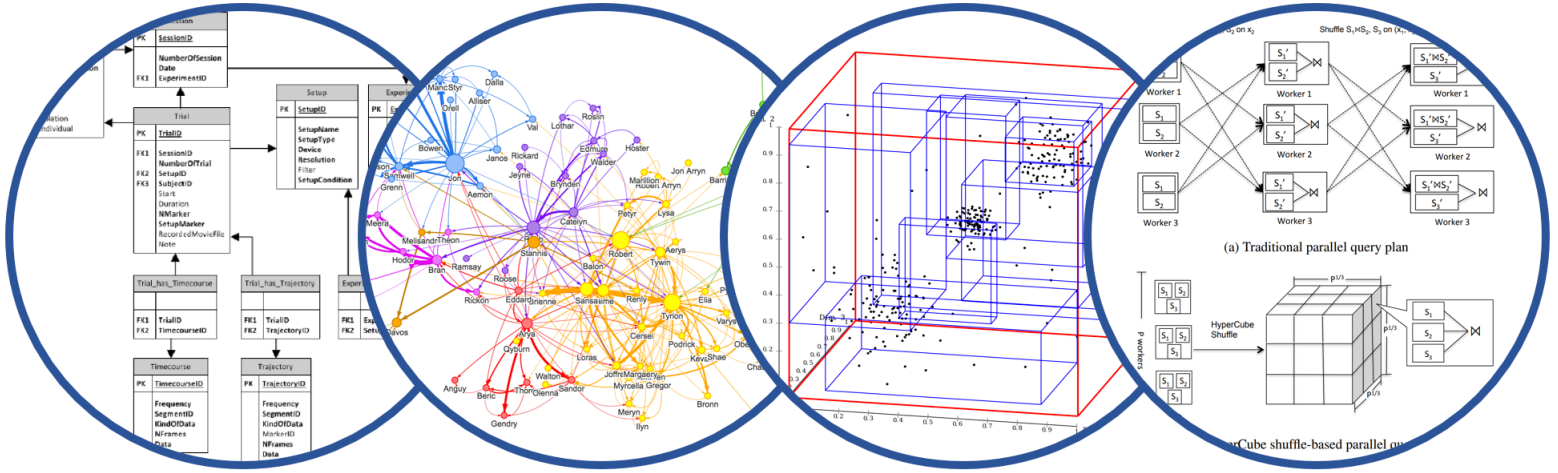


Database System Internals

Architecture

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- Lab 1 part 1 is due on Monday
  - "git pull upstream master" before building
  - Remember to git commit and git push often!
- HW1 is due next week on Friday
  - gradescope
- 544M paper review is due in two weeks
  - Email to me

What we already know…

- Database = collection of related files
- DBMS = program that manages the database

What we already know…

- Data models: relational, semi-structured (XML), graph (RDF), key-value pairs

- Relational model: defines only the logical model, and does not define a physical storage of the data

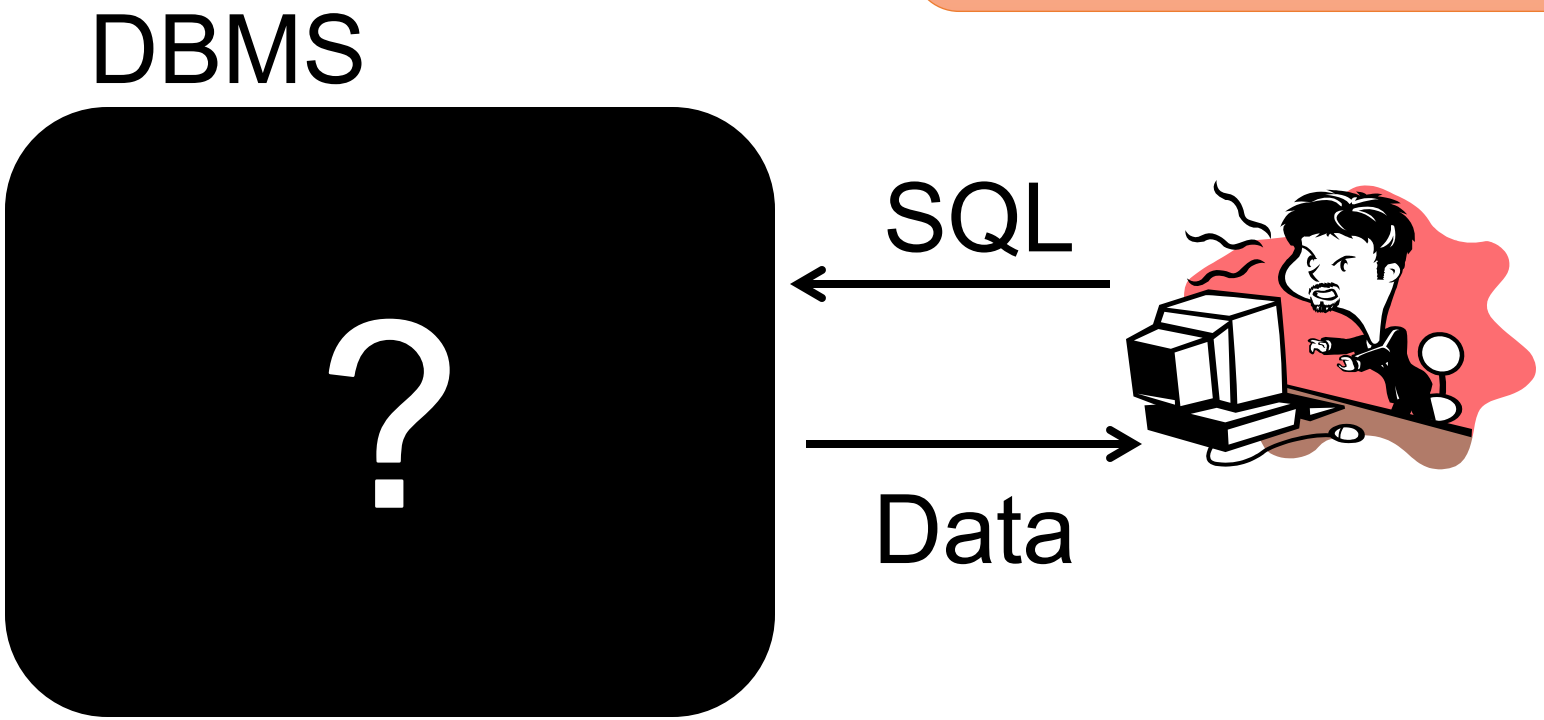
What we already know…

Relational Query Language:

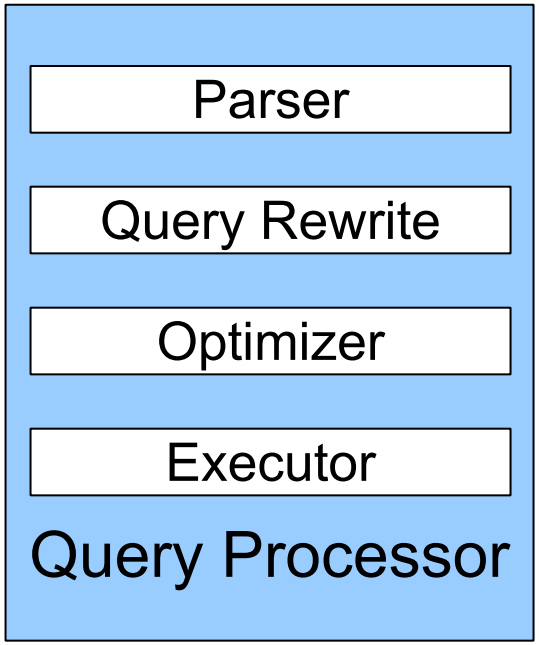
- Set-at-a-time: instead of tuple-at-a-time
- Declarative: user says what they want and not how to get it
- Query optimizer: from *what* to *how*

How to Implement a Relational DBMS?

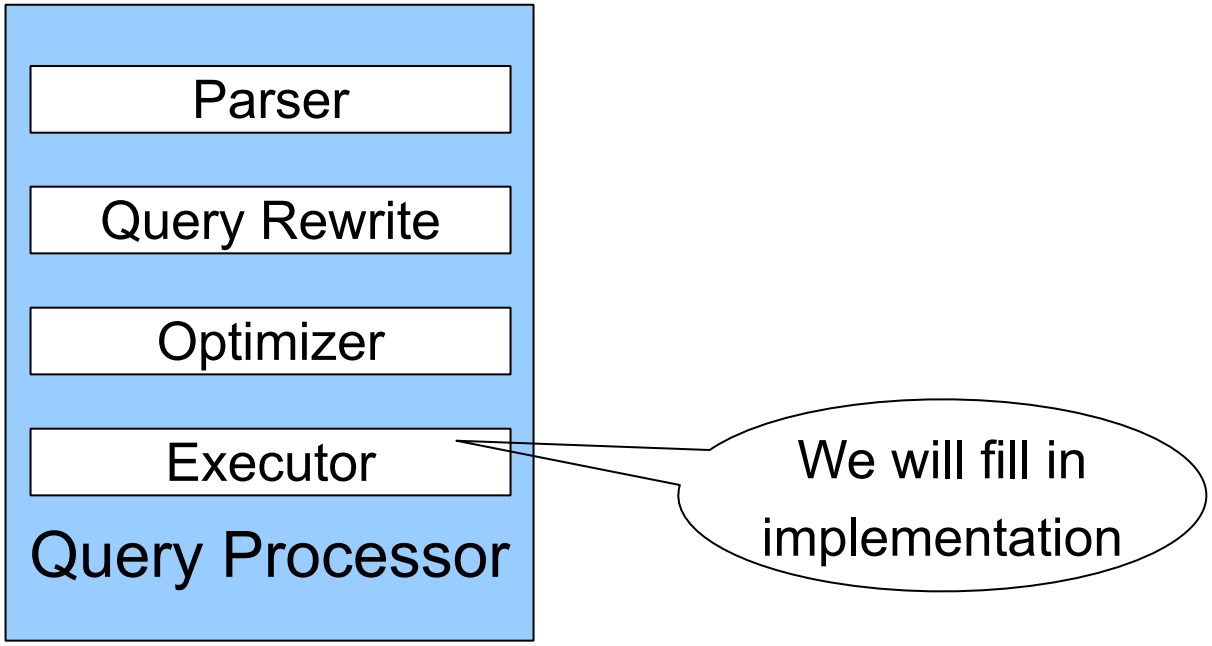
Key challenge:
Achieve high performance on large databases!



