Mesos and Borg (Lecture 17, cs262a)

Ion Stoica,
UC Berkeley
October 24, 2016

Today's Papers

Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center,

Benjamin Hindman, Andy Konwinski, Matei Zaharia,

Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, Ion Stoica, NSDI'11

(https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf)

Large-scale cluster management at Google with Borg,

Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, John Wilkes, EuroSys'15

(static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf)

Motivation

- Rapid innovation in cloud computing

Dryad

Hypertable

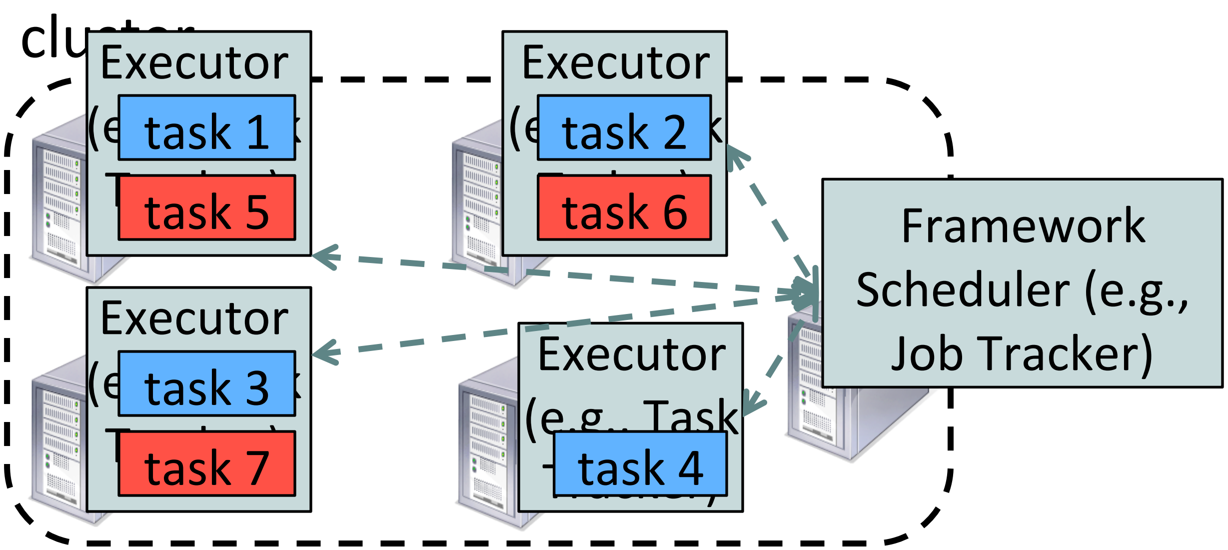
Cassandra

Pregel

- Today
  - No single framework optimal for all applications
  - Each framework runs on its dedicated cluster or cluster partition

Computation Model: Frameworks

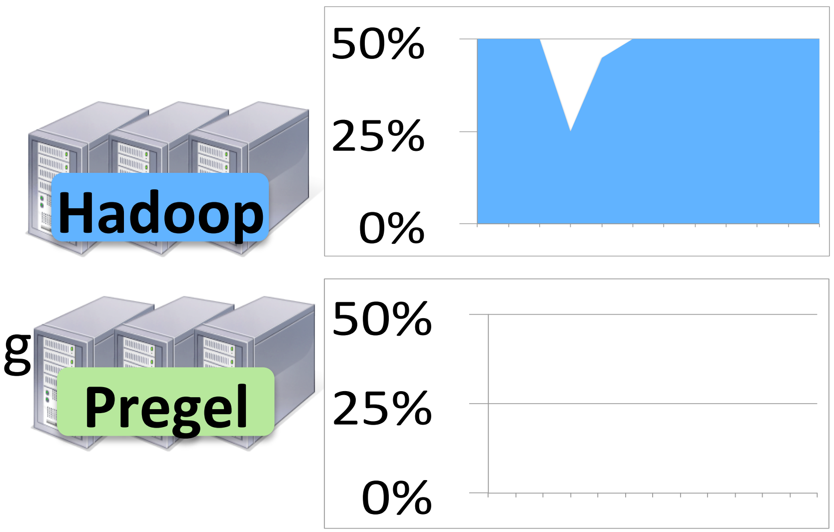
- A ***framework*** (e.g., Hadoop, MPI) manages one or more ***jobs*** in a computer cluster
- A ***job*** consists of one or more ***tasks***
- A ***task*** (e.g., map, reduce) is implemented by one or more processes running on a single machine



Job 1: tasks 1, 2, 3, 4
Job 2: tasks 5, 6, 7

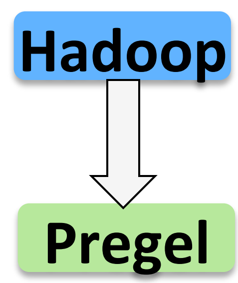
One Framework Per Cluster Challenges

- Inefficient resource usage
  - E.g., Hadoop cannot use available resources from Pregel's cluster
  - No opportunity for stat. multiplexing



