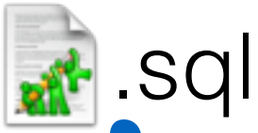
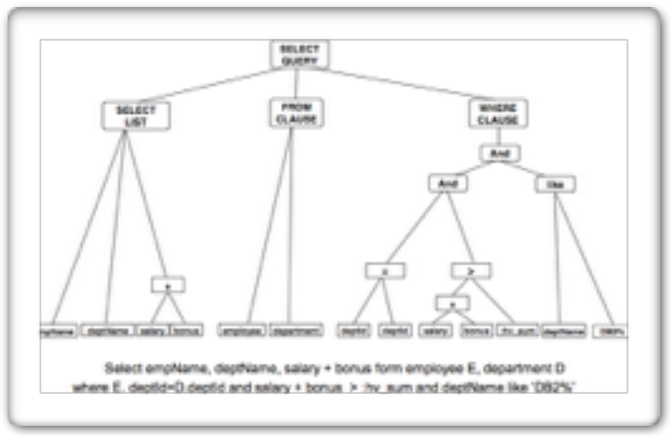
Translating SQL to RA

Database Systems: The Complete Book
Ch 16,16.1

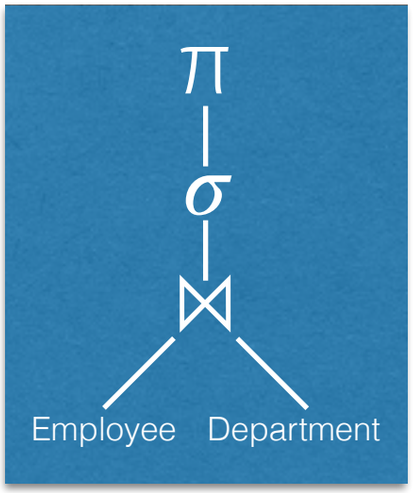
The Evaluation Pipeline


.sql

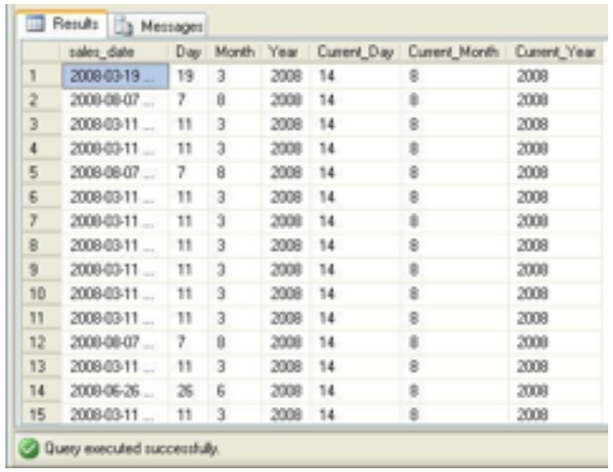
How does this work? (now)



Parsed Query




Data



Results

What does this look like? (last class)

How does this work? (later today?)

A Basic SQL Query

(optional) keyword indicating that the answer should **not** contain duplicates

```
SELECT  [DISTINCT] target-list
```

A list of attributes of relations in relation-list

```
FROM    relation-list
```

A list of relation names
(possibly with a range-variable after each name)

```
WHERE   condition
```

Comparisons (‘=’, ‘<>’, ‘<‘, ‘>’, ‘<=’, ‘>=’) and other boolean predicates,
combined using AND, OR, and NOT
(a boolean formula)

SQL

- SQL is a language for querying relations
  - **`SELECT`** to access (query) data
    - Different features for different access patterns.
  - **`INSERT INTO`**, **`DELETE FROM`** to modify data
  - **`CREATE TABLE`**, **`DROP TABLE`**, **`ALTER TABLE`** to modify relations

Relational Algebra Trees

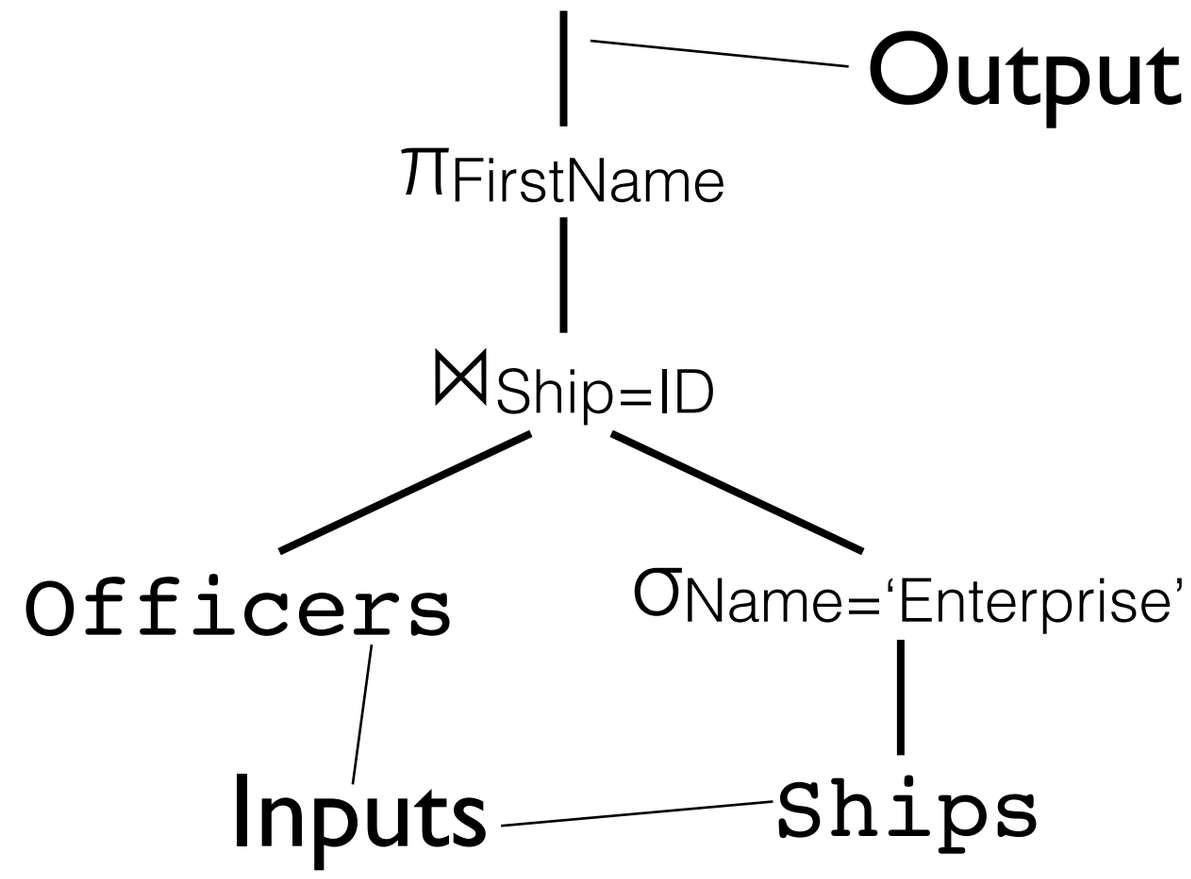
```
SELECT O.FirstName
FROM Officers O, Ships S
WHERE O.Ship = S.ID
      AND S.Name = 'Enterprise'
```

$\pi_{\text{FirstName}}(\texttt{Officers} \bowtie_{\text{Ship=ID}} (\sigma_{\text{Name='Enterprise'}} \texttt{Ships}))$

Relational Algebra Trees

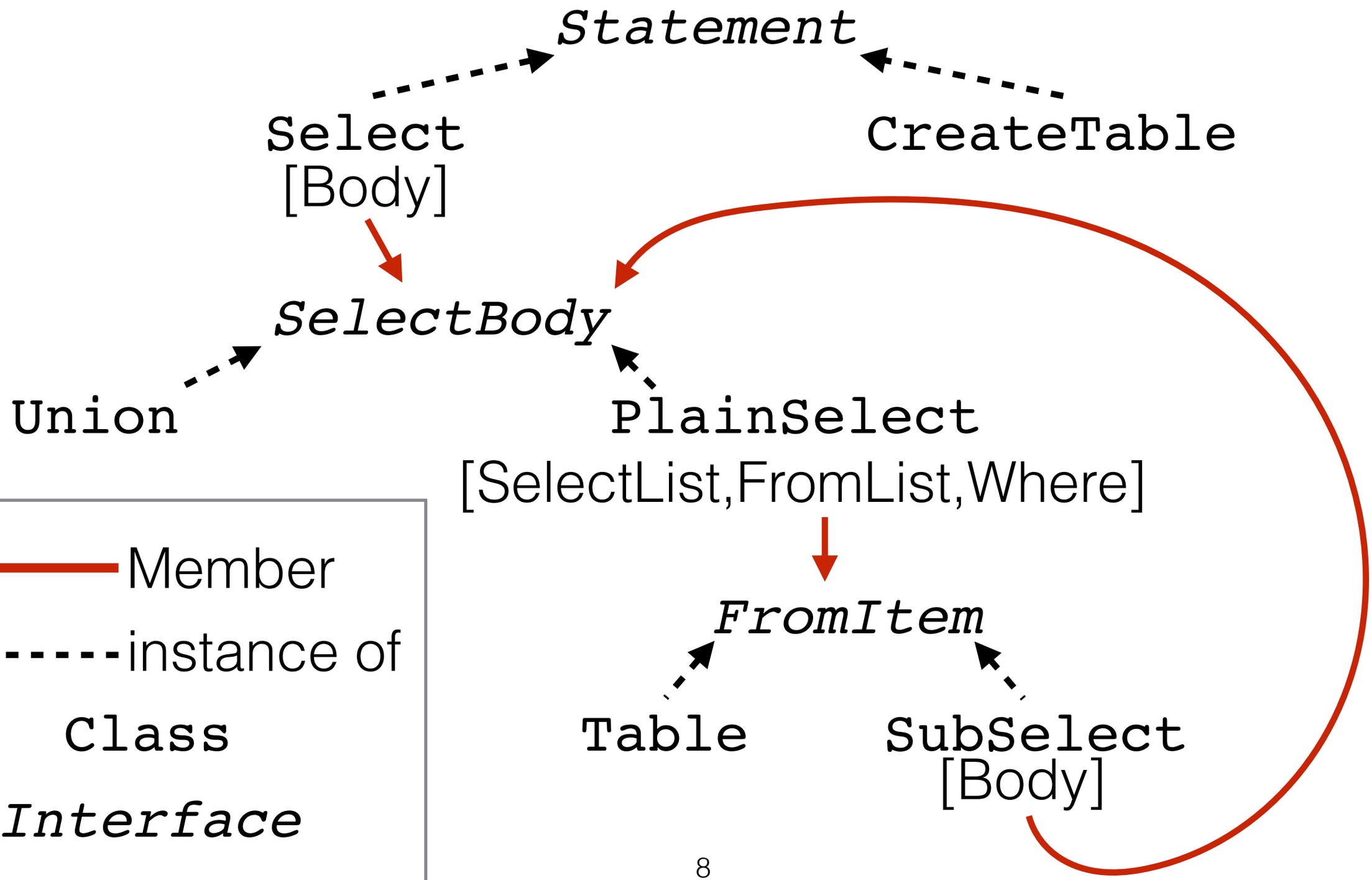
$\pi_{\text{FirstName}}(\mathbf{Officers} \bowtie_{\text{Ship=ID}}(\sigma_{\text{Name='Enterprise'}}\mathbf{Ships}))$

Relational Algebra Trees



$\pi_{\text{FirstName}}(\texttt{Officers} \bowtie_{\text{Ship=ID}} (\sigma_{\text{Name='Enterprise'}} \texttt{Ships}))$

Syntax Trees in Java



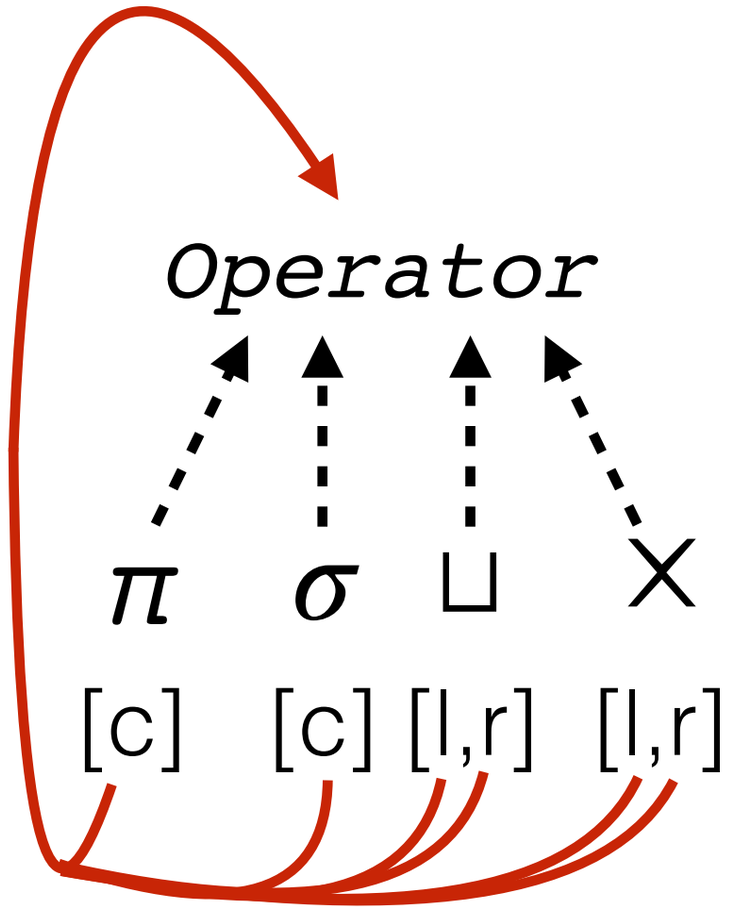
InstanceOf

```
Statement statement = parser.Statement();

if(statement instanceof Select) {
  Algebra raTree = parseTree((Select)statement);
  evaluate(raTree);

} else if(statement instanceof CreateTable) {
  loadTableSchema((CreateTable)statement);

}
```

Syntax Trees in Java

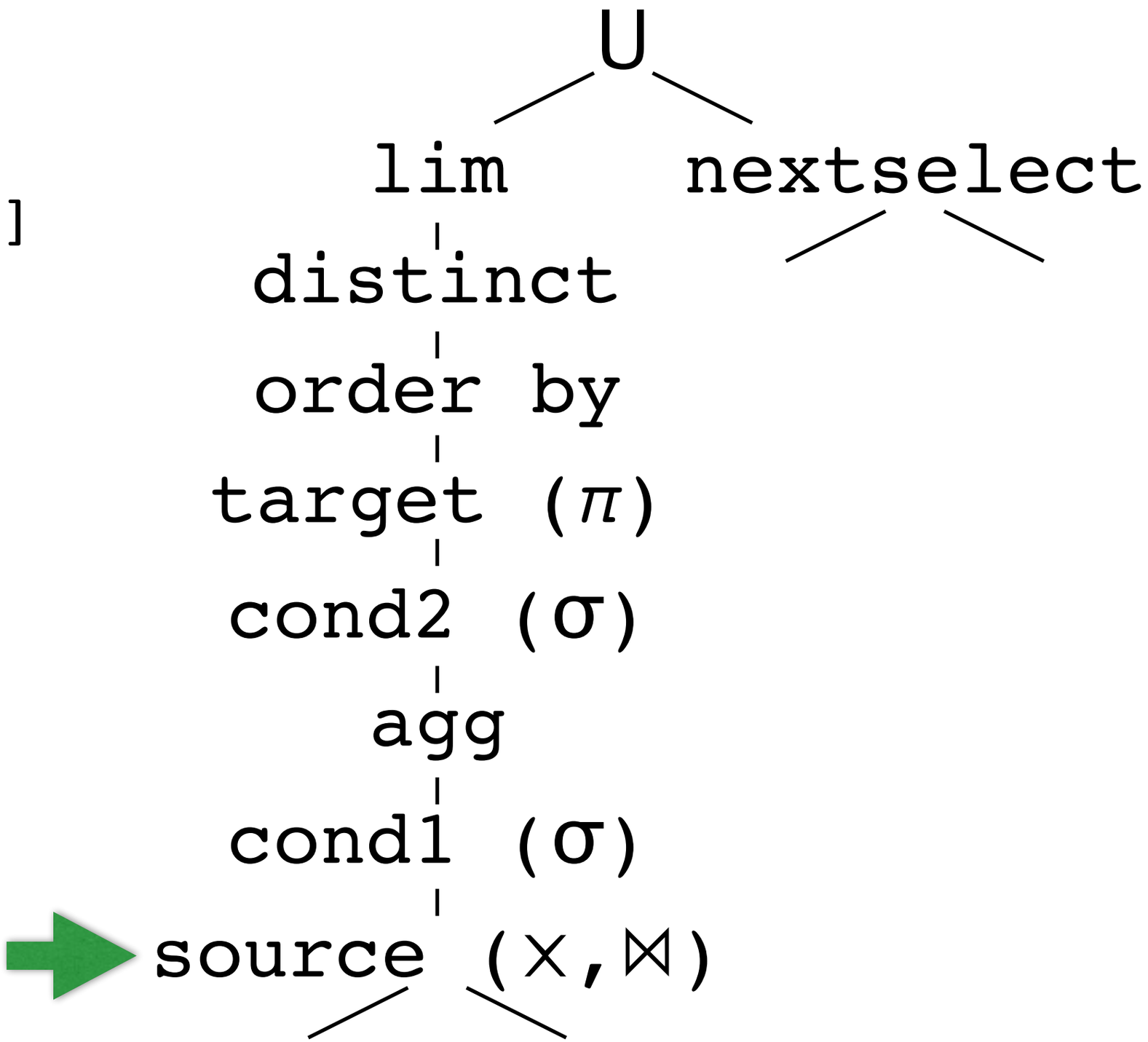
What would a class hierarchy look like for Relational Algebra?

