

Example $R1 \bowtie R2$ over common attribute C

$T(R1) = 10,000$ (Other notation: N_{R1})

$T(R2) = 5,000$

$L(R1) = L(R2) = 1/10$ block ($bf_{R1} = 10$)

Memory available = 101 blocks

→ Metric: # of IOs

(ignoring writing of result)

Options

- Transformations: $R1 \bowtie R2$, $R2 \bowtie R1$
- Join algorithms:
 - Iteration (nested loops)
 - Merge join
 - Join with index
 - Hash join

Example 1(a) Iteration Join $R1 \bowtie R2$

- Relations not contiguous (1 row/block)
- Recall $\left\{ \begin{array}{l} T(R1) = 10,000 \quad T(R2) = 5,000 \\ L(R1) = L(R2) = 1/10 \text{ block} \\ MEM = 101 \text{ blocks} \end{array} \right.$

Cost: for each R1 tuple:

[Read tuple + Read R2]

Total = $10,000 * [1 + 5000] = 50,010,000$ Ios

-> $T(R1) * B(R2) + B(R1)$ ($B(R1) = T(R1)$ and $B(R2) = T(R2)$ now)

- Can we do better?

Use our memory

- (1) Read 100 blocks of R1 (M-1 blocks)
- (2) Read all of R2 (using 1 block) + join
- (3) Repeat until done

Cost: for each R1 chunk:

Read chunk: 100 IOs

Read R2: 5000 IOs
5100

-> $B(R1)/(M-1)*B(R2)+B(R1)$

$$\text{Total} = \frac{10,000}{100} \times 5100 = 510,000 \text{ IOs}$$

- Can we do better?

☛ Reverse join order: $R2 \bowtie R1$

$$\text{Total} = \frac{5000}{100} \times (100 + 10,000) =$$

$$50 \times 10,100 = 505,000 \text{ IOs}$$

$$\rightarrow B(R2)/(M-1)*B(R1)+B(R2)$$

Example 1(b) Iteration Join $R2 \bowtie R1$

- Relations **contiguous** (10 rows/block)

Cost

For each R2 chunk:

Read chunk: 100 IOs

Read R1: 1000 IOs

1100

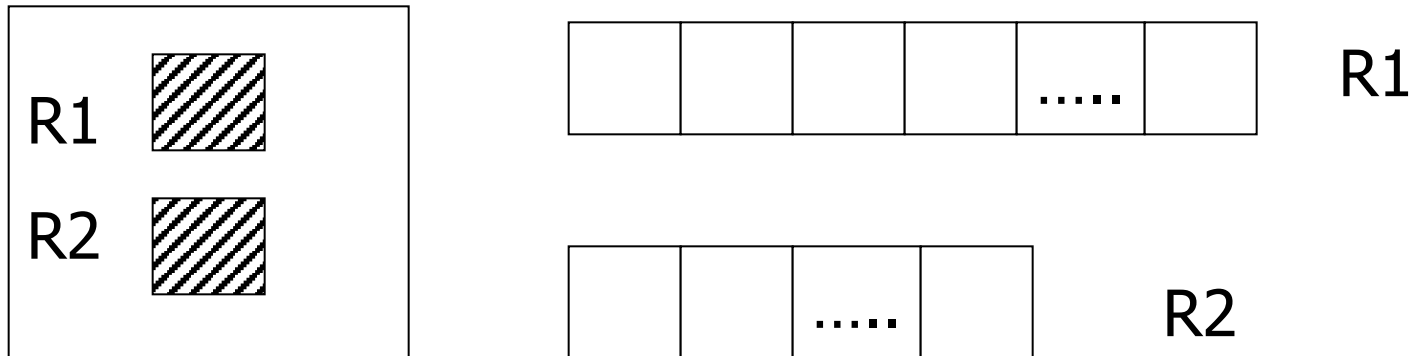
Total= 5 chunks x 1100 = **5500** IOs

-> $B(R2)/(M-1)*B(R1)+B(R2)$

Example 1(c) Merge Join

- Both **R1, R2 ordered** by C; relations contiguous

Memory



Total cost: Read R1 cost + read R2 cost
= 1000 + 500 = **1,500** IOs

Example 1(d) Merge Join

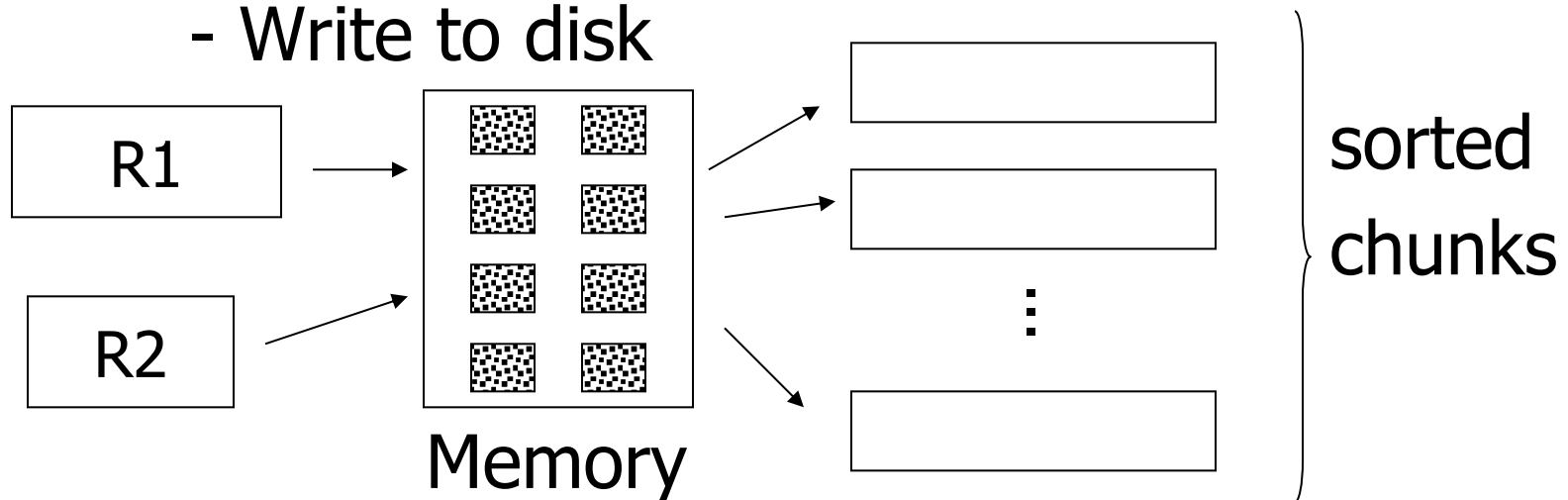
- R1, R2 not ordered, but contiguous

--> Need to sort R1, R2 first.... HOW?

