

CSE 132C

Database System Implementation

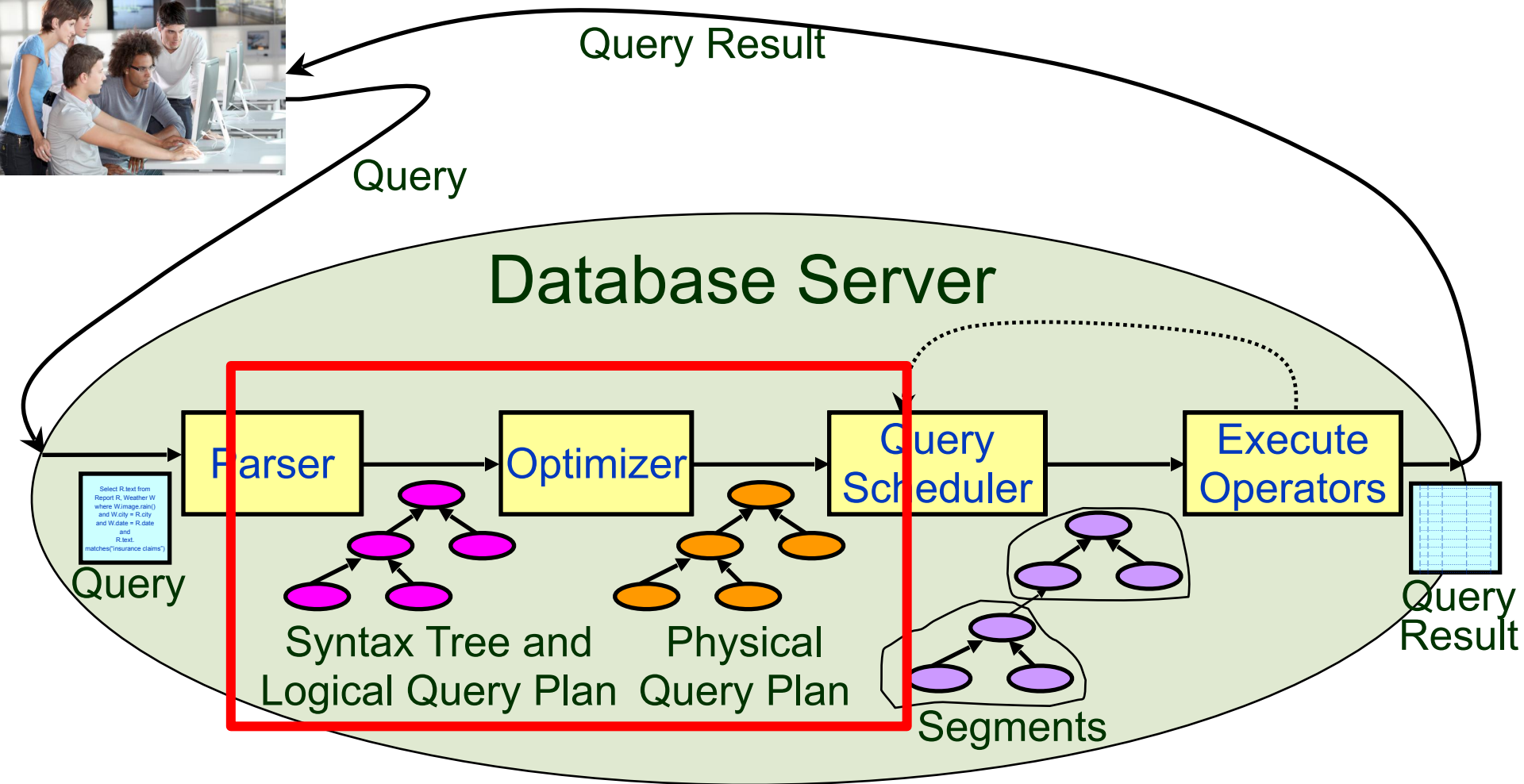
Arun Kumar

Topic 5: Query Optimization

Chapters 12.4-12.6 and 15.2-15.3 of Cow Book

Slide ACKs: Jignesh Patel

Lifecycle of a Query



Recall the Netflix Schema

Ratings

<u>RatingID</u>	Stars	RateDate	UID	MID
1	3.5	08/27/15	79	20
...	...	...	...	...

<u>UID</u>	Name	Age	JoinDate	Users
79	Alice	23	01/10/13	
80	Bob	41	05/10/13	

Movies

<u>MID</u>	Name	Year	Director
20	Inception	2010	Christopher Nolan
16	Avatar	2009	Jim Cameron

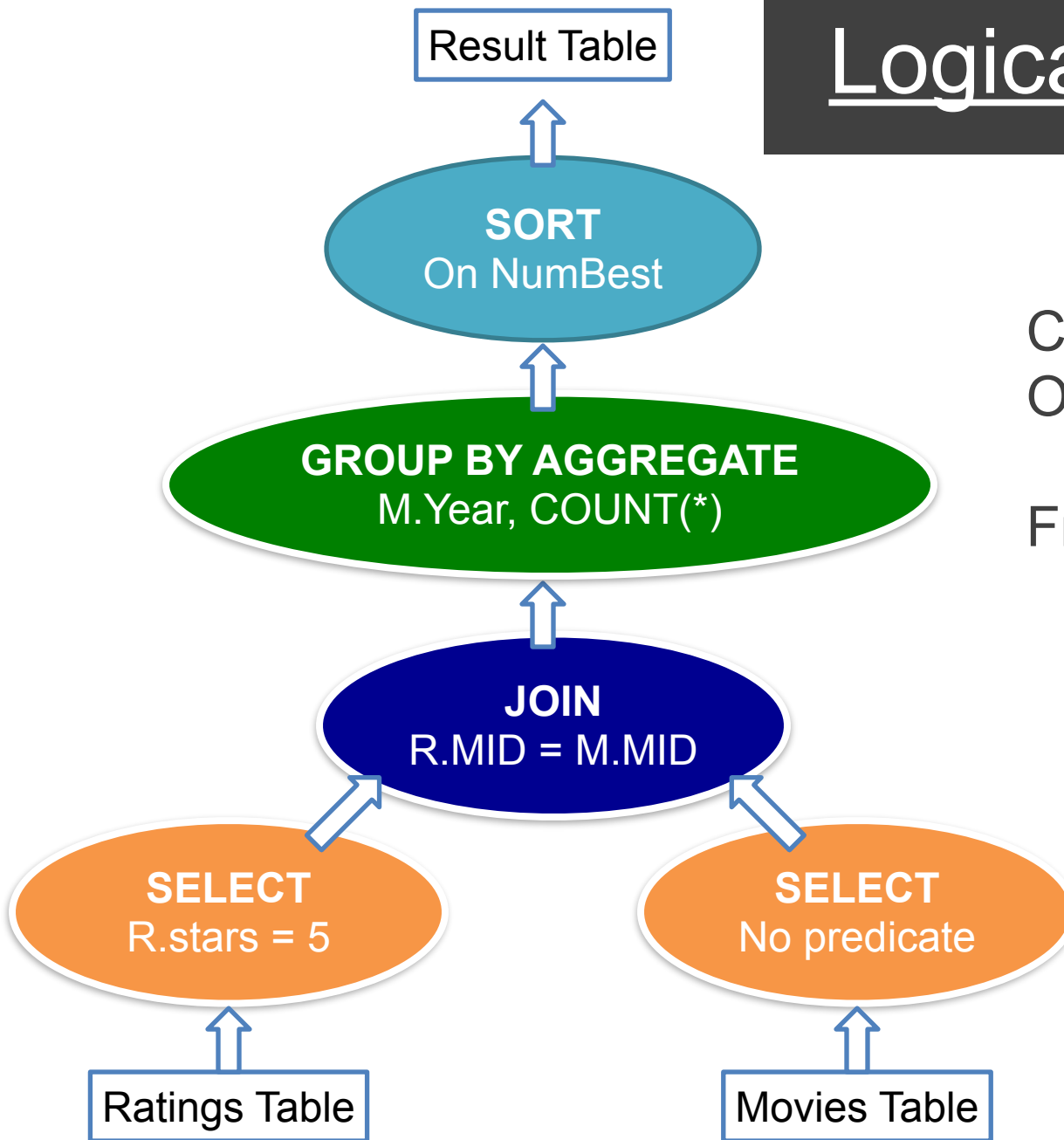
Example SQL Query

<u>RatingID</u>	Stars	RateDate	UID	MID
<u>UID</u>	Name	Age	JoinDate	
<u>MID</u>	Name	Year	Director	

```
SELECT      M.Year, COUNT(*) AS NumBest
FROM        Ratings R, Movies M
WHERE       R.MID = M.MID
            AND R.Stars = 5
GROUP BY    M.Year
ORDER BY    NumBest DESC
```

