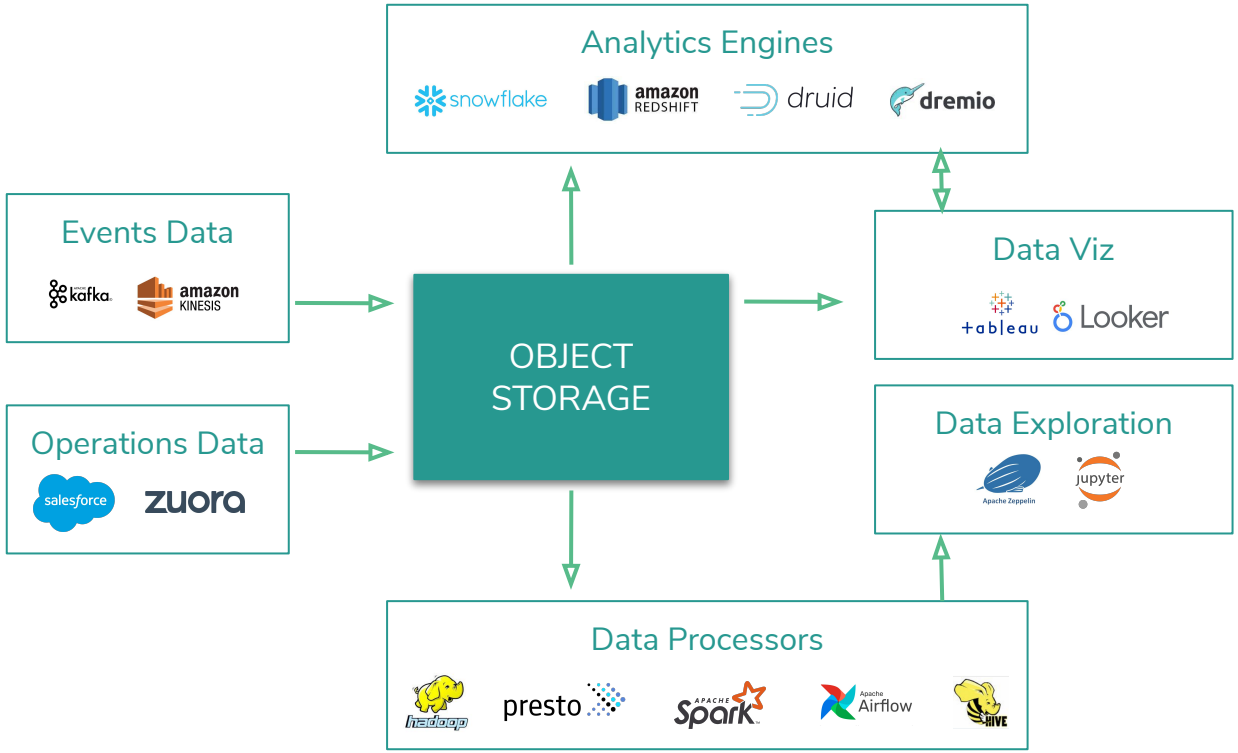
Introducing **lakeFS**

April 2021

lakeFS

Data Lake Architecture **Reference**



The Data Lake **Advantage**

1. **Scalability and cost effectiveness**
2. **Accessibility and ease of use**
3. **High throughput**
4. **Rich application ecosystem**

