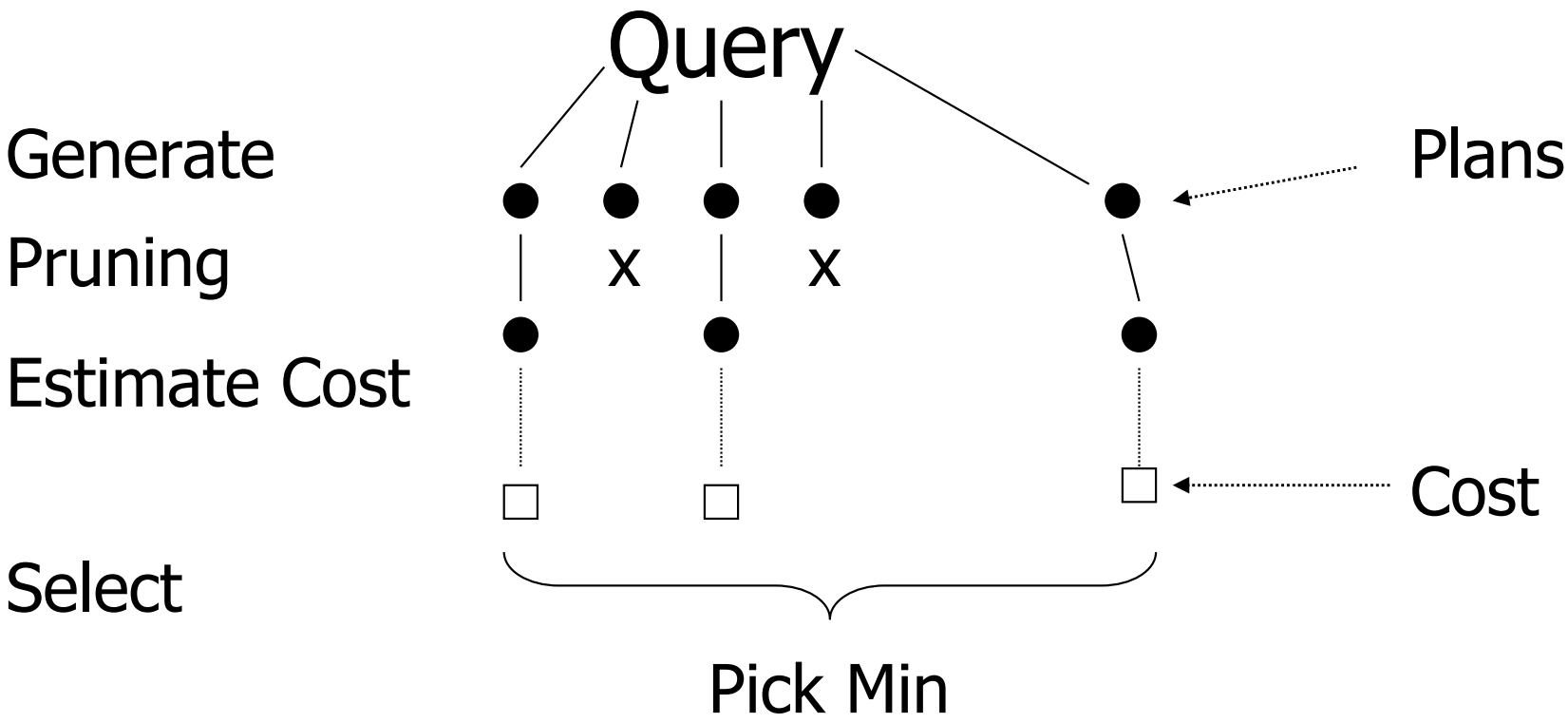
Query Optimization – Physical Query Plans

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:

the disk acces by far the most restraining part when doing a query compared to CPU,..

- 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

T(R) = # of tuples in a relation (all tuples)

T(R) > B(R)

the number of tuples themselves

the number of blocks containing the tuples

if store contiguously

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

→ height (how many level in the tree)

HT(i) = # levels in index i

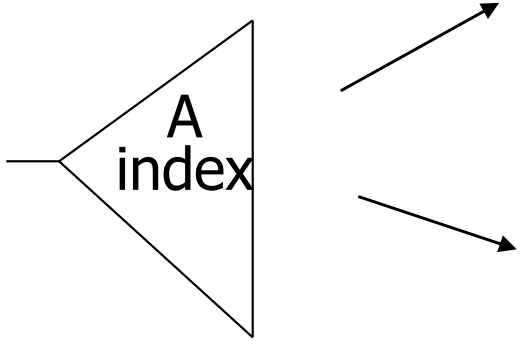
LB(i) = # of leaf blocks in index i

→ using B-Tree indexes

Clustering index

organizing in order of the key

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



A

10	
15	
17	

19	
35	
37	

Notions of clustering

blocks in the file can store tuples from multiple relations but they are ordered

- Clustered file organization

R1 R2 S1 S2	R3 R4 S3 S4	.....

- Clustered relation $\Rightarrow B(R) \ll T(R)$

R1 R2 R3 R4	R5 R5 R7 R8	.....

