CSE 344

APRIL 20TH – RDBMS INTERNALS

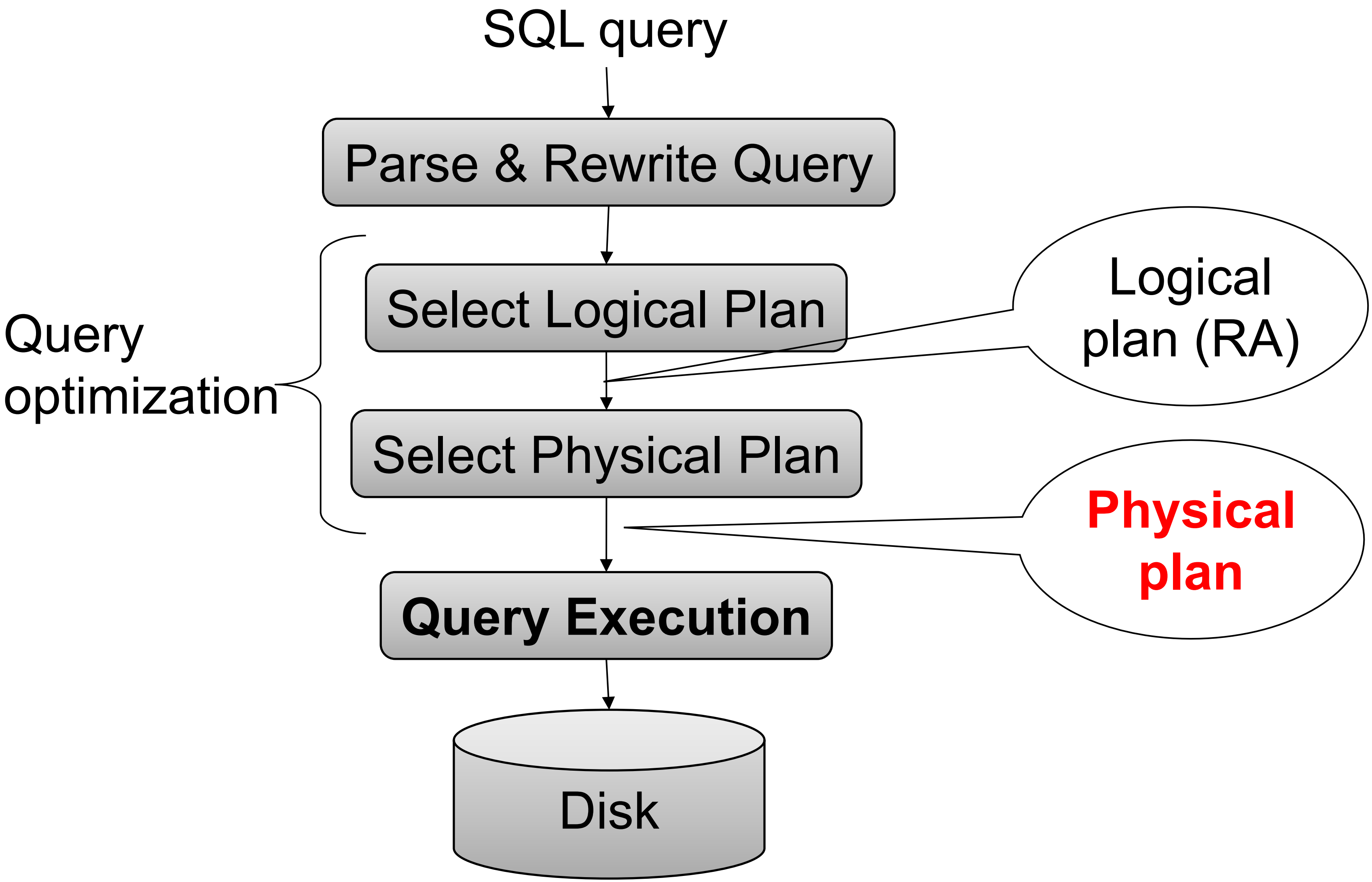
ADMINISTRIVIA

- **OQ5 Out**
  - Datalog – Due next Wednesday
- **HW4 Due next Wednesday**
  - Written portion (.pdf)
  - Coding portion (one .dl file)

TODAY

- **Back to RDBMS**
  - ”Query plans” and DBMS planning
  - Management between SQL and execution
  - Optimization techniques
  - Indexing and data arrangement

QUERY EVALUATION STEPS



LOGICAL VS PHYSICAL PLANS

Logical plans:

- Created by the parser from the input SQL text
- Expressed as a relational algebra tree
- Each SQL query has many possible logical plans

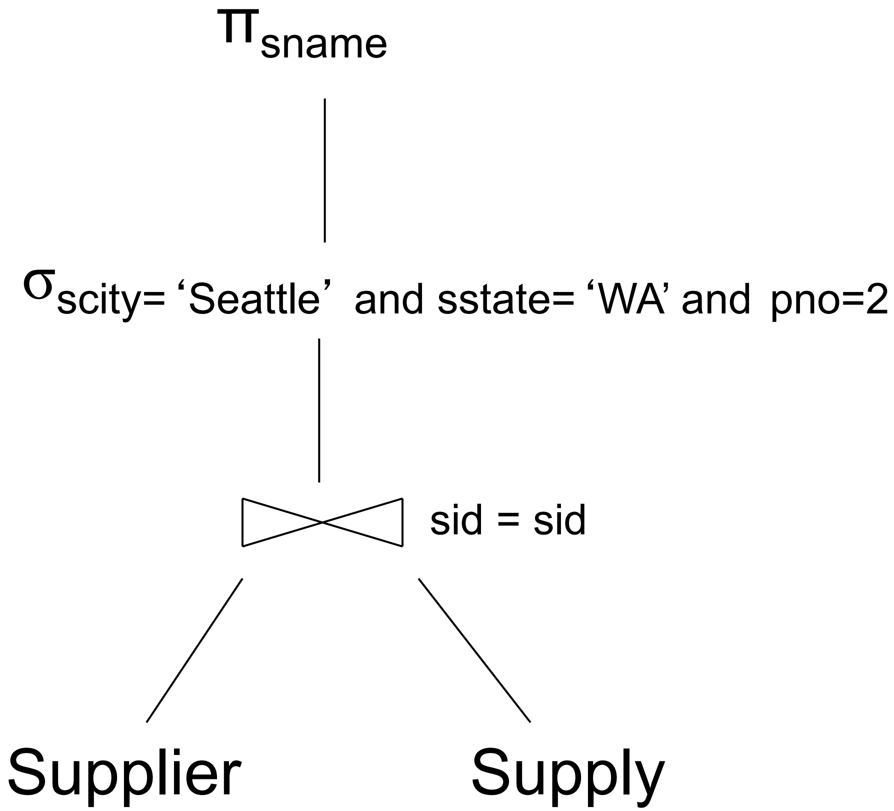
Physical plans:

- Goal is to choose an efficient implementation for each operator in the RA tree
- Each logical plan has many possible physical plans

REVIEW: RELATIONAL ALGEBRA

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```



π_{sname}

$\sigma_{\text{scity= ‘Seattle’ and sstate= ‘WA’ and pno=2}}$

$\bowtie$ sid = sid

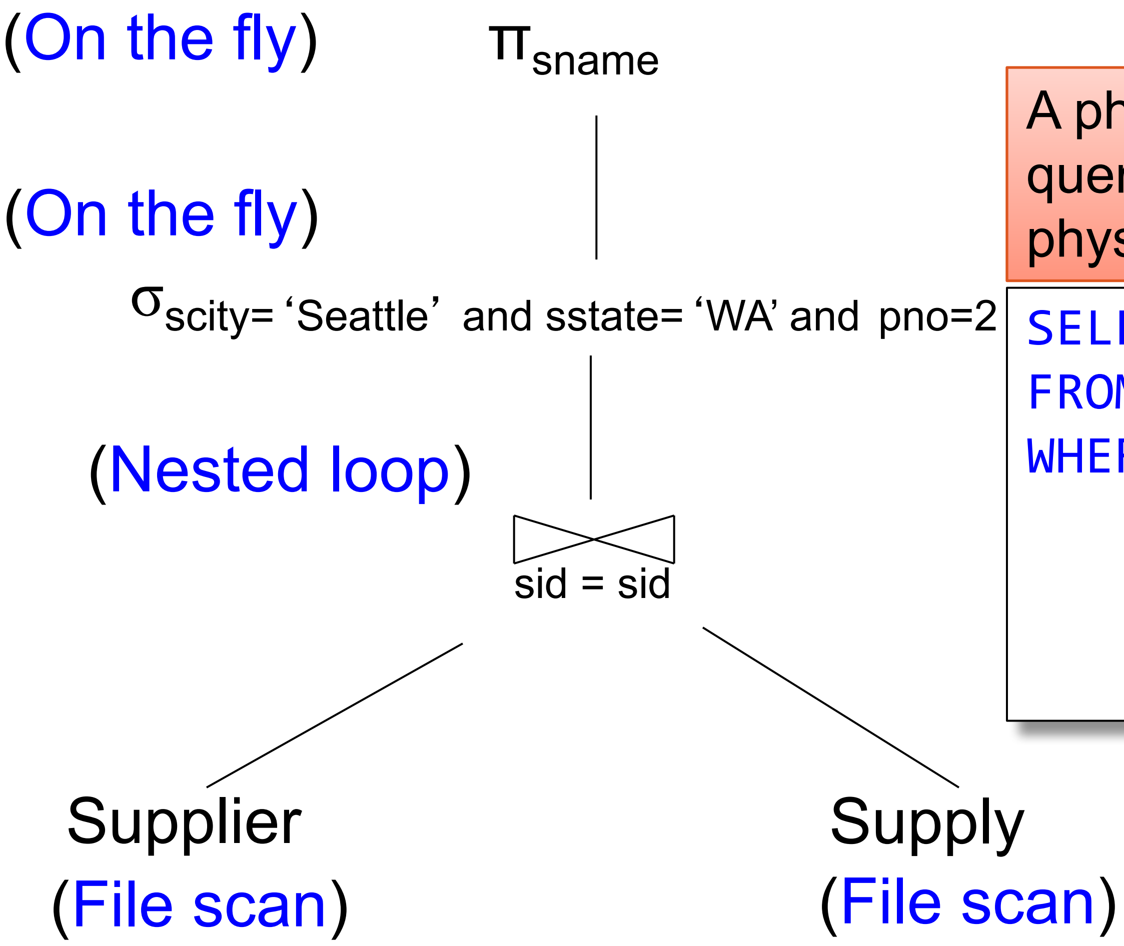
Supplier

Supply

Relational algebra expression is also called the “logical query plan”

