Return to Data's Inferno

Are the 7 layers of data testing hell still relevant?

Daniel van der Ende

Xebia

About me

Data Engineer at Xebia Data (formerly GoDataDriven)

Likes:

- Automation
- Robust data pipelines
- Memes (sorry)

Dislikes:

- Bad data
- Manual processes
- Data managed in Excel sheets

For the uninitiated

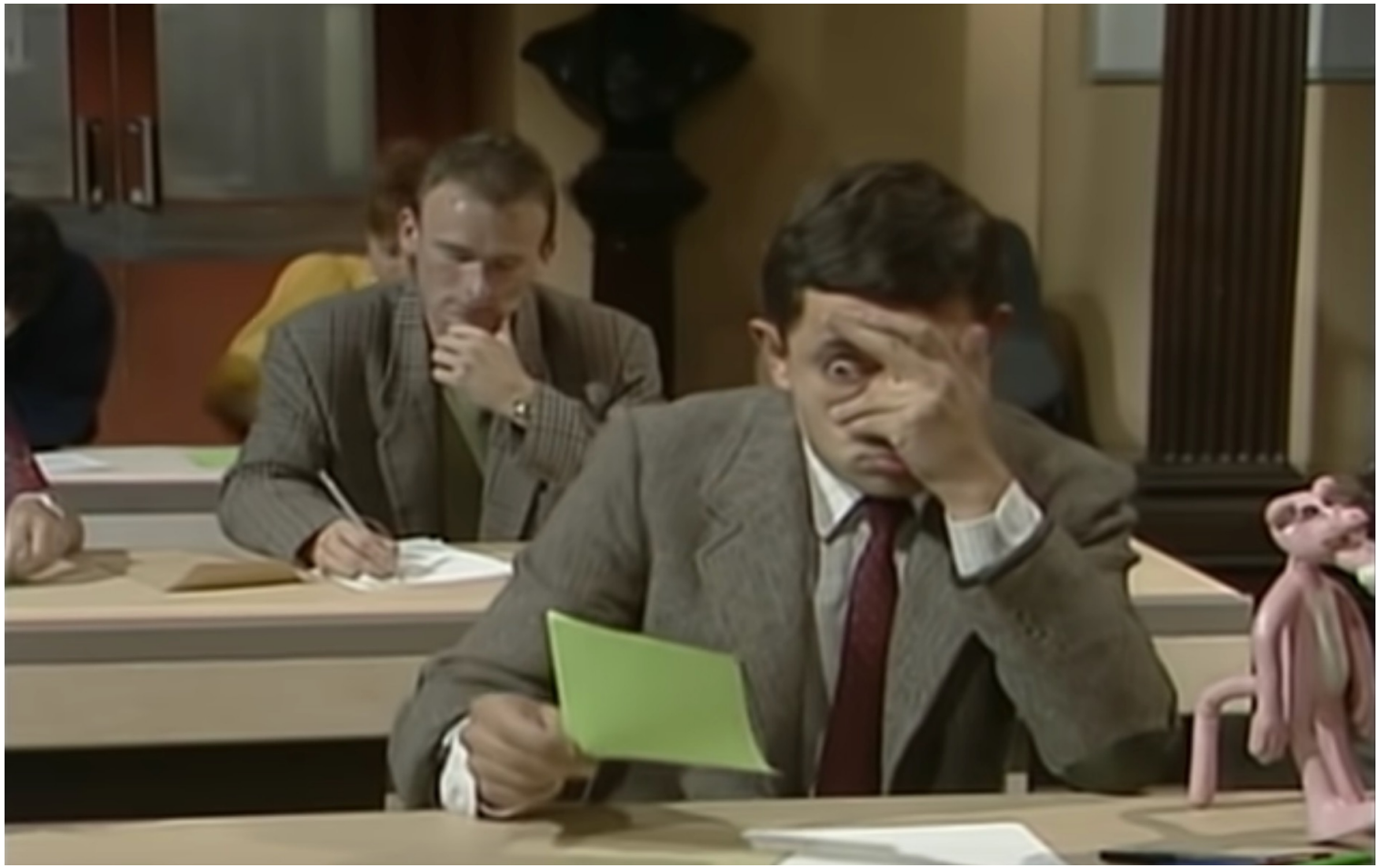
Meetup & Conference Talk in 2017/2018

Accompanying blogpost

As well as a Github repository

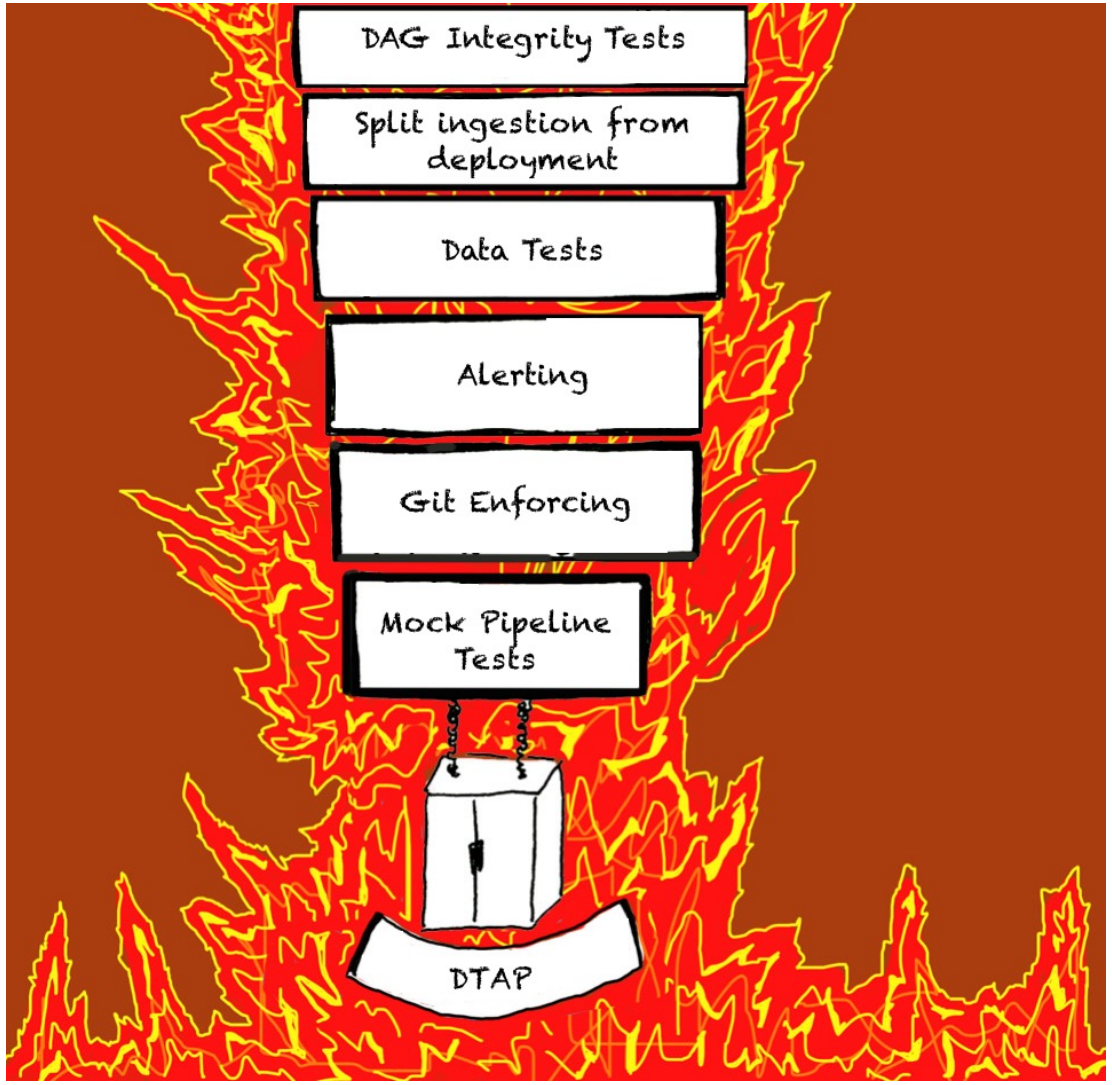
Data Quality

A trip down memory lane



Our first stop: the environment

Data's Inferno



