpyArray[np.int64]:
    yield batch * 2
```

https://s.apache.org/batched-dofns

Interoperating with element-wise DoFns

```
class MyVectorizedDoFn(DoFn):
  # element-wise fallback
  def process(self, element: np.int64) -> np.int64:
    yield element * 2

  def process_batch(self, batch: NumpyArray[np.int64]) -> NumpyArray[np.int64]:
    yield batch * 2
```

Most batch types in Python are ambiguous!

```
class MyVectorizedDoFn(DoFn):
  def process(self, element: np.int64) -> np.int64:
    yield element * 2

  def process_batch(self, batch: np.ndarray) -> np.ndarray:
    yield batch * 2
```

Batches of Schema'd Data

```
class MyVectorizedColumnarDoFn(DoFn):
  # MyRowType has an inferred schema
  def process(self, element: MyRowType) -> MyRowType:
    yield ...

  def process_batch(self, batch: pd.DataFrame) -> pd.DataFrame:
    yield ...
```

Batches of Schema'd Data

```
class MyVectorizedColumnarDoFn(DoFn):
  # MyRowType has an inferred schema
  def process(self, element: MyRowType) -> MyRowType:
    yield ...

  def process_batch(self, batch: pa.RecordBatch) -> pa.RecordBatch:
    yield ...
```

Timestamps and Windowing

```
class MyWindowingDoFn(DoFn):
  def process_batch(self, batch: np.ndarray,
                     timestamp=beam.DoFn.TimestampParam) -> np.ndarray:
    ...
    yield HomogeneousWindowedBatch(..., timestamp=..., window=...)
```

Timestamps and Windowing

```
class MyWindowingDoFn(DoFn):
  def process_batch(self, batch: np.ndarray,
                    timestamps=beam.DoFn.TimestampBatchParam) -> np.ndarray:
    ...
    yield HeterogeneousWindowedBatch(..., timestamps=...)
```

❗ This was proposed in https://s.apache.org/batched-dofns, but **does not exist yet**.

What's next?

Use Batched DoFns for:

- Beam DataFrame API
  - PCollection ↔ DataFrame conversion
  - Windowing with `df.rolling` ([#20911](#20911))
- IOs (e.g. `ParquetIO`)
- `RunInference` on structured data
- ⚡ Auto-vectorize [`beam.Select`](#) (e.g. with [`numba.vectorize`](#) or [`jax.vmap`](#))

Demo!
Java Projection Pushdown

We're going to run a test!

```
@Test
public void testBigQueryStorageReadProjectionPushdown() throws Exception {
  Pipeline p = Pipeline.create(options);
  PCollection<Long> count =
      p.apply(
              BigQueryIO.read(
                      record -> BigQueryUtils.toBeamRow(...)
                  .from(options.getInputTable())
                  .withMethod(Method.DIRECT_READ)))
          .apply(ParDo.of(new GetIntField()))
          .apply(Count.globally());
  PAssert.thatSingleton(count).isEqualTo(options.getNumRecords());
  p.run().waitUntilFinish();
}
```

This ParDo won’t do pushdown.

```
private static class GetIntField extends DoFn<Row, Long> {
   @ProcessElement
   public void processElement(ProcessContext context) {
     c.output(c.element().getValue("int_field"));
   }
 }
```

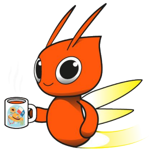
This ParDo provides metadata!

```
private static class GetIntField extends DoFn<Row, Long> {
   @FieldAccess("row")
   private final FieldAccessDescriptor fieldAccessDescriptor =
       FieldAccessDescriptor.withFieldNames("int_field");

   @ProcessElement
   public void processElement(@FieldAccess("row") Row row,
                              OutputReceiver<Long> outputReceiver) {
     outputReceiver.output(row.getValue("int_field"));
   }
 }
```

This is simple, provides metadata.