- Hard to share data
  - Copy or access remotely, expensive

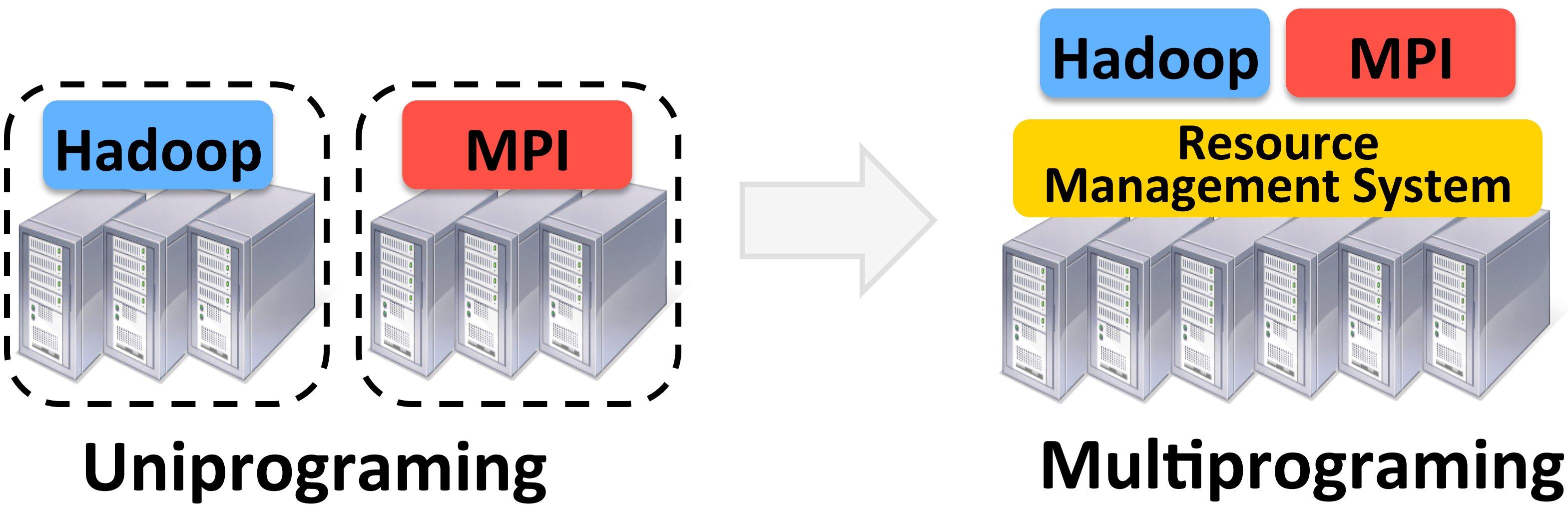
- Hard to cooperate
  - E.g., Not easy for Pregel to use graphs generated by Hadoop



Need to run multiple frameworks on same cluster

What do we want?

- Common resource sharing layer
  - Abstracts (“virtualizes”) resources to frameworks
  - Enable diverse frameworks to share cluster
  - Make it easier to develop and deploy new frameworks (e.g., Spark)



Fine Grained Resource Sharing

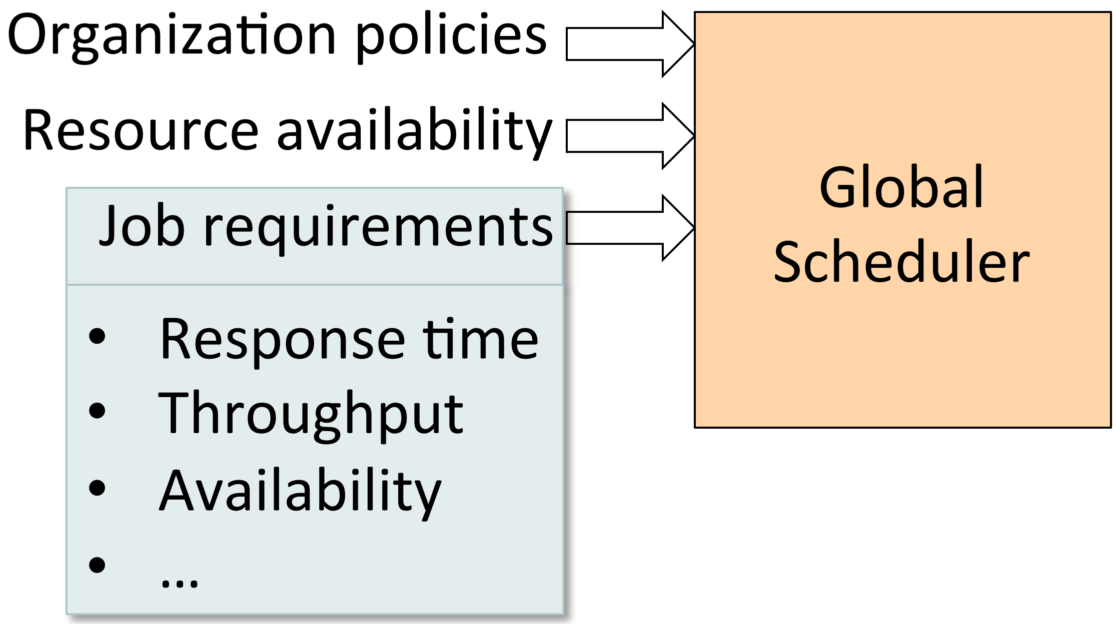
- Task granularity both in **time** & **space**
  - Multiplex node/time between tasks belonging to different jobs/frameworks
- Tasks typically short; median ~= 10 sec, minutes

- Why fine grained?
  - Improve data locality
  - Easier to handle node failures

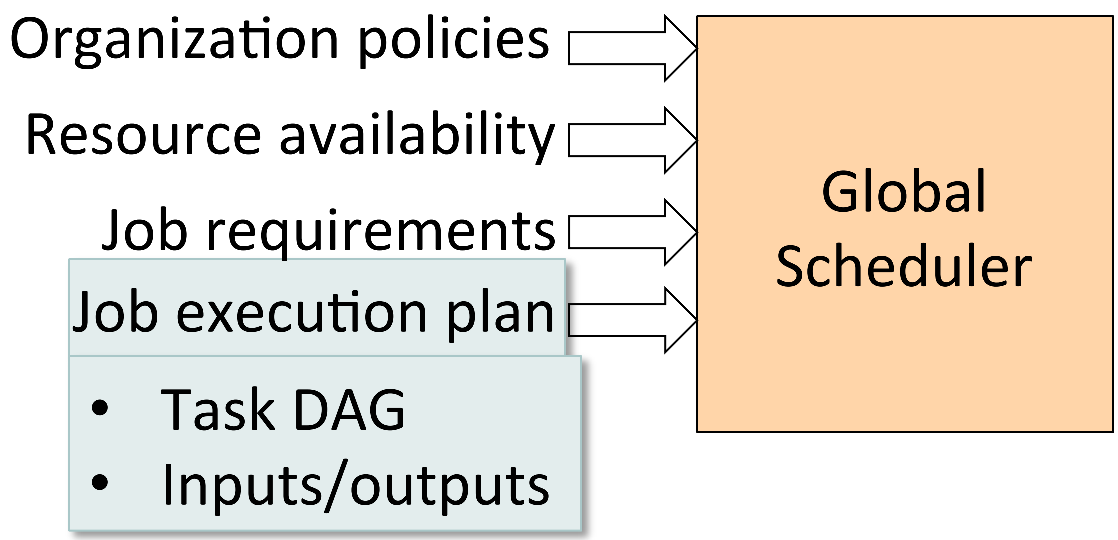
Goals

- **Efficient utilization** of resources
- **Support diverse frameworks** (existing & future)
- **Scalability** to 10,000's of nodes
- **Reliability** in face of node failures

