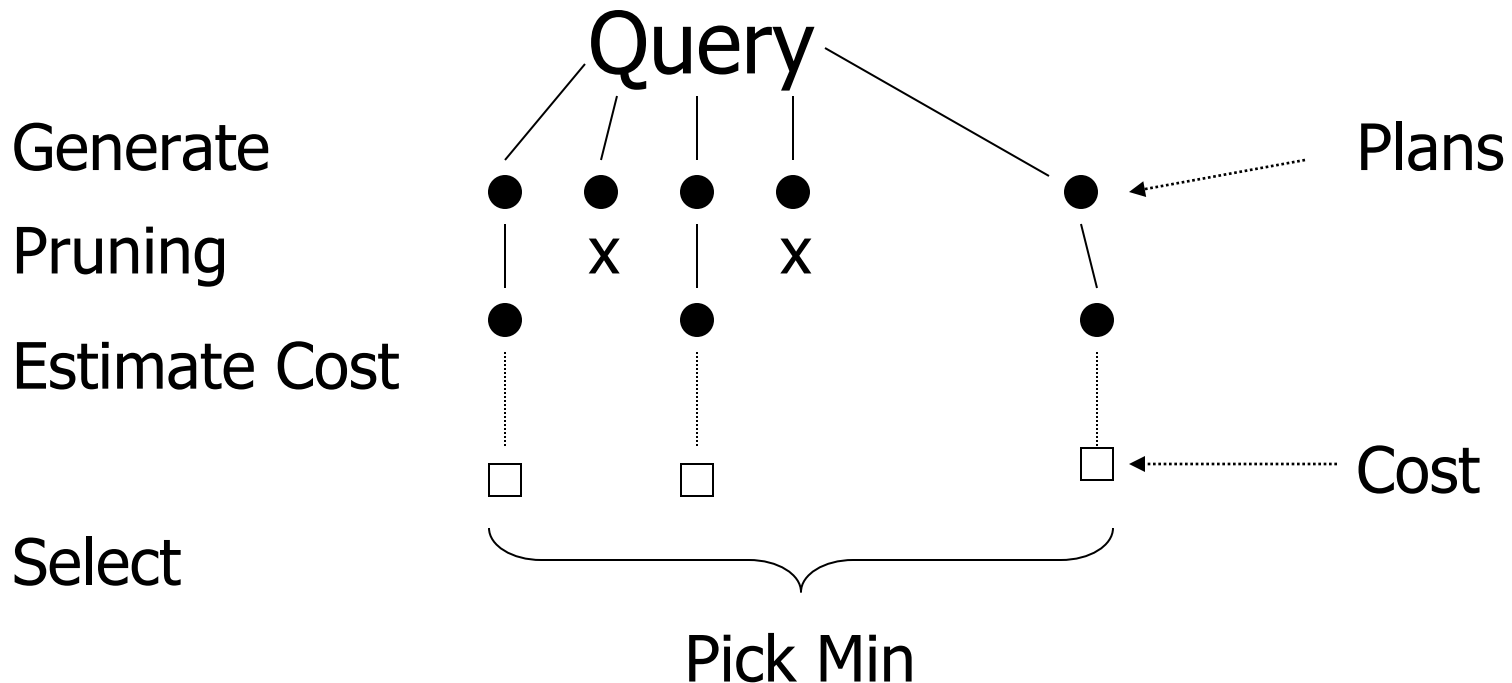


Query Optimization – Physical Query Plans

Hector Garcia-Molina and
Mahmoud Sakr

Query Optimization

--> Generating and comparing plans



To generate plans consider:

- Transforming relational algebra expression
(e.g. order of joins)
- Use of existing indexes
- Building indexes or sorting on the fly

- Implementation details:
 - e.g. - Join algorithm
 - Memory management
 - Parallel processing

Estimating IOs:

- Count # of disk blocks that must be read (or written) to execute query plan

To estimate costs, we may have additional parameters:

$B(R)$ = # of blocks containing R tuples

$f(R)$ = max # of tuples of R per block

M = # memory blocks available

To estimate costs, we may have additional parameters:

