
# CPSC 436C
# Cloud Computing for Data Science

## Resource Management

Maryam R.Aliabadi

mraiyata@cs.ubc.ca

Spring 2024

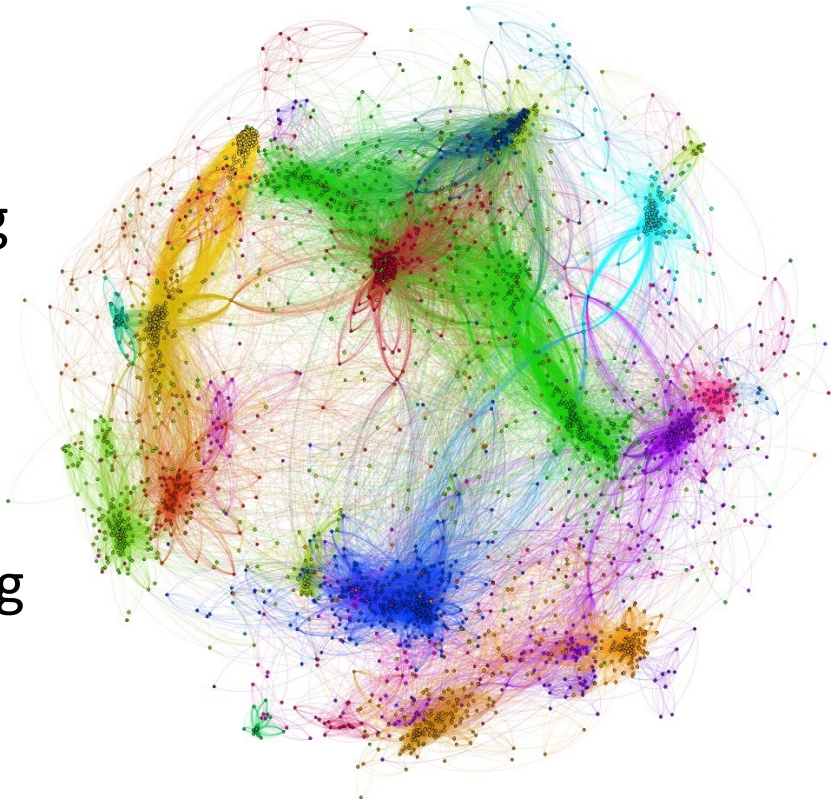
# Last Class's Review

- Think like a vertex
  - Pregel:
  - GraphLab
  - PowerGraph

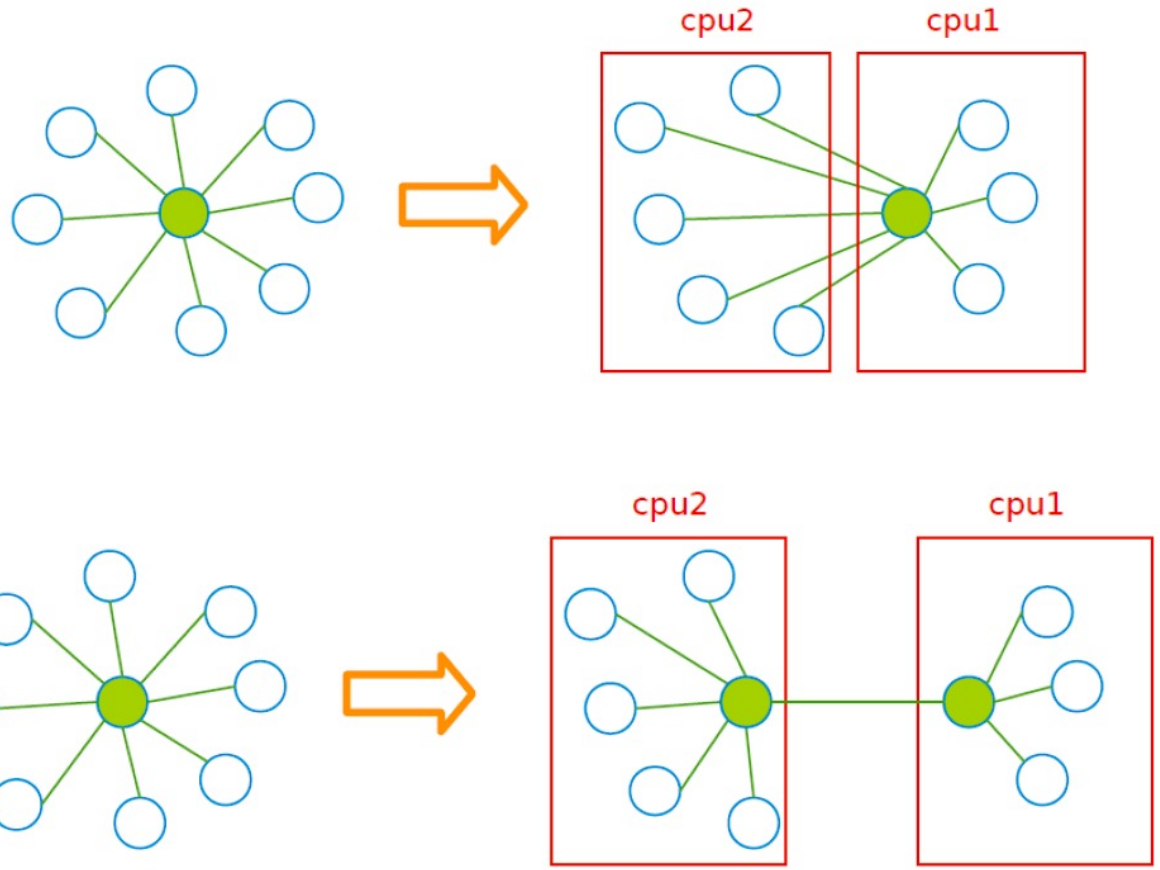
- Think like a table
  - Graphx: unifies data-parallel and graph-parallel systems.

# Large Graphs Challenges

- A large graph either cannot fit into memory of single computer or it fits with huge cost.
  - Difficult to extract parallelism based on partitioning of the data.
- Large graphs need large-scale processing.
  - Difficult to express parallelism based on partitioning of computation.

