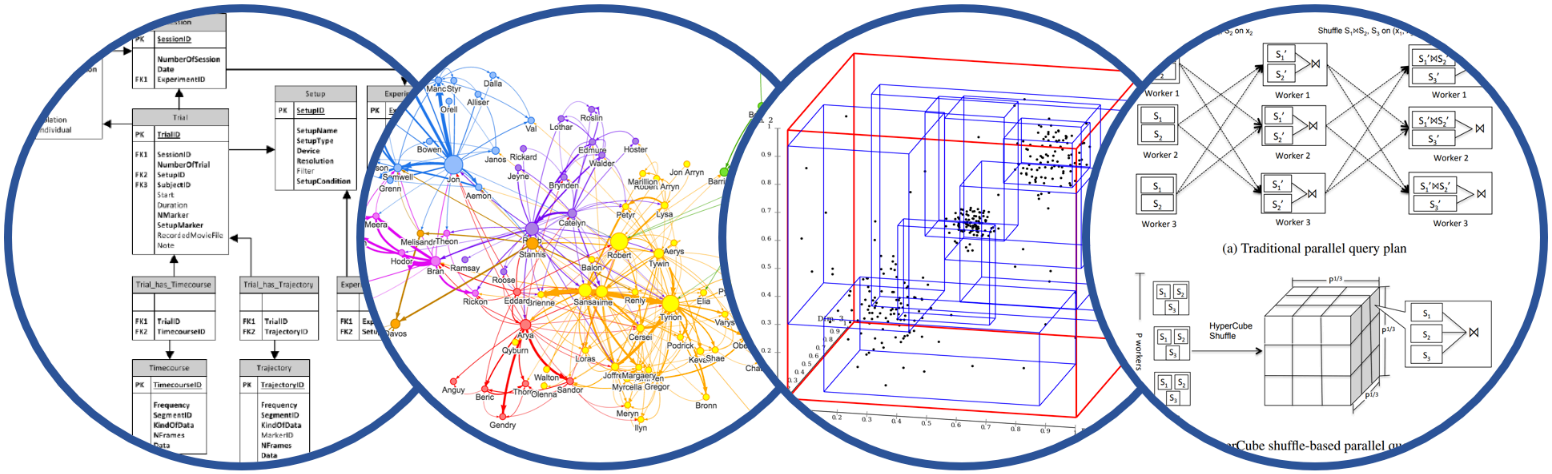


Database System Internals

Query Optimization (part 2)

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Today's Agenda

- Recap Partitioned Hash-join
  (from two lectures ago)

- Finish discussing cardinality estimation

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$

- Assuming $B(R_1)=B(R_2)=\ldots= B(R_k)$, we have
  $B(R_i) = B(R)/k$, for all i

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$
- Assuming $B(R_1)=B(R_2)=\ldots= B(R_k)$, we have
  $B(R_i) = B(R)/k$, for all i
- Goal: each R_i should fit in main memory:
  $B(R_i) \leq M$

Partitioned Hash Algorithms

- Partition R it into k buckets on disk:
  $R_1, R_2, R_3, \ldots, R_k$

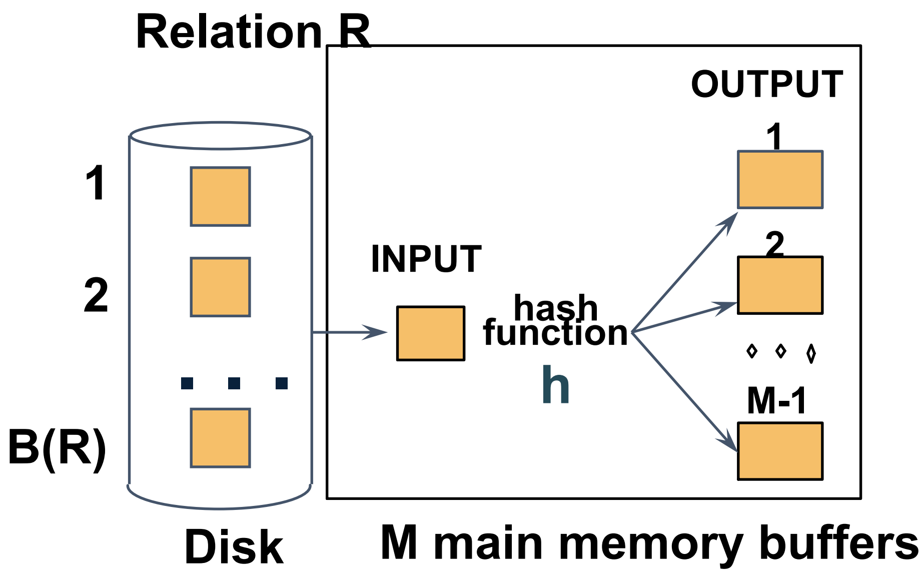
- Assuming $B(R_1)=B(R_2)=\ldots= B(R_k)$, we have
  $B(R_i) = B(R)/k$, for all i

- Goal: each R_i should fit in main memory:
  $B(R_i) \leq M$

How do we choose k?

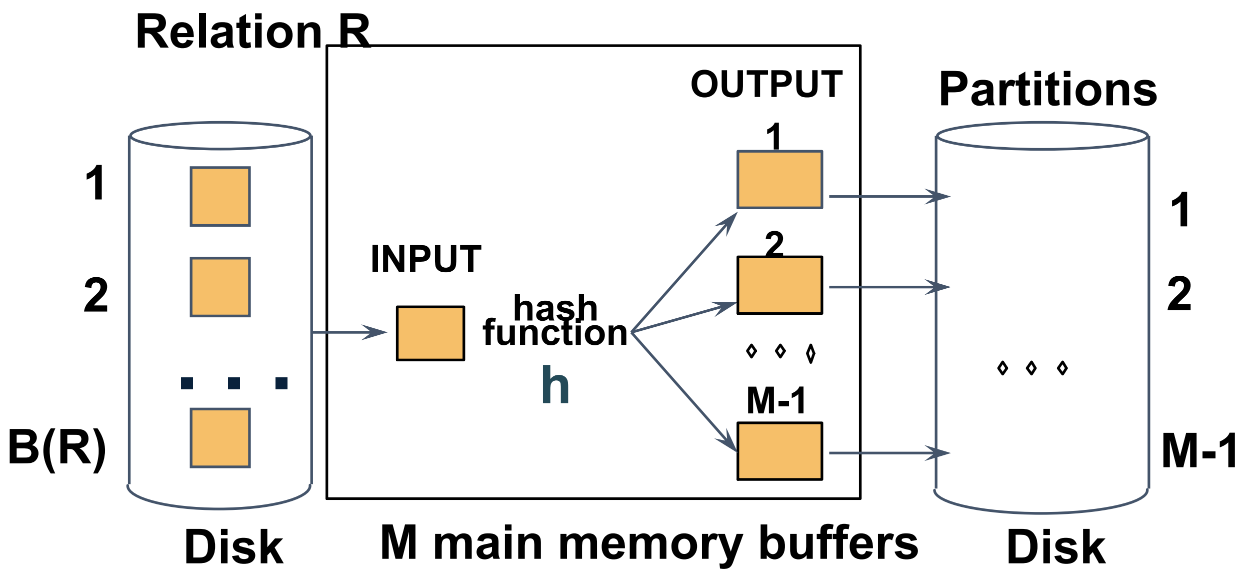
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



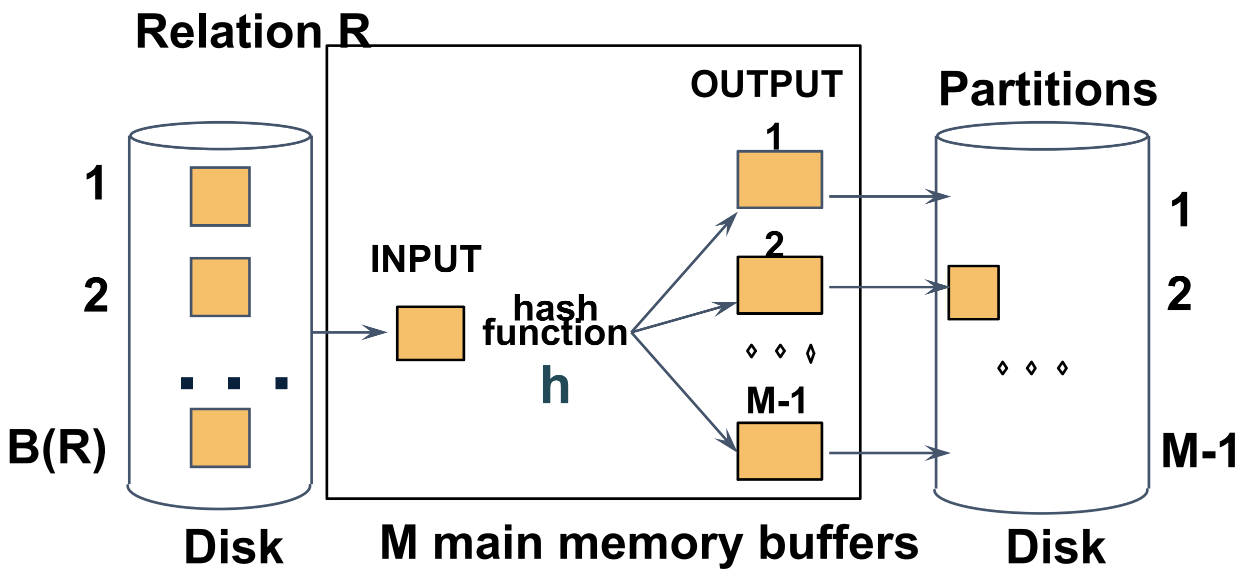
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



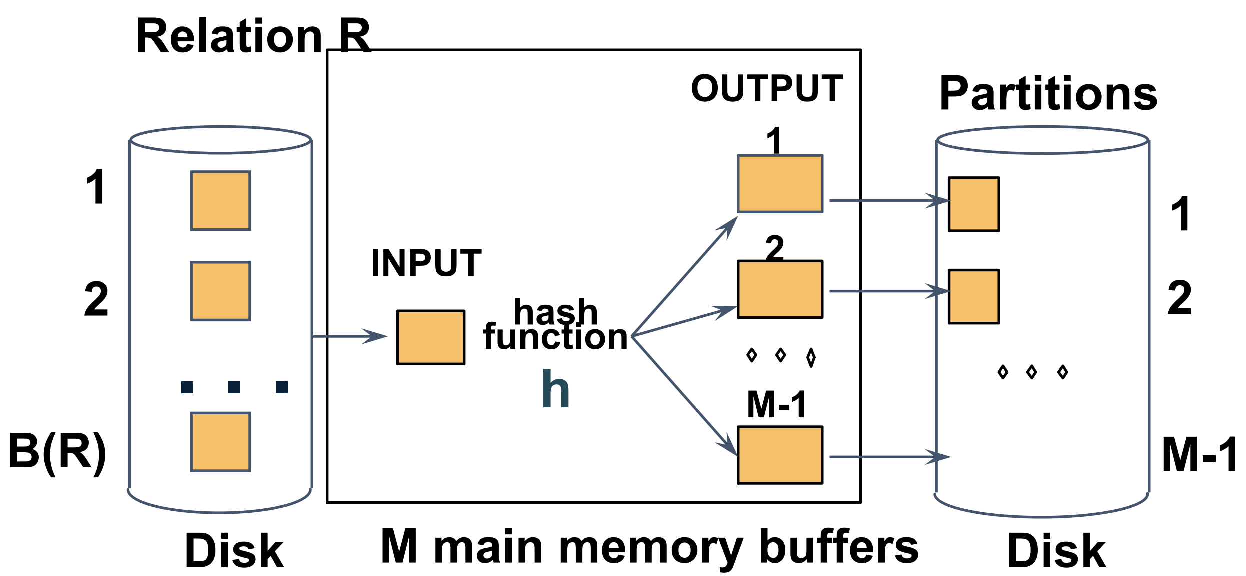
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



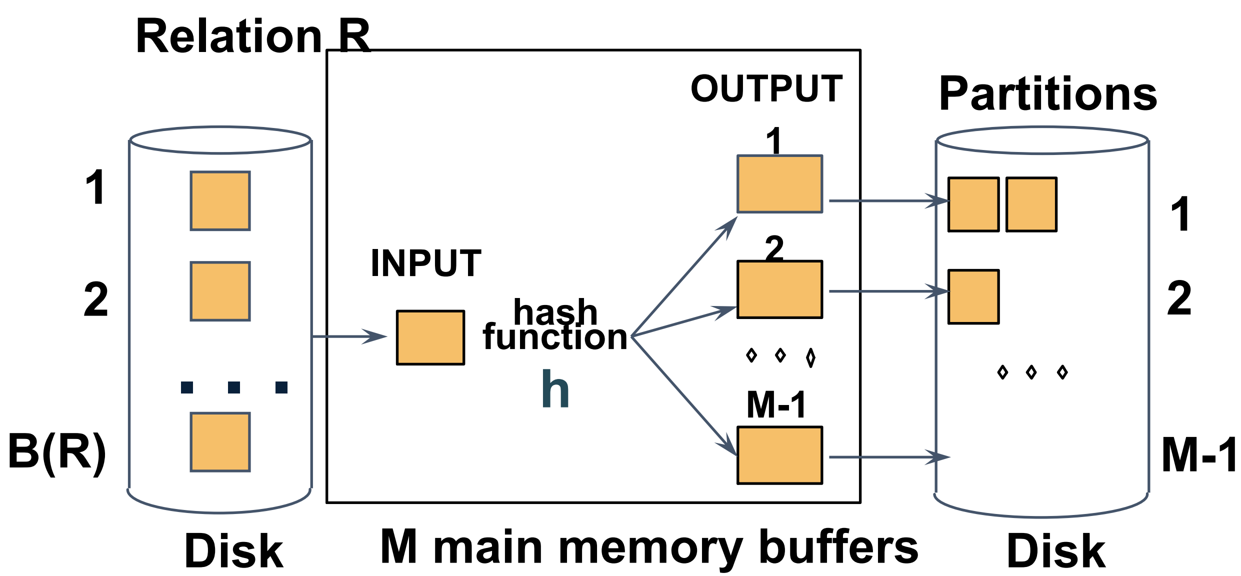
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



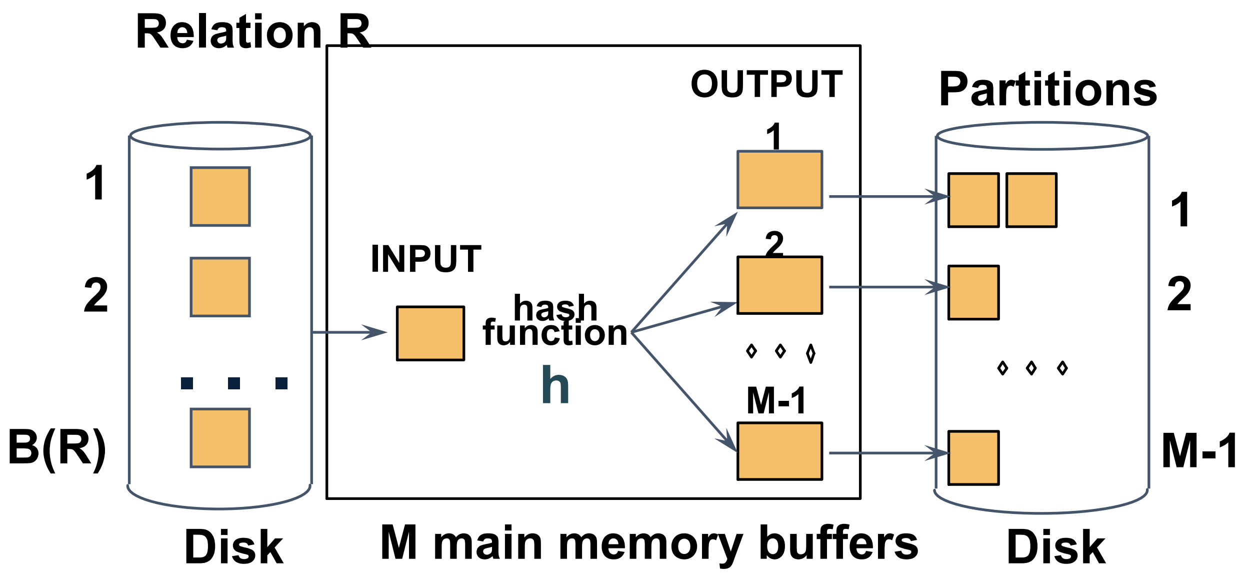
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



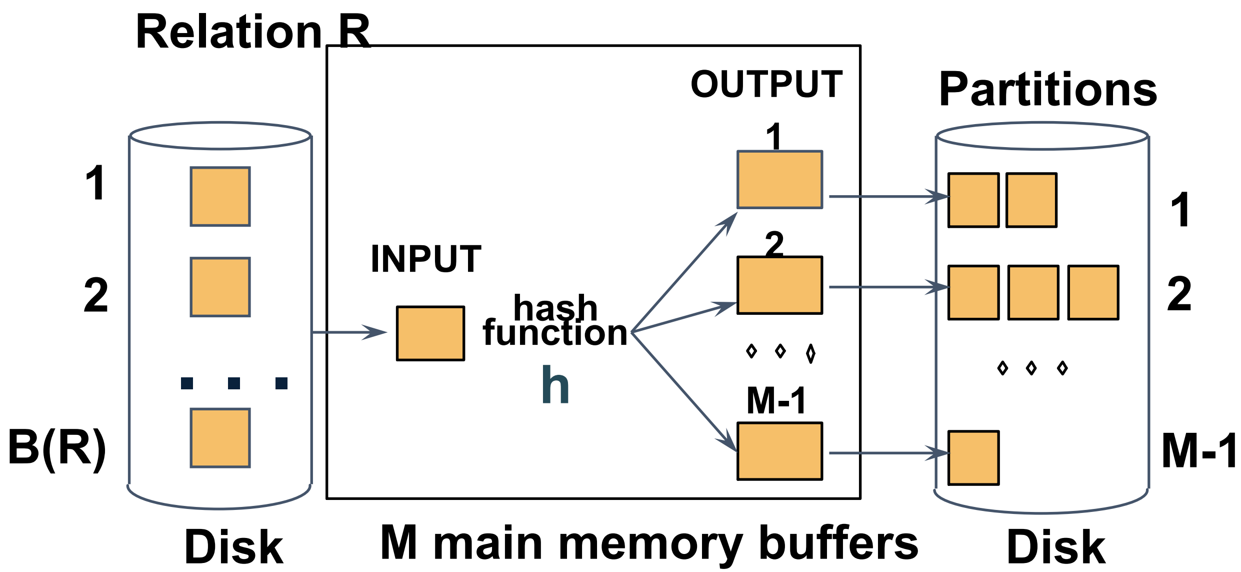
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