$B(R)$ = # of blocks containing R tuples

$f(R)$ = max # of tuples of R per block

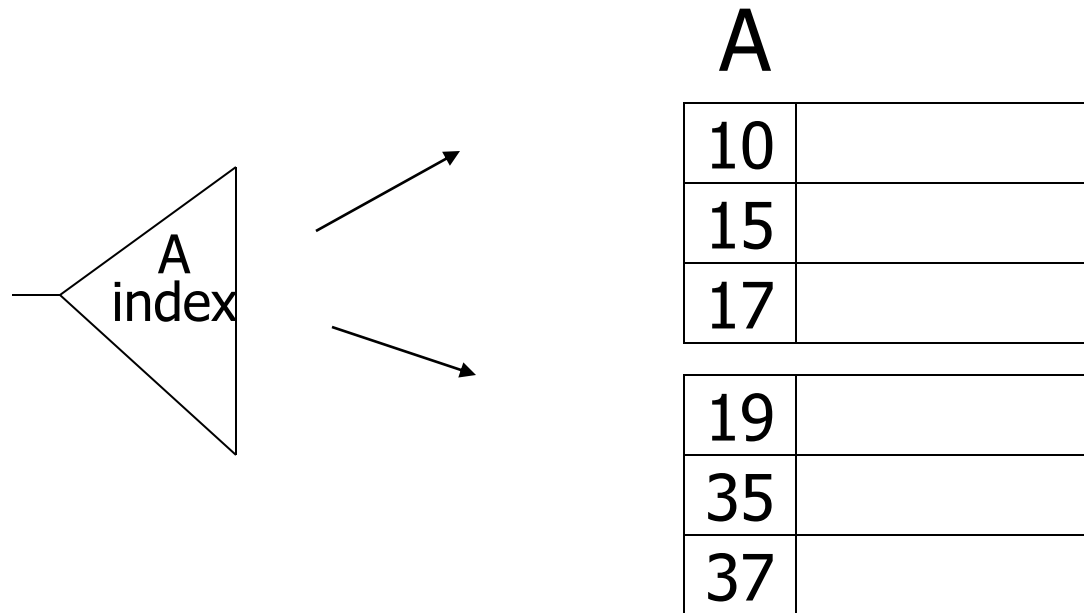
M = # memory blocks available

$HT(i)$ = # levels in index i

$LB(i)$ = # of leaf blocks in index i

Clustering index

Index that allows tuples to be read in an order that corresponds to physical order



Notions of clustering

- Clustered file organization

R1 R2 S1 S2

R3 R4 S3 S4

- Clustered relation

R1 R2 R3 R4

R5 R5 R7 R8

- Clustering index

Example $R1 \bowtie R2$ over common attribute C

$$T(R1) = 10,000$$

$$T(R2) = 5,000$$

$$S(R1) = S(R2) = 1/10 \text{ block}$$

$$\text{Memory available} = 101 \text{ blocks}$$

Example $R1 \bowtie R2$ over common attribute C

$$T(R1) = 10,000$$

$$T(R2) = 5,000$$

$$S(R1) = S(R2) = 1/10 \text{ block}$$

$$\text{Memory available} = 101 \text{ blocks}$$

→ Metric: # of IOs
(ignoring writing of result)

Options

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

- Iteration join (conceptually)
 for each $r \in R1$ do
 for each $s \in R2$ do
 if $r.C = s.C$ then output r,s pair

- Merge join (conceptually)

(1) if R1 and R2 not sorted, sort them

(2) $i \leftarrow 1; j \leftarrow 1;$

While $(i \leq T(R1)) \wedge (j \leq T(R2))$ do

if $R1\{i\}.C = R2\{j\}.C$ then outputTuples

else if $R1\{i\}.C > R2\{j\}.C$ then $j \leftarrow j+1$

else if $R1\{i\}.C < R2\{j\}.C$ then $i \leftarrow i+1$

Procedure Output-Tuples

While $(R1\{ i \}.C = R2\{ j \}.C) \wedge (i \leq T(R1))$ do

$[jj \leftarrow j;$

 while $(R1\{ i \}.C = R2\{ jj \}.C) \wedge (jj \leq T(R2))$ do

 [output pair $R1\{ i \}, R2\{ jj \}$;

$jj \leftarrow jj+1$]

$i \leftarrow i+1$]

Example

i	$R1\{i\}.C$	$R2\{j\}.C$	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

- Join with index (Conceptually)

For each $r \in R1$ do

Assume $R2.C$ index

[$X \leftarrow \text{index}(R2, C, r.C)$

for each $s \in X$ do

output r,s pair]

Note: $X \leftarrow \text{index}(\text{rel}, \text{attr}, \text{value})$

then $X = \text{set of rel tuples with attr} = \text{value}$

- Hash join (conceptual)
 - Hash function h , range $0 \rightarrow k$
 - Buckets for R_1 : $G_0, G_1, \dots G_k$
 - Buckets for R_2 : $H_0, H_1, \dots H_k$

