

Delta Lake data layout optimization

Sabir Akhadov

Software Engineer

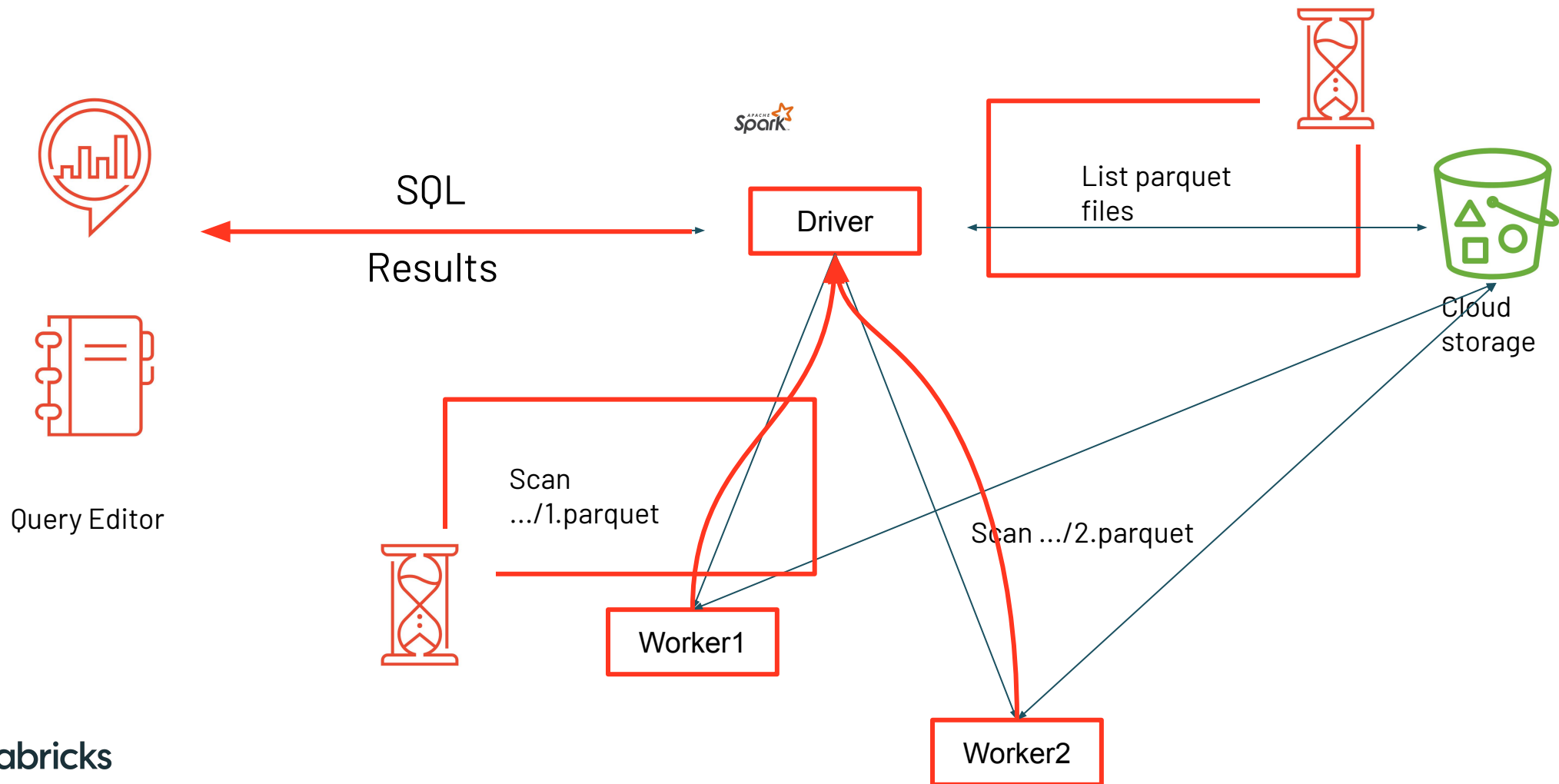
About Databricks and the presenter

Databricks: founded by the original creators of Apache Spark™ with 2000+ employees (engineering in San Francisco, Amsterdam, Toronto, Seattle and Berlin).

Sabir Akhadov: Software Engineer @ Databricks Amsterdam since 2019

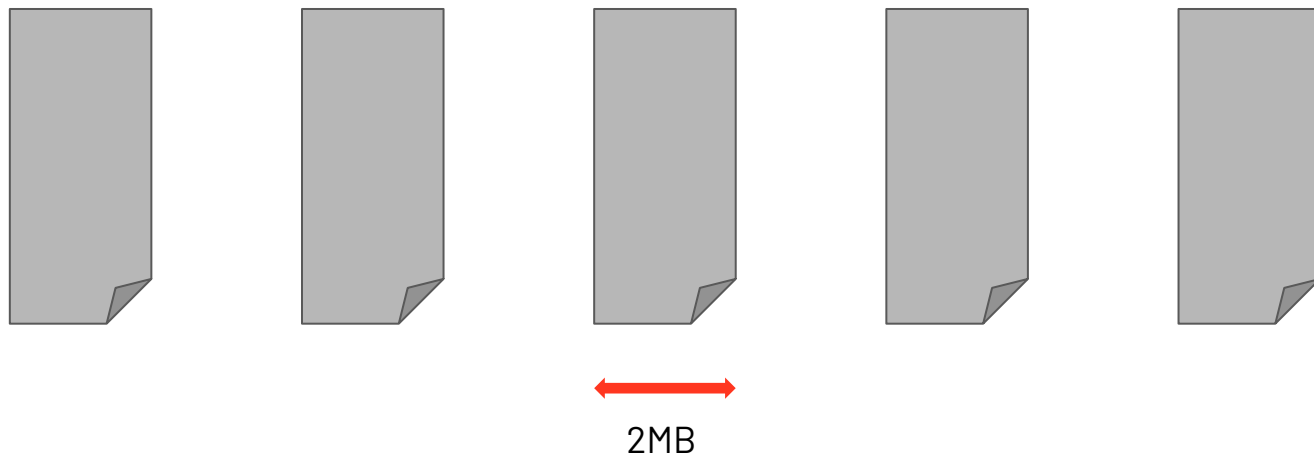
Data layouts team – data organization for the best read/write performance

Query execution Spark + Parquet



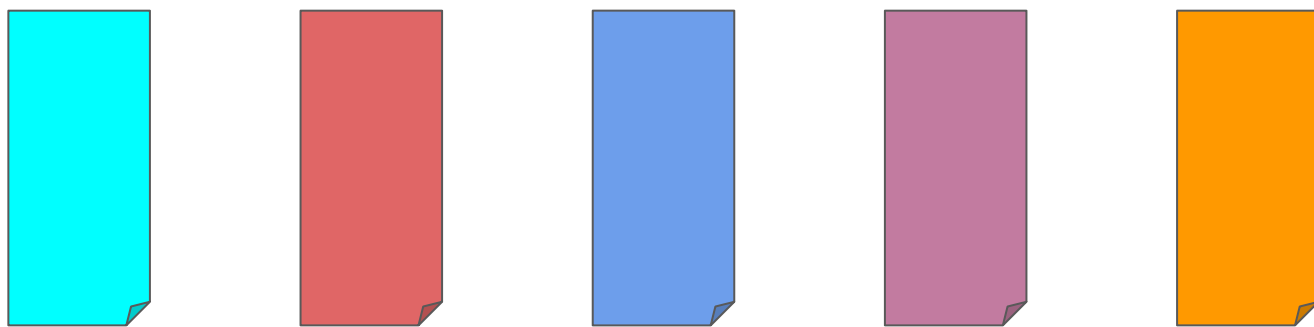
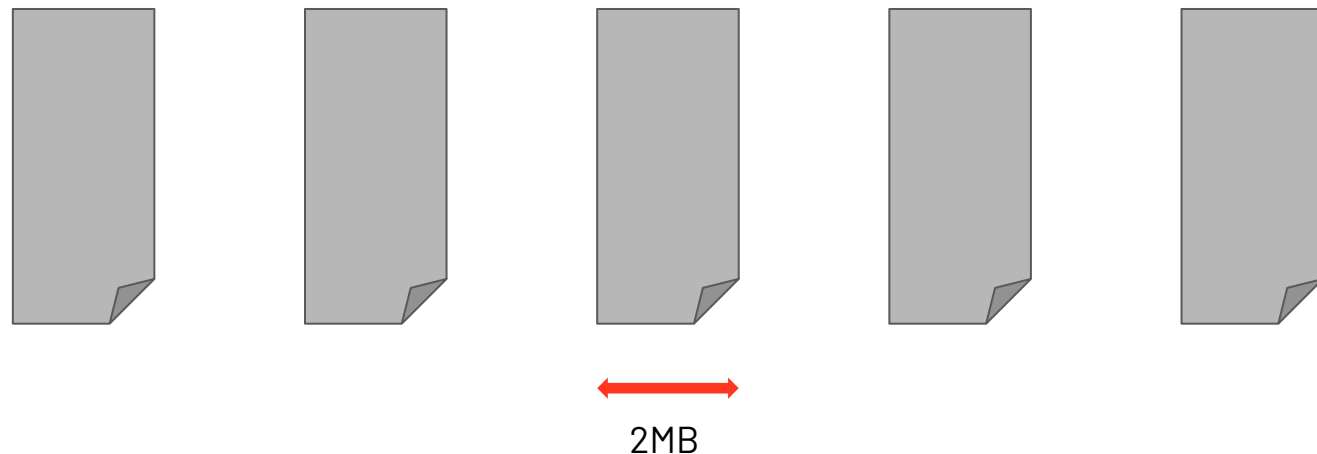
Data layouts