```
private static class GetIntField extends DoFn<Row, Long> {

   @ProcessElement
   public void processElement(@FieldAccess("int_field") int int_field,
                              OutputReceiver<Long> outputReceiver) {
     outputReceiver.output(int_field);
   }
 }
```

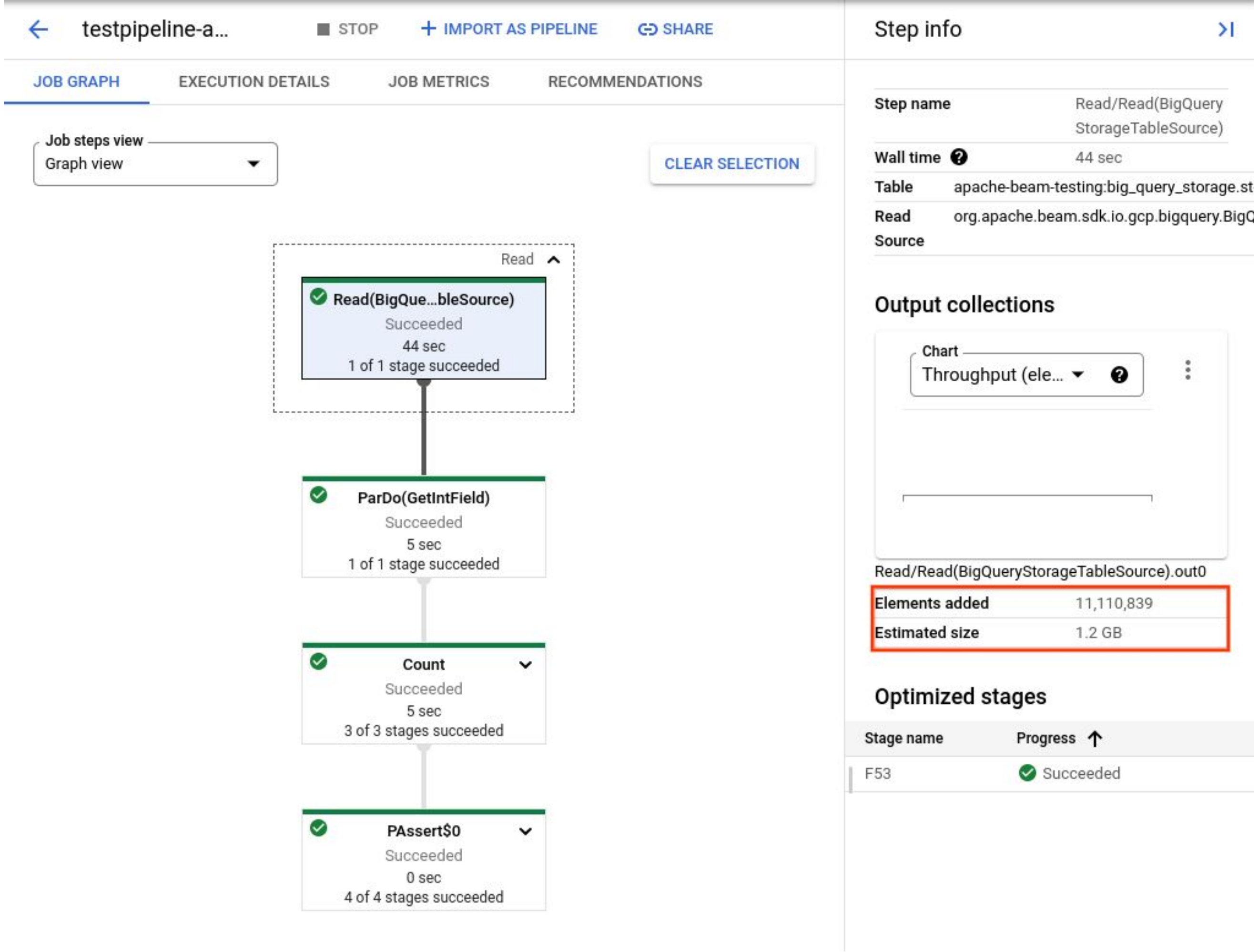
We don’t support this... yet.



```
private static class GetIntField extends DoFn<Row, Long> {

   @ProcessElement
   public int processElement(@FieldAccess("int_field") int int_field) {
     return int_field;
   }
 }
```

Not a live demo but a Beam test!

