

Himalayan Peaks of Testing Data Pipelines

Ksenia Tomak, Dodo Engineering
Pasha Finkelshteyn, JetBrains

Who we are

What is Big Data

Who are DEs?

What is a pipeline?

Who needs pipelines

QA of pipeline

QA ?= QC

QA of pipeline

QA ≠ QC

QA is about processes and not only about software quality.

Pyramid of testing. Unit



Typical pipeline



Unit testing of pipeline

What may we test here?

A pipeline should transform data correctly!

Correctness is a business term

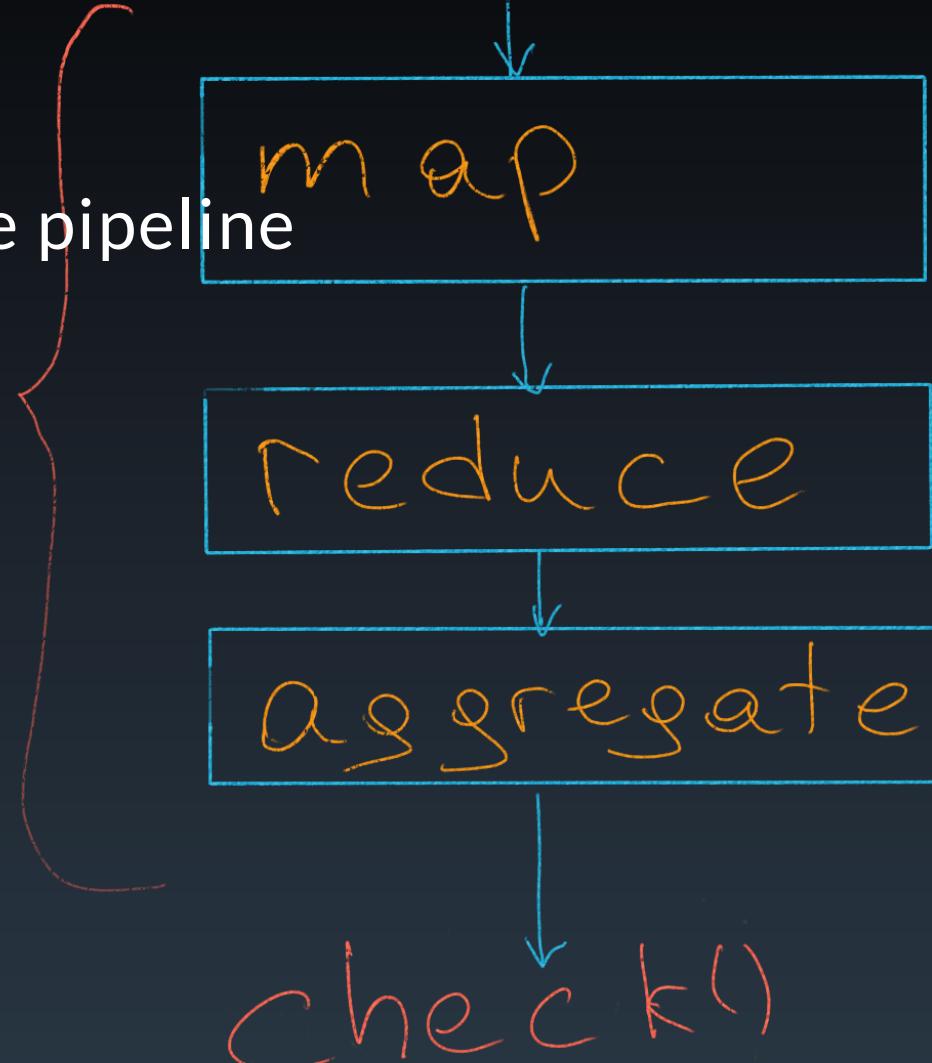
fake data

Let's paste fakes!

