#JCON2025
www.jcon.one

JAVAPRO

Himalayan Peaks

of Testing Data Pipelines

Kseniia Tomak, HelloFresh

Pasha Finkelshteyn, BellSoft

```
whoami
```

whoami

- Pasha Finkelshteyn

whoami

- Pasha Finkelshteyn
- Dev 🥑 at BellSoft

whoami

- Pasha Finkelshteyn
- Dev 🥑 at BellSoft
- ≈10 years in JVM. Mostly ☕ and

whoami

- Pasha Finkelshteyn
- Dev 🥑 at BellSoft
- ≈10 years in JVM. Mostly ☕ and
- And 🍃

whoami

- Pasha Finkelshteyn
- Dev 🥑 at BellSoft
- ≈10 years in JVM. Mostly ☕ and
- And 🍃
- 🐦 asm0di0

whoami

- Pasha Finkelshteyn
- Dev 🥑 at BellSoft
- ≈10 years in JVM. Mostly ☕ and
- And 🍃
- asm0di0
- @asm0dey@fosstodon.org

whoami

whoami

- Kseniia Tomak

whoami

- Kseniia Tomak
- Data Engineering Manager at HelloFresh

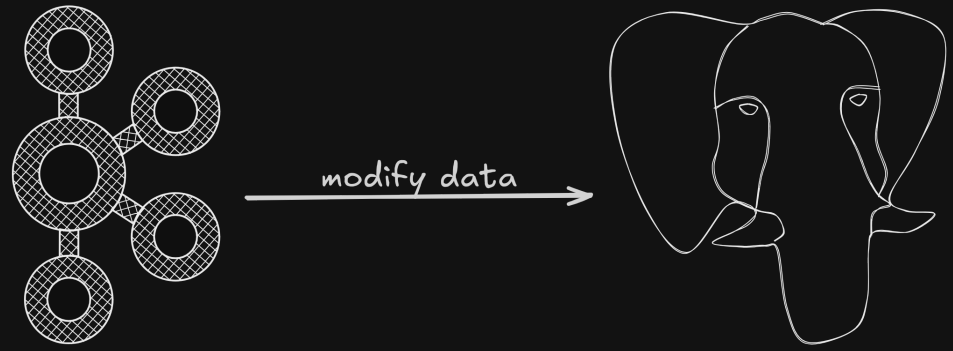
whoami

- Kseniia Tomak
- Data Engineering Manager at HelloFresh
- ≈14 years of Engineering experience. Mostly Data engineering

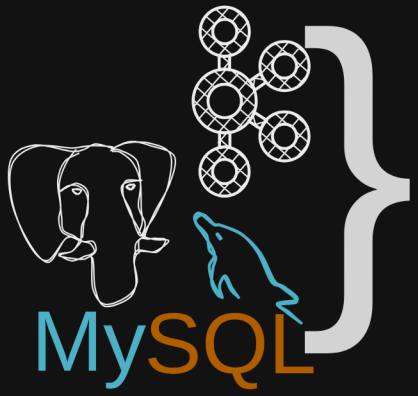
whoami

- Kseniia Tomak
- Data Engineering Manager at HelloFresh
- ≈14 years of Engineering experience. Mostly Data engineering
- ksenia-tomak

Data processing

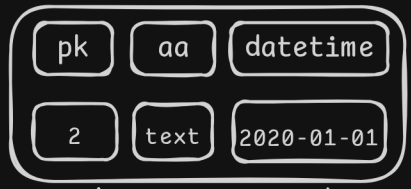


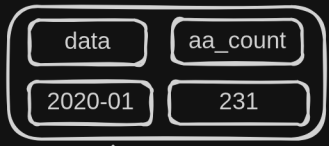
Data Lake?

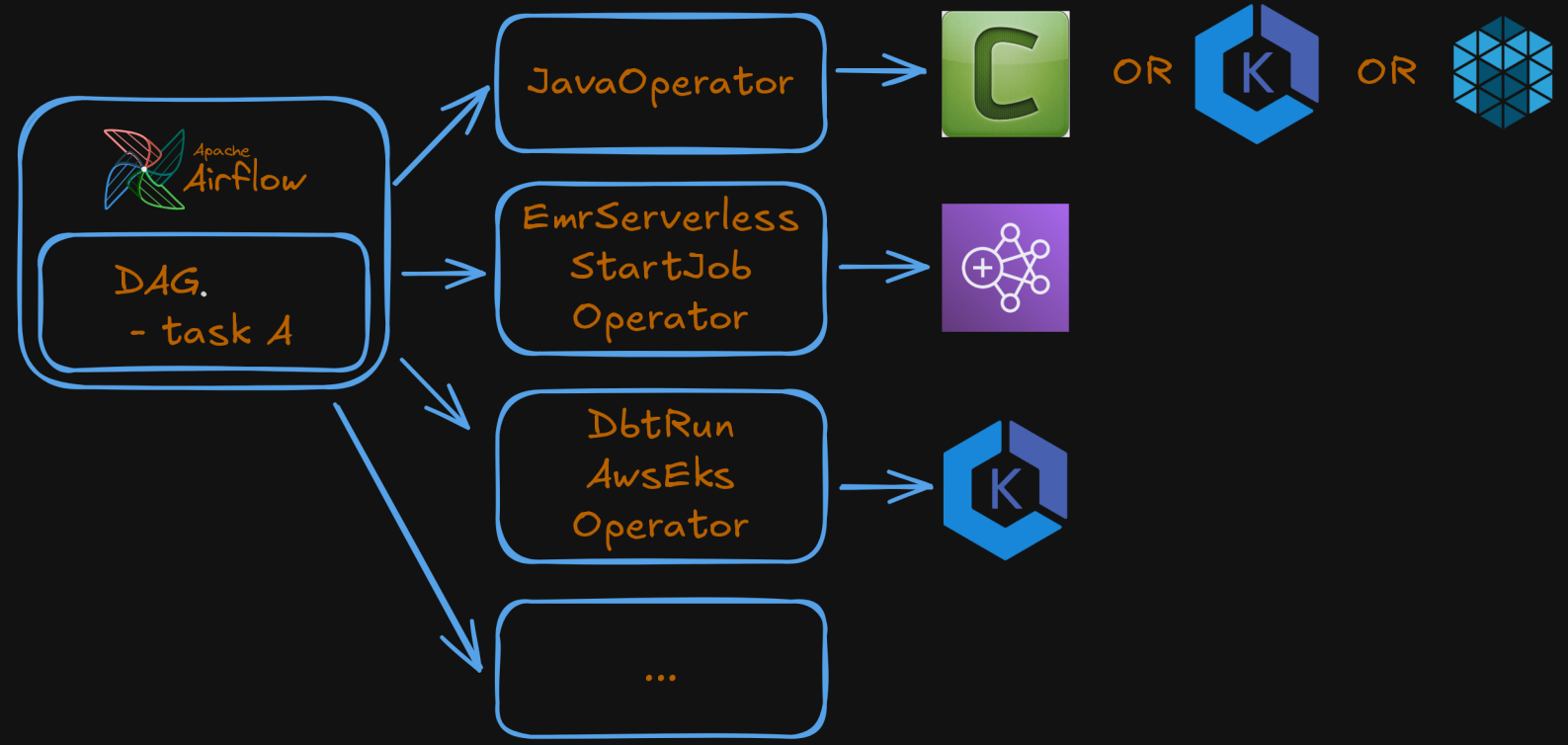


bronze

```
{
  "data": {
    "pk": 2,
    "aa": "text"
  },
  "op": "c",
  "ts_ms": 1287237862
}
```






Apache Airflow
DAG.
- task A
JavaOperator
EmrServerless StartJob Operator
DbtRun AwsEks Operator
...
OR
OR