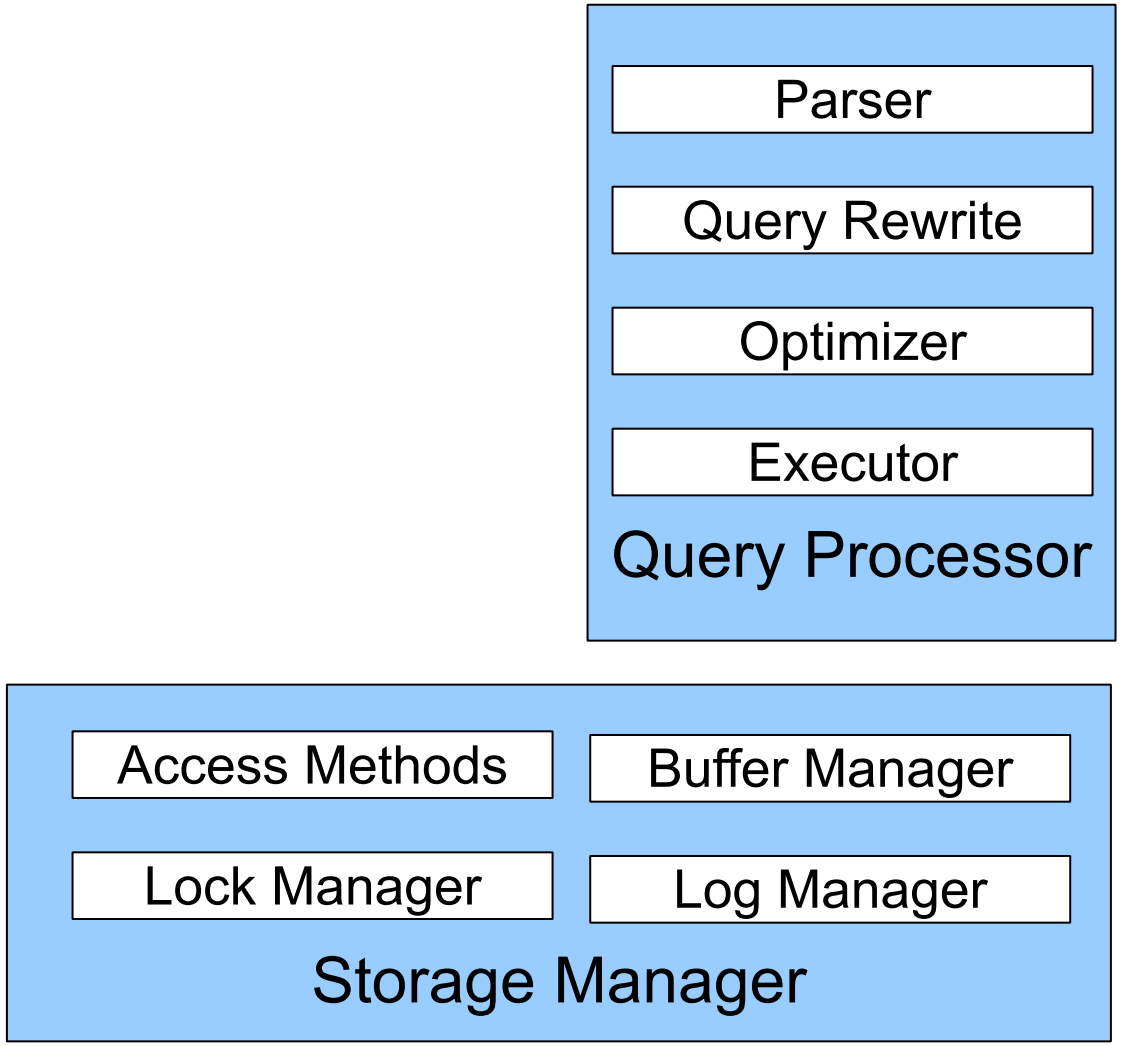
DBMS Architecture



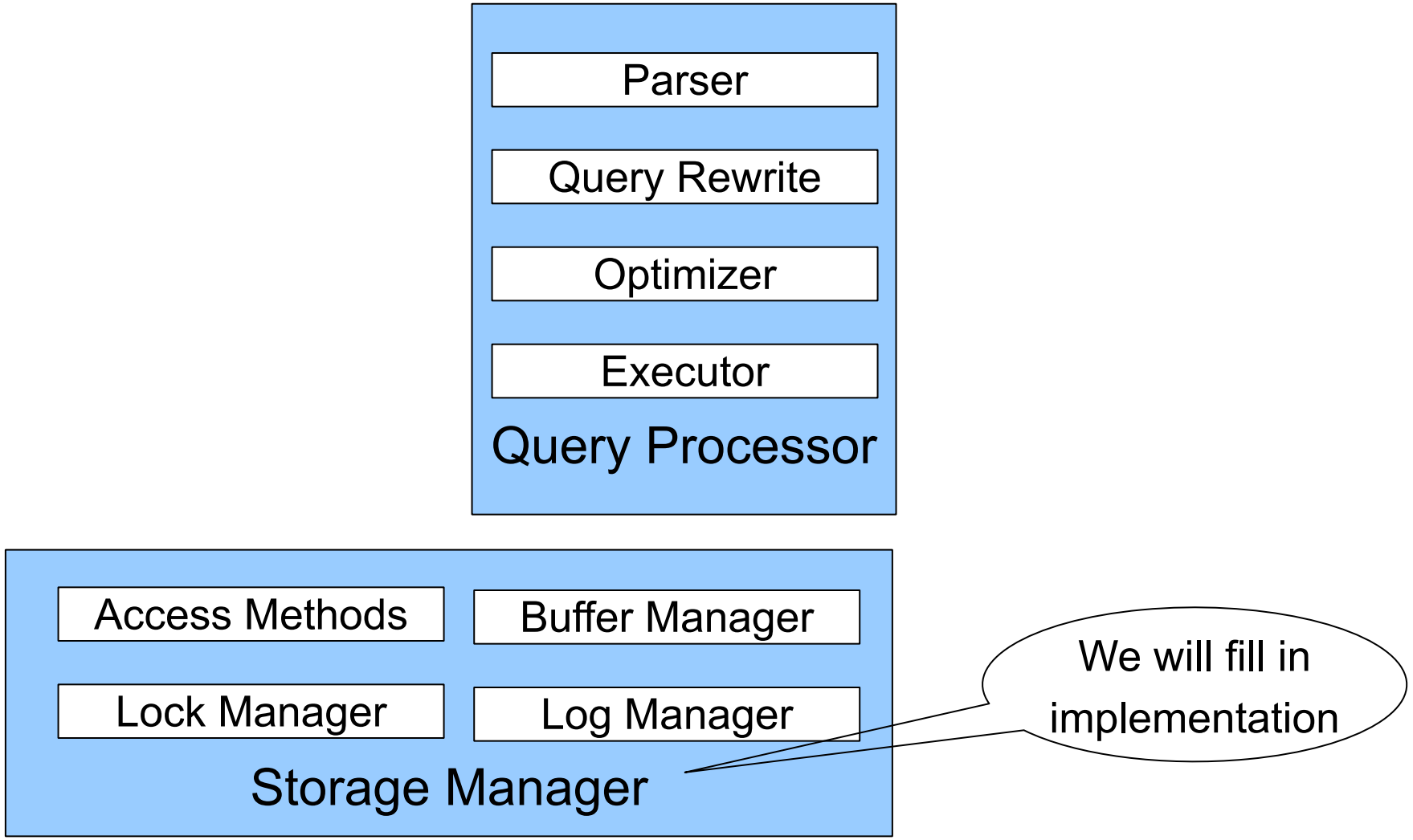
DBMS Architecture



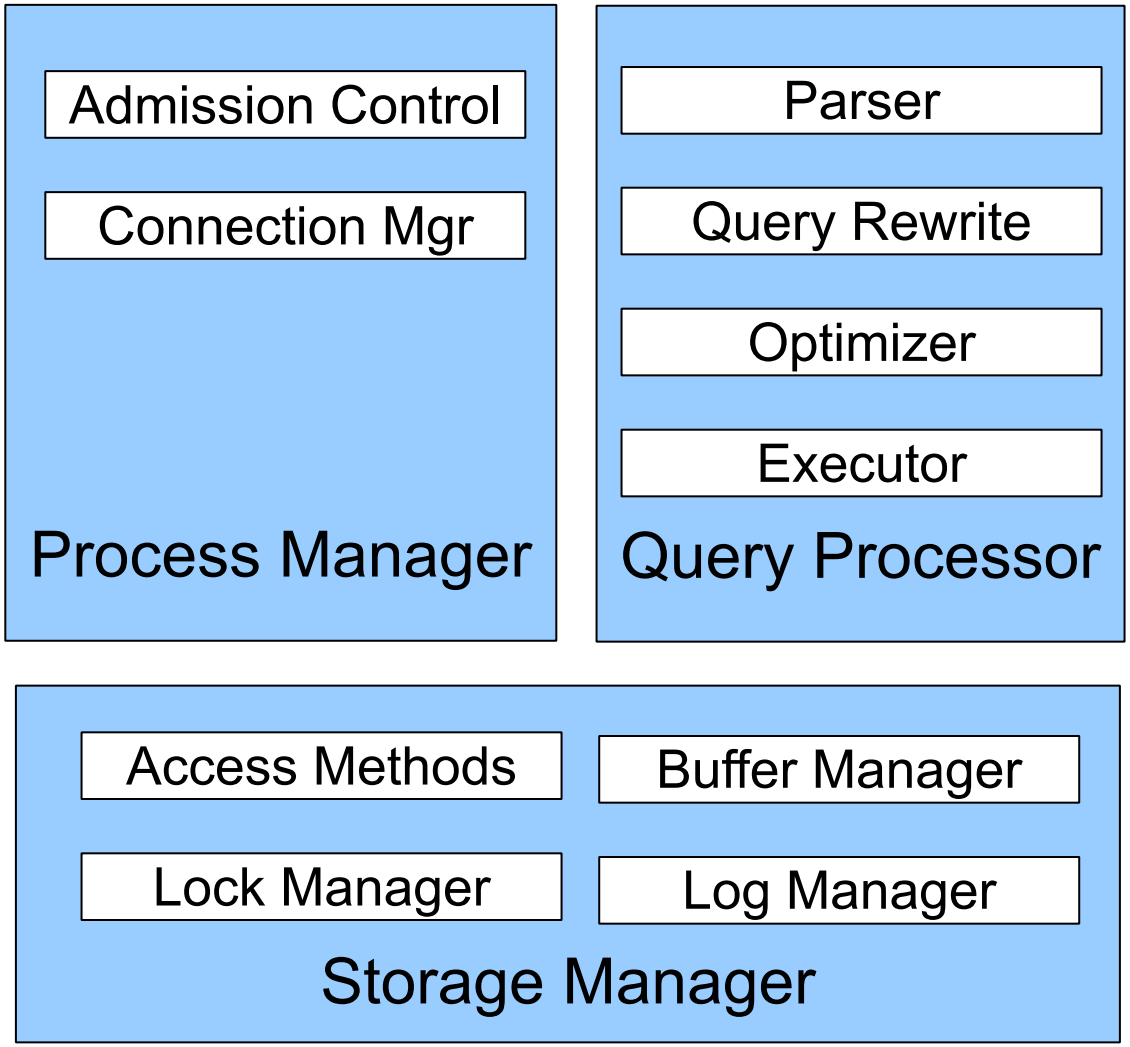
DBMS Architecture



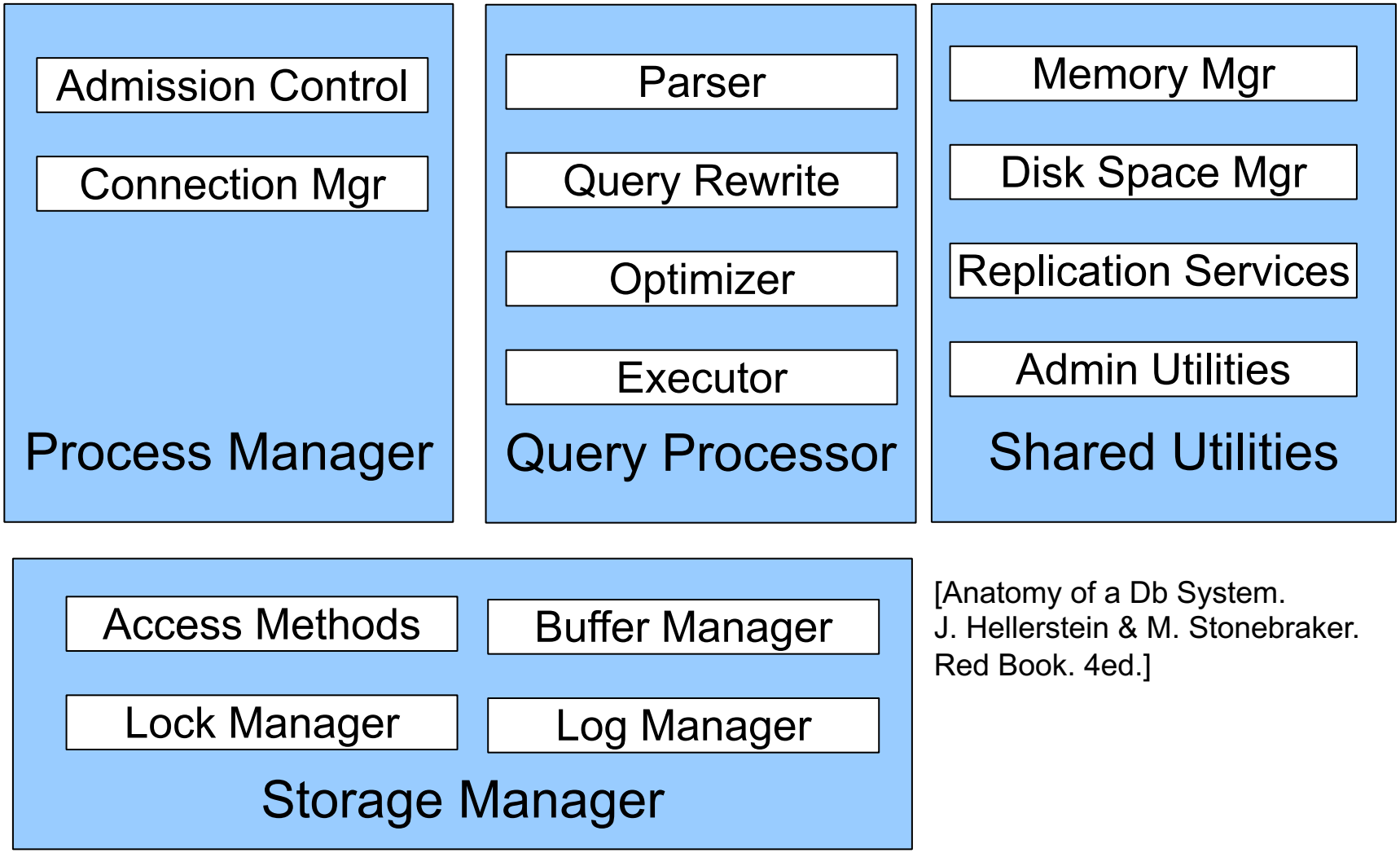
DBMS Architecture



DBMS Architecture



DBMS Architecture



[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Goal for Today

Overview of query execution

Overview of storage manager

Query Processor

Example Database Schema

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

View: Suppliers in Seattle

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
```

Example Query

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

- Find the names of all suppliers in Seattle who supply part number 2

```
SELECT sno, sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Query Processor

- **Step 1: Parser**
  - Parses query into an internal format
  - Performs various checks using **catalog**

- **Step 2: Query rewrite**
  - View rewriting, flattening, etc.

Rewritten Version of Our Query

