DATA+AI
SUMMIT 2022

Coral & Transport

Portable SQL & UDFs

For the interoperability of Spark and other engines

Walaa Eldin Moustafa
Senior Staff Software Engineer, LinkedIn

Wenye Zhang
Senior Software Engineer, LinkedIn

ORGANIZED BY databricks

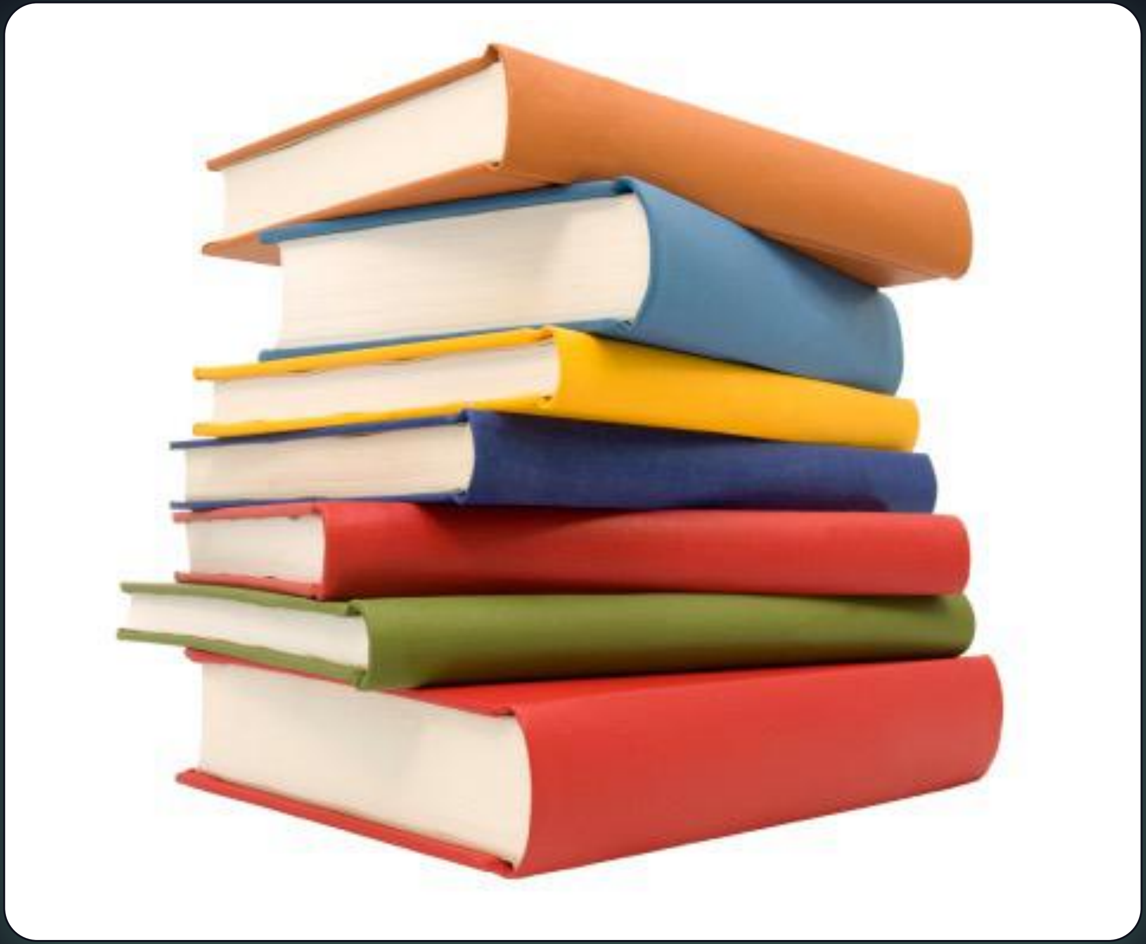
Modern Data Lake Architectures

Variety of query engines

Modern Data Lake Architectures

Variety of query languages

- Spark SQL
- Hive QL
- Presto SQL
- Trino SQL
- Flink SQL
- Other: Gremlin, SPARQL, Spark Scala, PySpark



