

VanillaCore Walkthrough

Part 2

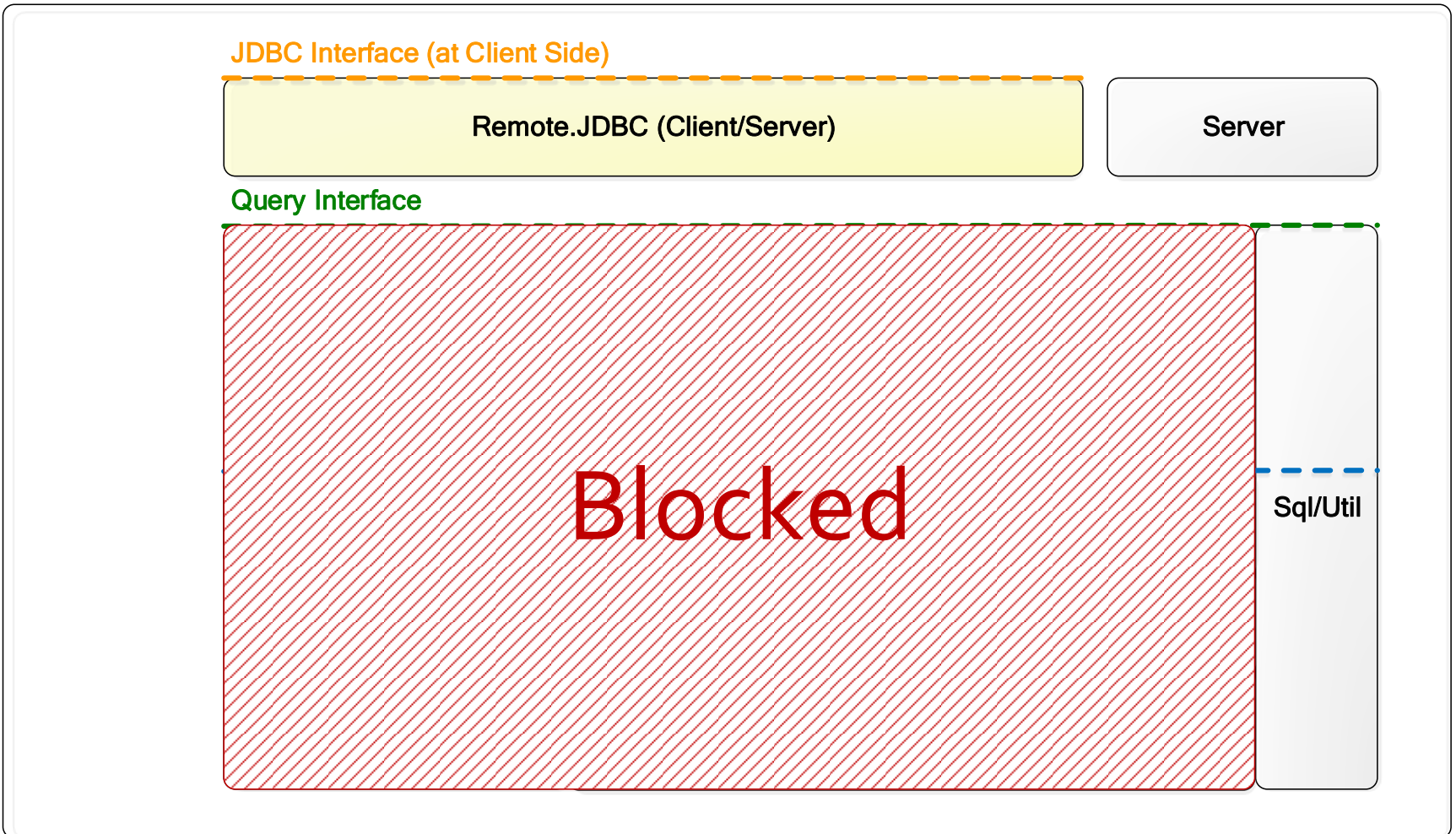
Cloud Databases

DataLab

CS, NTHU

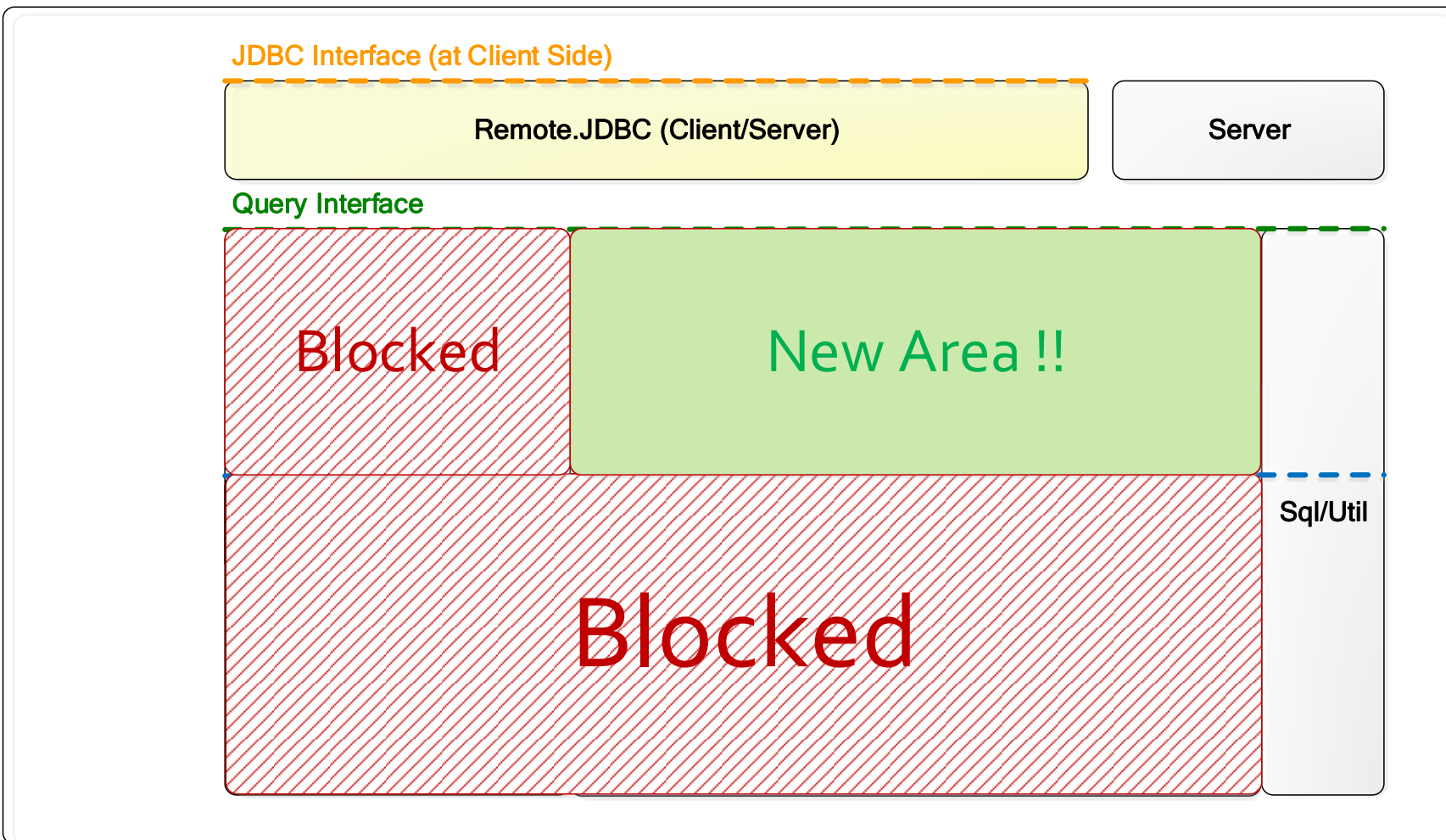
