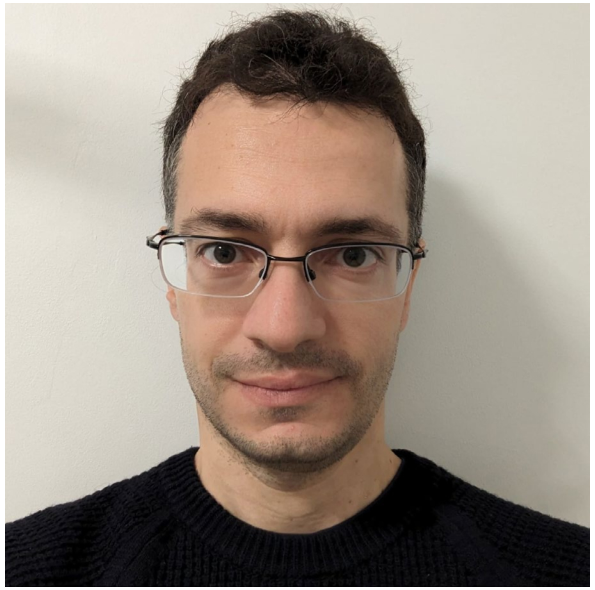
DATA+AI SUMMIT
BY databricks

DRASTICALLY REDUCING PROCESSING COSTS WITH DELTA LAKE

Generoso Pagano & Mauricio Jost
10-13 June 2024

About us



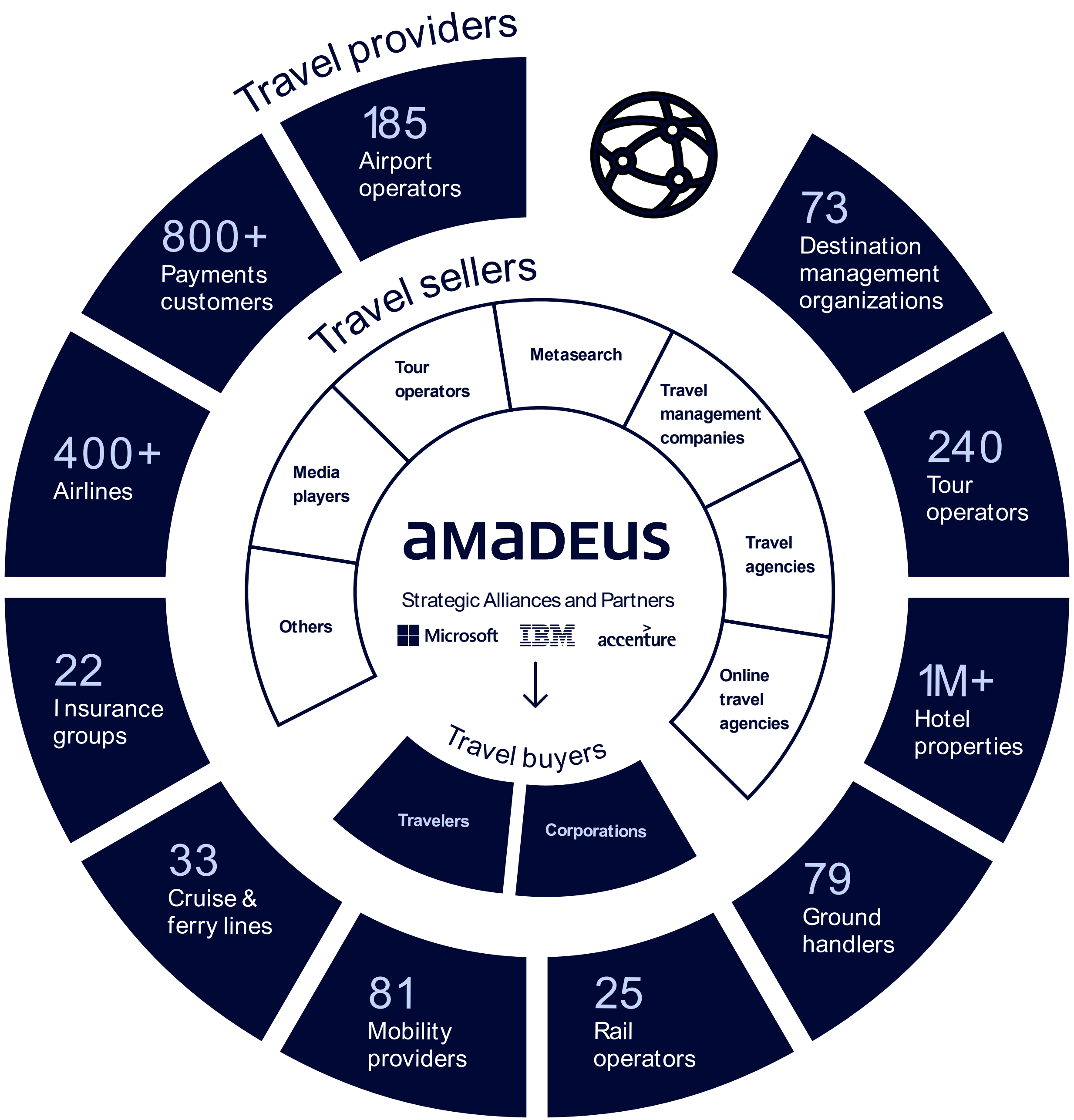
Generoso Pagano

Mauricio Jost

- **Principal Data Engineers** @Amadeus
- Mostly having fun with Scala, Spark and Delta Lake

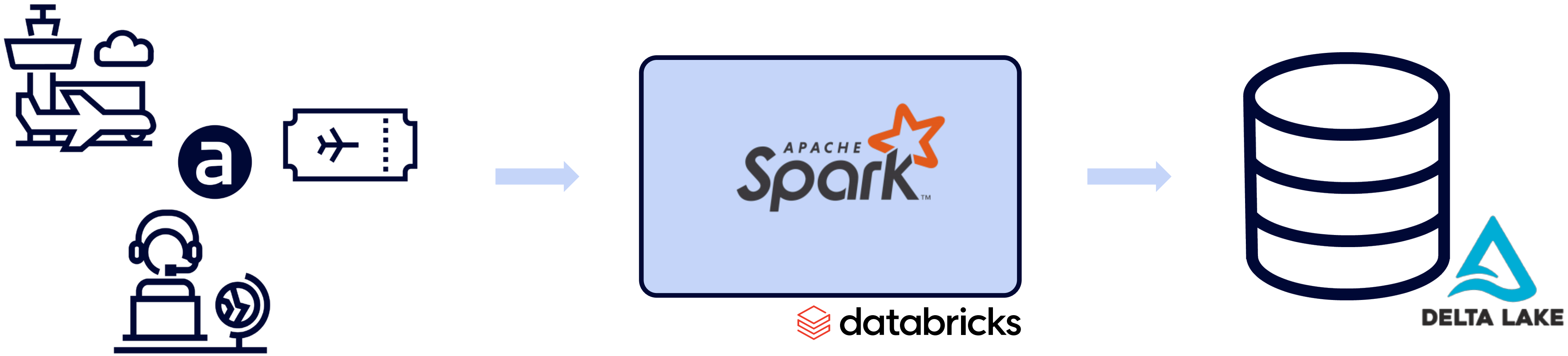
amadeus

Making travel simpler, smarter and smoother.



