OSA CON 25

Scaling Data Pipelines @Magenta Telekom

Georg Heiler
Senior Data Expert
geoheil.com

November 4-5, 2025

Georg Heiler

Solving challenges with data.

- Senior data expert @Magenta
- Research Software Engineer @ASCII & CSH
- Meetup organizer & frequent Speaker

geoheil.com, linkedin.com/in/geoheil

There is a chaos out there

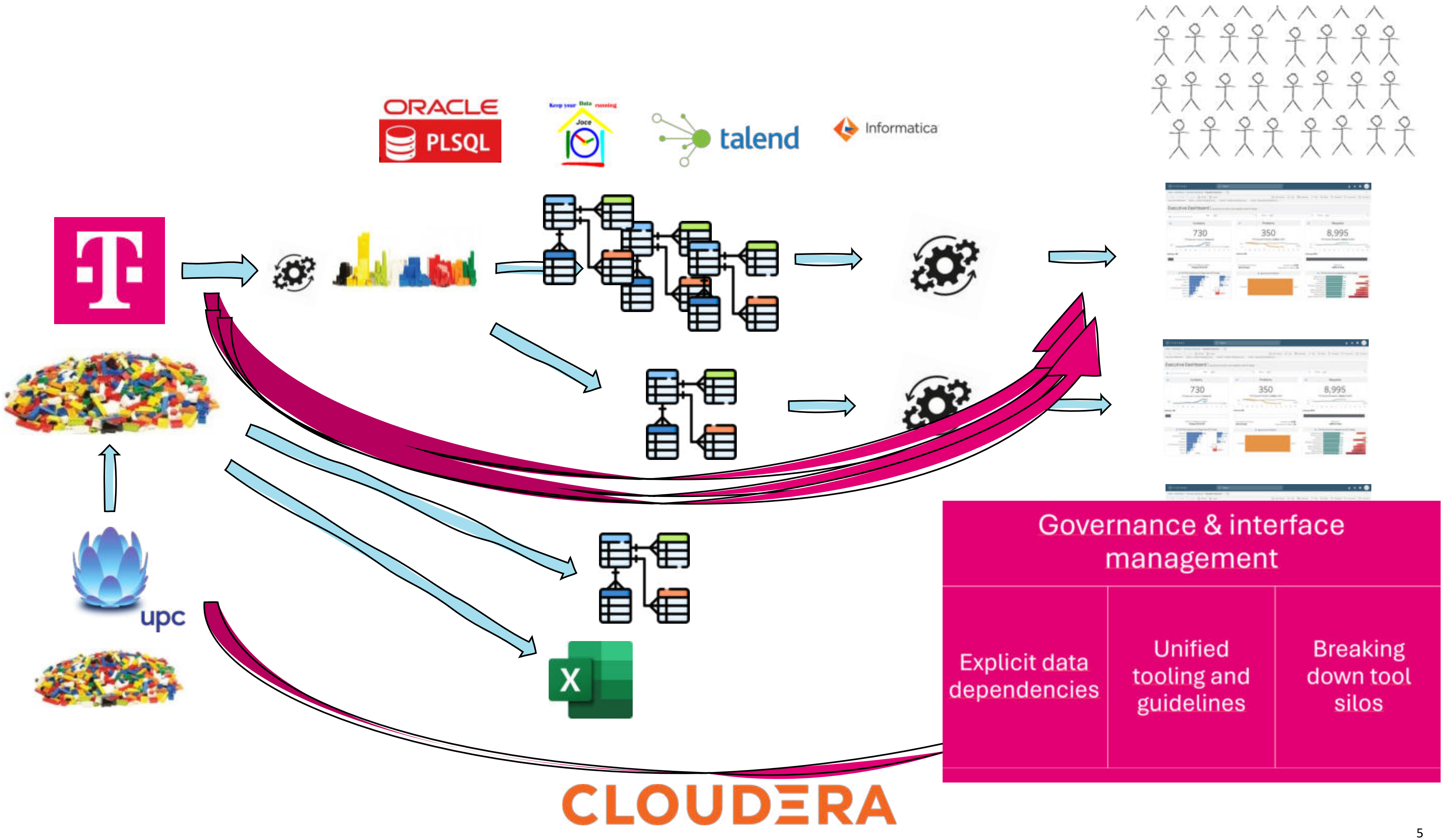
How did we end here? Time!

