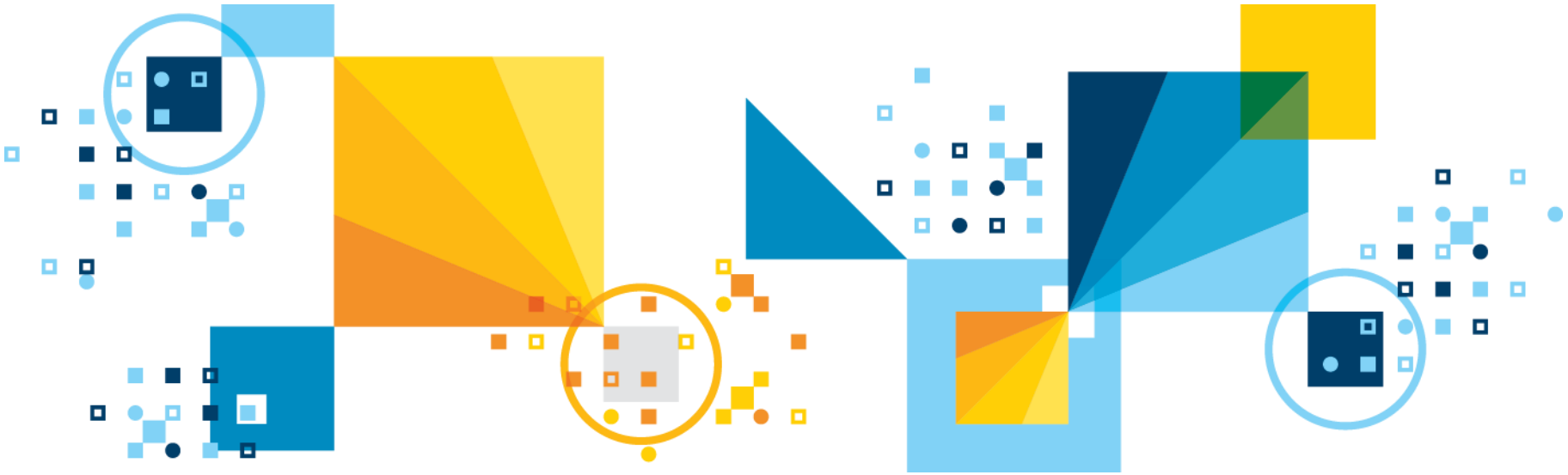
Integrating Apache Spark with an Enterprise Data Warehouse

Dr. Michael Wurst, IBM Corporation
Architect – Spark/R/Python Database Integration, In-Database Analytics

