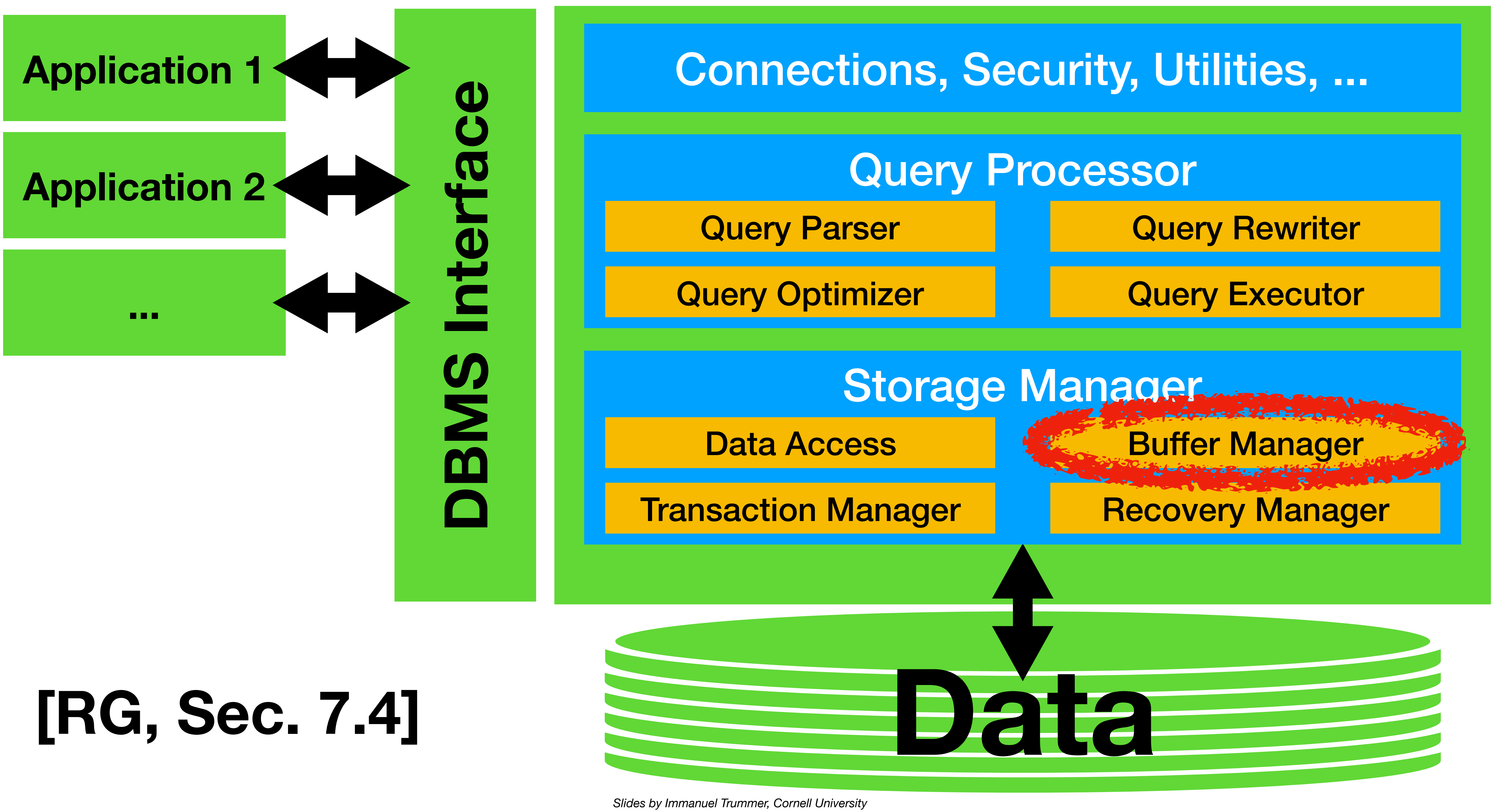
Query Processing Overview

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Database Management Systems (DBMS)