Last Time

VanillaDB



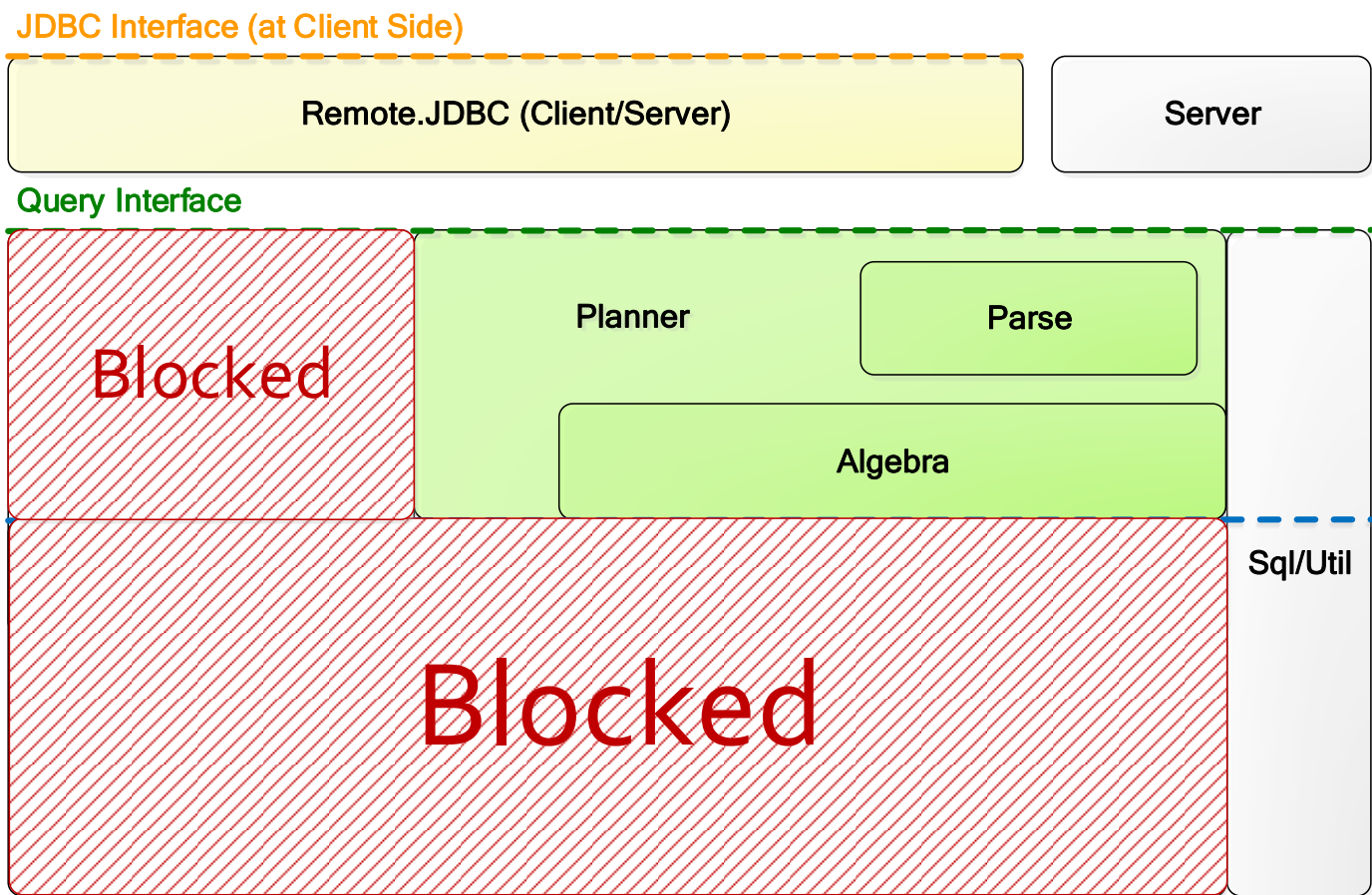
This Time

VanillaDB



This Time

VanillaDB



Outline

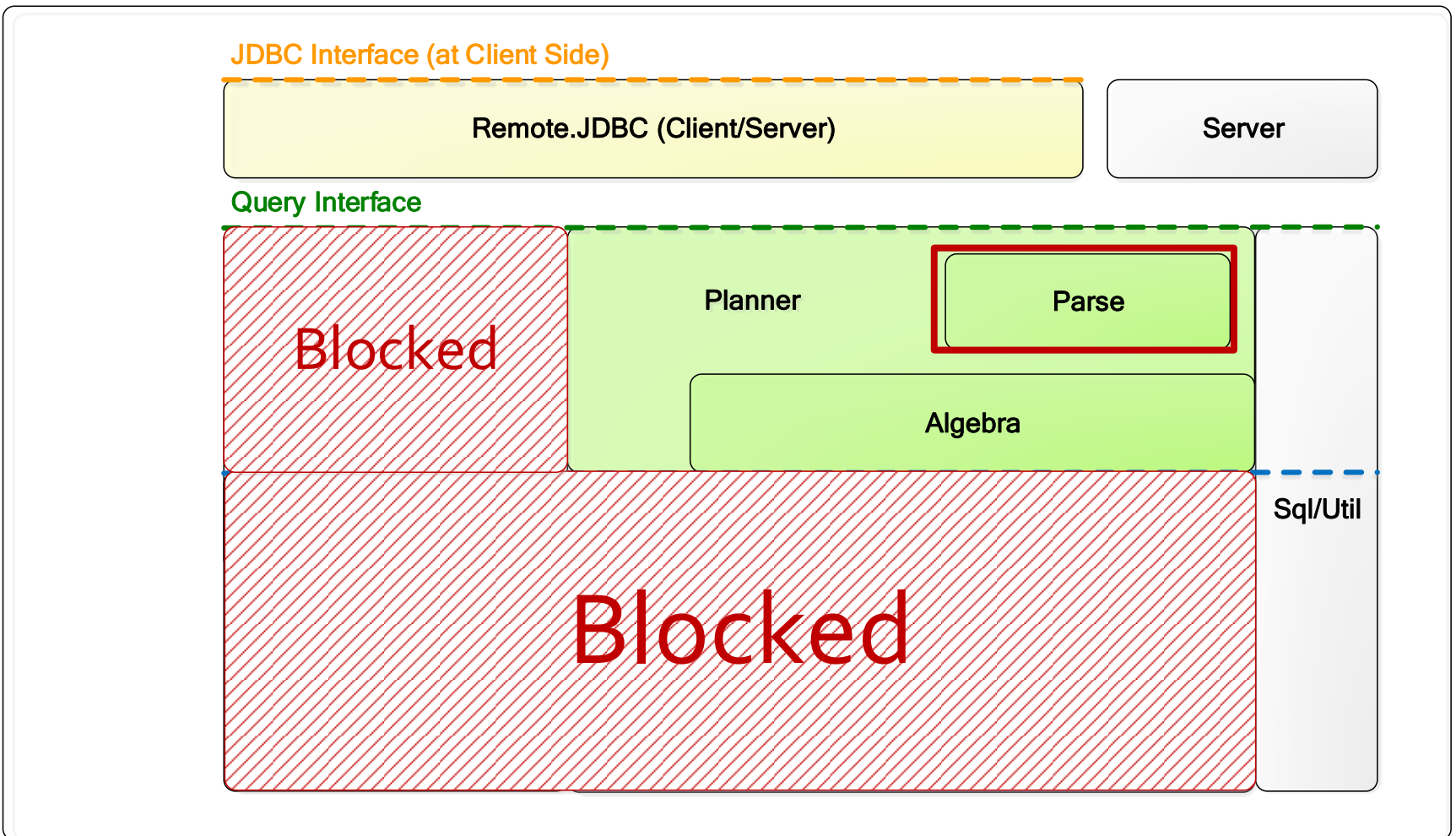
- Query package
 - Parse package
 - Algebra package
 - Planner package

Outline

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

Where Are We ?