Dr. Toni Bollinger, IBM Corporation
Senior Software Developer IBM BigSQL



Agenda

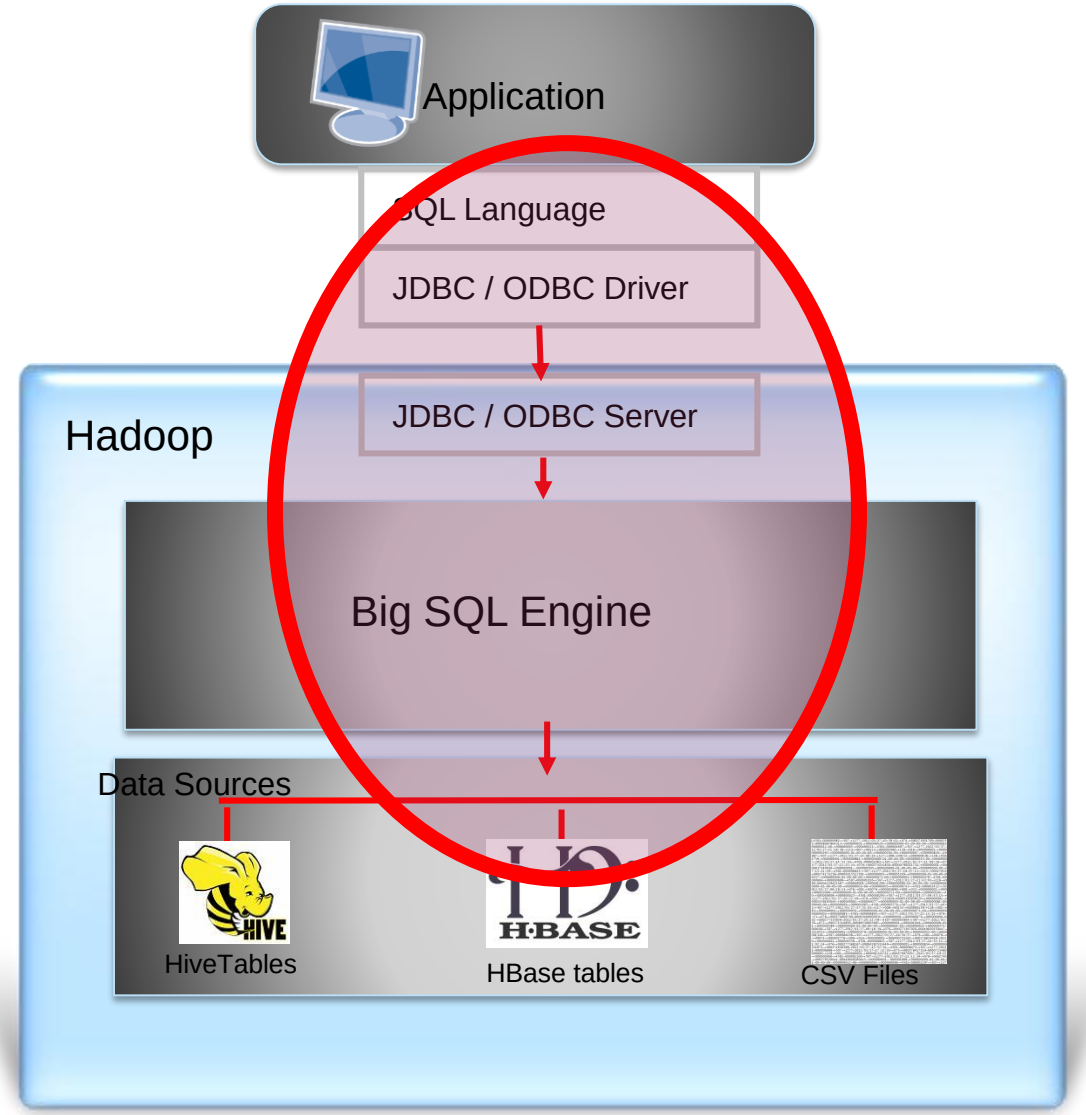
- Enterprise Data Warehouses and Open Source Analytics
- IBM BigSQL
- The impact of modern Data Warehouse technology
- Accelerating Apache Spark through SQL push-down
- Lessons Learned

