香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

Ack: Prof. Jignesh Patel @ CMU
Prof. Andy Pavlo @CMU

CSC3170
11: Join Algorithms

Chenhao Ma
School of Data Science
The Chinese University of Hong Kong, Shenzhen

Reminder

- **Assignment 2 will be due on 11:59pm Oct 27th**
- **Project will be due on 11:59pm Nov 23rd**

Why Do We Need To Join?

- We normalize tables in a relational database to avoid unnecessary repetition of information.

- We then use the **join operator** to reconstruct the original tuples without any information loss.

Join Algorithms

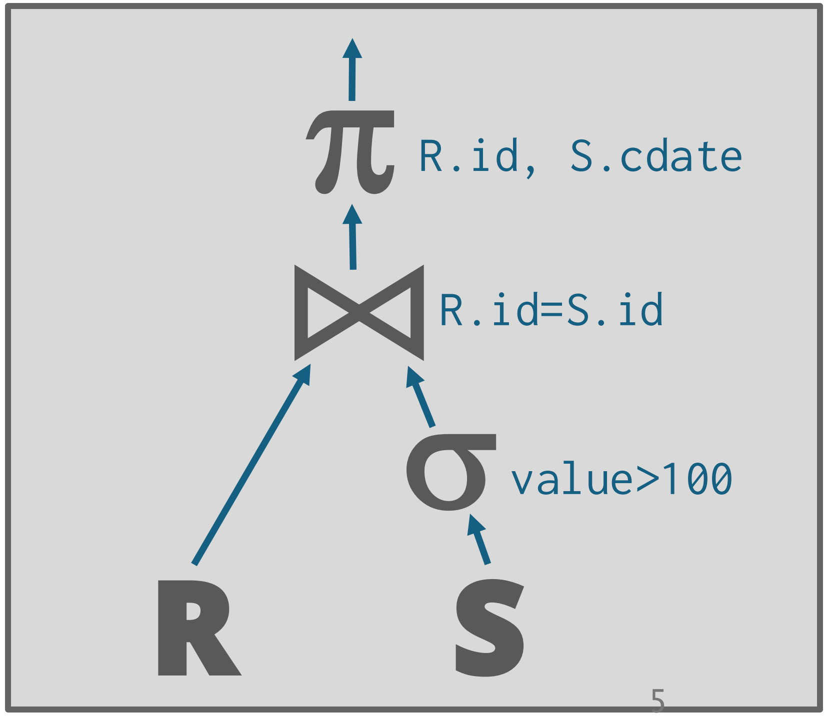
- We will focus on performing binary joins (two tables) using **inner equijoin** algorithms.
  - These algorithms can be tweaked to support other joins.
  - Multi-way joins exist primarily in research literature.

- In general, we want the smaller table to always be the left table ("outer table") in the query plan.
  - The optimizer will (try to) figure this out when generating the physical plan.

Query Plan

- The operators are arranged in a tree.
- Data flows from the leaves of the tree up towards the root.
  - We will discuss the granularity of the data movement next week.
- The output of the root node is the result of the query.

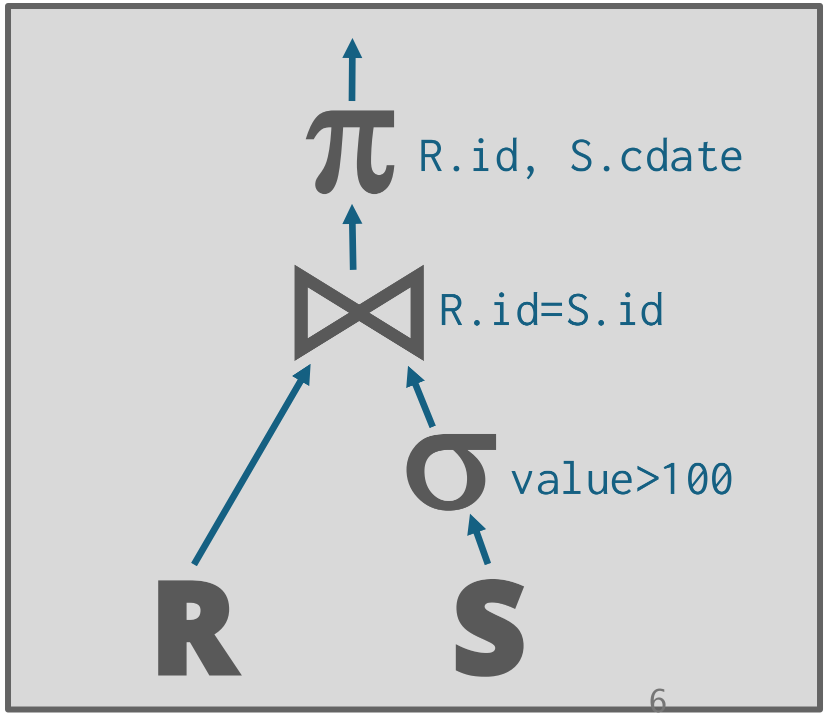
```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```



Join Operators

- **Decision #1: Output**
  - What data does the join operator emit to its parent operator in the query plan tree?
- **Decision #2: Cost Analysis Criteria**
  - How do we determine whether one join algorithm is better than another?

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

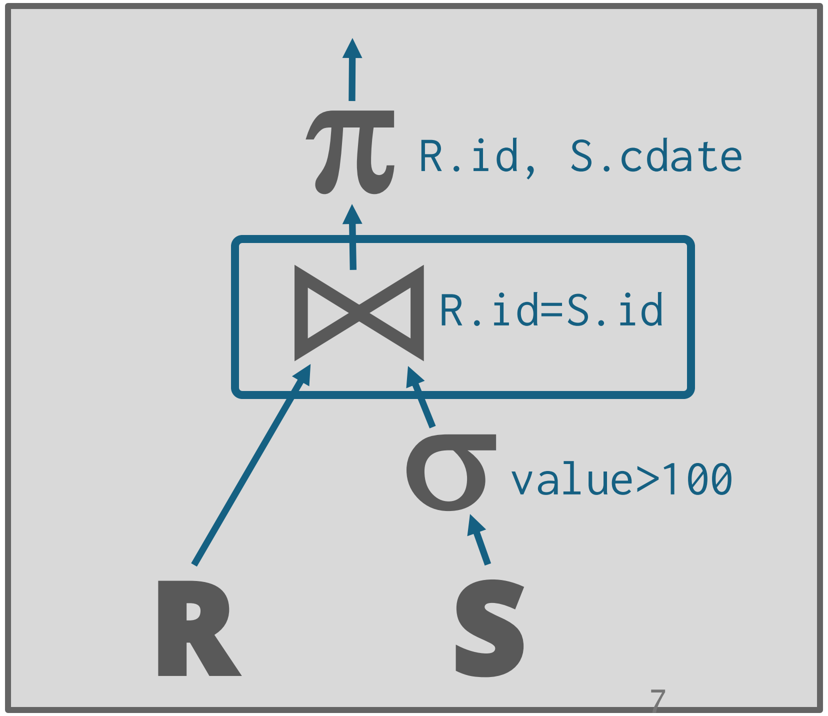


Operator Output

- For tuple `r∈R` and tuple `s∈S` that match on join attributes, concatenate `r` and `s` together into a new tuple.

- Output contents can vary:
  - Depends on processing model
  - Depends on storage model
  - Depends on data requirements in query

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```



SCHOOL OF DATA SCIENCE

Operator Output: Data

- **Early Materialization:**
  - Copy the values for the attributes in outer and inner tuples into a new output tuple.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Operator Output: Data

- **Early Materialization:**
  - Copy the values for the attributes in outer and inner tuples into a new output tuple.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R(id,name)

id	name
123	abc

S(id,value,cdate)

id	value	cdate
123	1000	10/26/2025
123	2000	10/26/2025

Operator Output: Data

- **Early Materialization:**
  - Copy the values for the attributes in outer and inner tuples into a new output tuple.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R(id,name)

id	name
123	abc

⋈

S(id,value,cdate)

id	value	cdate
123	1000	10/26/2025
123	2000	10/26/2025

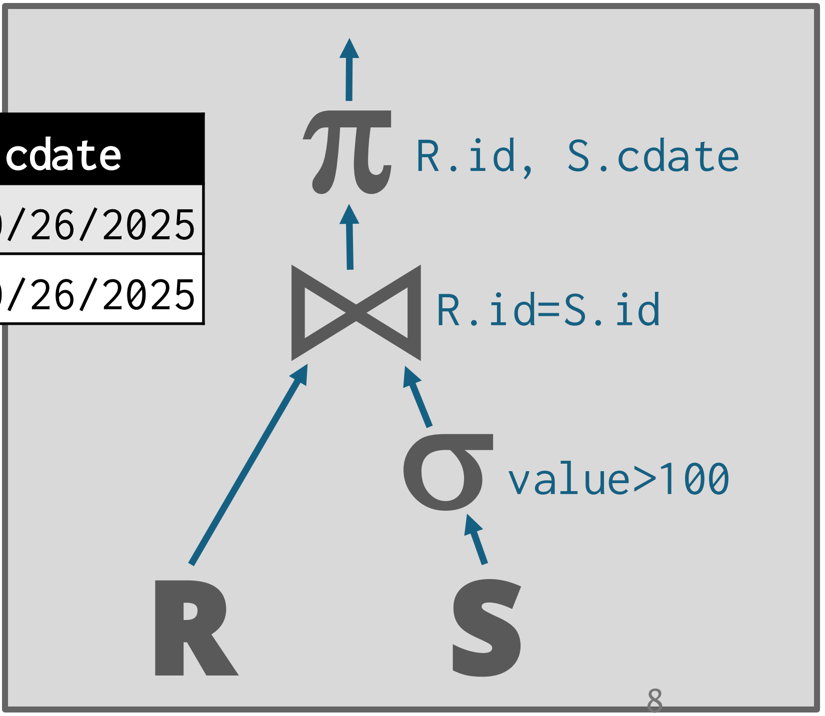
R.id	R.name	S.id	S.value	S.cdate
123	abc	123	1000	10/26/2025
123	abc	123	2000	10/26/2025

Operator Output: Data

- **Early Materialization:**
  - Copy the values for the attributes in outer and inner tuples into a new output tuple.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R.id	R.name	S.id	S.value	S.cdate
123	abc	123	1000	10/26/2025
123	abc	123	2000	10/26/2025

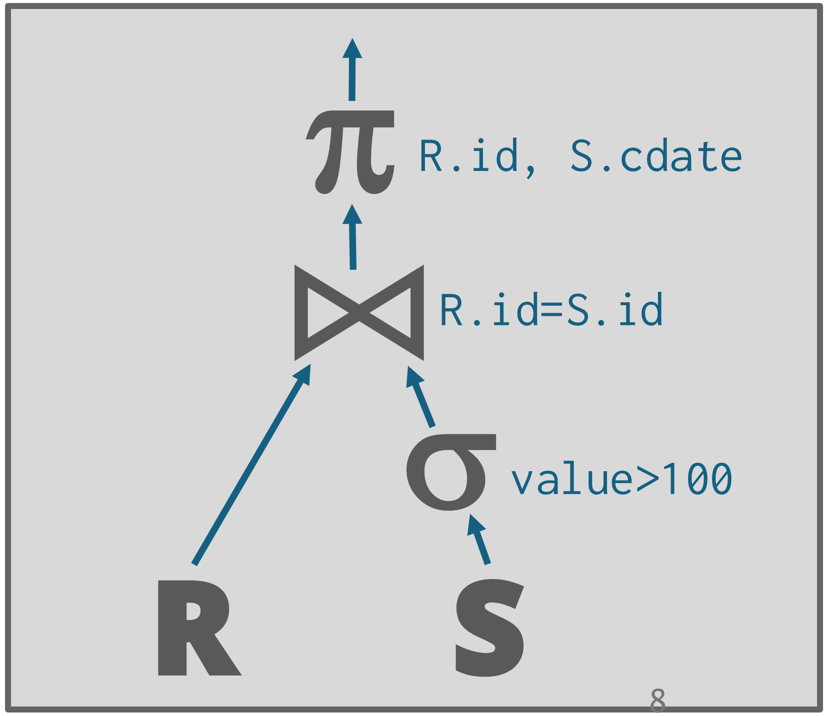


Operator Output: Data

- **Early Materialization:**
  - Copy the values for the attributes in outer and inner tuples into a new output tuple.

- Subsequent operators in the query plan never need to go back to the base tables to get more data.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```



Operator Output: Record IDs

- **Late Materialization:**
  - Only copy the joins keys along with the Record IDs of the matching tuples.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R(id,name)

id	name
123	abc

S(id,value,cdate)

id	value	cdate
123	1000	10/26/2025
123	2000	10/26/2025

Operator Output: Record IDs

- **Late Materialization:**
  - Only copy the joins keys along with the Record IDs of the matching tuples.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R(id,name)

id	name
123	abc

S(id,value,cdate)

id	value	cdate
123	1000	10/26/2025
123	2000	10/26/2025

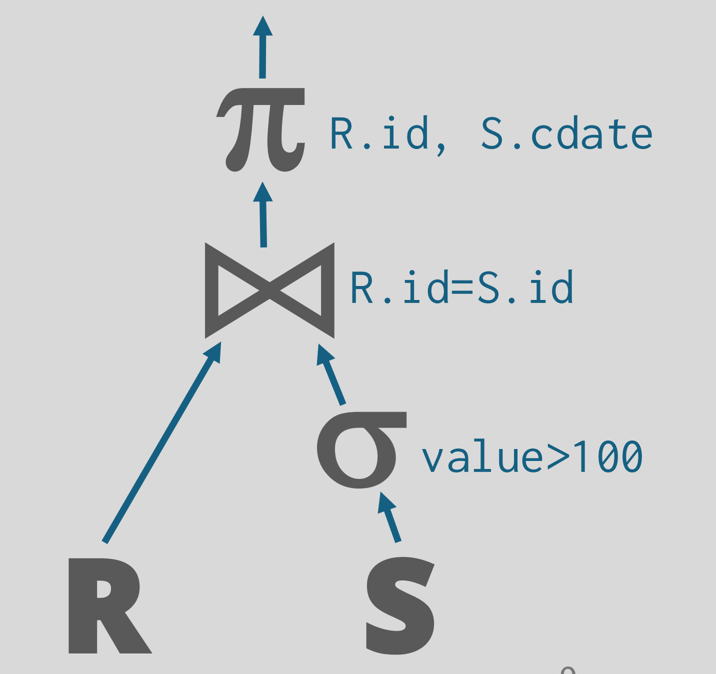
R.id	R.RID	S.id	S.RID
123	R.###	123	S.###
123	R.###	123	S.###

Operator Output: Record IDs

- **Late Materialization:**
  - Only copy the joins keys along with the Record IDs of the matching tuples.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R.id	R.RID	S.id	S.RID
123	R.###	123	S.###
123	R.###	123	S.###



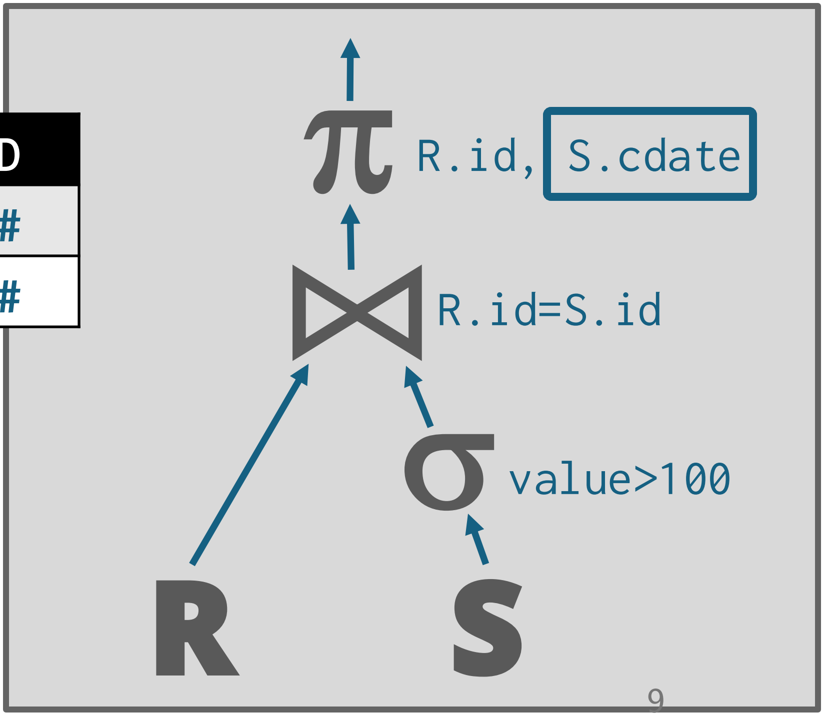
SCHOOL OF DATA SCIENCE

Operator Output: Record IDs

- **Late Materialization:**
  - Only copy the joins keys along with the Record IDs of the matching tuples.

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

R.id	R.RID	S.id	S.RID
123	R.###	123	S.###
123	R.###	123	S.###

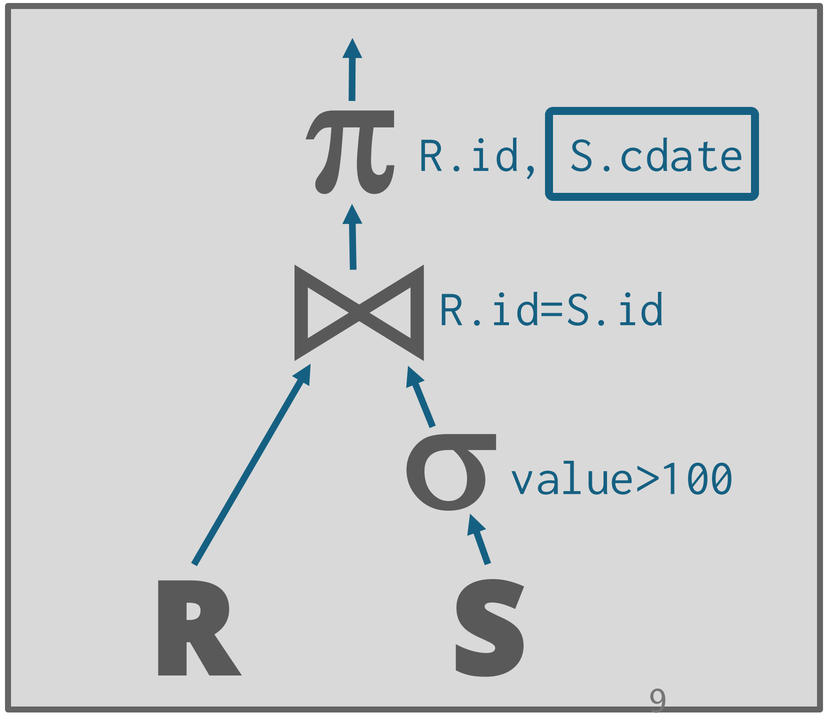


Operator Output: Record IDs

- **Late Materialization:**
  - Only copy the joins keys along with the Record IDs of the matching tuples.

- Ideal for column stores because the DBMS does not copy data that is not needed for the query.

