Hortonworks

Pig

Optimization and Execution

Alan F. Gates
@alanfgates

Who Am I?

- Pig committer and PMC Member
- HCatalog committer and mentor
- Member of ASF and Incubator PMC
- Co-founder of Hortonworks
- Author of *Programming Pig* from O'Reilly

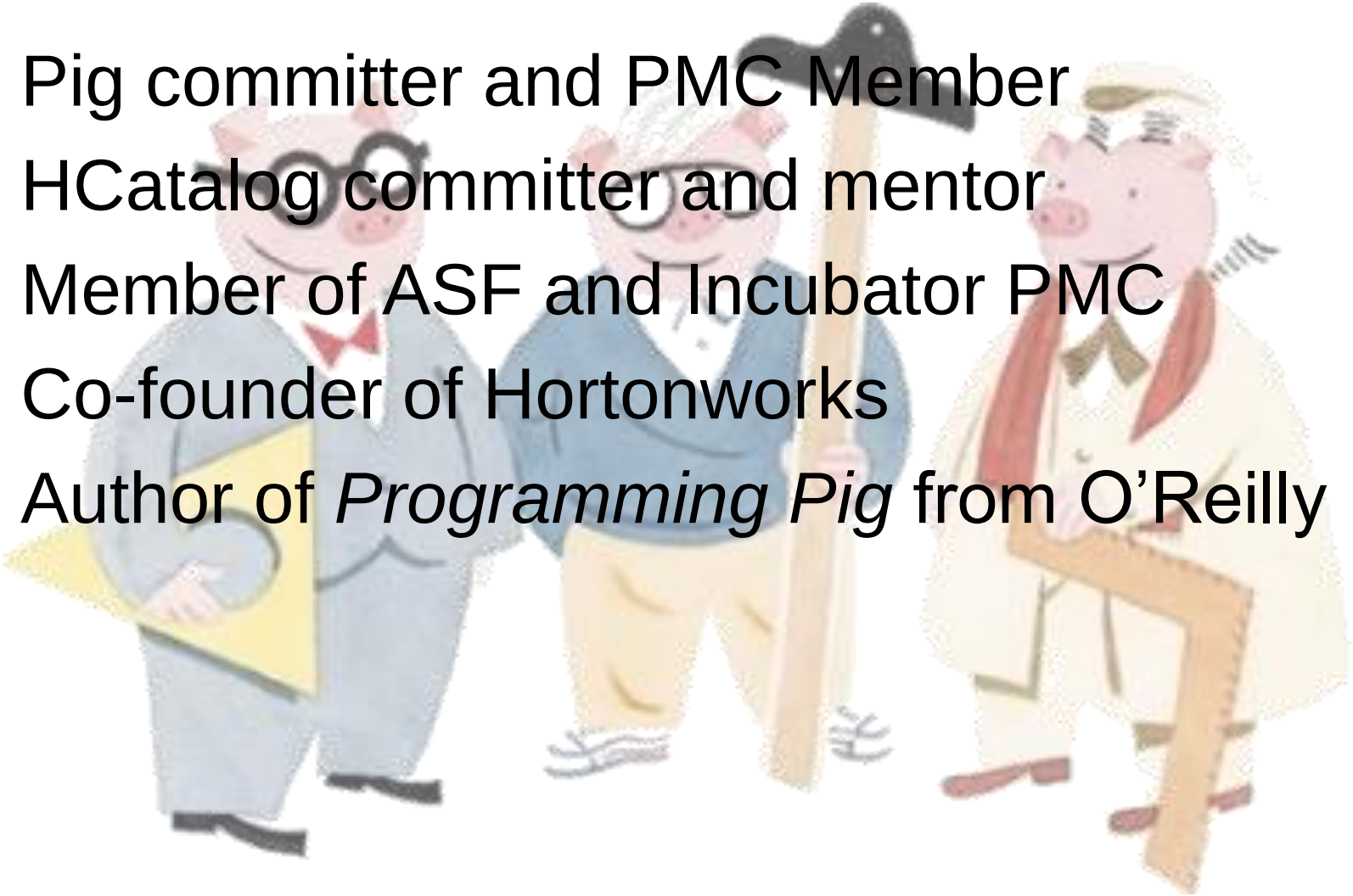


Photo credit: Steven Guarnaccia, *The Three Little Pigs*

Who Are You?

