PosDB in 2023: State of Affairs

George Chernishev
chernishev@gmail.com

Saint-Petersburg State University, Russia

ACM SIGMOD Moscow Chapter Talk
April 6th, 2023

Talk outline

- Starting point: PosDB in 2018[1] [CGG+17, CGG+18]
- Three basic query processing strategies [CGG+22]
  - \+ first benchmark vs industrial systems
- Hybrid materialization strategy (preprint in the works)
- Research:
  - Window function computation [MGC19]
  - Intermediate result caching [GKC20]
  - Data compression [SKSC21]
  - External sort [PGSC22]
  - Recursive query processing (preprint in the works)
- Technical Improvements
  - Disk sub-system with a proper buffer manager
  - Distributed join and distributed aggregation operators with data repartition
  - Catalogue evolution
  - Parser and a simple plan generator

[1] A recap of my previous talk, meeting 202 (November, 29, 2018) `https://synthesis.frccsc.ru/sigmod/seminar/s20181129.html`

Column-Store Basics

Columnar Approach: Contemporary View

Nowadays many industrial systems call themselves “columnar”.

They treat column-orientation as storage level-only:

- processing is usually organized as follows: “read, decompress data, construct tuples, continue to work as usual”;
  - early materialization
- allows to read only requested columns;
- efficient column-oriented compression.

Columnar Approach: Founders Vision

However, founders proposed not only column-oriented data ***storage***, but also column-oriented data ***processing*** [ABH13]:

- query plans allow operators exchange not only data, but also positions;
- an option to select tuple reconstruction time: transition from positions to records
  - materialization strategy
- additional benefits:
  - operating directly on compressed data
  - conserving I/O bandwidth further
  - reducing the CPU processing load

$\longrightarrow$ novel operators, novel query plans.

Starting point: PosDB in 2018

PosDB: why?

- Inspired by old-school column-stores such as C-Store [SAB+05] and MonetDB [IGN+12]
  - Position-enabled systems
- Existing (at the time of project start) works not touched:
  1. Positions-enabled processing (i.e. late materialization) for aggregation, subqueries, ...
  2. Distributed processing for such column-stores was not studied at all
- Why new prototype?
  - Project started in 2016, at that time there were no open-source disk-based column-stores $\longrightarrow$ Need a prototype of a new distributed column-store

PosDB — a disk-based column-store for research purposes:

- Relies on Volcano block-based iterator model [Gra93].
- Columnar: data is stored in columns, focus on analytic queries.
- Disk-based: data >> main memory.
- Distributed: has send & receive operators. Not mediator-based, but "true" distribution of data and queries.
- Parallel: any operator sub-tree can be executed in a separate thread.
- Query evaluation focuses on late materialization approach.

PosDB in 2018 II: core

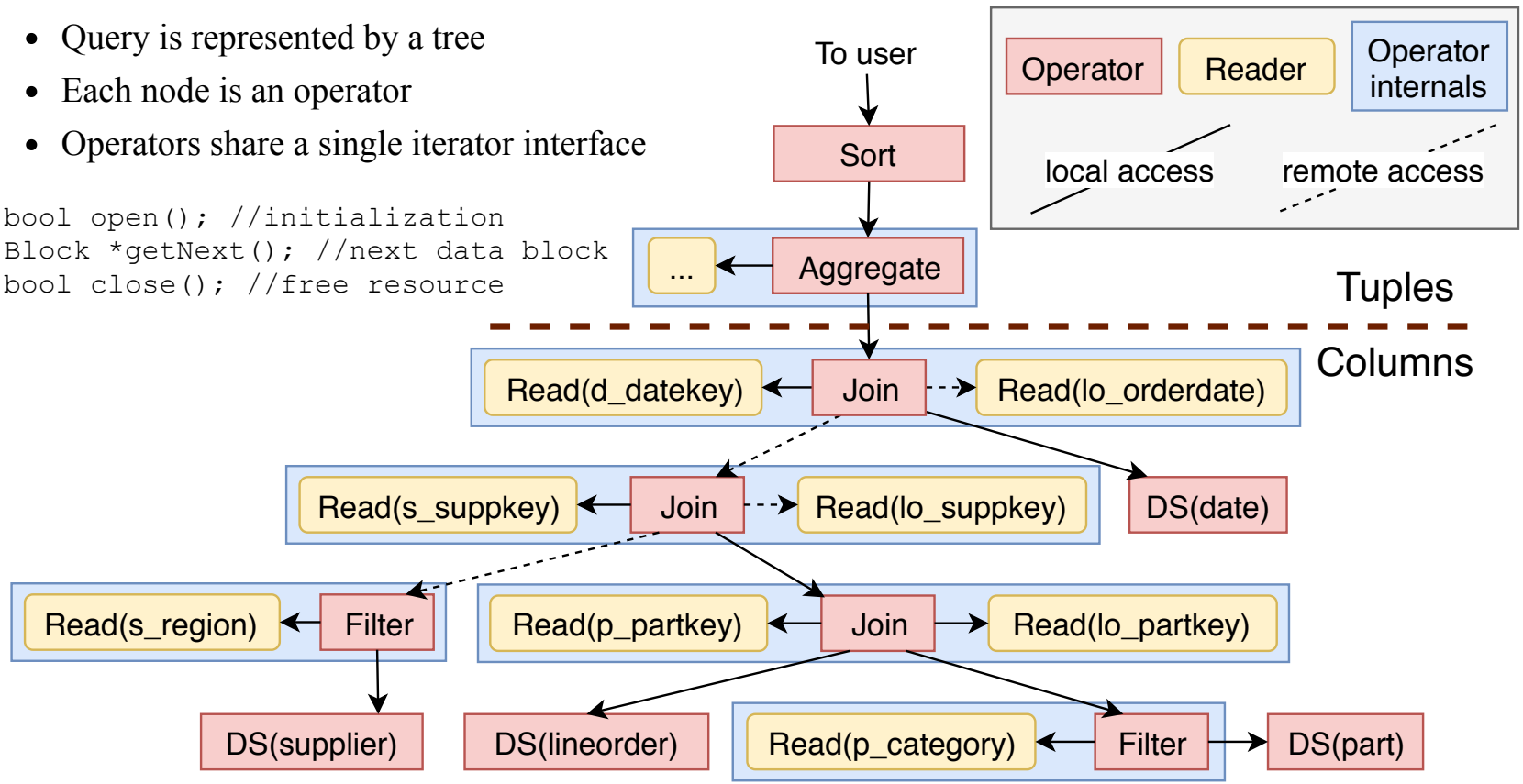
PosDB — a disk-based column-store for research purposes:

- Position-enabled column-store: operators pass not only data, but also positions
- Two types of data representation: tuple and positions (join index)
- Two types of operators: accepting positions or tuples
  - e.g. TupleHashJoin & HashJoin
- Reader: auxiliary entity controlled by operator, used to get data
- Materialization: process of turning positions into tuples
- Materialization point — place in a plan, where positions are turned into tuples

Query plans, Volcano model, Late materialization

- Query is represented by a tree
- Each node is an operator
- Operators share a single iterator interface

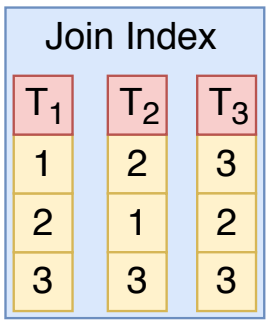
```
bool open(); //initialization
Block *getNext(); //next data block
bool close(); //free resource
```

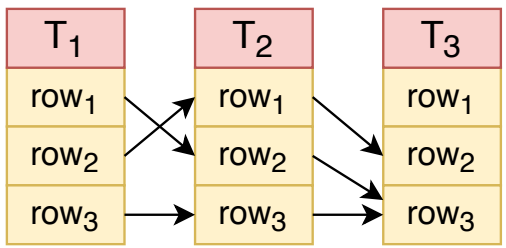


Join index

Query may contain joins, therefore a special data block is required:

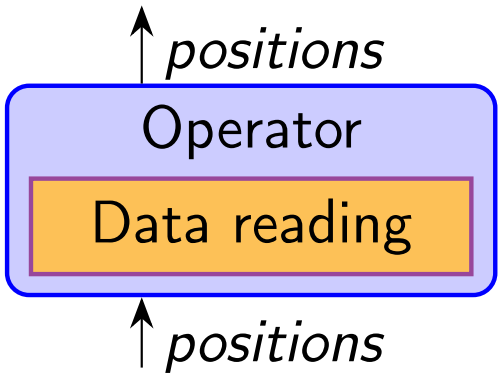
- Use classic data structure [Val87];
- Position lists, one per table;
- Two tables: a map of positions of T_1 into positions of T_2;
- N tables: a map of positions of T_1, T_2, T_{N-1} into positions of T_N

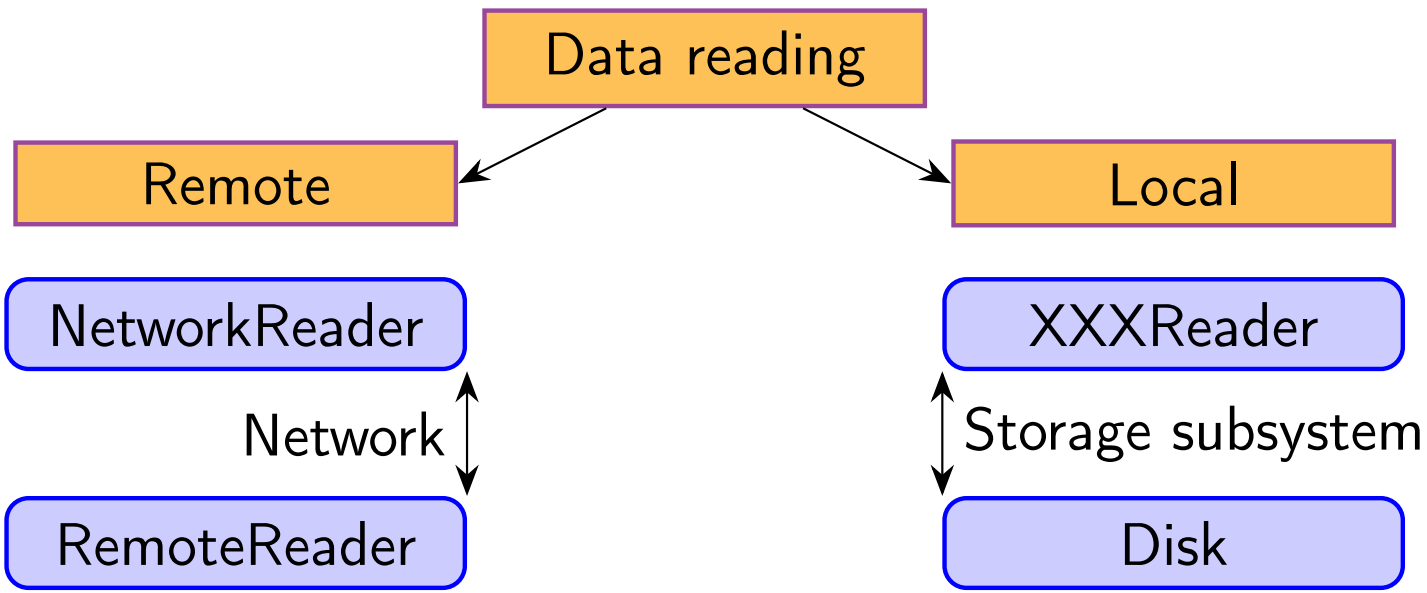




Acquiring data and positions

1. Initial `JoinIndex` acquisition happens in leaves of a tree via `DataSource` operator
2. All necessary data is read inside operators





Summary

(+)

- Focus on Star Schema Benchmark [SSB09];
- Distribution and parallelism on a plan level:
  - Both inter- and intra- query parallelism;
  - Both data fragmentation and replication;
- Lots of positional- and value- operators;

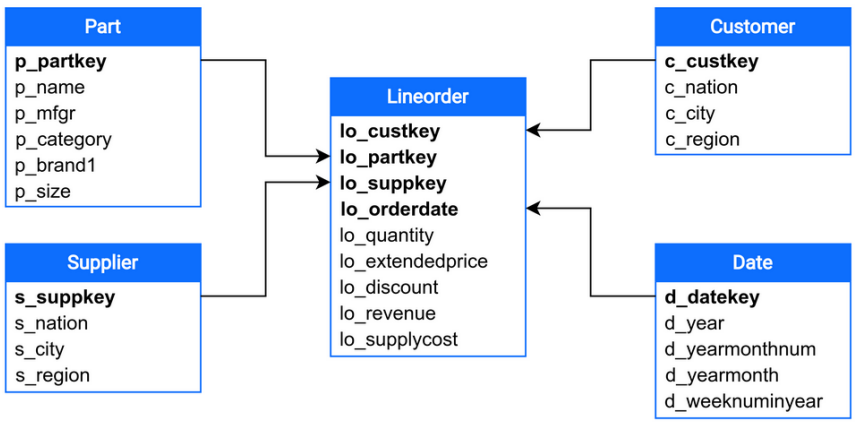
(–)

- Concentrated on query executor; no rewriter and query optimizer, statistics subsystem;
- No compression;
- No vectorized primitives and expression compilation;
- Only late materialization;

Why Star Schema Benchmark?

Star Schema Benchmark:

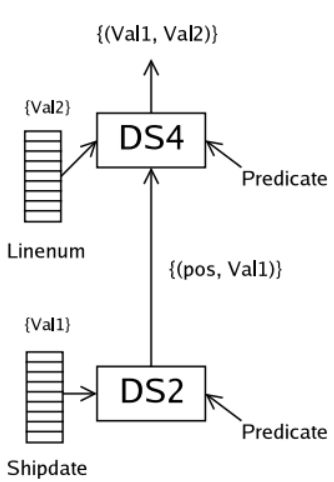
- mimics analytic workloads
- allows to benchmark engine, not optimizer
- synthetic: allows benchmarking with SF
- easy to implement: only SPJ + Groupby/Aggregate + OrderBy

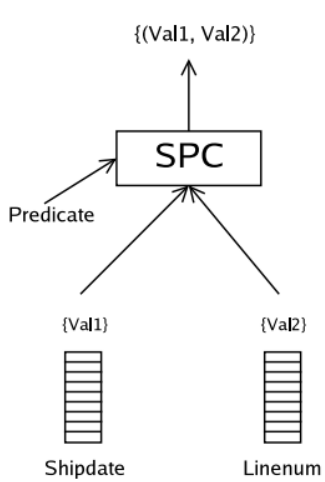


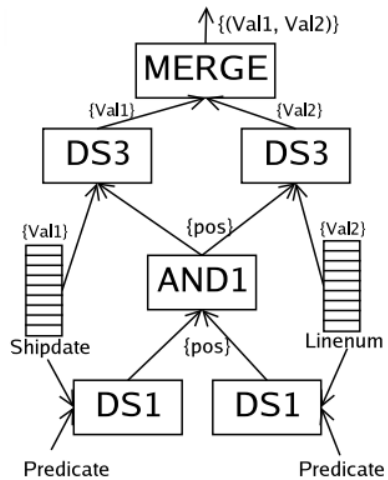
Three basic query processing strategies + first benchmark vs industrial systems [CGG+22]

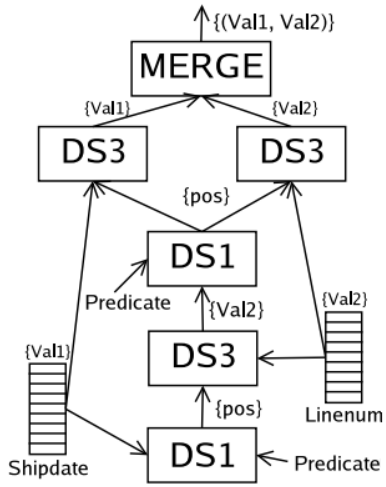
Existing Approaches to Late Materialization I

- Selections (e.g. C-Store) [AMDM07]:



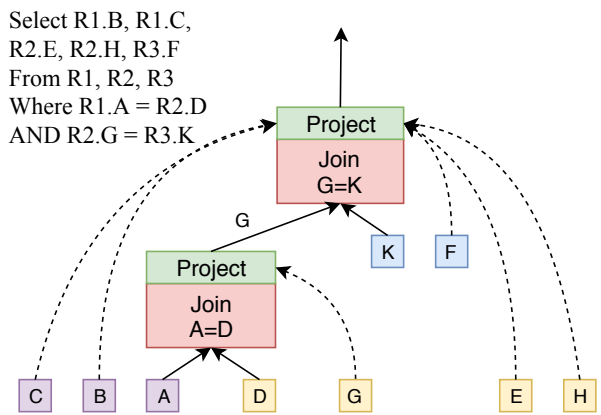






Query: *Shipdate* $<$ *const*1 *AND Linenum* $<$ *const*2

- Joins (e.g. [THS$^+$09])



Existing Approaches to Late Materialization II

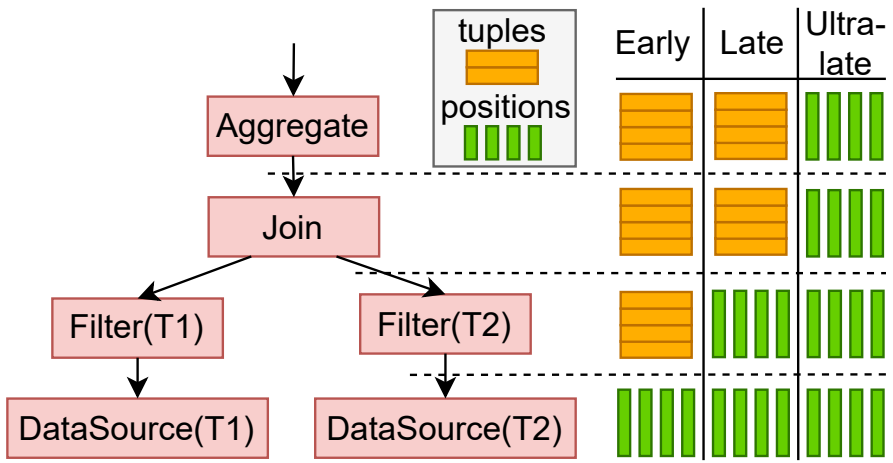
Study	Focus on LM in selections	Focus on LM in joins	Type
G. Copeland et al.[CK85, KCJ+87]	partial	full	disk
Fractured Mirrors [RDS02]	partial	partial	disk
FlashJoin [THS+09]	none	full	disk
C-Store [SAB+05, AMDM07]	full	partial	disk
MonetDB family [BK99, IGN+12]	full	full	mem
Hyrise [GKK+11]	full	none	mem

No studies which combine LM in selections and joins!