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PHYSICAL QUERY PLAN 1

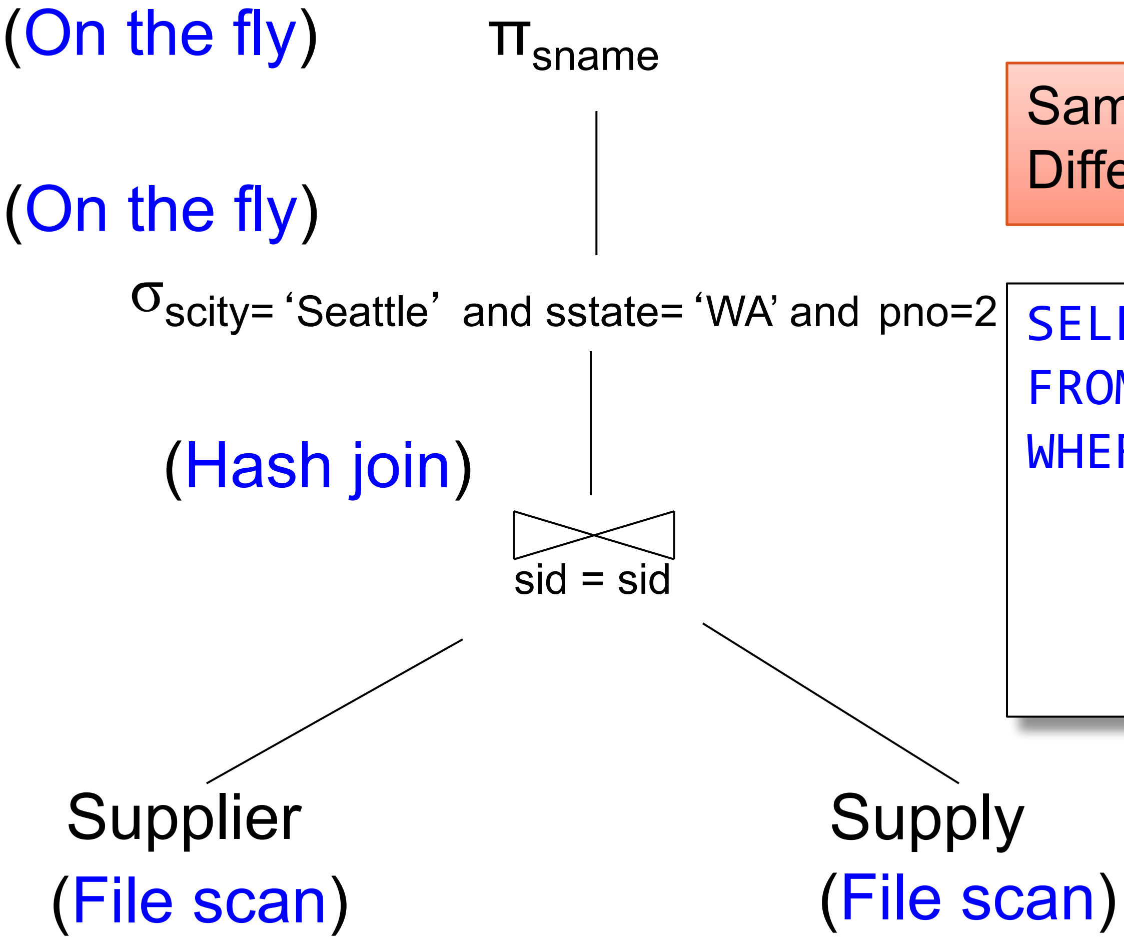


A physical query plan is a logical query plan annotated with physical implementation details

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PHYSICAL QUERY PLAN 2



(On the fly) π_{sname}

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Hash join) $\bowtie_{sid = sid}$

Supplier (File scan)

Supply (File scan)

Same logical query plan
Different physical plan

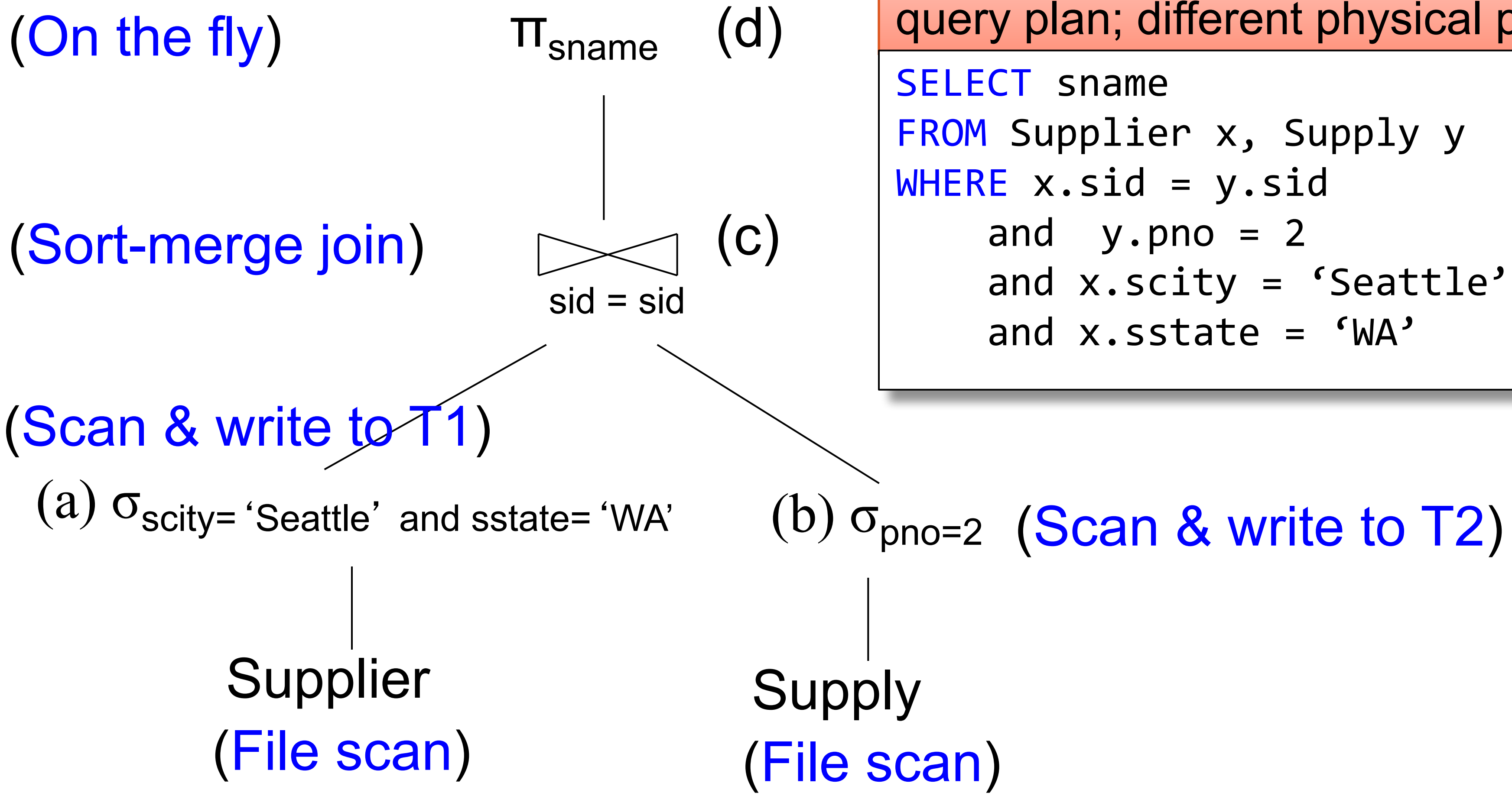
```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PHYSICAL QUERY PLAN 3

Different but equivalent logical query plan; different physical plan

```
SELECT sname
FROM Supplier x, Supply y
WHERE x.sid = y.sid
    and  y.pno = 2
    and x.scity = ‘Seattle’
    and x.sstate = ‘WA’
```



(On the fly) π_{sname} (d)

(Sort-merge join) $\bowtie_{sid = sid}$ (c)

(Scan & write to T1)

(a) $\sigma_{scity=\text{‘Seattle’ and sstate=‘WA’}}$

(b) $\sigma_{pno=2}$ (Scan & write to T2)

Supplier (File scan)

Supply (File scan)

QUERY OPTIMIZATION PROBLEM

For each SQL query… many logical plans

For each logical plan… many physical plans

Next: we will discuss physical operators;
how exactly are query executed?

PHYSICAL OPERATORS

Each of the logical operators may have one or more implementations = physical operators

Will discuss several basic physical operators, with a focus on join

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

MAIN MEMORY ALGORITHMS

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Propose three physical operators for the join, assuming the tables are in main memory:

1.

2.

3.

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

MAIN MEMORY ALGORITHMS

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Propose three physical operators for the join, assuming the tables are in main memory:

1. **Nested Loop Join O(??)**
2. **Merge join O(??)**
3. **Hash join O(??)**

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

MAIN MEMORY ALGORITHMS

Logical operator:

Supplier $\bowtie_{sid=sid}$ Supply

Propose three physical operators for the join, assuming the tables are in main memory:

1.	**Nested Loop Join**	**$O(n^2)$**
2.	**Merge join**	**$O(n \log n)$**
3.	**Hash join**	**$O(n) \ldots O(n^2)$**

BRIEF REVIEW OF HASH TABLES

A (naïve) hash function:

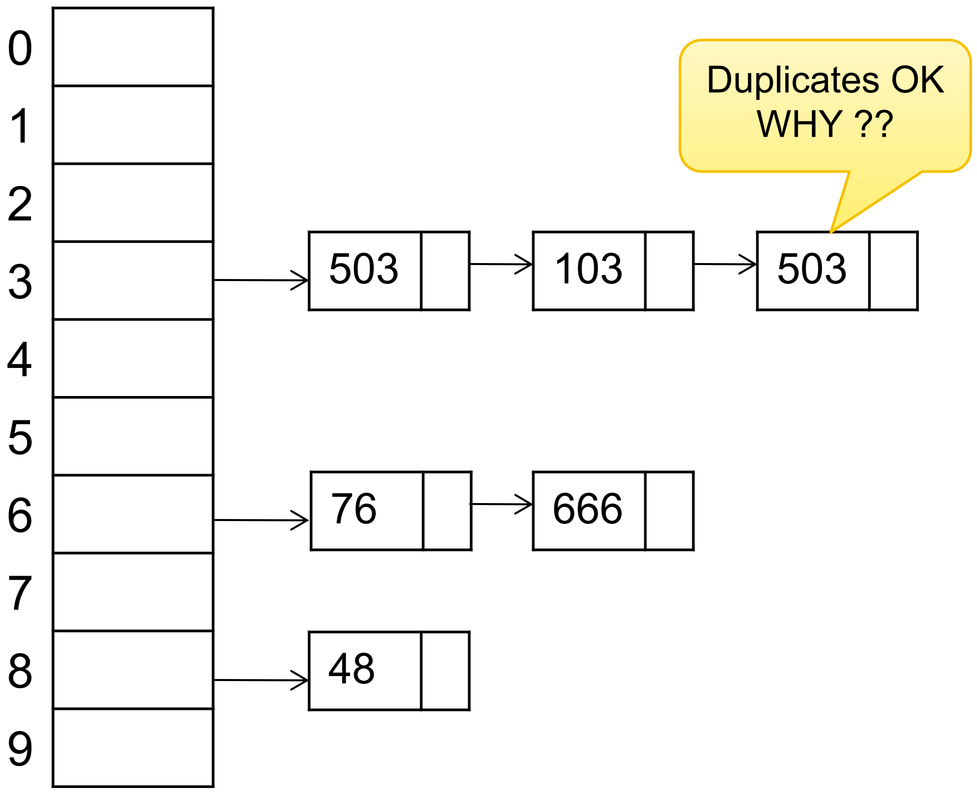
h(x) = x mod 10

Operations:

find(103) = ??

insert(488) = ??

Separate chaining:



Duplicates OK
WHY ??

BRIEF REVIEW OF HASH TABLES

insert(k, v) = inserts a key k with value v

Many values for one key

- Hence, duplicate k’s are OK

find(k) = returns the *list* of all values v associated to the key k

ITERATOR INTERFACE

Each operator implements three methods:

open()

next()

close()

ITERATOR INTERFACE

Example “on the fly” selection operator

