Global
Fabric
Community

Fabric Spark Notebooks: Monitoring and Performance Tuning

TORONTO
FABRIC DAY 2024

Toronto Fabric User Group

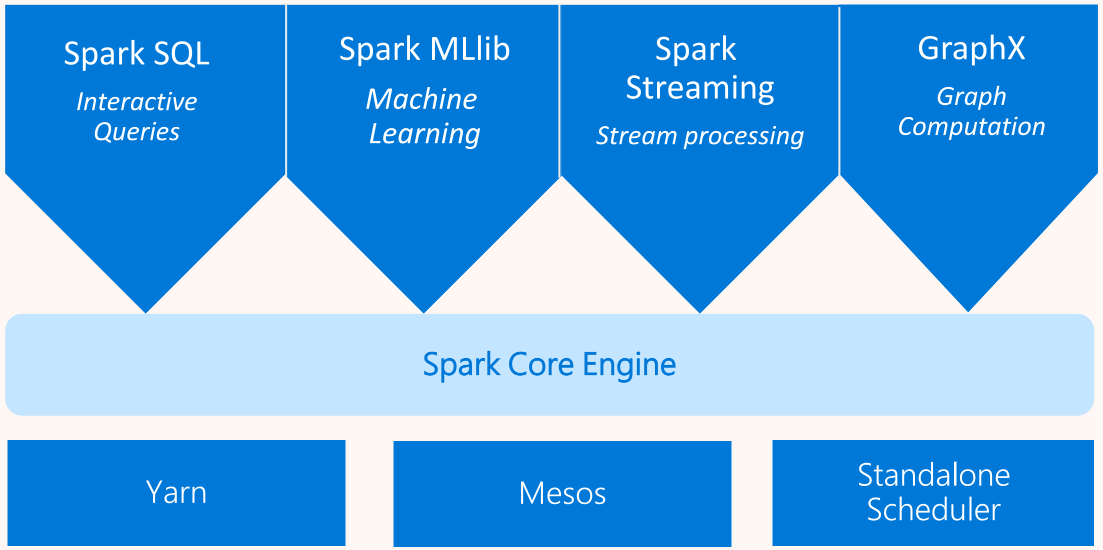
Agenda

- Spark Basics
- Best Practices
- Monitoring (Demo)
- Profiling: Sparklens (Demo)
- Fabric Native Execution Engine

Anuyogam Venkataraman
Senior Program Manager
Microsoft

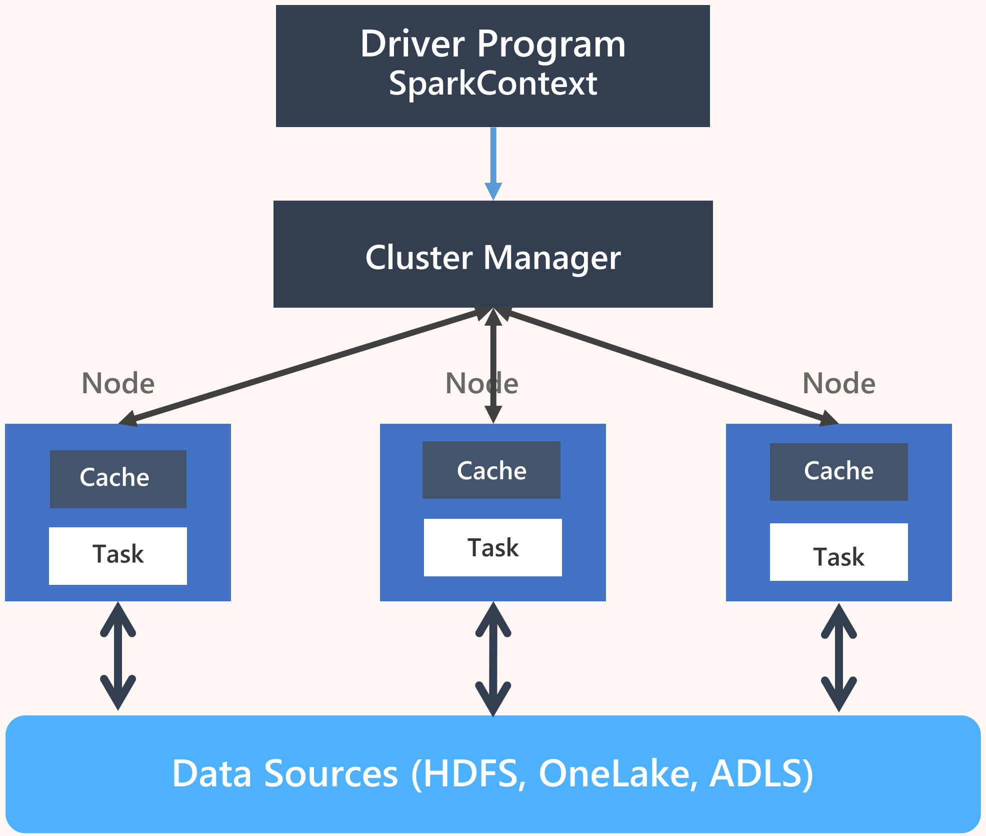
Linkedin:
https://www.linkedin.com/in/anuyogam-venkataraman-9908b022/

Apache Spark

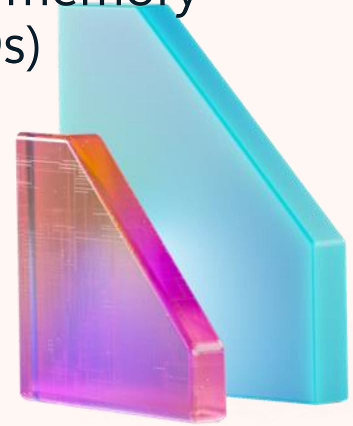


- An open-source unified analytics engine for large-scale data processing
- Supports batch, interactive and streaming data processing
- Massive in-memory distributed and parallel processing capabilities
- Allows writing code in Python, Scala, Java, R and SQL
- Commonly used for complex analytics, data transformation, machine learning and AI tasks on big data

