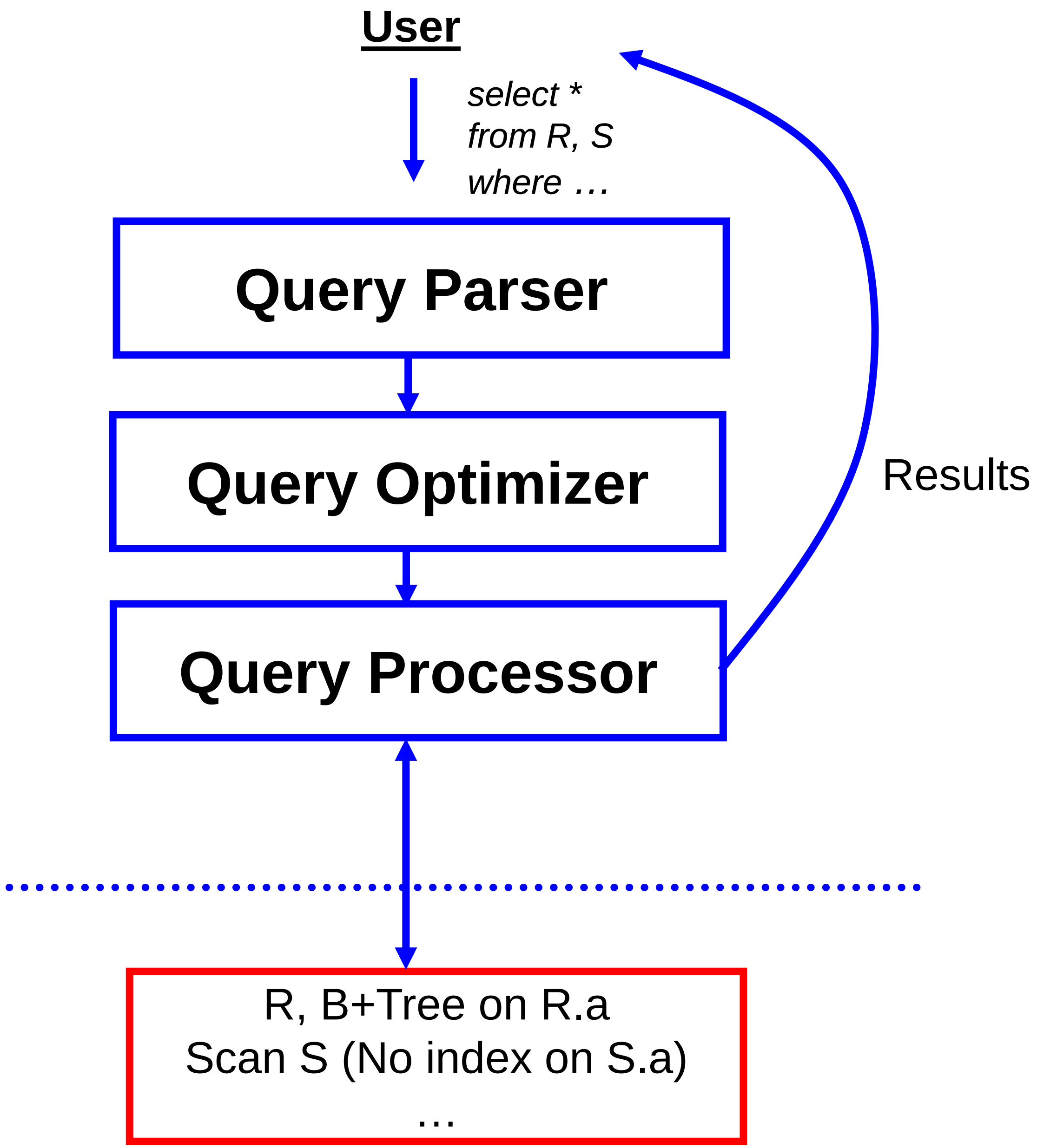
Query Processing

Overview



User

*select *
from R, S
where …*

Query Parser

Query Optimizer

Query Processor

Results

R, B+Tree on R.a
Scan S (No index on S.a)
…

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*
- …

Query planning

- Performed primarily by the optimizer
- SQL query just says “what” to get, we need to figure out “how”
- Basic process:
  - Enumerate different options
  - Assign costs to different options
  - Choose lowest cost
- Cost is not the same thing as response time

In theory what is relevant in calculating cost?