Suppose, we also have a B+Tree Index on Ratings (Stars)

Logical Query Plan

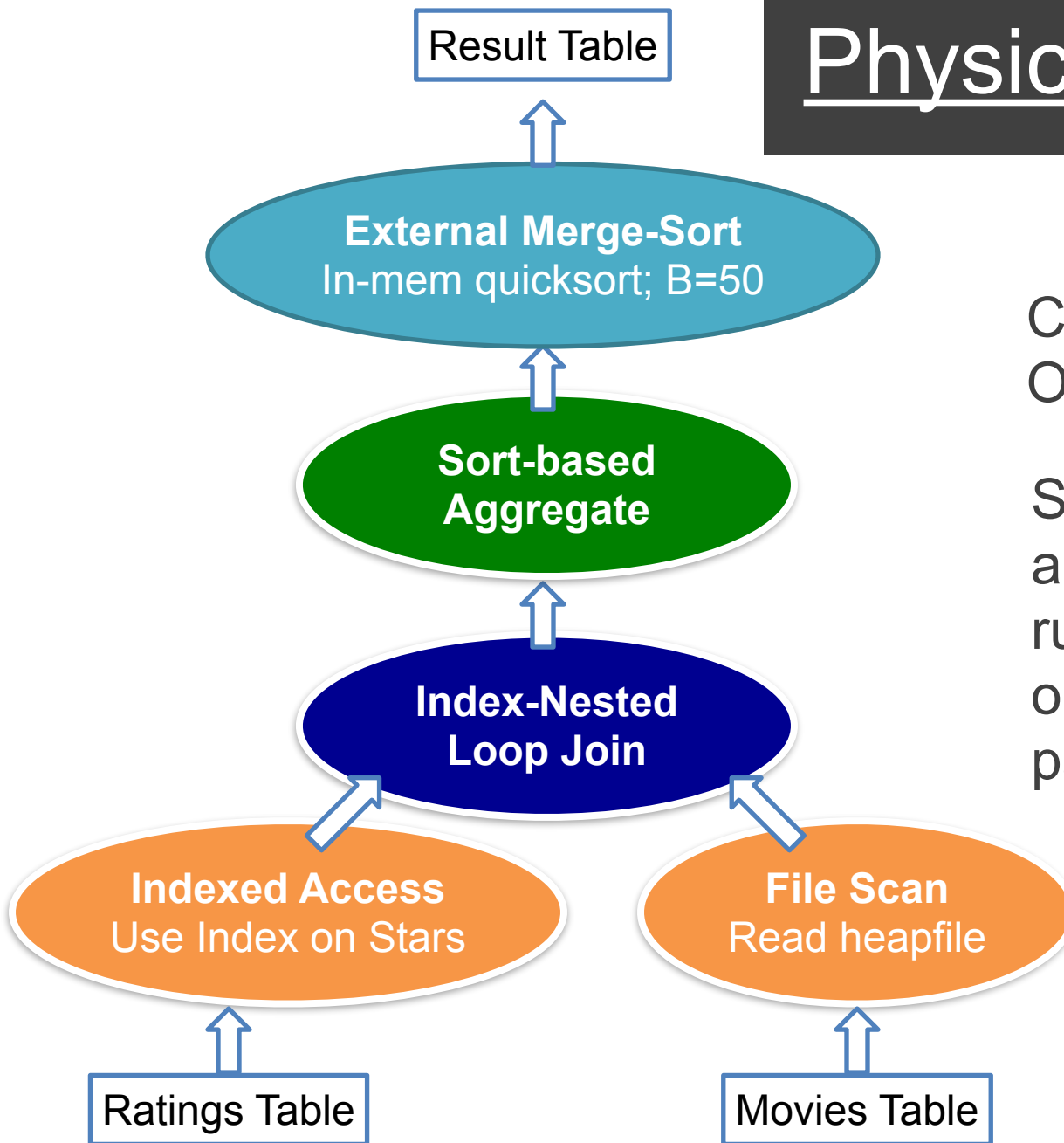


Called “**Logical**”
Operators

From extended RA

Each one has
alternate “physical”
implementations

Physical Query Plan

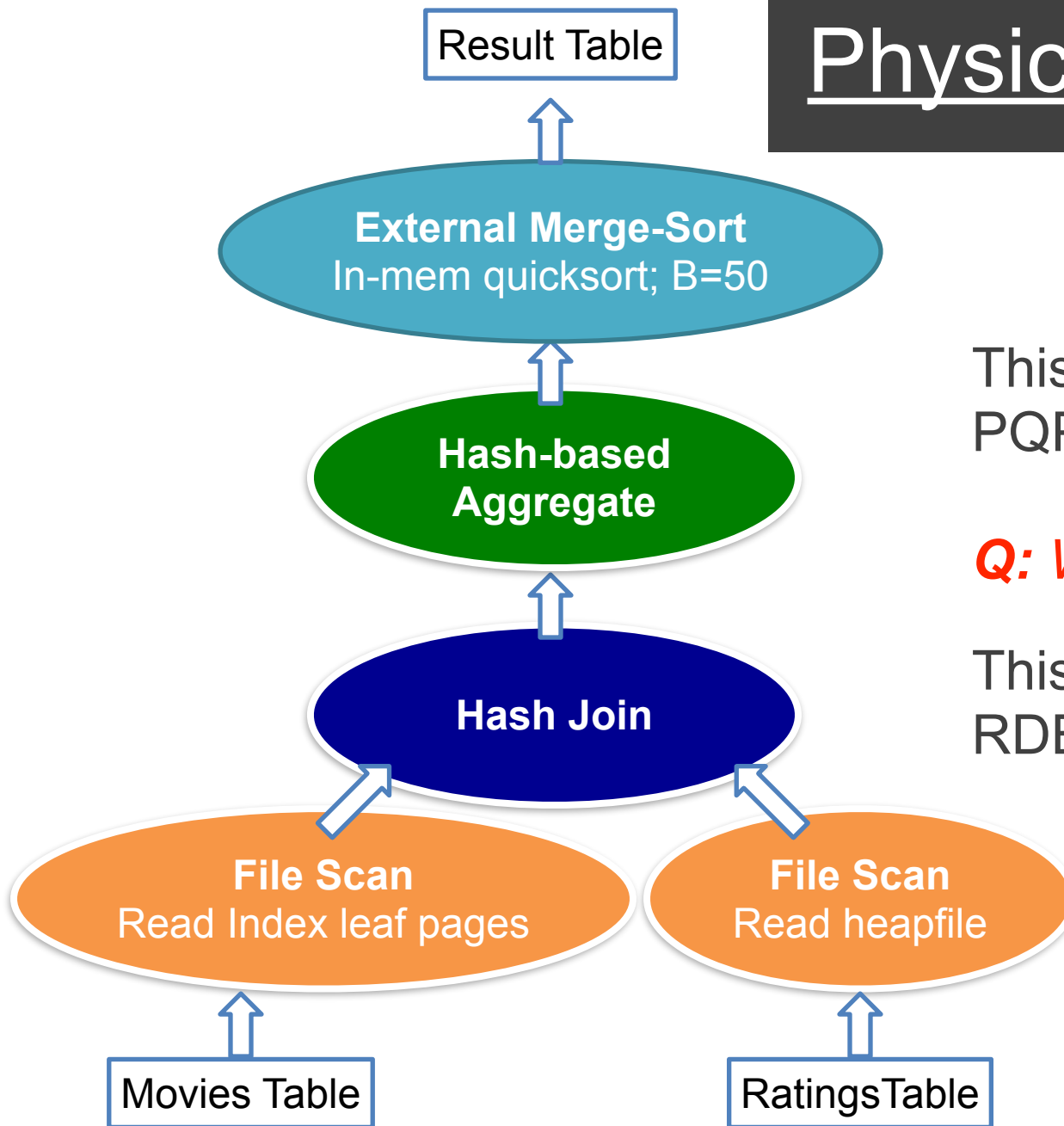


Called “**Physical**” Operators

Specifies exact algorithm/code to run for each logical operator, with all parameters (if any)

Aka “**Query Evaluation Plan**”

Physical Query Plan



This is also a correct PQP for the given LQP!

Q: Which PQP is faster?

This is a key job of the RDBMS Query Optimizer!

So, what is query optimization and how does it work?

Meet Query Optimization