Fake/mock input data

Reference data at the end of the pipeline

Separate
Function

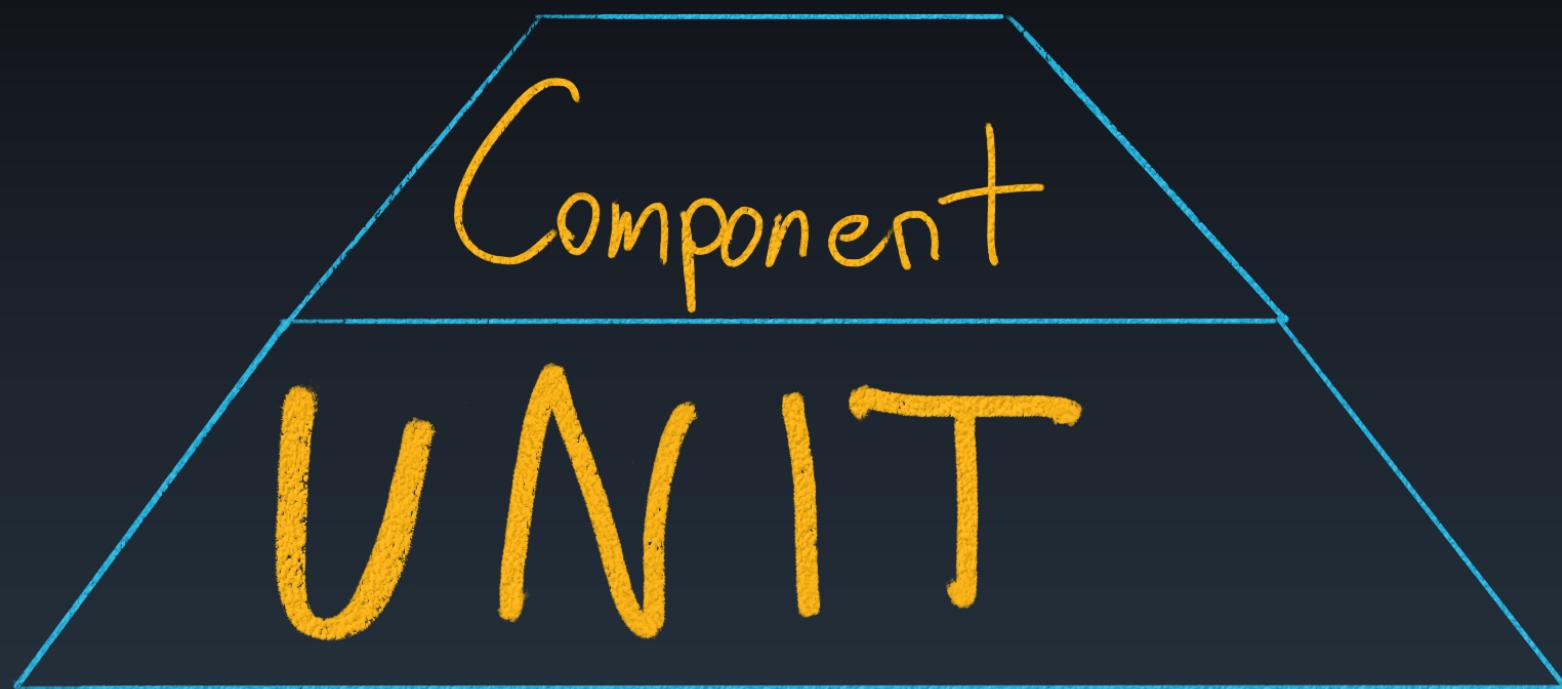


Tools

[holdenk/spark-testing-base](#) ← Tools to run tests

[MrPowers/spark-daria](#) ← tools to easily create test data

Component testing

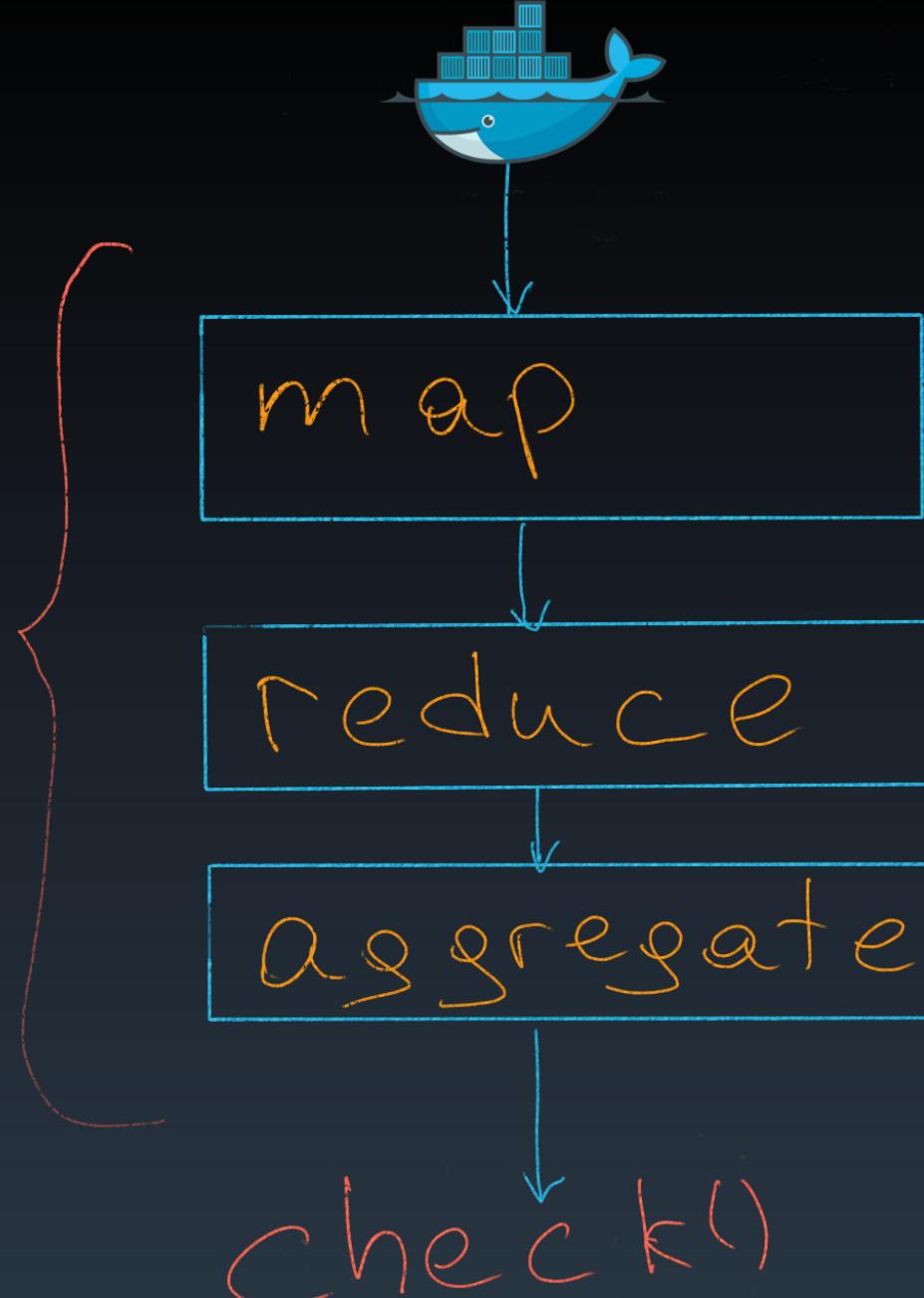




TEST CONTAINERS

TestContainers

Separate
Function



TestContainers

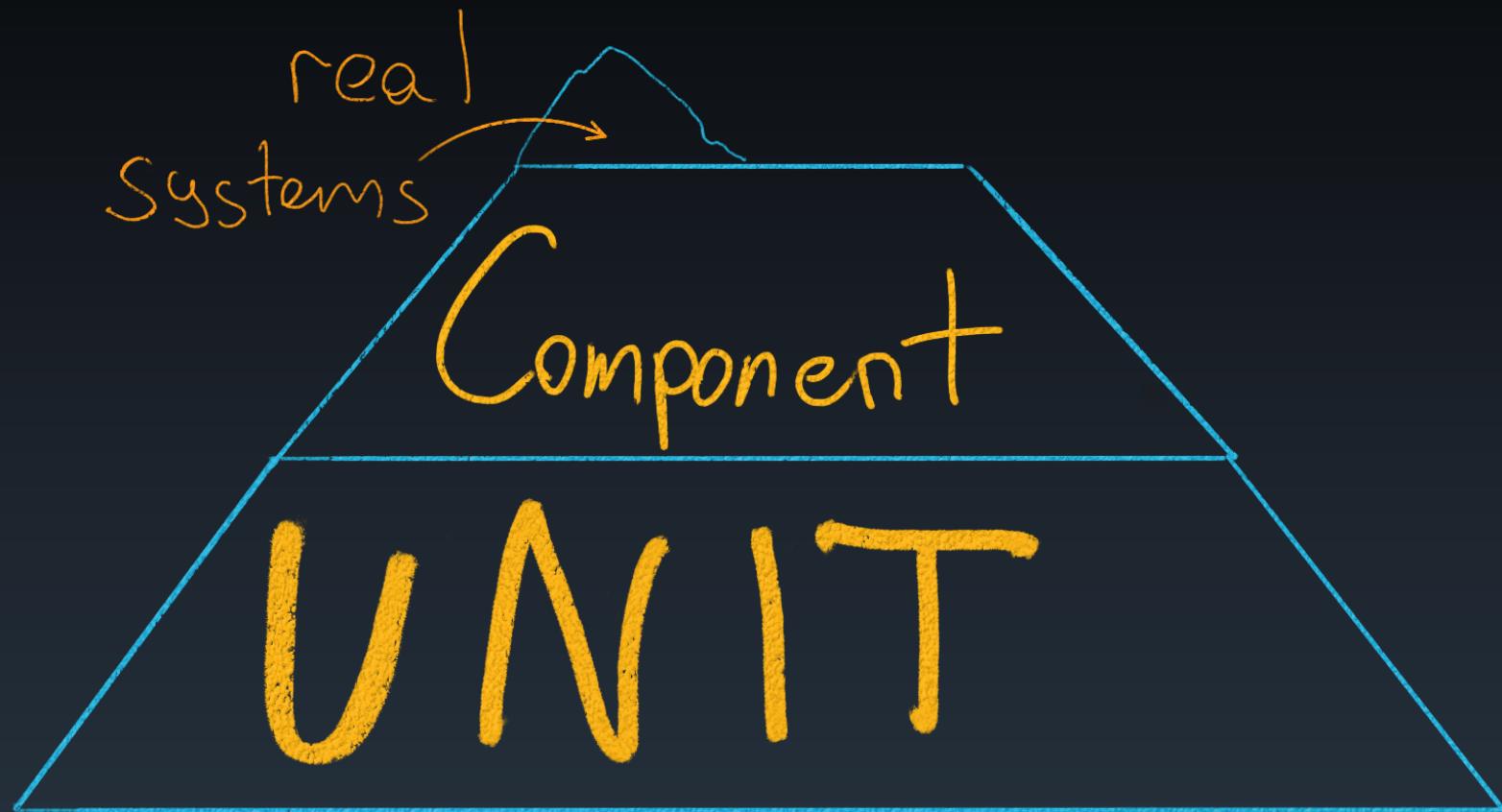