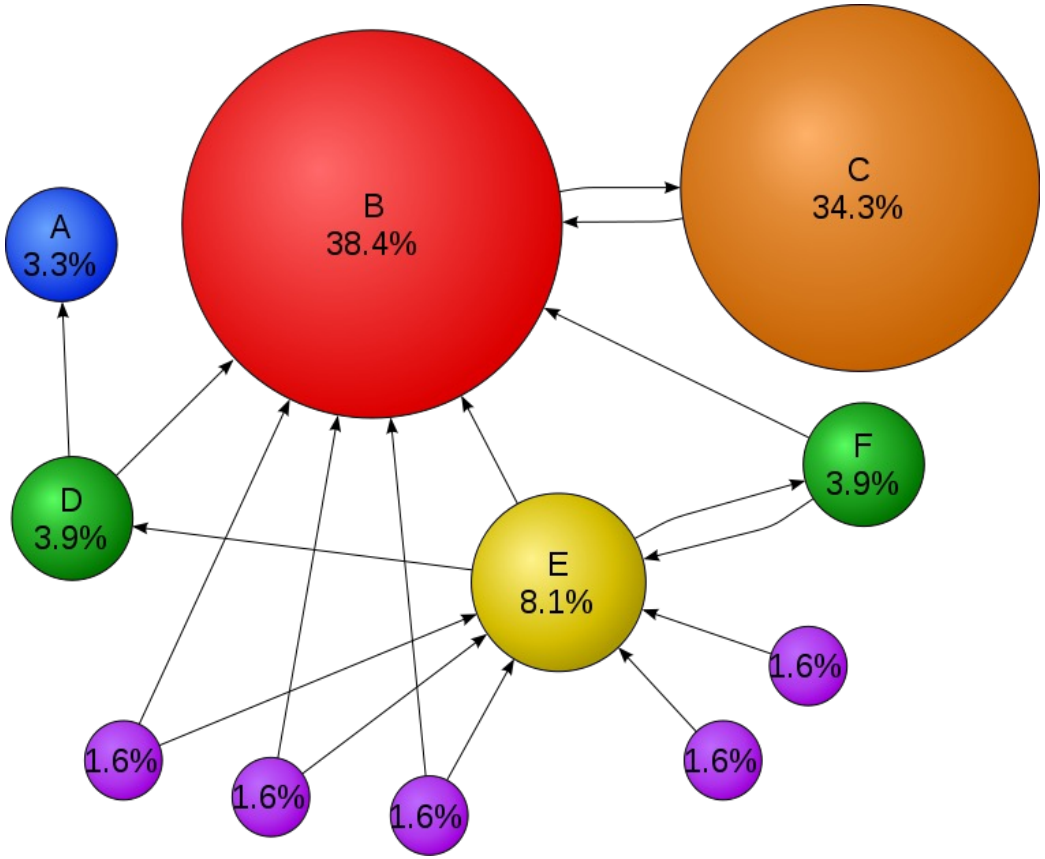


# Edge-Cut Vs. Vertex-Cut Graph Partitioning

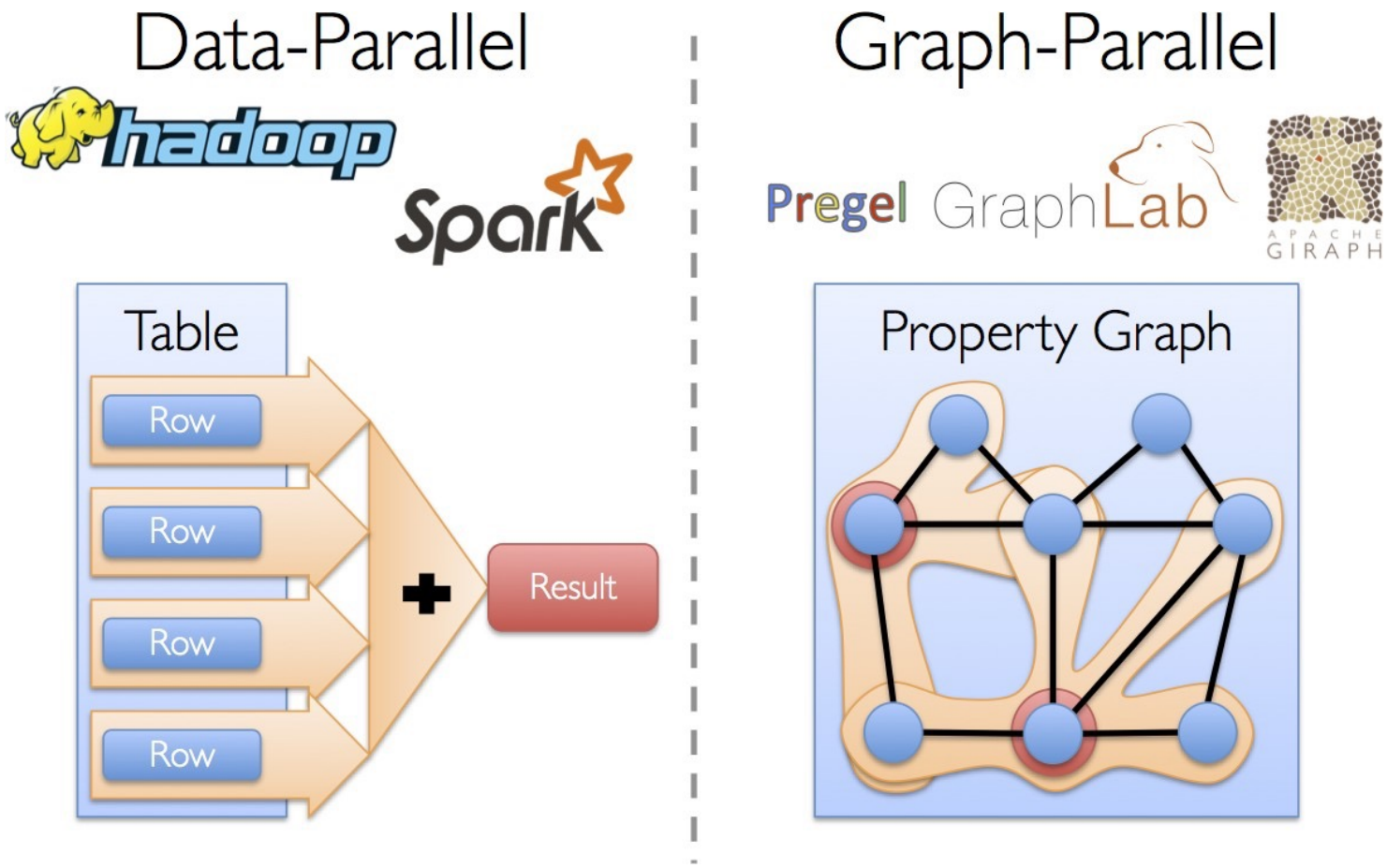


# PageRank

$$R[i] = \sum_{j \in Nbrs(i)} w_{ji} R[j]$$



# Think Like A Vertex



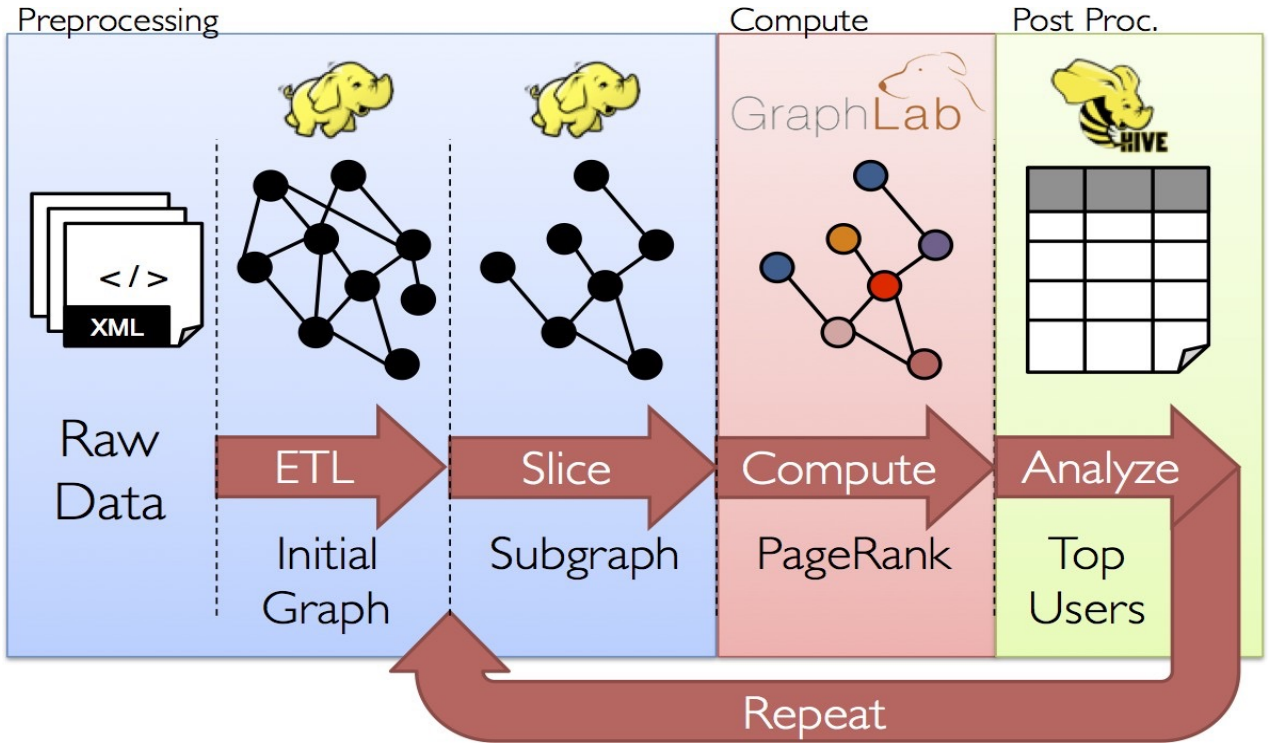
# Data-Parallel vs. Graph-Parallel Computation

- Data-parallel computation
  - Record-centric view of data.
  - Parallelism: processing independent data on separate resources.

- Graph-parallel computation
  - Vertex-centric view of graphs.
  - Parallelism: partitioning graph (dependent) data across processing resources and resolving dependencies (along edges) through iterative computation and communication.

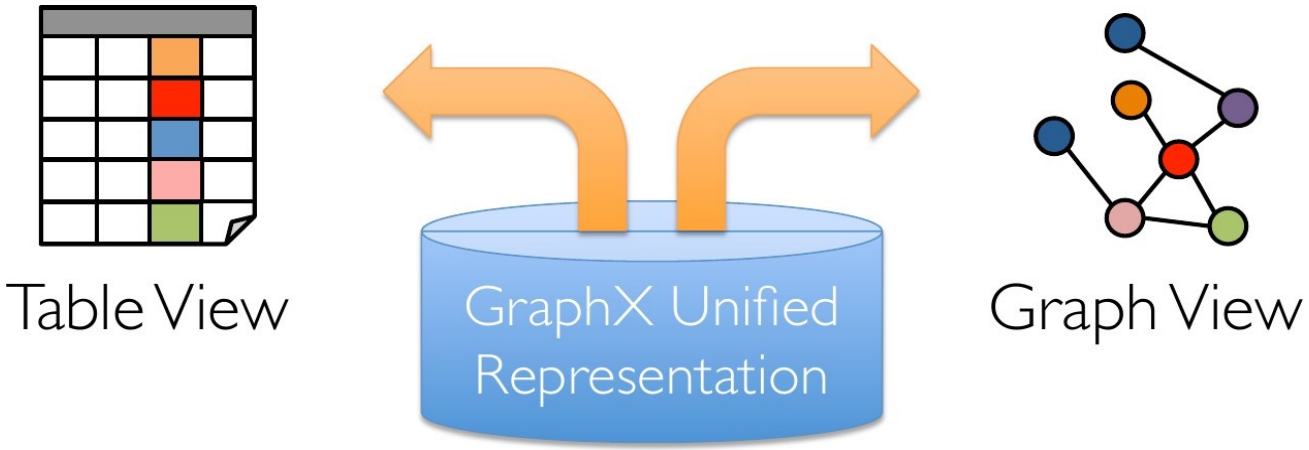
# Data-Parallel vs. Graph-Parallel Computation

- Graph-parallel computation: restricting the types of computation to achieve performance.



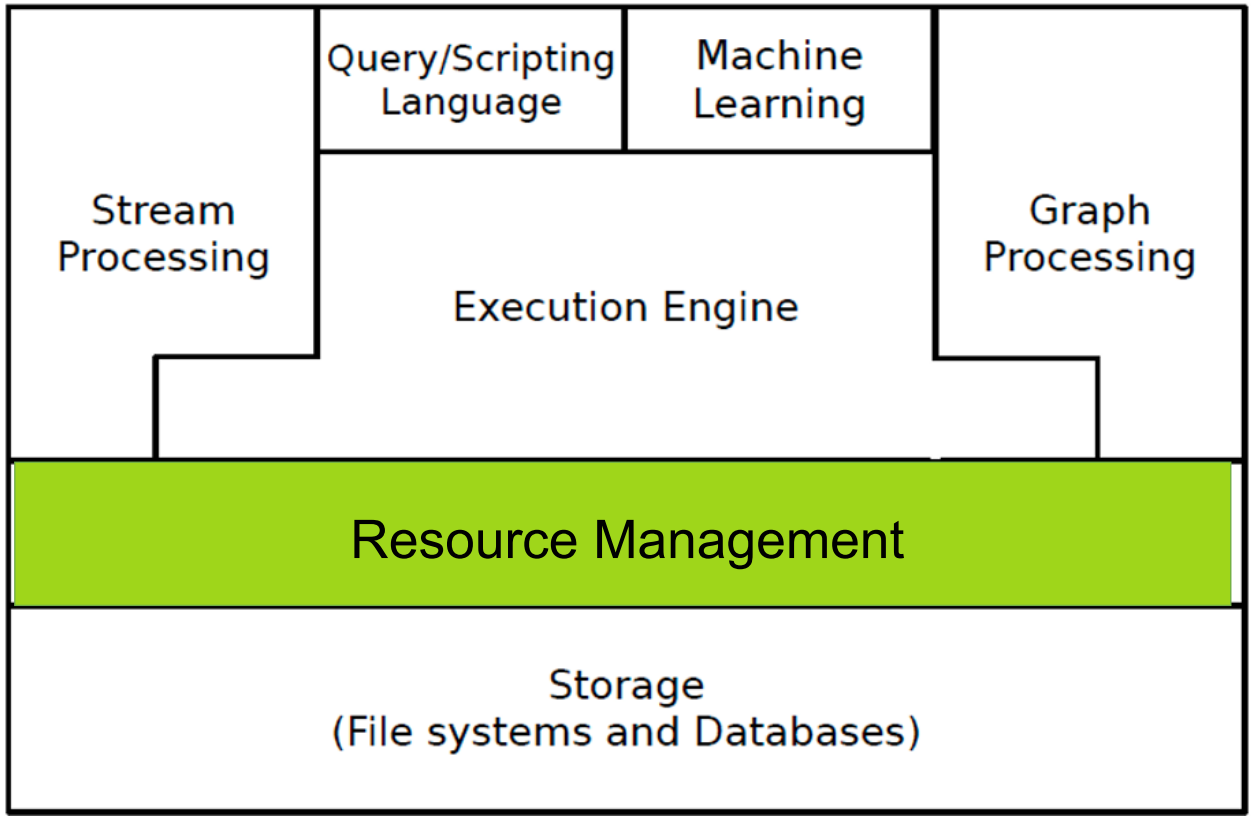
# Think like a Table

- Unifying Data-Parallel and Graph-Parallel Analytics
- Tables and Graphs are composable views of the same physical data.
- Each view has its own operators that exploit the semantics of the view to achieve efficient execution.



# Today's Topics

- Resource Management



# Motivation

- Rapid innovation in cloud computing.
- No single framework optimal for all applications.
- Running each framework on its dedicated cluster:
  - Expensive
  - Hard to share data

# Proposed Solution

- Running multiple frameworks on a single cluster.
- Maximize utilization and share data between frameworks.
- Three resource management systems:
  - Mesos
  - YARN
  - Borg