```
interface Operator {

  // initializes operator state
  // and sets parameters
  void open (...);


  // calls next() on its inputs
  // processes an input tuple
  // produces output tuple(s)
  // returns null when done
  Tuple next ();


  // cleans up (if any)
  void close ();
}
```

ITERATOR INTERFACE

```
interface Operator {

  // initializes operator state
  // and sets parameters
  void open (...);


  // calls next() on its inputs
  // processes an input tuple
  // produces output tuple(s)
  // returns null when done
  Tuple next ();


  // cleans up (if any)
  void close ();
}
```

Example “on the fly” selection operator

```
class Select implements Operator {...
  void open (Predicate p,
             Operator child) {
    this.p = p; this.child = child;
  }
}
```

ITERATOR INTERFACE

```
interface Operator {

  // initializes operator state
  // and sets parameters
  void open (...);


  // calls next() on its inputs
  // processes an input tuple
  // produces output tuple(s)
  // returns null when done
  Tuple next ();


  // cleans up (if any)
  void close ();
}
```

Example “on the fly” selection operator

```
class Select implements Operator {...
  void open (Predicate p,
             Operator child) {
    this.p = p; this.child = child;
  }
  Tuple next () {






  }
}
```

ITERATOR INTERFACE

```
interface Operator {

  // initializes operator state
  // and sets parameters
  void open (...);


  // calls next() on its inputs
  // processes an input tuple
  // produces output tuple(s)
  // returns null when done
  Tuple next ();


  // cleans up (if any)
  void close ();
}
```

Example “on the fly” selection operator

```
class Select implements Operator {...
  void open (Predicate p,
             Operator child) {
    this.p = p; this.child = child;
  }
  Tuple next () {
    boolean found = false;
    Tuple r = null;
    while (!found) {
       r = child.next();
       if (r == null) break;
       found = p(r);
    }
    return r;
  }
  void close () { child.close(); }
}
```

ITERATOR INTERFACE

```
interface Operator {

  // initializes operator state
  // and sets parameters
  void open (...);


  // calls next() on its inputs
  // processes an input tuple
  // produces output tuple(s)
  // returns null when done
  Tuple next ();


  // cleans up (if any)
  void close ();
}
```

Query plan execution

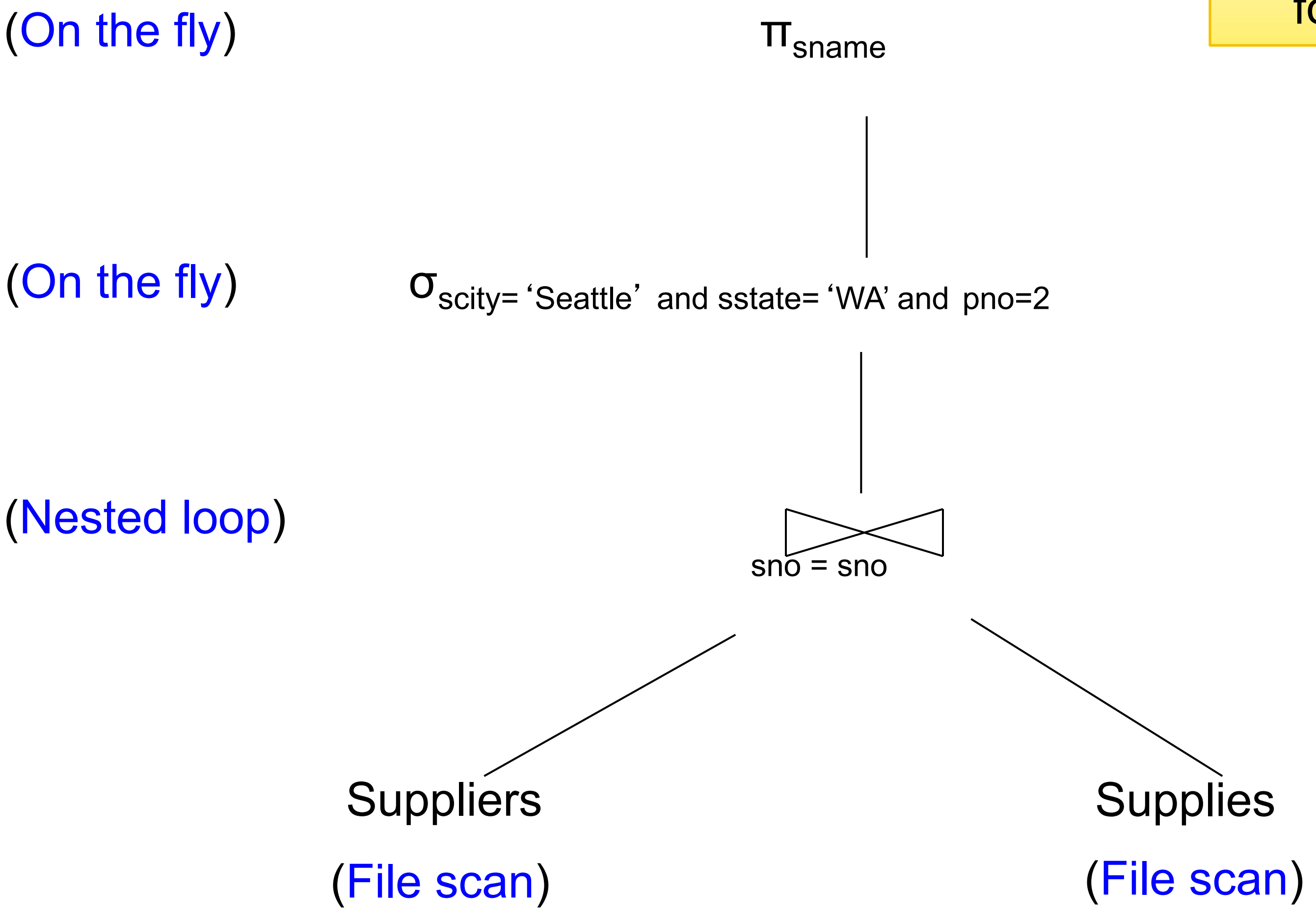
```
Operator q = parse("SELECT ...");
q = optimize(q);

