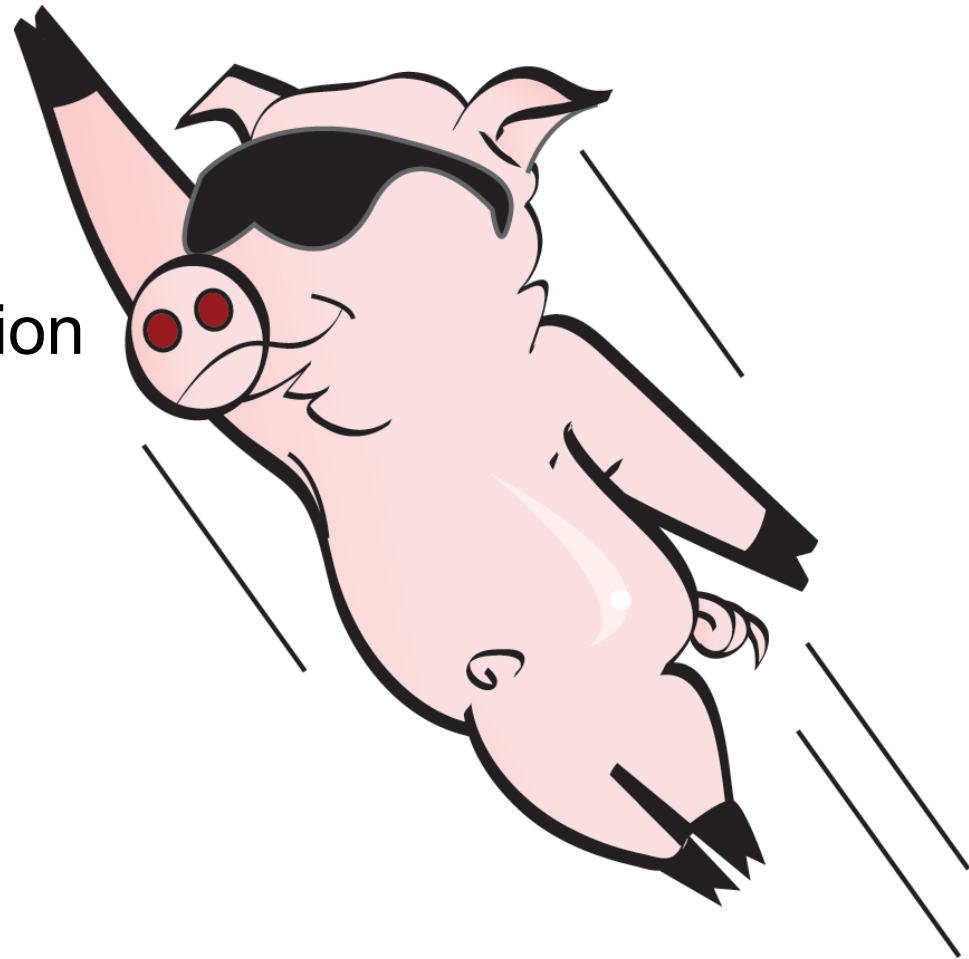

If Pigs Could Fly

Integrating Apache Pig and Stratosphere

DIMA@TU-Berlin Presentation

Vasiliki Kalavri
EMDC - KTH
kalavri@kth.se

May 14th, 2012



Who Am I?

- 2nd-year MSc student
 - European Master in Distributed Computing
 - KTH, UPC, IST
 - Currently doing my thesis at SICS
 - TA for KTH and Ericsson courses
 - Introduction to DS (Erlang)
 - Applied Programming (C++)
 - Functional Programming (Haskell)
 - Electrical and Computer Engineering, NTUA
-

Outline

- Inspiration, Motivation and Goals
 - The Pig System
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - Future Work
-

Outline

- **Inspiration, Motivation and Goals**
 - The Pig System
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - Future Work
-

Hadoop vs. Stratosphere Project

- Semester Project as KTH/SICS comparing Hadoop and Stratosphere
 - Systems' architecture
 - Programming models
 - Performance
 - Developed and evaluated a set of simple applications in both frameworks
-

Hadoop vs. Stratosphere Project Results

- Hadoop's data materialization and replication poses a significant overhead in multi-job applications
 - Match, Co-Group and Cross have a superior performance to their MapReduce emulations (in most of the cases tested)
 - Stratosphere allows flexible execution structures, e.g. aggregation trees.
-

Searching for a thesis project idea in papers' future work sections... ⁷

"All aforementioned languages [Pig, Hive, JAQL] could be changed to compile to PACT programs instead of MapReduce jobs, automatically benefiting from the PACT compiler optimizations..."

Motivation

Pig

- high-level
- simple
- independent of the execution engine



Hadoop

- Scalable
- Fault-tolerant

Motivation

Pig

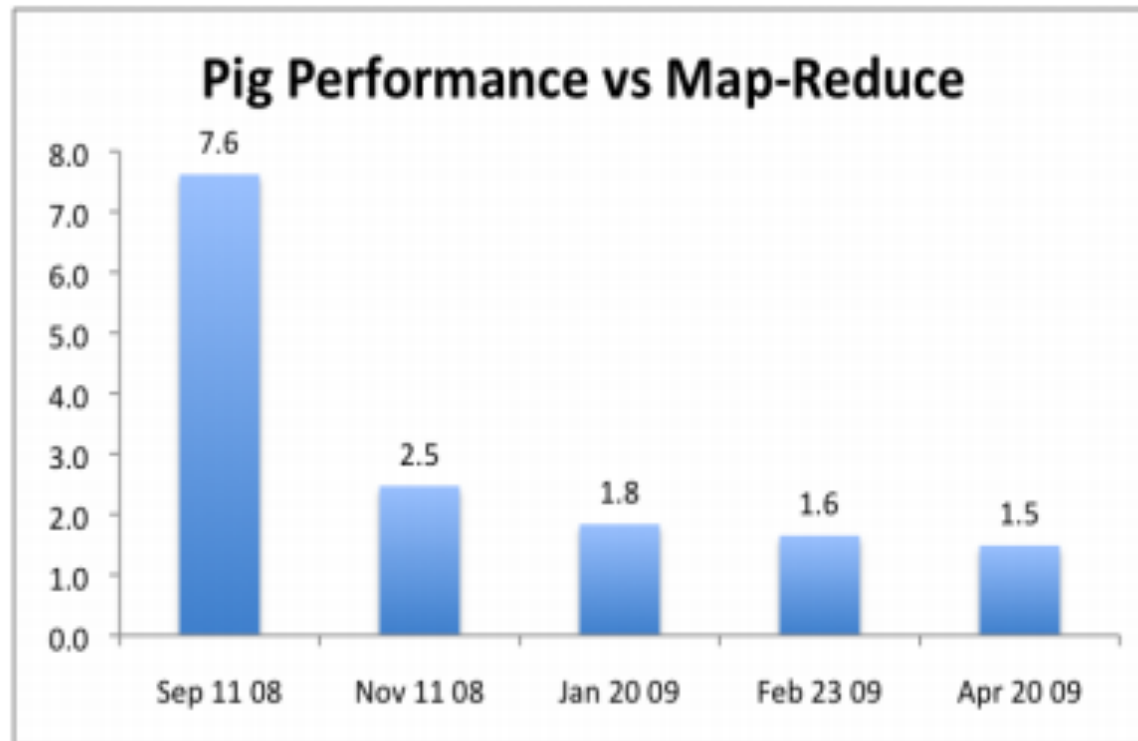
- high-level
- simple
- independent of the execution engine



Hadoop

- Scalable
- Fault-tolerant
- One Input
- Static Pipeline
- Data Materialization and Replication

Pig Performance: Can we do better?

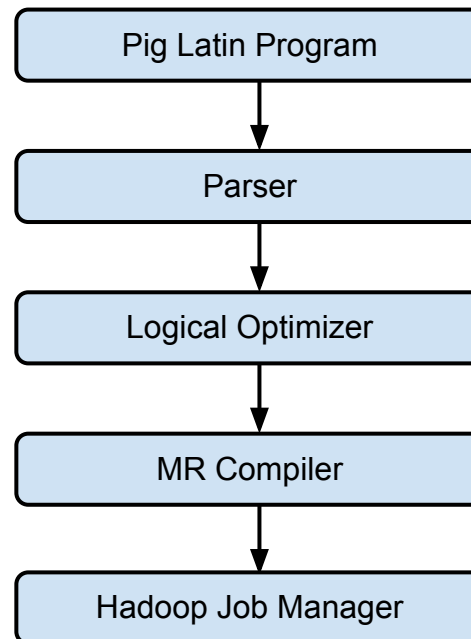


Outline

- Inspiration, Motivation and Goals
 - **Pig**
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - Future Work
-

What is Pig?

- A high-level dataflow system
 - A high-level language: Pig Latin
 - A set of compilers that generate Map-Reduce jobs



Why Pig?

