Collaboration on data with Lakehouses and Iceberg

Christian Thiel, Vakamo

+ Demo

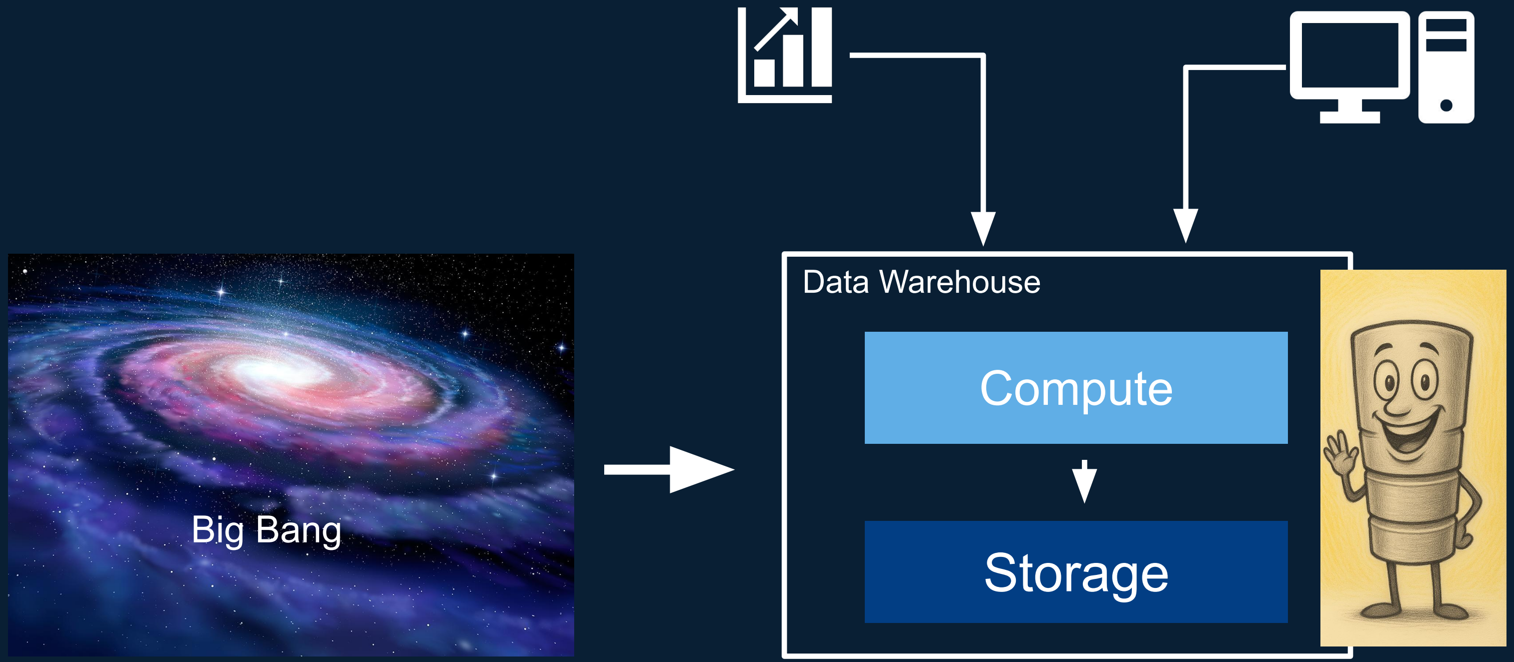
Vakamo

whoami

- Christian, OSS Enthusiast
- 📍 Hamburg
- Co-Founder & Software Engineer @ Vakamo
- Developing Lakekeeper
  A Rust-Native, Apache Licensed Iceberg Rest Catalog
- Apache Iceberg Contributor

