

VanillaCore Walkthrough

Part 2

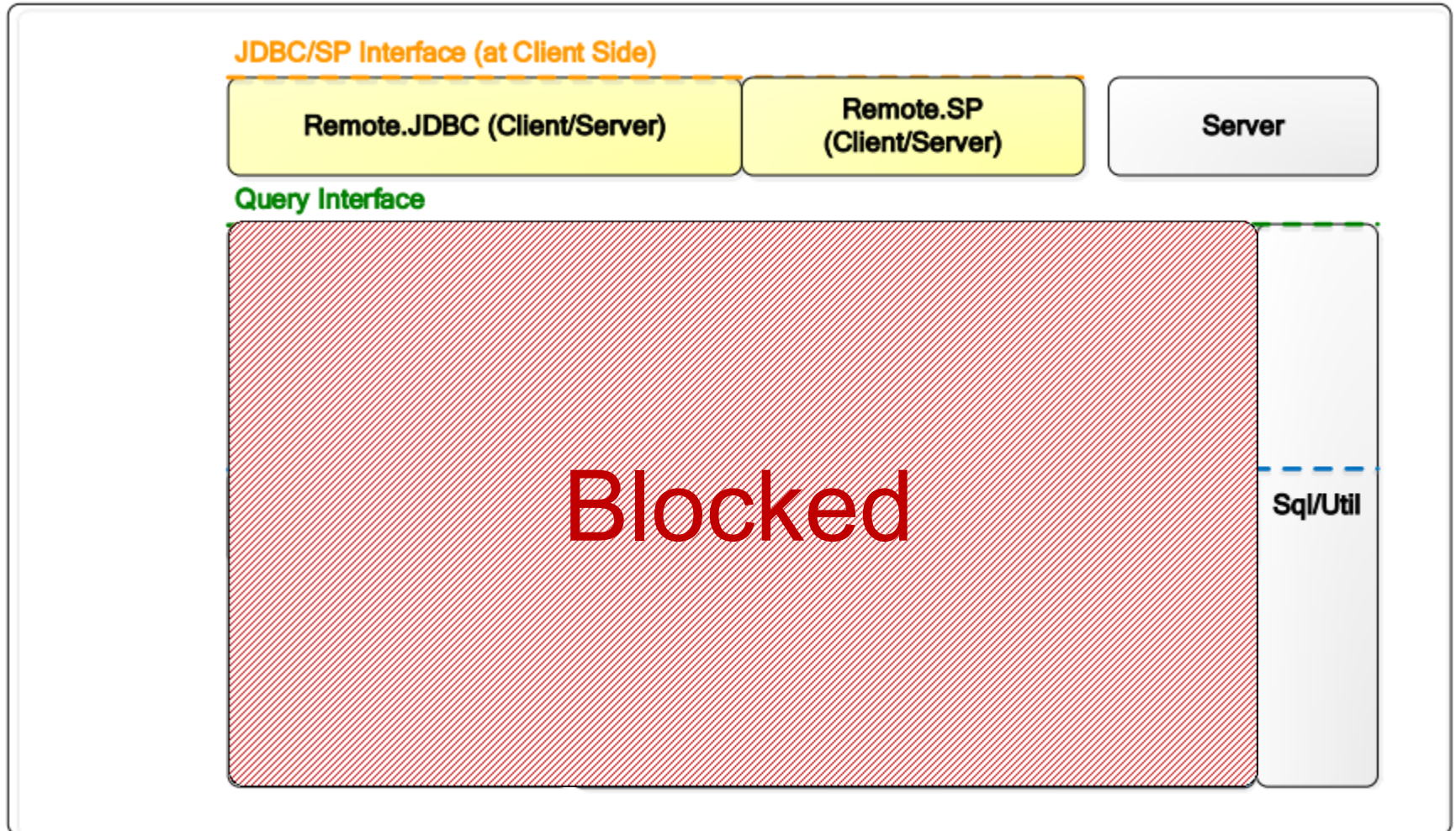
Introduction to Databases

DataLab

CS, NTHU

Last Time

VanillaDB



This Time

VanillaDB

JDBC/SP Interface (at Client Side)

Remote.JDBC (Client/Server)

Remote.SP
(Client/Server)

Server

Query Interface

Query

Sql/Util

This Time

VanillaDB

JDBC/SP Interface (at Client Side)

Remote.JDBC (Client/Server)

Remote.SP
(Client/Server)

Server

Query Interface

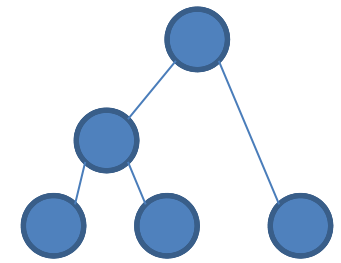
Planner

Parse

Algebra

Sql/Util

Where Are We ?



