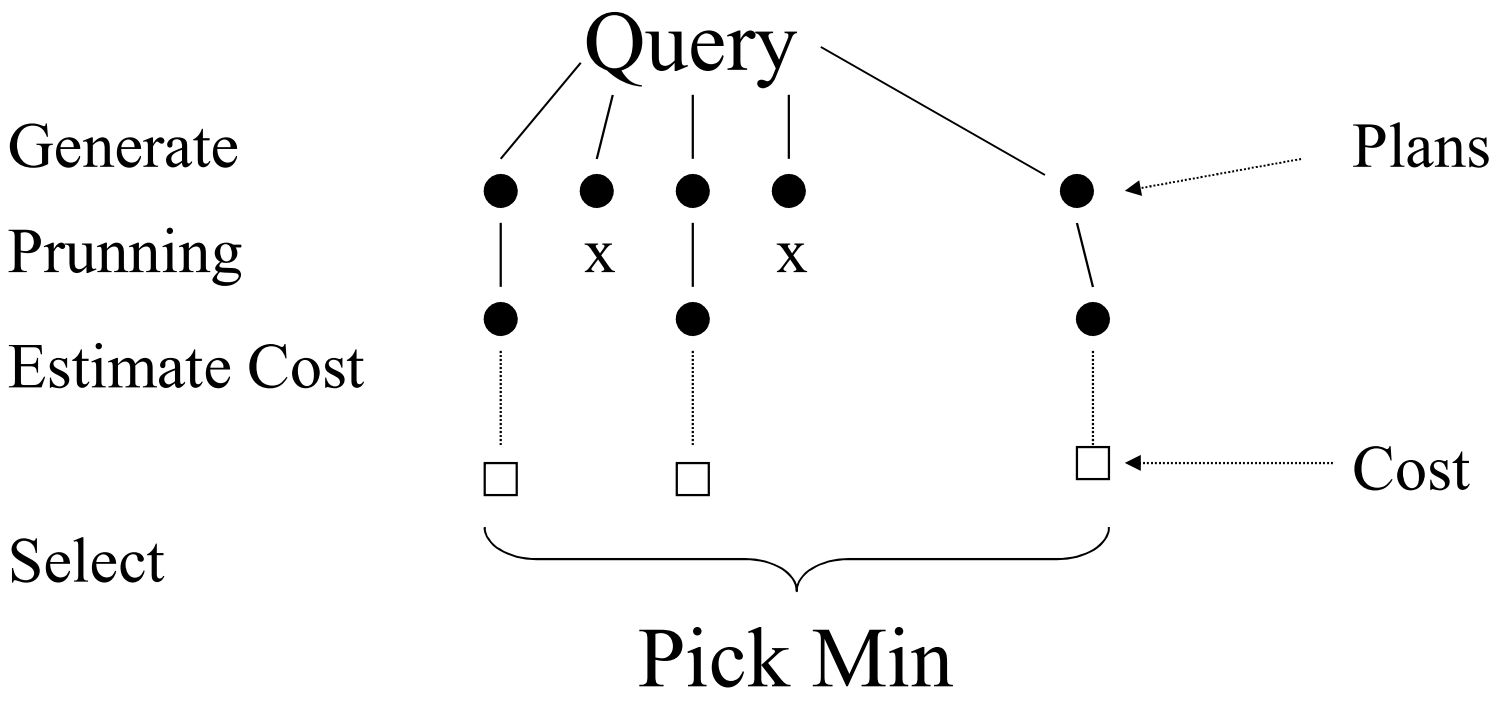
Procesamiento y optimización de consultas (III)

Query Optimization

--> Generating and comparing plans



To generate plans consider:

- Transforming relational algebra expression
- Use of existing indexes
- Is data contiguous on disk?
- Building indexes or sorting on the fly
- Implementation details:

→ 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 following parameters:

B(R) = # of blocks containing R tuples

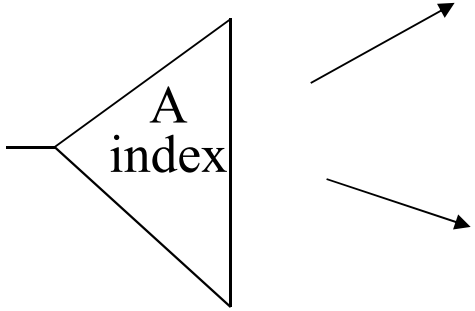
f(R) = max # of tuples of R per block or

S(R) = size of every tuple based on the size of a block.

