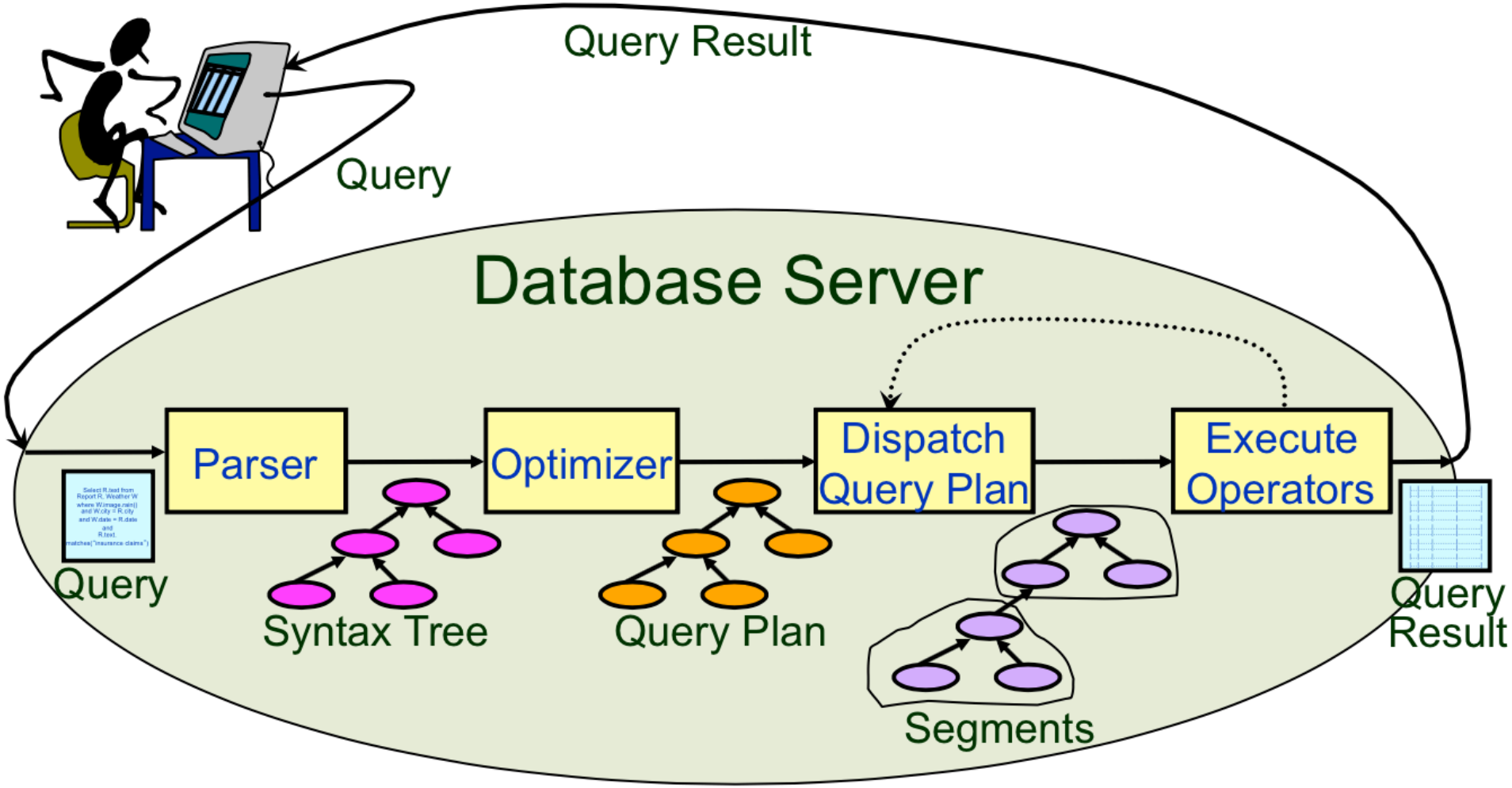
Projections

Chapter 12 and 14

Query Execution Life-Cycle

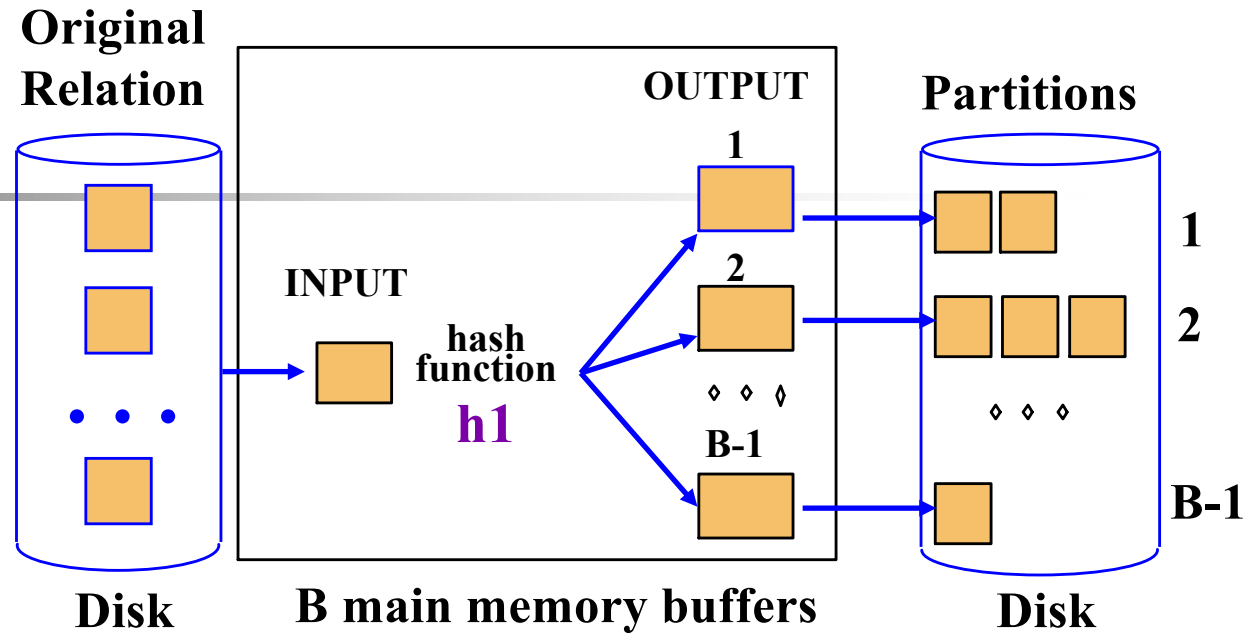


Projection

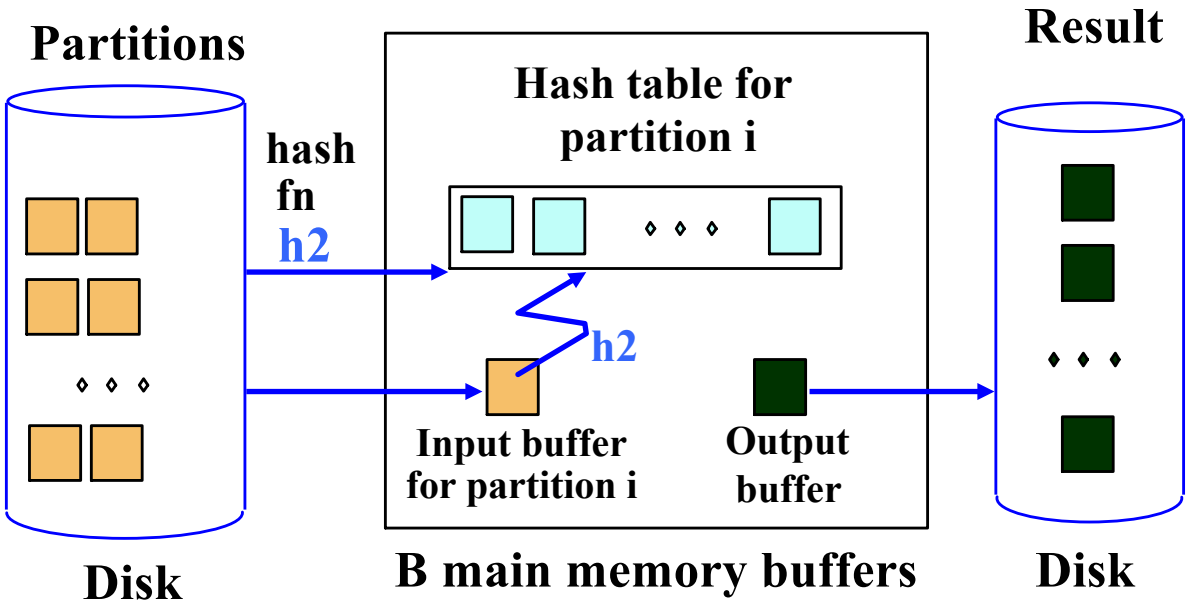
- Select R.a, R.d
  - Straightforward implementation!
- Select DISTINCT R.a, R.d
  - Remove attributes
  - Eliminate duplicates
- Algorithms for Projection DISTINCT:
  - Sorting: Sort on all the projected attributes
    - Pass 0: eliminate unwanted fields. Tuples in the sorted-runs may be smaller
    - Eliminate duplicates in the merge pass & sort
  - Hashing: Two phases
    - Partitioning
    - Duplicate elimination