The Data Lake **Challenge**

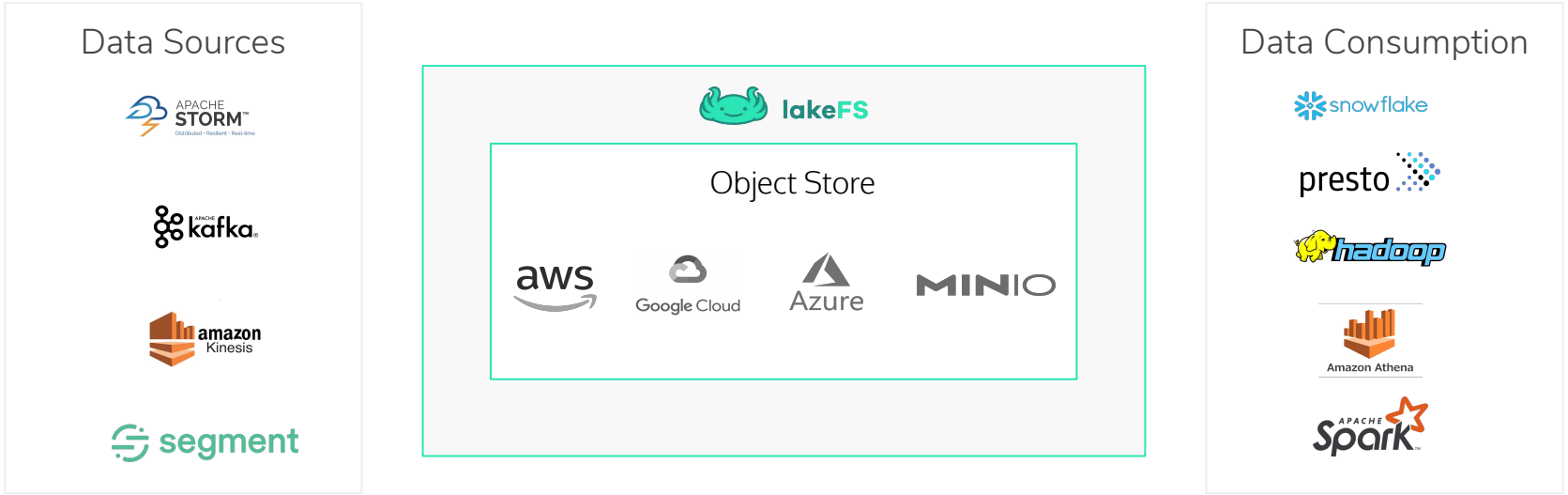
1. **In-ability to experiment, compare and reproduce**
   Example: add new metric for BI
2. **Difficult to enforce data best practices**
   Example: schema, format enforcement
3. **Hard to ensure high quality data**
   Example: validate statistical properties of the data