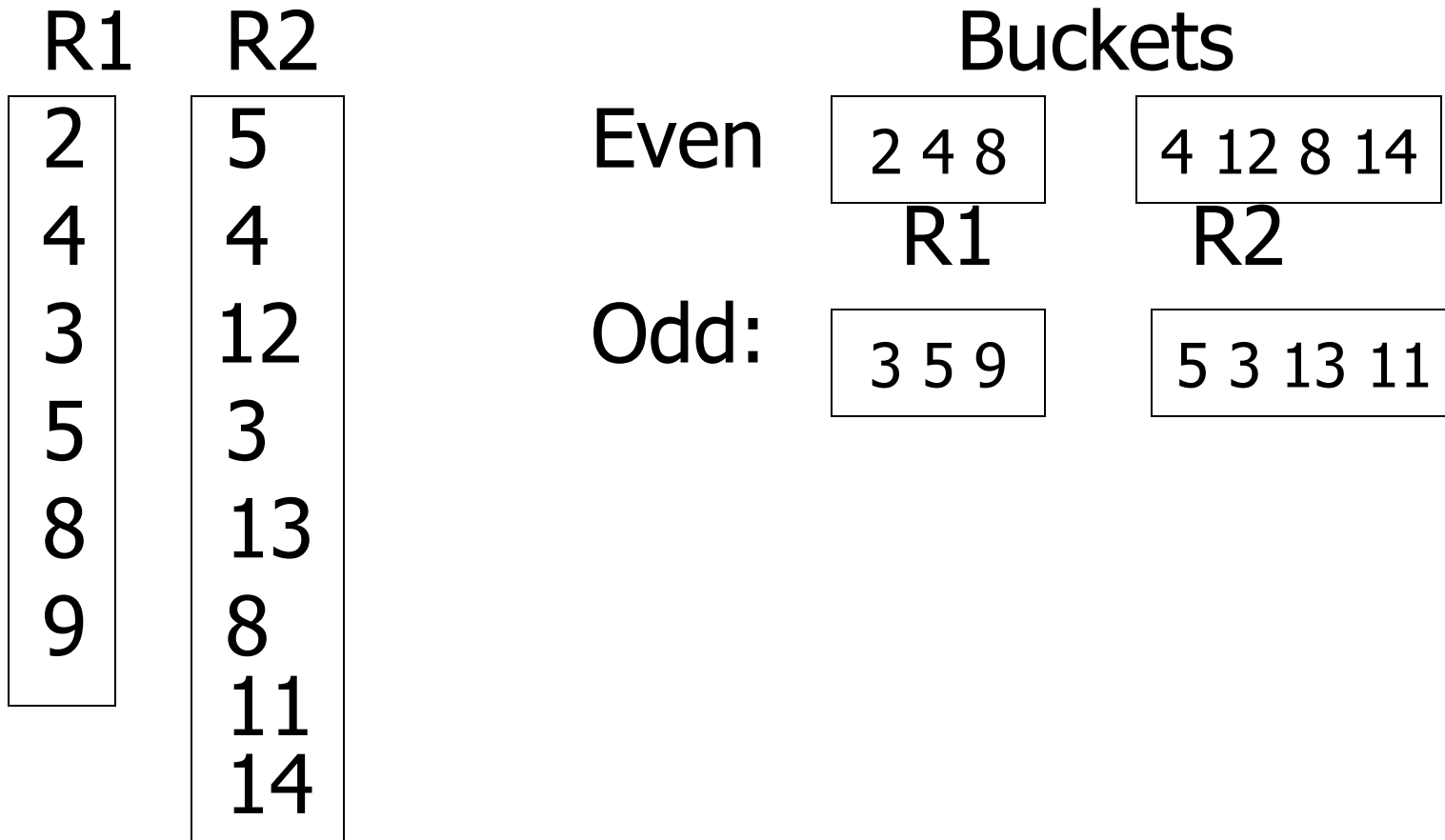
- Hash join (conceptual)
 - Hash function h , range $0 \rightarrow k$
 - Buckets for $R1$: $G_0, G_1, \dots G_k$
 - Buckets for $R2$: $H_0, H_1, \dots H_k$

Algorithm

- (1) Hash $R1$ tuples into G buckets
- (2) Hash $R2$ tuples into H buckets
- (3) For $i = 0$ to k do
 match tuples in G_i, H_i buckets

Simple example

hash: even/odd



Factors that affect performance

- (1) Tuples of relation stored physically together?
- (2) Relations sorted by join attribute?
- (3) Indexes exist?

