

GNARLY
Data_Waves

PRESENTED BY  **dremio**

EPISODE 41







ZeroETL & Virtual Data Marts: The Cutting Edge of Lakehouse Architecture

Get Started with the Dremio Lakehouse Platform

- Sub-second query on 1 billion rows of data joining Amazon S3 with a Postgres database
- Connect to Tableau or Power BI and build a dashboard with this dataset
- Everything hosted by Dremio - 100% free for you
- Either deploy a self-managed cluster with Dremio Software or enjoy a Fully Managed service with Dremio Cloud


Self Managed

Deploy and run Dremio Community Edition as self-managed software on your cloud or on-premises infrastructure.

	Docker	Docs / Deploy
	Azure	Docs / Marketplace
	AWS	Docs / Marketplace
	Google Compute Platform	Docs
	Kubernetes	Docs
	YARN - RPM, Tar	Docs / RPM · Tar
	Standalone - RPM, Tar	Docs / RPM · Tar

Fully Managed


Start with Dremio in a fully-managed Cloud environment.



Dremio Cloud
Open & fully-managed data lakehouse

Best option if your data is on AWS.

[Sign Up For Free](#)



Dremio Test Drive
Experience Dremio with sample data

The simplest way to try out Dremio.

[Start Test Drive](#)

dremio.com/get-started



Apache Iceberg: The Definitive Guide

O'REILLY®

Apache Iceberg The Definitive Guide

Data Lakehouse Functionality, Performance,
and Scalability on the Data Lake





Survey Report

2024 State of the Data Lakehouse

Gain unprecedented insights into the evolving landscape of data lakehouses and benchmark your organization with Dremio's State of the Data Lakehouse Survey Report. This survey of 500 full-time IT and data professionals from large enterprises offers fresh insights on data lakehouse trends, adoption and associated benefits.

[Download Free](#)



GNARLY Data_Waves

PRESENTED BY  **dremio**

EPISODE 41

ZeroETL & Virtual Data Marts: The Cutting Edge of Lakehouse Architecture



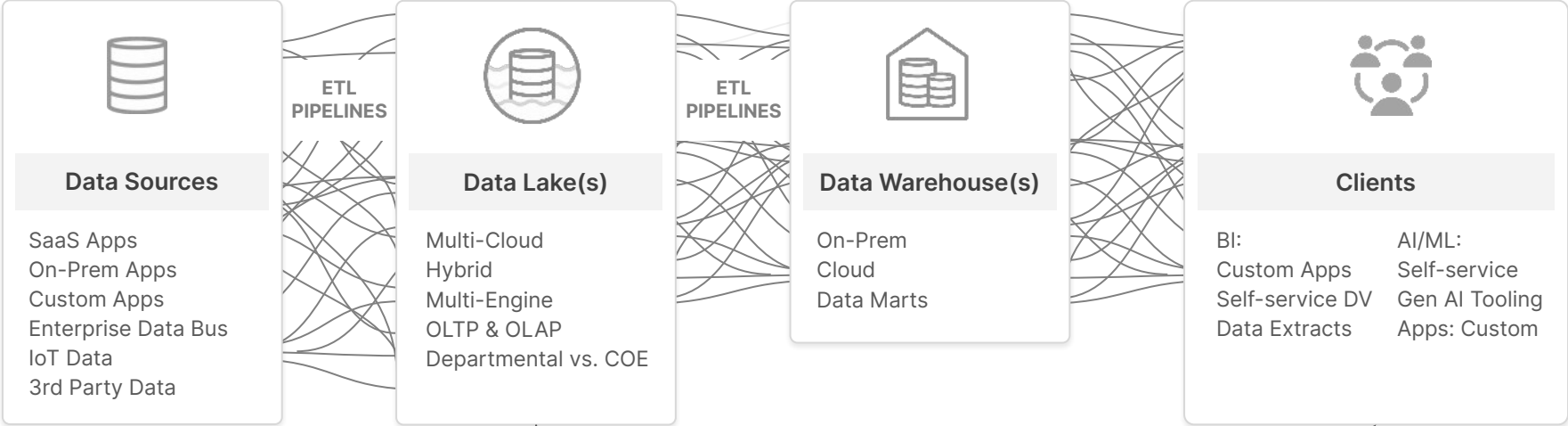
Alex Merced

Developer Advocate, Dremio

 December 5 at 8AM PST | 11AM EST | 4PM GMT

Part 1 - The Vision

Traditional Chain of Data Movement Pipelines



What's The Problem?

Broken Pipelines that require tedious backfilling





Angry Consumers from Late and Inconsistent Data

Cost of these Movements

1. Storage Costs
2. Compute/Processing Cost
3. Network & Egress Costs
4. Lost productivity in time it takes for all pipelines to proliferate data
5. Regulatory fees from governance and security risks in too many copies
6. Data Model Drift from as data models may become inconsistent over several movement
7. Cost of Bad Insights from Inconsistent Data from Data Copy Sprawl

