Unstructured data analytics using polymorphic table functions in Trino

Yonghwan Lee - Data Engineer @ SK telecom

Agenda

- About us
- Working with unstructured data
- Change your perspective
- What is a Polymorphic Table Function?
- Python File Query
- Python Meta Query
- Scripting PTF
- What’s next?
- Q & A

About us

Largest mobile operator in South Korea

About us

Largest mobile operator in South Korea

Huge advocate of Trino

About us

Largest mobile operator in South Korea

Huge advocate of Trino

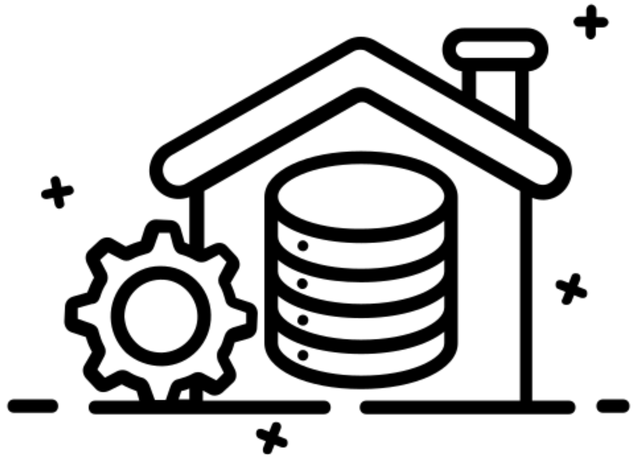
Partner with Starburst

Working with unstructured data

History of Data Engineering Architecture

Working with unstructured data

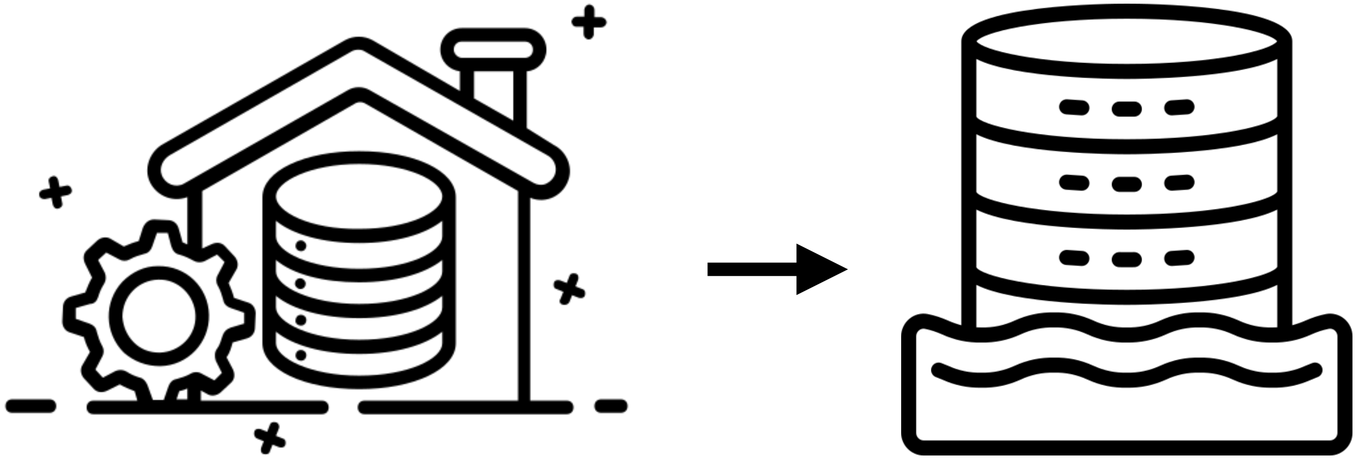
History of Data Engineering Architecture