© Amadeus IT Group and its affiliates and subsidiaries

Our product



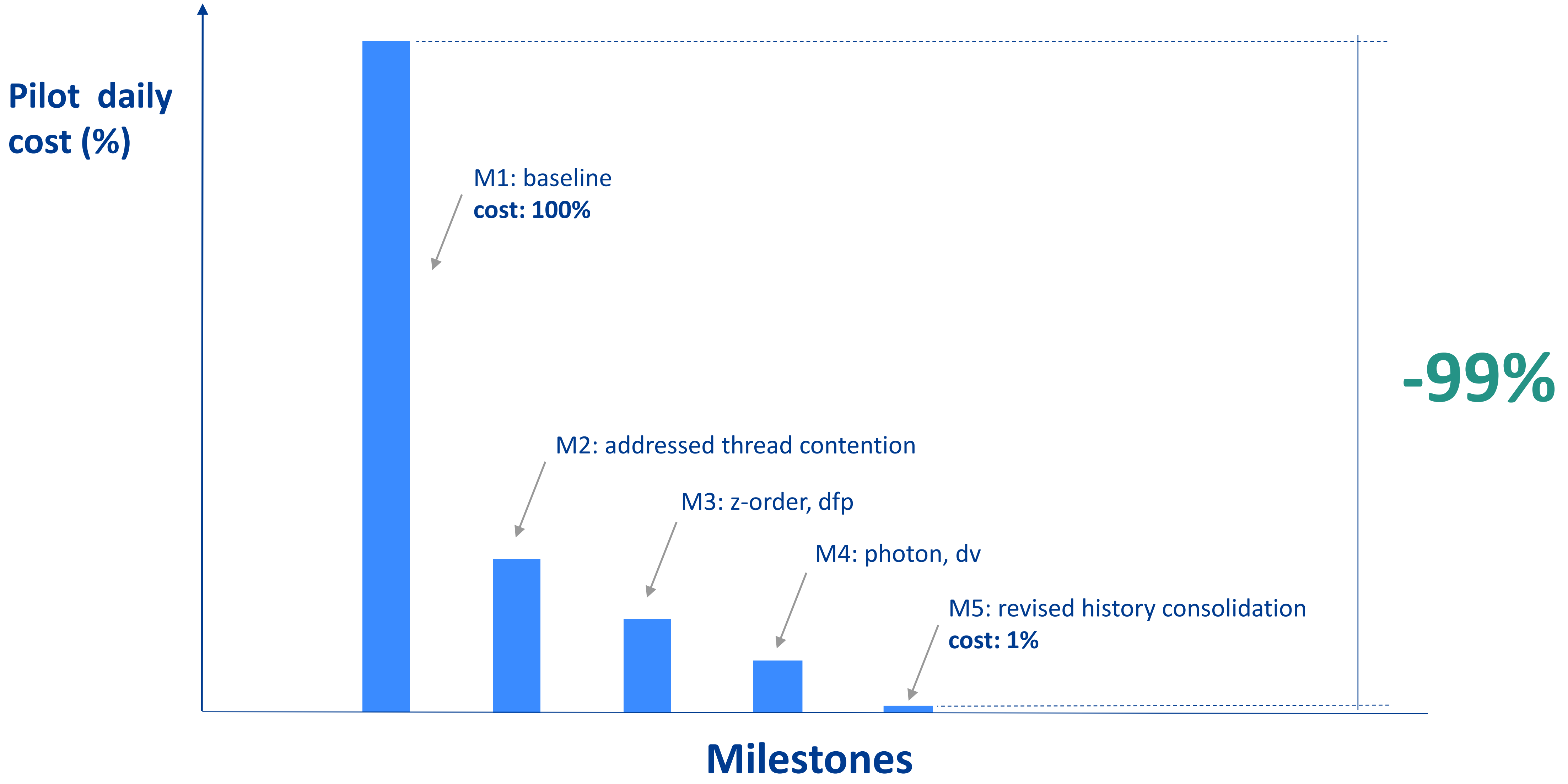
Challenging requirements

- 100s of output tables
- Several years of historical data
- History consolidation

A complex application

- Join/merge intensive
- 1000s of Spark jobs

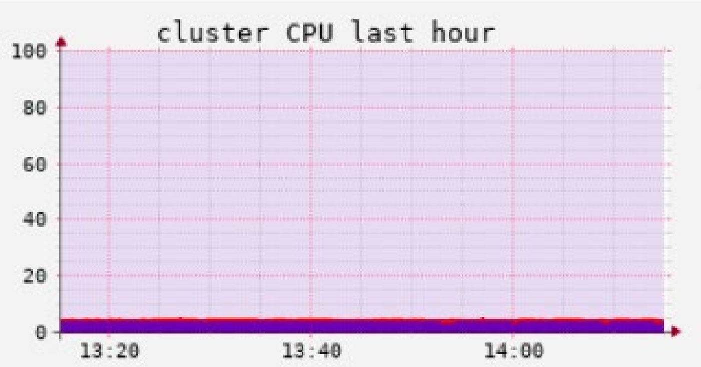
Our cost reduction journey



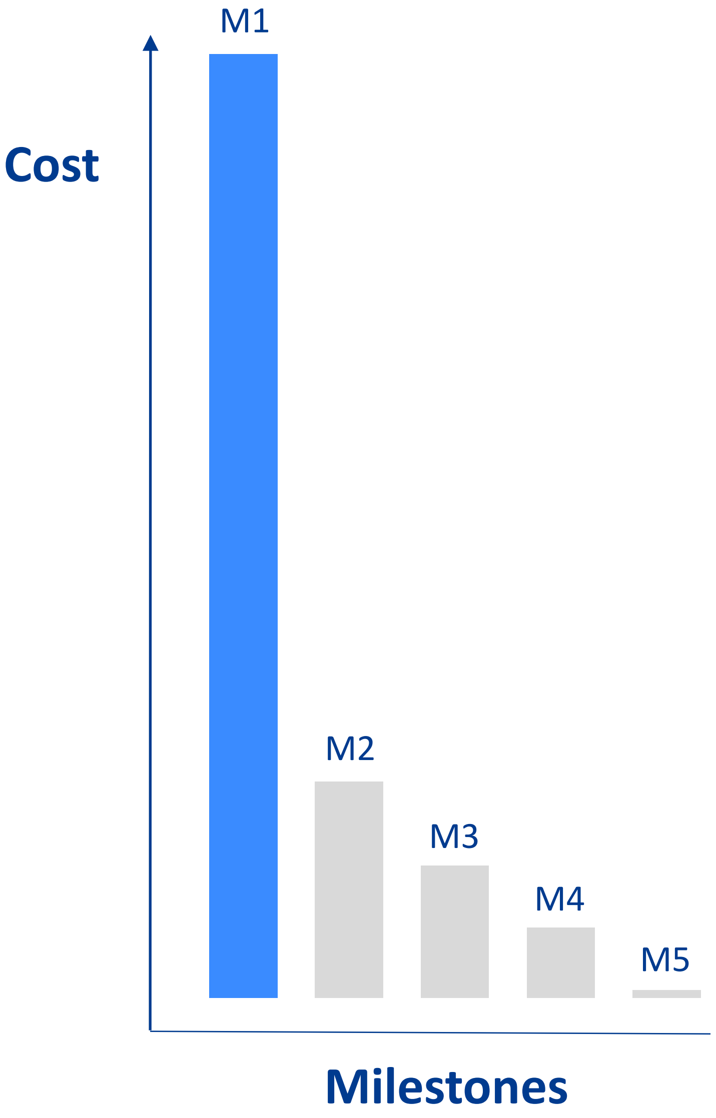
Journey Tracker #1

M1: Baseline (beginning of our journey)

