INARI™

Data Evolution at Inari: Harnessing Delta Live Tables & Unity Catalog

Kishore Sundar – Sr. Manager, Data Engineering
Jonathon Long – Staff Data Engineer

Speakers

Kishore Sundar

Sr. Manager, Data Engineering @ Inari

Jonathon Long

Staff Data Engineer @ Inari

INARI

We are the SEEDesign™ company.

We make seeds that address the world's needs, pushing the boundaries of what is possible for a more sustainable, nature-positive food system.

A combination of AI-powered predictive design and a pioneered multiplex gene editing toolbox is enabling us to unlock the full potential of seed.

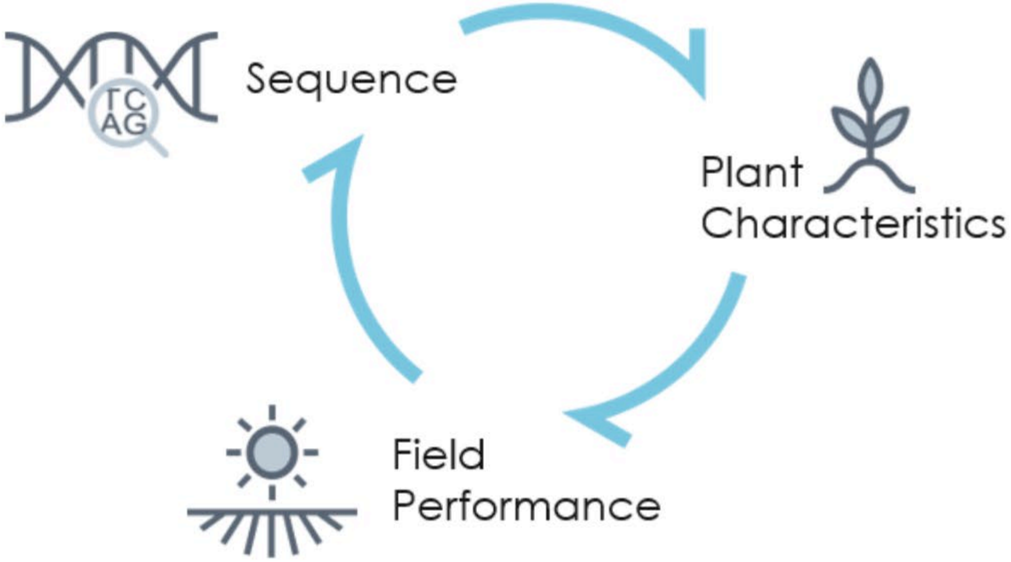
Our step-change products lead to more productive acres and a more sustainable future benefiting the population, the planet and the people who grow our food.

INARI™

Cutting-Edge Technology Platform

Predictive Design

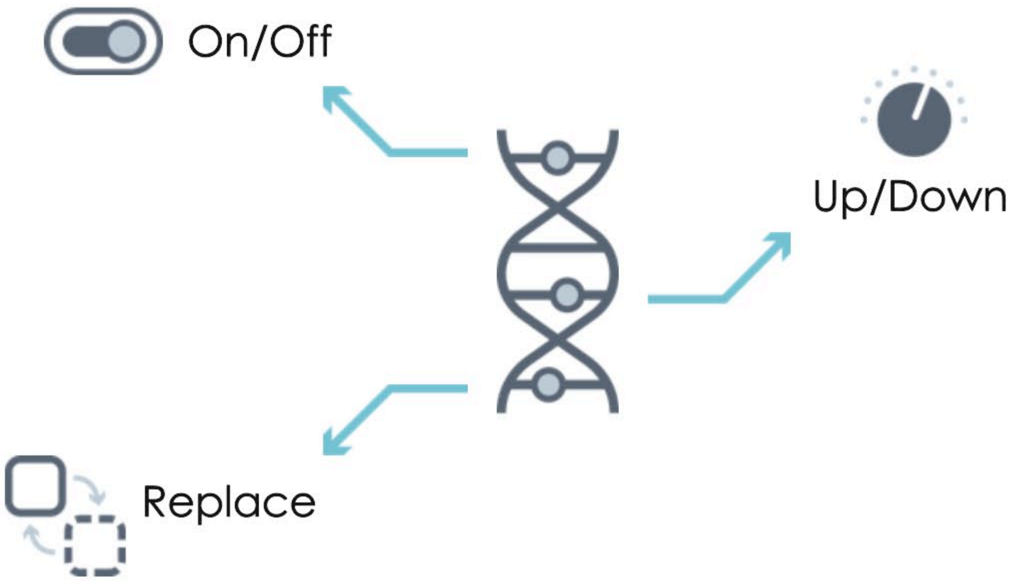
DEEP UNDERSTANDING OF NATURE'S COMPLEXITY



The Blueprint

Advanced Multiplex Gene Editing

MULTIPLE CHANGES AT THE SAME TIME



The Inari Toolbox

Inari At a Glance

FOUNDING
2016 (by Flagship Pioneering)

EQUITY CAPITAL RAISED
$575+ million to date

EMPLOYEES
>300 FTEs, with backgrounds across plant & human biology, physics, crop & data science, software, etc.

