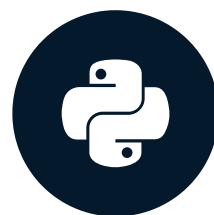


# Databases

INTRODUCTION TO DATA ENGINEERING



**Vincent Vankrunkelsven**

Data Engineer @ DataCamp

# What are databases?



- Holds data
- Organizes data
- Retrieve/Search data through DBMS

*A usually large collection of data organized especially for rapid search and retrieval.*

# Databases and file storage

## Databases



- Very organized
- Functionality like search, replication, ...

## File systems



- Less organized
- Simple, less added functionality

# Structured and unstructured data

**Structured:** database schema

- Relational database



**Semi-structured**

- JSON

```
{ "key": "value" }
```

**Unstructured:** schemaless, more like files

- Videos, photos



# SQL and NoSQL

## SQL

- Tables
- Database schema
- Relational databases



## NoSQL

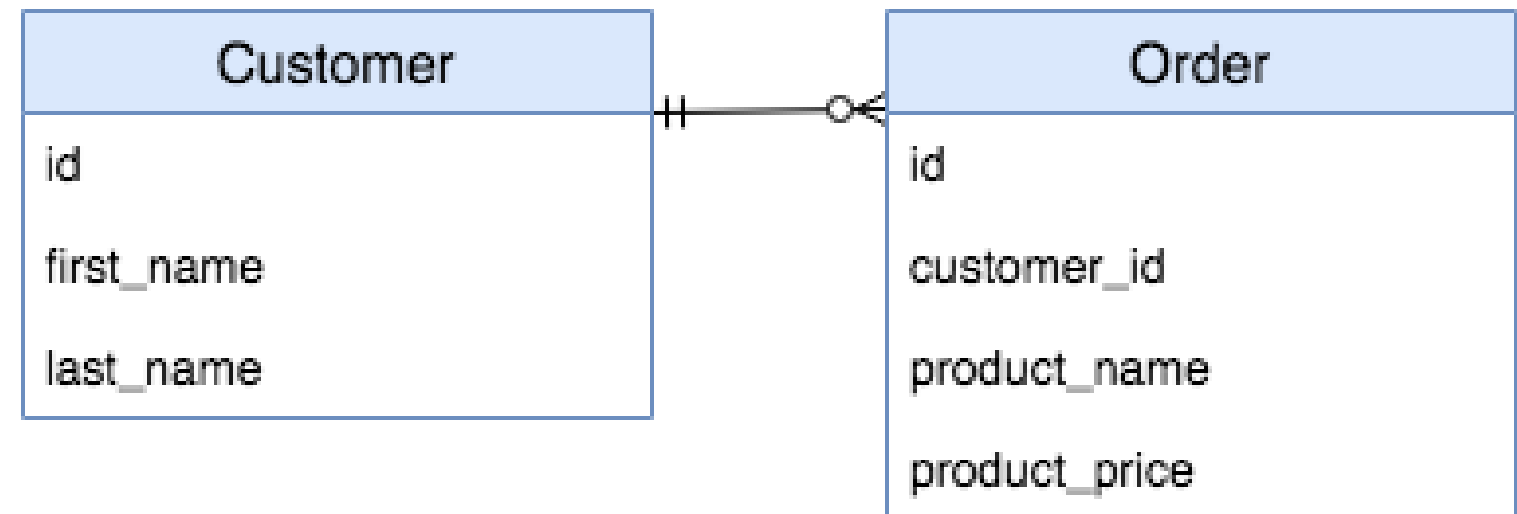
- Non-relational databases
- Structured or unstructured
- Key-value stores (e.g. caching)
- Document DB (e.g. JSON objects)



# SQL: The database schema

```
-- Create Customer Table
CREATE TABLE "Customer" (
  "id" SERIAL NOT NULL,
  "first_name" varchar,
  "last_name" varchar,
  PRIMARY KEY ("id")
);

-- Create Order Table
CREATE TABLE "Order" (
  "id" SERIAL NOT NULL,
  "customer_id" integer REFERENCES "Customer",
  "product_name" varchar,
  "product_price" integer,
  PRIMARY KEY ("id")
);
```

