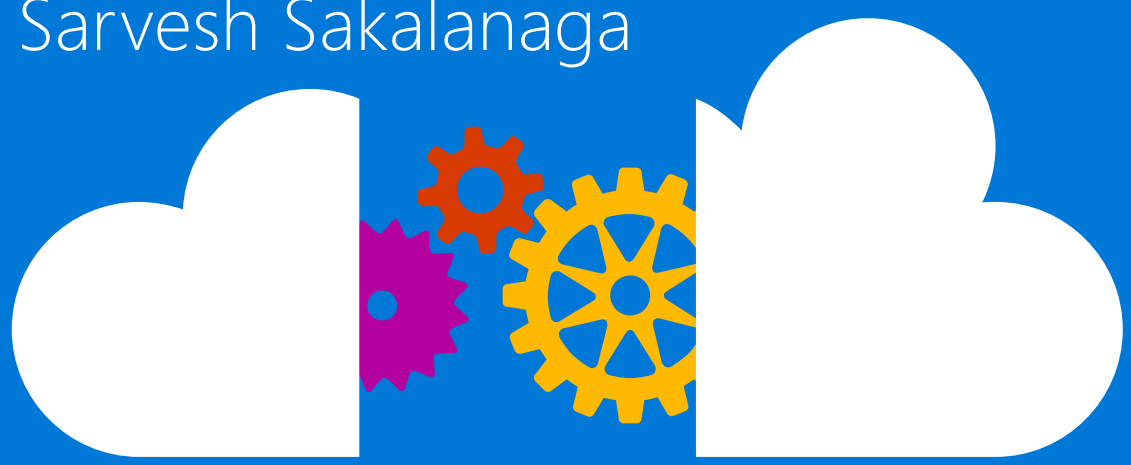
Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters

Konstantinos Karanasos, Sriram Rao, Carlo Curino,
Chris Douglas, Kishore Chaliparambil, Giovanni Furamola,
Solom Heddaya, Raghu Ramakrishnan, Sarvesh Sakalanaga



USENIX ATC

Santa Clara, July 10, 2015

Microsoft

Cloud and Information Services Lab

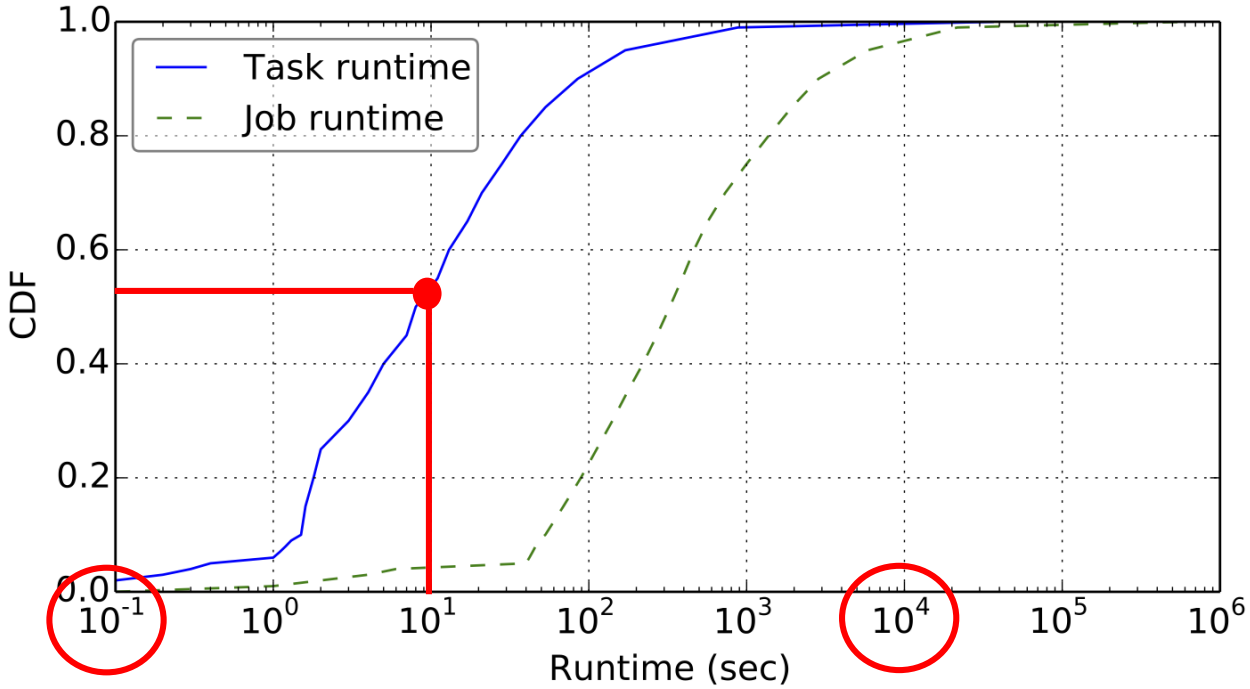
Workload Heterogeneity in Shared Clusters

- Cluster operators want to maximize cluster ROI

  Consolidate workloads

- Wide variety of workloads

  Long-running services, production SLA jobs, ad-hoc (best-effort) jobs



Workload heterogeneity in Cosmos

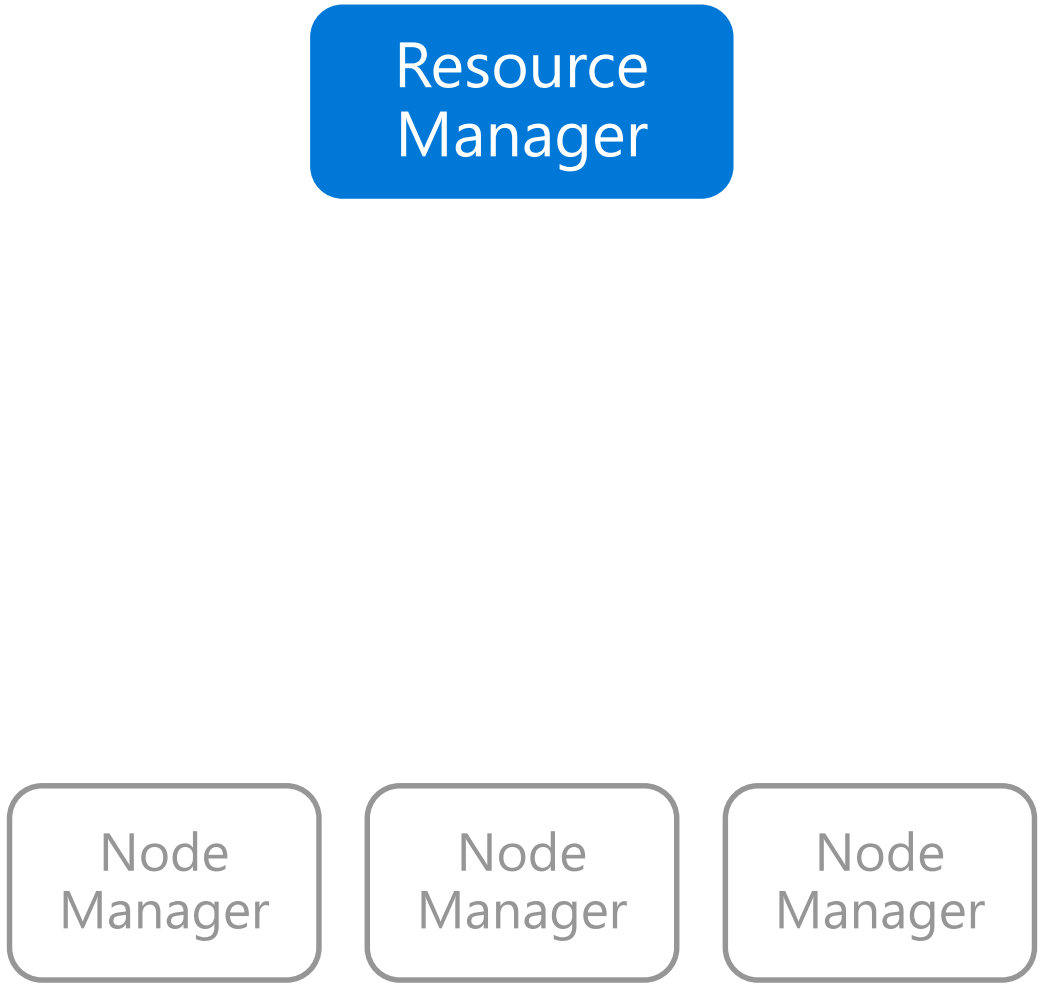
- Task runtime varies from sub-sec to 10,000+ sec
- 50% of tasks are shorter than 10 sec

Resource Management in Shared Clusters

- RM frameworks provide APIs to acquire resources and run jobs
  e.g., YARN, Apollo, Mesos, Omega, Borg
- Not easy to achieve high cluster utilization
- Either centralized *or* distributed approaches

Centralized Resource Management

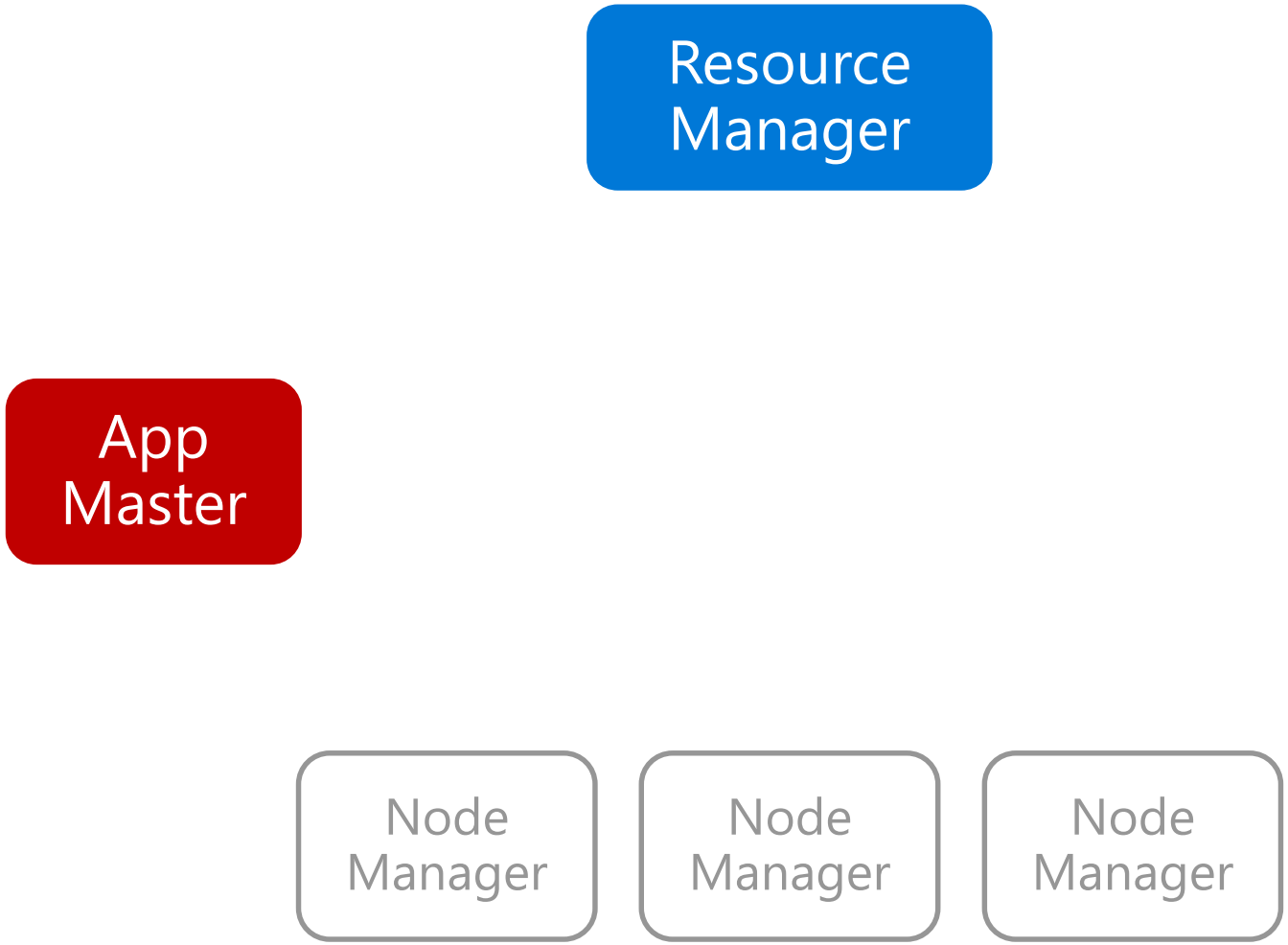
[YARN, Mesos, Omega, Borg]