Modern Data Lake Architectures

Variety of Data Sources

Tables

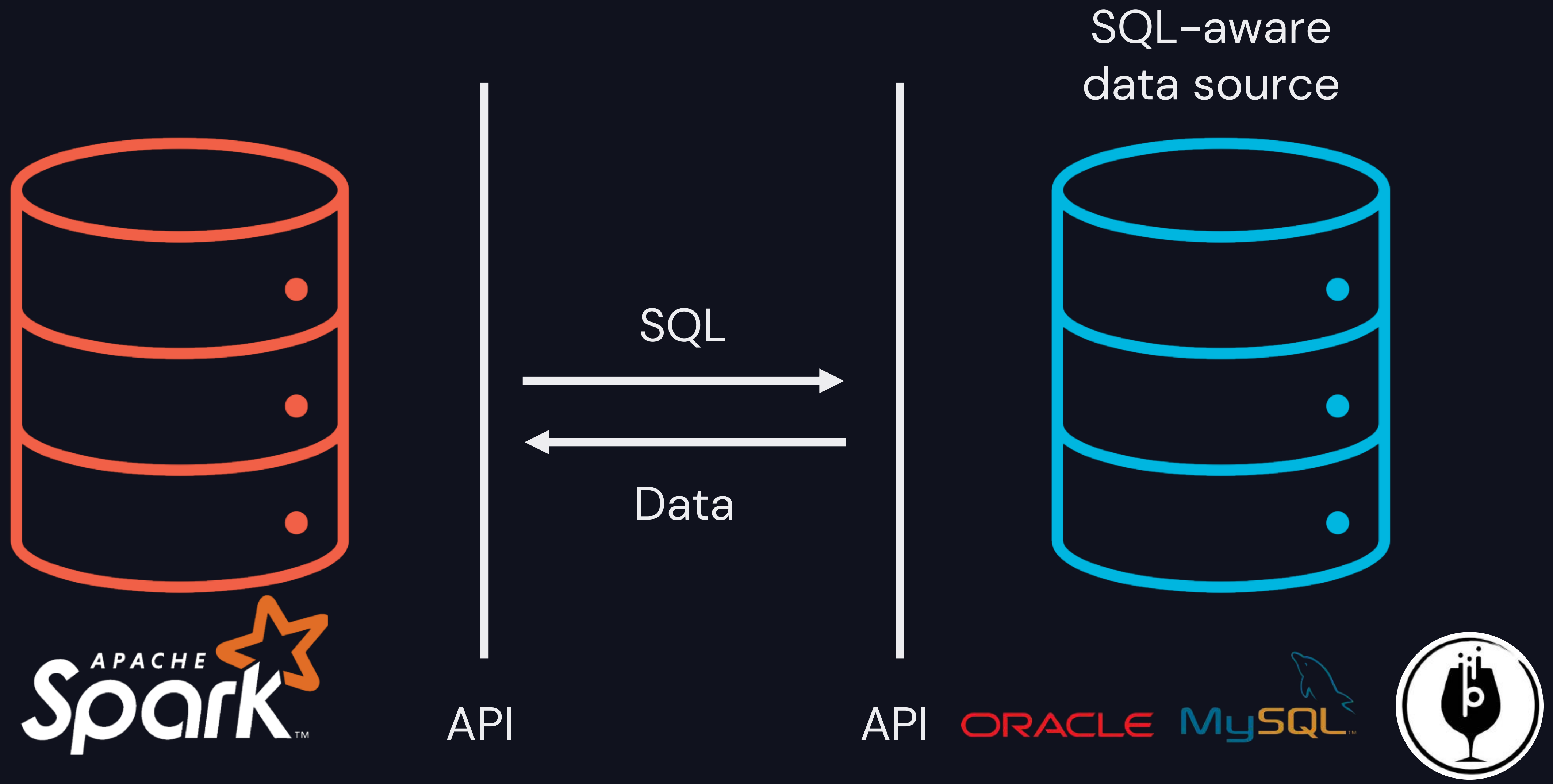
- Hive tables
- Delta Lake tables
- Iceberg tables
- Hudi tables
- Various file formats
  - Avro
  - ORC
  - Parquet