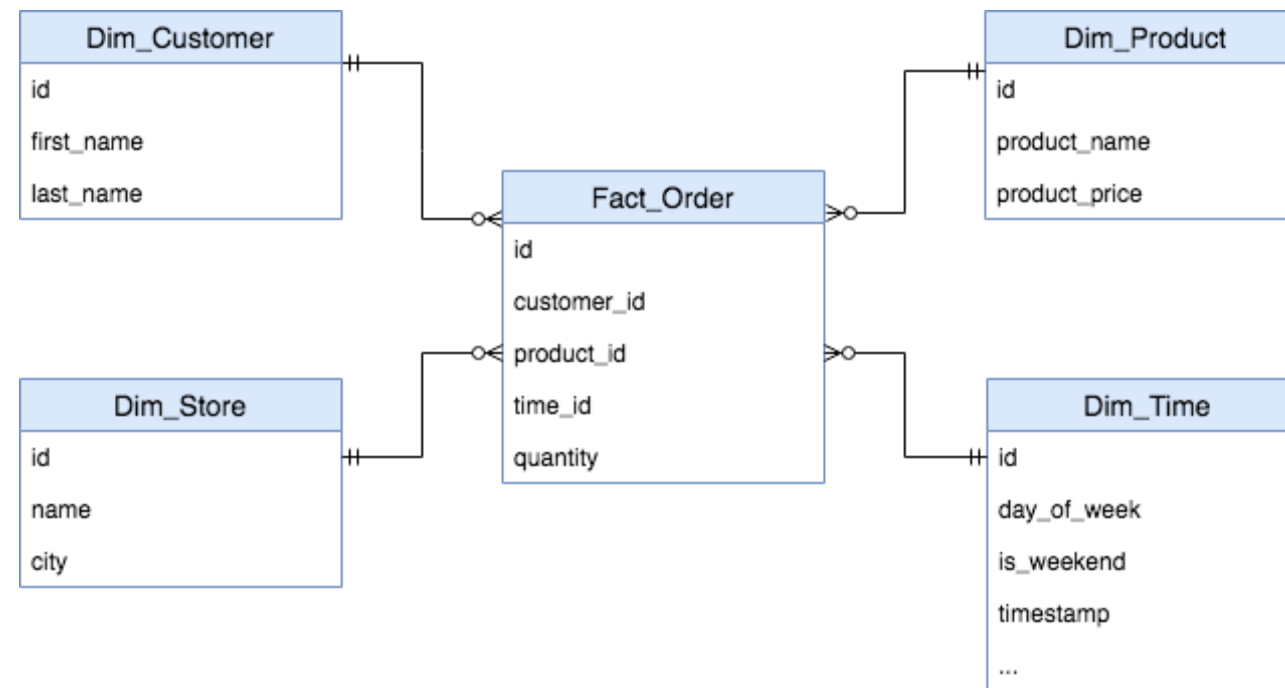


```
-- Join both tables on foreign key
SELECT * FROM "Customer"
INNER JOIN "Order"
ON "customer_id" = "Customer"."id";
```

id	first_name	...	product_price
1	Vincent	...	10

# SQL: Star schema

*The star schema consists of one or more fact tables referencing any number of dimension tables.*



- **Facts:** things that happened (eg. Product Orders)
- **Dimensions:** information on the world (eg. Customer Information)

<sup>1</sup> Wikipedia: [https://en.wikipedia.org/wiki/Star\\_schema](https://en.wikipedia.org/wiki/Star_schema)

