Halting Bad Data in Its Tracks

Real-time Quality Management
for Data Mesh

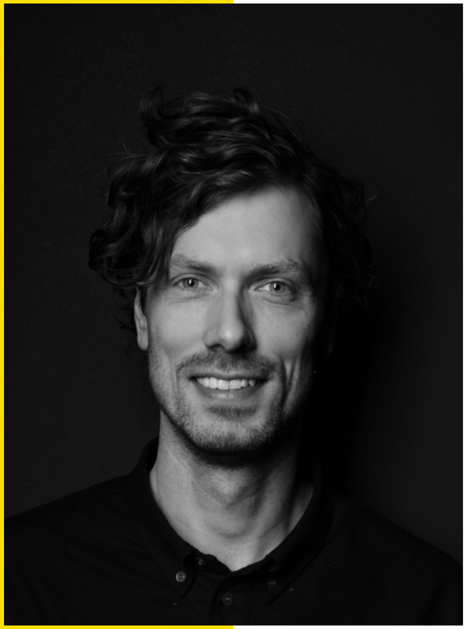
Reach out

Martijn Beenker

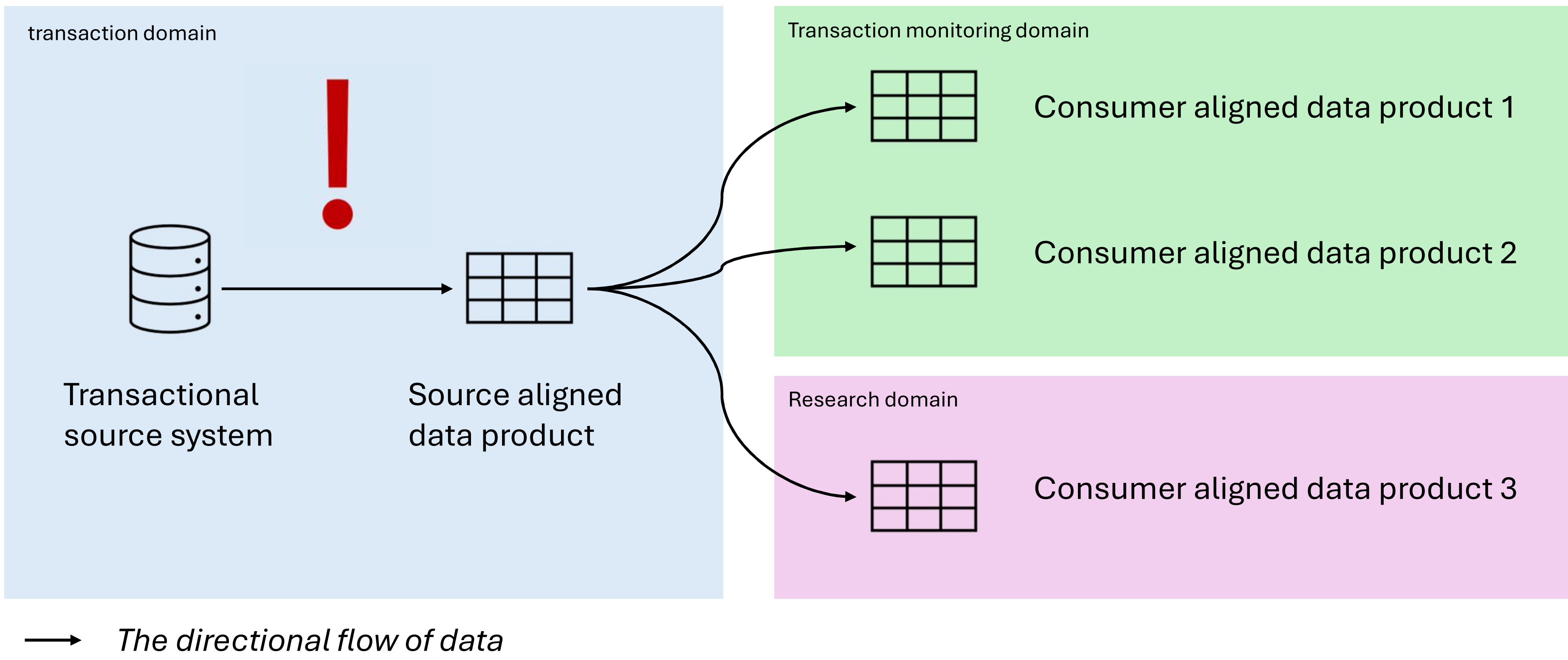
martijn.beenker@teamrockstars.nl

linkedin.com/in/martijnbeenker/

github.com/LowerKees



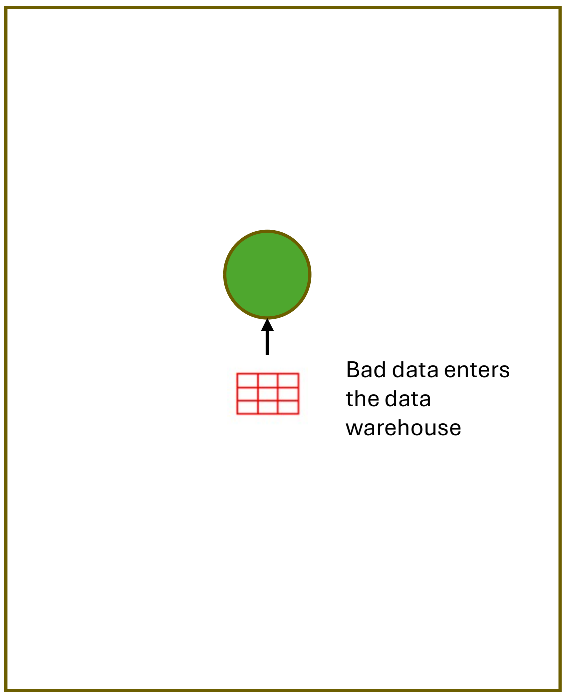
Bad data: a case study



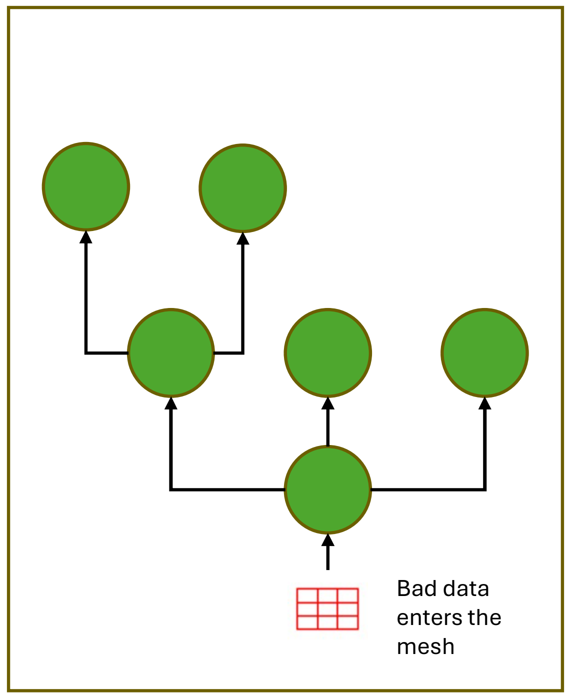
Why should you care?

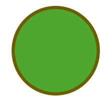
Problem 1: quality issues spread like wildfire through a data mesh

Data Warehouse



Data Mesh




Data team

Quality issue examples:
- Double records being included in the set
- Domain values that seem invalid (outdoor temperature reads of 60C+, financial transaction amount > 100 mln), public transport speeds > 150 kmph.
- Misuse of fields
- Misuse of tables
- …

Problem 2: Your tables aren't what they seem...

```python
...

df = spark.read.json("path/to/src")

...

(
    df.write.format("delta")
    .mode("overwrite")
    .option("overwriteSchema", "true")  # This changes the whole game
    .saveAsTable("sales", "path/to/table")
)
```

Quiz time!

id	amt	from	to	dts
e504e7dd-5563-40fd-9bed-1d04f4a4419b	1757452.42853700	GB70VJWD71301587833185	GB48REUN20049494565465	2021-06-03 12:12:06.797826
fdf80855-a3e8-4d1c-9f08-8b1f92581c2a	907518.39074000	GB46NJGF10225438688768	GB65CSMW47402471337538	2020-11-12 14:04:48.368358
9647257e-c2cb-4ce2-9519-fee6582ef0b5	1417463.81621066	GB35GIGV86014471414072	GB82HEAM95638897017052	2020-09-15 04:41:01.503714
45936109-530b-4c4c-aba9-df5c607b0ef9	3162210.80000000	GB96HFTW47400335289862	GB40KCJY52375325245936	2021-02-21 15:47:06.737337