A not too lengthy and reasonably accurate history of Data & Analytics systems

A long time ago …

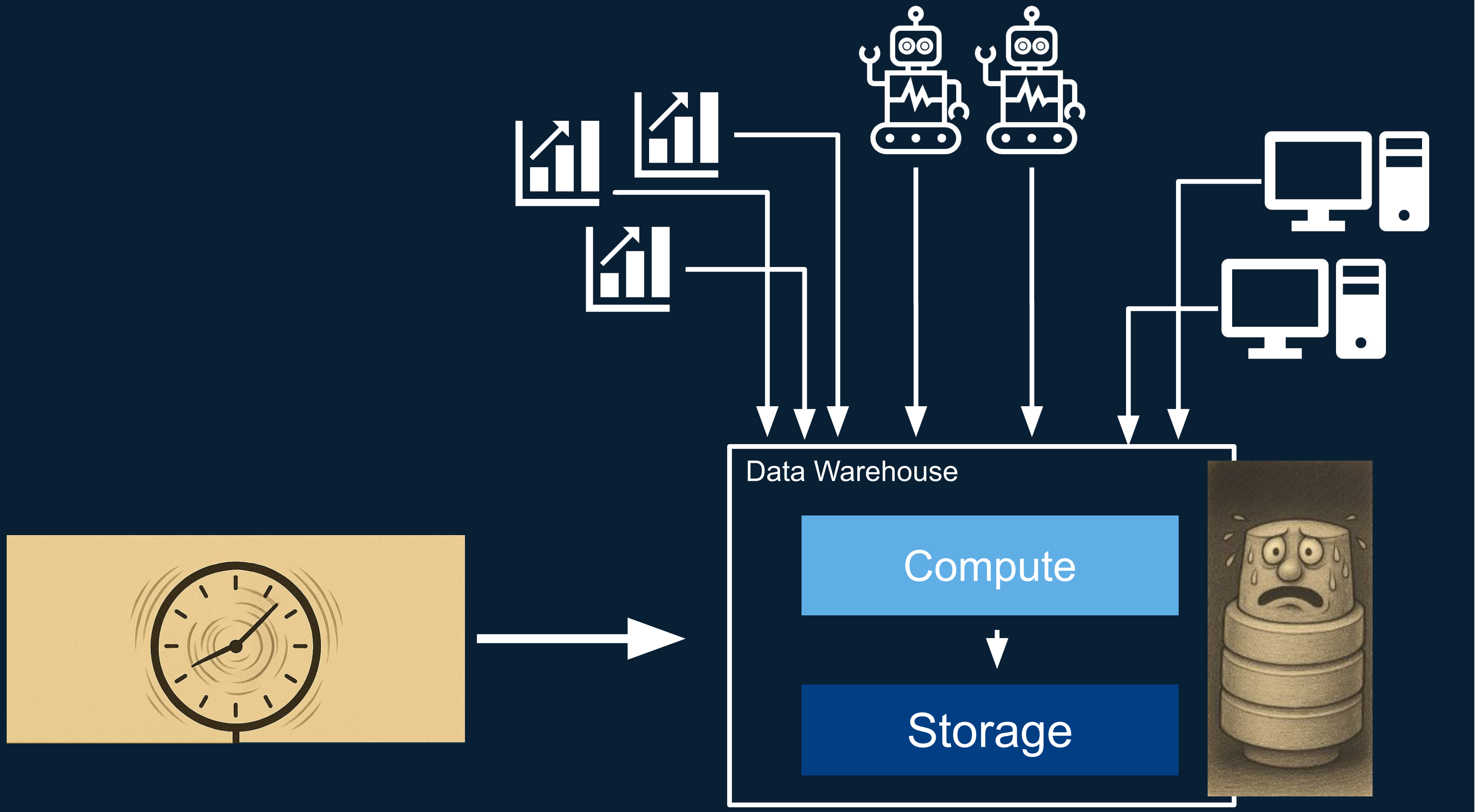


Transaction Safe (ACID)

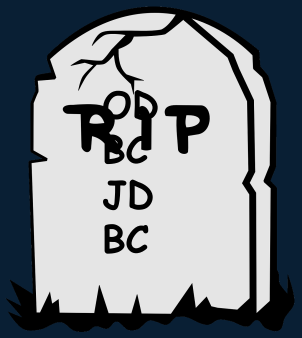
Schema Enforcement & Evolution

Governance & Access Control

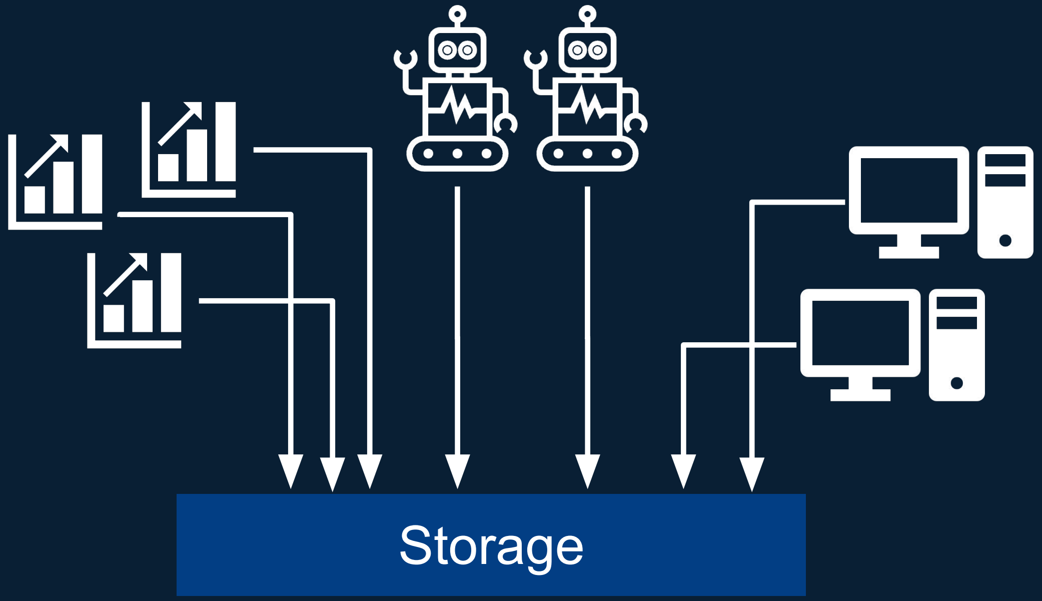
… more recently



- Central Compute is bottleneck
- Doesn’t scale well
- Expensive
- Very Slow Data Extraction
- Not open to all use cases and external Compute



