Himalayan Peaks of Testing Data Pipelines

Ksenia Tomak, Dodo Engineering
Pasha Finkelshteyn, JetBrains

Ksenia Tomak

- Tech Lead, Dodo Engineering
- @if_no_then_yes

Industrial IoT, DE, Storages

DE or DIE

Pasha Finkelshteyn

Developer 🥑 for Big Data @ JetBrains

@asm0di0

DE or DIE

What is Big Data

- Doesn't fit the single node (or Excel)
- Maybe scaled when growing
- Enough data to make reliable business solutions

Who are DEs

Plumber of data

Data is produces by

- sensors
- clickstreams
- etc.

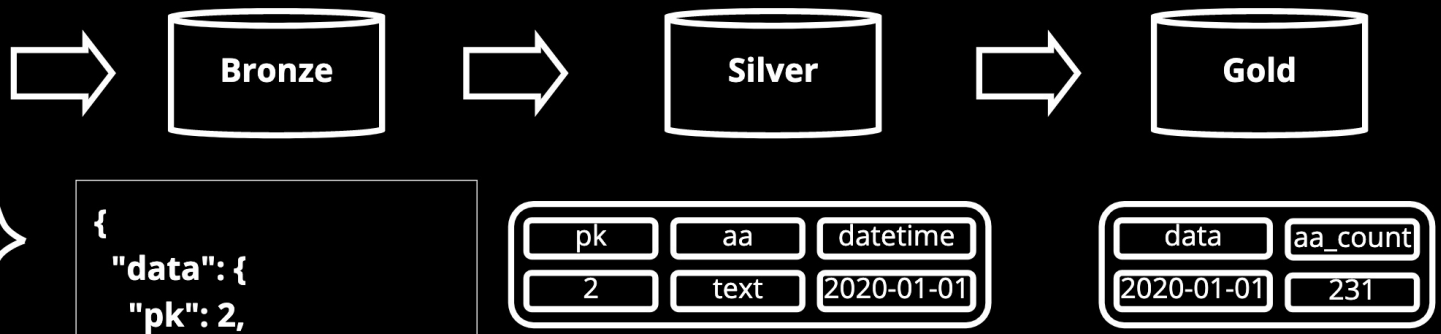
Data sinks

- HDFS
- S3, Azure Blob Storage
- etc.

What is a data pipeline?

Data processing

Data lake?




```
{
  "data": {
    "pk": 2,
    "aa": "text"
  },
  "op": "c",
  "ts_ms":
"1580390884335"
}
```

Who needs pipelines

- Data Scientists
- Data Analytics
- Marketing
- PO

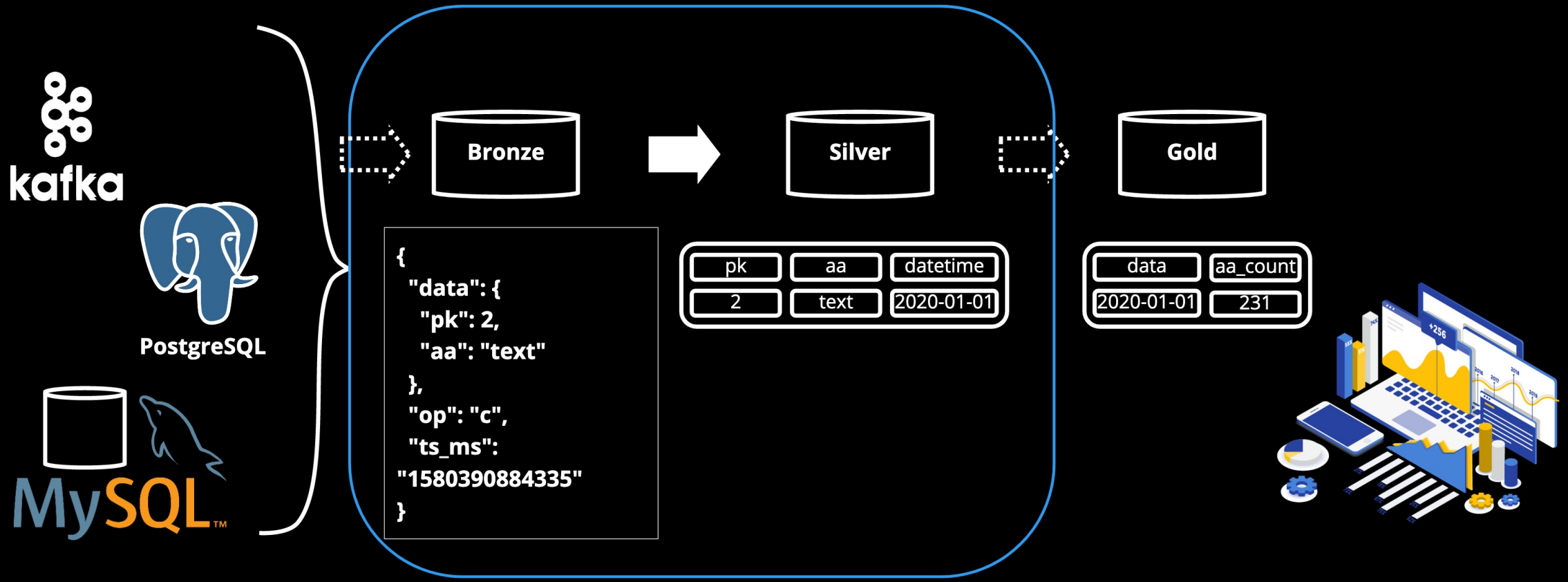
QA ?= QC

QA ≠ QC

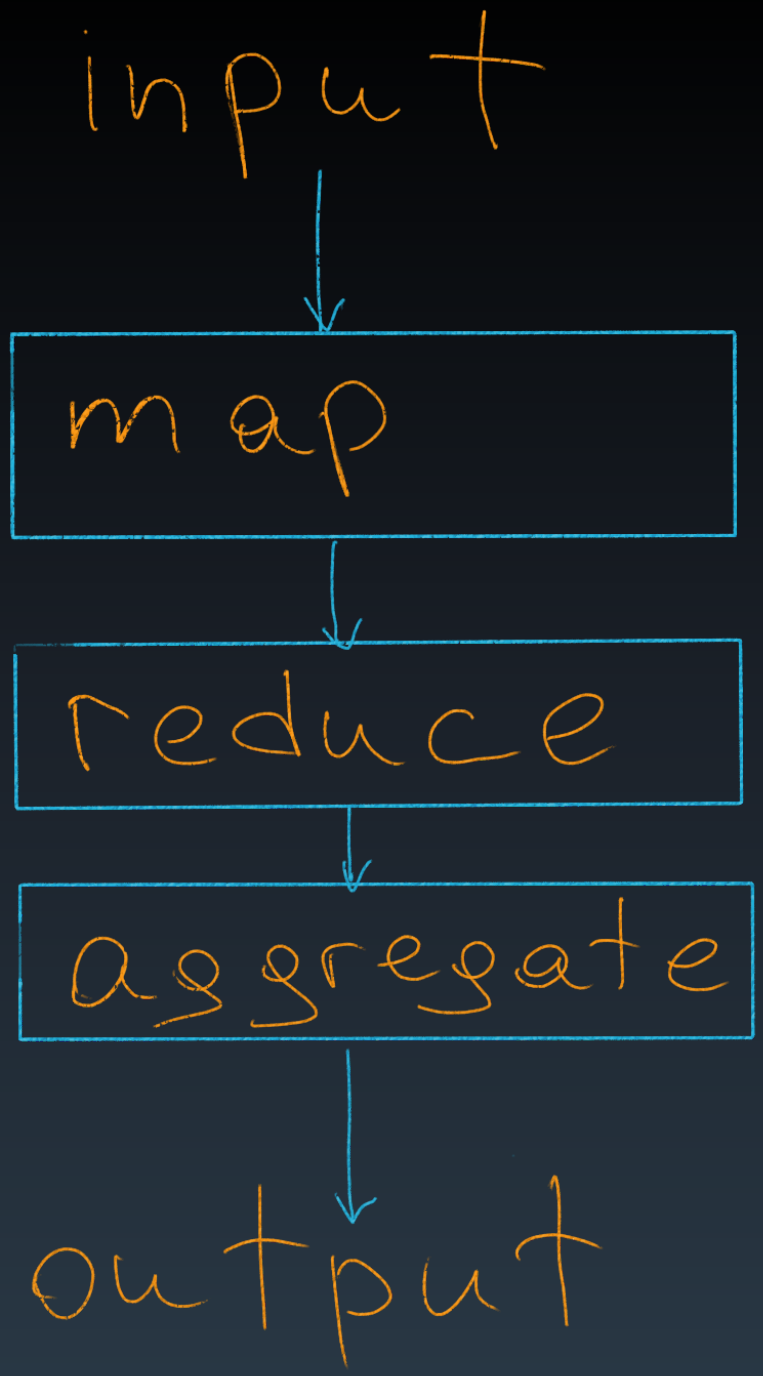
QA is about processes and not only about software quality.

