Query Execution

Database Systems

Kevin C.C. Chang

Concepts You Will Learn

- Logical/physical operators
- Cost parameters and sorting
- One-pass algorithms
  - Nested-loop join
- Two-pass algorithms
  - Sort-merge join
  - Hash-based join
- Index-based algorithms
  - Index-based join

Why Do We Learn This?

The Big Picture: Where We Are

Relational | ***NonRelational***

Database Systems | Toolkits

Structured

SemiStructured

Untructured

Query Language

Data/Query Processing

Data Access

Transaction Management

Data Acquisition

Data Modeling

Relational Databases

- SQL
- Relational Algebra
- Query Optimization
- Query Execution
- Indexing
- Concurrency Control
- Logging Recovery

XML Databases

NoSQL Databases

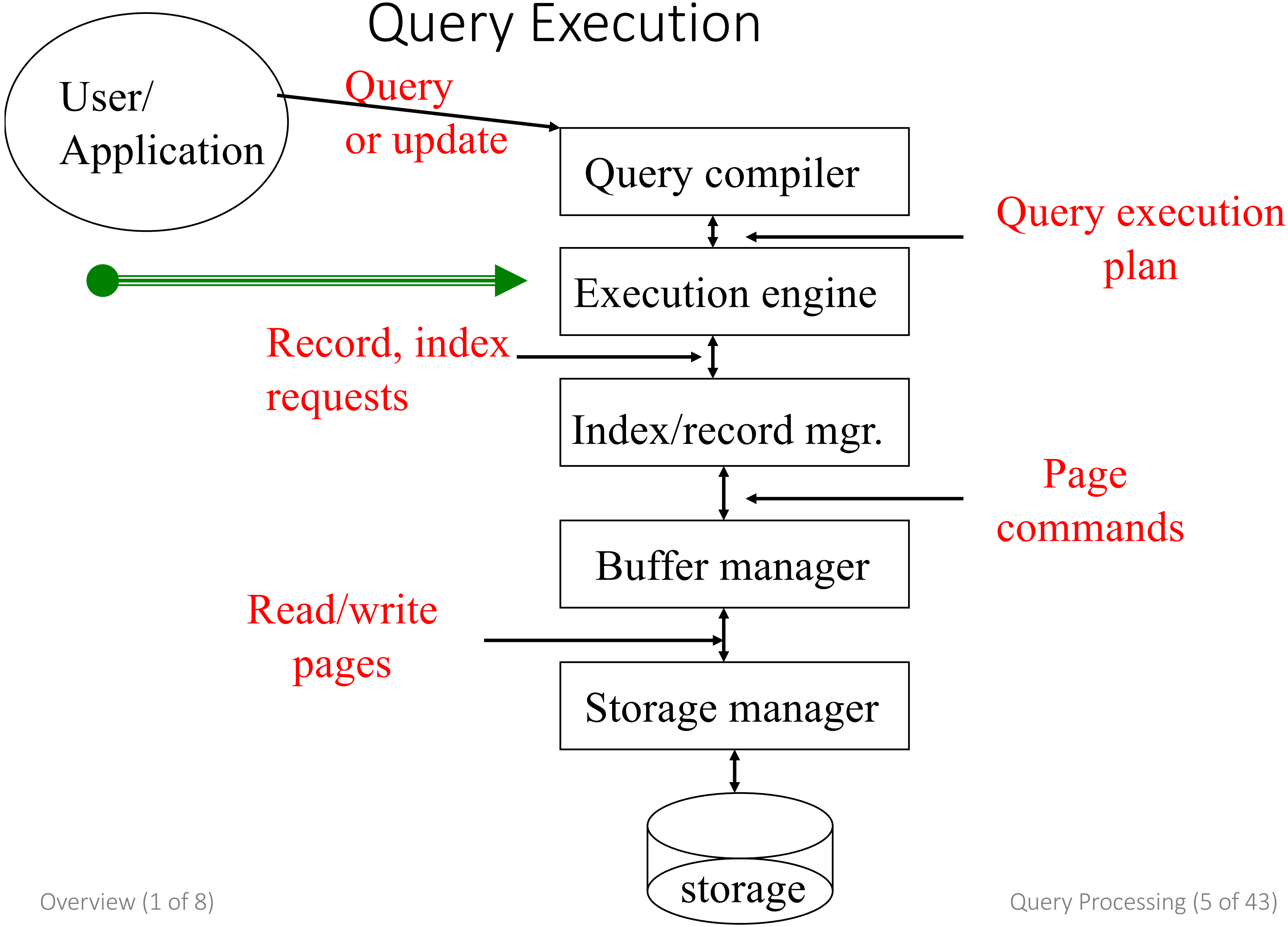
Map Reduce (Parallel)

