Apache Hadoop YARN

Enabling next generation data applications

Arun C. Murthy
Founder & Architect
@acmurthy (@hortonworks)

Hello!

- **Founder/Architect at Hortonworks Inc.**
  - Lead - Map-Reduce/YARN/Tez
  - Formerly, Architect Hadoop MapReduce, Yahoo
  - Responsible for running Hadoop MapReduce as a service for all of Yahoo (~50k nodes footprint)

- **Apache Hadoop, ASF**
  - Frmr. VP, Apache Hadoop, ASF (Chair of Apache Hadoop PMC)
  - Long-term Committer/PMC member (full time >6 years)
  - Release Manager for hadoop-2.x

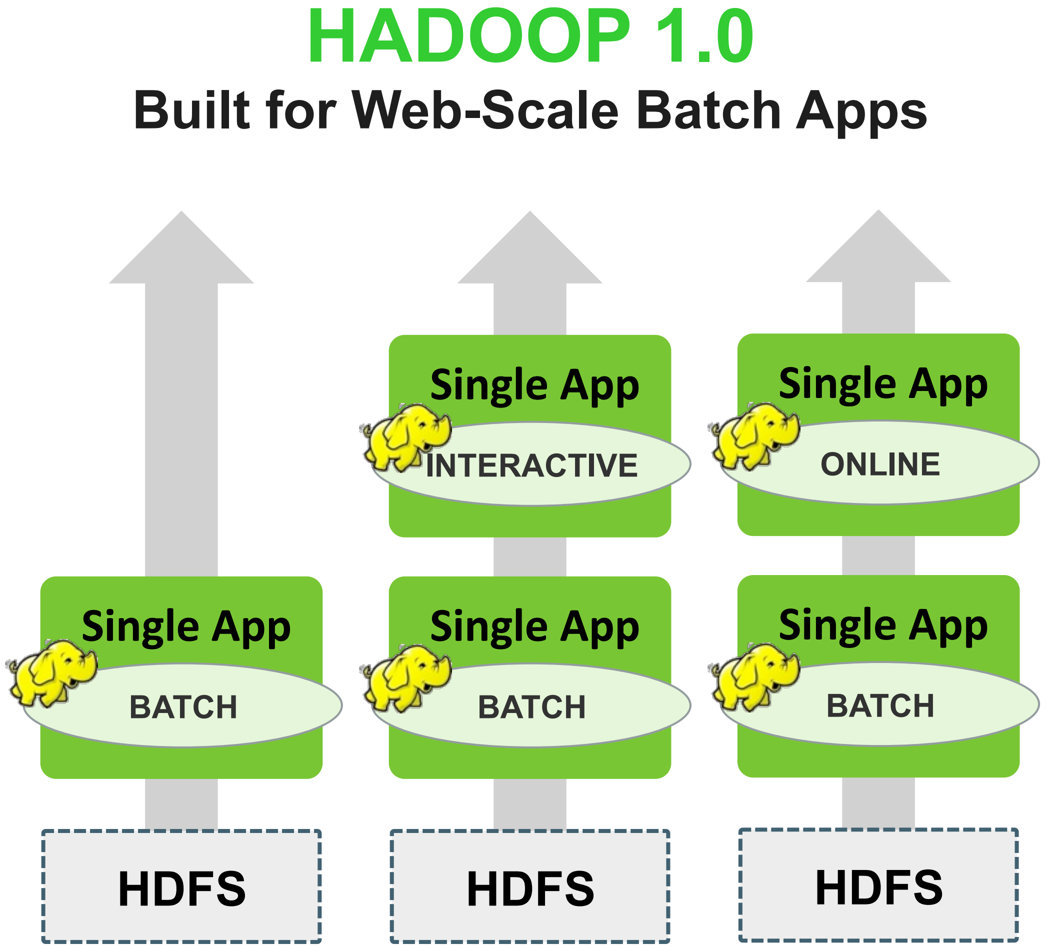
Agenda

- Why YARN?
- YARN Architecture and Concepts
- Building applications on YARN
- Next Steps

Agenda

- **Why YARN?**
- YARN Architecture and Concepts
- Building applications on YARN
- Next Steps

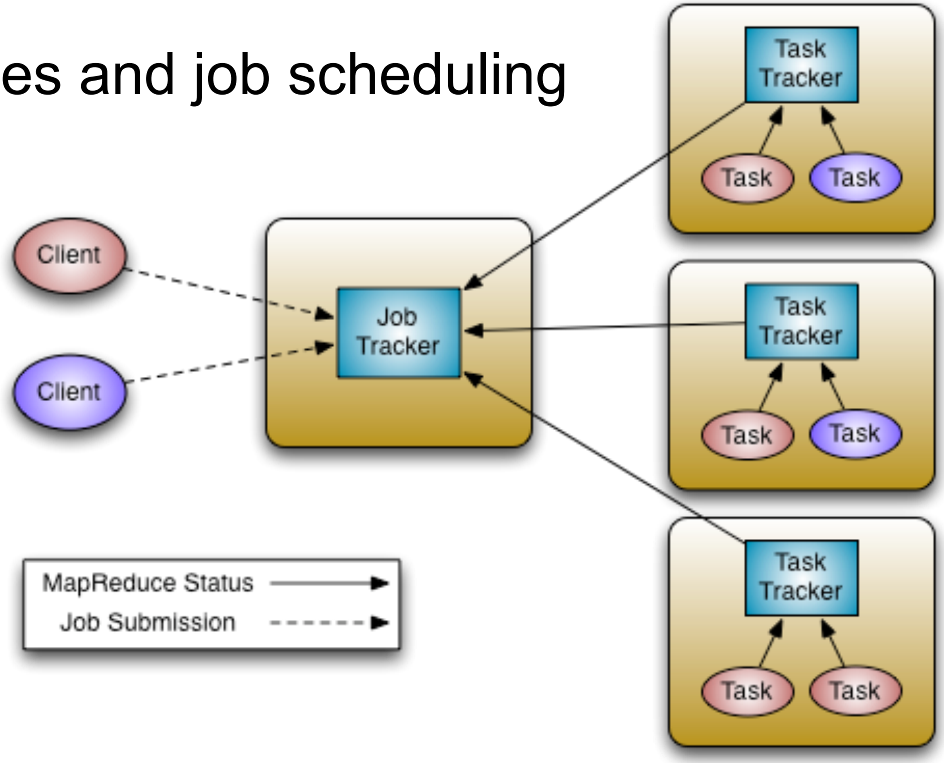
The 1st Generation of Hadoop: Batch



- All other usage patterns must leverage that same infrastructure
- Forces the creation of silos for managing mixed workloads

Hadoop MapReduce Classic

- **JobTracker**
  - Manages cluster resources and job scheduling
- **TaskTracker**
  - Per-node agent
  - Manage tasks



MapReduce Classic: Limitations

- **Scalability**
  - Maximum Cluster size – 4,000 nodes
  - Maximum concurrent tasks – 40,000
  - Coarse synchronization in JobTracker
- **Availability**
  - Failure kills all queued and running jobs
- **Hard partition of resources into map and reduce slots**
  - Low resource utilization
- **Lacks support for alternate paradigms and services**
  - Iterative applications implemented using MapReduce are 10x slower

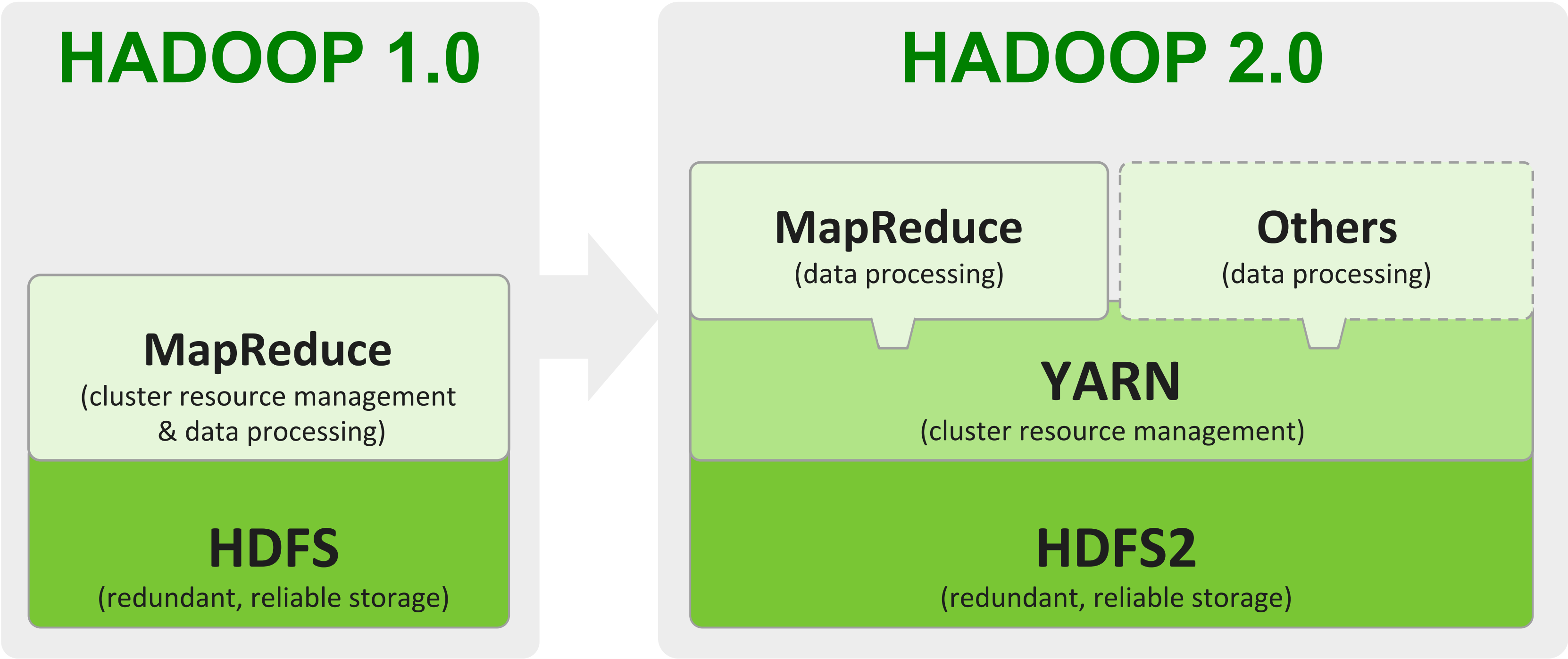
Our Vision: Hadoop as Next-Gen Platform

Single Use System

Batch Apps

Multi Purpose Platform

Batch, Interactive, Online, Streaming, …



HADOOP 1.0

MapReduce
(cluster resource management & data processing)