Pyramid of testing. Unit

Bronze → Silver pipeline



Typical pipeline



Typical pipeline

```python
from pyspark.sql import DataFrame, functions as F
from pyspark.sql.types import *

data_type = StructType([
    StructField("pk", LongType(), False),
    StructField("aa", StringType(), False)
  ])

transformed_data = (
  sample_df
  .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
  .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
  .selectExpr("parsed_struct.*")
)
```

Typical pipeline

```python
from pyspark.sql import DataFrame, functions as F
from pyspark.sql.types import *

data_type = StructType([
    StructField("pk", LongType(), False),
    StructField("aa", StringType(), False)
  ])

transformed_data = (
  sample_df
  .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
  .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
  .selectExpr("parsed_struct.*")
)
```

Typical pipeline

```python
from pyspark.sql import DataFrame, functions as F
from pyspark.sql.types import *

data_type = StructType([
    StructField("pk", LongType(), False),
    StructField("aa", StringType(), False)
  ])

transformed_data = (
  sample_df
  .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
  .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
  .selectExpr("parsed_struct.*")
)
```

Typical pipeline

```python
from pyspark.sql import DataFrame, functions as F
from pyspark.sql.types import *

data_type = StructType([
    StructField("pk", LongType(), False),
    StructField("aa", StringType(), False)
  ])

transformed_data = (
  sample_df
  .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
  .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
  .selectExpr("parsed_struct.*")
)
```

Typical pipeline

```python
from pyspark.sql import DataFrame, functions as F
from pyspark.sql.types import *

data_type = StructType([
    StructField("pk", LongType(), False),
    StructField("aa", StringType(), False)
  ])

transformed_data = (
  sample_df
  .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
  .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
  .selectExpr("parsed_struct.*")
)
```

Unit testing of pipeline

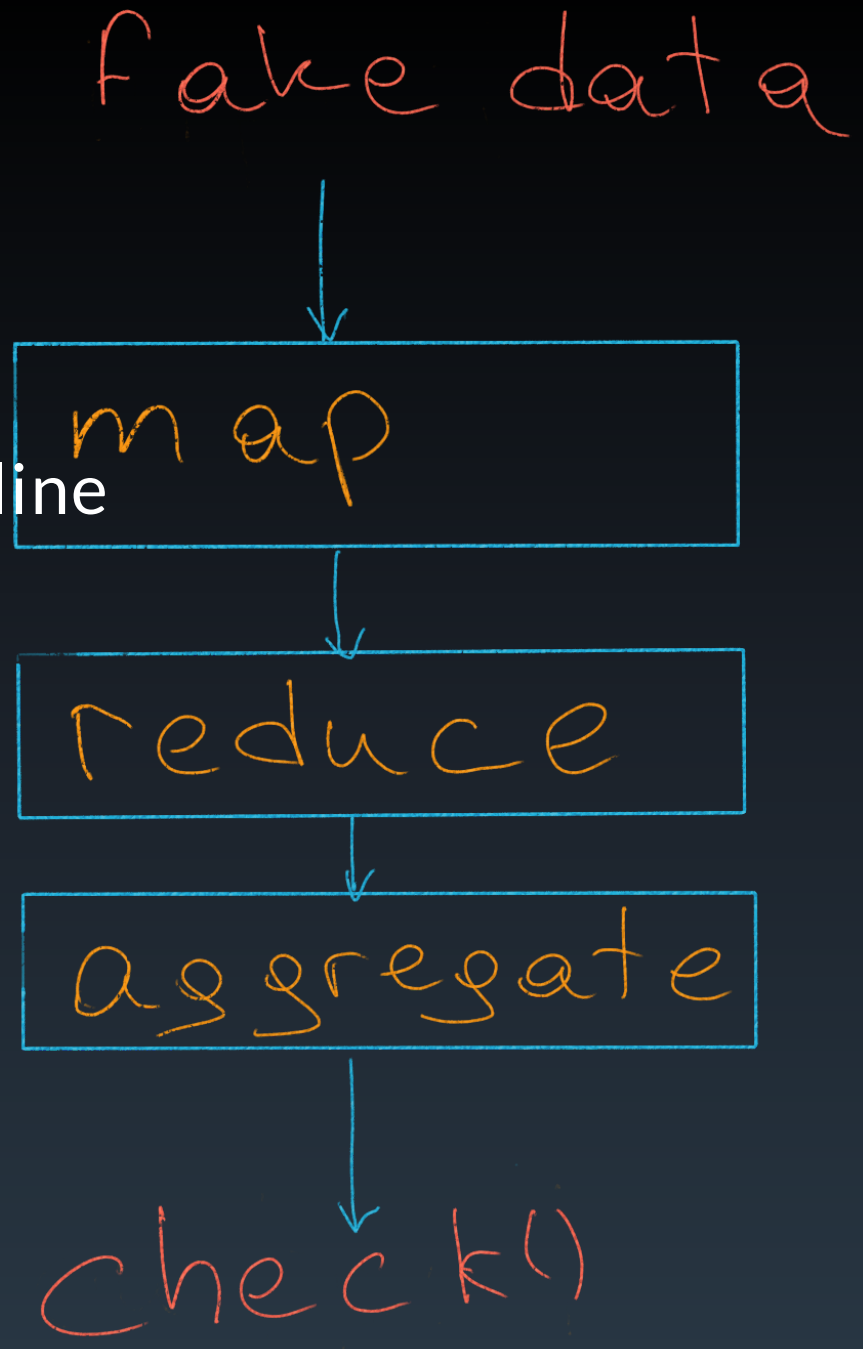
What may we test here?

A pipeline should transform data correctly!

Correctness is a business term

Let's paste fakes!