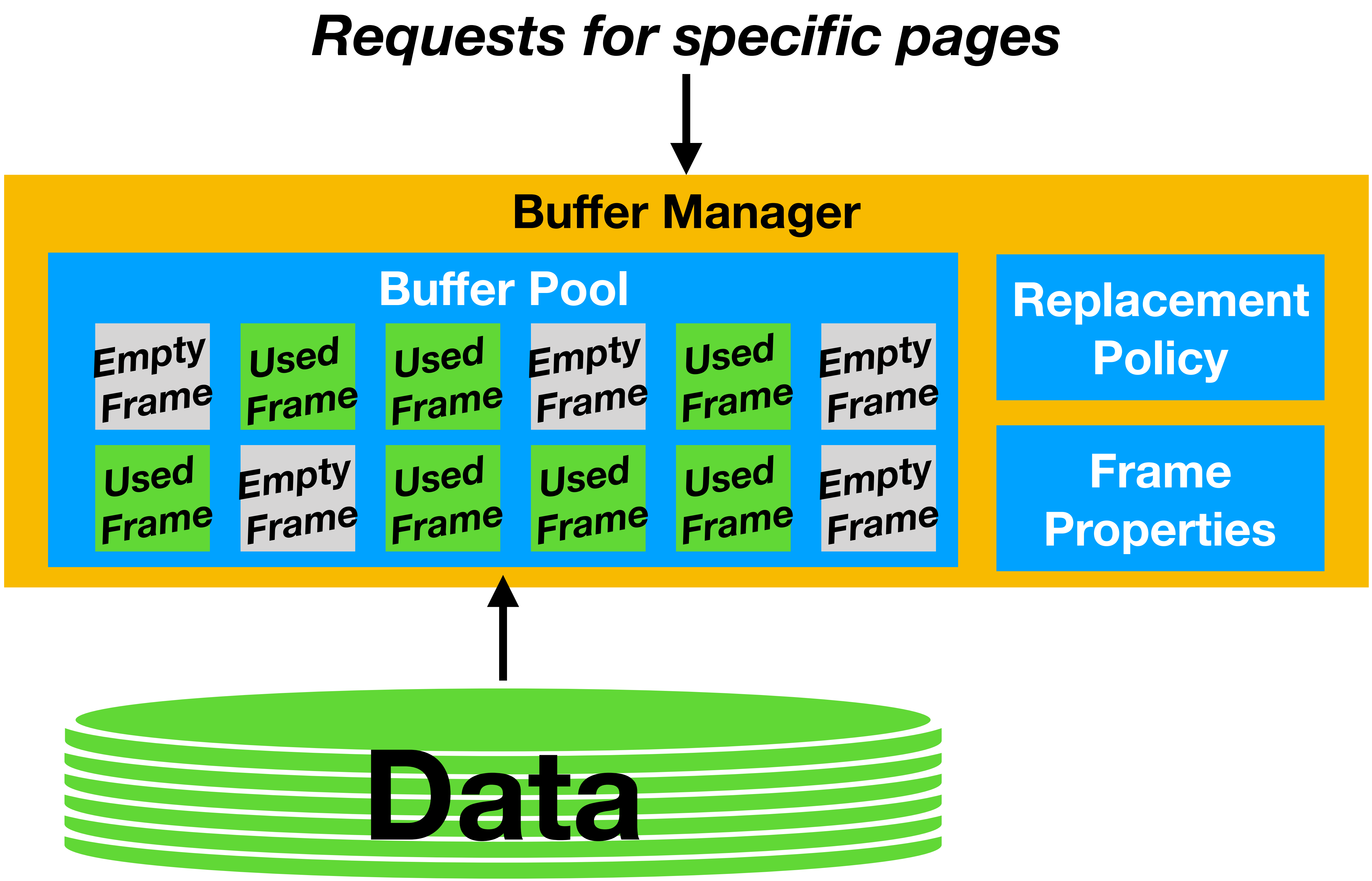


[RG, Sec. 7.4]

Buffer Manager

- Decides when to **move data** between disk and RAM
- Tries to **reduce** data movements using heuristics
- Buffer manager manages "**buffer pool**"
  - Buffer pool: main memory **reserved** for DBMS
  - Divided into page-sized slots called **"frames"**
  - Stores meta-data about each slot

Buffer Manager Illustration



Frame Properties

- **Page ID**: which page is currently stored in frame?
  - Allows **matching** page requests to frames
- **Pin count**: how many processes are using a page?
  - Can only **evict** page if pin count reaches zero
- **Dirty bit**: in-memory page deviates from disk version?
  - Must **write** page to disk before evicting it

Processing Page Requests

- Case 1: **Cache Hit** (requested page cached)
  - **Increase pin** count and **return page** address
- Case 2: **Cache Miss** (requested page not cached)
  - **Choose frame** for replacement (replacement policy)
  - If frame contains dirty page then **write** it to disk
  - **Read** requested page from disk and store in frame
  - **Increase pin** count and **return page** address

Why Not Rely on OS?

