Big Data Platforms - Recitation Workshops

Workshop III
Orchestration

4 January 2023

Yarden Fogel & Roei Arpaly

Orchestration - Agenda

Today's Topics - Recitation III

- Orchestration and Data pipelines
- ETL, ELT
- Batch, Streaming, Real time
- Data Granularity, Synthetic measures
- DAGs and Airflow
- Python Notebook Exercises

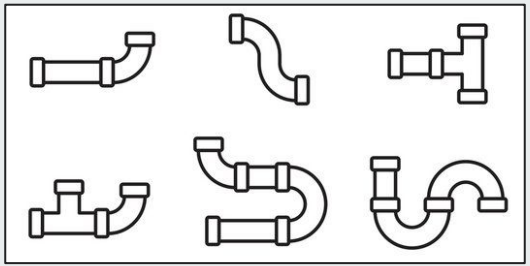
Reichman University

Data Orchestration

- Data orchestration describes the process of automating the data pipeline process (i.e. run a certain pipeline with a scheduler).

Data pipelines

- In computing, a data pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one.
  The elements of a pipeline are often executed in parallel.



Data pipelines

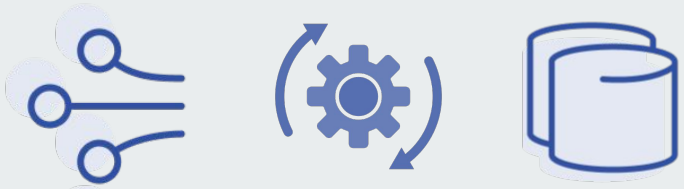
- ETL and ELT are data integration methods, which transfer data from one place to another. Their most important difference is that ETL transforms data before loading it on the server, while ELT transforms it afterward.
- ETL - Extract Transform Load
  - Transforms data **before** loading it to the Data Lake/Warehouse
  - The raw data is preprocessed, enriched and the output is saved
  - Raw data is not saved (only the transformed data)
- ELT - Extract Load Transform (More common in Big Data Environments as it is more flexible, efficient and scalable)
  - Transforms data **after** loading it to the Data Lake/Warehouse
  - The enrichment comes after storing the raw data
  - Raw data is available

ETL, ELT

Logs,
Files,
Databases,
Applications,
APIs

ETL

Extract → **Transform** → **Load**



Extract → **Load** → **Transform**

ELT

Analytics

Data Granularity

- Data Granularity -
  Describes the level of detail in a data structure.
- The highest granularity (raw) allows access to more details.
  Some of the details can be preserved even after aggregation.
- Low granularity is easier and faster to query, great for dashboards should be responsive and takes less storage (costs optimisation).

Data Granularity

Raw data → Hourly aggregation → Daily aggregation

timestamp	user_id	purchases
2022-01-01 00:00:00	12345	0.99
2022-01-01 00:01:01	12345	2.99
2022-01-01 02:02:02	12345	4.99
...	...	...
2022-12-31 23:59:59	12345	9.99

timestamp	user_id	purchases
2022-01-01 00	12345	3.98
2022-01-01 02	12345	4.99
...	...	...
2022-12-31 23	12345	9.99

date	user_id	purchases
2022-01-01	12345	8.97
...	...	...
2022-12-31	12345	9.99

Example for data pipeline

Synthetic measures

- **Synthetic measures definition:** A metric / measure / indicator which is derived from combining other metrics or metric and a value.
- These metrics usually do not reside in the data warehouse, and will be calculated on the fly using SQL or analytics tools (BI).

Synthetic measures

Raw data → Hourly aggregation → Daily aggregation

timestamp	user_id	purchases
2022-01-01 00:00:00	12345	0.99
2022-01-01 00:01:01	12345	2.99
2022-01-01 02:02:02	12345	4.99
...	...	...
2022-12-31 23:59:59	12345	9.99

timestamp	user_id	avg_purchase
2022-01-01 00	12345	1.99
2022-01-01 02	12345	4.99
...	...	...
2022-12-31 23	12345	9.99

date	user_id	avg_purchase
2022-01-01	12345	3.49
...	...	...
2022-12-31	12345	9.99

Synthetic measures

Raw data → Hourly aggregation → Daily aggregation

timestamp	user_id	purchases
2022-01-01 00:00:00	12345	0.99
2022-01-01 00:01:01	12345	2.99
2022-01-01 02:02:02	12345	4.99
...	...	...
2022-12-31 23:59:59	12345	9.99

timestamp	user_id	avg_purchase
2022-01-01 00	12345	1.99
2022-01-01 02	12345	4.99
...	...	...
2022-12-31 23	12345	9.99

date	user_id	avg_purchase
2022-01-01	12345	3.49
...	...	...
2022-12-31	12345	9.99