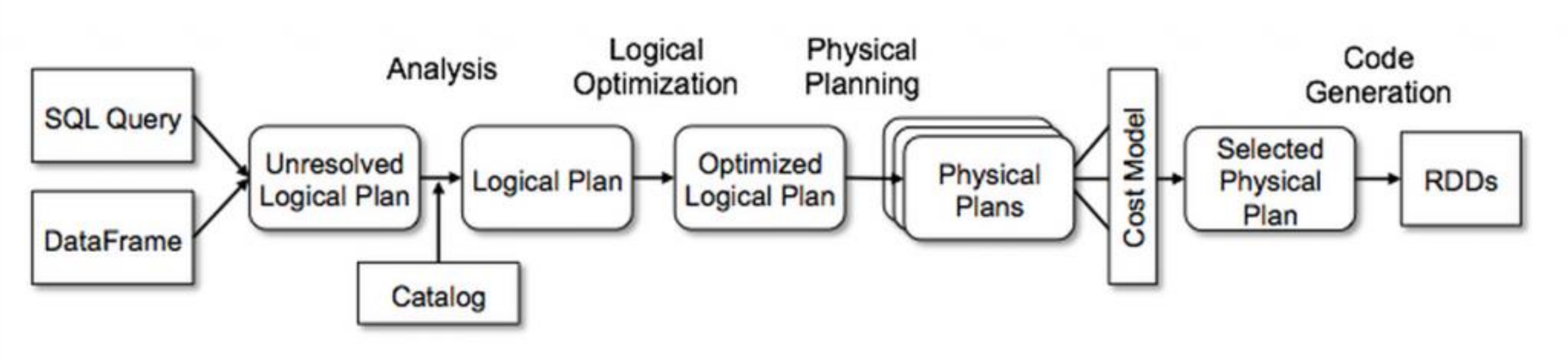
Spark design and job execution



- *Driver* runs the user's *main* function and executes the various parallel operations on the worker nodes.
- The worker nodes read and write data from/to data source.
- Spreads the processing and data onto different Worker modes
- The results of the operations are collected by the driver
- Worker nodes process data in memory using Resilient Data Sets (RDDs)



Catalyst Optimizer

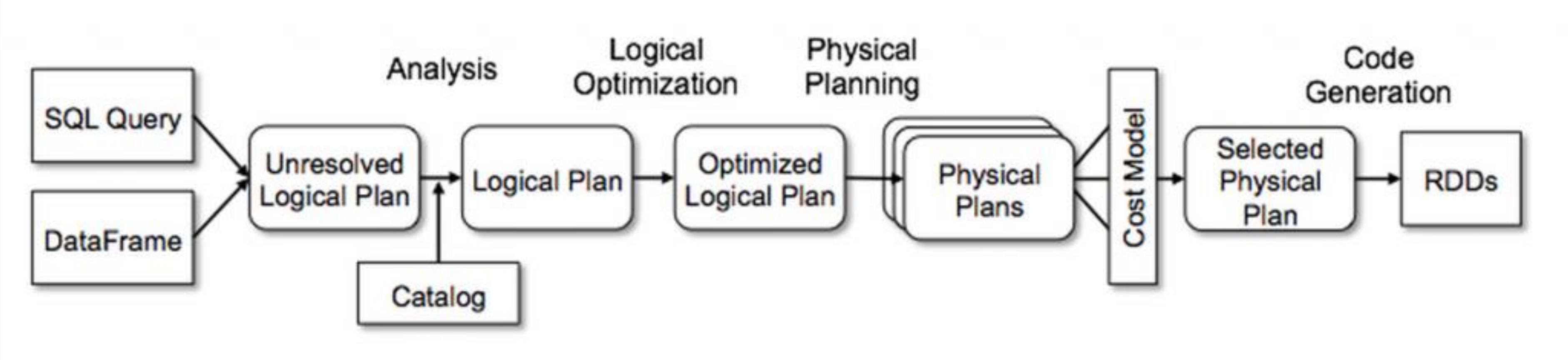



```
grouped_rdd = rdd.reduceByKey(add)
filtered_rdd = grouped_rdd.filter(lambda x: x[1] > 150)
results = filtered_rdd.collect()
```

```
filtered_df = df.groupBy("Customer") \
.agg(F.sum("Amount").alias("TotalSales")) \
.filter(F.col("TotalSales") > 150)
```

5

Catalyst Optimizer




```
filtered_df = df.groupBy("Customer") \
.agg(F.sum("Amount").alias("TotalSales")) \
.filter(F.col("TotalSales") > 150)
```

Before Catalyst Optimizer:

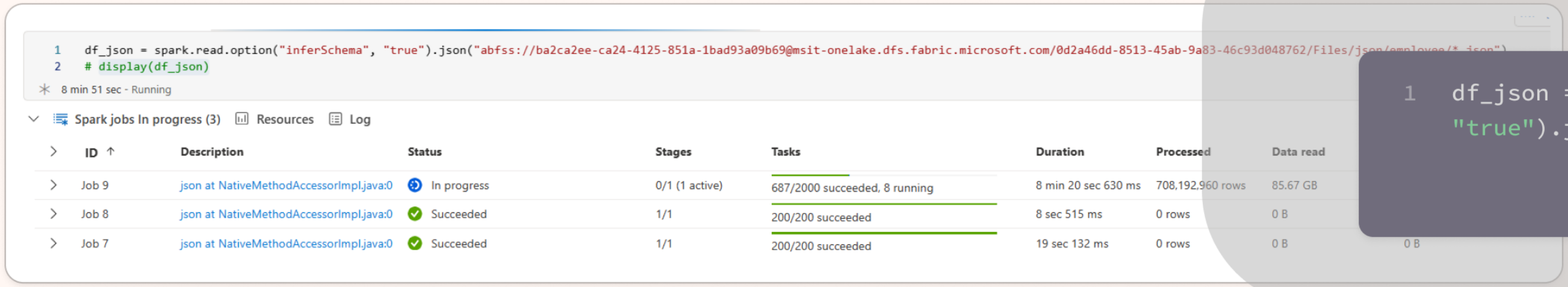
[RDD] --> [Aggregation] --> [Filter] --> [Result]

After Catalyst Optimizer (Predicate Pushdown, Projection Pruning):