Cost of Bad Insights from Inconsistent Data from Data Copy Sprawl

Storage Costs

Compute/Processing Cost

Network & Egress Costs

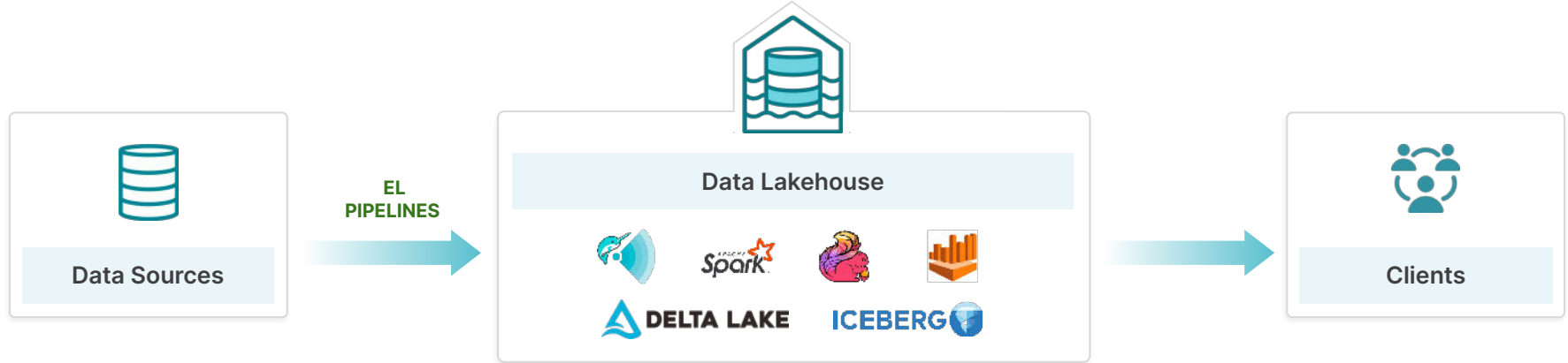
Lost productivity in time it
takes for all pipelines to
proliferate data

Regulatory fees from
governance and security
risks in too many copies

Data Model Drift from as
data models may become
inconsistent over several
movements

Cost of Bad Insights from Inconsistent Data from Data Copy Sprawl

The Dremio Approach (ZeroETL & Virtual Data Marts)



Federated Data Sources connected directly to Dremio (ZeroETL, Low-ETL)

Use Dremio's Semantic Layers to Model data for each business line into virtual data marts.

Dremio's Data Reflections eliminates the need for Cubes and Extracts

The Benefits of the Dremio Approach?

The Benefits of the Dremio Approach?

Less Pipelines,
Less Backfilling,
Fresher Data Sooner





Data Consumers
getting fresh,
correct and
fast data

Cost Benefits of this Approach

1. Less Copies, Less Storage Costs
2. Less Pipelines, Less Compute
3. Less Data Movement, Less Network Costs
4. Less Pipelines, All Datasets Remain Fresh
5. Less Copies in One Place, easier to comply with regulations
6. Less Copies, Less Chances of Data Models Changing
7. Fresher Data Means Better Insights

Less Copies,
Less Storage Costs

Less Data Movement,
Less Network Costs

Less Copies in One Place,
easier to comply with
regulations

Less Copies, Less Chances of Data Models Changing

Fresher Data Means Better Insights

How does Dremio make this possible at Scale?

High-Performance
Virtualization

Fast Query Engine Leveraging Apache Arrow and Apache Calcite

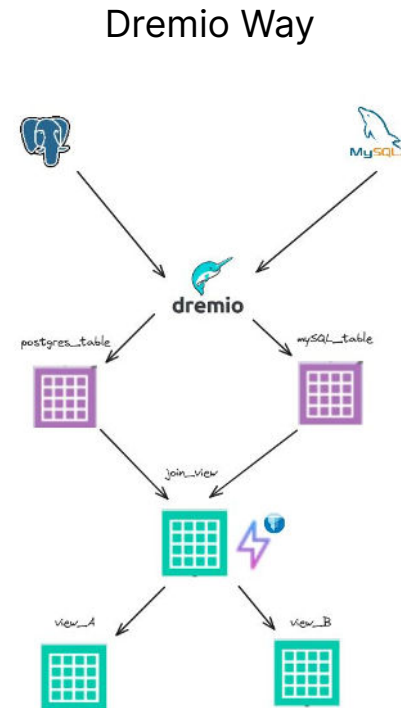
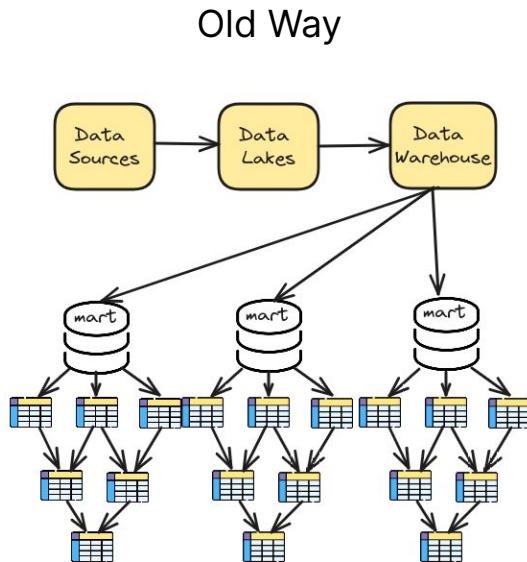


- Fast in-memory processing
- Pre-compiling of common routines into binary (Gandiva)
- Fast columnar data transfer (Flight)

- SQL Parsing and Validation
- Cost-Based Optimization
- Rule-based Optimization
- SQL Optimization

Raw Data Reflections to Enable Performant Virtualization

- Raw Reflections will create Apache Iceberg based relational caches
- Used to Accelerate Joins, Non-Data Lake Sources and more.
- Reflection refresh keeping cache fresh and performant
- Eliminates need to create materialized view and other structures that would require a manual pipeline to maintain.



