DATA+AI
SUMMIT 2022

Enabling BI in a Lakehouse Environment

How Spark and Delta Can Help With Automating a DWH Development

Ivana Pejeva
Cloud Solution Architect, Microsoft

Yoshi Coppens
Data Engineer, element61

ORGANIZED BY databricks

Agenda

- Evolution to a Lakehouse
- Why automating DWH development?
- Spark framework for automating DWH development
- DataOps for BI
- Bridge between BI and modern use cases

Evolution of data architectures

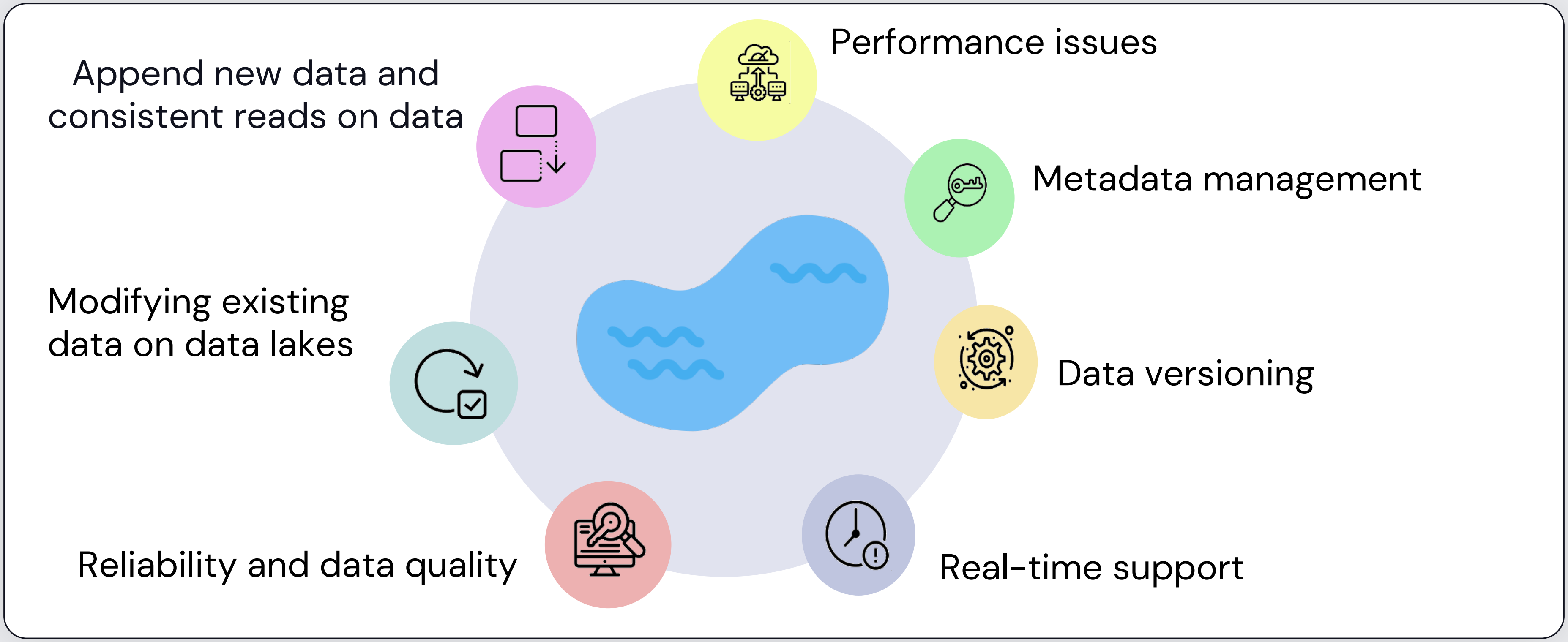
…from **data warehouses**

- Centralizes data from different sources
- Structured data
- Limitation with the increase in variety of data

to **data lakes…**

- Support for both structured and unstructured data
- Low-cost storage
- Open file formats

