



The Myria Big Data Management and Analytics System and Cloud Service

Jingjing Wang, Tobin Baker, Magdalena Balazinska, Daniel Halperin,
Brandon Haynes, Bill Howe, Dylan Hutchison, Shrainik Jain, Ryan Maas,
Parmita Mehta, Dominik Moritz, Brandon Myers, Jennifer Ortiz, Dan
Suciu, Andrew Whitaker, Shengliang Xu

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF WASHINGTON

<http://myria.cs.washington.edu>

Acknowledgments

The Myria Team!

Our science collaborators!!

- Andrew Connolly, Tom Quinn, Sarah Loebman, Ariel Rokem, Ginger Armbrust, Yejin Choi

Our sponsors!!!

- National Science Foundation, Moore & Sloan Foundations, Washington Research Foundation, eScience Institute, ISTC Big Data, Petrobras, EMC, Amazon, and Facebook

Big Data

Management

Analytics

Science Apps

Efficient

Easy



Goals of the Myria stack

- Advance state-of-the-art in big data systems
- Focus on efficiency and productivity
- Test on real applications and support real users

Deliverables:

- Built a new **big data mgmt & analytics system**
- Deployed and operate **Myria as a service**
- Source code and demo service:
<http://myria.cs.washington.edu>



Myria has been developed and is operated by

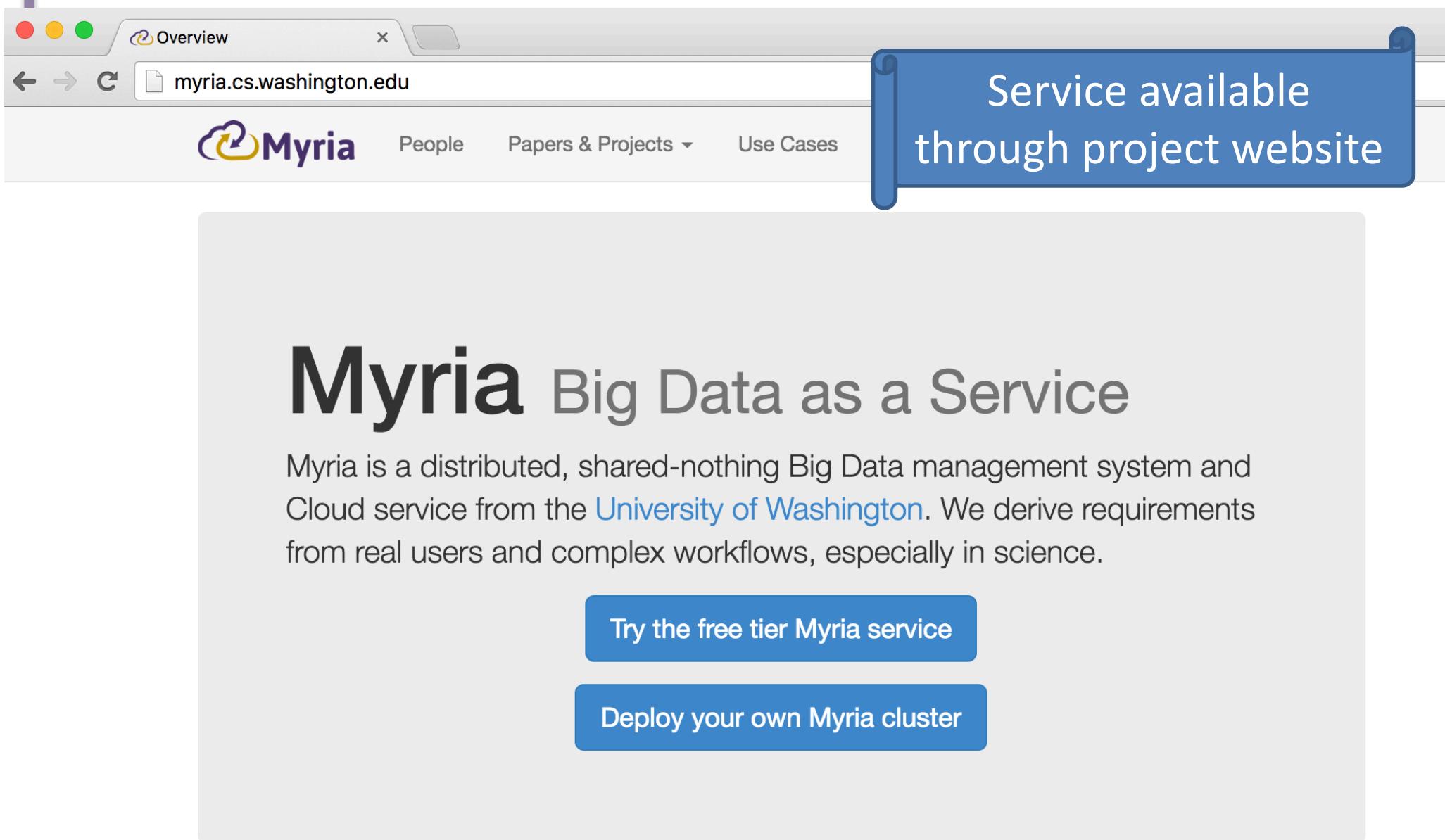
- Database Group in the Computer Science & Engineering Department at UW
- UW eScience Institute

Co-PIs: Dan Suciu and Bill Howe



Myria Demo

Myria Cloud Service



A screenshot of a web browser displaying the Myria Cloud Service website. The address bar shows 'myria.cs.washington.edu'. The page features a large 'Myria' logo with a cloud icon, and navigation links for 'People', 'Papers & Projects', and 'Use Cases'. A blue callout bubble on the right side contains the text 'Service available through project website'. The main content area has a light gray background and displays the heading 'Myria Big Data as a Service' in large, bold, dark font. Below it is a paragraph of text and two blue call-to-action buttons.

Service available through project website

Myria Big Data as a Service

Myria is a distributed, shared-nothing Big Data management system and Cloud service from the [University of Washington](#). We derive requirements from real users and complex workflows, especially in science.

[Try the free tier Myria service](#)

[Deploy your own Myria cluster](#)

Analysis in the Browser with Myria

Myria Web Frontend demo.myria.cs.washington.edu/editor Magdalena

Myria Editor Queries Datasets Report an issue 54.235.58.81:8753 [4/4]

Write your code here, perhaps starting from one of the examples at the right.

```
1 T1 = scan(TwitterK);
2 T2 = scan(TwitterK);
3 Joined = [from T1, T2
4     where T1.$1 = T2.$0
5     emit T1.$0 as src, T1.$1 as link, T2.$1 as dst];
6 store(Joined, TwoHopsInTwitter);
```

Execute the Query Parse Myria JSON

Query Language MyriaL

Developer Options

Declarative-imperative analysis with MyriaL and Python

Examples Datasets Query Plan Results

Visualization of the logical and optimized physical query plan.

Code parsed as Relational Algebra

Relational algebra converted and optimized into a Myria Physical Plan

```
graph TD
    subgraph Fragment2 [Fragment 2]
        S2[Scan(public:adhoc:TwitterK)] --> SP2[ShuffleProducer(h$11)]
    end
    subgraph Fragment3 [Fragment 3]
        S3[Scan(public:adhoc:TwitterK)] --> SP3[ShuffleProducer(h$00)]
    end
    subgraph Fragment1 [Fragment 1]
        SC1[ShuffleConsumer] --- SC2[ShuffleConsumer]
        SC1 --- SJ[SymmetricHashJoin($1 = $2); $0,$1,$3]
        SC2 --- SJ
    end
    SP2 -.-> SC1
    SP3 -.-> SC2
    SJ --> SP4[SplitProducer]
```

Myria Operates Directly on Data in S3

Write your code here, perhaps starting from one of the examples at the right.

```
1 T1 = load("https://s3-us-west-  
2 .amazonaws.com/uwdb/sampleData/TwitterK.csv",  
3 csv(schema(a:int, b:int), skip=0));  
3 store(T1, TwitterK, [a, b]);
```

▶ Execute the Query

Parse

Myria JSON

For efficient processing, caches query results internally in cluster

MyriaL is Imperative+Declarative with Iterations

Write your code here, perhaps starting from one of the examples at the right.

```
1 E = scan(TwitterK);
2 V = select distinct E.$0 from E;
3 CC = [from V emit V.$0 as node_id, V.$0 as component_id];
4 do
5     new_CC = [from E, CC where E.$0 = CC.$0 emit E.$1,
CC.$1] + CC;
6     new_CC = [from new_CC emit new_CC.$0, MIN(new_CC.$1)];
7     delta = diff(CC, new_CC);
8     CC = new_CC;
9 while [from delta emit count(*) > 0];
10 comp = [from CC emit CC.$1 as id, count(CC.$0) as cnt];
11 store(comp, TwitterCC);
```

