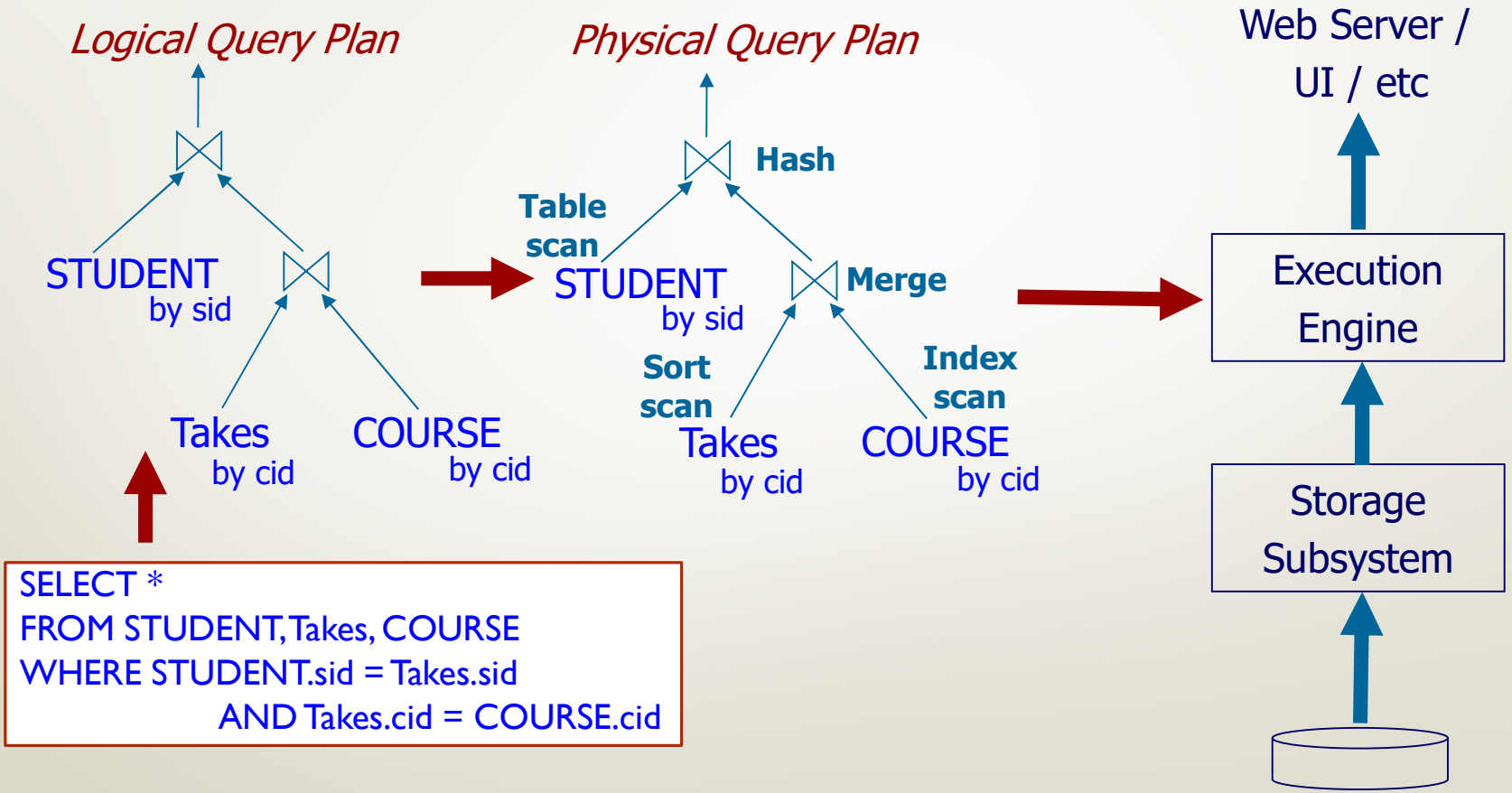
Query Processing: Physical Operators

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

QP: The Big Picture: SQL → Logical Query Plan → Physical Query Plan



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/sort-merge/hash/index join
  - In other words, physical operators are particular implementations of relational algebra operations
  - Physical operators also pertain to none RA operations, such as "scanning" a table.