VanillaDB

SQL

JDBC/SP Interface (at Client Side)

Remote.JDBC (Client/Server)

Remote.SP
(Client/Server)

Server

Query Interface

Planner

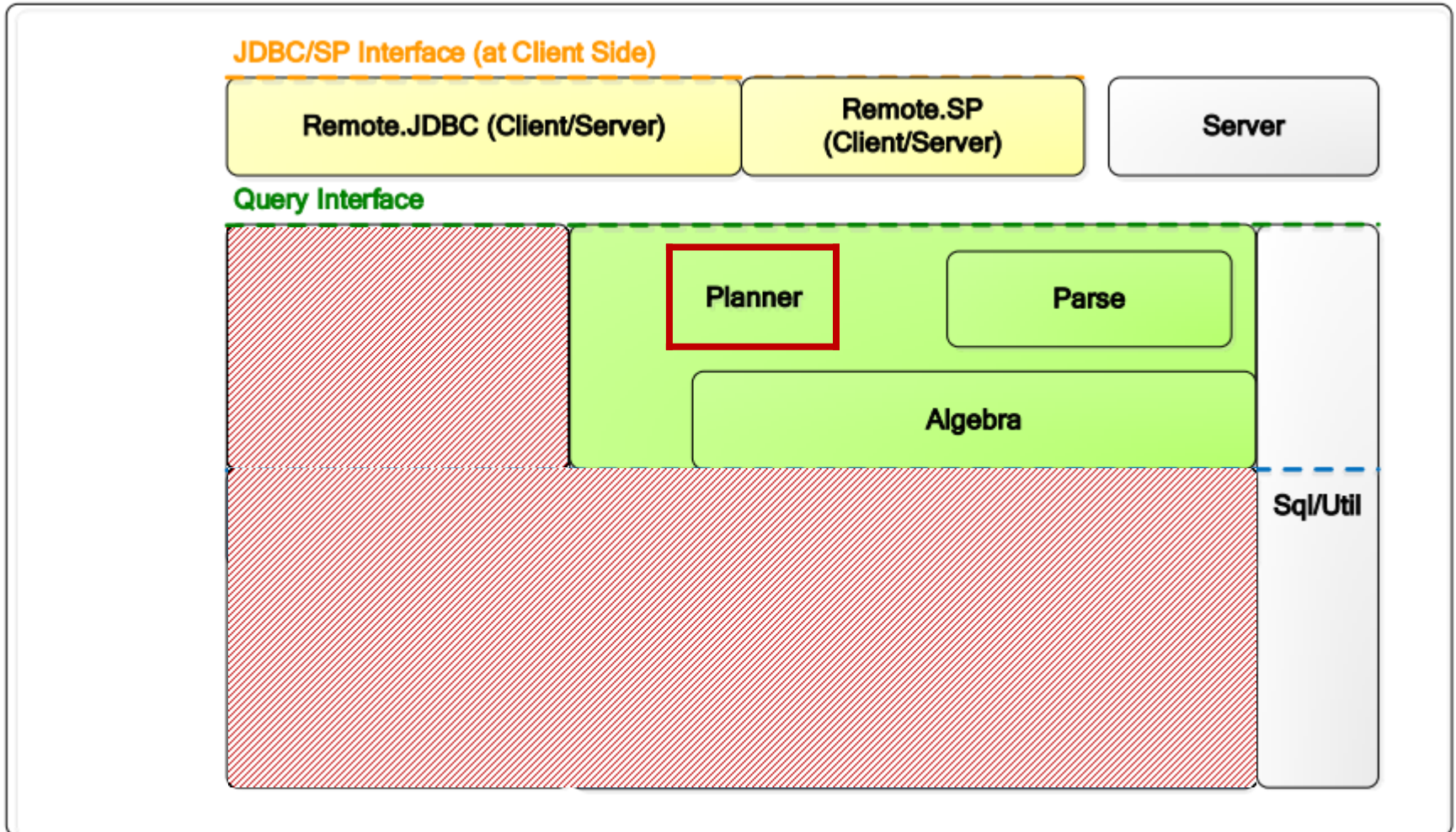
Parse

Algebra

Sql/Util

Where Are We ?

