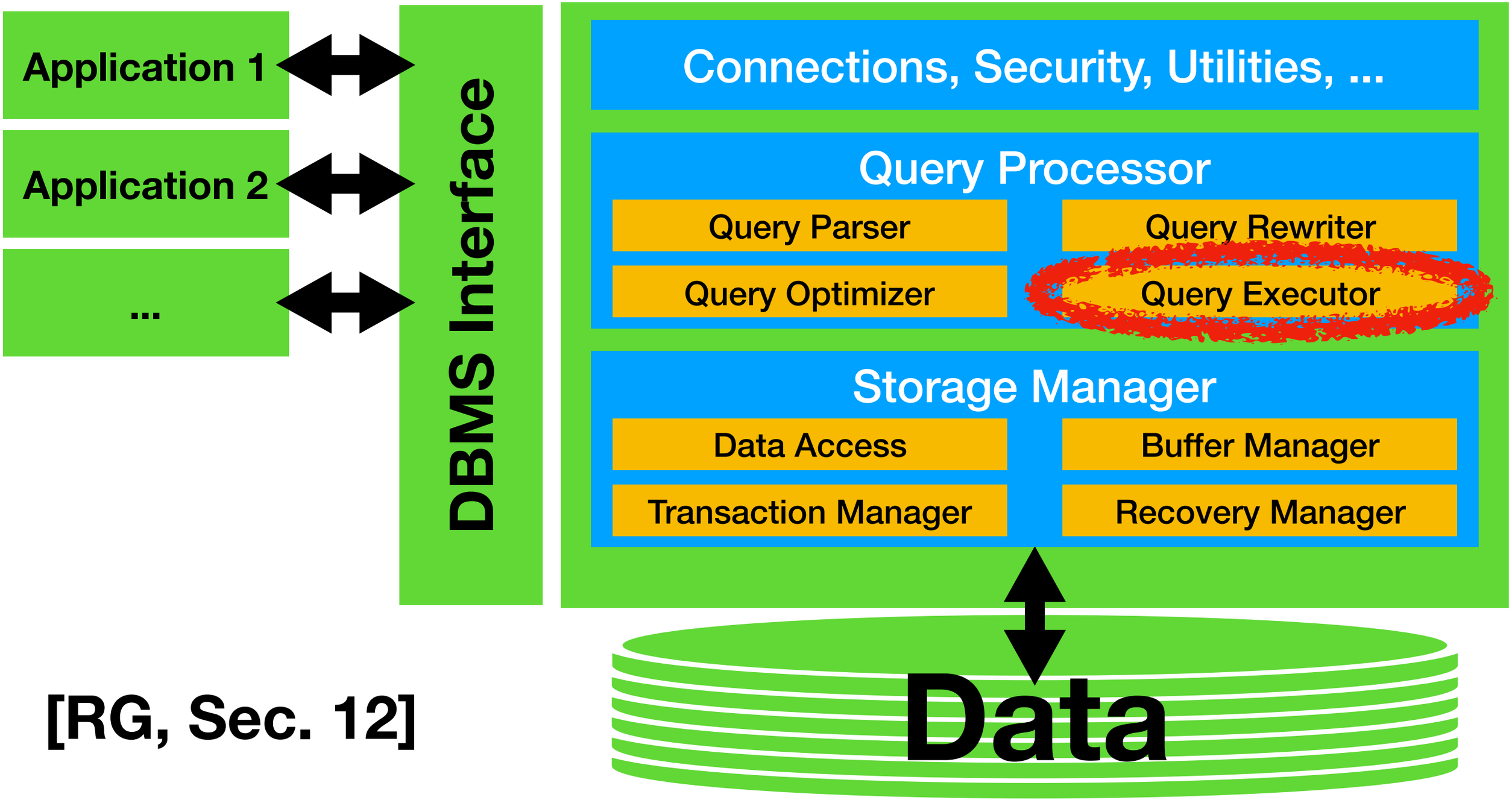
More Operators and Query Plans

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

Database Management Systems (DBMS)



[RG, Sec. 12]

Projection (π)

Projection Operator

- Straight forward for SELECT **without DISTINCT**
  - **Calculate SELECT items**, drop other columns
- More difficult if **DISTINCT keyword** is present
  - Need to **filter out duplicates** - multiple options:
    - Exploit **hash** function
    - Exploit **sorting**
    - Exploit **index**

Projection & Duplicate Elimination via Hashing

- Phase 1: **partition data** into hash buckets
  - **Scan input**, calculate projection, partition by hash function
  - Data partitions are written **back to hard disk**
- Phase 2: **eliminate duplicates** for each partition
  - **Read one partition** into memory and eliminate duplicates
  - Can use **second hash function** to detect duplicates
- Constraints on **memory** similar as for hash join
  - Count **hash buckets** for Phase 1, **bucket size** for Phase 2

Projection & Duplicate Elimination via Hashing

- Phase 1: **partition data** into hash buckets
  - **Scan input**, calculate projection, partition by hash function
  - Data partitions are written **back to hard disk**
- Phase 2: **eliminate duplicates** for each partition
  - **Read one partition** into memory and eliminate duplicates
  - Can use **second hash function** to detect duplicates
- Constraints on **memory** similar as for hash join
  - Count **hash buckets** for Phase 1, **bucket size** for Phase 2

(Cost is 3 * Number of Pages - at most)

What If Hash Buckets Do Not Fit in Memory?

