Many Formats
One Data Lake

Avro to the Rescue

Agenda

- Intro & Project Background
- Data Lakes & Non-Traditional Requirements
- Avro, Schema, and Sandwiches
- Writing Multiple Outputs
- Benefits and Drawbacks
- Code-Level Tips
- TransBeamer: A Library Implementation
- Questions

Who Am I

Peter Wagener

- Consultant @ SanusCorp
- Beam User for 3+ years
- Java/TS/Python Background
- It's not your connection ... I stutter

Geek Claim to Fame
I once received an email from Stephen Hawking saying the management software I built "... was OK".

Likes & Dislikes

Likes:

- The Beam Java SDK
- Old Logos
- Unit Testing

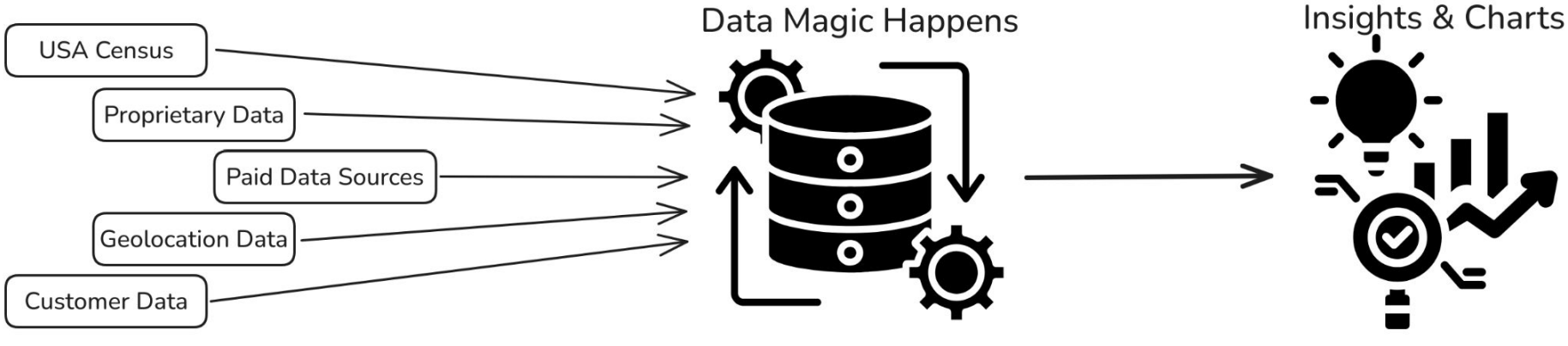
Dislikes:

- Containers in Containers
- Optional fields
- Debugging Claude Code

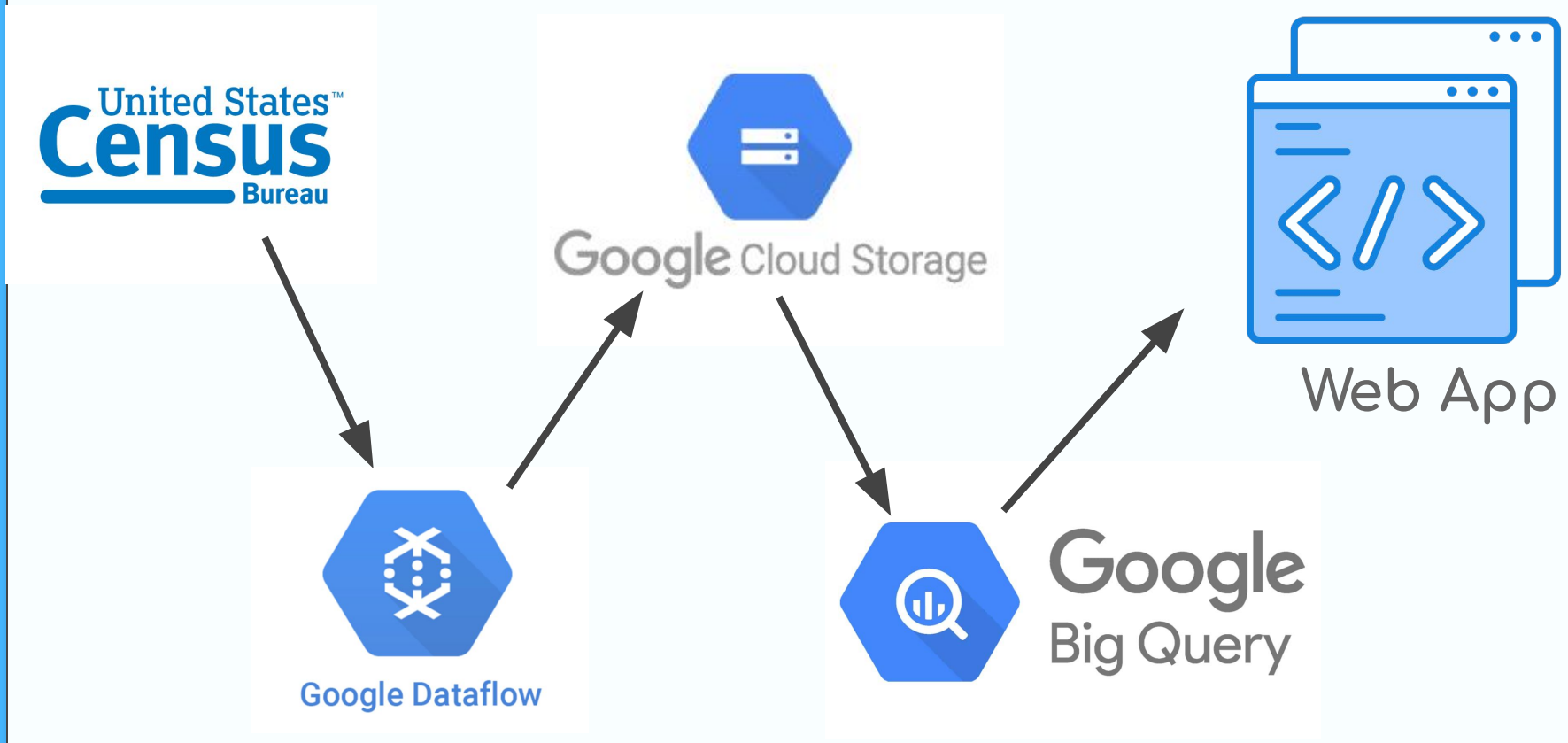
The Project

The ideas in this talk came from building a Community Impact Measurement Tool. It answers questions like:

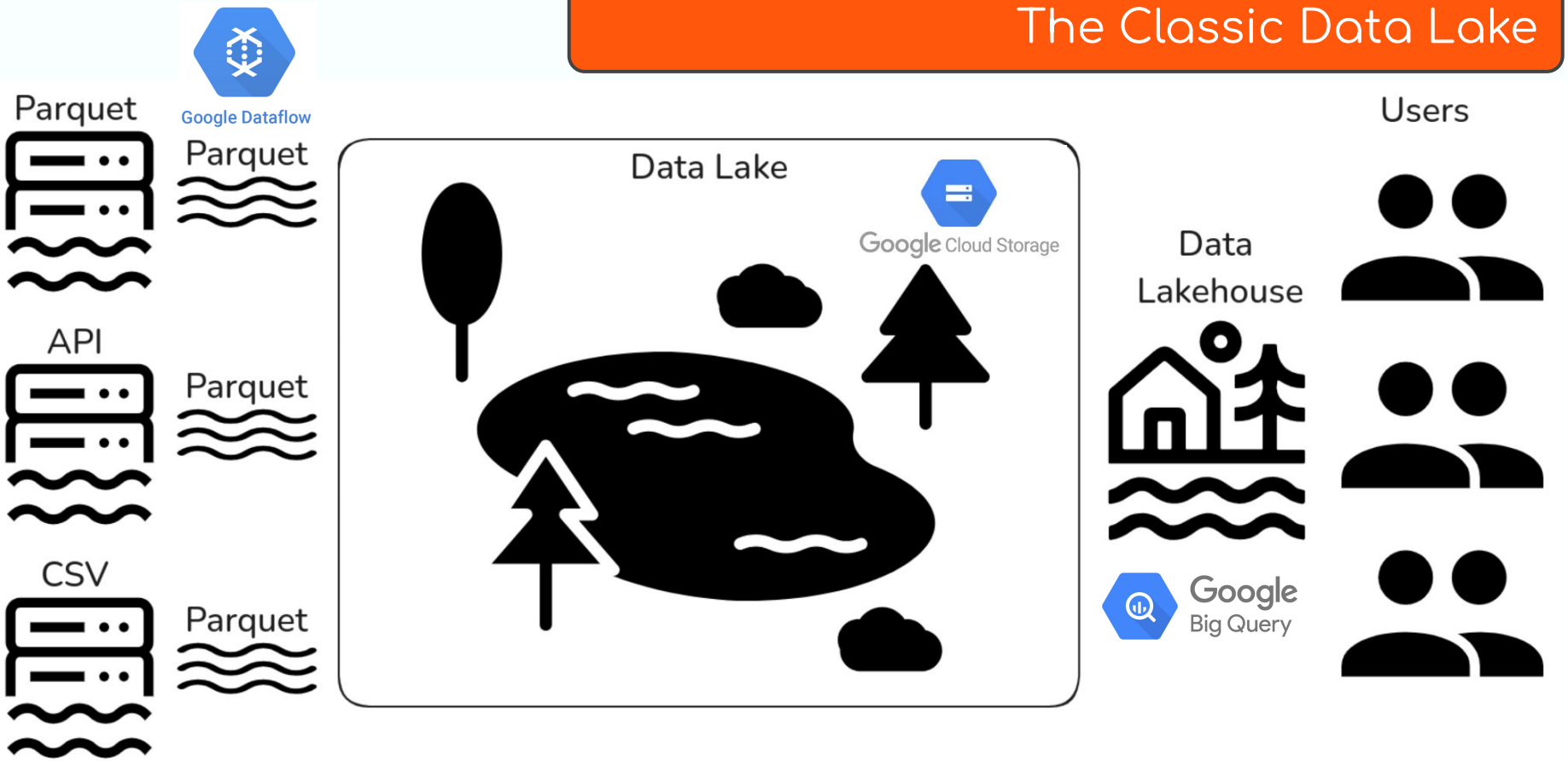
> “If [Company X] deployed capital to [Location Y] in a different way, how much larger or smaller would their overall financial impact be?”



