# Programming and Debugging Large-Scale Data Processing Workflows

Christopher Olston and many others

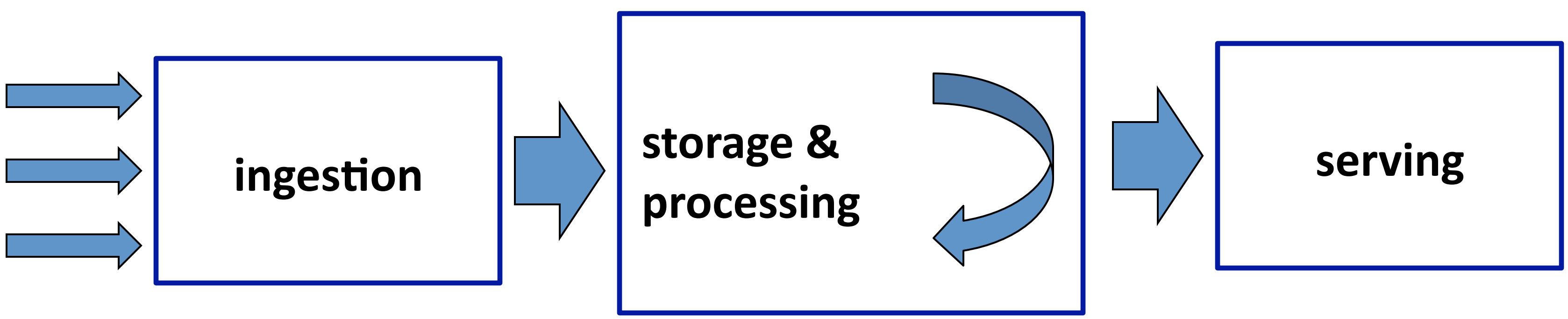
**Yahoo! Research**

# Context

- Elaborate processing of large data sets

  e.g.:

  - web search pre-processing
  - cross-dataset linkage
  - web information extraction



# Context

**storage & processing**

- workflow manager *e.g. Nova*
- dataflow programming framework *e.g. Pig*
- distributed sorting & hashing *e.g. Map-Reduce*
- scalable file system *e.g. GFS*

**Overview**

## Debugging aides:

- Before: example data generator
- During: instrumentation framework
- After: provenance metadata manager

**Detail:**
**Inspector Gadget**

# **Pig:** A High-Level Dataflow Language and Runtime for Hadoop

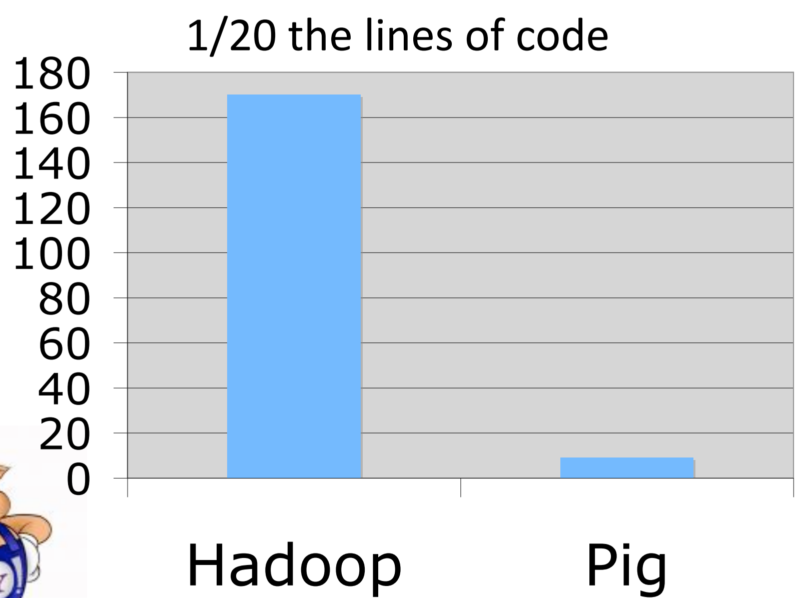
Web browsing sessions with "happy endings."

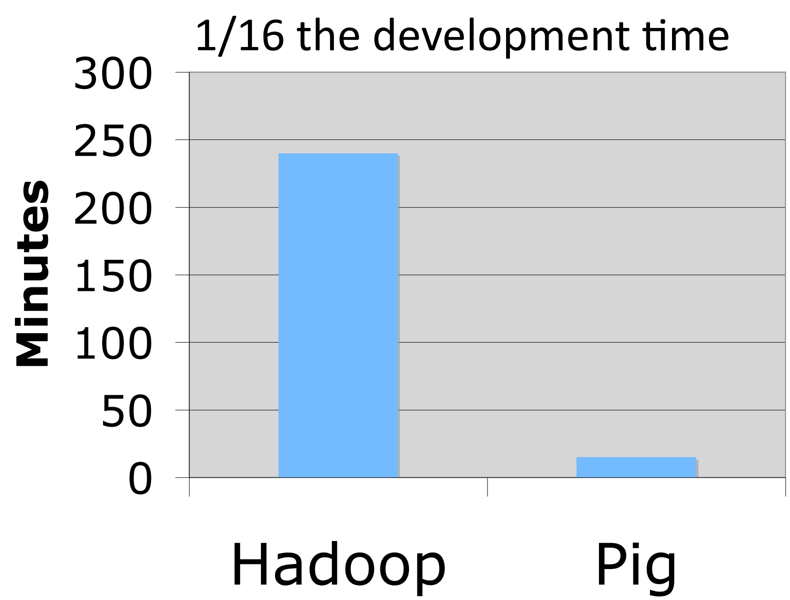
```
     Visits = load ‘/data/visits’ as (user, url, time);
     Visits = foreach Visits generate user, Canonicalize(url), time;

      Pages = load ‘/data/pages’ as (url, pagerank);

         VP = join Visits by url, Pages by url;
 UserVisits = group VP by user;
   Sessions = foreach UserVisits generate flatten(FindSessions(*));
HappyEndings = filter Sessions by BestIsLast(*);

      store HappyEndings into '/data/happy_endings';
```

# vs. map-reduce: less code!

"The [Hofmann PLSA E/M] algorithm was implemented in pig in 30-35 lines of pig-latin statements. Took a lot less compared to what it took in implementing the algorithm in Map-Reduce Java. Exactly that's the reason I wanted to try it out in Pig. It took 3-4 days for me to write it, starting from learning pig."

*-- Prasenjit Mukherjee, Mahout project*




**performs on par with raw Hadoop**

# vs. SQL:

## step-by-step style; lower-level control

"I much prefer writing in Pig [Latin] versus SQL. The step-by-step method of creating a program in Pig [Latin] is much cleaner and simpler to use than the single block method of SQL. It is easier to keep track of what your variables are, and where you are in the process of analyzing your data."