PATENTS
>125 patents filed and **~2,400 patented traits** filed

LOCATIONS

Cambridge, Massachusetts
Headquarters & Science

West Lafayette, Indiana
Product & Commercial

Ghent, Belgium
Research & Development

DRIVEN BY A DIVERSITY OF EXPERIENCE

OUTSIDE AG

Deloitte.
McKinsey & Company

AGRICULTURE

PIONEER.
CORTEVA agriscience
BASF
MONSANTO
syngenta
BAYER

ACADEMIA

HARVARD UNIVERSITY
MIT
UCLA
Cornell University
PURDUE UNIVERSITY.
Innovative Genomics Institute
CSH Cold Spring Harbor Laboratory

Introduction

WHAT TO EXPECT

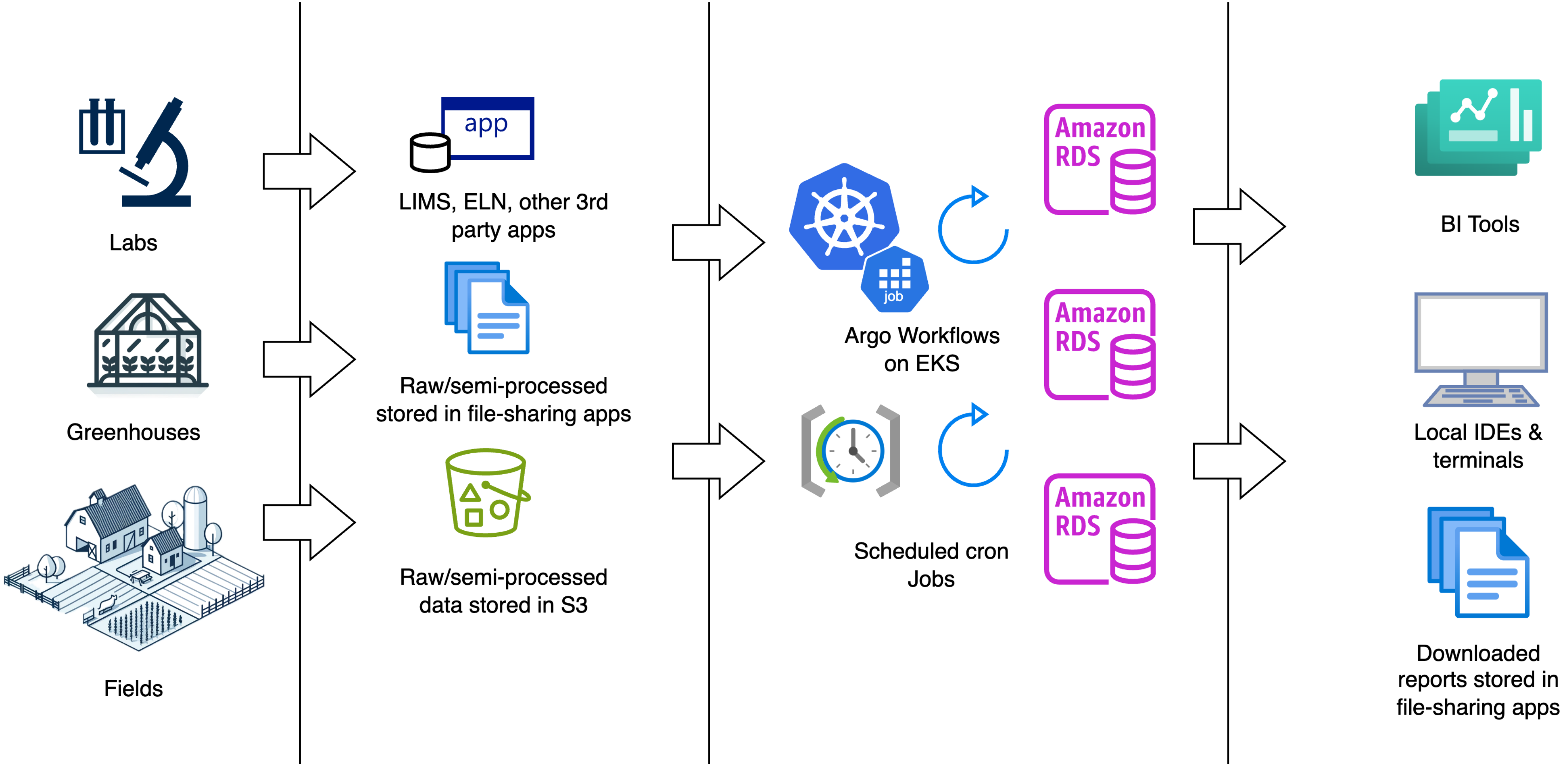
What is our talk about?

