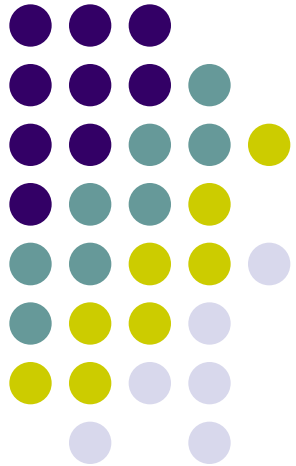
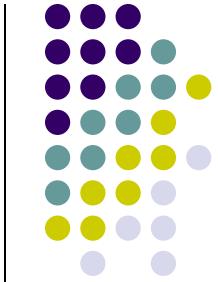
CMSC424: Database Design

Instructor: Amol Deshpande

amol@cs.umd.edu

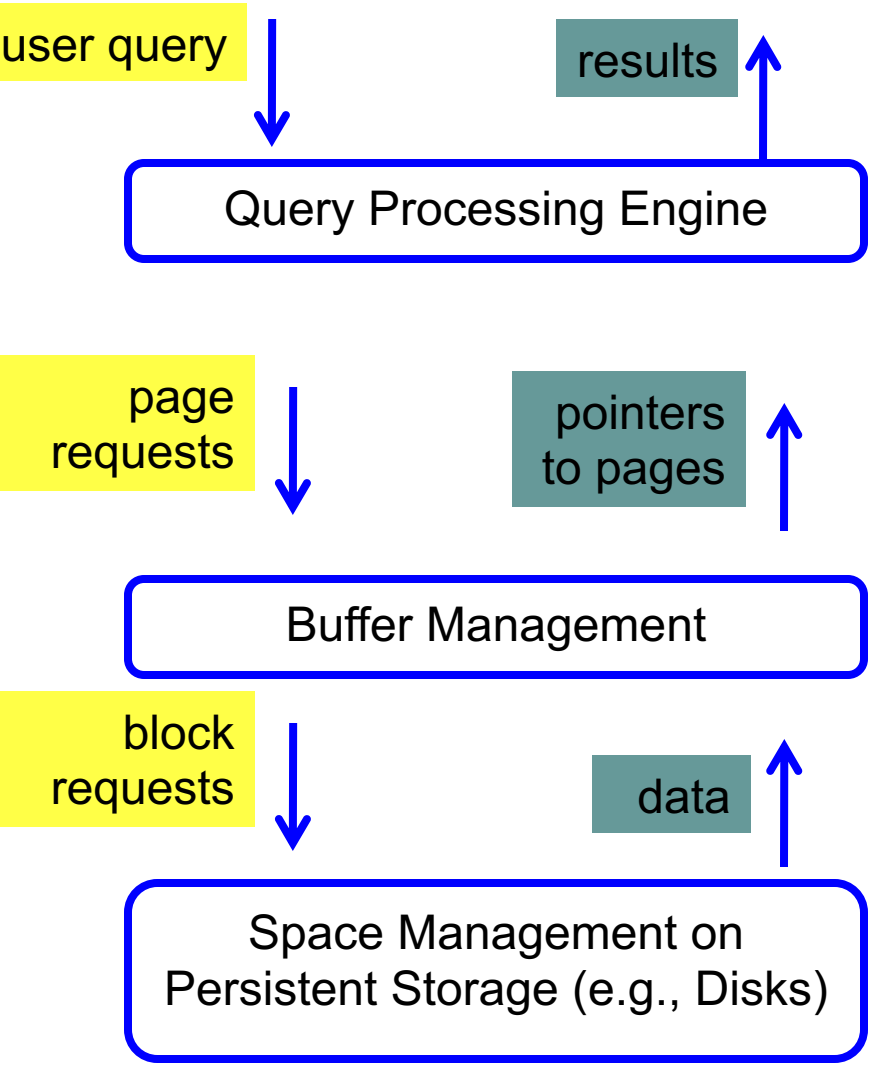


Spring 2020 – Online Instruction Plan



- Week 1: File Organization and Indexes
- Week 2 (Homework due Monday April 6):
  - Overview, Measures of Cost, Selections
  - Join Operation
  - Sorting, and Other Operators
- Week 3: Query Optimization; Transactions 1
- Week 4: Transactions 2
- Week 5: Parallel Database and MapReduce

