



Apache Tez : Next Generation Execution Engine upon Hadoop

Jeff Zhang

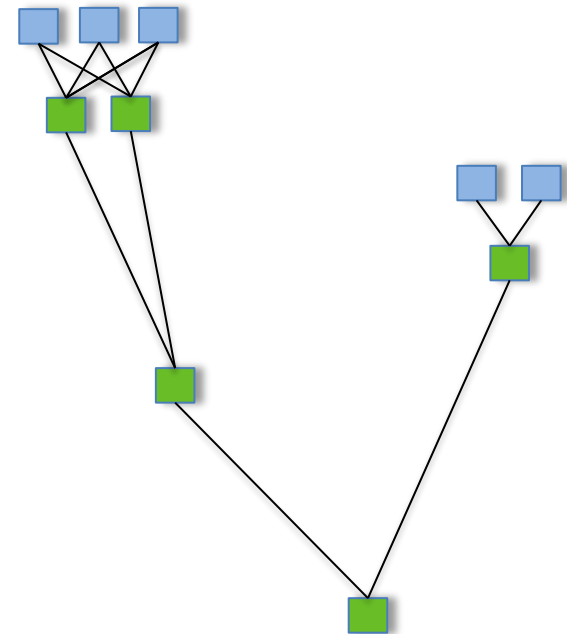


Outline

- **Tez Introduction**
- **Tez API**
- **Tez Internal**
- **Tez Project Status**
- **Q & A**

Tez – Introduction

- **Distributed execution framework targeted towards data-processing applications.**
- **Based on expressing a computation as a dataflow graph.**
- **Highly customizable to meet a broad spectrum of use cases.**
- **Built on top of YARN – the resource management framework for Hadoop.**
- **Open source Apache project and Apache licensed.**



Hadoop 1 -> Hadoop 2

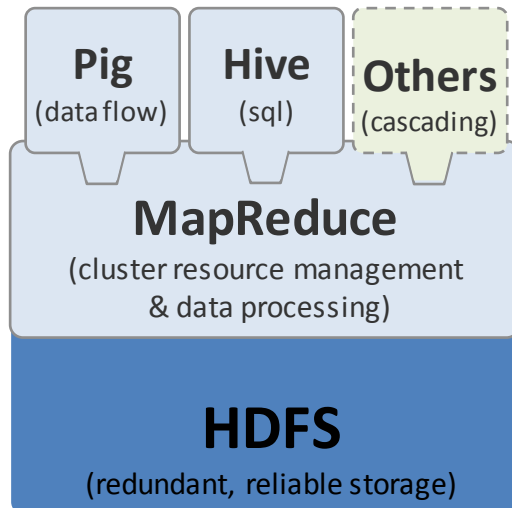
Monolithic

- Resource Management
- Execution Engine
- User API

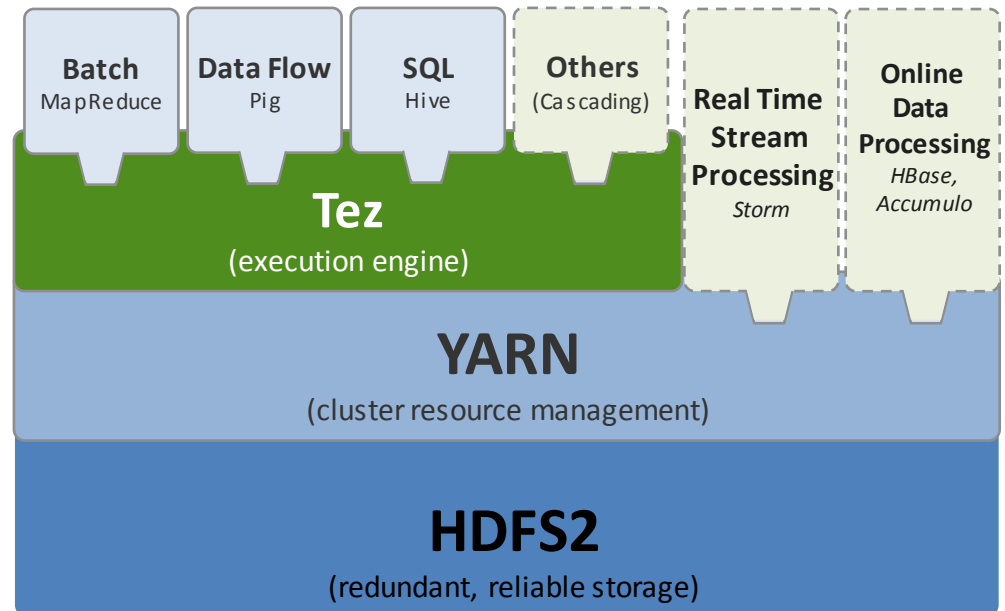
Layered

- Resource Management – YARN
- Execution Engine – Tez
- User API – Hive, Pig, Cascading, Your App!