Synthetic measures

Raw data → Hourly aggregation → Daily aggregation

timestamp	user_id	purchases
2022-01-01 00:00:00	12345	0.99
2022-01-01 00:01:01	12345	2.99
2022-01-01 02:02:02	12345	4.99
...	...	...
2022-12-31 23:59:59	12345	9.99

timestamp	user_id	sum_purchases	cnt_purchases
2022-01-01 00	12345	3.98	2
2022-01-01 02	12345	4.99	1
...	...	...	...
2022-12-31 23	12345	9.99	1

timestamp	user_id	sum_purchases	cnt_purchases
2022-01-01 00	12345	8.97	3
...	...	...	...
2022-12-31 23	12345	9.99	1

Batch, Streaming and Real-time

- **Batch:** data pipeline which runs at defined time intervals (minutes / hours / daily).
- **Streaming:** data pipeline which runs continuously, processing raw data and storing it. Usually these types of processes have some latency (few seconds / minutes).
- **Real time:** data pipeline which runs on demand (request) and provide data output quickly (milliseconds).

Processes and Threads

- **Process**

  A process is an instance of a computer program. Each process has its own memory space it uses to store the instructions being run, as well as any data it needs to store and access to execute.

  Multiprocessing is used in cases where the program is **CPU intensive** and doesn't have to do any IO or user interaction.

- **Thread**

  Threads are components of a process, which can run parallely. There can be multiple threads in a process, and they share the same memory space (of the parent process).

  Threading is used when the task is **IO bound or network bound**. Multiple threads can take care of the parallelization.

Processes and Threads

- **Race condition**

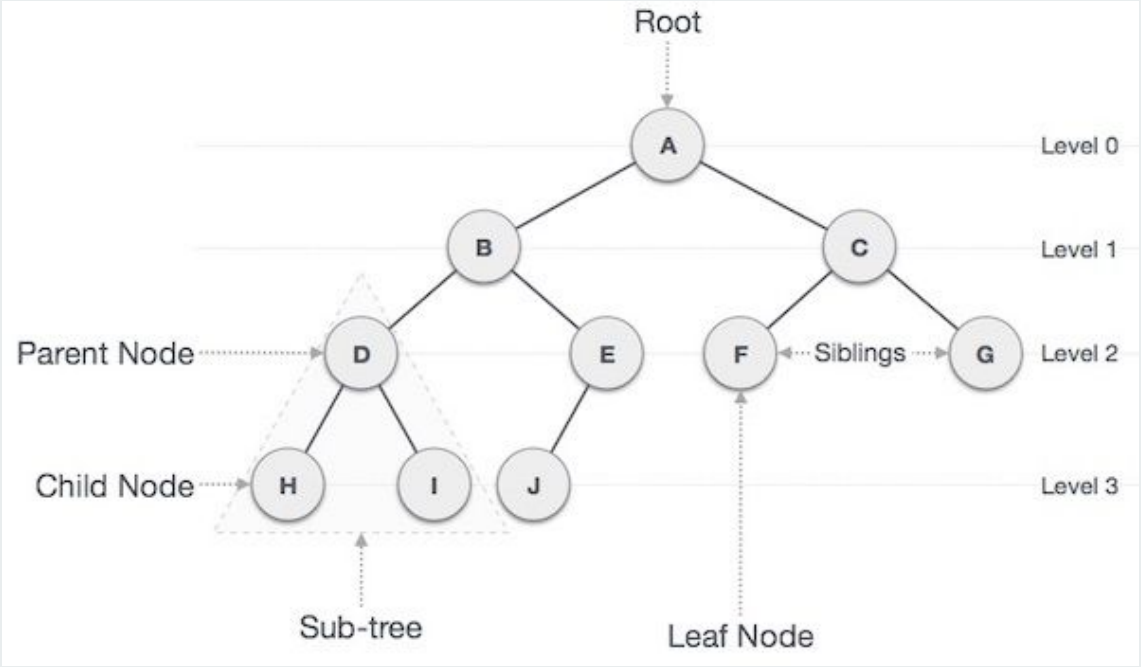
  Race condition is a significant problem in concurrent programming.
  The condition occurs when two or more threads try to modify a shared resource at the same time.

- **Locks (mutex)**

  A lock is a synchronization mechanism that enforces limits on access to a resource when there are many threads of execution.
  The threading/multiprocessing modules of Python include locks as a synchronization tool.
  - A lock has two states:
    - Locked: lock.acquire() method
    - Unlocked: lock.release() method

Trees - Refresher

- Root, Node, Leaf
- Parent, Child, Siblings
- Depth, Height
- Tree Traversal
  - Pre-order, NLR
  - Post-order, LRN
  - In-order, LNR



