Bucketing in Spark

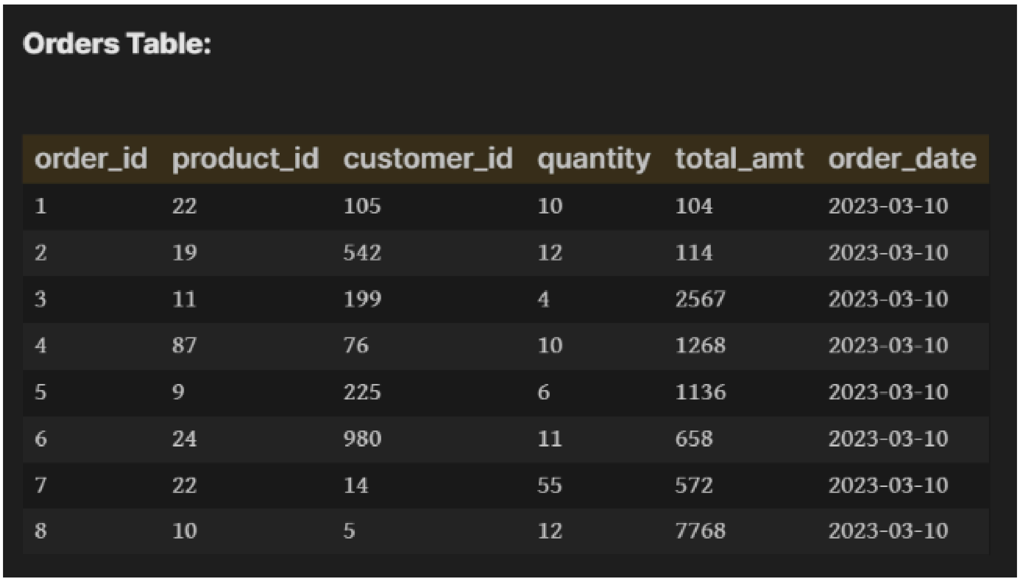
Bucketing

Bucketing is a data organization technique
in Apache Spark that divides datasets into
manageable chunks based on hash values.
This technique significantly improves
performance for operations like:

1. Filtering
2. Aggregations (group by)
3. Joins

```
df.filter(product_id = x)
```

```
     1. as-it-as: No organization
         Pro: simple
         con: full table scan
         use: small datasets only

     2. Partitioning
         Pro: good for low cardinality column
         Con: create small file problem
         use: date-based cols, categories

     3. Bucketing
         Pro: Avoids shuffle: optimal for high cardinality columns
         con: requires upfront planning
         use: high cardinality join/filter cols
```

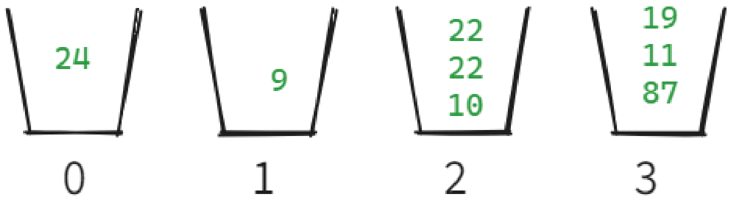


Orders Table:

order_id	product_id	customer_id	quantity	total_amt	order_date
1	22	105	10	104	2023-03-10
2	19	542	12	114	2023-03-10
3	11	199	4	2567	2023-03-10
4	87	76	10	1268	2023-03-10
5	9	225	6	1136	2023-03-10
6	24	980	11	658	2023-03-10
7	22	14	55	572	2023-03-10
8	10	5	12	7768	2023-03-10

bucket assignment =
hash(bucketing_column) % number_of_buckets
h(product_id) % 4

4 Buckets



1. filter

Without Bucketing:
    Query: WHERE product_id = 22
    Problem: Scans ALL records