Enterprise Data Warehouses and Open Source Analytics

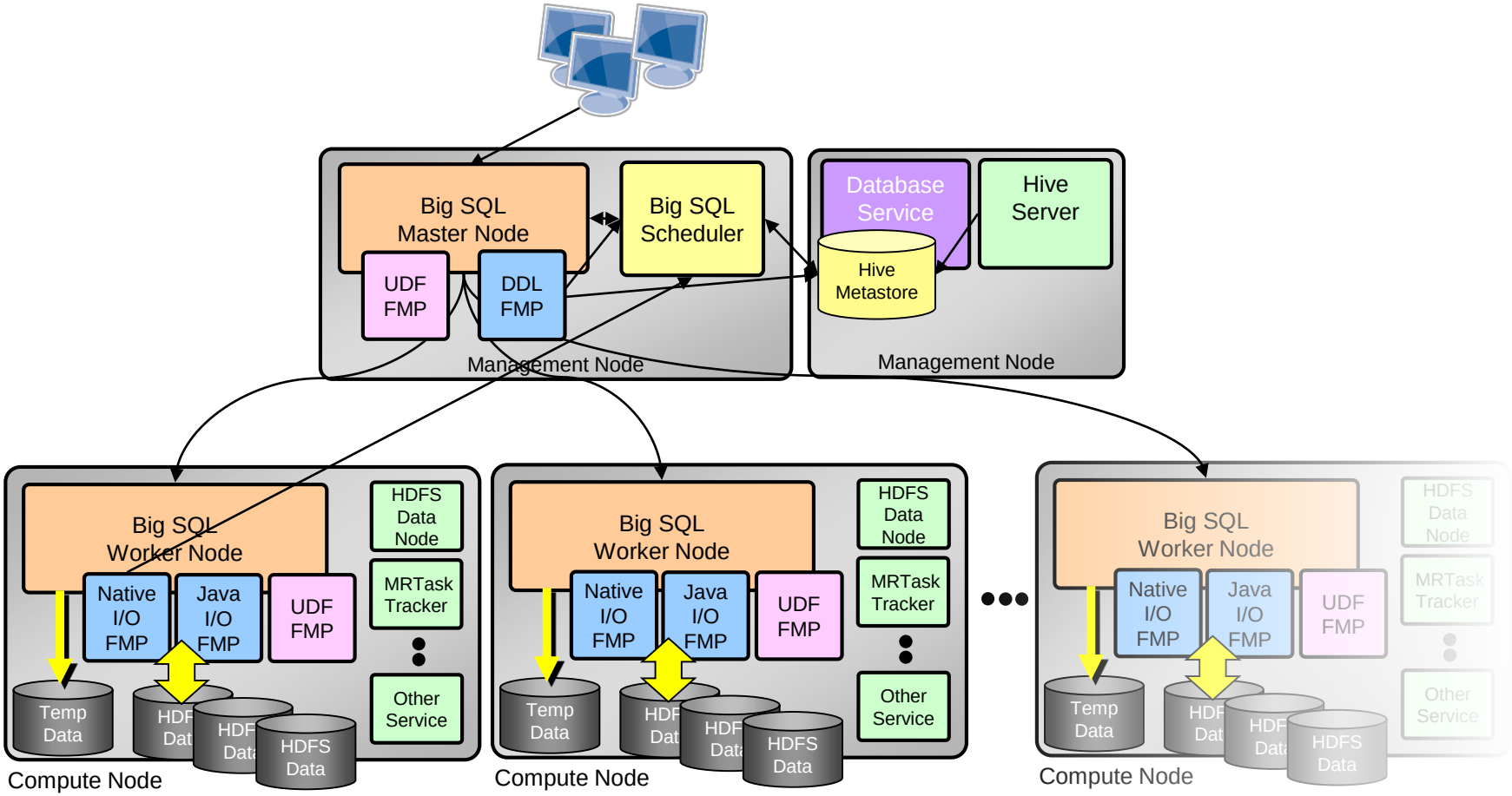
- Data Warehouses provide critical features like security, performance, backup/recovery etc. that make it the technology of choice for most enterprise uses.
- However, modern workloads like advanced analytics or machine learning require a more flexible and agile environment, like Apache Hadoop or Apache Spark.
- Combining both allows to get the best of both worlds – the enterprise-strength and performance of modern Data Warehouses and the flexibility and power of open source analytics.

BigSQL

- Most existing applications in the enterprise use SQL
- SQL bridges the chasm between existing apps and Big Data
- SQL access to all data stored in Hadoop
- Via JDBC/ODBC
- Using rich standard SQL
- Intelligently leverge database parallelism and query optimization



BigSQL – Architecture



*FMP = Fenced mode process

BigSQL Advantages

- Extends the realm of database / data warehousing techology to the Hadoop world
  - Broader functionality and better performance compared to native Hadoop SQL implementations
  - Existing skills (like SQL) can be reused
  - Existing applications (like for reporting) can be reused as well
- Allows the seamless exchange of data between Hadoop based applications and Data Warehouse applications through HDFS.
- This can be used to easily share data between Apache Spark and the BigSQL Data Warehouse.

The Impact of Modern Data Warehouse Technology

- Enterprise Data Warehouses technology evolved significantly in the recent years.
- Some of the technologies involved include:
  - Columnar storage
  - Improved Compression
  - Query execution on compressed data
  - In-Memory storage
  - SIMD optimization
- The IBM DB2 BLU Acceleration engine deployed in DB2 10 LUW and IBM dashDB is an example of such technology.

