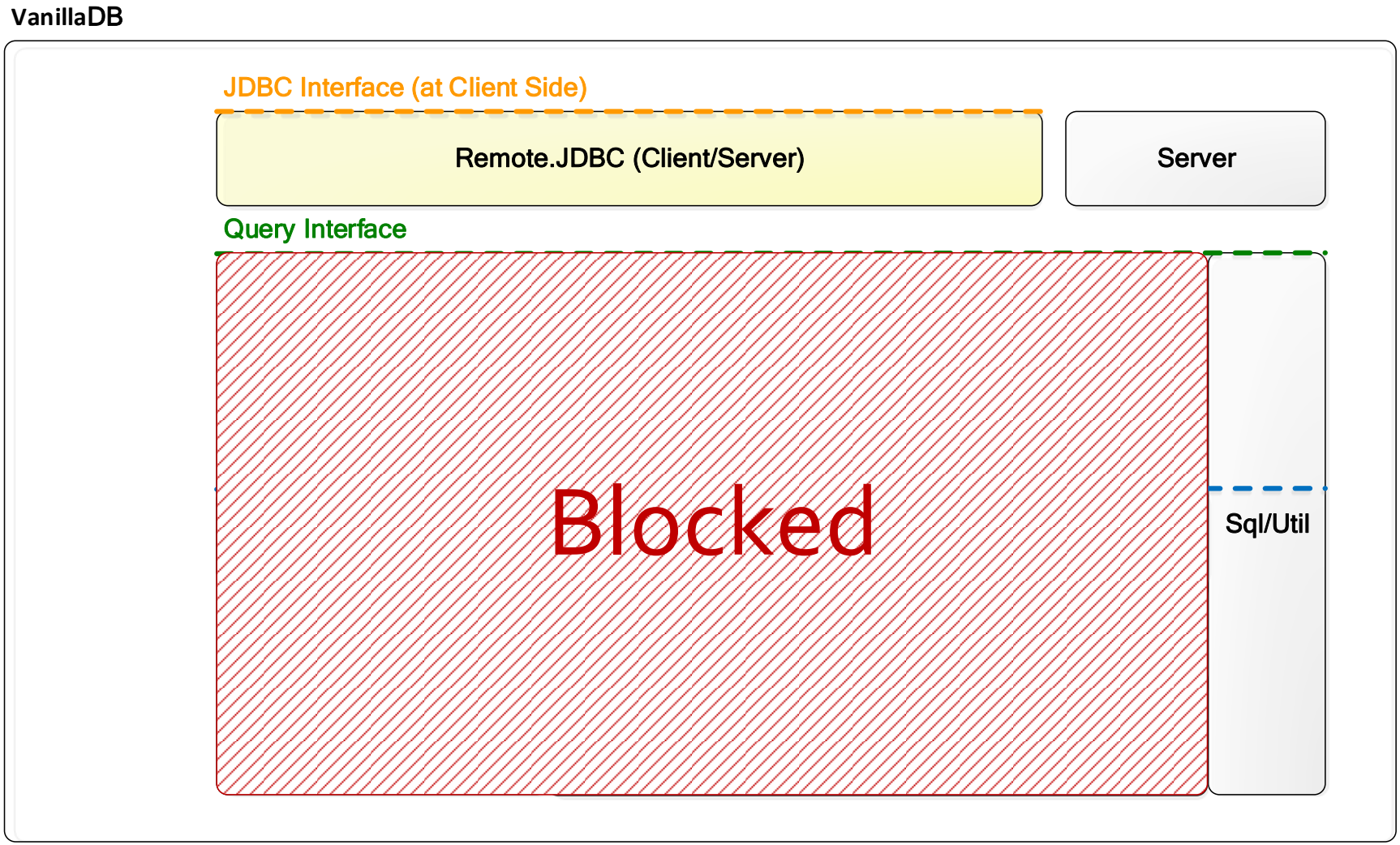
VanillaCore Walkthrough Part 2

Database systems

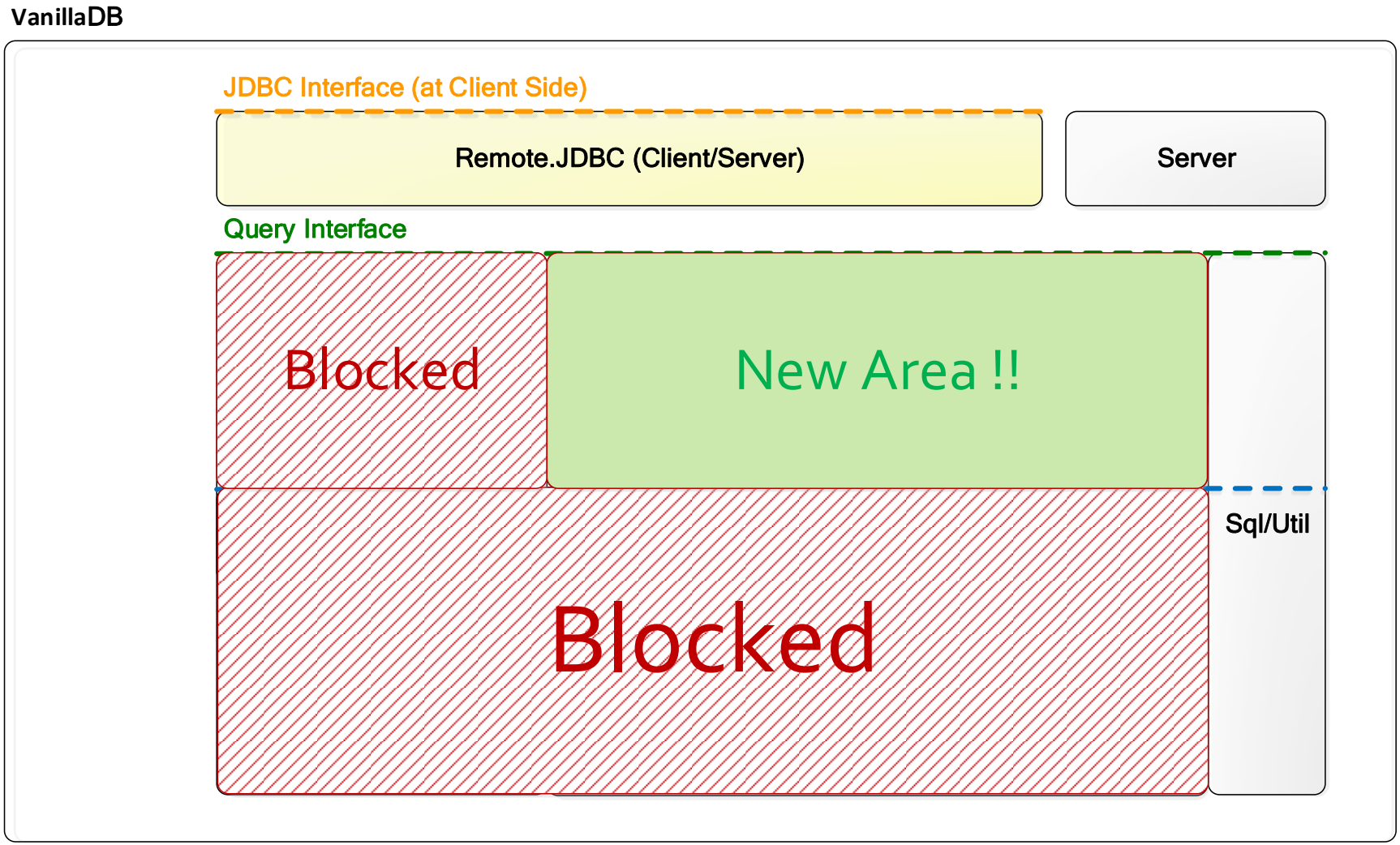
DataLab, CS, NTHU

Spring 2018

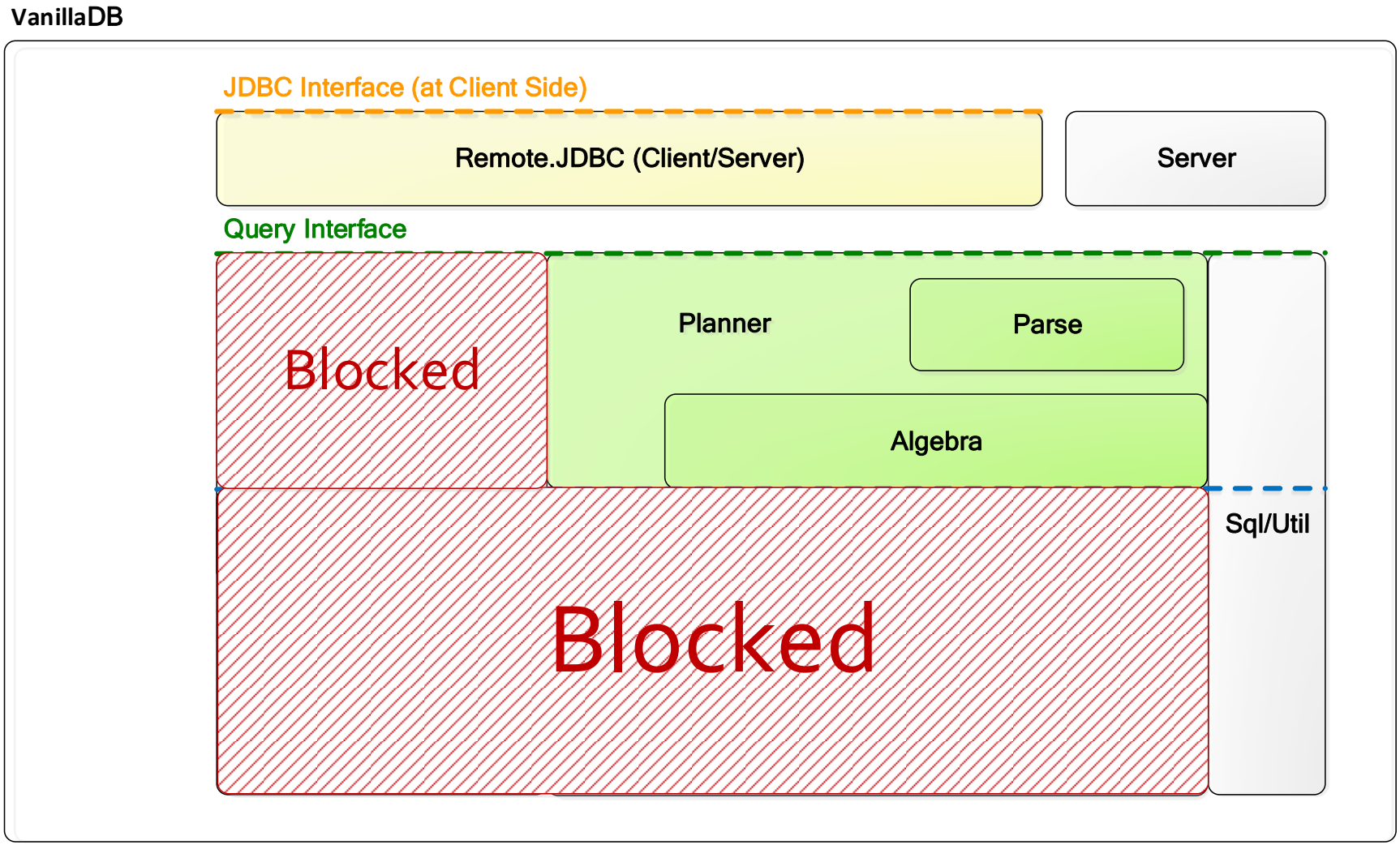
Last Time


VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
Sql/Util

This Time


VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
New Area !!
Sql/Util
Blocked

