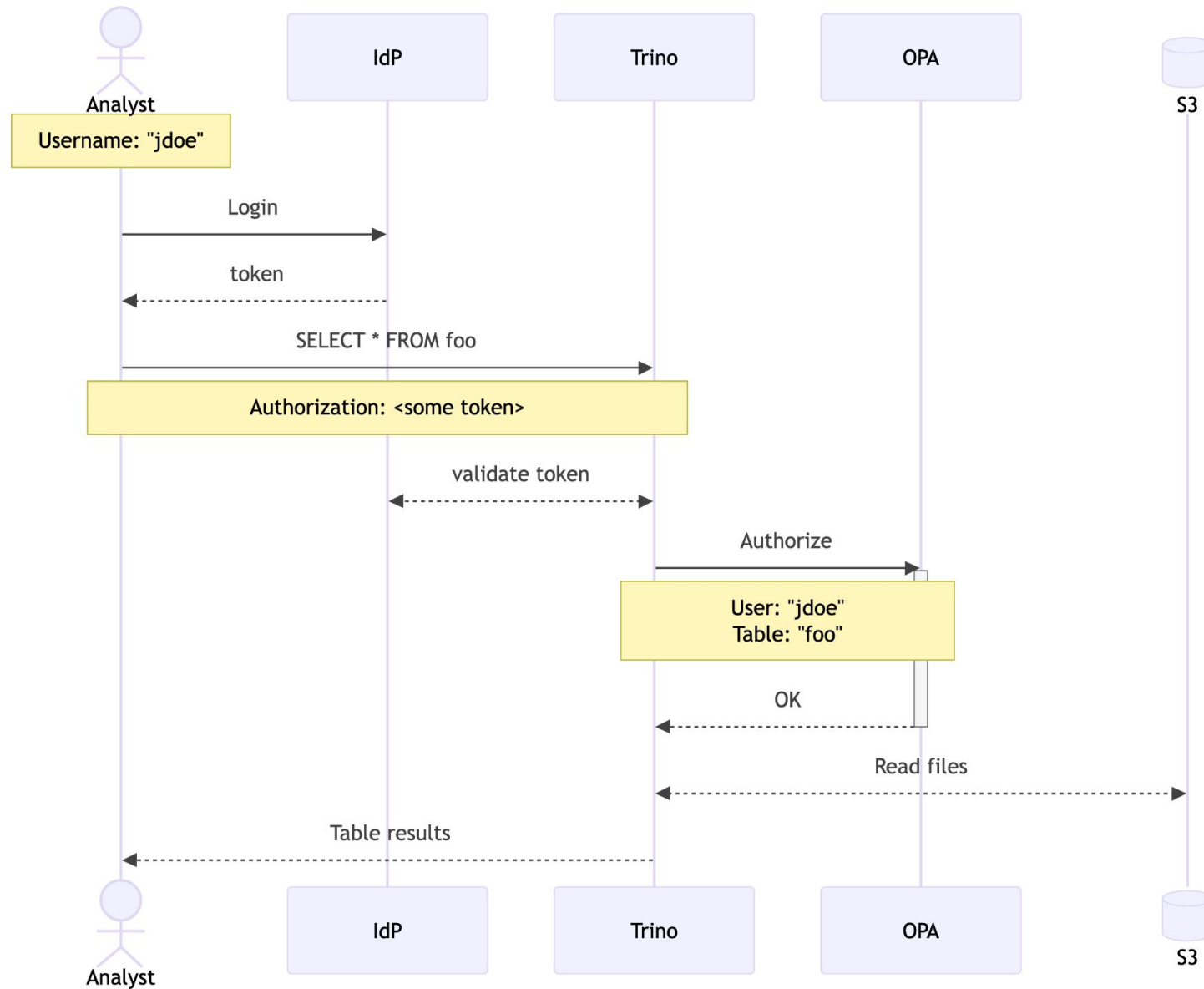