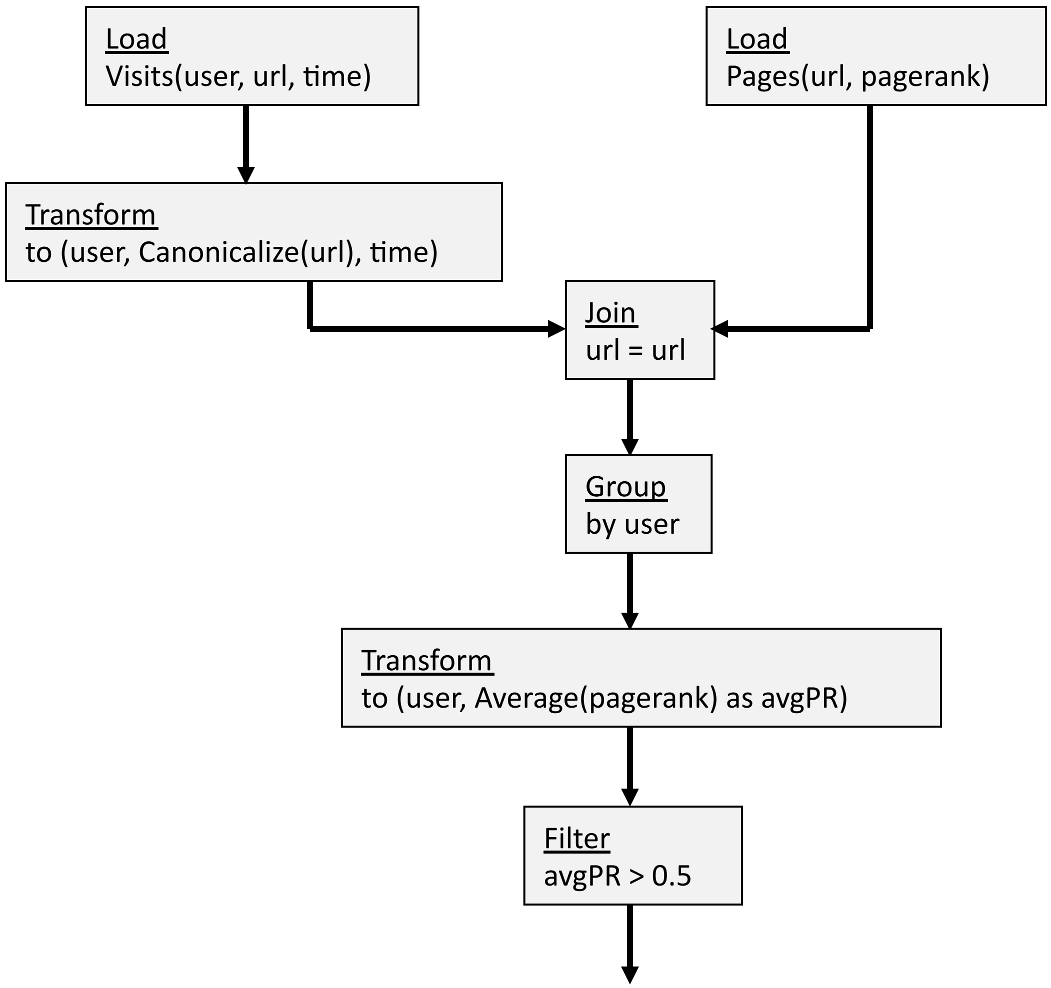
*-- Jasmine Novak, Engineer, Yahoo!*

"PIG seems to give the necessary parallel programming construct (FOREACH, FLATTEN, COGROUP .. etc) and also give sufficient control back to the programmer (which purely declarative approach like [SQL on top of Map-Reduce] doesn't)."

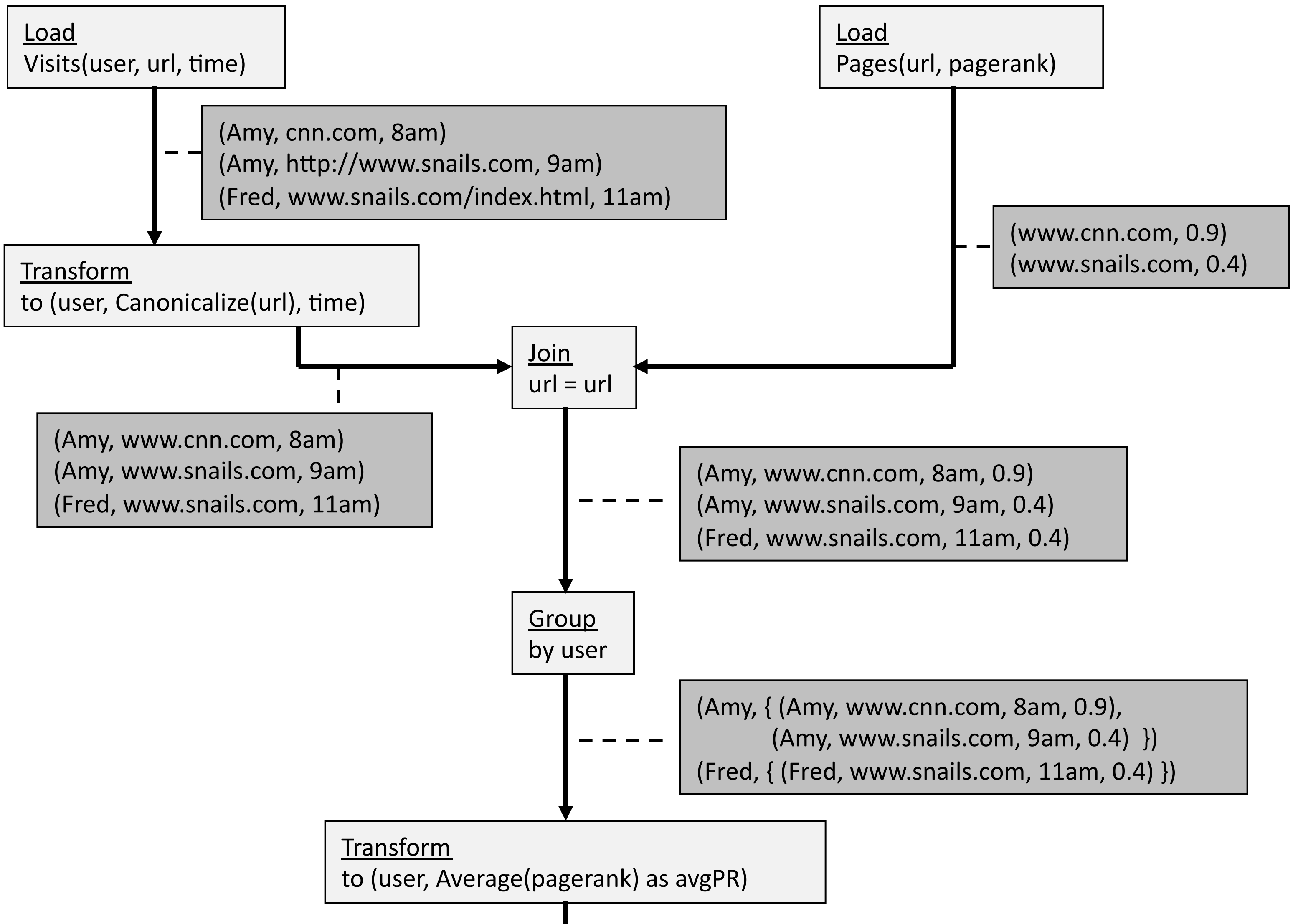
*-- Ricky Ho, Adobe Software*

# Conceptually: A Graph of Data Transformations

Find users who tend to visit "good" pages.