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

Original query:

```
SELECT sno, sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Original view:

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
```

Rewritten Version of Our Query

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

Original query:

```
SELECT sno, sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Original view:

```
CREATE VIEW NearbySupp AS
SELECT sno, sname
FROM Supplier
WHERE scity='Seattle' AND sstate='WA'
```

Rewritten query (view inlining plus query unnesting):

```
SELECT S.sno, S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

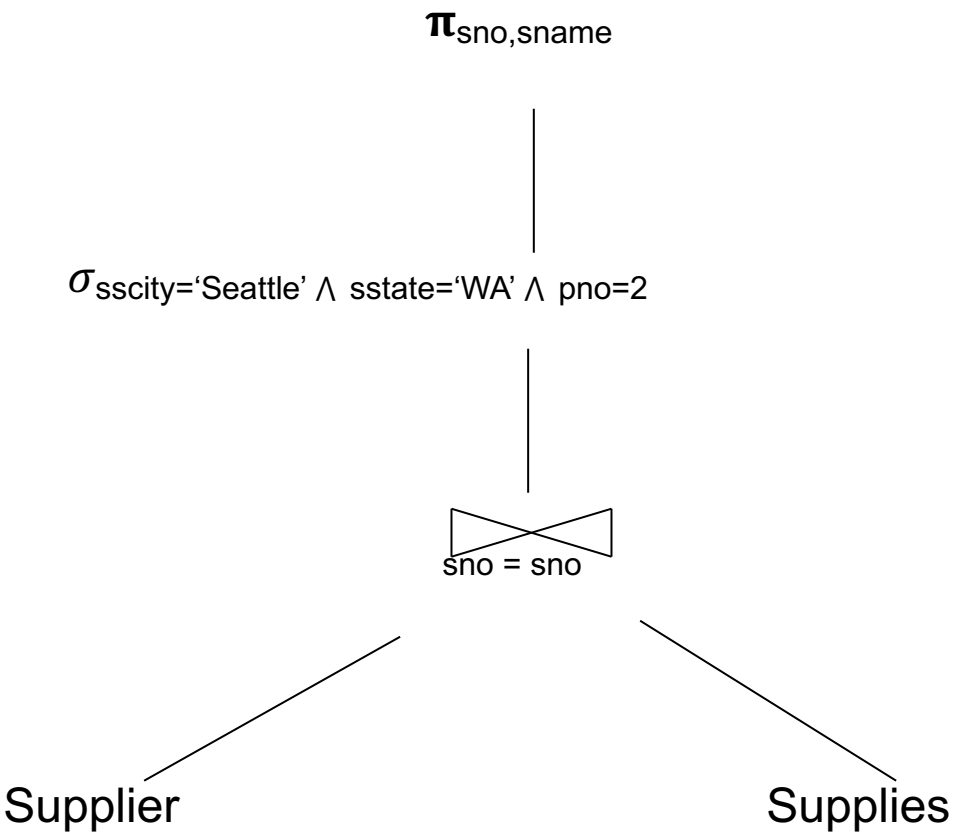
Query Processor

- **Step 3: Optimizer**
  - Find an efficient query plan for executing the query
  - A **query plan** is
    - **Logical**: An extended relational algebra tree
    - **Physical**: With additional annotations at each node
      - Access method to use for each relation
      - Implementation to use for each relational operator
- **Step 4: Executor**
  - Actually executes the physical plan