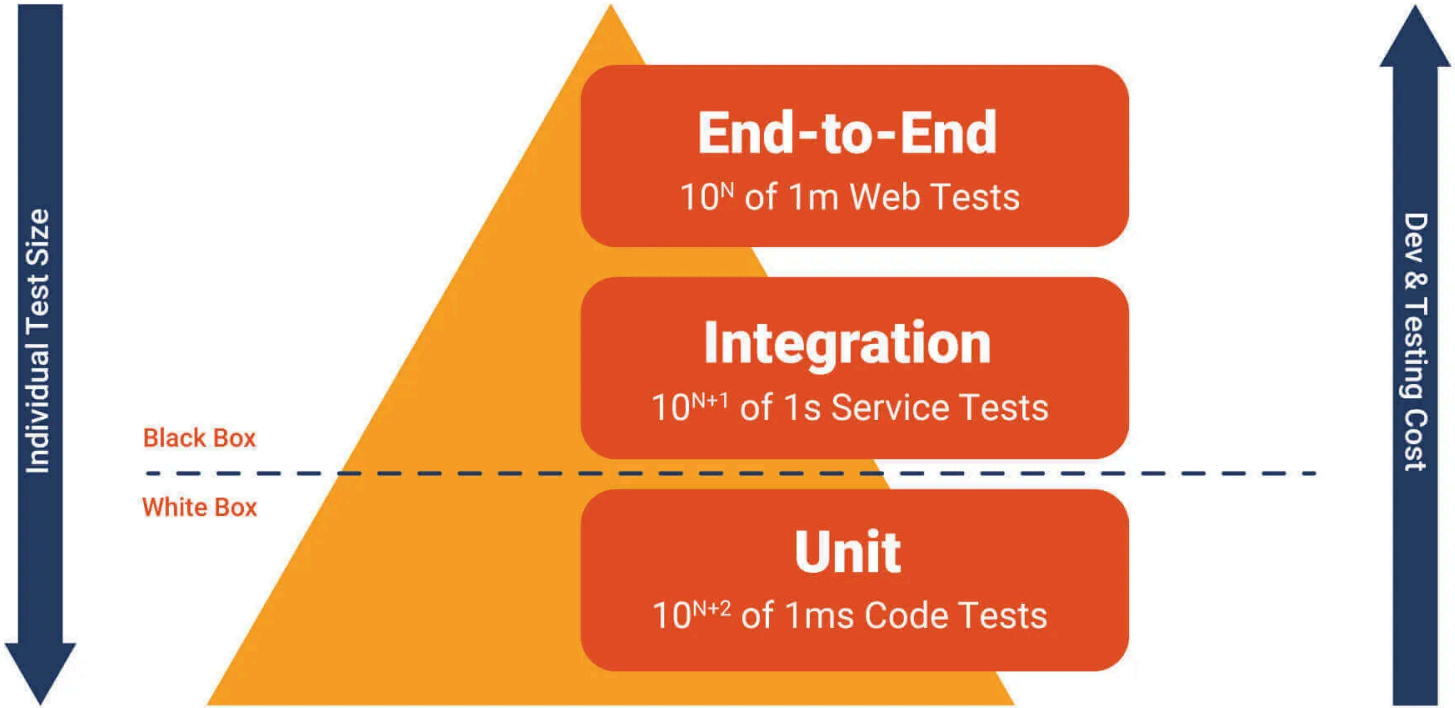
Who needs pipelines

- Data Scientists
- Data Analytics
- Data Engineers
- POs
- any data-driven person

It has to be tested

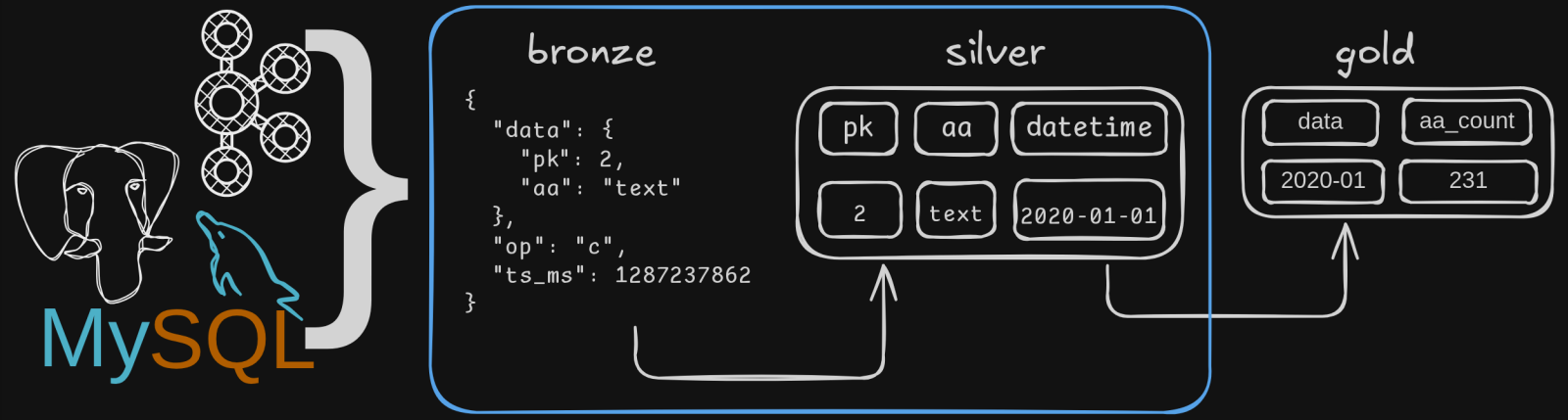
Pyramid of testing?

Pyramid of testing?


Individual Test Size
End-to-End
10^N of 1m Web Tests
Integration
10^N+1 of 1s Service Tests
Black Box
White Box
Unit
10^N+2 of 1ms Code Tests
Dev & Testing Cost

Pyramid of testing. Unit.