Projection & Duplicate Elimination via Sorting

- Idea: **sorting** rows helps finding duplicates
  - (Duplicates appear **consecutively**)
- Use variant of **external sort** algorithm seen before
  - Apply **projection** during first pass over data
  - **Eliminate in-memory duplicates** during all steps
  - The result is **duplicate-free** and sorted
  - Can reduce number of passes with more **main memory**

Projection & Duplicate Elimination via Sorting

- Idea: **sorting** rows helps finding duplicates
  - (Duplicates appear **consecutively**)
- Use variant of **external sort** algorithm seen before
  - Apply **projection** during first pass over data
  - **Eliminate in-memory duplicates** during all steps
  - The result is **duplicate-free** and sorted
  - Can reduce number of passes with more **main memory**

(Cost is external sorting cost - at most)

Projection & Duplicate Elimination via Index

- Assume **index key** includes projection columns
  - Can retrieve relevant data from **index alone**
    - Saves cost considering **index smaller** than data
- Even better: **tree index** with projections as **key prefix**
  - **Duplicates** retrieved consecutively, easy to eliminate

Projection & Duplicate Elimination via Index

- Assume **index key** includes projection columns
  - Can retrieve relevant data from **index alone**
    - Saves cost considering **index smaller** than data
- Even better: **tree index** with projections as **key prefix**
  - **Duplicates** retrieved consecutively, easy to eliminate

(Cost of reading index data)

Grouping (Γ) & Aggregation (∑ ...)

Aggregation without Groups

- SQL offers **Min, Max, Sum, Count, Avg**
- Scan input data and **update** in-memory aggregate
  - Can use **constant** amount of memory
  - Cost of **reading** input data once
- **Count distinct** requires duplicate elimination (see prior)

Aggregation with Groups

- Can use **hashing**
  - Maintain hash table of group keys with aggregates
- Can use **sorting**
  - Sort on group keys, aggregate groups consecutively
- Can use **indexes**
  - Index key must contain group-by keys

Set Operations (∩,∪,-)

Set Operations

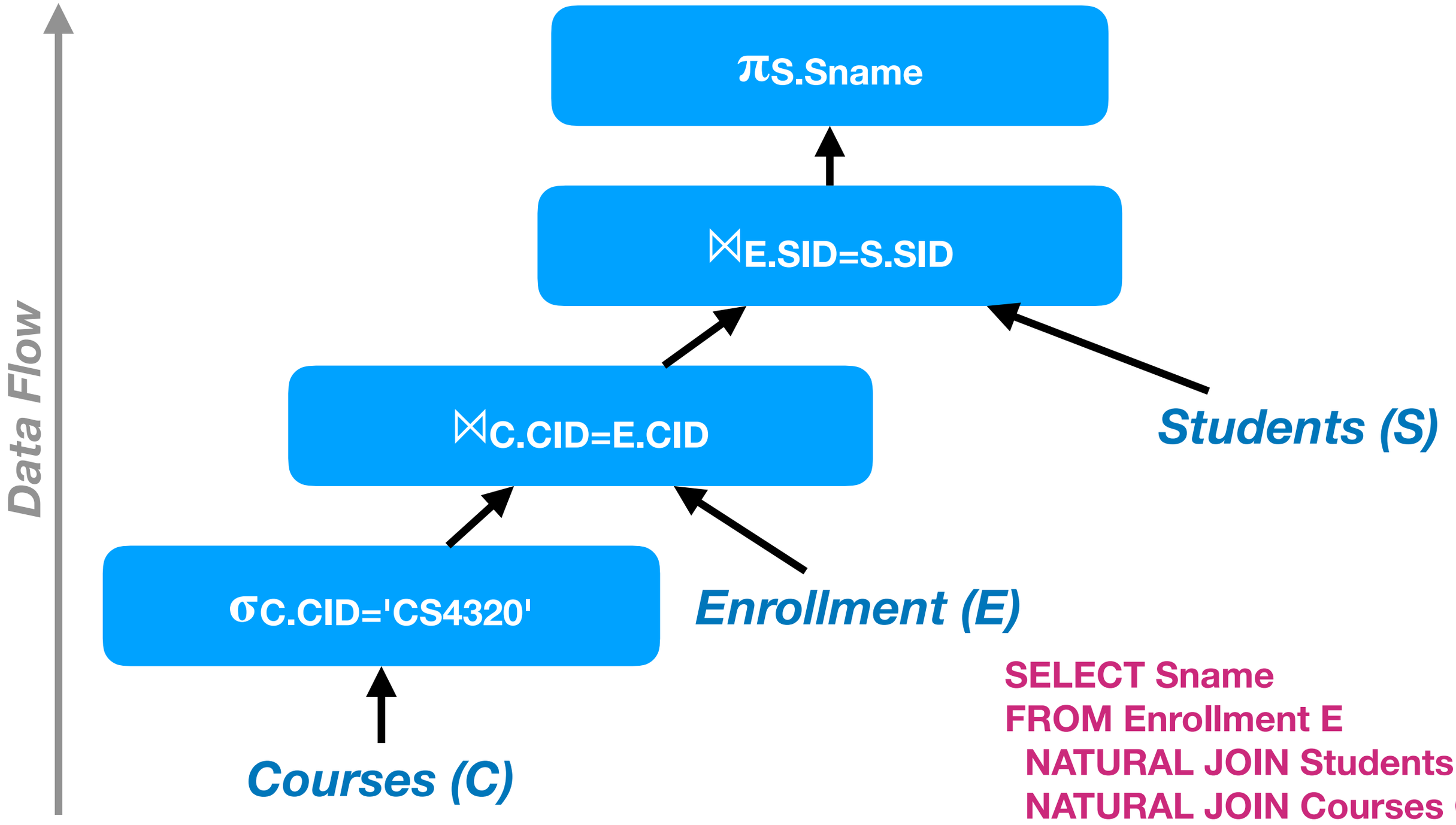
- **INTERSECT** can be handled like a join
  - Join condition is **equality** on all columns
- **UNION** eliminates duplicates (unlike UNION ALL)
  - Either **hash** and eliminate duplicates in each bucket
  - Or **sort** and eliminate duplicates during merging
- **R EXCEPT S**
  - Either partition via **hash**, then treat each bucket separately
  - Or **sort** and check whether R tuple in S during merge steps

