ORACLE®

ORACLE

Early Experiences in Using a Domain-Specific Language for Large-Scale Graph Analysis

Sungpack Hong, Jan van der Lugt,
Adam Welc, Raghavan Raman,
Hassan Chafi

Oracle Labs

http://tinyurl.com/olabs-grades2013-2

Large-Scale Graph Analysis

- Analyzing large-scale graph data requires special frameworks
  - Data does not fit in single address space ➔ *Distributed computation*
  - Lots of random-access ➔ *Frequent communication*
- There are frameworks for large-scale graph analysis:
  - GraphLab (CMU), Pregel/Giraph (Google/Apache), Grappa (U. Washington), …
  - Each framework adopts its own API / programming model
- However, such programming models may differ from the way graph algorithm is designed

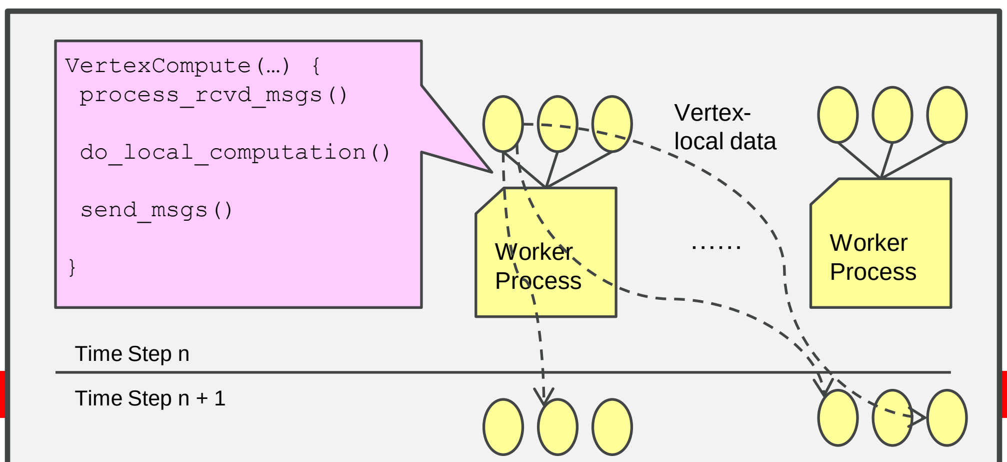
Giraph

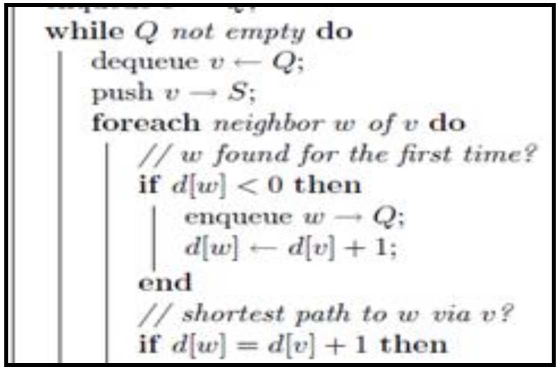
- A scalable graph analysis framework (Apache)
  - A clone of Google's Pregel [SIGMOD'10]
  - Running on top of Hadoop (HDFS)

- Giraph's programming model
  - Vertex-centric
  - Message-passing
  - Bulk-synchronous

≠

- Traditional algorithm design
  - Imperative
  - Random-access memory





```
while Q not empty do
    dequeue v ← Q;
    push v → S;
    foreach neighbor w of v do
        // w found for the first time?
        if d[w] < 0 then
            enqueue w → Q;
            d[w] ← d[v] + 1;
        end
        // shortest path to w via v?
        if d[w] = d[v] + 1 then
```