- Clustering index

Example R1 $\bowtie_{(C=C)}$ R2 over common attribute C

↳ equijoin

T(R1) = 10,000

T(R2) = 5,000

1 block can store 10 tuples

S(R1) = S(R2) = 1/10 block

Memory available = 101 blocks

size of one tuple in the relation

⟹ we can store 1010 tuples if contiguous in all the blocks available

Example R1 $\bowtie$ R2 over common attribute C

T(R1) = 10,000

T(R2) = 5,000

S(R1) = S(R2) = 1/10 block

Memory available = 101 blocks

$\rightarrow$ Metric: # of IOs

(ignoring writing of result)

Options

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

we can also invert the order of the join

set of options we can choose from

- Iteration join (conceptually)

```
for each r ∈ R1 do
    for each s ∈ R2 do
        if r.C = s.C then output r,s pair
```

two nested loops

and for each possibility we output the pair if it verifies the join condition

- Merge join (conceptually)

by C value

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

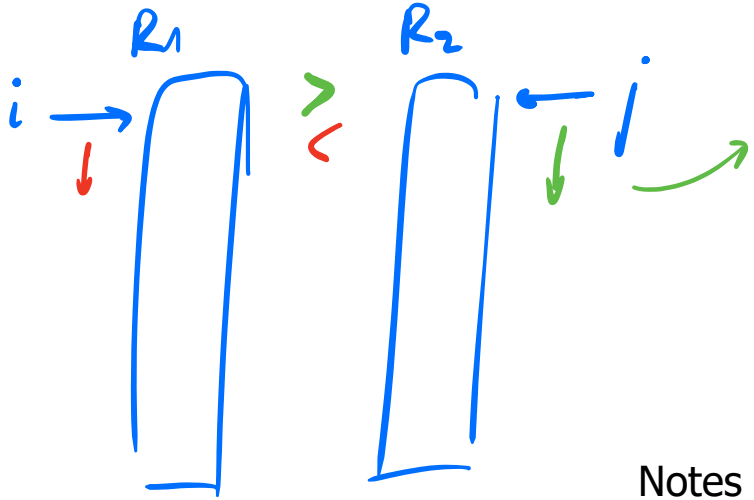
(2) $i \leftarrow 1$; $j \leftarrow 1$;

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 $i \leftarrow i+1$



R2[j] < R1[i]

⇒ we expect the gap to be Power by going further in R2 because sorted from Power to greater

Procedure Output-Tuples

While (R1{ i }.C = R2{ j }.C) ∧ (i ≤ T(R1)) do

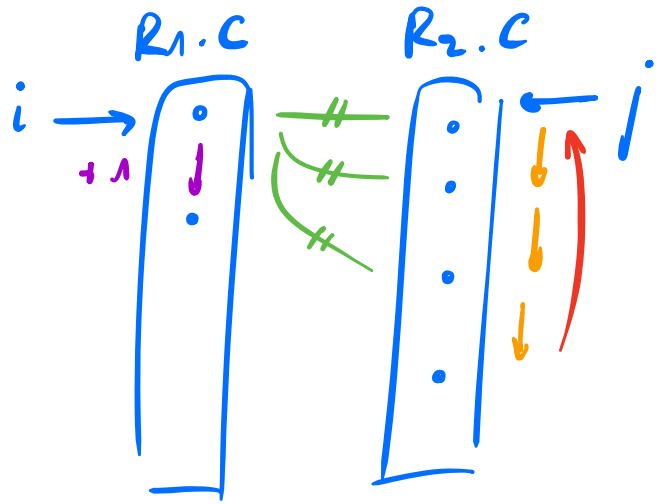
[jj ← j;

while (R1{ i }.C = R2{ jj }.C) ∧ (jj ≤ T(R2)) do

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

jj ← jj+1]

i ← 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)

Assume R2.C index

For each r $\in$ R1 do

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

for each s $\in$ X do

output r,s pair]

we retrieve the tuples with the same value as r.C or C

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

then X = set of rel tuples with attr = value

- Hash join (conceptual)
  - Hash function h, range 0 $\rightarrow$ k
  - Buckets for R1: G0, G1, ... Gk
  - Buckets for R2: H0, H1, ... Hk

- Hash join (conceptual)
  - Hash function h, range 0 $\rightarrow$ k
  - Buckets for R1: G0, G1, ... Gk
  - Buckets for R2: H0, H1, ... Hk

  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 Gi, Hi buckets

Simple example hash: even/odd

R1	R2
2	5
4	4
3	12
5	3
8	13
9	8
	11
	14

Buckets

	R1	R2
Even	2 4 8	4 12 8 14
Odd:	3 5 9	5 3 13 11

↳ then we can join buckets per buckets

⇒ smaller joins to compute

Factors that affect performance

(1) Tuples of relation stored physically together?

(2) Relations sorted by join attribute?

(3) Indexes exist?

↗ nested loop

Example 1(a) Iteration Join R1 ⋈ R2

(not sure if is)

- Relations not contiguous ⇒ tuples are not in the same blocks (side by side)
- Recall
  - T(R1) = 10,000 T(R2) = 5,000
  - S(R1) = S(R2) =1/10 block
  - MEM=101 blocks

Example 1(a) Iteration Join R1 ⋈ R2

- Relations not contiguous
- Recall { T(R1) = 10,000 T(R2) = 5,000
  S(R1) = S(R2) =1/10 block
  MEM=101 blocks

worst case: when reading next tuple we need to bring the next page into the memory

we are counting 1 I/O per read (as if we always needed the next page)

Cost: for each R1 tuple:

[Read tuple + Read R2]

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

for all tuples of R1

read the tuple R1 to can

whole R2 must be read to compare with the R1 tuple

- 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

$(1000 + 5000) \times 10 = 60000$

$< 50\,000\,000$

= 55000

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2: 5000 IOs

\+ 6000

what if we change the memory allocated to each table:

50 + 51

R1 R2

(500 + 5000) x 20

we still have to read the whole relation

Amiving factor

$R_2 \bowtie R_1$

100 1

(1000 + 10000) x 5

less chunks to read ⇒ better

= 55000

> 60000

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2: 5000 IOs

6000

Total = 10,000 / 1,000 x 6000 = 60,000 IOs

number of chunks of R1

read chunk + read whole R2

< 50.000.000 ⇒ way better

- Can we do better?

