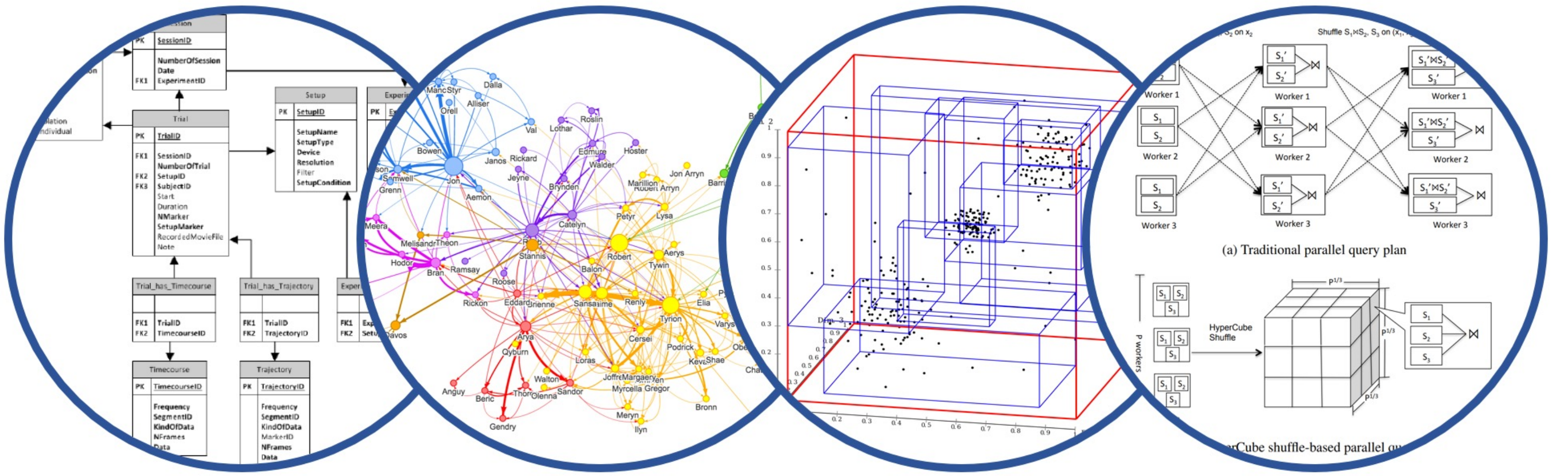


Database System Internals

Architecture

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

Important Note

- Lectures show principles
- Homeworks + Quizzes test the principles

- In SimpleDB, you often get to decide how to implement, there might be multiple ways to do something and you get to make a design decision.
  - It’s okay implement the simplest solutions!

- If you are confused, let us know!

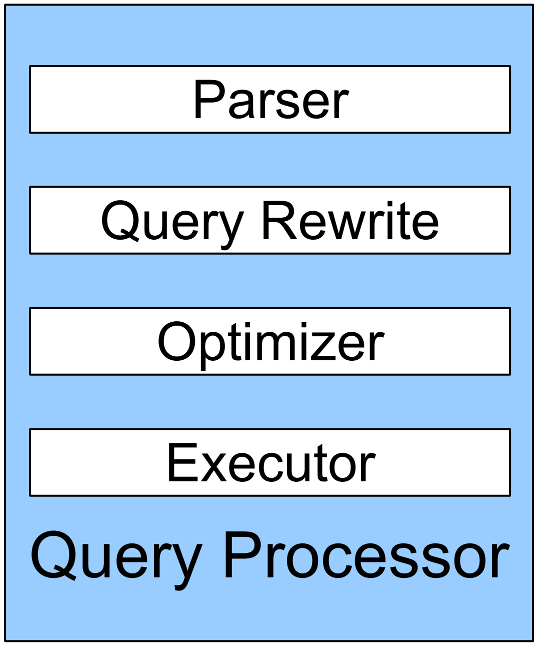
- SimpleDB not designed to be bullet-proof software

What we already know…

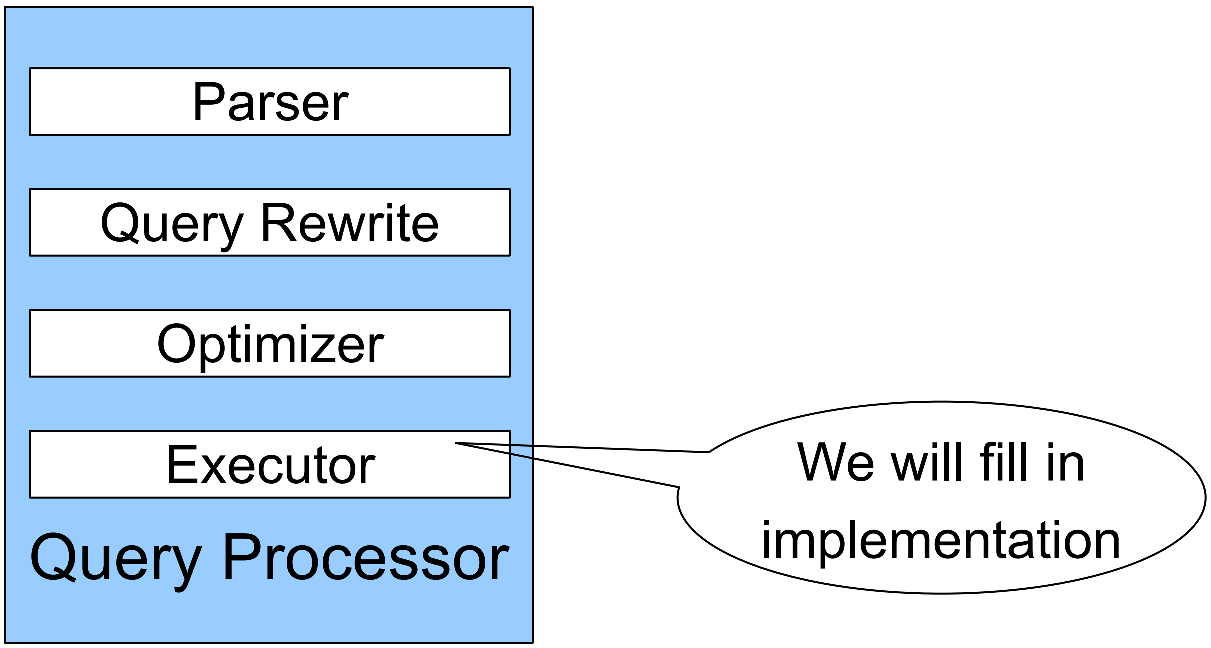
Relational Query Language:

- 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*

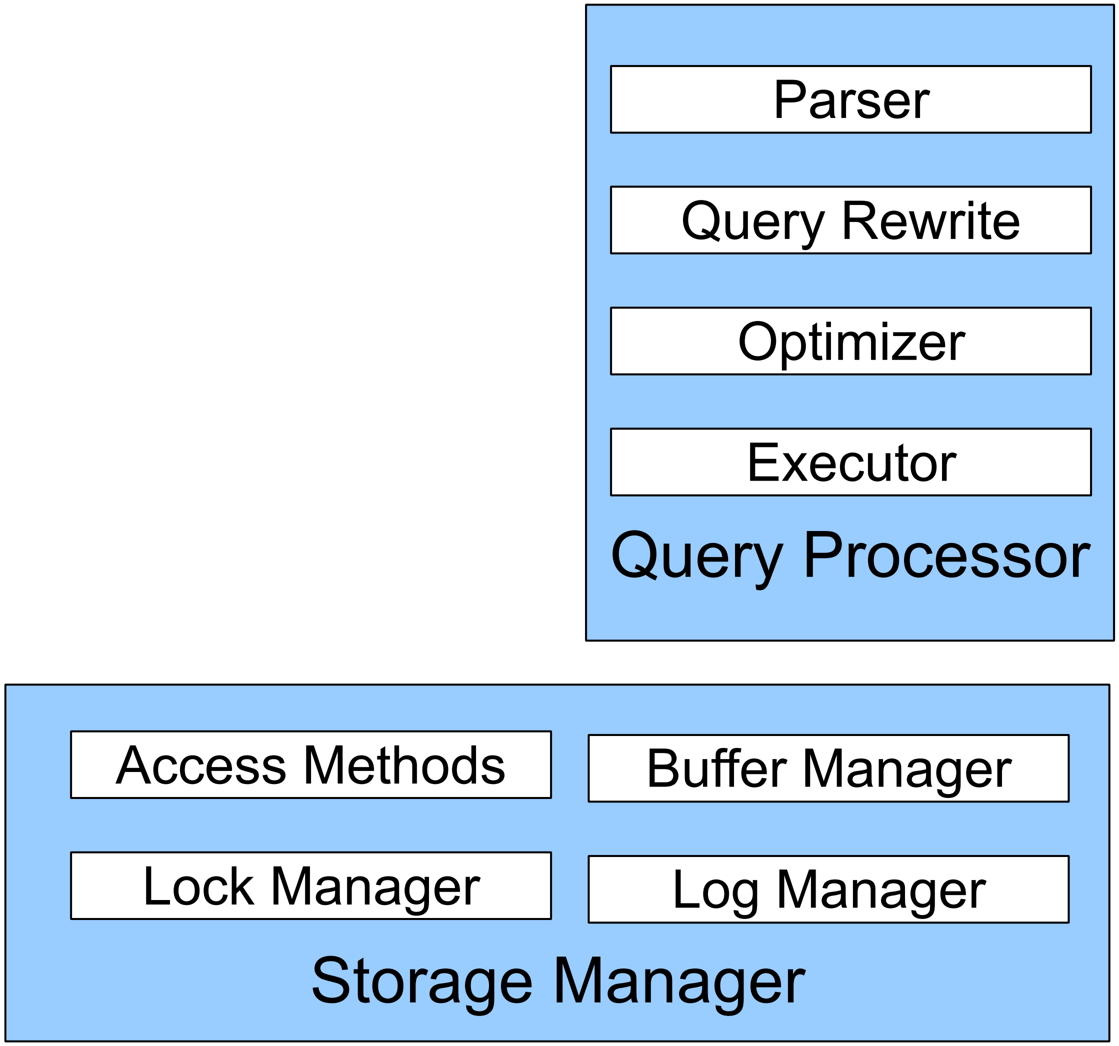
DBMS Architecture



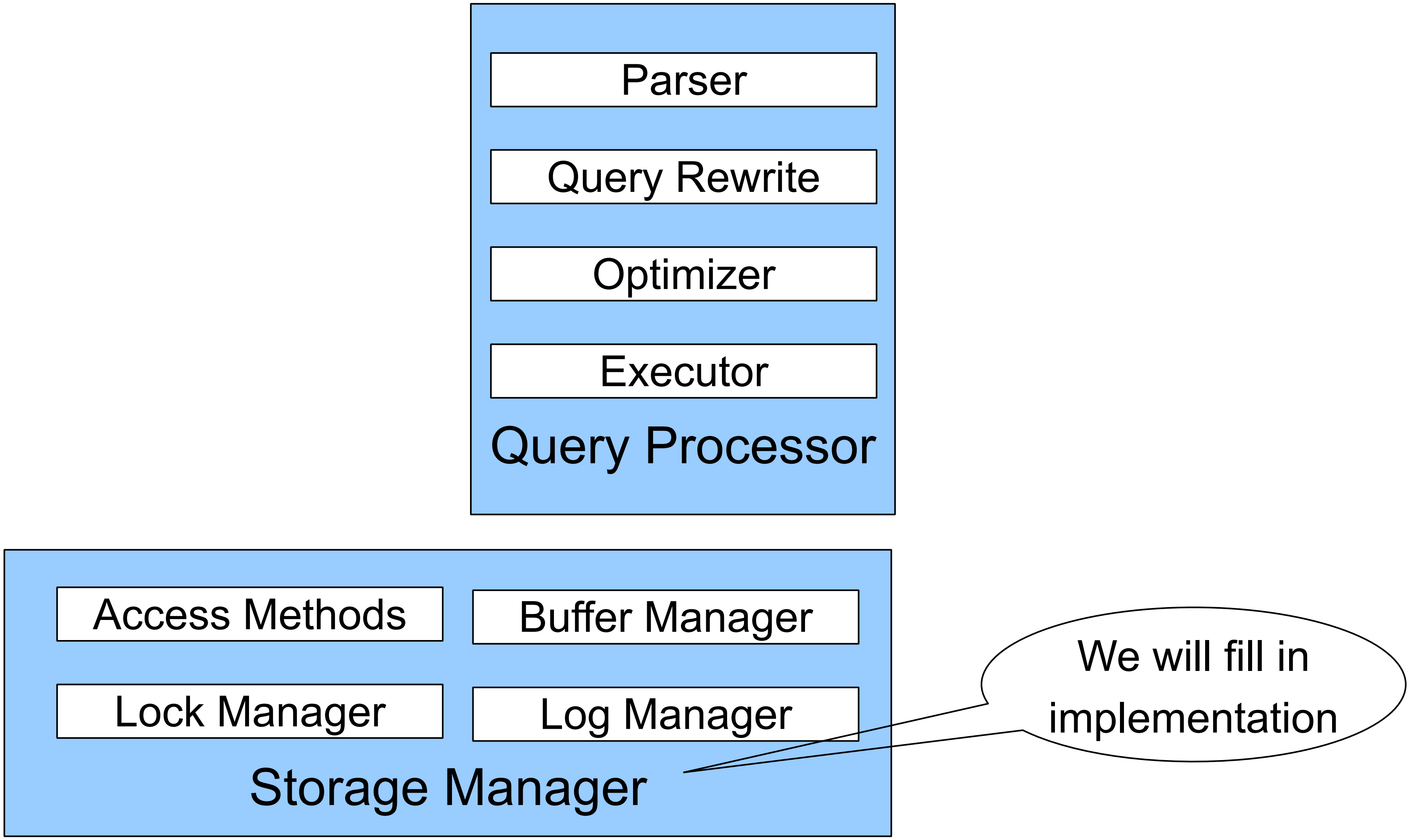
DBMS Architecture



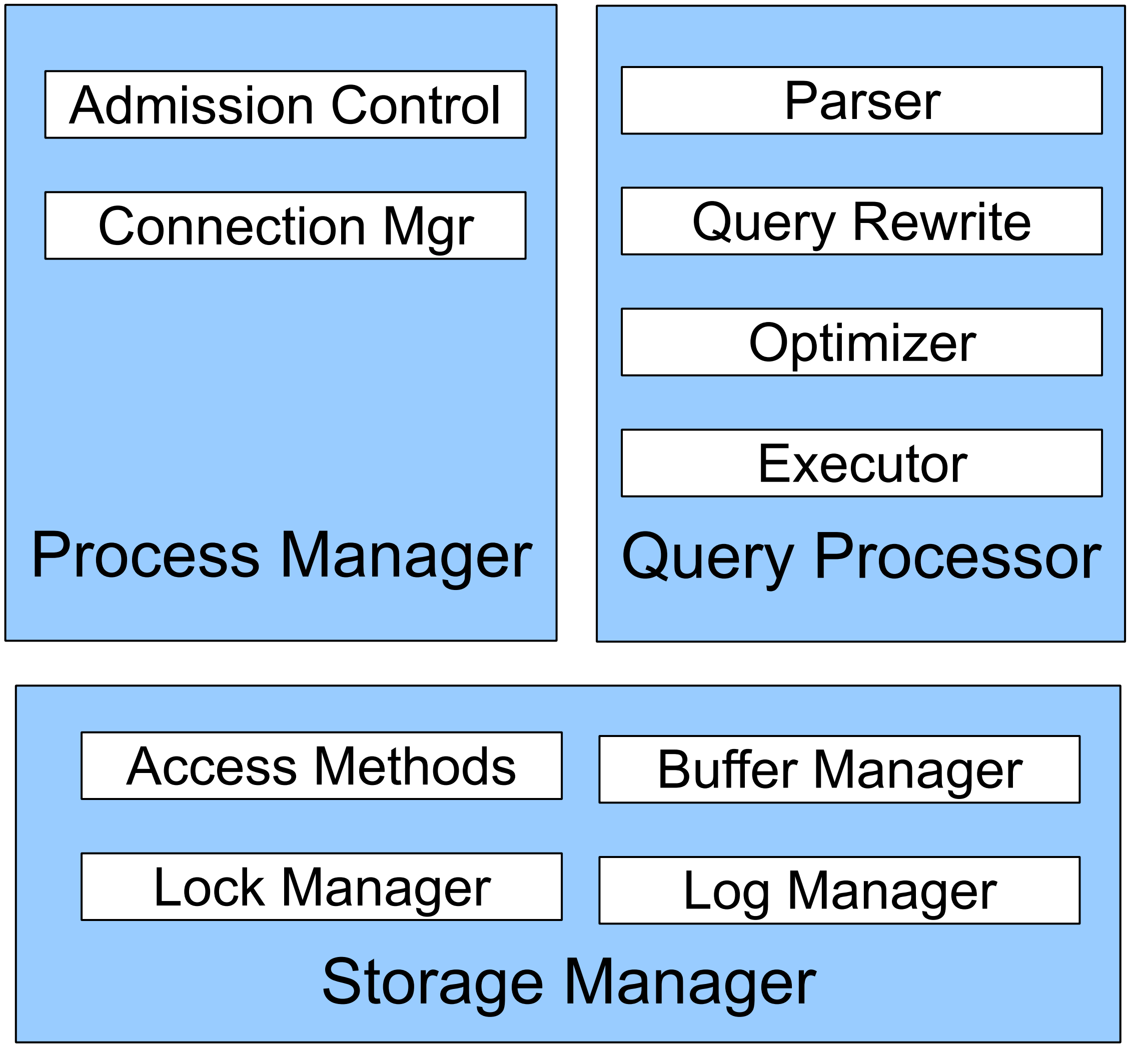
DBMS Architecture



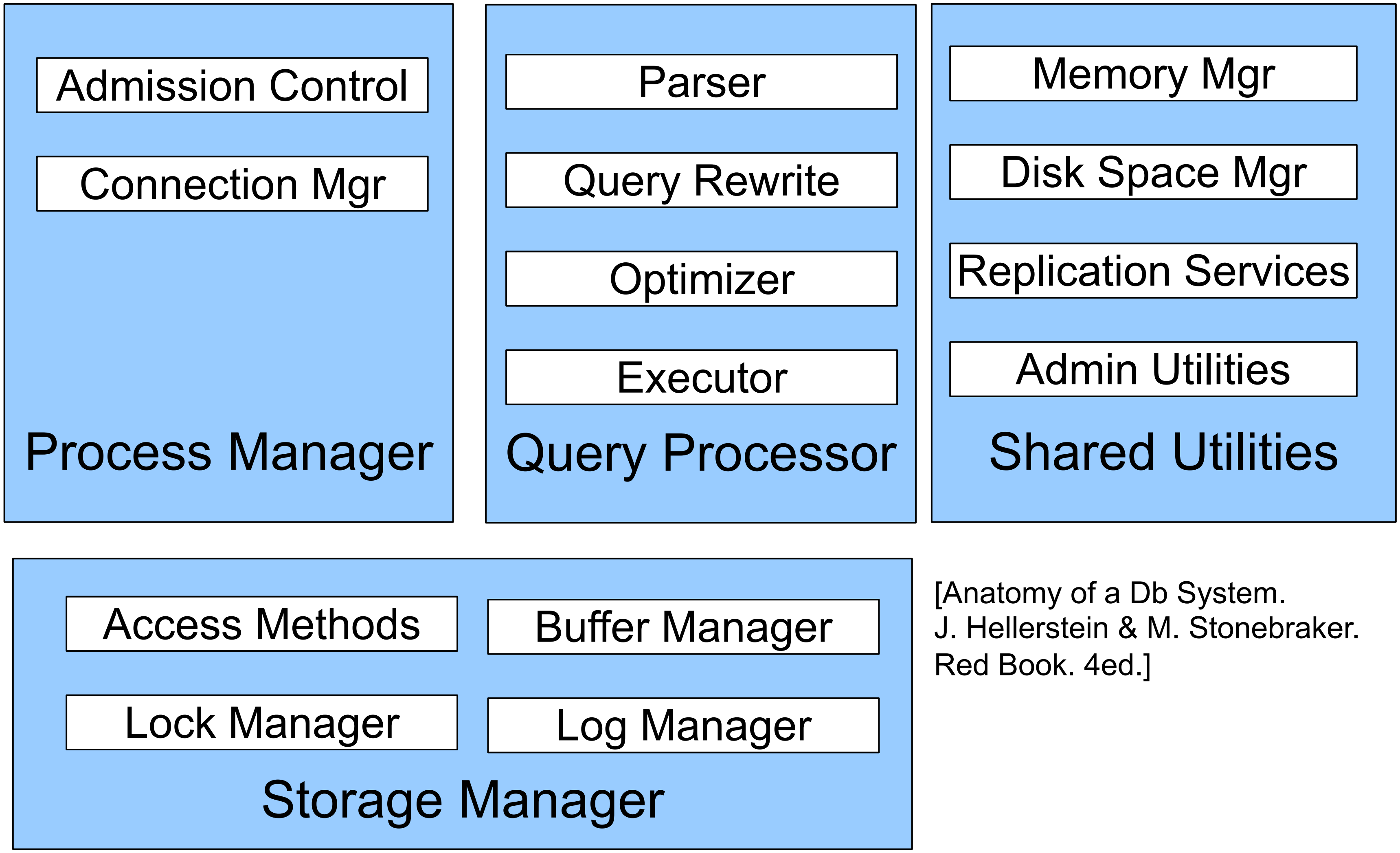
DBMS Architecture



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 = Derived Table: 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 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 sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Rewritten query (expanding NearbySupp view):

```
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;
```