BLU Acceleration used in IBM DB2 and IBM dashDB

Dynamic In-Memory

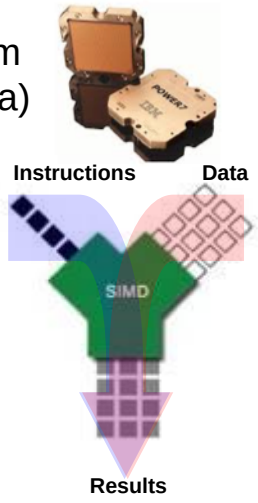
In-memory columnar processing with dynamic movement of data from storage

Actionable Compression

Patented compression technique that preserves order so data can be used without decompressing

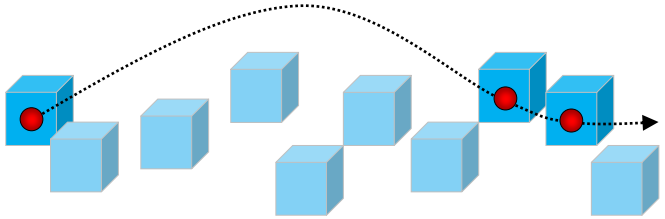
Parallel Vector Processing

Multi-core and SIMD parallelism (Single Instruction Multiple Data)



Data Skipping

Skips unnecessary processing of irrelevant data

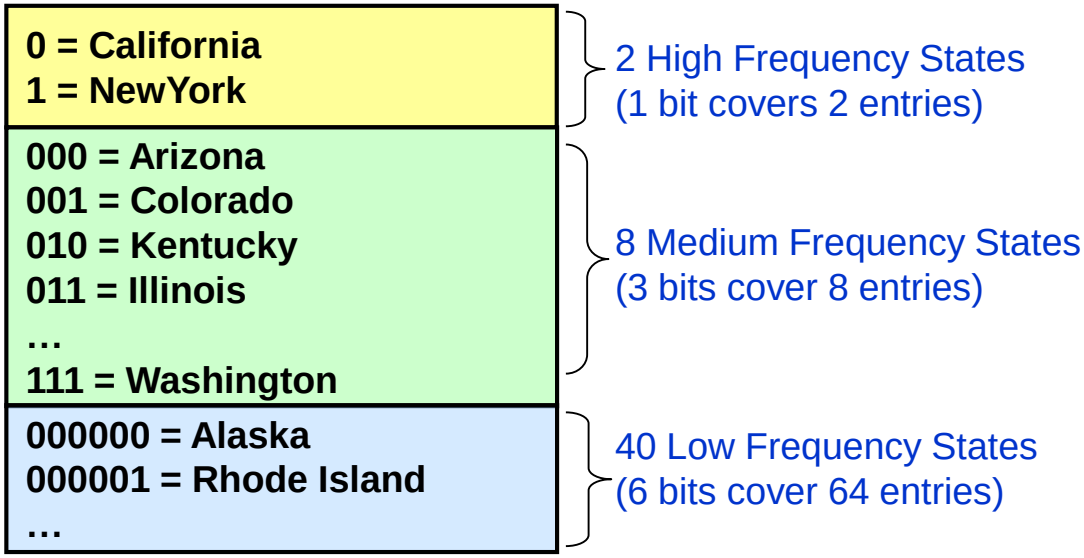


BLU Columnar Storage

- **Minimal I/O**
  - Only perform I/O on the columns and values that match query
  - As queries progresses through a pipeline the working set of pages is reduced
- **Work performed directly on columns**
  - Predicates, joins, scans, etc. all work on individual columns
  - Rows are not materialized until absolutely necessary to build result set
- **Improved memory density**
  - Columnar data kept compressed in memory
- **Extreme compression**
  - Packing more data values into very small amount of memory or disk
- **Cache efficiency**
  - Data packed into cache friendly structures

BLU uses Multiple Compression Techniques

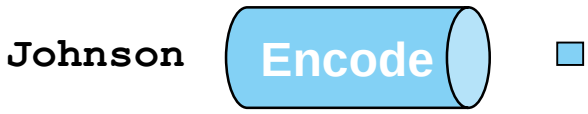
- Approximate Huffman-Encoding ("frequency-based compression"), prefix compression, and offset compression
- Frequency-based compression: Most common values use fewest bits



- Exploiting skew in data distribution improves compression ratio
- Very effective since all values in a column have the same data type
- Maps entire values to dictionary codes

Data Remains Compressed During Evaluation

- Encoded values do not need to be decompressed during evaluation
  - Predicates (=, <, >, >=, <=, Between, etc), joins, aggregations and more work directly on encoded values