This Time


VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
Planner
Parse
Algebra
Sql/Util
Blocked

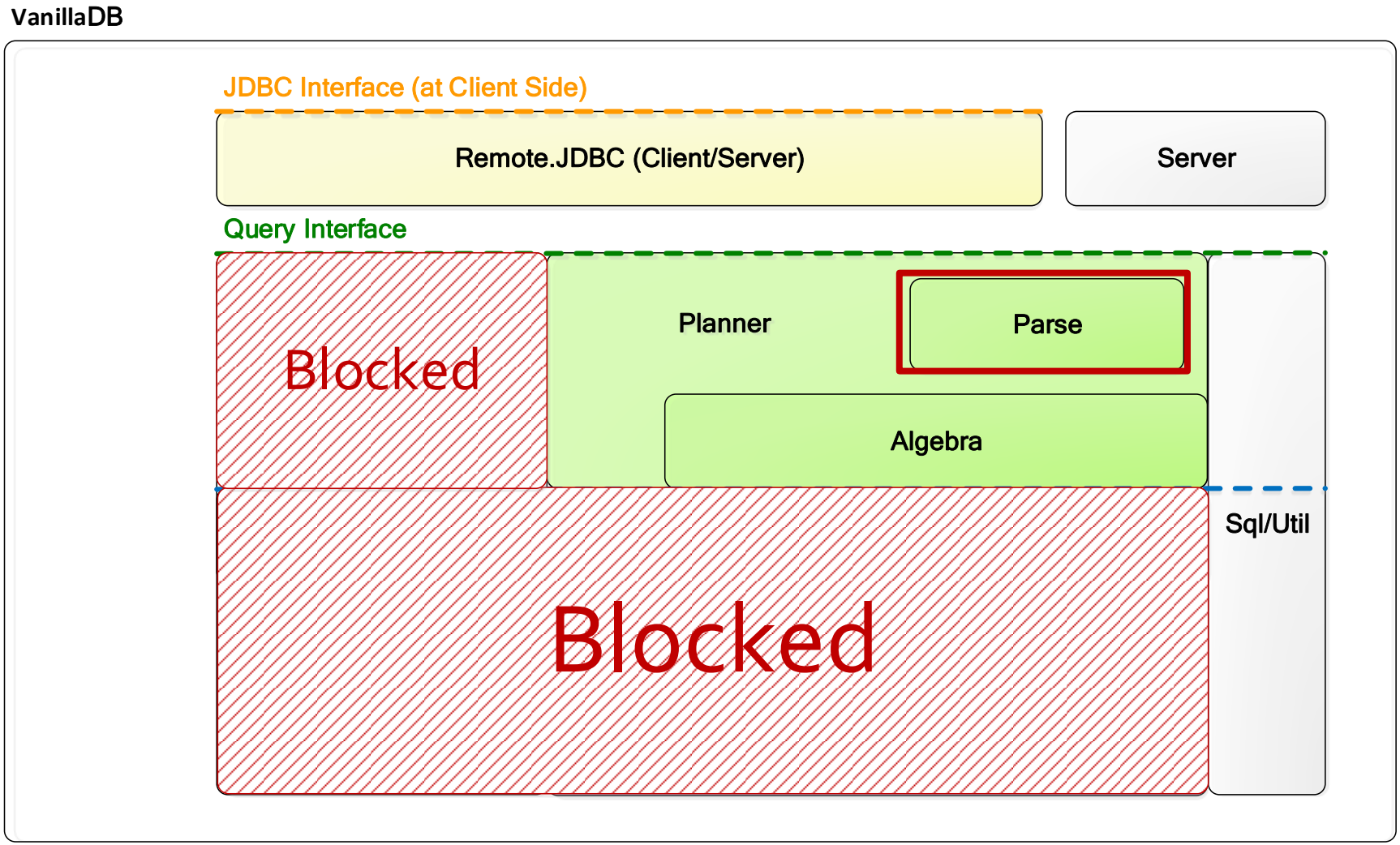
Outline

- Query package
  - Parse package
  - Algebra package
  - Planner package

Outline

- Query package
  - Parse package
  - Algebra package
  - Planner package

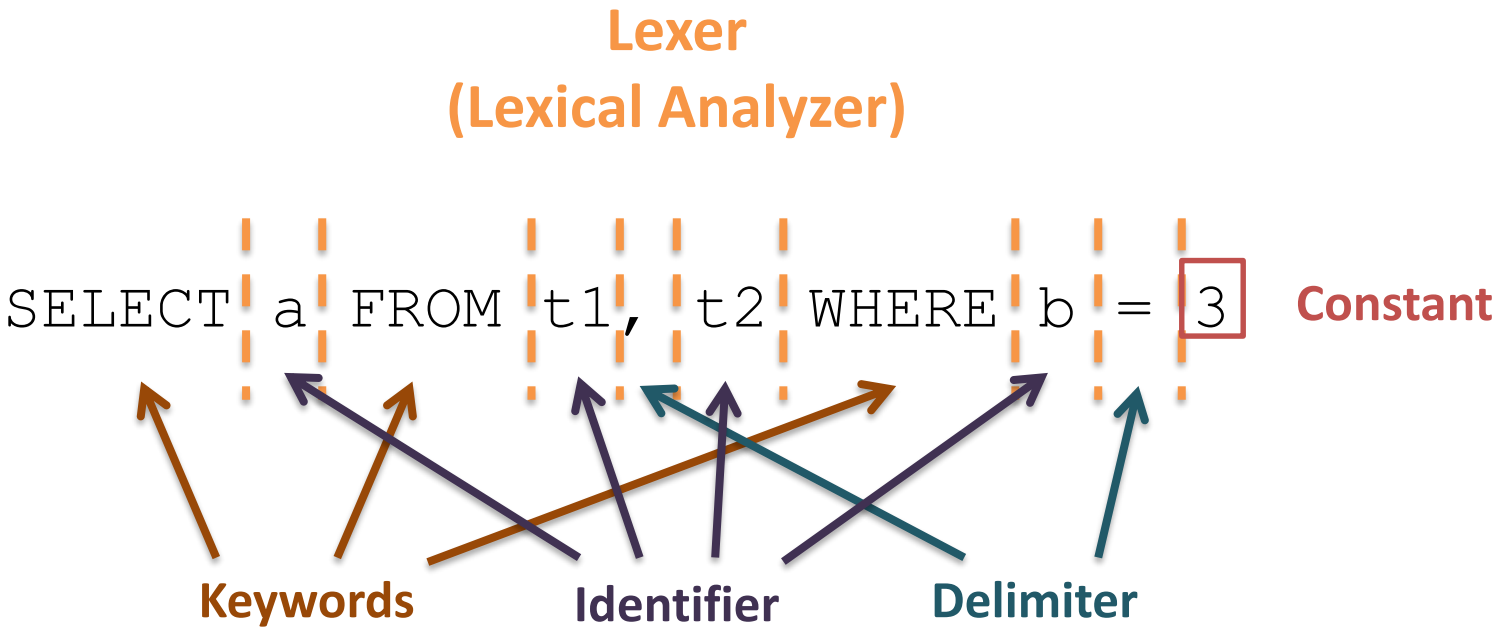
Where Are We ?


VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
Planner
Parse
Algebra
Sql/Util
Blocked

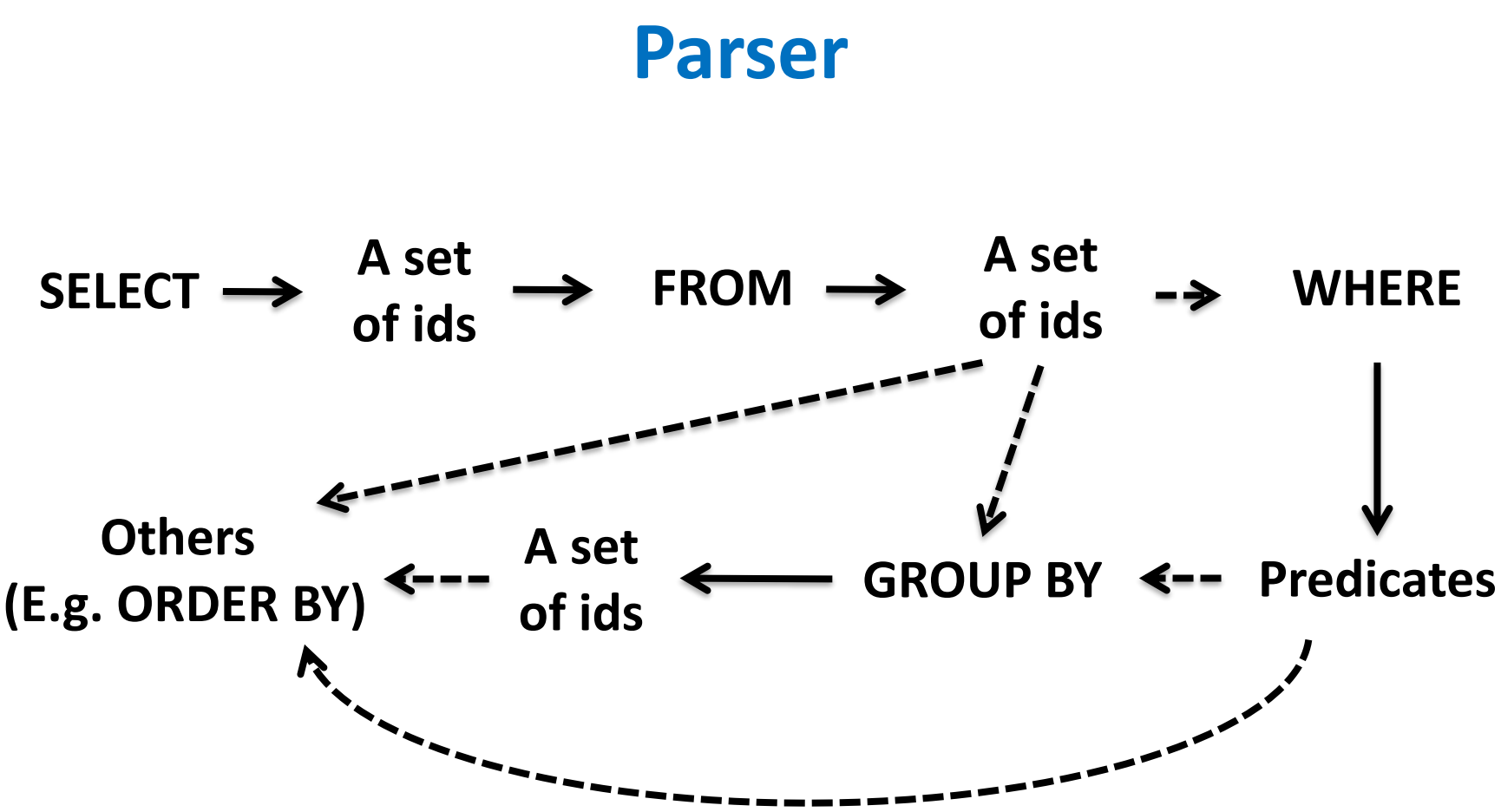
Two Steps of Parsing A SQL

- Lexer
  - Tokenizing
  - Identifying keywords, IDs, values, delimiters
- Parser
  - Checking syntax
  - Identifying the action and the parameters

How VanillaCore Parses A SQL (1/2)



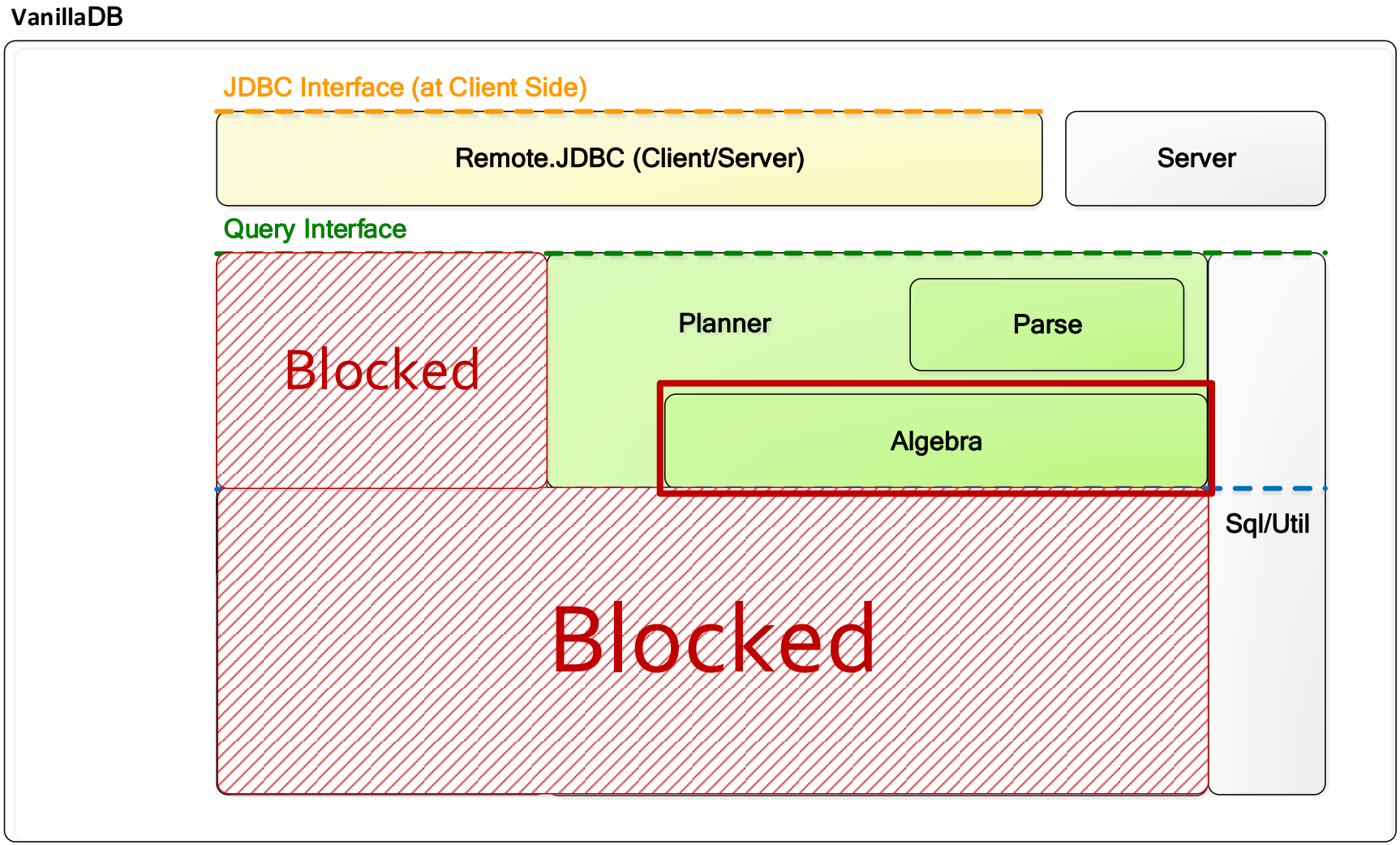
How VanillaCore Parses A SQL (2/2)