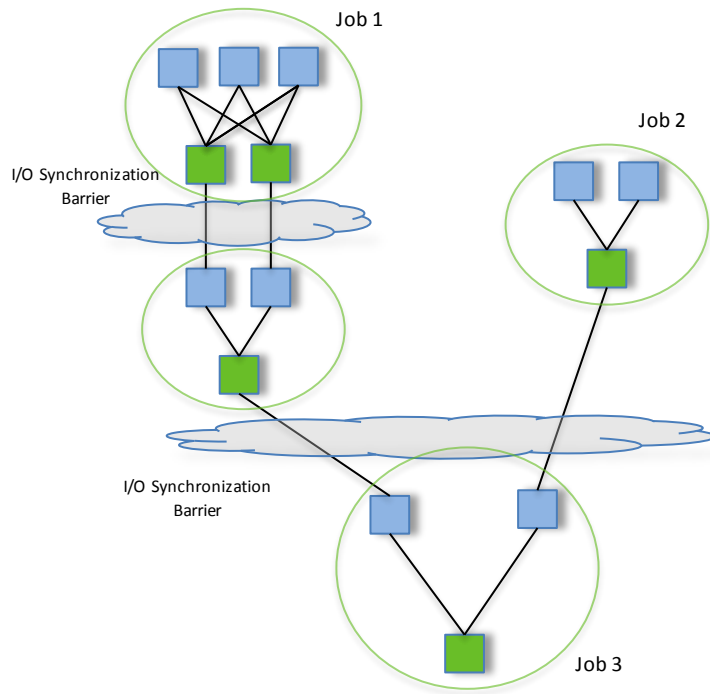
HADOOP 1.0



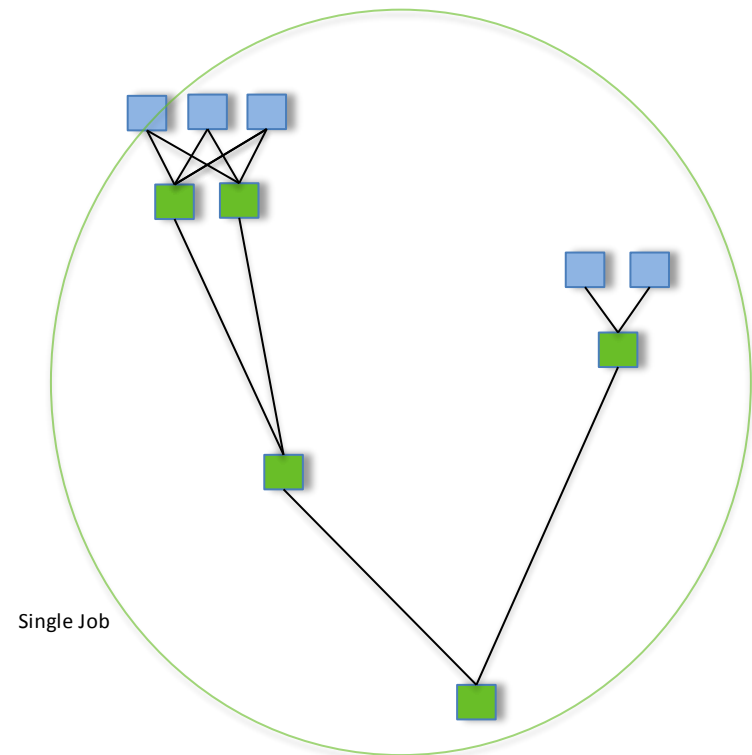
HADOOP 2.0



Pig/Hive-MR versus Pig/Hive-Tez



Pig/Hive - MR



Pig/Hive - Tez

Outline

- Tez Introduction
- Tez API
- Tez Internal
- Tez Project Info
- Q & A

Tez – Expressing the computation

Tez provides the following APIs to define the processing

- **DAG API (Vertex, Edge)**

- Defines the structure of the data processing and the relationship between producers and consumers
- Enable definition of complex data flow pipelines using simple graph connection API's. Tez expands the logical DAG at runtime
- This is how all the tasks in the job get specified

- **Runtime API (Task)**

- Defines the interfaces using which the framework and app code interact with each other
- App code transforms data and moves it between tasks
- This is how we specify what actually executes in each task on the cluster nodes

Tez – DAG API

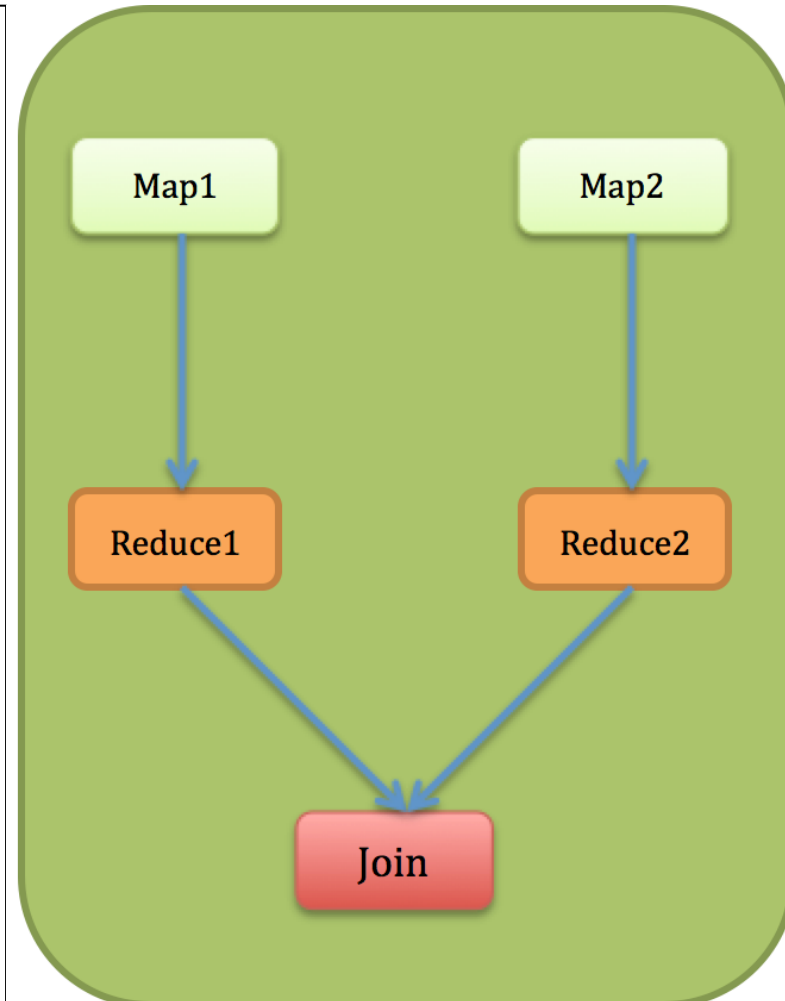
Simple DAG definition API

```
// Define DAG
DAG dag = new DAG();

// Define Vertex
Vertex map1 = new Vertex(MapProcessor.class);

// Define Edge
Edge edge1 = Edge(map1, reduce1, SCATTER_GATHER,
PERSISTED, SEQUENTIAL, Output.class, Input.class);

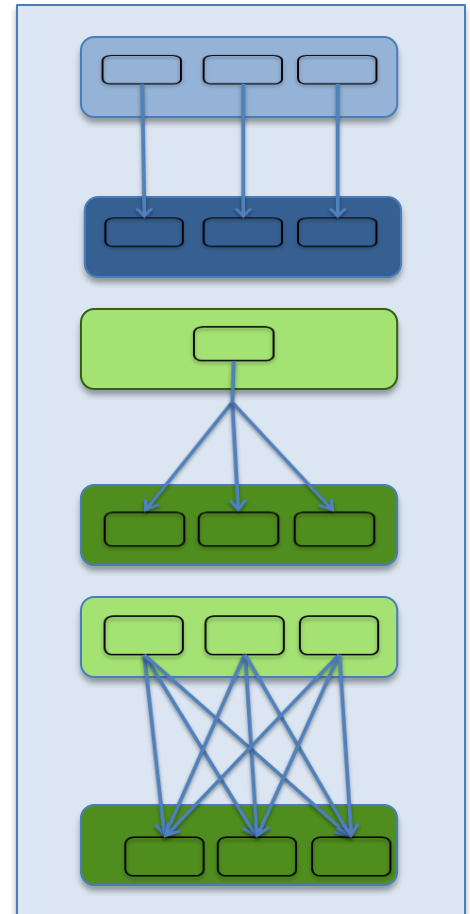
// Connect them
dag.addVertex(map1)
    .addEdge(edge)
    ...
```



Tez – DAG API

Edge properties define the connection between producer and consumer tasks in the DAG

- **Data movement – Defines routing of data between tasks**
 - **One-To-One** : Data from the i^{th} producer task routes to the i^{th} consumer task.
 - **Broadcast** : Data from a producer task routes to all consumer tasks.
 - **Scatter-Gather** : Producer tasks scatter data into shards and consumer tasks gather the data. The i^{th} shard from all producer tasks routes to the i^{th} consumer task.
- **Scheduling – Defines when a consumer task is scheduled**
 - **Sequential** : Consumer task may be scheduled after a producer task completes.
 - **Concurrent** : Consumer task must be co-scheduled with a producer task.
- **Data source – Defines the lifetime/reliability of a task output**
 - **Persisted** : Output will be available after the task exits. Output may be lost later on.
 - **Persisted-Reliable** : Output is reliably stored and will always be available
 - **Ephemeral** : Output is available only while the producer task is running



Tez – Runtime API (IPO)

Flexible Inputs-Processor-Outputs Model

- Thin API layer to wrap around arbitrary application code
- Compose inputs, processor and outputs to execute arbitrary processing
- Event routing based control plane architecture
- Applications decide logical data format and data transfer technology
- Customize for performance
- Built-in implementations for Hadoop 2.0 data services – HDFS and YARN ShuffleService. Built on the same API. Your impls are as first class as ours!

