清華大學
Tsinghua University

CWI
Centrum Wiskunde & Informatica

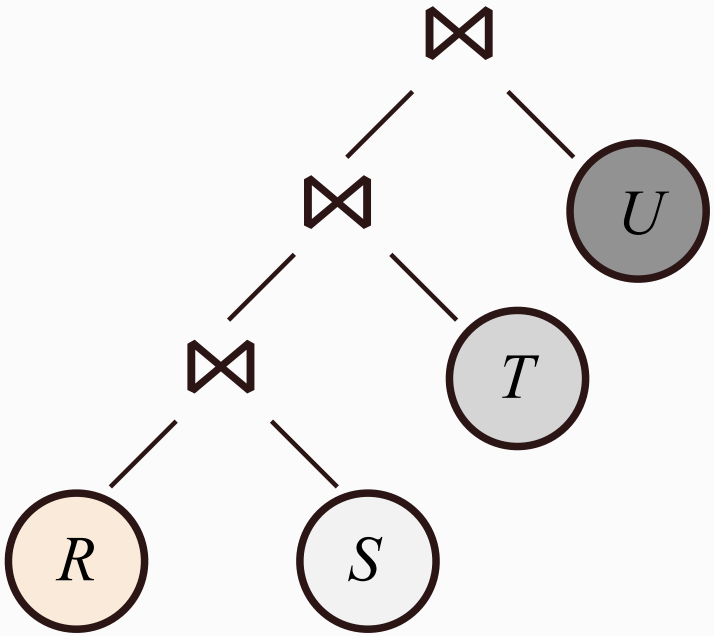
Robust Predicate Transfer With Dynamic Execution

Yiming Qiao[1], Peter Boncz[2], Huanchen Zhang[1]
[1]Tsinghua University [2]CWI

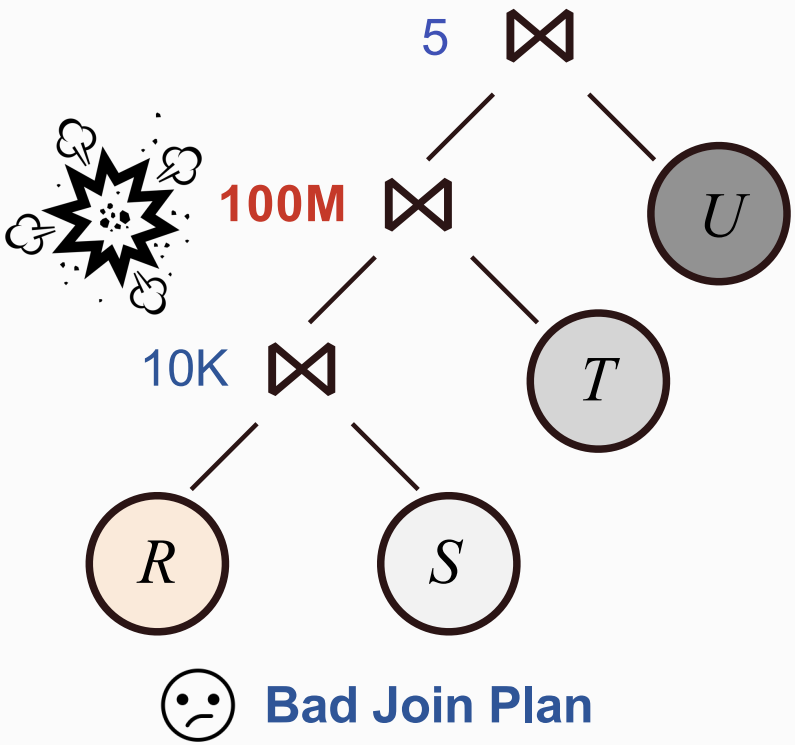
Join Order Matters

```
SELECT  *
FROM    R, S, T, U
WHERE   R.a = S.a
AND     R.b = T.b
AND     R.c = U.c
```

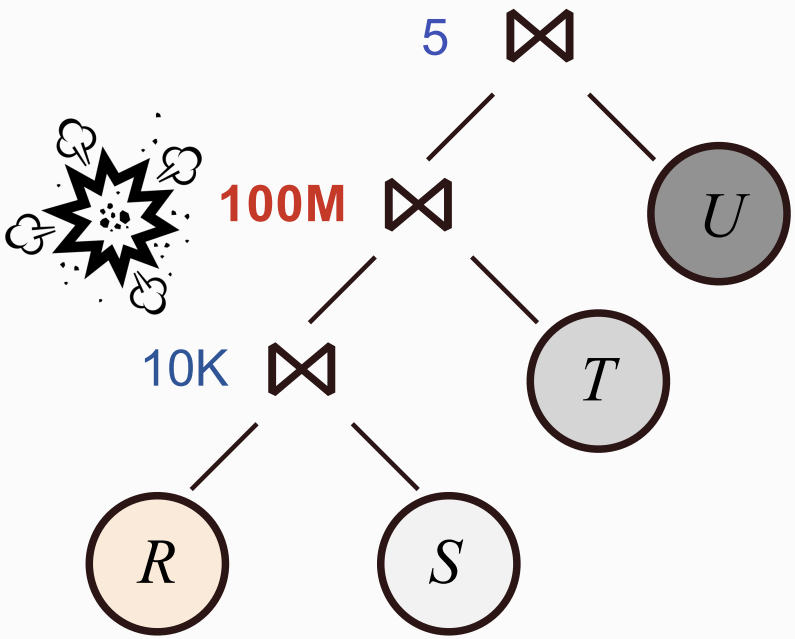
Join Plan

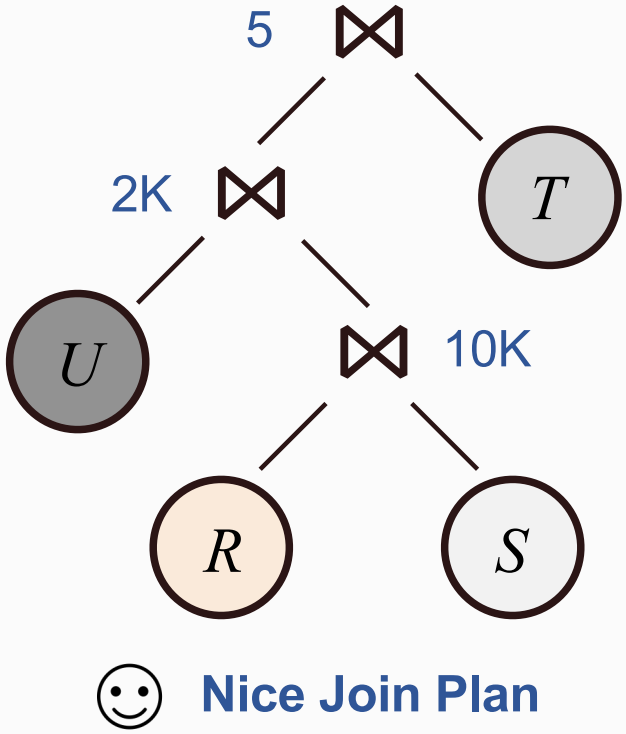


Join Order Matters



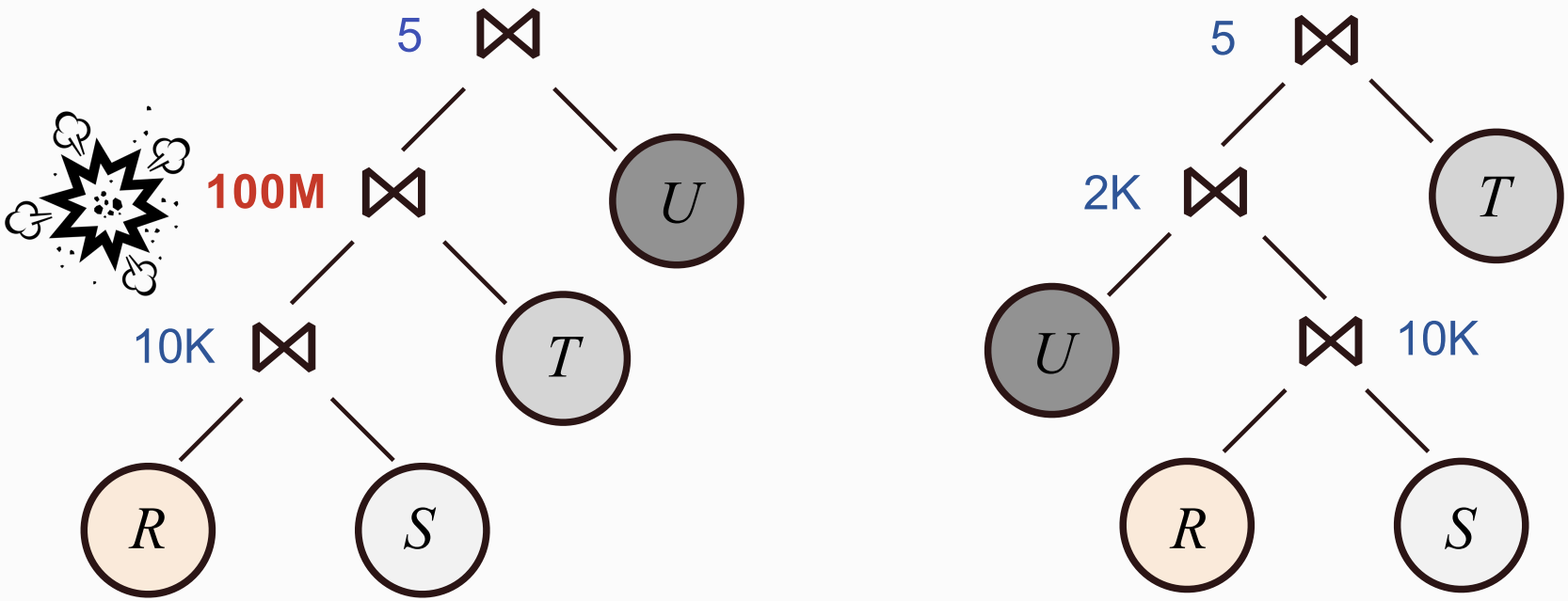
Join Order Matters





Nice Join Plan

Join Order Matters



Much prior work studies how to choose the join plan with minimum intermediate result size

From One Join Equality to Two Filters

SELECT *
FROM R, S, T, U
WHERE $R.a = S.a$
AND $R.b = T.b$
AND $R.c = U.c$

From One Join Equality to Two Filters

WHERE $R.a = S.a$ $\quad=\quad$ $R.a \subseteq S.a$ **and** $R.a \supseteq S.a$

From One Join Equality to Two Filters

Decompose One Join equality into Two Membership Filters

WHERE $R.a = S.a$ $\Leftrightarrow$ $R.a \subseteq S.a$ **and** $R.a \supseteq S.a$

Membership Filters (e.g. Bloom Filter)

From One Join Equality to Two Filters

$R.a \subseteq S.a$

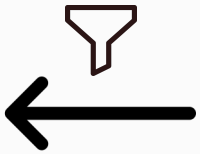
a	b	c
2	xx	xx
4	xx	xx
3	xx	xx
8	xx	xx

a	d
1	xx
4	xx
3	xx
7	xx
9	xx

From One Join Equality to Two Filters

$R.a$ $\subseteq$ $S.a$

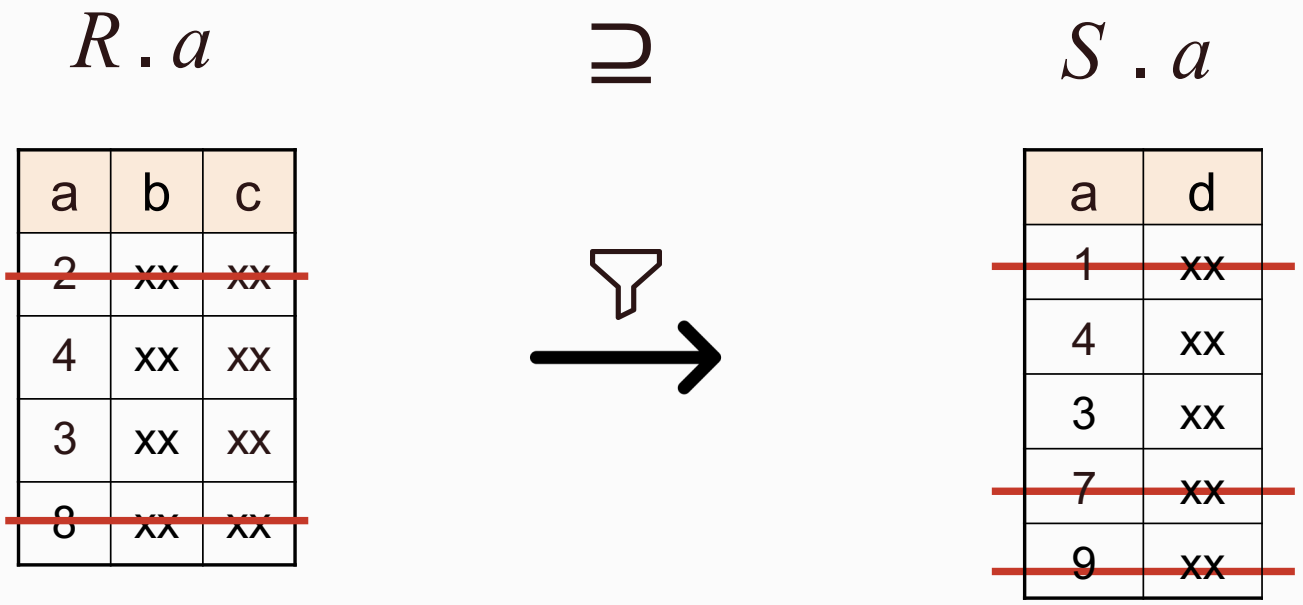
a	b	c
~~2~~	~~xx~~	~~xx~~
4	xx	xx
3	xx	xx
~~8~~	~~xx~~	~~xx~~