HDFS
(redundant, reliable storage)

HADOOP 2.0

MapReduce
(data processing)

Others
(data processing)

YARN
(cluster resource management)

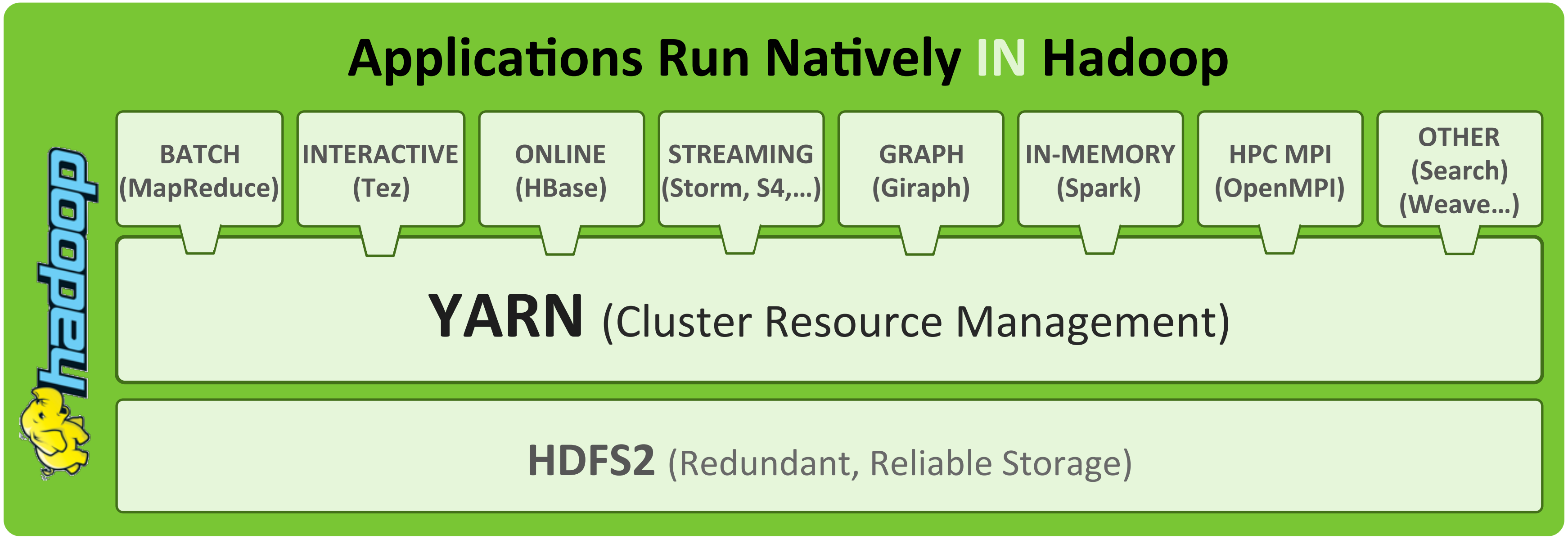
HDFS2
(redundant, reliable storage)

YARN: Taking Hadoop Beyond Batch

Store ALL DATA in one place…

Interact with that data in MULTIPLE WAYS

with Predictable Performance and Quality of Service



5 Key Benefits of YARN

1. **Scale**
2. **New Programming Models & Services**
3. **Improved cluster utilization**
4. **Agility**
5. **Beyond Java**

Agenda

- Why YARN
- **YARN Architecture and Concepts**
- Building applications on YARN
- Next Steps

A Brief History of YARN

- **Originally conceived & architected by the team at Yahoo!**
  - Arun Murthy created the original JIRA in 2008 and led the PMC

- **The team at Hortonworks has been working on YARN for 4 years**

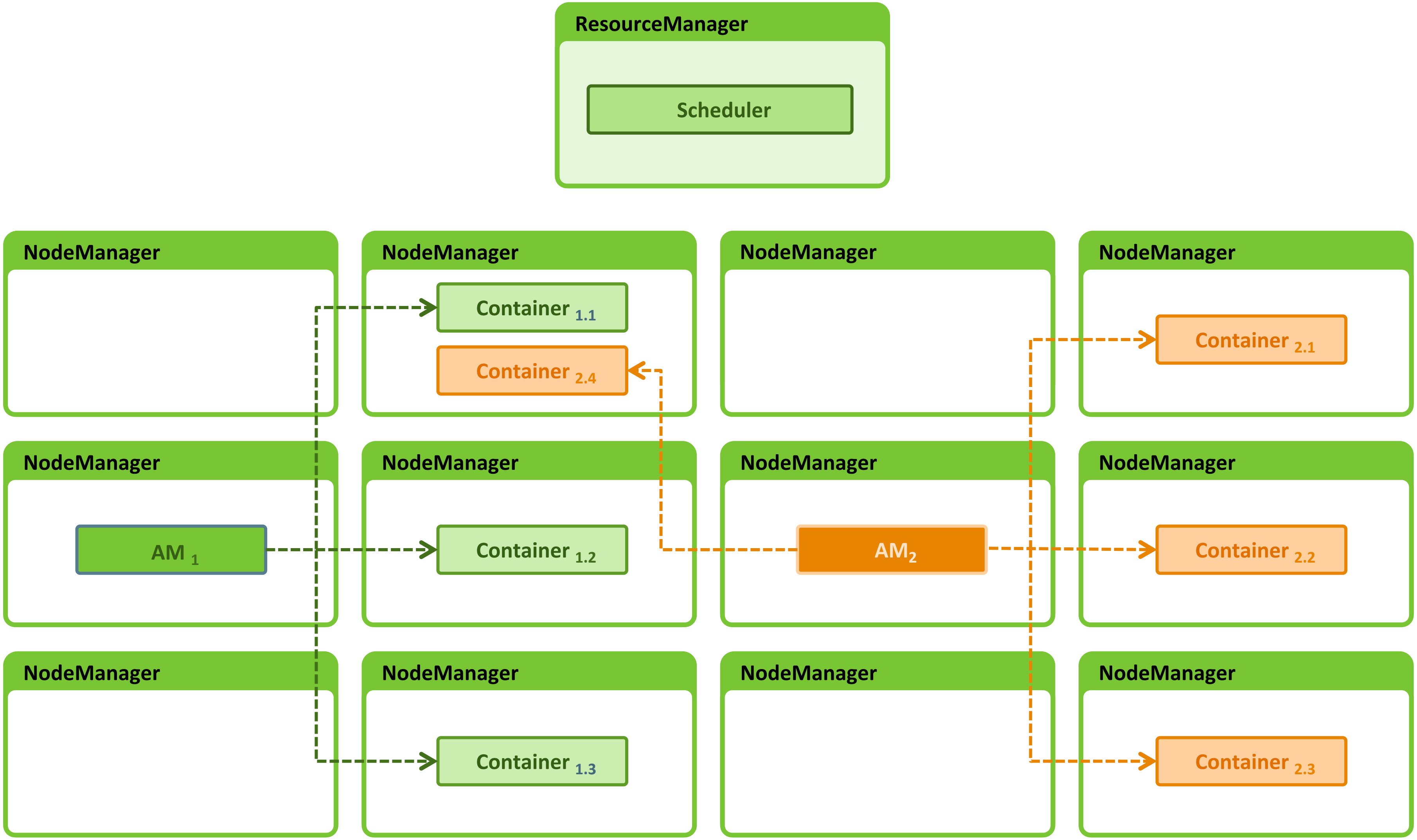
- **YARN based architecture running at scale at Yahoo!**
  - Deployed on 35,000 nodes for 6+ months

- **Multitude of YARN applications**

Concepts

- **Application**
  - Application is a job submitted to the framework
  - Example – Map Reduce Job
- **Container**
  - Basic unit of allocation
  - Fine-grained resource allocation across multiple resource types (memory, cpu, disk, network, gpu etc.)
    - container_0 = 2GB, 1CPU
    - container_1 = 1GB, 6 CPU
  - Replaces the fixed map/reduce slots