[DataFrame] --> [Catalyst Optimizer] --> [Filter] --> [Project] --> [Aggregate] --> [Result]

Spark Execution Model

Actions → Trigger Jobs → Stages → Tasks



Block 0

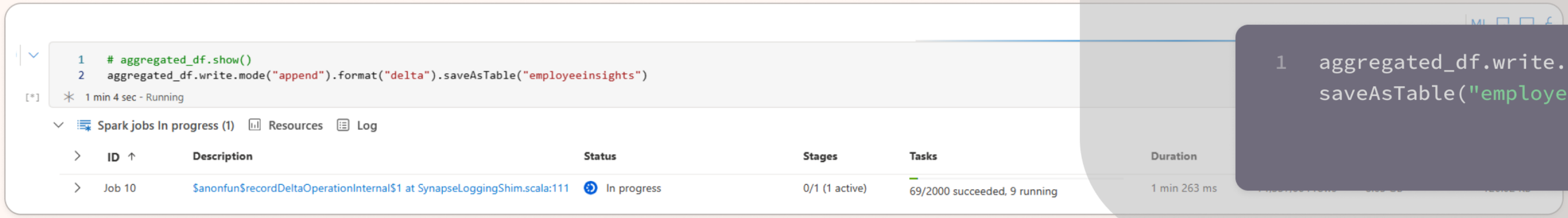
```
df_json = spark.read.option("inferSchema",
"true").json("file_path")
```

Action
Triggers Jobs


```
aggregated_df = df_json.groupBy("country",
"state").agg(sum("salary").alias("total_salary"),
avg("salary").alias("avg_salary"),
sum("salary").alias("total_salary"))
```

Transformation
(Wide Dependency)

Block 1




```
aggregated_df.write.mode("append").format("delta").
saveAsTable("employeeinsights")
```

Action
Triggers Jobs

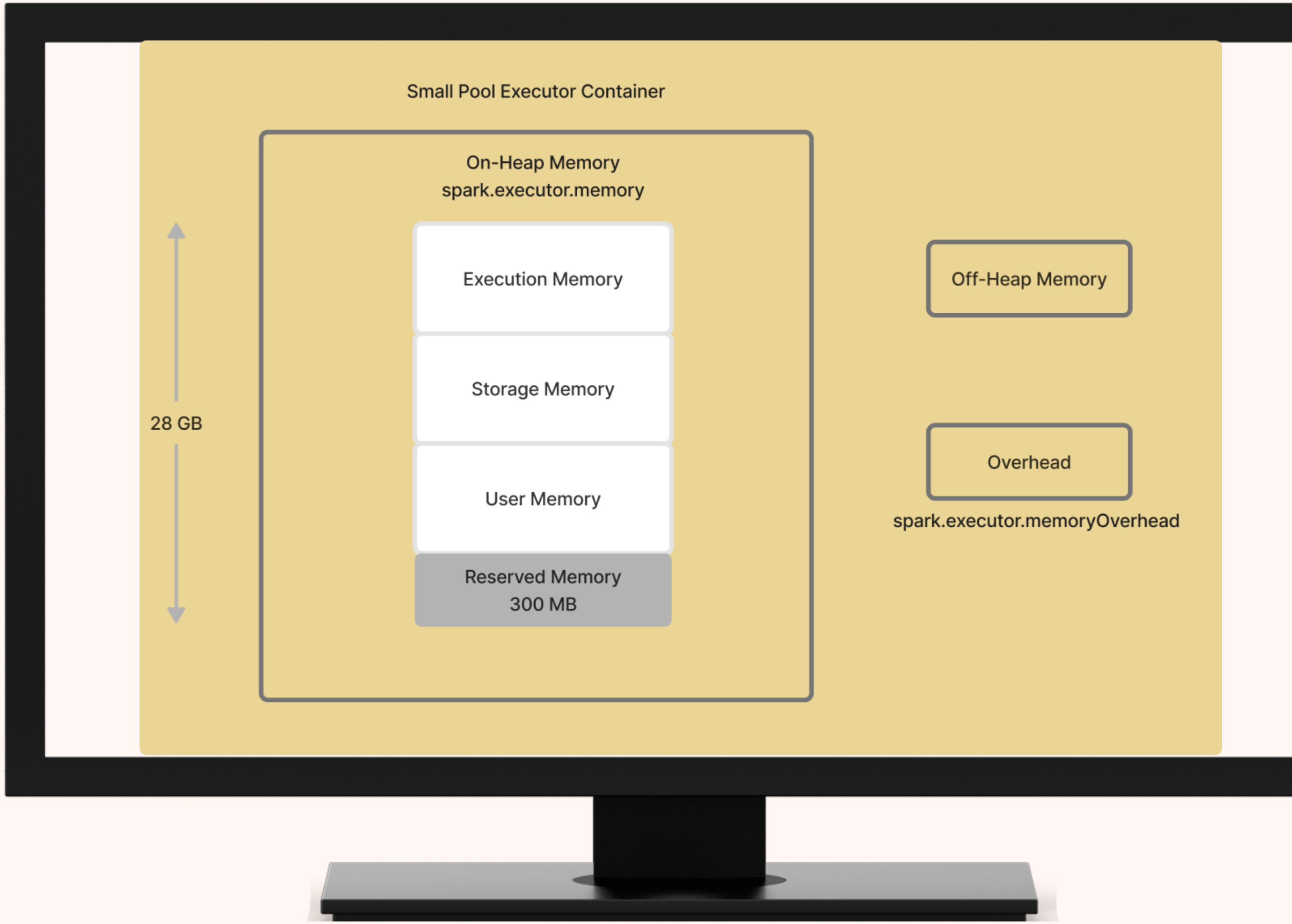
Executor Memory Management

On-Heap Memory:

- Reserved memory is fixed portion of memory that is allocated for system-related tasks.
- User Memory stores UDFs and user defined data structures like lists, Dictionaries etc,.
- Storage Memory stores all the cached/persisted data and broadcast variables.