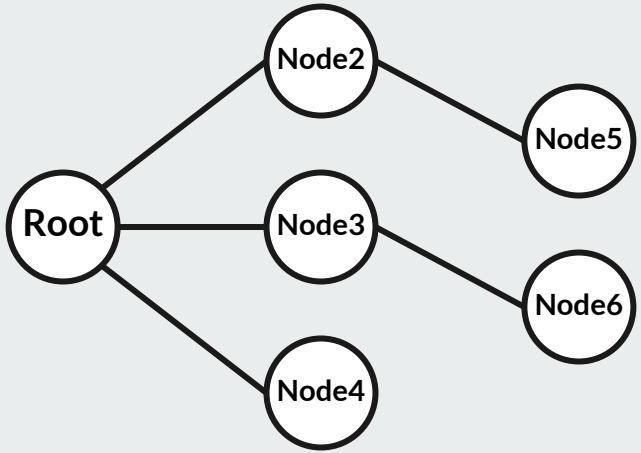
How to construct a tree

- Python code

```
class Node:
    def __init__(self, data):
        self.data = data
        self.children = []

root = Node(10)
node2 = Node(20)
node3 = Node(30)
node4 = Node(40)
node5 = Node(50)
node6 = Node(60)

root.children.extend([node2, node3, node4])
node2.children.append(node5)
node3.children.append(node6)
```

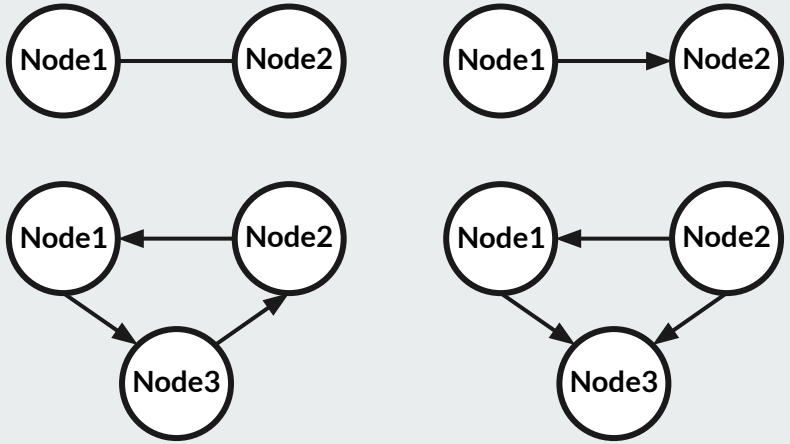


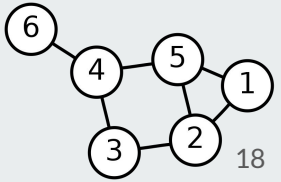
Reichman University

Graphs

- Undirected, Directed
- Cyclic, Acyclic

- Weighted, Unweighted
- Sparse, Dense
- Self loops, Bipartite, and more

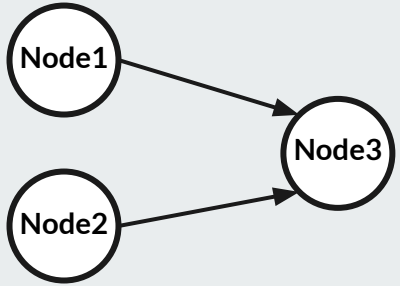




DAGs

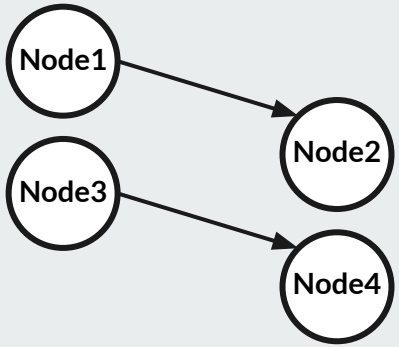
- DAG is an acronym for Directed Acyclic Graph
- DAG is not always a tree - example 1
- DAG can have disconnected components - as in Example 2 below
- JSON and YAML documents can be thought of as trees

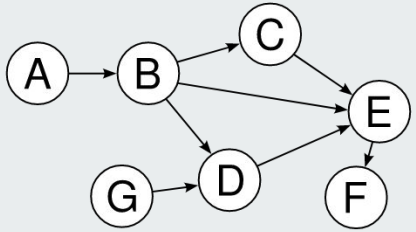
Example 1




```
{
 root: [node2, node3, node4],
 node2: [node5],
 node3: [node6],
 node4: [],
 node5: [],
 node6: [],
 }
```