Our Approach: Domain-Specific Language

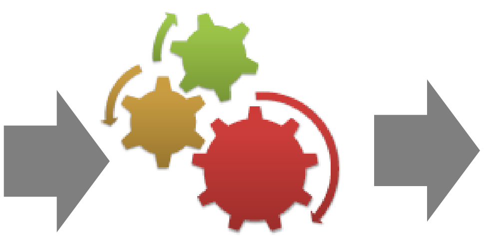
- Green-Marl
  - A DSL for graph analysis [ASPLOS 2012]
  - Designed for intuitive description of graph algorithms

Green-Marl

```
Do {
   diff = 0;
   Foreach(n: G.Nodes) {
     Double p_rank= (1-d) / N +
       d * Sum(w: n.InNbrs){
              w.PR/w.Degree()};
       …
   }
}
```

Imperative Random-Access Program

Green-Marl compiler



Giraph

Vertex-Centric,
Bulk-Synchronous,
Explicit Message Passing

```
class pagerankVertex extends …
{
  void compute(…)
  {  double p_val = (1-d) /N ;
     for ( message<…> m : Recvd)
     {
        p_val += m.getValueFloat()*d
     }
     sendNbrs(new Double(p_val) /
getNumNbrs);
  }
}
```


http://tinyurl.com/olabs-grades2013-2

Green-Marl Example: Pagerank

```
Procedure pagerank(G: Graph, e,d: Double,                // G is a graph
                   max: Int, PR: Node_Prop<Int>)         // PR is a node property
{
 Int iter = 0;
 Double diff = 0;
 Double N = (Double) G.NumNodes();
 G.PR = 1 / N;                                           // Initialize PR

 Do {                                                    // Main-loop
   diff = 0;
   iter ++;
   Foreach(n: G.Nodes) {                                 // For all nodes in G
     Double val = (1-d) / N +                            // compute pagerank by
         d * Sum(w: n.InNbrs){ w.PR/w.Degree()) };       // iterating neighborhoods

     diff += |n.PR - val|;                               // compute global difference
     n.PR <= val;                                        // update PR at the end of loop
   }
 } While (diff>e && iter<max);                           // loop until converged
}
```

C-like procedural syntax

High-level operations on abstract data-type

Easy and Intuitive Programming

ORACLE

Compiler Transformation

Explicit Loop Construction

Explicit parallel loops are detected / constructed

```
Procedure pagerank(G: Graph, … )
{
 Int iter = 0;
 Double diff = 0;
 Double N = (Double) G.numNodes();
 G.PR = 1 / N;

 Do {
   diff = 0;
   iter++;
   Foreach(n: G.Nodes) {
     Double val = (1-d) / N +
      d*Sum(w: n.InNbrs){w.PR/w.Degree()};

     diff += |w.PR - val|;
     w.PR <= val ;
   }
 } While ((diff>e) && (iter<max));
}
```

Syntax Expansion

```
{
 …
 Foreach(n: G.Nodes) {
   n.PR = 1 / N;
 }

 Node_Prop<Double> PR_nxt;

 Do {
   ...
   Foreach(n: G.Nodes) {
     Double _S = 0;
     Foreach(w: n.InNbrs)
        _S += w.PR/w.Degree();
     Double val = (1-d) / N + d* _S;
     ...
     W.PR_nxt = val;
   }
   Foreach(n: G.Nodes) {
      W.PR = W.PR_nxt;
   }
 } While ((diff>e) && (iter<max));
}
```

Compiler Transformation

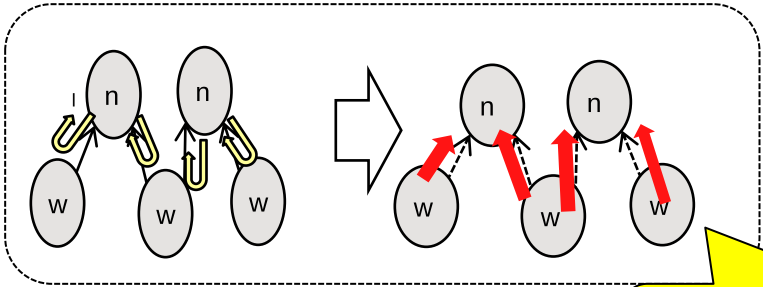
Optimization and Transformation

```
...

Foreach(n: G.Nodes) {
    Double  _S = 0;
    Foreach(w: n.InNbrs)
        _S += w.PR/w.Degree();
    Double val = (1-d) / N + d* _S;
    diff += |W.PR - val|;
    W.PR_nxt = val;
}

...
```