- Functional correctness ✓
- Technical stability ✓
- Throughput below expectations ⚠



- CPU usage below 10% ⚠

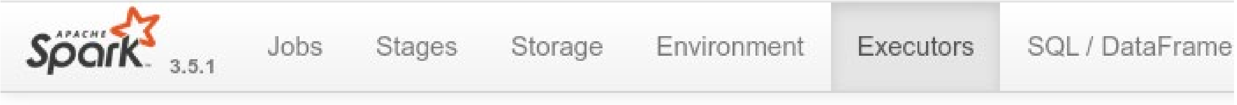


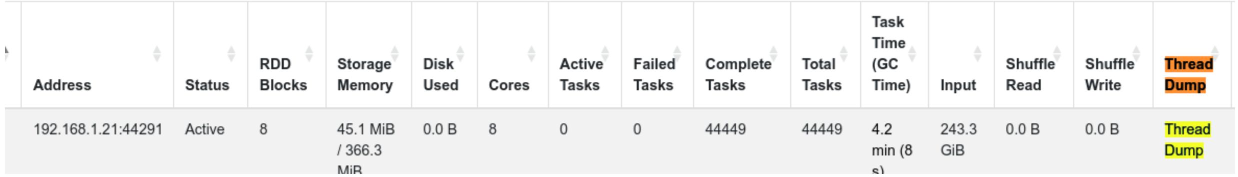
Why is CPU usage so low?

- But... JSON parsing is CPU intensive!

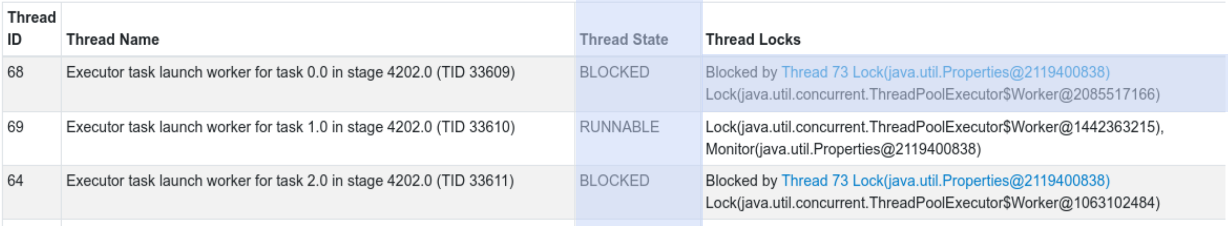
- Post-mortem Spark UI
  - Spark job not retained in UI ⚠
  - Unnamed jobs ⚠
  - Little workers information ⚠

- Live Spark UI
  - What are workers doing?
  - Most task threads BLOCKED ⚠
  - Thread contention (shared lock) ⚠

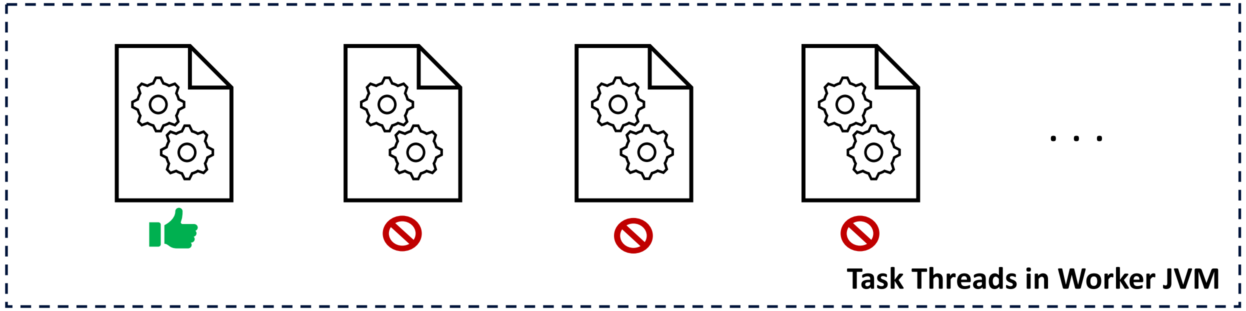




Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
192.168.1.21:44291	Active	8	45.1 MiB / 366.3 MiB	0.0 B	8	0	0	44449	44449	4.2 min (8 s)	243.3 GiB	0.0 B	0.0 B	Thread Dump

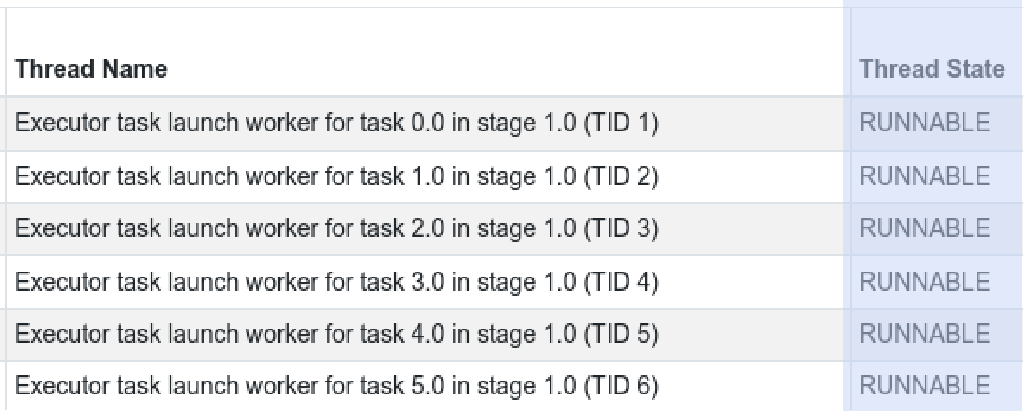


Thread ID	Thread Name	Thread State	Thread Locks
68	Executor task launch worker for task 0.0 in stage 4202.0 (TID 33609)	BLOCKED	Blocked by Thread 73 Lock(java.util.Properties@2119400838) Lock(java.util.concurrent.ThreadPoolExecutor$Worker@2085517166)
69	Executor task launch worker for task 1.0 in stage 4202.0 (TID 33610)	RUNNABLE	Lock(java.util.concurrent.ThreadPoolExecutor$Worker@1442363215), Monitor(java.util.Properties@2119400838)
64	Executor task launch worker for task 2.0 in stage 4202.0 (TID 33611)	BLOCKED	Blocked by Thread 73 Lock(java.util.Properties@2119400838) Lock(java.util.concurrent.ThreadPoolExecutor$Worker@1063102484)