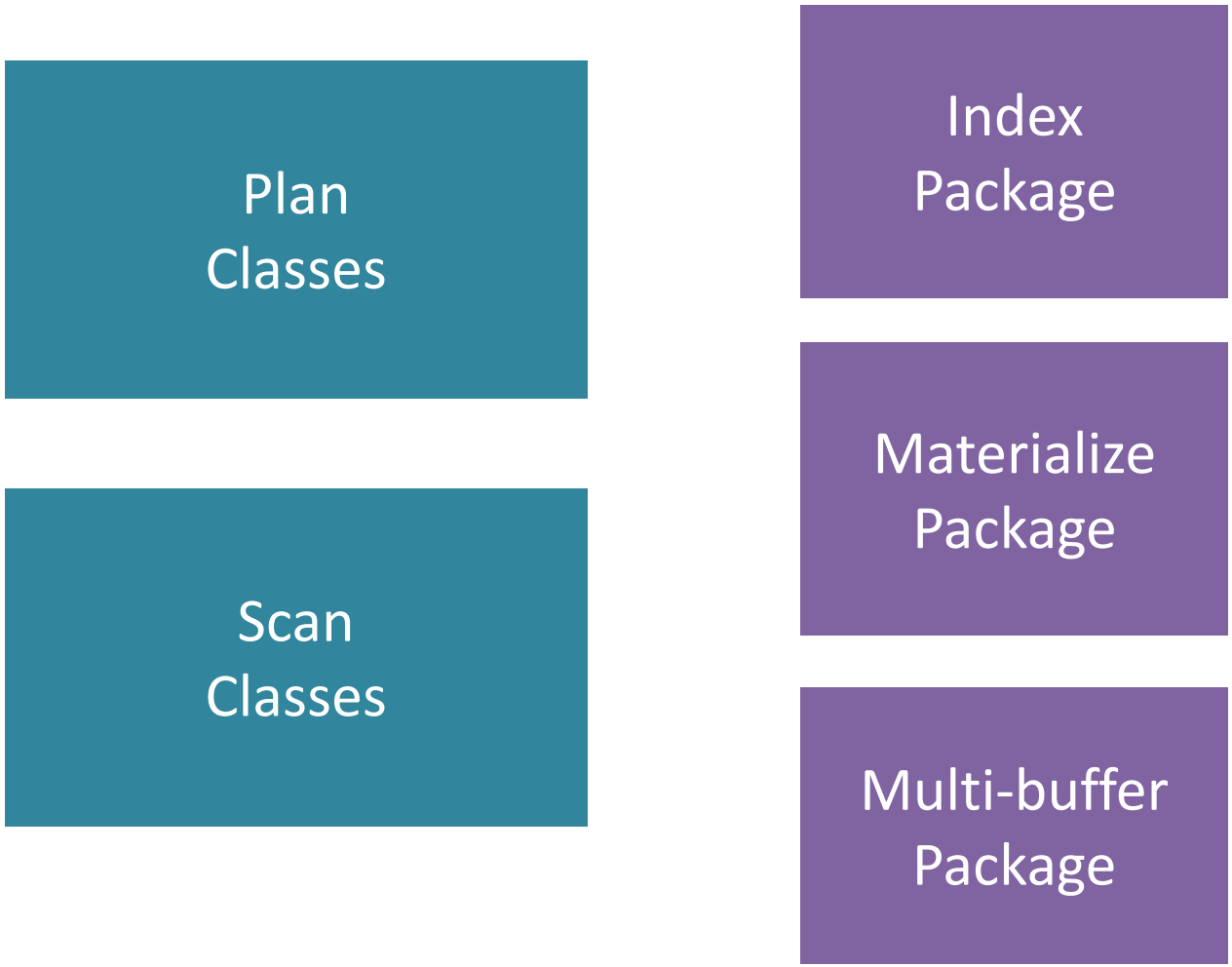
Outline

- Query package
  - Parse package
  - Algebra package
  - Planner package

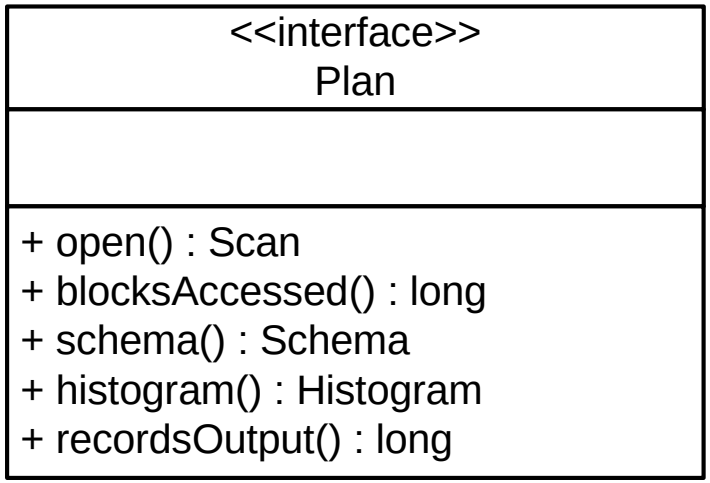
Where Are We ?

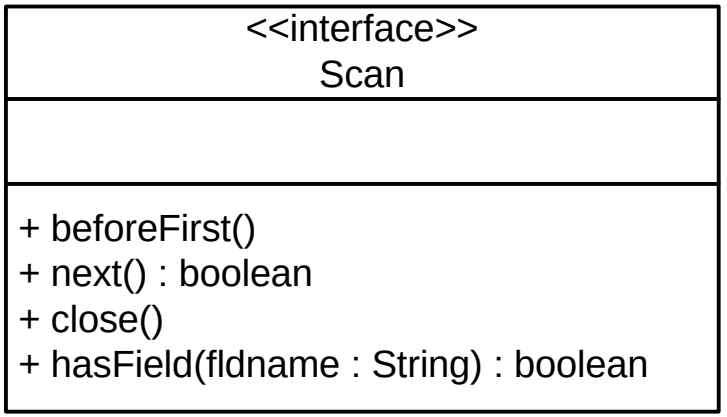

VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
Planner
Parse
Algebra
Sql/Util
Blocked

`algebra` Package



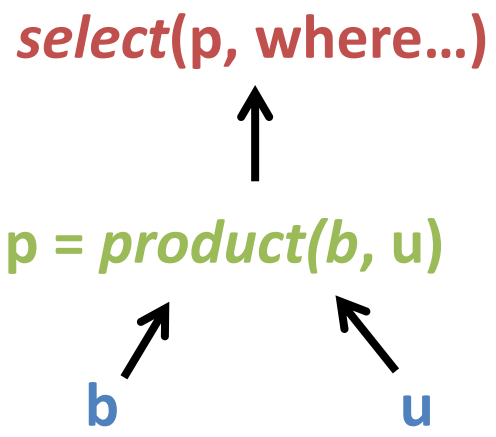
Plan & Scan


<<interface>>
Plan
+ open() : Scan
+ blocksAccessed() : long
+ schema() : Schema
+ histogram() : Histogram
+ recordsOutput() : long


<<interface>>
Scan
+ beforeFirst()
+ next() : boolean
+ close()
+ hasField(fldname : String) : boolean

