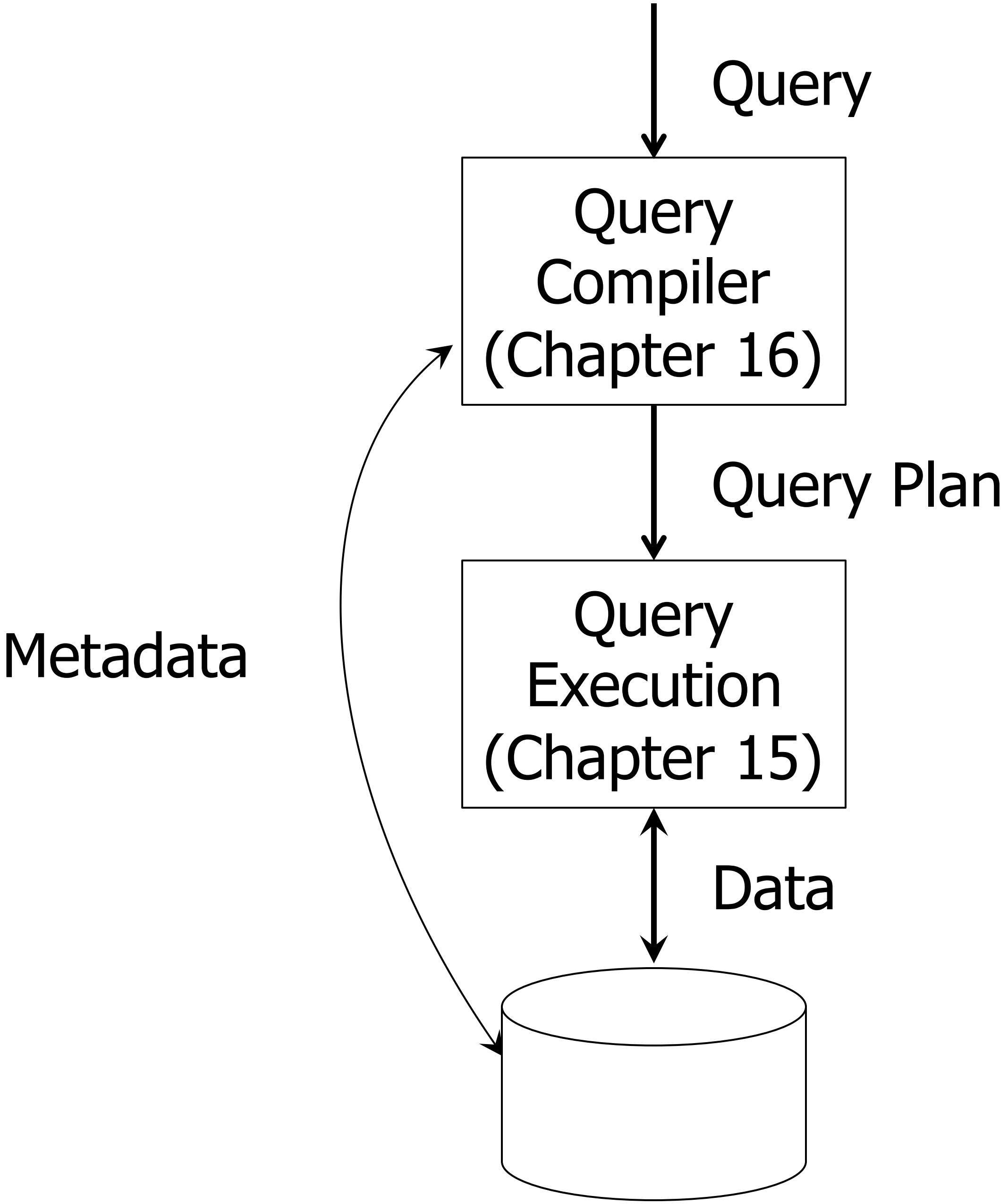
Query Execution



Query

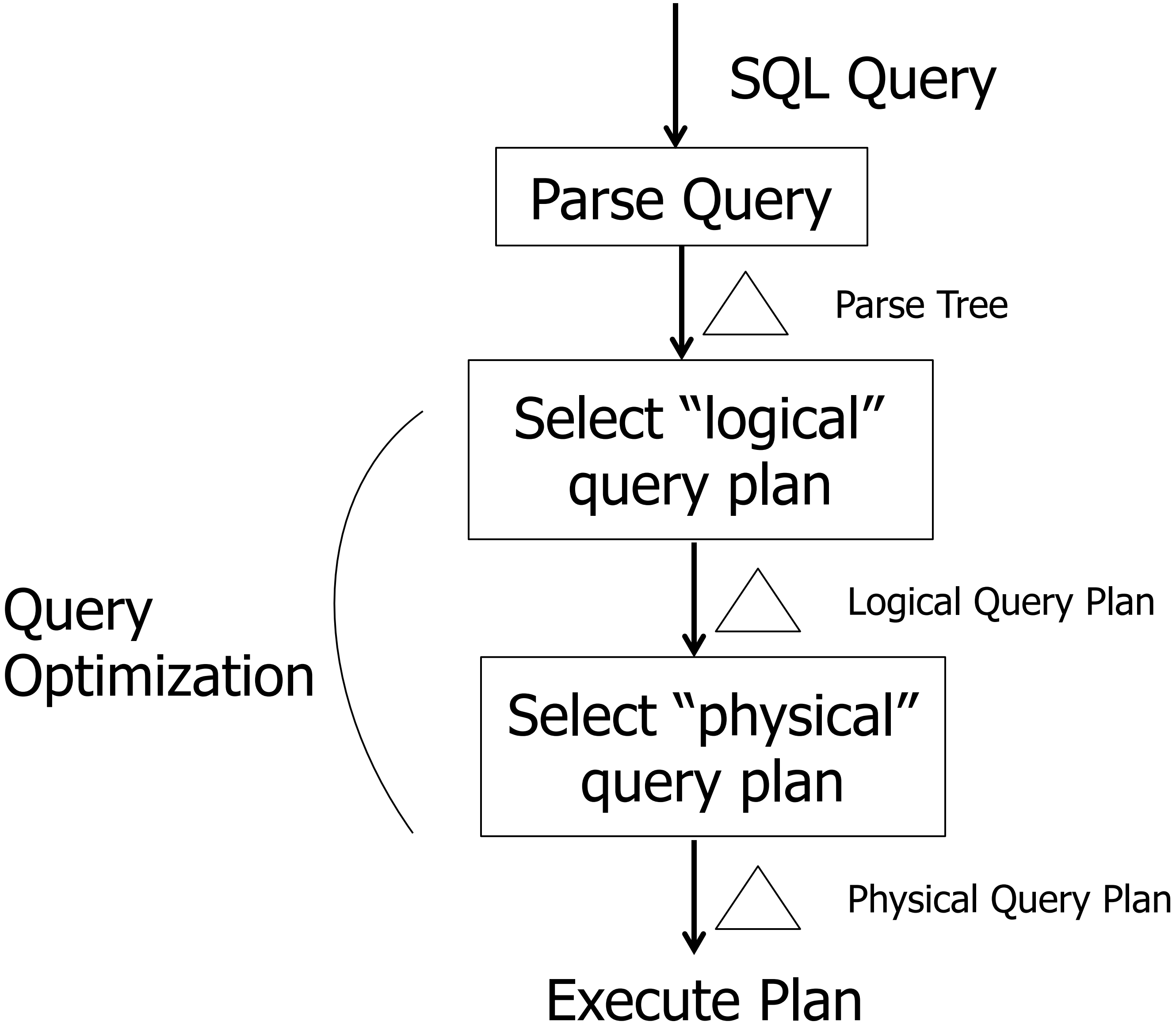
Query Compiler (Chapter 16)

Query Plan

Query Execution (Chapter 15)

Data

Metadata



SQL Query

Parse Query

Parse Tree

Select "logical" query plan

Logical Query Plan

Select "physical" query plan

Physical Query Plan

Execute Plan

Query Optimization

Plan

- Mostly we consider various algorithms to implement the JOIN operator.
- Then, we look at other operators.

COST: # of disk blocks read (or written) to execute (we ignore CPU costs).

Example   R1 $\bowtie$ R2 over common attribute C

Number of tuples: 10000 (R1), 5000 (R2)

Size of each tuple = 1/10 block

Memory available = 101 blocks

Assume tuples packed into blocks.

→ Metric: # of IOs

(ignoring writing of result)

Options

- Transformations: R1 $\bowtie$ R2, R2 $\bowtie$ R1

- Join algorithms:
  - Nested loops (Iteration)
  - Merge join
  - Join with index
  - Hash join

Nested Loop Join

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

Nested Loop Join (Block wise)

```
for each block X of R1 do
  for each block Y of R2 do
      Read X, Y.
      Take join of tuples in X and Y
```

Cost = 1000 + 1000 (500) = 501k

Better: Read R1 in Chunks

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

Cost = 1000 + 10 (500) = 6000.