- All scheduling decisions go through the central RM
- The RM resolves all conflicts and guarantees resources to applications

Centralized Resource Management

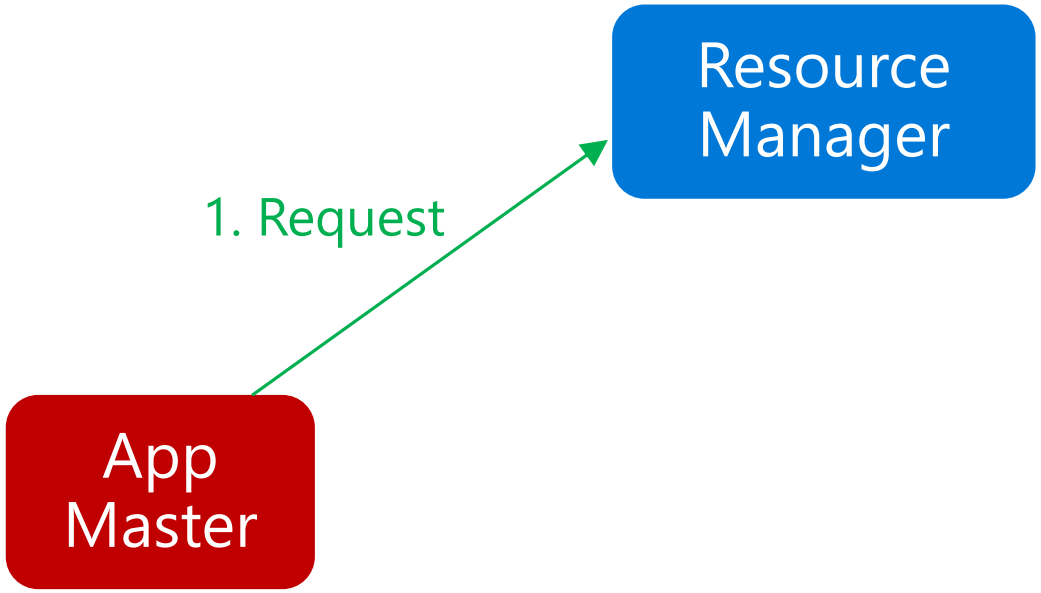
[YARN, Mesos, Omega, Borg]



- All scheduling decisions go through the central RM
- The RM resolves all conflicts and guarantees resources to applications

Centralized Resource Management

[YARN, Mesos, Omega, Borg]

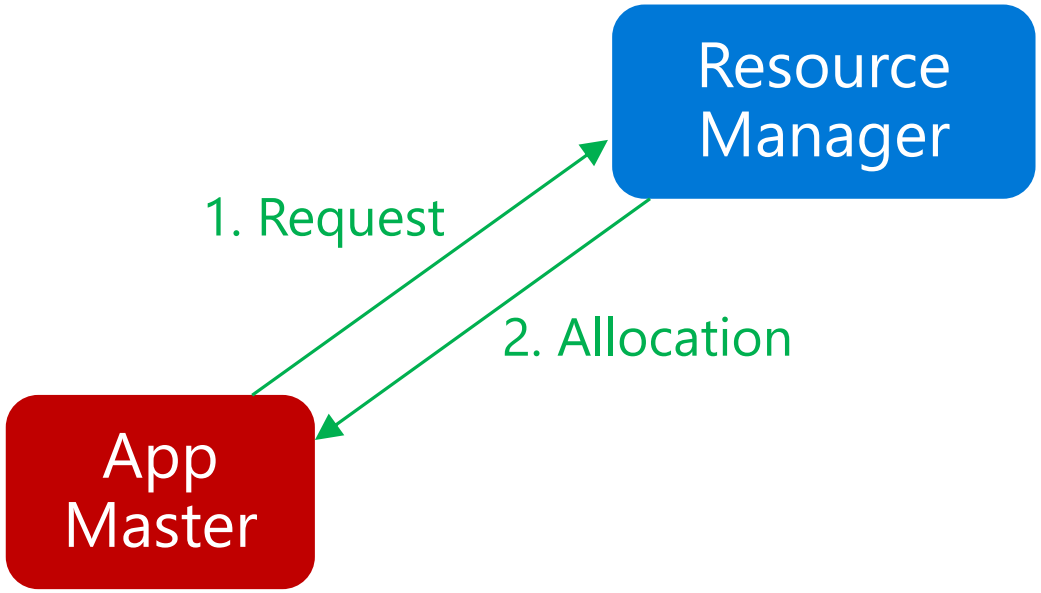


- All scheduling decisions go through the central RM
- The RM resolves all conflicts and guarantees resources to applications



Centralized Resource Management

[YARN, Mesos, Omega, Borg]

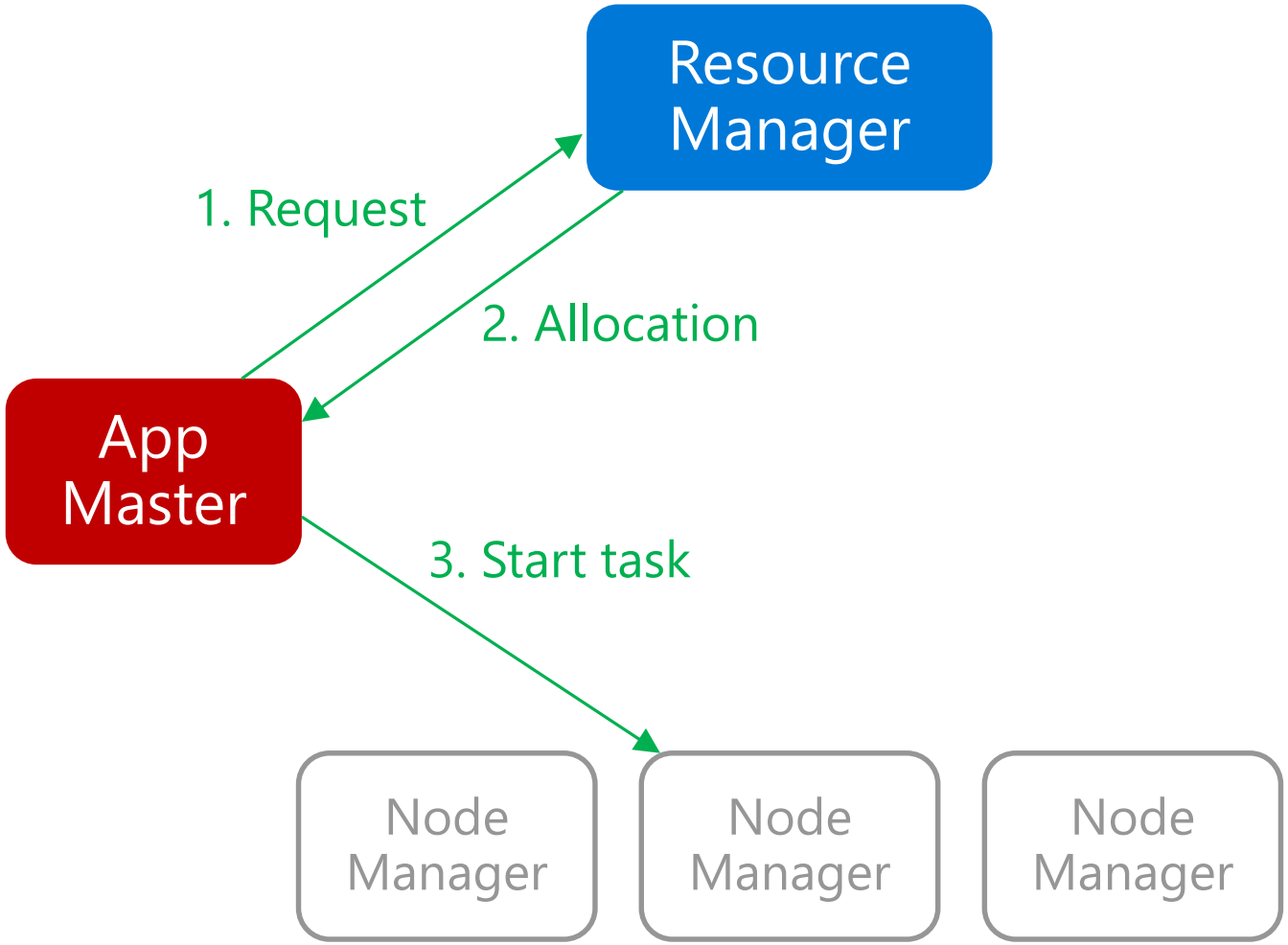


- All scheduling decisions go through the central RM
- The RM resolves all conflicts and guarantees resources to applications



Centralized Resource Management

[YARN, Mesos, Omega, Borg]

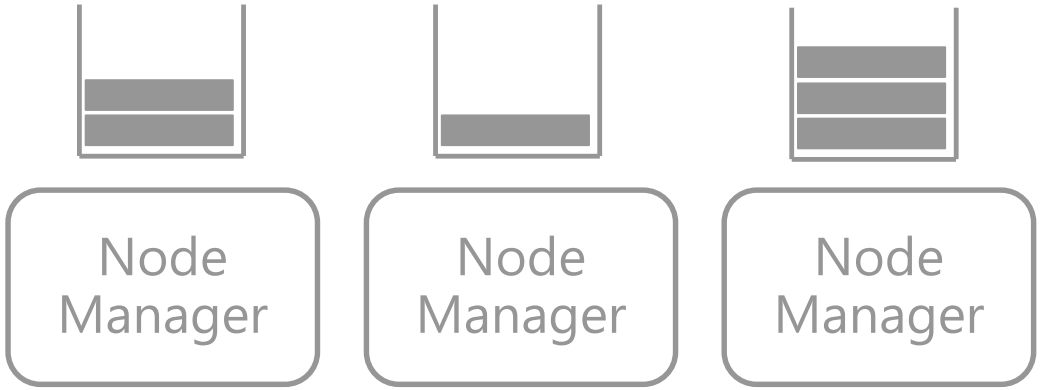


- All scheduling decisions go through the central RM
- The RM resolves all conflicts and guarantees resources to applications

Distributed Resource Management

[Apollo, Sparrow]

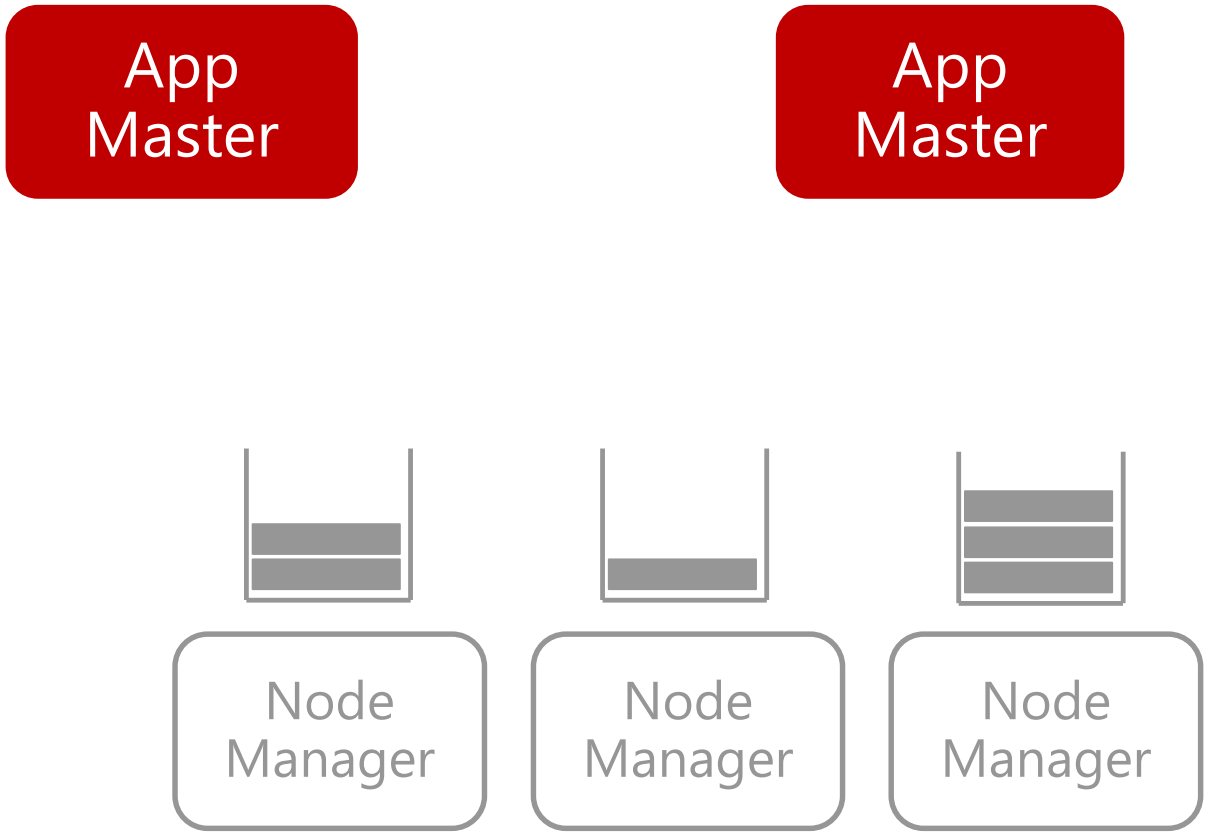
- AMs queue tasks directly to NMs
- Loose coordination through the Resource Monitor