Apply a set of Transformation Rules

```
...
N_P<Double> _tmpS;
Foreach(n: G.Nodes) {
    n._tmpS = 0;
}
Foreach(w: G.Nodes) {
   Foreach(n: w.OutNbrs)
      n._tmpS += w.PR/w.Degree();
}
Foreach(n: G.Nodes) {
   Double val = (1-d)/N + d*n._tmpS;
   diff += |W.PR - val|;
   W.PR_nxt = val;
}

...
```

Pregel-Canonical Form

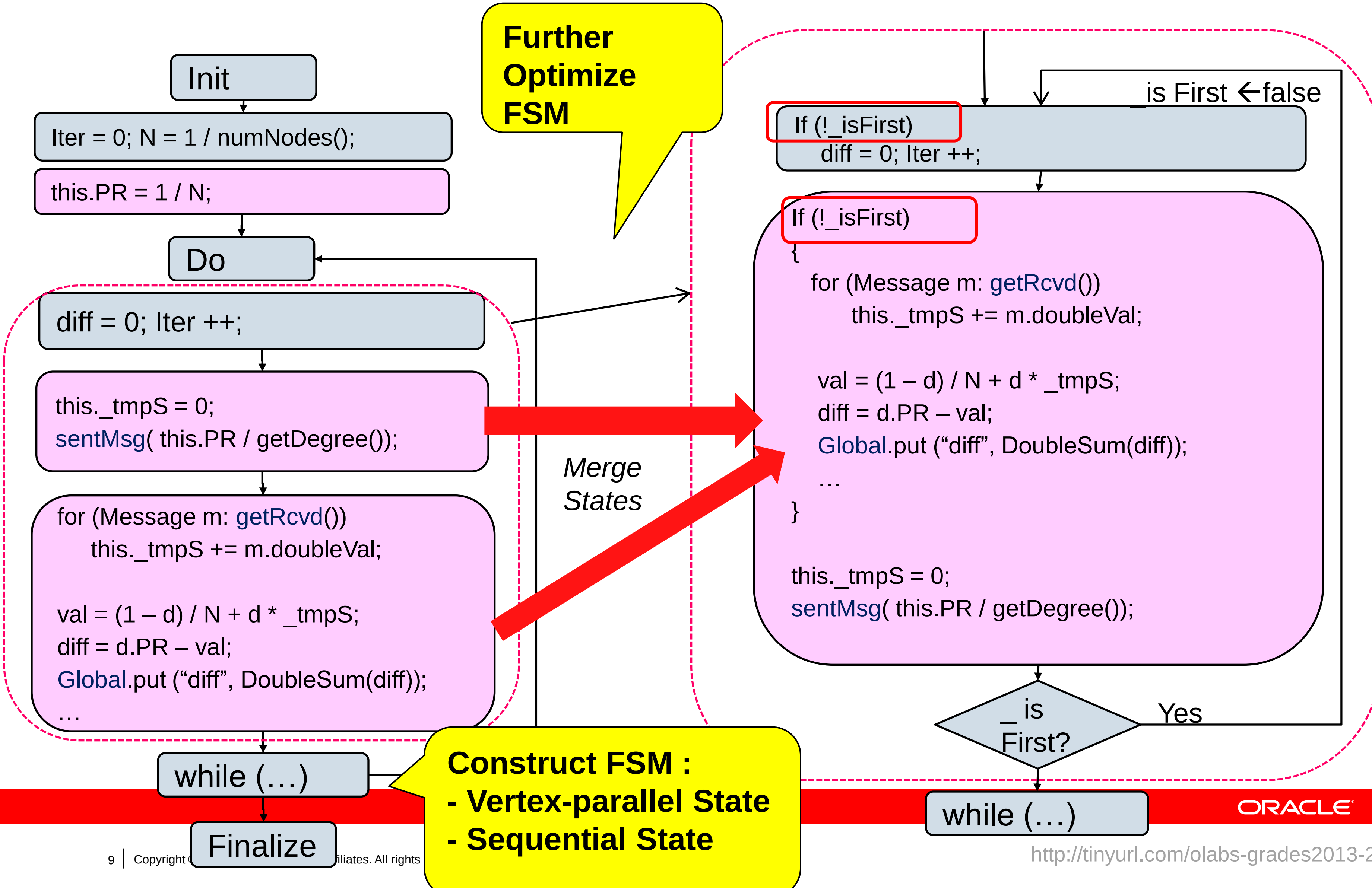


Remote read (Pull) is replaced with remote write (Push)