business grows (merger)
demand for data grows
methodology and tooling changes

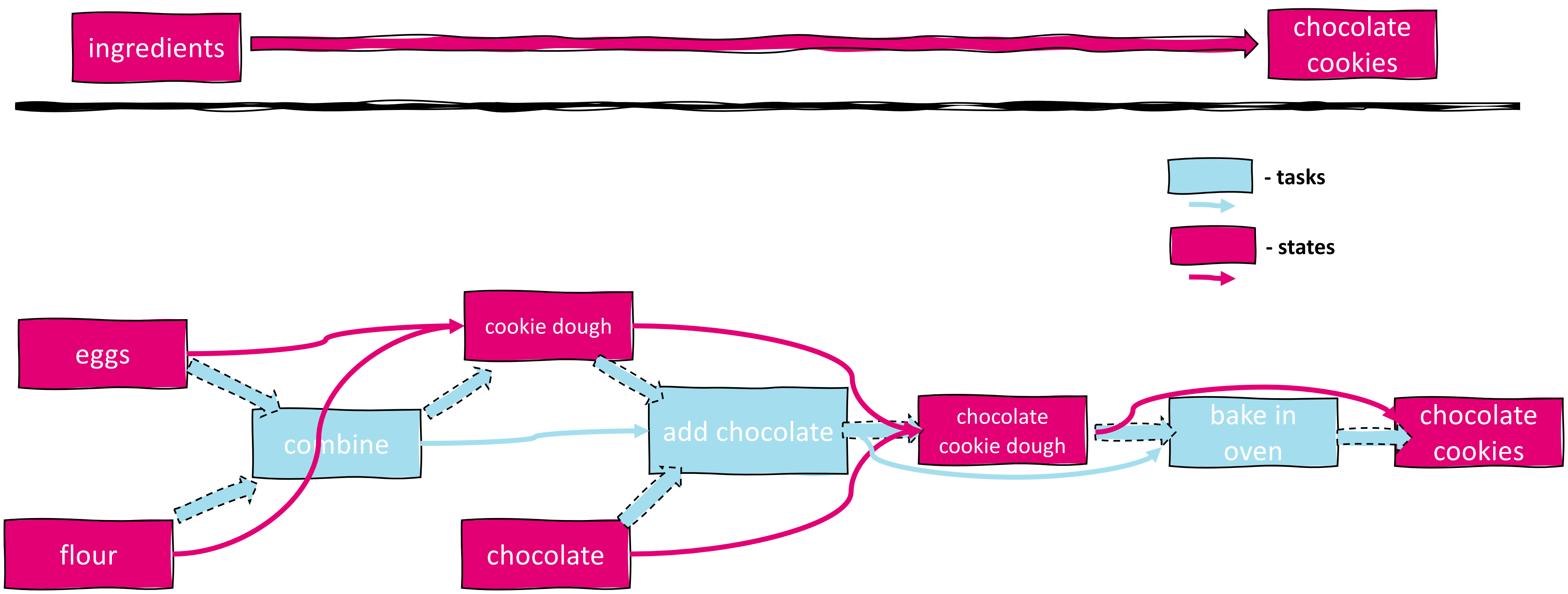
- Missing lineage
- Missing semantics
- Missing collaboration
- High lead times
- Limited quality

> “[...] model behavior is not determined by architecture, hyperparameters, or optimizer choices. It’s determined by your dataset, nothing else.”