Logical Query Plan

```
SELECT S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle'
AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```

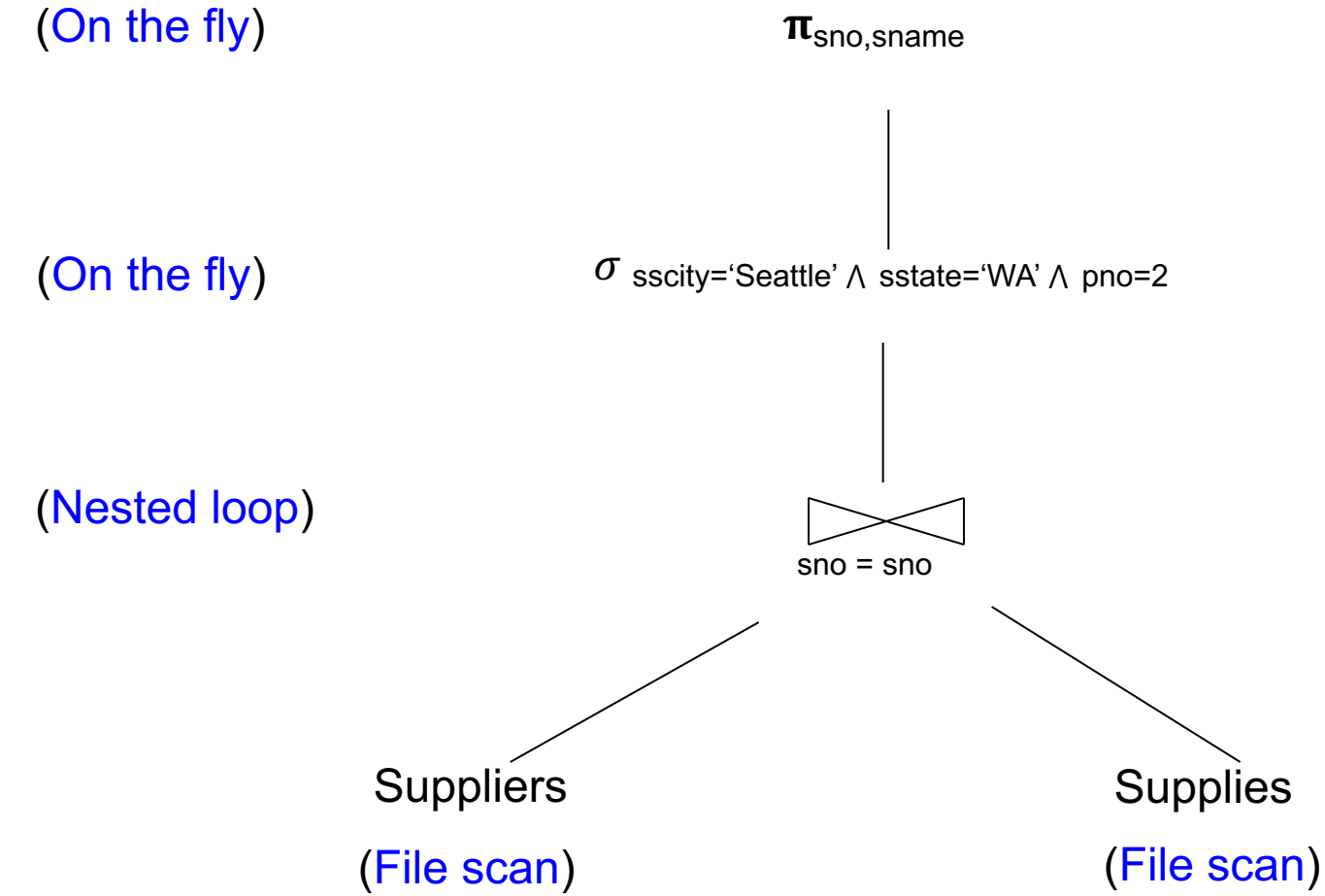


Physical Query Plan

- Logical query plan with extra annotations
- **Implementation choice** for each operator
- **Access path selection** for each relation
  - Bottom of tree = read from disk
  - Use a file scan or use an index

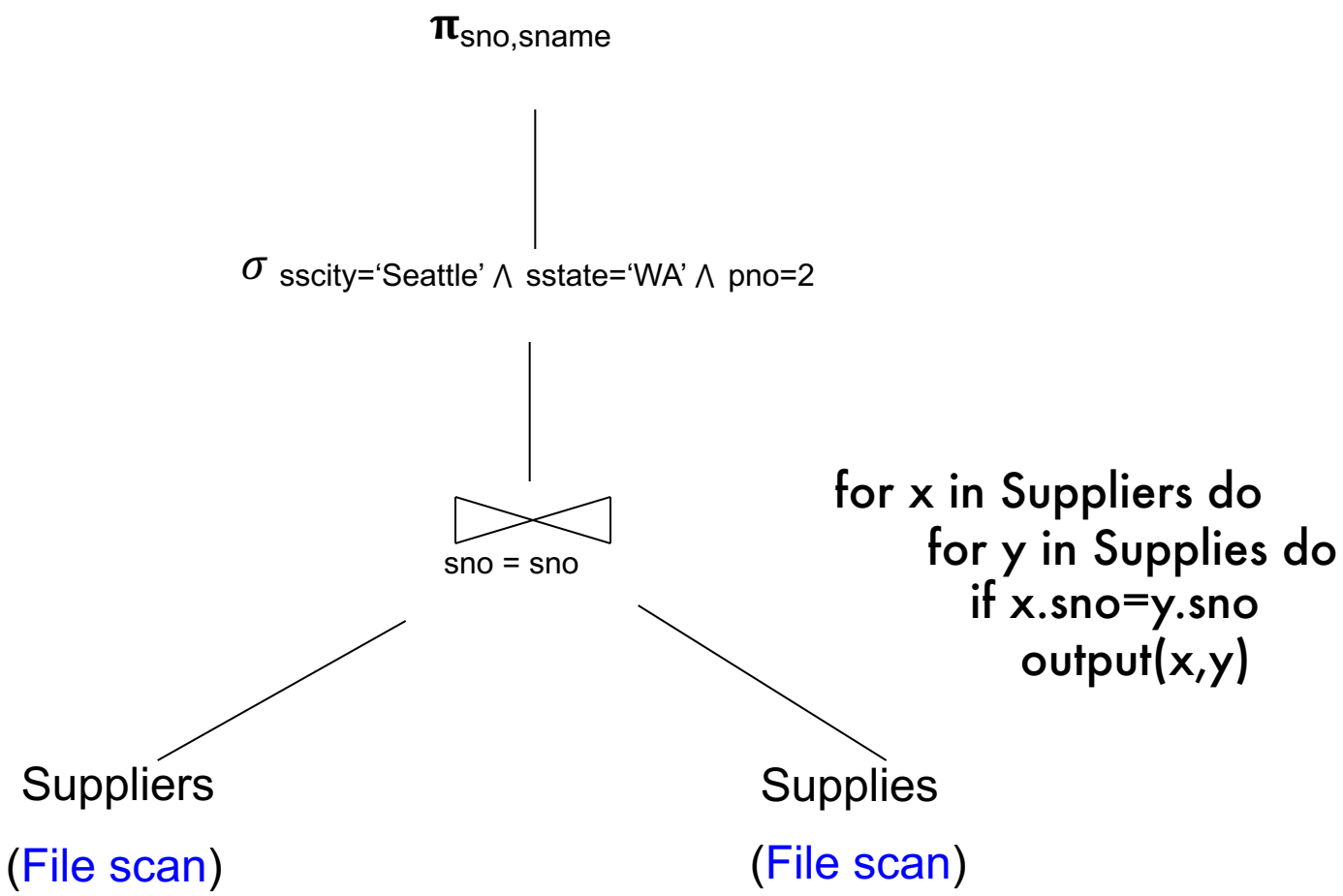
Physical Query Plan

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```



Physical Query Plan

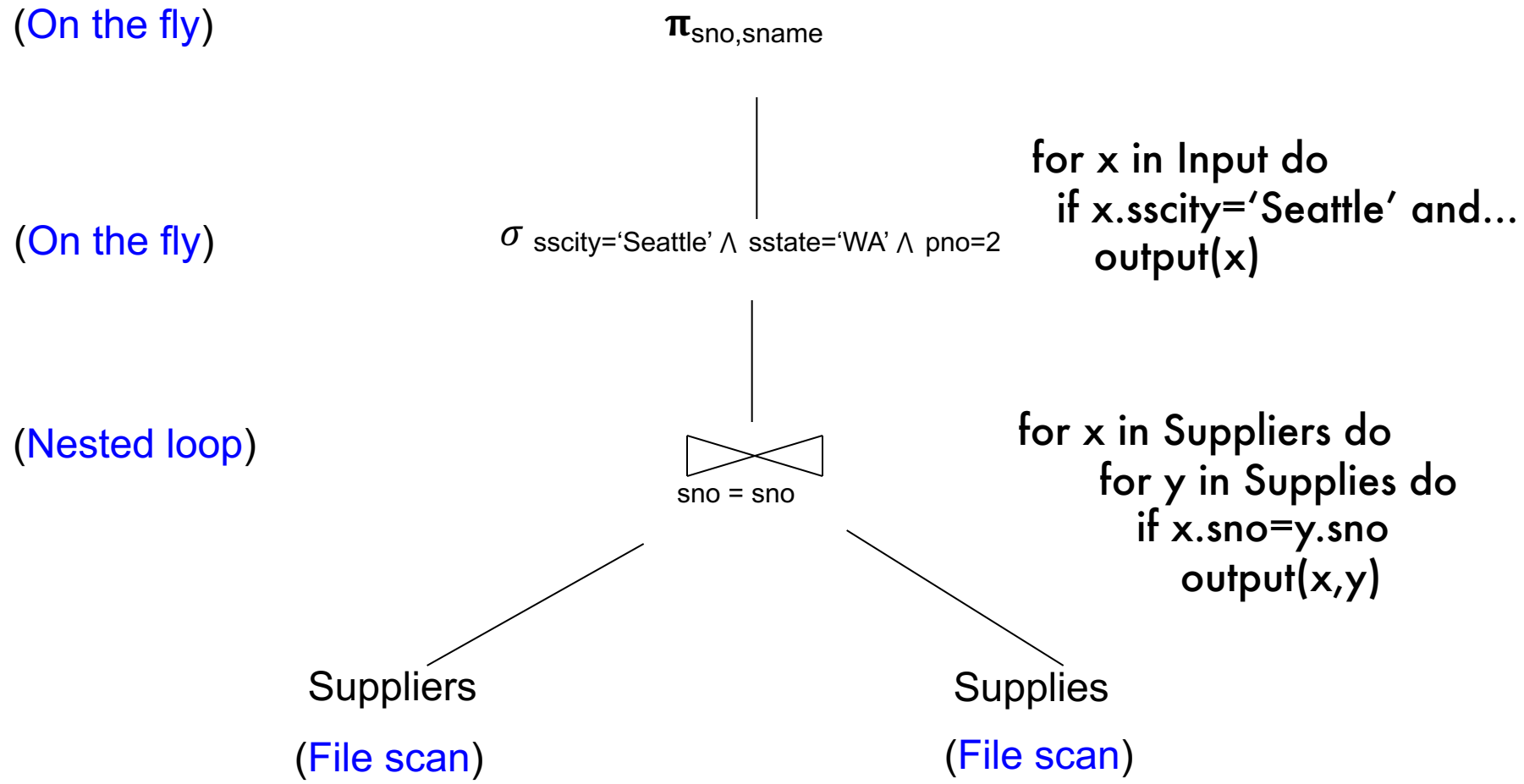
```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```




```
for x in Suppliers do
    for y in Supplies do
        if x.sno=y.sno
            output(x,y)
```