# Let's practice!

INTRODUCTION TO DATA ENGINEERING



# What is parallel computing

INTRODUCTION TO DATA ENGINEERING



**Vincent Vankrunkelsven**  
Data Engineer @ DataCamp

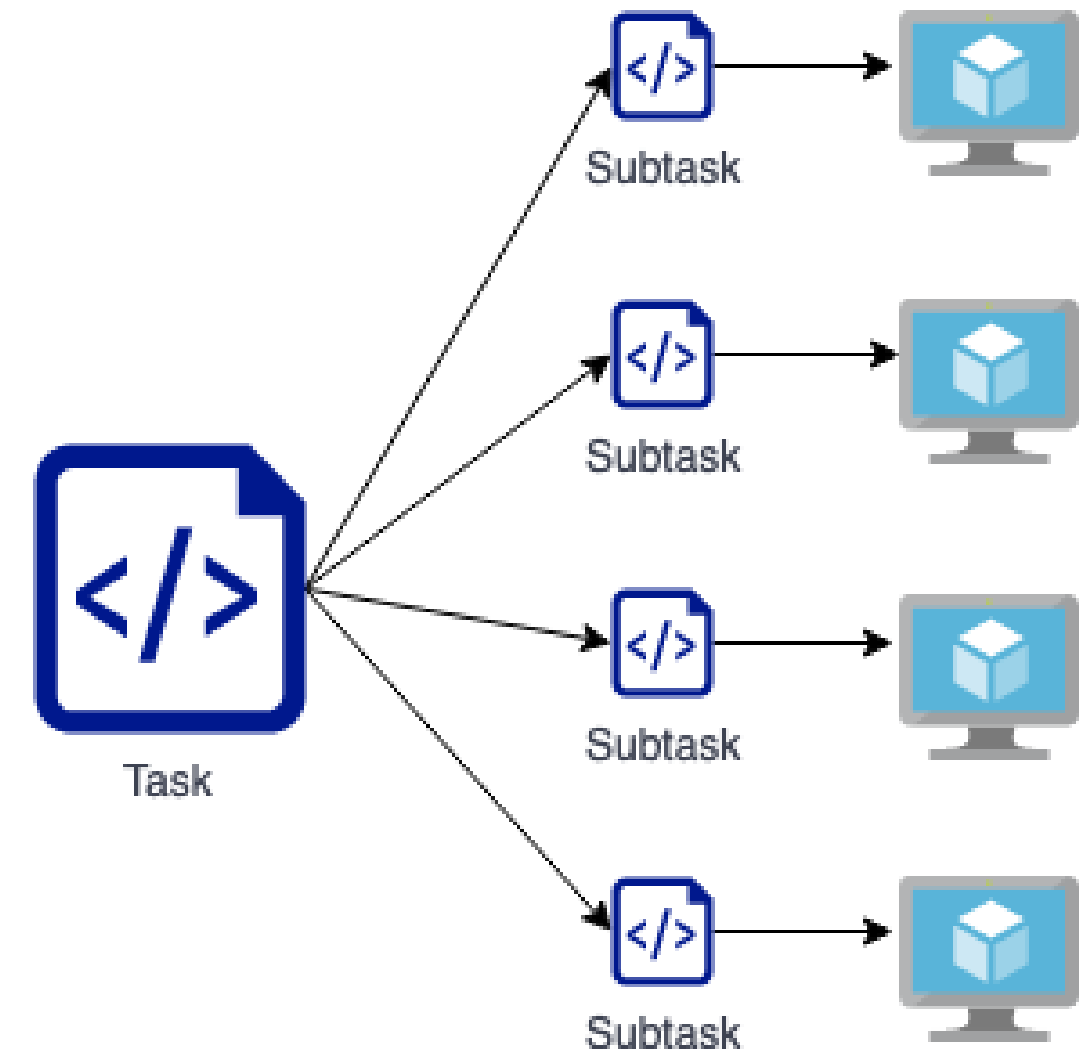
# Idea behind parallel computing

*Basis of modern data processing tools*

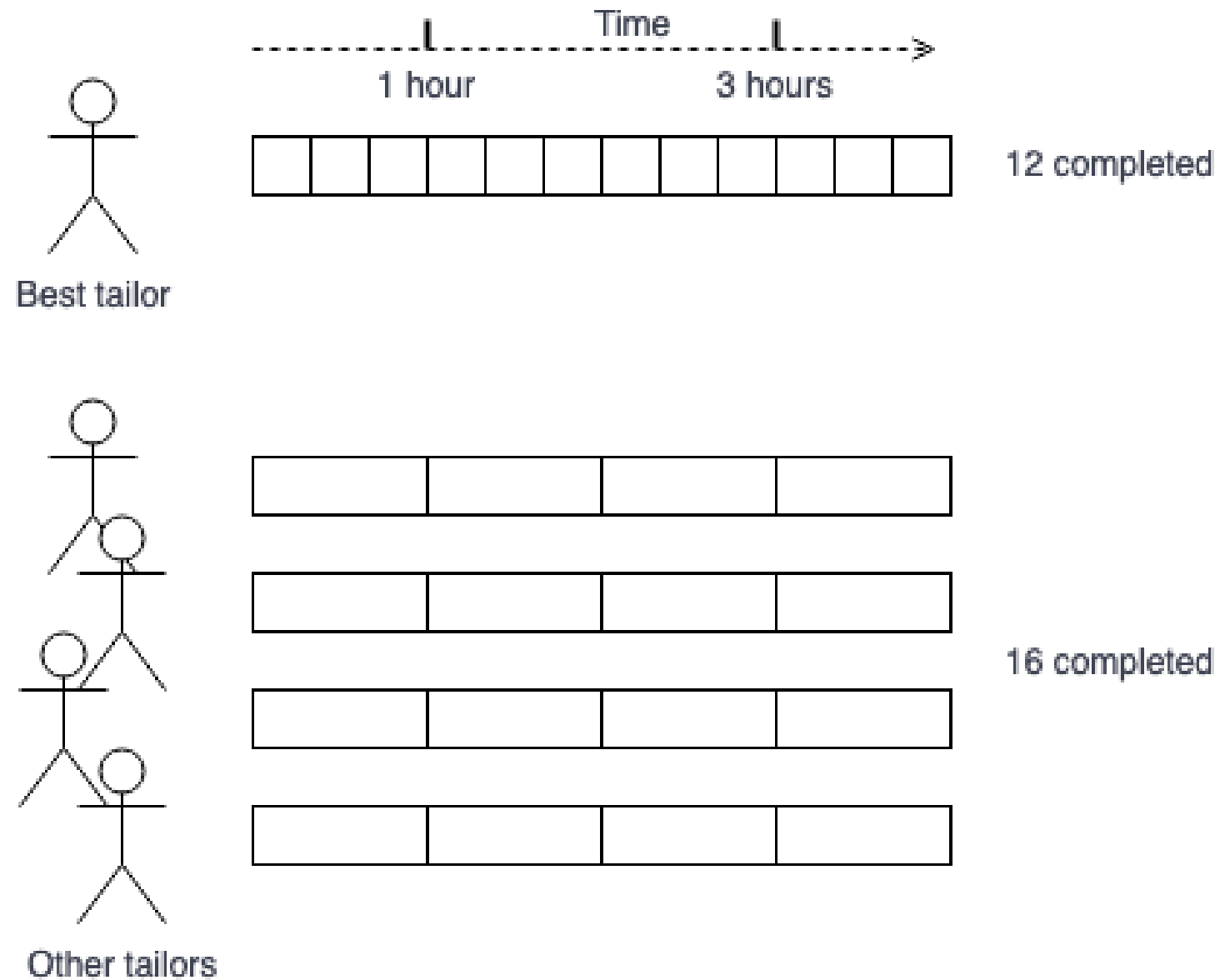
- Memory
- Processing power

## Idea

- Split task into subtasks
- Distribute subtasks over several computers
- Work together to finish task