VanillaDB

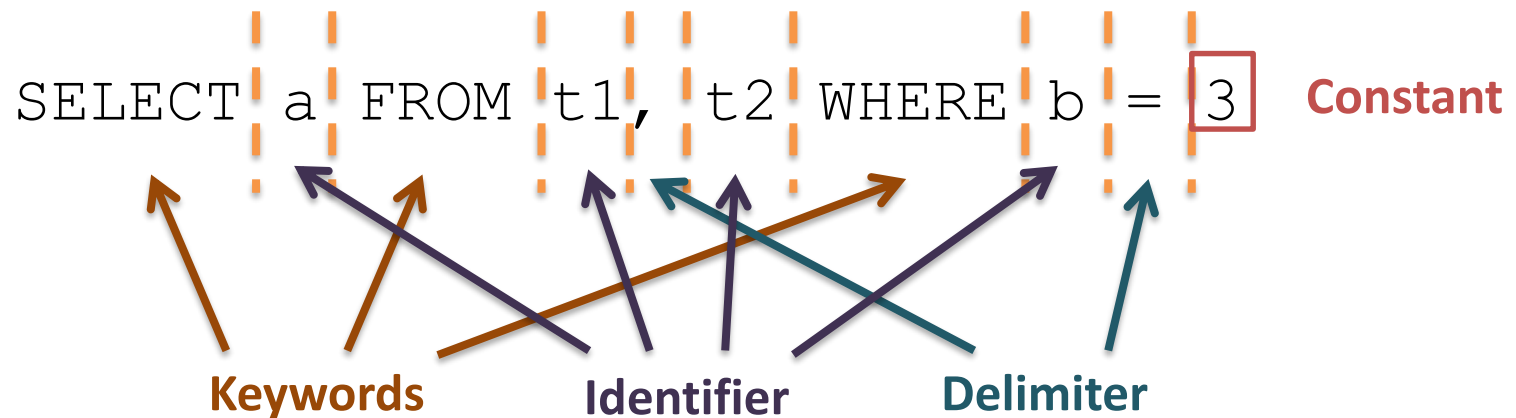


Two Steps of Parsing A SQL

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

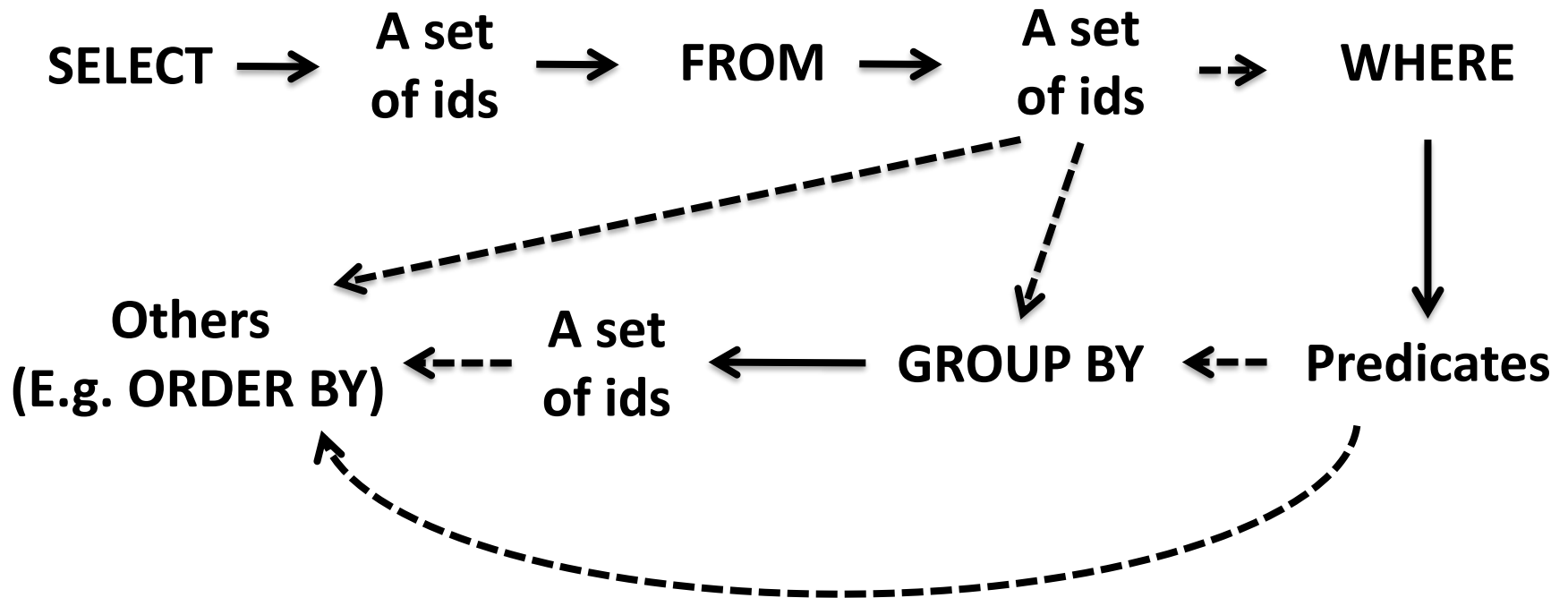
How VanillaCore Parses A SQL (1/2)

Lexer (Lexical Analyzer)



How VanillaCore Parses A SQL (2/2)

Paser

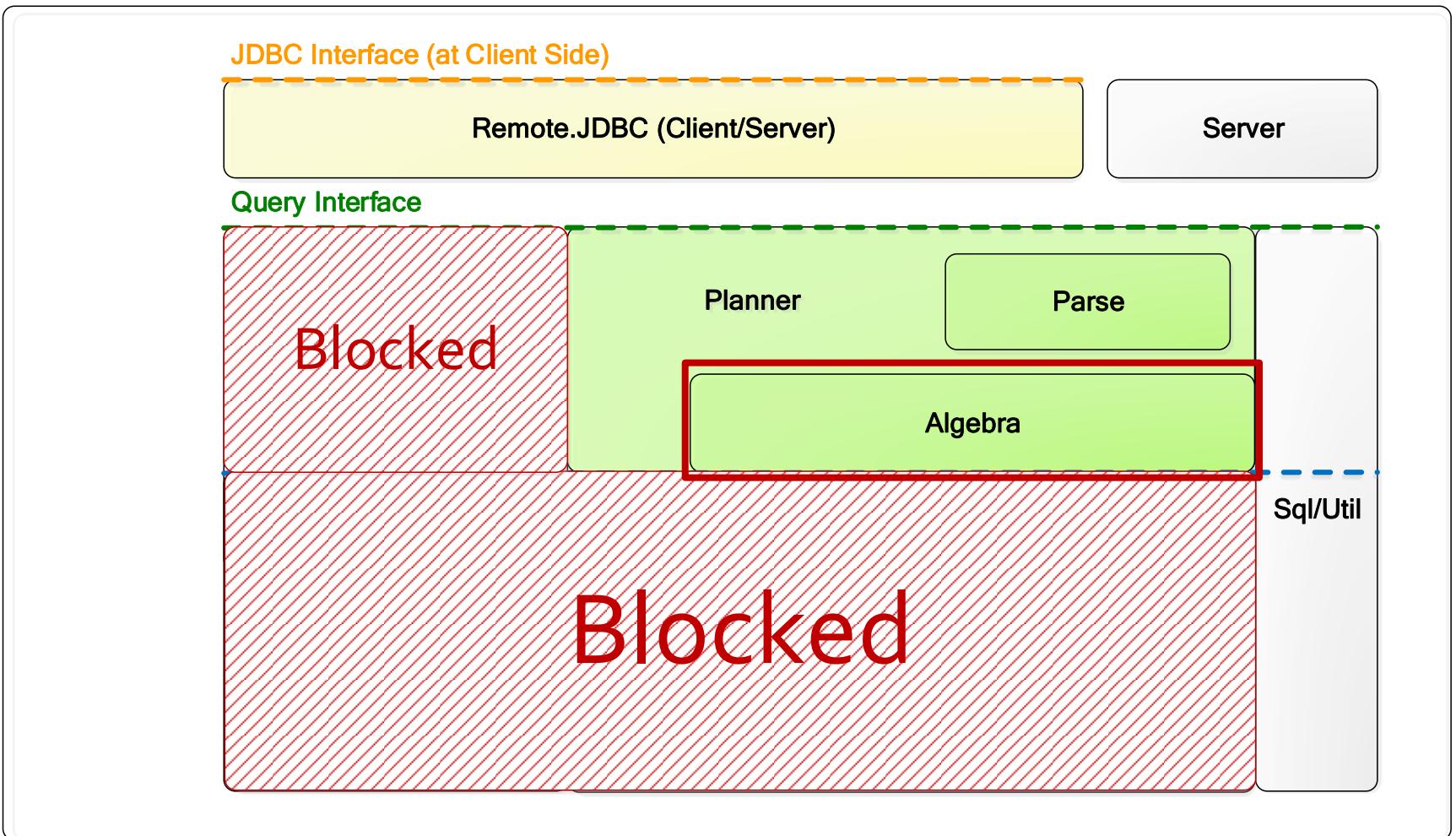


Outline

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

Where Are We ?