Syntax Trees in Java



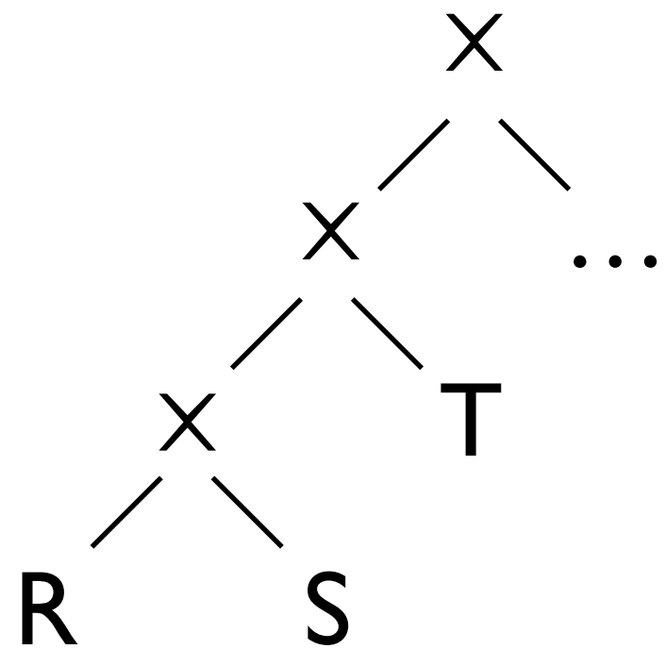
SQL to RA

```
SELECT [DISTINCT]
       target
FROM source
WHERE cond1
GROUP BY …
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect
```



FROM Clause

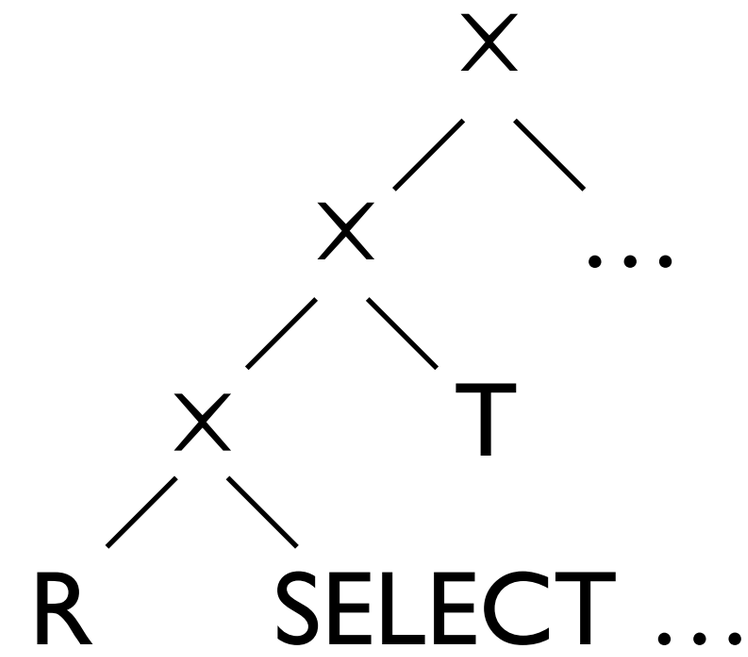
```
FROM R, S, T, …
```



What happens if I have a FROM-nested query?

FROM Clause

```
FROM R, (SELECT …) S, T, …
```



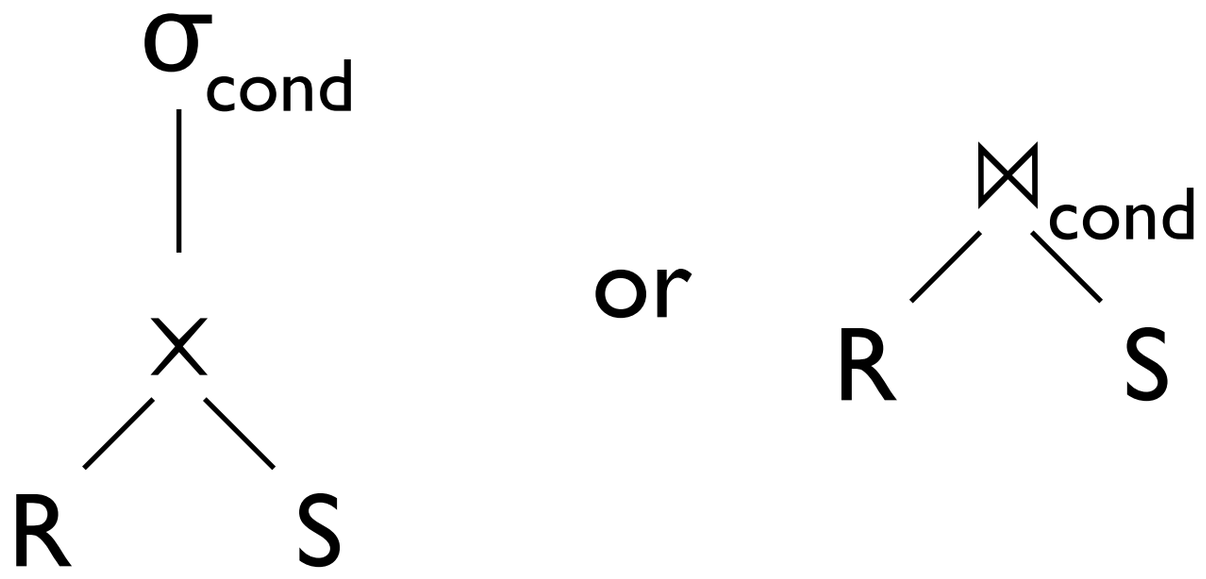
Selects are just relations!

FROM Clause

```
FROM R JOIN S ON cond
```

FROM Clause

```
FROM R JOIN S ON cond
```

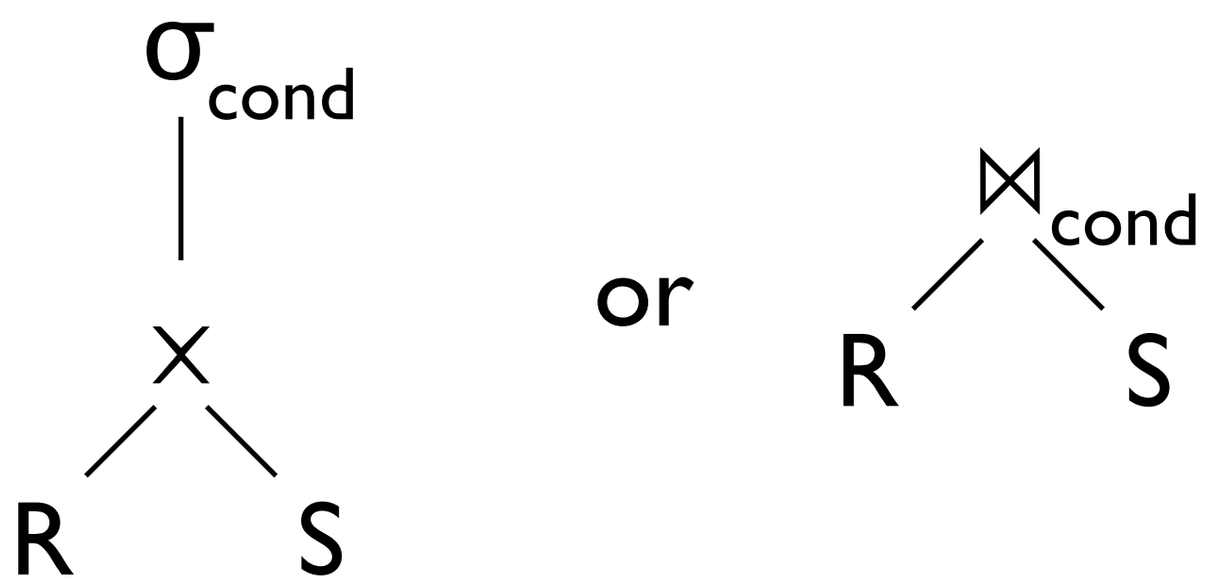


FROM Clause

```
FROM R NATURAL JOIN S
```

FROM Clause

```
FROM R NATURAL JOIN S
```

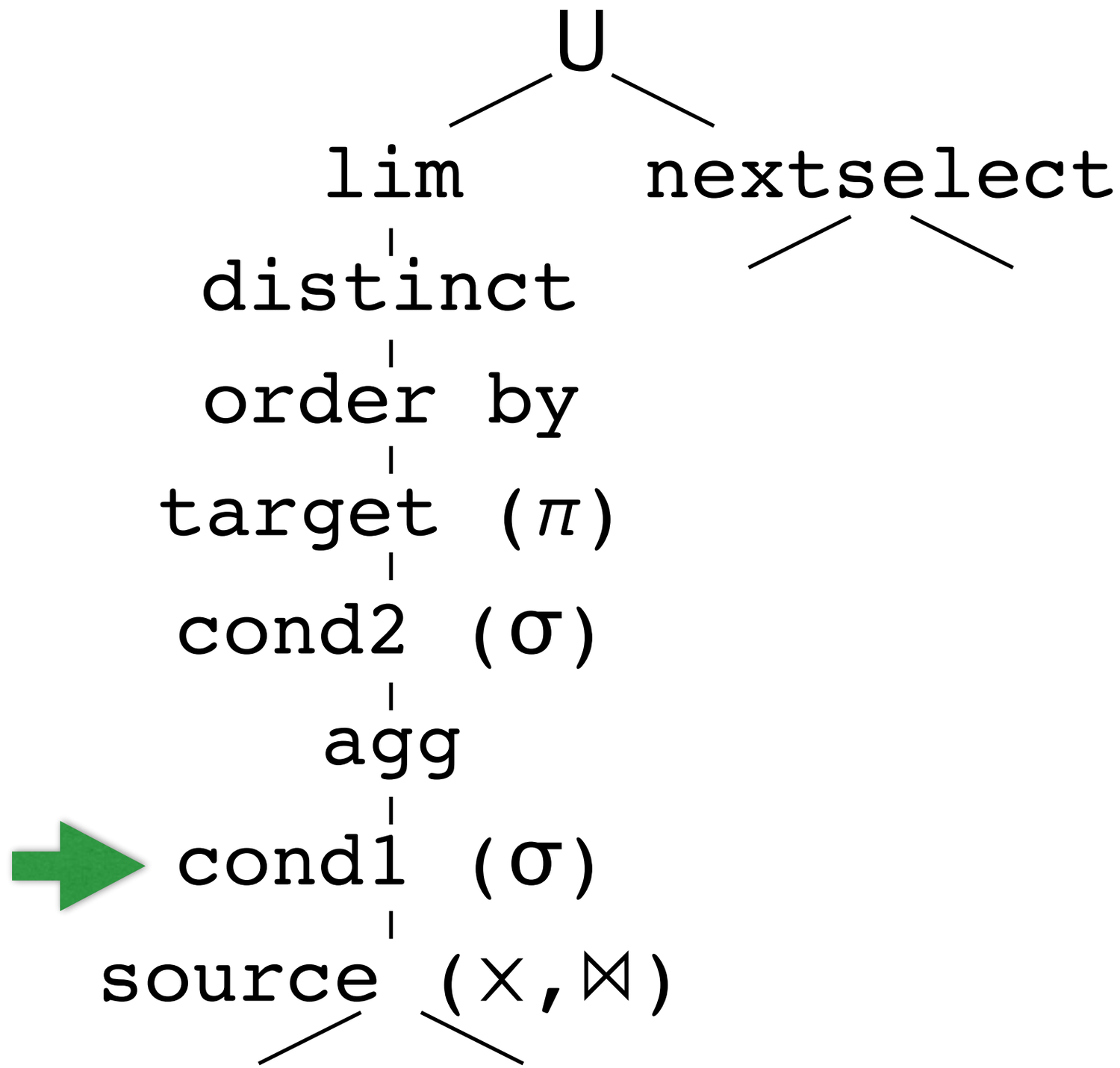



```
cond = schema(R) ∩ schema(S)
```

You need to be able to compute the schema of a RA operator

SQL to RA

```
SELECT [DISTINCT]
       target
FROM source
WHERE cond1
GROUP BY …
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect
```



SELECT (target) Clause

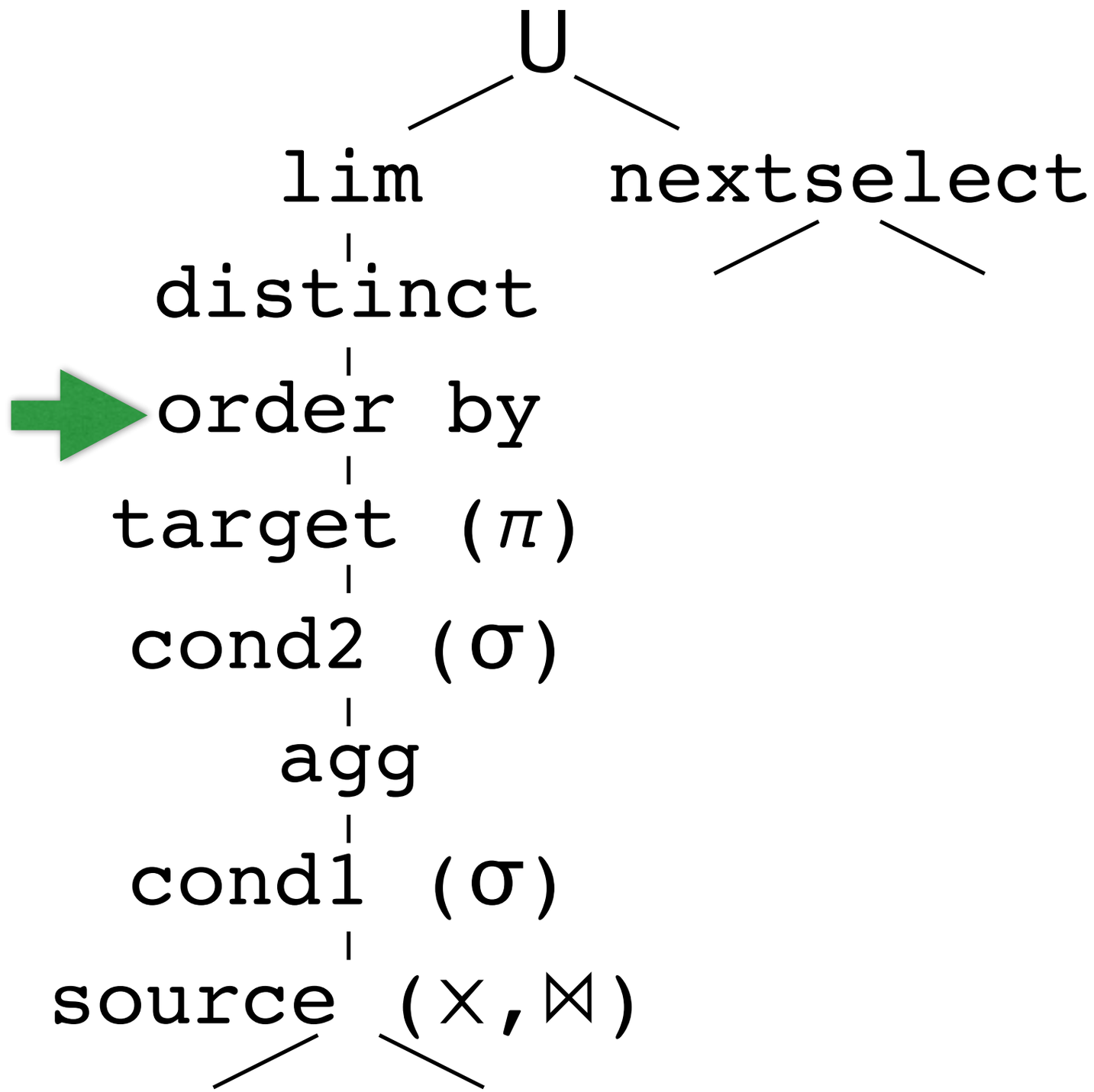

```
SELECT *
```
no π (or target = `schema(input)`)

```
SELECT A, B, …
  targets = A, B, …

SELECT R.*, S.*
  targets =
    schema(input) from R, S
```

Schemas need both Table Alias & Attribute Name (see Column class)

SQL to RA

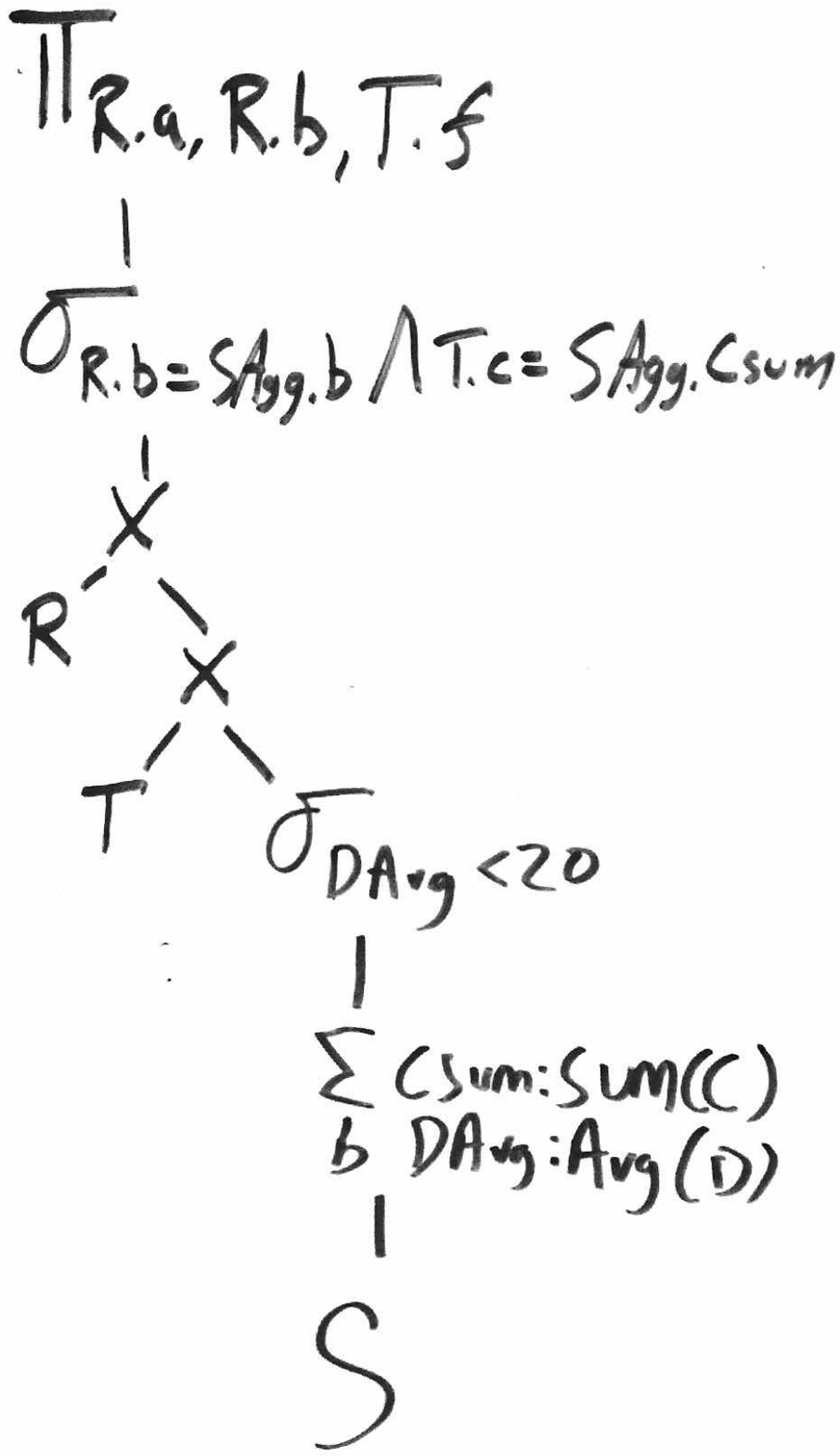
```
SELECT [DISTINCT]
       target
FROM source
WHERE cond1
GROUP BY …
HAVING cond2
ORDER BY order
LIMIT lim
UNION nextselect
```



Let's Try It...

```
CREATE TABLE R(a int, b int)
CREATE TABLE S(b int, c int, d int)
CREATE TABLE T(c int, e int, f string)
```

```
SELECT R.*, T.f
FROM R, T, (
    SELECT b, AVG(d) AS DAvg,
              SUM(c) AS CSum
    FROM S WHERE d > 10
    GROUP BY b HAVING DAvg < 20
  ) SAgg
WHERE R.b = SAgg.b AND T.c = SAgg.CSum
```


$\Pi_{R.a, R.b, T.f}$
$\sigma_{R.b = SAgg.b \wedge T.c = SAgg.Csum}$
X
R
X
T
$\sigma_{DAvg < 20}$
Σ Csum: Sum(C)
b DAvg: Avg(D)
S

… but that's stupid!