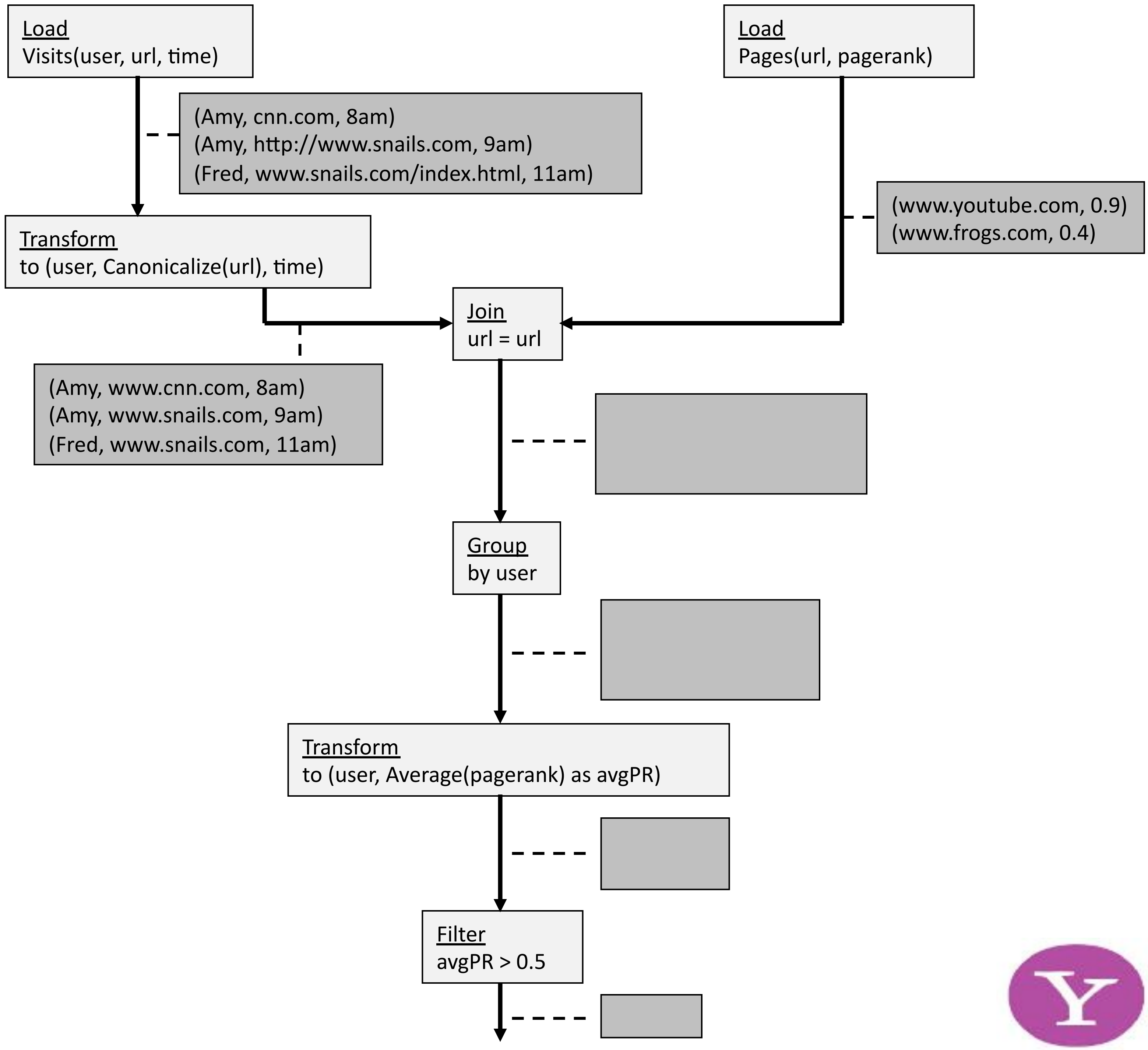
# Illustrated!



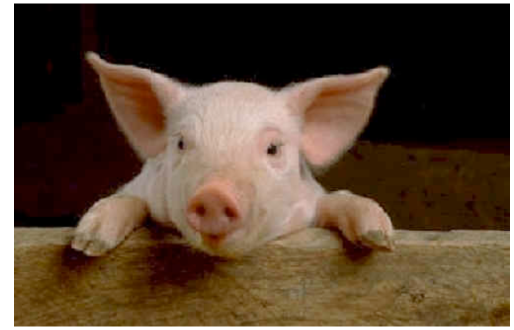

"ILLUSTRATE lets me check the output of my lengthy batch jobs and their custom functions without having to do a lengthy run of a long pipeline. [This feature] enables me to be productive."

*-- Russell Jurney, LinkedIn*

# (Naïve Algorithm)



# Pig Project Status



- Productized at Yahoo (~12-person team)
  - 1000s of jobs/day
  - 70% of Hadoop jobs
- Open-source (the Apache Pig Project)
- Offered on Amazon Elastic Map-Reduce
- Used by LinkedIn, Twitter, Yahoo, ...

# Next: NOVA

storage &
processing

workflow manager
*e.g. Nova*

dataflow programming framework ✔
*e.g. Pig*

distributed
sorting & hashing
*e.g. Map-Reduce*

scalable file system
*e.g. GFS*

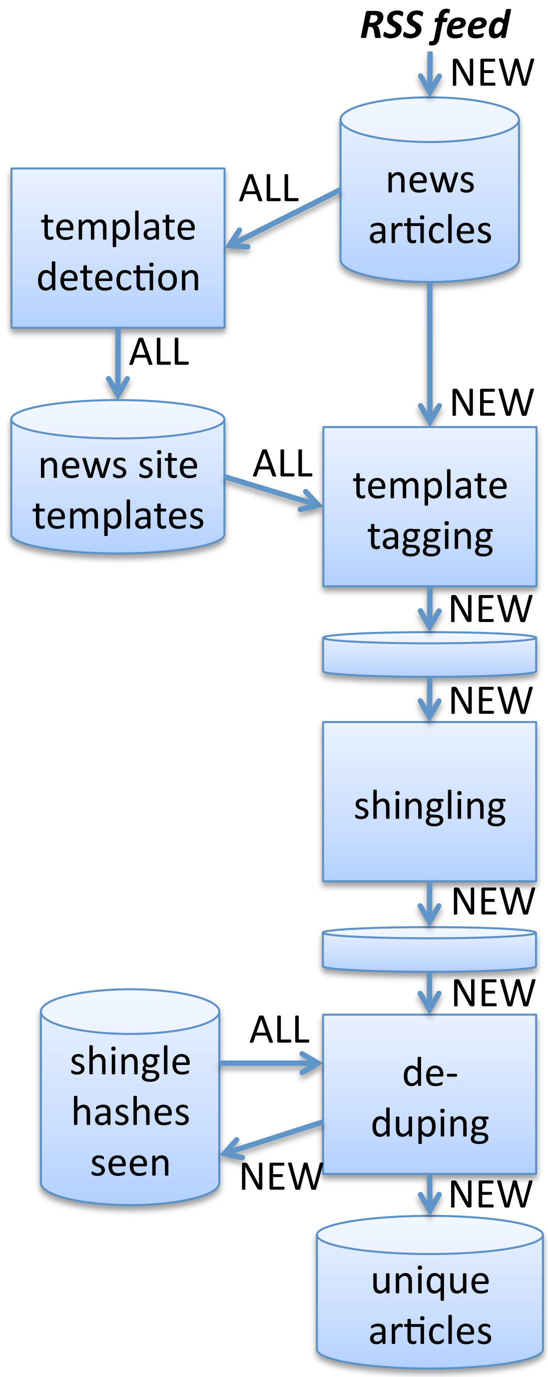
**Debugging aides:**

- Before: example data generator ✔
- During: instrumentation framework
- After: provenance metadata manager