- **OS caches** pages as well (virtual memory)
- Why do we want a **separate** buffer manager?
  - DBMS **knows** its access patterns ahead of time
    - Can exploit for smarter **replacements**
  - DBMS must control **page writes** for safety guarantees

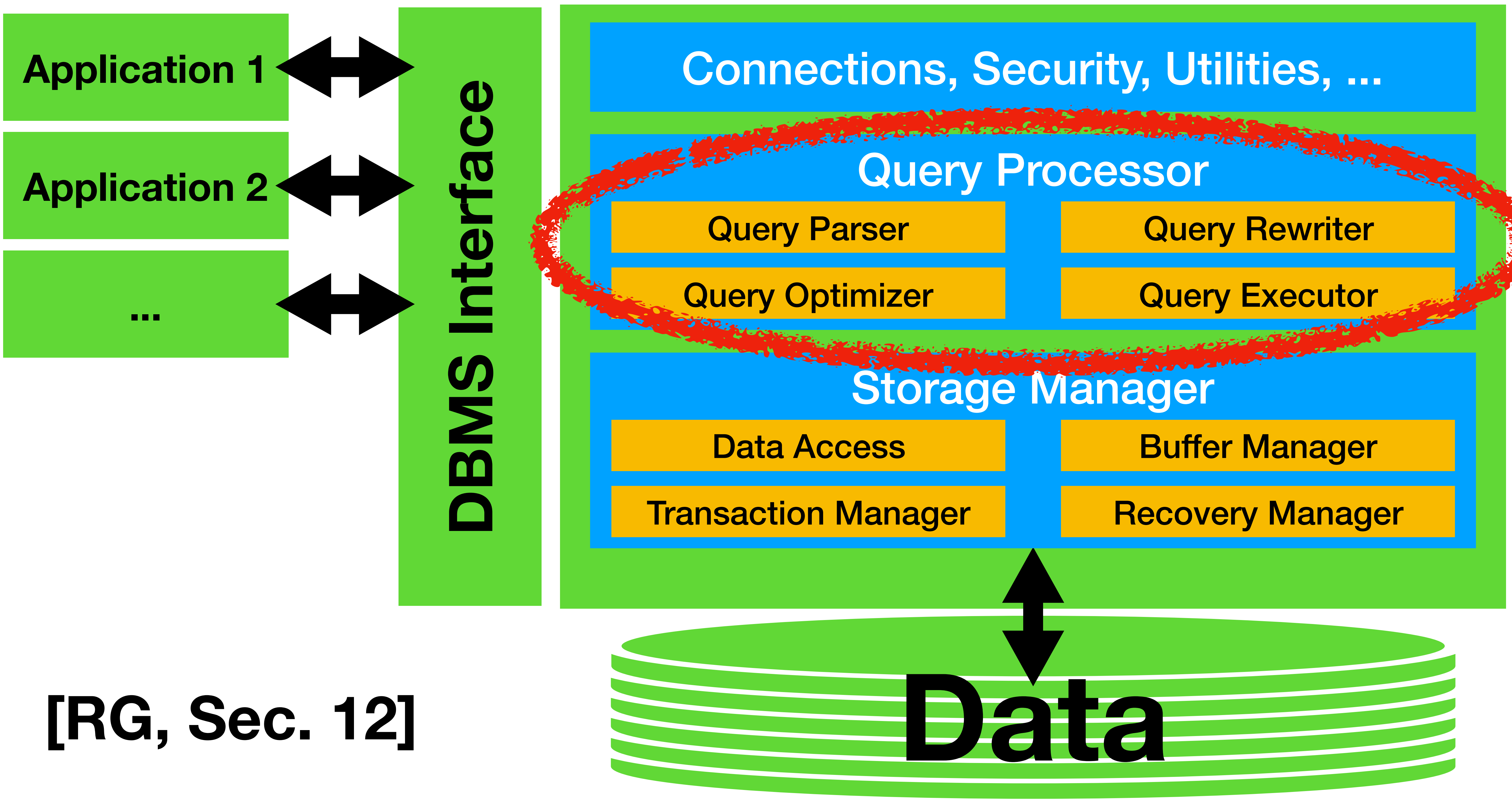
LRU Replacement Policy

- Want to replace page required **farthest** in the future
  - Doing so **reduces** expensive cache misses
- However: **difficult** to predict that in general
- Heuristic: remove **least recently used** page (LRU)
  - Did not use page for long time, **unlikely to do soon**

Sequential Flooding

- DBMS often have **particular** access patterns
- For instance: **keep scanning** pages in round robin mode
- Least recently used page is used again **soonest**
- Makes **LRU** policy highly **sub-optimal**
- Otherwise a reasonable strategy!