q.open();
while (true) {
  Tuple t = q.next();
  if (t == null) break;
  else printOnScreen(t);
}
q.close();
```

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



(On the fly) π_{sname}

(On the fly) $\sigma_{\text{scity= 'Seattle' and sstate= 'WA' and pno=2}}$

(Nested loop) $\bowtie_{\text{sno = sno}}$

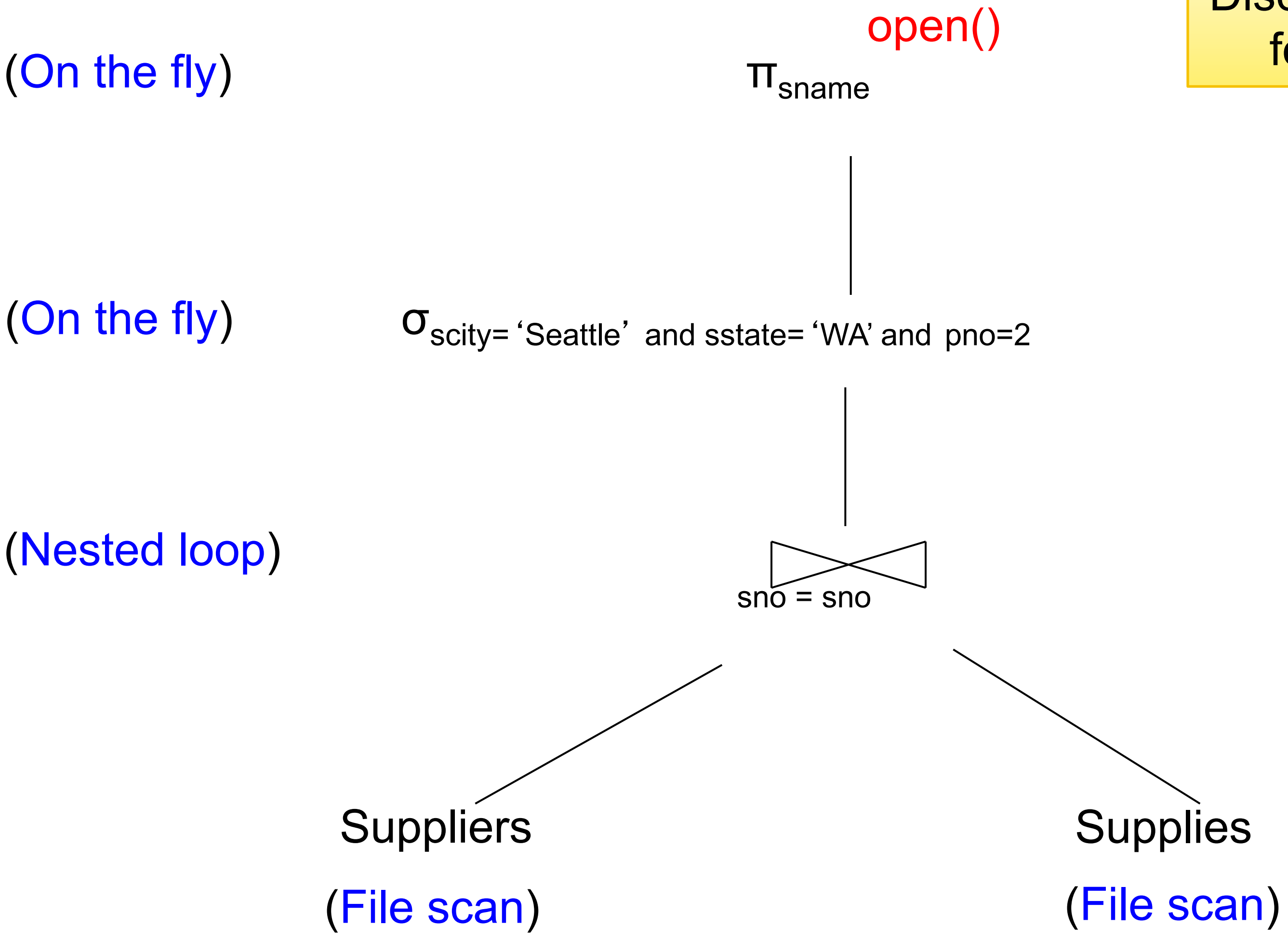
Suppliers (File scan)

Supplies (File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING

Discuss: open/next/close for nested loop join



open()

(On the fly) π_{sname}

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Nested loop) $\bowtie_{sno = sno}$

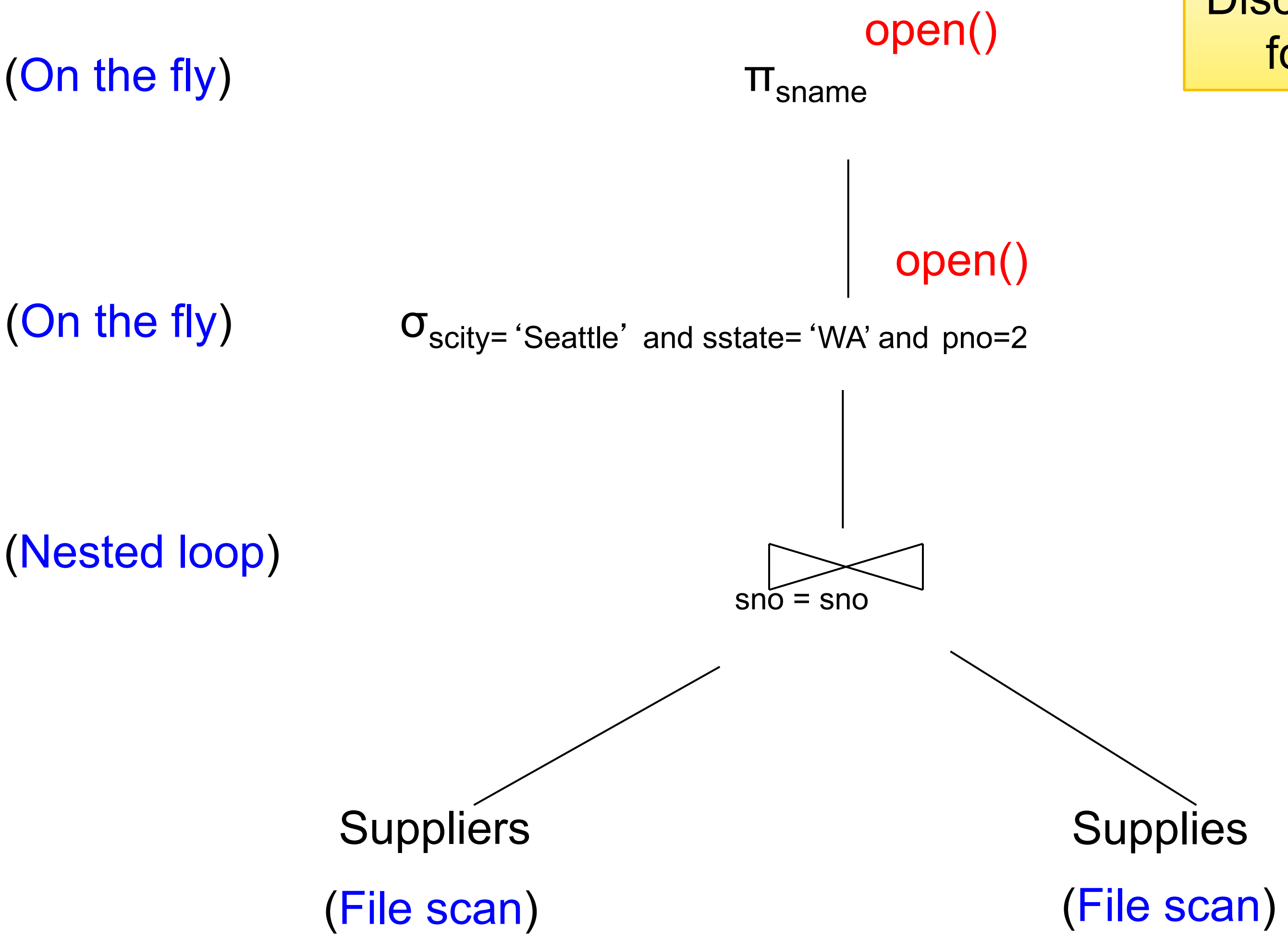
Suppliers (File scan)

Supplies (File scan)

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



open()

π_{sname}

(On the fly)

open()

$\sigma_{scity='Seattle' \text{ and } sstate='WA' \text{ and } pno=2}$

(On the fly)

$\bowtie_{sno = sno}$

(Nested loop)

Suppliers

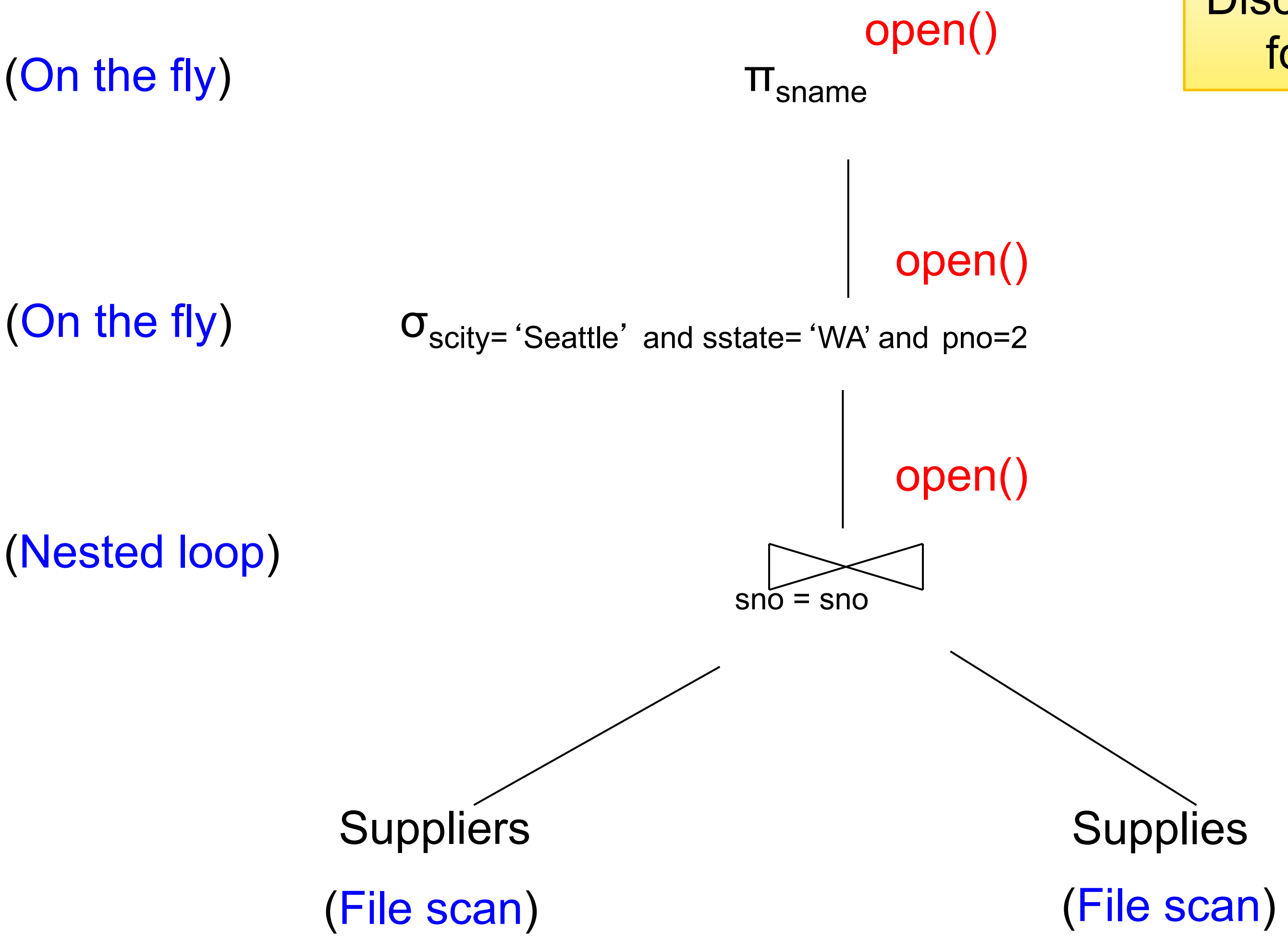
(File scan)

Supplies

(File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING



open()

(On the fly) π_{sname}

open()

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

open()

(Nested loop) ⋈ sno = sno

Suppliers

(File scan)

Supplies

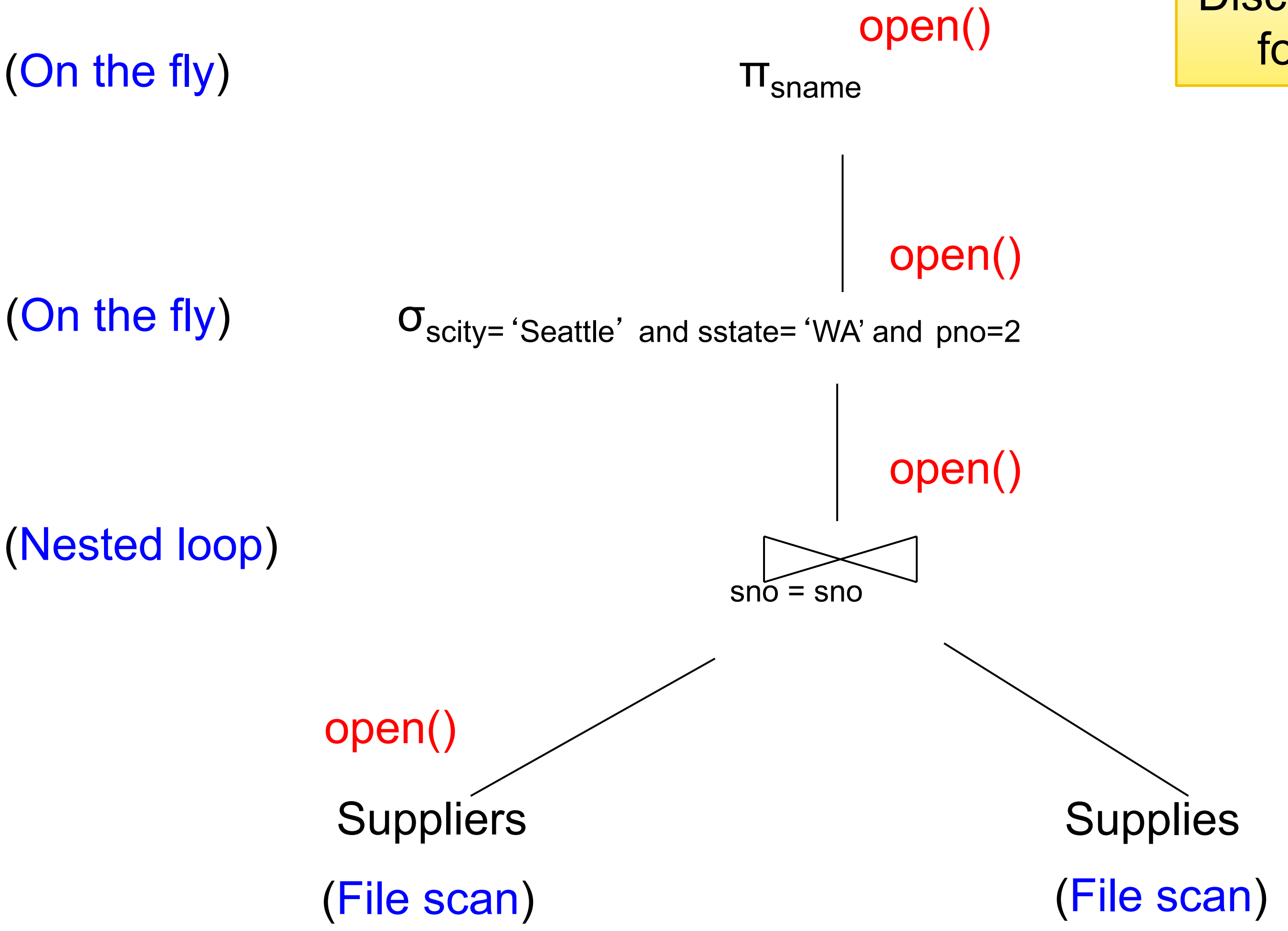
(File scan)

Discuss: open/next/close for nested loop join

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING

Discuss: open/next/close for nested loop join



(On the fly) π_{sname} open()

open()

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

open()

(Nested loop) $\bowtie_{sno = sno}$

open()

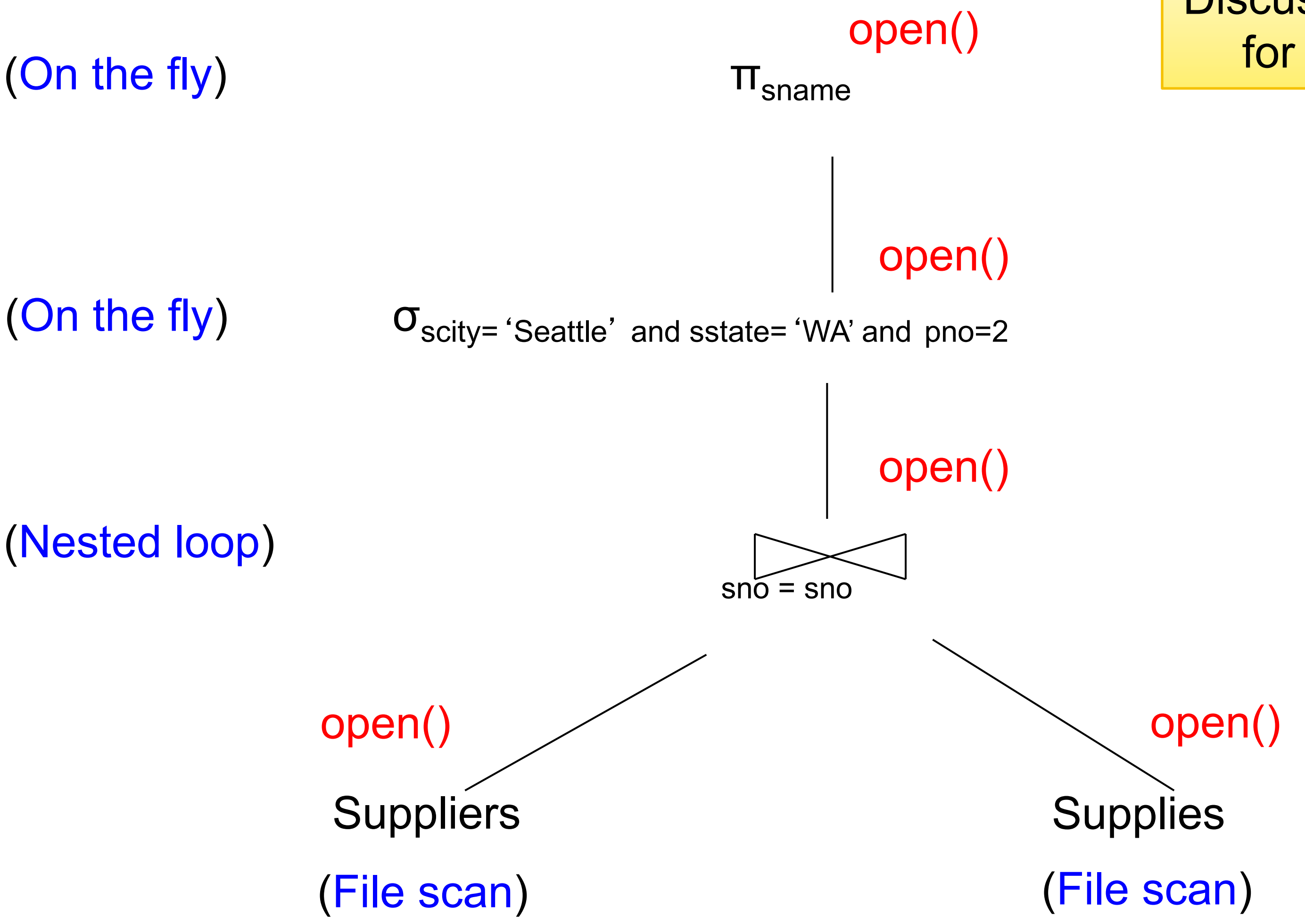
Suppliers (File scan)

Supplies (File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING

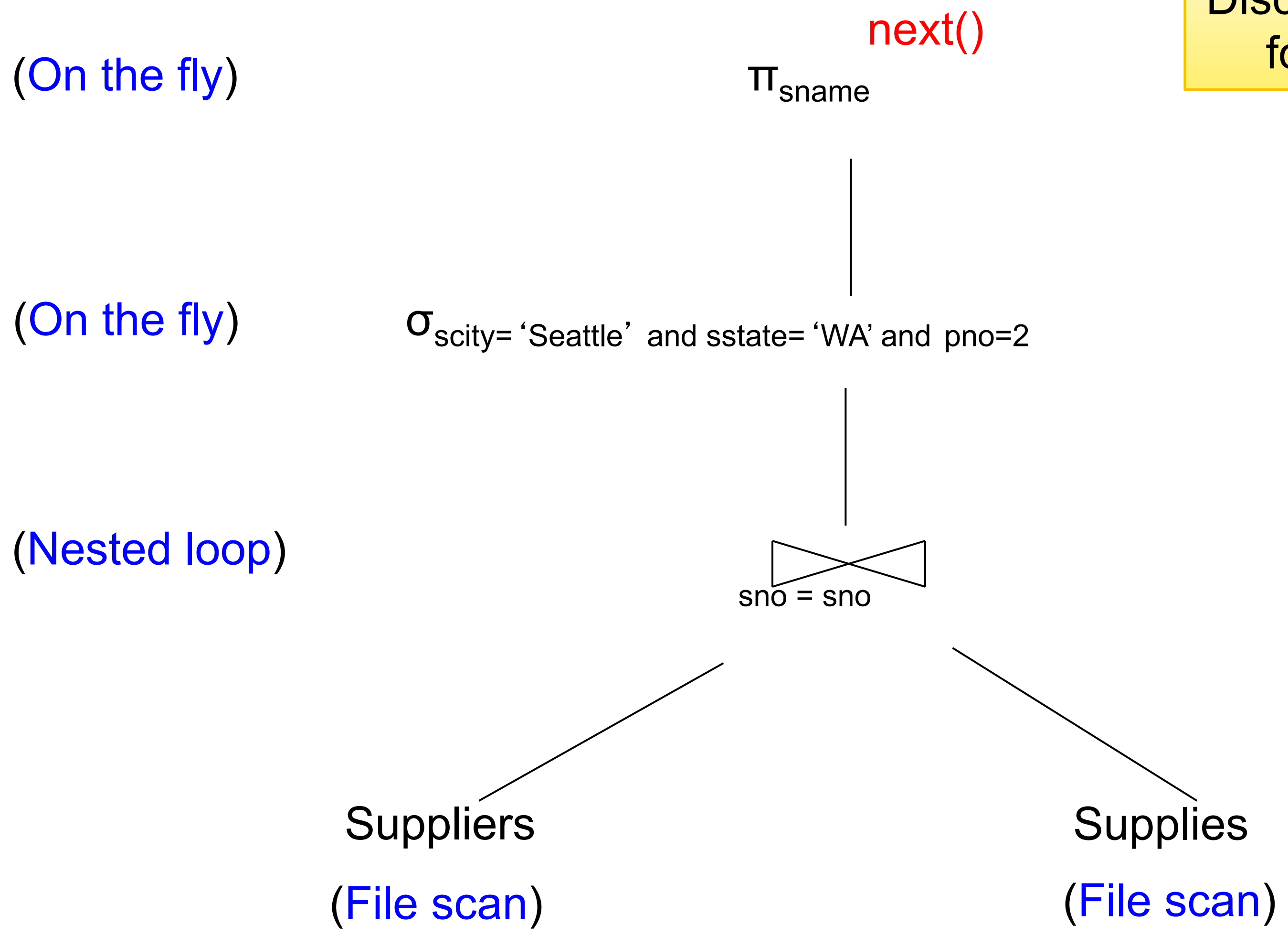
Discuss: open/next/close for nested loop join