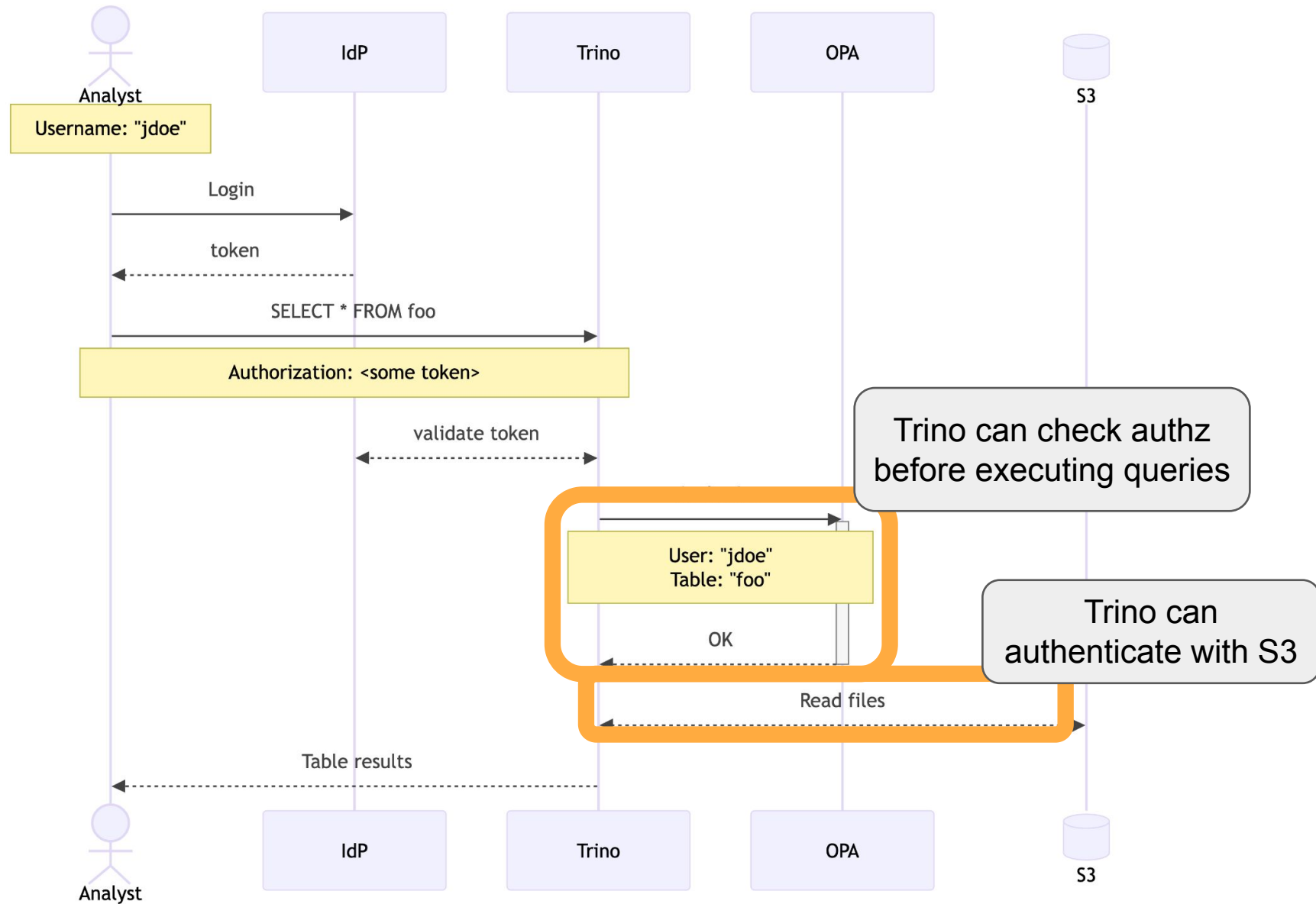