- Can we do better?

  - ☛ Reverse join order: R2 ⋈ R1

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

5 x 11,000 = 55,000 IOs

Example 1(b) Iteration Join R2 ⋈ R1

- Relations contiguous

nbr of chunks:

Size of table / Memory allocated

- $\frac{10000}{1000} \times (1000 + 5000) = 60000$
- $\left\lceil \frac{10000}{1000} \right\rceil \times (100 + 500) = 6000$

becomes

constant in regard to previous example (not contiguous)

PIP in the memory only required 100 I/O to retrieve 100 blocks ⇒ 1000 tuples

↳ blocks in disk are contiguous → 1 block = 10 relation tuples

⇒ 1 I/O return 10 tuples → 10 times less I/O

1 block = 1 tuple if previously

Example 1(b) Iteration Join R2 ⋈ R1

- Relations contiguous

5 (100 + 1000)
= 5500

Cost

For each R2 chunk:

Read chunk: 100 IOs

Read R1: 1000 IOs

1,100

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

of 1000 tuples

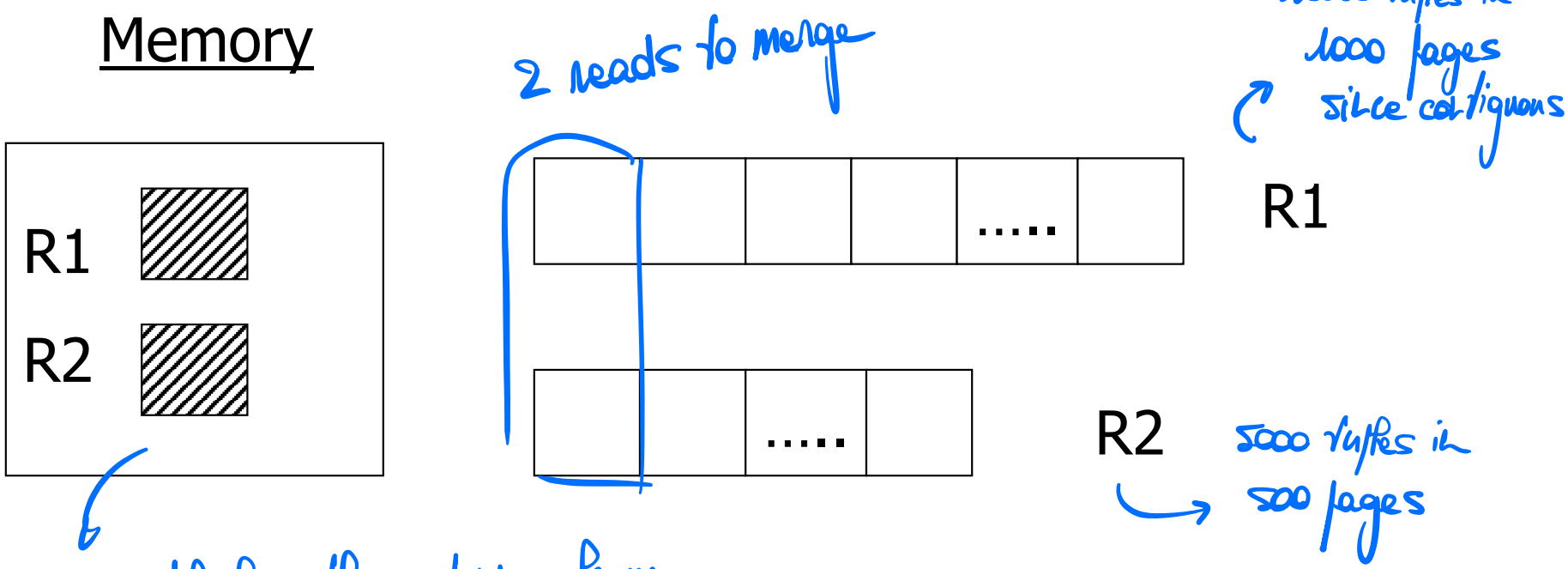
100 contiguous blocks

read 100 blocs of R2 + read 1000 blocks of R1 ↳ 10000 tuples

Example 1(c) Merge Join

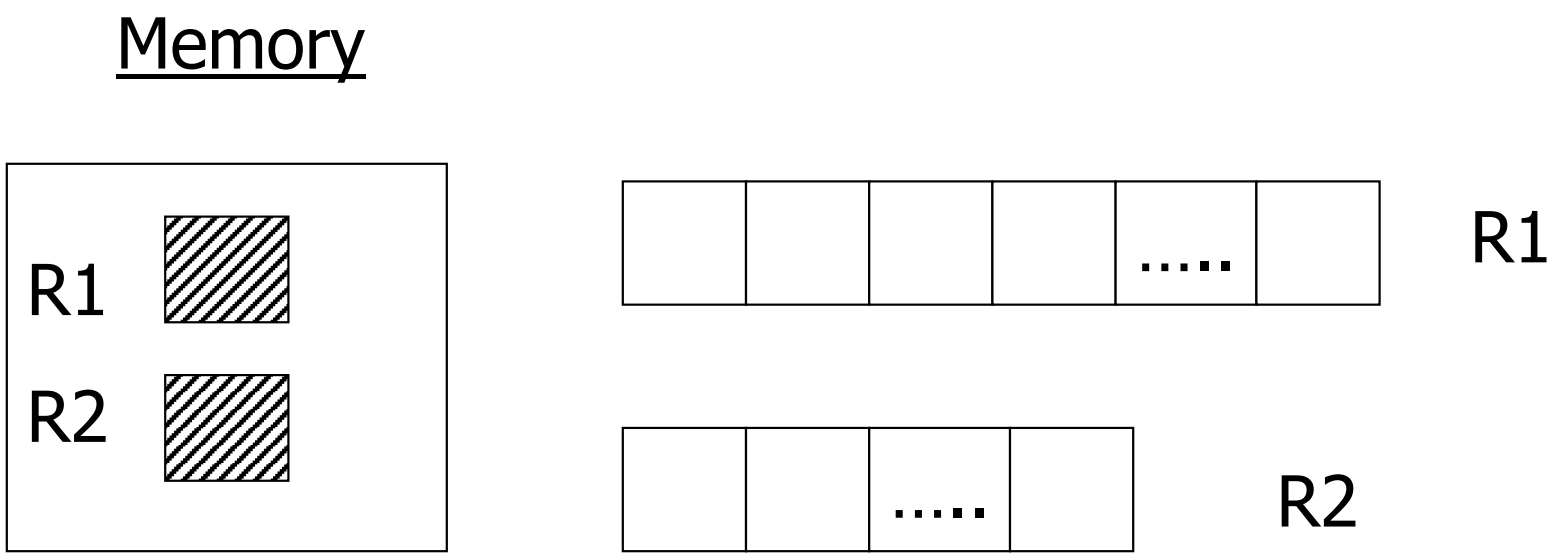
→ attribute of equijoin

- Both R1, R2 ordered by C; relations contiguous



Example 1(c) Merge Join

- Both R1, R2 ordered by C; relations contiguous



Total cost: Read R1 cost + read R2 cost

= 1000 + 500 = 1,500 IOs

Example 1(d) Merge Join

- R1, R2 not ordered, but contiguous

must be ordered for merge join

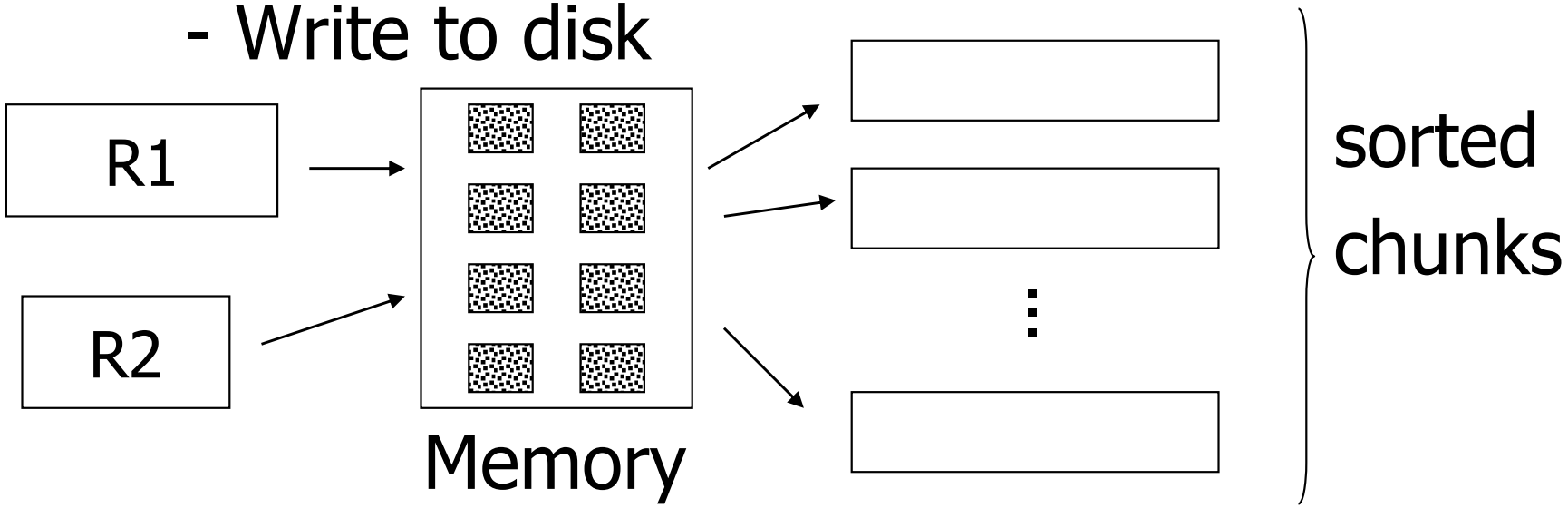
--> Need to sort R1, R2 first.... HOW?

