CS4432: Database Systems II

Lecture #16

Join Processing Algorithms

Professor Elke A. Rundensteiner

Join : $R1 \bowtie R2$, $R2 \bowtie R1$

- Iteration Join
- Sort-Merge Join
- Index-Join
- Hash Join

Factors that affect performance

(1) Tuples of relation stored physically together?

(2) Buffer-aware data movement

(3) Relations sorted by join attribute?

(3) Indexes exist?

Example R1 ⋈ R2 over common attribute C

T(R1) = 10,000

T(R2) = 5,000

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

Memory available = 101 blocks

→ Metric: # of IOs

(ignoring writing of result)

Join : $R1 \bowtie R2$, $R2 \bowtie R1$

- Iteration (nested loops)
- Merge join
- Join with index
- Hash join

- Iteration join (nested-loop join)

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

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

Cost: for each R1 tuple:

[Read 1 R1 tuple + Read R2]

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

- 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

$$\text{Total} = \frac{10{,}000}{1{,}000} \times (1000+5000) = 60{,}000 \text{ IOs}$$

Does Reverse Join Order Matter?

- Reverse join order: R2 ⋈ R1

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

5 x 11,000 = 55,000 IOs

Example Iteration Join R2 ⋈ R1

- Relations contiguous

Cost

For each R2 chunk:

Read chunk: 100 IOs

Read R1: 1000 IOs

~~1,100~~

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

So far

not contiguous

Iterate R2 ⋈ R1 50,010,000

Be memory access aware:

Iterate R2 ⋈ R1 55,000

---

contiguous

Blocked-access:
Iterate R2 ⋈ R1 5,500

Join : R1 ⋈ R2, R2 ⋈ R1

- Iteration (nested loops)
- Sort-Merge join
- Join with index
- Hash join

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]

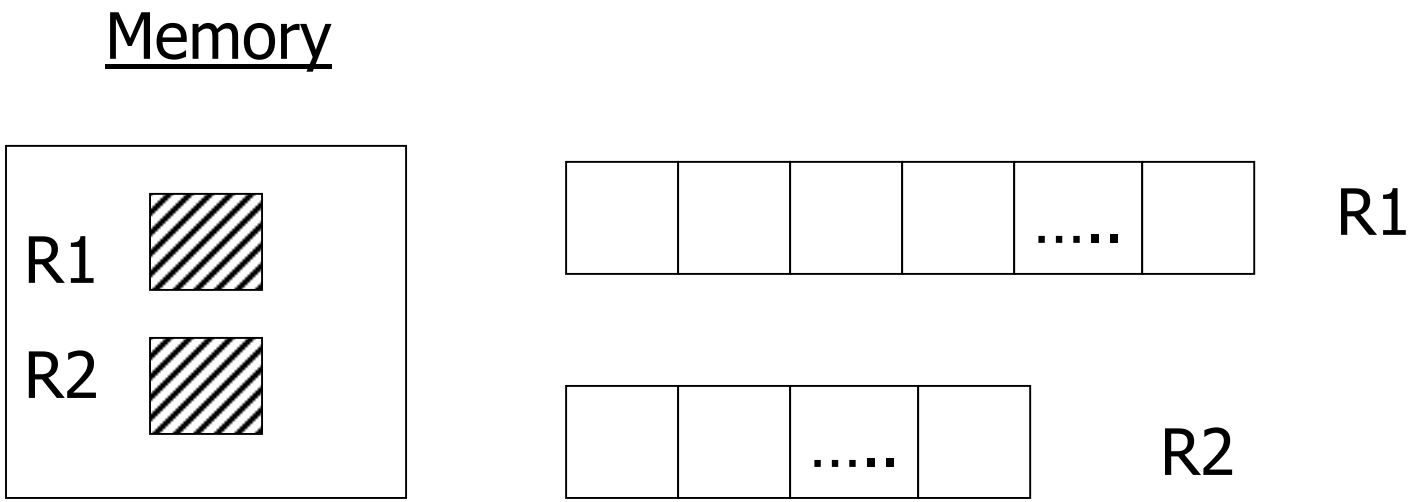
i $\leftarrow$ i+1]