- How the Data Engineering team at Inari onboarded their first project onto Databricks
- How we developed a strategy that would guide future data engineering projects at Inari, using Unity Catalog and Delta Live Tables.

Who is it for?

- Data Engineering
- Data Management
- Data Governance

Early Data Landscape (2022-23)



INARI

Opportunity

- Major LIMS migration project in mid-2023 gave us the opportunity to design a Databricks-centric solution.
- Existing data views were created with complex queries(~5 hours execution time) running as CRON jobs within SQL databases attached to the existing LIMS software.
- These data views were critical to key decisions made in our entire product pipeline.

INARI™

Opportunity

SPECS & REQUIREMENTS

Source data:

- 25+ tables, between 10-50 columns in each table, between 500k – 20M records each and always growing

Expected outcome:

- 15 tables, each being a product of joining and transforming several source tables and meant to answer specific operational or scientific questions.
- Must follow FAIR data principles
- Centralized governance and sharing
- Ensure data quality and freshness

INARI™

Unity Catalog Approach

INARI™

Why Unity Catalog?

- Unified governance and management solution for all data assets
- Allows sharing, governing, and managing data across all workspaces and external applications using SQL warehouses and service principals.
- Auditing and lineage capabilities
- Enables FAIR data.
- Became clear that features & improvements to Databricks would be built around Unity Catalog

Unity Catalog

MEDALLION ARCHITECTURE

Bronze

- Raw data ingested from a system of record.
- Pulled incrementally stored as delta tables/materialized views
- Can use DLT autoloader where applicable

Silver

- Enriched datasets
- Data from a single source joined, pivoted, or aggregated to create materialized views.
- Used for reporting, decision making.

Gold

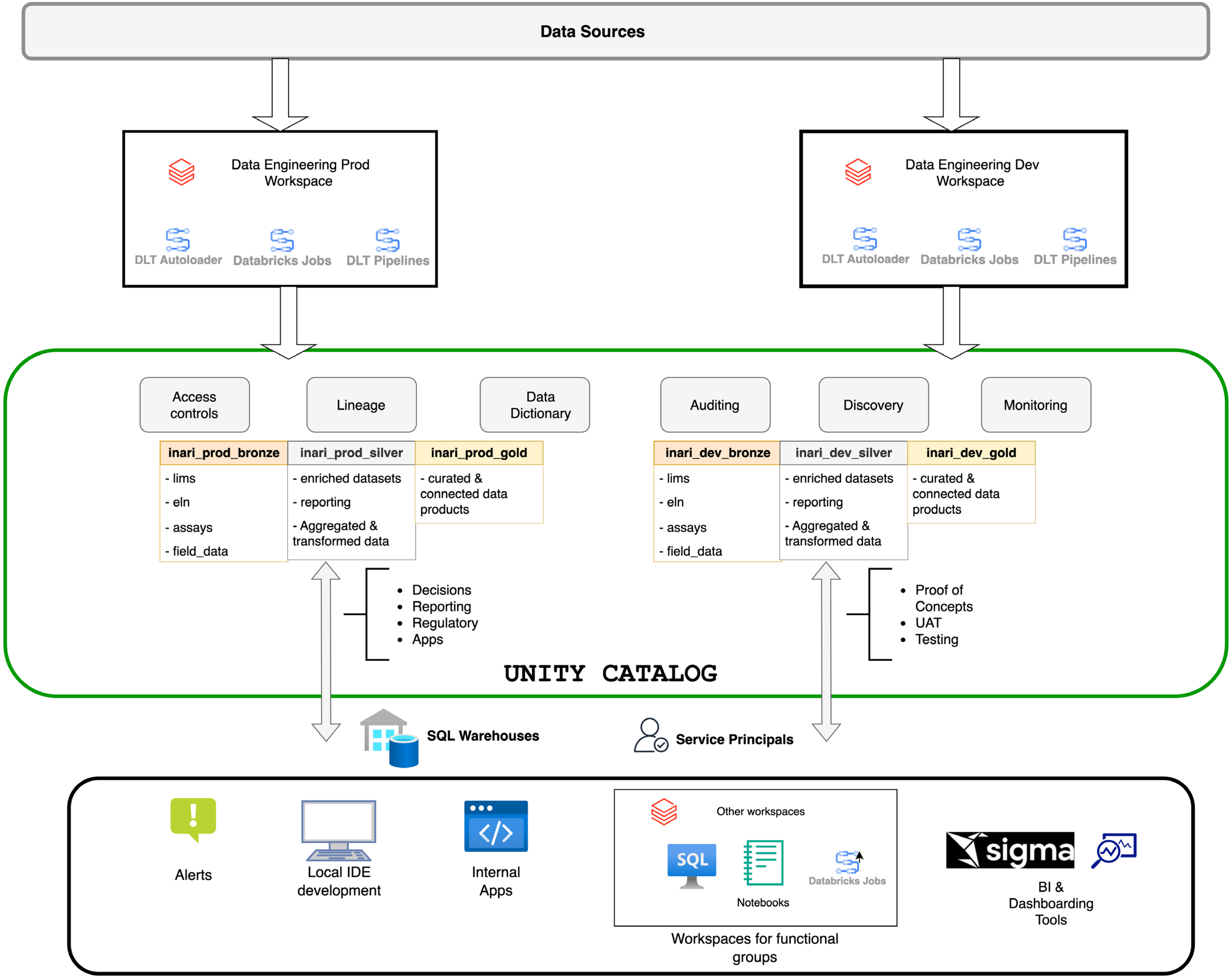
- One schema for each data product, which would consist of several tables made up of curated datasets
- Data from multiple bronze/silver data sources joined together to create materialized views.
- Used for reporting, analytics, and decision making.