Example 2

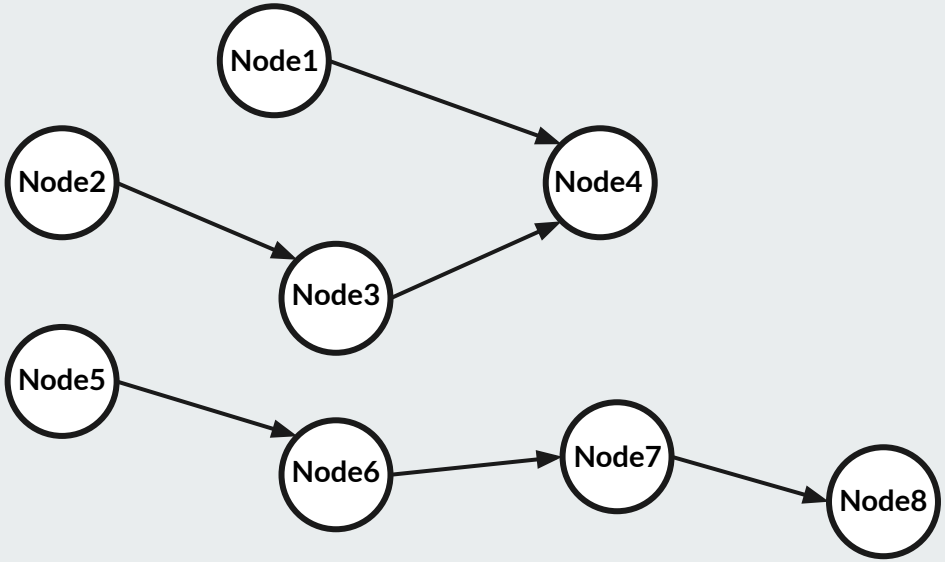




Reichman University

DAGs

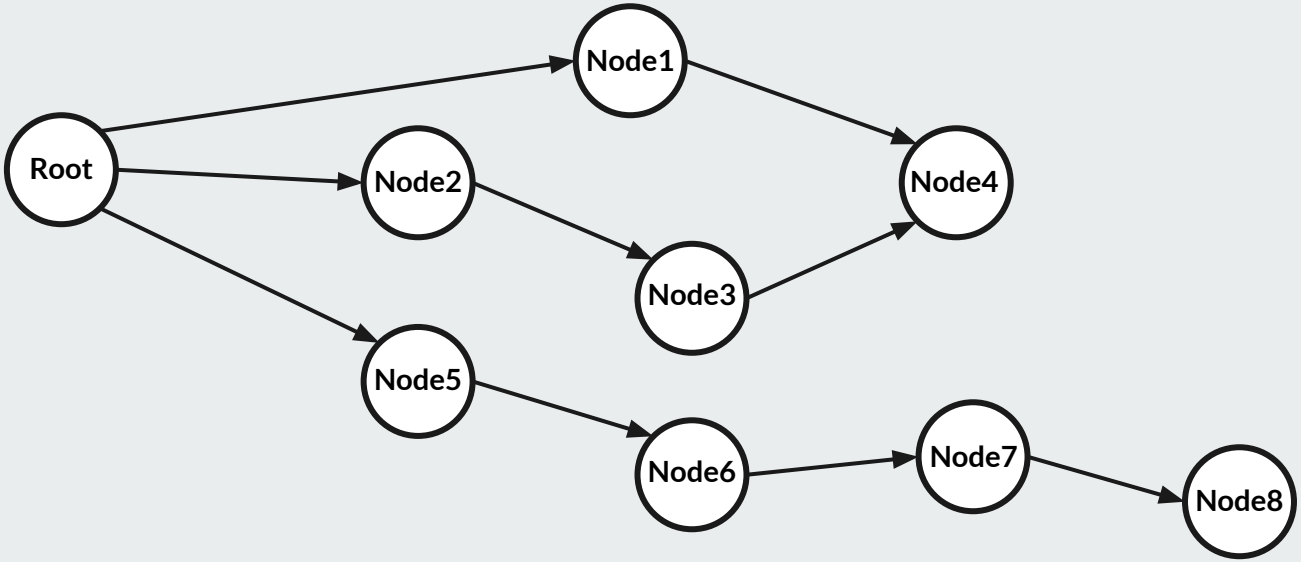
- How can we always easily connect disconnected components?



Reichman University

DAGs

- How can we always easily connect disconnected components? ⇒ add a root

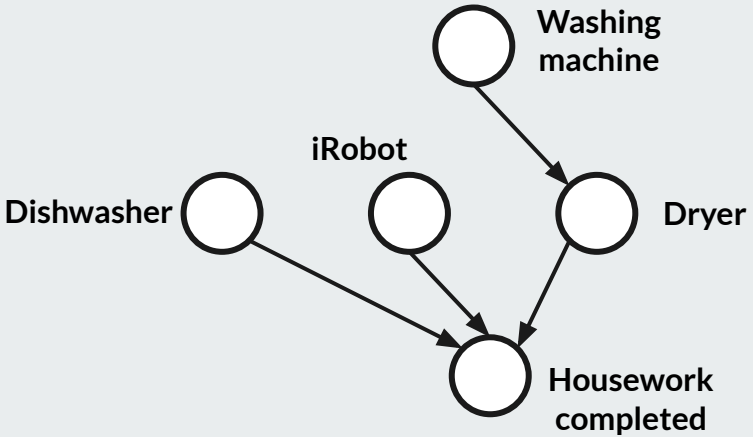


Topological Sorting

- Topological sorting of a directed graph is a linear ordering of its vertices such that for every directed edge *uv* from vertex *u* to vertex *v*, *u* comes before *v* in the ordering.
- A topological ordering is possible **if and only if the graph is directed acyclic graph** (DAG).
- Topological sorting is possible even when the DAG has **disconnected components**.
- A DAG can have more than one valid topological sort.
- **Generalization of Pre-order** (NLR) traversal for non-binary trees.