WAREHOUSE

Working with unstructured data

History of Data Engineering Architecture

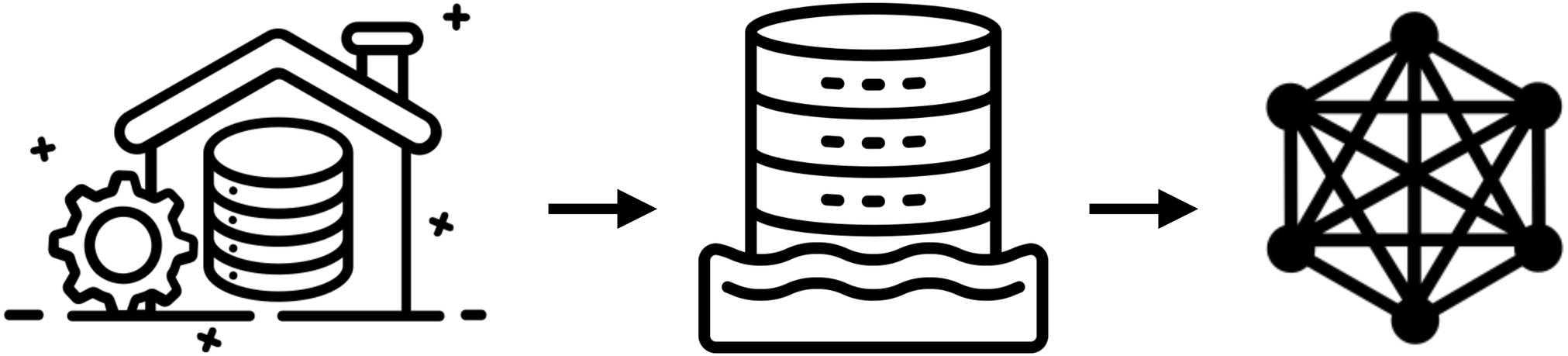


WAREHOUSE

LAKE

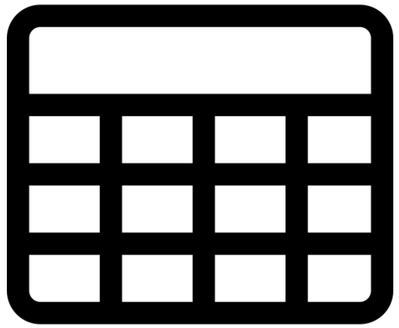
Working with unstructured data

History of Data Engineering Architecture



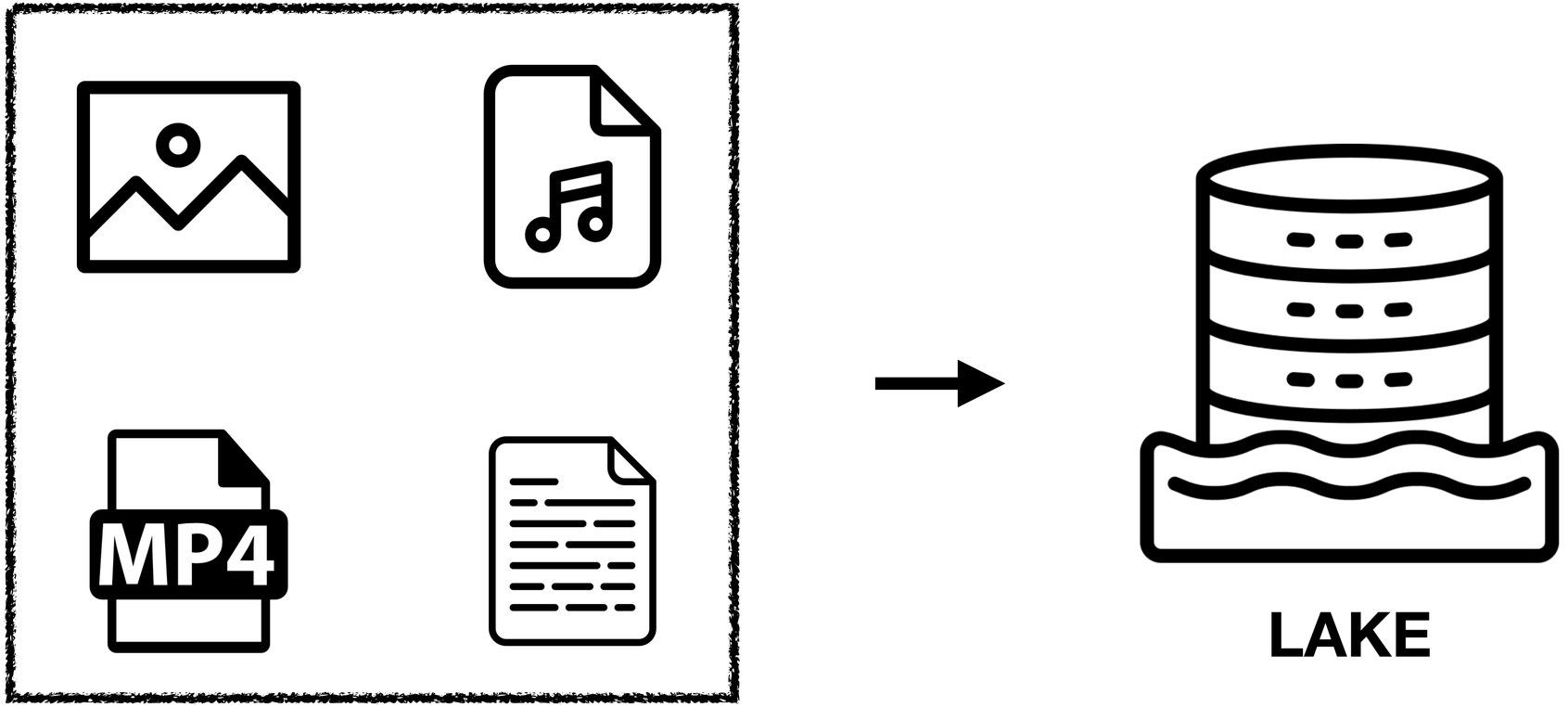
WAREHOUSE **LAKE** **MESH**

Working with unstructured data

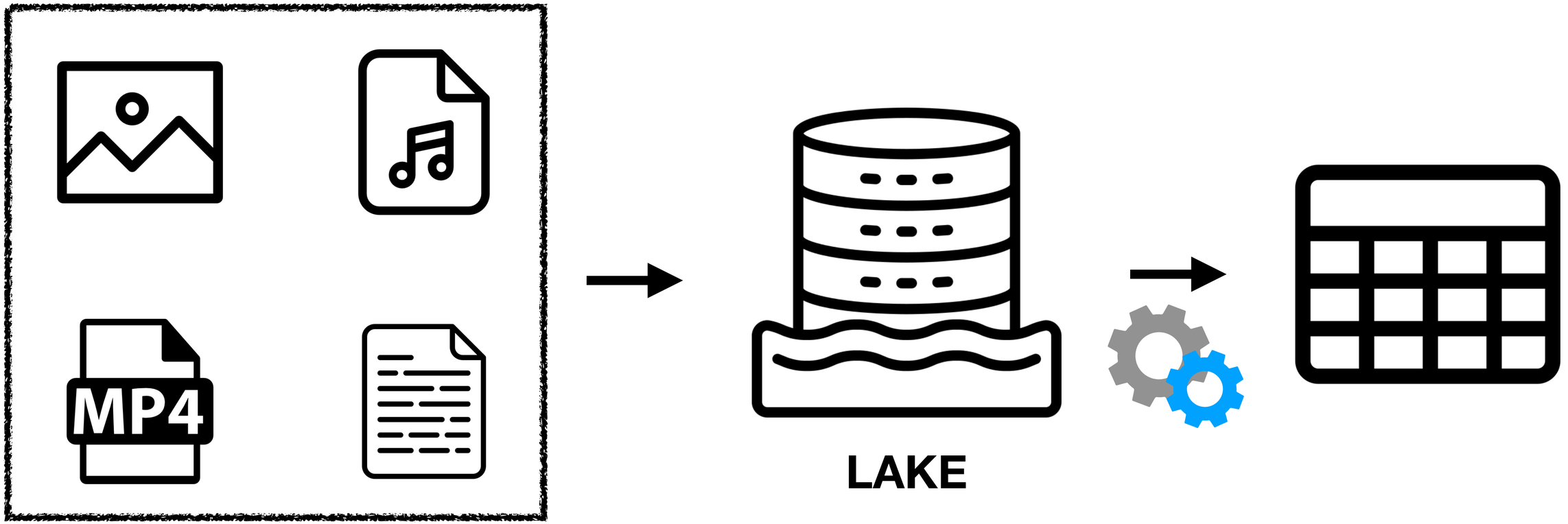


Warehouse

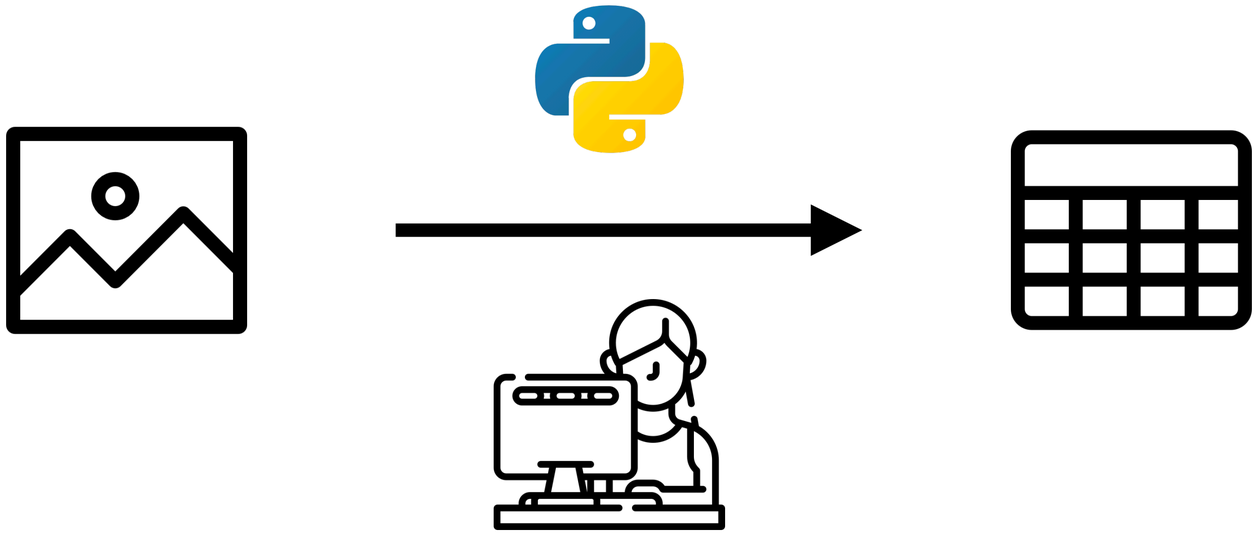
Working with unstructured data



Working with unstructured data



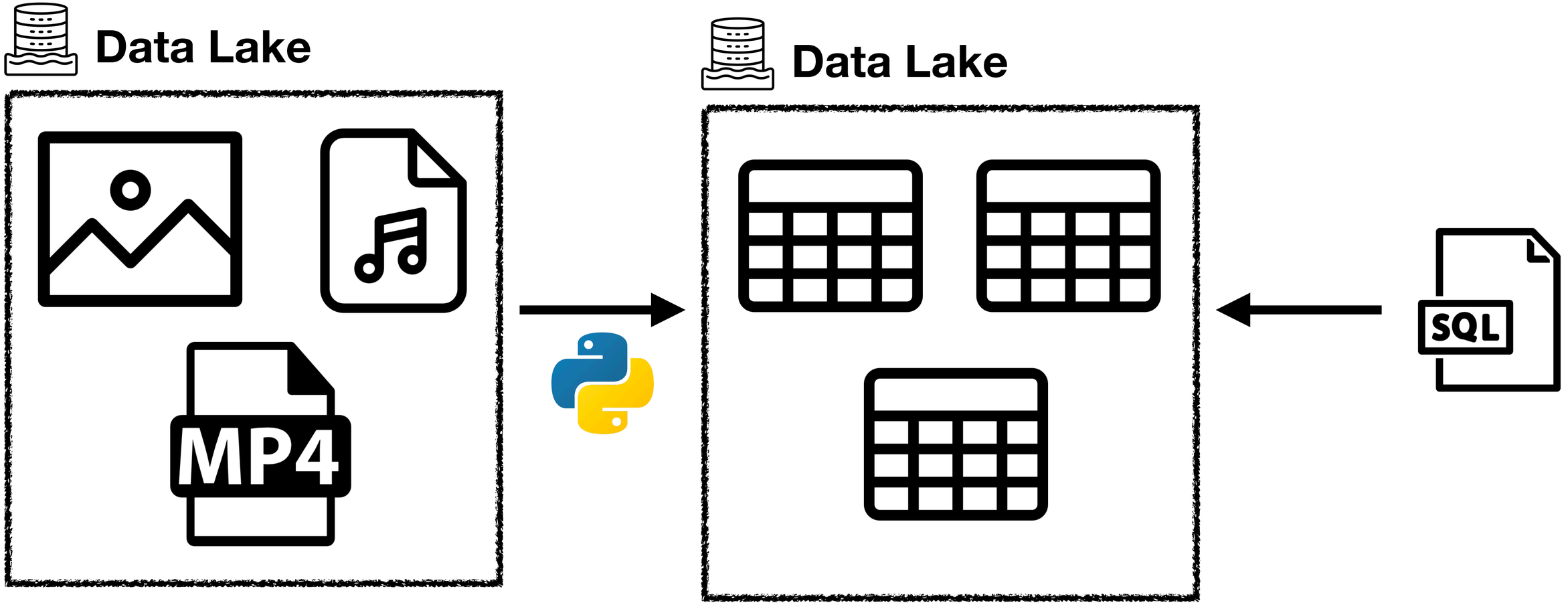
Working with unstructured data



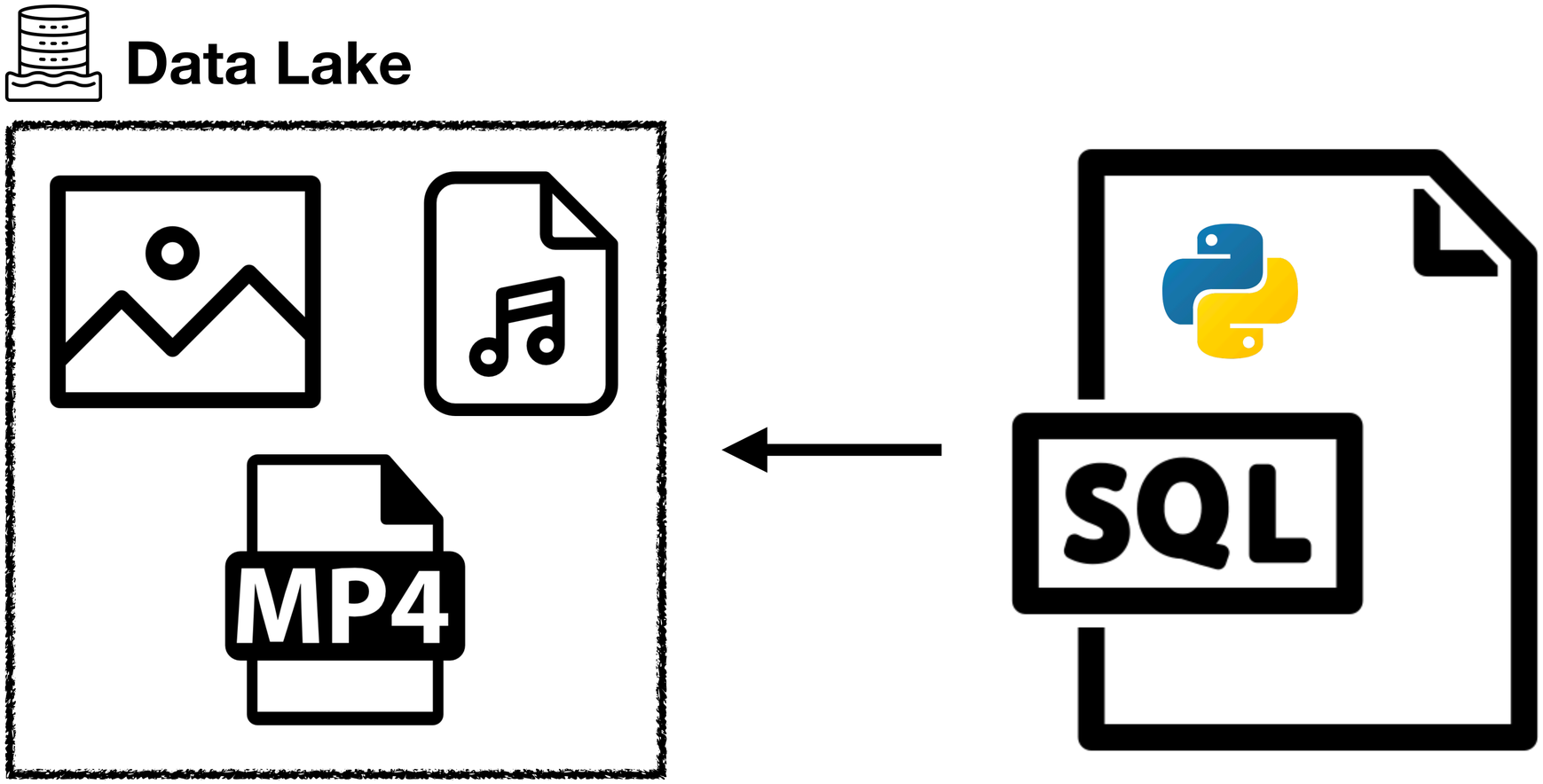
Working with unstructured data

Working with unstructured data

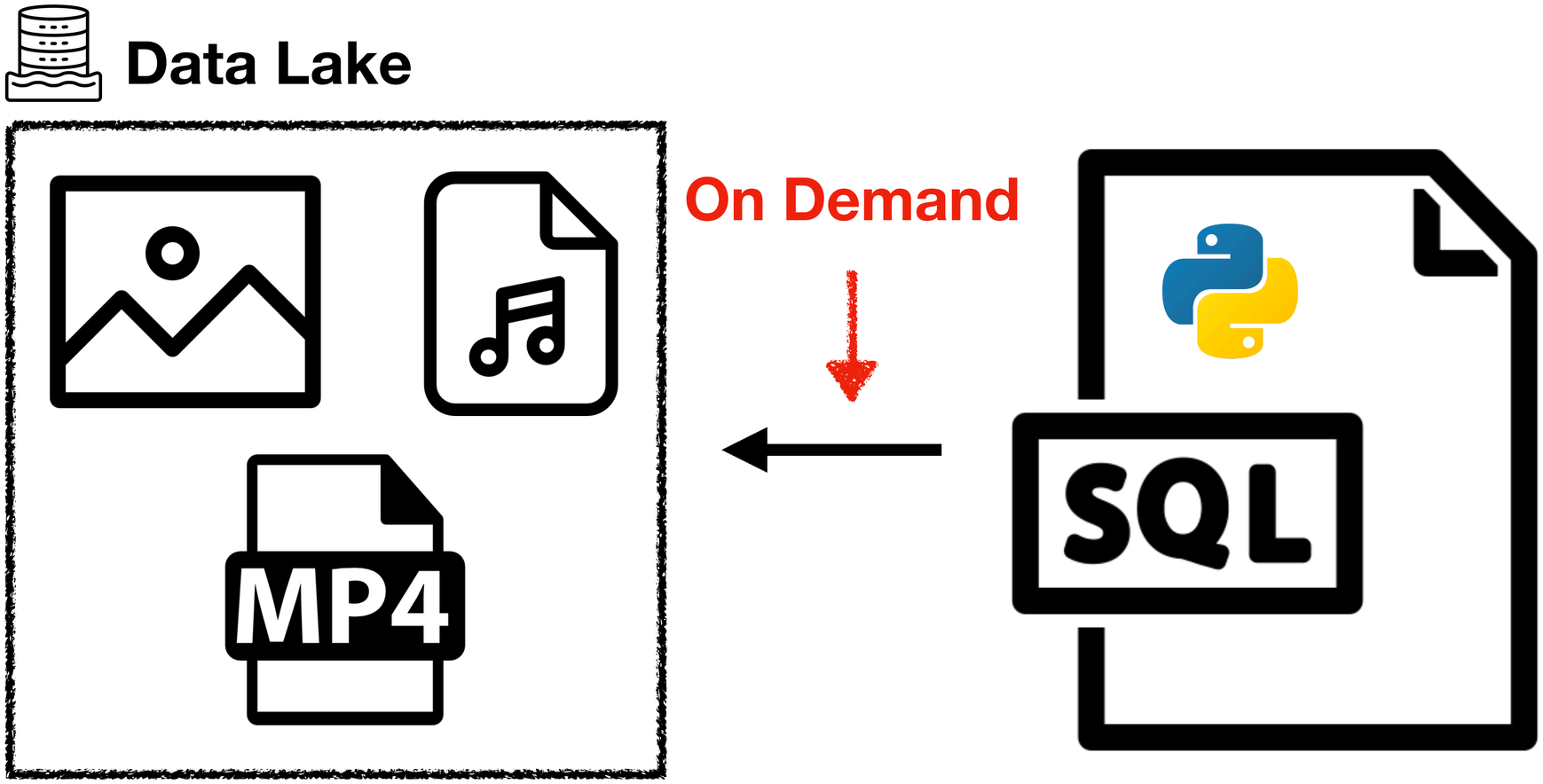
Change your perspective



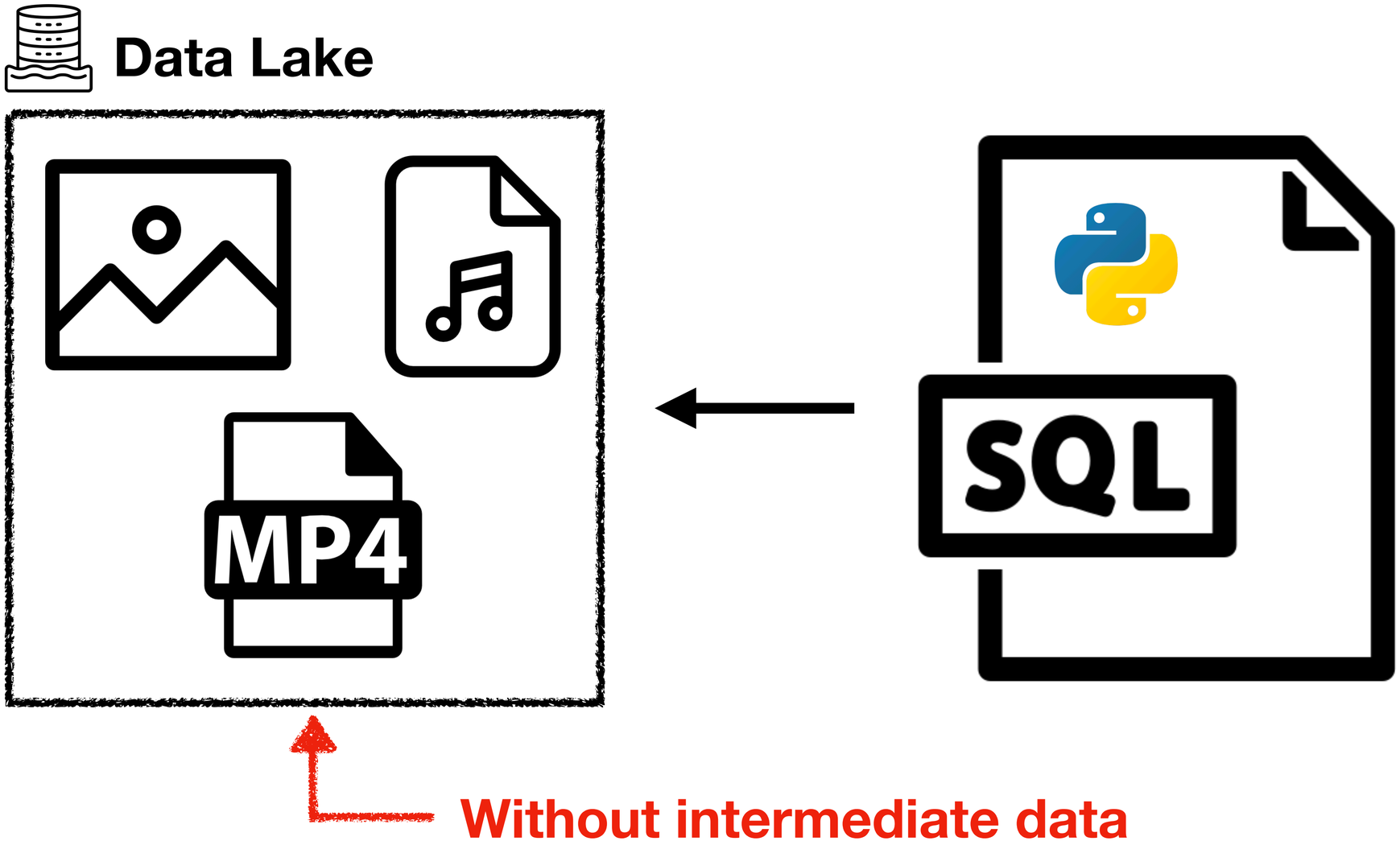
Change your perspective



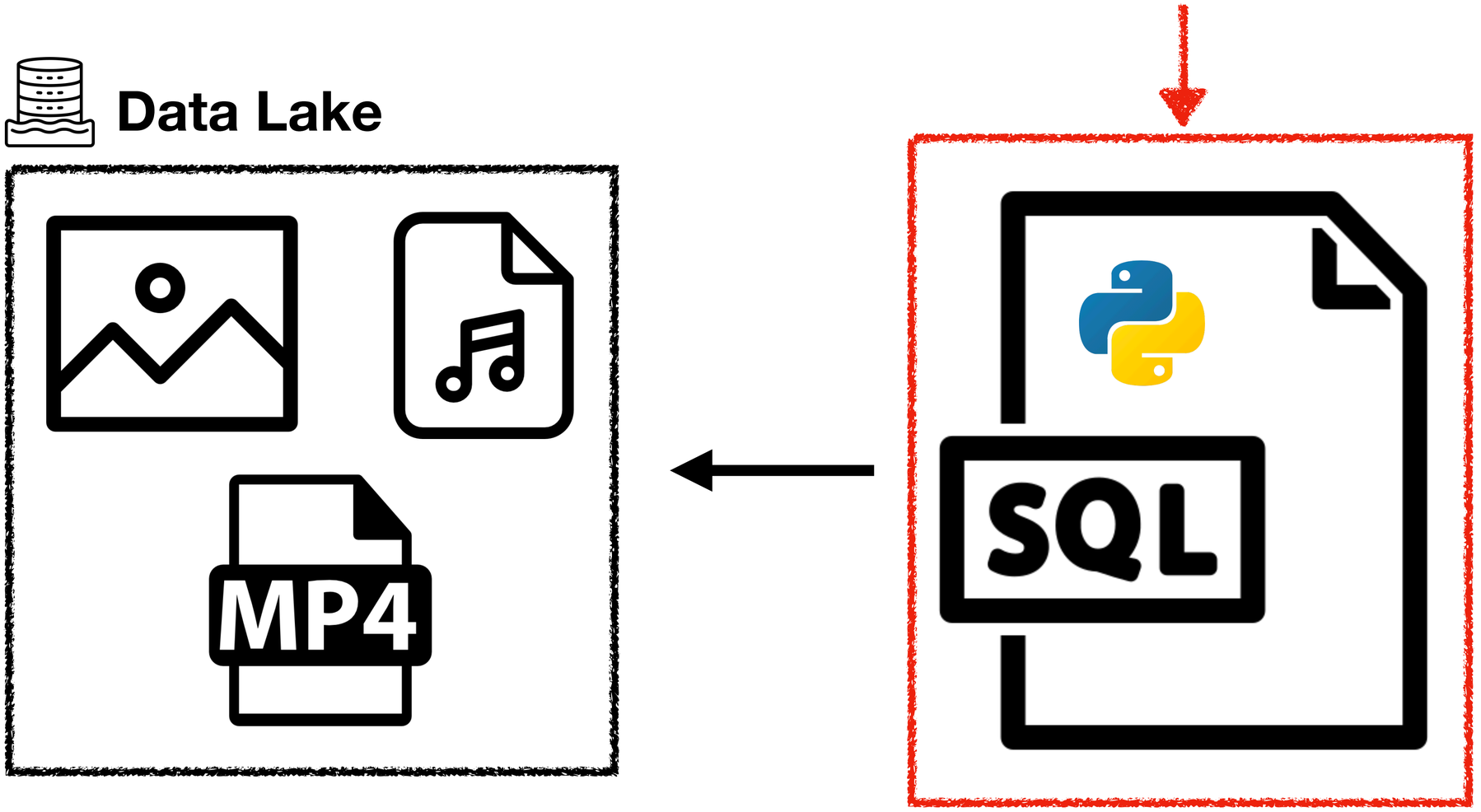
Change your perspective



Change your perspective



Change your perspective

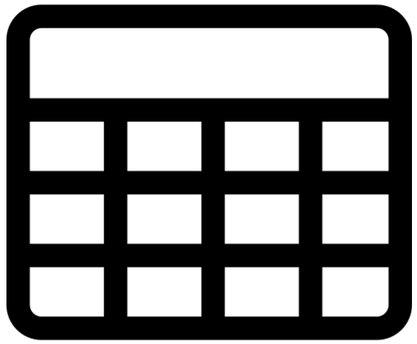


What is a polymorphic table function

- User Defined Function that returns table

F(X)

returns



Polymorphic Table Function

Table (or Dataset)

What is a polymorphic table function

- User Defined Function that returns table
- ISO Standard

What is a polymorphic table function

- User Defined Function that returns table
- ISO Standard
- Introduced in Trino 381

Diving into polymorphic table functions with Trino

Jul 22, 2022 • Kasia Findeisen, Brian Olsen, and Cole Bowden

In the Trino community, we know that being the coolest query engine is a tough job. We boldly face the intricacies of the SQL standard to bring you the newest and most powerful features. Today, we proudly announce that as of release 381, Trino is on its way to full support for polymorphic table functions (PTFs).

In this blog post, we are explaining the concept of table functions and exploring how they can be leveraged. We also look at what we have already implemented, and take a sneak peek into the future.

Table of contents

- Definition time
- OK, but why are we so excited?
- What is available in Trino today?
- Big ideas
- Looking forward

What is a polymorphic table function

Built-in PTFs

- JDBC : query
- ElasticSearch: raw_query
- Common: exclude_columns, sequence

What is a polymorphic table function

Built-in PTFs

- JDBC : query
- ElasticSearch: raw_query
- Common: exclude_columns, sequence

What is a polymorphic table function

Usual SELECT Statement

```
SELECT
  *
FROM
  postgresql.tpch.nation
WHERE
  nationkey = 0
```

What is a polymorphic table function

Usual SELECT Statement

```
SELECT
  *
FROM
  postgresql.tpch.nation
WHERE
  nationkey = 0
```

What is a polymorphic table function

SELECT Statement with PTF

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0'
  )
);
```

What is a polymorphic table function

SELECT Statement with PTF

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0'
  )
);
```

What is a polymorphic table function

SELECT Statement with PTF

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0'
  )
);
```

What is a polymorphic table function

SELECT Statement with PTF

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0'
  )
);
```

What is a polymorphic table function

SELECT Statement with PTF

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0'
  )
);
```

What is a polymorphic table function

Why we use PTF?

Overcome the limitations of query processing

What is a polymorphic table function

Why we use PTF?

Overcome the limitations of query processing

Create entirely new ways to generate data

What is a polymorphic table function

Overcome limitations of query processing

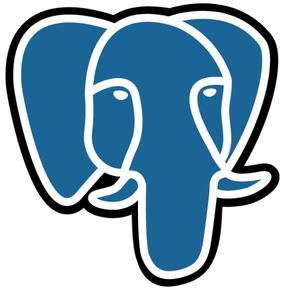
```
SELECT
  *
FROM
  postgresql.public.nation
WHERE
  nationkey = 0
```

What is a polymorphic table function

Overcome limitations of query processing