Unity Catalog

MANAGEMENT & SHARING

- Each data team at Inari operates in a different workspace and has groups associated with them
- Enabled data sharing at any level of granularity across multiple workspaces and groups.
- Service principals for access to external systems where applicable.
- Highly scalable SQL warehouses to query the data in any way possible

INARI™



Data Sources
Data Engineering Prod Workspace
DLT Autoloader
Databricks Jobs
DLT Pipelines
Data Engineering Dev Workspace
DLT Autoloader
Databricks Jobs
DLT Pipelines
Access controls
Lineage
Data Dictionary
Auditing
Discovery
Monitoring
inari_prod_bronze
- lims
- eln
- assays
- field_data
inari_prod_silver
- enriched datasets
- reporting
- Aggregated & transformed data
inari_prod_gold
- curated & connected data products
inari_dev_bronze
- lims
- eln
- assays
- field_data
inari_dev_silver
- enriched datasets
- reporting
- Aggregated & transformed data
inari_dev_gold
- curated & connected data products
• Decisions
• Reporting
• Regulatory
• Apps
• Proof of Concepts
• UAT
• Testing
UNITY CATALOG
SQL Warehouses
Service Principals
Alerts
Local IDE development
Internal Apps
Other workspaces
SQL
Notebooks
Databricks Jobs
Workspaces for functional groups
sigma
BI & Dashboarding Tools

INARI

Unity Catalog

CHALLENGES WE FACED

- Not all clusters defined within other workspaces were UC enabled
- Single user compute clusters cannot access materialized views and streaming tables created by DLT
- R is not supported in shared access mode
- Enforcing granular access control and governance on BI/dashboarding tools where service principals were used

INARI™

Delta Live Tables Approach

INARI™

Why Delta Live Tables?

Requirements

- Three month deadline
- 25 data source tables
- 15 aggregated data product tables

Considerations

- Small team
- New data models
- Company’s first Data Engineering project in Databricks
- Strict deadline

INARI™

Why Delta Live Tables?