In a perfect world

we would manage data from dev to production the way we manage code

Atomic versioned data lake on top of object storage

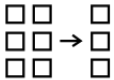
Manageability & Resilience Layer



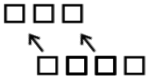
How it works

API Compatibility
with object stores

Versioning Engine
Transactional, atomic, isolated



Git Terminology
Branch, commit, merge



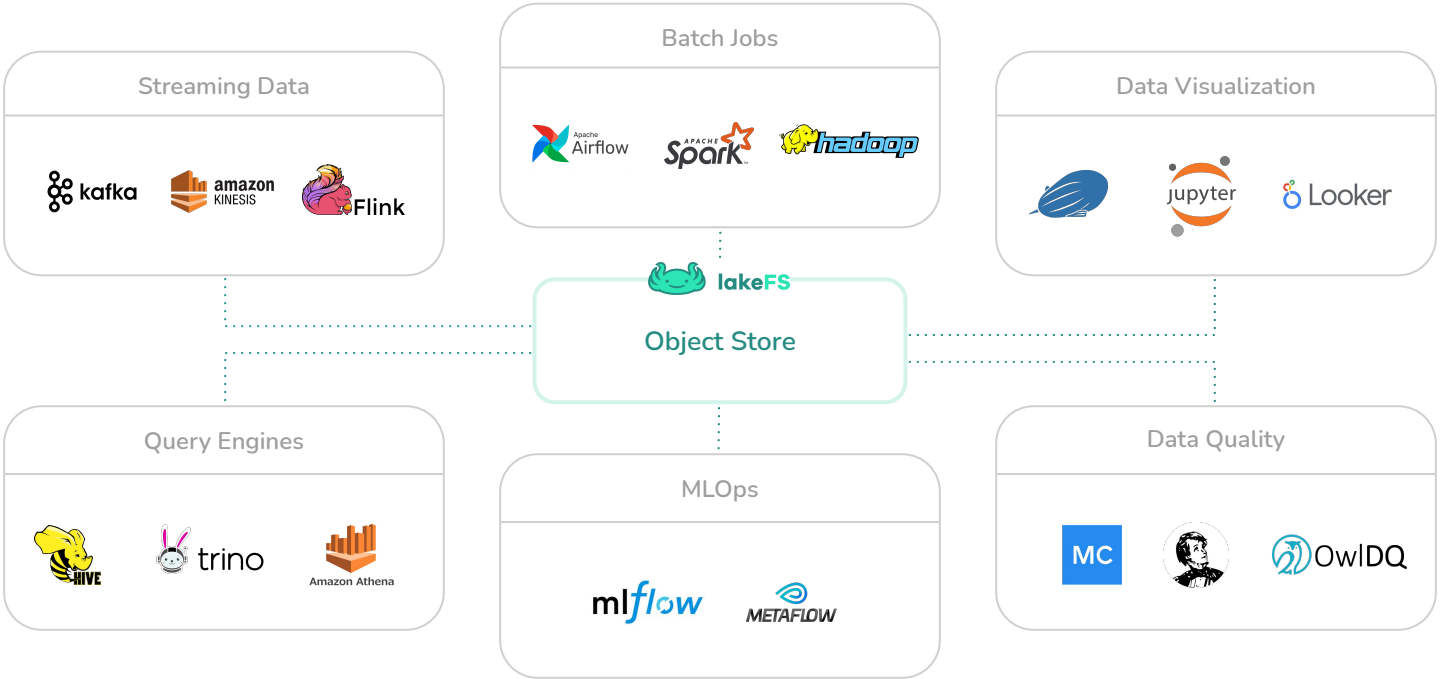
How it works

s3://data-bucket/collections/foo

s3://data-bucket/main/collections/foo

```
# Will create an external table using the same partitions and configurations
# but pointing at paths under the new branch
Lakectl metastore copy \
       --from-table user_events \
       --to-branch dev-experiment-1
```

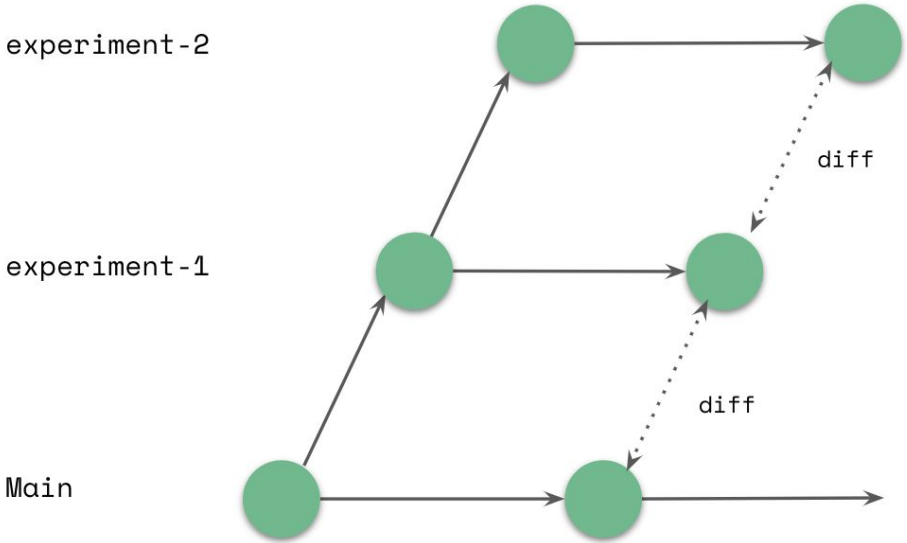
Integrates with your existing tools



Development Environment for Data

- **Experimentation** - try tools and code in isolation
- **Reproducibility** - go back to any point of time for both your code and your data
- **Compare** - tools, code or different versions of your data

Experiment Safely



Experimenting with Spark

```
lakectl branch create \
   lakefs://example-repo@testing-spark-3 \
   --source lakefs://example-repo@main
# output:
# created branch 'testing-spark-3.0', pointing to commit ID: 'd1e9adc71c10a'
```

```
val dfExperiment1 = sc.read.parquet("s3a://example-repo/experiment-1/events/by-date")
val dfExperiment2 = sc.read.parquet("s3a://example-repo/experiment-2/events/by-date")

dfExperiment1.groupBy(" ... ").count()
dfExperiment2.groupBy(" ... ").count() // now we can compare the properties of the data itself
```