# The tailor shop



## *Running a tailor shop*

**Goal:** 100 shirts

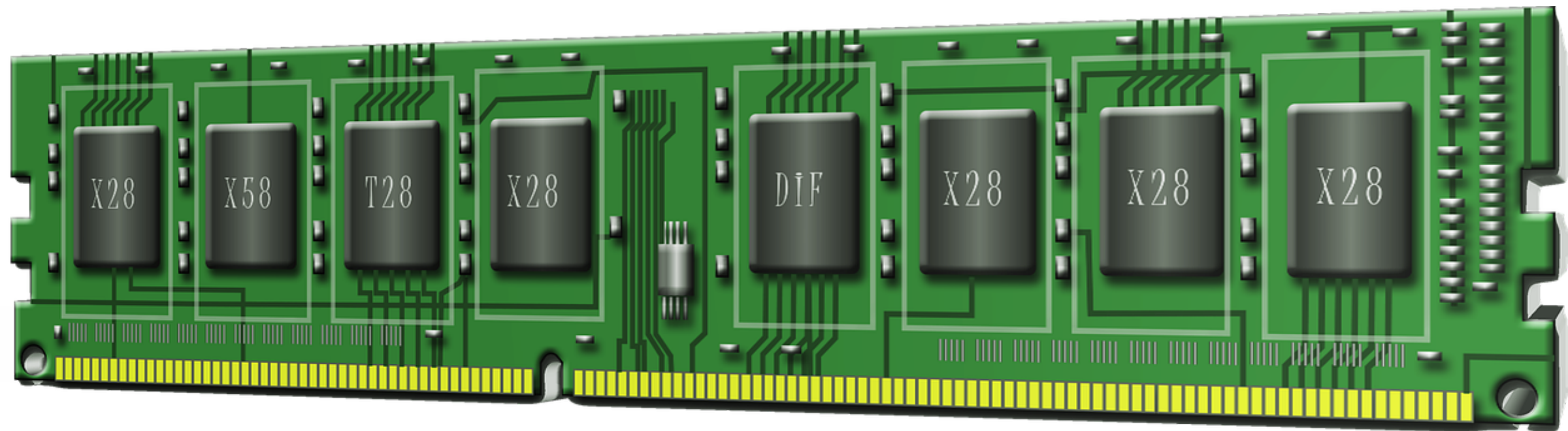
- Best tailor finishes shirt / 20 minutes
- Other tailors do shirt / 1 hour