That query will be slooooooooooow.

Translation is hard.

Don’t make your life harder.

Translate Dumb, Fix it in the Optimizer

Group Work

Write pseudocode translating
from a non-aggregate
SELECT … FROM … WHERE …
to a relational algebra expression

Evaluating RA

Database Systems: The Complete Book
Ch 15,15.1-15.3

Project Outline

SQL Query → Parser & Translator → Relational Algebra

.sql

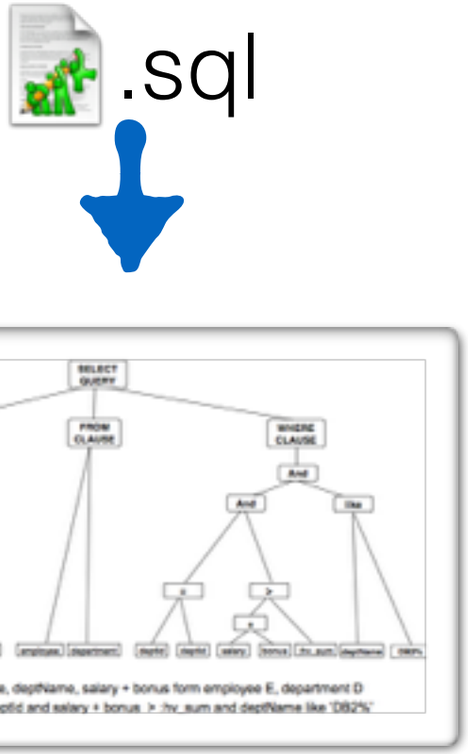
JSqlParser

???

Query Result

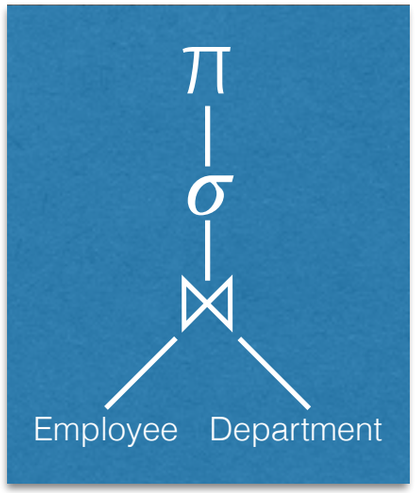
Here Lie Dragons?

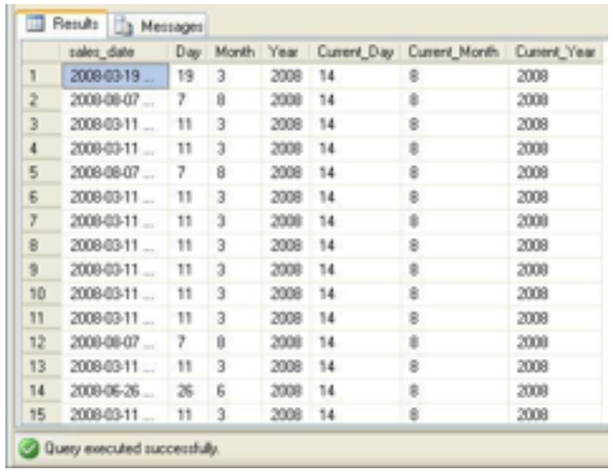
The Evaluation Pipeline



.sql

Parsed Query





Results

Data

How does this work? (now)

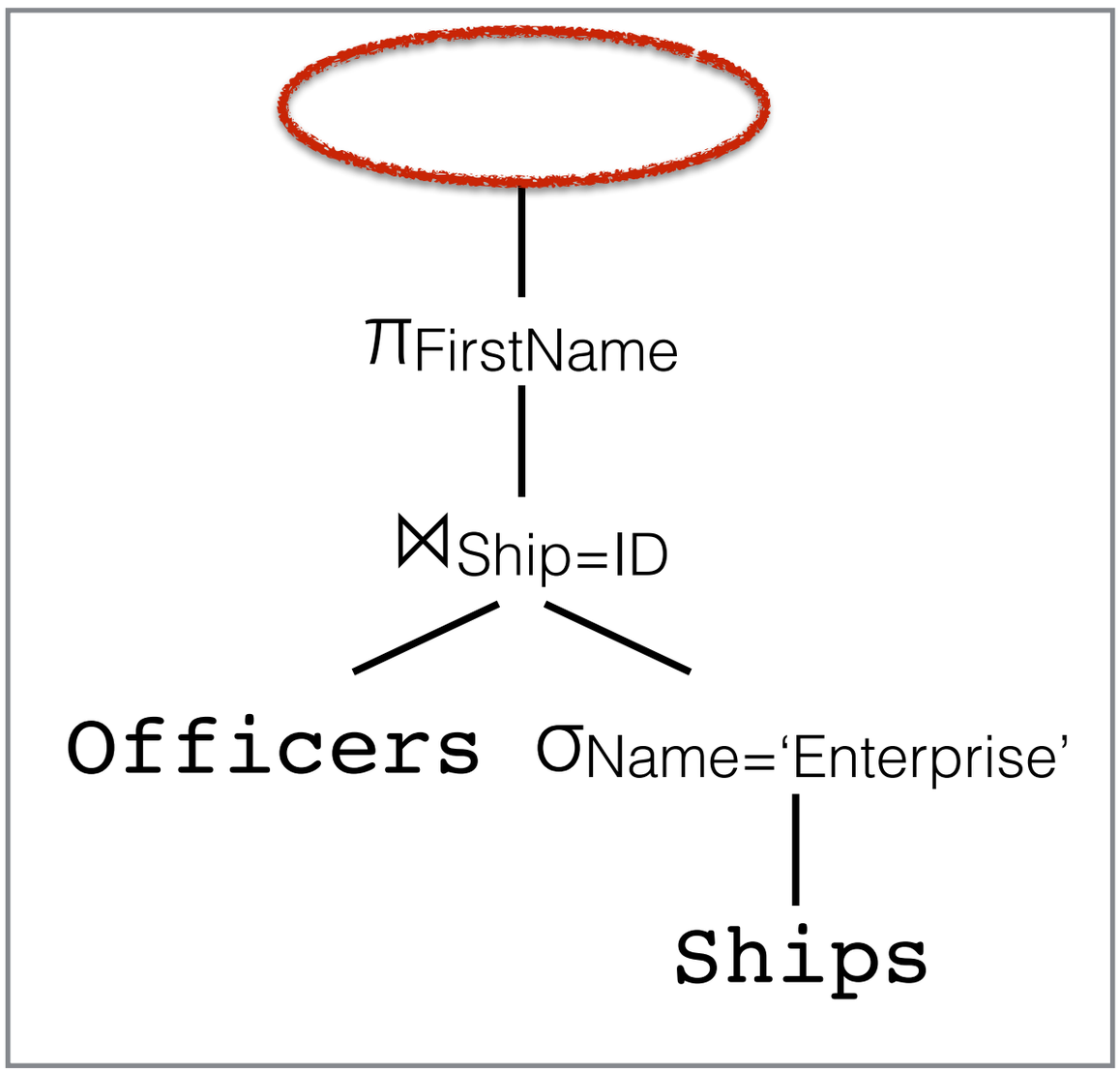
Evaluation Strategies

- **Staged Evaluation**: Start at leaves, Evaluate each operator as one step.

- **Pull Model**: Tuple-at-a-time Iterator for each operator (also called *Volcano Operators*) reads from source iterator(s).

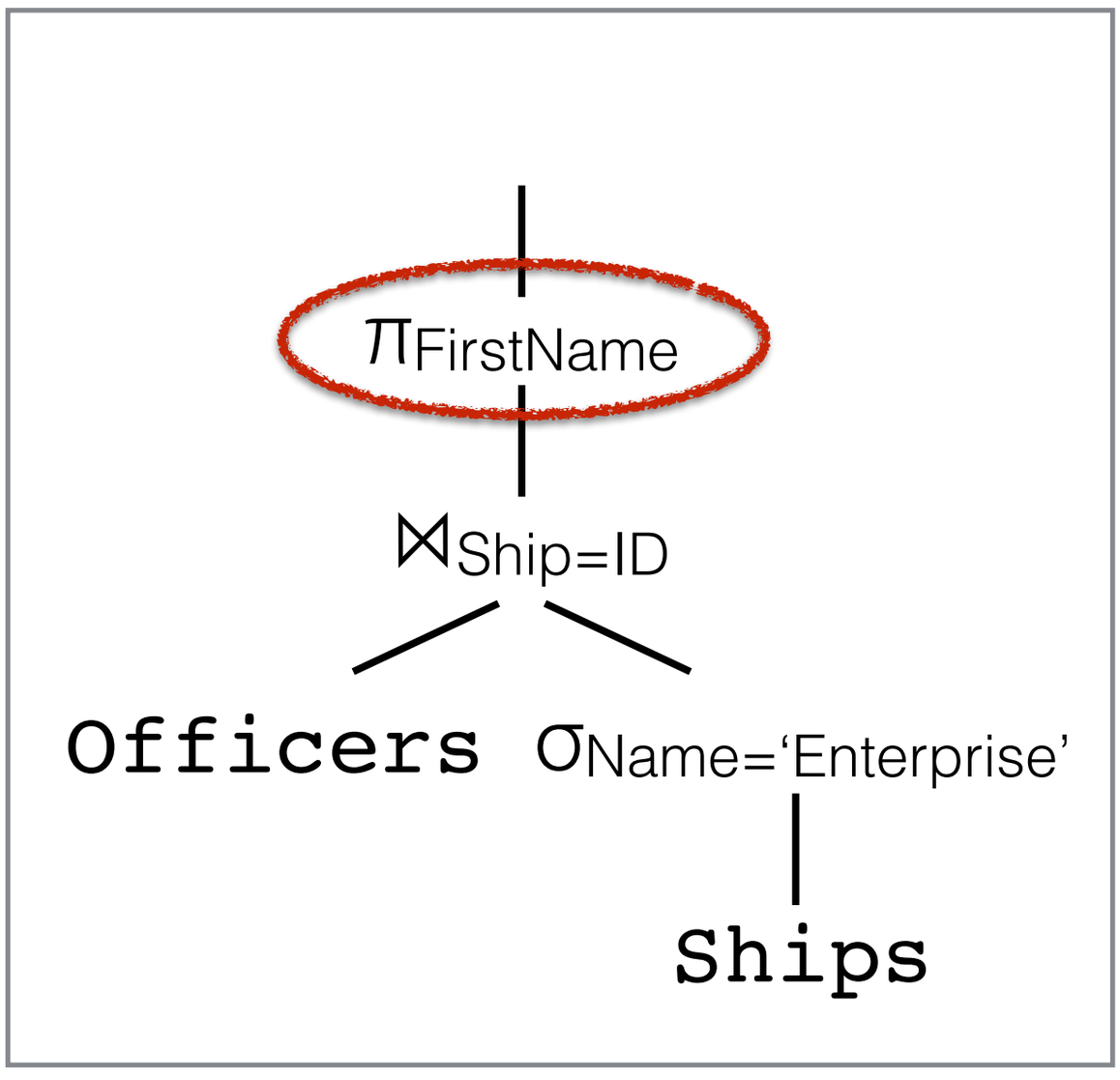
- **Push Model**: Thread-per operator reads from input buffer(s) and writes to output buffer.

Staged Evaluation



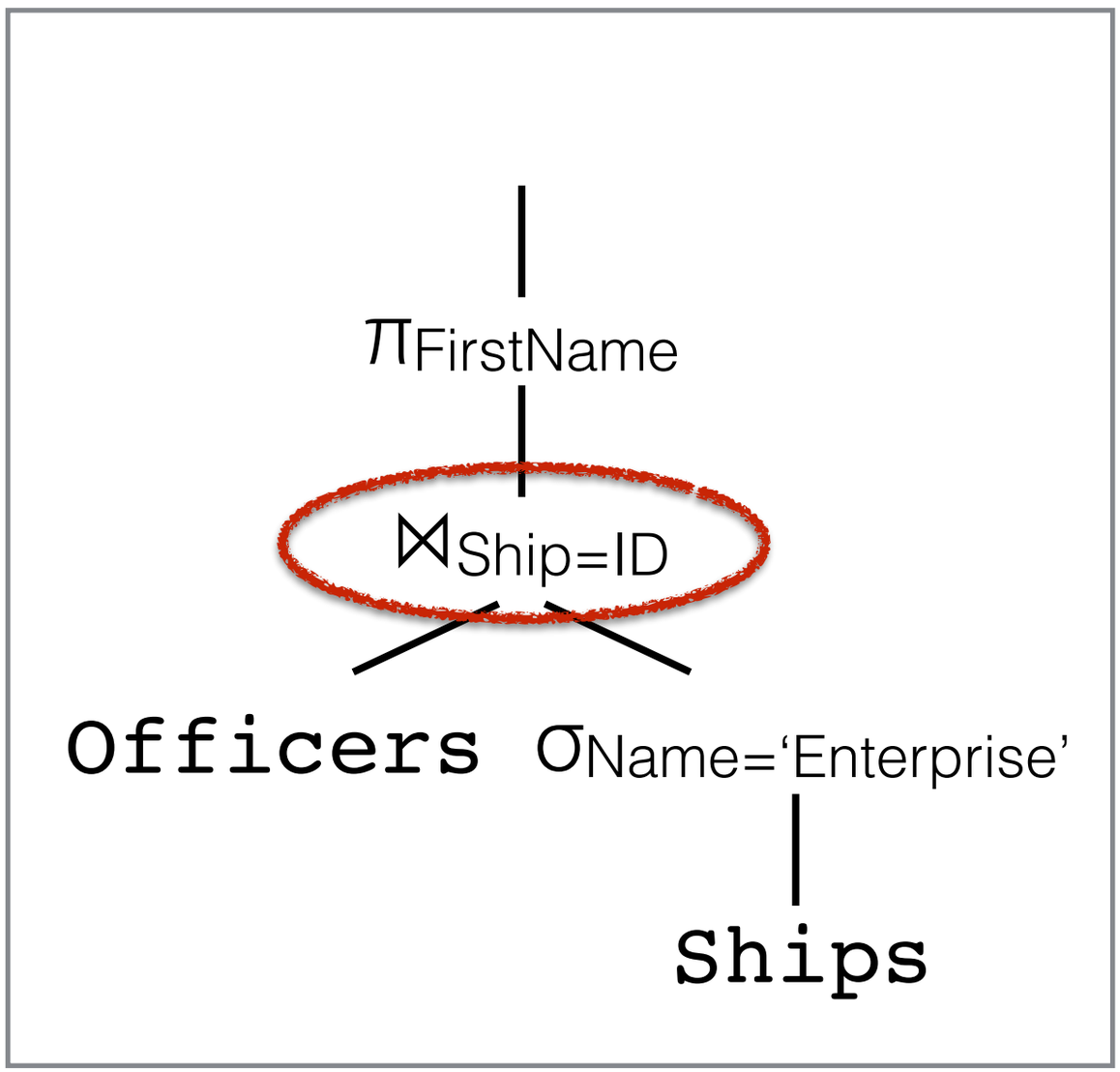
Compute the Output

Staged Evaluation



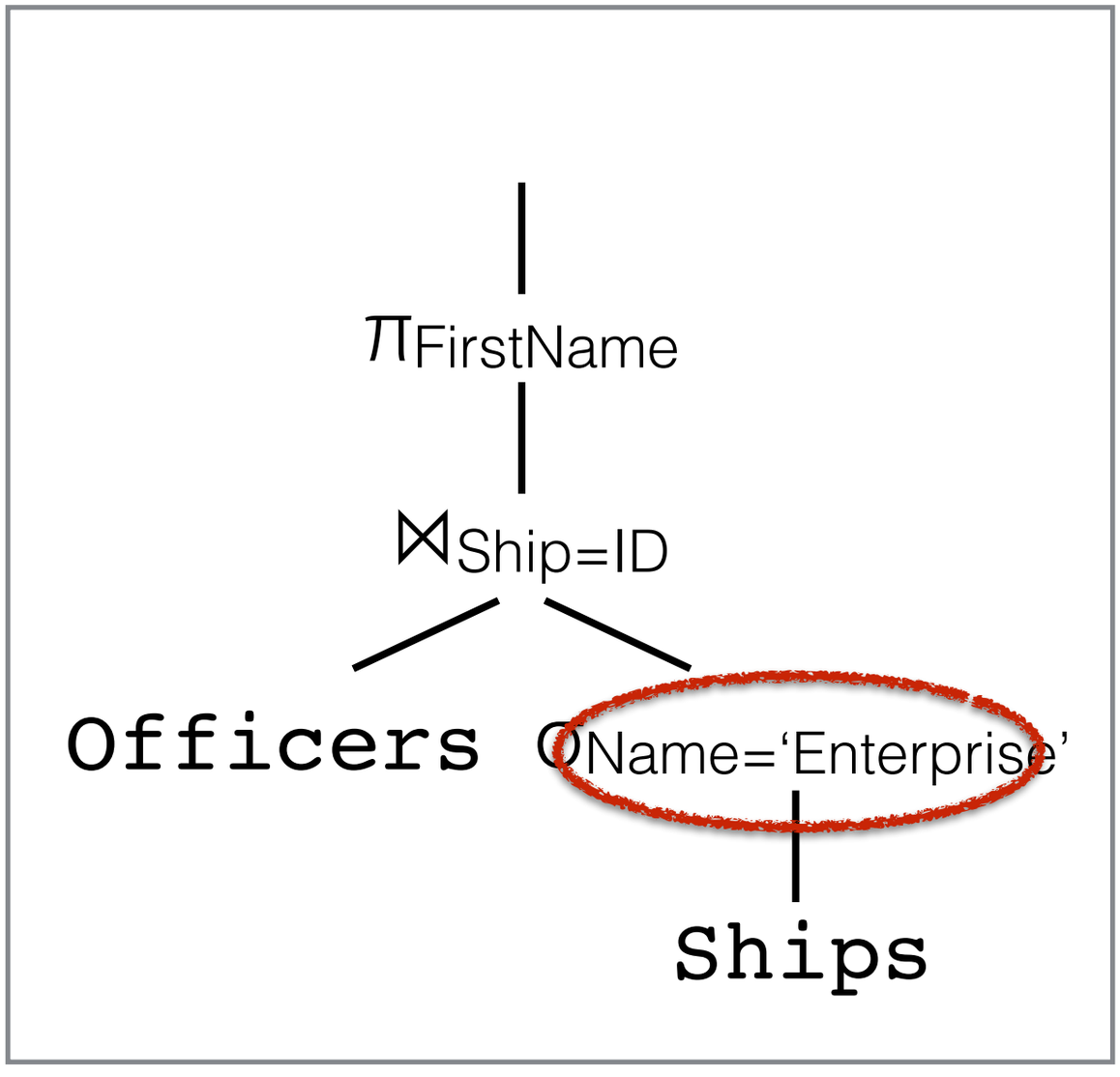
Compute the Output
Dependency: Compute π