Physical operators and costs

Before we describe implementations, we need to examine the *cost* involved

- Often, we have to make choices about which physical operators to use.
- For this, we need to estimate the "cost" of each physical operator.

Cost Parameters (or "statistics")

Estimating the cost:

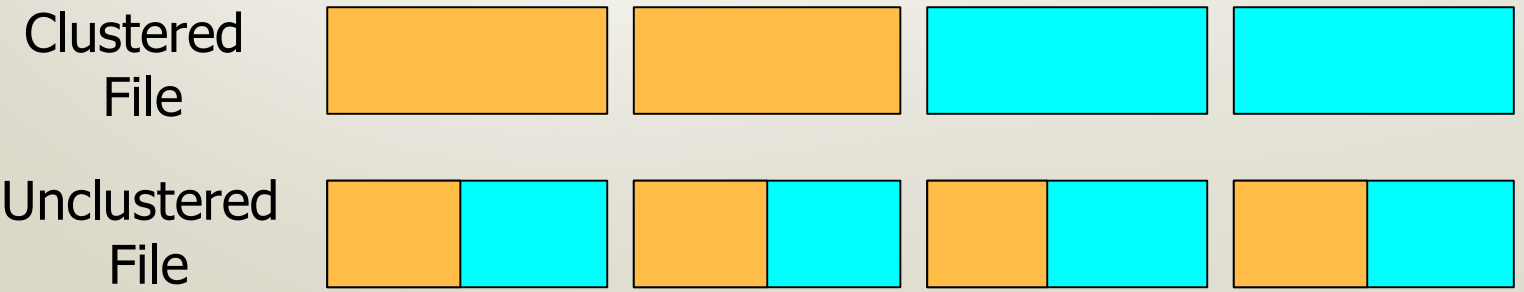
- Important in optimization (next topic: Qry Opt.)
  - Picking between plans
- Compute disk I/O cost only
  - Main memory operations are cheap
- We compute the cost to *read* the arguments of the operator
- We don't compute the cost to *write* the "final" result
  - Same for all types of operators

Cost Parameters (or "statistics")

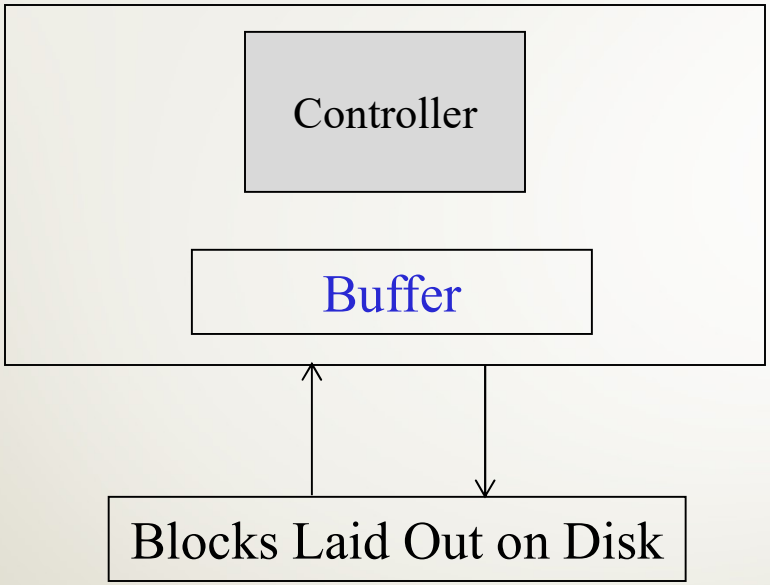
- Cost parameters
  - M = number of blocks that fit in main memory
  - B(R) = number of blocks needed to hold R (best case # blocks)
  - T(R) = number of tuples in R (worst case # of blocks)
  - V(R,a) = number of distinct values of the attribute a

T(R) / (number of tuples that fits in a block) = B (R)