FILE SIZE - I/O overhead



Data layouts

FILE SIZE - I/O overhead

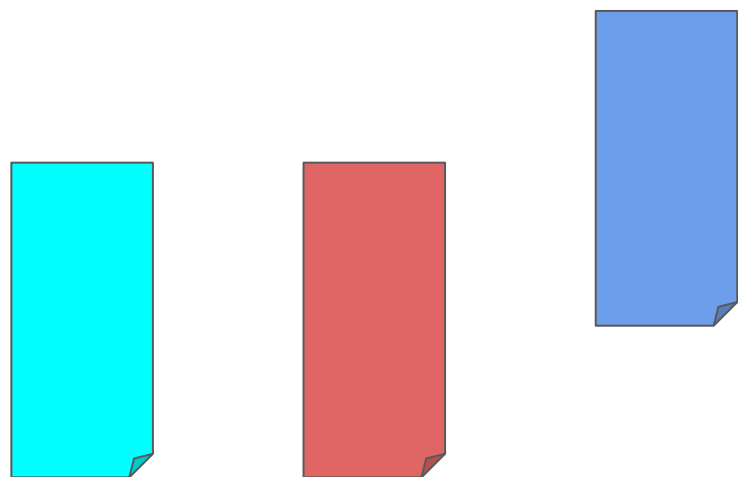


year=2021

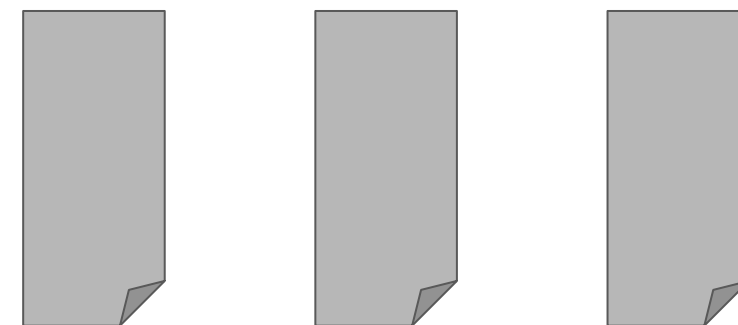
CLUSTERING
Data skipping

Data layouts

FILE SIZE - I/O overhead



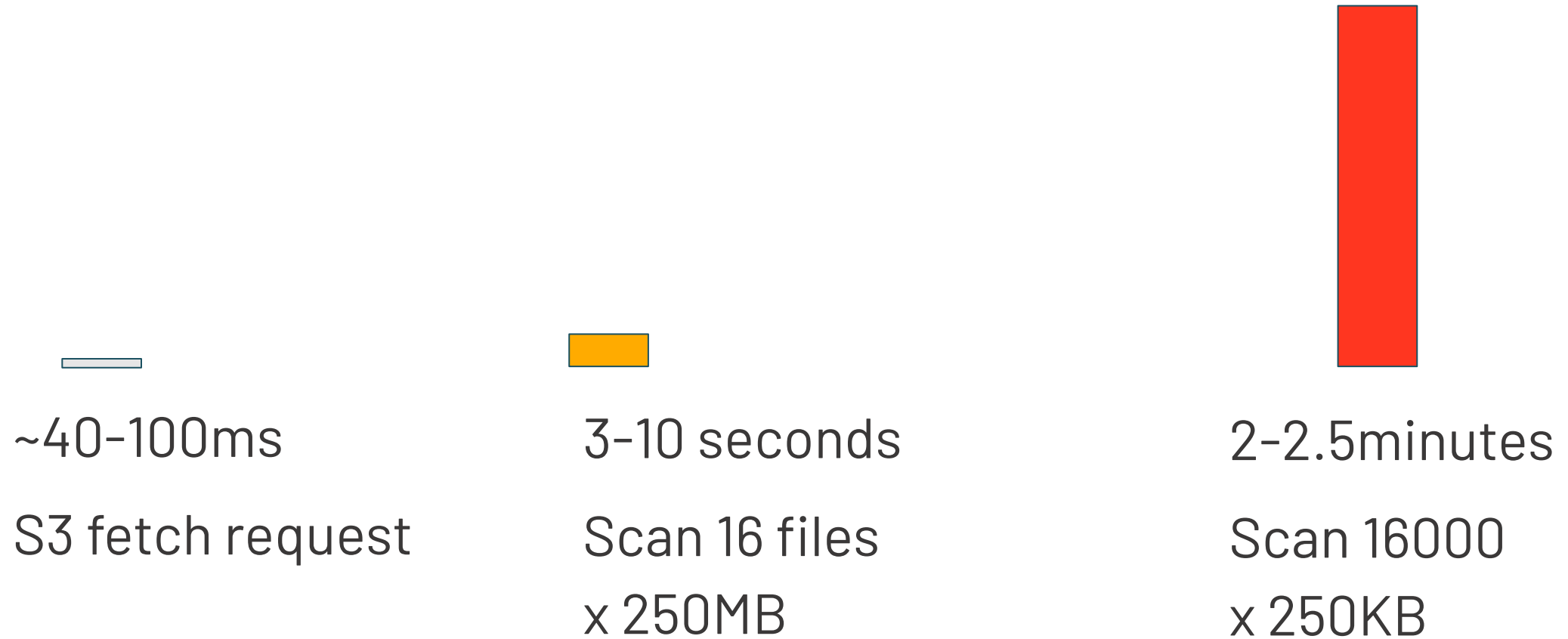
year=2021



2MB

CLUSTERING
Data skipping

Per-file overhead



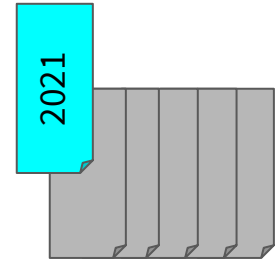
This talk

- Parquet tables, data skipping
- Delta lake
- Advisor/Auto optimizations
- Star schema optimizations

Parquet tables

Data filtering

- Partitions
- Buckets
- Files
- File “chunks”



	Col A	Col B	Col C
Row 0	A0	B0	C0
Row 1	A1	B1	C1
Row 2	A2	B2	C2
Row 3	A3	B3	C3
Row 4	A4	B4	C4
Row 5	A5	B5	C5

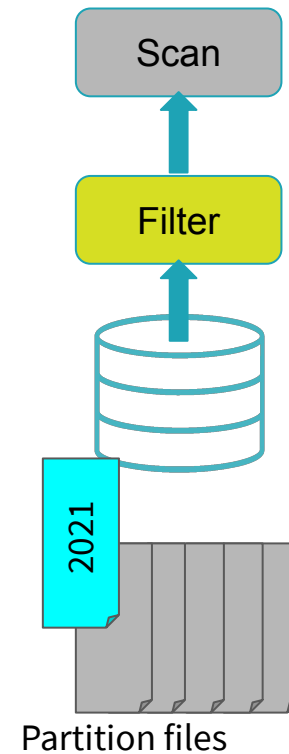
Partitioning



```
SELECT * FROM table WHERE year = 2021
```

Static partition pruning

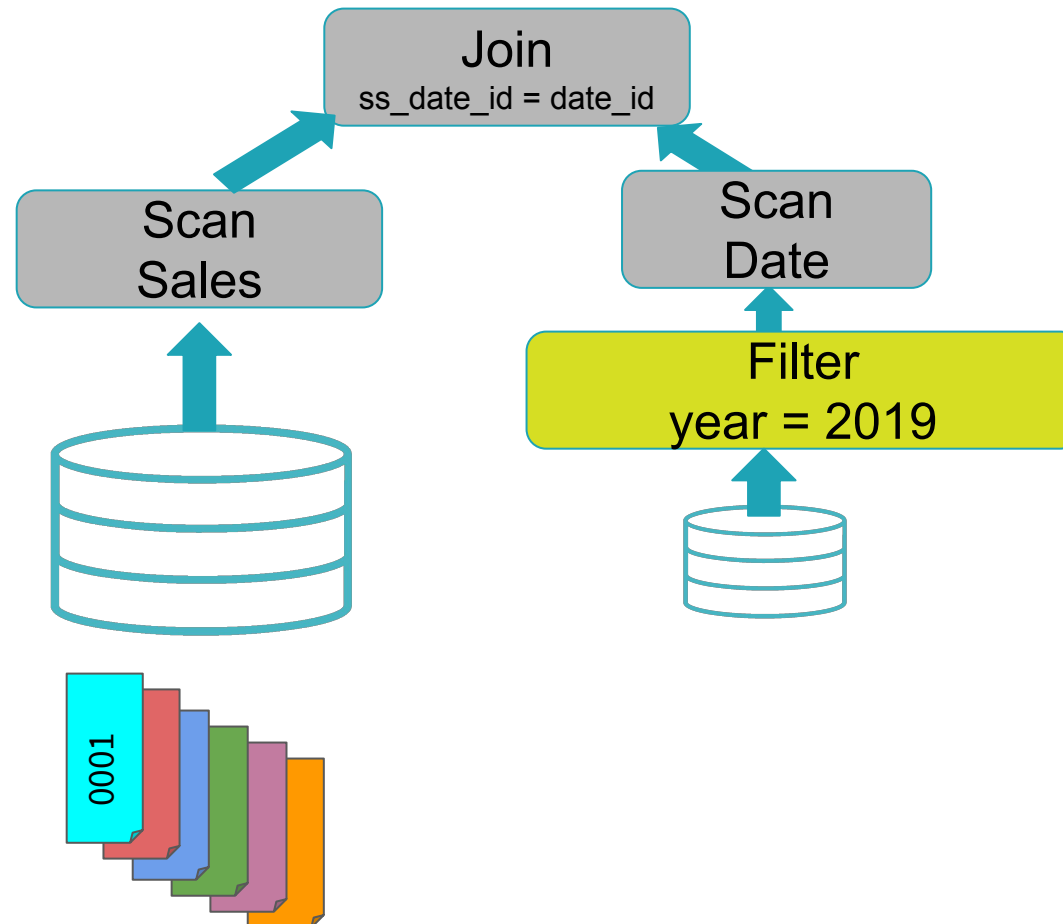
```
SELECT * FROM table WHERE year = 2021
```



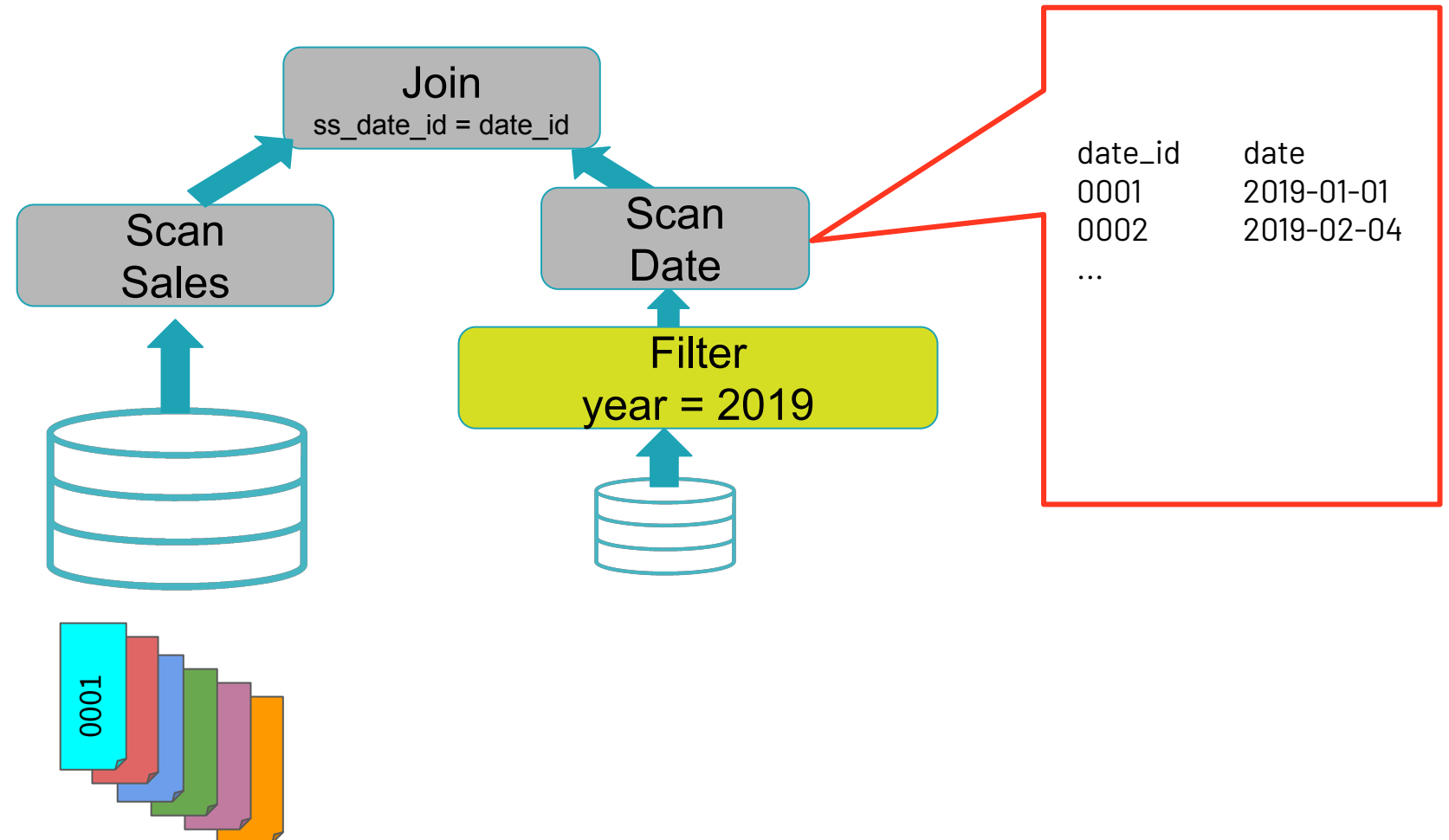
year=2021/
file1.parquet
file2.parquet
year=2020/
file3.parquet
year=2019/
...etc...