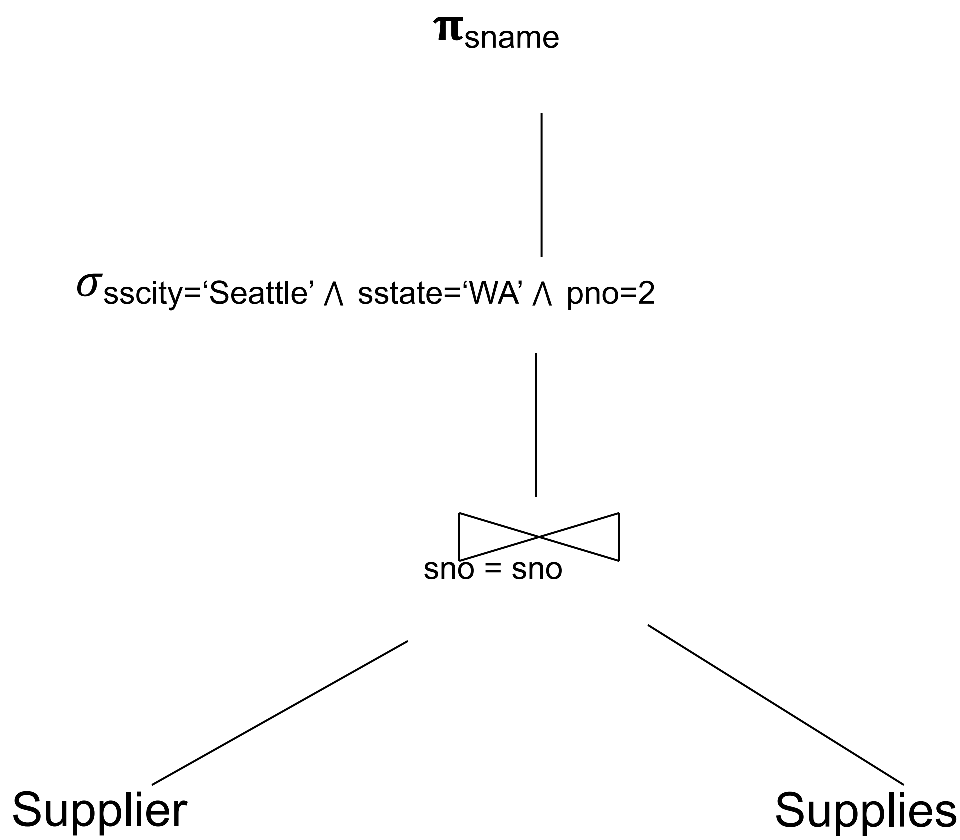
Query Processor

- **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)
```

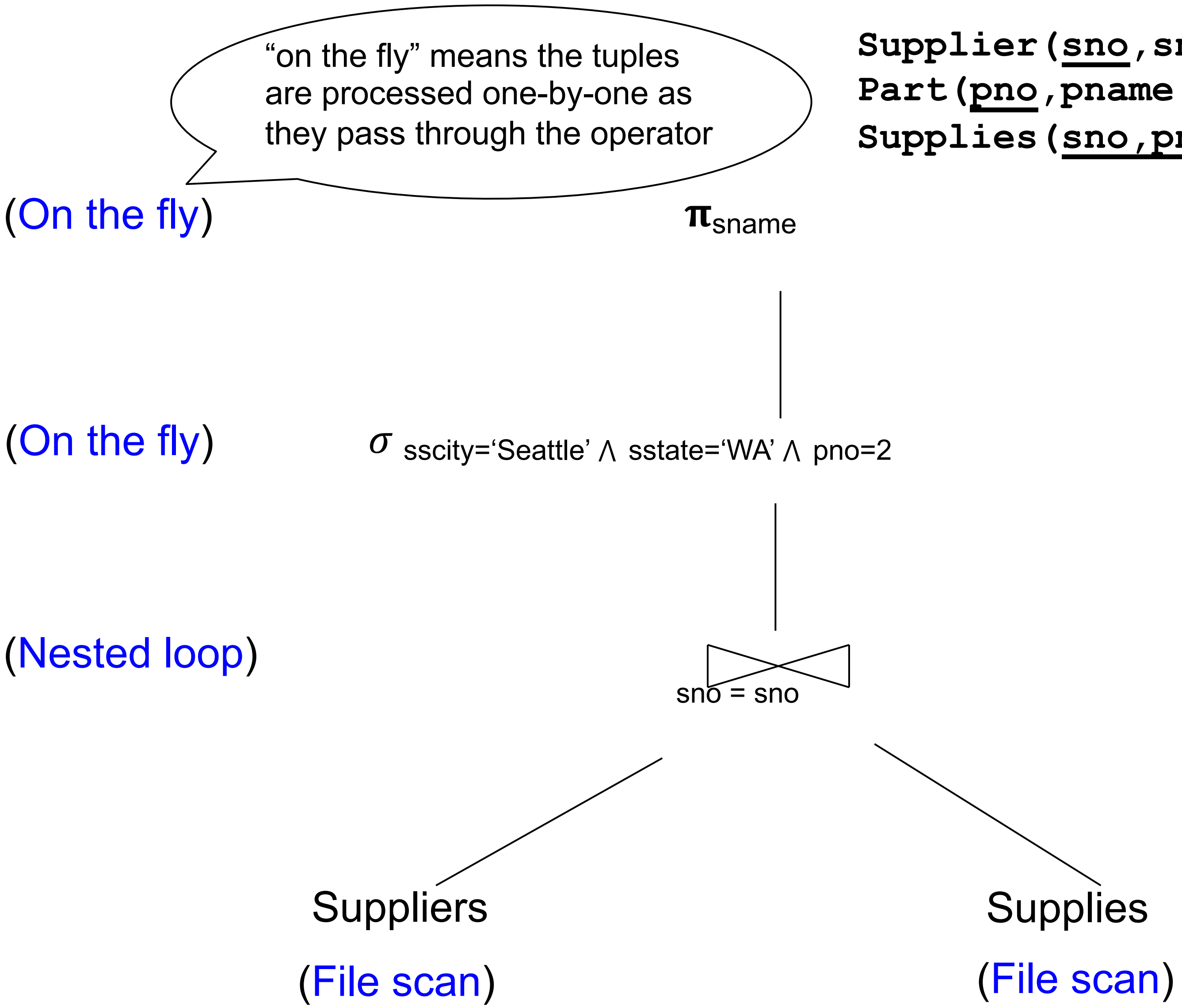


Physical Query Plan

- 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)



Query Executor

Tuples in SimpleDB

`Tuple.java` describes a row object in SimpleDB

- Rows are the objects passed through the database
- In the same way we conceptualize RA and a series of transformations to rows, so does it work in database

Push vs Pull based execution

- Pull: (1) “can I have a tuple?” (2) “here is a tuple”
- Push: (1) “here is a tuple”

- Many modern databases implement a push-based interface for operators