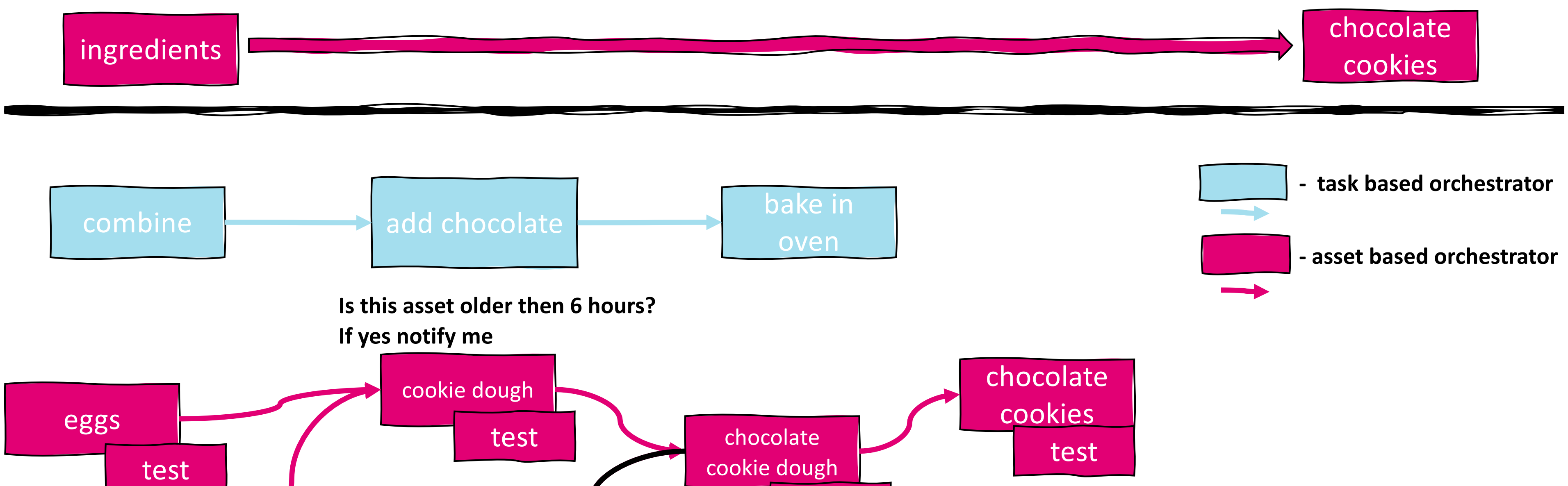
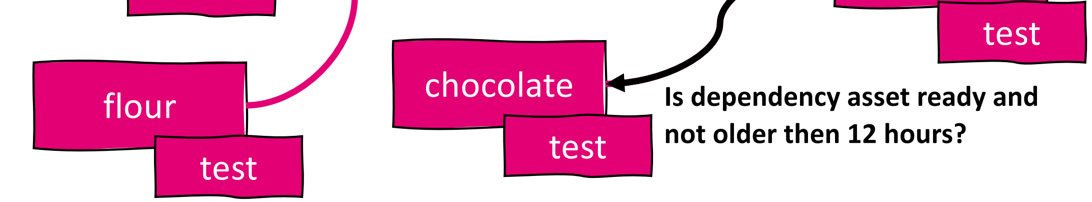
James Betker
Research Engineer, Open AI https://www.youtube.com/watch?v=IvhtTu9CTAU


ORACLE
PLSQL
talend
Informatica
730
350
8,995
upc
Governance & interface management
Explicit data dependencies
Unified tooling and guidelines
Breaking down tool silos
CLOUDERA

