Himalayan Peaks

of Testing Data Pipelines

Ksenia Tomak, HelloFresh
Pasha Finkelshteyn, JetBrains

Pasha Finkelshteyn

Developer 🥑 for Big Data @ JetBrains

@asm0di0

🐘 @asm0dey@fosstodon.org

Data processing

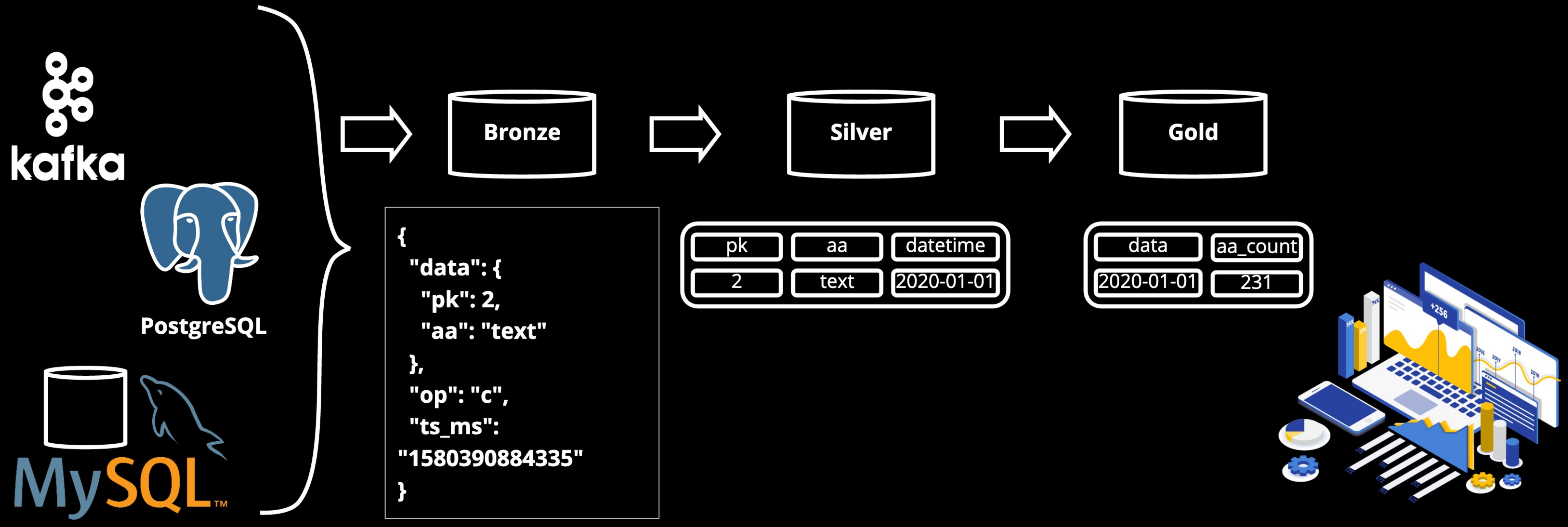
kafka

topic: actions

modify data

table: actions

Data lake?


kafka
PostgreSQL
MySQL
Bronze
Silver
Gold
{
"data": {
"pk": 2,
"aa": "text"
},
"op": "c",
"ts_ms":
"1580390884335"
}
pk
aa
datetime
2
text
2020-01-01
data
aa_count
2020-01-01
231

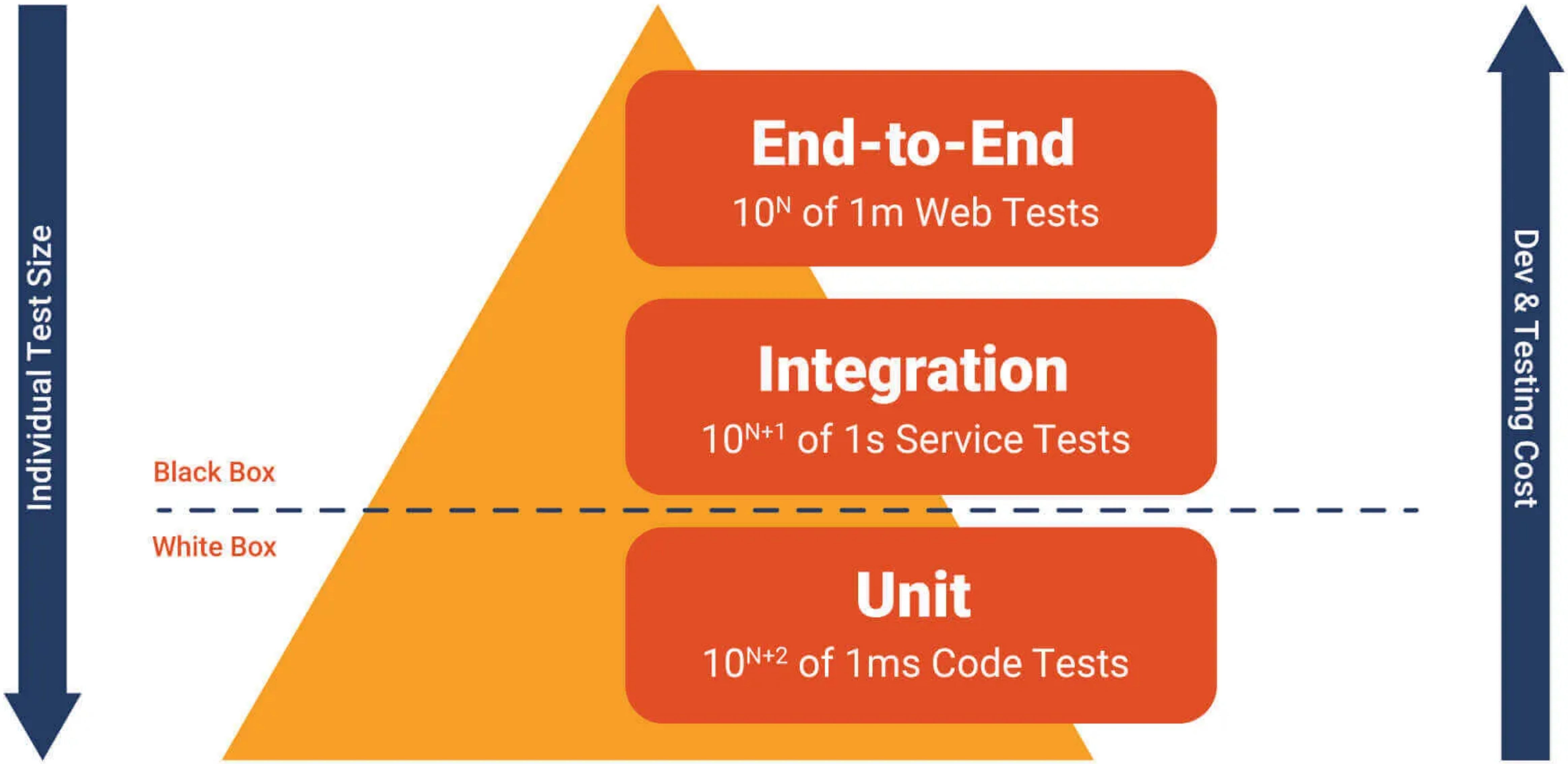
Who needs pipelines

- Data Scientists
- Data Analytics
- Marketing
- PO

It have to be tested!

Pyramid of testing?