```sql
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```



SCHOOL OF
DATA SCIENCE

Cost Analysis Criteria

- Assume:
  - M pages in table **R**, m tuples in **R**
  - N pages in table **S**, n tuples in **S**

```sql
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

- **Cost Metric: # of I/Os to compute join**

- We ignore overall output costs because it depends on the data and is the same for all algorithms .

Join vs Cross-Product

- R⋈S is the most common operation and thus must be carefully optimized.
- R×S followed by a selection is inefficient because the cross-product is large.

- There are many algorithms for reducing join cost, but no algorithm works well in all scenarios.

Join Algorithms

- Nested Loop Join
  - Naïve
  - Block
  - Index
- Sort-Merge Join
- Hash Join
  - Simple
  - GRACE (Externally Partitioned)
  - Hybrid

Nested Loop Join

Naïve Nested Loop Join

```
foreach tuple r ∈ R:
  foreach tuple s ∈ S:
    if r and s match then emit
```

R(id,name)

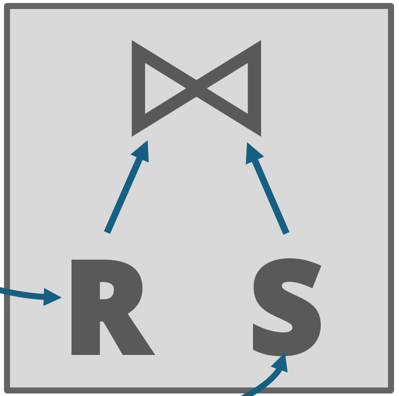
id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

Naïve Nested Loop Join

```
foreach tuple r ∈ R:    ← Outer
  foreach tuple s ∈ S:  ← Inner
    if r and s match then emit
```



R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

Naïve Nested Loop Join

- Why is this algorithm bad?
  - For every tuple in **R**, it scans **S** once

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

Naïve Nested Loop Join

- Why is this algorithm bad?
  - For every tuple in **R**, it scans **S** once
- **Cost: M + (m · N)**

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

M pages
m tuples

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Naïve Nested Loop Join

- Example database:
  - **Table R**: $M = 1000$, $m = 100{,}000$
  - **Table S**: $N = 500$, $n = 40{,}000$

Naïve Nested Loop Join

- Example database:
  - **Table R**: $M = 1000$, $m = 100{,}000$
  - **Table S**: $N = 500$, $n = 40{,}000$

- Cost Analysis:
  - $M + (m \cdot N) = 1000 + (100000 \cdot 500) =$ **50,001,000 IOs**
  - At 0.1 ms/IO, Total time ≈ 1.3 hours

Naïve Nested Loop Join

- Example database:
  - **Table R**: $M = 1000$, $m = 100{,}000$
  - **Table S**: $N = 500$, $n = 40{,}000$

- Cost Analysis:
  - $M + (m \cdot N) = 1000 + (100000 \cdot 500) =$ **50,001,000 IOs**
  - At 0.1 ms/IO, Total time ≈ 1.3 hours

- What if smaller table (S) is used as the outer table?
  - $N + (n \cdot M) = 500 + (40000 \cdot 1000) =$ **40,000,500 IOs**
  - At 0.1 ms/IO, Total time ≈ 1.1 hours

Naïve Nested Loop Join

- Example database:
  - **Table** R: $M = 1000$, $m = 100{,}000$
  - **Table** S: $N = 500$, $n = 40{,}000$

$\left.\right\}$ ***4 KB pages → 6 MB***

- Cost Analysis:
  - $M + (m \cdot N) = 1000 + (100000 \cdot 500) =$ **50,001,000 IOs**
  - At 0.1 ms/IO, Total time ≈ 1.3 hours

- What if smaller table (S) is used as the outer table?
  - $N + (n \cdot M) = 500 + (40000 \cdot 1000) =$ **40,000,500 IOs**
  - At 0.1 ms/IO, Total time ≈ 1.1 hours

Block Nested Loop Join

```
foreach block B_R ∈ R:
  foreach block B_S ∈ S:
    foreach tuple r ∈ B_R:
      foreach tuple s ∈ B_S:
        if r and s match then emit
```

R(id,name)

M pages
m tuples

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

N pages
n tuples

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

Block Nested Loop Join

- This algorithm performs fewer disk accesses.
  - For every block in **R**, it scans **S** once.
- **Cost: $M + (M \cdot N)$**

M pages
m tuples

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

SCHOOL OF DATA SCIENCE

Block Nested Loop Join

- The smaller table should be the outer table.
- We determine size based on the number of pages, <u>not</u> the number of tuples.

M pages
m tuples

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Block Nested Loop Join

- Example database:
  - **Table R**: $M = 1000$, $m = 100{,}000$
  - **Table S**: $N = 500$, $n = 40{,}000$
- Cost Analysis:
  - $M + (M \cdot N) = 1000 + (1000 \cdot 500) =$ **501,000 IOs**
  - At 0.1 ms/IO, Total time ≈ 50 seconds

SCHOOL OF DATA SCIENCE

Block Nested Loop Join

- If we have ***B*** buffers available:
  - Use ***B***-2 buffers for each block of the outer table.
  - Use one buffer for the inner table, one buffer for output.

M pages
m tuples

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Block Nested Loop Join

```
foreach B - 2 pages pR ∈ R:
  foreach page pS ∈ S:
    foreach tuple r ∈ B - 2 pages:
      foreach tuple s ∈ pS:
        if r and s match then emit
```

R(id,name)

M pages
m tuples

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Block Nested Loop Join

- This algorithm uses B-2 buffers for scanning **R**.
- **Cost:** $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$
- If the outer relation fits in memory ($M < B\text{-}2$):
  - **Cost:** $M + N$ = 1000 + 500 = **1500 I/Os**
  - At 0.1ms per I/O, Total time ≈ 0.15 seconds
- If we have B=102 buffer pages:
  - **Cost:** $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$ = 1000 + 10 · 500 = **6000 I/Os**
  - Or can switch inner/outer relations, giving us cost: 500 + 5 · 1000 = **5500 I/Os**
    ≈ 0.55 seconds

Block Nested Loop Join

- **Cost: $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$**
- If the outer relation fits in memory ($M < B\text{-}2$):
  - **Cost: $M + N$** = 1000 + 500 = **1500 I/Os**
  - At 0.1ms per I/O, Total time ≈ 0.15 seconds
- If we have B=102 buffer pages:
  - **Cost: $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$** = 1000 + 10 · 500 = **6000 I/Os**
  - Or can switch inner/outer relations, giving us cost: 500 + 5 · 1000 = **5500 I/Os**
    ≈ 0.55 seconds

Block Nested Loop Join

- If the outer relation fits in memory ($M < B\text{-}2$):
  - **Cost:** $M + N$ = 1000 + 500 = **1500 I/Os**
  - At 0.1ms per I/O, Total time ≈ 0.15 seconds
- If we have B=102 buffer pages:
  - **Cost:** $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$ = 1000 + 10 · 500 = **6000 I/Os**
  - Or can switch inner/outer relations, giving us cost: 500 + 5 · 1000 = **5500 I/Os**
    ≈ 0.55 seconds

Block Nested Loop Join

- If we have B=102 buffer pages:
  - **Cost: $M + (\lceil M / (B\text{-}2) \rceil \cdot N)$** = 1000 + 10 · 500 = **6000 I/Os**
  - Or can switch inner/outer relations, giving us cost: 500 + 5 · 1000 = **5500 I/Os**
    ≈ 0.55 seconds

Block Nested Loop Join

Nested Loop Join

- Why is the basic nested loop join so bad?
  - For each tuple in the outer table, we must do a sequential scan to check for a match in the inner table.

Nested Loop Join

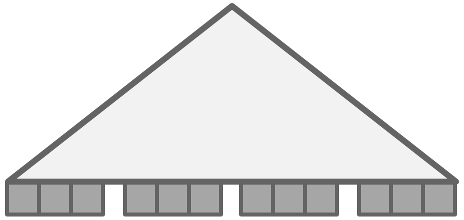
- Why is the basic nested loop join so bad?
  - For each tuple in the outer table, we must do a sequential scan to check for a match in the inner table.

- We can avoid sequential scans by using an index to find inner table matches.
  - Use an existing index for the join.

Index Nested Loop Join

```
foreach tuple r ∈ R:
  foreach tuple s ∈ Index(r_i = s_j):
    if r and s match then emit
```

Index(S.id)



M pages
m tuples

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Index Nested Loop Join

- Assume the cost of each index probe is some constant C per tuple.
- **Cost:** $M + (m \cdot C)$

M pages
m tuples

R(id,name)

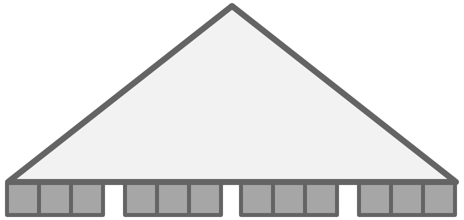
id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025

N pages
n tuples

Index(S.id)



Nested Loop Join Summary

- **Key Takeaways**
  - Pick the smaller table as the outer table.
  - Buffer as much of the outer table in memory as possible.
  - Loop over the inner table (or use an index).
- **Algorithms**
  - Naïve
  - Block
  - Index

Sort-Merge Join

Sort-Merge Join

- **Phase #1: Sort**
  - Sort both tables on the join key(s).
  - You can use any appropriate sort algorithm
  - These phases are distinct from the sort/merge phases of an external merge sort, from the previous class

- **Phase #2: Merge**
  - Step through the two sorted tables with cursors and emit matching tuples.
  - May need to backtrack depending on the join type.

Sort-Merge Join

sort R, S on join keys
$cursor_R \leftarrow R_{sorted}$, $cursor_S \leftarrow S_{sorted}$
while $cursor_R$ and $cursor_S$:
  if $cursor_R > cursor_S$:
    increment $cursor_S$
  if $cursor_R < cursor_S$:
    increment $cursor_R$
    backtrack $cursor_s$ *(if necessary)*
  elif $cursor_R$ and $cursor_S$ match:
    emit
    increment $cursor_S$

Sort-Merge Join

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
200	GZA
400	Raekwon

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Sort-Merge Join

R(id,name)

id	name
600	MethodMan
200	GZA
100	Andy
300	ODB
500	RZA
700	Ghostface
200	GZA
400	Raekwon

Sort!

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
500	7777	10/26/2025
400	6666	10/26/2025
1	9999	10/26/2025
200	8888	10/26/2025

Sort!

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

SCHOOL OF DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

Sort!

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
4	6666	10/26/2025
500	7777	10/26/2025

Sort!

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

SCHOOL OF DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

SCHOOL OF
DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```sql
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/2025

SCHOOL OF DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/2025

SCHOOL OF
DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/2025

SCHOOL OF
DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/202 5
100	9999	10/26/202 5
200	8888	10/26/202 5
400	6666	10/26/202 5
500	7777	10/26/202 5

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202 5

SCHOOL OF DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/202 5
100	9999	10/26/202 5
200	8888	10/26/202 5
400	6666	10/26/202 5
500	7777	10/26/202 5

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202 5

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	400	6666	10/26/202
500	RZA	500	7777	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/202
500	RZA	500	7777	10/26/2025

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/202
500	RZA	500	7777	10/26/2025

SCHOOL OF DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/202
500	RZA	500	7777	10/26/2025

SCHOOL OF
DATA SCIENCE

Sort-Merge Join

R(id,name)

id	name
100	Andy
200	GZA
200	GZA
300	ODB
400	Raekwon
500	RZA
600	MethodMan
700	Ghostface

S(id,value,cdate)

id	value	cdate
100	2222	10/26/2025
100	9999	10/26/2025
200	8888	10/26/2025
400	6666	10/26/2025
500	7777	10/26/2025