Multiple tailors working together > best tailor

# Benefits of parallel computing

- Processing power
- Memory: partition the dataset

*RAM memory chip:*

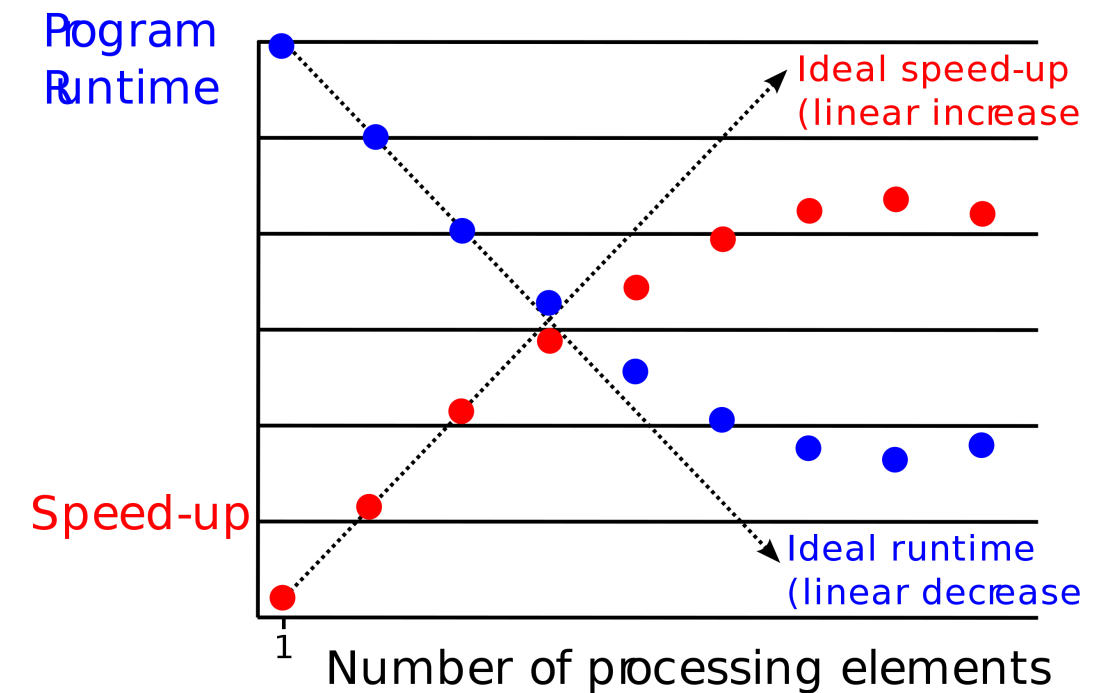


# Risks of parallel computing

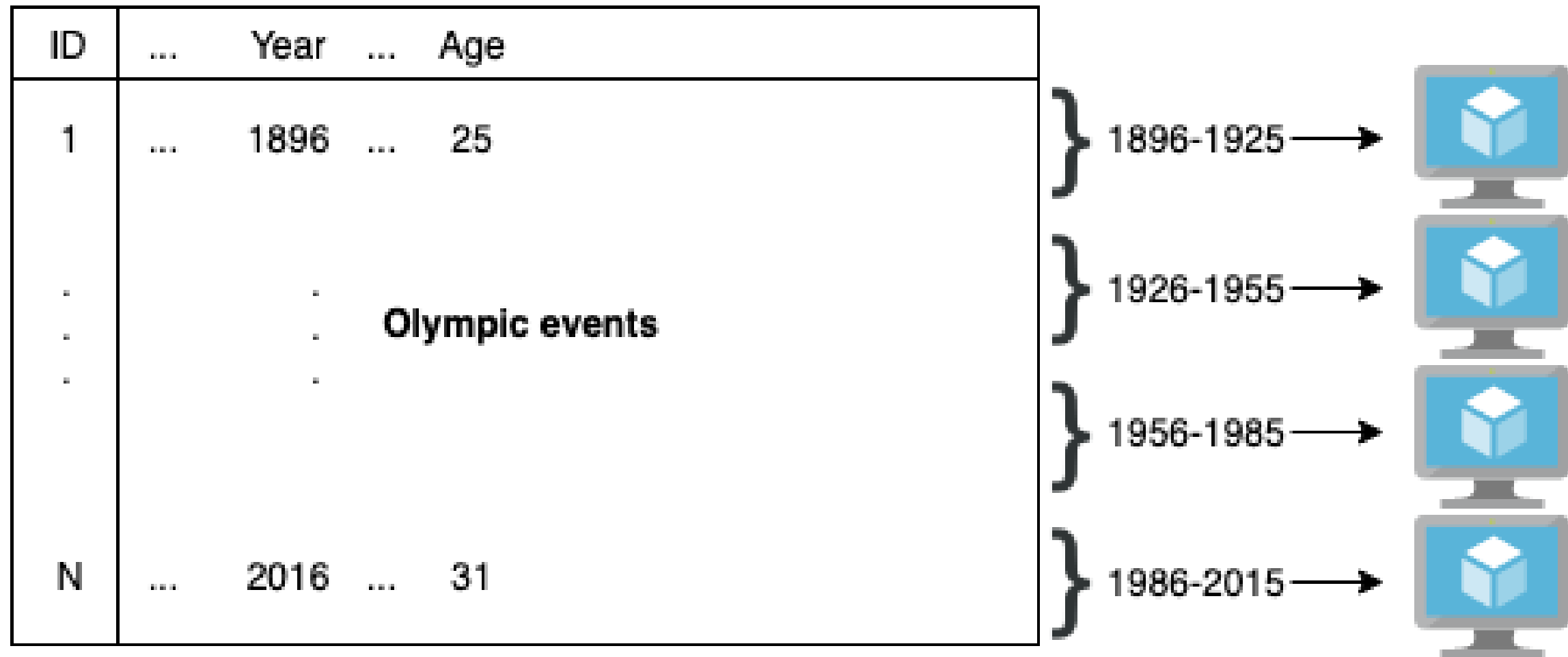
*Overhead due to communication*

- Task needs to be large
- Need several processing units

*Parallel slowdown:*



# An example



multiprocessing.Pool

```
from multiprocessing import Pool

def take_mean_age(year_and_group):
    year, group = year_and_group
    return pd.DataFrame({"Age": group["Age"].mean()}, index=[year])

with Pool(4) as p:
    results = p.map(take_mean_age, athlete_events.groupby("Year"))

result_df = pd.concat(results)
```

dask

```
import dask.dataframe as dd

# Partition dataframe into 4
athlete_events_dask = dd.from_pandas(athlete_events, npartitions = 4)

# Run parallel computations on each partition
result_df = athlete_events_dask.groupby('Year').Age.mean().compute()
```

