🪂 Parachute: Single-Pass Bi-Directional Information Passing

Mihail Stoian, Andreas Zimmerer, Skander Krid,
Amadou Ngom, Jialin Ding, Tim Kraska, Andreas Kipf

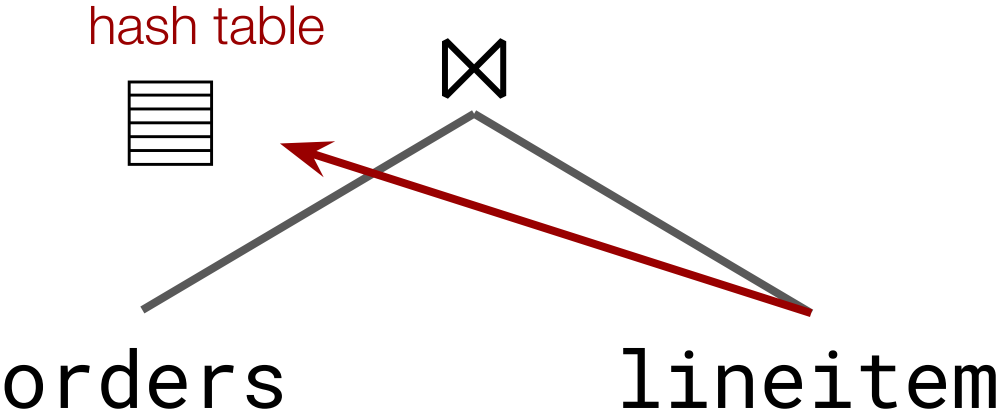
Data Systems Lab @UTN x Data Systems Group @MIT

Databases for OLAP @VLDB’25

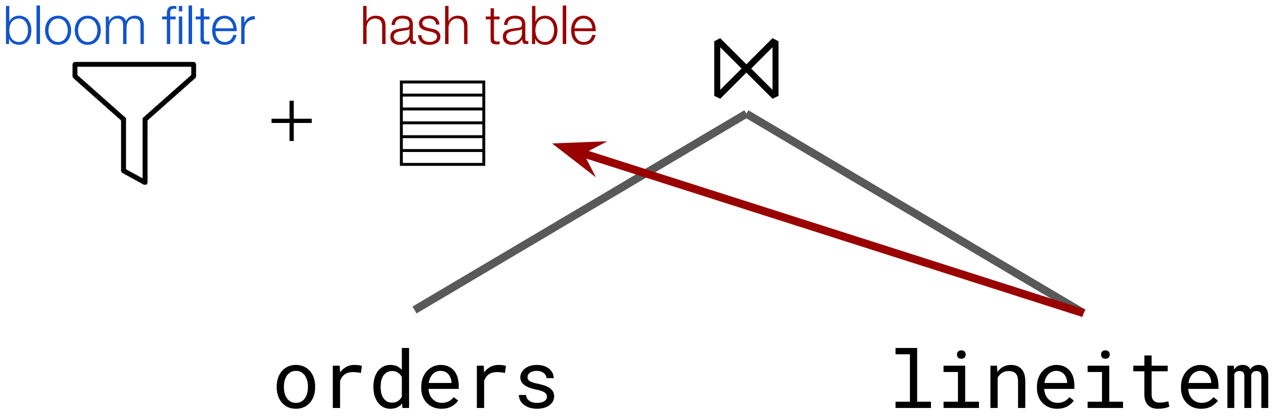
UTN Technische Universität Nürnberg

Massachusetts Institute of Technology

Semi-Join Filtering



Semi-Join Filtering



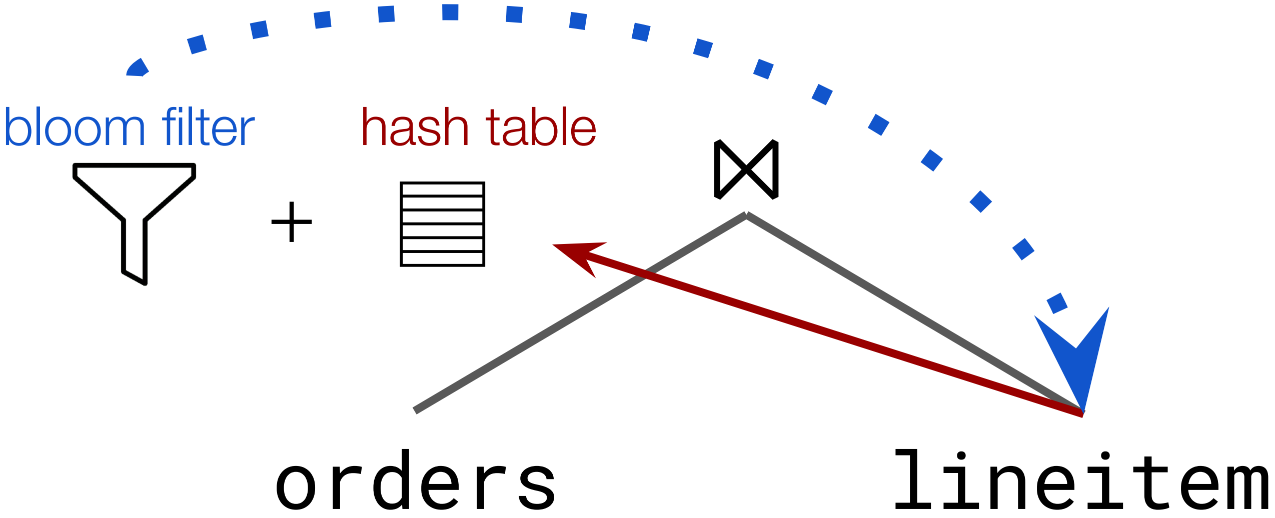
bloom filter + hash table

$\bowtie$

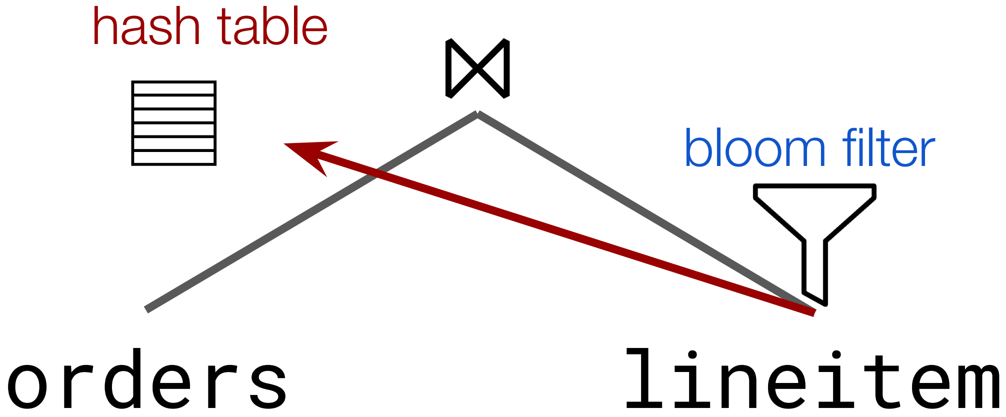
orders

lineitem

Semi-Join Filtering



Semi-Join Filtering



Semi-Join Filtering

- Widely used in production systems, e.g., Amazon Redshift.
- **Goal**: Reduce *expensive* hash table probes.
- Alternatively: Reduce the #***dangling*** tuples → instance-optimal algorithms.

Semi-Join Filtering

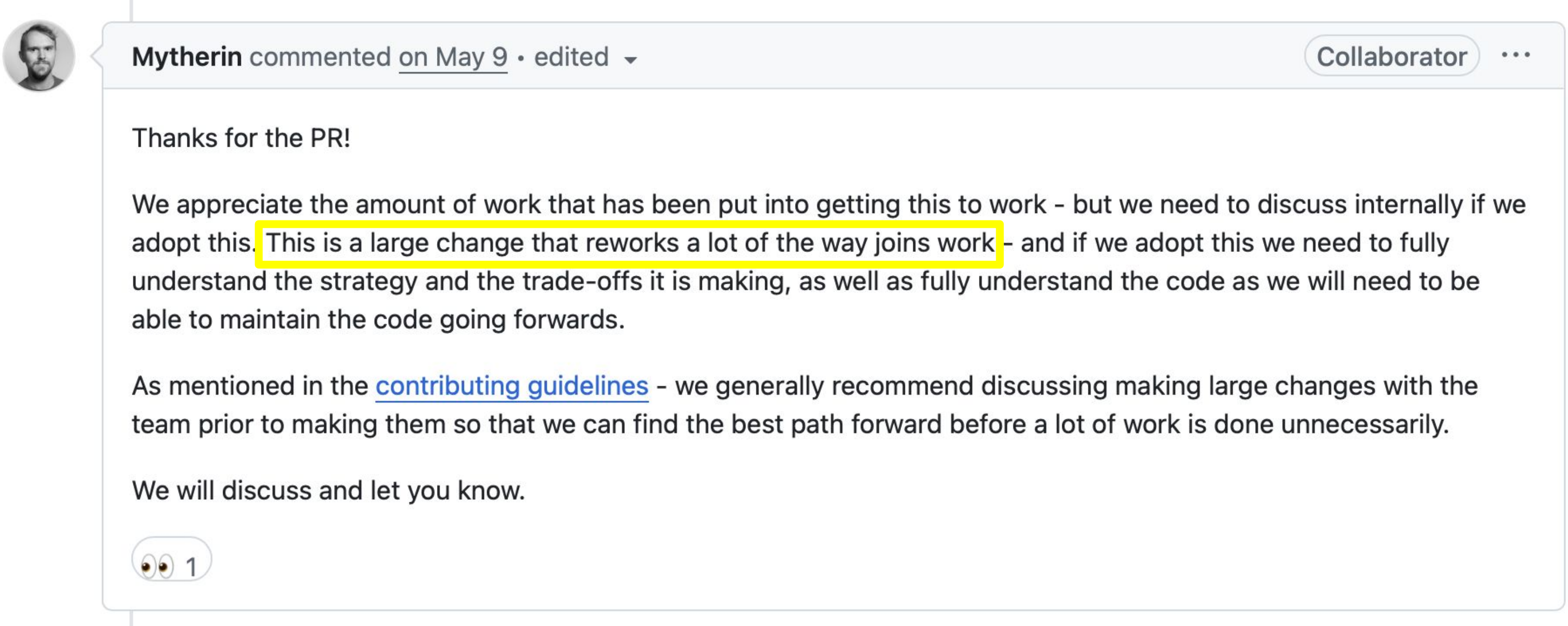
- Widely used in production systems, e.g., Amazon Redshift.
- **Goal**: Reduce *expensive* hash table probes.
- Alternatively: Reduce the #***dangling*** tuples → instance-optimal algorithms.
- Recent research makes instance-optimal algorithms practical:
    - Yannakakis [VLDB'81],
    - Predicate Transfer* [CIDR'24],
    - Lookup & Expand [VLDB'24],
    - Yannakakis++ [SIGMOD'25],
    - Robust Predicate Transfer (RPT) [SIGMOD'25].

Semi-Join Filtering

- Widely used in production systems, e.g., Amazon Redshift.
- **Goal**: Reduce *expensive* hash table probes.
- Alternatively: Reduce the #***dangling*** tuples → instance-optimal algorithms.
- Recent research makes instance-optimal algorithms practical:
    - Yannakakis [VLDB’81],
    - Predicate Transfer* [CIDR’24],
    - Lookup & Expand [VLDB’24],
    - Yannakakis++ [SIGMOD’25],
    - *Robust Predicate Transfer (RPT) [SIGMOD’25].*

So..

- RPT’s pull request (https://github.com/duckdb/duckdb/pull/17326).



Mytherin commented on May 9 • edited ▾ Collaborator ···

Thanks for the PR!

We appreciate the amount of work that has been put into getting this to work - but we need to discuss internally if we adopt this. This is a large change that reworks a lot of the way joins work - and if we adopt this we need to fully understand the strategy and the trade-offs it is making, as well as fully understand the code as we will need to be able to maintain the code going forwards.