Staged Evaluation



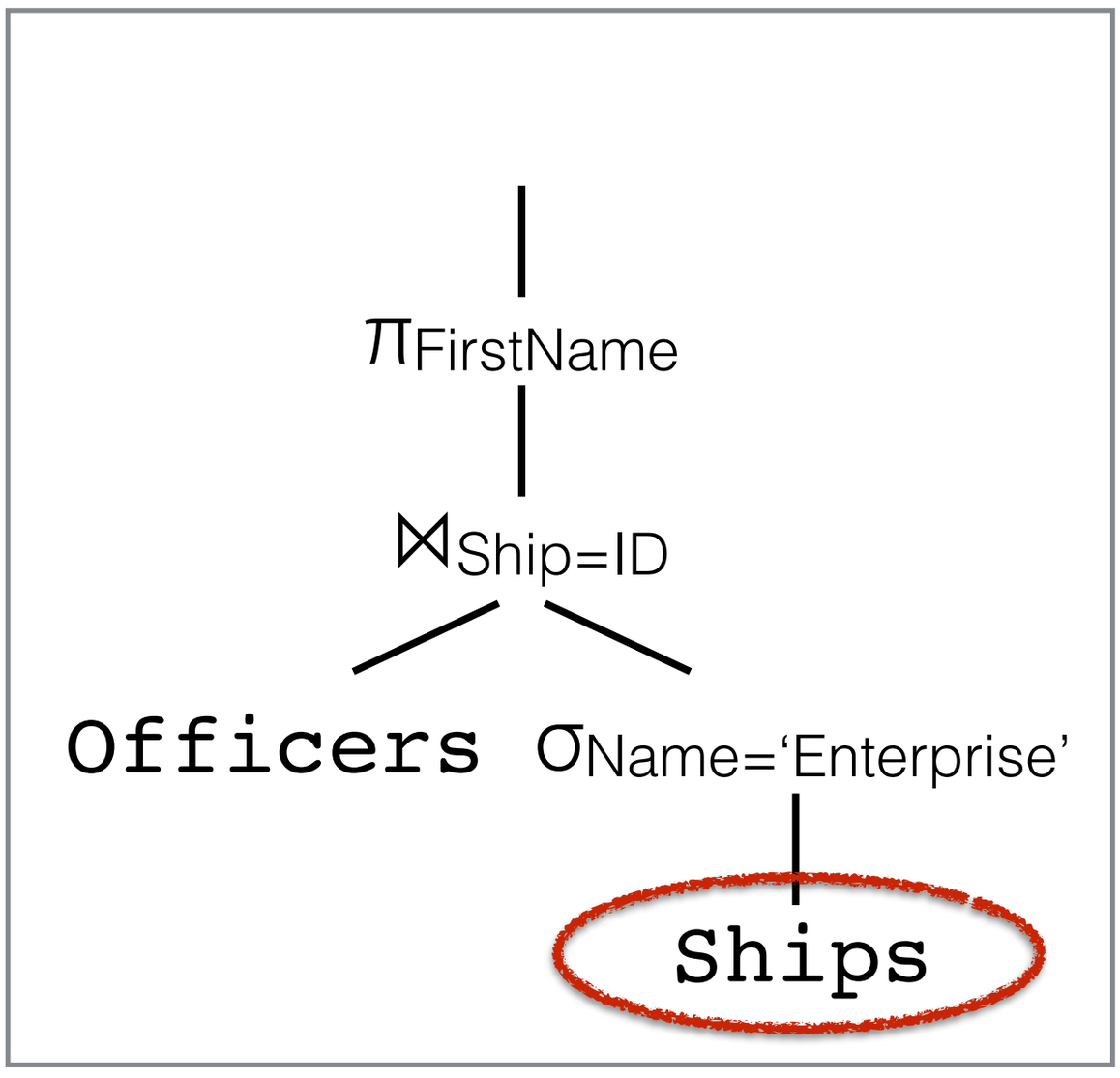
Compute the Output
Dependency: Compute π
Dependency: Compute ⋈

Staged Evaluation



Compute the Output
Dependency: Compute π
Dependency: Compute ⋈
Dependency: Compute σ

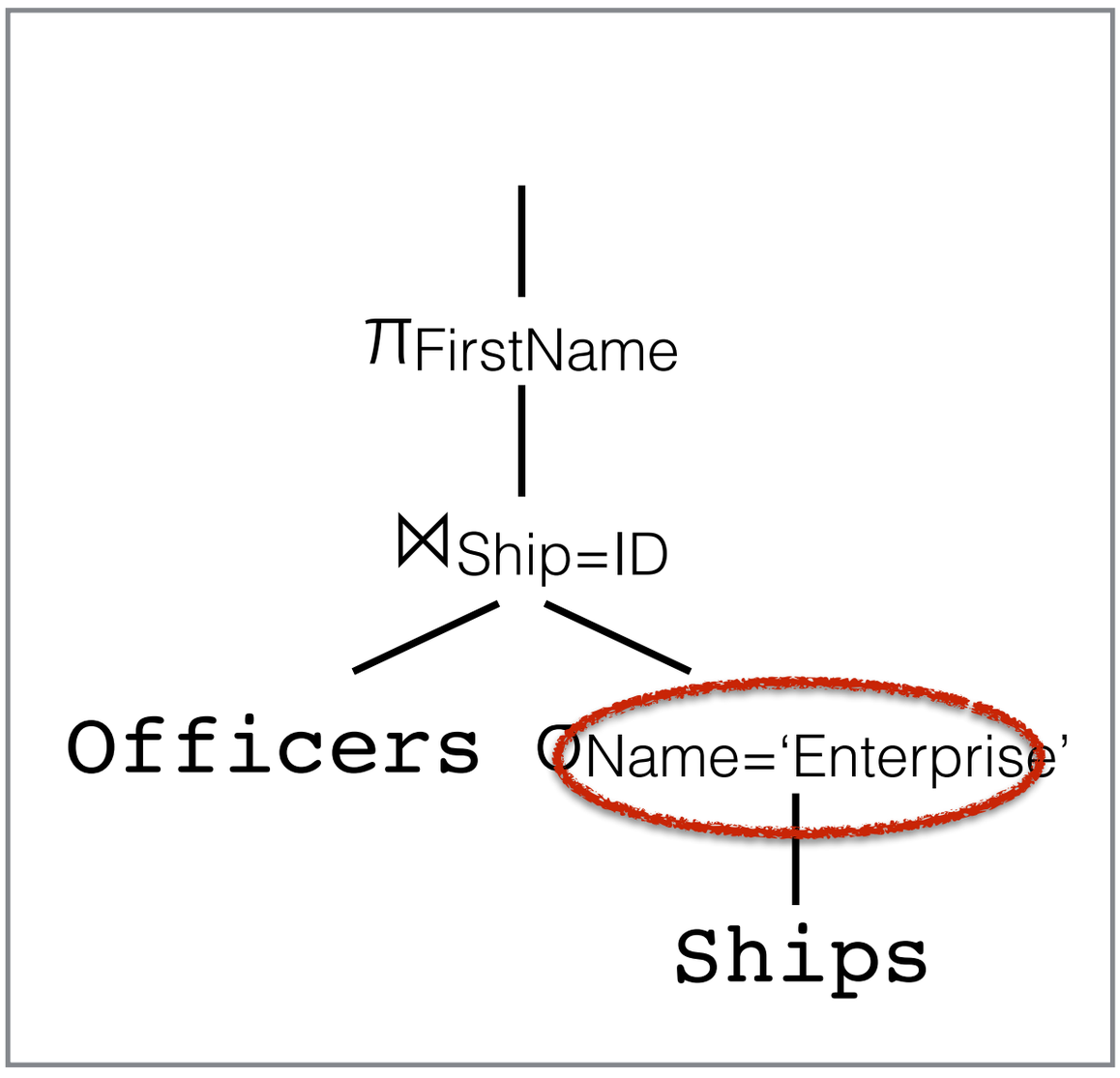
Staged Evaluation



Compute the Output
Dependency: Compute π
Dependency: Compute ⋈
Dependency: Compute σ
Dependency: Load `Ships`

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

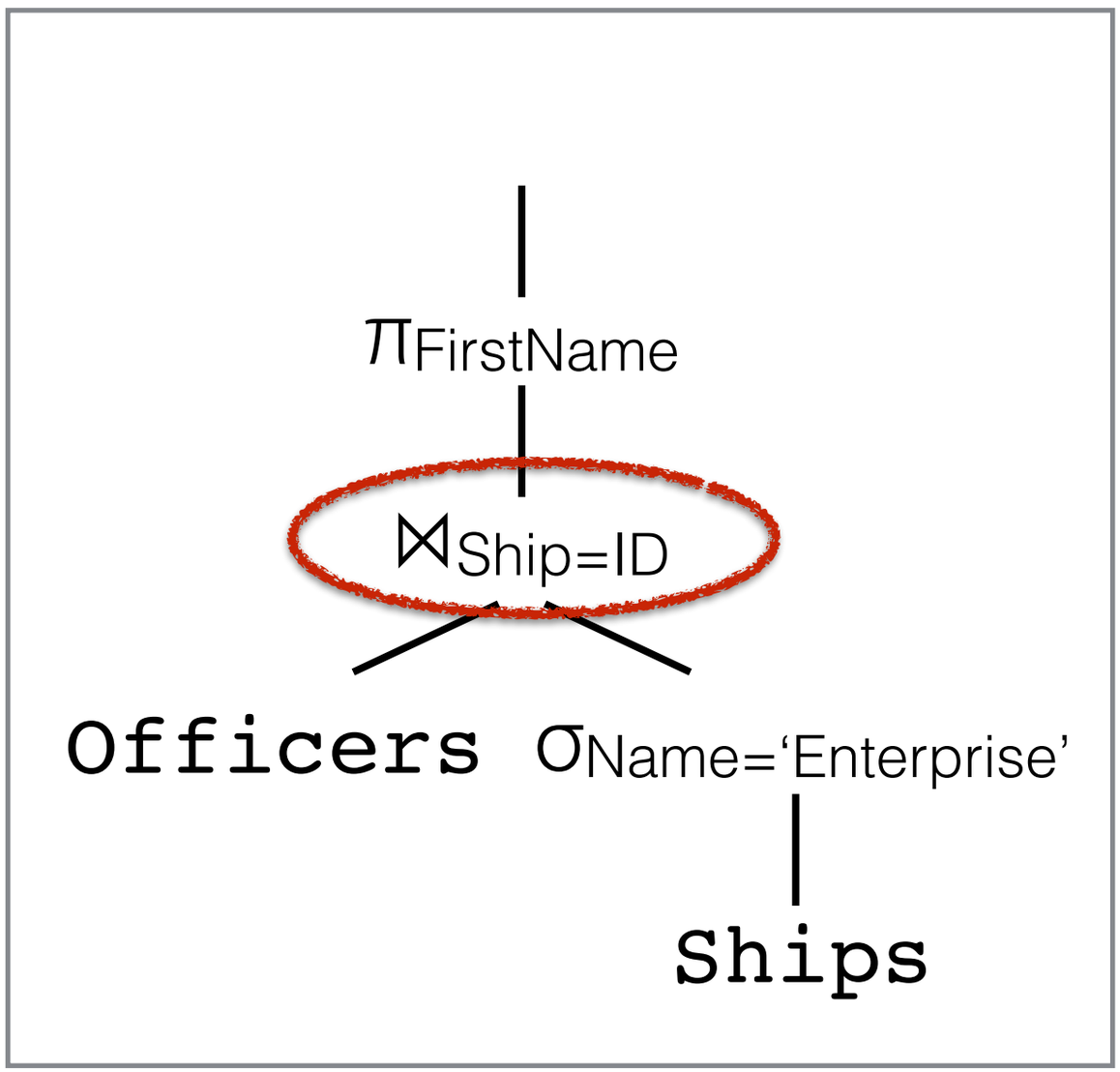
Staged Evaluation



Compute the Output
Dependency: Compute π
Dependency: Compute ⋈
Dependency: Compute σ

ID,	Name
[1701,	Enterprise]
[DS9,	Deep Space 9]
[74656,	Voyager]
[75633,	Defiant]

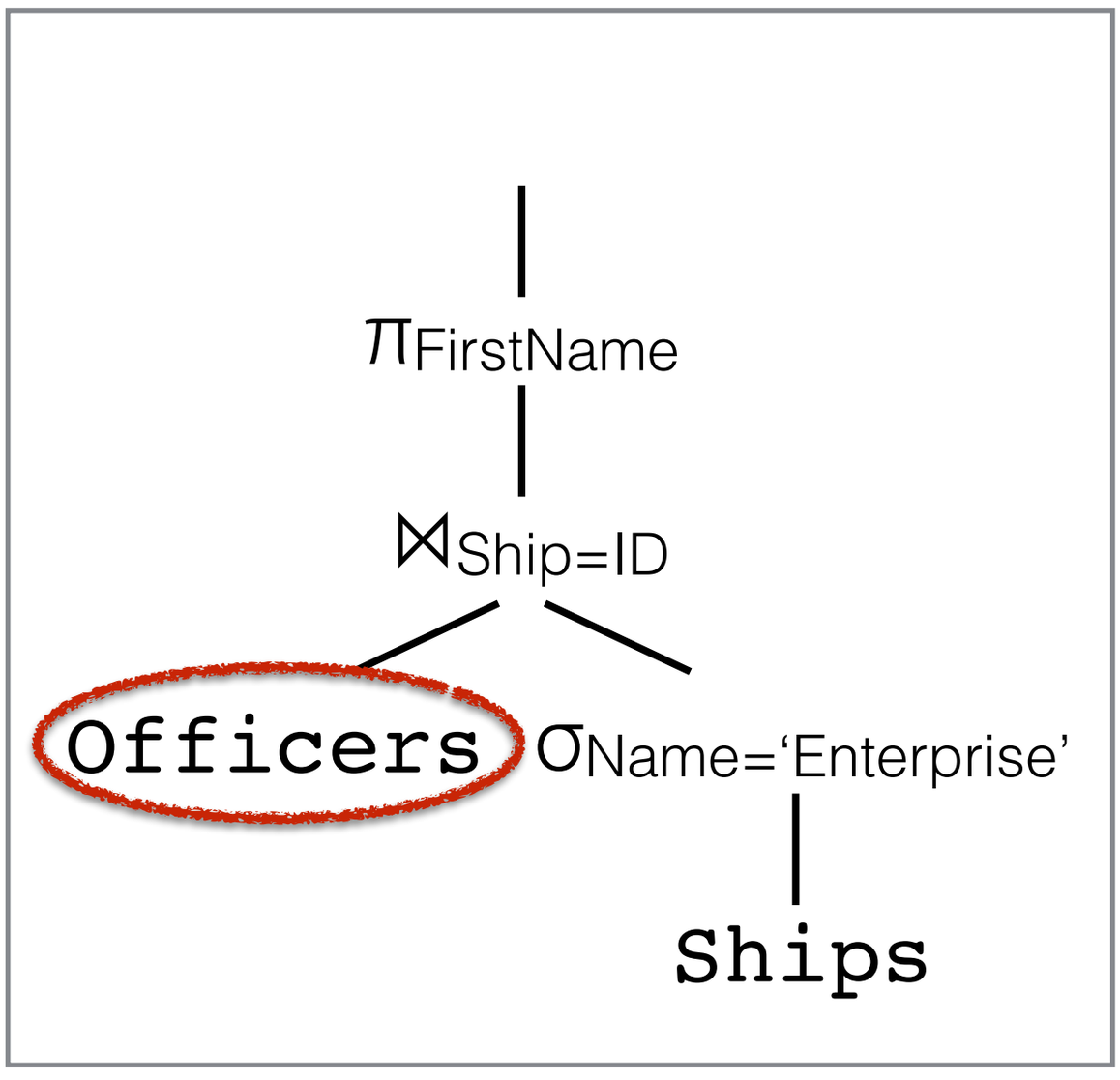
Staged Evaluation



Compute the Output
Dependency: Compute π
Dependency: Compute ⋈

ID,	Name
[1701,	Enterprise]