- Simple language
 - Easy for analysts familiar with SQL/Scripting languages
 - No need thinking in MapReduce terms
 - Lower cost to write and maintain
 - Common Operations already provided
 - JOIN, GROUP, FILTER, SORT
 - Scalability and Fault-tolerance of Hadoop
-

Pig Philosophy

- Pigs eats anything
 - relational, nested, ustructured data
 - files or key-value stores
 - Pigs live anywhere
 - intended to be independent of the execution engine
 - Pigs are domestic animals
 - allows integration of user code
 - Java, scripting languages, Map-Reduce jars
 - Pigs Fly
 - performance is the primary goal
-

Pig Latin: A Not-So-Foreign Language

- Declarative Data-Flow Language
 - Describes a DAG where edges are dataflows and nodes are operators
 - Consists of
 - **Statements** with **Variables** of specific **Types** and Built-In or User-Defined **Functions**
-

Pig Latin: Statements

- Loading and Storing
 - LOAD, STORE, DUMP
 - Filtering
 - FILTER, DISTINCT, FOREACH...GENERATE, STREAM
 - Grouping and Joining
 - JOIN, COGROUP, GROUP, CROSS
 - Sorting
 - ORDER, LIMIT
 - Combining and Splitting
 - UNION, SPLIT
-

Pig Latin: Types

- **Numeric**
 - int, long, float, double
 - **Text**
 - chararray
 - **Binary**
 - bytearray
 - **Complex**
 - tuple: sequence of fields of any type
 - bag: unordered set of tuples
 - map: set of key-value pairs
-

Pig Latin: Functions

- Eval
 - AVG, CONCAT, COUNT, DIFF, MAX, MIN, SIZE, SUM, TOKENIZE
 - Filter
 - IsEmpty
 - Load/Store
 - PigStorage, BinStorage, BinaryStorage, TextLoader, PigDump
-

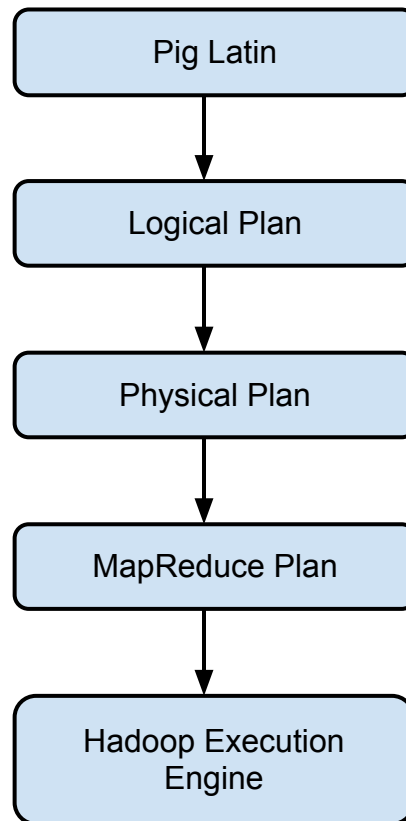
An Example

Dataset: `urls(url, category, pagerank)`

*Find the **top 10 urls by pagerank**,
for each sufficiently large category:*

```
urls = LOAD 'dataset' AS (url, category, pagerank);  
groups = GROUP urls BY category;  
bigGroups = FILTER groups BY COUNT(urls)>100000000;  
result = FOREACH bigGroups GENERATE  
        group, top10(urls);  
STORE result INTO 'output';
```

System Internals



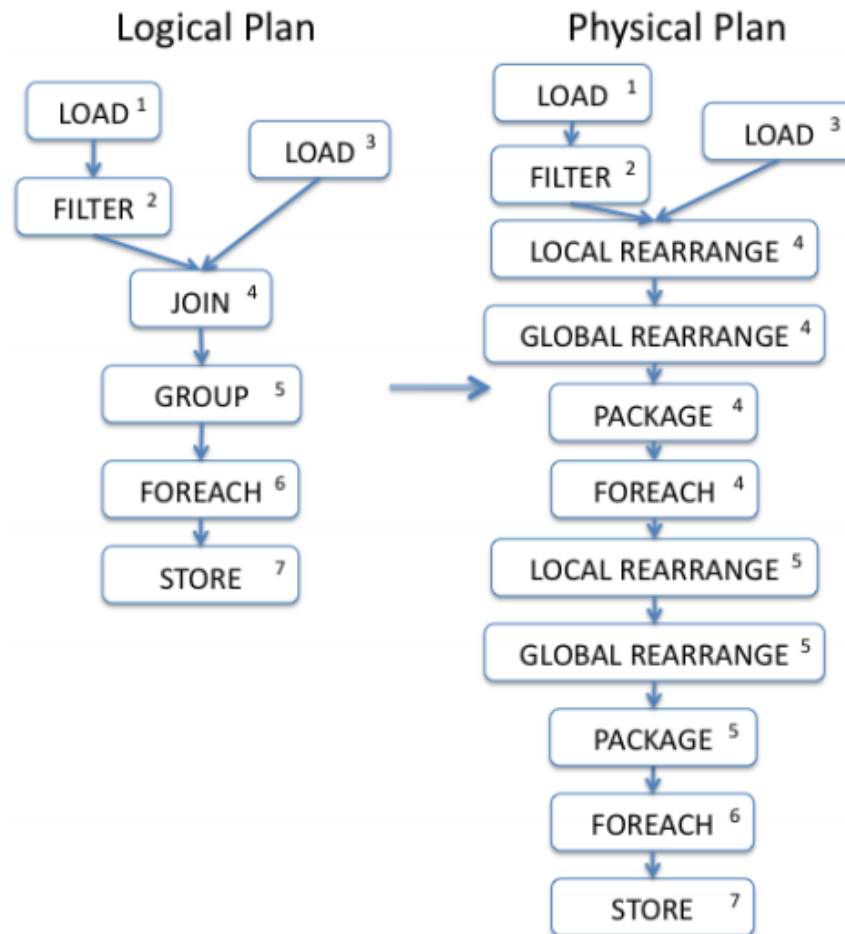
Logical Plan

```
A = LOAD 'file1' AS (x, y, z);  
B = LOAD 'file2' AS (t, u, v);  
C = FILTER A BY y>0;  
D = JOIN C BY x, B BY u;  
E = GROUP D BY z;  
F = FOREACH E GENERATE group, COUNT(D);  
STORE F INTO 'output';
```

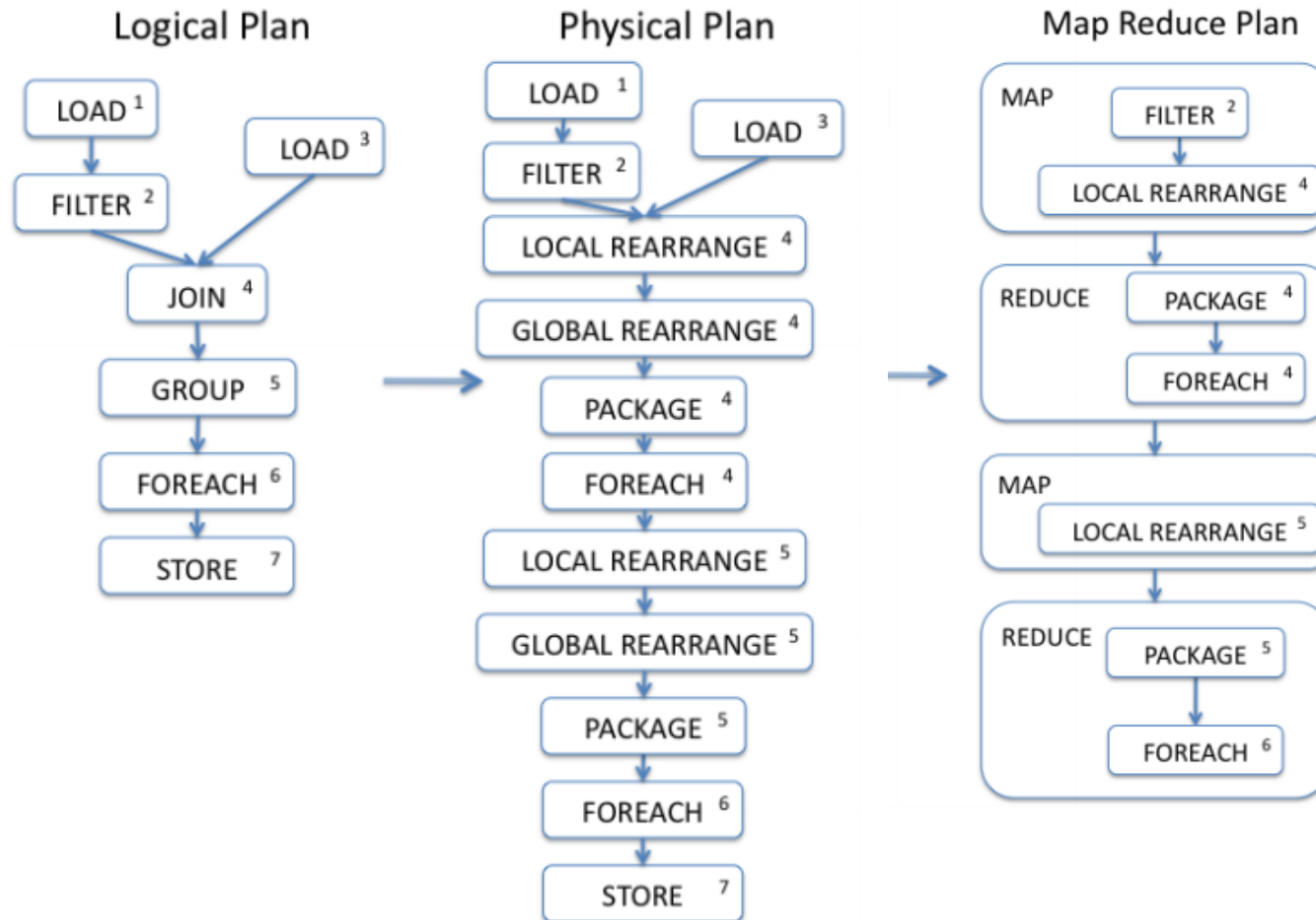
Logical Plan



Logical To Physical Plan



Physical Plan to Map-Reduce Plan



Logical Plan Optimizer

- Each Pig Latin Statement corresponds to one node in the initial Logical Plan
 - Depending on how the user has written the script, the generated plan might be highly inefficient
 - Before the transformation to a Physical Plan, a series of optimizations are performed
-