```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



next()

(On the fly) π_{sname}

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Nested loop) $\bowtie_{sno = sno}$

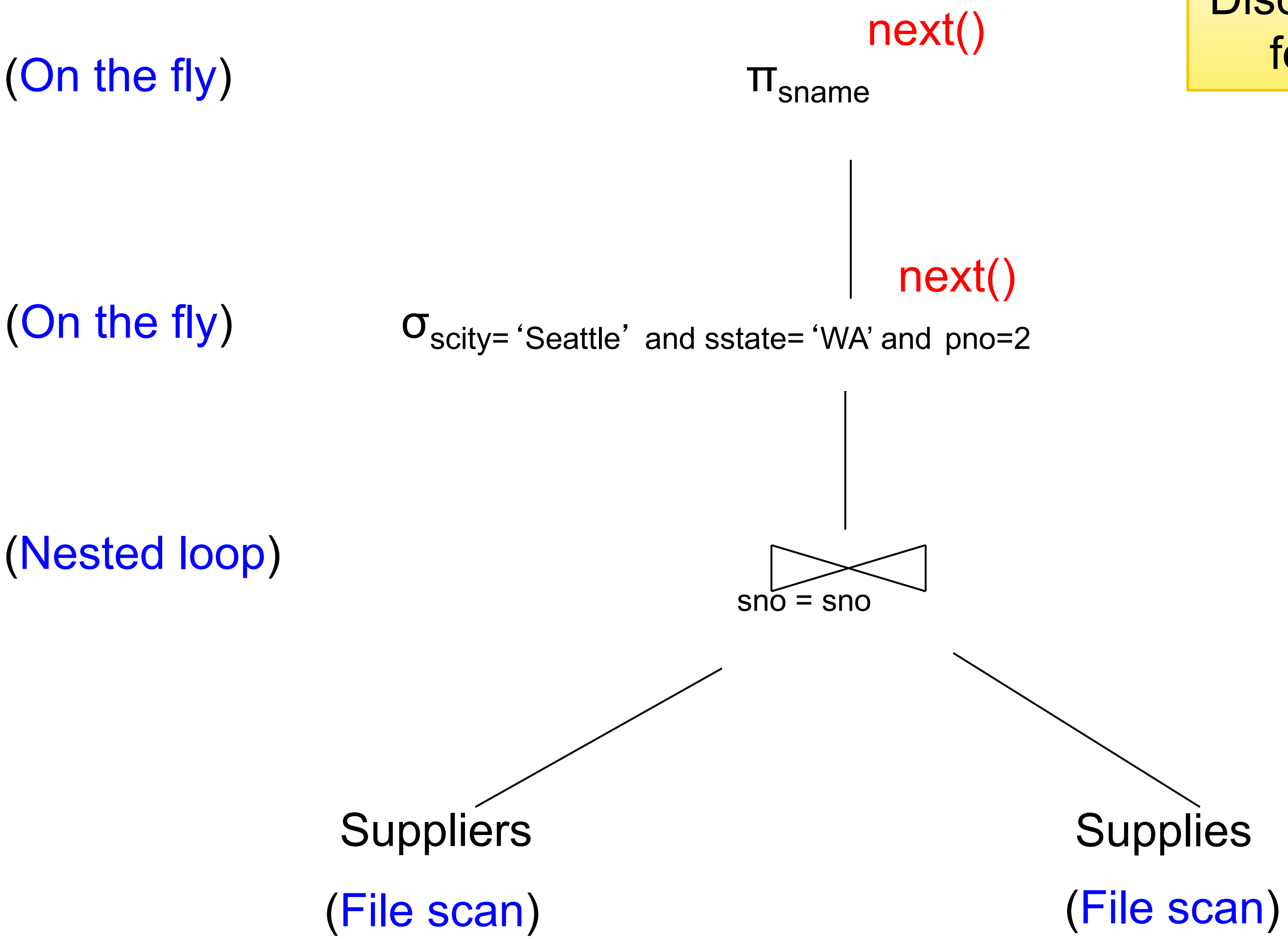
Suppliers (File scan)