95ffac03-d6ac-4ea9-a812-646a87f4cf6e	2315371.51762538	GB93QEJB68062095497929	GB50IMRB05601505677764	2024-04-13 02:55:53.38111

```python
true_schema = StructType(
    [
        StructField("id", StringType(), True),
        StructField("amt", DecimalType(16, 8), True),
        StructField("from", StringType(), True),
        StructField("to", StringType(), True),
        StructField("dts", TimestampType(), True),
    ]
)

string_schema = StructType(
    [
        StructField("id", StringType(), True),
        StructField("amt", StringType(), True),  # Changed to string
        StructField("from", StringType(), True),
        StructField("to", StringType(), True),
        StructField("dts", TimestampType(), True),
    ]
)

integer_schema = StructType(
    [
        StructField("id", StringType(), True),
        StructField("amt", DecimalType(16, 8), True),
        StructField("from", IntegerType(), True),  # changed from string to integer
        StructField("to", StringType(), True),
        StructField("dts", TimestampType(), True),
    ]
)
```

Problem 3: your code is riddled with runtime schema inspection

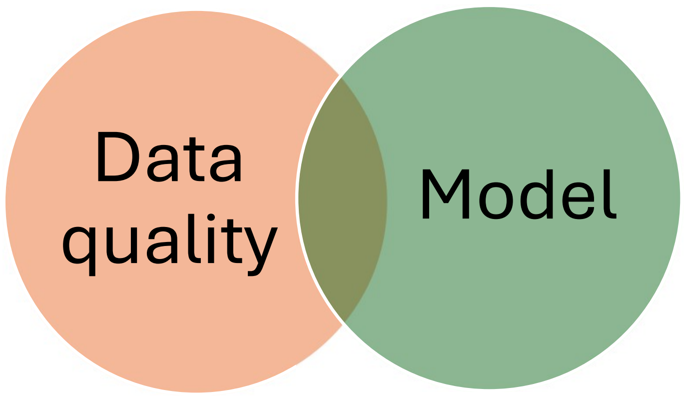
Go from this:

```
def get_year_boilerplate(df: DataFrame, col_name: str) -> DataFrame:
    """Get the year from a date or timestamp column"""
    if col_name not in df.columns:
        raise ValueError(
            "DataFrame does not contain a 'date' column“
        )

    if df.schema[col_name].dataType not in [
        DateType(),
        TimestampType()
    ]:
        raise TypeError(
            "The 'date' column is not of type date or timestamp“
        )

    return df.withColumn("year", year(col(col_name)))
```