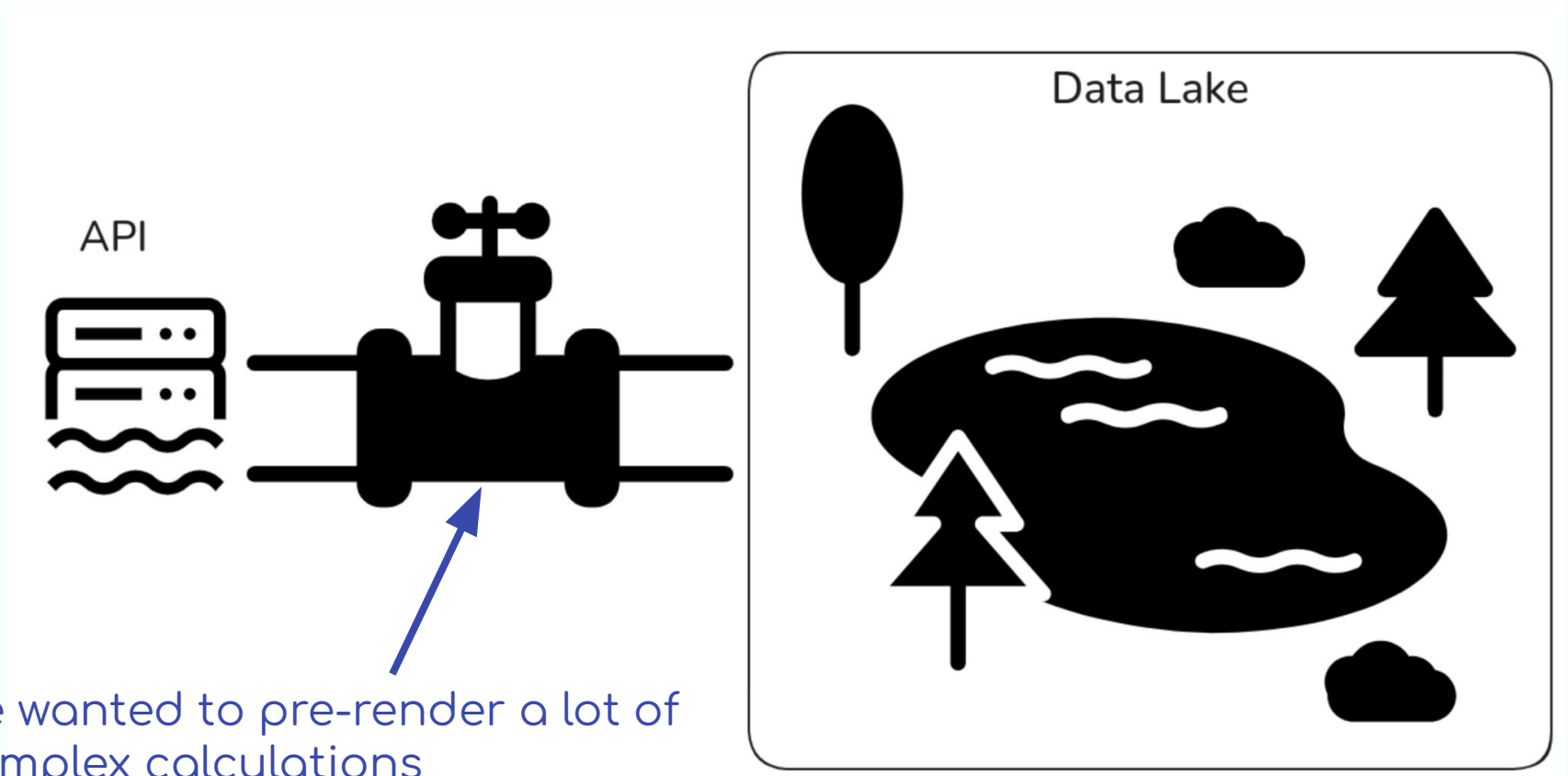
Typical Data Flow



The Classic Data Lake



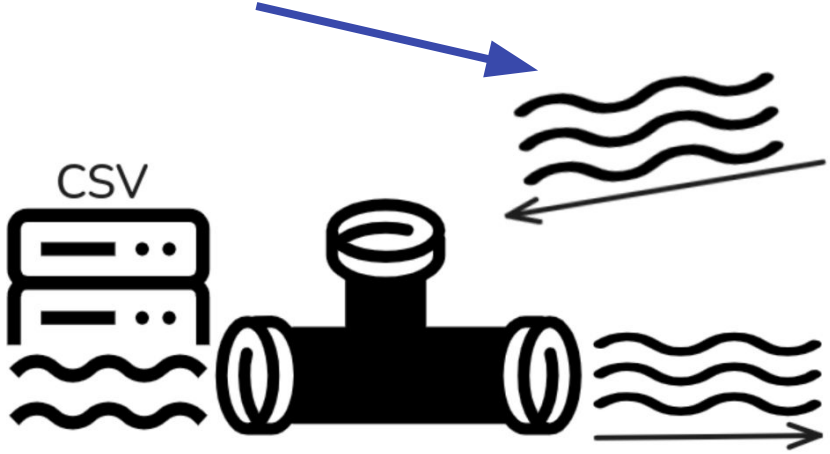
Quirk #1: Complex Transformations

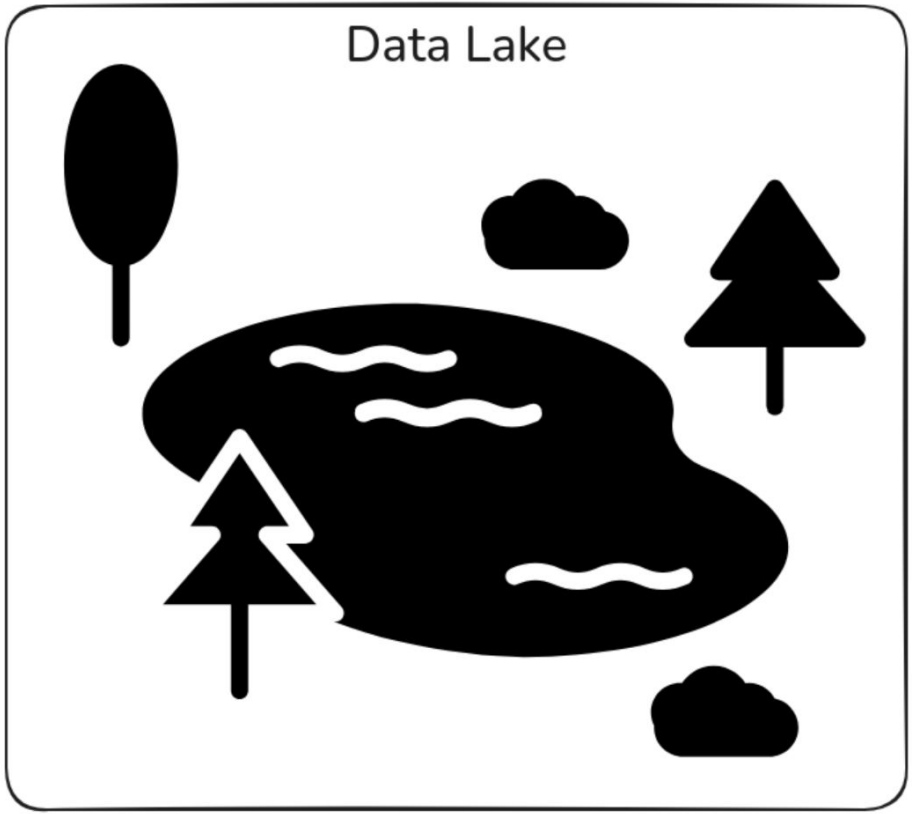