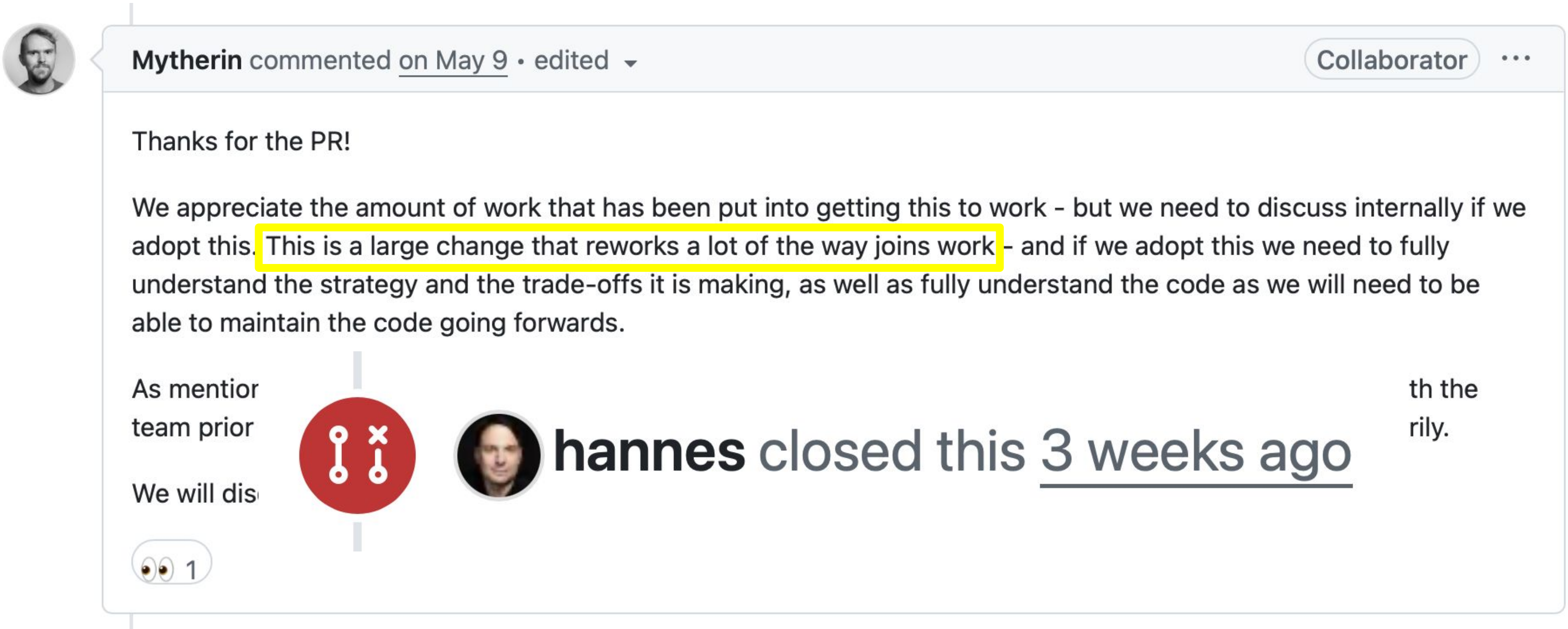
As mentioned in the contributing guidelines - we generally recommend discussing making large changes with the team prior to making them so that we can find the best path forward before a lot of work is done unnecessarily.

We will discuss and let you know.

👀 1

So..

- RPT’s pull request (https://github.com/duckdb/duckdb/pull/17326).



Mytherin commented on May 9 • edited ▾ Collaborator ···

Thanks for the PR!

We appreciate the amount of work that has been put into getting this to work - but we need to discuss internally if we adopt this. This is a large change that reworks a lot of the way joins work - and if we adopt this we need to fully understand the strategy and the trade-offs it is making, as well as fully understand the code as we will need to be able to maintain the code going forwards.

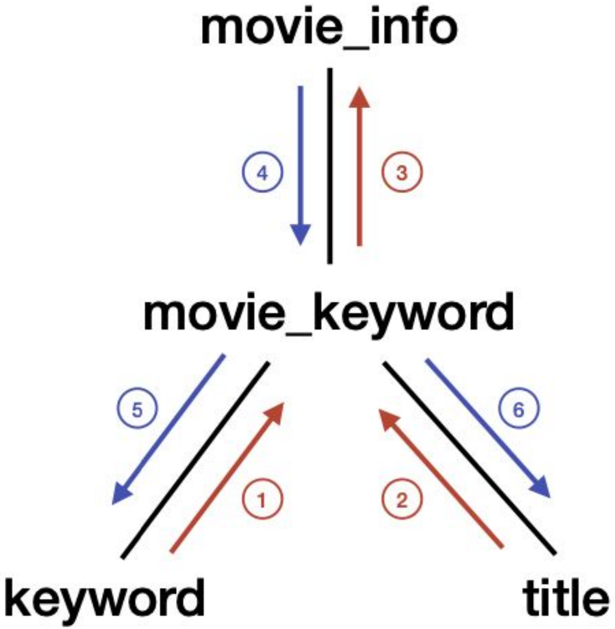
As mentior th the
team prior rily.

We will dis

👀 1

hannes closed this 3 weeks ago

But Why?



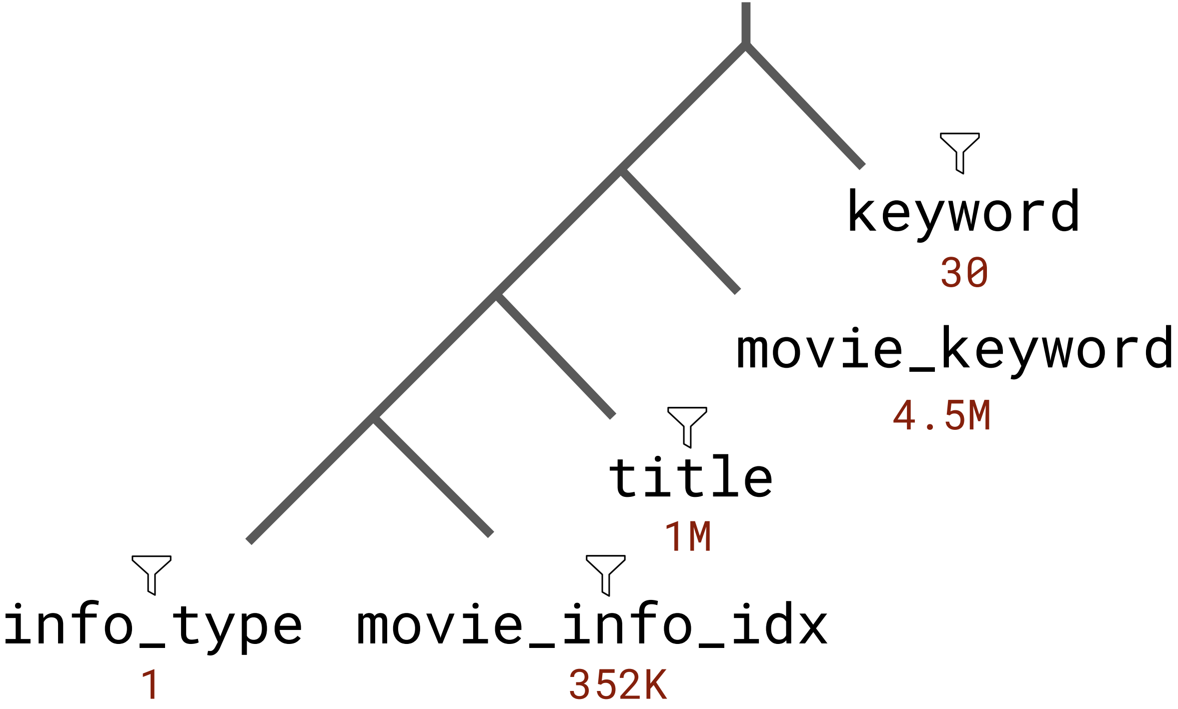
Forward Pass

1. movie_keyword $\ltimes$ keyword
2. movie_keyword $\ltimes$ title
3. movie_info $\ltimes$ movie_keyword

Backward Pass

4. movie_keyword $\ltimes$ movie_info
5. keyword $\ltimes$ movie_keyword
6. title $\ltimes$ movie_keyword