Supported languages:

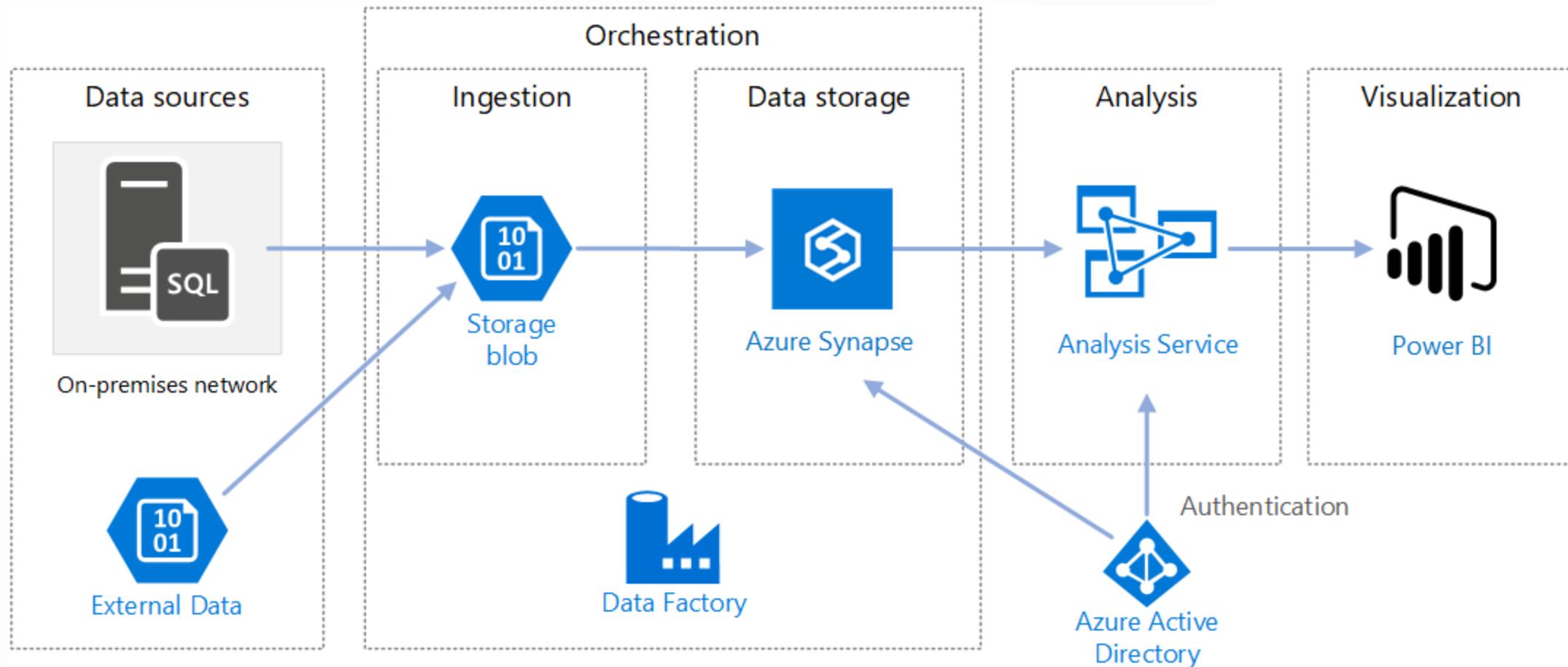
- Java (and compatibles: Scala, Kotlin, etc.)
- Python
- Go
- Node.js
- Rust
- .NET

Test Containers

```
import sqlalchemy
from testcontainers.mysql import MySqlContainer

with MySqlContainer('mysql:5.7.17') as mysql:
    engine = sqlalchemy.create_engine(mysql.get_connection_url())
    version, = engine.execute("select version()").fetchone()
    print(version) # 5.7.17
```