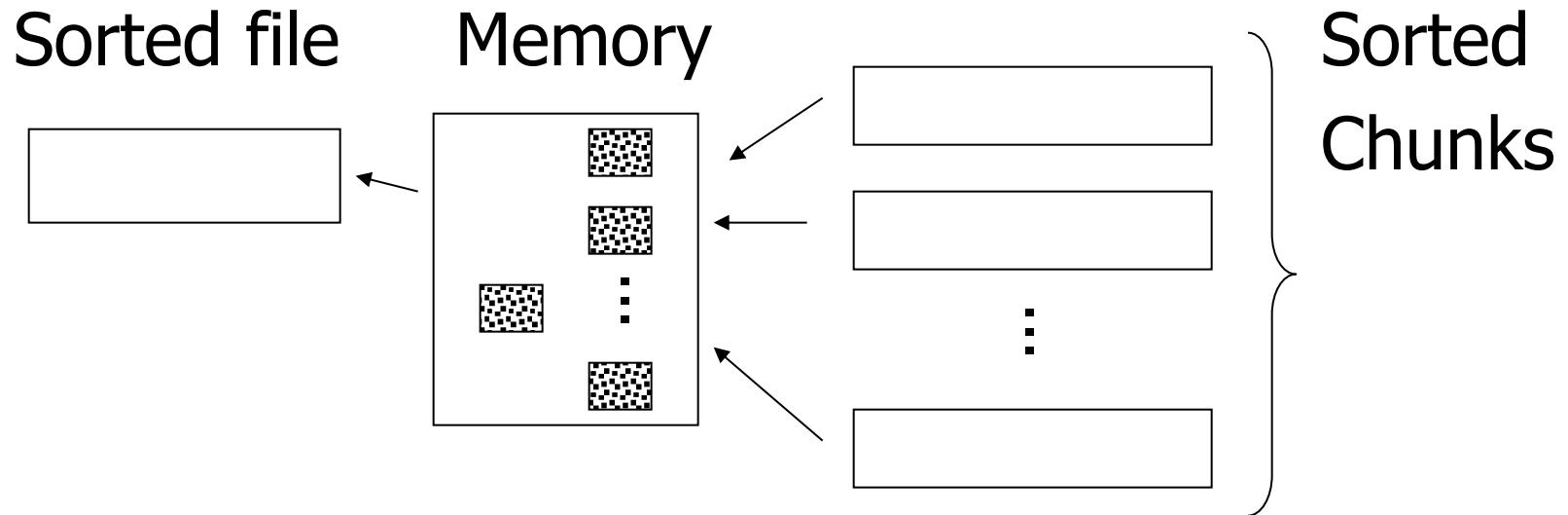
One way to sort: Merge Sort

(i) For each **100 blk chunk** of R:

- Read chunk
- Sort in memory
- Write to disk



(ii) Read all chunks + merge + write out



Cost: Sort

Each tuple is read, written,
read, written

SO...

Sort cost R1: $4 \times 1,000 = 4,000$

Sort cost R2: $4 \times 500 = 2,000$

Example 1(d) Merge Join (continued)

R1,R2 contiguous, but unordered

$$\begin{aligned}\text{Total cost} &= \text{sort cost} + \text{join cost} \\ &= 6,000 + 1,500 = 7,500 \text{ IOs}\end{aligned}$$

But: Iteration cost = 5,500
so merge join does not pay off!

But say R1 = 10,000 blocks contiguous
 R2 = 5,000 blocks not ordered

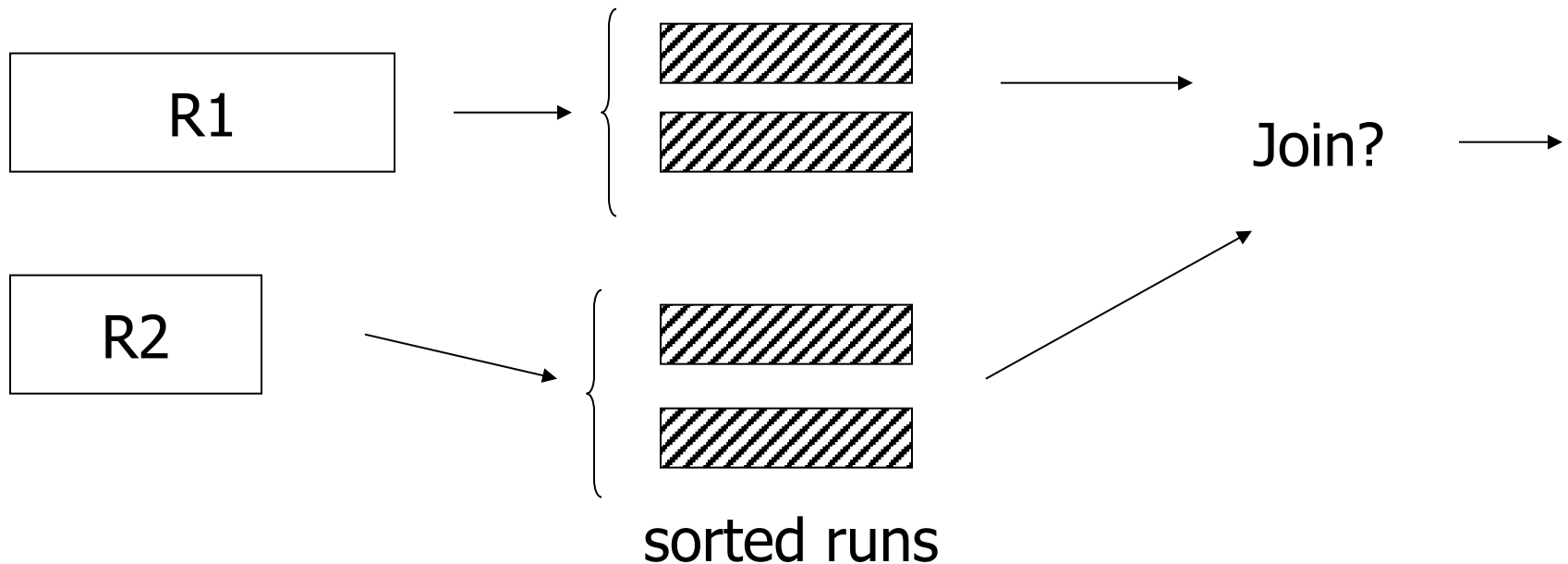
Iterate: $\frac{5000}{100} \times (100 + 10,000) = 50 \times 10,100$
 $= 505,000$ IOs