a	d
1	xx
4	xx
3	xx
7	xx
9	xx

Build a membership filter based on S.a to reduce the other

From One Join Equality to Two Filters



Build a membership filter based on R.a to reduce the other

From One Join Equality to Two Filters

$R.a$

a	b	c
~~2~~	~~xx~~	~~xx~~
4	xx	xx
3	xx	xx
~~8~~	~~xx~~	~~xx~~

$\bowtie$

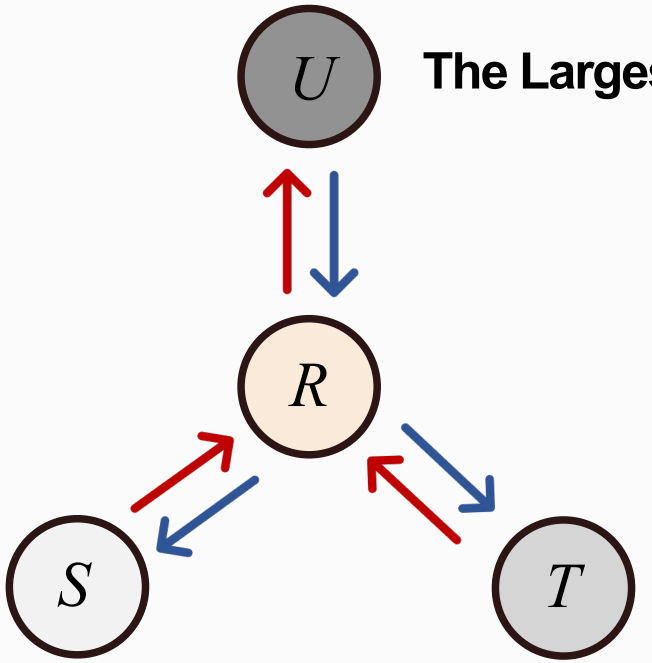
$S.a$

a	d
~~1~~	~~xx~~
4	xx
3	xx
~~7~~	~~xx~~
~~9~~	~~xx~~

Reduced Tables are then Consumed by other Relational Operators

Robust Predicate Transfer (RPT)

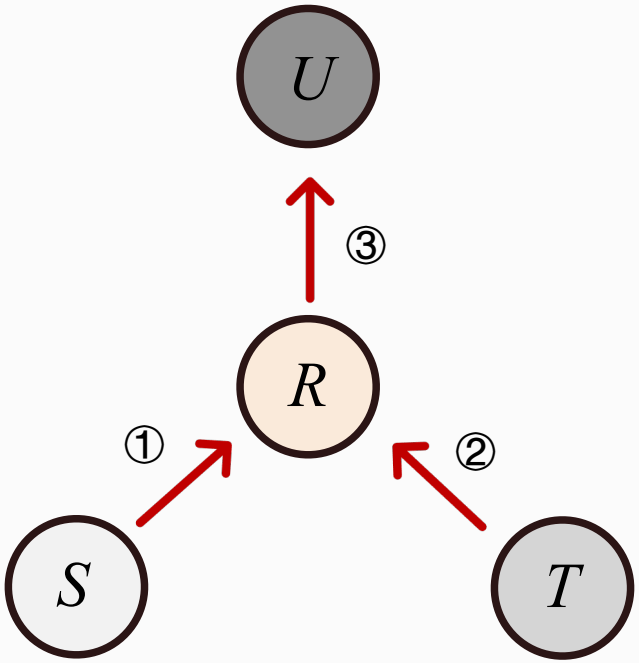
For more than two tables join, RPT introduces a transfer plan: LargestRoot



Filter Transfer Plan

Robust Predicate Transfer (RPT)

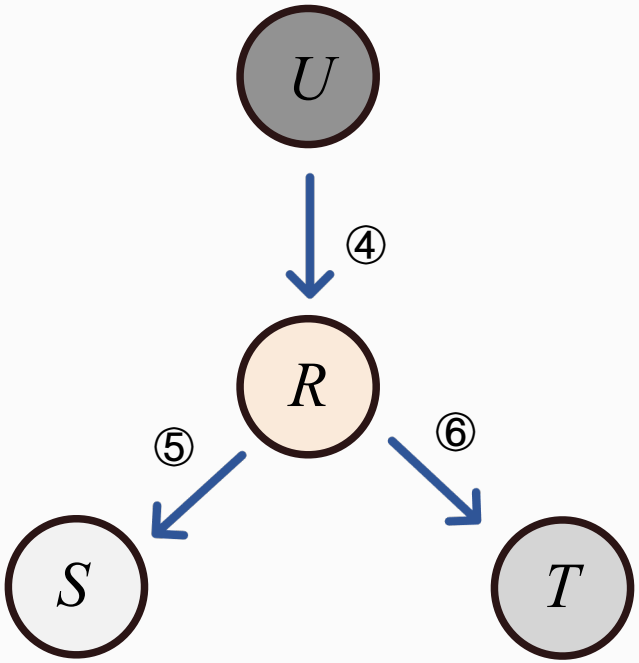
For more than two tables join, RPT introduces a transfer plan: LargestRoot



Forward Pass: $S \rightarrow R,\ T \rightarrow R,\ R \rightarrow U$

Robust Predicate Transfer (RPT)

For more than two tables join, RPT introduces a transfer plan: LargestRoot

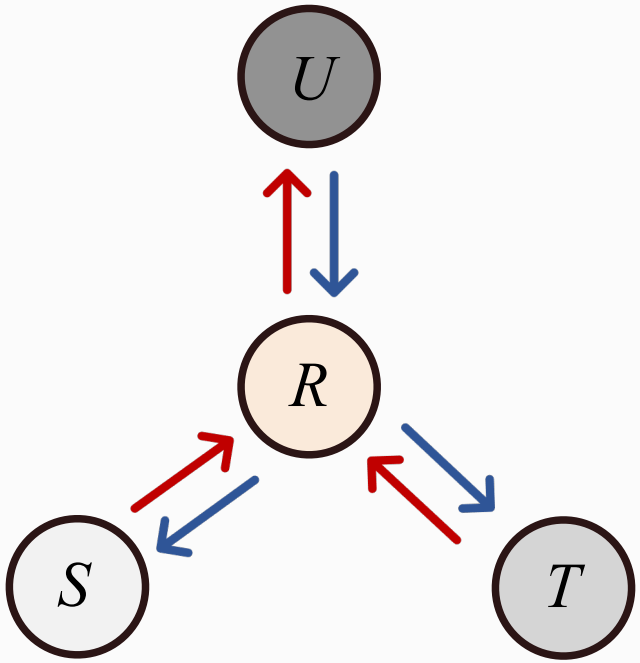


Forward Pass: $S \rightarrow R,\ T \rightarrow R,\ R \rightarrow U$

Backward Pass: $U \rightarrow R,\ R \rightarrow T,\ R \rightarrow S$

Robust Predicate Transfer (RPT)

For more than two tables join, RPT introduces a transfer plan: LargestRoot



Filter Transfer Plan

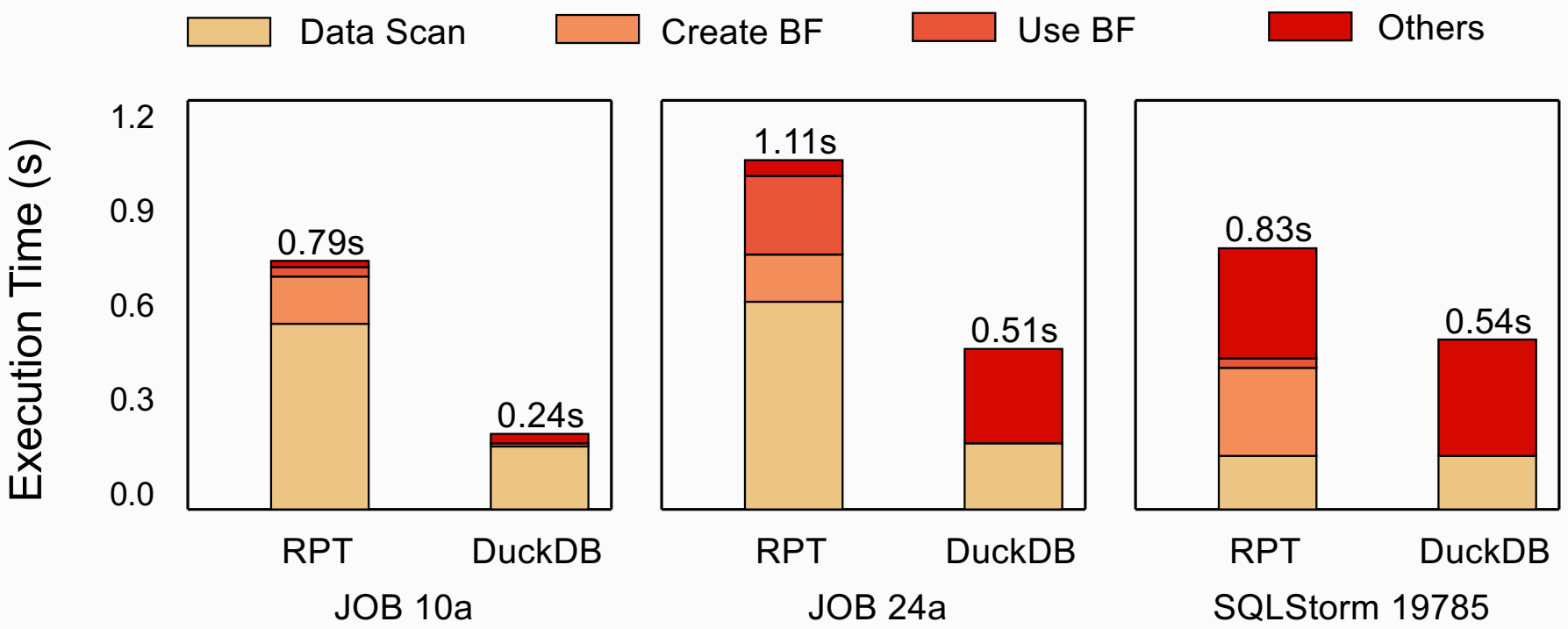
Forward Pass **is from Leaves to the Root**

Backward Pass **is from the Root to Leaves**

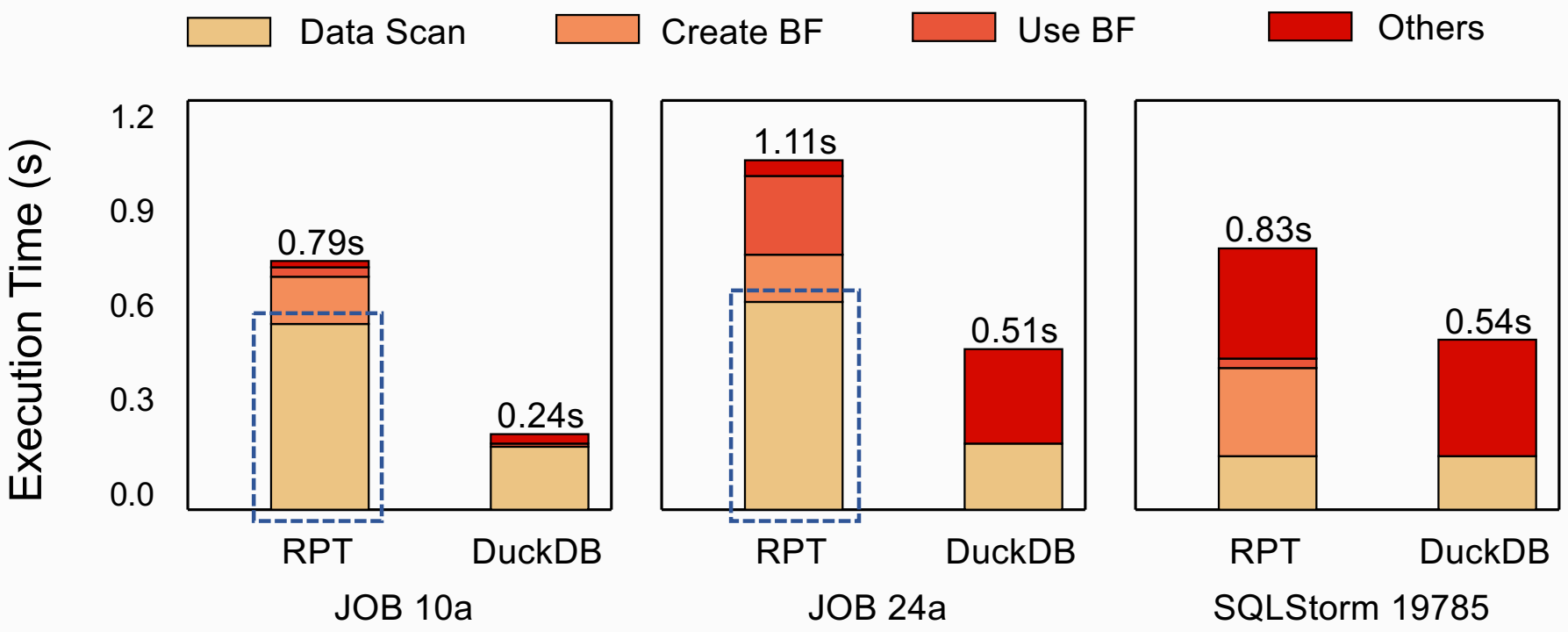
RPT preserves the robust guarantee $O(|IN| + |OUT|)$, which is **instance-optimal,** but pays