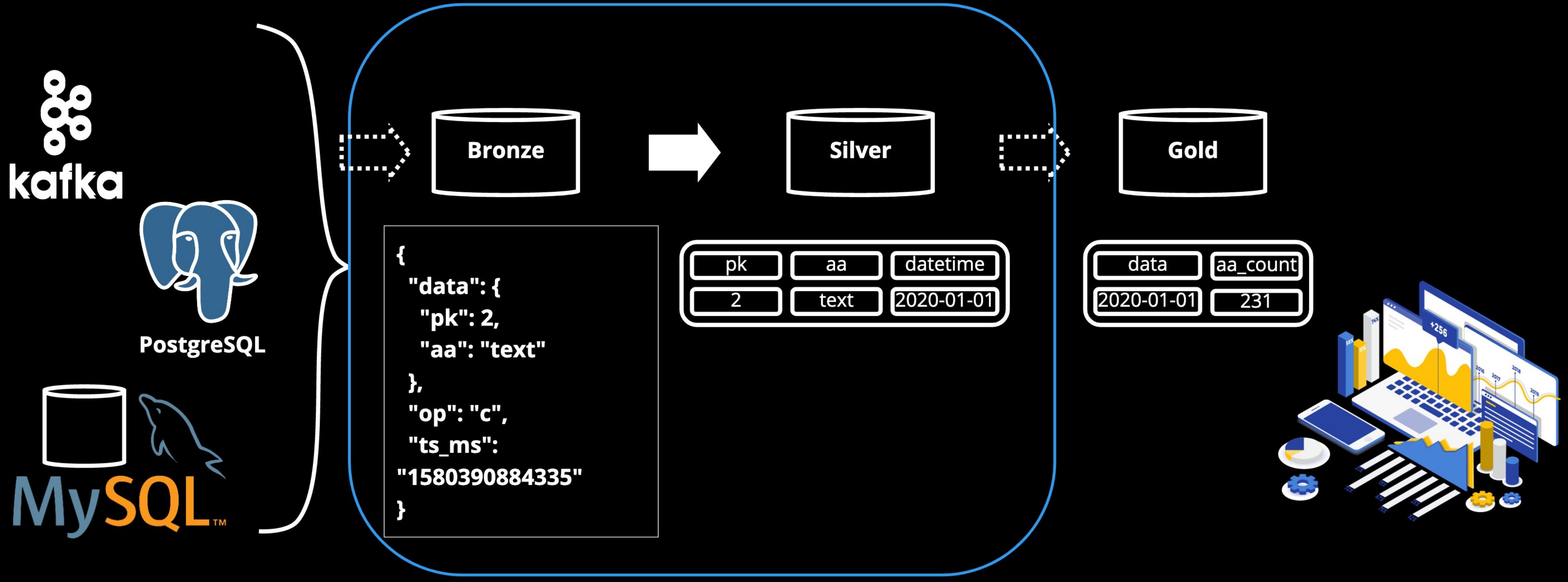
Pyramid of testing?



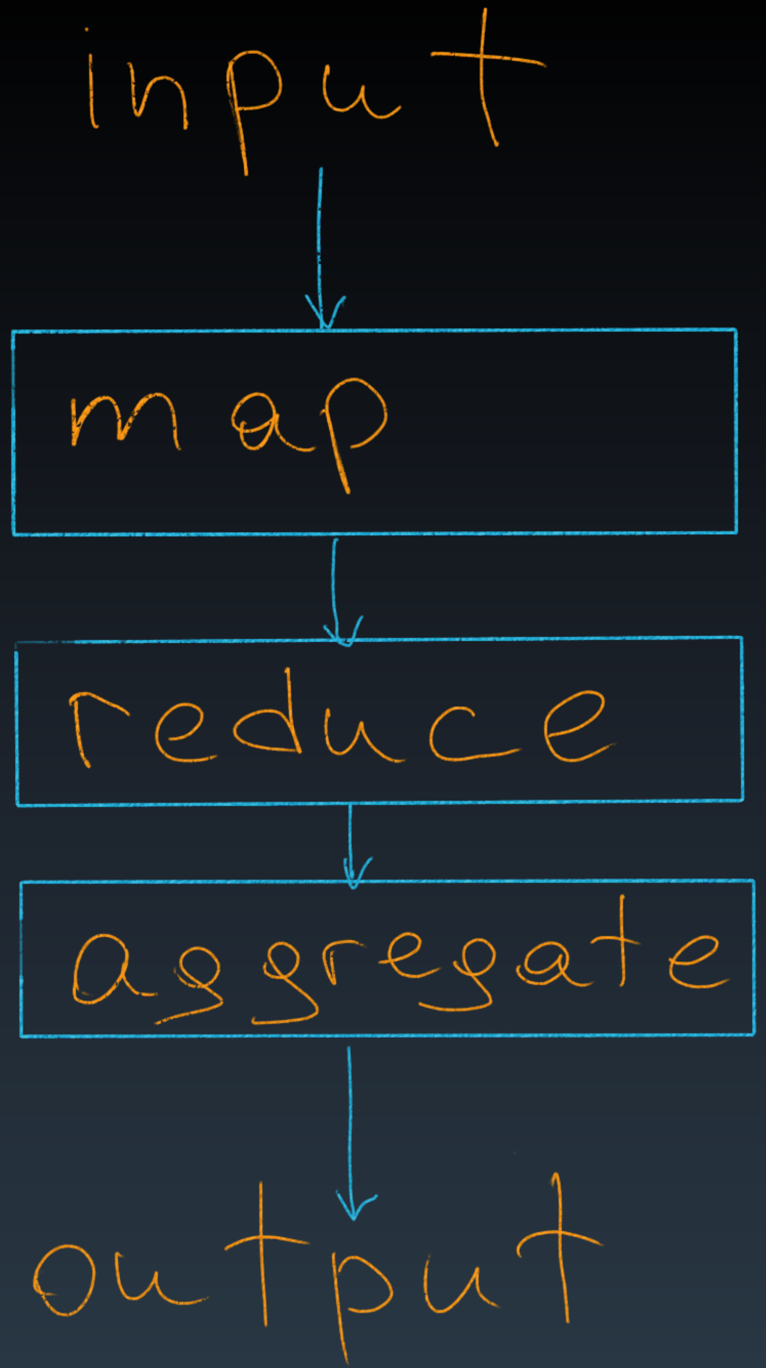
@asm0di0 @if_no_then_yes

Pyramid of testing. Unit

Bronze→Silver pipeline



Typical pipeline



Typical pipeline

```
StructType schema = new StructType(new StructField[]{
  new StructField("pk", DataTypes.LongType, false, Metadata.empty()),
  new StructField("aa", new DataTypes.StringType, false, Metadata.empty()),
});

spark.read
  .schema(schema)
  .csv(/* path */)
  .map(/* mapper */)
  .show() // terinal operation
```

Typical pipeline

```
StructType schema = new StructType(new StructField[]{
  new StructField("pk", DataTypes.LongType, false, Metadata.empty()),
  new StructField("aa", new DataTypes.StringType, false, Metadata.empty()),
});

spark.read
  .schema(schema)
  .csv(/* path */)
  .map(/* mapper */)
  .show() // terinal operation
```

Typical pipeline

```java
StructType schema = new StructType(new StructField[]{
  new StructField("pk", DataTypes.LongType, false, Metadata.empty()),
  new StructField("aa", new DataTypes.StringType, false, Metadata.empty()),
});

spark.read
  .schema(schema)
  .csv(/* path */)
  .map(/* mapper */)
  .show() // terinal operation
```

Typical pipeline

```
StructType schema = new StructType(new StructField[]{
  new StructField("pk", DataTypes.LongType, false, Metadata.empty()),
  new StructField("aa", new DataTypes.StringType, false, Metadata.empty()),
});

spark.read
  .schema(schema)
  .csv(/* path */)
  .map(/* mapper */)
  .show() // terinal operation
```

Typical pipeline

```
StructType schema = new StructType(new StructField[]{
  new StructField("pk", DataTypes.LongType, false, Metadata.empty()),
  new StructField("aa", new DataTypes.StringType, false, Metadata.empty()),
});

spark.read
  .schema(schema)
  .csv(/* path */)
  .map(/* mapper */)
  .show() // terinal operatio
```

Unit testing of pipeline

What may we test here?

A pipeline should transform data correctly!