RuleSets

- The optimizer is given a list of **RuleSets**
 - rules that **can be run together**
 - The rules are run **repeatedly**
 - until no rule is applicable anymore
 - until maxIter (500 by default)
 - Each RuleSet is checked **in order** and **only once**
-

Rules

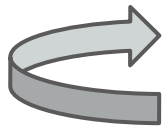
- WholePlanRules
 - Rules that operate on the whole Plan
 - Rules with a **pattern** and an associative **transformer**
 - If a pattern is **matched**
 - **check** if the rule should be run or not, e.g. if a filter is "pushable"
 - if yes, **transform** the Logical Plan
-

WholePlanRules

- ColumnMapKeyPrune
 - ColumnPrune
 - Removes a column if it is mentioned in a script, but never used
 - MapKeysPrune
 - Removes map keys that are not mentioned in the script
-

FilterAboveForeach

- Moves Filter above Foreach
- Checks if uid on which Filter works on is present in the predecessor of Foreach
- If so it transforms it



```
A = LOAD 'file' AS (a:int, b:int, c:int);  
B = FOREACH A GENERATE a+2, b;  
C = FILTER B BY b > 0;  
STORE C INTO 'out_foreach';
```

MergeFilter

- Merge 2 consecutive Filter operators, adding the condition of the 2nd Filter into the condition of the 1st Filter with an AND operator.

```
B = FILTER A BY a > 0;  
C = FILTER B BY b > 0;
```



```
B = FILTER A BY (a > 0 AND b > 0);
```

MergeForeach (1)

- The 2nd Foreach needs to have only LOMGenerate and LOMInnerLoad
- The 1st Foreach shouldn't have Flatten in its generate statement

```
A = GROUP words BY word;  
B = FOREACH A GENERATE group, COUNT(A) AS cnt;  
C = FOREACH B GENERATE upperCase(group), cnt;
```



```
B = FOREACH A GENERATE upperCase(group), COUNT(A)  
AS cnt;
```

MergeForeach (2)

- Check if none of the 1st Foreach output is referred more than once in 2nd Foreach
 - Otherwise, we may do expression calculation more than once, defeat the benefit of this optimization

```
A = LOAD 'input' AS (x, y, z);  
B = FOREACH A GENERATE 2*x+y AS b;  
C = FOREACH B GENERATE b, b+z;
```

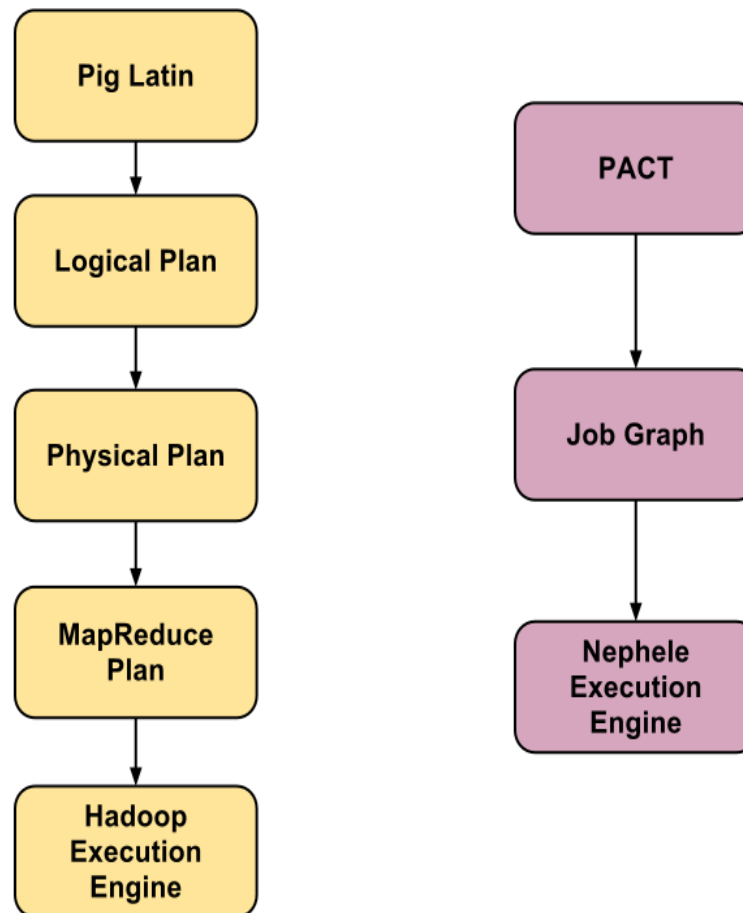
Other Optimizations

- PushUpFilter
 - PushDownForeachFlatten
 - LimitOptimizer
 - LogicalExpressionSimplifier
-

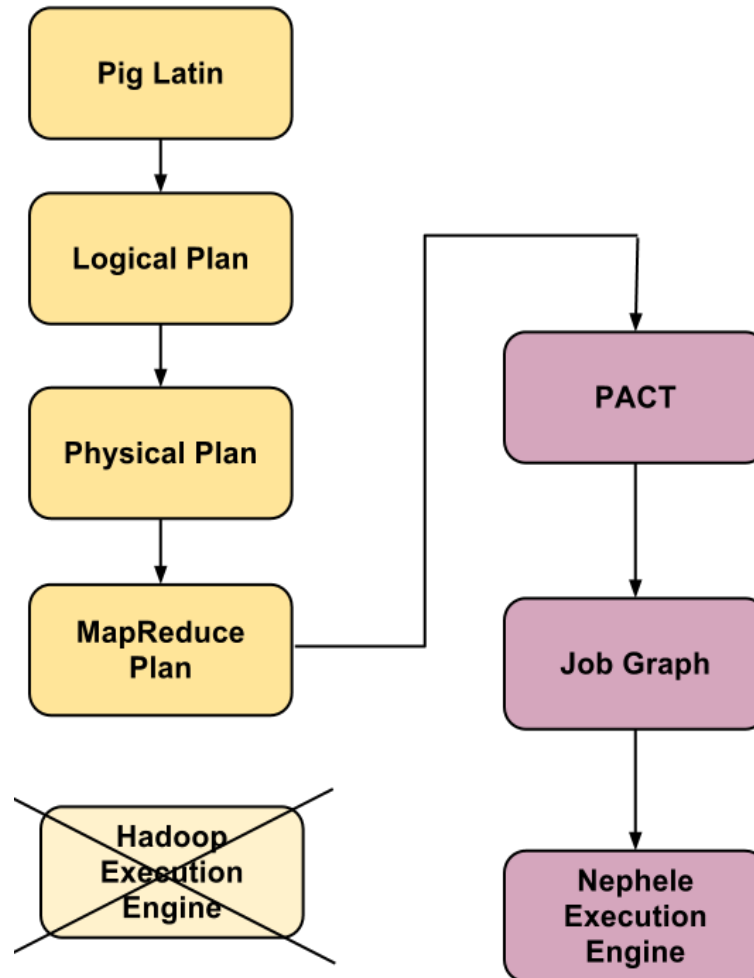
Outline

- Inspiration, Motivation and Goals
 - Pig
 - **Integration Alternatives**
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - Future Work
-