If whole of R1 can fit in main memory, then known as One-Pass Join Algorithm

Even Better: Reverse Join Order

Reverse join order: $R2 \bowtie R1$

Total = 500 + 5 (1000) = 5500 IOs

Options

- Transformations: R1 $\bowtie$ R2, R2 $\bowtie$ R1

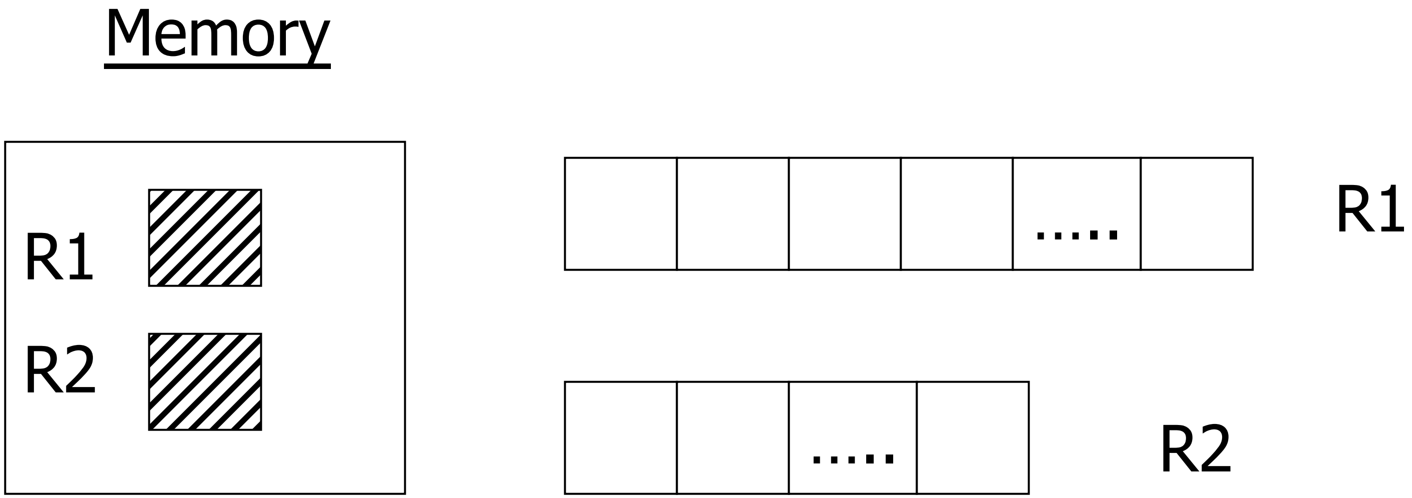
- Join algorithms:
  - Nested loops (Iteration)
  - Merge join
  - Join with index
  - Hash join

Merge Join

```
SORT R1 and R2 (if not already sorted)
$i \leftarrow 1$; $j \leftarrow 1$;
While $(i \le T(R1)) \wedge (j \le T(R2))$ do
        if R1{ i }.C = R2{ j }.C
            outputTuples        (ignoring details here)
        elseif R1{ i }.C > R2{ j }.C
            $j \leftarrow j+1$
        elseif R1{ i }.C < R2{ j }.C
            $i \leftarrow i+1$
```