Input

```
List<Event> initialize()  
handleEvents(List<Event> events)  
start()  
Reader getReader()  
close()
```

Processor

```
void initialize()  
run(Map<String, LogicalInput> inputs,  
    Map<String, LogicalOutput> outputs)  
handleEvents(List<Event> events)  
close()
```

Output

```
List<Event> initialize()  
handleEvents(List<Event> events)  
start()  
Writer getWriter()  
close()
```

Tez – Runtime API (VertexManager)

- **VertexManager**

- Control on the flow execution engine in vertex level

```
void initialize();
```

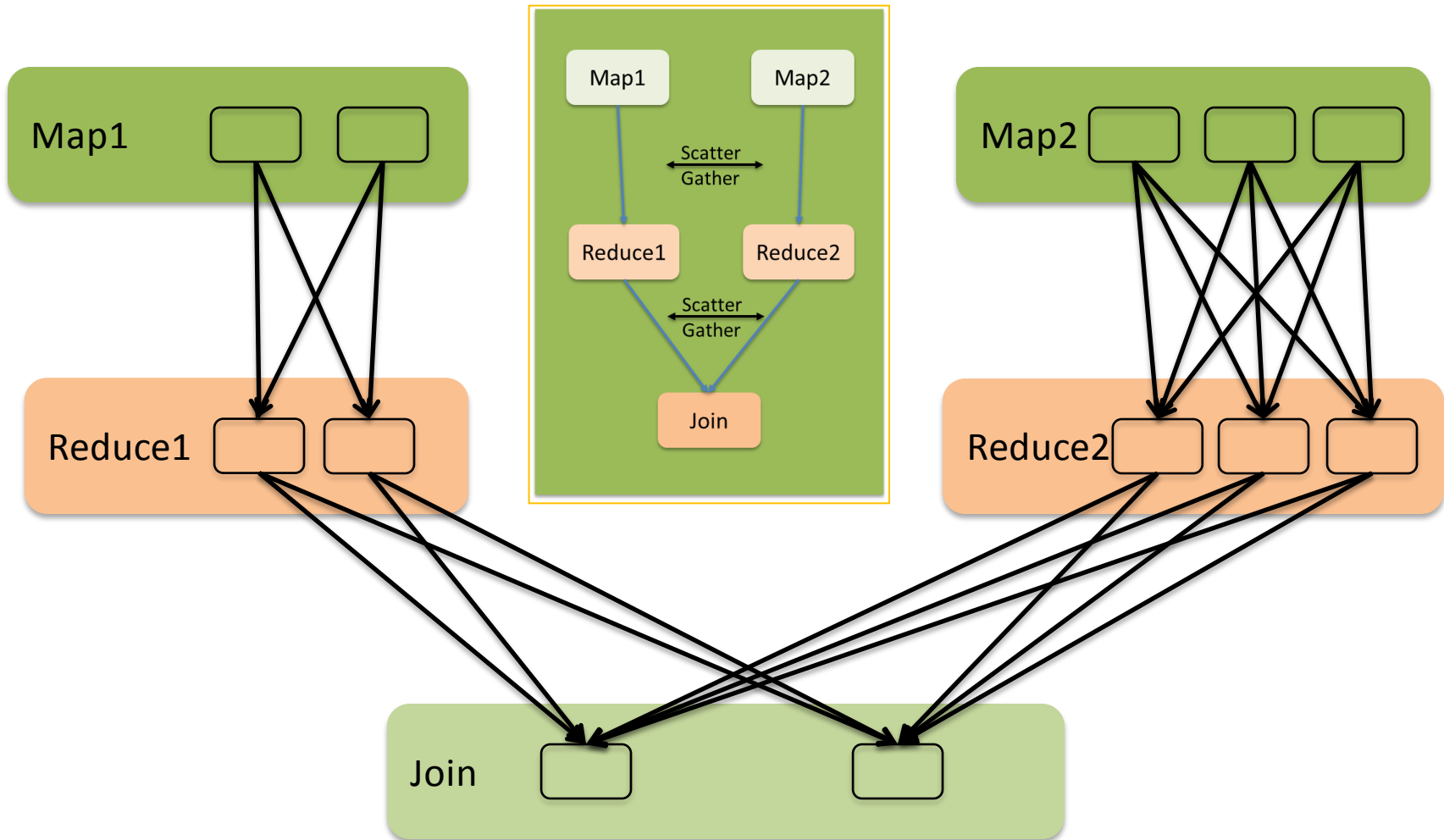
```
void onVertexStarted(Map<String, List<Integer>> completions);
```

```
void onSourceTaskCompleted(String srcVertexName, Integer taskId);
```

```
void onVertexManagerEventReceived(VertexManagerEvent vmEvent);
```

```
void onRootVertexInitialized(String inputName,  
    InputDescriptor inputDescriptor, List<Event> events);
```

Tez – Logical DAG expansion at Runtime



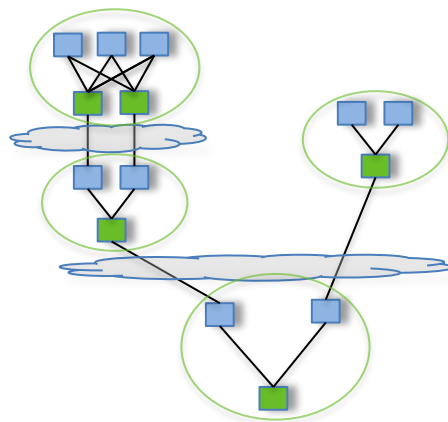
Tez – Performance

- **Benefits of expressing the data processing as a DAG**
 - Reducing overheads and queuing effects
 - Gives system the global picture for better planning
- **Efficient use of resources**
 - Re-use resources to maximize utilization
 - Pre-launch, pre-warm and cache
 - Locality & resource aware scheduling
- **Support for application defined DAG modifications at runtime for optimized execution**
 - Change task concurrency
 - Change task scheduling
 - Change DAG edges
 - Change DAG vertices