Views

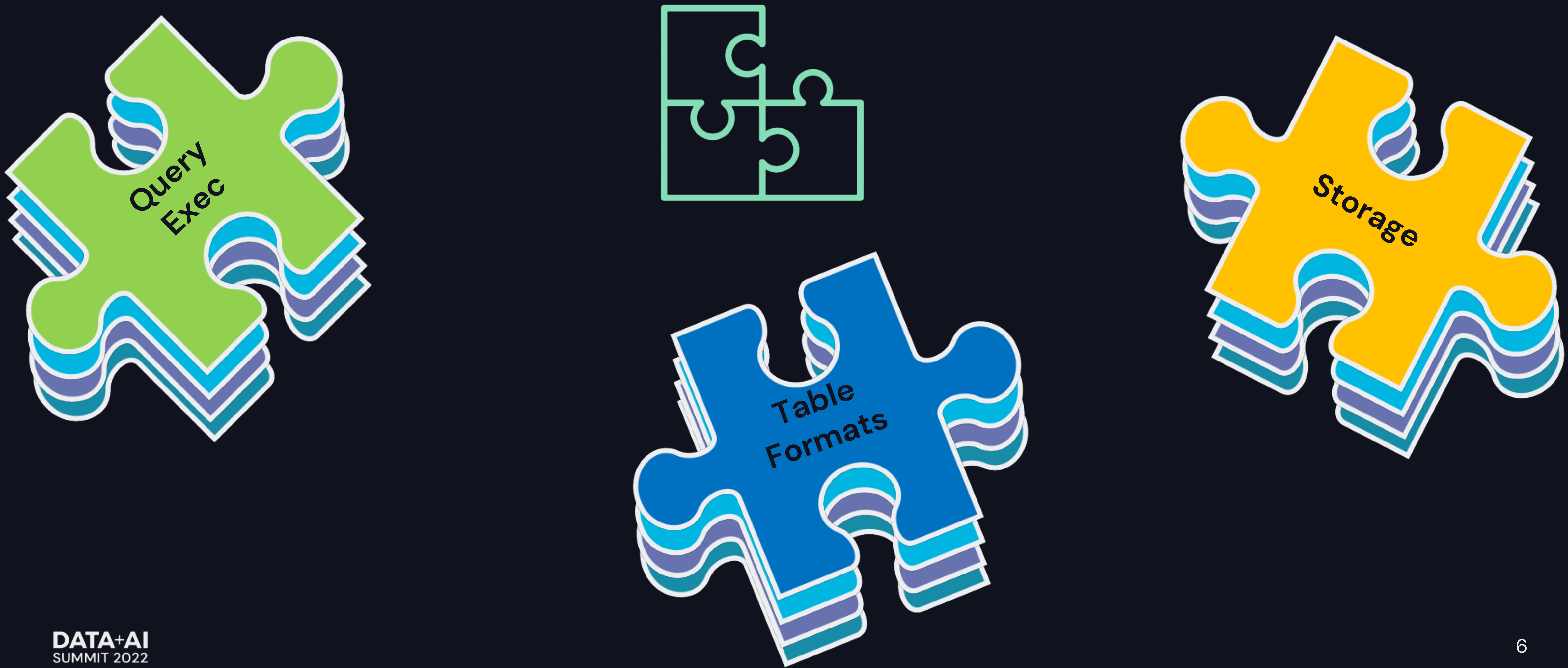
- Different query languages
- Different UDF APIs

Modern Data Lake Architectures

Even more data sources..