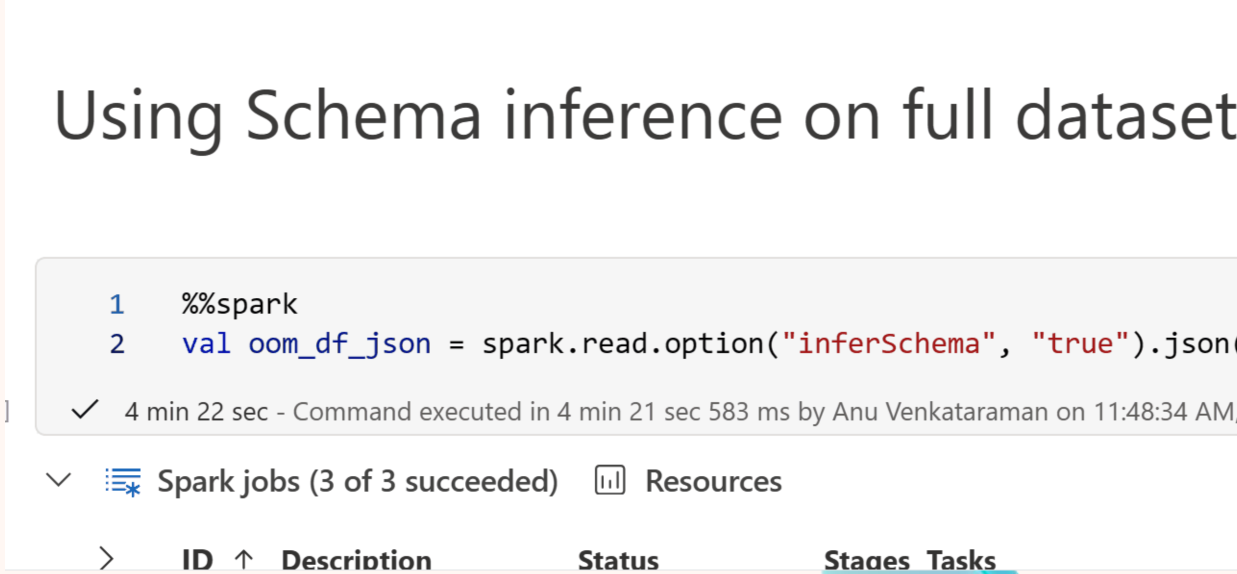
Handling Memory Limits:

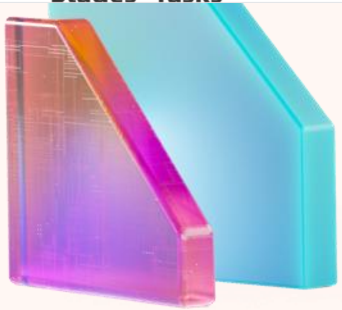
- If RDD or DF reaches limit of storage memory, it **spills to disk** which impacts performance.
- Execution Memory stores any data that is generated or required by processing. If reaches limit of Execution memory, it spills to disk.
- Boundary between Storage and Execution Memory is movable.



Spark Basics: Dos and Don'ts

- Use Serialized data formats like Avro, Parquet, ORC:
  - Stores data in Binary format. Optimized for storage and processing.
  - Embedded schema.
- When using non serialized formats like JSON or XML:
  - Human-readable text format, which tends to be less space-efficient. Formats that are slow to serialize objects into, or consume a large number of bytes, will greatly slow down the computation.
  - Parsing XML files can be slower than other formats due to the overhead of parsing the XML tags and type casting of every row and column.
  - If using schema inference, Spark (by default) reads its whole content to create a valid schema.
- If reading JSON or XML:
  - Use static master schema (recommended).
  - Use .options(samplingRatio=0.1) to speed up reads.
    - It doesn’t support schema evolution.
    - There's a risk if the sample size doesn't accurately represent the entire files, reads may fail.
    - To avoid this, infer and persist the schema from a set of sample files that accurately reflect the entire dataset.





Spark Basics: Dos and Don'ts

Using Schema inference on full dataset

```
%%spark
val oom_df_json = spark.read.option("inferSchema", "true").json("abfss:/
```

✓ 4 min 22 sec - Command executed in 4 min 21 sec 583 ms by Anu Venkataraman on 11:48:34 AM, 5/13/24

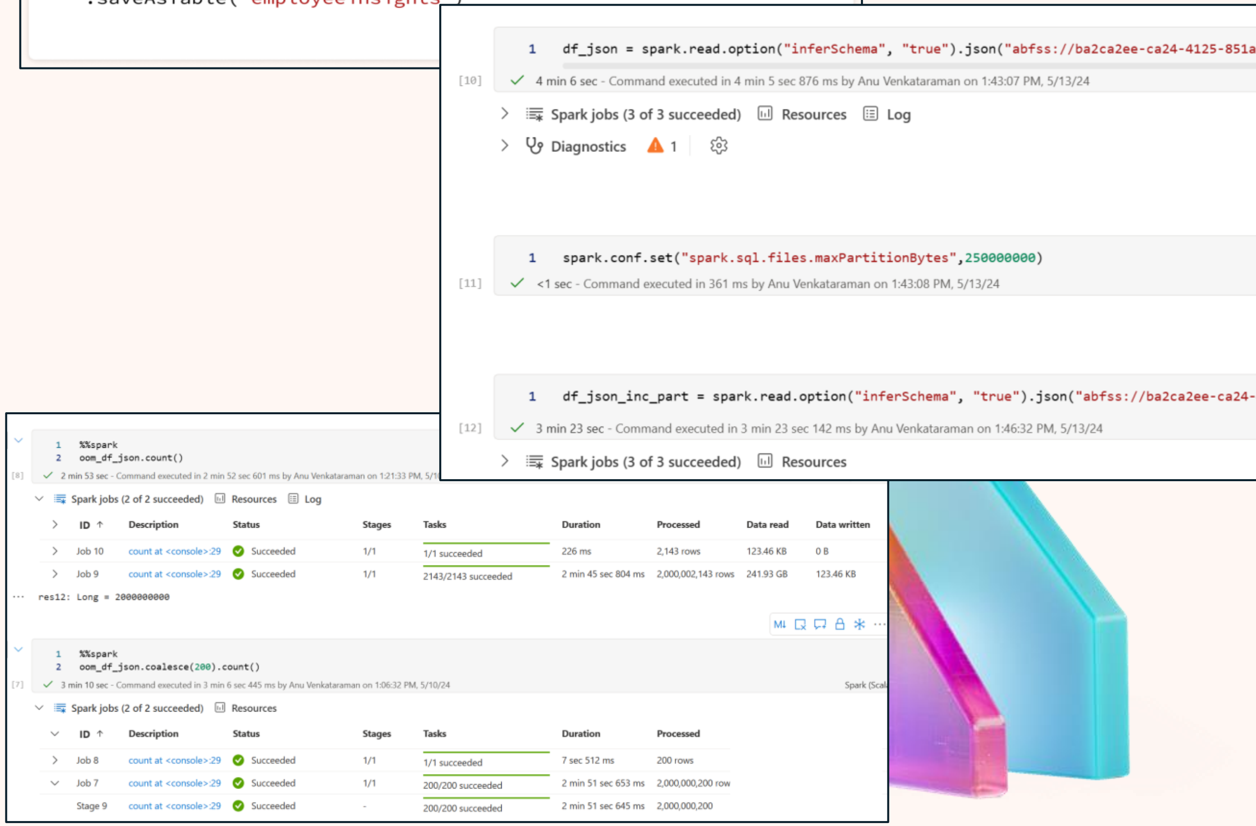
Spark jobs (3 of 3 succeeded)   Resources