What Should We Optimize?

- Minimize scans – Hadoop is still often I/O bound
- Minimize total number of MR jobs
- Minimize shuffle size and number of shuffles
- Avoid spills to disk
- Reduce or remove skew
- For small jobs, minimize start-up time

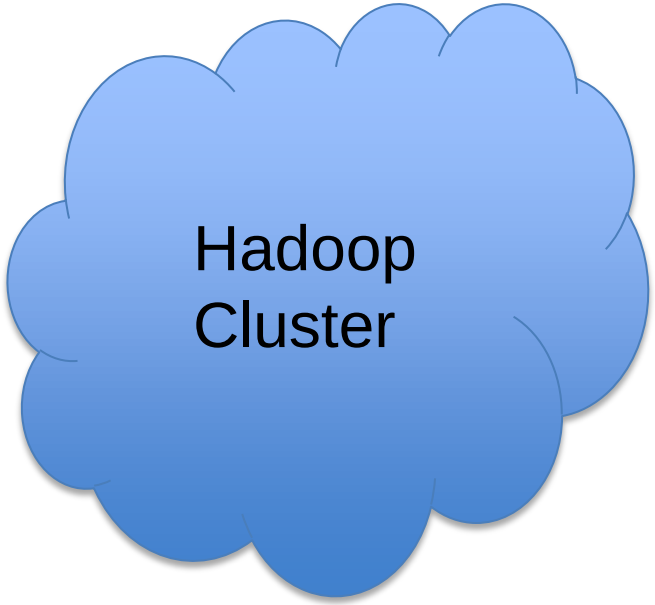
Pig Deployment

No server, all optimization and planning done on the launching machine

Job executes on cluster