```
SELECT R.id, S.cdate
  FROM R JOIN S
    ON R.id = S.id
 WHERE S.value > 100
```

Output Buffer

R.id	R.name	S.id	S.value	S.cdate
100	Andy	100	2222	10/26/202
100	Andy	100	9999	10/26/202
200	GZA	200	8888	10/26/202
200	GZA	200	8888	10/26/202
400	Raekwon	200	6666	10/26/202
500	RZA	500	7777	10/26/2025

Sort-Merge Join

- **Sort Cost (R):** $2M \cdot (1 + \lceil \log_{B-1} \lceil M / B \rceil \rceil)$
- **Sort Cost (S):** $2N \cdot (1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil)$
- **Merge Cost:** $(M + N)$

- **Total Cost: Sort + Merge**

Sort-Merge Join

- Example database:
  - **Table R**: $M = 1000$, $m = 100{,}000$
  - **Table S**: $N = 500$, $n = 40{,}000$

- With B=100 buffer pages, both **R** and **S** can be sorted in two passes:
  - Sort Cost (**R**) = $2000 \cdot (1 + \lceil \log_{99} 1000 / 100 \rceil)$ = **4000 I/Os**
  - Sort Cost (**S**) = $1000 \cdot (1 + \lceil \log_{99} 500 / 100 \rceil)$ = **2000 I/Os**
  - Merge Cost = (1000 + 500) = **1500 I/Os**
  - Total Cost = 4000 + 2000 + 1500 = **7500 I/Os**
  - At 0.1 ms/IO, Total time ≈ 0.75 seconds

Bonus

SCHOOL OF DATA SCIENCE

Sort-Merge Join: Advanced Ver.

- Merge the run-generation phase and the join phase.
  - Produce initial runs for R. These are ≈B pages long.
  - Produce initial runs for S. These are also ≈B pages long.
  - If there are enough pages to hold one page from each run in memory, do the join and merge passes together. Need $\lceil R / B \rceil + \lceil S / B \rceil$ pages
  - Thus, in 2 passes (1 initial run creation, then sort-merge-join pass) you can compute the join.
  - Some more modifications are possible, including making the initial runs be of size 2B , which guarantees that a 2-pass sort-merge join can be done if $B > \sqrt{MAX(R, S)}$

Bonus

Sort-Merge Join: Advanced Ver.

Relation R

...	...	...	...	...	...	...	...

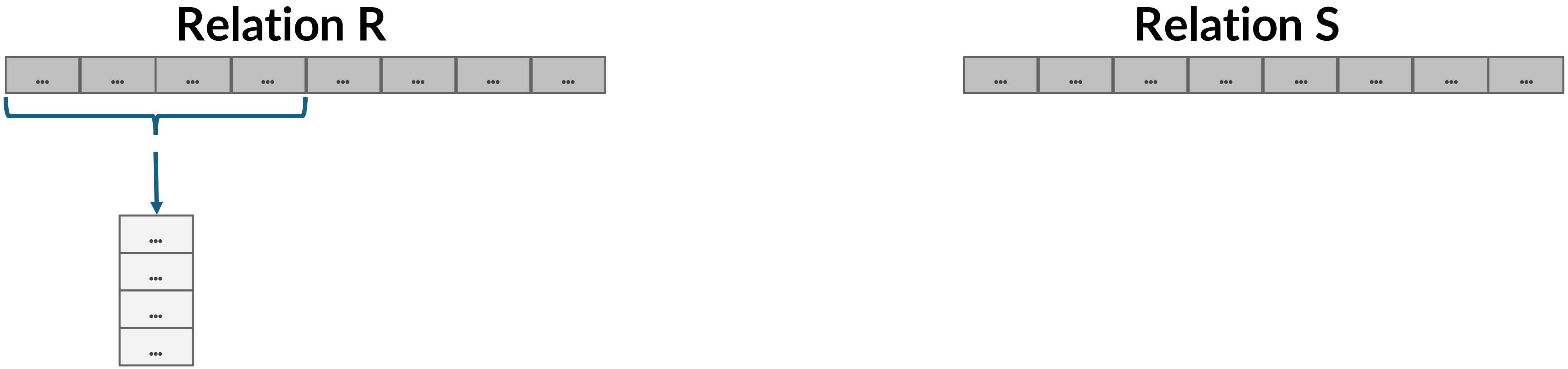
Relation S

...	...	...	...	...	...	...	...

Bonus

SCHOOL OF DATA SCIENCE

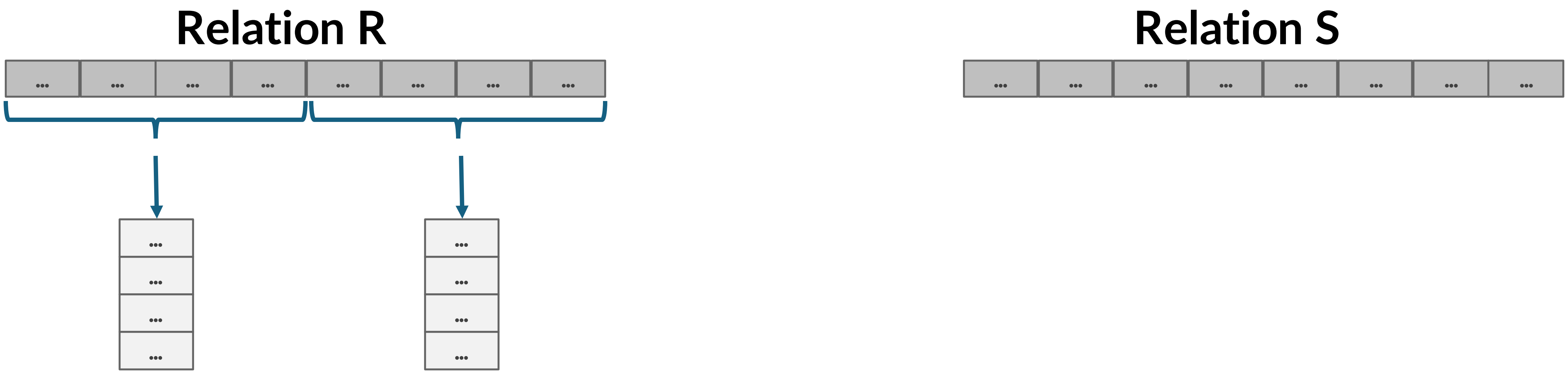
Sort-Merge Join: Advanced Ver.



Bonus

Sort-Merge Join: Advanced Ver.

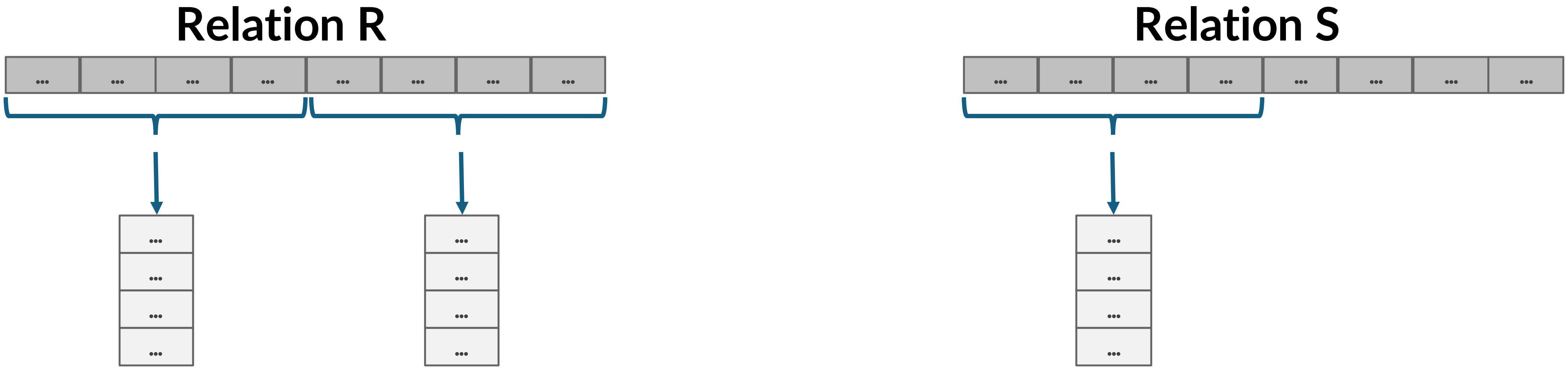
SCHOOL OF DATA SCIENCE



Bonus

Sort-Merge Join: Advanced Ver.

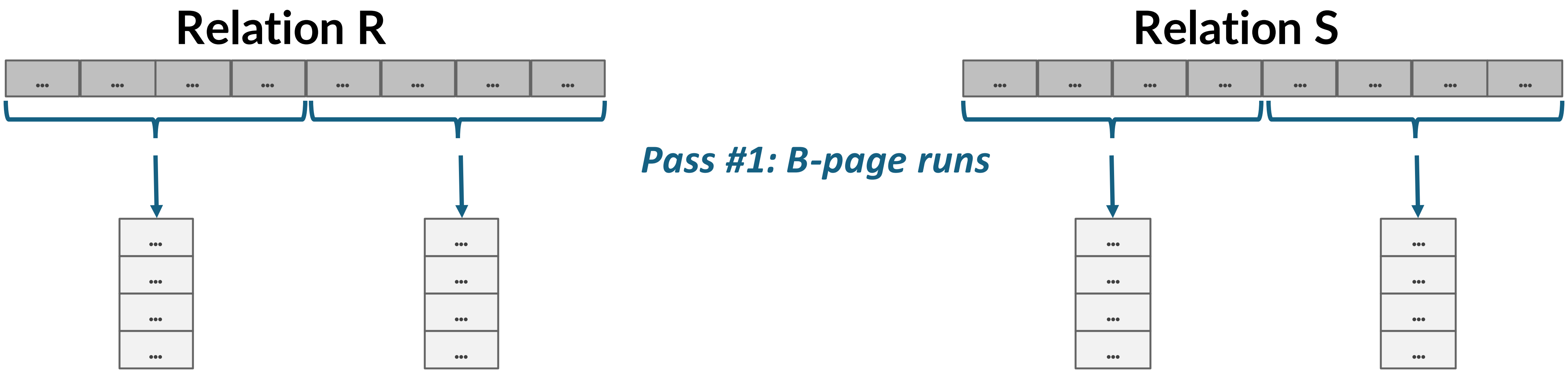
SCHOOL OF DATA SCIENCE



Bonus

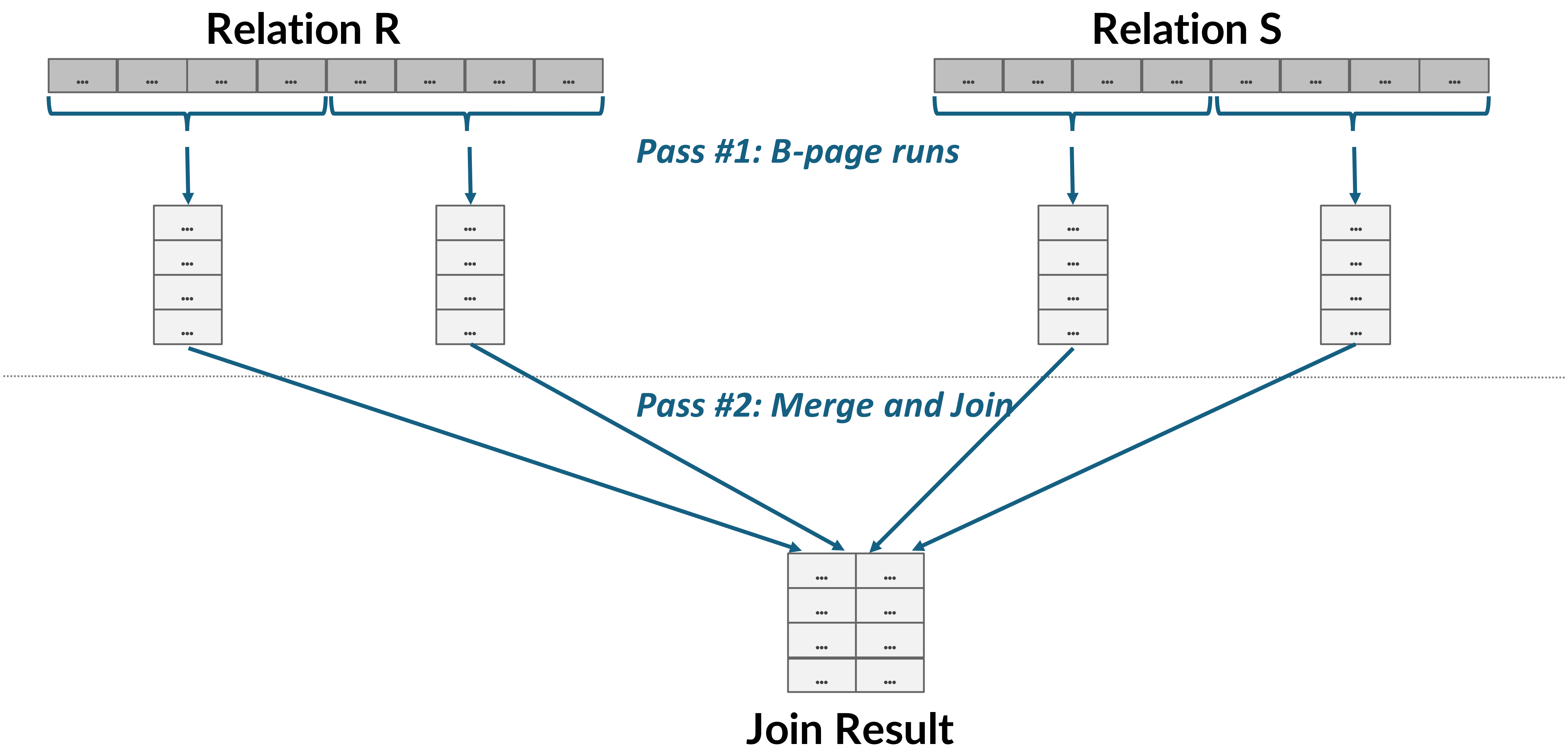
Sort-Merge Join: Advanced Ver.

SCHOOL OF DATA SCIENCE



Bonus

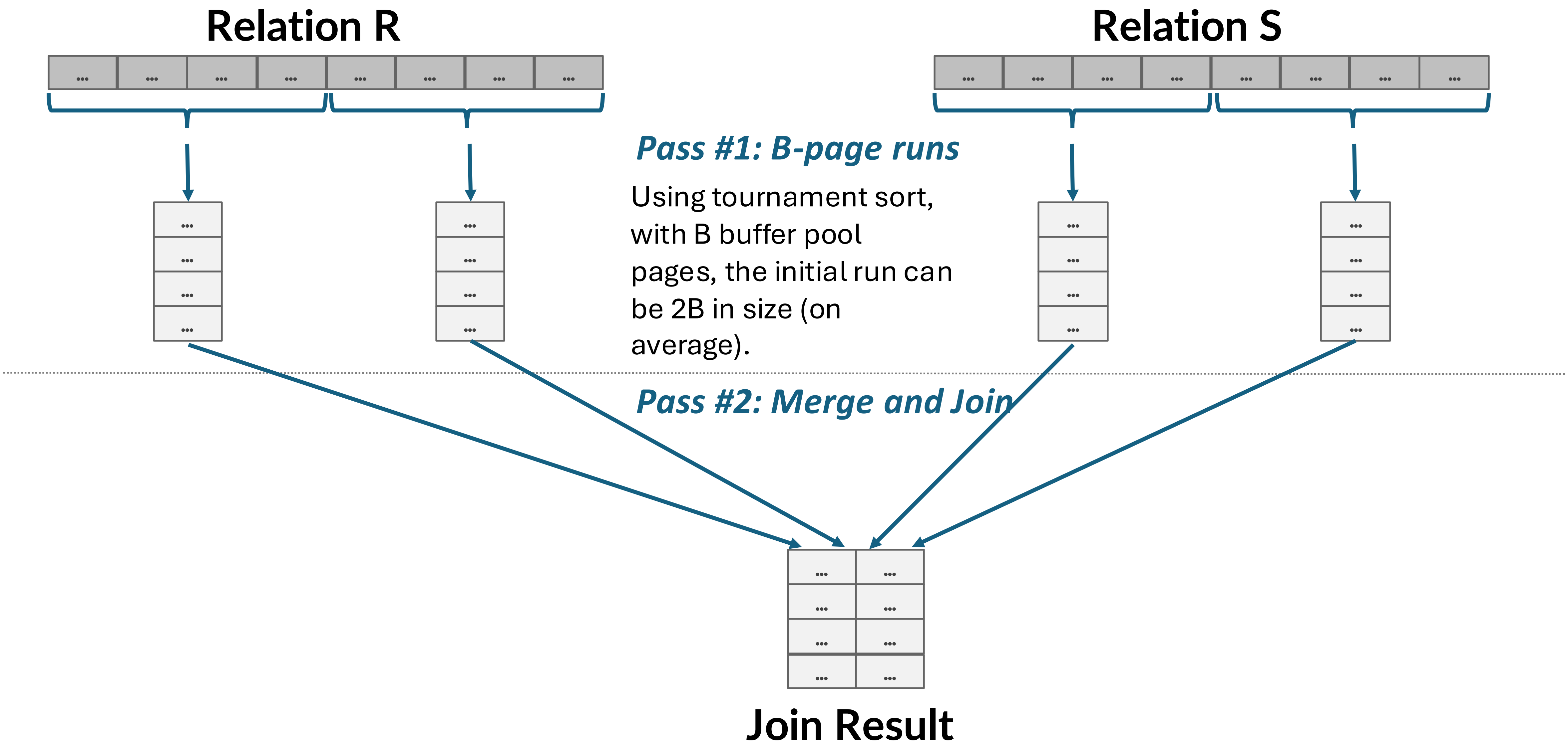
Sort-Merge Join: Advanced Ver.



Bonus

Sort-Merge Join: Advanced Ver.

SCHOOL OF DATA SCIENCE



Using tournament sort, with B buffer pool pages, the initial run can be 2B in size (on average).

Sort-Merge Join

- The worst case for the merging phase is when the join attribute of all the tuples in both relations contains the same value.
- **Cost: M + (M · N) + (sort cost)**

When is Sort-Merge Join Useful?

- One or both tables are already sorted on join key.
- Output must be sorted on join key.

- The input relations may be sorted either by an explicit sort operator, or by scanning the relation using an index on the join key.

Hash Joins

Hash Join

- If tuple $r \in R$ and a tuple $s \in S$ satisfy the join condition, then they have the same value for the join attributes.

- If that value is hashed to some partition i, the R tuple must be in r_i and the S tuple in s_i.

- Therefore, R tuples in r_i need only to be compared with S tuples in s_i.

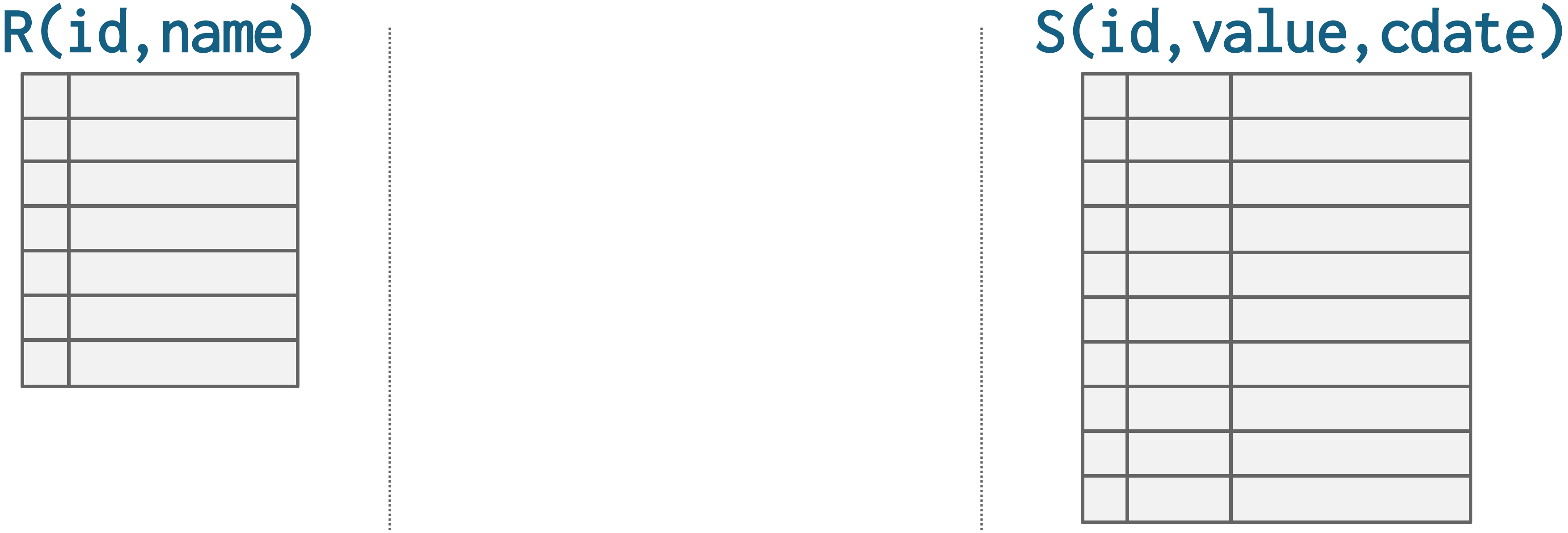
Simple Hash Join Algorithm

- **Phase #1: Build**
  - Scan the outer relation and populate a hash table using the hash function h_1 on the join attributes.
  - We can use any hash table that we discussed before but in practice linear probing works the best.

- **Phase #2: Probe**
  - Scan the inner relation and use h_1 on each tuple to jump to a location in the hash table and find a matching tuple.

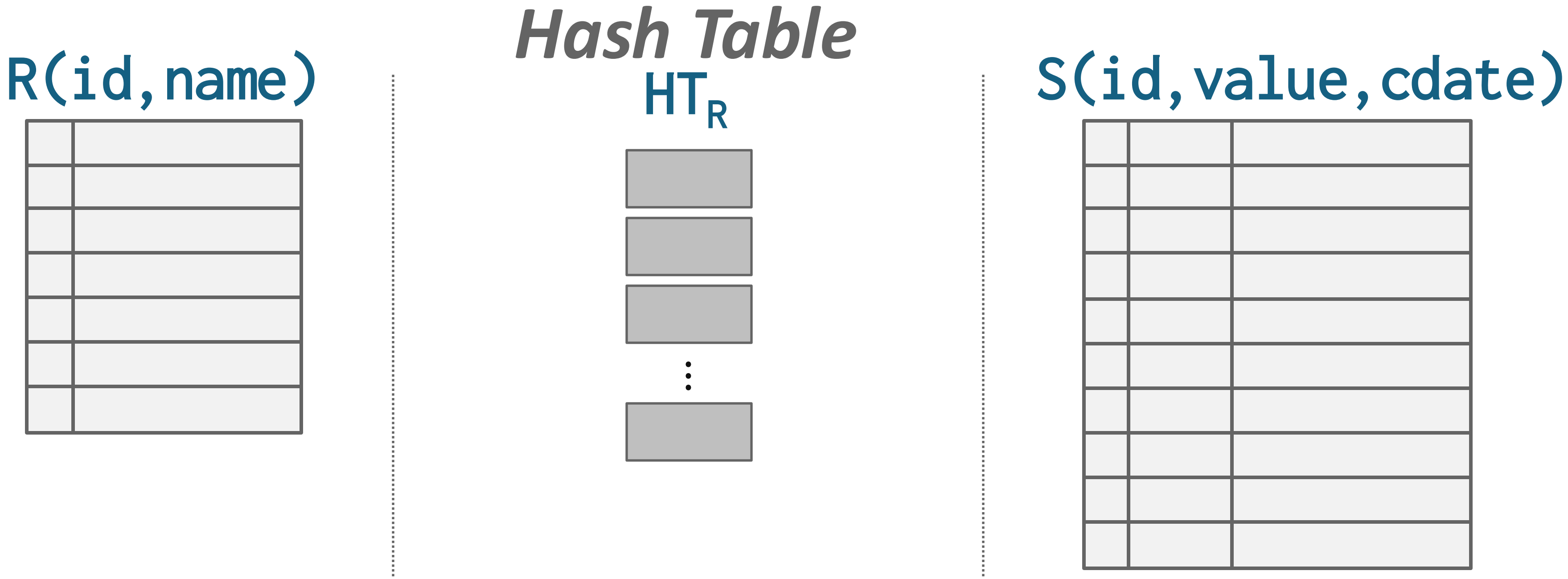
Simple Hash Join Algorithm

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h_1(s) ∈ HT_R
```