Merge join: $5(10,000 + 5,000) = 75,000$ IOs

Merge Join (with sort) WINS!

Can we **improve on merge join**?

Hint: do we really need the fully sorted files?



Cost of improved merge join:

$$\begin{aligned} C &= \text{Read } R1 + \text{write } R1 \text{ into runs} \\ &+ \text{read } R2 + \text{write } R2 \text{ into runs} \\ &+ \text{join} \\ &= 2000 + 1000 + 1500 = 4500 \end{aligned}$$

Example 1(e) Index Join

- Assume R1.C index exists; 2 levels
- Assume R2 contiguous, unordered
- Assume R1.C index fits in memory

Cost: Reads: 500 IOs

for each R2 tuple:

- probe index - free
- **if match**, read R1 tuple: **1 IO**

What is expected # of matching tuples?

(a) say R1.C is key, R2.C is foreign key
then expect = 1

(b) say $V(R1,C) = 5000$, $T(R1) = 10,000$
with uniform assumption
expect = $10,000/5,000 = 2$

What is expected # of matching tuples?

(c) Say $\text{DOM}(R1, C) = 1,000,000$

$T(R1) = 10,000$

with alternate assumption

$$\text{Expect} = \frac{10,000}{1,000,000} = \frac{1}{100}$$

Total cost with index join

(a) Total cost = $500 + 5000 * (1) * 1 = 5,500$

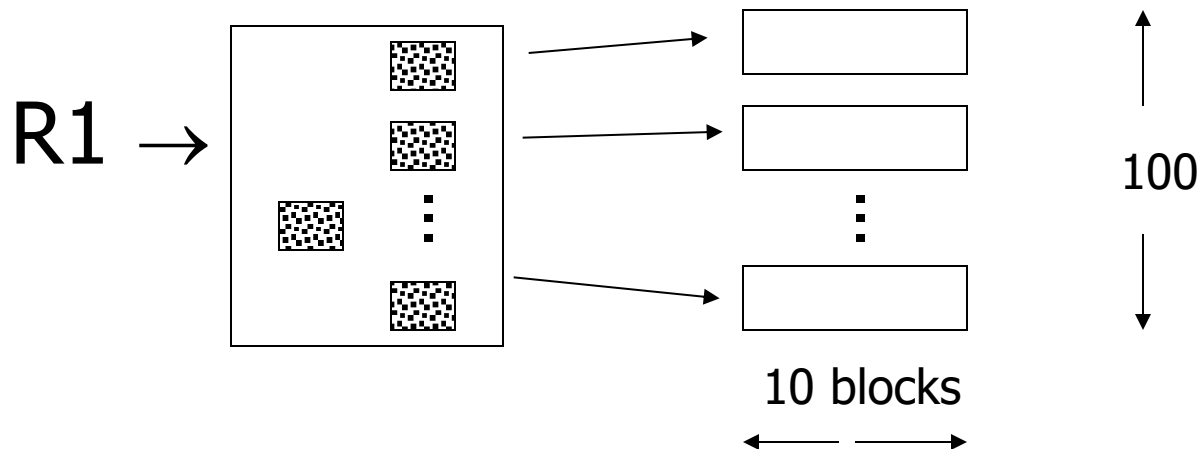
(b) Total cost = $500 + 5000 * (2) * 1 = 10,500$