Storm (Stream)

Information Extraction

ER → Relational Model

Query Execution



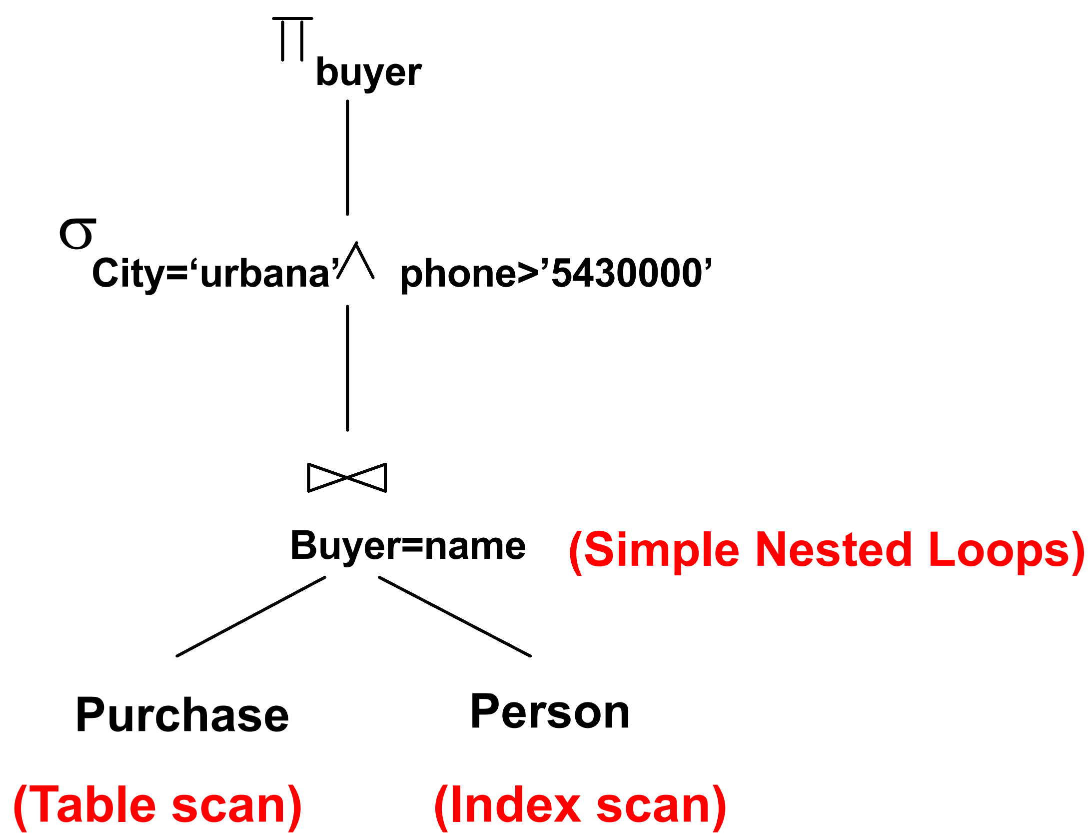
Overview

Logical v.s. Physical Operators

- Logical operators
  - *what* they do
  - e.g., union, selection, project, join, grouping
- Physical operators
  - *how* they do it
  - e.g., nested loop join, sort-merge join, hash join, index join

Query Execution Plans

```
SELECT   S.sname
FROM     Purchase P, Person Q
WHERE  P.buyer=Q.name AND
       Q.city='urbana' AND
       Q.phone > '5430000'
```



Query Plan:

- logical tree
- implementation choice at every node
- scheduling of operations.