Bronze→Silver 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() // terminal 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() // terminal 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() // terminal 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() // terminal 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() // terminal operation
```

Unit testing of pipeline

What may we test here?

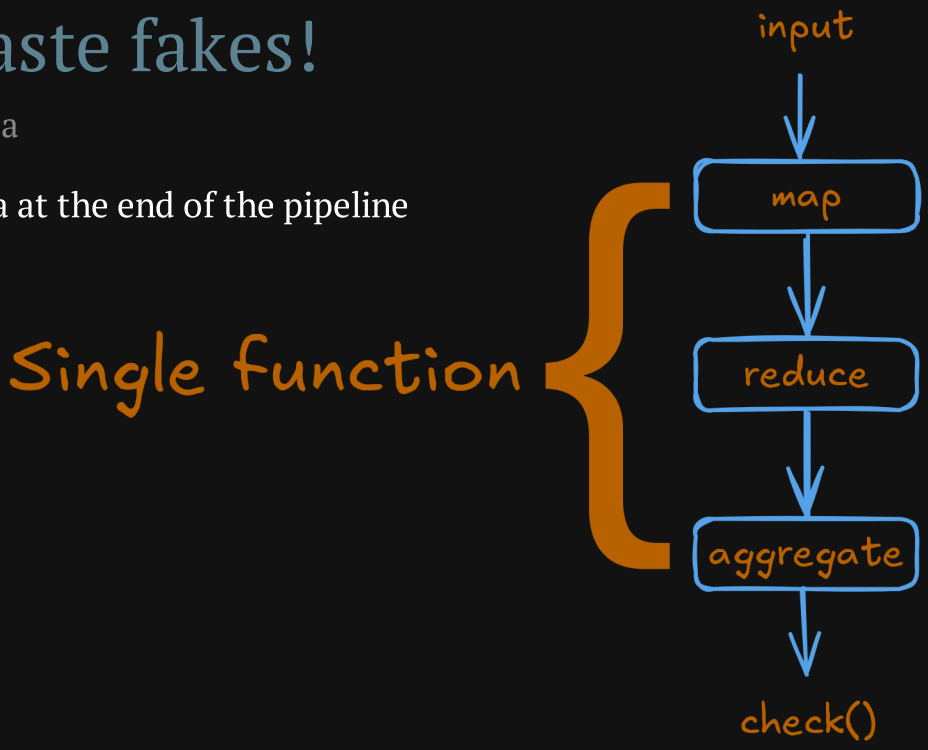
A pipeline should transform data correctly!

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

When Should You Run It?

On CI/CD

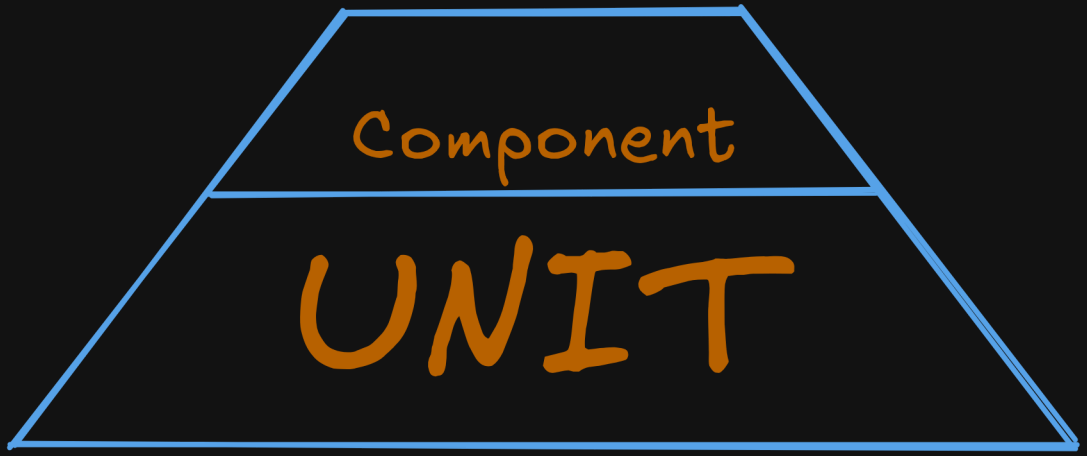
Things we usually forget to test

- empty source/target datasets,
- one scenario per one test

Things we usually forget after unit testing

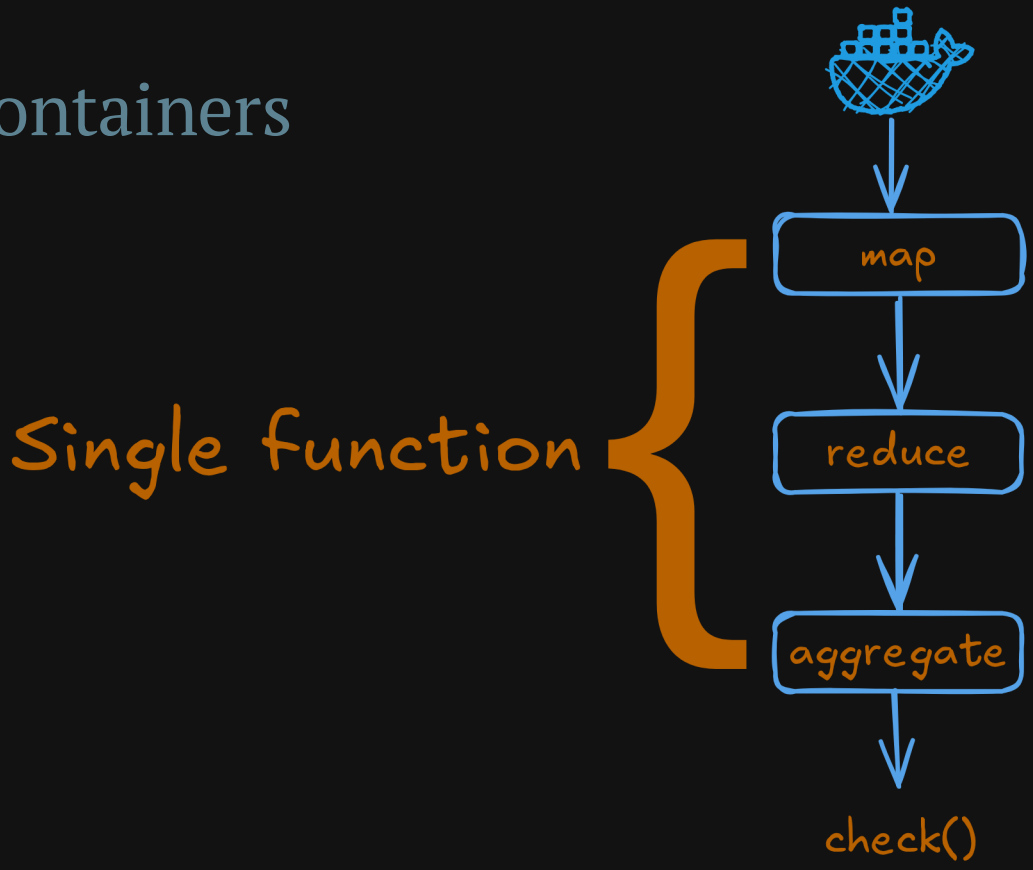
100% coverage doesn’t mean that we’re 100% safe

Component testing (or Integration testing)



TESTCONTAINERS

Testcontainers



Testcontainers

Supported languages:

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

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, passwor
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, passwor
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, password
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, passwor
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, passwor
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, passwor
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

Testcontainers

```
@Testcontainers
class PostgreSQLIntegrationTest {

    @Container
    static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>("postgres:17-alpine")
        .withDatabaseName("testdb")
        .withUsername("testuser")
        .withPassword("testpass");