(c) Total cost = $500 + 5000 * (1/100) * 1 = 550$

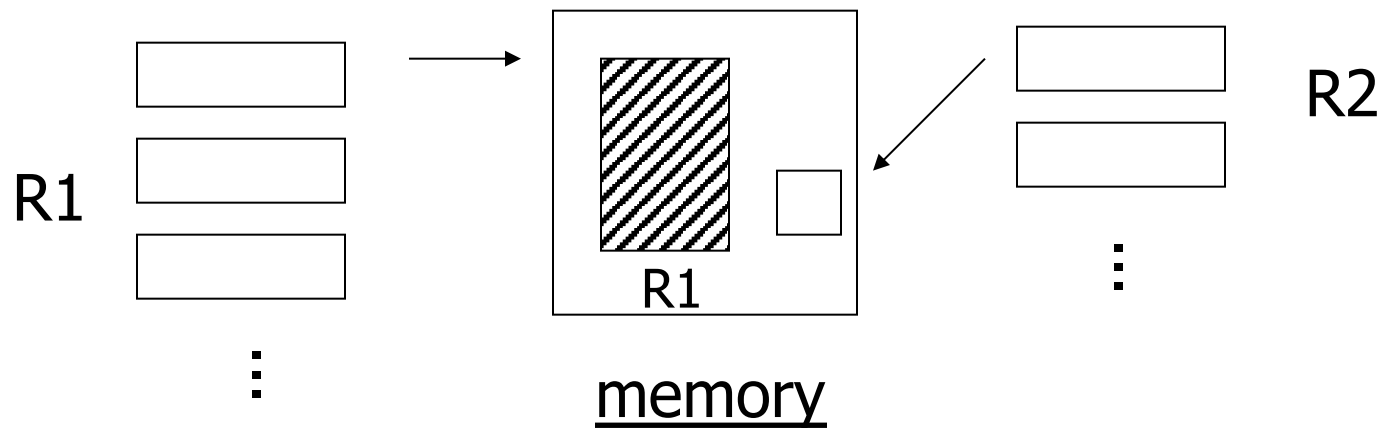
-> $B(R2) + T(R2) * T(R1) / V(R1, C)$

Example 1(f) Hash Join

- R1, R2 contiguous (un-ordered)
- Use 100 buckets
- Read R1, hash, + write buckets



- > Same for R2
- > Read one R1 bucket; build memory hash table
- > Read corresponding R2 bucket + hash probe



Then repeat for all buckets

Cost:

"Bucketize:" Read R1 + write

Read R2 + write

Join: Read R1, R2

Total cost = $3 \times [1000 + 500] = 4500$

-> $3 \times [B(R1) + B(R2)]$

Note: this is an approximation since buckets will vary in size and we have to round up to blocks

A hash join trick:

- Only write into buckets
 <val,ptr> pairs
- When we get a match in join phase,
 must fetch tuples

- To illustrate cost computation, assume:
 - 100 $\langle \text{val}, \text{ptr} \rangle$ pairs/block
 - expected number of result tuples is 100
- Build hash table for R2 **in memory** (no outp)
 5000 tuples $\rightarrow 5000/100 = 50$ blocks
- Read R1 and match
- Read ~ 100 R2 tuples

<u>Total cost</u> =	Read R2:	500
	Read R1:	1000
	Get tuples:	<u>100</u>
		1600

Summary

- Iteration ok for “small” relations
(relative to memory size)
- For equi-join, where relations not
sorted and no indexes exist,
hash join usually best

- Sort + merge join good for non-equi-join (e.g., $R1.C > R2.C$)
- If relations already sorted, use merge join
- If index exists, it could be useful (depends on expected result size)