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

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

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

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

Cost: for each R1 tuple:

[Read tuple + Read R2]

Total = 10,000 [1 + 5000] = 50,010,000 IOs

- Can we do better?

- Can we do better?

Use our memory

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

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2: 5000 IOs
6000

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2: $\frac{5000}{1} \text{ IOs}$
6000

$$\text{Total} = \frac{10,000}{1,000} \times 6000 = 60,000 \text{ IOs}$$

- Can we do better?

- Can we do better?

➡ Reverse join order: $R2 \bowtie R1$

$$\text{Total} = \frac{5000}{1000} \times (1000 + 10,000) =$$

$$5 \times 11,000 = 55,000 \text{ IOs}$$

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

- Relations contiguous

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

- Relations contiguous

Cost

For each R2 chunk:

Read chunk: 100 IOs

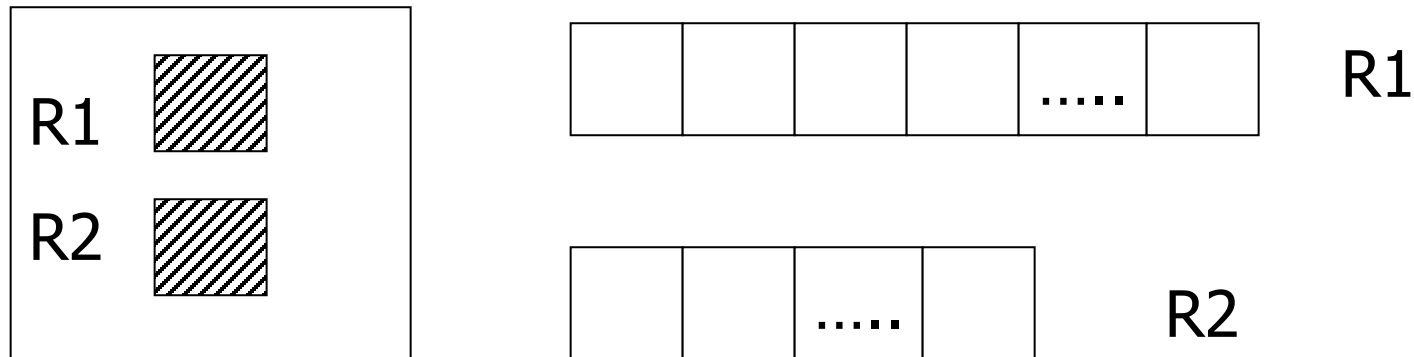
Read R1: $\frac{1000}{1} \text{ IOs}$
1,100

Total= 5 chunks x 1,100 = 5,500 IOs

Example 1(c) Merge Join

- Both R1, R2 ordered by C; relations contiguous

Memory



record kati choti read huncha herne

esma both relation ko records ek ek choti matra read huncha so cost=1000+500

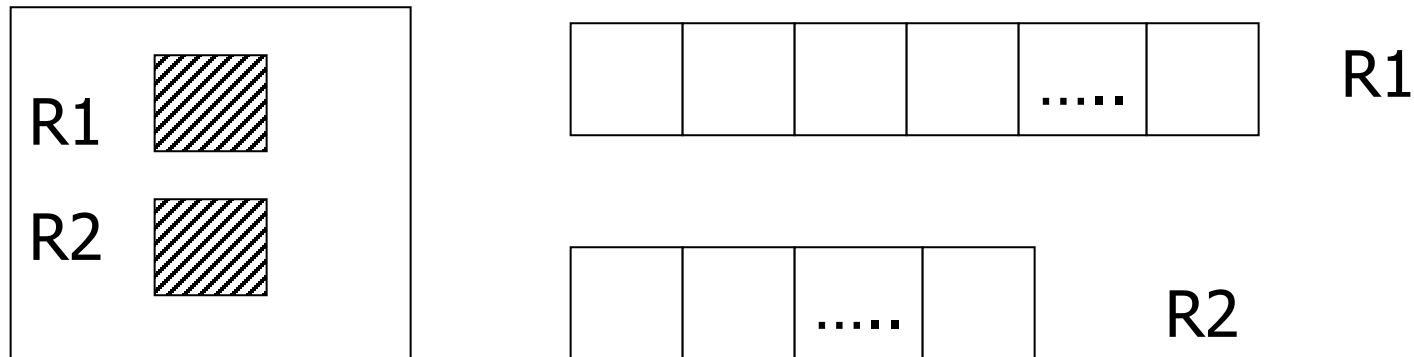
Is it using memory or not?