VanillaDB



algebra Package

Plan
Classes

Scan
Classes

Index
Package

Materialize
Package

Multi-buffer
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

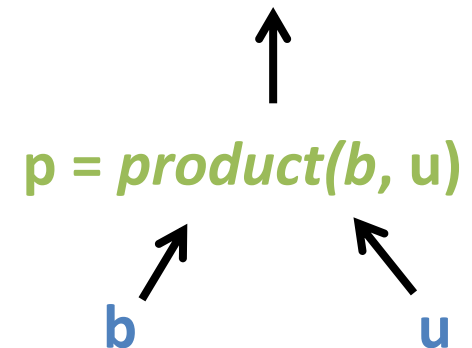
```
VanillaDb.init("studentdb");  
Transaction tx = VanillaDb.txMgr().newTransaction(  
    Connection.TRANSACTION_SERIALIZABLE, true);
```

```
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
```

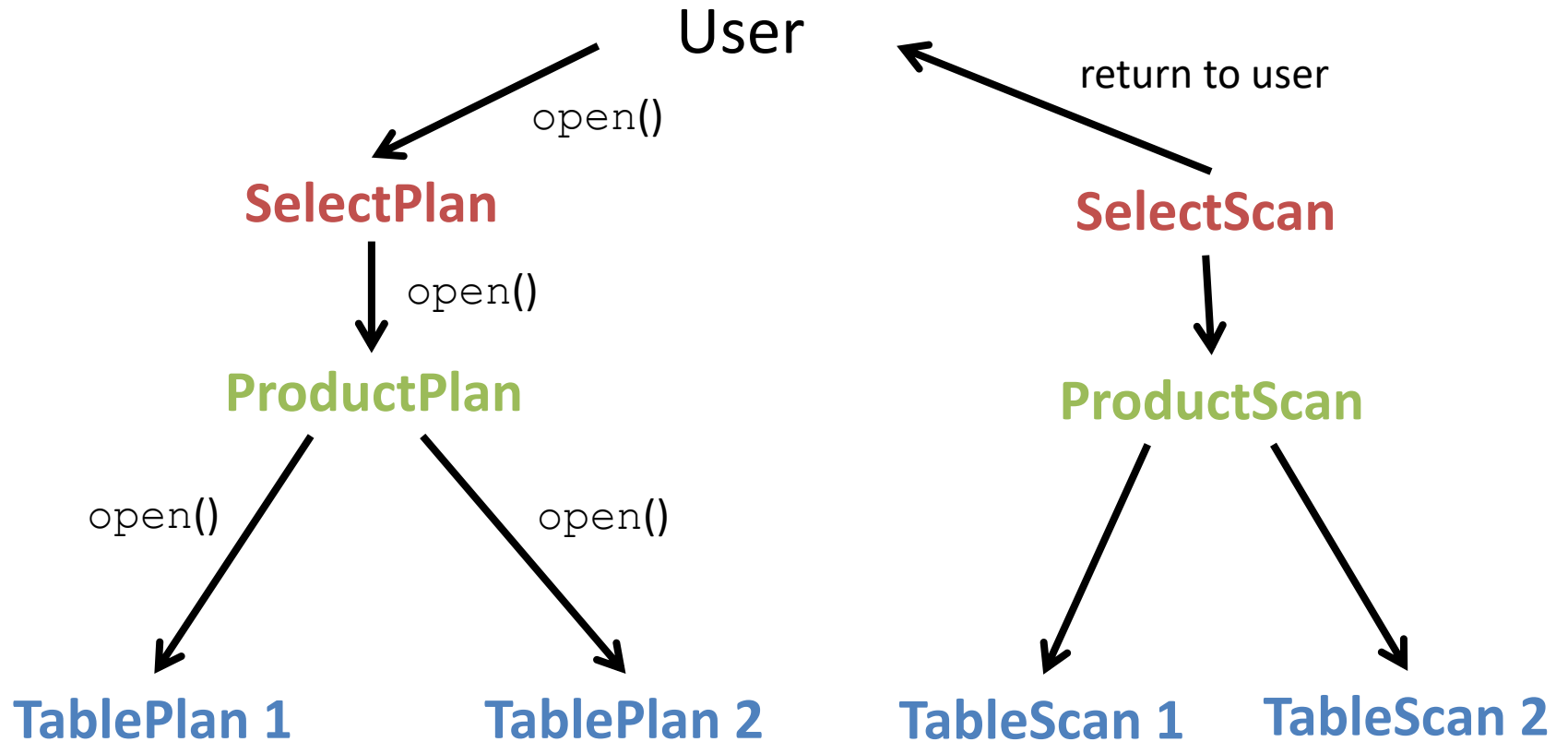
```
// open corresponding scan only if sp has low cost  
Scan s = sp.open();  
s.beforeFirst();  
while (s.next())  
    s.getVal("bid");  
s.close();
```

select(p, where...)



What Happened When We Called `open ()` ?

open ()



How Do Scans Work ?

Example

project(s, select blog_id)



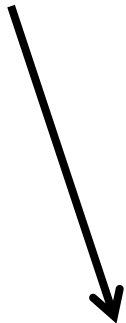
beforeFirst()

**select(p, where name = 'Picachu'
and author_id = user_id)**



beforeFirst()

product(b, u)



beforeFirst()

b

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

u

user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

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

Example

project(s, select blog_id)



beforeFirst()