Basic Idea: A given LQP could have several possible PQPs with very different runtime performance

Goal (Ideal): Get the optimal (fastest) PQP for a given LQP

Goal (Realistic): Find a PQP that is not obviously bad! Ps!



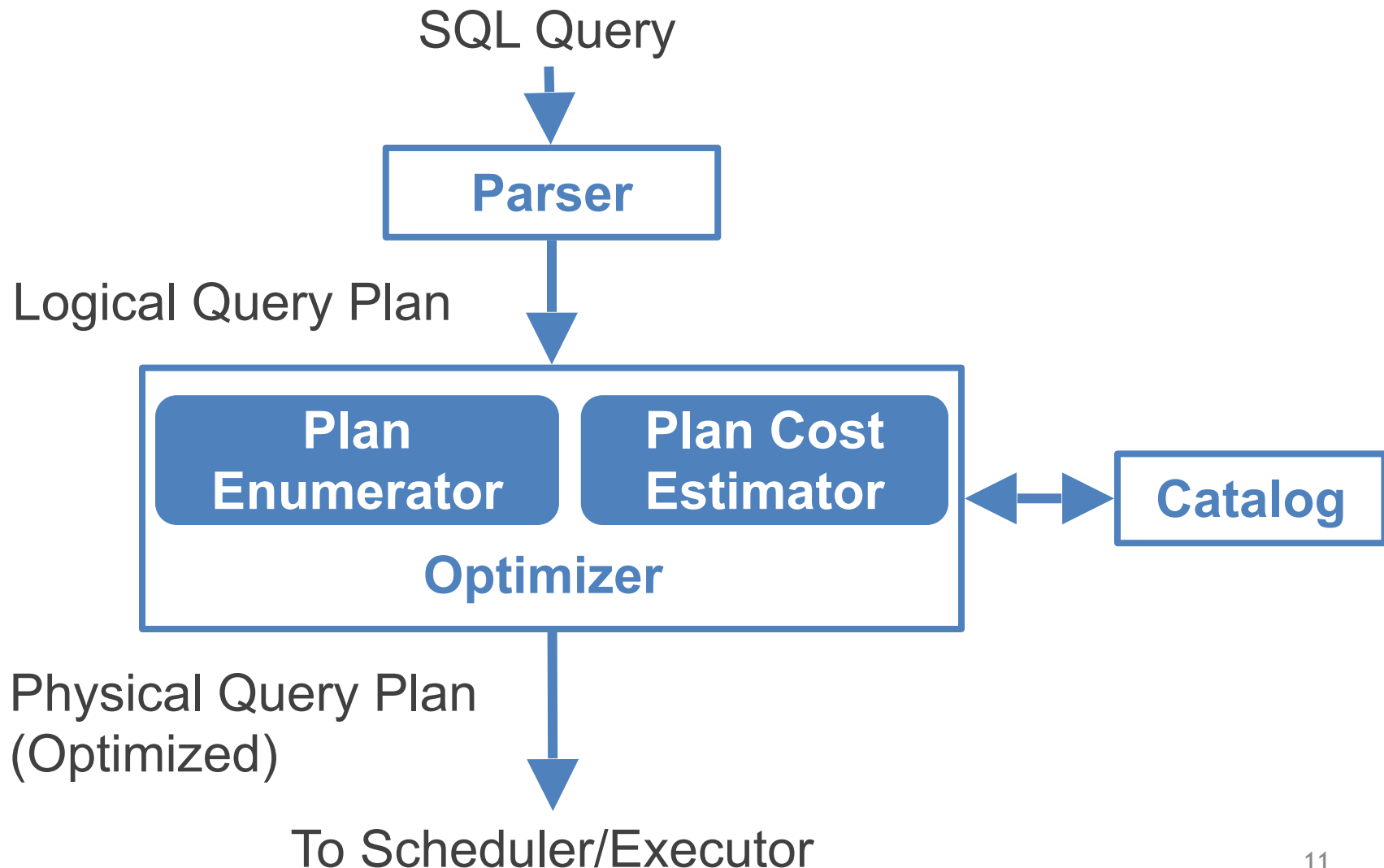
Jeff Naughton



Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Overview of Query Optimizer



System Catalog

- ❖ Set of pre-defined relations for metadata about DB (schema)
- ❖ For each **Relation**:
 - Relation name, File name
 - File structure (heap file vs. clustered B+ tree, etc.)
 - Attribute names and types; Integrity constraints; Indexes
- ❖ For each **Index**:
 - Index name, Structure (B+ tree vs. hash, etc.); Index key
- ❖ For each **View**:
 - View name, and View definition