Three Strategies

Star Schema Benchmark:

- Early Materialization: no positions
- Late Materialization: positions up to joins
- Ultra-Late Materialization: positions up sort



Our Proposal: Ultra-Late Materialization Model

Idea: support late materialization in

- selections, and,
- joins

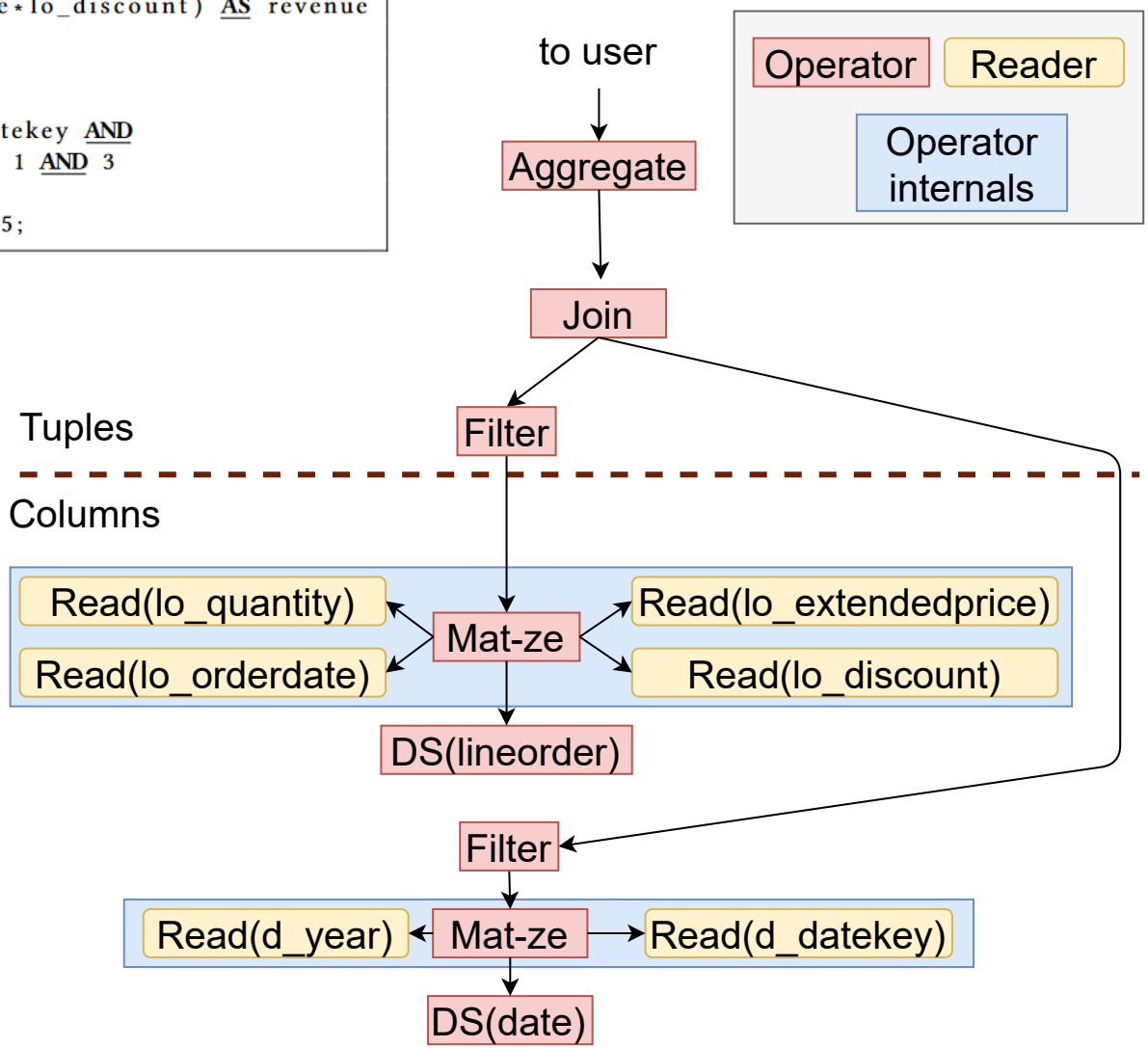
while supporting arbitrary number of joins.

Our thoughts:

- Design a new query evaluation model for disk-based systems.
- Study how will it work, compared to other models (strategies), to industrial row- and column-stores.
- Novel storage devices may have already alleviated out-of-order probing problem (at least for some cases)? If no, will external sort help?

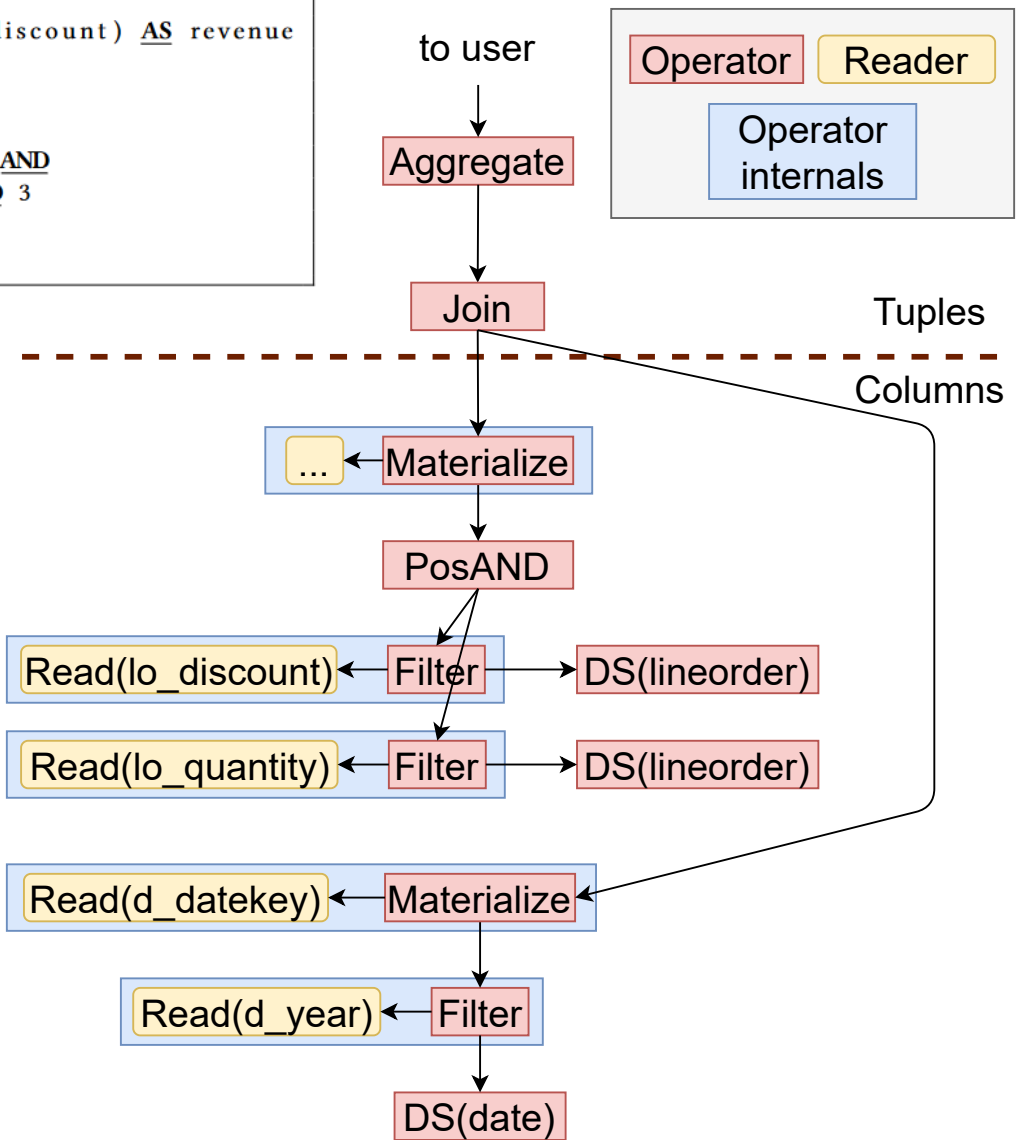
How Plans are Looking Like (EM)?

```
SELECT
    SUM(lo_extendedprice*lo_discount) AS revenue
FROM
    lineorder, date
WHERE
    lo_orderdate = d_datekey AND
    lo_discount BETWEEN 1 AND 3
    d_year = 1993 AND
    AND lo_quantity < 25;
```



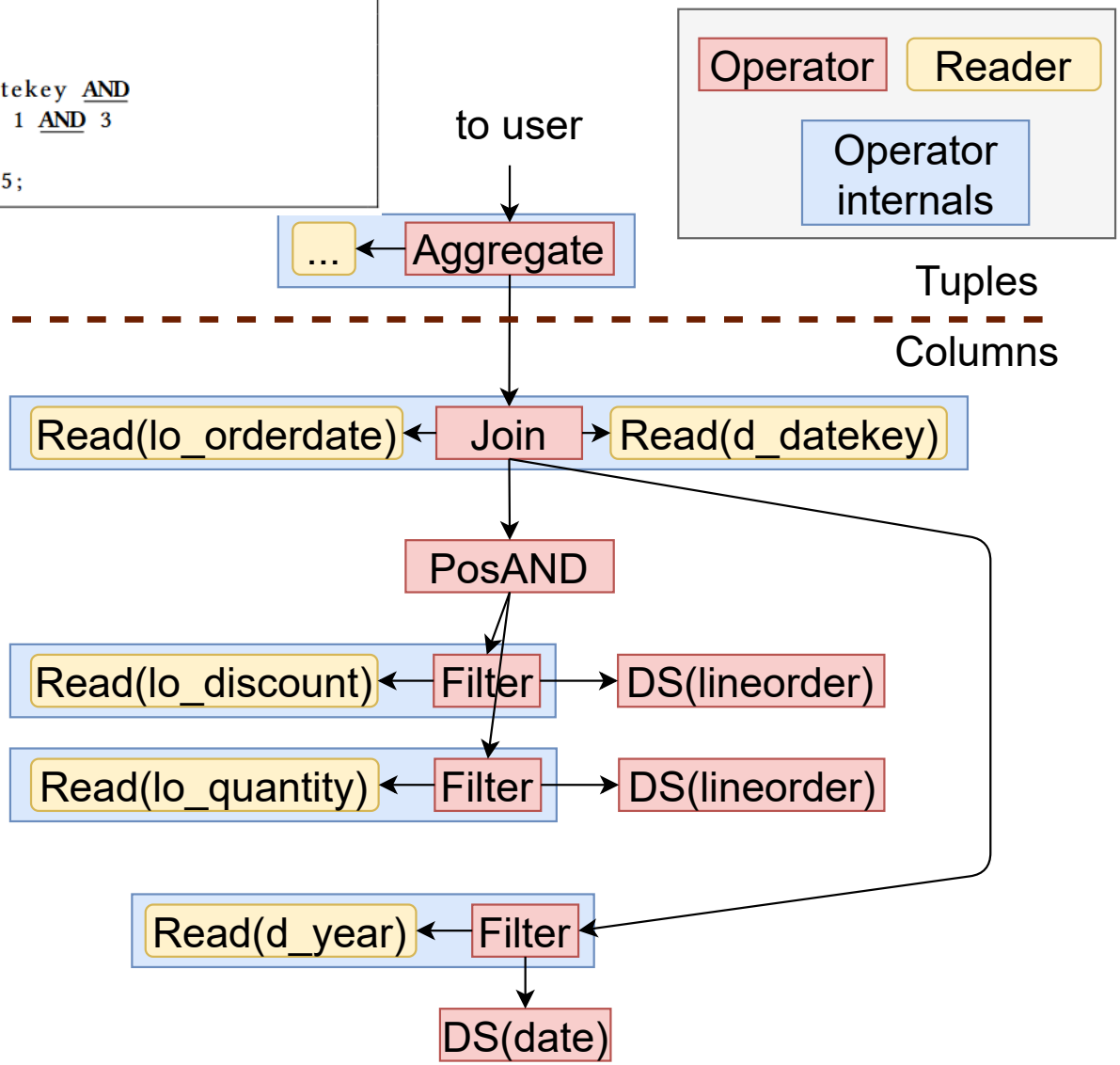
How Plans are Looking Like (LM)?

```
SELECT
    SUM(lo_extendedprice*lo_discount) AS revenue
FROM
    lineorder, date
WHERE
    lo_orderdate = d_datekey AND
    lo_discount BETWEEN 1 AND 3
    d_year = 1993 AND
    AND lo_quantity < 25;
```

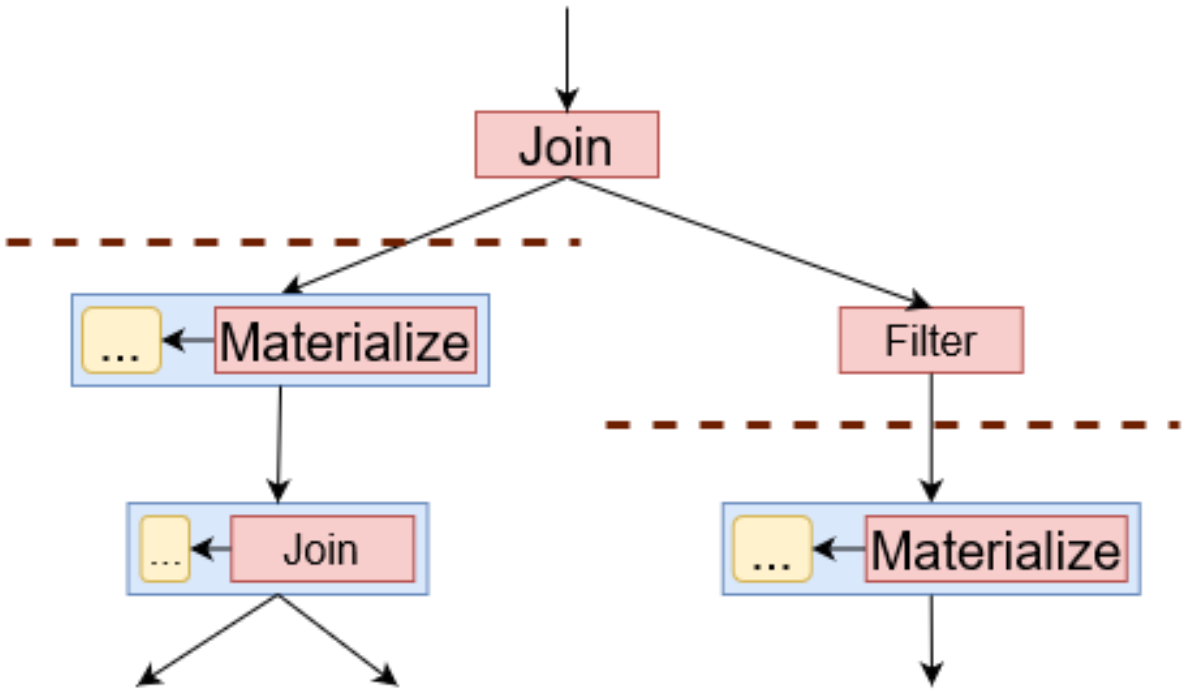


How Plans are Looking Like (ULM)?

```
SELECT
    SUM(lo_extendedprice*lo_discount) AS revenue
FROM
    lineorder, date
WHERE
    lo_orderdate = d_datekey AND
    lo_discount BETWEEN 1 AND 3
    d_year = 1993 AND
    AND lo_quantity < 25;
```



To benchmark these strategies we implemented tuple-based join, filter, cross-product, and aggregation operators.



$\longrightarrow$ As the result we can have multiple materialization points inside query plan, one per each root-leaf path. Novel query plans!