Challenges with Data Lakes



Append new data and consistent reads on data

Performance issues

Metadata management

Modifying existing data on data lakes

Data versioning

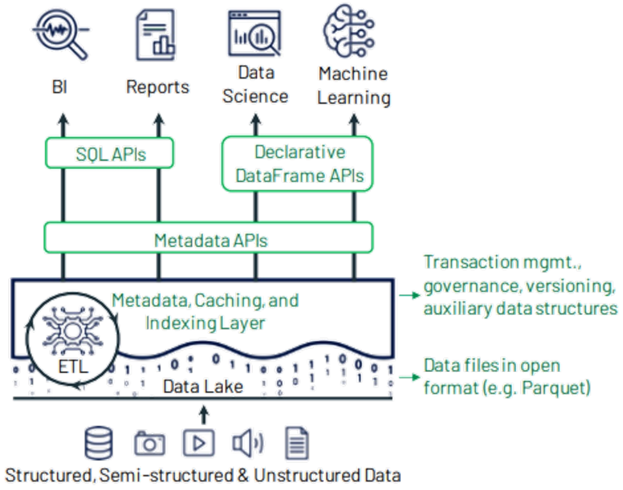
Reliability and data quality

Real-time support

DATA+AI SUMMIT 2022

Data Lakehouse design

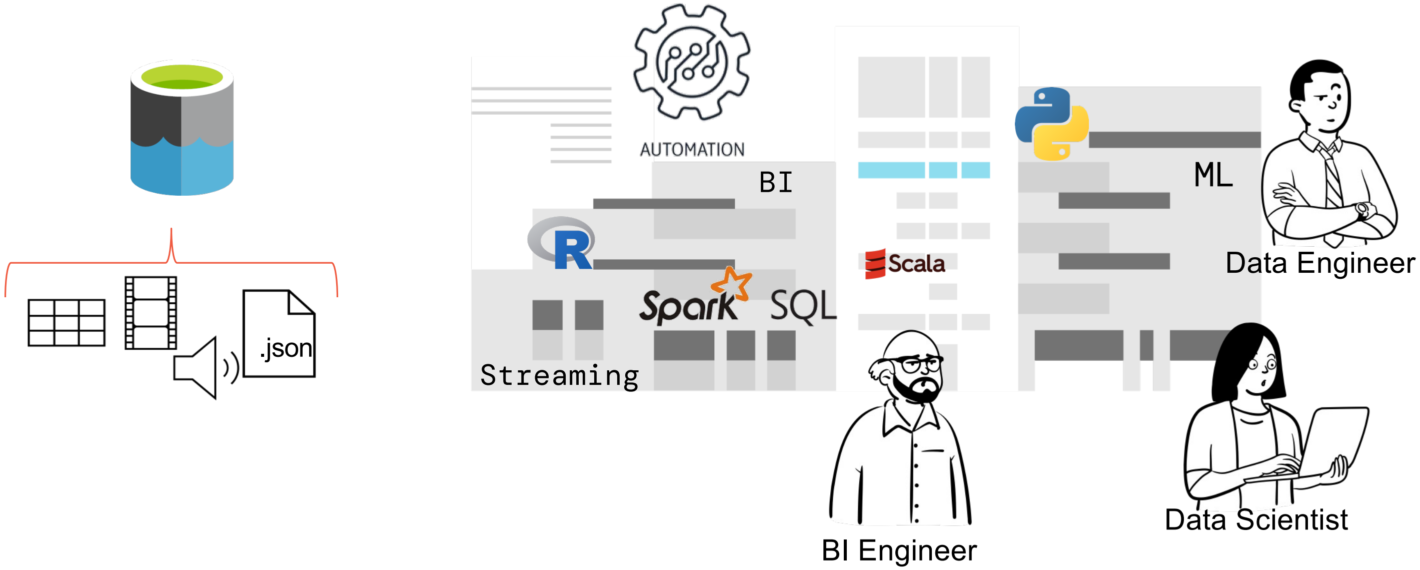
- Leverage data from data lakes
- Solving **reliability and quality** challenges in data lakes
- Optimized performance
- Support for **machine learning and BI** together
- Improved **governance** and security
- Extended file, tool and language support



BI
Reports
Data Science
Machine Learning
SQL APIs
Declarative DataFrame APIs
Metadata APIs
Metadata, Caching, and Indexing Layer
ETL
Data Lake
Transaction mgmt., governance, versioning, auxiliary data structures
Data files in open format (e.g. Parquet)
Structured, Semi-structured & Unstructured Data

Source: Databricks

Why running DWH workloads in a Lakehouse



Why we want to automate DWH development

Reading bronze tables

- **Reading data** from data lake from bronze and **creating dataframes and views with schema** in Spark

Creation of dimensions

- Create incremental integer primary keys
- Create dummy primary keys for missing records
- Auto increment keys for new records
- Write merge queries to "upsert" data to gold

Creation of facts

- Lookup of foreign keys to dimensions
  - Lookup to both SCD1 and SCD2 type of dimensions
- Write merge queries to "upsert" data to gold

Why we want to automate DWH development

Reading bronze tables

Hundreds of source tables

Creation of dimensions

- Create incremental integer primary keys
- Create dummy primary keys for missing records
- Auto increment keys for new records
- Write merge queries to “upsert” data to gold

Creation of facts

- Lookup of foreign keys to dimensions
    - Lookup to both SCD1 and SCD2 type of dimensions
- Write merge queries to “upsert” data to gold

Why we want to automate DWH development

Reading bronze tables

Hundreds of source tables

Creation of dimensions

Many dimension tables

Creation of facts

- Lookup of foreign keys to dimensions
    - Lookup to both SCD1 and SCD2 type of dimensions
- Write merge queries to “upsert” data to gold