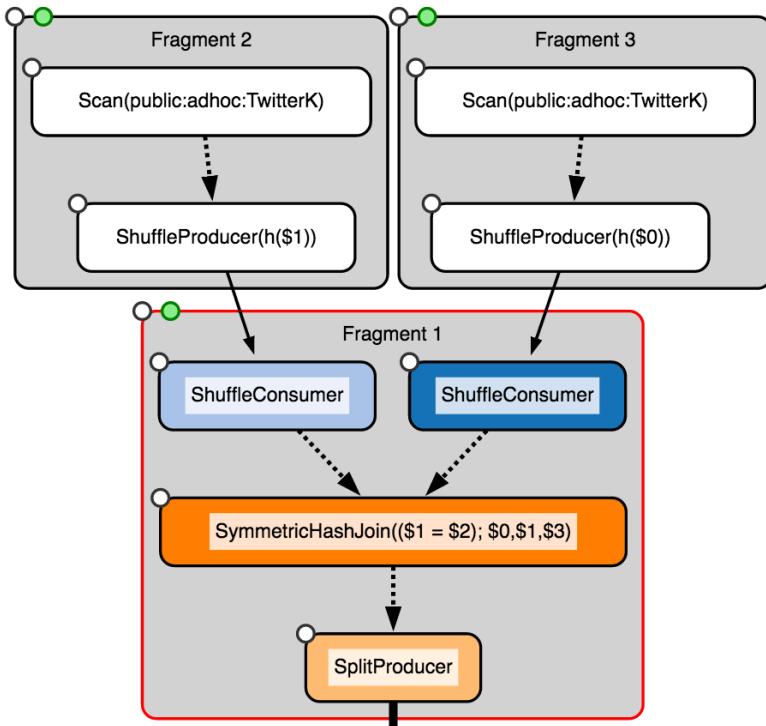
▶ Execute the Query

Parse

Myria JSON

Myria Provides Details of Query Execution

Physical Query Plan:

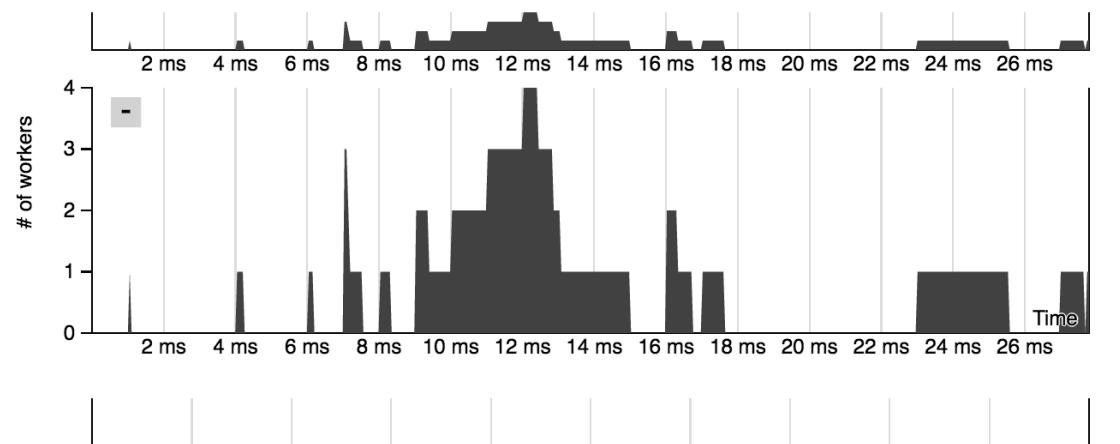


Overview / Operators inside fragment 1

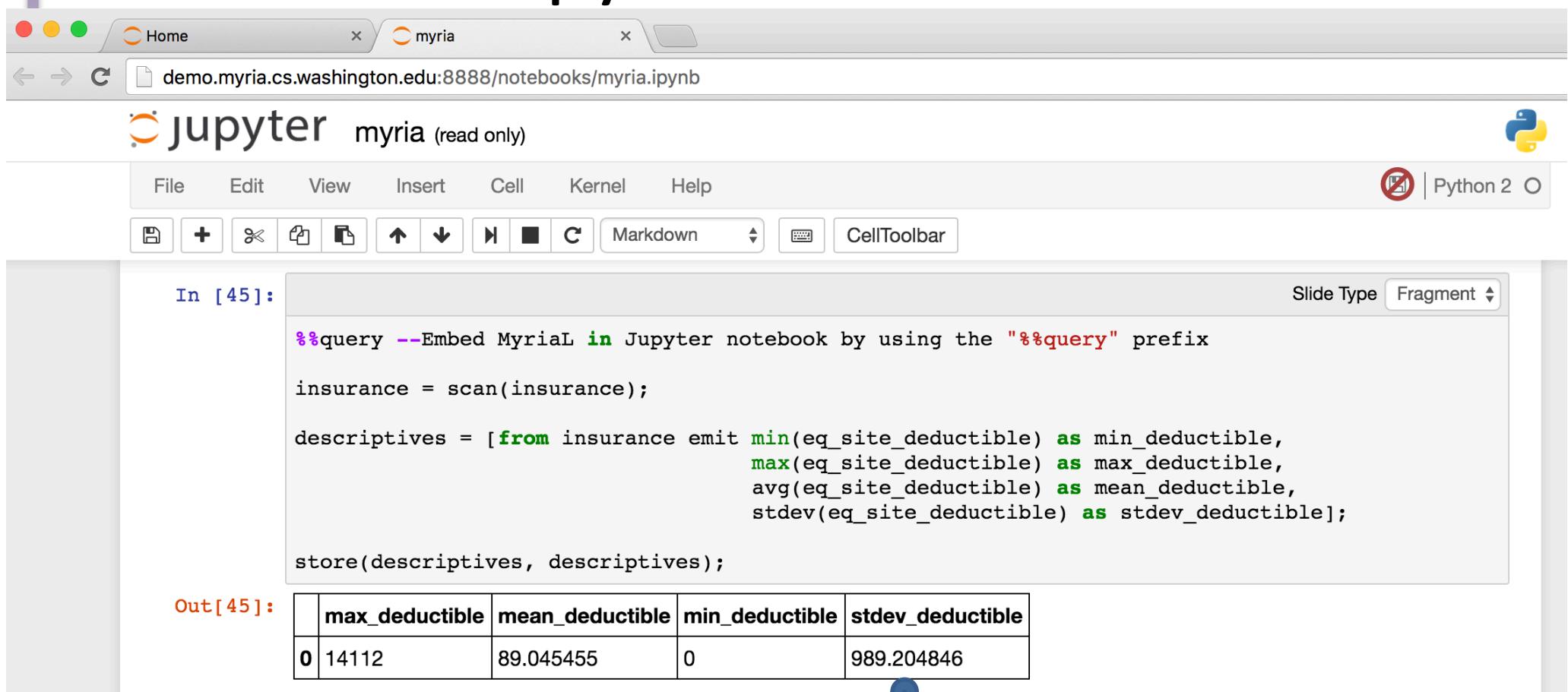
Query time contribution collapse/expand



Detailed execution



Myria Service includes Jupyter Notebook



The screenshot shows a Jupyter Notebook interface running on a Myria service. The browser tab is titled "myria" and the URL is "demo.myria.cs.washington.edu:8888/notebooks/myria.ipynb". The notebook title is "jupyter myria (read only)". The toolbar includes File, Edit, View, Insert, Cell, Kernel, Help, and a Python 2 kernel icon.

In [45]:

```
%%query --Embed MyriaL in Jupyter notebook by using the "%%query" prefix

insurance = scan(insurance);

descriptives = [from insurance emit min(eq_site_deductible) as min_deductible,
                max(eq_site_deductible) as max_deductible,
                avg(eq_site_deductible) as mean_deductible,
                stdev(eq_site_deductible) as stdev_deductible];

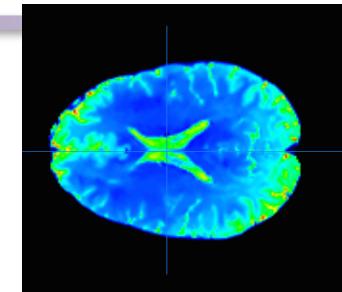
store(descriptives, descriptives);
```

Out[45]:

	max_deductible	mean_deductible	min_deductible	stdev_deductible
0	14112	89.045455	0	989.204846

Jupyter notebook available directly with Myria service

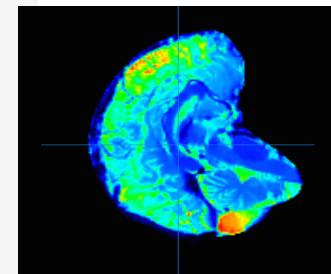
Myria Supports Python User-Defined Functions



```
#define python function
```

```
def denoise(dt):
    from dipy.denoise import nlmeans
    from dipy.denoise.noise_estimate import estimate_sigma
    image = dt[0]
    mask = dt[1]
    sigma = estimate_sigma(image)
    denoised_data = nlmeans.nlmeans(image, sigma=sigma, mask=mask)
    return denoised_data
```