Layer 1: Dag Integrity Tests

```
task_a = BashOperator(task_id='task_a', …)

test_task_a = BashOperator(task_id='test_task_a', …)

task_b = BashOperator(task_id='task_a', …)
```

```
714                 msg = "Cycle detected in DAG. Faulty task: {0}".format(task)
715 >               raise AirflowException(msg)
716 E               airflow.exceptions.AirflowException: Cycle detected in DAG. Faulty task: <Task(BashOperator): templated>
717
718 ../../../virtualenv/python3.5.3/lib/python3.5/site-packages/airflow/models.py:2364: AirflowException
719 ========================== 1 failed in 0.83 seconds ==========================
720
```

Layer 1: Dag Integrity Tests

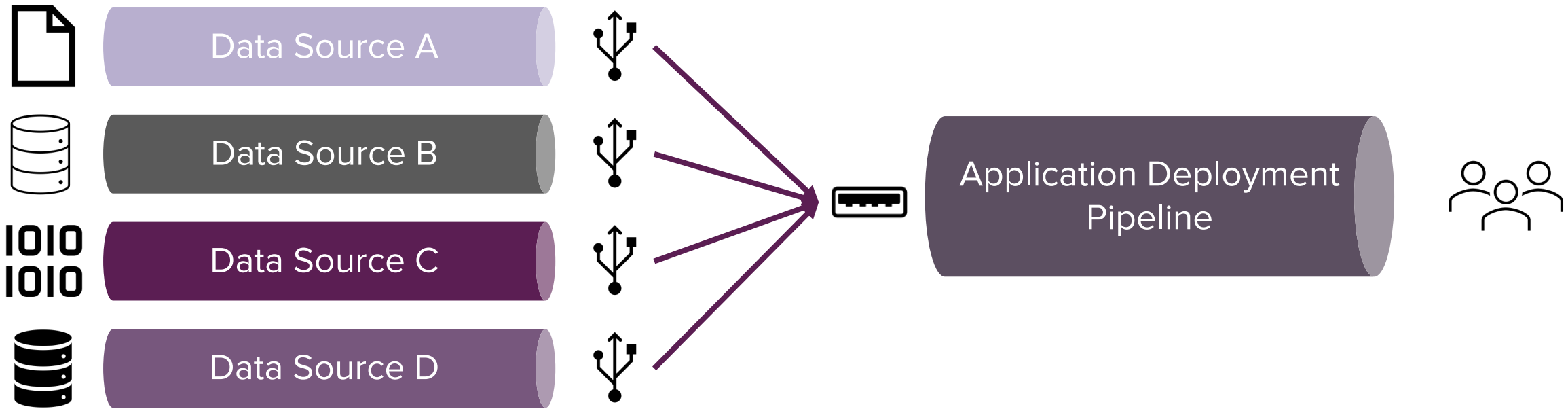
```
for all *.py in /dags/:
    objects = load_all_objects_from_files()
    dag_objects = filter_objects()

    if no dag_objects:
      raise Error("No DAGs detected")

    for dag in dag_objects:
        check_cycle(dag)
```

Full code available at:
github.com/danielvdende/data-testing-with-airflow/

Layer 1: Dag Integrity Tests

Still Relevant in 2023?

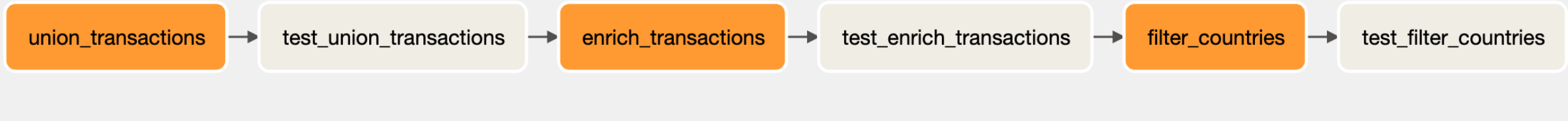
Layer 2: Split Ingestion from Deployments



Layer 2: Split Ingestion from Deployments

Still Relevant in 2023?