ORACLE
http://tinyurl.com/olabs-grades2013-2

Compiler Transformation

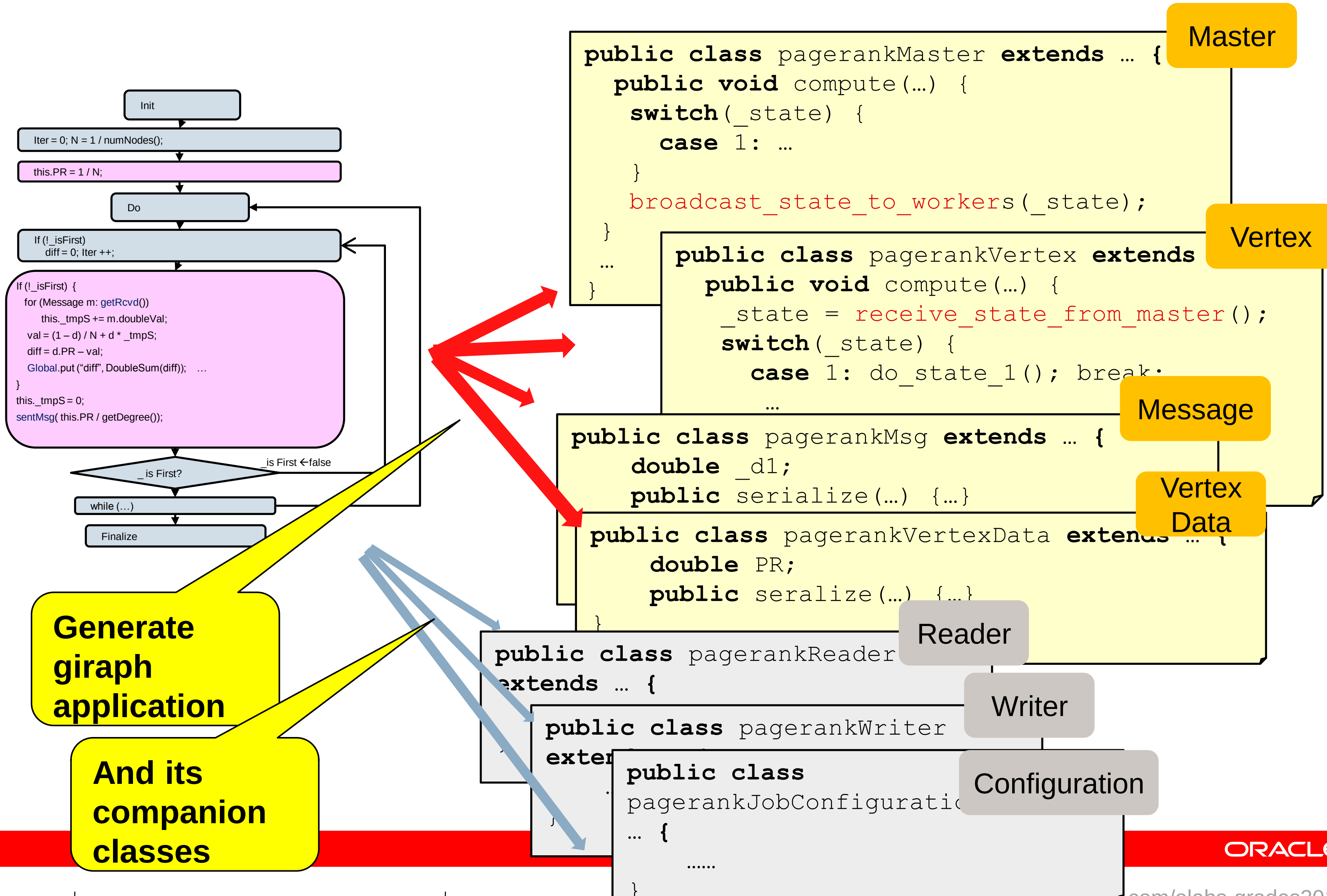
Finite State Machine (FSM) Construction



ORACLE
http://tinyurl.com/olabs-grades2013-2

Compiler Transformation

Code Generation



ORACLE
.com/olabs-grades2013-2

Early Experience and Evaluation

- Methodology

✓ Implement a few popular algorithms with Green-Marl

Pagerank,
Triangle Counting*,
Random Walk (Random Sampling)

✓ Feedbacks from external graph analysis experts

✓ Compile them into Giraph

✓ Compare them with manual Graph implenations

LiveJournal and Twitter Graph
Giraph run on 80 workers (10 machines)

Productivity Benefits and Challenges

- Shorter program

No boilerplate code at all

Line of Codes	G-M	Giraph (manual)
Pagerank	19	188
Triangle Counting	14	168
Random Walk Sampling	53	444

- Intuitive Programming Model
  - No low-level detail
  - Less error-prone
  - Ease of management

Benefits

- Learning Curve
  - Foreign language at first
  - Lack of user-community and documentation
- Inherently sequential algorithm:
  - There is no magic
  - The compiler emits translation failure (and why)