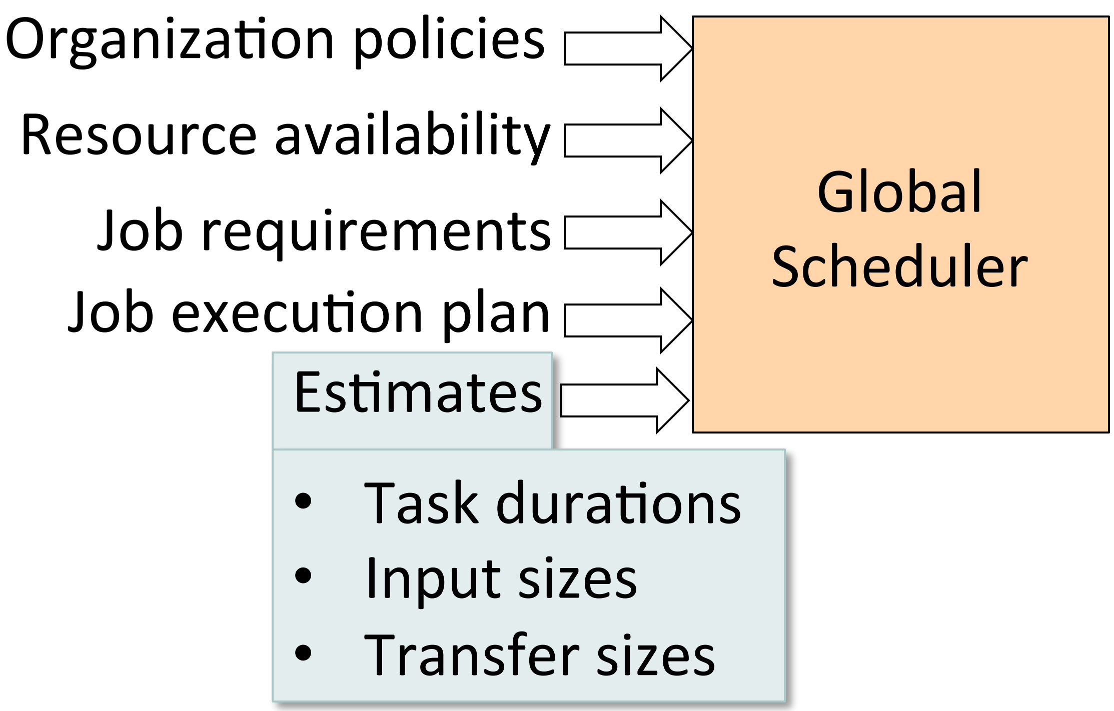
Approach: Global Scheduler



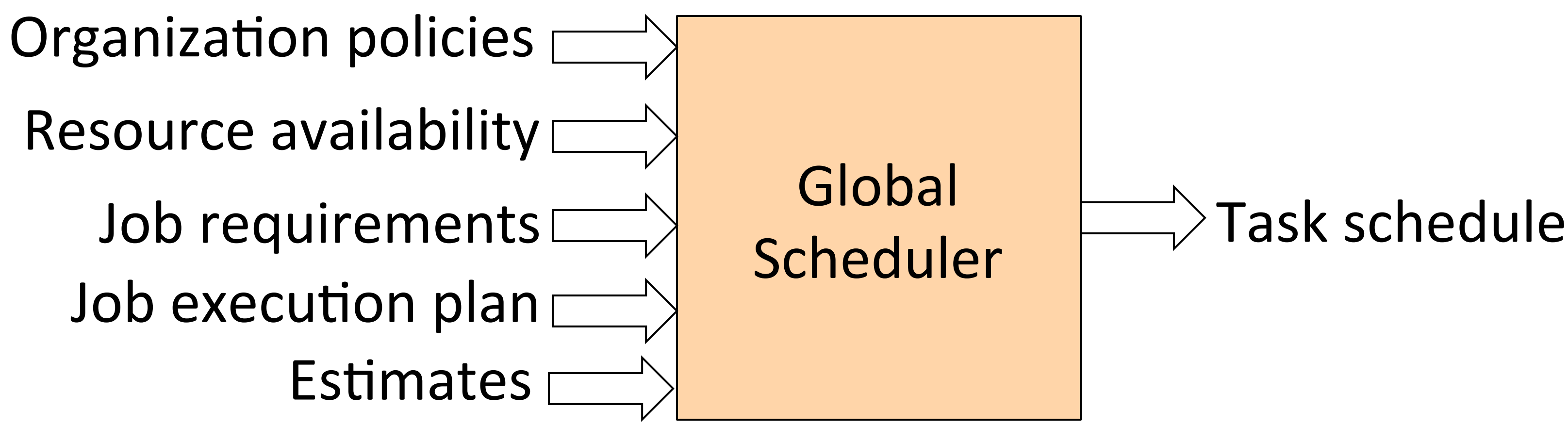
Approach: Global Scheduler



Approach: Global Scheduler



Approach: 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

Mesos

Resource Offers

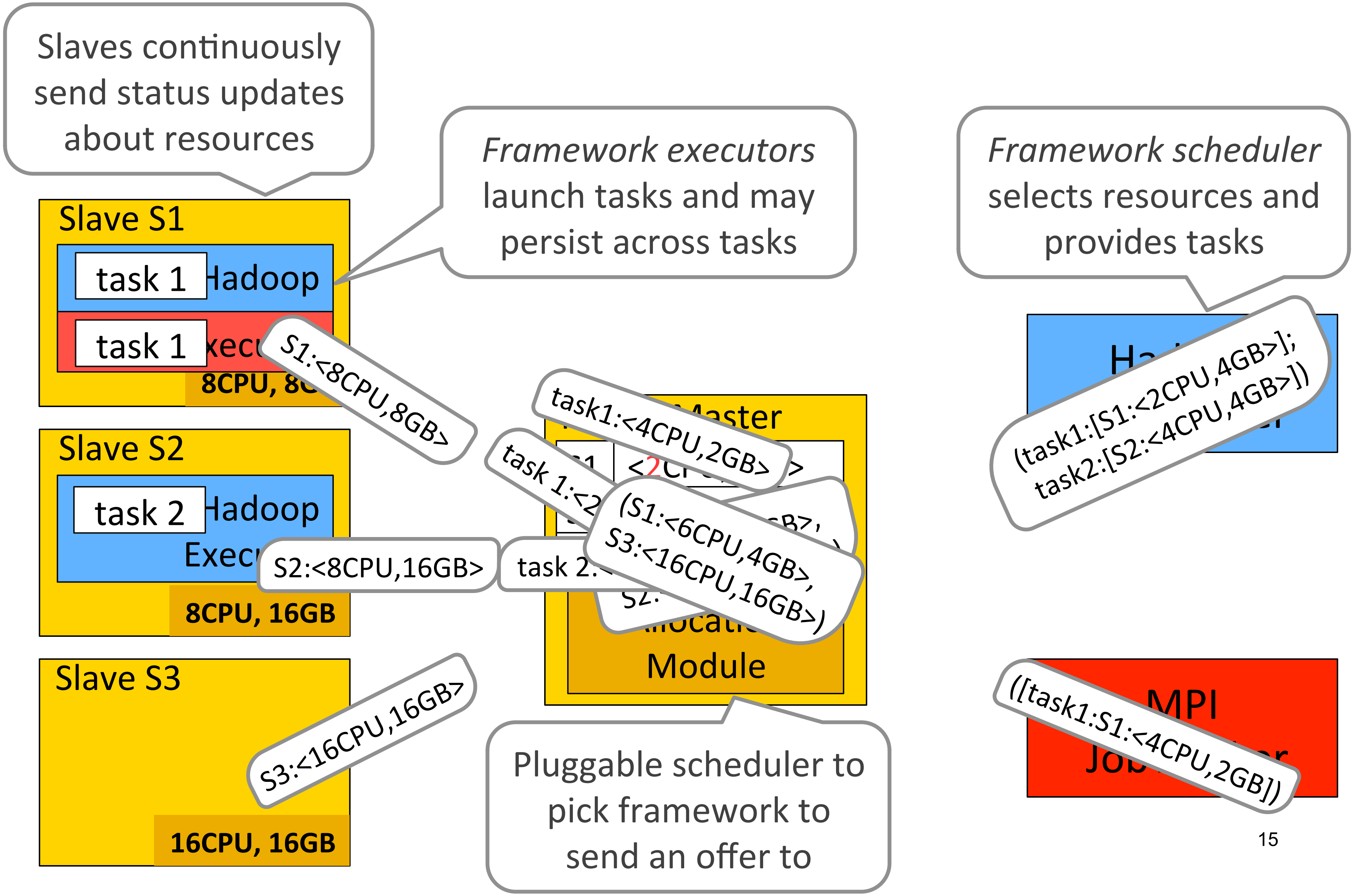
- Unit of allocation: ***resource offer***