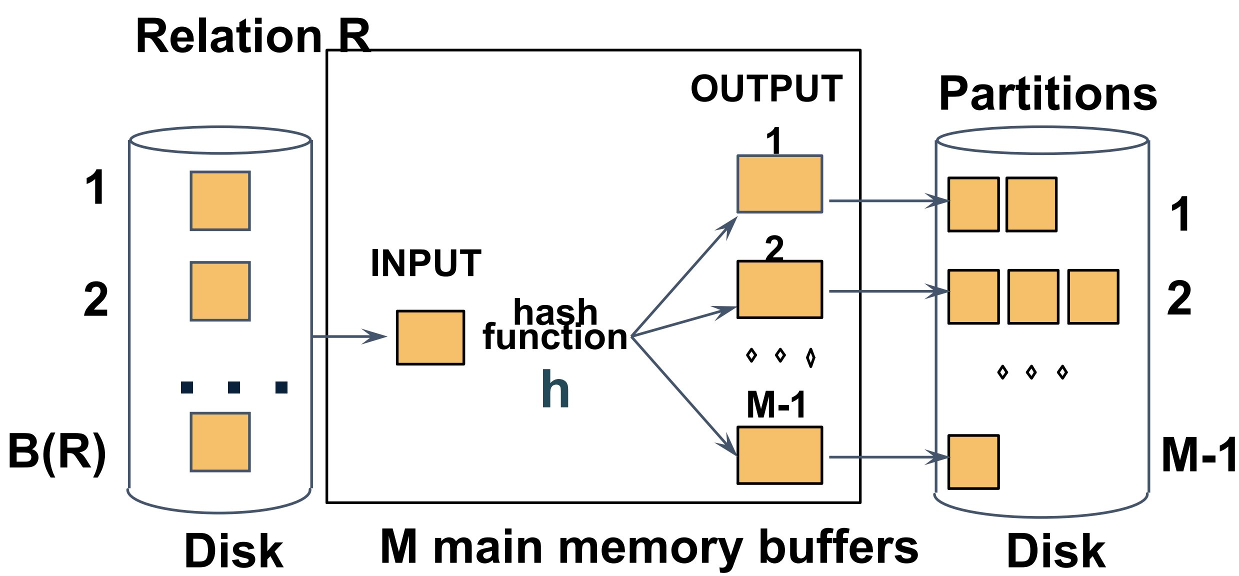
Partitioned Hash Algorithms

- We choose k = M-1
  Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



Partitioned Hash Algorithms

- We choose $k = M-1$
  Each bucket has size approx. $B(R)/(M-1) \approx B(R)/M$



Assumption: $B(R)/M \leq M$, i.e. $B(R) \leq M^2$

Partitioned Hash Join (Grace-Join)

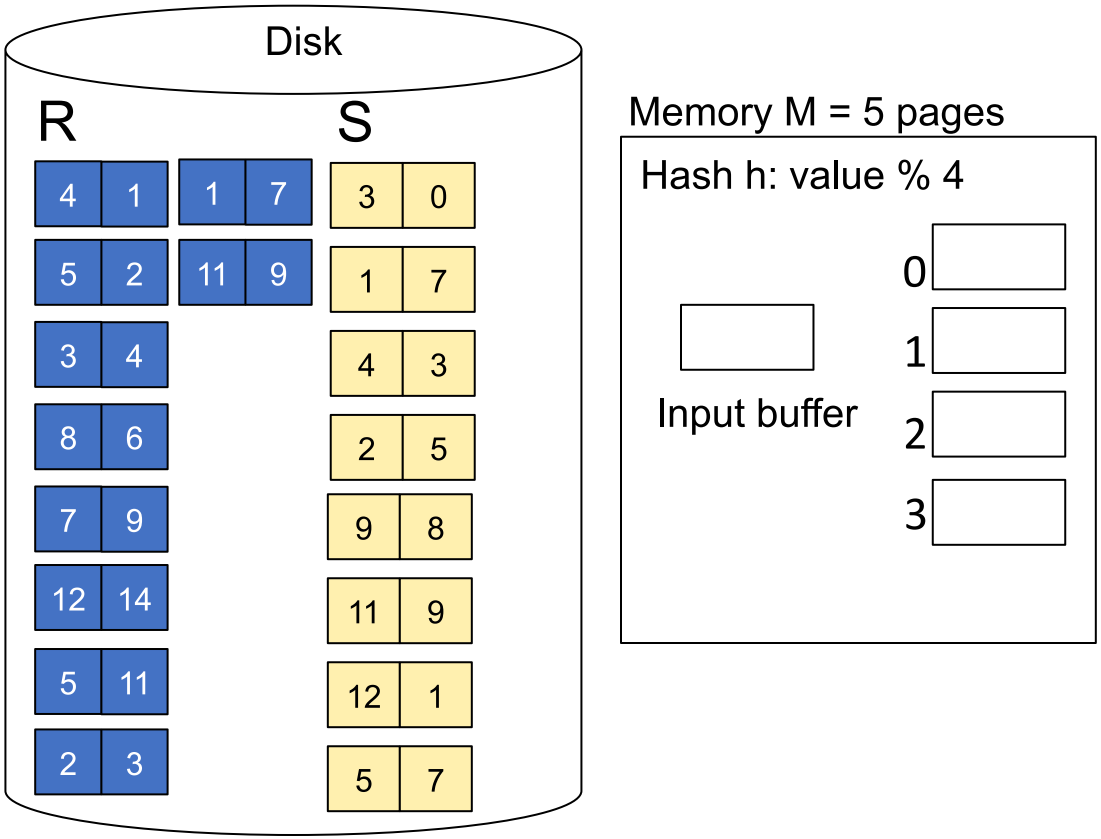
$R \bowtie S$

- Step 1:
  - Hash S into M-1 buckets
  - Send all buckets to disk
- Step 2
  - Hash R into M-1 buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

Note: partitioned hash-join is sometimes called *grace-join*

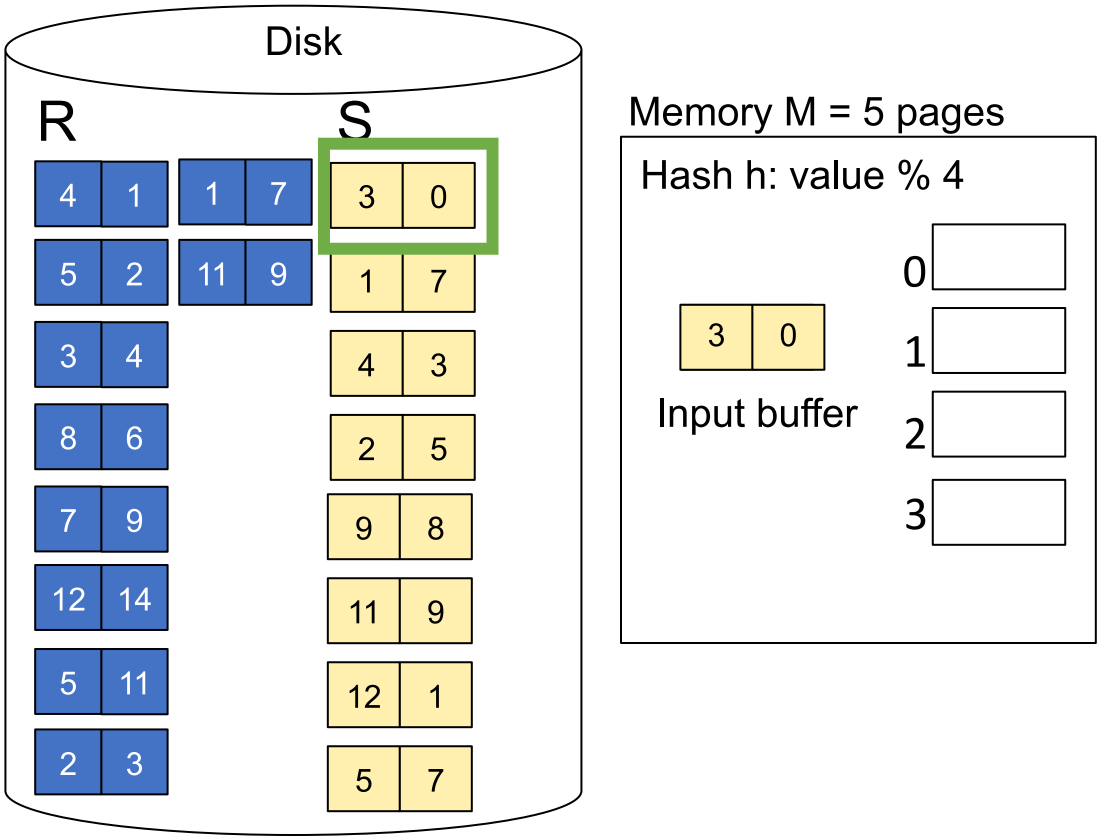
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into M-1 (=4 buckets)



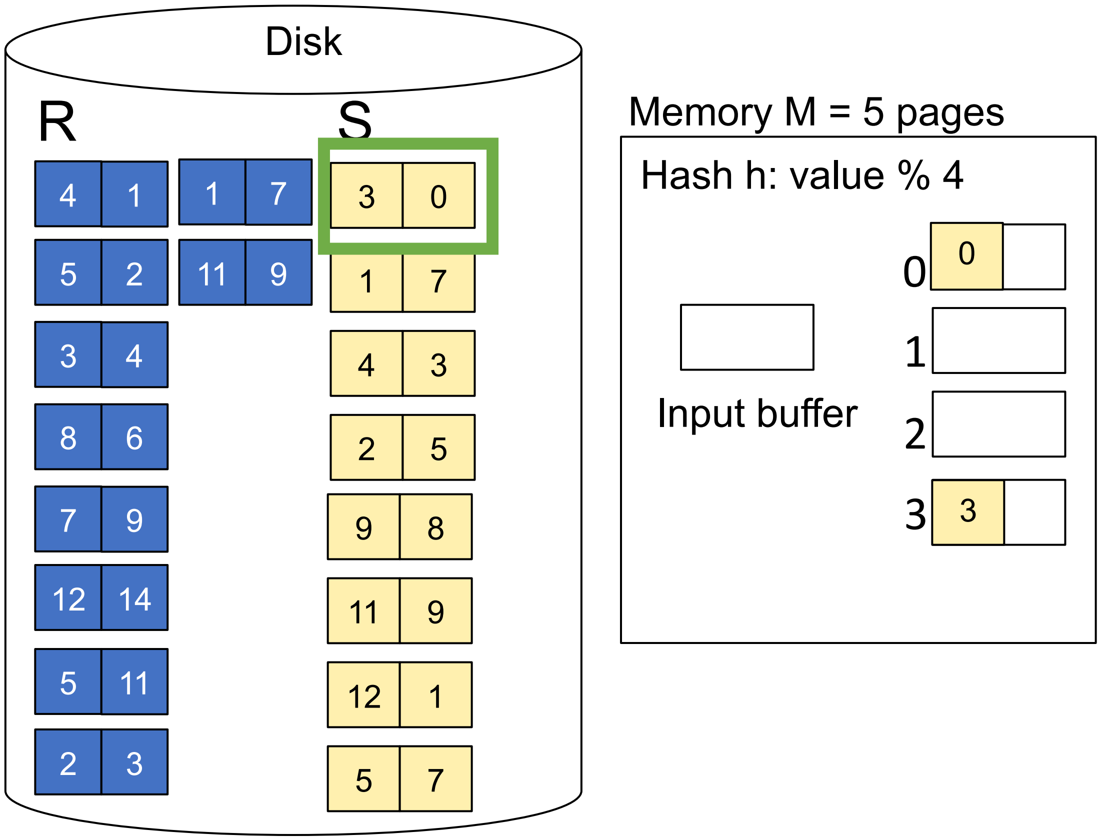
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into M-1 (=4 buckets)



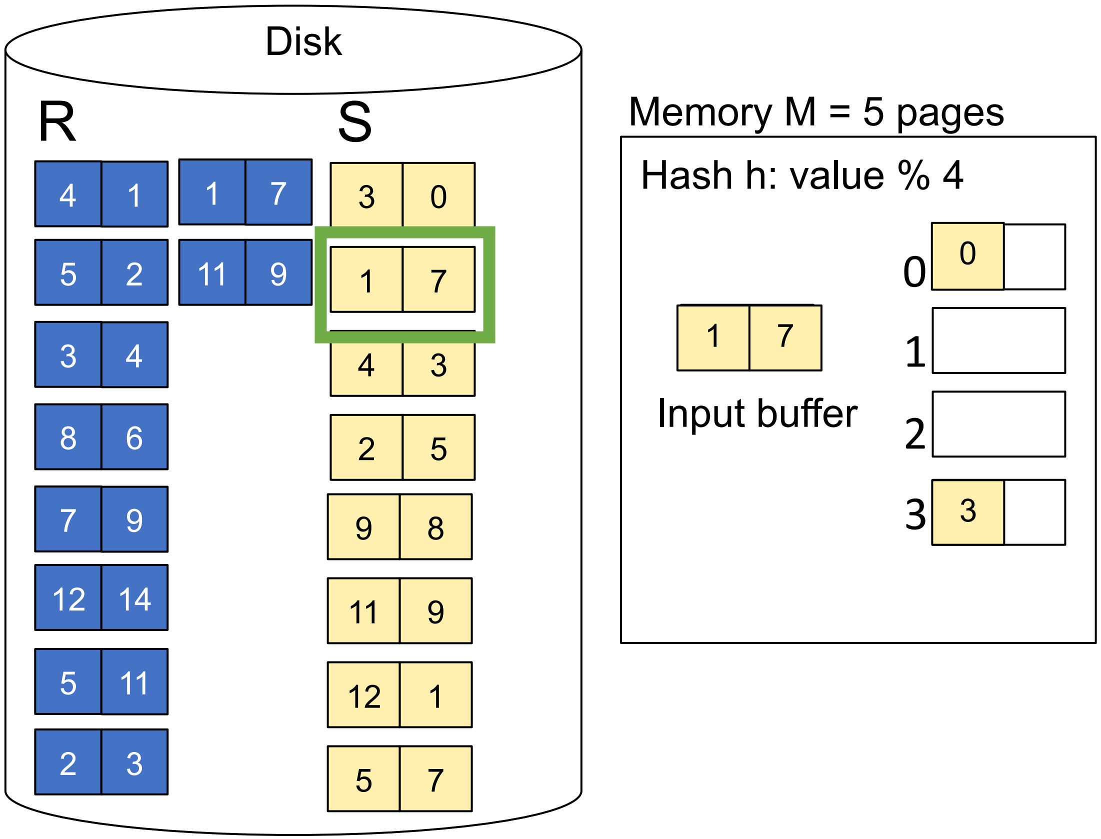
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets



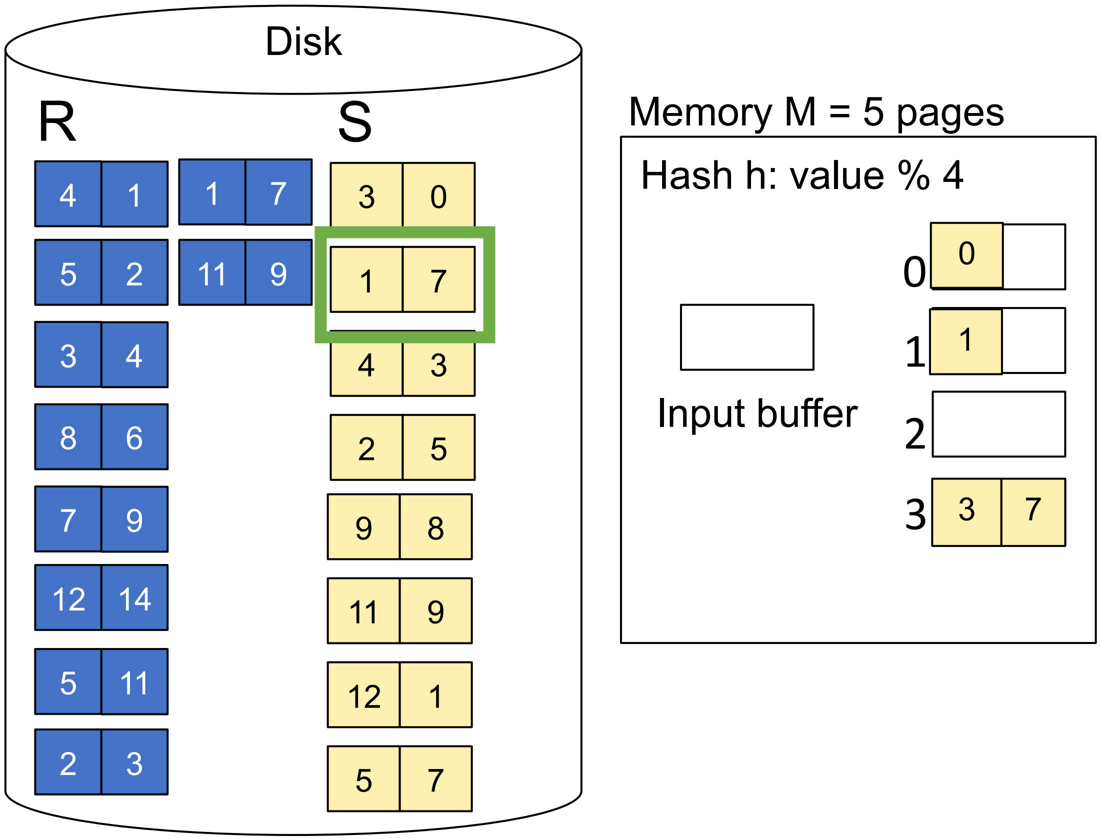
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets



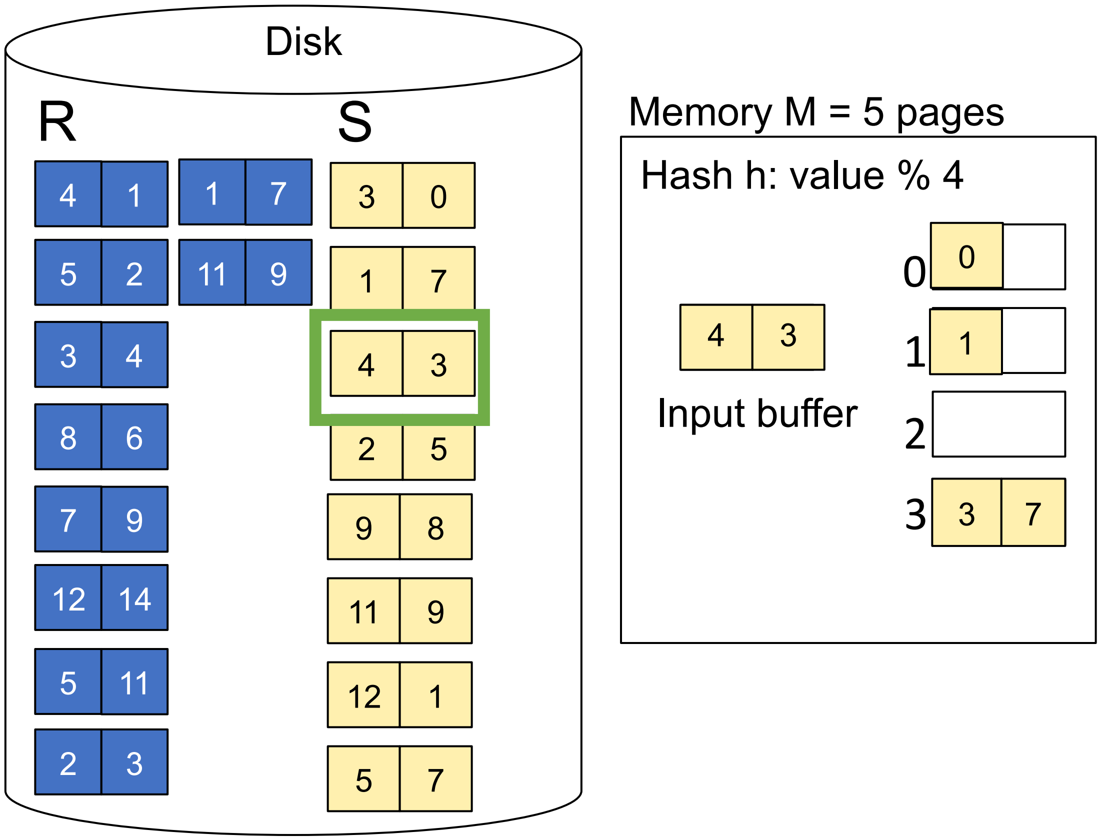
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets



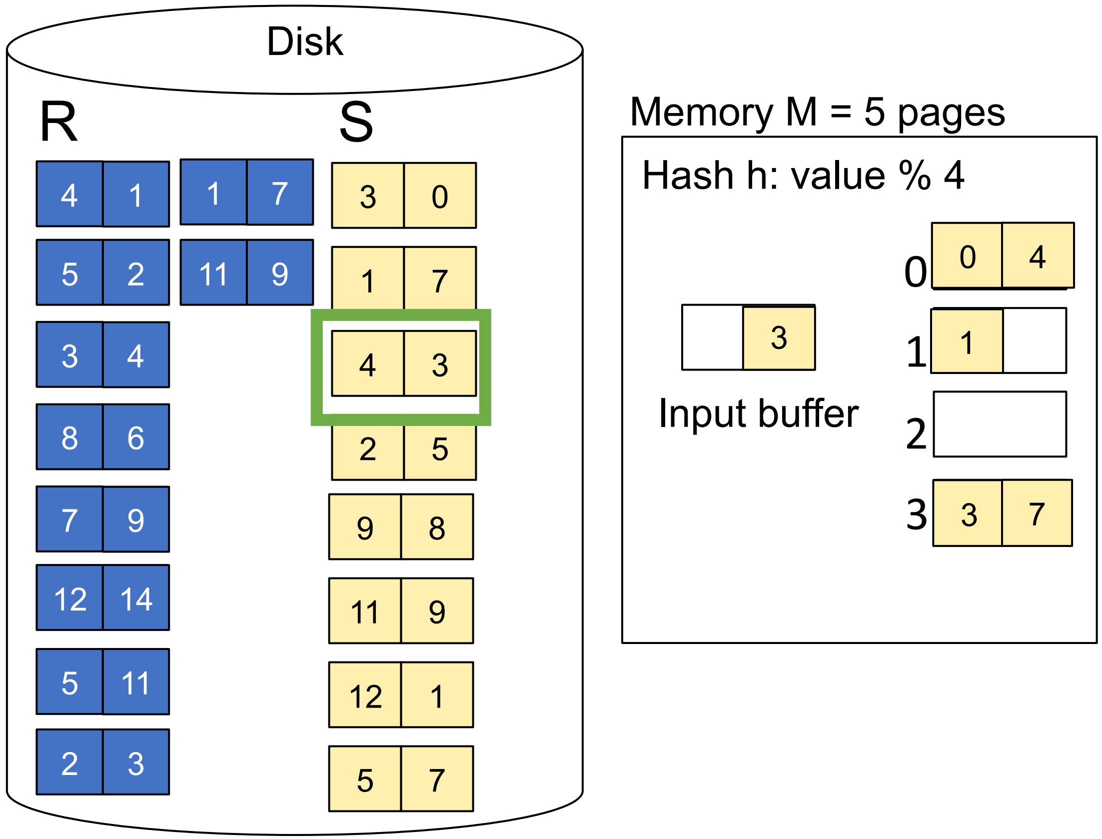
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets



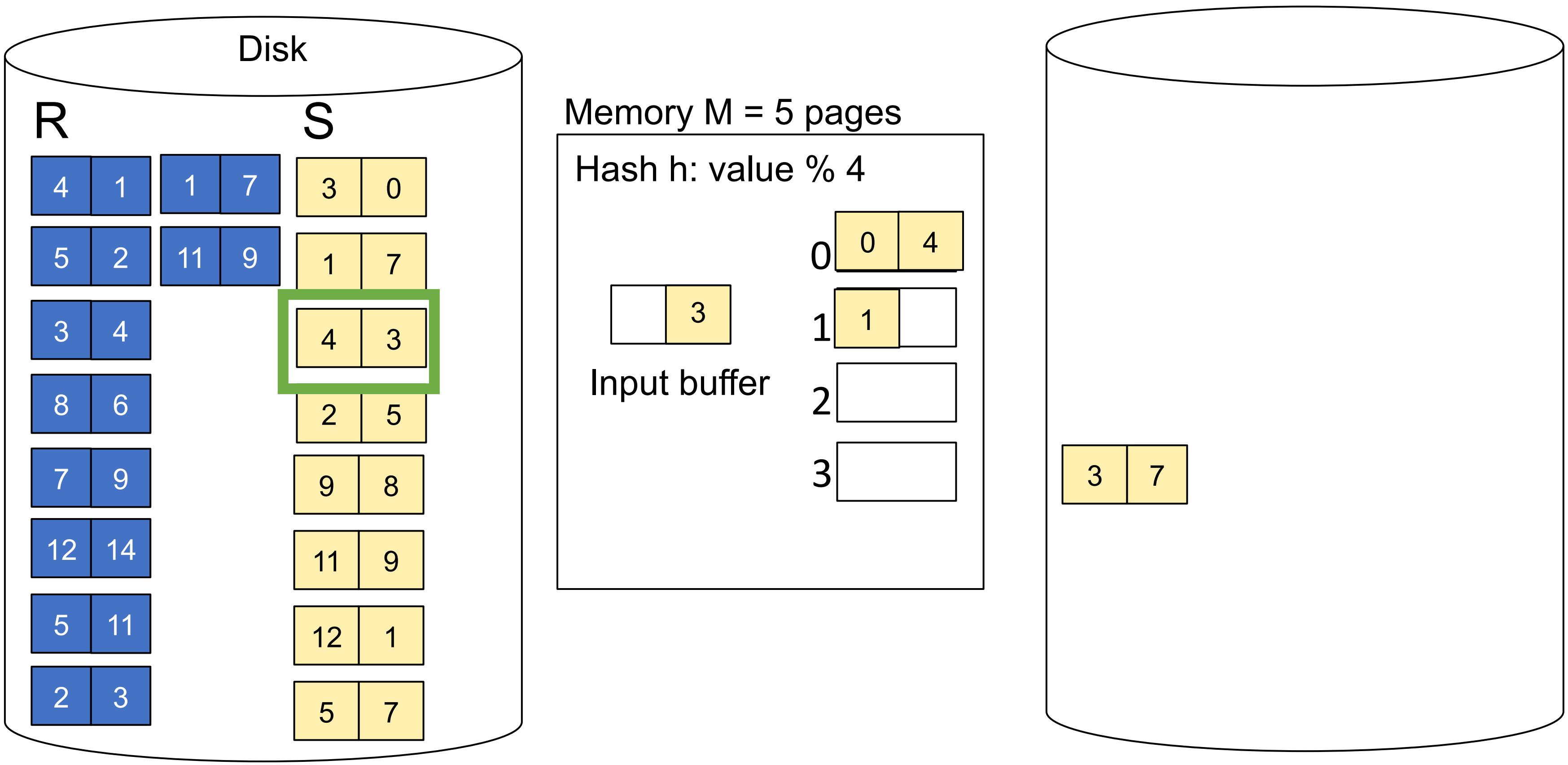
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk



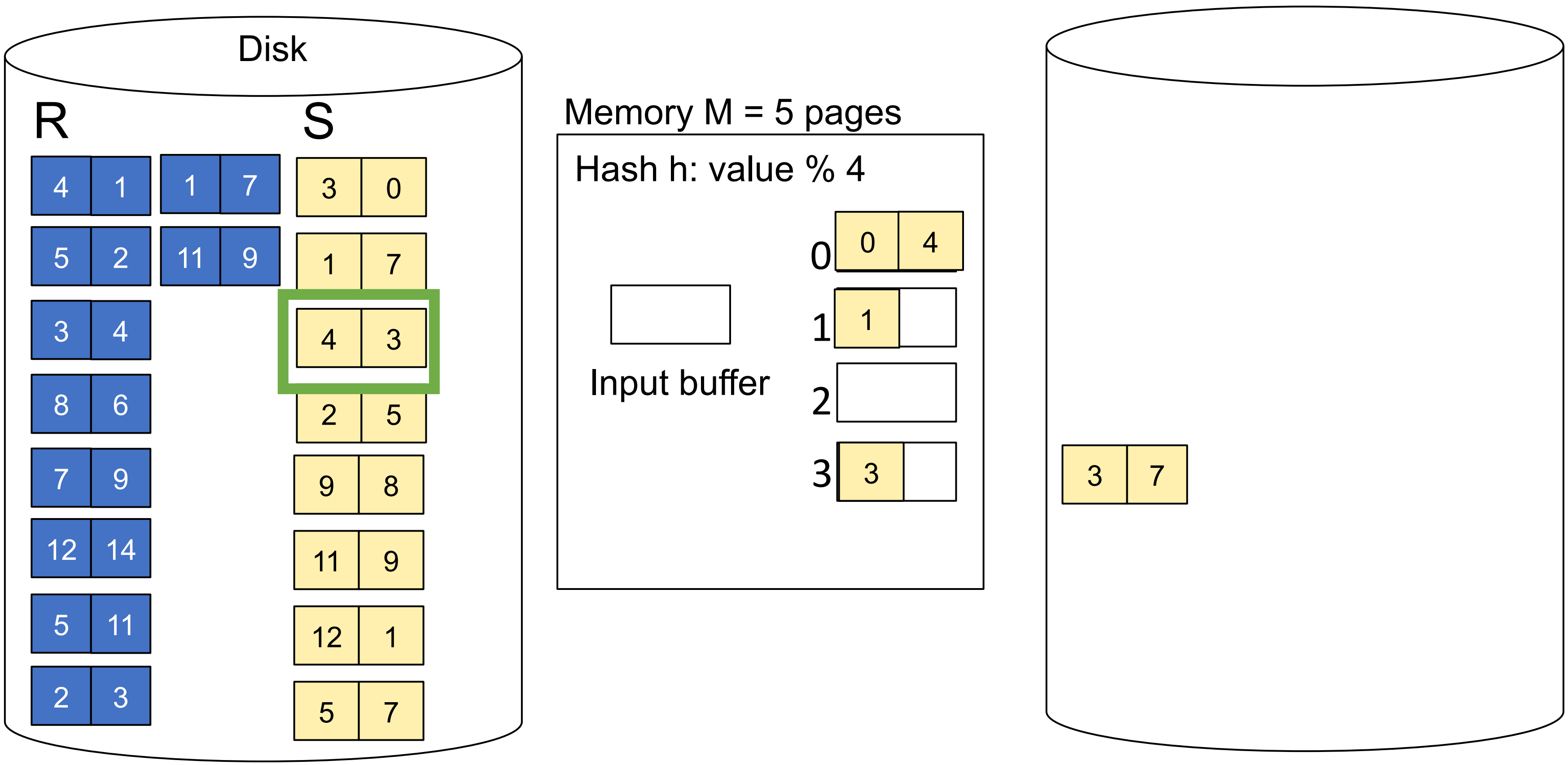
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk



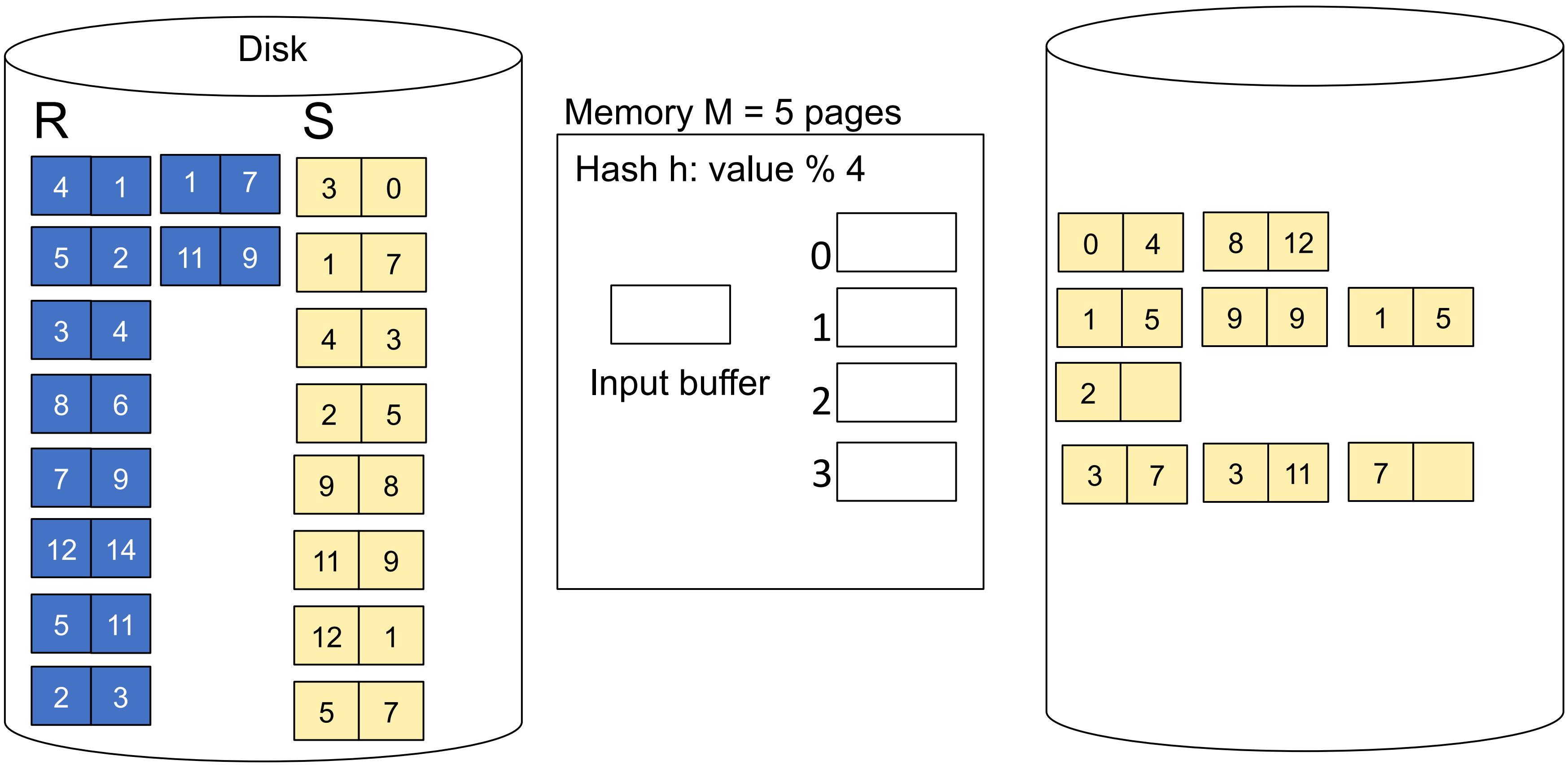
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
When a bucket fills up, flush it to disk