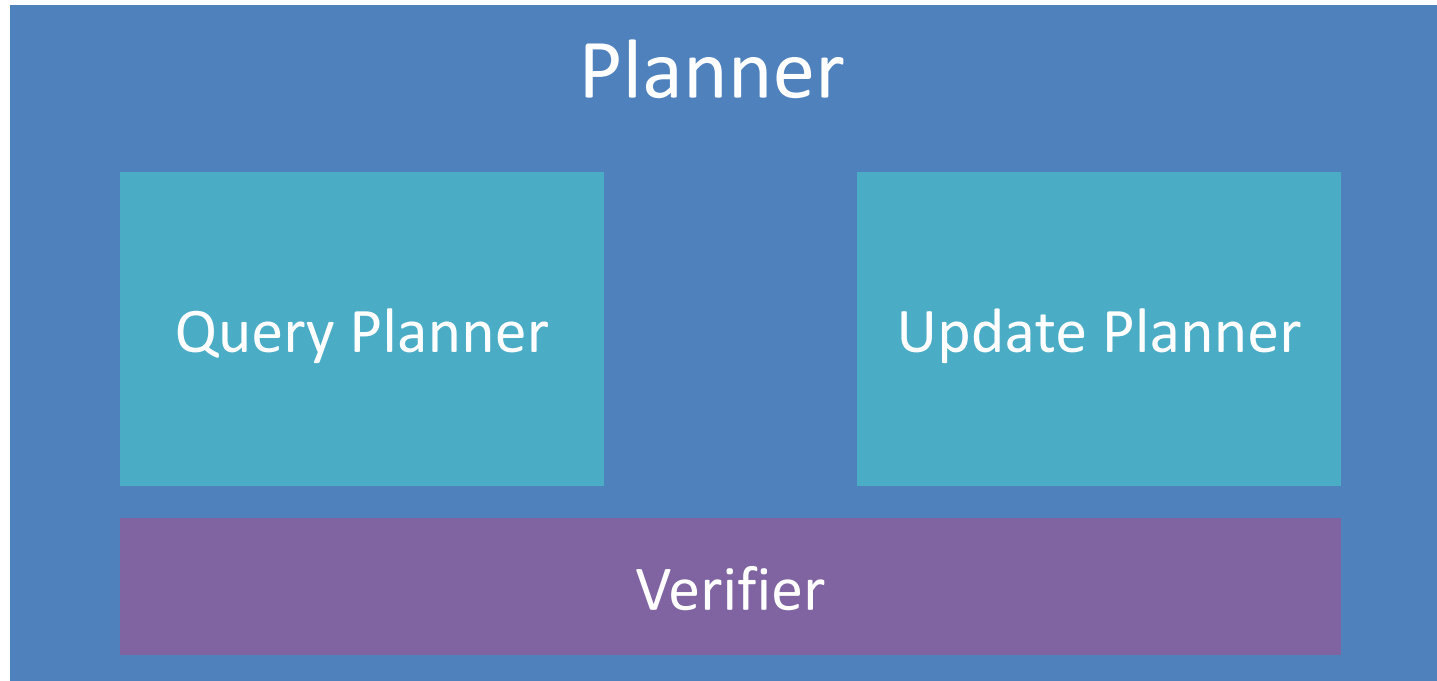
VanillaDB



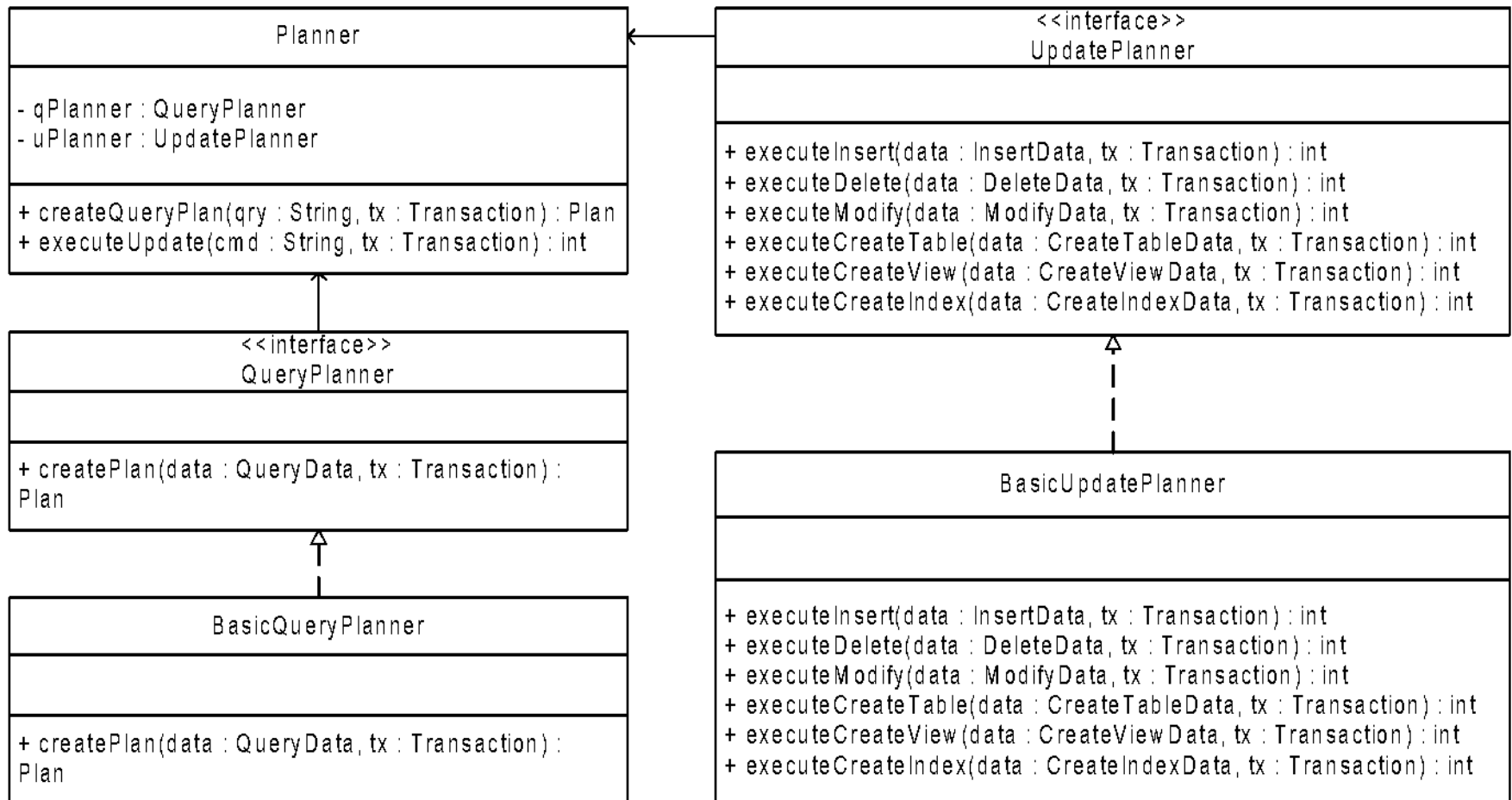
Planner

- The one puts all these together
 1. Accepts a query string
 2. Creates a parser to parse the query string
 3. Verifies the query
 4. Generates a plan tree according to the query