  - This is good for distributed systems since fewer network calls need to be made

- We implement pull-based operators in SimpleDB since it is simpler and all running in a single machine

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

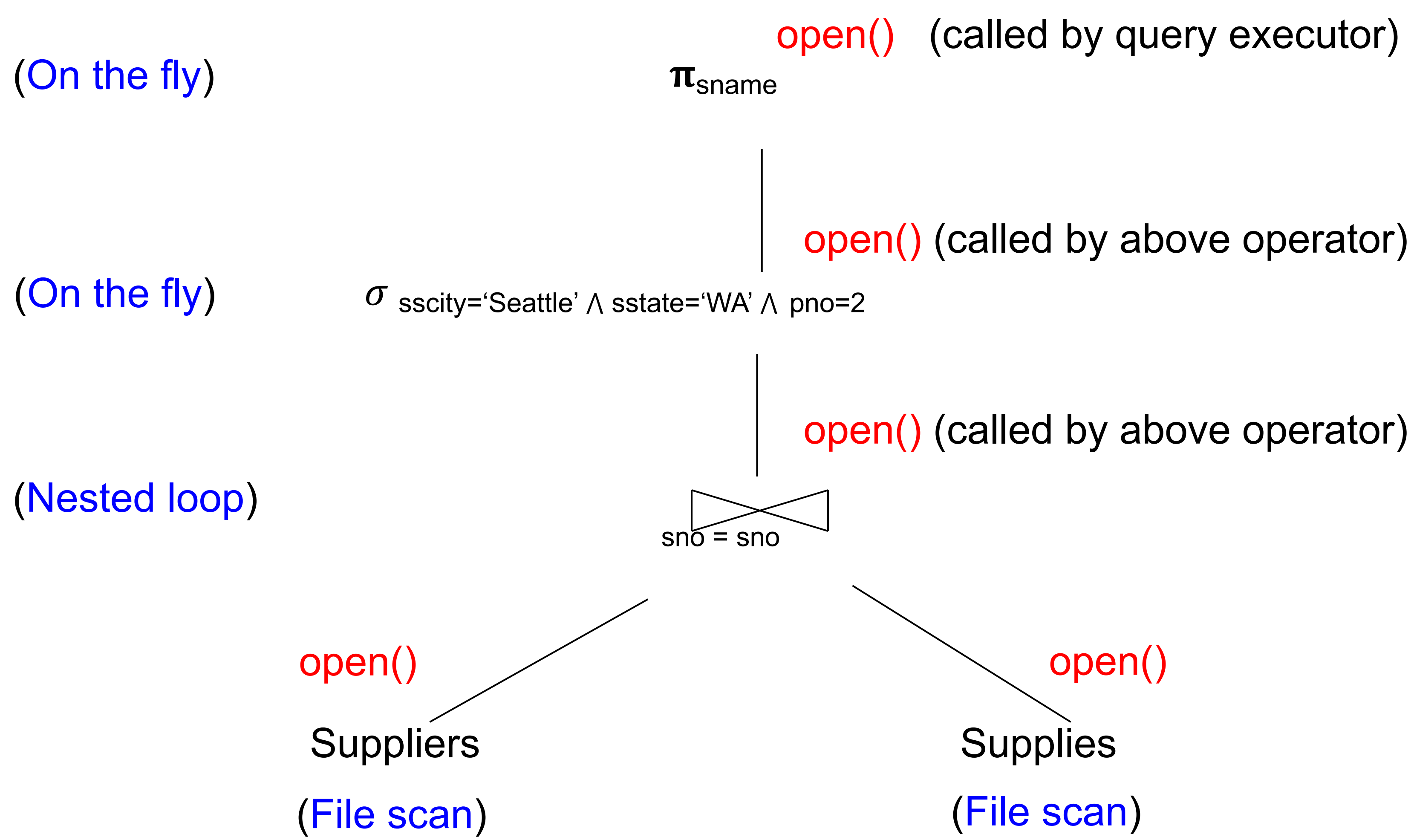
`close()` clean-up state

- Operators also have reference to their **child** operator in the query plan

  can call `child.open()` , `child.next()` etc..

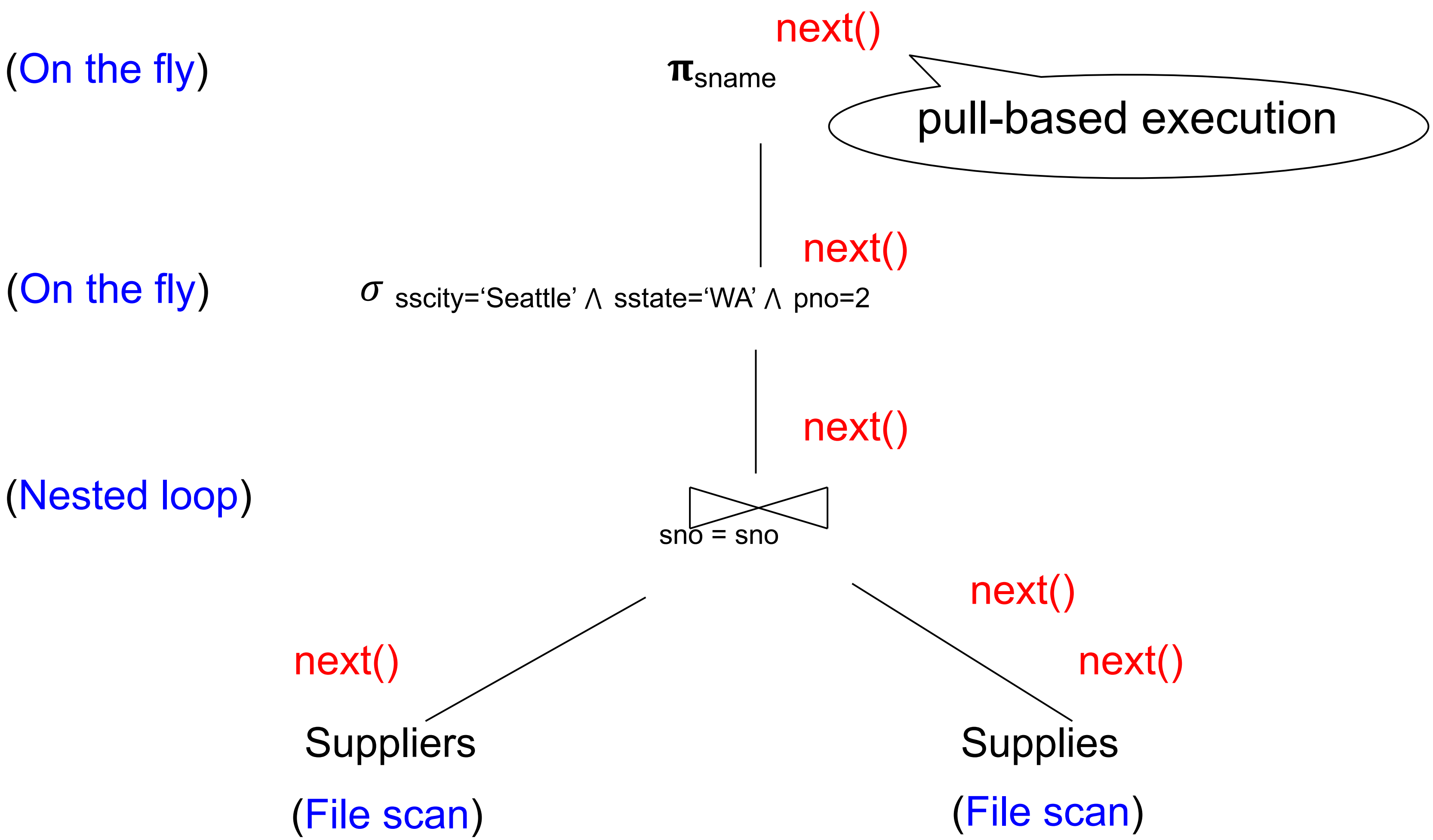
Query Execution

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



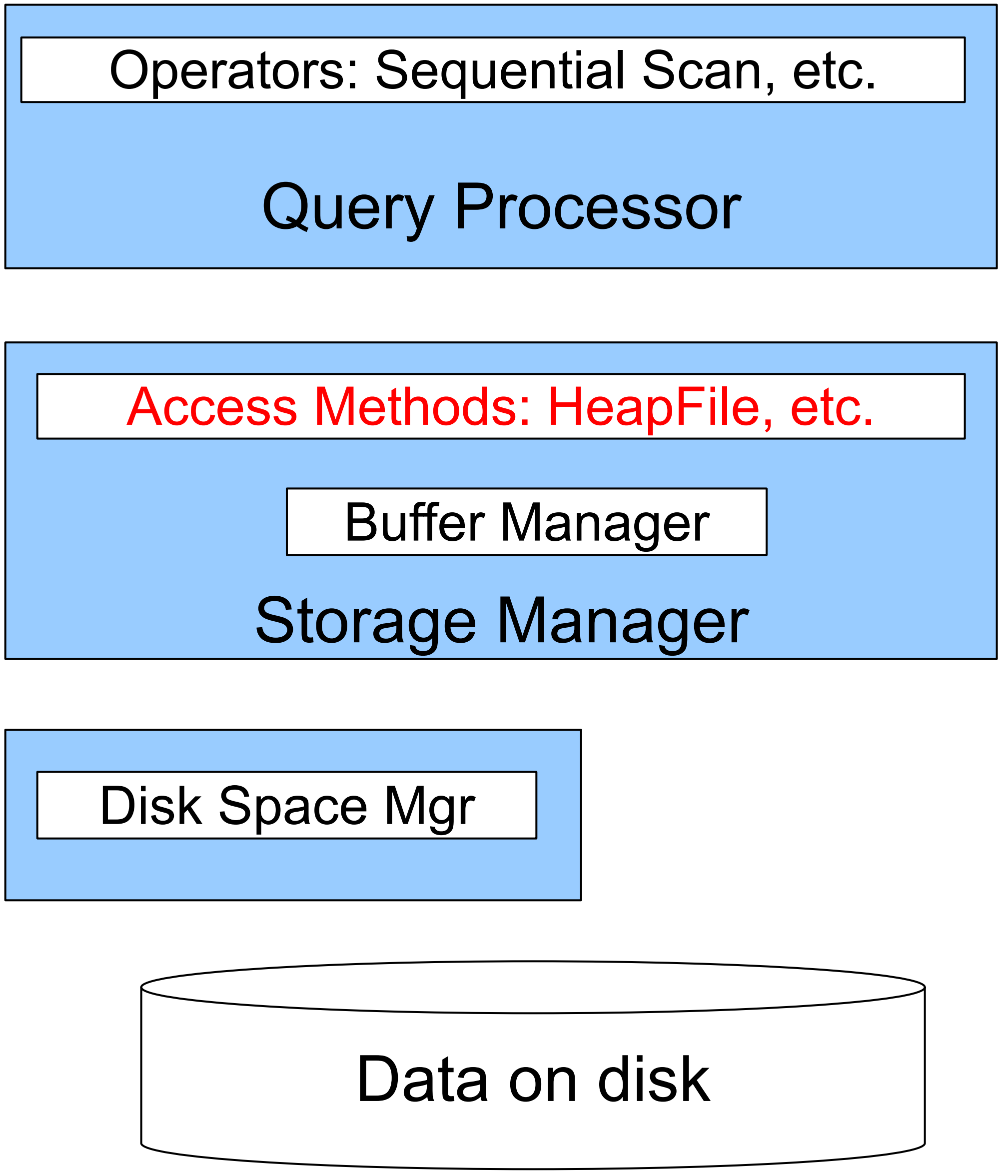
Query Execution

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



Storage Manager

Access Methods