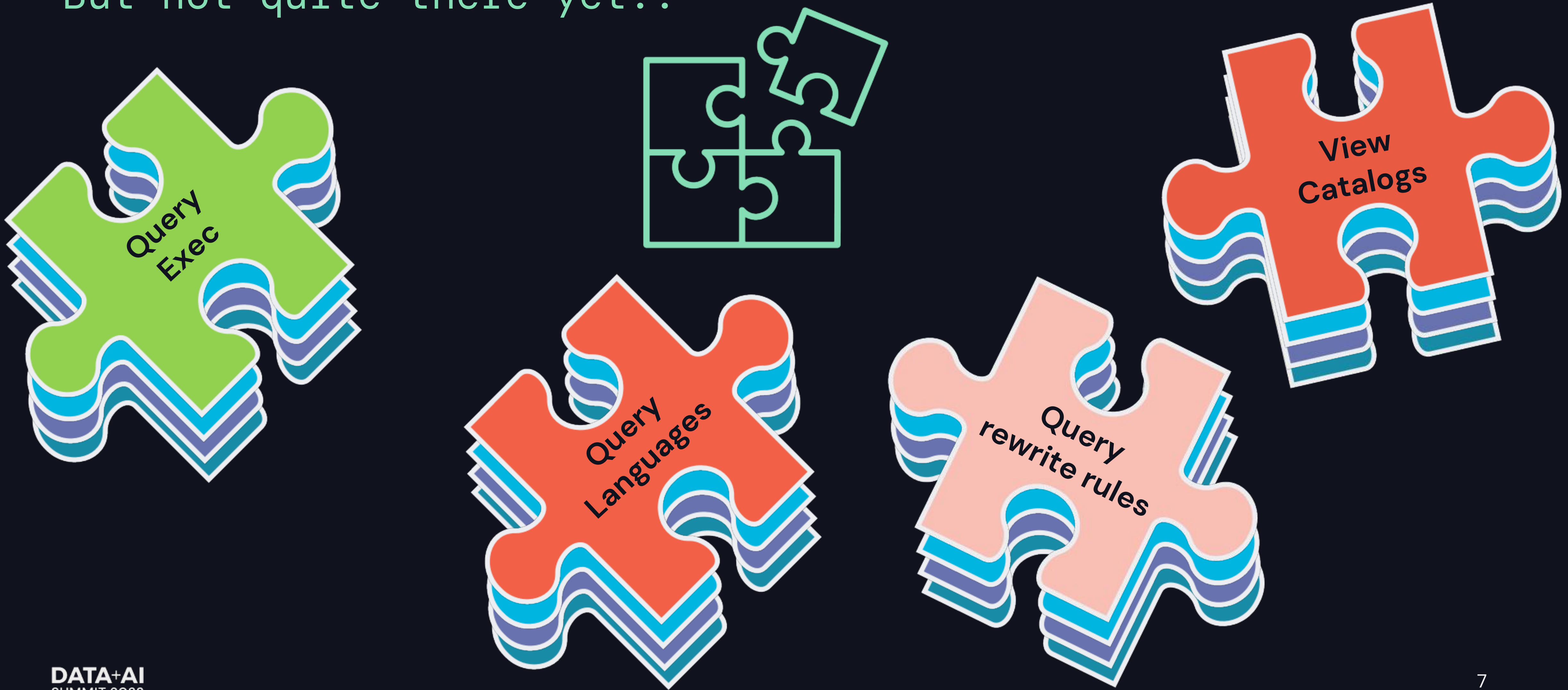


Composable Data Architectures



Composable Data Architectures

But not quite there yet..



Composable Data Architectures

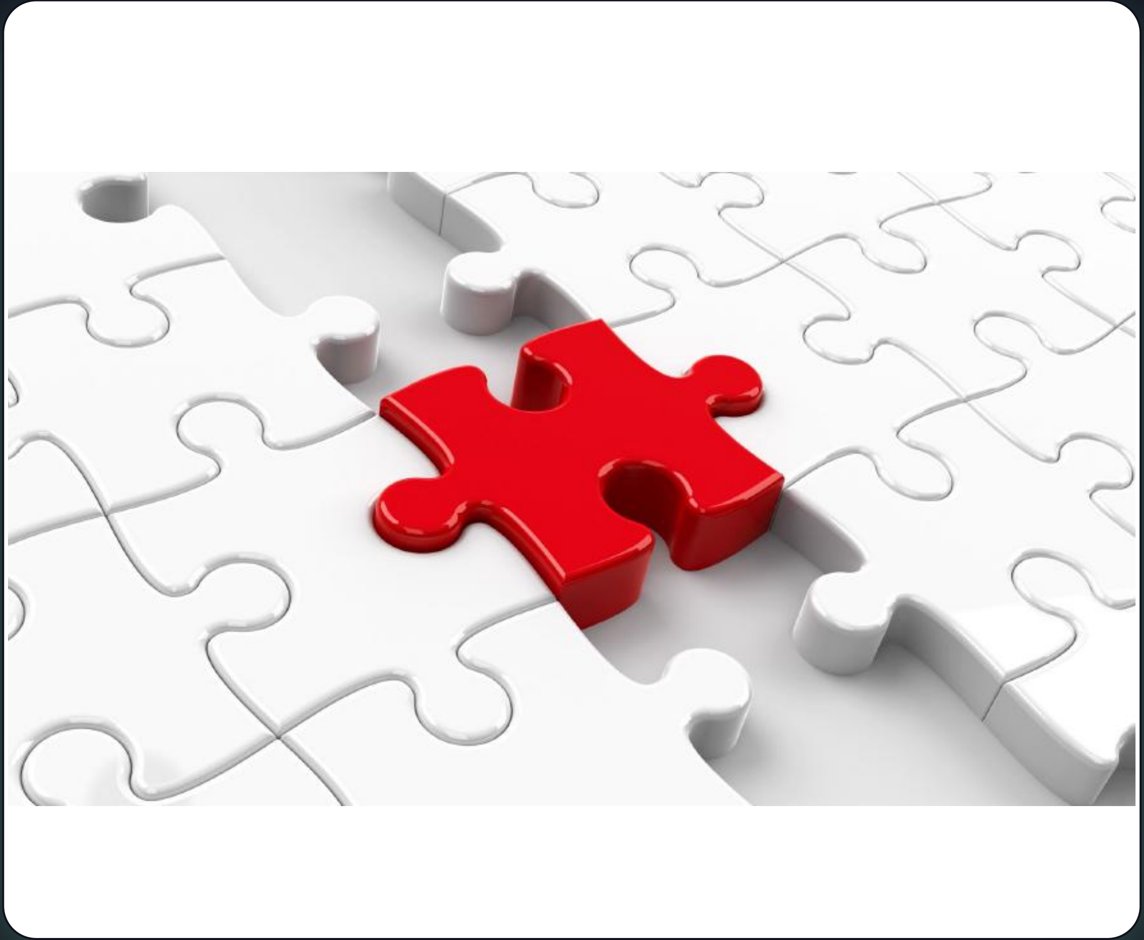
Logic interoperability

Common representation to capture

- Different SQL dialects
- View definitions
- Different engine plan representations
- SQL pushdown between engines
- Common query transformations

Adapters to transform

- From an input representation
- To an output representation



Composable Data Architectures

Coral

Common representation to capture

- Different SQL dialects
- View definitions
- Different engine plan representations
- SQL pushdown between engines
- Common query transformations