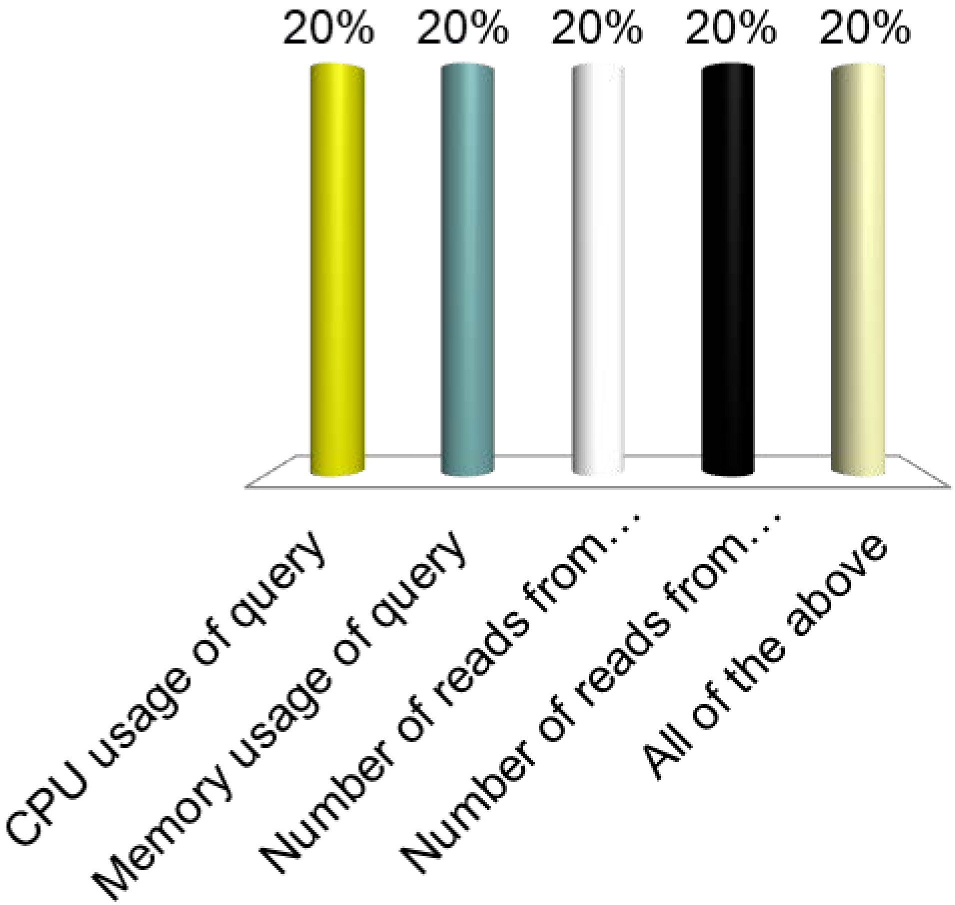
A. CPU usage of query

B. Memory usage of query

C. Number of reads from memory

D. Number of reads from disk

E. [✓ All of the above]



If we can only take into account one factor, which one should we choose?