To this:

```
def get_year(df: DataFrame, col_name: str) -> DataFrame:
    """Get the year from a date or timestamp column"""
    return df.withColumn("year", year(col(col_name)))
```

Clustering our problems



Owned by the data producing team	ownership	Owned by the data consuming team
Relatively expensive to scan	effort	Relatively cheap to scan

Significance depends on perspective of consumer

Recap

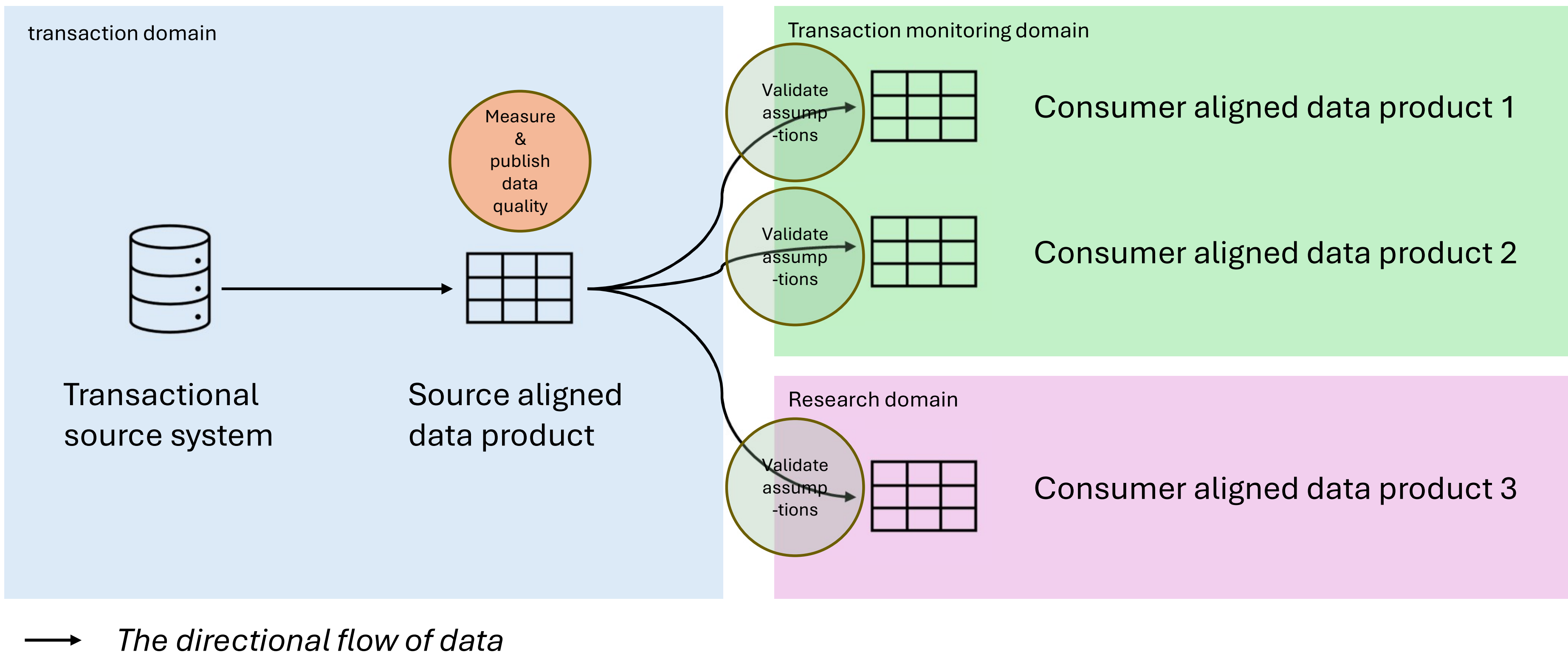
Data quality