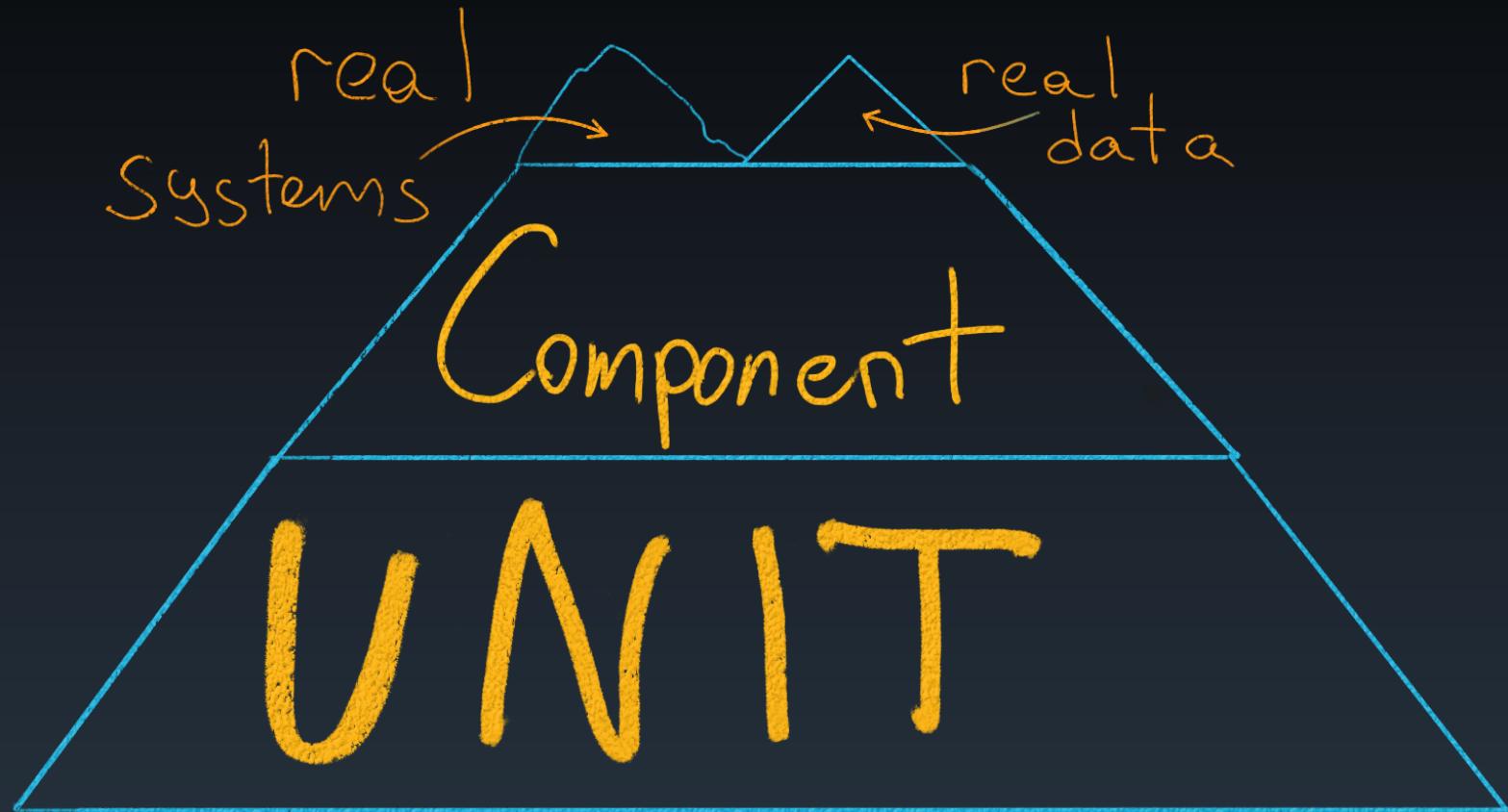




Real systems

Why are component tests not enough?

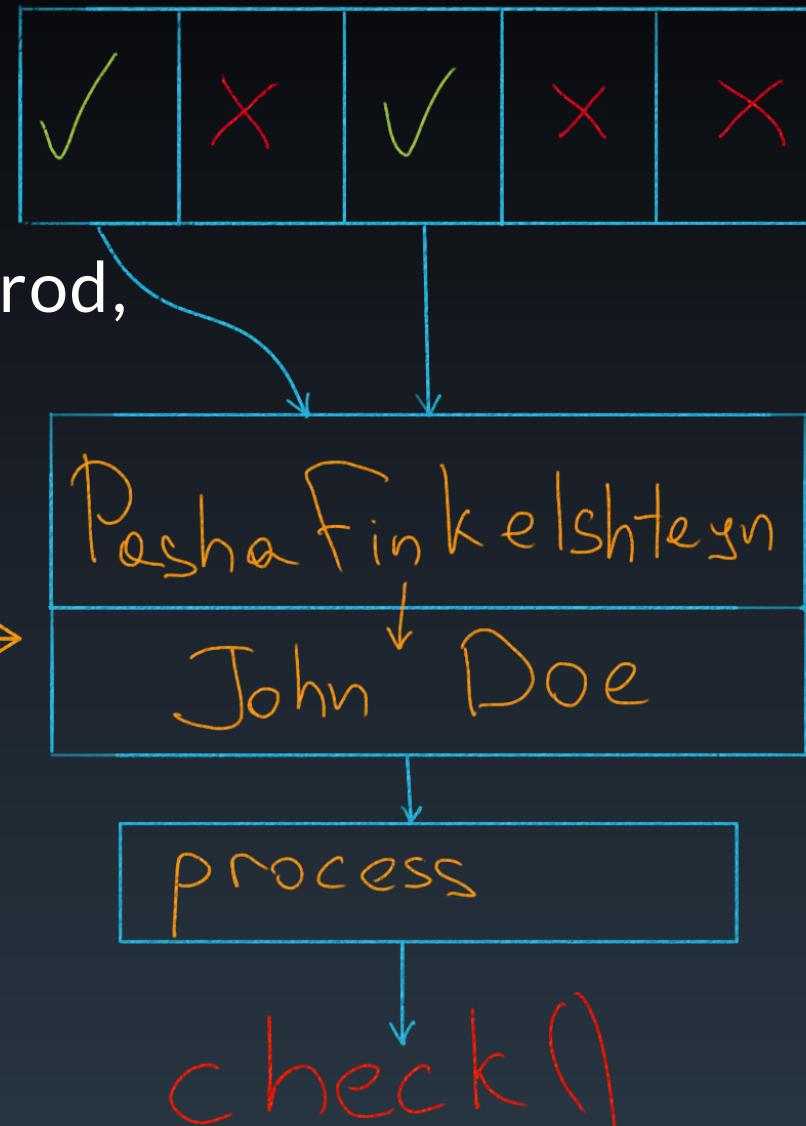
- vendor lock tools (DB, processing, etc.)
- external error handling