Some operators are from relational algebra, and others (e.g., scan, group) are not.

How do We Combine Operations?

The iterator model.

- Each operation is implemented by 3 functions:
  - Open: sets up the data structures and performs initializations
  - GetNext: returns the next tuple of the result.
  - Close: ends the operations. Cleans up the data structures.
- Enables pipelining!

Cost Parameters

- Cost parameters
  - M = number of blocks that fit in main memory
  - B(R) = number of blocks holding R
  - T(R) = number of tuples in R
  - V(R,a) = number of distinct values of the attribute a
- Estimating the cost:
  - Important in optimization (next lecture)
  - Compute I/O cost only
  - We compute the cost to *read* the tables
  - We don’t compute the cost to *write* the result (because pipelining)

Sorting

- Two pass multi-way merge sort
- Step 1:
  - Read M blocks at a time, sort, write
  - Result: have runs of length M on disk
- Step 2:
  - Merge M-1 at a time, write to disk
  - Result: have runs of length $M(M-1) \approx M^2$
- Cost: $3B(R)$, Assumption: $B(R) \leq M^2$

Scanning Tables

- The table is *clustered* (I.e. blocks consists only of records from this table):
  - Table-scan: if we know where the blocks are
  - Index scan: if we have index to find the blocks
- The table is unclustered (e.g. its records are placed on blocks with other tables)
  - May need one read for each record

Cost of the Scan Operator

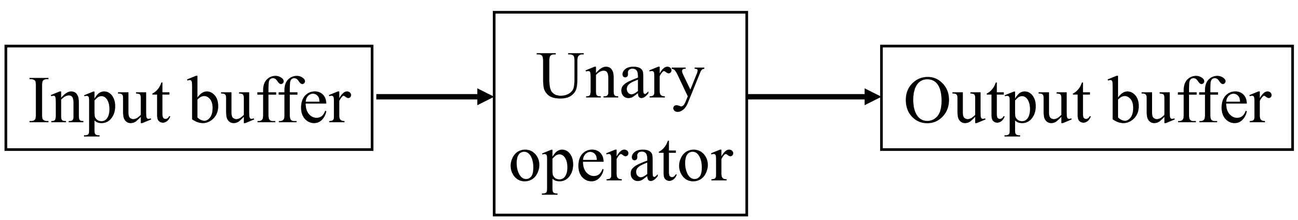
- Clustered relation:
  - Table scan: B(R); to sort: 3B(R)
  - Index scan: B(R); to sort: B(R) or 3B(R)
- Unclustered relation
  - T(R); to sort: T(R) + 2B(R)

One-Pass Algorithms

One-pass Algorithms

Selection σ(R), projection Π(R)

- Both are *tuple-at-a-Time* algorithms
- Cost: B(R)



One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep a dictionary in memory:
  - balanced search tree
  - hash table
  - etc
- Cost: $B(R)$
- Assumption: $B(\delta(R)) <= M$

One-pass Algorithms

Grouping: $\gamma_{\text{city, sum(price)}}$ (R)

- Need to keep a dictionary in memory
- Also store the sum(price) for each city
- Cost: B(R)
- Assumption: number of cities fits in memory

One-pass Algorithms

Binary operations: R ∩ S, R U S, R – S

- Assumption: min(B(R), B(S)) <= M
- Scan one table first, then the next, eliminate duplicates
- Cost: B(R)+B(S)

Nested Loop Joins

- Tuple-based nested loop $R \bowtie S$
- R=outer relation, S=inner relation

<u>for</u> each tuple r in R <u>do</u>
  <u>for</u> each tuple s in S <u>do</u>
    <u>if</u> r and s join <u>then</u> output (r,s)