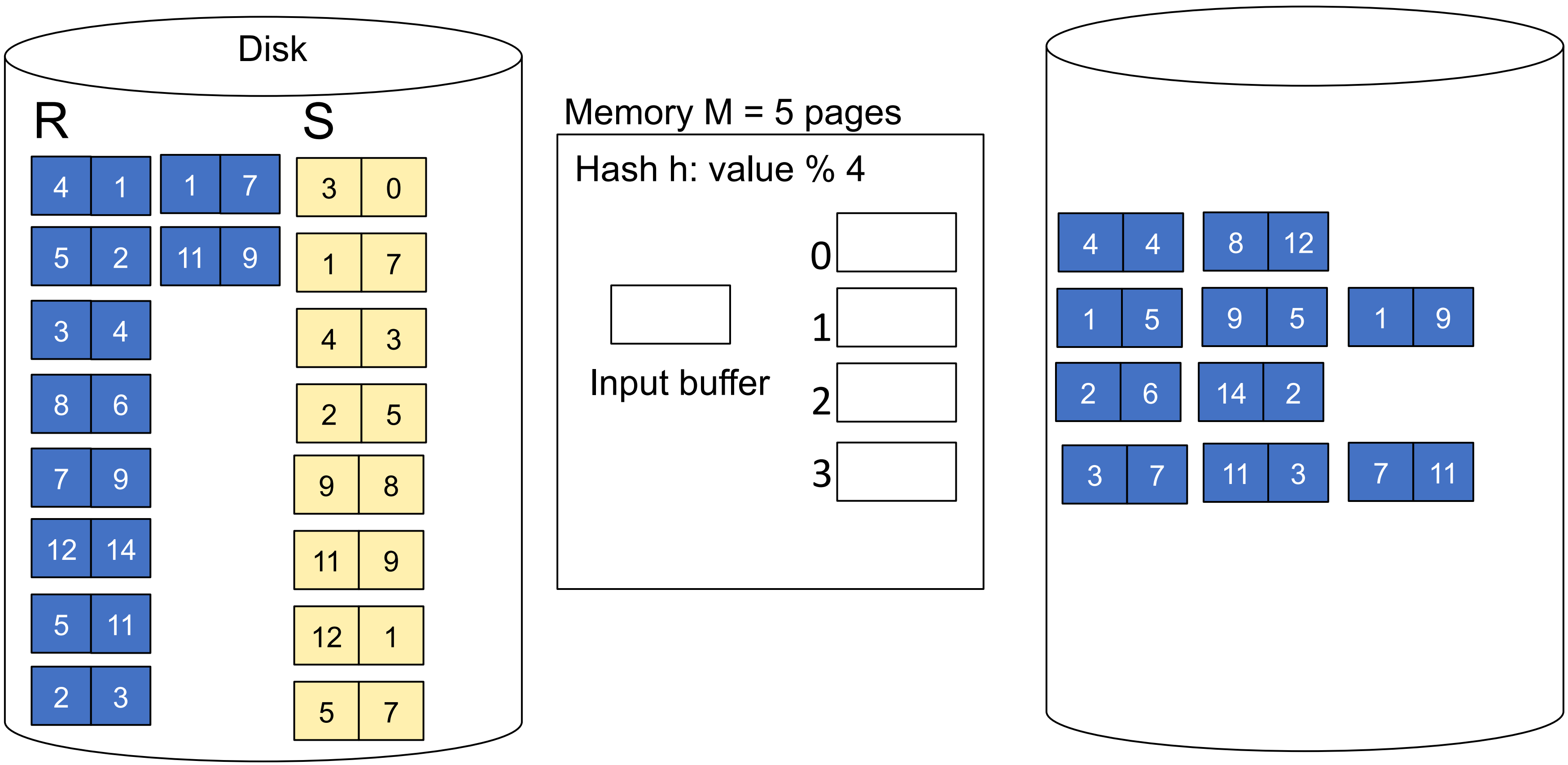
Partitioned Hash-Join Example

Step 1: Read relation S one page at a time and hash into the 4 buckets
At the end, we get relation S back on disk split into 4 buckets



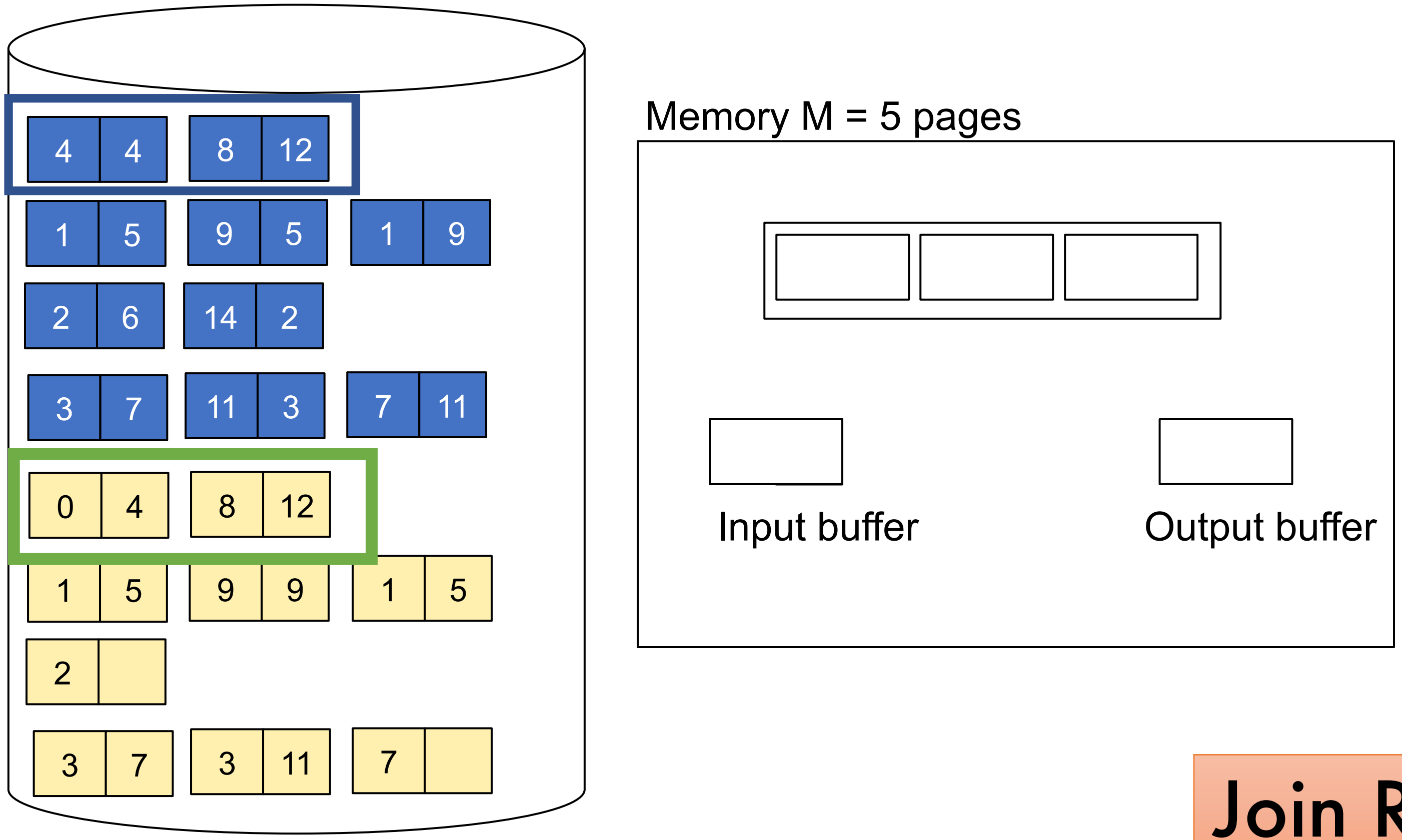
Partitioned Hash-Join Example

Step 2: Read relation R one page at a time and hash into same 4 buckets



Partitioned Hash-Join Example

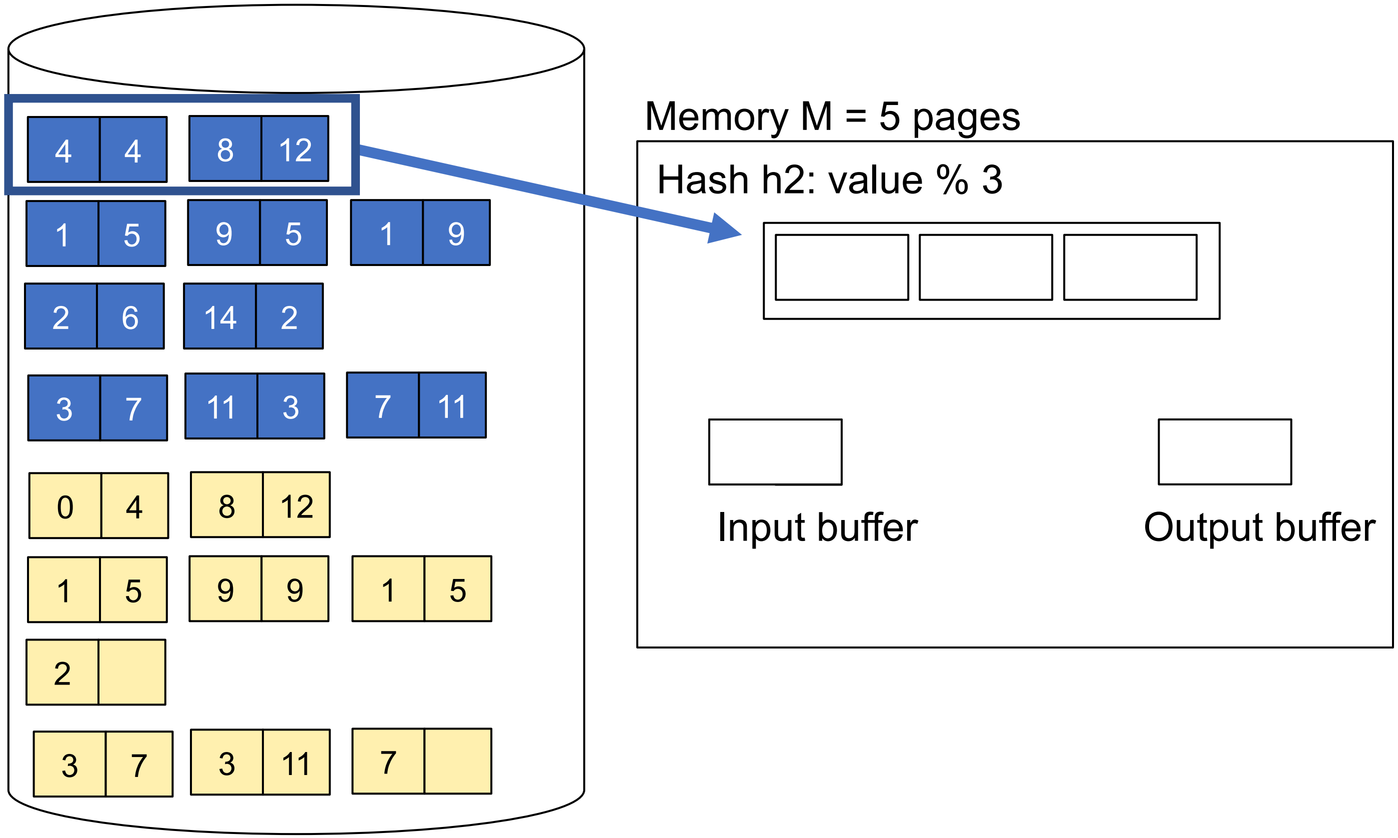
Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function



Join R1 with S1

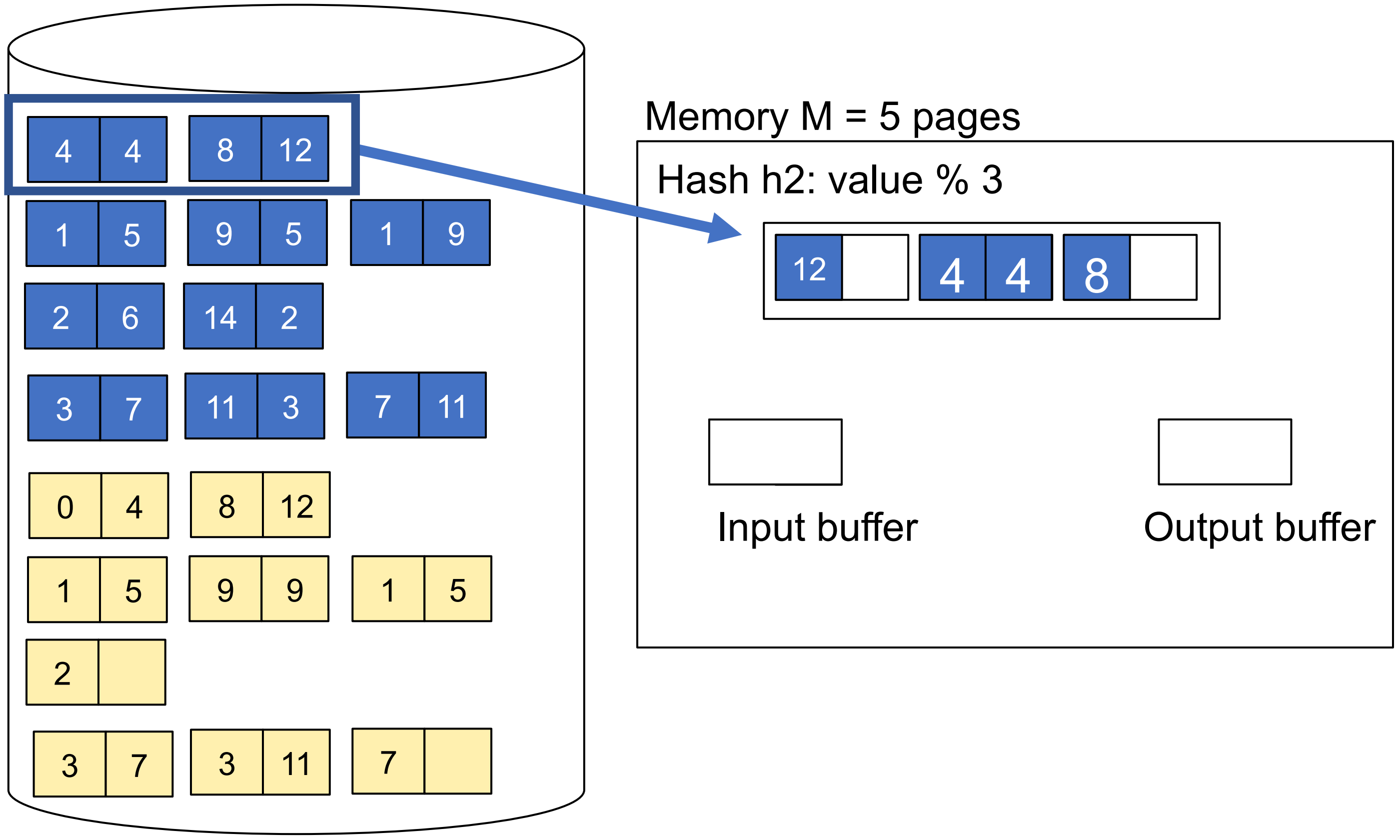
Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function



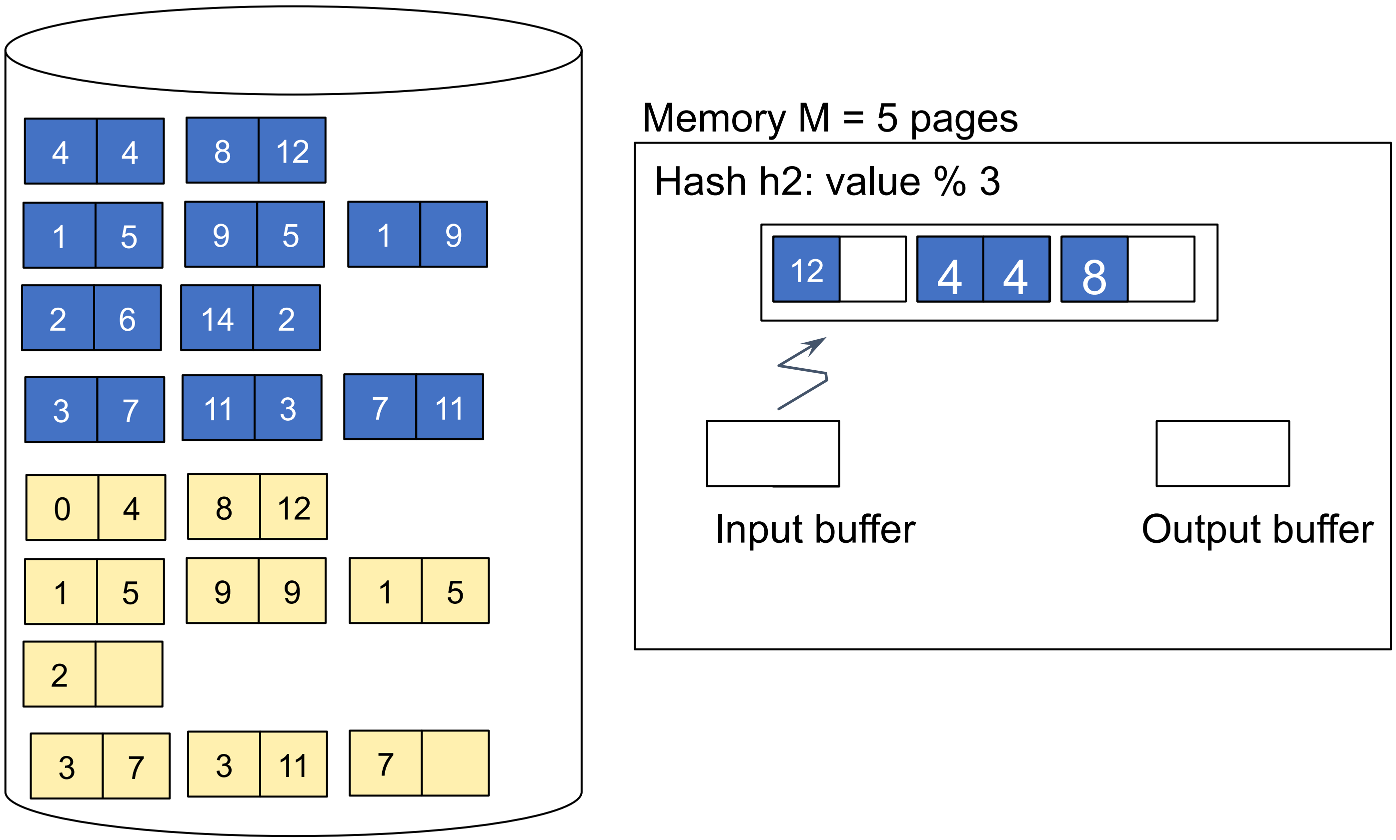
Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function