Real data

Get data samples from prod,
anonymize it

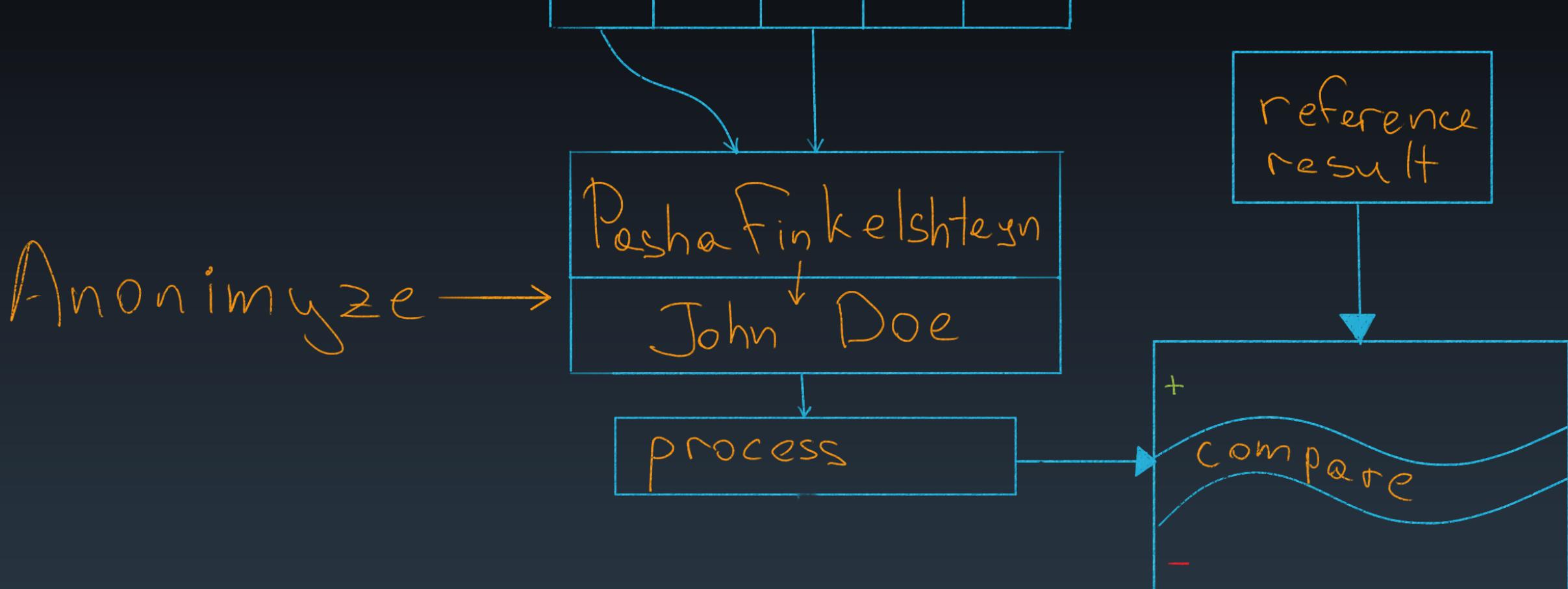
Anonymize →



Real data

Deploy full data backup on stage env,
anonymize it 😷

Compare to reference



In usual testing you won't trust your code

**In pipeline testing you won't trust
both your code and your data**

Real data expectations

Test:

-  no data
-  valid data
-  invalid data
-  illegal data format

Real data expectations. Tools:

- [great expectations](#),
- [Deequ](#)

```
from pyspark.sql.types import Row, StructType
from datetime import datetime

schema = {
    "type": "struct",
    "fields": [
        {"name": "Id", "type": "long", "nullable": False, "metadata": {}},
        {"name": "SaleDate", "type": "timestamp", "nullable": False, "metadata": {}},
        {"name": "Country", "type": "string", "nullable": False, "metadata": {}},
    ]
}

table_rows = [
    Row(1, datetime(2021, 1, 1, 10, 0, 0), "RU"),
    Row(2, datetime(1000, 1, 1, 10, 0, 0), "KZ"),
    Row(2, datetime(2018, 1, 1, 10, 0, 0), "AU"),
    Row(2, datetime(2019, 1, 1, 10, 0, 0), ""),
]

sample_df = spark.createDataFrame(table_rows, StructType.fromJson(schema))
```

Great expectations

```
from great_expectations.dataset.sparkdf_dataset import SparkDFDataset  
  
ge_sample_df = SparkDFDataset(sample_df)  
ge_sample_df.expect_column_values_to_be_in_set("Country", ["RU", "KZ"])
```

