CS 7172
Parallel and Distributed Computation

Scheduling

Kun Suo
Computer Science, Kennesaw State University
https://kevinsuo.github.io/

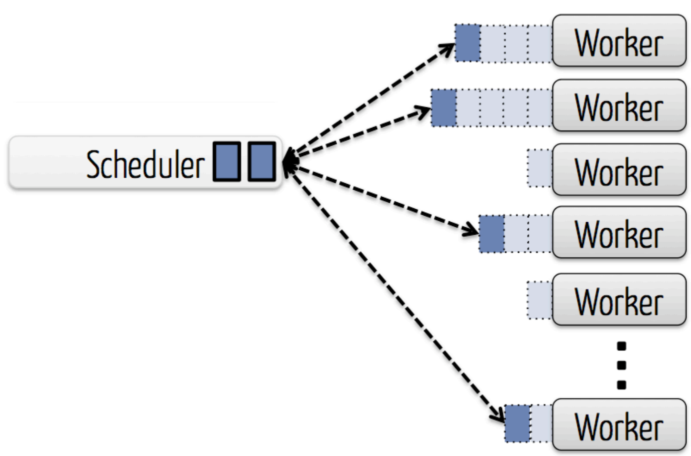
Outline

- Computer networks, primarily from an application perspective
- Protocol layering
- Client-server architecture
- End-to-end principle
- TCP
- Socket programming

Distributed Scheduling

- Distributed system architecture:
  - Centralized structure
  - Decentralized structure

- The purpose of the distributed system architecture is to manage multiple server resources and find the appropriate server to perform user tasks.



Distributed Scheduling

- What is the appropriate server? Many constraints:
  - Task priority
  - Resource availability
  - Load balance
  - …
- The process of finding the appropriate server in distributed systems for user tasks is called *distributed scheduling*

Distributed Scheduling

- The scheduler is one of the most important components in distributed system. The scheduler provides multiple scheduling strategies (FIFO, SJF, etc.) and is responsible for completing specific scheduling tasks.

Hadoop

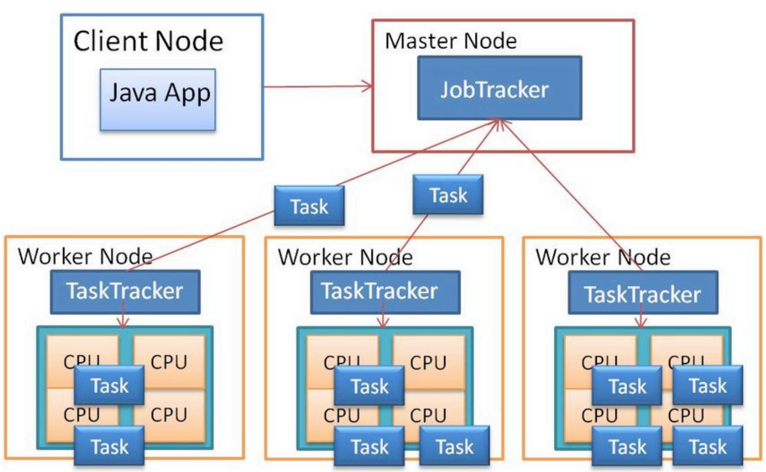
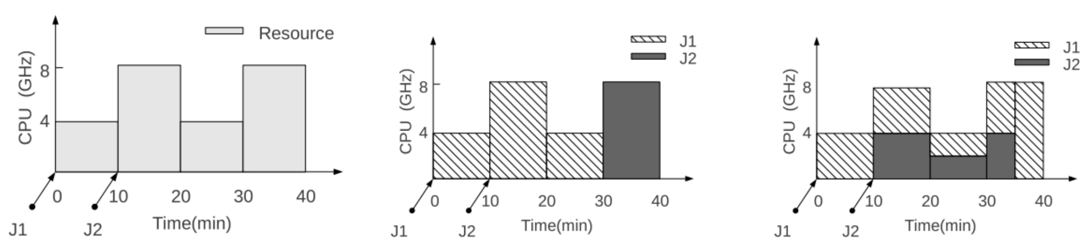


TABLE I
JOB 1 AND 2 SUBMISSION INFORMATION.

Jobs	Input	CPU demand	Arrival time	Deadline
J1	18 GB	120 GHz	0_{th} min	40_{th} min
J2	9 GB	60 GHz	10_{th} min	30_{th} min



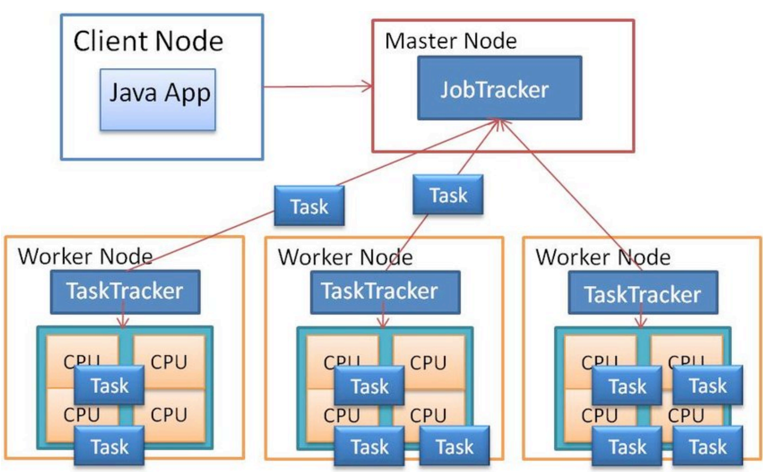
(a) Resource availability. (b) FIFO Scheduler. (c) Fair Scheduler.

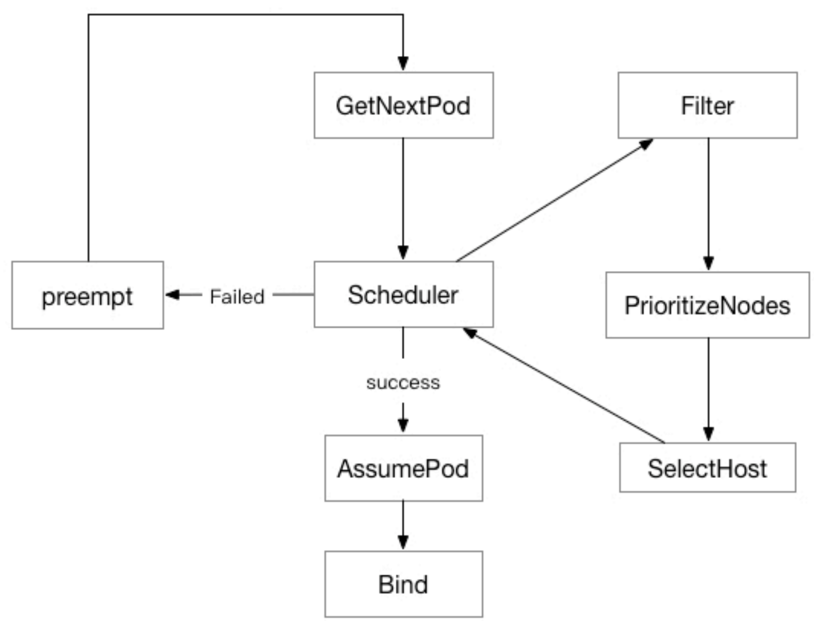
Distributed Scheduling

- The scheduler principle of different distributed architectures is different

Kubernetes

Hadoop

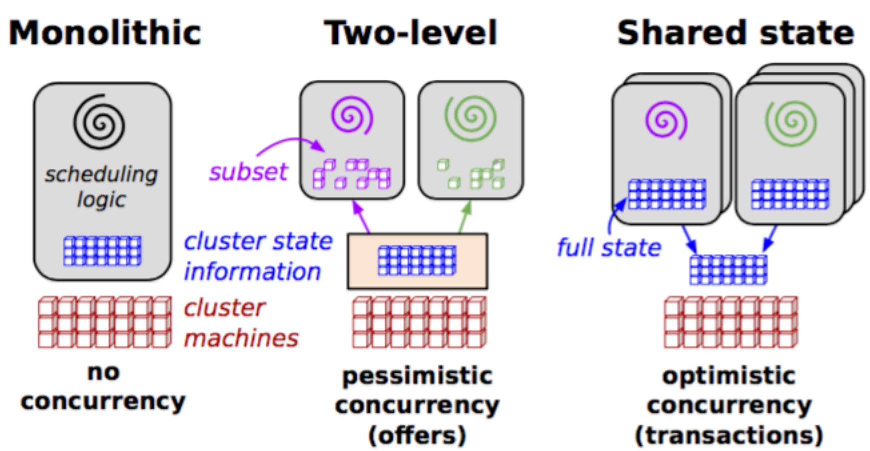




Distributed Scheduling

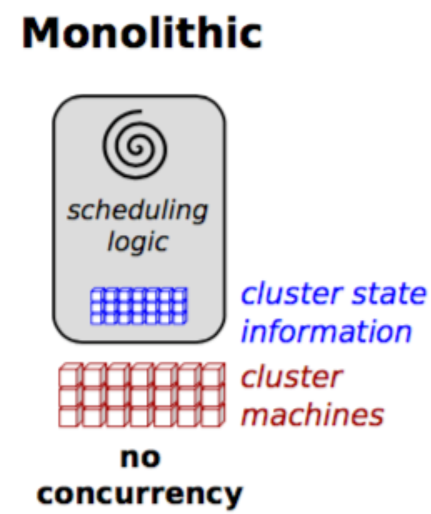
- The most common or intuitive scheduler is the monolithic scheduler, which matches the *user task* with the *idle resources* in the distributed systems.

- Monolithic scheduler manages both tasks and resources

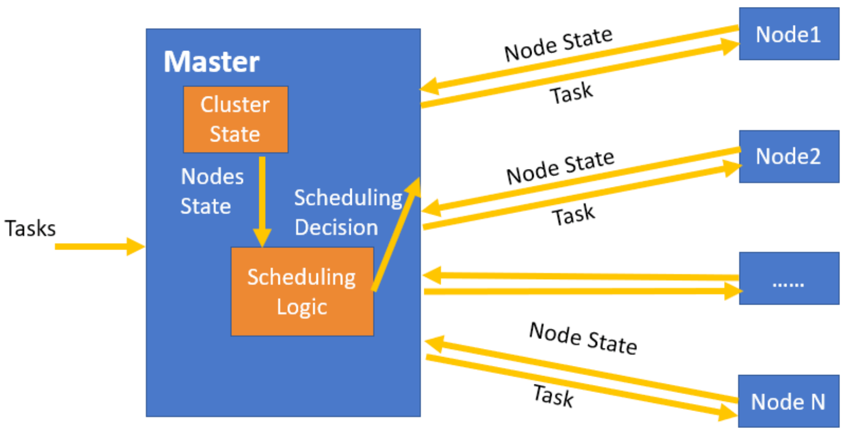


What is Monolithic Scheduler?

- Only one node in a cluster runs the scheduling process.
- This node can collect other node's resource information and status for unified management. According to the resource requirements of tasks, the scheduler matches tasks with available resources.
- The monolithic scheduler has a global view on resource and tasks, which can easily implement task constraints and global scheduling strategies

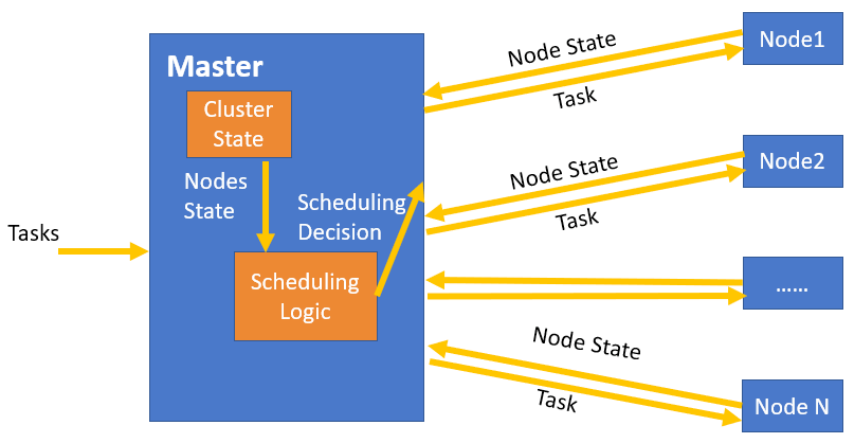


Monolithic Scheduler Example



- The scheduling process runs on the master node (responsible for resource management, tasks, and resource matching)
- Slave nodes report “Node state” to the Cluster State on master node, which manages the resources and states of nodes in the cluster