REDUCED COMPLEXITY FOR DEVELOPERS

All you need are DataFrames

- Given deadline, focus was entirely on learning new models and building out tables
- Everything in DLT is a Spark DataFrame
- Only concerns are data ingestion and transformation

No manual orchestration

- DLT handles dependencies between views and tables for you
- Creates a DAG for data loading
- Also allows for concurrent loading of data between non-dependent tables

No manual maintenance

- DLT manages maintenance of tables behind the scenes
- Tables are auto-vacuumed and optimized as part of the process
- No additional work required by developers to maintain tables

Handling Complexity

CODE BREAKDOWN

Common

- Views of the data that were shared across multiple data sources
- Created once, used many times
- After initial deployment – candidates for temporary table creation for improved performance

Intermediate

- Views of the data specific to enriched outputs
- Could draw on common views or direct from staging tables
- Where most of the business logic lives

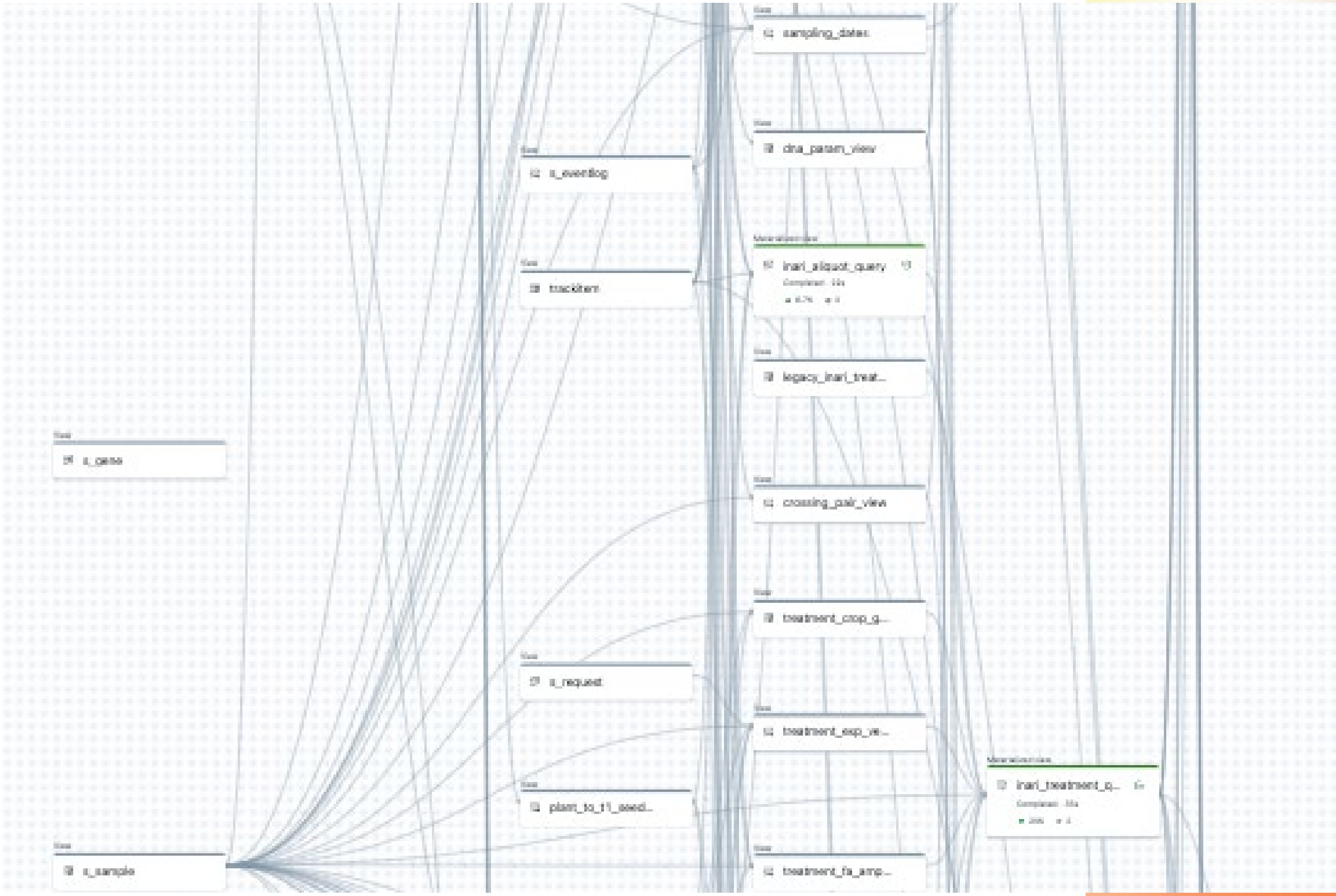
Enriched

- Final output tables
- Left joins on our intermediate views on primary key
- Little additional processing beyond the intermediate views

Handling Complexity

SO MANY VIEWS!