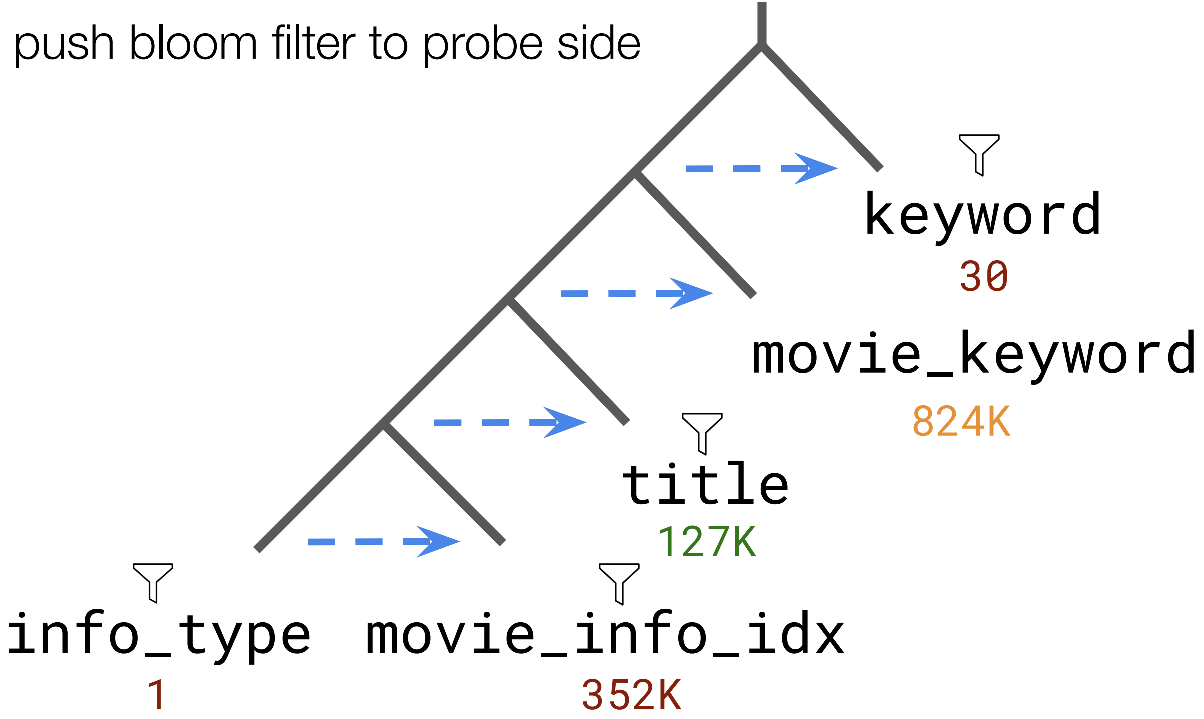
Semi-Join Filtering: Why We Love It



duckdb: 0.32s

Semi-Join Filtering: Why We Love It

⇢ = push bloom filter to probe side



keyword
30

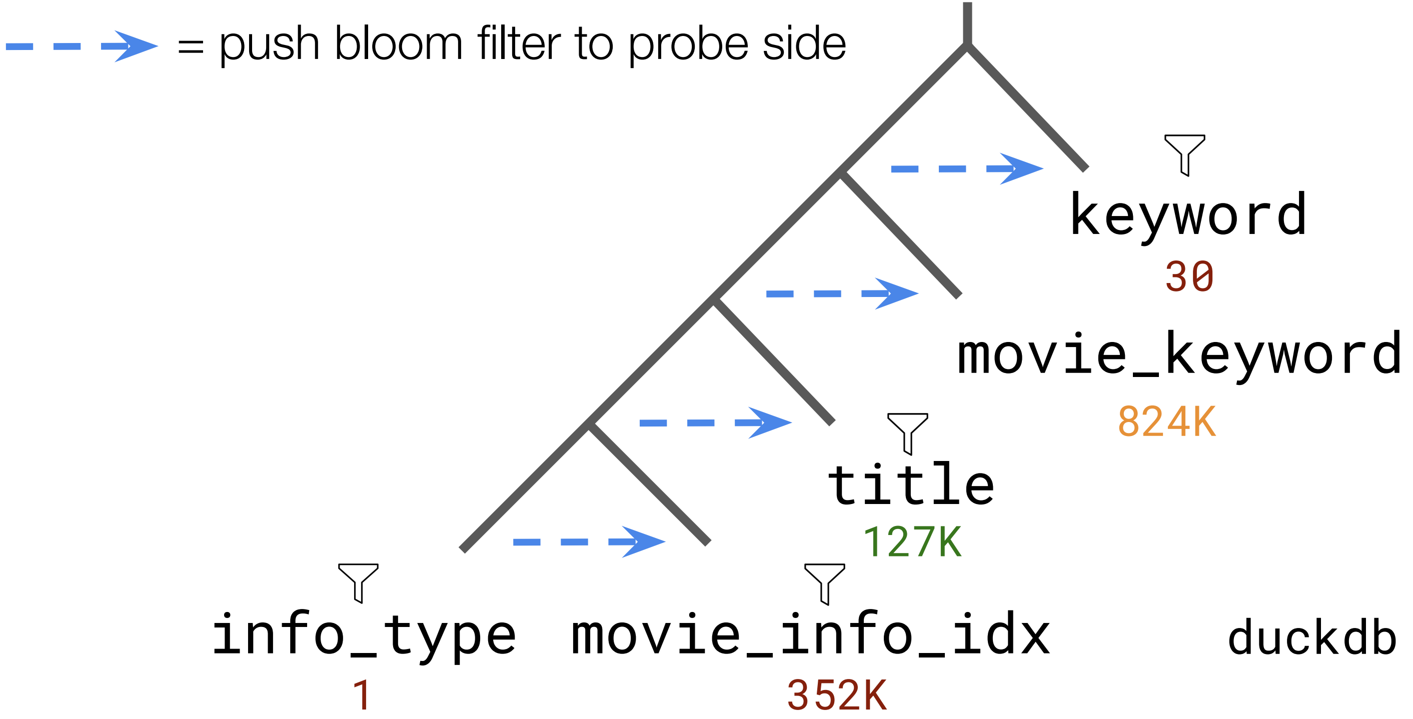
movie_keyword
824K

title
127K

info_type
1

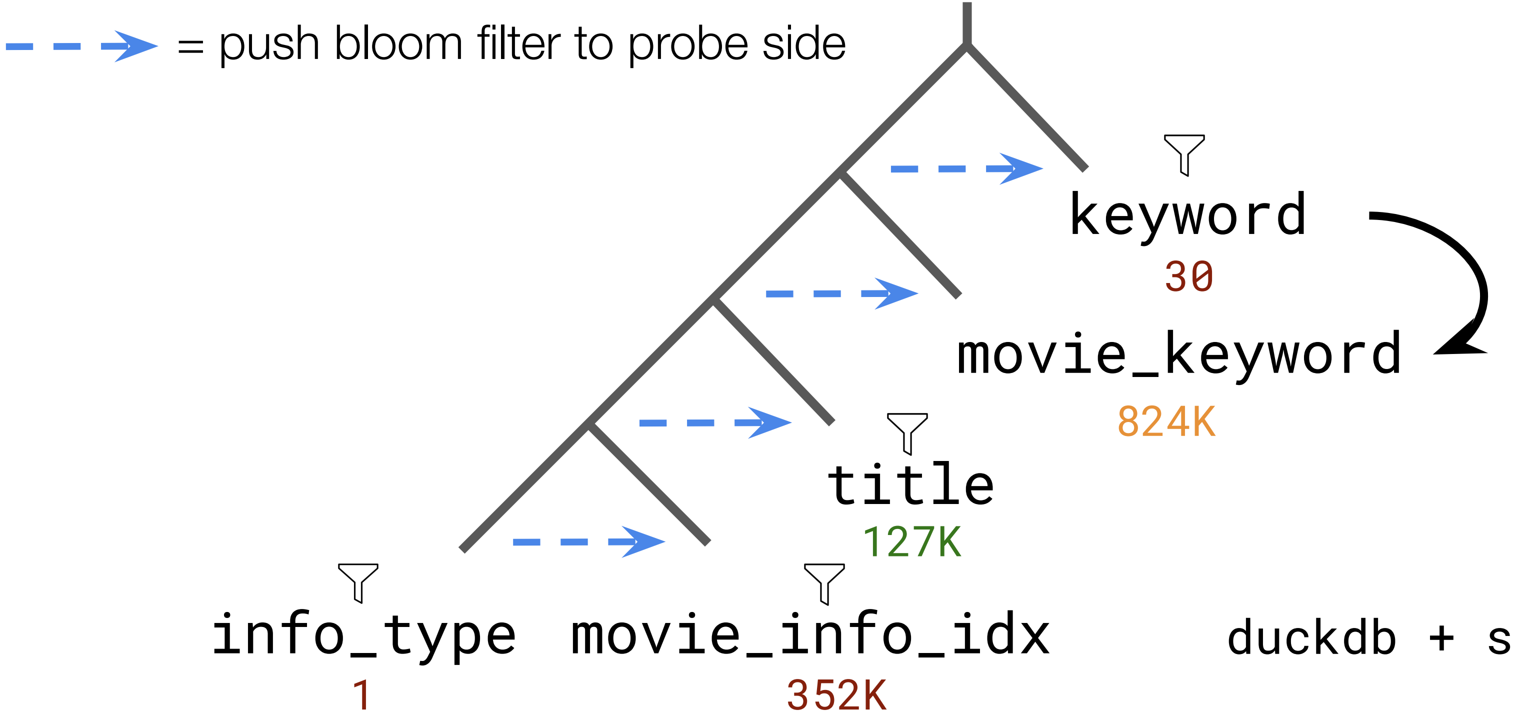
movie_info_idx
352K

Semi-Join Filtering: Why We Love It



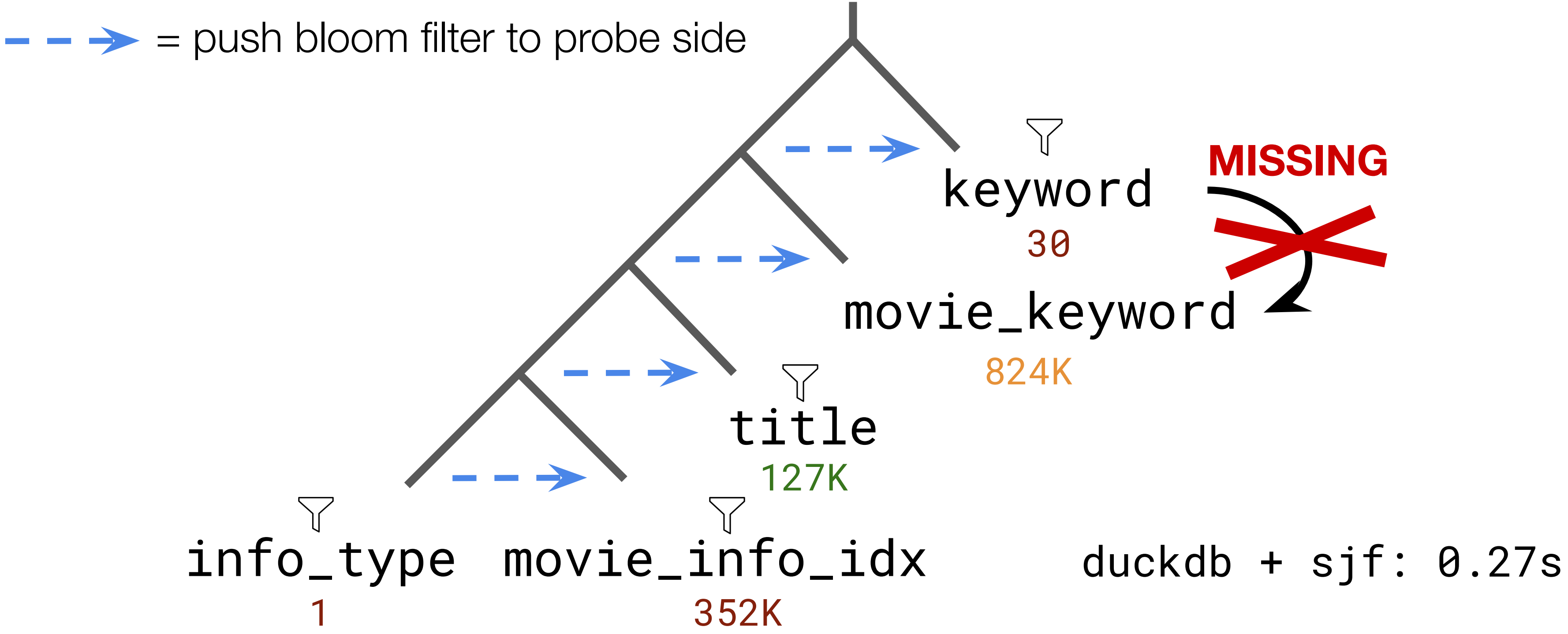
duckdb + sjf: 0.27s

Semi-Join Filtering: *The Missing Bit*



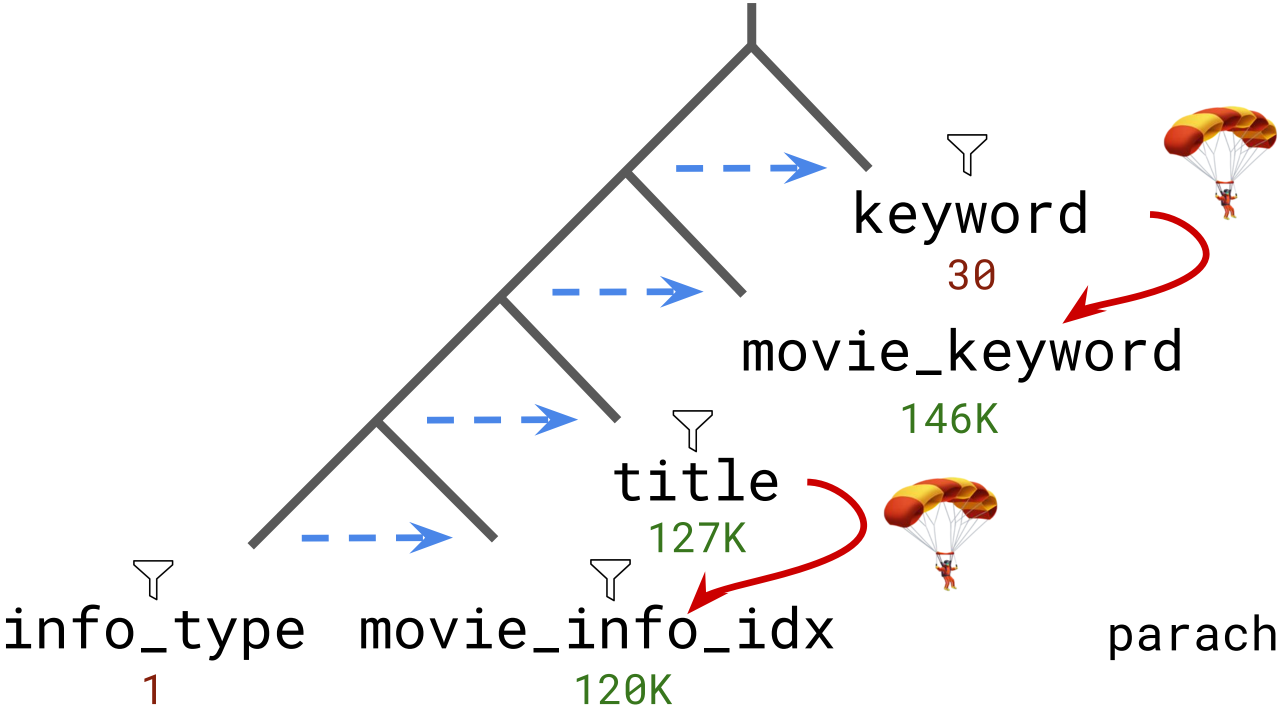
duckdb + sjf: 0.27s

Semi-Join Filtering: *The Missing Bit*



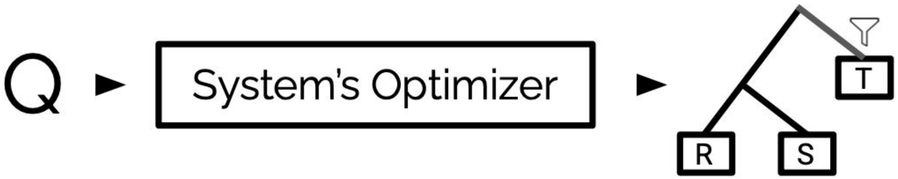
duckdb + sjf: 0.27s

🪂 Parachute: *Bi-Directional* Information Passing

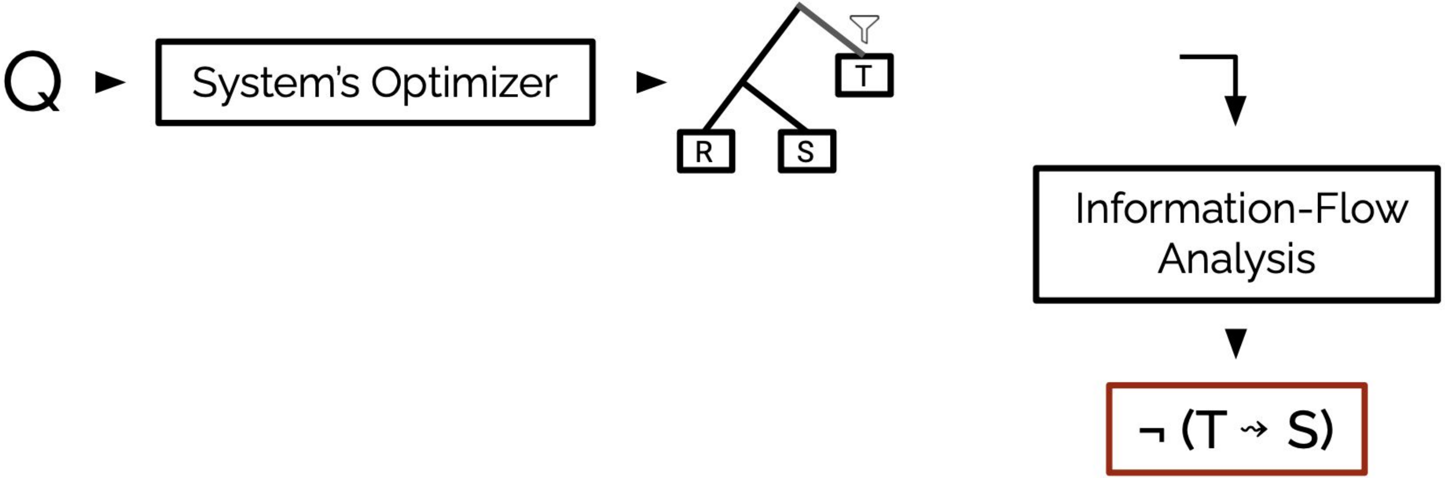


parachute: 0.21s

🪂 Parachute @Runtime

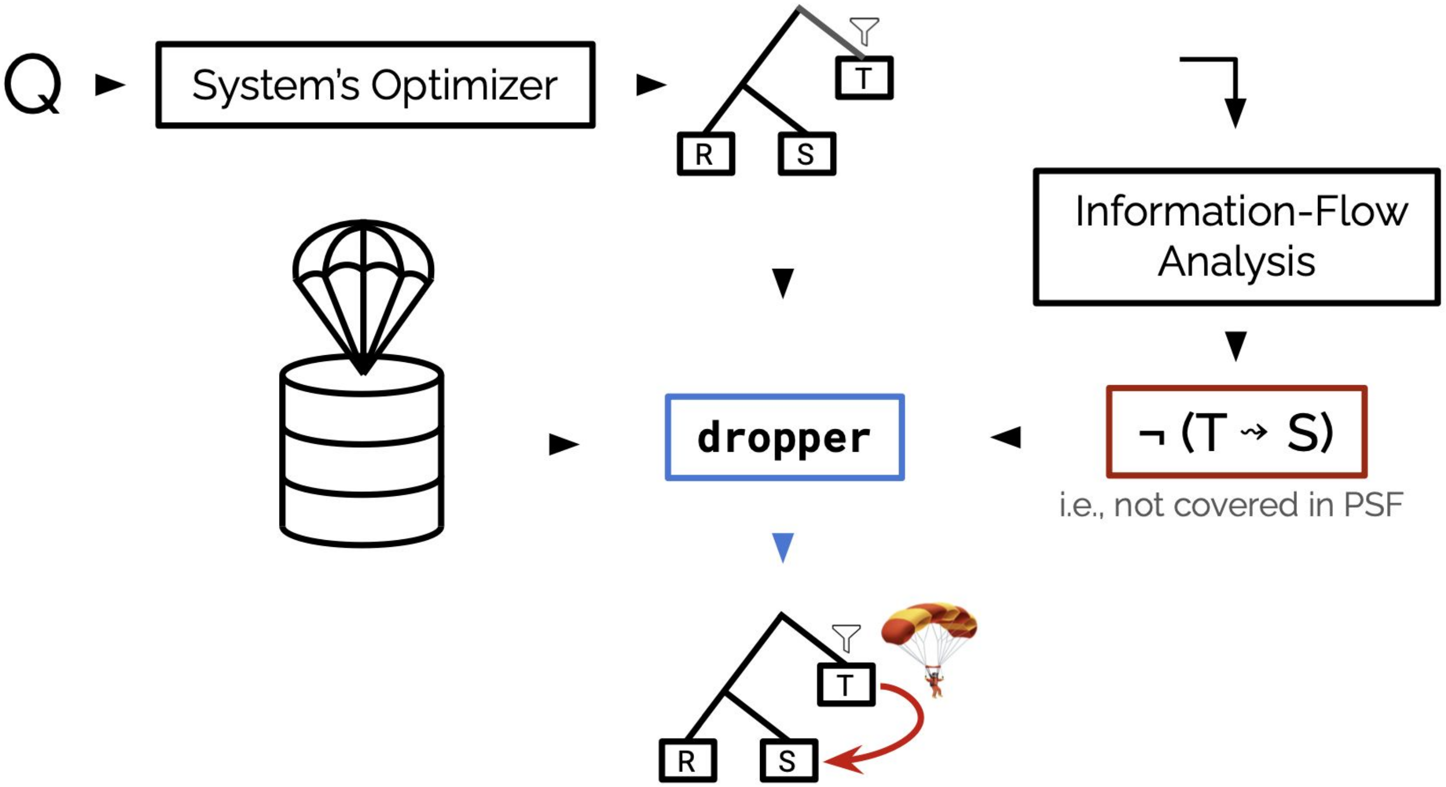


🪂 Parachute @Runtime

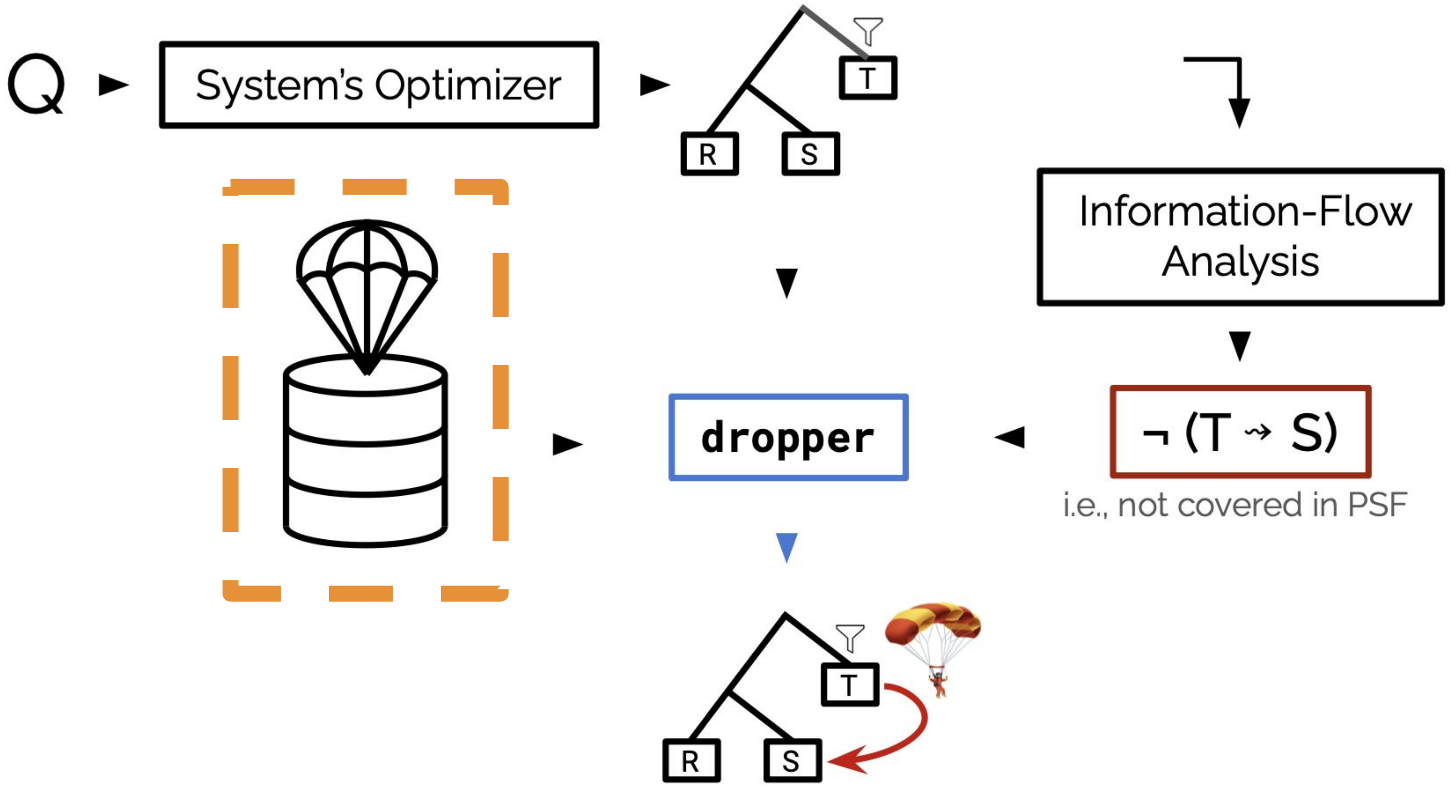


i.e., not covered in PSF

