Schemas

David Sabater

Outbound Product Manager,
Google Cloud

Agenda

Course Intro

Beam Concepts Review

Windows, Watermarks, and Triggers

Sources and Sinks

Schemas

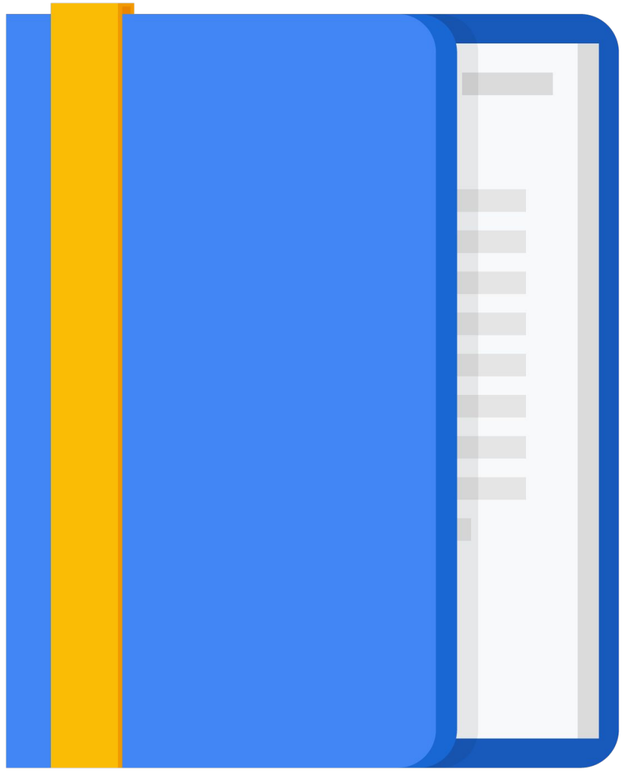
State and Timer

Best Practices

SQL and DataFrames

Beam Notebooks

Summary



Schemas

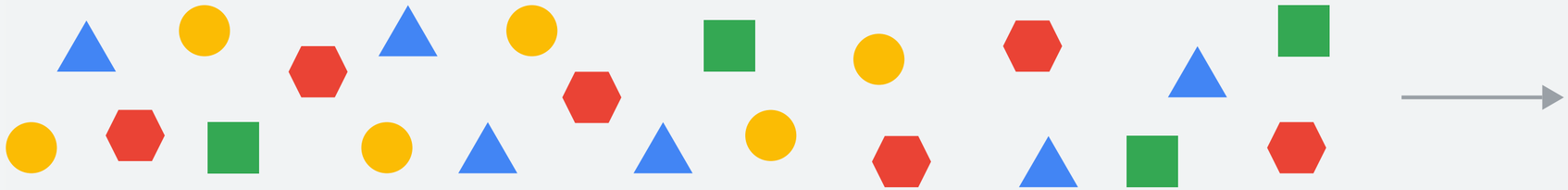
Agenda

Beam schemas

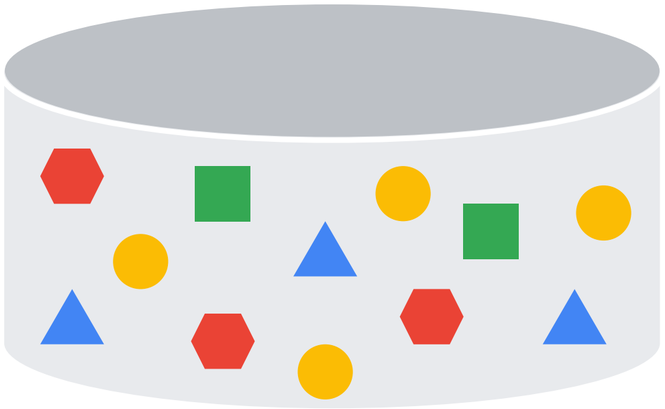
Code examples

“Classic” Beam: Elements processed as blobs

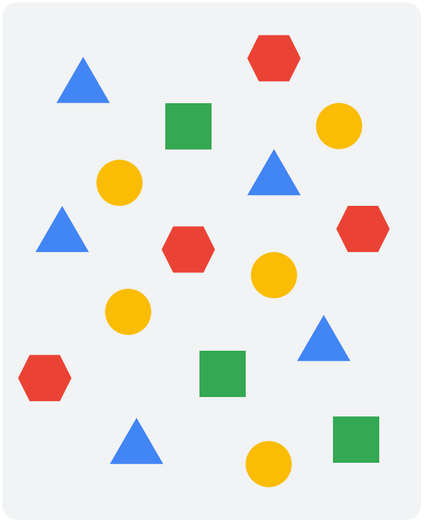
Unbounded collection



Bounded collection



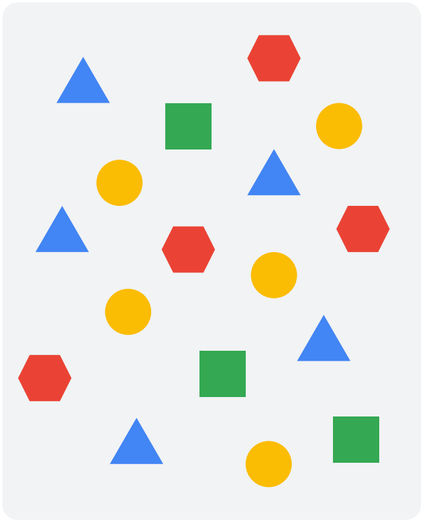
Converting elements into objects



byte[*&&J%$#] → `DoFn<byte[],...>`

You can access as bytes...

Converting elements into objects



OR

byte[*&&J%$#] →

`DoFn<byte[],...>`

You can access as bytes...

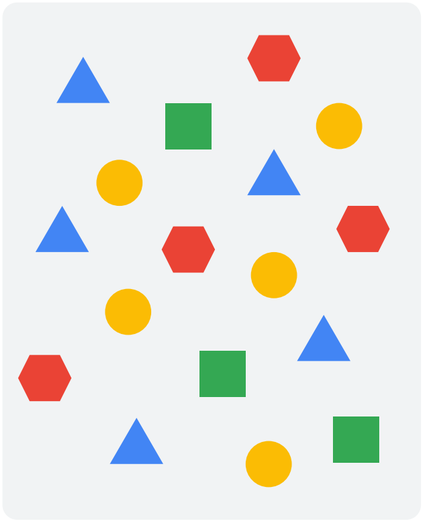
Proto object →

`DoFn<Transaction,...> .withCoder`

(Easy peasy Proto)

Beam can also infer the object using Proto, Avro, BigQuery...

Converting elements into objects



OR

OR

byte[*&&J%$#] →

`DoFn<byte[],...>`

You can access as bytes...

Proto object →

`DoFn<Transaction,...> .withCoder`

(Easy peasy Proto)

Beam can also infer the object using Proto, Avro, BigQuery...

Custom serialization →

`DoFn<Transaction,...>.withCoder`

(Coder thing....)

You can use a custom coder to encode / decode...not a lot of fun.

Schemas to the rescue!

By understanding the structure of a pipeline’s records,
we can provide much more concise APIs for data processing.

What is a schema?

- A schema describes a type in terms of fields and values.

```
Transactions
bank:                    STRING
transactionId :          STRING
purchaseAmountCnts :     LONG
```

What is a schema?

- A schema describes a type in terms of fields and values.
- Fields can have string names or be numerical indexed.

```
Transactions
bank:                    STRING
transactionId :          STRING
purchaseAmountCnts :     LONG
```

What is a schema?

- A schema describes a type in terms of fields and values.
- Fields can have string names or be numerical indexed.
- There is a known list of primitive types a field can have, like int, long, and string.

```
Transactions
bank:                    STRING
transactionId :          STRING
purchaseAmountCnts :     LONG
```

What is a schema?

- A schema describes a type in terms of fields and values.
- Fields can have string names or be numerical indexed.
- There is a known list of primitive types a field can have, like int, long, and string.
- Some fields can be marked as optional.

```
Transactions
bank:                    STRING
transactionId :          STRING
purchaseAmountCnts :     LONG
```

What is a schema?