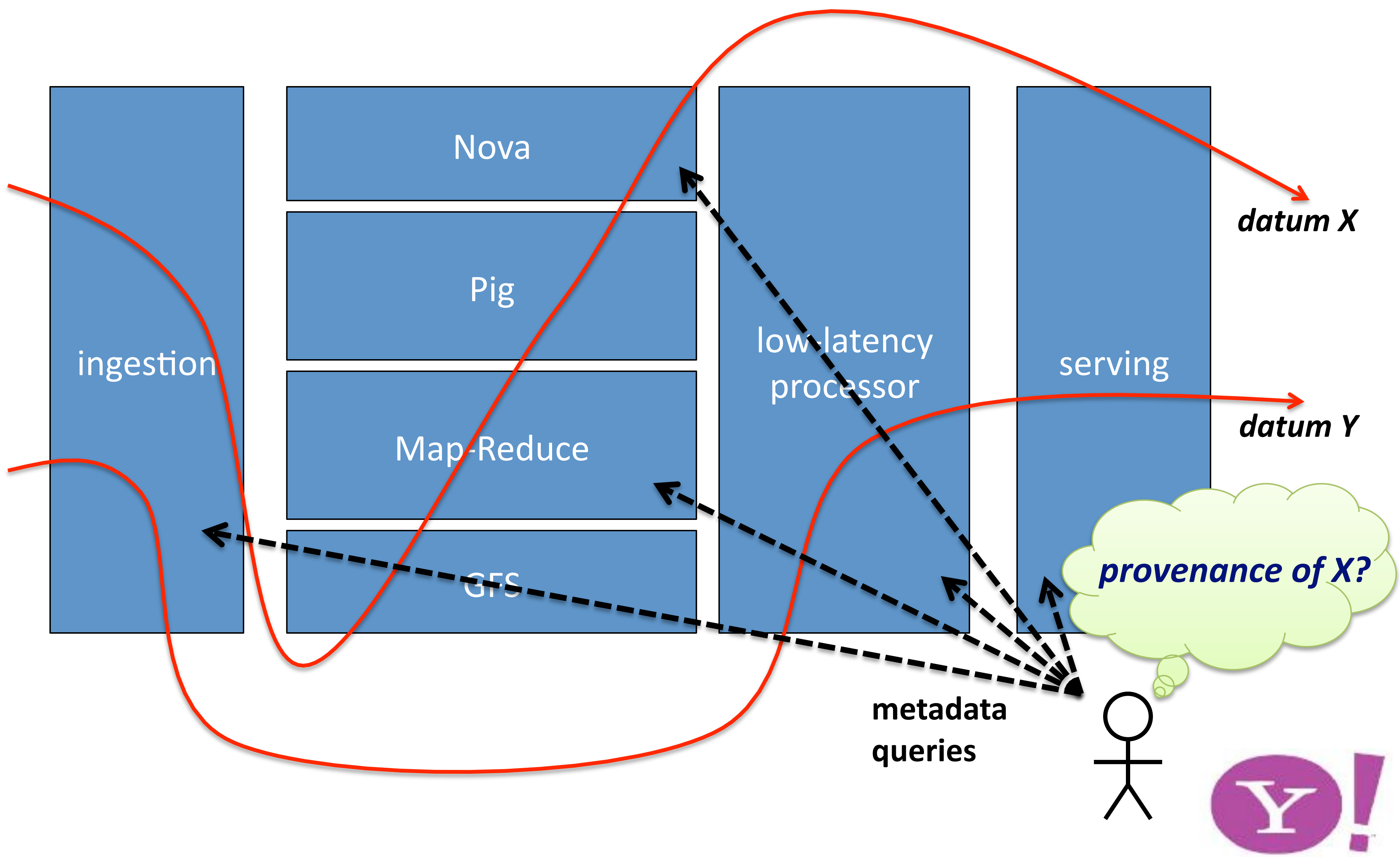
# Why a Workflow Manager?

- **Modularity:** a workflow connects N dataflow modules
  - Written independently, and re-used in other workflows
  - Scheduled independently
- **Optimization:** optimize across modules
  - Share read costs among side-by-side modules
  - Pipeline data between end-to-end modules
- **Continuous processing:** push new data through
  - Selective re-running
  - Incremental algorithms ("view maintenance")
- **Manageability:** help humans keep tabs on execution
  - Alerts
  - Metadata (e.g. data provenance)

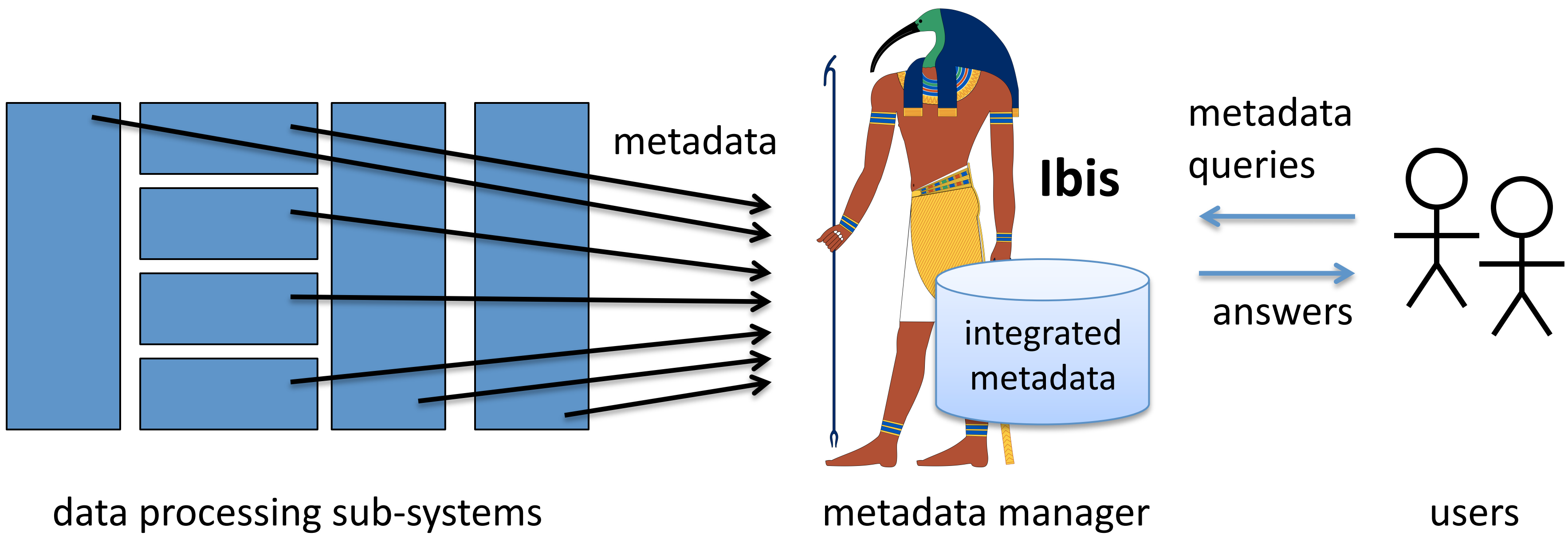
# Example Workflow



# Data Passes Through Many Sub-Systems



# Ibis Project




- Benefits:
  - Provide uniform view to users
  - Factor out metadata management code
  - Decouple metadata lifetime from data/subsystem lifetime
- Challenges:
  - Overhead of shipping metadata
  - Disparate data/processing granularities

