Yarn: Yet another resource negotiator

Presented by: Devin Petersohn

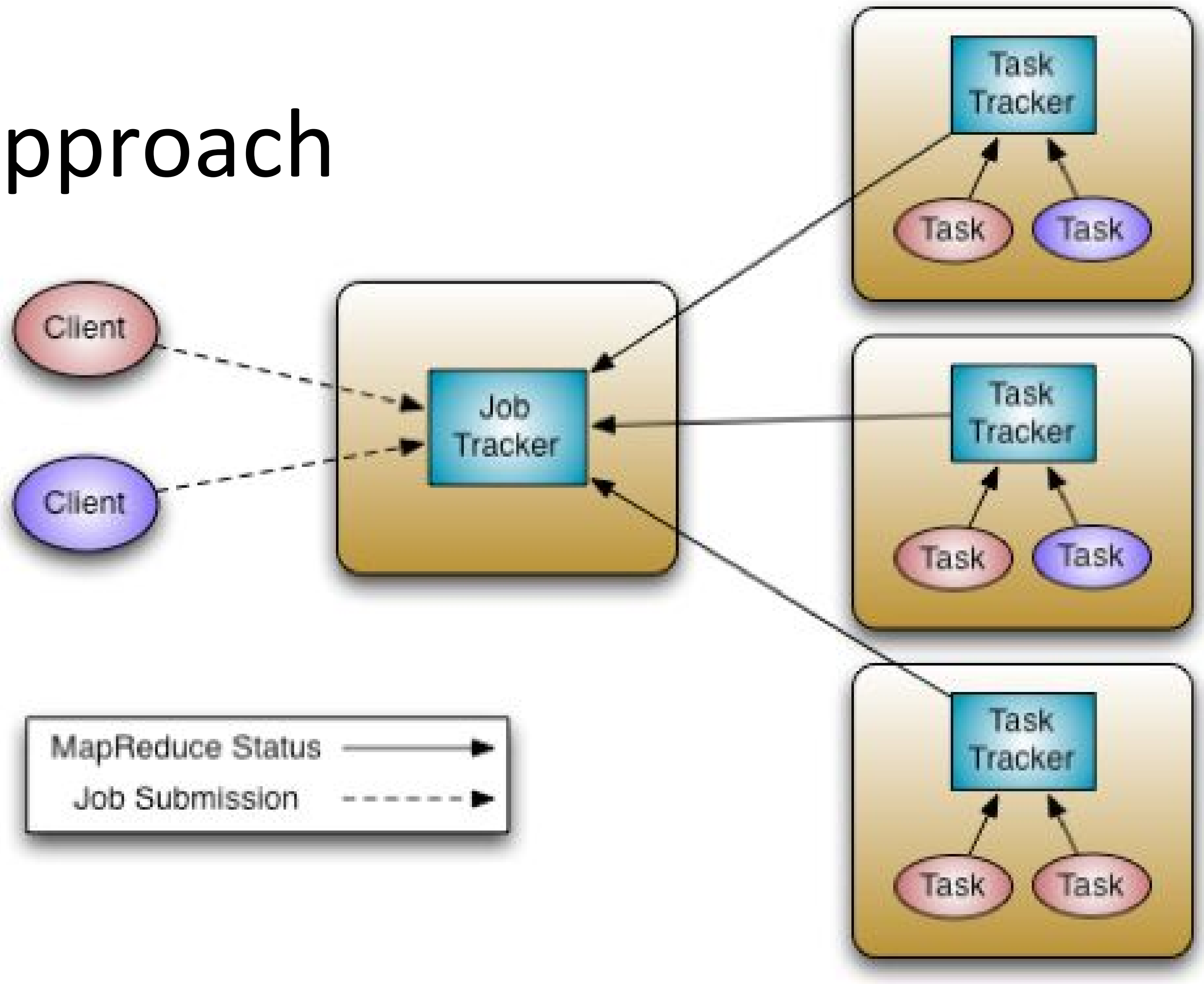
Introduction

- Hadoop MapReduce was highly adopted
  - Even for non-MapReduce jobs (Map-only jobs)
- Several problems arose:
  - MapReduce is very limiting in its capabilities
  - Hadoop’s Resource Manager was made for MapReduce jobs

- We will be focusing on the limitations of the Resource Manager in this discussion

Hadoop’s (v1.x) Approach

- Job tracker:
  - Resource tracking
    - Consumption/avail
  - Resource mgmt
  - Job life-cycle mgmt
- Task tracker:
  - Launch/teardown jobs
  - Task status info



What was wrong with HoD?