YARN Architecture



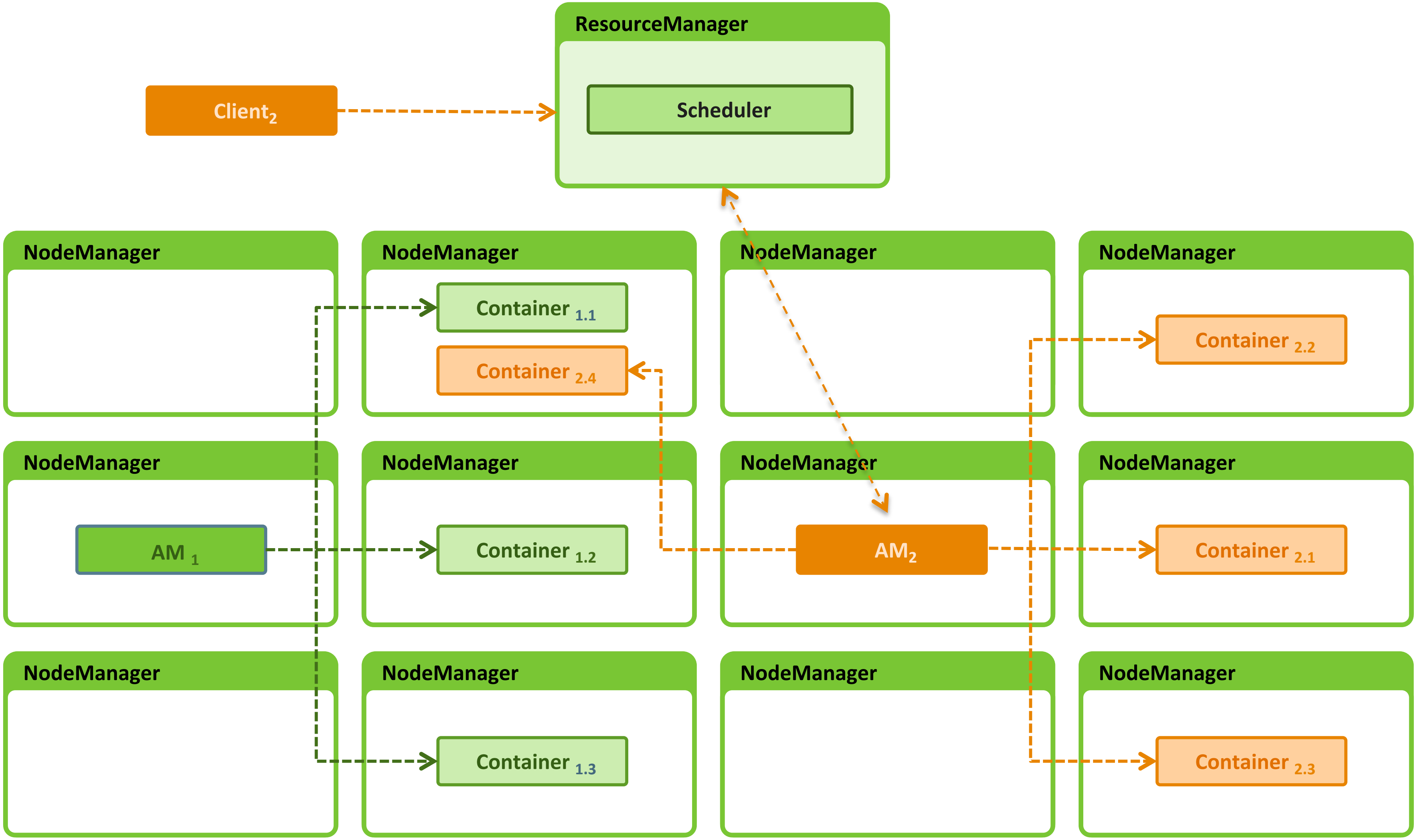
Architecture

- **Resource Manager**
  - Global resource scheduler
  - Hierarchical queues
- **Node Manager**
  - Per-machine agent
  - Manages the life-cycle of container
  - Container resource monitoring
- **Application Master**
  - Per-application
  - Manages application scheduling and task execution
  - E.g. MapReduce Application Master

Design Centre

- **Split up the two major functions of JobTracker**
  - Cluster resource management
  - Application life-cycle management
- **MapReduce becomes user-land library**

YARN Architecture - Walkthrough



Review - Benefits of YARN

1. **Scale**
2. **New Programming Models & Services**
3. **Improved cluster utilization**
4. **Agility**
5. **Beyond Java**

Agenda

- Why YARN
- YARN Architecture and Concepts
- **Building applications on YARN**
- Next Steps

YARN Applications

- **Data processing applications and services**
  - Online Serving – HOYA (HBase on YARN)
  - Real-time event processing – Storm, S4, other commercial platforms
  - Tez – Generic framework to run a complex DAG
  - MPI: OpenMPI, MPICH2
  - Master-Worker
  - Machine Learning: Spark
  - Graph processing: Giraph
  - Enabled by allowing the use of paradigm-specific application master

Run all on the same Hadoop cluster!

YARN – Implementing Applications

- **What APIs do I need to use?**
  - Only three *protocols*
    - Client to ResourceManager
      - Application submission
    - ApplicationMaster to ResourceManager
      - Container allocation
    - ApplicationMaster to NodeManager
      - Container launch
  - Use client libraries for all 3 actions
    - Module `yarn-client`
    - Provides both synchronous and asynchronous libraries
    - Use 3rd party like Weave
      - http://continuuity.github.io/weave/

YARN – Implementing Applications

- **What do I need to do?**
  - Write a submission Client
  - Write an ApplicationMaster (well copy-paste)
    - *`DistributedShell`* *is the new* `WordCount`
  - Get containers, run whatever you want!

YARN – Implementing Applications

- **What else do I need to *know*?**
  - Resource Allocation & Usage
    - `ResourceRequest`
    - `Container`
    - `ContainerLaunchContext`
    - `LocalResource`
  - ApplicationMaster
    - `ApplicationId`
    - `ApplicationAttemptId`
    - `ApplicationSubmissionContext`

YARN – Resource Allocation & Usage

- **`ResourceRequest`**
  - Fine-grained resource *ask* to the ResourceManager
  - Ask for a specific amount of resources (memory, cpu etc.) on a specific machine or rack
  - Use special value of * for resource name for *any* machine

`ResourceRequest`
`priority`
`resourceName`
`capability`
`numContainers`

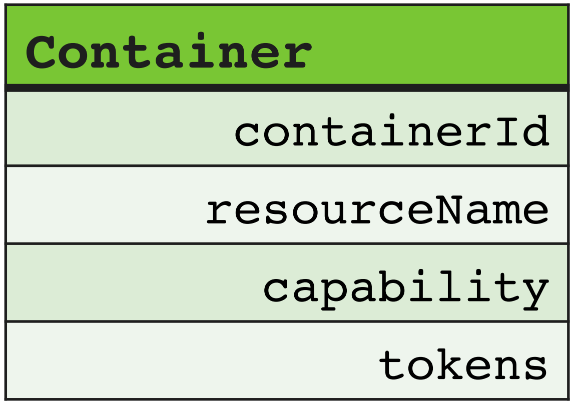
YARN – Resource Allocation & Usage

- **`ResourceRequest`**

priority	capability	resourceName	numContainers
0	<2gb, 1 core>	host01	1
		rack0	1
		*	1
1	<4gb, 1 core>	*	1

YARN – Resource Allocation & Usage

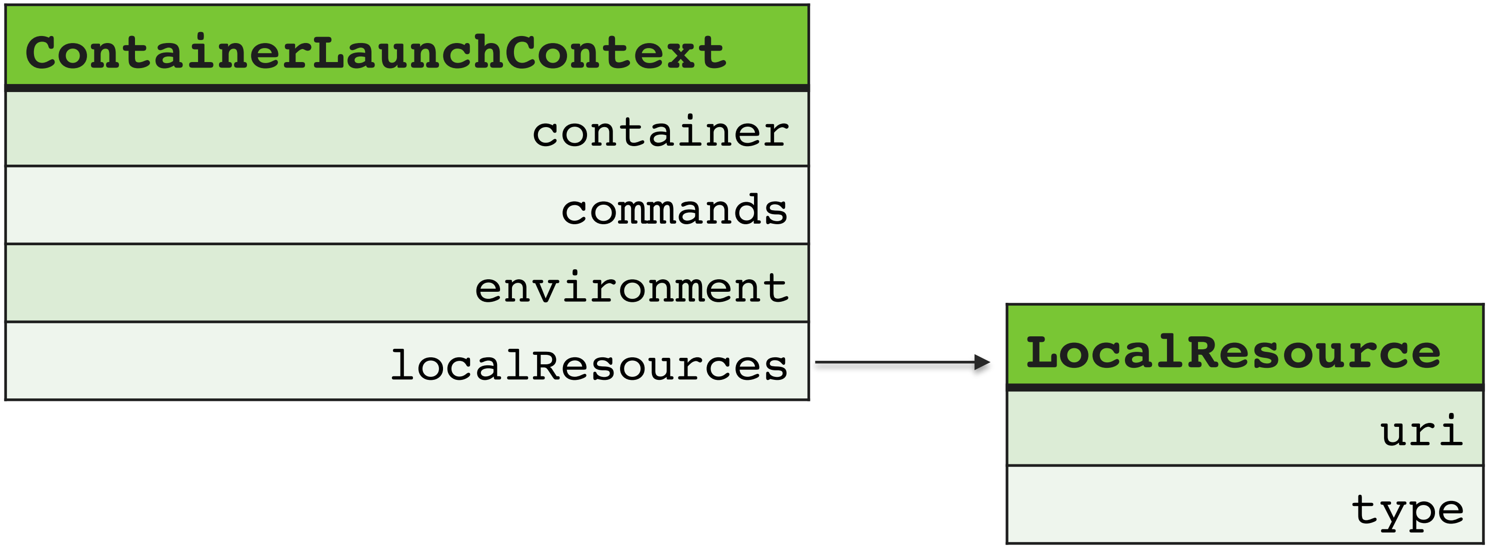
- **`Container`**
  - The basic unit of allocation in YARN
  - The *result* of the `ResourceRequest` provided by ResourceManager to the ApplicationMaster
  - A *specific amount of resources* (cpu, memory etc.) on a *specific machine*