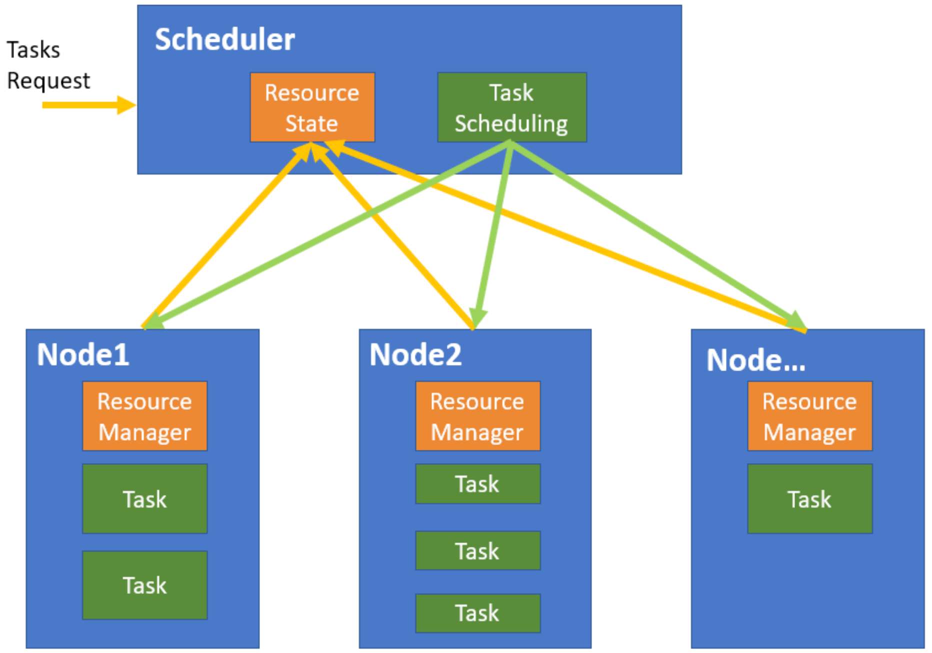
Monolithic Scheduler Example



- Cluster State reports resource states to Scheduling Logic module, which is responsible for matching between tasks and resources
- After scheduling decision is made, the task will be sent to corresponding node in the cluster

Monolithic Scheduler

- Monolithic scheduler is also called centralized scheduler, which manages the ***resources*** and ***tasks*** on a single node



How monolithic scheduler works? Using Borg as an example

- Monolithic scheduler: find a match between tasks and resources

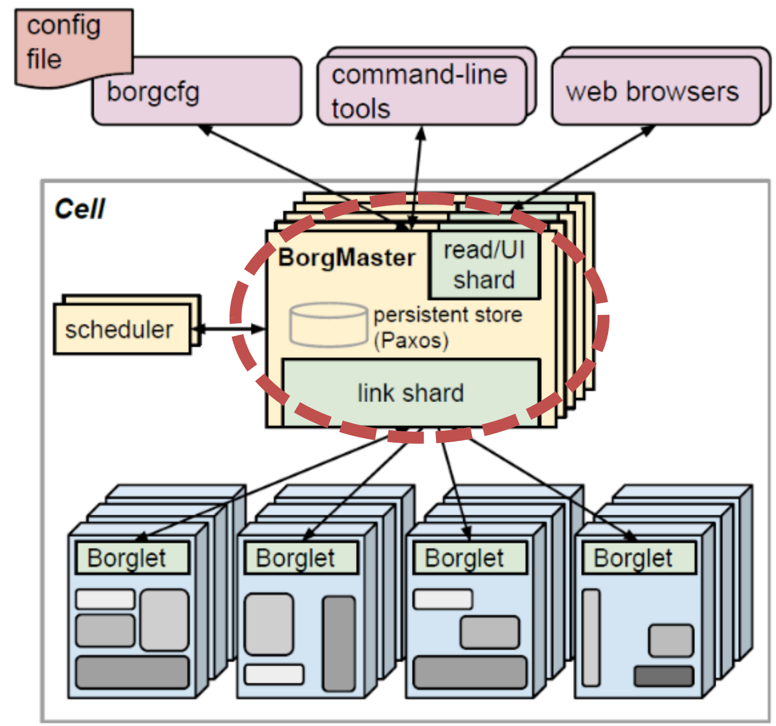
- Definition of jobs:
  - A job is composed of batches of tasks
  - A job can only execute on a cluster

- Definition of tasks
  - A task is composed of batches of processes
  - A task executes on a node in the cluster

How monolithic scheduler works? Using Borg as an example

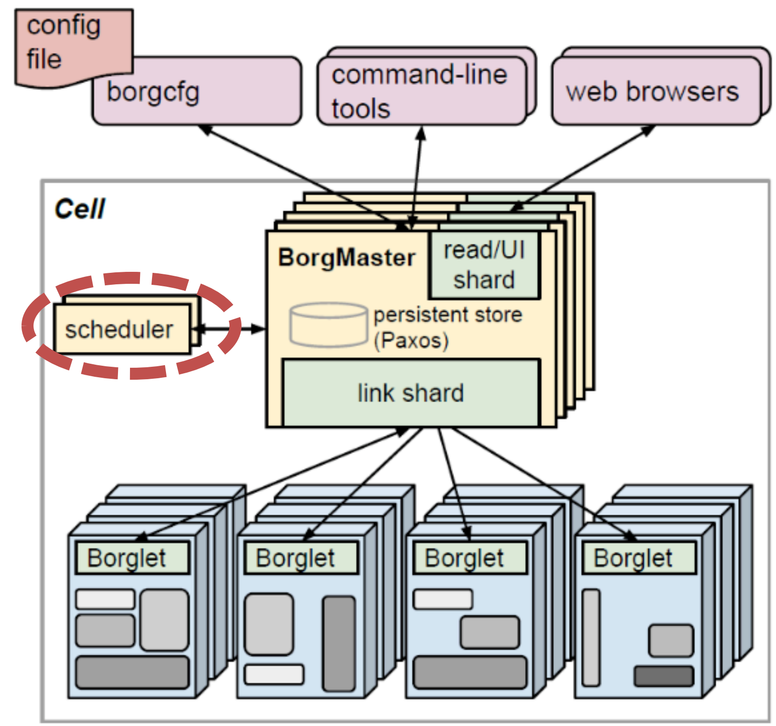
1. When a user submits a job to BorgMaster, it saves the job to the Paxos warehouse and add all tasks of this job to the waiting queue.

2. The scheduler scans the waiting queue asynchronously and assigns tasks to computing nodes that meet constraints and have enough resources. Here the scheduling unit is task not job.



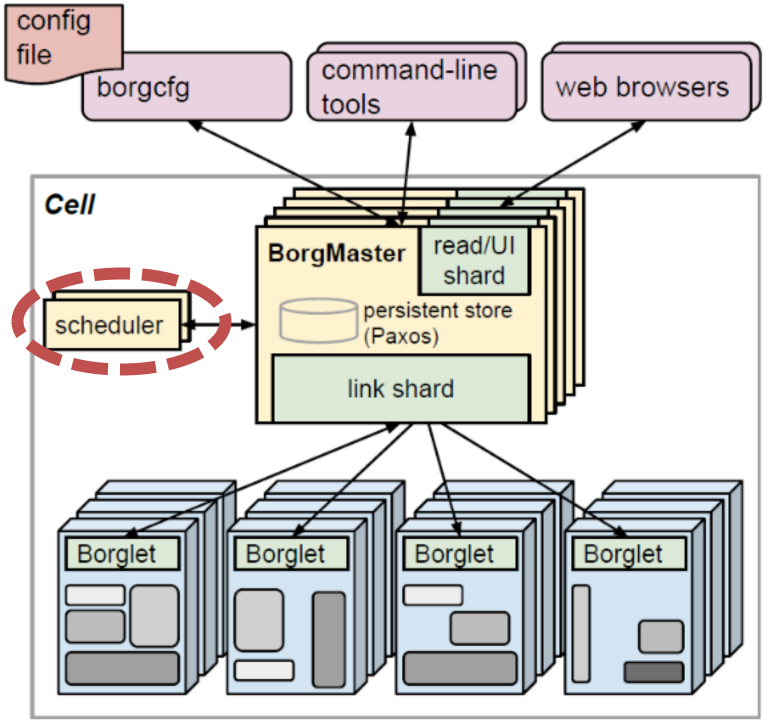
How monolithic scheduler works? Using Borg as an example

- Feasibility check:
  - the scheduler will find a set of machines that meet the task constraints and have enough available resources
  - Suppose Job A requires to be deployed to Node 1, 3 and 5 and consumes 0.5 CPU and 2MB memory. Scheduler will first filter Node 1, 3, 5 and then select one node has 0.5 CPU and 2MB memory.



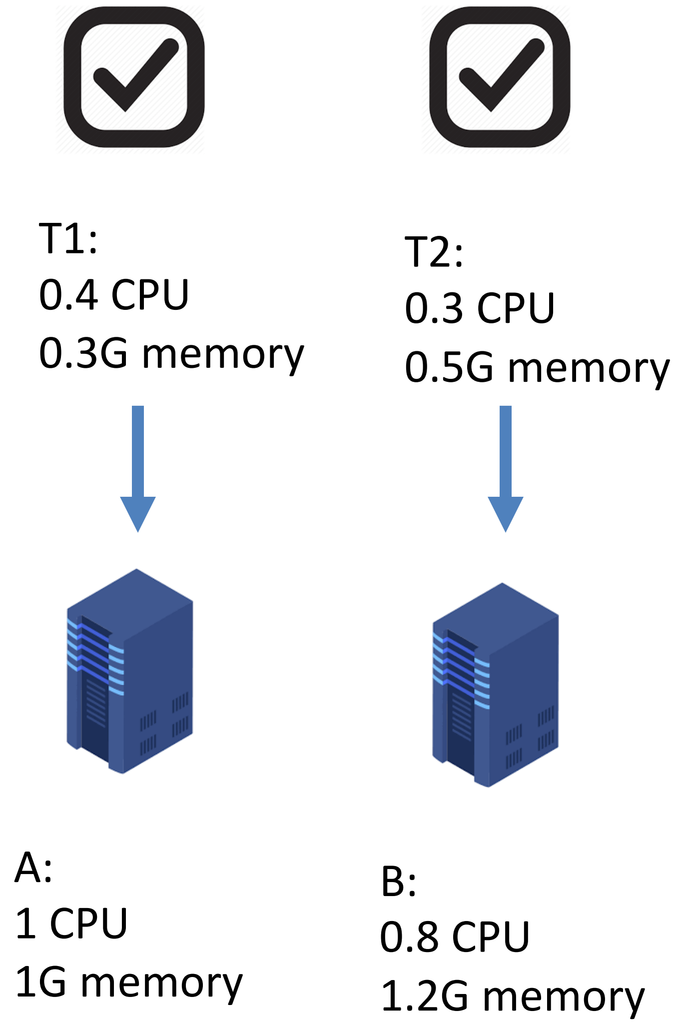
How monolithic scheduler works? Using Borg as an example

- Rating:
  - Borg scores the selected machines in the feasibility check stage according to a scoring mechanism and selects the one that is most suitable for scheduling
  - Common scoring algorithms, including "worst fit" and "best fit"



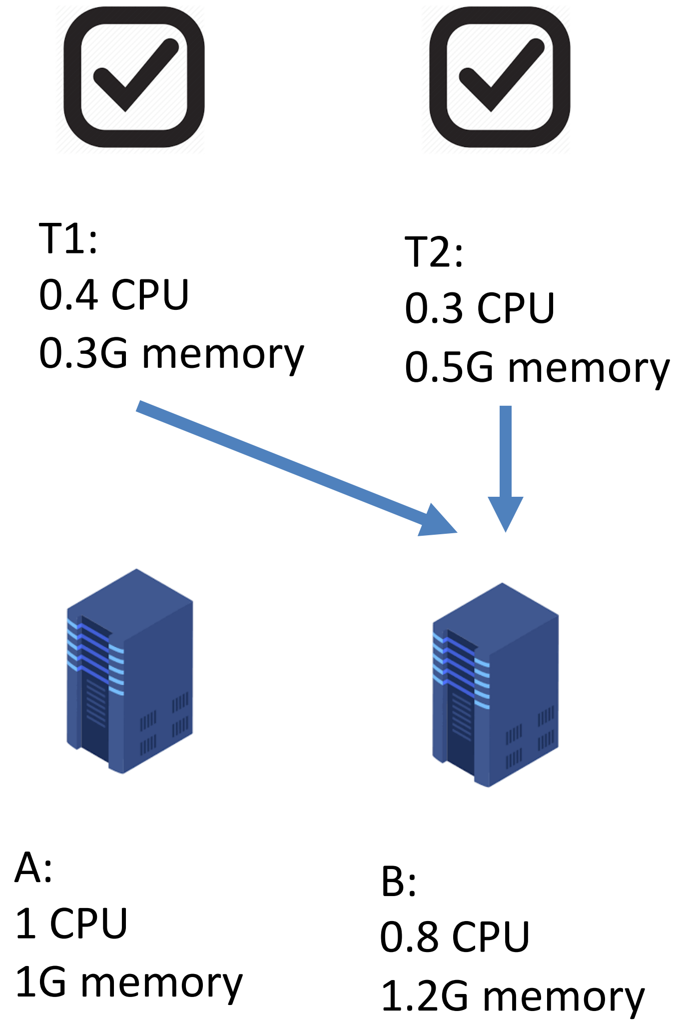
How monolithic scheduler works? Using Borg as an example

- Worst fit:
  - distribute tasks across as many as possible different machines
  - Possible problem: it results in a small amount of unusable remaining resources for each machine. Resource fragmentation.