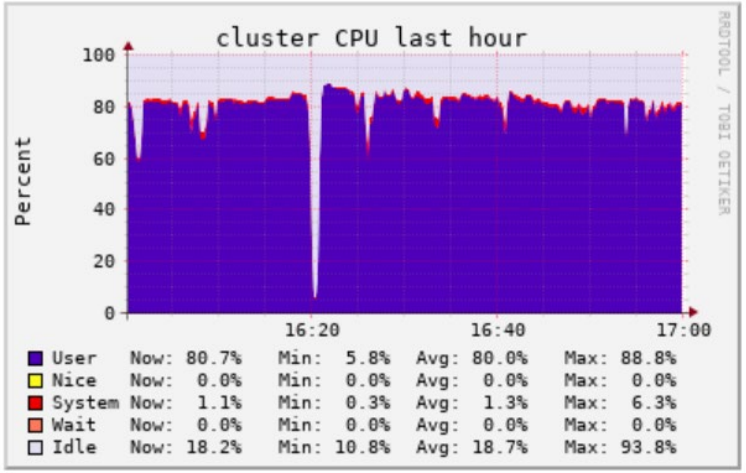


Addressing Thread Contention

- The culprit
  - Scala closure
  - Third-party library
  - Cache implementation
- Alternatives
  - Use built-in SQL functions
  - Change cache implementation ✓
- Change done, we retried and...
  - All threads RUNNABLE ✓
  - Much better CPU usage ✓



Thread Name	Thread State
Executor task launch worker for task 0.0 in stage 1.0 (TID 1)	RUNNABLE
Executor task launch worker for task 1.0 in stage 1.0 (TID 2)	RUNNABLE
Executor task launch worker for task 2.0 in stage 1.0 (TID 3)	RUNNABLE
Executor task launch worker for task 3.0 in stage 1.0 (TID 4)	RUNNABLE
Executor task launch worker for task 4.0 in stage 1.0 (TID 5)	RUNNABLE
Executor task launch worker for task 5.0 in stage 1.0 (TID 6)	RUNNABLE



Journey Tracker #2

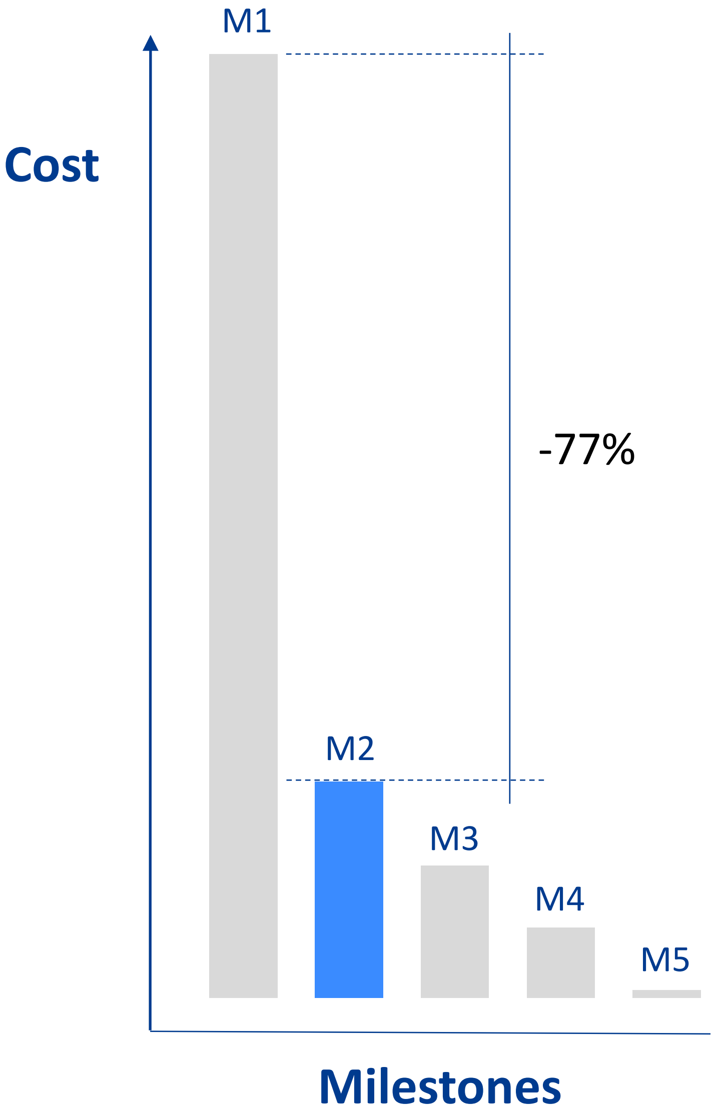
M2: Addressed thread contention

- Great cost reduction (-77%) ✓
- Costs still above target ⚠
- Difficult investigations ⚠

⬇

Metrics Unlocking

(Best Practices)



Metrics Unlocking (Best Practices)

Reproduce Perf. Problems in Notebooks

To **iterate fast**, **understand** the problem, identify metrics to **measure** it and **solve** it

https://github.com/AmadeusITGroup/spark-perf-hikes

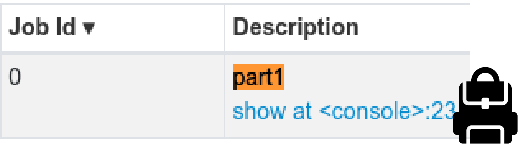
"Scenario reproduced"

Name every single Spark Job in the code

To quickly **associate** a Spark Job or SQL Query in the **UI** to the right section of **code**

SCALA

```scala
val sc = spark.sparkContext
sc.setJobDescription("part1")
// <part1 job here>
```



Job Id ▾	Description
0	part1 show at <console>:23

Persist Spark Events for post-mortems

To **dig into Spark Jobs stats at any time**, compare and understand them in depth

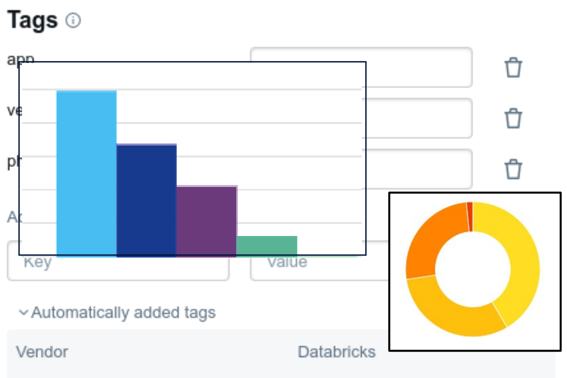
SCALA

```scala
val sc = spark.sparkContext
class L extends SparkListener {…}
sc.addSparkListener(new L())
```

```
+-------+---------+------------+---
| jobId | jobDesc | exCpuSecs |..
+-------+---------+------------+---
| 1     | part1   | 0.174051  |
+-------+---------+------------
```

Use Cluster or Pool Tags to measure costs

To **compare deployment costs & breakdown** via Cloud Provider cost dashboards

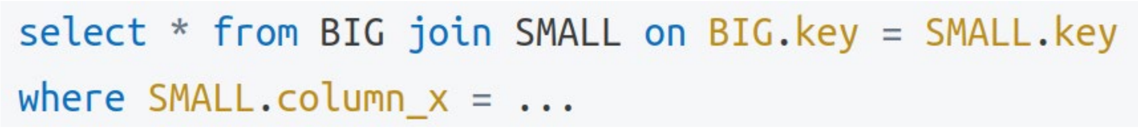


Why are network costs so high?

