CSE462/562: Database Systems (Fall 24)

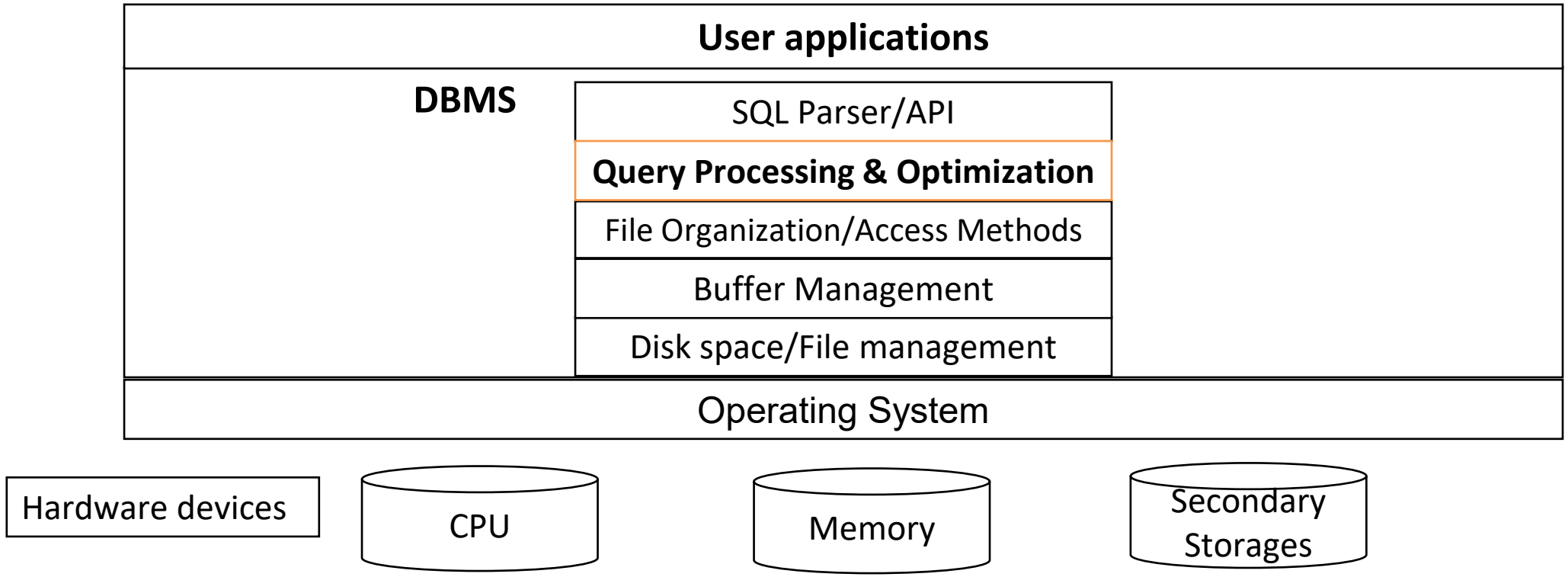
Lecture 8: Query Processing Overview

9/19/2024

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

Last updated: 9/10/2024 9:35 AM

Big picture


User applications
DBMS
SQL Parser/API
Query Processing & Optimization
File Organization/Access Methods
Buffer Management
Disk space/File management
Operating System
Hardware devices
CPU
Memory
Secondary Storages

What's discussed so far

- The lower-level storage layer in DBMS
  - Disk/file space management
  - Buffer management
  - File organization
  - Access methods
    - Indexing

- How to answer queries/perform updates?
  - Relational algebra vs SQL
  - *Correctness?*
  - *Efficiently?*

- Query processing & optimization

student

sid	name	login	major	adm_year
100	Alice	alicer34	CS	2021
101	Bob	bob5	CE	2020
102	Charlie	charlie7	CS	2021
103	David	davel	CS	2020

Find the names and the grades of all the students enrolled in the course 562 who were admitted in the year of 2021?

100	s22	562	4.0
102	s22	562	2.3
100	f21	560	3.7
101	s21	560	3.3
102	f21	560	4.0
103	s22	460	2.7
101	f21	560	3.3
103	f21	250	4.0

Simple select query and relational algebra

- Recall that the basic form of SELECT query can be translated into extended relational algebra
  - The conceptual way of answering the query
  - With some non-relational operators (notably Sort).