Why we want to automate DWH development

Reading bronze tables

Hundreds of source tables

Creation of dimensions

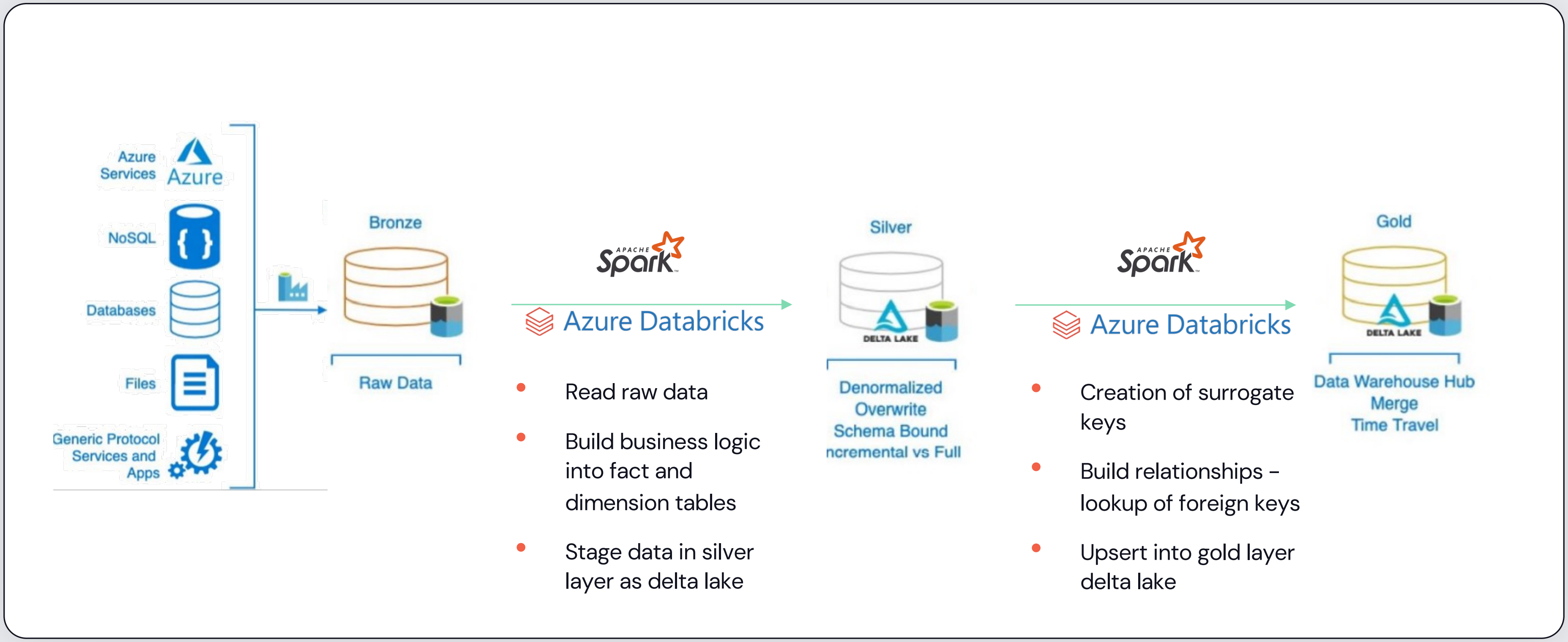
Many dimension tables

Creation of facts

Many fact tables

Data modelling in a Lakehouse

Modernize and automate DWH development



- Read raw data
- Build business logic into fact and dimension tables
- Stage data in silver layer as delta lake

- Creation of surrogate keys
- Build relationships – lookup of foreign keys
- Upsert into gold layer delta lake

How can we implement DWH principles on data lakes

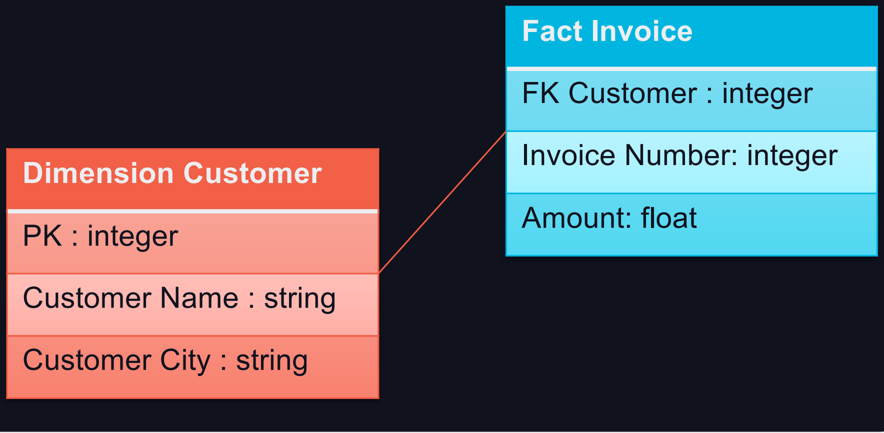
Delta Lake brings ACID transactions to Data lake

- Atomicity: every transaction is logged in transaction log
- Consistency: serializable isolation on write
- Isolation: concurrent writes
- Durability: available in case of failures

Spark framework for DWH development

Delta Lake for high data quality

- Transaction log to guarantee atomicity
- DML support – UPDATE/DELETE/MERGE
- Enforced schema and schema evolution
- Identity columns