Cost of Merge Join

- Both R1, R2 ordered by C

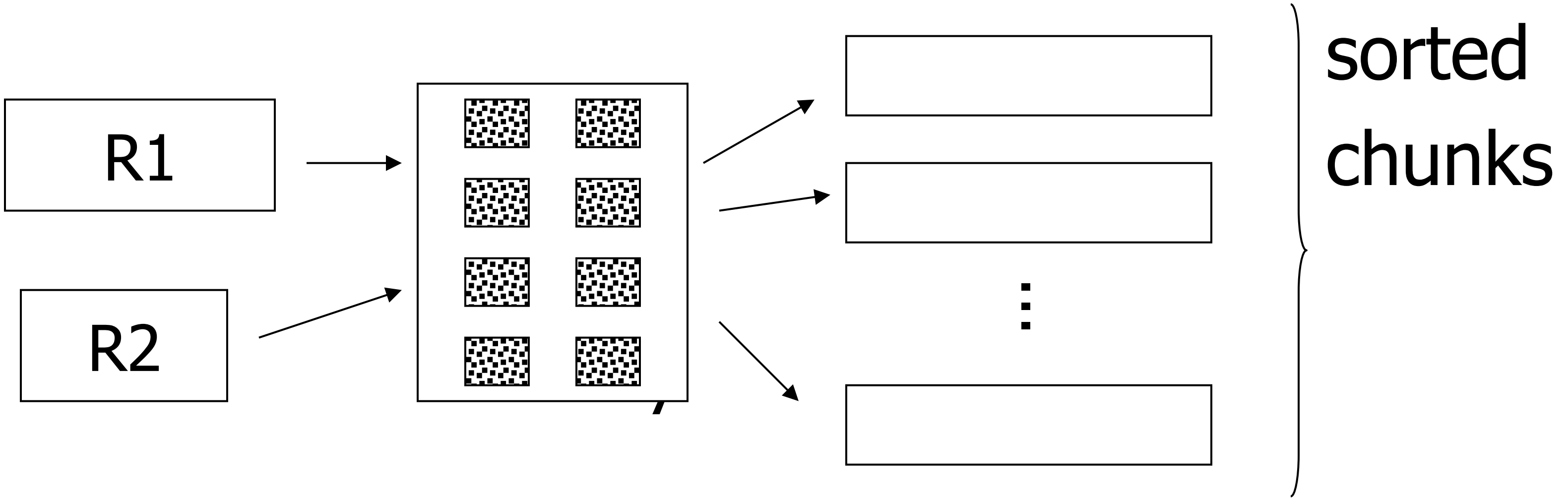


Cost: 1000 + 500 = 1,500 IOs

How to Sort a Table – 1

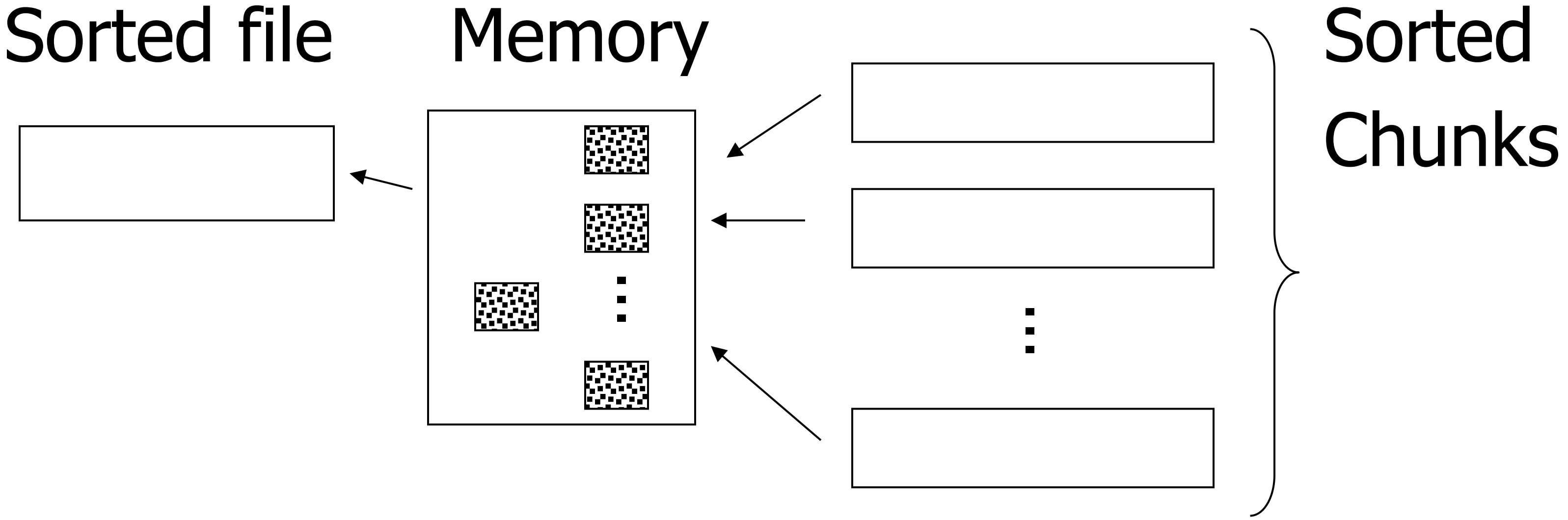
(i) For each 100 blk chunk of R:

- Read chunk
- Sort in memory
- Write to disk



How to Sort a Table – 2.

(ii) Read all chunks + merge + write out



How to Sort a Table – 3.

Cost for 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 Cost (with Sorting)

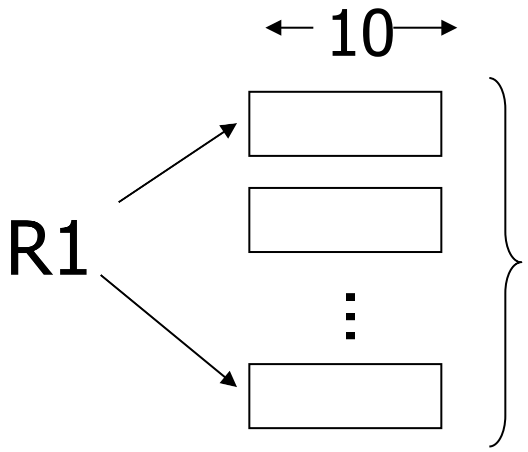
Total cost = sort cost + join cost
= 6,000 + 1,500 = 7,500 IOs

Cost is linear in sizes.

Thus, will beat nested-loop for large sizes.

Memory Requirements for Merge Sort

E.g: Say I have 10 memory blocks



100 chunks ⇒ to merge, need
100 blocks!

In general:

Say M blocks in memory

x blocks for relation sort

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

chunks $\leq$ buffers available for merge

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

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

In our example

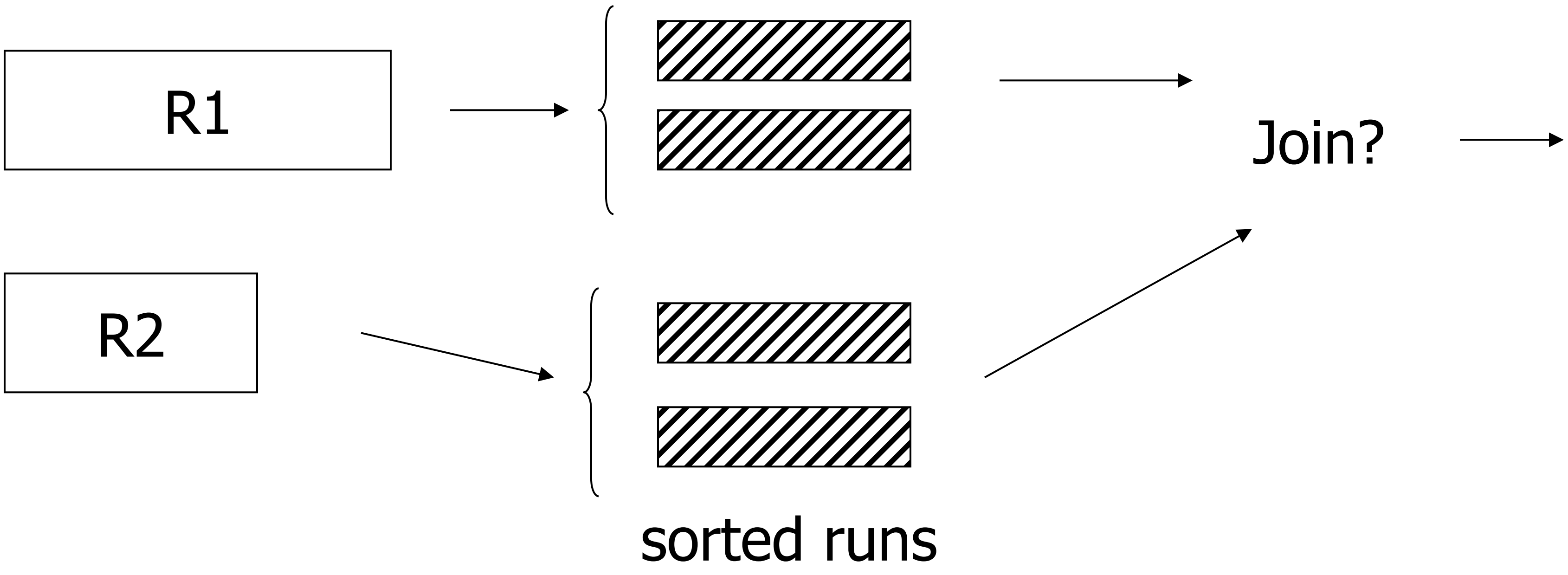
R1 is 1000 blocks, $M \geq 31.62$

R2 is 500 blocks, $M \geq 22.36$

Need at least 32 buffers

Improving Merge Join

Do we really need the fully sorted files?



Cost of improved merge join:

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

--> Memory requirement?

Options

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

Index Join

```
For each $s \in R2$ do
  [ $X \leftarrow$ index (R1, C, s.C)
      for each $r \in X$ do
          output r,s pair]
```

Assume R1.C index

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

Set X = set of rel tuples with attr = value

Index-Join cost

B(R2) + T(R2)(probe-cost + retrieving matches)

- Index is small enough in memory.

  Probe-cost = 0.

- Index is big.
  E.g., 201 blocks of B-Tree, 2-levels.
  Store root and half of the leaves in MM.
  Avg. probe-cost = 1/2.

Expected # of matching tuples

- If R1.C is key, then expected # of matches = 1
- If number of distinct C values = 5000 and T(R1) = 10,000, then expected # of matches = 2 (assuming uniformity).

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 → 4500
R1.C Index 5500
R2.C Index ________

Options

- Transformations: $R1 \bowtie R2$, $R2 \bowtie R1$

- Join algorithms:
  - Nested loops (Iteration)
  - Merge join
  - Join with index
  - Hash join

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) 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

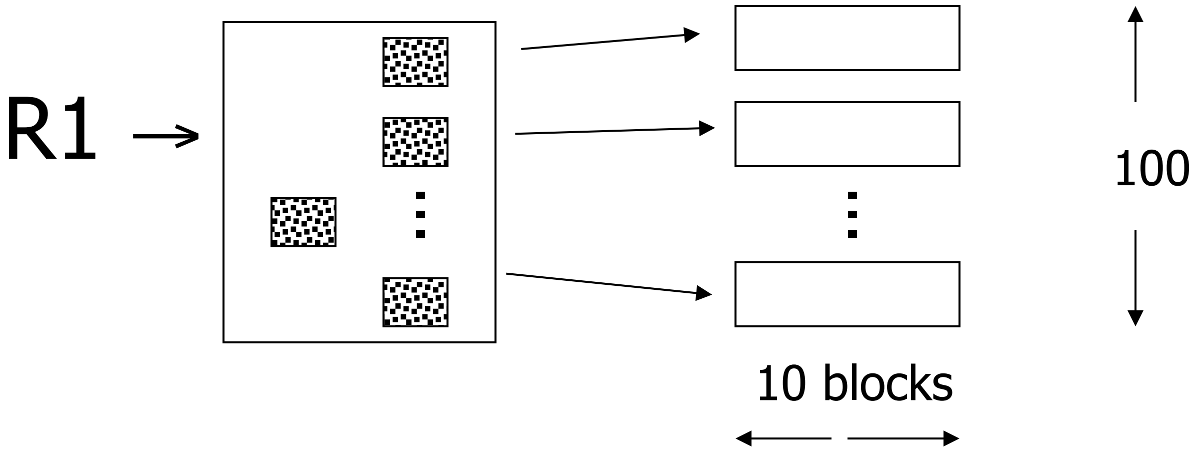
	R1	R2
Even	2 4 8	4 12 8 14
Odd:	3 5 9	5 3 13 11

Example 1(f) Hash Join

R1, R2 contiguous (un-ordered)