Fake input data
Reference data at the end of the pipeline



Tools

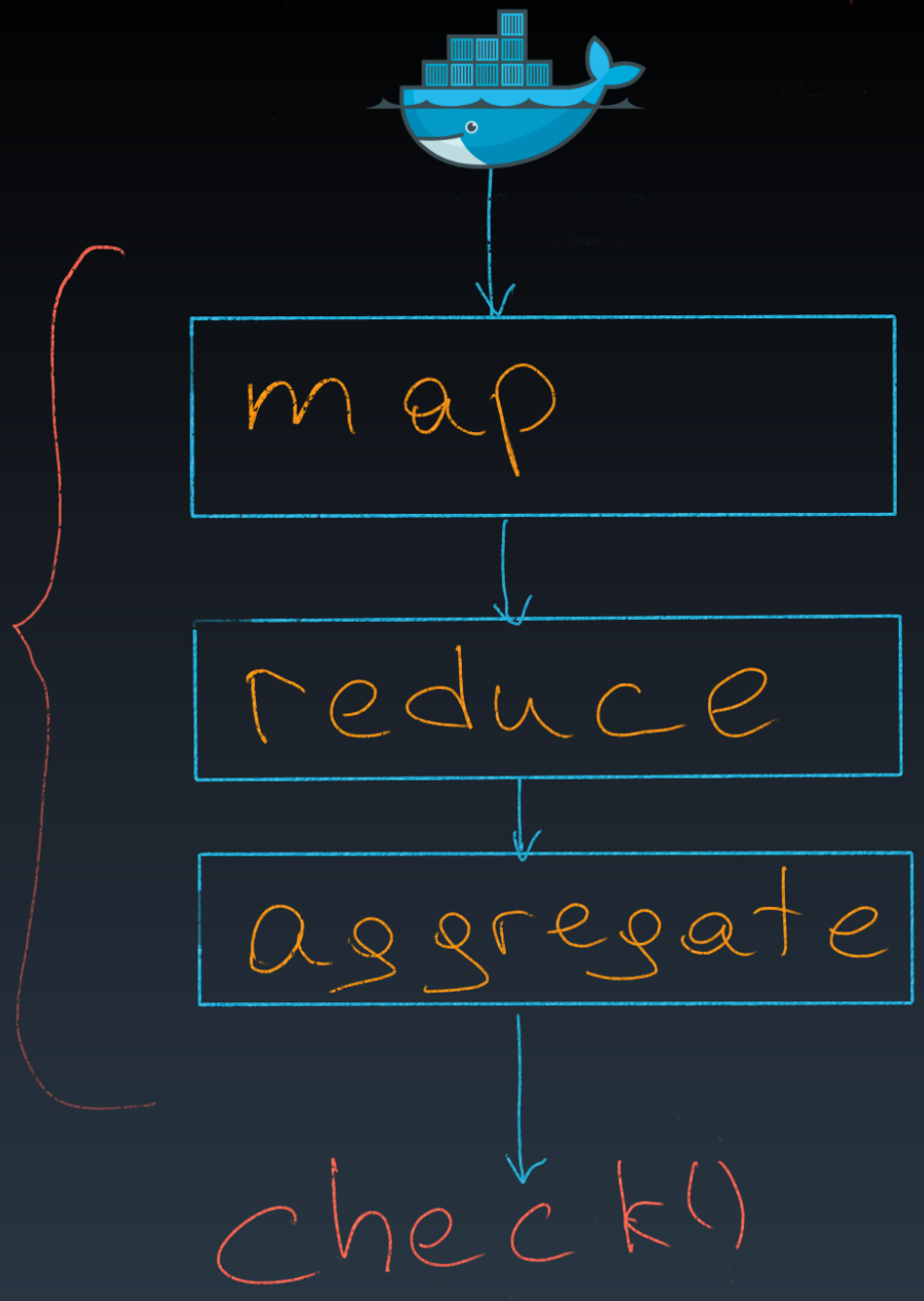
holdenk/spark-testing-base ← Tools to run tests
MrPowers/spark-daria ← tools to easily create test data

Component testing

TESTCONTAINERS

TestContainers

Separate Function



TestContainers

Supported languages:

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

Test Containers

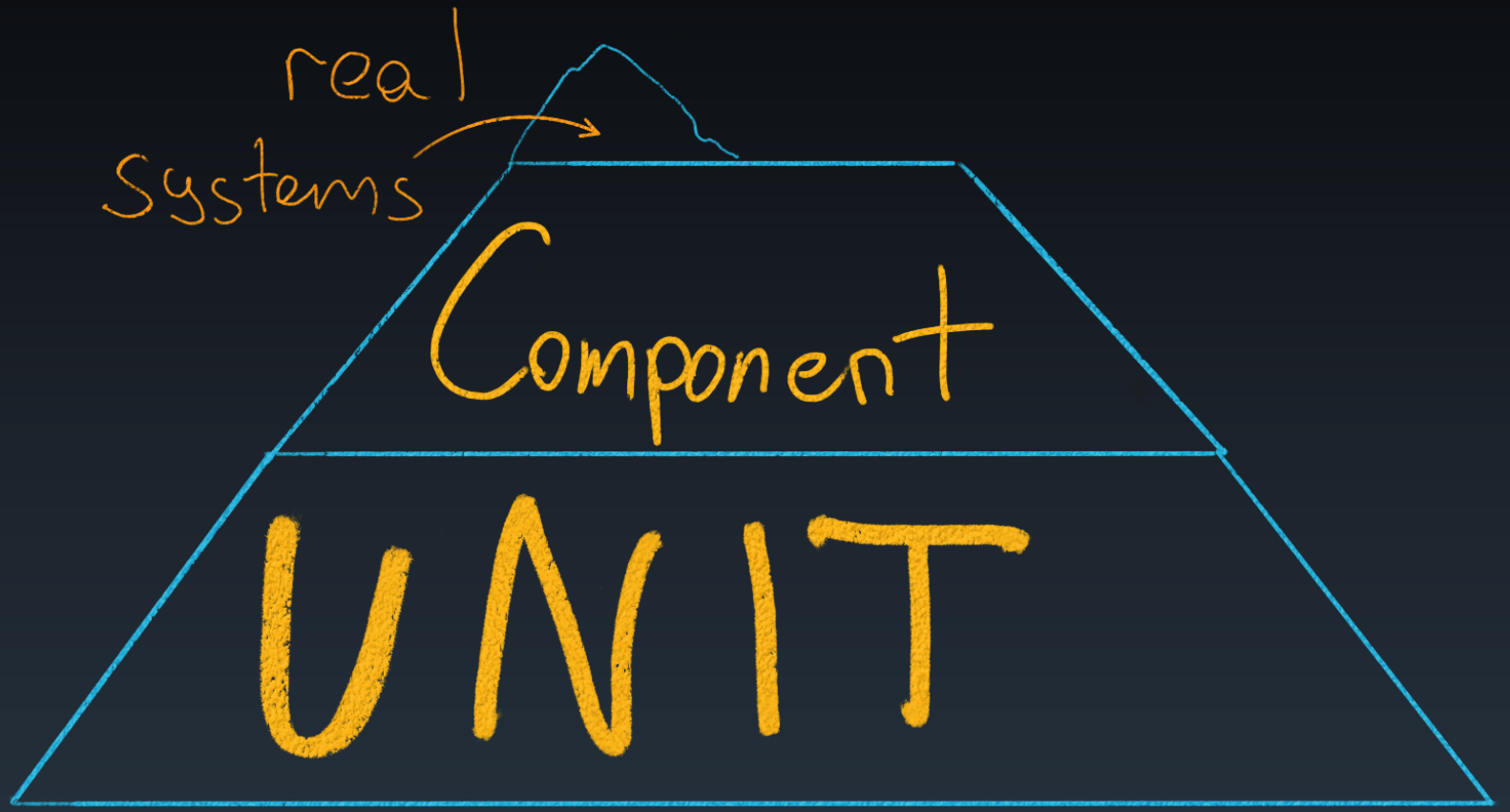
```python
import sqlalchemy
from testcontainers.mysql import MySqlContainer

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
```