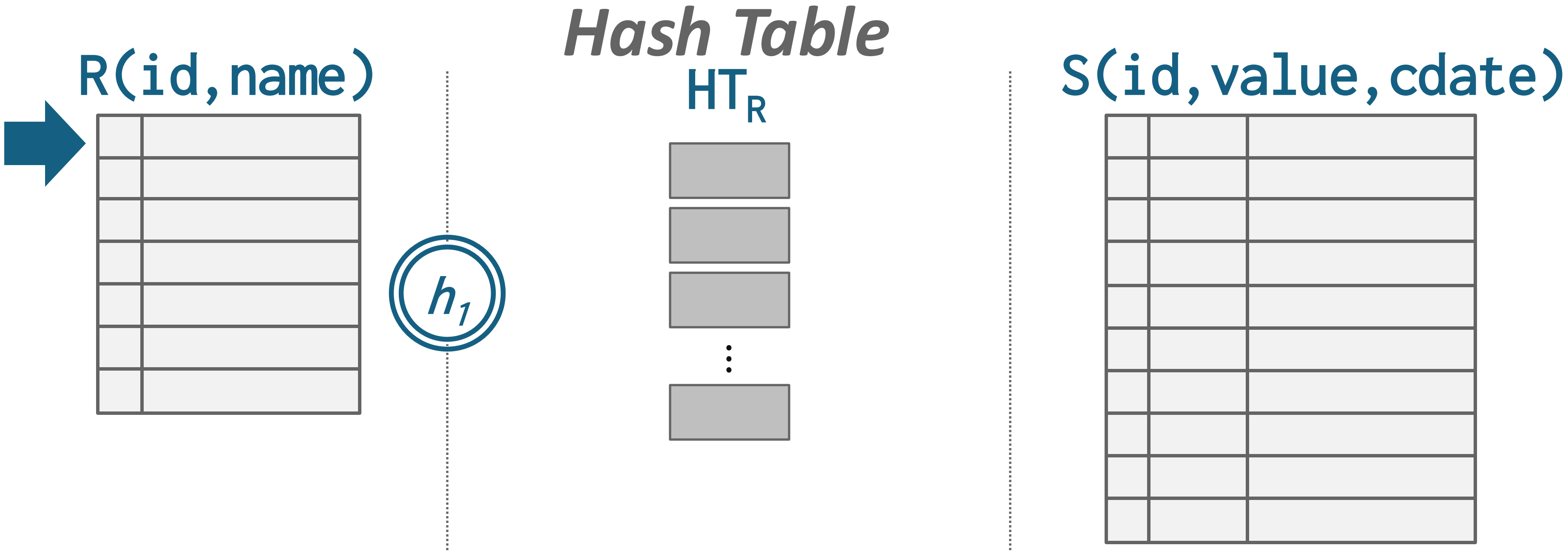
Simple Hash Join Algorithm

build hash table HT_R for R
foreach tuple $s \in S$
  output, if $h_1(s) \in HT_R$