Fact Invoice

FK Customer : integer

Invoice Number: integer

Amount: float

Dimension Customer

PK : integer

Customer Name : string

Customer City : string

Data warehouse principals in Lakehouse

SCD Type 1 and 2 Dimensions

Current customer table

PK	Customer_id	Customer_city	Is_current	Start_date	End_date	Hashed_key
1	1A	Brussels	True	20-03-2022	31-12-2999	hash1
2	2B	Antwerp	True	10-01-2021	31-12-2999	hash2

Data warehouse principals in Lakehouse

SCD Type 1 and 2 Dimensions

Current customer table

PK	Customer_id	Customer_city	Is_current	Start_date	End_date	Hashed_key
1	1A	Brussels	True	20-03-2022	31-12-2999	hash1
2	2B	Antwerp	True	10-01-2021	31-12-2999	hash2

New updates

Customer_id	Customer_city	Hashed_key
1A	Antwerp	hash3

Data warehouse principals in Lakehouse

SCD Type 1 and 2 Dimensions

Current customer table

PK	Customer_id	Customer_city	Is_current	Start_date	End_date	Hashed_key
1	1A	Brussels	True	20-03-2022	31-12-2999	hash1
2	2B	Antwerp	True	10-01-2021	31-12-2999	hash2

New updates

Customer_id	Customer_city	Hashed_key
1A	Antwerp	hash3

Updated Customer table

PK	Customer_id	Customer_city	Is_current	Start_date	End_date	Hashed_key
1	1A	Brussels	False	20-03-2022	29-06-2022	hash1
2	2B	Antwerp	True	10-01-2021	31-12-2999	hash2
3	1A	Antwerp	True	30-06-2022	31-12-2999	hash3

Data warehouse principals in Lakehouse

SCD Type 1 and 2 Dimensions

```
currentRecords = updates \
    .alias(“updates”) \
    .join(current_customer_table.alias(“current_customer_table”), “Customer_id”) \
    .where(current_customer_table.is_current=’true’ and
    current_customer_table.hashed_key <> updates.hashed_key”)


newUpdates= (
    currentRecords
    .selectExpr(“NULL as Customer_id”, “updates.*”)
    .union(updates.selectExpr(“Customer_id”, “*”))
    )
```

Data warehouse principals in Lakehouse

SCD Type 1 and 2 Dimensions

```
deltaTable
.alias(“current_customer_table")
.merge(
newUpdates.alias("updates"),
"current_customer_table.Customer_id = updates.Customer_id"
)
.whenMatchedUpdate(
set={ "current_customer_table.end_date" : current_date(), is_current: False }
).whenNotMatchedInsert(set = {all columns to updates.values, is_current to True})
.execute()
```

Data warehouse principals in Lakehouse

Surrogate keys

How we used to do it

- Find the max surrogate key in the table
- Use monotonically_increasing_id()

```
df = df.withColumn(“PK”, maxPk +
monotonically_increasing_id() )
```

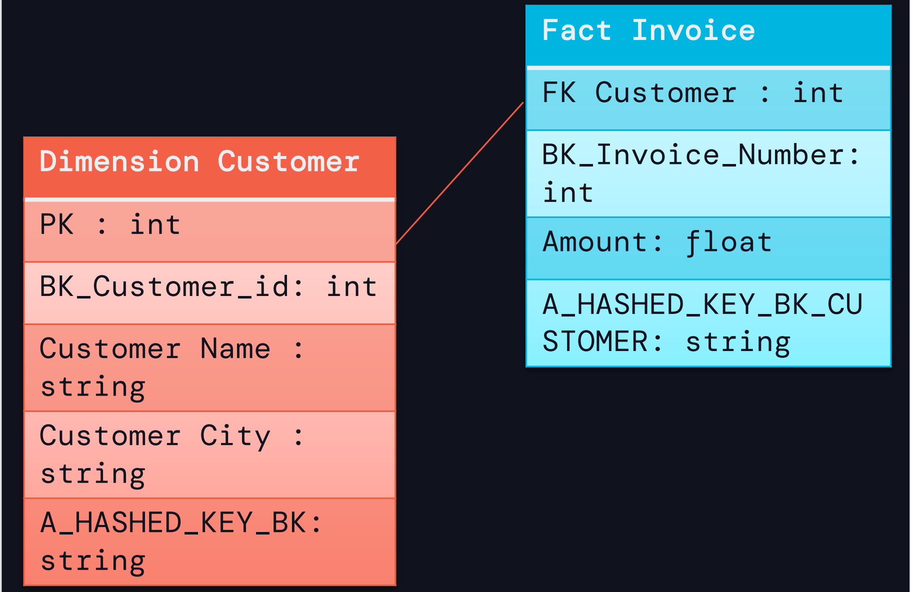
How we can do it now

- Use delta built-in functionality “IDENTITY”

```
CREATE TABLE customer
( PK int GENERATED ALWAYS AS IDENTITY
(START WITH 0 INCREMENT BY 1),
Customer_id string
)
```