Dynamic partition pruning

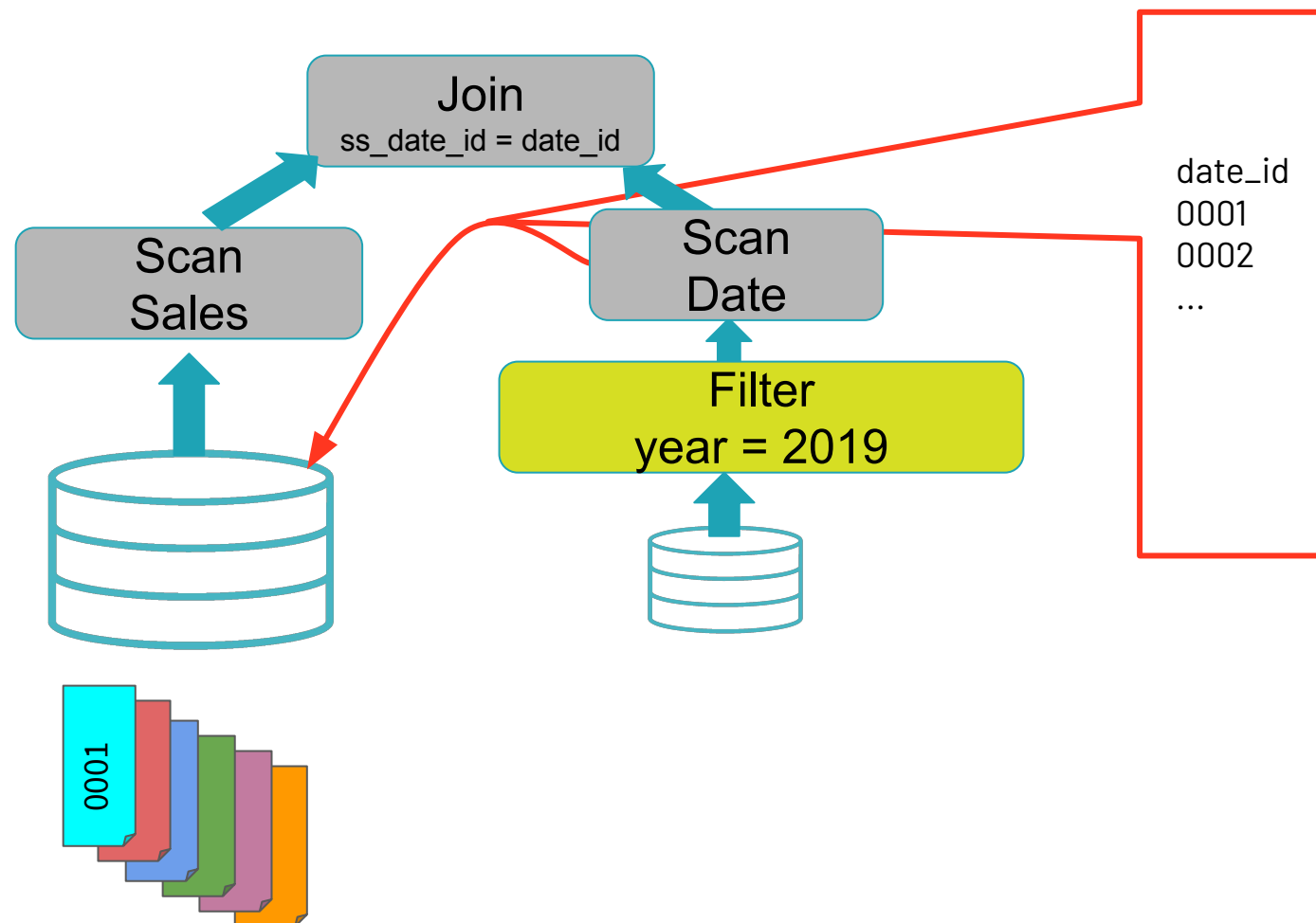
ss_date_id=0001/
file1.parquet
file2.parquet
ss_date_id=0002/
file3.parquet
ss_date_id=0003/
...etc...



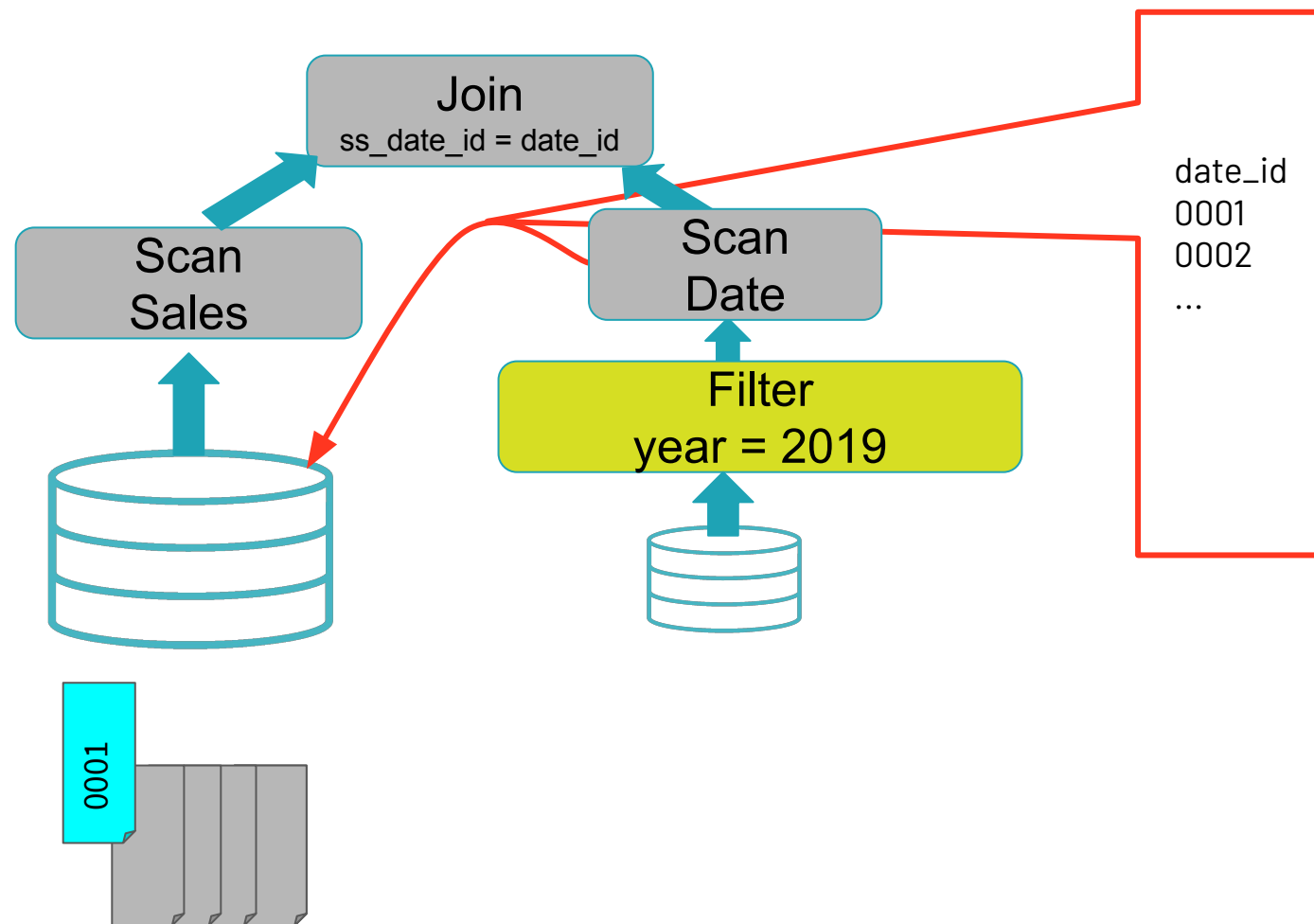
Dynamic partition pruning



Dynamic partition pruning



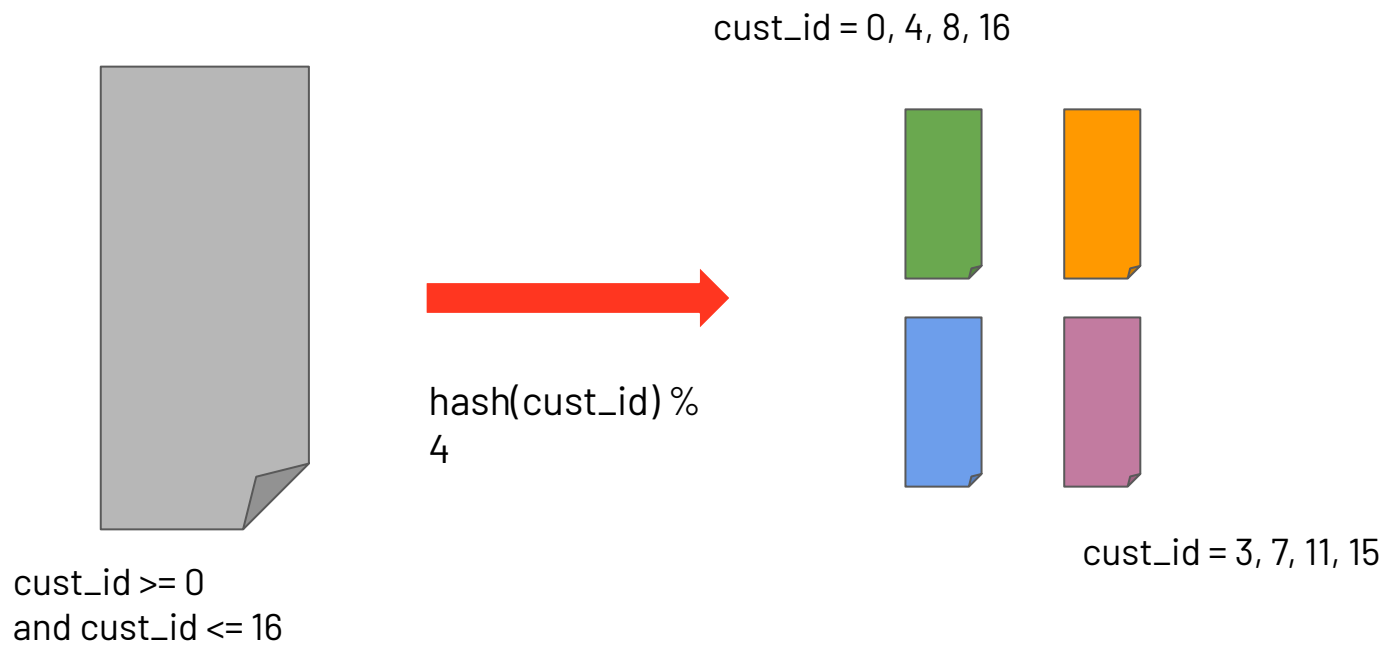
Dynamic partition pruning



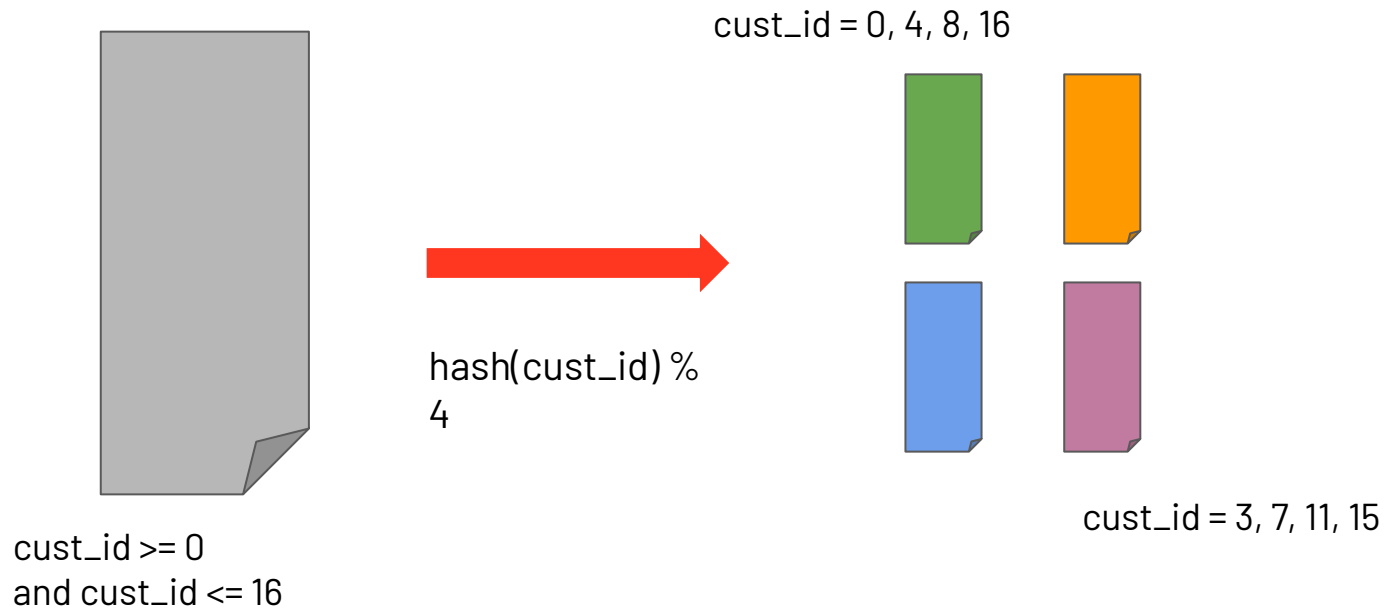
Partitioning disadvantages

- Low cardinality columns only - overpartitioning
- Data skew
- Underpartitioning - large files
- Static - data evolves, know your workloads at table creation