Distributed Resource Management

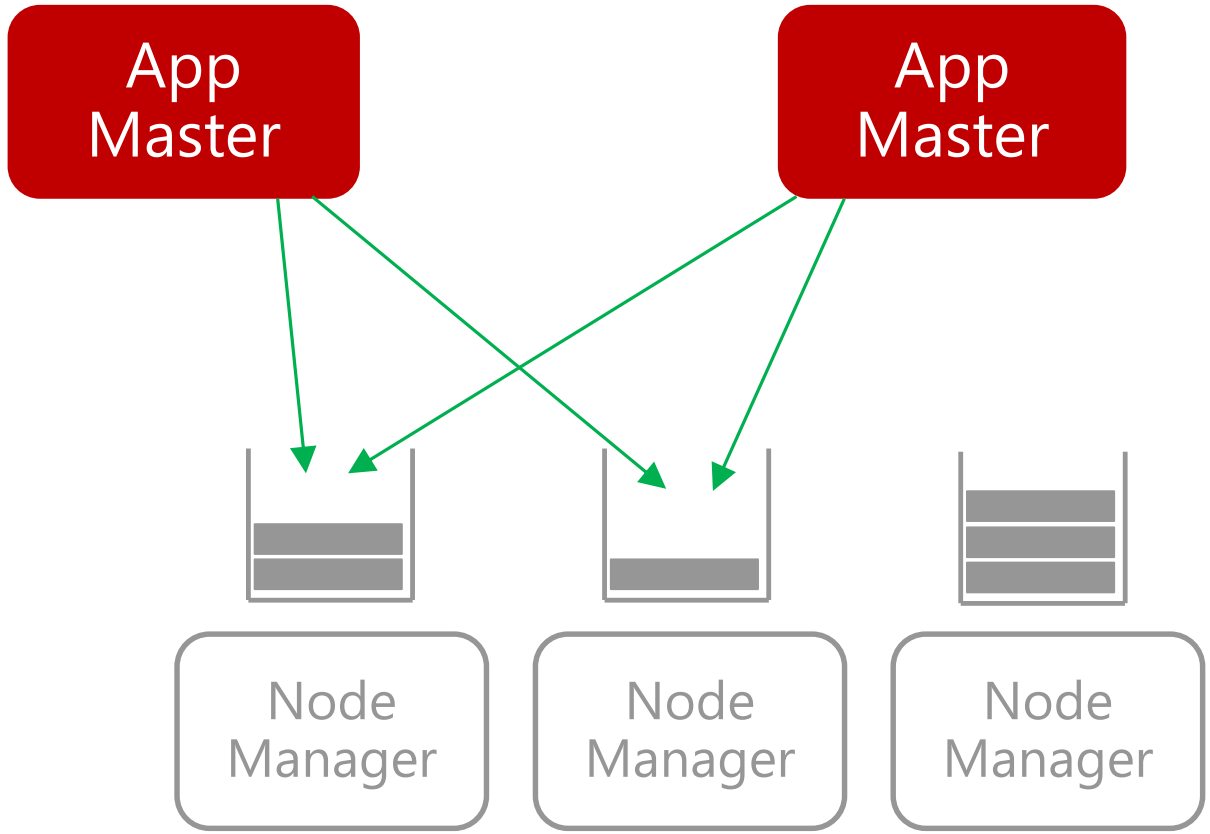
[Apollo, Sparrow]

- AMs queue tasks directly to NMs
- Loose coordination through the Resource Monitor



Distributed Resource Management

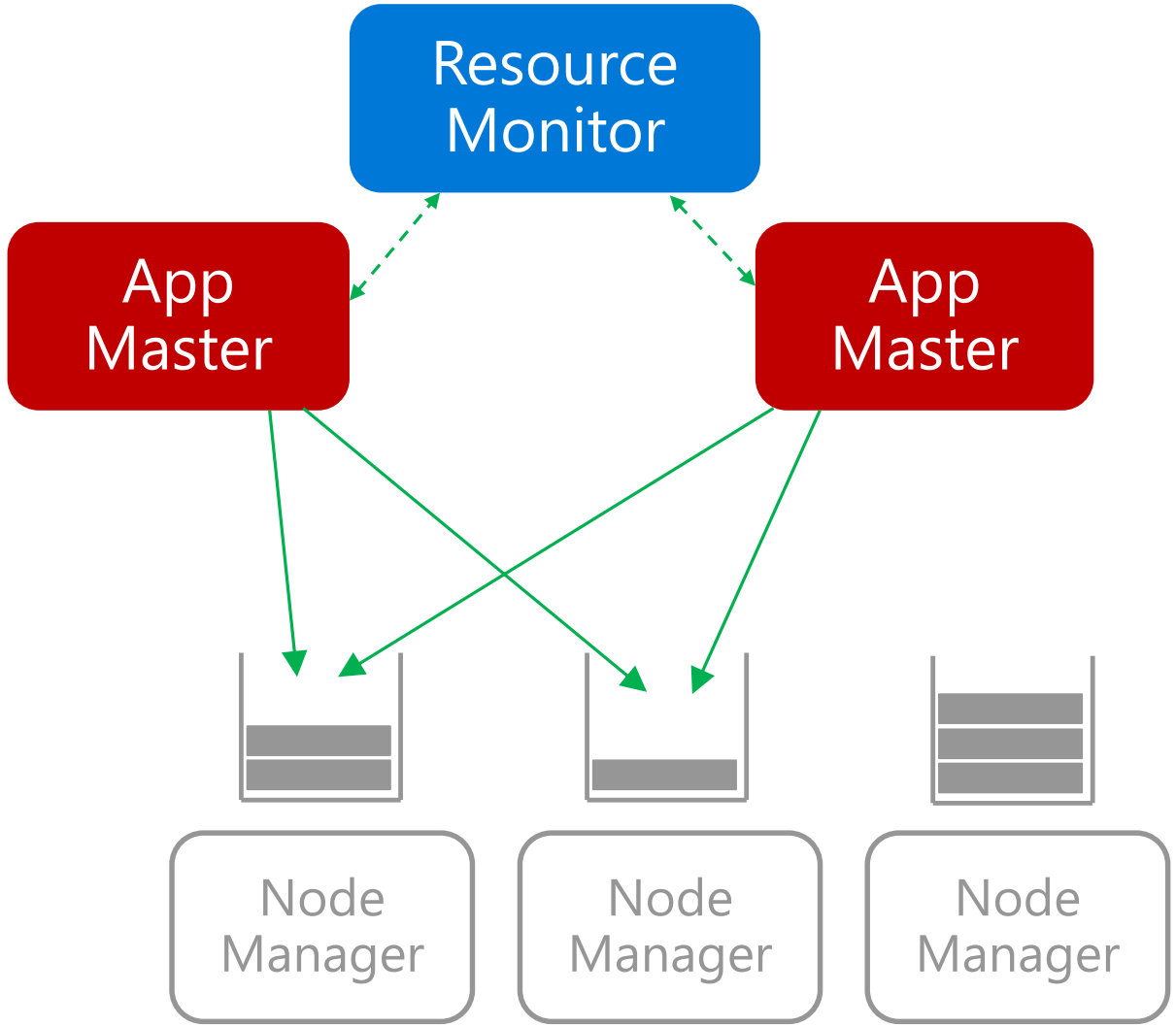
[Apollo, Sparrow]



- AMs queue tasks directly to NMs
- Loose coordination through the Resource Monitor

Distributed Resource Management

[Apollo, Sparrow]



- AMs queue tasks directly to NMs
- Loose coordination through the Resource Monitor

Centralized vs. Distributed Scheduling

	Centralized	Distributed
Workload heterogeneity	✓	
Task placement	✓	
Enforcing scheduling invariants	✓	
Allocation latency		✓
Slot utilization		✓
Scalability		✓

Mercury: Key Insight

- Centralized and distributed schedulers are complementary
- Hybrid approach
- "Trade performance guarantees for allocation latency"
- Applications can choose among scheduling types

  Based on job type, job characteristics, cluster load, etc.

  Mercury provides a programmatic way to use otherwise idle resources

Mercury: Key Insight

- Centralized and distributed schedulers are complementary
- Hybrid approach
- "Trade performance guarantees for allocation latency"
- Applications can choose among scheduling types

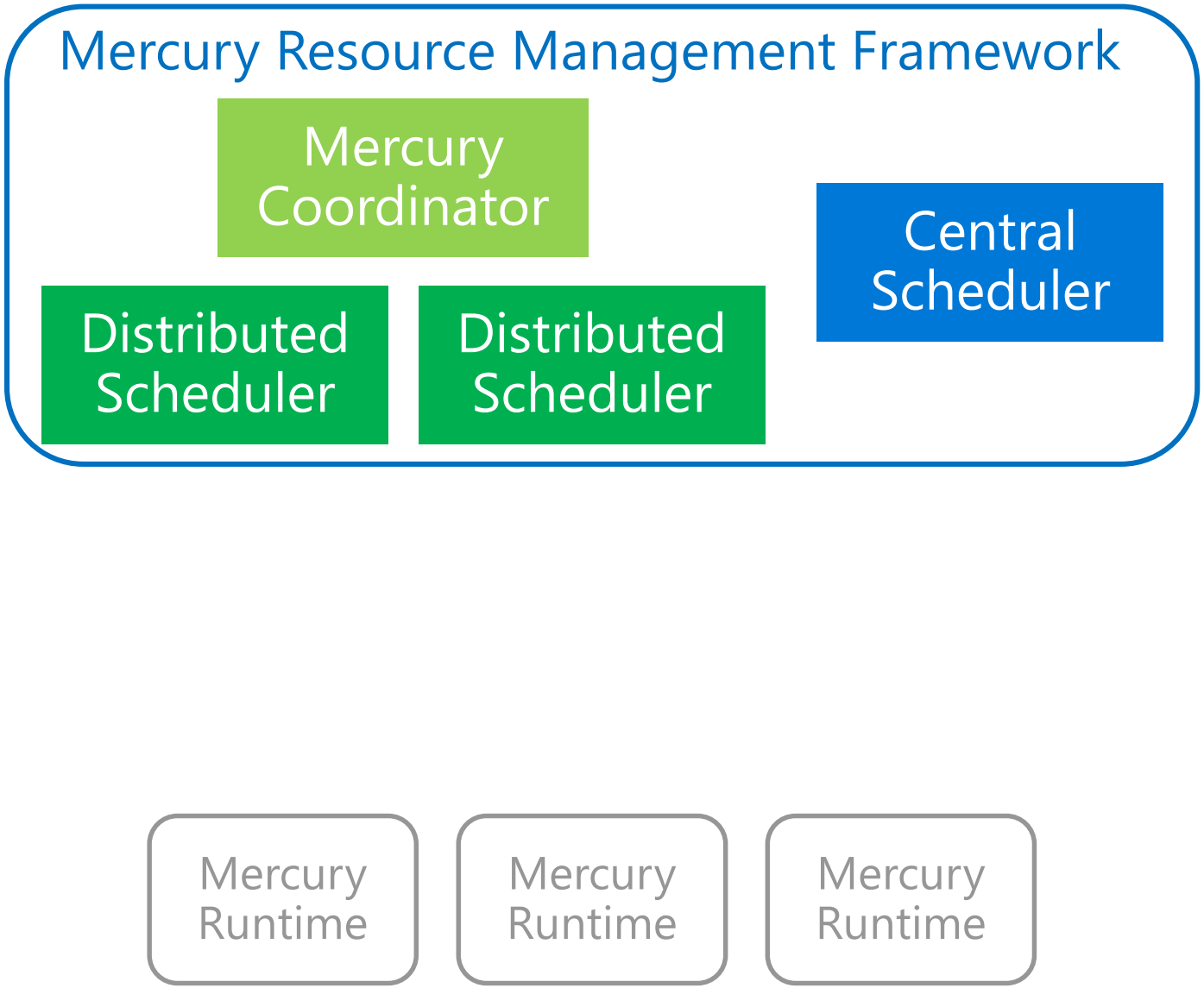
  Based on job type, job characteristics, cluster load, etc.