A Fixed Tax of Table Scan and Filter Transfer

Overhead of RPT

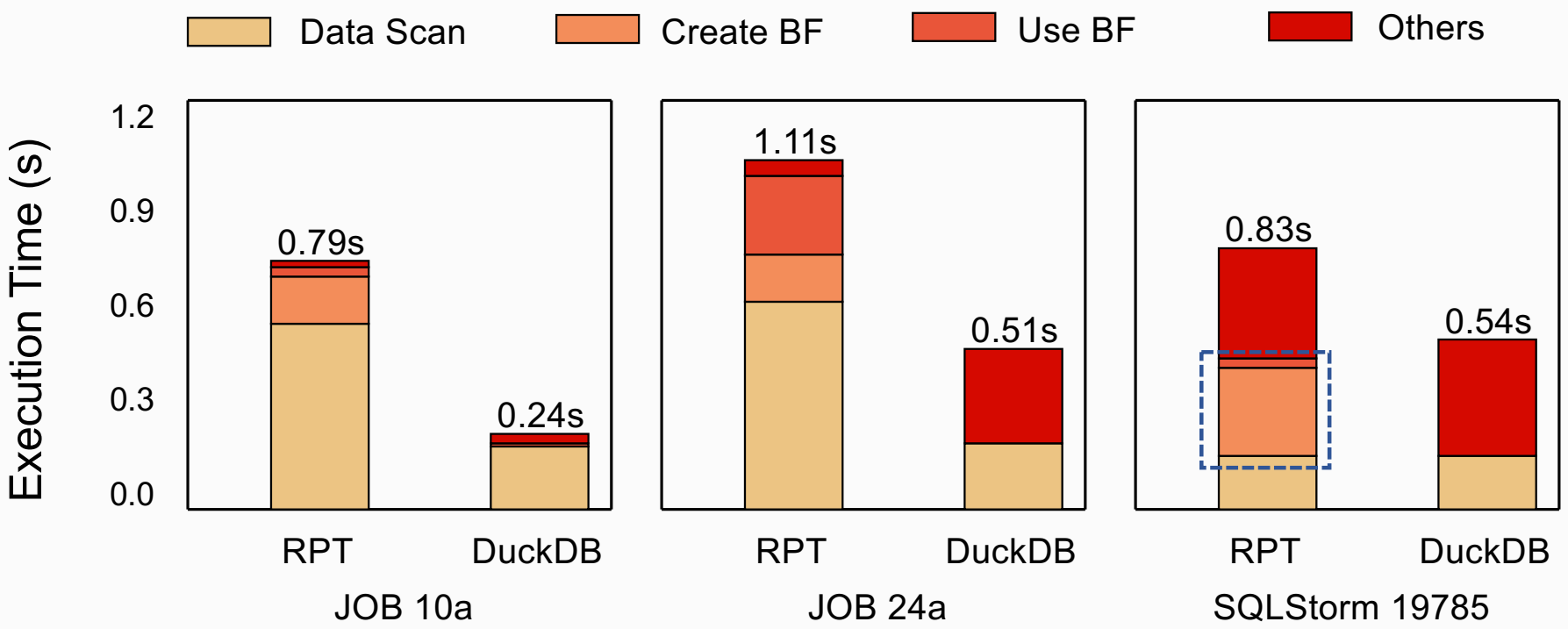


Overhead of RPT



Scan Overhead

Overhead of RPT



BF-Build Overhead

Where Does RPT Pay Overhead

1. Redundant Filter Builds

   Our solution: **Asymmetric Transfer Plan**

2. Storage-Format-Unaware Filtering (e.g., row group skipping)

   Our solution: **Cascade Filter**

3. Overhead on Non-Reducible Queries

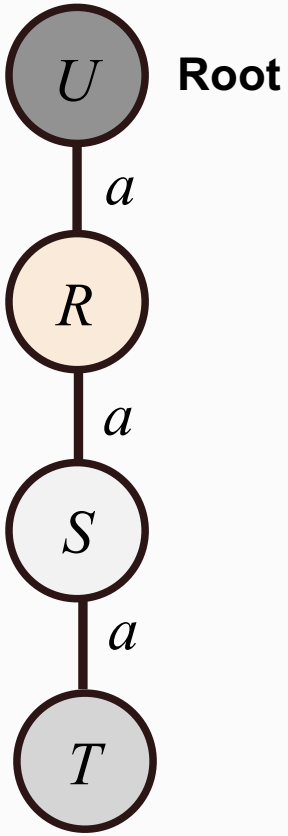
   Our solution: **Dynamic Pipeline**

1. Redundant Filter Builds

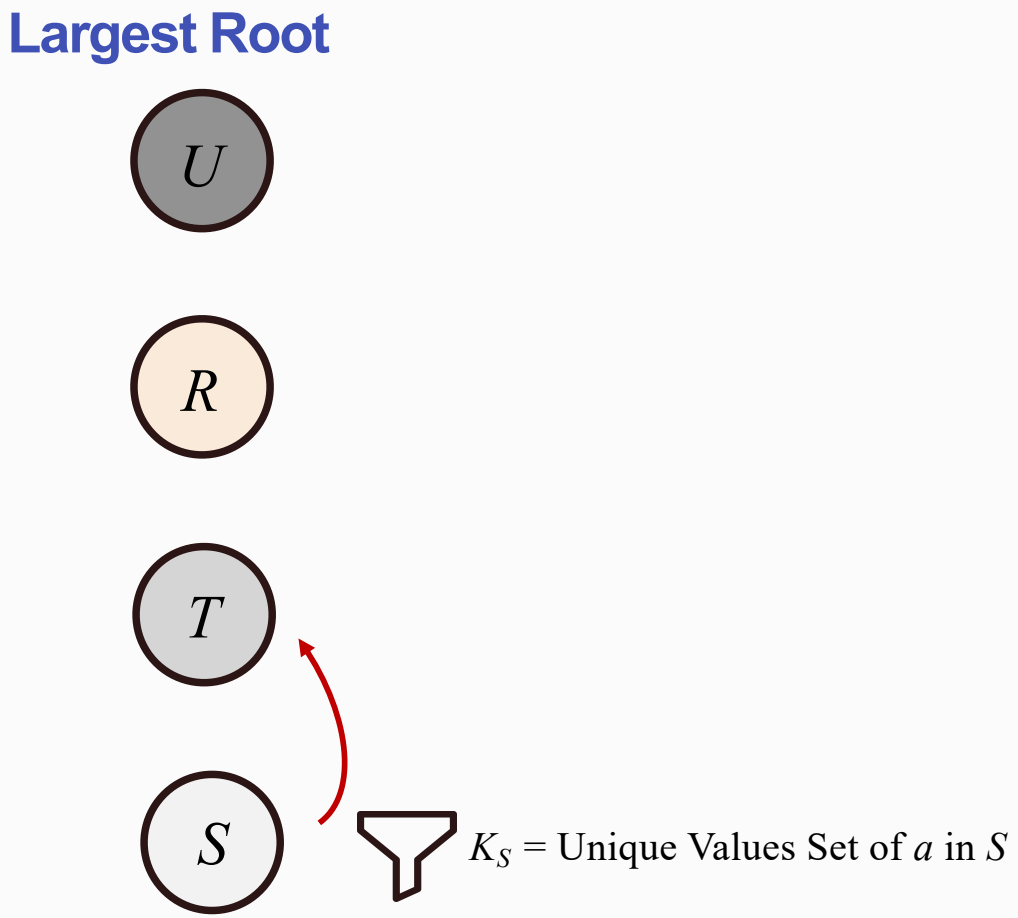
```
SELECT  *
FROM    R, S, T, U
WHERE   R.a = S.a
AND     R.a = T.a
AND     R.a = U.a
```

$$|T| < |S| < |R| < |U|$$

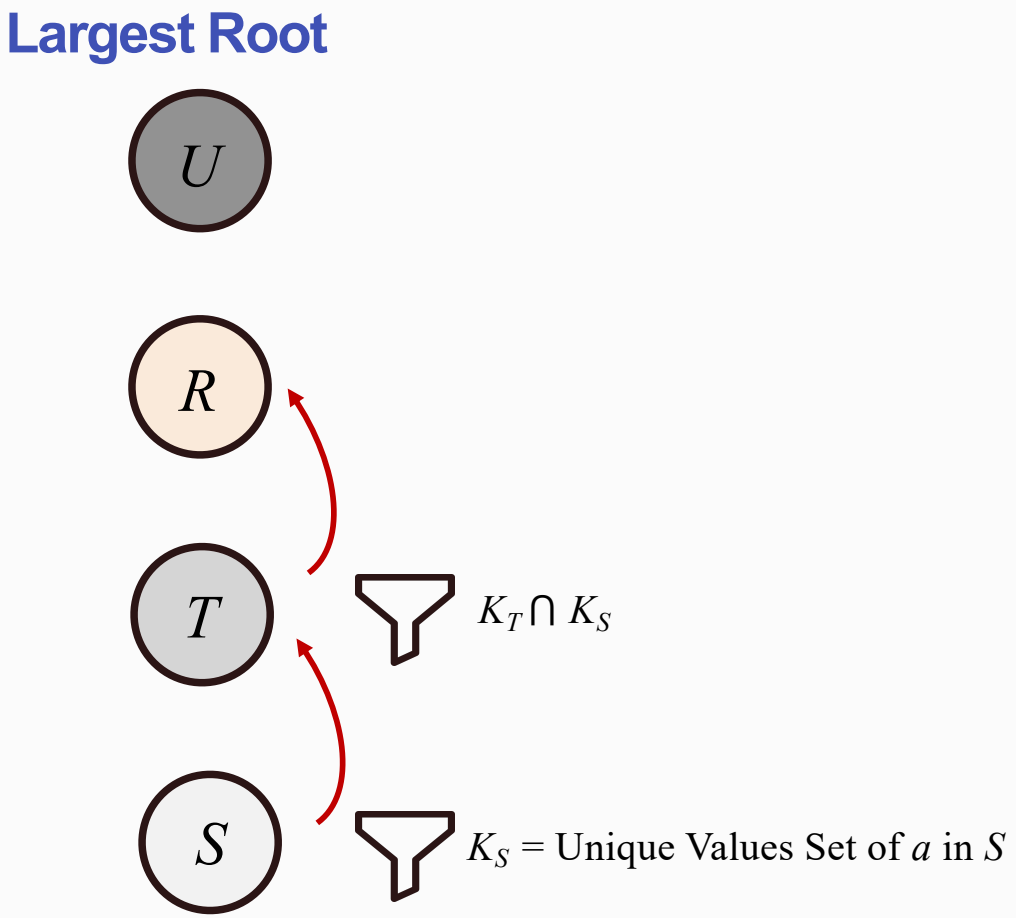
Largest Root (Transfer Plan)