Topological sorting

- A widely used application of topological sorting is in **scheduling a sequence of jobs or tasks based on their dependencies.** The jobs are represented by vertices, and there is an edge from x to y if job x must be completed before job y can be started. A topological sort gives an order in which to perform the jobs.



Possible Topological sorts:

[Dishwasher, iRobot, Washing machine, Dryer, Completed]
Dishwasher >> iRobot >> Washing machine >> Dryer >> Completed

[Washing machine, Dishwasher, iRobot, Dryer, Completed]
Washing machine >> Dishwasher >> iRobot >> Dryer >> Completed

But we want to do it in Parallel!
[Dishwasher, iRobot] >> Completed
Washing machine >> Dryer >> Completed

Reichman University

Topological sorting - Kahn's algorithm

```
L ← Empty list that will contain the sorted elements
S ← Set of all nodes with no incoming edge

while S is not empty do
    remove a node n from S
    add n to L
    for each node m with an edge e from n to m do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S

if graph has edges then
    return error   (graph has at least one cycle)
else
    return L   (a topologically sorted order)
```

Wait!

you forgot that this is a big data course...

Airflow

Apache Airflow is one of the most popular, widely used, open-source workflow management platform for data engineering pipelines.

It started at Airbnb in October 2014 as a solution to manage the company's increasingly complex workflows.

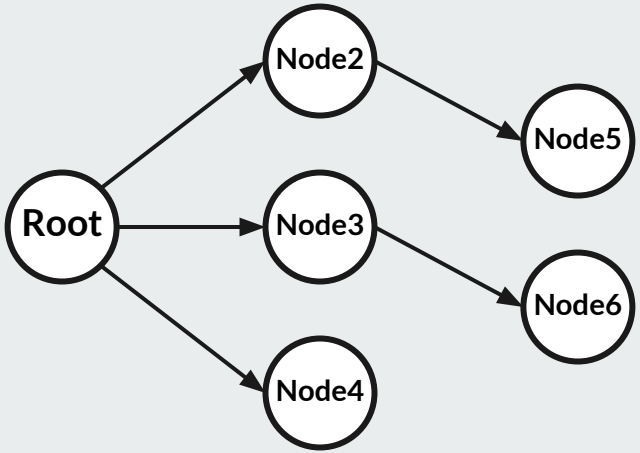
Airflow allows to author and schedule workflows and monitor them via a rich built-in user interface.

Airflow is all about DAGs!

github.com/apache/airflow

Airflow

- DAG
  A collection of tasks that follow a certain order and logic.
- Tasks, Operators
  A Task is the basic unit of execution in Airflow and Operator is a template for predefined task.
- Interval
  The running interval of the DAG, every x time
- Upstream, Downstream
  Go downstream the graph, or against the stream (upsteam)



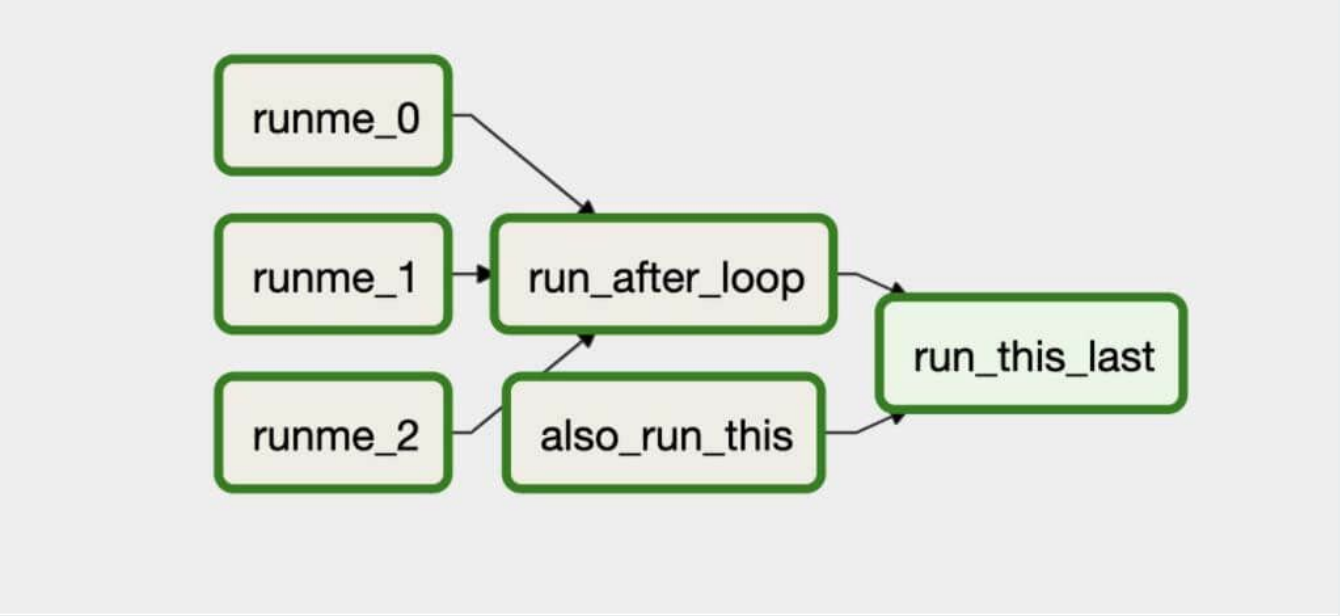
Parallel step

```
root >> [node2, node3, node4]
node2 >> node5
node3 >> node
```