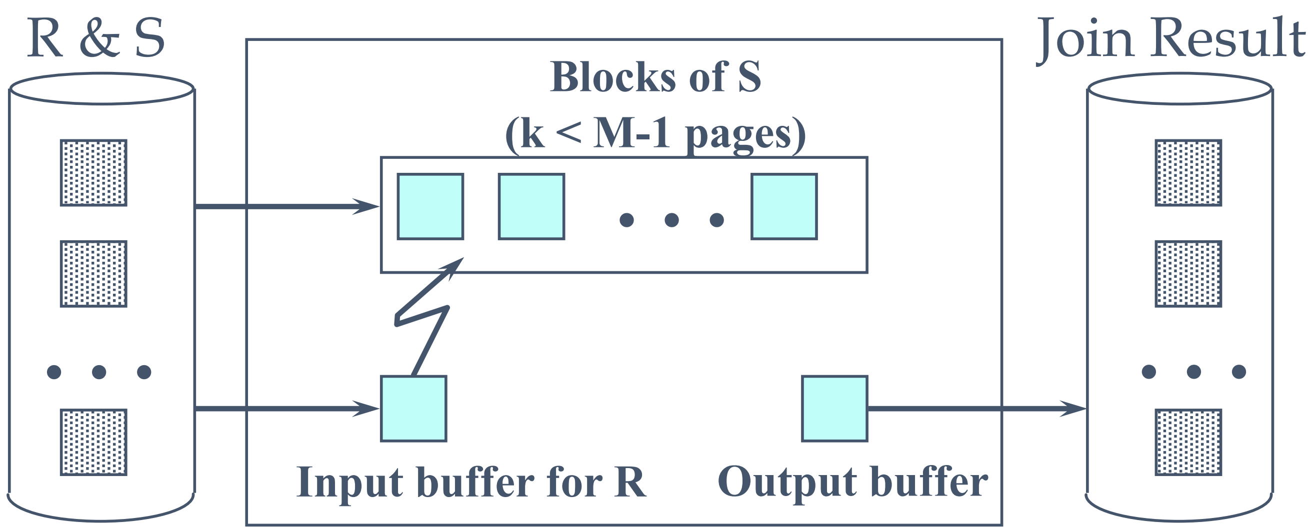
- Cost: T(R) T(S), sometimes T(R) B(S)

Nested Loop Joins

- Block-based Nested Loop Join

```
for each (M-1) blocks bs of S do
   for each block br of R do
         for each tuple s in bs do
              for each tuple r in br do
                   if r and s join then output(r,s)
```

Nested Loop Joins



Nested Loop Joins

- Block-based Nested Loop Join
- Cost:
  - Read S once: cost B(S)
  - Outer loop runs B(S)/(M-1) times, and each time need to read R: costs B(S)B(R)/(M-1)
  - Total cost: B(S) + B(S)B(R)/(M-1)
- Notice: it is better to iterate over the smaller relation first– i.e., S smaller

Two-Pass Algorithms

Two pass algorithms

Two-Pass Algorithms Based on Sorting

Duplicate elimination $\delta(R)$

- Simple idea: like sorting, but include no duplicates

- Step 1: sort runs of size M, write
  - Cost: 2B(R)
- Step 2: merge M-1 runs,

  but **include each tuple only once**
  - Cost: B(R)

- Total cost: 3B(R), Assumption: $B(R) <= M^2$

Q: What can sorting help? And, how?

- Selection?
- Projection?
- Set operations?
- Join?
- Duplicate elimination?
- Grouping?