MRI data analysis



Data from the Human Connectome project

```
#register python functions
```

```
connection.create_function("denoise", inspect.getsource(denoise), inSchema, outType, py, denoise)
```

```
#this query takes 2.40 mins
```

```
query = MyriaQuery.submit(
    """T1=scan(public:blob_operator:binarydata);
imgs = [from T1 emit PYUDF(denoise, T1.images, T1.mask) As denoised];
store(imgs, Denoised_imgs);"""
)
print query.status
```

Python UDFs enable running legacy code and complex analytics beyond SQL/MyriaL

Users Can Deploy Own Service

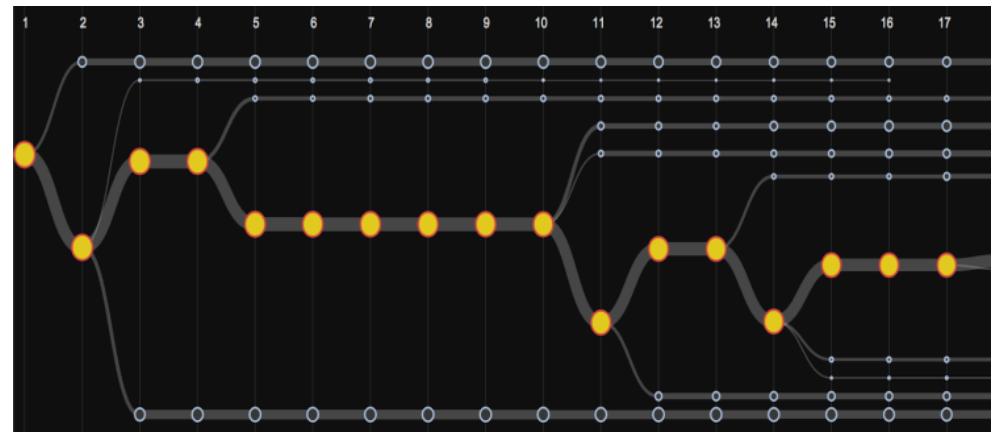
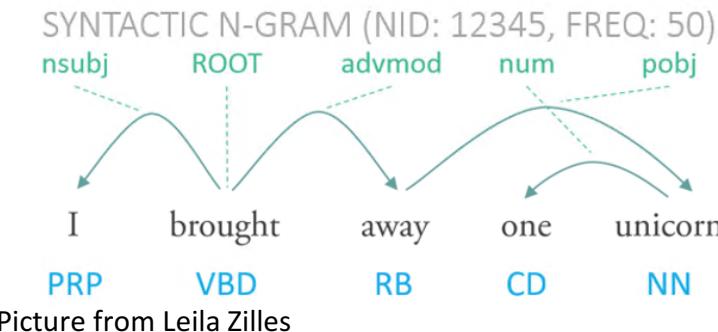
```
pip install myria-cluster
```

```
myria-cluster create [OPTIONS] CLUSTER_NAME
```

```
myria-cluster stop/start/destroy [...]
```

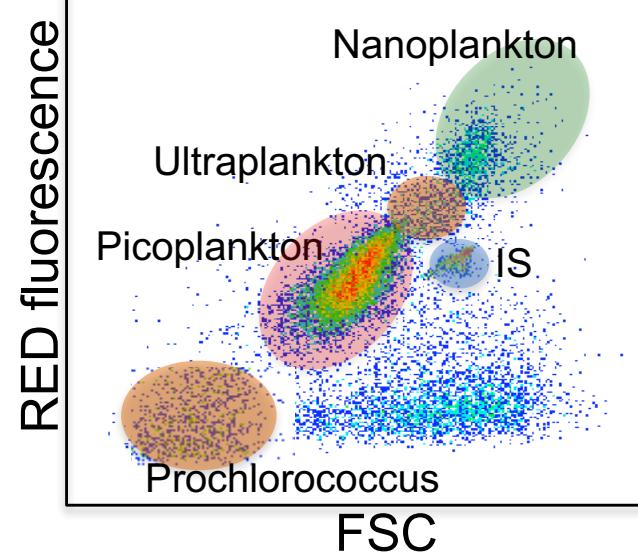
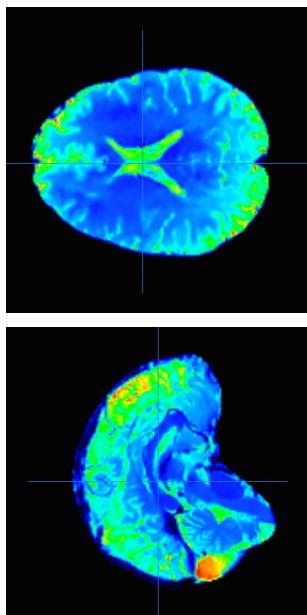
Example Myria Applications