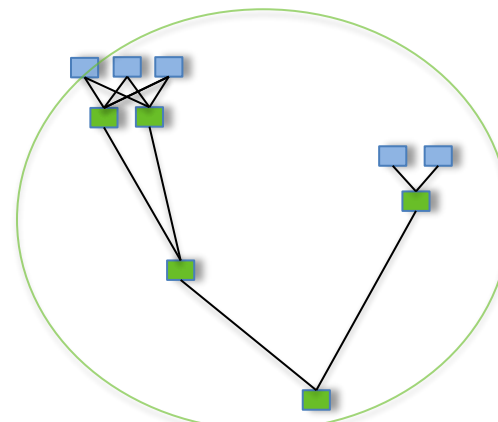
Tez – Benefits of DAG execution

Faster Execution and Higher Predictability

- Eliminate replicated write barrier between successive computations.
- Eliminate job launch overhead of workflow jobs.
- Eliminate extra stage of map reads in every workflow job.
- Eliminate queue and resource contention suffered by workflow jobs that are started after a predecessor job completes.
- Better locality because the engine has the global picture



Pig/Hive - MR

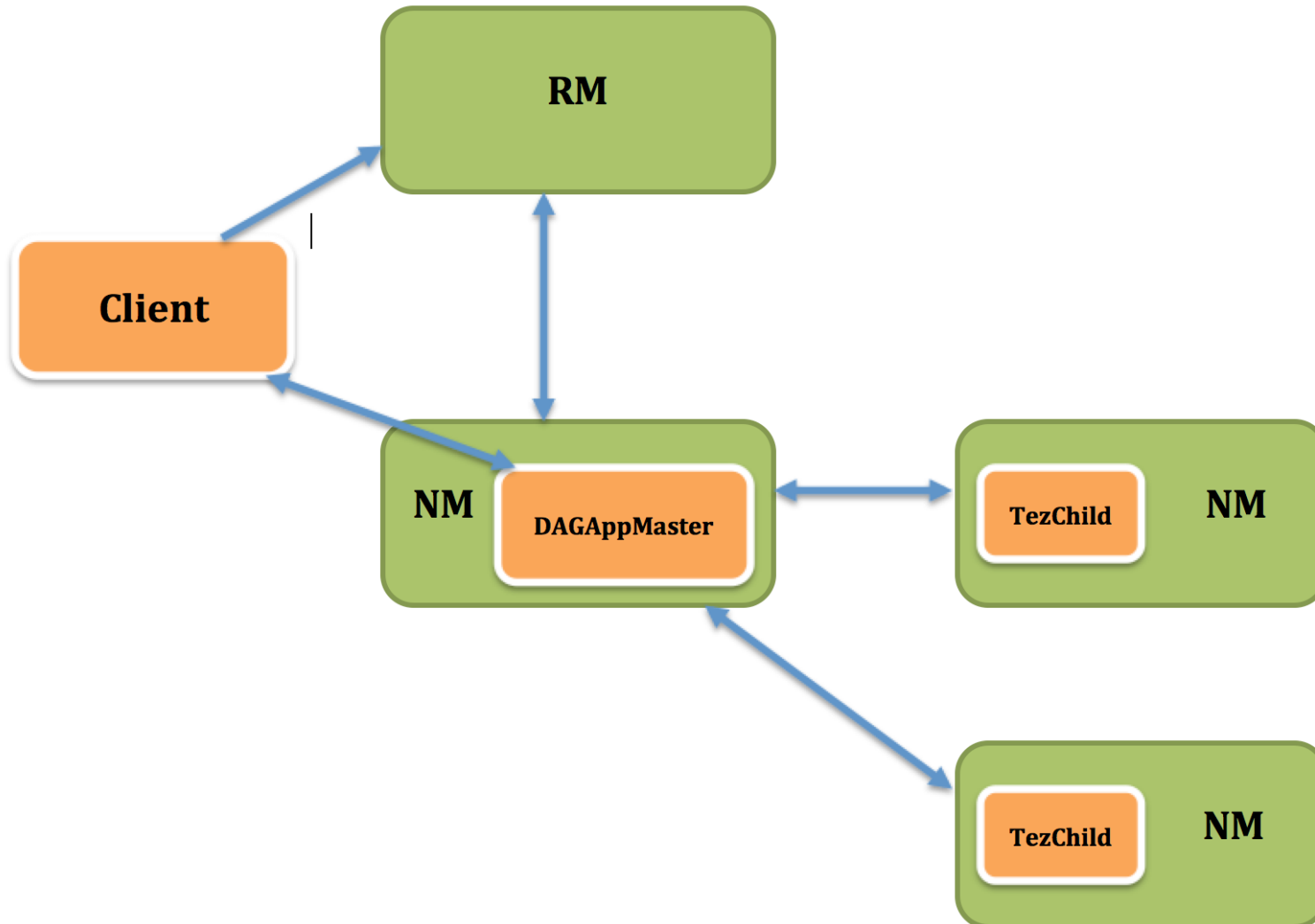


Pig/Hive - Tez

Outline

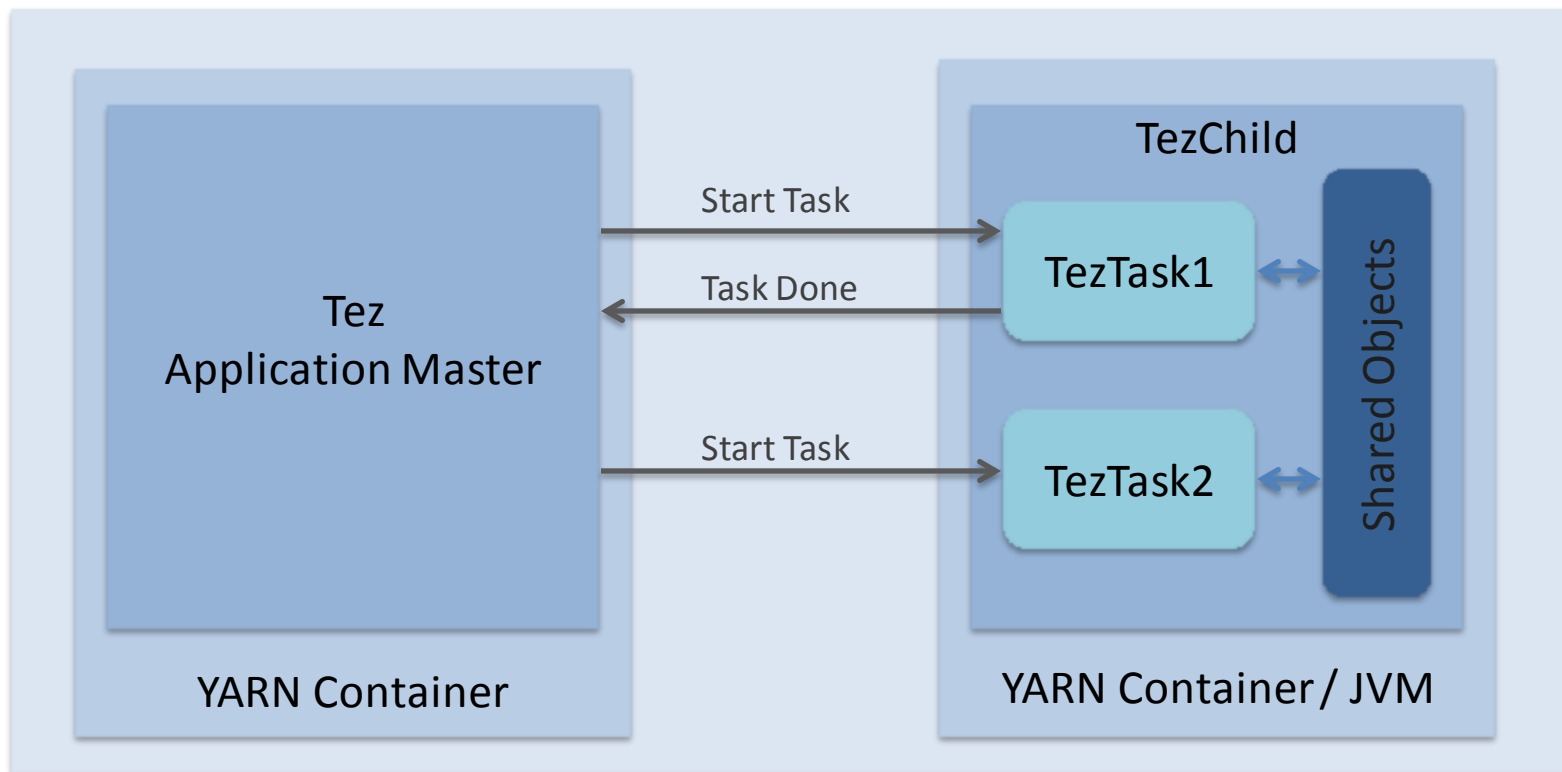
- Tez Introduction
- Tez API
- Tez Internal
- Tez Project Info
- Q & A