  - Vector of available resources on a node
  - E.g., node1: <1CPU, 1GB>, node2: <4CPU, 16GB>

- Master sends resource offers to frameworks

- Frameworks select which offers to accept and which tasks to run

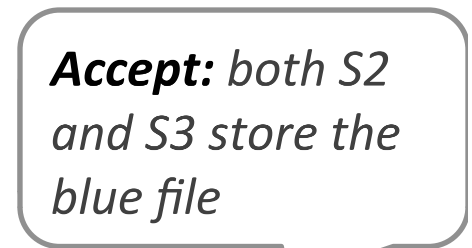
Push task scheduling to frameworks

Mesos Architecture: Example

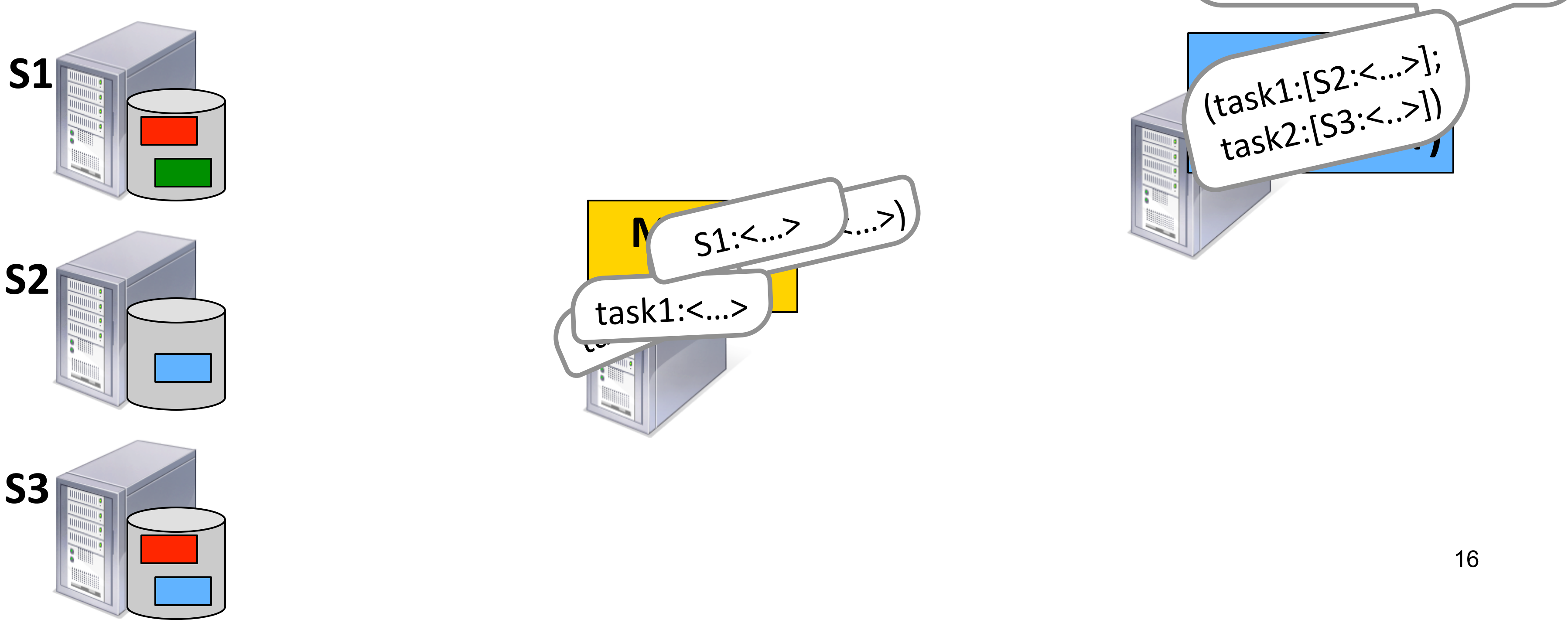


Why does it Work?