-- SQL SELECT with no aggregation

SELECT [DISTINCT] $E_1, E_2, \ldots, E_m$
FROM $R_1, R_2, \ldots, R_n$
[WHERE P]
[ORDER BY *expr* [ASC|DESC] [,…]]

non-relational

$$Sort\left(Distinct\left(\pi_{E_1,E_2,\ldots E_m}\sigma_P R_1 \times R_2 \times \cdots \times R_n\right)\right)$$

-- SQL with aggregation

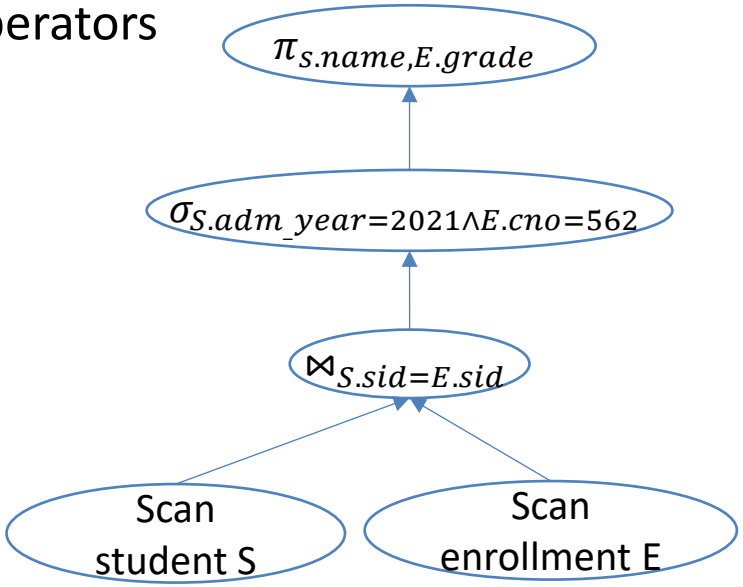
SELECT $E'_1, E'_2, \ldots, E'_m, F_1(E''_1), \ldots, F_k(E''_k)$
FROM $R_1, R_2, \ldots, R_n$
[WHERE P]
[GROUP BY $E_1, E_2, \ldots, E_l$
[HAVING P']]
[ORDER BY expr [ASC|DESC] [,…]]

$$Q \leftarrow \sigma_P R_1 \times R_2 \times \cdots \times R_n$$

$$Sort\left(\pi_{E'_1,E'_2,\ldots,E'_m,F_1(E''_1),\ldots,F_k(E''_k)}\sigma_{P'}\left(_{E_1,E_2,\ldots,E_l}\gamma_{F_1(E''_1),\ldots F_k(E''_k)}Q\right)\right)$$

Query processing overview

- DBMS translates SQL to a special internal language
  - Query plans
    - *logical:* extended relational algebra with some non-relational operators
    - *physical:* describes the actual implementation of the operators
- Think of query plans as data-flow graphs
  - Edges: flow of records
  - Vertices: relational and non-relational *operators*
    - Input/Output of the operators: relations
- Three stages of query processing
  - **Parsing & query rewriting:** SQL -> logical plan
  - **Query optimization:**
    logical plan -> optimized logical plan -> physical plan
  - **Query execution**: evaluating the physical plan over the database



An example of logical plan

Query processing overview

ODBC/JDBC/ command line frontend

**

SQL Query

```
SELECT S.name,E.grade
FROM student S, enrollment E
WHERE S.sid = E.sid
  AND S.adm_year = 2021
  AND E.cno = 562;
```