Container
containerId
resourceName
capability
tokens

YARN – Resource Allocation & Usage

- **`ContainerLaunchContext`**
  - The context provided by ApplicationMaster to NodeManager to launch the `Container`
  - *Complete specification for a process*
  - `LocalResource` used to specify container binary and dependencies
    - NodeManager responsible for downloading from shared namespace (typically HDFS)



ContainerLaunchContext

container

commands

environment

localResources

LocalResource

uri

type

YARN - ApplicationMaster

- **ApplicationMaster**
  - Per-application controller aka *`container_0`*
  - Parent for all containers of the application
    - ApplicationMaster negotiates all it's containers from ResourceManager
  - ApplicationMaster container is child of ResourceManager
    - Think *init* process in Unix
    - RM restarts the ApplicationMaster *attempt* if required (unique `ApplicationAttemptId`)
  - Code for application is submitted along with Application itself

YARN - ApplicationMaster

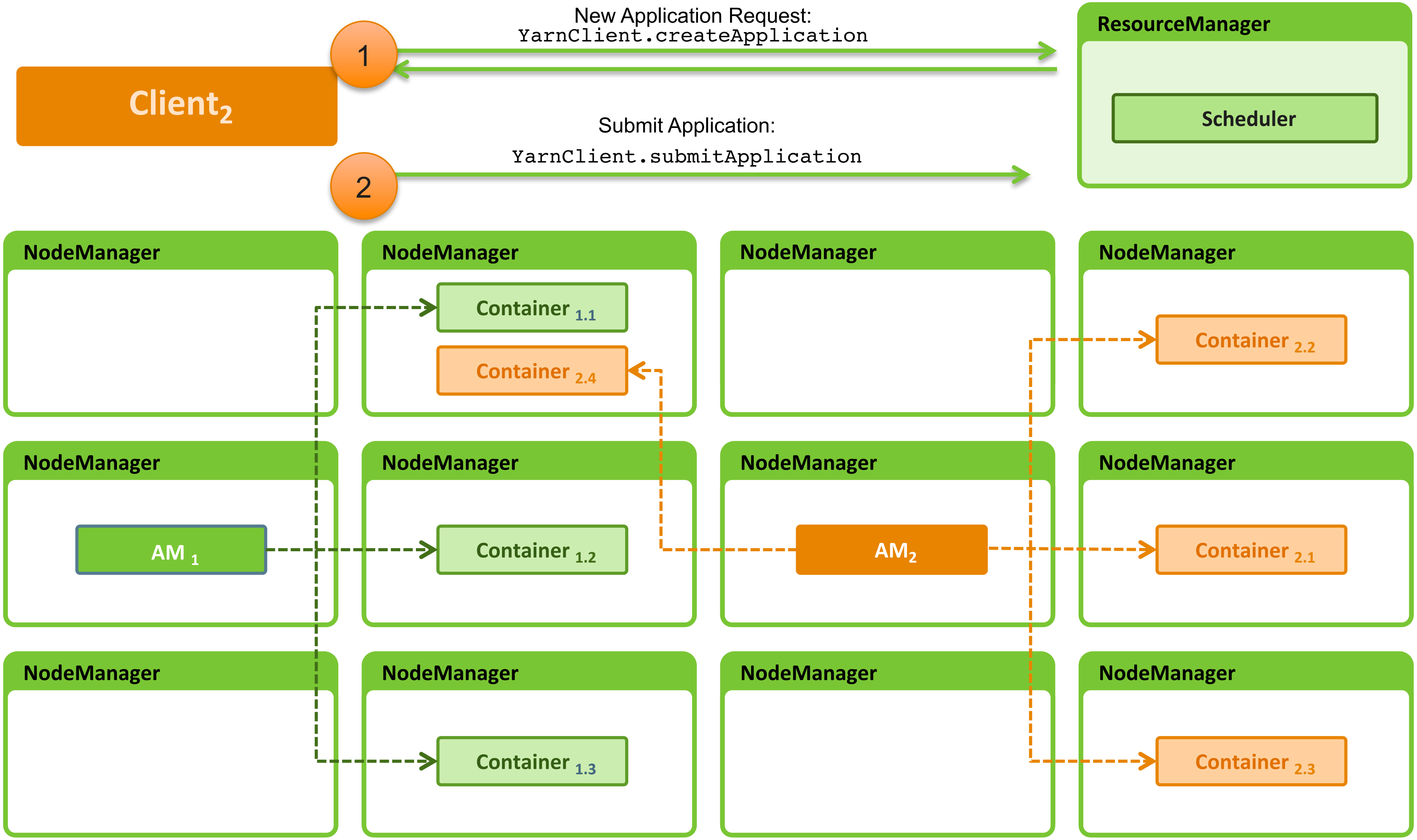
- **ApplicationMaster**
  - `ApplicationSubmissionContext` is the complete specification of the ApplicationMaster, provided by Client
  - ResourceManager responsible for *allocating* and *launching* ApplicationMaster container

ApplicationSubmissionContext
resourceRequest
containerLaunchContext
appName
queue

YARN Application API - Overview

- `hadoop-yarn-client` module
- `YarnClient` is submission client api
- Both synchronous & asynchronous APIs for resource allocation and container start/stop
- Synchronous API
  - `AMRMClient`
  - `AMNMClient`
- Asynchronous API
  - `AMRMClientAsync`
  - `AMNMClientAsync`

YARN Application API – The Client



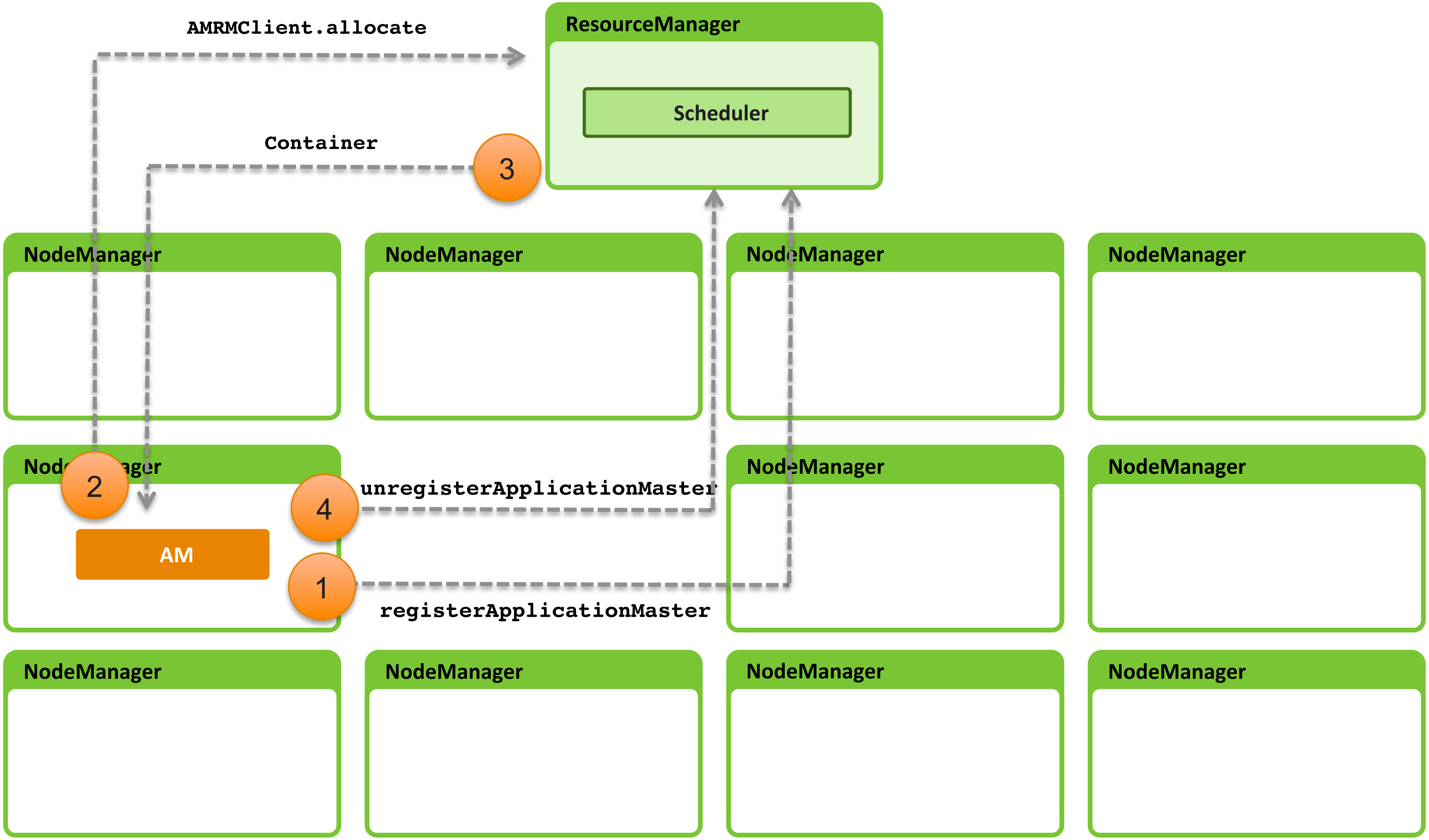
New Application Request:
`YarnClient.createApplication`