Experimenting with Presto

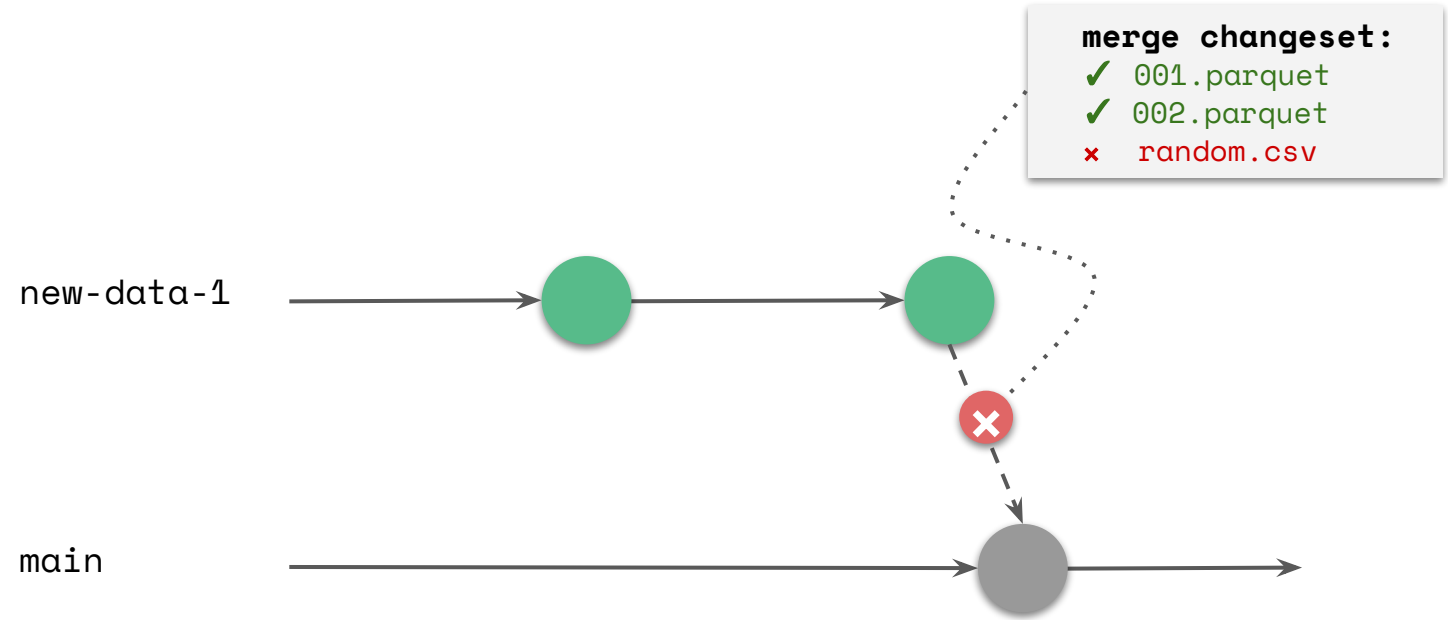
```
   lakectl branch create \
      lakefs://example-repo@testing-spark-3 \
      --source lakefs://example-repo@main
   # output:
   # created branch 'testing-spark-3.0', pointing to commit ID: 'd1e9adc71c10a'
```

```
CREATE TABLE master.request_logs (
  request_time timestamp,
  url varchar,
  ip varchar,
  user_agent varchar
)
WITH (
  format = 'TEXTFILE',
  external_location = 's3a://example/main/data/logs/'
);
```

Continuous Data Integration

- **Ingest new data safely**
- **Enforce best practices**
- **Metadata validation -** prevent breaking changes from entering your production data environment