Aggregate Reflections to Eliminate Extracts/Cubes

A better approach to caching pre-computed aggregations for accelerating BI dashboards. (Fast Dashboards made easy)

BI Extracts & Cubes

- **Static**
(Becomes Stale Quick)
- **Data Storage Maintenance**
(Storage Cost Creep without Manual Maintenance)
- **Manual Pipelines**
(To recreate and maintain)
- **Inflexible and Large**
(Only optimizes certain queries and can exceed size of available memory)

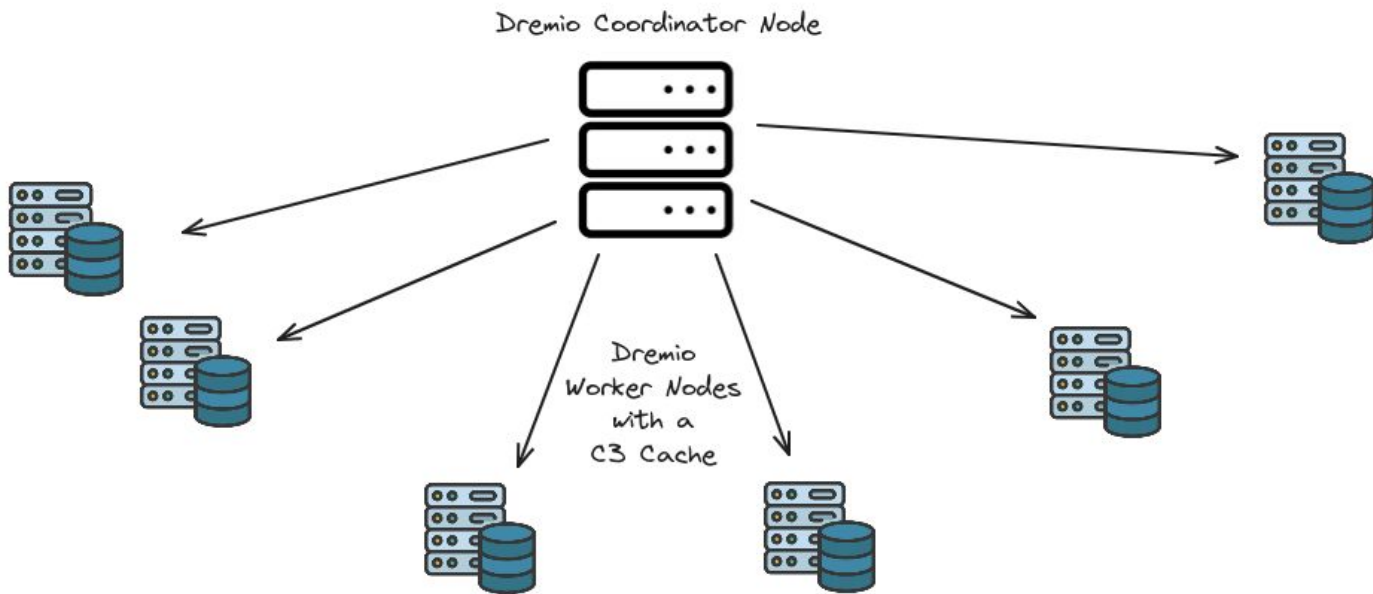
Dremio's Aggregate Reflections

- **Fresh**
(Can be updated incrementally)
- **Automatic Maintenance**
(Dremio will manage your Reflections Storage)
- **No Pipelines**
(Dremio handles updates)
- **Flexible**
(Structured in a way that is adaptable to more queries without memory challenges)

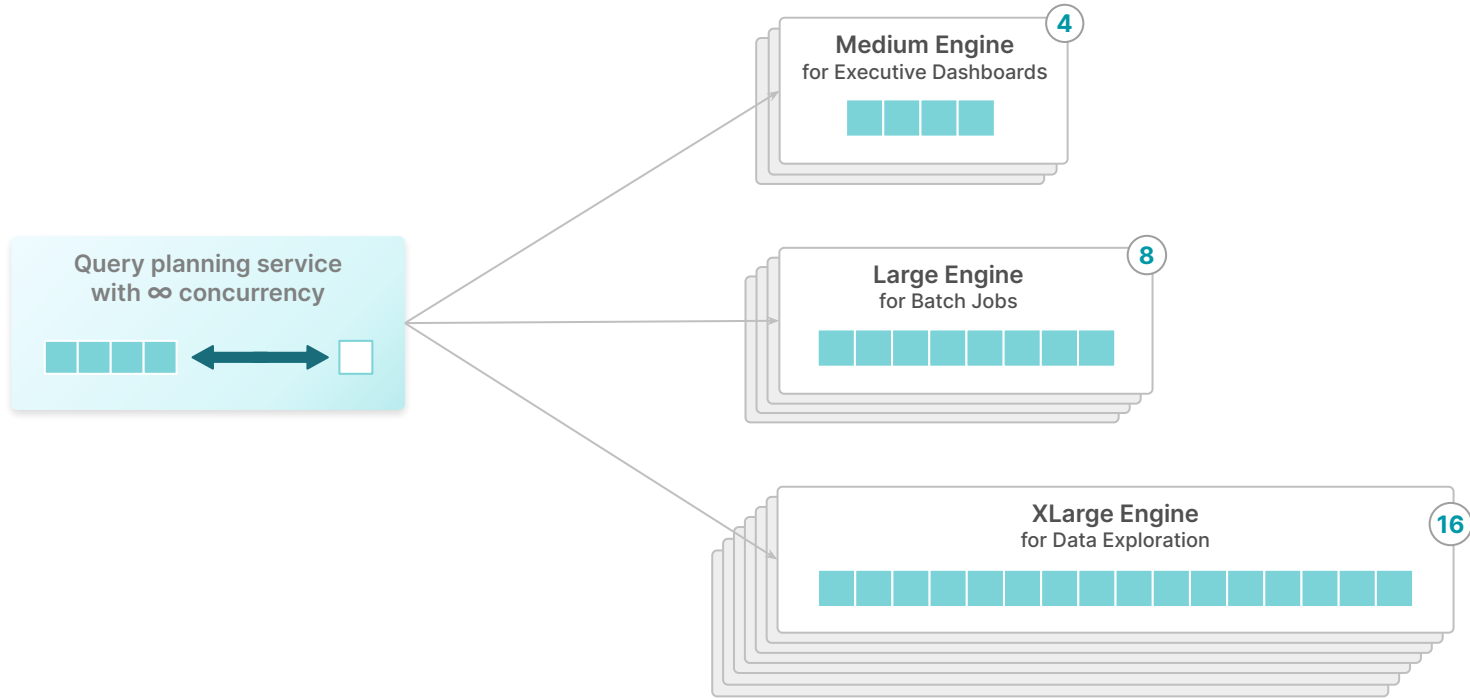
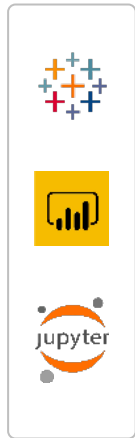
Columnar Cloud Cache (C3)