- Owned by the data producer
- More expensive to check than schema issues

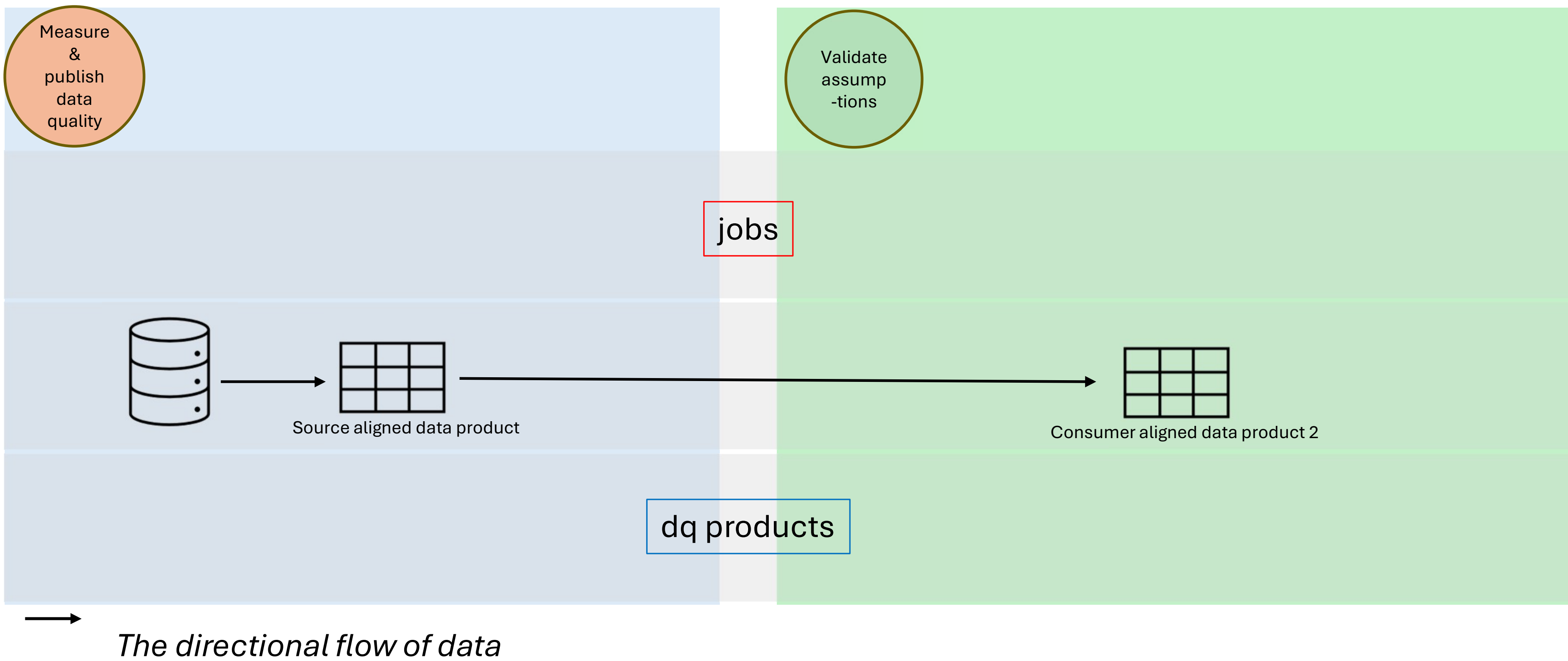
Schema

- Caused by flexibility of framework
- Owned by the data consumer
- Less expensive to check than schema issues

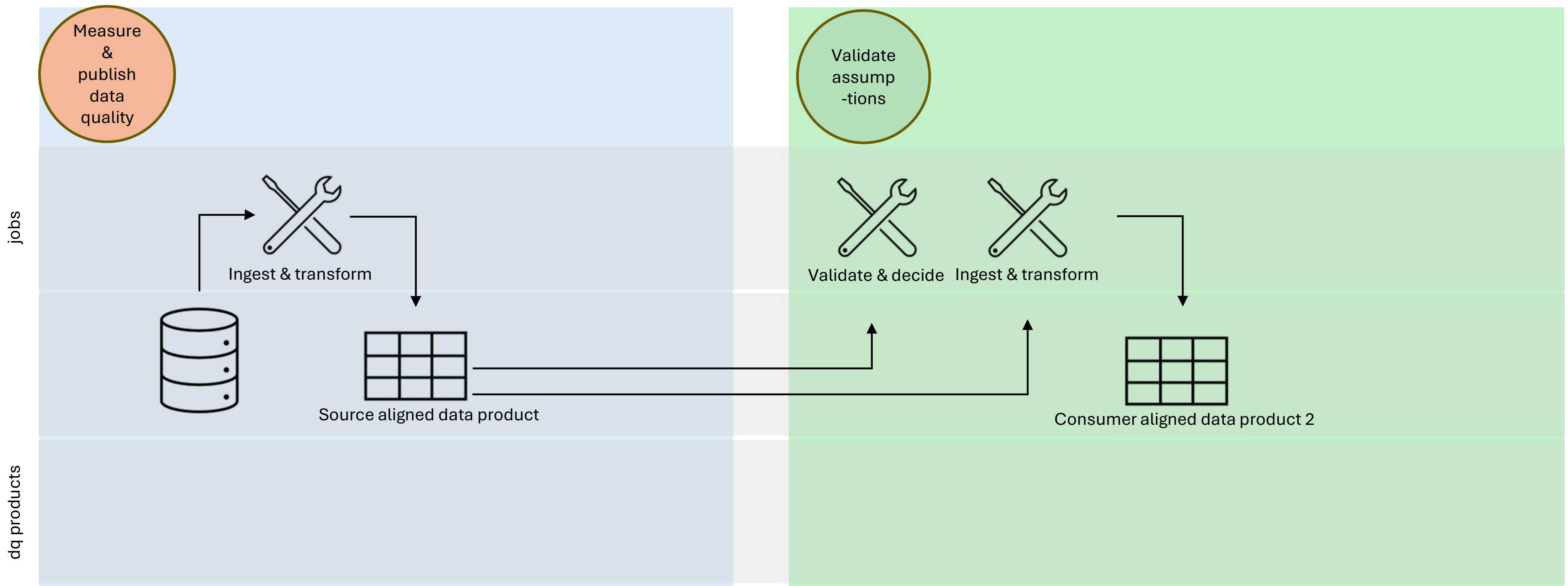
Plotting data quality and assumption validation in the case



Adding a jobs and dq products swim lane



Plotting data quality and assumption jobs and dq products in the case study (level 1)