Query Plans

Reminder: Query Plans

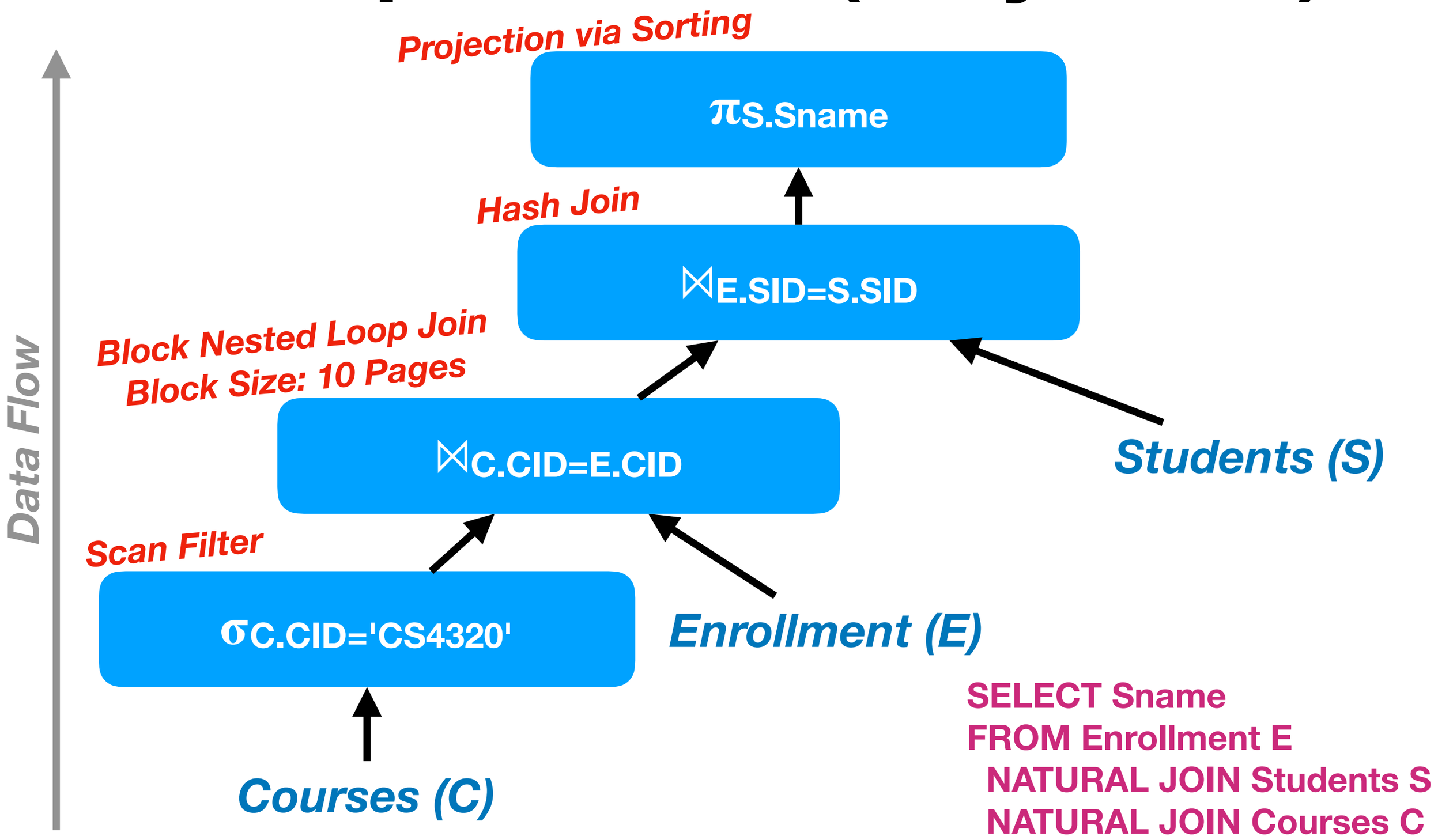
- Query plans describes query processing as **tree**
- Inner nodes are **operators**, leaf nodes are **tables**
- **Logical** query plan just specifies types of operations
- **Physical** query plan specifies implementation as well

Example Plan (Logical)




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

Example Plan (Physical)