Staged Evaluation

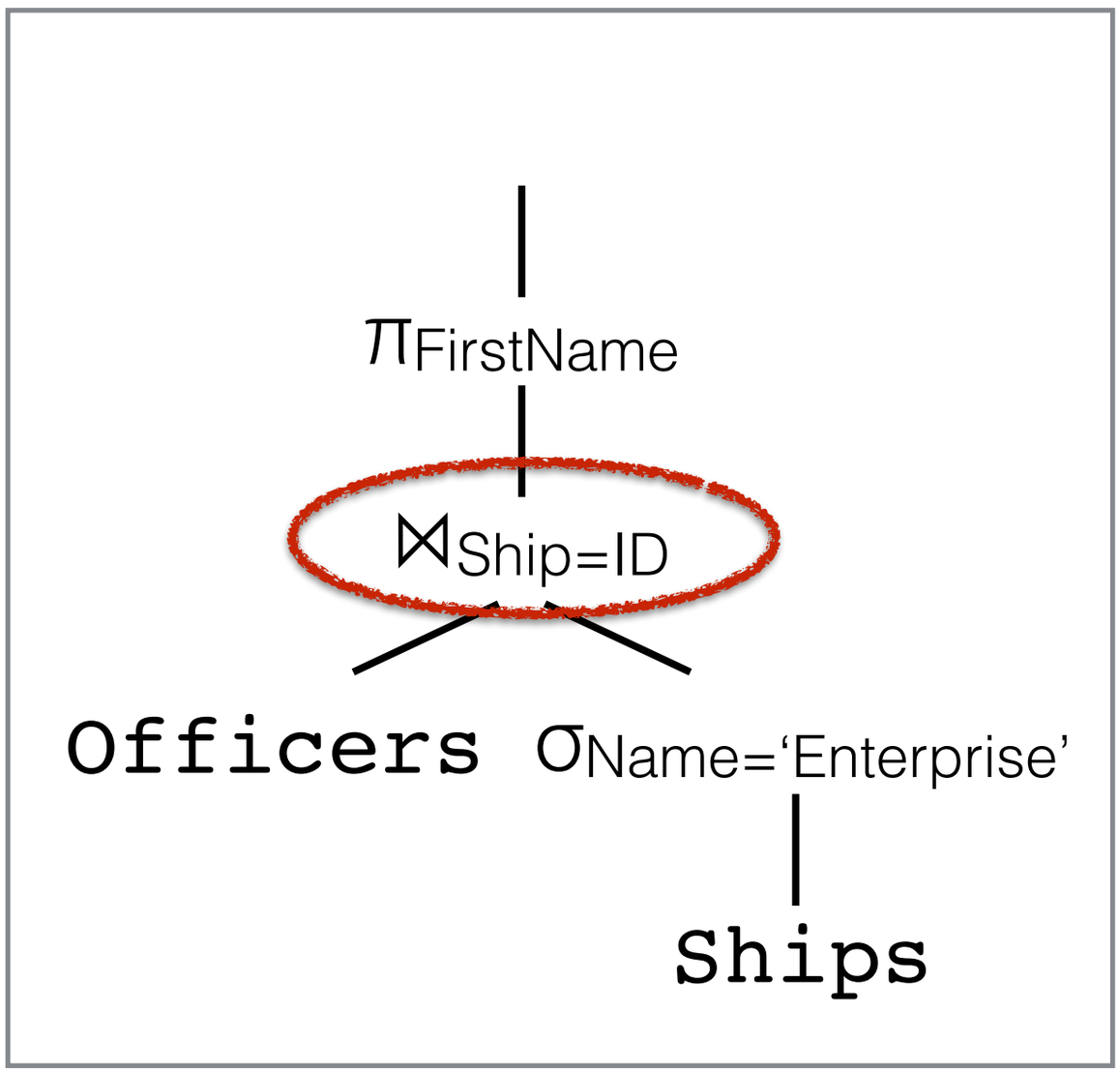


Compute the Output
Dependency: Compute π
Dependency: Compute ⋈
Dependency: Load `Officers`

ID,	Name
[1701,	Enterprise]

FirstName,	LastName,	Rank,	Ship
[James,	Kirk,	4.0,	1701]
[Jean Luc,	Picard,	4.0,	1701D]
[Benjamin,	Sisko,	3.0,	DS9]
[Kathryn,	Janeway,	4.0,	74656]
[Nerys,	Kira,	2.5,	75633]
[Spock,	NULL,	2.5,	1701]
[William,	Riker,	2.5,	1701D]
[Nerys,	Kira,	2.5,	DS9]
[Chakotay,	NULL,	3.0,	74656]

Staged Evaluation

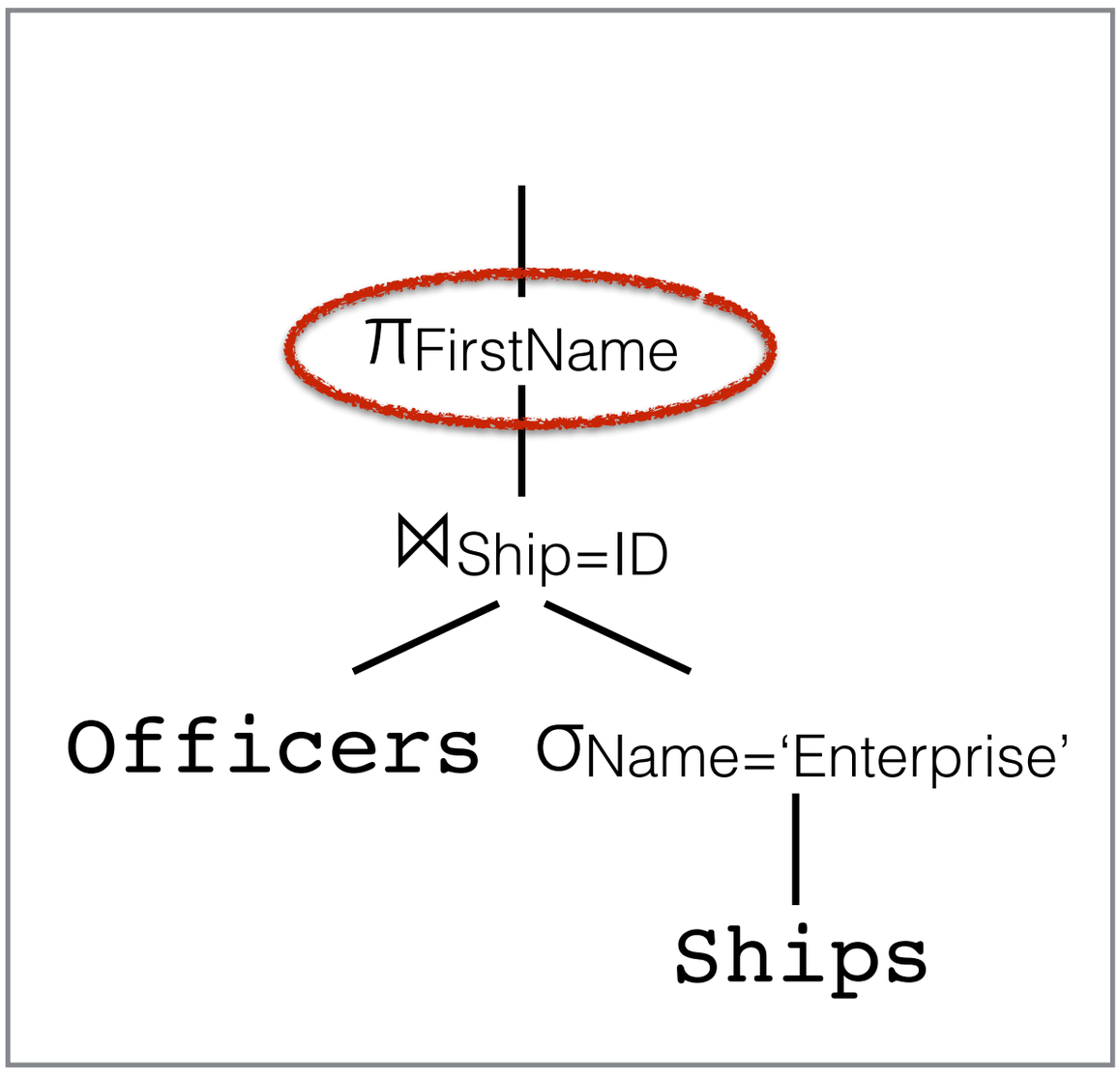


Compute the Output
Dependency: Compute π
Dependency: Compute ⋈

ID,	Name
[1701	Enterprise]

FirstName,	LastName,	Rank,	Ship,	Name
[James,	Kirk,	1701,	0,	Enterprise]
[Spock,	NULL,	1701,	0,	Enterprise]
[James,	NULL,	Kirk,	1701,	Enterprise]
[Jean Luc,		Picard,	4.0,	1701D]
[Benjamin,		Sisko,	3.0,	DS9]
[Kathryn,		Janeway,	4.0,	74656]
[Nerys,		Kira,	2.5,	75633]
[Spock,		NULL,	2.5,	1701]
[William,		Riker,	2.5,	1701D]
[Nerys,		Kira,	2.5,	DS9]
[Chakotay,		NULL,	3.0,	74656]

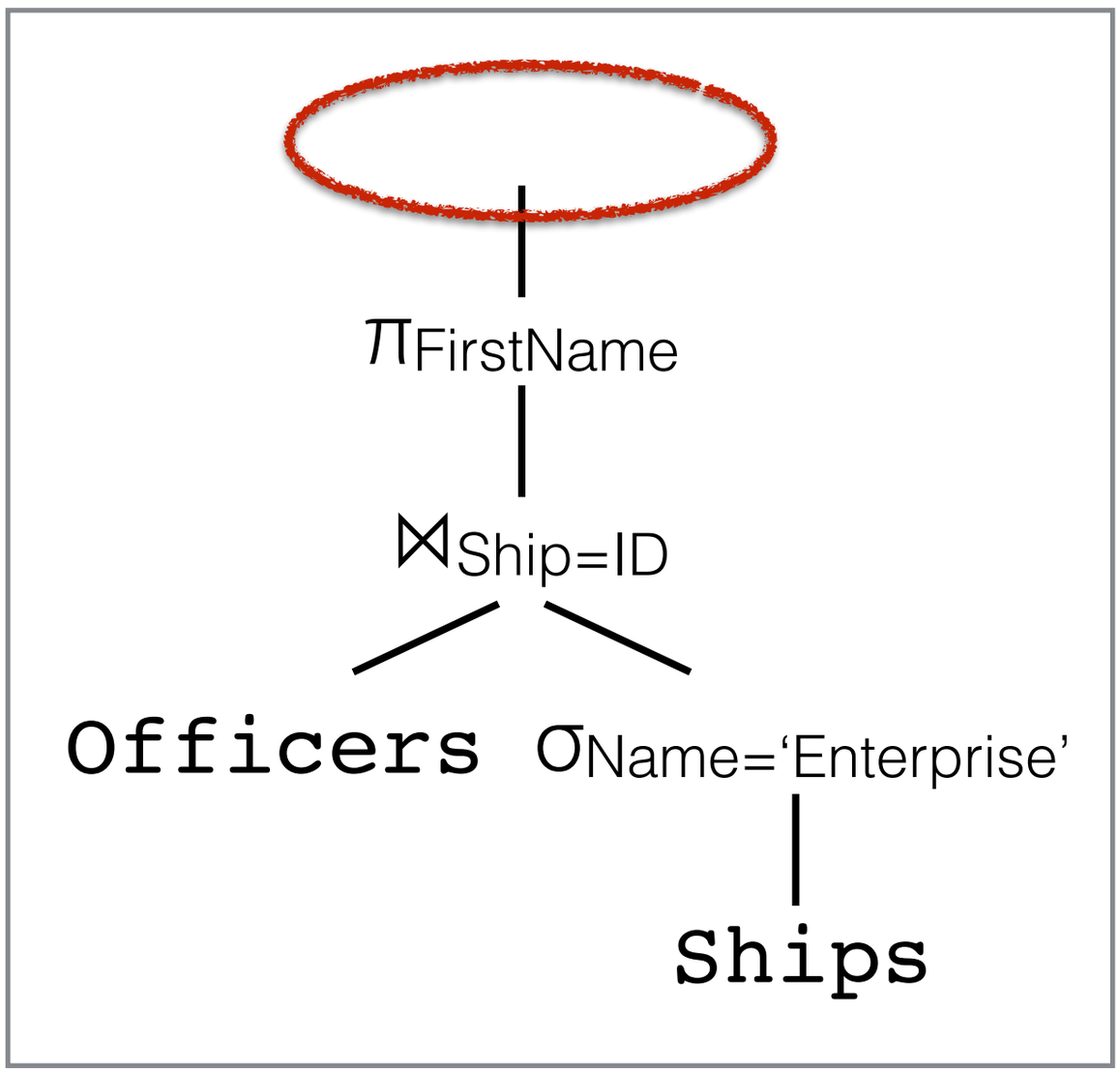
Staged Evaluation



Compute the Output
Dependency: Compute π

FirstName,	LastName,	Rank,	Ship,	ID,	Name
[James,]	Kirk,	4.0,	1701,	1701,	Enterprise]
[Spock,]	NULL,	2.5,	1701,	1701,	Enterprise]

Staged Evaluation



Compute the Output

FirstName
James
Spock

Staged Evaluation

Can we do better?

Staged Evaluation

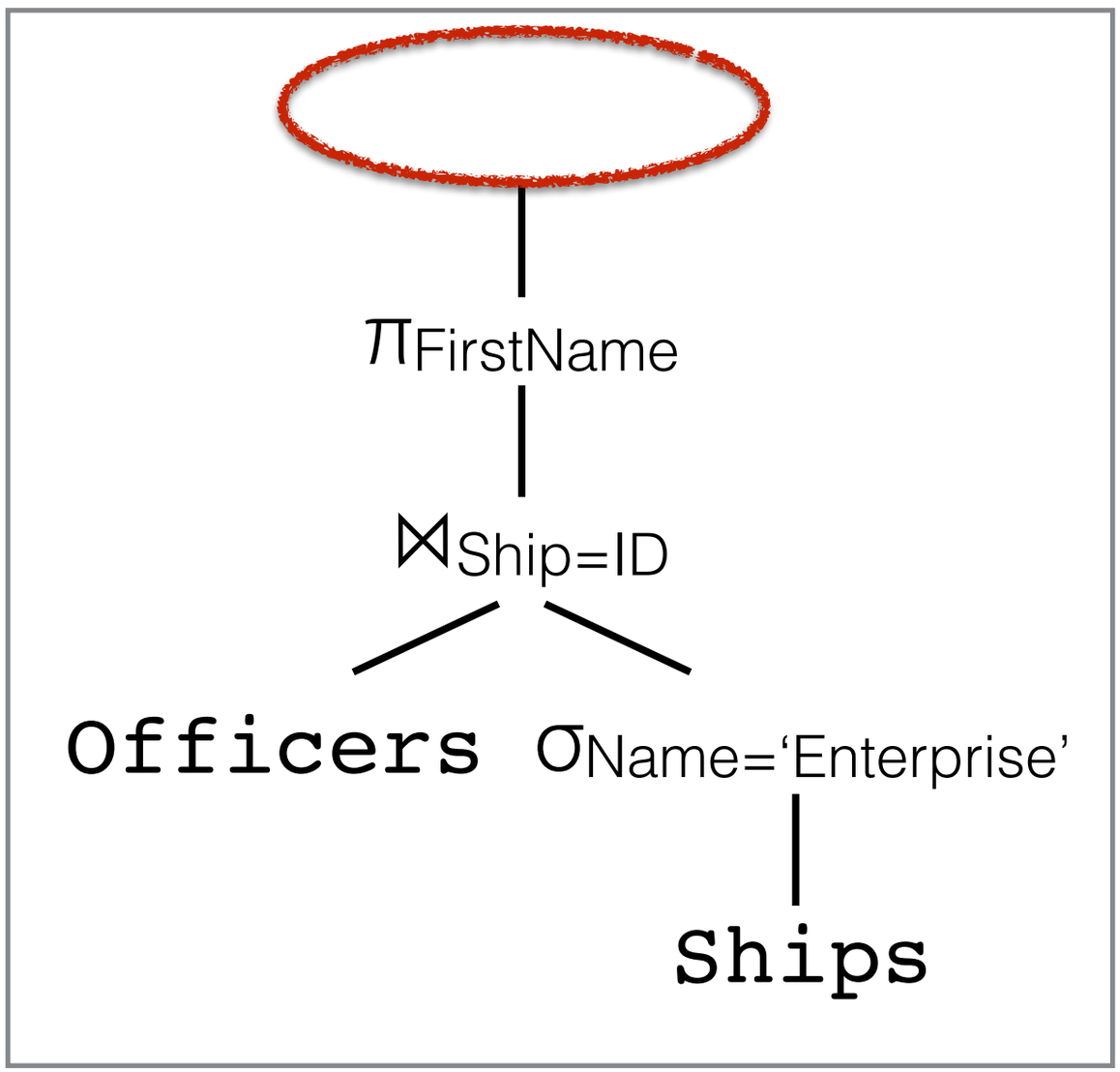
- **Expensive**: Lots of Bulk Copies
- **Cache Locality**: Repeated Scans over Full Tables
- **Memory Use**: Working Set is a Full Table (or more)

How do we do better?