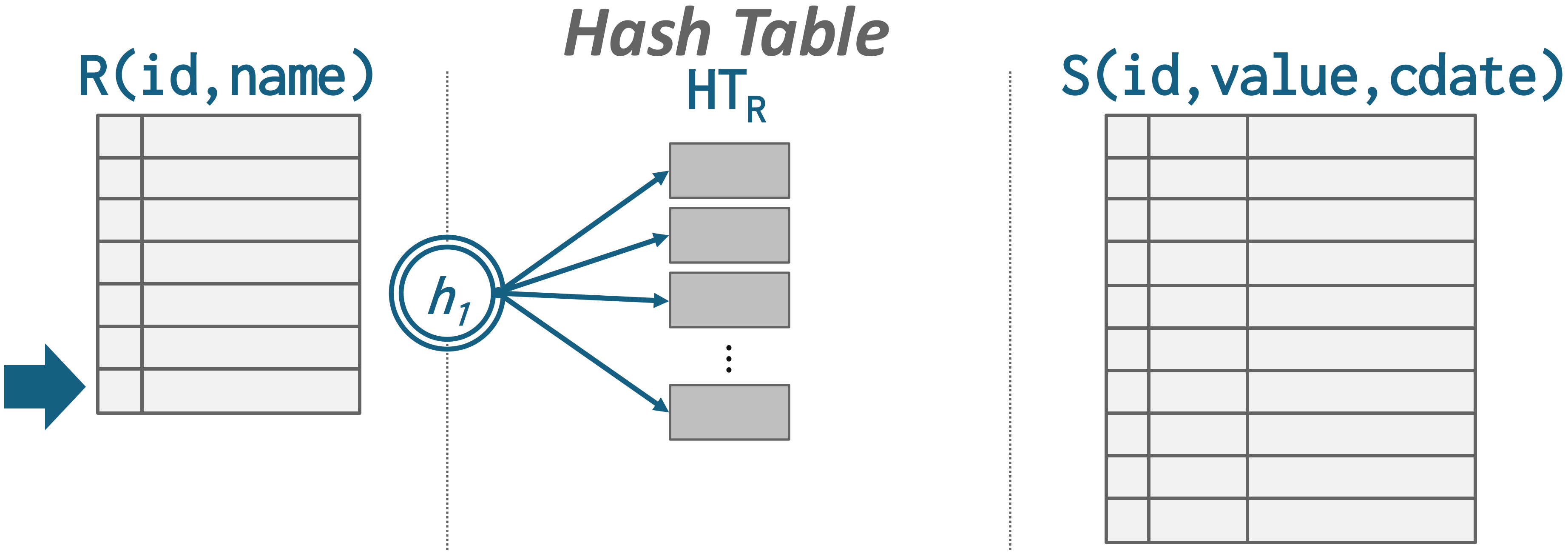
Simple Hash Join Algorithm

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h1(s) ∈ HT_R
```



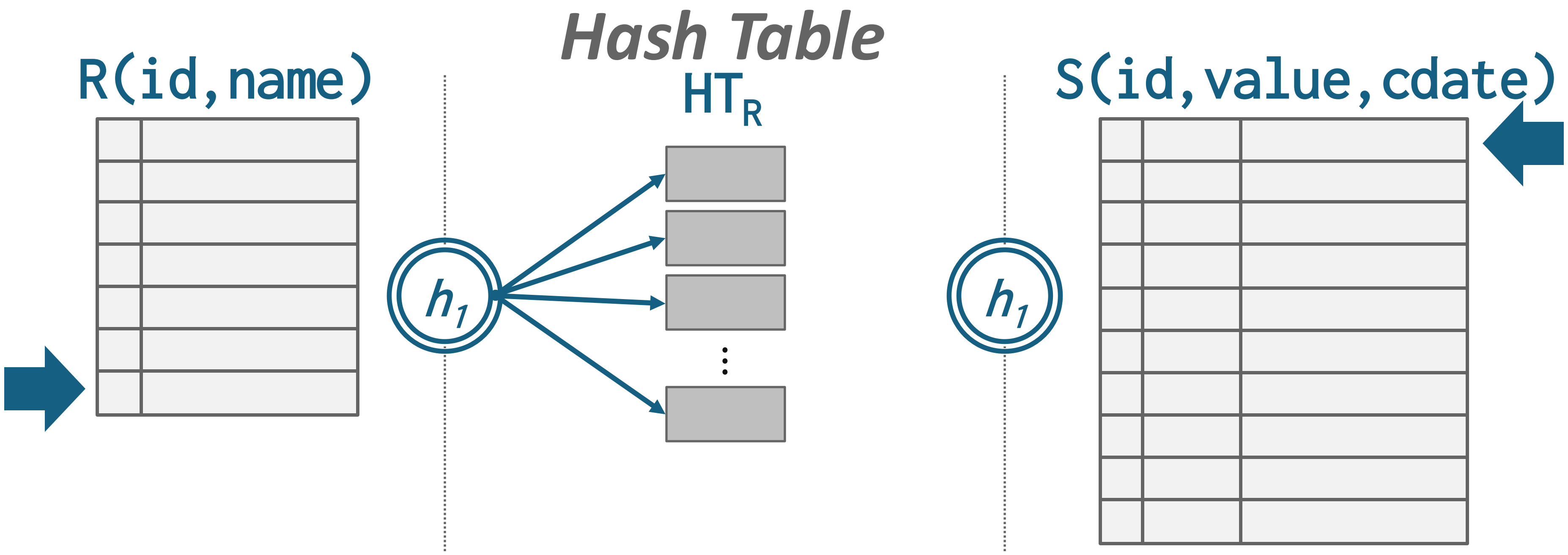
Simple Hash Join Algorithm

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h_1(s) ∈ HT_R
```



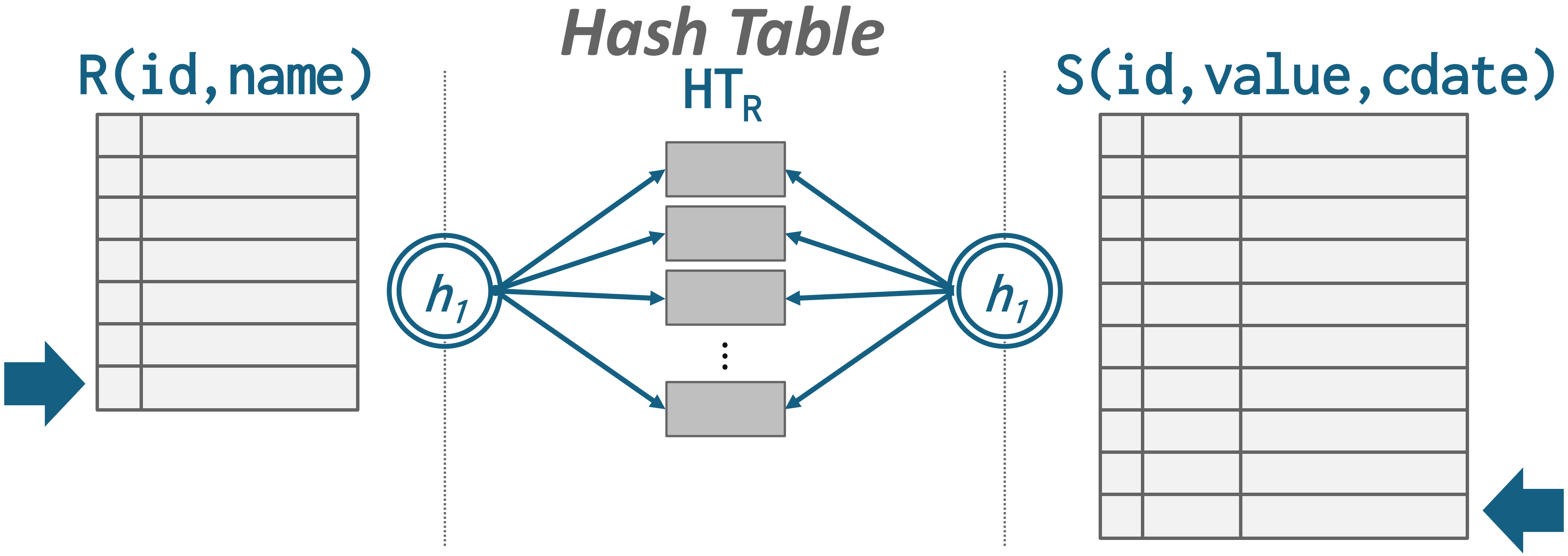
Simple Hash Join Algorithm

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h_1(s) ∈ HT_R
```



Simple Hash Join Algorithm