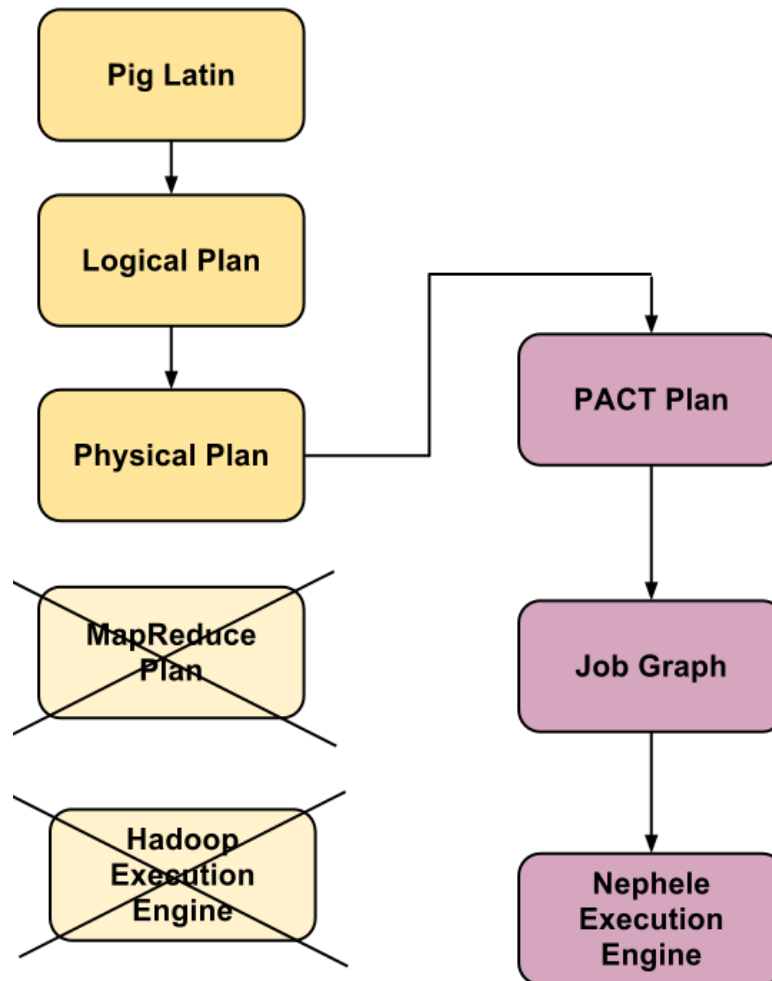
Merging Pig and Stratosphere Flows



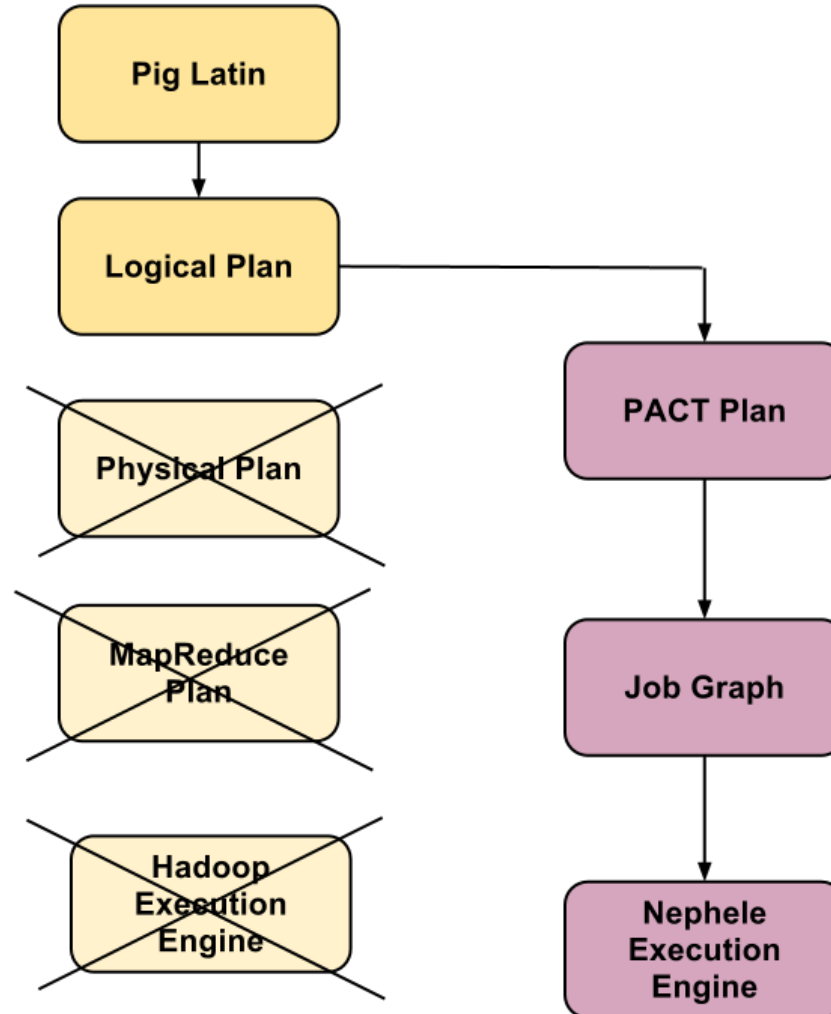
Merging Pig and Stratosphere Flows



Merging Pig and Stratosphere Flows



Merging Pig and Stratosphere Flows



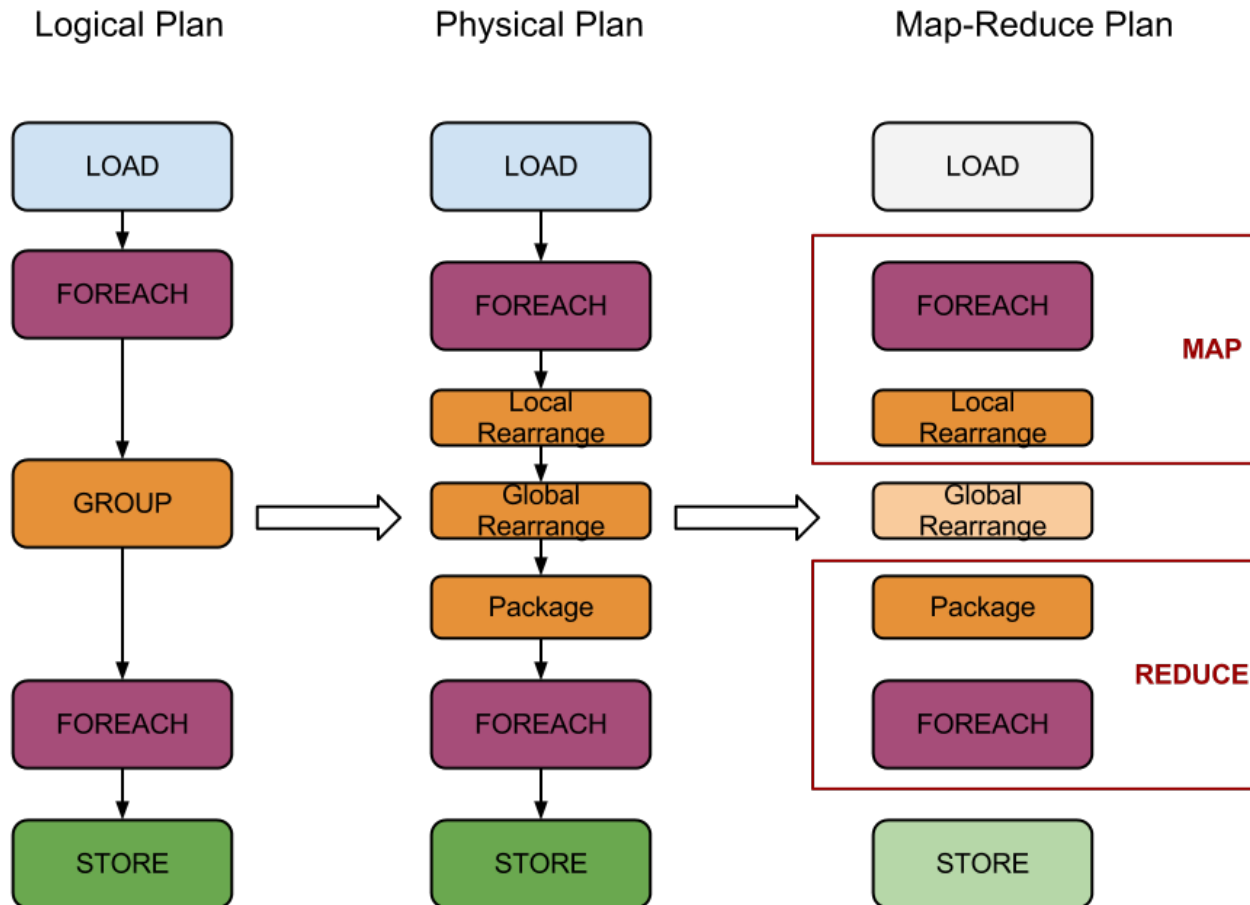
Outline

- Inspiration, Motivation and Goals
 - Pig
 - Integration Alternatives
 - **Compilation of Basic Pig Operations**
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - Future Work
-

Pig's Compilation Algorithm

- 1 MR Op ~ 1 MR Job and has a "Map-phase" and an optional "Reduce-phase"
 - The goal is to keep the number of MR Ops minimum
 - Categorize Physical Ops as blocking and non-blocking
 - Push all non-blocking Ops in current phase
 - Create a new phase for each blocking Op
-