We wanted to pre-render a lot of complex calculations

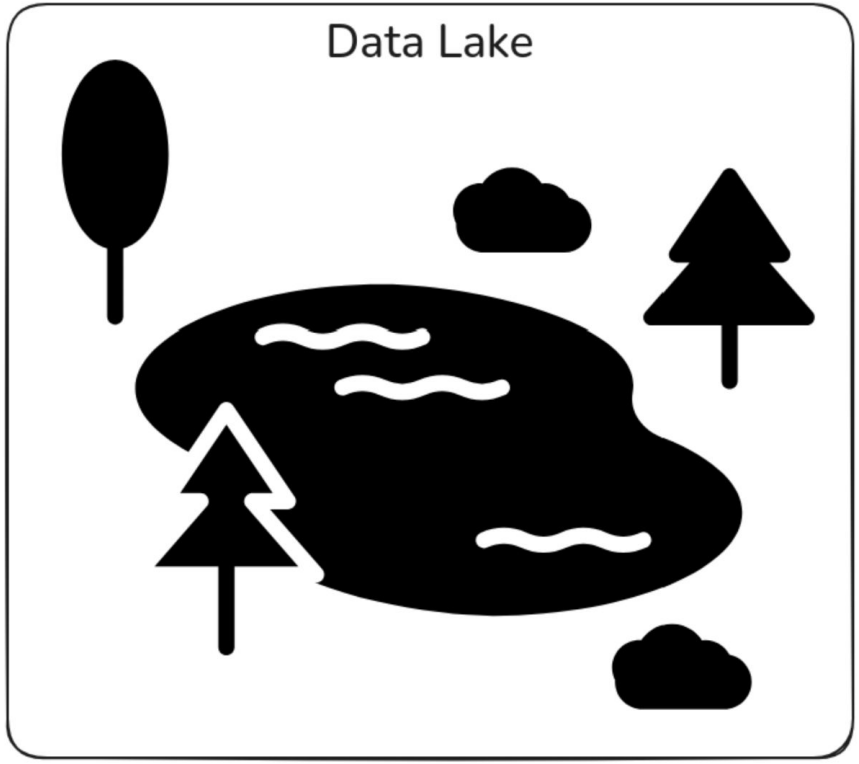
Quirk #2: Pipelines Reading Data Lake

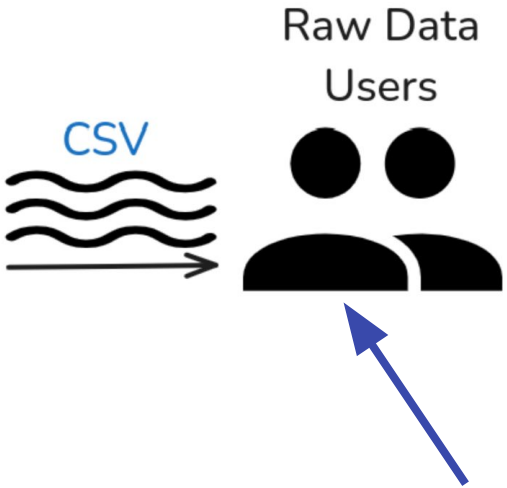
Existing data sets became foundational for other calculations