  Mercury provides a programmatic way to use otherwise idle resources

Gains over YARN:
Up to 40% task throughput
Up to 66% mean job latency

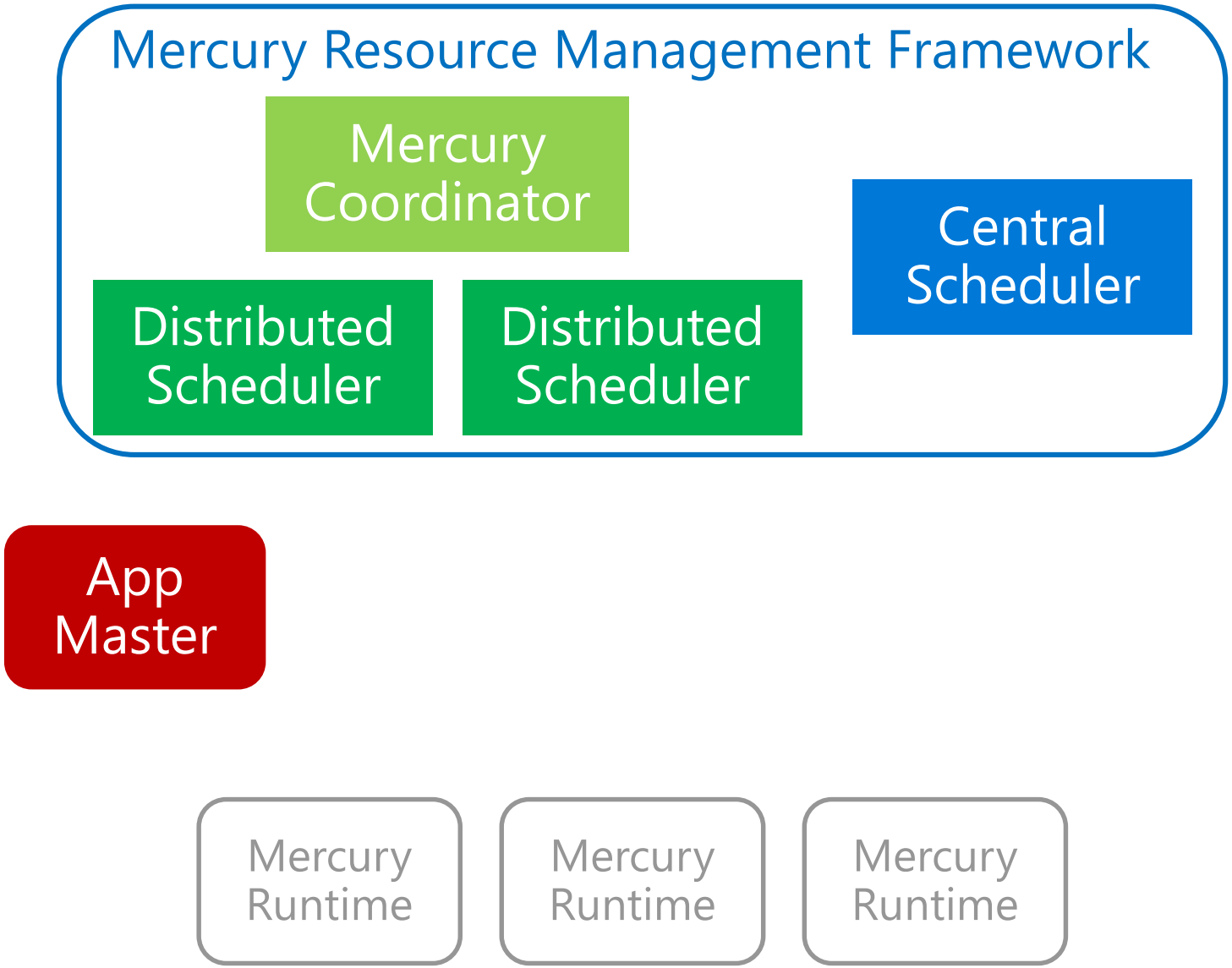
Mercury Design and Implementation

Mercury Architecture (Conceptual)



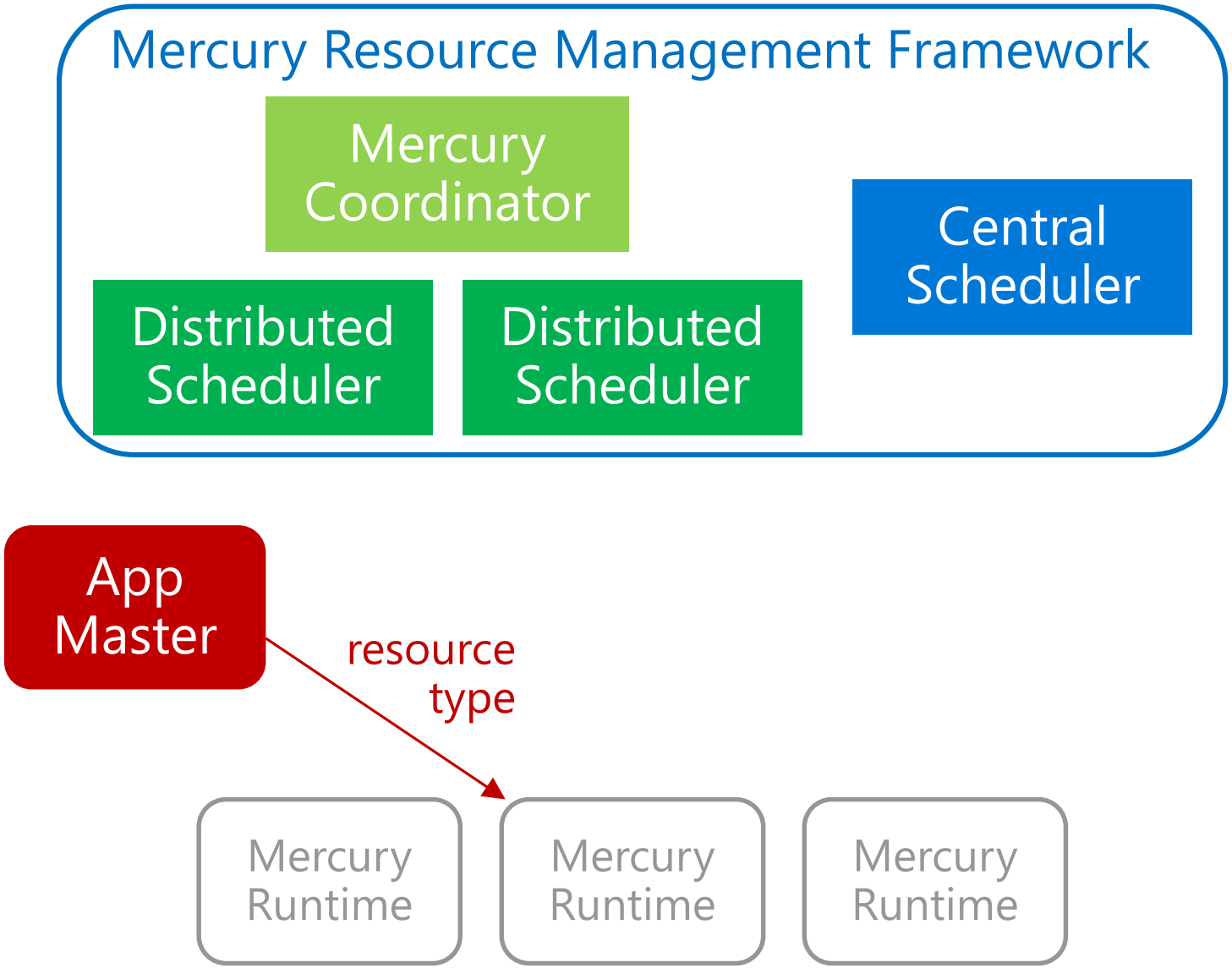
- Two types of schedulers
- Central scheduler
  Scheduling policies/guarantees
  Slow(er) decisions
- Distributed schedulers
  Fast/low-latency decisions
- AM specifies resource *type*

Mercury Architecture (Conceptual)



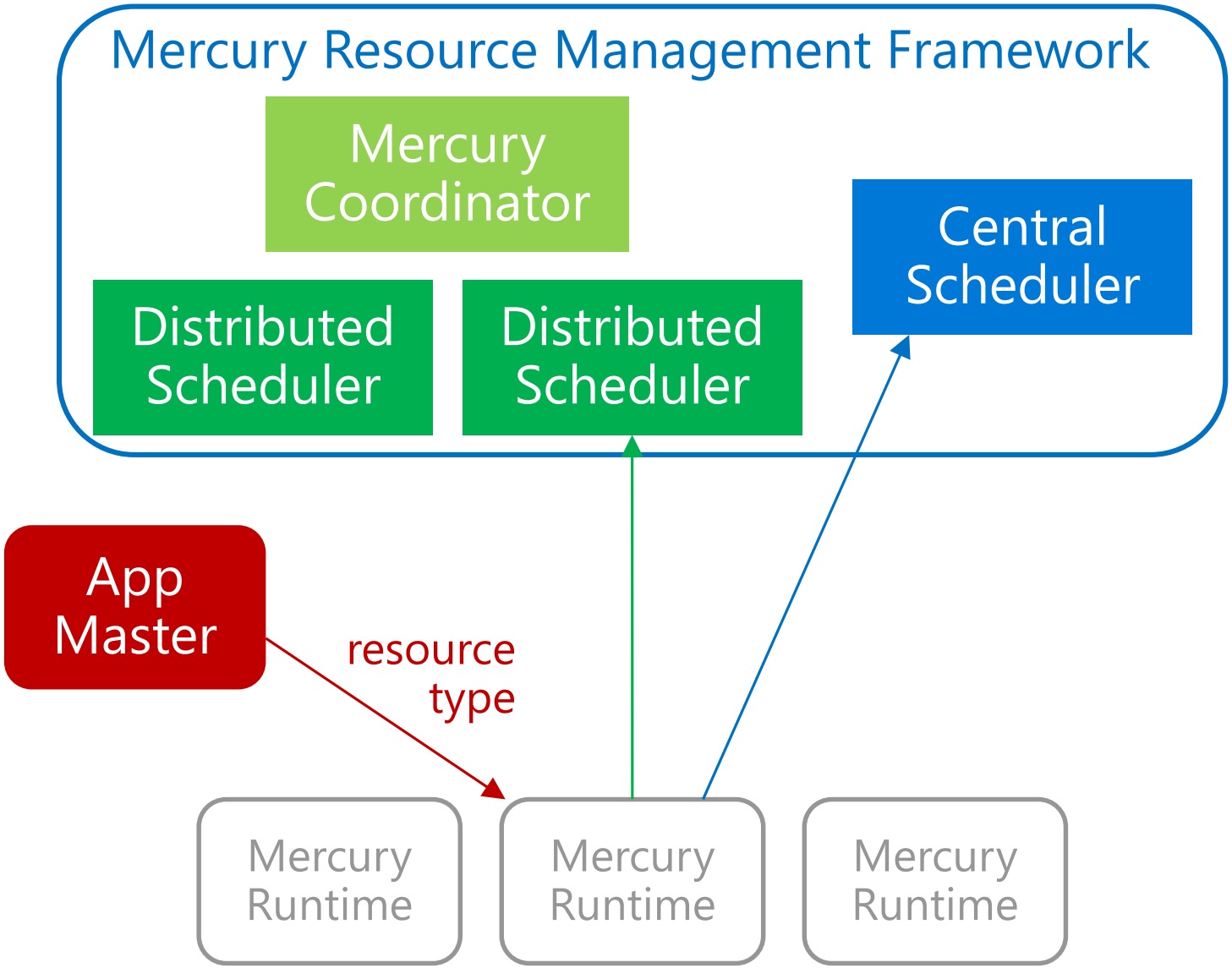
- Two types of schedulers
- Central scheduler
  Scheduling policies/guarantees
  Slow(er) decisions
- Distributed schedulers
  Fast/low-latency decisions
- AM specifies resource *type*

Mercury Architecture (Conceptual)



- Two types of schedulers
- Central scheduler
  Scheduling policies/guarantees
  Slow(er) decisions
- Distributed schedulers
  Fast/low-latency decisions
- AM specifies resource *type*