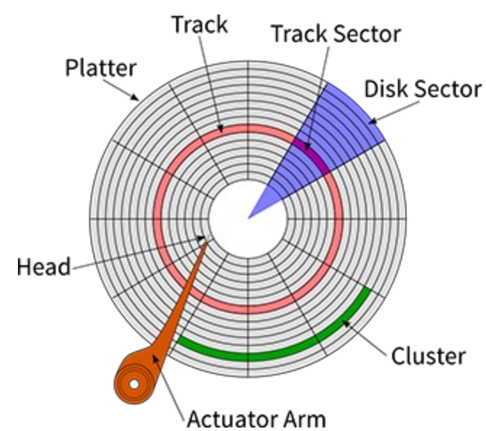


- **Operators:** Process data
- **Access methods**: Organize data to support fast access to desired subsets of records
- **Buffer manager**: Caches data in memory. Reads/writes data to/from disk as needed
- **Disk-space manager**: Allocates space on disk for files/access methods

Disk Storage

- Can only read **1 block per read operation**
  - Usually 512B to 4kB
- One blocks contains some Tuples
- [Cool Youtube visualization]
- **Sequential disk reads are faster than random ones**
  - Cost ~1-2% random scan = full sequential scan

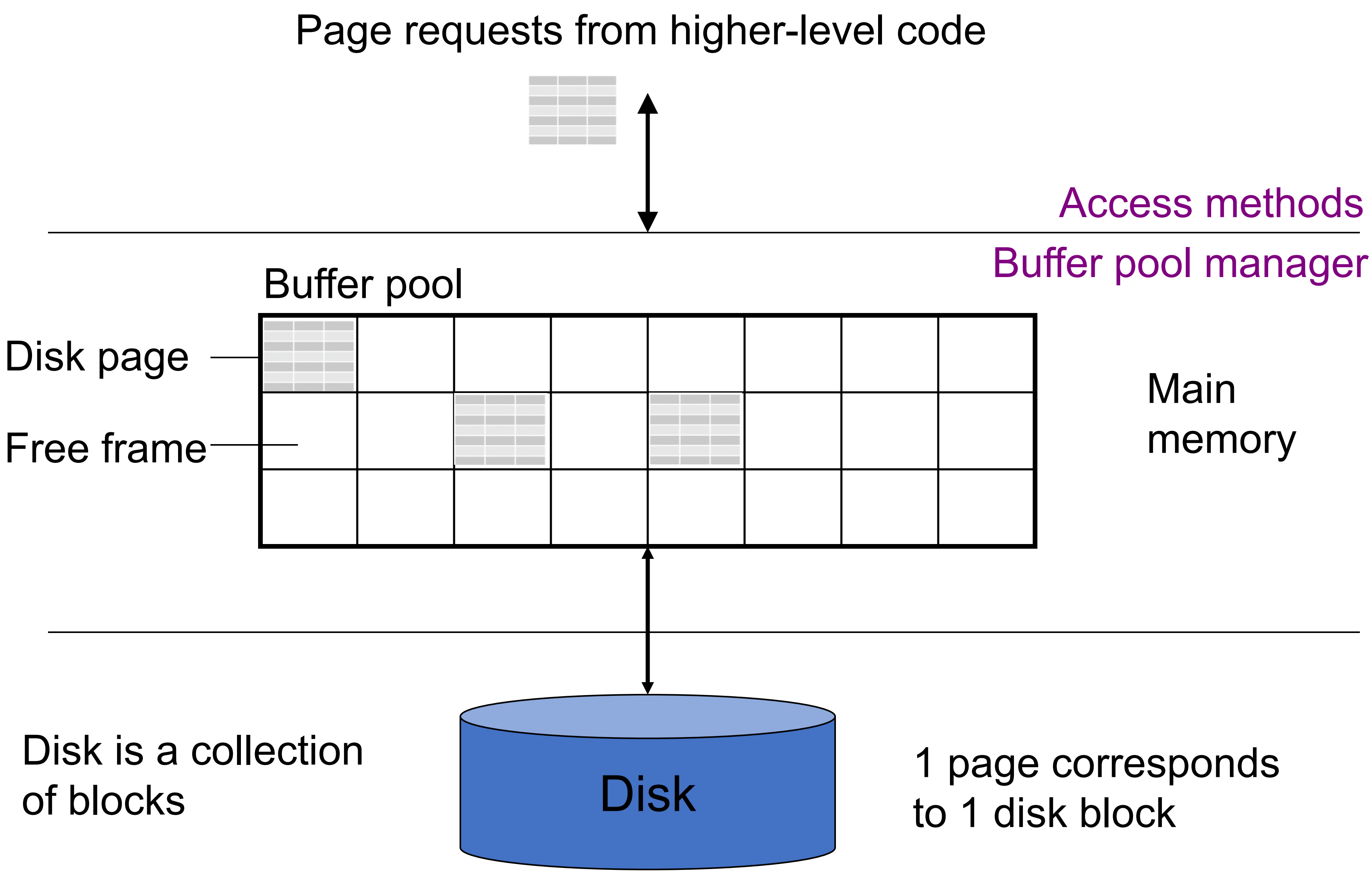
444	Ryan	G20
344	Ryan	134
544	Dan	134



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