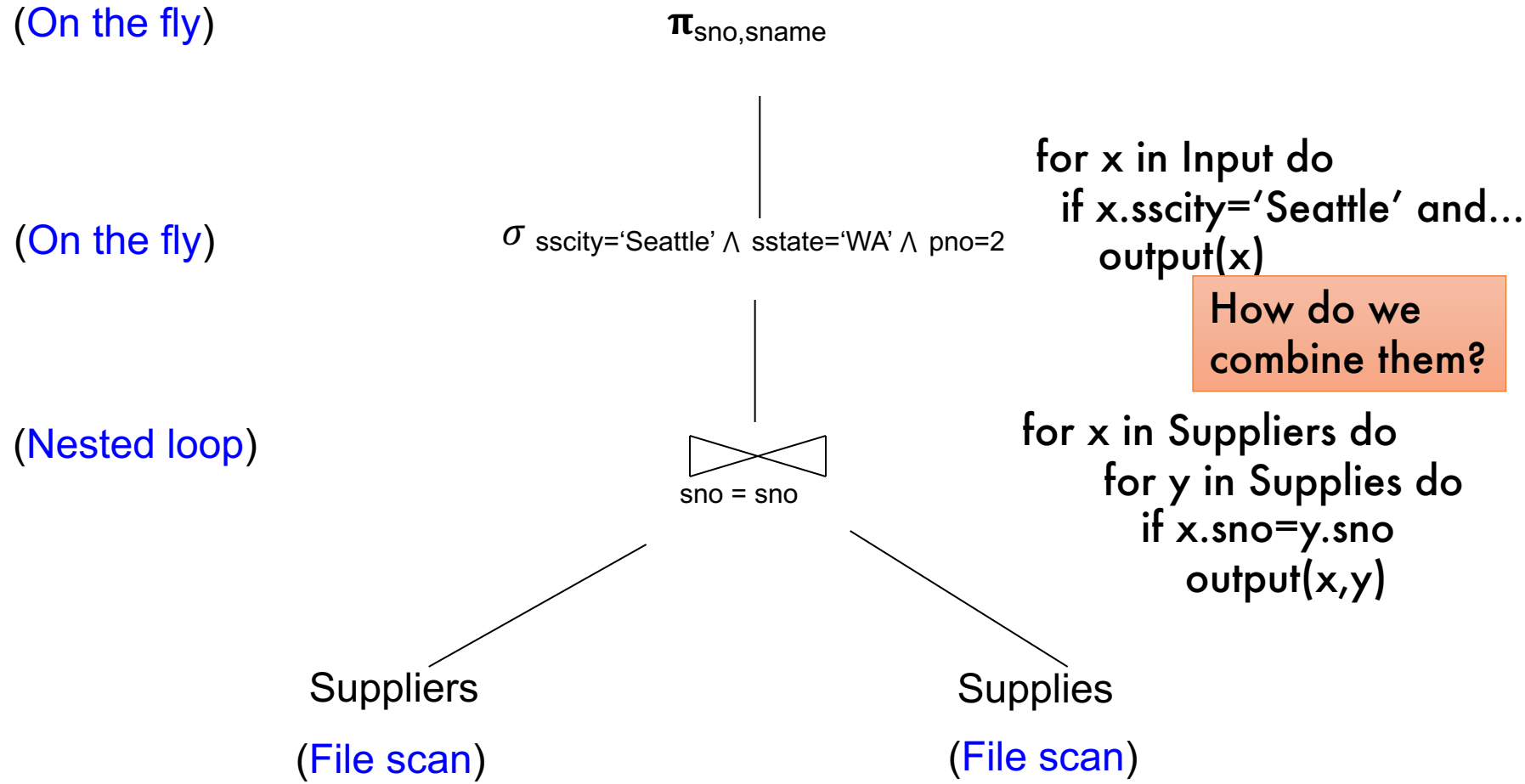
Physical Query Plan

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```



Physical Query Plan

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```



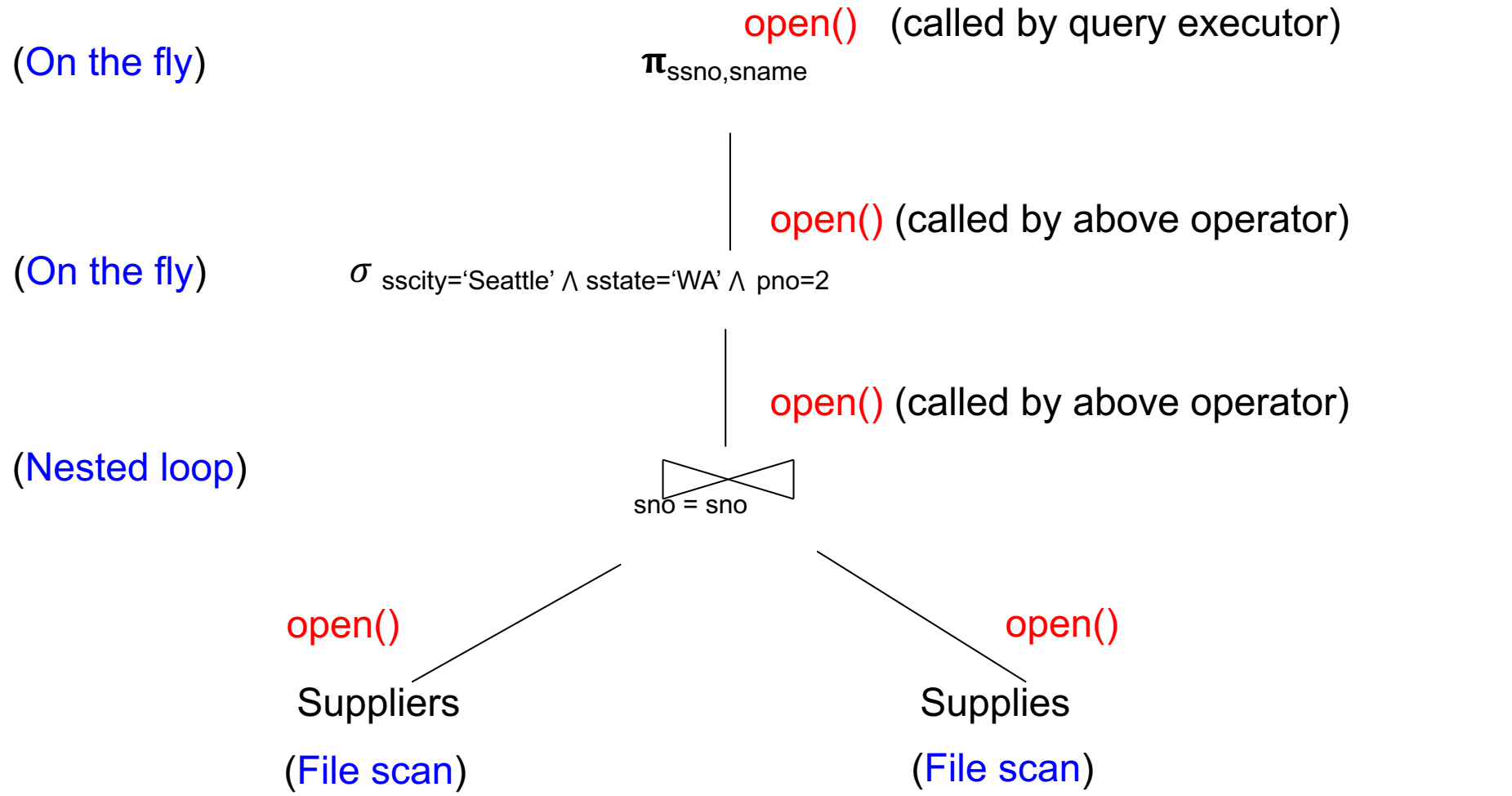
Query Executor

Iterator Interface

- Each **operator implements OpIterator.java**
- **open()**
  - Initializes operator state
  - Sets parameters such as selection predicate
- **next()**
  - **Returns a Tuple!**
  - Operator invokes next() recursively on its inputs
  - Performs processing and produces an output tuple
- **close()**: clean-up state
- Operators also have reference to their **child** operator in the query plan