```
SELECT
  *
FROM
  postgresql.public.nation
WHERE
  nationkey = 0
```

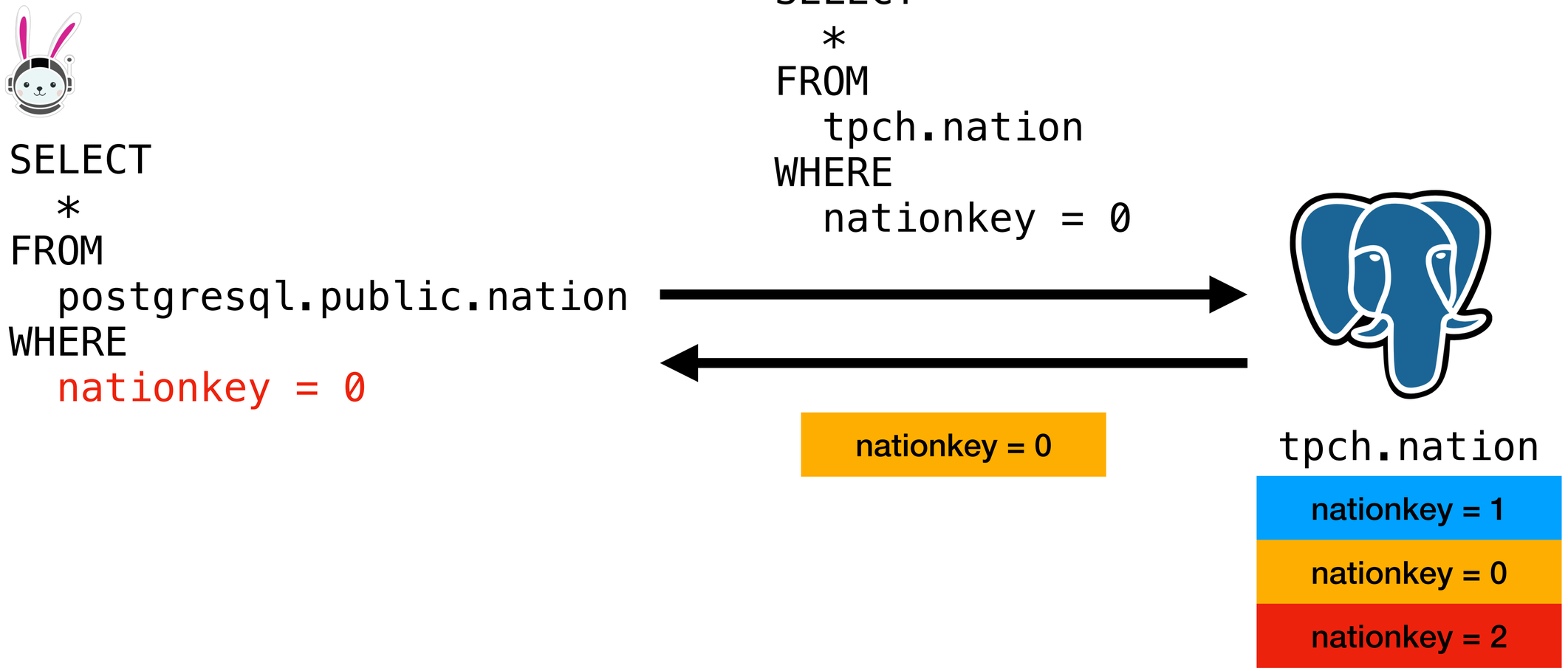
```
SELECT
  *
FROM
  tpch.nation
WHERE
  nationkey = 0
```



tpch.nation

What is a polymorphic table function

Overcome limitations of query processing



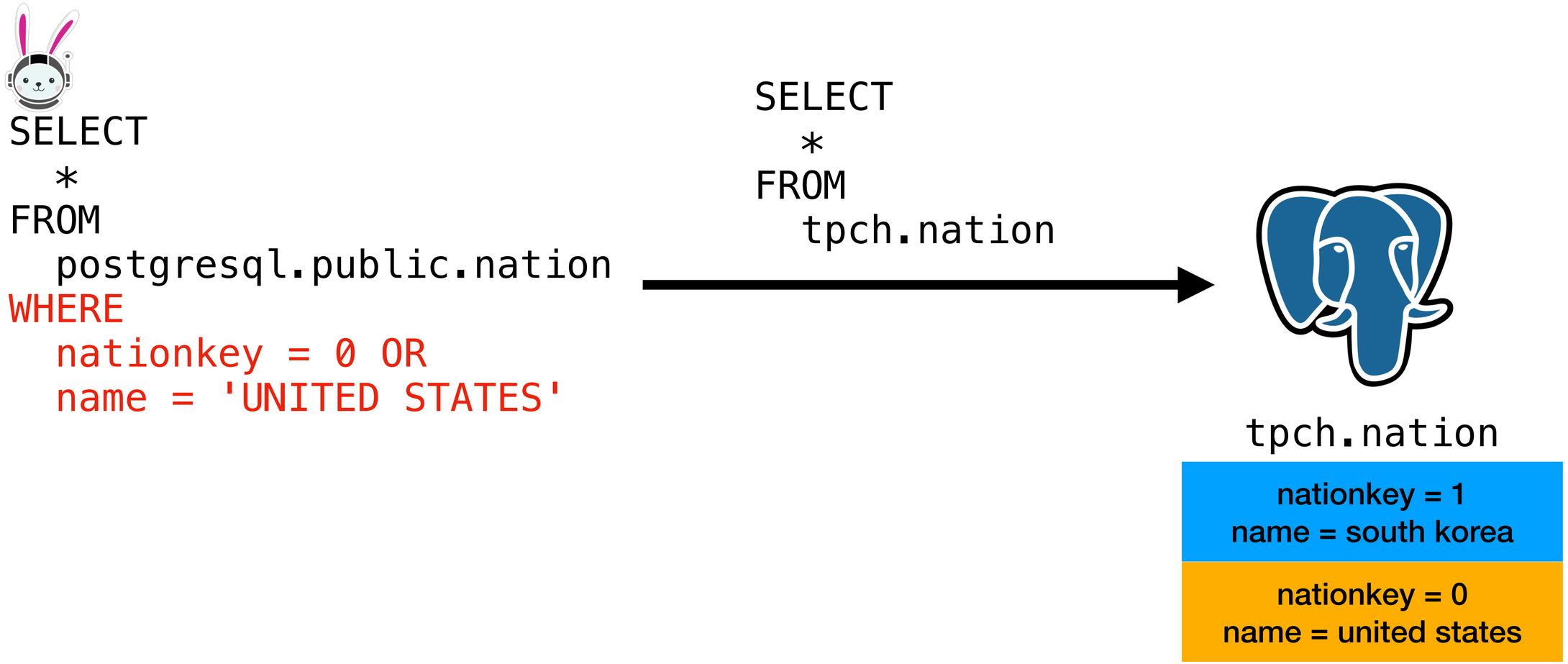
What is a polymorphic table function

Overcome limitations of query processing

```
SELECT
  *
FROM
  postgresql.public.nation
WHERE
  nationkey = 0 OR
  name = 'UNITED STATES'
```

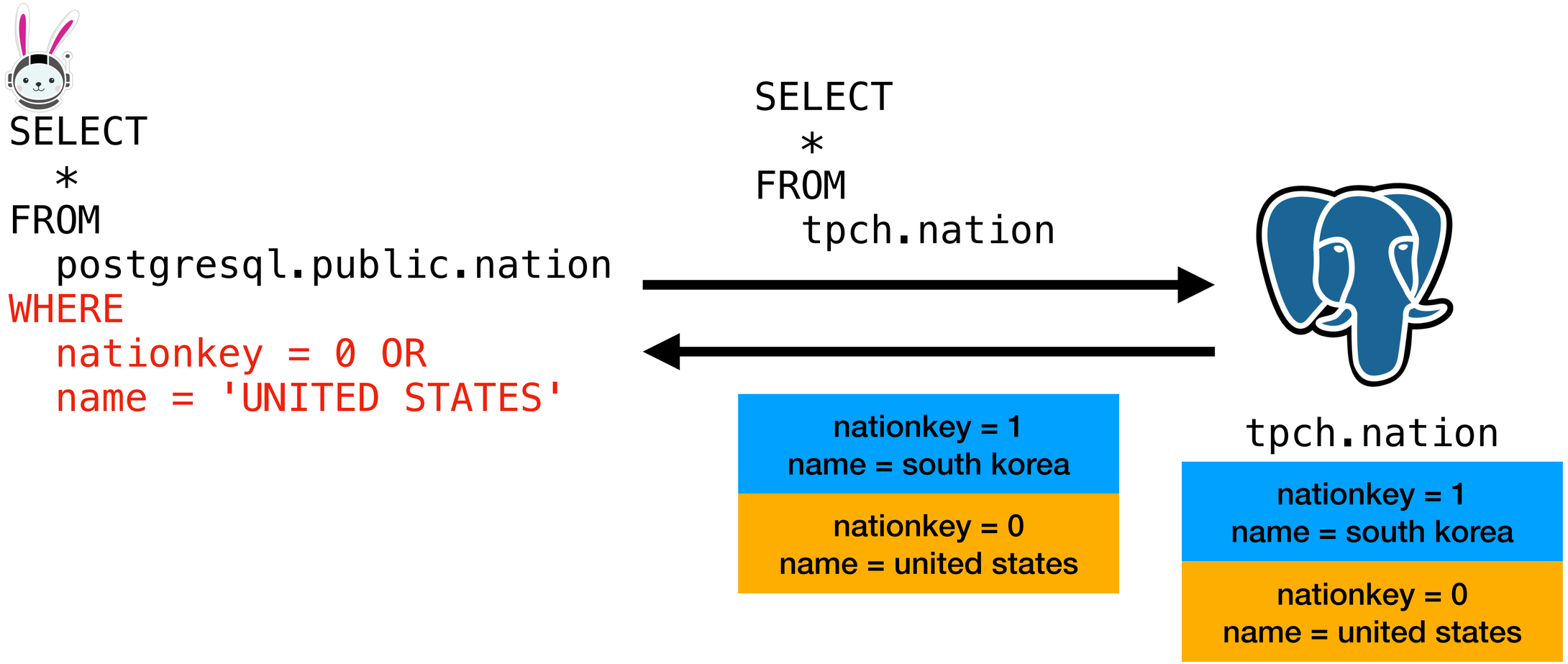
What is a polymorphic table function

Overcome limitations of query processing



What is a polymorphic table function

Overcome limitations of query processing



What is a polymorphic table function

Overcome limitations of query processing

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
          name
         FROM
          tpch.nation
         WHERE
          nationkey = 0 OR
          name = "United States"'
  )
);
```

SK telecom

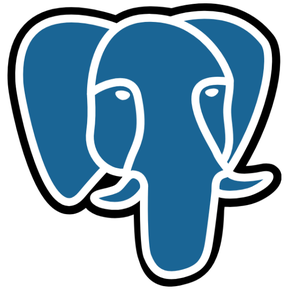
trino

What is a polymorphic table function

Overcome limitations of query processing

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
           name
         FROM
           tpch.nation
         WHERE
           nationkey = 0 OR
           name = "United States"'
  )
);
```

```
SELECT
  name
FROM
  tpch.nation
WHERE
  nationkey = 0 OR
  name = "United States"
```



tpch.nation

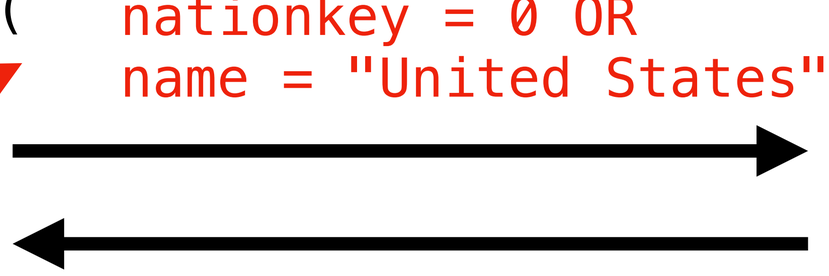
nationkey = 1 name = south korea
nationkey = 0 name = united states

What is a polymorphic table function

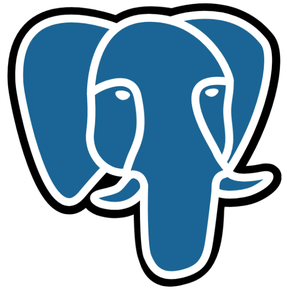
Overcome limitations of query processing

```
SELECT
  *
FROM
  TABLE(
    postgresql.system.query(
      query =>
        'SELECT
           name
         FROM
           tpch.nation
         WHERE
           nationkey = 0 OR
           name = "United States"'
  )
);
```