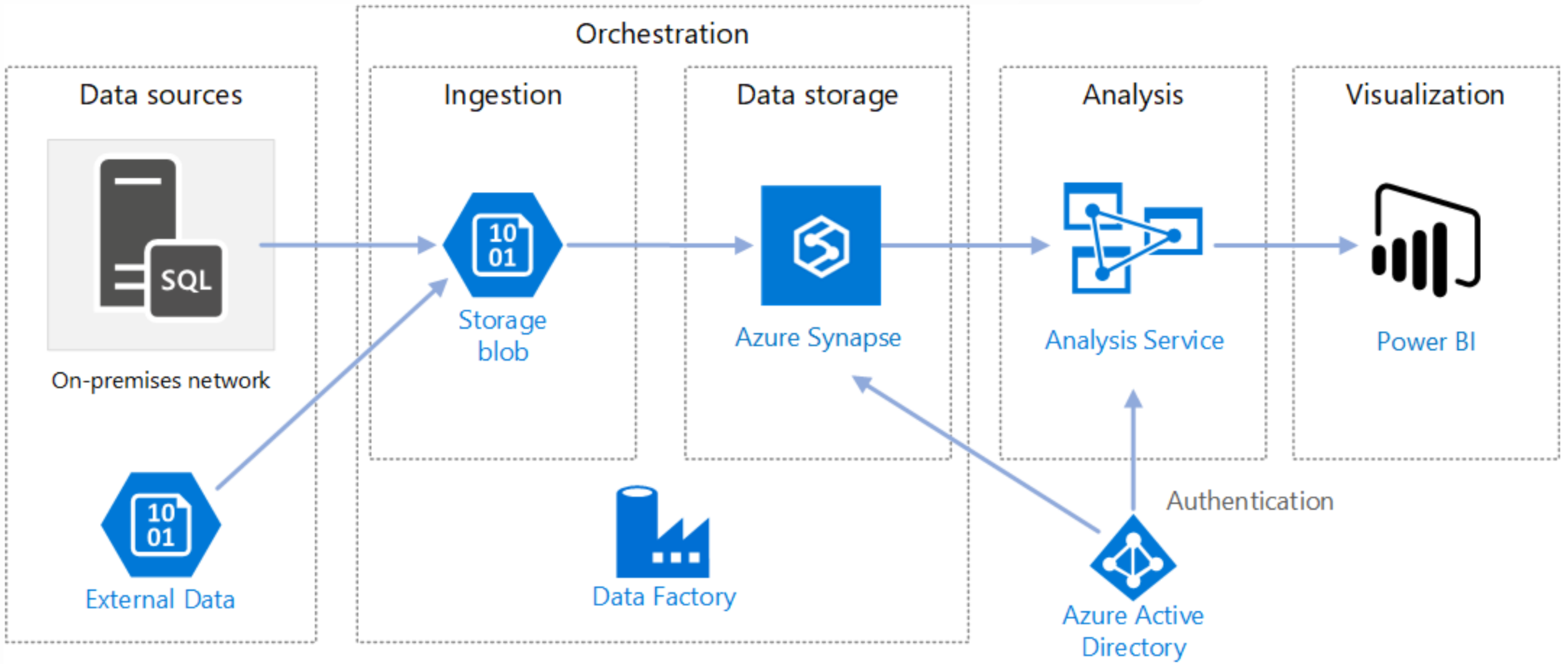
Test Containers

```python
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
```

Test Containers

```python
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
```

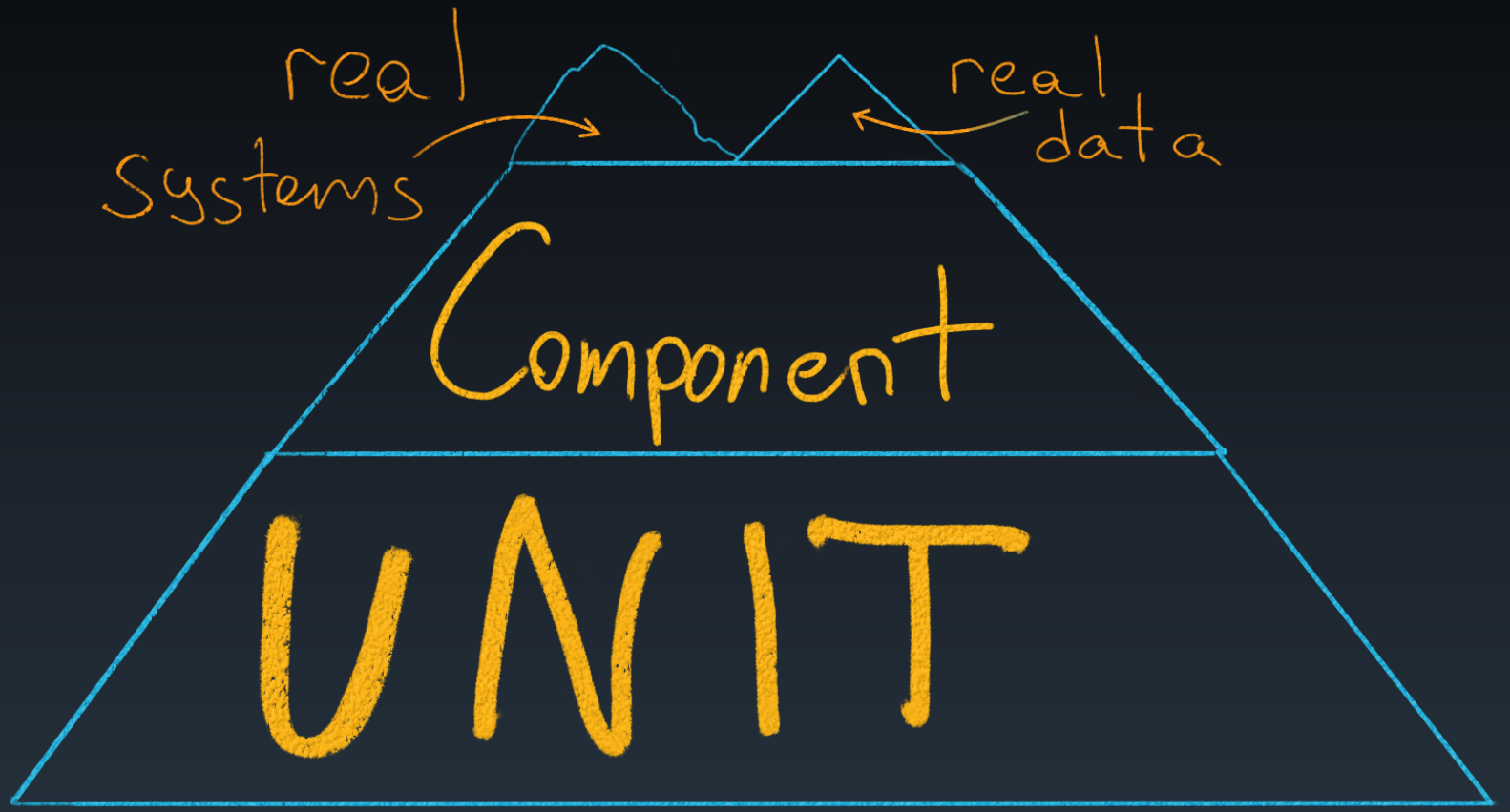

real
Systems
Component
UNIT


Orchestration
Data sources
Ingestion
Data storage
Analysis
Visualization
SQL
On-premises network
10
01
Storage
blob
Azure Synapse
Analysis Service
Power BI
10
01
External Data
Data Factory
Authentication
Azure Active
Directory

Real systems

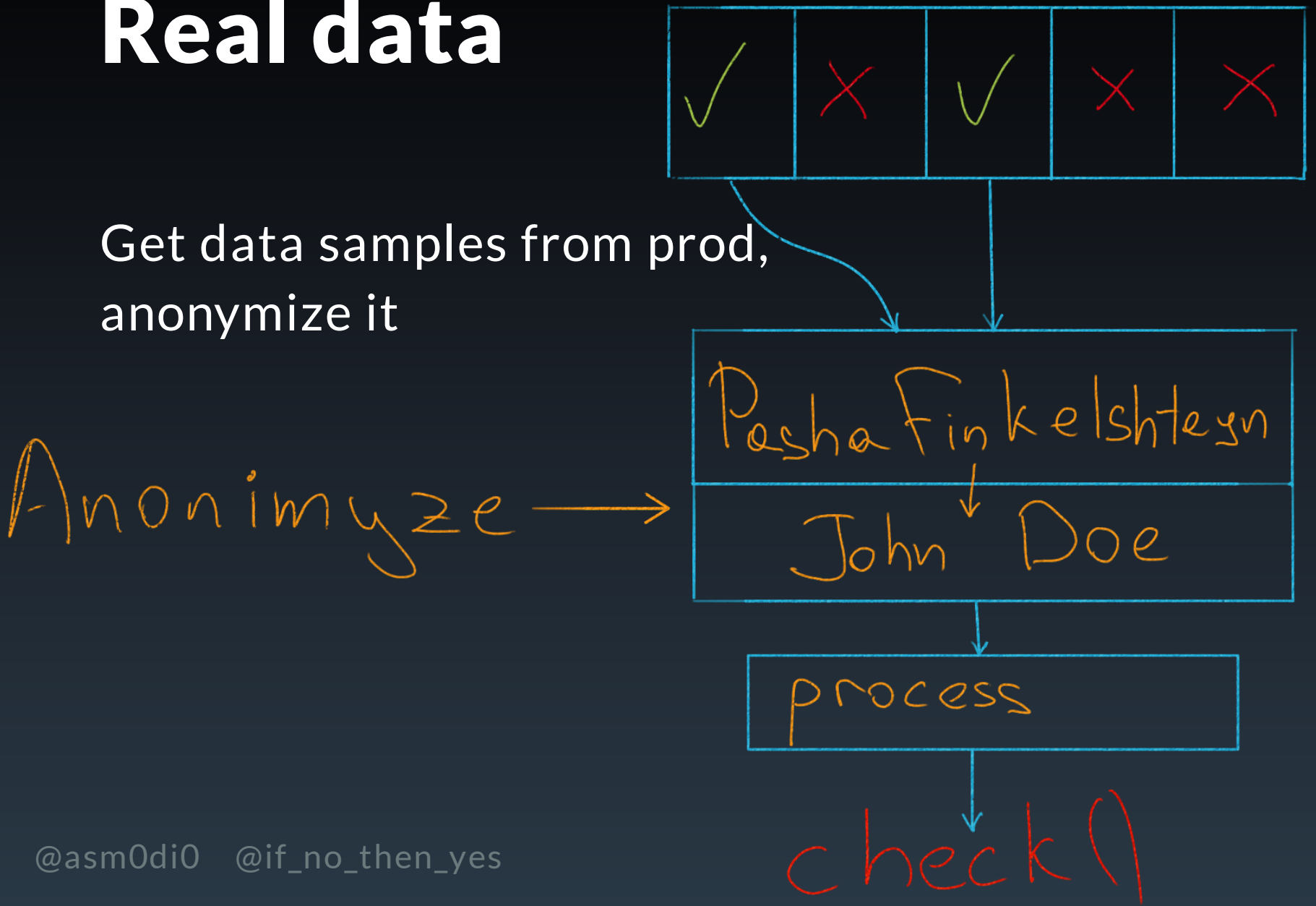
Why are component tests not enough?