Enforce Best Practices



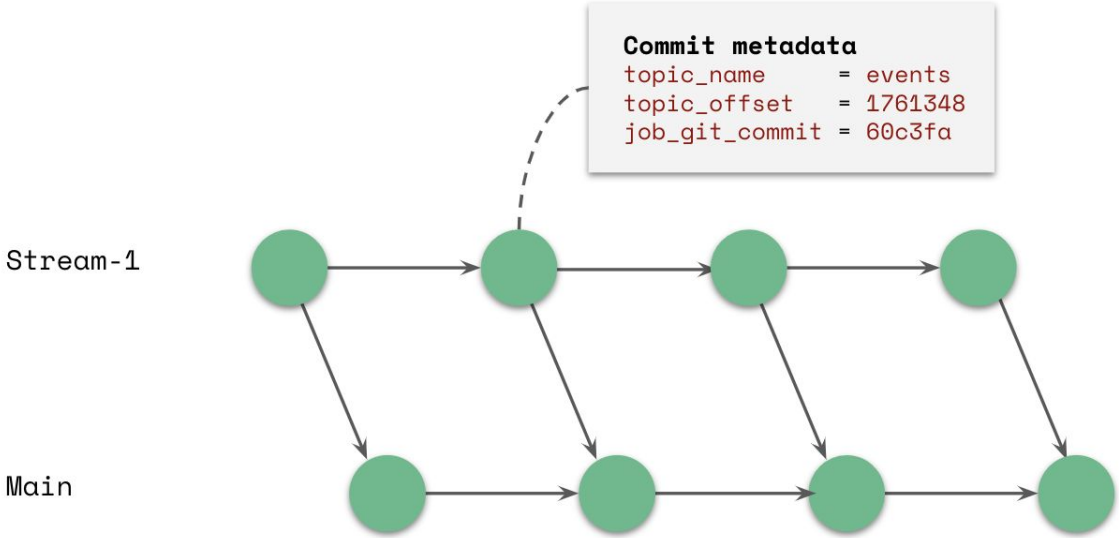
CI with PyArrow

```
import lakefs
import pyarrow.parquet as pq

# Setup a PyArrow FileSystem that we can use to query data in the source ref
fs = lakefs.get_filesystem(repo, source_ref)

for change in lakefs.diff(repo, source_ref, target_branch, prefix='public/'):
    if not change.path.endswith('.parquet'):
        continue
    # Read Parquet column metadata
    schema = pq.read_schema(fs.open_input_file(change.path))
    if filter(lambda column: column.name == 'user_id', schema):
        raise ValidationError('user_id column not allowed')
```

Continuous Data Deployment

- **Prevent data quality issues -** by testing production data before exposing it to users / consumers
- **Test -** intermediate results in your DAG to avoid cascading quality issues
- **Instantly revert changes to data**

Streaming Data with Kafka Connect



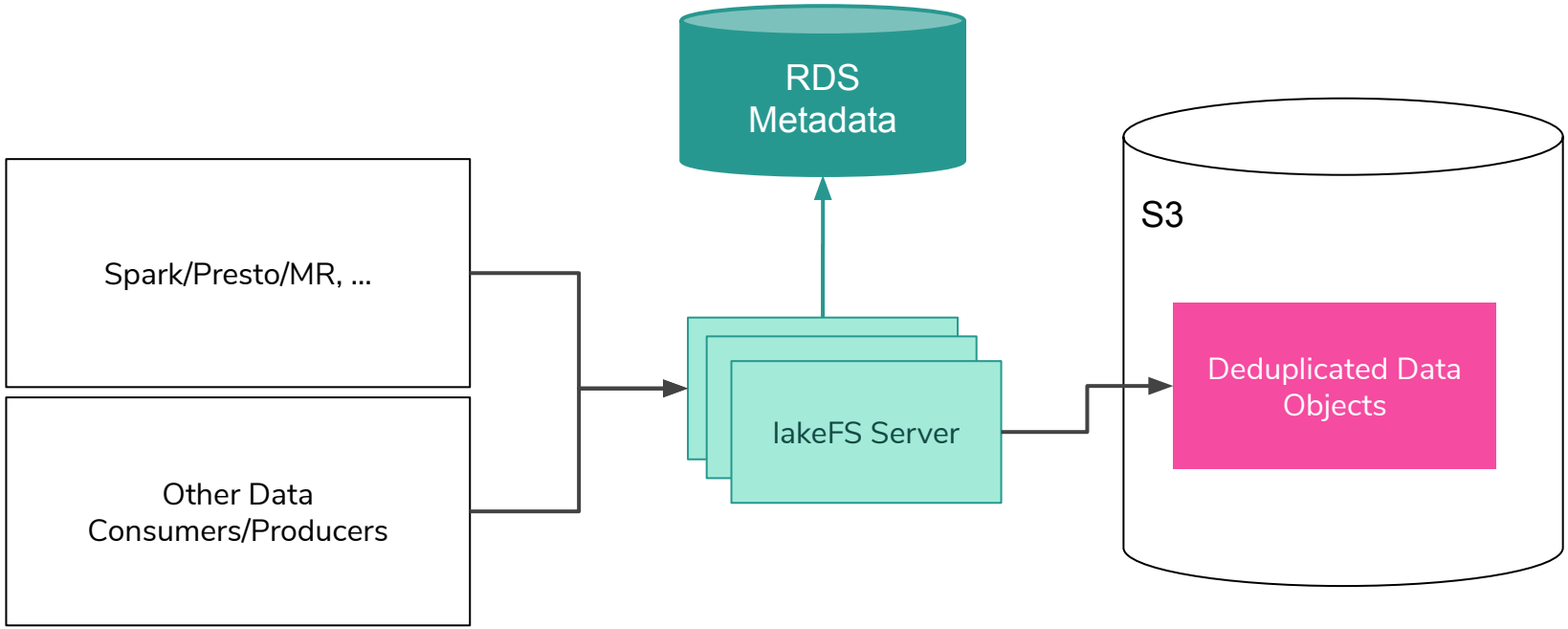
Streaming Data with Kafka Connect

```
# Your lakeFS repository
s3.bucket.name=example-repo

# Your lakeFS S3 endpoint and credentials
store.url=https://s3.lakefs.example.com
aws.access.key.id=AKIAIOSFODNN7EXAMPLE
aws.secret.access.key=wJalrXUtnFEMIK7MDENGbPxRfiCYEXAMPLEKEY

# master being the branch we want to write to
topics.dir=example-branch/topics
```