Submit Application:
`YarnClient.submitApplication`

YARN Application API – The Client

- **`YarnClient`**
  - `createApplication` to create application
  - `submitApplication` to start application
    - Application developer needs to provide `ApplicationSubmissionContext`
  - APIs to get other information from ResourceManager
    - `getAllQueues`
    - `getApplications`
    - `getNodeReports`
  - APIs to manipulate submitted application e.g. `killApplication`

YARN Application API – Resource Allocation



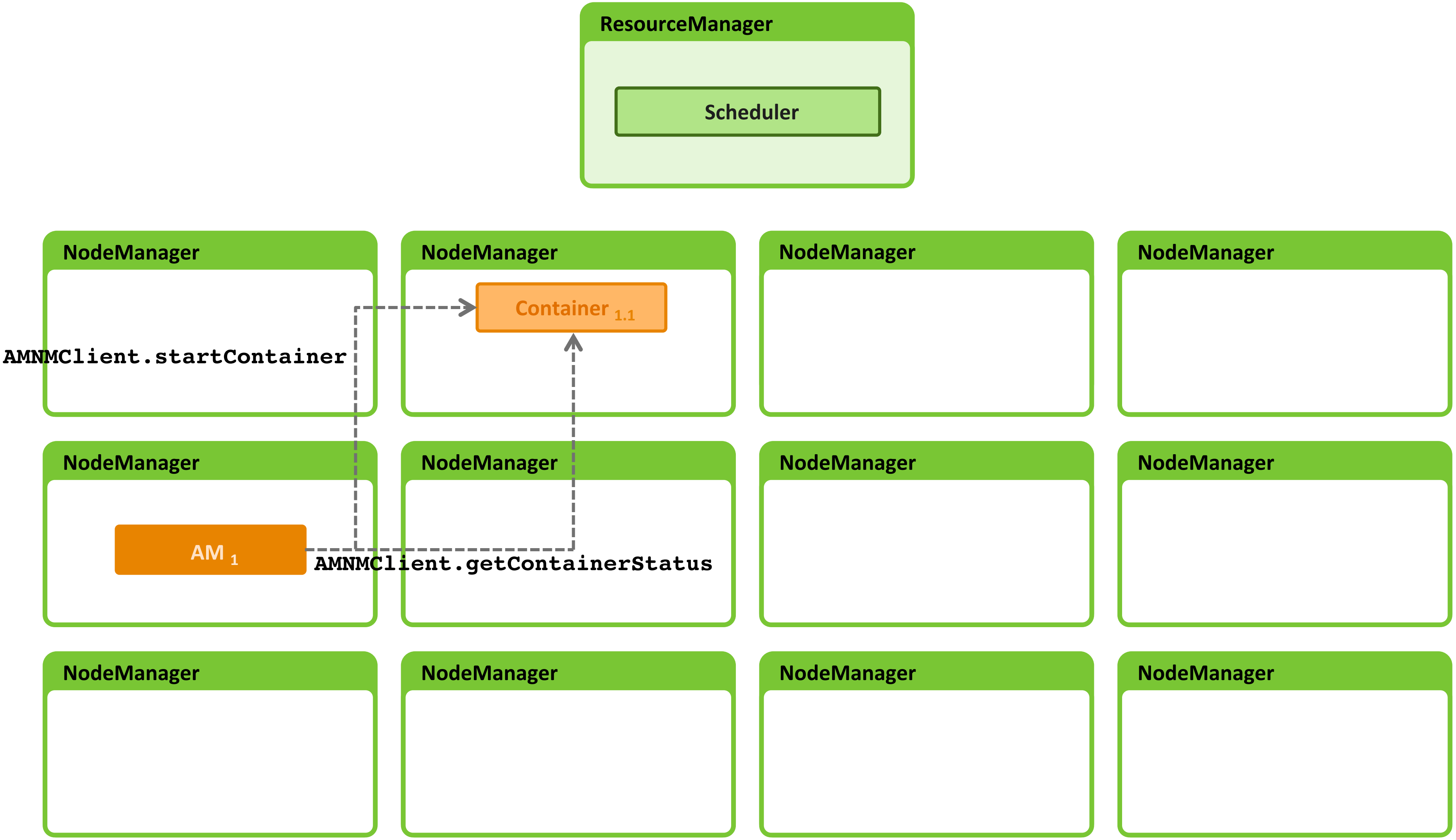
YARN Application API – Resource Allocation

- **`AMRMClient` - Synchronous API for ApplicationMaster to interact with ResourceManager**
  - Prologue / epilogue – `registerApplicationMaster` / `unregisterApplicationMaster`
  - Resource negotiation with ResourceManager
    - Internal book-keeping - `addContainerRequest` / `removeContainerRequest` / `releaseAssignedContainer`
    - Main API – `allocate`
  - Helper APIs for cluster information
    - `getAvailableResources`
    - `getClusterNodeCount`

YARN Application API – Resource Allocation

- **`AMRMClientAsync` - Asynchronous API for ApplicationMaster**
  - Extension of `AMRMClient` to provide asynchronous `CallbackHandler`
  - Callbacks make it easier to build mental model of interaction with ResourceManager for the application developer
    - `onContainersAllocated`
    - `onContainersCompleted`
    - `onNodesUpdated`
    - `onError`
    - `onShutdownRequest`

YARN Application API – Using Resources



YARN Application API – Using Resources

- **`AMNMClient` - Synchronous API for ApplicationMaster to launch / stop containers at NodeManager**
  - Simple (trivial) APIs
    - `startContainer`
    - `stopContainer`
    - `getContainerStatus`

YARN Application API – Using Resources

- **`AMNMClient` - Asynchronous API for ApplicationMaster to launch / stop containers at NodeManager**
  - Simple (trivial) APIs
    - `startContainerAsync`
    - `stopContainerAsync`
    - `getContainerStatusAsync`
  - `CallbackHandler` to make it easier to build mental model of interaction with NodeManager for the application developer
    - `onContainerStarted`
    - `onContainerStopped`
    - `onStartContainerError`
    - `onContainerStatusReceived`

YARN Application API - Development

- **Un-Managed Mode for ApplicationMaster**
  - Run ApplicationMaster on development machine rather than in-cluster
    - No submission client
  - `hadoop-yarn-applications-unmanaged-am-launcher`
  - Easier to step through debugger, browse logs etc.

```
$ bin/hadoop jar hadoop-yarn-applications-unmanaged-am-launcher.jar \
      Client \
      –jar my-application-master.jar \
      –cmd ‘java MyApplicationMaster <args>’
```

Sample YARN Application – DistributedShell

- **Overview**
  - YARN application to run *n* copies for a Shell command
  - Simplest example of a YARN application – get *n* containers and run a specific Unix command

```
$ bin/hadoop jar hadoop-yarn-applications-distributedshell.jar \
      org.apache.hadoop.yarn.applications.distributedshell.Client \
      —shell_command ‘/bin/date’ \
      —num_containers <n>
```

Code: *https://github.com/hortonworks/simple-yarn-app*

Sample YARN Application – DistributedShell

- **Code Overview**
  - User submits application to ResourceManager via `org.apache.hadoop.yarn.applications.distributedshell.Client`
    - `Client` provides `ApplicationSubmissionContext` to the ResourceManager
  - It is responsibility of `org.apache.hadoop.yarn.applications.distributedshell.ApplicationMaster` to negotiate *n* containers
    - `ApplicationMaster` launches containers with the user-specified *command* as `ContainerLaunchContext.commands`

Sample YARN Application – DistributedShell

- **Client – Code Walkthrough**
  - `hadoop-yarn-client` module
  - Steps:
    - `YarnClient.createApplication`
    - Specify `ApplicationSubmissionContext`, in particular, `ContainerLaunchContext` with `commands`, and other key pieces such as resource `capability` for ApplicationMaster container and `queue`, `appName`, `appType` etc.
    - `YarnClient.submitApplication`

```java
    // Create yarnClient
    YarnClient yarnClient = YarnClient.createYarnClient();
    yarnClient.init(new Configuration());
    yarnClient.start();

    // Create application via yarnClient
    YarnClientApplication app = yarnClient.createApplication();

```

Sample YARN Application – DistributedShell

- **Client – Code Walkthrough**

```
// Set up the container launch context for the application master
ContainerLaunchContext amContainer =
    Records.newRecord(ContainerLaunchContext.class);
List<String> command = new List<String>();
commands.add("$JAVA_HOME/bin/java");
commands.add("-Xmx256M");
commands.add(
    "org.apache.hadoop.yarn.applications.distributedshell.ApplicationMaster");
commands.add("--container_memory 1024” );
commands.add("--container_cores 1”);
commands.add("--num_containers 3");
amContainer.setCommands(commands);

// Set up resource type requirements for ApplicationMaster
Resource capability = Records.newRecord(Resource.class);
capability.setMemory(256);
capability.setVirtualCores(2);