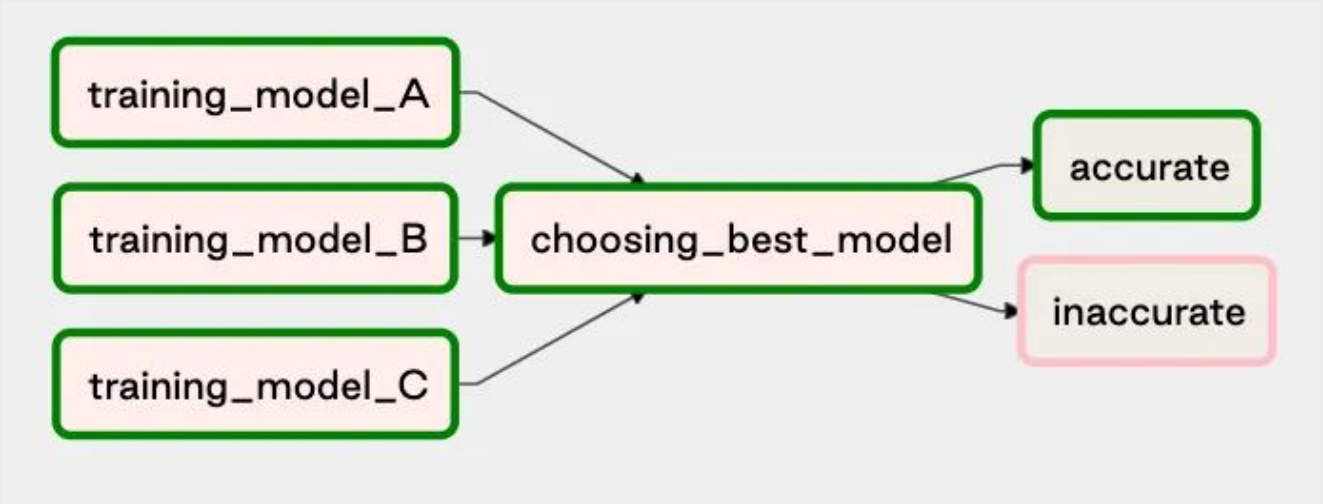
DAG examples

- Parallel tasks…
- Granularity…
- Multiple data sources
- Multiple Operators
- Machine Learning Pipeline