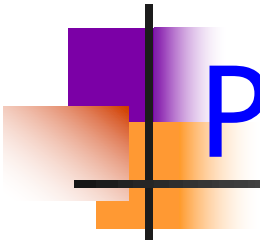
Hashing

Hash on projected Attributes to Eliminate duplicates



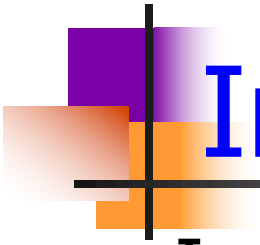
Recursively apply hash-based projection technique to handle partition overflow problem





Projection

- Sort-based approach for SELECT DISTINCT
  - better handling of skew
  - result is sorted
  - Thus, more commonly used than hash-based approach



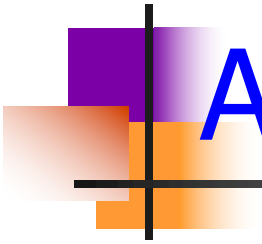
Index-only Scans

- Index-only scan can be much more efficient if
  - Projection attributes subset of index attributes
- Apply projection techniques to data entries (much smaller!)
- For handling SELECT DISTINCT, an additional optimization possible if the ordered (i.e., tree) index contains all projection attributes as prefix of search key:
  - Retrieve index data entries in order (no sorting necessary)
  - Discard unwanted fields
  - Compare adjacent entries to eliminate duplicates (if required)

Set Operations

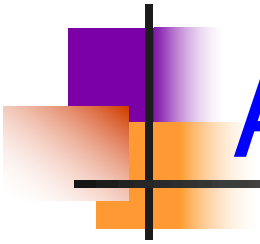
- ∩ and × special cases of join
- ∪ and − similar; we'll do ∪
  - Both require duplicate elimination
- Duplicate elimination algorithms for ∪:

1. Sorting:
   - Sort both relations (on all attributes).
   - Merge sorted relations eliminating duplicates.
2. Hashing:
   - Partition R and S
   - Build hash table for R_i.
   - Probe with tuples in S_i, add to table if not a duplicate



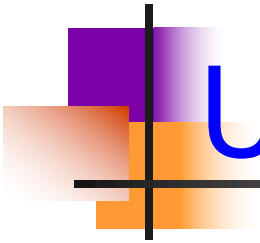
Aggregates

- Sorting Approach
  - Sort on GROUP BY attributes (if any)
  - Scan sorted tuples, computing running aggregate
    - Min, Max
    - Count
    - Sum
    - Average: compute from sum and count
  - During scan, when the group by attribute changes (e.g., 2, 2, 2, **3**), output aggregate result



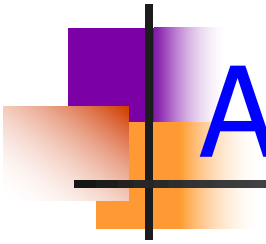
Aggregates

- Hashing Approach
  - Hash on GROUP BY attributes (if any)
    - Hash entry: grouped attributes + running aggregate
  - Scan tuples, probe hash table, update hash entry
  - Scan hash table, and output each hash entry
- Cost: Scan relation!



Using an Index for Aggregation

- Usually, index is not useful for aggregation operators.
- But sometimes it is:
  - When the search key contains all the relevant attributes: index-only scan may be feasible, rather than fetching all the data records.
  - If the GROUP BY attribute list is a prefix of the search key, we can retrieve data records in the order required for the grouping operation and thereby avoid the sorting step.



Announcements

- Optional Exercises:
  - 12.1 (1-4), 12.3, 12.5
  - 13.1, 13.3
  - 14.1 (2, 3, 4, 6, 7, 8, 9, 10), 14