Bucketing

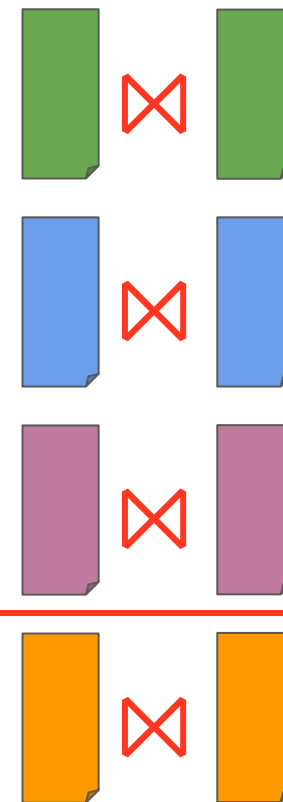
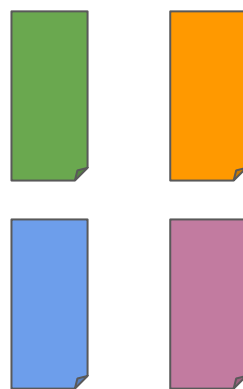
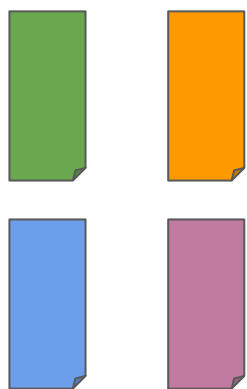


Bucketing



- Good for high cardinality columns - every write produces exactly the same number of files
- Pinpoint queries `cust_id = 1`

Bucketed join



Bucketing disadvantages

- Static, hard to evolve
 - How many buckets?
 - Adding new bucket key requires rewrite
- Data skew
- No range scans

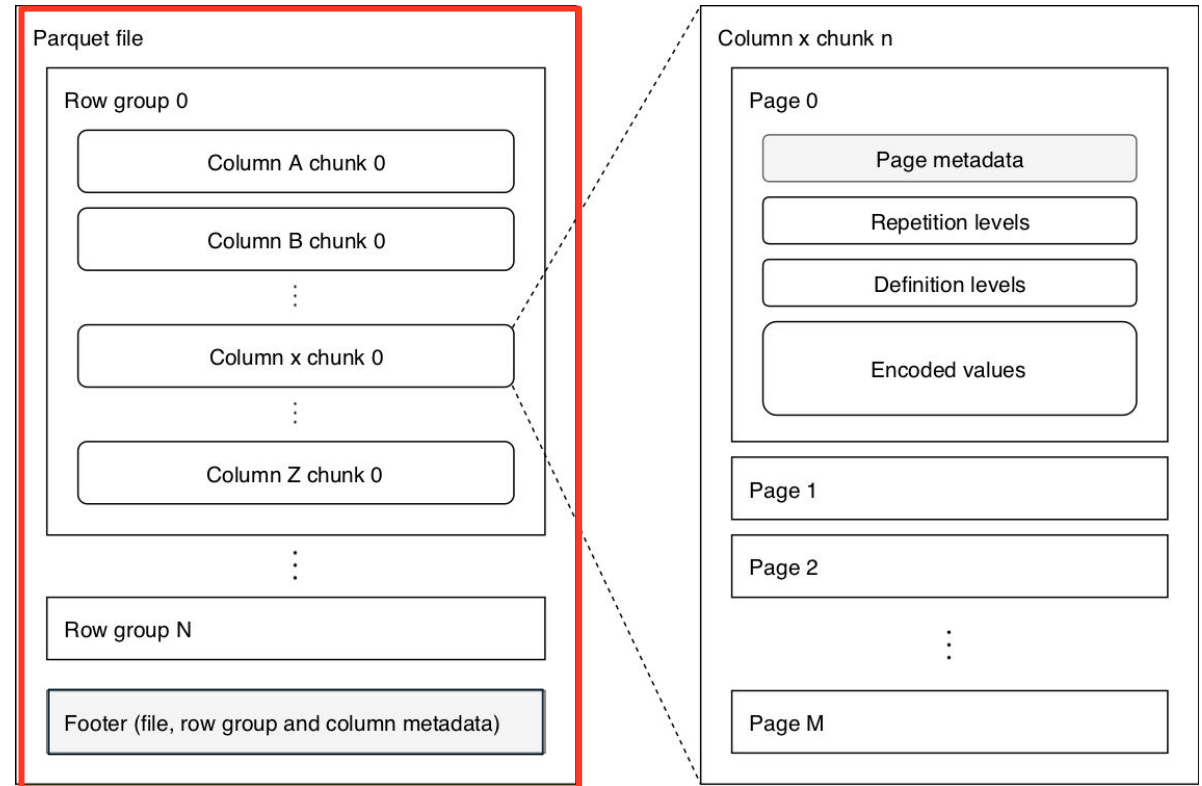
File skipping

- Determine from parquet footer if a file **could contain** values



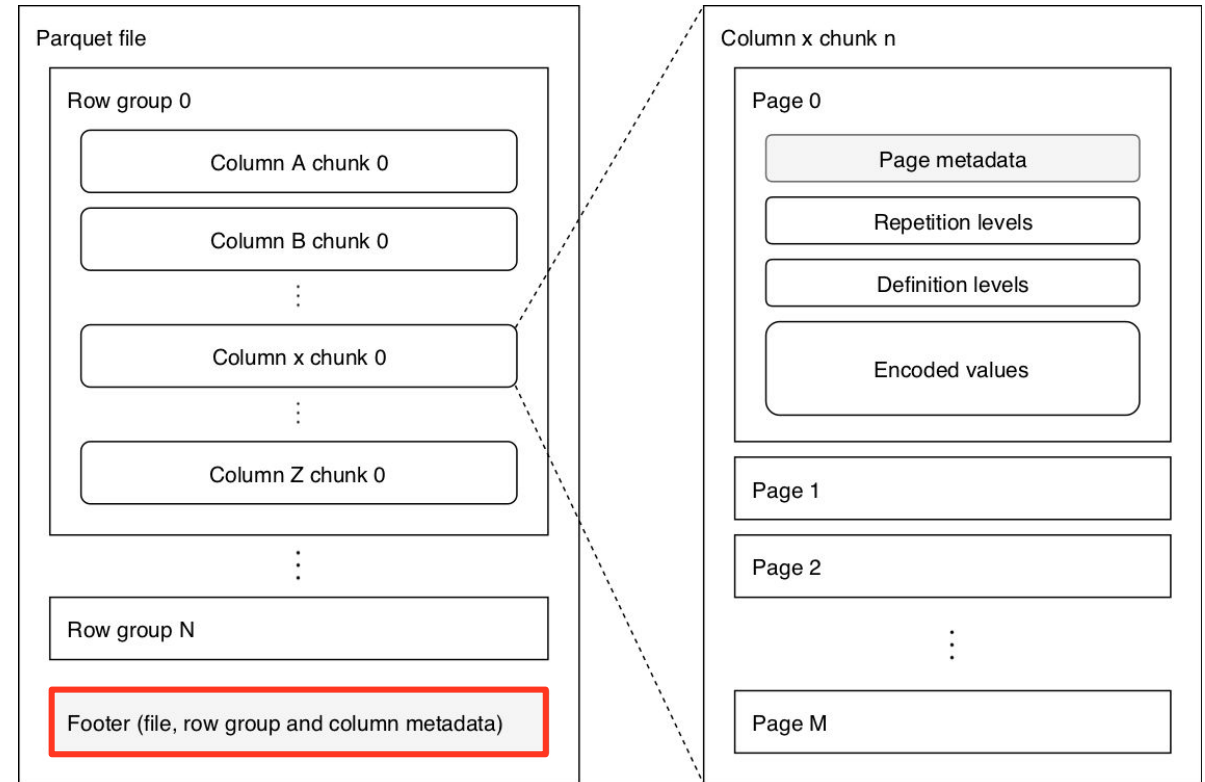
Parquet internals

- File



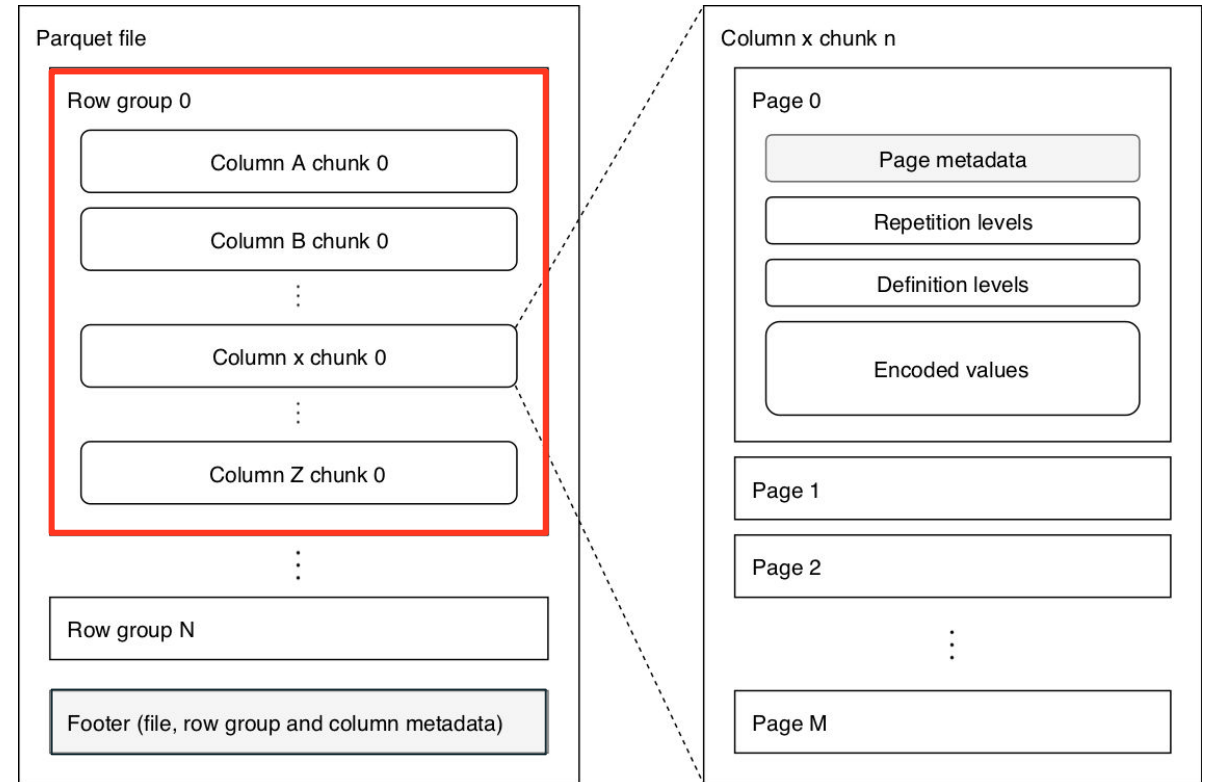
Parquet internals

- File
 - Footer
 - row-group min max stats



Parquet internals

- File
 - Footer
 - row-group min max stats
 - Row group
 - Column chunks
 - Pages - stats



File skipping

- Min-Max row-group statistics
- Determine from footer if a row group **could contain** values

```
SELECT * FROM table WHERE x > 5
```

```
Row-group 0: x: [min: 0, max: 9]
```

```
Row-group 1: x: [min: 3, max: 7]
```

```
Row-group 2: x: [min: 1, max: 4]
```