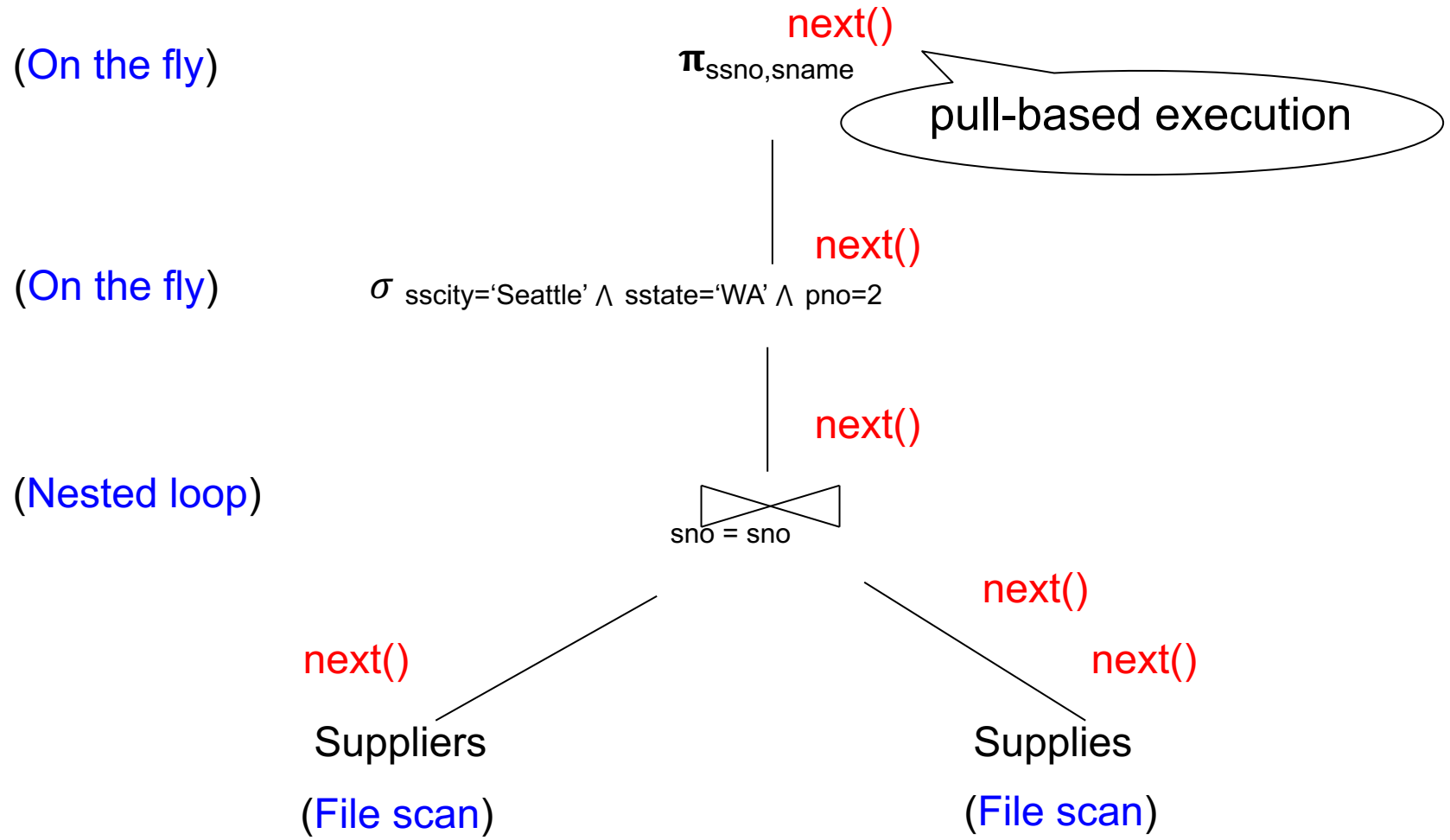
Query Execution

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```



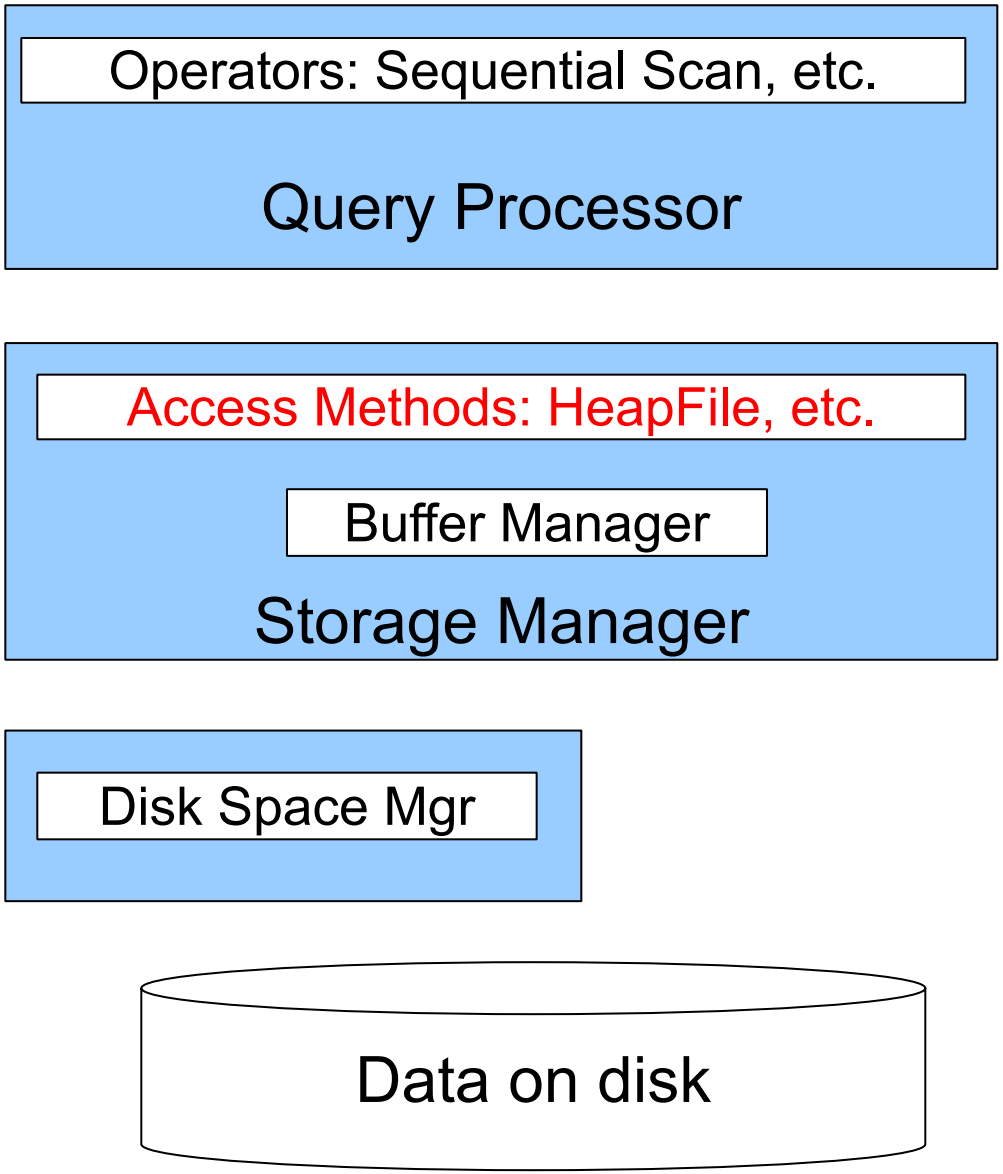
Query Execution

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supplies(sno,pno,price)
```



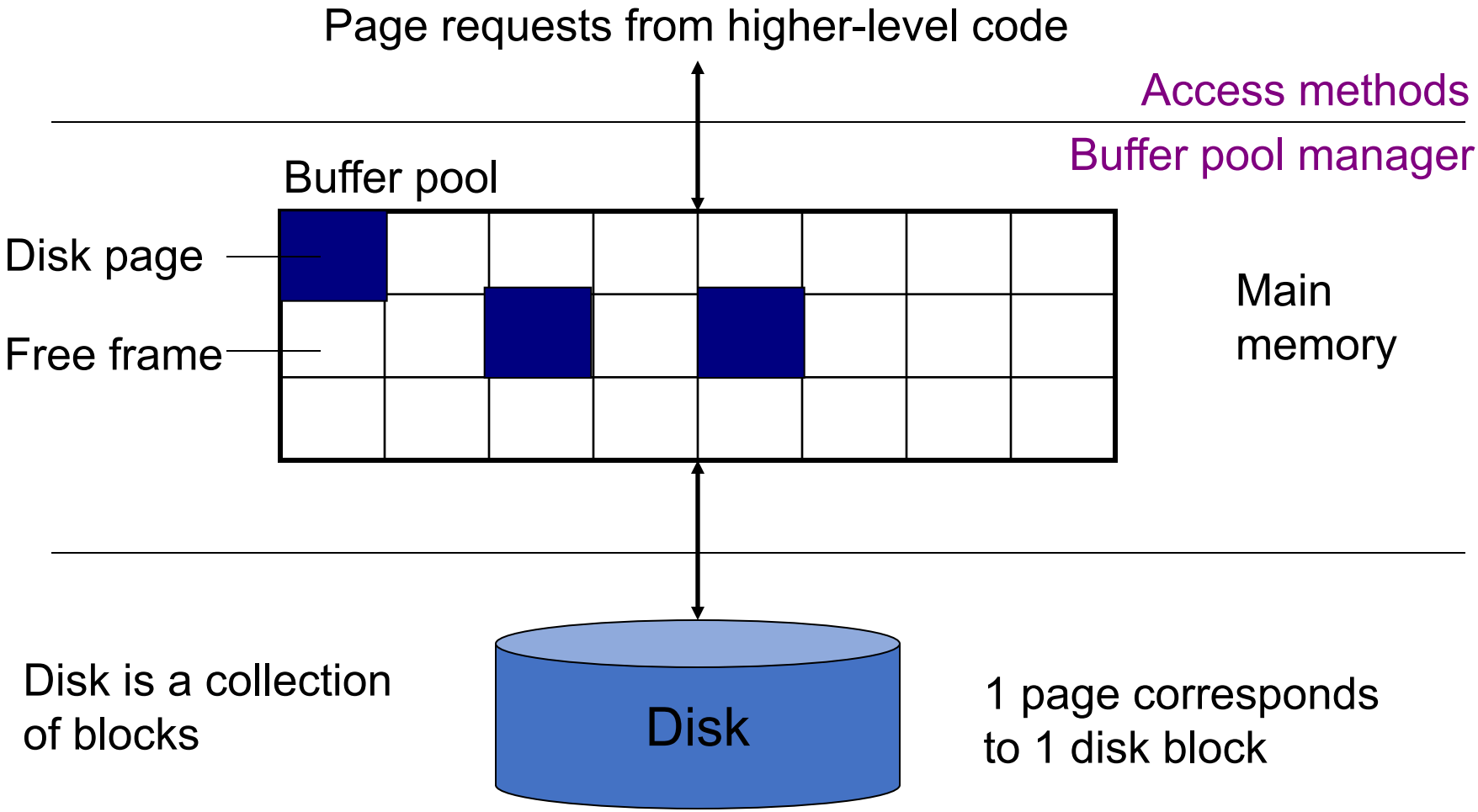
Storage Manager

Access Methods



- **Operators:** Process data
- **Access methods**: Organize data to support fast access to desired subsets of records
- **Buffer manager**: Caches data in memory. Reads/writes data to/from disk as needed
- **Disk-space manager**: Allocates space on disk for files/access methods

Buffer Manager (BufferPool in SimpleDB)



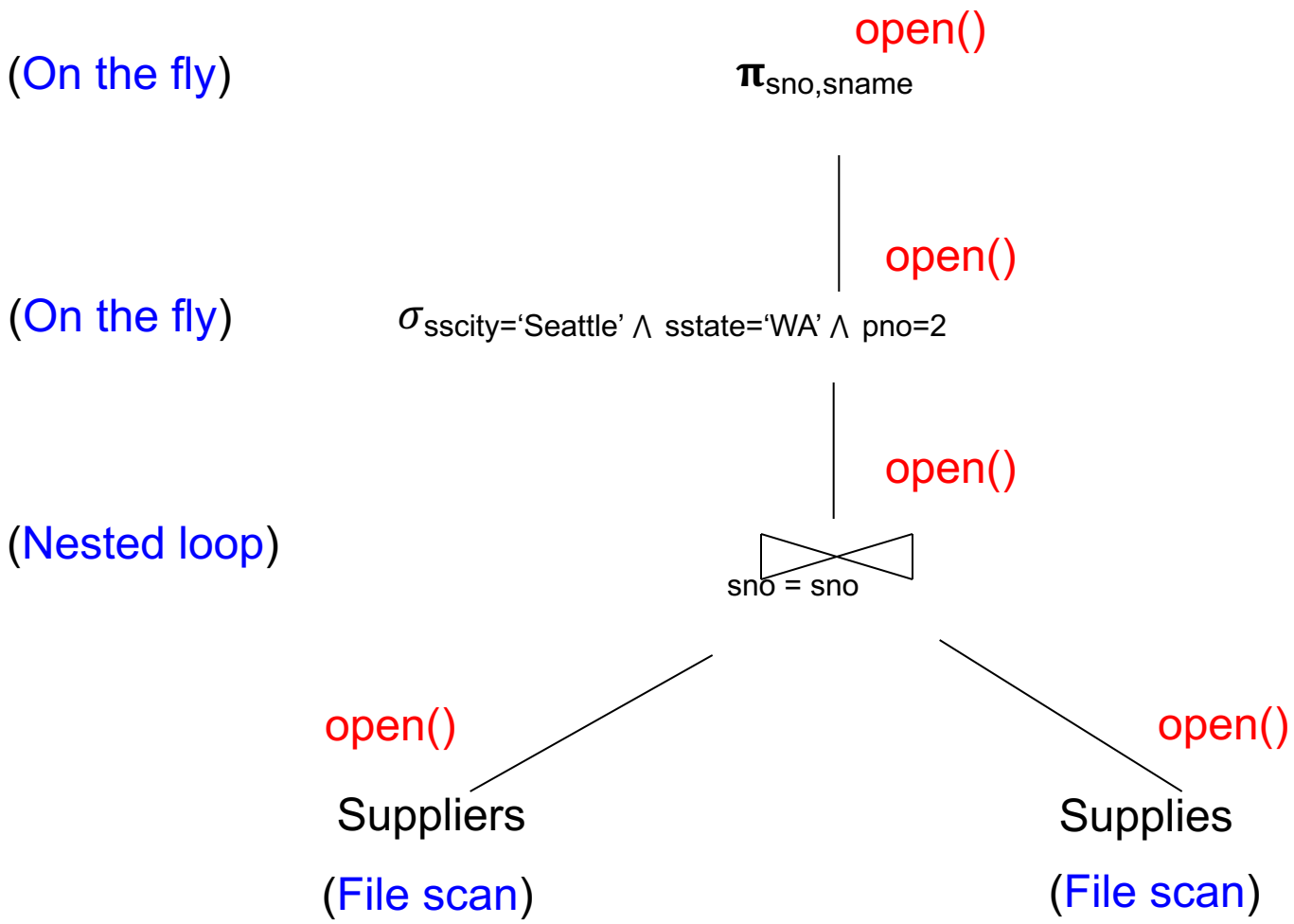
Buffer Manager

- Brings pages in from memory and caches them
- Eviction policies