Asset and Task based orchestration: Chocolate cookie example



Asset based orchestration

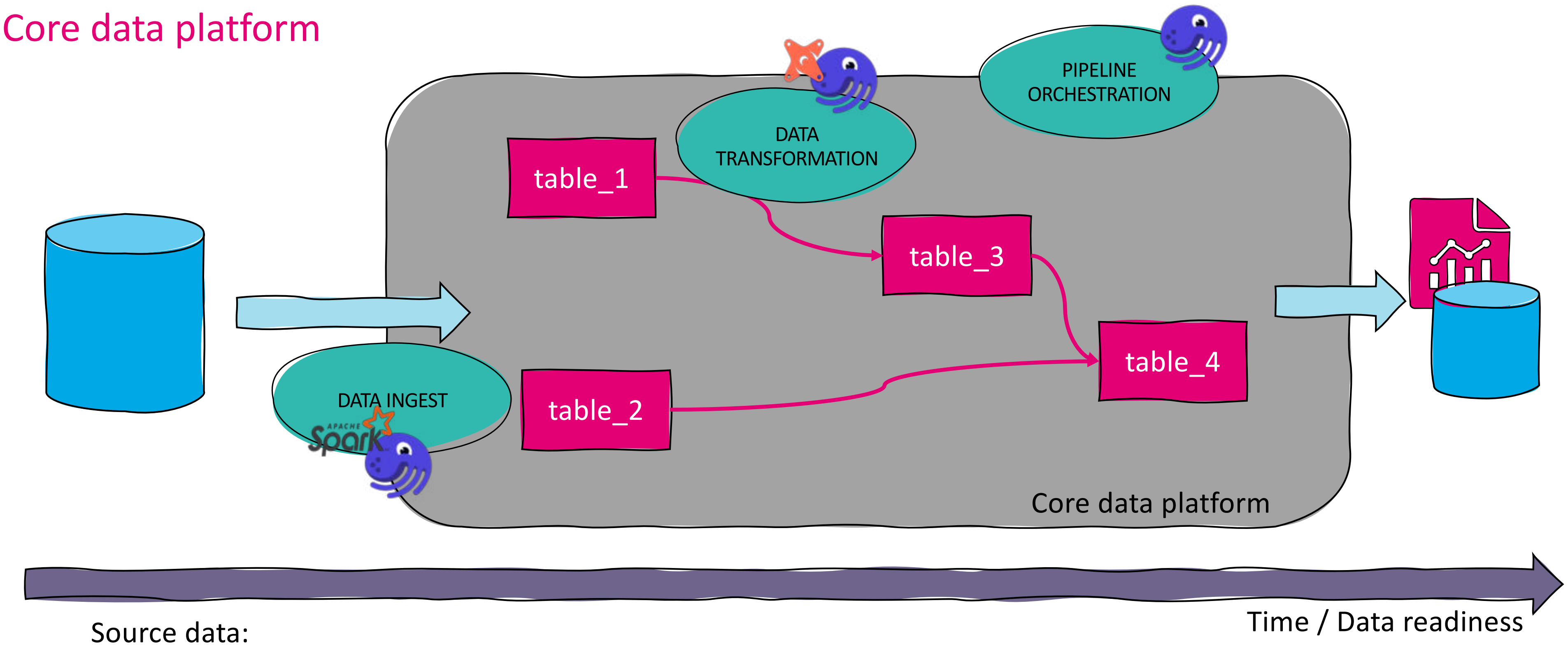
Advantages of asset-based orchestration:

- Asset testing

- Asset freshness

- Asset dependecy graph with granular declarative scheduling approach

Core data platform



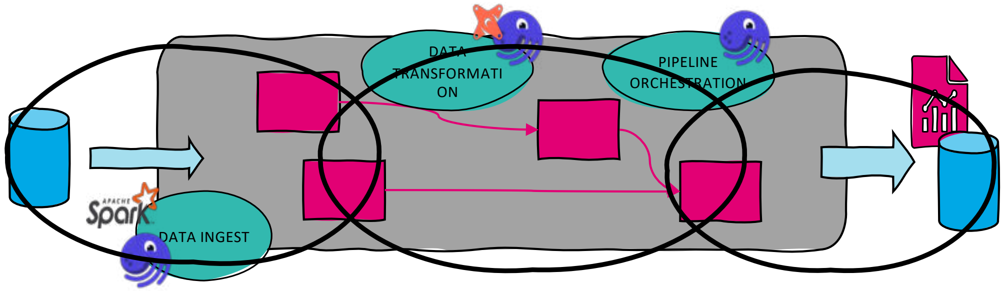
Source data:

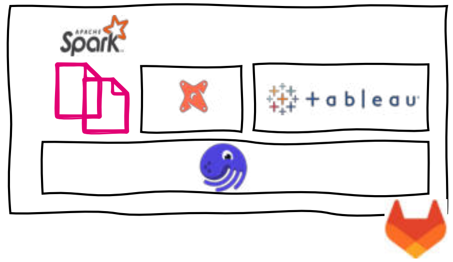
- Kafka
- Files
- Database systems

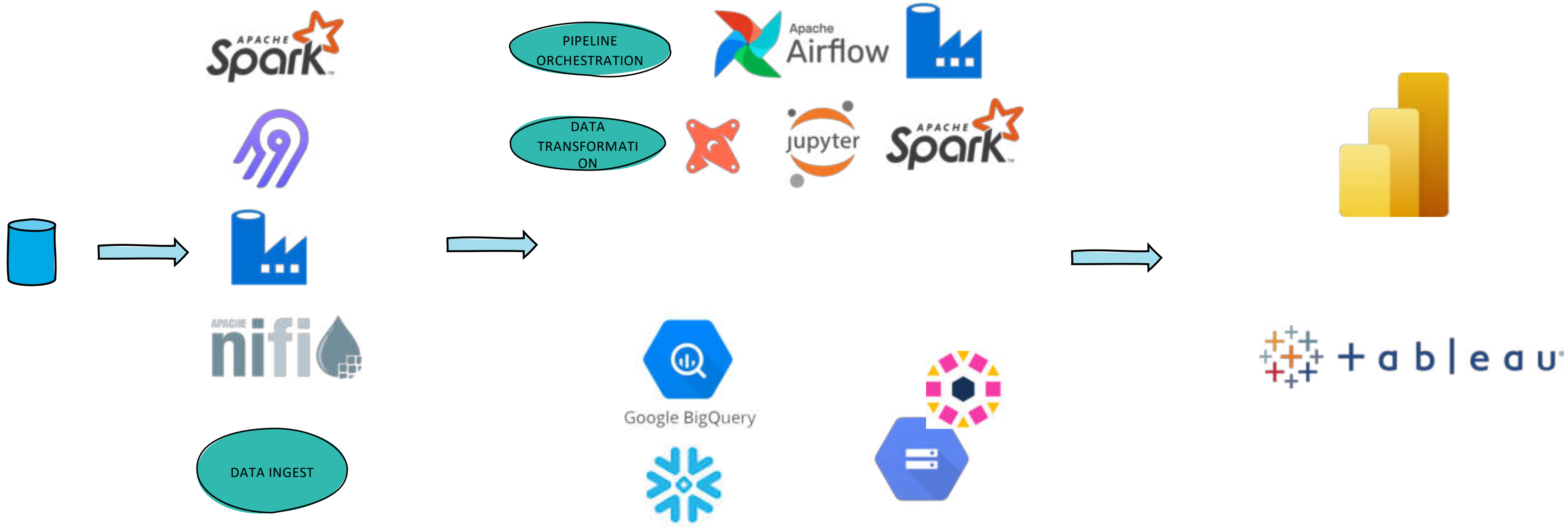
Understanding tool silos

What should i do to get E2E reporting use case done?