Experimental Evaluation

Setup:

- AMD Ryzen 9 3900X, GIGABYTE X570 AORUS ELITE, Kingston HyperX FURY Black HX434C16FB3K2/32 32GB, 512 GB SSD M.2 Patriot Viper VPN100-512GM28H.
- Ubuntu 20.04 LTS, GCC 9.3.0, PostgreSQL 12.5, MariaDB Column-Store 1.5.2 on MariaDB Community Server 10.5.8.
- Star Schema Benchmark with SF $\in [1; 100]$ (up to 60GB)

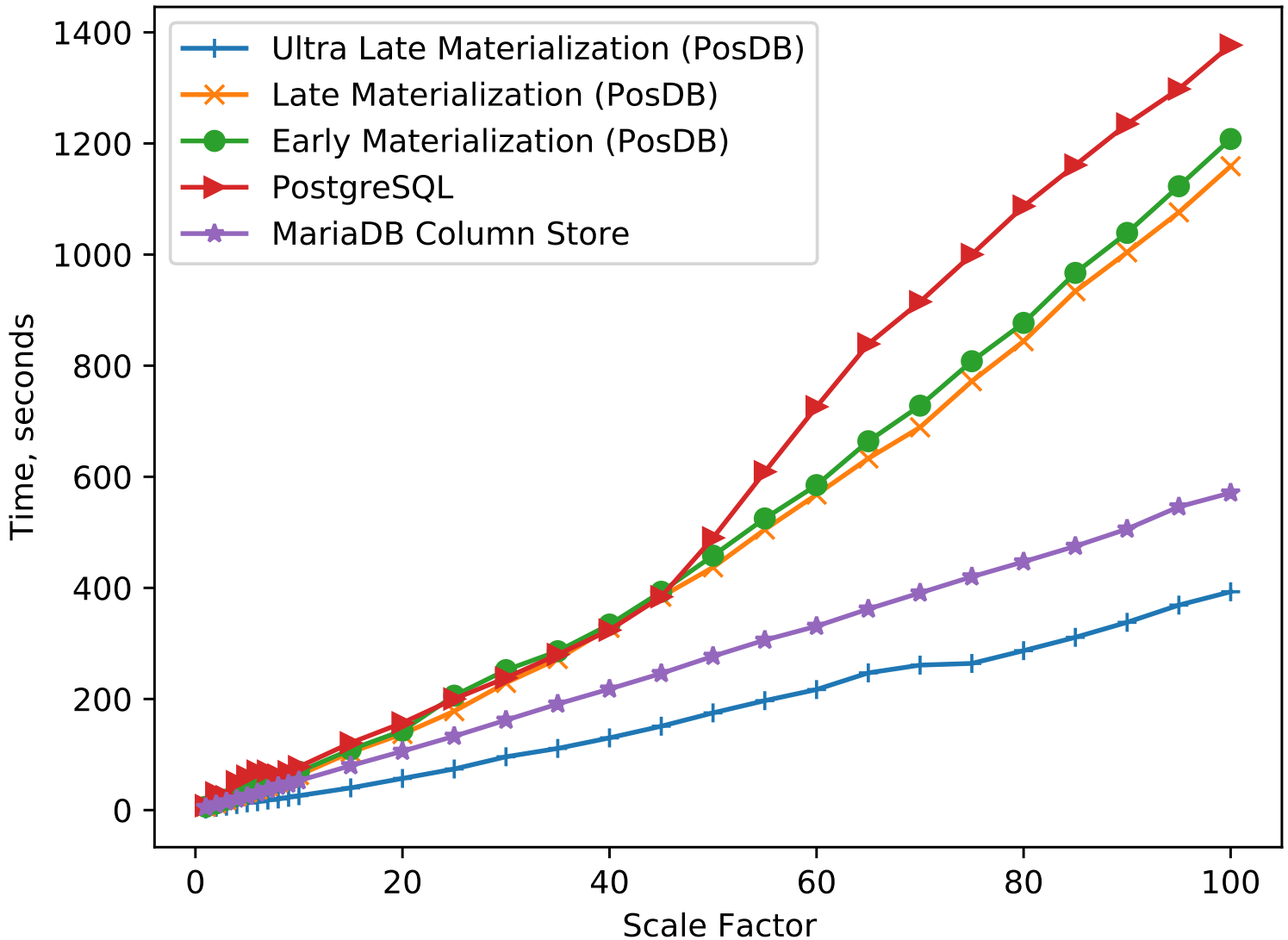
DBMS details:

- Data compression, JIT-compilation, SIMD, and indexes were not used.
- DBMSes were not tuned, default parameters were used.
- Default data plans were used.
- Hash-based versions of joins were used: the smaller table was kept.
- Intra- and inter- query parallelism was turned off + no distributed capabilities.

Studied Strategies:

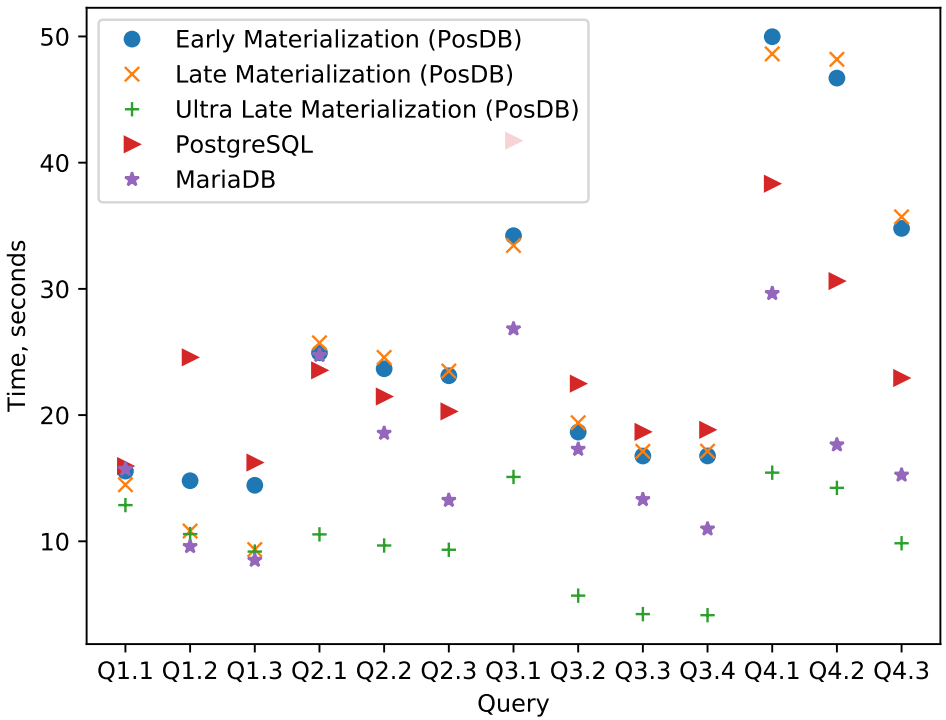
- Early Materialization,
- Late Materialization,
- Ultra-Late Materialization.

Performance on the whole Benchmark

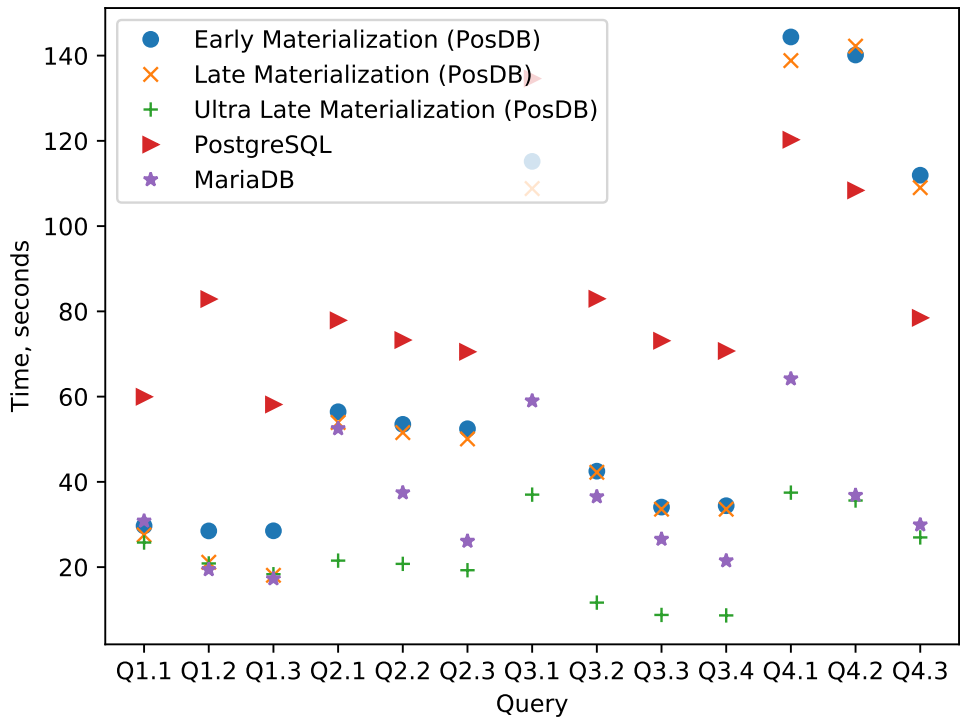


Performance on individual queries

Per-query breakdown:



(a) Scale Factor 40



(b) Scale Factor 80

Findings:

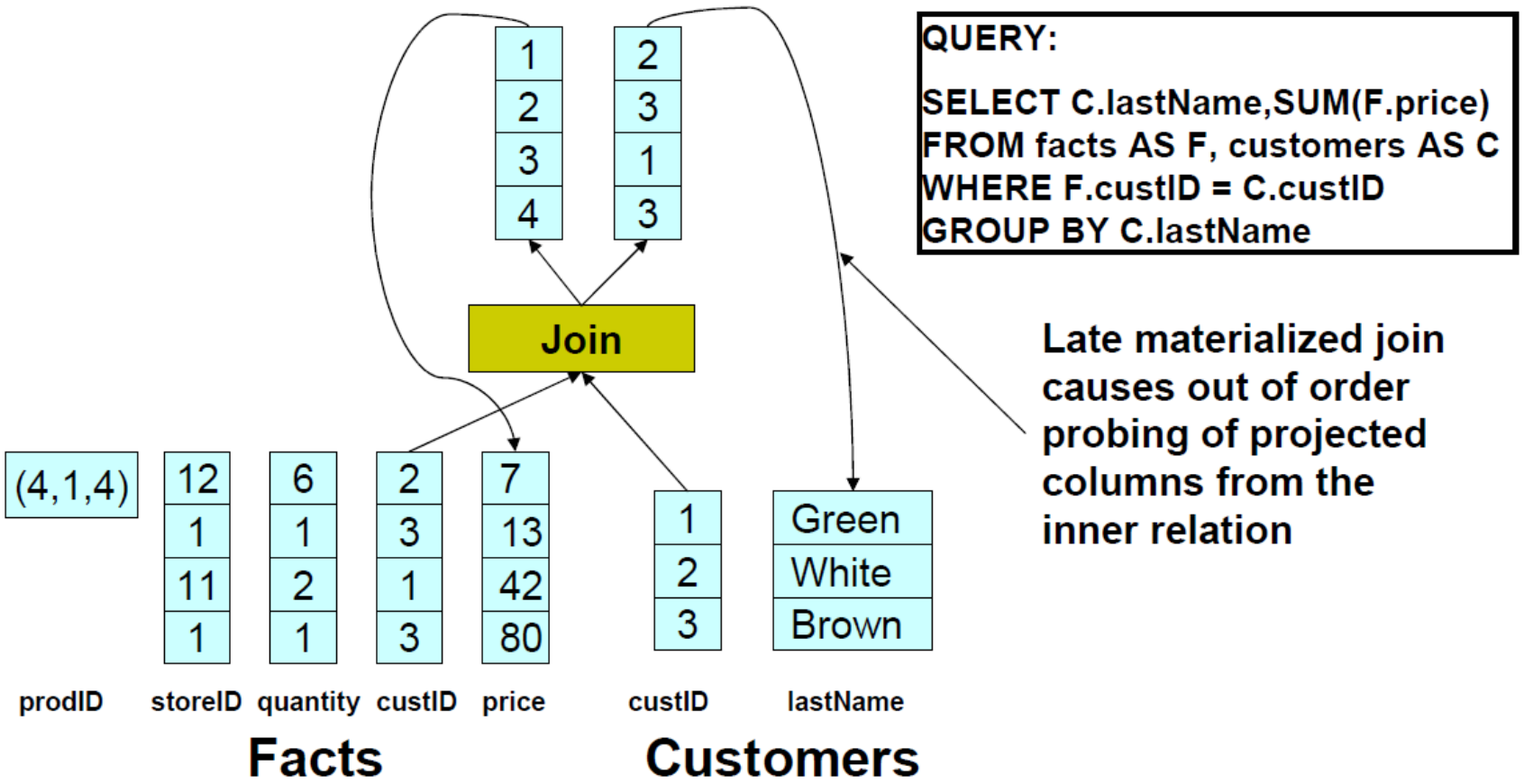
- ULM is the champion, on the whole benchmark, but there are queries where it loses.
- LM is consistently $\approx 5\%$ faster than EM.
- PostgreSQL loses about 15% to both EM and LM. This happens due to PostgreSQL running out of memory to cache pages in its buffer manager.
- MariaDB Column Store beats PostgreSQL by more 2x.

Next goals:

- Disk-spilling joins, i.e. out-of-order probing problem.
- Hybrid materialization.

Details are in our DOLAP@EDBT/ICDT'22 paper [CGG+22].

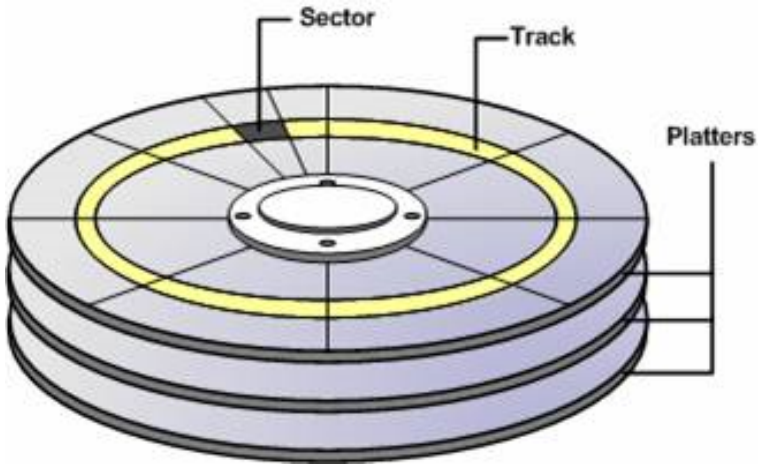
Out of order probing I



2

[2] Image taken from [HAB09] presentation.

Out of order Probing II



[3]

Time expenses:

- Seek time + Rotational latency = about 2.5 ms
- Command processing time = little
- Settle time = little

Read 100 items using random access — 250ms. Sequential: 2.5ms + 0.1ms.

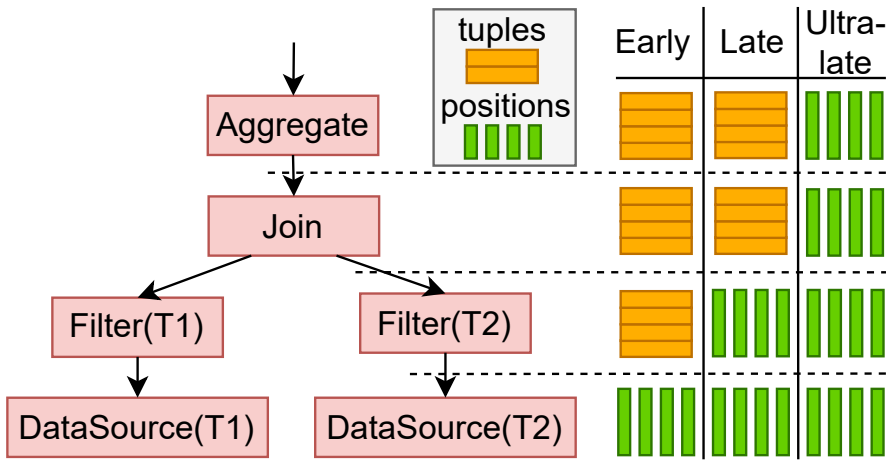
$\longrightarrow$ need to avoid out of order probing!

[3] Image taken from `http://www.applexsoft.com/glossary/hard-disk.html`

Three Strategies, once more

If we are thinking about real application, it is essential to include all three strategies into the engine.

- Early Materialization: no positions — classic approach, fallback strategy
- Late Materialization: positions up to joins — relatively safe, but no large gains
- Ultra-Late Materialization: positions up sort — very risky, more research is required



Is LM in Disk-Based Systems Relevant?

- All previous LM disk-based studies considered HDDs, except [THS+09].
  - Now, SSDs are ubiquitous, not a rarity
  - SSDs of 2008 != 2022s, they have been seriously improved
  - Good random access gains
  - Novel types of storage are appearing
  - Hope to address the out-of-order probing problem
- Resurgence of interest to late materialization-enabled systems, need of supporting provenance in systems for visual analytics [Wu21, PW18a, PW18b].
- Position-enabled processing will be extremely useful for implementing functional dependency predicates inside queries [Che20, BSC20].

$\longrightarrow$ Disk-based LM reevaluation is required.

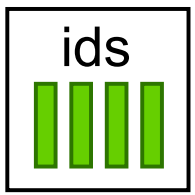
Hybrid materialization strategy (preprint will appear soon)

Aspects of materialization strategies:

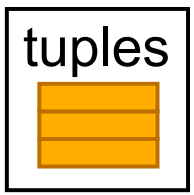
Strat.	Fast predicates	Re-read in predicates	Pre-read before joins	Re-read in joins	out of order probing
Early	No	No	Yes	No	No
Late	Yes	Yes	Yes	No	No
Ultra-late	Yes	Yes	No	Yes	Yes
Hybrid	?	?	?	?	?