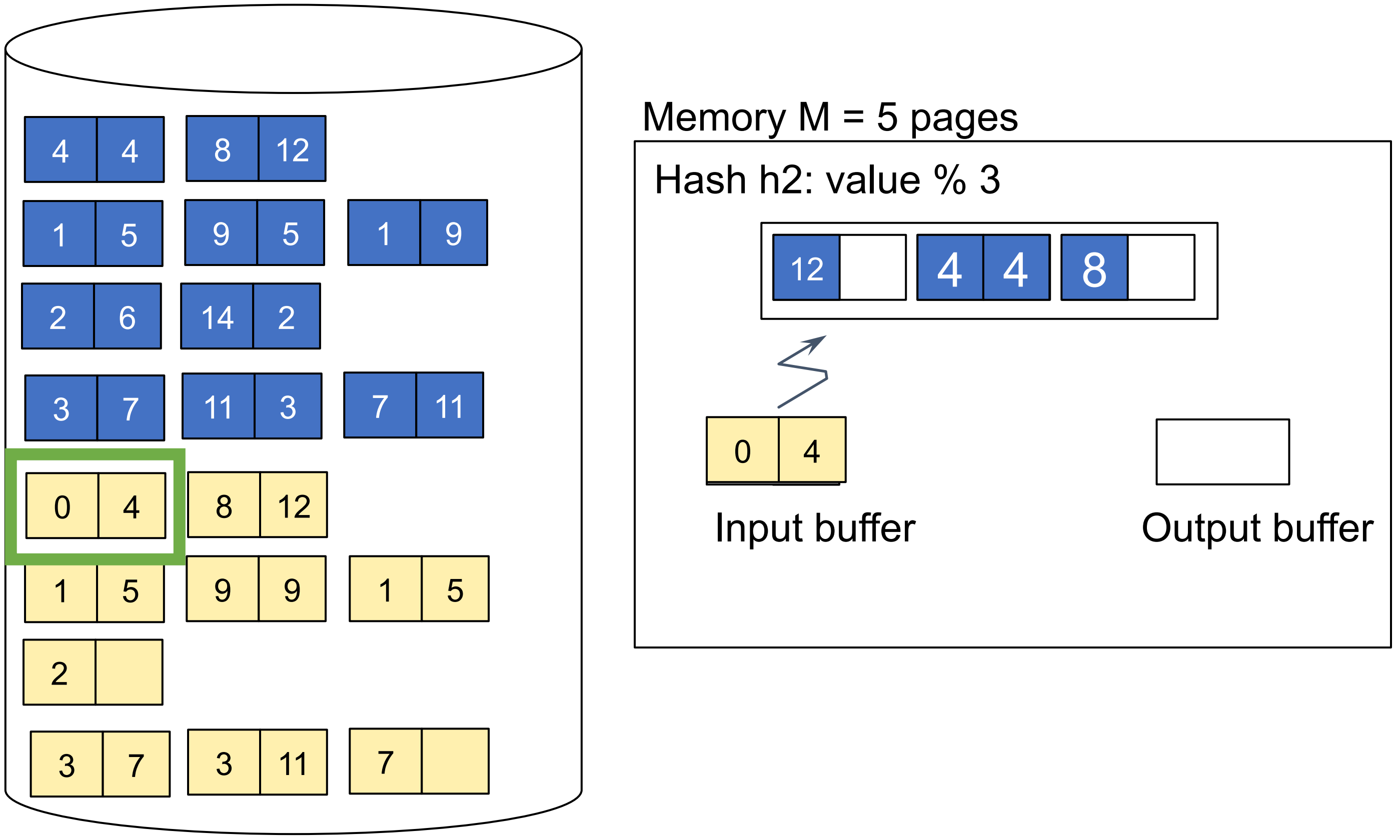
Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function



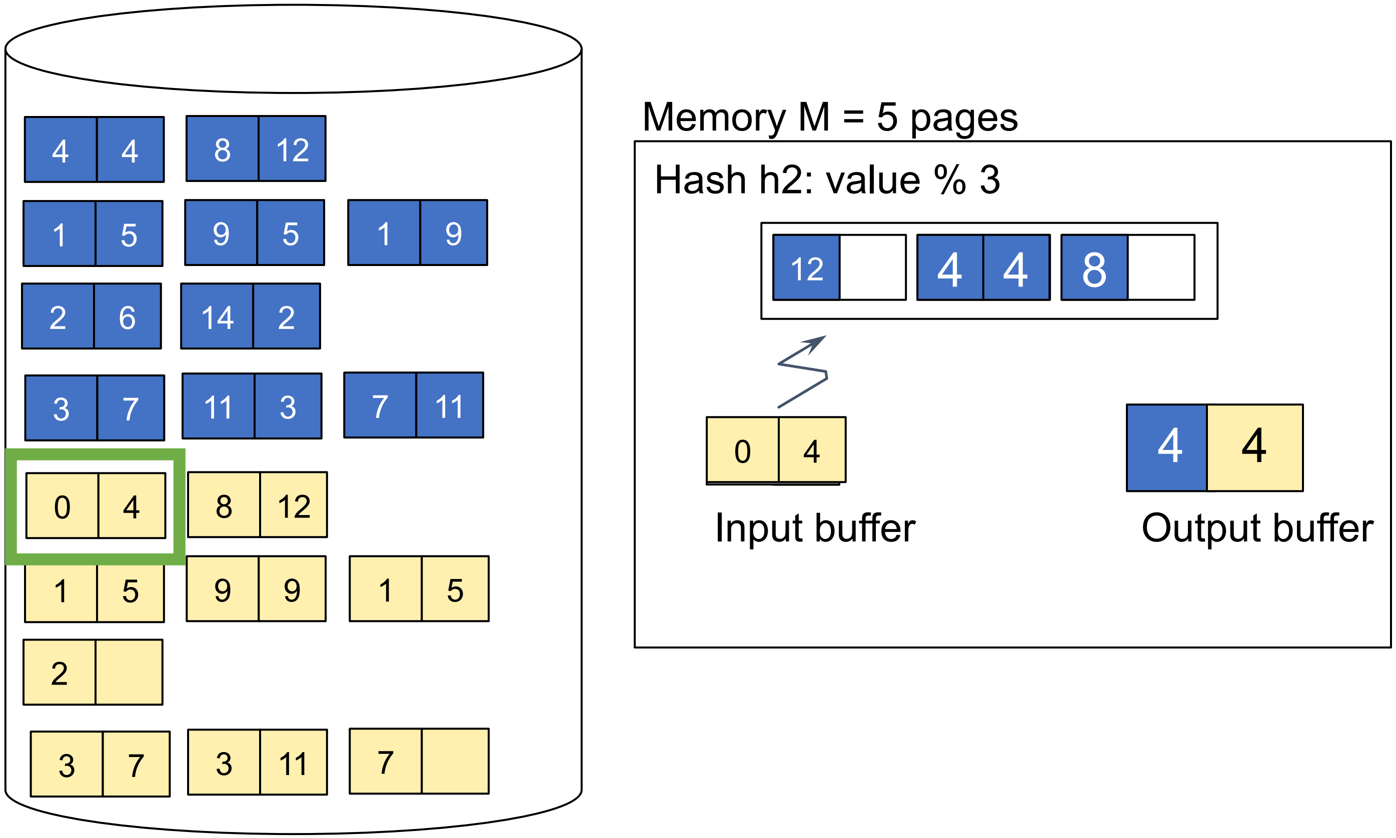
Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
Step 5: Repeat for all the buckets
Total cost: 3B(R) + 3B(S)



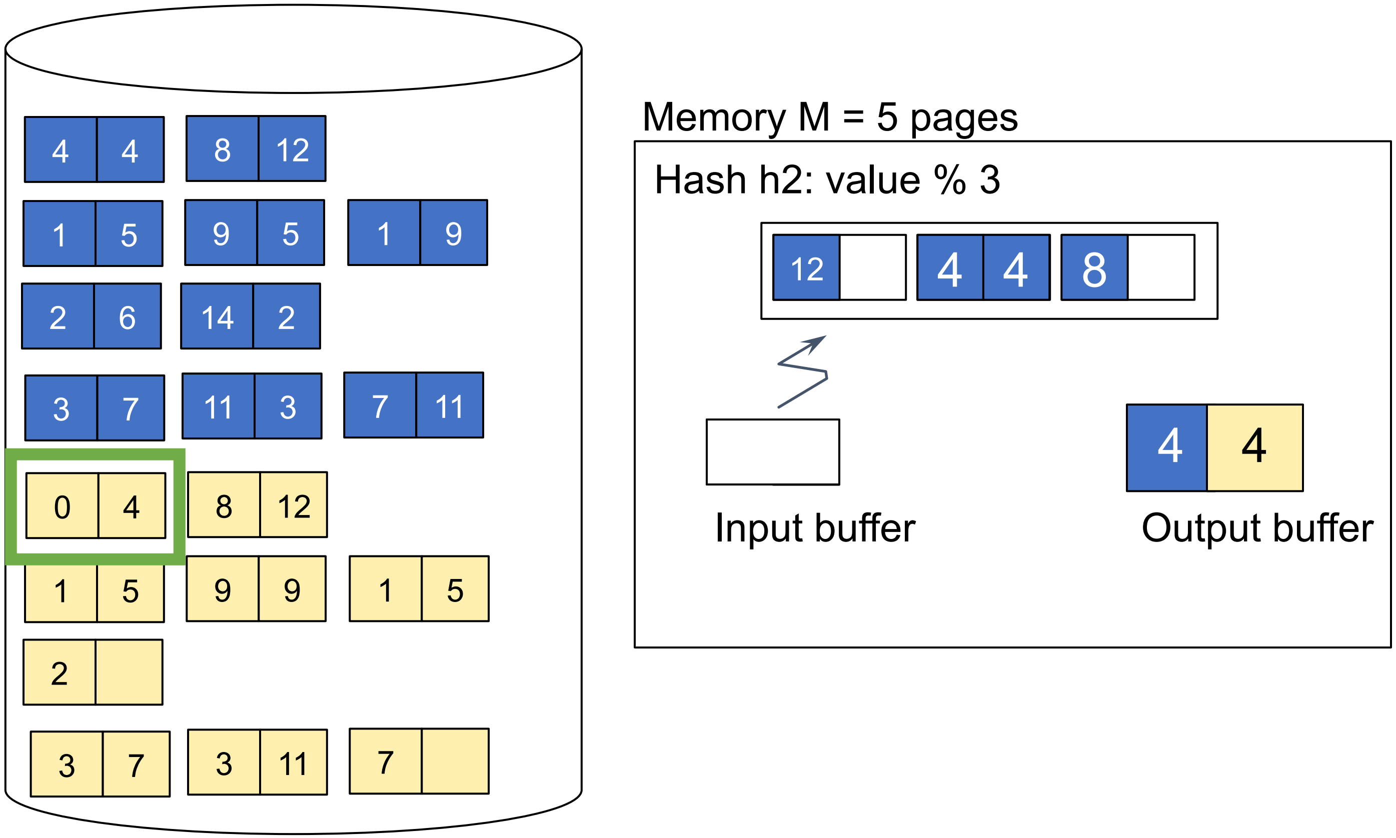
Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
Step 5: Repeat for all the buckets
Total cost: 3B(R) + 3B(S)



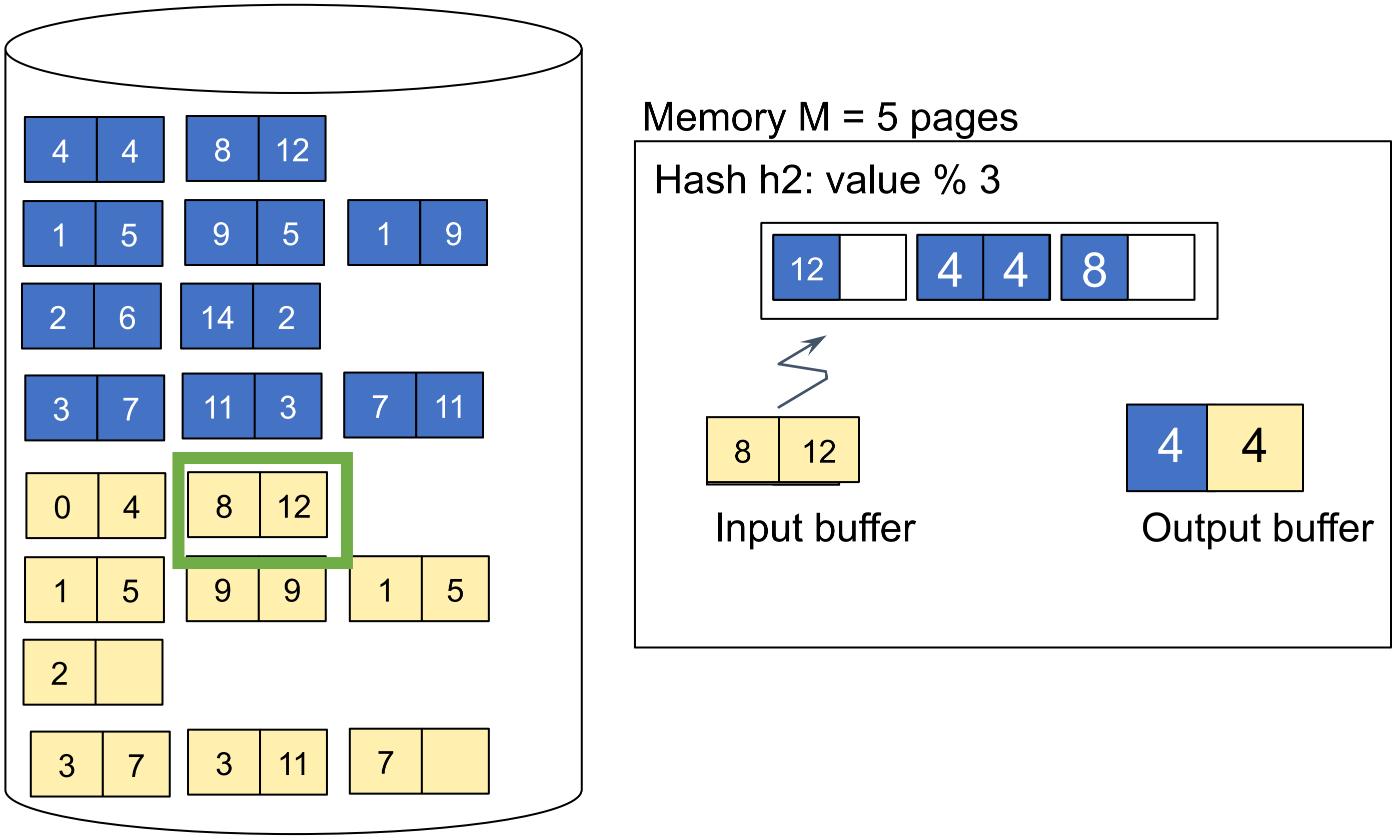
Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
Step 5: Repeat for all the buckets
Total cost: 3B(R) + 3B(S)



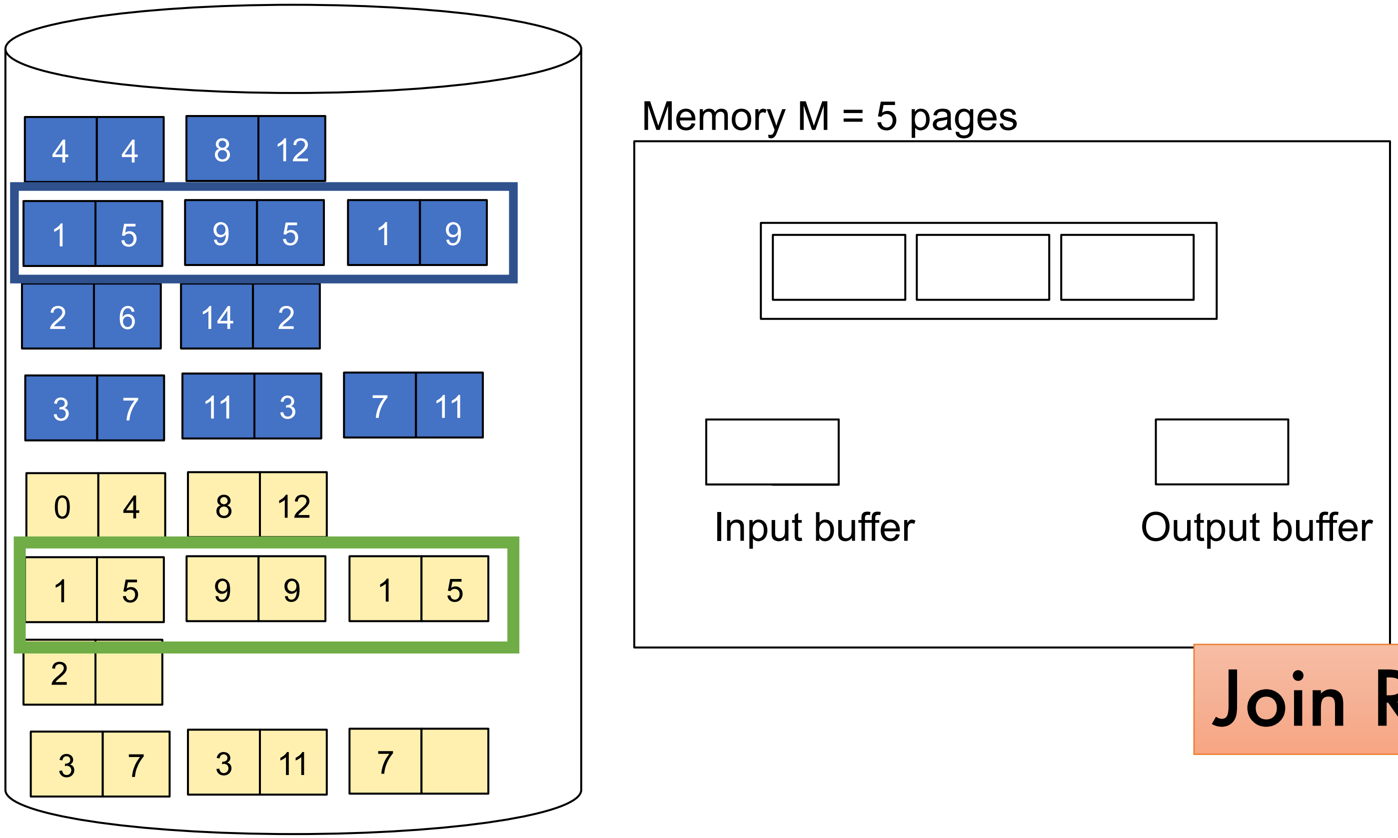
Partitioned Hash-Join Example

Step 4: Scan matching partition of S and probe the hash table
Step 5: Repeat for all the buckets
Total cost: 3B(R) + 3B(S)