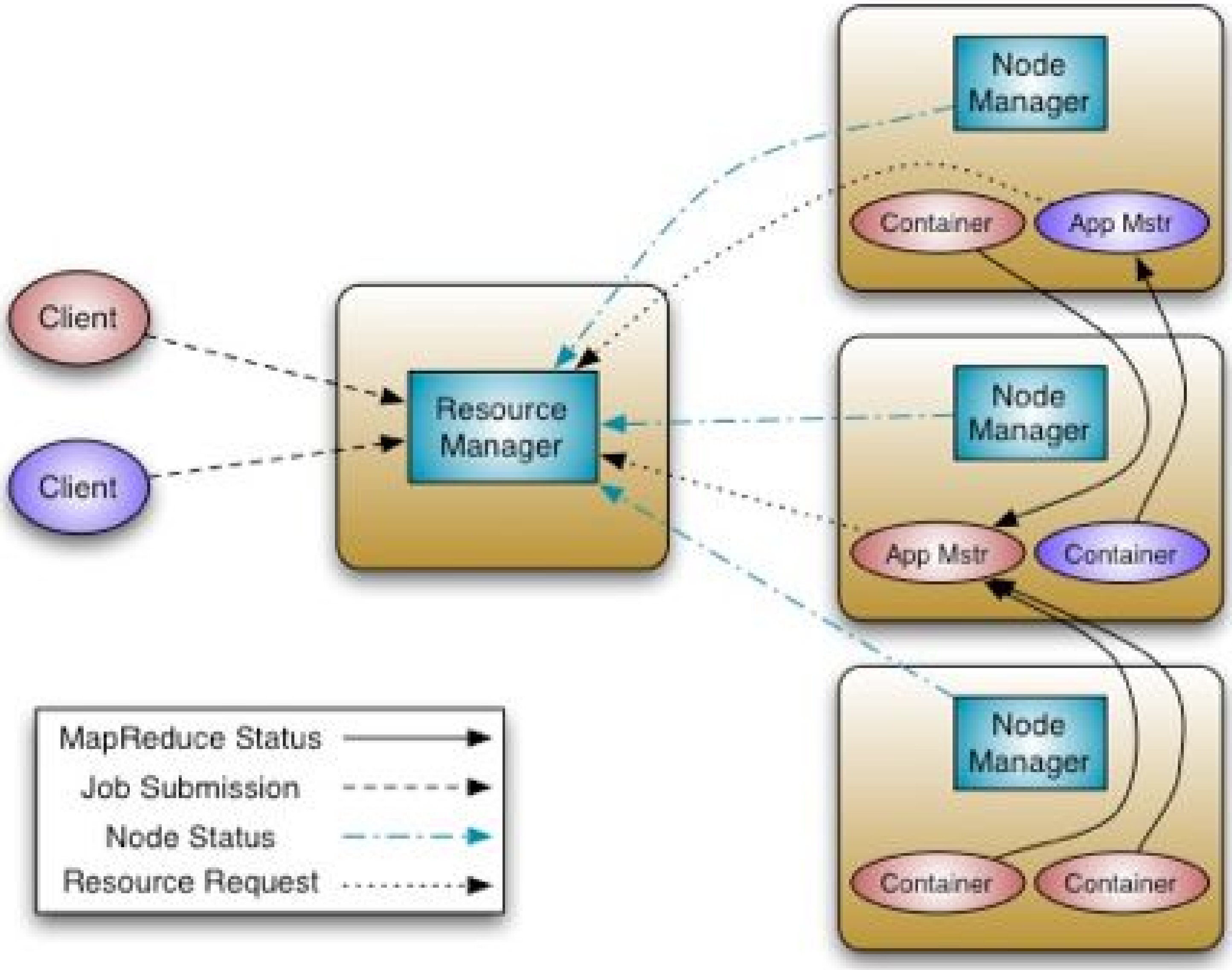
- Utilization: JobTracker viewed the cluster as nodes with distinct map slots and reduce slots
  - Slots are not fungible
  - Eg. map slots may be ”full” but reduce tasks could be empty
  - Single reduce task can prevent an entire cluster from being reclaimed
- Hadoop MapReduce workloads only
  - Support for “slightly” older versions of Hadoop
- Downtime of JobTracker
  - Loss of all running jobs
  - Users must manually recover workflows

Requirements for YARN

1. Scalability
2. Multi-tenancy
3. Serviceability
4. Locality awareness
5. High Cluster Utilization
6. Reliability/Availability
7. Secure and auditable operation
8. Support for Programming Model Diversity
9. Flexible Resource Model
10. Backward compatibility