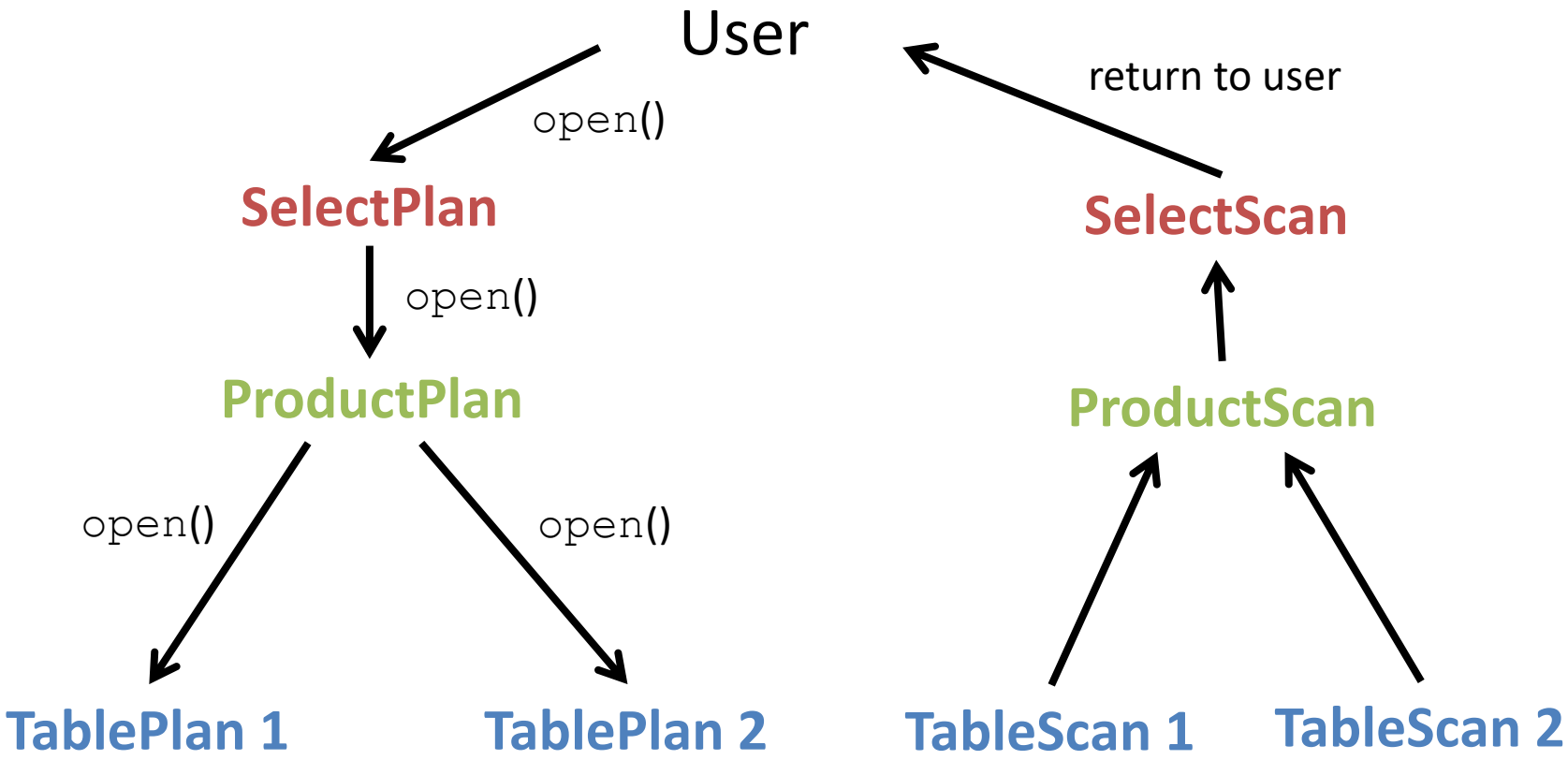
Using a Query Plan

```
VanillaDb.init("studentdb");
Transaction tx = VanillaDb.txMgr().newTransaction(
    Connection.TRANSACTION_SERIALIZABLE, true);

Plan pb = new TablePlan("b", tx);
Plan pu = new TablePlan("u", tx);
Plan pp = new ProductPlan(pb, pu);
Predicate pred = new Predicate("...");
Plan sp = new SelectPlan(pp, pred);

sp.blockAccessed(); // estimate #blocks accessed

// open corresponding scan only if sp has low cost
Scan s = sp.open();
s.beforeFirst();
while (s.next())
    s.getVal("bid");
s.close();
```



What Happened When We Called `open()`?

open()


User
open()
return to user
SelectPlan
SelectScan
open()
ProductPlan
ProductScan
open()
open()
TablePlan 1
TablePlan 2
TableScan 1
TableScan 2

How Do Scans Work ?

Example

project (s)

↓ `beforeFirst()`

select (p)

↓ `beforeFirst()`

product (b, u)

↘ `beforeFirst()`

```
SELECT blog_id FROM b, u
        WHERE name = "Brandon"
        AND author_id = user_id;
```

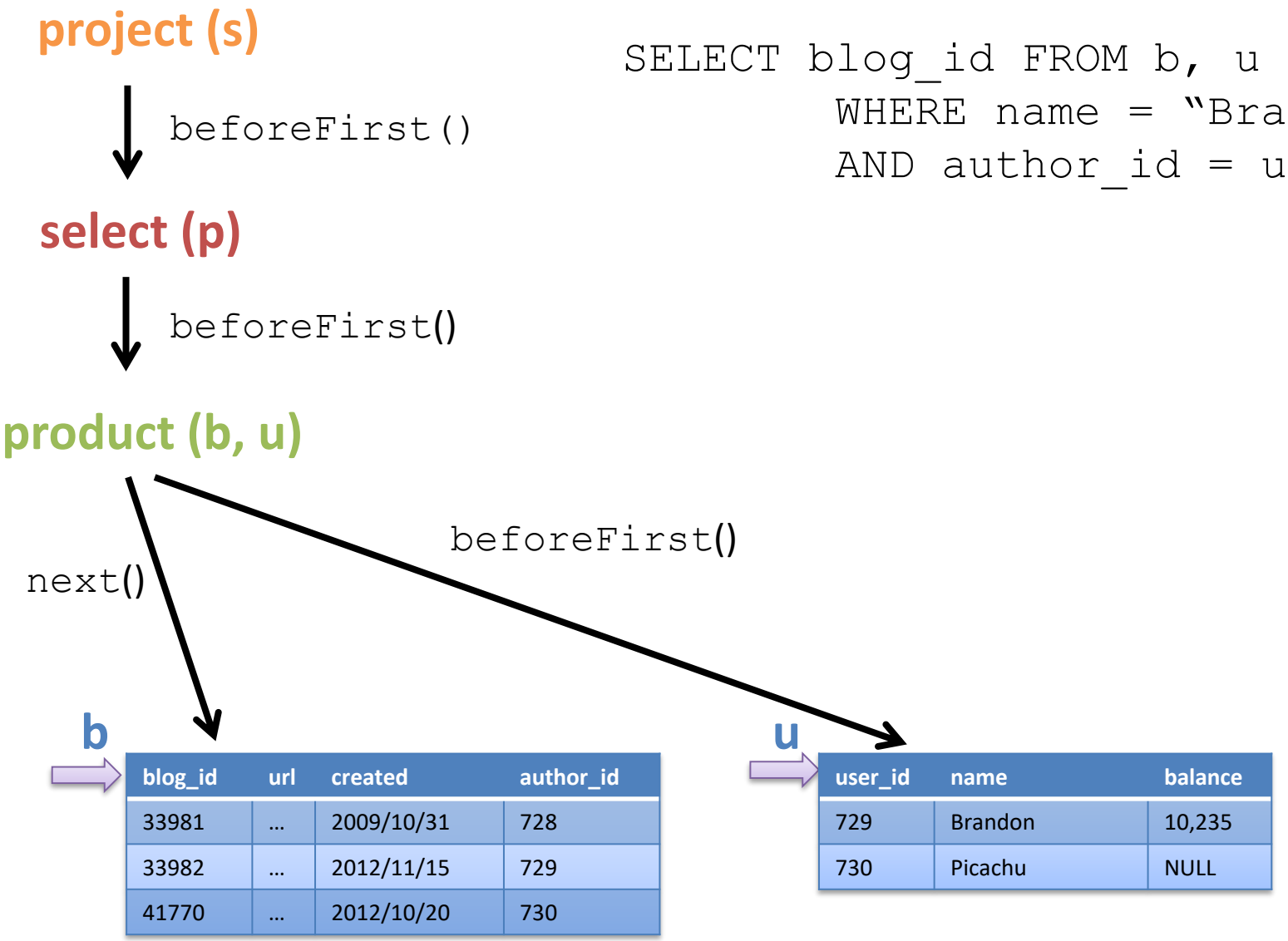
b

blog_id	url	created	author_id
33981	…	2009/10/31	728
33982	…	2012/11/15	729
41770	…	2012/10/20	730