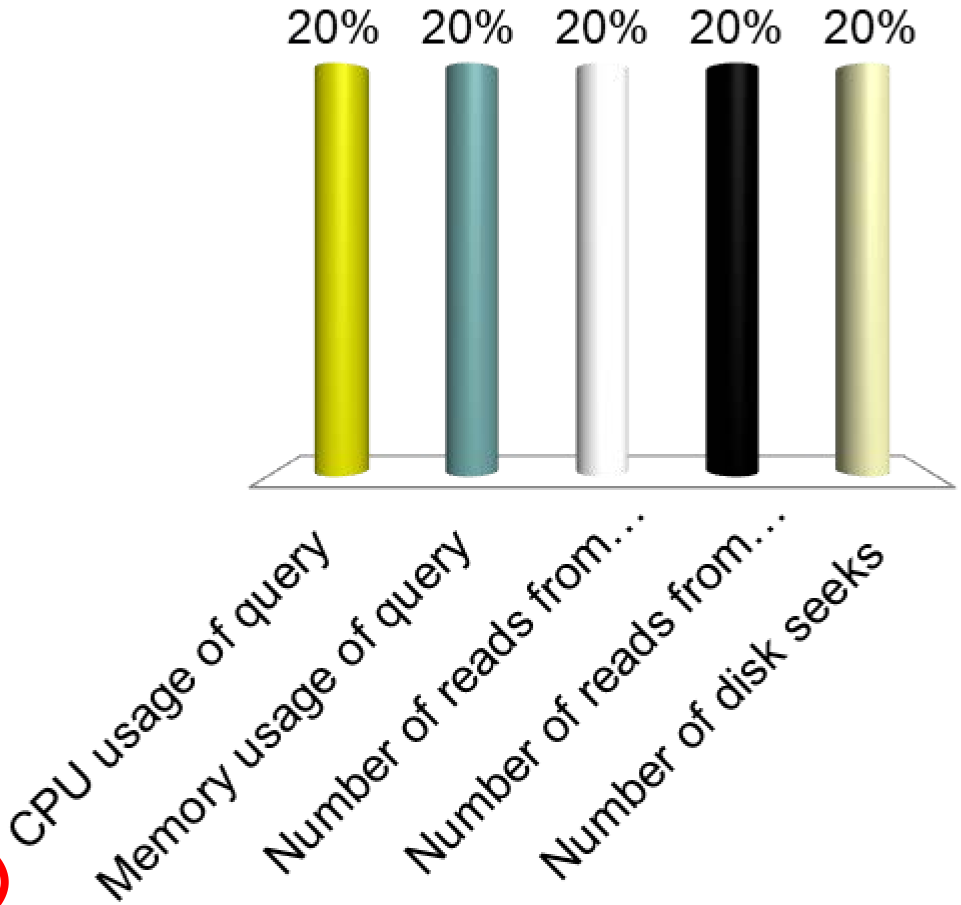
A. CPU usage of query

B. Memory usage of query

C. Number of reads from memory

D. Number of reads from disk

E. [✓ Number of disk seeks]



“Cost”

- Complicated to compute
- We will focus on disk:
  - 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*

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*

How does result change if predicate on candidate key?

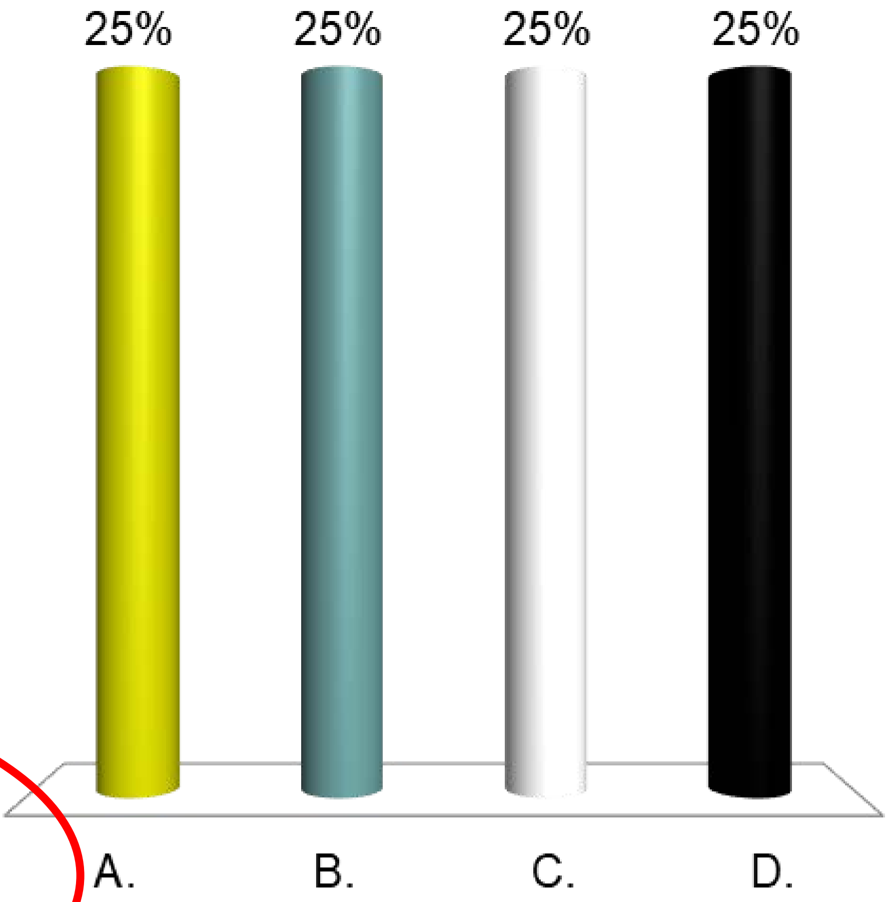
SELECT * FROM person WHERE SSN = “123”