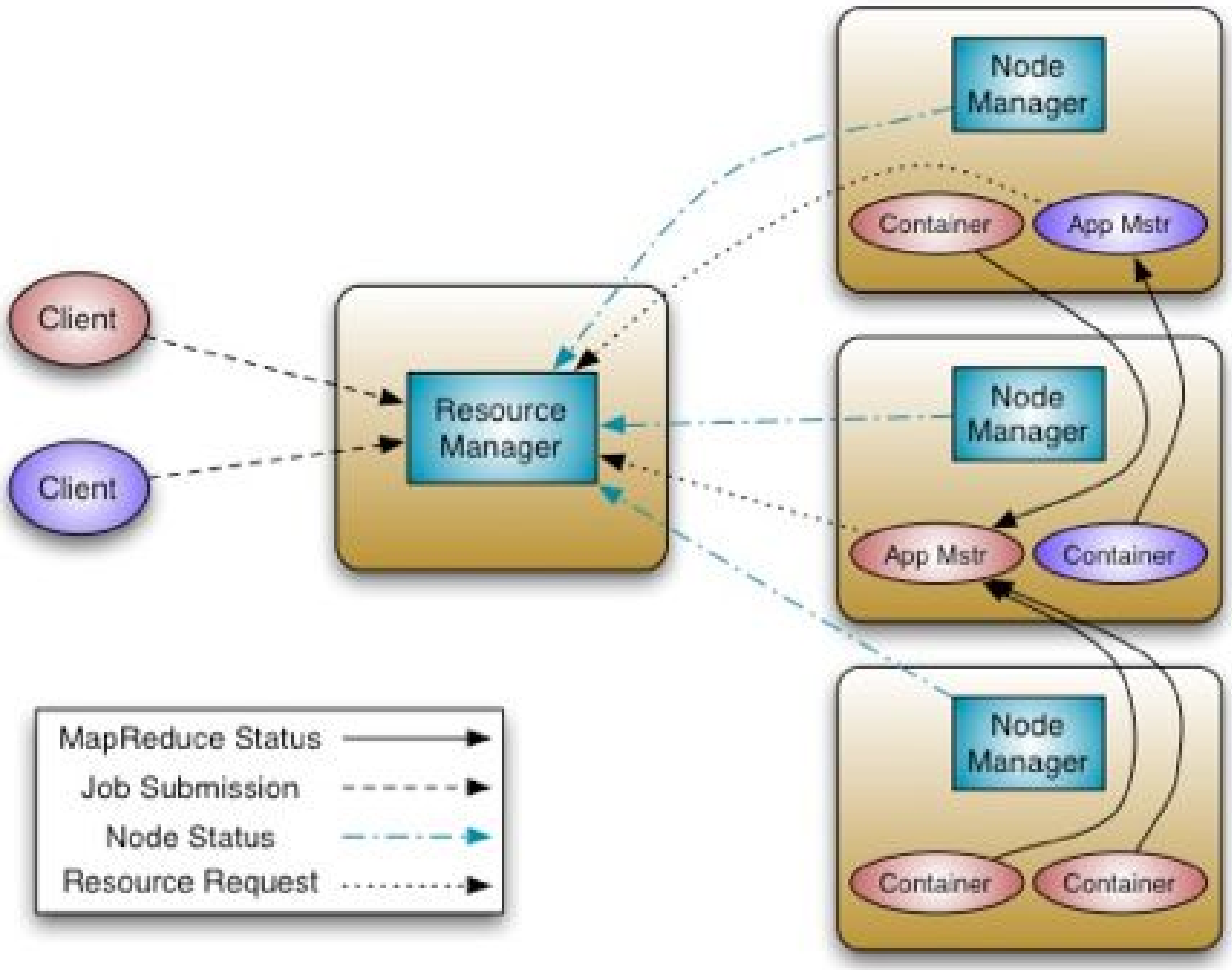
YARN's Approach

- Resource Manager (RM)
  - Arbitrates resources
  - Publicly Interfaces with:
    - Clients
    - Application Masters
  - Internally Interfaces with:
    - Node Managers
  - Pluggable Scheduler
    - No app monitoring/tracking
    - Based on resource reqmnt
    - Uses containers