If it using memory, what would be result without using memory? What would have been result is there was less or more memory

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

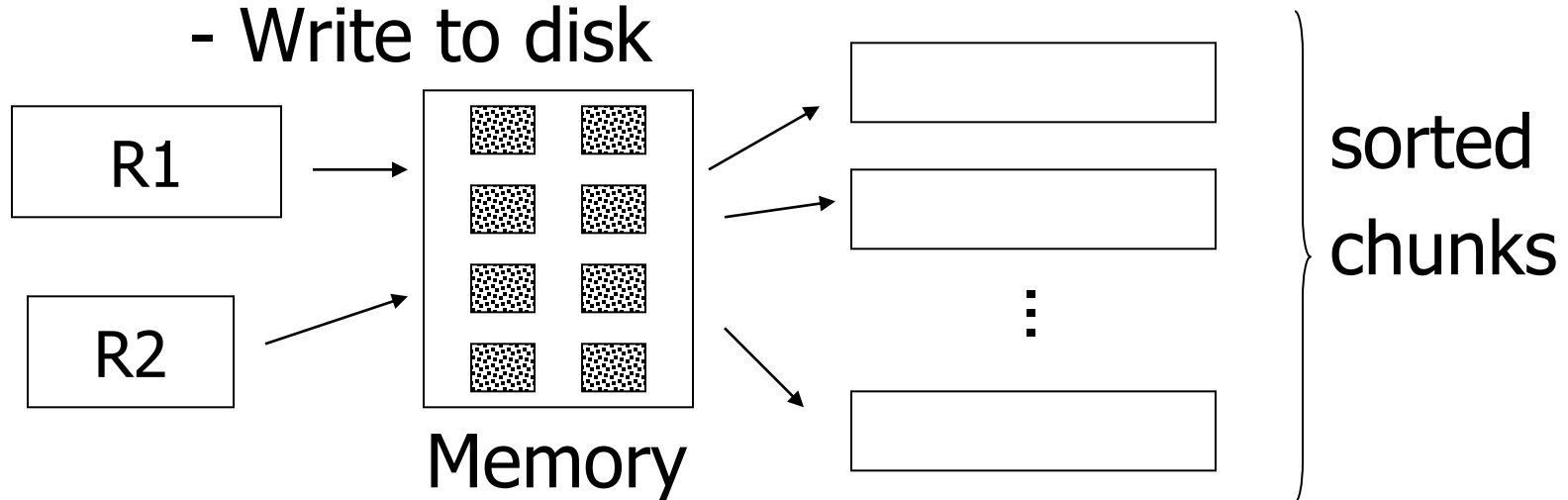
- R1, R2 not ordered, but contiguous

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

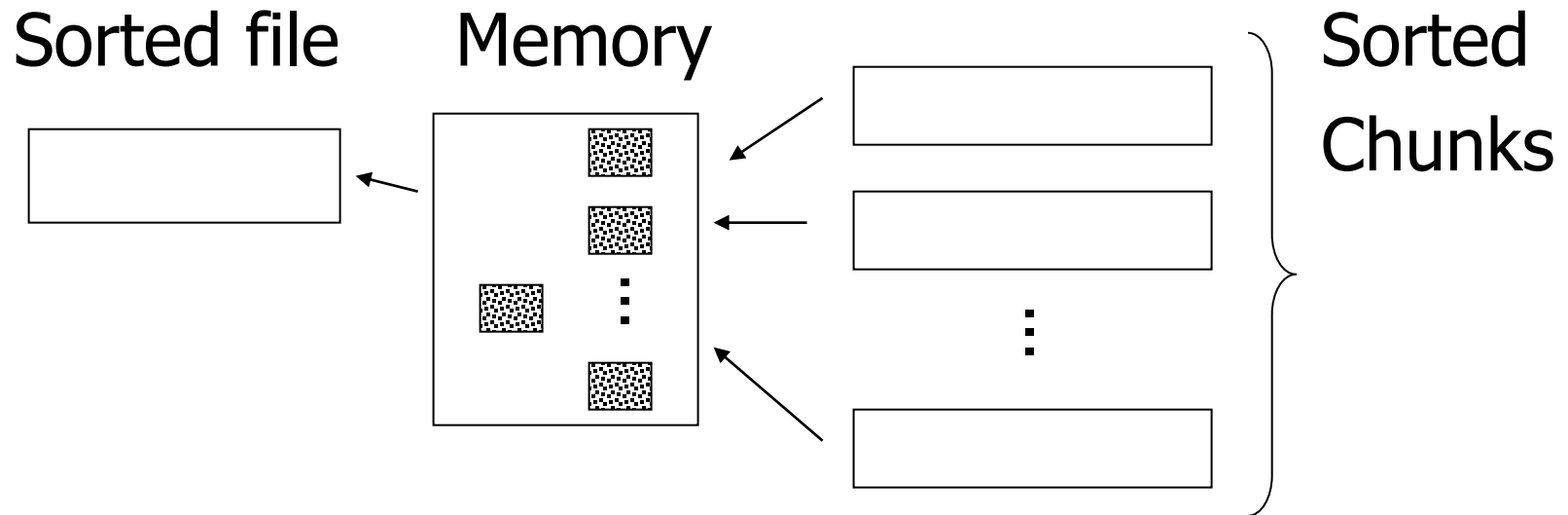
One way to sort: Merge Sort

(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: $4 \times 500 = 2,000$