Supplies (File scan)

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



next()

(On the fly) π_{sname}

next()

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Nested loop) $\bowtie_{sno = sno}$

Suppliers

(File scan)

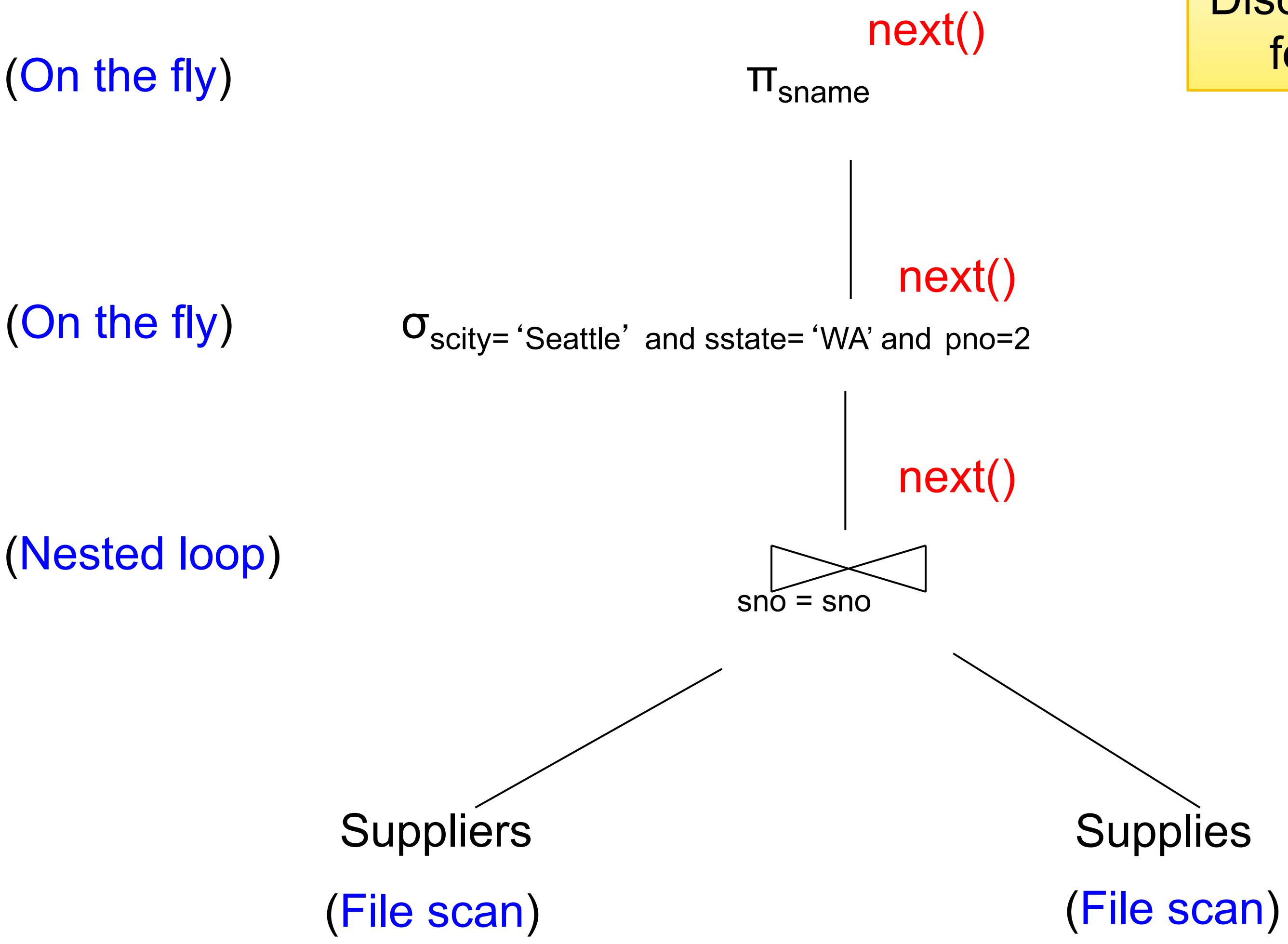
Supplies

(File scan)

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



next()

(On the fly) π_{sname}

next()

(On the fly) $\sigma_{scity='Seattle' \text{ and } sstate='WA' \text{ and } pno=2}$

next()

(Nested loop) $\bowtie_{sno\ =\ sno}$

Suppliers

(File scan)

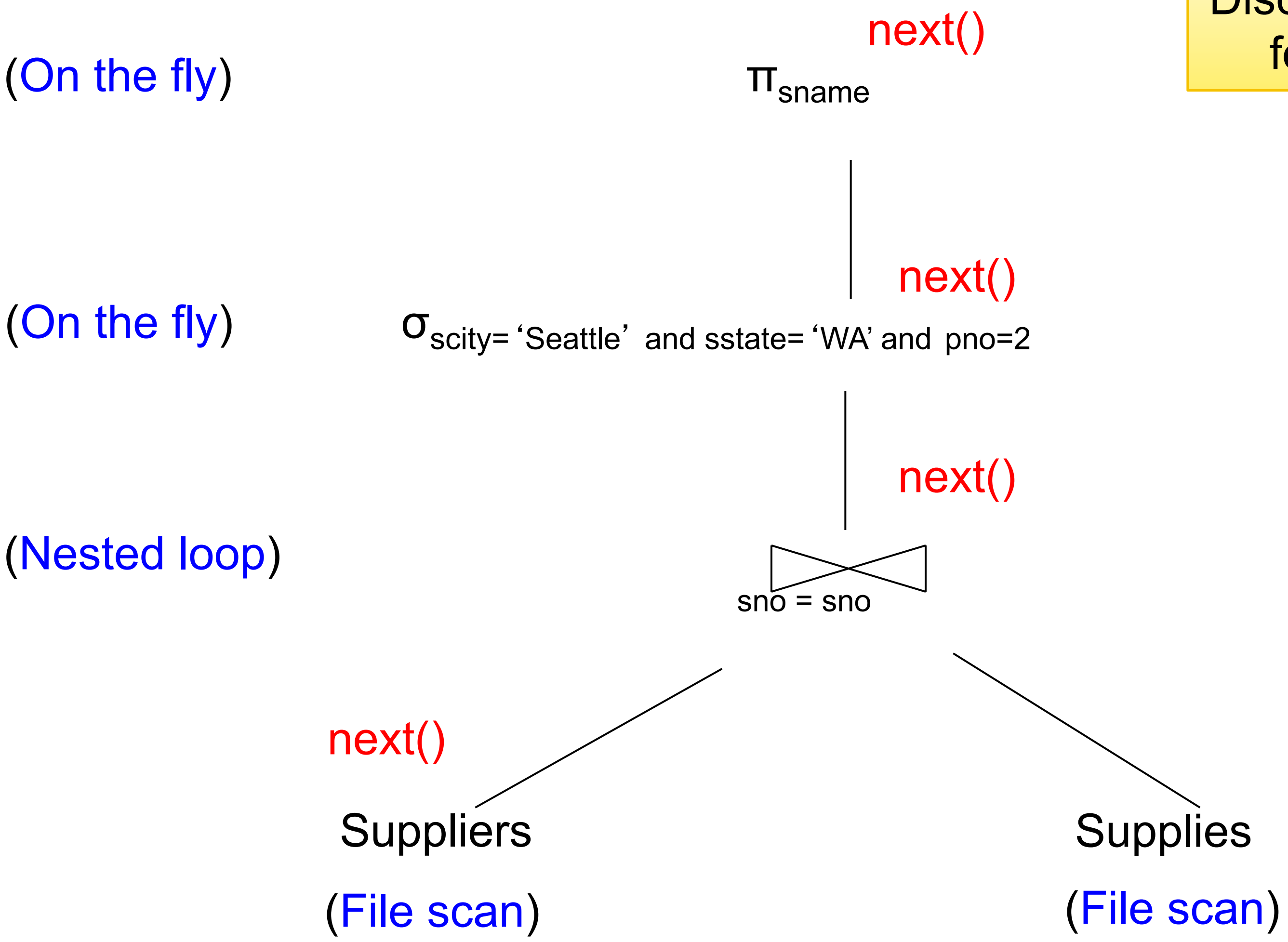
Supplies

(File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING

Discuss: open/next/close for nested loop join



next()

(On the fly) π_{sname}

next()

(On the fly) $\sigma_{scity='Seattle' \text{ and } sstate='WA' \text{ and } pno=2}$

next()

(Nested loop) $\bowtie_{sno = sno}$

next()

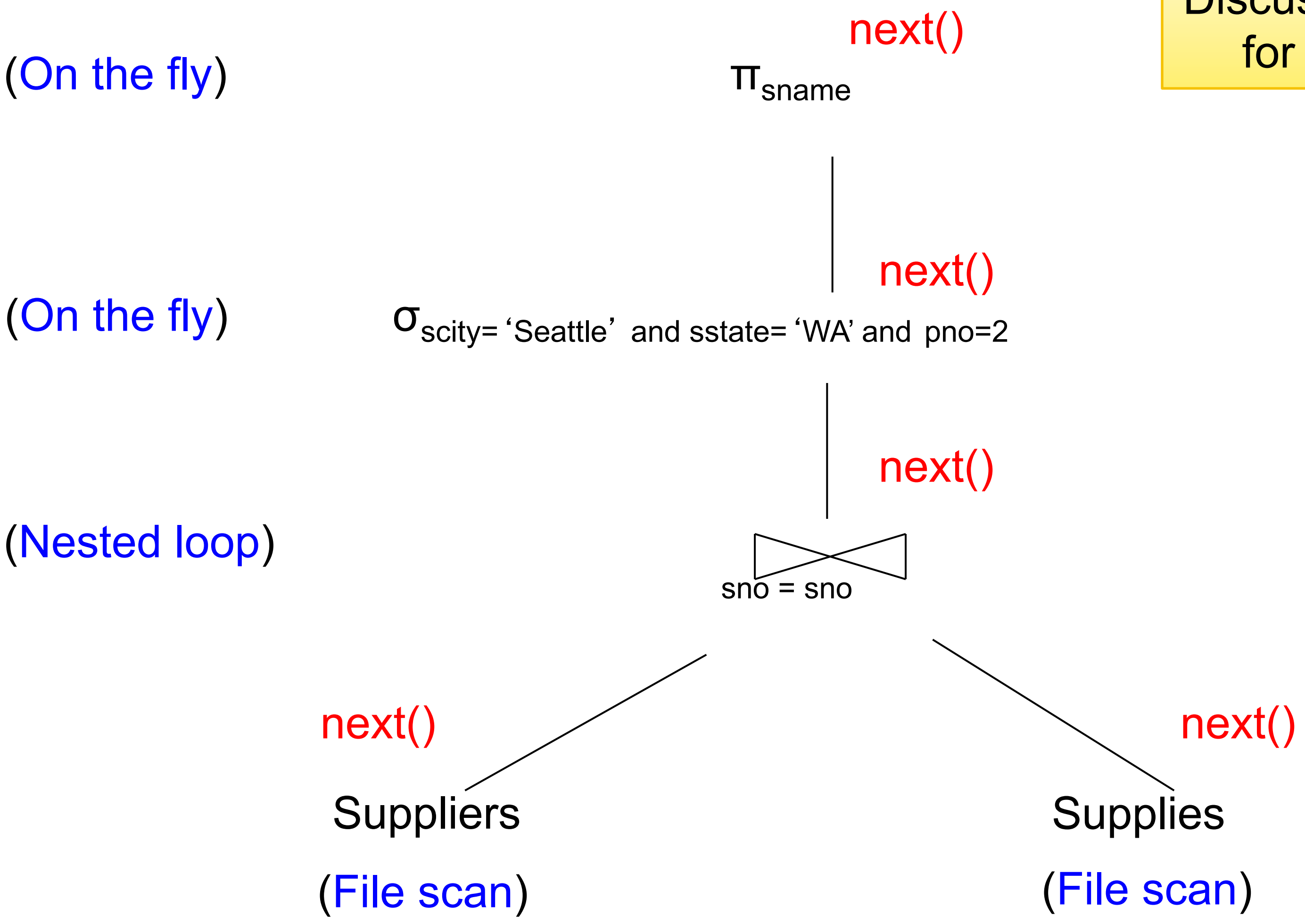
Suppliers (File scan)