    @Test
    void testDatabaseOperations() throws Exception {
        String jdbcUrl = postgres.getJdbcUrl();
        // omit here...
        try (Connection connection = DriverManager.getConnection(jdbcUrl, username, password
             Statement statement = connection.createStatement()) {
                ResultSet resultSet = statement.executeQuery("SELECT email FROM users WHERE
                assertTrue(resultSet.next());
                assertEquals("test@example.com", resultSet.getString("email"));
        }
```

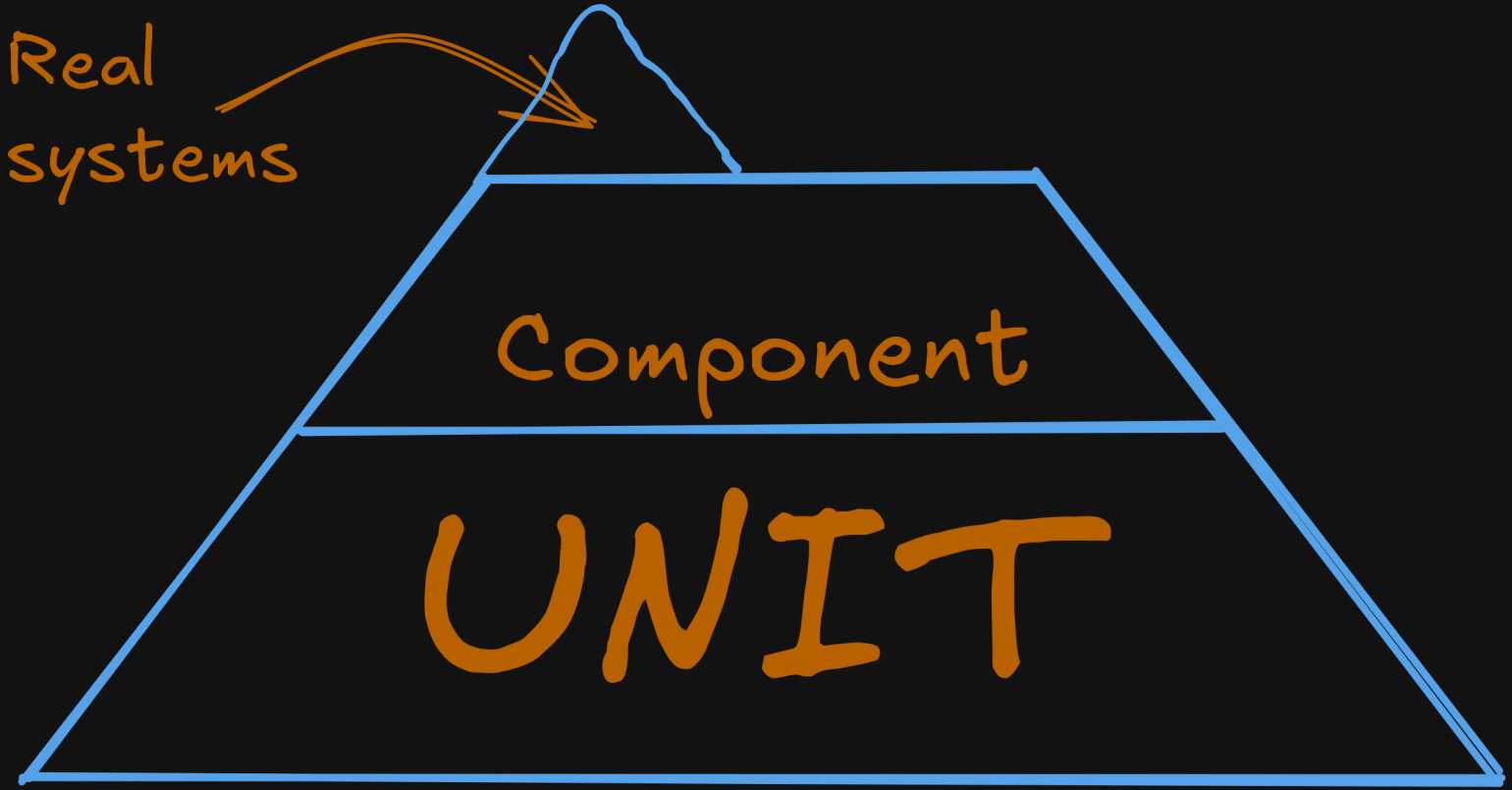
When Should You Run It?

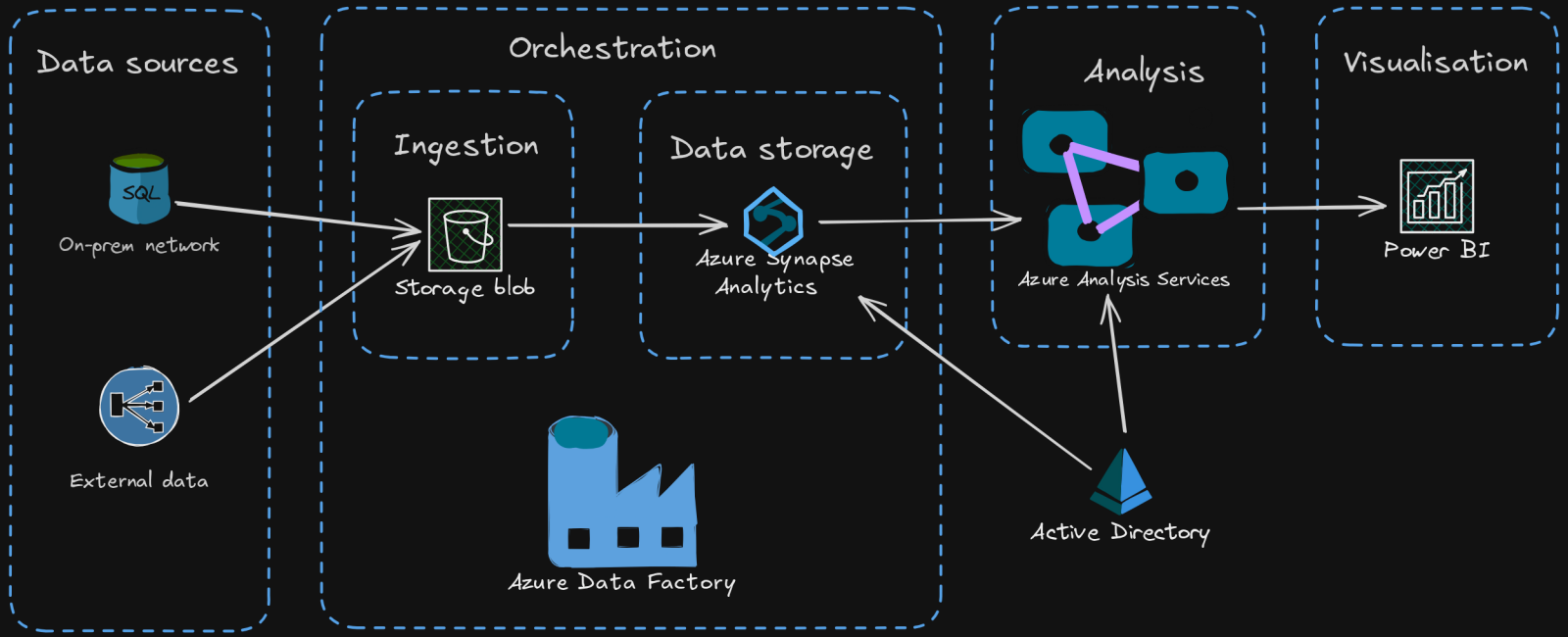
On CI/CD

consider file changes in specific files

Things we usually forget to test

- reruns (idempotency),
- data duplicates (inside/outside batch),
- incremental runs logic,
- metastore data (partitions, parameters, etc)

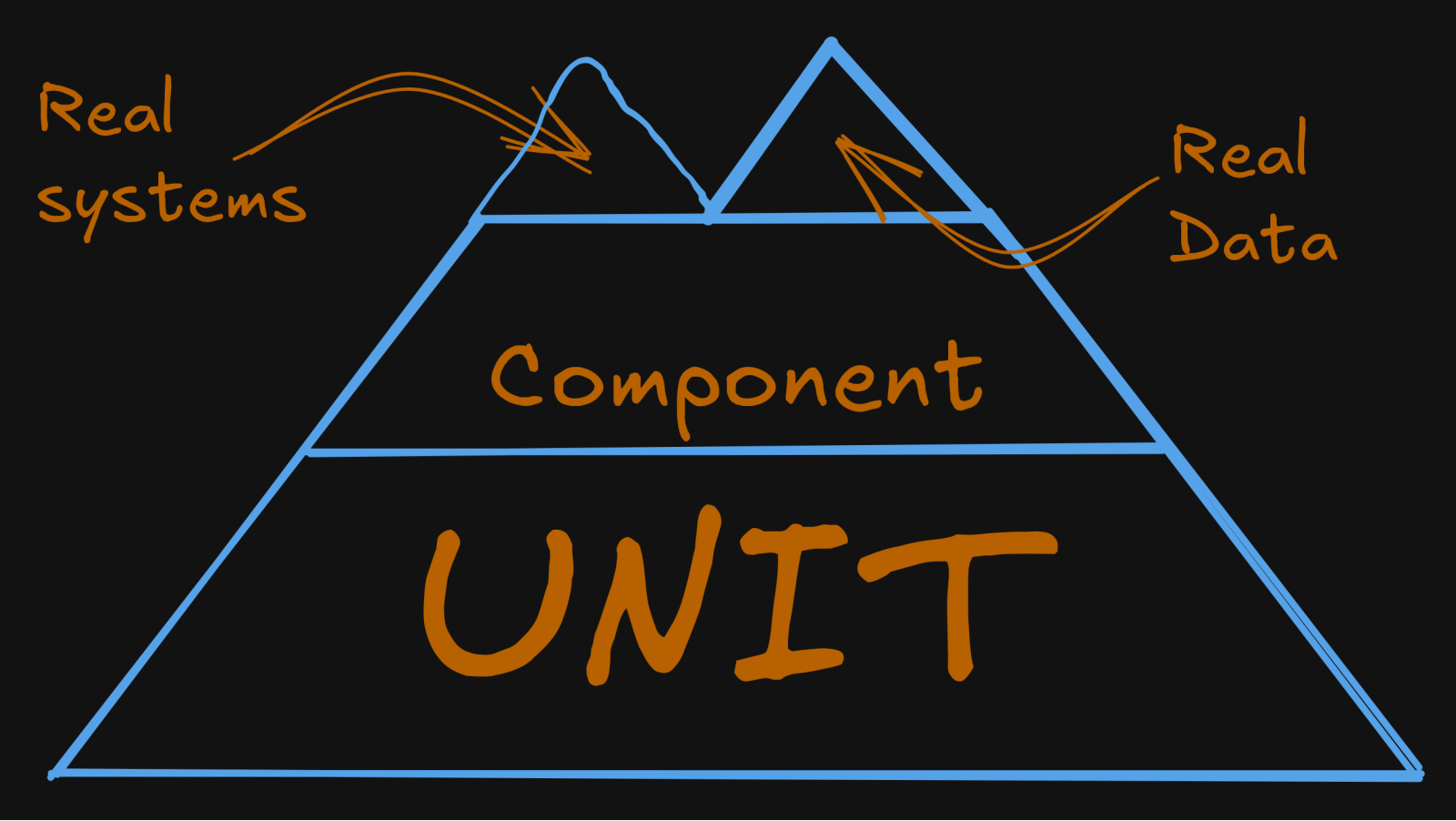

Real systems
Component
UNIT


Data sources
SQL