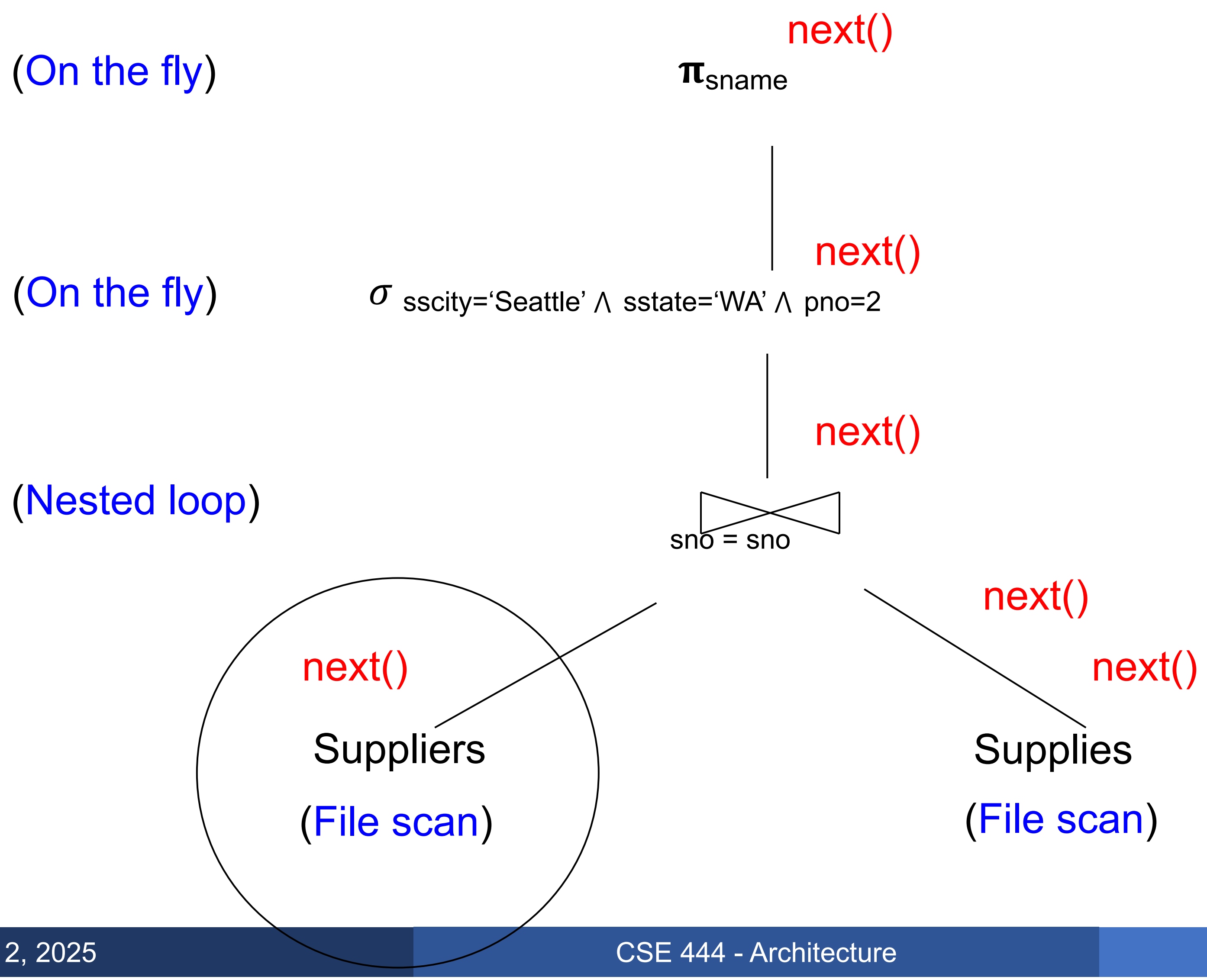
Buffer Manager (BufferPool in SimpleDB)



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

Query Execution



Query Execution In SimpleDB

open()

next()

SeqScan

Operator at bottom of plan

open()

next()

Heap File Access Method

In SimpleDB, SeqScan can find HeapFile in Catalog

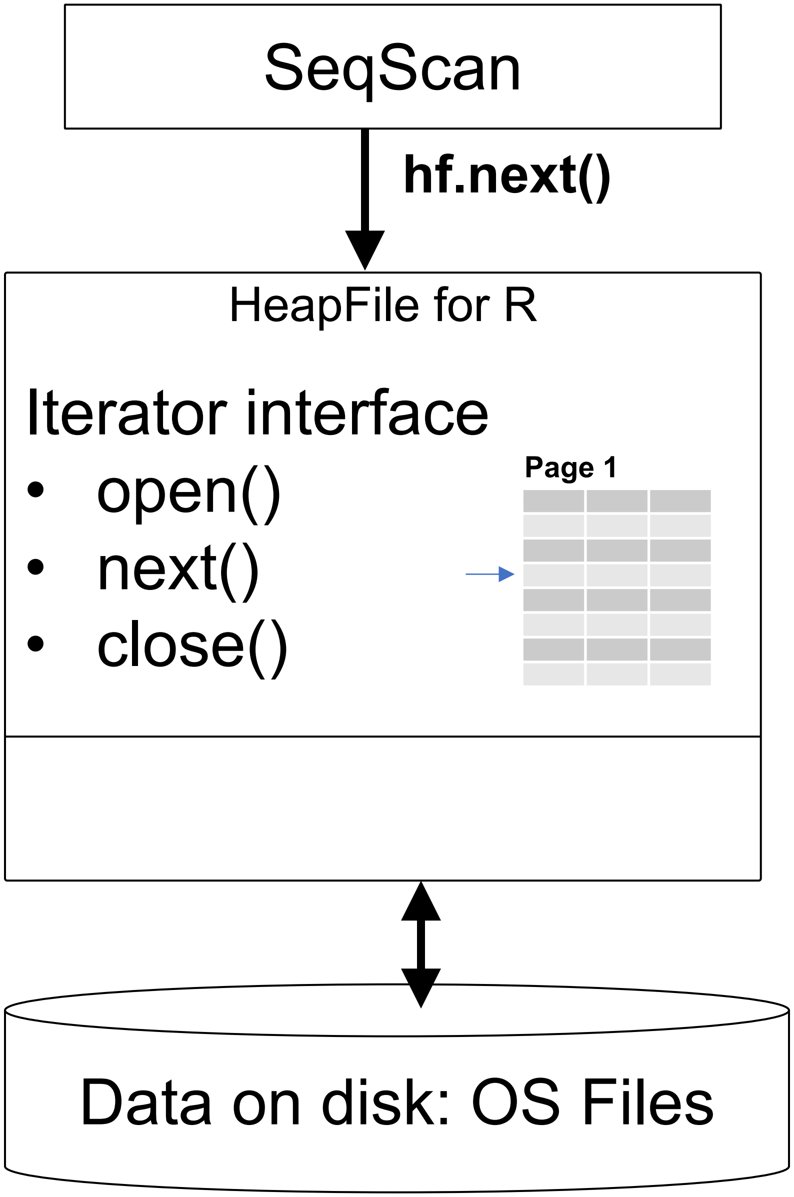
Offers iterator interface

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

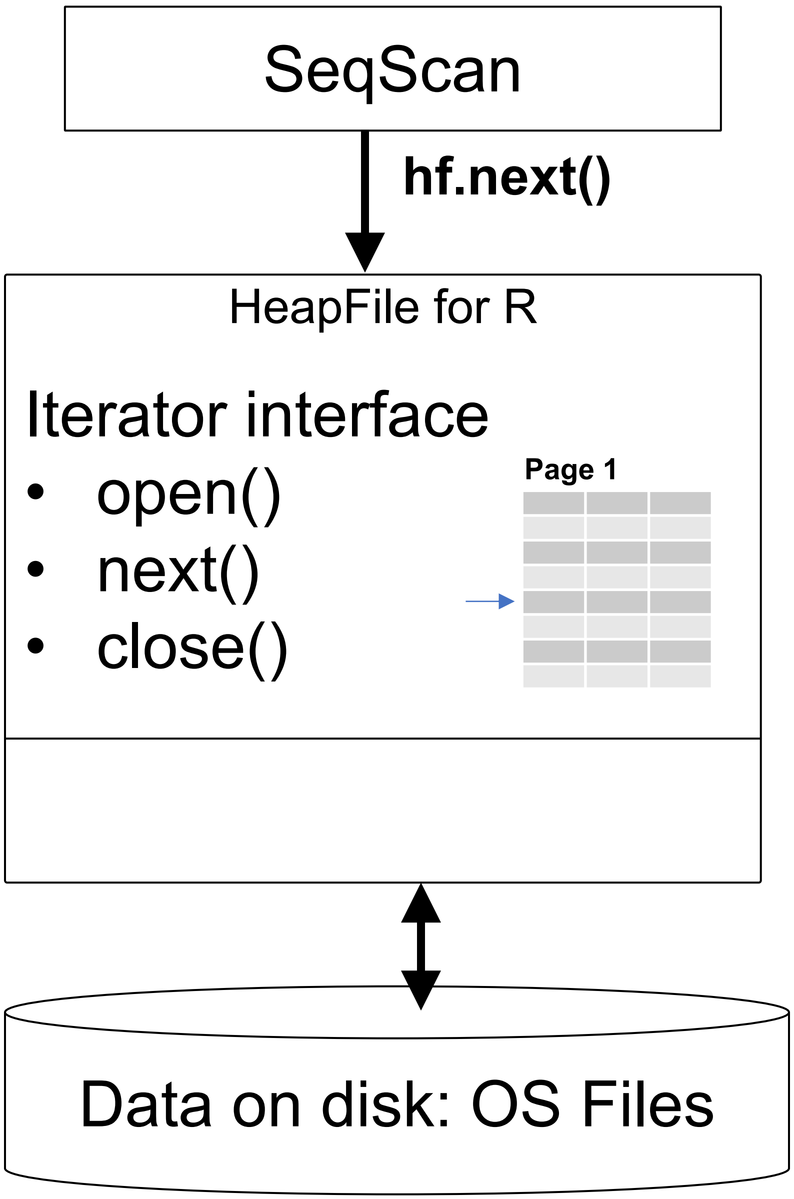
Knows how to read/write pages from disk

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

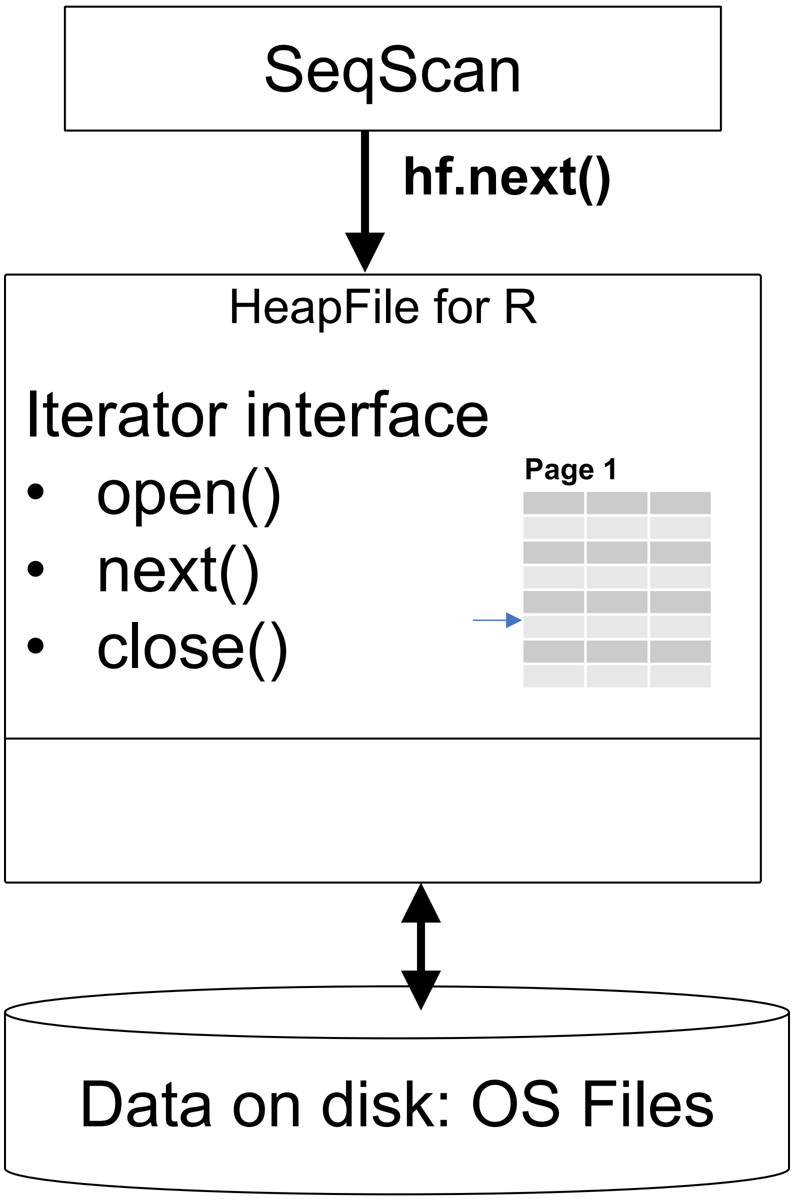
Query Execution In SimpleDB



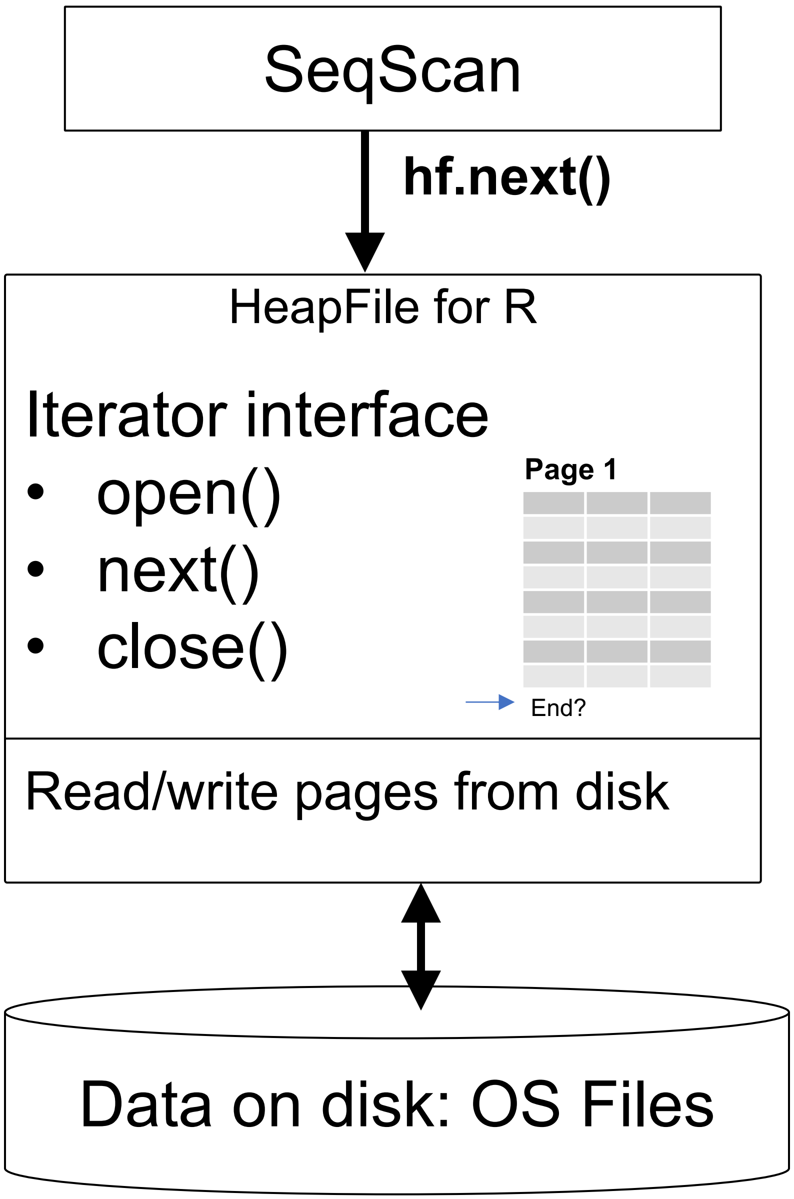