Mercury Architecture (Conceptual)



- Two types of schedulers
- Central scheduler
  Scheduling policies/guarantees
  Slow(er) decisions
- Distributed schedulers
  Fast/low-latency decisions
- AM specifies resource *type*

Container Types

GUARANTEED containers

- Guaranteed to start
- Guaranteed to run to completion

QUEUEABLE containers

- Executed opportunistically
- Can be queued
- Not guaranteed to start
- Not guaranteed to run to completion

  Can be pre-empted/killed in case of resource contention

Container Types

GUARANTEED containers

- Guaranteed to start
- Guaranteed to run to completion

- Allocated by (slow) central scheduler

QUEUEABLE containers

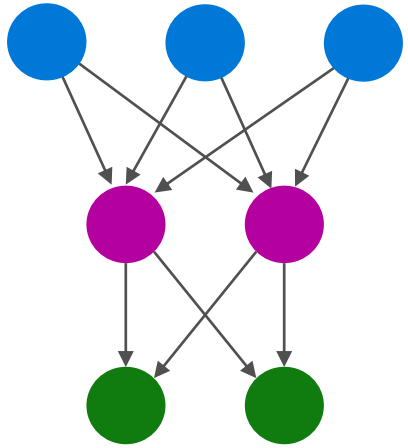
- Executed opportunistically
- Can be queued
- Not guaranteed to start
- Not guaranteed to run to completion

  Can be pre-empted/killed in case of resource contention

- Allocated by (fast) distributed scheduler

Use of Container Types in DAGs: Examples

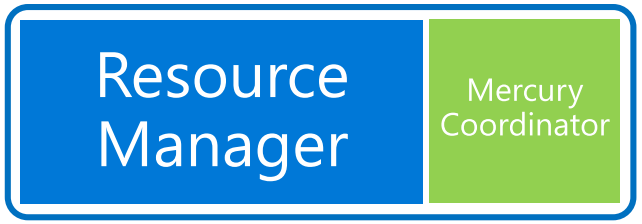
- Example 1 (based on priority of jobs)
  GUARANTEED containers for critical SLA-jobs
  QUEUEABLE for background jobs

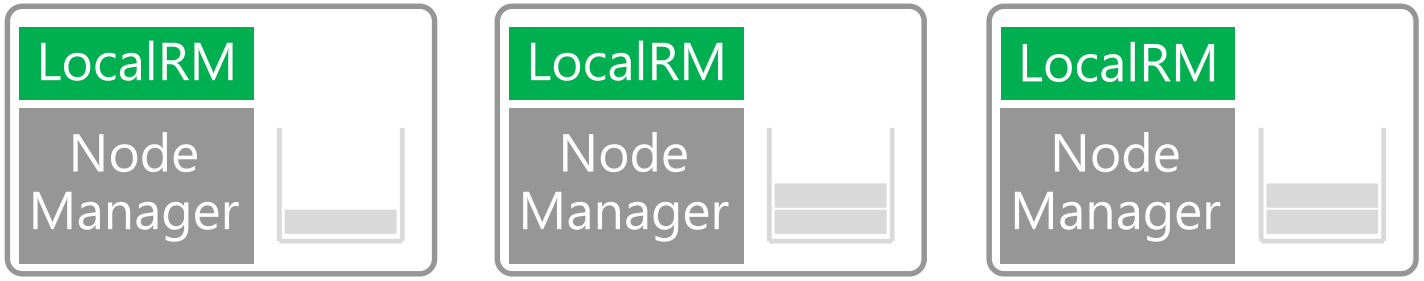


- Example 2 (based on task runtime)
  QUEUEABLE for map tasks (typically fast)
  GUARANTEED for reduce tasks

- Example 3 (on-the-fly DAG optimizations)
  GUARANTEED for initial stages of a DAG
  QUEUEABLE to sample downstream operators

Mercury Architecture over YARN

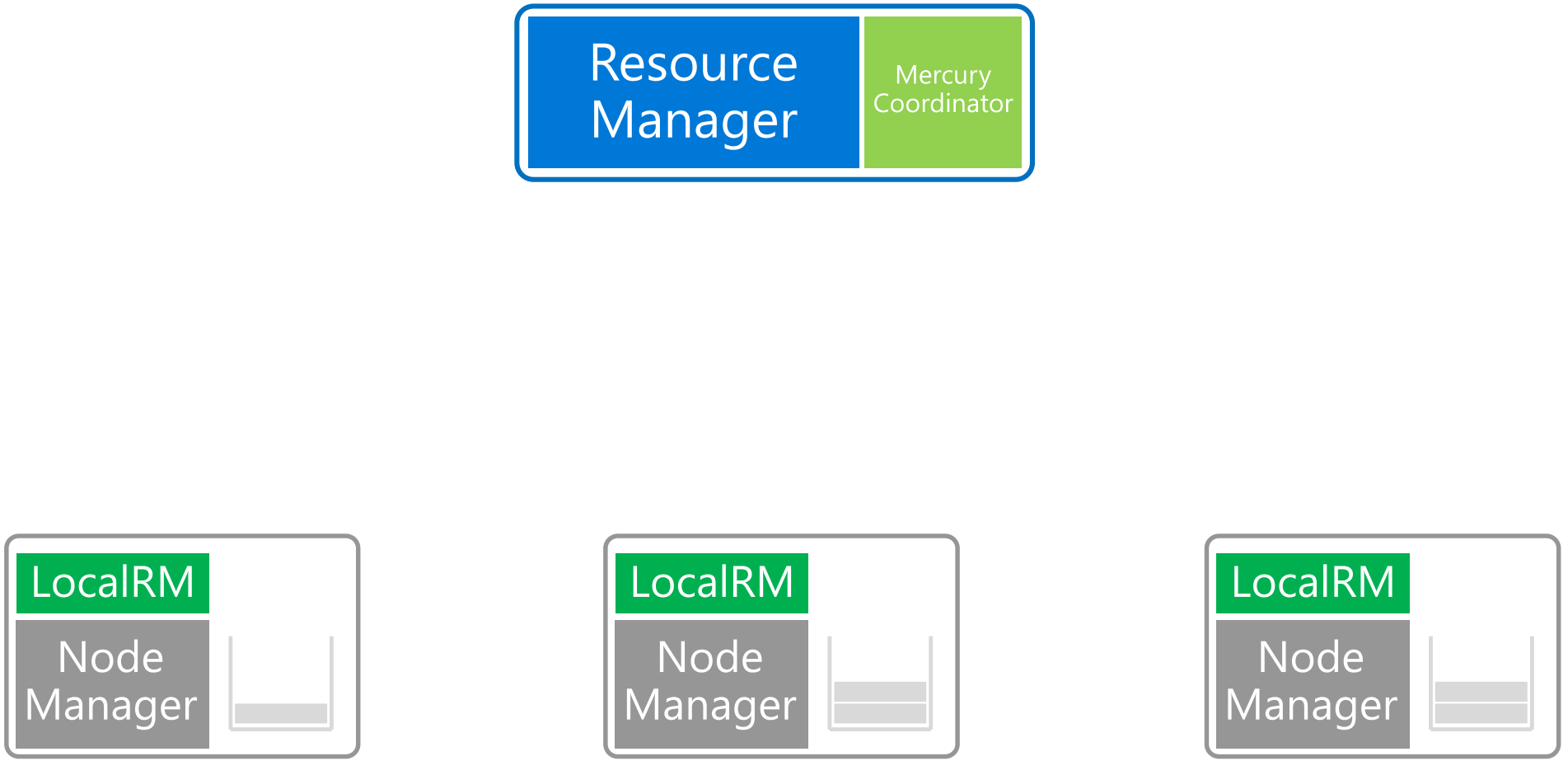




Overview of YARN extensions

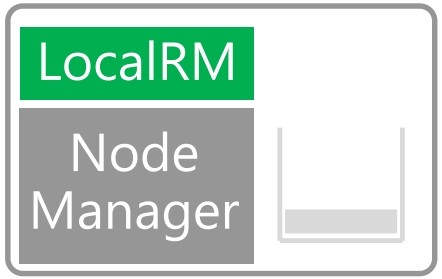
- LocalRM (distributed scheduling)
- Queuing of (QUEUEABLE) containers at the NMs
- Framework policies
- Application policies for determining container type per task