→ Use 100 buckets

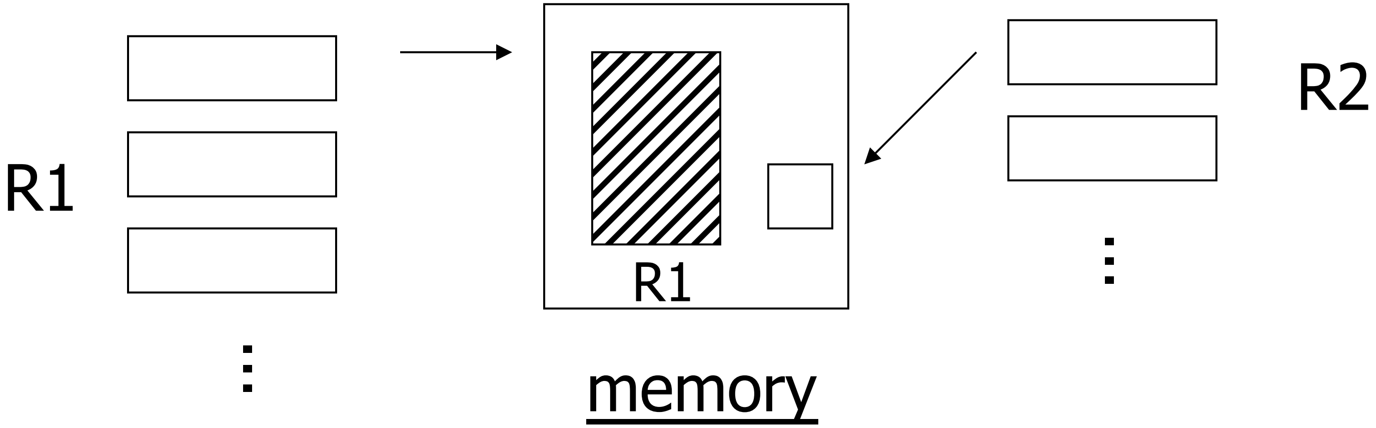
→ Read R1, hash, + write buckets



-> Same for R2

-> Read one R1 bucket; build memory hash table

-> Read corresponding R2 bucket + hash probe



Then repeat for all buckets

Hash Join 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

Let, M = number of memory buffers

B = number of R1 blocks

For efficient hashing:	Number of buckets $< M$
For efficient 2nd stage:	Size of a bucket $< M$