The Memory Hierarchy and You

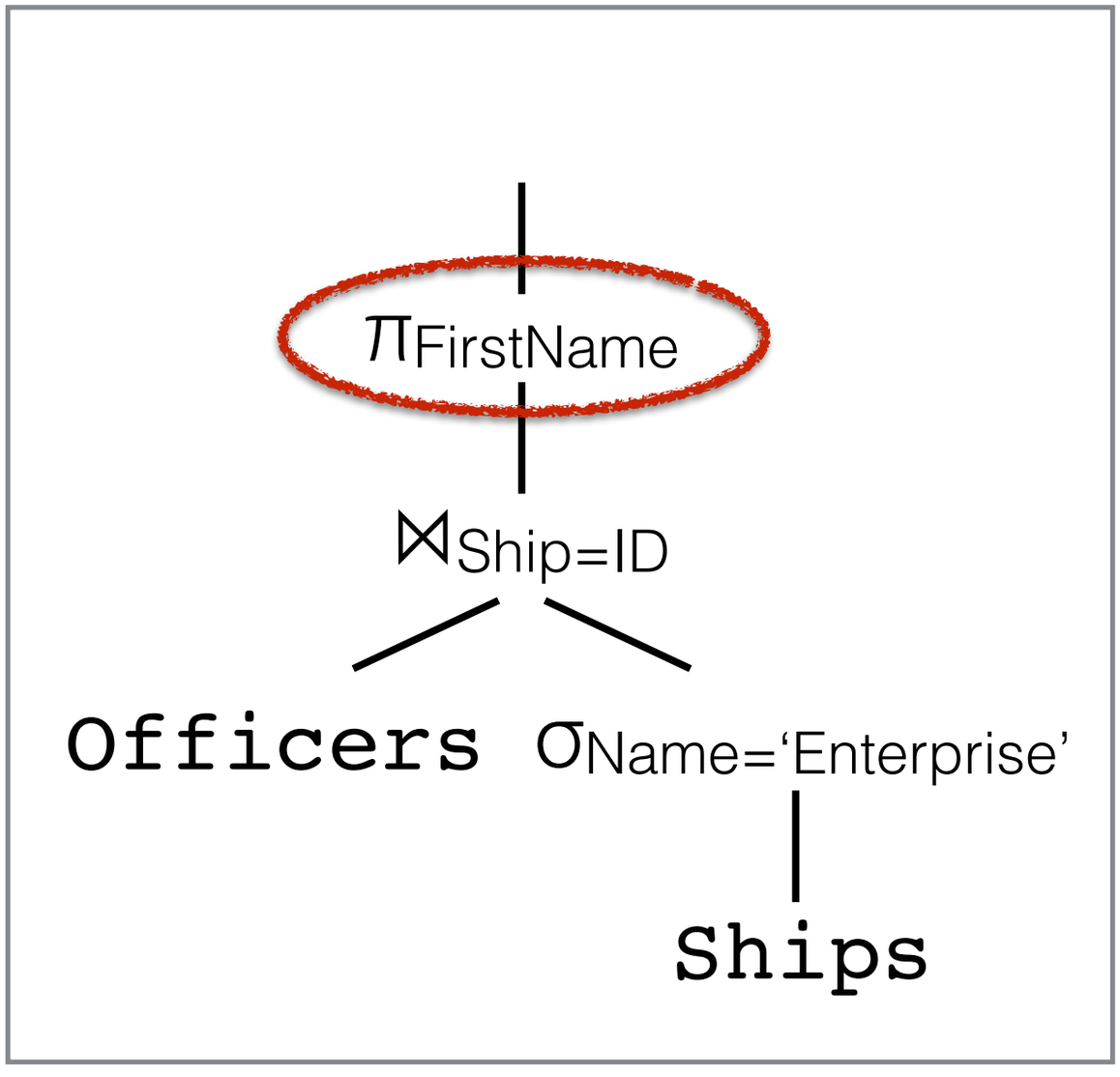
- We want to keep data as close to the CPU as possible
  - Faster memory == Smaller memory
- **Solution 1:** Minimize the Working Set Size!
  - (the memory used at any one time)
- **Solution 2:** Aggressively Batch & Reuse Data

Volcano Evaluation



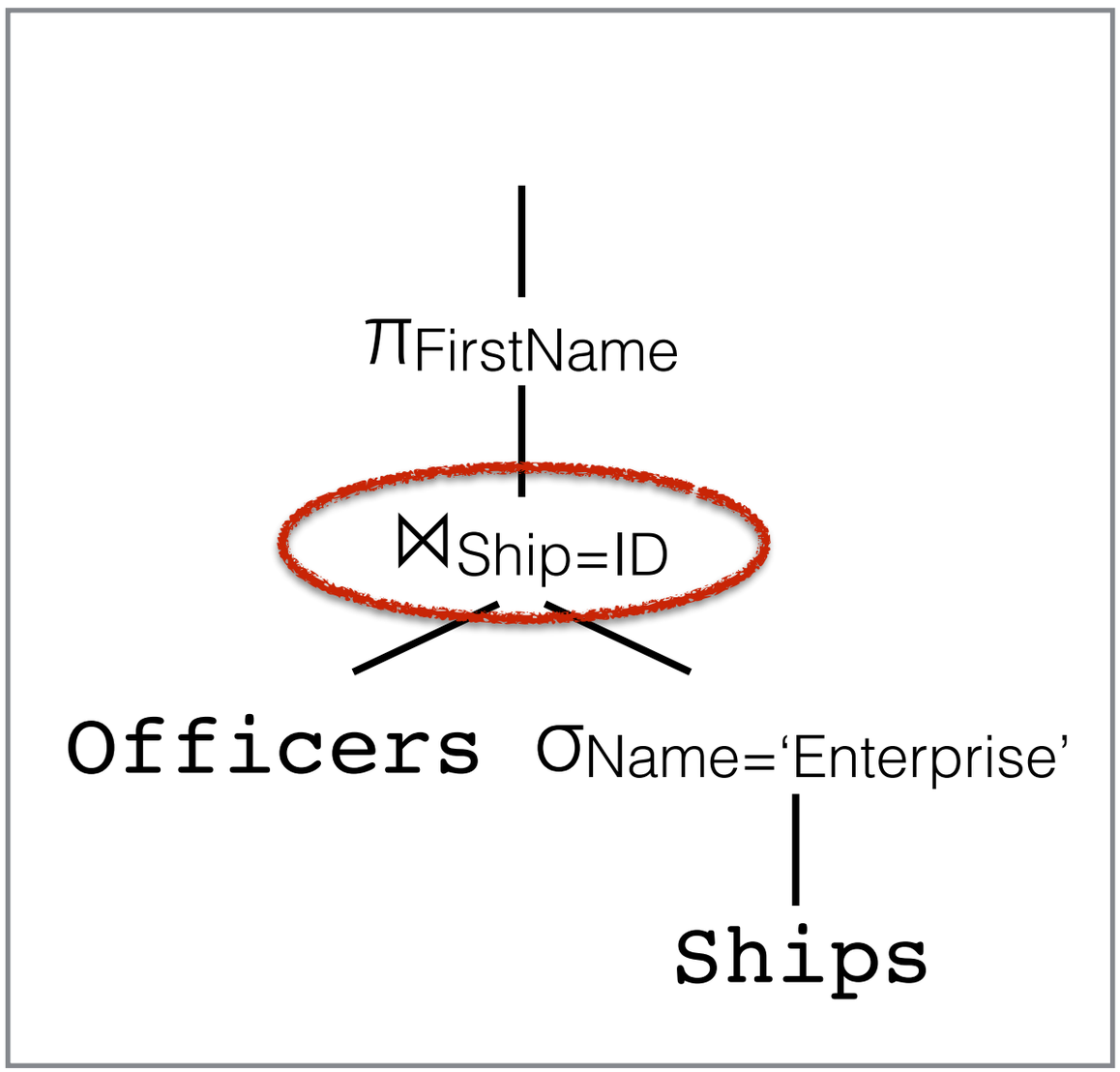
Compute one tuple

Volcano Evaluation



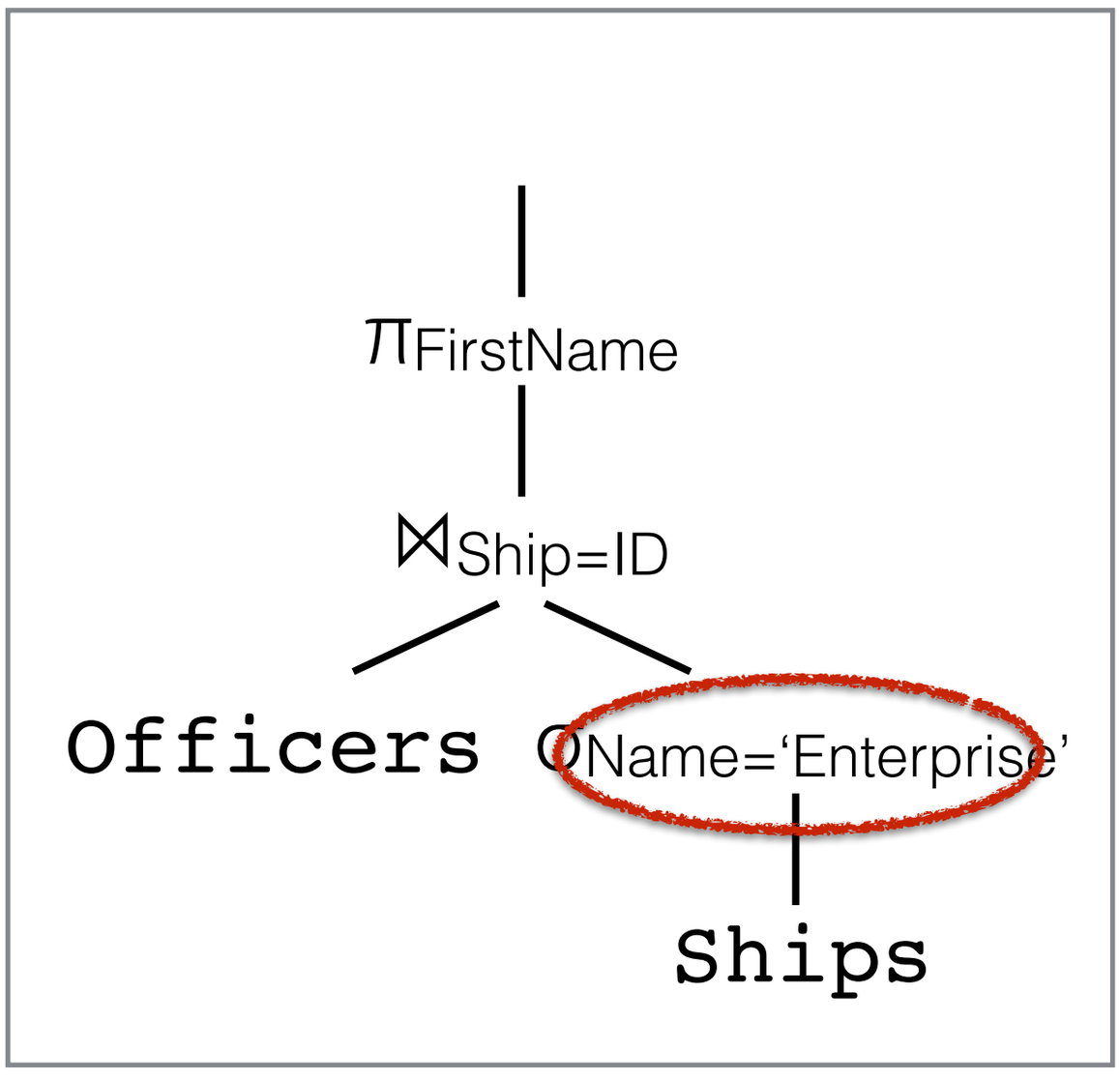
Compute one tuple
Dependency: Next tuple from π

Volcano Evaluation



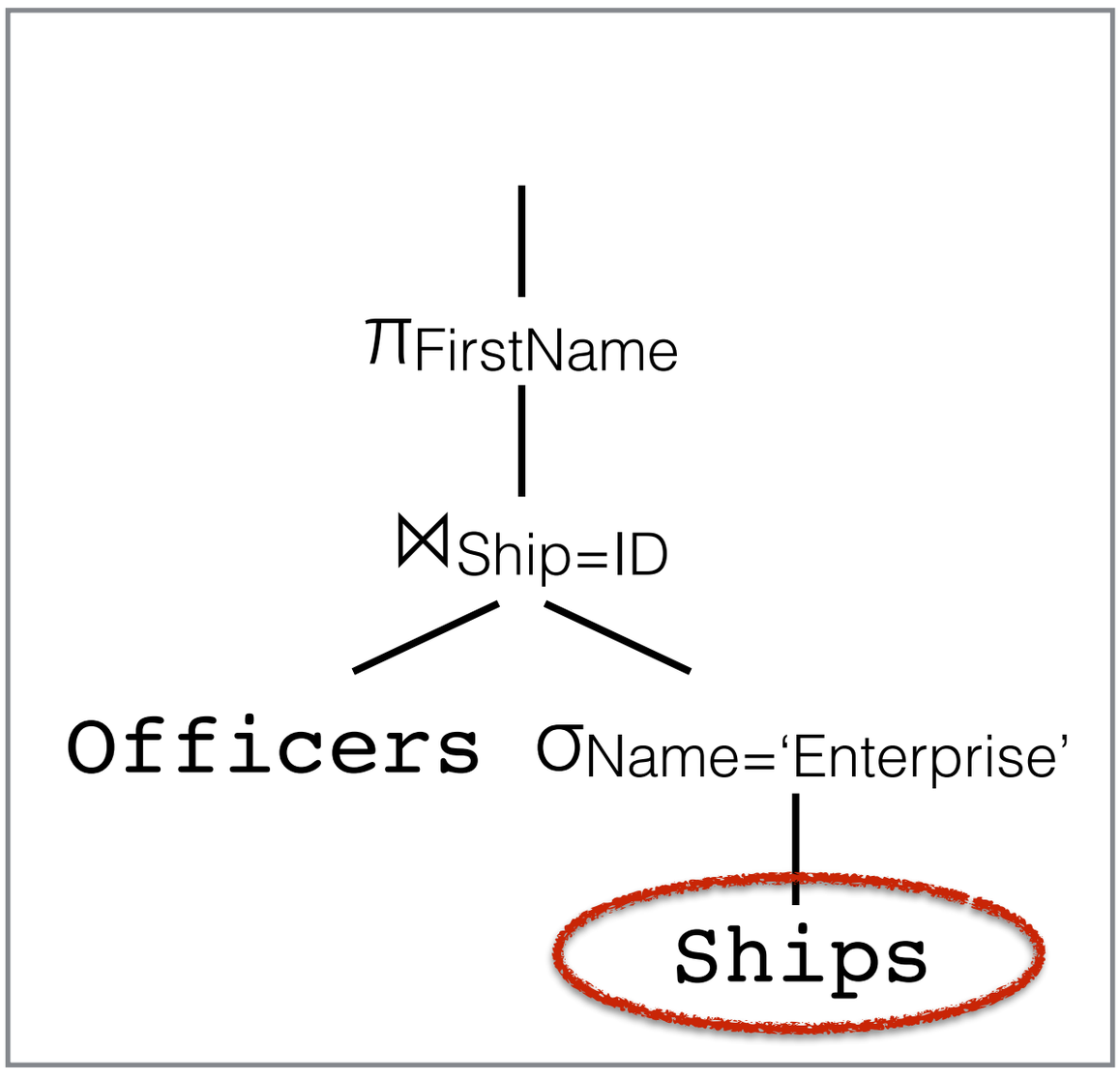
Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈

Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈
Dependency: Next tuple from σ

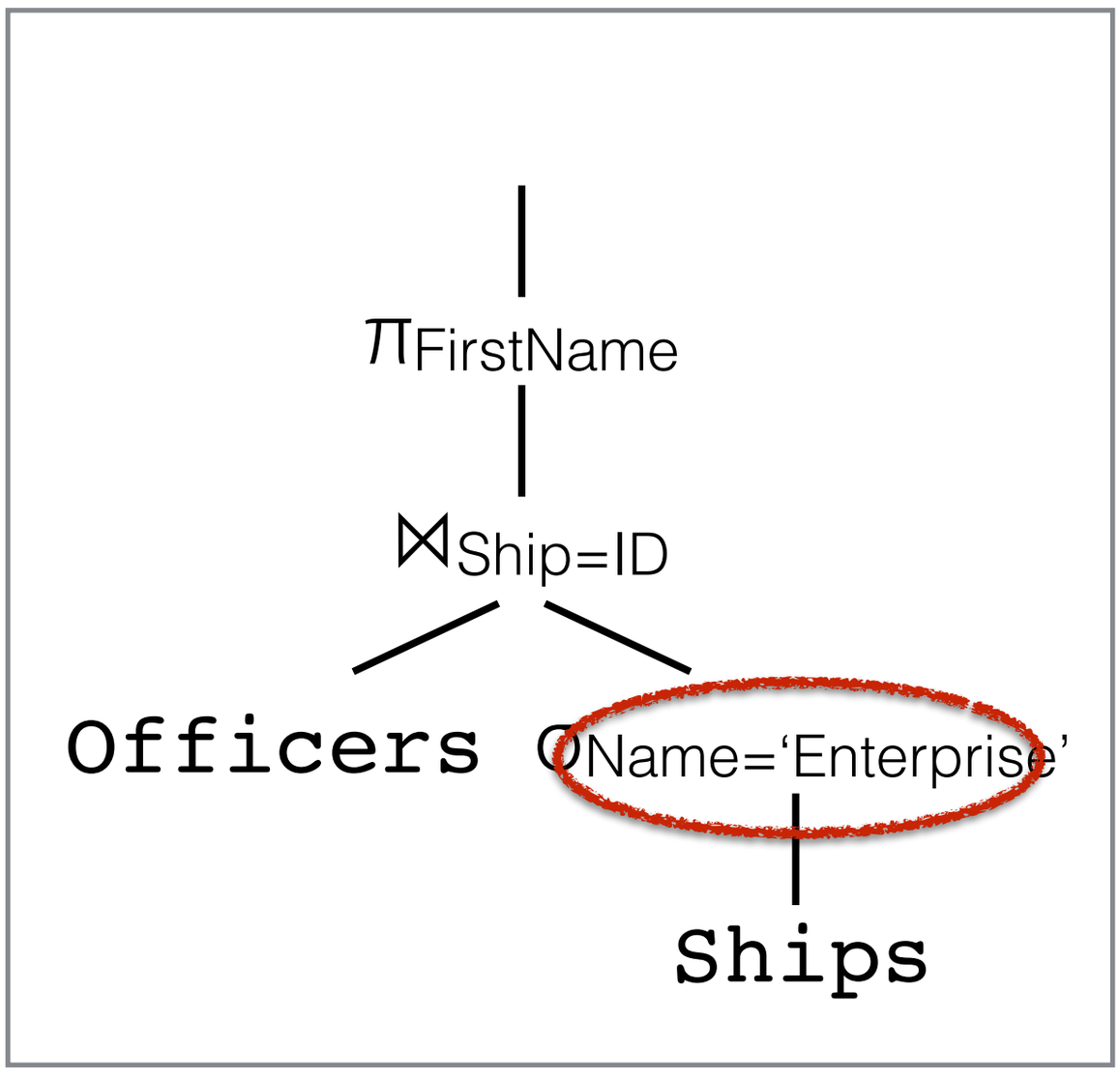
Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈
Dependency: Next tuple from σ
Dependency: Tuple from `Ships`

ID,	Name
[1701,	Enterprise]