Boosting Performance and Reducing Network Calls

A NVMe Cache on each worker node holding frequently used data to more quickly access it and avoid network requests.



Unlimited Concurrency and Auto-Scaling



In Summary...

- Dremio eliminates the need for most data ingestion pipelines through virtualization paired with raw reflections (ZeroETL)
- Dremio eliminates pipelines for creating, updated and maintaining BI Extracts and cubes with aggregate reflections
- Dremio's Semantic Layer allows you to organize, govern and distribute your data to your different business lines using logical views forming "Virtual Data Marts"
- Dremio's high-performance query engine, Columnar Cloud Cache, and Data Reflections enables fast queries that cut down on the network, compute and storage costs often associated with query performance.
- Dremio unlimited concurrency and auto-scaling allows Dremio to scale to any workload without using more resources than necessary cutting costs even further.

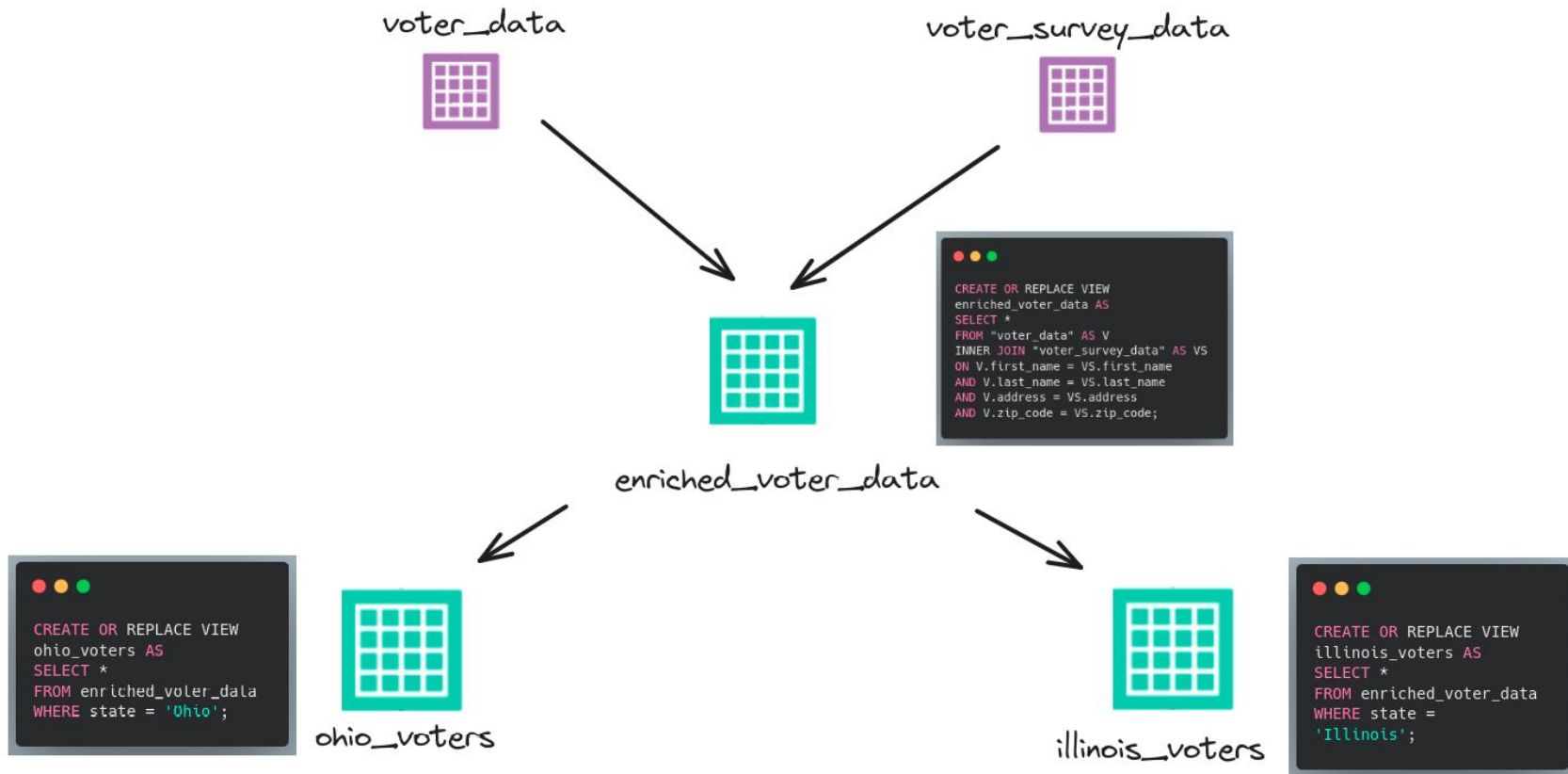
Less Pipelines. Less Costs. Easier Usage. Fresher Data.