Developer

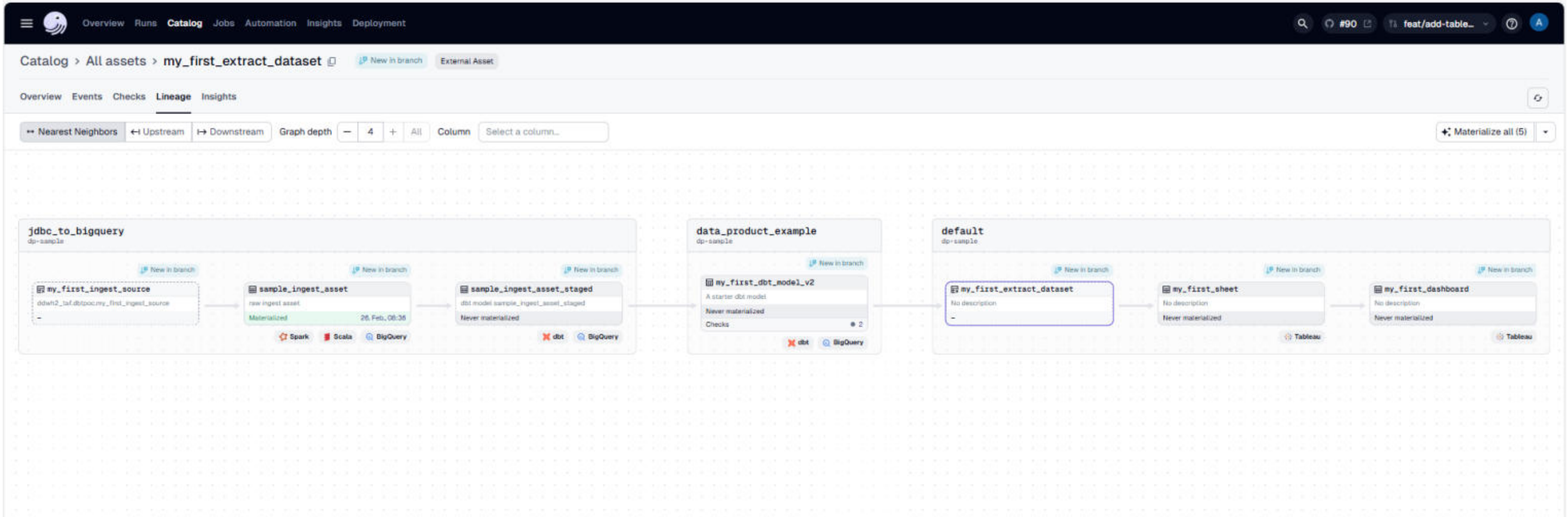




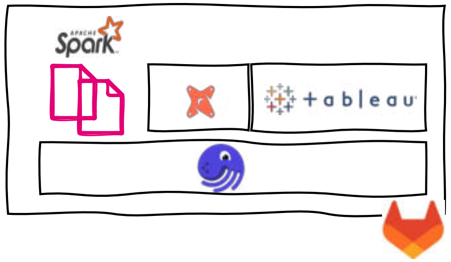


New enabled concepts

- Asset based graph
- Metadata driven pipeline creation
- Reusable Components
- ….



Machine-readable metadata pipeline generation



```python
metadata = collect_metadata()

list_of_assets = []
for each_metadata_node in metadata:
    #.. use_metadata
    @asset
    def asset():
        #..use_metadata

    list_of_assets.append(asset)

Definitions(assets = list_of_assets)
```

```python
tableau_server = TableauServer(creds)
#send rest api call to tableau server and get information
tableau_metadata = tableau_server.get_metadata()
list_of_assets = []
for each_tableau_object in tableau_metadata:
    tableau_object_deps = each_tableau_object.deps
    tableau_object_name = each_tableau_object.deps
    if each_tableau_object.type == extract_datasource:
        @asset(
            name = tableau_object_name,
            deps = tableau_object_deps
        )
        def refresh_extract(tableau_server):
            #send api call to refresh object
            tableau_server.refresh_extract(tableau_object_name)
        list_of_assets.append(run_dbt_asset)
    else:
        @asset(name = tableau_object_deps)
        def asset():
            pass
        list_of_assets.append(asset)

Definitions(assets = list_of_assets)
```

Reusable components

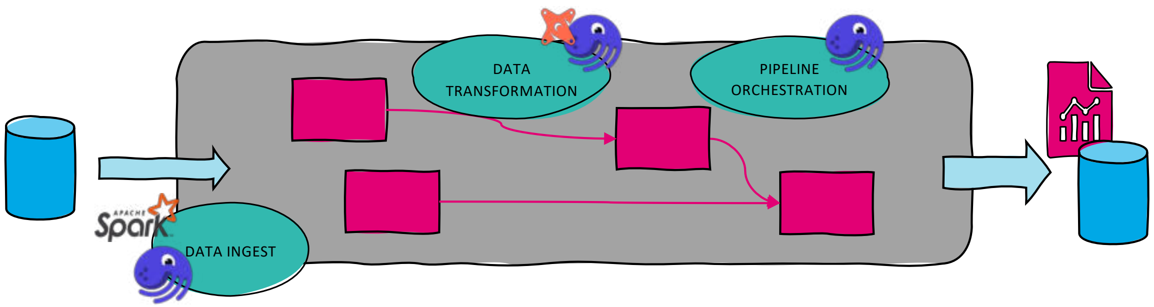
- Define once, test & reuse
- Resources → Encapsulate complex logic to interact with external systems
- IO manager → Make complex IO interactions substitutable & testable
- Benefits
  - Dependency injection
  - Day 1 productivity: Scale the data pipeline down to a single laptop
  - Increase self-service: Business/DS focus not required to handle complex IO