Natural Language Processing

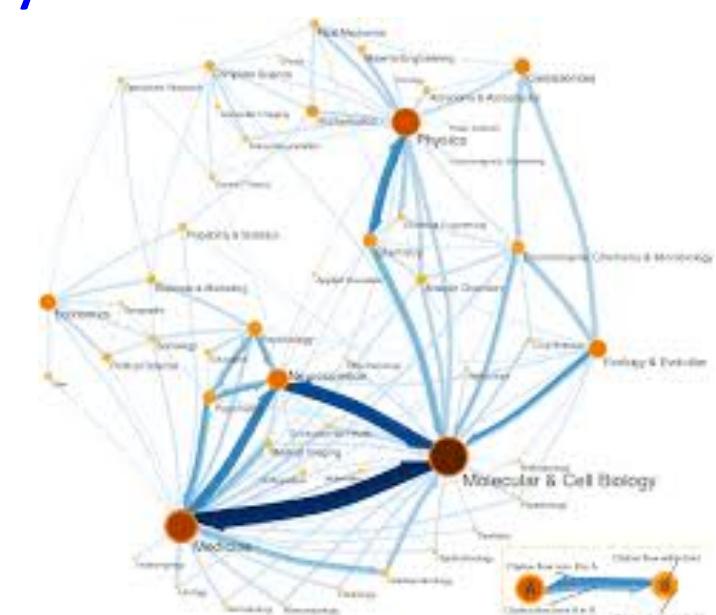


MyMergerTree Screenshot

Neuroscience



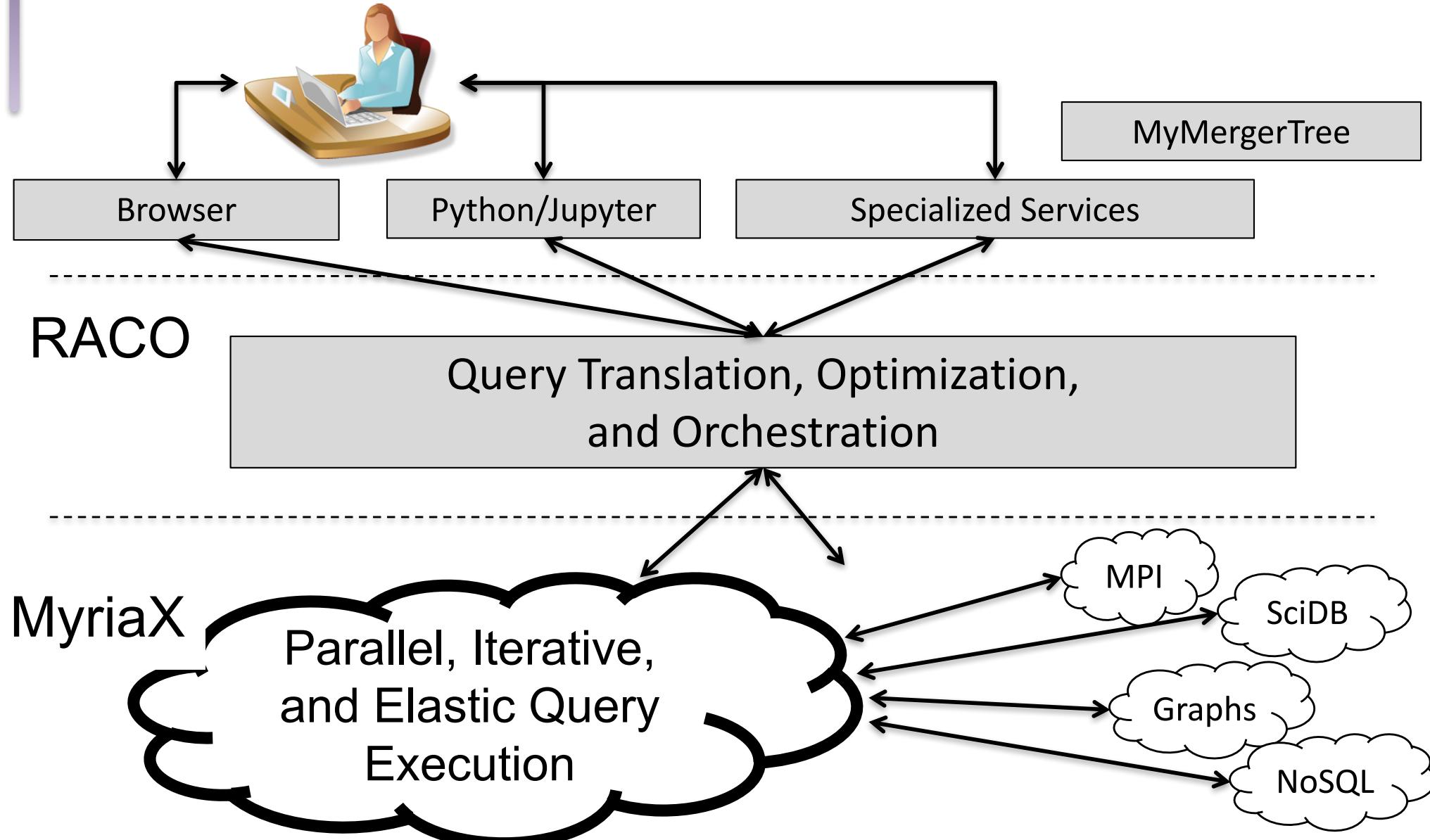
Oceanography



Bibliometrics

Myria Internals

Myria Polystore Stack



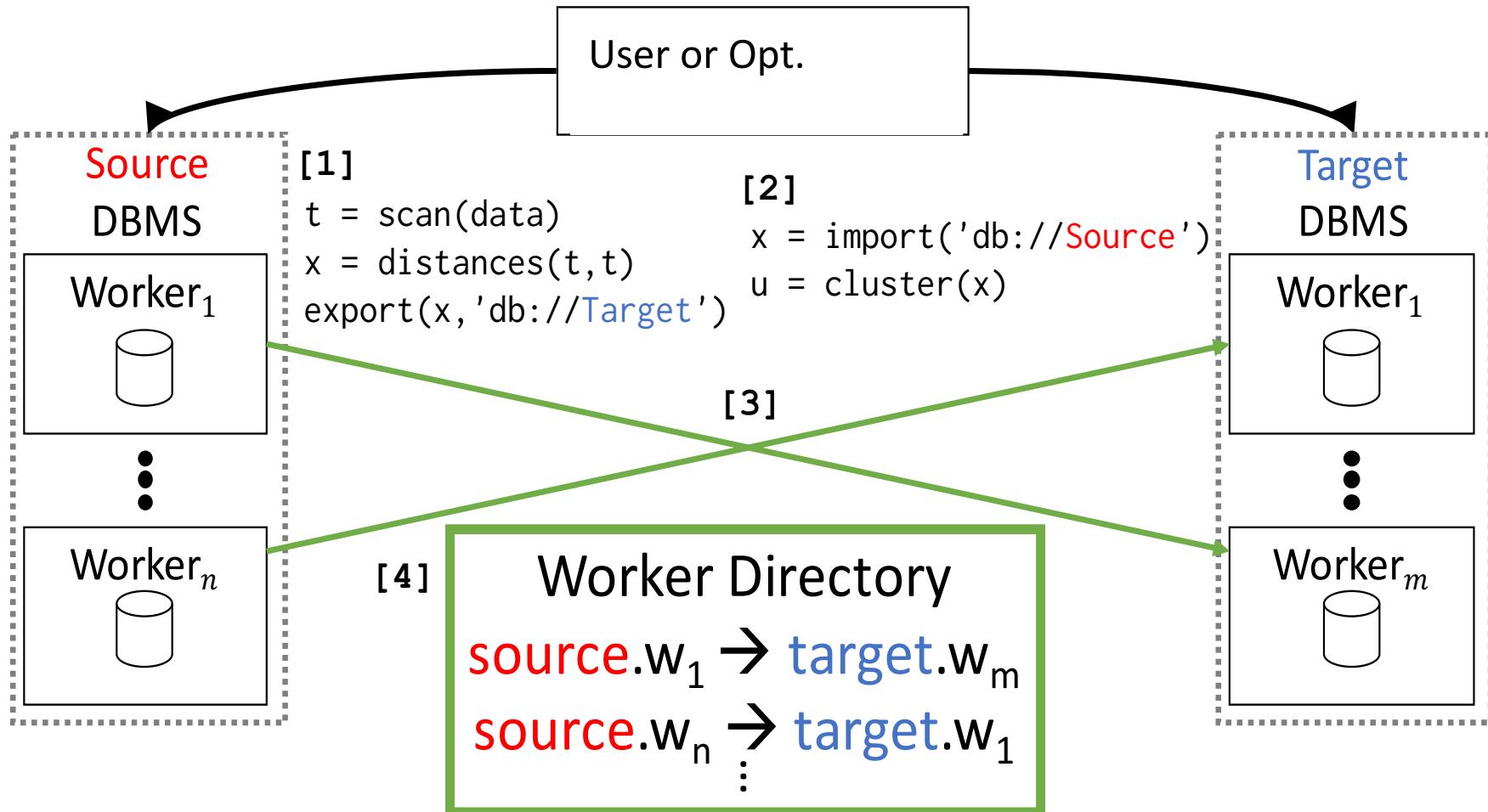
Myria's Data Model and Query Interface

- Relational Algebra Compiler (RACO)
 - Myria's query optimizer and federator
- RACO core: relational algebra extended with
 - Iterations for multi-pass algorithms
 - Flatmap to explode non-1NF attribute values into many tuples
 - Stateful apply for windowed and neighborhood functions
- Query language: MyriaL (Imperative+Declarative)
 - Each statement is declarative (SQL, comprehensions, function calls)
 - Statements are combined with imperative constructs
 - Variable assignment
 - Iteration
- Python UDFs/UDAs
 - Minimize barriers to adoption and run legacy code
- Python API
 - Fluent API with Python lambda functions

Polystore Optimization

- Rule-based opt. with three types of rules
 - Optimize logical Myria algebra plans
 - Translate logical plans into back-end specific physical plans
 - Optimize back-end specific physical plans
- To add a new back-end, developer must specify
 - Tree representation of query language
 - Rules that translate Myria algebra into this representation
 - Administrative functions including one to submit queries
- Data model independence
 - Myria hides the existence of various back-ends
 - Users write MyriaL scripts assuming relational model
 - Back-ends include select array, graph, and key-value systems