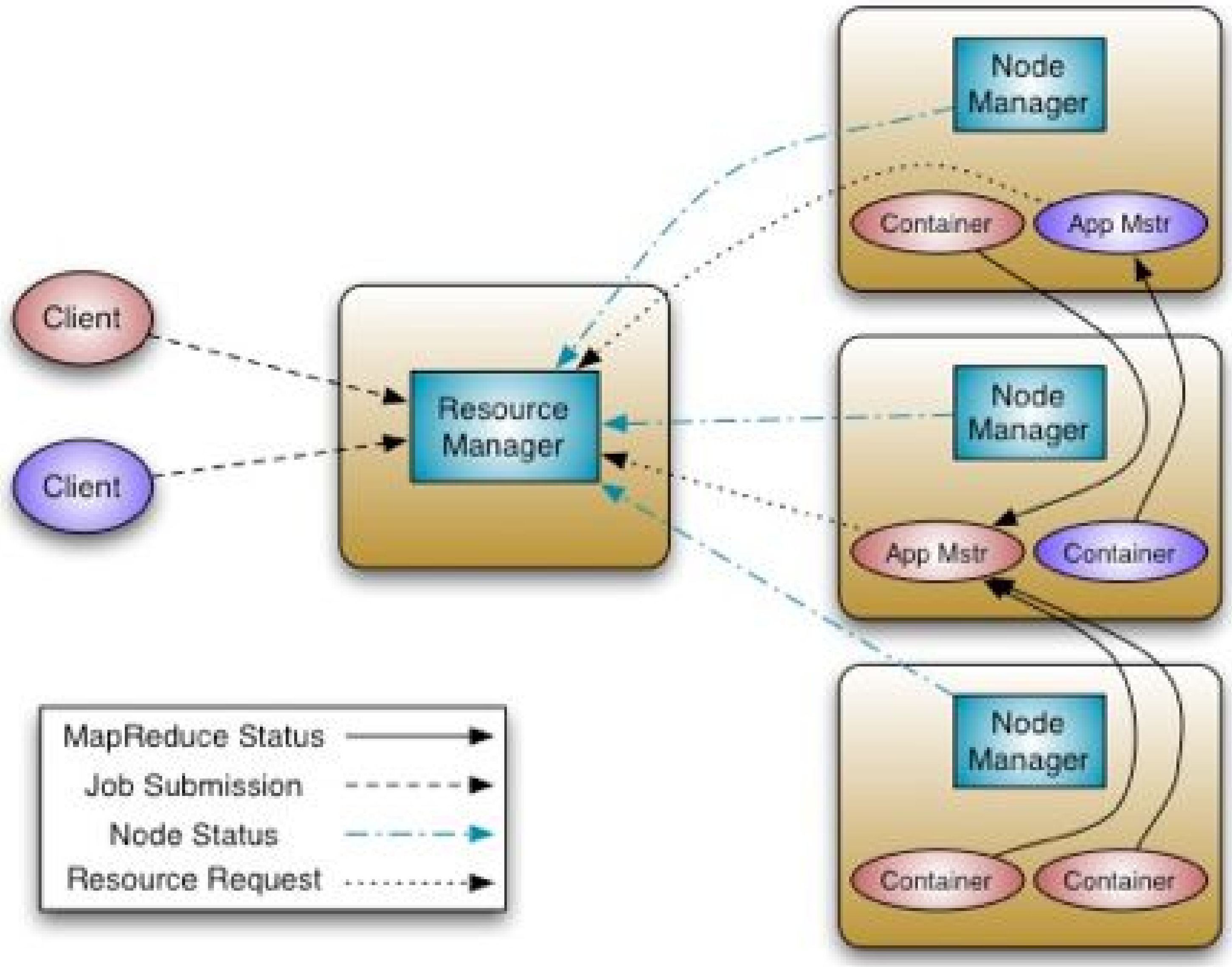
YARN's Approach

- Node Manager (NM)
  - Per-machine slave
  - Monitor resources of machine
  - Heartbeat based communication
  - Manages containers
  - Pluggable auxiliary services



YARN’s Approach

- Application Master (AM)
  - Negotiates resource containers
  - App monitoring/tracking
  - Runs as a container
- Container
  - Unit of allocation
  - Similar to MapReduce slots



YARN Execution sequence

AM – ApplicationMaster
NM – NodeManager
RM – ResourceManager

1. Client program submits application
2. RM allocates container to start the AM
3. AM registers with RM on startup
4. AM negotiates with RM for resource containers
5. AM contacts NM to launch container
6. Application code is executed within the container, AM is supplied execution status
7. During execution, client communicates directly with AM or RM to get status/progress updates
8. On application exit, AM unregisters with RM and exit cleanly

Fault tolerance

- Fault tolerance is a shared responsibility
  - ResourceManager and ApplicationMaster
- ResourceManager – the single point of failure
  - After recovery, all ApplicationMasters are killed
  - Users’ pipelines can be restored (for some frameworks)
    - Running tasks/tasks that completed during recovery are killed/re-run
- Container failure recovery is left to the frameworks