- Observations
  - Network is 70% of the costs
  - Join intensive application
    - *small* batches, joined with *big* tables
  - No data-skipping-friendly data layout

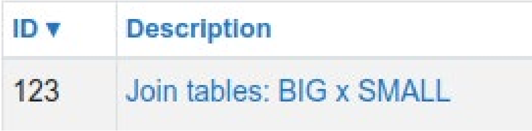
- Main suspect: **Read Amplification!**

- How do we assess that?



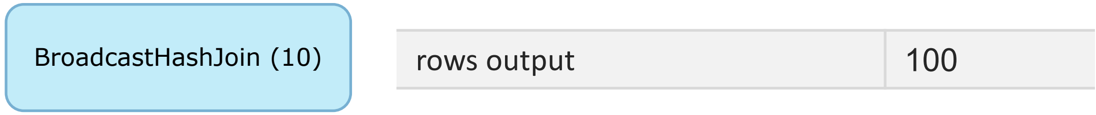
```
select * from BIG join SMALL on BIG.key = SMALL.key
where SMALL.column_x = ...
```

- SQL / DataFrame tab



ID ▾	Description
123	Join tables: BIG x SMALL

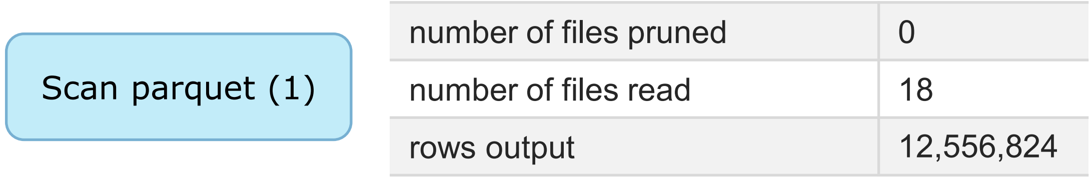
- Output rows in join



BroadcastHashJoin (10)

rows output	100

- Rows read in the BIG table

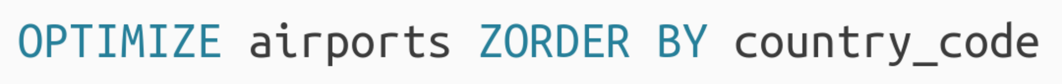


Scan parquet (1)

number of files pruned	0
number of files read	18
rows output	12,556,824

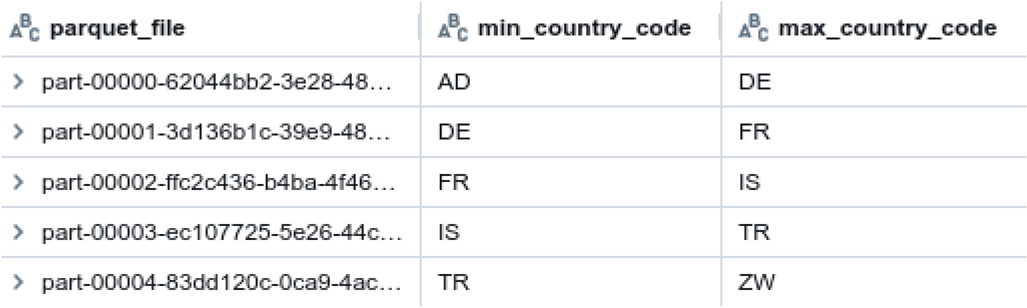
Z-ordering

- Co-locates related data in the same files
  - Enhances data skipping!
- Done within an OPTIMIZE
- Ensure column statistics are there!
- Explore delta log to see its effects



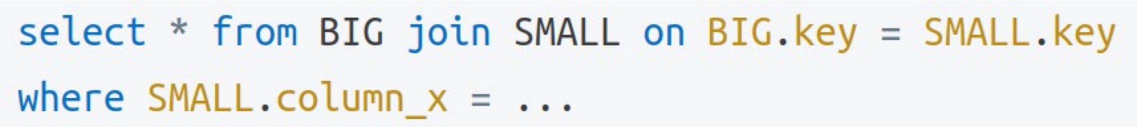
```
OPTIMIZE airports ZORDER BY country_code
```

CONFIG
delta.dataSkippingNumIndexedCols delta.dataSkippingStatsColumns



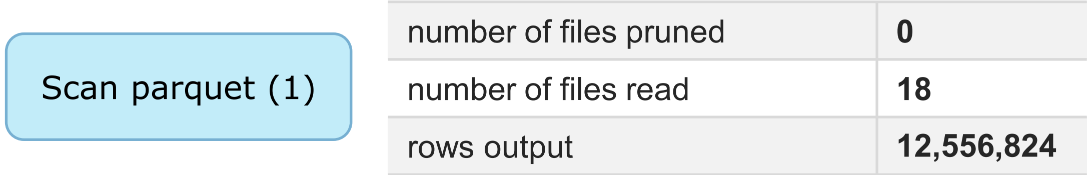
parquet_file	min_country_code	max_country_code
part-00000-62044bb2-3e28-48…	AD	DE
part-00001-3d136b1c-39e9-48…	DE	FR
part-00002-ffc2c436-b4ba-4f46…	FR	IS
part-00003-ec107725-5e26-44c…	IS	TR
part-00004-83dd120c-0ca9-4ac…	TR	ZW

Dynamic File Pruning (DFP)



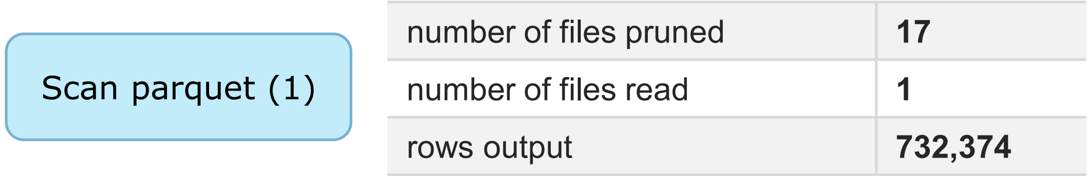
```sql
select * from BIG join SMALL on BIG.key = SMALL.key
where SMALL.column_x = ...
```

- *BIG* table z-ordered on *key*
- Dynamic filter based on *key* values in the *SMALL* table
- Filter pushed down to the scan phase of the *BIG* table



number of files pruned	**0**
number of files read	**18**
rows output	**12,556,824**

NO DFP



number of files pruned	**17**
number of files read	**1**
rows output	**732,374**

DFP

- Conditions for DFP to kick-in
  - Databricks
  - Broadcast join
  - Configuration