```
SELECT
  name
FROM
  tpch.nation
WHERE
  nationkey = 0 OR
  name = "United States"
```



nationkey = 0
name = united states



tpch.nation

nationkey = 1
name = south korea

nationkey = 0
name = united states

What is a polymorphic table function

Entirely new ways to generate data

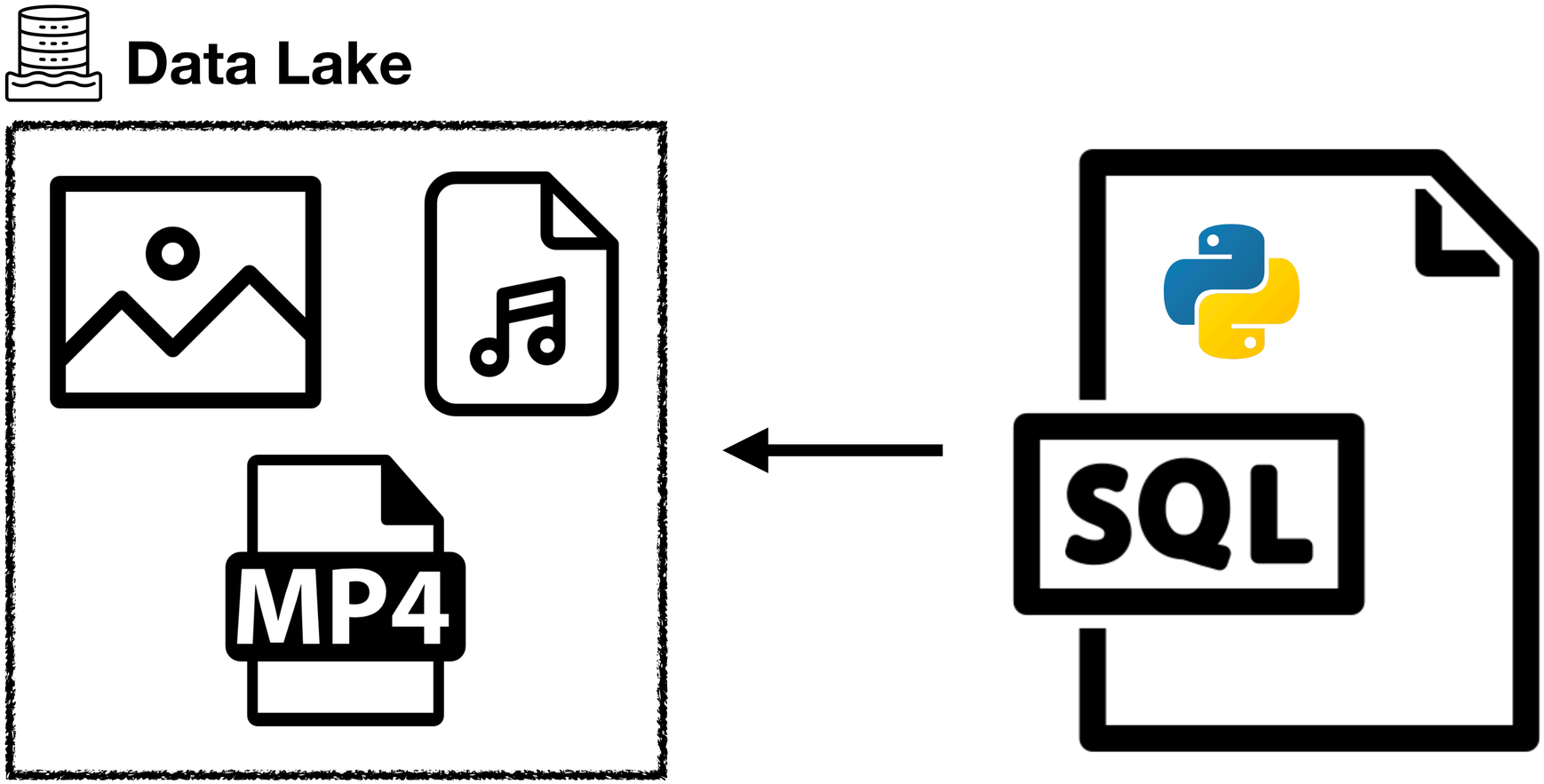
```
SELECT
  *
FROM
  TABLE(
    sequence(
      start => 1000000,
      stop => -2000000,
      Step => -3
    )
  );
```

SK telecom

trino

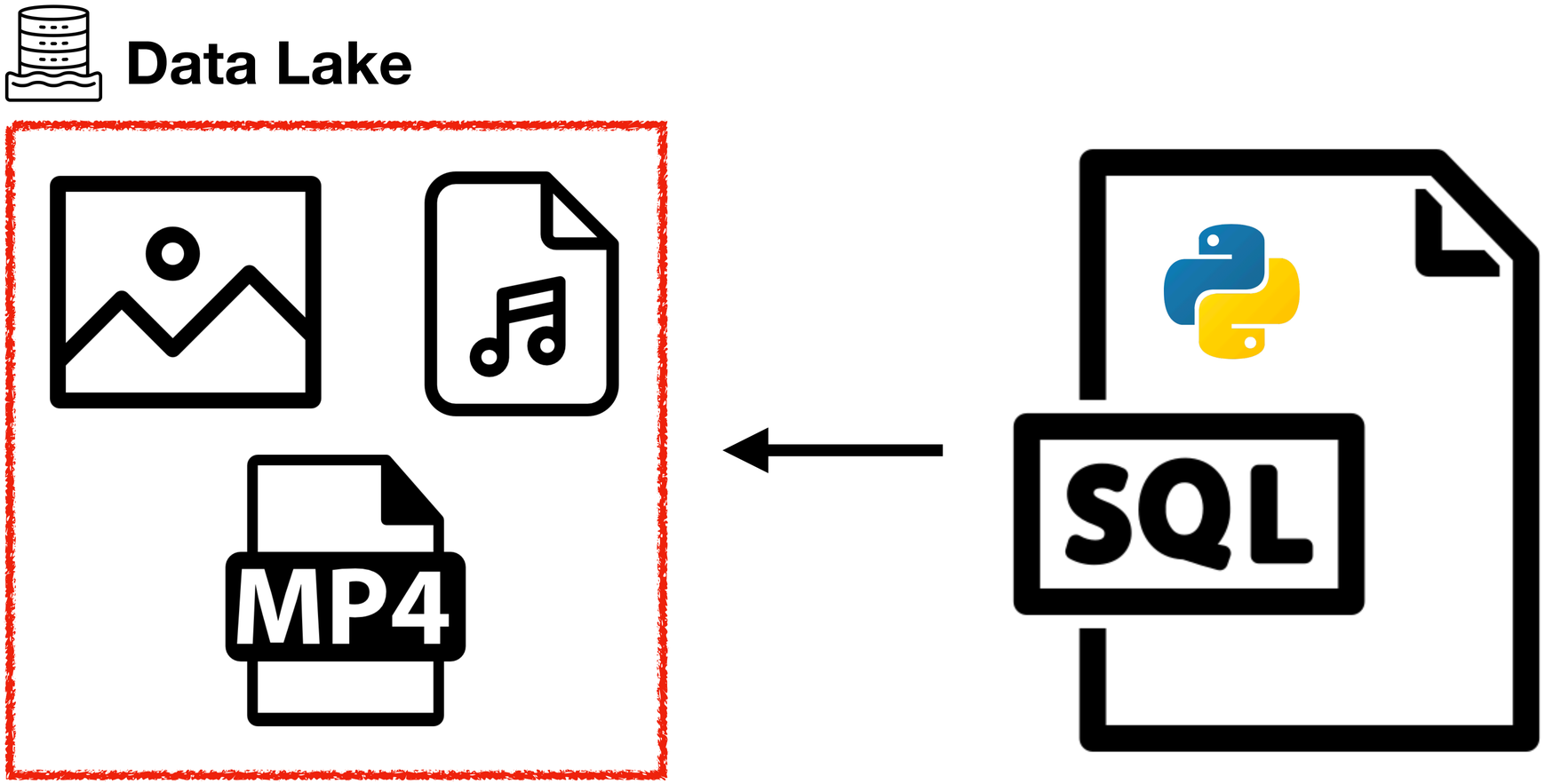
What is a polymorphic table function

What we want



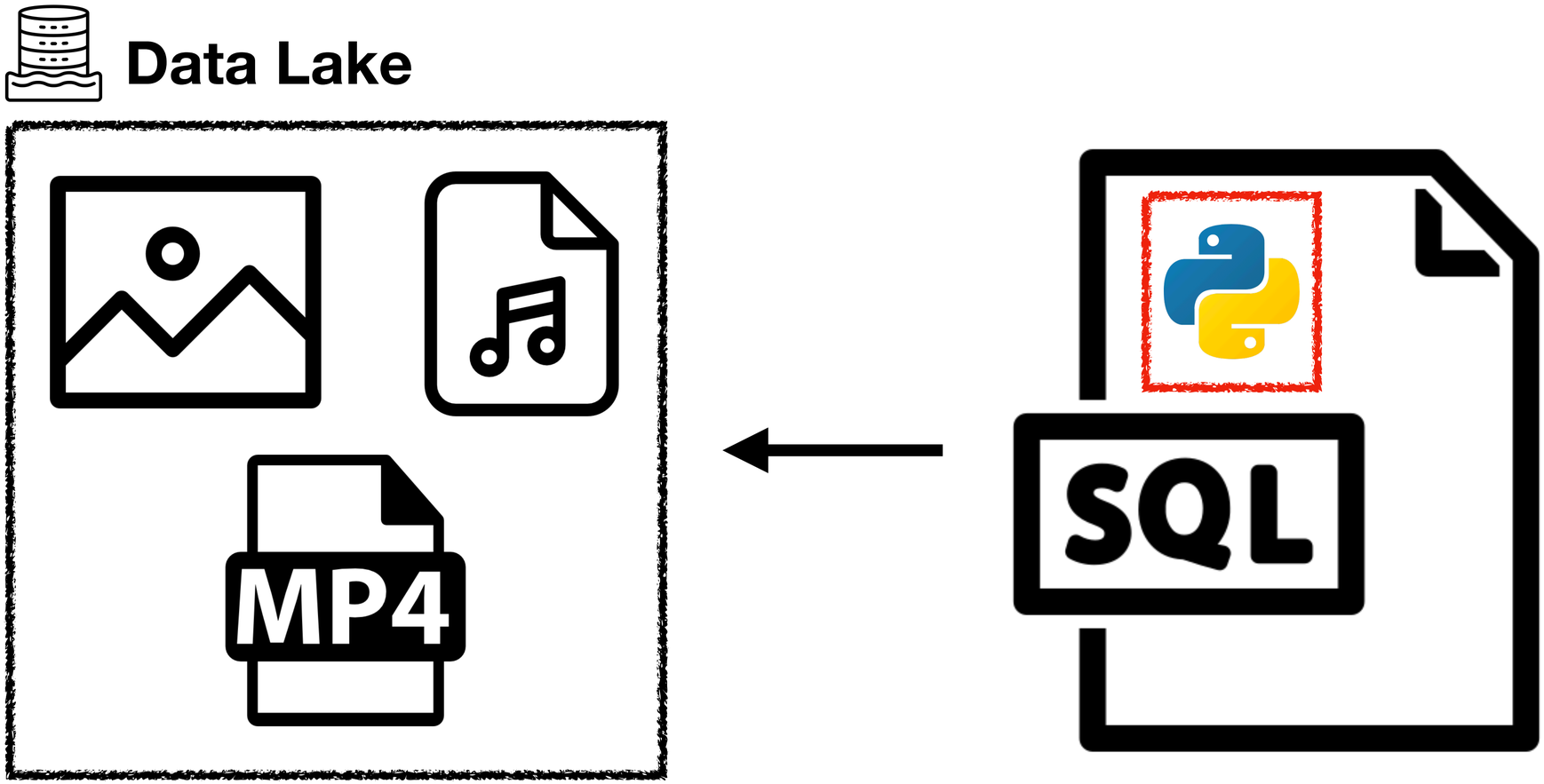
What is a polymorphic table function

What we want



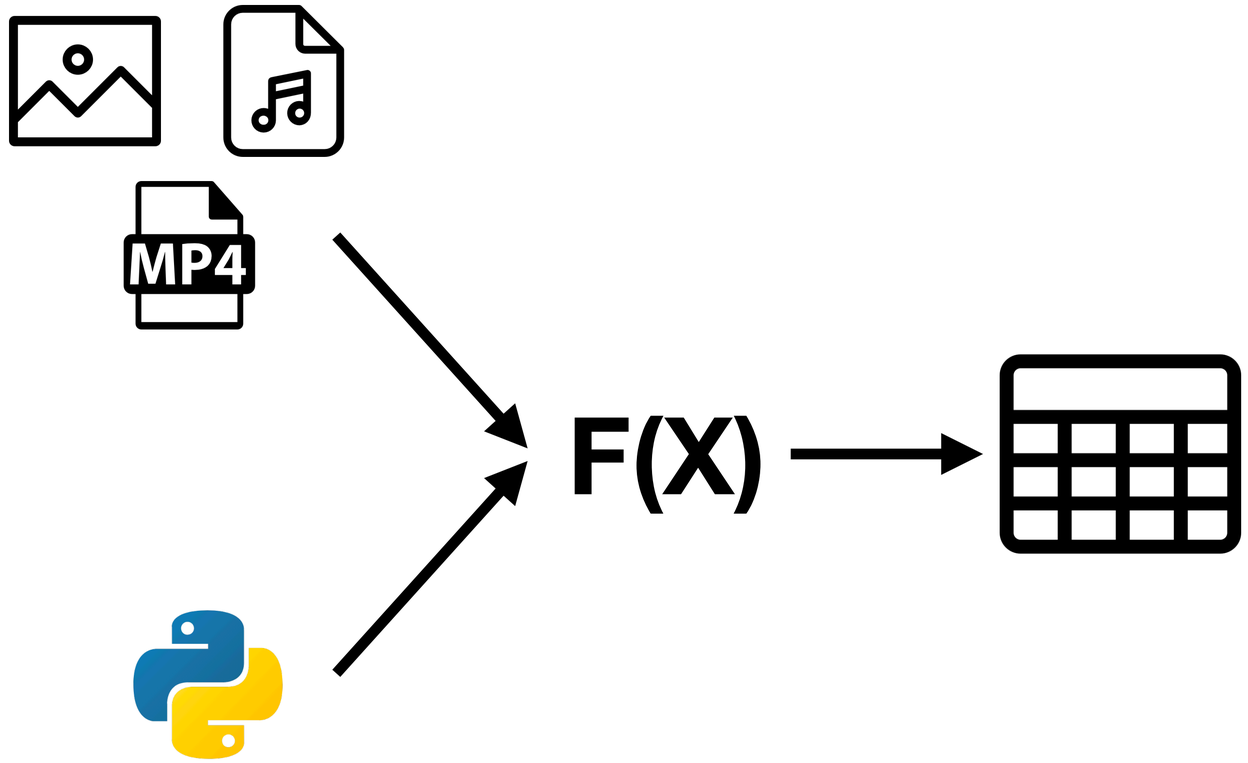
What is a polymorphic table function

What we want



What is a polymorphic table function

What we want

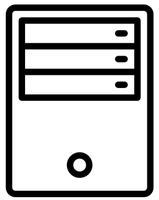


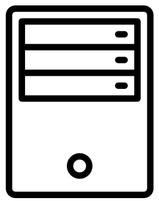
What is a polymorphic table function

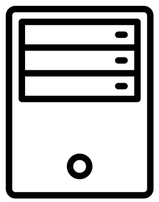
Usage Scenario

What is a polymorphic table function

Usage Scenario

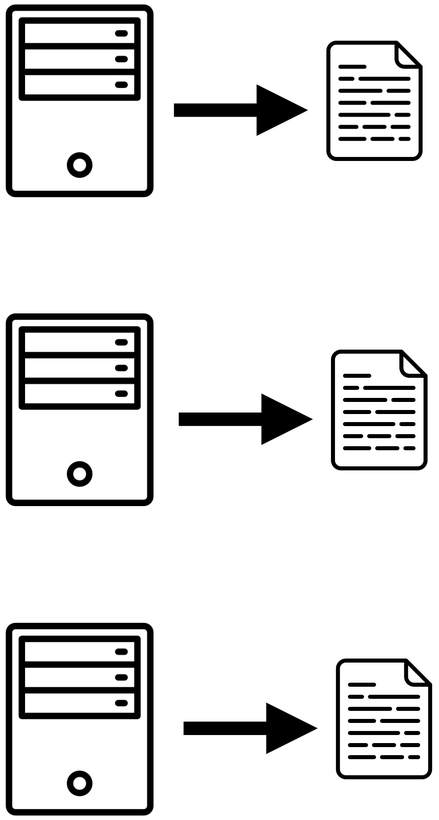






What is a polymorphic table function

Usage Scenario

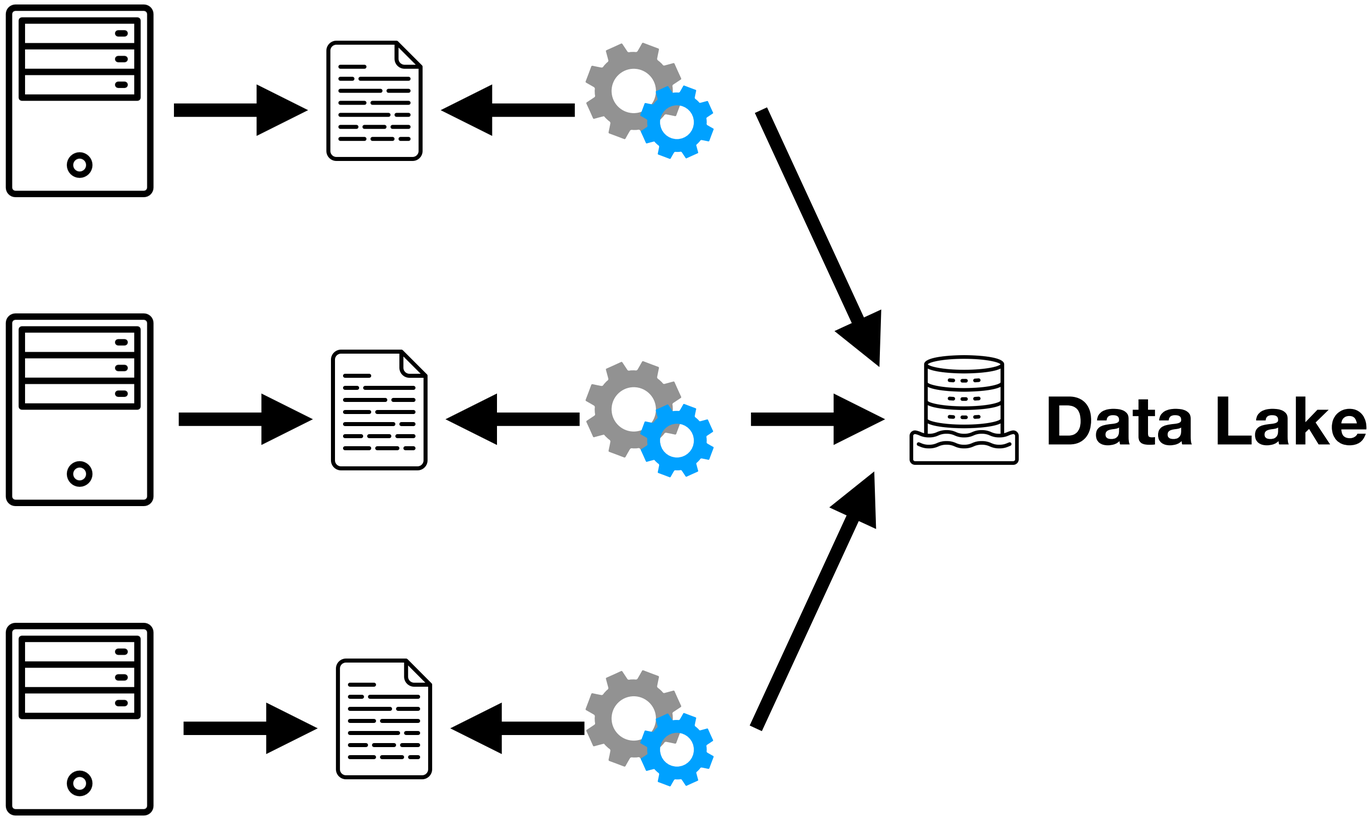


SK telecom

trino

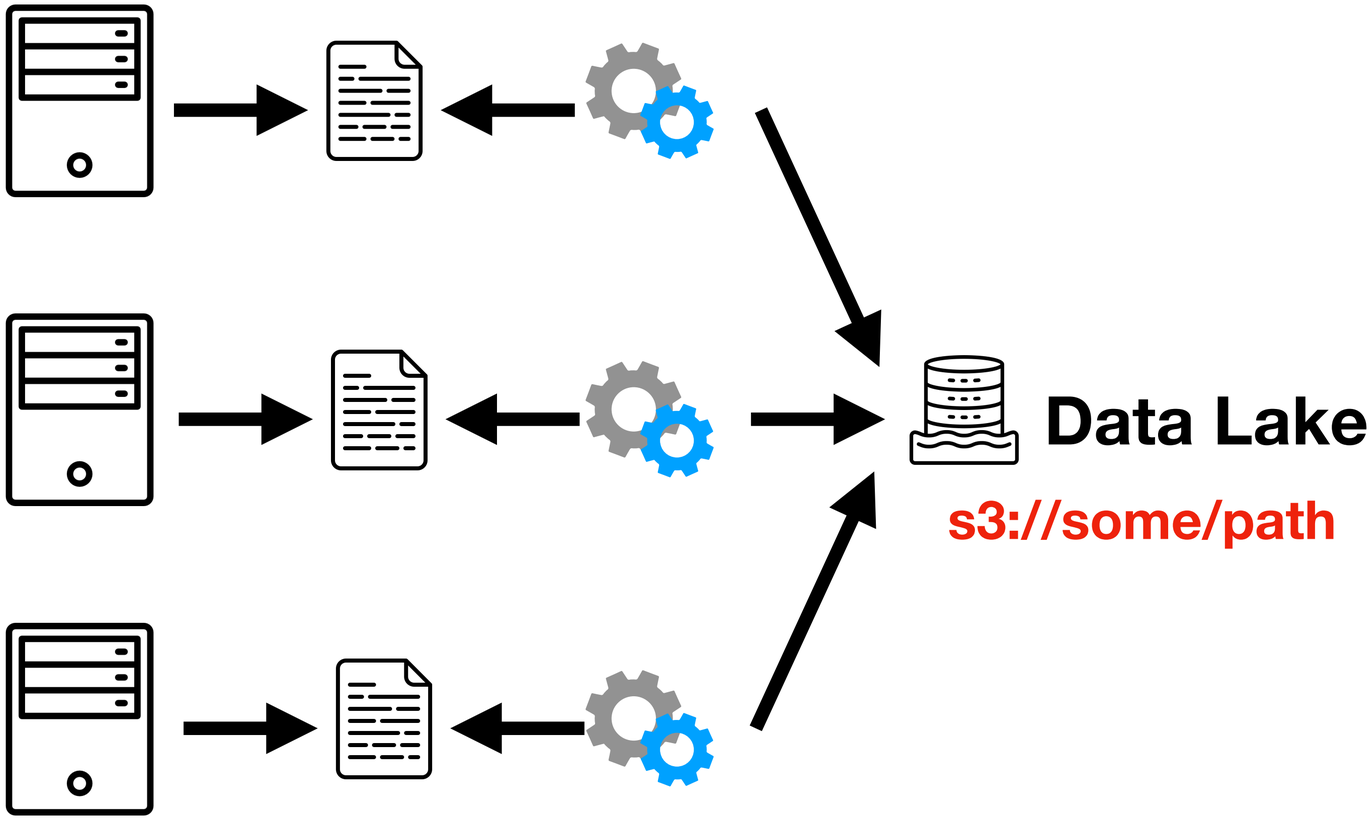
What is a polymorphic table function

Usage Scenario



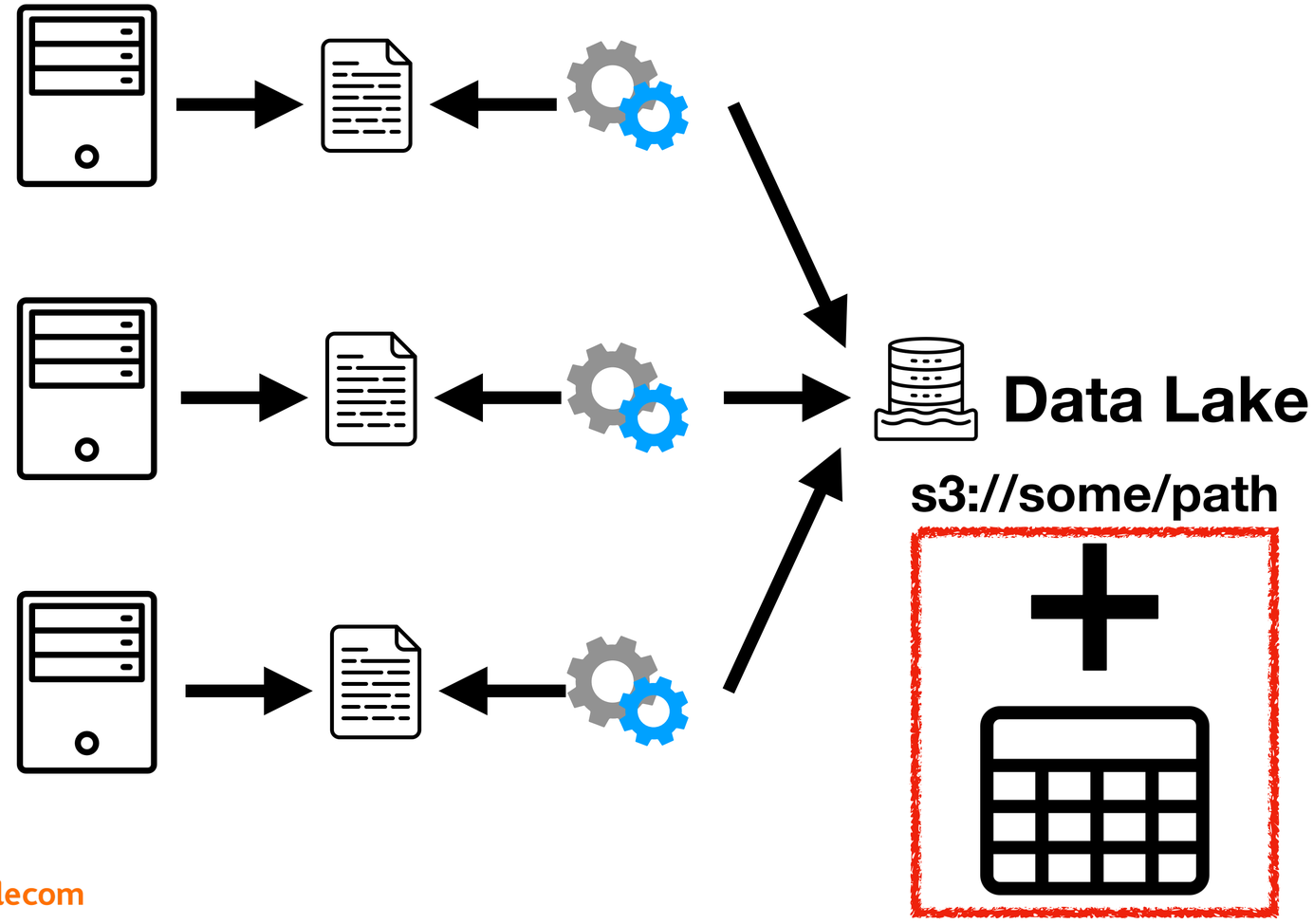
What is a polymorphic table function

Usage Scenario



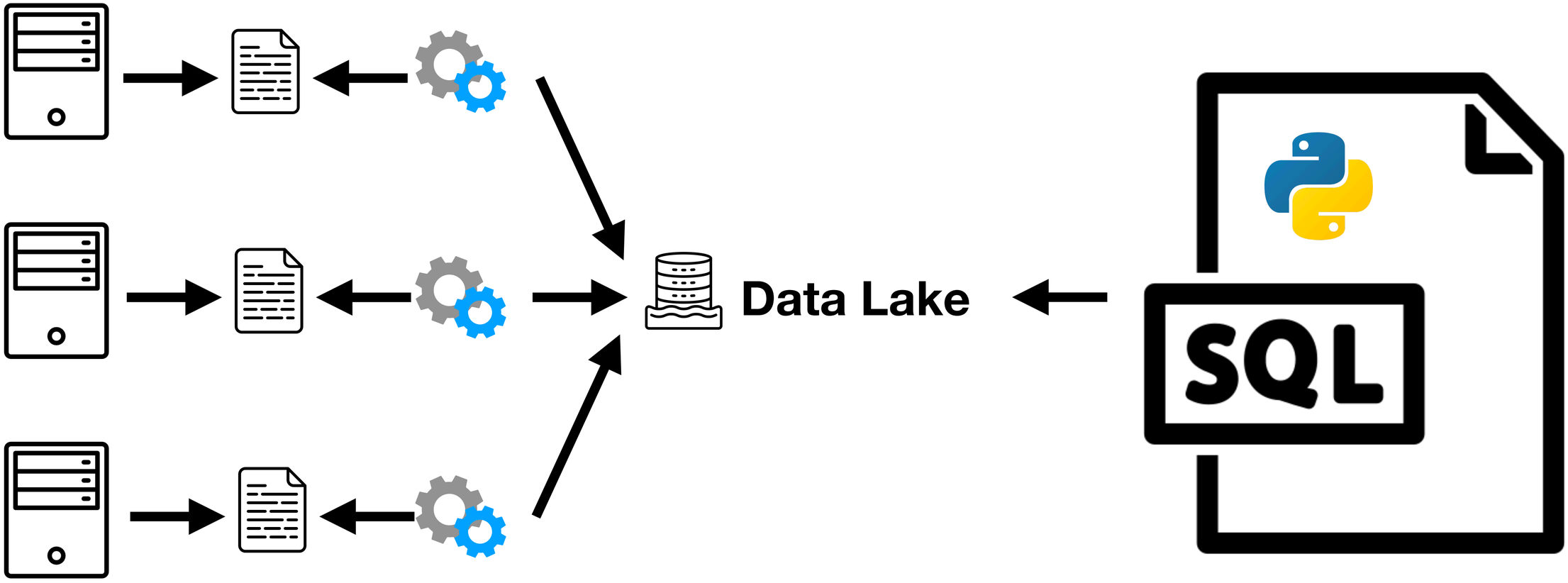
What is a polymorphic table function

Usage Scenario



What is a polymorphic table function

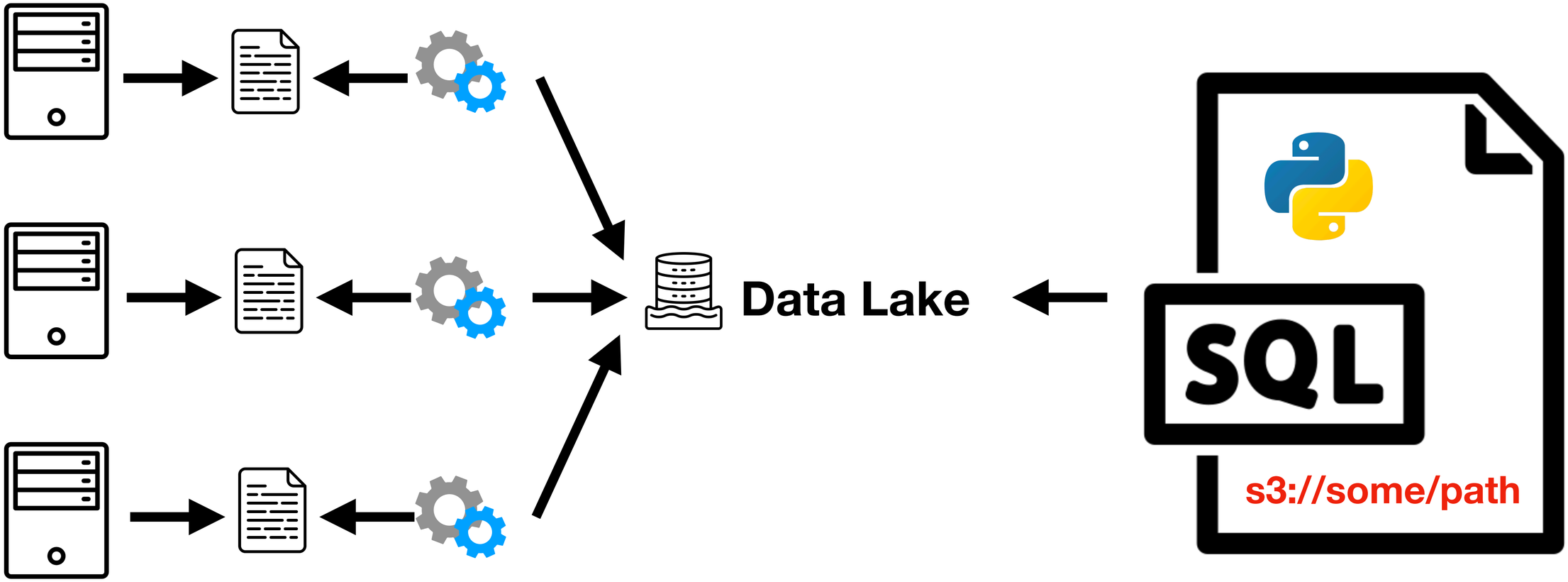
Usage Scenario



Python File Query

Python File Query

When to use it?



Python File Query

How to use?

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

SK telecom

trino

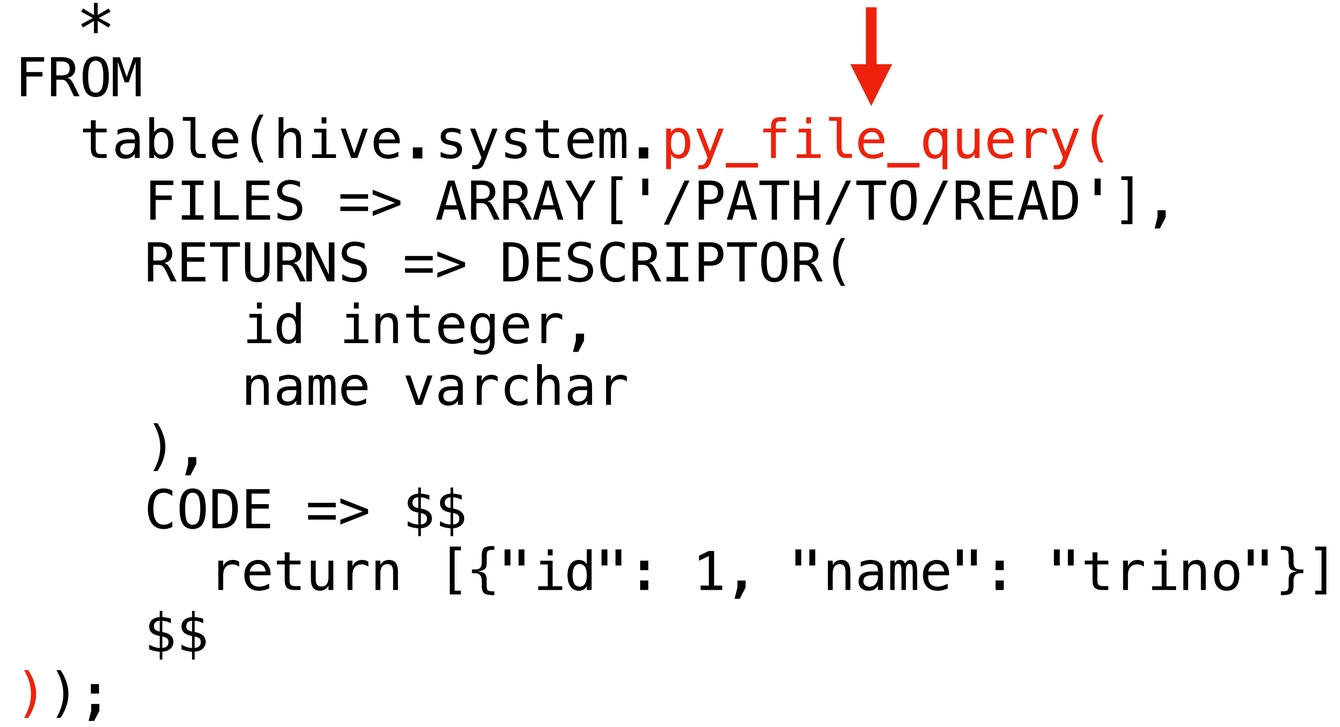
Python File Query

How to use?

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
      id integer,
      name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

Python File Query

How to use?