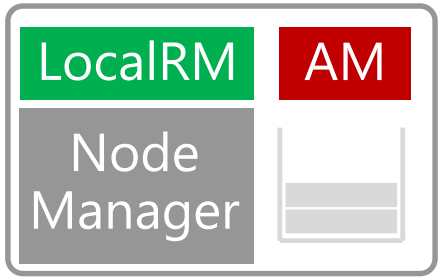
GUARANTEED Request and Allocation

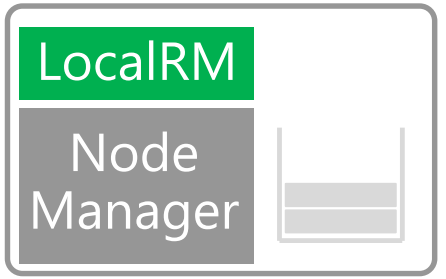


GUARANTEED Request and Allocation





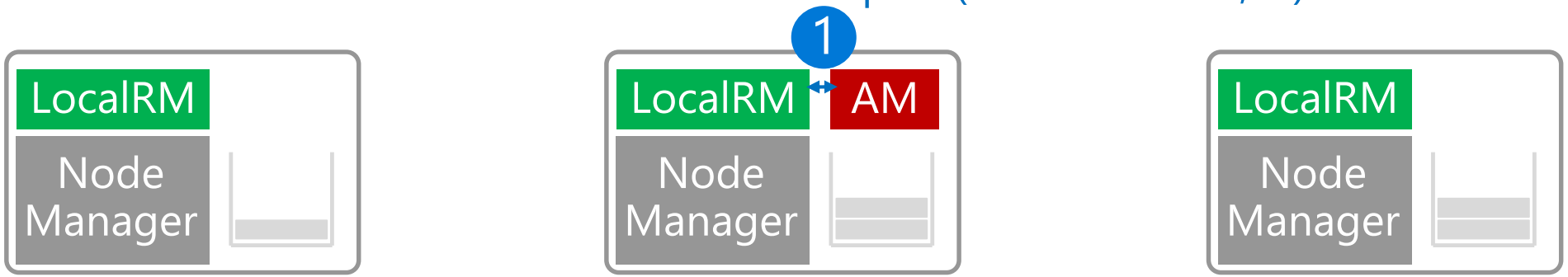




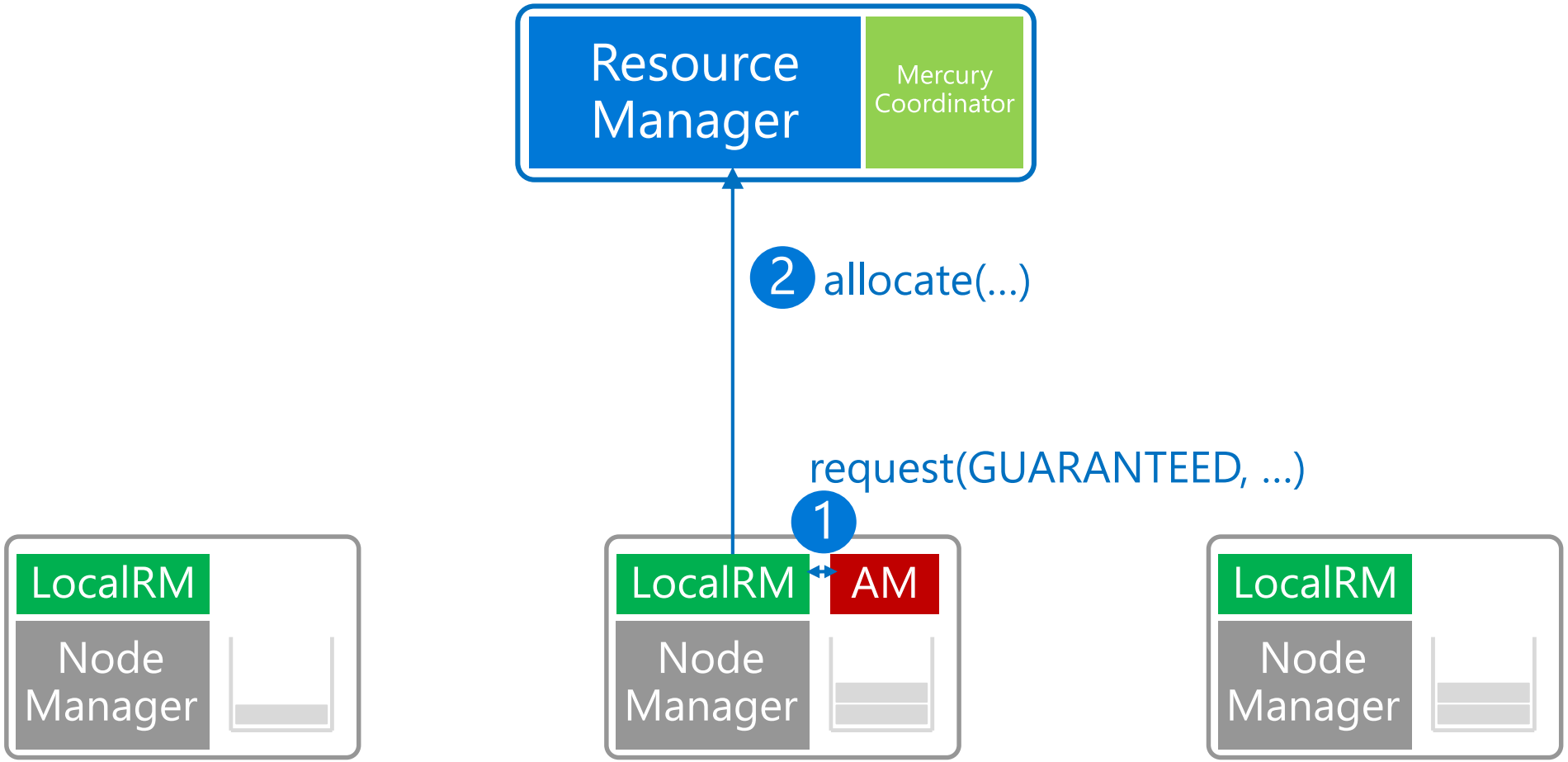
GUARANTEED Request and Allocation



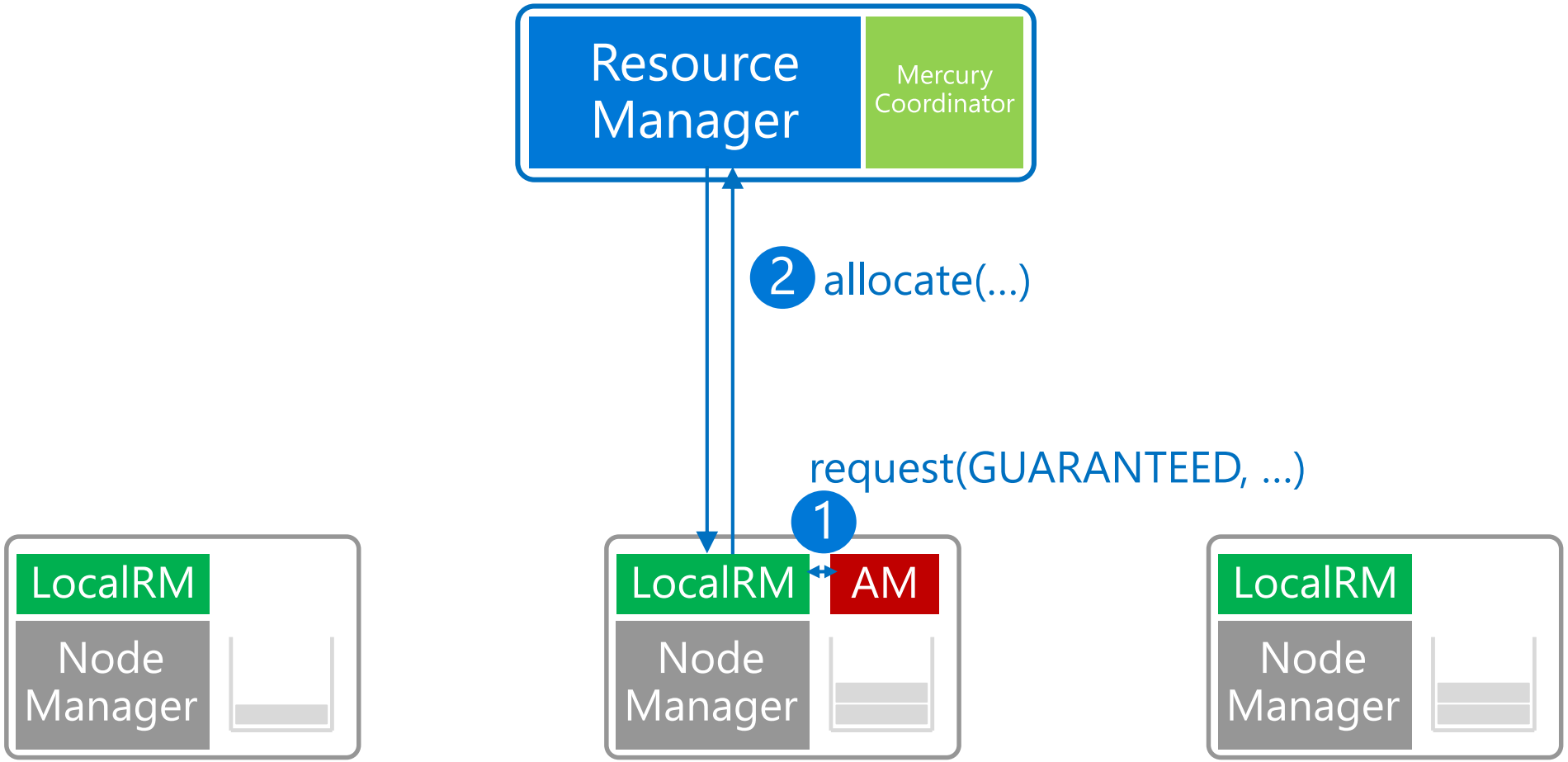
request(GUARANTEED, …)