Data warehouse principals in Lakehouse

Foreign keys lookup



Dimension Customer

PK : int

BK_Customer_id: int

Customer Name : string

Customer City : string

A_HASHED_KEY_BK: string

Fact Invoice

FK Customer : int

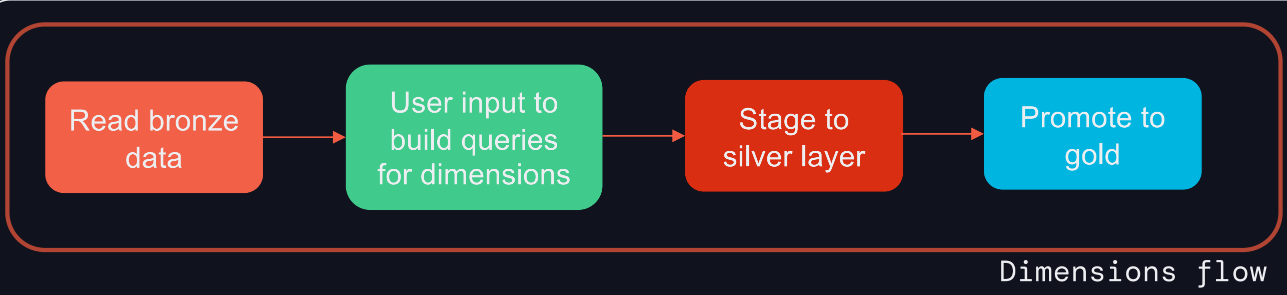
BK_Invoice_Number: int

Amount: float

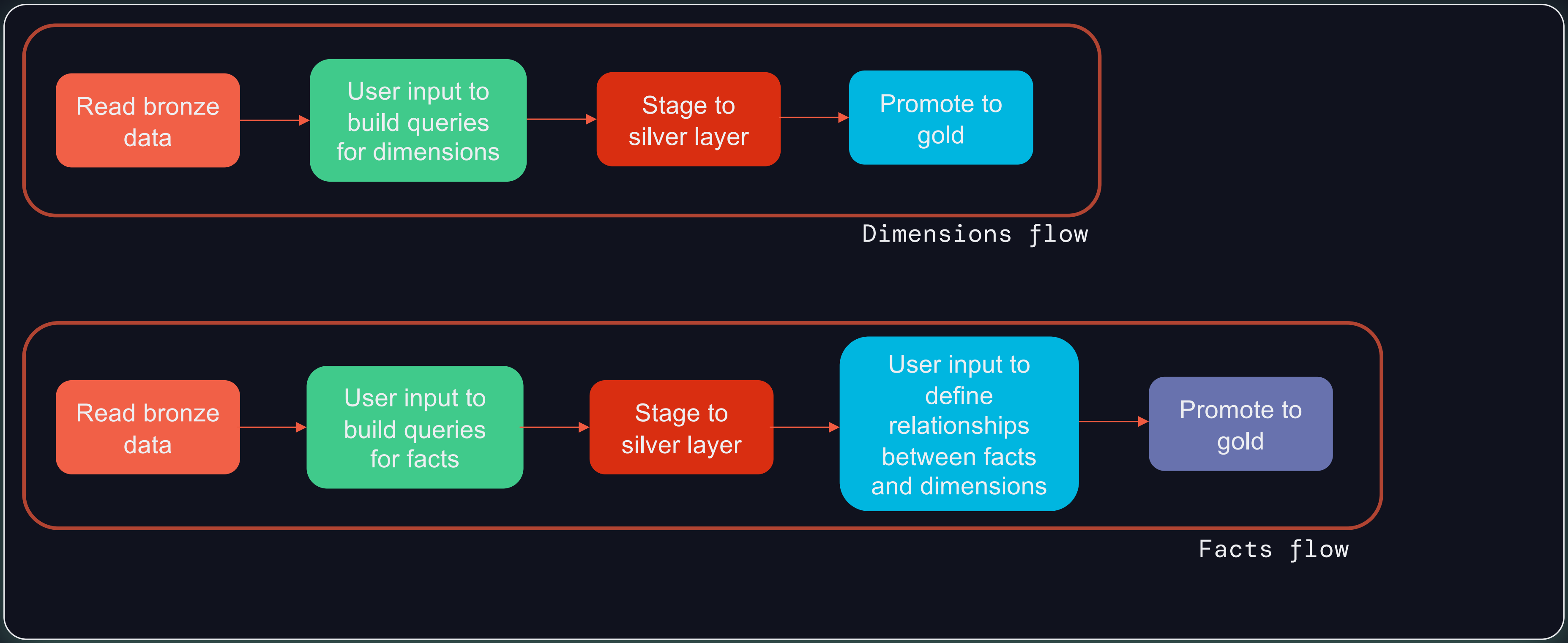
A_HASHED_KEY_BK_CUSTOMER: string

- User input is relationship between fact and dimension tables
- Automated lookup of foreign keys for dimensions (SCD1, SCD2, role playing)
- Uses range join optimization for SCD Type 2 dimensions