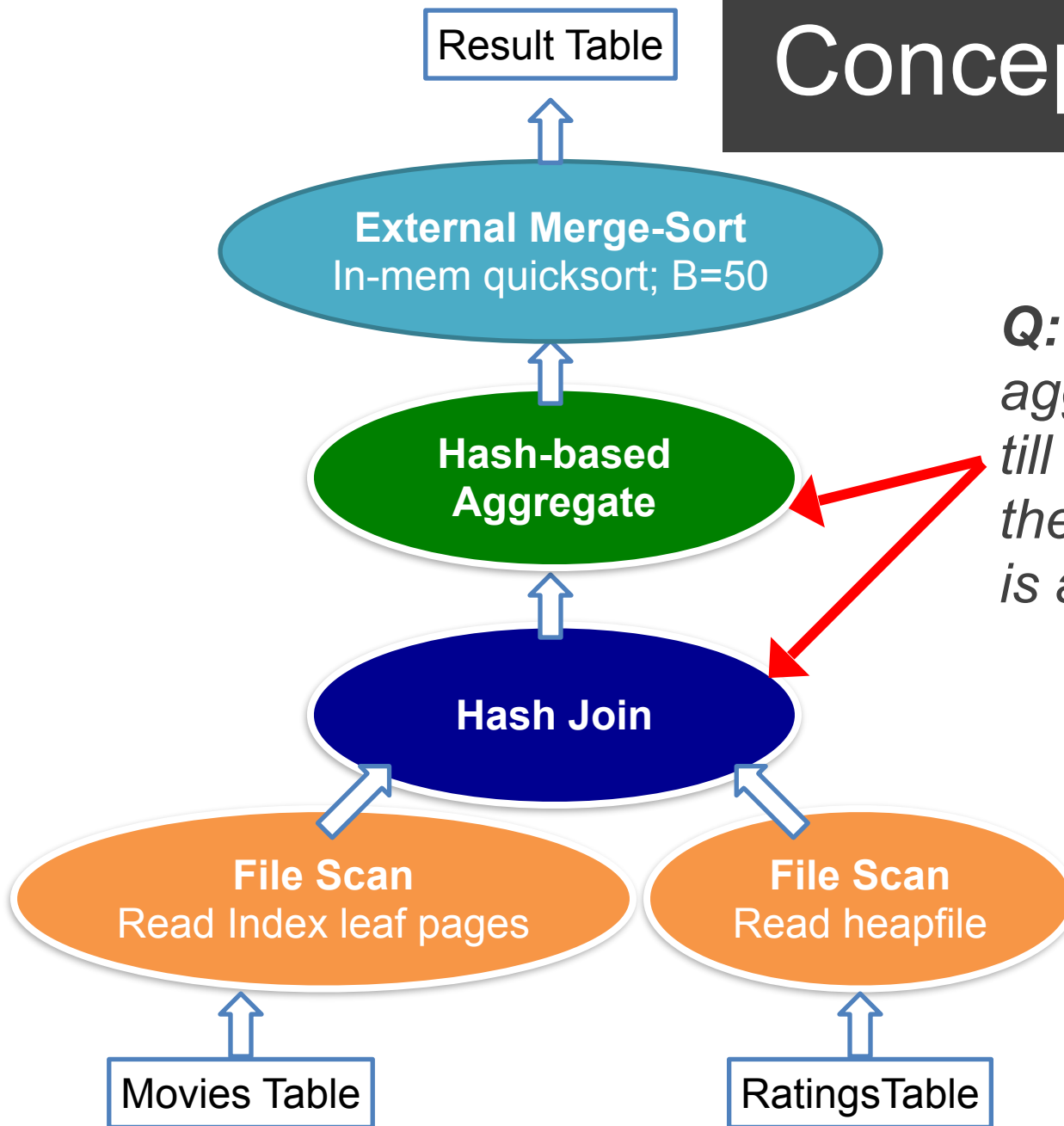
Statistics in the System Catalog

- ❖ RDBMS periodically collects stats about DB (instance)
- ❖ For each **Table R**:
 - Cardinality, i.e., number of tuples, **NTuples (R)**
 - Size, i.e., number of pages, **NPages (R)**, or just **N_R**
- ❖ For each **Index X**:
 - Cardinality, i.e., number of distinct keys **IKeys (X)**
 - Size, i.e., number of pages **IPages (X)** (for a B+ tree, this is the number of leaf pages only)
 - Height (for tree indexes) **IHeight (X)**
 - Min and max keys in index **ILow (X)**, **IHigh (X)**

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Concept: Pipelining



Q: Does the hash-based aggregate have to wait till the entire output of the “upstream” hash join is available?

No! We can “pipeline” the output of the join – pass on a join output tuple as soon as it is obtained!

Concept: Pipelining

Basic Idea: Do not force “downstream” physical operators to wait till the entire output is available

Benefits: Display output to the user incrementally
CPU Parallelism in multi-core systems!

Tuples



File Scan

Hash Join

Hash-based
Aggregate

Concept: Pipelining

- ❖ Crucial for PQPs with workflow of many phy. ops.
- ❖ Common feature of almost all RDBMSs
- ❖ Works for many operators: SCAN, HASH JOIN, etc.

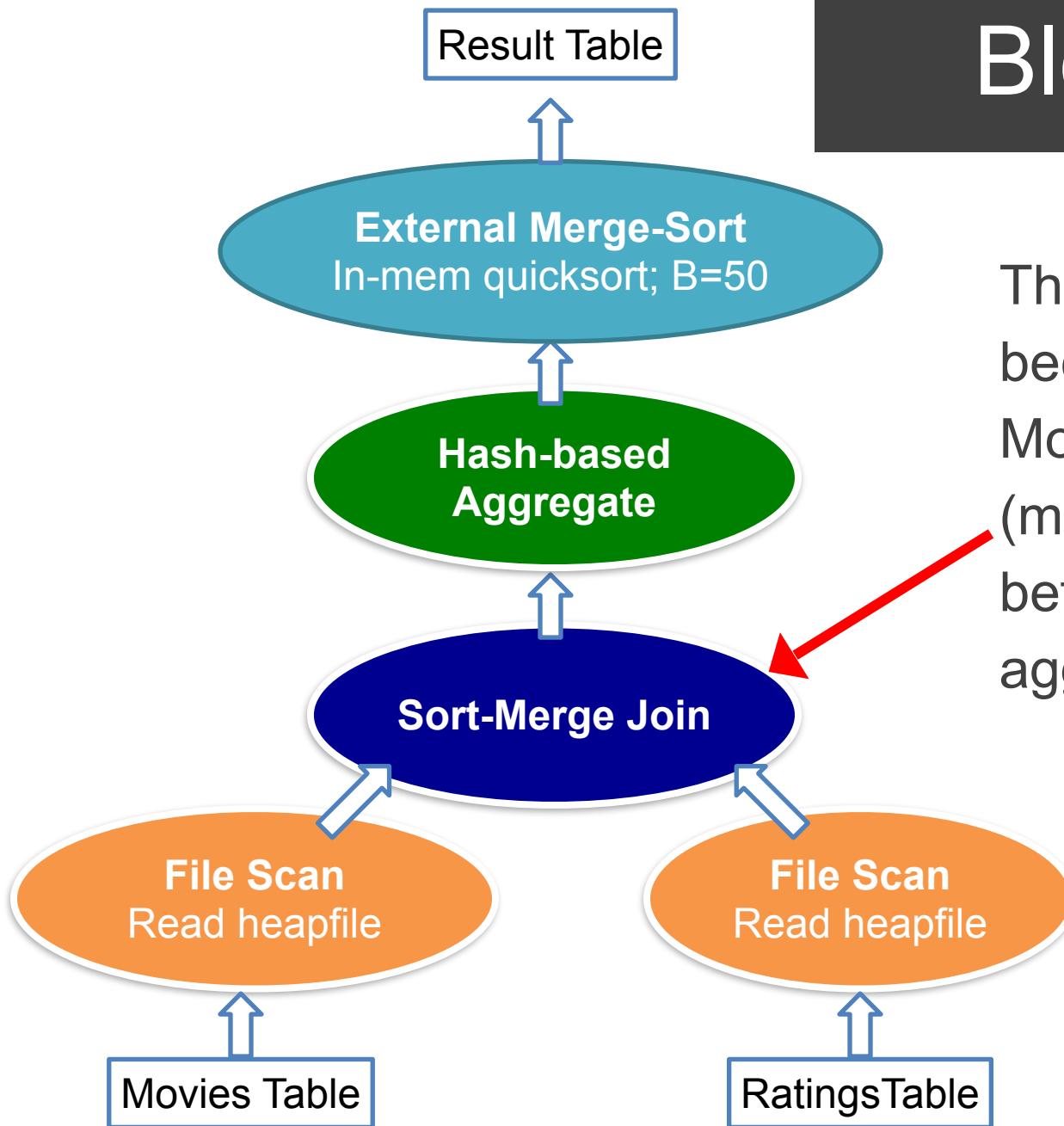
***Q:** Are all physical operators amenable to pipelining?*

No! Some may “stall” the pipeline: “**Blocking Op**”

A blocking op. requires its output to be **Materialized**
as a temporary table

Usually, any phy. op. involving sorting is blocking!

Blocking Op



This phy. op. is blocking because we need to sort Movies and sort Ratings (materialize the output) before we can start any aggregate computations!