Adapters to transform

- From an input representation
- To an output representation

Composable Data Architectures

Transport

Common API to express

- UDF semantics
- Type validation and inference

Adapters to transform

- To any engine UDF

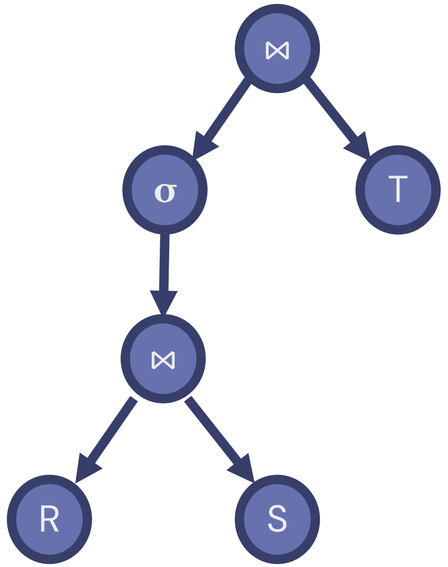
Coral

- Open-source project since 2020
- https://github.com/linkedin/coral
- Extends Calcite logical plan to represent logic
- Intermediate representation called Coral IR

Coral

IR, Transformations

- Coral IR captures query semantics using standard operators
- Supported Transformations
  - Hive QL (optionally Spark SQL) to Coral IR
  - Trino SQL to Coral IR (WIP)
  - Coral IR to Trino SQL
  - Coral IR to Spark SQL (optionally Hive QL)
  - Coral IR to Avro schema



Coral IR

Example

Spark SQL

Example Query

```
SELECT instr(R.x[0], 'foo')
FROM   R
WHERE  ! y
```

Operators

- instr(a, b): returns index of b in a
- x[i]: returns element i in array x, 0-based index
- ! y: negates y

Example

Trino SQL

Example Query

```
SELECT strpos(element_at(R.x, 1), 'foo')
FROM   R
WHERE  NOT y
```

Operators

- strpos(a, b): returns index of b in a
- element_at(x, i): returns element i in array x, 1-based index
- Not y: negates y

Transformations

Saprk QL to Coral IR conversion

Spark SQL		Coral IR
instr(x, y)	→	instr(x, y)
x[i]	→	x[i+1]
!x	→	NOT x

Transformations

Coral IR to Trino SQL conversion

Coral IR		Trino SQL
instr(x, y)	→	strpos(x, y)
x[i]	→	element_at(x, i)
NOT x	→	NOT x

Transformations

More complex transformations

- Lateral view joins
- User defined table functions
- Window functions
- Common table expressions