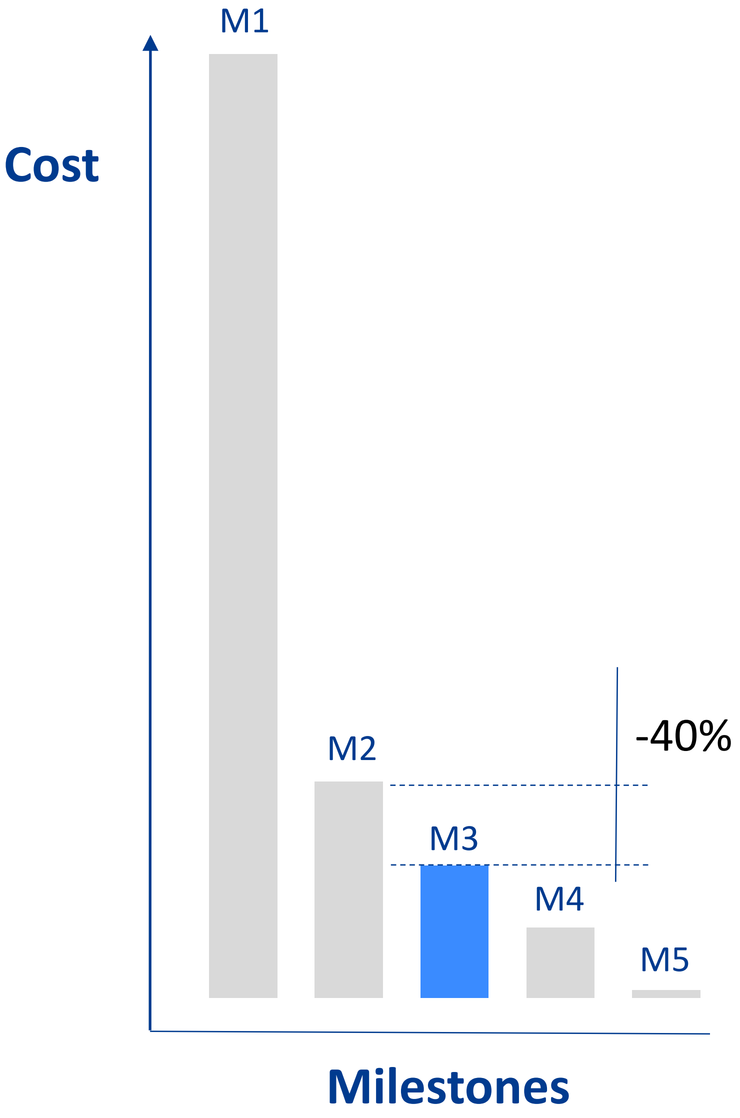
CONFIG

```
spark.databricks.optimizer.dynamicFilePruning
spark.databricks.optimizer.deltaTableSizeThreshold
spark.databricks.optimizer.deltaTableFilesThreshold
```

Journey Tracker #3

M3: Introduced Z-Order and DFP for joins

- Good cost reduction (-40%, mostly network) ✓
- Bad surprise
  - data skipping increased, but still low ⚠
  - keys hitting most files
- Where did the cost reduction come from?
  - optimize + z-order data compression ✓
  - co-locality of different versions for a given key
- Next: **focus on writes**

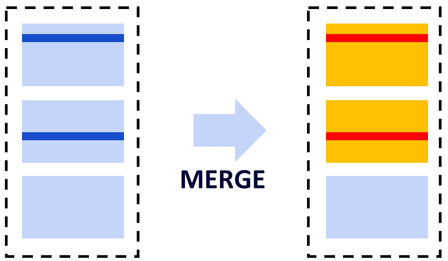


Write Amplification (WA) and Deletion Vectors (DV)

How to assess WA? Use History *operationMetrics*

- Observations
  - Expected to update ~1% of rows (**write**)
  - Expected to **read** 90% of rows
  - Measured high cost of writes, why? ⚠

Copy on Write



update 1% of rows spread on 2 parquet files

2 of 3 files completely rewritten

Write Amplification!

- Thanks Data & AI Summit 2023!
  - Copy on Write & **Merge on Read**
  - Predictive I/O: **Deletion Vectors** + Photon

SQL

```
-- enable deletion vectors
ALTER TABLE table SET TBLPROPERTIES(
  delta.enableDeletionVectors = true);
-- simple upsert
MERGE INTO table USING miniBatch
ON table.id = miniBatch.id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *;
```

SCALA

```
scala> table.history()/*...*/.show()
+---------+-------------+-------------+
|operation|numSourceRows|numOutputRows|
+---------+-------------+-------------+
|MERGE    |10   ←✓→     | 10  ~~660~~ |
+---------+-------------+-------------+
```

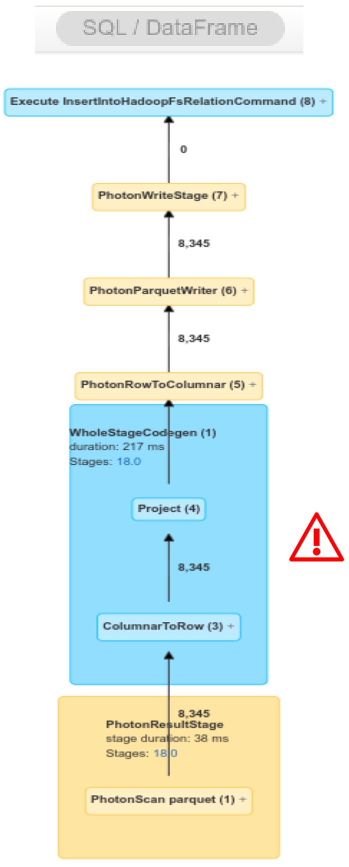
Deletion Vectors (on Merge) and Photon

- Photon needed (DBR 13.3LTS)
  - Enabled, but overall cost increased ⚠
- Thanks for the help Databricks!
- Photon underused ⚠
  - Query not fully supported
  - Incompatible Spark Settings ⚠

CONFIG
`spark.memory.offHeap.enabled = false`

- Enabled Off-Heap and...
  - Deletion Vectors on Merge active ✓
  - Photon much better used ✓

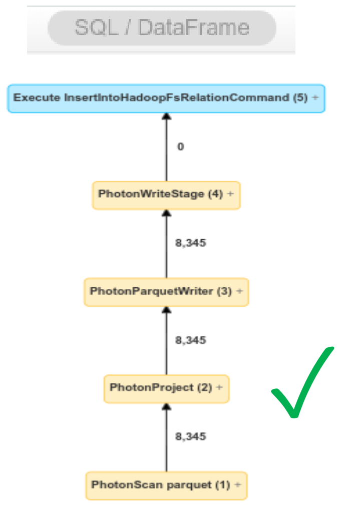
`UDF (not supported by Photon)`



▼Details

```
== Photon Explanation ==
Photon does not fully
support the query
because: UDF(name#24) is
not supported:...
```