Tez System Diagram



Tez – Container Re-Use

- Reuse YARN containers/JVMs to launch new tasks
- Reduce scheduling and launching delays
- Shared in-memory data across tasks
- JVM JIT friendly execution

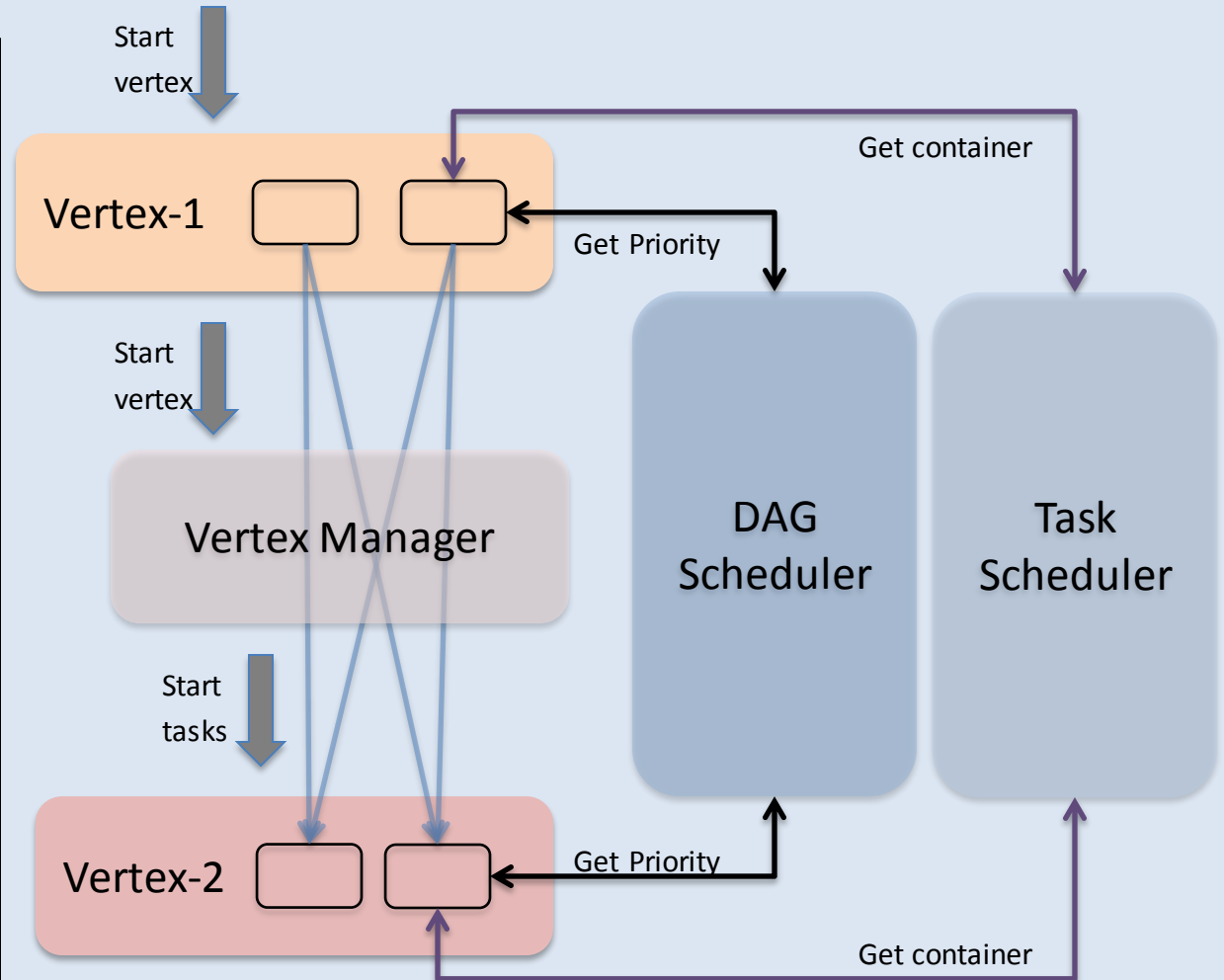


Tez – Customizable Core Engine

- **Vertex Manager**
 - Determines task parallelism
 - Determines when tasks in a vertex can start.
- **DAG Scheduler**