Layer 3: Data Tests



Layer 3: Data Tests

In 2017...

```
spark.sql("""
    SELECT
        t.dt                                dt,
        t.amount                            amount,
        t.payer_account                     payer_account,
        pa.name                             payer_name,
        pa.country                          payer_country,
        t.beneficiary_account               beneficiary_account,
        ba.name                             beneficiary_name,
        ba.country                          beneficiary_country
    FROM transactions_union t
    LEFT JOIN account_info pa ON t.payer_account = pa.account
    LEFT JOIN account_info ba ON t.beneficiary_account = ba.account
    """).write \
    .saveAsTable('bank.enriched_transactions', format='parquet', mode='overwrite')
```

Layer 3: Data Tests

In 2017...

```
assert spark.sql("""
SELECT
SUM(IF(payer_country IS NULL OR beneficiary_country IS NULL,1,0)) null_count
FROM enrich_transactions
""").first().null_count == 0
```

Layer 3: Data Tests

In 2023...

```
SELECT
    t.dt                                    dt,
    t.amount                                amount,
    t.payer_account                         payer_account,
    pa.name                                 payer_name,
    pa.country                              payer_country,
    t.beneficiary_account                   beneficiary_account,
    ba.name                                 beneficiary_name,
    ba.country                              beneficiary_country
FROM bank.transactions_union t
LEFT JOIN bank.account_info pa ON t.payer_account = pa.account
LEFT JOIN bank.account_info ba ON t.beneficiary_account = ba.account
```

Layer 3: Data Tests

In 2023...

```sql
SELECT null_count
FROM (
    SELECT COUNT(*) AS null_count
    FROM bank.enriched_transactions
    WHERE payer_country IS NULL OR beneficiary_country IS NULL
)
WHERE null_count >0
```

Layer 3: Data Tests

Still Relevant in 2023?

Layer 4: Alerting

Chuck Norris APP 10:28
domino_data_deployment_dev Failure! Task failed: <Task(SparkSubmitOperator): name_matching> Check log at: manual__2017-12-13T16:04:44

daniel 11:49
@aerdem @jonas ^

Chuck Norris APP 12:21
domino_data_deployment_dev Failure! Task failed: <Task(BashOperator): promote_branch_to_tst> Check log at: manual__2017-12-13T16:04:44

Chuck Norris APP 14:04
grid_ingestion Failure! Task failed: <Task(BashOperator): pyspark_data_test> Check log at: manual__2017-12-14T10:59:02.592838

Chuck Norris APP 15:02
peer_detection_dev Failure! Task failed: <Task(SparkSubmitOperator): precompute_customer_supplier_peers> Check log at: manual__2017-11-27T16:00:47.332361

Chuck Norris APP 15:25
domino_data_deployment_dev Failure! Task failed: <Task(SparkSubmitOperator): name_matching> Check log at: None

Chuck Norris APP 15:32
peer_detection_dev Failure! Task failed: <Task(SparkSubmitOperator): precompute_customer_supplier_pe
domino_data_deployment_dev Failure! Task failed: <Task(SparkSubmitOperator): name_matching> Check

Chuck Norris APP 16:10
domino_data_deployment_dev Failure! Task failed: <Task(SparkJDBCOperator): sqark_company_info> Che
domino_data_deployment_dev Failure! Task failed: <Task(SparkSubmitOperator): name_matching> Check

Chuck Norris APP 16:22
domino_data_deployment_dev Failure! Task failed: <Task(SparkSubmitOperator): name_matching> Check

1 reply 4 days ago

Chuck Norris APP 18:34
domino_data_deployment_dev Failure! Task failed: <Task(BashOperator): promote_branch_to_tst> Check

Layer 4: Alerting

```
def slack_alert(message):
  ...
  slack_cmd = f"curl -x proxy:port -X POST --data-urlencode '{message}' {url}"

  slack_alert = BashOperator(
      task_id='slack_alert',
      dag=dag,
      bash_command=slack_cmd,
  )
  slack_alert.execute(context)

BashOperator(
    task_id="mytask",
    on_failure_callback=[
        slack_alert({
            text="The task {{ ti.task_id }} failed",
            channel="#general",
            username="Chuck Norris",
        })
    ],
    bash_command="fail",
)
```

In 2017...

This was home-built...