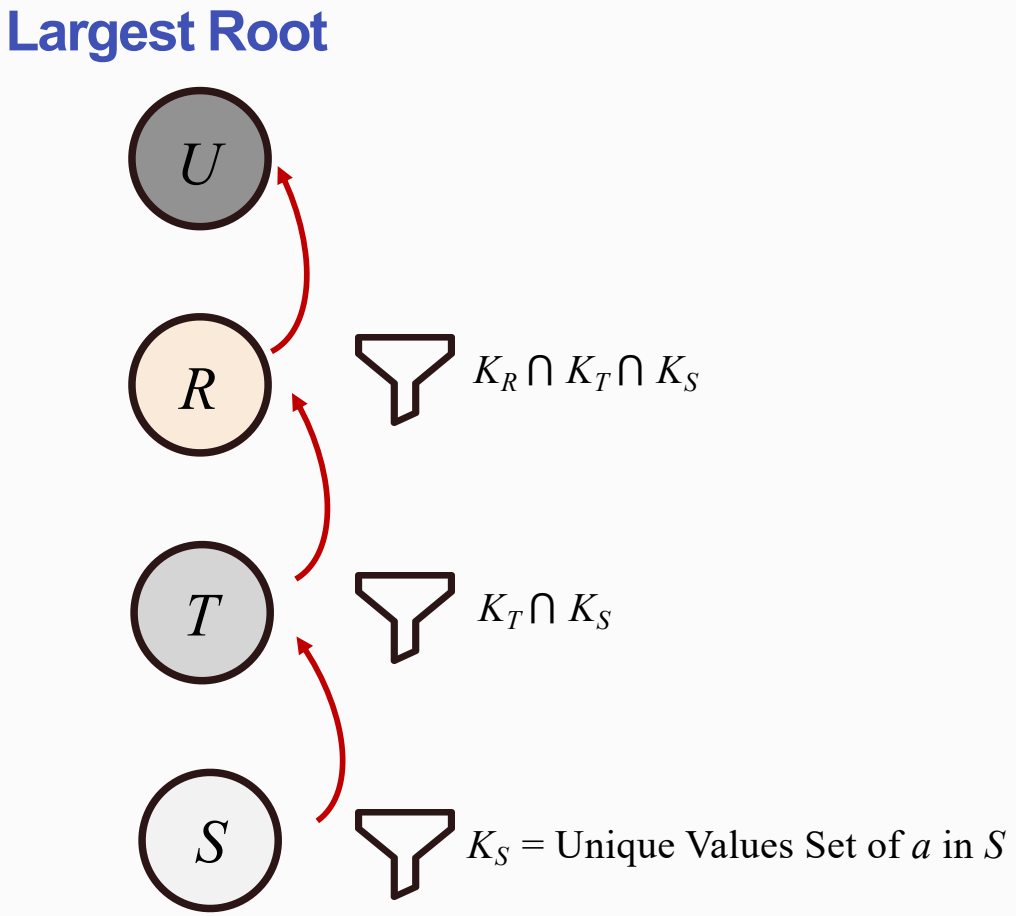
1. Redundant Filter Builds



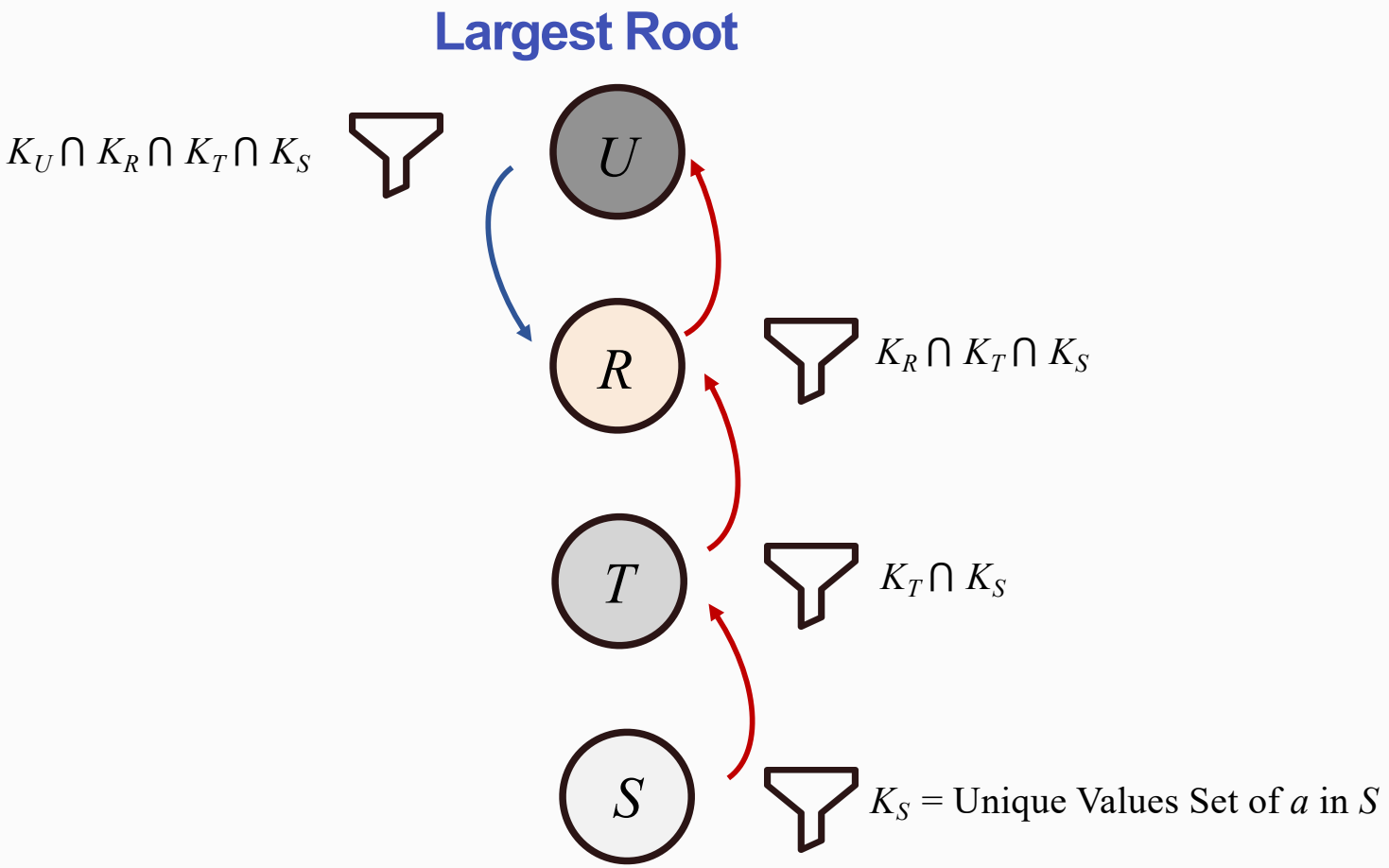
1. Redundant Filter Builds



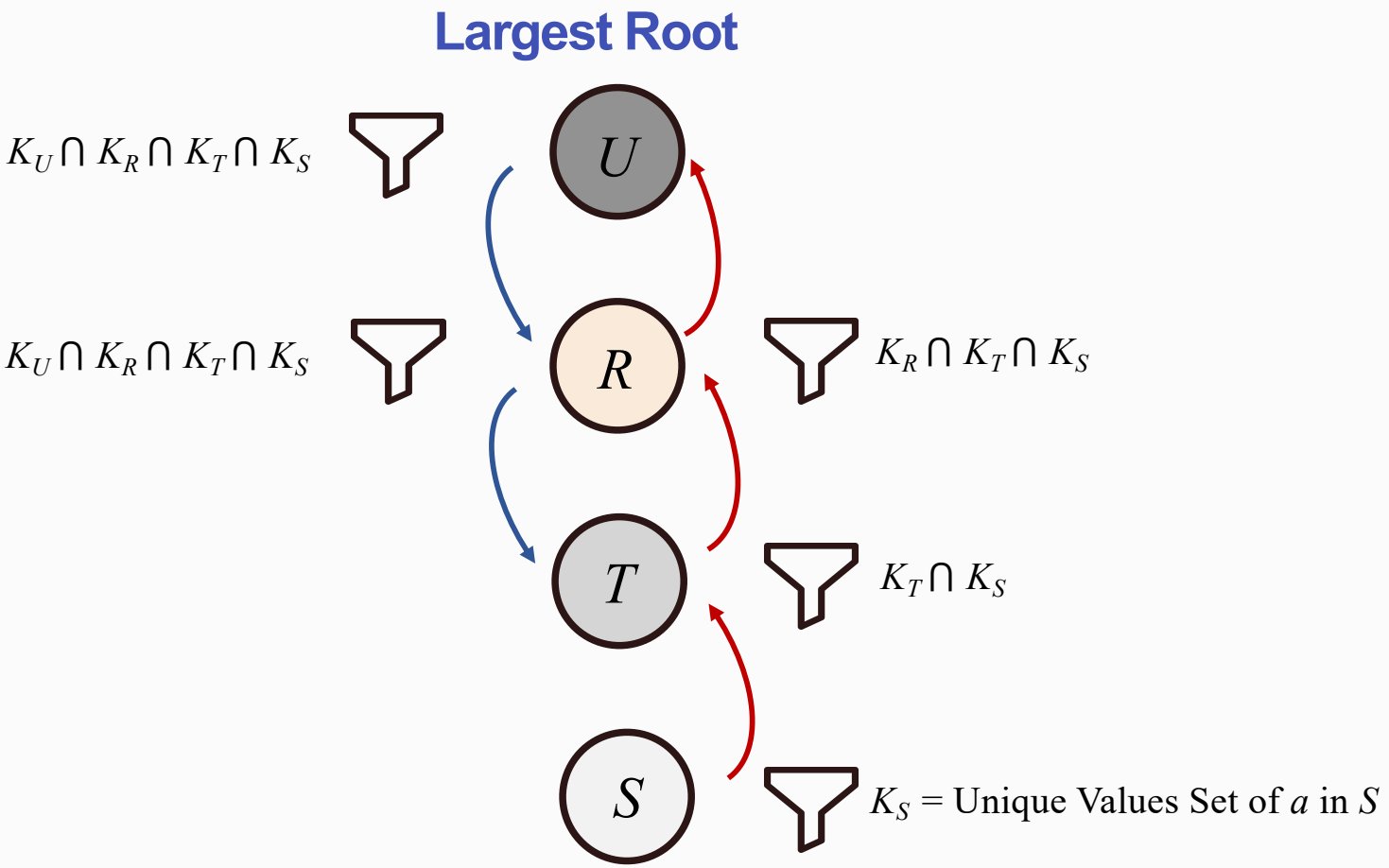
1. Redundant Filter Builds



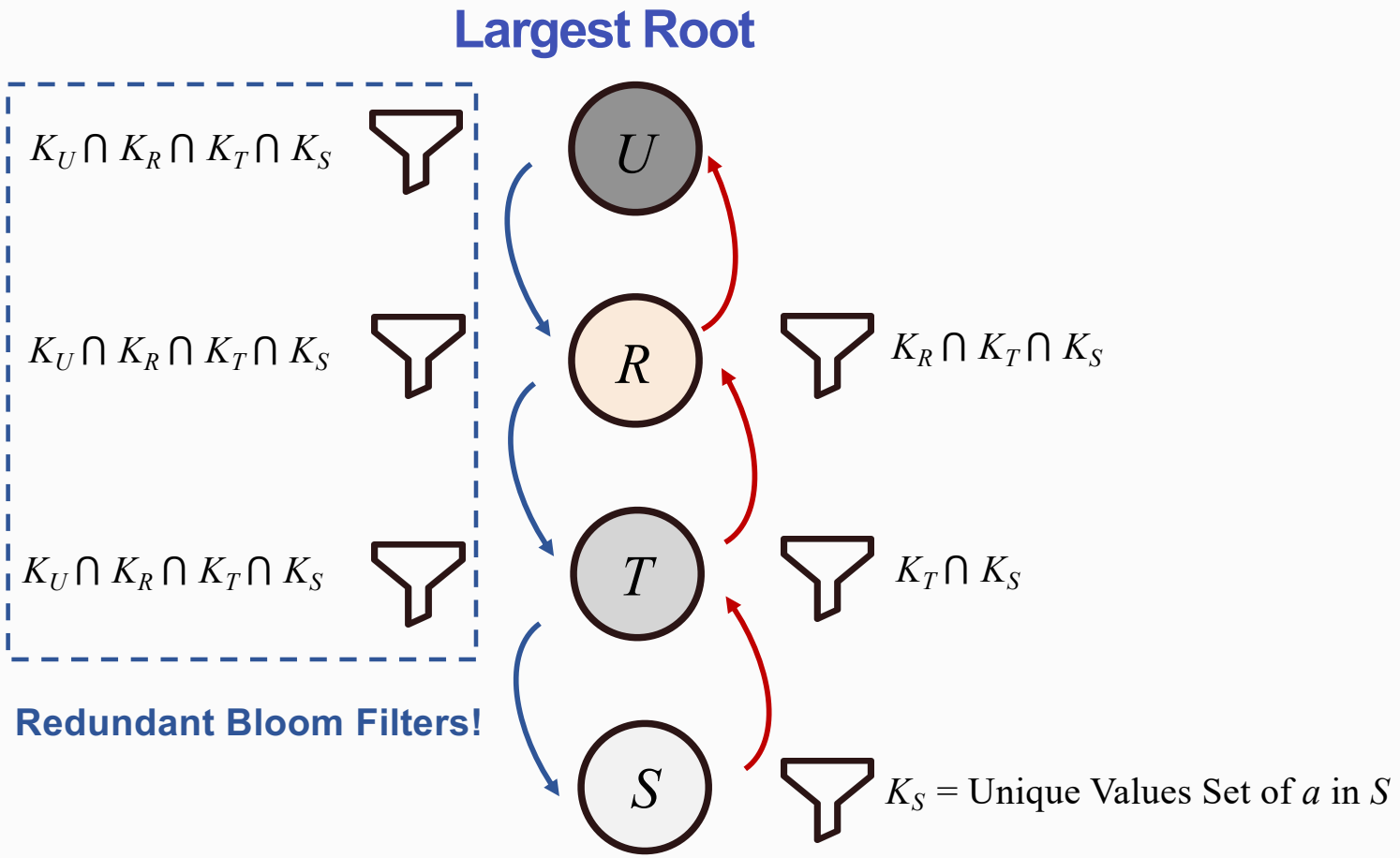
1. Redundant Filter Builds



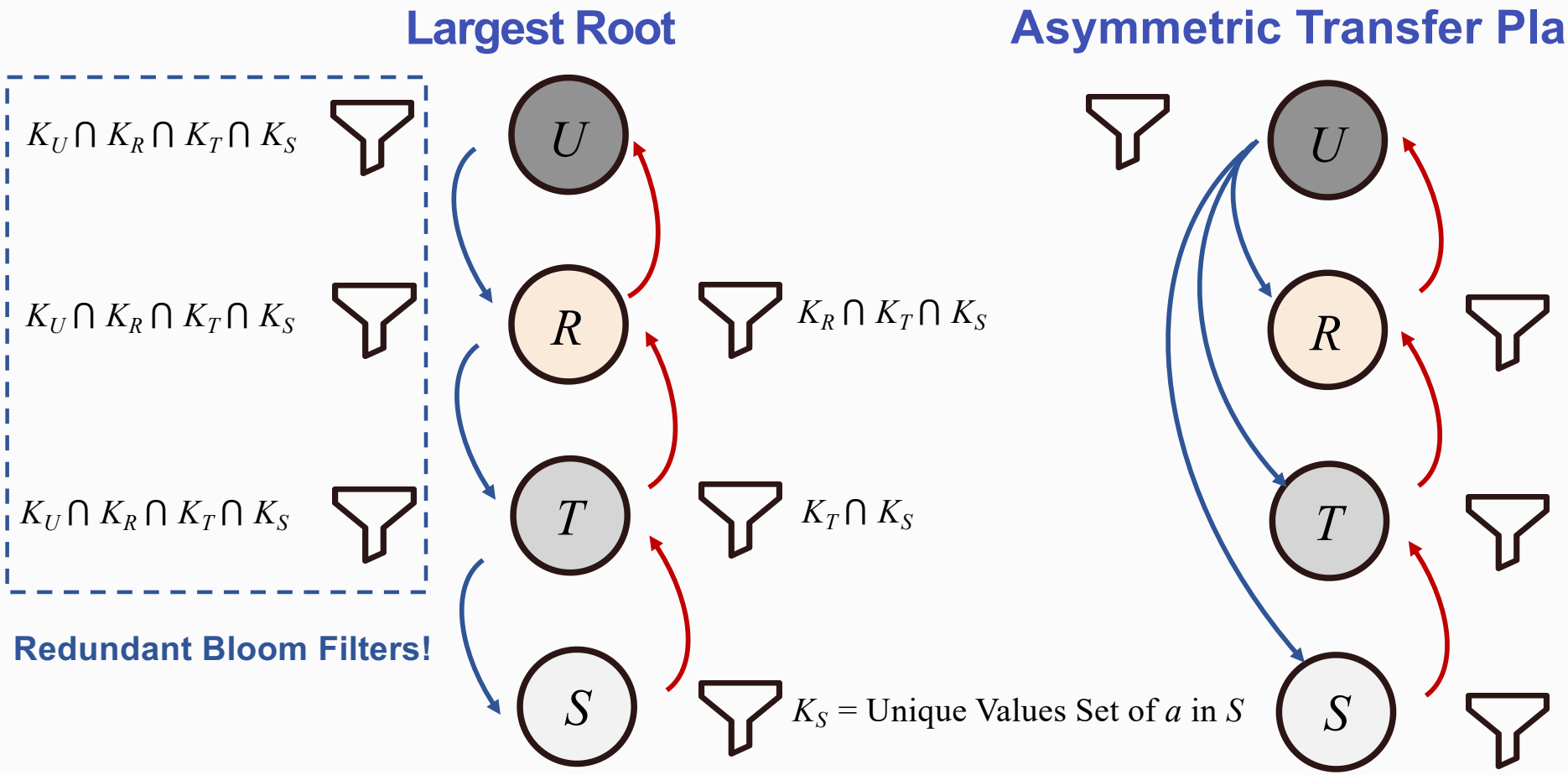
1. Redundant Filter Builds



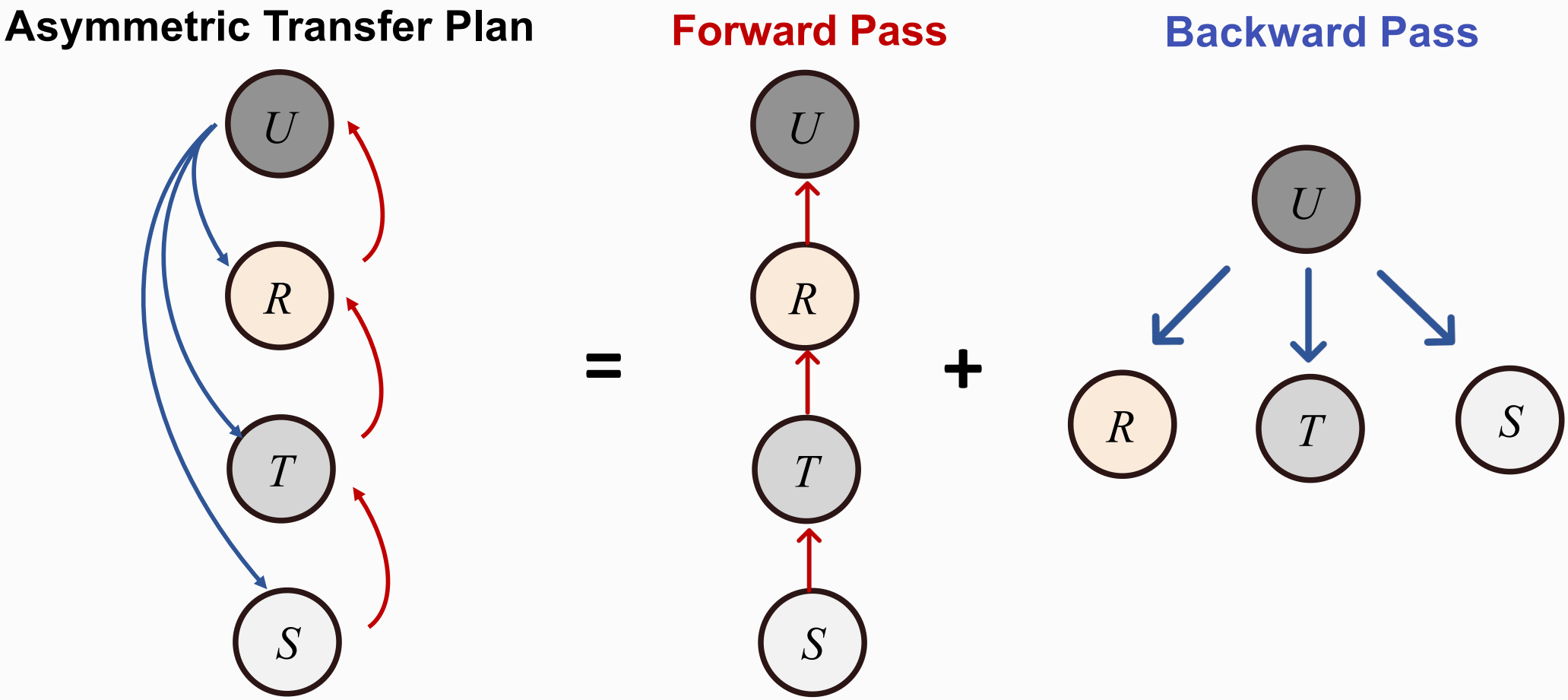
1. Redundant Filter Builds



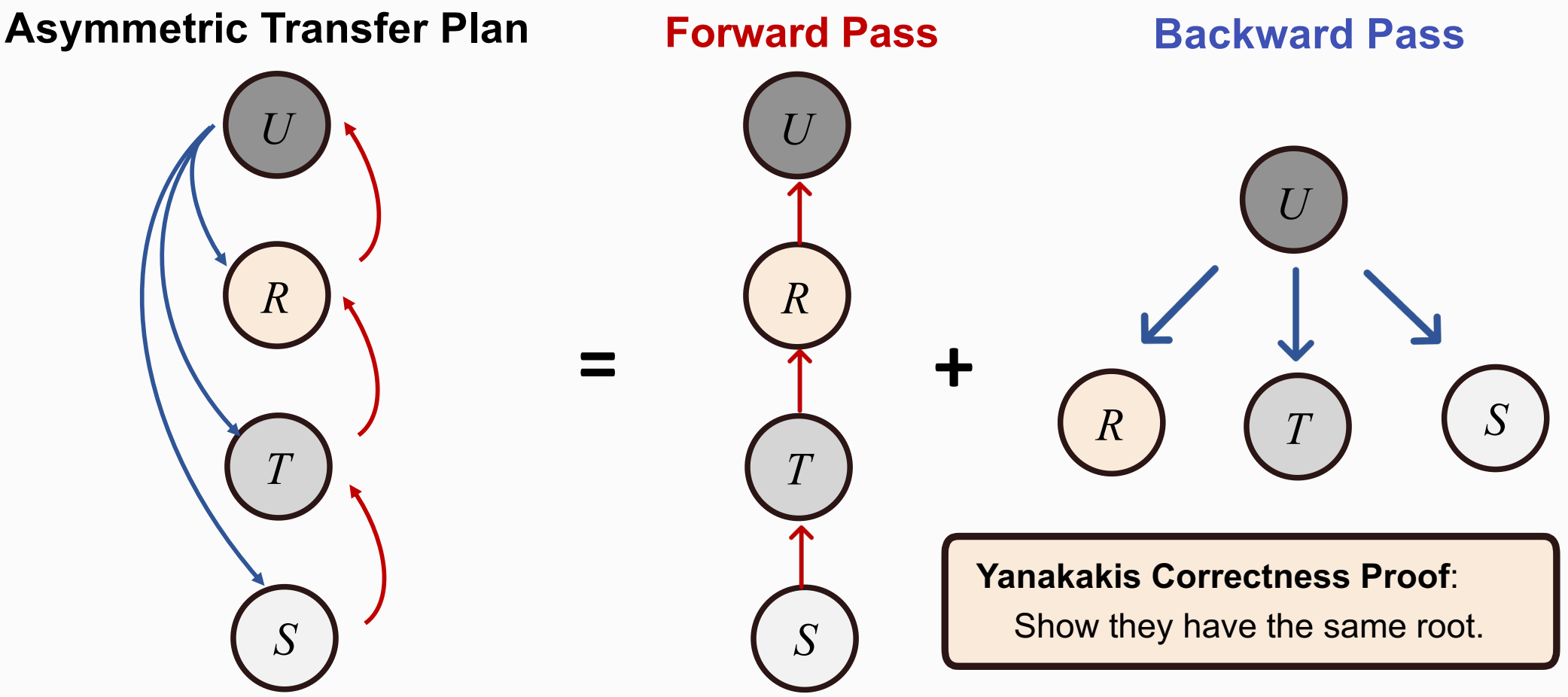
Asymmetric Transfer Plan



Asymmetric Transfer Plan



Asymmetric Transfer Plan

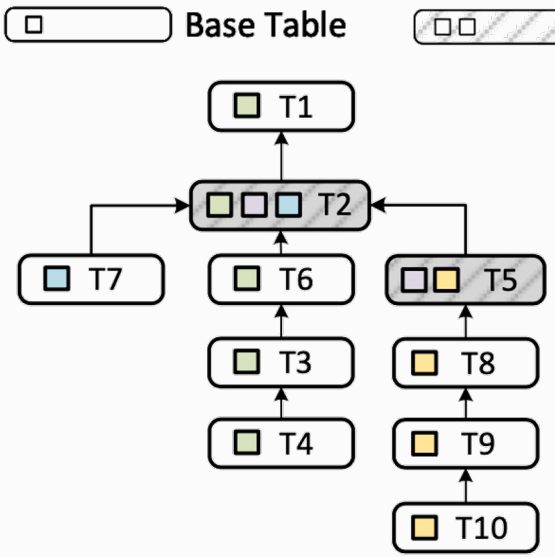


Asymmetric Transfer Plan

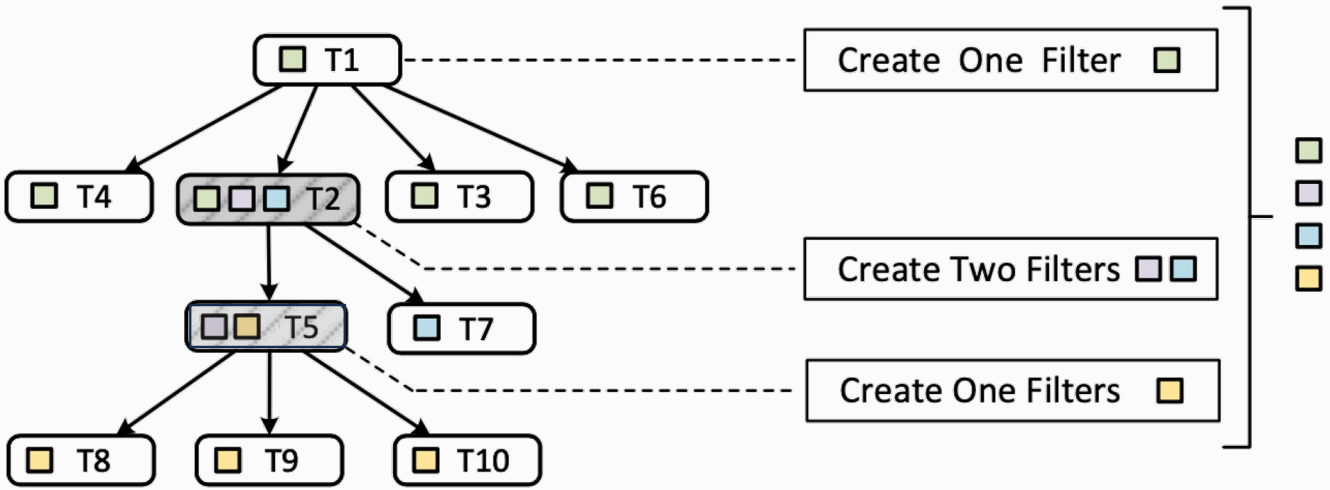