One way to sort: Merge Sort

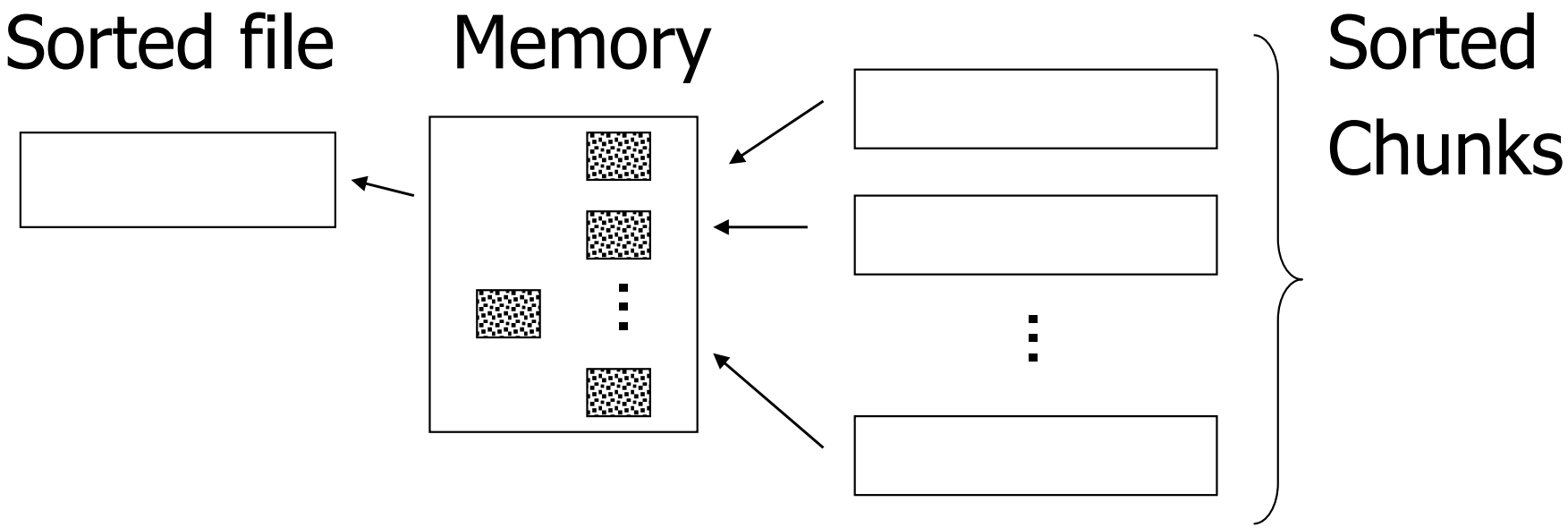
(i) For each 100 blk chunk of R:

- Read chunk
- Sort in memory
- Write to disk

we sort each relation independently from the other



(ii) Read all chunks + merge + write out



Cost: Sort

Each tuple is read,written,
read, written

so...

Sort cost R1: 4 x 1,000 = 4,000

Sort cost R2: 4 x 500 = 2,000

Merge Join vs Iterative Join on contiguous

- $T(R1) = 10000 \mid T(R2) = 5000$
- $S(R1) = S(R2) = 1/10$
- MEM = 101 blocks

a. Not contiguous

1. Iterative Join:
   - $R1 \bowtie R2$: $\frac{10.000}{1.000} \times (1000 + 5000) = 60.000$
   - $R2 \bowtie R1$: $\frac{5.000}{1.000} \times (1000 + 10000) = 55.000$

2. Merge Join: require sequential + ordered
   - sequentiality: $\begin{cases} \text{read } R1 + \text{write } R1 = 10000 + 1000 = 11.000 \\ \underbrace{\text{read } R2}_{\text{not seq.}} + \underbrace{\text{write } R2}_{\text{seq.}} = 5000 + 500 = 5.500 \end{cases}$ $\Big) 16.000$
   - order (already sequential): $\begin{cases} 2 \times (\text{read } R1 + \text{write } R1) = 2 \times 2000 = 4000 \\ 2 \times (\text{read } R2 + \text{write } R2) = 2 \times 1000 = 2000 \end{cases}$ $\Big) 6.000$
   - join: read R1 + read R2 = 1000 + 500 = 1.500 $\Longrightarrow$ total: 23.500

b. Contiguous

1. Iterative Join:
   - $R1 \bowtie R2$: $\frac{10.000}{1.000} \times (100 + 500) = 6.000$
   - $R2 \bowtie R1$: $\frac{5.000}{1.000} \times (100 + 1000) = 5.500$

2. Merge Join:
   - order: $\begin{cases} 2\,(\overset{\text{read}}{1000} + \overset{\text{write}}{1000}) = 4000 \\ 2\,(500 + 500) = 2000 \end{cases}$ $= 6.000$
   - join: read R1 + read R2 = 1500 $\Longrightarrow$ total = 7500

c. Contiguous + ordered:

$\longrightarrow$ Iterative: 5.500 $\longrightarrow$ nothing change

$\longrightarrow$ Merge: 1.500 $\longrightarrow$ no need to sort first

Example 1(d) Merge Join (continued)

R1,R2 contiguous, but unordered

Total cost = sort cost + join cost

= 6,000 + 1,500 = 7,500 IOs

Contiguous and unordered

iteration : $\frac{B(R_1)}{M-1}\left(M-1 + B(R_2)\right)$

merge :
- sort $R_1 = 4B(R_1)$
- sort $R_2 = 4B(R_2)$
- merge $= B(R_1) + B(R_2)$