Database Management Systems (DBMS)



[RG, Sec. 12]

Query Processing

- Input query is parsed (**Parser**) and simplified (**Rewriter**)
- **Query optimizer** generates optimized execution plan
- Executing plan (**Executor**) produces query result

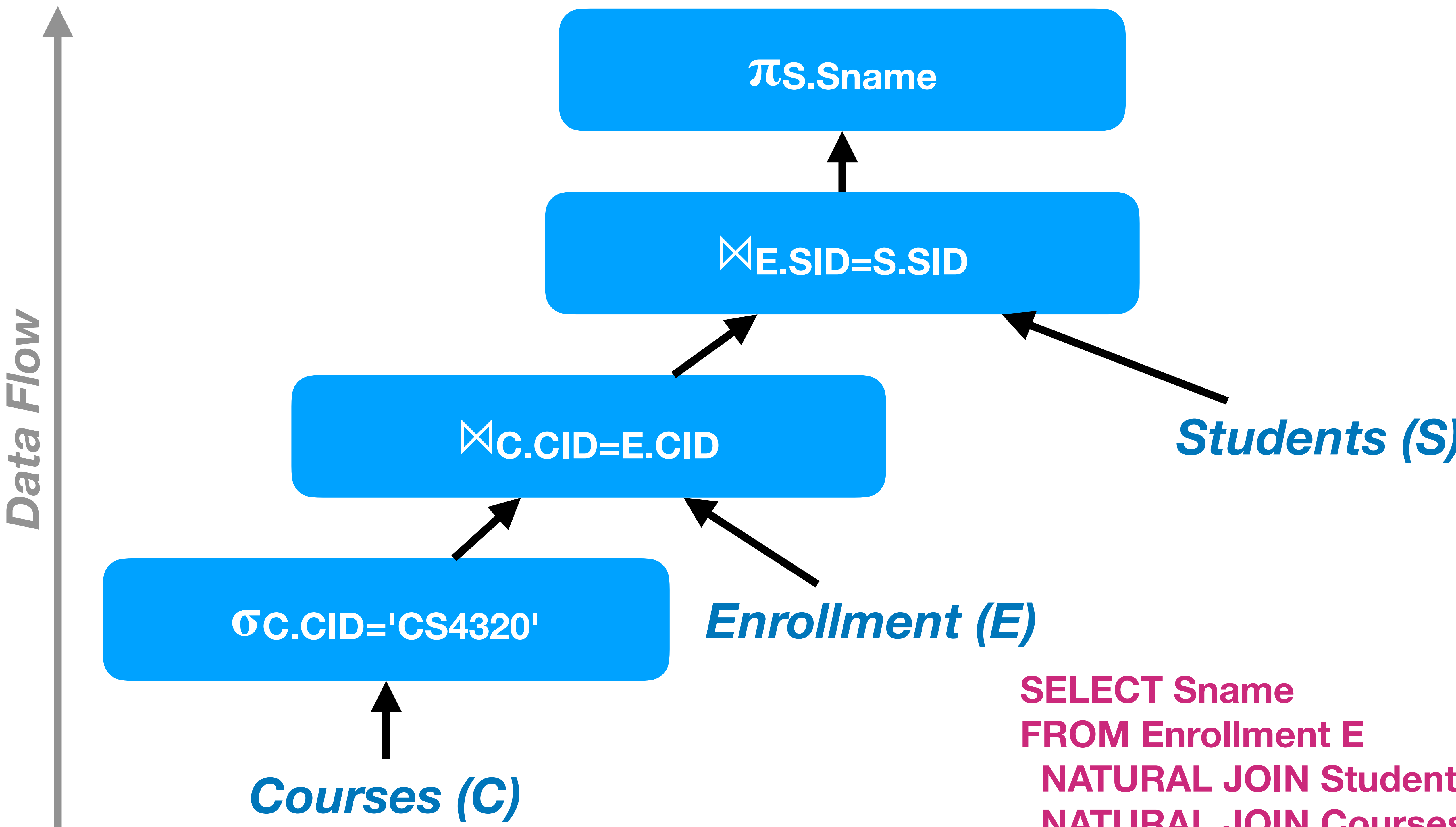
Query Plans

- Describe **how** to generate required data
- Typically represented as a **tree**
- Each leaf node represents a database **table**
- Each inner node represents an **operation**
- Tree edges represent **data flow**

Operators

- Query plans use fixed set of **standard operators**
- Consumes relation(s) and produces one **relation**
- **Filter operator (σ)**: discard rows based on condition
- **Projection operator (π)**: discard columns
- **Join operator ($\bowtie$)**: find matching tuple pairs
- …