  - Random page (ok for SimpleDB)
  - Least-recently used
  - The "clock" algorithm (see book)
- Keeps track of which **pages are dirty**
  - A dirty page has changes not reflected on disk
  - Implementation: Each page includes a dirty bit

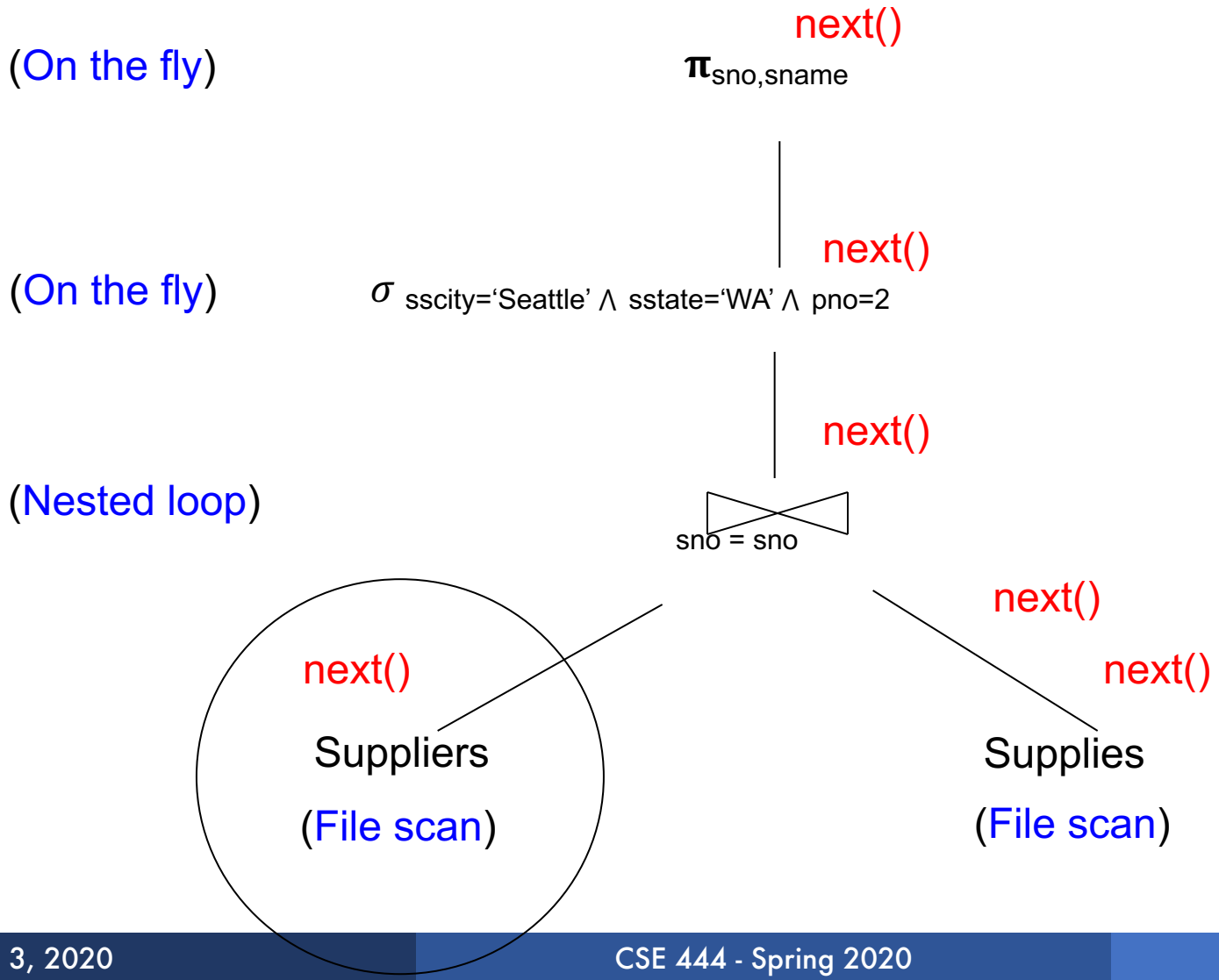
Access Methods

- A DBMS stores data on disk by breaking it into *pages*
  - A page is the size of a disk block.
  - A page is the unit of disk IO
- Buffer manager caches these pages in memory
- Access methods do the following:
  - They organize pages into collections called DB *files*
  - They organize data inside pages
  - They provide an API for operators to access data in these files
- Discussion:
  - OS vs DBMS files
  - OS vs DBMS buffer manager

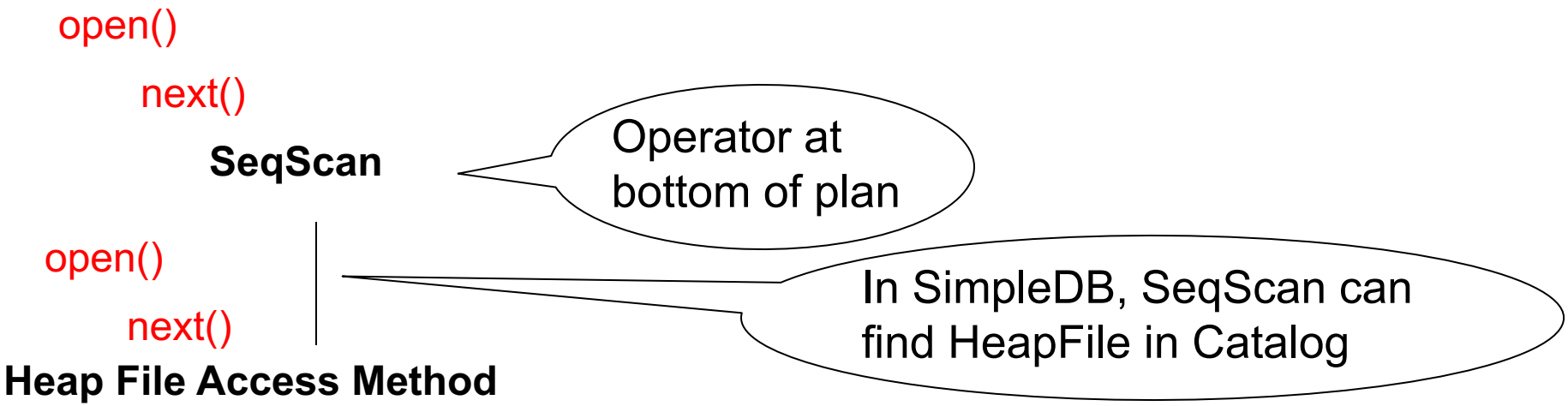