Example

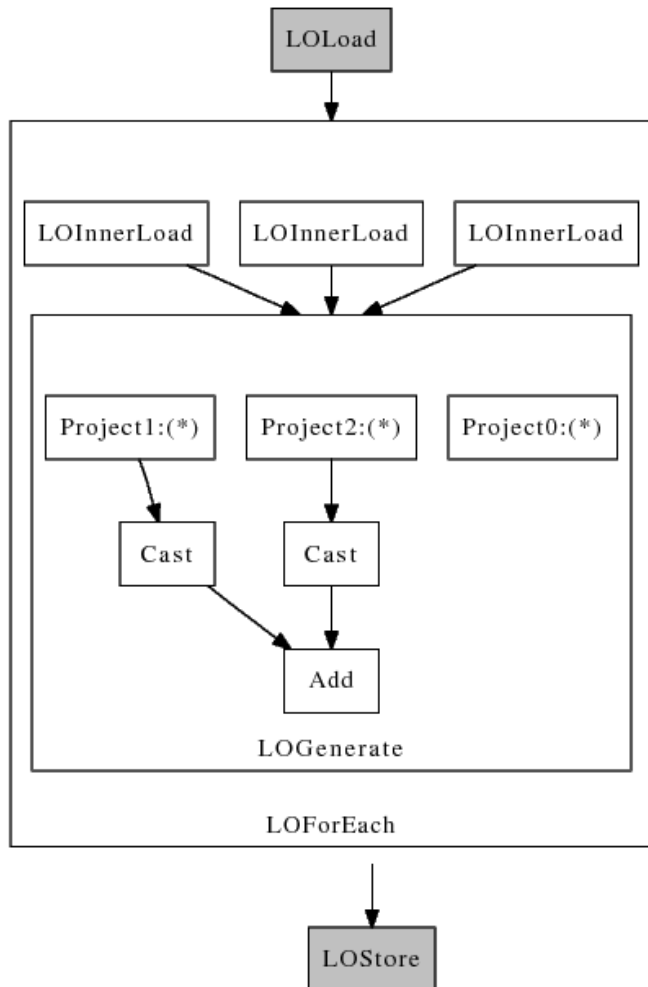


FOREACH

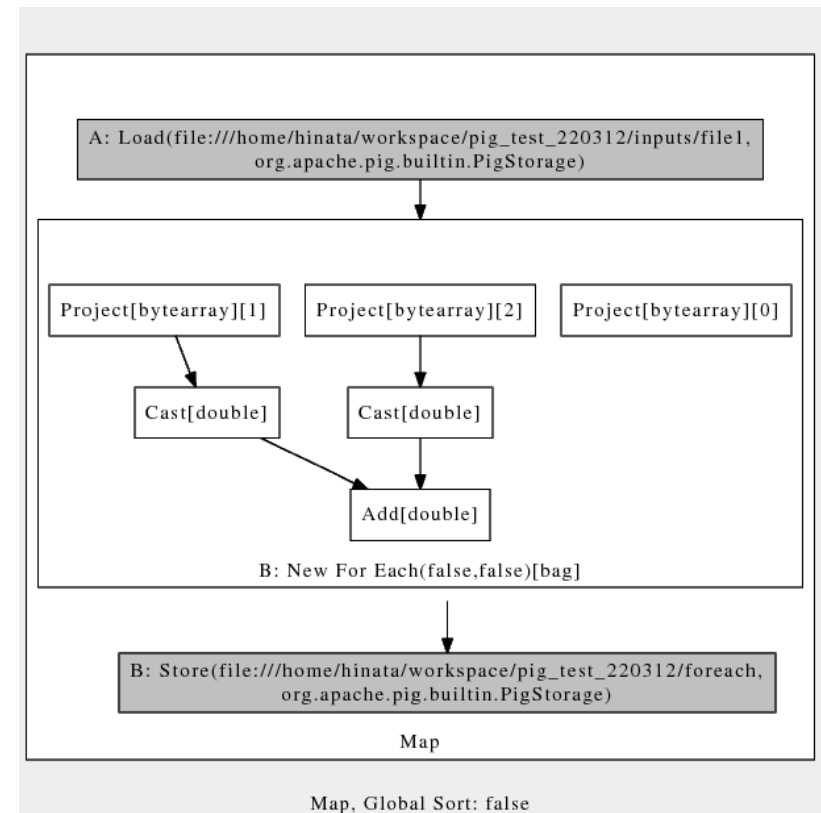
- Applies a set of expressions to each record and generates new records
- Can be merged in the previous Map/Reduce Operators

```
A = LOAD 'input' AS (user_id, name, income_a, income_b);  
B = FOREACH A GENERATE user_id, income_a + income_b AS total_income;
```

FOREACH - Example



A Map-Only Job



FILTER

- Selects records of the data for which a given predicate is true
- Can be merged in the previous Map/Reduce Operators

```
A = LOAD 'input' AS (name, surname, salary);  
B = FILTER A BY salary > 10000;
```

GROUP

- Collects all records with the same key in a bag
- The output records have two fields:
 - the key and the bag with the collected records

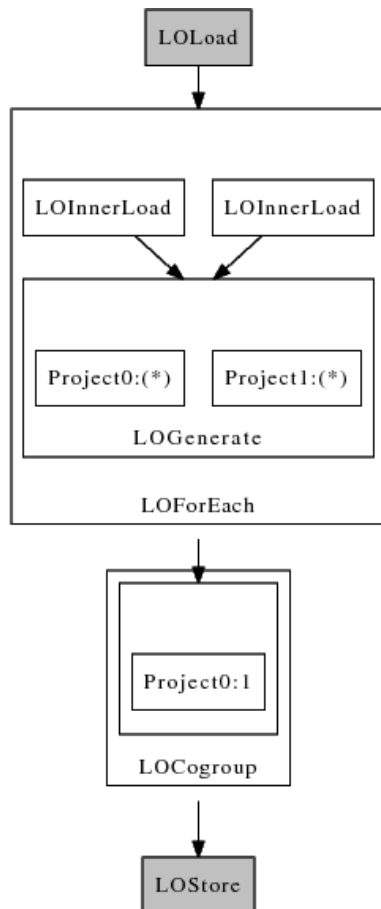
```
A = LOAD 'input' AS (userid, pageid);  
B = GROUP A BY pageid;  
DESCRIBE B;
```

```
B:{group: bytearray, A: {(userid: bytearray, pageid:  
bytearray) }}
```

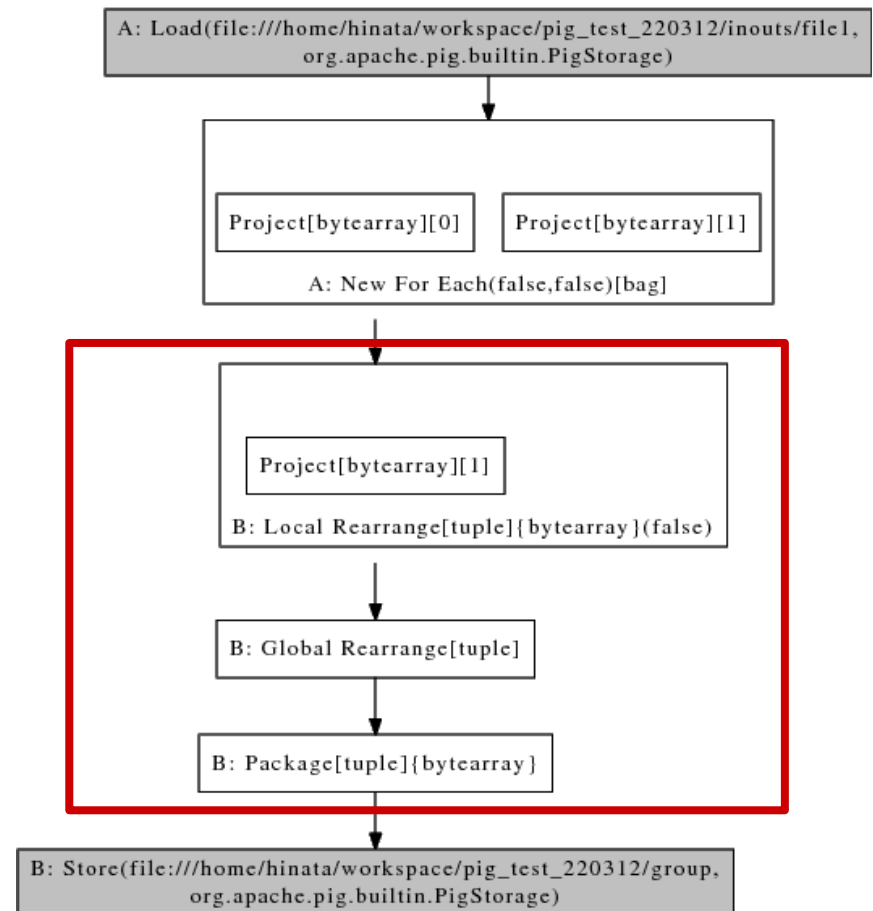
GROUP

- We can Group on multiple fields
 - We can Group All
 - The record coming out of GROUP ALL has the literal "all" as a key
 - Usually used to pass the output to an aggregate function such as COUNT
 - Group will force the creation of a Reduce phase
-