Supplies (File scan)

```
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
```

PIPELINING

Discuss: open/next/close for nested loop join



(On the fly) next() π_{sname}

(On the fly) next() $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

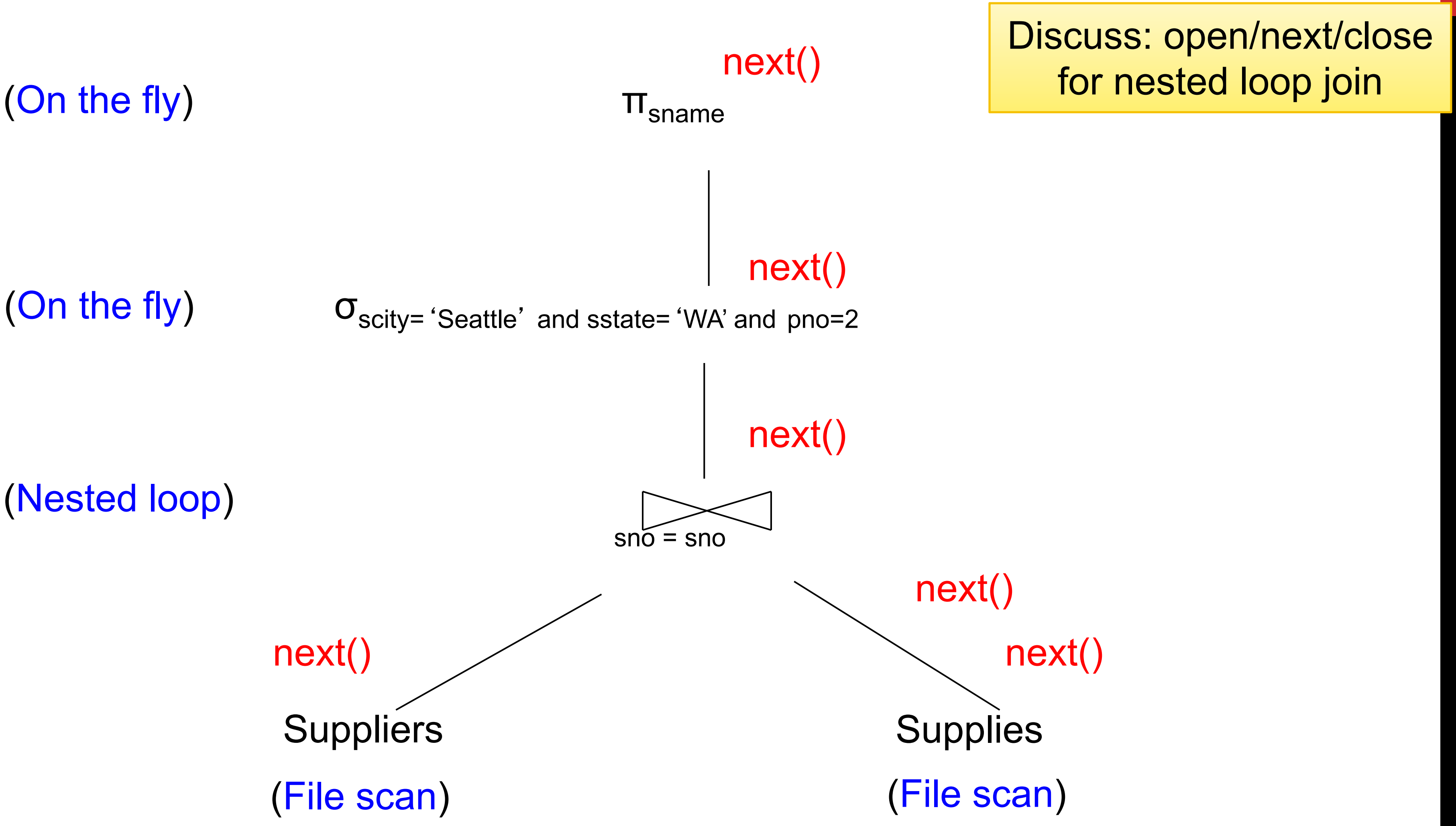
(Nested loop) next() $\bowtie_{sno = sno}$

next() Suppliers (File scan)

next() Supplies (File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

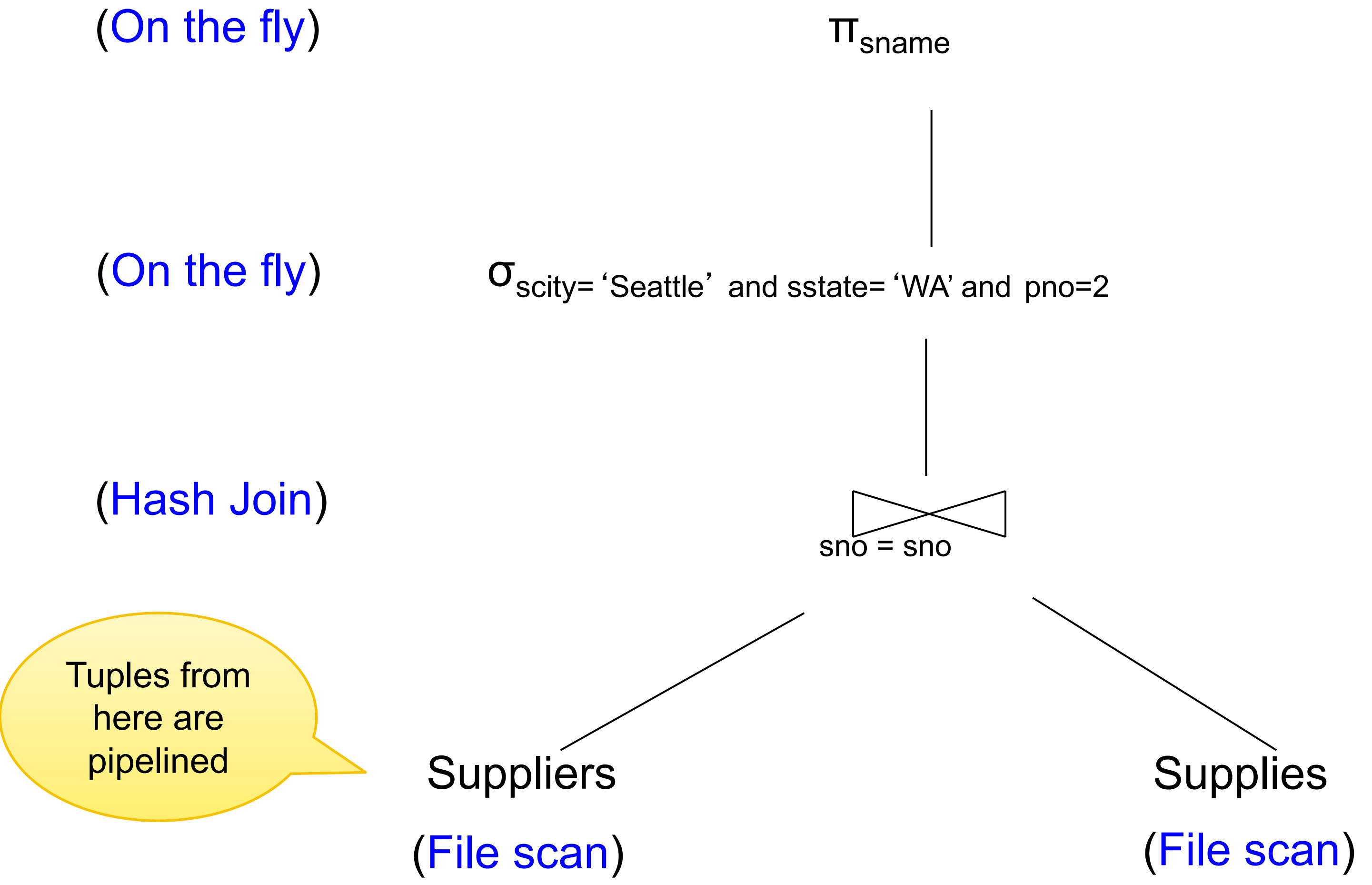
PIPELINING



Discuss: open/next/close for nested loop join

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING



(On the fly)

π_{sname}

(On the fly)

$\sigma_{\text{scity='Seattle' and sstate='WA' and pno=2}}$

(Hash Join)