Query Execution



Query Execution



Query Execution In SimpleDB



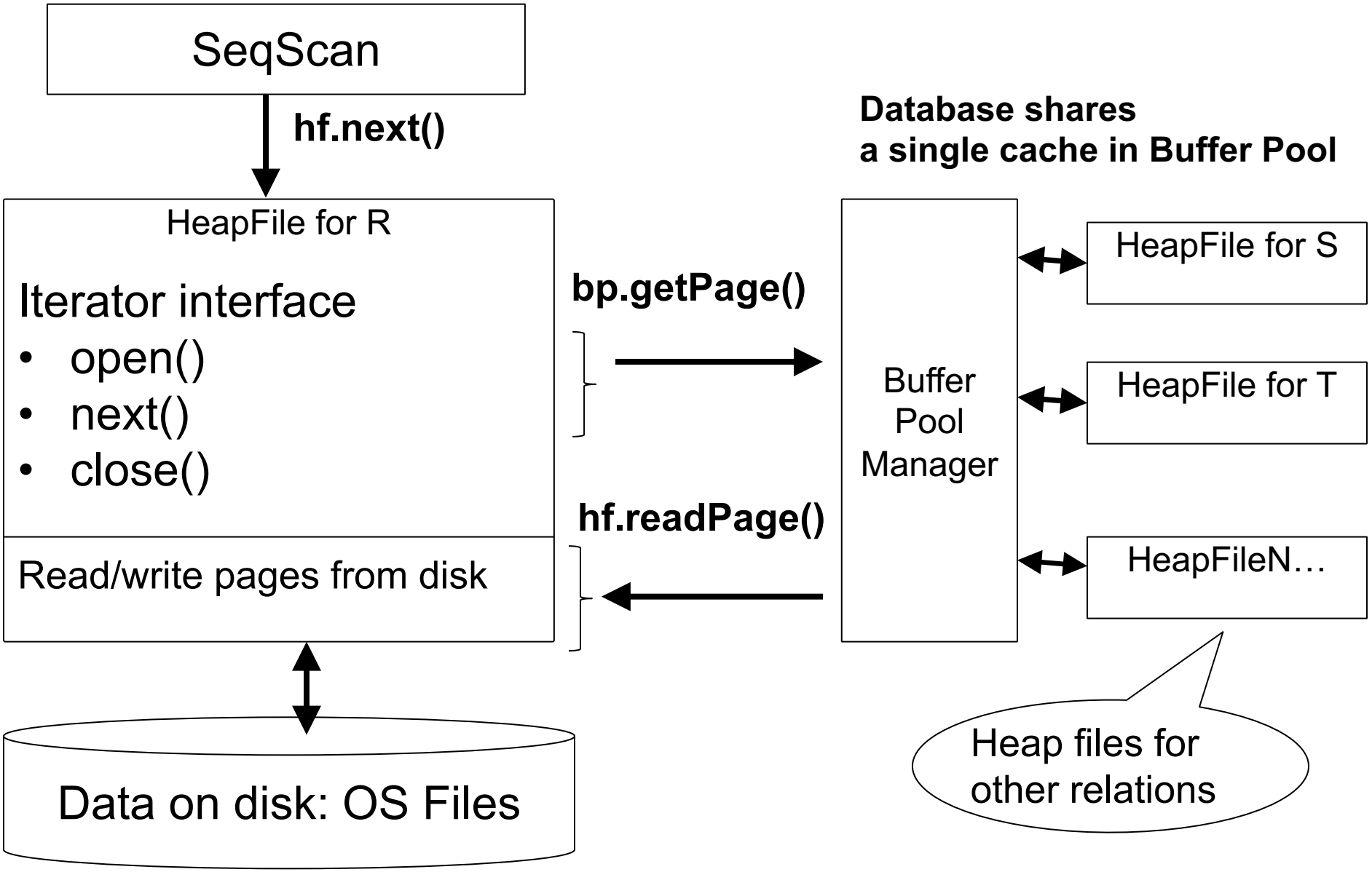
Offers iterator interface

- open()
- next()
- close()

Knows how to read/write pages from disk

But if Heap File reads data directly from disk, it will not stay cached in Buffer Pool!

Query Execution In SimpleDB



HeapFile In SimpleDB

- Data is stored on disk in an OS file. HeapFile class knows how to "decode" its content
- Control flow:

  SeqScan calls methods such as "iterate" on the HeapFile Access Method

  During the iteration, the HeapFile object needs to call the BufferManager.getPage() method to ensure that necessary pages get loaded into memory.

  The BufferManager will then call HeapFile .readPage()/writePage() page to actually read/write the page.