Hybrid materialization strategy II: data blocks

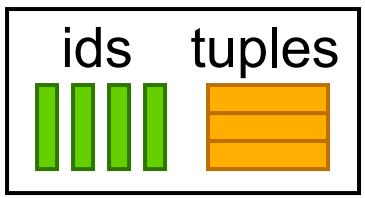
Block structures in PosDB'23:



(a) Positional



(b) Tuple



(c) Hybrid

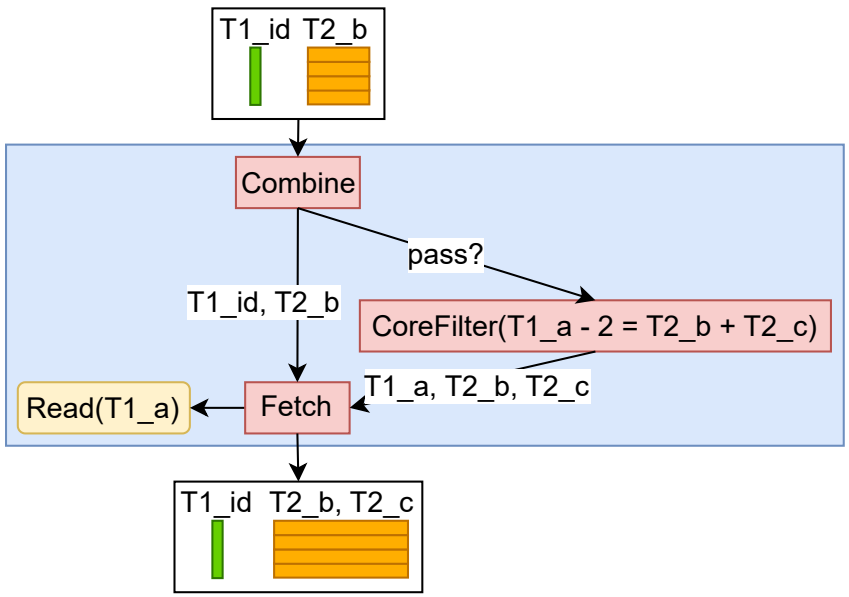
Idea:

- introduce a data block which can store positions and tuples
- extend engine with operators to use it

Hybrid materialization strategy III: hybrid operator

Hybrid operator consists of:

- fetch
- core
- combine



SELECT T1.id, T2.b FROM T AS T1, T AS T2 WHERE T1.a - 2 = T2.b + T2.c

Hybrid materialization strategy IV: algebra

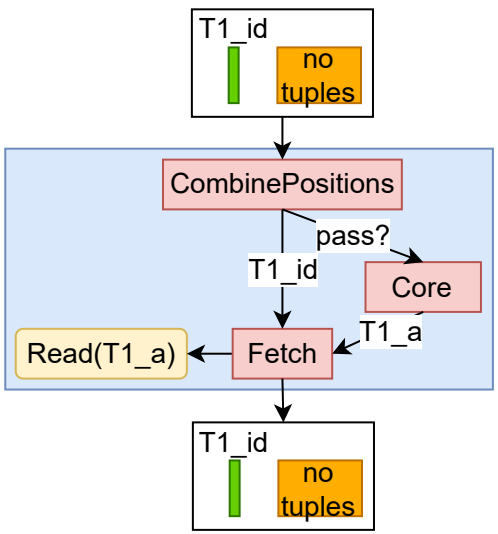
Extending algebra of operators:

1. HYDataSource: Nil $\rightarrow$ HybridBlocks,
2. HYFilter, HYProject, HYMaterialize: HybridBlocks $\rightarrow$ HybridBlocks;
3. HYHashJoin, HYNestedLoopJoin: {HybridBlocks1, HybridBlocks2} $\rightarrow$ HybridBlocks;
4. HYToTuple: HybridBlocks $\rightarrow$ TupleBlocks.

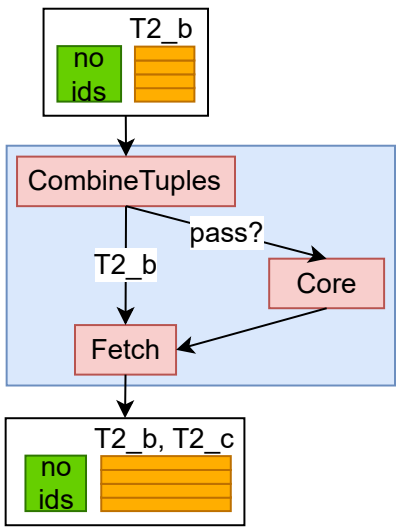
No aggregation/sort operators yet, we concentrate on SPJ queries.

Hybrid materialization strategy V: operator specializations

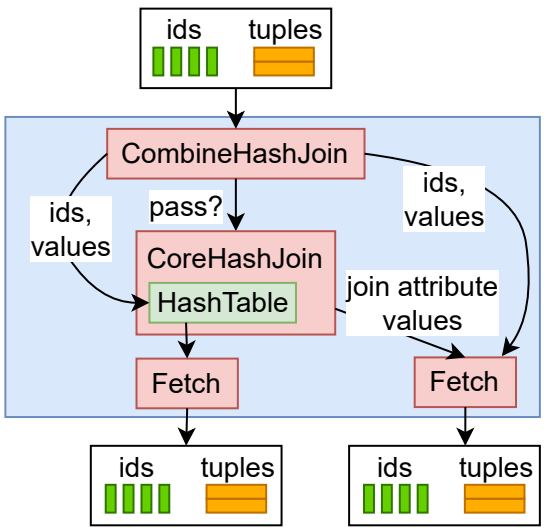
Fetch-Combine specializations:



(a) For positions



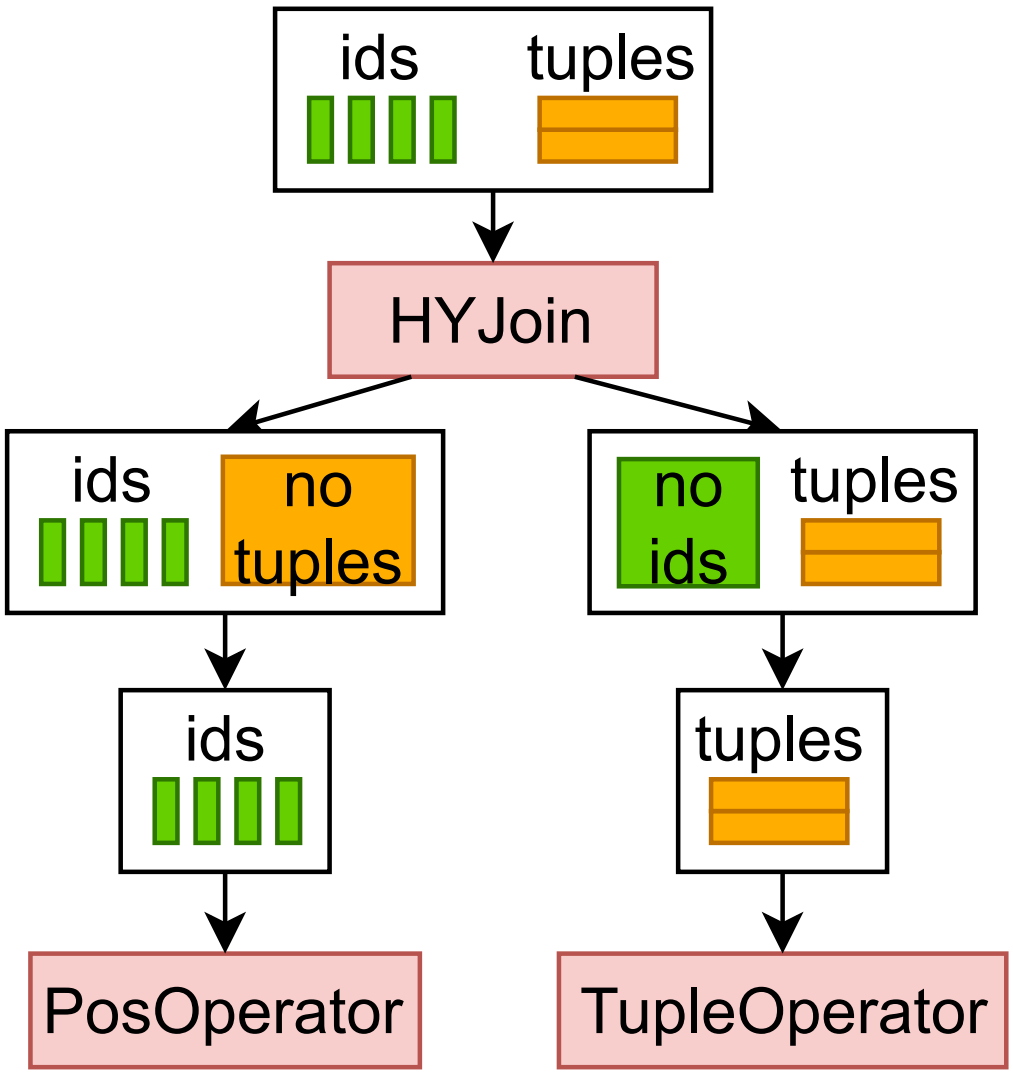
(b) For tuples



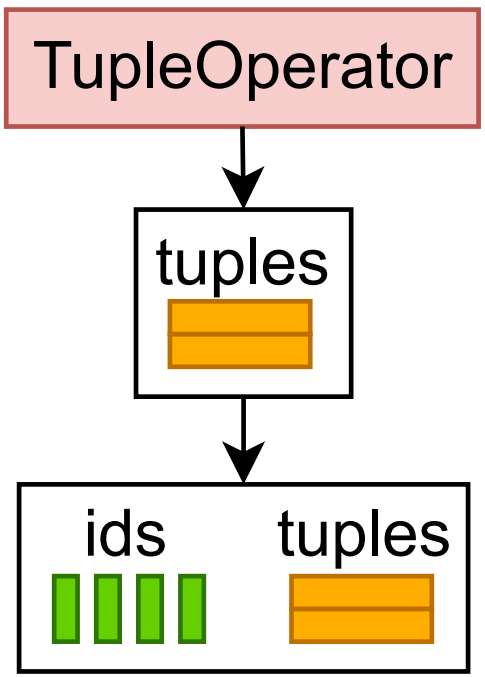
(c) For HashJoin

Hybrid materialization strategy VI: data transitions

Transition between materialization strategies



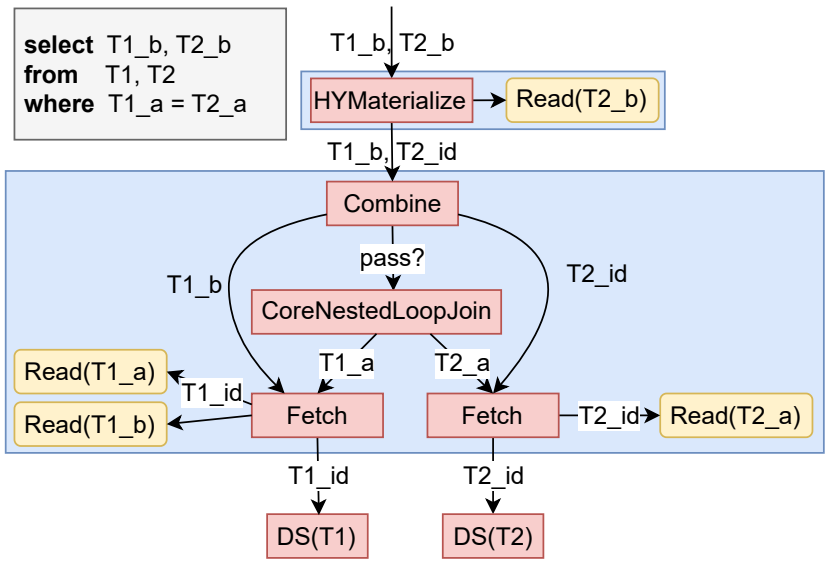
(a) To hybrid



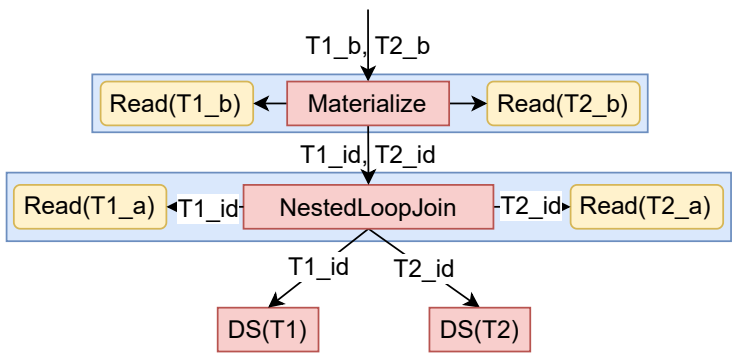
(b) From hybrid

Plans in Hybrid Model vs ULM

Join between two big tables:



(a) Hybrid materialization



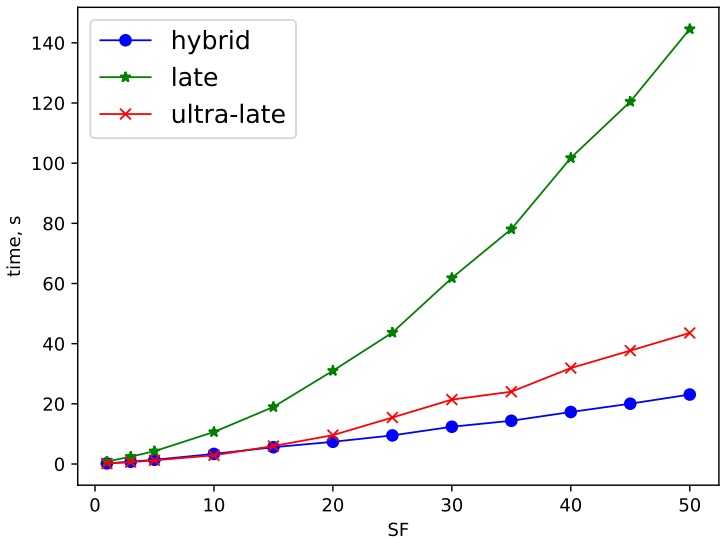
(b) Ultra-late materialization

Modified TPC-H Q9:

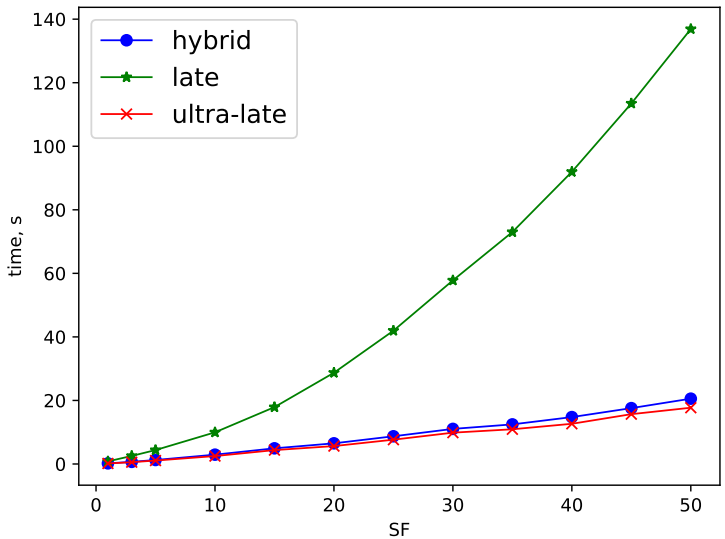
```
SELECT
    c_name, o_totalprice , o_shippriority , l_orderkey ,
    l_extendedprice * (1 - l_discount )
FROM
    nation , customer, orders , lineitem
WHERE
    n_name = ALGERIA AND
    n_nationkey = c_nationkey AND
    c_custkey = o_custkey AND
    o_orderkey = l_orderkey ;
```

This time three large tables.

Evaluation II: results



(a) With out-of-order probing

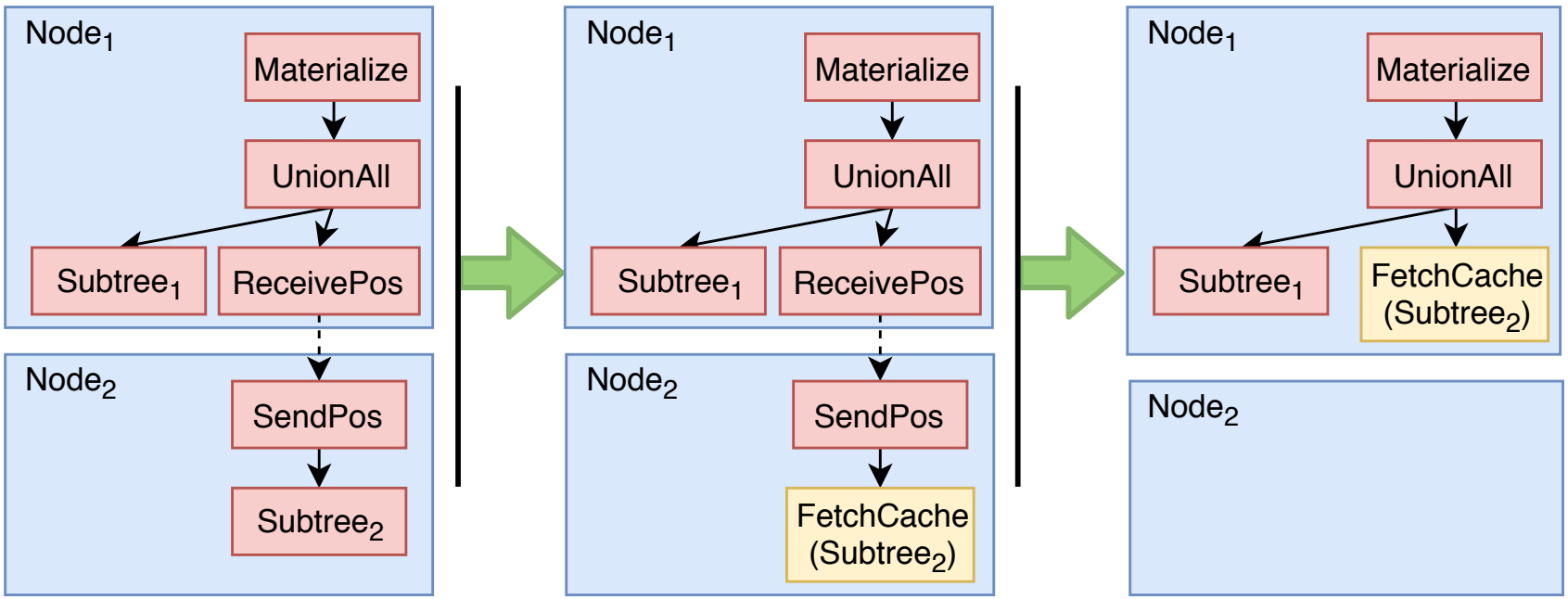


(b) Without out-of-order probing

For more details consult preprint/published version.