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

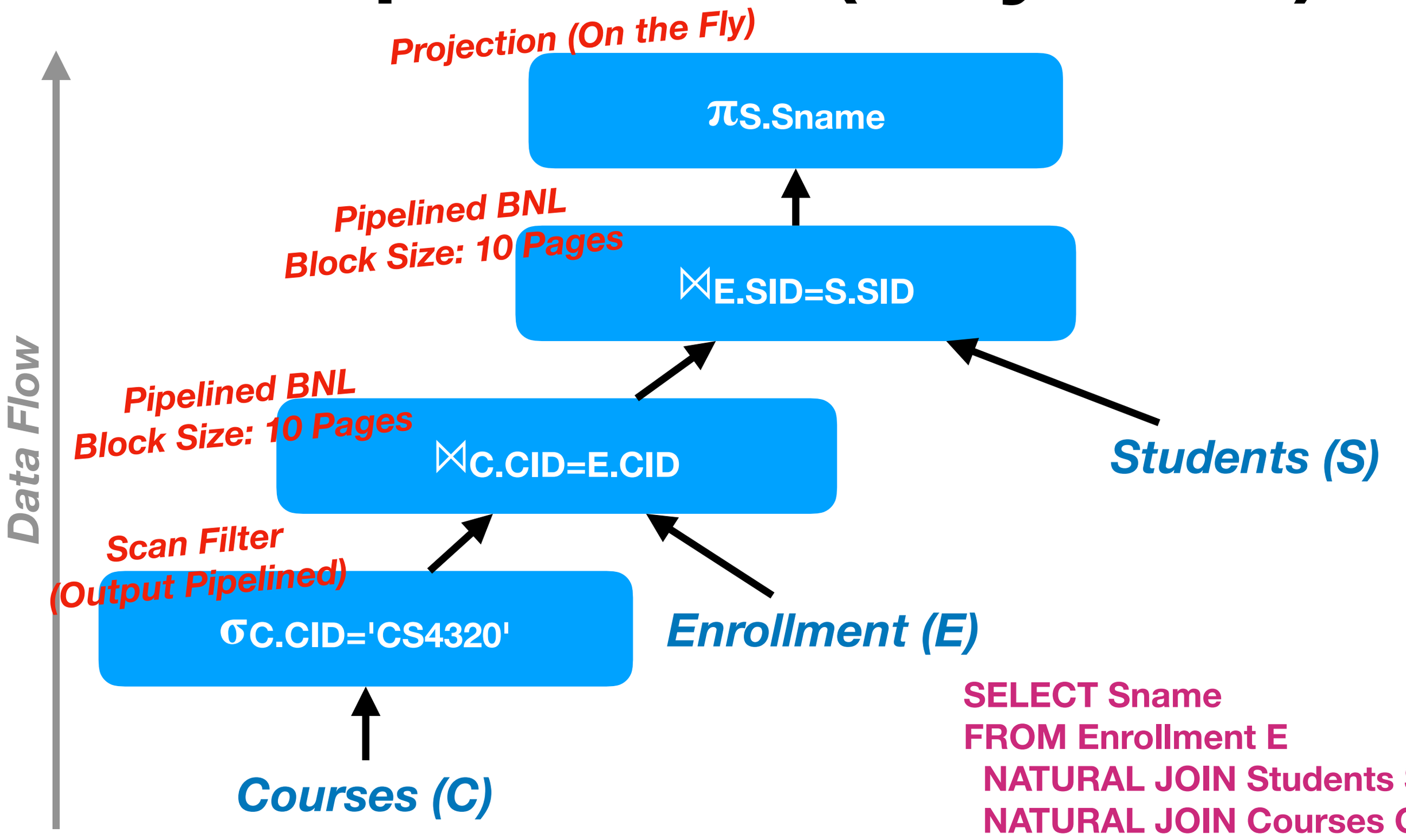
Passing Results Between Operators

- Simplest version: each operator **writes output to disk**
- May lead to unnecessary read/write **overheads**!
- Better: keep intermediate results in main **memory**
- This may not always be possible, depending on **size**
- **Physical plan** specifies how results are passed on
  - **Pipelined** operator passes result in-memory to next operation
  - Label "**On the fly**" for unary operators means pipelined input

Pipelined Nested Loop Joins

- Full join output may be **too large** to fit in memory
- Hence, produce small join **result parts** consecutively
- Directly invoke next operator for result part **in memory**
- Can easily **chain nested loop joins** in this way
  - Can start producing output with **part** of left input

Example Plan (Physical)