File skipping

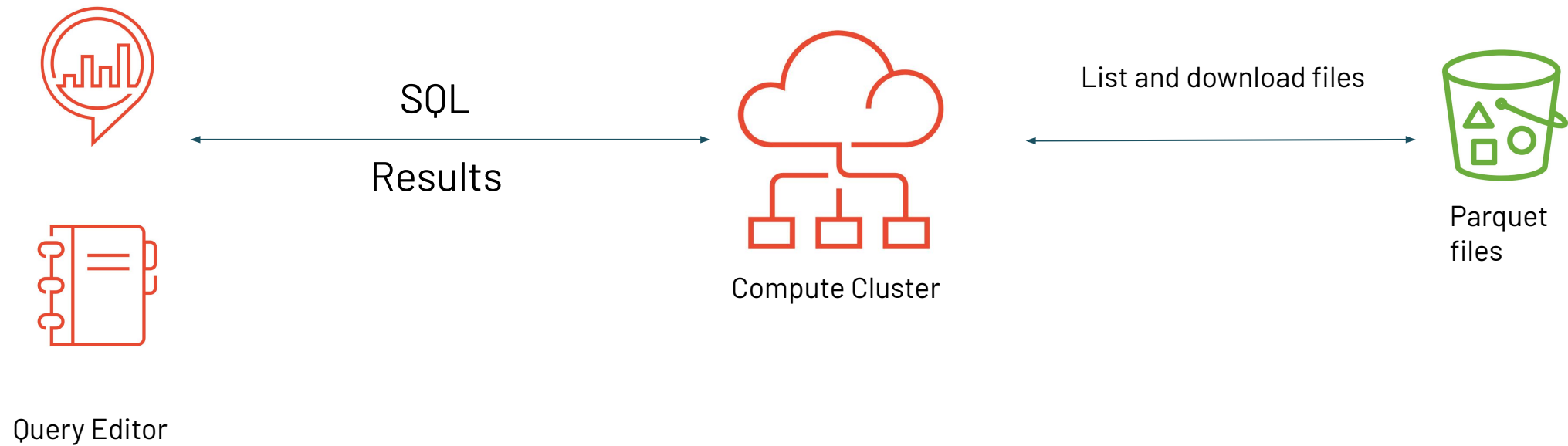
- Min-Max row-group statistics
- Determine from footer if a row group **could contain** values

I have to read a file in order to skip it?!

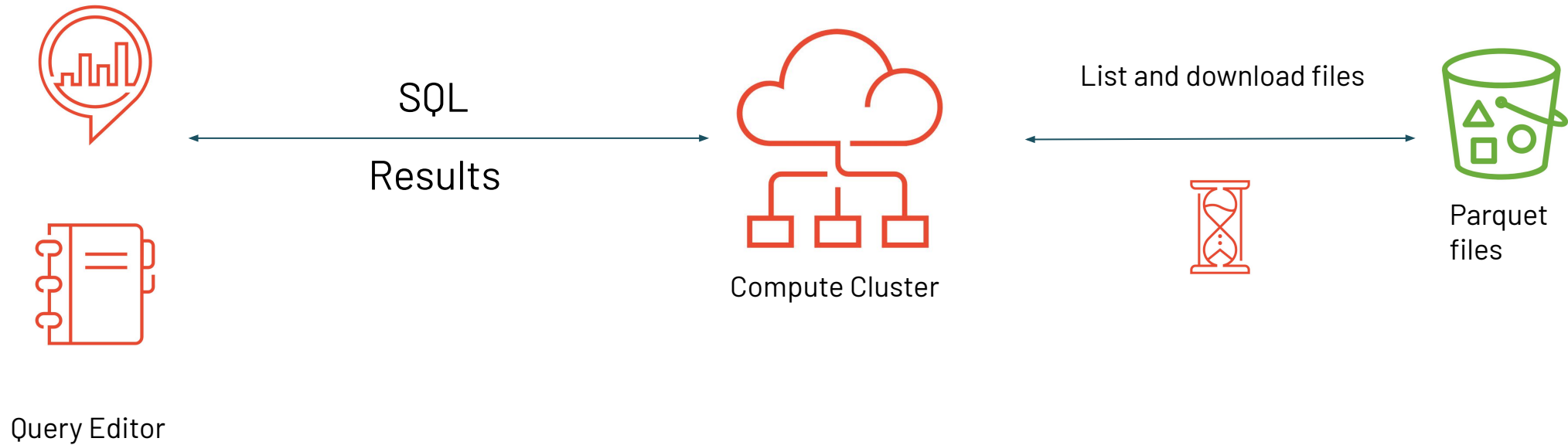
Need to schedule Spark tasks to read files.

Delta Tables

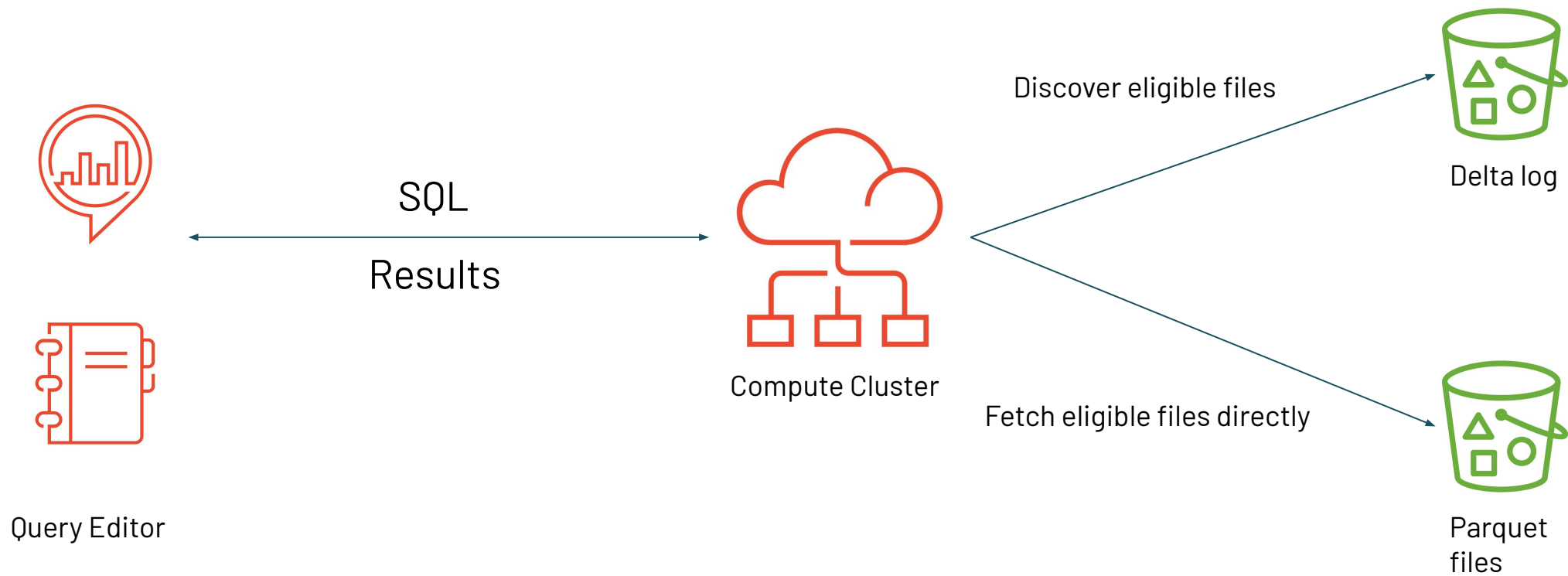
File listing is slow



File listing is slow



Delta Log



Delta Table



Transaction log

Partition directories(optional)

Data files

- my_table/
 - _delta_log/
 - 000000.json
 - 000001.json
 -
 - 000010.checkpoint.parquet
 - year=2021
 - file-01.parquet
 - file-02.parquet
 - ...
 - year=2020
 - file-00.parquet
 - ...

Delta log functions

- ACID transactions
- History - time travel
- **Files present**
- **File statistics**

Inspecting delta log - added files

```
import com.databricks.sql.transaction.tahoe.DeltaLog
import org.apache.spark.sql.catalyst.TableIdentifier

val db = "tpcds_sf100_delta"
val table = "store_sales"
val deltaLog = DeltaLog.forTable(spark, TableIdentifier(table, Some(db)))
deltaLog.snapshot.allFiles.show(3)
```

path	partitionValues	size	modificationTime	dataChange	stats	tags
ss sold date sk=2...	{ss sold date sk ...	4234199	1610537322000	false	{"numRecords":880...	null
ss sold date sk=2...	{ss sold date sk ...	4162372	1610537322000	false	{"numRecords":866...	null
ss sold date sk=2...	{ss sold date sk ...	4157255	1610537323000	false	{"numRecords":865...	null

Inspecting delta log - added files

```
import com.databricks.sql.transaction.tahoe.DeltaLog
import org.apache.spark.sql.catalyst.TableIdentifier

val db = "tpcds_sf100_delta"
val table = "store_sales"
val deltaLog = DeltaLog.forTable(spark, TableIdentifier(table, Some(db)))
deltaLog.snapshot.allFiles.show(3)
```

path	partitionValues	size	modificationTime	dataChange	stats	tags
ss sold date sk=2...	{ss sold date sk ...	4234199	1610537322000	false	"numRecords":880...	null
ss sold date sk=2...	{ss sold date sk ...	4162372	1610537322000	false	{"numRecords":866...	null
ss sold date sk=2...	{ss sold date sk ...	4157255	1610537323000	false	{"numRecords":865...	null

Added file stats

```
{  
  "numRecords": 88099,  
  "minValues": {  
  
    ...  
    "ss_net_profit": -9592  
  },  
  "maxValues": {  
  
    ...  
    "ss_net_profit": 8301  
  },  
  ...  
}
```

Added file stats



```
{  
  "numRecords": 88099,  
  "minValues": {  
  
    ...  
    "ss_net_profit": -9592  
  },  
  "maxValues": {  
  
    ...  
    "ss_net_profit": 8301  
  },  
  ...  
}
```

```
SELECT * FROM store_sales WHERE ss_net_profit < 0
```

Improving data layout for skipping

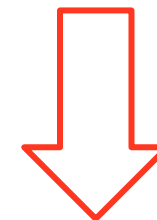
File size

- Files too small -> more IO, delta log processing slower
- Files too large -> cannot skip file, large UPDATE overhead
- 32MB - 128MB and 1GB for large tables > 10TB

Compaction

```
path = "..."  
numFiles = 16
```

```
(spark.read  
  .format("delta")  
  .load(path)  
  .repartition(numFiles)  
  .write  
  .option("dataChange", "false")  
  .format("delta")  
  .mode("overwrite")  
  .save(path))
```

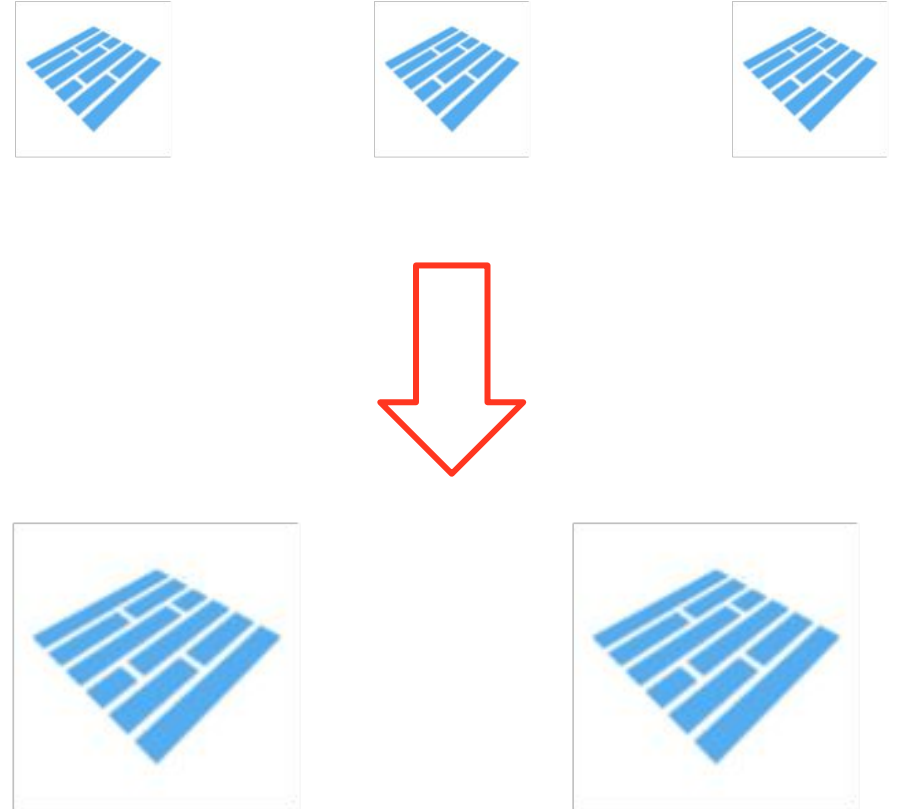


Compaction

```
path = "..."  
numFiles = 16
```

```
(spark.read  
  .format("delta")  
  .load(path)  
  .repartition(numFiles)  
  .write  
  .option("dataChange", "false")  
  .format("delta")  
  .mode("overwrite")  
  .save(path))
```

- Number of target files?