$\bowtie_{\text{sno = sno}}$

Tuples from here are pipelined

Suppliers

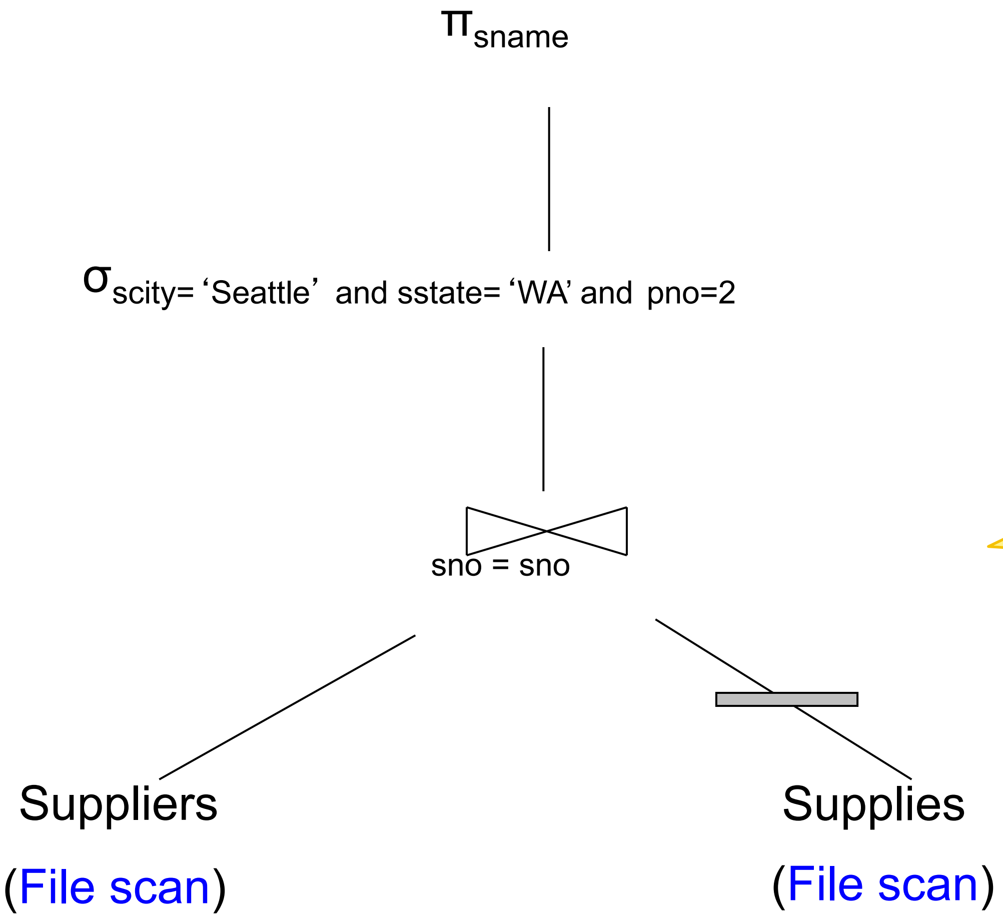
(File scan)

Supplies

(File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

PIPELINING



(On the fly)

π_{sname}

(On the fly)

$\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Hash Join)

sno = sno

Tuples from here are "blocked"

Tuples from here are pipelined

Suppliers

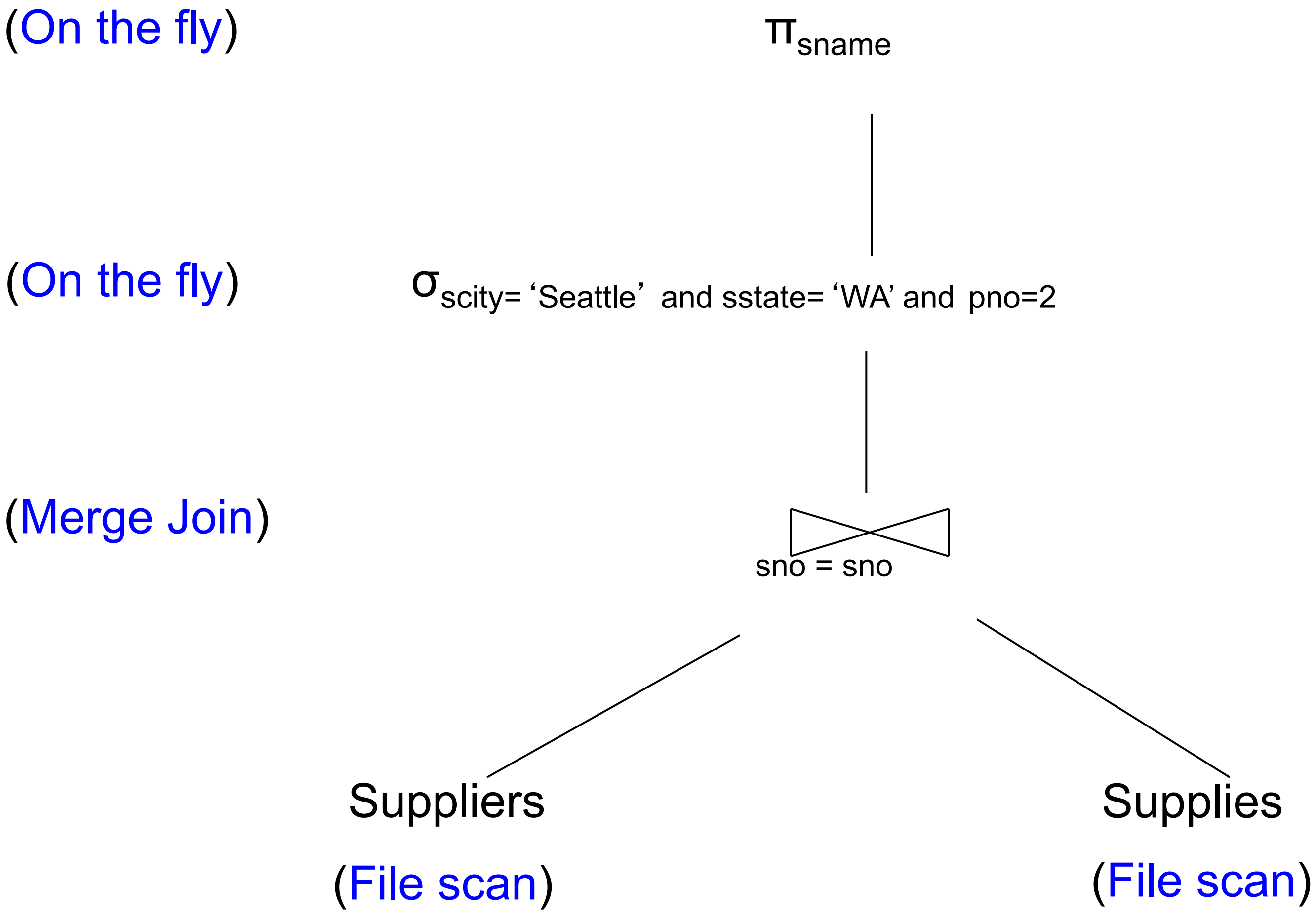
(File scan)

Supplies

(File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

BLOCKED EXECUTION



(On the fly) π_{sname}

(On the fly) $\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

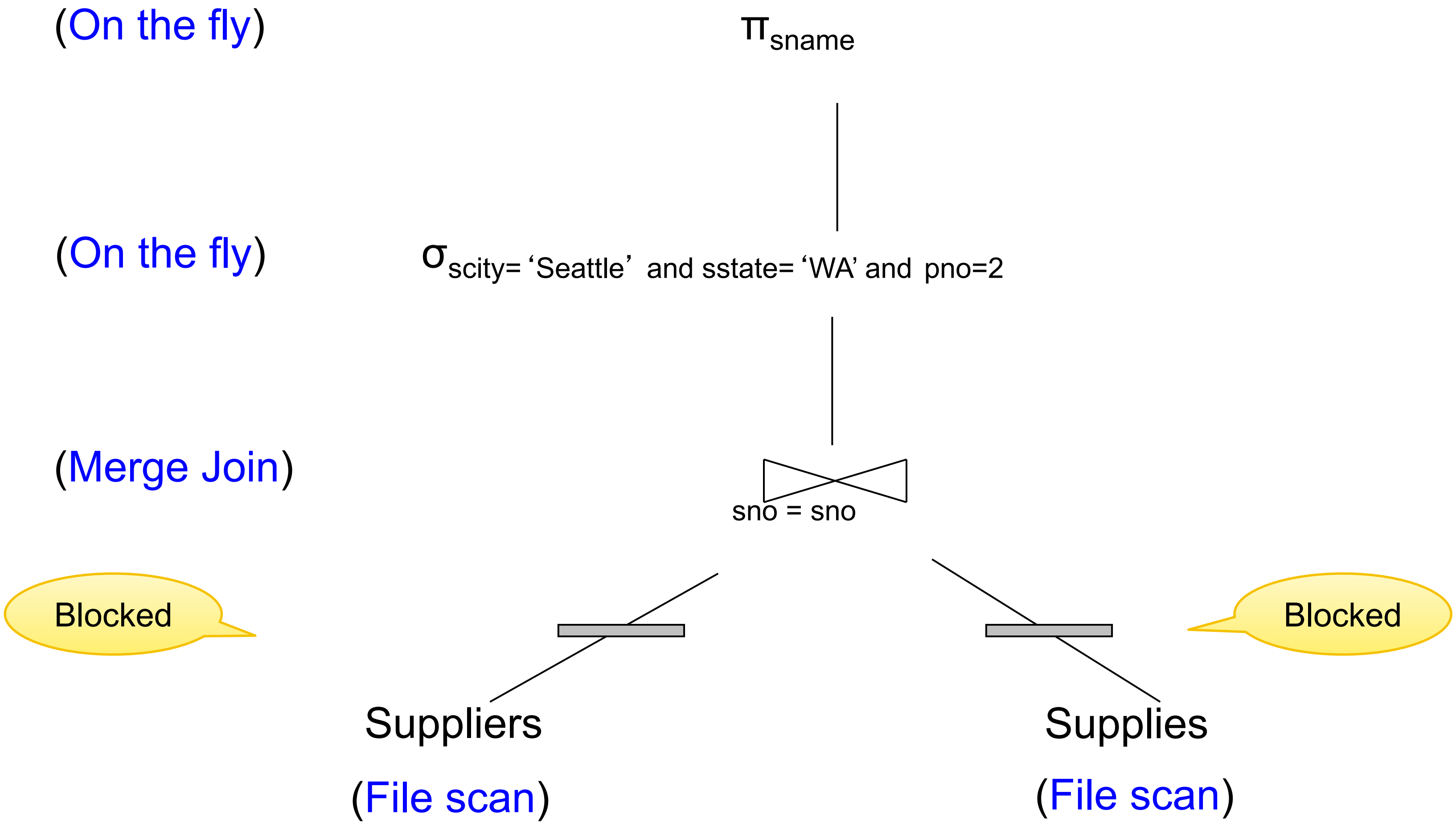
(Merge Join) $\bowtie_{sno = sno}$

Suppliers (File scan)

Supplies (File scan)

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

BLOCKED EXECUTION



(On the fly)

π_{sname}

(On the fly)

$\sigma_{scity=\text{'Seattle'} \text{ and } sstate=\text{'WA'} \text{ and } pno=2}$

(Merge Join)

$\bowtie$ sno = sno

Blocked

Blocked

Suppliers

(File scan)

Supplies

(File scan)

PIPELINED EXECUTION

Tuples generated by an operator are immediately sent to the parent

Benefits:

- No operator synchronization issues
- No need to buffer tuples between operators
- Saves cost of writing intermediate data to disk
- Saves cost of reading intermediate data from disk

This approach is used whenever possible

QUERY EXECUTION BOTTOM LINE

SQL query transformed into physical plan

- **Access path selection** for each relation
  - Scan the relation or use an index (next lecture)
- **Implementation choice** for each operator
  - Nested loop join, hash join, etc.
- **Scheduling decisions** for operators
  - Pipelined execution or intermediate materialization

Pipelined execution of physical plan

RECALL: PHYSICAL DATA INDEPENDENCE

Applications are insulated from changes in physical storage details

SQL and relational algebra facilitate physical data independence

- Both languages input and output relations
- Can choose different implementations for operators

QUERY PERFORMANCE

My database application is too slow… why?

One of the queries is very slow… why?

To understand performance, we need to understand:

- How is data organized on disk
- How to estimate query costs

- In this course we will focus on **disk-based** DBMSs