A deep, **chaining tree** is ideal for the forward pass,
whereas a wide, **broadcast tree** is best for the backward pass.

Asymmetric Transfer Plan

A deep, **chaining tree** is ideal for the forward pass,
whereas a wide, **broadcast tree** is best for the backward pass.



Forward Pass: Chaining Tree

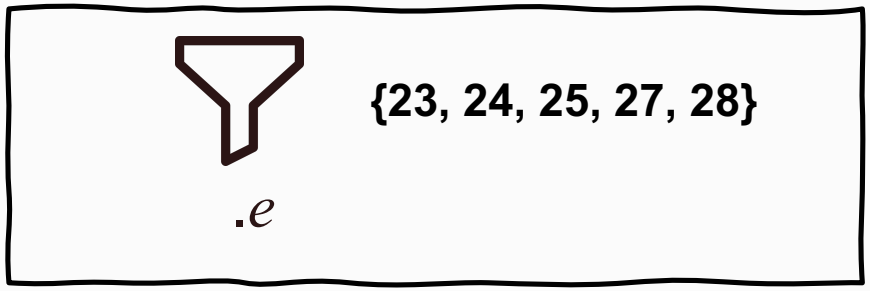


Backward Pass: Broadcast Tree

2. Storage-Format-Unaware Filtering

Bloom Filter cannot help skip row groups.

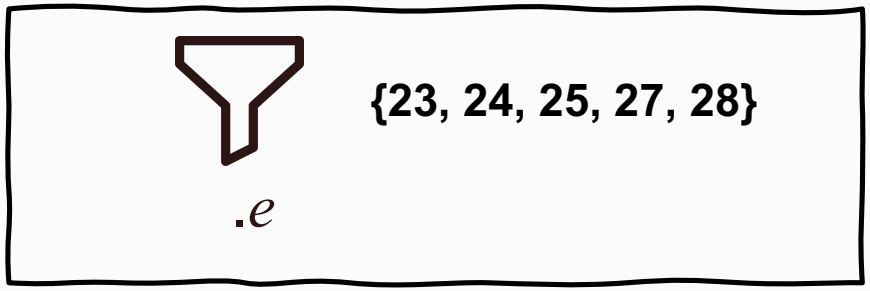
	a	c	e
Row Group 1	2	100	21
	4	100	22
Row Group 2	2	200	25
	1	100	26
Row Group 3	4	500	27
	1	300	31



2. Storage-Format-Unaware Filtering

Bloom Filter cannot help skip row groups.