```

Command to launch ApplicationMaster process

Resources required for ApplicationMaster `container`

Sample YARN Application – DistributedShell

- **Client – Code Walkthrough**

```
// Finally, set-up ApplicationSubmissionContext for the application
ApplicationSubmissionContext appContext =
    app.getApplicationSubmissionContext();
appContext.setQueue("my-queue");            // queue
appContext.setAMContainerSpec(amContainer);
appContext.setResource(capability);
appContext.setApplicationName("my-app");               // application name
appContext.setApplicationType(”DISTRIBUTED_SHELL");    // application type

// Submit application
yarnClient.submitApplication(appContext);
```

ApplicationSubmissionContext
for
ApplicationMaster

Submit application to
ResourceManager

Sample YARN Application – DistributedShell

- **ApplicationMaster – Code Walkthrough**
  - Again, `hadoop-yarn-client` module
  - Steps:
    - `AMRMClient.registerApplication`
    - Negotiate containers from ResourceManager by providing `ContainerRequest` to `AMRMClient.addContainerRequest`
    - Take the resultant `Container` returned via subsequent call to `AMRMClient.allocate`, build `ContainerLaunchContext` with `Container` and `commands`, then launch them using `AMNMClient.launchContainer`
      - Use `LocalResources` to specify software/configuration dependencies for each worker container
    - Wait till done… `AllocateResponse.getCompletedContainersStatuses` from subsequent calls to `AMRMClient.allocate`
    - `AMRMClient.unregisterApplication`

Sample YARN Application – DistributedShell

- **ApplicationMaster – Code Walkthrough**

```
// Initialize clients to ResourceManager and NodeManagers
Configuration conf = new Configuration();

AMRMClient rmClient = AMRMClientAsync.createAMRMClient();
rmClient.init(conf);
rmClient.start();

NMClient nmClient = NMClient.createNMClient();
nmClientAsync.init(conf);
nmClientAsync.start();

// Register with ResourceManager
rmClient.registerApplicationMaster("", 0, "");
```

Initialize clients to ResourceManager and NodeManagers

Register with ResourceManager

Sample YARN Application – DistributedShell

- **ApplicationMaster – Code Walkthrough**

```
// Priority for worker containers - priorities are intra-application
Priority priority = Records.newRecord(Priority.class);
priority.setPriority(0);

// Resource requirements for worker containers
Resource capability = Records.newRecord(Resource.class);
capability.setMemory(128);
capability.setVirtualCores(1);

// Make container requests to ResourceManager
for (int i = 0; i < n; ++i) {
  ContainerRequest containerAsk = new ContainerRequest(capability, null, null,
priority);
  rmClient.addContainerRequest(containerAsk);
}
```

Setup requirements for worker containers

Make resource requests to ResourceManager

Sample YARN Application – DistributedShell

- **ApplicationMaster – Code Walkthrough**

```
// Obtain allocated containers and launch
int allocatedContainers = 0;
while (allocatedContainers < n) {
  AllocateResponse response = rmClient.allocate(0);
  for (Container container : response.getAllocatedContainers()) {
    ++allocatedContainers;

    // Launch container by create ContainerLaunchContext
    ContainerLaunchContext ctx = Records.newRecord(ContainerLaunchContext.class);
    ctx.setCommands(Collections.singletonList(”/bin/date”));
    nmClient.startContainer(container, ctx);
  }
  Thread.sleep(100);
}
```

Setup requirements for worker containers

Make resource requests to ResourceManager

Sample YARN Application – DistributedShell

- **ApplicationMaster – Code Walkthrough**

Wait for containers to complete successfully

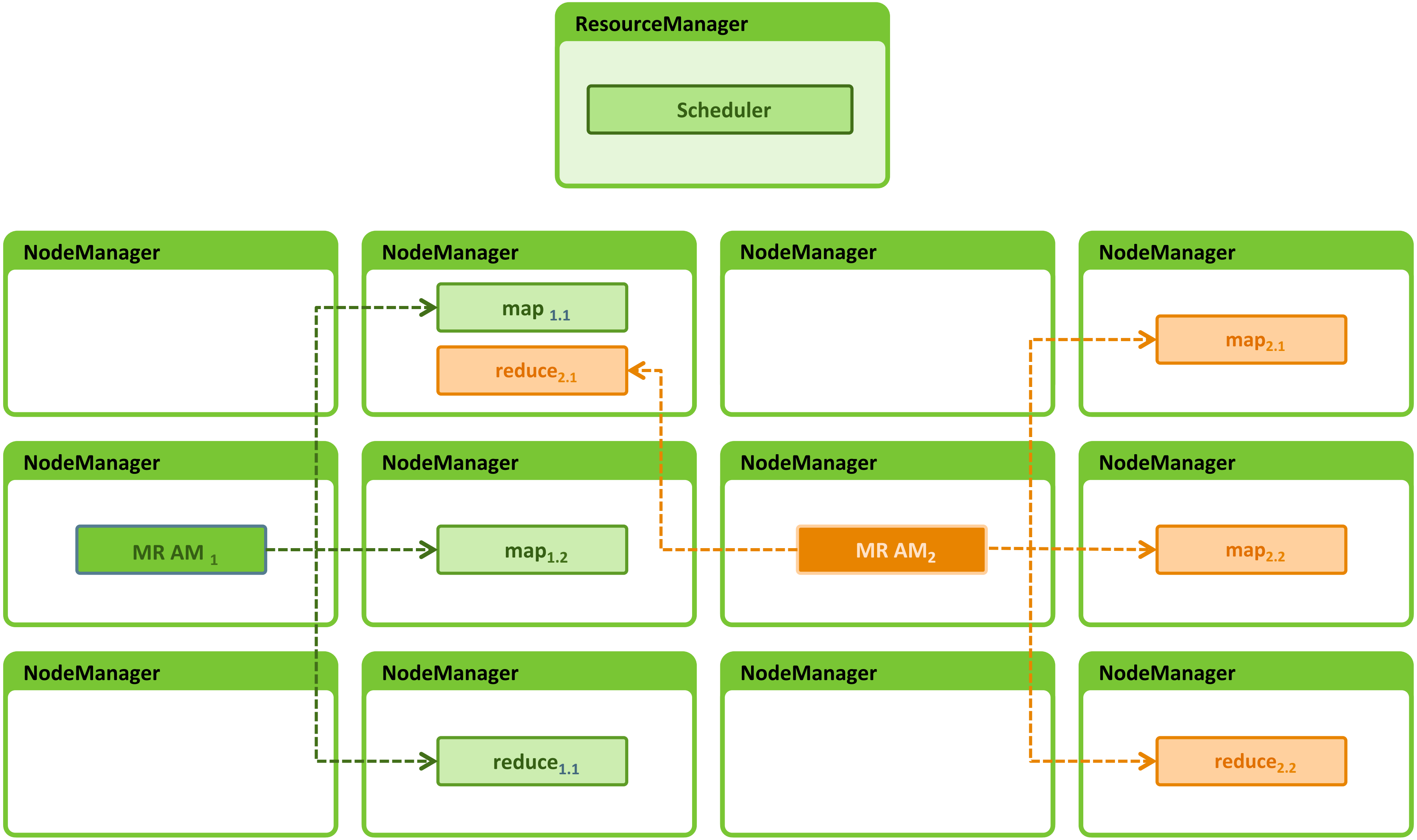
```java
// Now wait for containers to complete
int completedContainers = 0;
while (completedContainers < n) {
  AllocateResponse response = rmClient.allocate(completedContainers/n);
  for (ContainerStatus status : response.getCompletedContainersStatuses()) {
    if (status.getExitStatus() == 0) {
      ++completedContainers;
    }
  }
  Thread.sleep(100);
}

// Un-register with ResourceManager
rmClient.unregisterApplicationMaster(SUCCEEDED, "", "");
```