On-prem network
External data
Orchestration
Ingestion
Storage blob
Data storage
Azure Synapse Analytics
Azure Data Factory
Analysis
Azure Analysis Services
Active Directory
Visualisation
Power BI

Real systems

Why are component tests not enough?

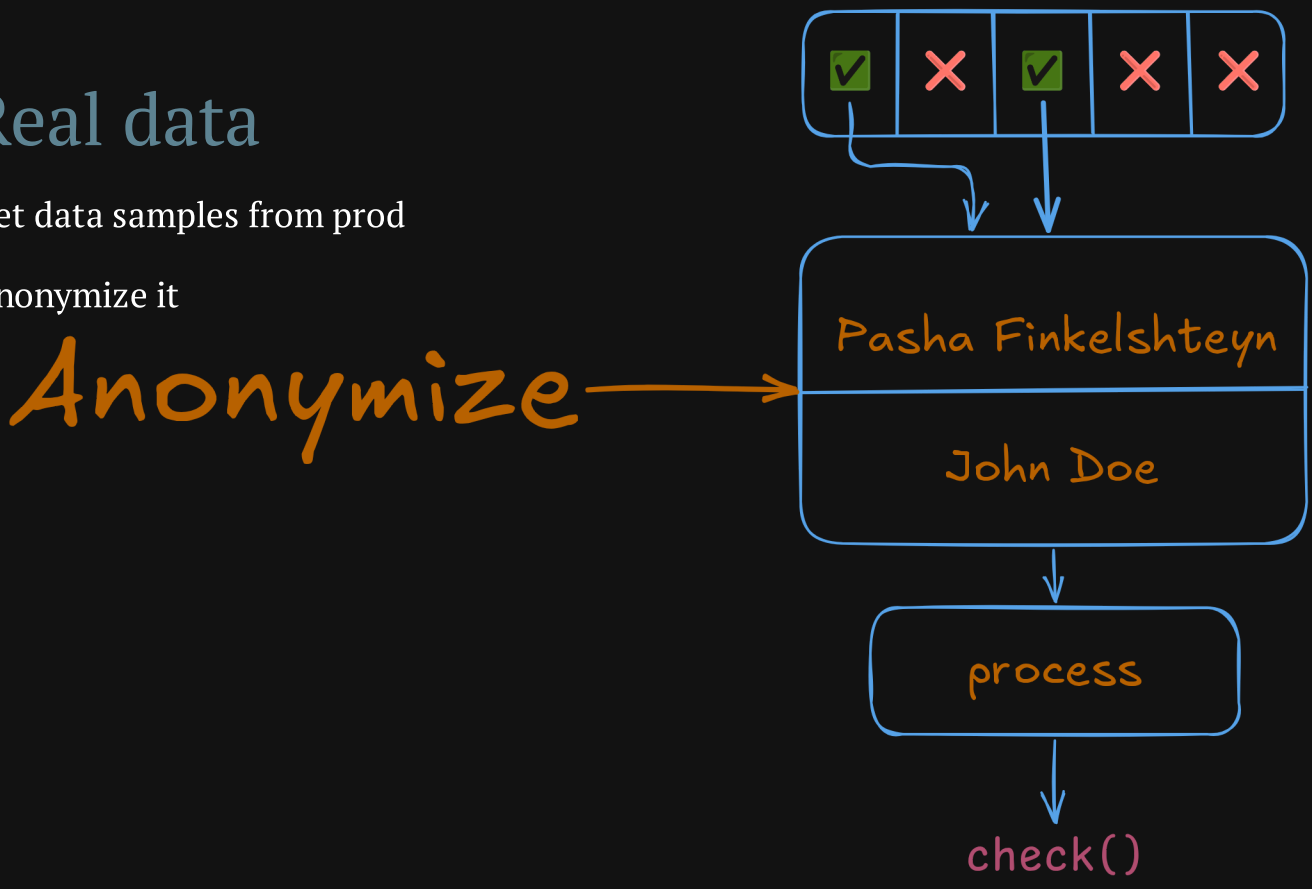
- vendor lock tools (DB, processing, etc.)
- external error handling
- access to data
- overall integration (especially for platform tooling)


Real systems
Real Data
Component
UNIT

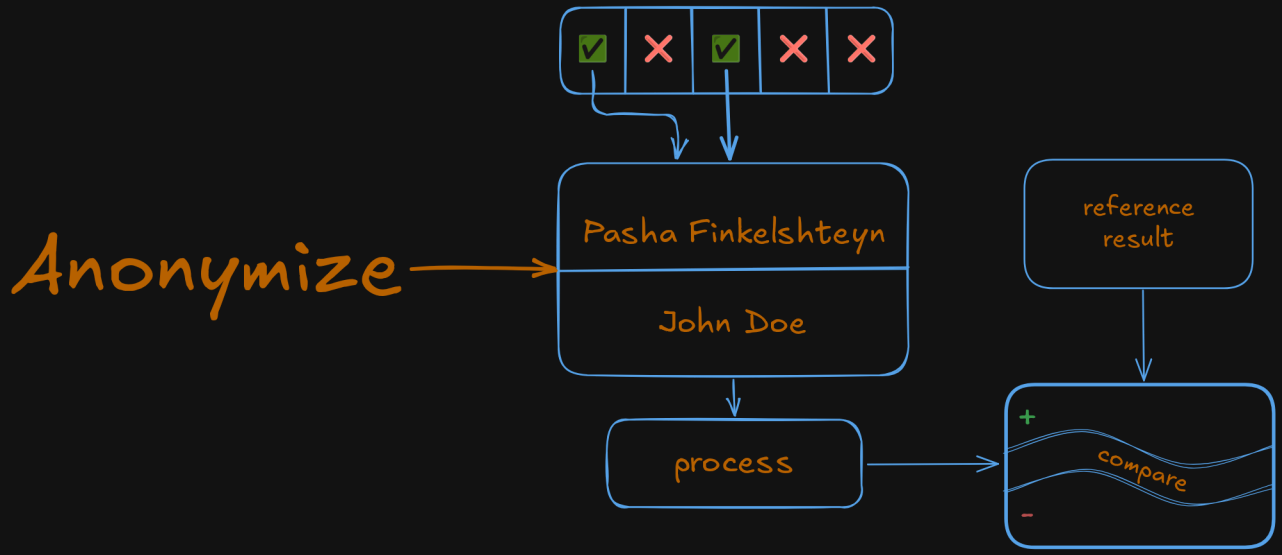
Real data

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 neither your code nor 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, StructureType
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(2025, 1, 1, 10, 0, 0), "DE"),
    Row(2, datetime(1000, 1, 1, 10, 0, 0), "NL"),
    Row(3, datetime(2023, 1, 1, 10, 0, 0), "NO"),
    Row(4, datetime(2024, 1, 1, 10, 0, 0), ""),
]

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