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Mechanism: Iterator Interface

- ❖ Software API to process PQP; makes pipelining easy to impl.
- ❖ Enables us to abstract away individual phy. op. impl. details
- ❖ Three main functions in usage interface of each phy. op.:

Open(): Initialize the phy. op. “state”, get arguments
Allocate input and output buffers

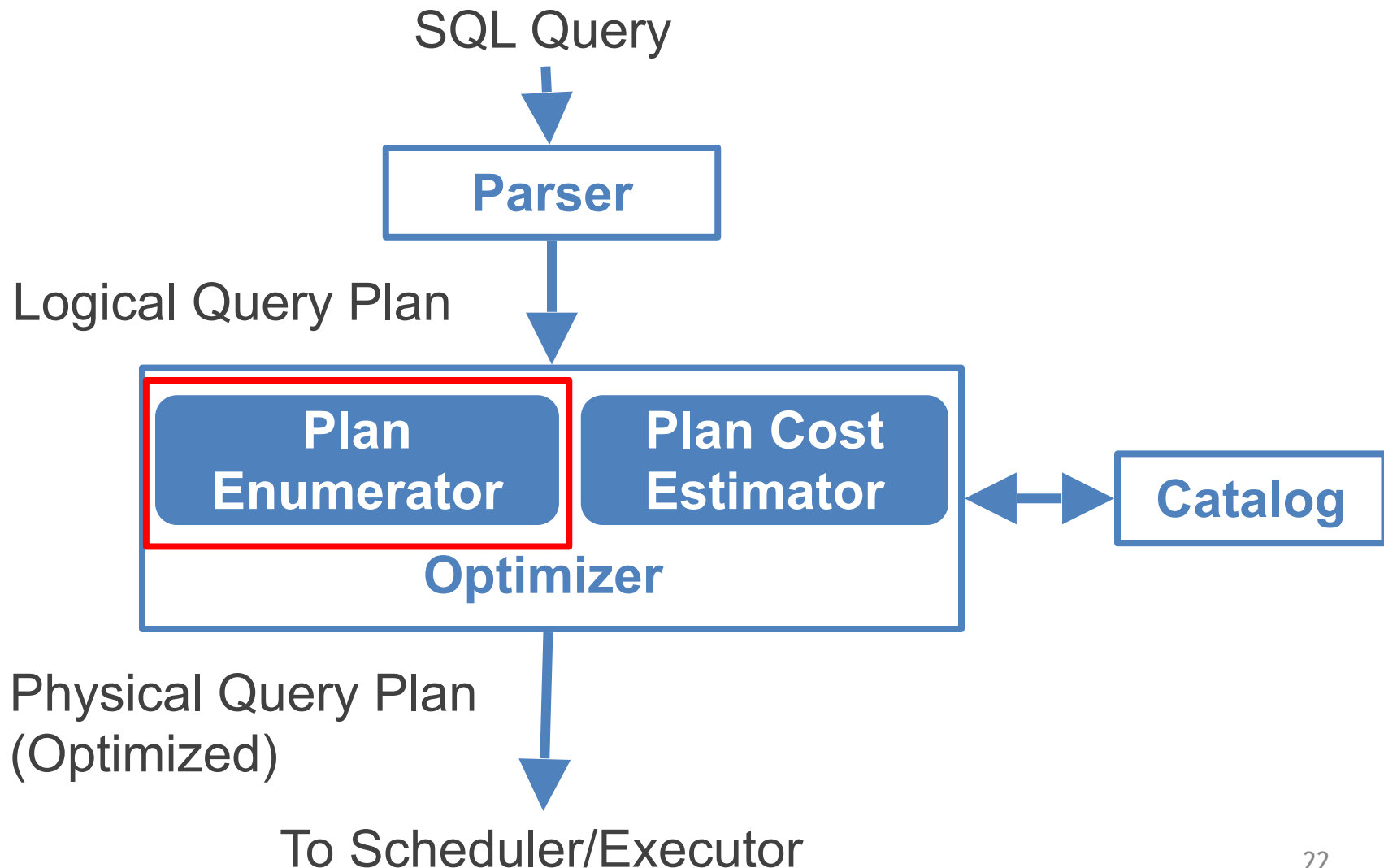
GetNext(): Ask the phy. op. impl. to “deliver” next
output tuple; pass it on; if blocking, wait

Close(): Clear phy. op. state, free up space

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Overview of Query Optimizer



Enumerating Alternative PQPs

- ❖ Plan Enumerator explores various PQPs for a given LQP
- ❖ **Challenge:** Space of plans is huge! How to make it feasible?
- ❖ RDBMS Plan Enumerator has **Rules** to help determine what plans to enumerate, and also consults **Cost models**
- ❖ Two main sources of Rules for enumerating plans:

Logical: Algebraic Rewrites:

Use relational algebra equivalence to rewrite LQP itself!

Physical: Choosing Phy. Op. Impl.:

Use different phy. op. impl. for a given log. op. in LQP

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Algebraic Rewrite Rules

- ❖ Rewrite a given RA query in to another that is equivalent (a logical property) but might be faster (a physical property)
- ❖ RA operators have some formal properties we can exploit
- ❖ We will cover only a few rewrite rules:

Single-operator Rewrites

Unary operators

Binary operators

Cross-operator Rewrites

Unary Operator Rewrites

❖ Key unary operators in RA: σ π

❖ Commutativity of σ

$$\sigma_{p_1}(\sigma_{p_2}(\mathbf{R})) = \sigma_{p_2}(\sigma_{p_1}(\mathbf{R}))$$

❖ Cascading of σ

$$\sigma_{p_1}(\sigma_{p_2}(\dots \sigma_{p_n}(\mathbf{R}) \dots)) = \sigma_{p_1 \wedge p_2 \wedge \dots \wedge p_n}(\mathbf{R})$$

❖ Cascading of π $A_i \subseteq A_{i+1} \forall i = 1 \dots (n - 1)$

$$\pi_{A_1}(\pi_{A_2}(\dots \pi_{A_n}(\mathbf{R}) \dots)) = \pi_{A_1}(\mathbf{R})$$

Q: *Why are cascading rewrites beneficial?*

Binary Operator Rewrites

- ❖ Key binary operator in RA: $\bowtie$
- ❖ Commutativity of $\bowtie$ $R \bowtie S = S \bowtie R$
- ❖ Associativity of $\bowtie$ $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Q: Why are these properties beneficial?

Q: What other binary operators in RA satisfy these?

Cross-Operator Rewrites

- ❖ Commuting σ and π $A \subseteq B$

$$\sigma_{p(A)}(\pi_B(R)) = \pi_B(\sigma_{p(A)}(R))$$

- ❖ Combining σ and $\times$

$$\sigma_p(R \times S) = R \bowtie_p S$$

- ❖ “Pushing the select” $A \subseteq R.*$

$$\sigma_{p(A)}(R \bowtie S) = \sigma_{p(A)}(R) \bowtie S$$

$$\sigma_{p(A)}(R \times S) = \sigma_{p(A)}(R) \times S$$

- ❖ Commuting π with $\times$ and $\bowtie$

$$\pi_A(R \times S) = \pi_{A \cap R.*}(R) \times \pi_{A \cap S.*}(S) \quad B \subseteq A$$

$$\pi_A(R \bowtie_{p(B)} S) = \pi_{A \cap R.*}(R) \bowtie_{p(B)} \pi_{A \cap S.*}(S)$$

Review Question

❖ Which of the following hold?

$$\pi_A(R \times S) = \pi_A(R) \times S \quad A \subseteq R.*$$

$$C = A \cup B$$

$$\pi_A(R \bowtie_{p(B)} S) = \pi_A(\pi_{C \cap R.*}(R) \bowtie_{p(B)} \pi_{C \cap S.*}(S))$$

$$\sigma_{p_1 \wedge p_2 \vee p_3}(R) = \sigma_{p_1}(R) \cap \sigma_{p_2}(R) \cup \sigma_{p_3}(R)$$

$$\sigma_{p(A) \wedge q(B)}(R \bowtie S) = \sigma_{p(A)}(R) \bowtie \sigma_{p(B)}(S)$$

$$A \subseteq R.* \text{ and } B \subseteq S.*$$

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Choosing Phy. Op. Impl.