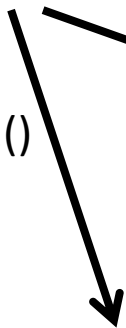
**select(p, where name = 'Picachu'
and author_id = user_id)**



beforeFirst()

product(b, u)


next()



beforeFirst()




b



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

u

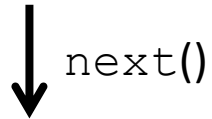


user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

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

Example

project(s, select blog_id)



select(p, where name = 'Picachu'
and author_id = user_id)



product(b, u)

blog_id	url	created	author_id	user_id	name	balance
33981	...	2009/10/31	729	729	Steven Sinofsky	10,235

next()

b

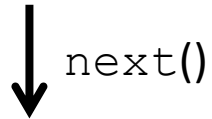
blog_id	url	created	author_id
33981	...	2009/10/31	729
33982	...	2012/11/15	730
41770	...	2012/10/20	729

u

user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

Example

project(s, select blog_id)



select(p, where name = 'Picachu'
and author_id = user_id)



product(b, u)

blog_id	url	created	author_id	user_id	name	balance
33981	...	2009/10/31	729	730	Picachu	NULL

next()

b

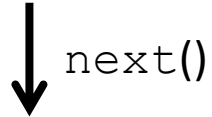
blog_id	url	created	author_id
33981	...	2009/10/31	729
33982	...	2012/11/15	730
41770	...	2012/10/20	729

u

user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

Example

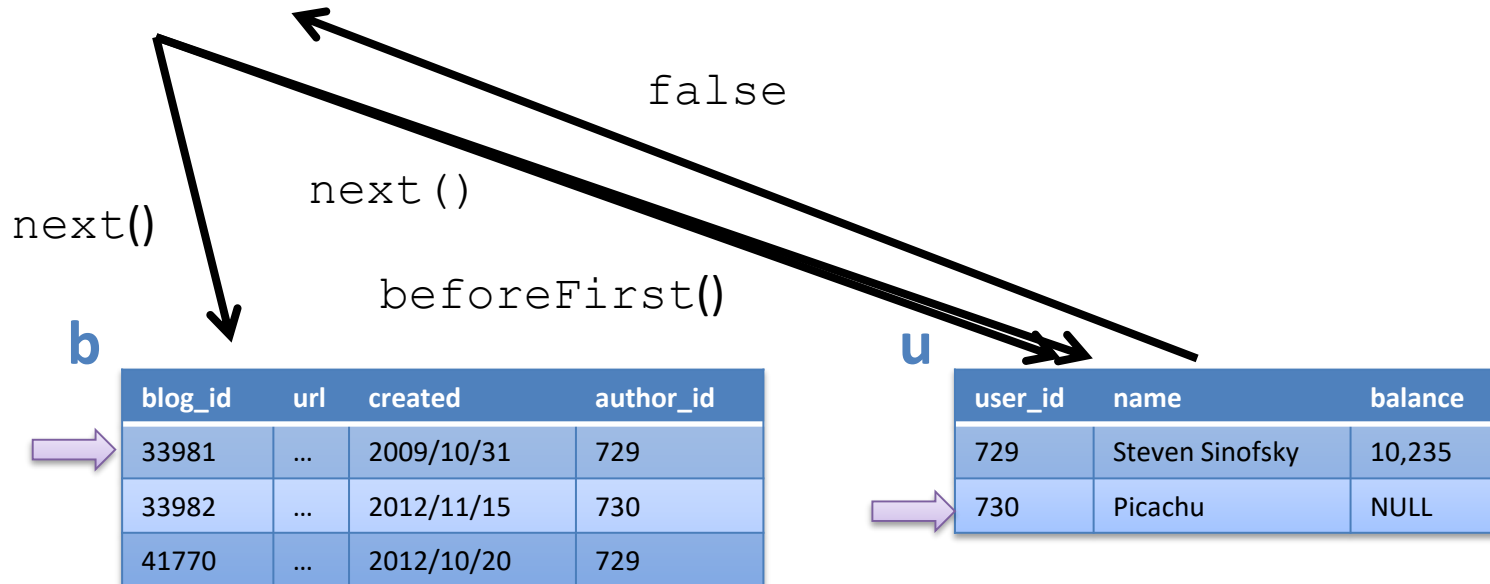
project(s, select blog_id)



**select(p, where name = 'Picachu'
and author_id = user_id)**

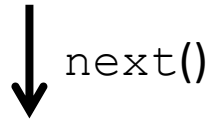


product(b, u)



Example

project(s, select blog_id)



**select(p, where name = 'Picachu'
and author_id = user_id)**



product(b, u)

blog_id	url	created	author_id	user_id	name	balance
33982	...	2012/11/15	730	729	Steven Sinofsky	10,235

next()

b

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

u

user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

Example

project(s, select blog_id)

blog_id
33982

next()

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

**select(p, where name = 'Picachu'
and author_id = user_id)**

next()



product(b, u)

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

next()

b

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

u

user_id	name	balance
729	Steven Sinofsky	10,235
730	Picachu	NULL

Example

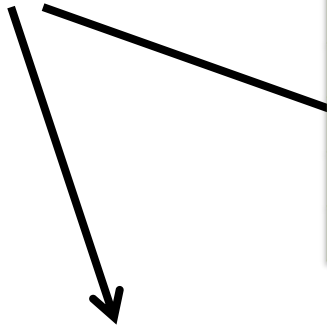
project(s, select...)



select(p, where
name = 'Picachu')



product(b, u)



blog_id
33982



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
33981	...	2009/10/31	729	729	Steven Sinofsky	10,235
33981	...	2009/10/31	729	730	Picachu	NULL
33982	...	2012/11/15	730	729	Steven Sinofsky	10,235
33982	...	2012/11/15	730	730	Picachu	NULL
41770	...	2012/10/20	729	729	Steven Sinofsky	10,235
41770	...	2012/10/20	729	730	Picachu	NULL

b

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



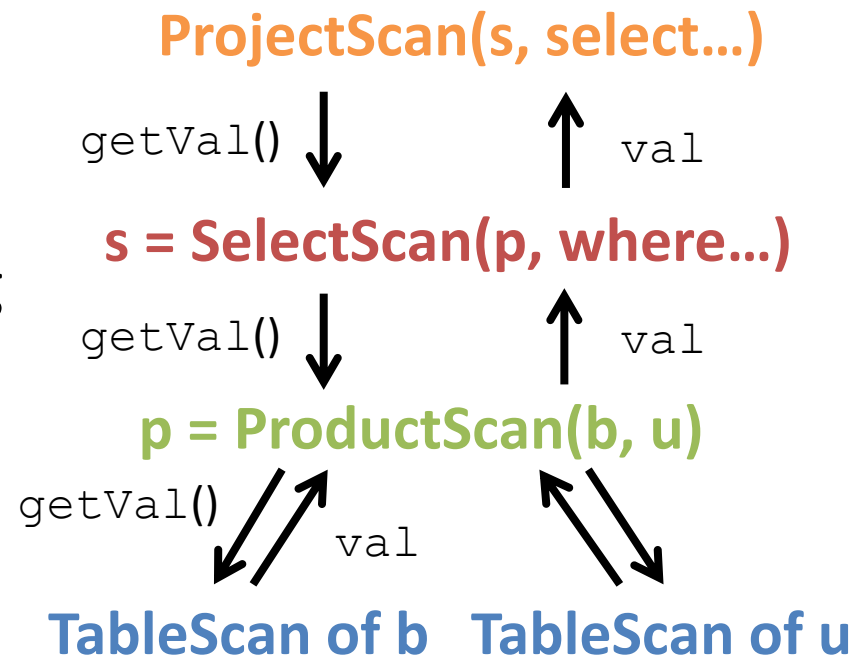
u

user_id	name	balance
729	Steven Sinofsky	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