M = # memory blocks available

Primary index

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



A

10	
15	
17	

19	
35	
37	

Example R1 $\bowtie_C$ R2

T(R1) = 10,000

T(R2) = 5,000

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

Available memory= 101 blocks

→ Measure: # of I/Os

(ignoring writing of results)

Options

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

- Iteration join

for each r ∈ R1 do

for each s ∈ R2 do

if r.C = s.C then output r,s

- Merge join

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

  (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 output Tuples

  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) ∧ (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

Assume a R2.C index

For each r ∈ R1 do

[X ← index (R2, C, r.C)

for each s ∈ X do

output r,s]

Note: X ← index(rel, atr, value)

X = set of tuples with atr=value

- Hash join
  - Hash function h, range 0 → k
  - Buckets for R1: G0, G1, ... Gk
  - Buckets for R2: H0, H1, ... Hk

Algorithm

(1) Hash R1 tuples into G buckets

(2) For each r ∈ R2 do

(3) h(r) = j

(4) Match r with tuples in Gj

Easy 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

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 ⋈ R2

- Relations not contiguous. Mem is not empty.
- Recall

$$\begin{cases} T(R1) = 10{,}000 \quad T(R2) = 5{,}000 \\ S(R1) = S(R2) = 1/10 \text{ bloque} \\ MEM = \text{the minimum neccesary} \end{cases}$$

Cost: for each R1 tuple:

[Read tuple + Read R2]

Total = 10,000 + [10000 * 5000] = 50,010,000 IOs

- Can we do better?

Use our memory

MEM=101 blocks

(1) Read 100 blocks of R1

(2) Read all of R2 (using 1 block) + join

(3) Repeat until done

Relations not contiguous. Mem is empty.

Cost: for each R1 chunk:

Read chunk: 1000 IOs

Read R2: 5000 IOs

6000

$$\text{Total} = \frac{10{,}000}{1{,}000} \times 6000 = 60{,}000 \text{ I/Os}$$

- Can we do better?

☛ Reverse join order: R2 ⋈ R1

$$\text{Total} = \frac{5000}{1000} \times (1{,}000 + 10{,}000) =$$

$$5 \times 11{,}000 = 55{,}000 \text{ I/Os}$$

Example 1(b) Iteration Join R2 ⋈ R1

- Relation contiguos

Cost

For each R2 chunk:

Read chunk: 100 IOs

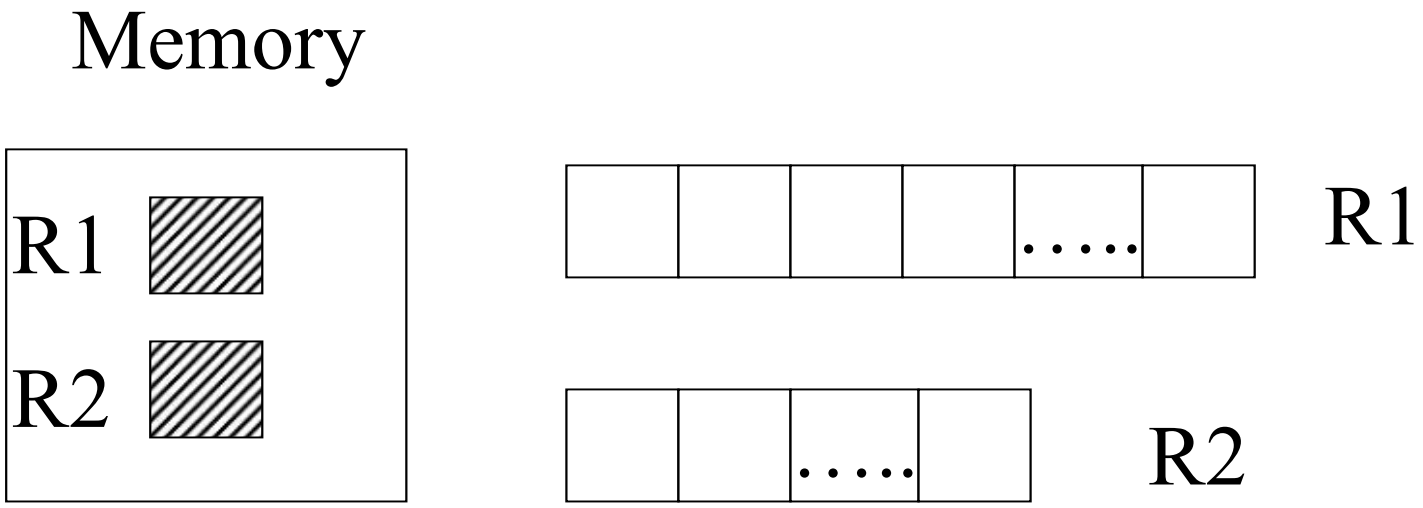
Read R1: 1000 IOs

1,100

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

Example 1(c) Merge Join

- Both R1, R2 ordered by C; relations contiguous



Total cost: Read R1 cost + read R2 cost

= 1000 + 500 = 1,500 IOs

Ejemplo 1(d) Merge Join

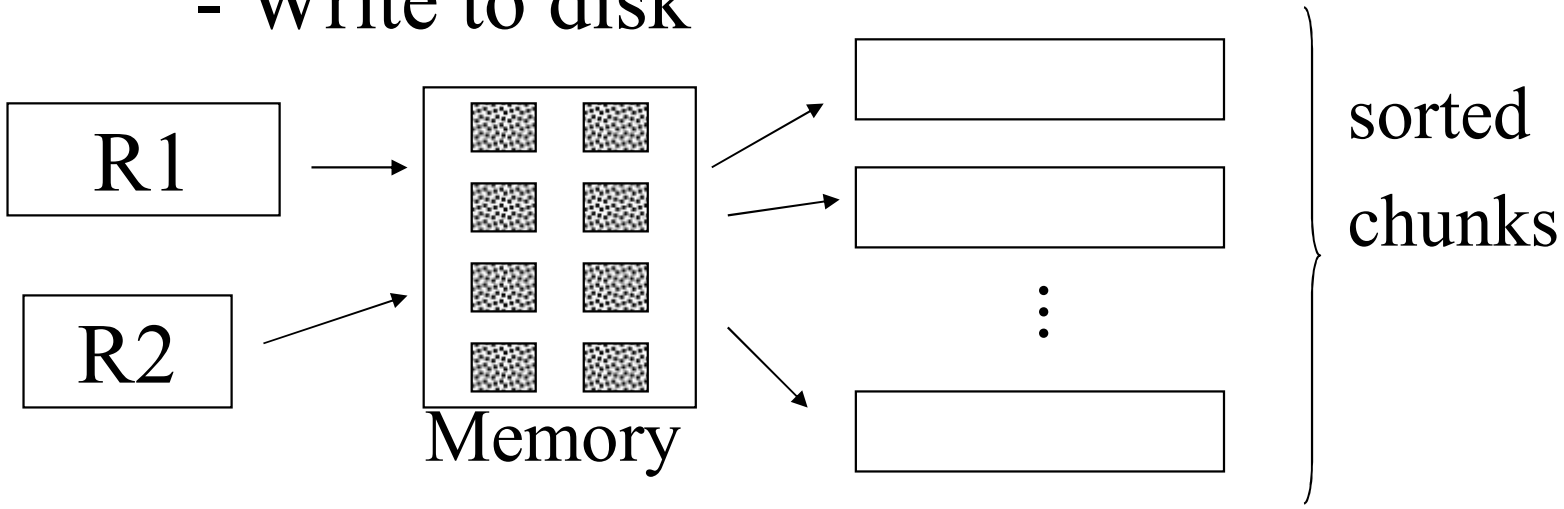