Airflow

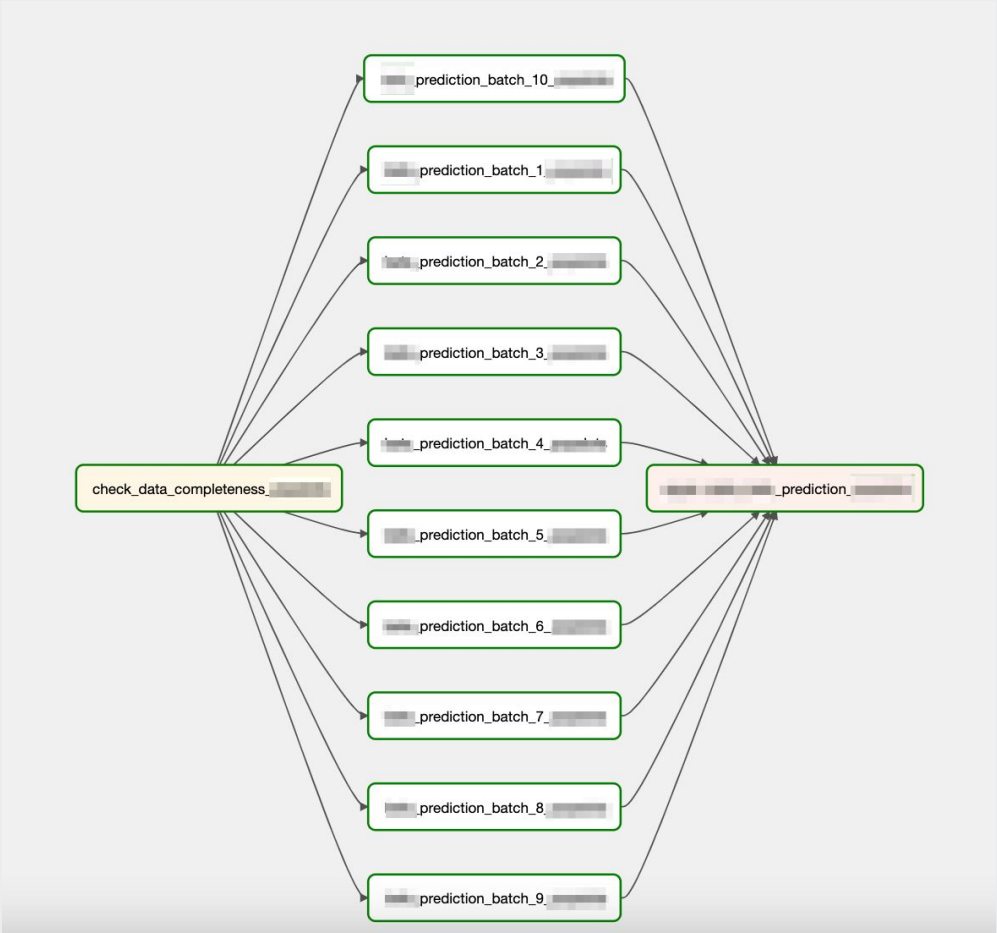

runme_0
runme_1
runme_2
run_after_loop
also_run_this
run_this_last

Airflow

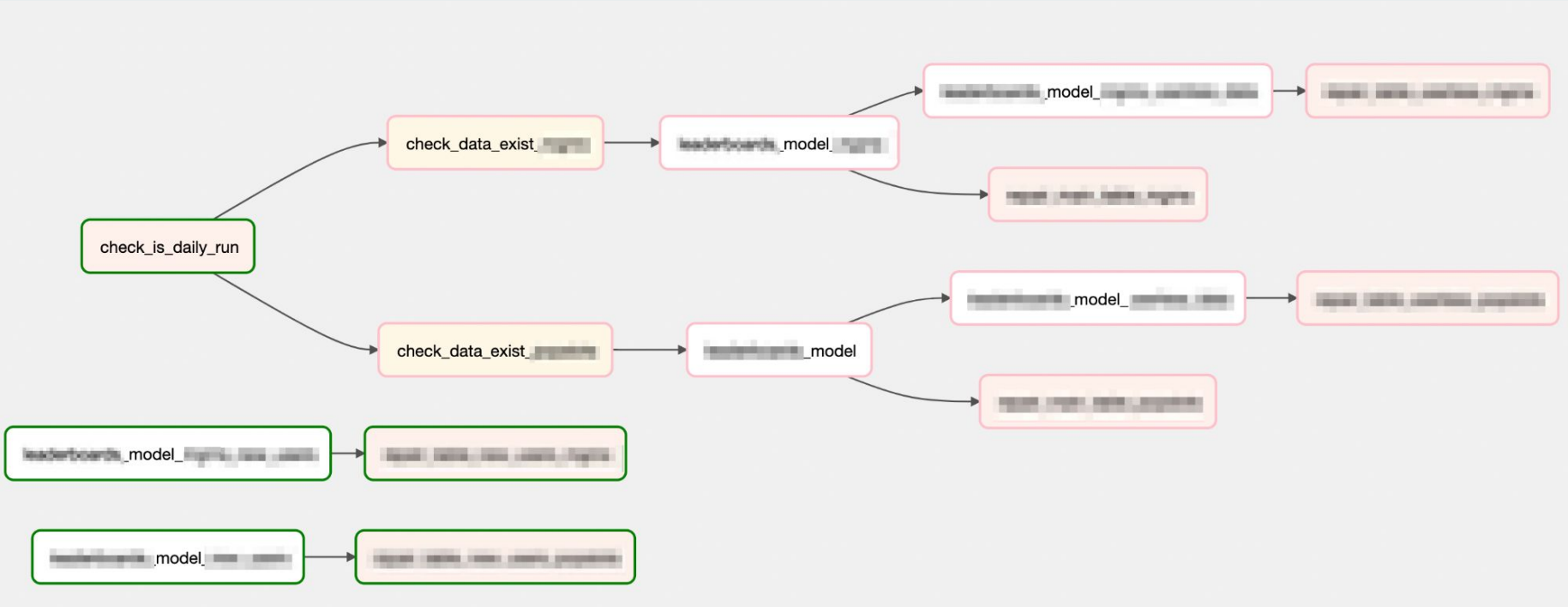

training_model_A
training_model_B
training_model_C
choosing_best_model
accurate
inaccurate