```
build hash table HT_R for R
foreach tuple s ∈ S
  output, if h1(s) ∈ HT_R
```



Hash Table Contents

- **Key:** The attribute(s) that the query is joining on
  - The hash table needs to store the key to verify that we have a correct match, in case of hash collisions.

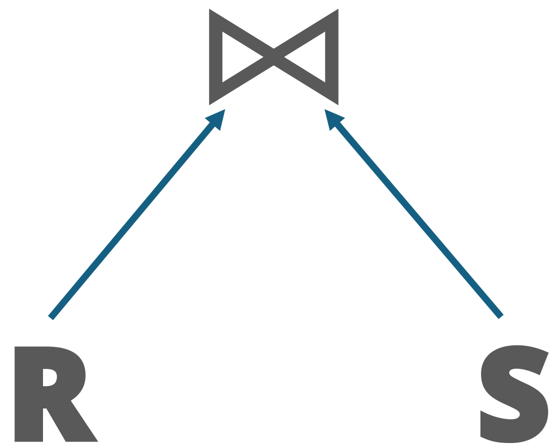
- **Value:** It varies per DBMS
  - Depends on what the next query operators will do with the output from the join
  - Early vs. Late Materialization

Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***

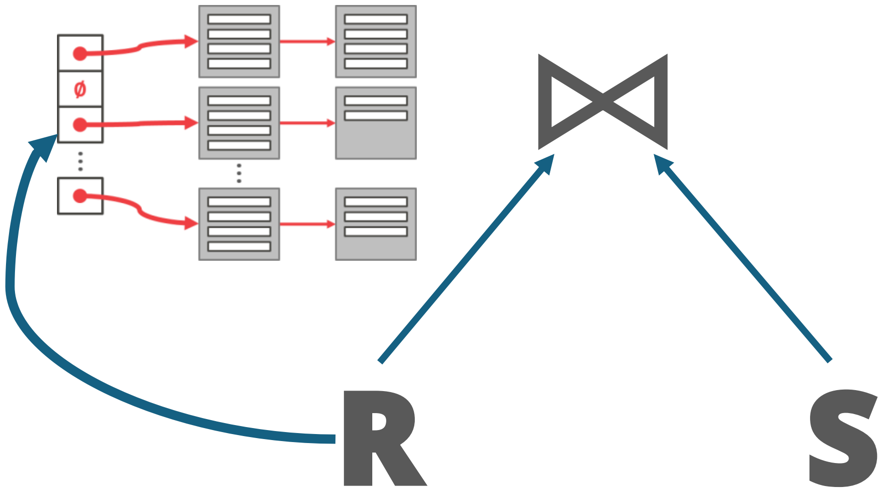
Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



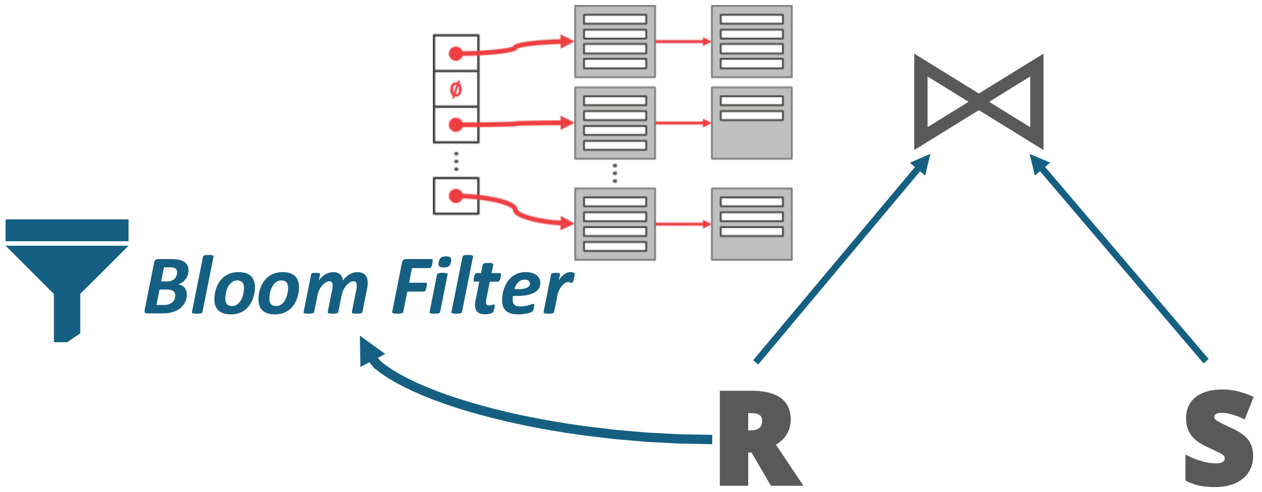
Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



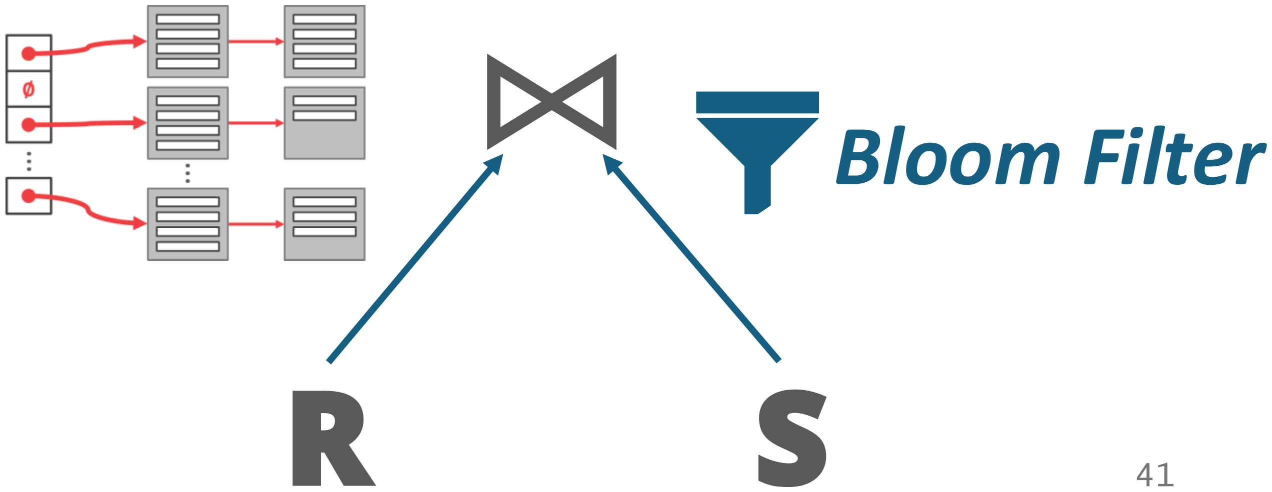
Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



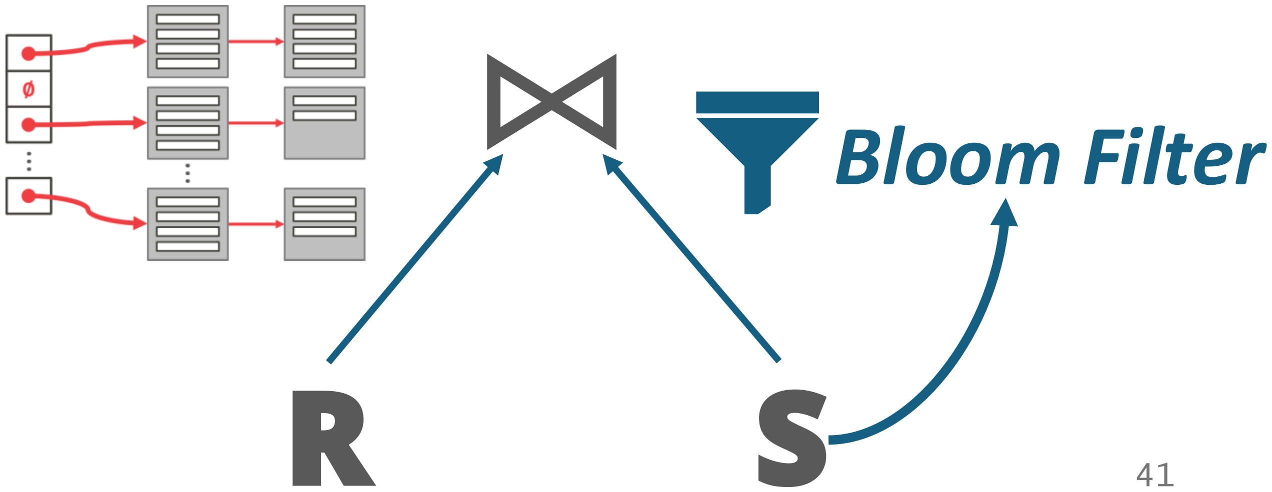
Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



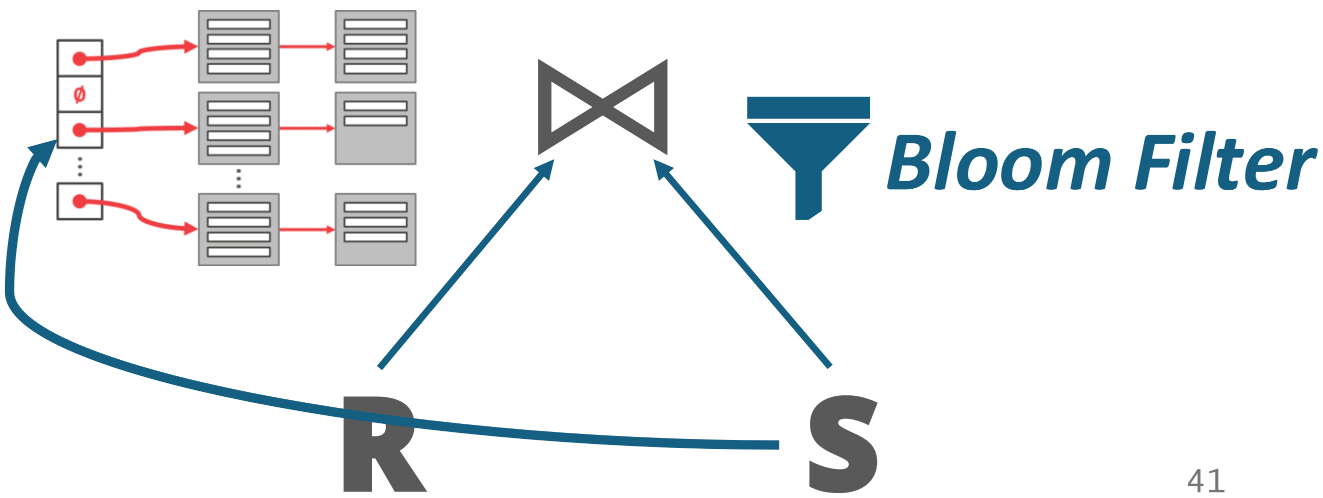
Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



Optimization: Probe Filter

- Create a probe filter (such as a Bloom Filter) during the build phase if the key is likely to not exist in the inner relation
  - Check the filter before probing the hash table
  - Fast because the filter fits in CPU cache
  - Sometimes called ***sideways information passing.***



Hash Joins of Large Relations

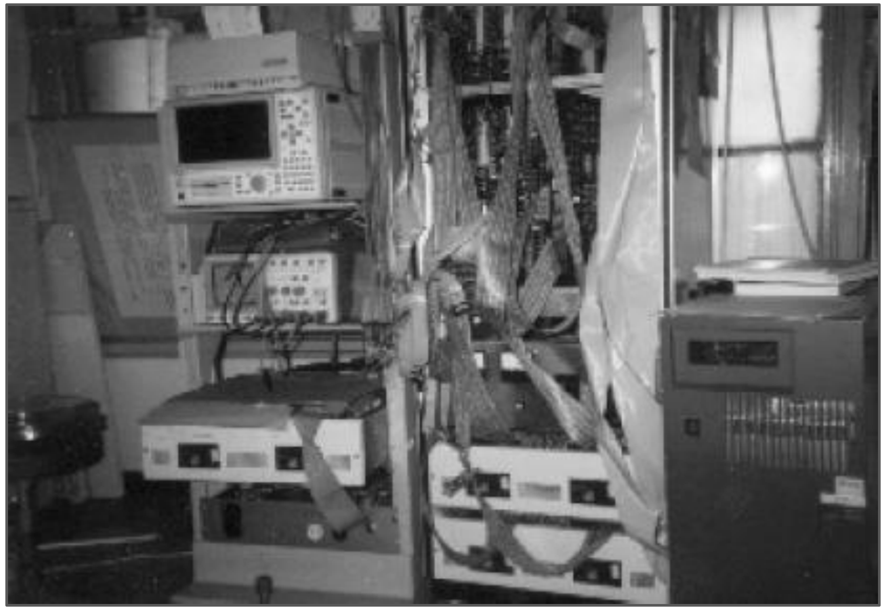
- What happens if we do not have enough memory to fit the entire hash table?

- We do not want to let the buffer pool manager swap out the hash table pages at random.

Partitioned Hash Join

- Hash join when tables do not fit in memory.
    - **Partition Phase:** Hash both tables on the join attribute into partitions.
    - **Probe Phase:** Compares tuples in corresponding partitions for each table.

- Sometimes called **GRACE Hash Join.**
    - Named after the GRACE database machine from Japan in the 1980s.

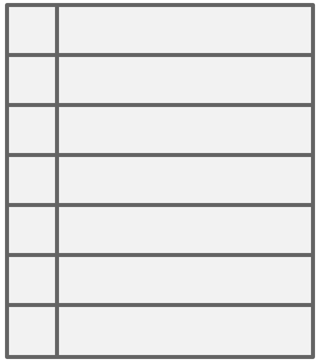


GRACE
University of Tokyo

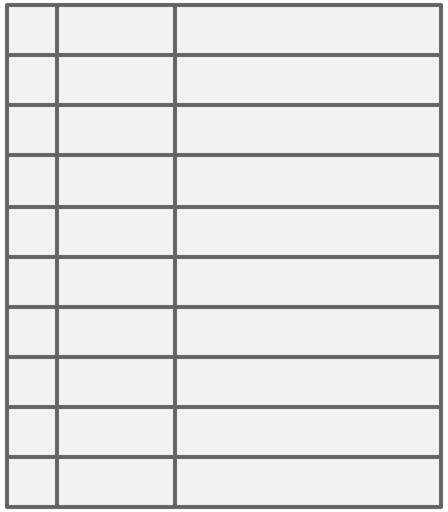
Partitioned Hash Join: Partition Phase

- Hash `R` into k buckets.
- Hash `S` into k buckets with same hash function.
- Write buckets to disk when they get full.

`R(id,name)`



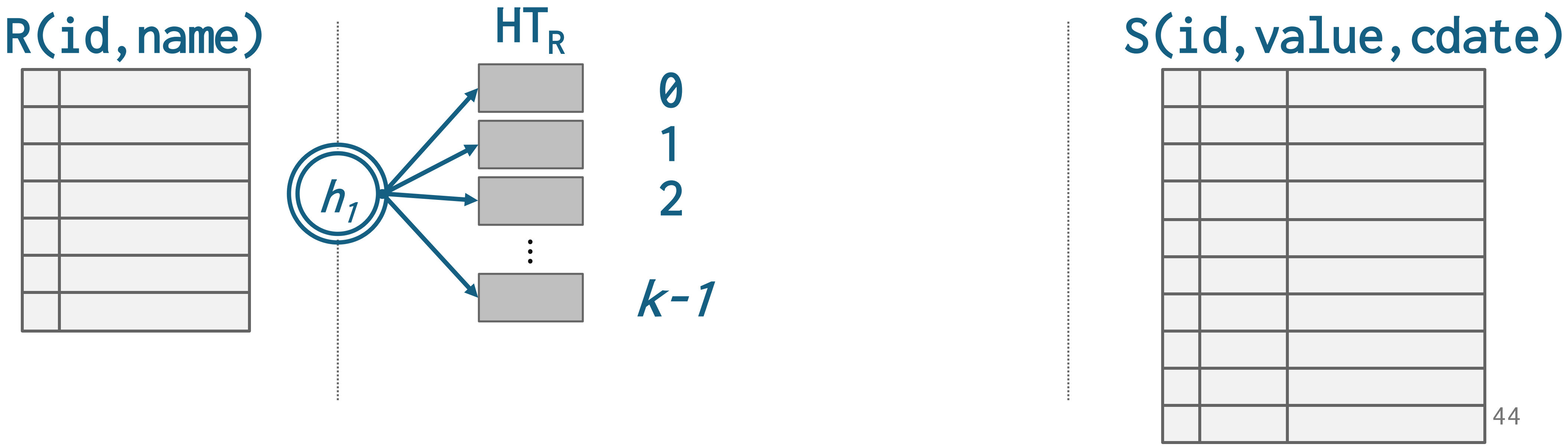
`S(id,value,cdate)`



SCHOOL OF DATA SCIENCE

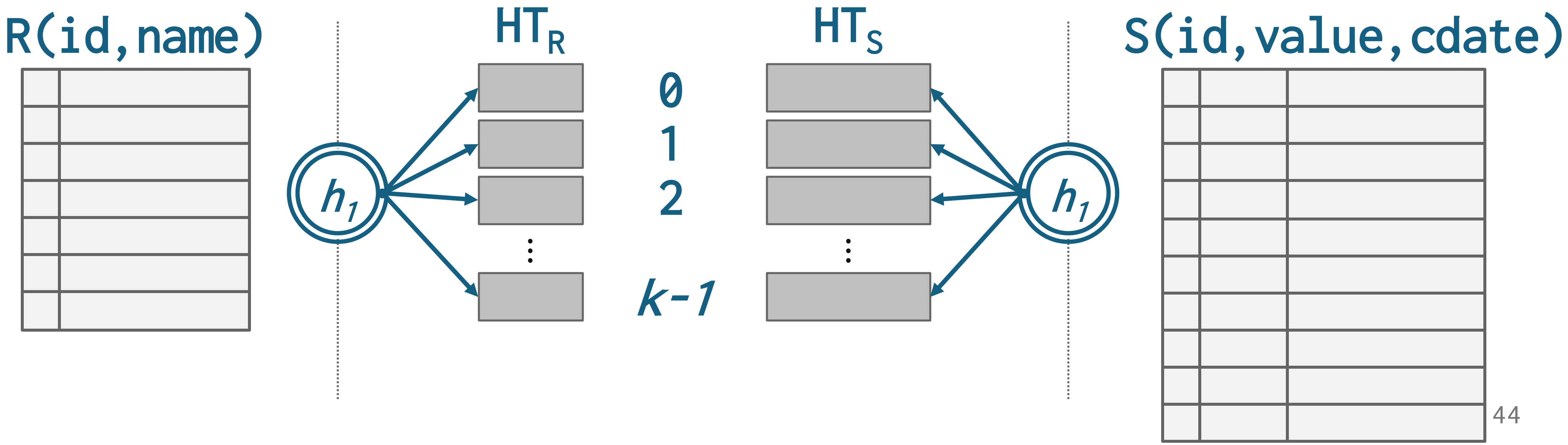
Partitioned Hash Join: Partition Phase

- Hash `R` into *k* buckets.
- Hash `S` into *k* buckets with same hash function.
- Write buckets to disk when they get full.



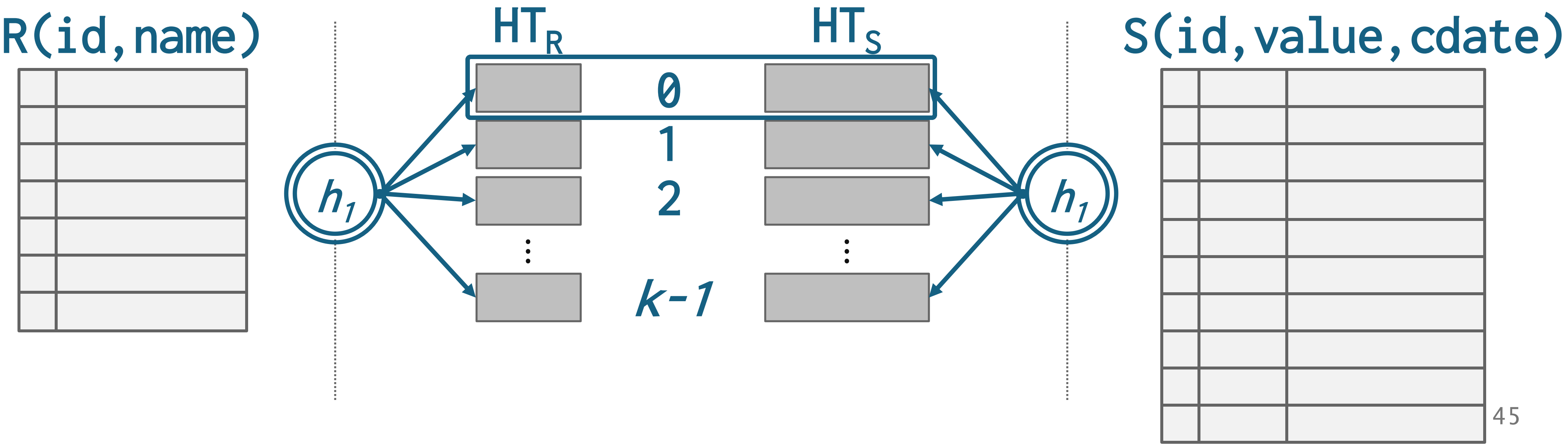
Partitioned Hash Join: Partition Phase