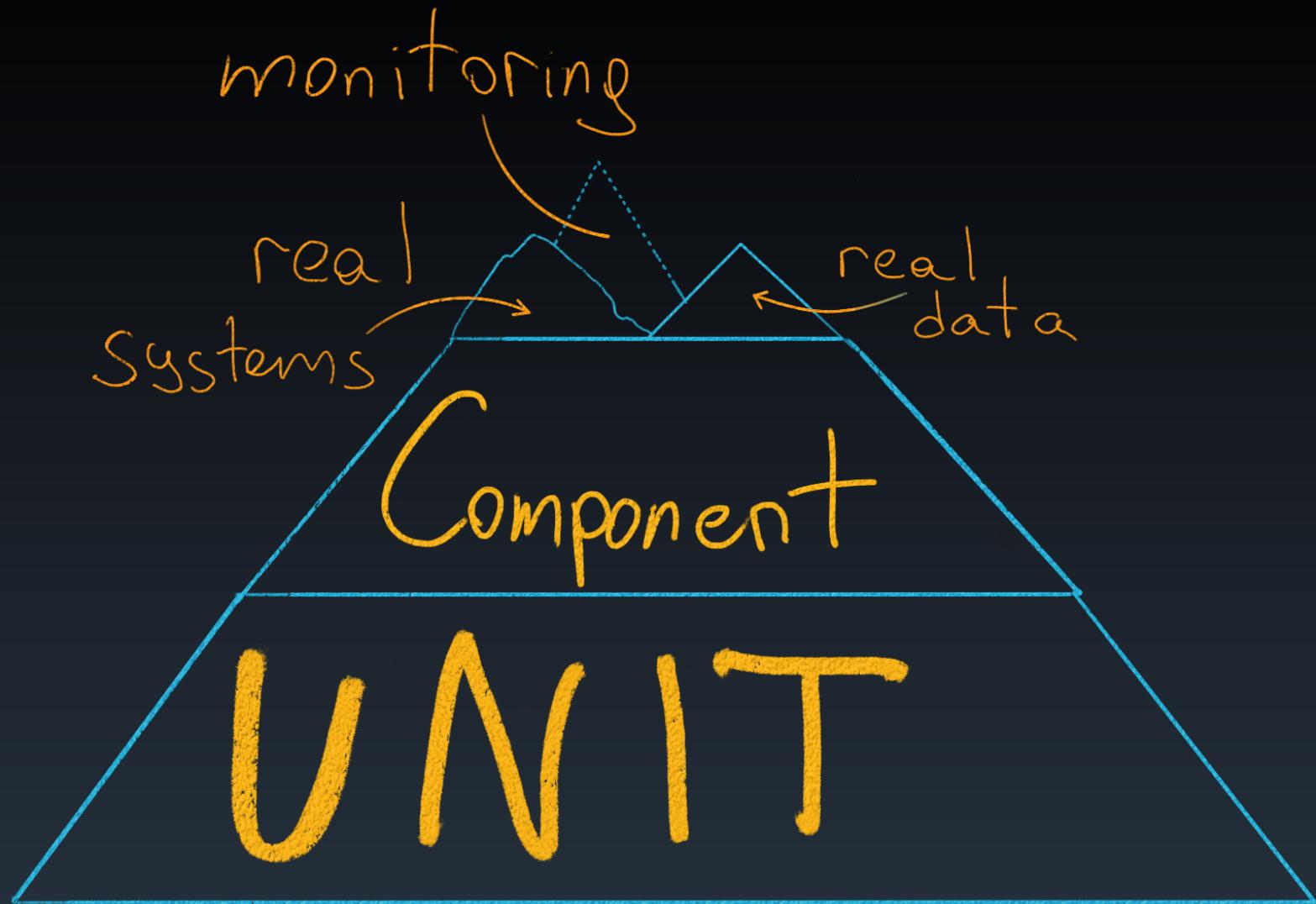
Great expectations

```
"result": {  
    "element_count": 4,  
    "unexpected_count": 2,  
    "unexpected_percent": 50.0,  
    "partial_unexpected_list": [ "AU", "" ]  
},  
"success": false,  
"expectation_config": {  
    "kwargs": {  
        "column": "Country",  
        "value_set": [ "RU", "KZ" ]  
    }  
}
```

Python Deequ

```
# No Spark 3.0 support yet
from pydeequ.checks import *
from pydeequ.verification import *

check = Check(spark, CheckLevel.Warning, "Country Check")
checkResult =(
    VerificationSuite(spark)
        .onData(sample_df)
        .addCheck(
            check.isContainedIn("Country", [ "RU", "KZ"]))
        .run()
)
checkResult_df = VerificationResult.checkResultsAsDataFrame(spark, checkResult)
checkResult_df.show()
```



Monitoring

Why?

- The only REAL testing is production
- Data tends to change over time

Monitoring

What?

- data volumes
- counters
- time
- dead letter queue monitoring
- service health

Monitoring

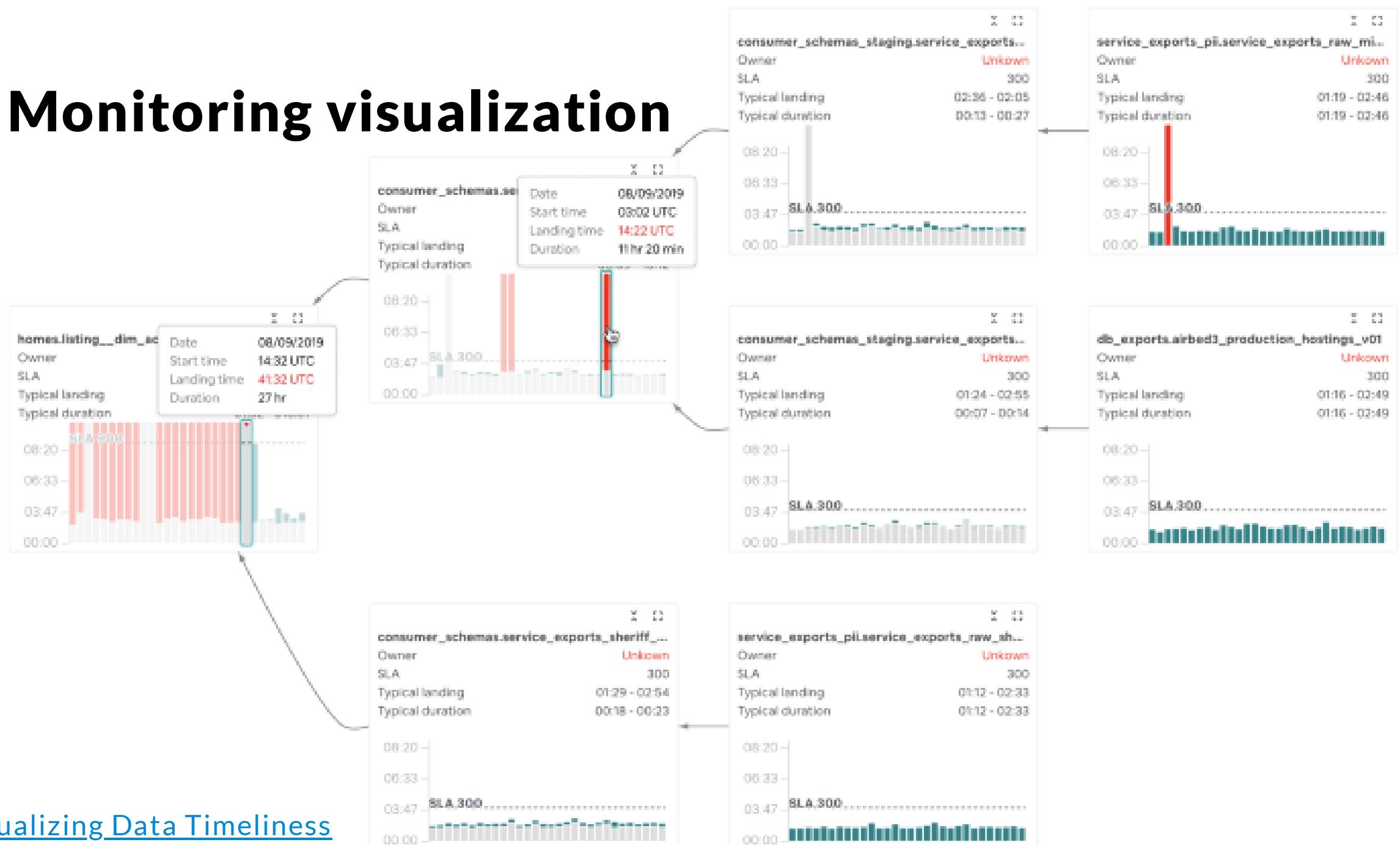
How?