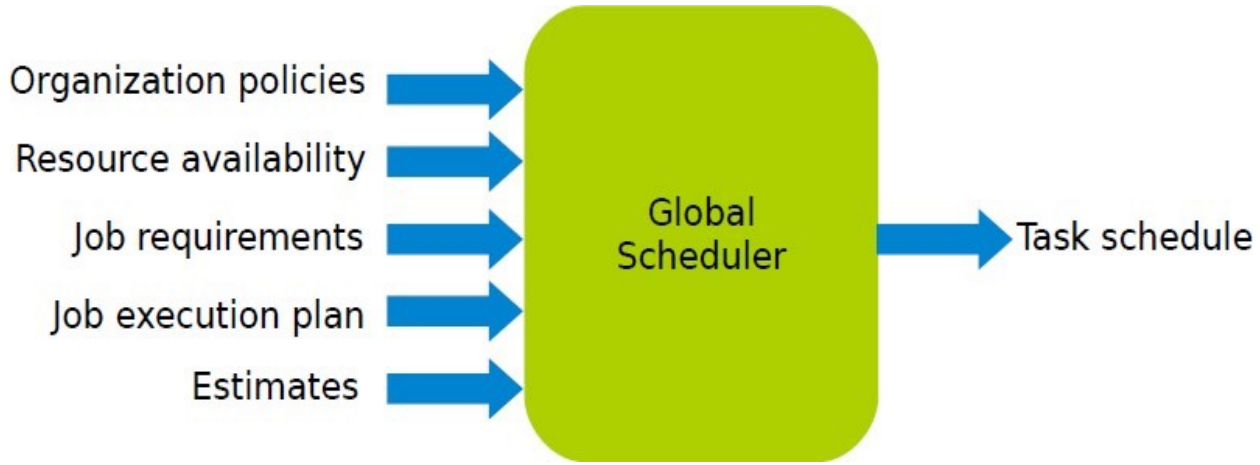
# Scheduler Frameworks

- Global scheduler

- Distributed scheduler

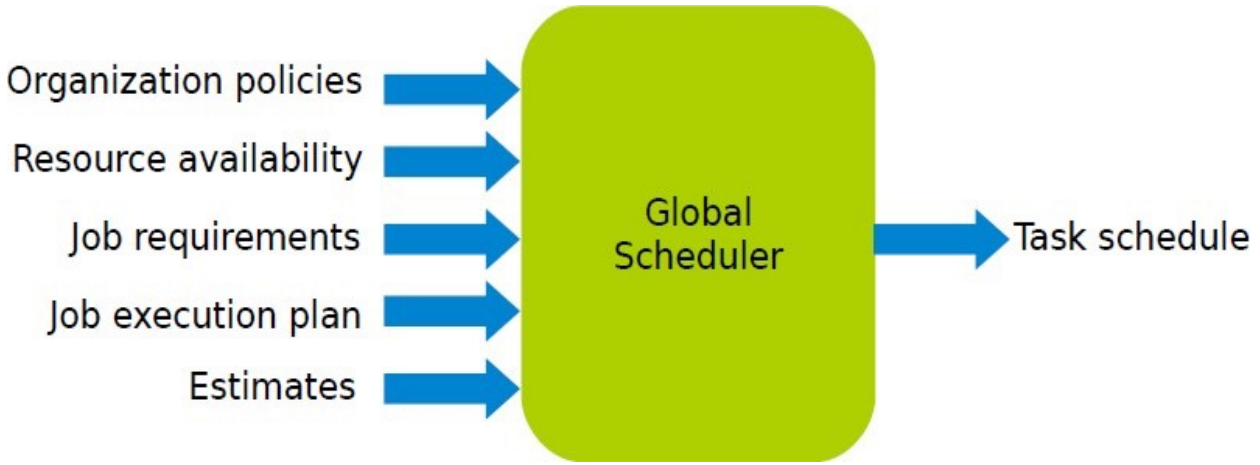
# Global Scheduler

- Job requirements
  - Response time
  - Throughput
  - Availability



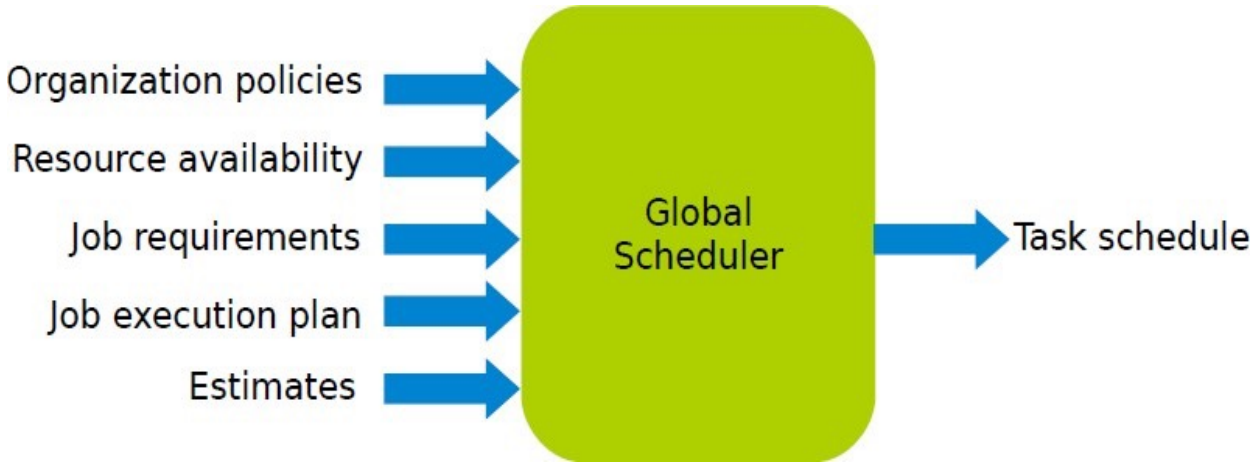
# Global Scheduler

- Job requirements
  - Response time
  - Throughput
  - Availability
- Job execution plan
  - Task DAG
  - Inputs/outputs



# Global Scheduler

- Job requirements
  - Response time
  - Throughput
  - Availability
- Job execution plan
  - Task DAG
  - Inputs/outputs
- Estimates
  - Task duration
  - Input sizes
  - Transfer sizes

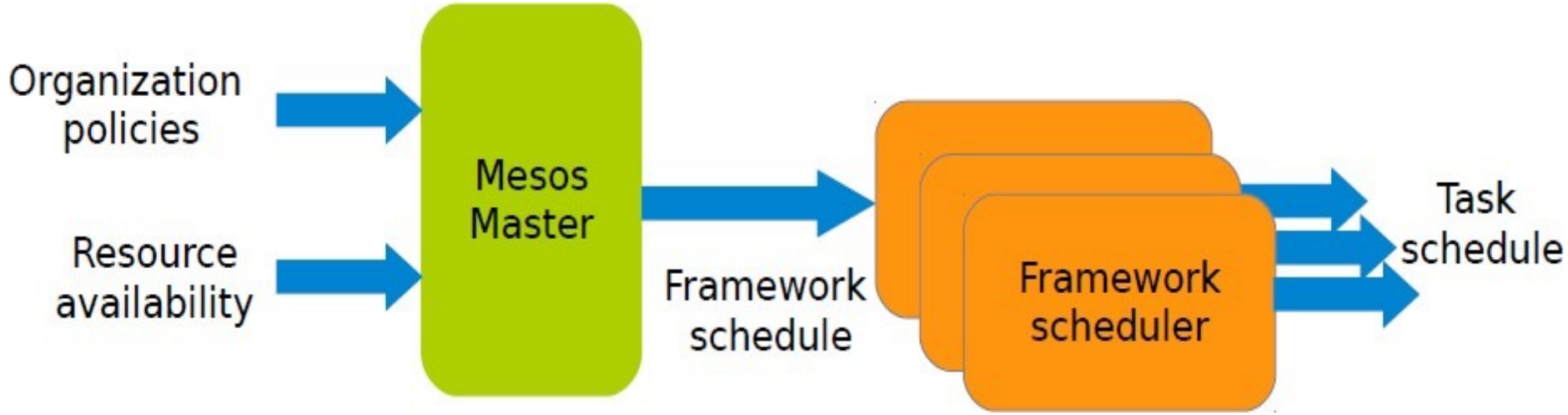


# Global Scheduler

- Advantages
  - Can achieve optimal schedule.

- Disadvantages
  - Complexity: hard to scale and ensure resilience.
  - Hard to anticipate future frameworks requirements.
  - Need to refactor existing frameworks.

# Distributed Scheduler



# Distributed Scheduler

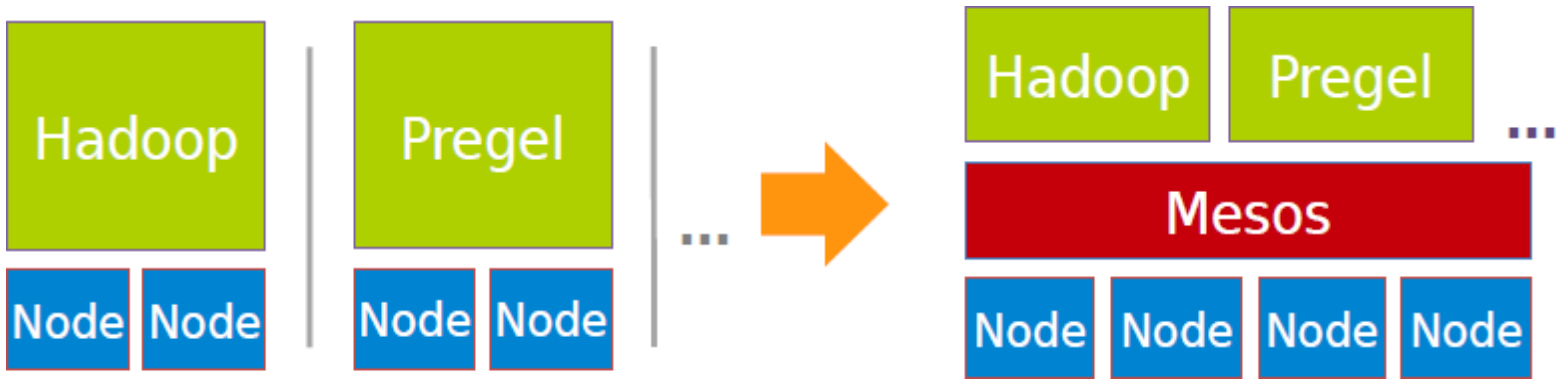
- Advantages
  - Simple: easier to scale and make resilient.
  - Easy to port existing frameworks, support new ones.

- Disadvantages
  - Distributed scheduling decision: not optimal.
    - Limited information
    - Resource conflicts

# Mesos

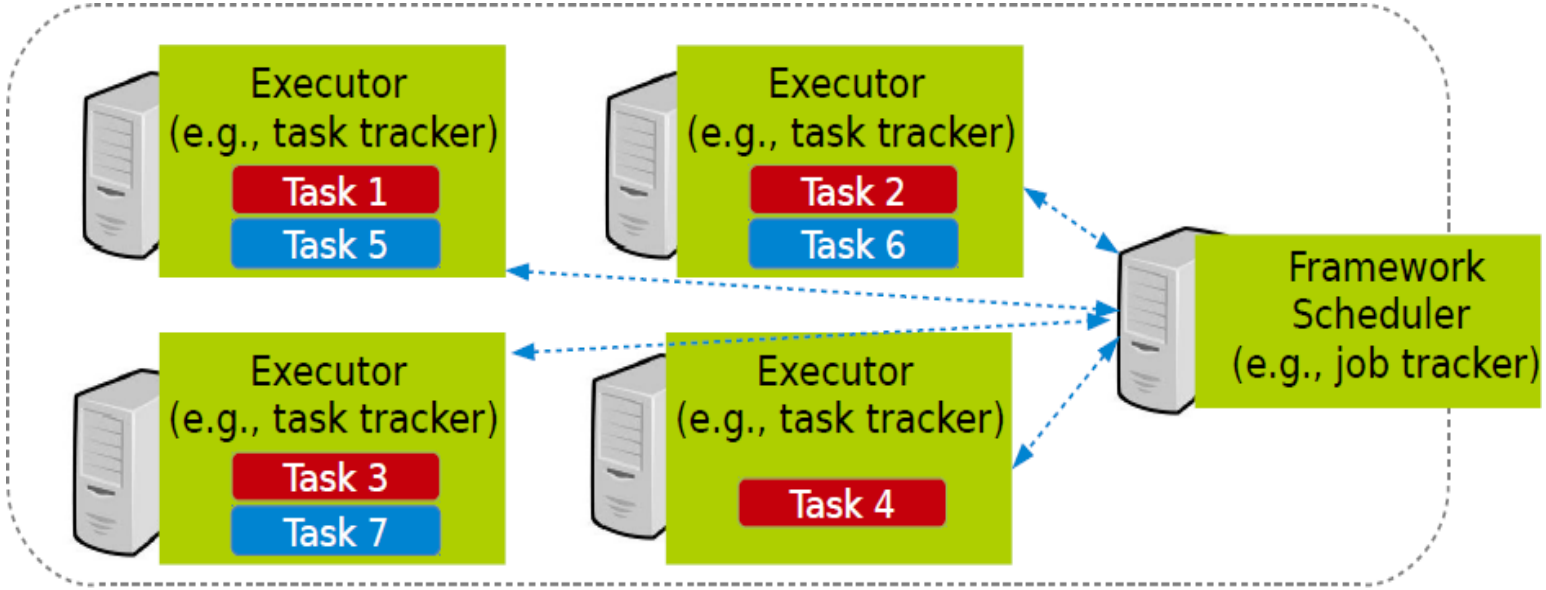
# Mesos

- Mesos is a common resource sharing layer, over which diverse frameworks can run.