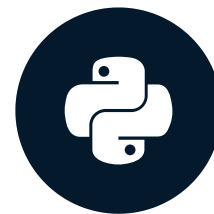


# Let's practice!

INTRODUCTION TO DATA ENGINEERING

# Parallel computation frameworks

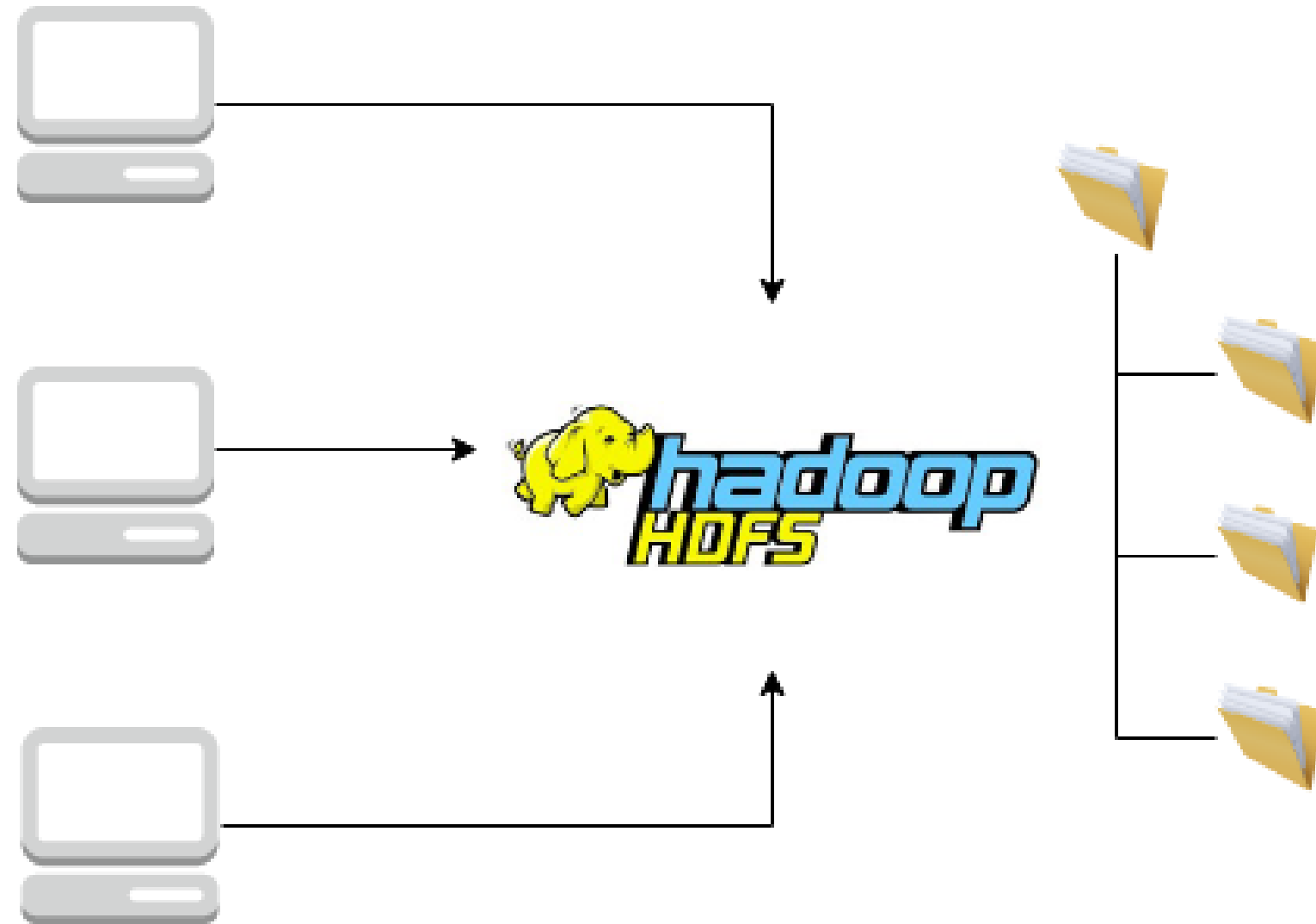
INTRODUCTION TO DATA ENGINEERING



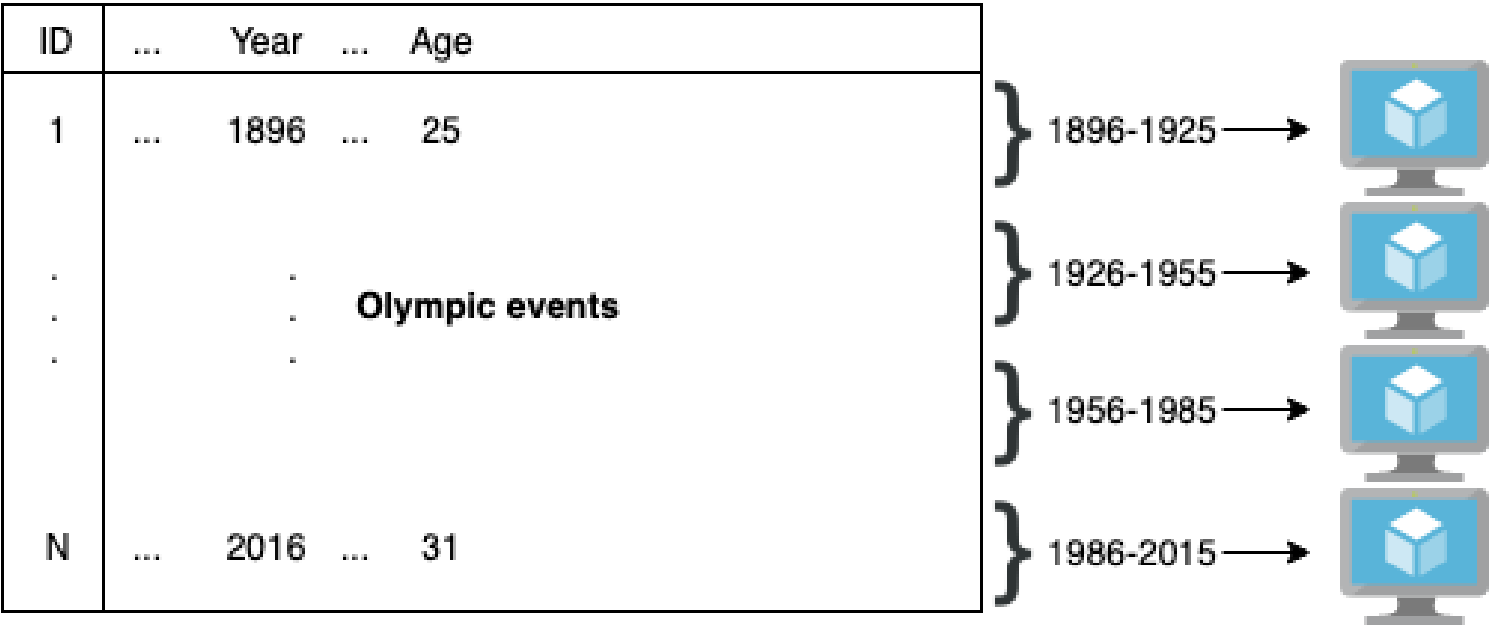
**Vincent Vankrunkelsven**  
Data Engineer @ DataCamp