Pipelined vs. Materialized

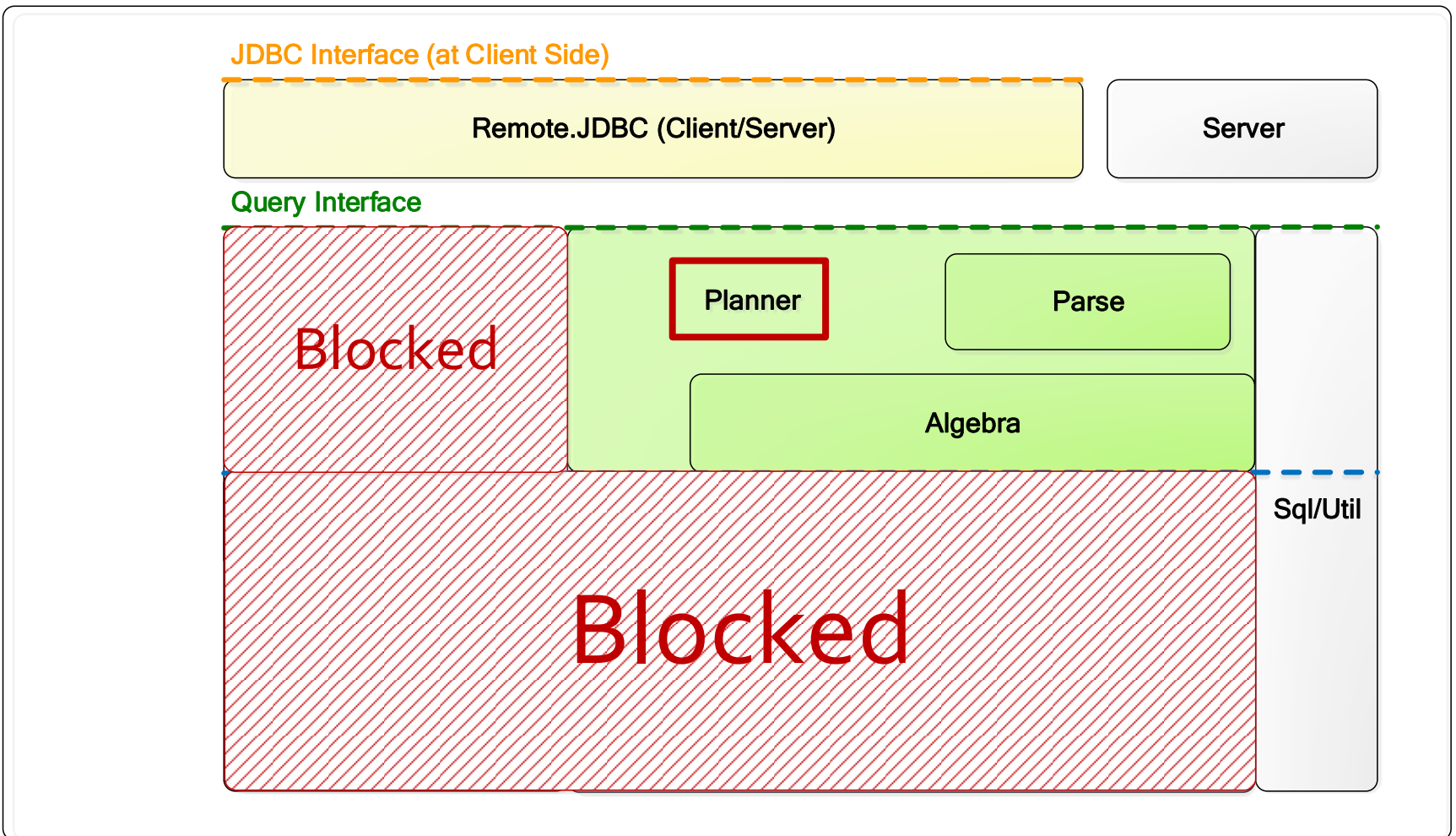
- Despite its simplicity, pipelined scanning is inefficient in some cases
 - E.g., when implementing `SortScan` (for `ORDER BY`)
 - It needs to iterate all children to find the next record
- For such cases, we use ***materialized scanning***
 - Intermediate records are materialized to a temp table (file)
 - E.g., the `SortScan` can use an external sorting algorithm to sort all records at once, save them, and return each record upon `next()` is called
- Pipelined or materialized?
 - Saving in scanning cost vs. materialization overhead

Outline

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

Where Are We ?