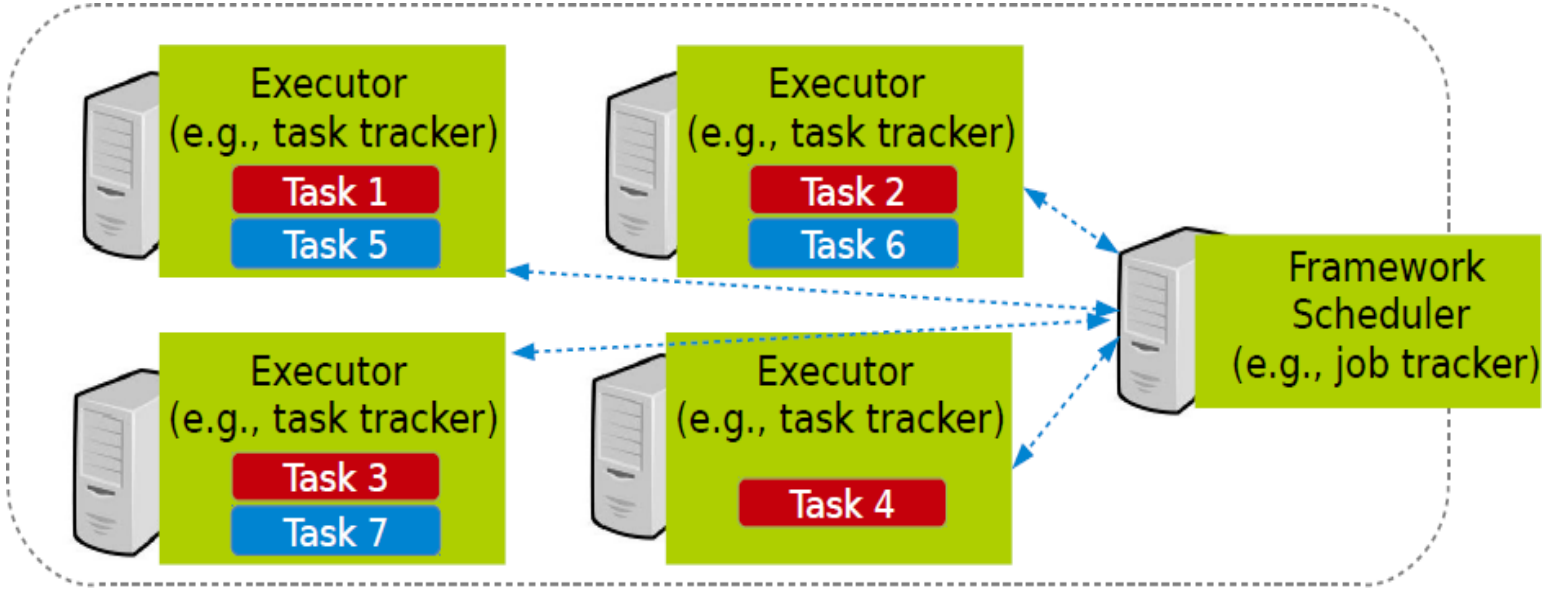
# Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.



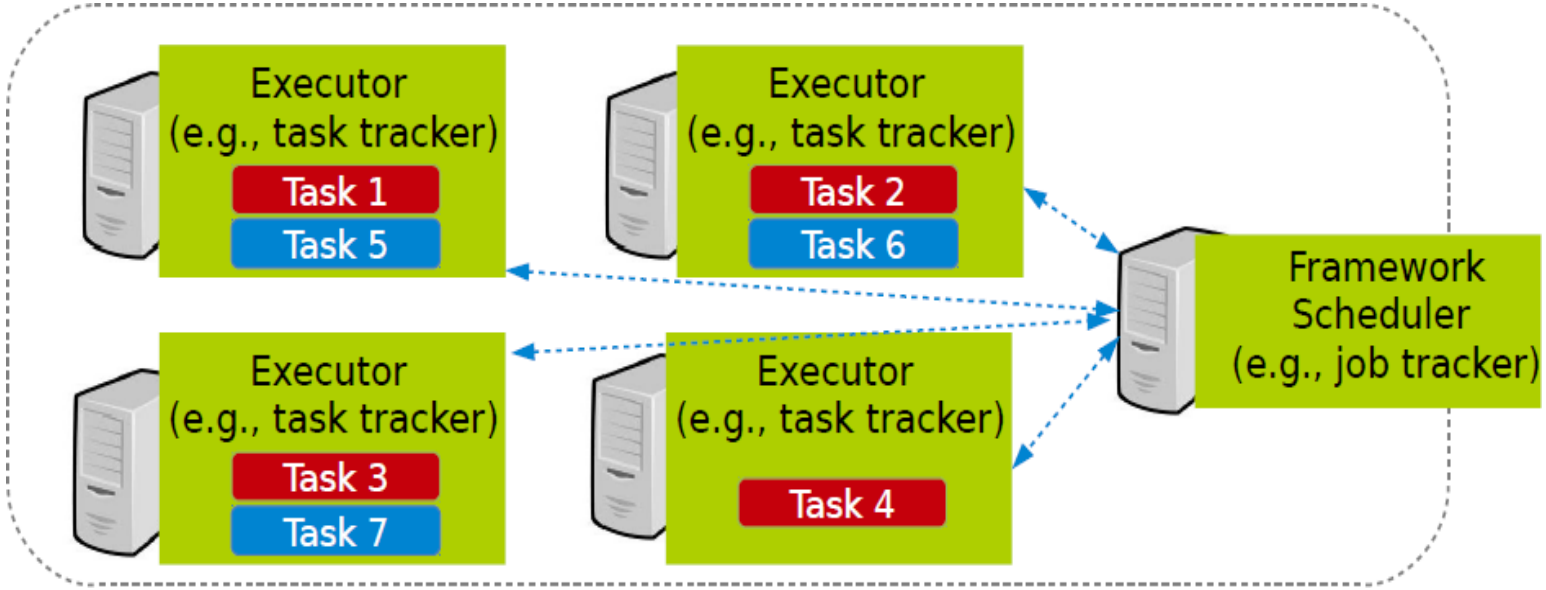
# Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.
- A job consists of one or more tasks.



# Computation Model

- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.
- A job consists of one or more tasks.
- A task (e.g., map, reduce) consists of one or more processes running on same machine.

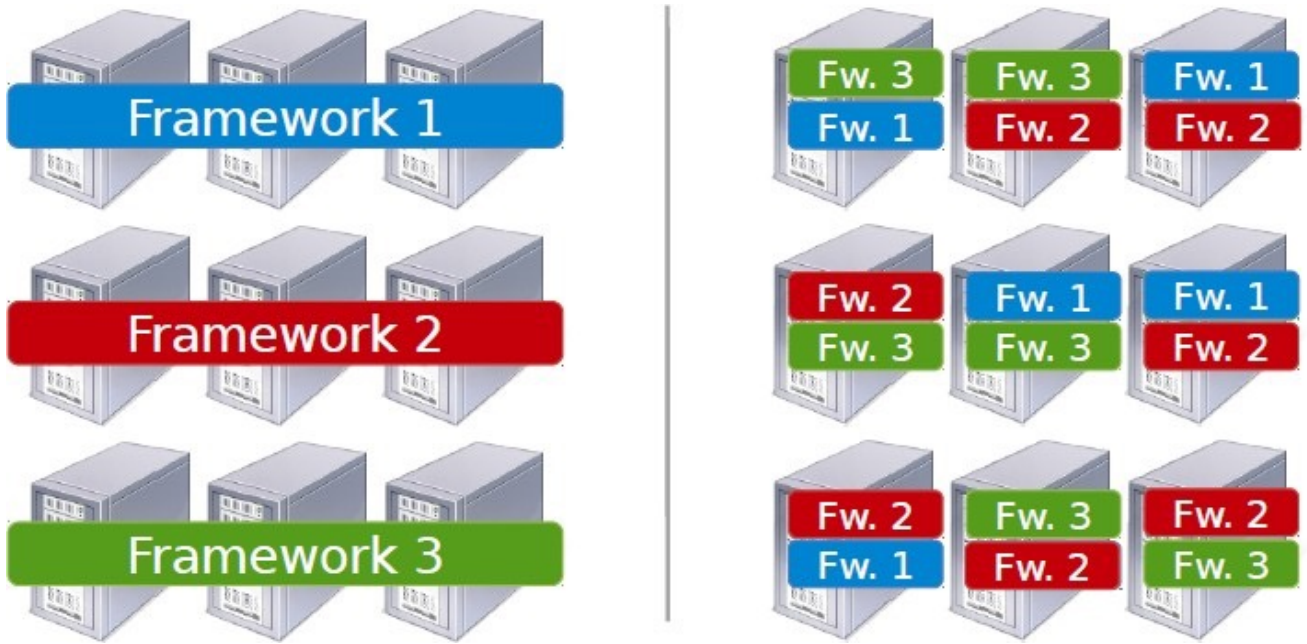


# Mesos design elements

- Fine-grained sharing
- Resource offers

# Fine-grained sharing

- Allocation at the level of tasks within a job.
- Improves utilization, latency, and data locality.