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

Example : Merge Join

- Both R1, R2 ordered by C; relations contiguous



Total cost: Read R1 cost + read R2 cost

= 1000 + 500 = 1,500 IOs

Example Merge Join

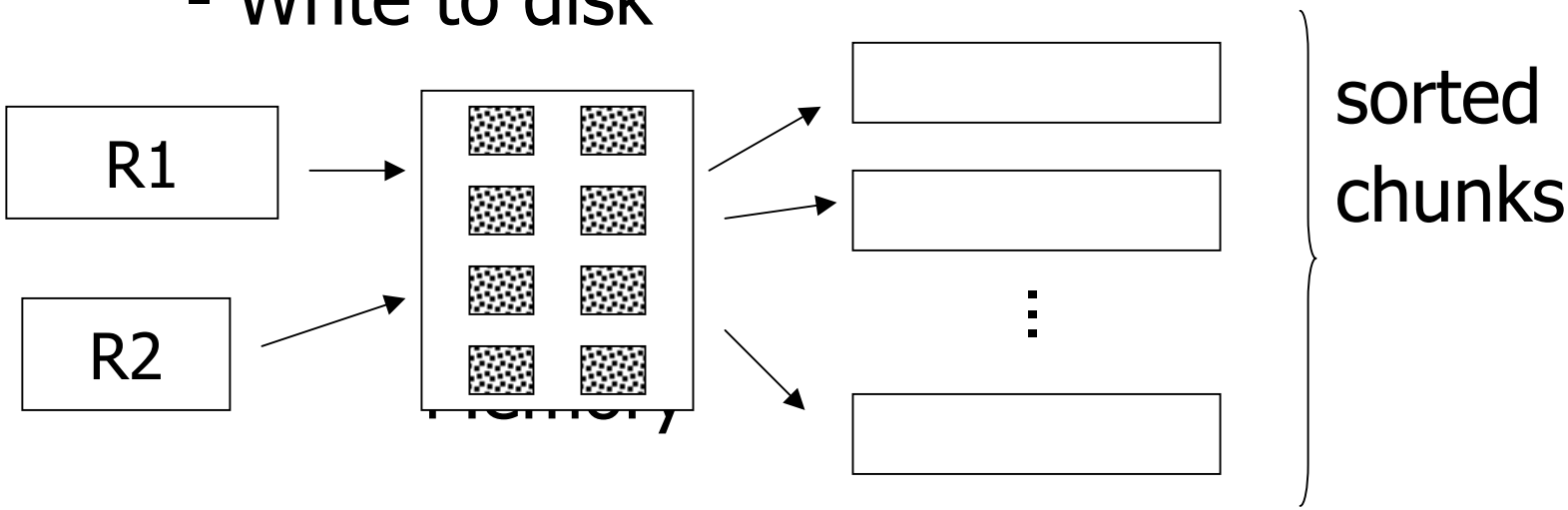
- R1, R2 not ordered, but contiguous

--> Need to sort R1, R2 first…. HOW?

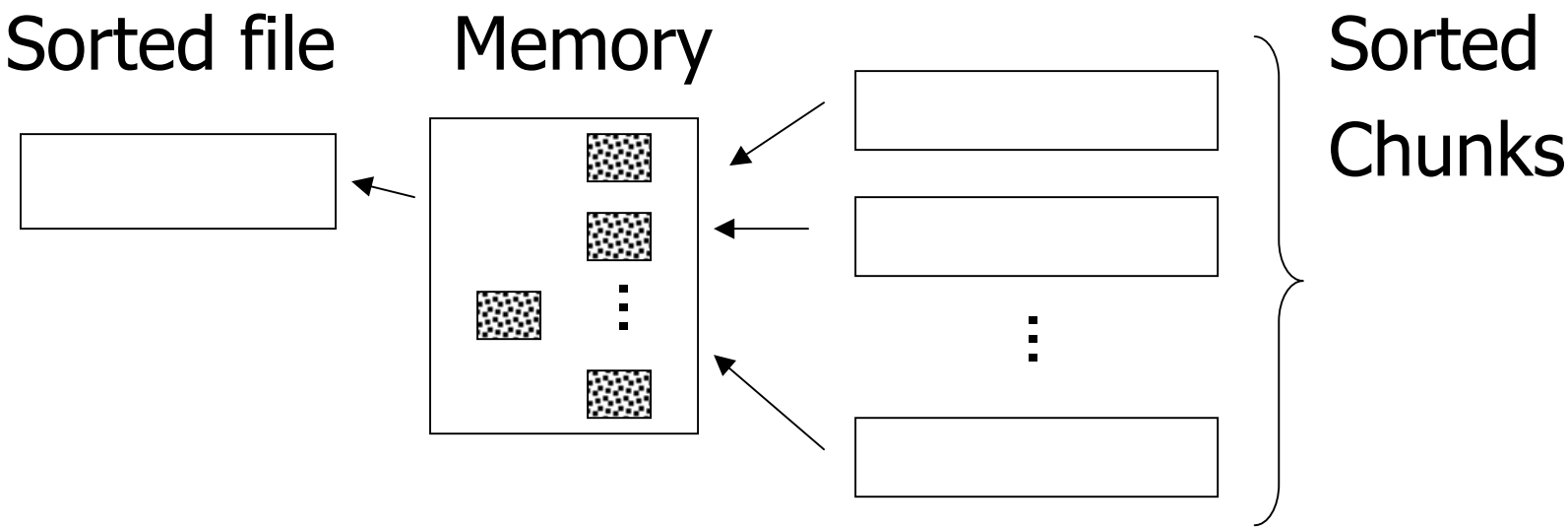
One way to sort: Merge Sort

(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 x 1,000 = 4,000

Sort cost R2: 4 x 500 = 2,000

Example Merge Join

R1,R2 contiguous, but unordered

Total cost = sort cost + join cost

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

Comparison ?

Iteration cost = 5,500 IOs
Merge cost = 7,500 IOs

Conclusion : so merge join does not pay off?

True sometimes? or always ???

Next a case

Where sort-merge join beats the iteration join.

For : 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!