Quirk #3: Raw Data Users





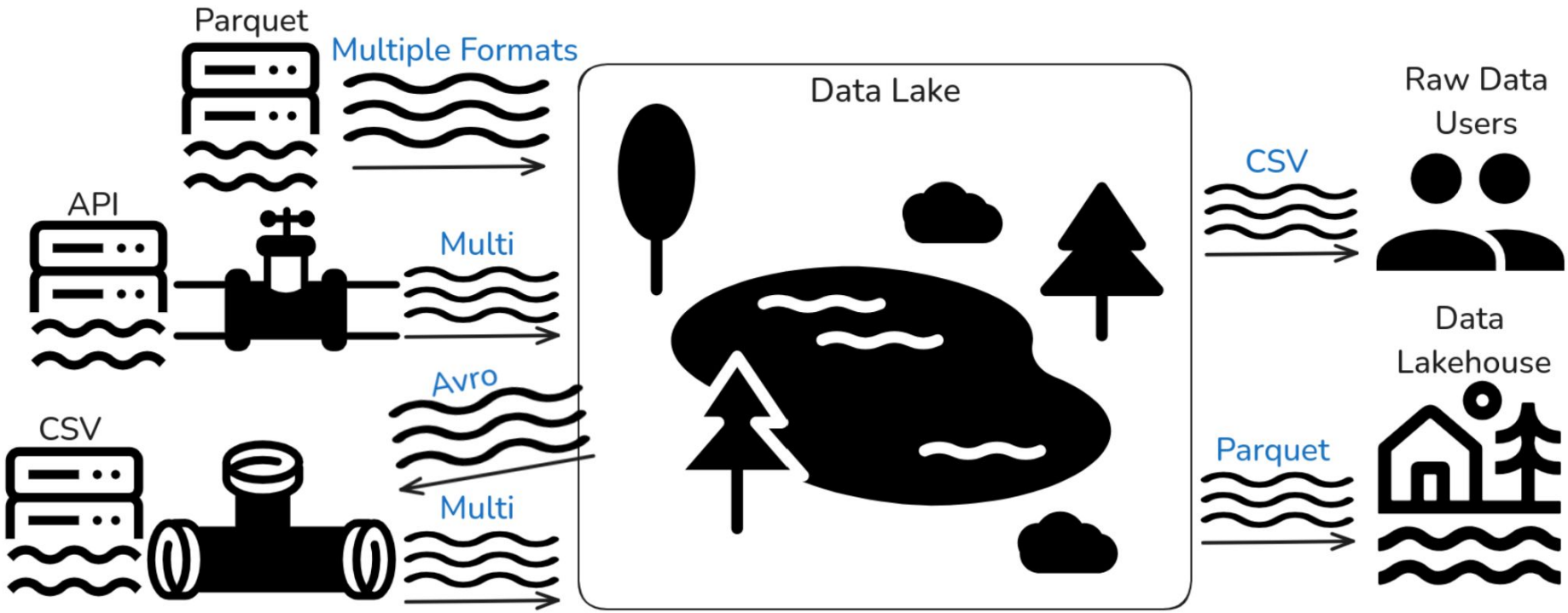
A certain class of users wanted "raw" access to some data sets

Quirk #4: Future Formats

Apache orc™

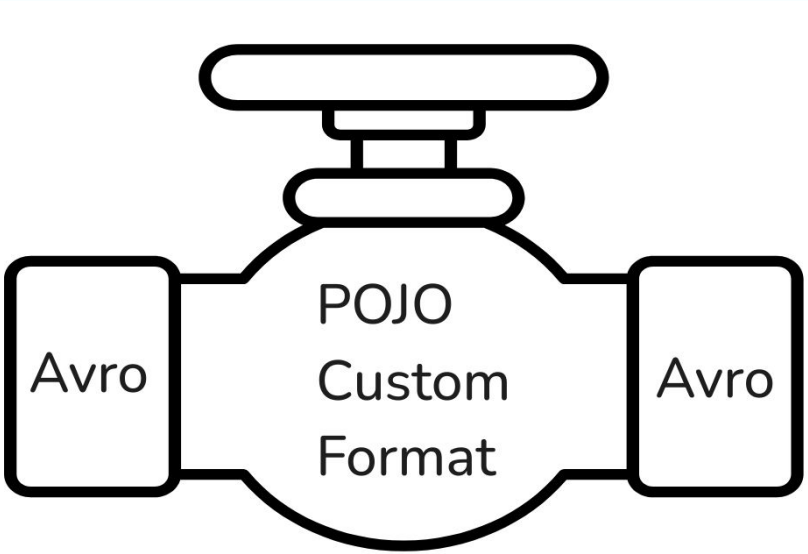
Need to Ingest ...	Need to Produce ...
Parquet (Big Data Formats)	Parquet (Lakehouse Access)
CSV (Common Exported Data)	CSV (Raw Data Users)
Avro (Re-Entrant Pipelines)	Avro (Re-Entrant Pipelines)
Protobuf? ORC? XML? JSON?	Iceberg (Snapshotting, etc.)