Integrations

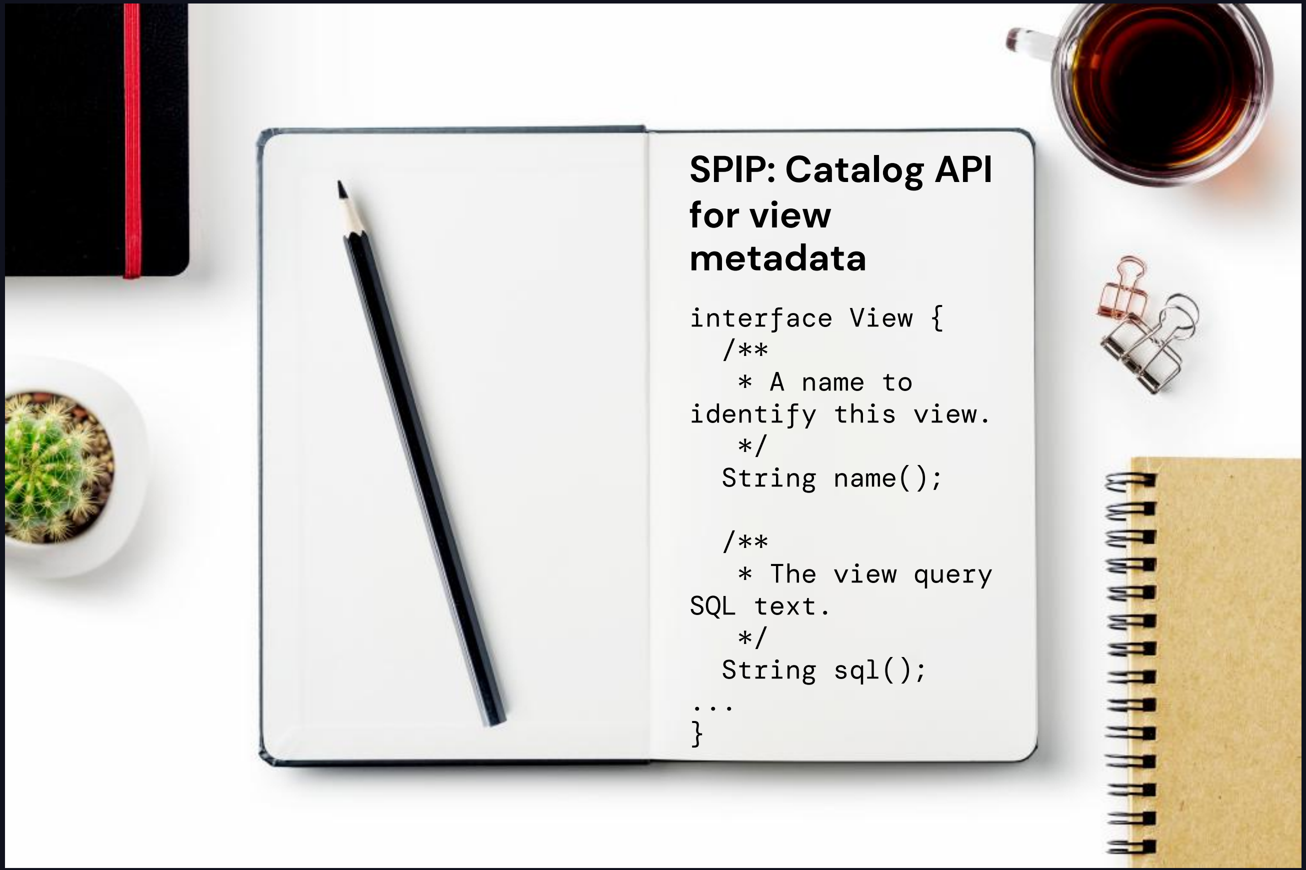
Notable integrations

- OSS Trino
  - Resolve Hive views in Trino
- LinkedIn's fork of Spark
  - Access Hive and Trino views (Trino in WIP)
  - Preserve view dataframe nullability, casing through inference
  - Perform schema evolution automatically
  - Register view UDFs automatically
- Spark Dataset API
  - Through Avro Specific record classes
  - Blog post: *Advanced schema management for Spark applications at scale*
  - https://engineering.linkedin.com/blog/2020/advanced-schema-management-for-spark

Apache Spark Integration

SPARK-31357

- Spark improvement to introduce top-level view abstractions
  - ViewCatalog API
  - View API
- Enable custom implementations for view SQL and schema resolution
- Envision Coral integration to Apache Spark through this API



Standalone mode

Coral-as-a-service

```
$ curl --header "Content-Type: application/json" \
  --request POST \
  --data '{
    "fromLanguage": "hive",
    "toLanguage": "trino",
    "query": "SELECT * FROM db1.airport"
  }' http://localhost:8080/api/translations/translate
```

Try it today! https://github.com/linkedin/coral

Future Extensions

- Spark catalyst plan to Coral IR
  - POC in Coral-Spark-Plan
  - Enables translation of all Spark APIs
    - Scala
    - Java
    - Python
- Common query rewrites
  - Materialized view substitution
  - Incremental view maintenance
  - Data governance (e.g., automatic obfuscation of PII)

Future Extensions

SPARK-37960

- Spark data source integration
  - Push functions to data sources
    - Delta Lake
    - Iceberg
  - Push SQL expressions to SQL data sources
    - Trino
    - Presto
    - Pinot

Transport

Translatable, Portable UDFs

Motivation

- SQL has pretty well-understood IR: Relational Algebra
  - Scan, Filter, Project, Join, Group By, etc
- UDFs
  - Opaque
  - Use imperative language
  - Not portable or translatable

UDF Denormalization

Duplication

Multiple versions of the same UDF. Not clear which is the source of truth.

Inconsistency

Duplicate implementations can diverge causing data inconsistency

Low Productivity

Developers need to learn multiple APIs, implement same logic multiple times.

Low Performance

In some cases, use tuple conversion adapters to enable portability.

A UDF Primer

UDFs 101

Example Hive UDF

```
public class Instr extends GenericUDF {

  @Override
  public ObjectInspector initialize(ObjectInspector[] args) {
    if (arguments.length != 2) {
      error();
    }

    for (int i = 0; i < arguments.length; i++) {
      if (args[i].getCategory() != PrimitiveCategory.STRING) {
        error();
      }
    }
    return PrimitiveObjectInspectorFactory.writableIntObjectInspector;
  }

  @Override
  public Object evaluate(DeferredObject[] args) {
    if (arguments[0].get() == null || arguments[1].get() == null) {
      return null;
    }

    Text text = (Text) (arguments[0].get());
    Text subtext = (Text) (arguments[1].get());
    return instr(text, subtext);
  }
}
```