```
$ ./gradlew :runners:google-cloud-dataflow-java:googleCloudPlatformLegacyWorkerIntegrationTest
--tests "org.apache.beam.sdk.io.gcp.bigquery.
BigQueryIOStorageReadIT.testBigQueryStorageReadProjectionPushdown" --info
...
> :runners:google-cloud-dataflow-java:googleCloudPlatformLegacyWorkerIntegrationTest > Executing test
...
org.apache.beam.runners.core.construction.graph.ProjectionPushdownOptimizer optimize
    INFO: Optimizing transform BigQueryIO.TypedRead: output Tag<output> will contain reduced field set
[int_field]
...
BUILD SUCCESSFUL in 5m 32s
```



Job steps view: Graph view

Read(BigQue...bleSource)
Succeeded
44 sec
1 of 1 stage succeeded

ParDo(GetIntField)
Succeeded
5 sec
1 of 1 stage succeeded

Count
Succeeded
5 sec
3 of 3 stages succeeded

PAssert$0
Succeeded
0 sec
4 of 4 stages succeeded

Step info

Step name	Read/Read(BigQuery StorageTableSource)
Wall time	44 sec
Table	apache-beam-testing:big_query_storage.st
Read Source	org.apache.beam.sdk.io.gcp.bigquery.BigQ

Output collections

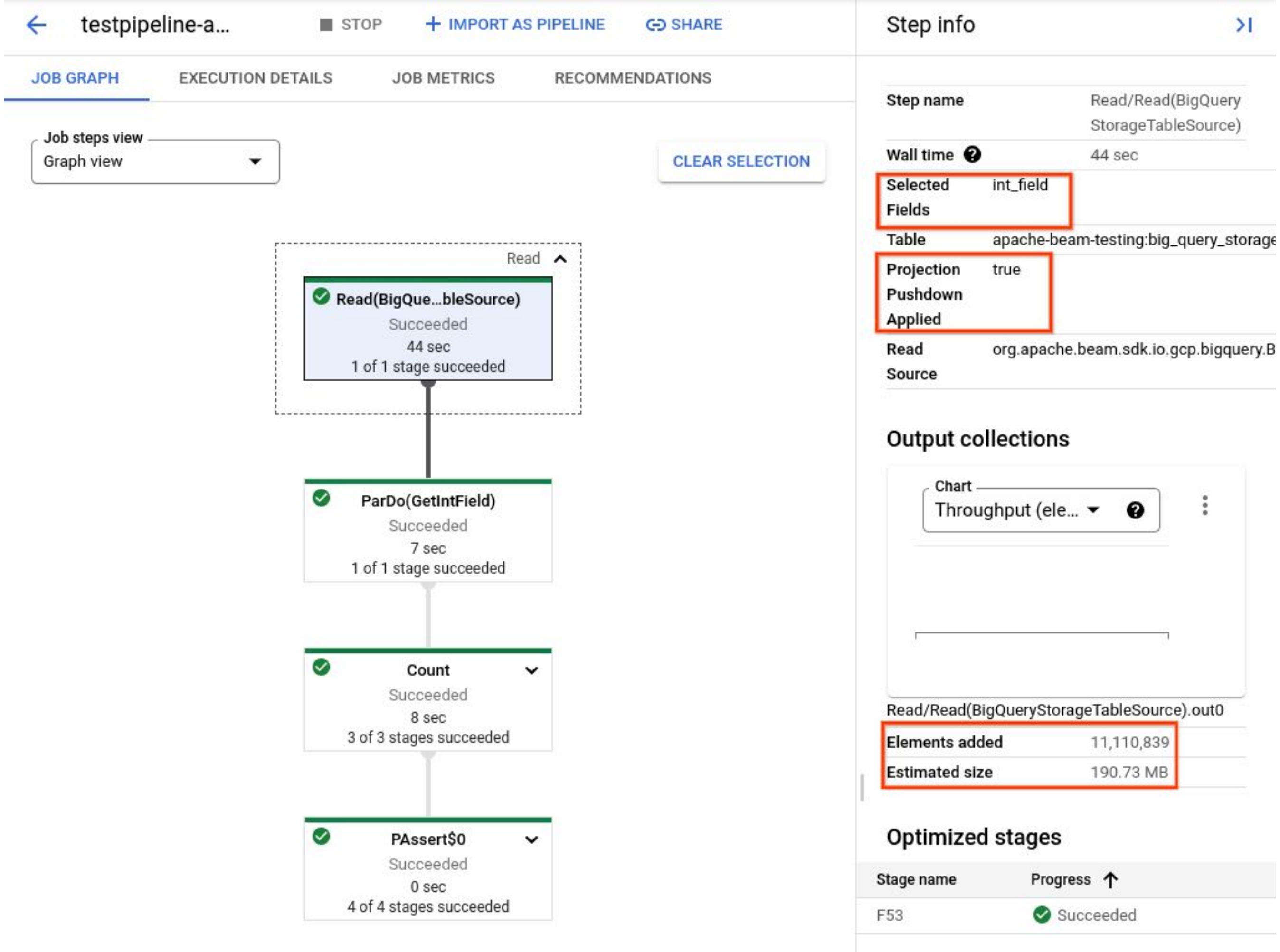
Chart: Throughput (ele...

Read/Read(BigQueryStorageTableSource).out0

Elements added	11,110,839
Estimated size	1.2 GB

Optimized stages

Stage name	Progress ↑
F53	✓ Succeeded



Read(BigQue...bleSource)
Succeeded
44 sec
1 of 1 stage succeeded

ParDo(GetIntField)
Succeeded
7 sec
1 of 1 stage succeeded

Count
Succeeded
8 sec
3 of 3 stages succeeded

PAssert$0
Succeeded
0 sec
4 of 4 stages succeeded

Step info

Step name	Read/Read(BigQuery StorageTableSource)
Wall time	44 sec
Selected Fields	int_field
Table	apache-beam-testing:big_query_storage
Projection Pushdown Applied	true
Read Source	org.apache.beam.sdk.io.gcp.bigquery.B

Output collections

Read/Read(BigQueryStorageTableSource).out0

Elements added	11,110,839
Estimated size	190.73 MB

Optimized stages

Stage name	Progress ↑
F53	✓ Succeeded

Automatically optimize your pipeline

- Only works with BigQueryIO so far.
- On by default for Batch since Beam 2.38.0.
- On by default for Streaming in Beam 2.41.0.

Best Practices

Java: Use FieldAccess and OutputReceiver