Determines priority of task
- **Task Scheduler**

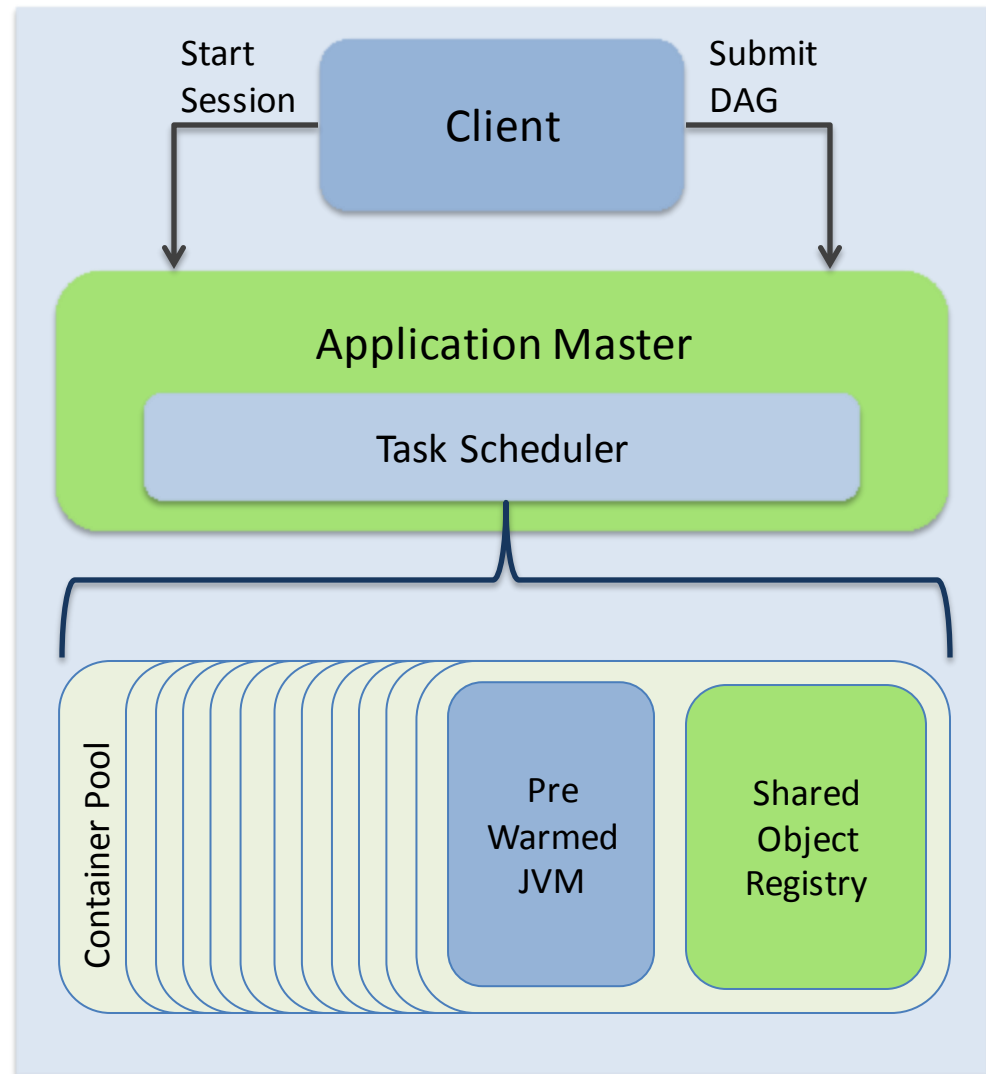
Allocates containers from YARN and assigns them to tasks



Tez – Sessions

Sessions

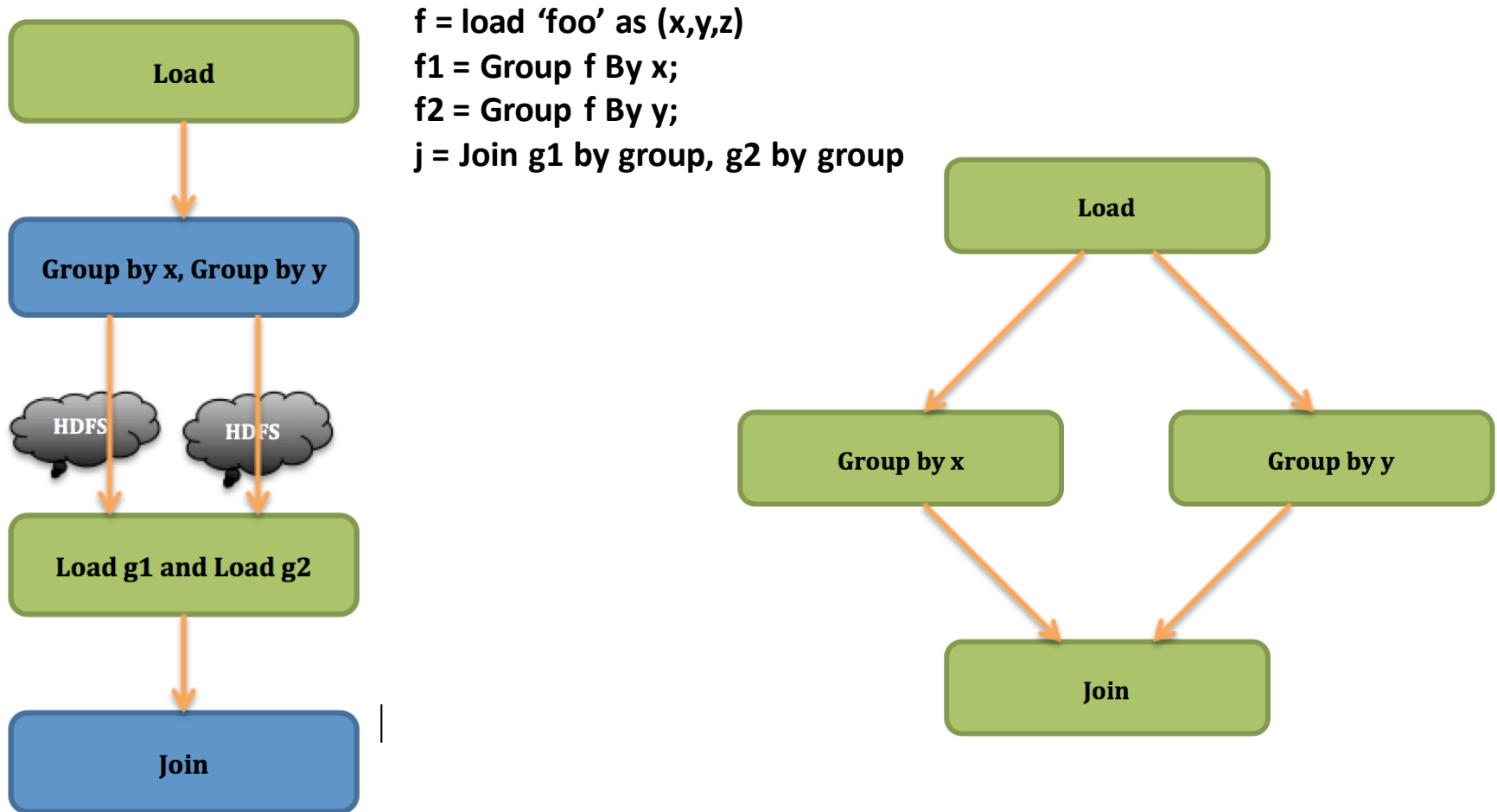
- Standard concepts of pre-launch and pre-warm applied
- Key for interactive queries
- Represents a connection between the user and the cluster
- Multiple DAGs executed in the same session
- Containers re-used across queries
- Takes care of data locality and releasing resources when idle