Reichman University

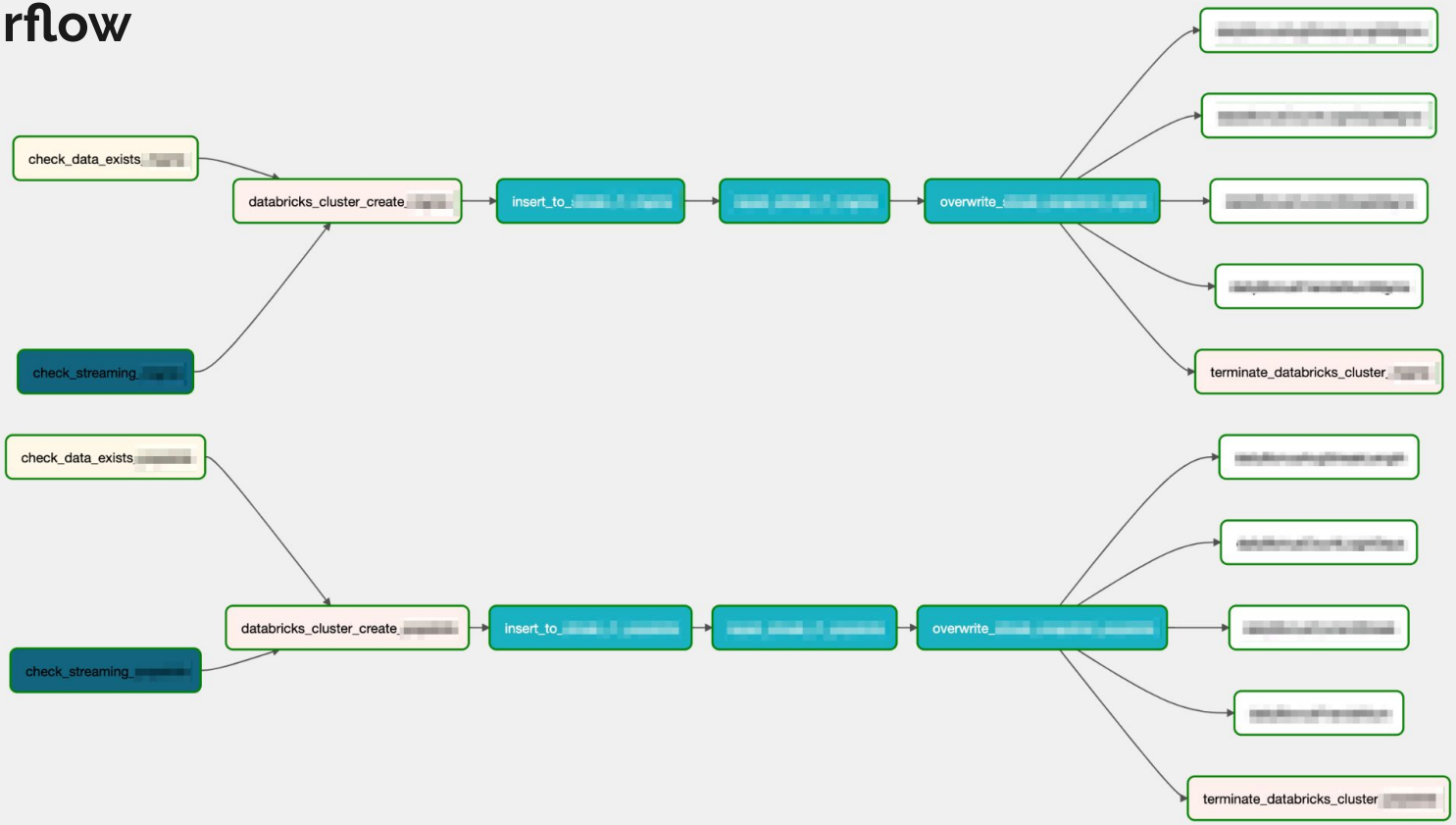
Airflow



Airflow


check_is_daily_run
check_data_exist_
model
check_data_exist_
_model
model
model
model
model

Airflow


check_data_exists
databricks_cluster_create
insert_to
overwrite
terminate_databricks_cluster
check_streaming
check_data_exists
databricks_cluster_create
insert_to
overwrite
terminate_databricks_cluster
check_streaming

Exercise

- **Exercise goals:**
  - Python project (coding exercise)
  - Refresher for CS concepts: OOP, override, parallelization, locks, graphs, data structures, etc
  - Get familiar with data pipelines (Airflow convention)

Open the notebook