```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

Python File Query

How to use?

File paths to preprocess

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

Python File Query

How to use?

Columns returned from code

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
      id integer,
      name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

Python File Query

How to use?

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

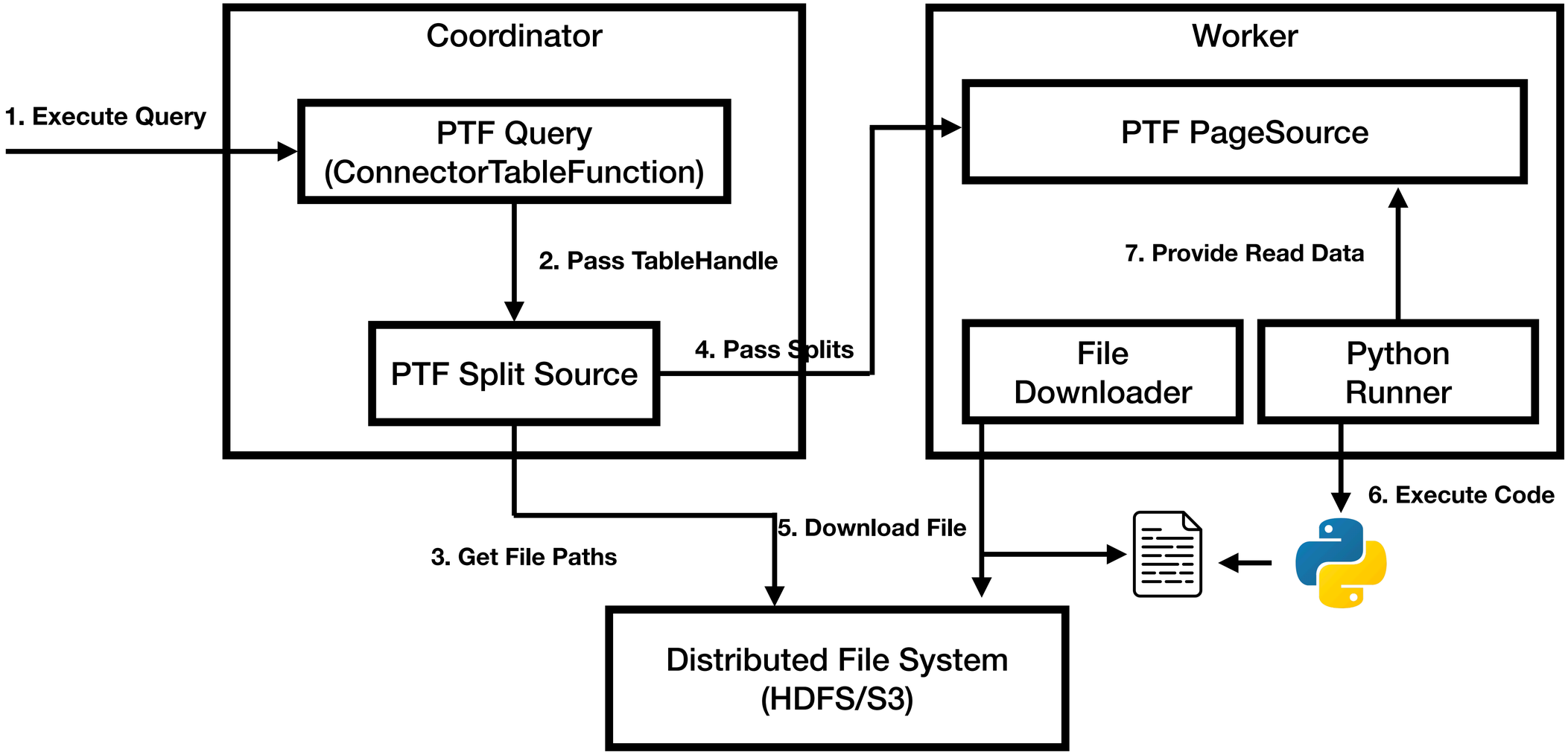
Pre-processing code

Python File Query

DEMO

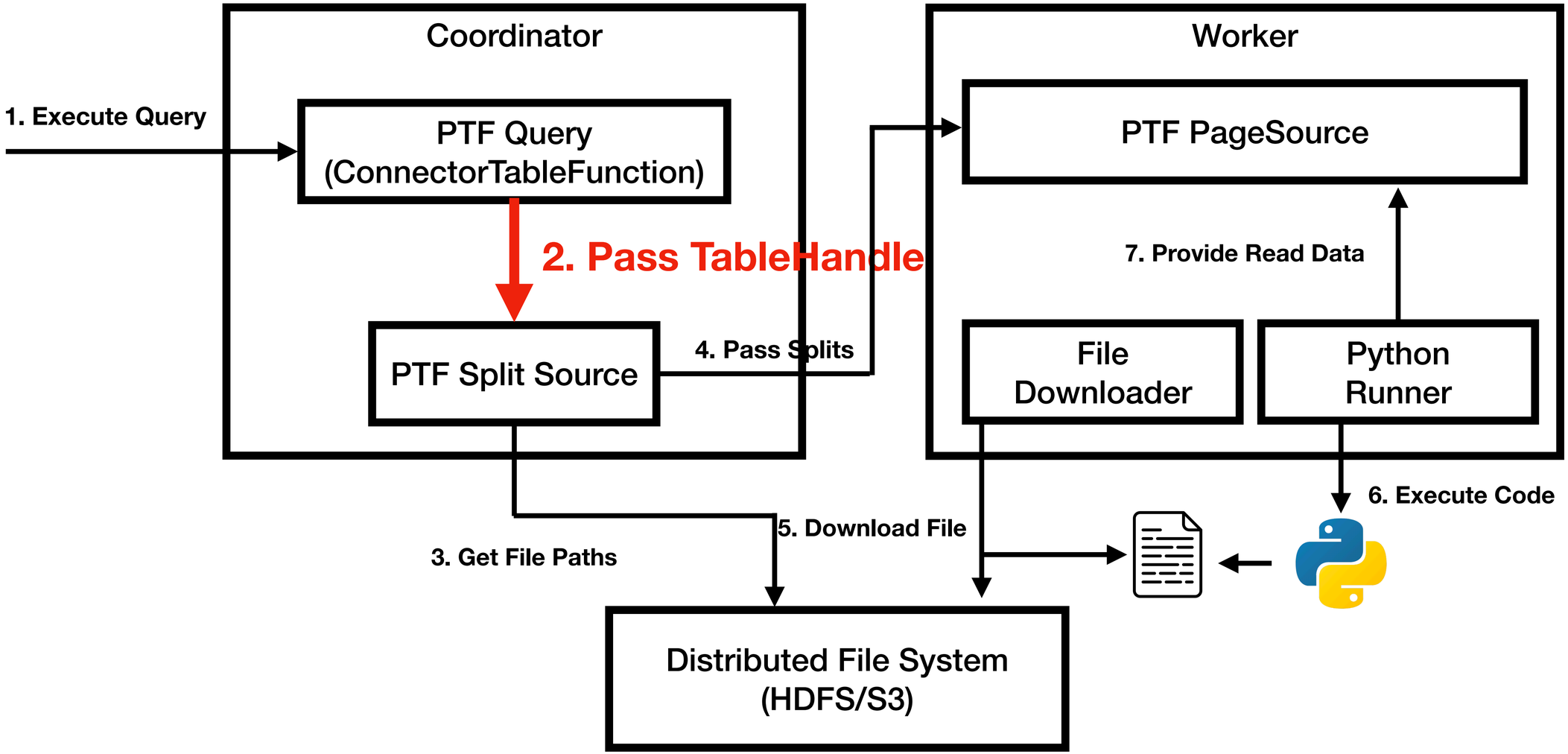
Python File Query

Implementation - Overview



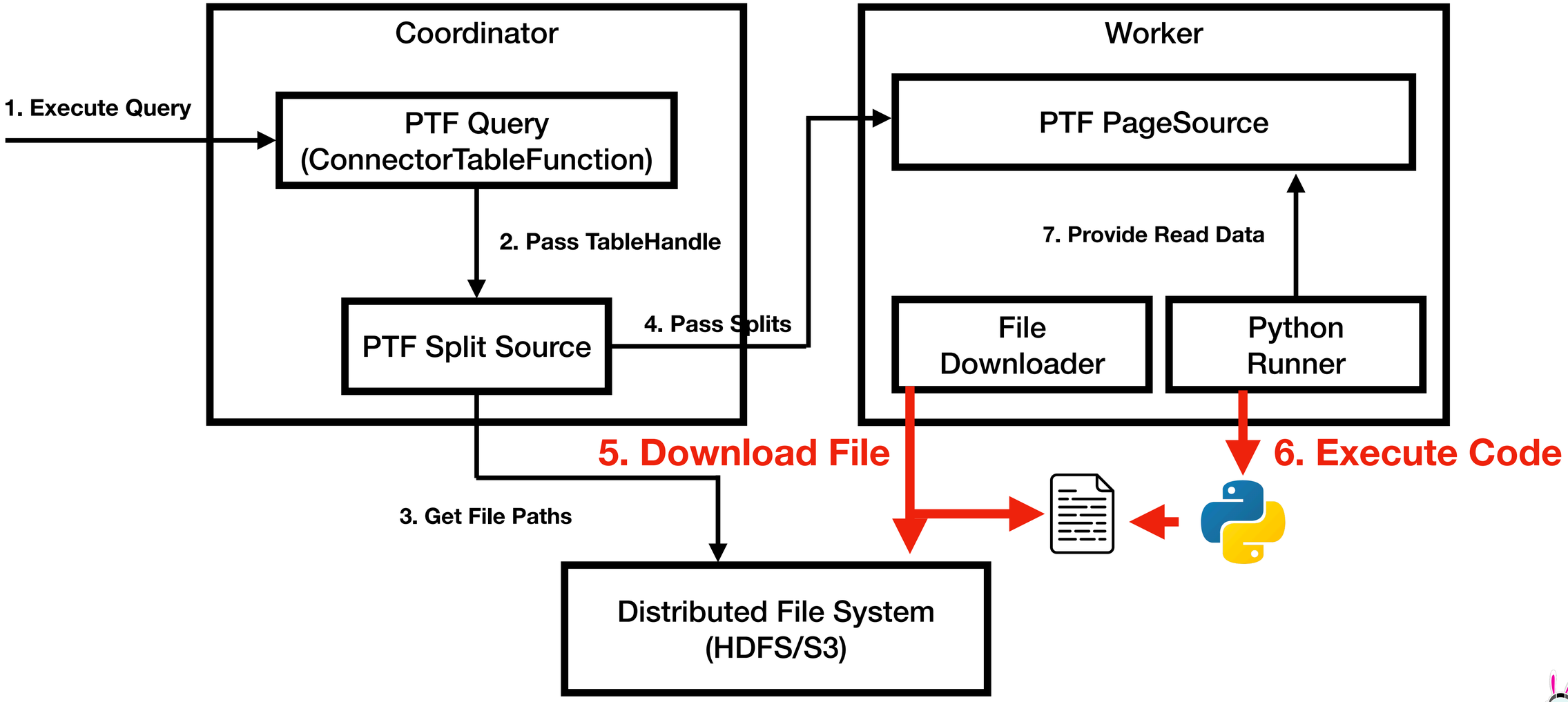
Python File Query

Implementation - Overview



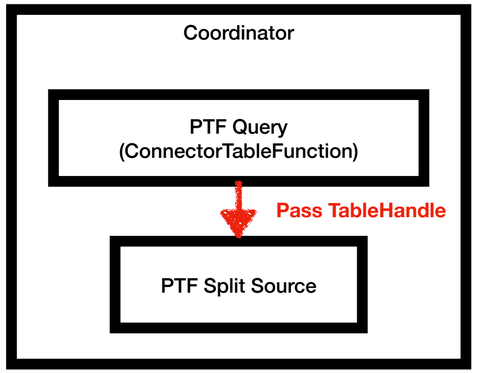
Python File Query

Implementation - Overview



Python File Query

Implementation - Generate TableHandle




```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

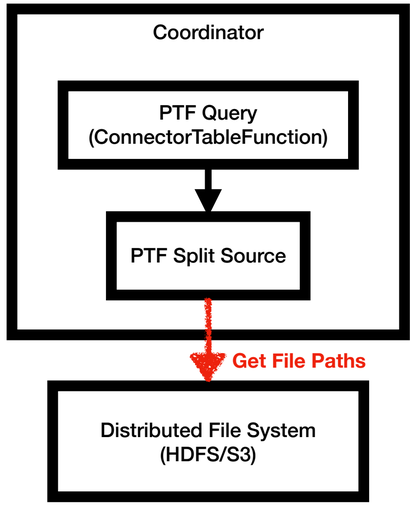
```
public class HiveFileTableHandle
        implements ConnectorTableHandle
{
    private final List<String> files;
    private final List<HiveColumnHandle> columnHandles;
    private final String code;

    @JsonCreator
    public HiveFileTableHandle(
            @JsonProperty("files") List<String> files,
            @JsonProperty("columns") List<HiveColumnHandle> columnHandles,
            @JsonProperty("code") String code)
    {
        this.files = files;
        this.columnHandles = columnHandles;
        this.code = code;
    }
}
```

SK telecom

trino