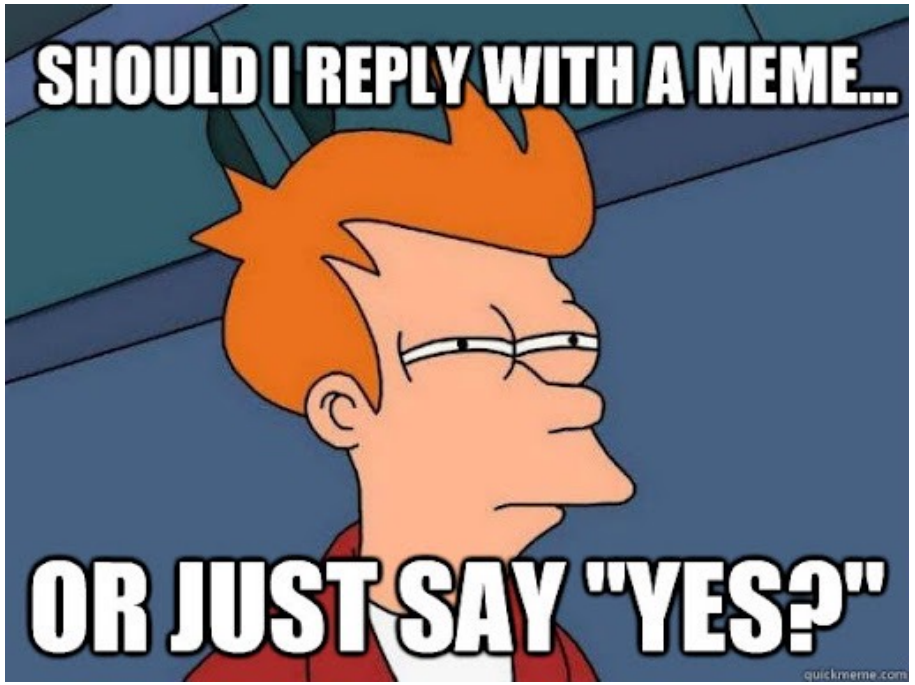
Layer 4: Alerting

In 2023...

```
from airflow.providers.slack.notifications.slack import send_slack_notification

BashOperator(
    task_id="mytask",
    on_failure_callback=[
        send_slack_notification(
            text="The task {{ ti.task_id }} failed",
            channel="#general",
            username="Chuck Norris",
        )
    ],
    bash_command="fail",
)
```

Layer 4: Alerting

Still Relevant in 2023?