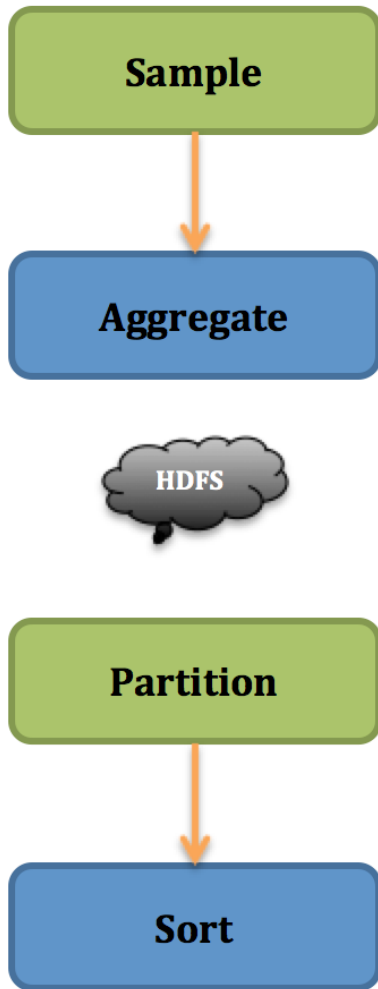
Use case of Tez

- **Split Group by + Join**
- **Orderby**
- **Automatic Reduce Parallelism**
- **Reduce Slow Start/Pre-launch**

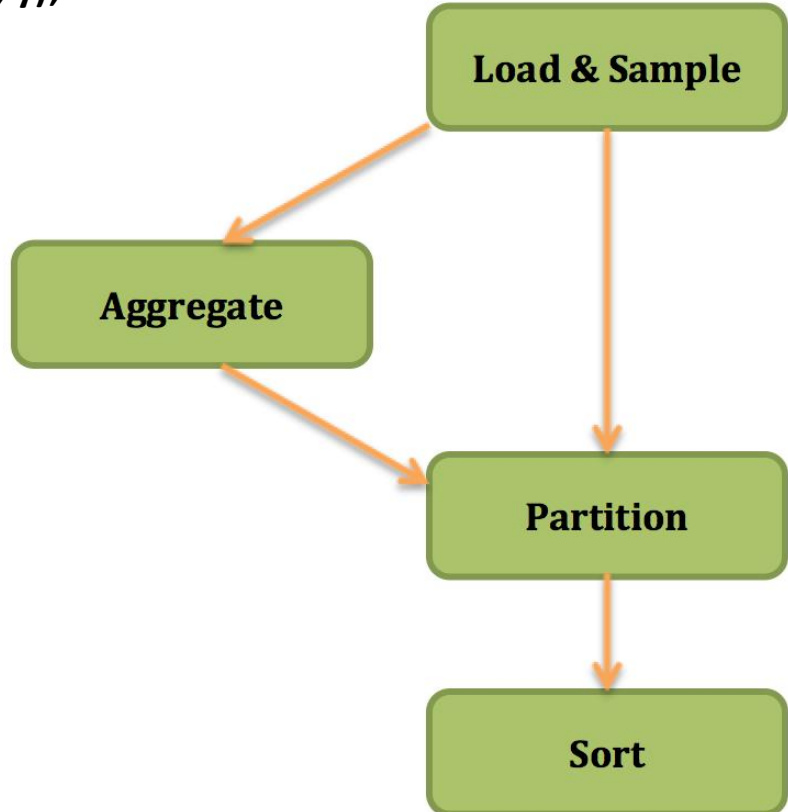
Split Group by + Join



Orderby



f = Load 'foo' as (x, y);
o = Order f by x;



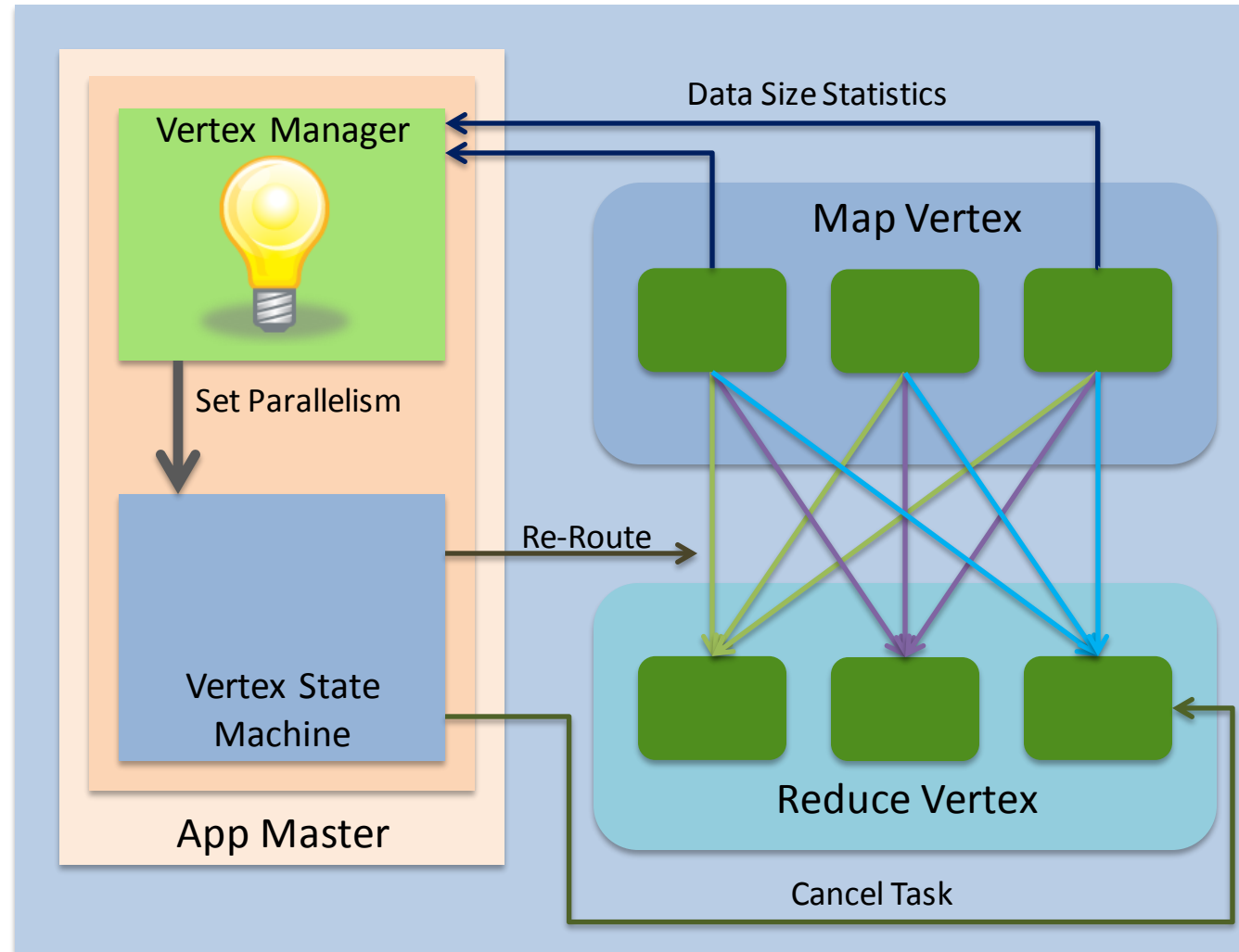
Tez – Automatic Reduce Parallelism

Event Model

Map tasks send data statistics events to the Reduce Vertex Manager.