u

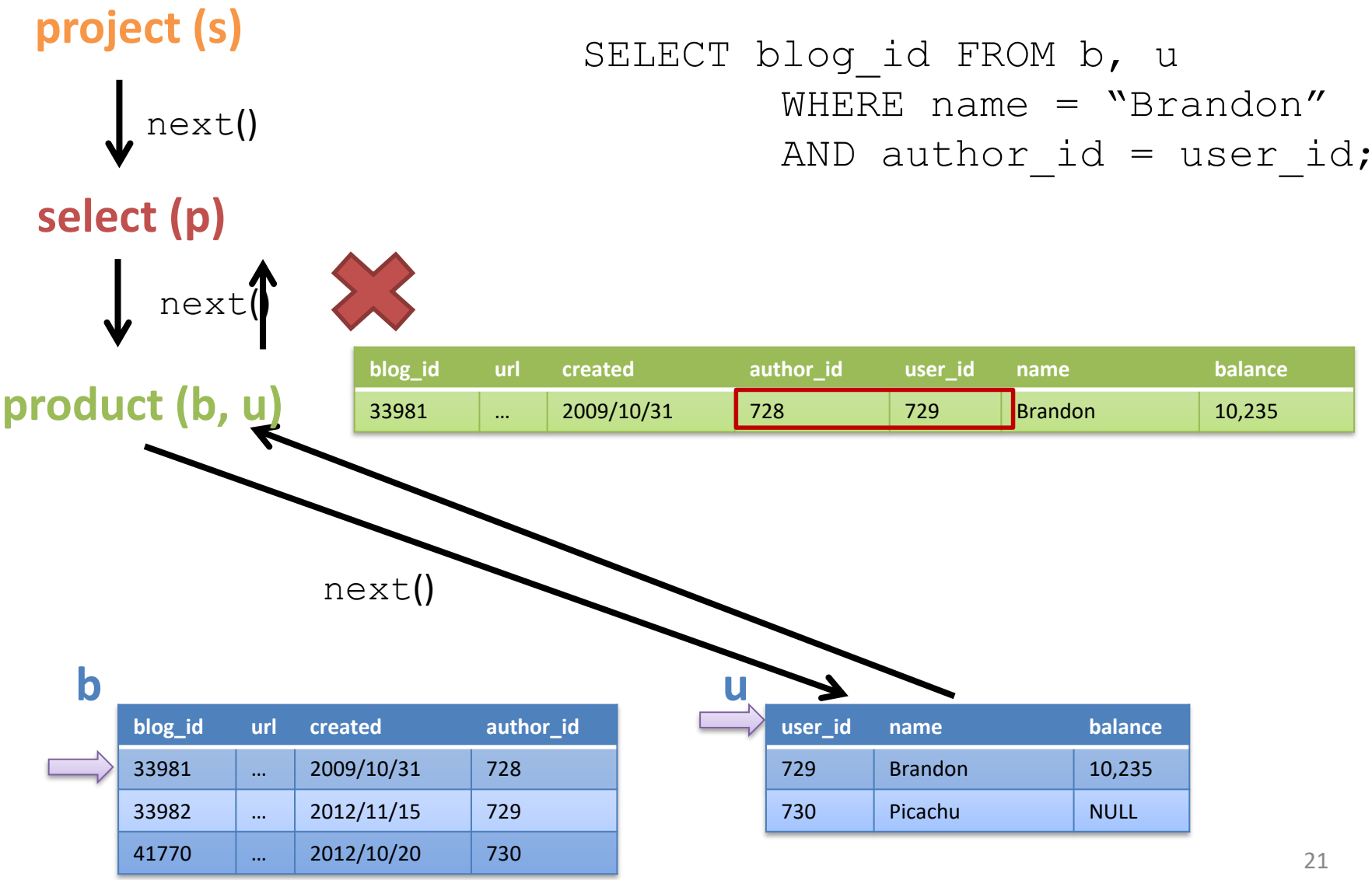
user_id	name	balance
729	Brandon	10,235
730	Picachu	NULL

Example




```
SELECT blog_id FROM b, u
        WHERE name = "Brandon"
        AND author_id = user_id;
```

Example




```
SELECT blog_id FROM b, u
        WHERE name = "Brandon"
        AND author_id = user_id;
```

blog_id	url	created	author_id	user_id	name	balance
33981	…	2009/10/31	728	729	Brandon	10,235

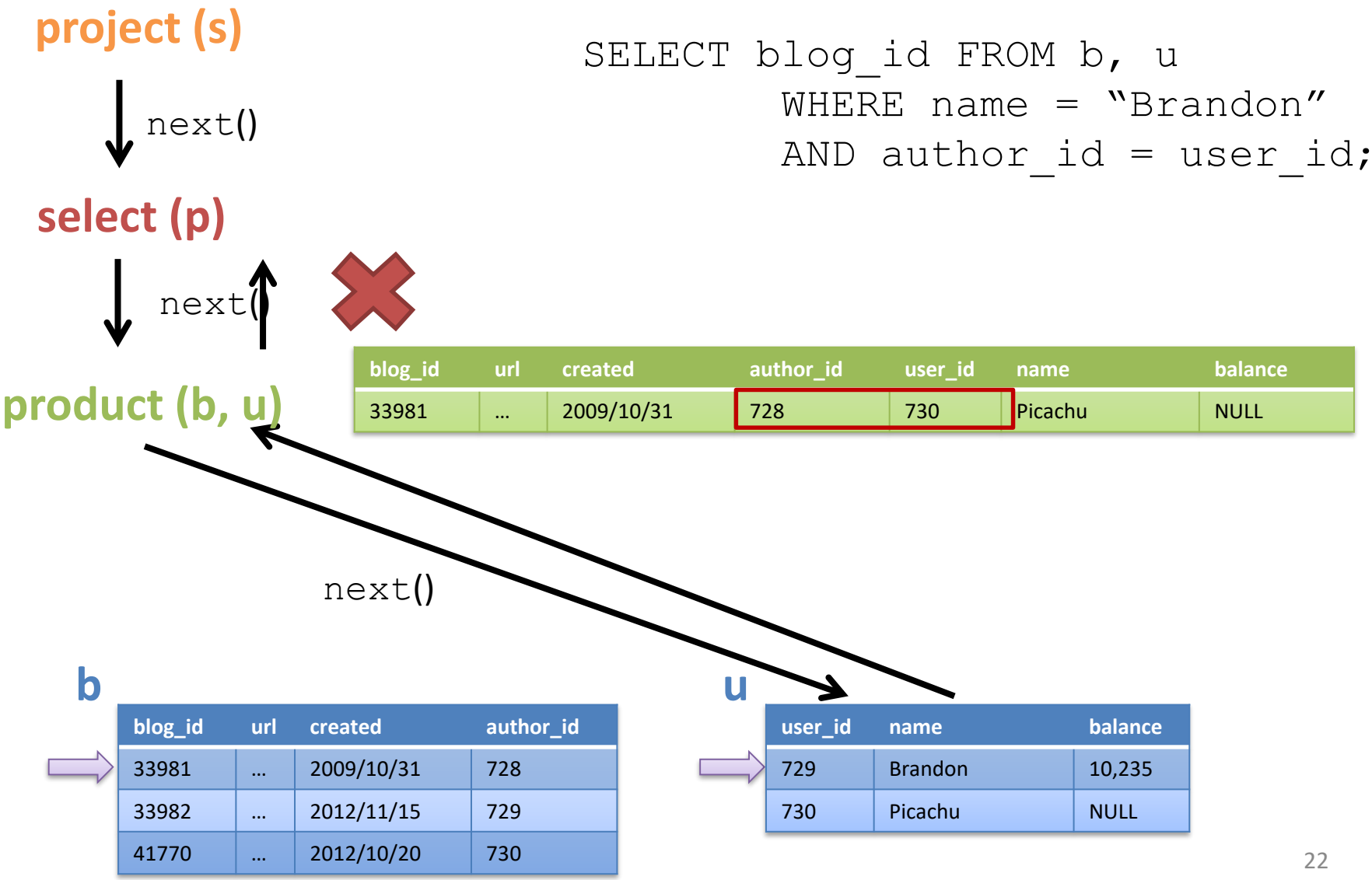
b

blog_id	url	created	author_id
33981	…	2009/10/31	728
33982	…	2012/11/15	729
41770	…	2012/10/20	730