Python File Query

Implementation - Generate Splits



- /some/path1
  - file1
  - file2
- /some/path2/file3


```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/SOME/PATH1', '/SOME/PATH2/FILE3'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      return [{"id": 1, "name": "trino"}]
    $$
));
```

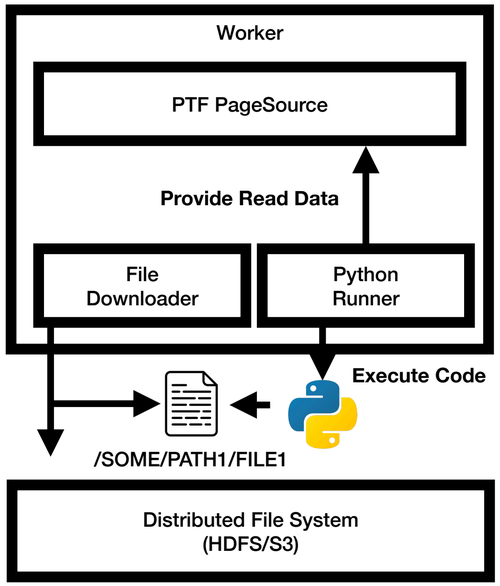
```
public class HiveFileSplit
        implements ConnectorSplit
{
    private final String path;
    private final List<HostAddress> addresses;

    @JsonCreator
    public HiveFileSplit(
            @JsonProperty("path") String path,
            @JsonProperty("addresses") List<HostAddress> addresses)
    {
        this.path = path;
        this.addresses = addresses;
    }
}
```

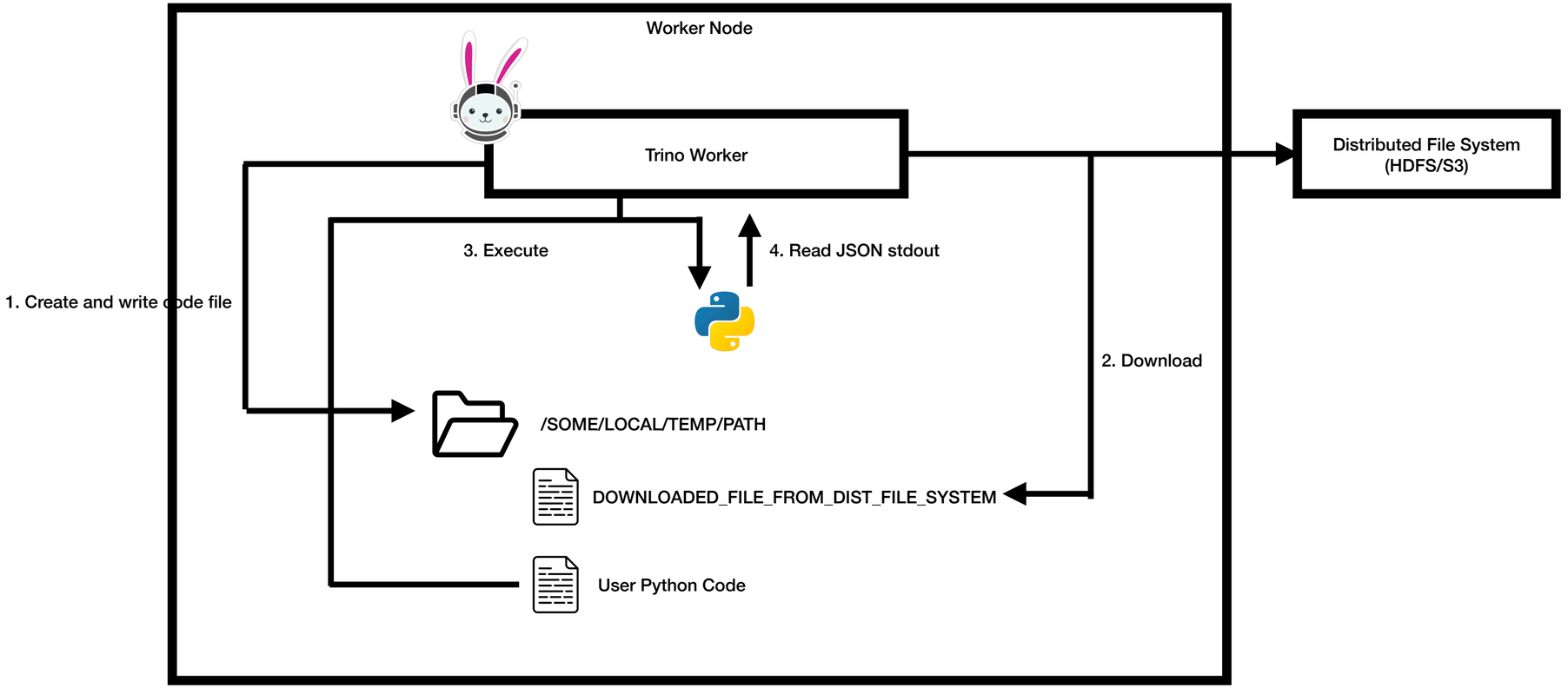
SK telecom

trino

Python File Query

Implementation - Read data



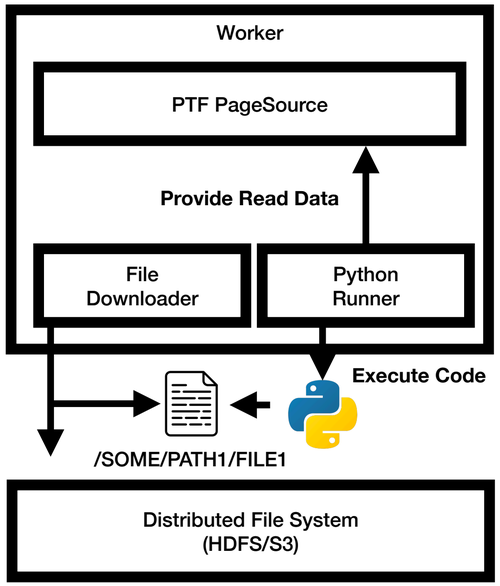


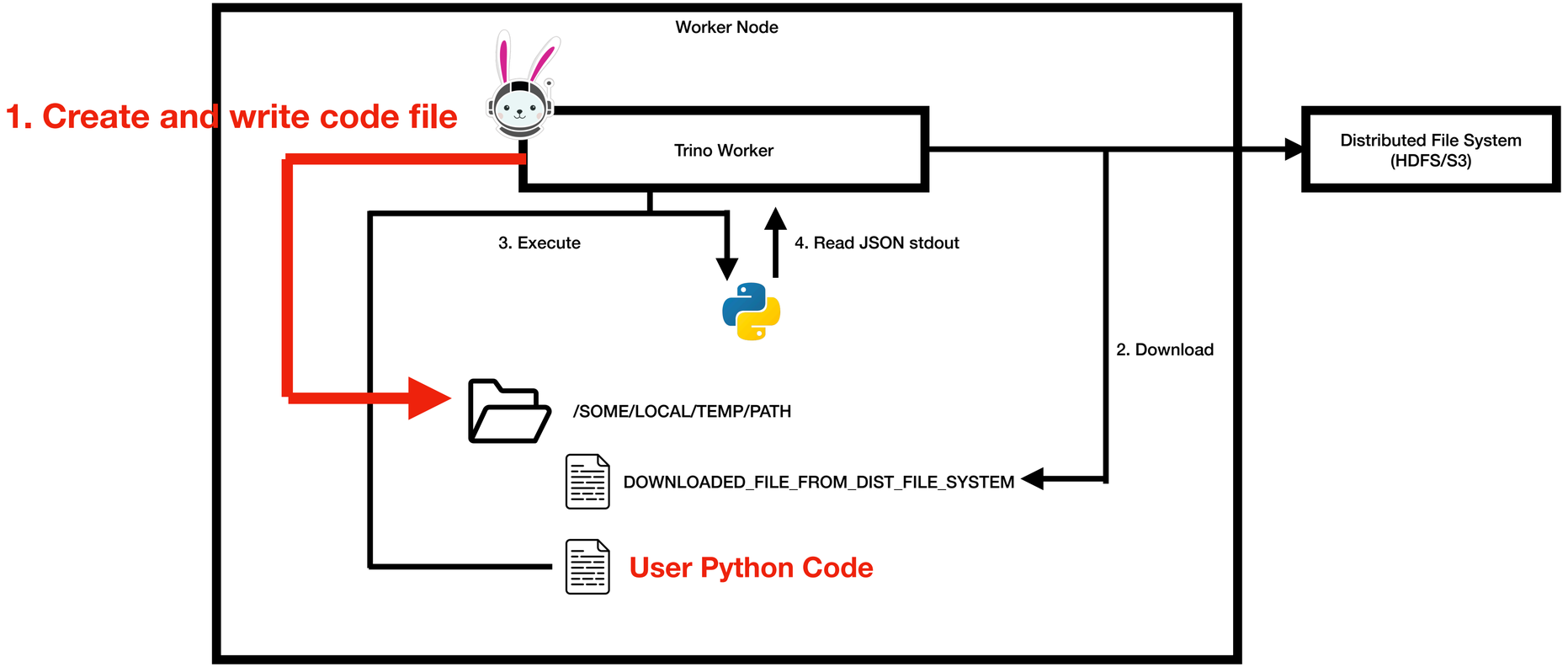
SK telecom

trino

Python File Query

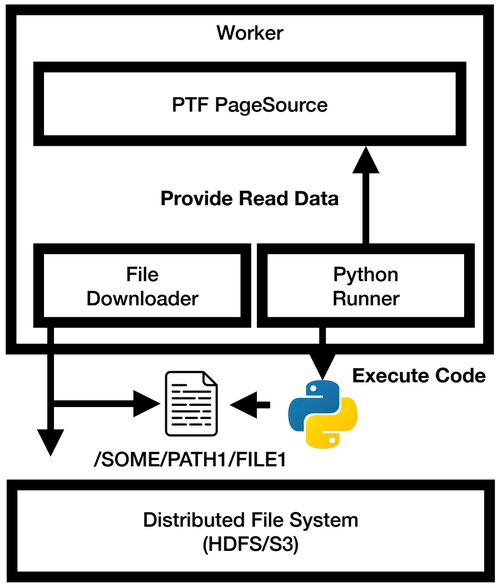
Implementation - Read data

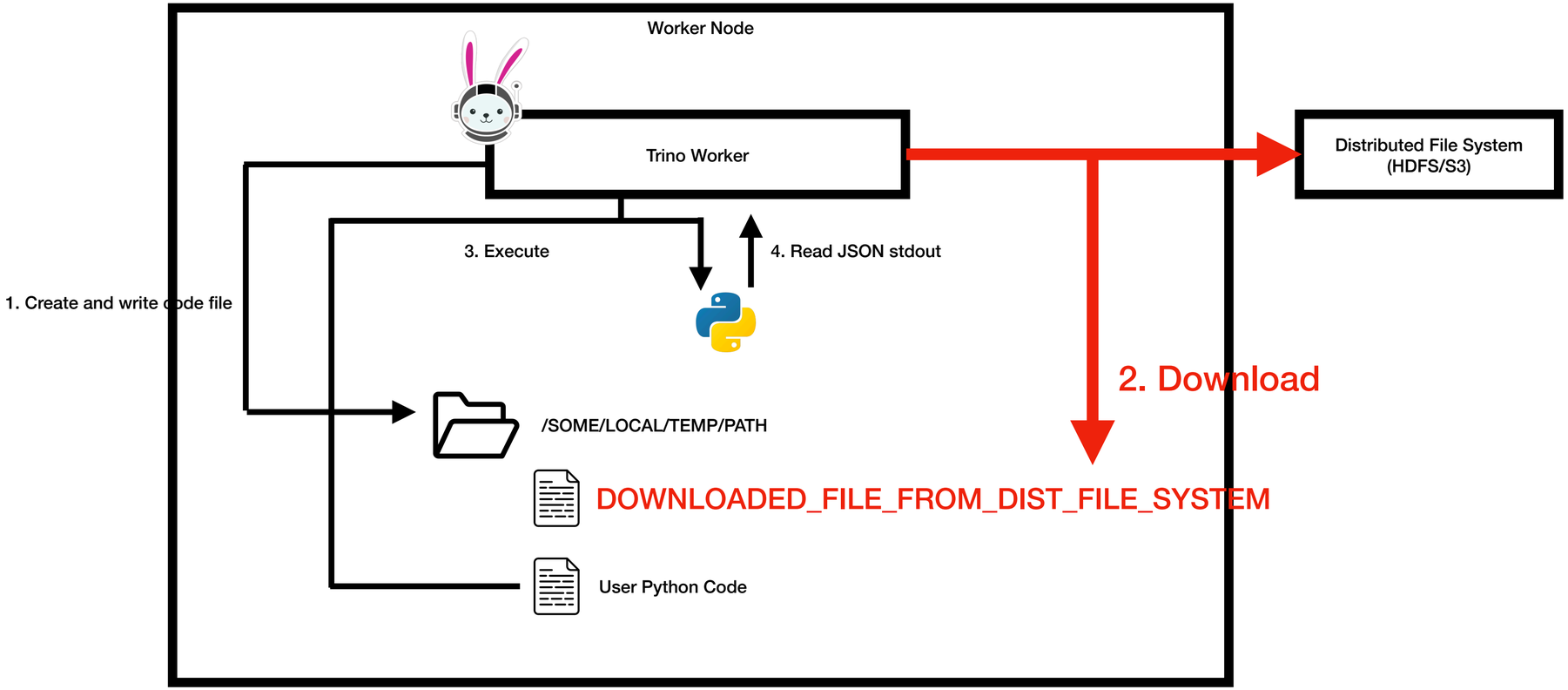




Python File Query

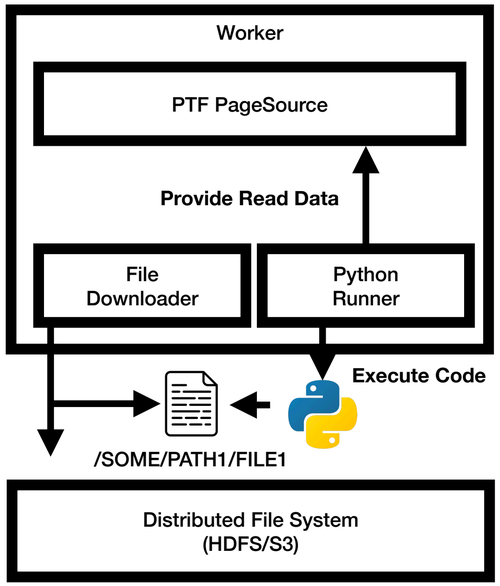
Implementation - Read data

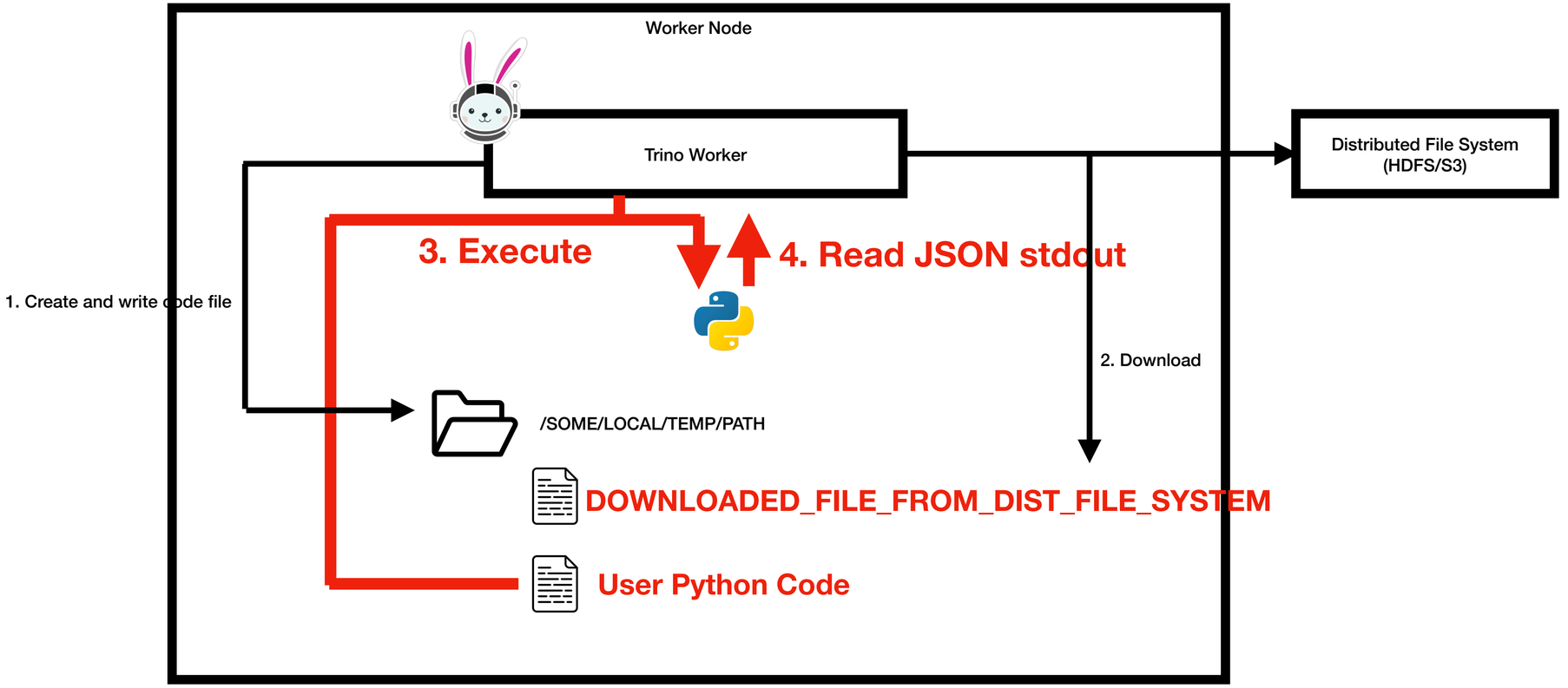




Python File Query

Implementation - Read data

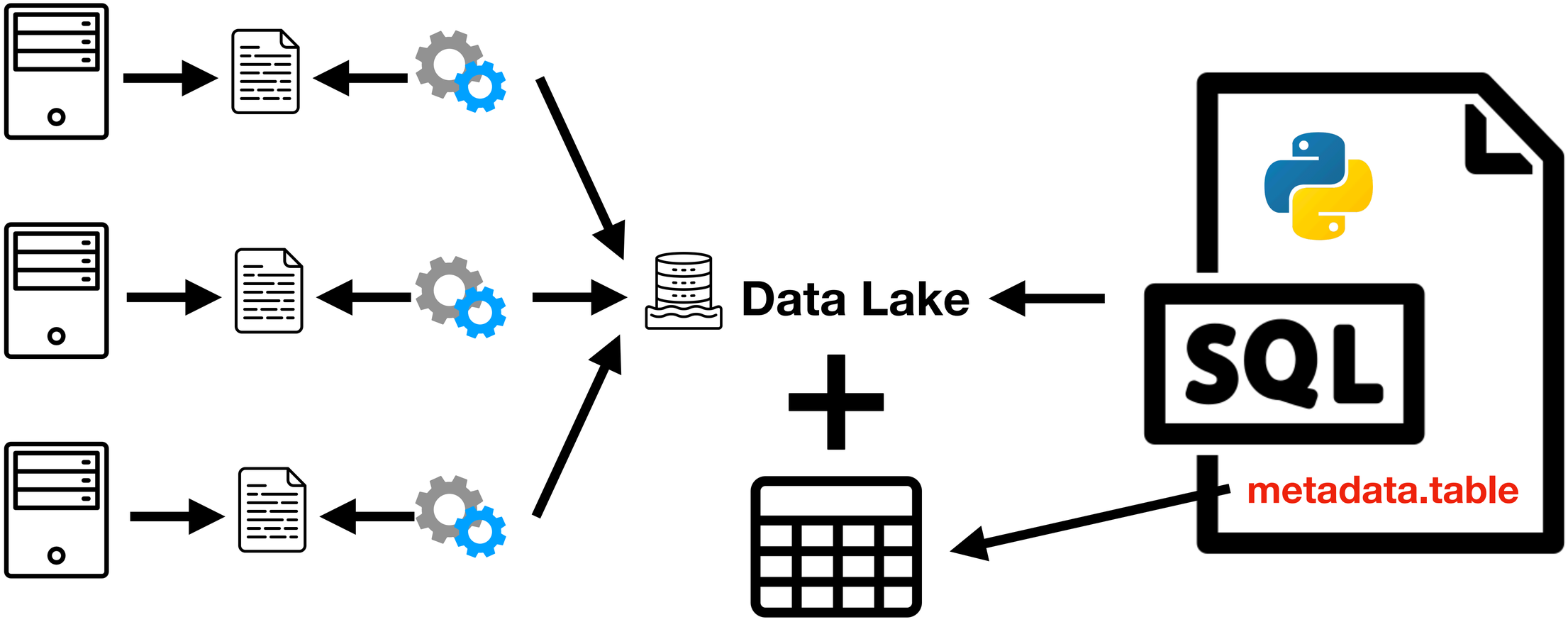




Python Meta Query

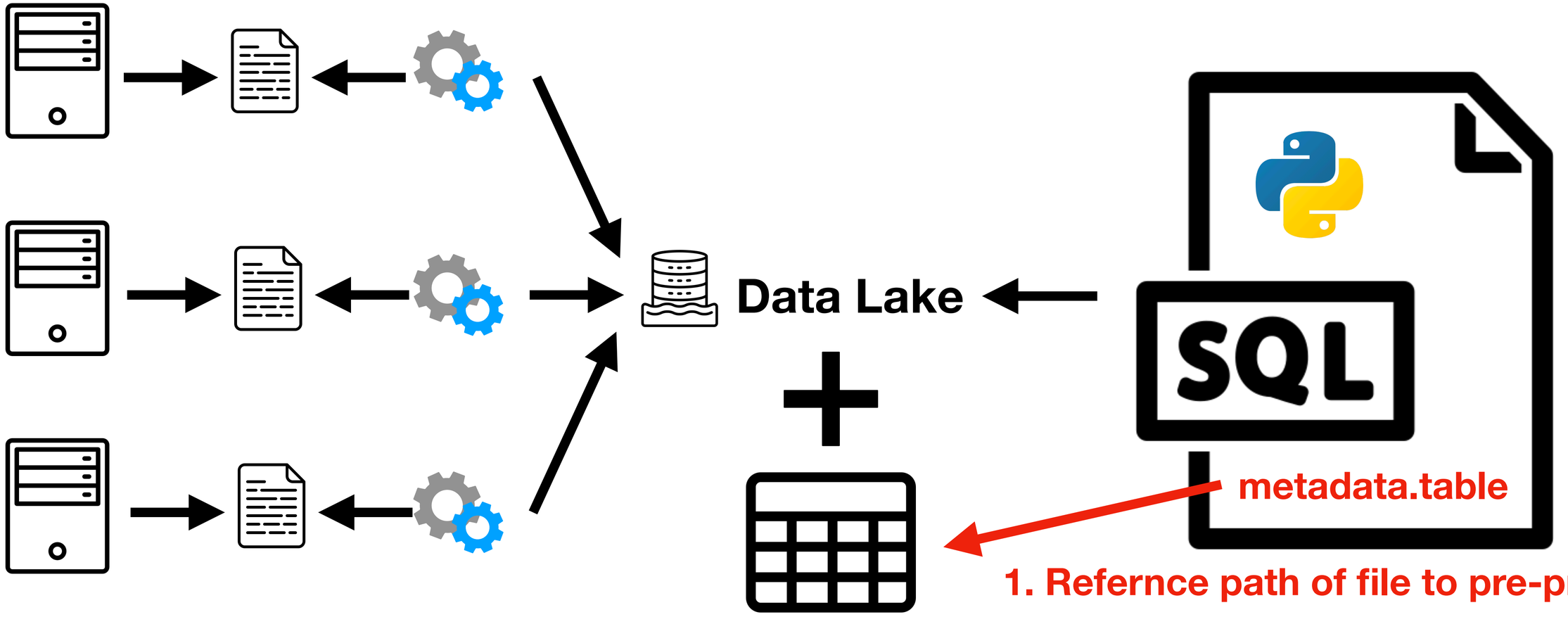
Python Meta Query

When to use it?



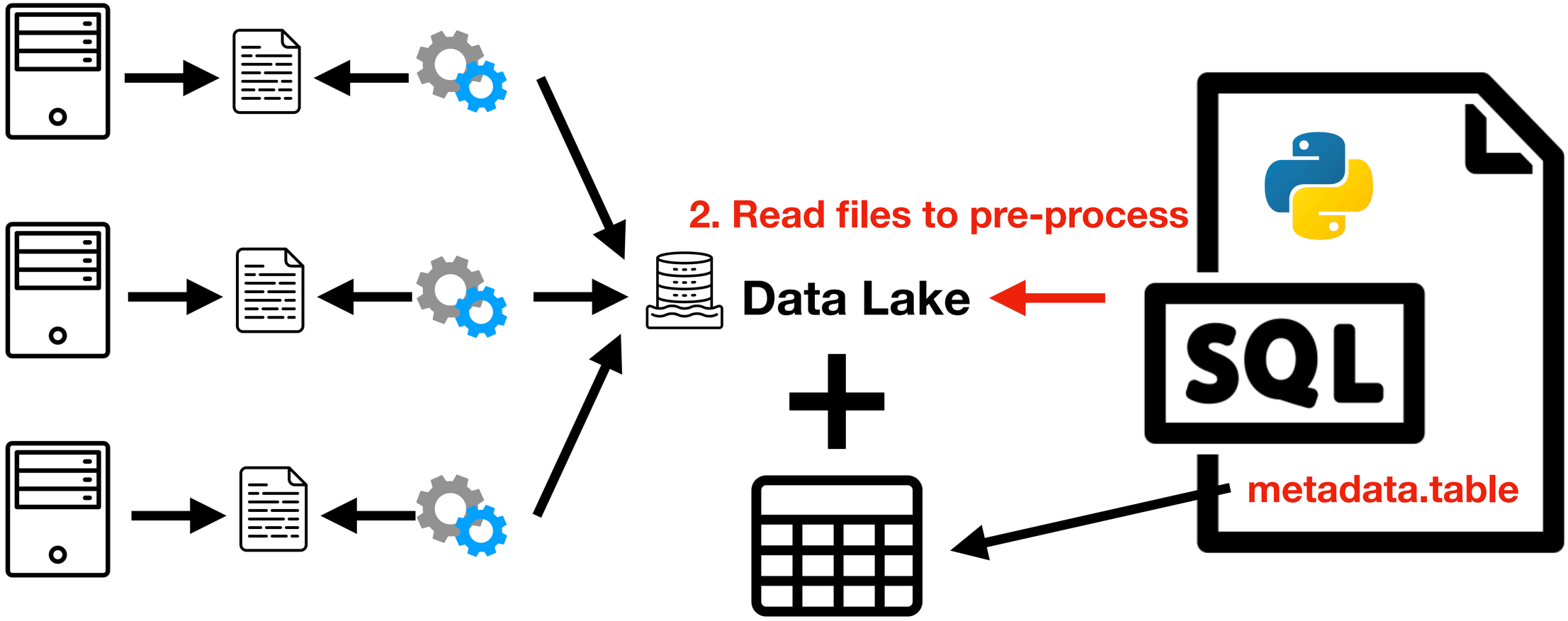
Python Meta Query

When to use it?



Python Meta Query

When to use it?



Python Meta Query

How to use?