ID ↑	Description	Status	Stages	Tasks	Duratio

Spark Basics: Dos and Don'ts

- Partitions are the basic units of parallelism and are key for performance.
  - If we partition a 1 GB data into 100 partitions, Spark will concurrently process these 100 partitions as individual tasks in parallel.
  - Reads: Spark decides on the number of partitions based on the file size input. Tweak *spark.sql.files.maxPartitionBytes* and benchmark.
  - Coalesce vs Repartition:
    - Repartition is used to increase or decrease partitions. Repartition is an **expensive operation as it shuffles** the data. Results in almost equal sized partitions.
    - Coalesce only reduces partitions and it avoids shuffles.
  - Tweak *spark.sql.shuffle.partitions* (default is 200) to optimize the number of partitions when shuffling large data
  - It often involves **experimentation and tuning** to find out optimal number of partitions *(and this changes with data volume and shape, thus can evolve)*

```python
from pyspark.sql.functions import col, sum, avg

df_json = spark.read.option("inferSchema", "true").json("file_path")

aggregated_df = df_json.groupBy("country", "state").agg(
    avg("salary").alias("avg_salary"),
    sum("salary").alias("total_salary")
)

aggregated_df.write \
    .mode("append") \
    .format("delta") \
    .saveAsTable("employeeinsights")
```



Spark Basics: Fabric Advantage

- Fabric autotuning enhances query performance after 20 to 25 iterations based on historical workload data to adjust parameters such as:
  - spark.sql.shuffle.partitions
  - spark.sql.autoBroadcastJoinThreshold
  - spark.sql.files.maxPartitionBytes

- Fabric's default setting enables Adaptive Query Execution (AQE) that uses runtime statistics to optimize performance:
  - Changing join type
  - Handles partition skews
  - Coalescing shuffle partitions

Evolution

- Spark 1.X introduced Catalyst optimizer and Tungsten execution engine.
- Spark 2.X introduced cost-based optimization.
- Spark 3.0 introduced AQE, taking optimization to the next level by using runtime statistics for dynamic reoptimization.

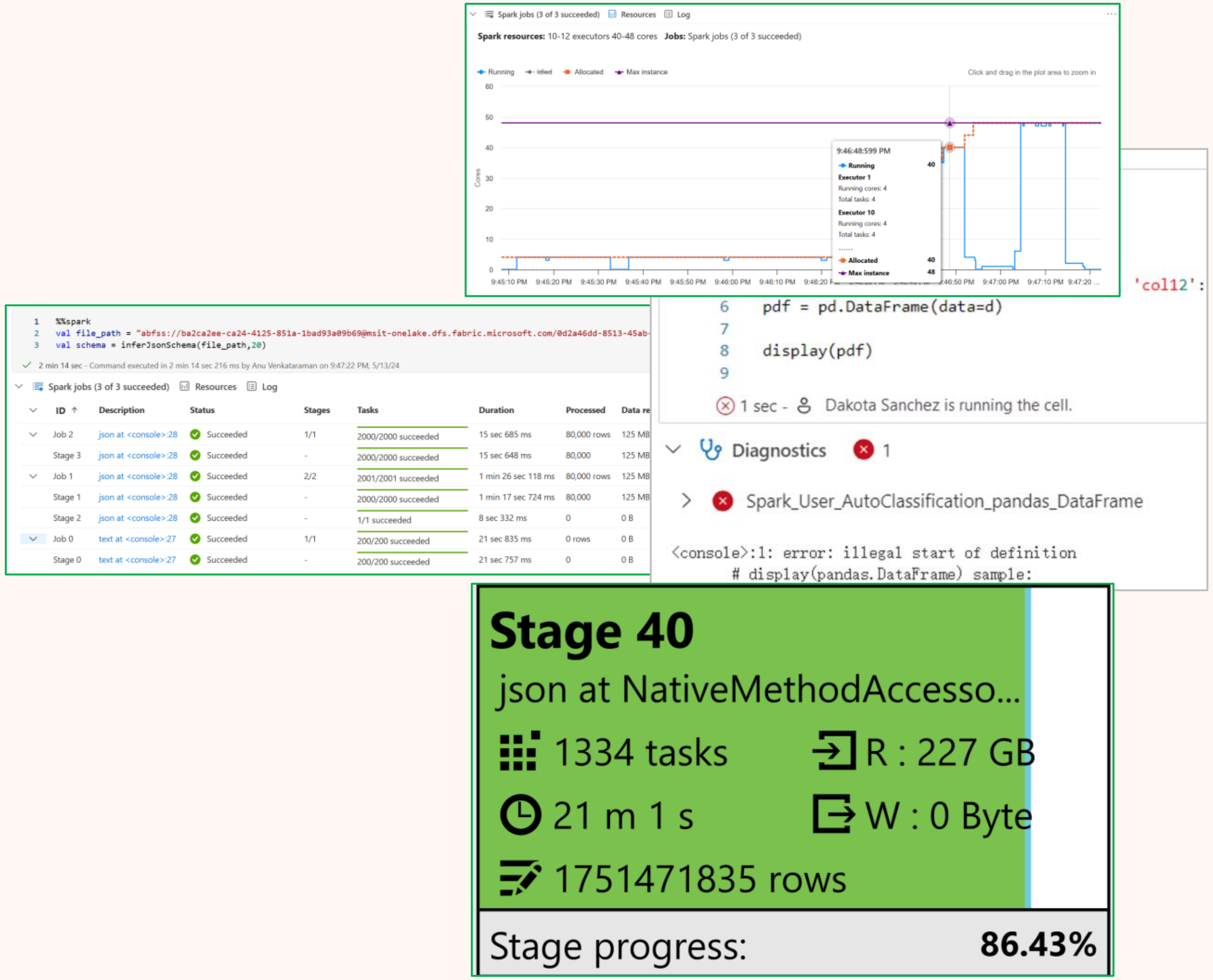
Spark Basics: Dos and Don'ts

```
 2024-05-10 13:11:47,326 INFO CosmosItemsDataSource [Thread-33]: Instantiated CosmosI
 2024-05-10 13:11:48,359 INFO InMemoryFileIndex [Thread-33]: It took 1024 ms to list
 2024-05-10 13:11:48,415 INFO VegasOptimizerRule$ [Thread-33]: Cache size Some(50)
✓2024-05-10 13:11:48,416 INFO VegasOptimizerRule$ [Thread-33]: Cache size Some(50)
 2024-05-10 13:11:48,416 INFO VegasOptimizerRule$ [Thread-33]: Cache size Some(50)
 2024-05-10 13:11:48,422 INFO  [Thread-33]: [Autotune] Autotune query tuning is enabl
 2024-05-10 13:11:48,435 WARN MetricsConfig [Thread-33]: Cannot locate configuration:
✓2024-05-10 13:11:48,439 INFO MetricsSystemImpl [Thread-33]: Scheduled Metric snapsho
 2024-05-10 13:11:48,439 INFO MetricsSystemImpl [Thread-33]: azure-file-system metric
 2024-05-10 13:11:48,684 INFO  [Thread-33]: [Autotune] Success. Query tuning complete
✓2024-05-10 13:11:48,692 INFO FileSourceScanPlan [Thread-33]: Pushed Filters:
 2024-05-10 13:11:48,692 INFO FileSourceScanPlan [Thread-33]: Post-Scan Filters: (len
 2024-05-10 13:11:48,699 INFO MemoryStore [Thread-33]: Block broadcast_89 stored as v
 2024-05-10 13:11:48,713 INFO MemoryStore [Thread-33]: Block broadcast_89_piece0 stor
```