Data Lakes

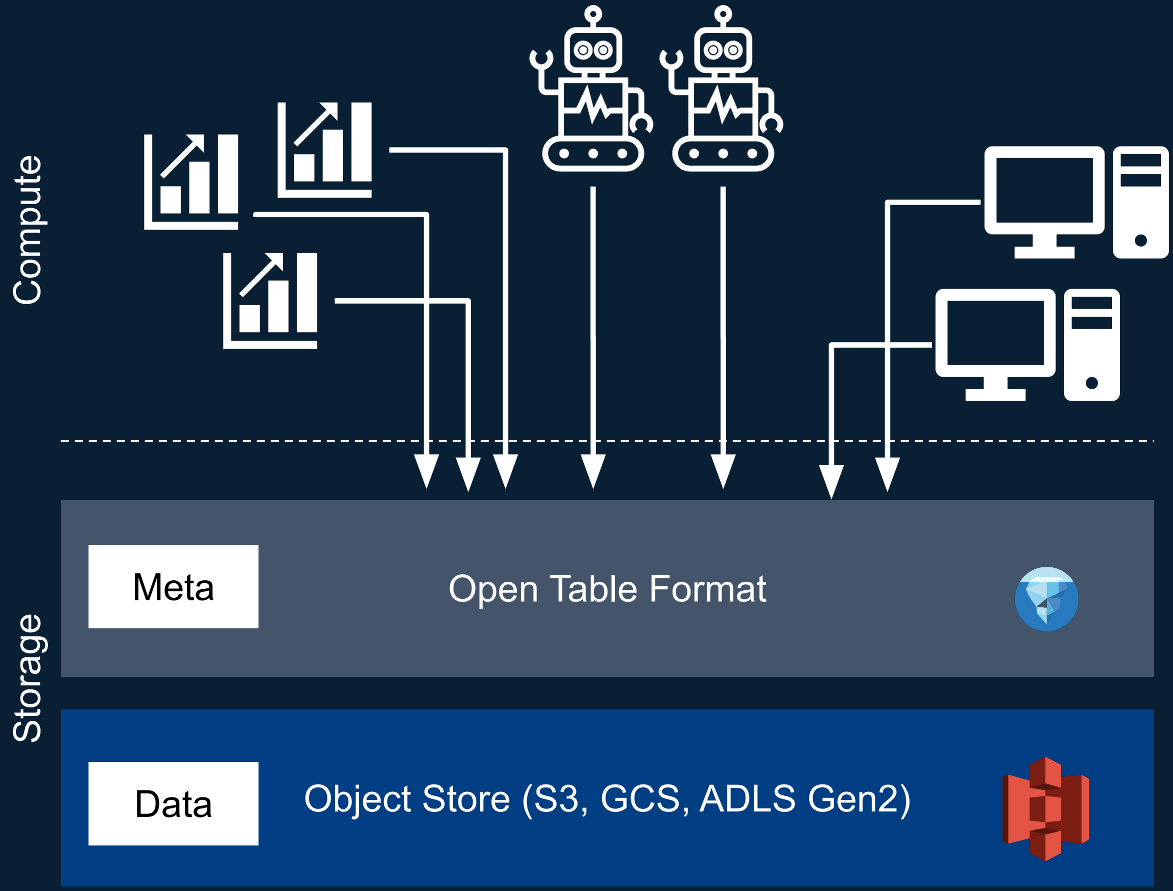


Scalable & Performant

Heterogeneous Compute Engines

Cost Reduction

Lakehouse / The Marketing Slide



Best of both Worlds

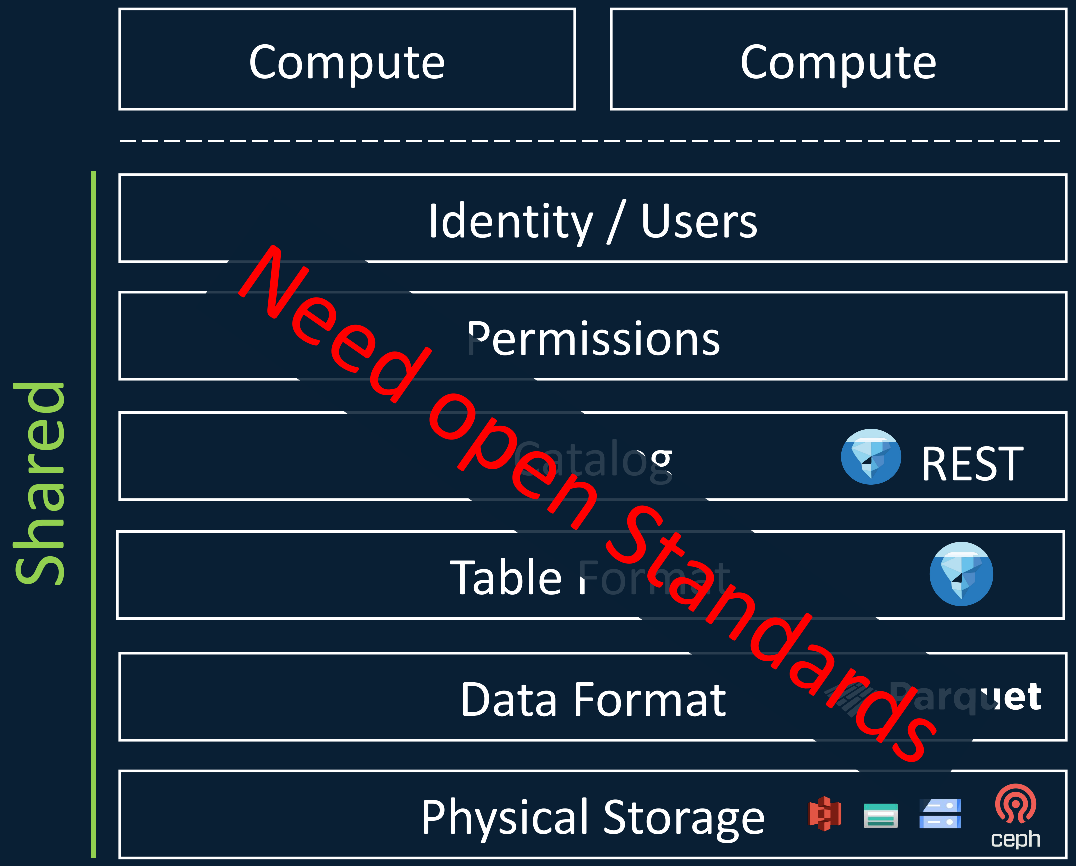
Warehouse

- Transaction Safe (ACID)
- Schema Evolution
- Governance

Data Lake

- Scalable & Performant
- Open, Heterogeneous Compute
- Cost Reduction

Collaborating on Data is Hard



Compute

Compute

Shared

Identity / Users

Permissions

Catalog

REST

Table Format

Data Format

Parquet

Physical Storage

ceph

Need open Standards

What is Apache Iceberg?