Un-register with ResourceManager

Apache Hadoop MapReduce on YARN

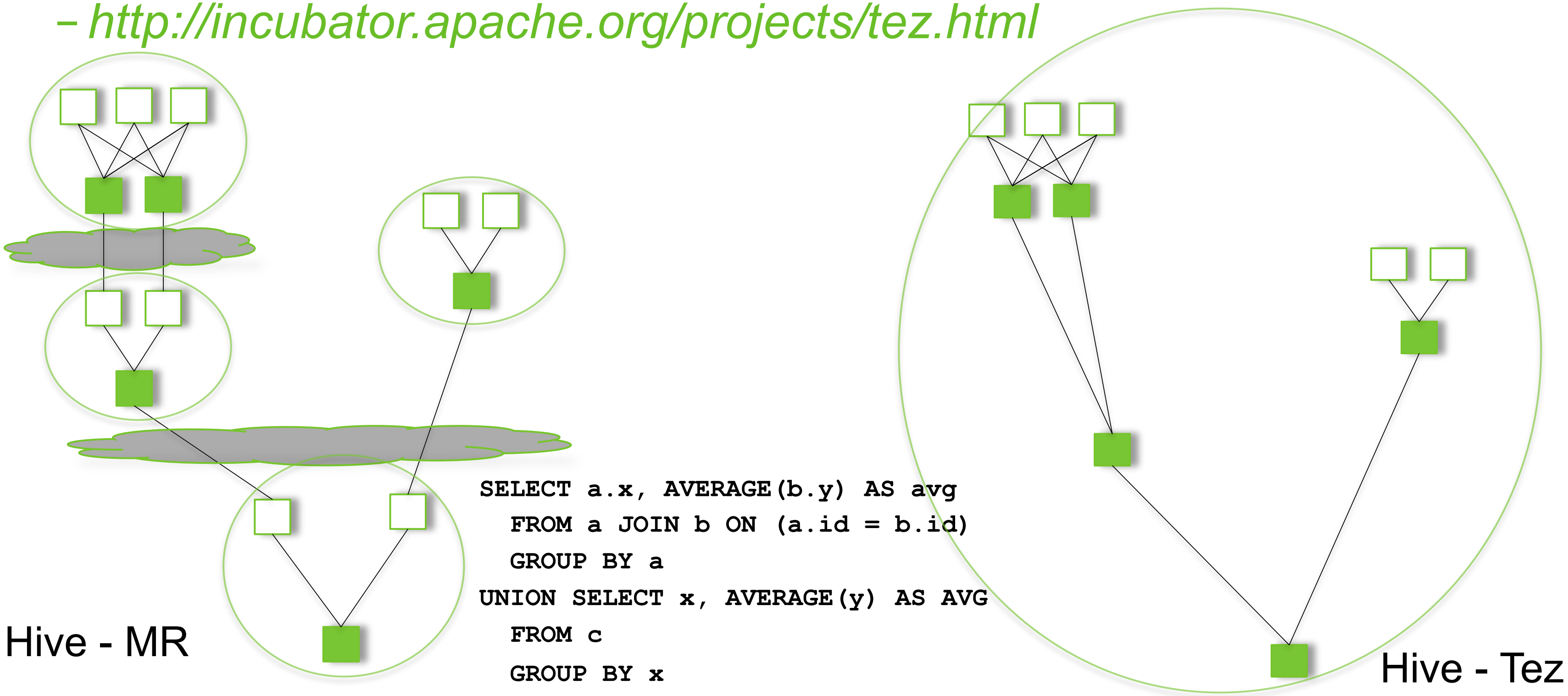
- **Original use-case**
- **Most complex application to build**
  - Data-locality
  - Fault tolerance
  - ApplicationMaster recovery: Check point to HDFS
  - Intra-application Priorities: Maps v/s Reduces
    - Needed complex feedback mechanism from ResourceManager
  - Security
  - Isolation
- **Binary compatible with Apache Hadoop 1.x**

Apache Hadoop MapReduce on YARN



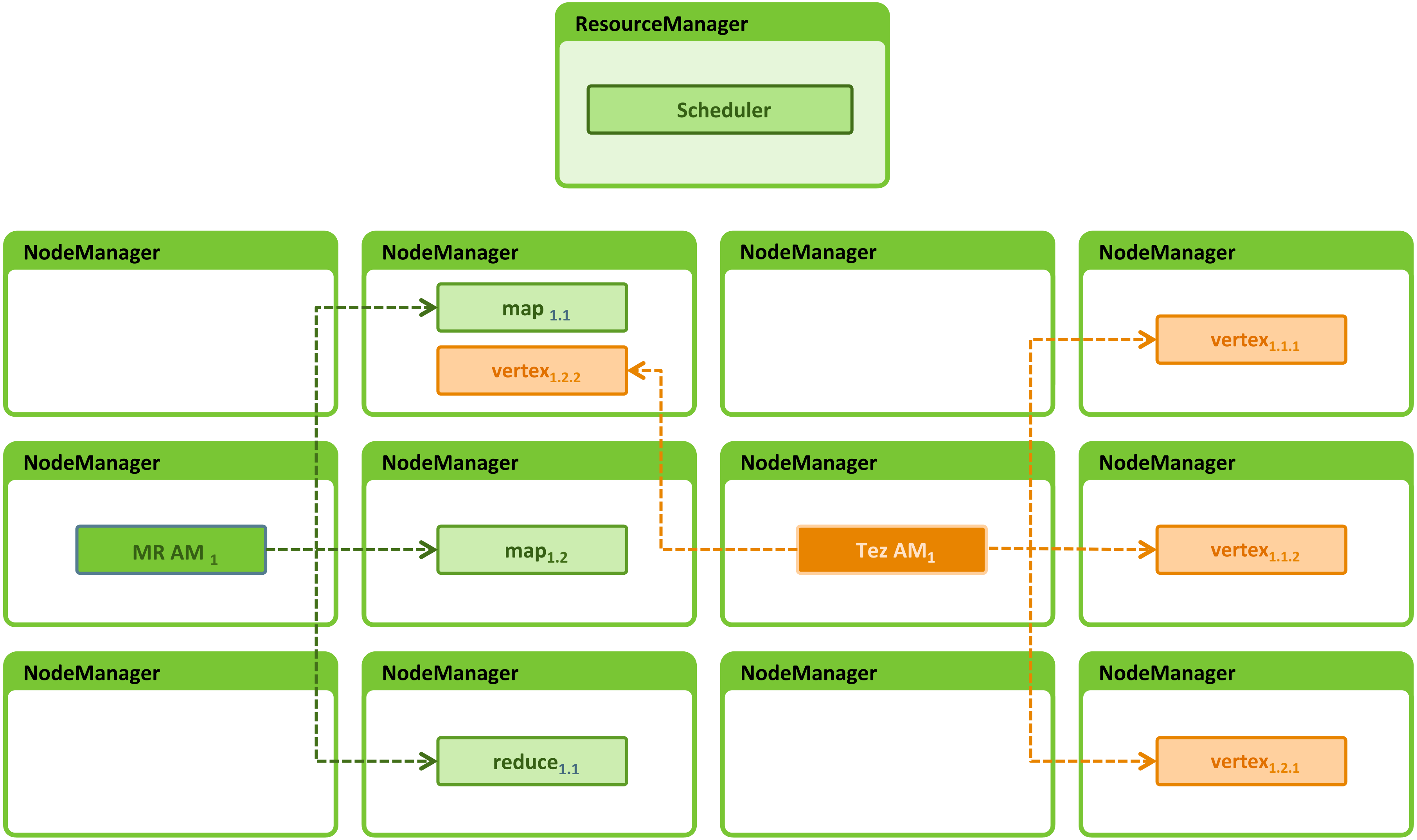
Apache Tez on YARN

- **Replaces MapReduce as primitive for Pig, Hive, Cascading etc.**
  - Smaller latency for interactive queries
  - Higher throughput for batch queries
  - *http://incubator.apache.org/projects/tez.html*



```
SELECT a.x, AVERAGE(b.y) AS avg
  FROM a JOIN b ON (a.id = b.id)
  GROUP BY a
UNION SELECT x, AVERAGE(y) AS AVG
  FROM c
  GROUP BY x
ORDER BY AVG;
```

Apache Tez on YARN

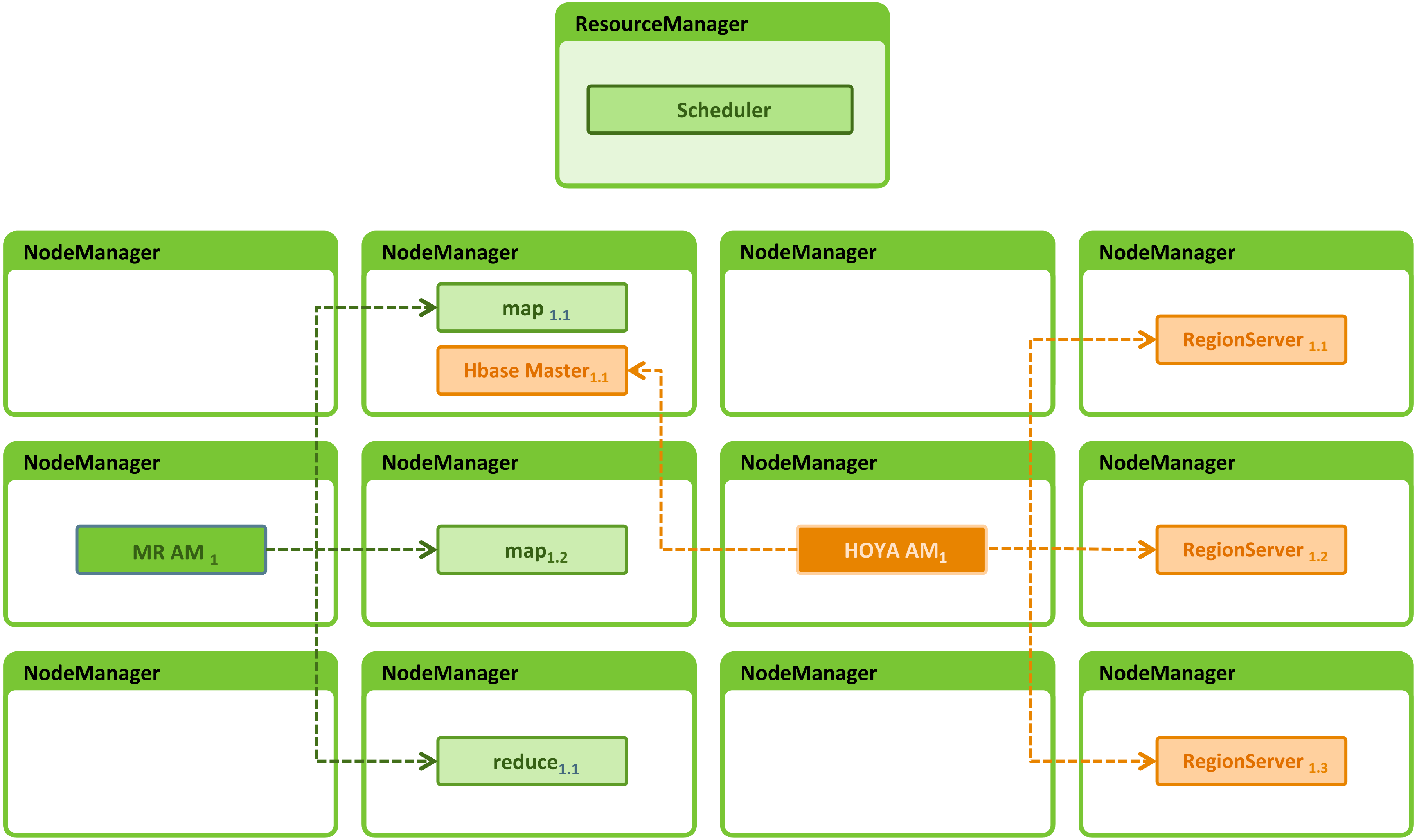


HOYA - Apache HBase on YARN

- **Hoya – Apache HBase becomes *user-level* application**
- **Use cases**
  - Small HBase cluster in large YARN cluster
  - Dynamic HBase clusters
  - Transient/intermittent clusters for workflows
- **APIs to create, start, stop & delete HBase clusters**
- **Flex cluster size: increase/decrease size with load**
- **Recover from Region Server loss with new container.**