*SQL Parser**

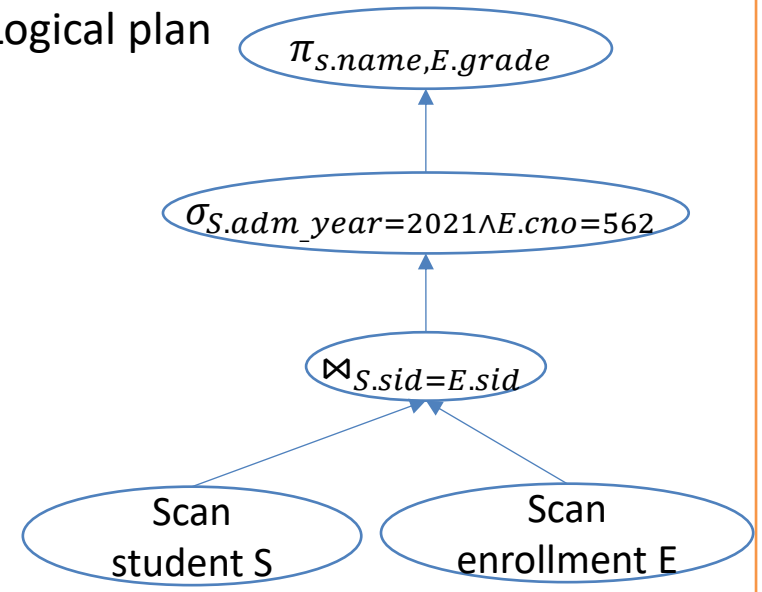
** include multiple intermediate steps (e.g., parsing tree/analysis/rewriting)*

(Extended) Relational Algebra

$\pi_{S.name,E.grade}\sigma_{S.adm\_year=2021 \wedge E.cno=562} S \bowtie_{S.sid=E.sid} E$