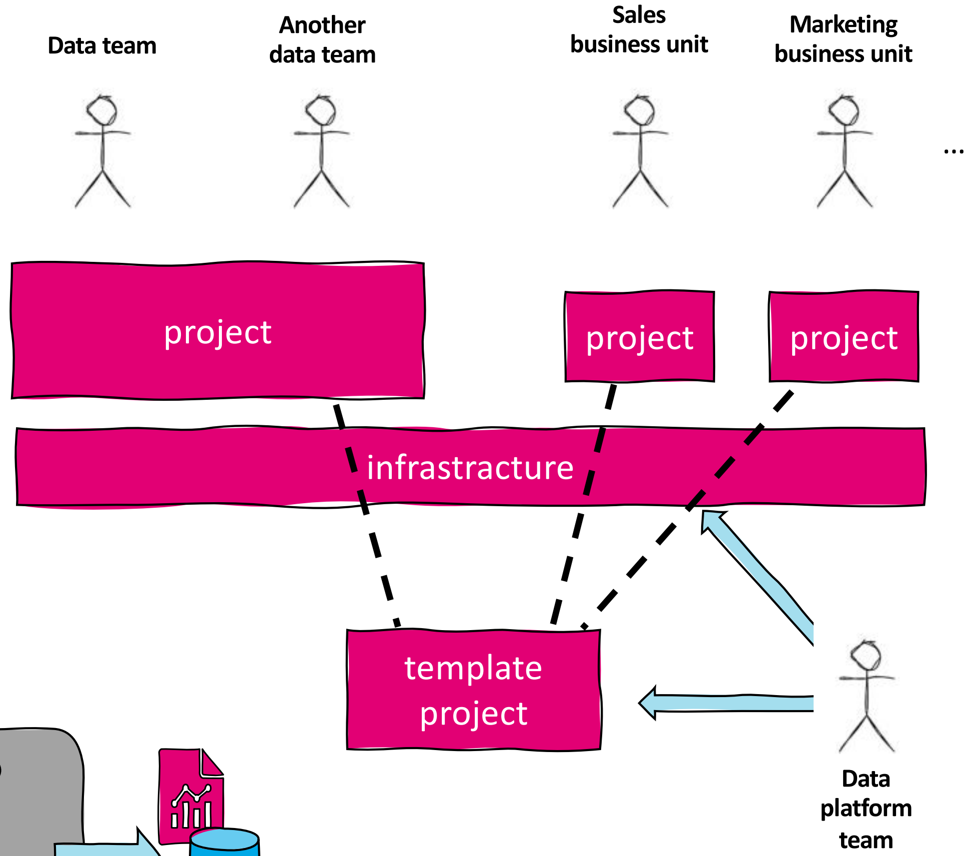
```python
@asset(
    io_manager_key="bigquery_io_manager",
)
def awesome_ml_model(context, reference_addresses: pd.DataFrame, bigquery: BigQueryResource) -> pd.DataFrame:
    # simple normal python code here
    # IO is abstracted
    context.log.info(f"from source: \n{reference_addresses.head()}")
    # auth & complexity (imagine web API) is abstracted
    with bigquery.get_client() as client:
        job = client.query("select * from example.upstream")
        query_result = job.result().to_dataframe()
        context.log.info(f"direct query: \n{query_result.head()}")
    return pd.DataFrame({"foo": [1,2,3]})
```

Observation

- Process is straight forward: ingest, transform, use
- Everything we do - we do for business to provide better service
- Hard to scale across company

- Divide people into **develop framework** and **use framework** groups
- Thinking in **building blocks**
- Tooling supporting software engineering practices: **dbt, dagster, pixi, docker**
- Introduction of **new processes**, **modeling** and **metadata tooling** for better governance