- use Listeners
- use data aggregations

Data pipelines is always DAG

Monitoring should visualize it

Monitoring visualization

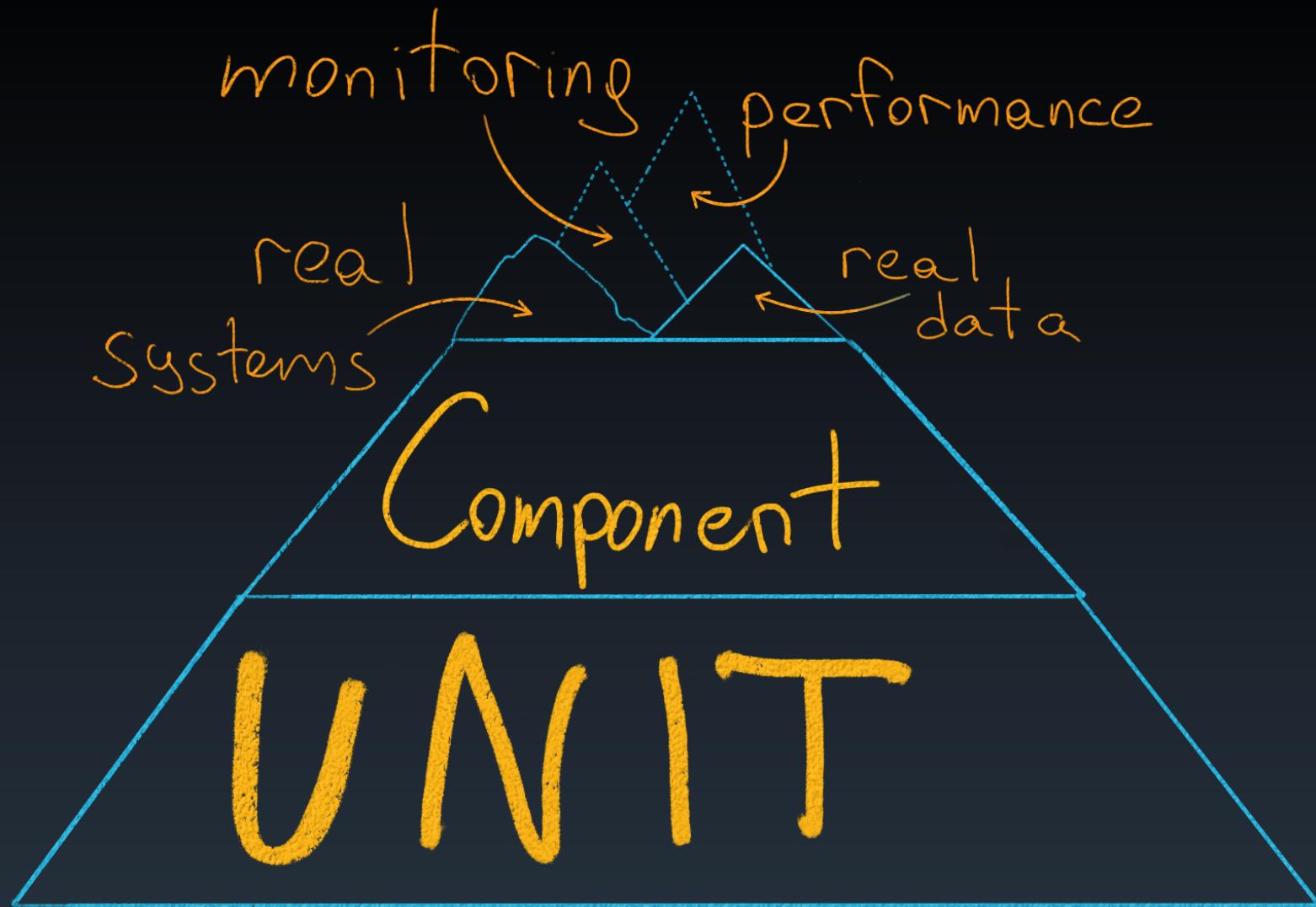


End-to-End tests

Compare with reports, old DWH

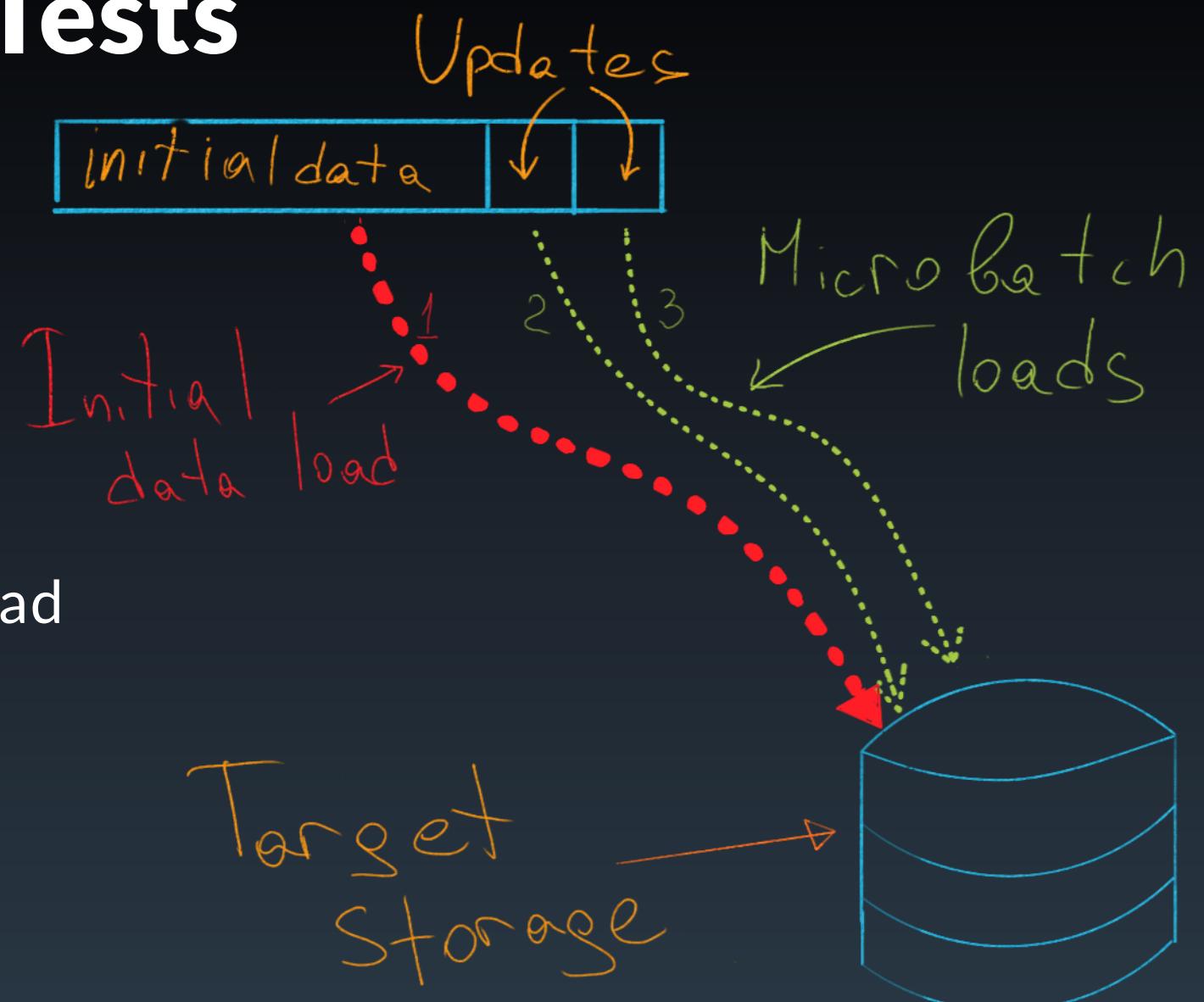
Multiple dimensions:

- data
- data latency
- performance, scalability



Performance Tests

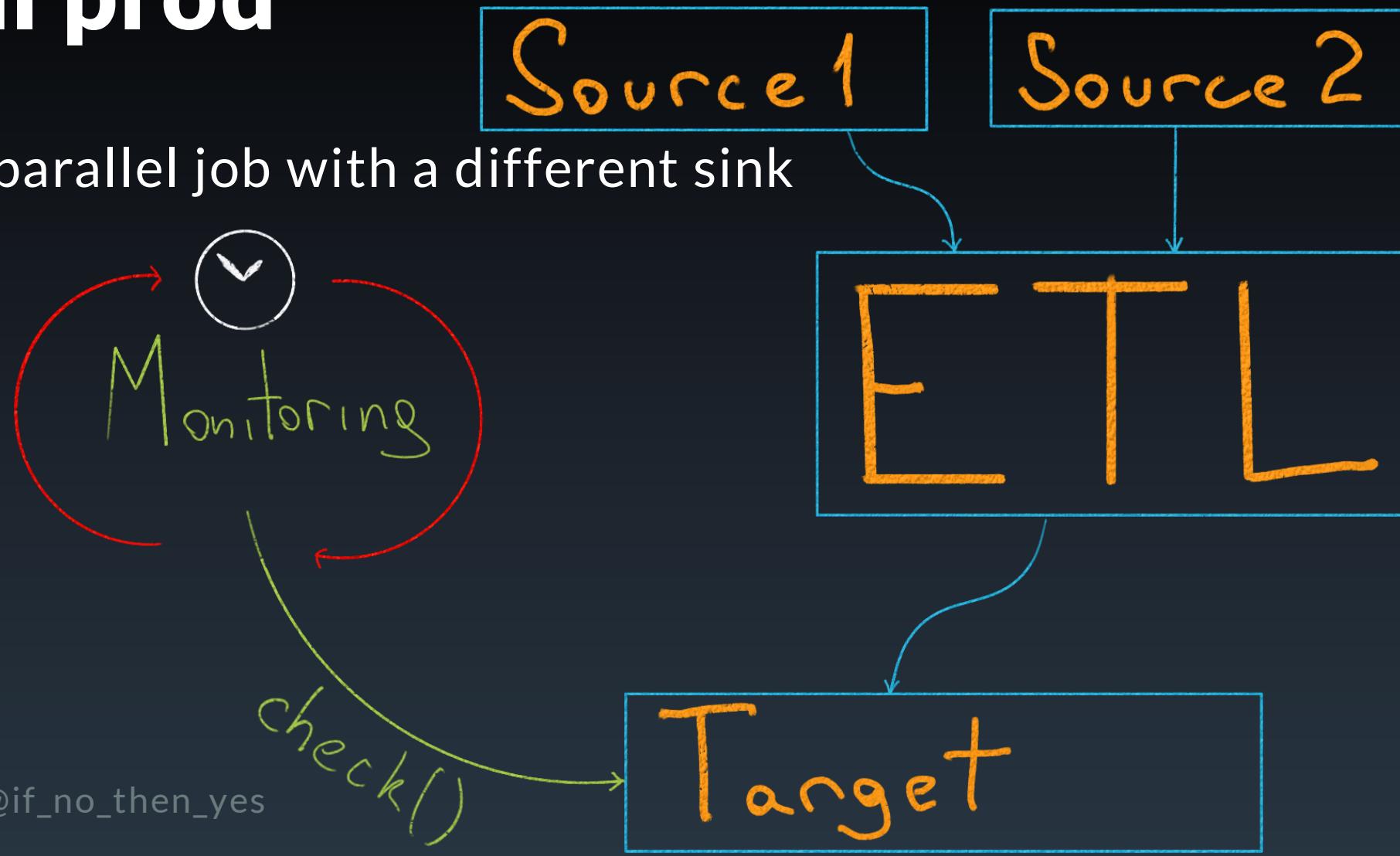
- start with SLA
- test your initial data load



How to apply?

Real prod

Run a parallel job with a different sink



Using production data for testing in a post GDPR world