This is the Data Lake They Needed



“An Avro Sandwich”

A talk from 2023 described their Pipelines as an “Avro Sandwich”:



“A Beginners Guide to Avro ...”
– Devon Peticolas (2023)

Avro

Avro Schema

Defining a schema

Avro schemas are defined using JSON. Schemas are composed of primitive types (null, boolean, int, long, float, double, bytes, and string) and complex types (record, enum, array, map, union, and fixed). You can learn more about Avro schemas and types from the specification, but for now let's start with a simple schema example, user.avsc:

```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Avro

Avro Code Generation

Serializing and deserializing with code generation

Compiling the schema

Code generation allows us to automatically create classes based on our previously-defined schema. Once we have defined the relevant classes, there is no need to use the schema directly in our programs. We use the avro-tools jar to generate code as follows:

```
java -jar /path/to/avro-tools-1.11.1.jar compile schema <schema file> <destination>
```

This will generate the appropriate source files in a package based on the schema's namespace in the provided destination folder. For instance, to generate a User class in package example.avro from the schema defined above,

Avro

Generated Code

```
package example.avro;
/* ... */
public class User extends SpecificRecordBase implements SpecificRecord {
  public static final org.apache.avro.Schema SCHEMA$ = /* ... */;
  private static final BinaryMessageEncoder<User> ENCODER = /* ... */;
  private static final BinaryMessageDecoder<User> DECODER = /* ... */;

  /* ... */
  public User() {}
  public User(String name, Integer favorite_number, String favorite_color) {
    this.name = name;
    this.favorite_number = favorite_number;
    this.favorite_color = favorite_color;
  }

  public String getName() { return name; }
  public Integer getFavoriteNumber() { return favorite_number; }
  /* ... */
```

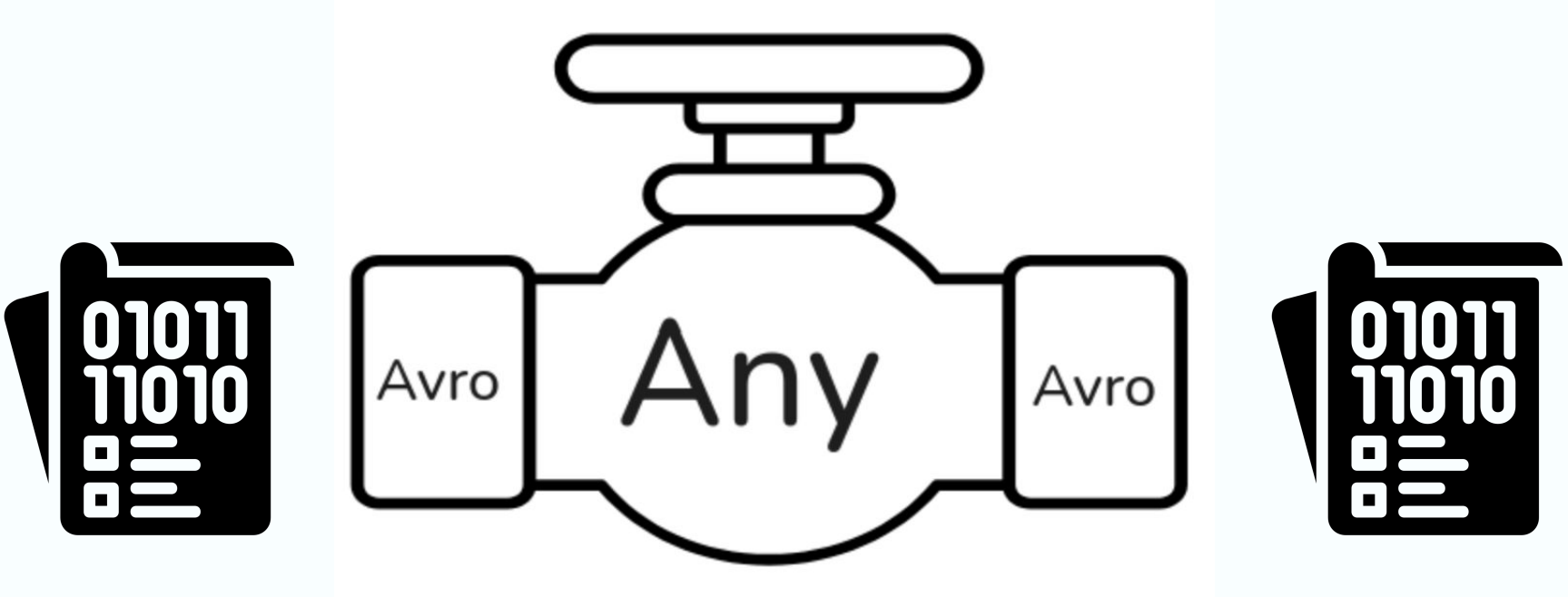
Interlude: Related Old Logos

Back to Our Talk: Avro's Benefits

Avro

Avro Benefit: Strong Tool Support

Avro Benefit: Pipeline Borders



Avro Benefit: Built-in Beam Schema

6.3. Schema definition

The schema for a `PCollection` defines elements of that `PCollection` as an ordered list of named fields. Each field has a name, a type, and possibly a set of user options. The type of a field can be primitive or composite. The following are the primitive types currently supported by Beam:

Type	Description
BYTE	An 8-bit signed value
INT16	A 16-bit signed value
INT32	A 32-bit signed value
INT64	A 64-bit signed value
DECIMAL	An arbitrary-precision decimal type
FLOAT	A 32-bit IEEE 754 floating point number
DOUBLE	A 64-bit IEEE 754 floating point number
STRING	A string
DATETIME	A timestamp represented as milliseconds since the epoch
BOOLEAN	A boolean value
BYTES	A raw byte array

(also support Nested, Array, Iterable, and Map types)

Avro Schema

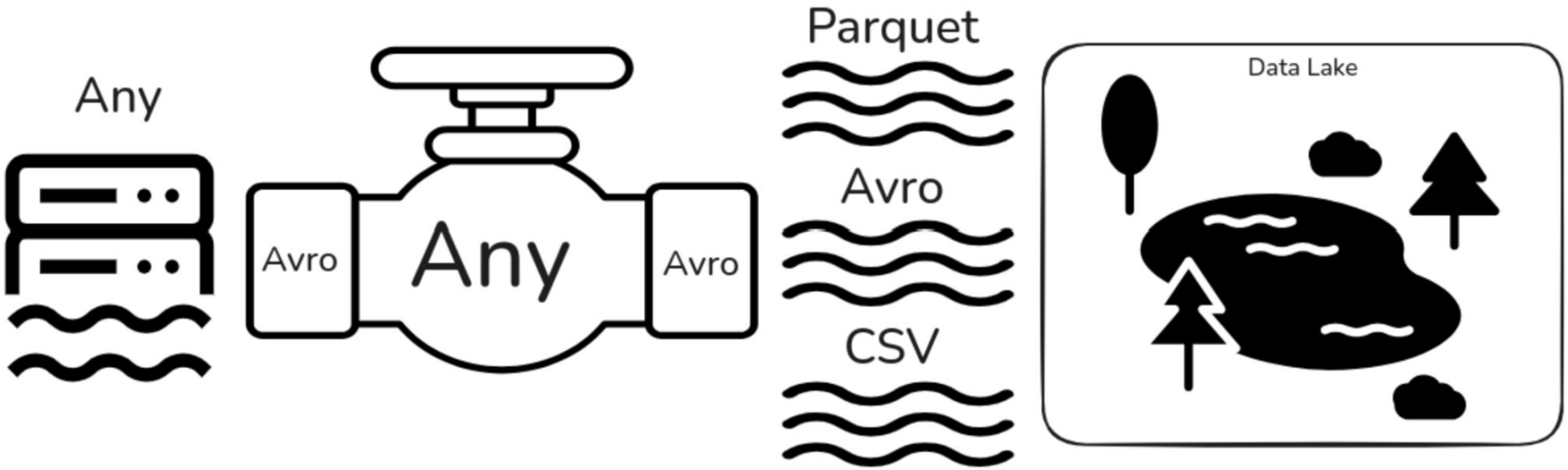
- Useful Within and Outside the Pipeline
- Reading via AvroIO will automatically attach a Beam Schema
- Generating Avro instances within a pipeline *may not* automatically attach a Beam Schema

Beam Schema

- Useful Within the Pipeline
- Enables Transforms:
  - `Convert.to(...)`
  - `Join.<Row, Row>(...)`
  - `Select.fieldNames(...)`
  - `Group.byFieldNames(...)`
  - `Filter.whereFieldName(...)`
- Enables DoFn Features:
  - @FieldAccess(...)
  - @Element(...) conversion with matching schema

Writing Multiple Outputs

Our pipelines took the “Avro Sandwich” approach, with the nuance that inputs & outputs could be any supported text file format. In practice, we wrote multiple copies of the outputs.



Benefits

Avro is Stable & Flexible
Boring technology is a good thing.

Built-in Beam Schemas
This enables some great built-in transforms and IO support

User-Specific Formats
Different formats are good for different users

Drawbacks

"Does It All Have to be Avro?"
Anything coming in or going out must be described by an Avro Schema.

"Too Much IO ... ?"
If your pipeline is IO bound, duplicating IO won't help.

"Data Overload"
If your data lake is already a data swamp, this will make it worse.

Code Tip: Use Avro Annotations

Avro Schema become generated Java classes. Declare Java annotations in your Avro Schema. Specifically, the *DefaultSchema* and *DefaultCoder*.