```
private static class GetIntField extends DoFn<Row, Long> {

   @ProcessElement
   public void processElement(@FieldAccess("int_field") int int_field,
                              OutputReceiver<Long> outputReceiver) {
     outputReceiver.output(int_field);
   }
 }
```

Go: Schemas by Default!

- Go has Schemas by Default!
- Use go structs with Capitalized Identifiers to export fields
  - Or the `beam:”field_name”` tag
- Use SqlTransform
- Unfortunately other relational features aren’t supported.

Python: Use explicitly structured data types

❌
```
beam.Map(lambda some_data: {“foo”: ..., “bar”: ..., “baz”: …})
```

✔
```
beam.Map(lambda some_data: beam.Row(foo=...,
                                    bar=...,
                                    baz=...))
```

✔
```
class MyRowType(NamedTuple):
  foo: int
  bar: str
  baz: float
```

See Schema documentation for details

Python: Use relational transforms

✓
```
beam.Select(‘foo’, ‘bar’, baz=lambda row: row.x + row.y)
```

✓
```
beam.GroupBy(‘foo’).aggregate_fields(‘bar’, sum)
```

✓
```
from apache_beam.dataframe.io import read_csv
# DataFrame sources always produce schemas!
beam_df = p | read_csv(“...”)
```

Questions?

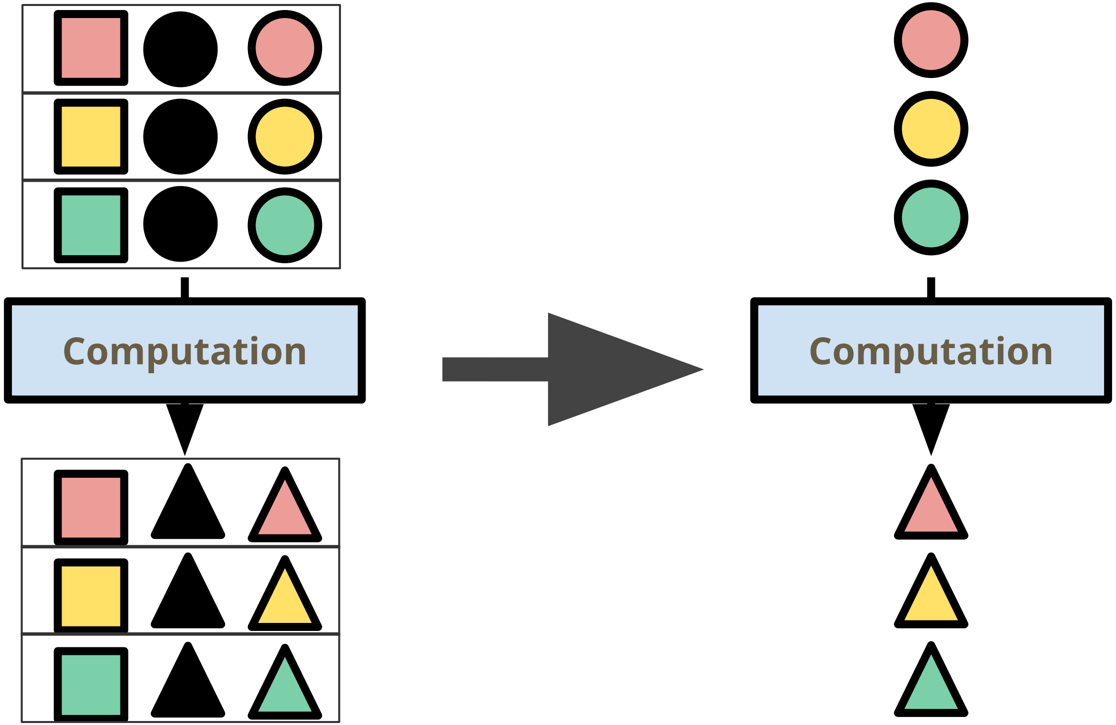
Relational Beam: Process columns, not rows!

Andrew Pilloud / apilloud@apache.org
Brian Hulette / bhulette@apache.org

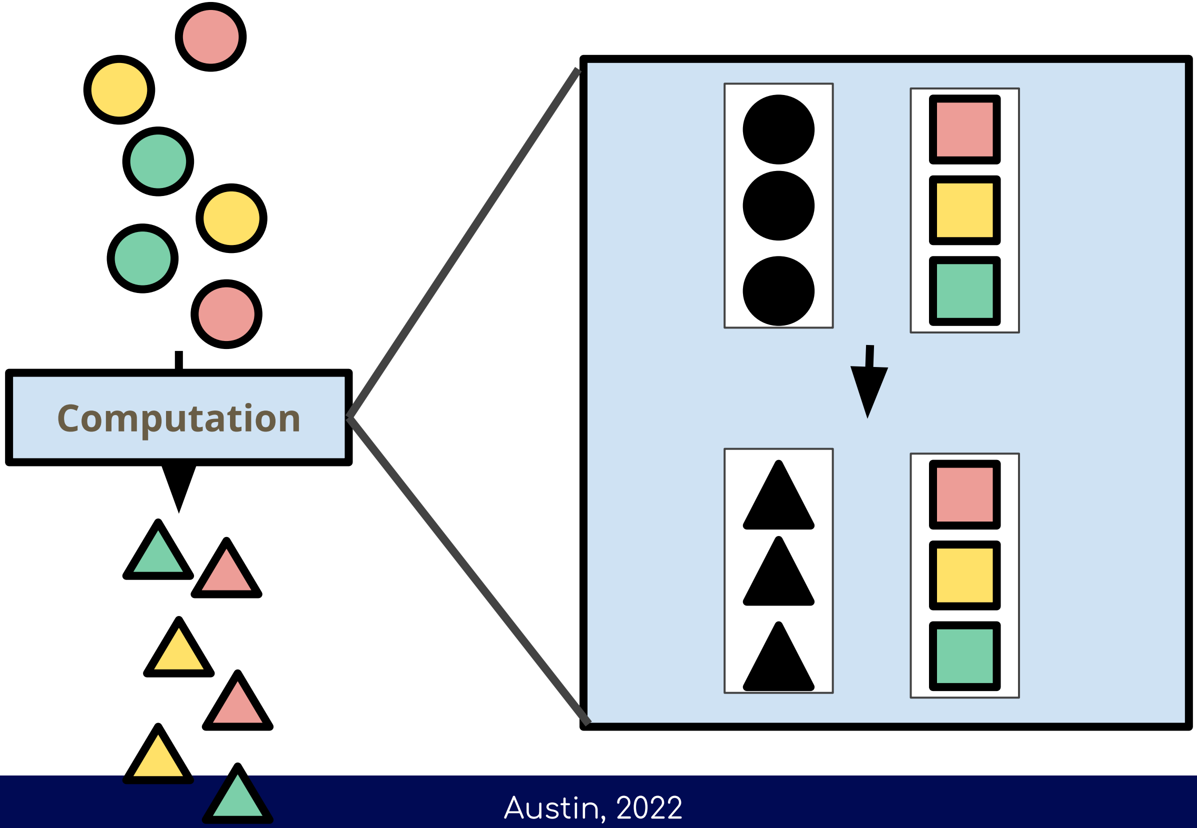
https://s.apache.org/beam-relational-2022

How can we optimize with Relational?

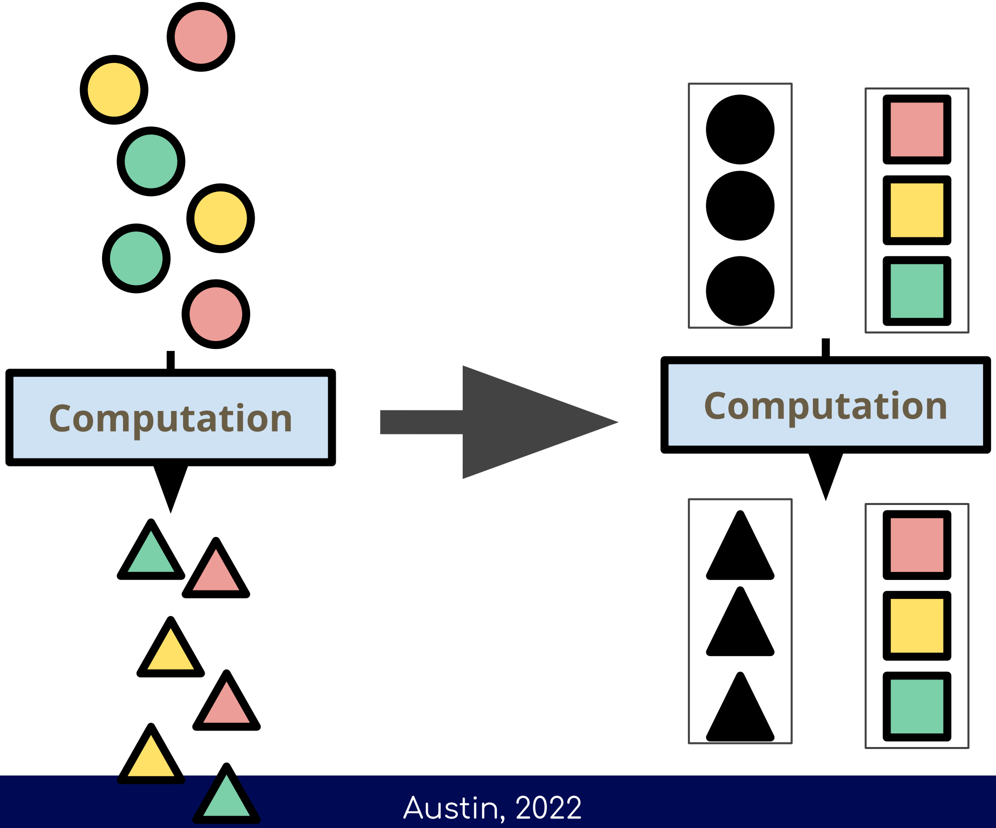
Runner Visibility into Row type



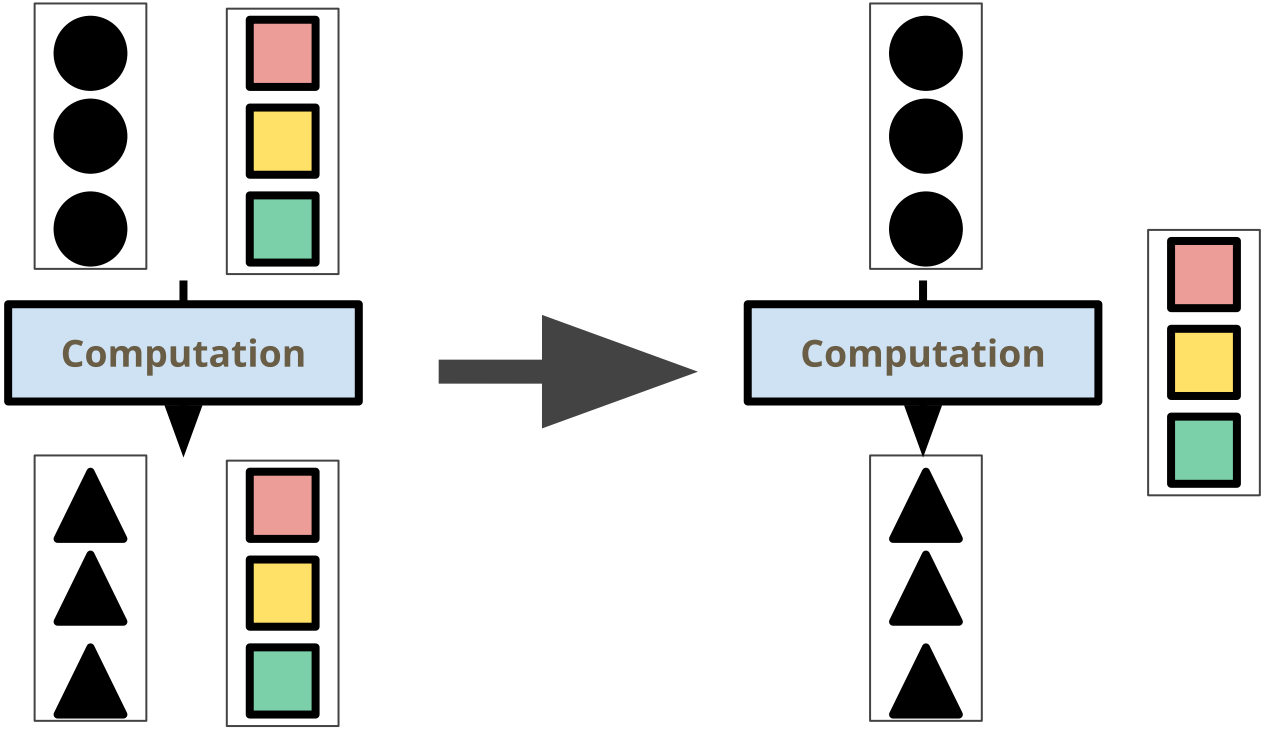
Vectorized Execution



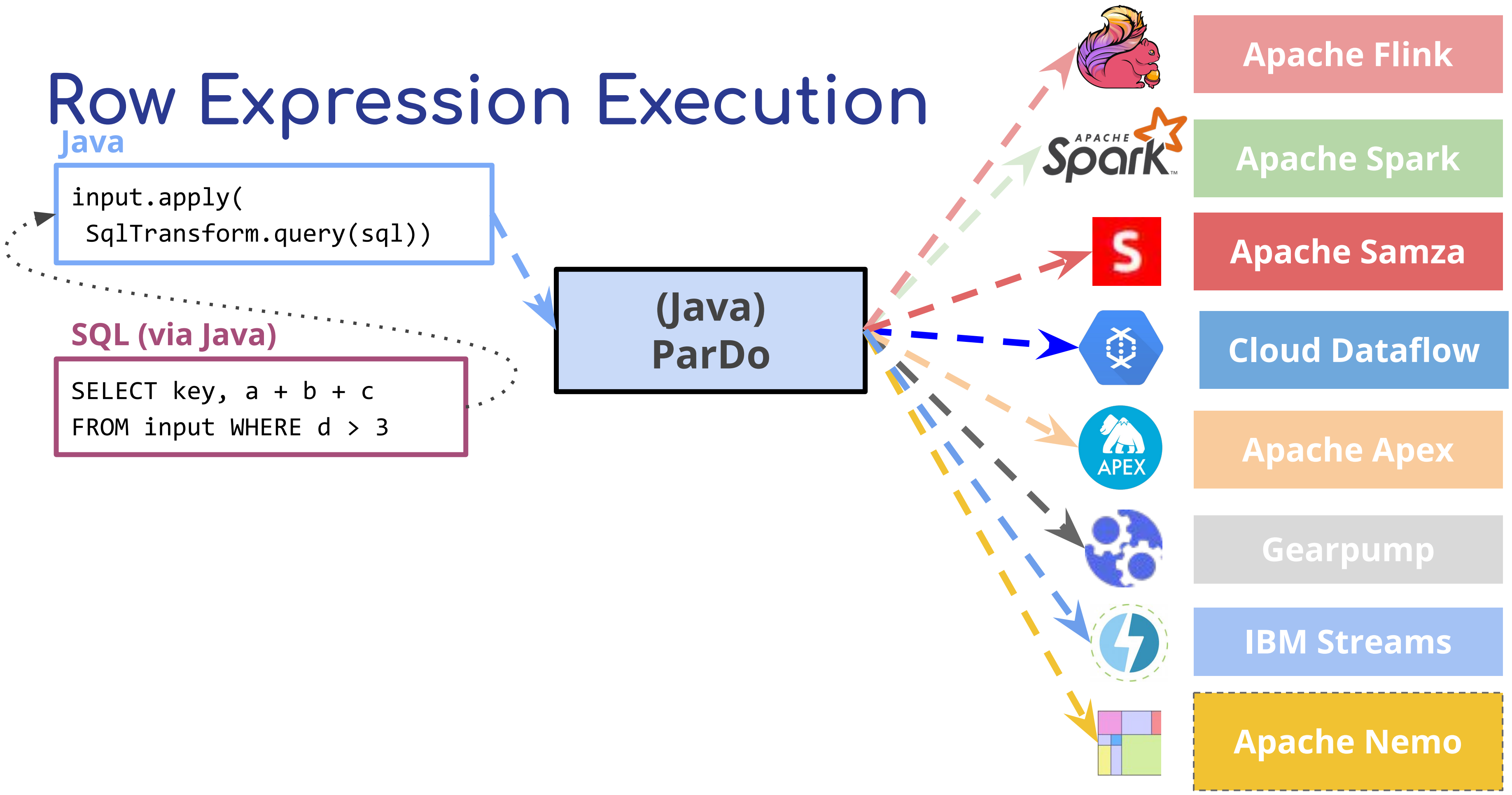
Columnar Coder



Zero-Copy Project and Deferred Deserialization



Row Expression Execution



Java