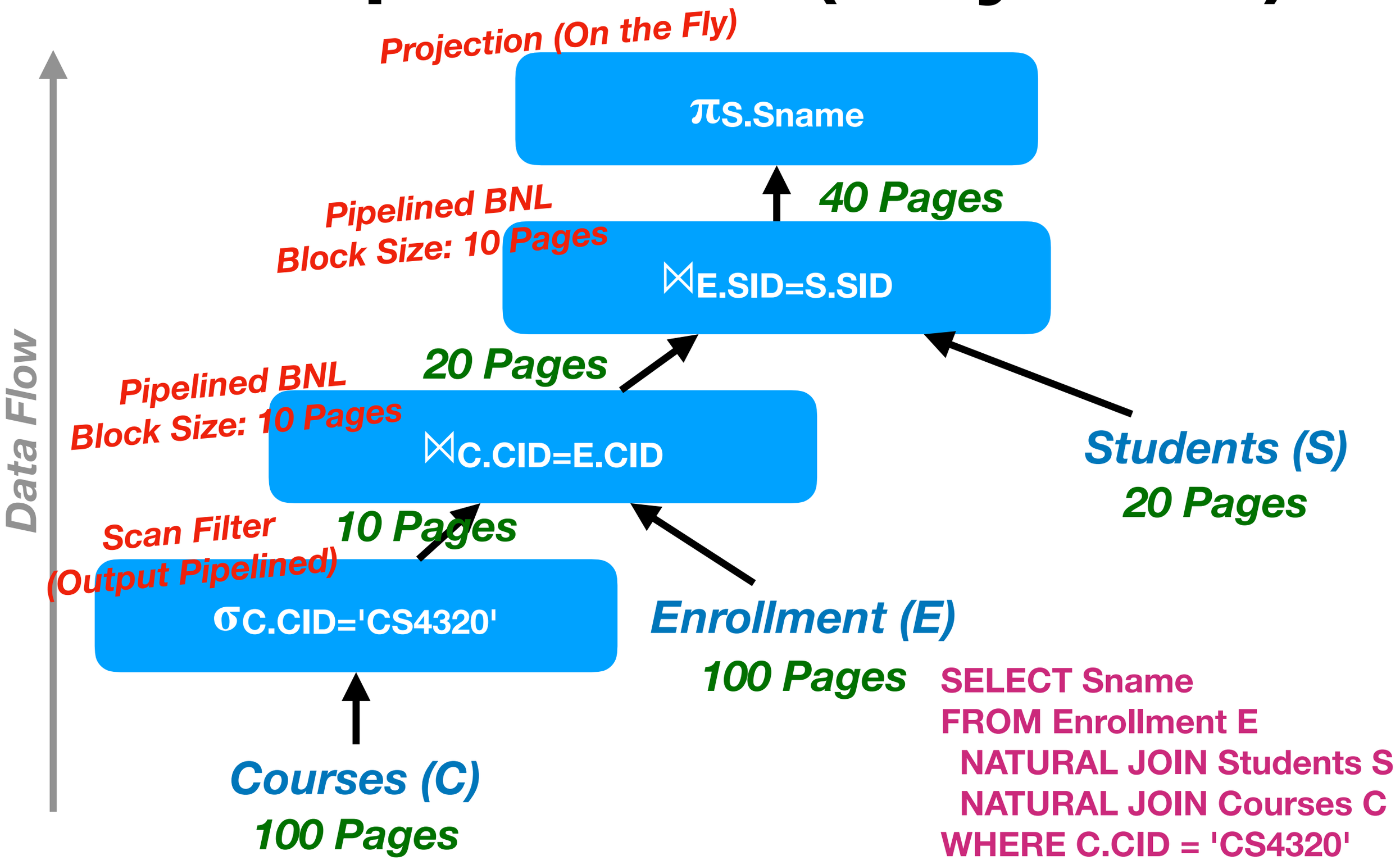



```
SELECT Sname
FROM Enrollment E
  NATURAL JOIN Students S
  NATURAL JOIN Courses C
WHERE C.CID = 'CS4320'
```

Plan Cost Estimation

- (Calculate intermediate **result sizes** if not given)
- Calculate **cost** of each operator
  - Take into account how data is **passed on**
- Do not count output cost of **final operator**
- **Sum up** cost of all operators in plan

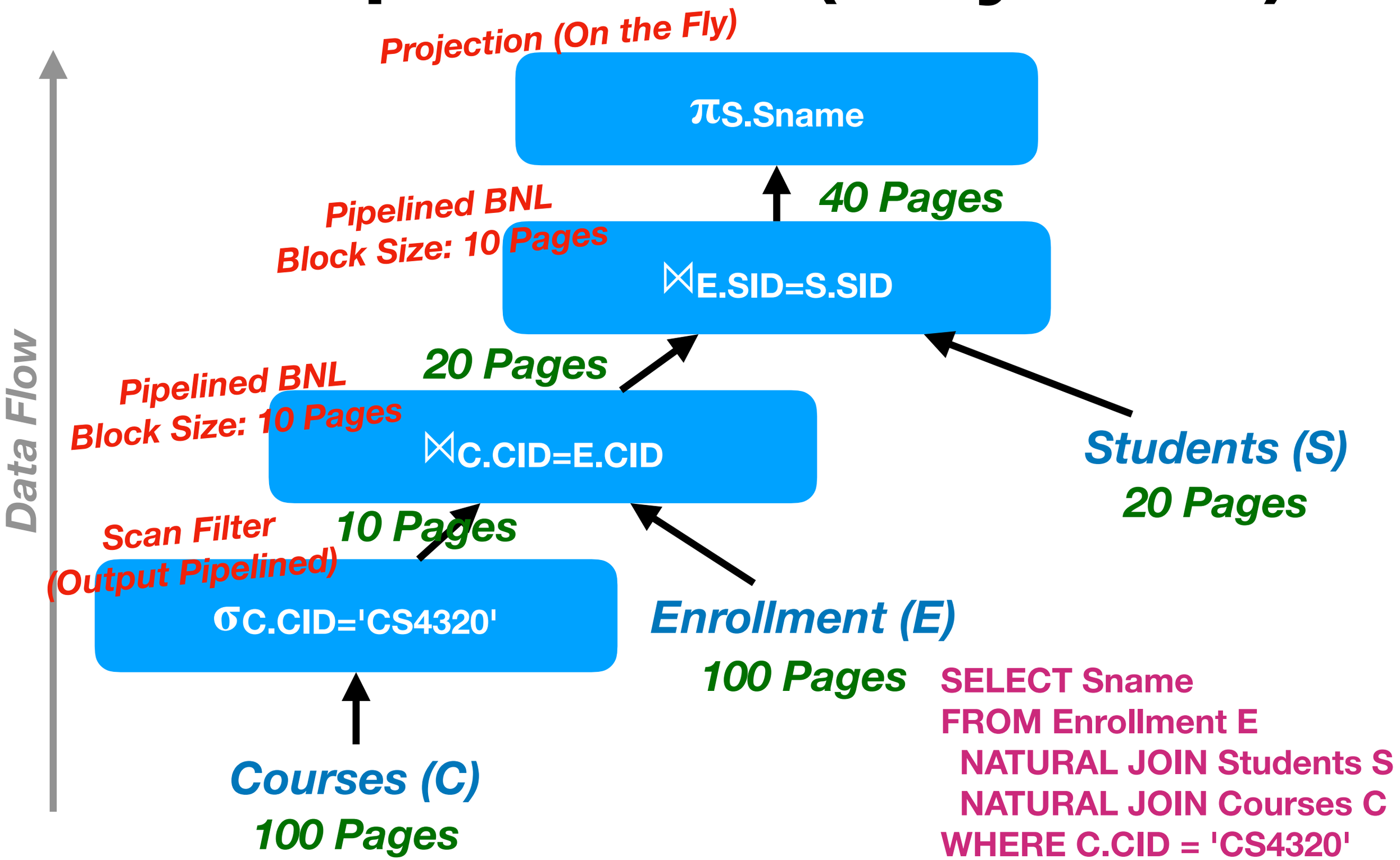
Example Plan (Physical)