“The open table format for analytic datasets”

- Developed at Netflix 2017
- Apache Project since 2018

- High-Performant format for huge analytic tables
- Brings reliability and simplicity of SQL tables to analytical data

Contributors from all Major D&A Vendors

REST Catalog

OpenAPI Specification as part of the Iceberg Project
-> Multiple Implementations, i.e. Lakekeeper

- Catalog is Source-of-Truth for Table Metadata
- Transactions over multiple Tables (ACID)
- Table Discovery
- Change-based Commits
- Performant load of Table Metadata
- Hand out temporary Credentials for Data Access
- Language agnostic REST Interface, Replaces Legacy Catalogs:
  Hadoop, Hive, Glue, JDBC, Dynamo, ECS, Nessie, …

Supported by all Major Query Engines:

PyIceberg

Apache Spark

kafka

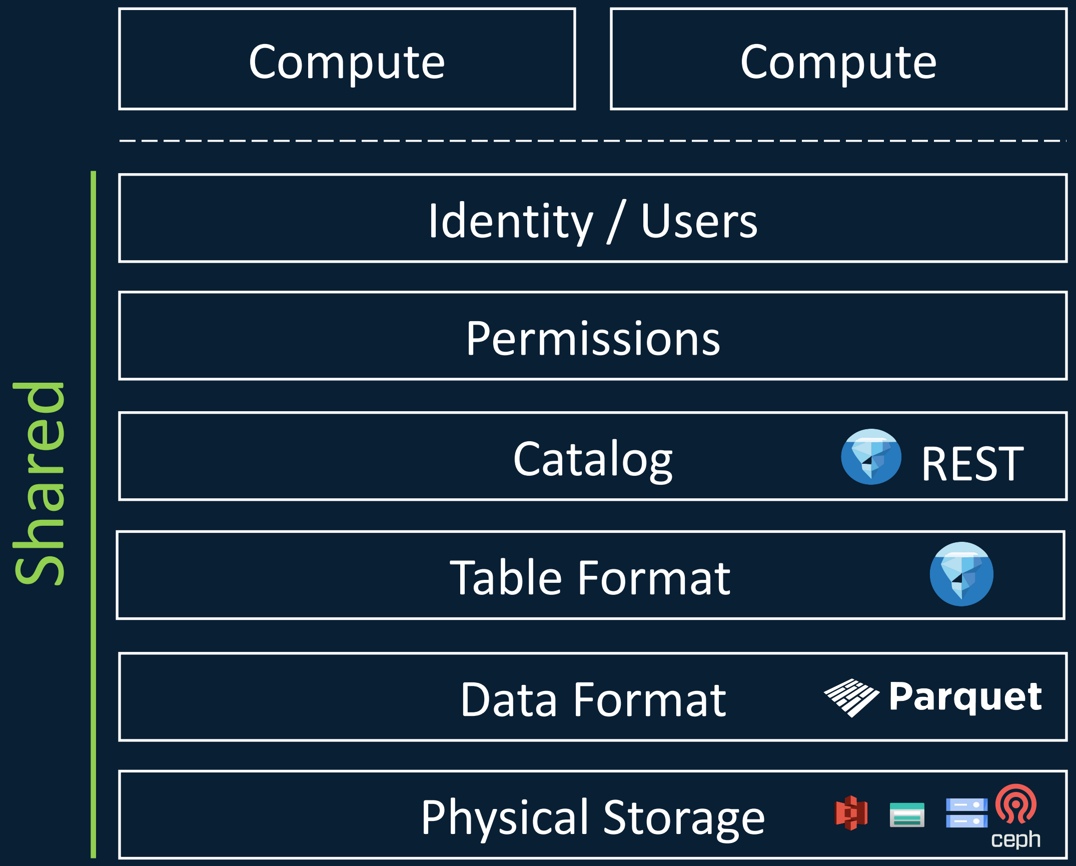
trino

DuckDB

Apache DATAFUSION

aws

Collaborating on Data is Hard



Compute | Compute

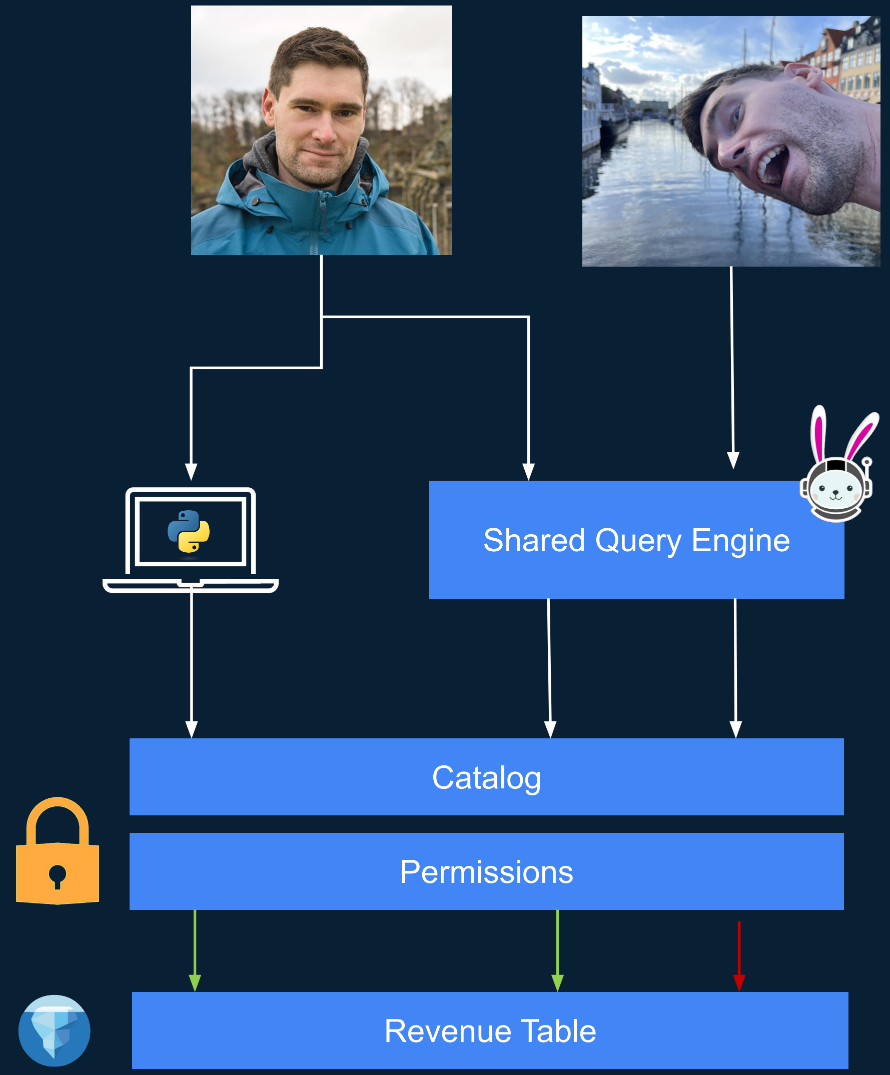
Shared

- Identity / Users
- Permissions
- Catalog — REST
- Table Format
- Data Format — Parquet
- Physical Storage — ceph

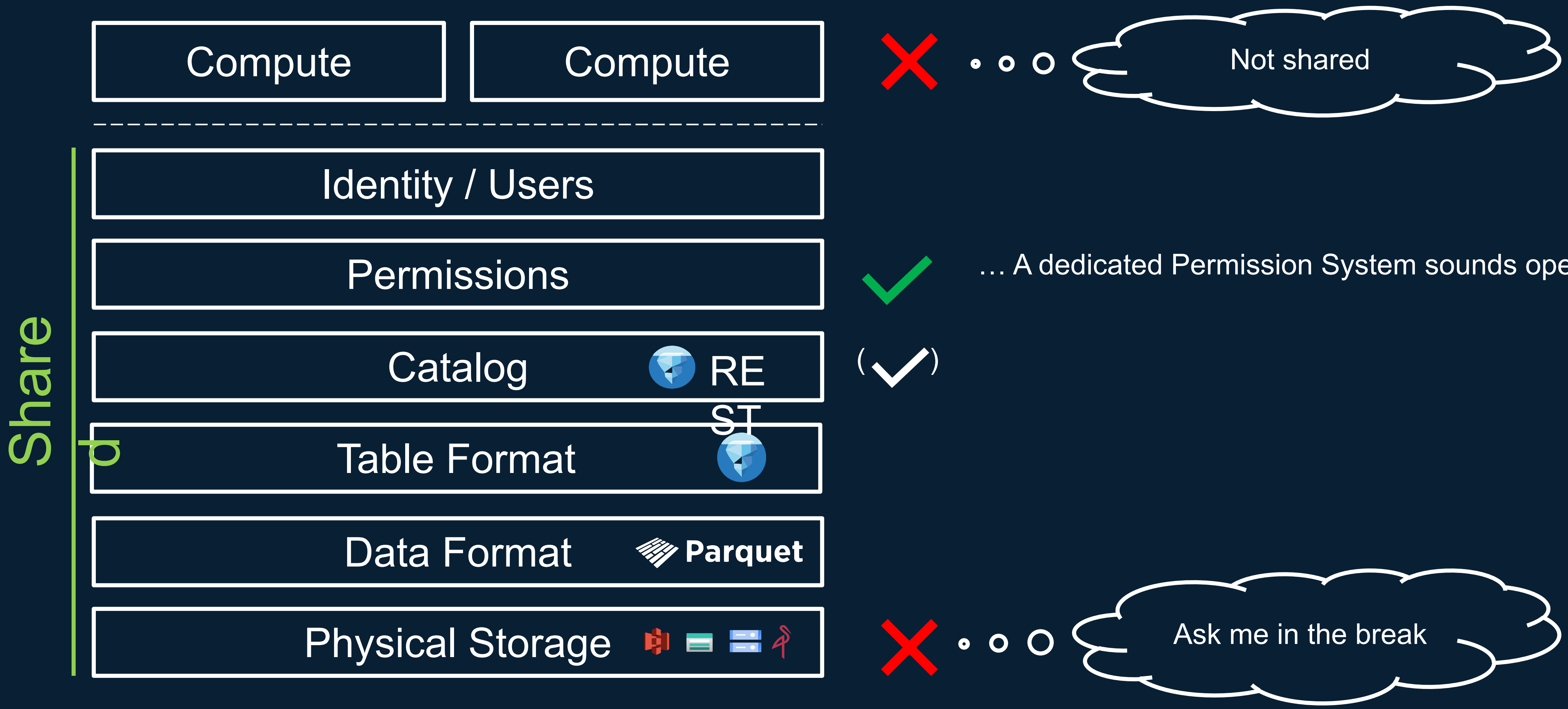
Confidential - Do Not Share or Distribute

The Goal

> ”
>
> **I need access controls for my Iceberg Tables!**
>
> - Naive Business Users



Where should permissions be Stored?



Compute

Compute

Identity / Users

Permissions

✓

… A dedicated Permission System sounds open

Catalog

RE
ST

(✓)