```
input.apply(
 SqlTransform.query(sql))
```

SQL (via Java)

```
SELECT key, a + b + c
FROM input WHERE d > 3
```

Apache Flink

Apache Spark

Apache Samza

Cloud Dataflow

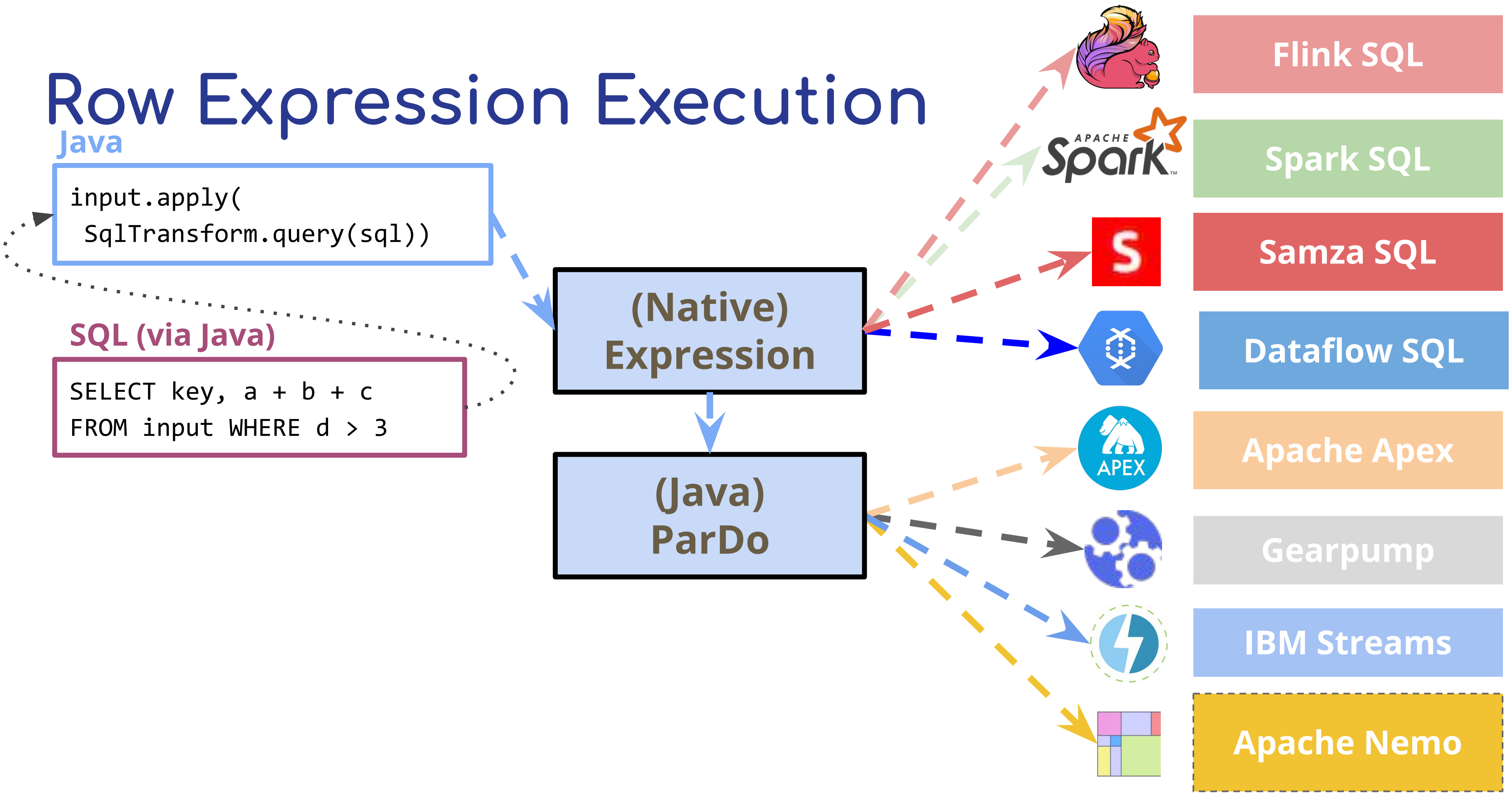
Apache Apex

Gearpump

IBM Streams

Apache Nemo

Row Expression Execution



Java