Two-Pass Algorithms Based on Sorting

Grouping: $\gamma_{\text{city, sum(price)}}$ (R)

- Same as before: sort, then compute the sum(price) for each group
- As before: compute sum(price) during the merge phase.
- Total cost: 3B(R)
- Assumption: $B(R) <= M^2$

Two-Pass Algorithms Based on Sorting

Binary operations: $R \cap S$, $R \cup S$, $R - S$

- Idea: sort R, sort S, then do the right thing
- A closer look:
  - Step 1: split R into runs of size M, then split S into runs of size M. Cost: 2B(R) + 2B(S)
  - Step 2: merge *all* x runs from R; merge all y runs from S; ouput a tuple on a case by cases basis ($x + y \le M$)
- Total cost: 3B(R)+3B(S)
- Assumption: $B(R)+B(S) \le M^2$

Sort-Merge Join

Join $R \bowtie S$

- Start by sorting both R and S on the join attribute:
  - Cost: 4B(R)+4B(S) (because need to write to disk)
- Read both relations in sorted order, match tuples
  - Cost: B(R)+B(S)
- Difficulty: many tuples in R may match many in S
  - If at least one set of tuples fits in M, we are OK
  - Otherwise need nested loop, higher cost
- Total cost: 5B(R)+5B(S)
- Assumption: B(R) <= M^2, B(S) <= M^2

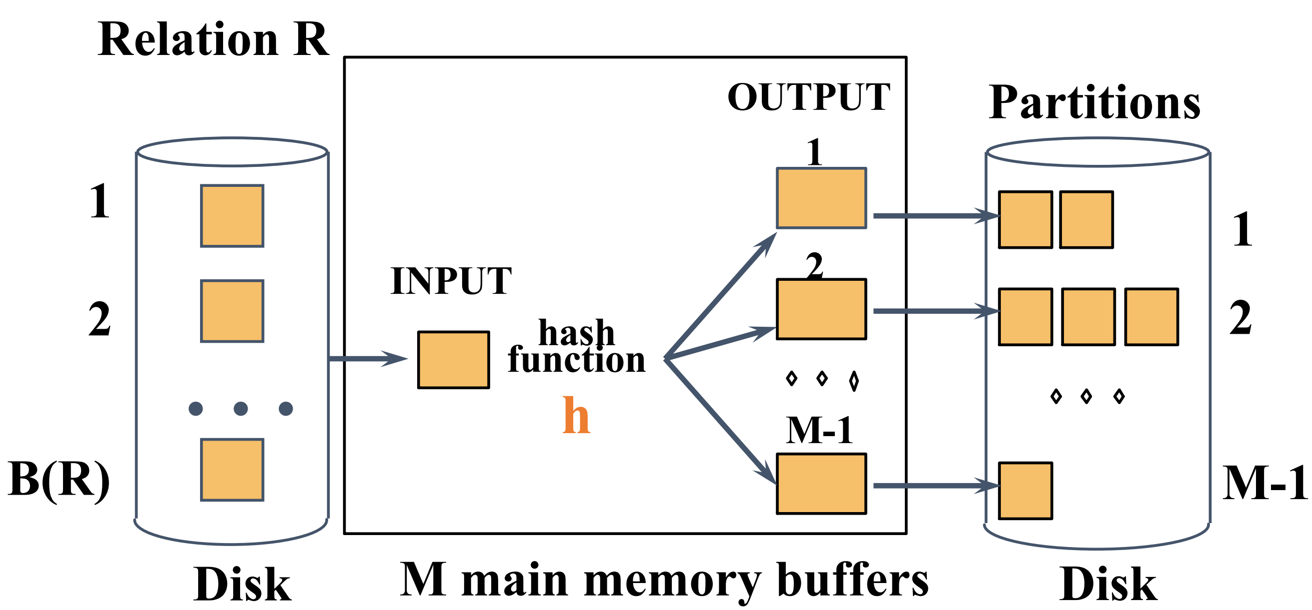
Q: Why is sorting-based “two” pass?

- Pass 1?

- Pass 2?

Two Pass Algorithms Based on Hashing

- Idea: partition a relation R into buckets, on disk
- Each bucket has size approx. B(R)/M



- Does each bucket fit in main memory ?
  - Yes if B(R)/M <= M,   i.e. B(R) <= M^2

Q: What can hashing help? And, how?

- Selection?
- Projection?
- Set operations?
- Join?
- Duplicate elimination?
- Grouping?