`Built-in function (supported)`



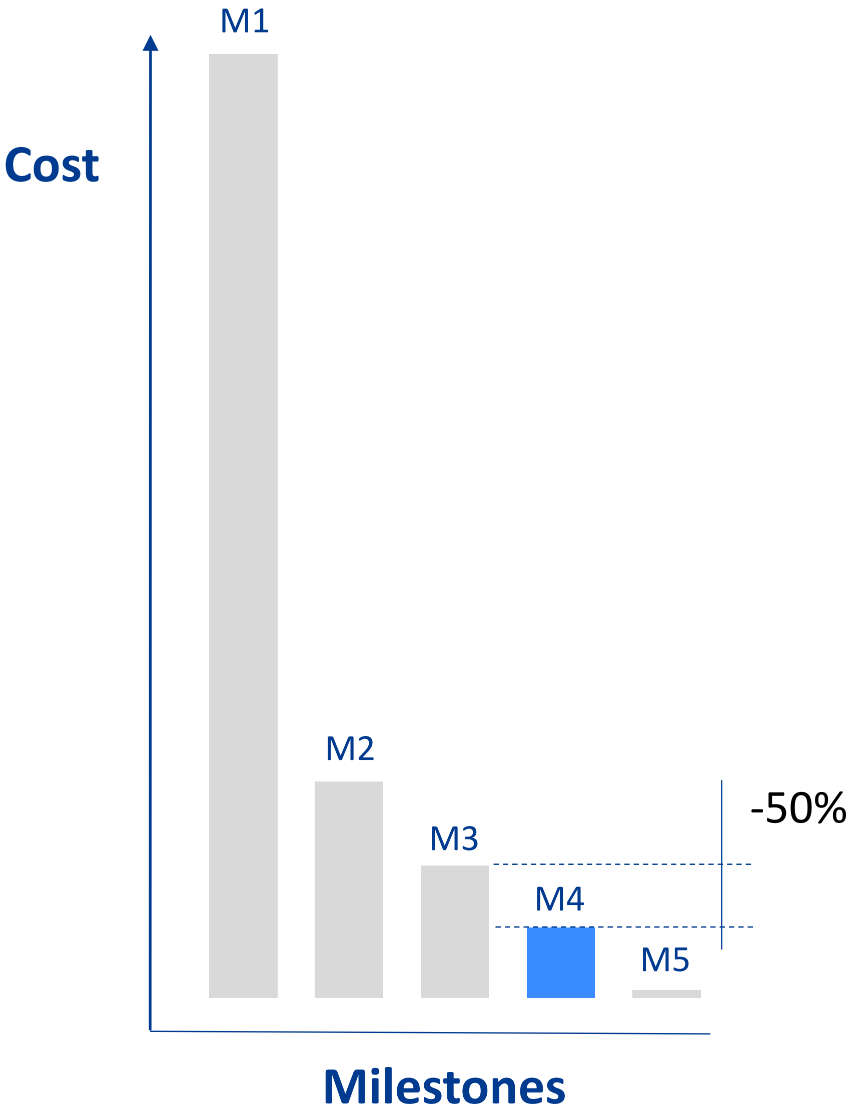
▼Details

```
== Photon Explanation ==
The query is fully
supported by Photon.
```

Journey Tracker #4

M4: Enabled Photon and Deletion Vectors

- Good cost reduction (-50%) ✓
- Spark Settings are good enough ✓
- Read amplification still biggest fish ⚠



History consolidation

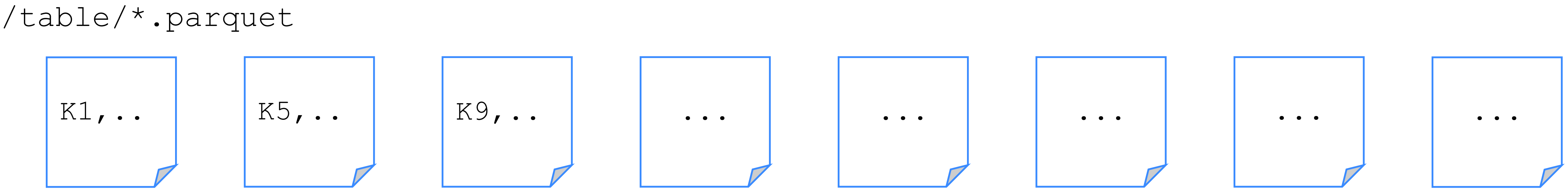
INPUT

key	version
K1	4
..	..

TABLE

key	version	last
K1	1	false
K1	2	false
K1	3	**~~true~~false**
K1	**4**	**true**
..	..	..

- Compute a patch
  - Join
  - Window function
- Merge it



- Most files contain at least one of the input keys ⚠
- High read amplification ⚠

We only need to read the version where *last = true*

Partition pruning to the rescue

INPUT

key	version
K1	4
..	..

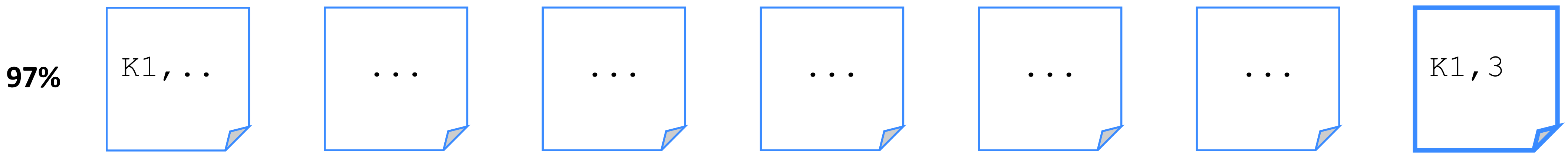
TABLE

key	version	last
K1	1	false
K1	2	false
K1	3	~~true~~false
K1	**4**	**true**
..	..	..