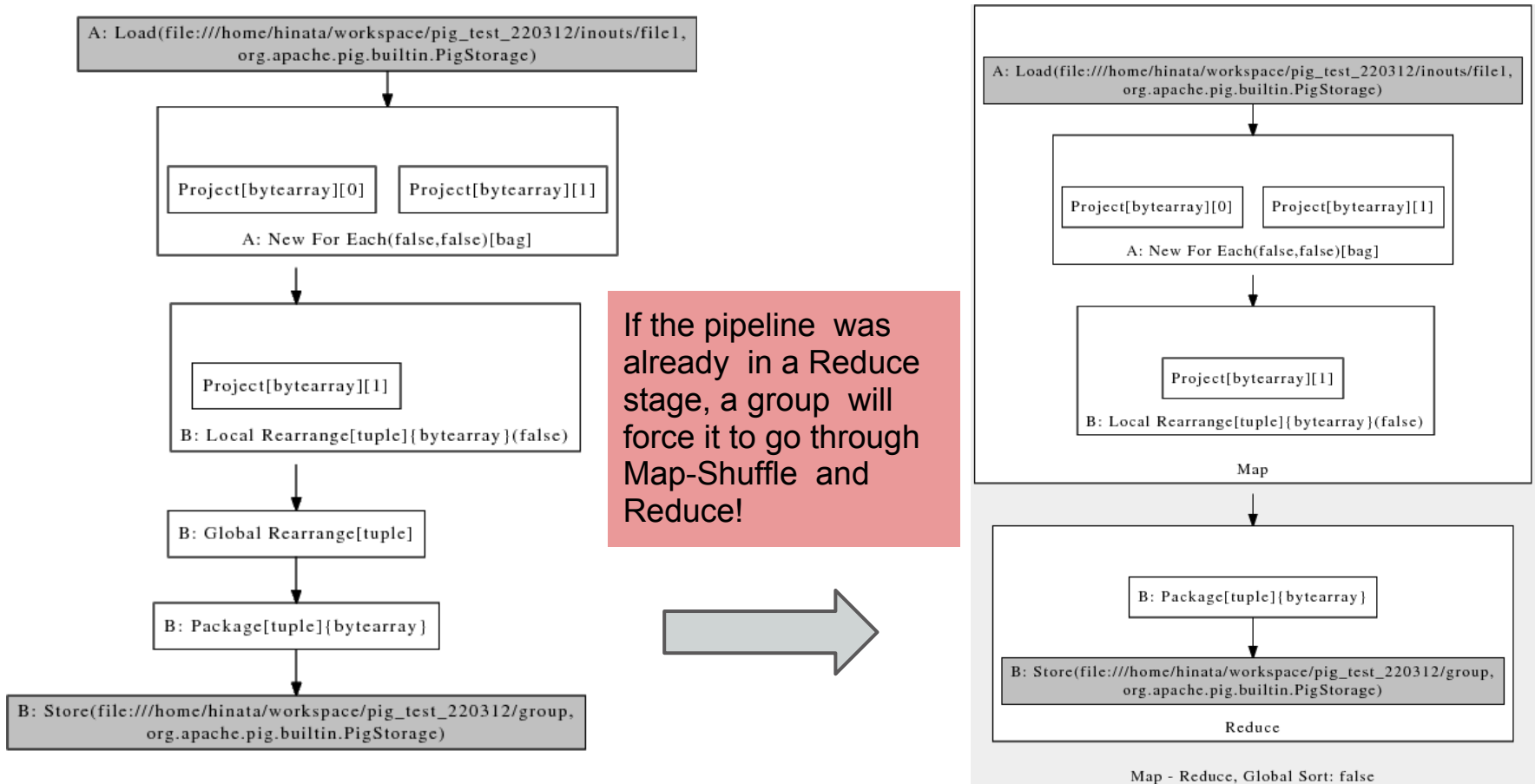
GROUP - example



3 Physical Operators



GROUP - example

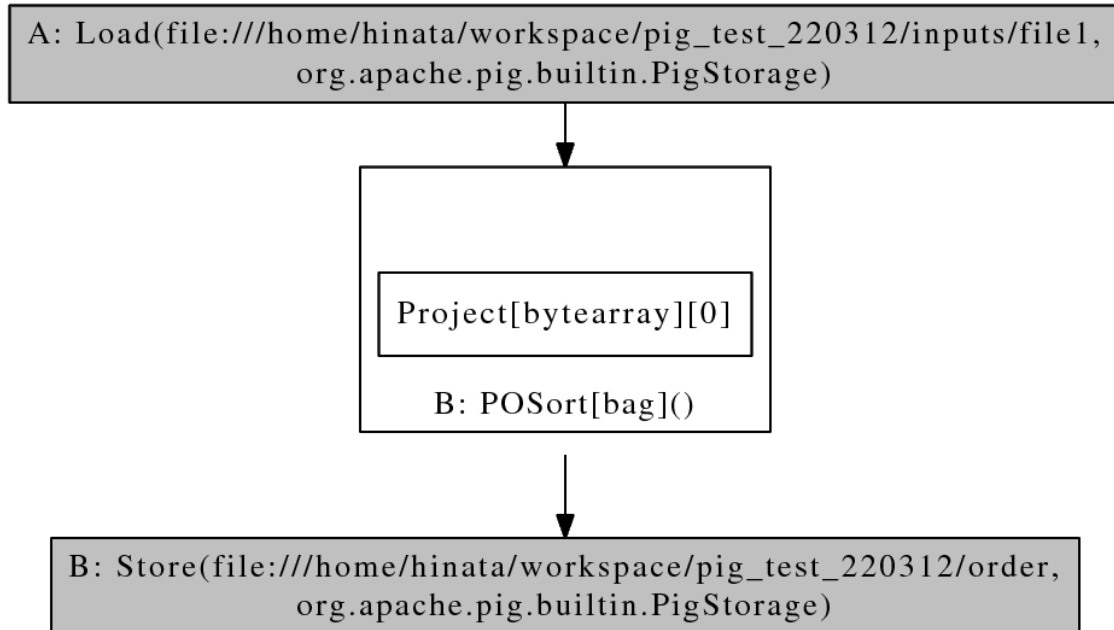


ORDER

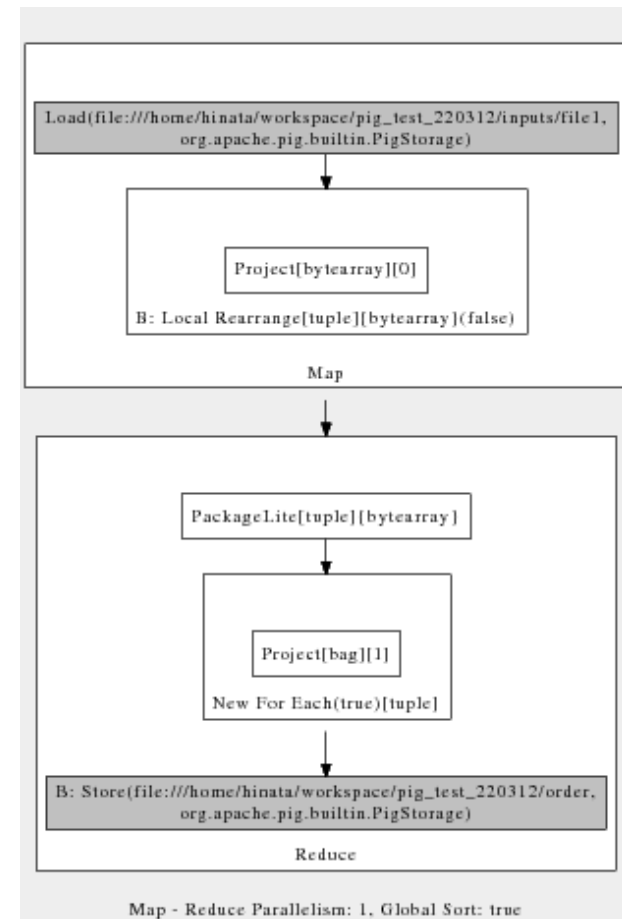
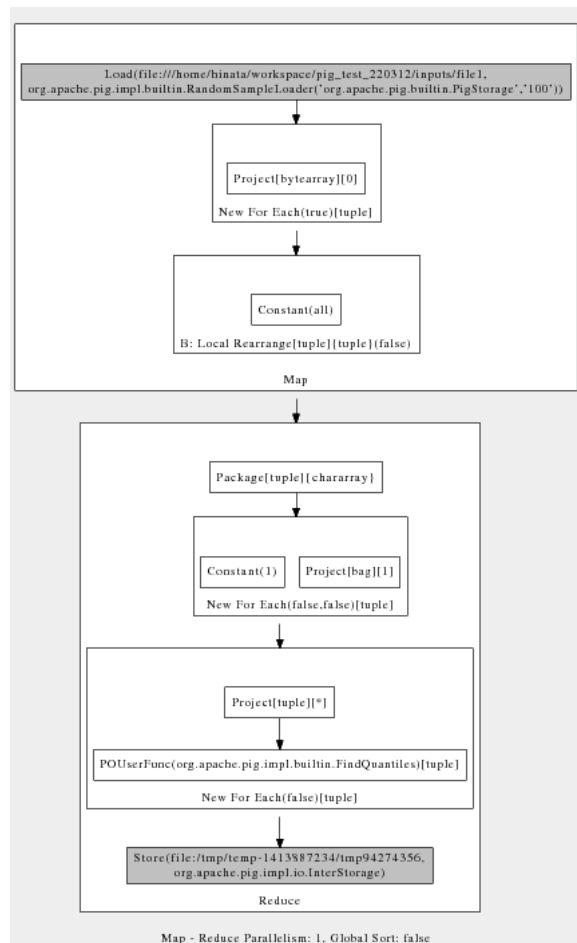
- Sorts the data by one or more keys
 - Can be ASC or DESC
 - Will force the creation of 2 MR Jobs
 - the 1st for key sampling and balancing the load among reducers
 - the 2nd for the sorting itself
-

ORDER - example

```
A = LOAD 'input';  
B = ORDER A BY $0;  
STORE B INTO 'order';
```



ORDER - example



JOIN

- Selects records from one input to put together with records from another input when they share the same key
 - The default join type in Pig is inner
 - Self Joins and, Outer Joins and Join on multiple keys also allowed
-

JOIN

- 1 MR Job
 - Inputs are annotated in the map phase
 - The Join key is used as the Shuffle phase key
 - Reducer implements a cross product
 - Advanced Join Techniques available
 - Fragment - Replicate
 - Merge
 - Skew
-

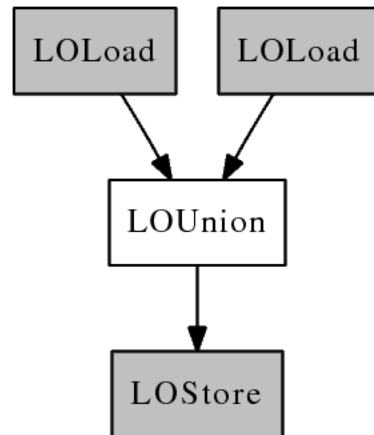
CoGroup

- Generalization of Group
 - Collects records of n inputs based on a key
 - The result is a record with a key and one bag for each input
 - It requires a reduce phase
-