Research papers: intermediate result caching [GKC20]

Intermediate results caching



- Before materialization; only positions are stored
- Intermediates are stored in-memory
- Allows compression of stored results to reduce memory footprint

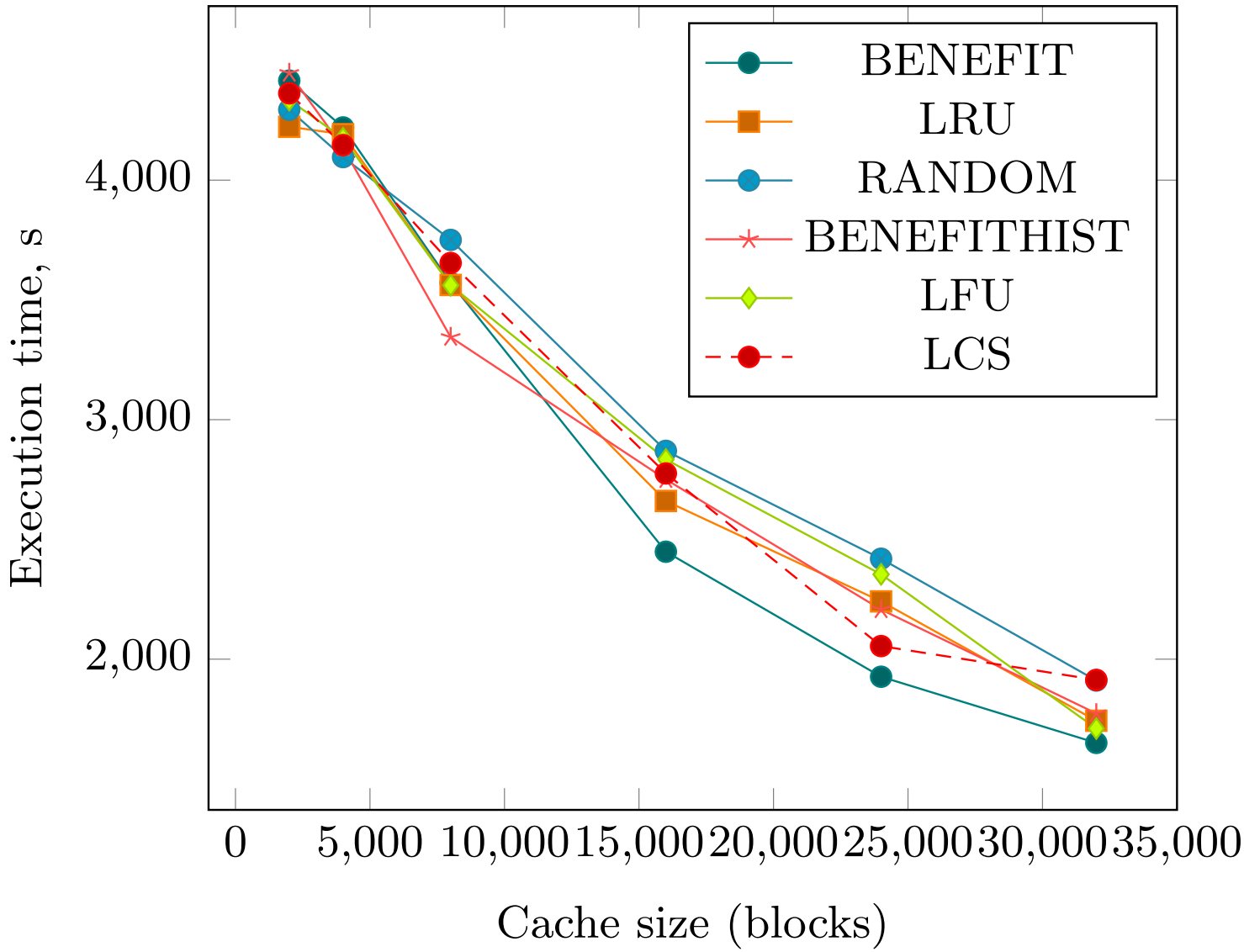
Intermediate results caching: the main idea

```
struct QueryDescription {
    set<Partition> partitions;
    set<pair<Column, Column>> joins;
    set<ConstPredicate> const_predicates;
    set<ValueList> specific_values;
    Buffer plan;

    bool contains(const QueryDescription &other);
    double complexity();
    size_t expectedBlocks();
};
```

1. Reduce every subplan to a descriptive structure
2. Keep track of N last queries
3. Estimate every result's benefit as a function of computational complexity and size

Intermediate results caching: performance



Details in our DOLAP@EDBT/ICDT'20 paper [GKC20].

Research papers: window function computation [MGC19]

Window functions processing in PosDB

Implemented window functions inside PosDB. Contribution:

- Proposed three possible materialization strategies and memory consumption models for them
- Segment tree generalization
- Segment tree application for evaluation of `RANGE`-based window functions

Window functions: syntax and concepts

```
window_function(column) OVER (
   [ PARTITION BY column [ , ... ] ]
   [ ORDER BY column [ ASC | DESC ] ]
   [ { ROWS | RANGE } BETWEEN frame_start AND frame_end ]
),
```

where *frame_start* may be

- UNBOUNDED PRECEDING
- ***offset*** PRECEDING
- CURRENT ROW

and *frame_end* may be

- CURRENT ROW
- ***offset*** FOLLOWING
- UNBOUNDED FOLLOWING

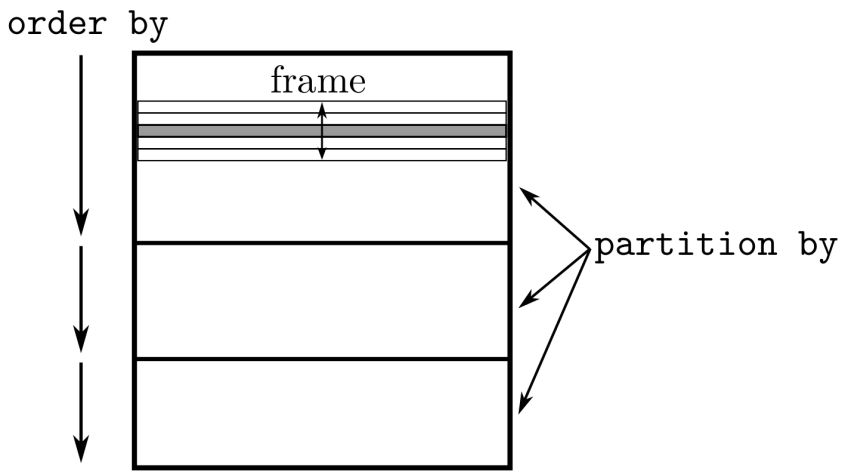


Image is taken from the paper "Efficient Processing of Window Functions in Analytical SQL Queries" of Viktor Leis et al., VLDB, 2015

We propose following strategies:

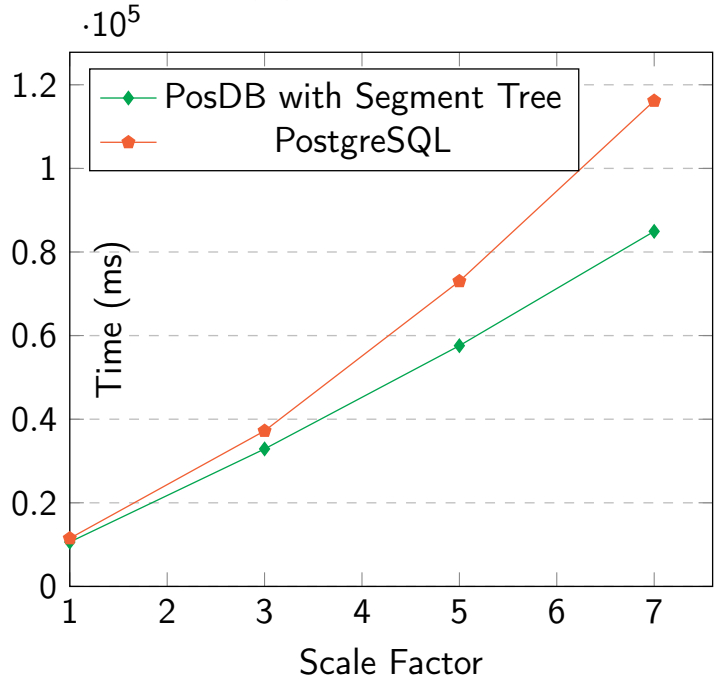
1. Tuples are materialized during partitioning
2. Only keys are materialized during partitioning, positions are stored as values in hash table
   a. All required attributes are materialized at the beginning of group processing.
   b. Only attributes required for ordering are materialized at the beginning of group processing; after ordering we can move through associated positions and materialize data on demand

For all of these strategies we have devised cost models for memory consumption

Experimental evaluation (strategy 1)

```
SELECT lo_orderpriority, SUM(lo_ordtotalprice) OVER (
  PARTITION BY lo_orderpriority ORDER BY lo_ordtotalprice
  RANGE BETWEEN offt PRECEDING AND offt FOLLOWING
) AS sum
FROM lineorder ORDER BY lo_orderpriority ASC
```

$offt = 10$



@offt	DBMS	SF=1	SF=3	SF=5	SF=7
10	PosDB	10611	32888	57484	84935
	Postgres	11498	37205	73003	116160
100	PosDB	11454	35979	63658	93743
	Postgres	11536	38046	65834	111100
1K	PosDB	11543	36390	64042	95245
	Postgres	11828	38192	66061	113689
10K	PosDB	12116	38001	67239	100130
	Postgres	11909	38460	67449	113798
100K	PosDB	12713	39973	70159	105051
	Postgres	11924	38552	N/A	N/A
1M	PosDB	13272	41552	72982	107273
	Postgres	N/A	N/A	N/A	N/A
10M	PosDB	12693	39580	69677	101774
	Postgres	N/A	N/A	N/A	N/A

Details can be found in the MEDI'19 paper "Implementing Window Functions in a Column-Store with Late Materialization"

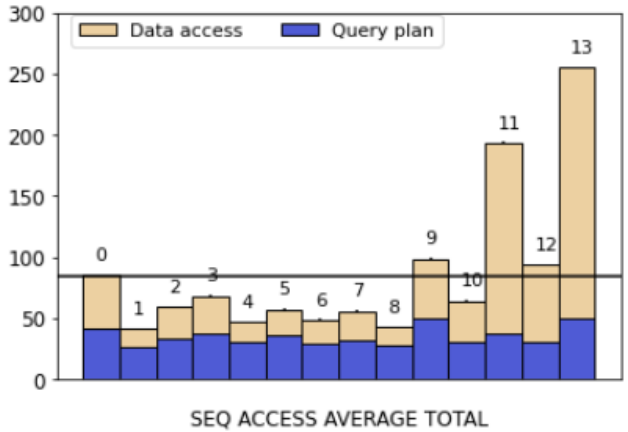
Research papers: data compression [SKSC21]

Implemented data compression in PosDB, benchmarked several algorithms.

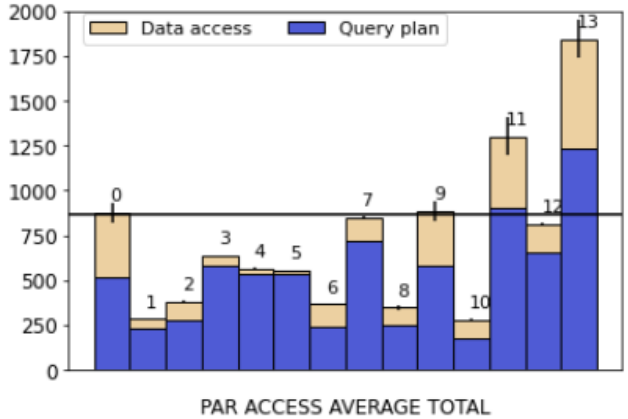
compression	year	SIMD	type	source
PFOR	2006	-	light	link[1]
SIMDFPFOR128	2014	+		
SIMDBP128	2014	+		
SIMDFPFOR128Delta	2014	+		
SIMDBP128Delta	2014	+		
VByte	2010	-		
Snappy	2011	-		link[2]
ZSTD	2015	-		
LZ4	2011	-		
Brotli	2013	-	heavy	
BSC	2009	-		
CRUSH	2013	-		
Bzip2	1996	-		
LZMA	2005	-		

1 https://github.com/lemire, 2 https://quixdb.github.io/squash/

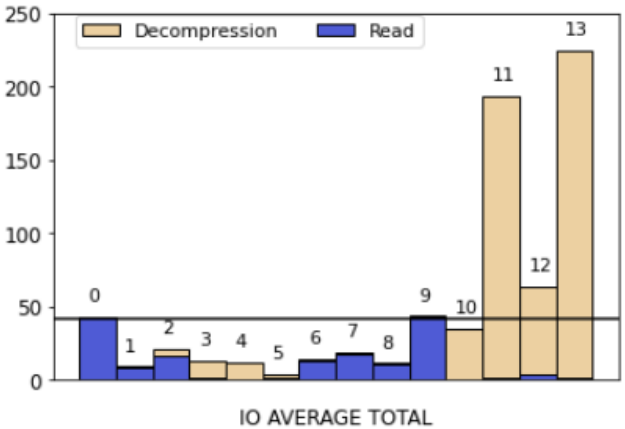
Data Compression: results



(a) System run time break down for "sequential" scenario (Seconds)



(b) System run time break down for "parallel" scenario (Seconds)



(c) IO thread action breakdown (Seconds)

0	No Compression	7	SIMDFPFor128Delta
1	PFor	8	SIMDBPacking
2	VByte	9	SIMDBPackingDelta
3	Snappy	10	Brotli
4	ZSTD	11	Bzip2
5	LZ4	12	CRUSH
6	SIMDFPFor128	13	LZMA

(d) Legend

For more information see our MEDI'21 paper and its extended version.

Research papers: external sort [PGSC22]

External Sort

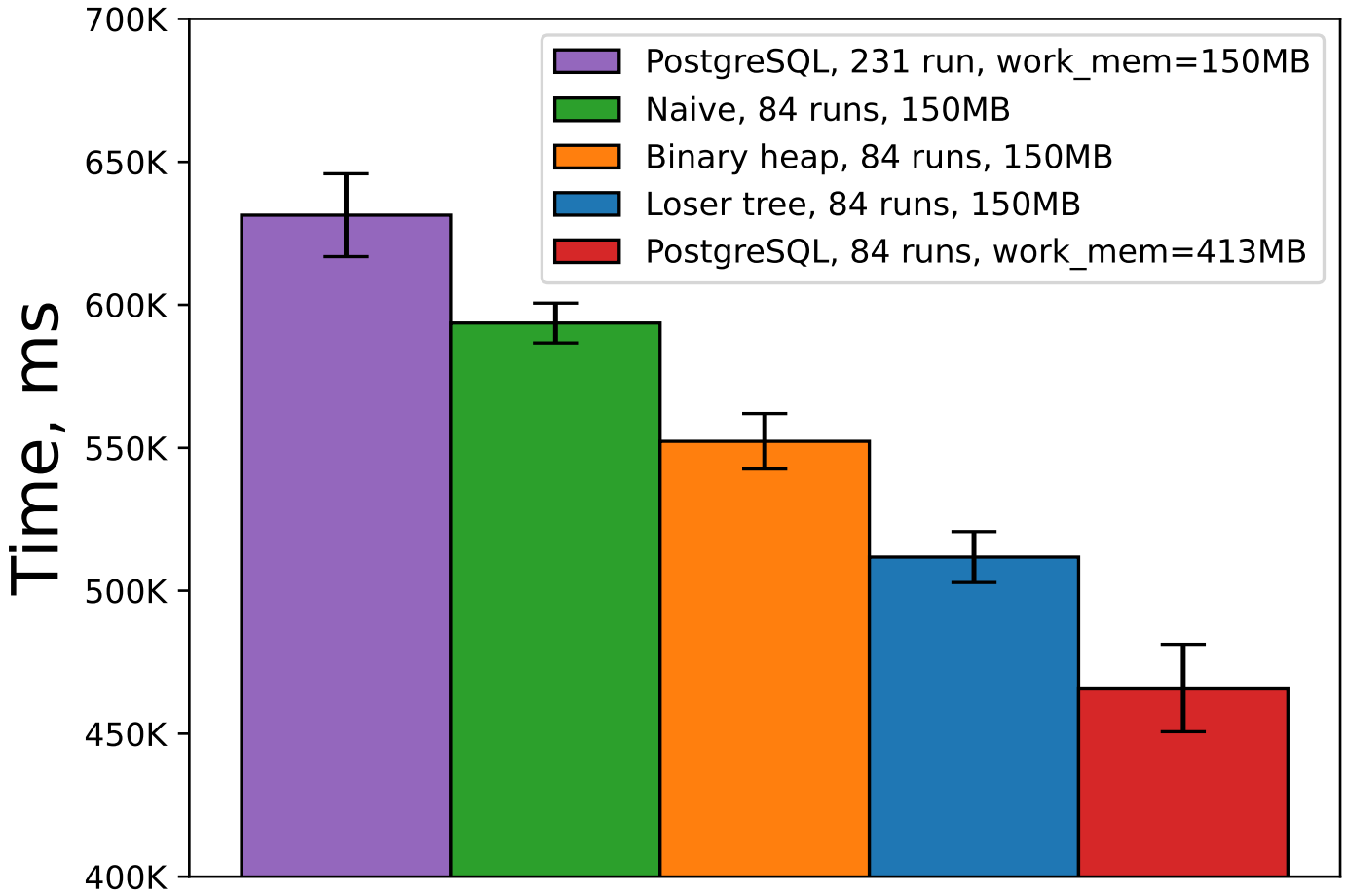
Implemented comparison-based external sort in PosDB.
External sort operator types:

- Position-based value sort: accepts positions, sorts values
- Position-based position sort: accepts positions, sorts positions — a possible solution for out of order probing problem
- Tuple-based: accepts values, sorts values

Tuple-based operator:

- Sorts pointers to tuples
- Stores runs on abstract tapes
- Merges runs using polyphase merge
- Generates runs using Introsort
- Writes generated runs directly to the disk

External Sort: Results



Details are in Megadata@ADBIS'22 paper.