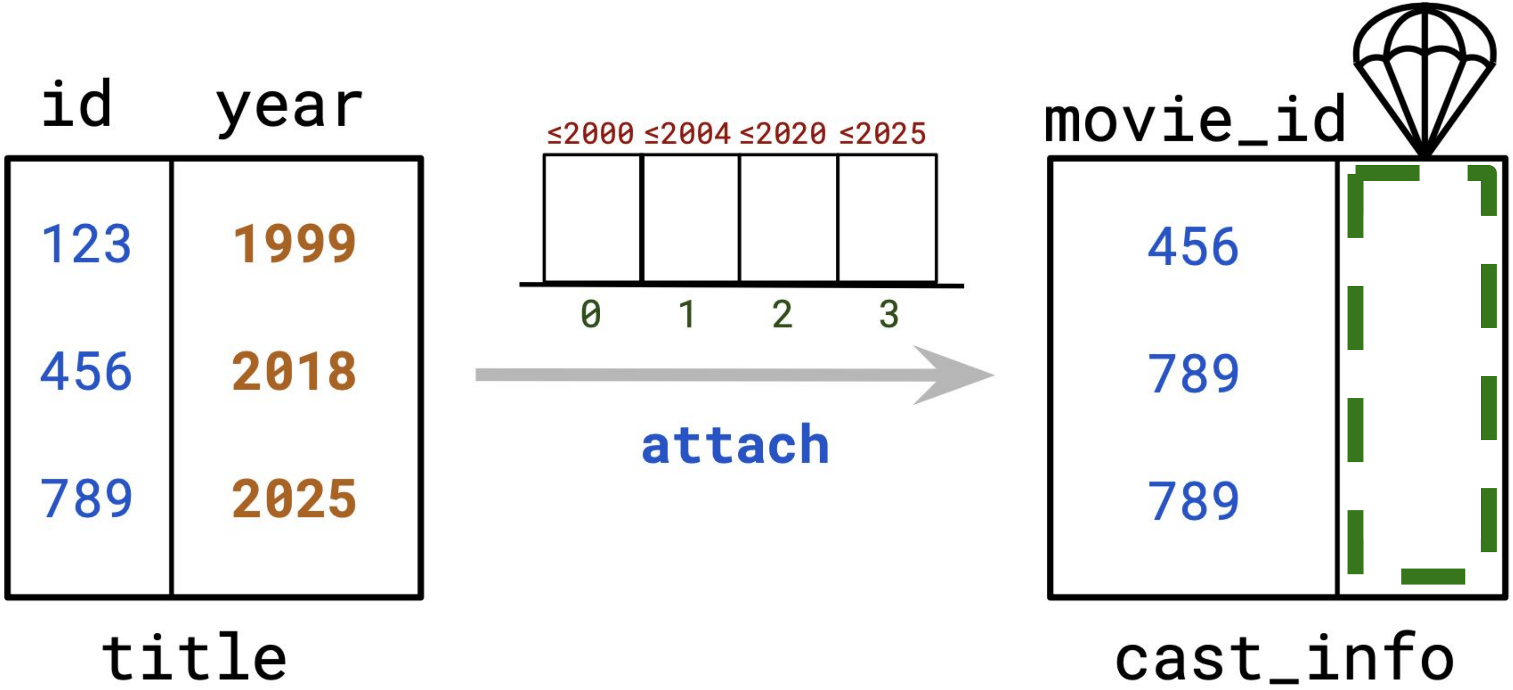
🪂 Parachute @Runtime



🪂 Parachute @Runtime



🪂 Parachute Columns



title

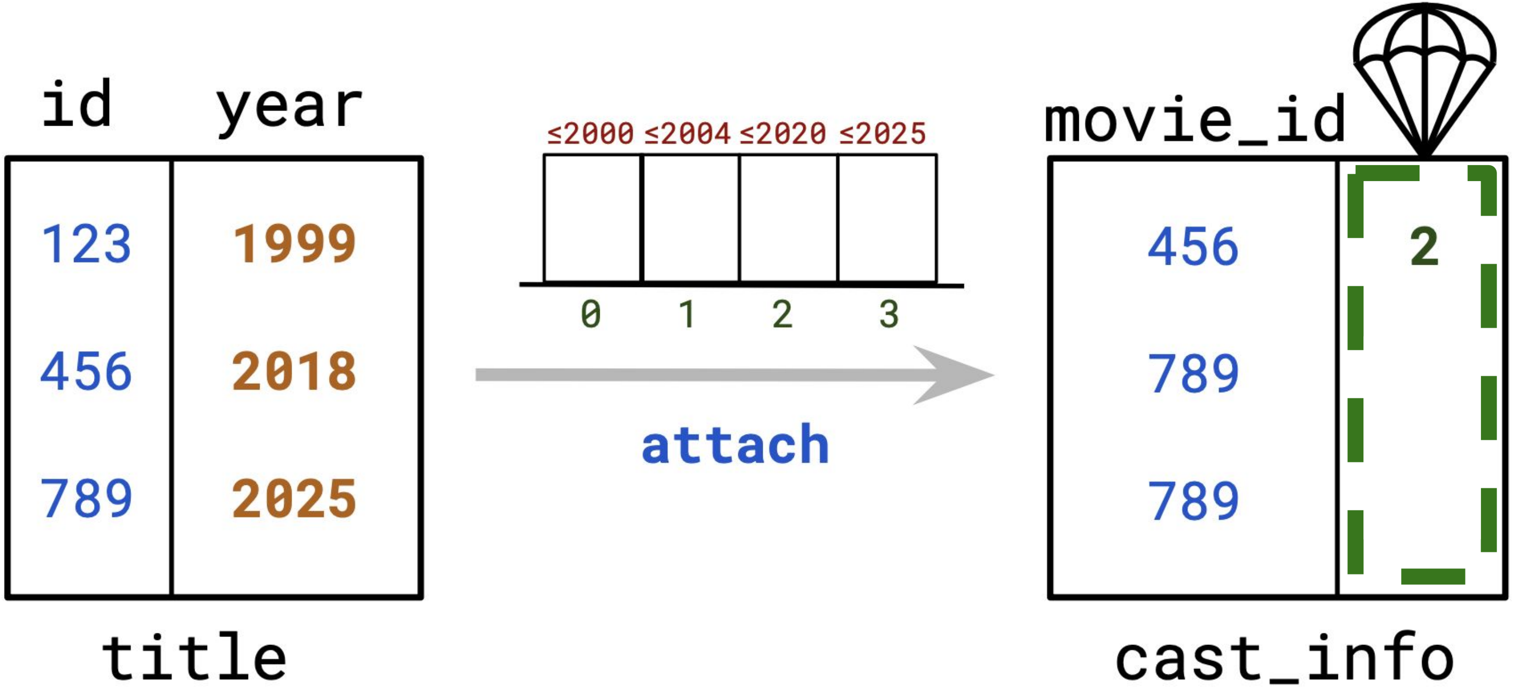
id	year
123	1999
456	2018
789	2025

≤2000	≤2004	≤2020	≤2025
0	1	2	3

cast_info

movie_id	
456	
789	
789	

🪂 Parachute Columns



title

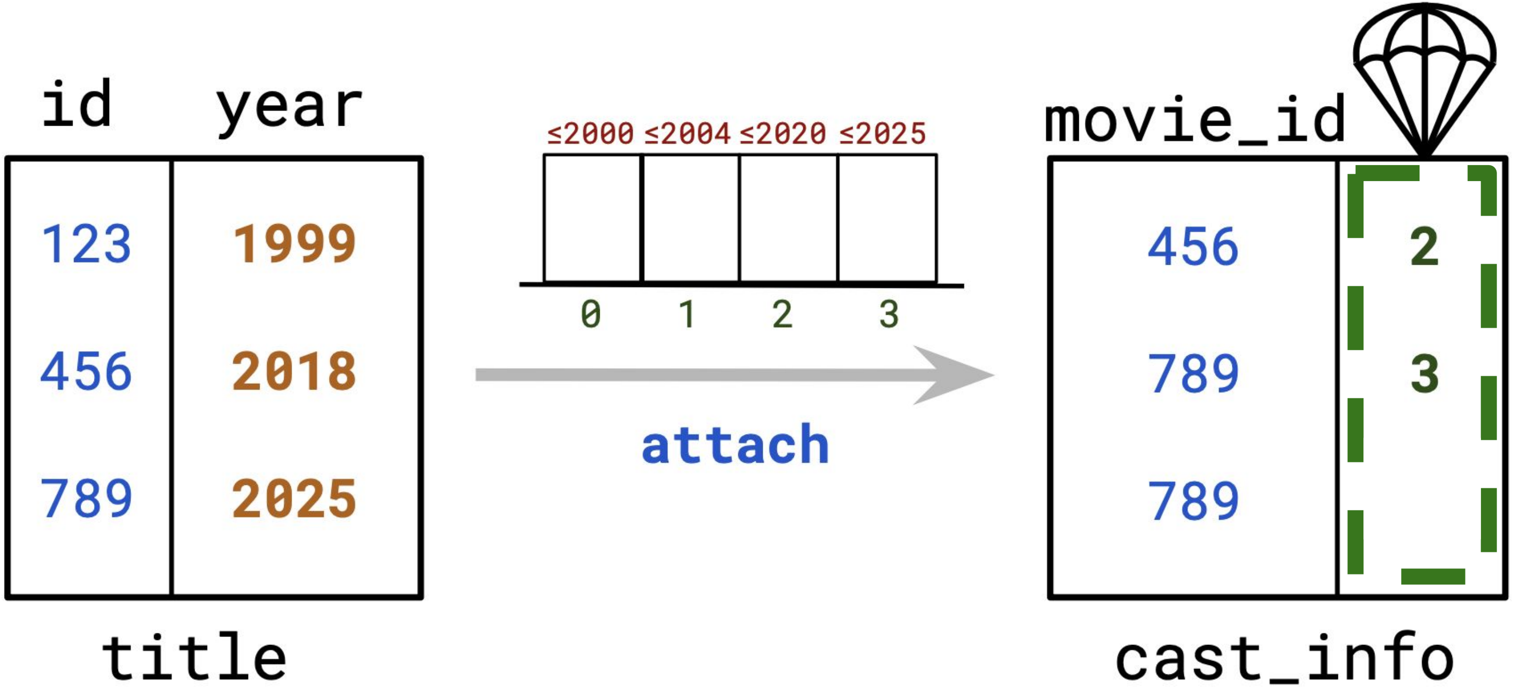
id	year
123	1999
456	2018
789	2025

0	1	2	3
≤2000	≤2004	≤2020	≤2025

cast_info

movie_id	
456	2
789	
789	

🪂 Parachute Columns



id	year
123	1999
456	2018
789	2025

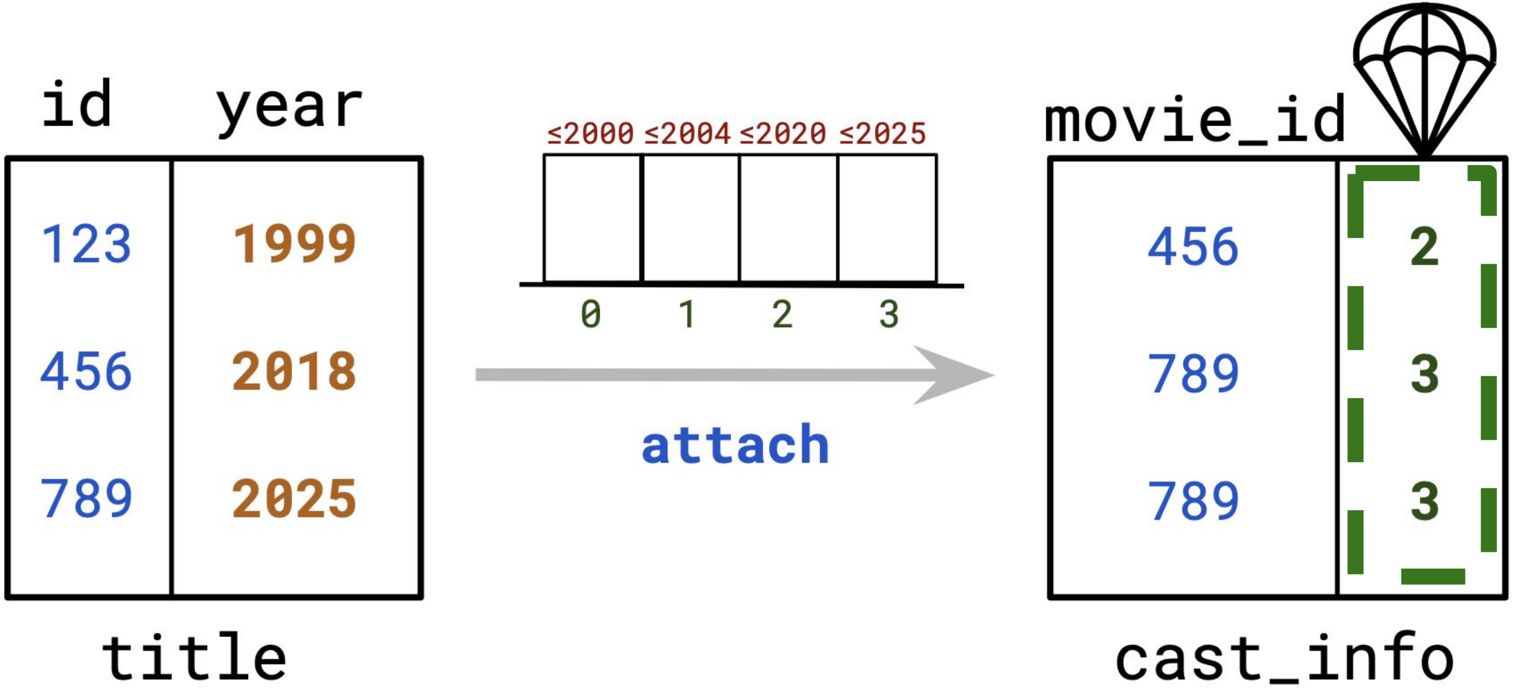
title

≤2000	≤2004	≤2020	≤2025
0	1	2	3

movie_id	
456	2
789	3
789	

cast_info

🪂 Parachute Columns



title

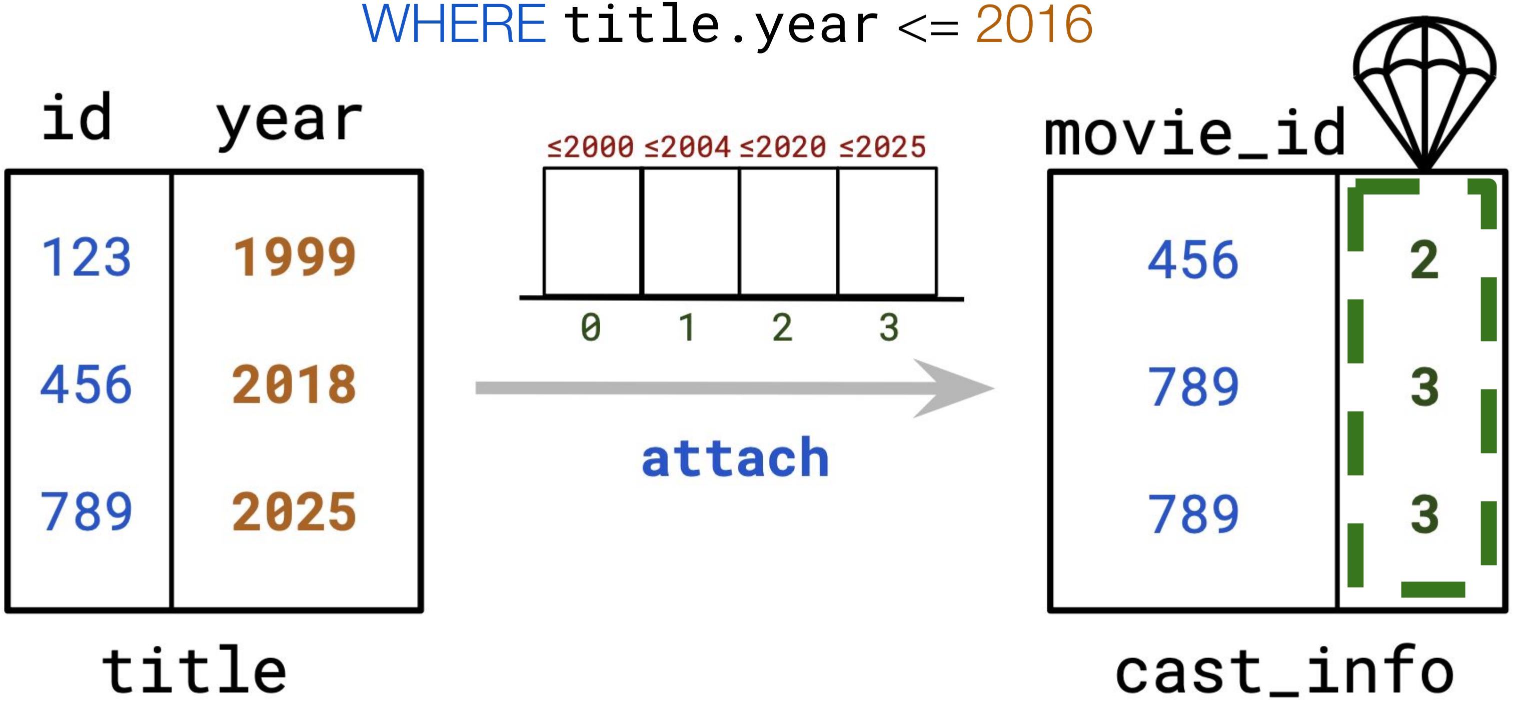
id	year
123	1999
456	2018
789	2025

cast_info

movie_id	
456	2
789	3
789	3

🪂 Parachute Predicates

WHERE title.year <= 2016



title

id	year
123	1999
456	2018
789	2025

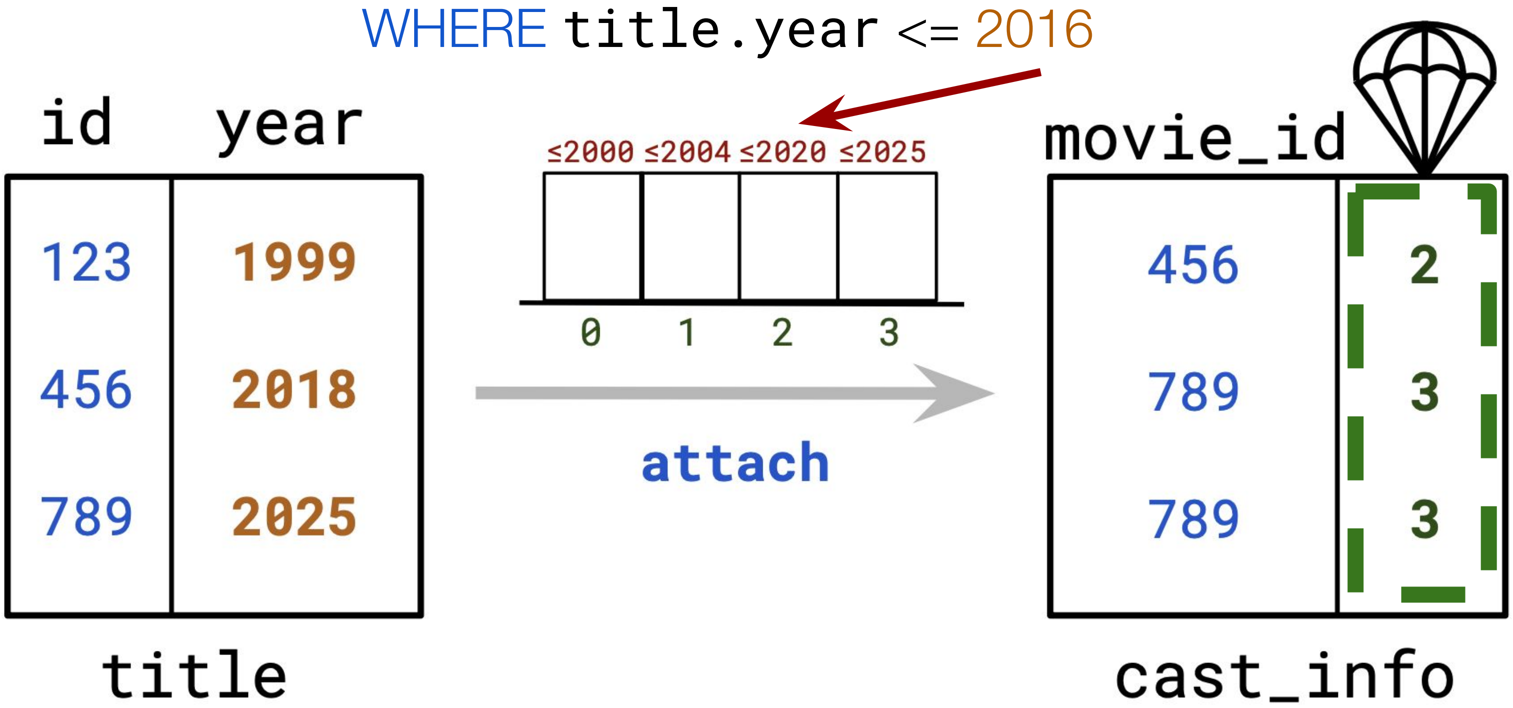
≤2000	≤2004	≤2020	≤2025
0	1	2	3