Federated Query Execution

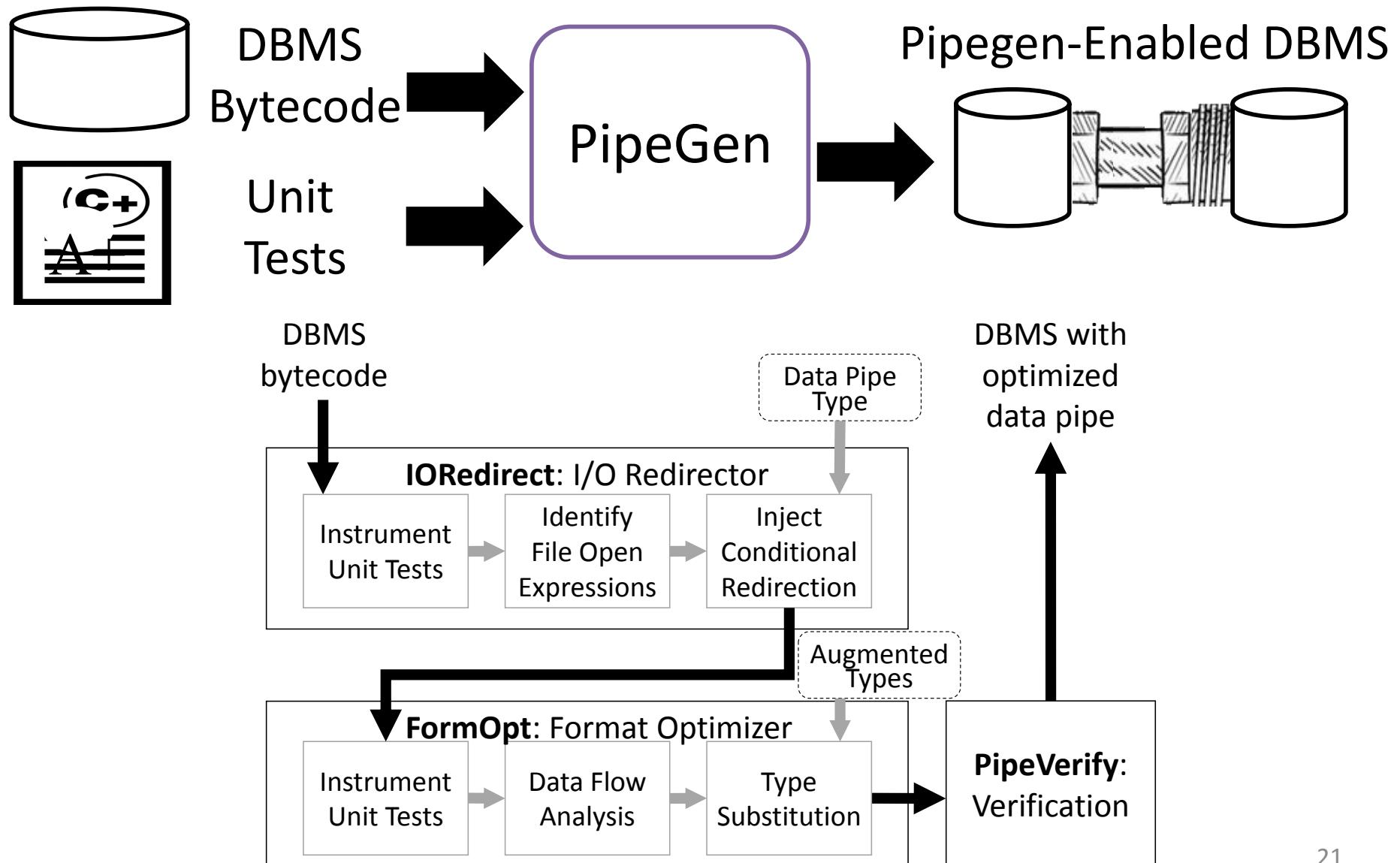


Federated plans require fast data movement

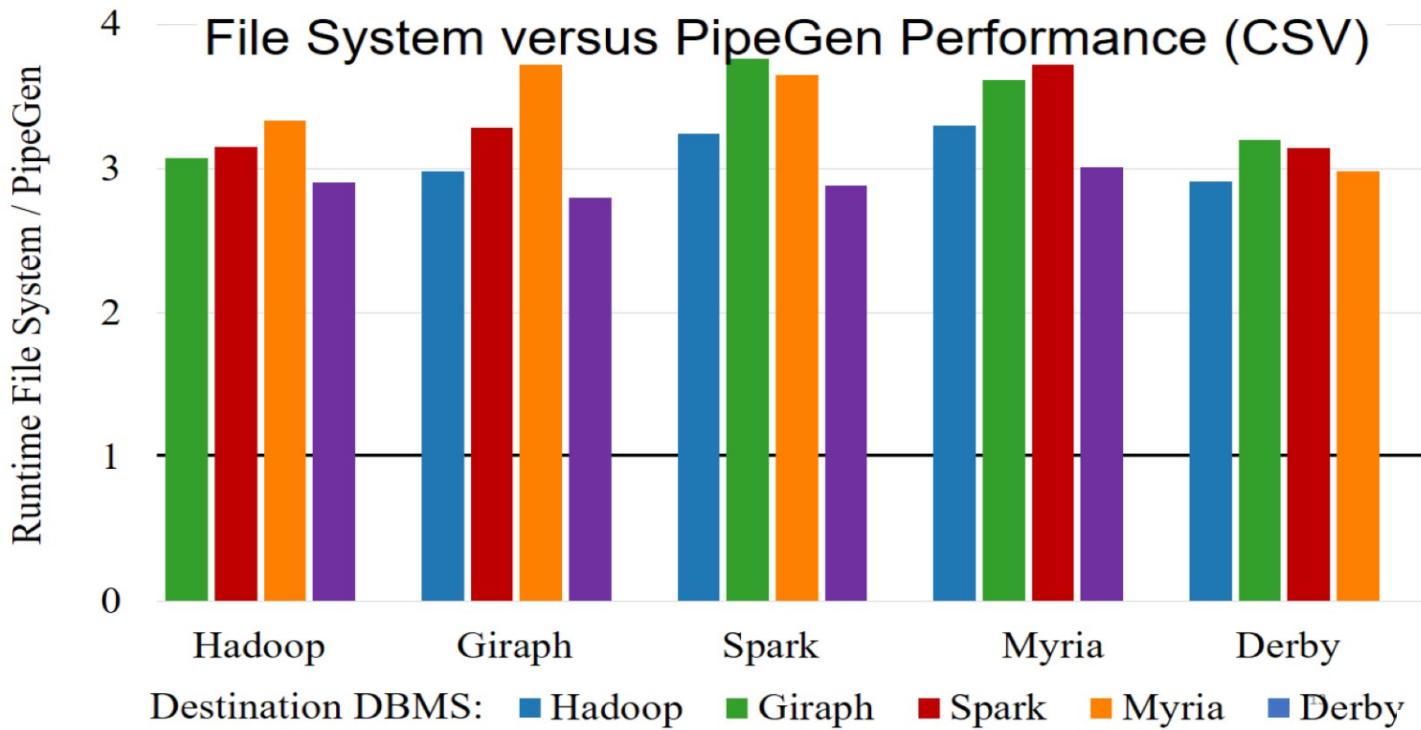
Data Movement with PipeGen

[PipeGen: Data Pipe Generator for Hybrid Analytics](#)

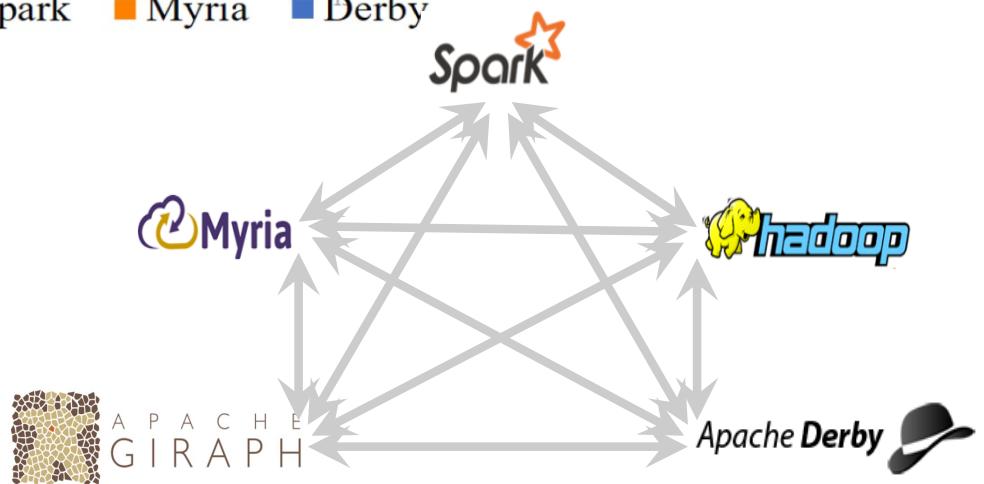
Brandon Haynes, Alvin Cheung, and Magdalena Balazinska. SOCC 2016.