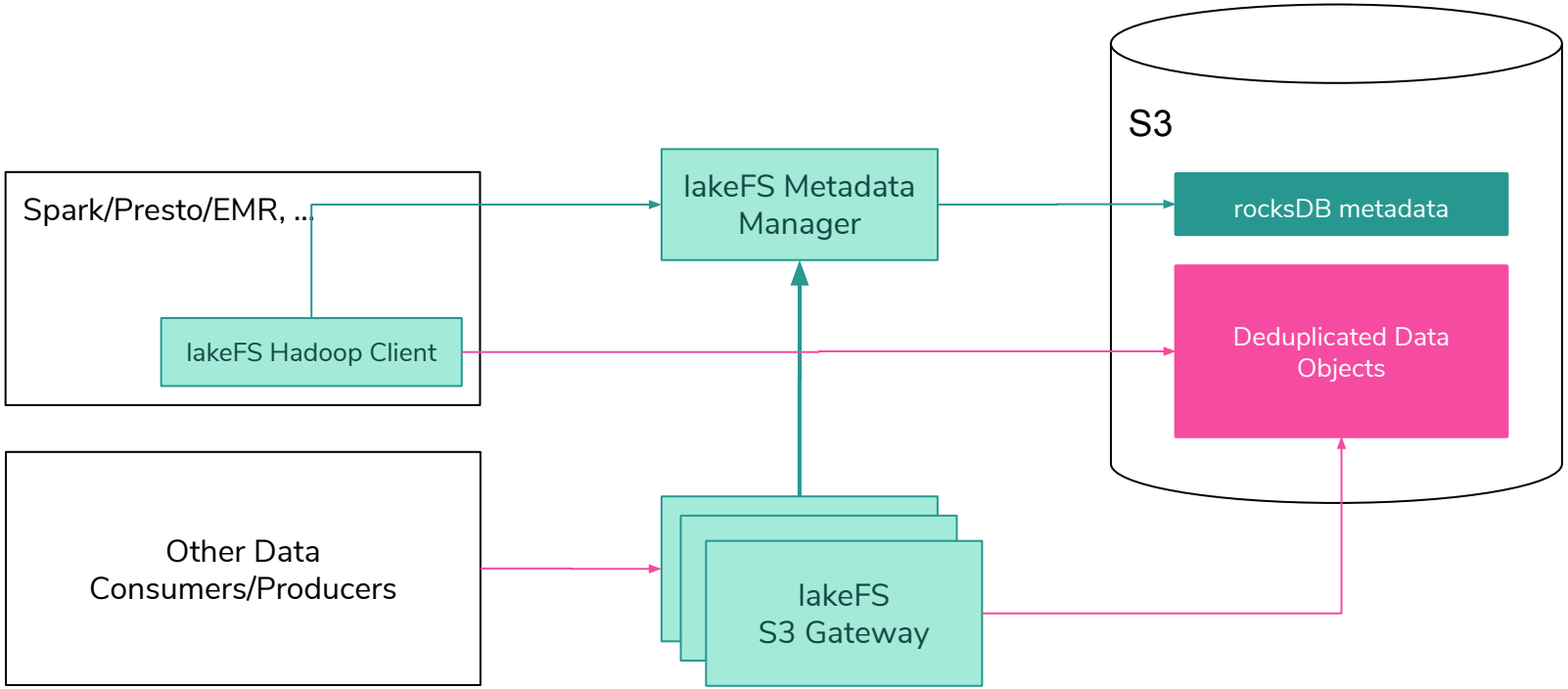
```
Lakectl branch revert lakefs://example-repo@stream-1-branch --commit dd8a60d5ef70809
```