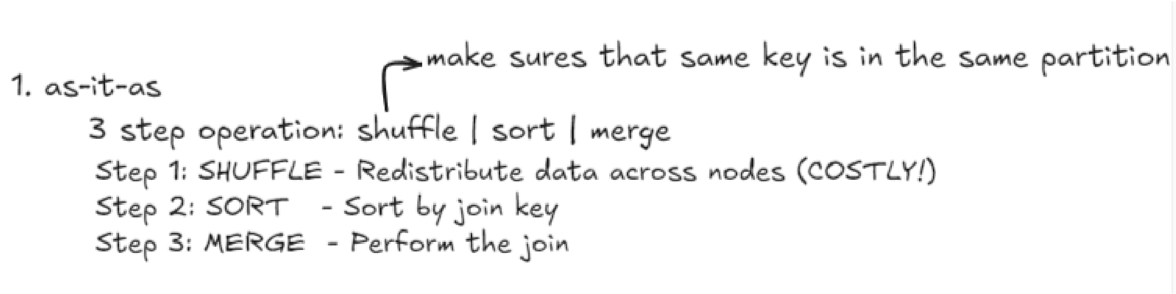
With Bucketing

Query: WHERE product_id = 22
Optimization:
  1. Calculate: 22 % 4 = 2
  2. Read ONLY Bucket 2
  3. Skip Buckets 0, 1, 3

Result: Reduced search space by 75%!

This is called "Bucket Pruning"

2. Joins



1. as-it-as

make sures that same key is in the same partition

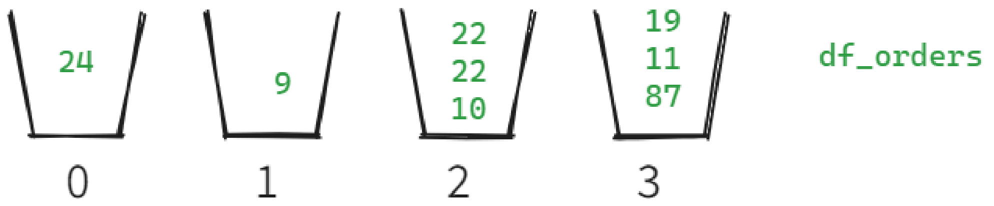
3 step operation: shuffle | sort | merge

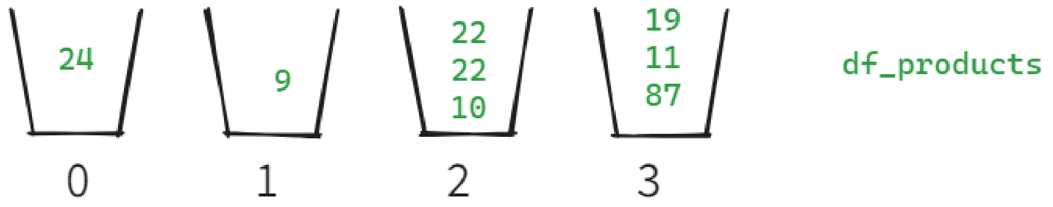
Step 1: SHUFFLE - Redistribute data across nodes (COSTLY!)
Step 2: SORT - Sort by join key
Step 3: MERGE - Perform the join