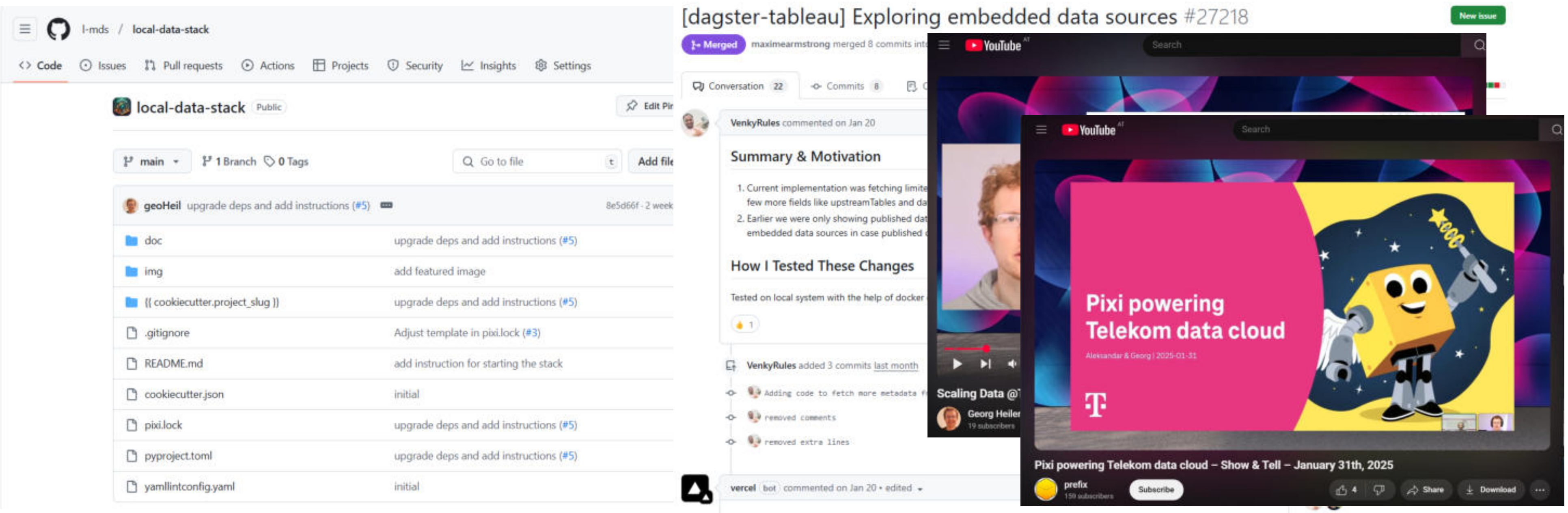




Takeaways

- Integrated asset-based graph is key (from ingest, transformation, reporting, tests – to AI)
  - Event driven connection
  - Better collaboration (scaling)
  - Can enable execution environment re-targeting in advanced cases
- Software engineering principles enable business self service
  - Blueprint
  - Automate all the things: CI/CD (stateful & stateless)
  - DRY: build tested foundation – dependency injection
  - Make business departments part of the key processes and pipelines
- Executable specification (metadata, contracts)
  - Interface Mangement
  - Preserve semantics
  - Preserve compliance (security classification, PII, retention)

Explore for yourself!



[Scaling data pipelines @Telekom](#)
[Pixi powering Telekom data cloud](#)
[Declarative Execution](#)
[In-depth explanation](#)
[dagster-ducklake](#)
[dagster-vertexai](#)

Data platform is team work and
we are very proud and excited about the journey ahead

OLAP ELO Ranking

- Relative
- Continuously improving
- Robust

georgheiler.com/post/elo-data-challenge

1	*StarRocks	2017	
2	*Kinetica	2015	
3	*Apache Hudi	2012	-2
4	*Databricks SQL	2005	-5
	*DuckDB	2004	-20
	*Clickhouse	2004	+19
	*Delta Lake	1985	-2
8	*TrinoDB	1985	
9	*Snowflake	1981	-5
10	*Apache Spark	1975	