Pig resides on user machine or gateway

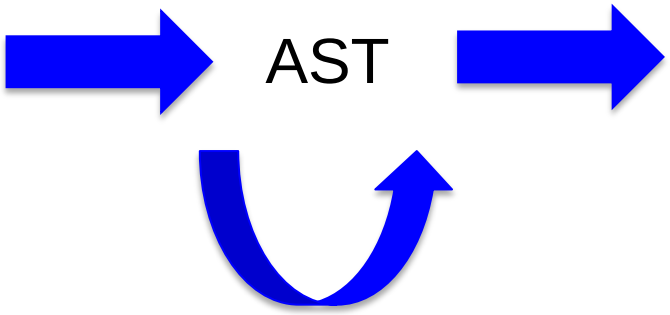
User machine



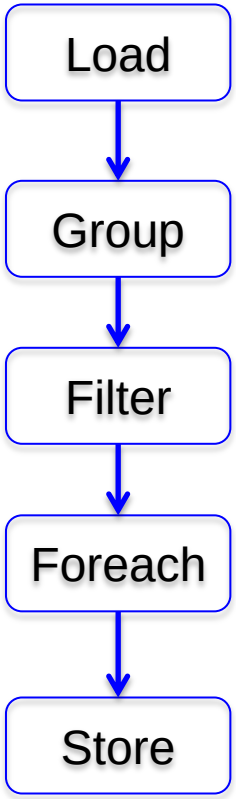
Pig Guts (i.e. Pig Architecture), p. 1

Pig Latin

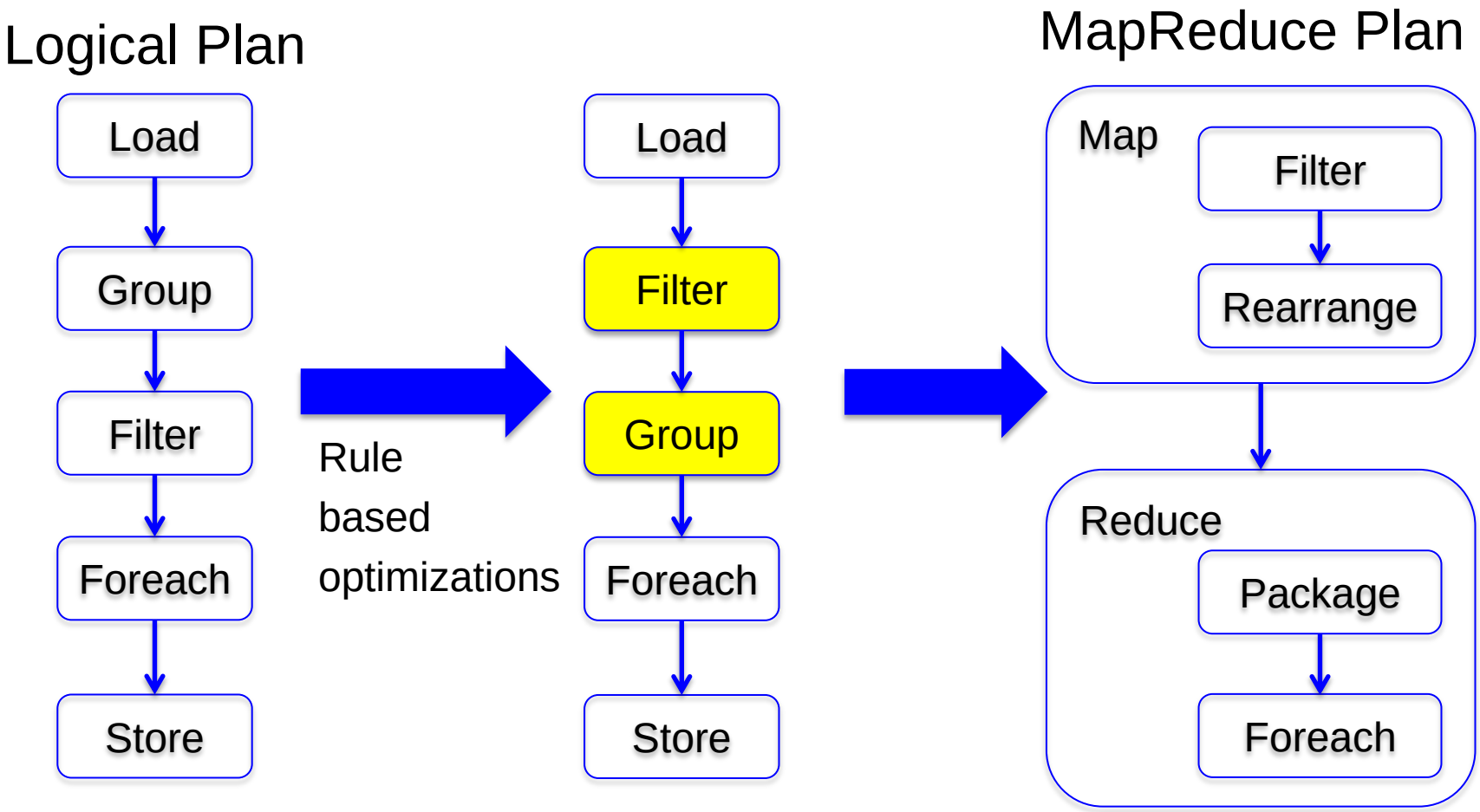
```
A = LOAD 'myfile'
    AS (x, y, z);
B = GROUP A by x;
C = FILTER B by
    group > 0;
D = FOREACH C GENERATE
    group, COUNT(A);
STORE D INTO 'output';
```