$= 5\left(B(R_1) + B(R_2)\right)$

$$\frac{B(R_1)}{M-1}\left(M-1+B(R_2)\right) = 5B(R_1)+5B(R_2)$$

$$\Leftrightarrow \quad B(R_1)\left(\frac{1}{M-1}\left(M-1+B(R_2)\right)-5\right) = 5B(R_2)$$

$$\Leftrightarrow \quad B(R_1)\left(\frac{B(R_2)}{M-1}-4\right) = 5B(R_2)$$

$$\Leftrightarrow \quad B(R_1) = \frac{5B(R_1)}{\frac{B(R_2)}{M-1}-4}$$

<u>example</u> : $M = 101$

$B(R_2) = 5{,}000$

$$\hookrightarrow \quad \frac{25{,}000}{\frac{5{,}000}{100}-4} = \frac{25{,}000}{46} \approx 544$$

Heroku > Merge

if $B(R_1) < \dfrac{5B(R_1)}{\frac{B(R_2)}{M-1}-4}$

Example 1(d) Merge Join (continued)

R1,R2 contiguous, but unordered

Total cost = sort cost + join cost

= 6,000 + 1,500 = 7,500 IOs

But: Iteration cost = 5,500

so merge joint does not pay off!

But say R1 = 10,000 blocks contiguous

R2 = 5,000 blocks not ordered

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

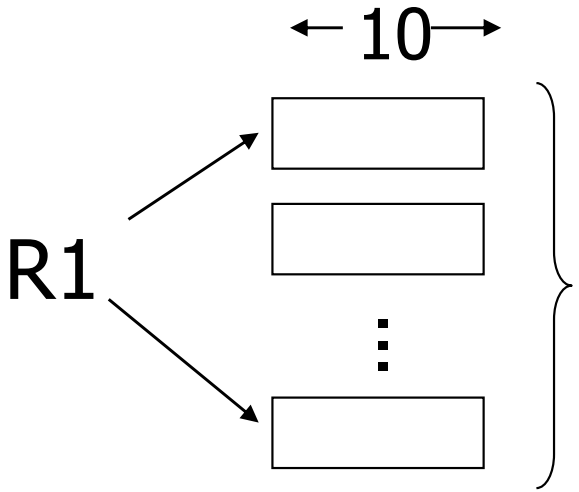
= 505,000 IOs

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

Merge Join (with sort) WINS!

How much memory do we need for merge sort?

E.g: Say I have 10 memory blocks (only)



and I made chunks of 10 blocks each from R1

↳ 100 chunks ⇒ to merge, need 100 blocks!

of 10 blocks each

read first block of each at the same time

⇒ we don't have enough memory

need to have a min of chunck that allows for merge sort

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 $k \geq \sqrt{x}$

k: number of page available in memory

$\sqrt{x}$: $\sqrt{\text{number of blocks in the relation}}$

In our example

R1 is 1000 blocks, $k \geq 31.62$ 32 needed

R2 is 500 blocks, $k \geq 22.36$ 23 needed

Need at least 32 buffers

Memory requirements of merge sort:

- R1 : $T(R1) = 10.000$ tuples

  $S(R1) = 1/10$ block/tuple $\Rightarrow$ $f(R) = 10$ tuples/block

  require 1000 contiguous blocks to store R1 entirely $= \gamma$

↳ we want : # chunks $\leq$ # blocks in memory $= \ell$ $\Leftrightarrow$ $\frac{\gamma}{\ell} \leq \ell$