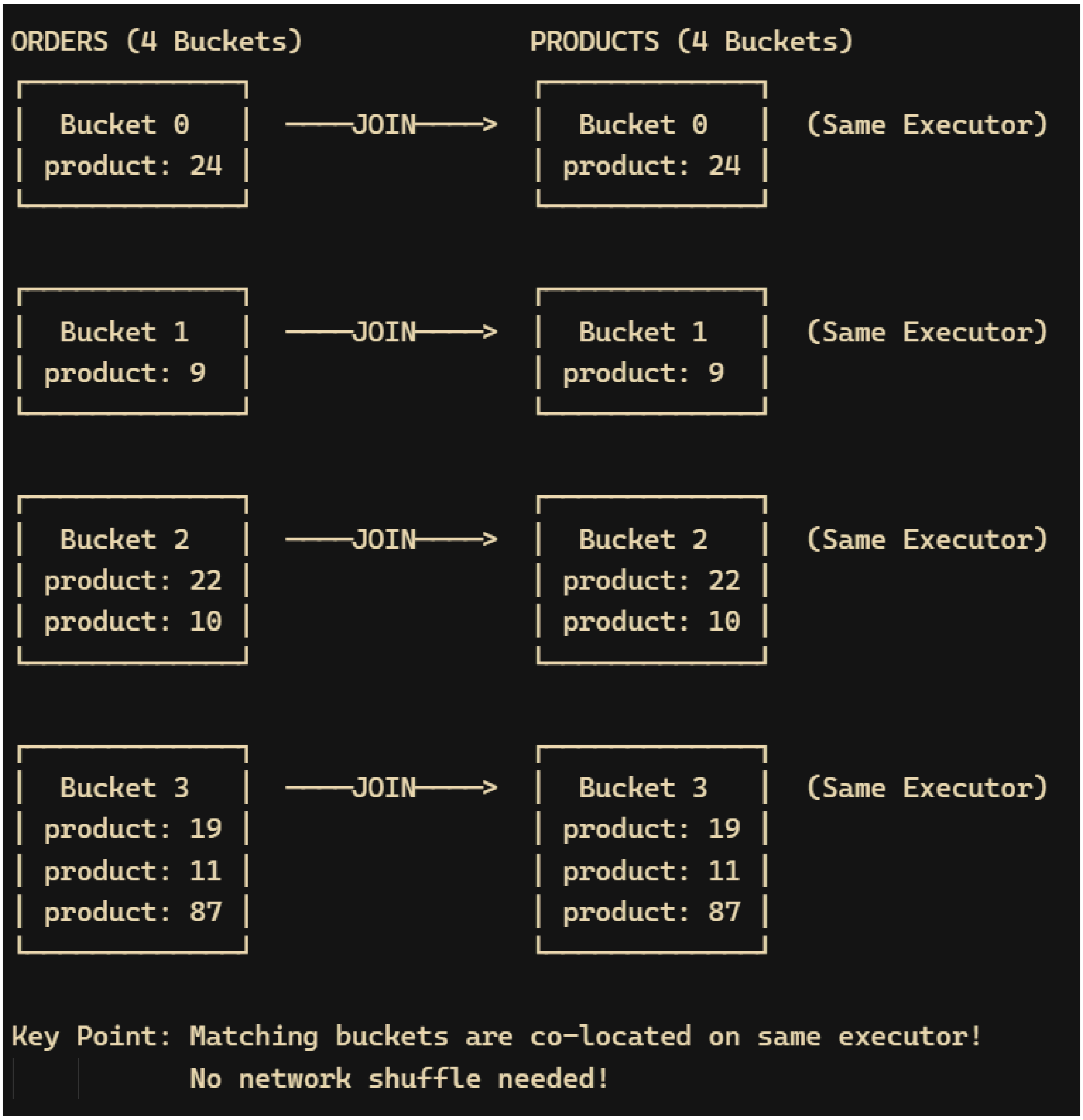
```
# Physical Plan

Exchange/Shuffle (Orders)  ←—— EXPENSIVE
    ↓
Sort (Orders)
    ↓
Exchange/Shuffle (Products) ←—— EXPENSIVE
    ↓
Sort (Products)
    ↓
SortMergeJoin
```