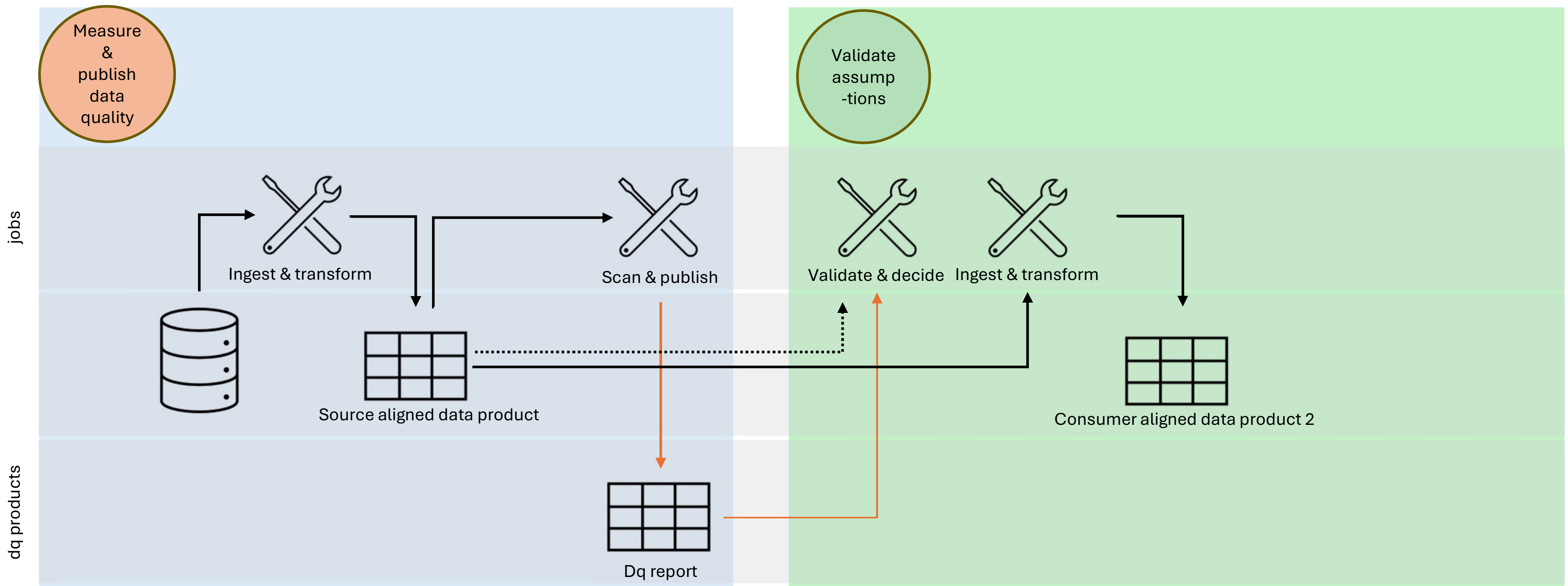


The directional flow of business data

Professionalize your poor man's approach…

- Well maintained
- Fits your current processing framework
- Allows for separation of checks and implementation
- Produces standardized outcome that can be parsed into a new data product

Plotting data quality and assumption jobs and dq products in the case study



The directional flow of business data

The directional flow of data quality data

Conclusion

Summing up producer and consumer benefits

Producer

1. Create transparency on the subject of data quality
2. Enable consumers to make data driven decisions in data usage
3. Enable data stewards to make data driven decisions
4. Perform expensive checks only once

Consumer

1. Safe-guard the integrity of your data product
   - Only read DQ checks that impact your data products
   - Add DQ checks that are uniquely yours
2. Reduce code complexity by minimizing the need for schema inspection

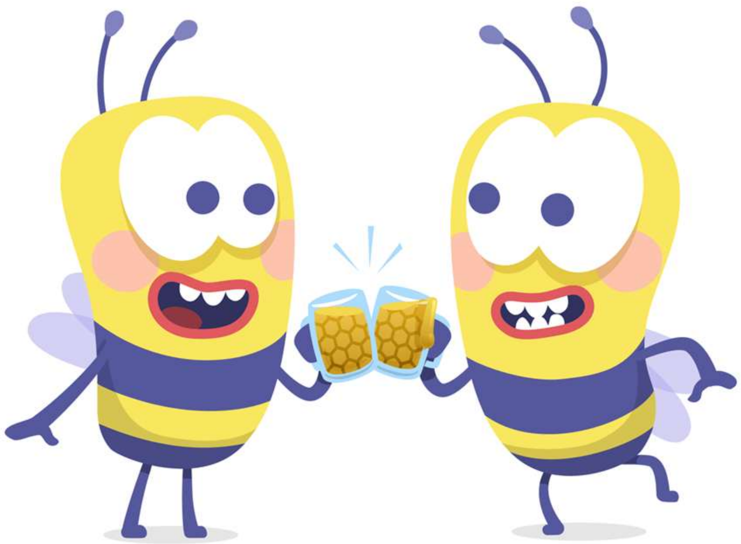
Is… is that it?

Yes: it’s late and you want to go home

But…

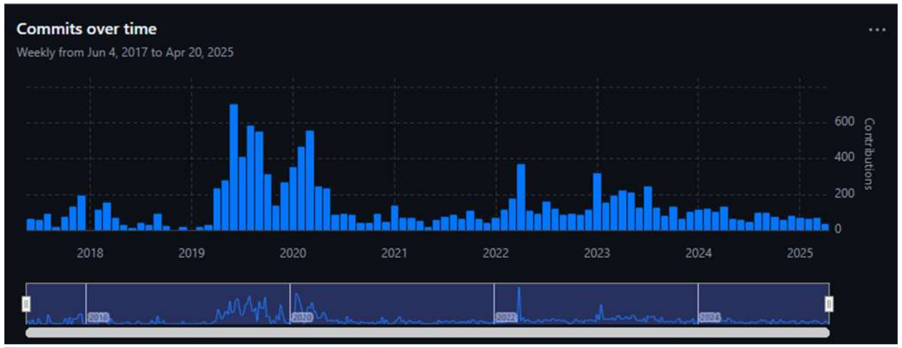
- More scripts in the repo that we didn’t discuss
- More slides in the deck that help with implementation (and navigating dependency hell) and determining the maturity level and matching approach
- More DQ issues in the repo for you to try out