```
input.apply(
 SqlTransform.query(sql))
```

SQL (via Java)

```
SELECT key, a + b + c
FROM input WHERE d > 3
```

(Native) Expression

(Java) ParDo

Flink SQL

Spark SQL

Samza SQL

Dataflow SQL

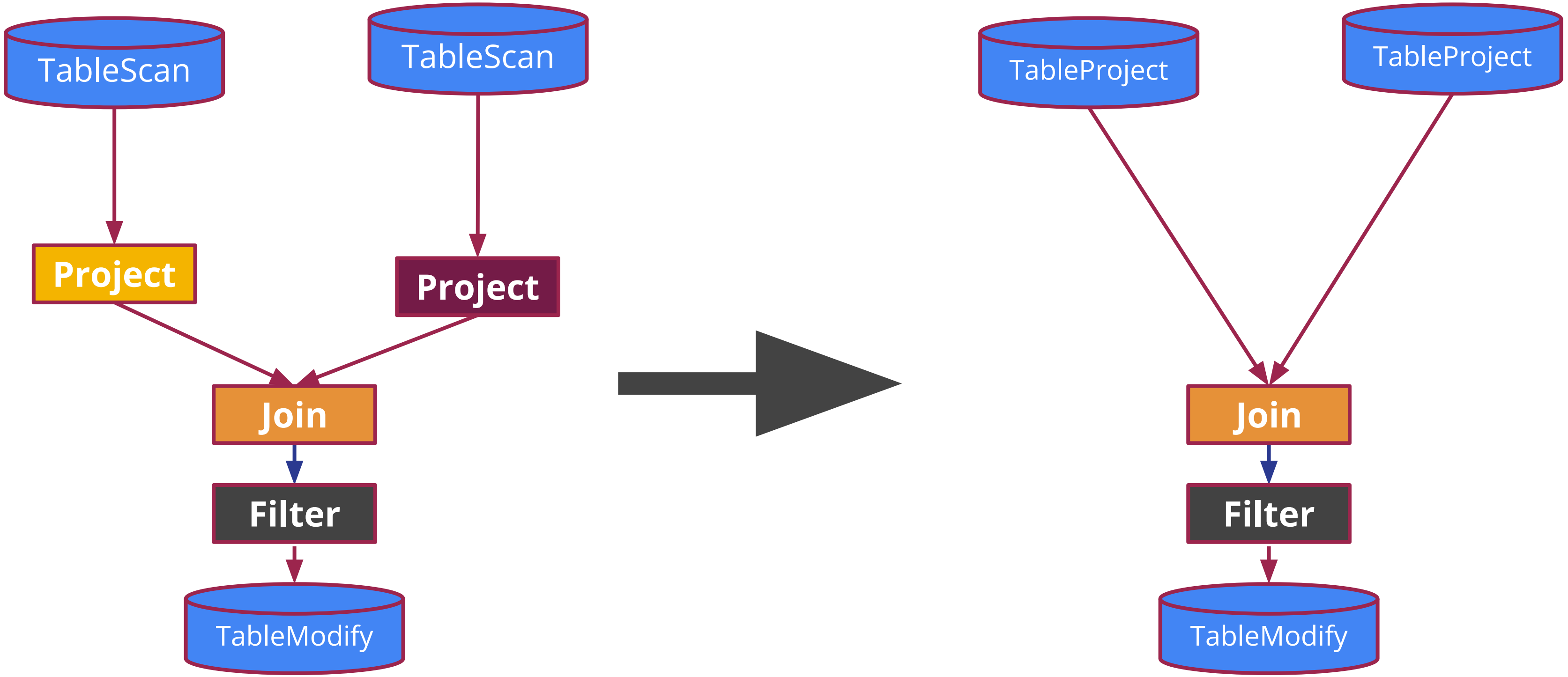
Apache Apex

Gearpump

IBM Streams

Apache Nemo

Global Relational Optimizer



Even More

- Order Aware Pcollections
- Retractions
- Hand optimized type conversions
- Even More

Questions?

Relational Beam: Process columns, not rows!

Andrew Pilloud / apilloud@apache.org
Brian Hulette / bhulette@apache.org

https://s.apache.org/beam-relational-2022

BEAM SUMMIT Austin, 2022