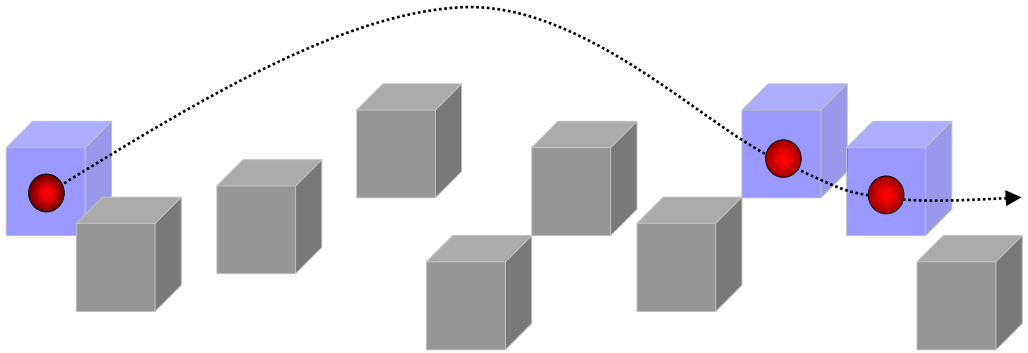


LAST_NAME	Encoding
Brown	
Johnson	
Johnson	
Johnson	
Johnson	
Brown	
Johnson	
Gilligan	
Wong	
Johnson	

```
SELECT COUNT(*) FROM T1 WHERE LAST_NAME = 'Johnson'
```

BLU Data skipping

- Automatic detection of large sections of data that do not qualify for a query and can be ignored
- Order of magnitude savings in all of I/O, RAM, and CPU
- No DBA action to define or use – truly invisible
  - Persistent storage of min and max values for sections of data values



BLU Synopsis Table

- Meta-data that describes which *ranges* of values exist in which parts of the user table

SYN13033016521627515 2_SALES_COL

TSNMIN	TSNMAX	S_DATEMIN	S_DATEMAX	...
0	1023	2005-03-01	2006-10-17	...
1024	2047	2006-08-25	2007-09-15	...
...				

TSN = Tuple Sequence Number

- Enables DB2 to skip portions of a table when scanning data to answer a query
- Benefits from data clustering, loading pre-sorted data

User table: **SALES_COL**

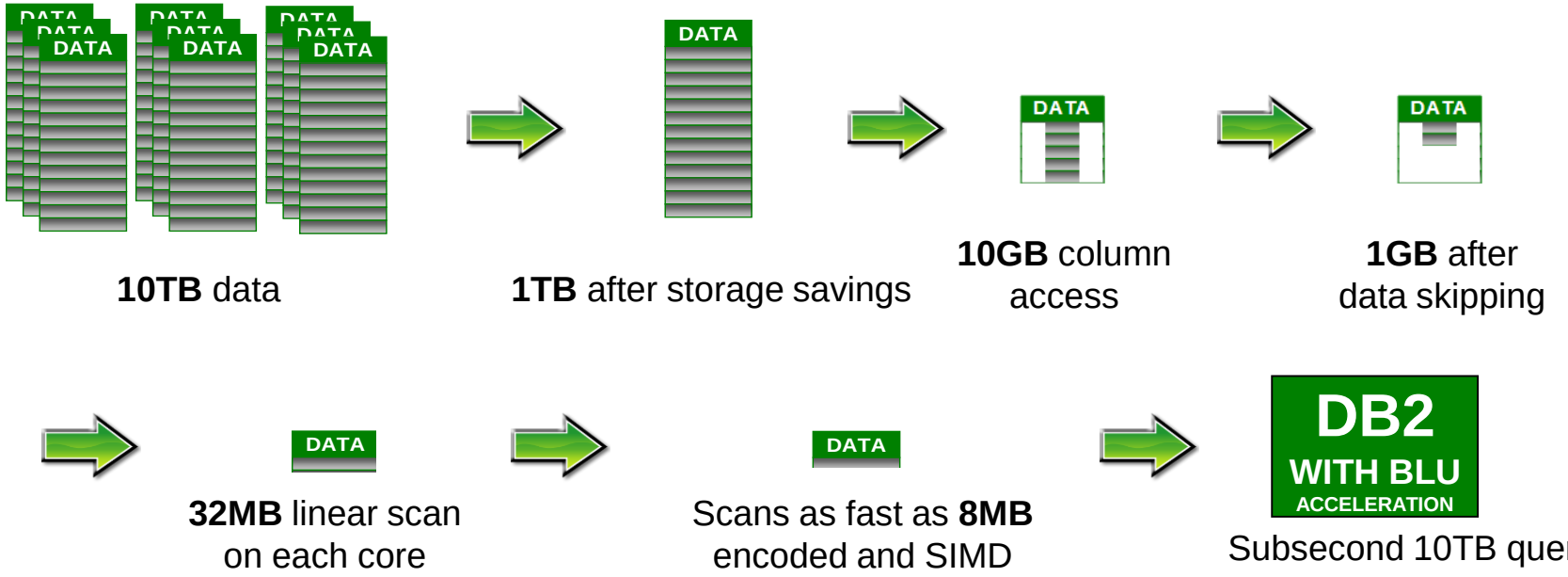
S_DATE	QTY	...
2005-03-01	176	...
2005-03-02	85	...
2005-03-02	267	
2005-03-04	231	
...		
...		
...		