# What's Hard About Multi-Granularity Provenance?

- ***Inference:*** Given relationships expressed at one granularity, answer queries about other granularities *(the semantics are tricky here!)*

- ***Efficiency:*** Implement inference without resorting to materializing everything in terms of finest granularity (e.g. cells)

# Next: INSPECTOR GADGET

storage & processing

- workflow manager *e.g. Nova*
- dataflow programming framework *e.g. Pig*
- distributed sorting & hashing *e.g. Map-Reduce*
- scalable file system *e.g. GFS*

**Debugging aides:**

- Before: example data generator
- During: instrumentation framework
- After: provenance metadata manager

# Motivated by User Interviews

- Interviewed 10 Yahoo dataflow programmers (mostly Pig users; some users of other dataflow environments)
- Asked them how they (wish they could) debug

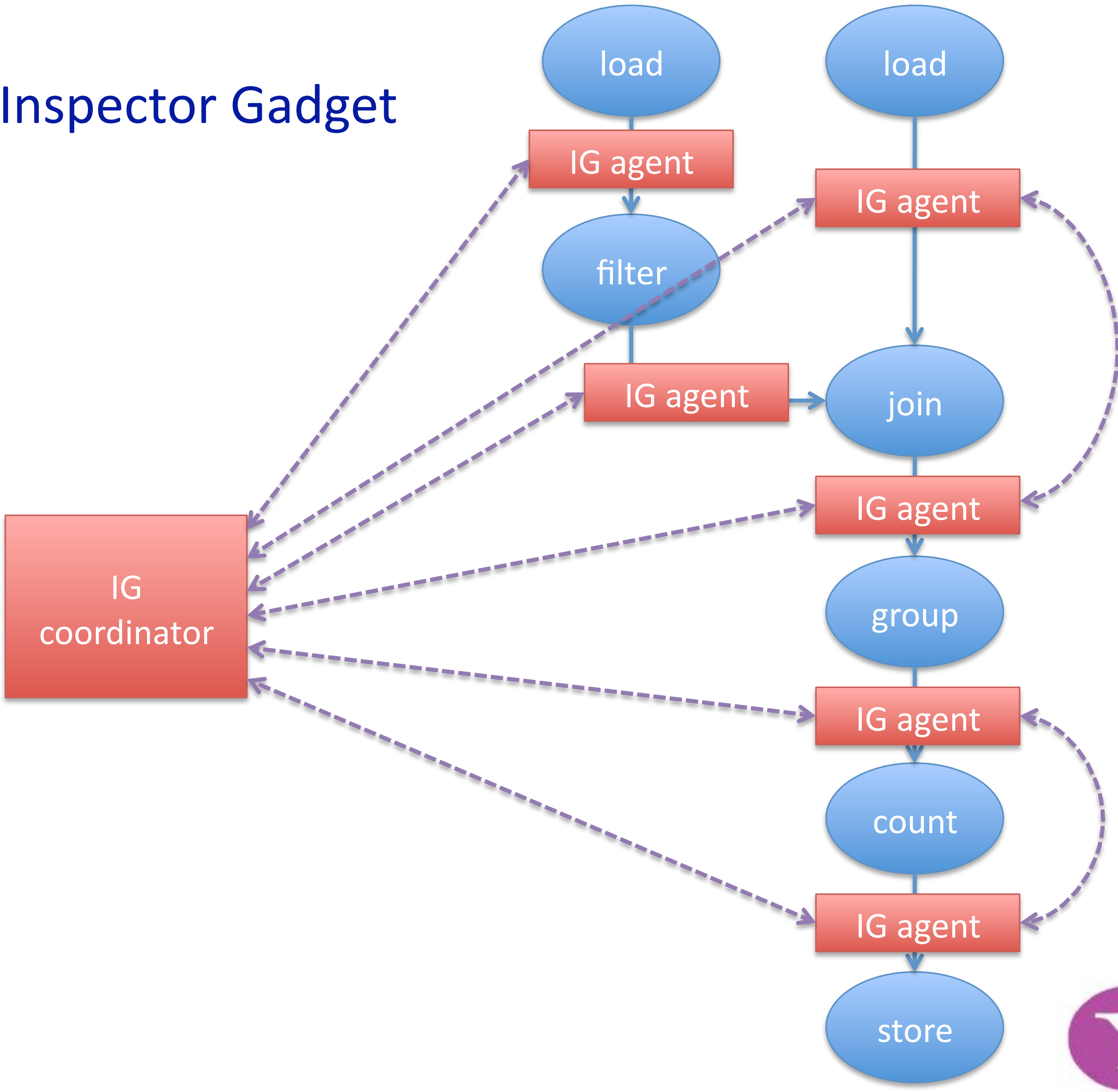
# Summary of User Interviews

| # of requests | feature |
|---|---|
| 7 | crash culprit determination |
| 5 | row-level integrity alerts |
| 4 | table-level integrity alerts |
| 4 | data samples |
| 3 | data summaries |
| 3 | memory use monitoring |
| 3 | backward tracing (provenance) |
| 2 | forward tracing |
| 2 | golden data/logic testing |
| 2 | step-through debugging |
| 2 | latency alerts |
| 1 | latency profiling |
| 1 | overhead profiling |
| 1 | trial runs |