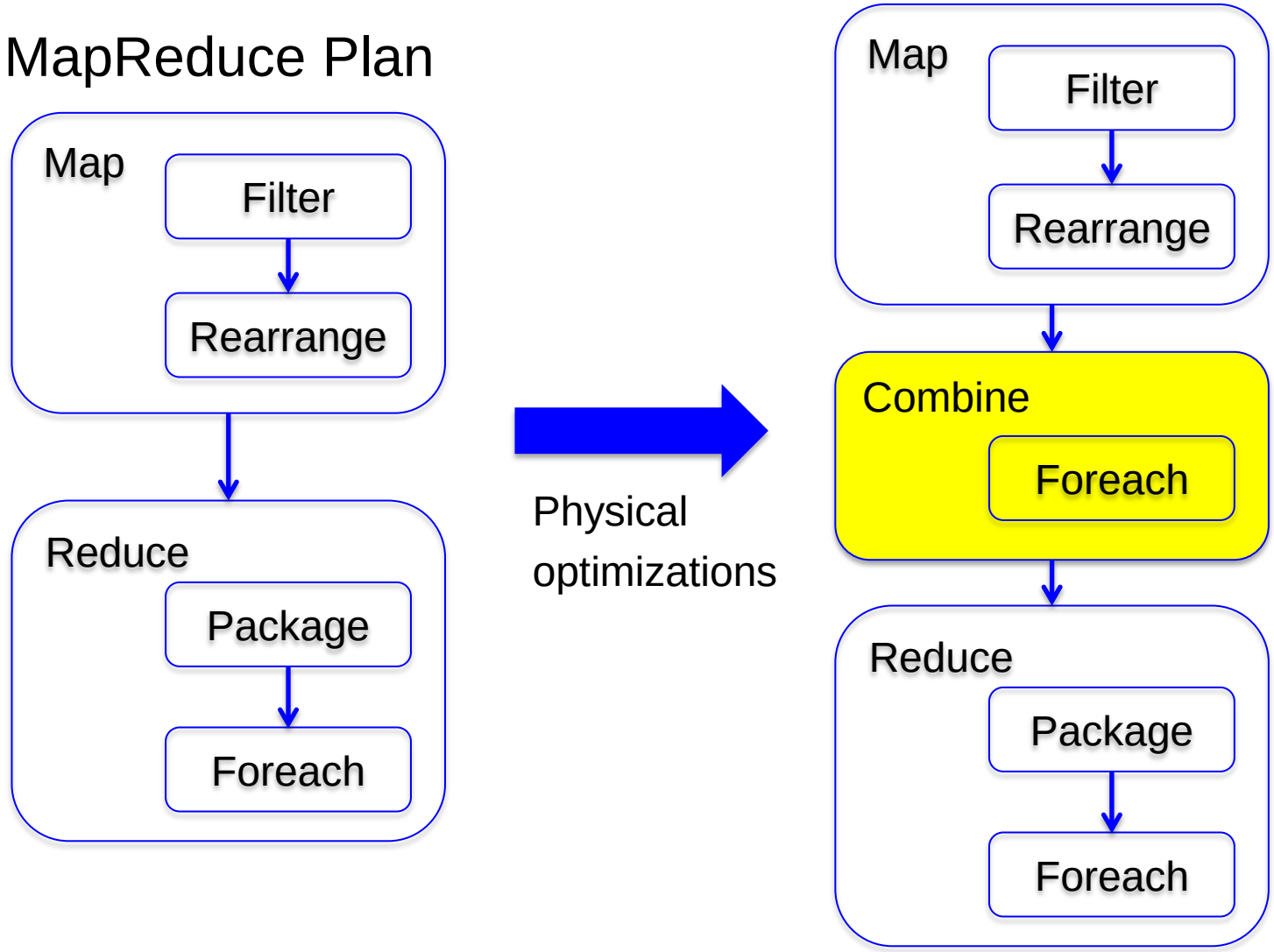
Logical Plan



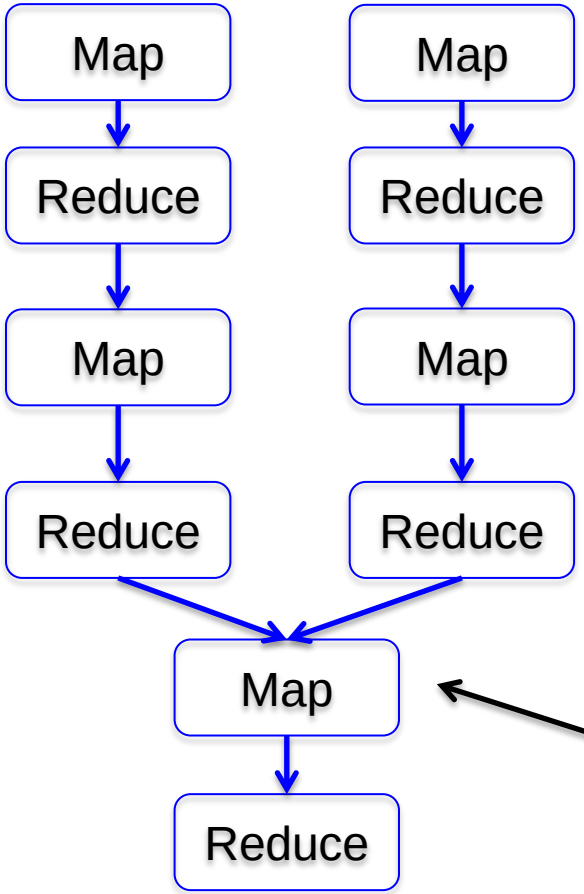
Pig Guts, p. 2


Logical Plan
Load
Group
Filter
Foreach
Store
Rule
based
optimizations
Load
Filter
Group
Foreach
Store
MapReduce Plan
Map
Filter
Rearrange
Reduce
Package
Foreach

Pig Guts, p. 3


MapReduce Plan
Map
Filter
Rearrange
Reduce
Package
Foreach
Physical optimizations
Map
Filter
Rearrange
Combine
Foreach
Reduce
Package
Foreach

It would be really cool if…



What's the right join algorithm here? Even with statistics it would be hard to know.

Need on the fly execution plan rewrites.

Memory

- Java + memory management = oil + water
  - Java types inefficient memory users (~4x disk size)
  - Very difficult to tell how much memory you are using
- Originally tried to monitor memory use via MXBeans: FAIL!
- Now estimate number of records we can hold in memory and spill when we exceed; allow user to tune guess

Reducing Spills to Disk

- Select Map size and io.sort.mb size such that 1 Map produces 1 Combiner
- Would be nice if Pig did this automatically
- Recent improvements: hash based aggregation in 0.10

Skew

- You are only as fast as your slowest reducer
- Data often power law distributed, means one reducer gets 10x+ the data of others
- Solution 1, use combiner whenever possible