Review: Query Processing/Storage

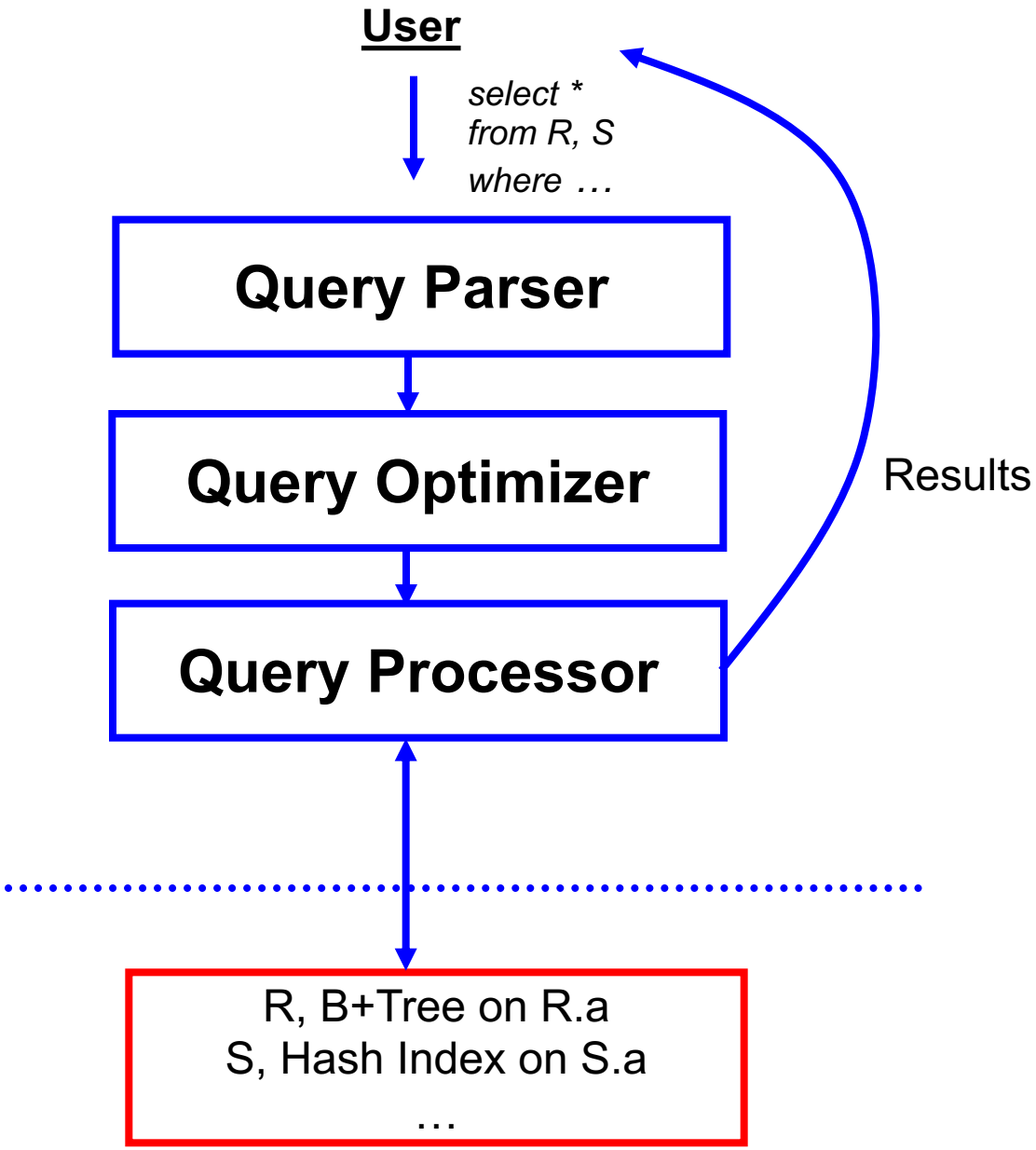


- Given a input user query, decide how to "execute" it
- Specify sequence of pages to be brought in memory
- Operate upon the tuples to produce results

- Bringing pages from disk to memory
- Managing the limited memory

- Storage hierarchy
- How are relations mapped to files?
- How are tuples mapped to disk blocks?