How does Dremio make this possible at Scale?

1. Fast Query Engine Leveraging Apache Arrow and Apache Calcite
2. Raw Data Reflections to enable performant virtualization
3. Aggregate Reflections to eliminate Extracts/Cubes
4. Columnar Cloud Cache (C3) Boosting Performance and Reducing Network Calls
5. Unlimited Concurrency and Auto-Scaling

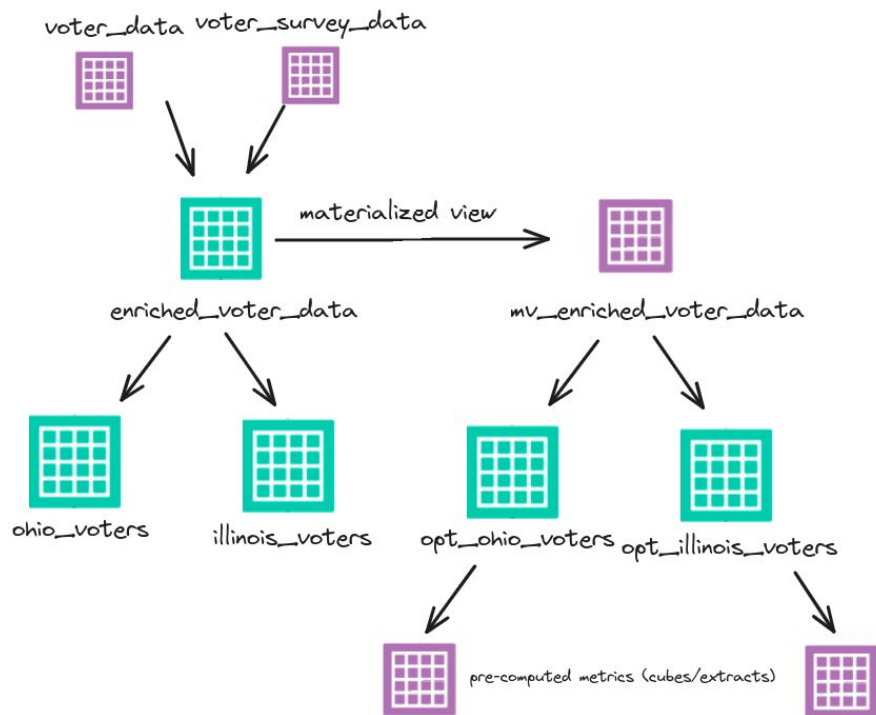
Understanding why Dremio is the “Intelligent Query Engine”

Modeling with Logical Views Only



Problem: queries on all views result in full table scans

Self-Curated Copies as Traditional Solution

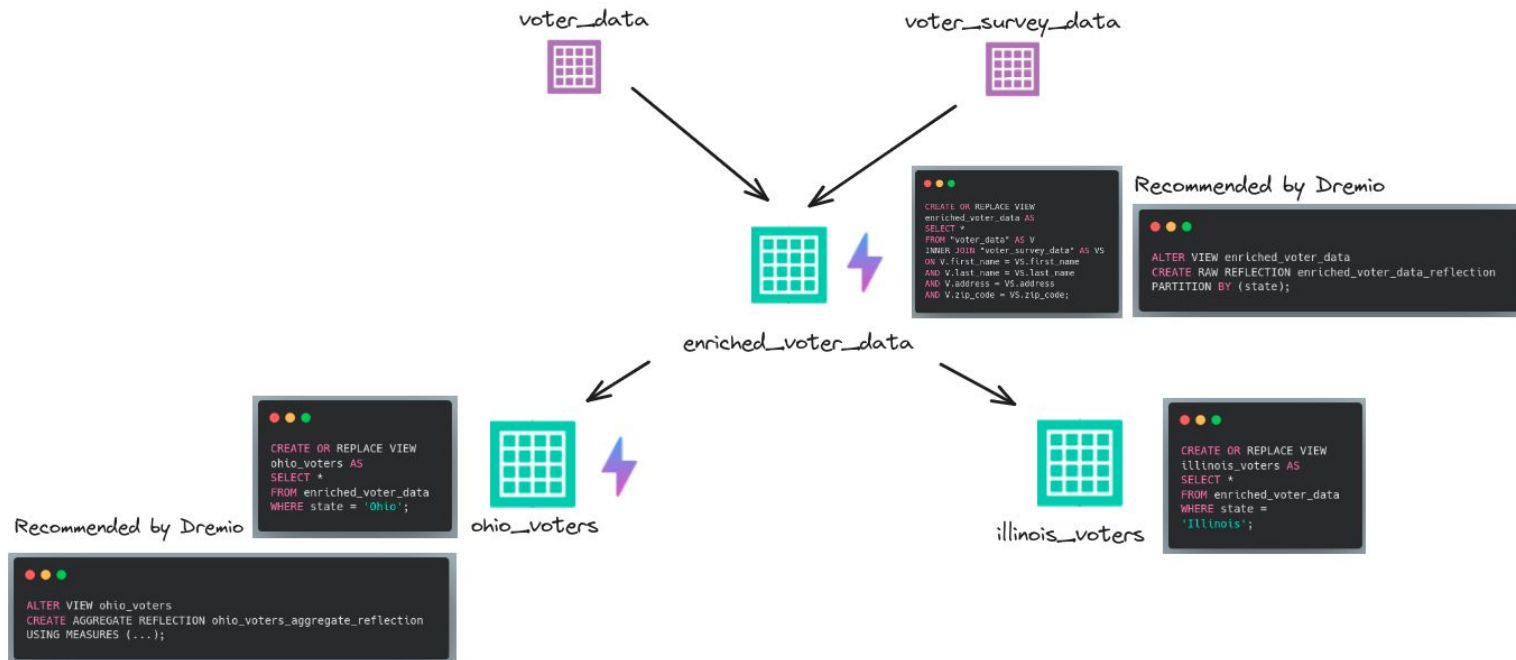


PROBLEMS:

- Don't auto-update
- Manual Maintenance
- Up to you to update queries
- Not intelligently used

Problem: queries on all views result in full table scans

Return to Logical Modeling with Dremio's Intelligent QUerying



Queries are fast and headaches are gone on logical views

Part 2 - Reference Example

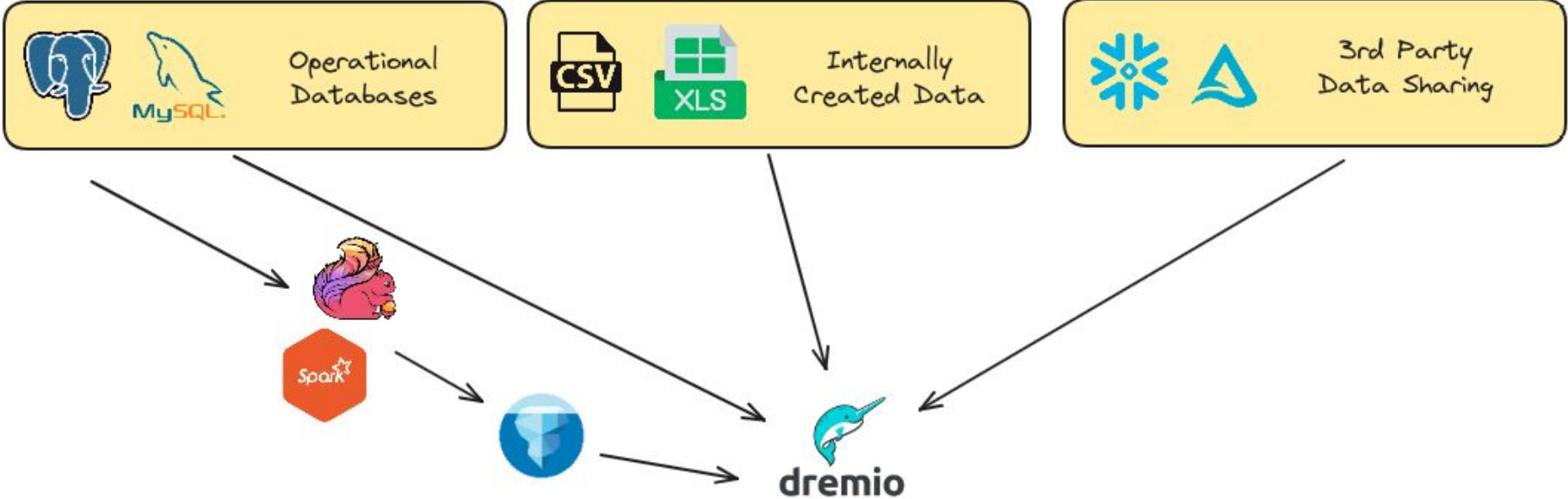
Refresh...

- Dremio eliminates the need for most data ingestion pipelines through virtualization paired with raw reflections (ZeroETL)
- Dremio eliminates pipelines for creating, updated and maintaining BI Extracts and cubes with aggregate reflections
- Dremio's Semantic Layer allows you to organize, govern and distribute your data to your different business lines using logical views forming "Virtual Data Marts"
- Dremio's high-performance query engine, Columnar Cloud Cache, and Data Reflections enables fast queries that cut down on the network, compute and storage costs often associated with query performance.
- Dremio unlimited concurrency and auto-scaling allows Dremio to scale to any workload without using more resources than necessary cutting costs even further.

Less Pipelines. Less Costs. Easier Usage. Fresher Data.

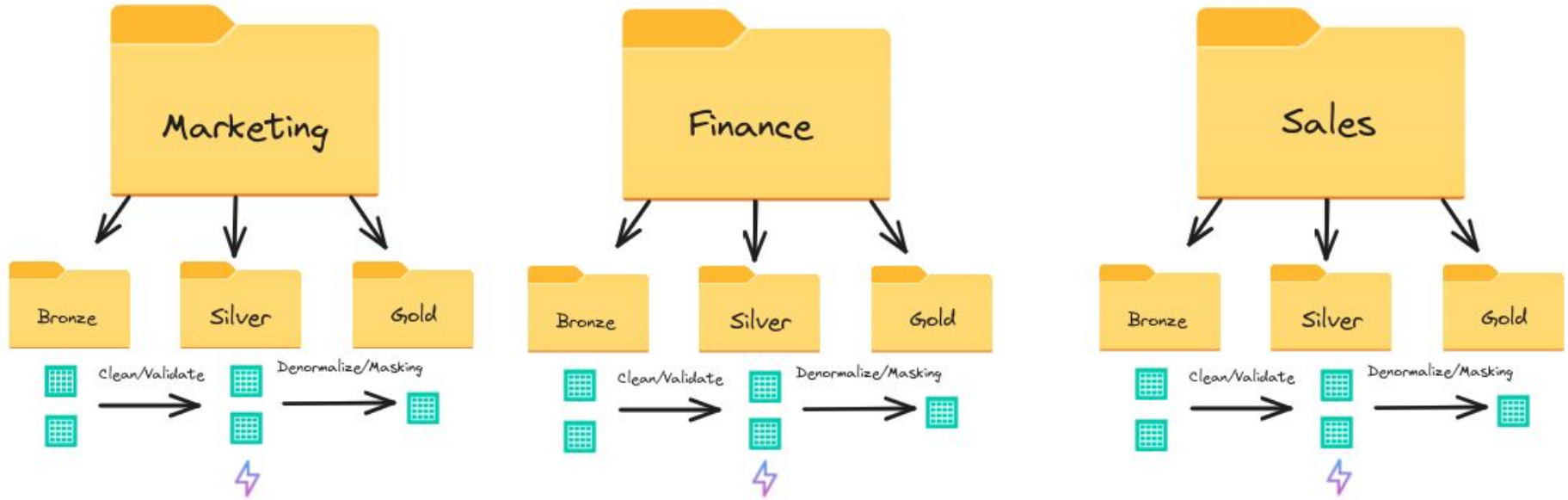
The Story Begins - Connect Your Data Sources