Query Execution In SimpleDB



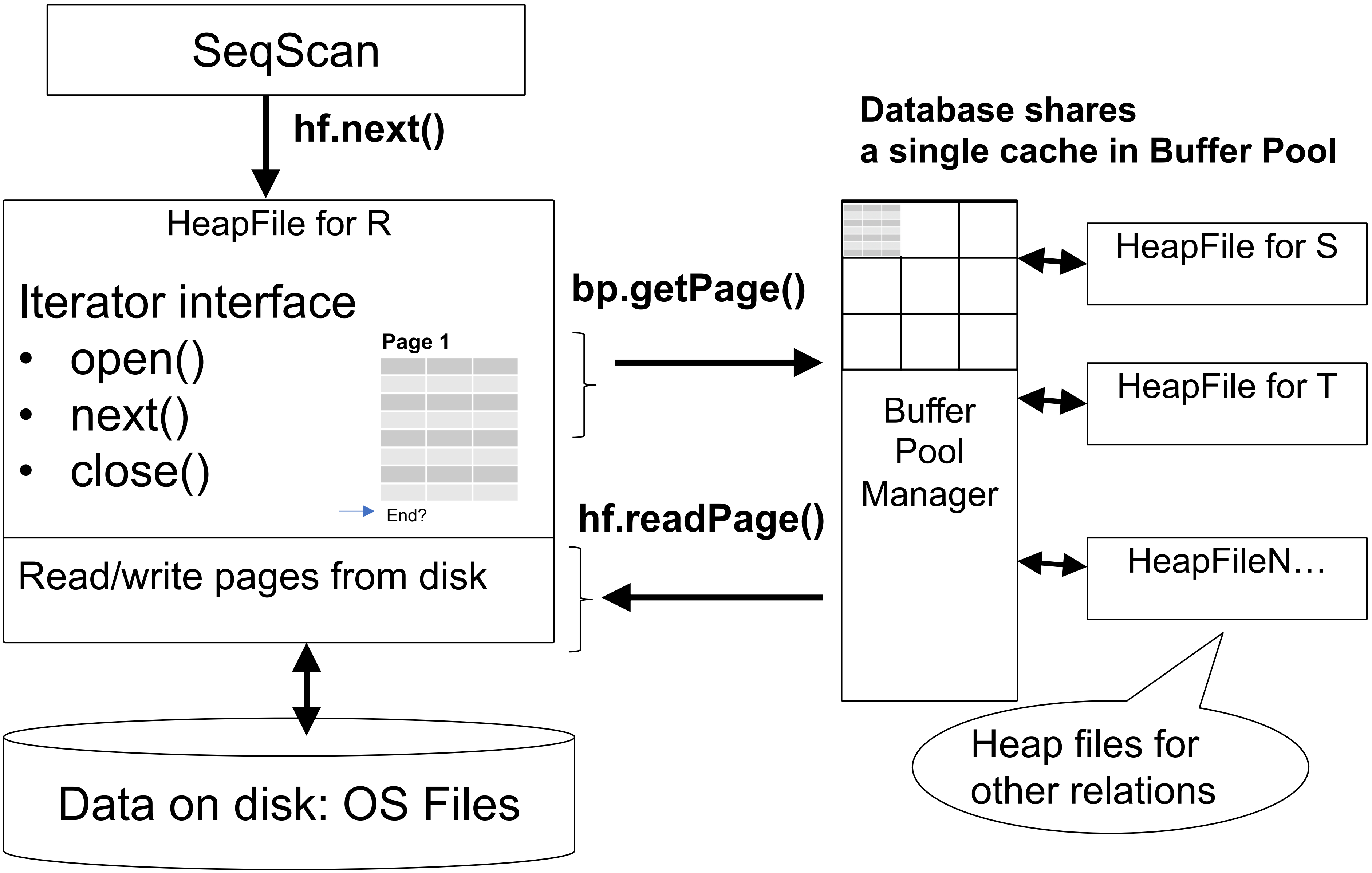
Query Execution In SimpleDB



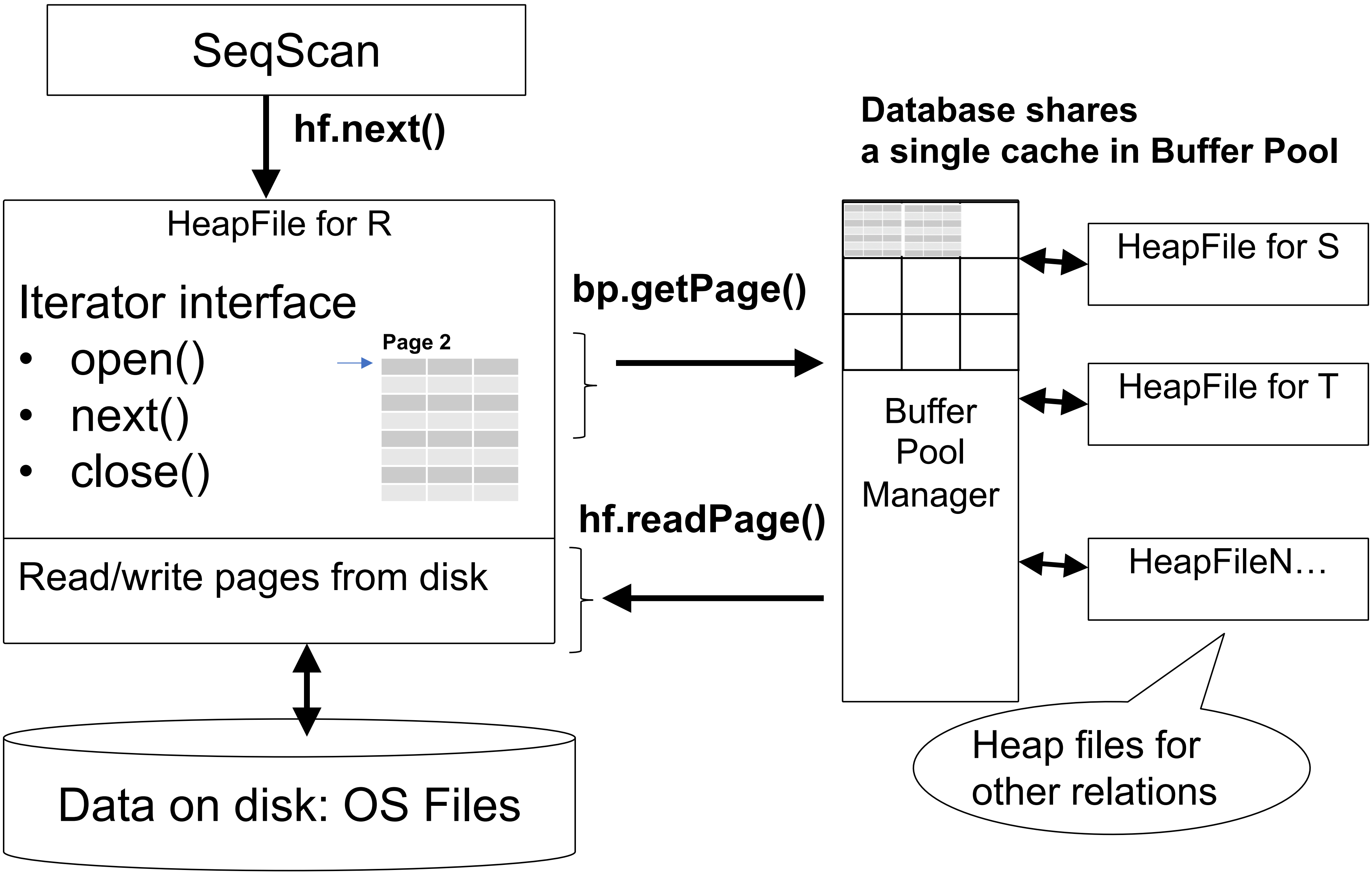
Query Execution In SimpleDB



Query Execution In SimpleDB



Query Execution In SimpleDB



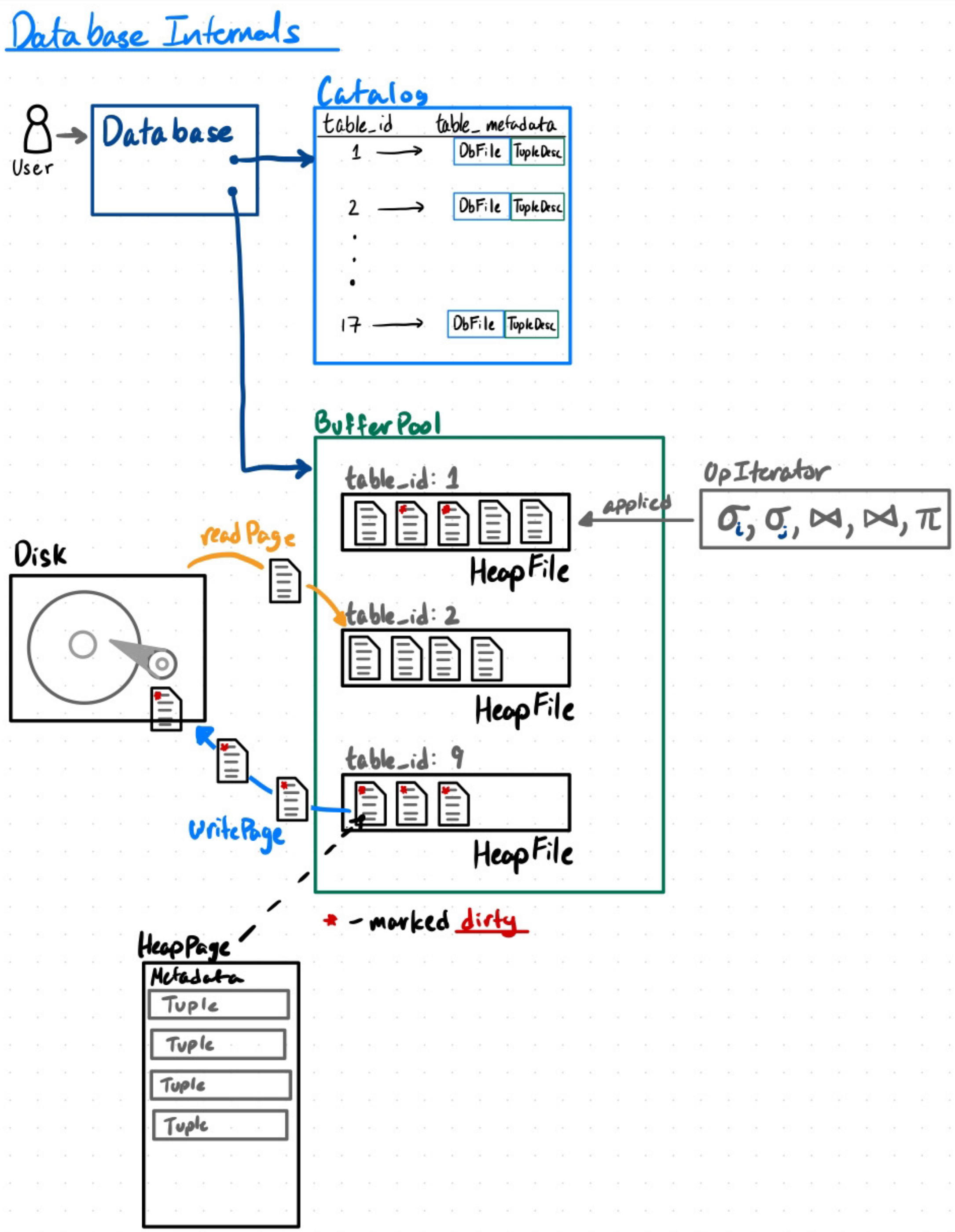
HeapFile In SimpleDB

- 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.



Database Internals
User
Database
Catalog
table_id
table_metadata
1
DbFile
TupleDesc
2
DbFile
TupleDesc
17
DbFile
TupleDesc
BufferPool
table_id: 1
HeapFile
table_id: 2
HeapFile
table_id: 9
HeapFile
applied
OpIterator
σ_i, σ_j, ⋈, ⋈, π
Disk
readPage
writePage
HeapPage
Metadata
Tuple
Tuple
Tuple
Tuple
* - marked dirty