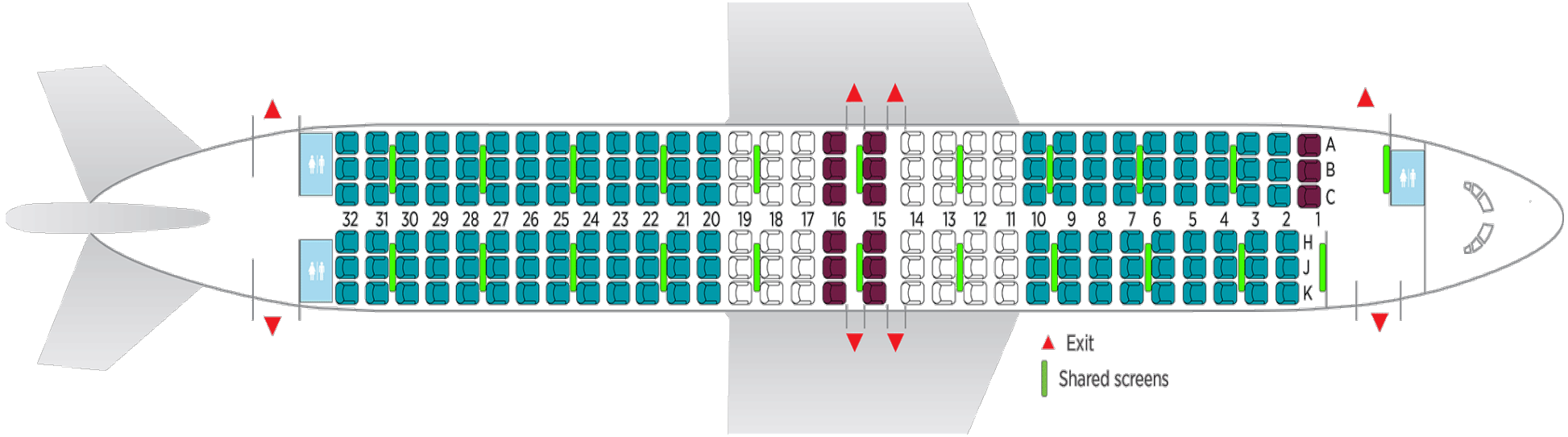



Coarse-grained sharing

Fine-grained sharing

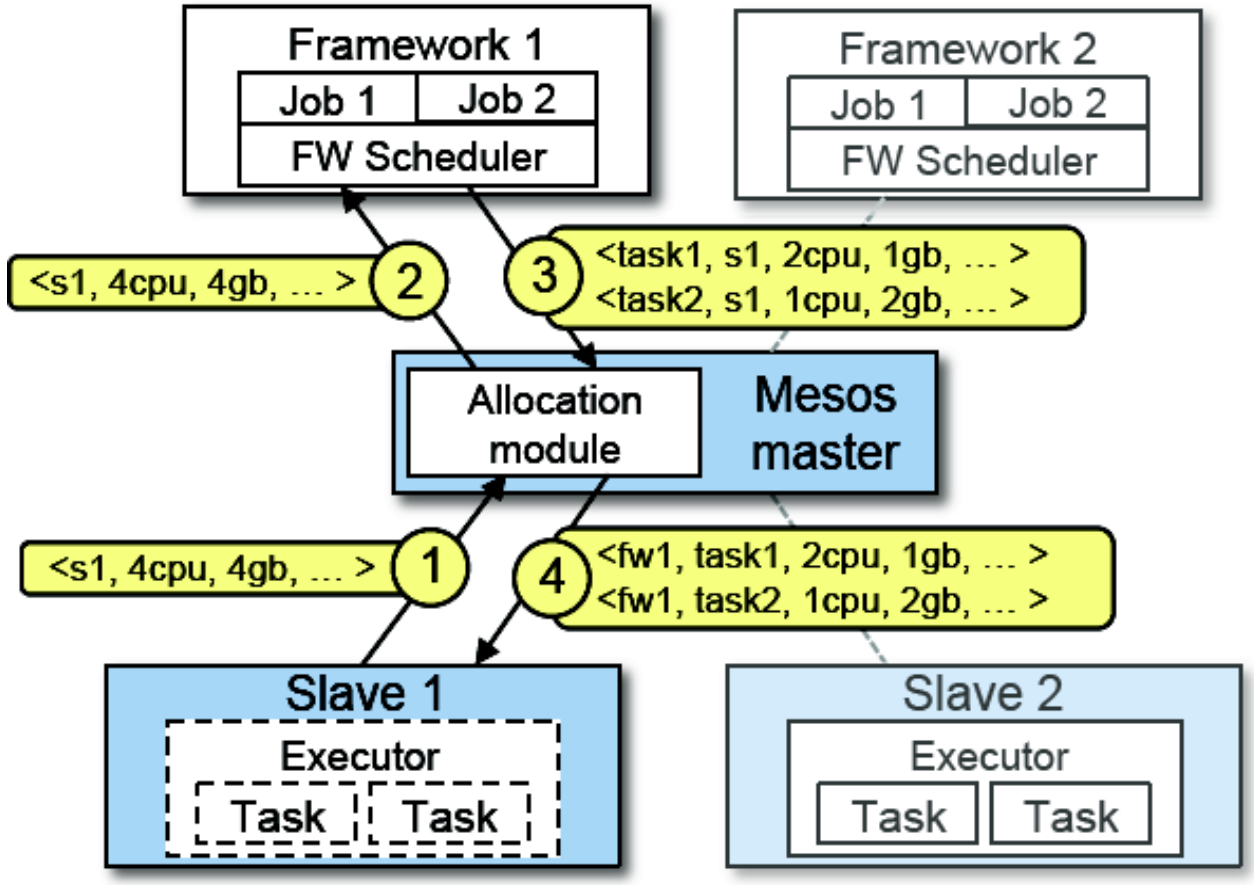
# Resource offer

- Offer available resources to frameworks, let them pick which resources to use and which tasks to launch.
- Keeps Mesos simple, lets it support future frameworks.

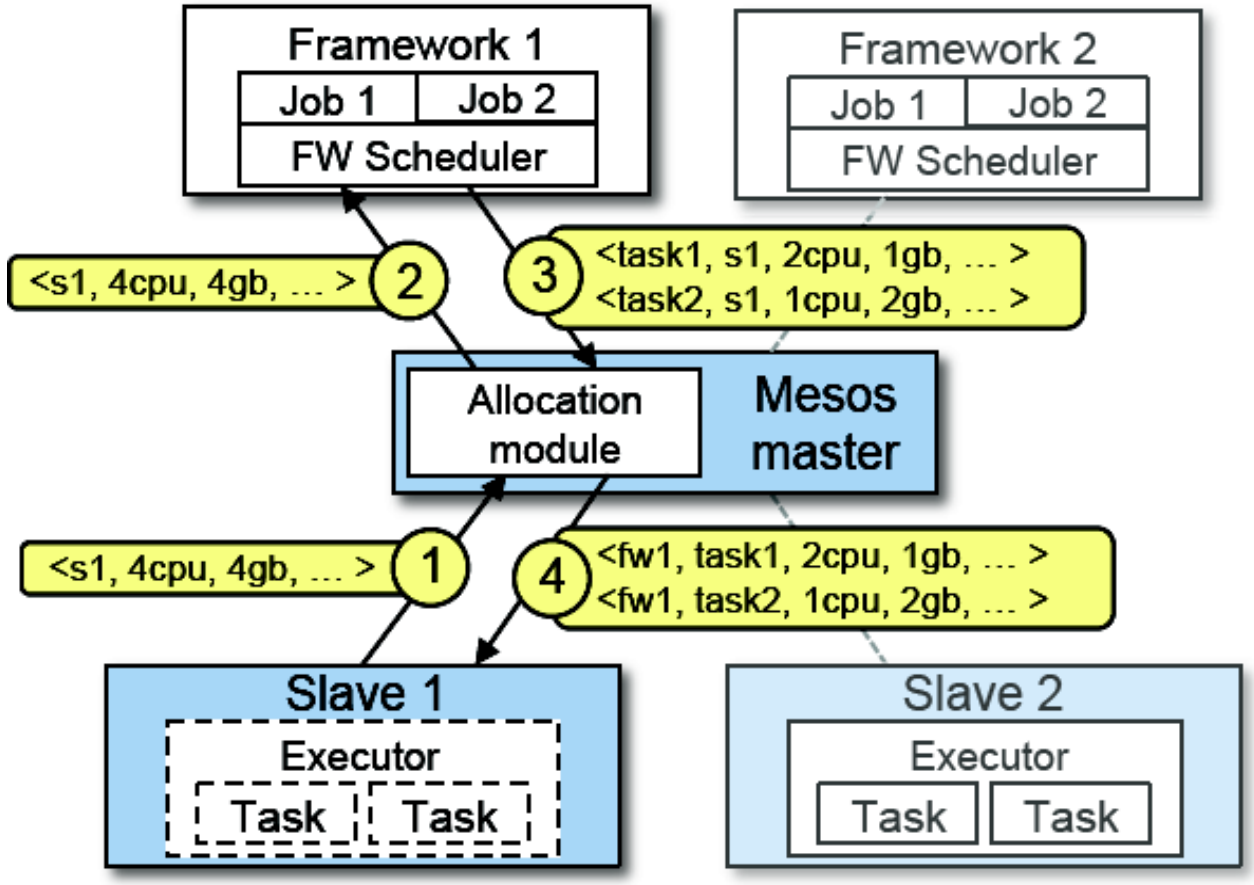


# How to schedule resource offering among frameworks?

# Mesos Architecture

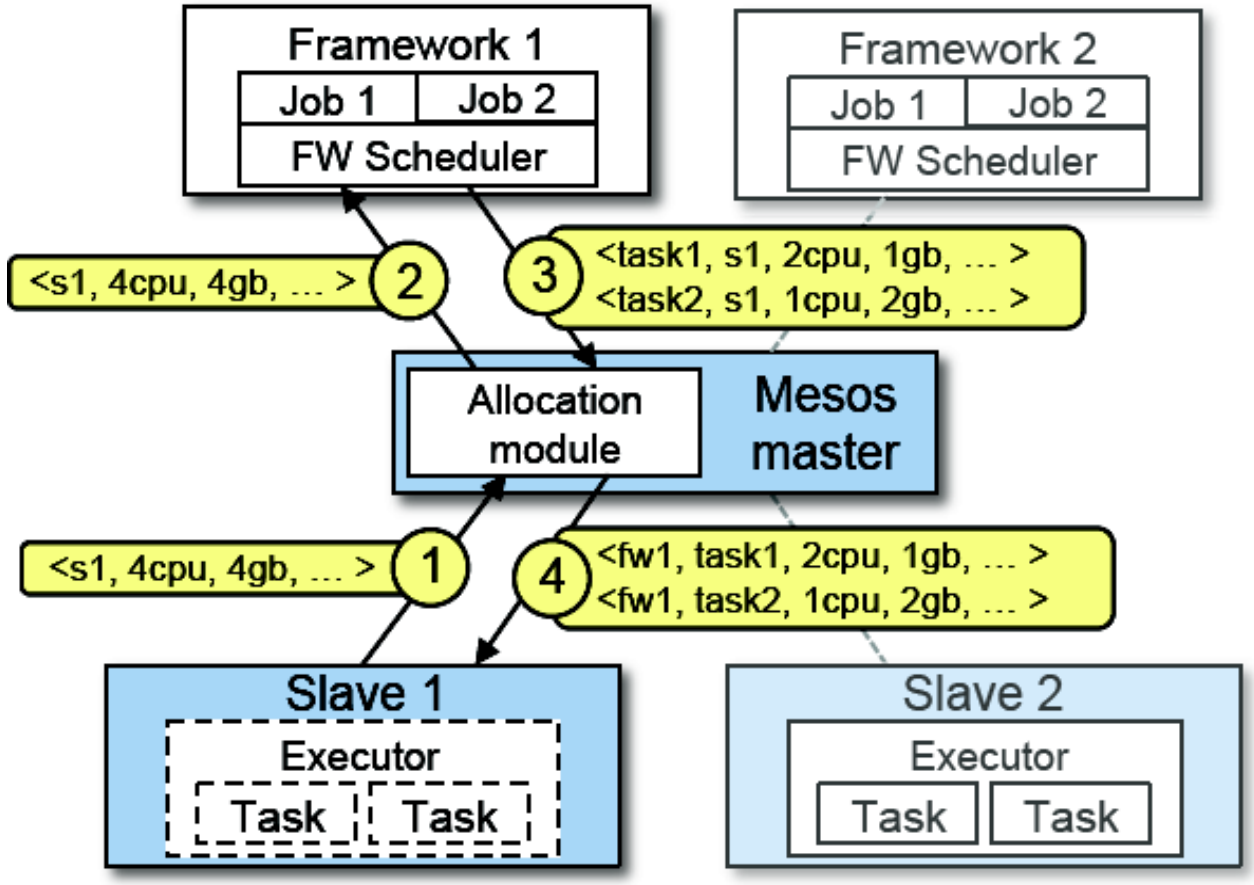


# Mesos Architecture



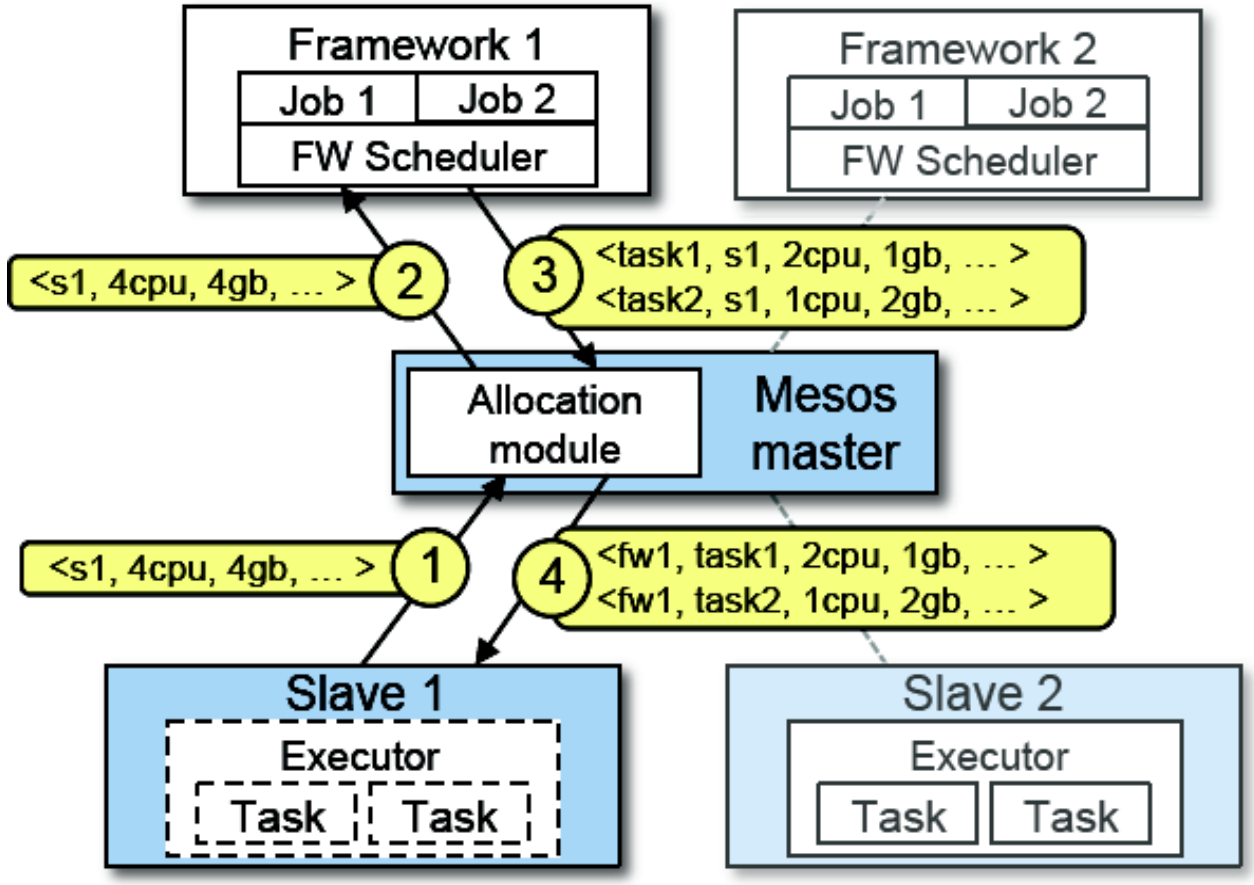

- Slaves continuously send status updates about resources to the Master.