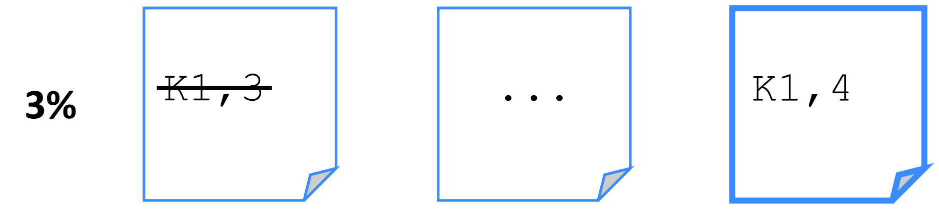
In JOIN & MERGE

TABLE.last = true

/table/**last=false**/*.parquet



/table/**last=true**/*.parquet

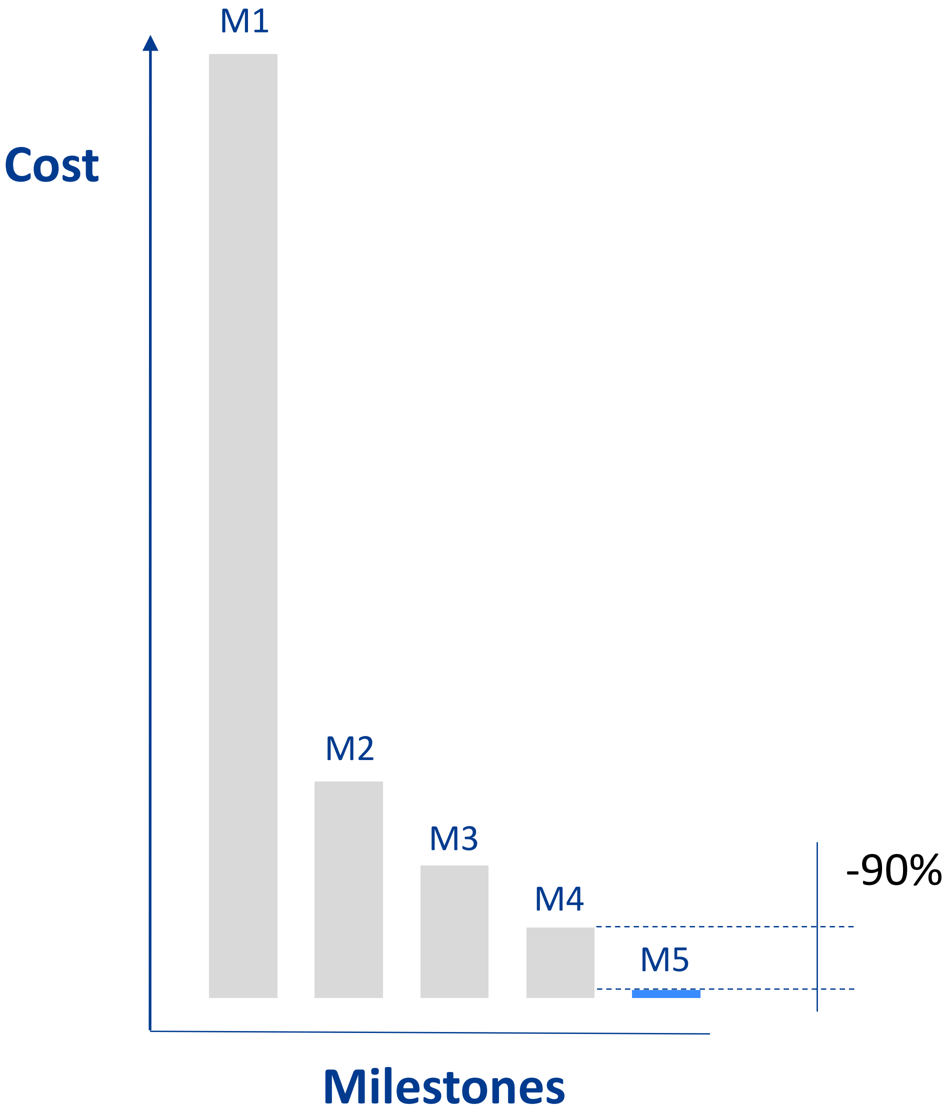


- Only read 3% of the data ✓
- Only do soft delete and append ✓

Journey Tracker #5

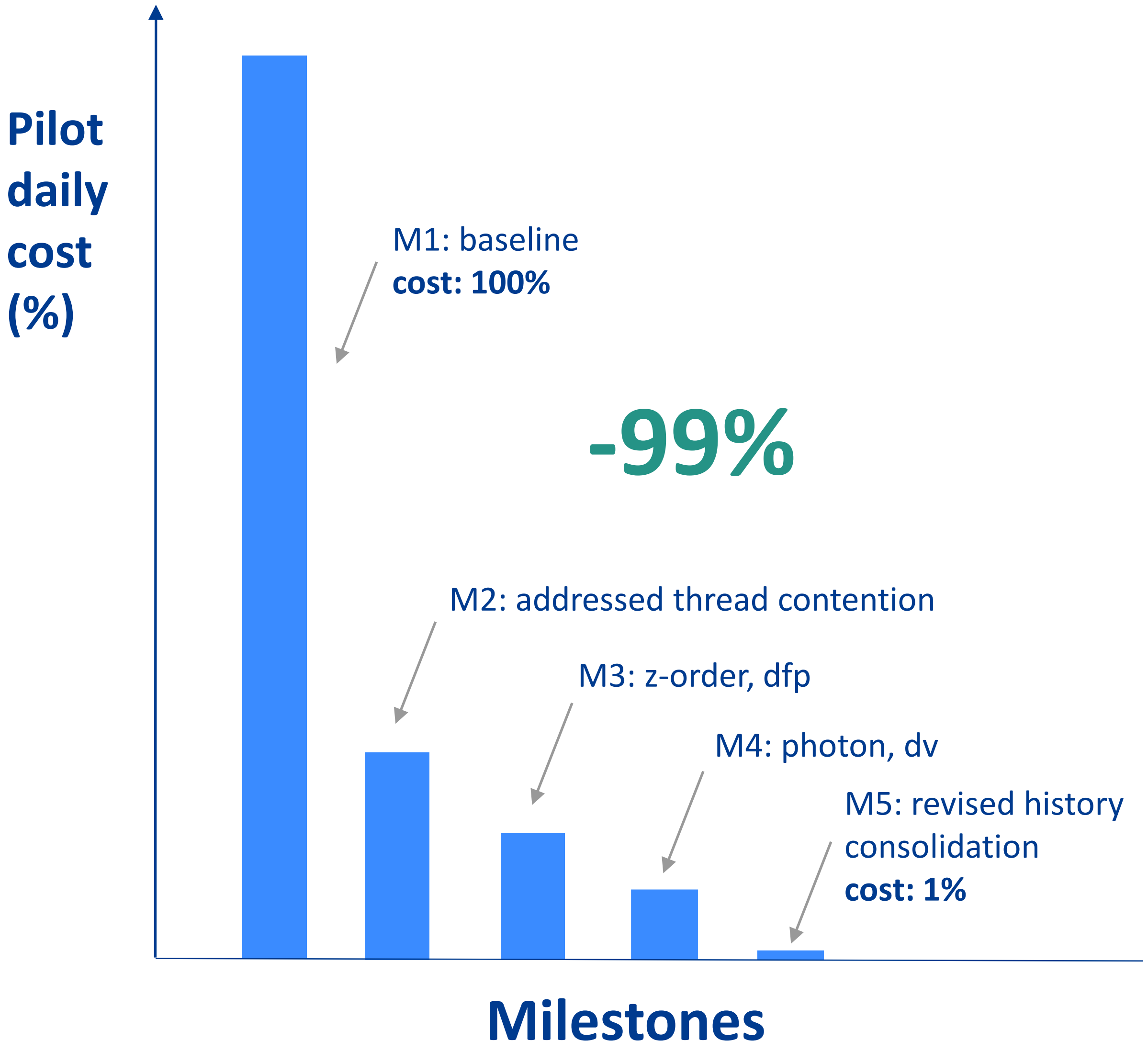
M5: Revised history consolidation

- Huge cost reduction (**-90%**) ✓
- Target cost point reached ✓
- Lesson learned
  - Technical + functional understanding = best performance
- What's next in the journey?
  - Share it at **DAIS 2024** :)



aMaDEUS | databricks

Conclusions



Lessons learned

- Be ready to iterate ✓
- Investigate rigorously ✓
- Ask for help ✓
- Be ambitious ✓

amadeus

Thank you

https://github.com/AmadeusITGroup/spark-perf-hikes

DATA+AI SUMMIT