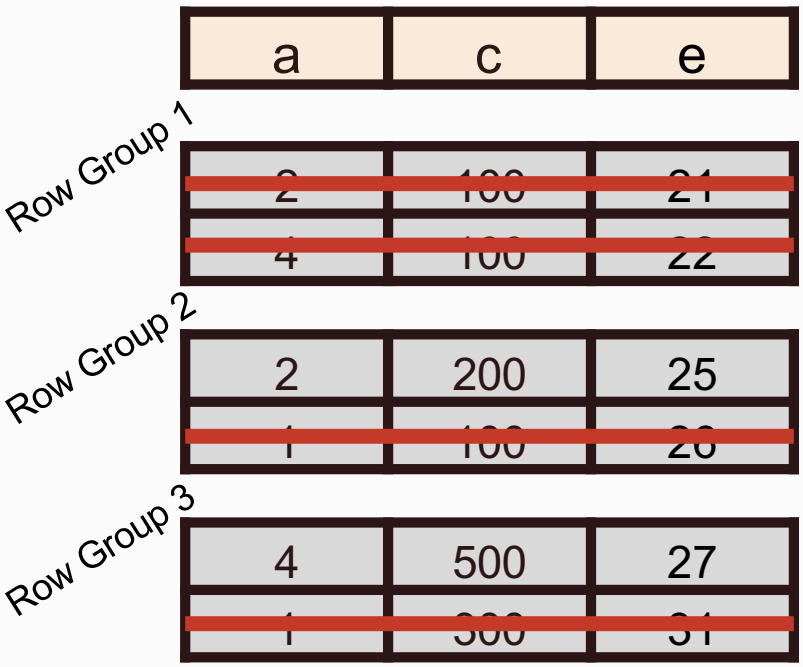
	a	c	e
Row Group 1	2	100	21
	4	100	22
Row Group 2	2	200	25
	1	100	26
Row Group 3	4	500	27
	1	300	31

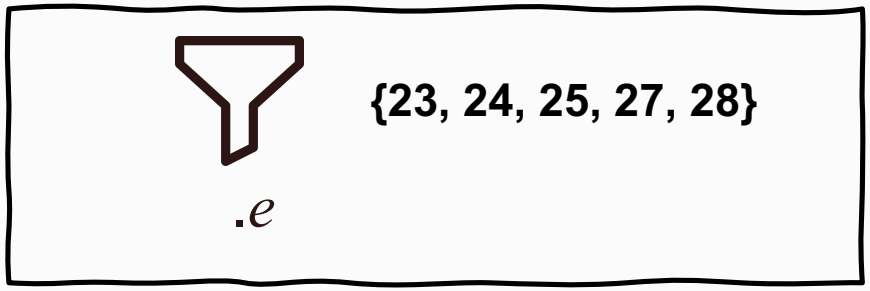


Stage 1: Data Scanning (#rows = 6)

2. Storage-Format-Unaware Filtering

Bloom Filter cannot help skip row groups.





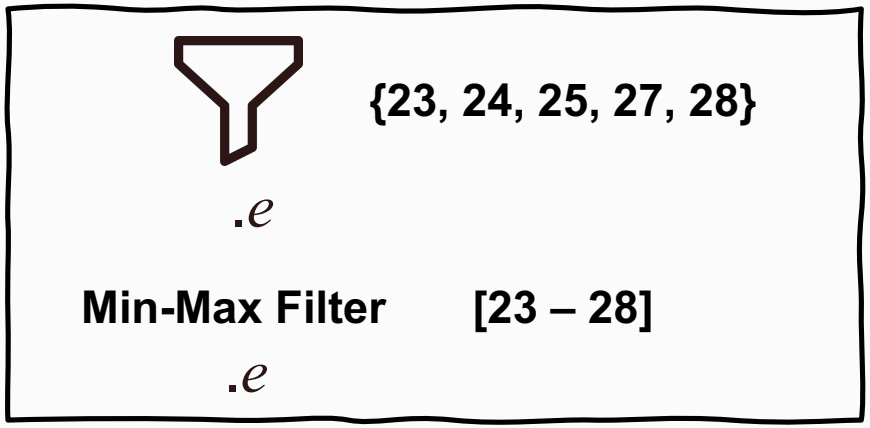
Stage 1: Data Scanning (#rows = 6)

Stage 2: Bloom Filter Probing (#rows = 6)

Cascade Filter

Cascade Filter = Min-max Filter + Bloom Filter + others (optional)

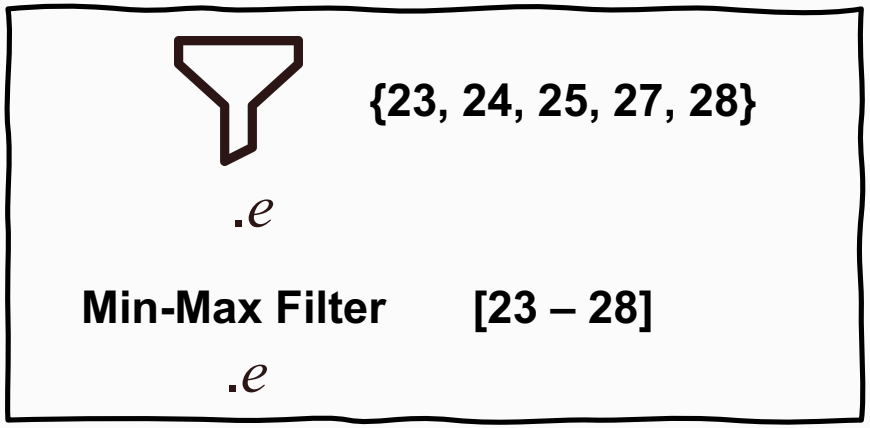
	a	c	e
Row Group 1	2	100	21
	4	100	22
Row Group 2	2	200	25
	1	100	26
Row Group 3	4	500	27
	1	300	31



Cascade Filter

Cascade Filter = Min-max Filter + Bloom Filter + others (optional)

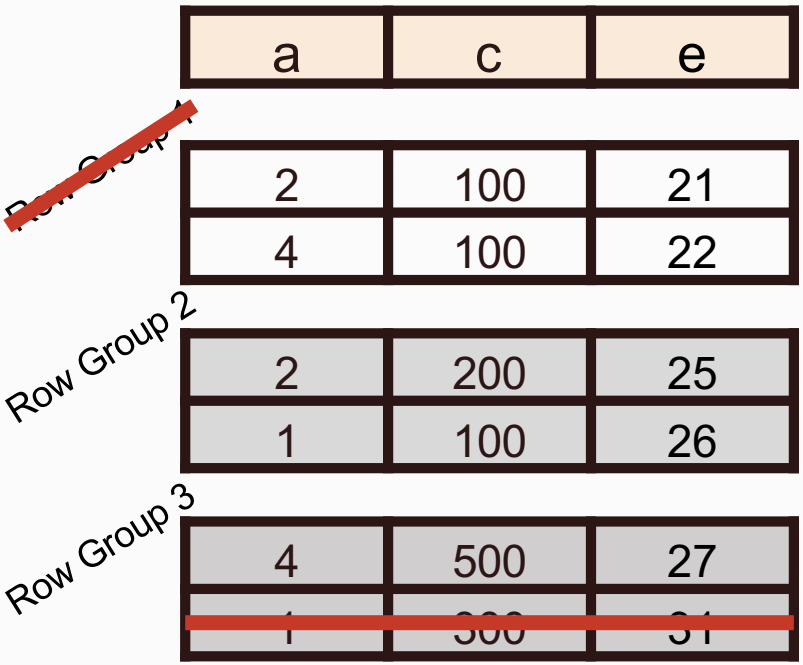
a	c	e
Row Group 1		
2	100	21
4	100	22
Row Group 2		
2	200	25
1	100	26
Row Group 3		
4	500	27
1	300	31

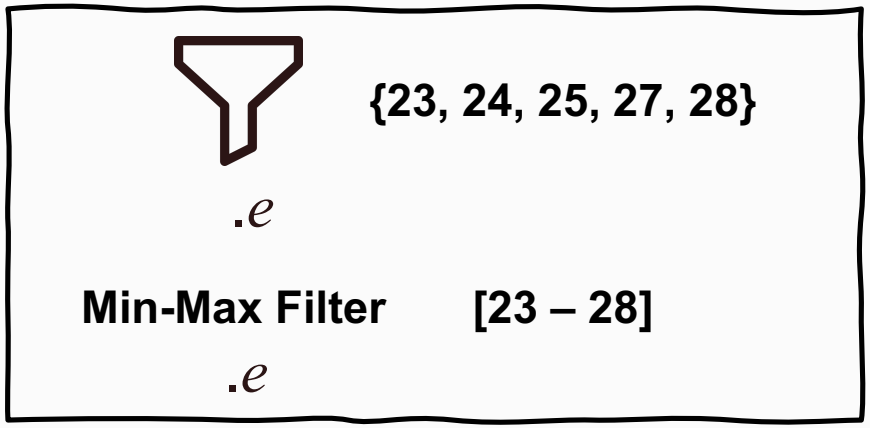


Stage 1: Data Scanning (#rows = 4)

Cascade Filter

Cascade Filter = Min-max Filter + Bloom Filter + others (optional)



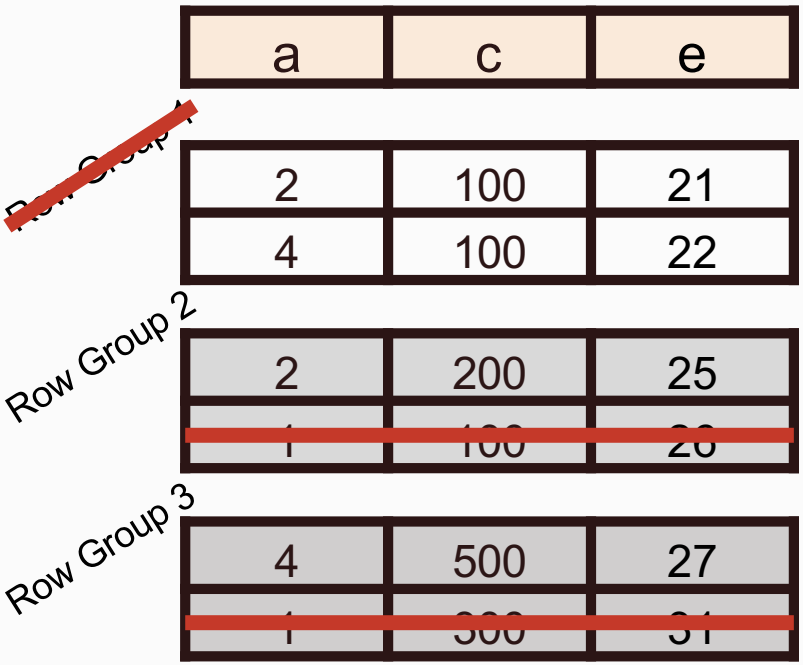


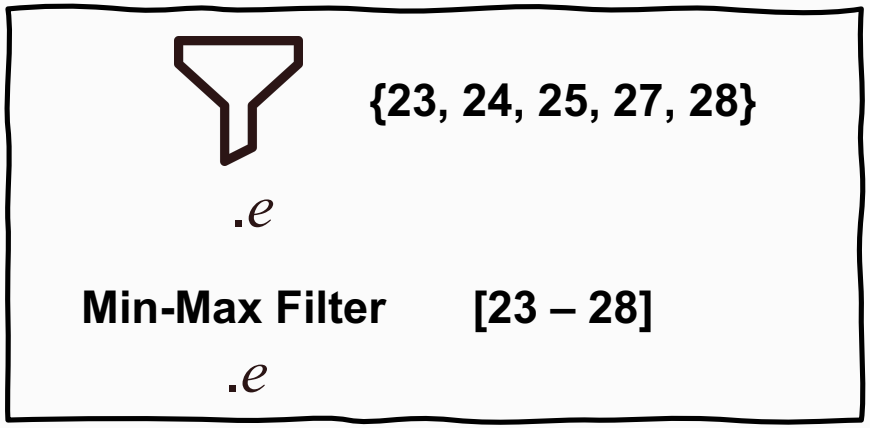
Stage 1: Data Scanning (#rows = 4)

Stage 2: Min-Max Probing (#rows = 4)

Cascade Filter

Cascade Filter = Min-max Filter + Bloom Filter + others (optional)





Stage 1: Data Scanning (#rows = 4)

Stage 2: Min-Max Probing (#rows = 4)

Stage 3: Bloom Filter Probing (#rows = 3)