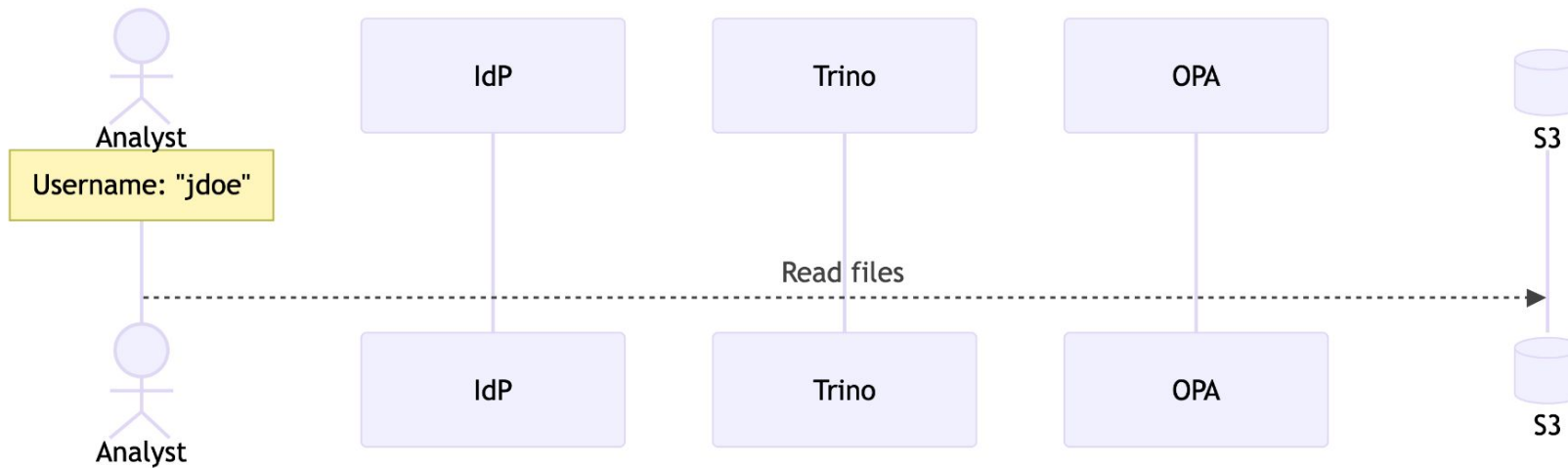
A **massively** over-simplified Apache Iceberg read process