u

user_id	name	balance
729	Brandon	10,235
730	Picachu	NULL

Example



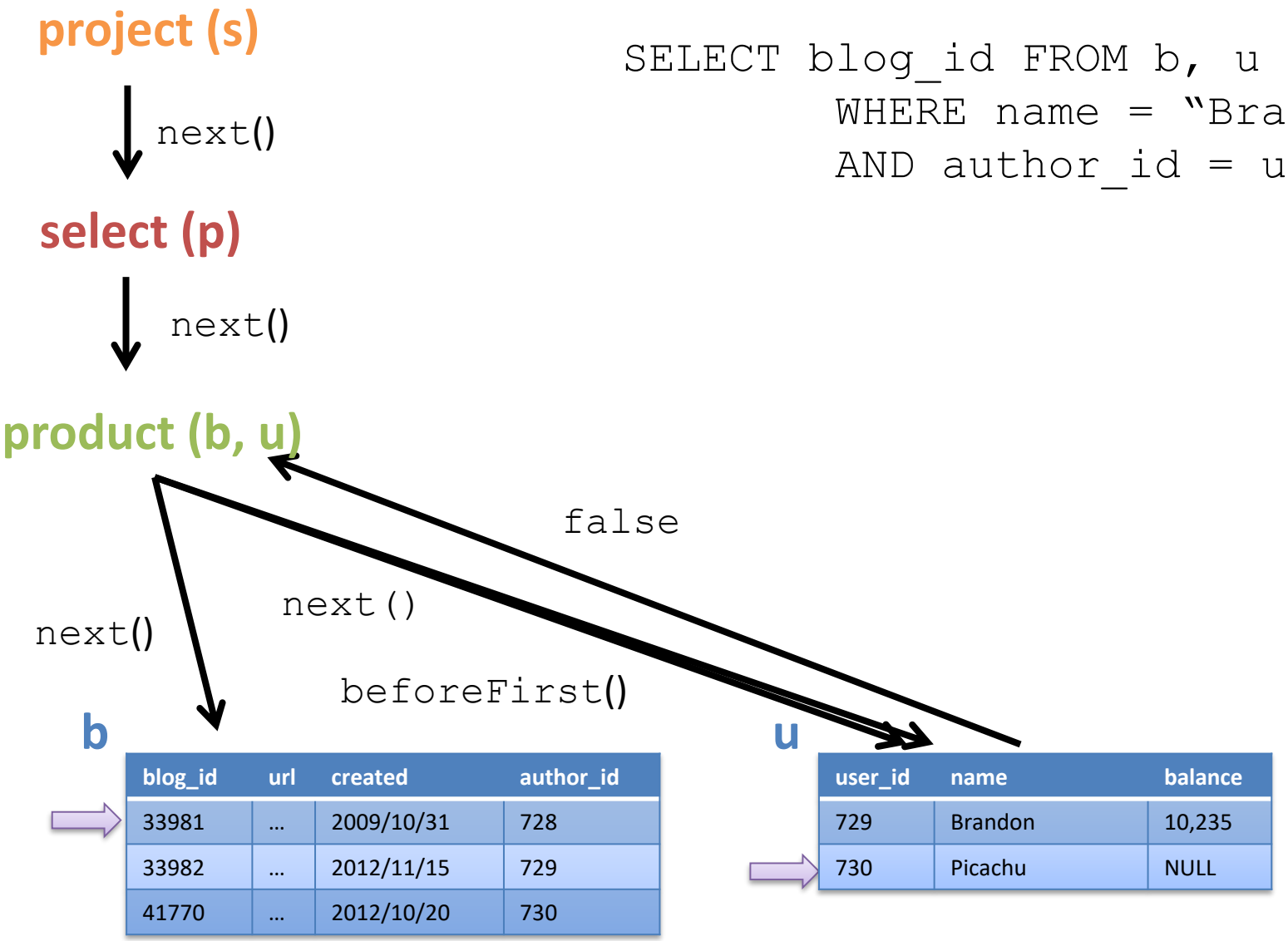

```
SELECT blog_id FROM b, u
        WHERE name = "Brandon"
        AND author_id = user_id;
```

blog_id	url	created	author_id	user_id	name	balance
33981	…	2009/10/31	728	730	Picachu	NULL

blog_id	url	created	author_id
33981	…	2009/10/31	728
33982	…	2012/11/15	729
41770	…	2012/10/20	730

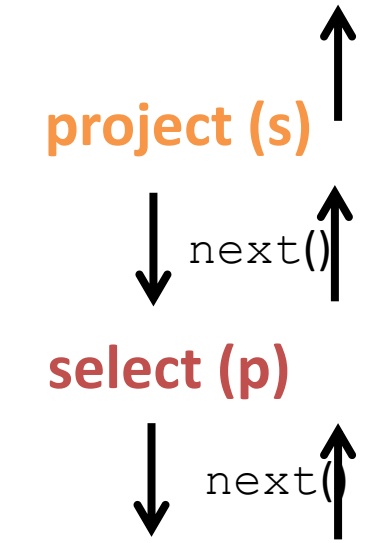
user_id	name	balance
729	Brandon	10,235
730	Picachu	NULL

Example




```
SELECT blog_id FROM b, u
        WHERE name = "Brandon"
        AND author_id = user_id;
```

Example



blog_id
33982

```
SELECT blog_id FROM b, u
```

blog_id	url	created	author_id	user_id	name	balance
33982	…	2012/11/15	730	730	Picachu	NULL

blog_id	url	created	author_id	user_id	name	balance
33982	…	2012/11/15	729	729	Brandon	10,235

product (b, u)

next()

b

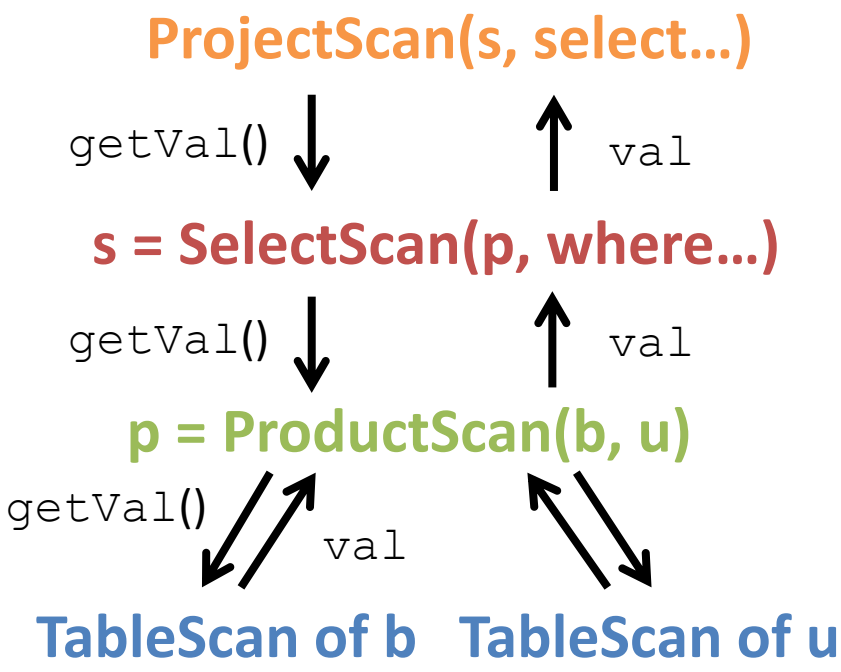
blog_id	url	created	author_id
33981	…	2009/10/31	728
33982	…	2012/11/15	729
41770	…	2012/10/20	730

u

user_id	name	balance
729	Brandon	10,235
730	Picachu	NULL

Pipelined Scanning

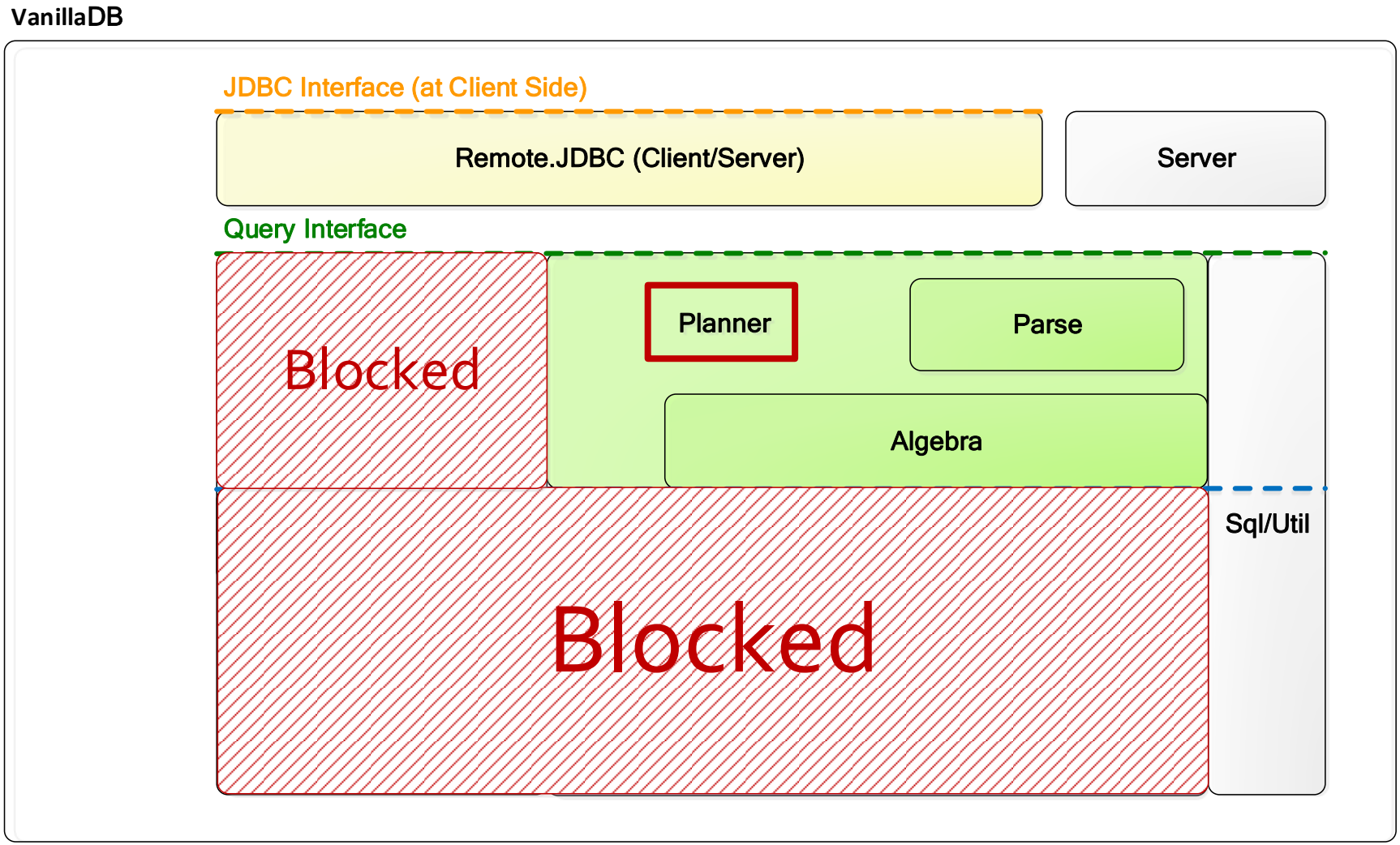
- The above operators implement ***pipelined scanning***
  - Calling a method of a node results in recursively calling the same methods of child nodes on-the-fly
  - Records are computed one at a time as needed --- no intermediate records are saved