- Solution 2, break rule that all records for a given key go to one reducer; works for order by and join

Reducing your Reducers

- Whenever possible use algorithms that can be done with no reduce
  - Fragment-replicate join
  - Merge join
  - Collected group

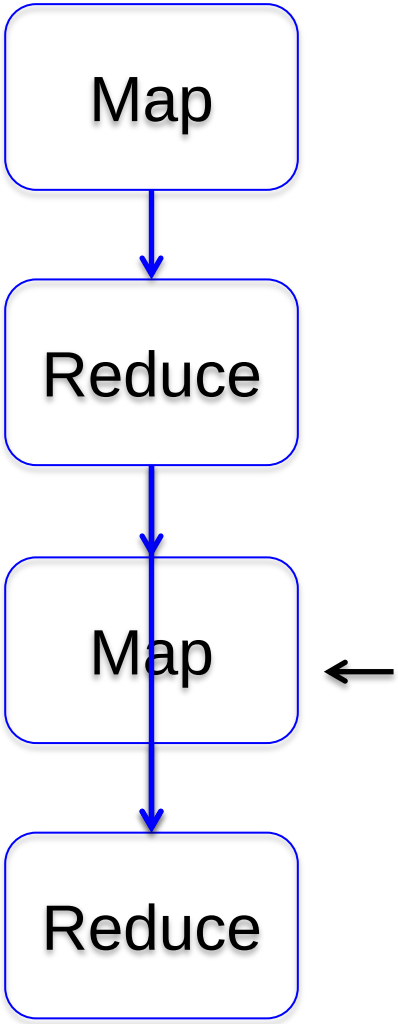
(De)serialization

- Data moves between memory and disk often
- Need to highly optimize, more work to be done here
- Need to do lazy deserialization

Faster Job Startup

- Should be using the distributed cache for Pig jar and UDFs
- For small jobs could use LocalJobRunner
- Need to try Tenzing approach of having a few tasks spun up and waiting for small jobs

Improved Execution Models



This is unnecessary. Anything that can be done in this map can be pushed to the previous reduce.

Need MR*

Code Generation

- Currently Pig physical operators are packaged in jar and pieced together on the backend to construct the data pipeline
- Tenzing and others have tried generating code on the fly instead, have seen significant improvements
- Downside, need javac on client machine

Learn More

- Read the online documentation: http://pig.apache.org/
- *Programming Pig* from O'Reilly Press
- Join the mailing lists:
  - user@pig.apache.org for user questions
  - dev@pig.apache.com for developer issues
- Follow me on Twitter, @alanfgates

Questions?