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

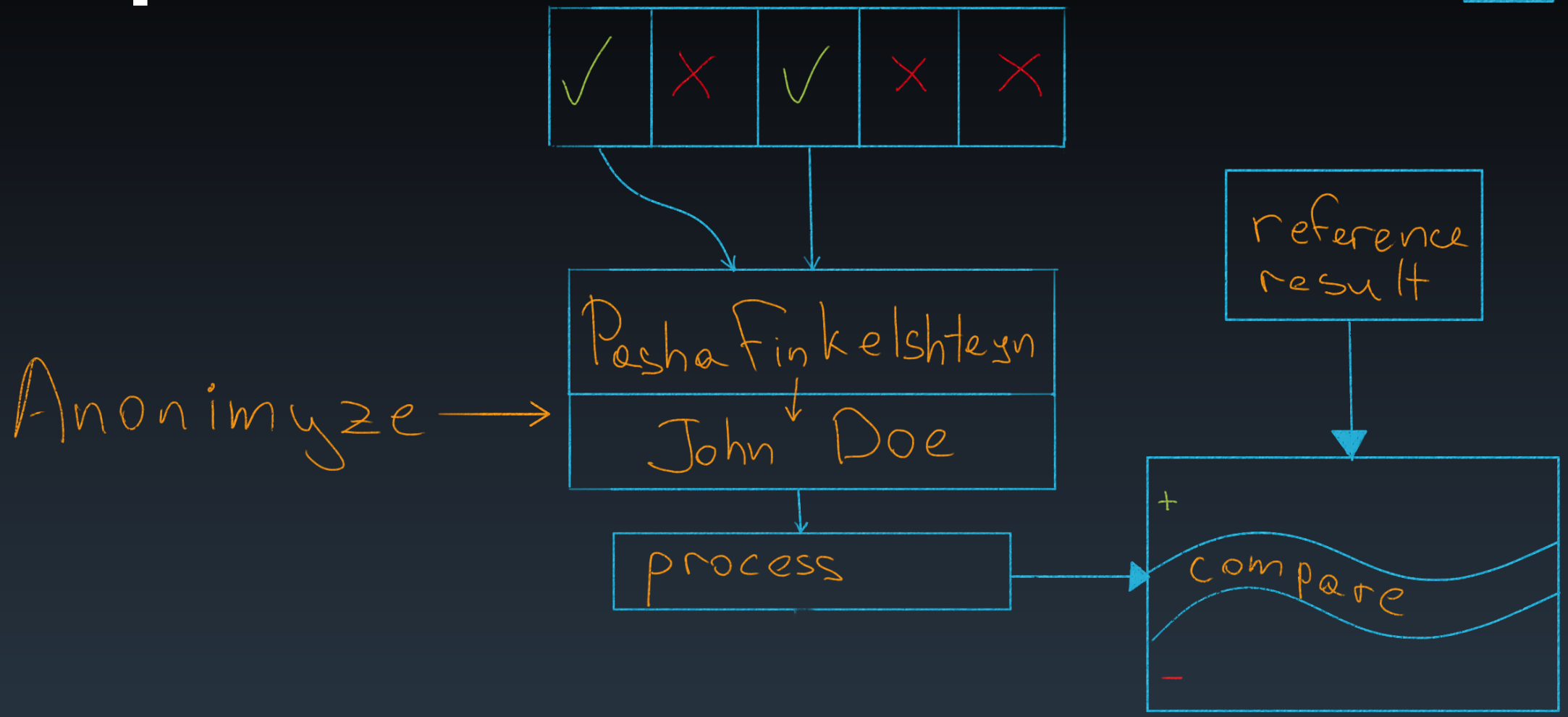

real
Systems
real
data
Component
UNIT

Real data

Get data samples from prod,
anonymize it



Compare to reference



Comparison example

gender	reference	id	match
m	m	1	TRUE
f	c	2	FALSE
u	u	3	TRUE
c	c	4	TRUE
m	f	5	FALSE

Real data

Deploy full data backup on stage env, anonymize it 🤑

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

Real data expectations

- profilers
- constraint suggestions
- constraint verification
- metrics
- metrics strores

```python
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))
```

```python
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))
```

```python
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))
```

```python
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

```
1 from great_expectations.dataset.sparkdf_dataset import SparkDFDataset
2
3 ge_sample_df = SparkDFDataset(sample_df)
4 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

```
check = Check(spark, CheckLevel.Warning, 'Review Check')

checkResult = VerificationSuite(spark)
  .onData(sample_df)
  .addCheck(check
            .isComplete('Country')
            .isContainedIn('Country', ['RU', 'KZ']))
  .run()

checkResult_df = VerificationResult.checkResultsAsDataFrame(spark, checkResult)
checkResult_df.show()
```

Testing data quality at scale with PyDeequ

PyDeequ documentation

Python Deequ

```python
check = Check(spark, CheckLevel.Warning, 'Review Check')

checkResult = VerificationSuite(spark)
  .onData(sample_df)
  .addCheck(check
            .isComplete('Country')
            .isContainedIn('Country', ['RU', 'KZ']))
  .run()

checkResult_df = VerificationResult.checkResultsAsDataFrame(spark, checkResult)
checkResult_df.show()
```

Testing data quality at scale with PyDeequ

PyDeequ documentation

Python Deequ

```
check = Check(spark, CheckLevel.Warning, 'Review Check')

checkResult = VerificationSuite(spark)
  .onData(sample_df)
  .addCheck(check
            .isComplete('Country')
            .isContainedIn('Country', ['RU', 'KZ']))
  .run()

checkResult_df = VerificationResult.checkResultsAsDataFrame(spark, checkResult)
checkResult_df.show()
```

Testing data quality at scale with PyDeequ

PyDeequ documentation

Python Deequ

```python
check = Check(spark, CheckLevel.Warning, 'Review Check')

checkResult = VerificationSuite(spark)
  .onData(sample_df)
  .addCheck(check
            .isComplete('Country')
            .isContainedIn('Country', ['RU', 'KZ']))
  .run()

checkResult_df = VerificationResult.checkResultsAsDataFrame(spark, checkResult)
checkResult_df.show()
```