# HDFS



# MapReduce



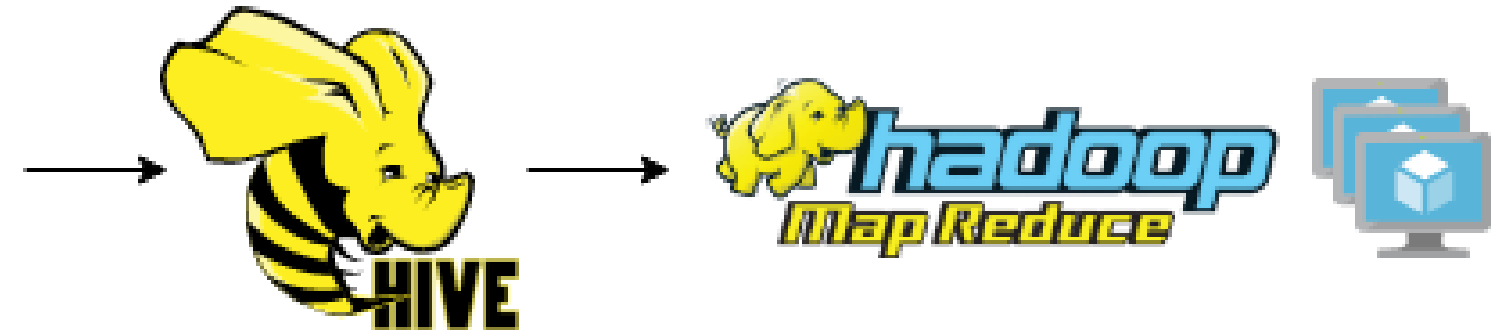
# Hive

- Runs on Hadoop
- Structured Query Language: Hive SQL
- Initially MapReduce, now other tools



# Hive: an example

```
SELECT year, AVG(age)
FROM views.athlete_events
GROUP BY year
```





- Avoid disk writes
- Maintained by Apache Software Foundation



# Resilient distributed datasets (RDD)

- Spark relies on them
- Similar to list of tuples
- Transformations: `.map()` or `.filter()`
- Actions: `.count()` or `.first()`

# PySpark

- Python interface to Spark
- DataFrame abstraction
- Looks similar to Pandas

# PySpark: an example

```
# Load the dataset into athlete_events_spark first
```

```
(athlete_events_spark  
  .groupBy('Year')  
  .mean('Age')  
  .show())
```

```
SELECT year, AVG(age)  
FROM views.athlete_events  
GROUP BY year
```