- Over 100 views feeding our materialized views
- Lineage tracked in Unity Catalog
- Creation of these views within the pipeline completely managed by DLT



INARI

Programmatic Views

DRY DATAFRAME CREATION

- DLT allows for creating views programmatically
- Define a function that creates different views based on passed arguments
- Reduces code re-use, likelihood of errors

```python
import dlt

from lims_delta_pipeline.staging.util import C_LIMS_TABLES


def create_view(spark, name, catalog, schema):
    @dlt.view(name=name)
    def t():
        return spark.table(f"{catalog}.{schema}.{name}")


def create_staging_views(spark, catalog, schema):
    for t in C_LIMS_TABLES:
        create_view(spark, t, catalog, schema)
```

INARI

Programmatic Views

- Internal data lineage managed within a single table
- Over 20 parent to child sample relationships
- Need views for many of these relationships for our outputs
- Creating a new view – adding a tuple to a list

```python
def create_lineage_view(parent_sampletype: str, child_sampletype: str):
    parent = parent_sampletype.lower().replace(" ", "_")
    child = child_sampletype.lower().replace(" ", "_")

    name = f"{parent}_to_{child}"

    parent_id = f"{parent}_sampleid"
    child_id = f"{child}_sampleid"

    @dlt.view(name=name)
    def t():
        df = (
            dlt.read("all_lineage")
            .filter(F.col("parent_sampletype") == parent_sampletype)
            .filter(F.col("child_sampletype") == child_sampletype)
            .withColumnRenamed("sampleid", parent_id)
            .withColumnRenamed("childsampleid", child_id)
        )
        return df


def create_lineage_views():
    """Creates DLT views for all lineages in SAMPLE_LINEAGES."""
    for sl in SAMPLE_LINEAGES:
        create_lineage_view(sl[0], sl[1])
```

INARI

Testing Transformation Logic

MAKING SURE THE DATA IS RIGHT

- One of our use cases – grouping many rows into a single JSON string
- Keys and values for JSON needed transformation
- Deduping to take latest
- Extracting transformation code to functions allows for local testing (pytest, unittest)

- Can't test this locally

```
@dlt.view()
def plant_samples_jsonplant():
    df = dlt.read("plant_plantsamples_dna_view")
    df = create_jsonplant_column(df, "sampleid")
    return df
```

- Can test this!

```
def create_jsonplant_column(
    df: DataFrame,
    groupbycol: str
) -> DataFrame:
```

Developer Experience Improvements

Typing Stubs

GETTING AUTO-COMPLETE IN YOUR IDE!

- As of January 2024, Databricks provides a PyPi package **databricks-dlt**
- **Use this package!**
- Prior to this, could not get code hints, code completion, etc from IDE without own typing stub

typings > dlt.pyi > ...

```python
from pyspark.sql import DataFrame
from pyspark.sql.types import StructType

from typing import Union

def read(tbl_name: str) -> DataFrame: ...
def table(
    name: str,
    comment: str,
    spark_conf: dict,
    table_properties: dict,
    path: str,
    partition_cols: list,
    schema: Union[str, StructType],
    temporary: bool,
) -> None: ...
def view(name: str, comment: str) -> None: ...
```

Development on Clusters

FAKING DLT FUNCTIONALITY... AT YOUR OWN RISK

```
class FauxDLT:
    def __init__(self, catalog="staging", schema="lims"):
        spark.sql(f"USE CATALOG {catalog}")
        spark.sql(f"USE SCHEMA {schema}")

    def read(self, name):
        return spark.read.table(name)

    def view(self, name=None, comment=None):
        def wrapper(func):
            @functools.wraps(func)
            def create_view(*args, **kwargs):
                df = func(*args, **kwargs)
                if name is not None:
                    temp_name = name
                else:
                    temp_name = func.__name__
                df.createOrReplaceTempView(f"{temp_name}")
                return df

            return create_view

        return wrapper
```

- Cannot run DLT on an All-Purpose cluster, requires running pipeline
- This can slow down development
- Can mimic functionality with a faux DLT class
- This is only for ad-hoc development!

VSCode Tasks and Databricks CLI

HOW TO RUN YOUR DEVELOPMENT PIPELINE IN TWO COMMANDS

- Process to update Databricks Repo and re-run pipeline via GUI can be tedious
- Same commands exist in Databricks CLI
- Use the Databricks CLI and VSCode tasks to make it smoother!