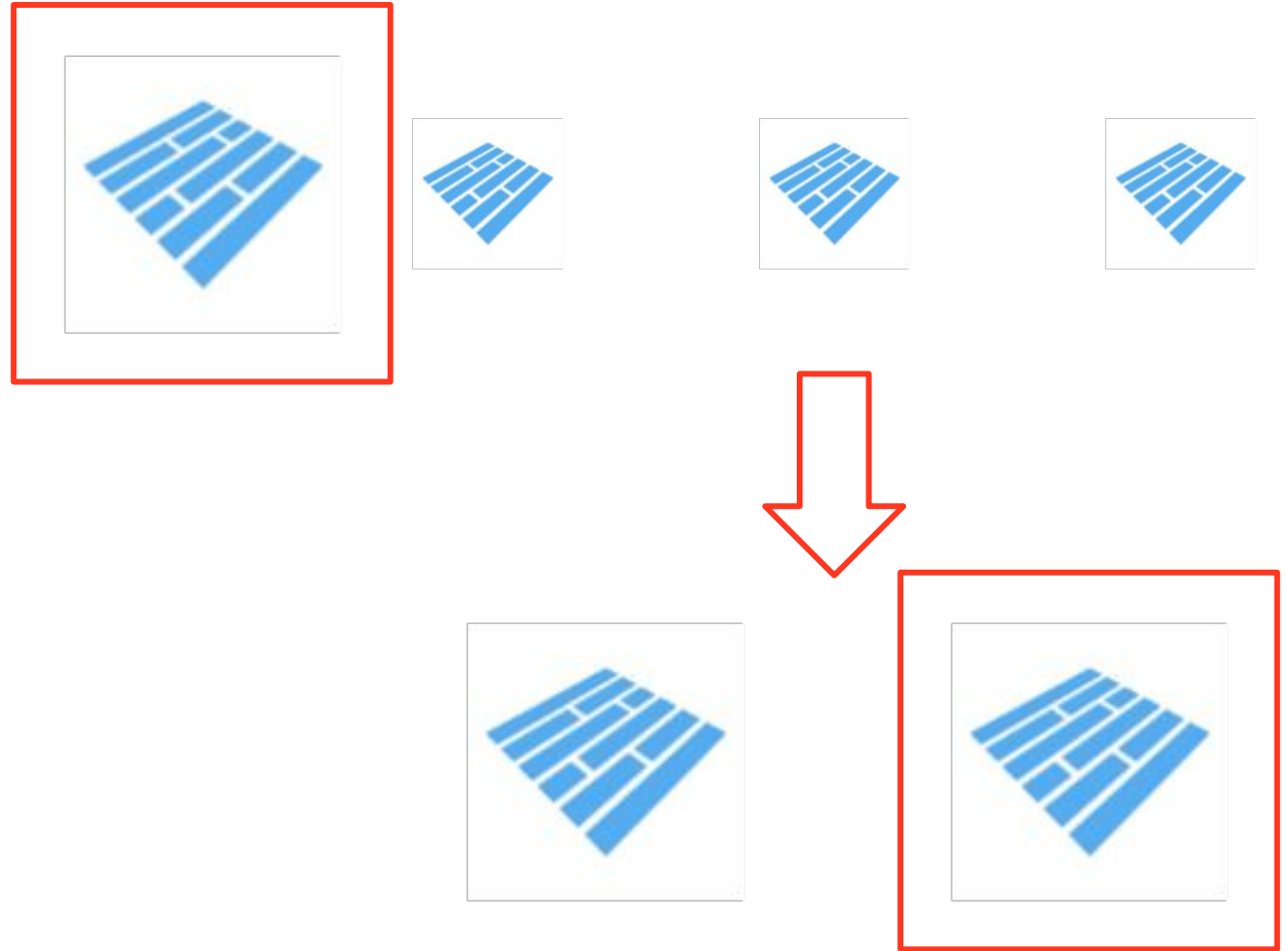


Compaction

```
path = "..."  
numFiles = 16
```

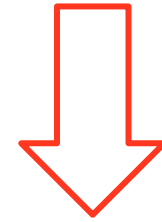
```
(spark.read  
  .format("delta")  
  .load(path)  
  .repartition(numFiles)  
  .write  
  .option("dataChange", "false")  
  .format("delta")  
  .mode("overwrite")  
  .save(path))
```

- Number of files?
- Overwrites the whole table(partition)



Optimize command

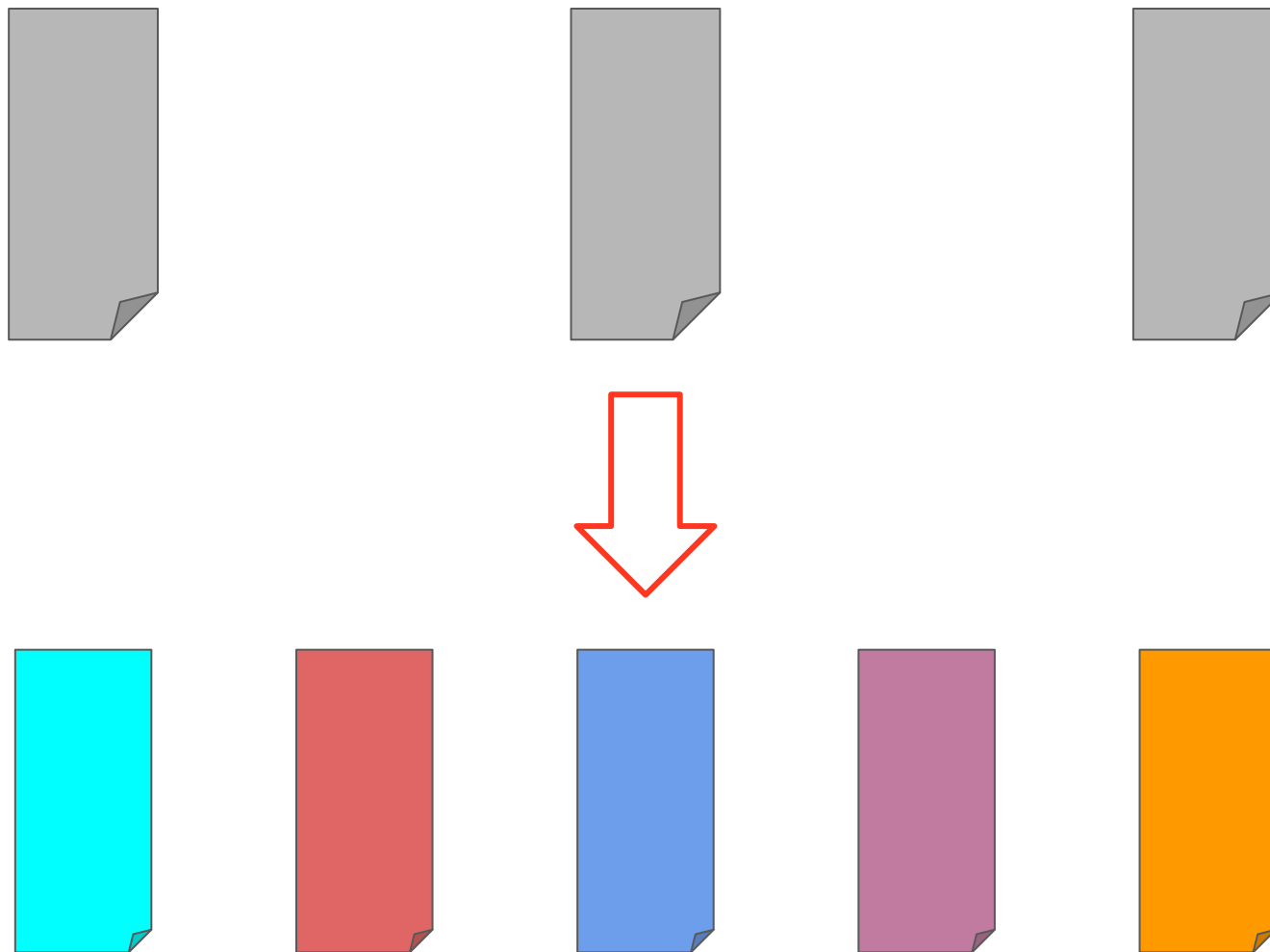
`OPTIMIZE my_table`



- Compacts only files that are too small
- Creates "good" file sizes(64MB-1GB)



Data clustering



Sorting - 1 dimension

X	Y
1	7
1	3
3	4
4	5

000.parquet

$x \geq 1 \ \&\& \ x \leq 4$



X	Y
5	1
6	2
6	3
7	5

001.parquet

$x \geq 5 \ \&\& \ x \leq 7$

```
SELECT * FROM table WHERE x = 5
```

Sorting - 2 dimension

X	Y
1	7
1	3
1	4
1	5

000.parquet

X	Y
1	1
1	2
6	3
7	5

001.parquet

```
SELECT * FROM table WHERE x = 1 AND y = 3
```

Sorting - 2 dimension



X	Y
1	1
1	2
1	3
1	3

000.parquet

X	Y
1	4
1	5
6	5
7	7

001.parquet

```
SELECT * FROM table WHERE x = 1 AND y = 3
```

Sorting - 2 dimension



X	Y
1	1
2	2
2	3
3	5

000.parquet



X	Y
4	4
5	3
6	5
7	7

001.parquet

```
SELECT * FROM table WHERE y = 3
SELECT * FROM table WHERE x = 1 OR y = 3
```


Z-order - Multi dimensional clustering

`OPTIMIZE store_sales ZORDER BY x, y`



X	Y
1	1
2	2
3	3
5	3

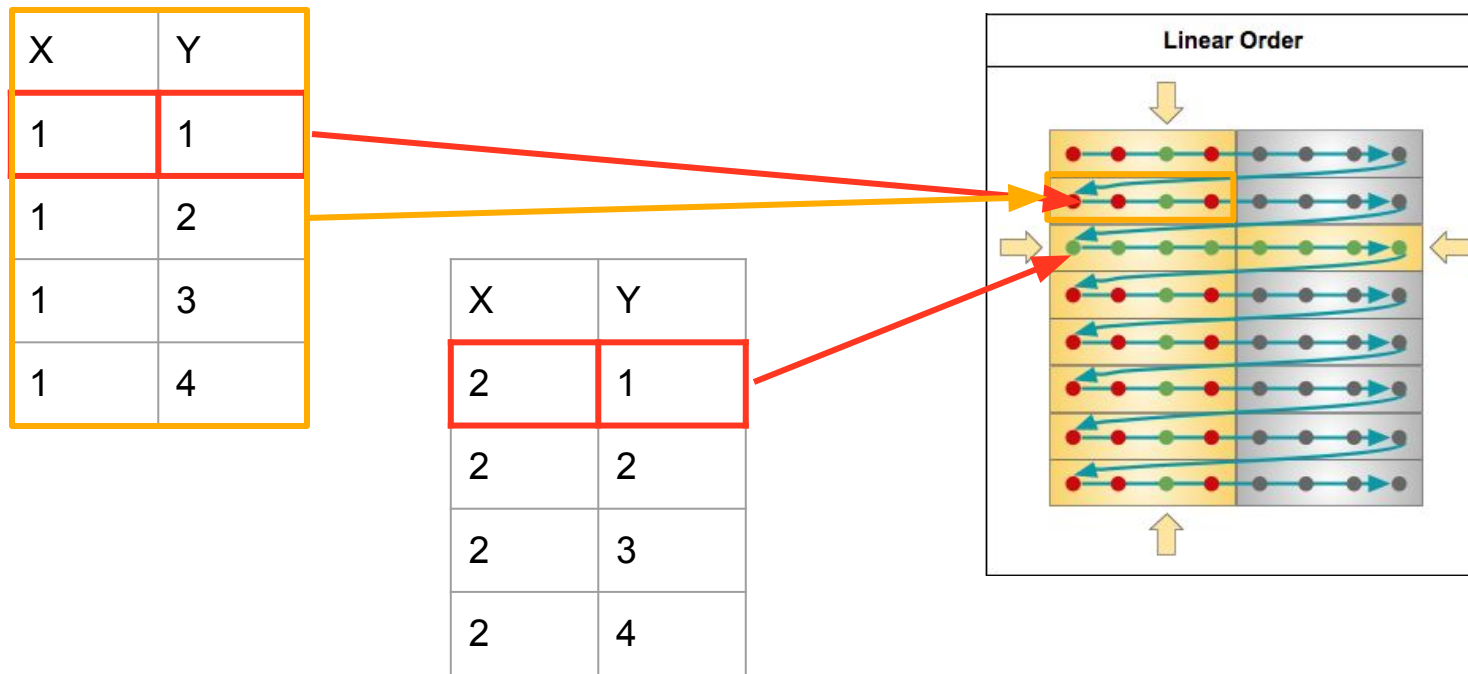
000.parquet

X	Y
3	4
5	4
6	5
7	7

001.parquet

```
SELECT * FROM table WHERE y = 3
SELECT * FROM table WHERE x = 1 OR y = 3
```