0
1023
1024
2047

How DB2 with BLU Acceleration Helps

~Sub second 10TB query

- The system – 32 cores, 10TB table with 100 columns, 10 years of data
- The query: `SELECT COUNT(*) from MYTABLE where YEAR = '2010'`
- The result: sub second 10TB query! Each CPU core examines the equivalent of just 8MB of data



10TB data

1TB after storage savings

10GB column access

1GB after data skipping

32MB linear scan on each core

Scans as fast as **8MB** encoded and SIMD

Subsecond 10TB query

How to easily Benefit from modern Data Warehouse Technology from Apache Spark?

- Use the Warehouse as pure data source and pull all or selected data into Spark RDDs
  Spark benefits from fast data access, but none of the DB indexing structures is used fully and data is replicated in Spark requiring additional memory.

- Push down processing from Spark into the underlying Data Warehouse
  Full exploitation of the features of the underlying data base engine.

SQL Pushdown of (Complex) Operations from Spark

- Assumption: Source data is already stored in the Data Warehouse or is loaded into the Data Warehouse to speed up processing.
- Instead of creating Spark RDDs of the input data and applying (statistical) processing to these RDDs, we can do some of the necessary processing directly in-database.
- The Spark DataFrame already supports some operations that could directly benefit from database engine indexing structures, e.g. group by, selection, projection, join, etc.

However, we can even go a step further and speed up more complex statistical operations as, e.g. regression models.

SQL Pushdown of (Complex) Operations from Spark to BLU

- Idea: Push data intensive parts of algorithms to the database engine and do the remaining processing in Spark.
- Showcase: Linear Regression with few predictors
- Can be split into two steps:
  1. Calculating the covariance matrix
  2. Calculating the actual linear model
- The first step can be pushed to the data warehouse engine by issuing a simple select
  SELECT SUM(V1*V2), … FROM INPUT
- The second step is not data intensive and easy to compute
- Performance (based on IBM dashDB enterprise plan)

  100.000.000 rows
  10 numeric predictors
  Calculating a linear model in-database takes <3sec which is less than even loading the data into an Spark RDD

Towards Data Warehouse Aware Implementations of Spark Functions

- Idea: Create implementations of data intensive Spark algorithms that are aware of and can use the underlying Data Warehouse engine to push-down part of the processing.
- Some examples of algorithms that could profit heavily from this kind of processing:
  - Naïve Bayes
  - k-means
  - Tree models
  - Calculation of statistics
  - …
- All these could benefit from fast aggregation, selection, etc. offered by Data Warehouses without replicating the data into Spark or creating additional index structures.

Lessons Learned

- Data Warehouse technology made some significant advances in the recent years.
- Combining the enterprise-strength and performance of modern Data Warehouses with the flexibility and power of Spark is a perfect match.
- To make full use of the underlying Data Warehouse, Spark processing should be pushed into the Data Warehouse engine wherever possible.
- This also avoids additional memory consumption, as data does not need to be cached in Spark to achieve high performance.
- Using SQL push-down offers a simple and generic way to achieve this.