Thank you very much for being here

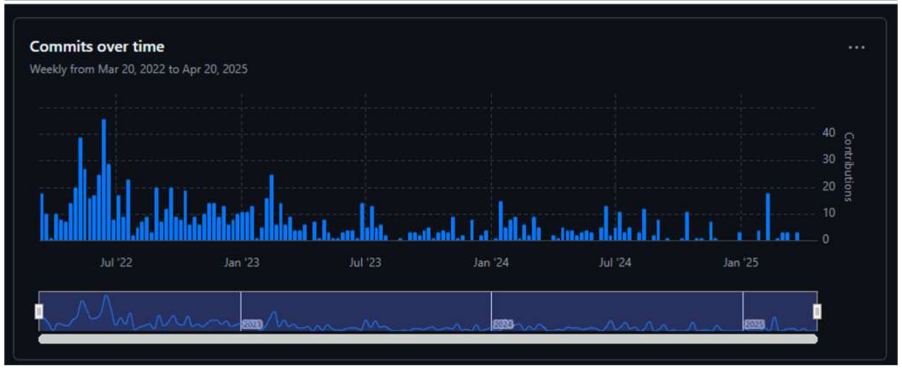


Selecting a framework that's right for you

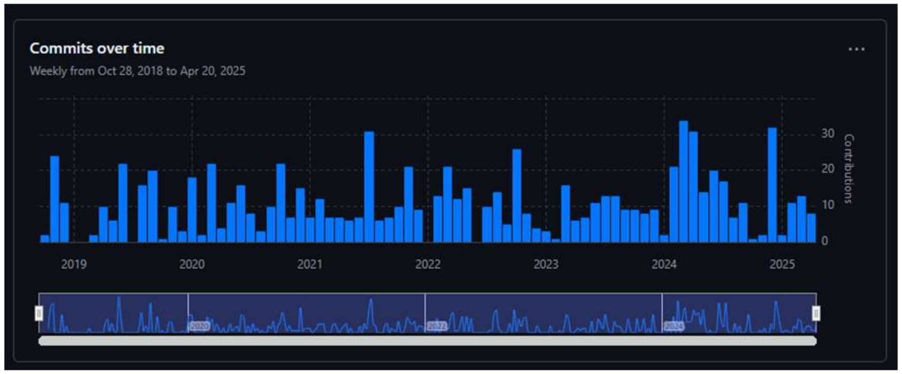
	Great expectations	Soda CL	Data contract cli	Pandera
Maintenance	Good	Sparsely	Good	Good
Fit processing framework	Supports Spark data frame	Doesn't support Spark data frames, requires a SQL interface.	Supports parquet, delta, json and csv formats	Only supports pandas data frames, so conversion is needed
Separates checks and interpretation	Yes	Yes	Yes	Yes, but validates at runtime using decorators
Standardized outcome	Json format available	Supports local output in JSON or CSV	Json format available	
Additional considerations	Heavy push to paid version	Heavy push to a paid version	Depends on availability of server type	Supports pandas, pyspark, polars, dask, modin