Partitioned Hash-Join Example

Step 3: Read one partition of R and create hash table in memory using a ***different*** hash function



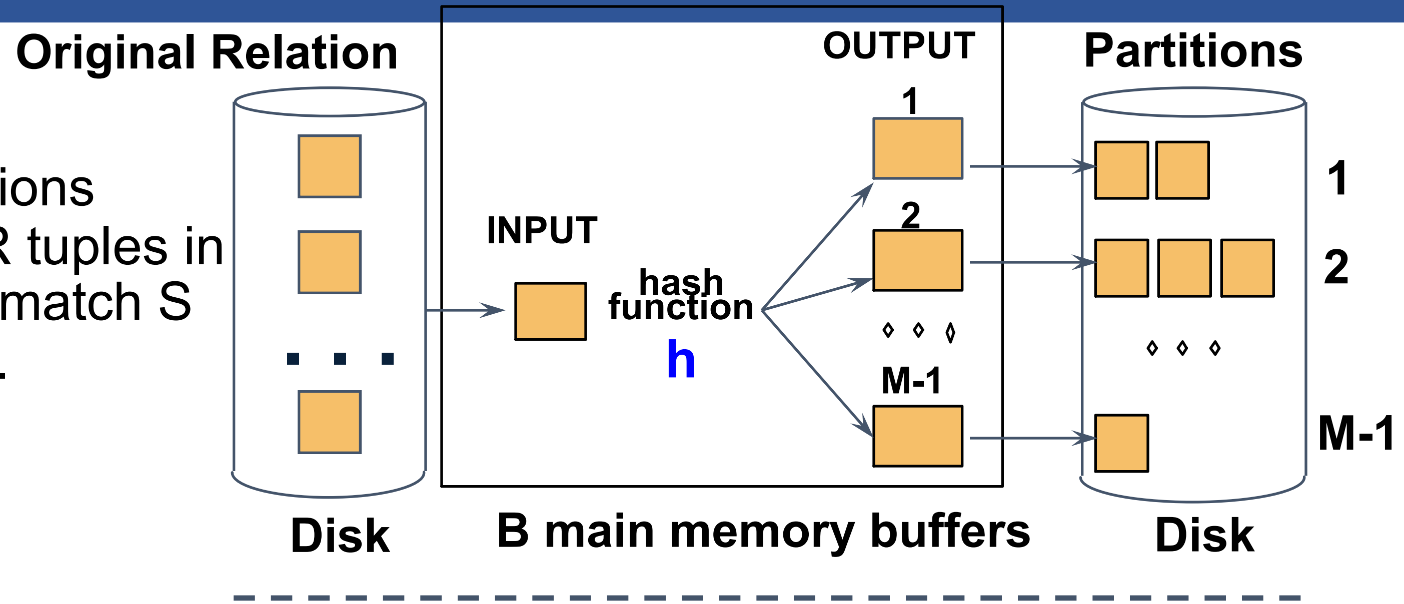
Join R2 with S2

…

Join Rk with Sk

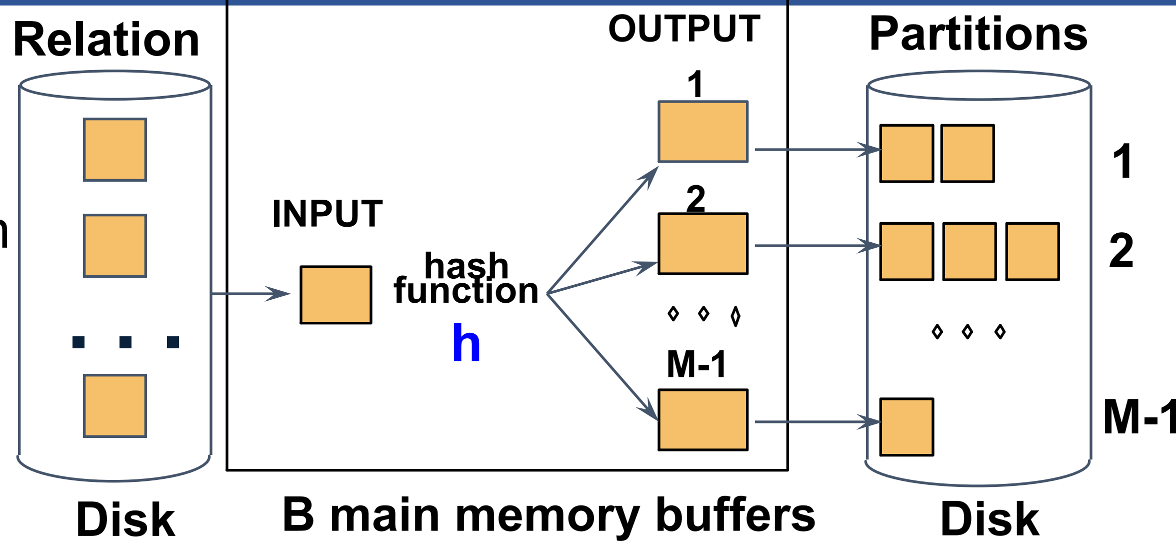
Partitioned Hash-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.

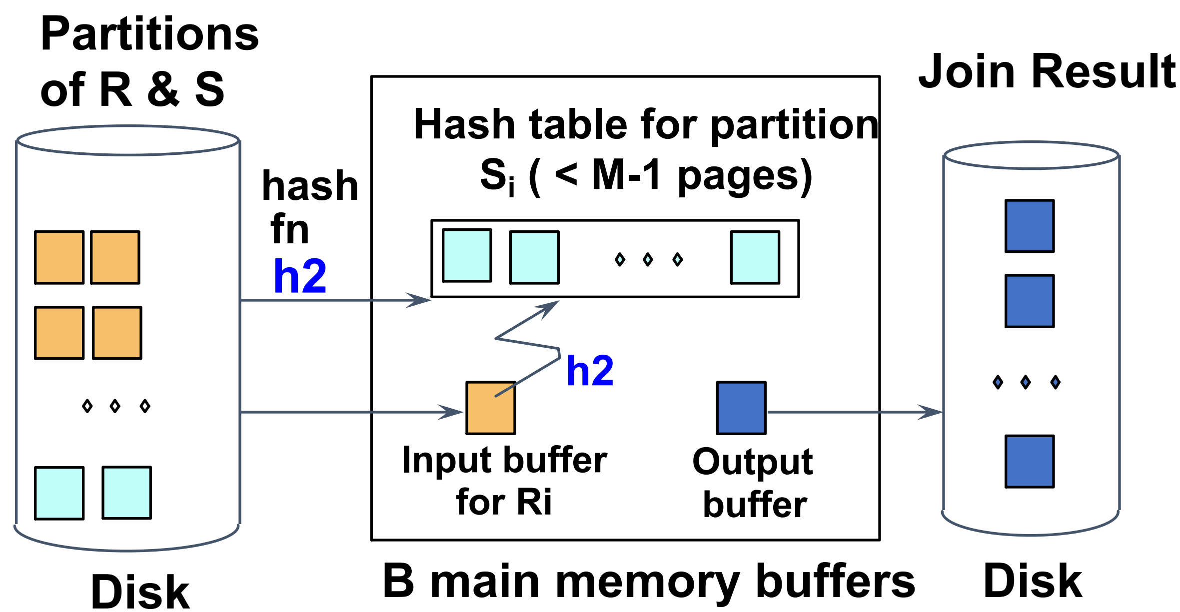


Partitioned Hash-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only match S tuples in partition i.



- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.



Partitioned Hash-Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

Hybrid Hash Join Algorithm (see book)

- Partition S into k << M buckets
  t buckets S_1 , ..., S_t stay in memory
  k-t buckets S_{t+1}, ..., S_k to disk

- Partition R into k buckets
  - First t buckets join immediately with S
  - Rest k-t buckets go to disk

- Finally, join k-t pairs of buckets:
  (R_{t+1},S_{t+1}), (R_{t+2},S_{t+2}), ..., (R_k,S_k)

Works well when B(S) > M but is not very large

Query Optimization

- Returning to query optimization…

- Three components:
  - Cost/cardinality estimation
  - Search space
  - Search algorithm

- We are discussing cardinality estimation:
  - Main idea: selectivity factor
  - Many assumptions: uniformity, independence, preservation of values, inclusion of values

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

- How large is T(Q)?
- Answer 1: T(Q) = T(Supply) (why?)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

- How large is T(Q)?
- Answer 1: T(Q) = T(Supply) (why?)
- Answer 2:

$$T(Q) = T(Supply \bowtie Supplier)$$
$$= T(Supply)*T(Supplier) / \max(V(Supply,sid),V(Supplier,sid))$$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

- How large is T(Q)?
- Answer 1: T(Q) = T(Supply) (why?)
- Answer 2:

We know this

V(Supplier,sid) = T(Supplier)
V(Supply,sid) ≤ V(Supplier,sid)

$$T(Q) = T(Supply \bowtie Supplier)$$
$$= T(Supply)*T(Supplier) / \max(V(Supply,sid),V(Supplier,sid))$$

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

- How large is T(Q)?

```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

- Answer 1: T(Q) = T(Supply) (why?)

- Answer 2:

We know this

V(Supplier,sid) = T(Supplier)
V(Supply,sid) ≤ V(Supplier,sid)

$$
\begin{aligned}
T(Q) &= T(\text{Supply} \bowtie \text{Supplier}) \\
&= T(\text{Supply})*T(\text{Supplier}) \,/\, \max(V(\text{Supply},\text{sid}),V(\text{Supplier},\text{sid})) \\
&= T(\text{Supply})*T(\text{Supplier}) \,/\, V(\text{Supplier},\text{sid}) \quad = T(\text{Supply})
\end{aligned}
$$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000

Key Foreign-key Join

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
```

- Answer 1: T(Q) = T(Supply) (why?)

- Answer 2:

We know this

V(Supplier,sid) = T(Supplier)
V(Supply,sid) ≤ V(Supplier,sid)

Containment of values assumption

T(Q) = T(Supply ⋈ Supplier)
= T(Supply)*T(Supplier) / max(V(Supply,sid),V(Supplier,sid))
= T(Supply)*T(Supplier) / V(Supplier,sid) = T(Supply)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000
V(Supplier, sstate) = 10

Preservation of Values

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
 and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000
V(Supplier, sstate) = 10

Preservation of Values

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
 and x.sstate = 'WA'
```

$T(Q) = T(\sigma_{sstate='WA'}(Supply \bowtie Supplier))$
$\quad = T(Supply \bowtie Supplier) / V(Supply \bowtie Supplier, sstate)$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000
V(Supplier, sstate) = 10

Preservation of Values

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
 and x.sstate = 'WA'
```

Preservation of values assumption

$$V(\text{Supply} \bowtie \text{Supplier}, \text{sstate}) = V(\text{Supplier}, \text{sstate}) = 10$$

$$\begin{aligned} T(Q) &= T(\sigma_{\text{sstate='WA'}}(\text{Supply} \bowtie \text{Supplier})) \\ &= T(\text{Supply} \bowtie \text{Supplier}) / V(\text{Supply} \bowtie \text{Supplier}, \text{sstate}) \end{aligned}$$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supply) = 10000
T(Supplier) = 1000
V(Supplier, sstate) = 10

Preservation of Values

- How large is T(Q)?

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
 and x.sstate = 'WA'
```

Preservation of values assumption

$V(Supply \bowtie Supplier, sstate) = V(Supplier, sstate) = 10$

$$
\begin{aligned}
T(Q) &= T(\sigma_{sstate='WA'}(Supply \bowtie Supplier)) \\
&= T(Supply \bowtie Supplier) / V(Supply \bowtie Supplier, sstate) \\
&= T(Supply \bowtie Supplier) / V(Supplier, sstate) \\
&= T(Supply) / 10
\end{aligned}
$$

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Example

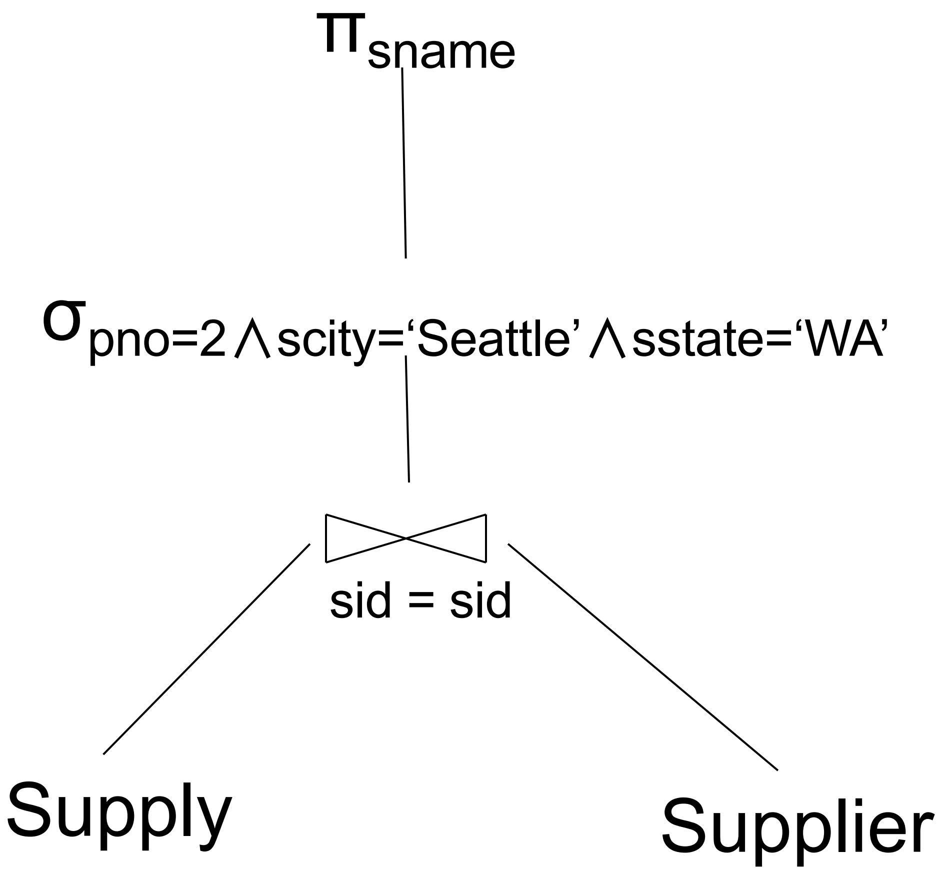
- Enumerate logical plans, estimate T(temp relations)

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

- For each logical plan, enumerate physical plans, estimate Cost

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

π_{sname}

Estimated (why?)

$\sigma_{pno=2 \wedge scity=\text{'Seattle'} \wedge sstate=\text{'WA'}}$

T = 10000

$\bowtie_{sid = sid}$

Supply

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Supplier

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 1

Estimated (why?)

π_{sname}

T < 1

$\sigma_{pno=2 \wedge scity=\text{'Seattle'} \wedge sstate=\text{'WA'}}$

T = 10000

$\bowtie_{sid = sid}$

Supply

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

Supplier

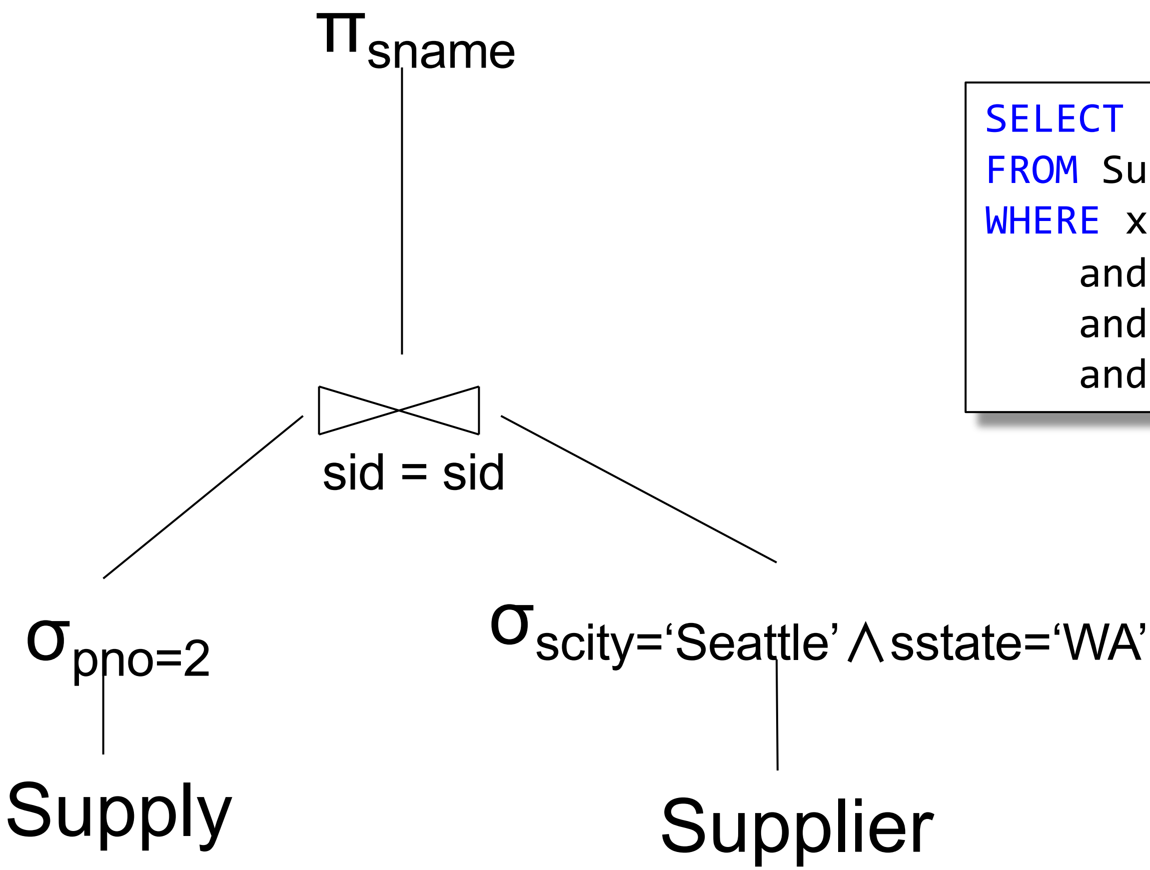
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

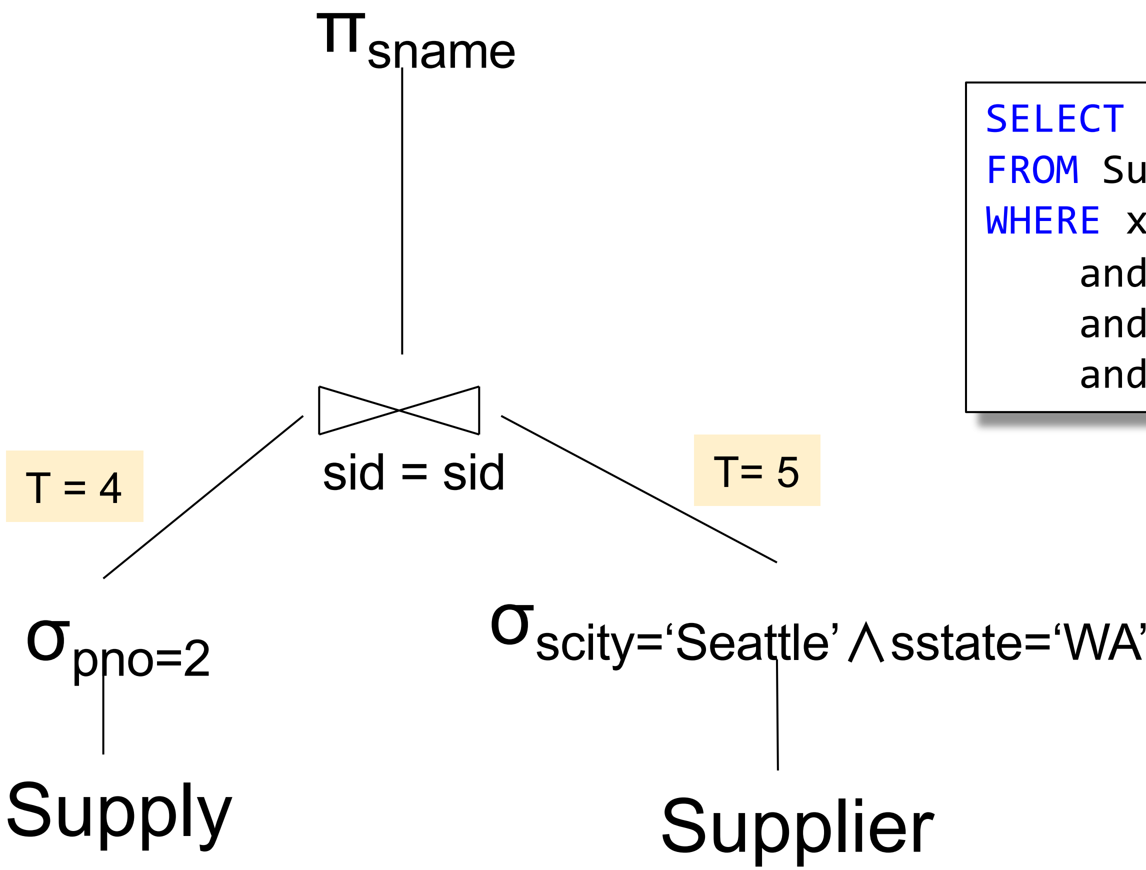
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2