  ➔ User still needs to re-design the algorithm for giraph

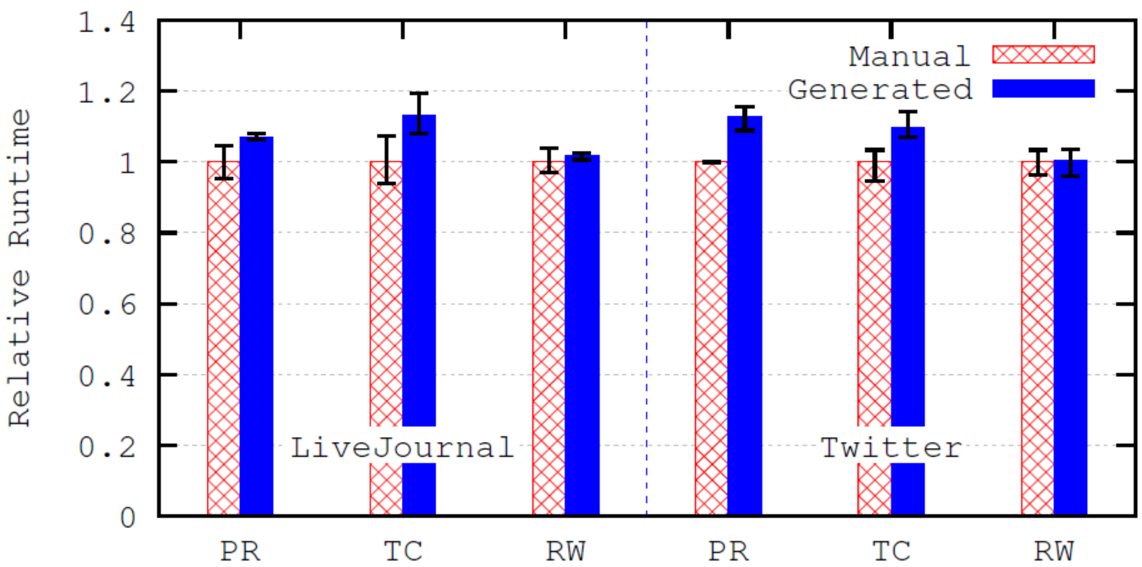
Challenges

ORACLE

Performance of Compiler-Generated Programs

- Decent Performance
  - 0 ~17 % slower than manual
  - TC: faster than first manual implementation (due to human error)

- Sub-optimal (yet)
  - Adding more optimization
- Cannot overcome fundamental limitation of the framework
  - *TC using Giraph crashes with high-degree nodes*.



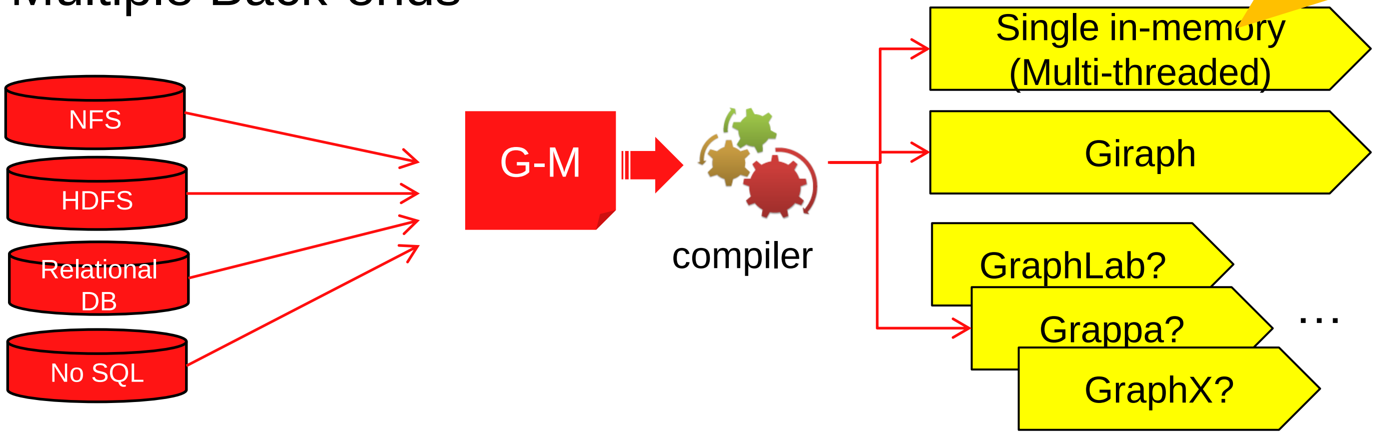
*For TC, we filtered out all high-degree nodes in the graph

ORACLE

Other issues and discussions

Fast execution for small(-er) graph

- Multiple Back-ends



- Multiple Data-source
  - Loading different formats
  - Declaring graphs from random relationship
  - Defining and filtering sub-graphs

"Let entities A becomes node, and relation B becomes edge."

Summary

- Using DSL for large-scale graph analysis
  - Demonstrated possibility
  - Promising productivity benefits
  - Decent performance against manual implementation; being improved
- Future works
  - Compiling into other backends
  - System Integration: graph data acquisition and management

ORACLE

Acknowledgement

- We appreciate Sam Shah, Roshan Sumbaly, and Evion Kim at LinkedIn for their valuable collaboration in this study.