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "label": "Update my DBX Repo",
      "type": "shell",
      "command": "databricks repos update /Repos/jlong@inari.com/lims-delta-pipeline --branch $(git rev-parse --abbrev-ref HEAD)",
      "group": "build",
      "presentation": {
        "reveal": "never",
        "panel": "shared"
      }
    },
    {
      "label": "Run my DLT pipeline",
      "type": "shell",
      "command": "databricks pipelines start-update 01234567-89ab-cdef-0123-456789abcdef -p dev",
      "group": "build",
      "presentation": {
        "reveal": "never",
        "panel": "dedicated"
      }
    },
  ]
}
```

INARI

Results

INARI™

Success!

Successful project completion

- Able to complete the project in the 3-month timeframe without any interruption to our product pipeline

FAIR data

- Simplified, yet governed data sharing
- Ability to join and query across data products seamlessly

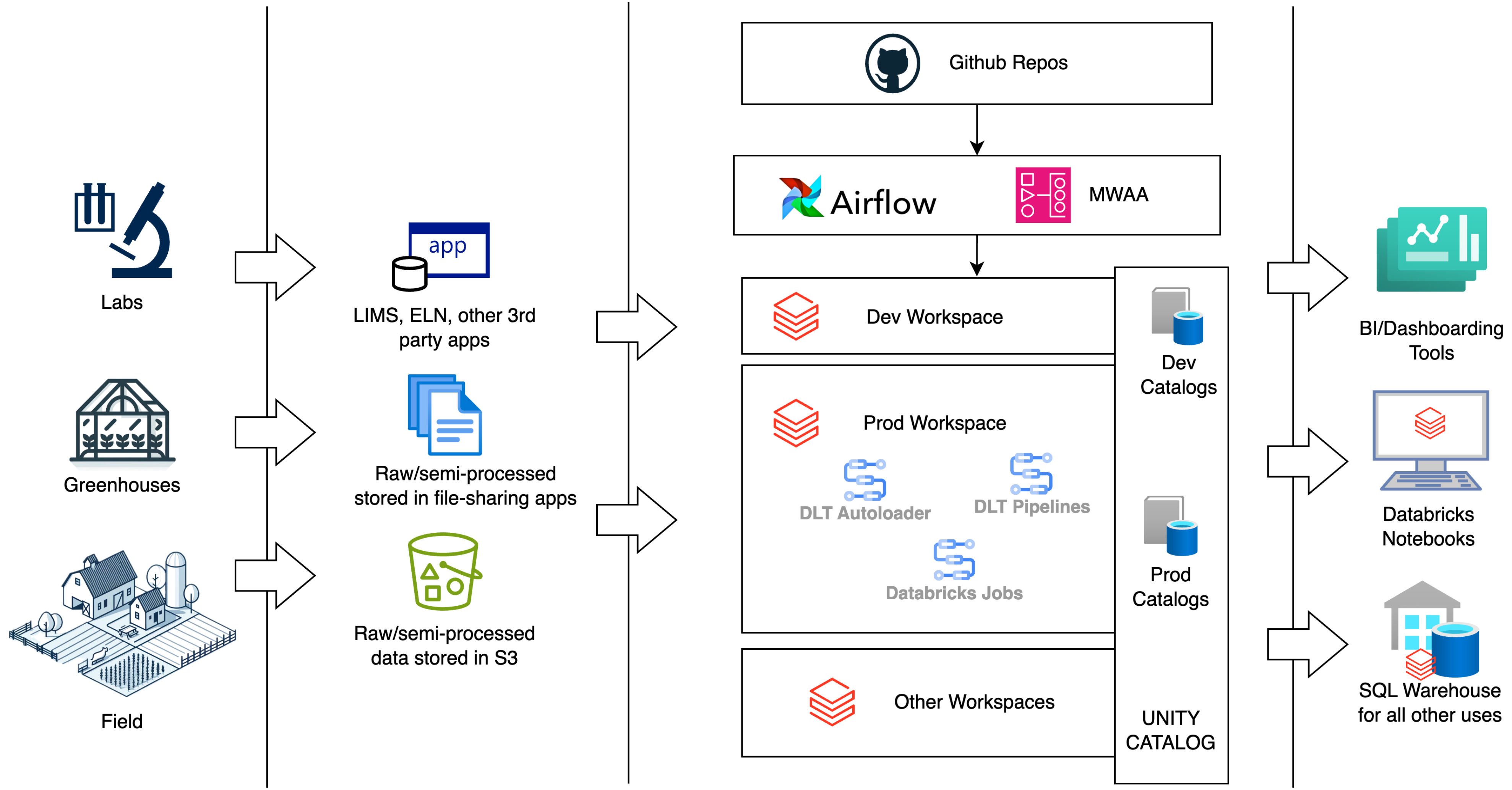
Strong foundations and better performance

- Greatly improved processing times backed by Apache Spark means fresh data available for making decisions