Automate DWH development



Automate DWH development



Read bronze data

User input to build queries for dimensions

Stage to silver layer

Promote to gold

Dimensions flow

Read bronze data

User input to build queries for facts

Stage to silver layer

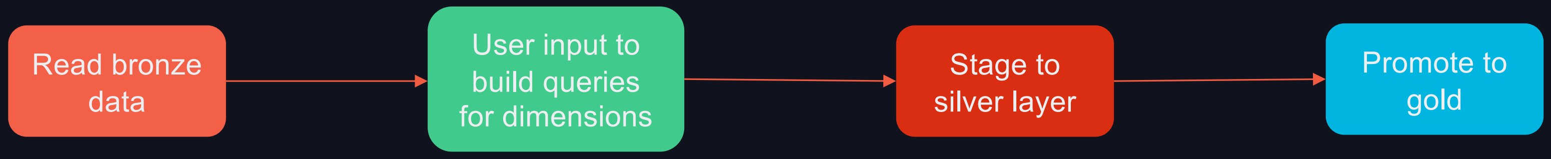
User input to define relationships between facts and dimensions

Promote to gold

Facts flow

Automate DWH development

Dimensions flow



- Dynamically read data from bronze

table	incremental
customer	No
invoices	Yes

- Create queries to define dimensions using business logic

e.g.

```
SELECT
Id as BK_customer_id,
Name as customer_name,
City as customer_city
FROM customer
```

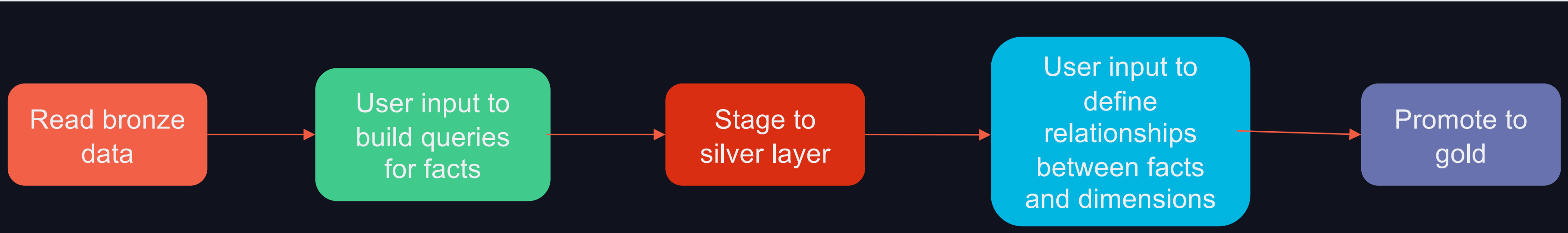
Dynamically

- Create hashed key
- Enforce schema
- Write to silver
- Check if some DWH principals are applied
- Data quality checks

- Create surrogate keys
- Merge delta table to gold

Automate DWH development

Facts flow



Read bronze data
User input to build queries for facts
Stage to silver layer
User input to define relationships between facts and dimensions
Promote to gold

- Dynamically read data from bronze

table	incremental
customer	No
invoices	Yes

- Create queries to define facts using business logic

e.g.

```
SELECT
Invoice_number as
BK_invoice_number,
Amount
FROM invoices
```

Dynamically

- Create hashed key
- Enforce schema
- Write to silver
- Check if some DWH principals are applied
- Data quality checks

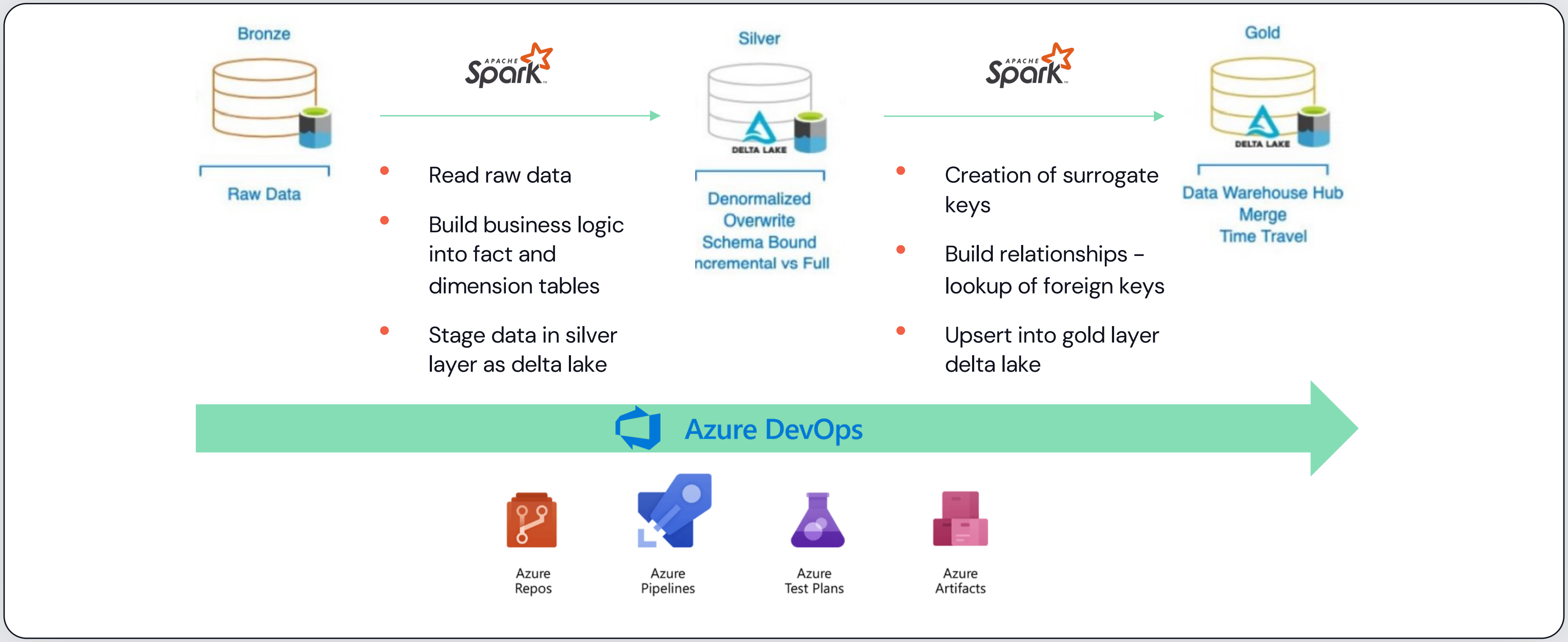
- Define relations between facts and dimensions