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

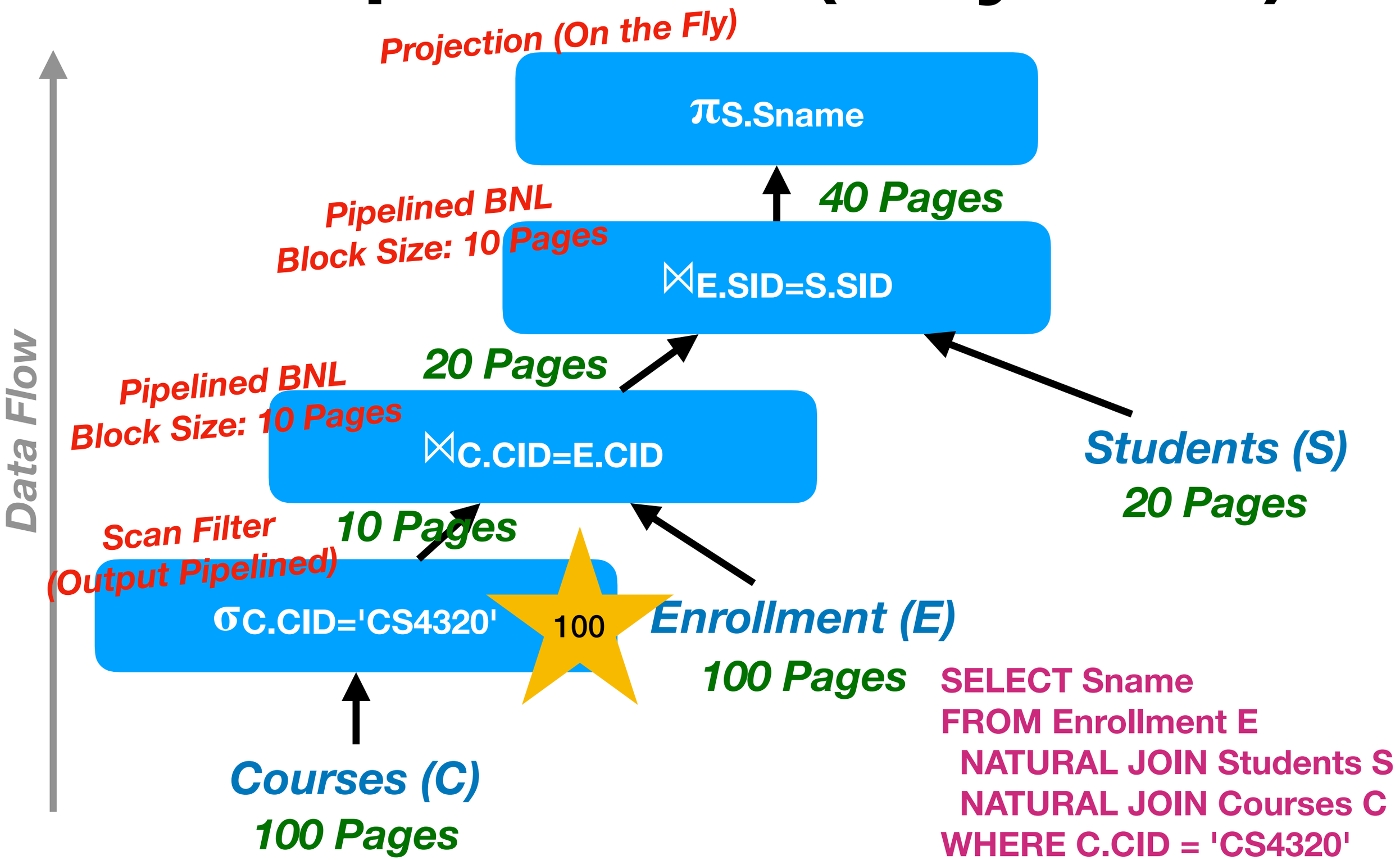
What is the Plan Execution Cost?

Example Plan (Physical)