$\Leftrightarrow \ell^2 \geq \gamma$

$\Leftrightarrow \ell \geq \sqrt{\gamma}$

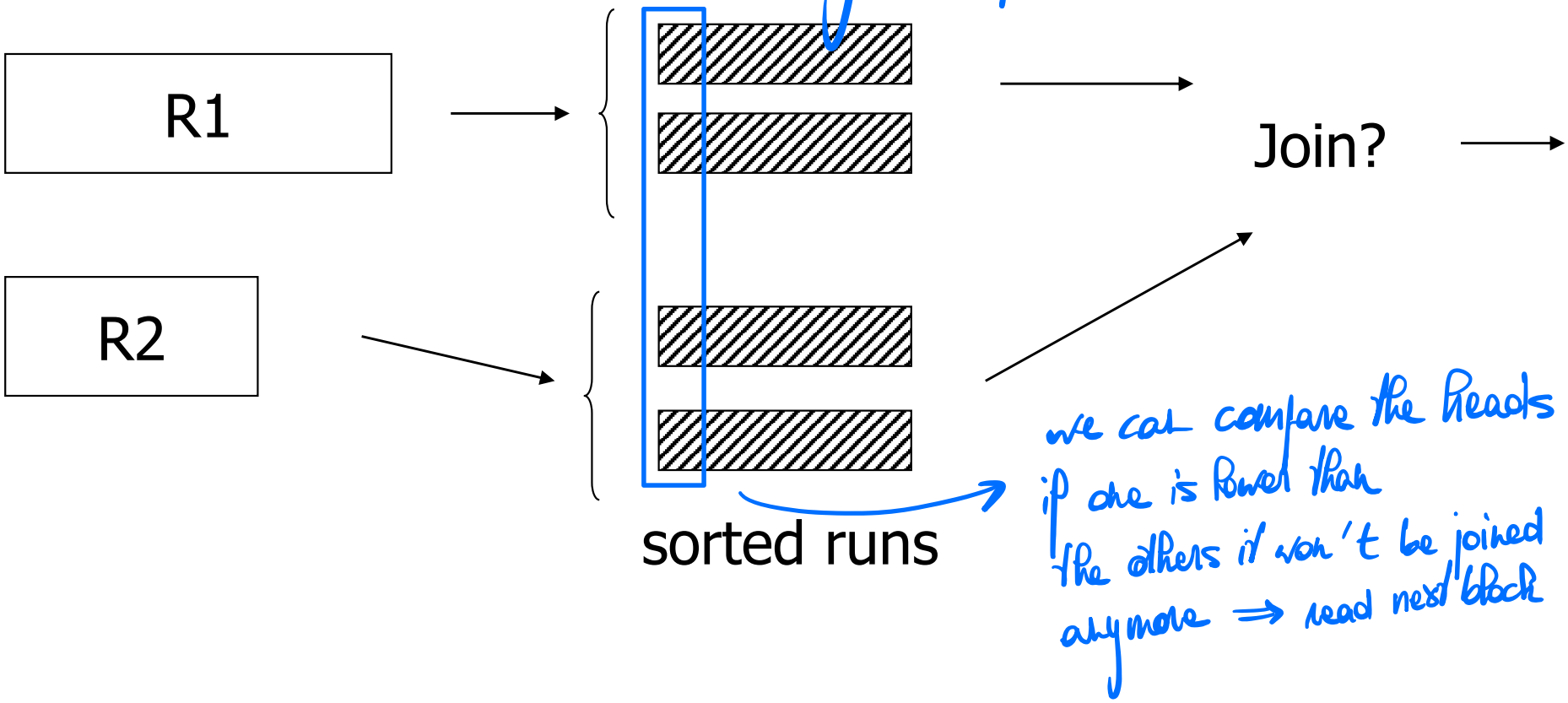
↳ Here $\sqrt{\gamma} \approx 32$

$\Rightarrow$ require 32 blocks of mem.

$$\frac{\#\text{ blocks of } R}{\#\text{ blocks in memory}} = \frac{\gamma}{\ell}$$

Can we improve on merge join?

Hint: do we really need the fully sorted files? → do we need to merge completely ?



Cost of improved merge join:

C = Read R1 + write R1 into runs
+ read R2 + write R2 into runs
+ join
= 2000 + 1000 + 1500 = 4500

we do not merge them anymore

--> Memory requirement?

we need one memory buffer per sorted runs to apply the merge join

Example 1(e) Index Join

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

- Assume R1.C index fits in memory

⟹ querying the index is free

↳ read all tuples of R2 is the only cost

↳ happy case

Cost: Reads: 500 IOs

because 5000 tuples in blocks of 10 tuples ⇒ 500 blocks to read

for each R2 tuple:

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

⚠ output of a join require to read a block of R1 ⇒ 1 IO per match

What is expected # of matching tuples?

(a) say R1.C is key, R2.C is foreign key

then expect = 1 → maximum one matching value since keys are unique

→ # of different values (distinct values)

(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?

domain of C (possible range)

(c) Say DOM(R1, C)=1,000,000

T(R1) = 10,000

with alternate assumption

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

% of matching tuples

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

⟹ requires one more parameter "the selectivity"

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

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

cost if in memory (0 because access in memory is free)

cost of bringing a page to memory (if not in memory)

Total cost (including probes)

= 500+5000 [Probe + get records]

= 500+5000 [0.5+2] uniform assumption

= 500+12,500 = 13,000 (case b)

Total cost (including probes)

= 500+5000 [Probe + get records]

= 500+5000 [0.5+2] uniform assumption

= 500+12,500 = 13,000 (case b)

For case (c):

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

= 500+2500+50 = 3050 IOs

So far

not contiguous

Iterate R2 $\bowtie$ R1	55,000 (best)
Merge Join	_____
Sort+ Merge Join	_____
R1.C Index	_____
R2.C Index	_____