```sql
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

π_{sname}

$\bowtie_{sid = sid}$

T = 4

T= 5

Very wrong!
Why?

$\sigma_{pno=2}$

$\sigma_{scity=\text{‘Seattle’} \wedge sstate=\text{‘WA’}}$

Supply

Supplier

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

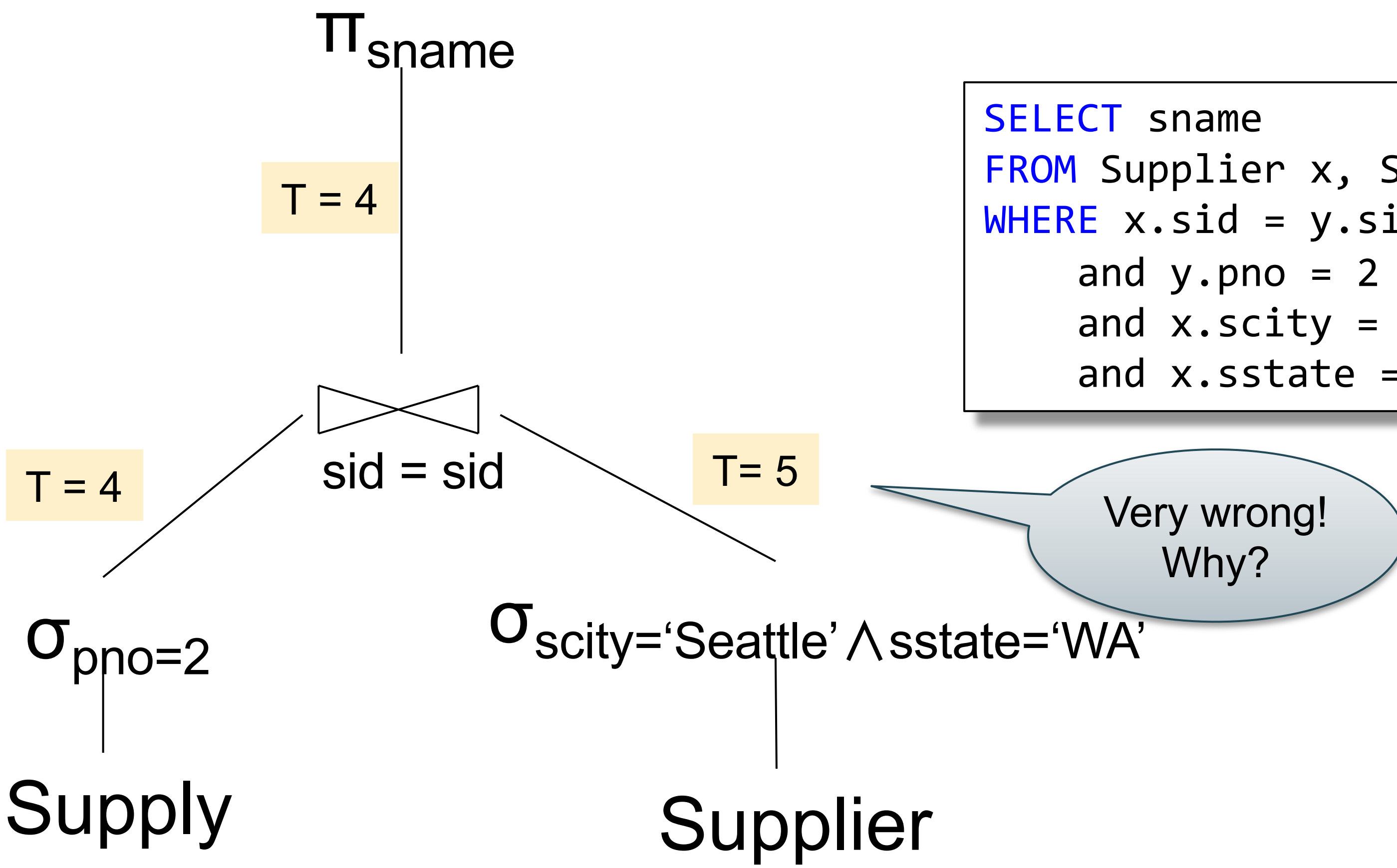
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Logical Query Plan 2

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```



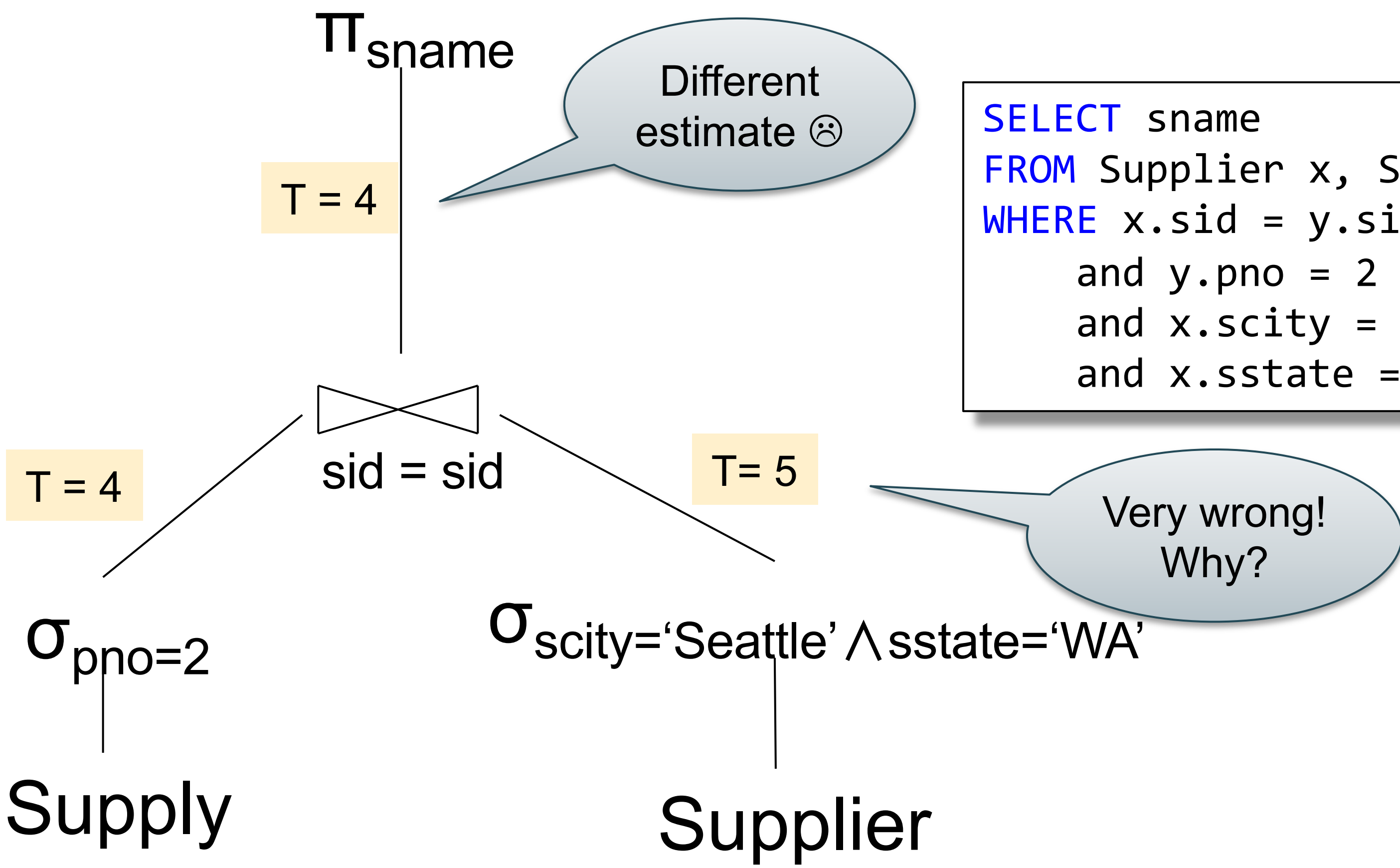
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