Example Trino UDF

```
@ScalarFunction("array_remove")
public final class ArrayRemoveFunction {

    private ArrayRemoveFunction() {}

    @TypeParameter("E")
    @SqlType("array(E)")
    public static Block remove(@OperatorDependency(operator = EQUAL,
                    returnType = StandardTypes.BOOLEAN,
                    argumentTypes = {"E", "E"}) MethodHandle equalsFunction,
            @TypeParameter("E") Type type,
            @SqlType("array(E)") Block array,
            @SqlType("E") long value) {
        return remove(equalsFunction, type, array, (Object) value);
    }

    @TypeParameter("E")
    @SqlType("array(E)")
    public static Block remove(@OperatorDependency(operator = EQUAL,
            returnType = StandardTypes.BOOLEAN,
            argumentTypes = {"E", "E"}) MethodHandle equalsFunction,
            @TypeParameter("E") Type type,
            @SqlType("array(E)") Block array,
            @SqlType("E") boolean value) {
        return remove(equalsFunction, type, array, (Object) value);
    }
}
```

UDF APIs

- API Complexity
  - APIs expose low-level details of engines
  - Data types may not intuitively map to SQL type-system
- API Disparity
  - APIs differ in what to expect from developer
  - APIs differ in features they can provide

Transport UDFs

```java
public class MapFromTwoArrays
    extends StdUDF2<StdArray, StdArray, StdMap> {

  @Override
  public List<String> getInputParameterSignatures() {
    return ImmutableList.of(
        "array(K)",
        "array(V)"
    );
  }

  @Override
  public String getOutputParameterSignature() {
    return "map(K,V)";
  }

  @Override
  public StdMap eval(StdArray a1, StdArray a2) {
    StdMap map = getStdFactory().createMap(
        getOutputParameterSignature());
    for (int i = 0; i < a1.size(); i++) {
      map.put(a1.get(i), a2.get(i));
    }
    return map;
  }
}
```

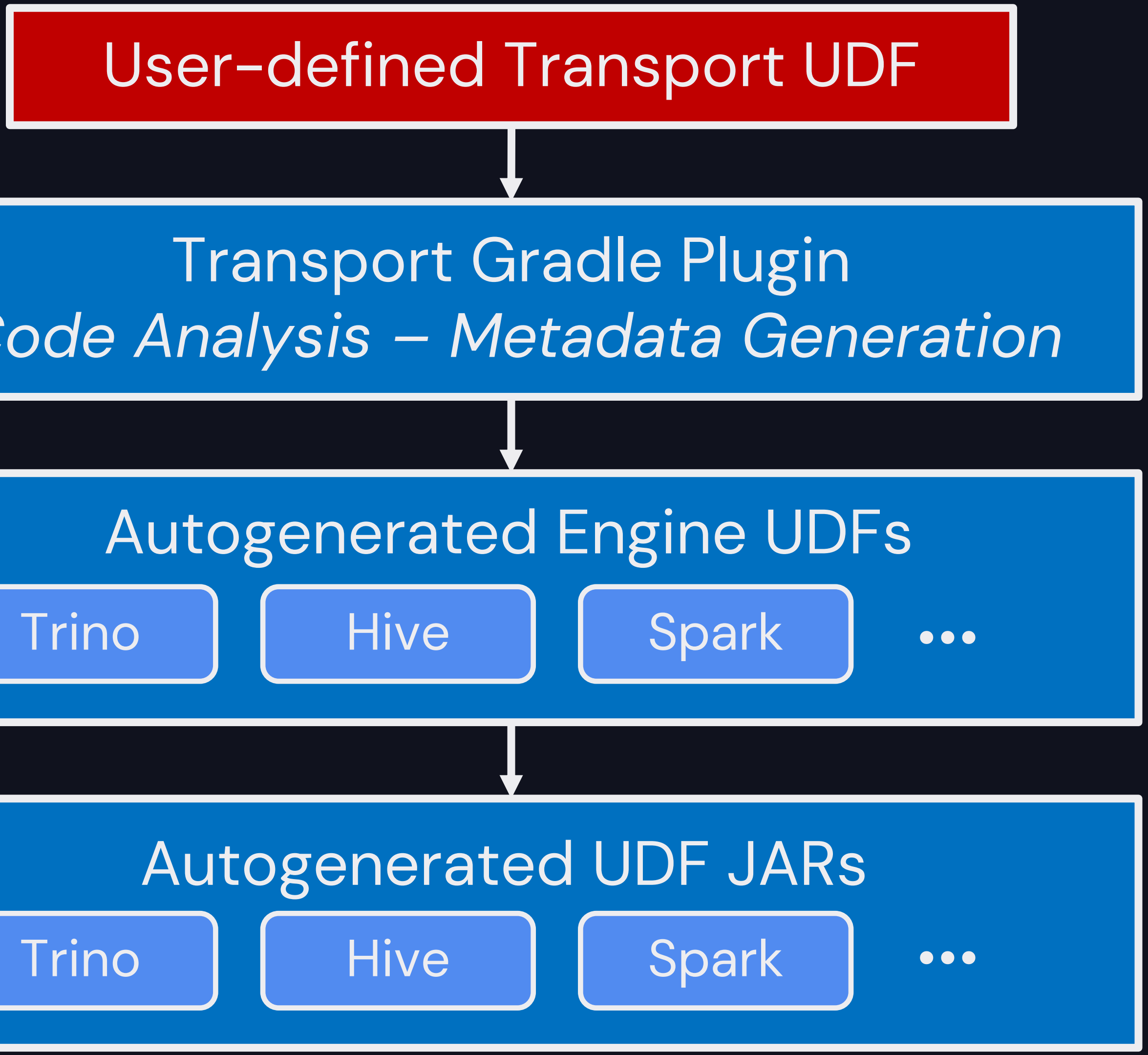
Then What?