Monitor Spark Jobs in Fabric Notebook

To monitor inside the Notebook:

- Spark Job Progress
- Resource Usage
- Spark Advisor Recommendations
- Spark Advisor Skew Detection
- Driver Logs
- Spark UI
  - ✓ Spark metrics
  - ✓ DAG
  - ✓ Executors
  - ✓ Spark SQL Execution plan



Sparklens: Profiling Tool

Open-source Spark profiling tool

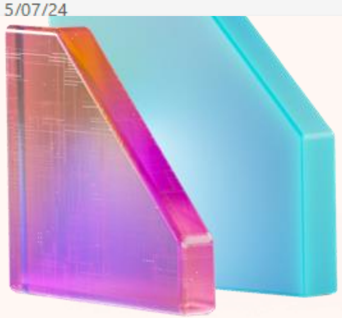
- Note: Sparklens is not developed or owned by Microsoft. Please refer to the Sparklens github page to learn more.
- When you are working with a spark application, you would typically use the profiler in the following scenarios:
- To reduce the Spark application execution time
- To evaluate if the application will be performant even with lesser resources

Sparklens reports:

- Driver and Executor wall clock time
- Critical path
- Simulates wall clock time by adding or reducing executors
- Compute wastage and utilization
- Task Skew

```
QNL = sc._jvm.com.qubole.sparklens.QuboleNotebookListener.registerAndGet(sc._jsc.sc())

if (QNL.estimateSize() != QNL.getMaxDataSize()):
    QNL.purgeJobsAndStages()
    startTime = int(round(time.time() * 1000))

    df_json = spark.read.option("inferSchema", "true").json("abfss://ba2ca2ee-ca24-4125-851a-1bad93a09b6

    aggregated_df = df_json.groupBy("country", "state") \
    .agg(avg("salary").alias("avg_salary"),
         sum("salary").alias("total_salary"))

    aggregated_df.write.mode("append").format("delta").saveAsTable("employeeinsights")

    endTime = int(round(time.time() * 1000))
    time.sleep(QNL.getWaiTimeInSeconds())
    print(QNL.getStats(startTime, endTime))
```

✓ 56 min 58 sec - Command executed in 56 min 57 sec 788 ms by Anu Venkataraman on 9:28:04 PM, 5/07/24