2. Partitioning
   - small file problem
   - product_id is a high cardinality column

3. Bucketing

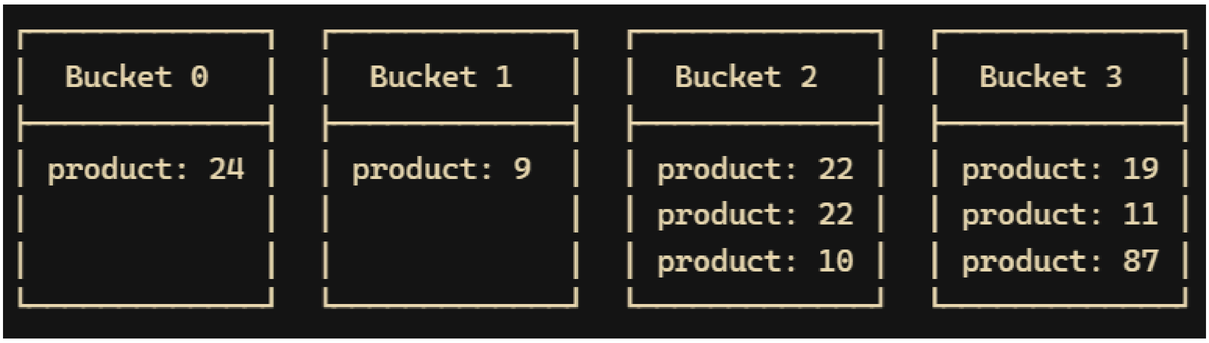




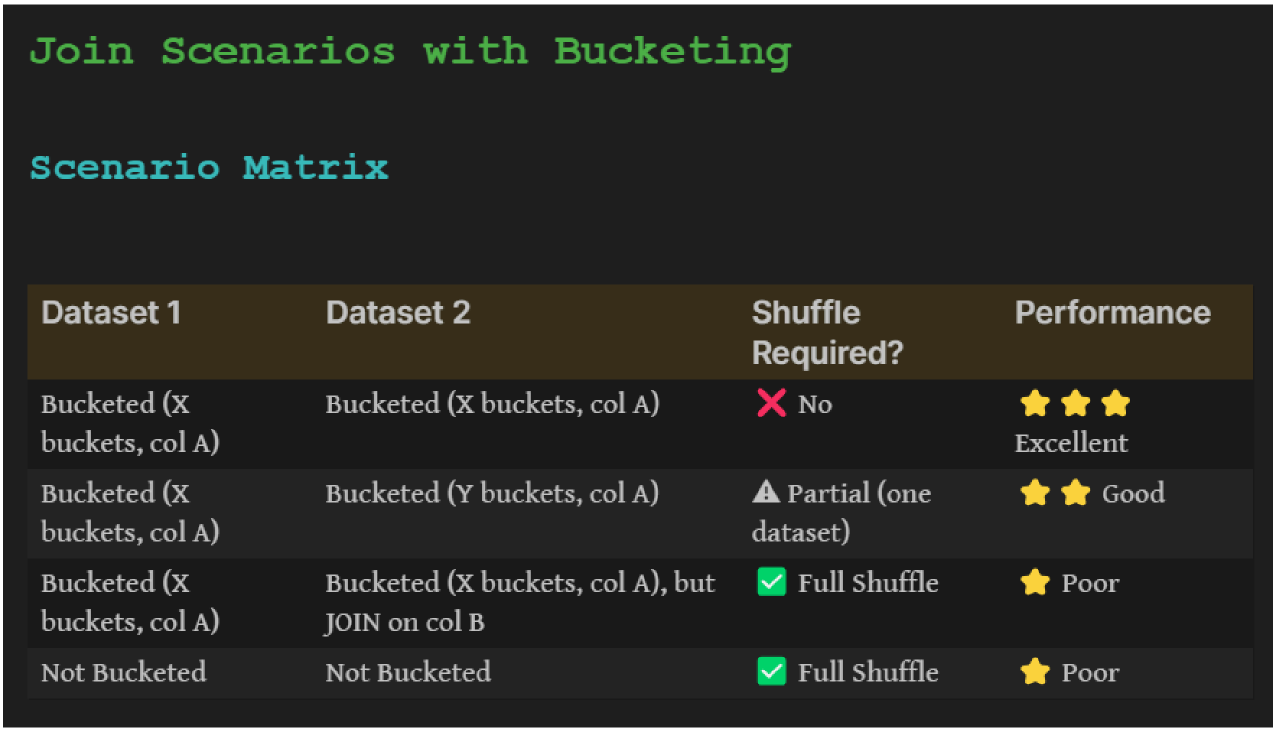


```
ORDERS (4 Buckets)                PRODUCTS (4 Buckets)
┌─────────────┐                  ┌─────────────┐
│  Bucket 0   │  ────JOIN────>   │  Bucket 0   │  (Same Executor)
│ product: 24 │                  │ product: 24 │
└─────────────┘                  └─────────────┘

┌─────────────┐                  ┌─────────────┐
│  Bucket 1   │  ────JOIN────>   │  Bucket 1   │  (Same Executor)
│ product: 9  │                  │ product: 9  │
└─────────────┘                  └─────────────┘

┌─────────────┐                  ┌─────────────┐
│  Bucket 2   │  ────JOIN────>   │  Bucket 2   │  (Same Executor)
│ product: 22 │                  │ product: 22 │
│ product: 10 │                  │ product: 10 │
└─────────────┘                  └─────────────┘

┌─────────────┐                  ┌─────────────┐
│  Bucket 3   │  ────JOIN────>   │  Bucket 3   │  (Same Executor)
│ product: 19 │                  │ product: 19 │
│ product: 11 │                  │ product: 11 │
│ product: 87 │                  │ product: 87 │
└─────────────┘                  └─────────────┘

Key Point: Matching buckets are co-located on same executor!
           No network shuffle needed!
```

