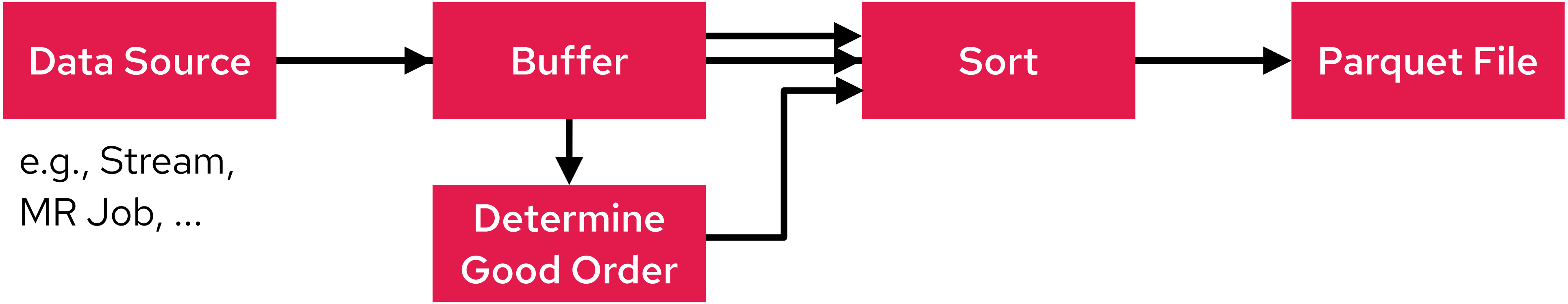
# Schema-based Column Reordering for Dremel-encoded Data

Patrick Hansert and Sebastian Michel

BiDEDE@SIGMOD 23
Seattle, WA, USA

DBIS LAB
University of Kaiserslautern-Landau (RPTU)

# Ingestion in Data Lakes

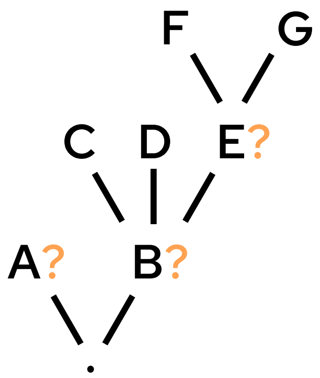



Buffer and determining the sort order can be memory intensive!

→ Can we determine the sort order before any data is buffered,
i.e., just with the schema?

# Dremel Encoding[1]

- Column-oriented storage of nested data
- Requires schema
- One column for each root-to-leaf path
- Encode NULL values as **definition level** (DL):
  Number of present optional steps




| | A | B.C | B.D | B.E.F | B.E.G |
|---|---|---|---|---|---|
| {"B": {"C": 3, "D": 7, "E": {"F": 5, "G": 2}}} | 0 | 1 | 1 | 2 | 2 |
| {"B": {"C": 3, "D": 7}} | 0 | 1 | 1 | 1 | 1 |
| {"A": 4} | 1 | 0 | 0 | 0 | 0 |

[1] Melnik et al. “Dremel: Interactive Analysis of Web-Scale Datasets”, VLDB 2010

# Column Reordering

- Run-length encoding employed for definition levels & Boolean columns
- Sensitive to row order
  → Optimal order is NP-hard[2]
- Heuristic: Sort lexicographically
  → Optimal column order is still NP-hard[2]
- Increasing-cardinality heuristic[2]:
  **Sort rows lexicographically, considering columns in the order of their increasing cardinality**

| A | B.C | B.D | B.E.F | B.E.G |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 2 | 2 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 | 2 |

→ 22 Runs

[2] Lemire and Kaser "Reordering Columns for Smaller Indexes", Inf. Sci. , Vol. 181, No. 12

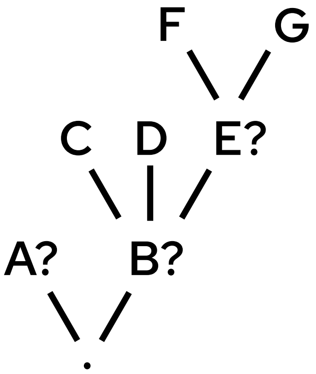
# Schema:
# Block-based Sorting

# Key Observation: Interdependence

For definition levels, the schema details:

- Min & Max value → cardinality
- Dependencies

Each DL determines the DL of all paths with a prefix of its optional nodes



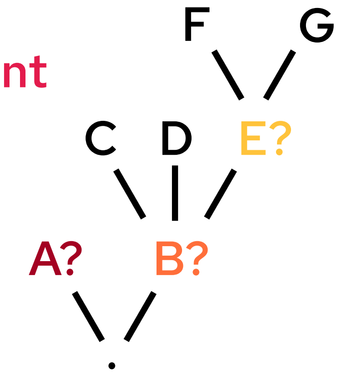

| A | B.C | B.D | B.E.F | B.E.G |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 2 | 2 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 | 2 |

**But: the increasing-cardinality heuristic assumes data independence!**

# Blocks of Dependent Data

Consider data block each optional node **affects**:

- Columns it **appears** on
- Rows where **parent is present**
- Nested wrt. schema
- Repeated wrt. sort order




B.E.F : 2

A : 1

B.C : 2

| A | B.C | B.D | B.E.F | B.E.G |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 2 | 2 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 2 | 2 |

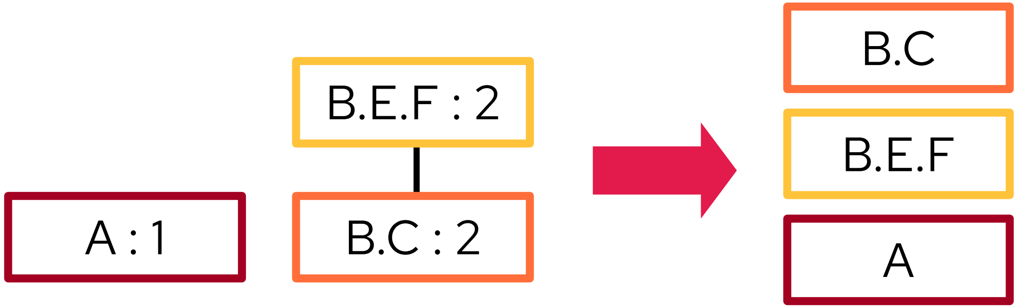
**→ Upper bound:**
**Each block at most doubles runs**

# Sort Blocks by Decreasing Size

- All block orderings adhere to tree order
- Nodes with **many columns first**
  → upper bound is minimized
- Deriving and ordering blocks in $\mathcal{O}$(n log n) using heapsort

| A | B.C | B.D | B.E.F | B.E.G |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 2 | 2 |
| 1 | 1 | 1 | 2 | 2 |