Great expectations

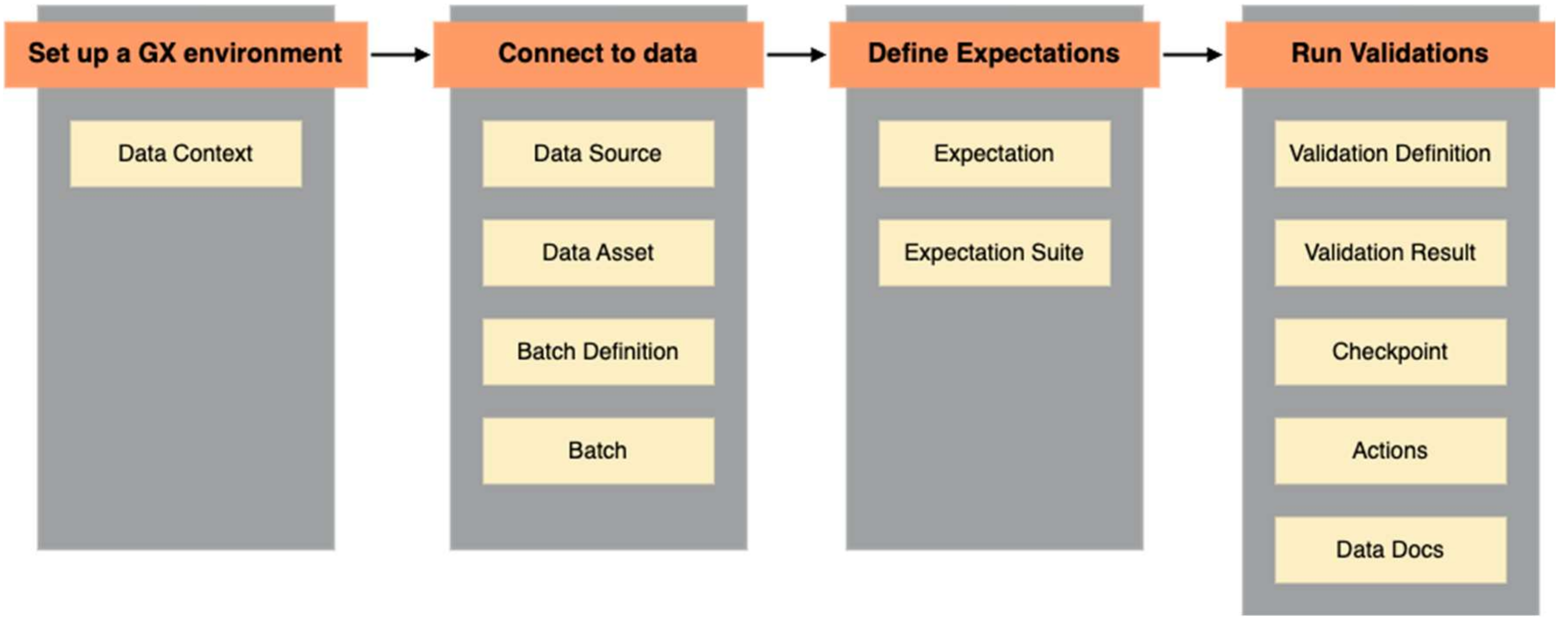


Soda core

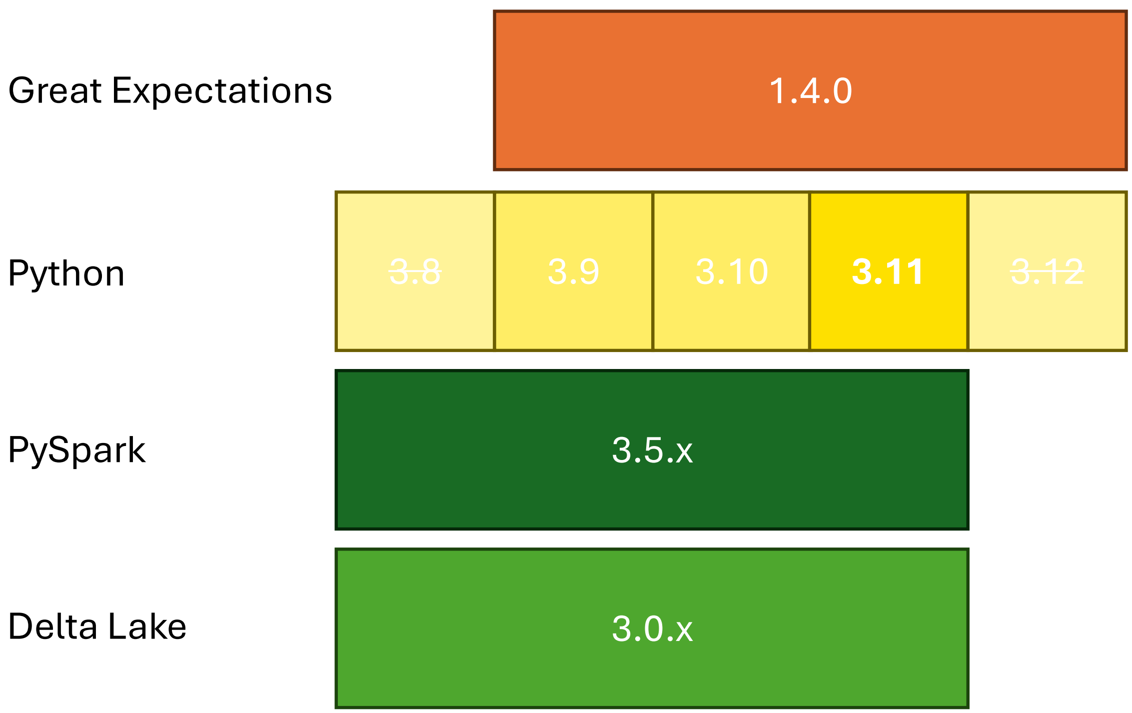


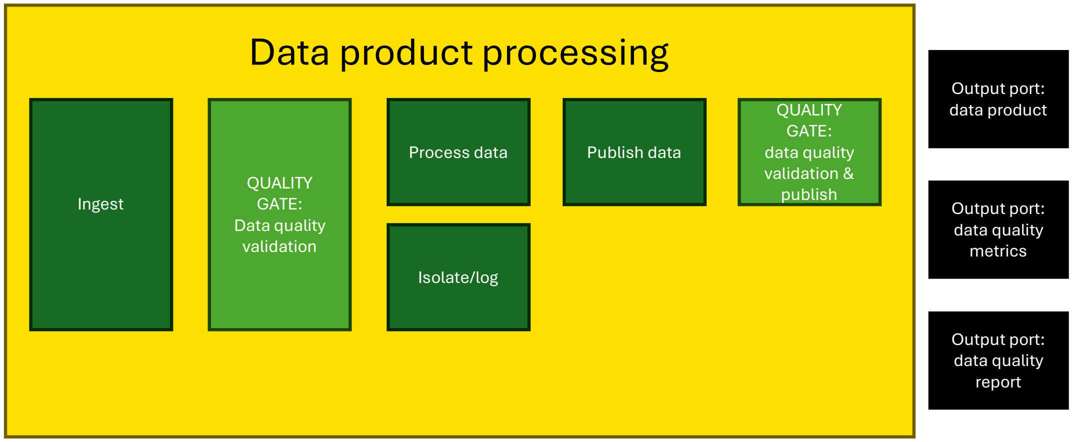
pandera

Great Expectations terminology



Great expectations, pyspark, delta, python




Data product processing
Ingest
QUALITY GATE: Data quality validation
Process data
Isolate/log
Publish data
QUALITY GATE: data quality validation & publish
Output port: data product
Output port: data quality metrics
Output port: data quality report

sources

- https://martinfowler.com/articles/data-mesh-principles.html
- https://datacontract.com
- https://greatexpectations.io/expectations/

Solidifying your quality gate approach

Maturity level of the organization

- Level 5: I’ve never seen this…
- Level 4: pipelines respond automatically to data quality findings
- Level 3: data governance is guided by data driven quality insights
- Level 2: teams write more robust pipelines based on validated schema assumptions
- Level 1: no checks and balances

Technical next step

- Automate verification of data contract compliance
- Consume dq metrics and automate data processing response
- Implement data quality metric publication and consumption
- Implement input validation for consuming teams