attach

cast_info

movie_id	
456	2
789	3
789	3

🪂 Parachute Predicates



WHERE `title.year <= 2016`

id	year
123	1999
456	2018
789	2025

title

≤2000	≤2004	≤2020	≤2025
0	1	2	3

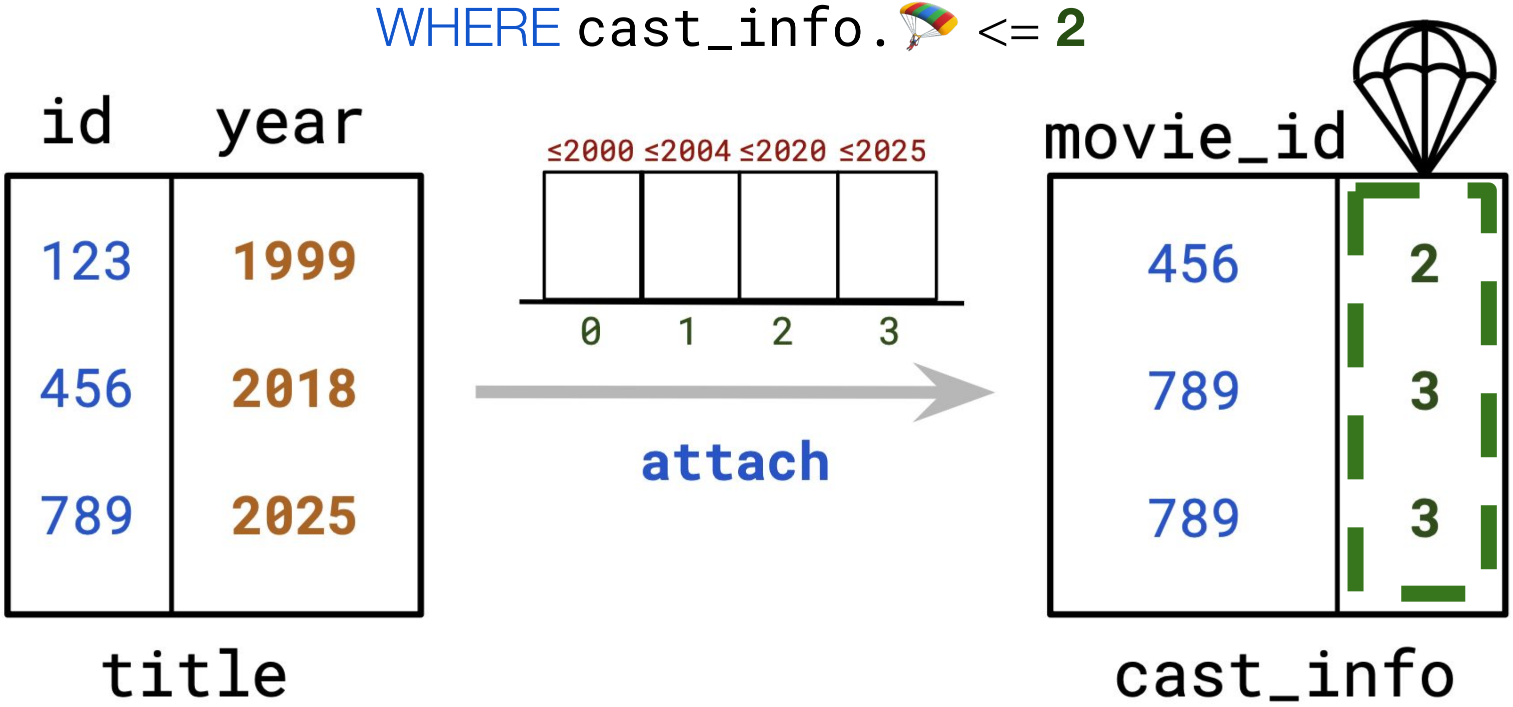
attach

movie_id	
456	2
789	3
789	3

cast_info

🪂 Parachute Predicates

WHERE `cast_info.🪂 <= 2`



title

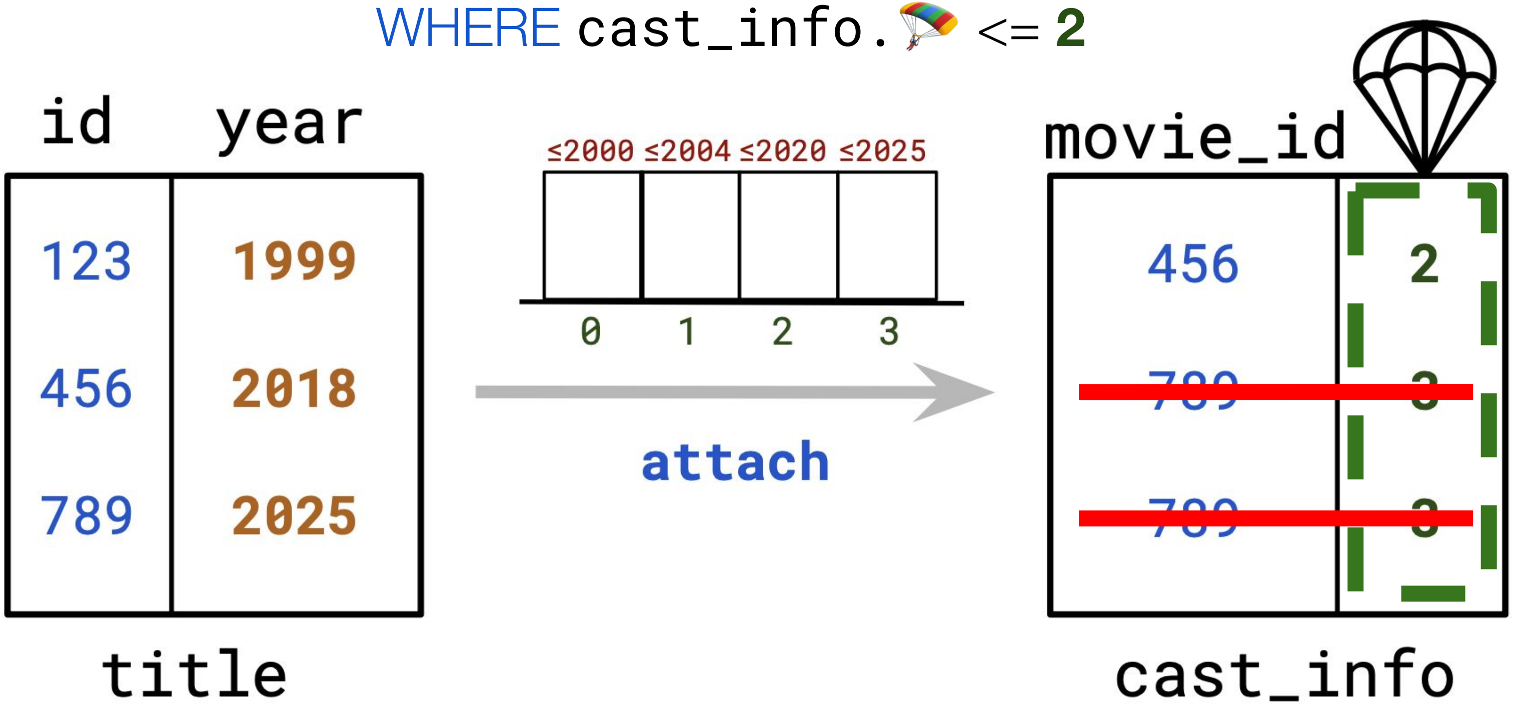
id	year
123	1999
456	2018
789	2025

cast_info

movie_id	🪂
456	2
789	3
789	3

🪂 Parachute Predicates

WHERE cast_info.🪂 <= 2



id	year
123	1999
456	2018
789	2025

title

movie_id	
456	2
~~789~~	~~3~~
~~789~~	~~3~~

cast_info

Evaluation

- **Benchmarks**:
  - JOB: 113 queries.
  - CEB: > 13K queries.

Evaluation

- **Benchmarks**:
  - JOB: 113 queries.
  - CEB: > 13K queries.
- **Competitors**:
  - Vanilla DuckDB v1.2: **`duckdb v1.2`**.
  - DuckDB v1.2 w/ semi-join filtering: **`duckdb v1.2 + sjf`** (by @andizimmerer).

Evaluation

- **Benchmarks**:
  - JOB: 113 queries.
  - CEB: > 13K queries.
- **Competitors**:
  - Vanilla DuckDB v1.2: **`duckdb v1.2`**.
  - DuckDB v1.2 w/ semi-join filtering: **`duckdb v1.2 + sjf`** (by @andizimmerer).
  - Parachute on vanilla DuckDB: **`parachute[duckdb]`**.
  - Parachute on DuckDB w/ semi-join filtering: **`parachute[duckdb + sjf]`**.

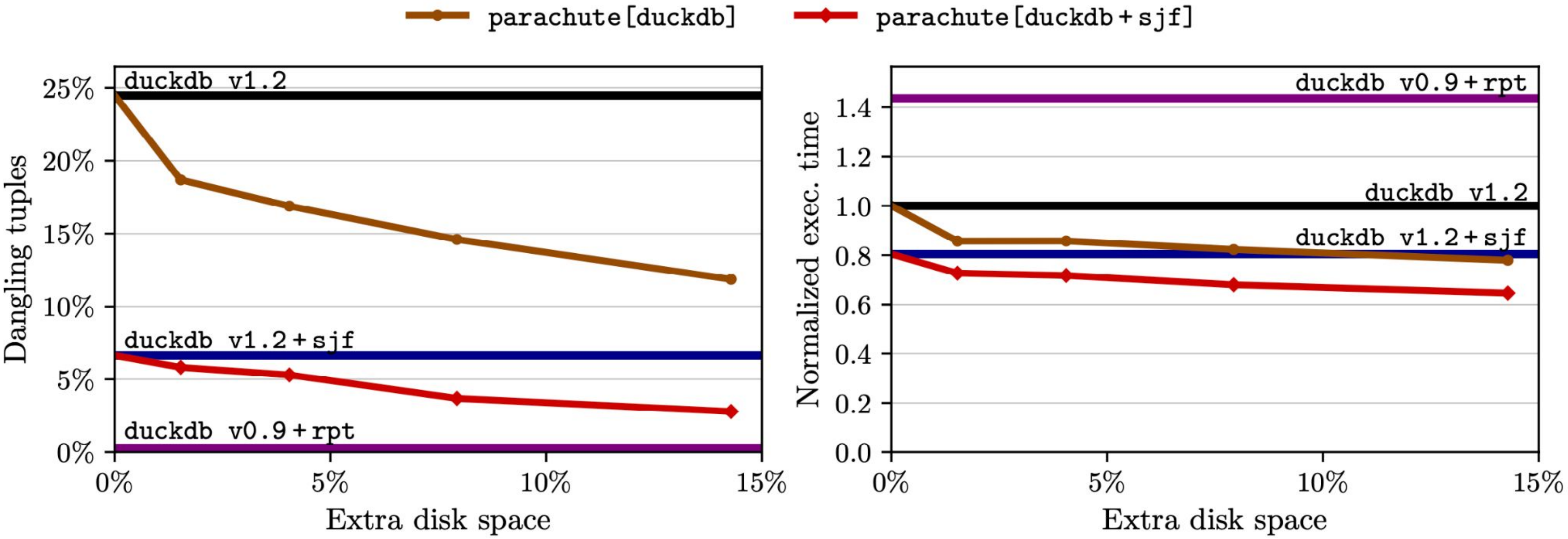
Evaluation

- **Benchmarks**:
  - JOB: 113 queries.
  - CEB: > 13K queries.
- **Competitors**:
  - Vanilla DuckDB v1.2: **`duckdb v1.2`**.
  - DuckDB v1.2 w/ semi-join filtering: **`duckdb v1.2 + sjf`** (by @andizimmerer).
  - Parachute on vanilla DuckDB: **`parachute[duckdb]`**.
  - Parachute on DuckDB w/ semi-join filtering: **`parachute[duckdb + sjf]`**.
  - *Bonus*: RPT as @SIGMOD’25: **`duckdb v0.9 + rpt`**.

Evaluation

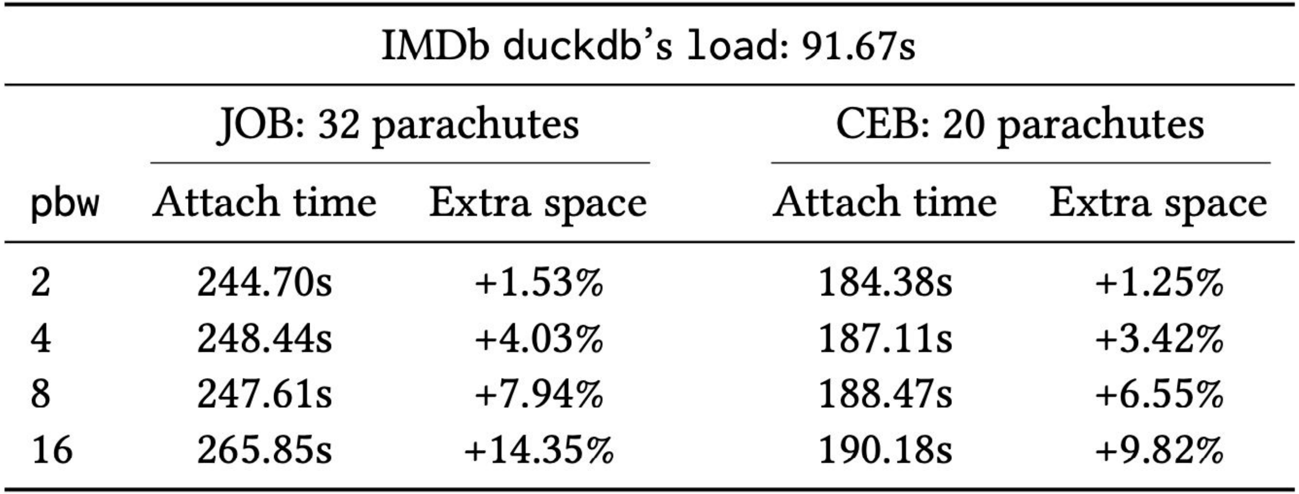
- **Benchmarks**:
  - JOB: 113 queries.
  - CEB: > 13K queries.
- **Competitors**:
  - Vanilla DuckDB v1.2: **`duckdb v1.2`**.
  - DuckDB v1.2 w/ semi-join filtering: **`duckdb v1.2 + sjf`** (by @andizimmerer).
  - Parachute on vanilla DuckDB: **`parachute[duckdb]`**.
  - Parachute on DuckDB w/ semi-join filtering: **`parachute[duckdb + sjf]`**.
  - *Bonus*: RPT as @SIGMOD'25: **`duckdb v0.9 + rpt`**.
- Parachute column bit-widths ∈ {2, 4, 8, 16}.

JOB: #DanglingTuples & Exec. Time



Build Overhead

- JOB: Parachute’s build amortizes in the 4th run.
- CEB: The overhead is amortized already in the 1st run.



IMDb duckdb’s load: 91.67s				
	JOB: 32 parachutes		CEB: 20 parachutes	
pbw	Attach time	Extra space	Attach time	Extra space
2	244.70s	+1.53%	184.38s	+1.25%
4	248.44s	+4.03%	187.11s	+3.42%
8	247.61s	+7.94%	188.47s	+6.55%
16	265.85s	+14.35%	190.18s	+9.82%

Future Work

- Parachute columns ~ “cached bloom filters”:

  ⇒ Parachute columns can be used for partition pruning — think zonemaps.