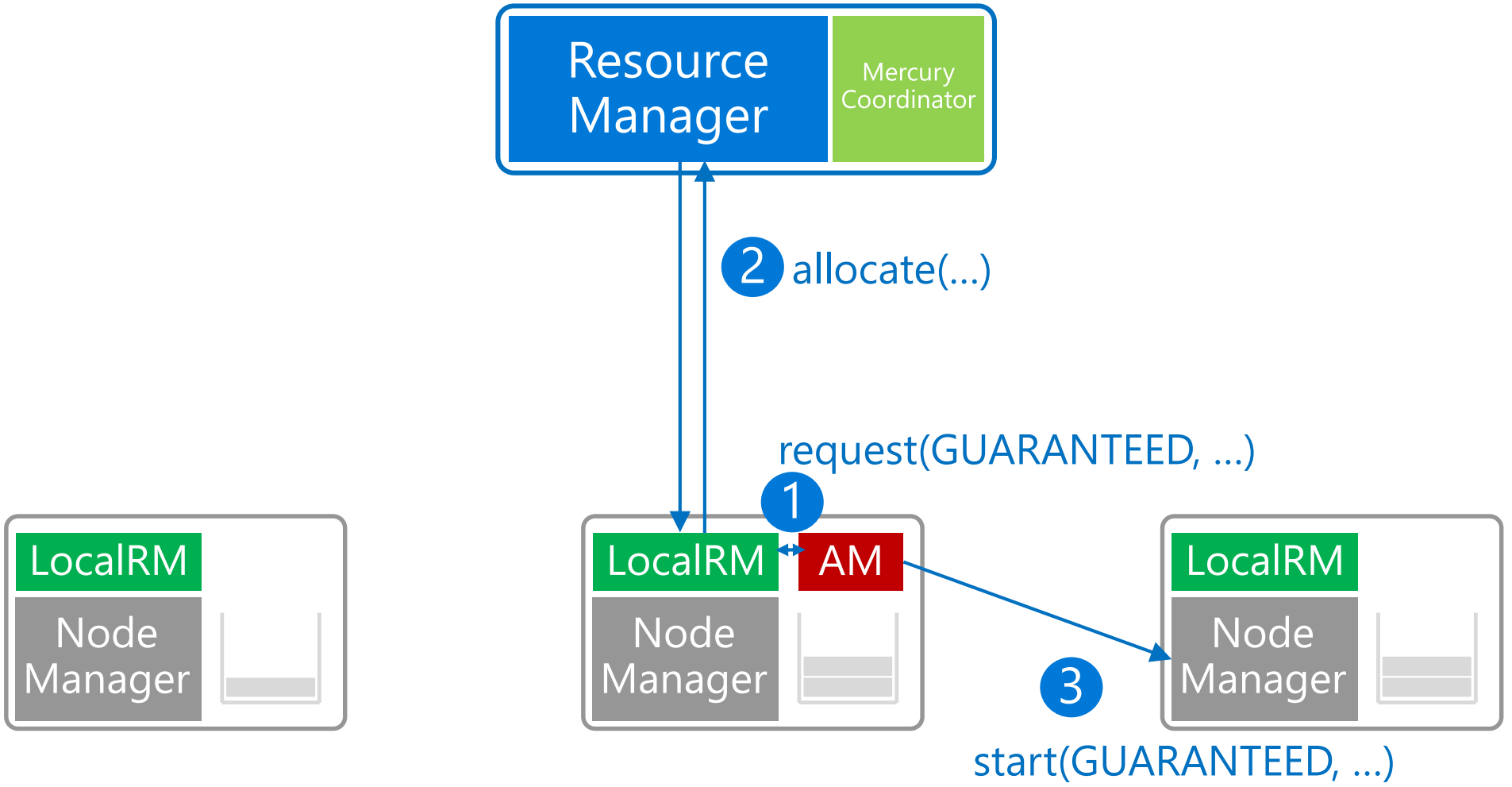
GUARANTEED Request and Allocation



GUARANTEED Request and Allocation

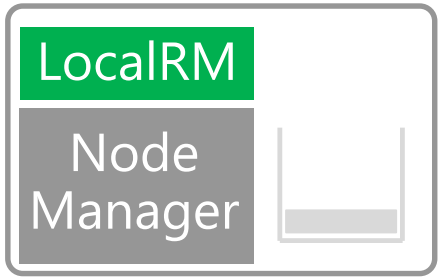


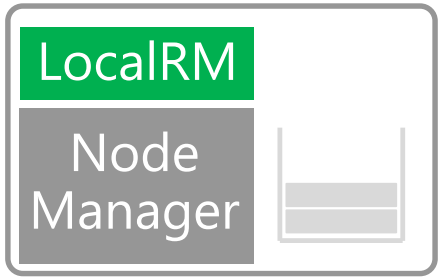
GUARANTEED Request and Allocation

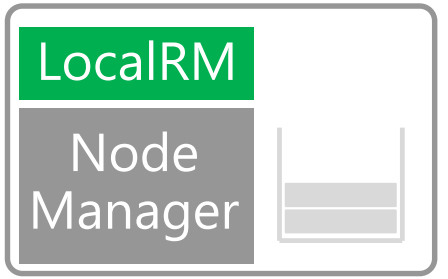


QUEUEABLE Request and Allocation



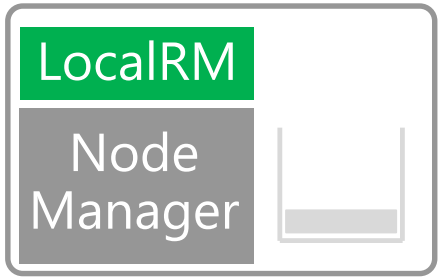


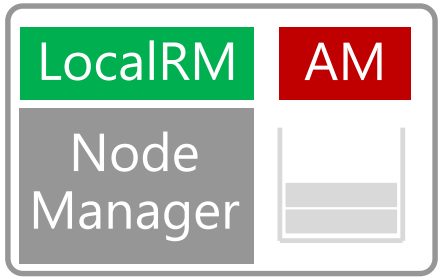


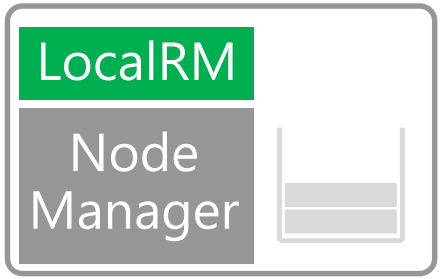


QUEUEABLE Request and Allocation





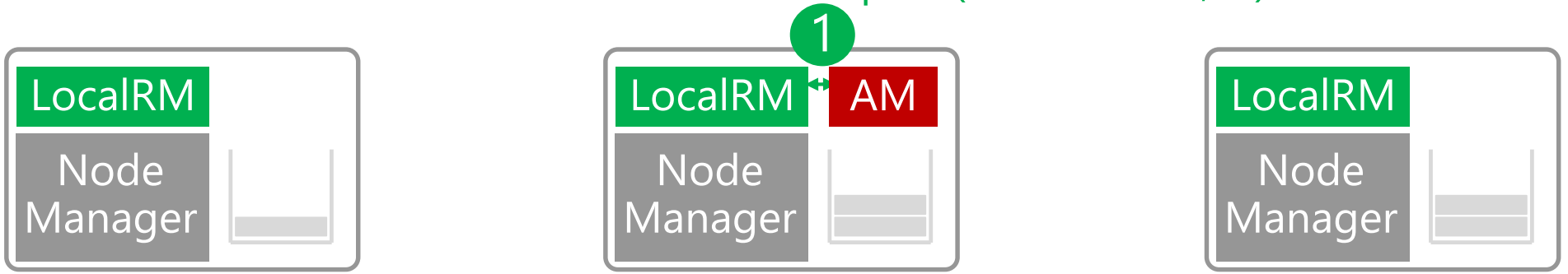




QUEUEABLE Request and Allocation

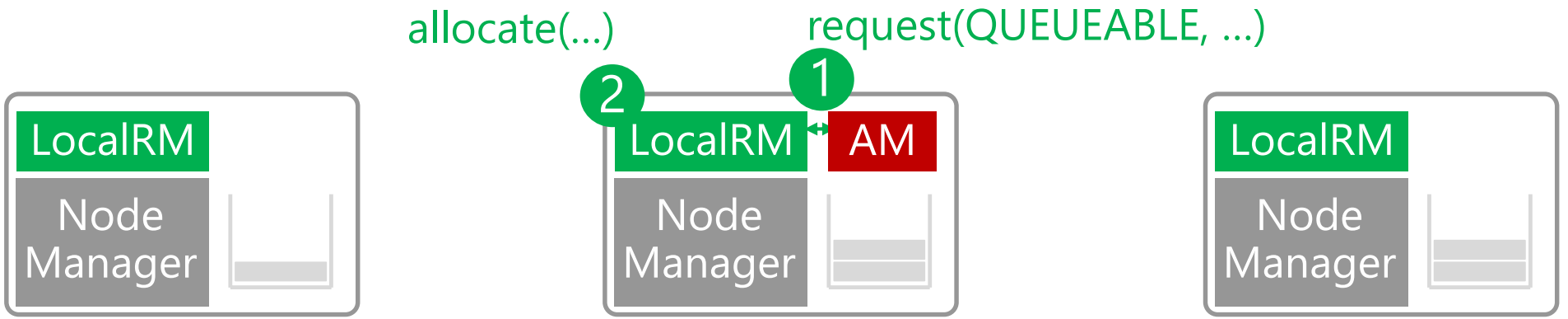


request(QUEUEABLE, …)



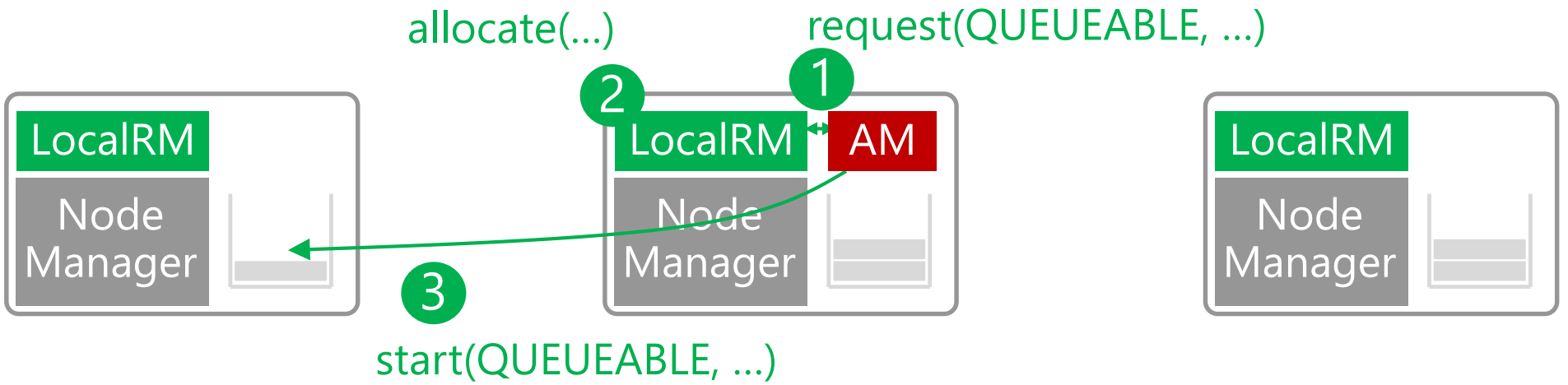
QUEUEABLE Request and Allocation





QUEUEABLE Request and Allocation





Task Execution: Conflict Resolution

… due to containers of two priorities, handed by two types of schedulers, over shared resources

- GUARANTEED – GUARANTEED

  Not possible: central scheduler will not oversubscribe a node with GUARANTEED
- GUARANTEED – QUEUEABLE

  GUARANTEED wins, QUEUEABLE will be pre-empted/killed
- QUEUEABLE – QUEUABLE

  Resolve through the queuing mechanism

Application Policies

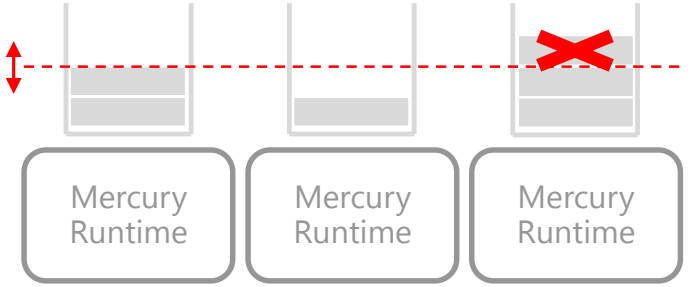
- Mercury exposes the API for apps to choose container type
- App policies determine the container type to be requested for each task
- `hybrid-GQ(`t_d`, `p_q`)`
  Give QUEUEABLE containers to all tasks shorter than t_d secs in p_q percent of the cases
- Flexible enough to perform:
  - Fully centralized scheduling (stock YARN)
  - Fully distributed scheduling
  - Distributed for (a percentage of) the small tasks
    e.g., <10sec-70%-Q

Framework Policies

- Enforcing invariants
  - Capacity/fairness for GUARANTEED containers
  - Quotas for QUEUEABLE containers
- Placement policies
  - Locality constraints
  - Minimize queuing delays for QUEUEABLE
    (top-k based placement using queuing delay estimates)
- Load Shaping policies
  - Maximize cluster efficiency

Load Shaping Policies

- Dynamically rebalance load across nodes
  Imbalance due to stale or imprecise queuing delay estimates
  Re-queue tasks when queuing delay higher than *mean+2σ*



- Queue reordering
  Currently based on job arrival time: improves job tail latency and task throughput
  Take task duration into account?

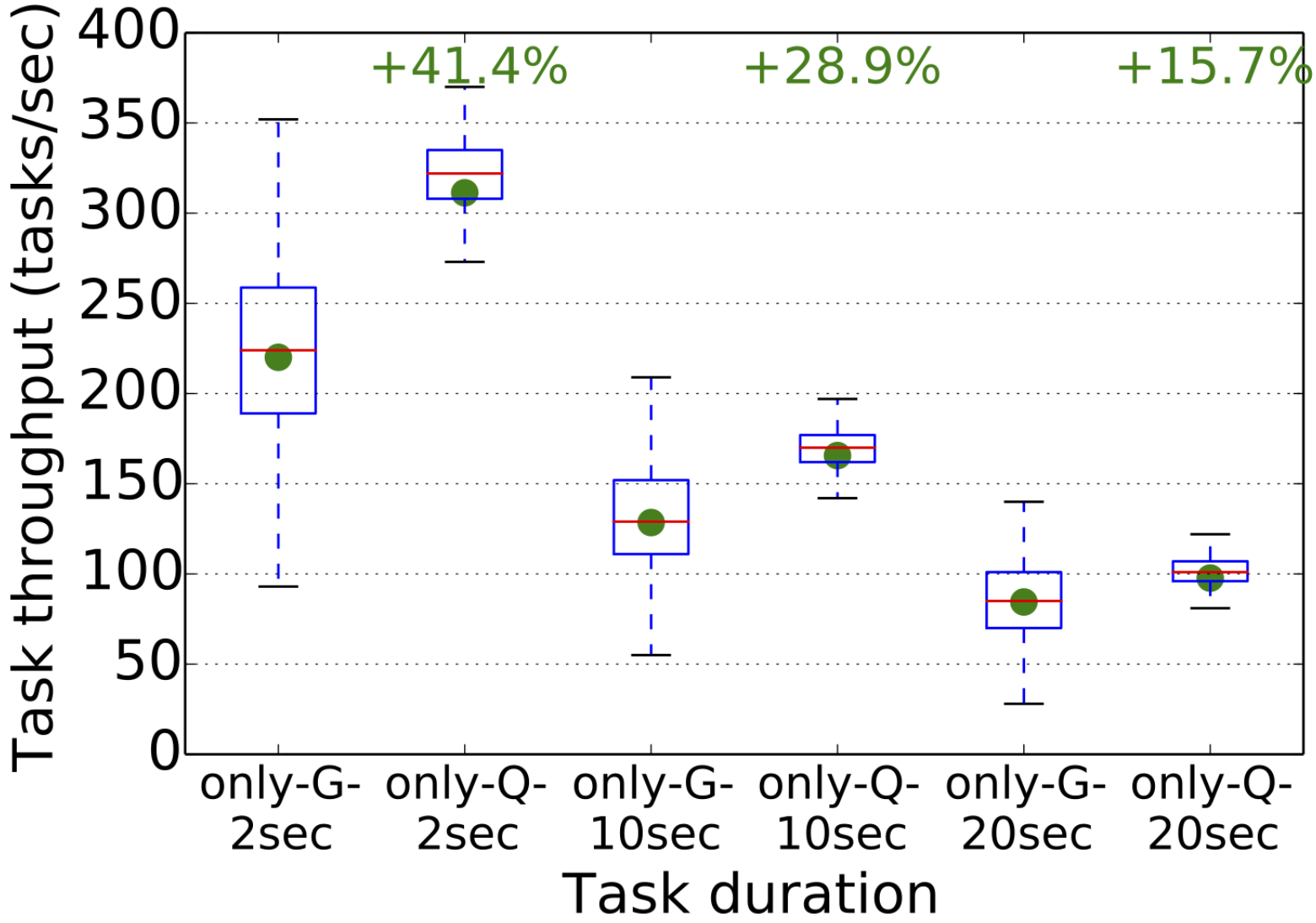
- Resource policing
  Limits for QUEUEABLE containers per node based on (expected) cluster utilization
  Minimize killing of QUEUEABLE containers