- Hash `R` into k buckets.
- Hash `S` into k buckets with same hash function.
- Write buckets to disk when they get full.



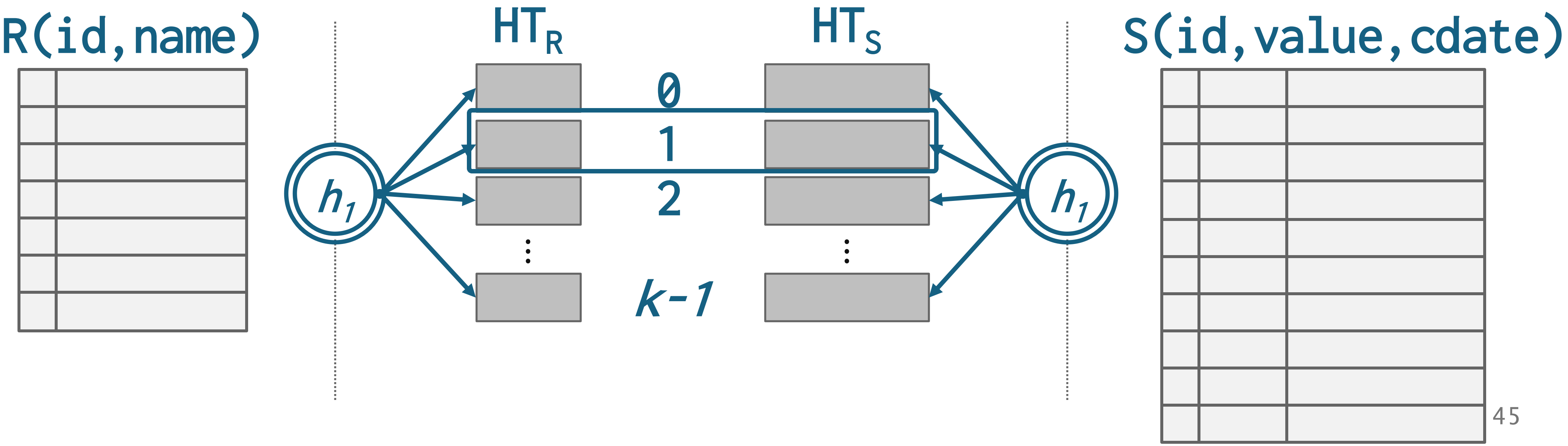
Partitioned Hash Join: Probe Phase

- Read corresponding partitions into memory one pair at a time, hash join their contents.



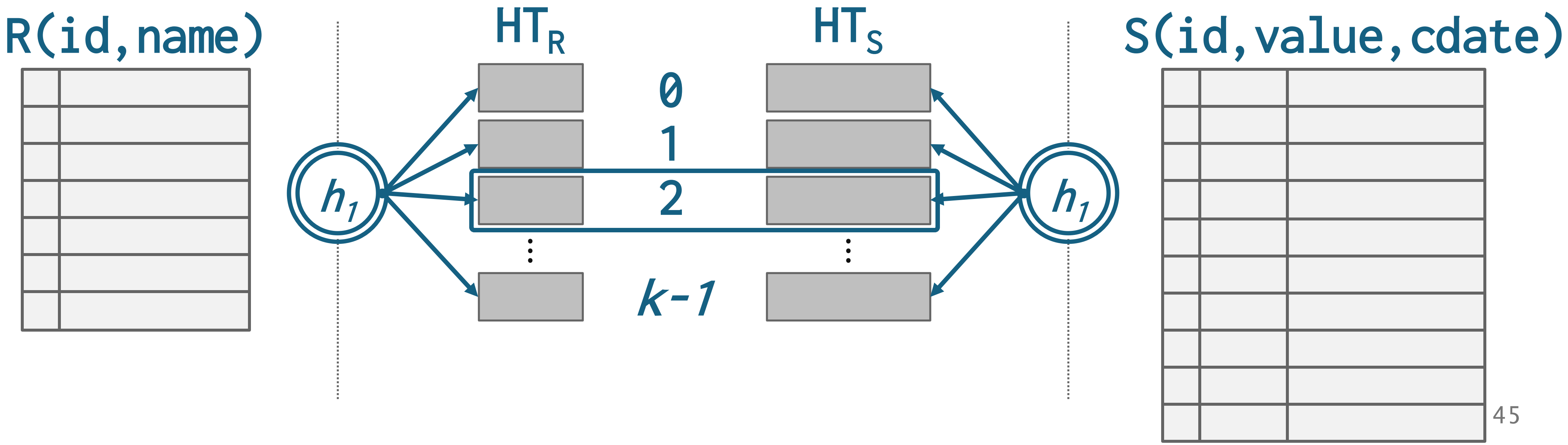
Partitioned Hash Join: Probe Phase

- Read corresponding partitions into memory one pair at a time, hash join their contents.



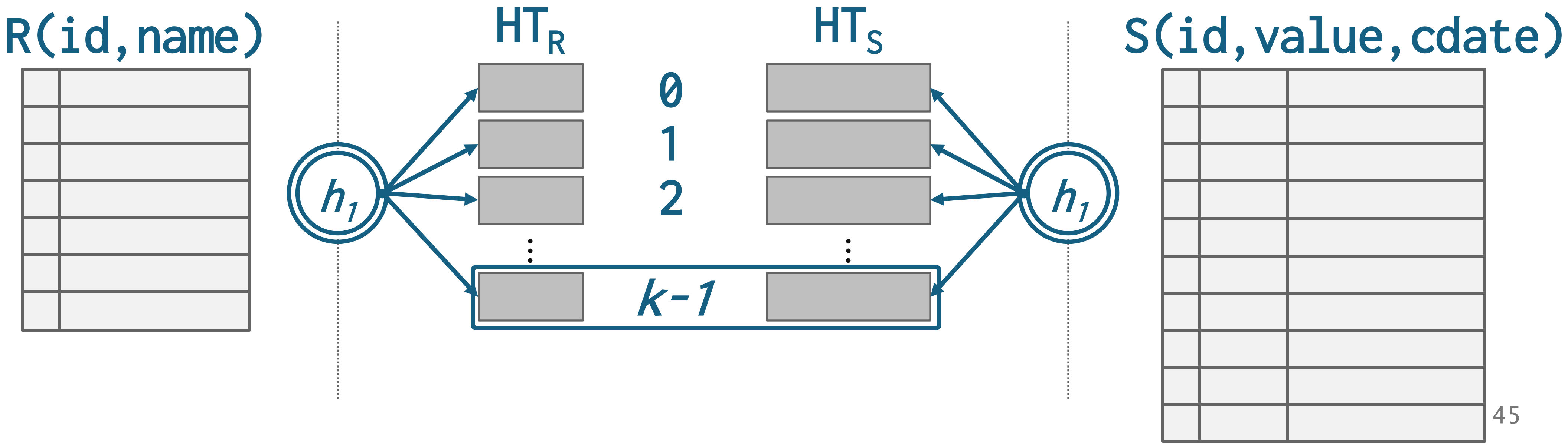
Partitioned Hash Join: Probe Phase

- Read corresponding partitions into memory one pair at a time, hash join their contents.



Partitioned Hash Join: Probe Phase

- Read corresponding partitions into memory one pair at a time, hash join their contents.



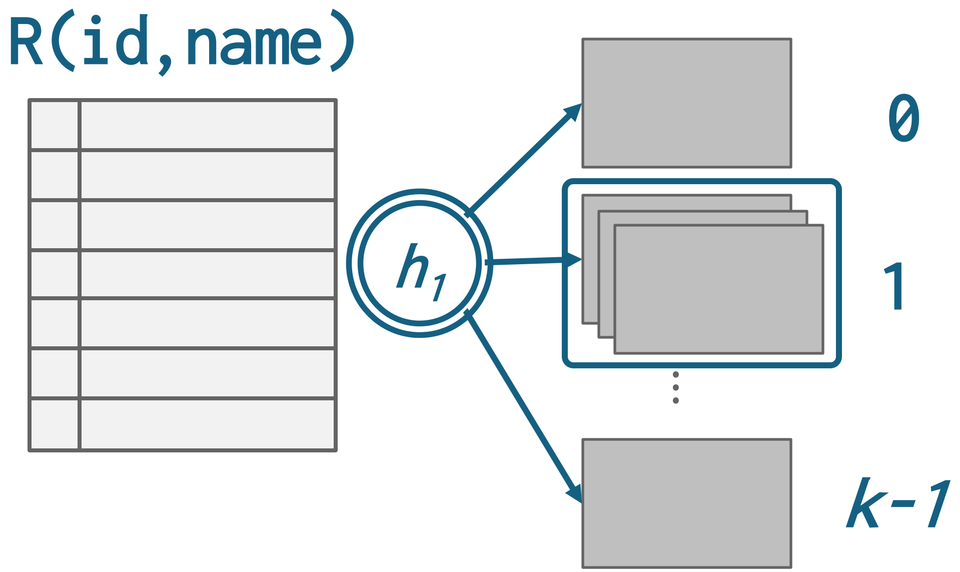
SCHOOL OF DATA SCIENCE

Partitioned Hash Join Edge Cases

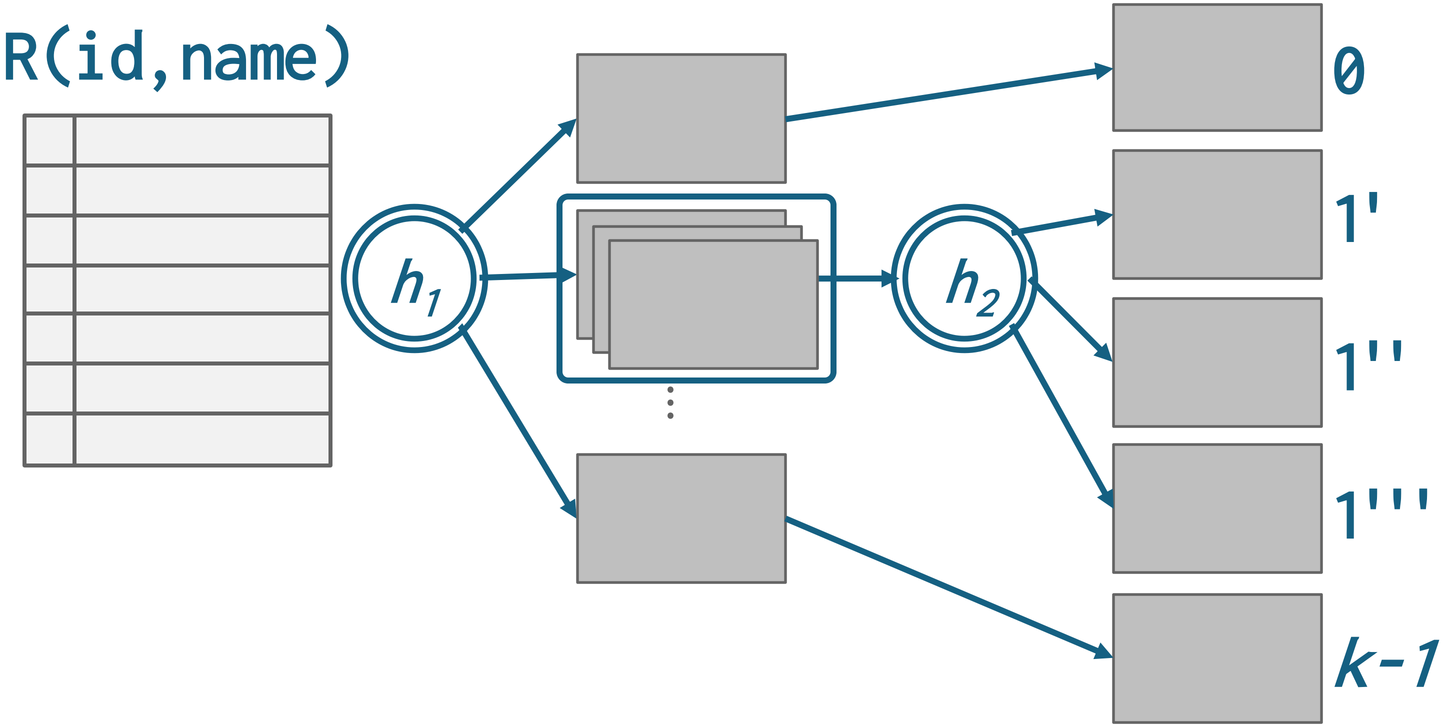
- If a partition does not fit in memory, recursively partition it with a different hash function
  - Repeat as needed
  - Eventually hash join the corresponding (sub-)partitions

- If a single join key has so many matching records that they don't fit in memory, use a block nested loop join for that key

Recursive Partitioning

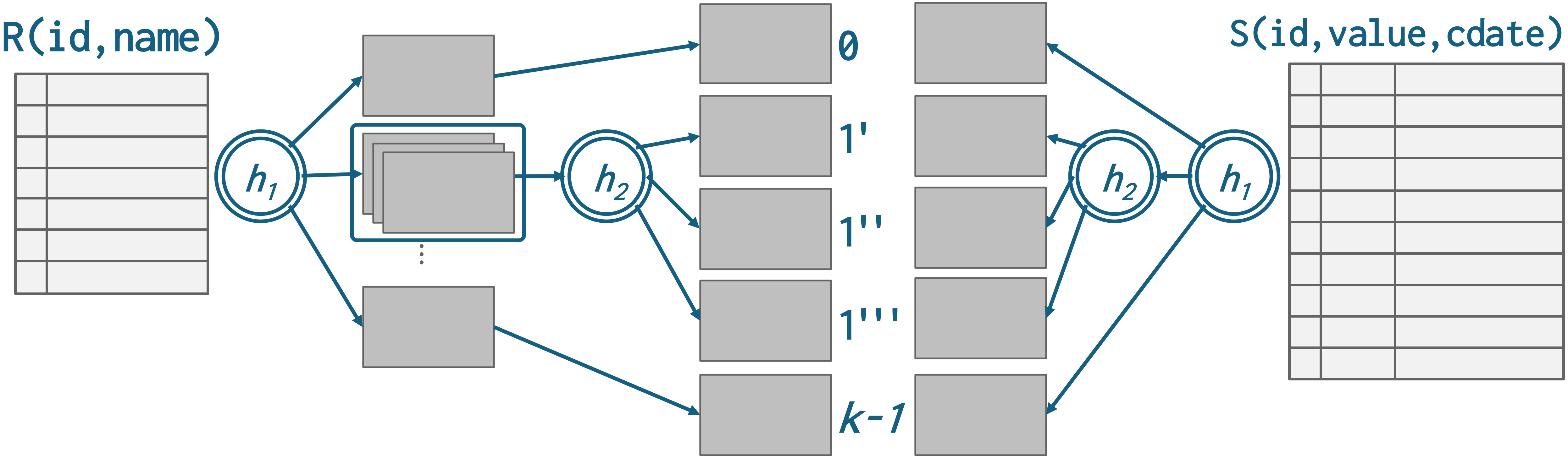


Recursive Partitioning



SCHOOL OF
DATA SCIENCE

Recursive Partitioning



Cost of Partitioned Hash Join

- If we do not need recursive partitioning:
  - Cost: $3(M + N)$

- **Partition phase:**
  - Read+write both tables
  - $2(M+N)$ I/Os
- **Probe phase:**
  - Read both tables (in total, one partition at a time)
  - M+N I/Os

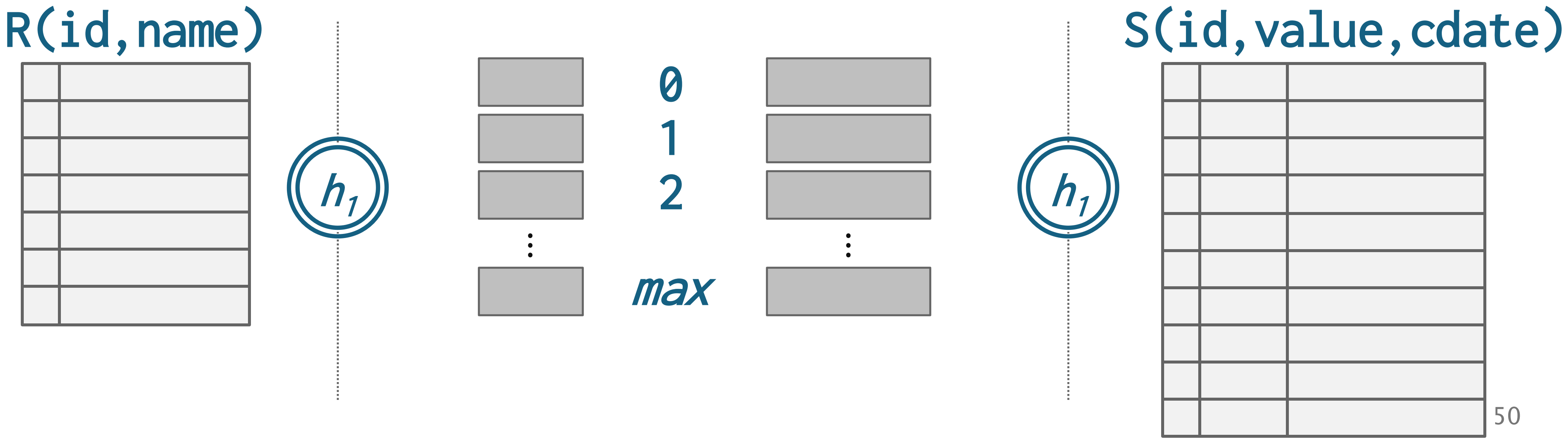
Partitioned Hash Join

- Example database:
  - $M = 1000$, $m = 100{,}000$
  - $N = 500$, $n = 40{,}000$

- Cost Analysis:
  - $3 \cdot (M + N) = 3 \cdot (1000 + 500) =$ **4,500 IOs**
  - At 0.1 ms/IO, Total time ≈ 0.45 seconds

Optimization: Hybrid Hash Join

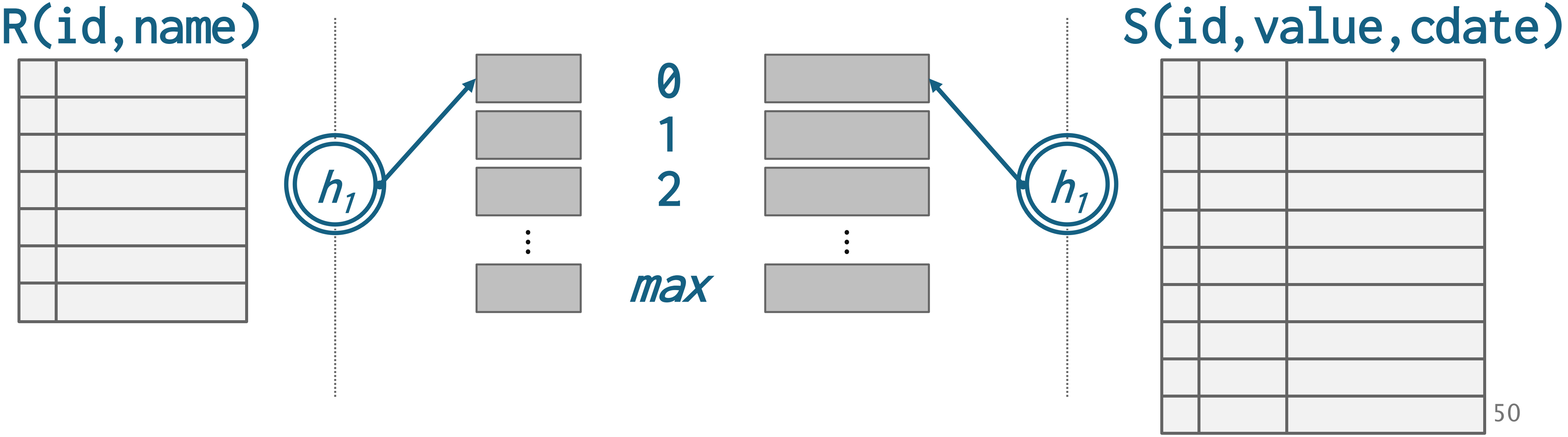
- During the partitioning phase, keep Partition 0 for R (aka. R_0) in memory.
  - e.g. B = 1000 and R = 5000. Only need to partition R 5-ways (so each partition can fit in memory for the second phase). Keep R_0 in memory, in a hash table.
  - In the second phase, for tuples that map to $S_{0,}$ simply probe the R_0 hash table.
  - If the keys are skewed, could adjust the hash function to map the hot keys to R0.
  - Difficult to get to work correctly. Rarely done in practice. Better to "over partition."



SCHOOL OF DATA SCIENCE

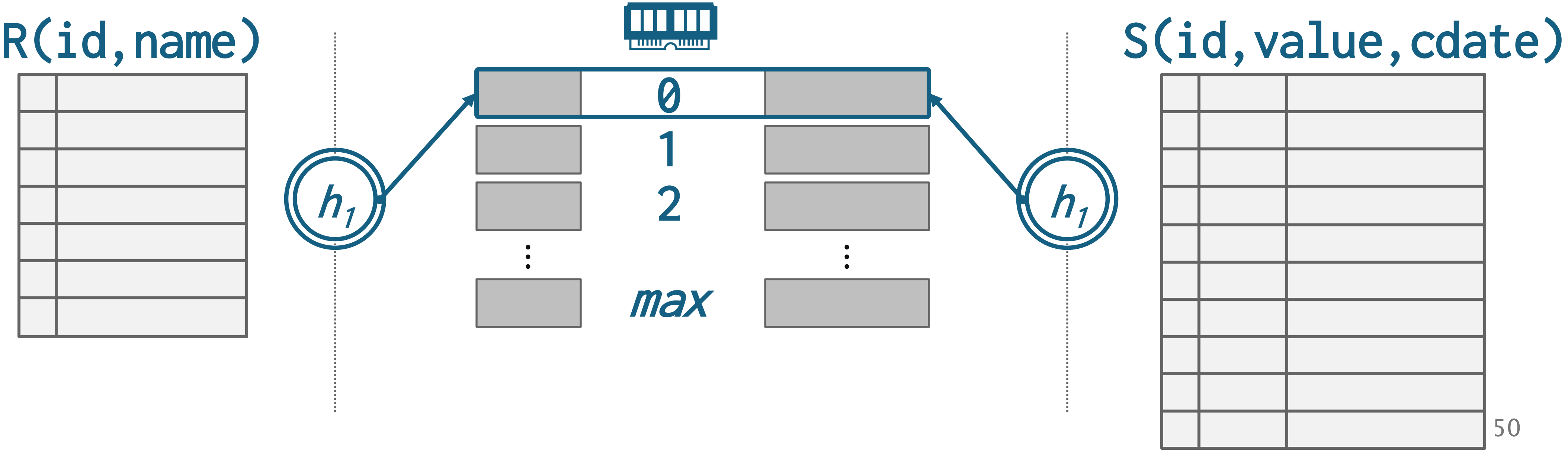
Optimization: Hybrid Hash Join