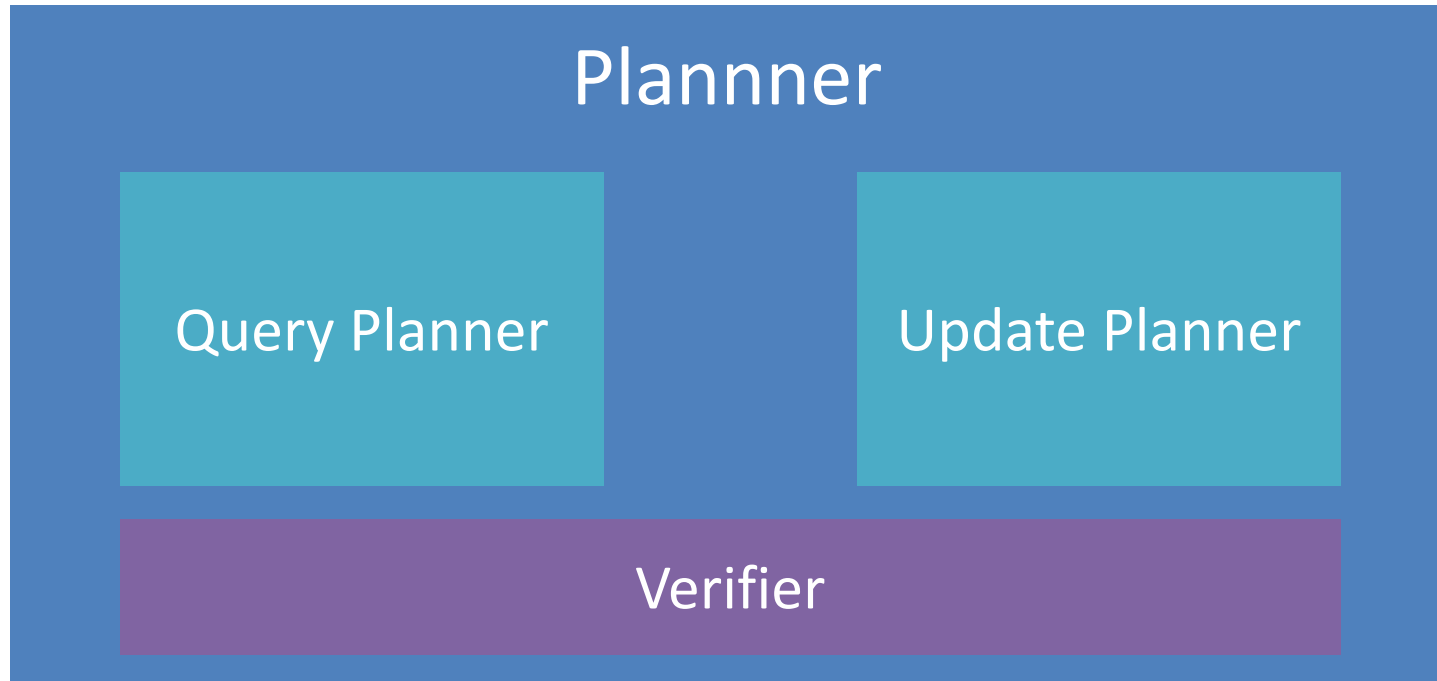
VanillaDB



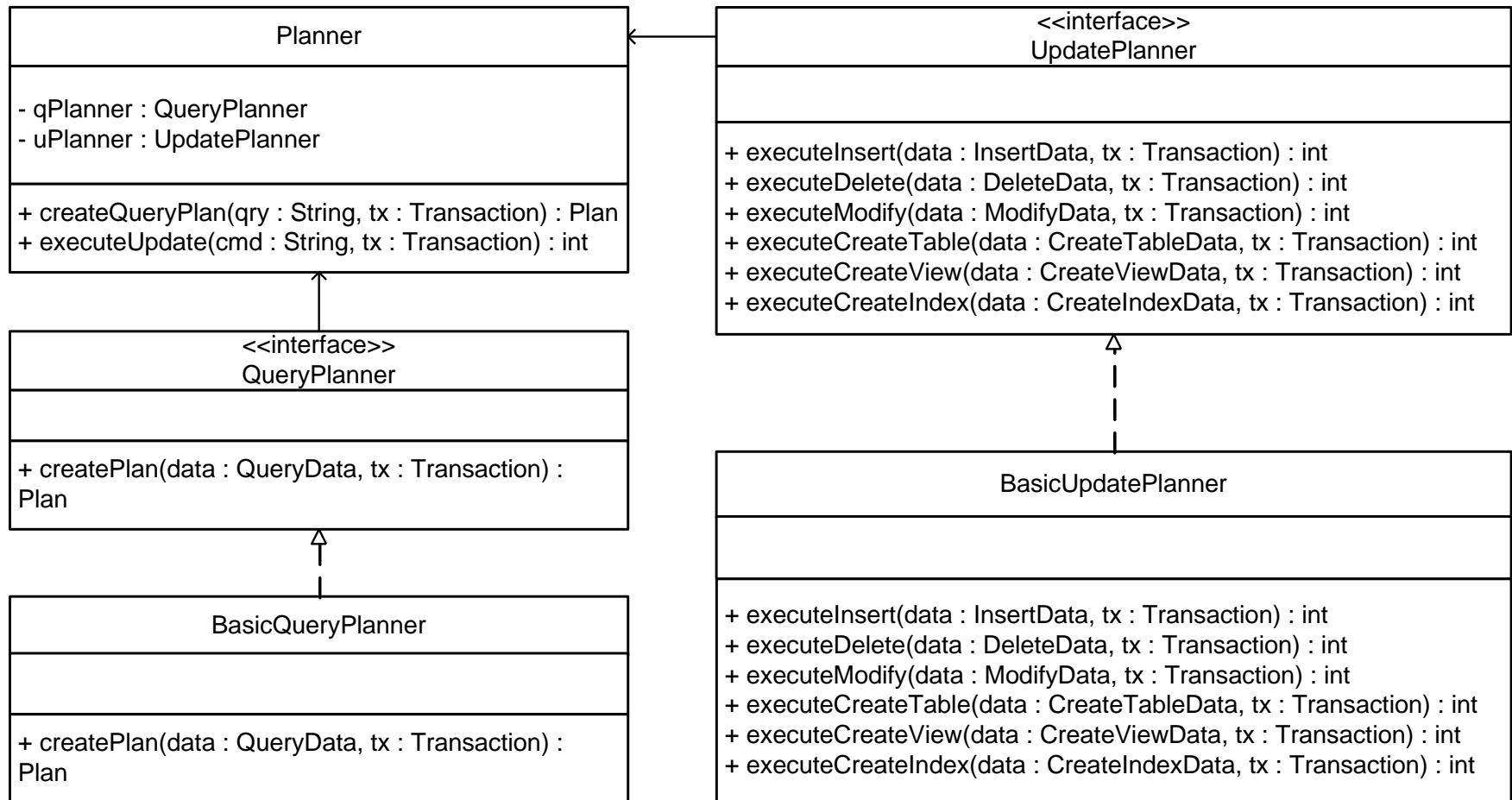
Planner

- 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 a plan tree according to the query

planner Package



Basic Implementation



Advanced Implementation