Sparklens: Actions after Profiling

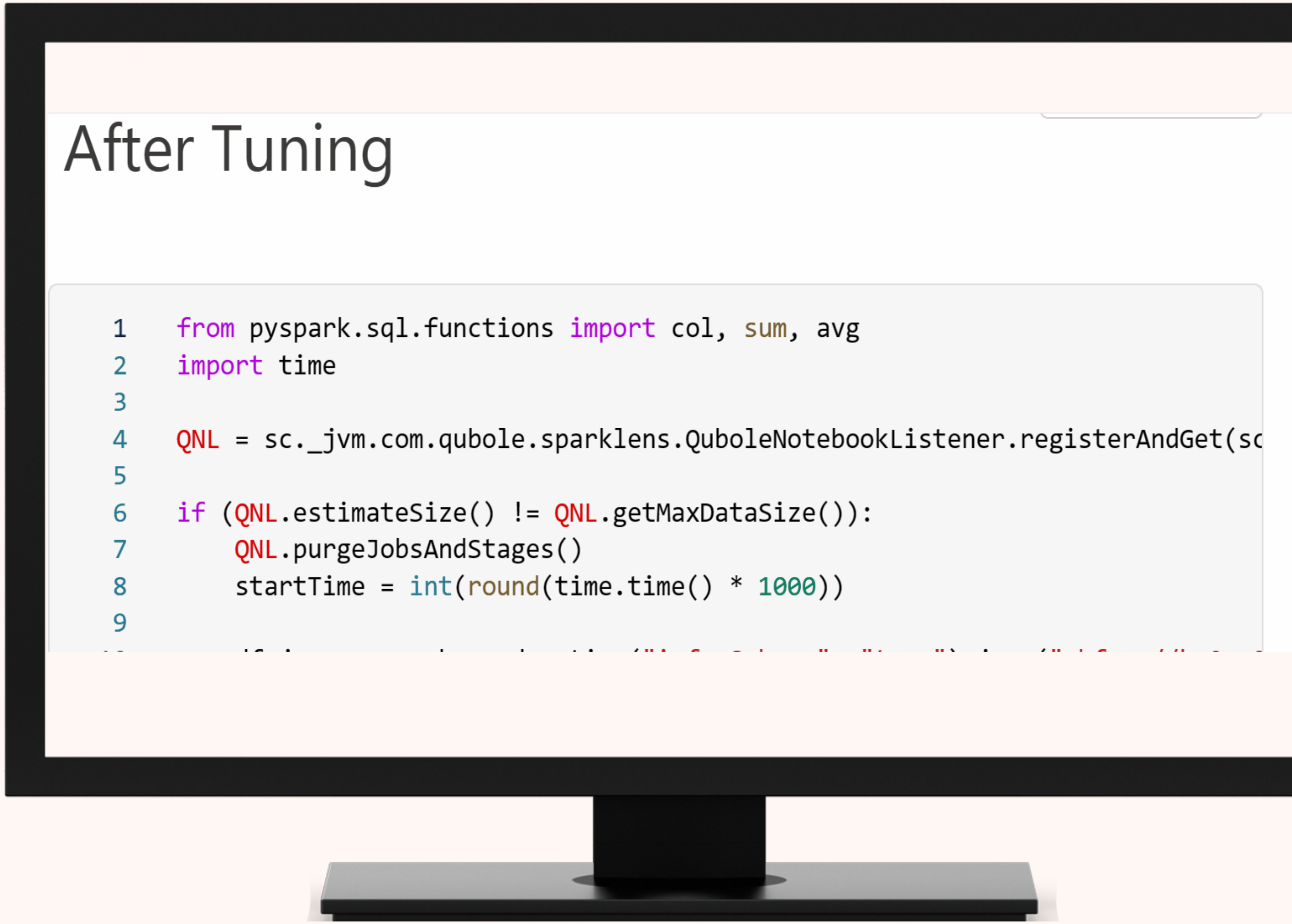
Running Sparklens after increasing the min executors to 5 (8 Cores and 56 GB Memory) from 2 (4 Cores). Execution time is reduced to 8 min and 46 sec from ~58 mins.

Based on the profiling report, you can customize:

- Driver Cores
- Executor Cores
- Autoscaling Max Nodes
- Min and Max Executor Instances

Application

- Optimize Parallelism and data partitioning
  - Increasing the number of partitions can also lead to higher overhead in terms of task scheduling. So benchmark for optimal partitions.
- Tune Spark Parameters