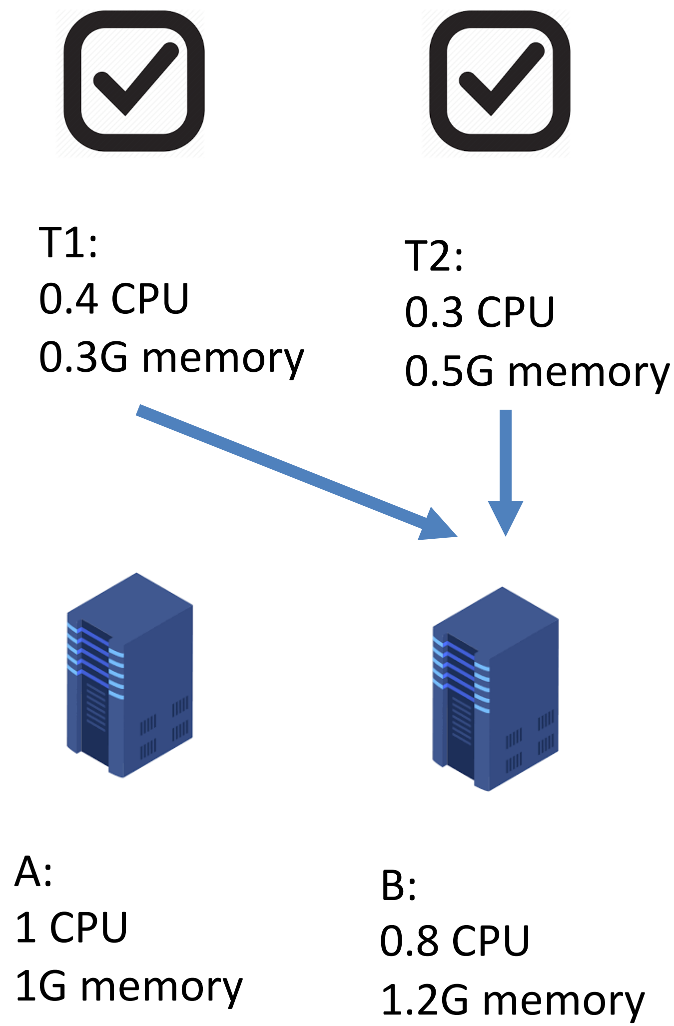
How monolithic scheduler works? Using Borg as an example

- Best fit:
  - distribute tasks across as less as possible different machines
  - T1 and T2 will be put onto the same server
  - Possible problem: is not good for applications with sudden loads and requiring immediate handling. Everything on one server causes single node failure and low reliability.



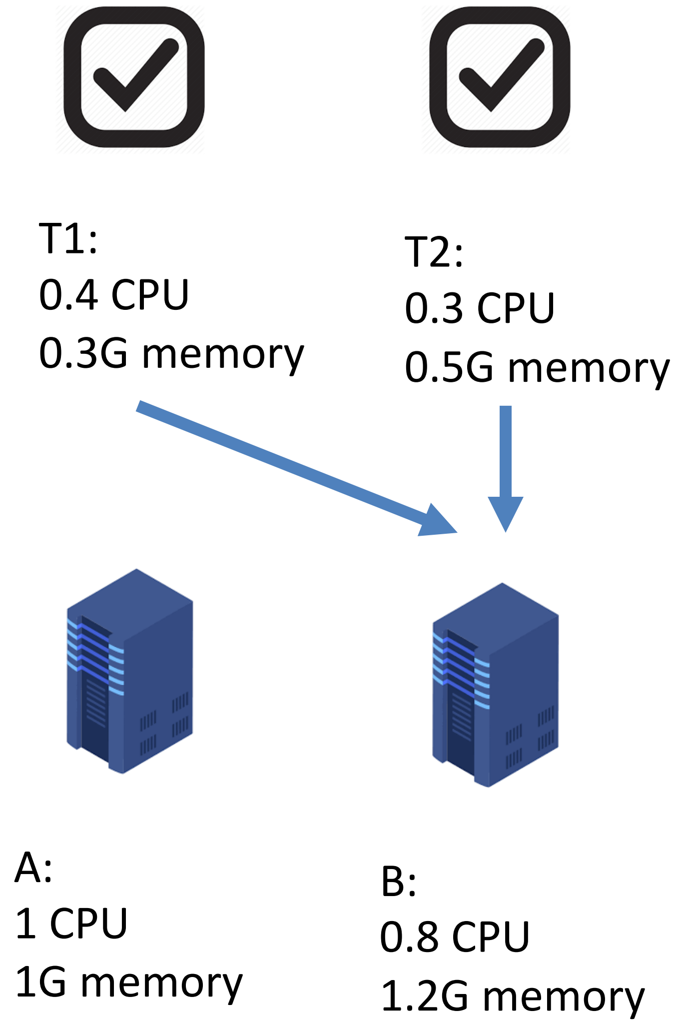
How monolithic scheduler works? Using Borg as an example

- For scenarios where resources are limited, and business traffic is relatively regular, and there are basically no sudden traffic, the best fit works better;
- If resources are abundant and business traffic often occurs unexpectedly, the worst fit algorithm should be used.



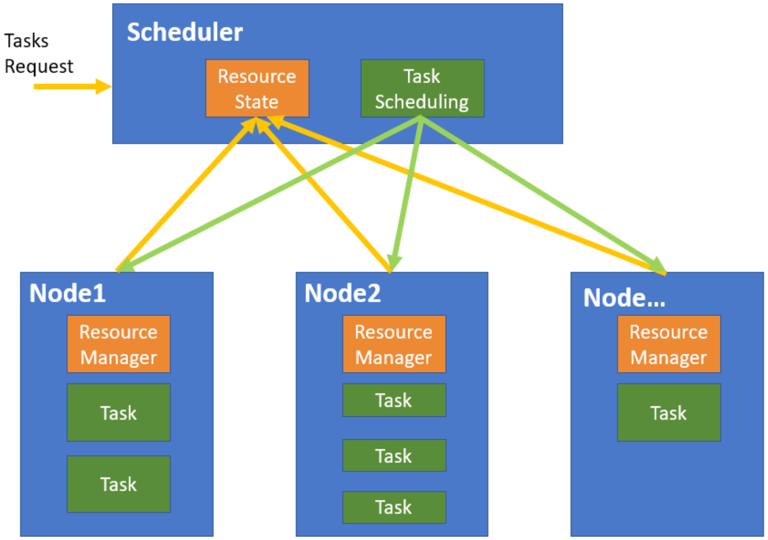
How monolithic scheduler works? Using Borg as an example

- Besides best fit and worst fit, other possible scheduling policies include:
  - Priority-based
  - Locality-based
  - …



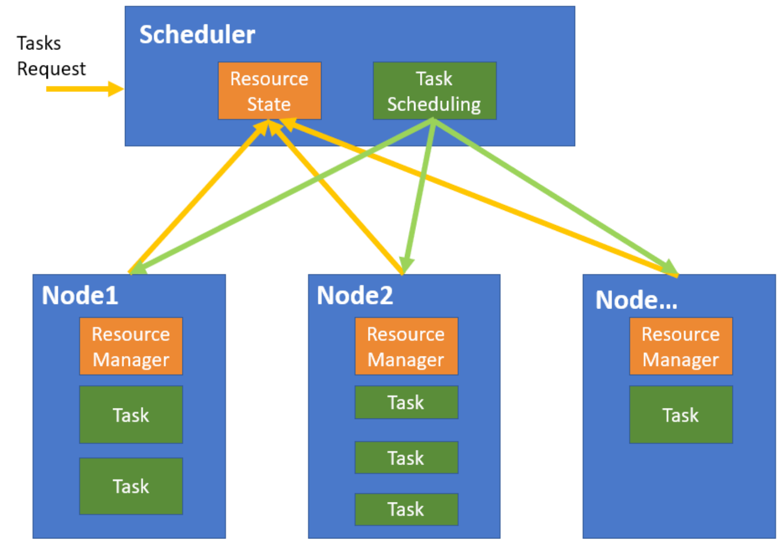
Monolithic Scheduling

- Possible issues:
  - The central server can easily become a single point of bottleneck, which will directly lead to the limitation of the scale
  - Low reliability, single node failure
  - Limited to support jobs with different characteristics (e.g., supporting two policies at the same time? Batching jobs and latency-sensitive jobs at the same time?)



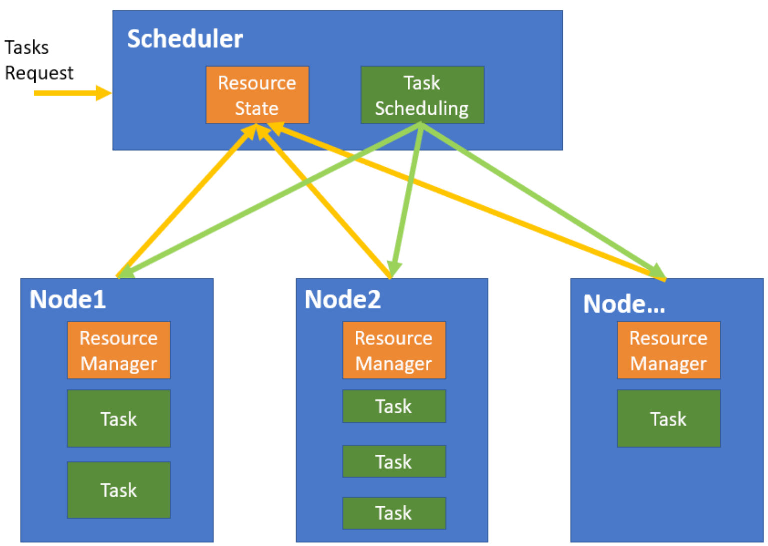
Monolithic Scheduling

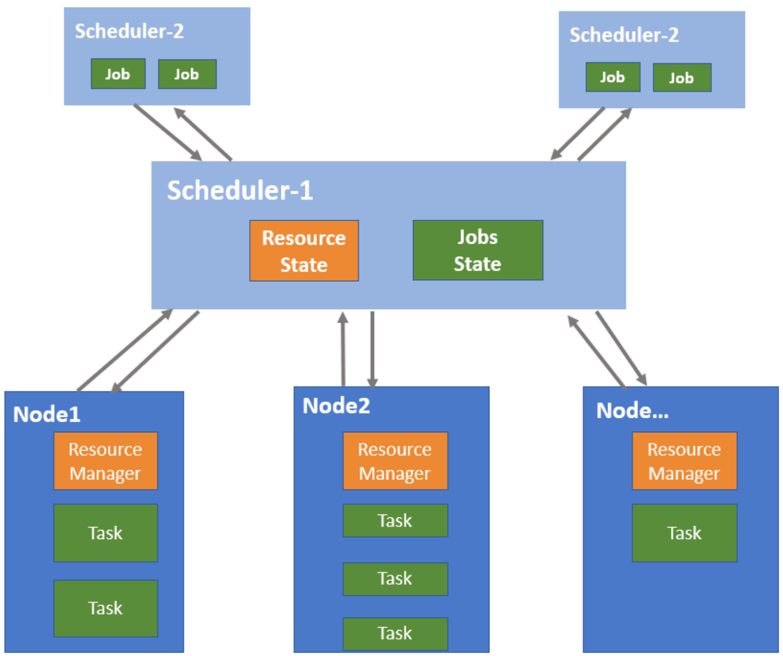
- Possible issues:
  - Limited to support jobs with different characteristics
    - E.g., supporting two policies at the same time?
    - E.g., batching jobs (static content) and streaming jobs (dynamic content) at the same time?
    - Monolithic scheduling becomes more complex as the category of jobs increases.