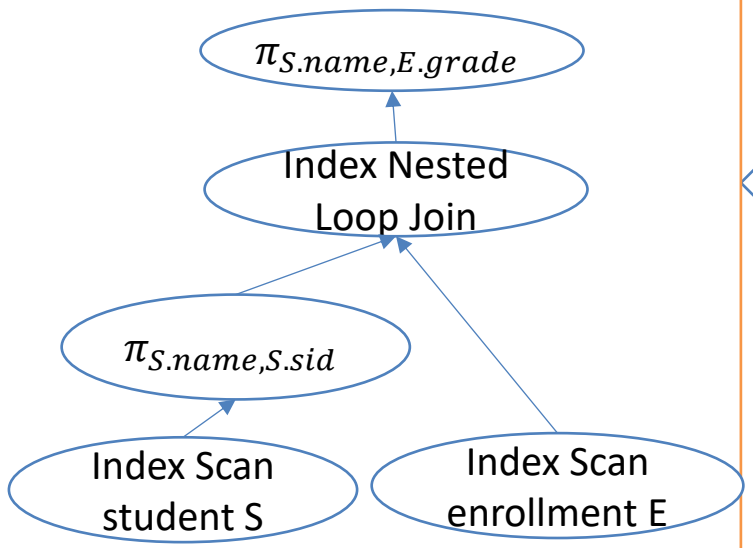
Internally represented as

Logical plan



Query Optimizer

Physical plan



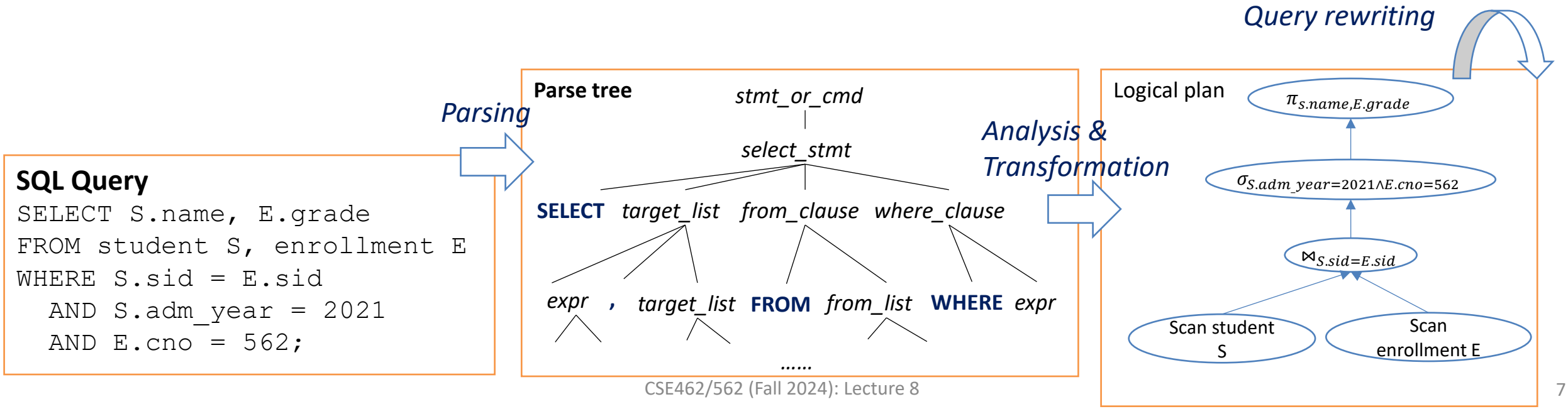
Query Execution

Query result

```
S.name  | E.grade
Alice   | 4.0
Charlie | 2.3
(2 rows)
```

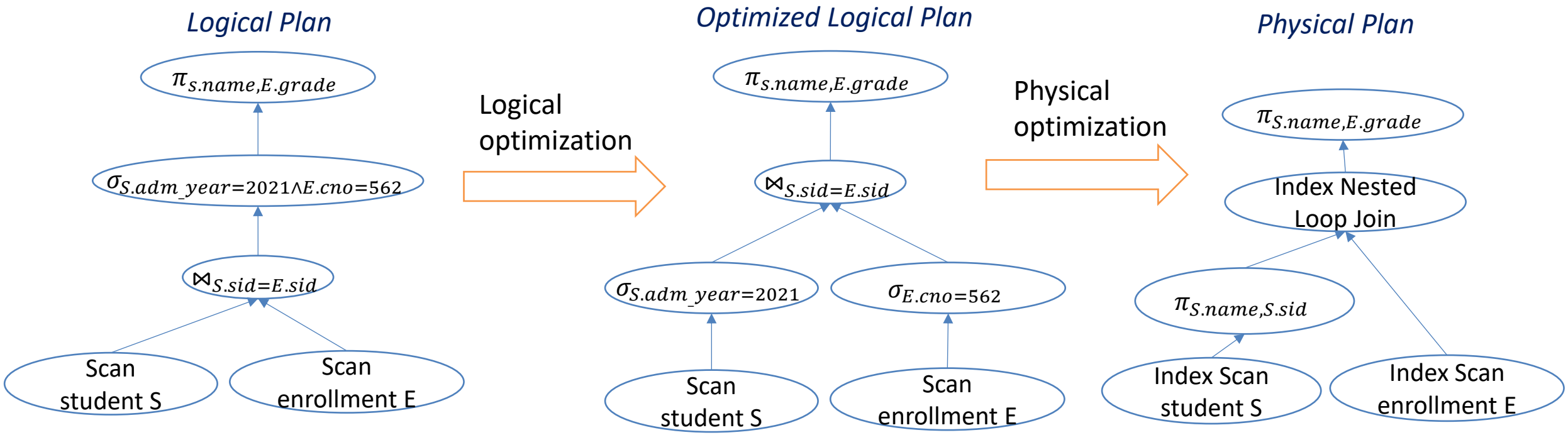
Parsing and query rewriting

- SQL Parser are usually generated from a context-free grammar using compiler tools
  - e.g., antlr (LL grammar), lex+yacc/flex+bison (LR grammar/LALR(1) grammar)
  - We'll omit the details which are covered in compiler courses
  - Produces a *parse tree* for a SQL query
- Analysis and transformation into logical plan
  - A parse tree represents the syntactical structure of a SQL query -- not suitable for query processing
  - Needs to be translated into a logical plan
  - Catalog information helps resolving tables/columns/types/expressions/functions
- Query rewriting
  - User defined/system defined rules for transforming queries (e.g., non-materialized views, customized rewriting rules)



Query optimization (a preview)

- Many equivalent plans exist for the same query
  - Efficiency varies
- Query optimization
  - Finding ~~*the best*~~ *a not-too-bad plan* with reasonable overhead
  - Generally divided into two phases



Query execution

- Query executor needs to evaluate the result of a physical plan over a database instance

- Query interpretation vs compilation
  - To date, most DBMS uses a single piece of binary code that "interprets" the query plans
    - Uses run-time information to determine which function(s) to call
    - Easy to implement with runtime polymorphism (e.g., C++/Java/Scala)
  - Some modern DBMS compiles query plans into binary code for efficiency (e.g., [1])
    - Avoids virtual function call overhead in tight loops
    - More efficient for queries over large database
    - Overhead for compilation (LLVM to the rescue) and a bit harder to implement

  - Can take hybrid approach:
    - e.g., only compiling expression trees into binary code, while interpreting the physical plan