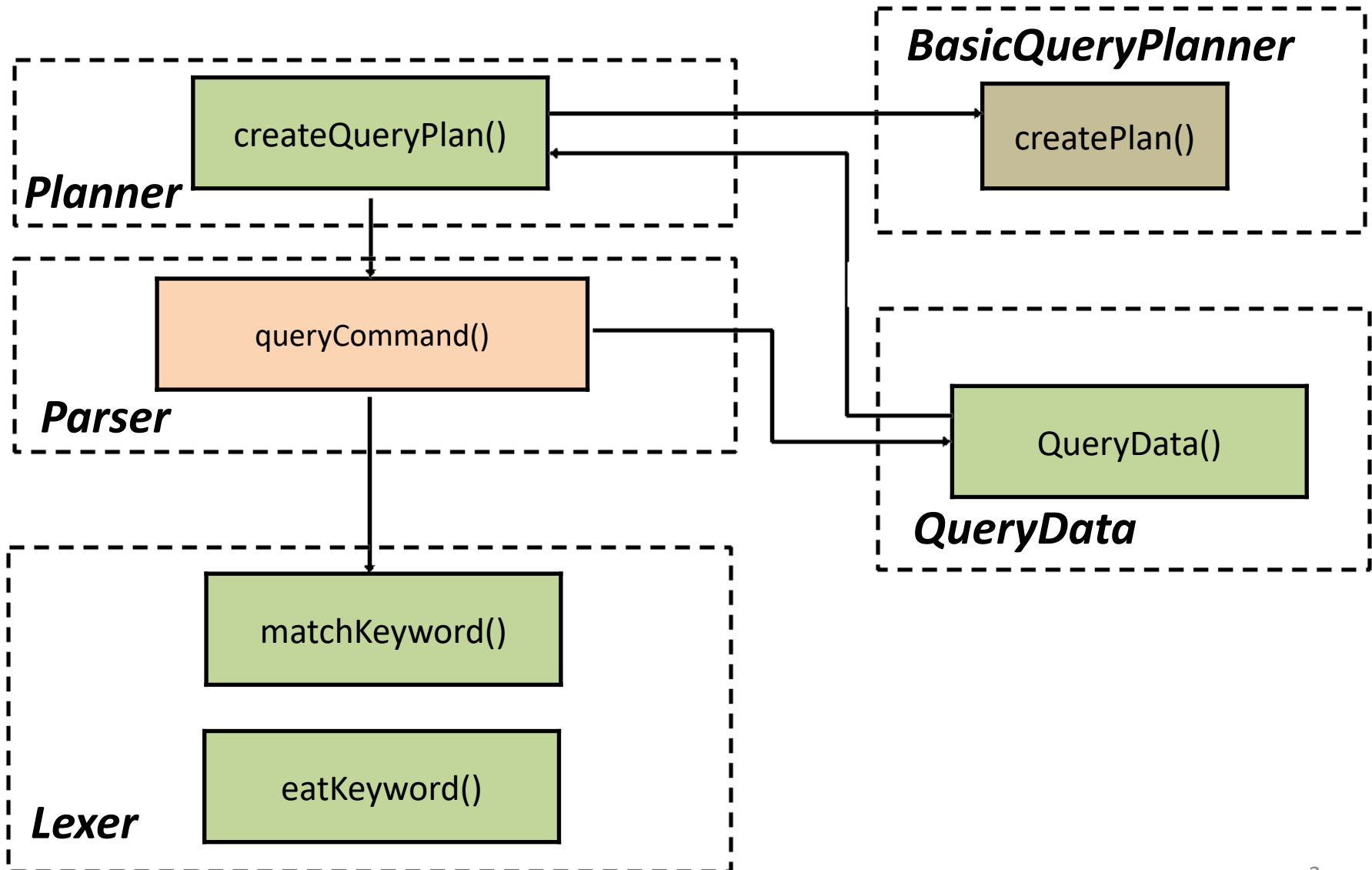
planner Package



Basic Implementation



Overview



BasicQueryPlanner

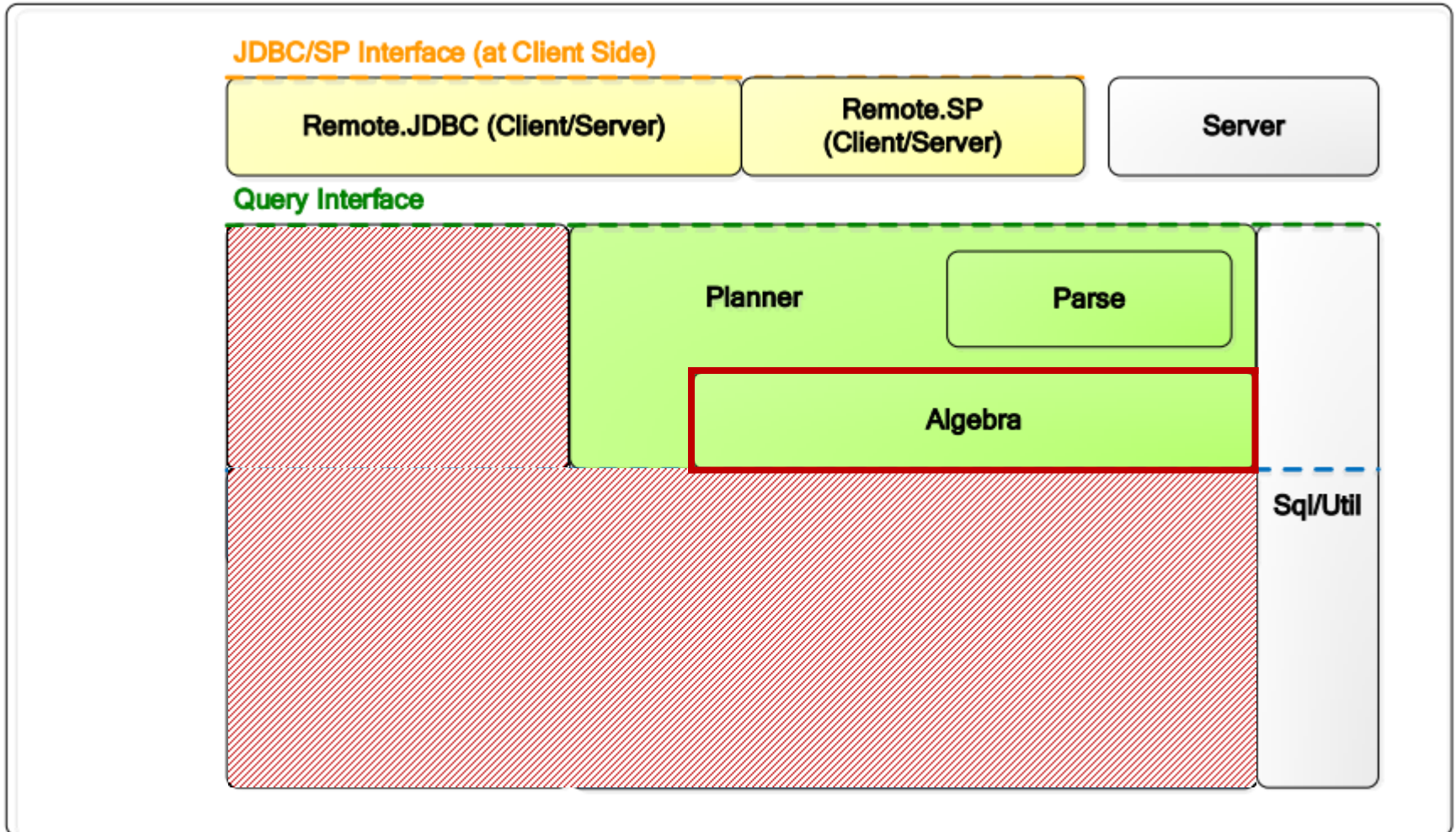
```
public class BasicQueryPlanner implements QueryPlanner {

    /**
     * Creates a query plan as follows. It first takes the product of all tables
     * and views; it then selects on the predicate; and finally it projects on
     * the field list.
     */
    @Override
    public Plan createPlan(QueryData data, Transaction tx) {
        // Step 1: Create a plan for each mentioned table or view
        List<Plan> plans = new ArrayList<Plan>();
        for (String tblname : data.tables()) {
            String viewdef = VanillaDb.catalogMgr().getViewDef(tblname, tx);
            if (viewdef != null)
                plans.add(VanillaDb.newPlanner().createQueryPlan(viewdef, tx));
            else
                plans.add(new TablePlan(tblname, tx));
        }
        // Step 2: Create the product of all table plans
        Plan p = plans.remove(0);
        for (Plan nextplan : plans)
            p = new ProductPlan(p, nextplan);
        // Step 3: Add a selection plan for the predicate
        p = new SelectPlan(p, data.pred());
        // Step 4: Add a group-by plan if specified
        if (data.groupFields() != null) {
            p = new GroupByPlan(p, data.groupFields(), data.aggregationFn(), tx);
        }
        // Step 5: Project onto the specified fields
        p = new ProjectPlan(p, data.projectFields());
        // Step 6: Add a sort plan if specified
        if (data.sortFields() != null)
            p = new SortPlan(p, data.sortFields(), data.sortDirections(), tx);

        return p;
    }
}
```

Where Are We ?

VanillaDB



algebra Package

```
graph TD; A[Plan Classes] --- B[Scan Classes]; A --- C[Materialize Package]; B --- C;
```