Hash Based Algorithms for δ

- Recall: $\delta(R) =$ duplicate elimination
- Step 1. Partition R into buckets
- Step 2. Apply δ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption:B(R) <= M^2

Hash Based Algorithms for γ

- Recall: γ(R) = grouping and aggregation
- Step 1. Partition R into buckets
- Step 2. Apply γ to each bucket (may read in main memory)

- Cost: 3B(R)
- Assumption: B(R) <= M^2

Hash-based Join

$R \bowtie S$

- Simple version: *main memory hash-based join*
  - Scan S, build buckets in main memory
  - Then scan R and join

- Requirement: min(B(R), B(S)) <= M

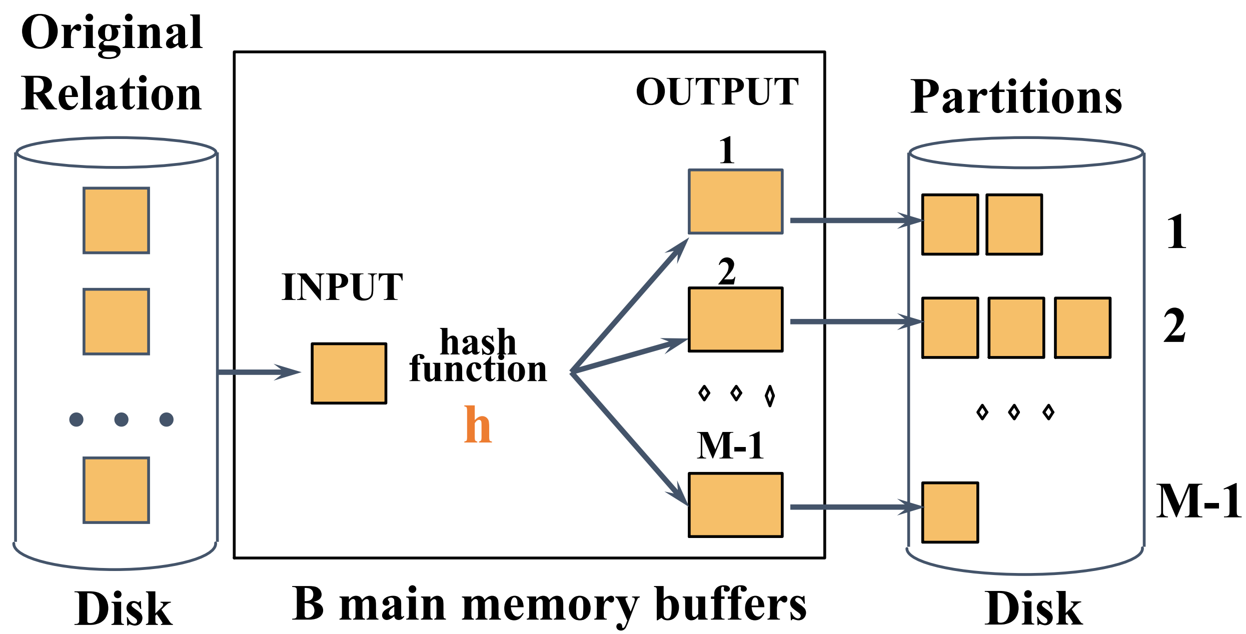
Partitioned Hash Join

$R \bowtie S$

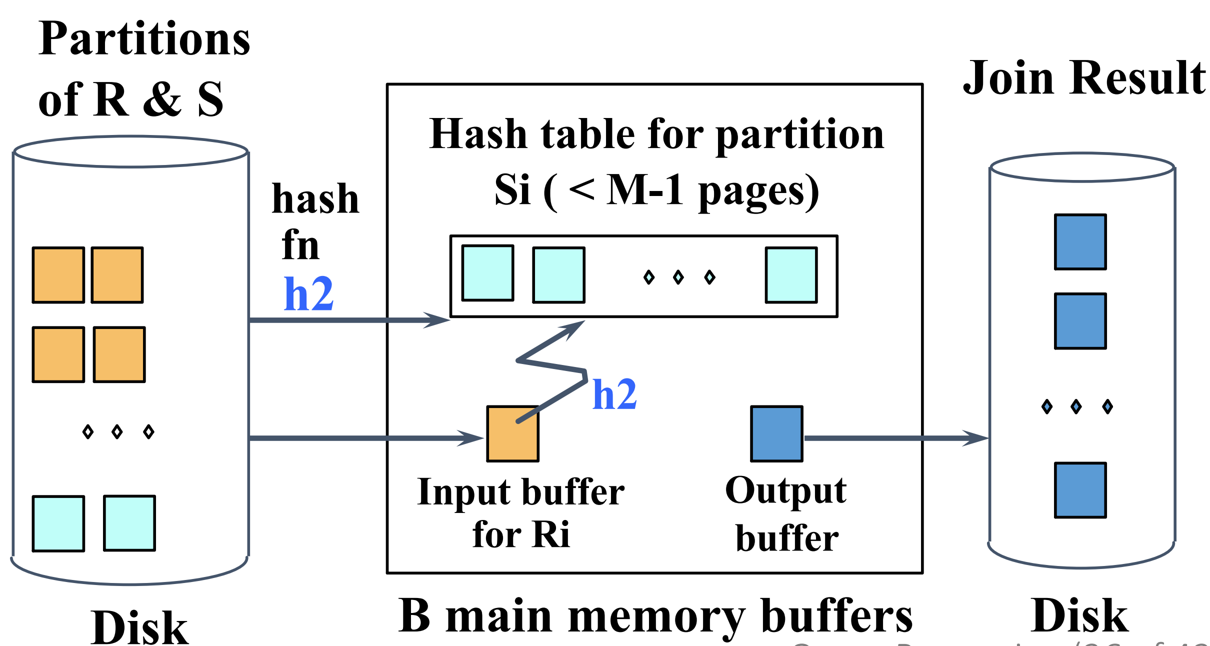
- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

Partitioned Hash-Join

- Partition both relations using hash fn h: R tuples in partition i will only match S tuples in partition i.



- Read in a partition of S, hash it using h2 (<> h!). Scan matching partition of R, search for matches.



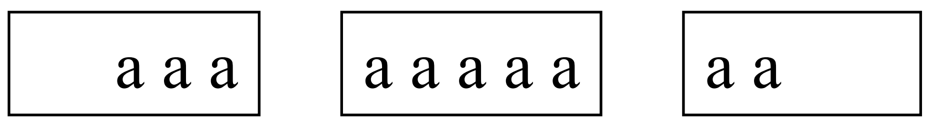
Partitioned Hash Join

- Cost: $3B(R) + 3B(S)$
- Assumption: At least one full bucket of the smaller rel must fit in memory: $\min(B(R), B(S)) \le M^2$