e.g.

```
factsLink =
[
{"F_INVOICES":
   ["D_CUSTOMERS"]
}
]
```

- Lookup foreign keys
- Merge delta table to gold

DataOps for enabling BI in a Lakehouse



Bronze

Raw Data

- Read raw data
- Build business logic into fact and dimension tables
- Stage data in silver layer as delta lake

Silver

Denormalized
Overwrite
Schema Bound
ncremental vs Full

- Creation of surrogate keys
- Build relationships – lookup of foreign keys
- Upsert into gold layer delta lake

Gold

Data Warehouse Hub
Merge
Time Travel

Azure DevOps

Azure Repos

Azure Pipelines

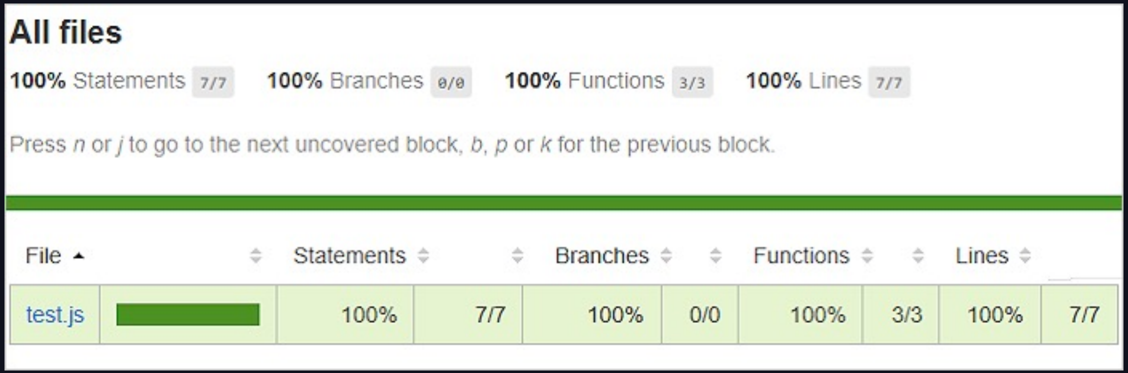
Azure Test Plans

Azure Artifacts

DataOps for enabling BI in a Lakehouse

Testing framework

Unit tests
Integration tests



All files

100% Statements 7/7 **100%** Branches 0/0 **100%** Functions 3/3 **100%** Lines 7/7