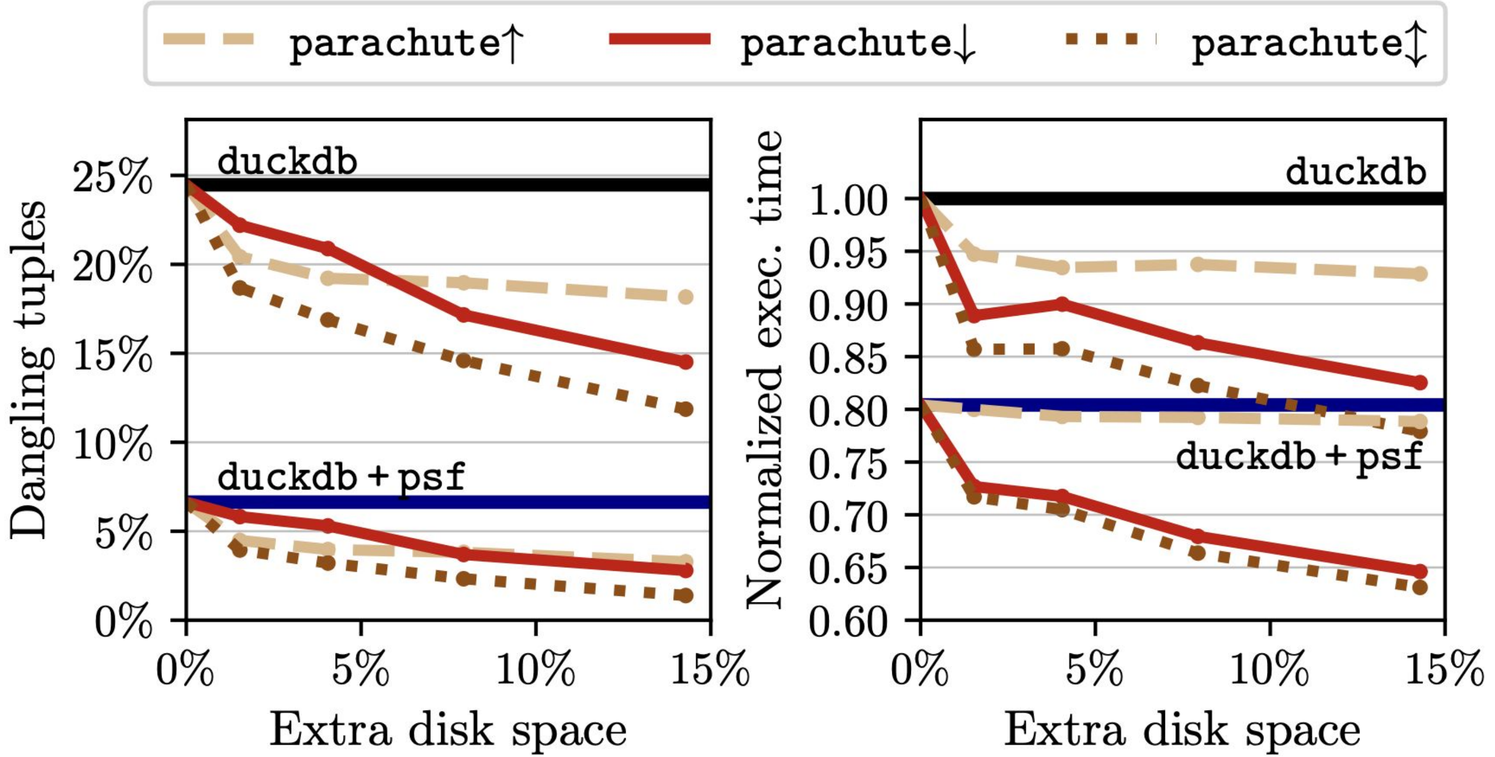
- Instance-optimized parachute columns.

  ⇒ Check out *string fingerprints* in AIDB’25 ⇒ 🚀 1.36x faster `LIKE` scans.

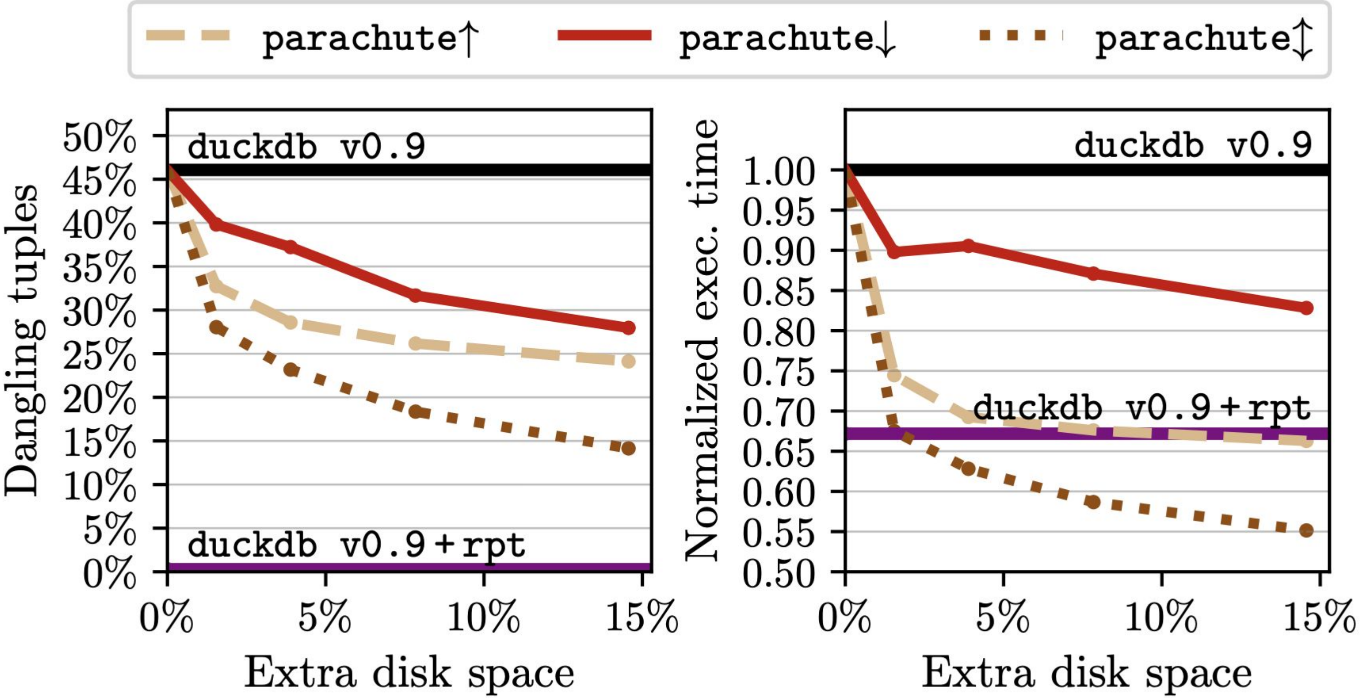
📖 `@utndatasystems/parachute`

Back-up Slides

🪂 Parachute Modes



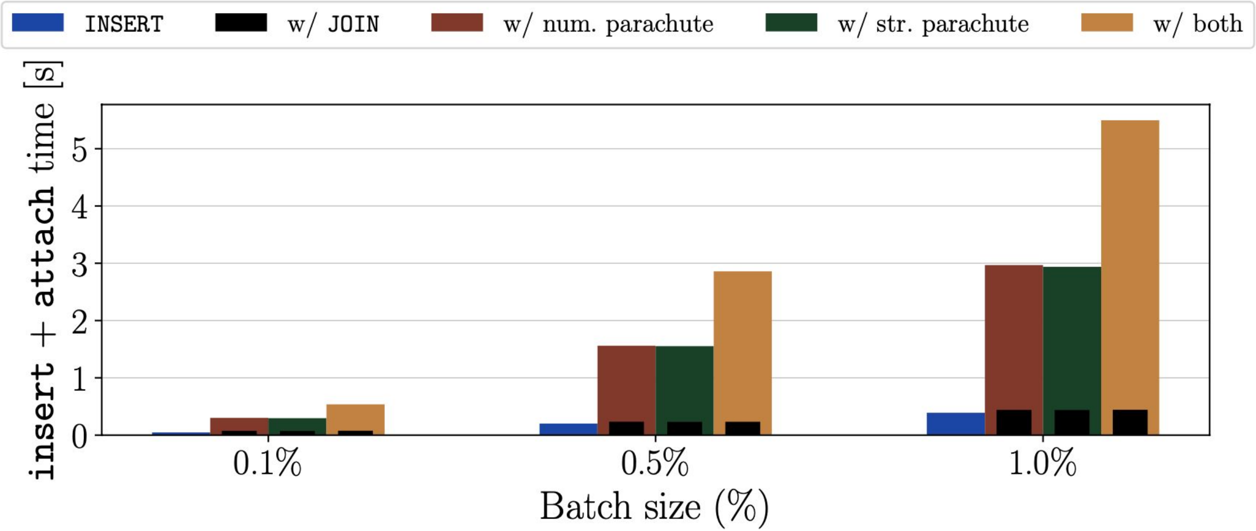
JOB on DuckDB v0.9: 🪂 Parachute vs. RPT



Update Overhead

- DuckDB v1.2 runs `UPDATE` single-threaded.

  ⇒ `JOIN` is *not* the main bottleneck.



Information Flow

- **Formalization:**

$$R \rightsquigarrow S :\Longleftrightarrow (R \prec S) \land (R \leftrightarrow S) \land \mathtt{is\_probe}(S),$$

$$R \rightsquigarrow^{n+1} S :\Longleftrightarrow \exists T.\, R \rightsquigarrow^{n} T \rightsquigarrow S,$$

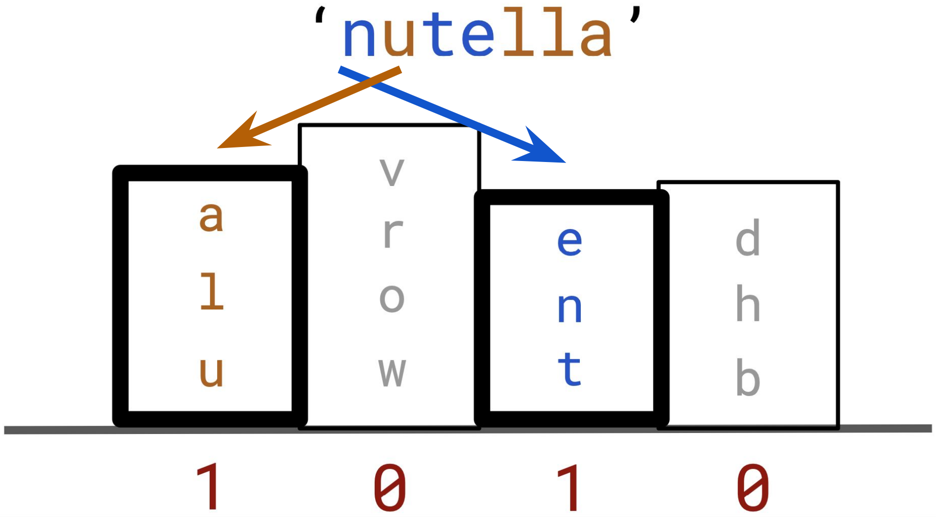
$$R \rightsquigarrow^{*} S :\Longleftrightarrow \exists n > 0.\, R \rightsquigarrow^{n} S.$$

- **Precedence Relation:**

$$R \prec S :\Longleftrightarrow \begin{array}{l} (\mathtt{pipeline}(R) < \mathtt{pipeline}(S)) \\ \lor\ (\mathtt{pipeline}(R) = \mathtt{pipeline}(S) \land \mathtt{is\_probe}(S)). \end{array}$$

🪂 Parachute’s String Fingerprints

1. **Partition** the letter space in a fixed number of bins.
2. Compute a **bitmask** of bin indices



‘nutella’
a
l
u
v
r
o
w
e
n
t
d
h
b
1
0
1
0