```
{
  "namespace": "example.avro",
  "type": "record",
  "name": "User",
  "fields": [ /* ... */ ],
  "javaAnnotation": [
"org.apache.beam.sdk.schemas.annotations.DefaultSchema(
    org.apache.beam.sdk.extensions.avro.schemas.AvroRecordSchema.class
)", "org.apache.beam.sdk.coders.DefaultCoder(
    org.apache.beam.sdk.extensions.avro.coders.AvroCoder.class
)"
  ]
}
```

Code Tip: Love the Generated Classes

Learn to love the Avro-Generated Java Classes.

- They are JavaBeans
- They support the Builder pattern
- They embed the Avro Schema
- They (can) support immutability
- They are ugly... but readable ...

```
@org.apache.beam.sdk.schemas.annotations.DefaultSchema(org.apache.beam.sdk.extens
@org.apache.beam.sdk.coders.DefaultCoder(org.apache.beam.sdk.extensions.avro.code
@org.apache.avro.specific.AvroGenerated
public class StarWarsMovie extends org.apache.avro.specific.SpecificRecordBase
    implements org.apache.avro.specific.SpecificRecord {
  private static final long serialVersionUID = -5793972743567361116L;
  public static final org.apache.avro.Schema SCHEMA$ = new org.apache.avro.Schema
```

Code Tip: Alphabetize Fields

You end up writing a lot of Avro Schema files. And certain formats in Beam IO (i.e. Parquet) require consistent field ordering. Alphabetizing just makes your life easier.