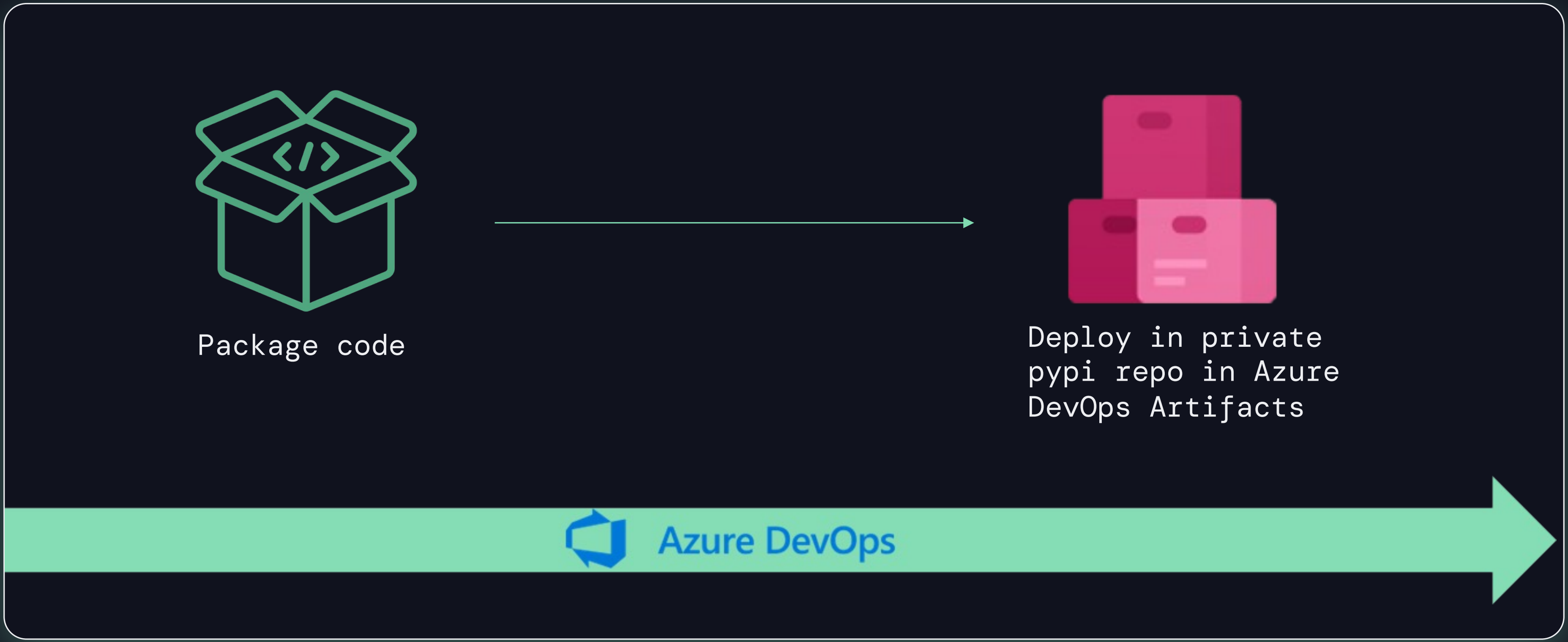
Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File ▲		Statements		Branches		Functions		Lines	
test.js	🟩	100%	7/7	100%	0/0	100%	3/3	100%	7/7

Azure DevOps

DataOps for enabling BI in a Lakehouse

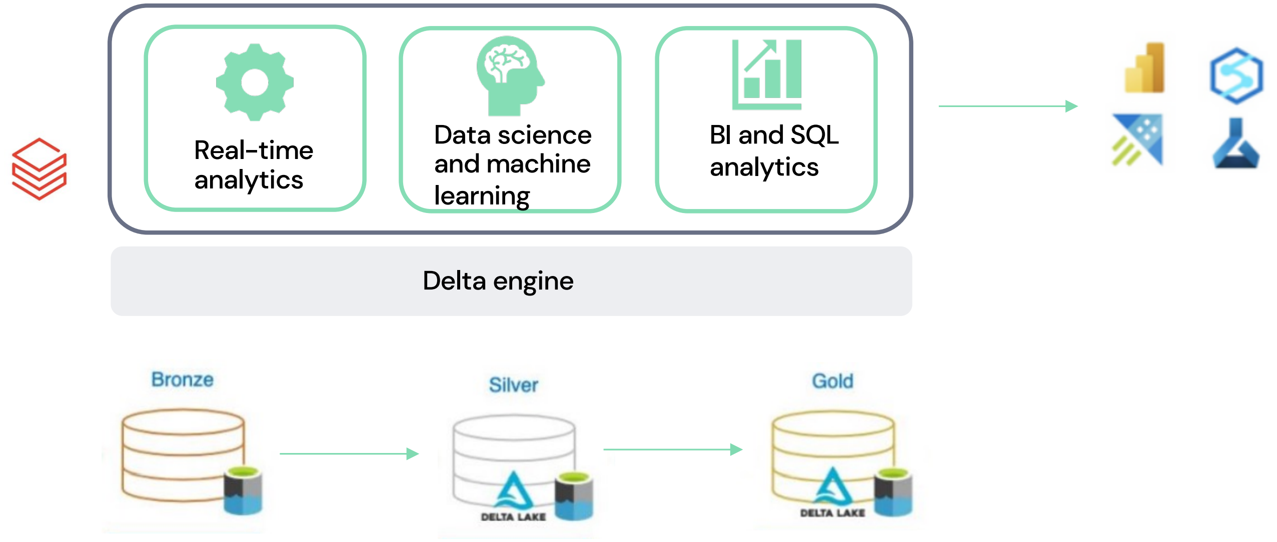
Package framework



Package code

Deploy in private pypi repo in Azure DevOps Artifacts

Bridge between BI and modern-day use cases



Real-time analytics

Data science and machine learning

BI and SQL analytics

Delta engine

Bronze

Silver

Gold

Conclusion

- Lakehouse **solves some of the shortcomings of data lakes and data warehouses**
- Data warehousing development can be **easily modernized and automated** in a Lakehouse
- **One architecture** to cover the needs of data scientists, data engineers, BI engineers

DATA+AI SUMMIT 2022

Thank you

Ivana Pejeva & Yoshi Coppens