Low-ETL or ZeroETL



Data with strictest freshness requirements moved into Iceberg to take advantage of Incremental Reflection Updates

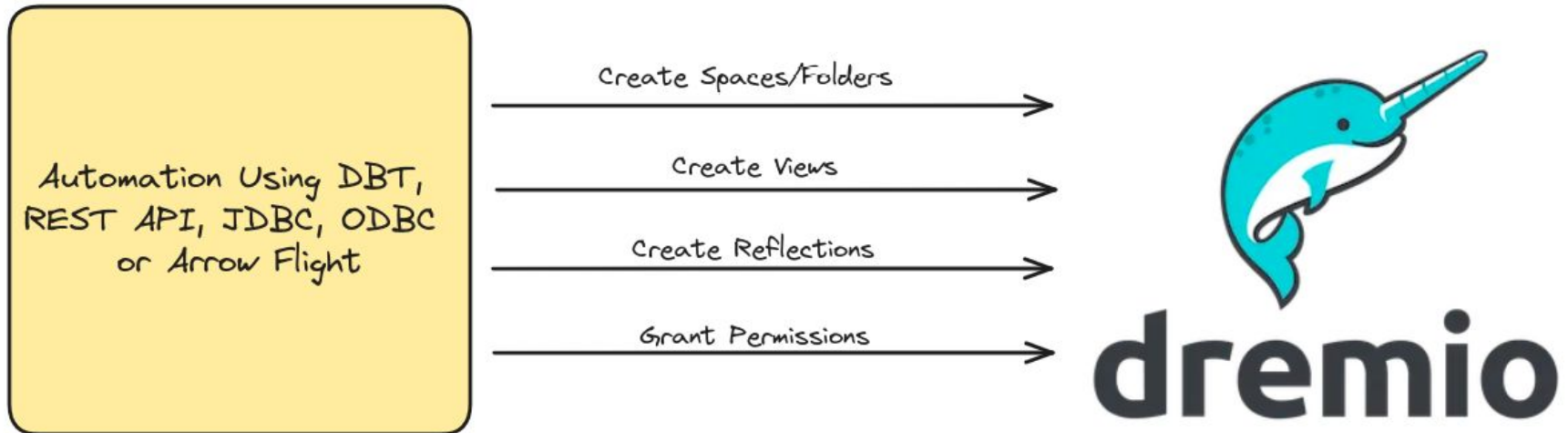
Model Your Lakehouse - Virtual Data Marts



Land unaltered views of the data in a bronze folder for each business line they can curate along with data the business line connects to create data products. (Data Mesh Approach)
Allow the reflection recommender to help identify views turn reflections on based on your query patterns.

Methods to Generate Your Virtual Data Marts

1. Manually with the **Dremio UI**
2. Automate with any language and the **Dremio REST API**
3. Automate with Dremio's **DBT integration**
4. Automate with SQL using **JDBC/ODBC/Arrow Flight**



Govern Your Data - Role Based Access Controls



```
GRANT SELECT ON TABLE TableA1 TO USER user1;

GRANT SELECT ON ALL DATASETS IN PROJECT project1 TO USER user1;

GRANT SELECT ON FOLDER Folder3 TO USER user1;

GRANT SELECT ON FOLDER Folder3 TO USER user1;

GRANT INSERT ON TABLE TableB1 TO USER user1;

GRANT DELETE ON ALL DATASETS IN PROJECT project2 TO ROLE Data_Deletion;

GRANT CREATE TABLE ON FOLDER Folder4 TO USER user2;

GRANT MANAGE GRANTS ON TABLE DatasetC1 TO ROLE Data_Admin;
```


Govern Your Data - Column Based Access Controls



-- Create a UDF for column masking

```
CREATE FUNCTION protect_ssn(ssn VARCHAR(11))
RETURNS VARCHAR(11)
RETURN SELECT CASE
    WHEN query_user()='jdoe@dremio.com'
    OR is_member('Accounting') THEN ssn
    ELSE CONCAT('XXX-XXX-', SUBSTR(ssn, 9, 3))
END;
```

-- Apply the column-masking policy to a column in a table or view

```
ALTER TABLE your_table_name
SET MASKING POLICY protect_ssn(your_column_name);
```

Govern Your Data - Row Based Access Control

```
  
-- Create a UDF for row-based access control  
CREATE FUNCTION country_filter(country VARCHAR)  
RETURNS BOOLEAN  
RETURN SELECT query_user()='jdoe@dremio.com' OR  
(is_member('Accounting') AND country = 'CA');  
  
-- Apply the row-based access control policy to a table  
ALTER TABLE your_table_name  
ADD ROW ACCESS POLICY country_filter(your_country_column);
```