Getting Deeper into Query Processing

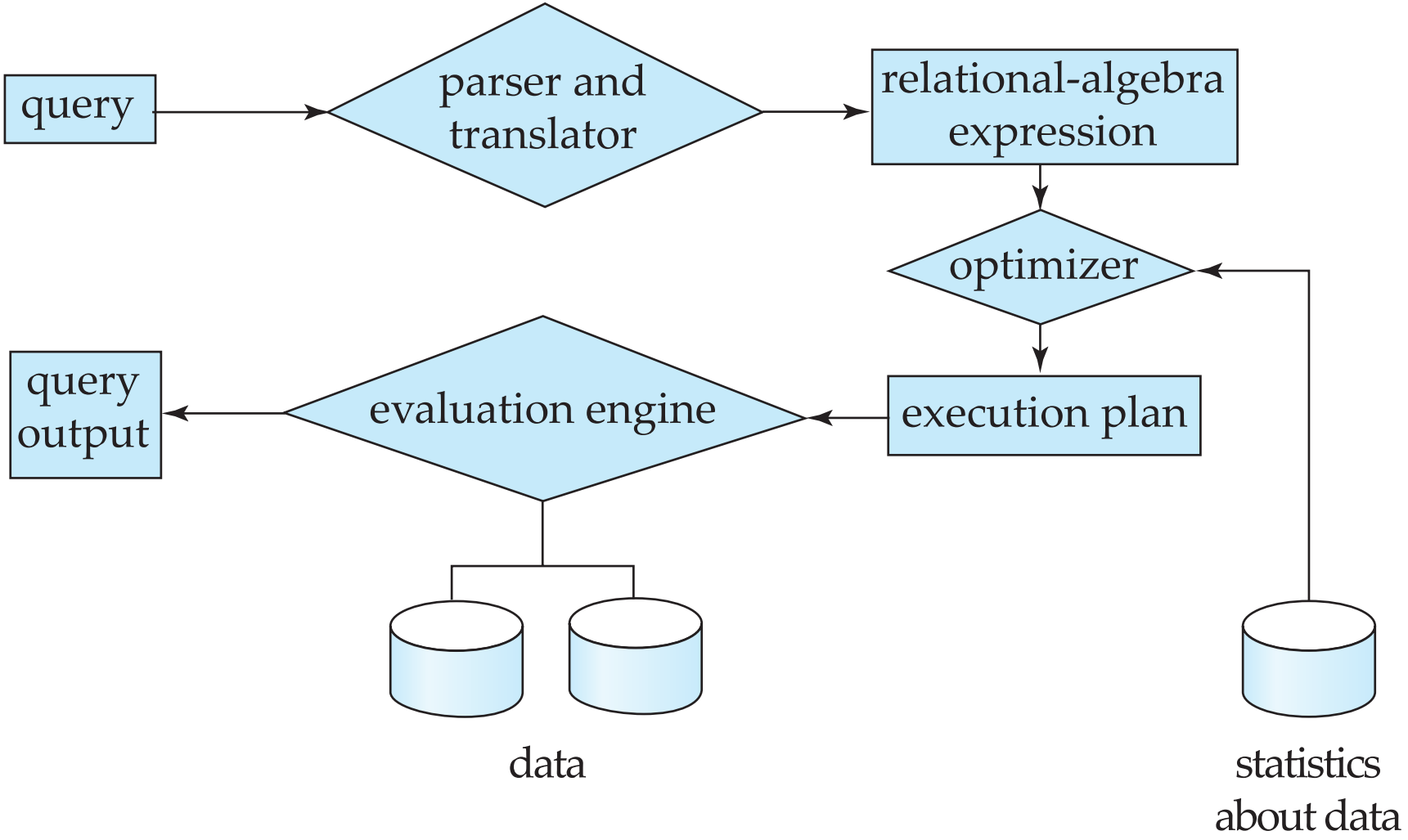


Resolve the references,
Syntax errors etc.
Converts the query to an internal format
relational algebra like

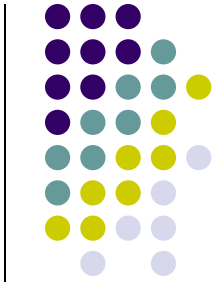
Find the *best* way to evaluate the query
Which index to use ?
What join method to use ?
…

Read the data from the files
Do the query processing
joins, selections, aggregates
…

Getting Deeper into Query Processing

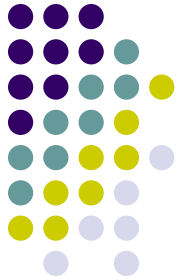


Cost Measures and Selections



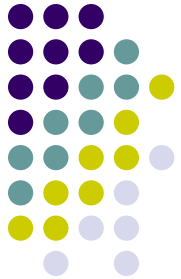
- Book Chapters
  - 12.1, 12.2, 12.3
- Key topics:
  - How to measure the ”cost” of an operation so we can compare alternatives?
  - Different ways to do a ”selection” operation (“where” clause) based on the properties of the predicates and the availability of indexes