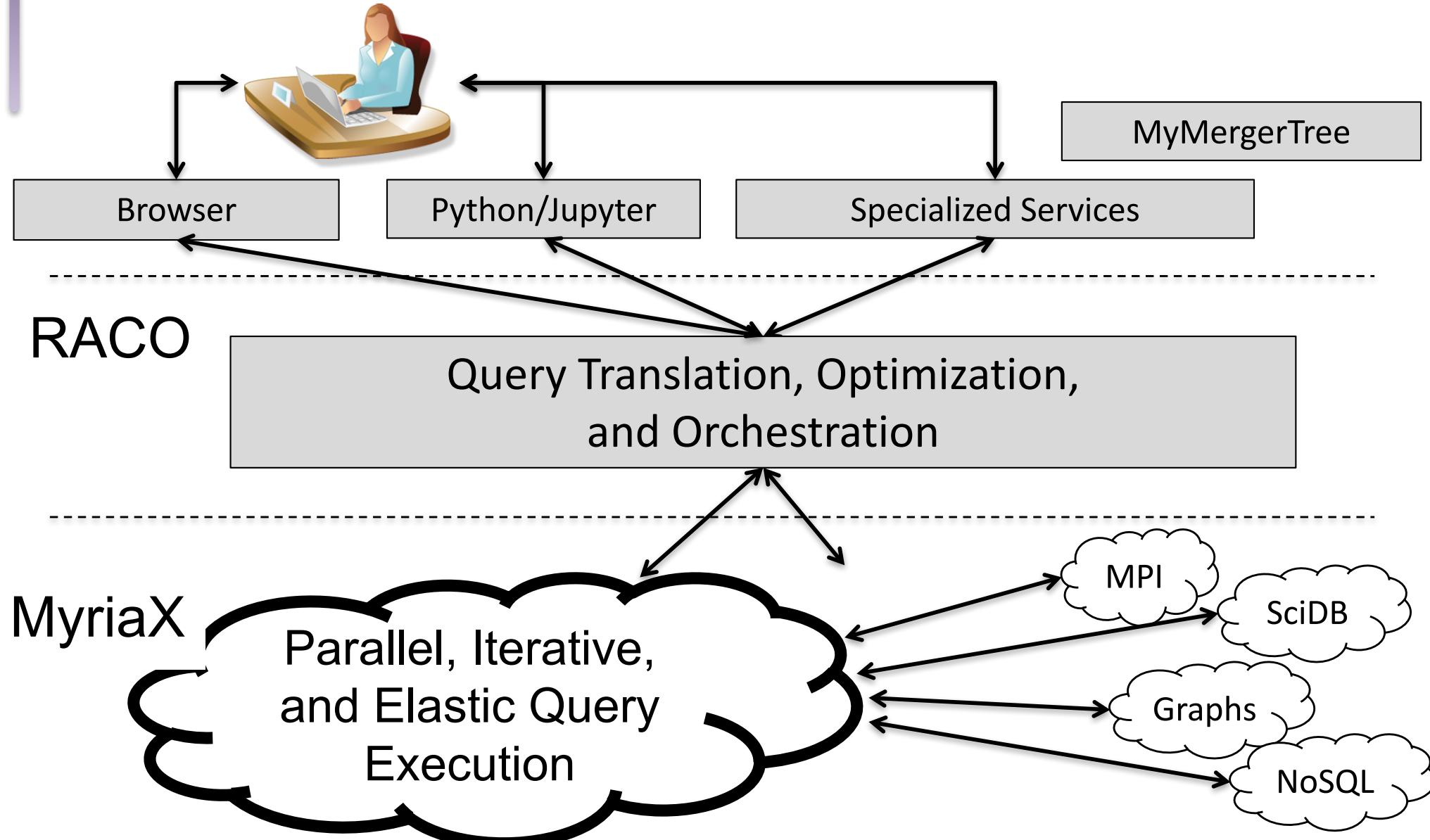
PipeGen's Performance



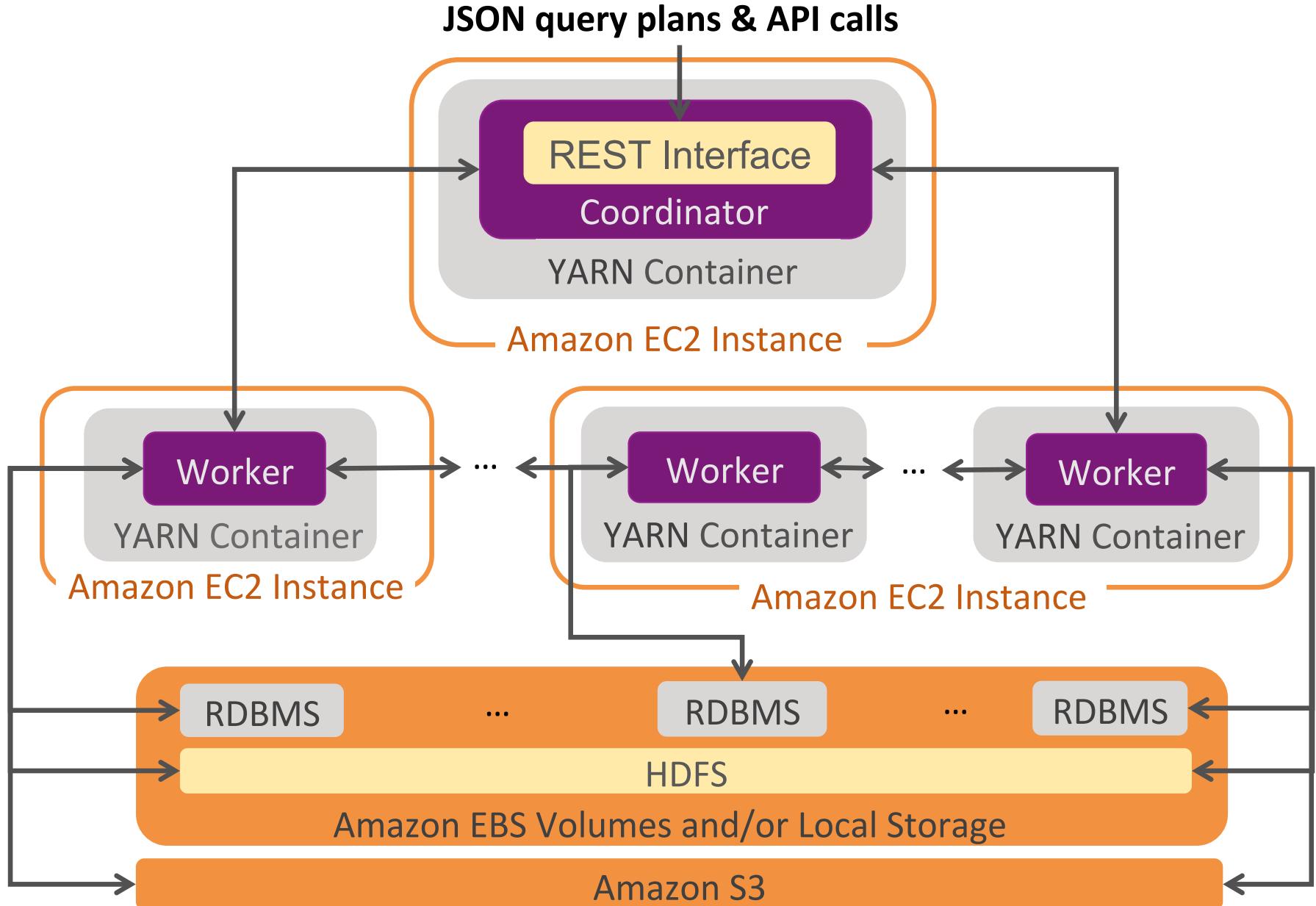
16-node cluster with 16 workers/tasks
Transfer 10^9 tuples with 4 ints and 3 doubles



Myria Polystore Stack



MyriaX Engine and Cloud Deployment



MyriaX Overview

- Data storage
 - Read data from S3, HDFS, local files
 - Parse CSV, TSV, and various scientific file formats
 - Store data in local relational DBMS instances
 - Fast storage with physical tuning (indexing, hash-partitioning)
- Query execution
 - Fundamentally a parallel DBMS
 - Fast, pipelined query execution
 - But scheduling more flexible to support elasticity
 - Novel features: Multiway joins and iterations
- Resource management
 - Executes on top of the YARN resource manager

Efficient Iterative Processing

[Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines](#)

Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. **PVLDB** 8(12): 1542-1553 (2015)

- User specifies query declaratively
 - Subset of Datalog with aggregation
- Generate efficient, shared-nothing query plan
 - Small extensions to existing shared-nothing systems
- Plan amenable to runtime optimizations
 - Synchronous vs asynchronous
 - Different processing priorities
- Optimizations significantly affect performance

Myria's Optimized Iterations Example

[Asynchronous and Fault-Tolerant Recursive Datalog Evaluation in Shared-Nothing Engines](#)

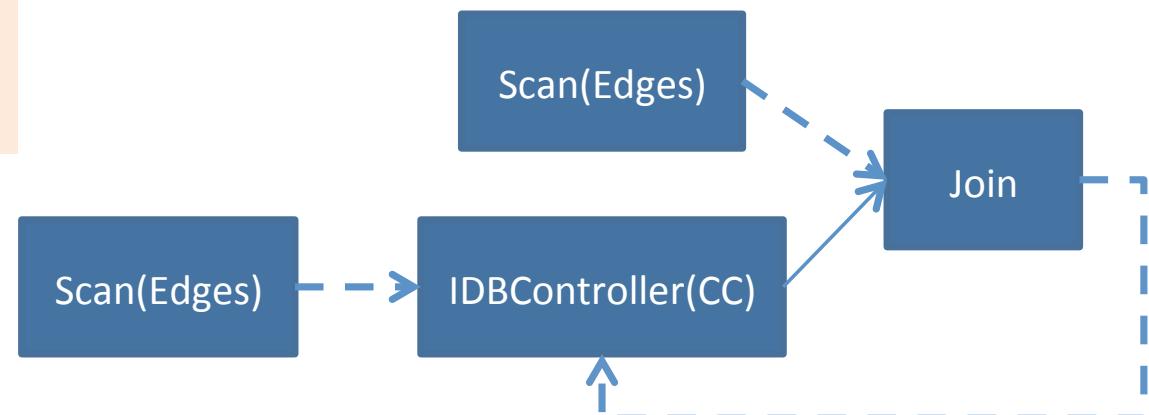
Jingjing Wang, Magdalena Balazinska, and Daniel Halperin. **PVLDB** 8(12): 1542-1553 (2015)