Size of R1 bucket = (B/number of buckets) $> B/M$

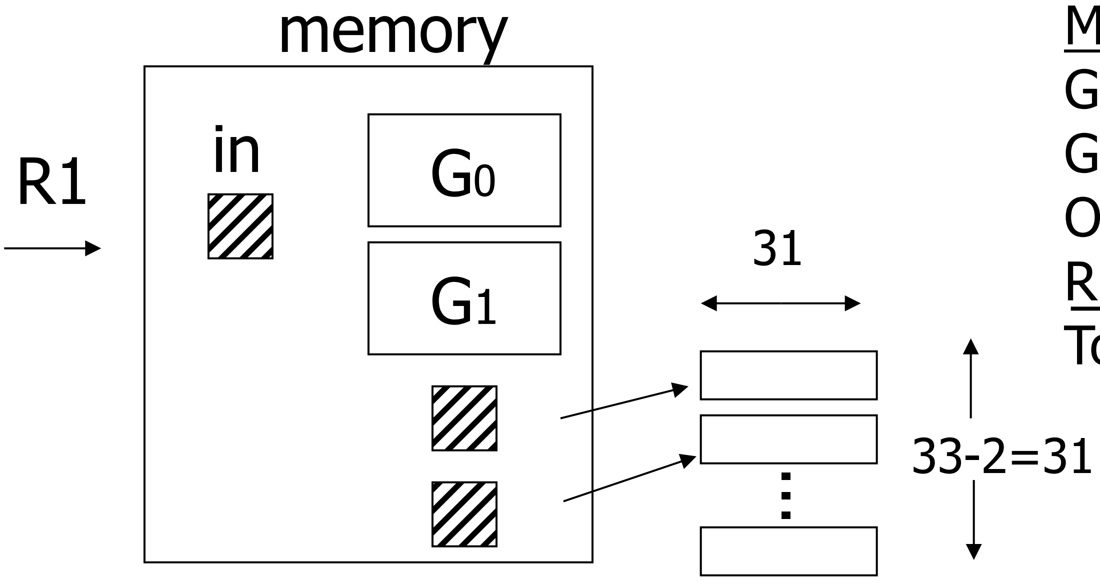
$B/M < M \Rightarrow M > \sqrt{B}$

Need B+1 total memory buffers

Trick #1 : keep some buckets in MM

E.g., 33 buckets; R1 Bucket = 31 blocks

Keep 2 R1 buckets in memory



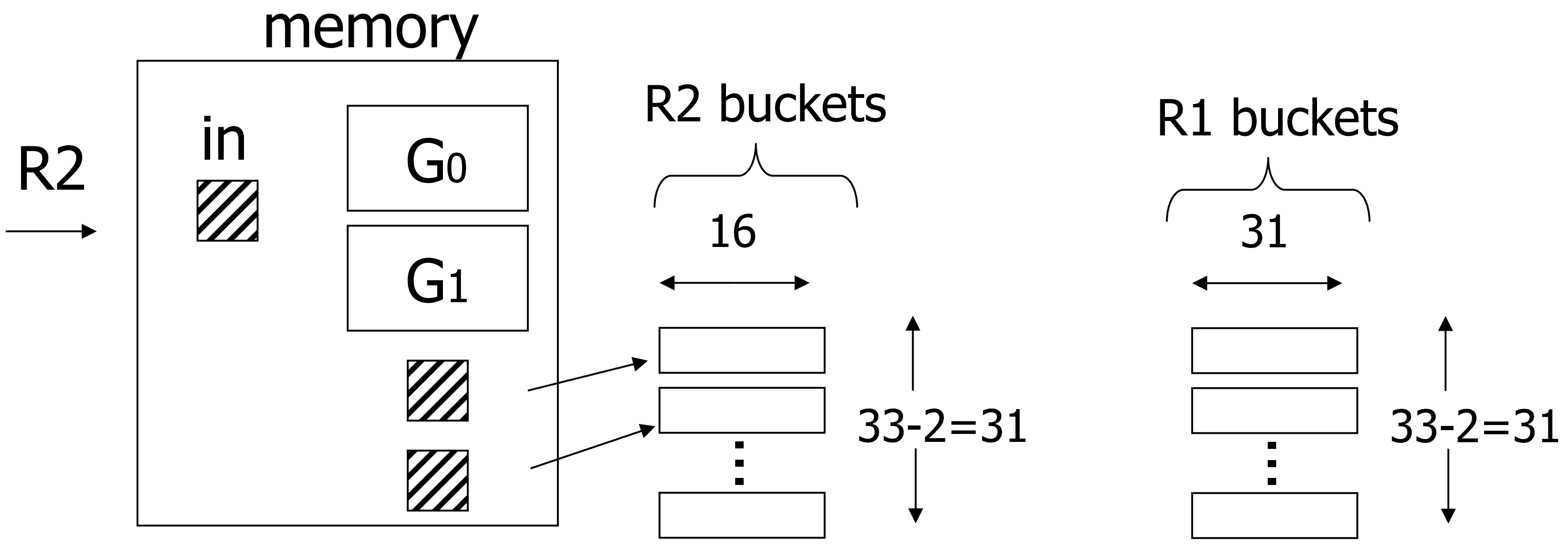
Memory use:	
G0	31 buffers
G1	31 buffers
Output	33-2 buffers
R1 input	1
Total	94 buffers

6 buffers to spare!!

called hybrid hash-join

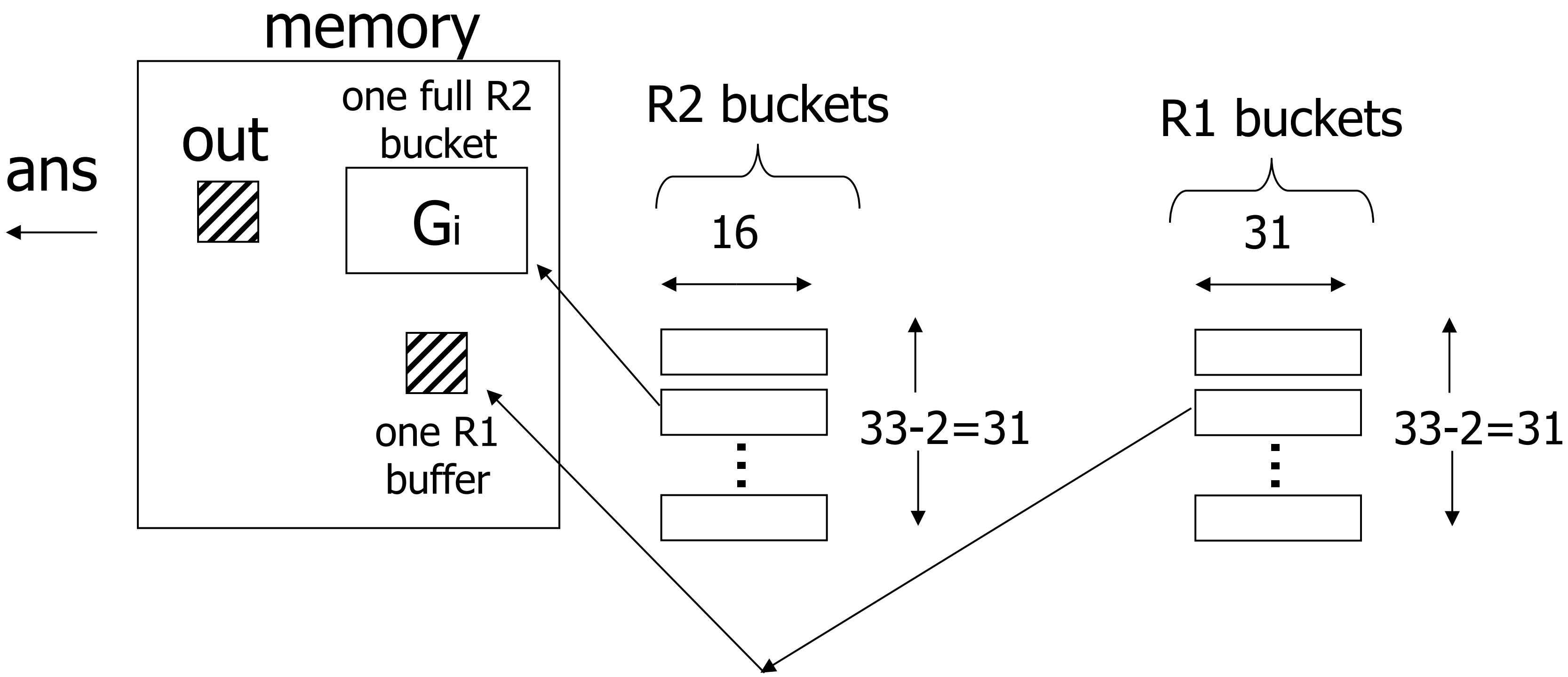
Next: Bucketize R2

- R2 buckets =500/33= 16 blocks
- Two of the R2 buckets joined immediately with G0,G1