- R1, R2 not ordered, but contiguous

--> Need to sort R1, R2 first…. HOW?

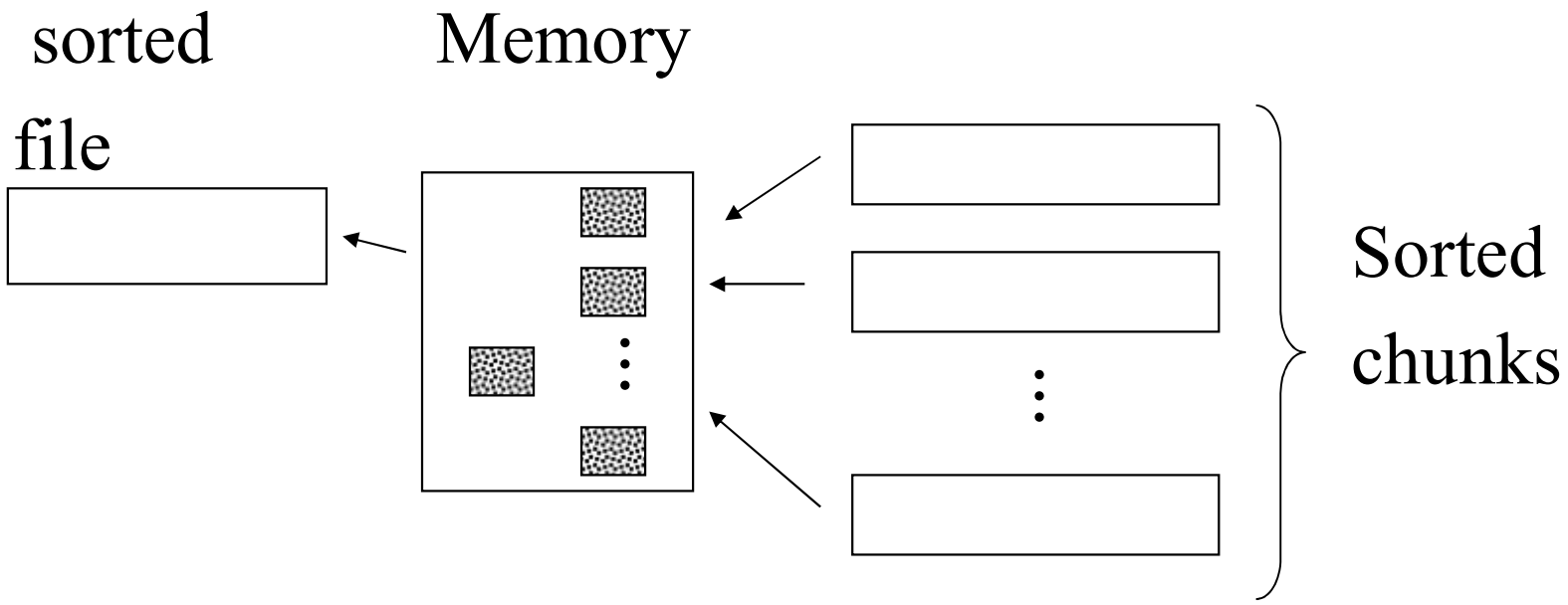
One way to sort: Merge Sort

(i) For each 100 blocks chunk of R:

- Read chunk
- Sort in memory
- Write to disk



(ii) Read all chunks + merge + write out



Cost: Sort

Each block is read and written,

read and written

so ...