What would have been result if there was no memory or memory size was less or greater than provided?

Example 1(d) Merge Join (continued)

R1,R2 contiguous, but unordered

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

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

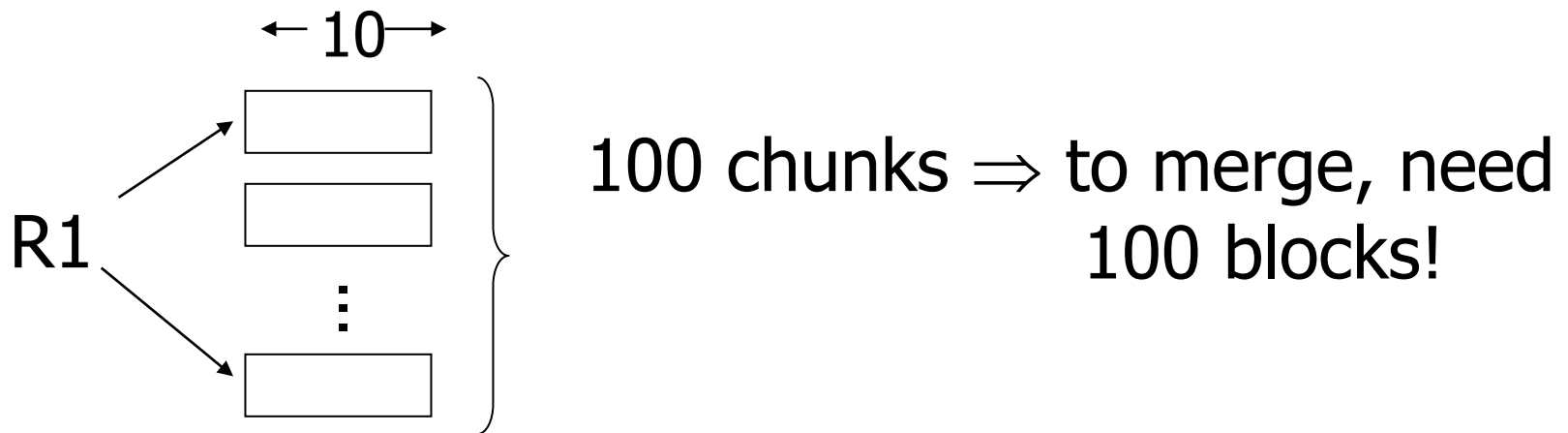
Iterate: $\frac{5000}{100} \times (100 + 10,000) = 50 \times 10,100$
 $= 505,000 \text{ IOs}$

Merge join: $5(10,000 + 5,000) = 75,000 \text{ IOs}$

Merge Join (with sort) WINS!

How much memory do we need for merge sort?