# Mesos Architecture



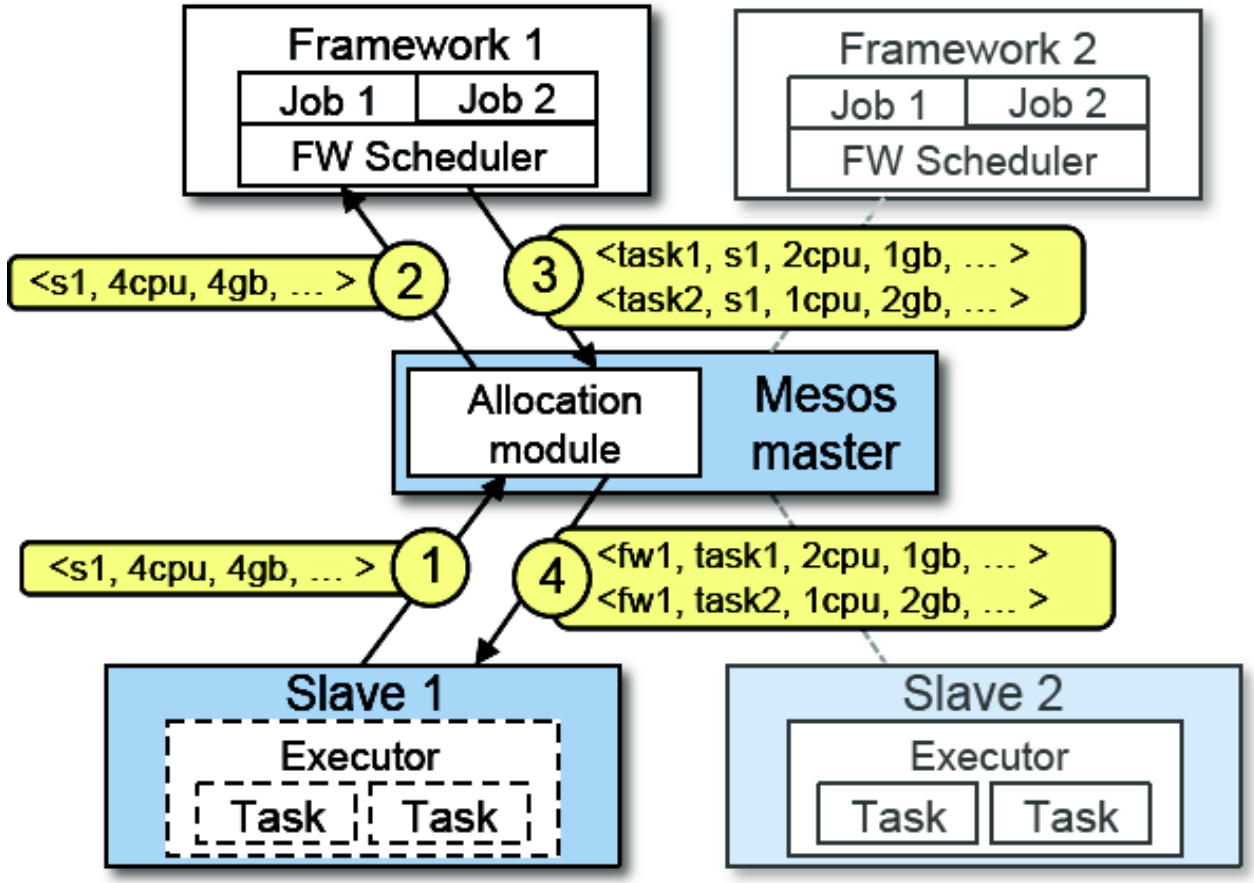

- Mesos Master sends resource offers to frameworks.

# Mesos Architecture




- Framework scheduler selects resources and provides tasks.

# Mesos Architecture

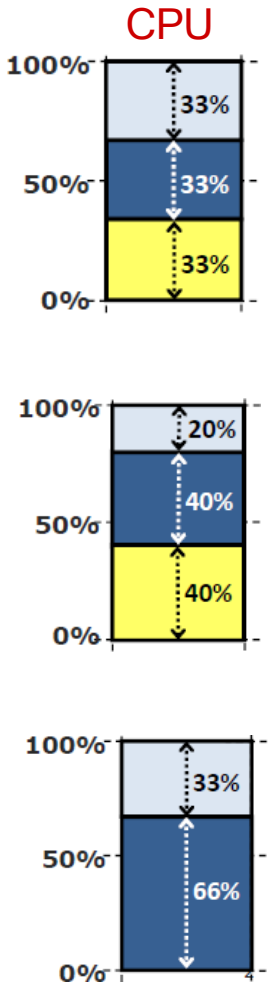



- Framework executors launch tasks.

# How to allocate resources for different frameworks?

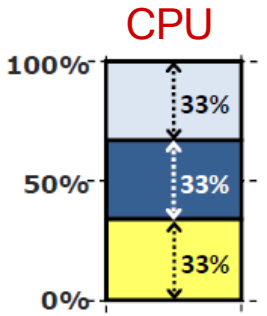
# Single resource: Fair Sharing

- **n** users want to share a resource, e.g., CPU.
  - Solution: allocate each $\frac{1}{n}$ of the shared resource.

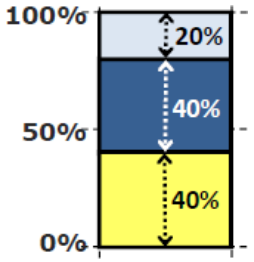


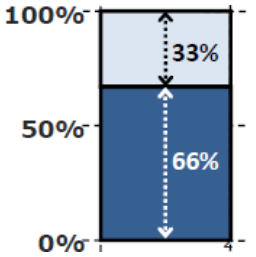
# Single resource: Fair Sharing

- **n** users want to share a resource, e.g., CPU.
  - Solution: allocate each $\frac{1}{n}$ of the shared resource.



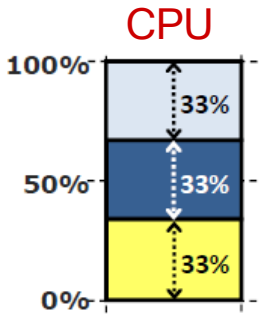

- Generalized by max-min fairness.
  - Handles if a user wants less than its fair share.
  - E.g., user 1 wants no more than 20%.



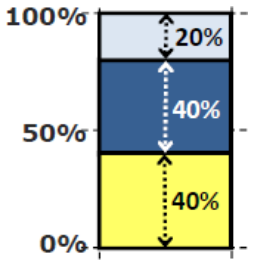



# Single resource: Fair Sharing

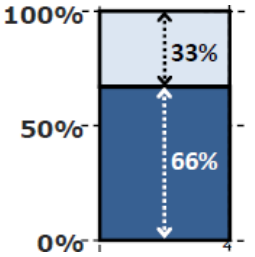
- **n** users want to share a resource, e.g., CPU.
  - Solution: allocate each $\frac{1}{n}$ of the shared resource.




- Generalized by max-min fairness.
  - Handles if a user wants less than its fair share.
  - E.g., user 1 wants no more than 20%.




- Generalized by weighted max-min fairness.
  - Give weights to users according to importance.
  - E.g., user 1 gets weight 1, user 2 weight 2.



# Max-Min Fairness

- 1 resource: CPU
- Total resources: 20 CPU
- User 1 has x tasks and wants $\langle 1CPU \rangle$ per task
- User 2 has y tasks and wants $\langle 2CPU \rangle$ per task

$$\max(x, y) \quad \text{(maximize allocation)} \quad \text{subject to}$$
$$x + 2y \leqslant 20 \quad \text{(CPU constraint)}$$
$$x = 2y$$

so

$$x = 10$$
$$y = 5$$

# When is Max-Min Fairness NOT enough?

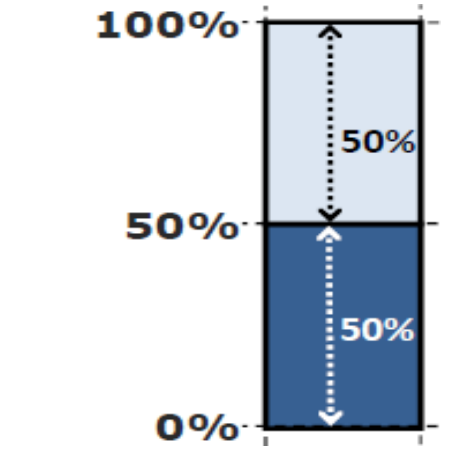
- Need to schedule multiple, heterogeneous resources, e.g., CPU, memory, etc.

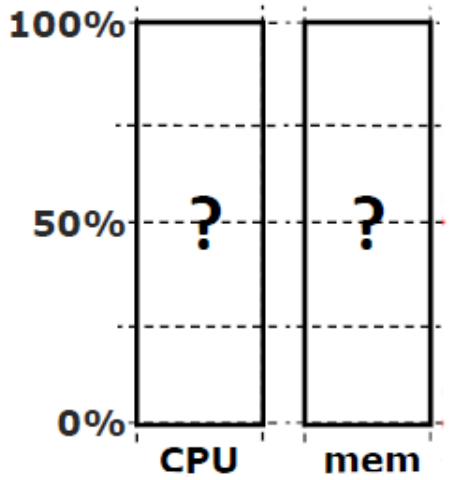
# Problem

- Single resource example
  - 1 resource: CPU
  - User 1 wants $\langle 1CPU \rangle$ per task
  - User 2 wants $\langle 2CPU \rangle$ per task

- Multi-resource example
  - 2 resources: CPUs and mem
  - User 1 wants $\langle 1CPU, 4GB \rangle$ per task
  - User 2 wants $\langle 2CPU, 1GB \rangle$ per task

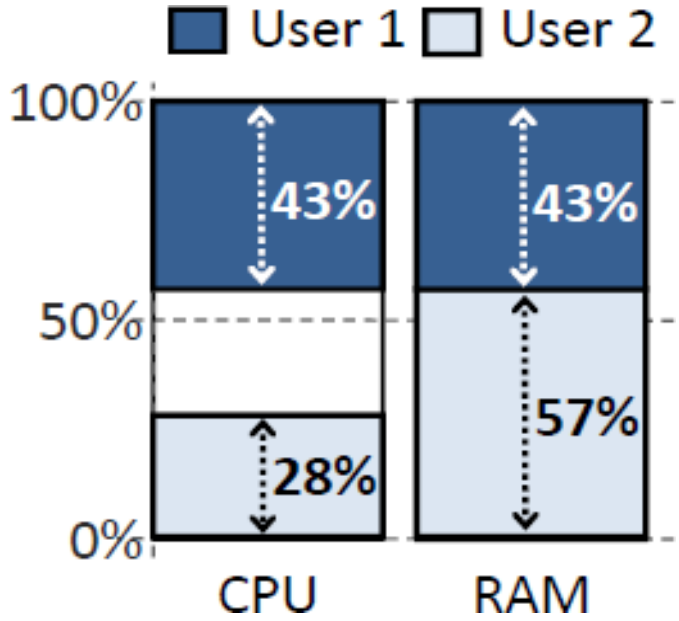
  - What is a fair allocation?