Declarative Query

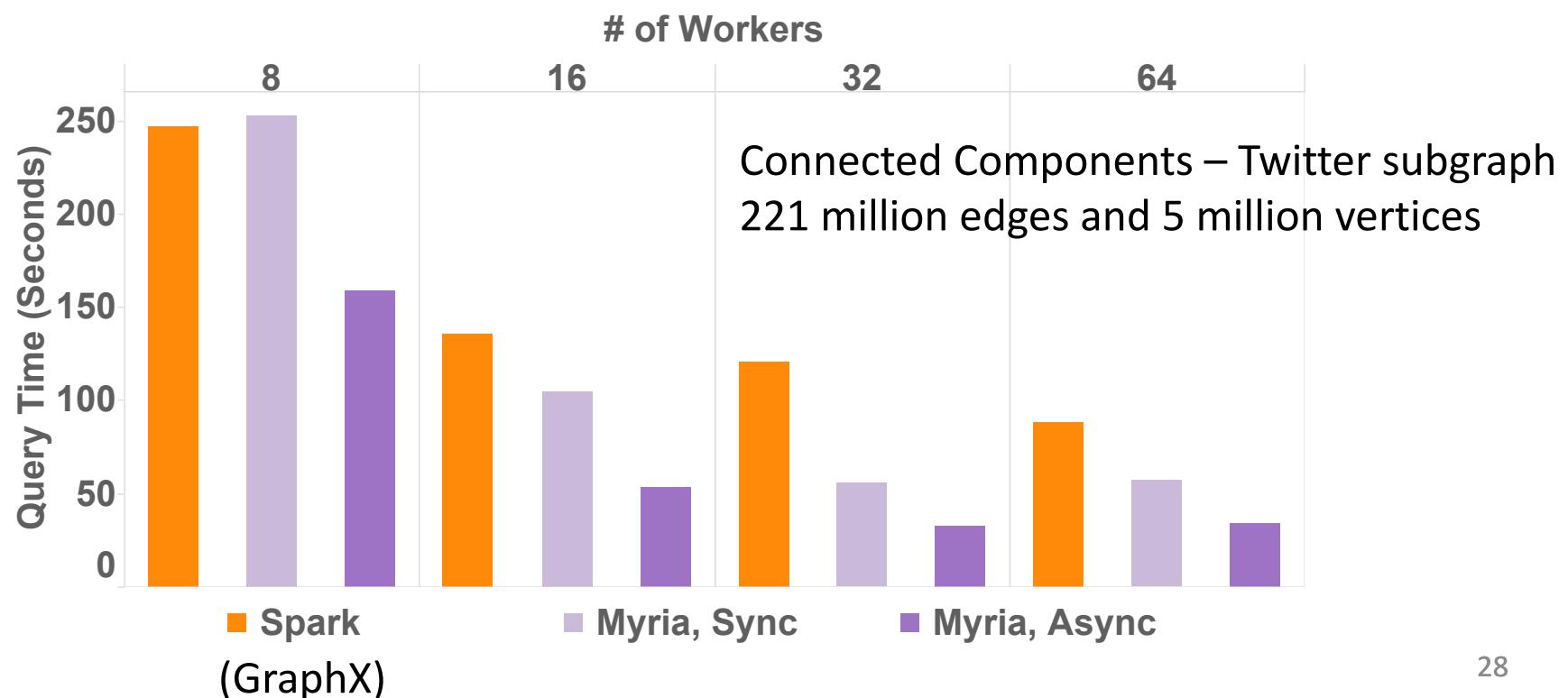
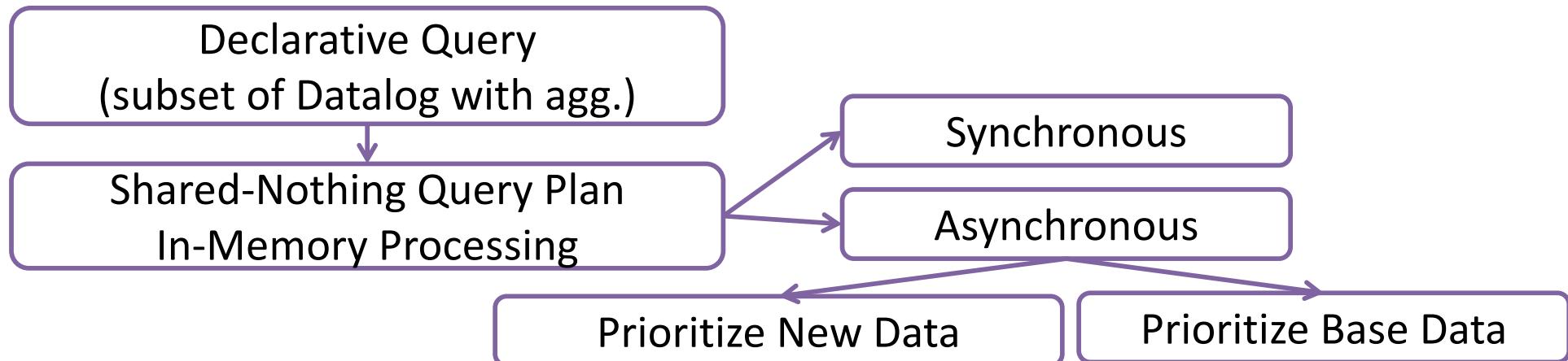
```
E = scan(jwang:cc:graph);  
V = select distinct E.$0 from E;  
do  
    CC := [ $0, MIN($1) ] <-  
        [from V emit V.$0 as x, V.$0 as y] +  
        [from E, CC where E.$0 = CC.$0 emit E.$1, CC.$1];  
until convergence;  
store(CC, CC);
```

// Can have multiple relations
// with recursive dep.