Two-level Scheduling

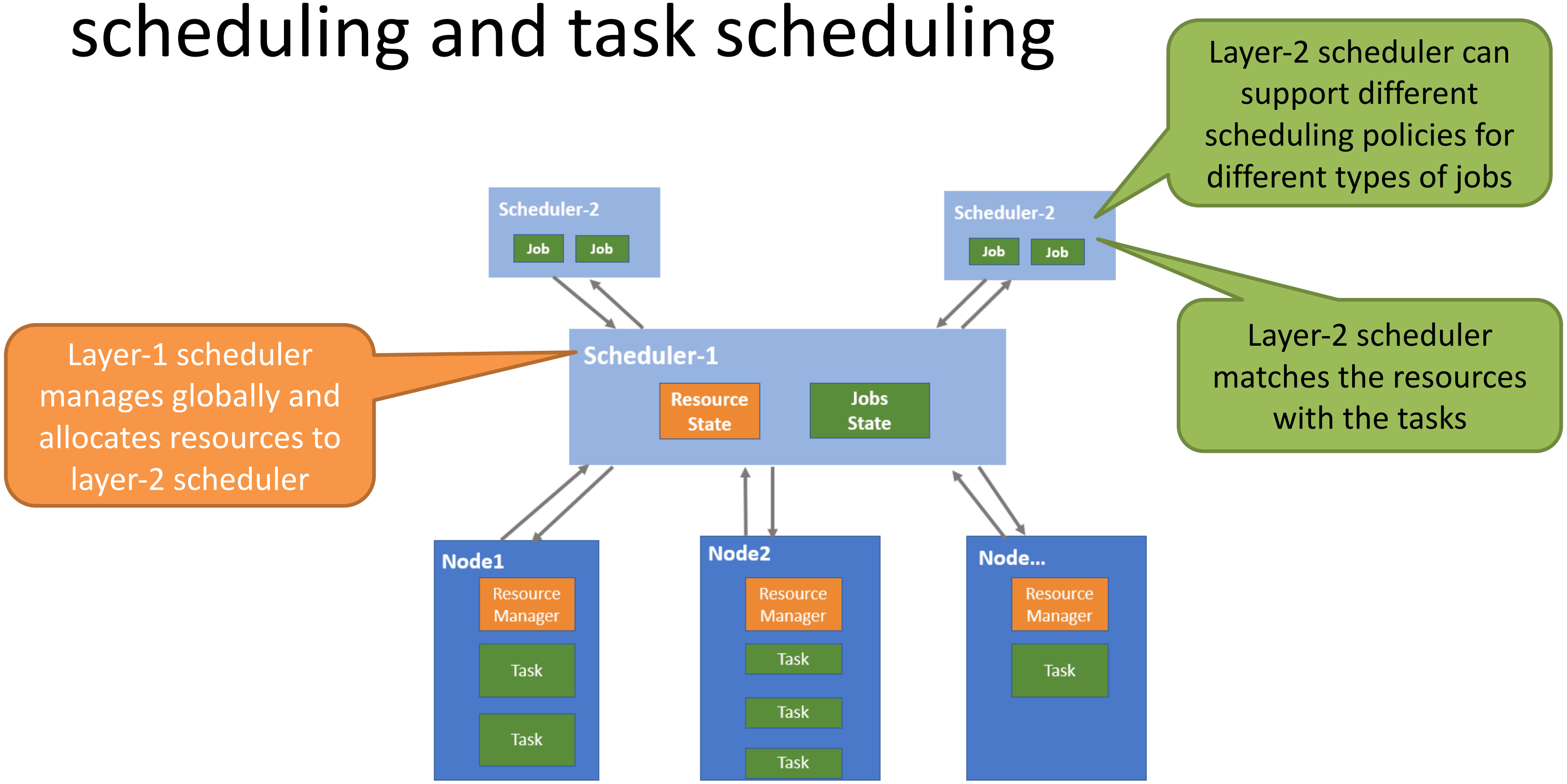
- Two-level Scheduling: separate the resource scheduling and task scheduling





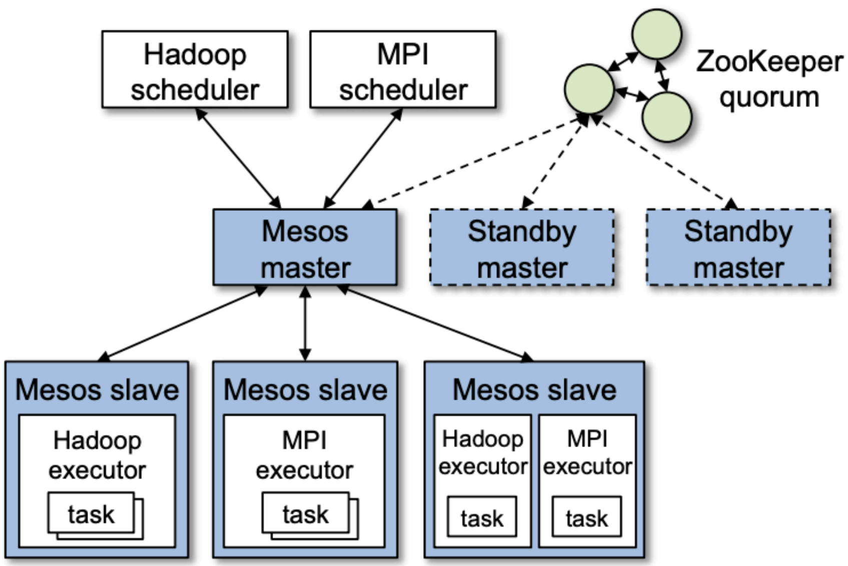
Two-level Scheduling

- Two-level Scheduling: separate the resource scheduling and task scheduling



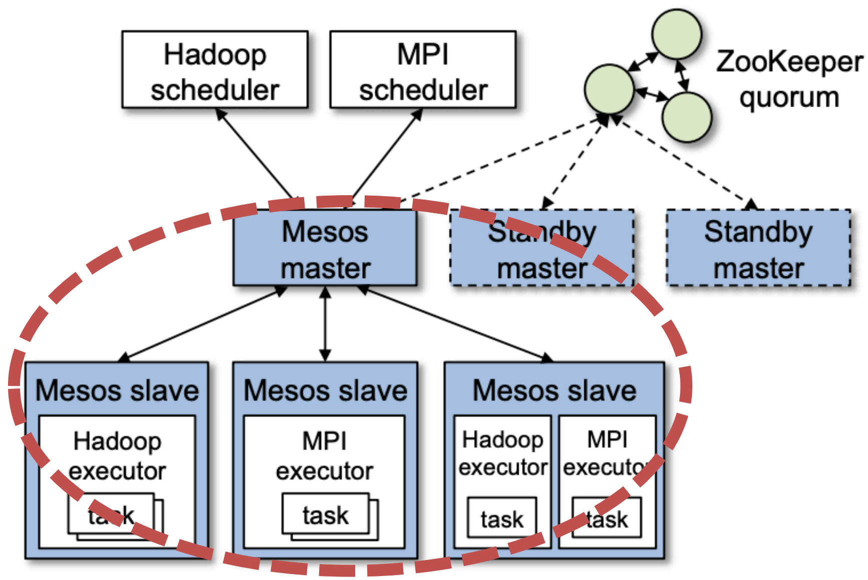
Example of Two-level Scheduling: Mesos

- Two-level Scheduling
  - Mesos, first-level scheduling, is only responsible for the management and allocation of the underlying resources, and does not involve functions such as storage and task scheduling
  - The second-level task scheduling is through the framework, such as Hadoop, Spark, etc., to complete



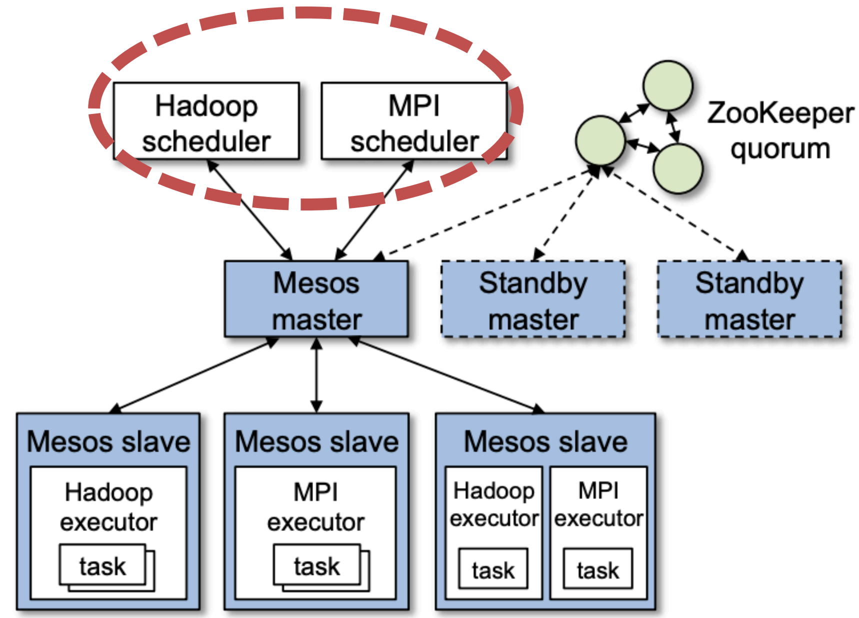
Example of Two-level Scheduling: Mesos

- First-level Scheduling
  - The resource management cluster is a centralized system composed of a master node and multiple slave nodes.
  - The cluster has only one master node, which is responsible for managing slave nodes and connecting to the upper-level framework
  - The slave nodes periodically report resource status information to the master node and execute tasks submitted by the framework.



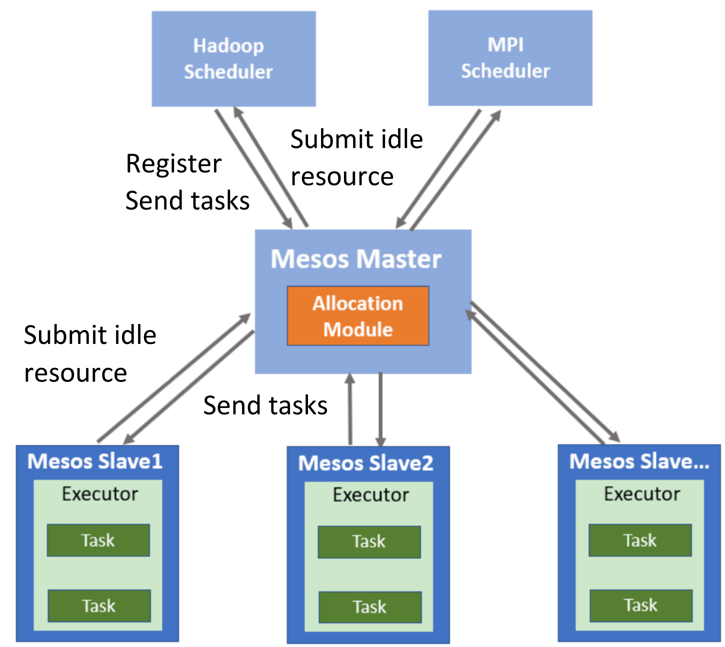
Example of Two-level Scheduling: Mesos

- Second-level Scheduling
  - Frameworks, such as Hadoop, Spark, MPI, and Marathon, run on Mesos and are "components" responsible for application management and scheduling.
  - Different frameworks are used to complete various tasks, such as batch tasks, real-time analysis tasks, etc.



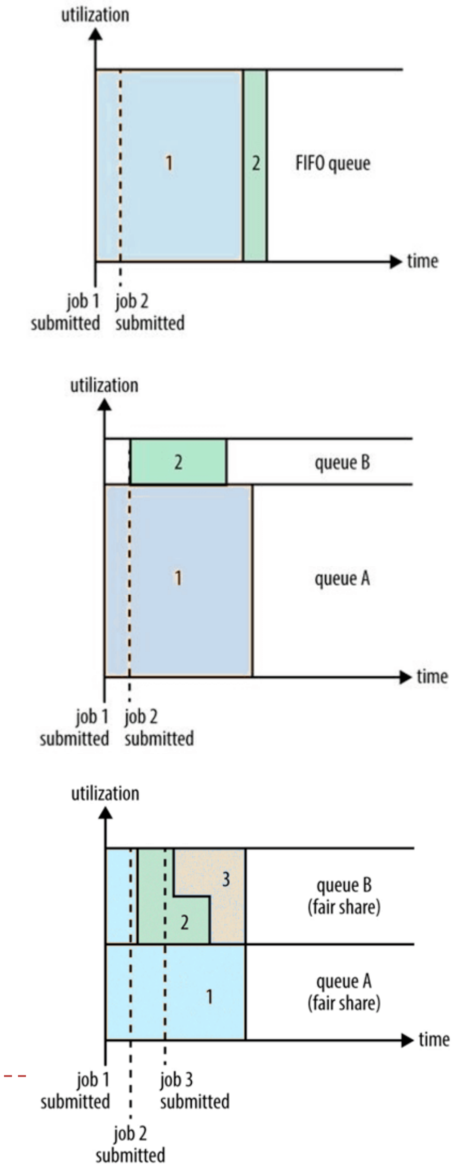
Example of Two-level Scheduling: Mesos

- Workflows in Mesos
  1. The framework registers with the Mesos Master;
  2. The Mesos Slave node reports the free resources of the node to the Mesos Master periodically;
  3. The Scheduler process on the Mesos Master collects the free resource information and sends it to the registered framework;
  4. After the framework's Scheduler receives the resources, it performs task scheduling and matching. If matches, the result is sent to the Mesos Master, and then forwarded to the corresponding node's executor to execute the task.