Finally: Join remaining buckets

- for each bucket pair:
  - read one of the buckets into memory
  - join with second bucket

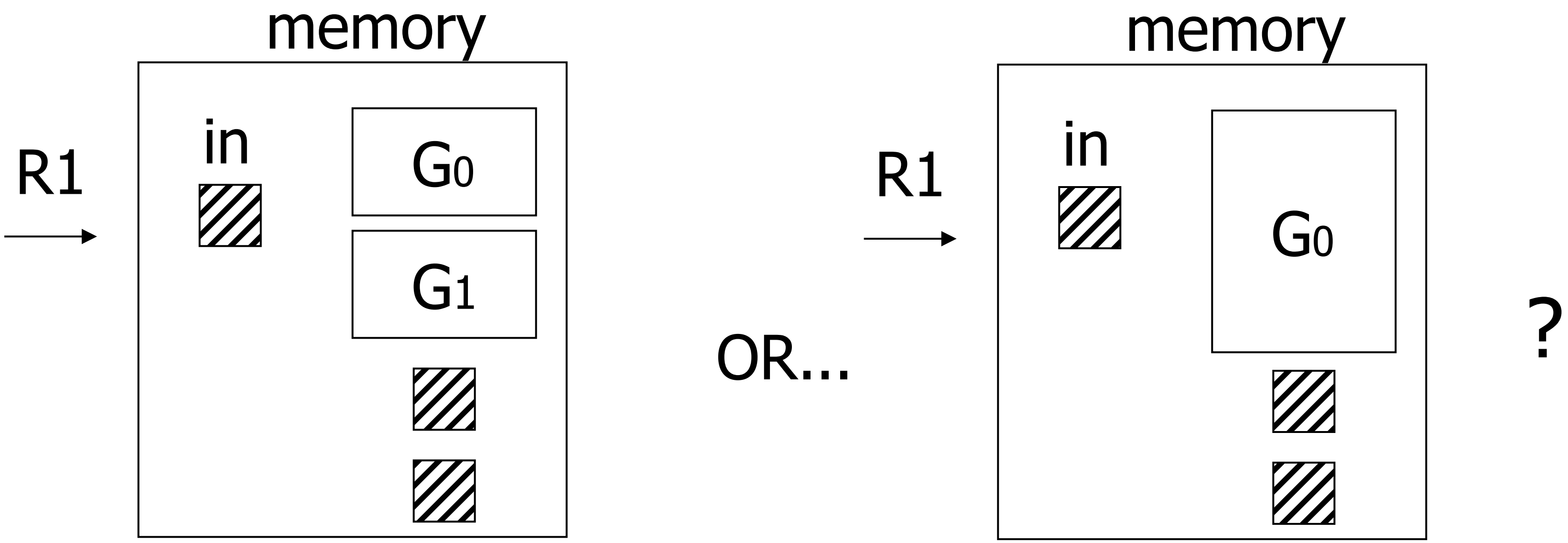


Cost

- Bucketize R1 = 1000+31×31=1961
- To bucketize R2, only write 31 buckets:
  so, cost = 500+31×16=996
- To compare join (2 buckets already done)
  read 31×31+31×16=1457

Total cost = 1961+996+1457 = 4414

How many buckets in memory?



See textbook for answer…

Trick #2: Store only <val, ptr> pairs in buckets

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

- To illustrate cost computation, assume:
  - 100 <val,ptr> pairs/block
  - expected number of result tuples is 100
- Build hash table for R2 in memory
  5000 tuples → 5000/100 = 50 blocks
- Read R1 and match
- Read ~ 100 R2 tuples

Total cost =	Read R2:	500
	Read R1:	1000
	Get tuples:	100
		1600

So far:

contiguous

Iterate	5500
Merge join	1500
Sort+merge joint	7500
R1.C index	5500
R2.C index	_____
Build R.C index	_____
Build S.C index	_____
Hash join	4500+
with trick,R1 first	4414
with trick,R2 first	_____
Hash join, pointers	1600

Summary

- Iteration ok for "small" relations (relative to memory size)
- If sorted, use merge-join.
- If index available, consider index-join.
- For equi-join, use hash join.
- Else, use sort + merge-join.

[Above, just guidelines]

Other Operations

- Duplicate Elimination, Aggregation
  - Keep one group/copy in memory
  - Use efficient main-memory data structure.

- Union, Intersection, Minus:
  - Either “one-pass”, or if MM is too small, use sorting, hashing, or index-based strategies.