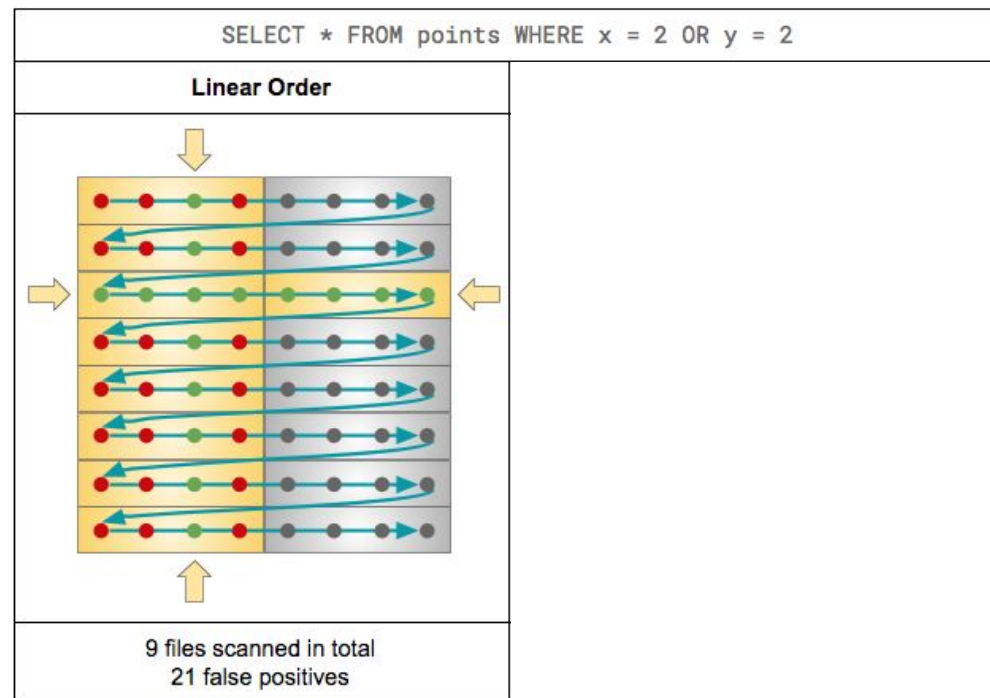
Z-order - Multi dimensional clustering



Z-order - Multi dimensional clustering

X	Y
1	1
1	2
1	3
1	4

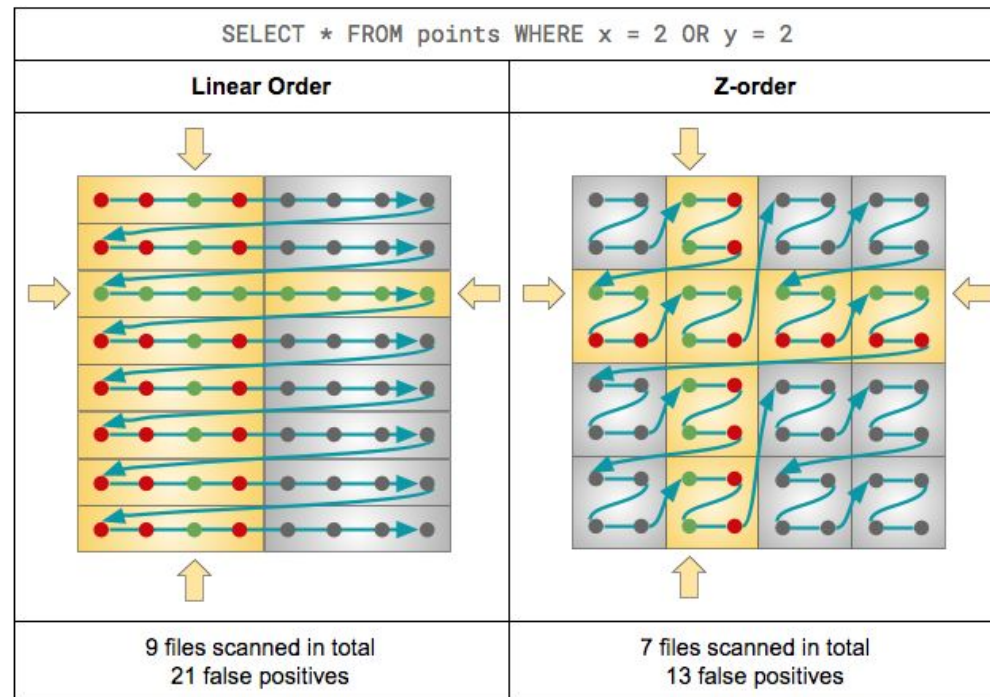
X	Y
2	1
2	2
2	3
2	4



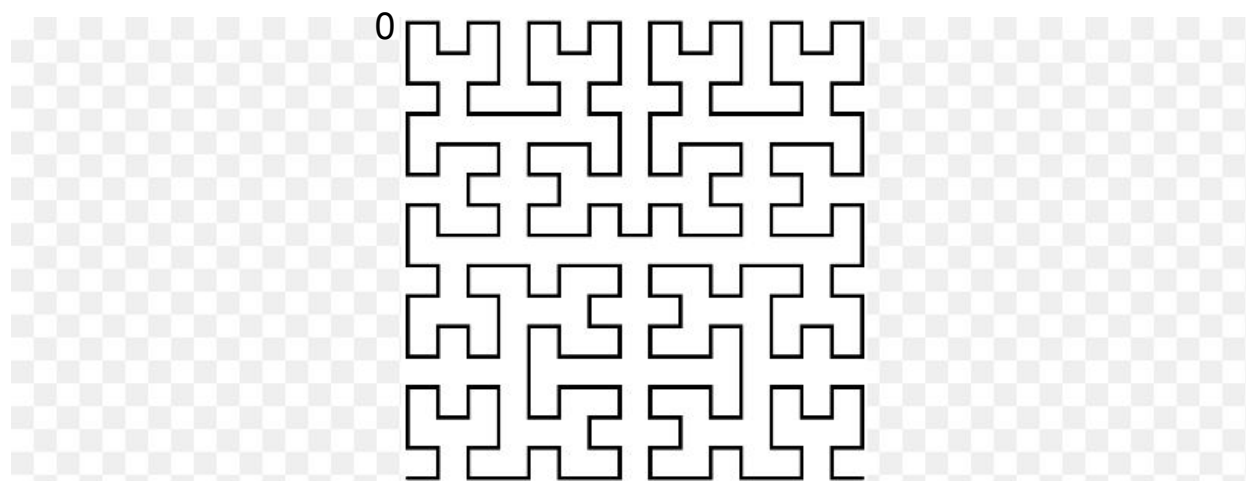
Z-order - Multi dimensional clustering

X	Y
1	1
1	2
2	1
2	2

X	Y
1	1
1	2
2	3
2	4



Hilbert curve



Auxiliary index - Bloom Filter

```
CREATE BLOOMFILTER INDEX  
ON TABLE my_table  
FOR COLUMNS(cust_id OPTIONS (fpp=0.1, numItems=50000000))
```

Is this value in this file?

- ☐ Maybe
- ☐ No

Bloom Filter

0 0 0 0 0

Bloom Filter

1

0	0	0	0	0
0	0	0	1	0

Bloom Filter

1
3

0	0	0	0	0
0	0	0	1	0
0	1	0	1	0

Bloom Filter

1
3
7

0	0	0	0	0
0	0	0	1	0
0	1	0	1	0
0	1	1	1	0

Bloom Filter

1
3
7

0	0	0	0	0
0	0	0	1	0
0	1	0	1	0
0	1	1	1	0

- Is 1 in the bitmap -> **MAYBE**
- Is 0 in the bitmap -> **NO**
- Is 2 in the bitmap -> **MAYBE - False positive**

Data layouts are hard

- Compaction jobs need to be scheduled
- Many knobs to turn
- Workloads change
- Data changes - skew

Steps to automation at Databricks

Auto-compaction

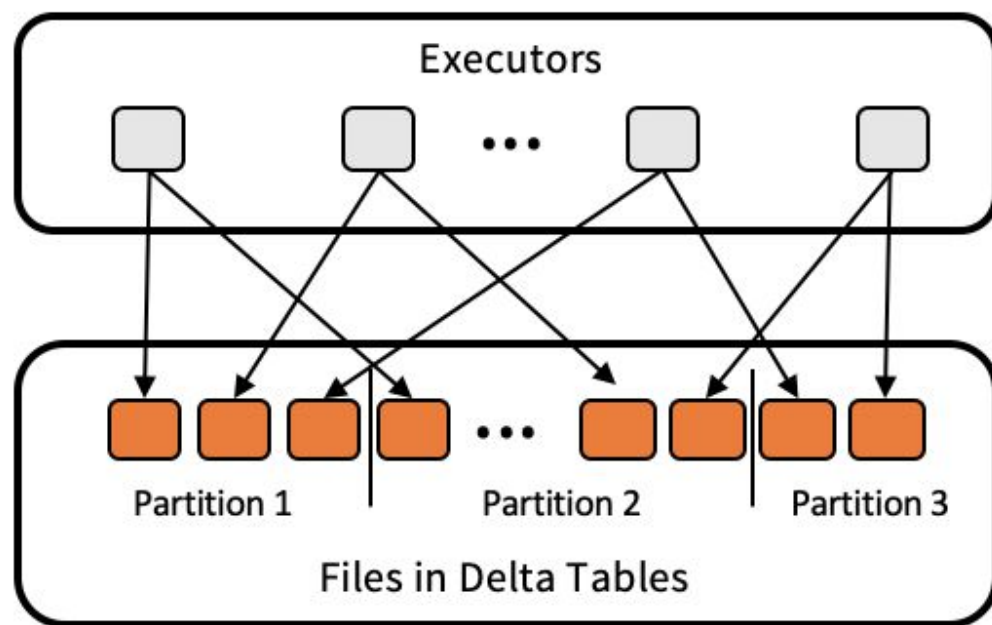
- Compact files after every commit automatically
- Should not run for too long
- Could conflict with other transactions