→ 16 Runs



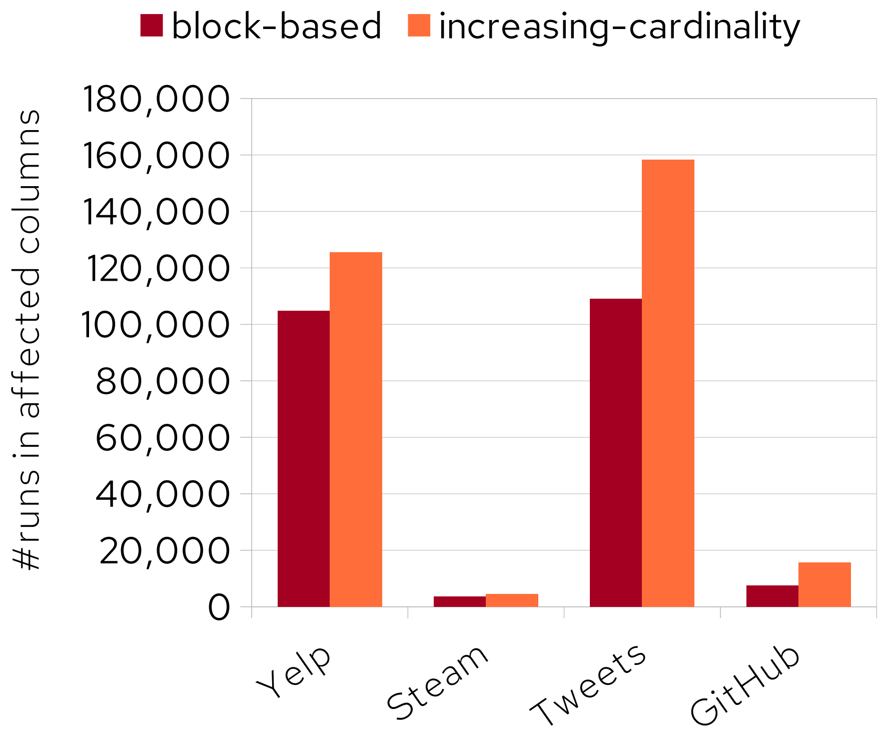
# Evaluation

# Experimental Setup

- Spark 3.2.0 & Parquet
- Extract schema and then sort in multiple ways:
  - → Unsorted
  - → Increasing-cardinality (schema)
  - → Block-based (schema)
  - → Exact

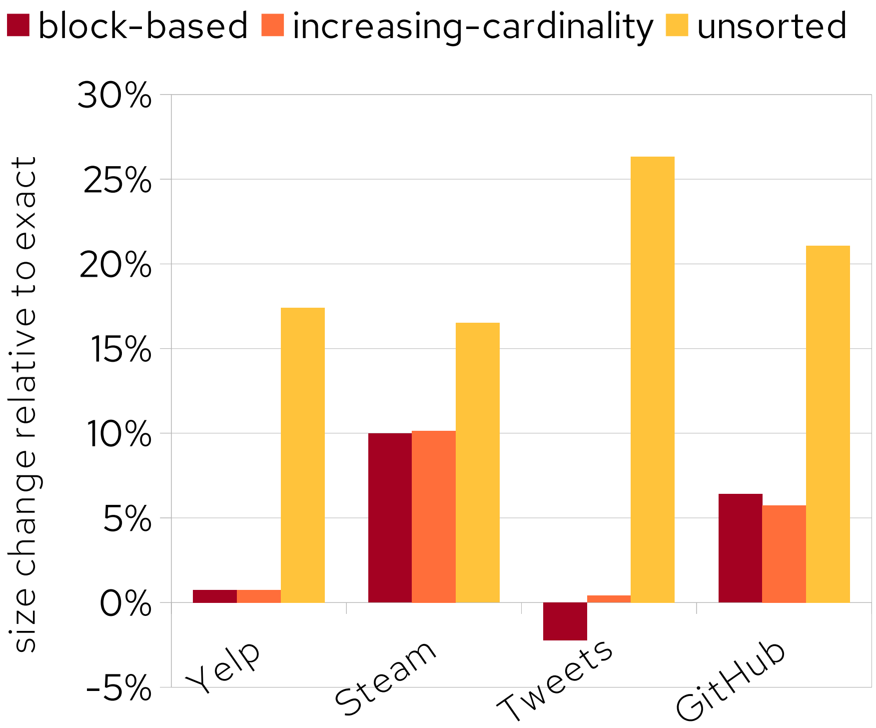
| | Yelp | Steam | Tweets | GitHub |
|---|---|---|---|---|
| **Total Nodes** | 61 | 54 | 1,101 | 705 |
| **Optional Nodes** | 49 | 18 | 748 | 134 |
| **Boolean Leaves** | 0 | 0 | 58 | 43 |
| **Document Count** | 150,346 | 74,821 | 79,219 | 218,939 |

# Decreased Runcount




- Number of runs in definition level and Boolean columns
- Relevant metric for bitmap Indexes
- Block-based between **factor 1.19 and 2.06 better**
- Relative gains smaller when considering all columns

# Compression Rate




- File size relative to increasing-cardinality with exact cardinalities
- Yelp and Tweets very **close to Exact**
- Steam and GitHub still within 10% → lower optional node count
- All results better than unsorted
- Block-based **on average 0.53% better than increasing-cardinality**

# Conclusion

- Schemas provide **ample information** for column reordering
- Block-based improves runcount between **factor 1.19 and 2.04** over increasing-cardinality
  → Good for **bitmap indexes** over the structure
- **Comparable to compression rates** (within 10% or less) despite:
  - Use less information
  - Less computationally intensive
  - Fewer columns considered