Example Plan

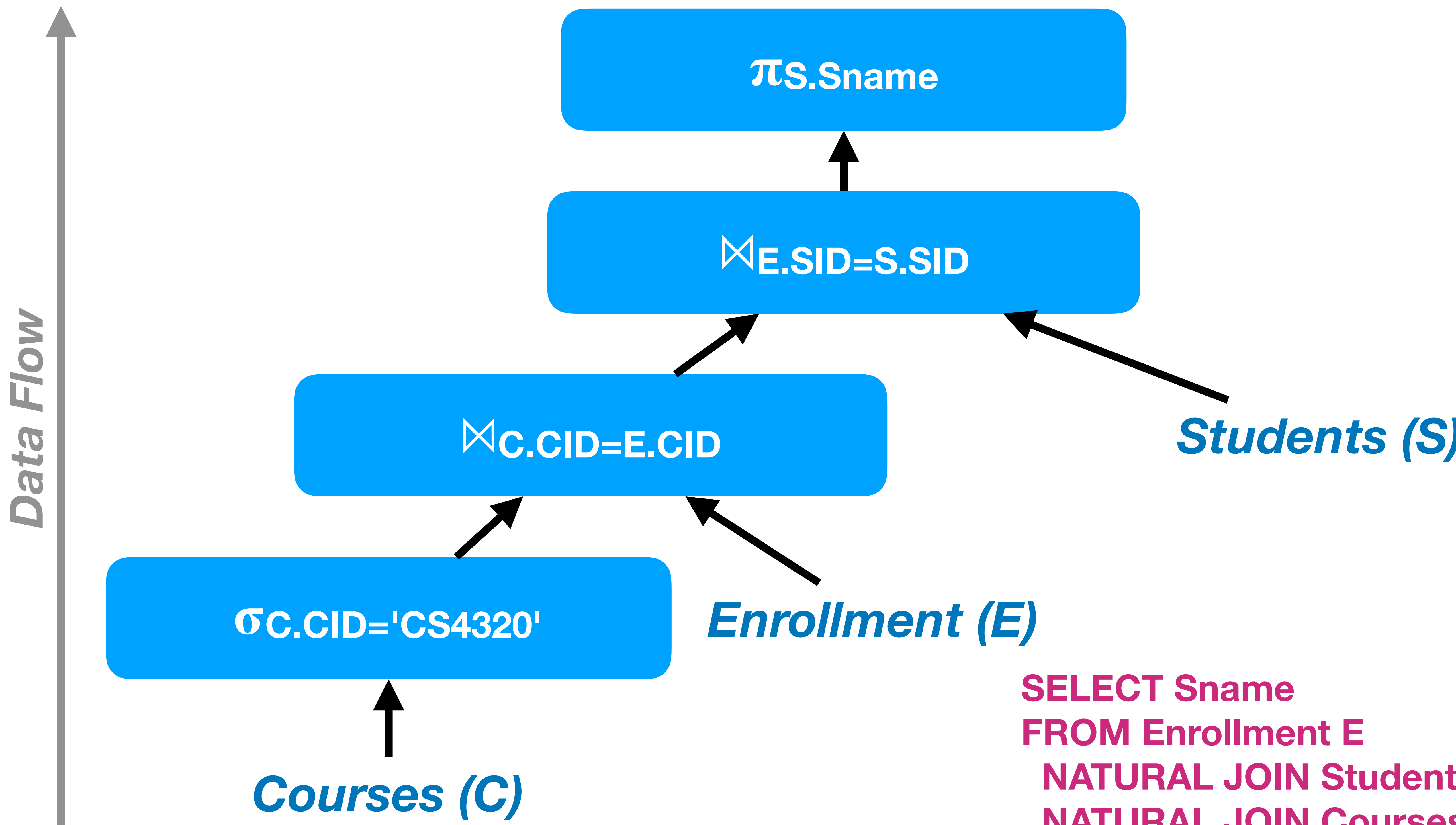


SELECT Sname
FROM Enrollment E
NATURAL JOIN Students S
NATURAL JOIN Courses C
WHERE C.CID = 'CS4320'

Relational Algebra

- Mathematical **foundations** for describing query plans
- Represents query plan as "mathematical **expression**"
- Won't discuss in too much detail here

Example Plan




```sql
SELECT Sname
FROM Enrollment E
  NATURAL JOIN Students S
  NATURAL JOIN Courses C
WHERE C.CID = 'CS4320'
```

Relational Algebra Representation

$$\pi_{Sname}\Big(\big(\sigma_{C.CID='CS4320'}(C) \bowtie_{C.CID=E.CID} E \big) \bowtie_{E.SID=S.SID} S \Big)$$

(Evaluated from inner to outer expressions)