1. bash

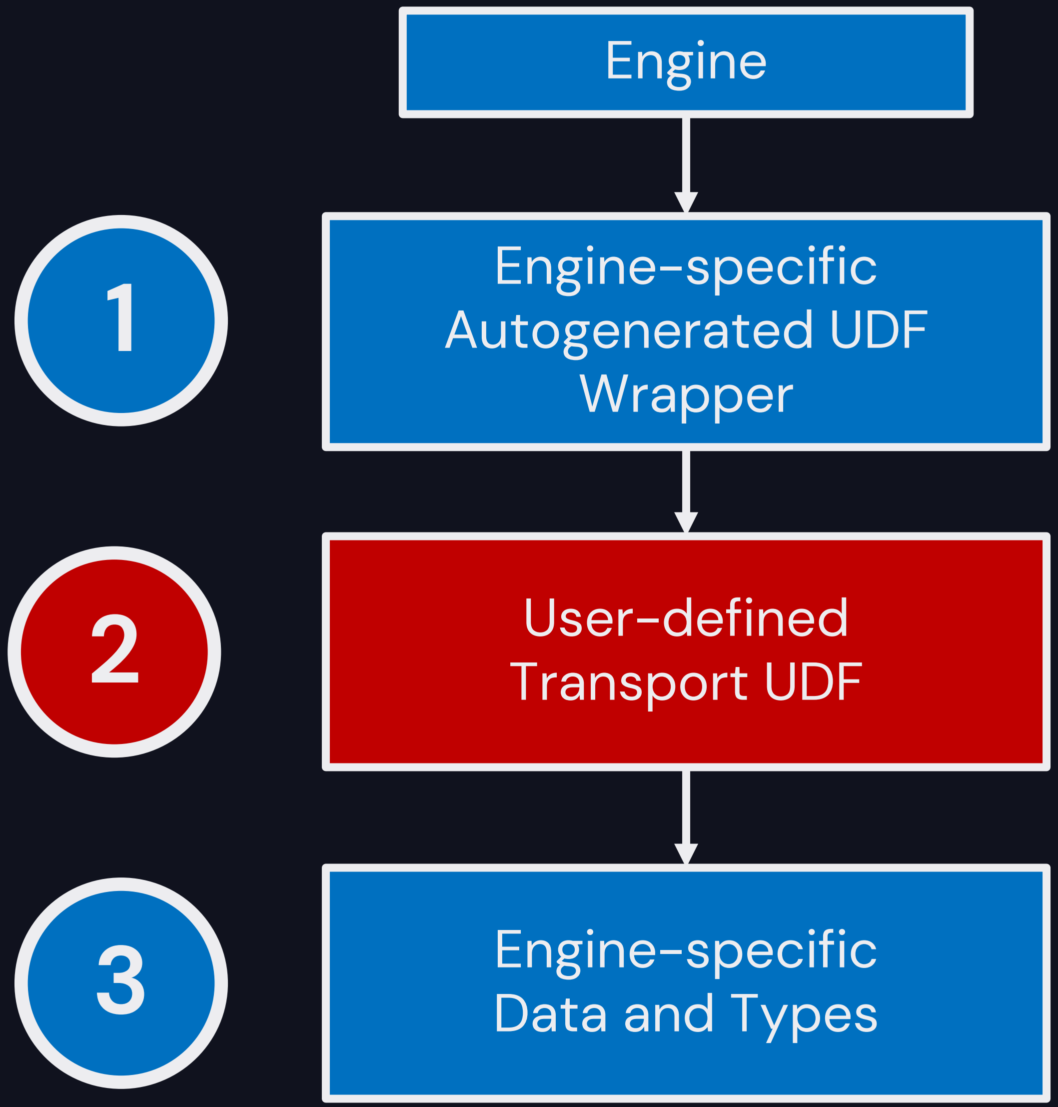
```
> gradle build


> ls build/my-udfs/libs
  my-udfs-trino.jar
  my-udfs-hive.jar
  my-udfs-spark.jar
```

Auto-generated UDFs



Architecture



Conclusions

Transport UDFs API

Simple

Only implement what is needed to define logic. No boilerplate code.

Feature-rich

Declarative type signatures with generics. getRequiredFiles() support.

Translatable

Can run on multiple platforms.
Code specific to platform is auto-generated.

Performant

Direct access to native platform data.