A **massively** over-simplified Apache Iceberg read process



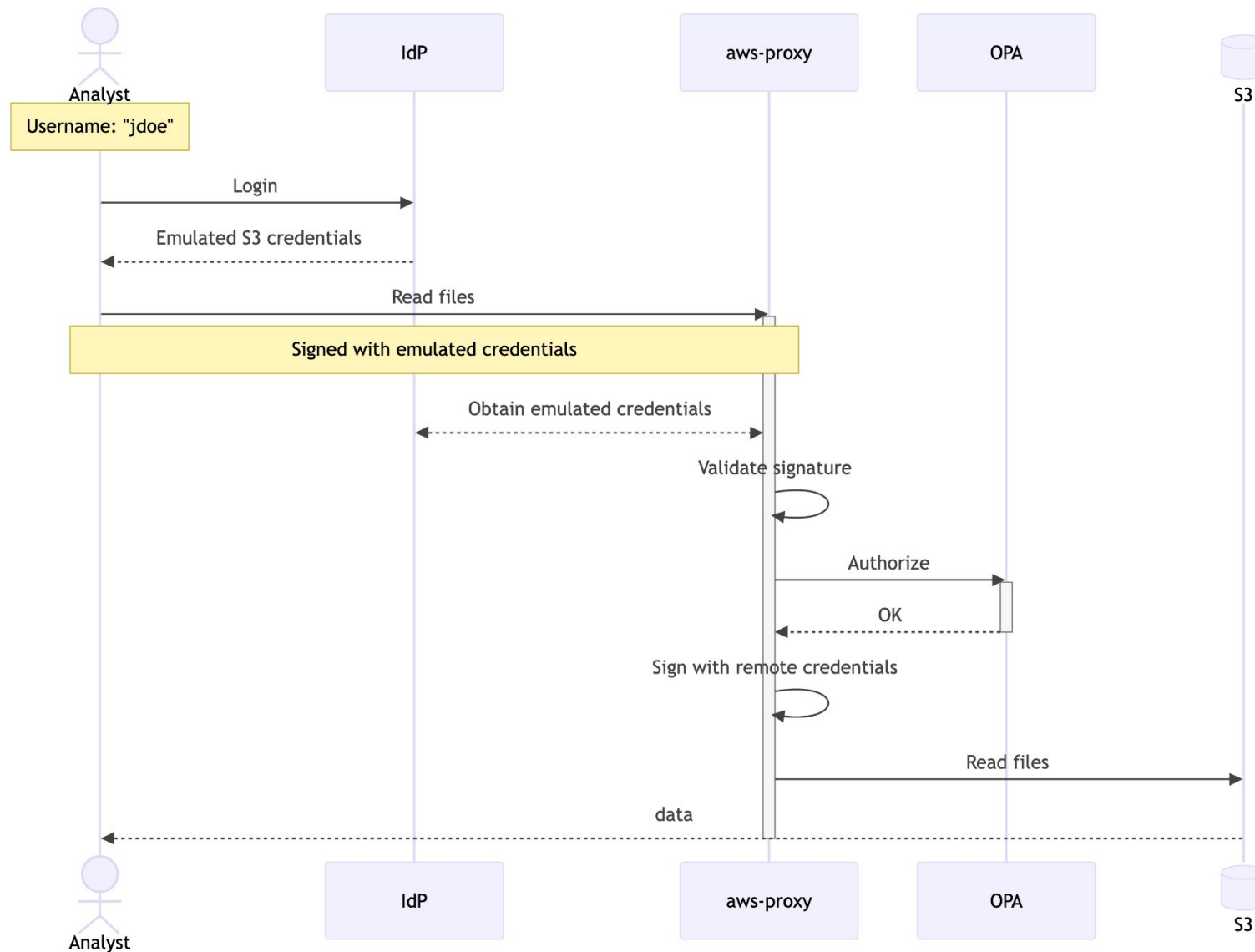
Trying to read Iceberg data from S3 directly





Some issues:

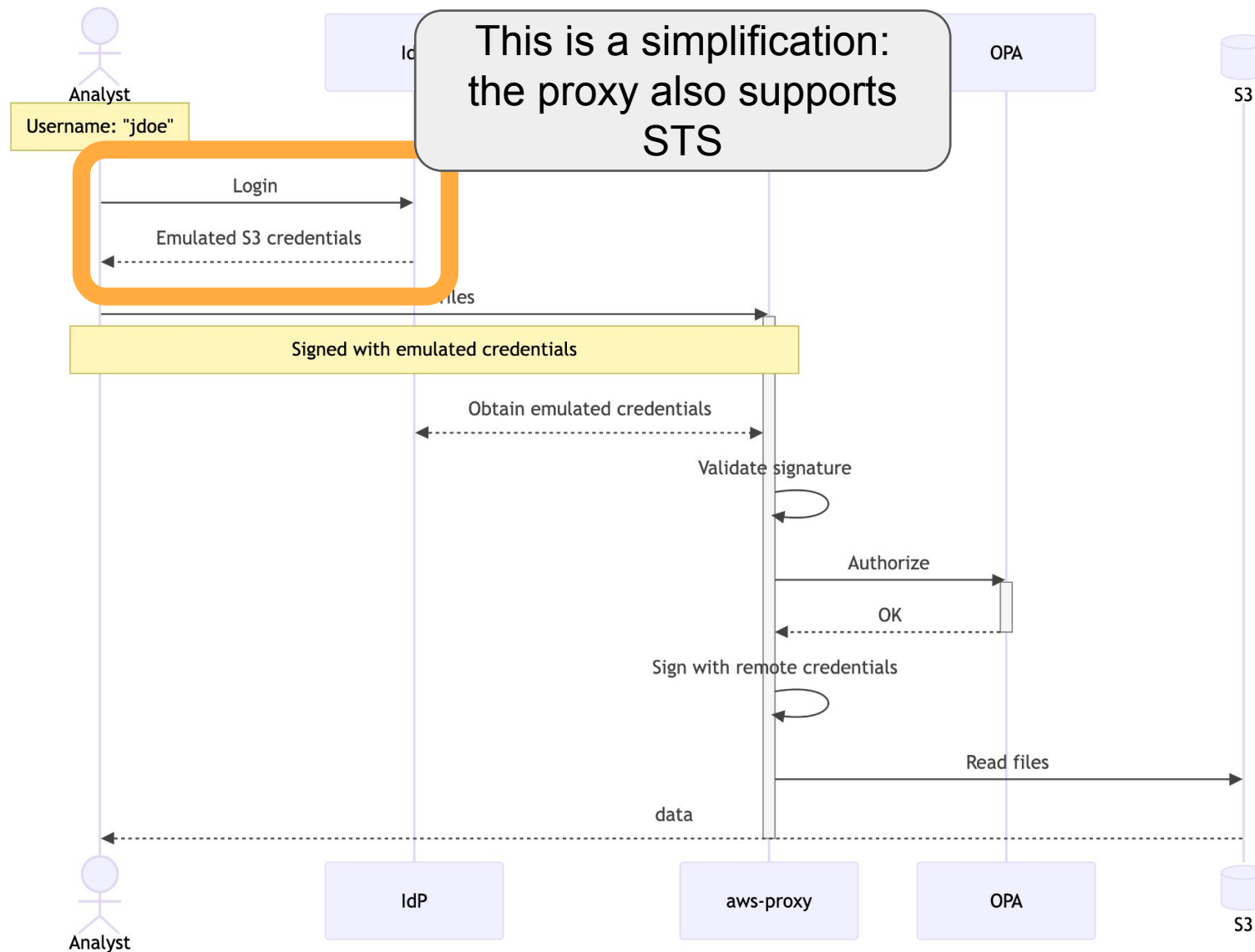
- User identity
 - SSO integration
 - Credential scoping
- How to enforce authz
- What Iceberg catalog?
- Heavily tied to the underlying storage