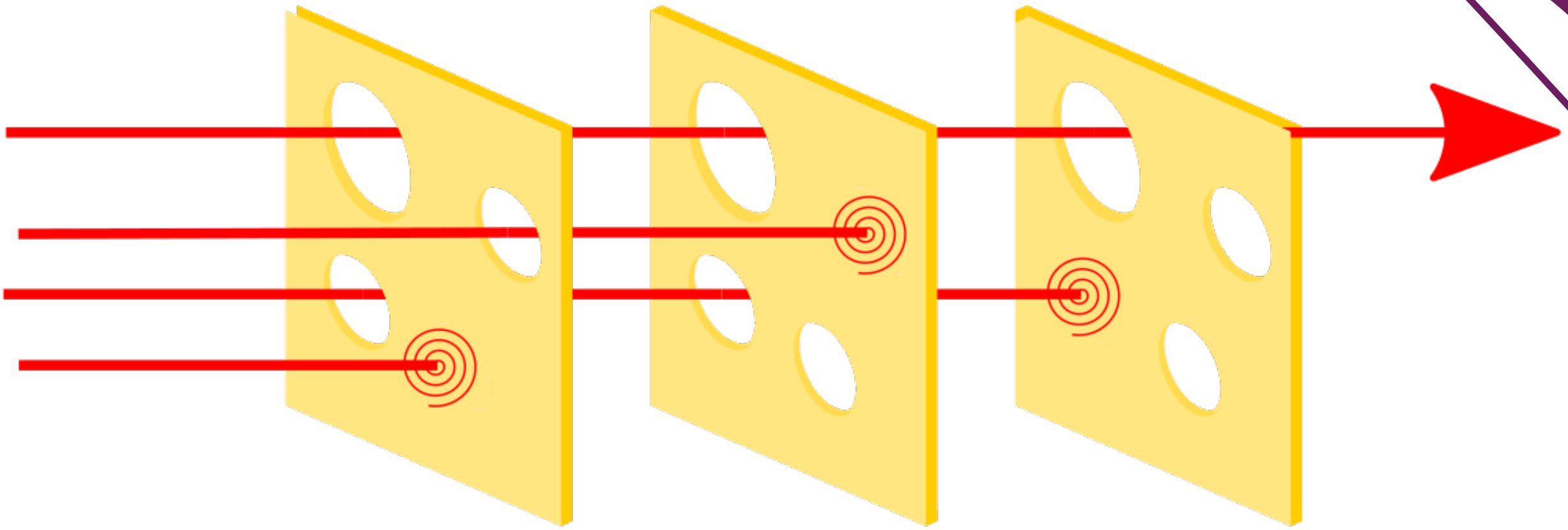


Layer 5: Git Enforcing

Layer 5: Git Enforcing

Still Relevant in 2023?

Layer 6: Mock Pipeline Tests



Layer 6: Mock Pipeline Tests

In 2017

```
PERSONS =  [('name': 'Donald Duck', 'country': 'Duckburg', 'iban': 'DB99DBBK0000000313'), …]
TRANSACTIONS= [('iban': 'DB99DBBK0000000313', 'amount': 10 ), …]
```

```
filter_data(persons, transactions)
```

```
assert spark.sql("""SELECT COUNT(*) ct FROM filtered_data WHERE iban = 'DB99DBBK0000000313'""").first().ct == 0
```

Layer 6: Mock Pipeline Tests

In 2023

```
PERSONS =  [('name': 'Donald Duck', 'country': 'Duckburg', 'iban': 'DB99DBBK0000000313'), …]
TRANSACTIONS= [('iban': 'DB99DBBK0000000313', 'amount': 10 ), …]
```

```
dbt test --target mock_pipeline
```

```
SELECT *
FROM filtered_data
WHERE iban = 'DB99DBBK0000000313'
```

Layer 6: Mock Pipeline Tests

Still Relevant in 2023?

Layer 7: DTAP

Separate environments with different data, but the same code

DEV

- Quickly run your pipeline on a very small subset of your data
- In our case 0.0025% of all data
- Nothing will make sense, but it's a nice integration test

TST

- Select a subset of your data for data that you know
- Immediately see if something is off
- Still quick to run

ACC

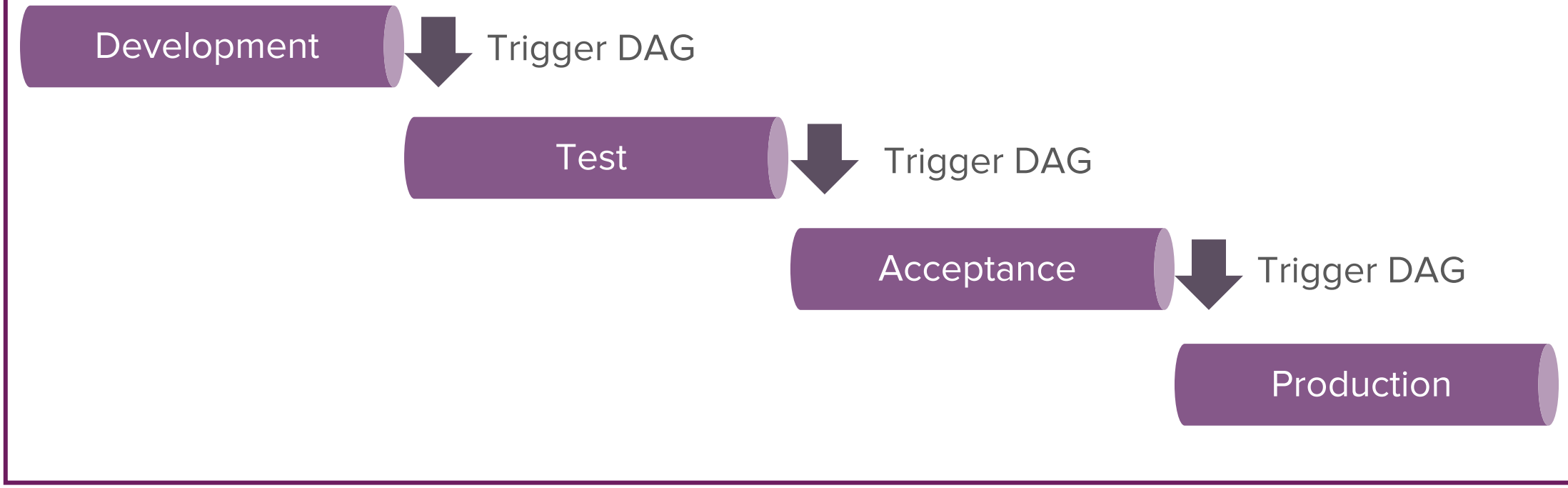
- Carbon copy of production
- You can check if you feel comfortable pushing to PRD
- Give access to a Product Owner for them to check