Index-based Algorithms (Zero-Pass)

Indexed Based Algorithms

- In a clustered index all tuples with the same value of the key are clustered on as few blocks as possible.



Index Based Selection

- Selection on equality: $\sigma_{a=v}(R)$
- Clustered index on a: cost B(R)/V(R,a)
- Unclustered index on a: cost T(R)/V(R,a)

Index Based Selection

- Example: B(R) = 2000, T(R) = 100,000, V(R, a) = 20, compute the cost of $\sigma_{a=v}(R)$
- Cost of table scan:
  - If R is clustered: B(R) = 2000 I/Os
  - If R is unclustered: T(R) = 100,000 I/Os
- Cost of index based selection:
  - If index is clustered: B(R)/V(R,a) = 100
  - If index is unclustered: T(R)/V(R,a) = 5000
- Notice: when V(R,a) is small, then unclustered index is useless

Index Based Join

$R \bowtie S$

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S
- Assume R is clustered. Cost:
    - If index is clustered: B(R) + T(R)B(S)/V(S,a)
    - If index is unclustered: B(R) + T(R)T(S)/V(S,a)

Index Based Join

- Assume both R and S have a sorted index (B+ tree) on the join attribute
- Then perform a merge join (called zig-zag join)
- Cost: B(R) + B(S)