```
{
  "namespace": "example.avro",
  "type": "record",
  "name": "User",
  "fields": [
     {"name": "favorite_color", "type": ["null", "string"]}
     {"name": "favorite_number",  "type": ["null", "int"]},
     {"name": "name", "type": "string"},
  ]
}
```

Code Tip: Use A Library

README | MIT license

TransBeamer

maven-central v1.2.0 | License MIT

The TransBeamer library provides utilities for reading and writing data of various formats in Apache Beam pipelines, populating Avro-based PCollections as interim values.

The goal of the library is to make it easy for Beam pipelines to read in any text-based format into a `PCollection` backed by elements described by Avro schema. Then, when the pipeline is done processing data, make it easy to write that data back out to a variety of formats.

Features

- **Multiple Format Support**: Read and write CSV, Avro, Parquet, and NDJson formats
- **Consistent Reading/Writing API**: One API for multiple formats
- **Extensible Format Support**: Write your own formats as needed
- **Avro-Centric**: Uses Avro as the intermediate data format for strong schema, coder support

TransBeamer Library

Aside: Reading Non-Trivial CSVs? Hmm...

org.apache.beam.sdk.io.csv

Class CsvIO

java.lang.Object
    org.apache.beam.sdk.io.csv.CsvIO

```
public class CsvIO
extends java.lang.Object
```

PTransforms for reading and writing CSV files.

Reading CSV files

Reading from CSV files is not yet implemented. Please see https://github.com/apache/beam/issues/24552.

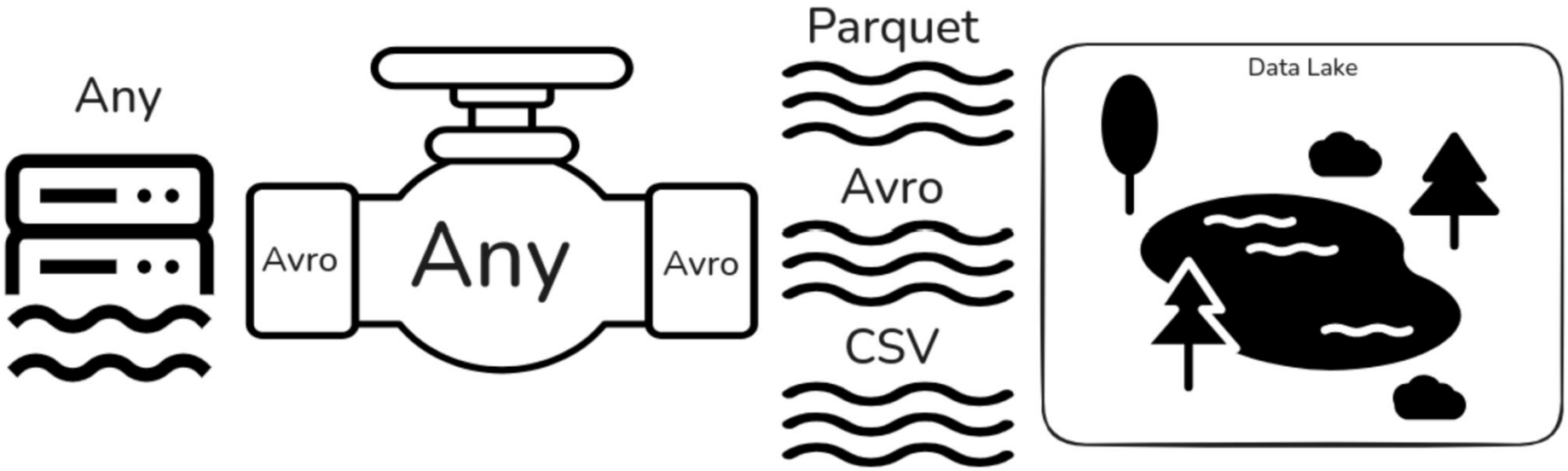
Reading CSV Files via TransBeamer

```
// Read from s3://my-bucket/data/Movies*.csv
PCollection<Movie> movies = pipeline.apply(
    TransBeamer.newReader(
        CsvFormat.create(),       // The format to Read
        "s3://my-bucket/data",    // The location
        Movie.class               // The Avro class
    )
    .withFilePrefix("Movies")     // Filtering Files
)
```

Writing Avro & Parquet via TransBeamer

```
// Write to s3://my-bucket/results/ModMovies*.parquet, .avro
modifiedMovies.apply(
    TransBeamer.newWriter(
        AvroFormat.create(),       // The Format to write
        "s3://my-bucket/results", // The location (s3:, gs:)
        ModifiedMovie.class        // The Avro Class
    ).withFilePrefix("ModMovies") // A file prefix
);
modifiedMovies.apply(
    TransBeamer.newWriter(
        ParquetFormat.create(),  // Or any other format
        "gs://my-other-bucket/results",
        ModifiedMovie.class
    ).withFilePrefix("ModMovies")
);
```

Our Pipelines All Ended Up Like This



QUESTIONS?

Many Formats, One Data Lake

A Beginners Guide to Avro – Devon Peticolas (2023)

TransBeamer Library

BEAM SUMMIT NYC 2025

Peter Wagener, peter@sanuscorp.com
https://linkedin.com/in/peterjwagener