# A Natural Policy

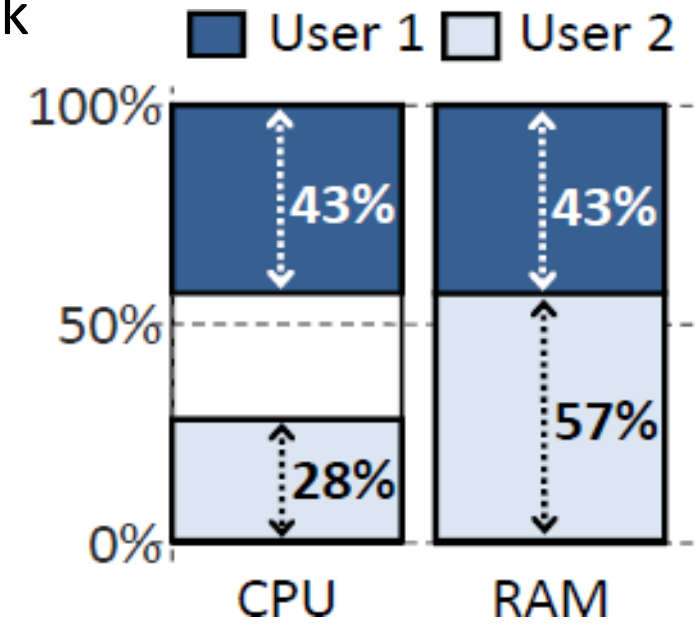
- Asset fairness: give weights to resources (e.g., 1 CPU = 1 GB) and equalize total value given to each user.



# A Natural Policy

- Asset fairness: give weights to resources (e.g., 1 CPU = 1 GB) and equalize total value given to each user.

- Total resources: 28 CPU and 56GB RAM (e.g., 1 CPU = 2 GB)
  - User 1 has x tasks and wants $\langle 1CPU, 2GB \rangle$ per task
  - User 2 has y tasks and wants $\langle 1CPU, 4GB \rangle$ per task



# A Natural Policy

- Asset fairness: give weights to resources (e.g., 1 CPU = 1 GB) and equalize total value given to each user.

- Total resources: 28 CPU and 56GB RAM (e.g., 1 CPU = 2 GB)
  - User 1 has x tasks and wants $\langle 1CPU, 2GB \rangle$ per task
  - User 2 has y tasks and wants $\langle 1CPU, 4GB \rangle$ per task
- Asset fairness yields:

$\max(x, y)$

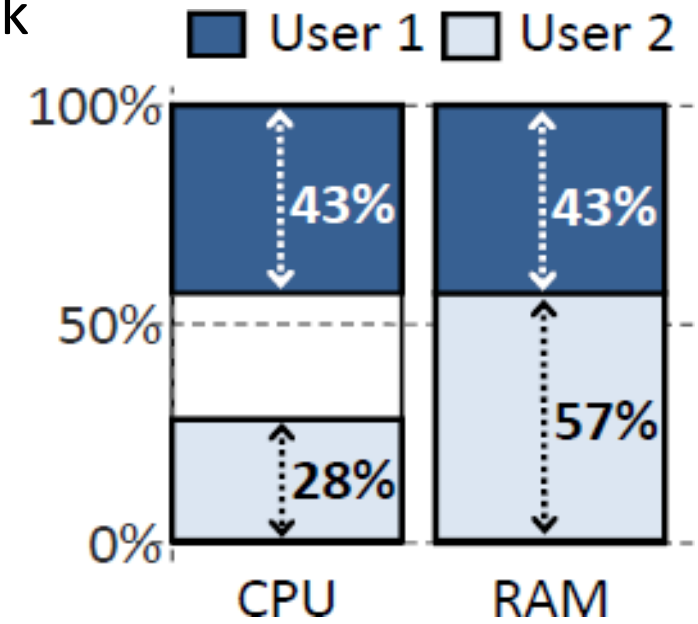
$x + y \leq 28$

$2x + 4y \leq 56$

$2x = 3y$

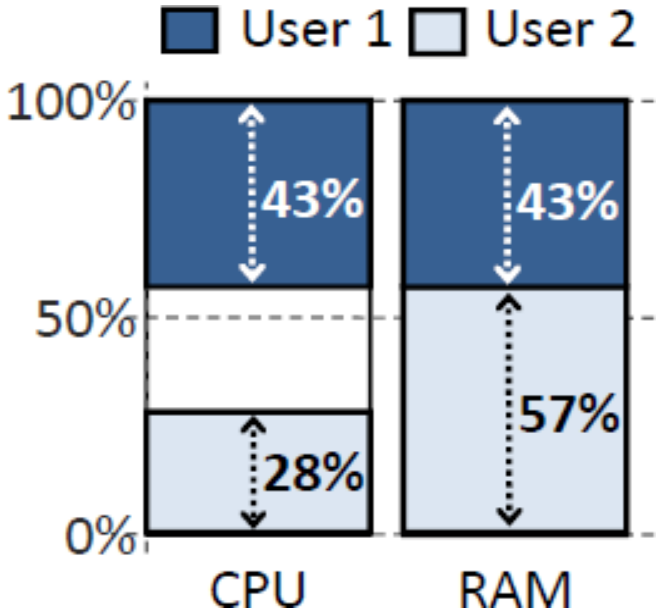
User 1: $x = 12$: $\langle 43\%CPU, 43\%GB \rangle$ ($\sum = 86\%$)

User 2: $y = 8$: $\langle 28\%CPU, 57\%GB \rangle$ ($\sum = 86\%$)



# A Natural Policy

- Problem: violates share grantee.
- User 1 gets less than 50% of both CPU and RAM.
- Better off in a separate cluster with half the resources.



# Dominant Resource Fairness (DRF)

- Dominant resource of a user: the resource that user has the biggest share of.
  - Total resources: $\langle 8CPU, 5GB \rangle$
  - User 1 allocation: $\langle 2CPU, 1GB \rangle$

$$\frac{2}{8} = 25\%\text{CPU and } \frac{1}{5} = 20\%\text{RAM}$$

  - Dominant resource of User 1 is CPU (25% > 20%)
- Dominant share of a user: the fraction of the dominant resource she is allocated.
  - User 1 dominant share is 25%.

# DRF

- Apply max-min fairness to dominant shares: give every user an equal share of her dominant resource.
- Equalize the dominant share of the users.
  - Total resources: ⟨9*C P U*, 18*G B* ⟩
    User 1 wants ⟨1*C P U*, 4*G B* ⟩; Dominant resource: RAM
    User 2 wants ⟨3*C P U*, 1*G B* ⟩; Dominant resource: CPU

$\frac{1}{9} < \frac{4}{18}$

$\frac{3}{9} > \frac{1}{18}$

- max(x, y)

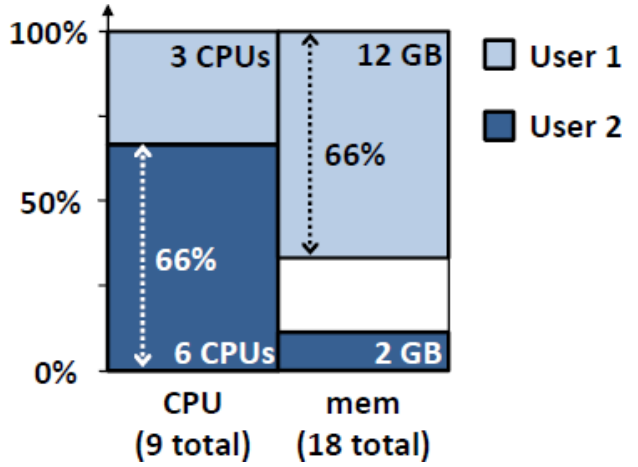
$x + 3y \leq 9$

$4x + y \leq 18$

$\frac{4x}{18} = \frac{3y}{9}$

User 1: x = 3: ⟨33%*C P U*, 66%*G B* ⟩

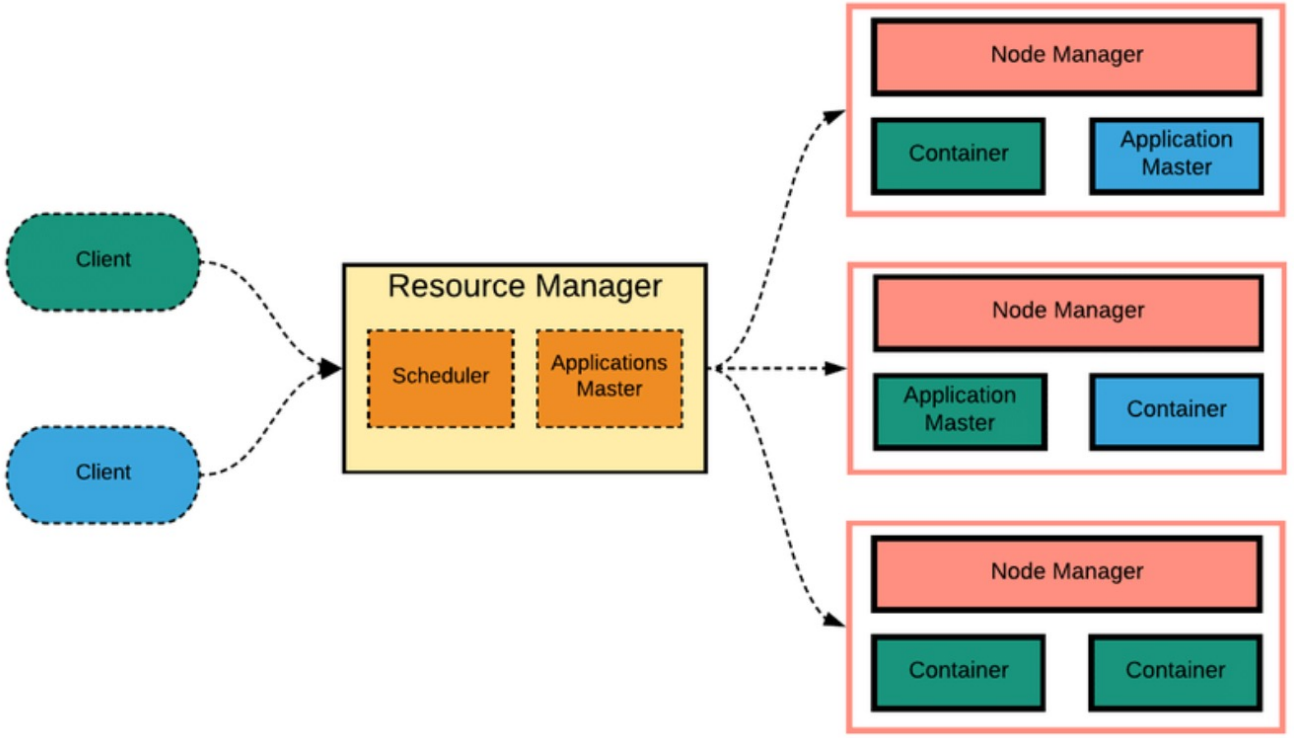
User 2: y = 2: ⟨66%*C P U*, 16%*G B* ⟩



# YARN
# (Yet Another Resource Negotiator)

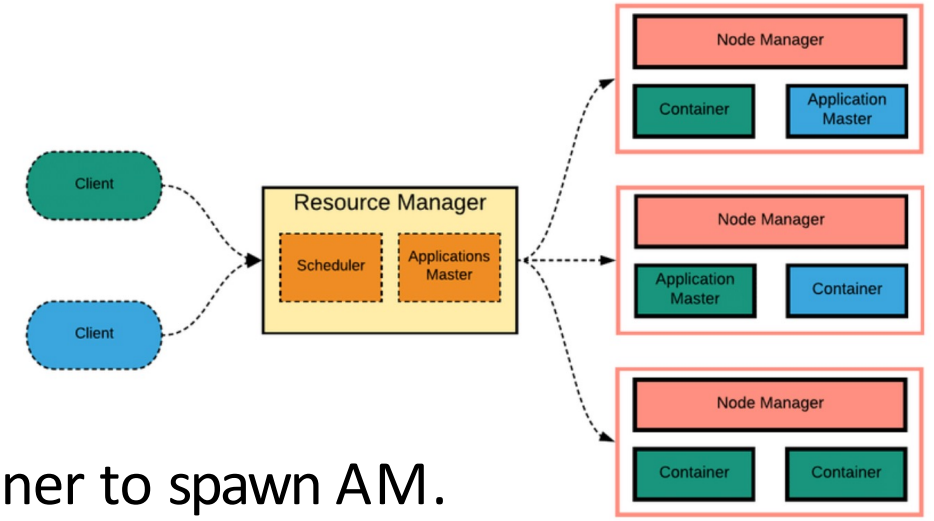
# Yarn Architecture

- Resource Manager (RM)
- Application Master (AM)
- Node Manager (NM)



# Yarn Architecture – Resource Manager

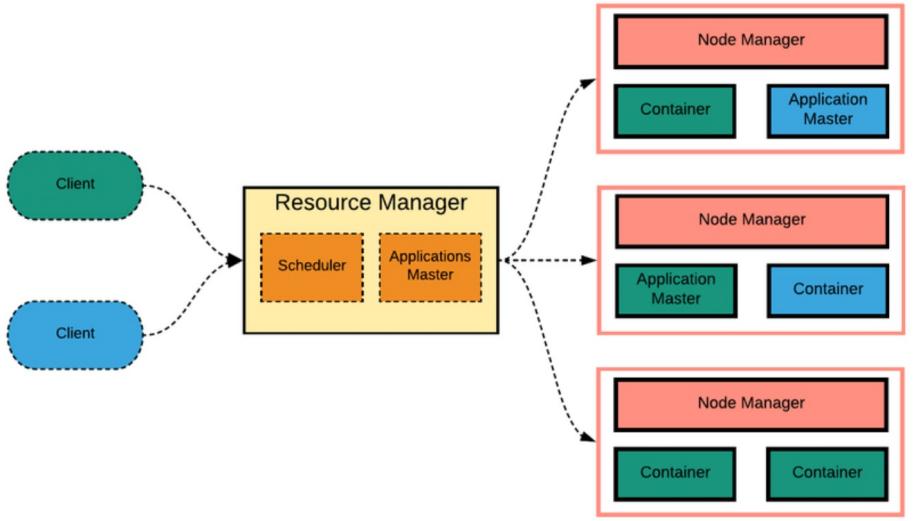
- One per cluster
  - Central: global view
- Resource Manager (RM)
  - Scheduler
  - Application Manager
- Container
  - Logical bundle of resources (CPU/memory).
- Job requests are submitted to RM.
  - To start a job (application), RM finds a container to spawn AM.