- A framework can just wait for an offer that matches its constraints or preferences!
  - Reject offers it does not like



- Example: Hadoop's job input is *blue* file



Two Key Questions

- How long does a framework need to wait?
- How do you allocate resources of different types?
  - E.g., if framework A has (1CPU, 3GB) tasks, and framework B has (2CPU, 1GB) tasks, how much we should allocate to A and B?

Two Key Questions

- How long does a framework need to wait?

- How do you allocate resources of different types?

How Long to Wait?

- Depend on
  - Distribution of task duration
  - "Pickiness" – set of resources satisfying framework **constraints**

- **Hard constraints:** cannot run if resources violate constraints
  - Software and hardware configurations (e.g., OS type and version, CPU type, public IP address)
  - Special hardware capabilities (e.g., GPU)
- **Soft constraints:** can run, but with degraded performance
  - Data, computation locality

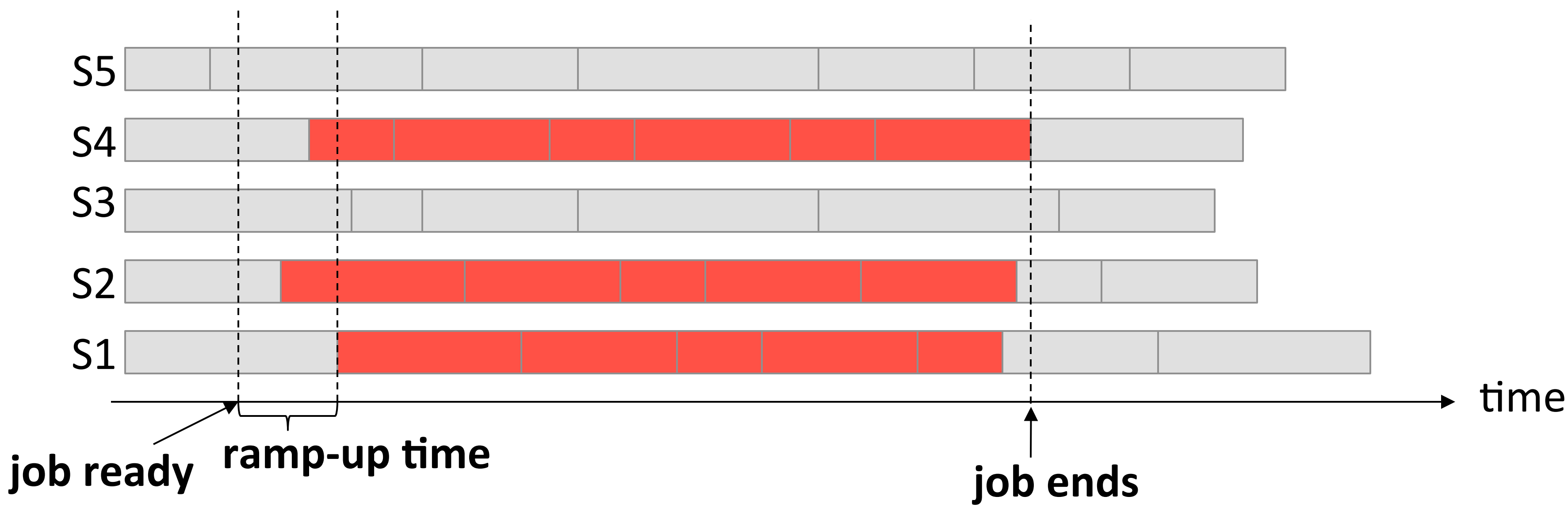
Model

- One job per framework
- One task per node
- No task preemption

- **Pickiness,** $p = k/n$
  - k – number of nodes required by job, e.g., it's target allocation
  - n – number of nodes satisfying framework's constraints in the cluster

Ramp-Up Time

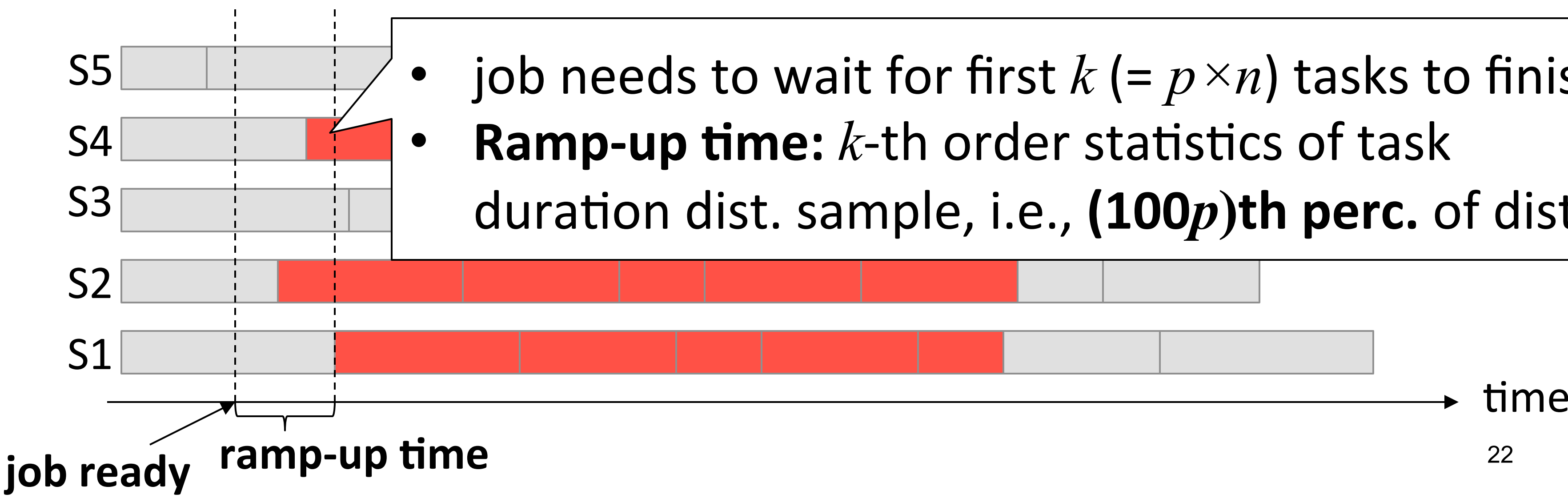
- **Ramp-Up Time**: time job waits to get its target allocation
- Example:
  - Job's target allocation, $k = 3$
  - Number of nodes job can pick from, $n = 5$



Pickiness: Ramp-Up Time

Estimated **ramp-up** time of a job with pickiness $\boldsymbol{p}$ is ≅ (100$\boldsymbol{p}$)**-th percentile** of task duration distribution

- E.g., if p = 0.9, estimated ramp-up time is the 90-th percentile of task duration distribution (T)
- Why? Assume: $k = 3,\ n = 5,\ p = k/n$



- job needs to wait for first k (= $p \times n$) tasks to finish
- **Ramp-up time:** k-th order statistics of task duration dist. sample, i.e., **(100p)th perc.** of dist.