Outline

- Query package
  - Parse package
  - Algebra package
  - Planner package

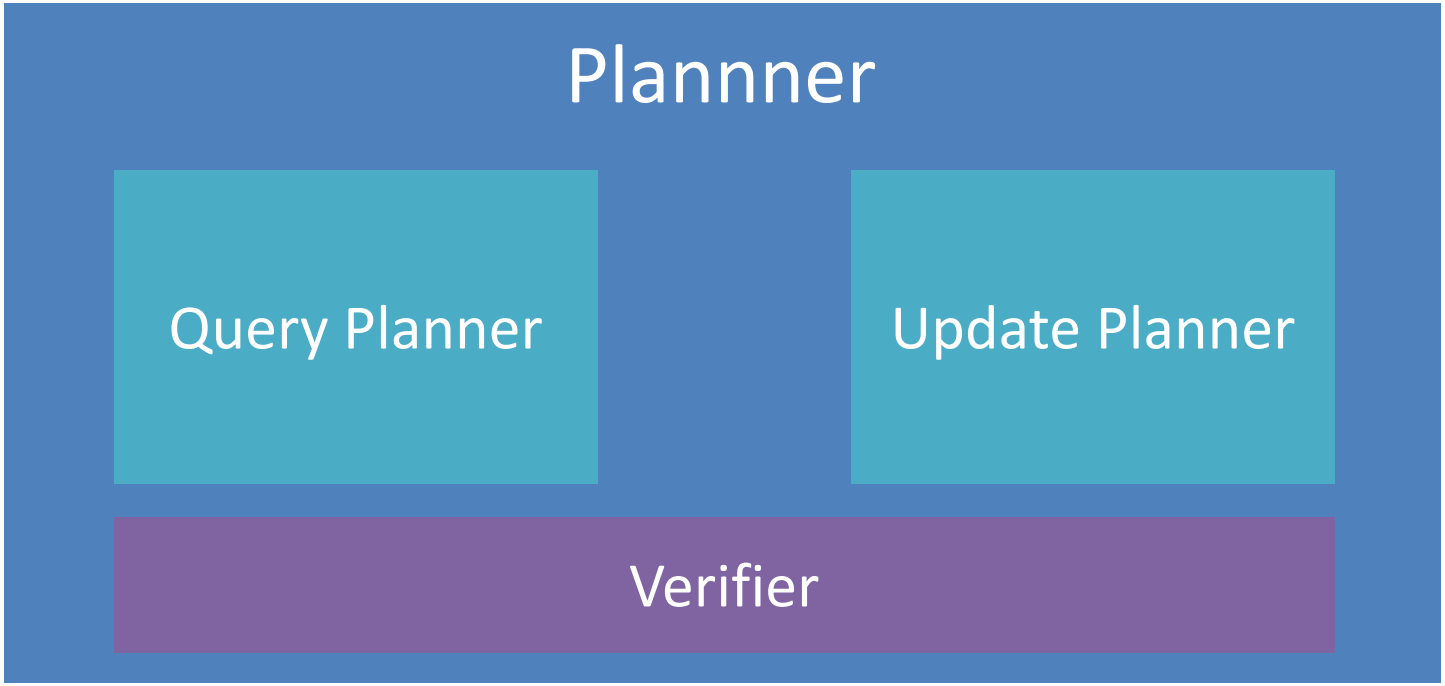
Where Are We ?


VanillaDB
JDBC Interface (at Client Side)
Remote.JDBC (Client/Server)
Server
Query Interface
Blocked
Planner
Parse
Algebra
Sql/Util
Blocked

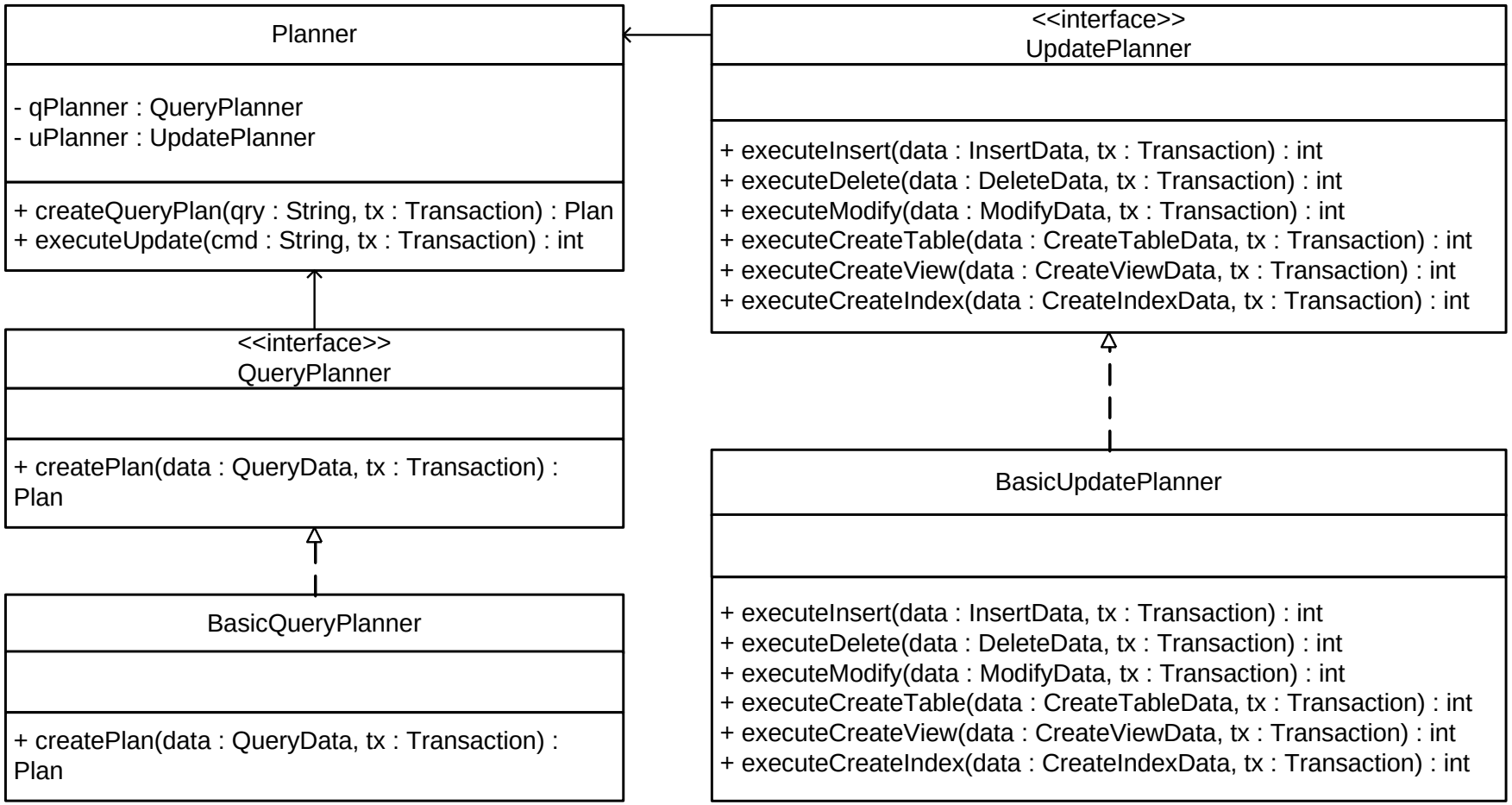
Planner

- The one puts all these together
  1. Accepts a query
  2. Creates a parser to parse the query
  3. Verifies all parameters are reasonable
  4. Generates some plan trees according to the query
  5. Choose the cheapest one

`planner` Pcakage



Basic Implementation



Advanced Implementation