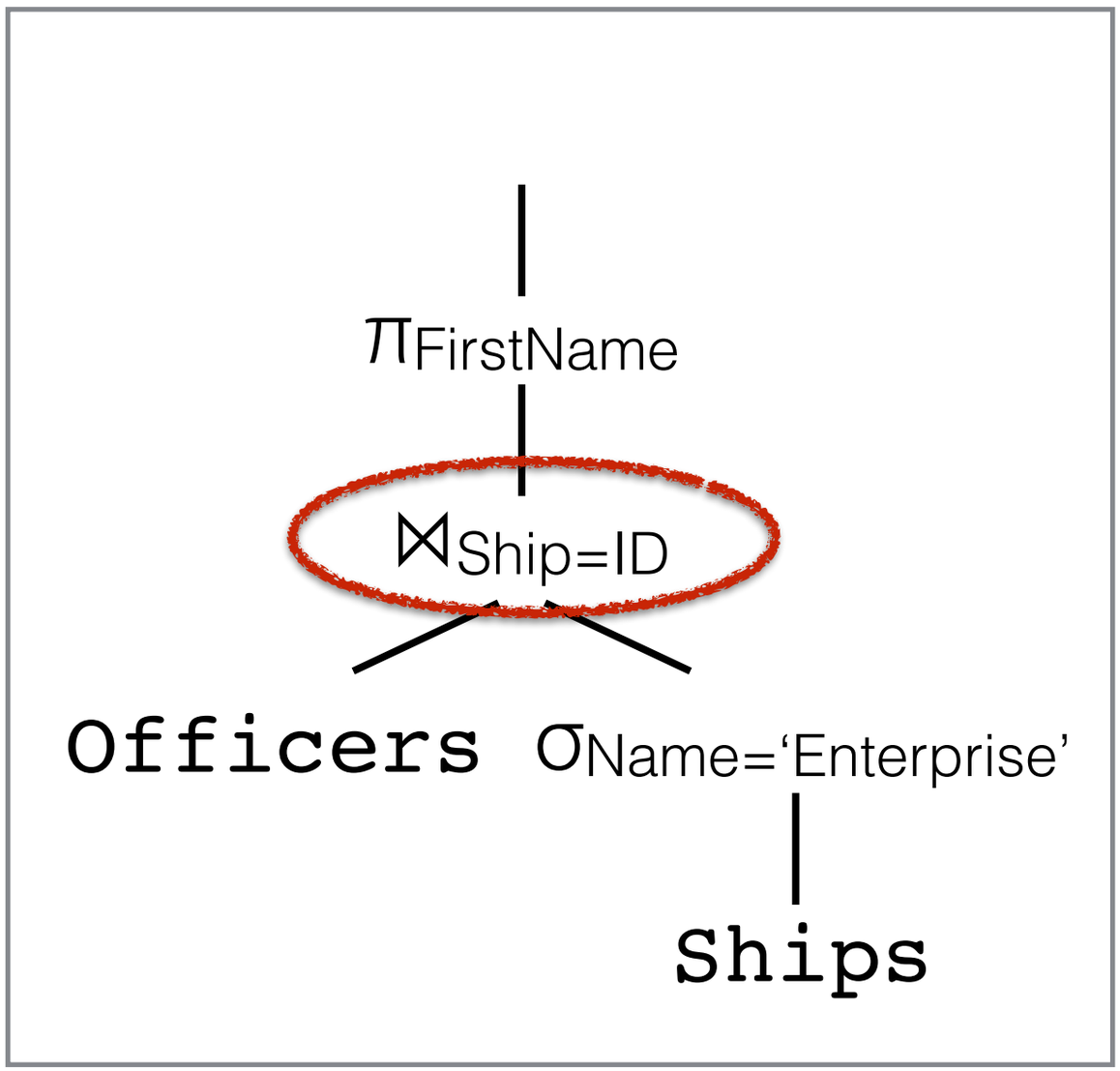
Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈
Dependency: Next tuple from σ

ID,	Name
[1701,	Enterprise]

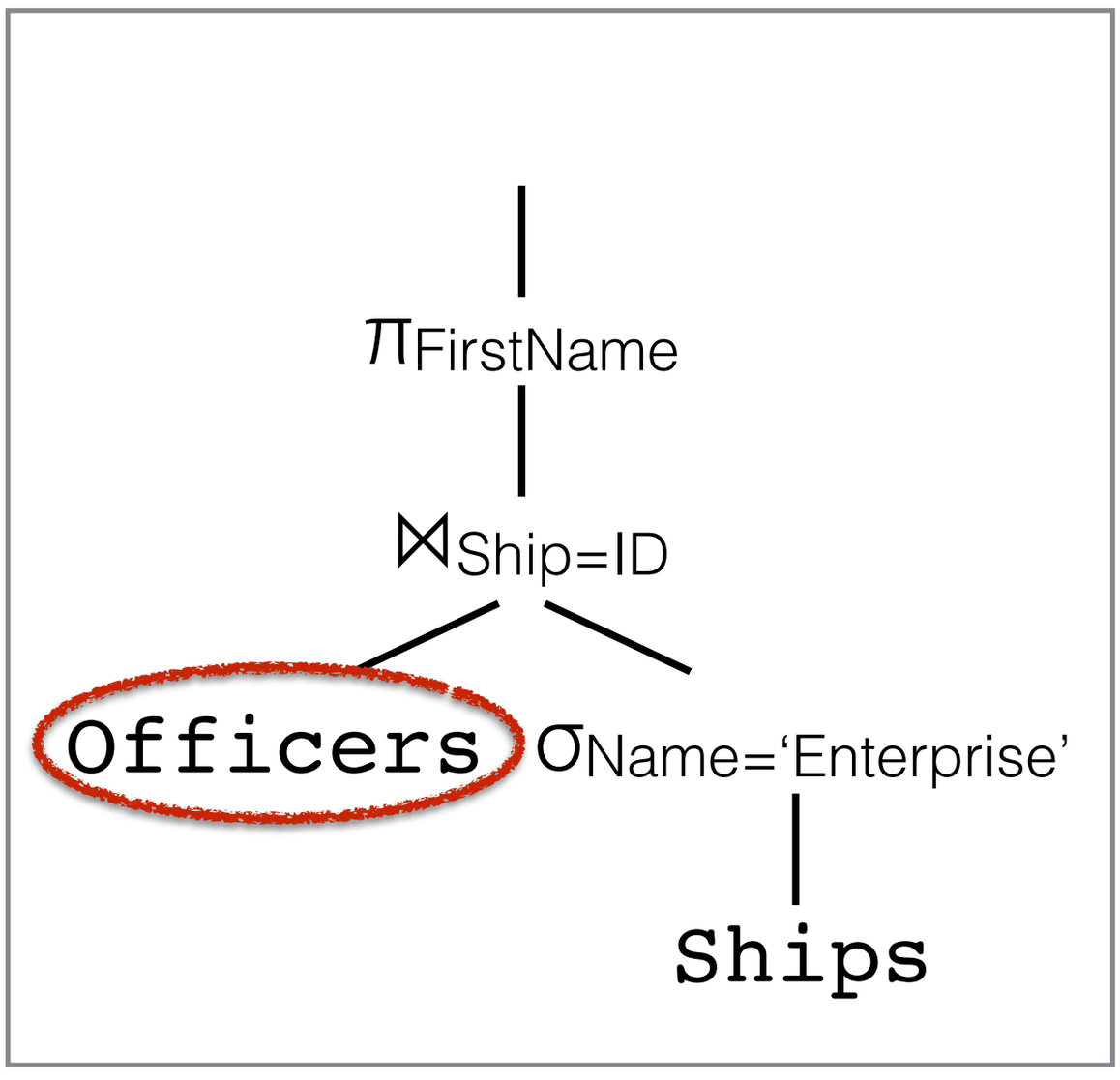
Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈

ID,	Name
[1701,	Enterprise]

Volcano Evaluation

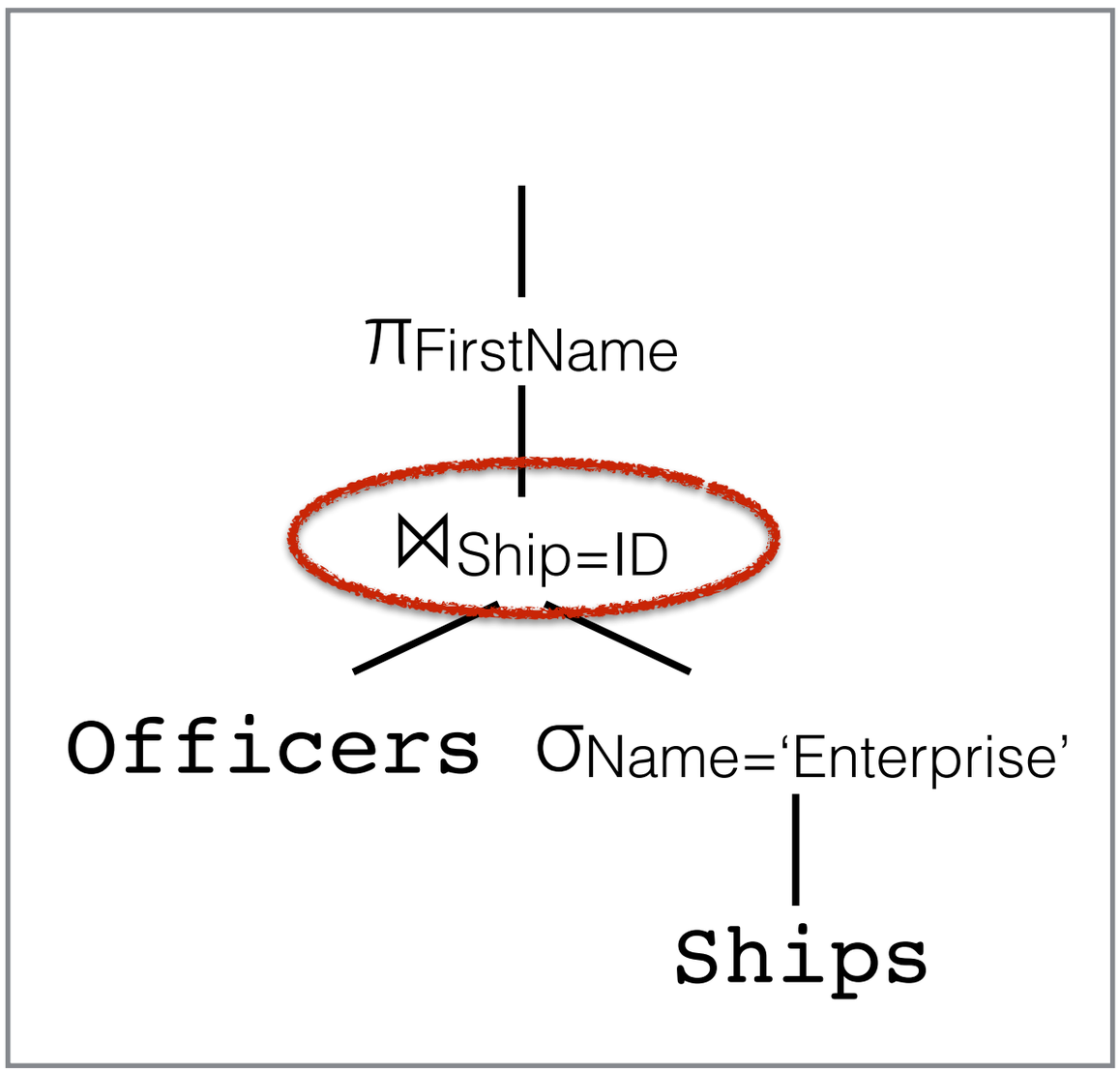


Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈
Dependency: Tuple f. `Officers`

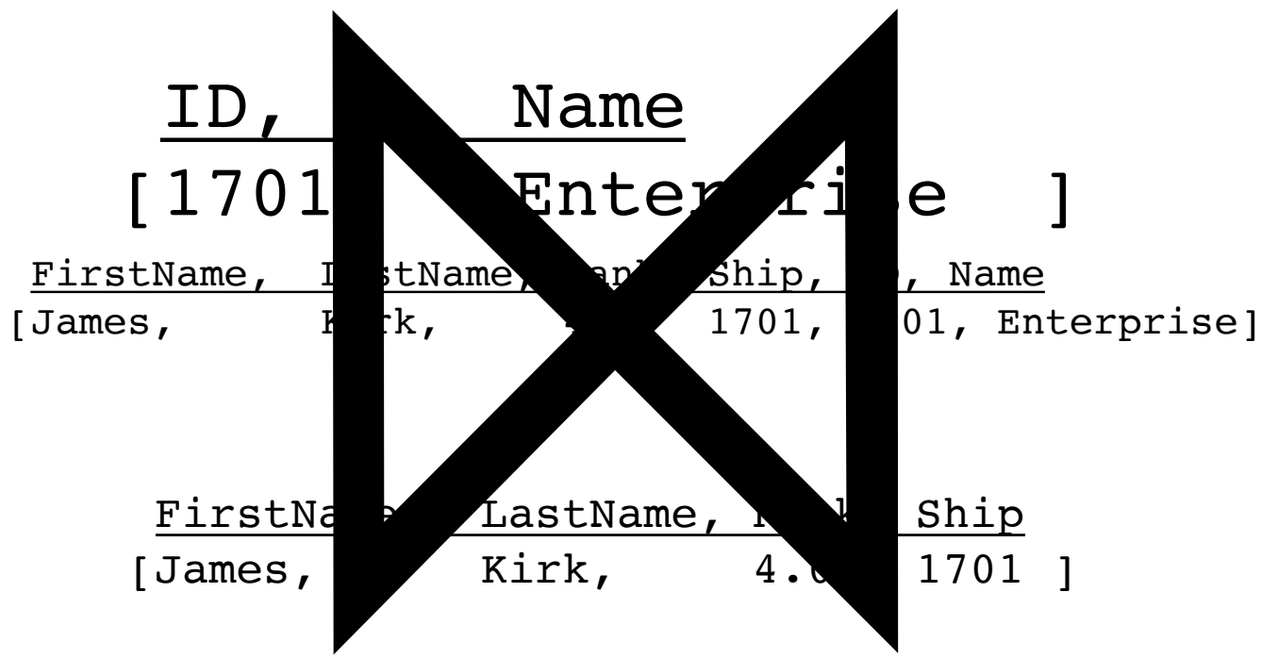
ID,	Name
[1701,	Enterprise]

FirstName,	LastName,	Rank,	Ship
[James,	Kirk,	4.0,	1701]

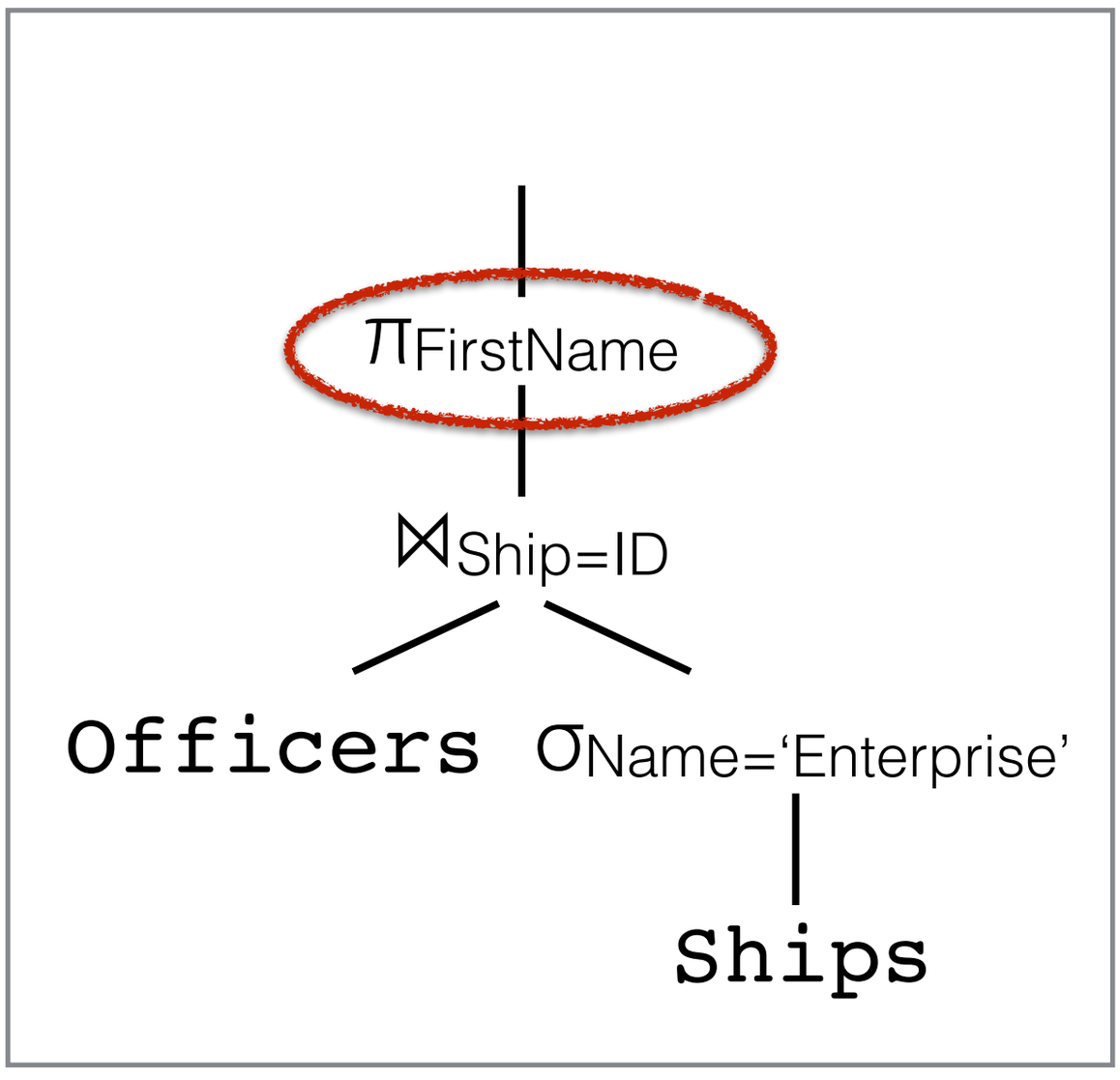
Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π
Dependency: Next tuple from ⋈