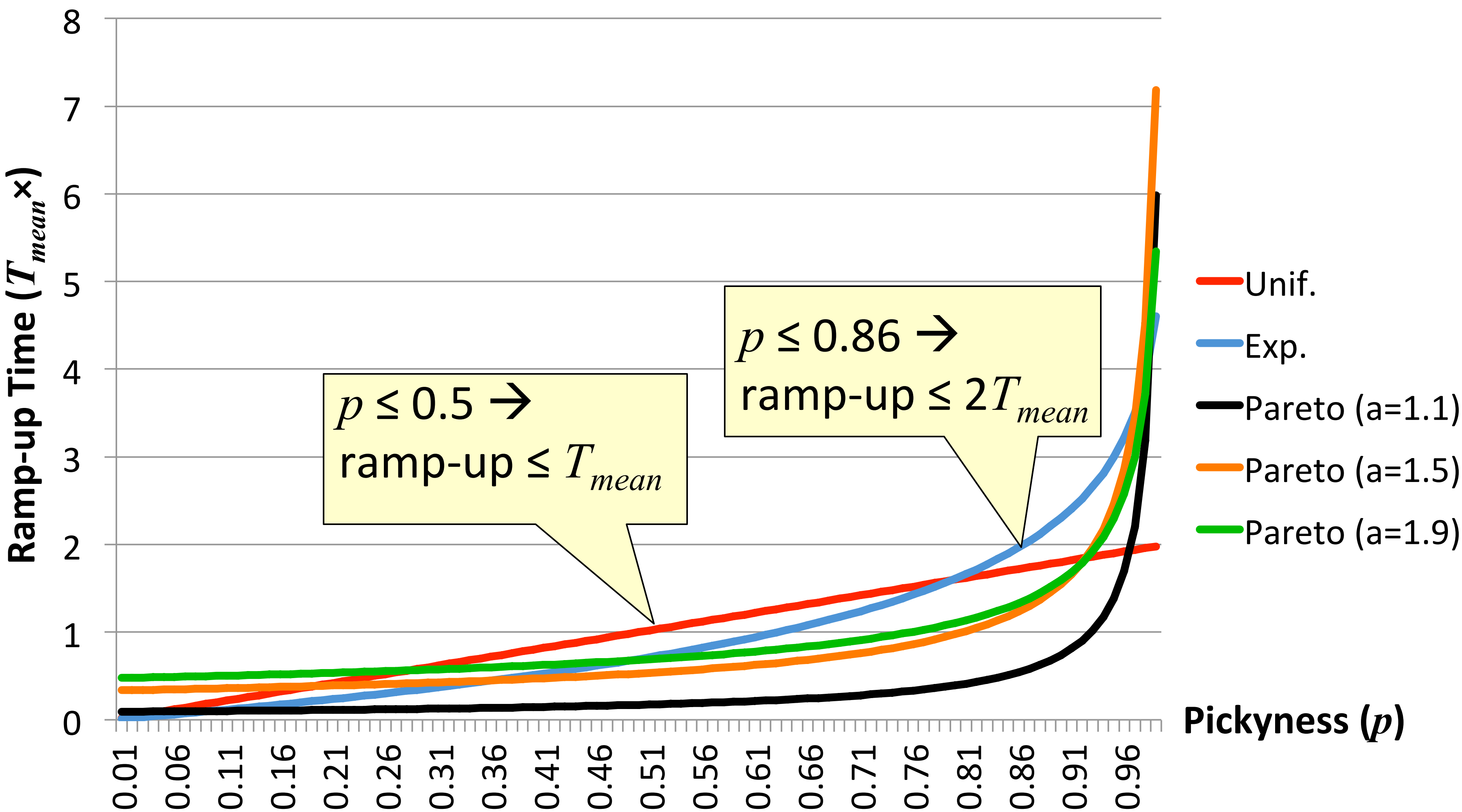
Alternate Interpretations

- If p = 1, estimated time of a job getting fraction q of its allocation is ≅ (100q)-th percentile of T
  - E.g., estimate time of a job getting 0.9 of its allocation is the 90-th percentile of T

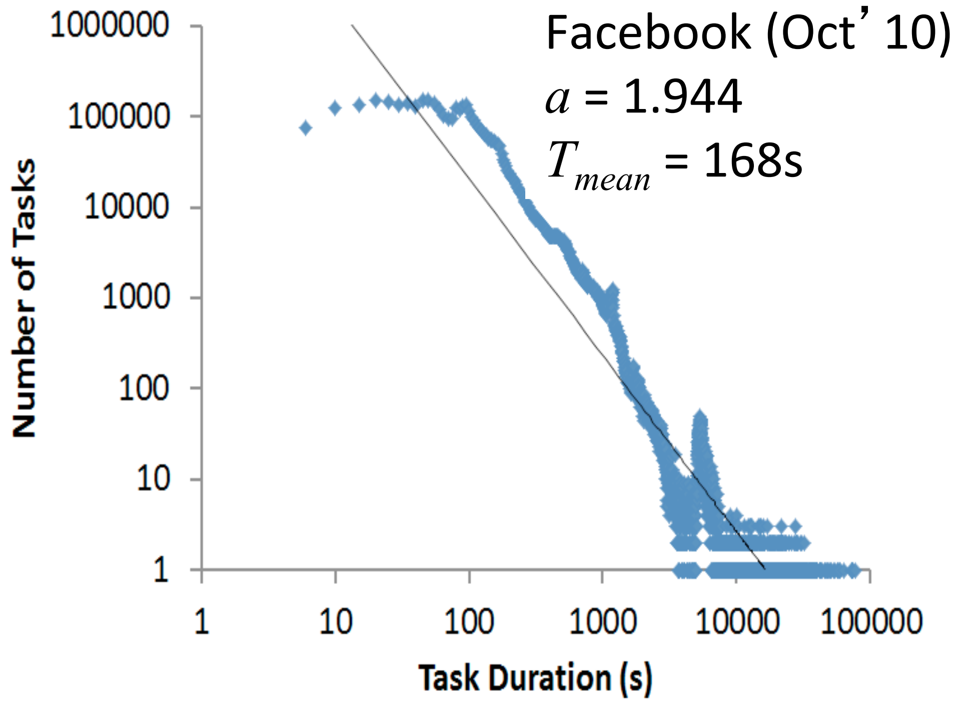
- If utilization of resources satisfying job's constraints is q, estimated time to get its allocation is ≅ (100q)-th perc. of T
  - E.g., if resource utilization is 0.9, estimated time of a job to get its allocation is the 90-th percentile of T