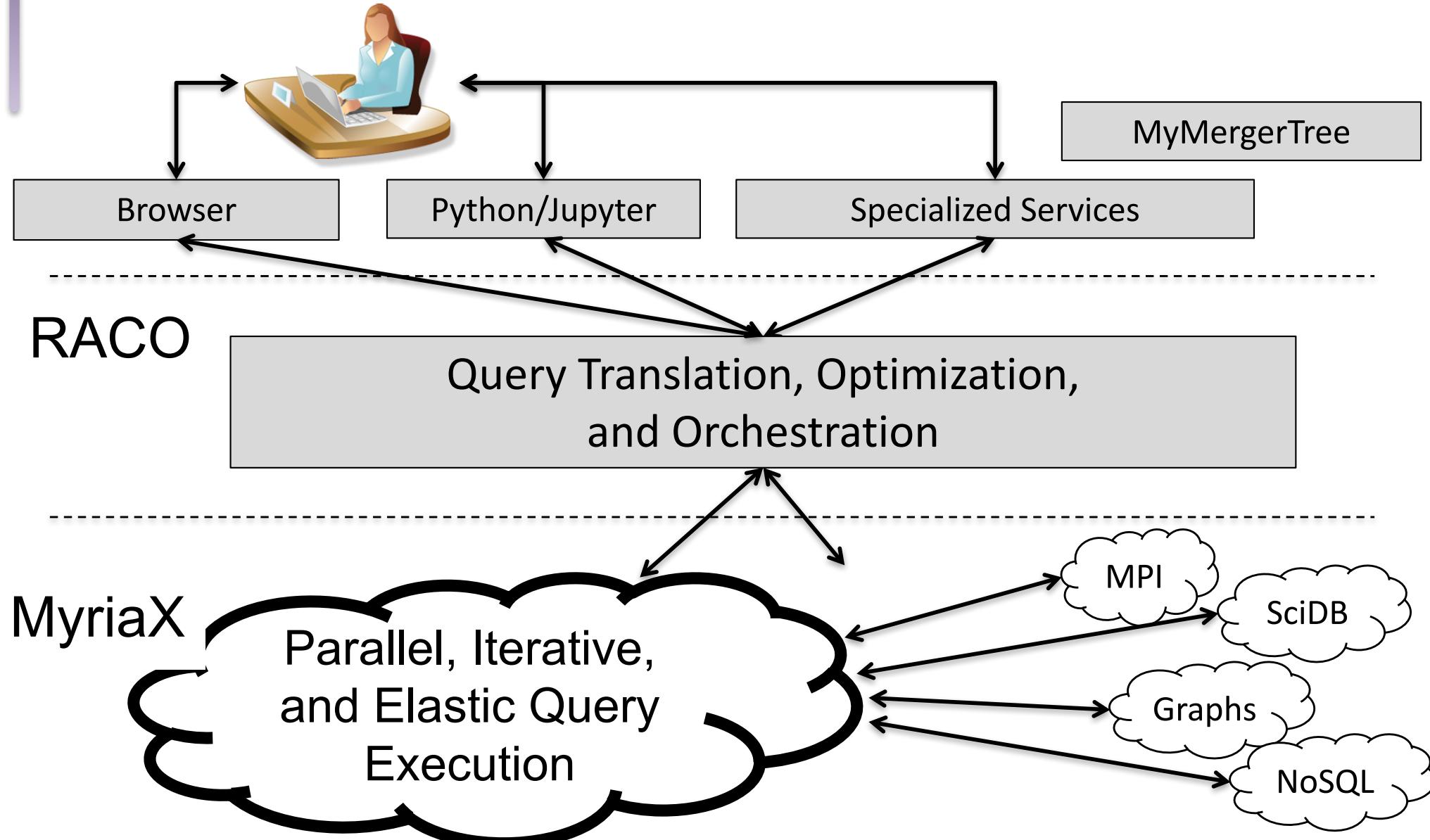
Compiled to a Distributed Query Plan



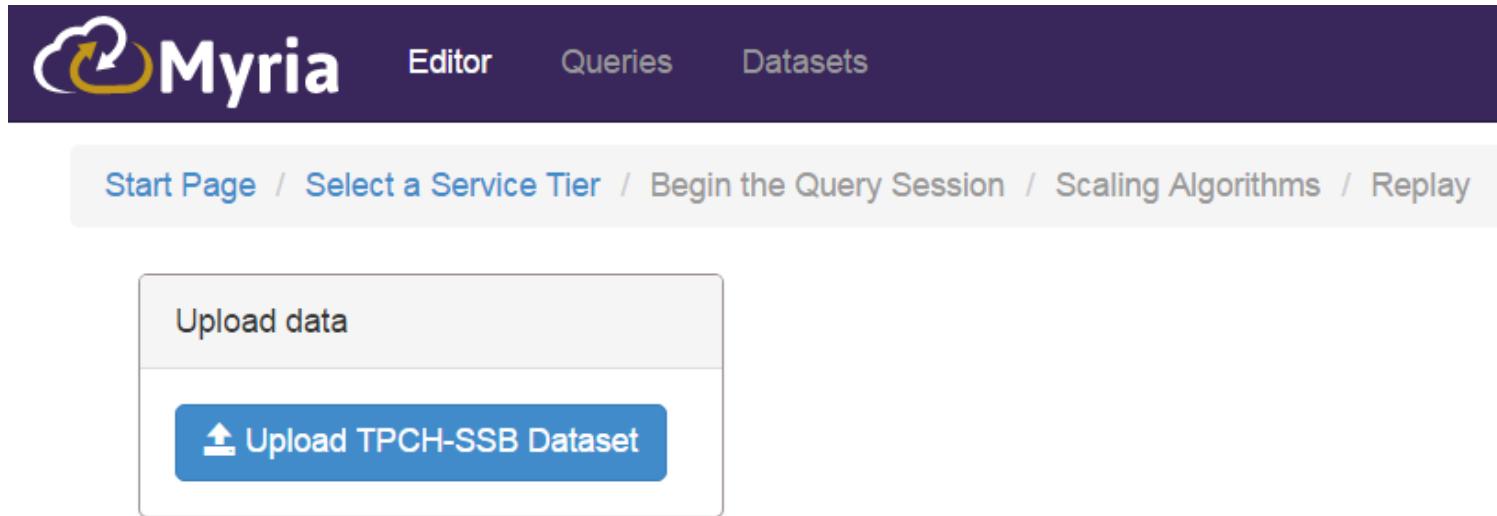
Performance Comparison with Spark



Myria Polystore Stack



Cloud Operation in Myria



The screenshot shows the Myria web interface. At the top, there is a dark blue header bar with the Myria logo on the left and three navigation items: "Editor", "Queries", and "Datasets". Below the header, a light gray navigation bar contains links: "Start Page", "Select a Service Tier", "Begin the Query Session", "Scaling Algorithms", and "Replay". The main content area features a white rectangular box with rounded corners. Inside this box, the text "Upload data" is centered above a blue button. The button has a white arrow icon pointing upwards and the text "Upload TPCH-SSB Dataset".

Or point to data in Amazon S3

Myria's Personalized Service Level Agreements

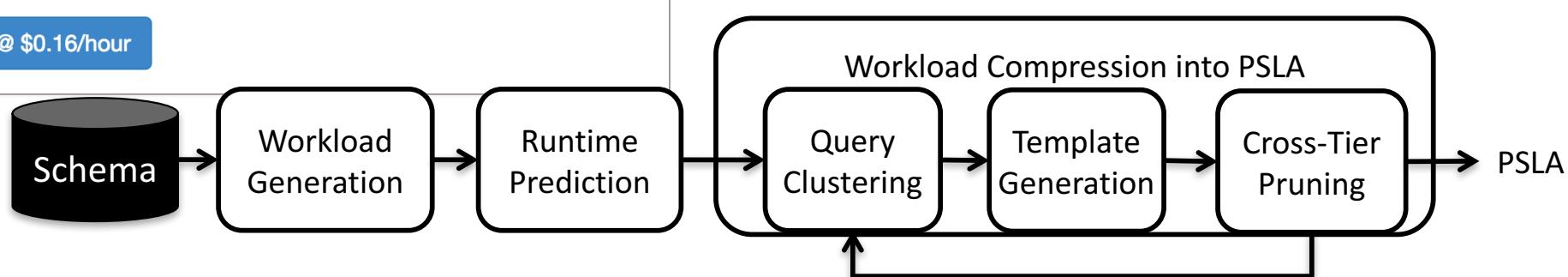
[Changing the Face of Database Cloud Services with Personalized Service Level Agreements](#)

Jennifer Ortiz, Victor T. Almeida, and Magdalena Balazinska. **CIDR 2015**

Tier #1	
Query Template	Runtime (seconds)
SELECT (17 attributes) FROM (lineitem) WHERE (1% of data selected)	10
SELECT (9 attributes) FROM (part) WHERE (100% of data selected)	
SELECT (9 attributes) FROM (customer) WHERE (100% of data selected)	
SELECT (17 attributes) FROM (date) WHERE (100% of data selected)	
SELECT (35 attributes) FROM (3 TABLES) WHERE (100% of data selected)	60
SELECT (48 attributes) FROM (5 TABLES) WHERE (10% of data selected)	
SELECT (60 attributes) FROM (5 TABLES) WHERE (1% of data selected)	
SELECT (49 attributes) FROM (5 TABLES) WHERE (100% of data selected)	300
SELECT (60 attributes) FROM (5 TABLES) WHERE (10% of data selected)	
SELECT (60 attributes) FROM (5 TABLES) WHERE (100% of data selected)	600

Tier #2	
Query Template	Runtime (seconds)
SELECT (26 attributes) FROM (2 TABLES) WHERE (100% of data selected)	10
SELECT (60 attributes) FROM (5 TABLES) WHERE (10% of data selected)	
SELECT (60 attributes) FROM (5 TABLES) WHERE (100% of data selected)	300
 Purchase @ \$0.64/hour	

Myria's SLA generation



Myria's PerfEnforce Subsystem

[PerfEnforce Demonstration: Data Analytics with Performance Guarantees](#)

Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. **SIGMOD 2016**.

Query Session

Write a Query...

```
1 SELECT *
2 FROM "public:adhoc:lineitem" AS L
3 WHERE l_linenumber = 7;
```

See SLA Runtime

Execute

Query Information

Expected Runtime (from SLA): 11.756 seconds

status: SUCCESS

seconds elapsed: 1.168460994

Cluster is using 12 workers

Previous Queries Log

Query: SELECT * FROM "public:adhoc:lineitem" AS L WHERE l_linenumber = 7;

Actual Runtime: 1.168460994

Expected Runtime: 11.756

Cluster Size Ran: 12

Myria's PerfEnforce Subsystem

[PerfEnforce Demonstration: Data Analytics with Performance Guarantees](#)

Jennifer Ortiz, Brendan Lee, and Magdalena Balazinska. **SIGMOD 2016**.

Query Session

How can the cloud provider guarantee these runtimes?

Write a Query...

```
1 SELECT *\n2 FROM "public:adhoc:lineitem" AS L\n3 WHERE l_linenumber = 7;
```

See SLA Runtime

Execute

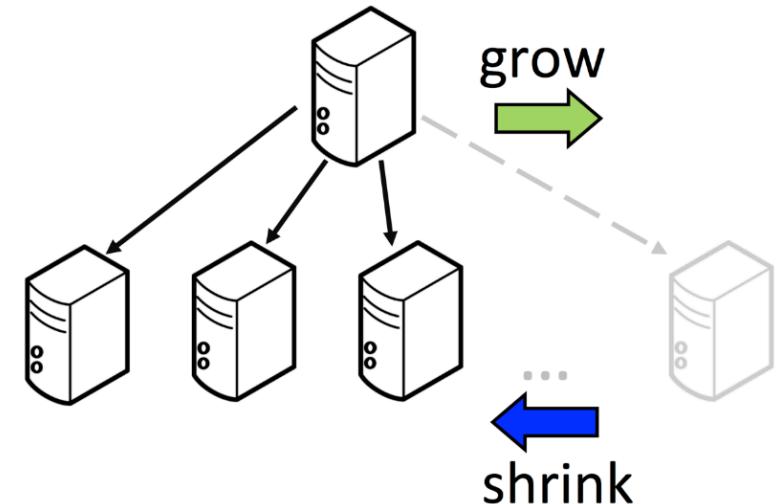
Query Information

Expected Runtime (from SLA): 11.756 seconds

status: RUNNING

seconds elapsed: 1.048921161

Cluster is using 6 workers



Previous Queries Log

Query: SELECT * FROM "public:adhoc:lineitem" AS L WHERE l_linenumber = 7;

Actual Runtime: 1.168460994

Expected Runtime: 11.756

Cluster Size Ran: 12

Cluster size changes
during query session

Myria's Innovations Summary

Efficient Processing &
Complex Analytics with MyriaX

Efficient Multi-Join

Iterative Queries

Data Summaries

Image Processing

Myria Polystore

Federated Analytics

Automatic Data Pipes

Perf. Debugging

Myria Cloud Operation

Cloud PSLAs

Performance Guarantees

Elastic Memory

Conclusion

- **Highly expressive**
 - MyriaL (RA+iterations) & Python
- **Polystore with hybrid analytics**
- **High performance on variety of queries**
- **Available as a service**
 - Focus on low barrier to entry
 - And turning users into self-sufficient experts
 - Also focus on the service provider: Operate Myria
- **Source code and more info (includes videos)**

<http://myria.cs.washington.edu/>