Experimental Results

Experimental Setup

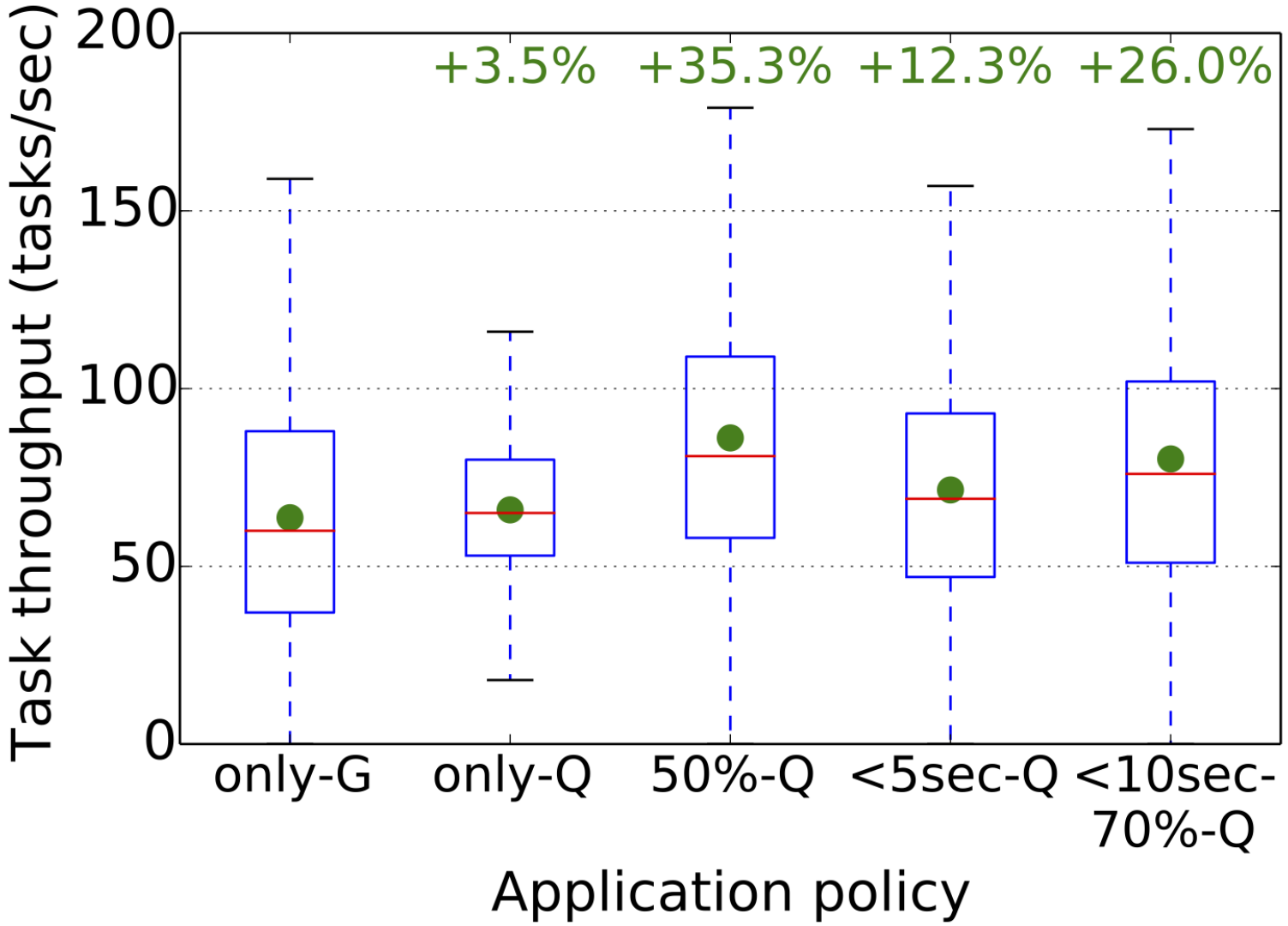
- 256-node cluster
  - 2 x 8-core Intel Xeon processors per node
  - 128GB RAM per node
  - Network: 10 GBps within rack, 6 GBps across racks
- YARN 2.4.1 with Mercury extensions
- Tez 0.4.1 as the execution framework
- Gridmix for workload generation and submission
- Both synthetic and Microsoft-based workloads

Task Throughput for Increasing Task Duration



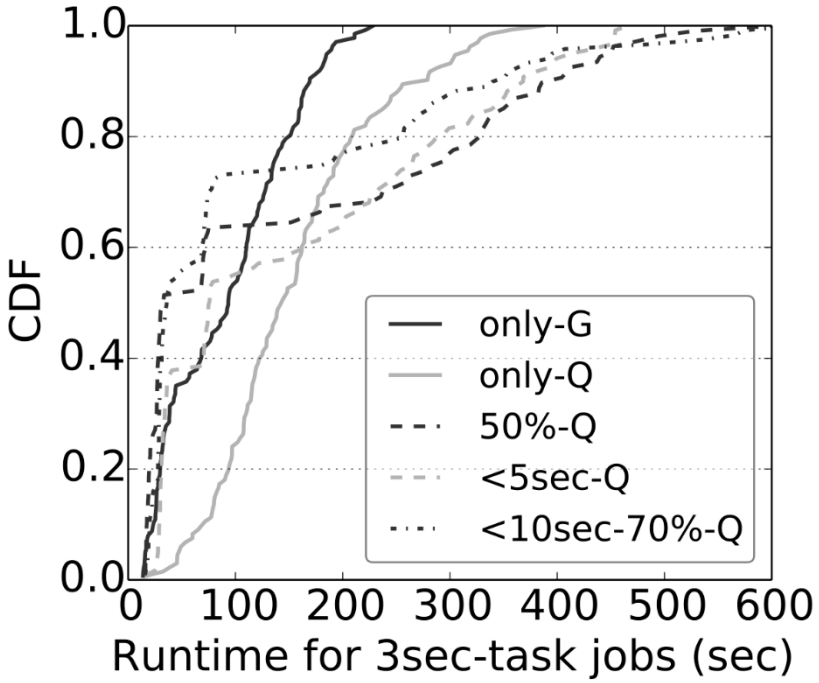
- 41% task throughput improvement for short tasks
- Improvement drops for longer task durations

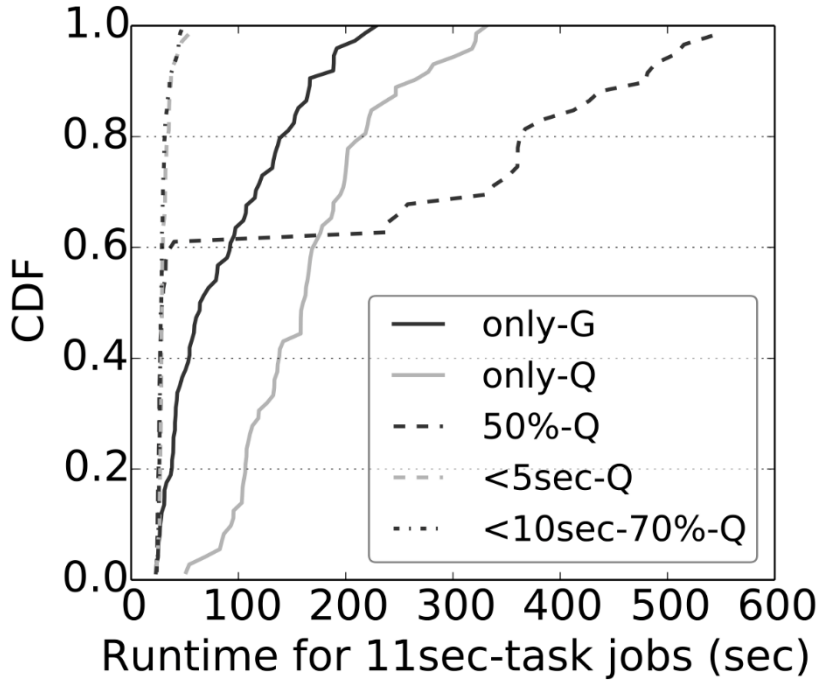
Cosmos-based Workload: Task Throughput

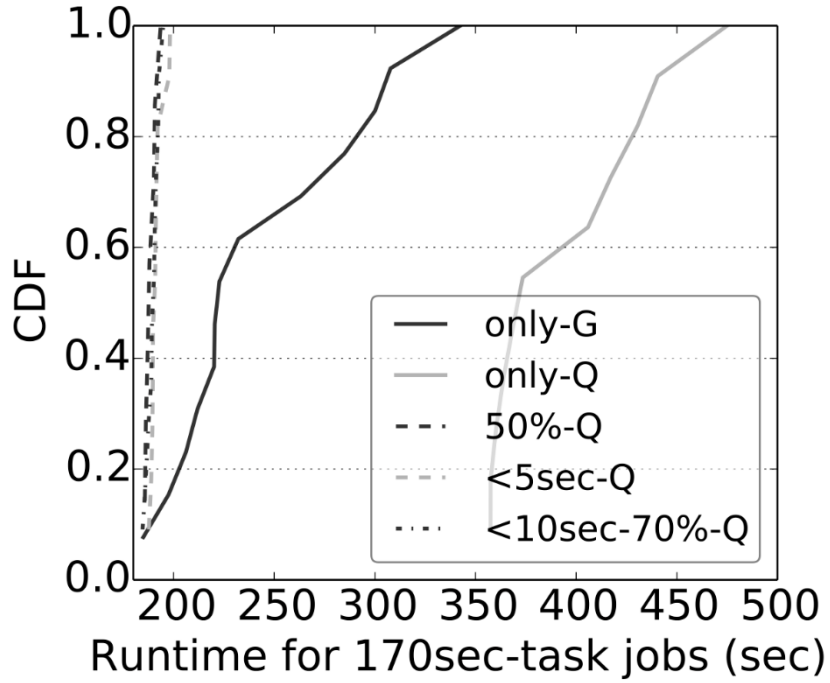


- Up to 35% task throughput improvement
  - Depending on the application policy

Cosmos-based Workload: Job Latency







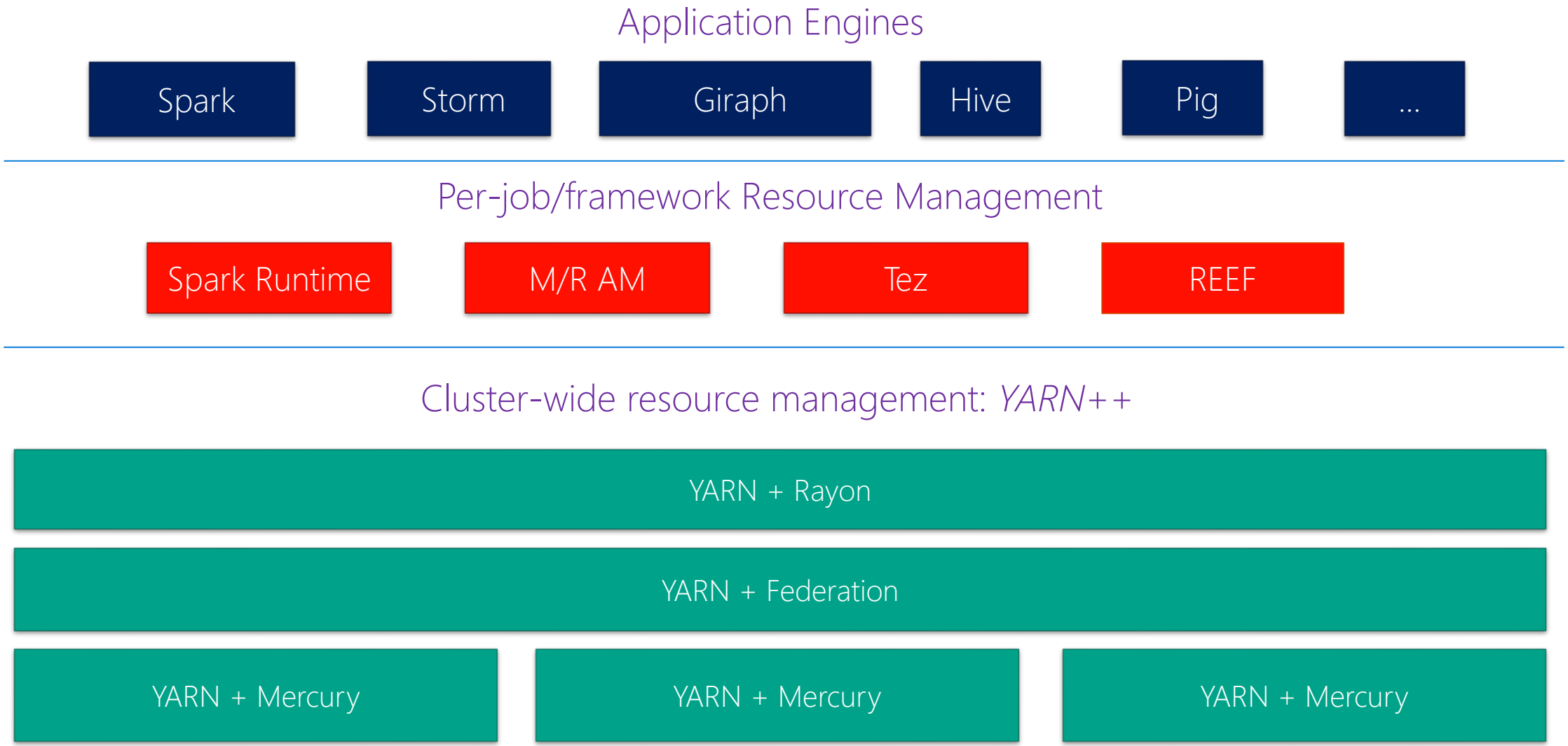
- `50%-Q` cannot translate throughput win to job latency win
- `<10sec-70%-Q`

  Comparable to only-G for short tasks

  Significant improvement for longer ones

Conclusion

The Bigger Picture



Conclusion

- Resource management
  for heterogeneous applications
  in large shared clusters
- Combine best of centralized and distributed schedulers
- Provide applications with API to choose scheduling type
- Up to 40% task throughput and 66% mean job latency gain
- Currently contributing code to Hadoop [*Apache JIRA-2877*]

Thank you!

Questions?

© 2014 Microsoft Corporation. All rights reserved.