- ❖ Given a (rewritten) LQP, pick phy. op. impl. for each log. op.
- ❖ Recall various RA op. impl. with their I/O (and CPU costs)

σ File scan vs Indexed (B+ Tree vs Hash)

π Hashing-based vs Sorting-based vs Indexed

$\bowtie$ BNLJ vs INLJ vs SMJ vs HJ

etc.

$\pi_B(\sigma_{p(A)}(R) \bowtie S)$

3 options 3 options 4 options = **36 PQPs!**

***Q:** With algebraic rewrites?!*

Phy. Op. Impl.: Other Factors

- ❖ Are the indexes clustered or unclustered?
- ❖ Are there multiple matching indexes? Use multiple?
- ❖ Are index-only access paths possible for some ops?
- ❖ Are there “interesting orderings” among the inputs?
- ❖ Would sorted outputs benefit downstream ops?
- ❖ Estimation of cardinality of intermediate results!
- ❖ How best to reorder multi-table joins?

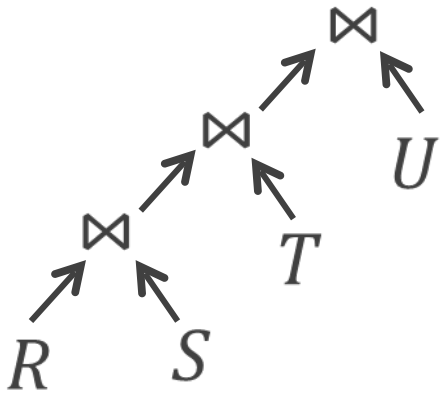
Query optimizers are complex beasts!

*Still a hard, open
research problem!*

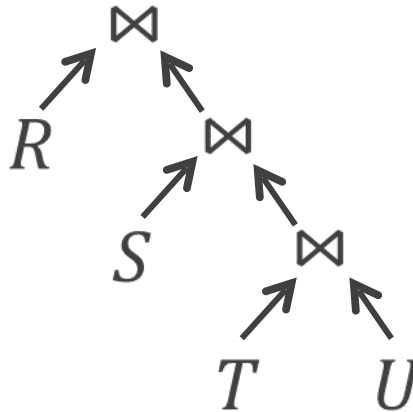
Phy. Op. Impl.: Join Orderings

- ❖ Since joins are associative, exponential number of orderings!

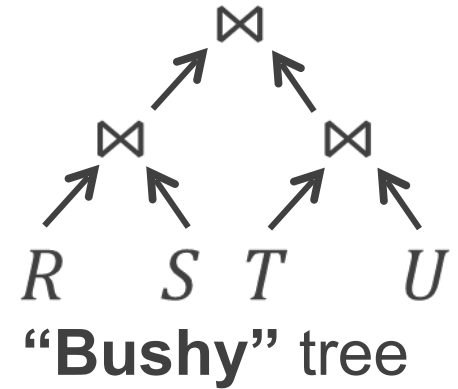
$$R \bowtie S \bowtie T \bowtie U$$



Left Deep tree



Right Deep tree



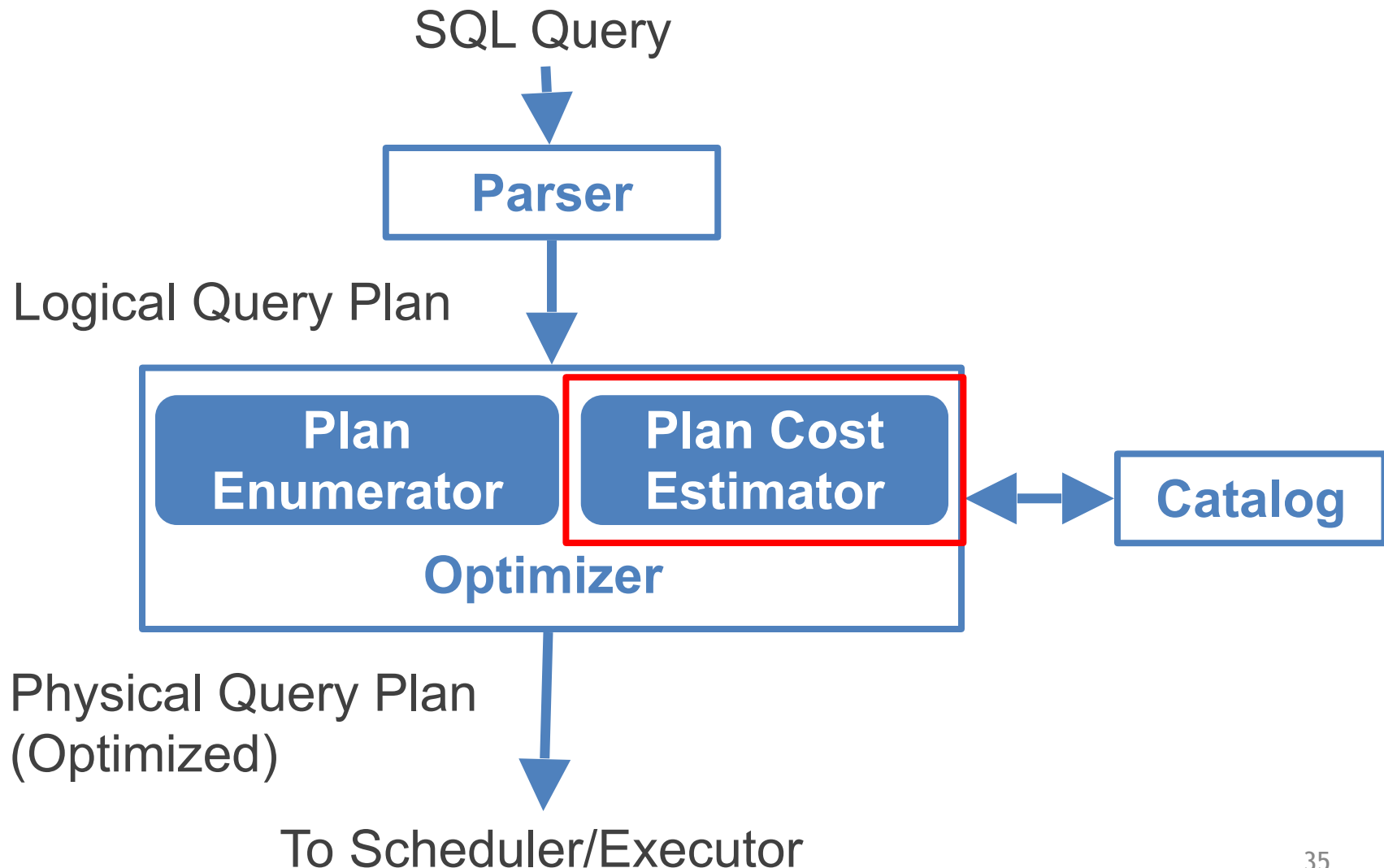
“Bushy” tree

- ❖ Almost all RDBMSs consider only left and right deep join trees
Enables easy pipelining! Why?
- ❖ “Interesting orderings” idea from System R optimizer paper
- ❖ Dynamic program to combine enumeration and costing

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Overview of Query Optimizer



Costing PQPs

- ❖ For each PQP considered by the Plan Enumerator, the Plan Cost Estimator computes “**Cost**” of the PQP

Weighted sum of I/O cost and CPU cost

(Distributed RDBMSs also include Network cost)

- ❖ **Challenge:** Given a PQP, compute overall cost

- ❖ **Issues to consider:**

Pipelining vs. blocking ops; cannot simply add costs!