```python
from pyspark.sql.types import Row, StructureType
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(2025, 1, 1, 10, 0, 0), "DE"),
    Row(2, datetime(1000, 1, 1, 10, 0, 0), "NL"),
    Row(3, datetime(2023, 1, 1, 10, 0, 0), "NO"),
    Row(4, datetime(2024, 1, 1, 10, 0, 0), ""),
]

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

```python
from pyspark.sql.types import Row, StructureType
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(2025, 1, 1, 10, 0, 0), "DE"),
    Row(2, datetime(1000, 1, 1, 10, 0, 0), "NL"),
    Row(3, datetime(2023, 1, 1, 10, 0, 0), "NO"),
    Row(4, datetime(2024, 1, 1, 10, 0, 0), ""),
]

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

```python
from pyspark.sql.types import Row, StructureType
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(2025, 1, 1, 10, 0, 0), "DE"),
    Row(2, datetime(1000, 1, 1, 10, 0, 0), "NL"),
    Row(3, datetime(2023, 1, 1, 10, 0, 0), "NO"),
    Row(4, datetime(2024, 1, 1, 10, 0, 0), ""),
]

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

Great Expectations

```
{
  "result": {
    "element_count": 4,
    "unexpected_count": 2,
    "unexpected_percent": 50.0,
    "partial_unexpected_list": ["NO", ""]
  },
  "success": false,
  "expectation_config": {
    "kwargs": {
      "column": "Country",
      "value_set": ["DE", "NL"]
    }
  }
}
```

Great Expectations


```
{
  "result": {
    "element_count": 4,
    "unexpected_count": 2,
    "unexpected_percent": 50.0,
    "partial_unexpected_list": ["NO", ""]
  },
  "success": false,
  "expectation_config": {
    "kwargs": {
      "column": "Country",
      "value_set": ["DE", "NL"]
    }
  }
}
```

Great Expectations

```
{
  "result": {
    "element_count": 4,
    "unexpected_count": 2,
    "unexpected_percent": 50.0,
    "partial_unexpected_list": ["NO", ""]
  },
  "success": false,
  "expectation_config": {
    "kwargs": {
      "column": "Country",
      "value_set": ["DE", "NL"]
    }
  }
}
```

Great Expectations

```
{
  "result": {
    "element_count": 4,
    "unexpected_count": 2,
    "unexpected_percent": 50.0,
    "partial_unexpected_list": ["NO", ""]
  },
  "success": false,
  "expectation_config": {
    "kwargs": {
      "column": "Country",
      "value_set": ["DE", "NL"]
    }
  }
}
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ

```
SparkSession spark = SparkSession.builder().config(sparkConf).getOrCreate();

Dataset<Row> data = spark.sqlContext().read().parquet("/some/path/dataset");

Check check = new Check(spark, CheckLevel.Error(), "Review Check");

VerificationSuite checkResult = new VerificationSuite(spark)
    .onData(data)
    .addCheck(check
        .isComplete("Country")
        .isContainedIn("Country", List.of("DE", "NL")))
    .run();