Vertex Manager

Pluggable application logic that understands the data statistics and can formulate the correct parallelism. Advises vertex controller on parallelism



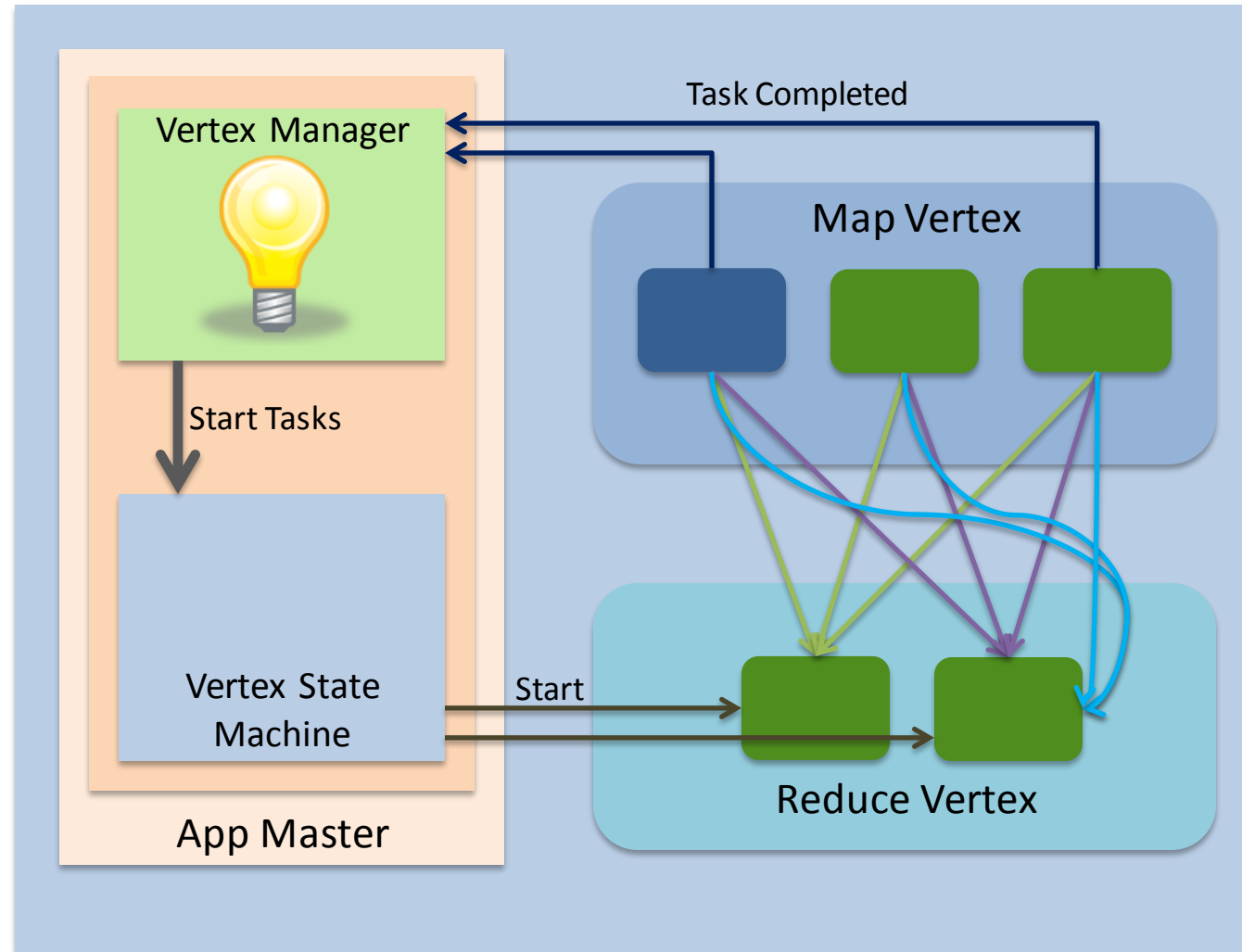
Tez – Reduce Slow Start/Pre-launch

Event Model

Map completion events sent to the Reduce Vertex Manager.

Vertex Manager

Pluggable application logic that understands the data size. Advises the vertex controller to launch the reducers before all maps have completed so that shuffle can start.



Outline

- Tez Introduction
- Tez API
- Tez Internal
- Tez Project Status
- Q & A

Tez – Current status

- **Apache Top Level Project**

- Rapid development. Over 1500 jiras opened. Over 1100 resolved
- Growing community of contributors and users
- Latest release is 0.5

- **Focus on stability**

- Testing and quality are highest priority
- Code ready and deployed on multi-node environments at scale

- **Support for a vast topology of DAGs**

- Already functionally equivalent to Map Reduce. Existing Map Reduce jobs can be executed on Tez with few or no changes
- Apache Hive 0.13 release supports Tez as an execution engine (HIVE-4660)
- Apache Pig port to Tez is also done (PIG-3446)
- Cascading 3.0 support Tez

Tez – Adoption

- **Apache Hive**
 - Hadoop standard for declarative access via SQL-like interface
- **Apache Pig**
 - Hadoop standard for procedural scripting and pipeline processing
- **Cascading**
 - Developer friendly Java API and SDK
 - Scalding (Scala API on Cascading)
- **Commercial Vendors**
 - ETL : Use Tez instead of MR or custom pipelines
 - Analytics Vendors : Use Tez as a target platform for scaling parallel analytical tools to large data-sets

Tez – Community

- **Early adopters and code contributors welcome**
 - Adopters to drive more scenarios. Contributors to make them happen.
- **Tez meetup for developers and users**
 - <http://www.meetup.com/Apache-Tez-User-Group>
- **Technical blog series**
 - <http://hortonworks.com/blog/apache-tez-a-new-chapter-in-hadoop-data-processing>
- **Useful links**
 - Work tracking: <https://issues.apache.org/jira/browse/TEZ>
 - Code: <https://github.com/apache/tez>
 - Developer list: dev@tez.apache.org
User list: user@tez.apache.org
Issues list: issues@tez.apache.org

Tez VS Spark

Tez	Spark
Solve DAG Computation	Solve DAG Computation
Integrate Yarn from its beginning	
Borrow lots of work from MapReduce (e.g. Shuffle)	Start from scratch
Design for computing engine	For general application developer
More API on the execution engine	More friendly on API
Good performance (have potential to improve once the streaming shuffle is implemented or the in-memory HDFS is integrated)	Better performance
Better scalability and stable	

Thank You!

Questions & Answers