Solidifying your quality gate approach

Maturity level of the organization

- Level 5: I’ve never seen this…
- Level 4: pipelines respond automatically to data quality findings
- Level 3: data governance is guided by data driven quality insights
- Level 2: teams write more robust pipelines based on validated schema assumptions
- Level 1: no checks and balances

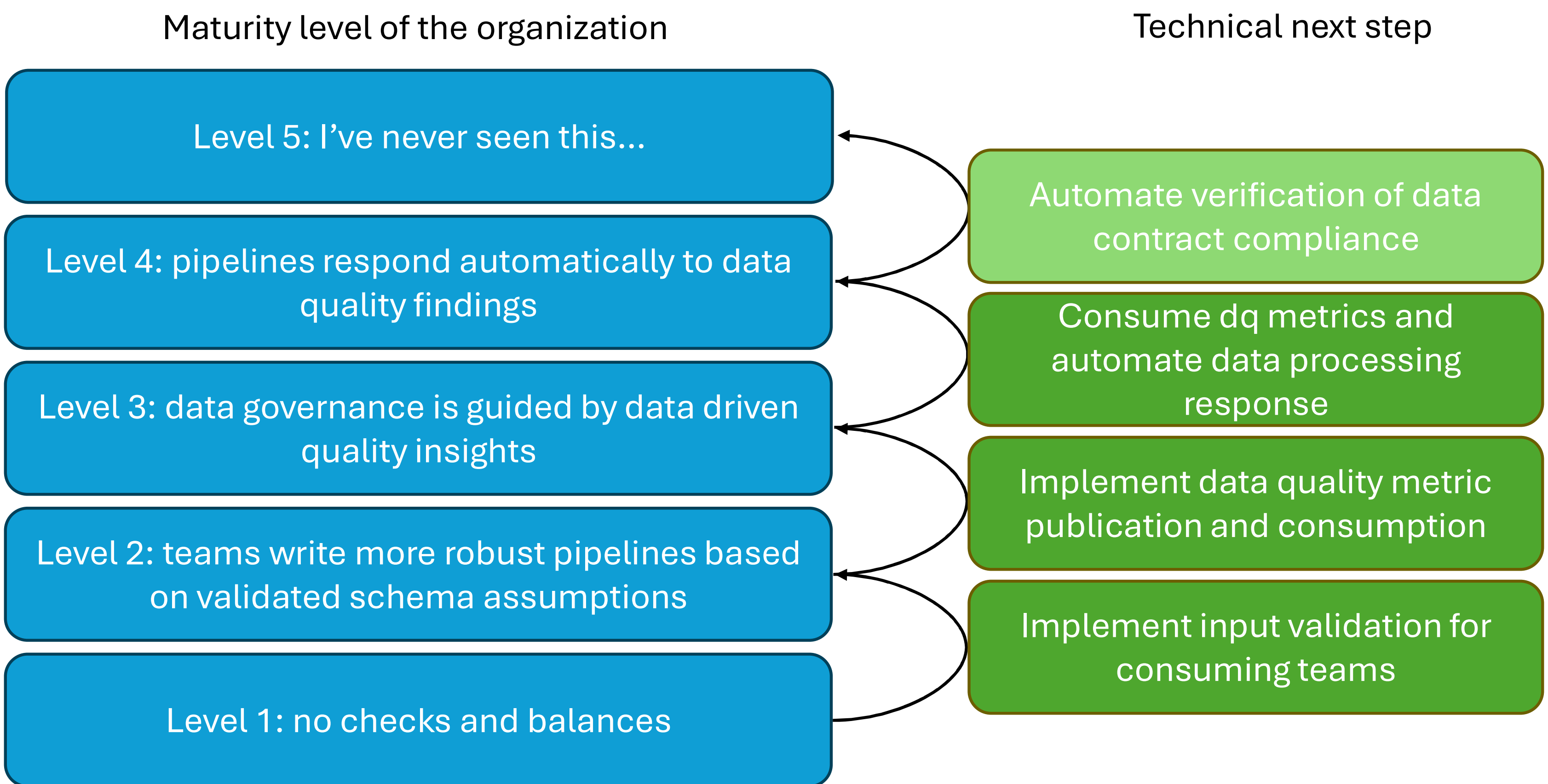
Technical next step

- Automate verification of data contract compliance
- Consume dq metrics and automate data processing response
- Implement data quality metric publication and consumption
- Implement input validation for consuming teams

Solidifying your quality gate approach

Maturity level of the organization	Technical next step
Level 5: I've never seen this…	
Level 4: pipelines respond automatically to data quality findings	Automate verification of data contract compliance
Level 3: data governance is guided by data driven quality insights	Consume dq metrics and automate data processing response
Level 2: teams write more robust pipelines based on validated schema assumptions	Implement data quality metric publication and consumption
Level 1: no checks and balances	Implement input validation for consuming teams

Solidifying your quality gate approach



A poor man's input validation

```python
def check_column_names(
    actual_schema: StructType,
    expected_schema: StructType,
    error_on_additional_cols: bool = False,
) -> None:
    expected_column_names = [field.name for field in expected_schema]
    actual_column_names = [field.name for field in actual_schema]

    not_found = []
    for expected_column_name in expected_column_names:
        if expected_column_name not in actual_column_names:
            not_found.append(expected_column_name)

    if len(not_found) > 0:
        raise ValueError(f"Expected column name(s) {not_found} not found in source")

    if error_on_additional_cols:
        check_column_names(
            actual_schema=expected_schema,
            expected_schema=actual_schema,
            error_on_additional_cols=False,
        )
```

Pros
- Quick to build
- Matches your team's need
- Low code complexity

Cons:
- Harder to learn for newcomers
- Hard to standardize output over teams