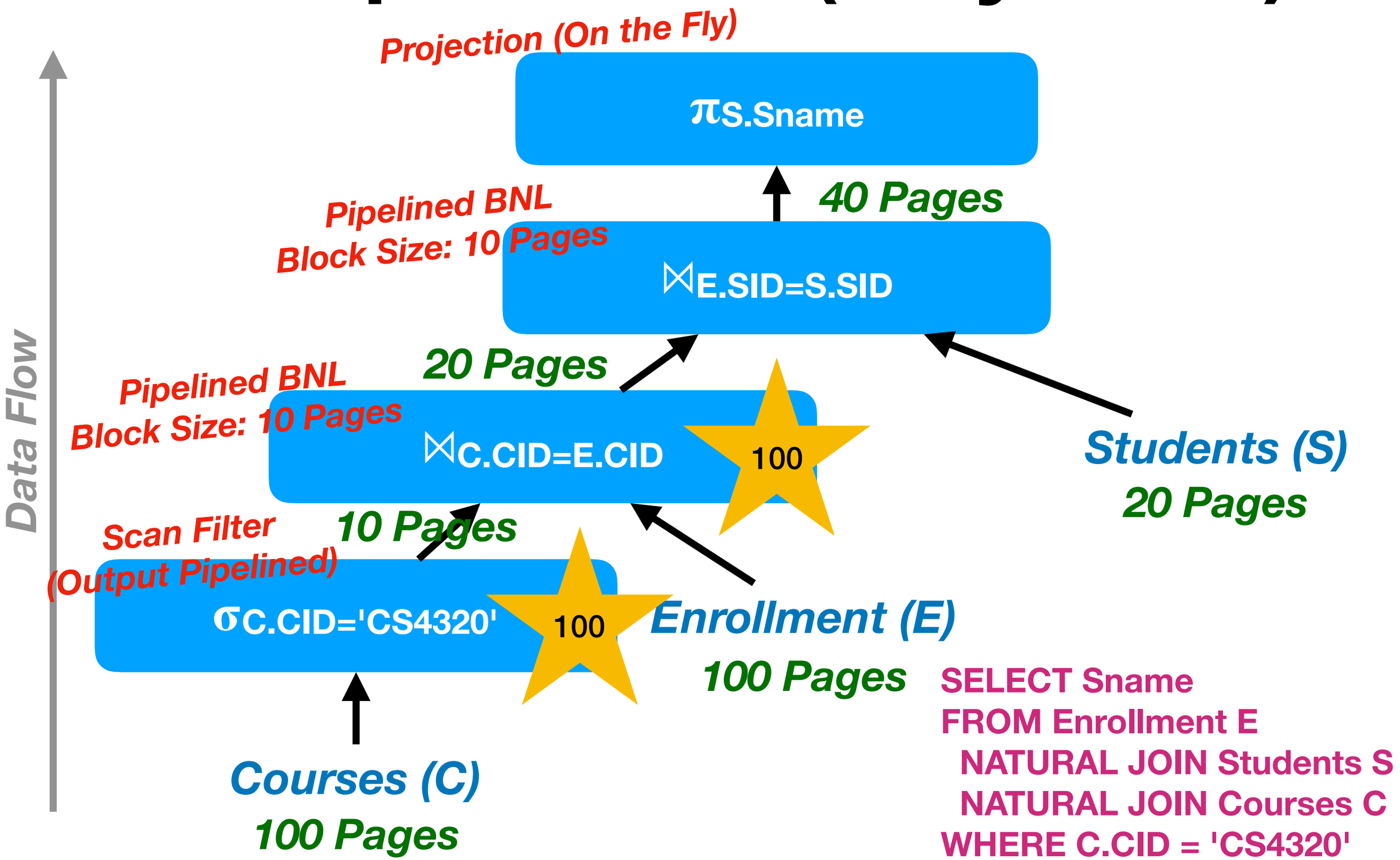



```
SELECT Sname
FROM Enrollment E
  NATURAL JOIN Students S
  NATURAL JOIN Courses C
WHERE C.CID = 'CS4320'
```

Example Plan (Physical)