- Ability to answer greater breadth of research & development questions through ingesting different data products into Unity

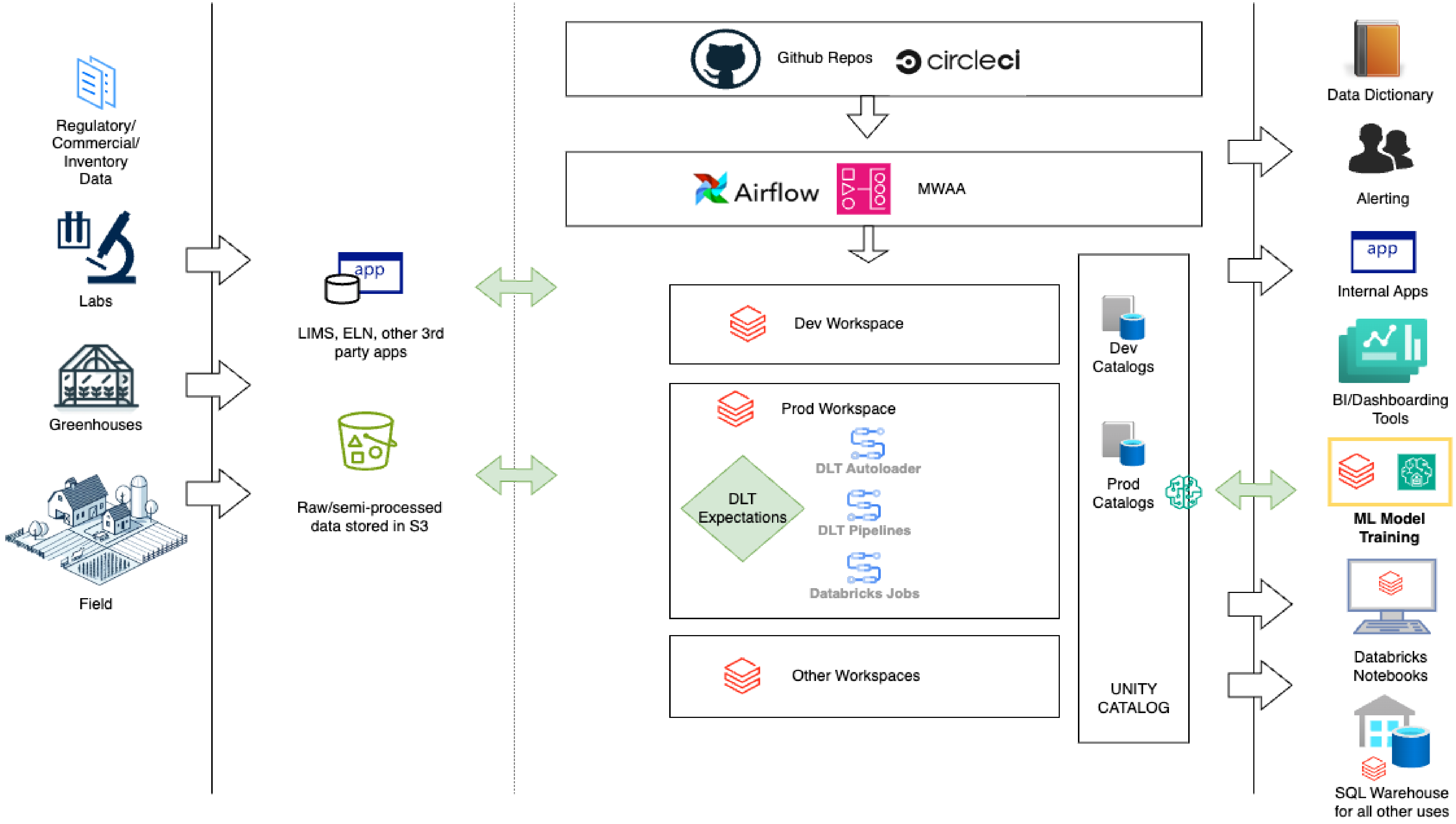
Improved Development Practices

- Code readability greatly increased
- Testing for complex aggregations
- Simple and efficient deployment process

Current Data Landscape



What the future holds…



INARI

Summary

- Unity Catalog helped us break down data siloes and connect data between data products which was not easy to do
- Unity Catalog puts us on a path of unified governance, allows us to track lineage, and monitor data quality
- Delta Live Tables works seamlessly with Unity Catalog and allows making changes rapidly and deploy into production
- Data processing times are down to a handful of minutes without any significant optimization efforts, compared to several hours previously
- This approach has allowed us to reuse code, configuration, and CI/CD
- Cluster start times remain a bottleneck, and we are exploring serverless or cluster pooling

Thank You

INARI™