```
SELECT
  *
FROM
  table(hive.system.py_meta_query(
    DB => 'unstructured',
    TBL => 'table',
    PATH_FIELD => 'path',
    RETURNS => descriptor(
      dat varchar
    ),
    CODE => $$
      return [{"dat": "trino"}]
    $$
));
```

SK telecom

trino

Python Meta Query

How to use?

Schema that have metadata

Table that have metadata

```
SELECT
  *
FROM
  table(hive.system.py_meta_query(
    DB => 'unstructured',
    TBL => 'table',
    PATH_FIELD => 'path',
    RETURNS => descriptor(
      dat varchar
    ),
    CODE => $$
      return [{"dat": "trino"}]
    $$
));
```

Python Meta Query

How to use?

Field name in metadata ref path

```
SELECT
  *
FROM
  table(hive.system.py_meta_query(
    DB => 'unstructured',
    TBL => 'table',
    PATH_FIELD => 'path',
    RETURNS => descriptor(
      dat varchar
    ),
    CODE => $$
      return [{"dat": "trino"}]
    $$
));
```

Python Meta Query

How to use?

Columns returned from code

Python code to process file

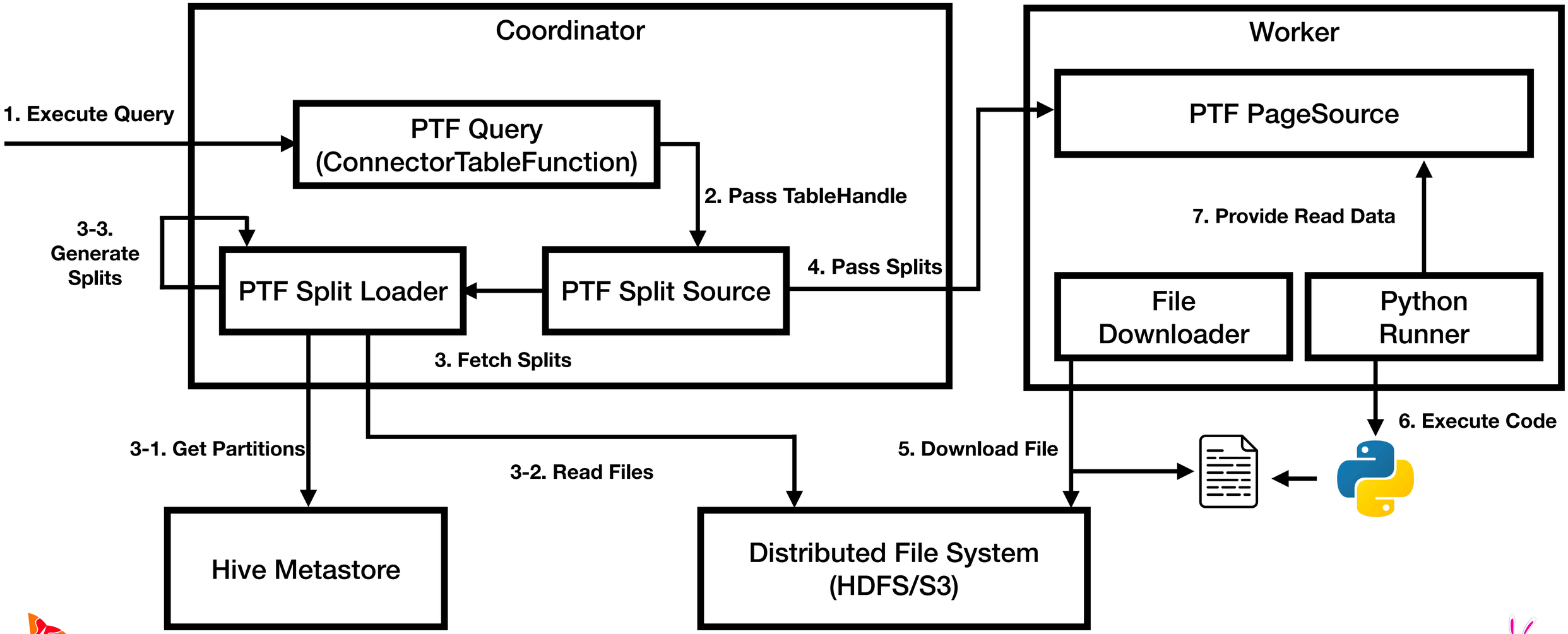
```
SELECT
  *
FROM
  table(hive.system.py_meta_query(
    DB => 'unstructured',
    TBL => 'table',
    PATH_FIELD => 'path',
    RETURNS => descriptor(
      dat varchar
    ),
    CODE => $$
      return [{"dat": "trino"}]
    $$
));
```

Python Meta Query

DEMO

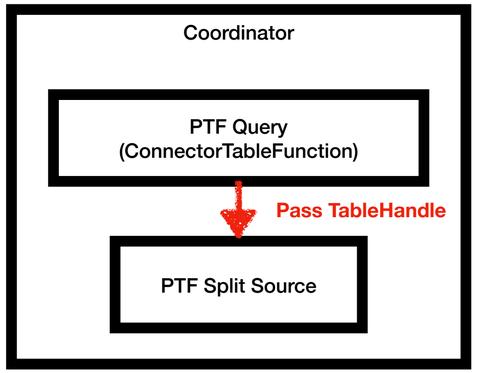
Python Meta Query

Implementation - Overview



Python Meta Query

Implementation - Generate TableHandle

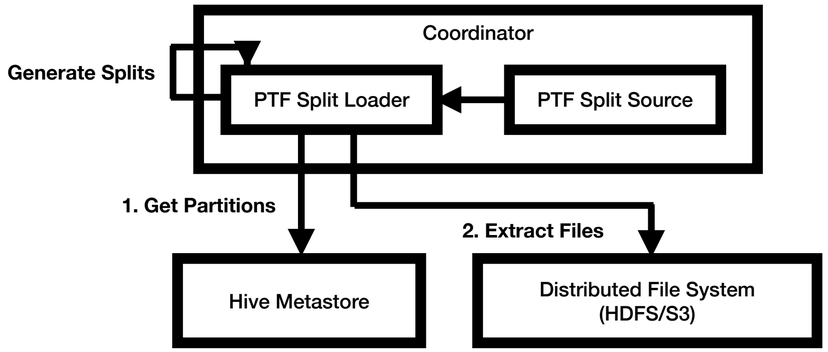