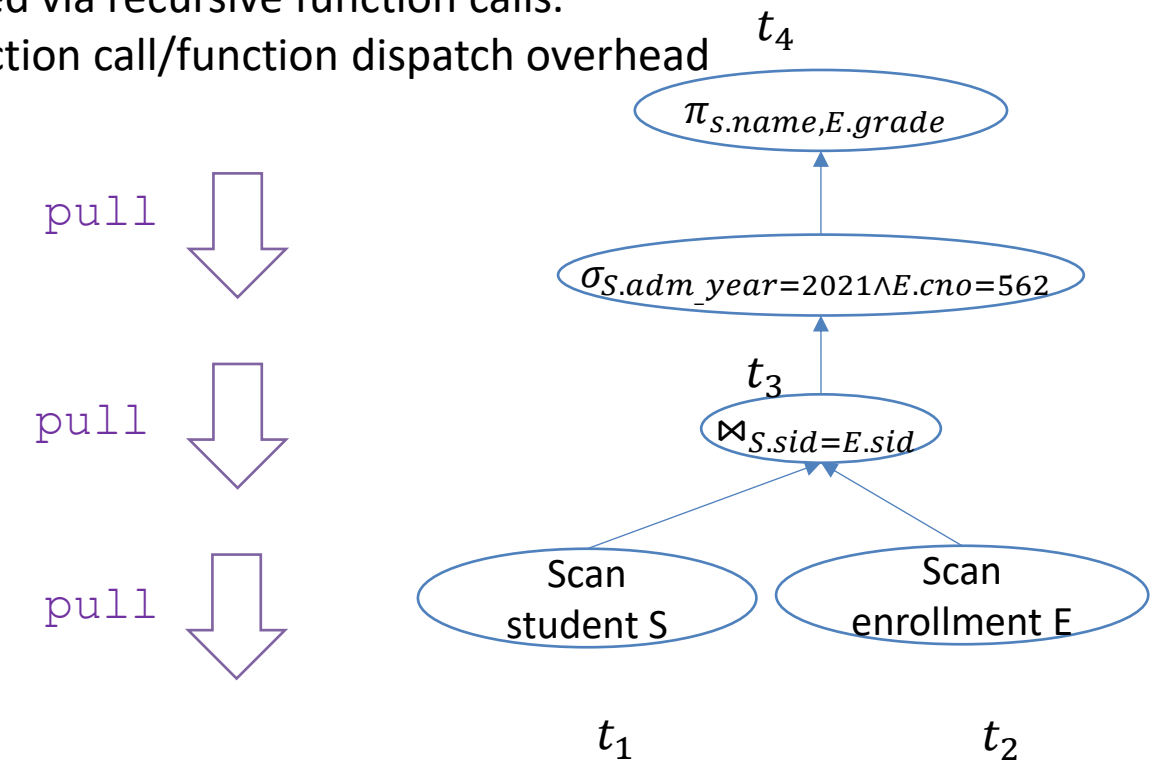
[1] Efficiently Compiling Efficient Query Plans for Modern Hardware. Thomas Neumann, 2011.

Query execution (cont'd)

- *Pull-based* vs *push-based* query execution

Pull-based query execution

- Start from root and pull data from children
- Tuple passed via recursive function calls.
- Virtual function call/function dispatch overhead