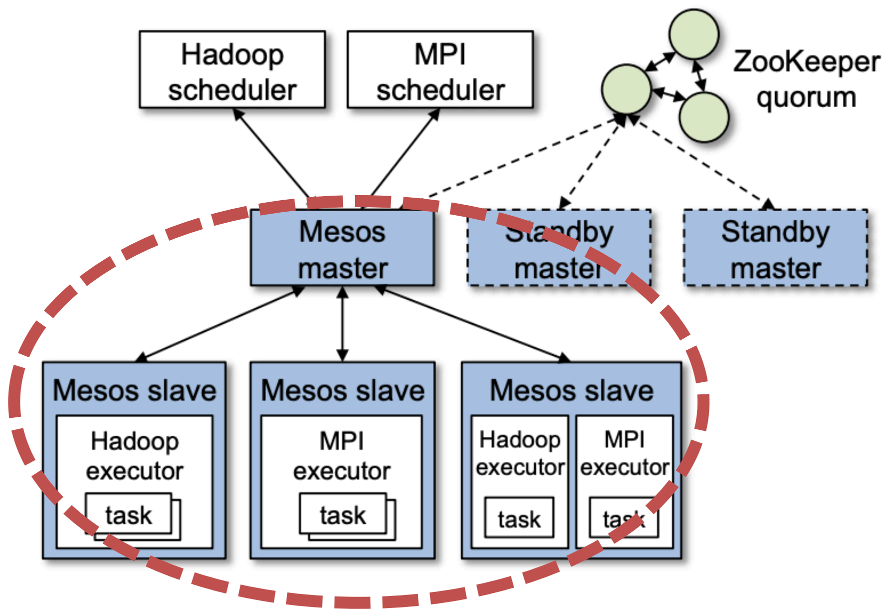
Example of Two-level Scheduling: Mesos

- Scheduling policy in frameworks
  - Hadoop: http://www.corejavaguru.com/bigdata/hadoop-tutorial/yarn-scheduler
    - FIFO
    - Capacity
    - Fair
    - …
  - Spark: https://databricks.com/session/apache-spark-scheduler



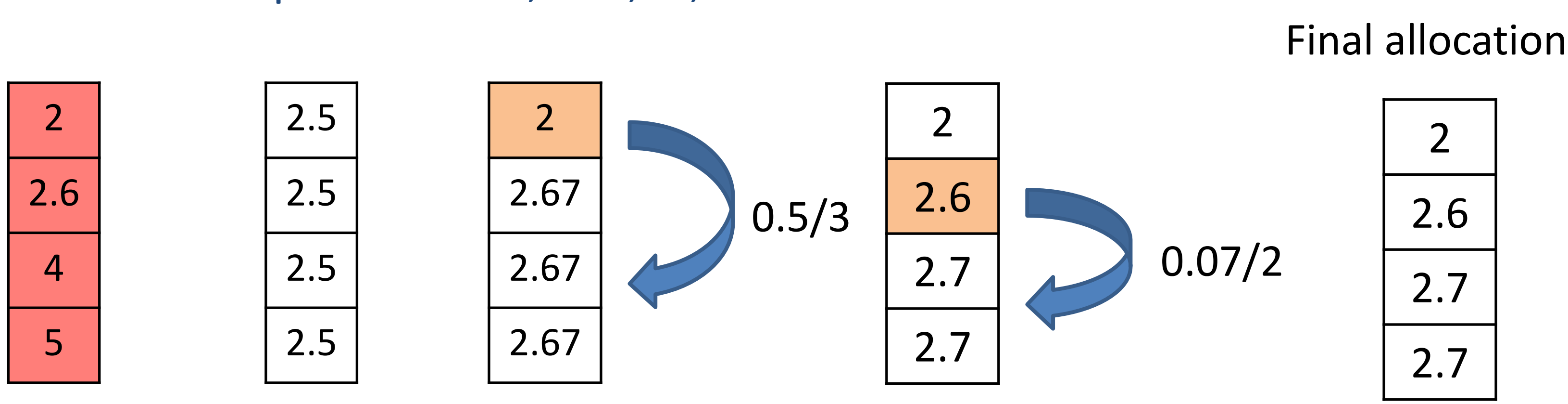
Example of Two-level Scheduling: Mesos

- Scheduling algorithm at the first-level:
  - Max-min Fairness, MMF
  - Dominant Resource Fairness, DRF



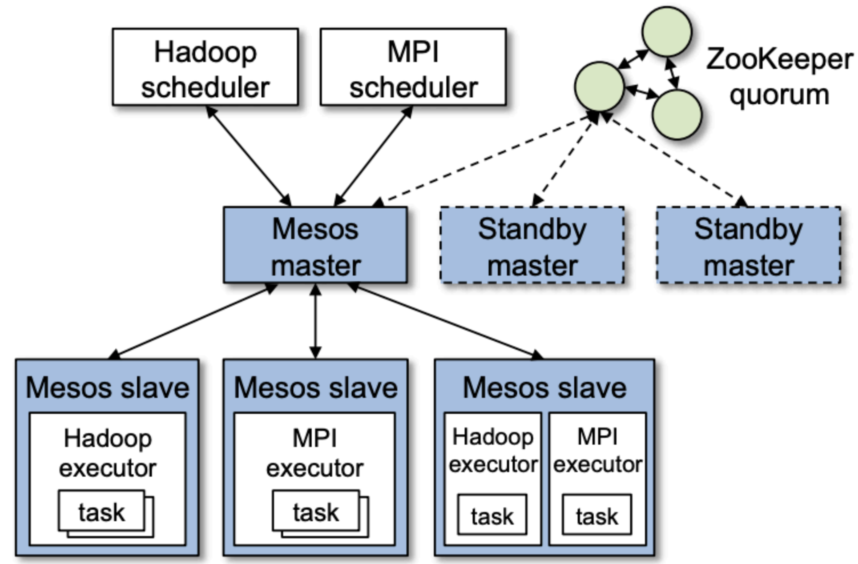
Example of Two-level Scheduling: Mesos

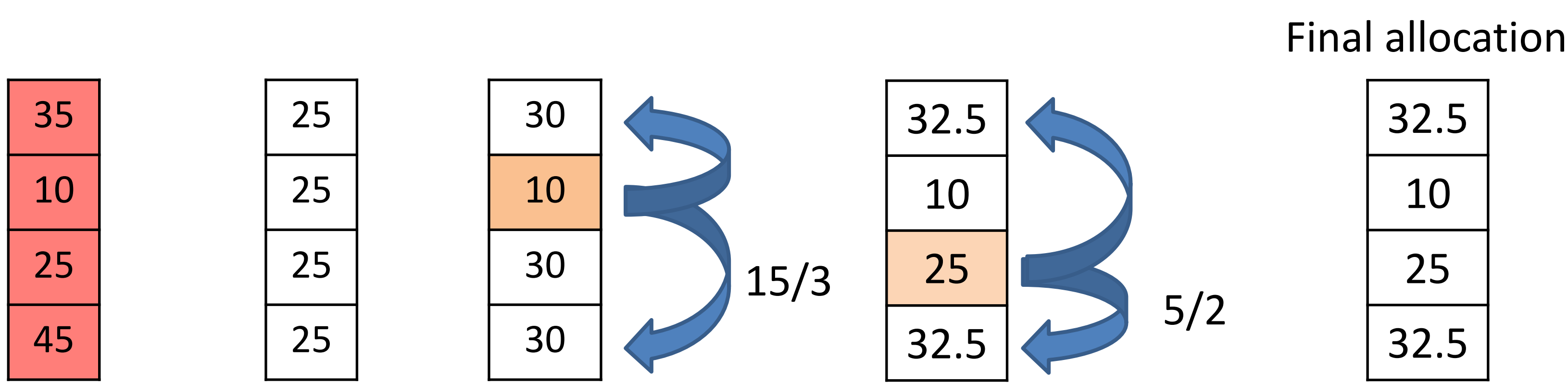
- Max-min Fairness algorithm:
  - Fairly share the minimum requirements that each user needs, and then allocate the unused resources evenly to users who need extra resources.
- Example
  - Total resource: 10
  - User number: 4
  - User requirement: 2, 2.6, 4, 5



Example of Two-level Scheduling: Mesos

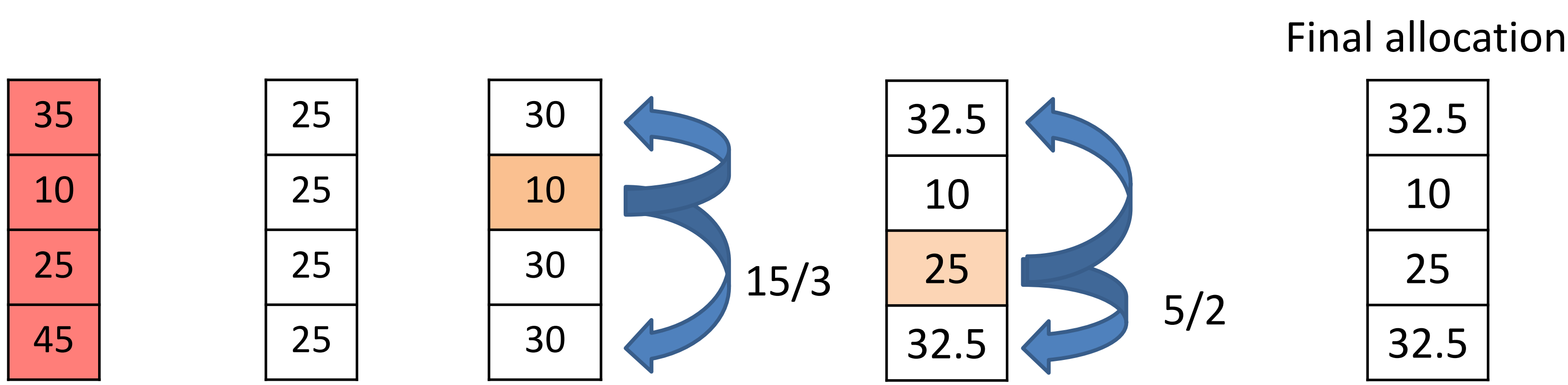
- Example of MMF
  - Total idle resource: 100
  - User number: 4
  - User requirement: 35, 10, 25, 45





Example of Two-level Scheduling: Mesos

- Problems of MMF
  - Allocating resources in an absolutely fair way will cause a lot of waste of resources
    - For example, user A and user D with user demand of 35 and 45 are all allocated 32.5 free resources, but because the resources do not meet the demand, both users cannot use and run tasks.



Example of Two-level Scheduling: Mesos

- Dominant Resource Fairness, DRF
  - Besides the fairness among users, it also considers the needs of users for different types of resources
  - Can satisfy as many users as possible

Example of Two-level Scheduling: Mesos

- Dominant Resource Fairness Case
  - Suppose we have 18 CPUs and 36GB memory
  - A: memory-bound task, <2 CPU, 8 GB>
  - B: CPU-bound task, <6 CPU, 2 GB>

- Step 1: Calculate resource allocation
  - Suppose the number of task A and B are x and y, respectively
  - A needed resources: <2x, 8x>
  - B needed resources: <6y, 2y>
  - Total resources needed: <2x+6y, 8x+2y>
  - Resource constrains: $$\begin{aligned} 2x + 6y &\le 18 \\ 8x + 2y &\le 36 \end{aligned}$$

Example of Two-level Scheduling: Mesos

- Dominant Resource Fairness Case
  - Suppose we have 18 CPUs and 36GB memory
  - A: memory-bound task, <2 CPU, 8 GB>
  - B: CPU-bound task, <6 CPU, 2 GB>

	CPU	Memory
A	2x/18	8y/36
B	6x/18	2y/36

- Step 2: Define the main resource limitation for each task
  - For each A, it needs 2/18 system CPU, 8/36 system memory, so memory limits A's number
  - For each B, it needs 6/18 system CPU, 2/36 system memory, so CPU limits B's number

Example of Two-level Scheduling: Mesos

- Dominant Resource Fairness Case
  - Suppose we have 18 CPUs and 36GB memory
  - A: memory-bound task, <2 CPU, 8 GB>
  - B: CPU-bound task, <6 CPU, 2 GB>

	CPU	Memory
A	2x/18	8x/36
B	6y/18	2y/36

- Step 3: balance the resource utilization of all tasks, maximum the resource usage which does not limit the task number
  - $$2x + 6y \le 18$$ $$8x + 2y \le 36$$ $$\frac{8x}{36} = \frac{6y}{18}$$ ⇨ X=3 Y=2 ⇨ A consumes 2/3 of memory resource B consumes 2/3 of CPU resource

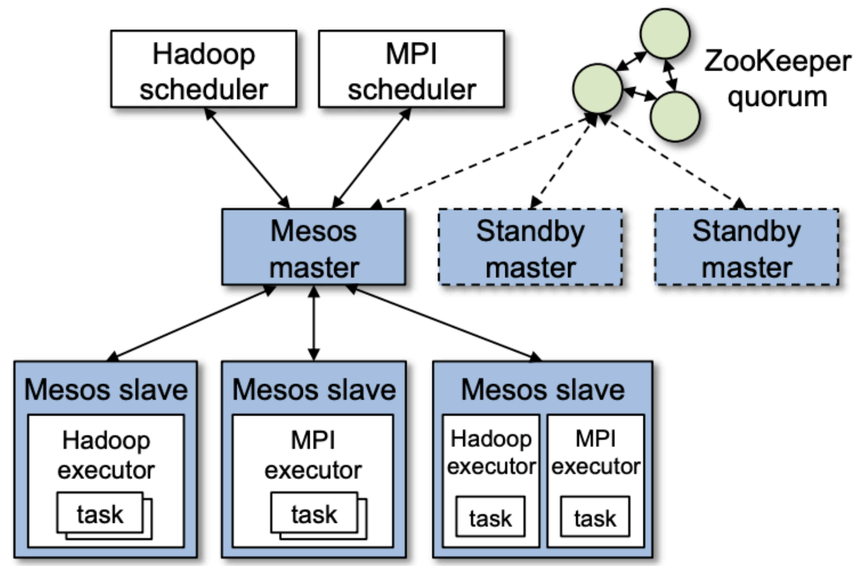
Example of Two-level Scheduling: Mesos

- Max-min Fairness algorithm:
  - Suitable for a single type of resource allocation scenario
  - Assign no more resources than required to each user