Cardinality estimation for intermediate tables!

Q: What statistics does the catalog have to help?

Costing PQPs

- ❖ Most RDBMSs use various heuristics to make costing tractable; so, it is approximate!
- ❖ **Example:** Complex predicates

$$\sigma_{p_1 \wedge p_2}(R)$$

Suppose selectivity of p_1 is 5%
and selectivity of p_2 is 10%

Q. What is the selectivity of $p_1 \wedge p_2$? Not enough info!

But, most RDBMSs use the **independence** heuristic!

Selectivity of conjunction = Product of selectivities

Thus, $\approx 0.05 * 0.1 = 0.005$, i.e., 0.5%

Query Optimization: Summary

- ❖ Plan Enumerator and Cost Estimator work in lock step:

 - Rules** determine what PQPs are enumerated

 - Logical: Algebraic rewrites of LQP

 - Physical: Op. Impl. and ordering alternatives

 - Cost models** and **heuristics** help cost the PQPs

- ❖ Still an active research area!

 - Parametric Q.O., Multi-objective Q.O.,

 - Multi-objective parametric Q.O., Multiple Q.O.,

 - Online/Adaptive Q.O., Dynamic re-optimization, etc.

Query Optimization

- ❖ Overview of Query Optimizer
- ❖ Physical Query Plan (PQP)
 - Concept: Pipelining
 - Mechanism: Iterator Interface
- ❖ Enumerating Alternative PQPs
 - Logical: Algebraic Rewrites
 - Physical: Choosing Phy. Op. Impl.
- ❖ Costing PQPs

Review Question

RatingID

Stars

Date

UID

MID

20m pages

Page size 8KB; Buffer memory 8GB; 8B for each field

SELECT COUNT(DISTINCT Date) FROM Ratings

Propose an efficient physical plan and compute its I/O cost.

COUNT Aggregate

buffer frames = 1m

Hash-based

Read
 $N_T = 4m$

Pipeline COUNT with
deduplication phase

Project on Date

Write T to disk
 $N_T = 4m$

Cannot fit hash table on
T entirely; partition it

Ratings

Read $N_R = 20m$

Total I/O = **28m pages**

Review Question

RatingID

Stars

Date

UID

MID

20m pages

Page size 8KB; Buffer memory 8GB; 8B for each field

SELECT COUNT(DISTINCT Date) FROM Ratings

We are given a clustered AltRID B+ tree index on Date.
Propose an efficient physical plan and compute its I/O cost.

COUNT Aggregate

(Assume RecordID is 8B too)

Sort-based
Project on Date

ProjectionList is prefix of IndexKey
No GROUP BY; trivial pipelining
Overall, just one scan of leaf level

Index on Date

Leaf level size = $2 / 5 * N_R = 8m$ pages

Review Question

<u>RatingID</u>	Stars	Date	UID	MID
-----------------	-------	------	-----	-----

10m pages

Page size 8KB; Buffer memory 4GB; 8B for each field

```
SELECT COUNT(DISTINCT UID) FROM Ratings
```

Propose an efficient physical plan and compute its I/O cost.

Query plan is a Project on UID, followed by counting

Size of intermediate file T = 2m pages, i.e., 16 GB

Size of hash table on T: $F \times 16 \text{ GB} > 4 \text{ GB RAM}$

But RAM is enough for regular hash-based Project

Pipeline counting on top of output of Project

I/O cost = $10\text{m} + 2 \times 2\text{m} = 14\text{m pages}$

Review Question

<u>RatingID</u>	Stars	Date	UID	MID	10m pages
<u>MID</u>	Name	Year	Director	100k pages	

Page size 8KB; Buffer memory 4GB

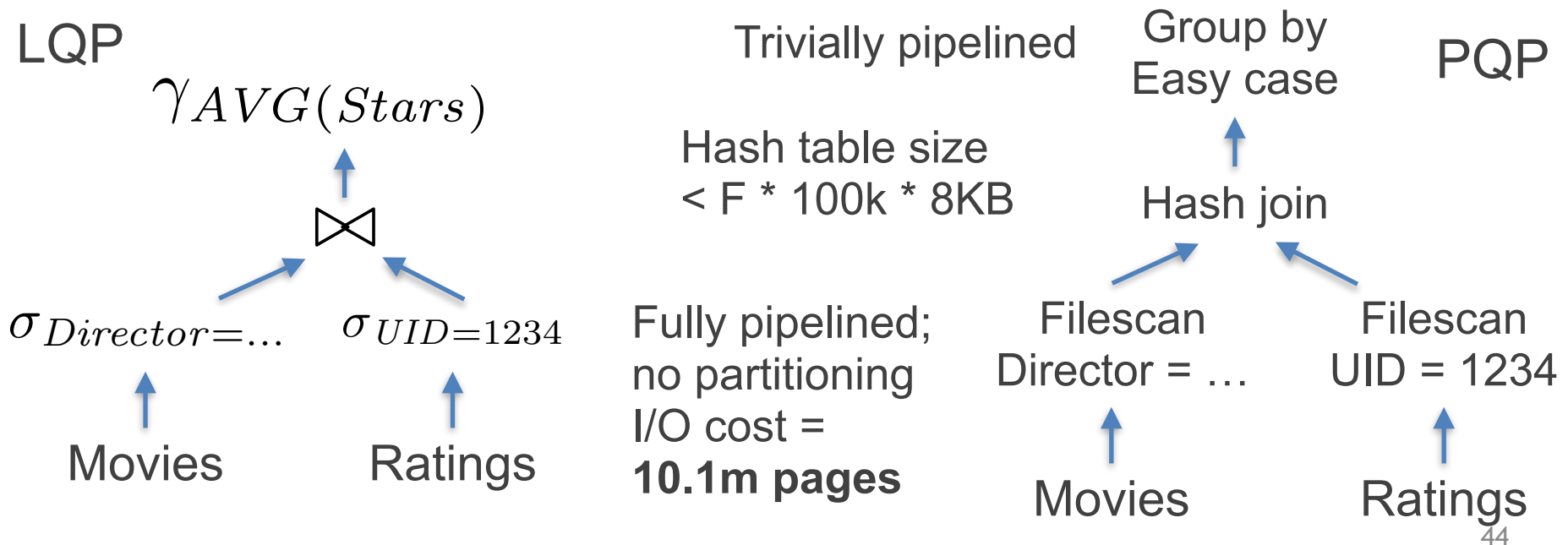
```
SELECT AVG(Stars) FROM Ratings R, Movies M
WHERE R.MID = M.MID AND
      M.Director = "Christopher Nolan" AND
      R.UID = 1234;
```

Propose an efficient physical plan that does not materialize any intermediate data (fully pipelined) and compute its I/O cost.