verificationDf = VerificationResult.checkResultsAsDataFrame(spark, checkResult);
verificationDf.show()
```

Java Deequ. Results

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

Real data ecpectations. Use cases

Real data ecpectations. Use cases

- pre-ingestion and post-ingestion data validaton

Real data ecpectations. Use cases

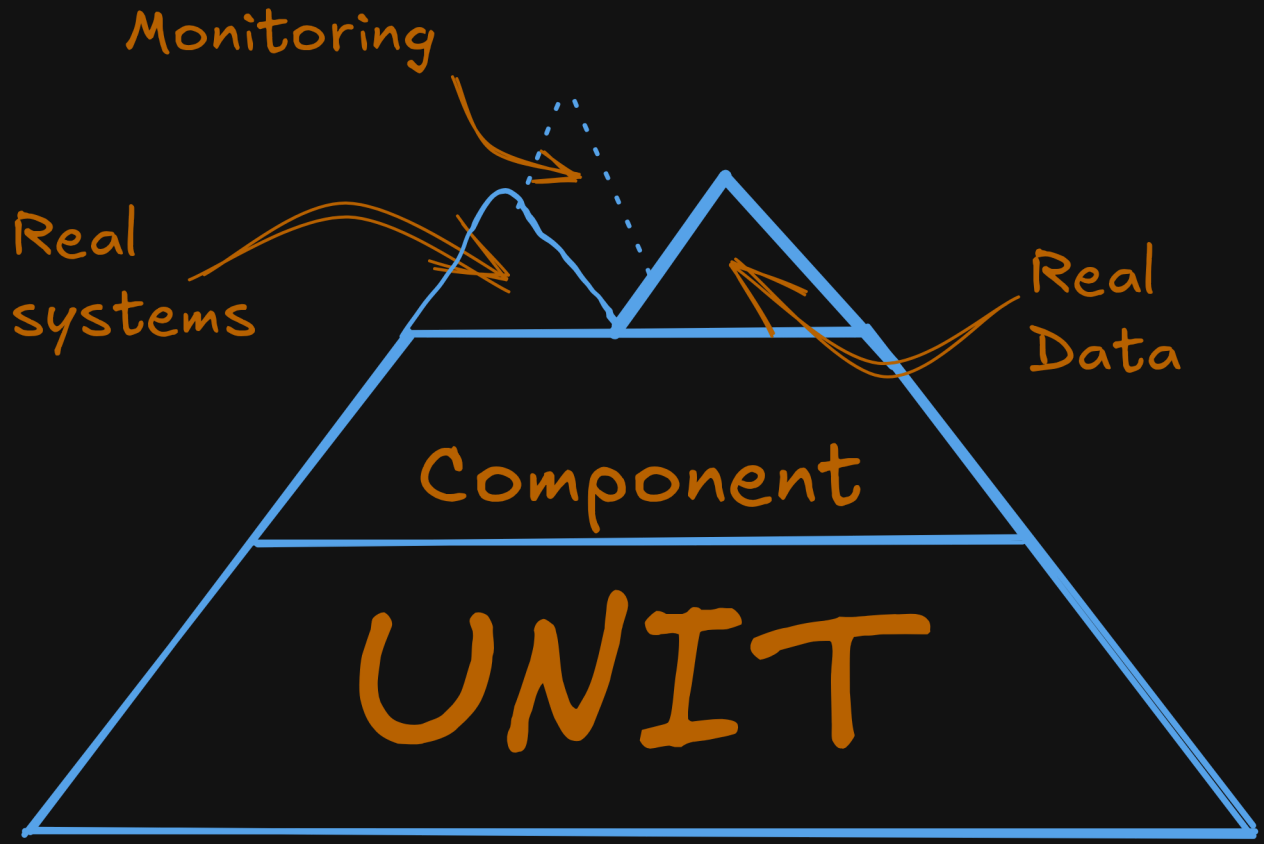
- pre-ingestion and post-ingestion data validaton
- before pipeline development

Real data ecpectations. Use cases

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

When Should You Run It?

In DAG run/on schedule


Monitoring
Real systems
Real Data
Component
UNIT

Monitoring

Why?

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

Monitoring

What?

Monitoring

What?

- data volumes

Monitoring

What?

- data volumes
- time (SLAs)

Monitoring

What?

- data volumes
- time (SLAs)
- dead letter queue monitoring

Monitoring

What?

- data volumes
- time (SLAs)
- dead letter queue monitoring
- service health

Monitoring

What?

- data volumes
- time (SLAs)
- dead letter queue monitoring
- service health
- business metrics

Monitoring

How?

Monitoring

How?

- use Listeners

Monitoring

How?

- use Listeners
- Data Aggregators