```
SELECT
  *
FROM
  table(hive.system.py_meta_query(
    DB => 'unstructured',
    TBL => 'table',
    PATH_FIELD => 'path',
    RETURNS => descriptor(
      path varchar,
      dat varchar
    ),
    CODE => $$
      return [{"dat": "trino"}]
    $$
));
```

```
public class MetaPTFHandle
        extends PTFHandle
{
    private final String code;
    private final String pathField;

    @JsonCreator
    public MetaPTFHandle(
            @JsonProperty("ptfType") Type ptfType,
            @JsonProperty("code") String code,
            @JsonProperty("pathField") String pathField)
    {
        super(ptfType);
        this.code = requireNonNull(code, "code is null");
        this.pathField = requireNonNull(pathField, "pathField is null");
    }
}
public class HiveTableHandle
        implements ConnectorTableHandle
{
    private final String schemaName;
    private final String tableName;

    ...
    private final Optional<PTFHandle> ptfHandle;
}
```

Python Meta Query

Implementation - Generate Splits



Partitions

dt = 20231203

dt = 20231204

Directories

└ /dt=20231203

└ file1

└ file2

└ /dt=20231204

└ file3

server	load_time	path
host1	2023-12-03 12:00:00	s3://some/path1
host1	2023-12-03 12:00:00	s3://some/path2

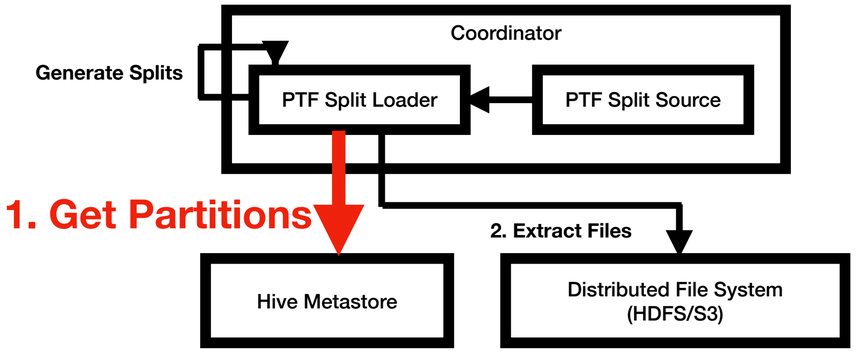
path = s3://some/path1
code = "some code"
addresses = […]
columnDataMapping =
{ “server”: “host1”, “load_time”:”2023-12-03 12:00:00”}

server	load_time	path
host2	2023-12-03 12:00:00	s3://some/path3
host3	2023-12-03 12:00:00	s3://some/path4

server	load_time	path
host1	2023-12-04 12:00:00	s3://some/path5
host1	2023-12-04 12:00:00	s3://some/path6
host2	2023-12-04 12:00:00	s3://some/path7
host3	2023-12-04 12:00:00	s3://some/path8

Python Meta Query

Implementation - Generate Splits



Partitions

Directories

dt = 20231203 → ㄴ /dt=20231203

ㄴ file1

ㄴ file2

dt = 20231204 → ㄴ /dt=20231204

ㄴ file3

server	load_time	path
host1	2023-12-03 12:00:00	s3://some/path1
host1	2023-12-03 12:00:00	s3://some/path2

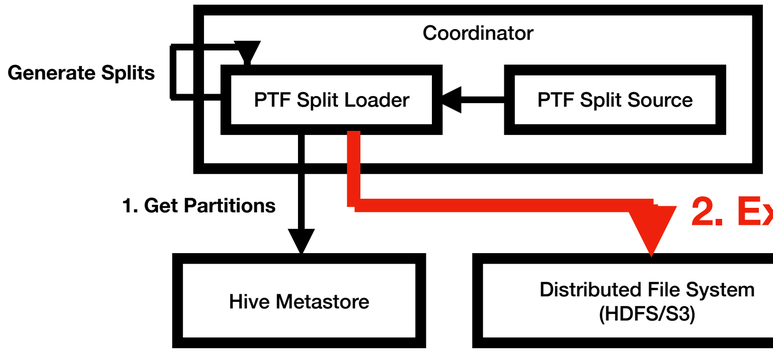
path = s3://some/path1
code = "some code"
addresses = […]
columnDataMapping =
{ “server”: “host1”, “load_time”:”2023-12-03 12:00:00”}

server	load_time	path
host2	2023-12-03 12:00:00	s3://some/path3
host3	2023-12-03 12:00:00	s3://some/path4

server	load_time	path
host1	2023-12-04 12:00:00	s3://some/path5
host1	2023-12-04 12:00:00	s3://some/path6
host2	2023-12-04 12:00:00	s3://some/path7
host3	2023-12-04 12:00:00	s3://some/path8

Python Meta Query

Implementation - Generate Splits



Partitions

dt = 20231203

dt = 20231204

Directories

└ /dt=20231203

└ file1

└ file2

└ /dt=20231204

└ file3

server	load_time	path
host1	2023-12-03 12:00:00	s3://some/path1
host1	2023-12-03 12:00:00	s3://some/path2

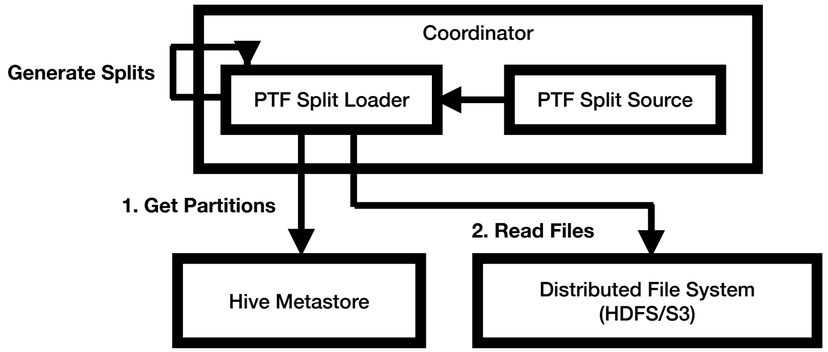
path = s3://some/path1
code = "some code"
addresses = […]
columnDataMapping =
{ “server”: “host1”, “load_time”:”2023-12-03 12:00:00”}

server	load_time	path
host2	2023-12-03 12:00:00	s3://some/path3
host3	2023-12-03 12:00:00	s3://some/path4

server	load_time	path
host1	2023-12-04 12:00:00	s3://some/path5
host1	2023-12-04 12:00:00	s3://some/path6
host2	2023-12-04 12:00:00	s3://some/path7
host3	2023-12-04 12:00:00	s3://some/path8

Python Meta Query

Implementation - Generate Splits



Partitions

dt = 20231203

dt = 20231204

Directories

└ /dt=20231203

└ file1

└ file2

└ /dt=20231204

└ file3

server	load_time	path
host1	2023-12-03 12:00:00	s3://some/path1
host1	2023-12-03 12:00:00	s3://some/path2

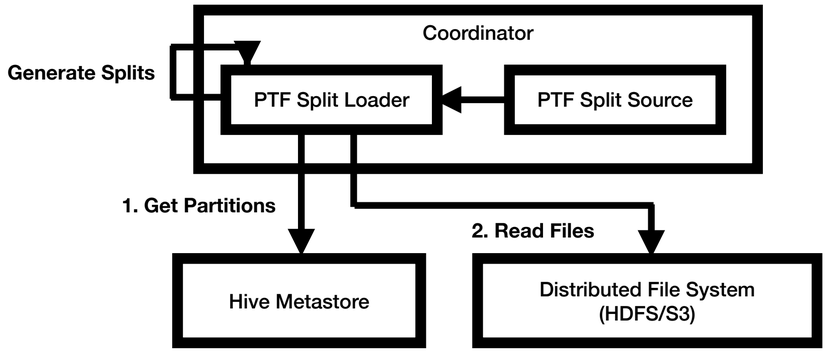
path = s3://some/path1
code = "some code"
addresses = […]
columnDataMapping =
{ “server”: “host1”, “load_time”:”2023-12-03 12:00:00”}

server	load_time	path
host2	2023-12-03 12:00:00	s3://some/path3
host3	2023-12-03 12:00:00	s3://some/path4

server	load_time	path
host1	2023-12-04 12:00:00	s3://some/path5
host1	2023-12-04 12:00:00	s3://some/path6
host2	2023-12-04 12:00:00	s3://some/path7
host3	2023-12-04 12:00:00	s3://some/path8

Python Meta Query

Implementation - Generate Splits



Partitions

dt = 20231203

dt = 20231204

Directories

└ /dt=20231203

└ file1

└ file2

└ /dt=20231204

└ file3

server	load_time	path
host1	2023-12-03 12:00:00	s3://some/path1
host1	2023-12-03 12:00:00	s3://some/path2

RecordReader

File Status API(HDFS)

```
path = s3://some/path1
code = "some code"
addresses = [...]
columnDataMapping =
{ “server”: “host1”, “load_time”:”2023-12-03 12:00:00”}
```

HivePartition Iterator

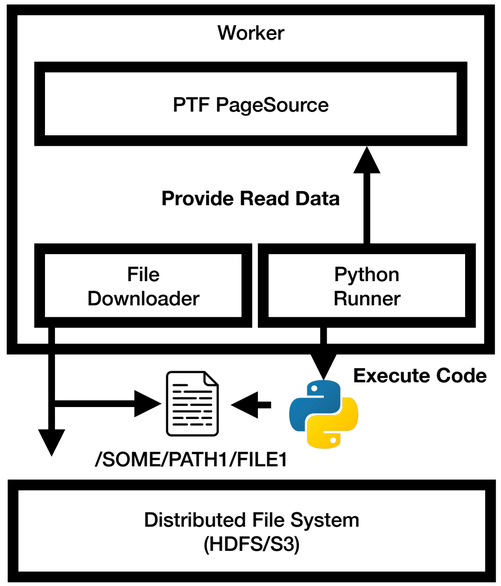
DirectoryLister

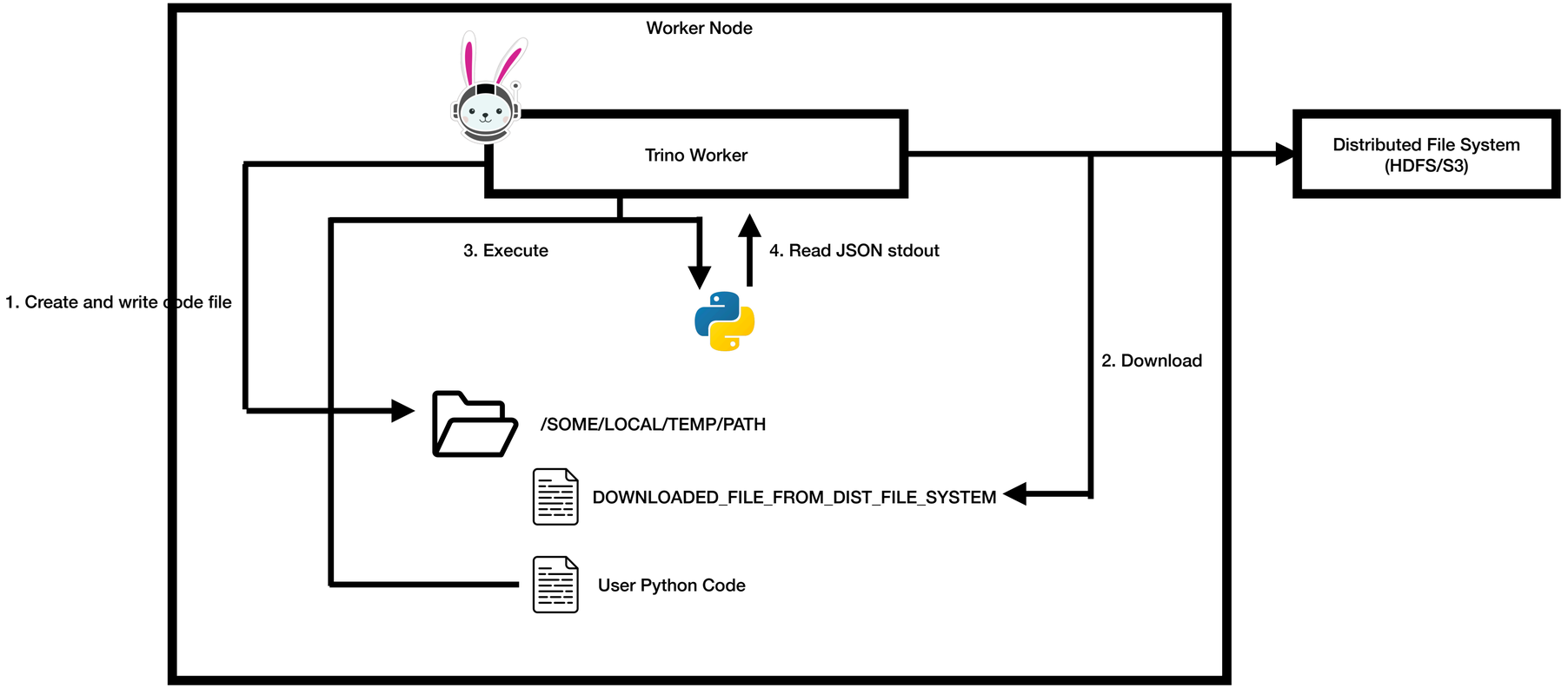
Slow API <- We need read in parallel!!

Reference: BackgroundHiveSplitLoader

Python Meta Query

Implementation - Generate TableHandle





Scripting PTF

Overview

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      LONG ~~~~ CODE
    $$
));
```

We want to store and reuse it!

Scripting PTF

Overview

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      LONG ~~~~ CODE
    $$
));
```

```
CREATE SCRIPT 'parse_access_log'
returns (
  id integer,
  name varchar
)
AS $$
  LONG ~~~~ CODE
$$
```

Scripting PTF

DEMO

Scripting PTF

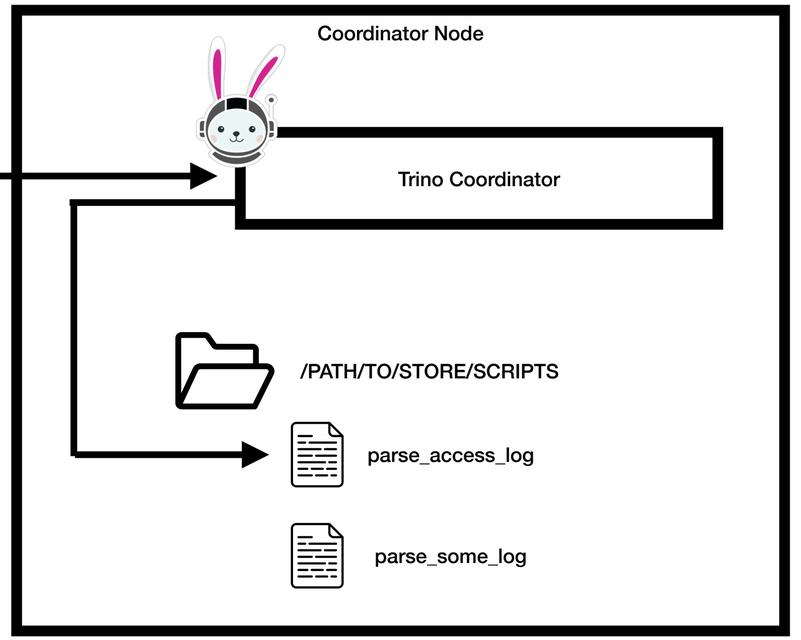
Implementation - Extends SQL

```
CREATE (OR REPLACE)?
        SCRIPT name=string
        RETURNS '(' tableElement (',' tableElement)* ')'
        AS script=function_body    #createOrReplaceScript |
DROP SCRIPT name=string            #dropScript |
SHOW SCRIPTS                       #showScripts
```

Scripting PTF

Implementation - Storage Logic

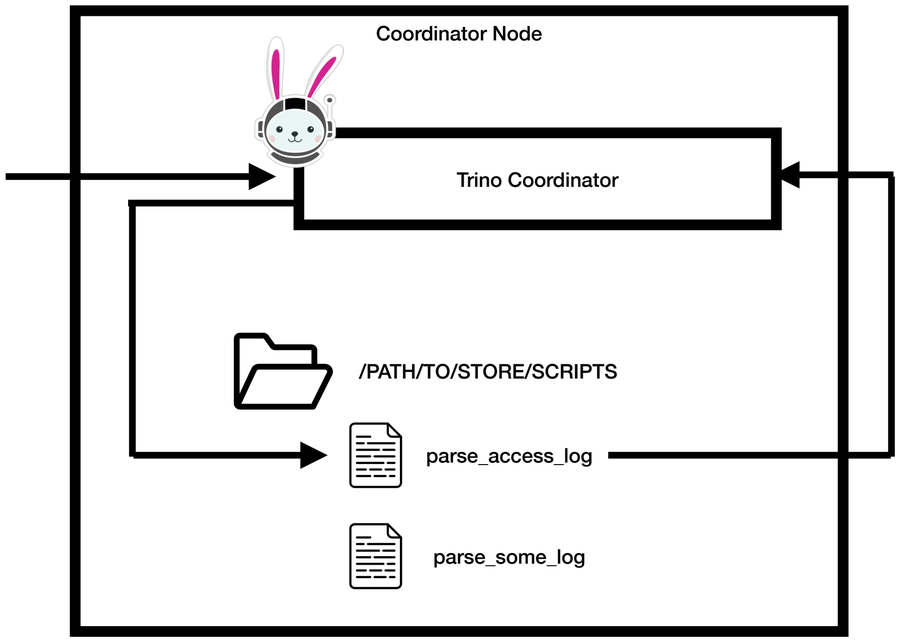
```
CREATE SCRIPT 'parse_access_log'
returns (
  id integer,
  name varchar
)
AS $$
  LONG ~~~~ CODE
$$
```



Scripting PTF

Implementation - Transformation

```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    SCRIPT => 'parse_access_log'
));
```




```
SELECT
  *
FROM
  table(hive.system.py_file_query(
    FILES => ARRAY['/PATH/TO/READ'],
    RETURNS => DESCRIPTOR(
       id integer,
       name varchar
    ),
    CODE => $$
      LONG ~~~~ CODE
    $$
));
```

What’s next

Improve Split Generation

Easy Debug

Improve Predicate

Thanks to

Jennifer Oh

Seonghwa Ahn

Emerging DP

Starburst 

Q & A