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

Sort Cost R2: 4 x 500 = 2,000

Example1(d) Merge Join (continued)

- R1, R2 not ordered, but contiguous

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 if T(R1) = 100,000 rows contiguous

T(R2) = 50,000 rows not ordered

B(R1) = 10,000 blocks

B(R2) = 5,000 block

Iterate: 50 x (100+10,000) =

= 50 x 10,100 = 505,000 I/Os

Merge join: 5(10,000+5,000) = 75,000 I/Os

Merge Join (with sort) WINS!

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:

for each R2 tuple: 500 IOs

- probe index - free

- if match, read R1 tuples: k IO

COST = 500 IO + 5000 * k

What is the expected value of k?

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

then expect k = 1

(b) say V(R1,C) = 5000, T(R1) = 10,000

with uniform assumption

expect k = 10,000/5,000 = 2

Total cost with index join

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

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

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

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

Total cost (including probes)

= 500+5000 [Probe + get records]

= 500+5000 [0.5+2] uniform assumption

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

So far

not contiguous

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

---

contiguous

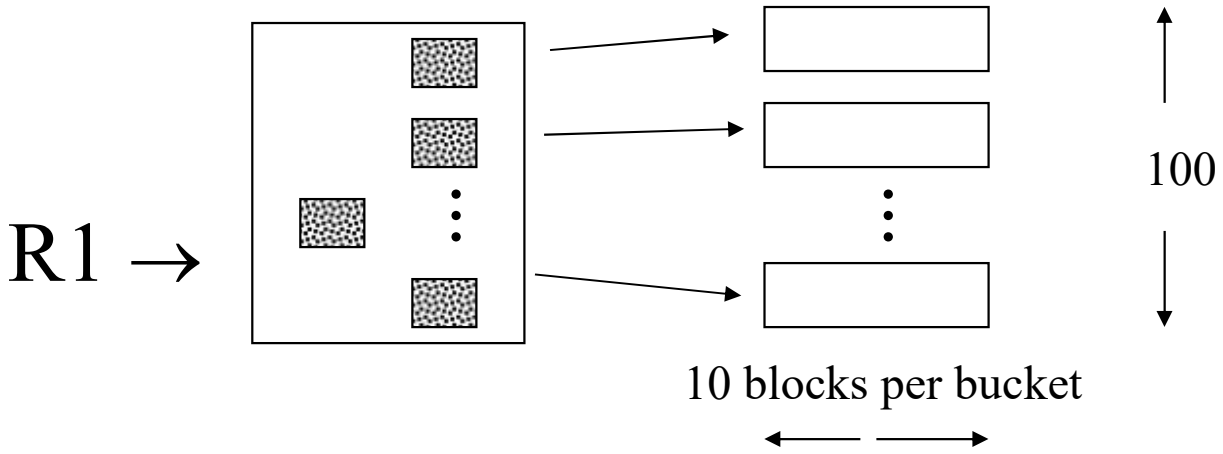
Iterate R2 ⋈ R1	5500
Merge join	1500
Sort+Merge Join	7500
R1.C Index	5500
R2.C Index	_____

Example 1(f) Hash Join

- R1, R2 contiguous (un-ordered)