- During the partitioning phase, keep Partition 0 for R (aka. R_0) in memory.
  - e.g. B = 1000 and R = 5000. Only need to partition R 5-ways (so each partition can fit in memory for the second phase). Keep R_0 in memory, in a hash table.
  - In the second phase, for tuples that map to S_0, simply probe the R_0 hash table.
  - If the keys are skewed, could adjust the hash function to map the hot keys to R0.
  - Difficult to get to work correctly. Rarely done in practice. Better to "over partition."



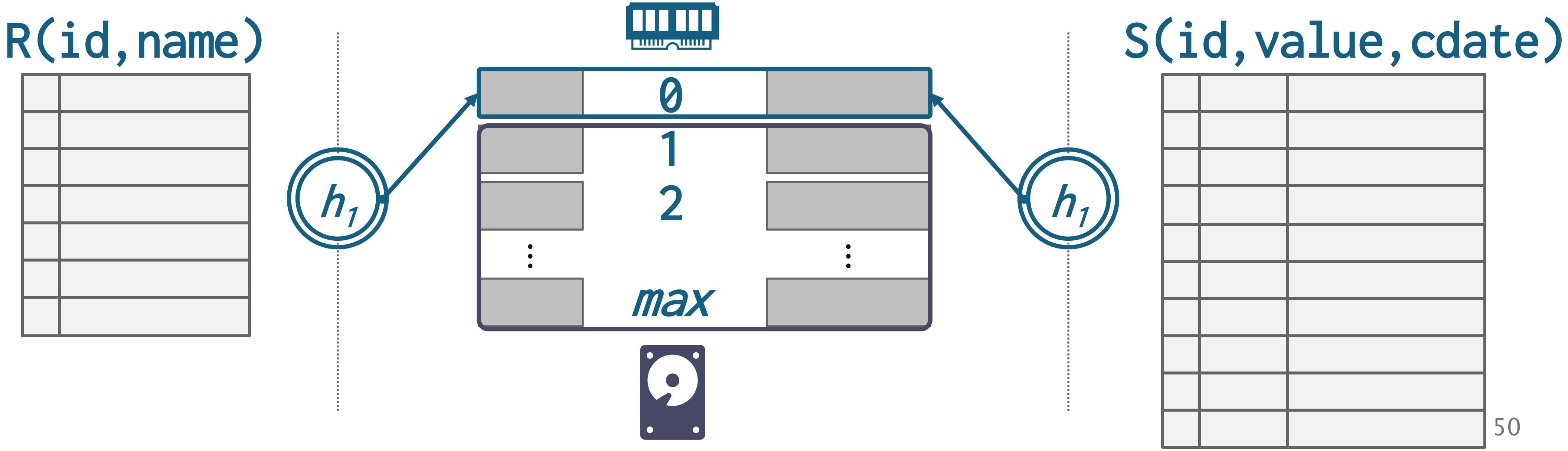
Optimization: Hybrid Hash Join

- During the partitioning phase, keep Partition 0 for R (aka. R_0) in memory.
  - e.g. B = 1000 and R = 5000. Only need to partition R 5-ways (so each partition can fit in memory for the second phase). Keep R_0 in memory, in a hash table.
  - In the second phase, for tuples that map to $S_{0,}$ simply probe the R_0 hash table.
  - If the keys are skewed, could adjust the hash function to map the hot keys to R0.
  - Difficult to get to work correctly. Rarely done in practice. Better to “over partition.”



Optimization: Hybrid Hash Join

- During the partitioning phase, keep Partition 0 for R (aka. R_0) in memory.
  - e.g. B = 1000 and R = 5000. Only need to partition R 5-ways (so each partition can fit in memory for the second phase). Keep R_0 in memory, in a hash table.
  - In the second phase, for tuples that map to S_0, simply probe the R_0 hash table.
  - If the keys are skewed, could adjust the hash function to map the hot keys to R0.
  - Difficult to get to work correctly. Rarely done in practice. Better to “over partition.”



Hash Join Observations

- The probe-side table can be any size.
  - Only the build-side table (or its partitions) need to fit in memory
- If we know the size of the build-side table, then we can use a static hash table.
  - Less computational overhead
- If we do not know the size, then we must use a dynamic hash table or allow for overflow pages.

SCHOOL OF DATA SCIENCE

Join Algorithms: Summary

Algorithm	IO Cost	Example
Naïve Nested Loop Join	$M + (m \cdot N)$	1.3 hours
Block Nested Loop Join	$M + (\lceil M / (B\text{-}2) \rceil \cdot N)$	0.55 seconds
Index Nested Loop Join	$M + (m \cdot C)$	Variable
Sort-Merge Join	$M + N$ + (sort cost)	0.75 seconds
Hash Join	$3 \cdot (M + N)$	0.45 seconds

Conclusion

- Hashing is almost always better than sorting for operator execution.
- Caveats:
  - Sorting is better on non-uniform data.
  - Sorting is better when result needs to be sorted.
- Good DBMSs use either (or both).

Next Lecture

- Composing operators together to execute queries.