- Dominant Resource Fairness algorithm
  - Suitable for scenarios where multiple types of resources are mixed
  - Make the most use of resources so that as many tasks as possible can be performed

Two-level Scheduling

- Possible issues:
  - Decouple the task scheduling with the resource availability might make frameworks only know part of the resource information and cannot guarantee global optimal



Shared state scheduling -- Why

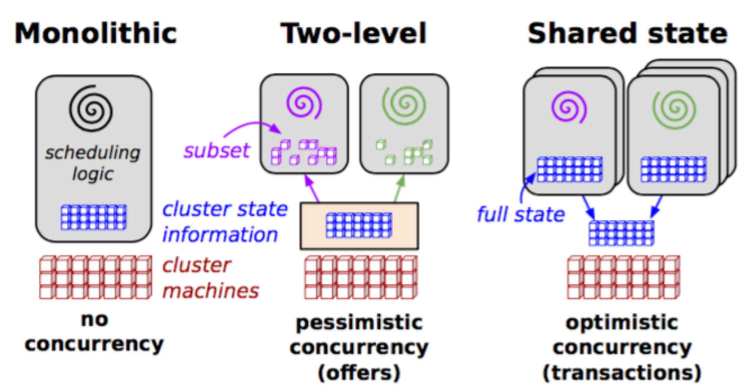
- Two types of objects to be managed in the cluster:

    1. Resource allocation and usage status;
    2. Task scheduling and execution status;

Monolithic scheduling manages the two at the same time:

+ Global resource management

- Poor scalability and single node bottleneck



Shared state scheduling -- Why

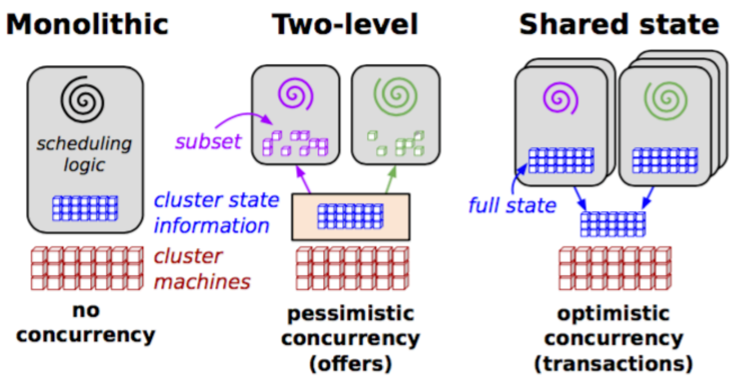
- Two types of objects to be managed in the cluster:

  1. Resource allocation and usage status;
  2. Task scheduling and execution status;

Two-level scheduling manages the two at different levels:

- Cannot provide optimal resource in global scope

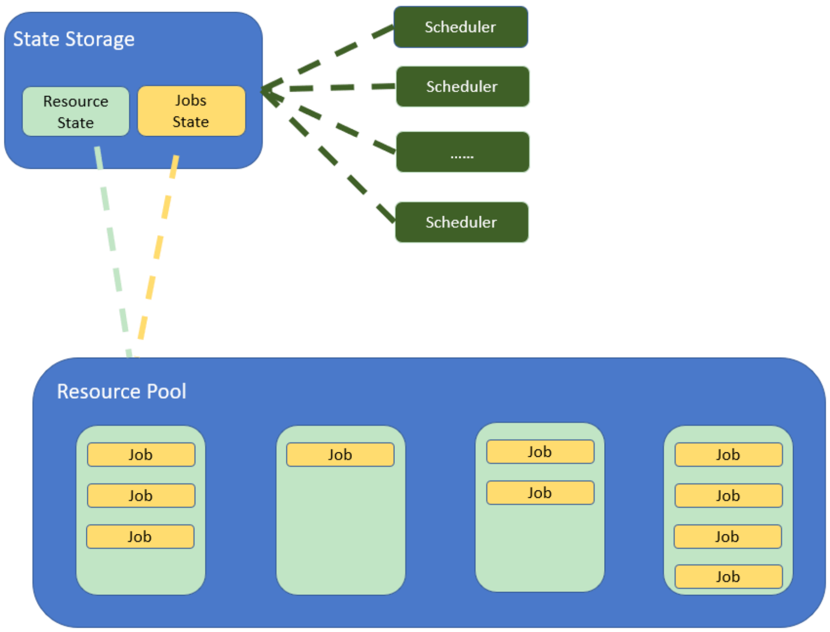
+ Good scalability



Shared state scheduling

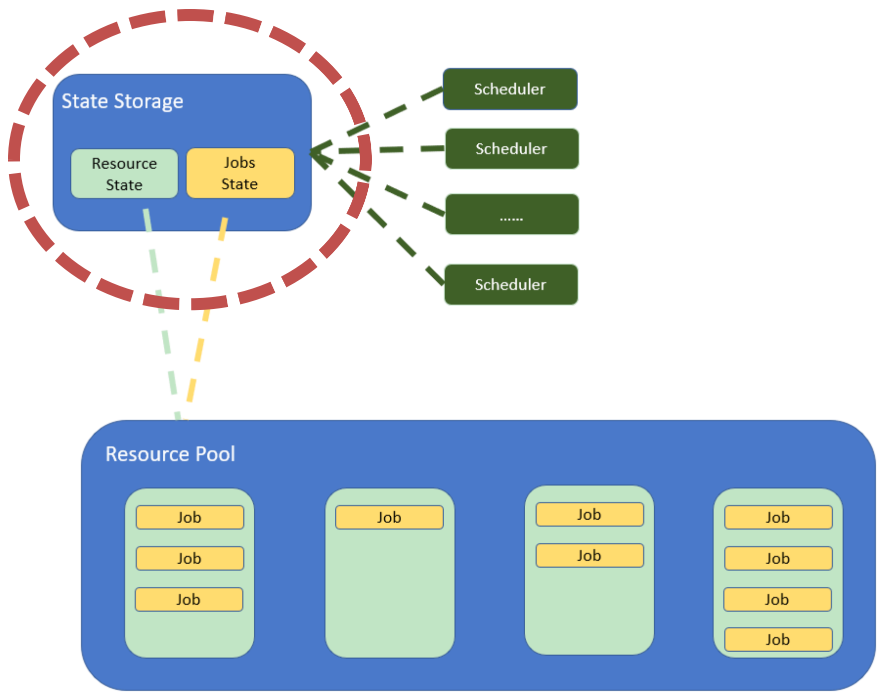
- Break down a single scheduler into multiple schedulers
  - --> Better scalability and no single node failure

- Each scheduler has global resource status information
  - --> Optimal task scheduling



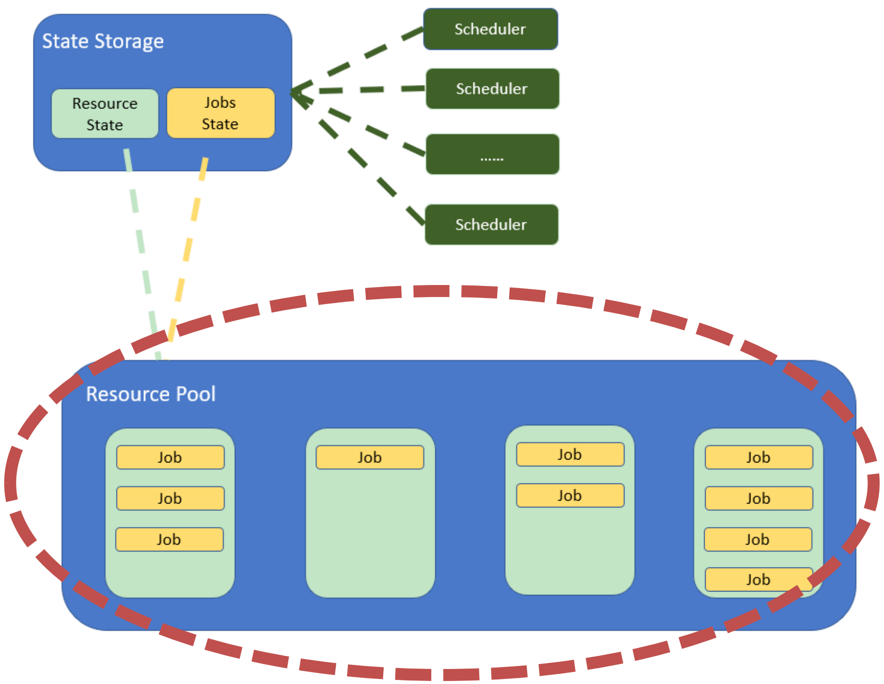
Architecture of Shared State Scheduling

- State storage:
  - Responsible for storing and maintaining resource and task state, so Schedulers can query available resource when scheduling tasks



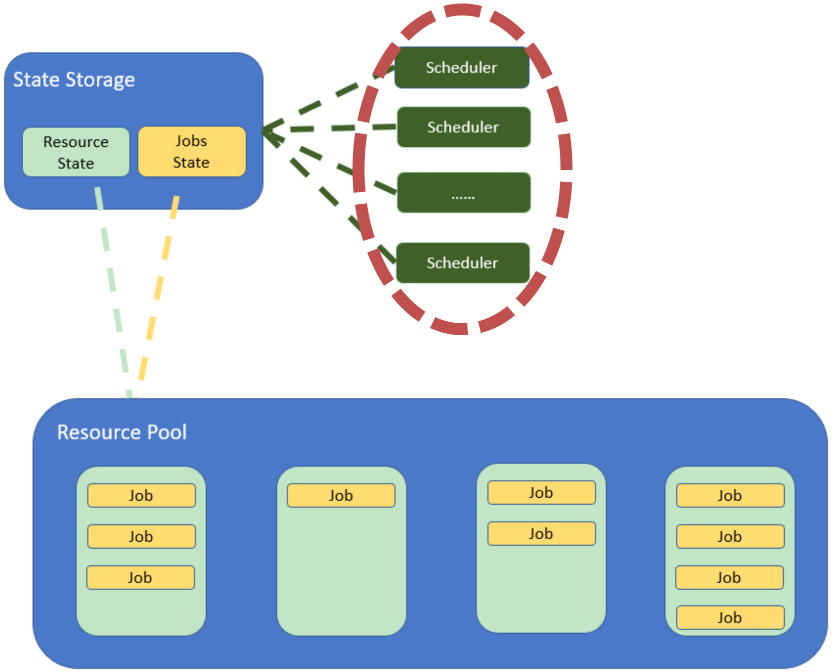
Architecture of Shared State Scheduling

- Resource Pool:
  - Multiple node clusters that receive and execute tasks scheduled by the Scheduler; send state of resources and job execution to the Storage