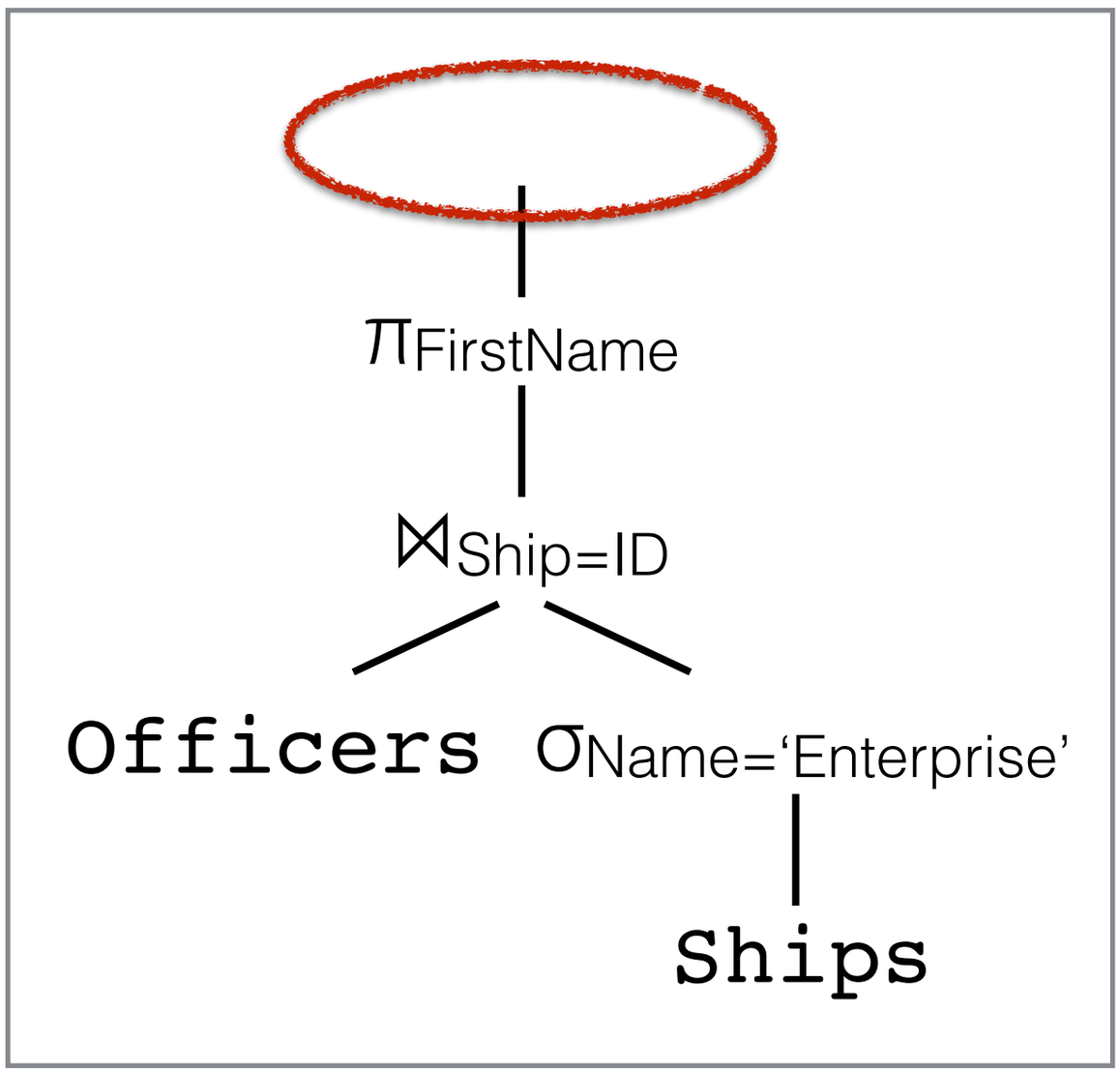
Volcano Evaluation



Compute one tuple
Dependency: Next tuple from π

FirstName,	LastName,	Rank,	Ship,	ID,	Name
[James,]	Kirk,	4.0,	1701,	1701,	Enterprise]

Volcano Evaluation



Compute one tuple

FirstName
[James]

Iterators

```
void open() {
  // call open() on child iterators
  // prepare the iterator
}

Tuple getNext() {
  // read, process, and return a tuple
}

void close() {
  // clean-up the iterator
  // call close() on child iterators
}
```

GetNext()

Relation

Read One Line from File

↓

Split Line into Fields

↓

Parse Field Types

↓

Return Tuple

What is the Working Set Size?

GetNext()

Projection (π)

Read One Tuple

↓

Compute Projected Attributes

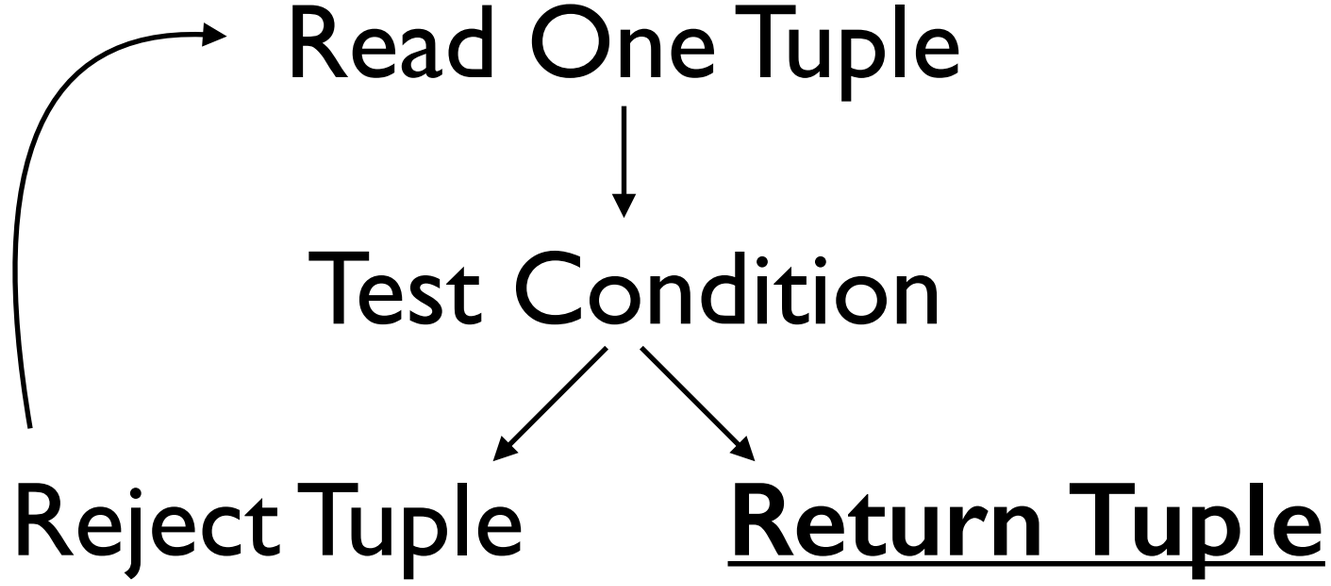
↓

Return Tuple

What is the Working Set Size?

GetNext()

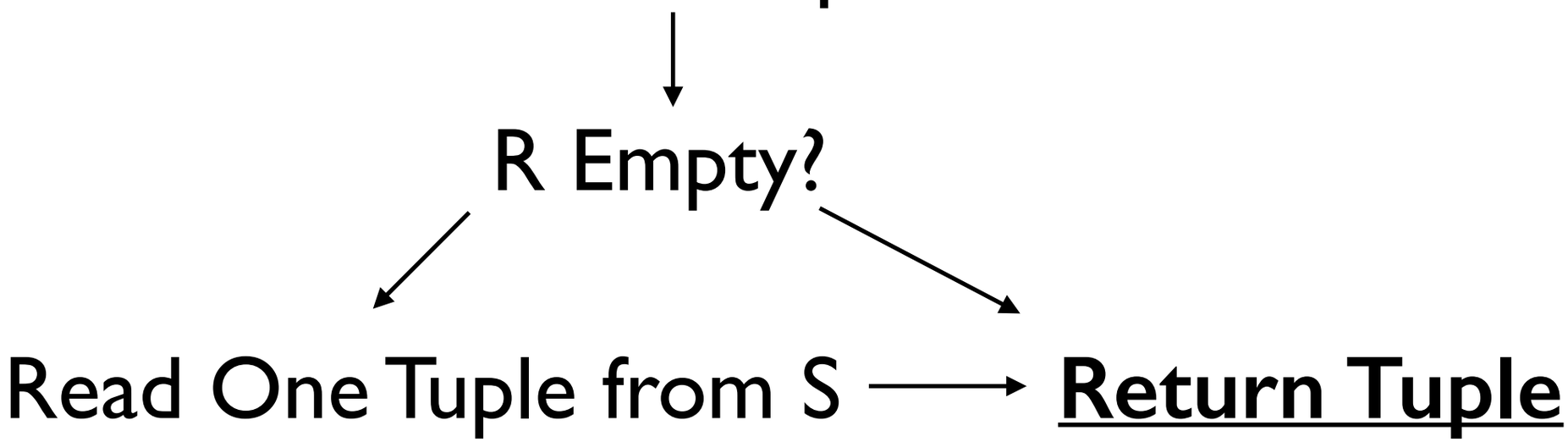
Selection (σ)



What is the Working Set Size?

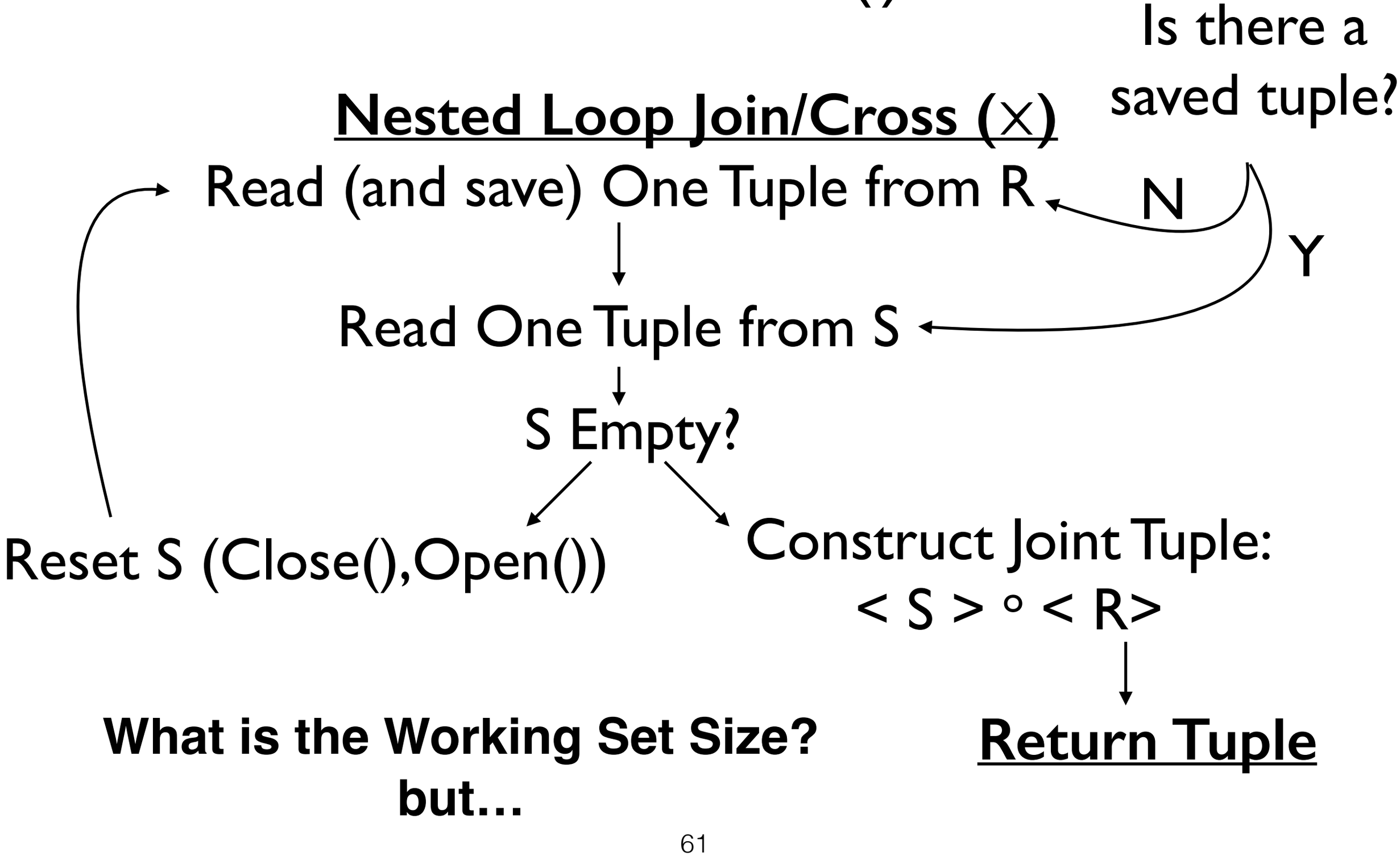
GetNext()

Union (U)



What is the Working Set Size?

GetNext()



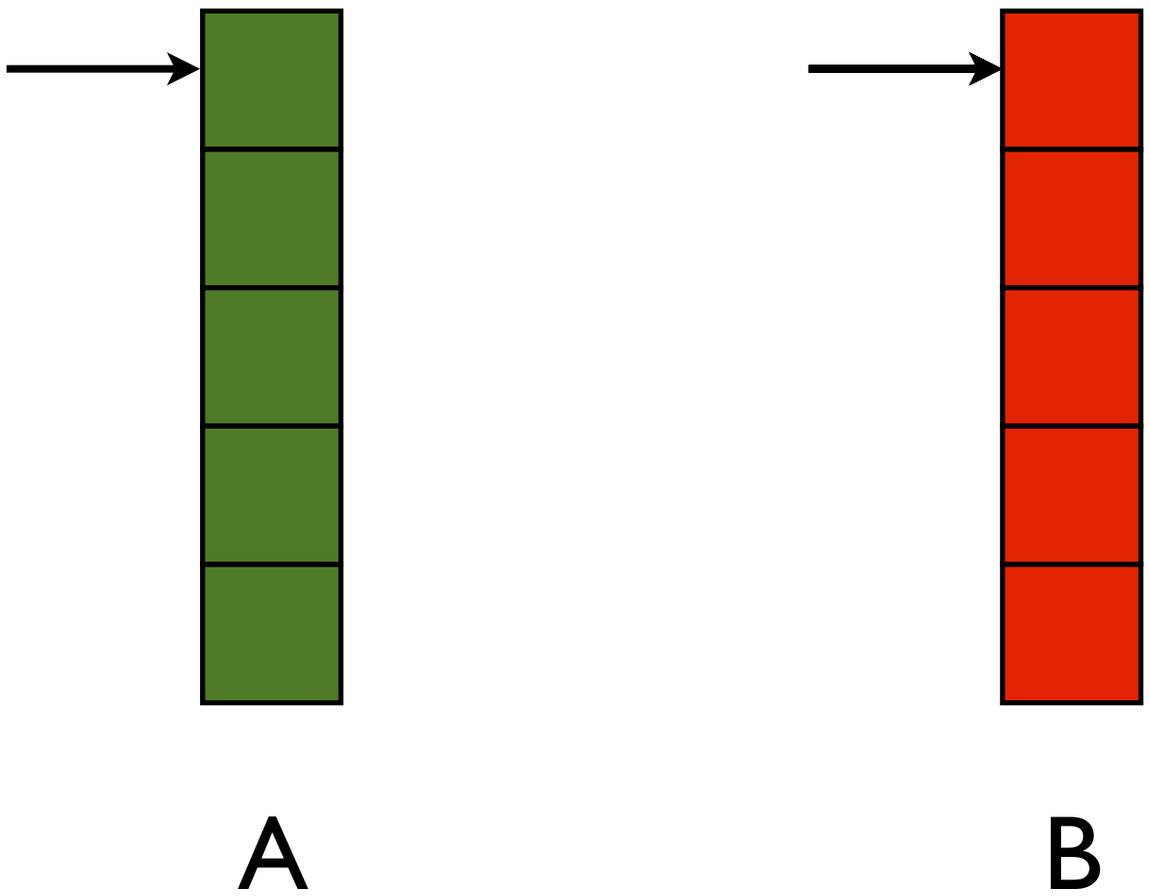
What is the Working Set Size?

but…

Implementing: Joins

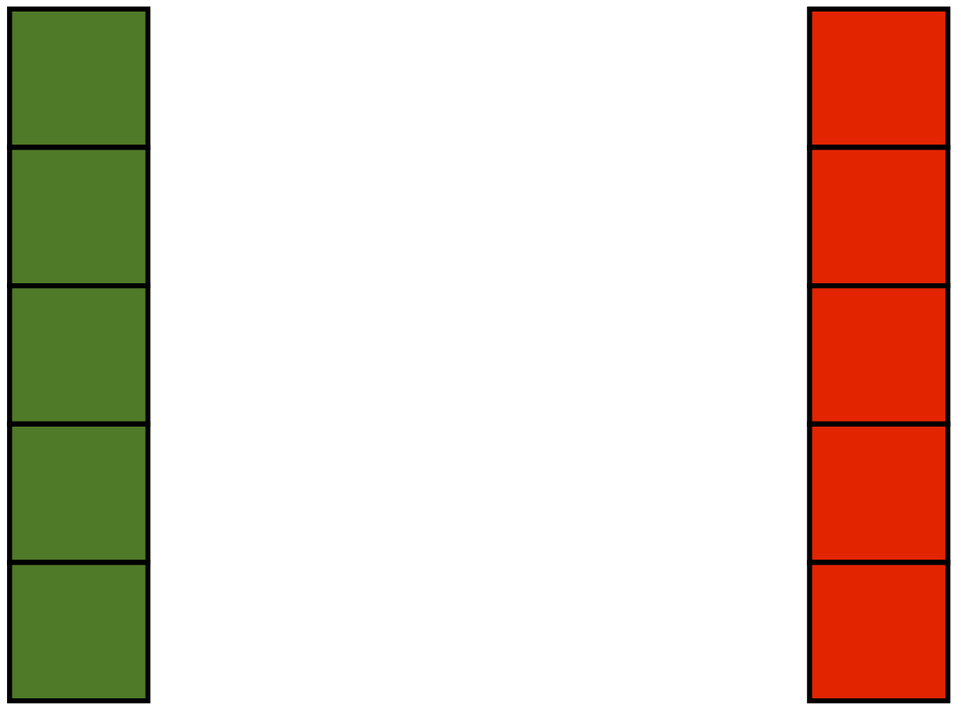
Solution 1 (Nested-Loop)

For Each (a in A) { For Each (b in B) { emit (a, b); }}



Implementing: Joins

Solution 2 (Block-Nested-Loop)



Implementing: Joins

Solution 2 (Block-Nested-Loop)

1) Partition into Blocks 2) NLJ on each pair of blocks