A. There is always an index on a candidate key, and we should use that instead.

B. Since the data is sorted, we can use binary search.

C. We know for sure that the predicate will fail, so the cost is 0.

[✓ D. Once we hit the first result, we can stop. So we can assume cost is approximately half of the amount from the previous slide.]



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 short hops, but we ignore such small effects

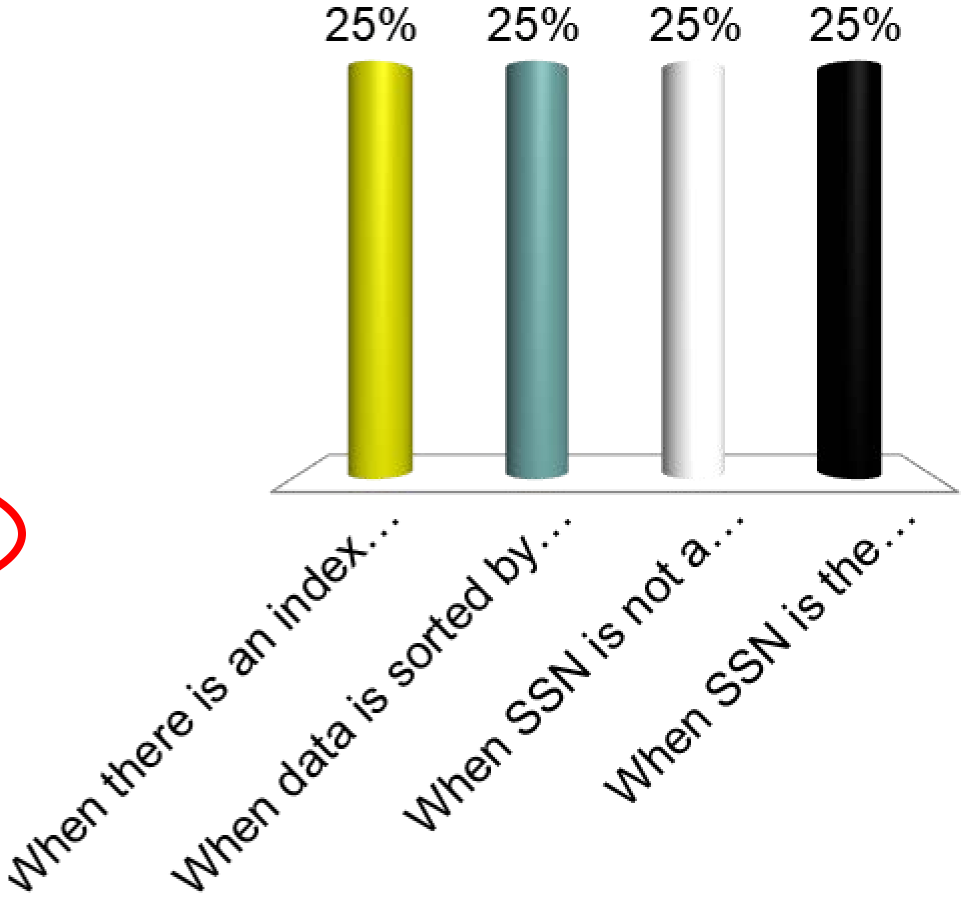
When is $\log_2(b_r) * (t_T + t_S)$ (from previous slide) too low of an estimate?

A. When there is an index on SSN

B. When data is sorted by SSN

[✓ C. When SSN is not a candidate key]

D. When SSN is the primary key



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

why?	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

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 condition 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