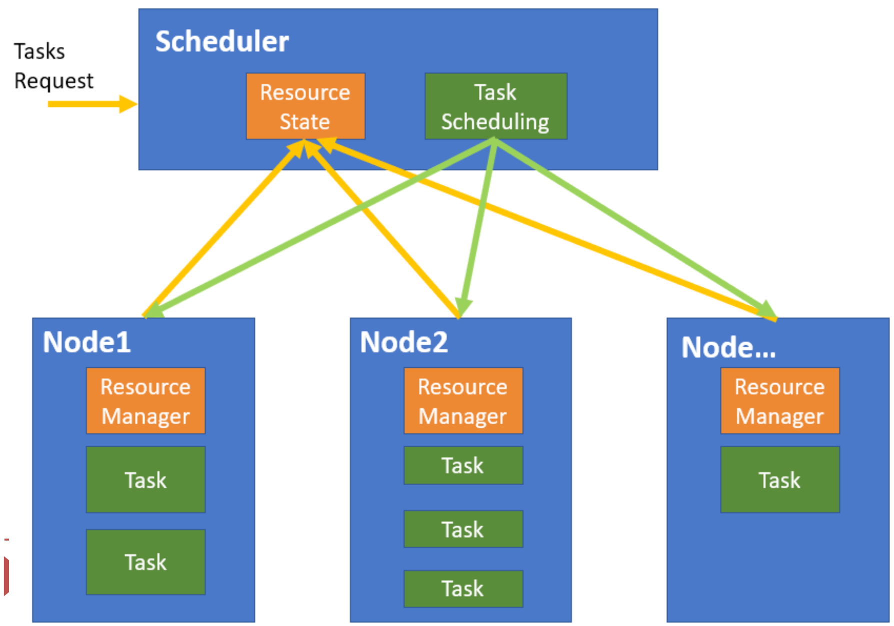
Architecture of Shared State Scheduling

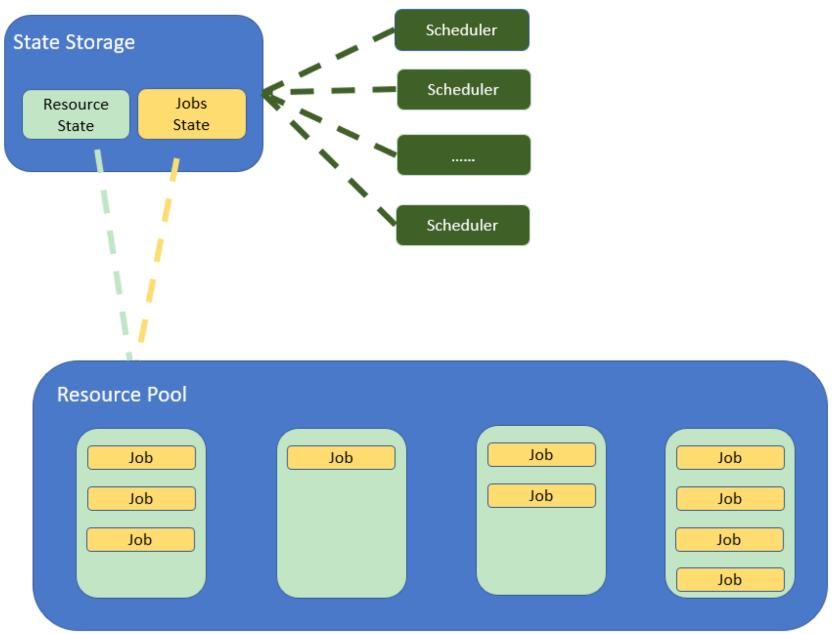
- Scheduler:
  - Just for concrete task scheduling operations



Monolithic vs. Shared State Scheduling

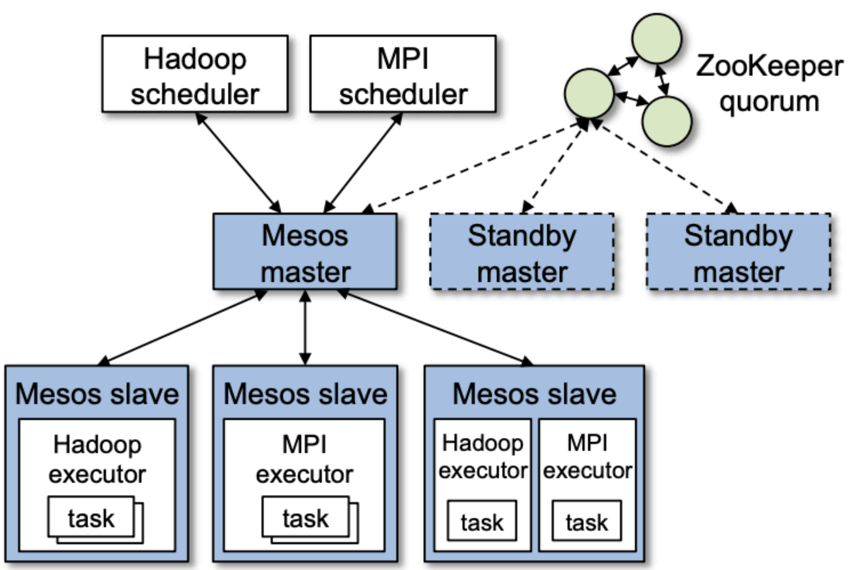
- Different from the monolithic scheduling, the Scheduler in shared state scheduling does not need to manage the cluster resources

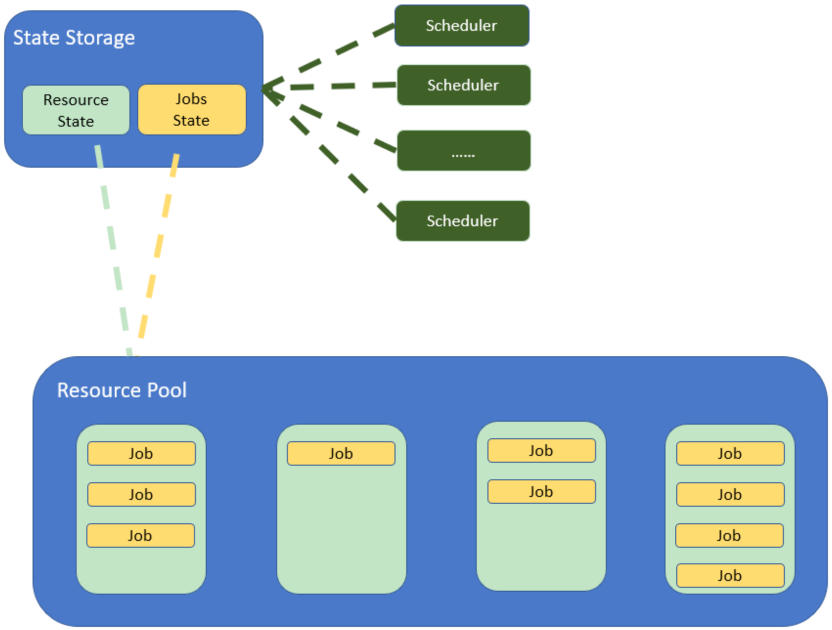




Two-level vs. Shared State Scheduling

- All schedulers in shared state scheduling can access the resource information directly and make optimal scheduling on a global view





Example of Shared State Scheduling

- Google Omega: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/41684.pdf

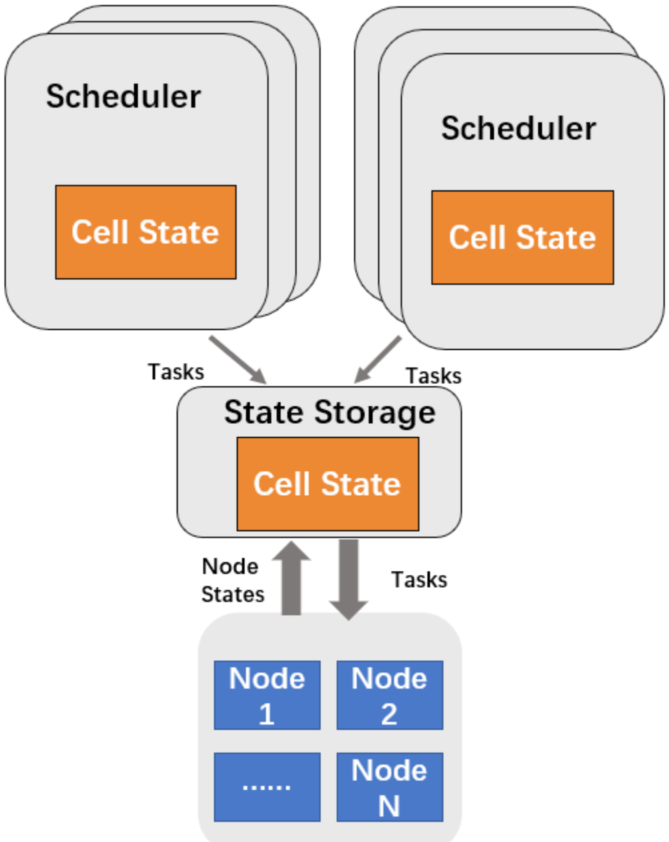
- Microsoft Apollo: https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-boutin_0.pdf

- Hashicorp Nomad: https://www.nomadproject.io/

Example of Shared State Scheduling: Omega

- Cell: a group of servers
  - A cluster can have multiple cells
- State storage: store and maintain the states of resources and tasks
- Cell State: records the global cluster states
- Every scheduler has one copy of the Cell State → sharing the state of the cluster

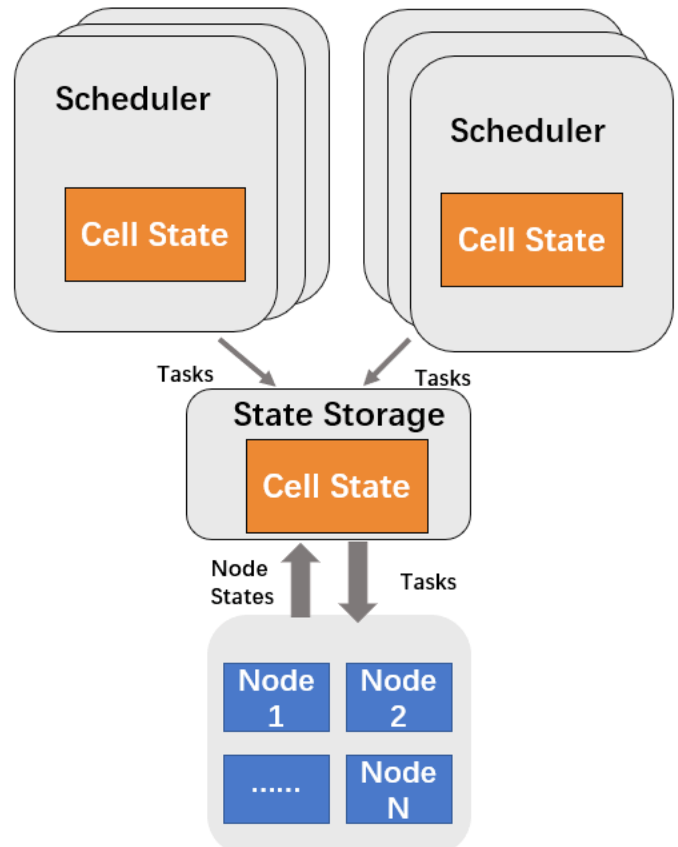
Omega architecture



Example of Shared State Scheduling: Omega

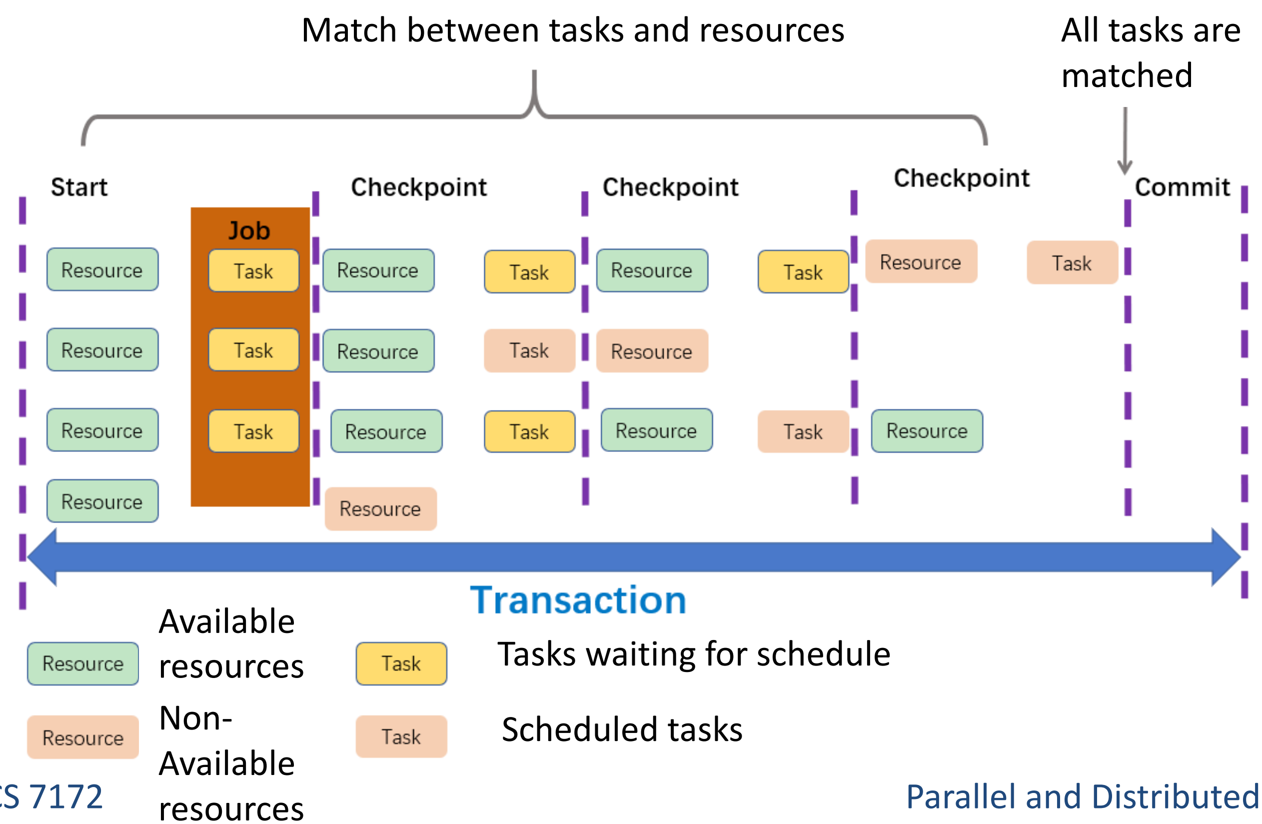
- What will happen when two scheduler try to use the same resource?
  - Omega adopts transaction model in database to maintain the consistency and atomicity for the same resource
  - In database, only one query can update one record at one time