Code: *https://github.com/hortonworks/hoya*

HOYA - Apache HBase on YARN



HOYA - Highlights

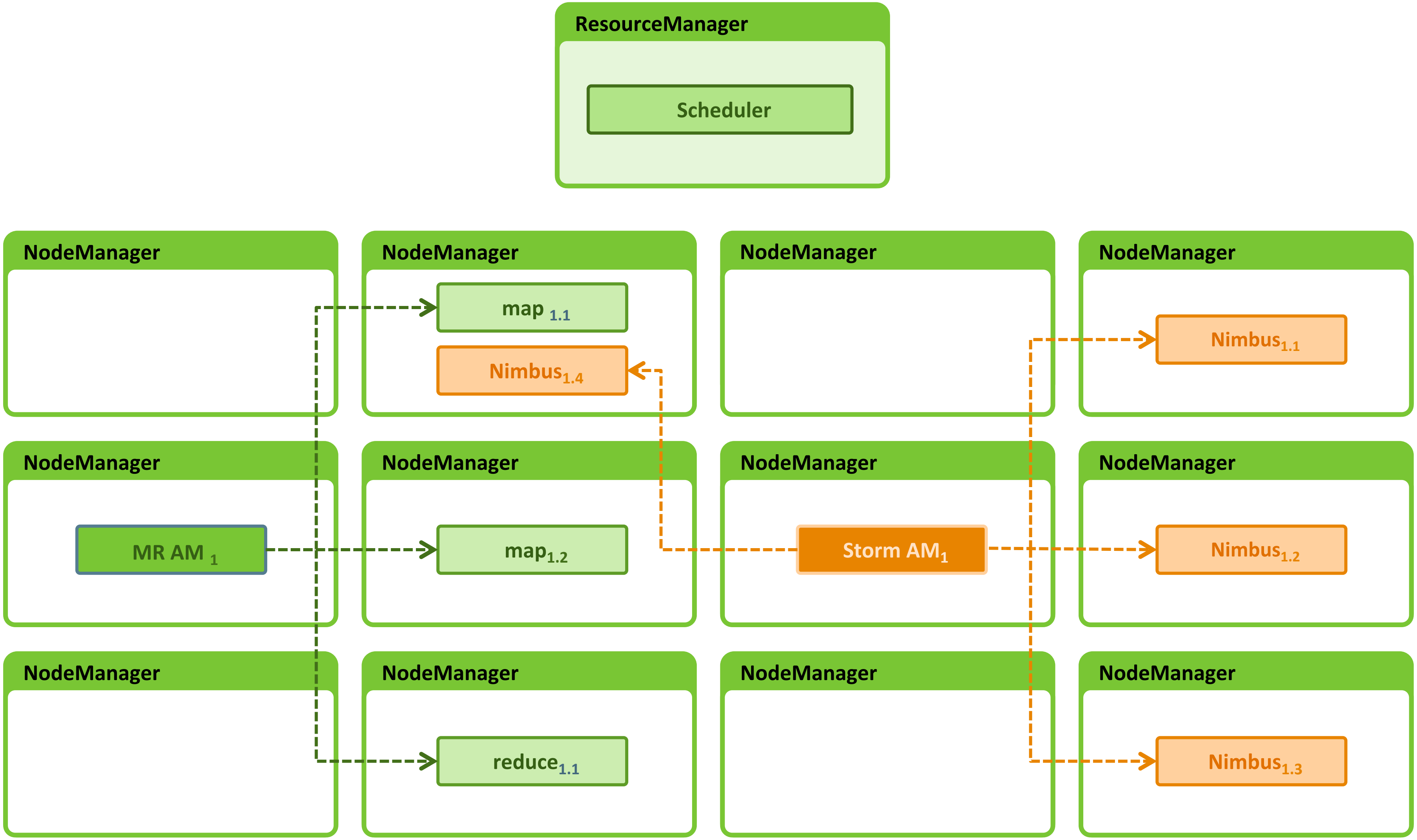
- **Cluster specification stored as JSON in HDFS**
- **Config directory cached - dynamically patched before pushing up as local resources for Master & RegionServers**
- **HBase tar file stored in HDFS -clusters can use the same/different HBase versions**
- **Handling of cluster flexing is the same code as unplanned container loss.**
- **No Hoya code on RegionServers: client and AM only**

Storm on YARN

- **Ability to deploy multiple Storm clusters on YARN for real-time event processing**
- **Yahoo – Primary contributor**
  - 200+ nodes in production
- **Ability to recover from faulty nodes**
  - Get new containers
- **Auto-scale for load balancing**
  - Get new containers as load increases
  - Release containers as load decreases

Code: *https://github.com/yahoo/storm-yarn*

Storm on YARN



General Architectural Considerations

- **Fault Tolerance**
  - Checkpoint
- **Security**
- **Always-On services**
- **Scheduler features**
  - Whitelist resources
  - Blacklist resources
  - Labels for machines
    - License management

Agenda

- Why YARN
- YARN Architecture and Concepts
- Building applications on YARN
- **Next Steps**

HDP 2.0 Community Preview & YARN Certification Program

Goal: Accelerate # of certified YARN-based solutions

- **HDP 2.0 Community Preview**
  - Contains latest community Beta of Apache Hadoop 2.0 & YARN
  - Delivered as easy to use Sandbox VM, as well as RPMs and Tarballs
  - Enables YARN Cert Program

    Community & commercial ecosystem to test and certify new and existing YARN-based apps

- **YARN Certification Program**
  - More than 14 partners in program at launch
    - *Splunk**
    - *Elastic Search**
    - *Altiscale**
    - *Concurrent**
    - *Microsoft*
    - *Platfora*
    - *Tableau*
    - *(IBM) DataStage*
    - *Informatica*
    - *Karmasphere*
    - *and others*

* Already certified

Forum & Office Hours

- **YARN Forum**
  - Community of Hadoop YARN developers
  - Focused on collaboration and Q&A
  - *http://hortonworks.com/community/forums/forum/yarn*

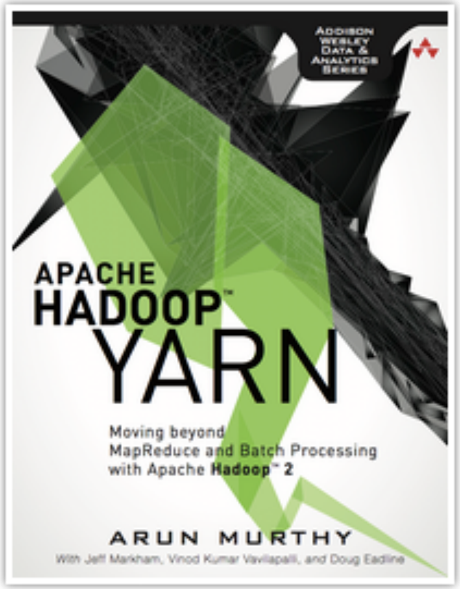
- **Office Hours**
  - *http://www.meetup.com/HSquared-Hadoop-Hortonworks-User-Group/*
  - Bi-weekly office hours with Hortonworks engineers & architects
  - Every other Thursday, starting on Aug 15th from 4:30 – 5:30 PM (PST)
  - At Hortonworks Palo Alto HQ
    - West Bayshore Rd. Palo Alto, CA 94303 USA

Technical Resources

- **ASF hadoop-2.1.0-beta**
  - Coming soon!
  - http://hadoop.apache.org/common/releases.html
  - Release Documentation: http://hadoop.apache.org/common/docs/r2.1.0-beta

- **Blogs:**
  - http://hortonworks.com/hadoop/yarn
  - http://hortonworks.com/blog/category/apache-hadoop/yarn/
  - http://hortonworks.com/blog/introducing-apache-hadoop-yarn/
  - http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/

What Next?


Addison Wesley Data & Analytics Series
APACHE HADOOP™ YARN
Moving beyond MapReduce and Batch Processing with Apache Hadoop™ 2
ARUN MURTHY
With Jeff Markham, Vinod Kumar Vavilapalli, and Doug Eadline

- **Download the Book**
- **Download the Community preview**
- **Join the Beta Program**
- **Participate via the YARN forum**
- **Come see us during the *YARN Office Hours***
- **Follow us… @hortonworks**

Thank You!

http://hortonworks.com/hadoop/yarn