- A schema describes a type in terms of fields and values.
- Fields can have string names or be numerical indexed.
- There is a known list of primitive types a field can have, like int, long, and string.
- Some fields can be marked as optional.
- Schemas can be nested arbitrarily, and can contain repeated or map fields as well!

```
Transactions
bank:                    STRING
transactionId :          STRING
purchaseAmountCnts :     LONG
```

Schemas: Inferred at sources

One stream, encoded with Avro

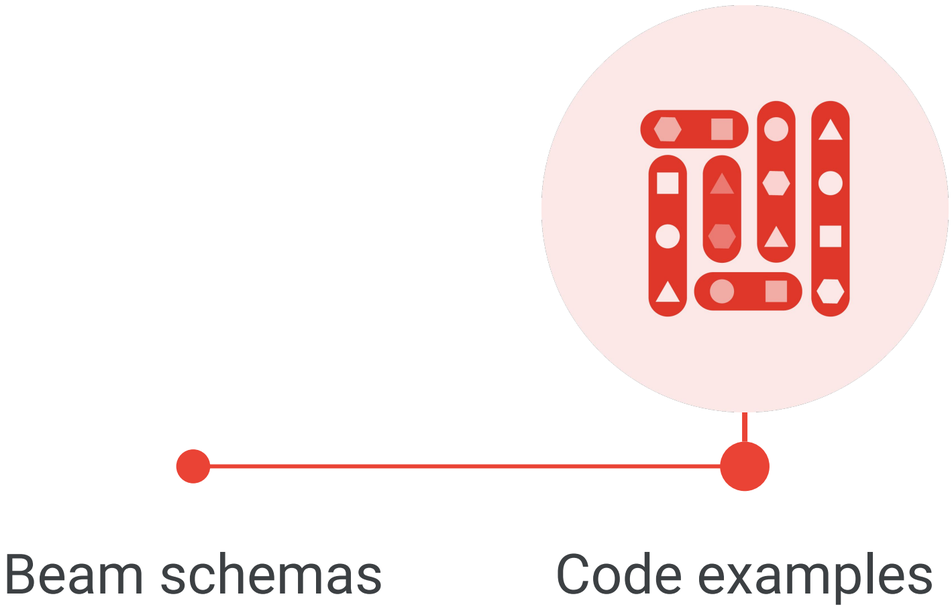
Schemas: Inferred at sources

One stream, encoded with Avro

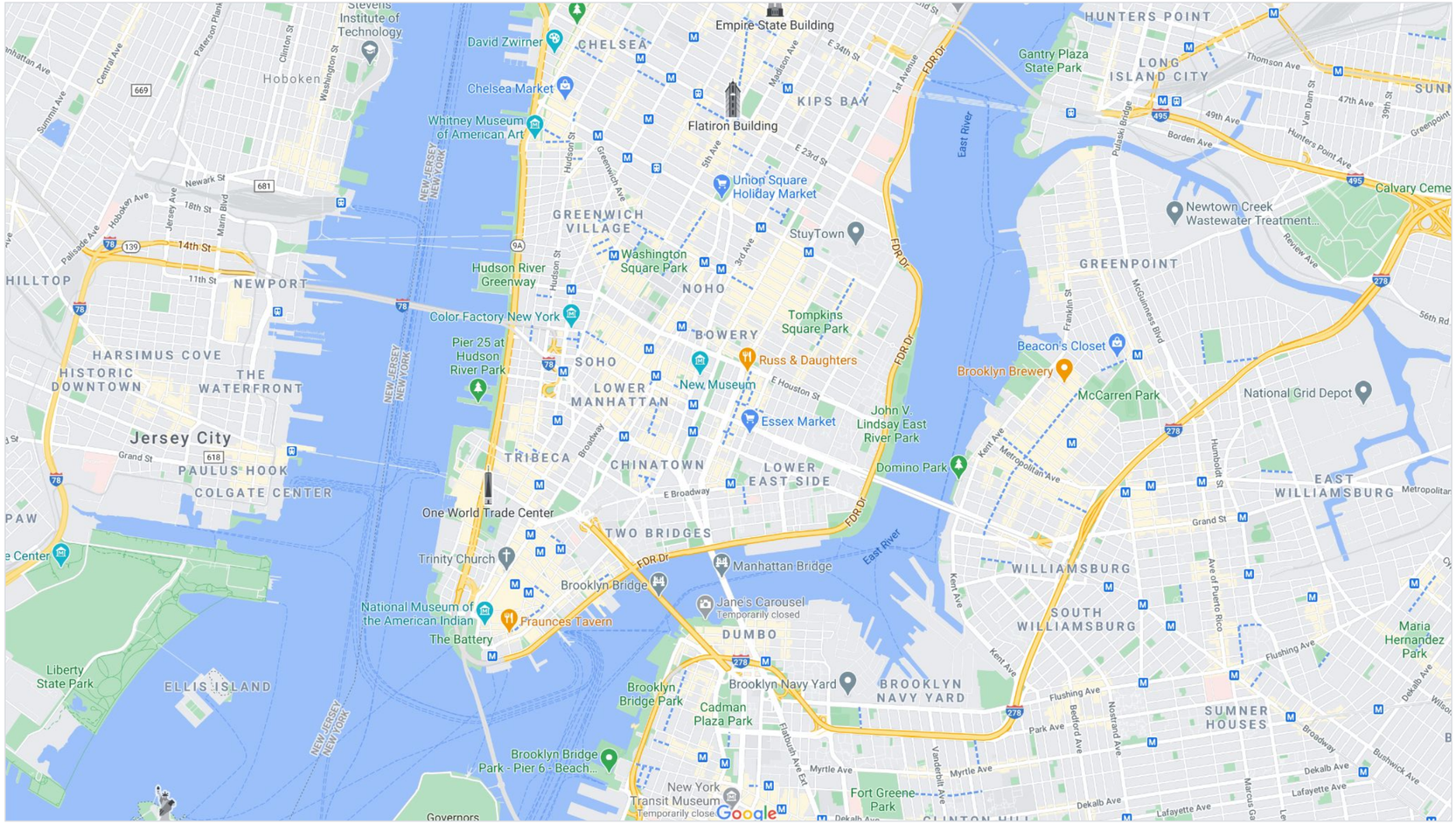
```
PCollection<Purchase> purchases = p.apply(
    PubSubIO.readAvros(Purchase.class).fromTopic(purchaseTopic));
```