Omega architecture



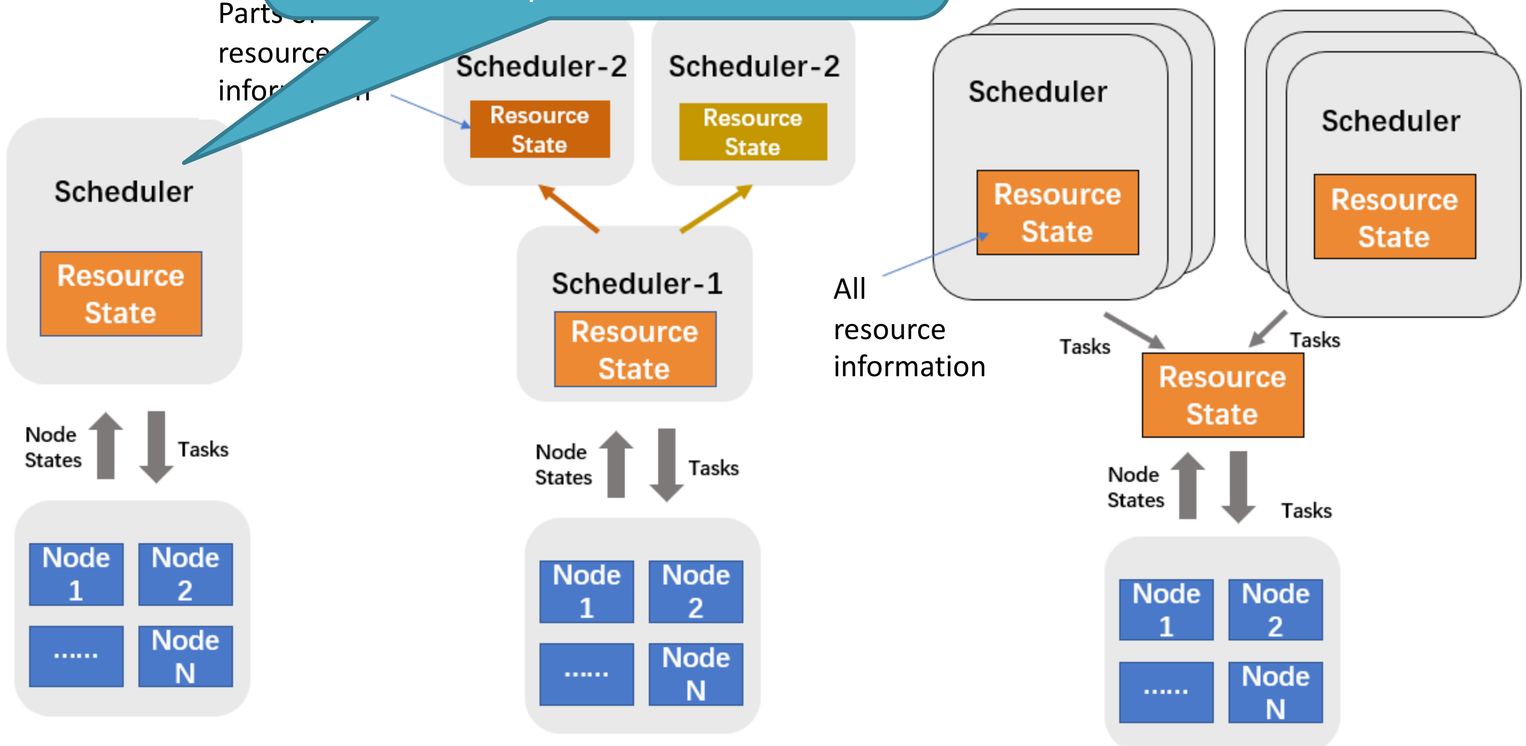
Example of Shared State Scheduling: Omega

- The scheduler tries to match the tasks and resources
- Scheduler will check whether the resources have been used during the checkpoints



Compar

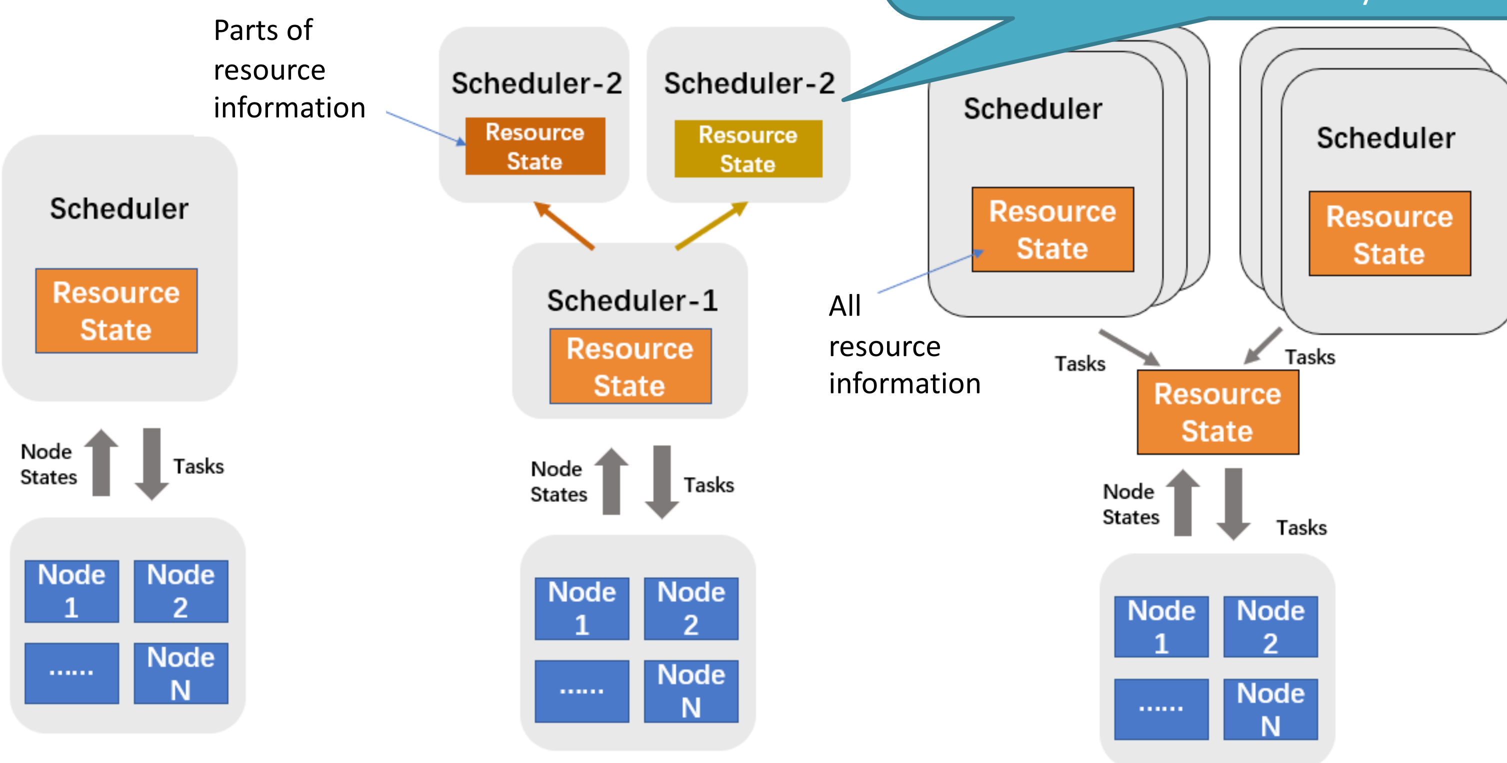
A central scheduler manages the resource information and task scheduling of the entire cluster. All tasks can only be scheduled by the central scheduler



- Adv: single point bottleneck
- Disadv: No scheduling concurrency, single point bottleneck problem, only suitable for small-scale clusters

Comparison

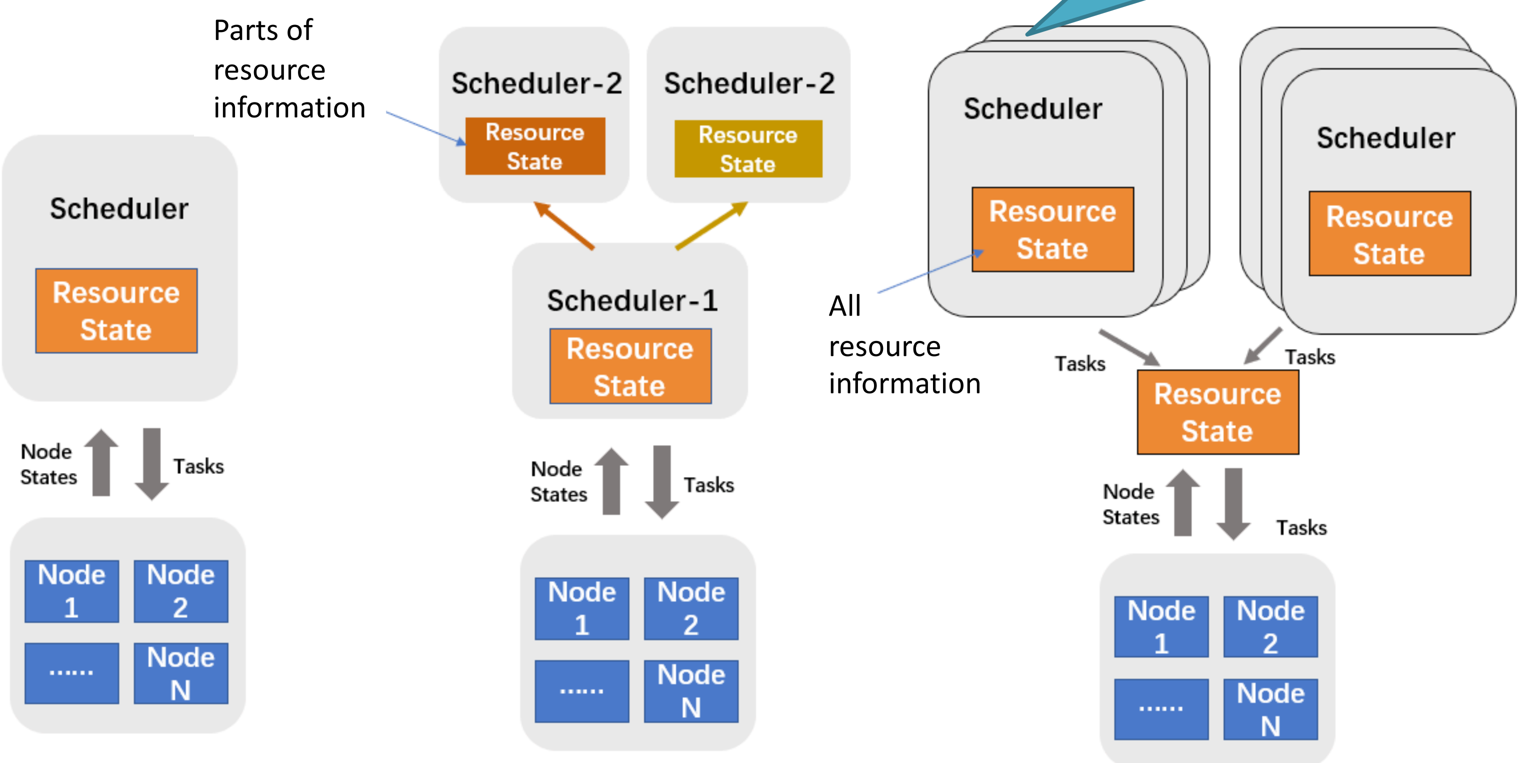
The resource management and task scheduling are divided into two layers for scheduling. The first layer scheduler is responsible for cluster resource management and sends the available resources to the second layer scheduling.



- Adv: no single point bottleneck and higher scalability
- Disadv: 2rd level scheduler only has parts of the resource information and cannot provide

Comparison

Multiple schedulers, each scheduler can see the global resource information of the cluster through state sharing



- Adv: Each scheduler can obtain global resource information in the cluster, so the task matching algorithm can achieve global optimality
- Disadv: multiple scheduler might cause resource competition and conflict

Conclusion

	Monolithic scheduling	Two-level scheduling	Sharing state scheduling
Architecture	Centralized, one node	Two schedulers	Distributed, multiple schedulers
Unit	Task	Task	Task
Parallel scheduling	No	Yes	Yes
Global optimal	Yes	No	Yes
Performance	Sharing state scheduling > Two-level scheduling > Monolithic scheduling		
Scalability	Sharing state scheduling > Two-level scheduling > Monolithic scheduling		
Suitable scenario	Small cluster	Medium cluster	Large scale cluster
Example	Borg	Mesos, YARN	Omega