Monitoring

How?

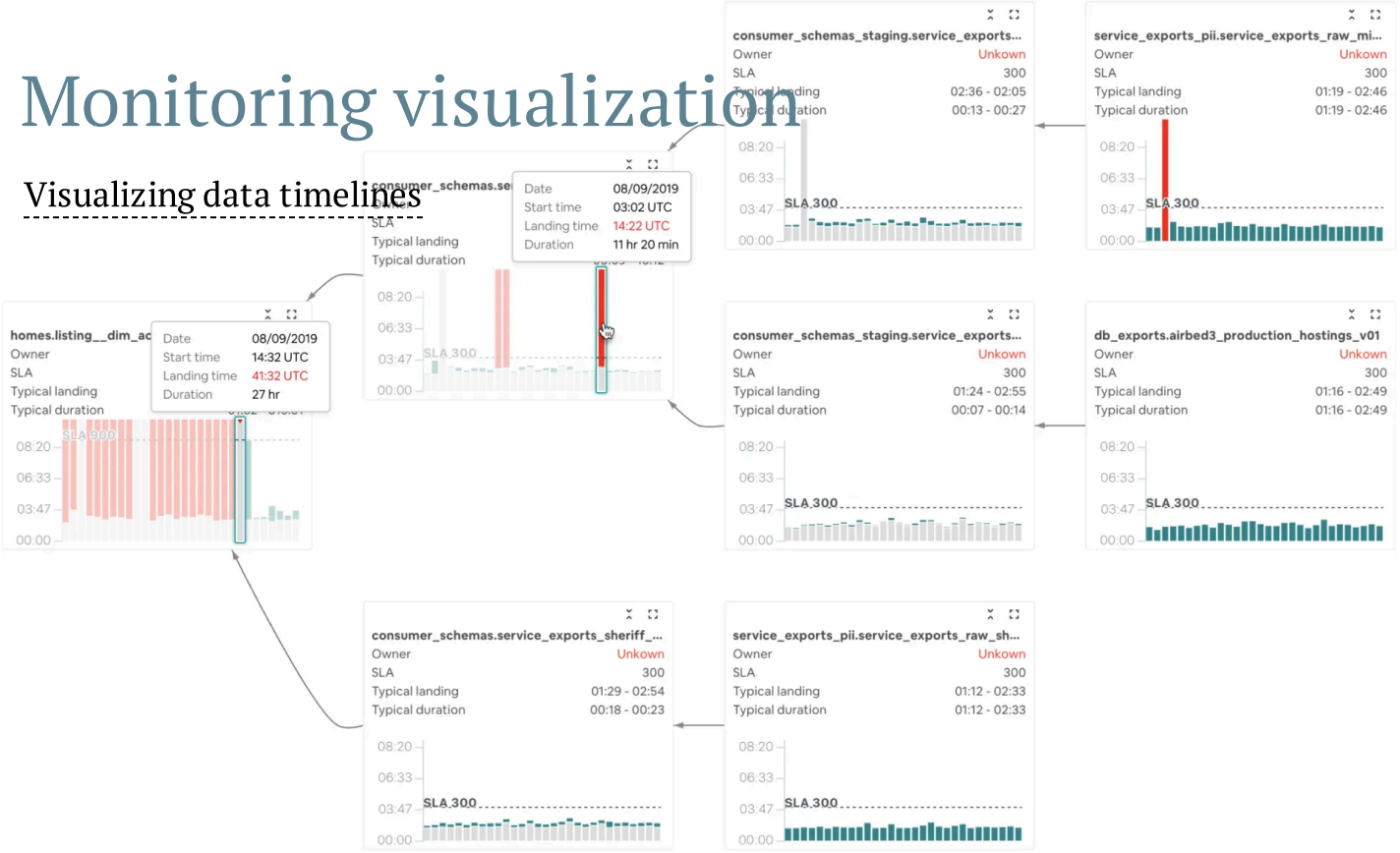
- use Listeners
- Data Aggregators
- Airflow (Dagster, Prefect, etc)

Data pipeline is always a DAG

Monitoring should visualize it

Monitoring visualization

Visualizing data timelines

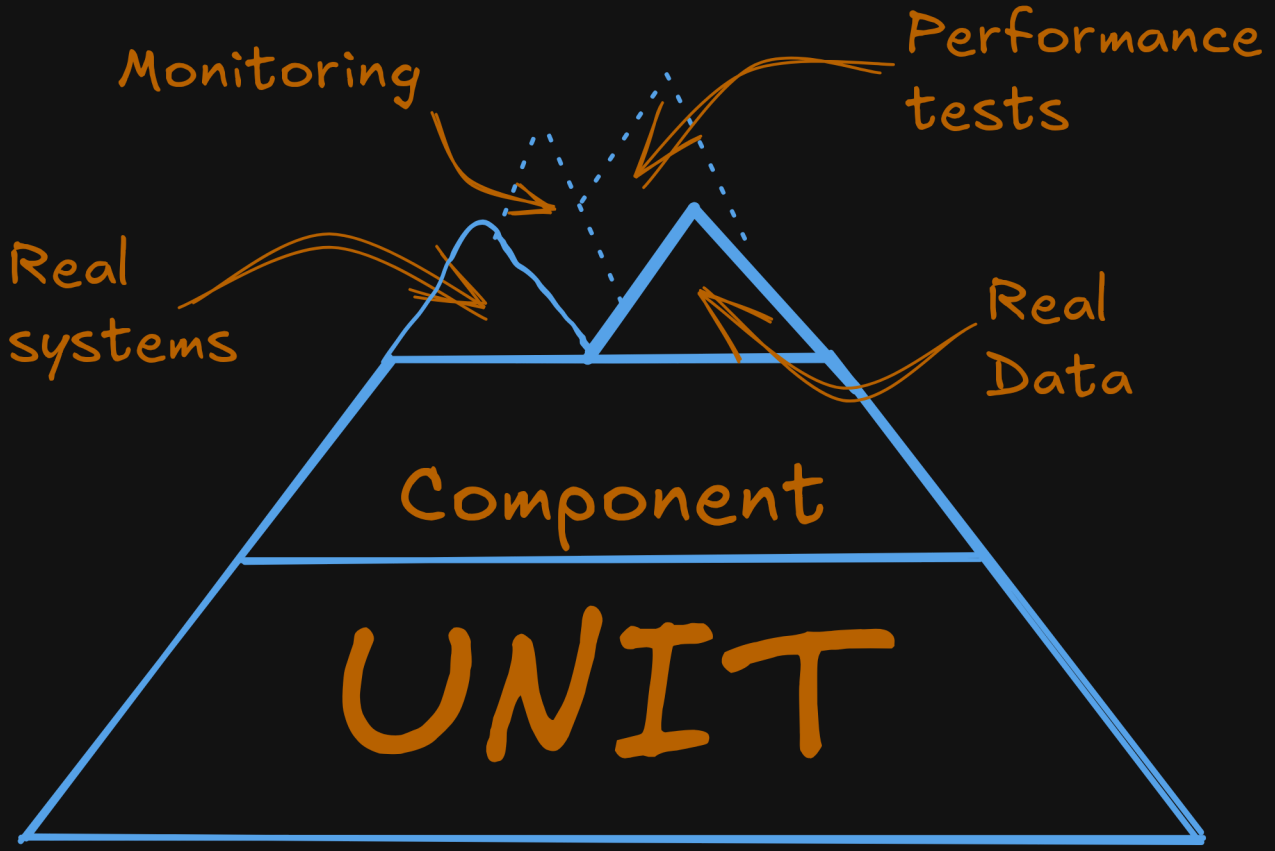


End-to-End tests

Compare with reports, old DWH

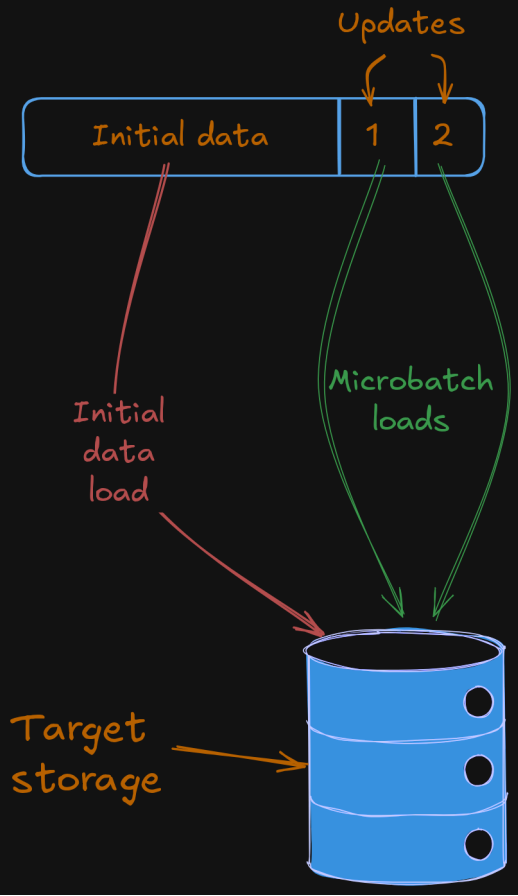
Multiple dimensions:

- data
- data latency
- performance, scalability


Monitoring
Performance tests
Real systems
Real Data
Component
UNIT

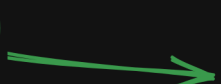
Performance Tests

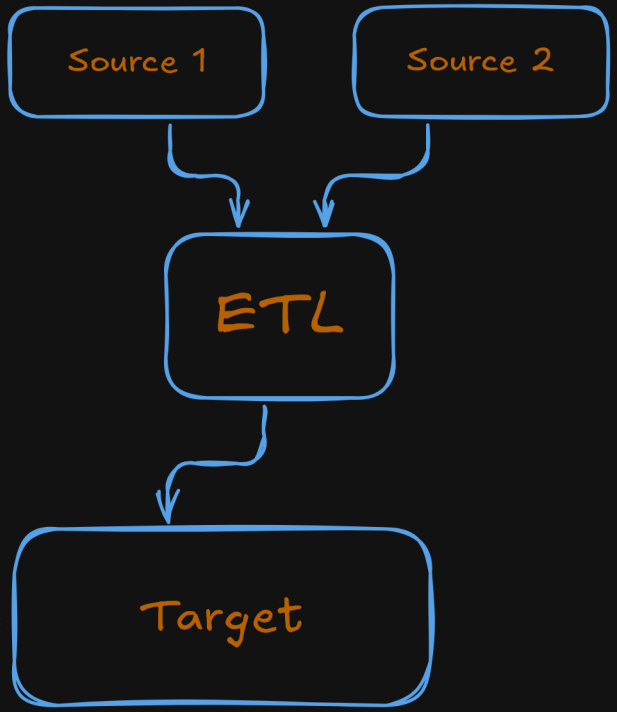
- start with SLO
- test your initial data load

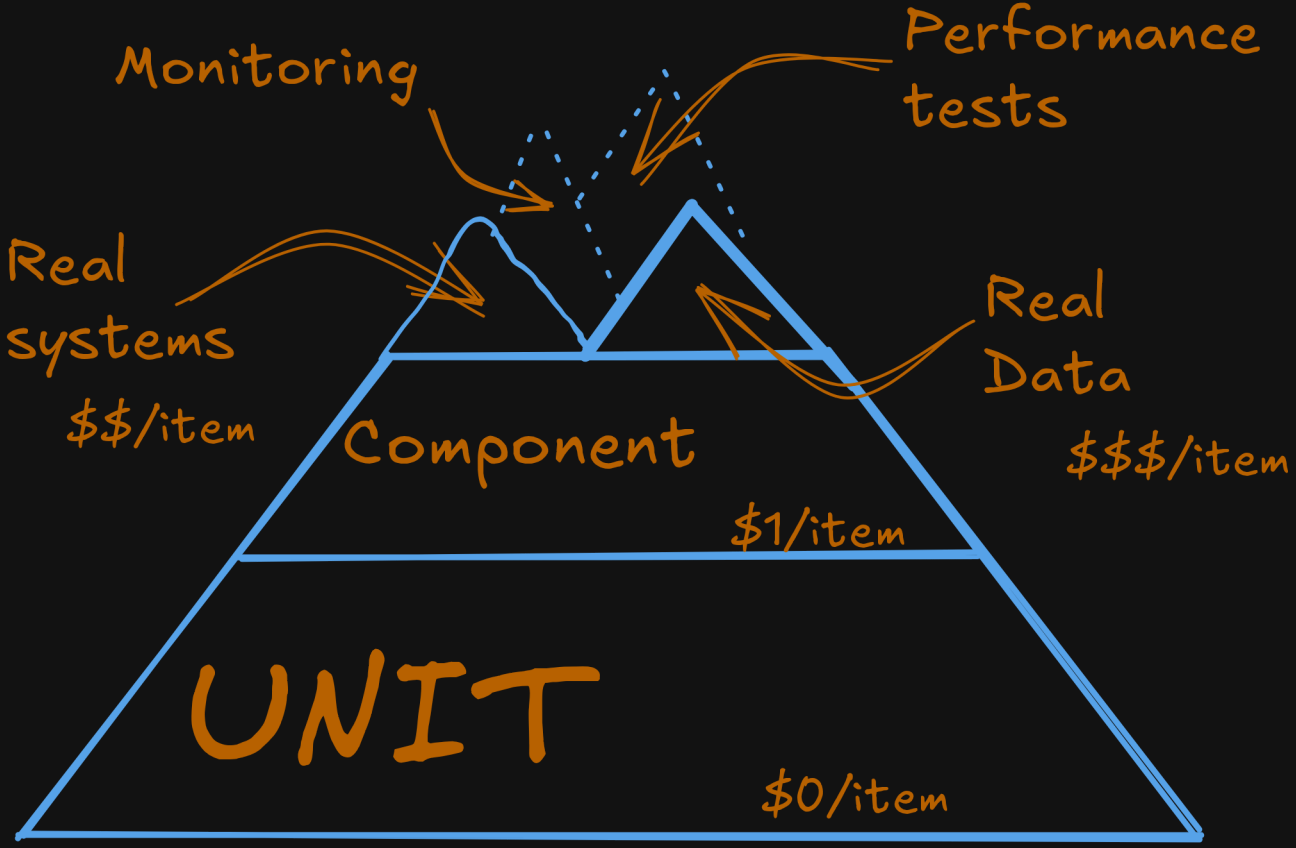


Real prod

Run a parallel job with a different sink






Monitoring
Performance tests
Real systems
$$/item
Real Data
$$$/item
Component
$1/item
UNIT
$0/item

Summary

Summary

- Testing pipeline is like testing code

Summary

- Testing pipeline is like testing code
- Testing pipelines is not like testing code

Summary

- Testing pipeline is like testing code
- Testing pipelines is not like testing code
- Pipeline quality is not only about testing

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

END