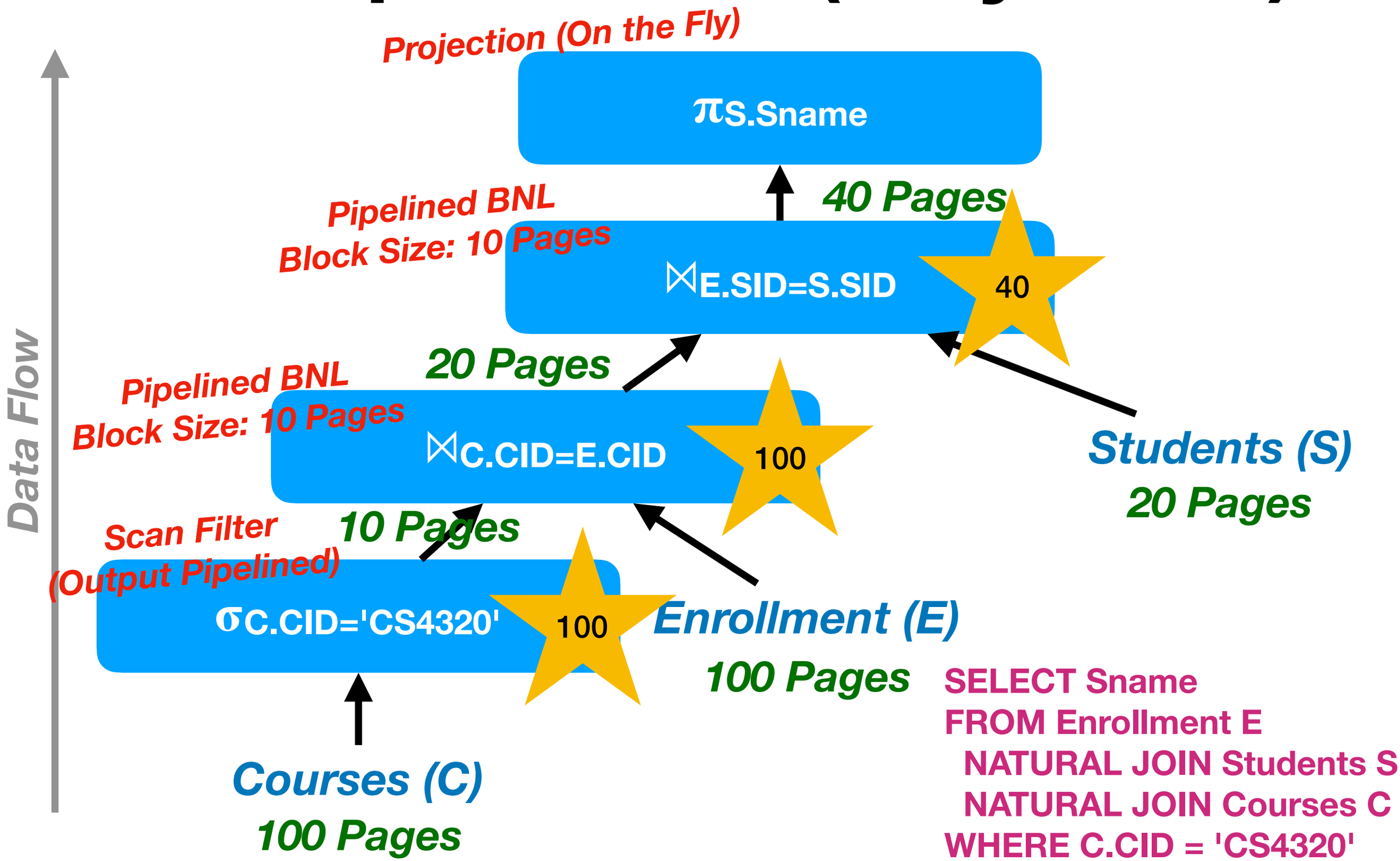


Example Plan (Physical)



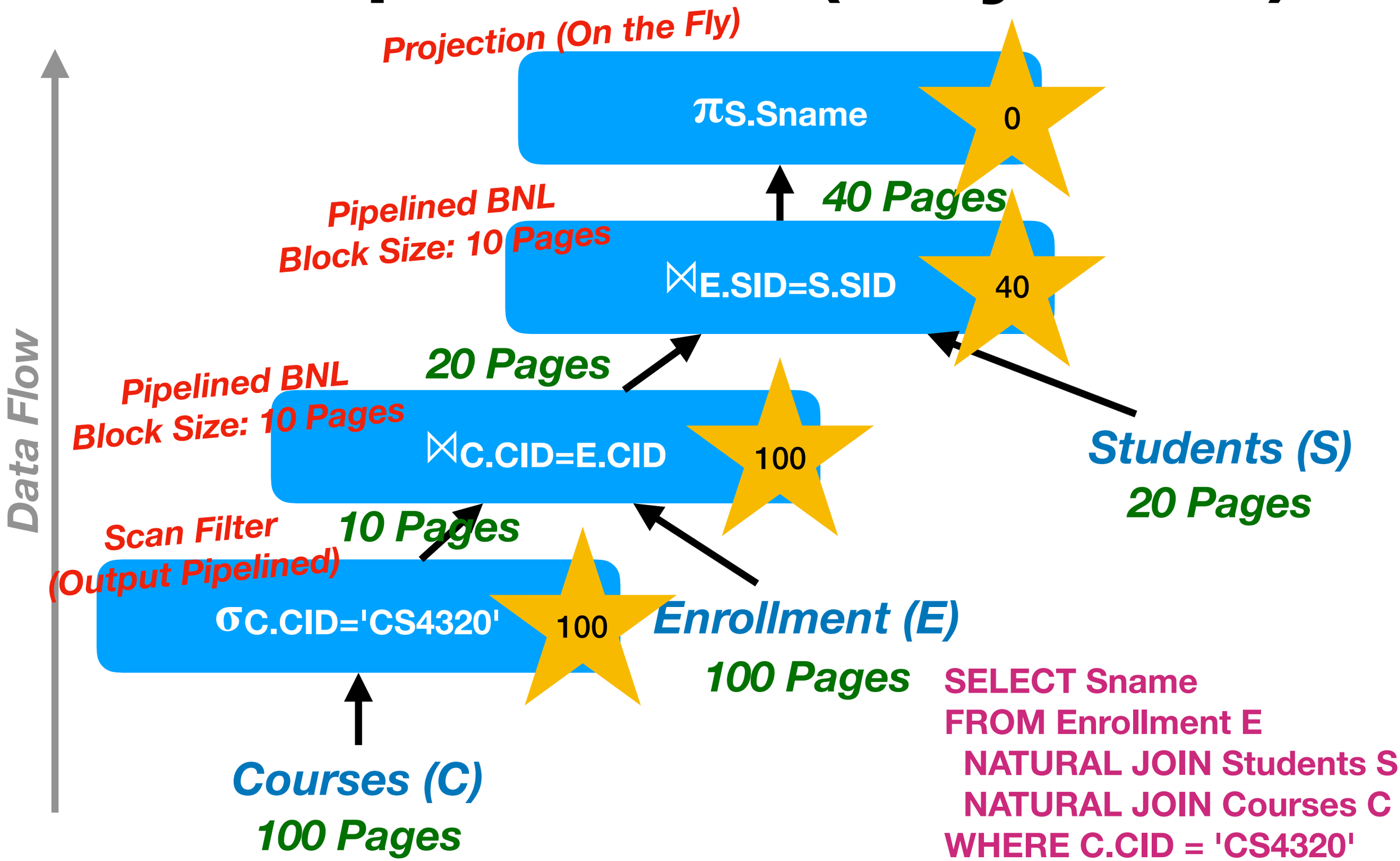

```
SELECT Sname
FROM Enrollment E
  NATURAL JOIN Students S
  NATURAL JOIN Courses C
WHERE C.CID = 'CS4320'
```

Example Plan (Physical)




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

Example Plan (Physical)




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

Example Plan (Physical)