“Cost”



- Complicated to compute, but very important to decide early on
  - Need to know what you are “optimizing” for
- Many competing factors in today’s computing environment
  - CPU Instructions
  - Disk I/Os
  - Network Usage – either peak or average (for distributed settings)
  - Memory Usage
  - Cache Misses
  - … and so on
- Want to pick the one (or combination) that’s actually a bottleneck
  - No sense in optimizing for “memory usage” if you have a TB of memory and a single disk
  - Can do combinations by doing a weighted sum: e.g., 10 * Memory + 50 * Disk I/Os

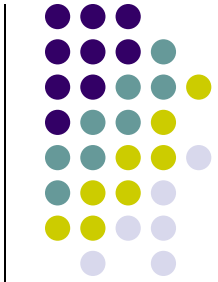
"Cost"



- We will focus on disk for simplicity:
  - Number of I/Os ?
    - Not sufficient
    - Number of seeks matters a lot… why ?
  - t_T – time to transfer one block
  - t_S – time for one seek
  - Cost for *b* block transfers plus *S* seeks

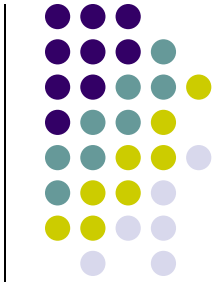
    $$b * t_T + S * t_S$$
  - Measured in *seconds*
- Real systems do take CPU cost into account

Selection Operation



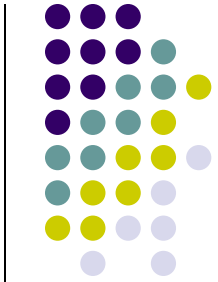
- select * from person where SSN = “123”
- Option 1: Sequential Scan
  - Read the relation start to end and look for “123”
    - Can always be used (not true for the other options)
  - Cost ?
    - *Let b_r = Number of relation blocks*
    - Then:
      - 1 seek and b_r block transfers
    - So:
      - $t_S + b_r * t_T$ *sec*
    - *Improvements:*
      - *If SSN is a key, then can stop when found*
        - *So on average, $b_r/2$ blocks accessed*

Selection Operation



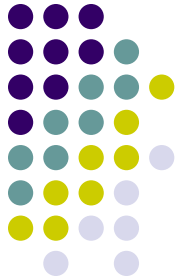
- select * from person where SSN = “123”
- Option 2 : Binary Search:
  - Pre-condition:
    - *The relation is sorted on SSN*
    - *Selection condition is an equality*
      - E.g. can’t apply to “*Name like ‘%424%’*”
  - Do binary search
    - Cost of finding the *first* tuple that matches
      - $\lceil \log_2(b_r) \rceil * (t_T + t_S)$
      - All I/Os are random, so need a seek for all
        - The last few are closeby, but we ignore such small effects
  - Not quite: What if 10000 tuples match the condition ?
    - Incurs additional cost

Selection Operation



- select * from person where SSN = “123”
- Option 3 : Use Index
  - Pre-condition:
    - *An appropriate index must exist*
  - Use the index
    - Find the first leaf page that contains the search key
    - Retrieve all the tuples that match by following the pointers
      - If primary index, the relation is sorted by the search key
        - Go to the relation and read blocks sequentially
      - If secondary index, must follow all pointers using the index

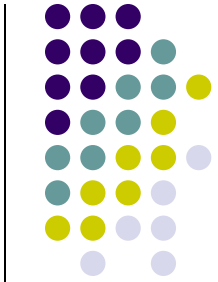
Selection w/ B+-Tree Indexes



	cost of finding the first leaf	cost of retrieving the tuples
primary index, candidate key, equality	$h_i * (t_T + t_S)$	$1 * (t_T + t_S)$
primary index, not a key, equality	$h_i * (t_T + t_S)$	$1 * (t_T + t_S) + (b - 1) * t_T$ *Note: primary == sorted* *b = number of pages that contain the matches*
secondary index, candidate key, equality	$h_i * (t_T + t_S)$	$1 * (t_T + t_S)$
secondary index, not a key, equality	$h_i * (t_T + t_S)$	$n * (t_T + t_S)$ *n = number of records that match* This can be bad