Testing data quality at scale with PyDeequ

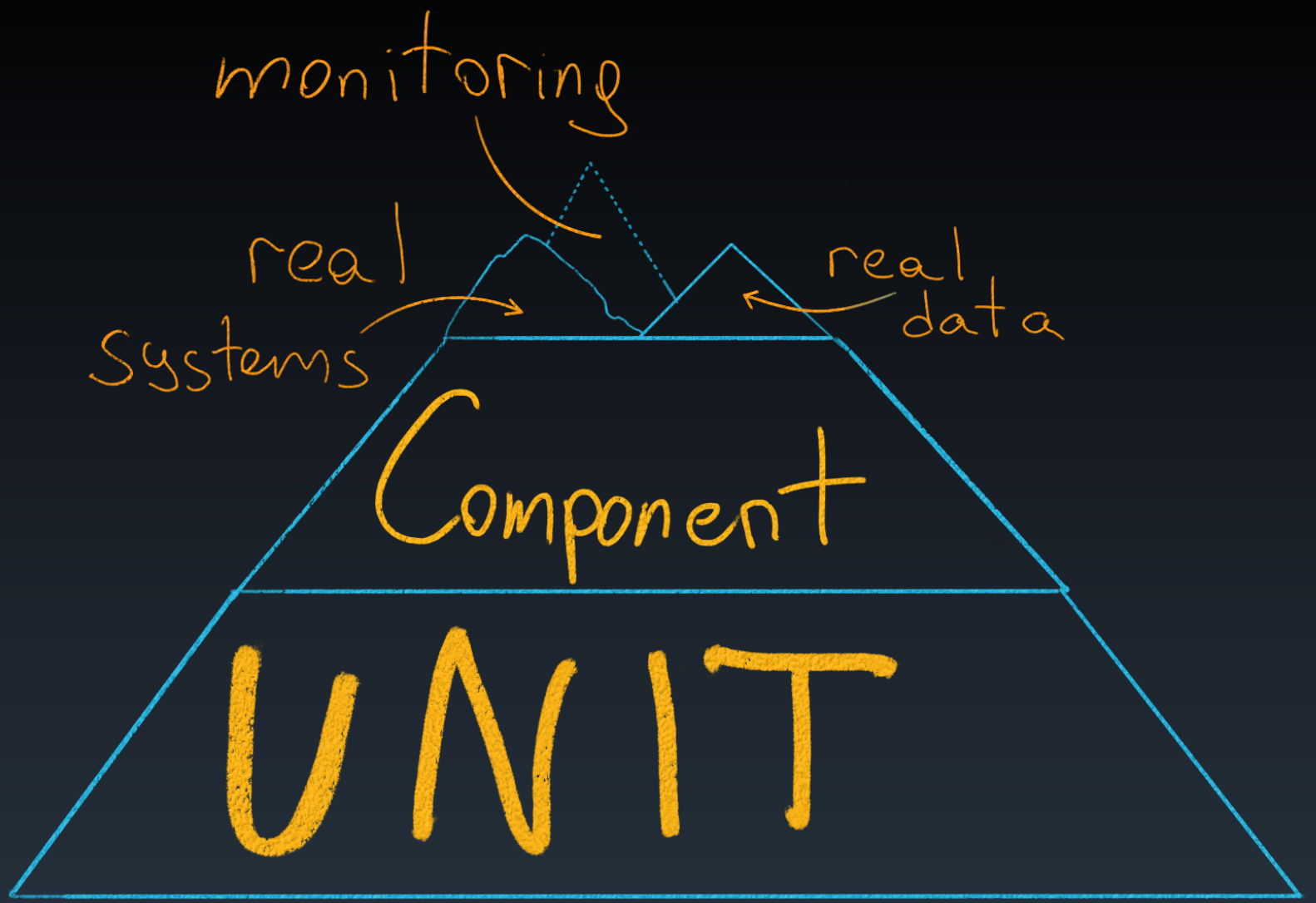
PyDeequ documentation

Python Deequ.

constraint	constraint_status	constraint_message
CompletenessConstraint	Success	
ComplianceConstraint	Failure	Value: 0.5 does not meet the constraint requirement!

Real data ecpectations. Use cases

- pre-ingestion and post-ingestion data validaton
- before pipeline development
- monitoring and alerting


monitoring
real
Systems
real
data
Component
UNIT

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
- business metrics

Monitoring

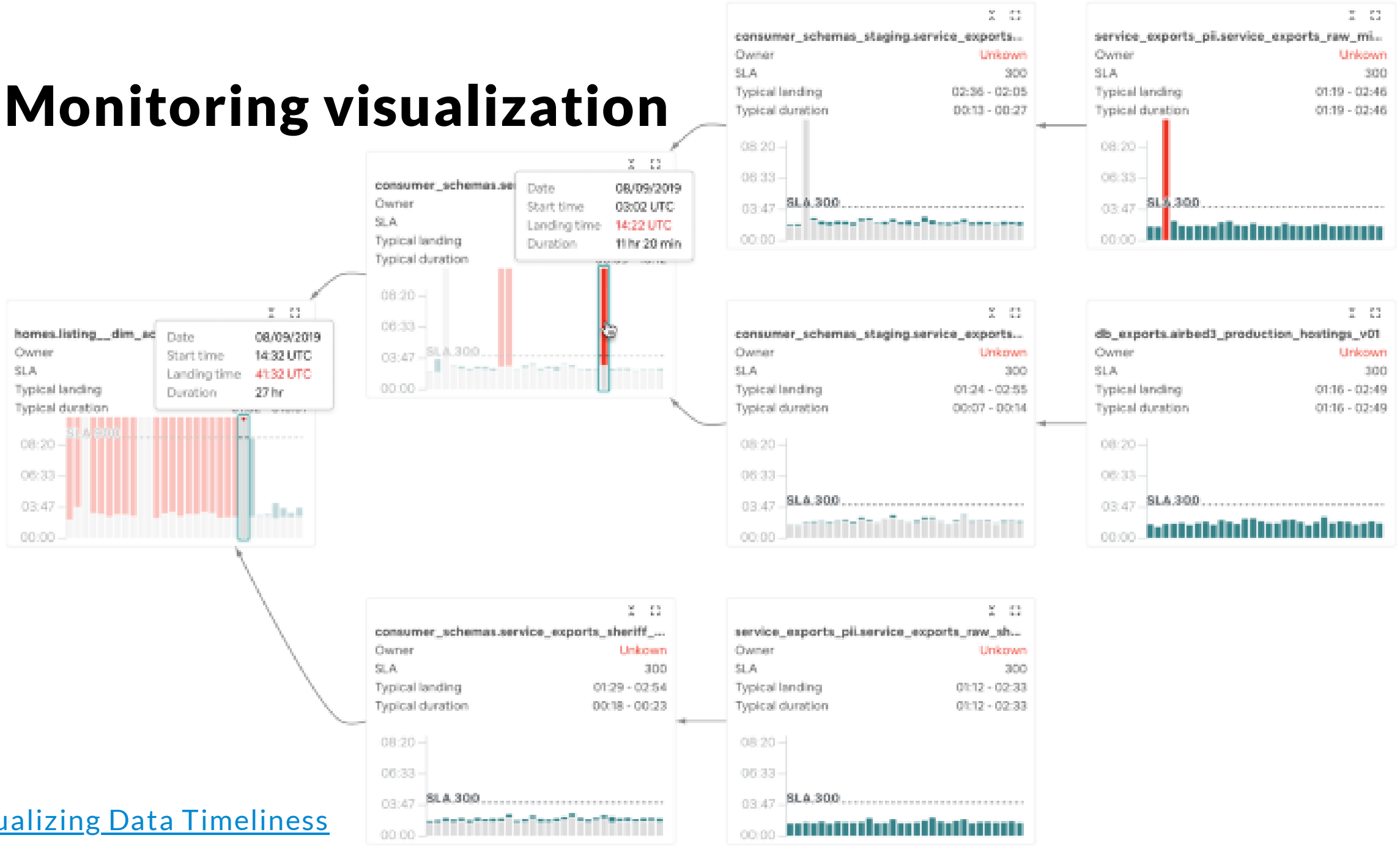
How?