Table Format

Data Format

Parquet

Physical Storage

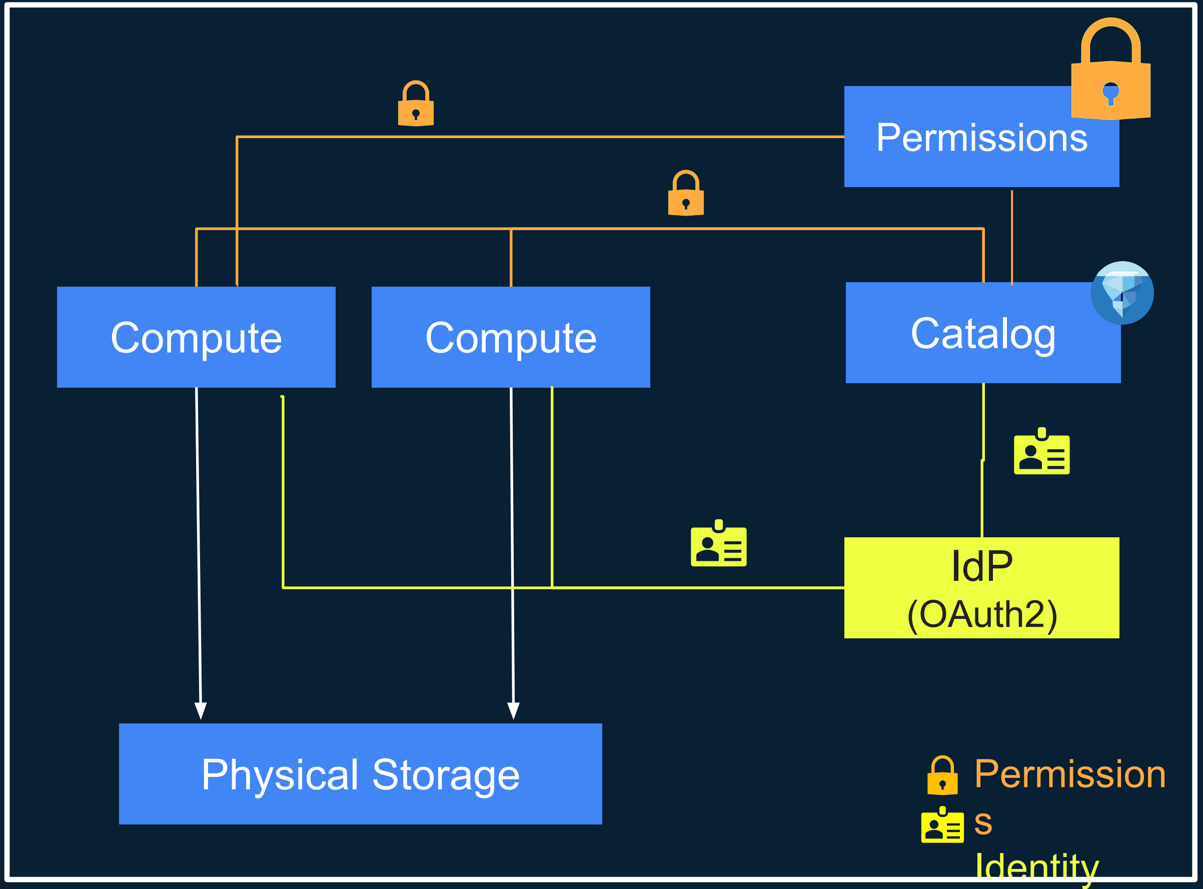
Share
d

✗ ∘ ○ ○ Not shared

✗ ∘ ○ ○ Ask me in the break

Permissions in Dedicated Systems

Open Policy Agent OpenFGA™



- Single place for shared permissions
- Semantic understanding with permission hierarchy
- Consistent across engines
- Share permissions across Catalogs and Table Formats and Applications

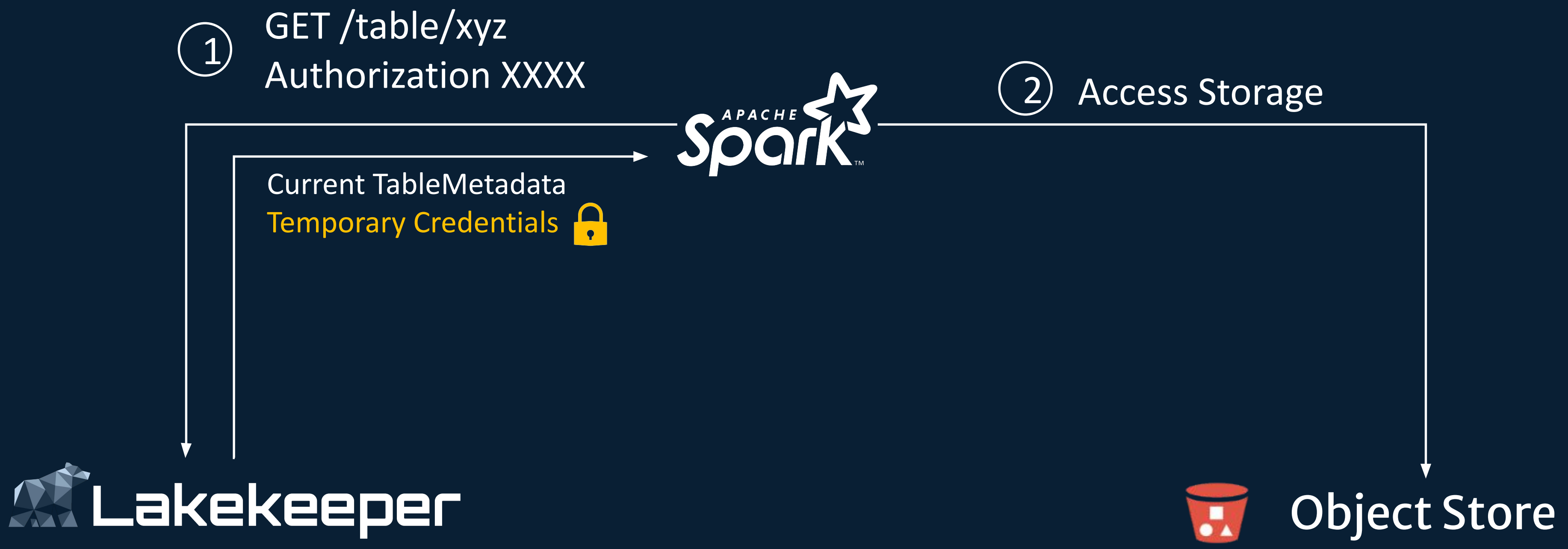
⚠ System that create PATs or Client Credentials or PWs are IdPs and a security risk!

How Permissions can be Enforced?

How can Security be Enforced?