E.g: Say I have 10 memory blocks



In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

chunks \leq buffers available for merge

In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

chunks \leq buffers available for merge

so... $(x/k) \leq k$

or $k^2 \geq x$ or $k \geq \sqrt{x}$

In our example

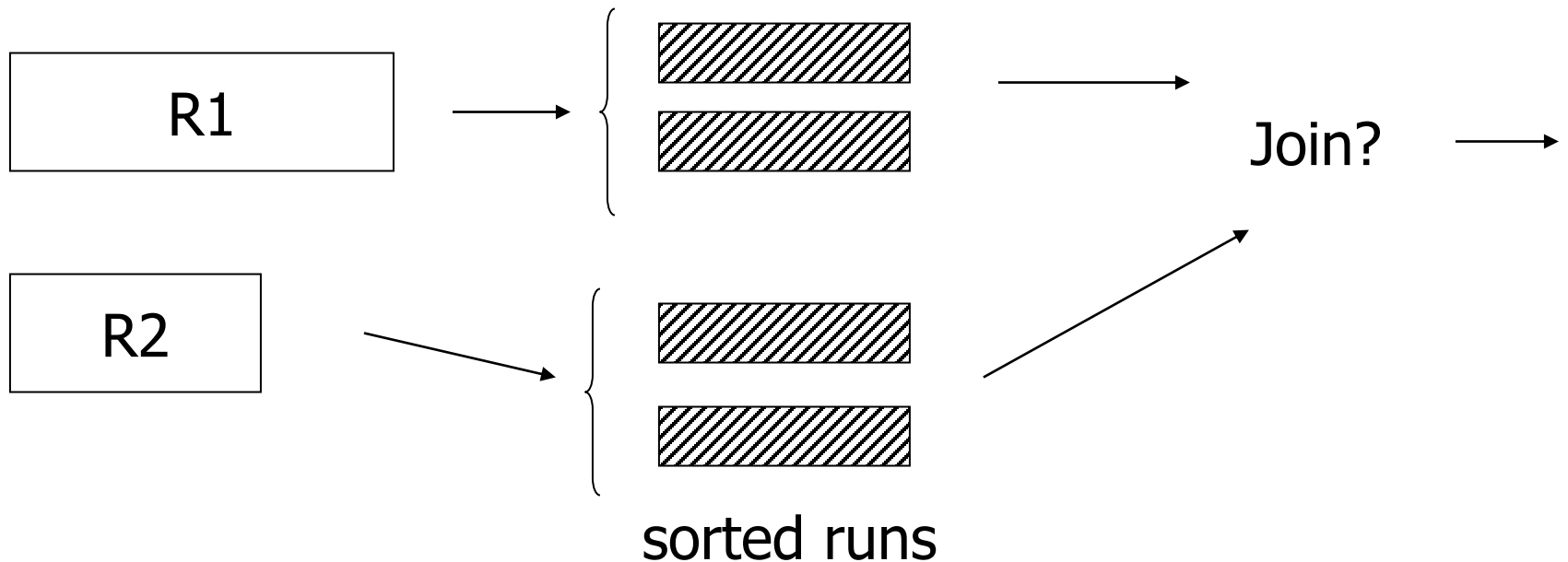
R1 is 1000 blocks, $k \geq 31.62$

R2 is 500 blocks, $k \geq 22.36$

Need at least 32 buffers

Can we improve on merge join?

Hint: do we really need the fully sorted files?



Cost of improved merge join:

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

--> Memory requirement?

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 $\text{expect} = 1$

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

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

500 - Number of IO needed to bring outer relation to memory

5000 - Number of records in outer relation - kati ota relation ko lagi seek garni paryo

1- number of match

1- cost of read

$$(a) \text{ Total cost} = 500 + 5000(1)1 = 5,500$$

$$(b) \text{ Total cost} = 500 + 5000(2)1 = 10,500$$

$$(c) \text{ Total cost} = 500 + 5000(1/100)1 = 550$$

What if index does not fit in memory?

Example: say R1.C index is 201 blocks

- Keep root + 99 leaf nodes in memory
- Expected cost of each probe is

$$E = \frac{(0)99}{200} + \frac{(1)101}{200} \approx 0.5$$

Total cost (including probes)

$$= 500 + 5000 [\text{Probe} + \text{get records}]$$

$$= 500 + 5000 [0.5 + 2] \quad \text{uniform assumption}$$

$$= 500 + 12,500 = 13,000 \quad (\text{case b})$$

Total cost (including probes)

$$= 500 + 5000 [\text{Probe} + \text{get records}]$$

$$= 500 + 5000 [0.5 + 2] \quad \text{uniform assumption}$$

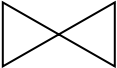
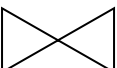
$$= 500 + 12,500 = 13,000 \quad (\text{case b})$$

For case (c):