PRD

- Greenlight procedure for merging from ACC to PRD
- Manual operation

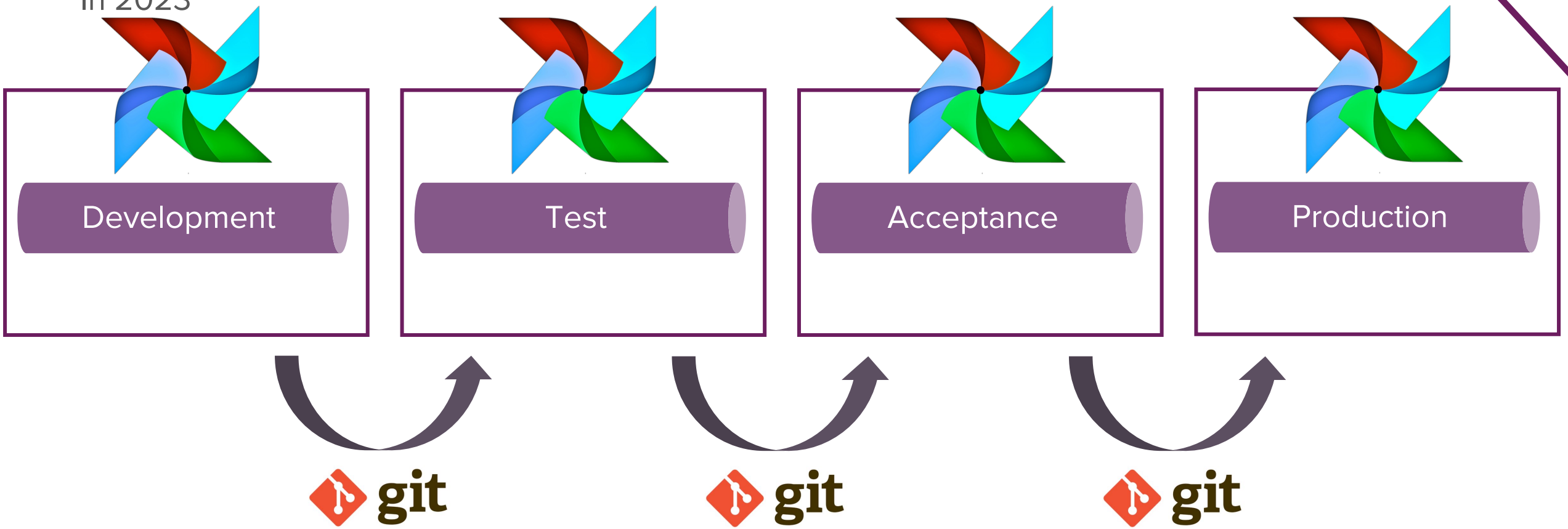
Layer 7: DTAP

In 2017



Layer 7: DTAP

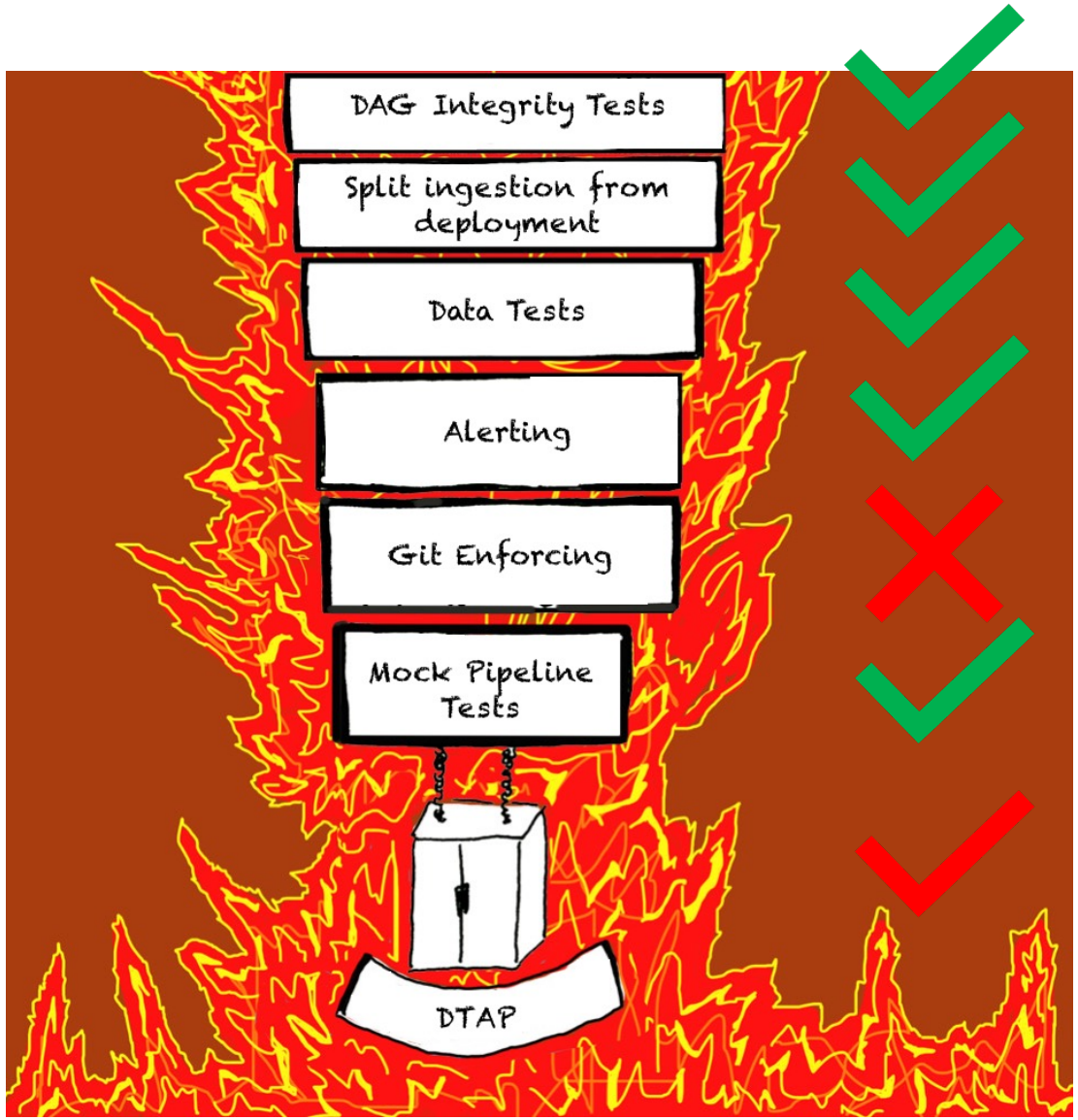
In 2023



Layer 7: DTAP

Still Relevant in 2023?

In Summary...


DAG Integrity Tests
Split ingestion from deployment
Data Tests
Alerting
Git Enforcing
Mock Pipeline Tests
DTAP

Taking a step back

- Code Integrity/Validity
- Pipeline Structure
- Data Quality
- Alerting
- Separation of Concerns

Can we now leave hell?

Working with real data can still be really hard:

- It’s hard to know what to test.
- It’s hard to always catch all errors.
- It always springs surprises on you

This was the case in 2017, and it’s still the case now.

The tooling in 2023 has improved dramatically though, making this a lot easier

Thank you!

github.com/danielvdende/data-testing-with-airflow