Schemas

Agenda



Java example 1: Filter purchases



Java example 1: Filter purchases

Without schemas

```
purchases.apply(Filter.by(purchase -> {
  return purchase.location.lat < 40.720 && purchase.location.lat > 40.699
    && purchase.location.lon < -73.969 && purchase.location.lon > -74.747}));
```

With schemas

```
purchases.apply(
     Filter.whereFieldName("location.lat", (double lat) -> lat < 40.720 && lat >
40.699)
           .whereFieldName("location.lon", (double lon) -> lon < -73.969 && lon >
-74.747));
```

Example 2: Review total purchases per transaction

Java

1. Join transactions with purchases.
2. One transaction always has one or more purchases, so we need an inner join.
3. We want to calculate the total purchases per transaction, grouped by user ID.

Example 2: Review total purchases per transaction

Java—Without schema

```
final TupleTag<Purchase> lhsTag = new TupleTag<>();
final TupleTag<Transaction> rhsTag = new TupleTag<>();
PCollection<KV<Long, Purchase>> keyedPurchases = purchases.apply(WithKeys.of(p -> p.transactionId)
               .withKeyType(TypeDescriptors.longs()));
PCollection<KV<Long, Transaction>> keyedTransactions = transactions.apply(WithKeys.of(t -> t.transactionId).
               .withKeyType(TypeDescriptors.longs()));
PCollection<KV<String, Long>> userSpend =
    KeyedPCollectionTuple.of(lhsTag, keyedPurchases)
             .and(rhsTag, keyedTransactions)
             .apply(CoGroupByKey.<Long>create())
             .apply(Values.create())
             .apply(ParDo.of(new DoFn<CoGbkResult, KV<String, Long>>() {
                @ProcessElement
                public void process(@Element CoGbkResult result, OutputReceiver<KV<String, Long>> o) {
                  Purchase purchase = result.getOnly(lhsTag);
                for (Transaction transaction : result.getAll(rhsTag)) {
                    o.output(KV.of(purchase.userId, transaction.purchaseAmount));
                 }
             }))
  .apply(Sum.longsPerKey());
```

Example 2: Review total purchases per transaction

Java —With schema

```
PCollection<UserPurchases> userSums =
purchases.apply(Join.innerJoin(transactions).using(“transactionId”))
         .apply(Select.fieldNames(“lhs.userId”, “rhs.totalPurchase”))
         .apply(Group.byField(“userId”).aggregateField(Sum.ofLongs(), “totalPurchase”));
```