Research papers: recursive query processing (preprint will appear soon)

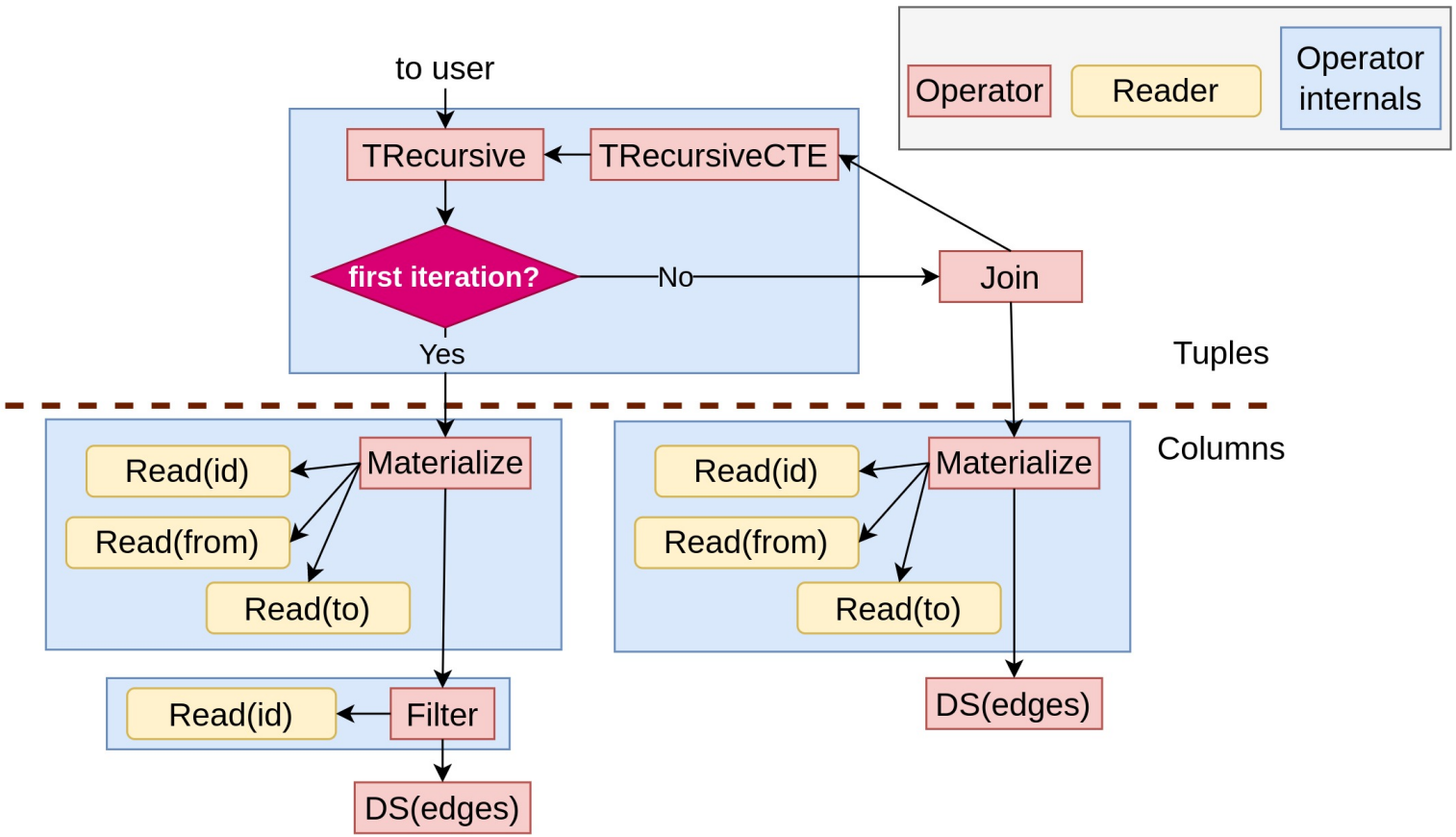
BFS calculation in SQL:

```
1  WITH RECURSIVE edges_cte (id, from_v, to_v, depth) AS
2       (SELECT edges.id, edges.from_v, edges.to_v, 0
3        FROM edges WHERE edges.from_v = startId
4        UNION ALL
5        SELECT edges.id, edges.from_v, edges.to_v,
6        e.depth + 1 FROM edges JOIN  edges_cte AS e
7        ON edges.from_v = e.to_v AND e.depth < maxDepth)
8  SELECT edges_cte.id, edges_cte.from_v, edges_cte.to_v
9  FROM edges_cte;
```

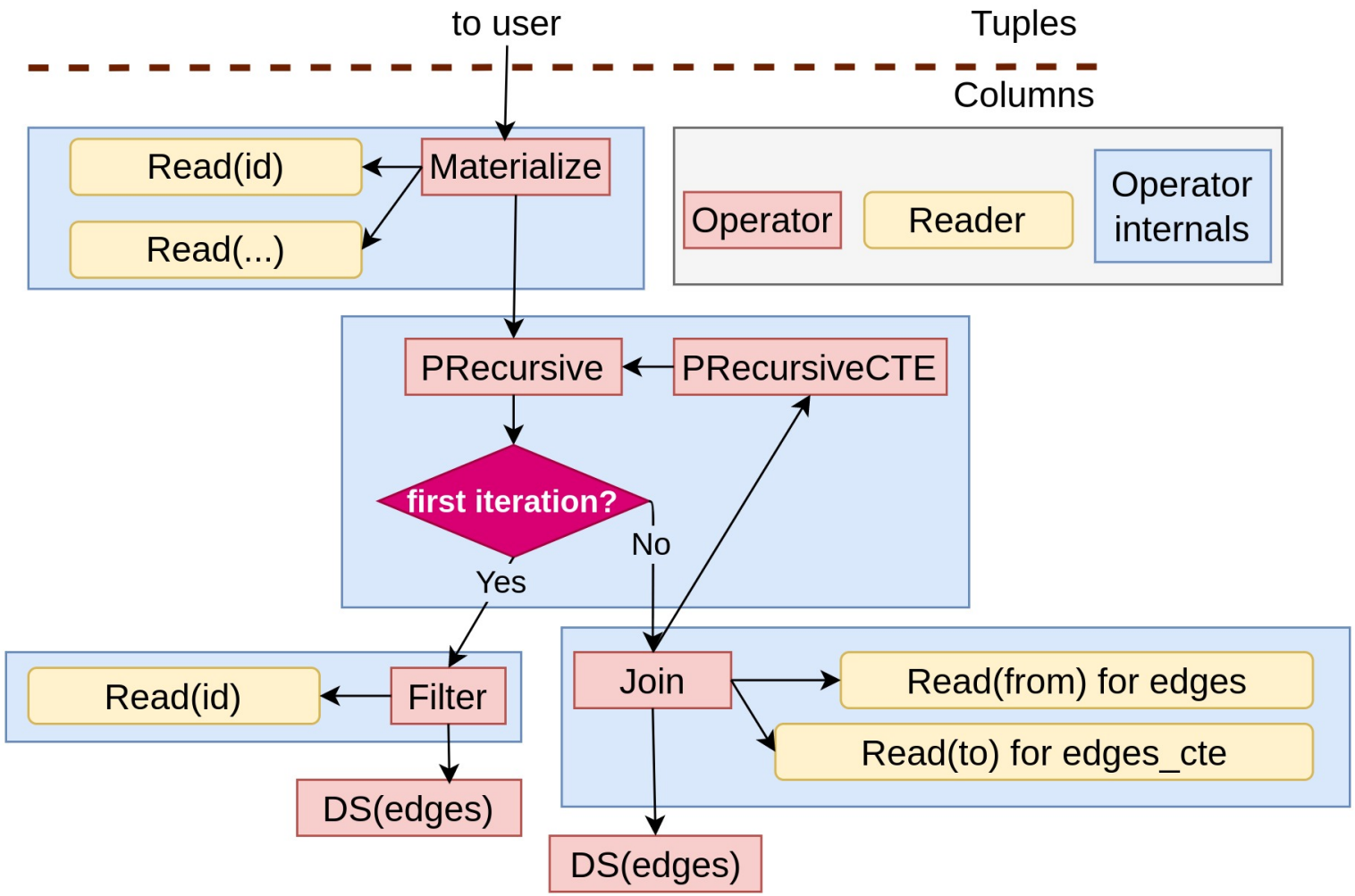
Equivalent Cypher query used for Neo4j:

```
1  MATCH (n:Node { id:startId })-[:OUTCOME *0..maxDepth]
2  ->(next: Node)
3  RETURN n.id, next.id;
```

Recursive Queries: TRecursive

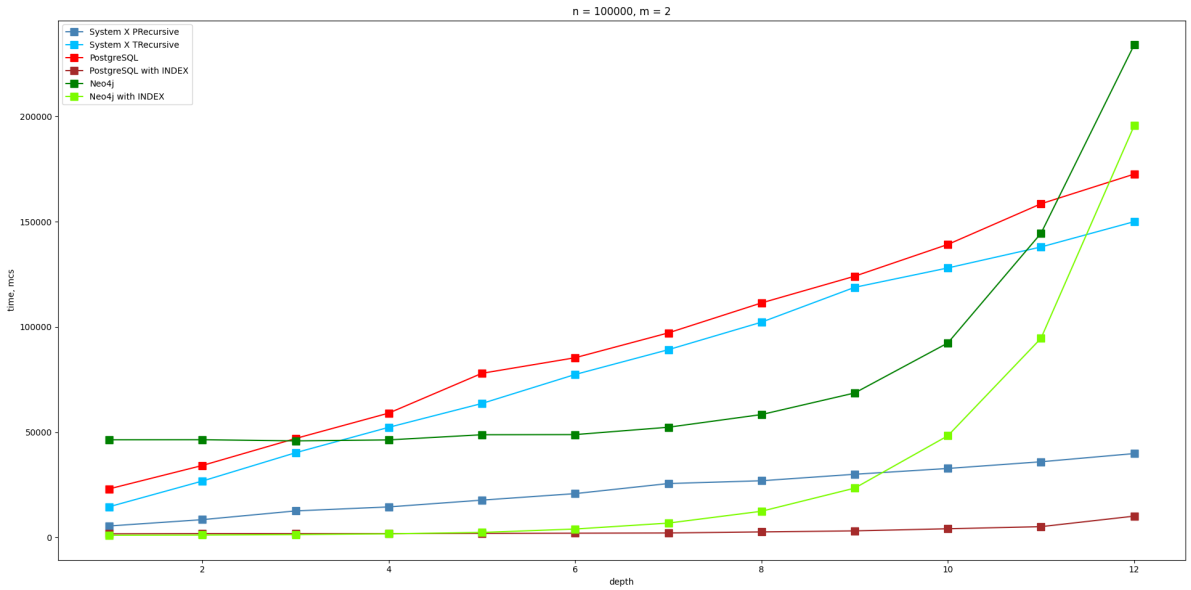


Recursive Queries: PRecursive

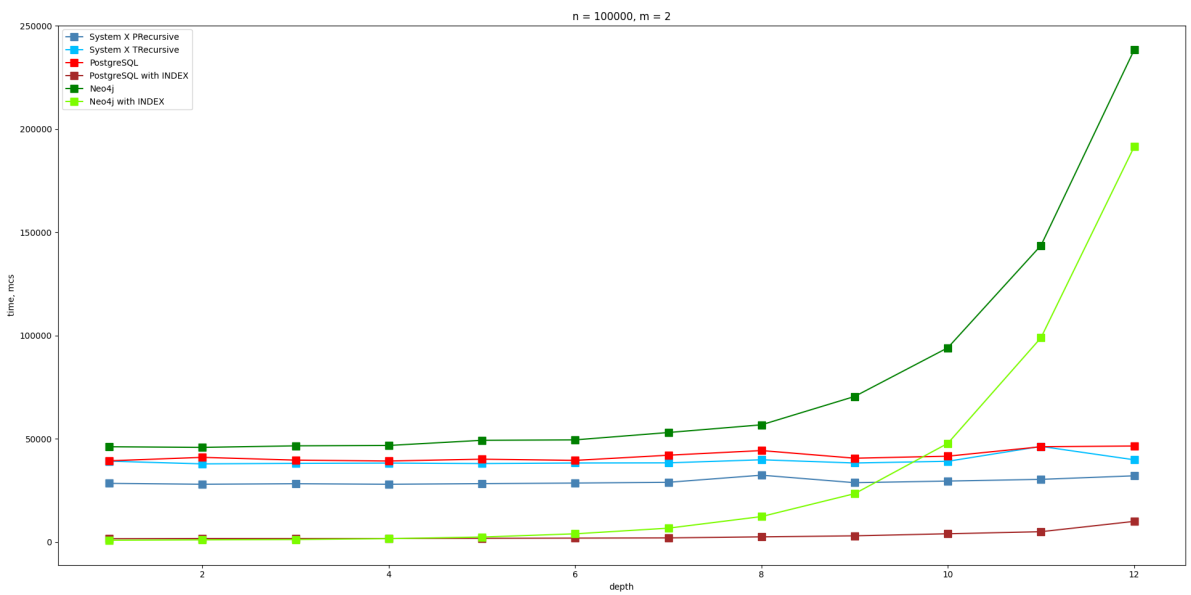


Recursive Queries: Experiment 1

BFS with the `CTE` hashed, first experiment set:

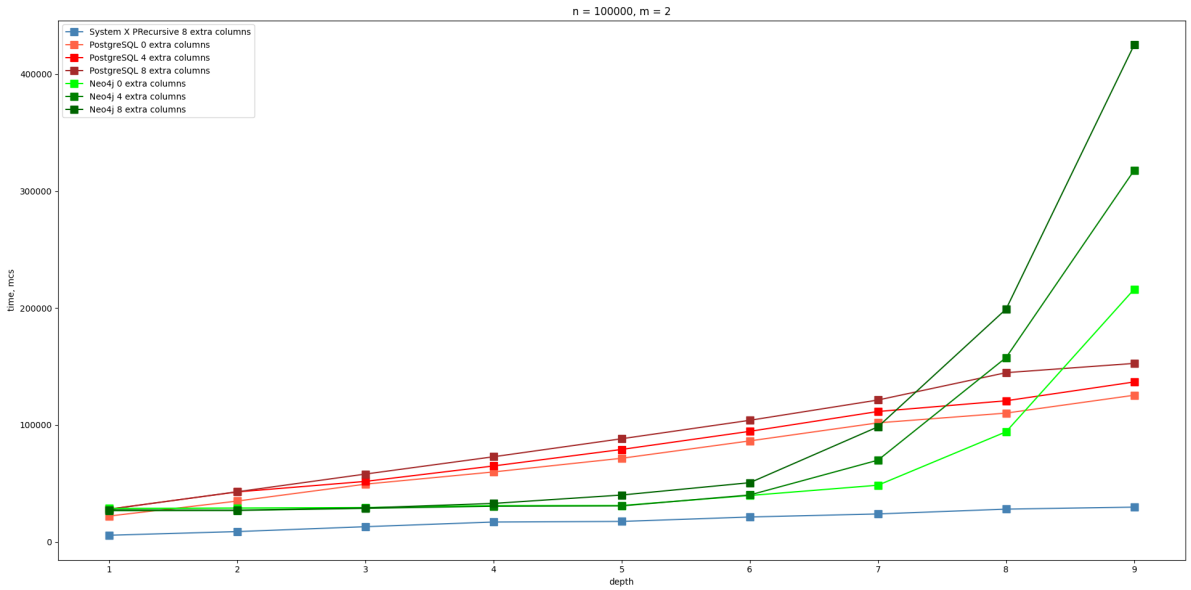


BFS with the `edges` hashed, first experiment set:

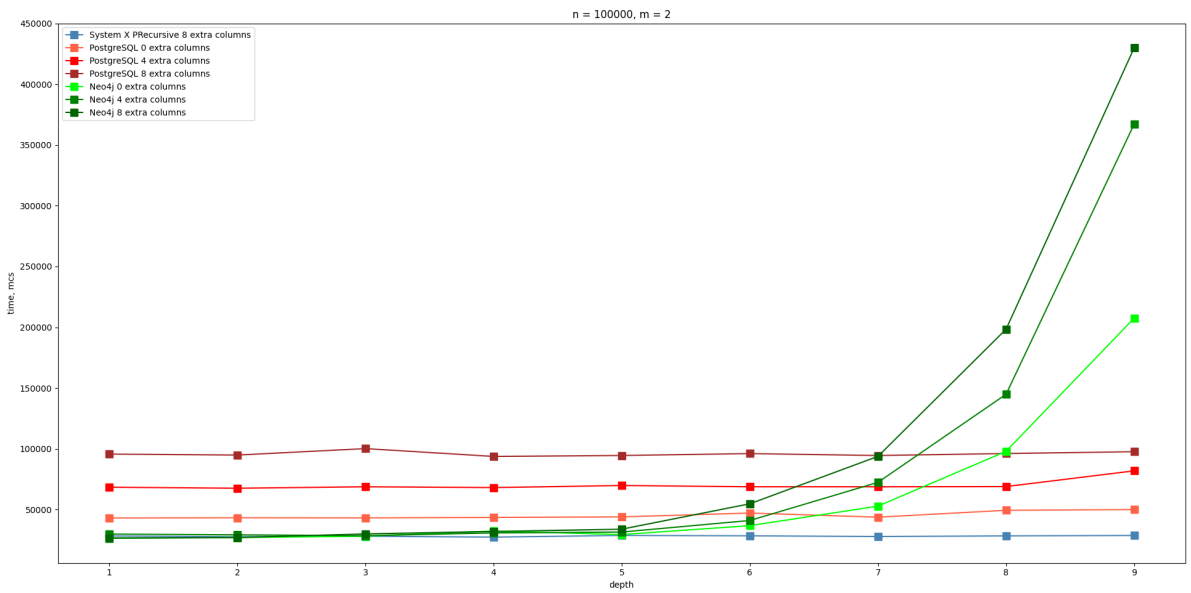


Recursive Queries: Experiment 2

BFS with the `CTE` hashed, first experiment set:



BFS with the `edges` hashed, first experiment set:



Technical: distributed processing

General idea

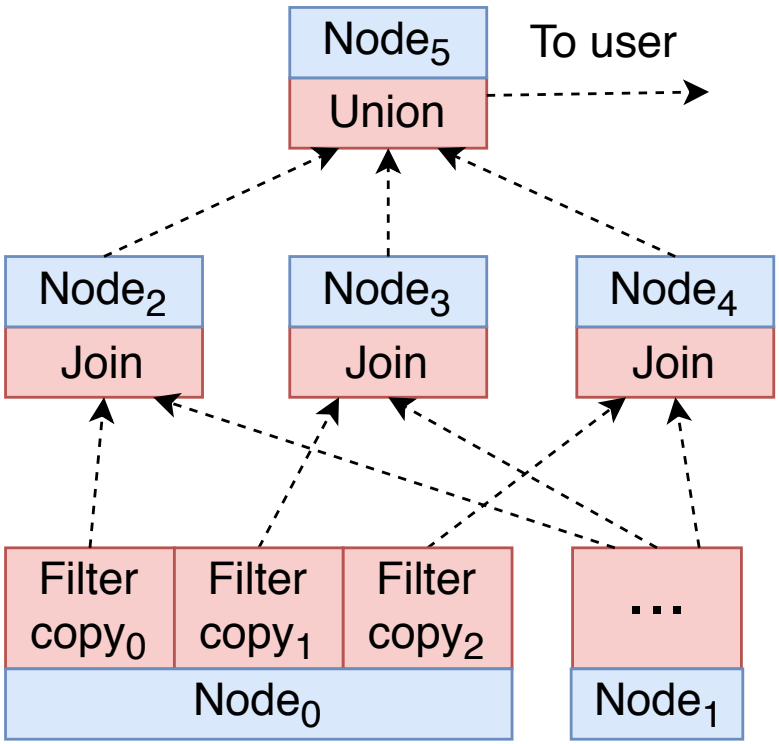
1. Appropriately distribute data
2. Compute local intermediates
3. Merge them to obtain total result

Problems

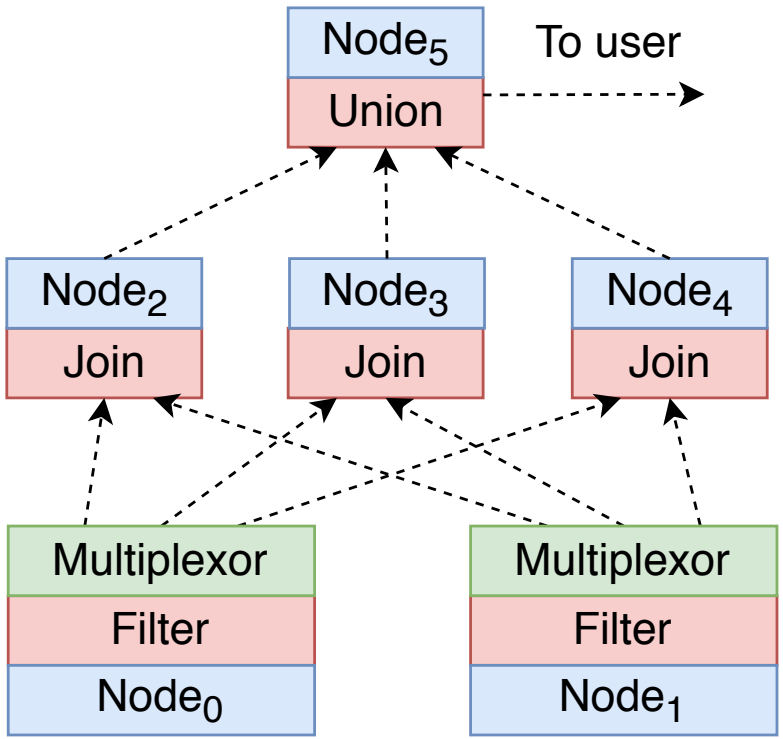
- Minimize network I/O
- Avoid DAG query model
- Provide flexible and efficient distributed data model

1. Distributed join: reshuffle, local join, union
2. Distributed aggregation: decompose, local preaggregate, combine

Distributed join: DAG?

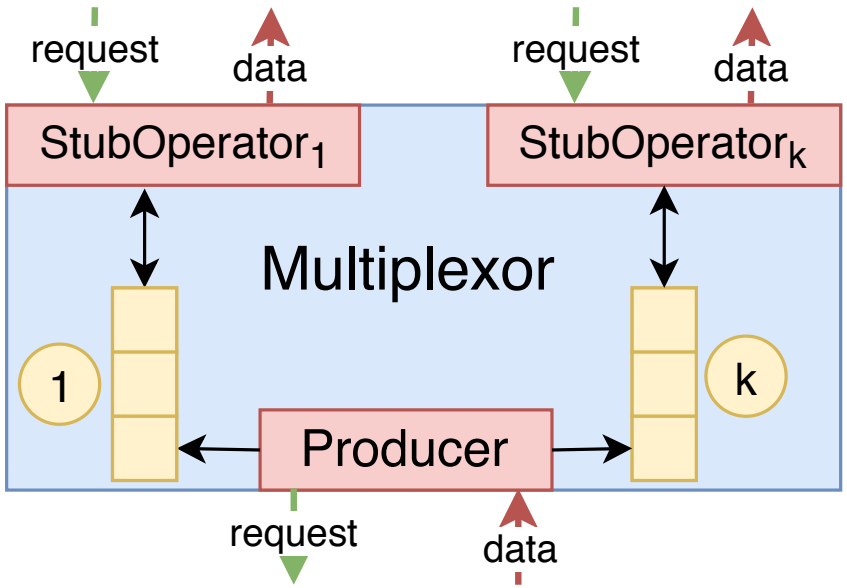


(a) Duplicate approach

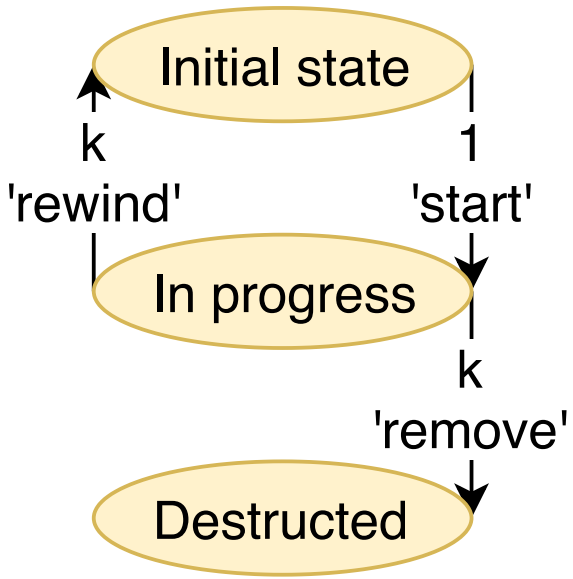


(b) Multiplexor approach

Multiplexor: 1 to k

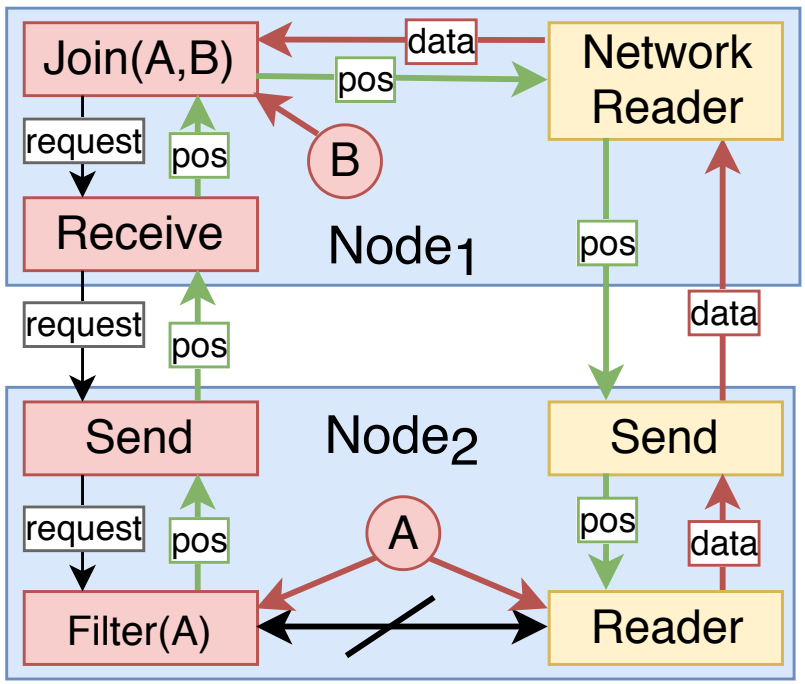


(a) Module architecture

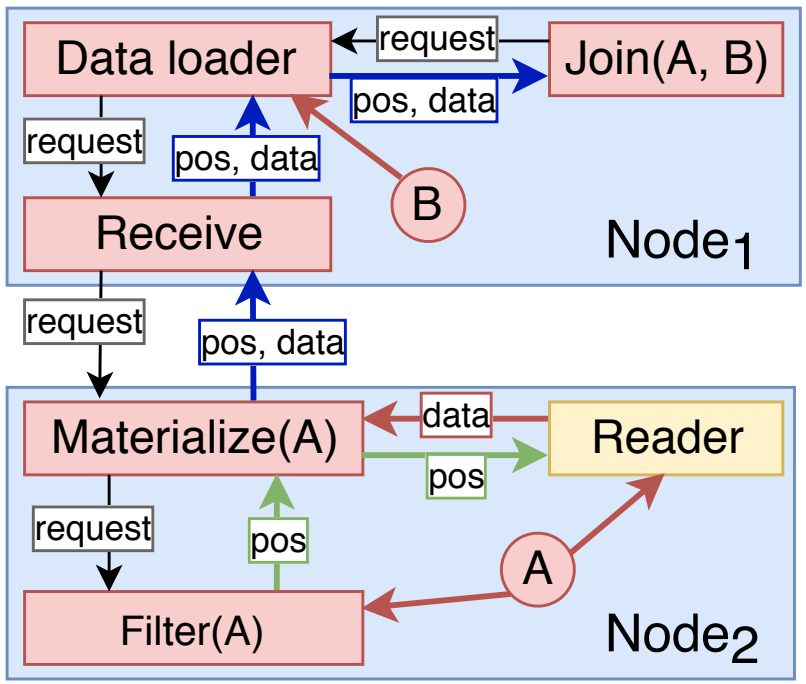


(b) State diagram

Hybrid materialization: optimize network I/O

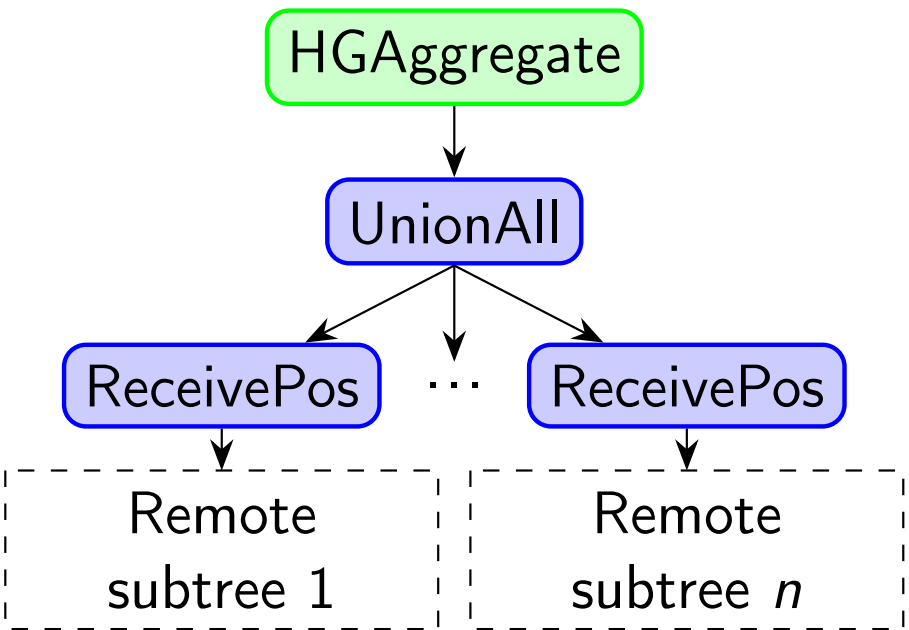


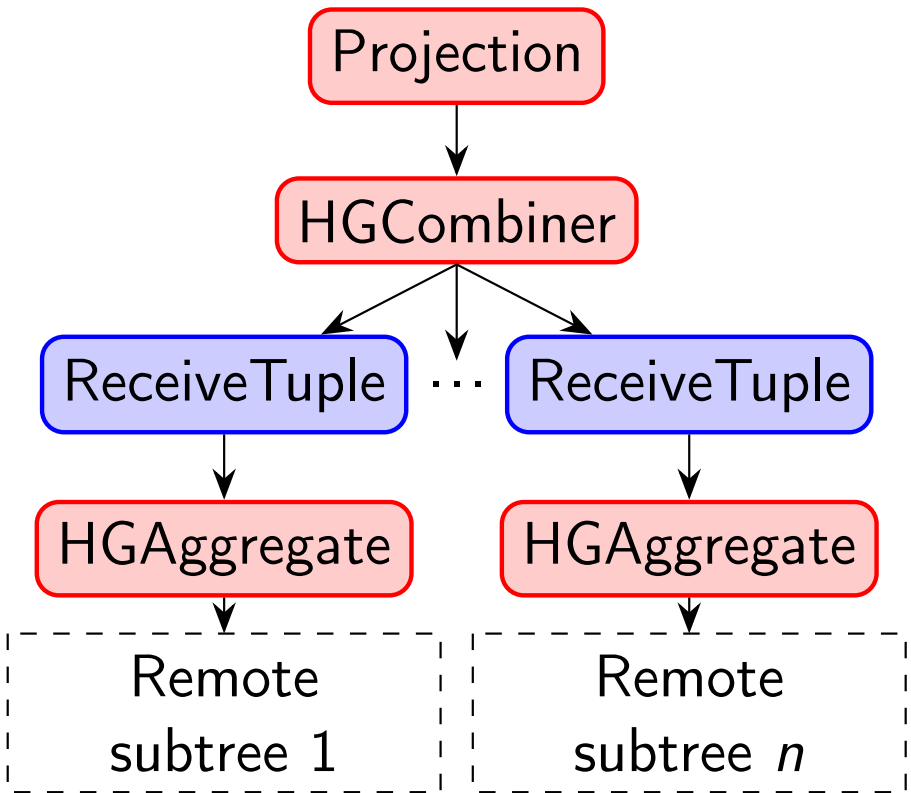
(a) Late materialization



(b) Hybrid materialization

Distributed aggregation: earlier and now





Definition (from "Efficient Evaluation of Aggregates on Bulk Types" [SG95])