1 Catalog enforces Permissions
(Iceberg REST Vended Credentials)

Catalog enforces Security: Vended Credentials



① GET /table/xyz
Authorization XXXX

Current TableMetadata
Temporary Credentials 🔒

② Access Storage

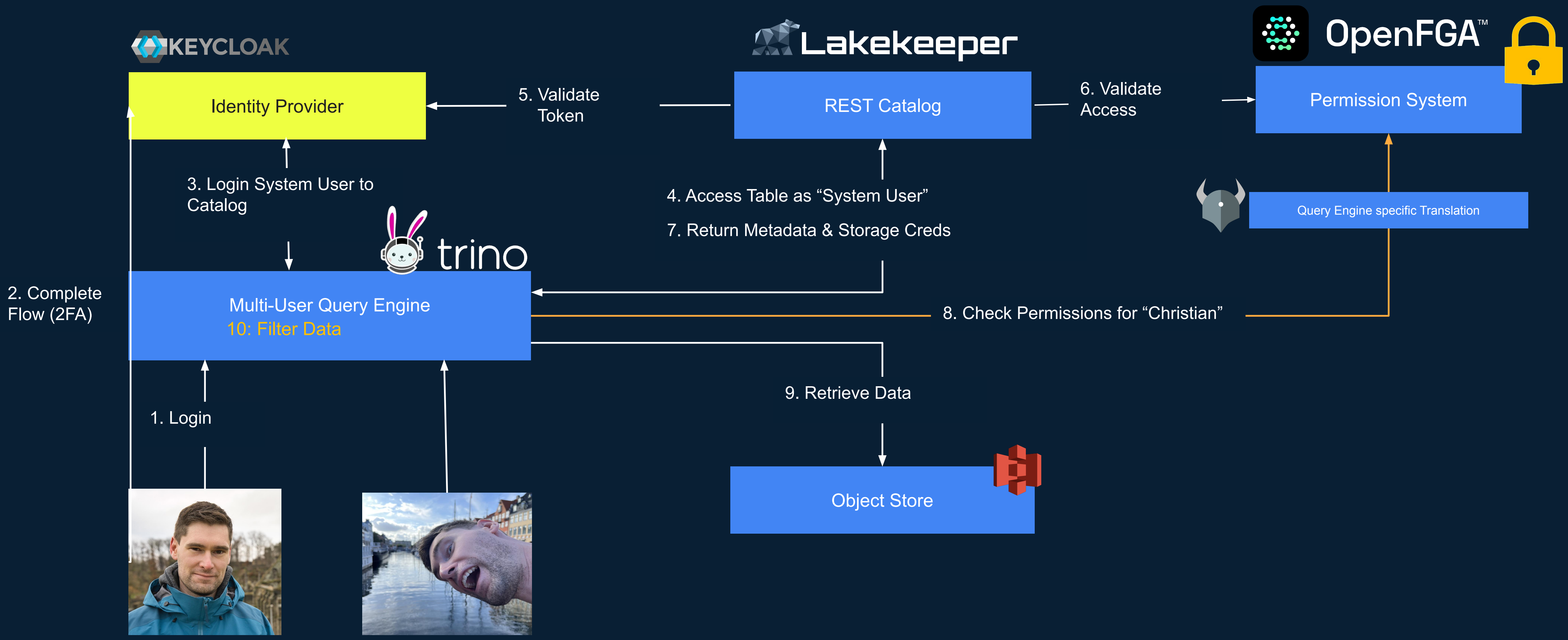
Vended Credentials & Remote Signing work on File-Level
-> Table-Level Security

How can Security be Enforced?

1. Catalog enforces Permissions
   (Iceberg REST Vended Credentials)

2. Compute enforces Permissions

Query engine enforced Security



KEYCLOAK

Identity Provider

5. Validate Token

Lakekeeper

REST Catalog

6. Validate Access

OpenFGA™

Permission System

3. Login System User to Catalog

4. Access Table as “System User”

7. Return Metadata & Storage Creds

Query Engine specific Translation

trino

2. Complete Flow (2FA)

Multi-User Query Engine

10: Filter Data

8. Check Permissions for “Christian”

9. Retrieve Data

1. Login

Object Store

How can security be enforced?

Catalog enforced Security

+ Table-Level Security

+ Doesn’t rely on filtering in client: Works for untrusted Clients

+ Standardized: Works with all Iceberg REST Clients

– No row or column level security

Query engine enforced Security

+ Row / Column Level Security

+ Performance optimizations in Query Engine

○ Requires secure, isolated User Sessions

– Requires trusted compute

We need Both!