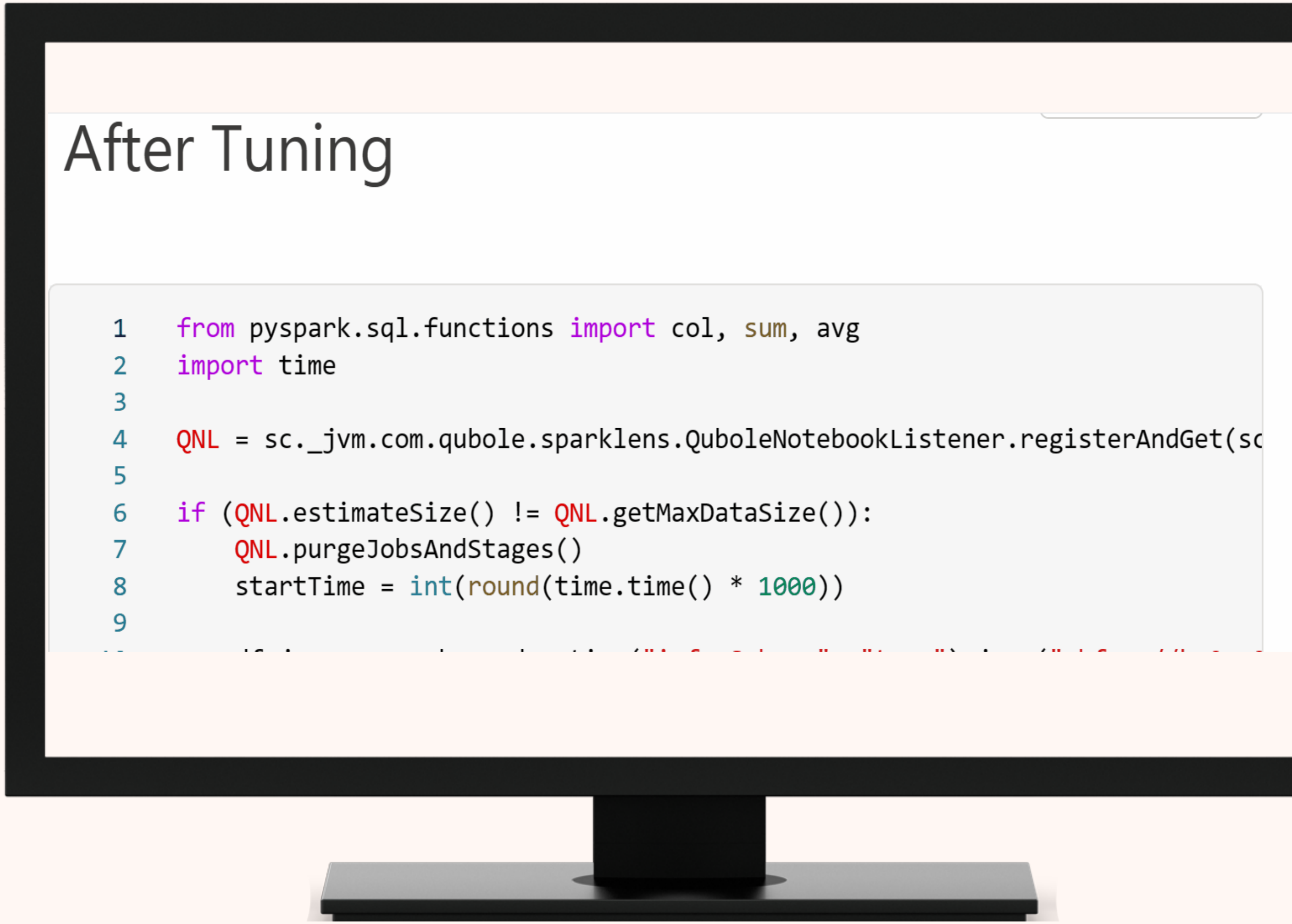
Sparklens: Advantages and Limitations

Advantages

- Compatibility: Sparklens is compatible with Spark 3.x after configuring the build.sbt file
- License: It's an open-source tool and free to use.
- Ease of use: Sparklens' reports are user-friendly and easy to interpret.

Limitations

- Not owned by Microsoft.
- Dynamic Allocation Impact: With dynamic allocation enabled, model error increases.
- Contributor Status: Unfortunately, there have been no active contributors for the past 3 years.



Fabric Spark Native Execution Engine

Native Execution Engine for Fabric Runtime 1.2 is currently available in public preview.

Spark processes run on Java Virtual Machine (JVM).

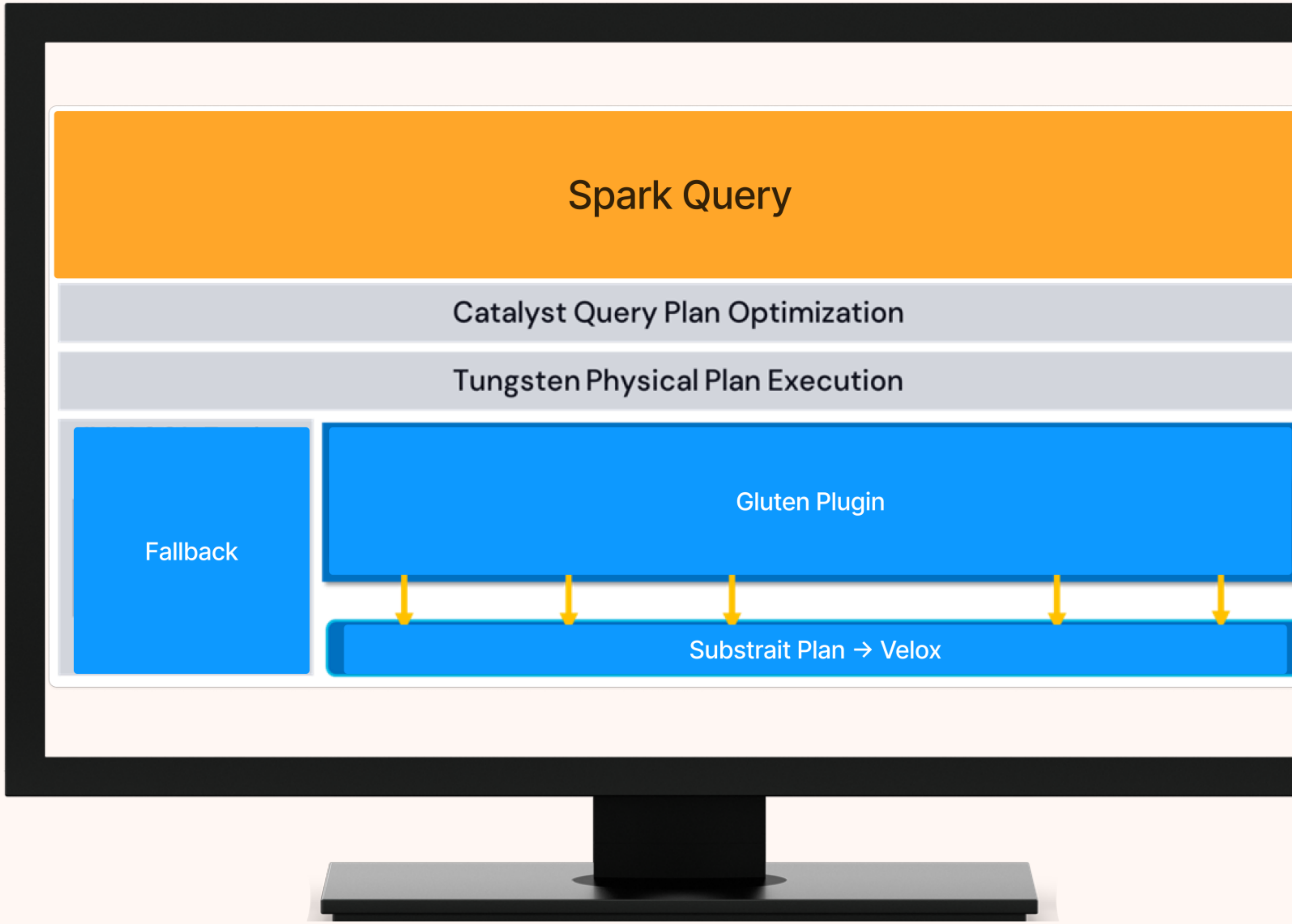
- Not compiled into machine code (like C++) that hardware can use
- garbage collection overhead

Gluten:

- Transforms Spark's whole stage physical plan to Substrait plan and send to native.
- Offloads performance-critical data processing to native library.

Velox: C++ engine that can execute code close to the machine, operates in columnar mode and uses vectorized processing

4x speed-up on the sum of execution time of all 99 queries in the TPC-DS 1TB.



Thanks to our sponsors and volunteers

slalom

avanade

cognizant

ADASTRA

Microsoft

Hitachi Solutions

Nigel Frank International
The Global Leader in Microsoft Recruitment
A Tenth Revolution Group Company