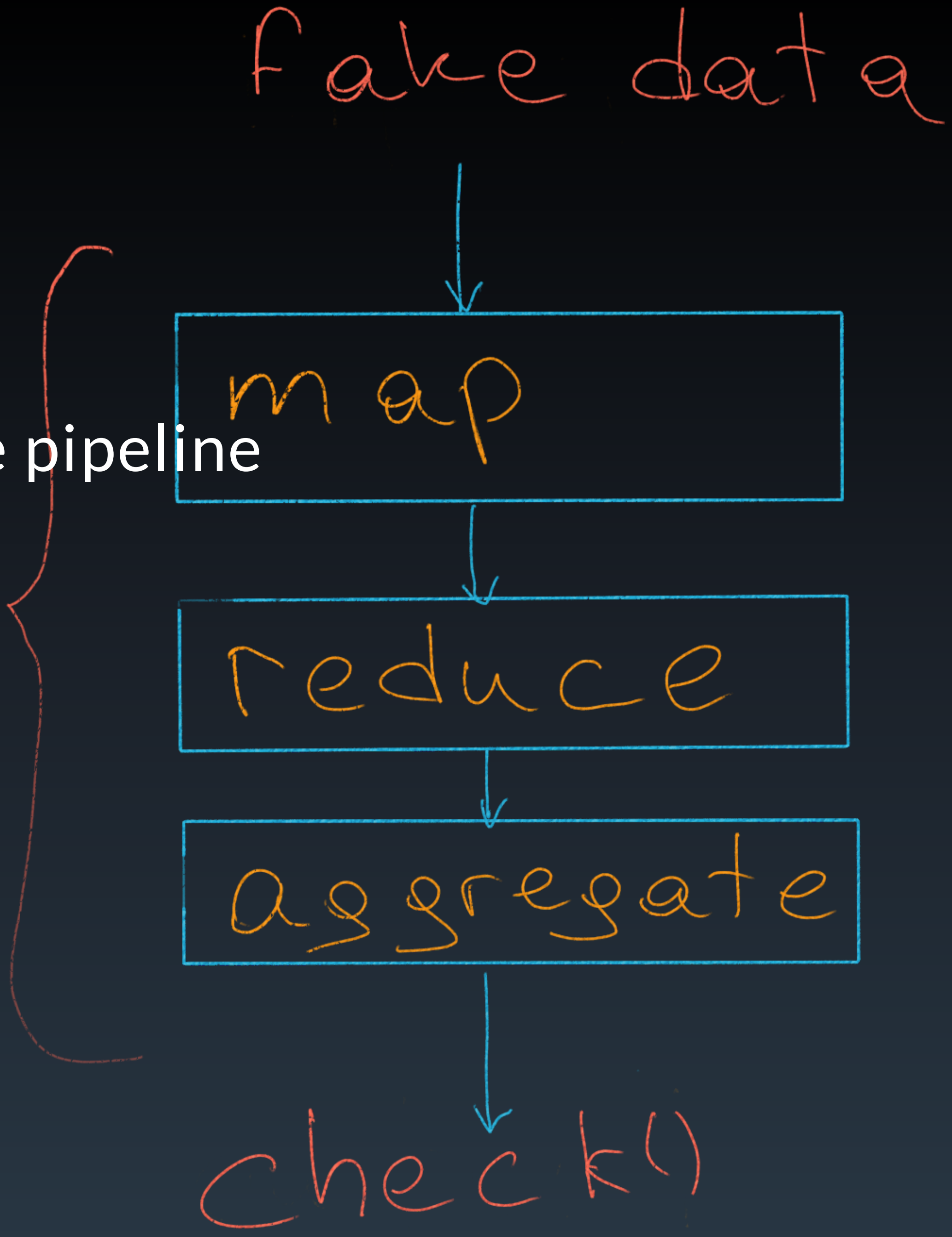
Correctness is a business term

Let's paste fakes!

Fake input data

Reference data at the end of the pipeline

Separate function



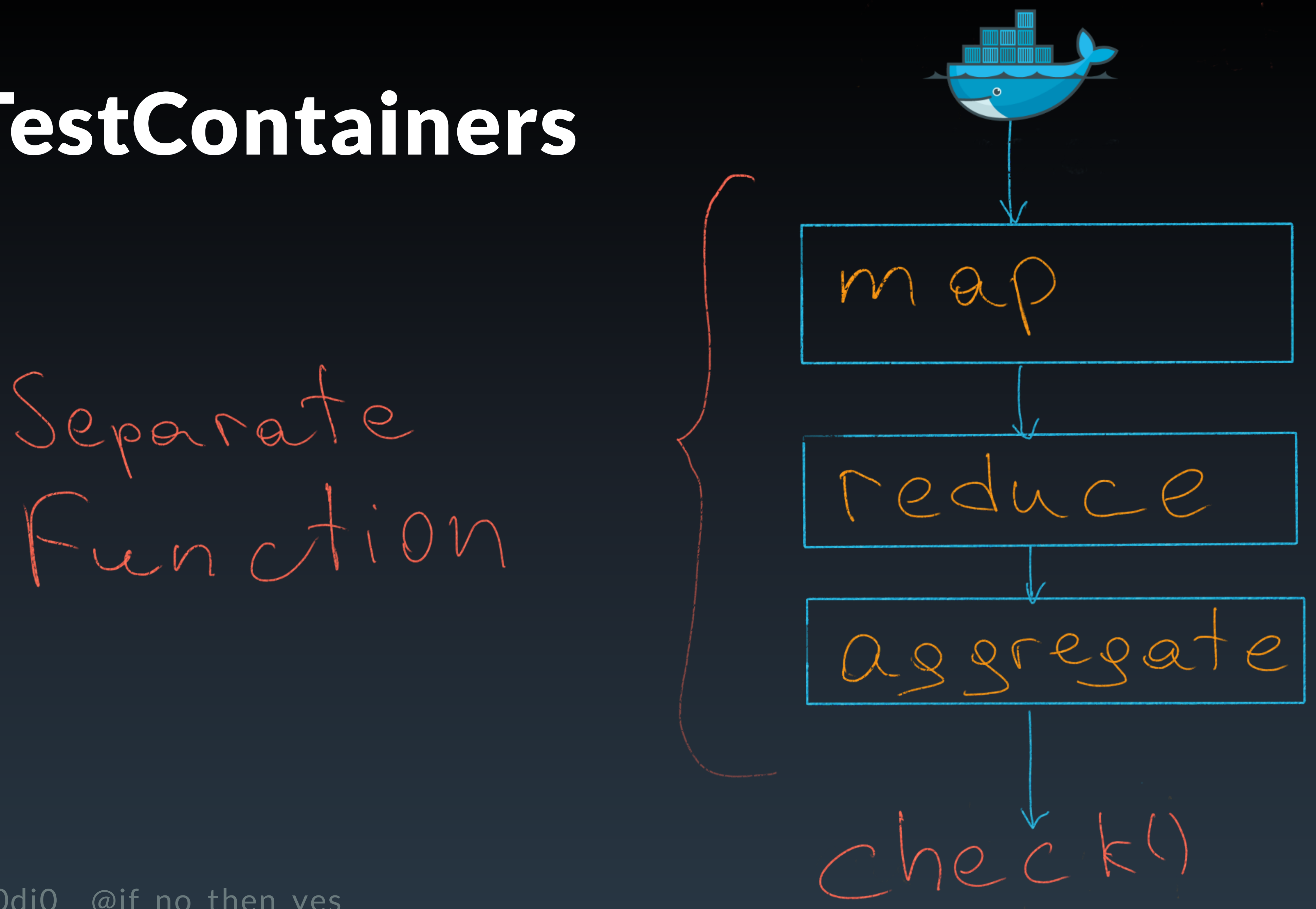
Tools

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

Component testing

TESTCONTAINERS

TestContainers



TestContainers

Supported languages:

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

Test Containers

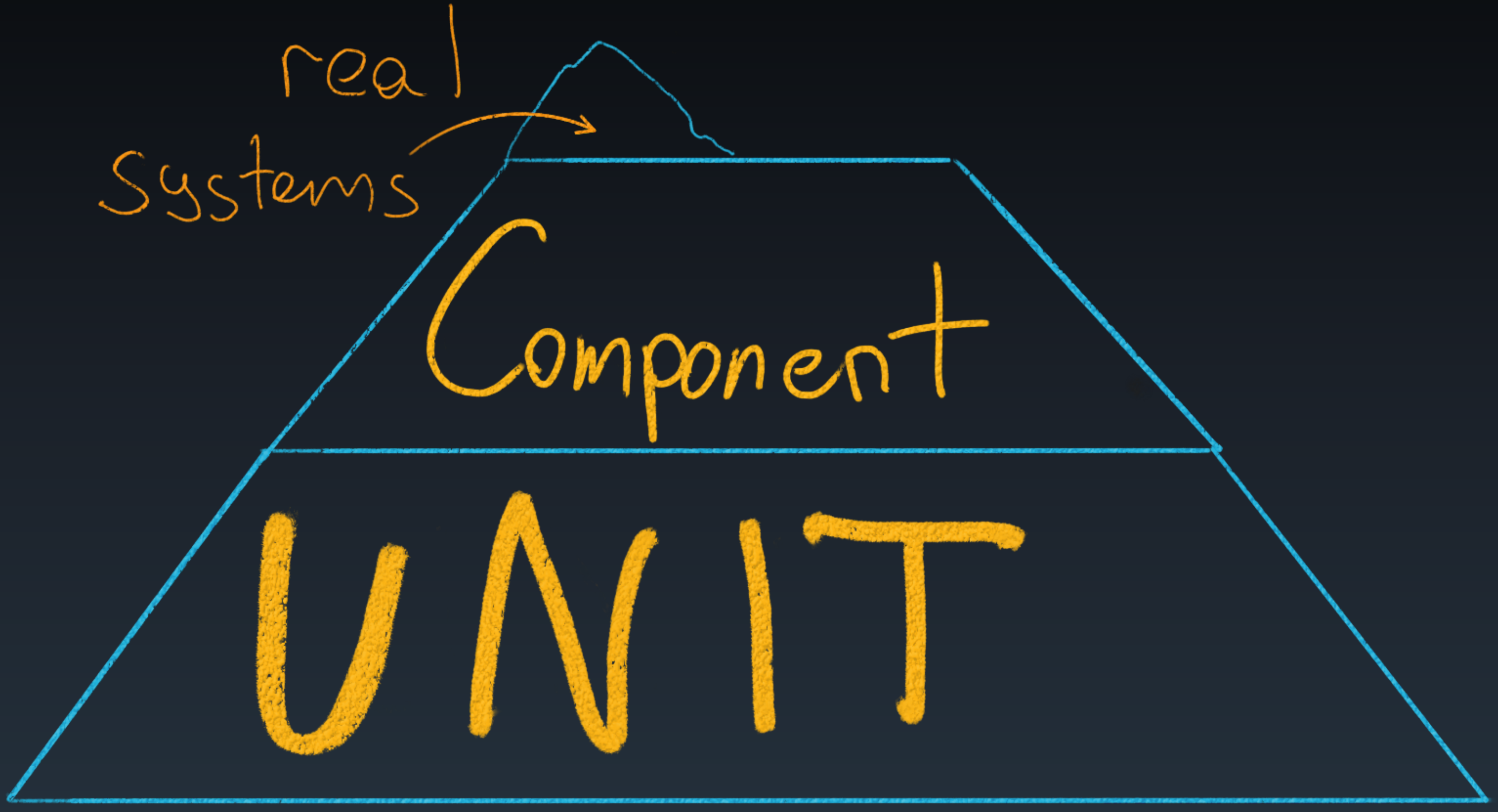
```python
from testcontainers.postgres import PostgresContainer
import sqlalchemy

postgres_container = PostgresContainer("postgres:9.5")
with postgres_container as postgres:
  e = sqlalchemy.create_engine(postgres.get_connection_url())
  result = e.execute("select version()")
  version, = result.fetchone()
version
```

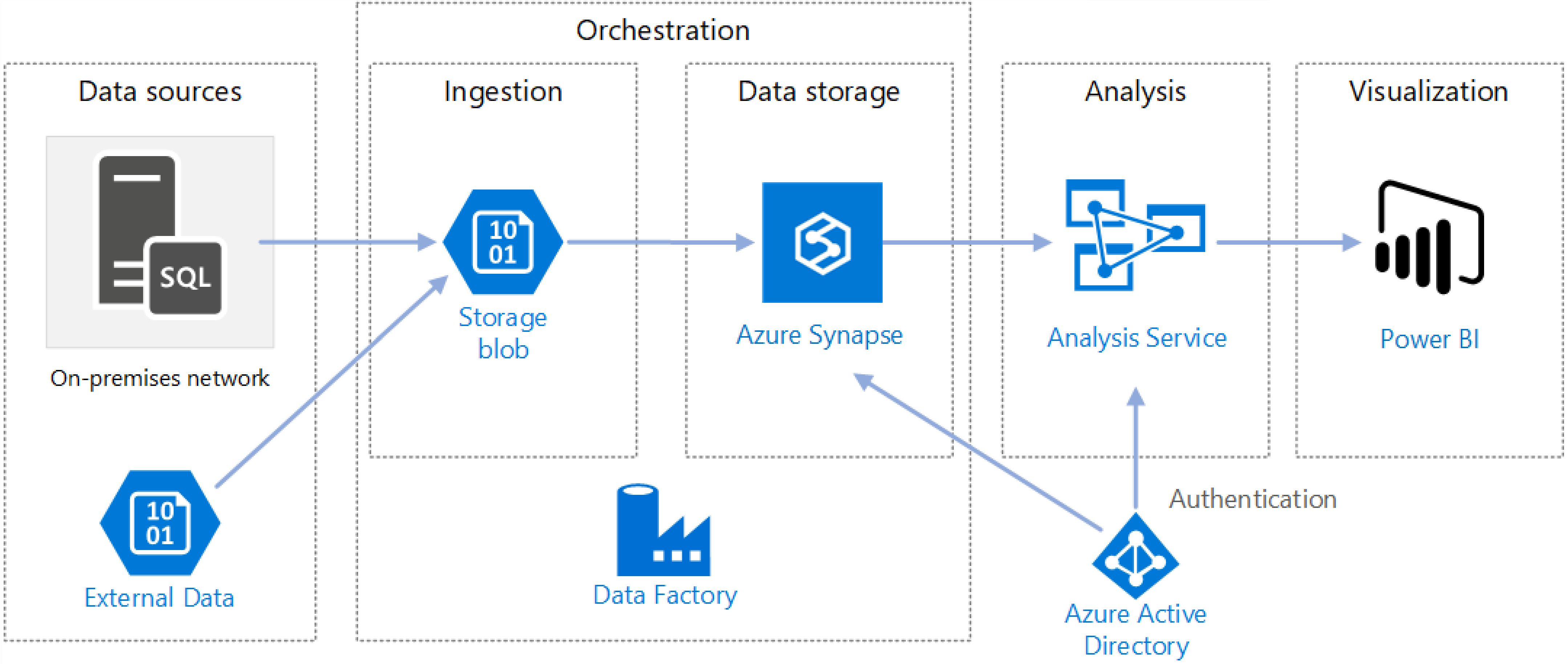
Test Containers

```python
from testcontainers.postgres import PostgresContainer
import sqlalchemy

postgres_container = PostgresContainer("postgres:9.5")
with postgres_container as postgres:
  e = sqlalchemy.create_engine(postgres.get_connection_url())
  result = e.execute("select version()")
  version, = result.fetchone()
version
```

Test Containers

```python
from testcontainers.postgres import PostgresContainer
import sqlalchemy

postgres_container = PostgresContainer("postgres:9.5")
with postgres_container as postgres:
  e = sqlalchemy.create_engine(postgres.get_connection_url())
  result = e.execute("select version()")
  version, = result.fetchone()
version
```