A scalar aggregation function $f : \mathit{bulk}(\tau) \to \mathcal{N}$ is called *decomposable*, if there exist functions

$$\begin{aligned} \alpha : \mathit{bulk}(\tau) &\to \mathcal{N}' \\ \beta : \mathcal{N}', \mathcal{N}' &\to \mathcal{N}' \\ \gamma : \mathcal{N}' &\to \mathcal{N}, \end{aligned}$$

with

$$f(Z) = \gamma(\beta(\alpha(X), \alpha(Y)))$$

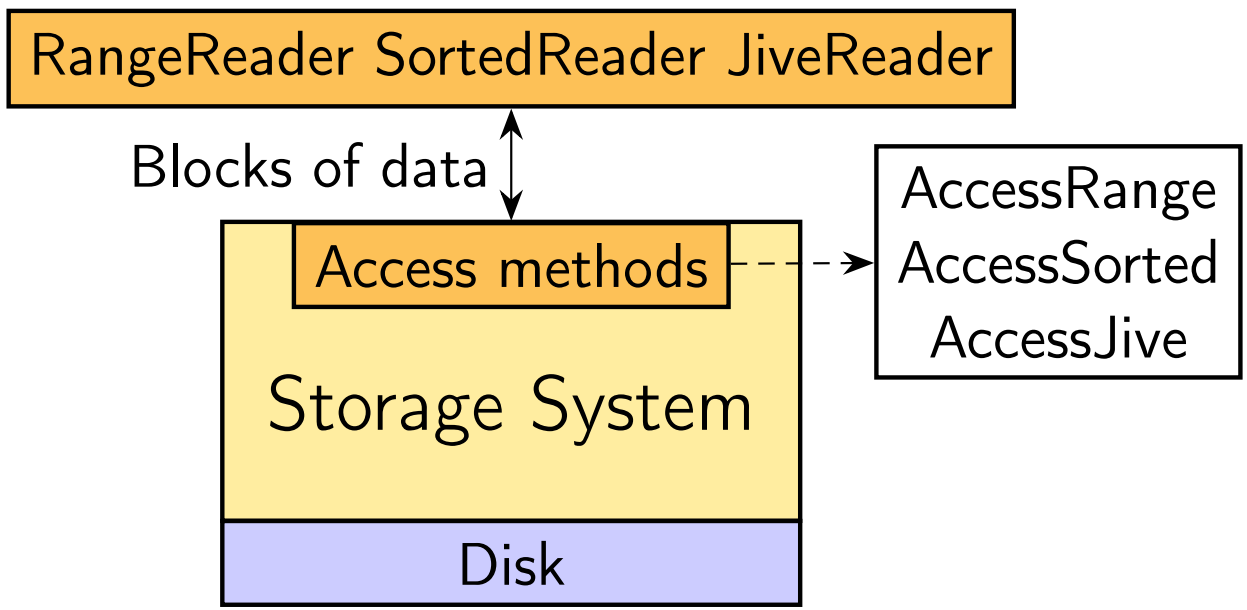
for all X, Y, and Z with $Z = X \cup Y$.

From algorithmic perspective α, β, and γ are phases of processing:

- α — preagreggation on remote nodes
- β — combining of preaggregation results from remote nodes
- γ — projection, final transformation

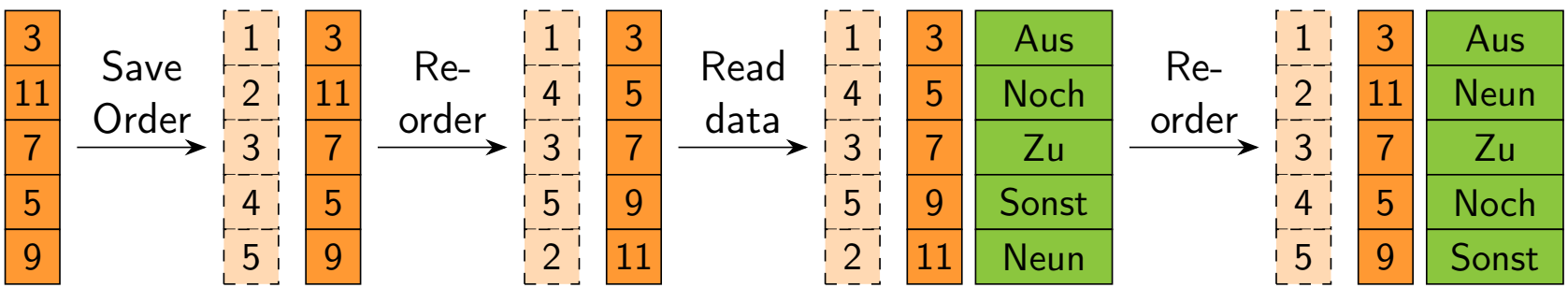
Technical: disk sub-system & buffer manager

- **Reader**: global strategy to get data for a position *stream*
- **Access method**: *local* access to data for a position *block*



For a position block access data *efficiently*:

1. contiguous position range (`AccessRange`): `for` loop, sequential pages
2. sparse sorted list of positions (`AccessSorted`): similar to `for` loop, page number always increasing (one-direction file read)
3. sparse unordered list of positions (`AccessJive`) – ?



Mixed stream of positions

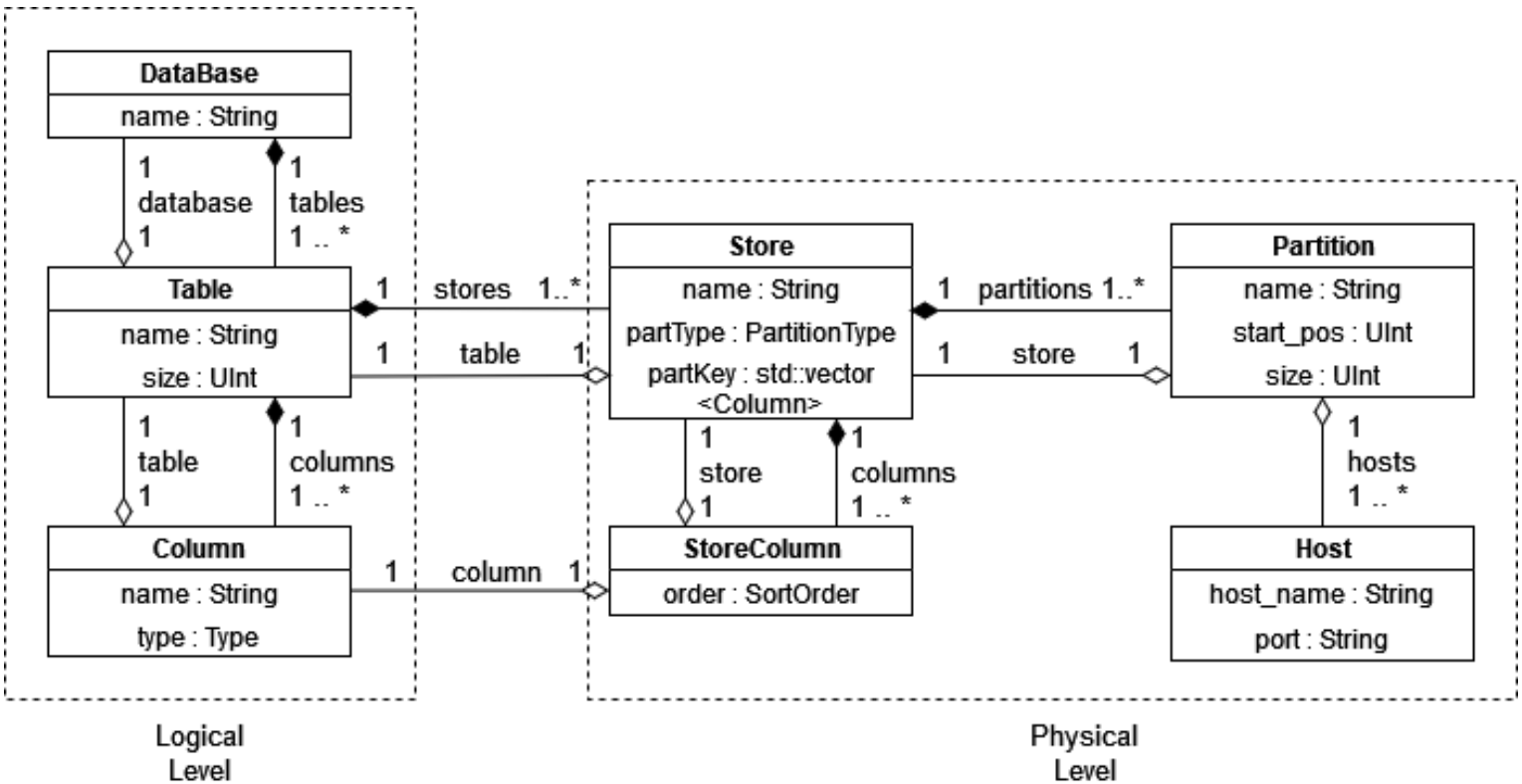
What if stream of positions is local but heterogeneous?

- Select an optimal access method for each individual block (dynamically)
- `AccessJive` worse `AccessSorted` worse `AccessRange`

```
class MixedReader {
    AccessRange   range;
    AccessSorted  sorted;
    AccessJive    jive;
    AccessMethod *current; // points to  range, sorted or jive
}
```

Technical: catalogue evolution

Switched from column-based partitioning to table-based one.
Reasons: hard to maintain, impossible to optimize.



Technical: parser and a simple plan generator

Parser and a simple plan generator

We have implemented parser and a simple plan generator. Results:

1. Making custom parser, not reusing PostgreSQL one, allowed us to see many interesting things:
   - You need query hypergraph, not query graph to describe join order in contemporary queries
   - There are many corner cases in PosDB, e.g. materialization operator without readers?
   - You need a fake table for running SELECT 1+1; and many systems do it this way :)
   - In our code base, predicates not bound to any table, require (!) query rewriter e.g. WHERE $1 > 2$
   - ...
2. Plan generator that currently covers three basic strategies (PosDB'21)
3. As the result, we have a demo web site, where you can run queries and browse query plans: `https://pos-db.com/`

Conclusion and future work

Conclusion

We have come a long way from a toy engine, used to teach students, to a larger research prototype, allowing to perform serious studies. Furthermore, PosDB is close to be usable in production.

Future work

- Extending class of supported queries, move towards TPC-H and TPC-DS support — subqueries, first of all;
- Indexes;
- Deferred evaluation of generated attributes;
- Memory management subsystem;
- Query optimizer;
- More technical stuff: REPL, visualization, ...
- ...

References I

Daniel Abadi, Peter Boncz, and Stavros Harizopoulos.
The Design and Implementation of Modern Column-Oriented Database Systems.

Now Publishers Inc., Hanover, MA, USA, 2013.

Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel Madden.
Materialization strategies in a column-oriented dbms.
In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, ICDE, pages 466–475. IEEE, 2007.

Peter A. Boncz and Martin L. Kersten.
Mil primitives for querying a fragmented world.
The VLDB Journal, 8(2):101–119, October 1999.

Nikita Bobrov, Kirill Smirnov, and George A. Chernishev.
Extending databases to support data manipulation with functional dependencies: a vision paper.
CoRR, abs/2005.07992, 2020.

References II

George A. Chernishev, Viacheslav Galaktionov, Valentin D. Grigorev, Evgeniy Klyuchikov, and Kirill Smirnov.
PosDB: A distributed column-store engine.
In Alexander K. Petrenko and Andrei Voronkov, editors, Perspectives of System Informatics - 11th International Andrei P. Ershov Informatics Conference, PSI 2017, Moscow, Russia, June 27-29, 2017, Revised Selected Papers, volume 10742 of Lecture Notes in Computer Science, pages 88–94. Springer, 2017.

George A. Chernishev, Vyacheslav Galaktionov, Valentin D. Grigorev, Evgeniy Klyuchikov, and Kirill Smirnov.
PosDB: An architecture overview.
Program. Comput. Softw., 44(1):62–74, 2018.

George A. Chernishev, Viacheslav Galaktionov, Valentin V. Grigorev, Evgeniy Klyuchikov, and Kirill Smirnov.
A comprehensive study of late materialization strategies for a disk-based column-store.
In Kostas Stefanidis and Lukasz Golab, editors, Proceedings of the 24th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with the 25th International Conference on Extending Database Technology and the 25th International Conference on Database Theory (EDBT/ICDT 2022), Edinburgh, UK, March

References III

29, 2022, volume 3130 of CEUR Workshop Proceedings, pages 21–30. CEUR-WS.org, 2022.

George A. Chernishev.
Making dbmses dependency-aware.
In 10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings. www.cidrdb.org, 2020.

George P. Copeland and Setrag N. Khoshafian.
A decomposition storage model.
SIGMOD Rec., 14(4):268–279, 1985.

Viacheslav Galaktionov, Evgeniy Klyuchikov, and George A. Chernishev.
Position caching in a column-store with late materialization: An initial study.
In Il-Yeol Song, Katja Hose, and Oscar Romero, editors, Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT 2020 Joint Conference, DOLAP@EDBT/ICDT 2020, Copenhagen, Denmark, March 30, 2020, volume 2572 of CEUR Workshop Proceedings, pages 89–93. CEUR-WS.org, 2020.

References IV

Martin Grund, Jens Krueger, Matthias Kleine, Alexander Zeier, and Hasso Plattner.
Optimal query operator materialization strategy for hybrid databases.
In Proceedings of the 2011 Third International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA '11, pages 169–174. IARIA, 2011.

Goetz Graefe.
Query Evaluation Techniques for Large Databases.
ACM Comput. Surv., 25(2):73–169, June 1993.

Stavros Harizopoulos, Daniel Abadi, and Peter Boncz.
Column-oriented database systems, vldb 2009 tutorial., 2009.

Stratos Idreos, Fabian Groffen, Niels Nes, Stefan Manegold, K. Sjoerd Mullender, and Martin L. Kersten.
MonetDB: Two Decades of Research in Column-oriented Database Architectures.
IEEE Data Eng. Bull., 35(1):40–45, 2012.

References V

Setrag Khoshafian, George P. Copeland, Thomas Jagodis, Haran Boral, and Patrick Valduriez.
A query processing strategy for the decomposed storage model.
In Proceedings of the Third International Conference on Data Engineering, pages 636–643, Washington, DC, USA, 1987. IEEE Computer Society.

Nadezhda Mukhaleva, Valentin D. Grigorev, and George A. Chernishev.
Implementing window functions in a column-store with late materialization.
In Klaus-Dieter Schewe and Neeraj Kumar Singh, editors, Model and Data Engineering - 9th International Conference, MEDI 2019, Toulouse, France, October 28-31, 2019, Proceedings, volume 11815 of Lecture Notes in Computer Science, pages 303–313. Springer, 2019.

Michael Polyntsov, Valentin Grigorev, Kirill Smirnov, and George Chernishev.
Implementing the comparison-based external sort.
In Silvia Chiusano, Tania Cerquitelli, Robert Wrembel, Kjetil Nørvåg, Barbara Catania, Genoveva Vargas-Solar, and Ester Zumpano, editors, New Trends in Database and Information Systems, pages 500–511, Cham, 2022. Springer International Publishing.

References VI

Fotis Psallidas and Eugene Wu.
Demonstration of smoke: A deep breath of data-intensive lineage applications.
In Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein, editors, Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, pages 1781–1784. ACM, 2018.

Fotis Psallidas and Eugene Wu.
Smoke: Fine-grained lineage at interactive speed.
Proc. VLDB Endow., 11(6):719–732, 2018.

Ravishankar Ramamurthy, David J. DeWitt, and Qi Su.
A case for fractured mirrors.
In Proceedings of the 28th international conference on Very Large Data Bases, VLDB '02, pages 430–441. VLDB Endowment, 2002.

Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik.
C-store: A column-oriented dbms.
In Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05, pages 553–564. VLDB Endowment, 2005.

References VII


Cluet Sophie and Moerkotte Guido.
Efficient evaluation of aggregates on bulk types.
In Proceedings of the Fifth International Workshop on Database Programming Languages, page 8, 1995.


Alexander Slesarev, Evgeniy Klyuchikov, Kirill Smirnov, and George A. Chernishev.
Revisiting data compression in column-stores.
In J. Christian Attiogbé and Sadok Ben Yahia, editors, Model and Data Engineering - 10th International Conference, MEDI 2021, Tallinn, Estonia, June 21-23, 2021, Proceedings, volume 12732 of Lecture Notes in Computer Science, pages 279–292. Springer, 2021.


P. E. O'Neil, E. J. O'Neil and X. Chen. The Star Schema Benchmark (SSB).
`http://www.cs.umb.edu/~poneil/StarSchemaB.PDF`, 2009.
Accessed: 10/09/2017.


Dimitris Tsirogiannis, Stavros Harizopoulos, Mehul A. Shah, Janet L. Wiener, and Goetz Graefe.
Query Processing Techniques for Solid State Drives.
In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, SIGMOD '09, pages 59–72, New York, NY, USA, 2009. ACM.

References VIII

Patrick Valduriez.
Join indices.
ACM Trans. Database Syst., 12(2):218–246, June 1987.

Eugene Wu.
Systems for human data interaction (keynote).
In Davide Mottin, Matteo Lissandrini, Senjuti Basu Roy, and Yannis Velegrakis, editors, Proceedings of the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA-Data 2021) co-located with 47th International Conference on Very Large Data Bases (VLDB 2021), Copenhagen, Denmark, August 20, 2021, 2021.