Plan
Classes

Scan
Classes

Materialize
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

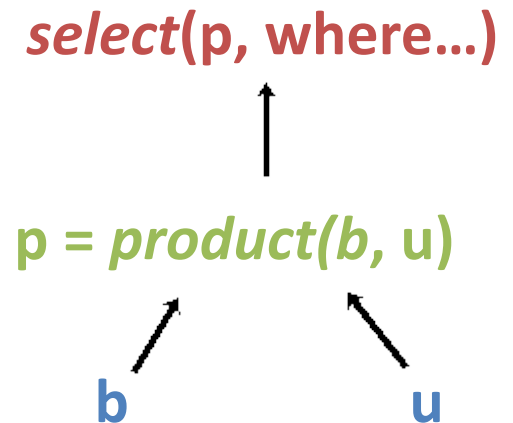
Using a Query Plan

```
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);

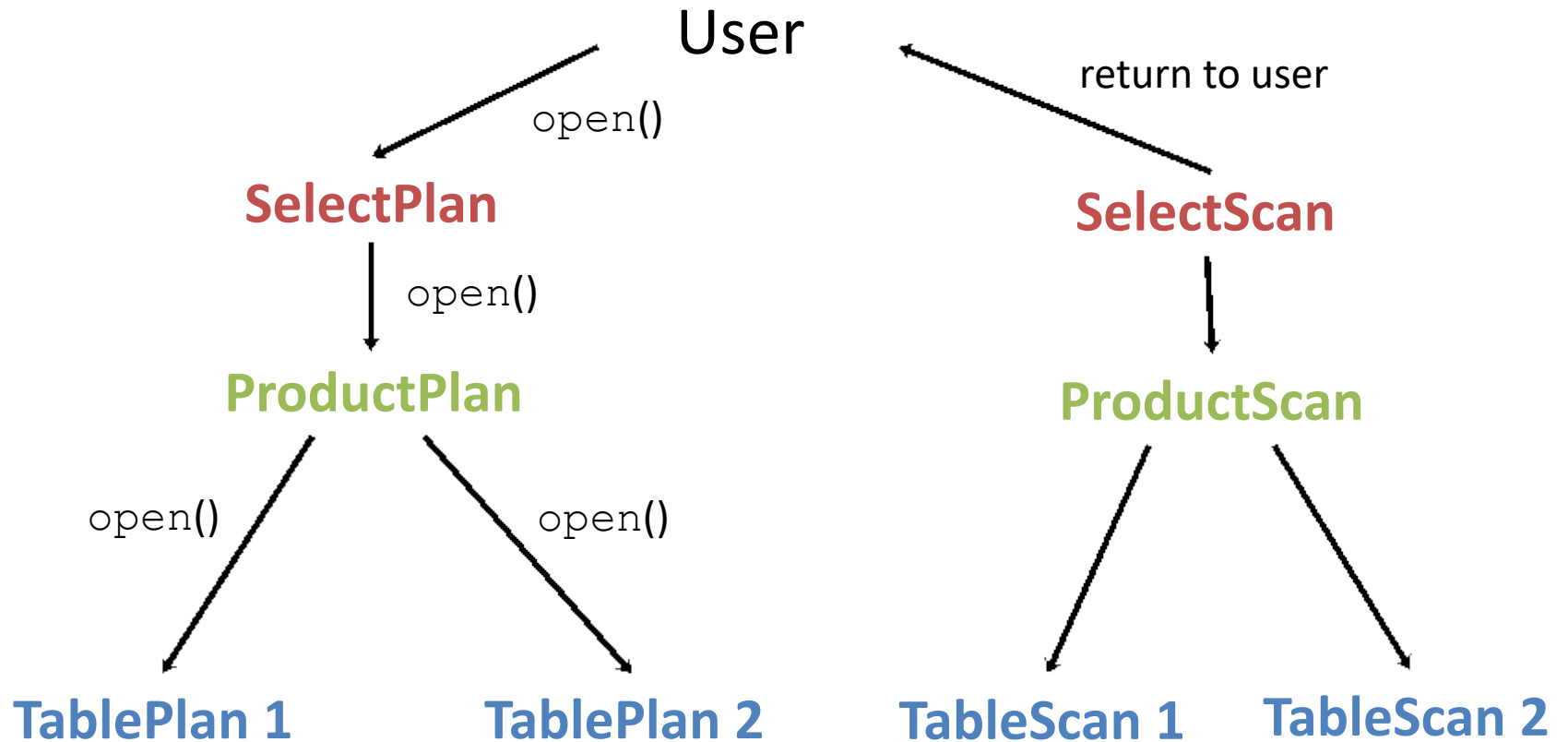
sp.blockAccessed(); // estimate #blocks accessed

// 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 ()



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()

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

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)



beforeFirst()

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

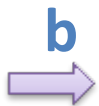


beforeFirst()

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)

next()

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

next()

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)

next()

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

next()

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

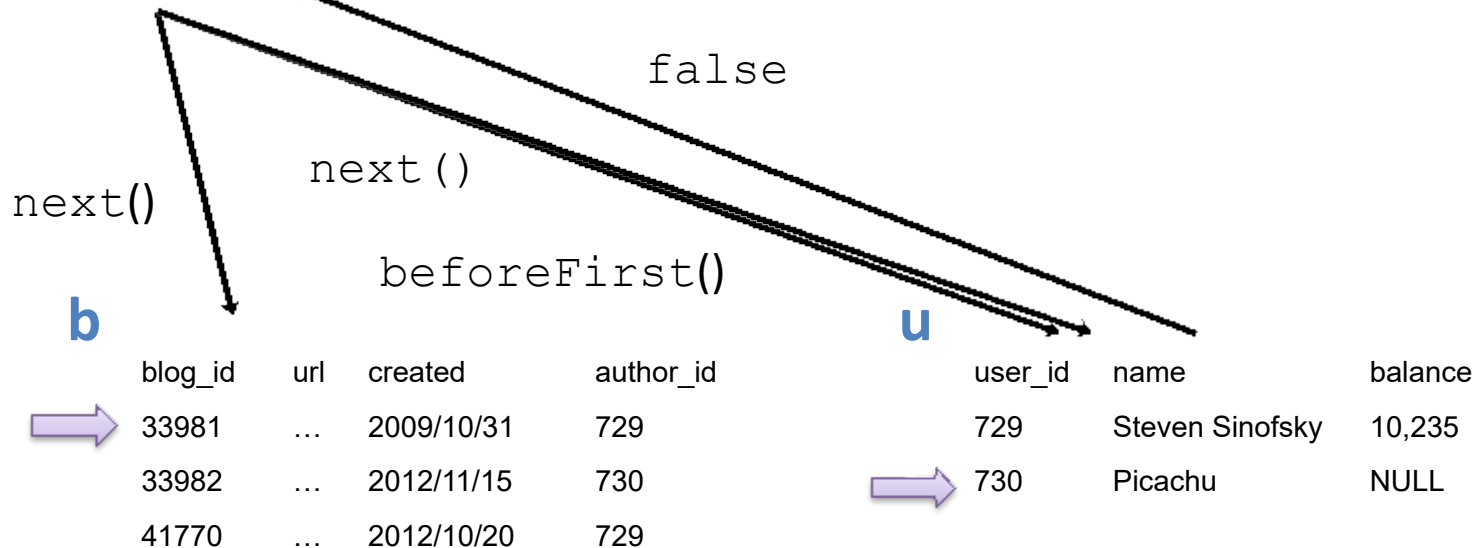