Executor 1　　Executor 2　　Executor 3　　Executor 4



```
┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐
│  Bucket 0   │  │  Bucket 1   │  │  Bucket 2   │  │  Bucket 3   │
├─────────────┤  ├─────────────┤  ├─────────────┤  ├─────────────┤
│ product: 24 │  │ product: 9  │  │ product: 22 │  │ product: 19 │
│             │  │             │  │ product: 22 │  │ product: 11 │
│             │  │             │  │ product: 10 │  │ product: 87 │
└─────────────┘  └─────────────┘  └─────────────┘  └─────────────┘
```

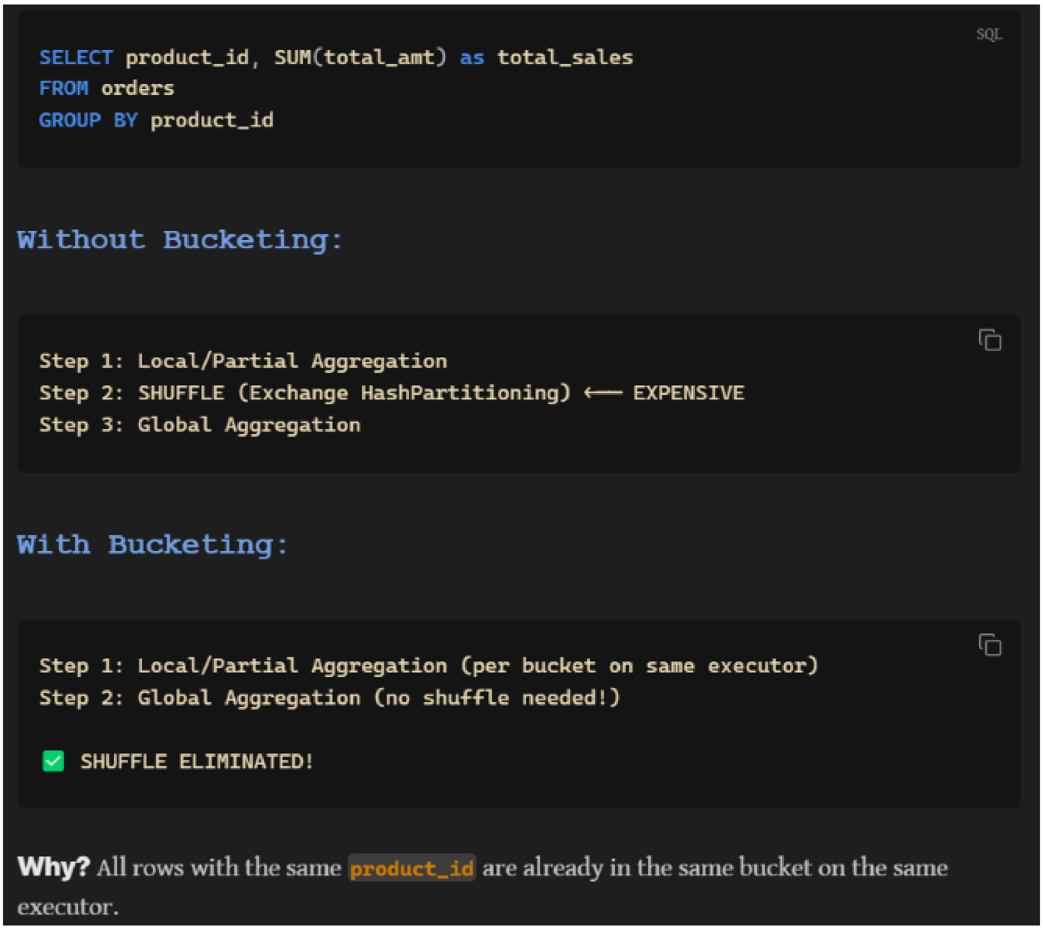


Join Scenarios with Bucketing

Scenario Matrix

Dataset 1	Dataset 2	Shuffle Required?	Performance
Bucketed (X buckets, col A)	Bucketed (X buckets, col A)	❌ No	⭐⭐⭐ Excellent
Bucketed (X buckets, col A)	Bucketed (Y buckets, col A)	⚠ Partial (one dataset)	⭐⭐ Good
Bucketed (X buckets, col A)	Bucketed (X buckets, col A), but JOIN on col B	✅ Full Shuffle	⭐ Poor
Not Bucketed	Not Bucketed	✅ Full Shuffle	⭐ Poor