lakeFS Architecture

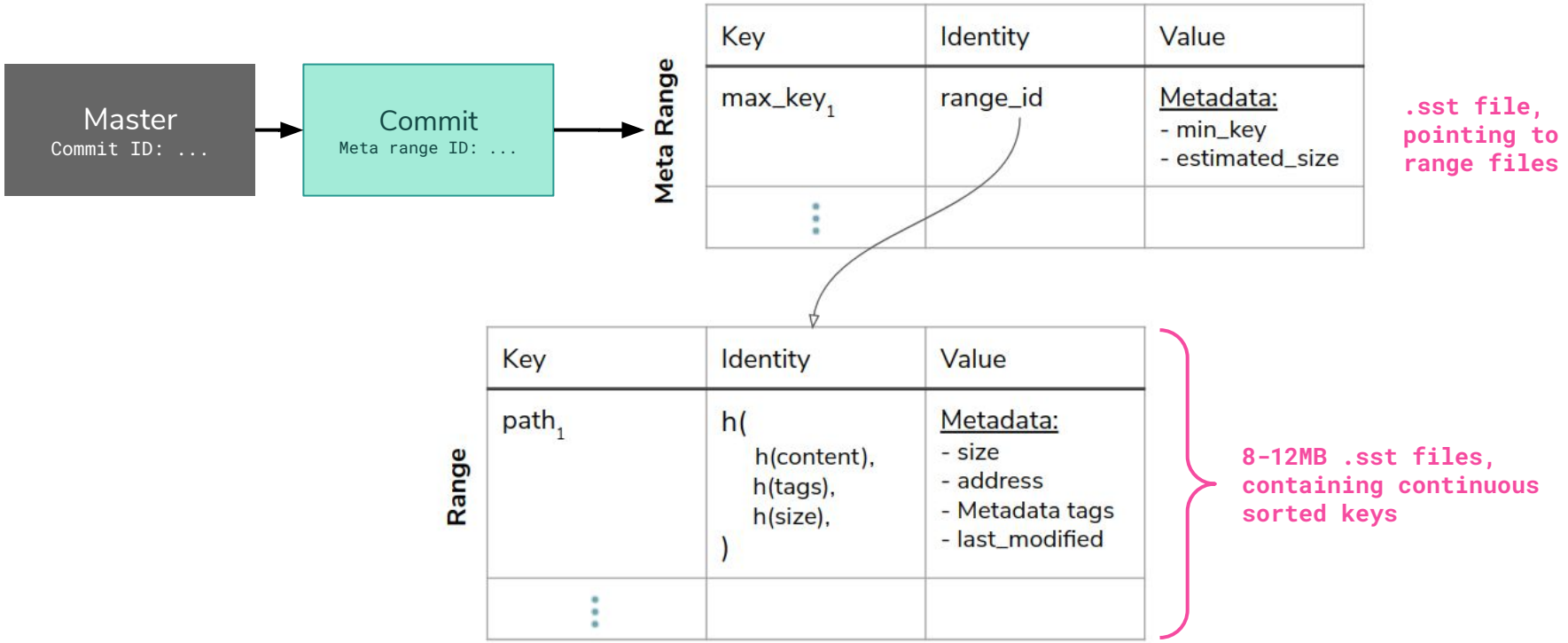
Old Architecture



In-Development Architecture



Future Architecture - Data Format



Further Reading

Additional Resources

Getting started

Check out the docs

Join the lakeFS Slack Channel

Contribute and star the repo

Thanks!

lakeFS

Join the community

GitHub

einat.orr@treeverse.io

The open core model