real
systems
Component
UNIT

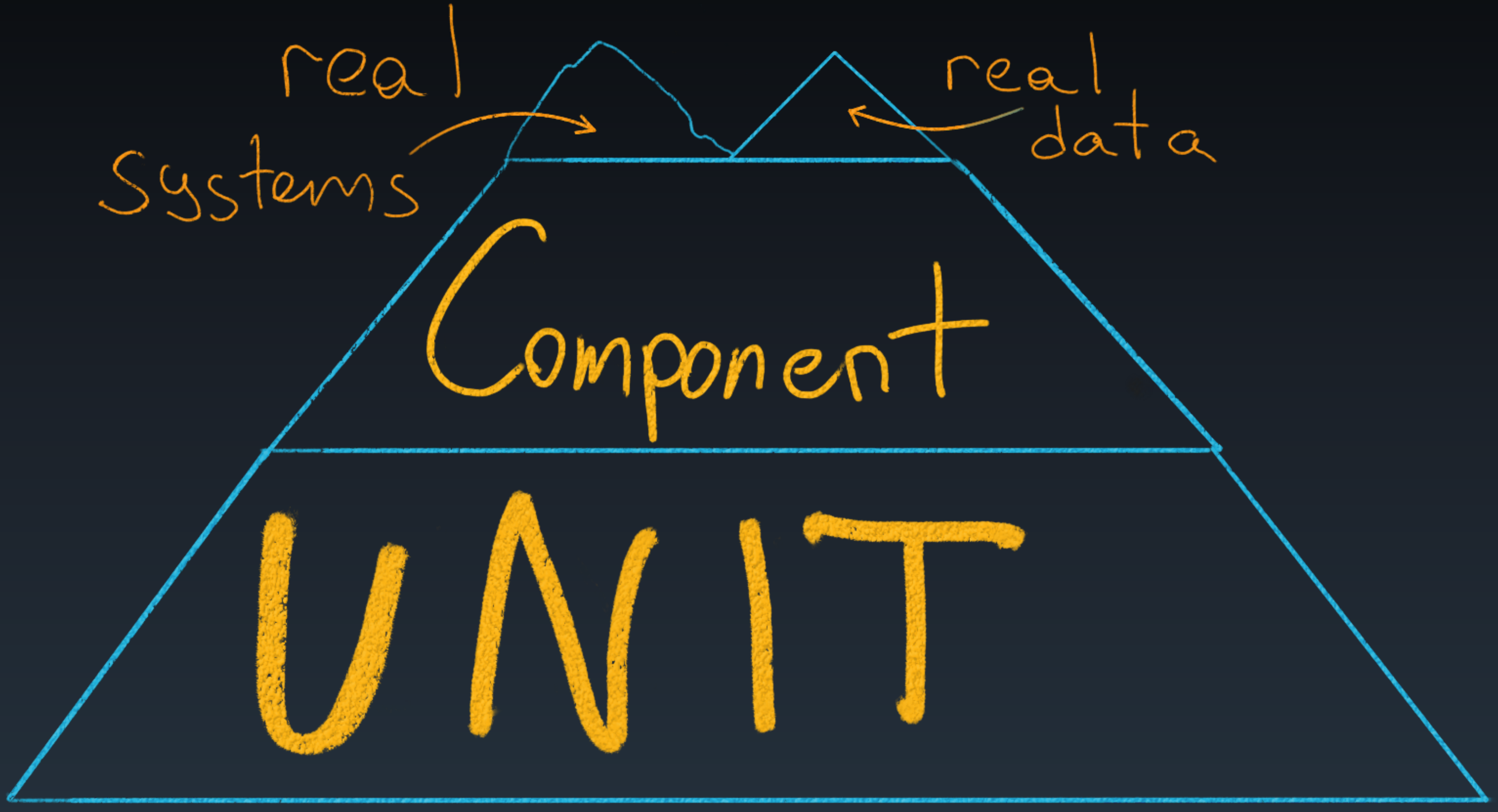


@asm0di0 @if_no_then_yes

Real systems

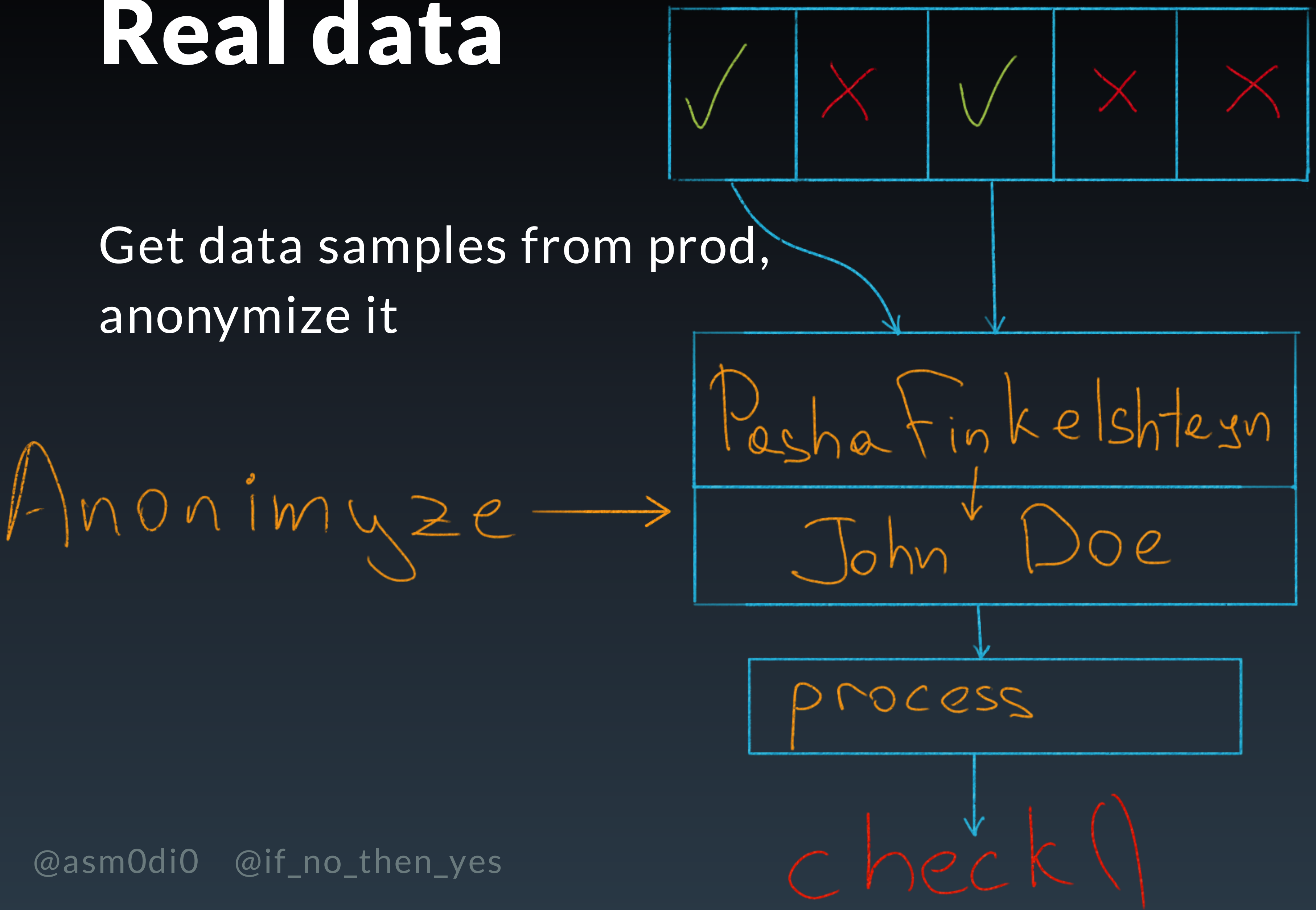
Why are component tests not enough?