Logical Query Plan 2




```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1

π_{sname}

T < 1

$\sigma_{pno=2 \wedge scity=\text{'Seattle'} \wedge sstate=\text{'WA'}}$

T = 10000

$\bowtie$ sid = sid

Block nested loop join

Scan Supply

Scan Supplier

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

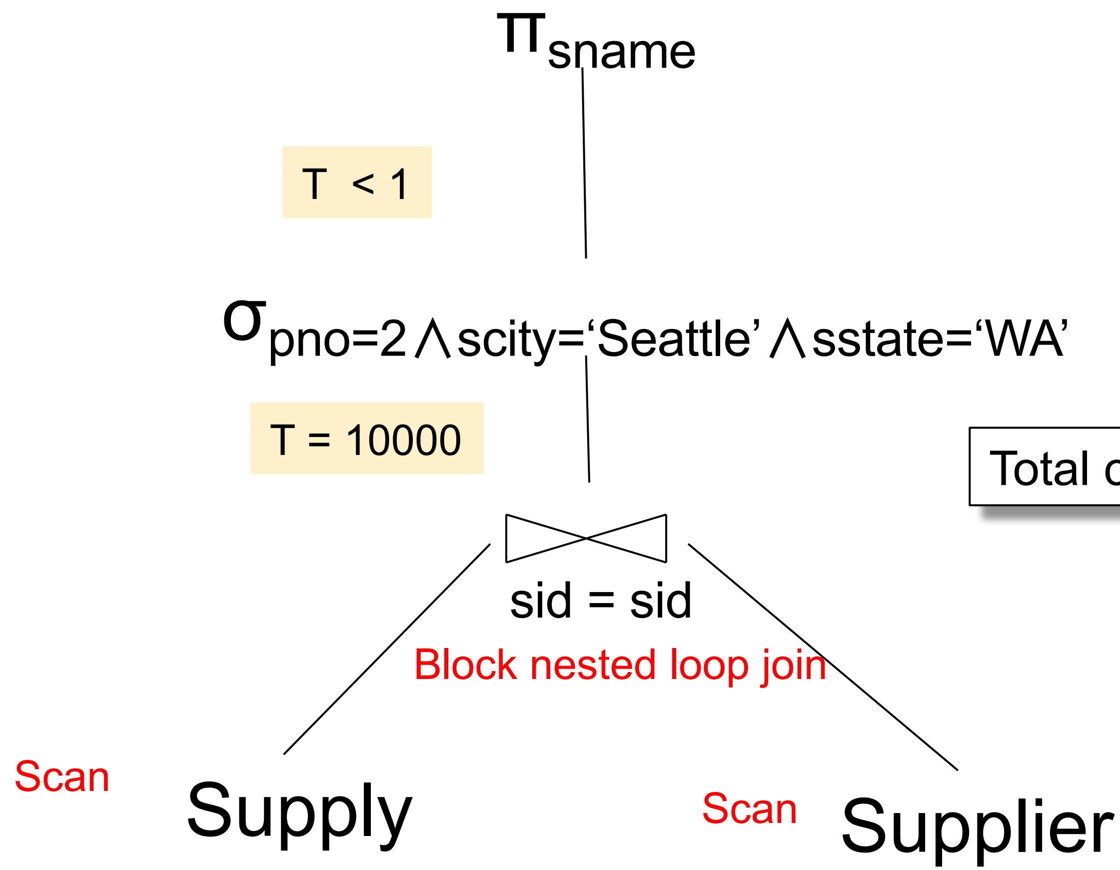
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

Total cost:

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 1



Total cost: 100+100*100/10 = 1100

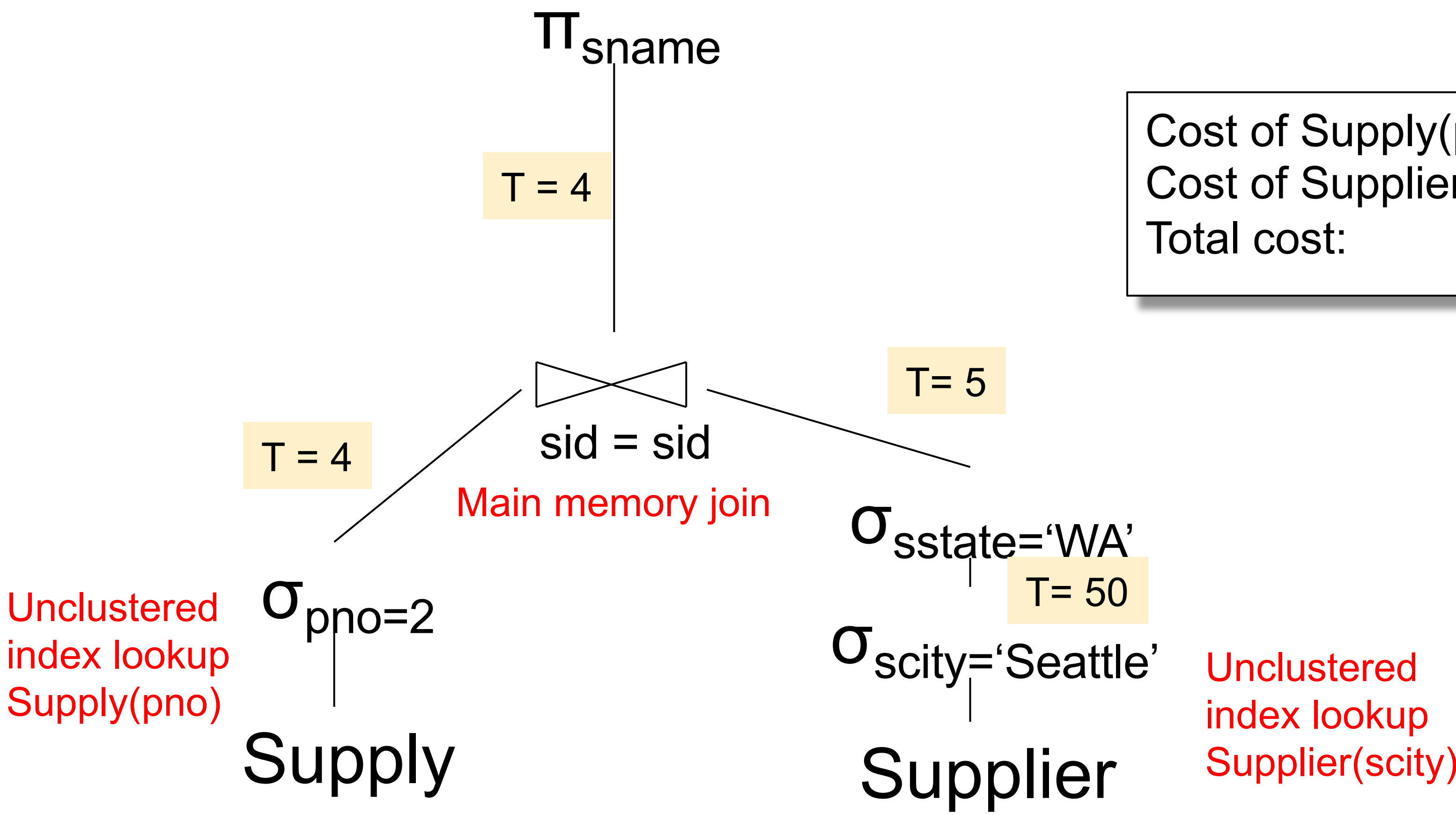
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2



Cost of Supply(pno) =
Cost of Supplier(scity) =
Total cost:

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

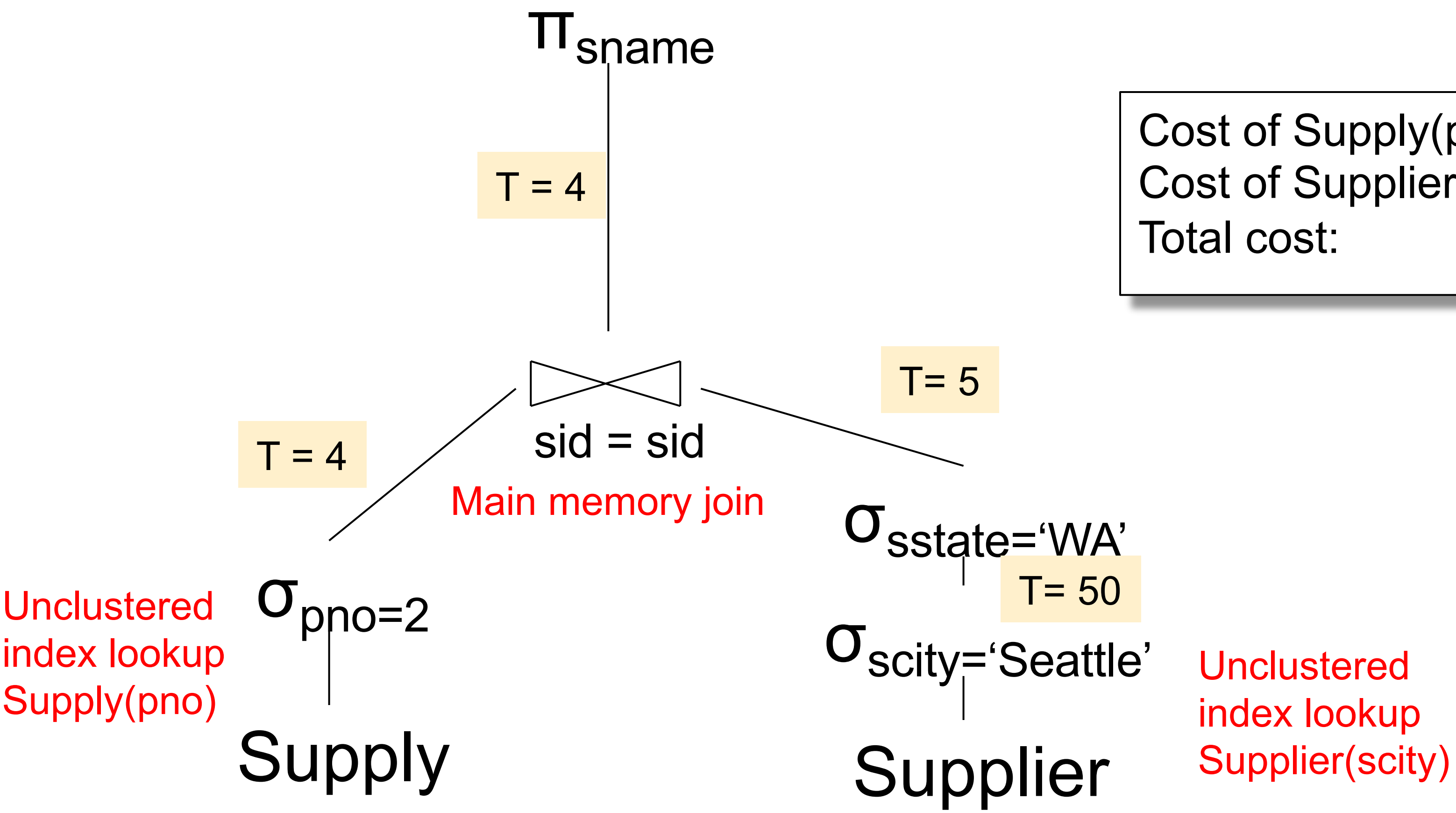
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2

Cost of Supply(pno) = 4
Cost of Supplier(scity) =
Total cost:



T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

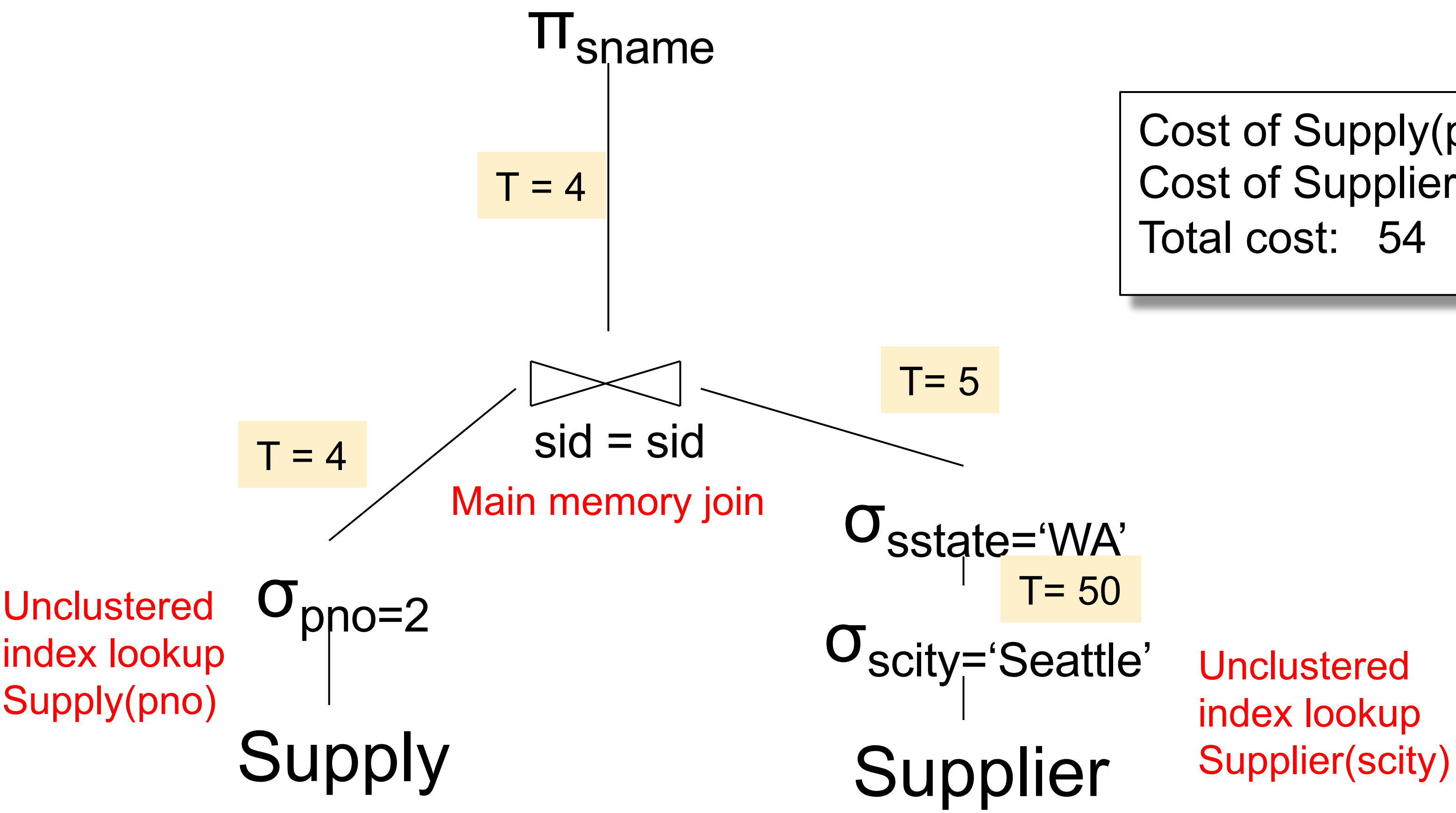
T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 2

Cost of Supply(pno) = 4
Cost of Supplier(scity) = 50
Total cost: 54



T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3

π_{sname}

T = 4

$\sigma_{scity=\text{'Seattle'} \land sstate=\text{'WA'}}$

$\bowtie_{sid = sid}$

Clustered
Index join

T = 4

Unclustered
index lookup
Supply(pno)

$\sigma_{pno=2}$

Supply

Supplier

Cost of Supply(pno) =
Cost of Index join =
Total cost:

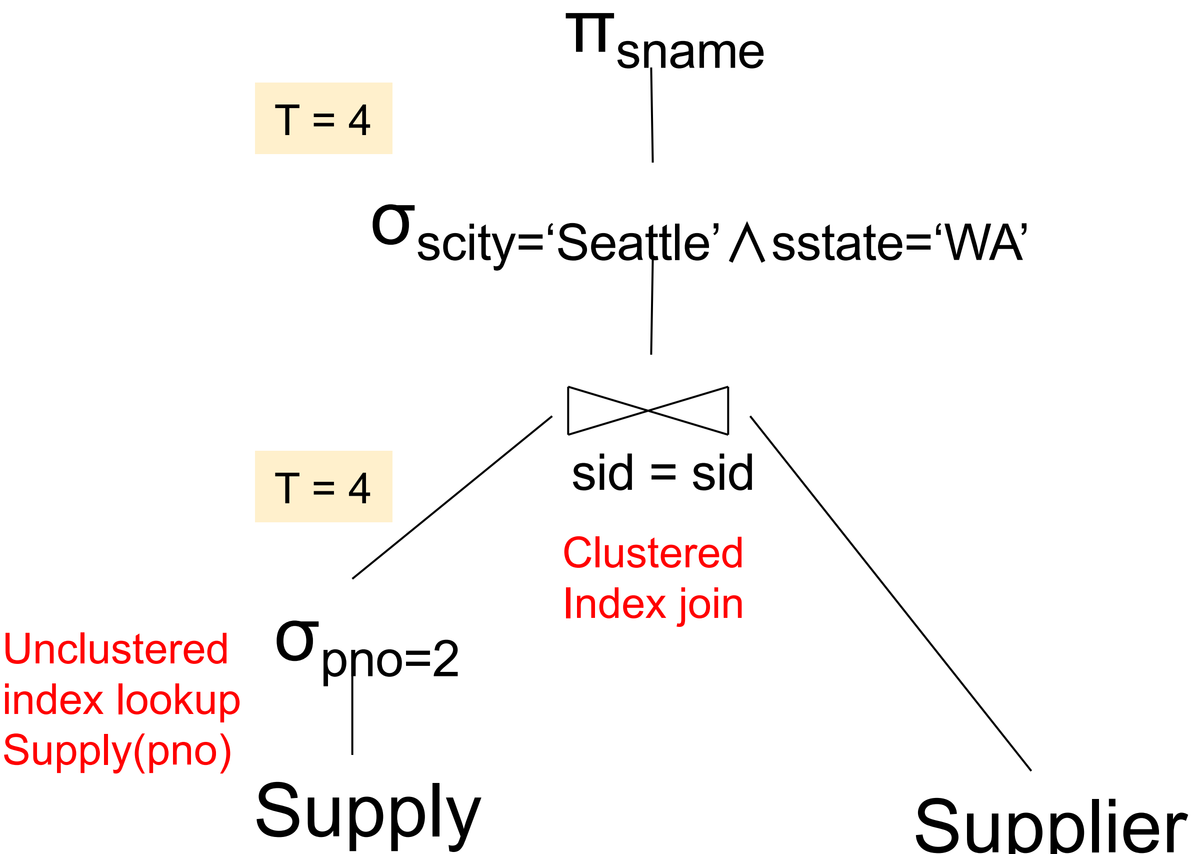
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) = 4
Cost of Index join =
Total cost:

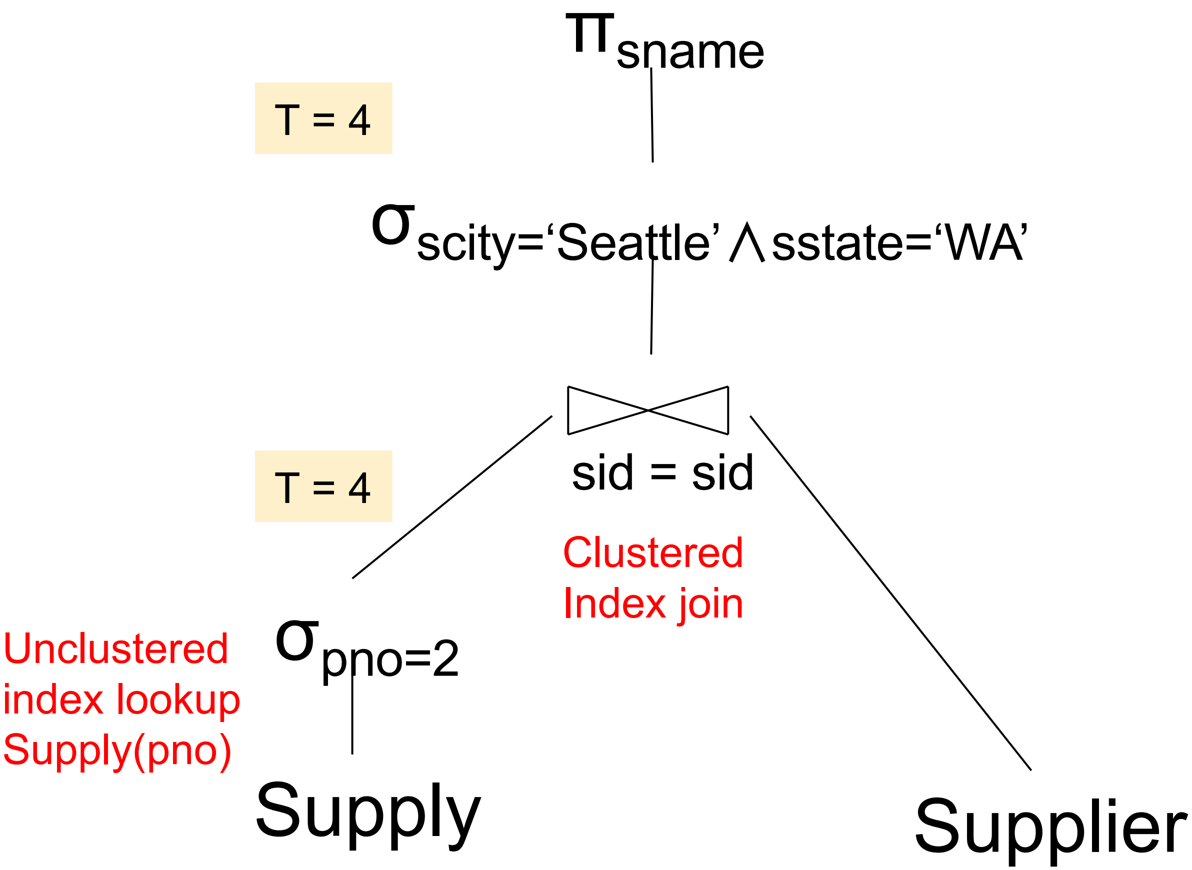
T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Physical Plan 3



Cost of Supply(pno) = 4
Cost of Index join = 4
Total cost: 8

T(Supply) = 10000
B(Supply) = 100
V(Supply, pno) = 2500

T(Supplier) = 1000
B(Supplier) = 100
V(Supplier, scity) = 20
V(Supplier, state) = 10

M=11

Histograms

- Relax uniformity assumption:
  $T(\sigma_{A=v}(R)) = T(R) / V(R,A)$

- Histogram:
  - Partition R into buckets by the values of A
  - For each bucket, store T(bucket) and other stats

- RDBMS maintain histograms on some attributes of some tables; recomputed periodically using sampling

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}(\text{Empolyee}) = ?$ $\sigma_{age>28 \text{ and } age<35}(\text{Empolyee}) = ?$

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}(\text{Empolyee}) = ?$ $\sigma_{age>28 \text{ and } age<35}(\text{Empolyee}) = ?$

Estimate = 25000 / 50 = 500

Estimate = 25000 * 6 / 50 = 3000

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

Age:	0-20	20-29	30-39	40-49	50-59	> 60
Tuples	200	800	5000	12000	6500	500

Histograms

Employee(ssn, name, age)

T(Employee) = 25000, V(Empolyee, age) = 50
min(age) = 19, max(age) = 68

$\sigma_{age=48}$(Empolyee) = ? $\sigma_{age>28 \text{ and } age<35}$(Empolyee) = ?

Age:	0..20	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5000	12000	6500	500

Estimate = 1200

Estimate = 1*80 + 5*500 = 2580

Types of Histograms

- How should we determine the bucket boundaries in a histogram?

Types of Histograms

- How should we determine the bucket boundaries in a histogram?

- Eq-Width
- Eq-Depth
- Compressed
- V-Optimal histograms

Histograms

Employee(ssn, name, age)

Eq-width:

Age:	0..20	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5000	12000	6500	500

Eq-depth:

Age:	0-33	33-38	38-43	43-45	45-54	> 54
Tuples	1800	2000	2100	2200	1900	1800

Compressed: store separately highly frequent values: (48,1900)

V-Optimal Histograms

- Defines bucket boundaries in an optimal way, to minimize the error over all point queries
- Computed rather expensively, using dynamic programming
- Modern databases systems use V-optimal histograms or some variations

Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY?

- *Not* updated during database update, but recomputed periodically
  - WHY?

- Multidimensional histograms rarely used
  - WHY?

Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY? All histograms are kept in main memory during query optimization; plus need fast access
- *Not* updated during database update, but recomputed periodically
  - WHY?
- Multidimensional histograms rarely used
  - WHY?

Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY? All histograms are kept in main memory during query optimization; plus need fast access
- *Not* updated during database update, but recomputed periodically
  - WHY? Histogram update creates a write conflict; would dramatically slow down transaction throughput
- Multidimensional histograms rarely used
  - WHY?

Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY? All histograms are kept in main memory during query optimization; plus need fast access

- *Not* updated during database update, but recomputed periodically
  - WHY? Histogram update creates a write conflict; would dramatically slow down transaction throughput

- Multidimensional histograms rarely used
  - WHY? Too many possible multidimensional histograms, unclear which ones to choose