$$= 500 + 5000 [0.5 \times 1 + (1/100) \times 1]$$

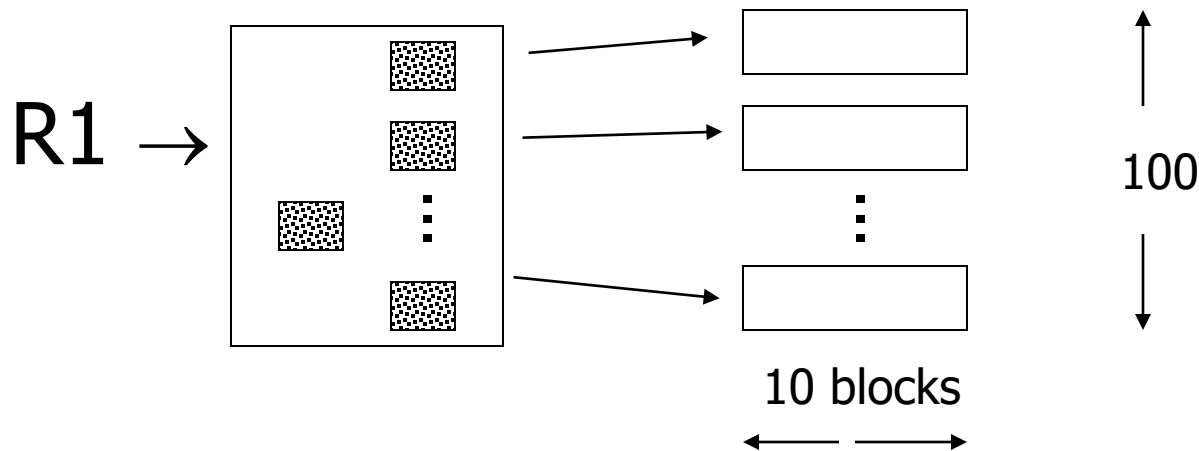
$$= 500 + 2500 + 50 = 3050 \text{ IOs}$$

So far

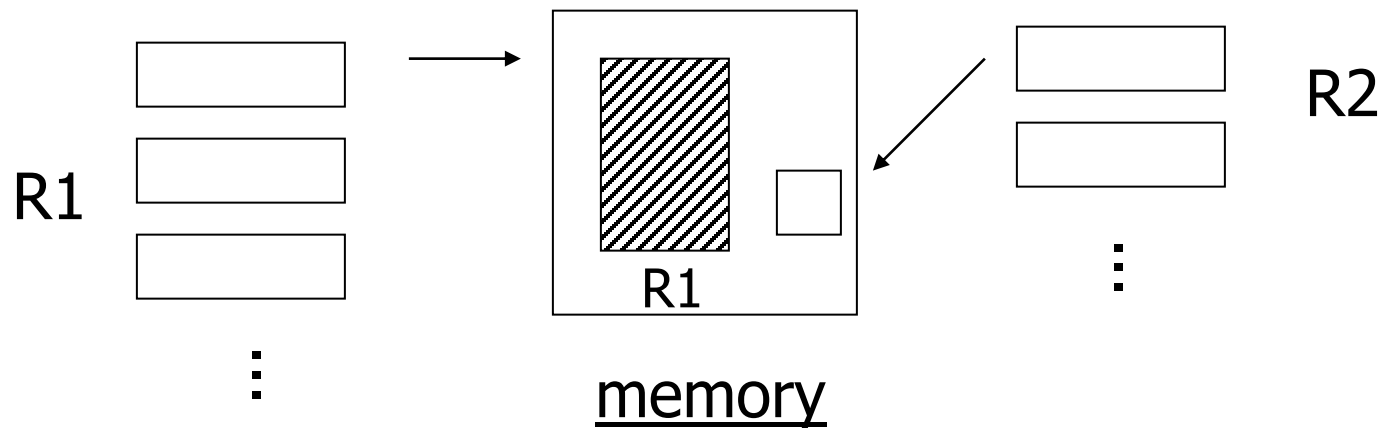
not contiguous	{ Iterate R2  R1	55,000 (best)
	Merge Join	_____
	Sort+ Merge Join	_____
	R1.C Index	_____
	R2.C Index	_____
<hr/>		
contiguous	{ Iterate R2  R1	5500
	Merge join	1500
	Sort+Merge Join	7500 → 4500
	R1.C Index	5500 → 3050 → 550
	R2.C Index	_____

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

read records from first table -> bucketize -> write to partition them
read records from second table -> bucketize -> write to partition them
between each partition check if there are match, for this left ko each euta ko lagi right ko each euta read garni parcha

Cost:

"Bucketize:" Read R1 + write

Read R2 + write

Join: Read R1, R2

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

Cost:

"Bucketize:" Read R1 + write

Read R2 + write

Join: Read R1, R2

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

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

Minimum memory requirements:

Is it using memory?

What would be result if memory available was less or more?

Size of R1 bucket = (x/k)

k = number of memory buffers

x = number of R1 blocks

So... $(x/k) < k$

$k > \sqrt{x}$ need: $k+1$ total memory buffers

Readings

- DATABASE SYSTEMS - The Complete Book, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Second Edition.
- Chapter 16.7.1, 16.7.2, and relevant sections in Chapter 15.