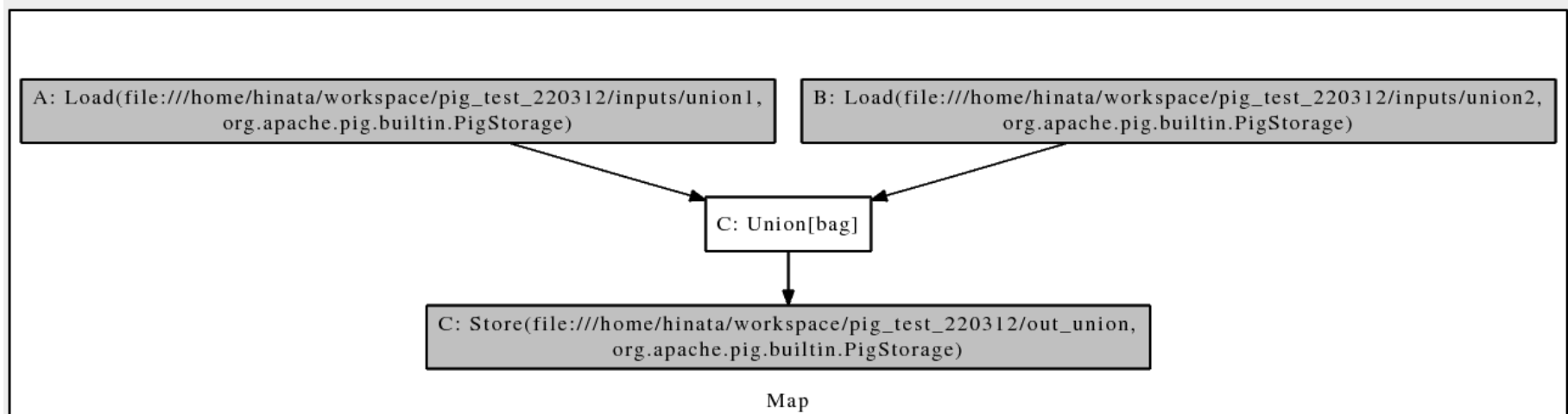
Union

- Concatenates 2 or more datasets without joining them
 - Inputs can have different schemas
 - Duplicates are not eliminated
 - Does not require a separate reduce phase
-

Union - Example



```
A = LOAD 'input1';  
B = LOAD 'input2';  
C = UNION A, B;  
STORE C INTO 'out_union';
```



Cross

- Takes every record of one input and matches it with every record in another input
 - Produces a lot of data
 - Given inputs with n and m records, the Cross result will have $n*m$ records
 - To implement Cross, Pig:
 - generates a synthetic key
 - replicates rows
 - performs the Cross as a Join
-

Cross

```
A = LOAD 'file1';  
B = LOAD 'file2';  
C = CROSS A, B;
```



CROSS = FOREACH + FOREACH + COGROUP + FOREACH

```
A = LOAD 'file1';  
B = LOAD 'file2';  
C = FOREACH A GENERATE FLATTEN(GFCross(0, 2)), FLATTEN(*);  
D = FOREACH B GENERATE FLATTEN(GFCross(1, 2)), FLATTEN(*);  
E = COGROUP C BY ($0, $1), D BY ($0, $1) parallel 10;  
F = FOREACH E GENERATE FLATTEN(C), FLATTEN(D);
```

Cross - The GFCross UDF

Input number

```
C = FOREACH A GENERATE FLATTEN(GFCross(0, 2)), FLATTEN(*);
D = FOREACH B GENERATE FLATTEN(GFCross(1, 2)), FLATTEN(*);
```

number of inputs

```
E = COGROUP C BY ($0, $1), D BY ($0, $1) parallel 10;
F = FOREACH E GENERATE FLATTEN(C), FLATTEN(D);
```

C {(3, 0), (3, 1), (3, 2), (3, 3)}

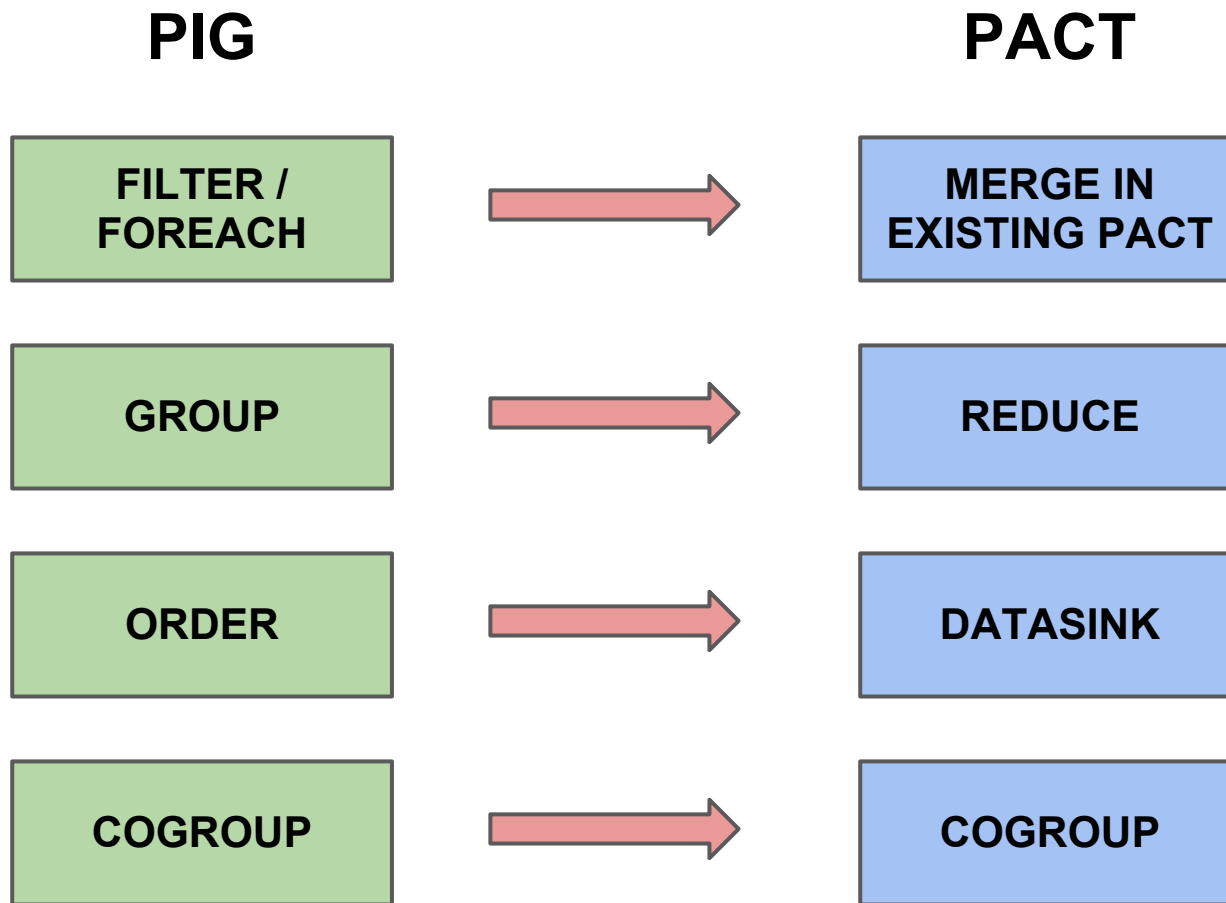
D {(0, 2), (1, 2), (2, 2), (3, 2)}

- The output bag contains 4 records, where 4 = $\text{round}(\text{sqrt}(10))$
- A random number between 0 and 3
- The other field counts from 0 to 3
- 4 copies of each record
- Only 1 match of the artificial keys

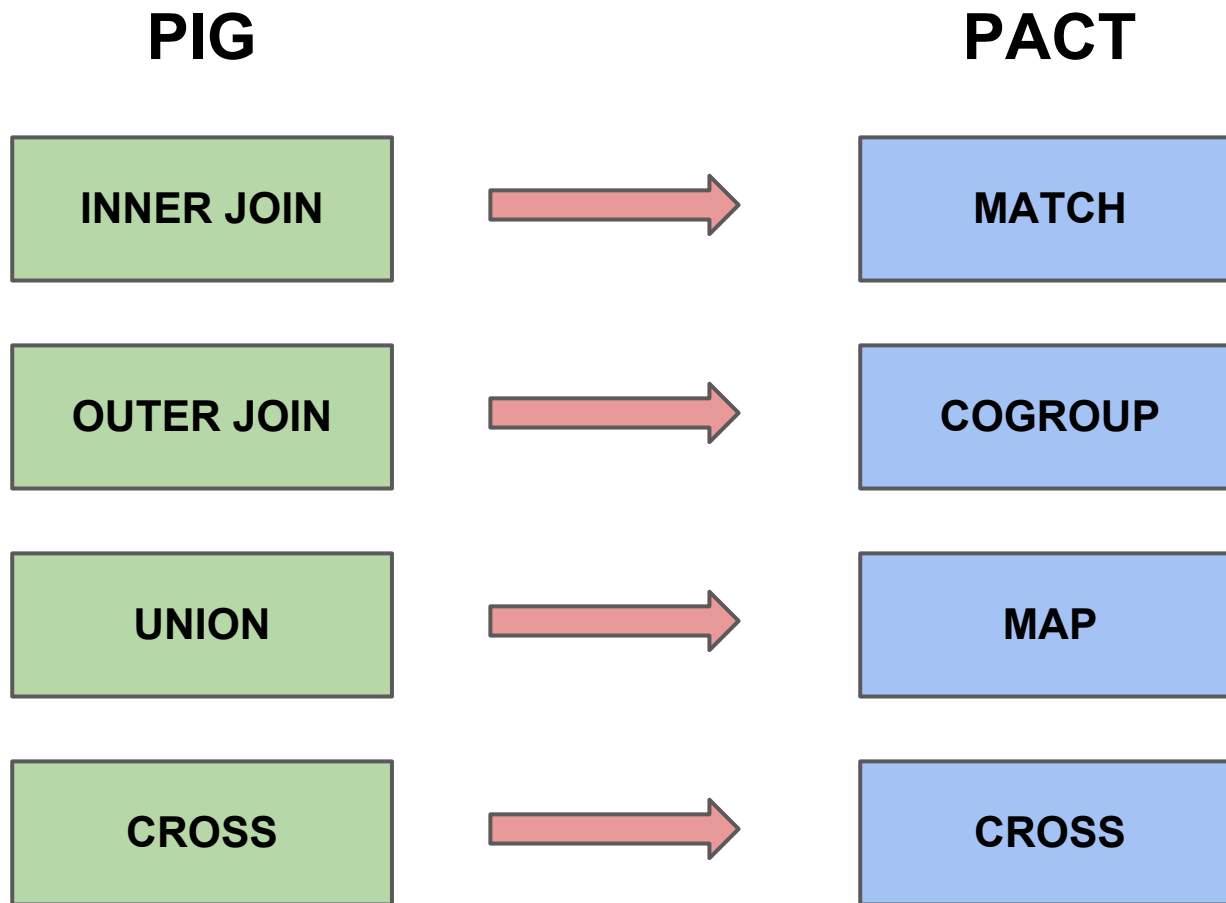
Outline

- Inspiration, Motivation and Goals
 - Pig
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - **Pig to PACT Compilation**
 - Current Status
 - What's Next
 - Future Work
-