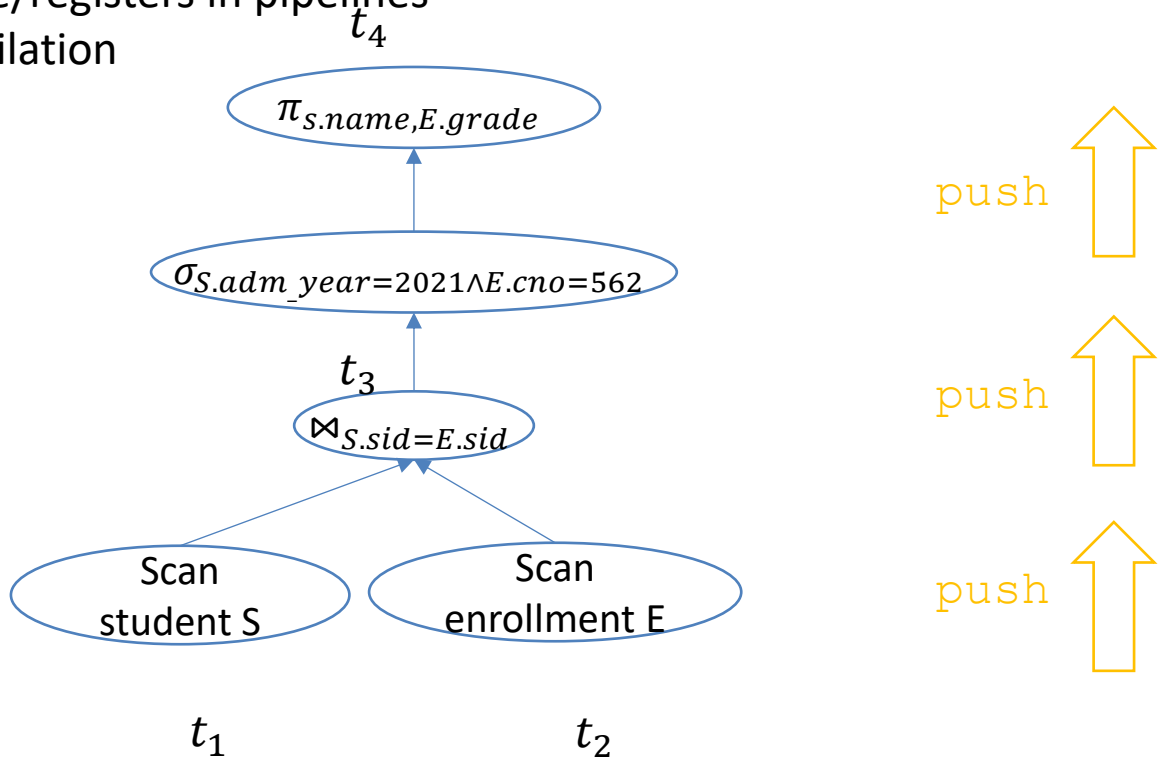


Query execution (cont'd)

- *Pull-based* vs *push-based* query execution

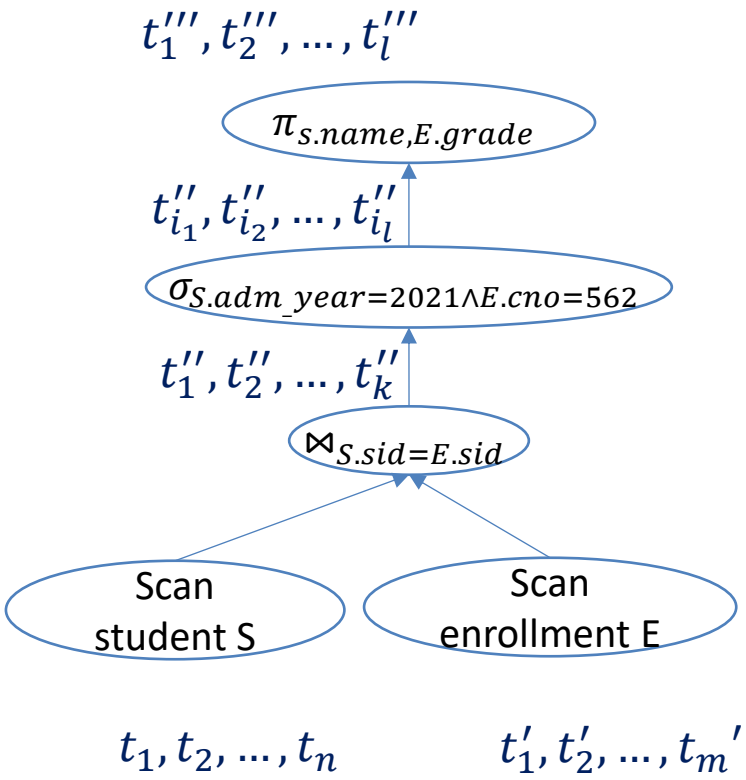
Push-based query execution

- Start from leaf and push data to parent
- Allows more efficient use of cache/registers in pipelines
  - when used with query compilation



Query execution (cont'd)

- *Pull-based* vs *push-based* query execution
- *Pipelining* vs *materialization*



Query execution models

- Several models for implementing the operators
  - Volcano model (aka iterator model)
    - most traditional and widely used one
    - pull-based execution
  - Materialization model
  - Vectorization model

- Running example

```
SELECT * FROM student
WHERE major='CS' ORDER BY adm_year;
```