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

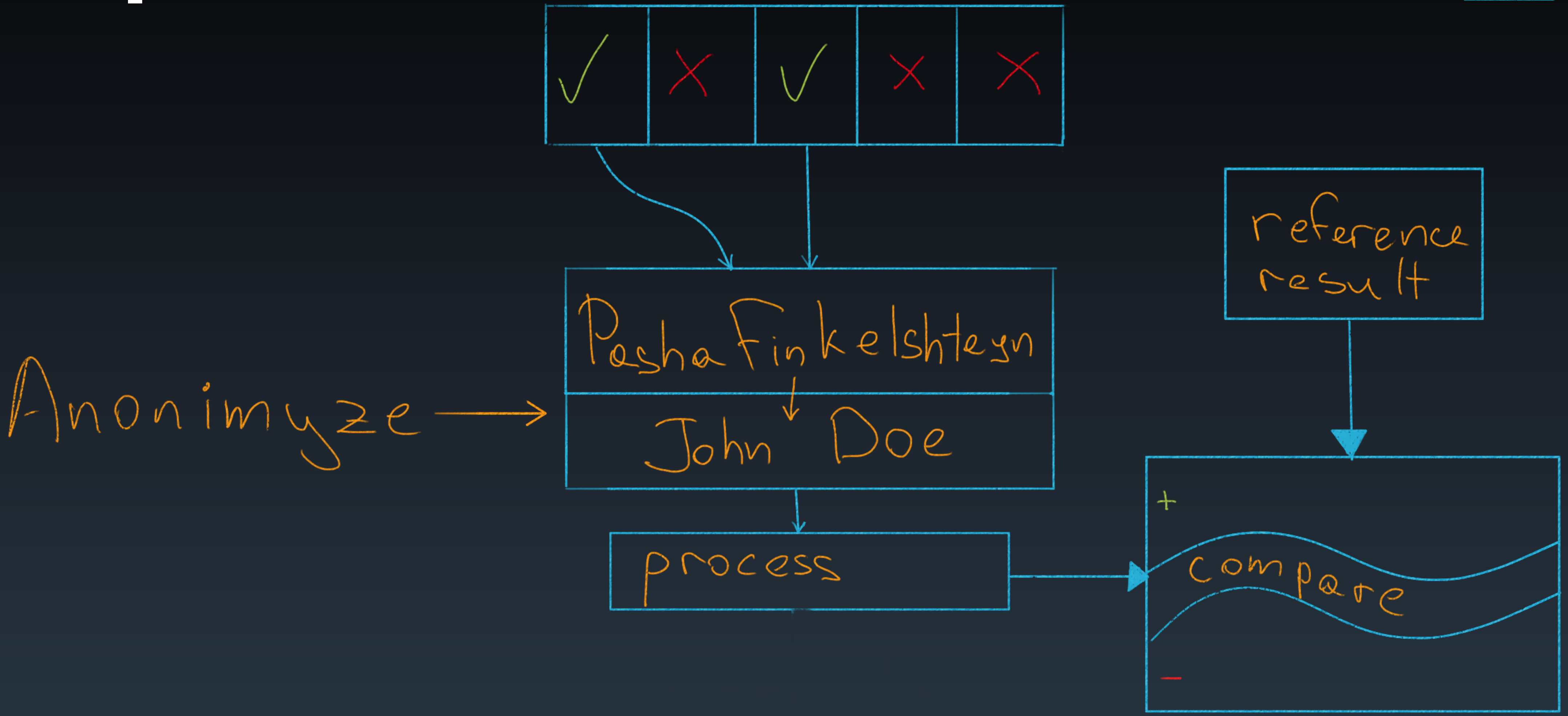

real
systems
real
data
Component
UNIT

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

Automated Testing For Protecting Data Pipelines from Undocumented Assumptions

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))
```

```
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

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

```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

```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]

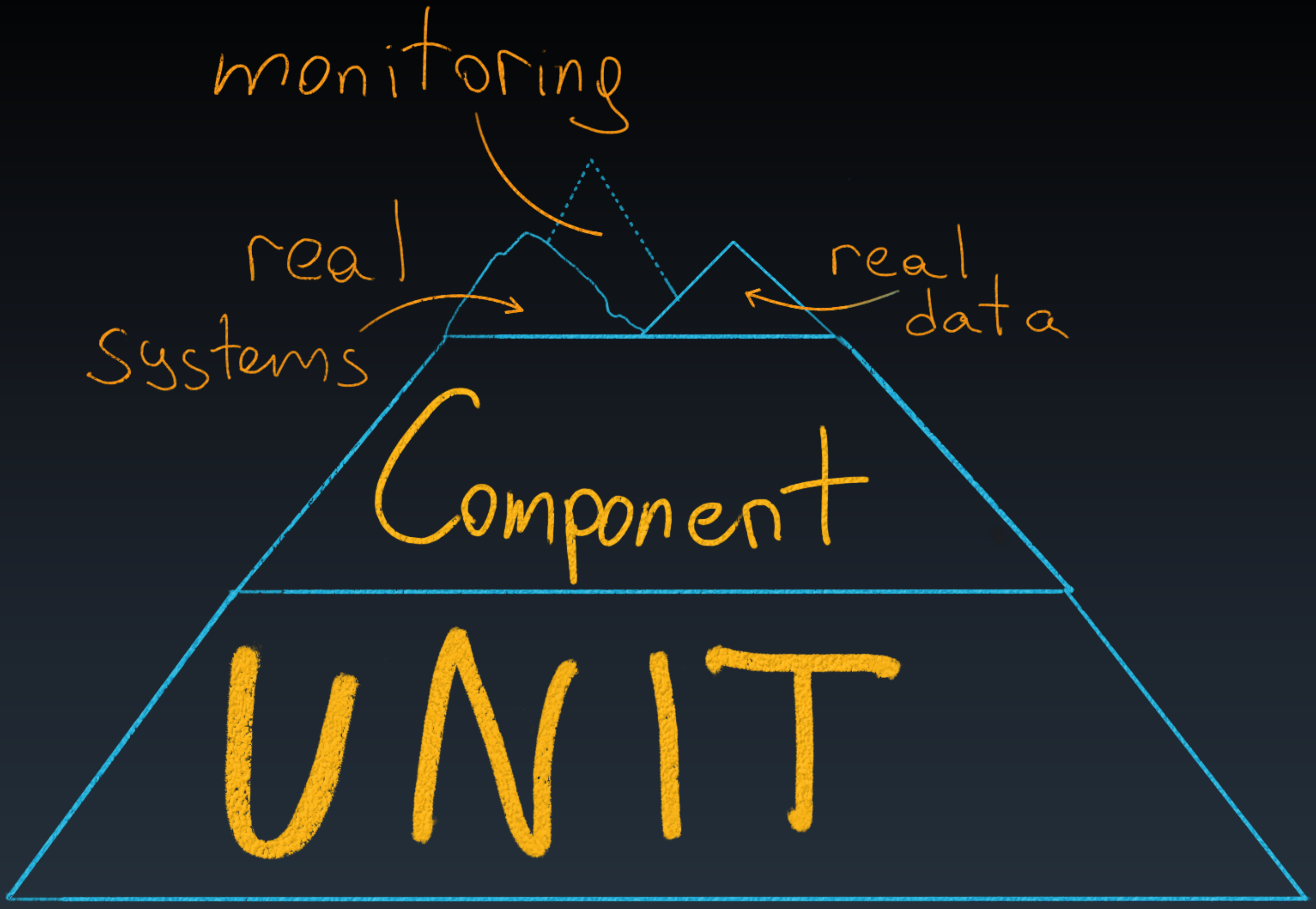
[PyDeequ documentation]

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

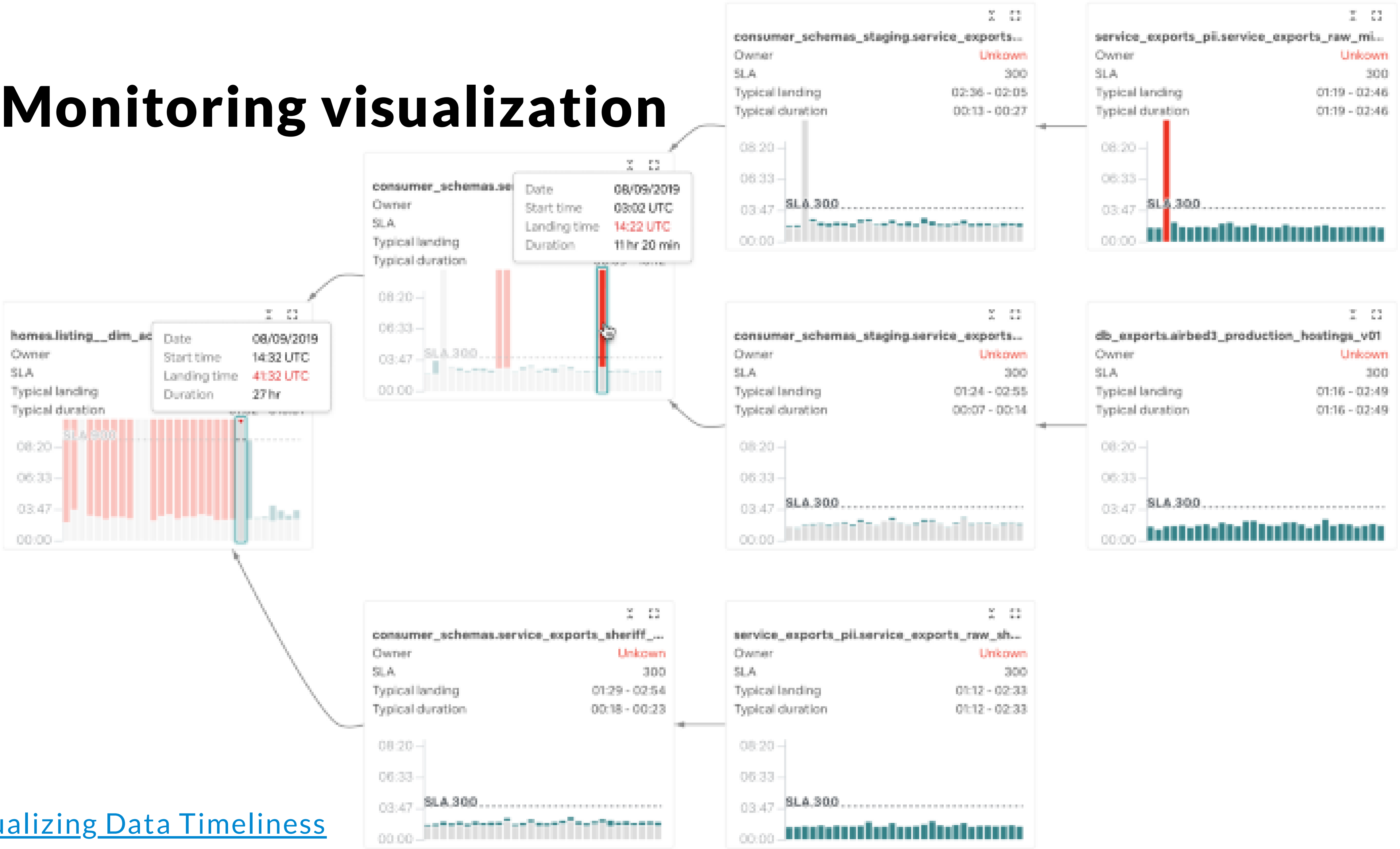
How?