Review Question

```
SELECT AVG(Stars) FROM Ratings R, Movies M
WHERE R.MID = M.MID AND
      M.Director = "Christopher Nolan" AND
      R.UID = 1234;
```

8 KB pages; 4 GB RAM



Review Question

Example:

<u>RatingID</u>	Stars	Date	UID	MID	1b pages	
<u>UID</u>	Name	Age		<u>MID</u>	Name	Year Director
10m pages				100k pages		

Page size 8KB; node has 256GB RAM

```
SELECT      M.Year, AVG(R.Stars)
FROM        Ratings R, Movies M, Users U
WHERE       R.MID = M.MID AND R.UID = U.UID
            AND U.Age <= 30
GROUP BY    M.Year
```

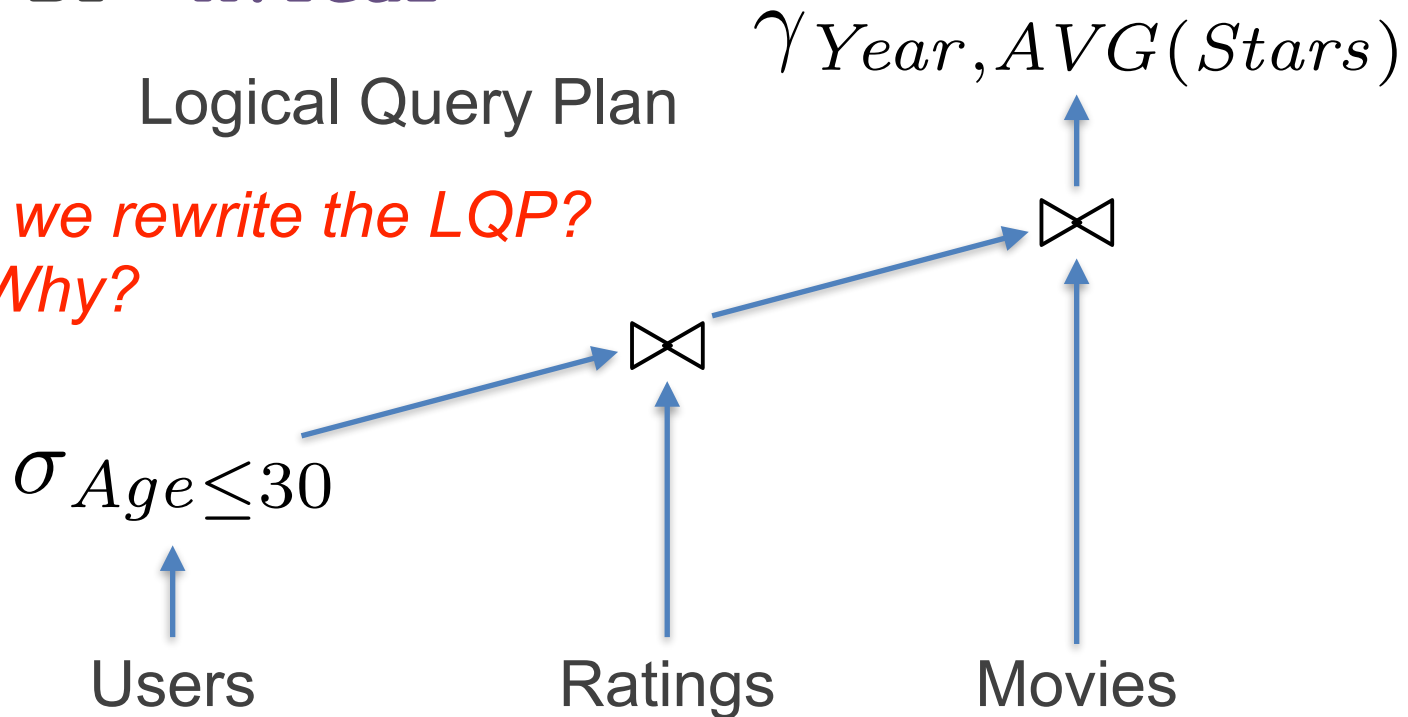
Propose an efficient physical plan that does not materialize any intermediate data (fully pipelined) and compute its I/O cost.

Review Question

```
SELECT      M.Year, AVG(R.Stars)
FROM        Ratings R, Movies M, Users U
WHERE       R.MID = M.MID AND R.UID = U.UID
            AND U.Age <= 30
GROUP BY    M.Year
```

Logical Query Plan

*Should we rewrite the LQP?
How? Why?*



Review Question

$\gamma Year, AVG(Stars)$

What about this group by?

LQP with right deep join tree

What physical op to use for each logical op? Pipelined?

Can hash tables fit in RAM?



$\sigma Age \leq 30$

Movies

Users

Ratings

100k pages

10m pages

1b pages

= $F * 100k * 8KB$

< $F * 10m * 8KB$

Page size 8KB

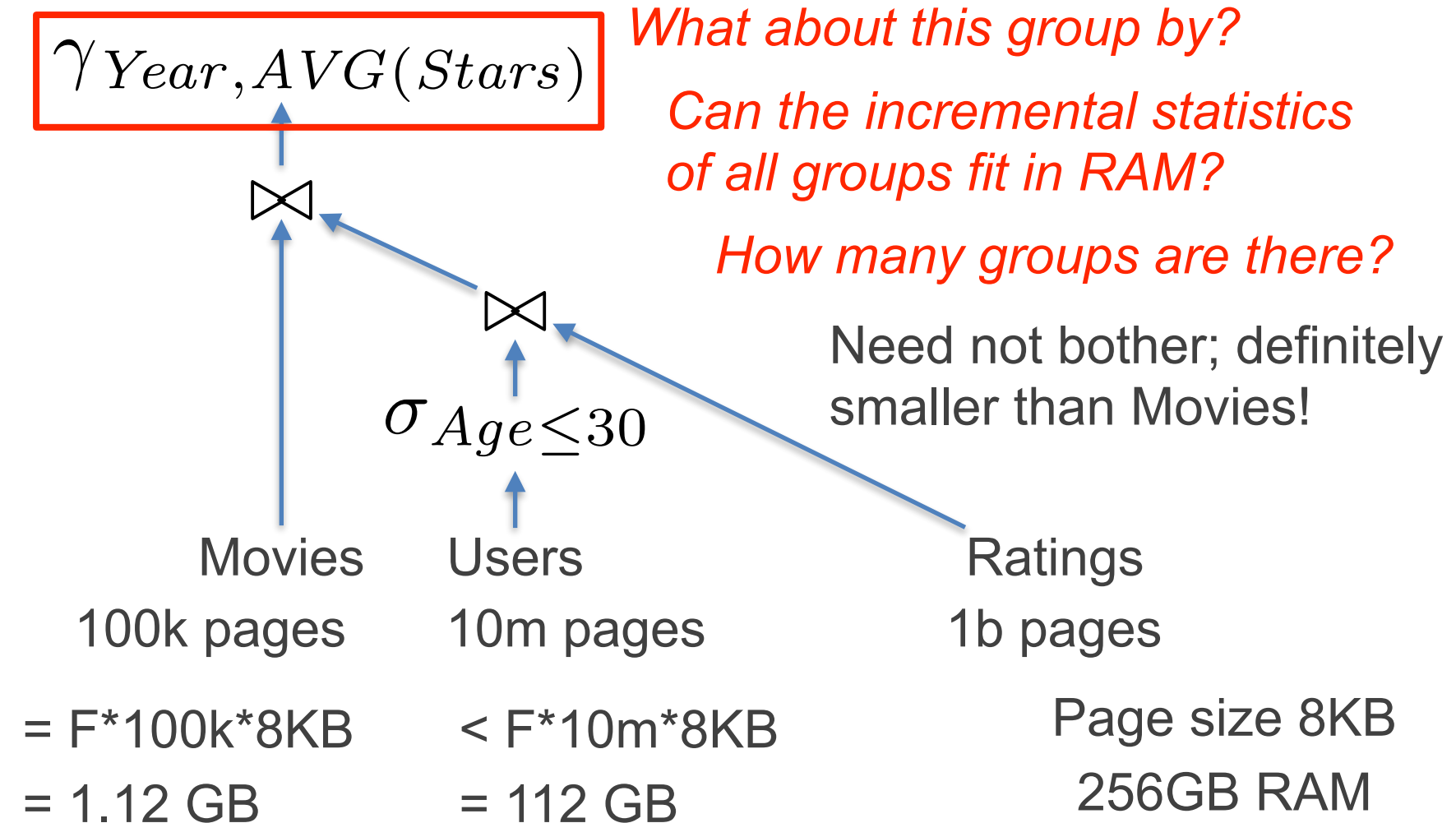
= 1.12 GB

= 112 GB

256GB RAM

Both hash tables fit in RAM together; fully pipelined join!

Review Question



So, hash-based group by aggregate; partitioning not needed

Review Question