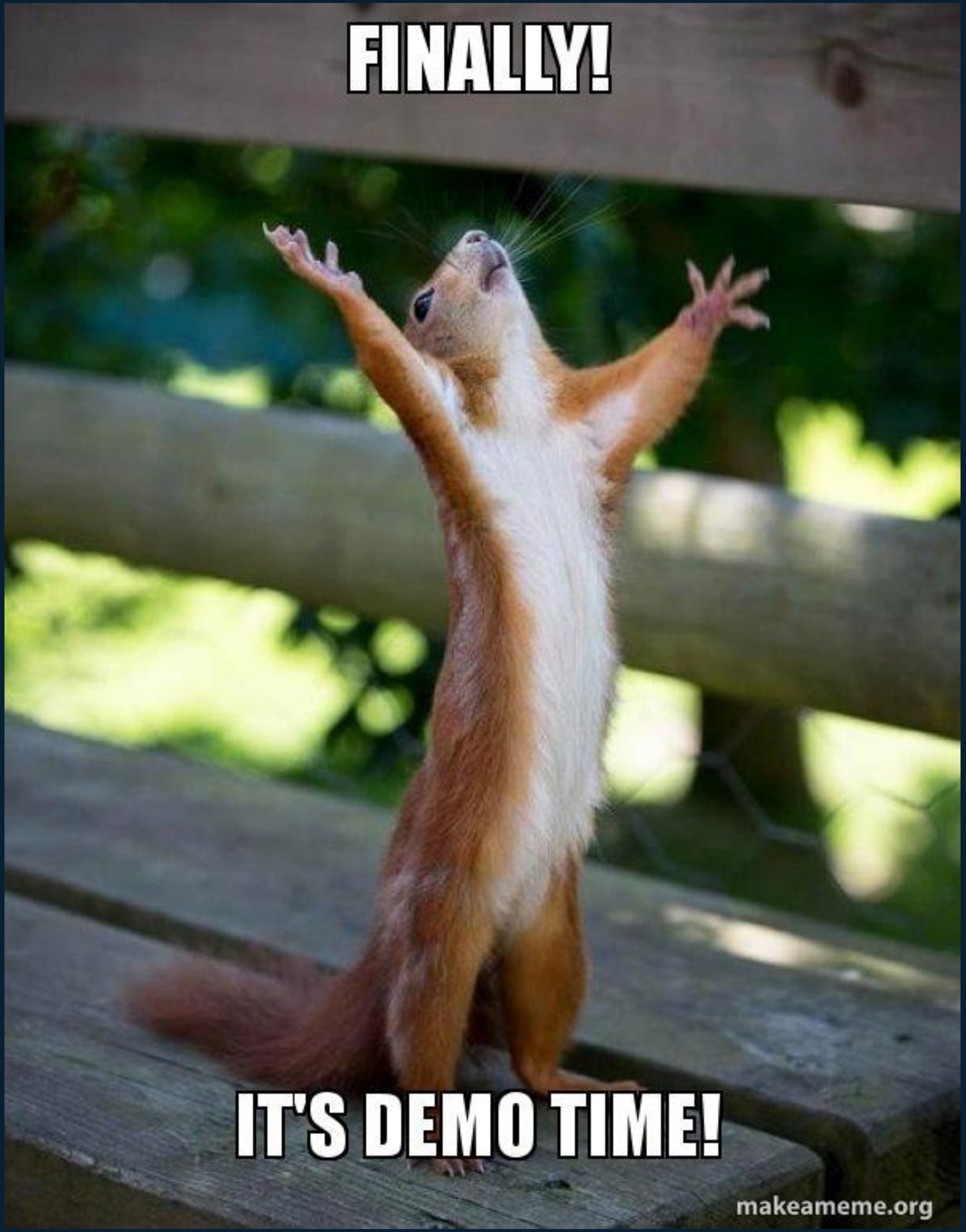
How can Security be Enforced?

1. Catalog enforces Permissions
(Iceberg REST Vended Credentials, Table Level)

2. Compute enforces Permissions
(Requires trusted Compute)

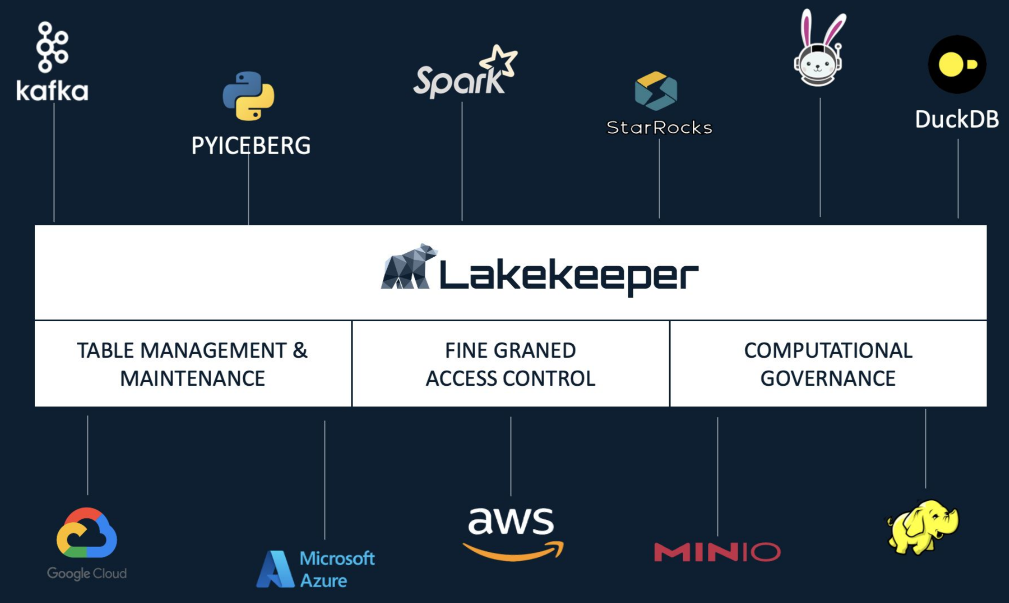
3. Storage enforces Permissions
(Requires Iceberg-Aware Storage)

4. Encryption of Files or Columns
(Application level)


FINALLY!
IT'S DEMO TIME!
makeameme.org

Lakekeeper

- Iceberg REST Catalog implementation
- Apache Licensed
- Written in Rust
- Kubernetes Integration
- Secure: OAuth2 and Permissions
- Production Ready & Easy to Use
- Extendible (Rust Traits!)
- Multi-IdP Support (Kubernetes + OIDC)



Open?!

LOUDER CROWDER
LOUDER CROWDER
Your Datawarehouse will never need another migration after moving to an open Iceberg Lakehouse.
CHANGE MY MIND

Permissions stored in open, independent System

Authentication using YOUR Identity Provider

Open Lakehouse with Open Governance is possible today!
Try it now!

```
git clone https://github.com/lakekeeper/lakekeeper.git
cd /examples/access-control-advanced
docker compose up

-> localhost:8888
```

1 Identity Provider
1 Source of truth for Permissions
Different Computes
Some of them shared
100% OSS

Lakekeeper

Lakekeeper is a Rust-Native, Apache Licensed Iceberg Rest Catalog