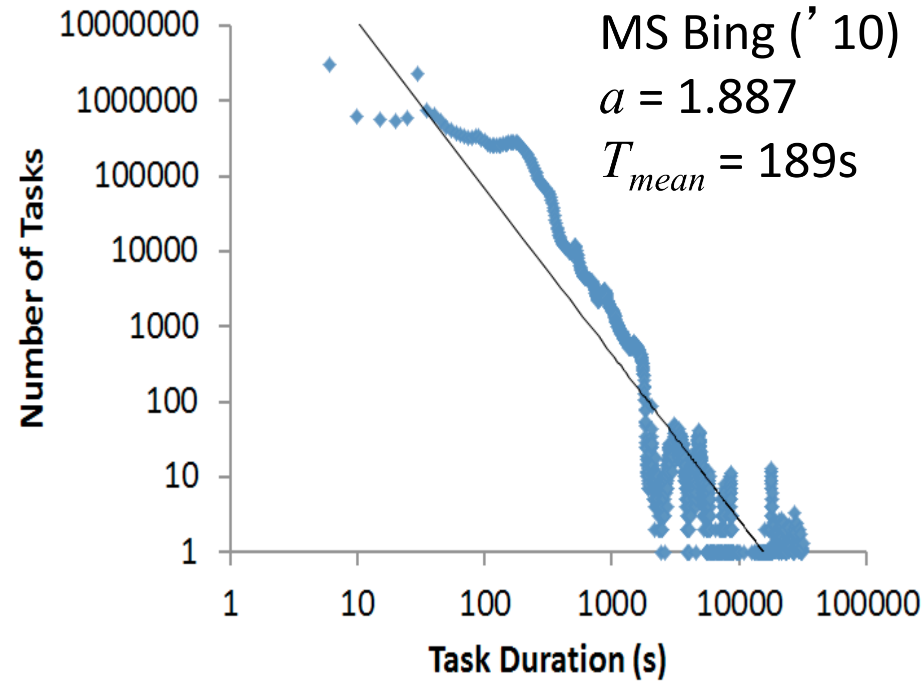
Ramp-Up Time: Mean

- Impact of heterogeneity of task duration distribution



Ramp-up Time: Traces





shape parameter, $a = 1.9$

Ramp-up	formula	p =0.1	p =0.5	p =0.9	p =0.98
mean (μ)	$\frac{(a-1)}{a} \times \frac{T_{mean}}{(1-p)^{1/a}}$	0.5 T_{mean}	0.68 T_{mean}	1.59 T_{mean}	3.71T_{mean}
stdev (σ)	$\frac{\mu}{a} \times \sqrt{\frac{p}{n(1-p)}}$	0.01 T_{mean}	0.04 T_{mean}	0.25 T_{mean}	1.37T_{mean}

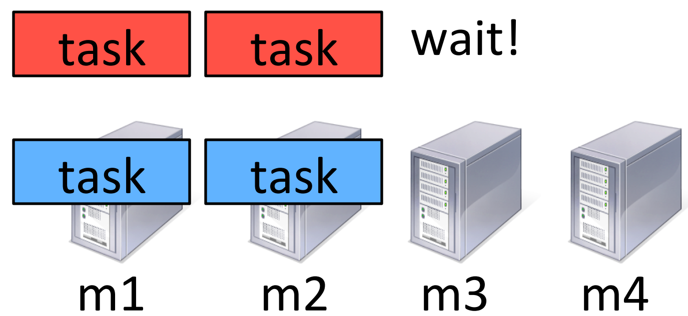
Improving Ramp-Up Time?