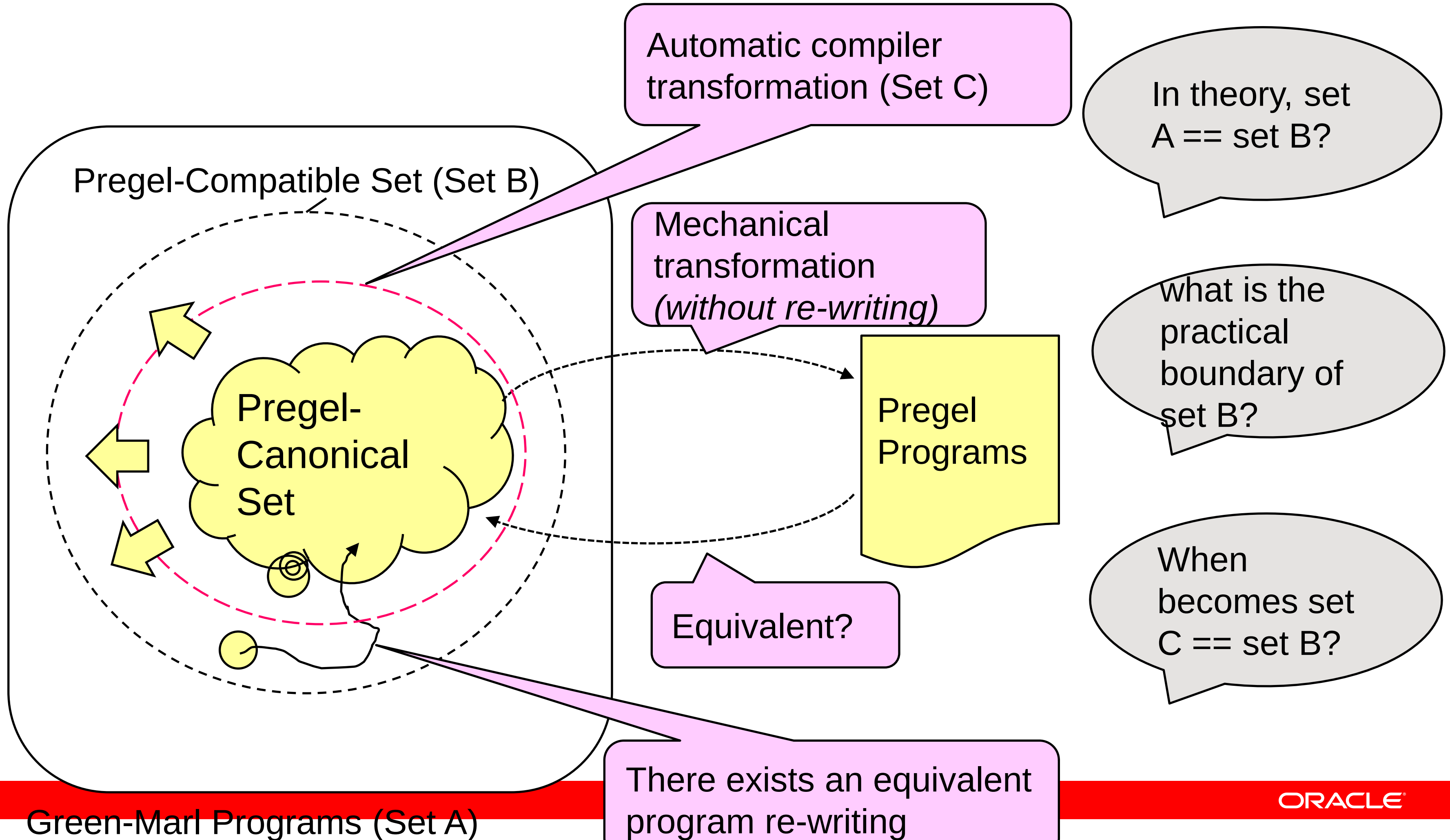
ORACLE

Hardware and Software

ORACLE

Engineered to Work Together

ORACLE®



http://tinyurl.com/olabs-grades2013-2