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

Data pipelines is always DAG

Monitoring should visualize it

Monitoring visualization

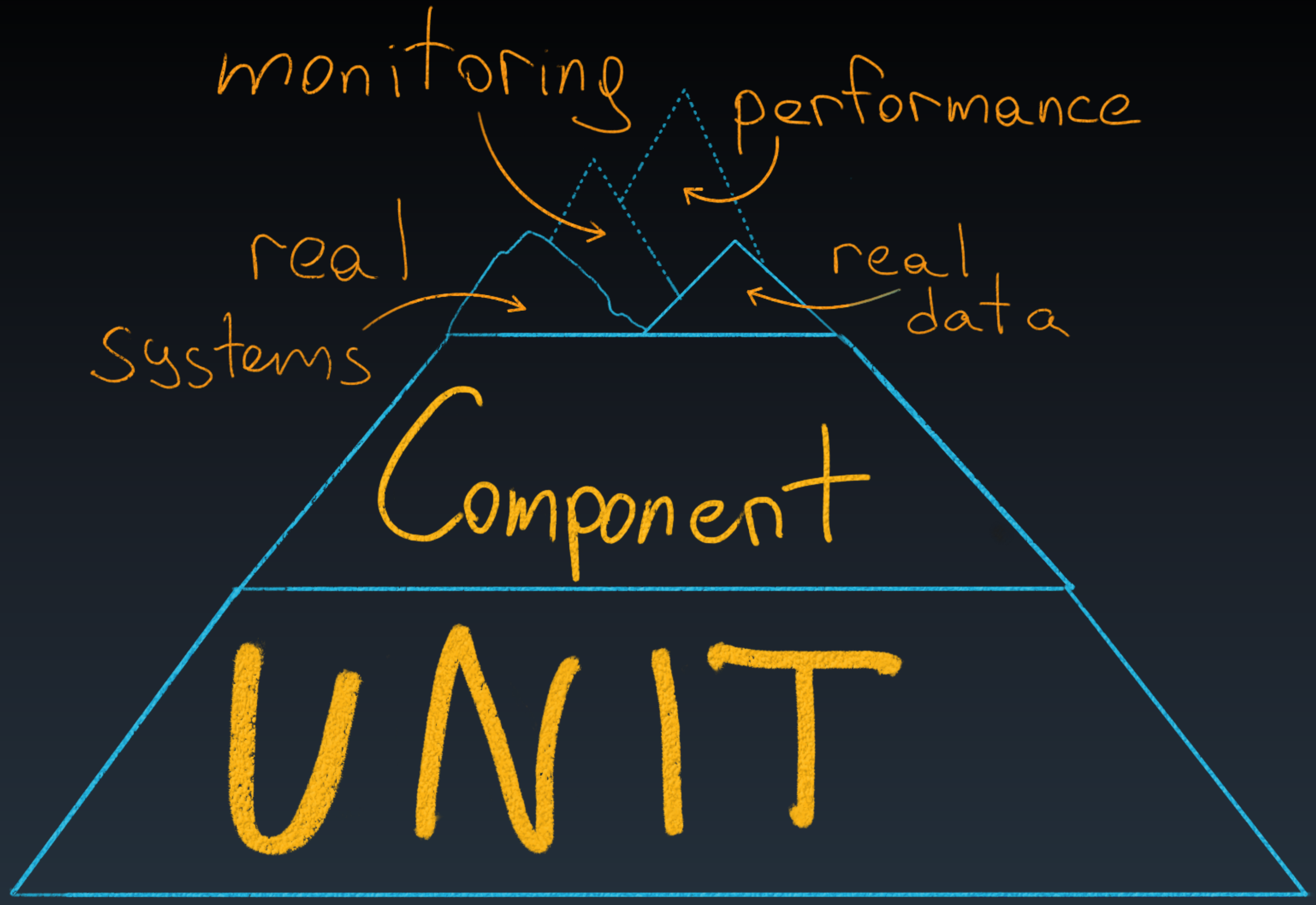


End-to-End tests

Compare with reports, old DWH

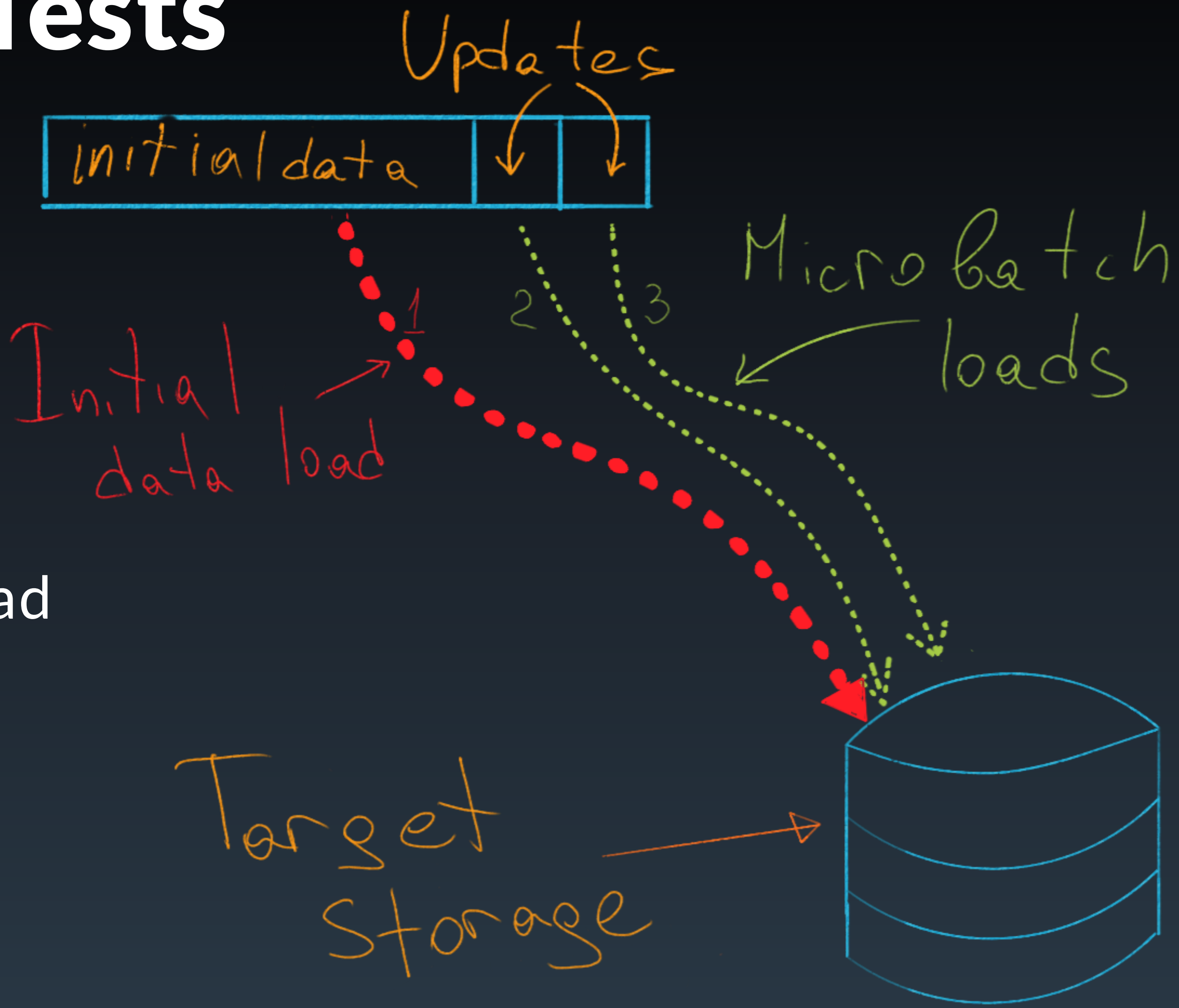
Multiple dimensions:

- data
- data latency
- performance, scalability


monitoring
performance
real
systems
real
data
Component
UNIT

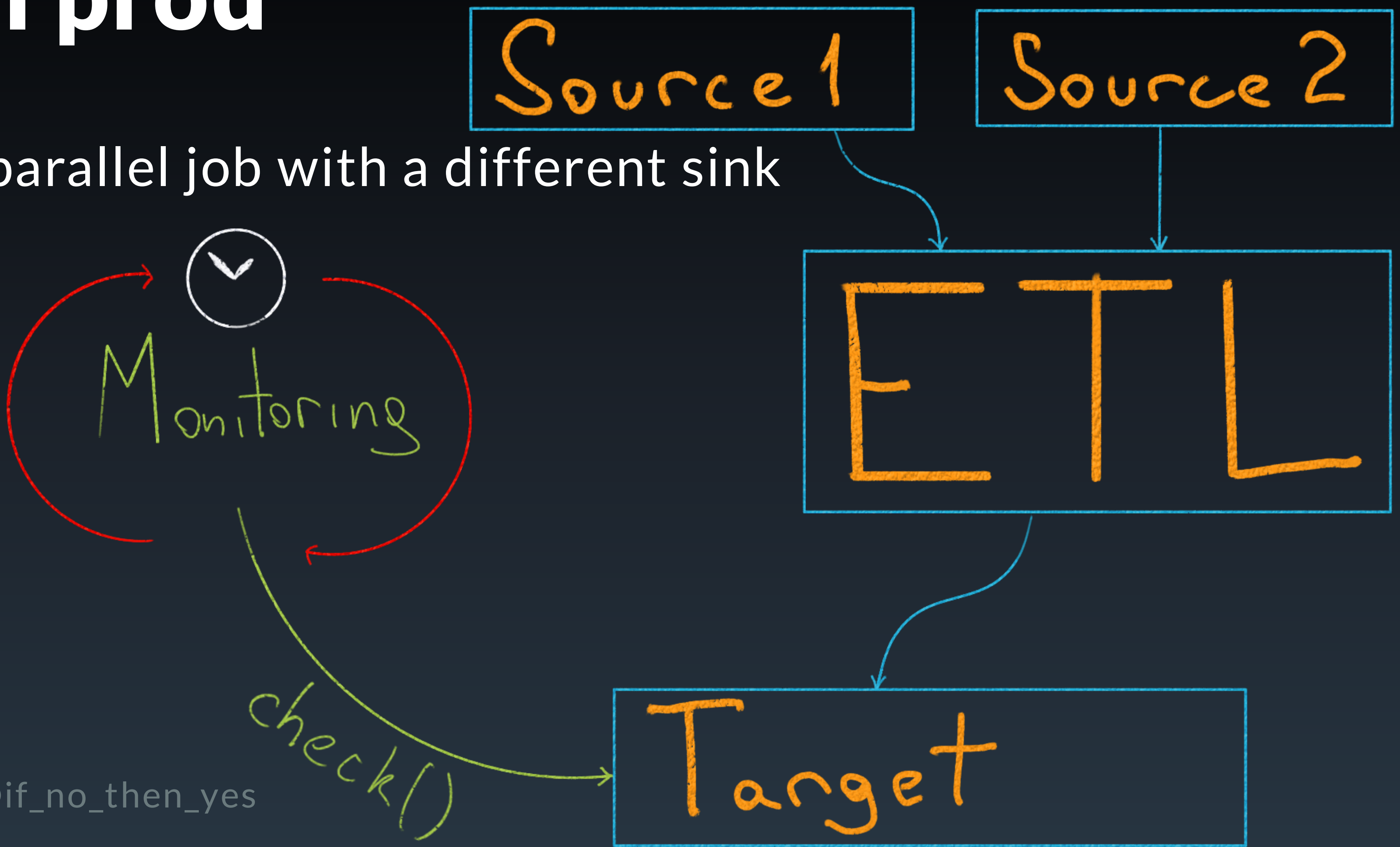
Performance Tests

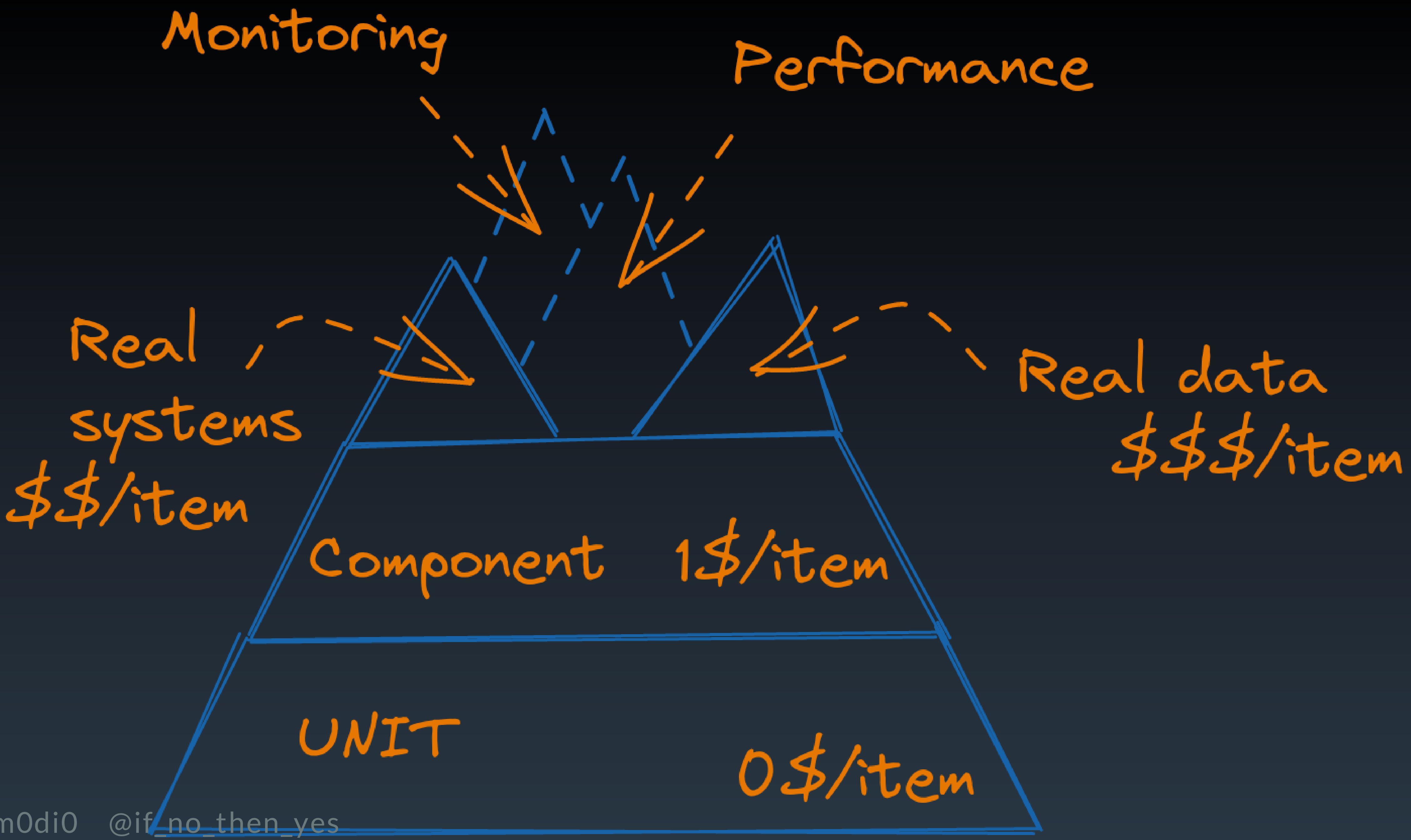
- start with SLO
- test your initial data load



Real prod

Run a parallel job with a different sink




Monitoring
Performance
Real systems
$$/item
Real data
$$$/item
Component
1$/item
UNIT
0$/item

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

🐘 @asm0dey@fosstodon.org