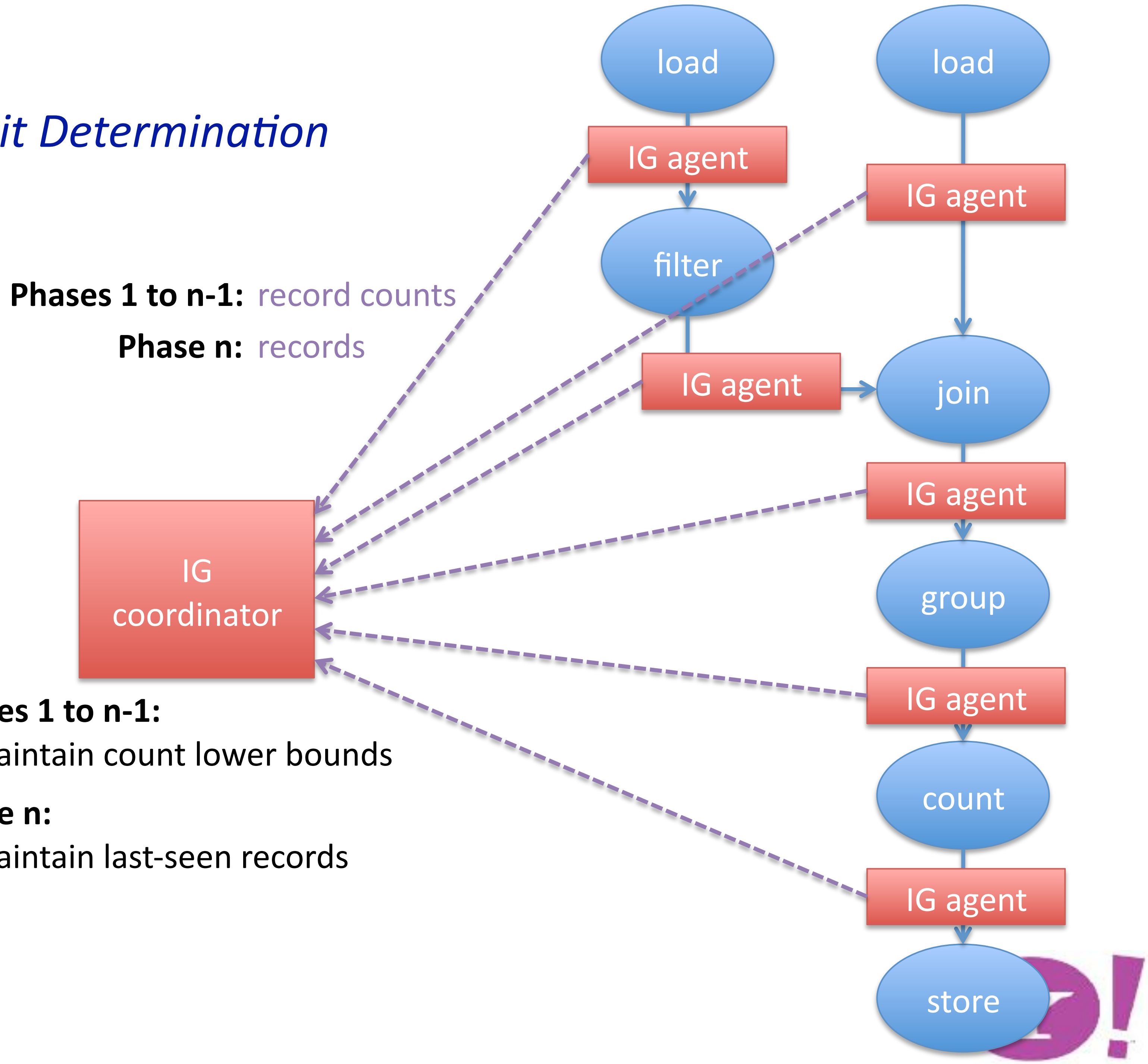
# Our Approach

- Goal: a programming framework for adding these behaviors, and others, to Pig
- Precept: avoid modifying Pig or tampering with data flowing through Pig
- Approach: perform Pig script rewriting – insert special UDFs that look like no-ops to Pig