Trying to read Iceberg data from S3 directly



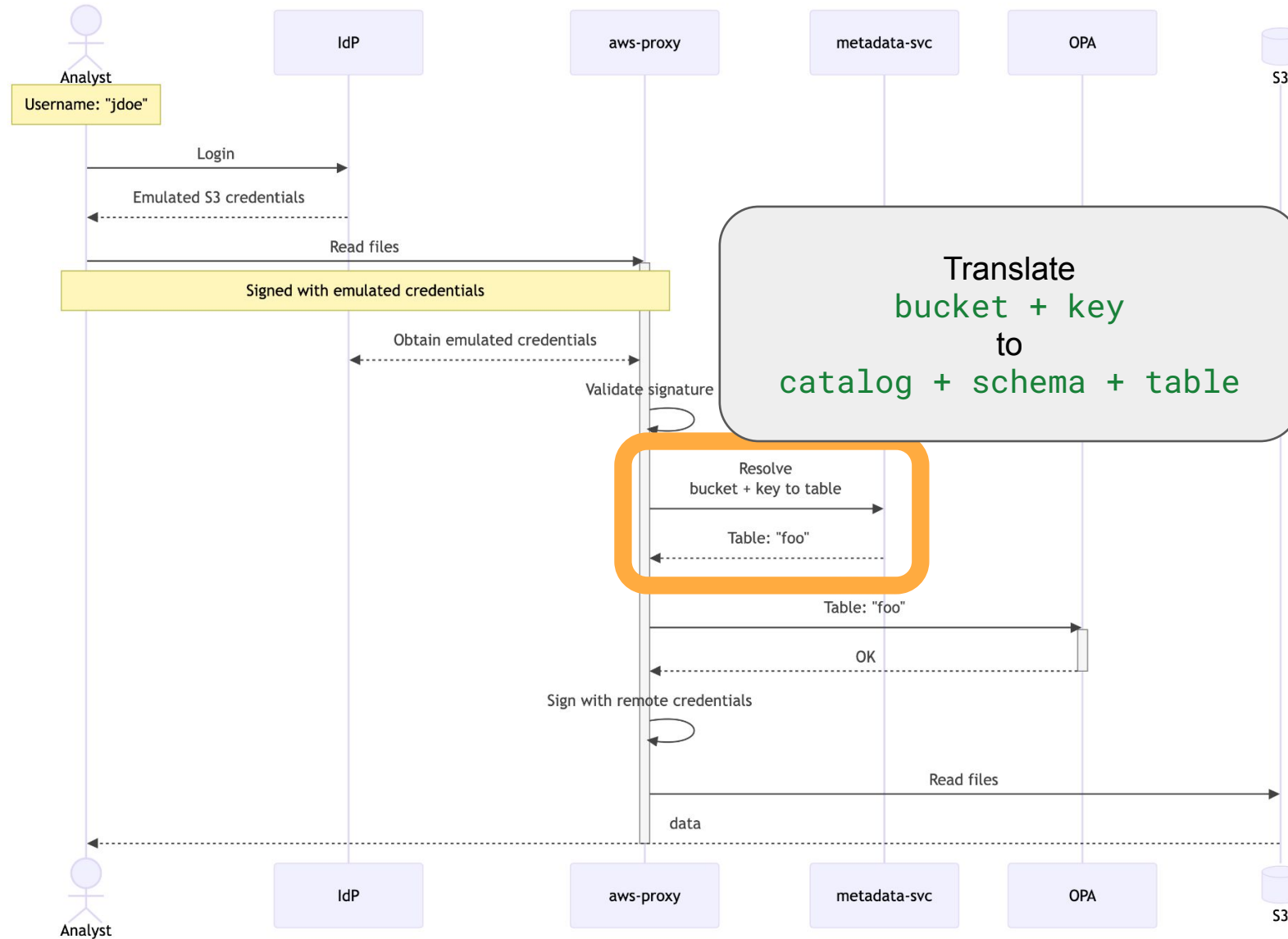
-  User identity:
 - SSO integration
 - Credential scoping
 - User never deals with real S3 credentials
-  How to enforce authz
 - OPA (or others!)
 - **? But what policy?**
- **? What Iceberg catalog?**
 - Let's assume user provides a specific metadata filename

Trying to read Iceberg data from S3 directly



-  User identity:
 - SSO integration
 - Credential scoping
 - User never deals with real S3 credentials
-  How to enforce authz
 - OPA (or others!)
 - **? But what policy?**
- **? What Iceberg catalog?**
 - Let's assume user provides a specific metadata filename