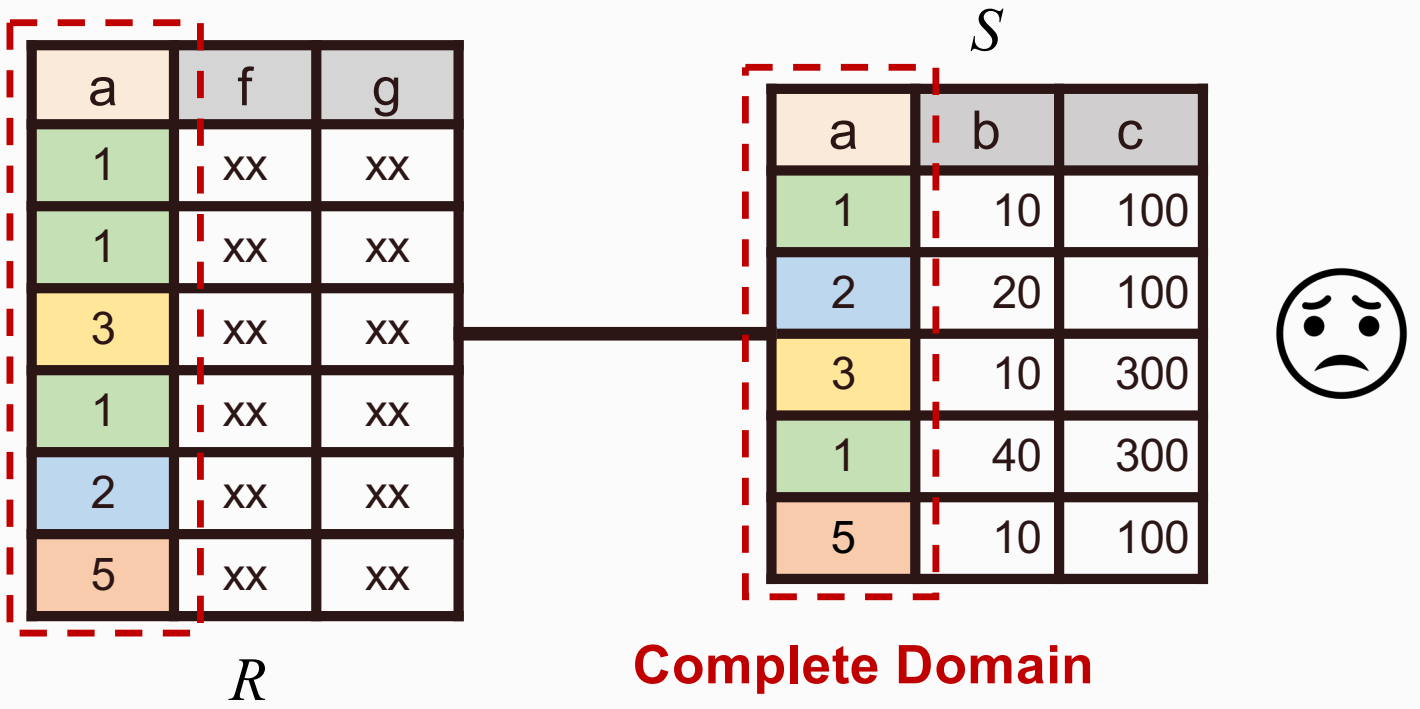
Non-Reducible Queries

If all base table rows already contribute to the join results before filtering, RPT causes regressions.

Non-Reducible Queries

If all base table rows already contribute to the join results before filtering, RPT causes regressions.

```
SELECT *
FROM   R, S
WHERE  R.a = S.a
```



Dynamic Execution

Goal: collect "Information" from efficiently filtered tables only

A Table is efficiently filtered if and only if :

Filterable: It has at least one user-defined predicate.

Selective: Its predicates are selective.

Runtime Estimation

Transfer Plan Modification

Dynamic Execution

Goal: collect "Information" from efficiently filtered tables only

A Table is efficiently filtered if and only if :

Filterable: It has at least one user-defined predicate.

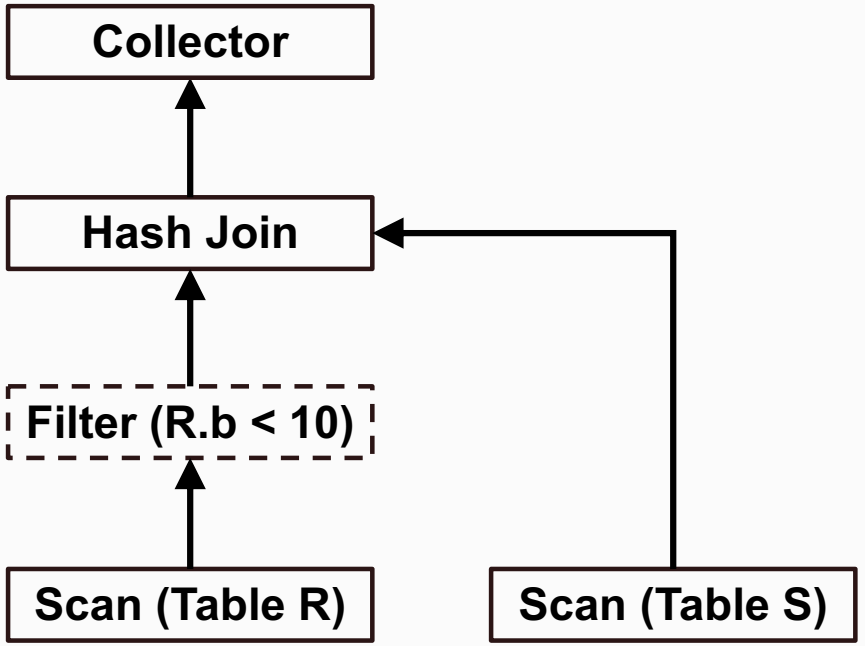
Selective: Its predicates are selective.

Runtime Estimation

Transfer Plan Modification
(refer to the paper)

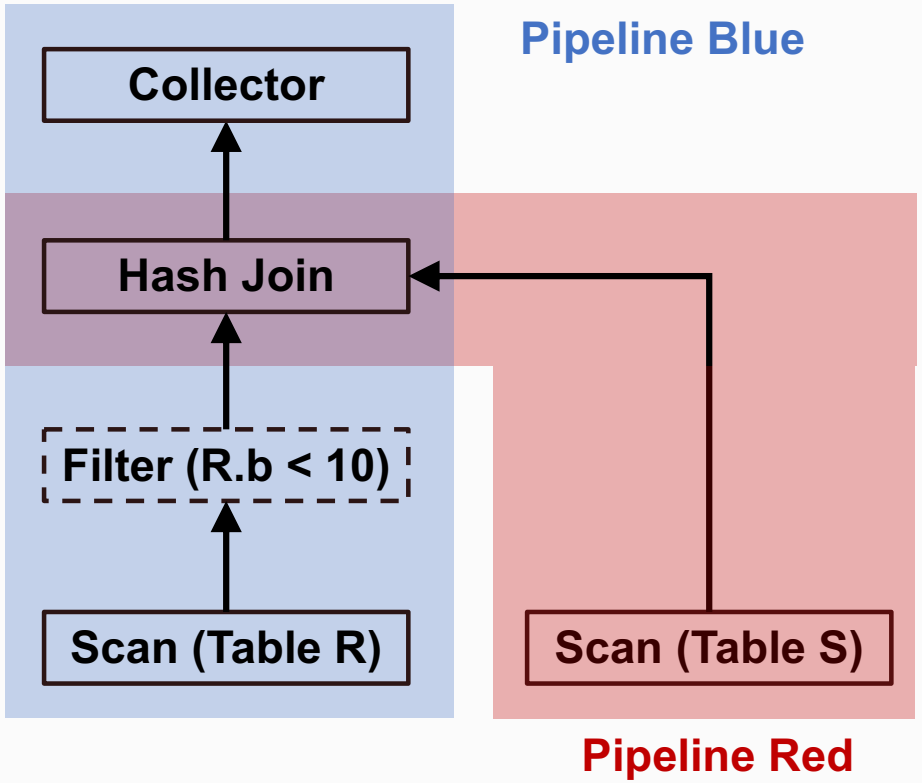
Estimate Selectivity at Runtime

```
SELECT *
FROM   R, S
WHERE  R.a = S.a
AND    R.b < 10
```



Estimate Selectivity at Runtime

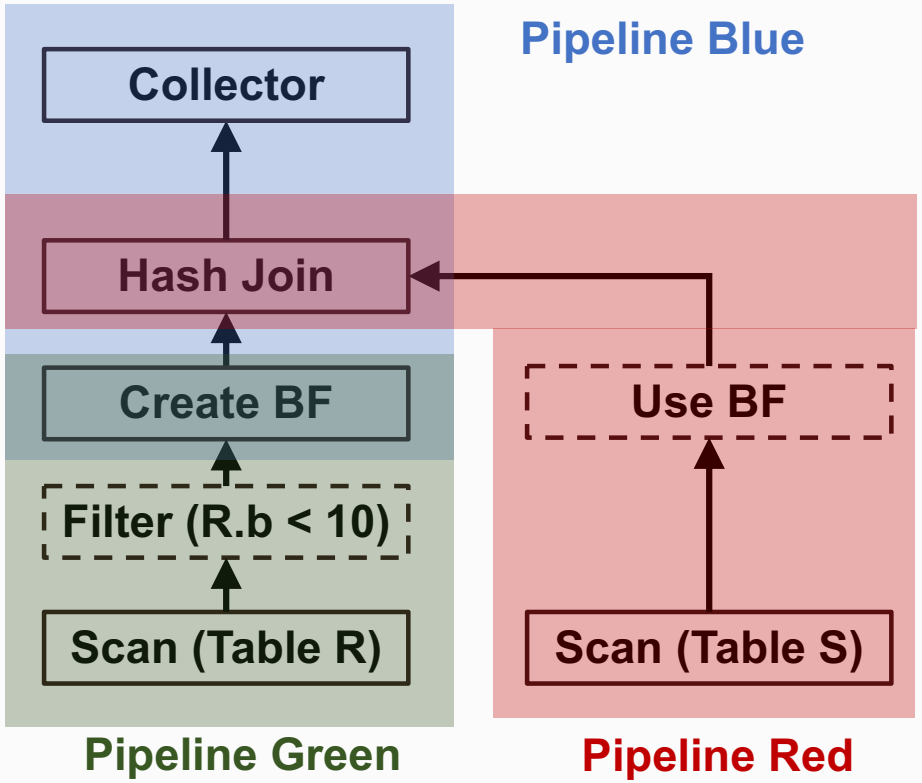
```
SELECT *
FROM   R, S
WHERE  R.a = S.a
AND    R.b < 10
```



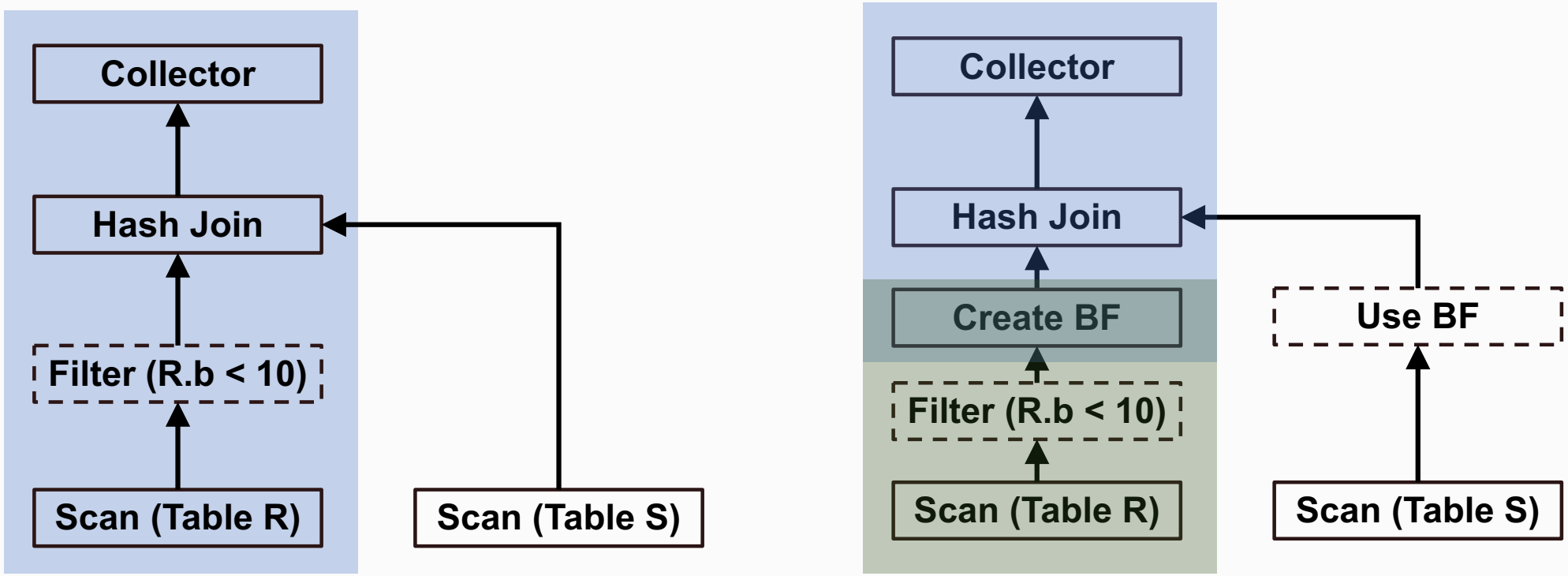
Estimate Selectivity at Runtime

```
SELECT *
FROM   R, S
WHERE  R.a = S.a
AND    R.b < 10
```