- use Listeners
- use data aggregations
- AirFlow (or another orchestrator)

Data pipelines is always DAG

Monitoring should visualize it

Monitoring visualization



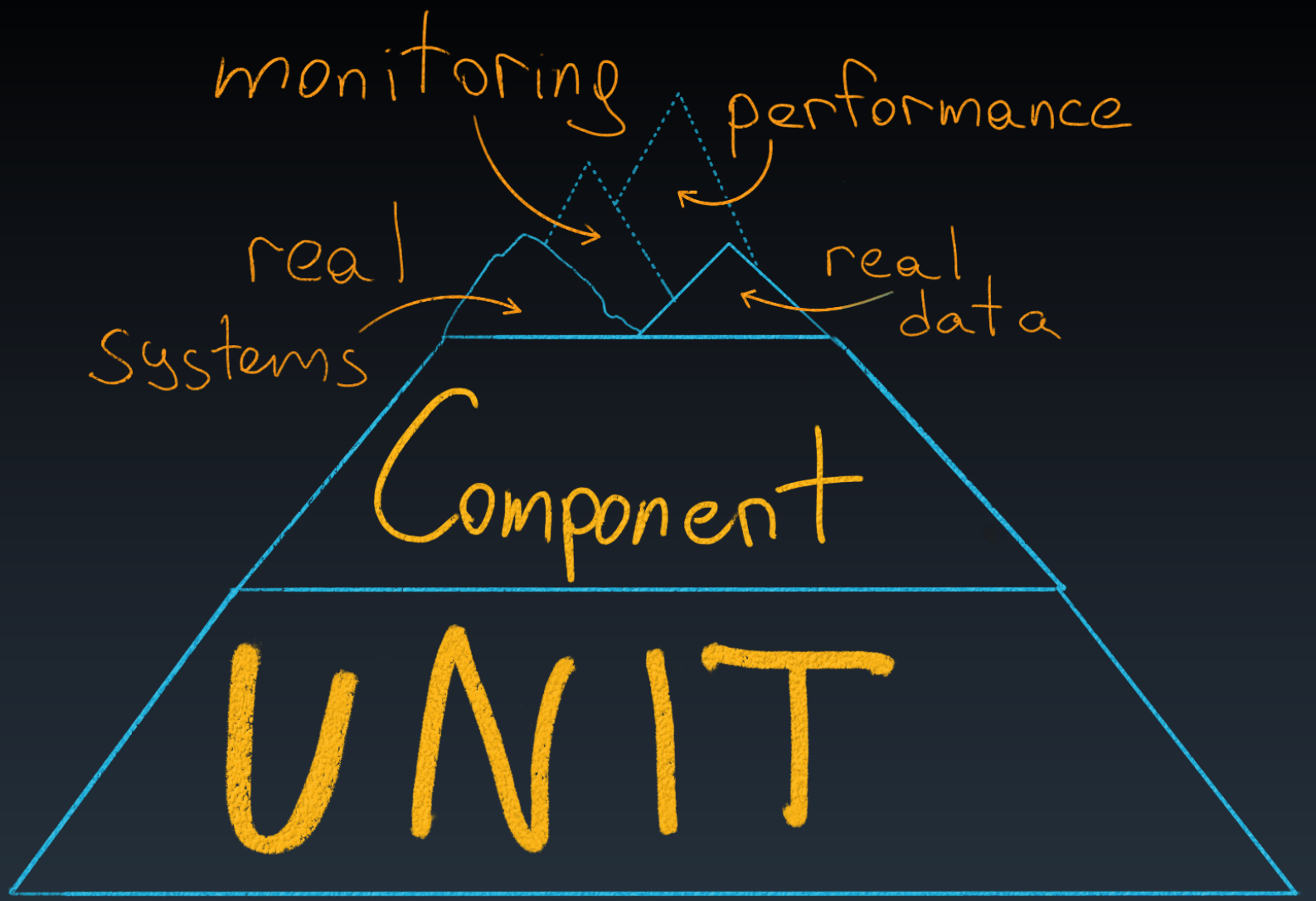
[Visualizing Data Timeliness]

End-to-End tests

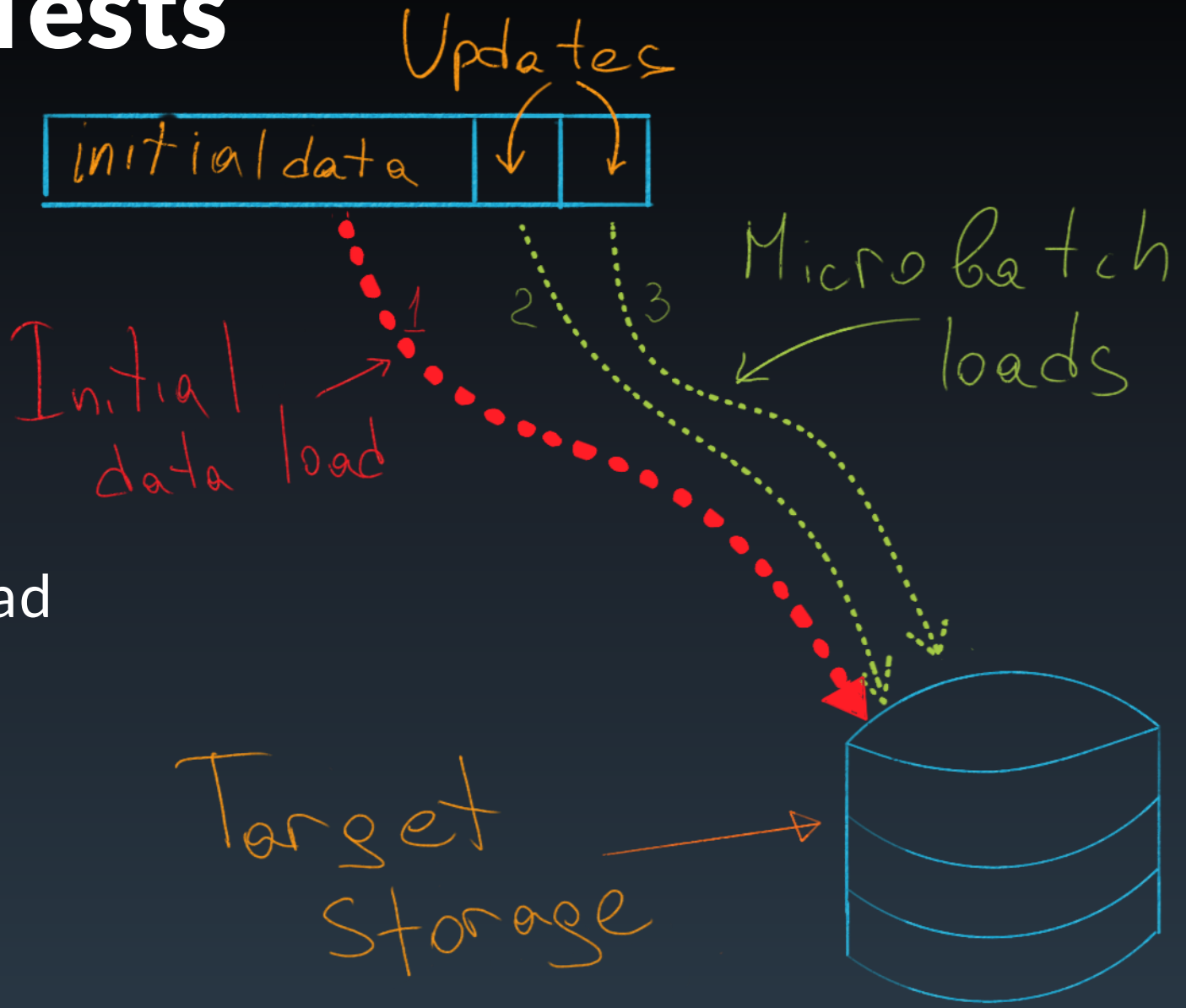
Compare with reports, old DWH

Multiple dimensions:

- data
- data latency
- performance, scalability


monitoring
performance
real
Systems
real
data
Component
UNIT

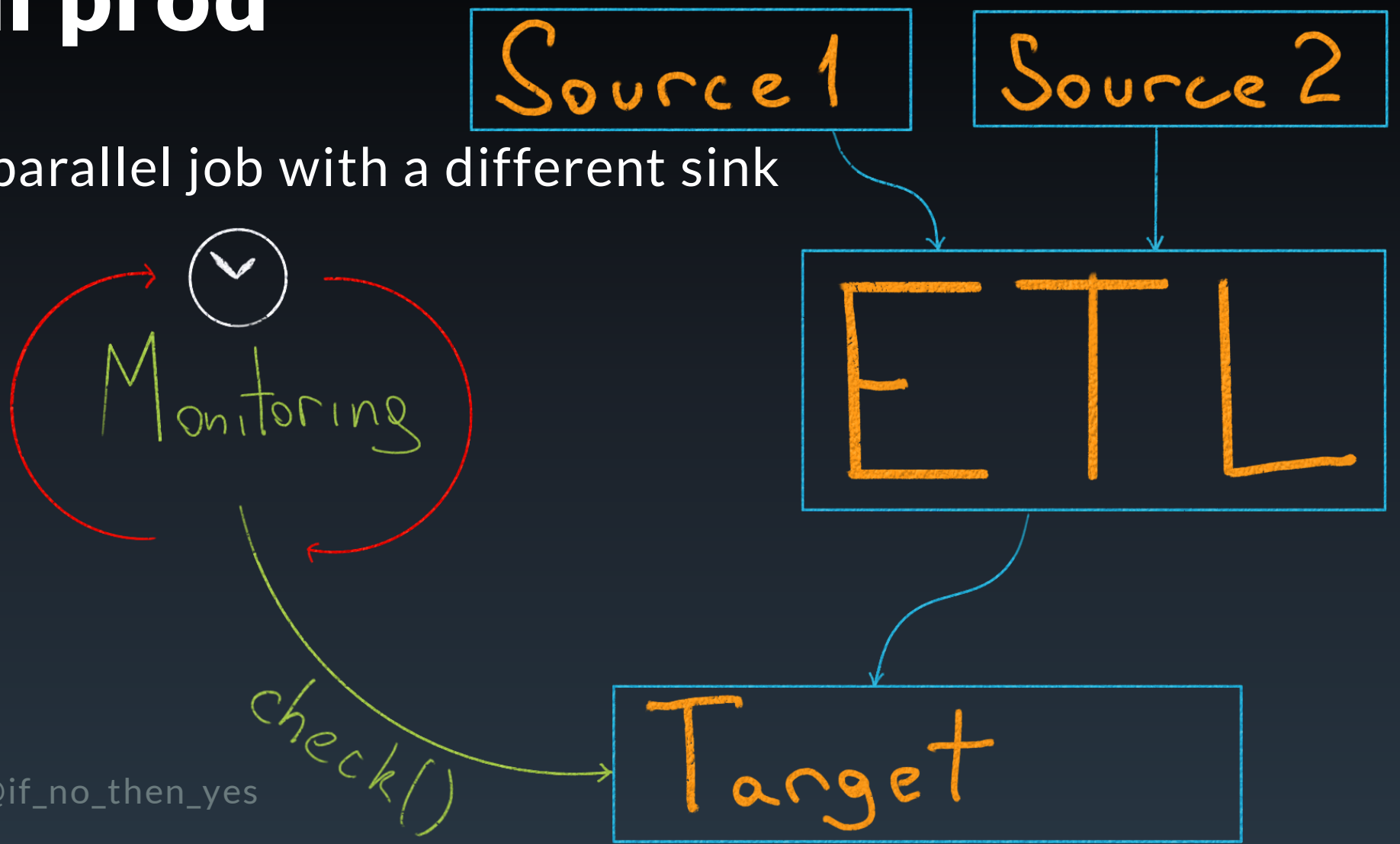
Performance Tests



- start with SLO
- test your initial data load

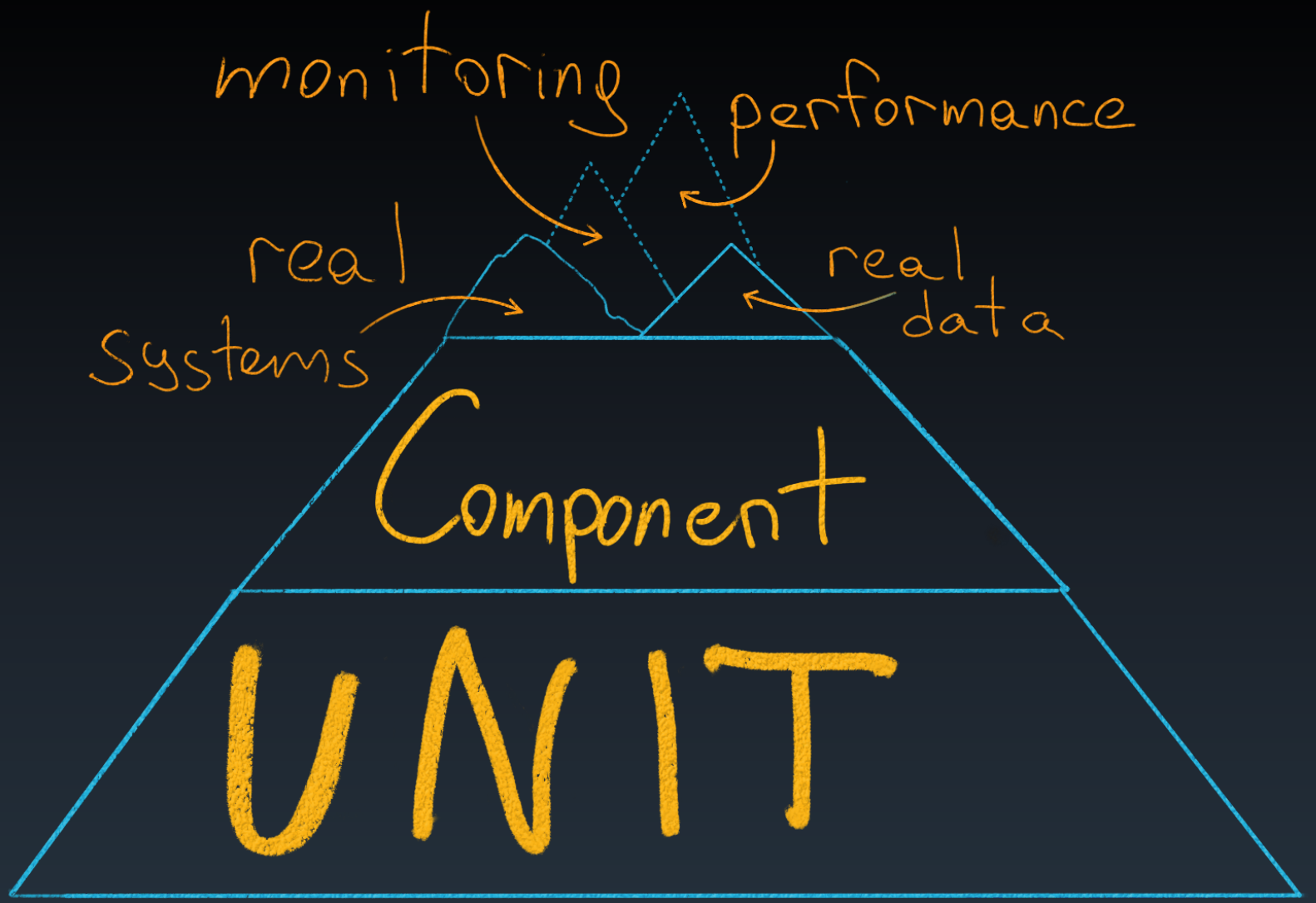
Real prod

Run a parallel job with a different sink



@asm0di0 @if_no_then_yes

[Using production data for testing in a post GDPR world]


monitoring
performance
real
Systems
real
data
Component
UNIT

Summary

- Testing pipeline is like testing code
- Testing pipelines is not like testing code
- Pipeline quality is not only about testing
- Sometimes testing outside of production is tricky

Thanks!

Questions? 🤷

@asm0di0
@if_no_then_yes