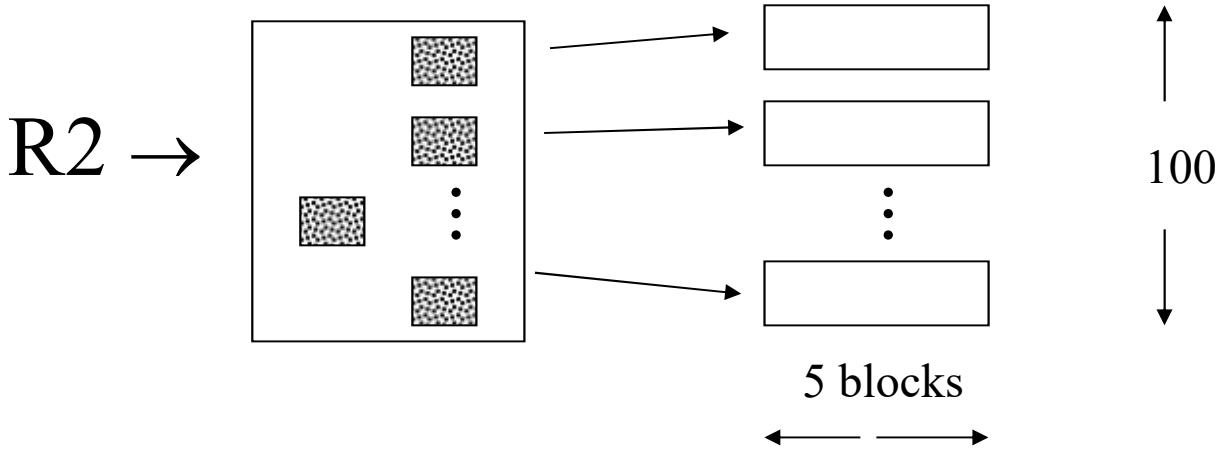
→ Use 100 buckets

→ Read R1, hash, + write buckets



Example 1(f) Hash Join

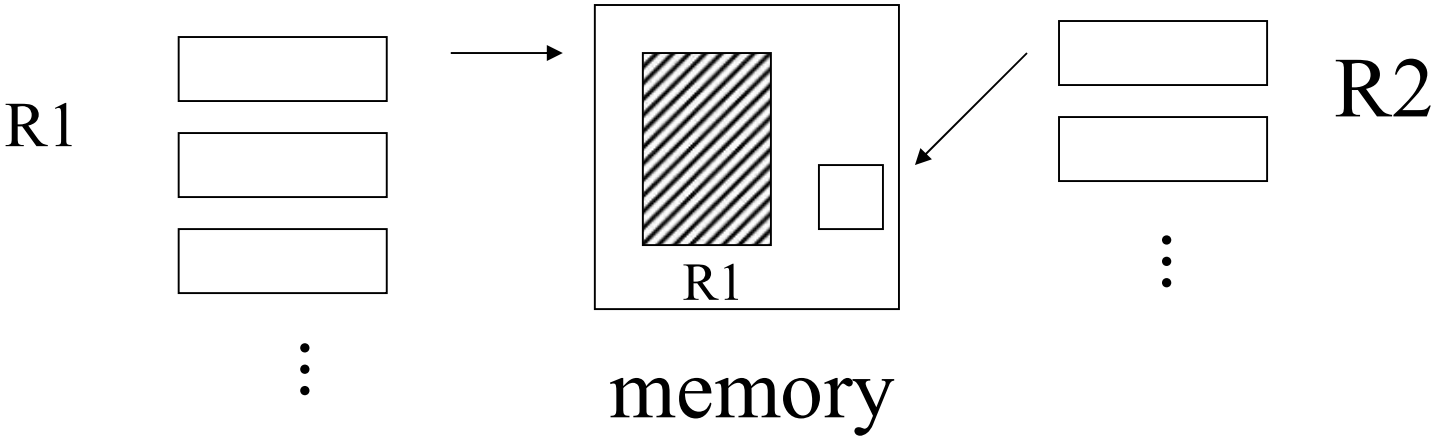
→ Read R2, hash, + write buckets



Join process

-> Read one R1 bucket; build memory hash table

-> Read corresponding R2 bucket + hash probe



Then repeat for all buckets

Cost:

"Bucketize:" Read R1 + write buckets

Read R2 + write buckets

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

contiguous

Iteraratio	5500
Merge join	1500
Sort+merge join	7500→ 4500
R1.C index	5500
R2.C index	_____
Build R.C index	_____
Build S.C index	_____
Hash join	4500+

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)