# Pig w/ Inspector Gadget



# Example:
## *Crash Culprit Determination*



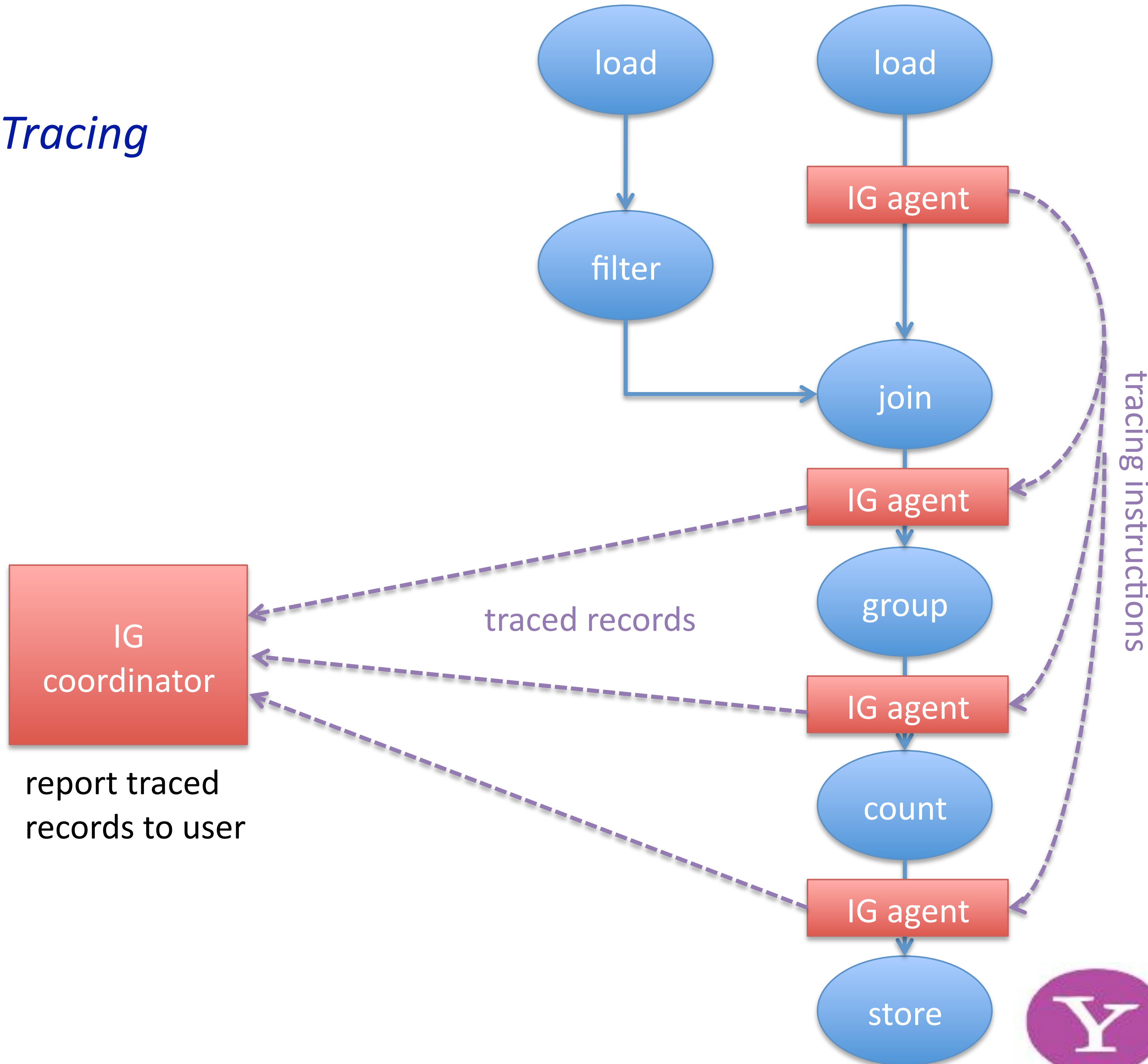

**Phases 1 to n-1:**

maintain count lower bounds

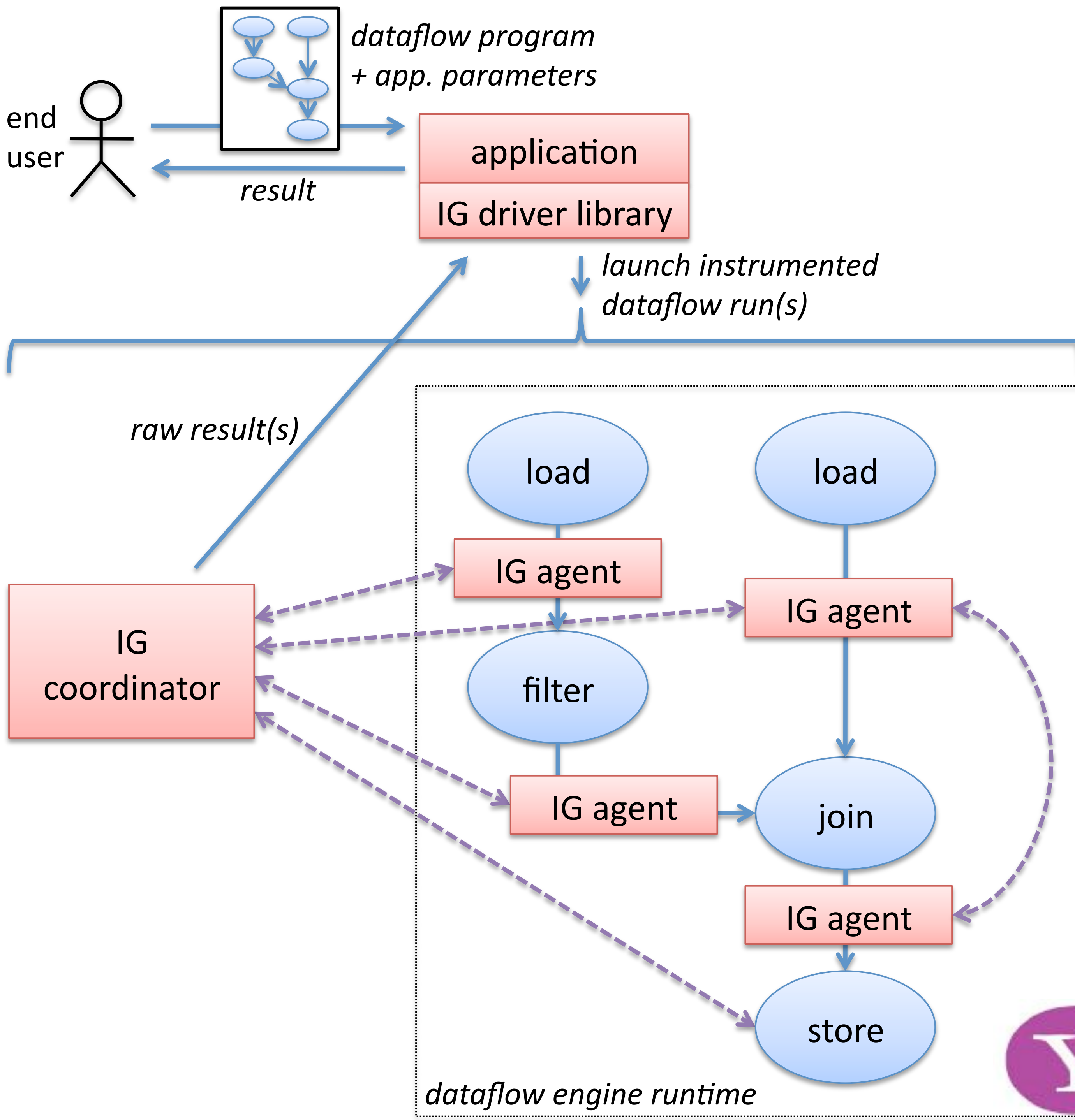
**Phase n:**

maintain last-seen records

# Example:
## *Forward Tracing*



# Flow



end
user
dataflow program
+ app. parameters
result
application
IG driver library
launch instrumented
dataflow run(s)
raw result(s)
IG
coordinator
load
load
IG agent
IG agent
filter
IG agent
join
IG agent
store
dataflow engine runtime

# Agent & Coordinator APIs

| Agent Class |
| --- |
| init(args) |
| tags = observeRecord(record, tags) |
| receiveMessage(source, message) |
| finish() |

| Agent Messaging |
| --- |
| sendToCoordinator(message) |
| sendToAgent(agentId, message) |
| sendDownstream(message) |
| sendUpstream(message) |

| Coordinator Class |
| --- |
| init(args) |
| receiveMessage(source, message) |
| output = finish() |

| Coordinator Messaging |
| --- |
| sendToAgent(agentId, message) |

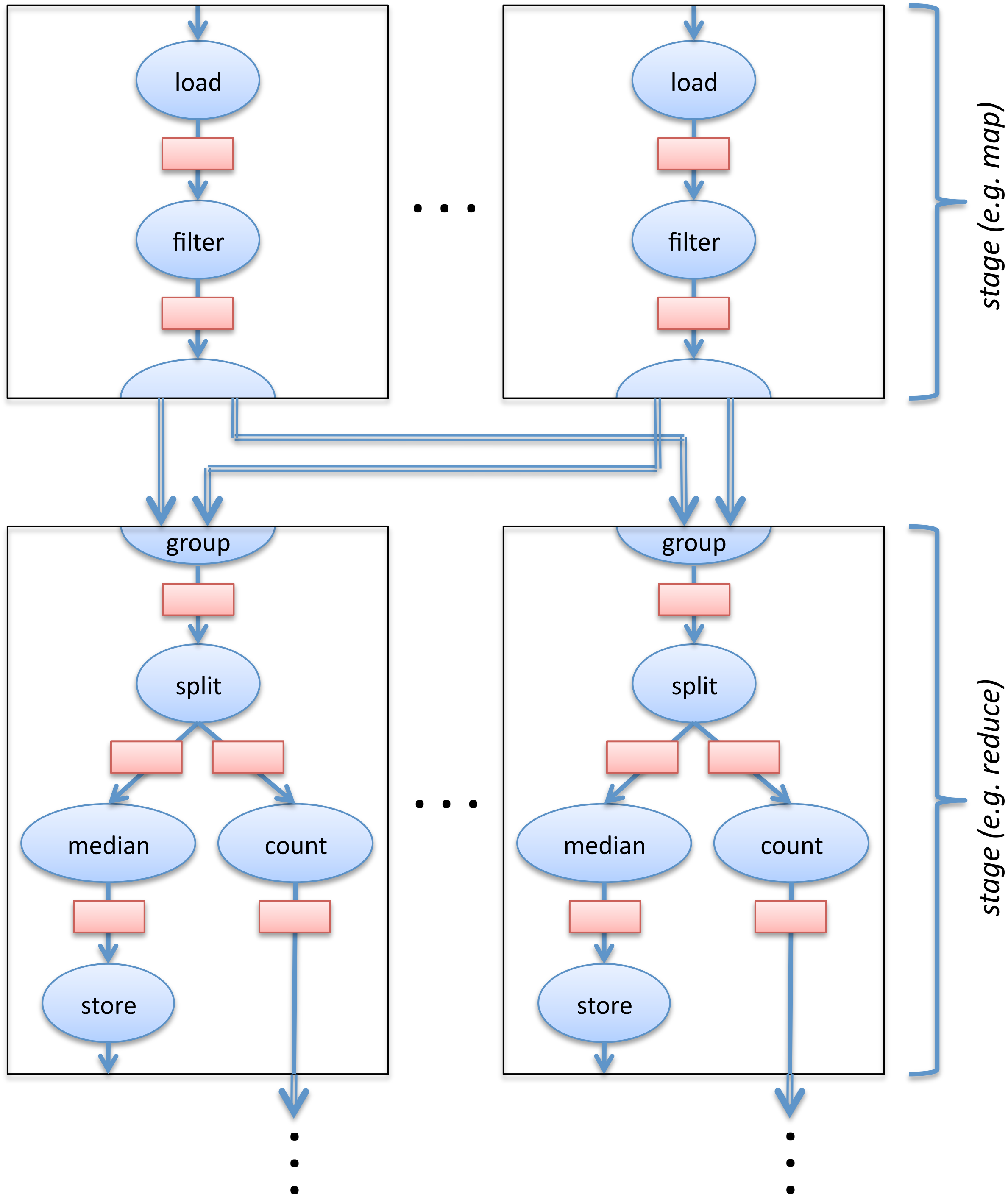
# Applications Developed Using IG

| # of requests | feature | lines of code (Java) |
|---|---|---|
| 7 | crash culprit determination | 141 |
| 5 | row-level integrity alerts | 89 |
| 4 | table-level integrity alerts | 99 |
| 4 | data samples | 97 |
| 3 | data summaries | 130 |
| 3 | memory use monitoring | N/A |
| 3 | backward tracing (provenance) | 237 |
| 2 | forward tracing | 114 |
| 2 | golden data/logic testing | 200 |
| 2 | step-through debugging | N/A |
| 2 | latency alerts | 168 |
| 1 | latency profiling | 136 |
| 1 | overhead profiling | 124 |
| 1 | trial runs | 93 |