# Yarn Architecture – Scheduler

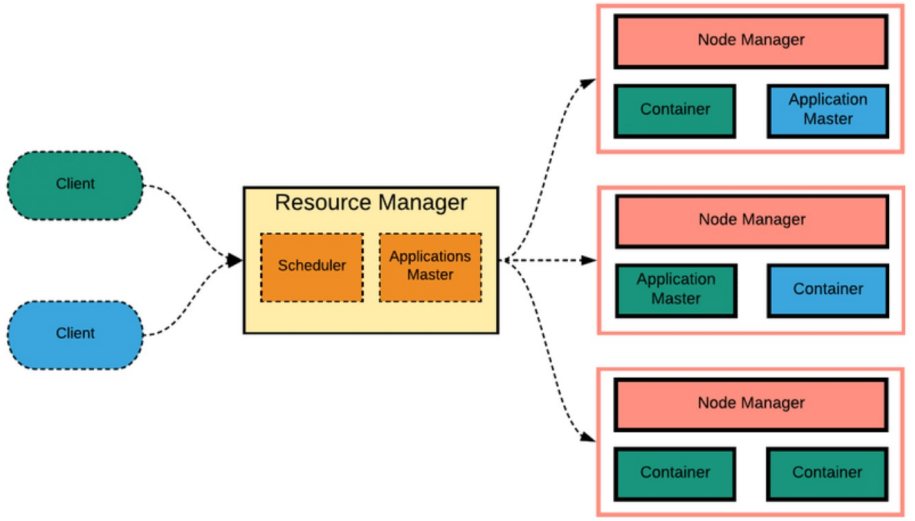
- Responsible for allocating resources to the various running applications subject to constraints of capacities, queues etc.

- Responsible for partitioning the cluster resources among the various applications.



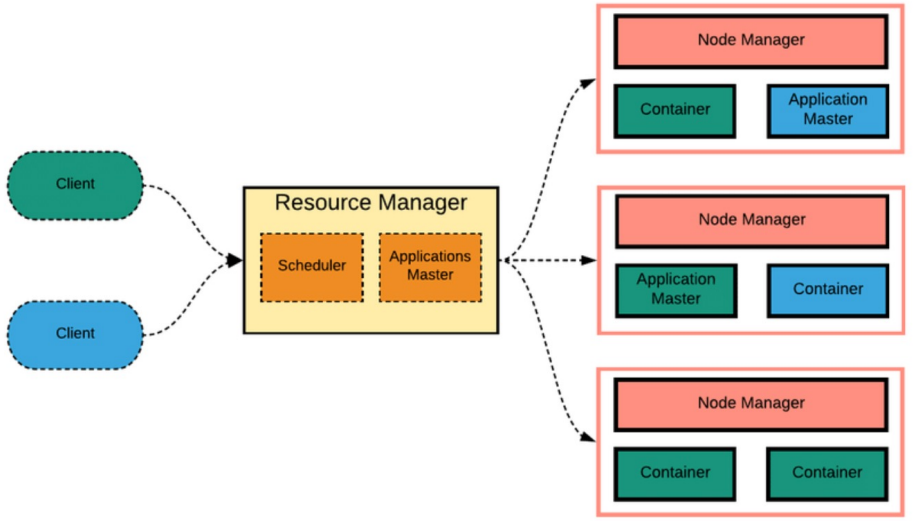
# Yarn Architecture – Application Manager

- Responsible for accepting job submissions.
- Negotiates the first container from the Resource Manager for executing the application specific Application Master.
- Manages running the Application Masters in a cluster and provides service for restarting the Application Master container on failure.
-



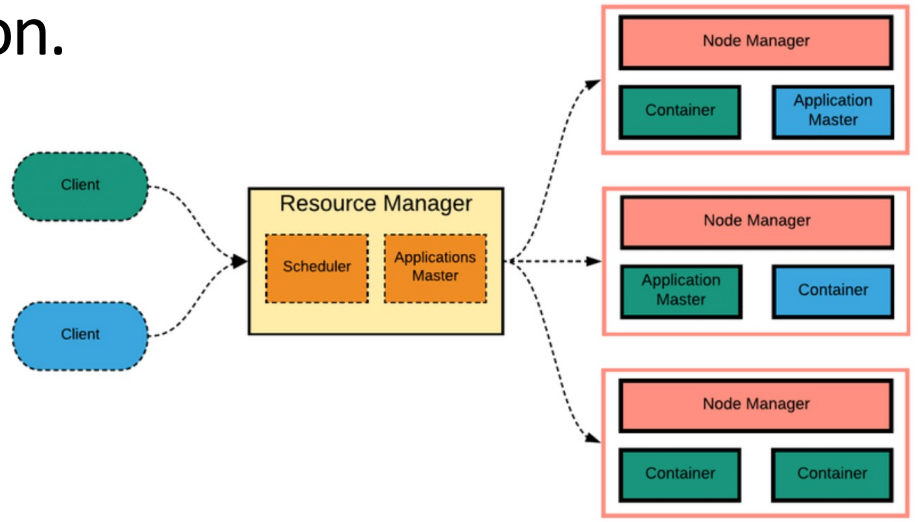
# Yarn Architecture – Application Master

- The head of a job.
- Runs as a container.
- Request resources from RM for running different tasks.



# Yarn Architecture – Node Manager

- The worker daemon.
- Registers with RM.
- One per node.
- Report resources to RM: memory, CPU, …
- Configure the environment for task execution.
- Garbage collection.



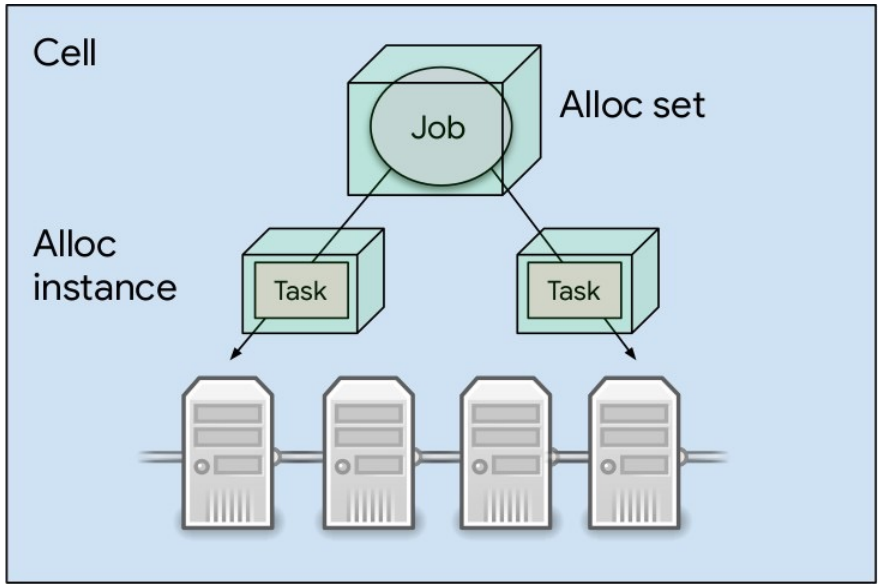
# Borg

# Borg

- Cluster Management System at Google

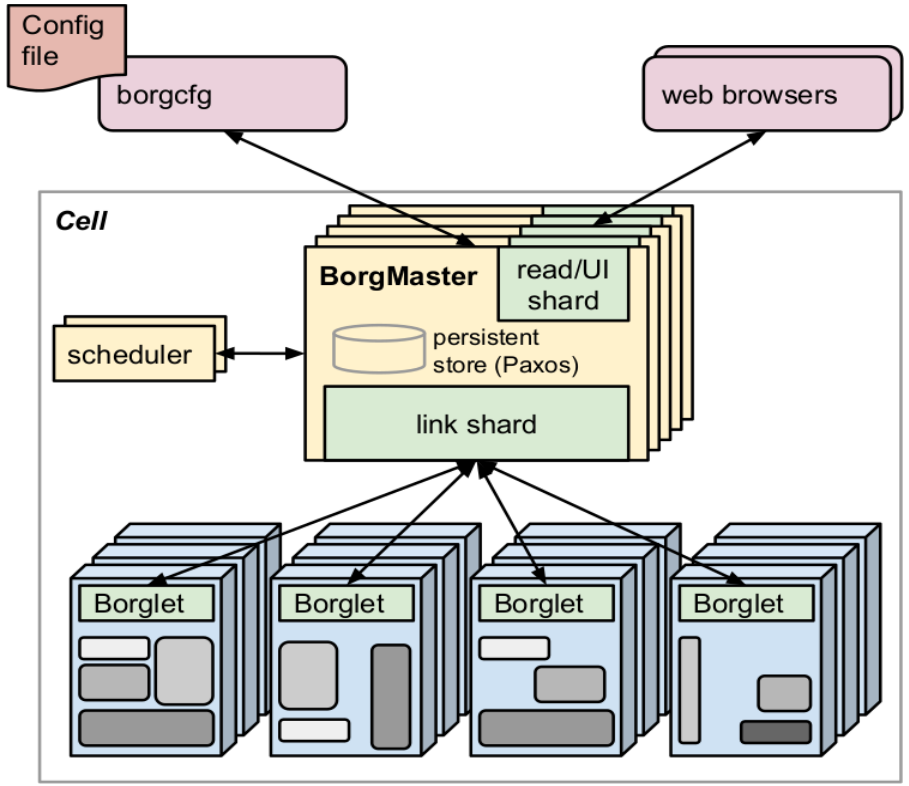
# Borg

- Cell: a set of machines managed by Borg as one unit.
- Job: users submit work in the form of jobs.
- Task: each job contains one or more tasks.
- Alloc: reserved set of resources and a job can run in an alloc set.
- Alloc instance: making each of its tasks run in an alloc instance.



# Borg Architecture

- BorgMaster
  - The central brain of the system
  - Holds the cluster state
  - Replicated for reliability (using paxos)
  - Scheduling: where to place tasks?
- Borglet
  - Manage and monitor tasks and resource
  - BorgMaster polls Borglet every few seconds



# Borg Scheduler

- Feasibility checking: find machines for a given job
- Scoring: pick one machine
- According to the users preferences and built-in criteria

# Kubernetes

- Kubernetes is the Google open source project loosely inspired by Borg.
- Directly derived
  - Borglet → Kubelet
  - alloc → pod
  - Borg containers → docker
  - Declarative specifications

# Recap

- Resource management Systems
  - Mesos
    - Offered-based
    - Max-Min fairness: DRF
  - YARN
    - Request-based
    - RM, AM, NM
  - Borg
    - Request-based
    - BorgMaster, Borglet
    - Kubernetes

# Next Class

# Guest Speaker - Cloud Security