contiguous

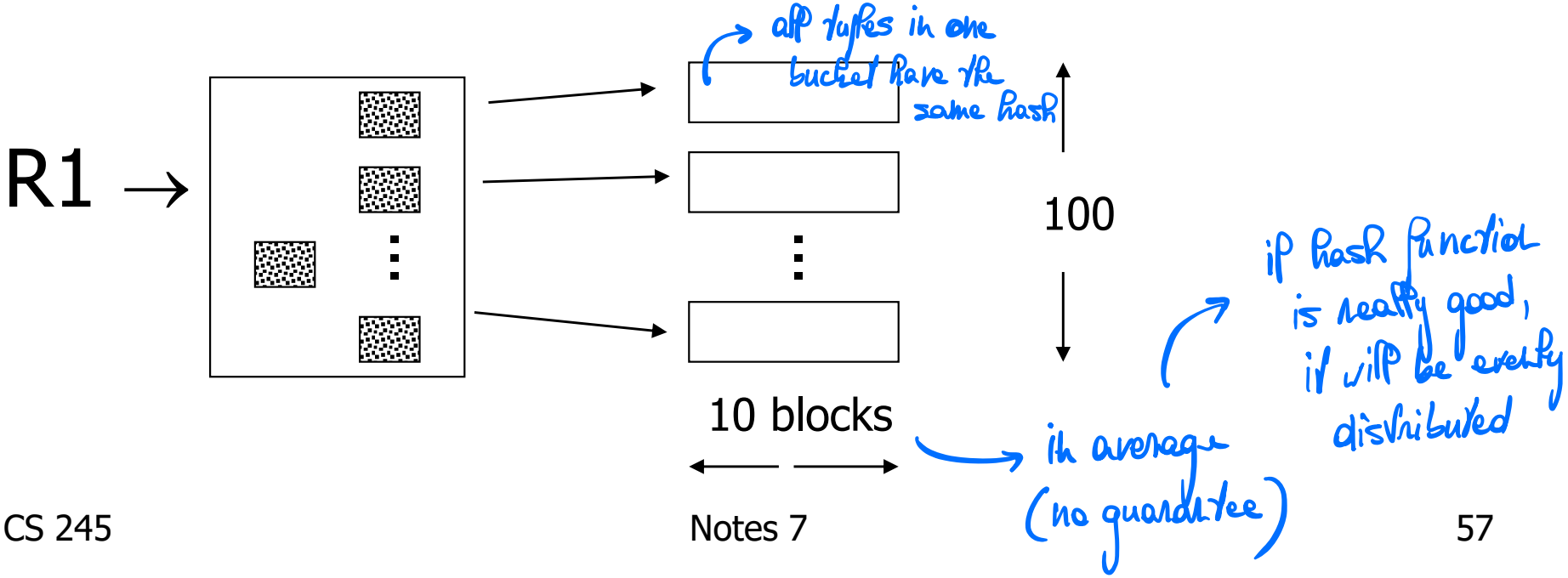
Iterate R2 $\bowtie$ R1	5500
Merge join	1500
Sort+Merge Join	7500 $\rightarrow$ 4500
R1.C Index	5500 $\rightarrow$ 3050 $\rightarrow$ 550
R2.C Index	_____

Example 1(f) Hash Join

- R1, R2 contiguous (un-ordered)

→ Use 100 buckets

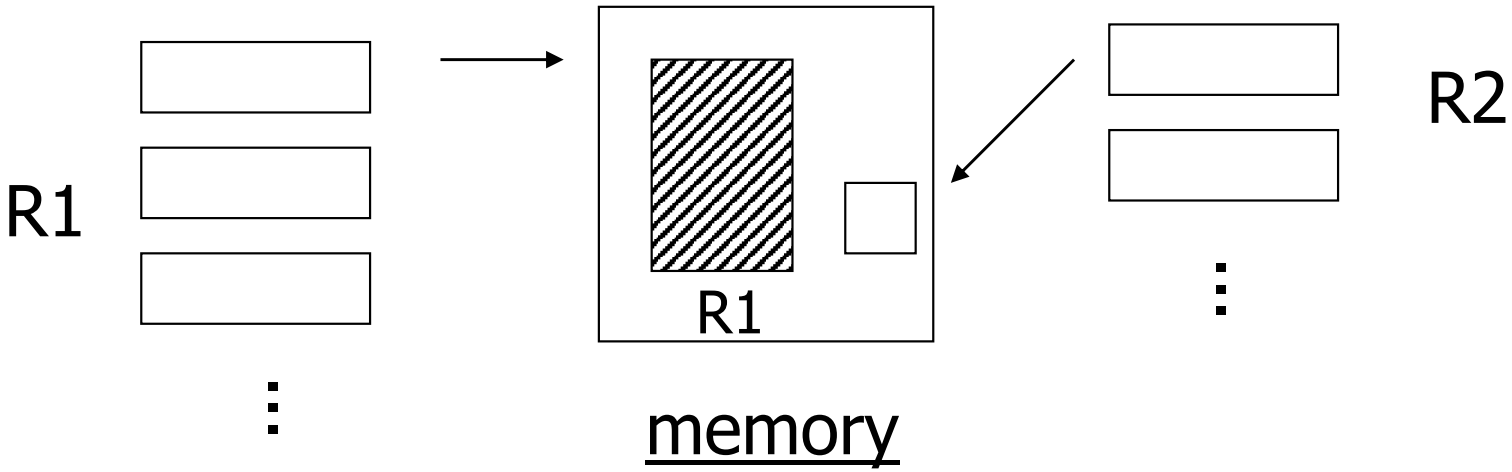
→ Read R1, hash, + write buckets



-> Same for R2

-> Read one R1 bucket; build memory hash table

-> Read corresponding R2 bucket + hash probe



Then repeat for all buckets

Cost:

"Bucketize:" Read R1 + write

Read R2 + write

Join: Read R1, R2

Total cost = 3 x [1000+500] = 4500

Cost:

"Bucketize:" Read R1 + write

Read R2 + write

Join: Read R1, R2

Total cost = 3 x [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:

size of the buckets

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

Memory Requirements:

- Iterative join: { 1 buffer for second relation read through
  rest of them to hold part of the first relation

  → 101 → 100/1
  1001 → 1000/1

↳ why so? Let { $T(R_1) = 10.000$ and $S(R_1) = S(R_2) = 1/10$
$T(R_2) = 5.000$

↳ $\rho(R_1) = \rho(R_2) = 10$

→ with 101 MEM: 100/1

$$\frac{10.000}{1000} \times (1000 + 5.000) = 6000 \times 10 = 60.000$$

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.