3. GroupBy



```sql
SELECT product_id, SUM(total_amt) as total_sales
FROM orders
GROUP BY product_id
```

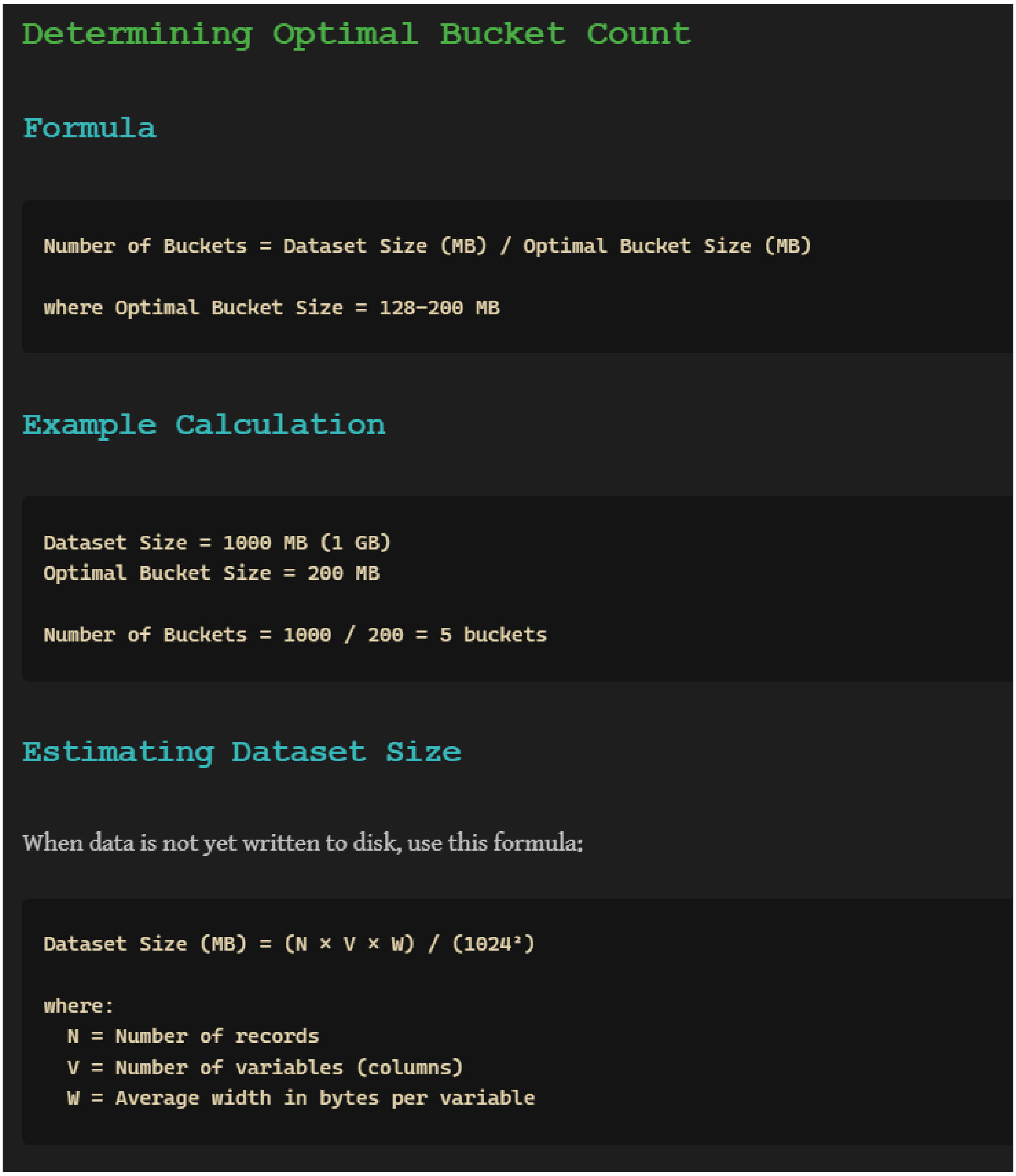
Without Bucketing:

```
Step 1: Local/Partial Aggregation
Step 2: SHUFFLE (Exchange HashPartitioning) ←── EXPENSIVE
Step 3: Global Aggregation
```

With Bucketing:

```
Step 1: Local/Partial Aggregation (per bucket on same executor)
Step 2: Global Aggregation (no shuffle needed!)

✅ SHUFFLE ELIMINATED!
```

Why? All rows with the same `product_id` are already in the same bucket on the same executor.



Determining Optimal Bucket Count

Formula

```
Number of Buckets = Dataset Size (MB) / Optimal Bucket Size (MB)

where Optimal Bucket Size = 128-200 MB
```

Example Calculation

```
Dataset Size = 1000 MB (1 GB)
Optimal Bucket Size = 200 MB

Number of Buckets = 1000 / 200 = 5 buckets
```

Estimating Dataset Size

When data is not yet written to disk, use this formula:

```
Dataset Size (MB) = (N × V × W) / (1024²)

where:
  N = Number of records
  V = Number of variables (columns)
  W = Average width in bytes per variable
```

Suggested Readings:

1. https://umbertogriffo.gitbook.io/apache-spark-best-practices-and-tuning/parallelism/sparksqlshufflepartitions_draft
2. https://medium.com/globant/how-to-solve-a-large-number-of-small-files-problem-in-spark-21f819eb36d3