Optimized Writes

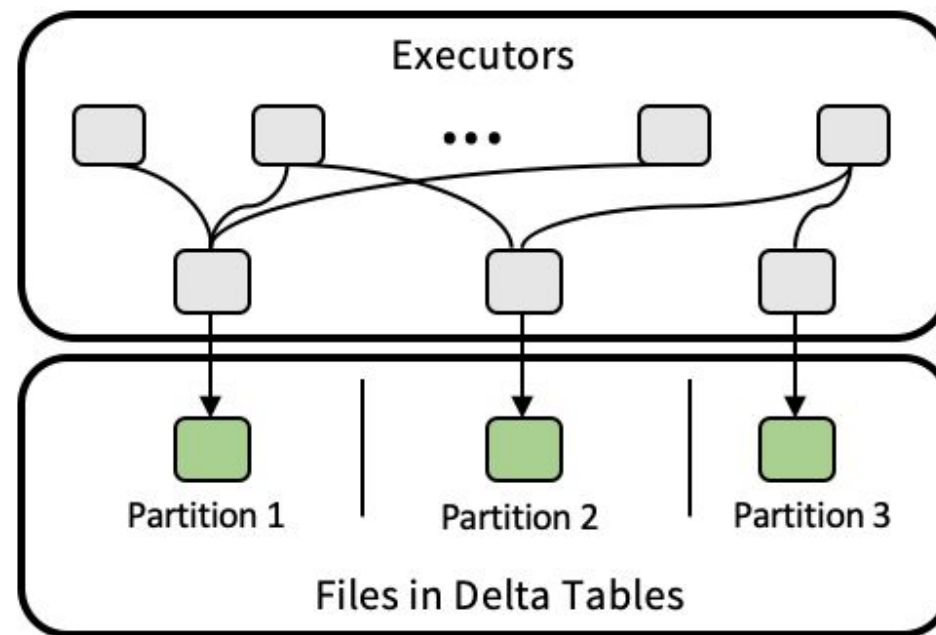
- Shuffled writes for partitioned tables
- Keeps files well sized

Optimized Writes

Traditional Writes



Optimized Writes



Insertion order

- Preserve the original data clustering
- Good for time series data, event logs



min: 2021-01-01
max: 2021-02-01



min: 2021-02-01
max: 2021-03-01



min: 2021-03-01
max: 2021-04-01

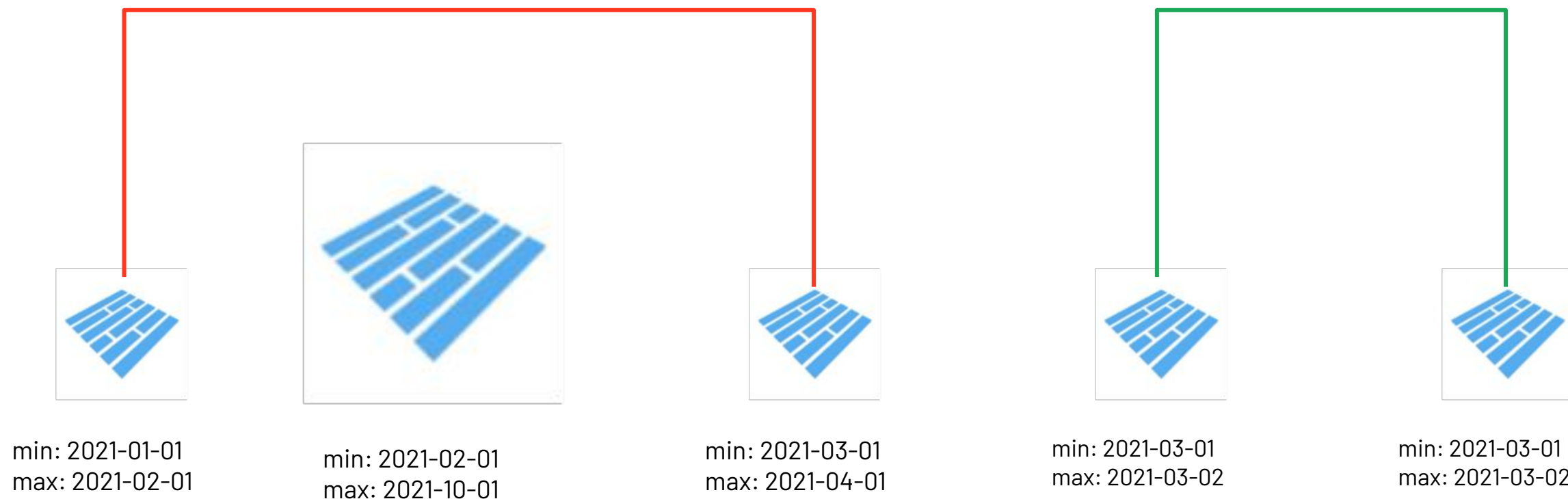


min: 2021-03-01
max: 2021-03-02



min: 2021-03-01
max: 2021-03-02

Compaction challenge



- Compact only neighboring files in timeline

Update/Delete challenge



min: 2021-01-01
max: 2021-02-01



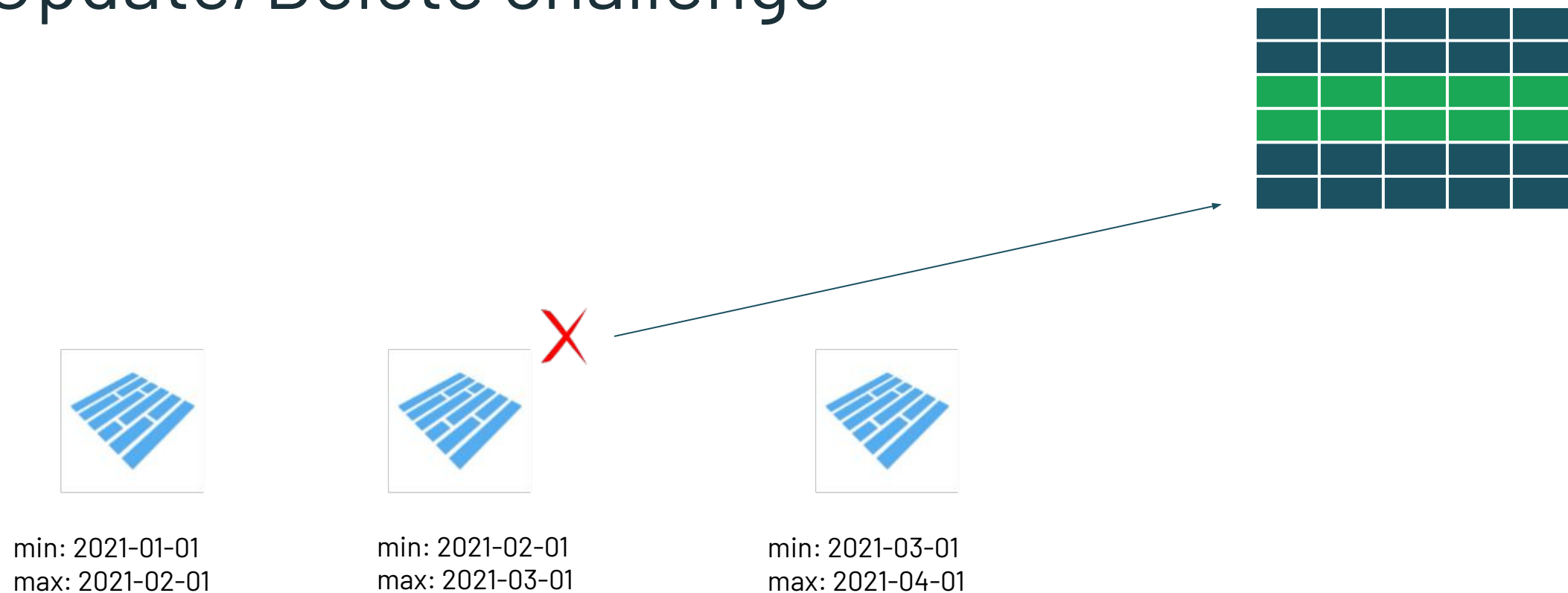
min: 2021-02-01
max: 2021-03-01



min: 2021-03-01
max: 2021-04-01

`DELETE FROM table WHERE date = '2021-02-01'`

Update/Delete challenge



Update/Delete challenge



min: 2021-01-01
max: 2021-02-01



min: 2021-03-01
max: 2021-04-01

Update/Delete challenge



min: 2021-01-01
max: 2021-02-01



min: 2021-02-02
max: 2021-03-01



min: 2021-03-01
max: 2021-04-01

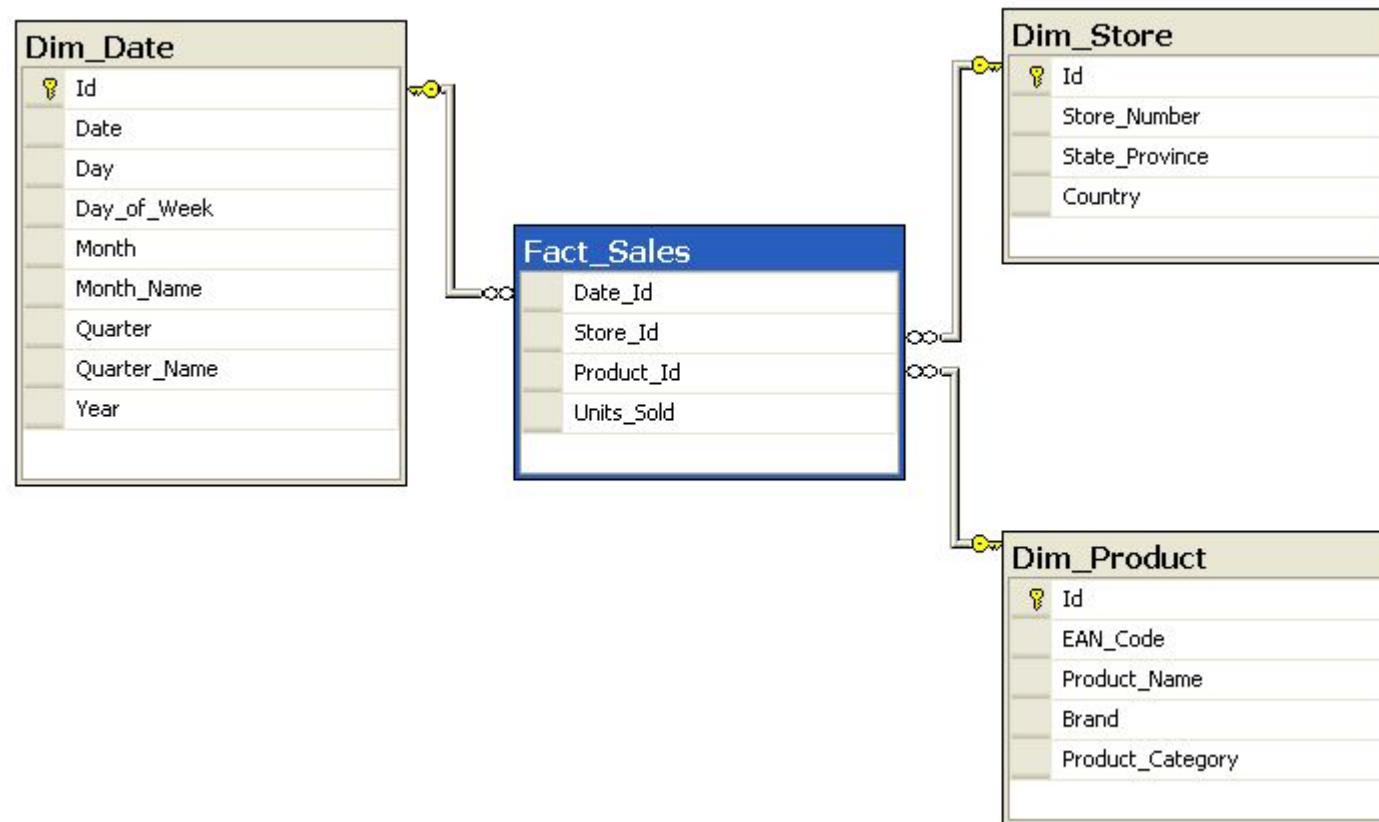


Future: Advisor/background optimizations

- Recognize common query patterns
- Propose optimal data layout
- Run in the background on idle machines

Star schema pattern

Star schema



Star schema

- Consider denormalizing, precomputed joins, storage is cheap
- Use DELTA for your fact and dimension tables
- OS Delta: partition the fact by date and compact files from time to time
- Z-ORDER fact table by “foreign” keys columns
- Z-ORDER dimension tables by “primary” key columns
- Consider creating bloom filter index for pinpoint predicates

Thanks!

We're hiring for internships and full time engineers in Amsterdam, Berlin, San Francisco, Seattle and Toronto!

databricks.com/company/careers

Sabir Akhadov
linkedin.com/in/akhadov