# Rest of talk: IG DETAILS

- Semantics under parallel/distributed execution
- Messaging & tagging implementation
- Limitations
- Performance experiments
- Related work

# Parallel/Distributed Execution



# Messaging Details

- Semantics:

| Message Request | Semantics |
|---|---|
| sendToCoordinator(message) | asynchronous, guaranteed delivery |
| sendToAgent(agentId, message) | asynchronous, best-effort delivery |
| sendDownstream(message) | “follow the arrows,” guaranteed delivery |
| sendUpstream(message) | (same-stage only:) “invert the arrows,” guaranteed |

- Implementation:
  - Within-process: shared memory
  - Cross-process: relay through coordinator (coordinator buffers message for recipients that haven’t started yet)

# Tagging Implementation

- Uses messaging APIs
- Within-stage:
  - Leverage "iterator model" synchronous pipeline execution
    1. sendDownstream("tag future outputs with T"); release output record
    2. sendDownstream("stop tagging")
- Cross-stage:
  - Leverage Pig operator semantics (group-by, cogroup, join, order-by)
  - Group/cogroup: use group key
  - Join/order-by: use all record fields *(back-tags dups!)*

# Limitations of the IG Approach

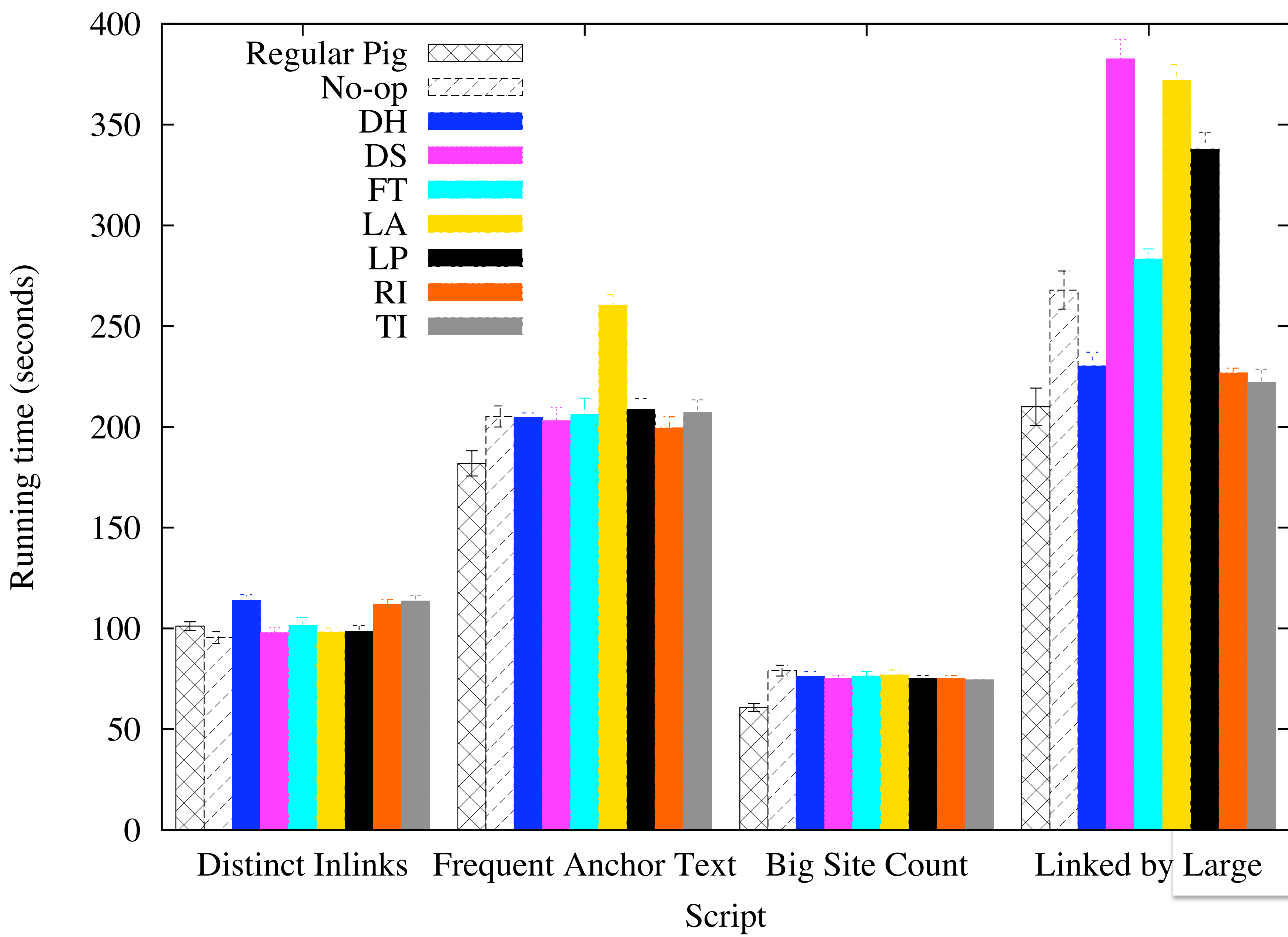
- Assumes query optimization nonexistent/disabled
- IG sits on top of Pig, so hard to correlate with lower-level logs/errors
- Crash/re-start results in record being seen by agents multiple times
  – Fortunately, all apps we've written can tolerate this, e.g. data only sent in finish(); rely on idempotence
- Tagging implementation not scalable
- Tagging implementation relies on Pig details

# Performance Experiments

- 15-machine Pig/Hadoop cluster (1G network)
- Four dataflows over a small web crawl sample (10M URLs):

| Dataflow Program | Early Projection Optimization? | Early Aggregation Optimization? | Number of Map-Reduce Jobs |
|---|---|---|---|
| Distinct Inlinks | N | N | 1 |
| Frequent Anchortext | Y | N | 1 |
| Big Site Count | Y | Y | 1 |
| Linked By Large | N | Y | 2 |

# Dataflow Running Times



# Summary

**storage & processing**

- workflow manager *e.g. Nova* ← **Nova**
- dataflow programming framework *e.g. Pig* ← **Pig**
- distributed sorting & hashing *e.g. Map-Reduce*
- scalable file system *e.g. GFS*

## Debugging aides:

- Before: example data generator ← **Dataflow Illustrator**
- During: instrumentation framework ← **Inspector Gadget**
- After: provenance metadata manager ← **Ibis**

# Related Work

- **Pig:** DryadLINQ, Hive, Jaql, Scope, *relational query languages*
- **Nova:** BigTable, CBP, Oozie, Percolator, *scientific workflow, incremental view maintenance*
- **Dataflow illustrator:** [Mannila/Raiha, PODS'86], *reverse query processing, constraint databases, hardware verification & model checking*
- **Inspector gadget:** XTrace, *taint tracking, aspect-oriented programming*
- **Ibis:** Kepler COMAD, ZOOM user views, *provenance management for databases & scientific workflows*

# Collaborators


Shubham Chopra
Anish Das Sarma
Alan Gates
Pradeep Kamath
Ravi Kumar
Shravan Narayanamurthy
Olga Natkovich
Benjamin Reed
Santhosh Srinivasan
Utkarsh Srivastava
Andrew Tomkins