# Let's practice!

INTRODUCTION TO DATA ENGINEERING

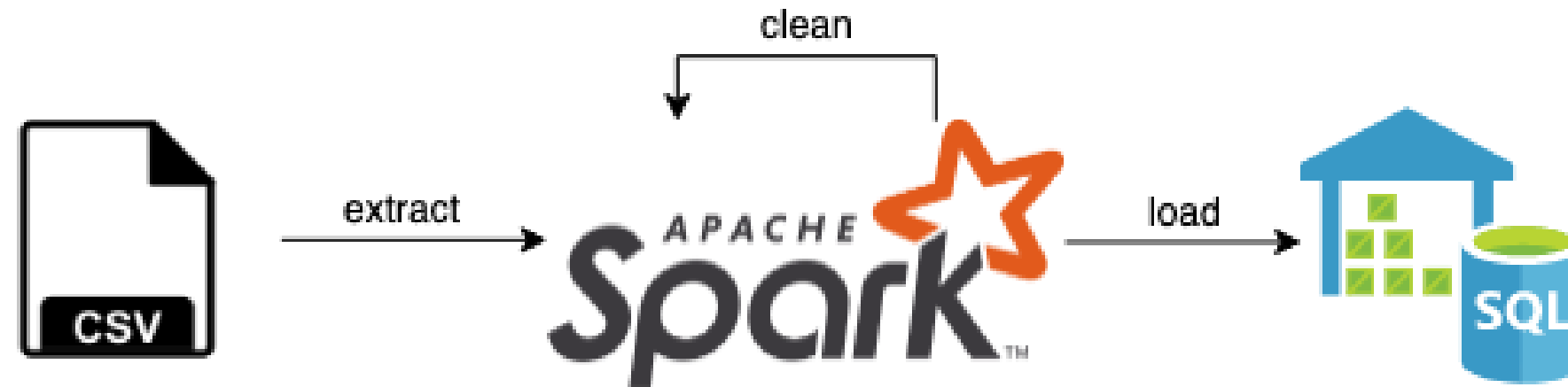
# Workflow scheduling frameworks

INTRODUCTION TO DATA ENGINEERING



**Vincent Vankrunkelsven**  
Data Engineer @ DataCamp

# An example pipeline



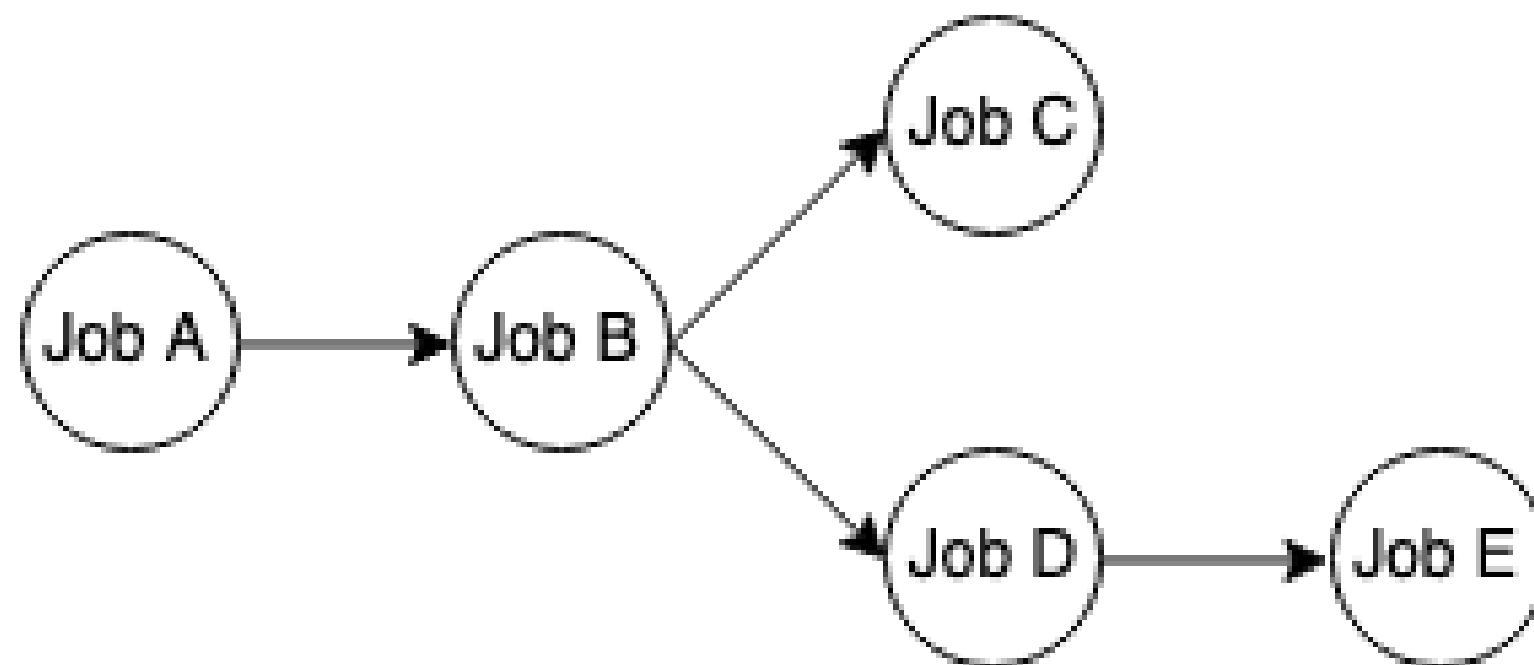
*How to schedule?*

- Manually
- `cron` scheduling tool
- What about dependencies?

# DAGs

*Directed Acyclic Graph*

- Set of nodes
- Directed edges
- No cycles



# The tools for the job

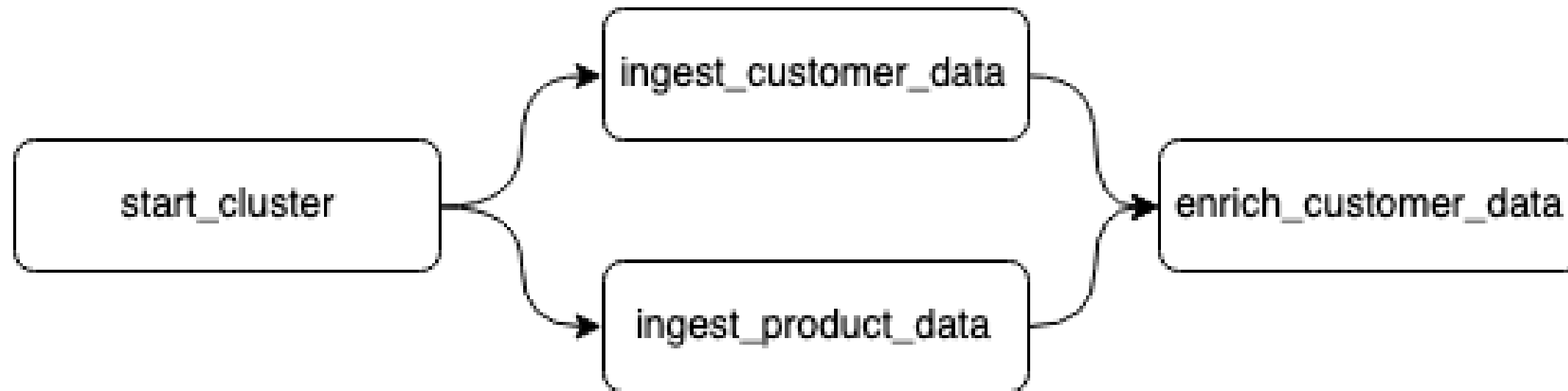
- Linux's `cron`
- Spotify's Luigi
- Apache Airflow





- Created at Airbnb
- DAGs
- Python

# Airflow: an example DAG



# Airflow: an example in code

```
# Create the DAG object
dag = DAG(dag_id="example_dag", ..., schedule_interval="0 * * * *")

# Define operations
start_cluster = StartClusterOperator(task_id="start_cluster", dag=dag)
ingest_customer_data = SparkJobOperator(task_id="ingest_customer_data", dag=dag)
ingest_product_data = SparkJobOperator(task_id="ingest_product_data", dag=dag)
enrich_customer_data = PythonOperator(task_id="enrich_customer_data", ..., dag = dag)

# Set up dependency flow
start_cluster.set_downstream(ingest_customer_data)
ingest_customer_data.set_downstream(enrich_customer_data)
ingest_product_data.set_downstream(enrich_customer_data)
```

# Let's practice!

INTRODUCTION TO DATA ENGINEERING