Trying to read Iceberg data from S3 directly

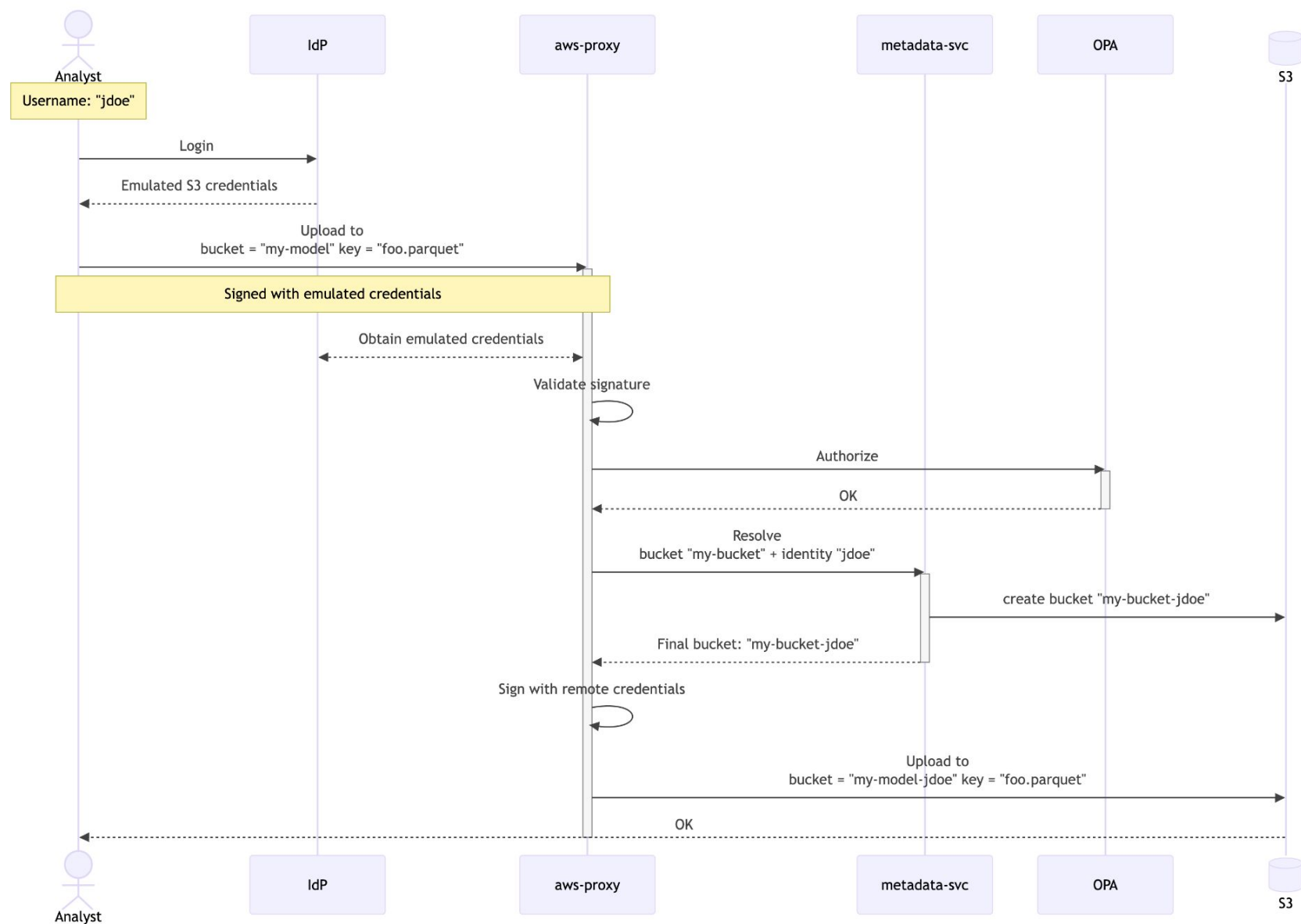


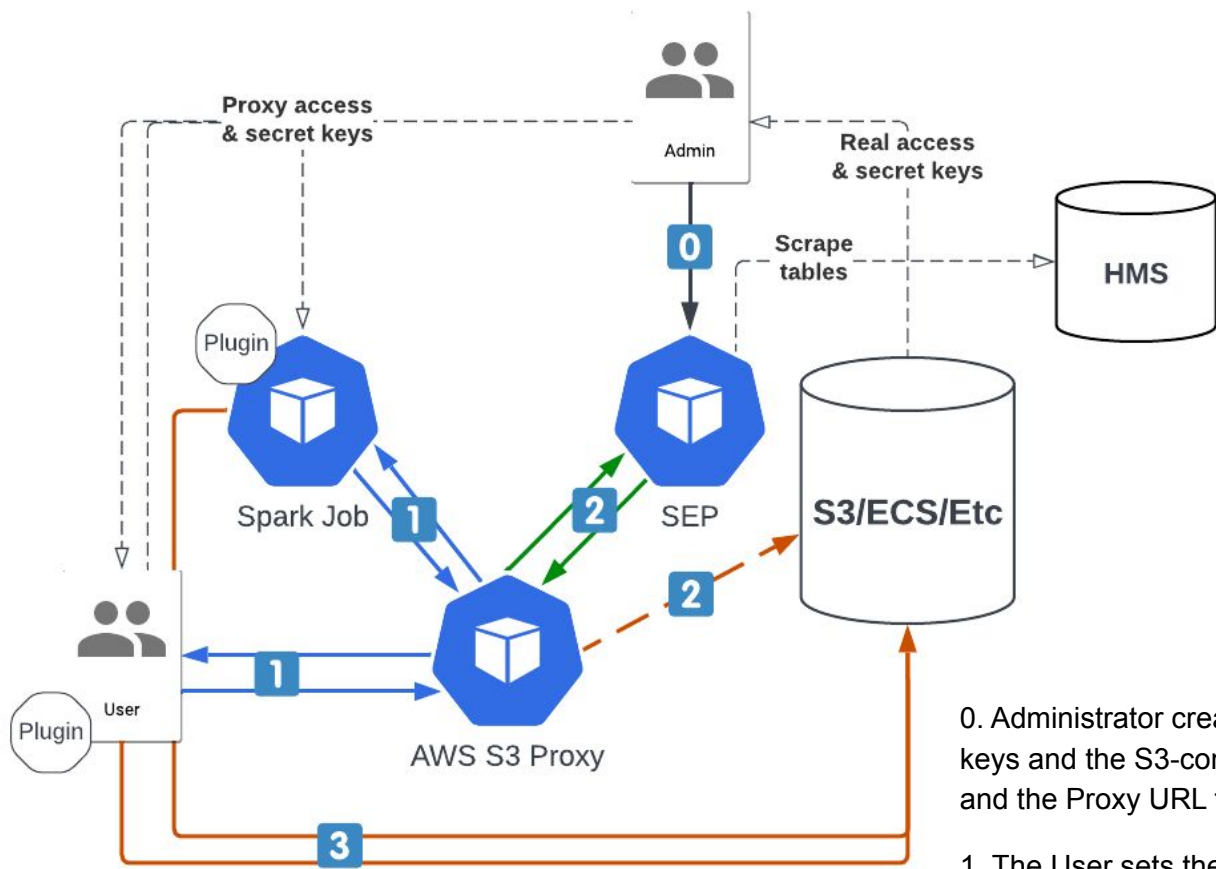
- ☒ User identity:
 - SSO integration
 - Credential scoping
 - User never deals with real S3 credentials
- ☒ How to enforce authz
 - OPA (or others!)
 - ☒ Can share the same policies as a Trino SELECT
- ☐ What Iceberg catalog?
 - Let's assume user provides a specific metadata filename

What Iceberg catalog should we use?

- Existing catalogs may not be easy to integrate:
 - e.g., a JDBC-based catalog doesn't have any easily-pluggable authentication
- Some tools just don't want to deal with another moving piece - they just want a metadata file name
- Say hello to redirection!
 - `proxy/my_catalog/my_schema/my_table/metadata.json`
 - Redirected to the latest metadata file for `my_catalog.my_schema.my_table`
 - Java-based API - make the logic as simple or as complex as you want
- **This is simply a convenience**
 - It is not intended to become a standalone Iceberg catalog
 - Use it in conjunction with *real* Iceberg catalogs like Apache Polaris

Self-service file sharing - it's not just Iceberg!





0. Administrator creates Proxy access and secret keys for a user in SEP by entering the real access and secret keys and the S3-compatible object store endpoint. The Administrator distributes the Proxy access and secret keys and the Proxy URL to the user.

1. The User sets the S3 client config endpoint to be the Proxy URL and access and secret config to be the Proxy access and secret keys. For Spark, Users set Spark config to use the Proxy URL and Proxy access and secret keys when submitting jobs or using Spark Connect. Whether via an S3 client or a Spark job, when an S3 object request is made (for upload, get, etc.) it goes first to the AWS Proxy.

2. The AWS Proxy sends the Proxy access key and secret and the bucket/key to SEP BIAC. SEP identifies the user and checks to see if the user has permission for the bucket/key. If possible, the bucket/key are translated into a table. If they have permission, the request is allowed. If it's a meta-request (i.e. not an object get or put) the AWS Proxy fulfills the request via the real object store. If the request is an object get/put, a pre-signed URL is returned to the client.

3. The client uses the pre-signed URL (via the plugin/custom S3 client) to make a direct call to the object store to access the real bucket/key.