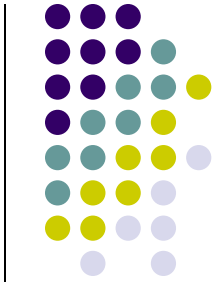
h_i = height of the index

Selection Operation



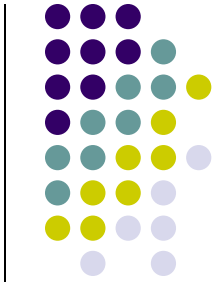
- Selections involving ranges
  - *select * from accounts where balance > 100000*
  - *select * from matches where matchdate between '10/20/06' and '10/30/06'*
  - Option 1: Sequential scan
  - Option 2: Using an appropriate index
    - Can't use hash indexes for this purpose
    - Cost formulas:
      - Range queries == "equality" on "non-key" attributes
      - So rows 3 and 5 in the preceding page

Selection Operation



- Complex selections
  - *Conjunctive:* *select * from accounts where balance > 100000 and SSN = "123"*
  - Disjunctive: *select * from accounts where balance > 100000 or SSN = "123"*
  - Option 1: Sequential scan
  - Option 2 *(Conjunctive only)*: Using an appropriate index *on one of the conditions*
    - E.g. Use SSN index to evaluate SSN = "123". Apply the second condtion to the tuples that match
    - Or do the other way around (if index on balance exists)
    - Which is better ?
  - Option 3 *(Conjunctive only)* : Choose a multi-key index
    - Not commonly available

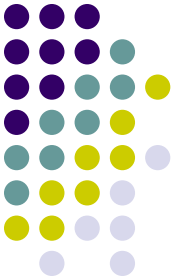
Selection Operation



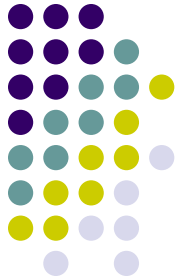
- Complex selections
  - *Conjunctive: select * from accounts where balance > 100000 and SSN = "123"*
  - Disjunctive: *select * from accounts where balance > 100000 or SSN = "123"*
  - Option 4: Conjunction or disjunction of *record identifiers*
    - Use indexes to find all RIDs that match each of the conditions
    - Do an intersection (for conjunction) or a union (for disjunction)
    - Sort the records and fetch them in one shot
    - Called "Index-ANDing" or "Index-ORing"
  - Heavily used in commercial systems

From the Book

	Algorithm	Cost	Reason
A1	Linear Search	$t_S + b_r * t_T$	One initial seek plus b_r block transfers, where b_r denotes the number of blocks in the file.
A1	Linear Search, Equality on Key	Average case $t_S + (b_r/2) * t_T$	Since at most one record satisfies condition, scan can be terminated as soon as the required record is found. In the worst case, b_r blocks transfers are still required.
A2	Primary B^+-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	(Where h_i denotes the height of the index.) Index lookup traverses the height of the tree plus one I/O to fetch the record; each of these I/O operations requires a seek and a block transfer.
A3	Primary B^+-tree Index, Equality on Nonkey	$h_i * (t_T + t_S) + b * t_T$	One seek for each level of the tree, one seek for the first block. Here b is the number of blocks containing records with the specified search key, all of which are read. These blocks are leaf blocks assumed to be stored sequentially (since it is a primary index) and don't require additional seeks.
A4	Secondary B^+-tree Index, Equality on Key	$(h_i + 1) * (t_T + t_S)$	This case is similar to primary index.
A4	Secondary B^+-tree Index, Equality on Nonkey	$(h_i + n) * (t_T + t_S)$	(Where n is the number of records fetched.) Here, cost of index traversal is the same as for A3, but each record may be on a different block, requiring a seek per record. Cost is potentially very high if n is large.
A5	Primary B^+-tree Index, Comparison	$h_i * (t_T + t_S) + b * t_T$	Identical to the case of A3, equality on nonkey.
A6	Secondary B^+-tree Index, Comparison	$(h_i + n) * (t_T + t_S)$	Identical to the case of A4, equality on nonkey.



Summary



- Important to choose a measure of “cost” that is easy to compute and relates to the “real cost” in some way
- For simplicity, we use a measure of disk I/Os
  - But account of random vs sequential
- Even for the “selection” operation, many different ways to do it
  - Depending on whether there are indexes available, how the relation is sorted, how many results to expect etc.