Pig to PACT algorithm (1)



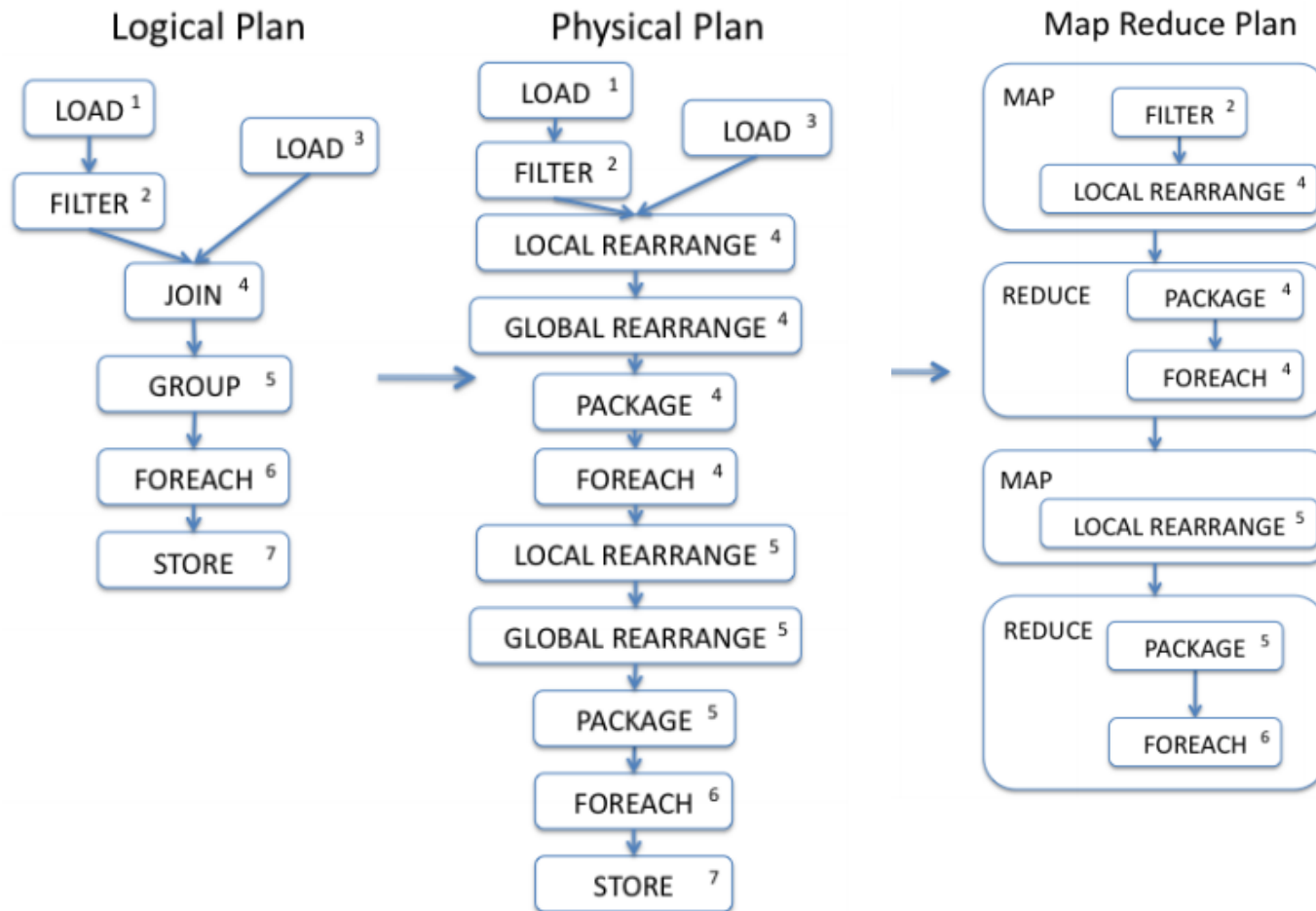
Pig to PACT algorithm (2)



Example

```
A = LOAD 'file1' AS (x, y, z);  
B = LOAD 'file2' AS (t, u, v);  
C = FILTER A BY y>0;  
D = JOIN C BY x, B BY u;  
E = GROUP D BY z;  
F = FOREACH E GENERATE group, COUNT(D);  
STORE F INTO 'out';
```

Example

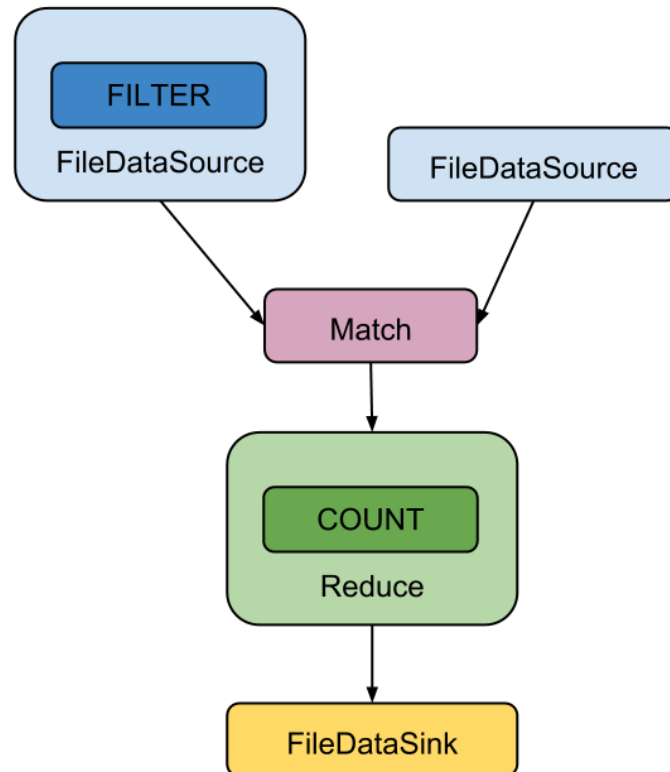


Example

Logical Plan



Pact Plan



Outline

- Inspiration, Motivation and Goals
 - Pig
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - **Current Status**
 - What's Next
 - Future Work
-

Implementation Challenges

- Pig was designed to be "independent" of the execution engine, but the line of separation is not clear
 - Started from MapReduce Plan trying to find equivalents for MROper, Job, JobControl etc.
 - Moved up to PhysicalPlan
 - Moved up to LogicalPlan
-

Implementation Challenges

- Dependencies even on high levels
 - Pig's loaders are tightly coupled to Hadoop's InputFormat
 - Re-implementing a lot of classes with only slight changes
 - Difficult to test functionality of small parts
-

Outline

- Inspiration, Motivation and Goals
 - Pig
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - **What's Next**
 - Future Work
-

Evaluation

- Based on Pig Performance Document from Pig Wiki
 - Test Data
 - 2 datasets of the same scheme
 - {name: string, age: integer, gpo: float}
 - 200 million rows and 10.000 rows
-

Evaluation

- Test Cases
 - Load/Store Baseline
 - Filter that removes 10% or 90% of the data
 - Foreach-Generate with basic arithmetic
 - (Co)Group
 - Default Join
 - Pig on Hadoop vs. Pig on Stratosphere
 - Pig on Stratosphere vs. native Map-Reduce
 - Pig on Stratosphere vs. native PACT
-

Outline

- Inspiration, Motivation and Goals
 - Pig
 - Integration Alternatives
 - Compilation of Basic Pig Operations
 - Pig to PACT Compilation
 - Current Status
 - What's Next
 - **Future Work**
-

Open Issues

- Implement all operators
 - DISTINCT, LIMIT, nested operators
 - Match vs. available Pig Join strategies
 - replicated
 - sort-merge
 - skew
 - Exploit Output Contracts
-

Evaluation with PigMix

- A set of queries to test Pig performance
 - latency
 - scalability
 - Includes a set of MapReduce Java programs to run equivalent MapReduce jobs directly
 - Used to measure the performance gap between running MapReduce directly and running Pig
 - 4 datasets are provided with zipf or uniform distribution
-

References and Links

- pig.apache.org
 - Introduction to Apache Pig,
<http://www.cloudera.com/?resource=introduction-to-apache-pig>
 - PigMix, <http://cwiki.apache.org/confluence/display/PIG/PigMix>
 - Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava, **Building a high-level dataflow system on top of map-reduce: the pig experience.**
 - Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins, Pig latin: **a not-so-foreign language for data processing.**
 - Alan F. Gates (September 2011), **Programming Pig: Dataflow Scripting with Hadoop**, O'Reilly Media
-