Deliver Your Data - BI Dashboards



Aggregate
Reflection



Sub-Second Query Performance



BI Tools

HEX

Apache Superset

Looker

Deliver Your Data - Notebooks

ODBC/JDBC

```
#-----  
# IMPORTS  
#-----  
  
## Import pyodbc  
import pyodbc  
  
## Import pandas  
import pandas as pd  
  
## Import environment variables  
from os import environ  
  
#-----  
# SETUP  
#-----  
  
token=environ.get("token", "personal token not defined")  
connector="Driver={Dremio ODBC Driver 64-  
bit};ConnectionType=Direct;HOST=sql.dremio.cloud;PORT=443;Authenticatio  
nType=Plain;" + f"UID=$token;PWD={token};ssl=true;"  
  
#-----  
# CREATE CONNECTION AND CURSOR  
#-----  
  
# establish connection  
cnxn = pyodbc.connect(connector, autocommit=True)  
  
# set encoding  
cnxn.setdecoding(pyodbc.SQL_CHAR, encoding='utf-8')  
  
# creating a cursor to send messages through the connection  
cursor = cnxn.cursor()  
  
#-----  
# RUN QUERY  
#-----  
  
## run a query  
rows = cursor.execute("SELECT * FROM \\'@dremio.demo@gmail.com\\'.\\'nyc-  
taxi-data\\' limit 1000000").fetchall()  
  
##convert into pandas dataframe  
df = pd.DataFrame([tuple(t) for t in rows])  
|  
print(df)
```

PyArrow

```
#-----  
# IMPORTS  
#-----  
## Import Pyarrow  
from pyarrow import flight  
from pyarrow.flight import FlightClient  
  
## Import pandas  
import pandas as pd  
  
## Get environment variables  
from os import environ  
  
const token = environ.get('token', 'no personal token defined')  
  
#-----  
# Setup  
#-----  
  
## Headers for Authentication  
headers = [  
    (b'authorization', f'bearer {token}'.encode("utf-8"))  
    ]  
  
## Create Client  
client = FlightClient(location=(f"grpc+tls://data.dremio.cloud:443"))  
  
#-----  
# Function Definitions  
#-----  
  
## makeQuery function  
def make_query(query, client, headers):  
  
    ## Get Schema Description and build headers  
    flight_desc = flight.FlightDescriptor.for_command(query)  
    options = flight.FlightCallOptions(headers=headers)  
    schema = client.get_schema(flight_desc, options)  
  
    ## Get ticket to for query execution, used to get results  
    flight_info = client.get_flight_info(flight.FlightDescriptor.for_command(query), options)  
  
    ## Get Results  
    results = client.do_get(flight_info.endpoints[0].ticket, options)  
    return results  
  
#-----  
# Run Query  
#-----  
  
results = make_query("SELECT * FROM \\'@dremio.demo@gmail.com\\'.\\'nyc-taxi-data\\' limit  
1000000", client, headers)  
  
# convert to pandas dataframe  
df = results.read_pandas()  
  
print(df)
```

Deliver Your Data - Data Applications (Use Dremio's REST API)

Python FASTapi Application

```
from fastapi import FastAPI, HTTPException, Form, Request, Depends
from fastapi.templating import Jinja2Templates
import aiohttp
import json

app = FastAPI()
templates = Jinja2Templates(directory="templates")

# Replace with your Dremio API endpoint and personal access token
DREMIO_API_ENDPOINT = "https://apl.dremio.cloud/v0"
PERSONAL_ACCESS_TOKEN = "YOUR_PERSONAL_ACCESS_TOKEN"

async def run_dremio_sql_query(sql: str):
    async with aiohttp.ClientSession() as session:
        headers = {
            "Authorization": f"Bearer {PERSONAL_ACCESS_TOKEN}",
            "Content-Type": "application/json"
        }
        data = {
            "sql": sql
        }
        response = await session.post(f"{DREMIO_API_ENDPOINT}/projects/YOUR_PROJECT_ID/sql",
            headers=headers, json=data)
        return await response.json()

@app.get("/")
async def home(request: Request):
    return templates.TemplateResponse("index.html", {"request": request})

@app.post("/query/")
async def execute_query(request: Request, sql: str = Form(...)):
    try:
        result = await run_dremio_sql_query(sql)
        # Assuming the response contains data in a format that can be used by Chart.js
        # You may need to adapt this part to match your Dremio query result format
        labels = [row["label"] for row in result["data"]]
        values = [row["value"] for row in result["data"]]
        chart_data = {
            "labels": labels,
            "values": values,
        }
        return templates.TemplateResponse("chart.html", {"request": request, "chart_data":
            json.dumps(chart_data)})
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Jinja Templates

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Dremio Query App</title>
</head>
<body>
  <h1>Dremio Query App</h1>
  <form method="post" action="/query/">
    <label for="sql">Enter SQL Query:</label>
    <textarea id="sql" name="sql" rows="5" required></textarea>
    <br>
    <button type="submit">Run Query</button>
  </form>
</body>
</html>

<!-- chart.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Query Result Chart</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <h1>Query Result Chart</h1>
  <canvas id="myChart" width="400" height="400"></canvas>
  <script>
    const chartData = {{ chart_data | safe }};
    const ctx = document.getElementById('myChart').getContext('2d');
    new Chart(ctx, {
      type: 'bar',
      data: {
        labels: chartData.labels,
        datasets: [{
          label: 'Values',
          data: chartData.values,
          backgroundColor: 'rgba(75, 192, 192, 0.2)',
          borderColor: 'rgba(75, 192, 192, 1)',
          borderWidth: 1
        }]
      },
      options: {
        scales: {
          y: {
            beginAtZero: true
          }
        }
      }
    });
  </script>
</body>
</html>
```