Here T(R) = 4; B(R) = 2



A Mental Model for Costing Operator Implementations



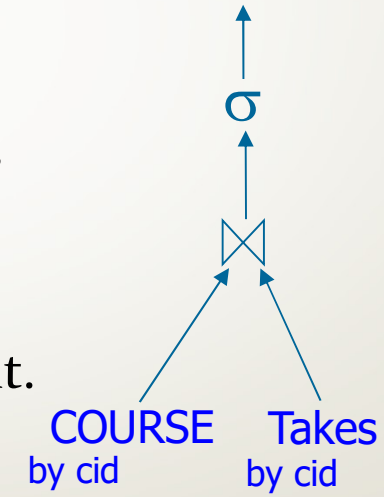
Temp computation space in main memory

We compute the cost to *read* the arguments of the operator
We don't compute the cost to *write* the result

The iterator model for implementing operators

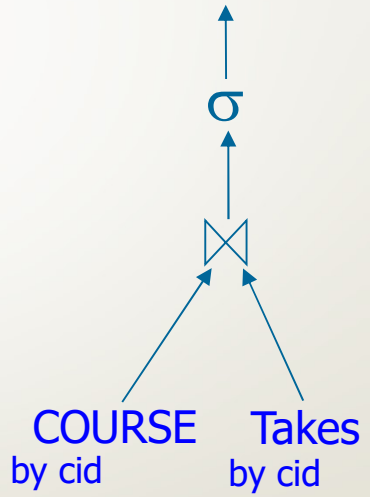
Each (physical) operation is implemented by 3 functions:

- **Open**: sets up the data structures and performs initializations
- **GetNext**: returns the the next tuple of the result.
- **Close**: ends the operations. Cleans up the data structures.



The iterator model for implementing operators

- Enables pipelining!
  - As opposed to: execute each operator in entirety, store its results on disk or in main memory
  - Many operators can be "active" simultaneously
- Pipelining: Not always possible (or meaningful):
  - E.g., "sort scan".
  - "Blocking" operators

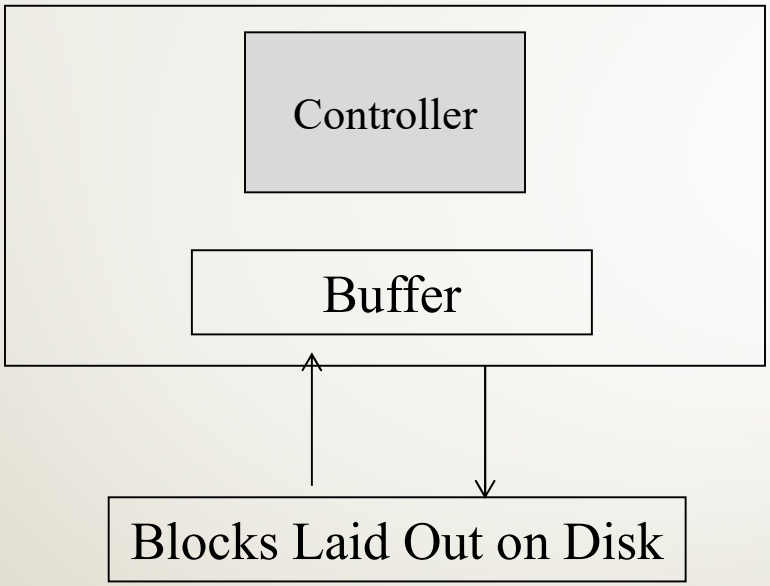


Overview of operator implementations: first classification

Operator algorithms mostly of one of these types:

- Sorting-based
- Hash-based
- Index-based

Another classification of operator implementations



***One pass*:**

- reading the data only once from disk.
- Typically, at least one of the arguments must fit in memory.

***Two pass*:**

- Data need not fit in memory but is not "too large".

***Multipass*:**

- No limit on data size.

How to think about operators

- Classification 1: Hash/Index/Sort
- Classification 2: One/Two/Many pass

- Metric 1: I/O Cost
- Metric 2: Buffer requirements (how large should M be)