project(s, select blog_id)

↓
next()

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

↓
next()

product(b, u)



Example

project(s, select blog_id)

next()

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	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)

next()

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

next()

product(b, u)

next()

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
33982	...	2012/11/15	730	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

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

algebra Package

Plan
Classes

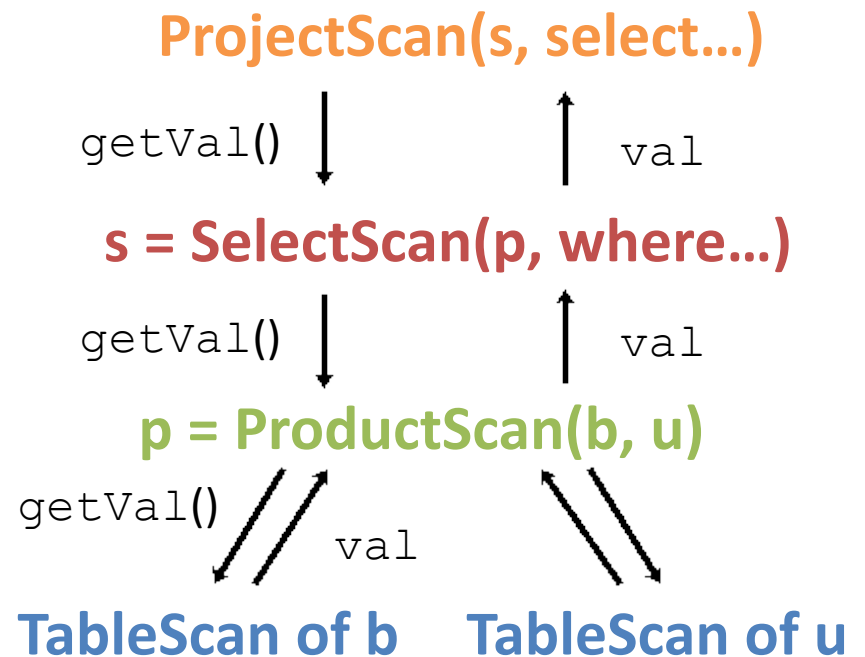
Scan
Classes

?

Materialize
Package

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

Where Are We ?

VanillaDB

JDBC/SP Interface (at Client Side)

Remote.JDBC (Client/Server)

Remote.SP
(Client/Server)

Server

Query Interface

Planner

Parse

Algebra

Sq/Util

Parsing a SQL Query

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

Summary