To Create Runtime Filters, we Split the Pipeline



Estimate Selectivity at Runtime



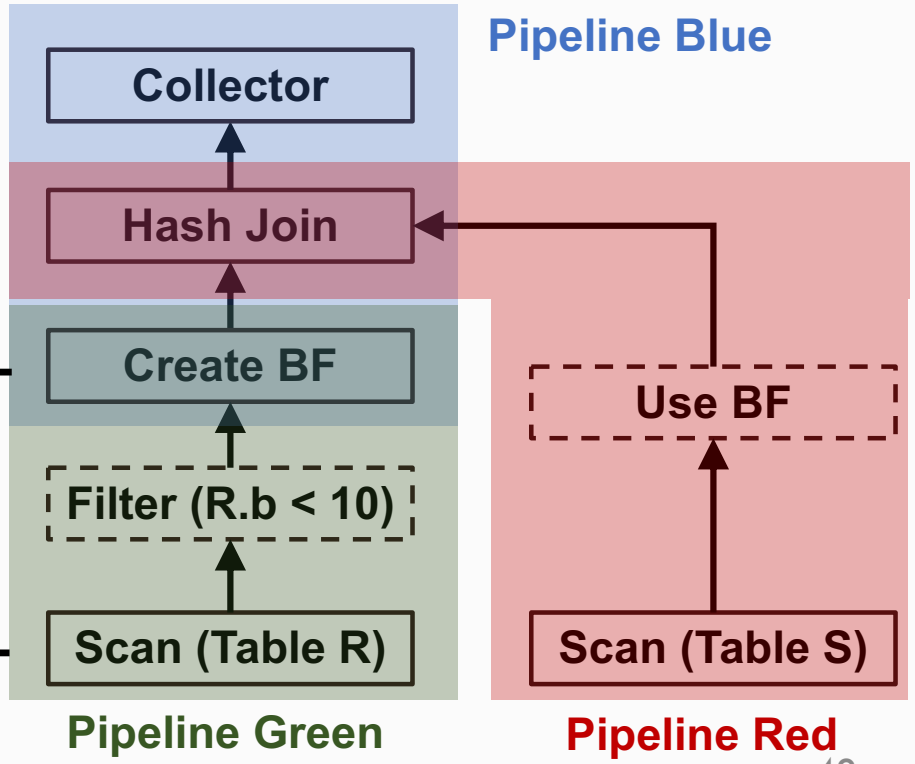
One CreateBF = One Table Materlization

Estimate Selectivity at Runtime

Monitor chunk statistics at runtime to estimate selectivity

② **Check Selectivity After** Receiving ≥10K Rows
with Sel = 10K / #Input

① **Count Input Rows Before Filtering**

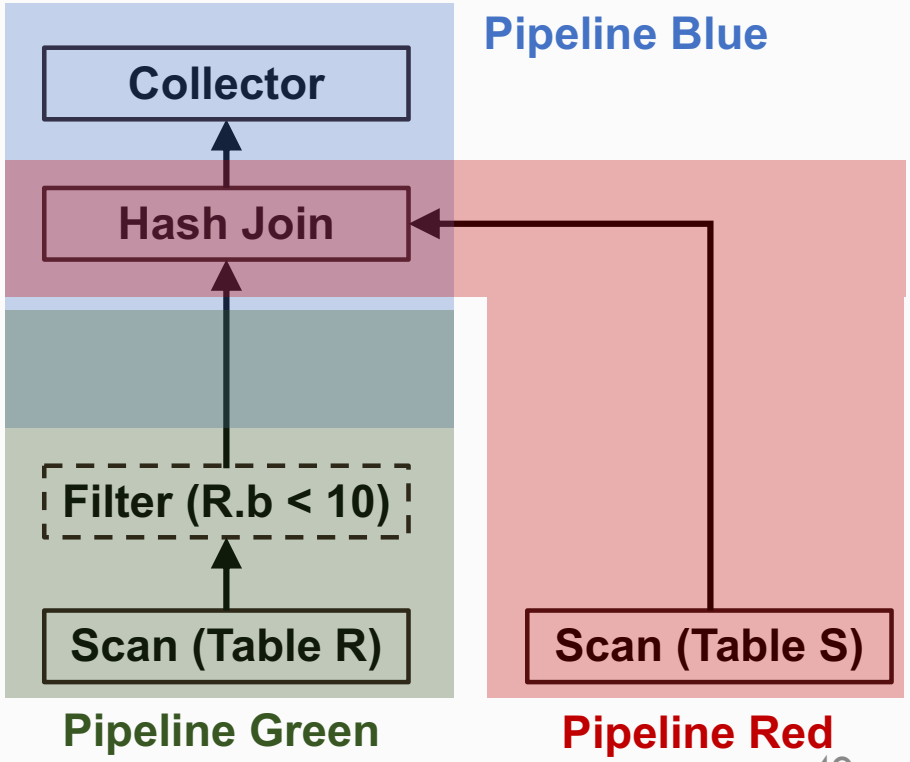


Estimate Selectivity at Runtime

Monitor chunk statistics at runtime to estimate selectivity

If the estimated selectivity is not desired,

1. **Cancel** BF construction

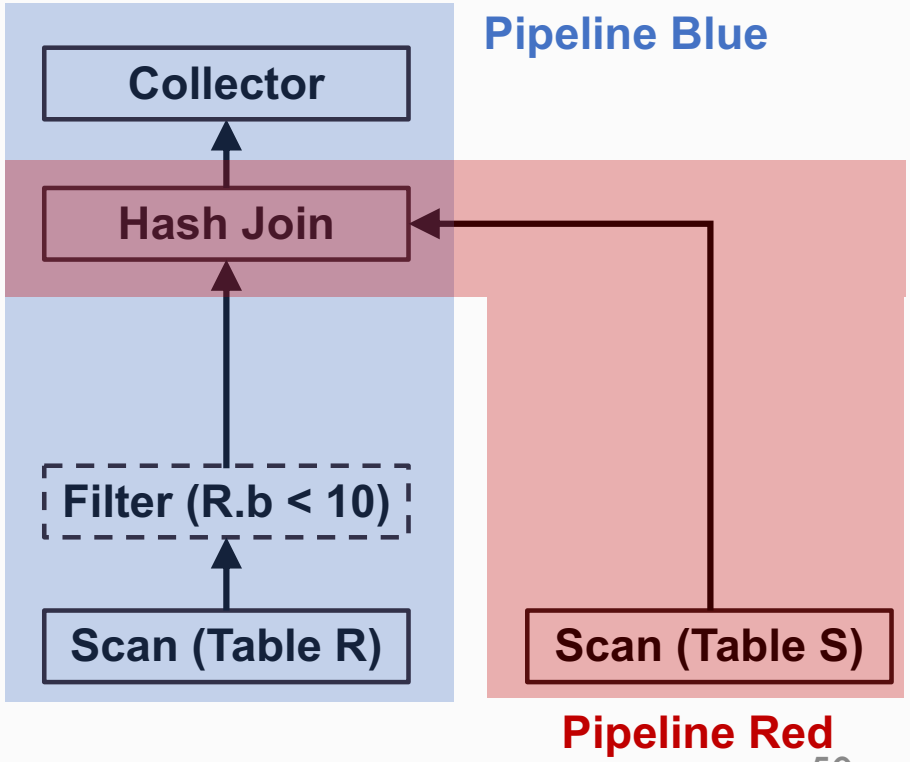


Estimate Selectivity at Runtime

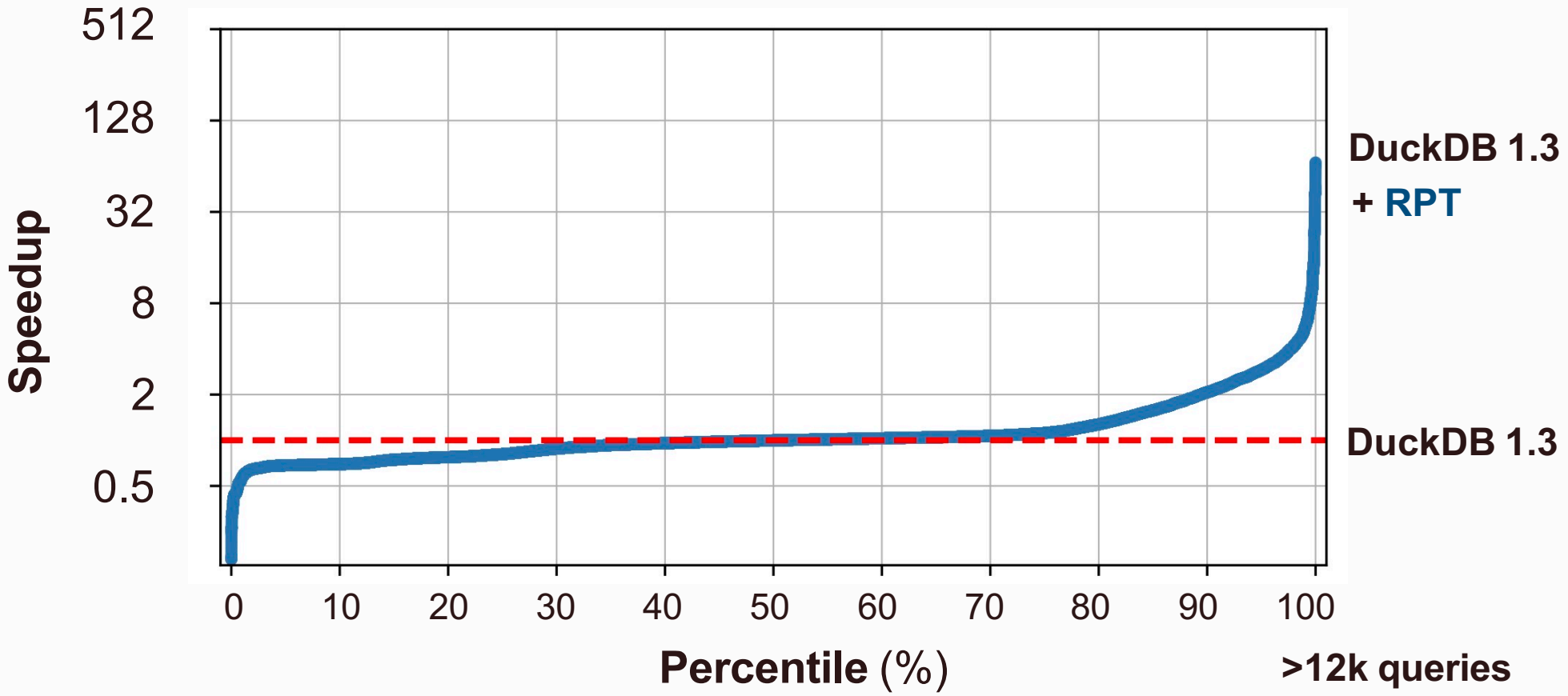
Monitor chunk statistics at runtime to estimate selectivity

If the estimated selectivity is not desired,

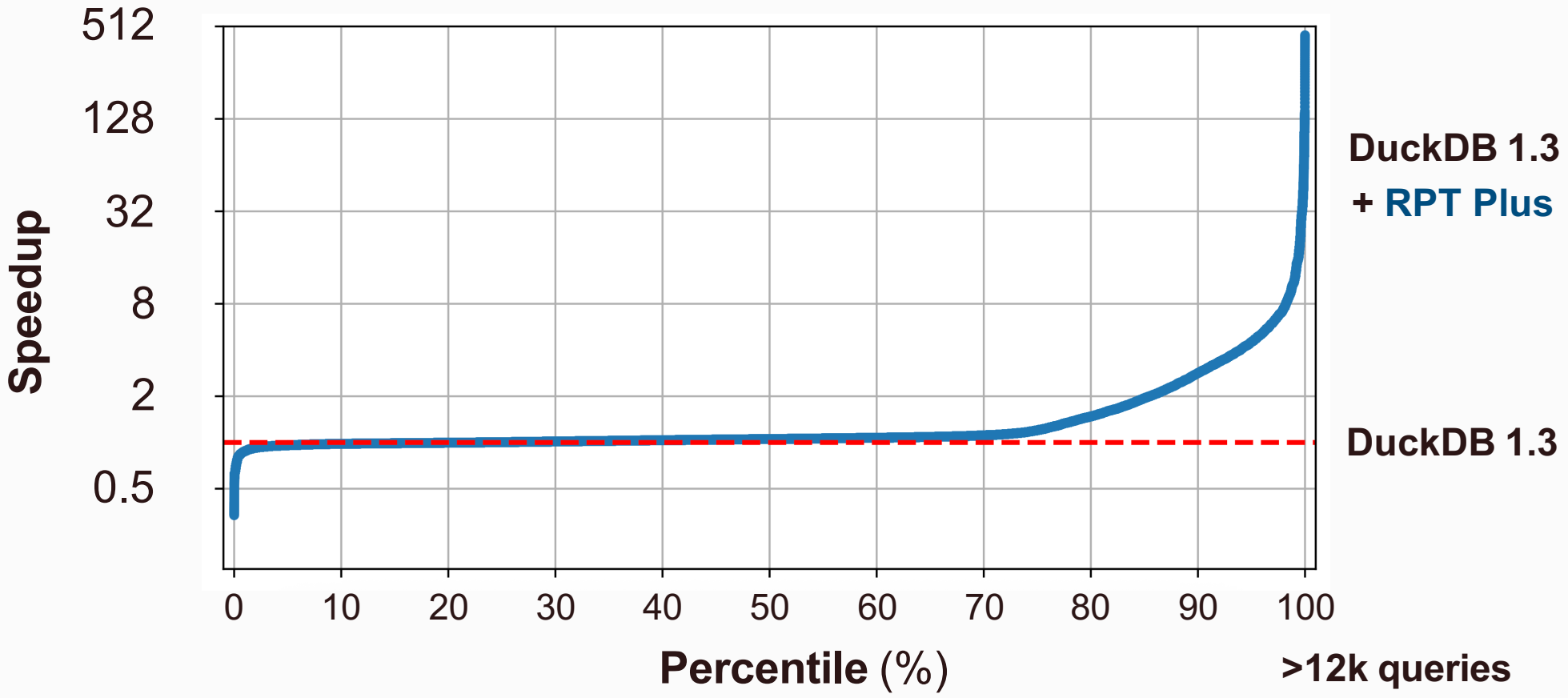
1. **Cancel** BF construction
2. **Move** operators across pipelines



SQLStorm: RPT vs. DuckDB 1.3



SQLStorm: RPT Plus vs. DuckDB 1.3



Takeaway

- Runtime filtering + data pruning = faster queries.
- Adapt at runtime, rather than trust cardinality estimates.
- Make predicate transfer a standalone execution layer.

Bloom: Beyond DuckDB

Towards Robust Join Acceleration for Every Execution Engine.

DuckDB Extension

Postgres Extension

Datafusion Extension

Same idea. Different engines. Robust speedup.