- **Preemption:** preempt tasks

- **Migration:** move tasks around to increase choice, e.g.,

  Job 1 constraint set = {m1, m2, m3, m4}

  Job 2 constraint set = {m1, m2}



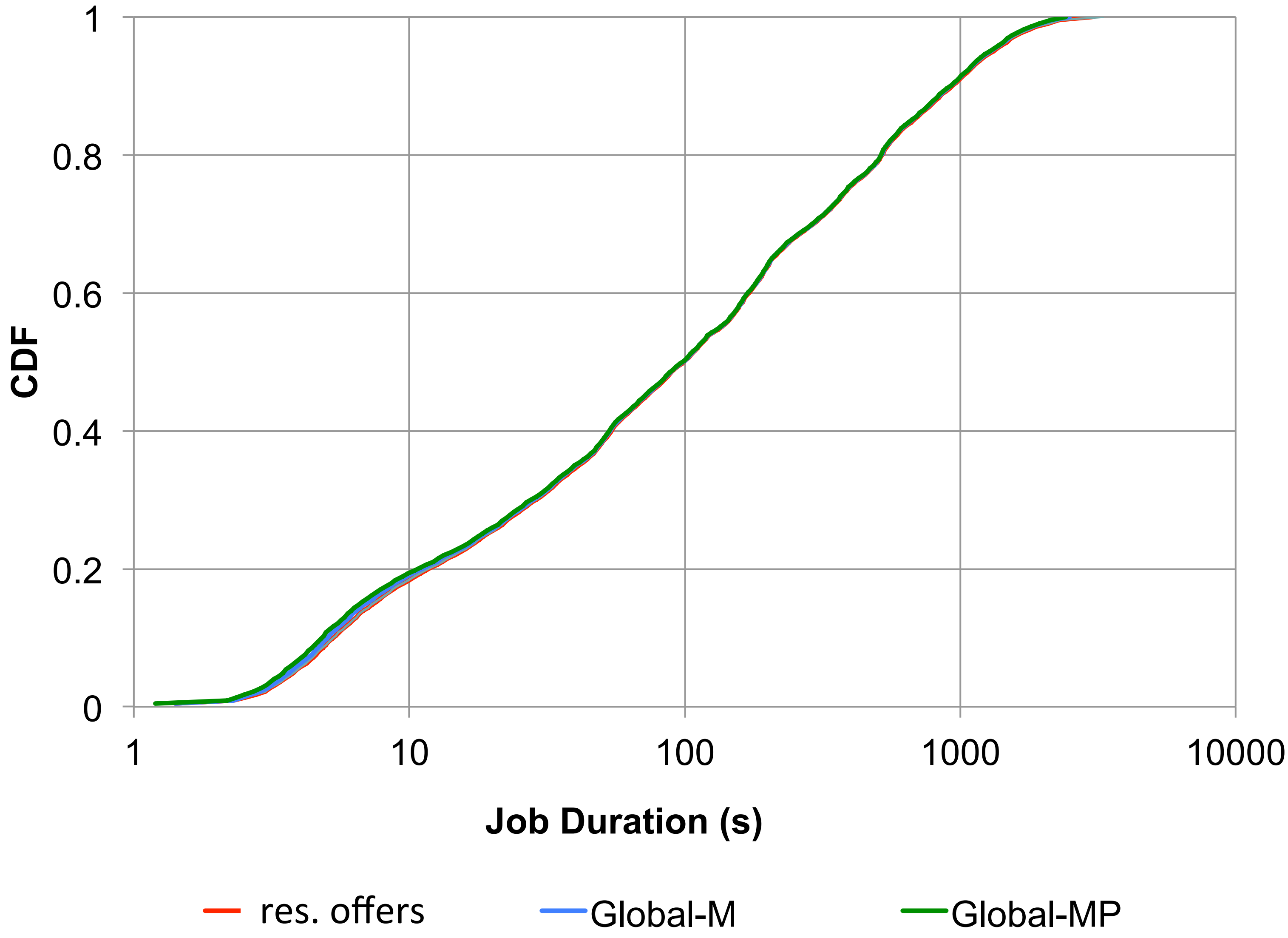
- Existing frameworks implement
  - **No** migration: expensive to migrate short tasks
  - Preemption with task killing (e.g., Dryad's Quincy): expensive to checkpoint data-intensive tasks

Macro-benchmark

- Simulate an 1000-node cluster
  - Job and task durations: Facebook traces (Oct 2010)
  - Constraints: modeled after Google*
- Allocation policy: fair sharing
- Scheduler comparison
  - **Resource Offers**: no preemption, and no migration (e.g., Hadoop's Fair Scheduler + constraints)
  - **Global-M**: global scheduler with migration
  - **Global-MP**: global scheduler with migration and preemption

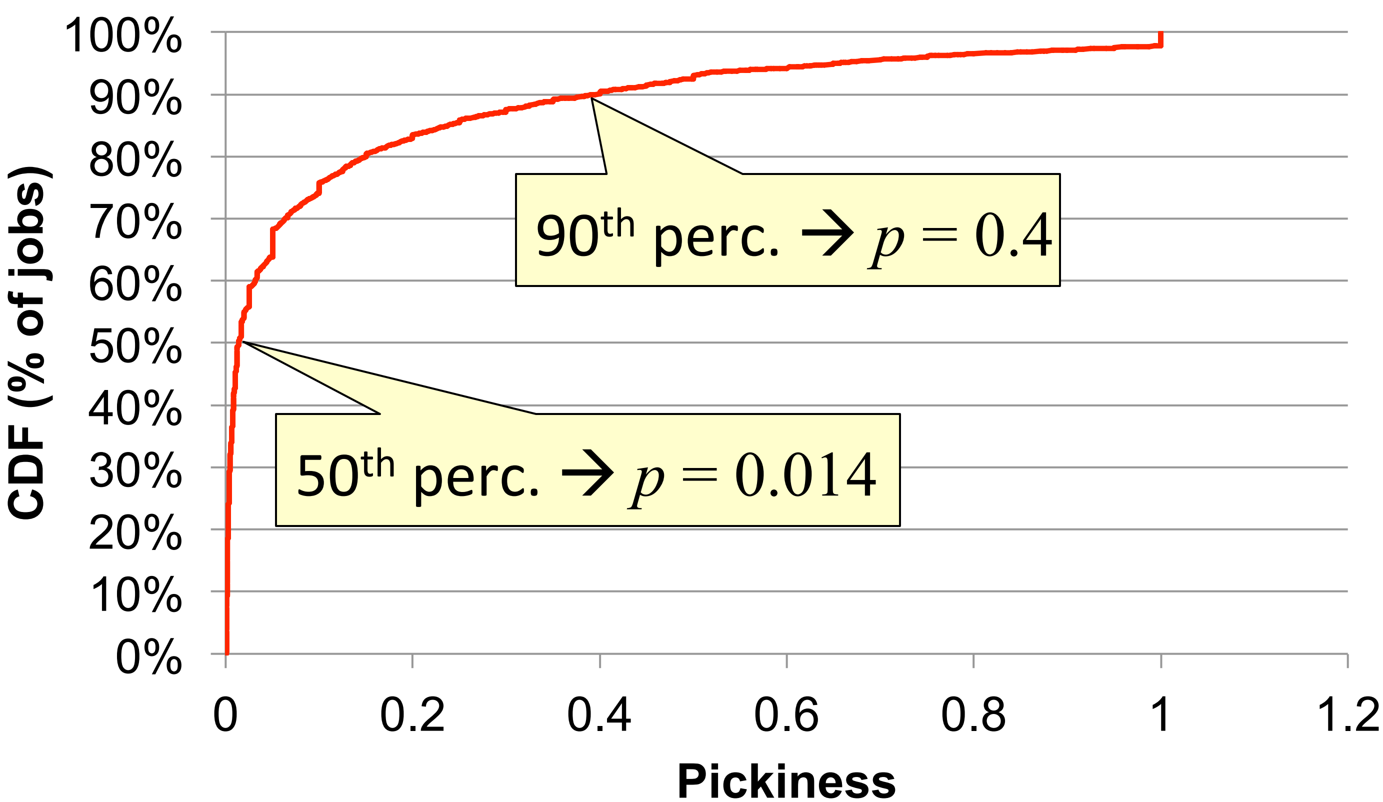
*Sharma et al., "Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters", ACM SoCC, 2011.

Facebook: Job Completion Times



Facebook: Pickiness

- Average cluster utilization: 82%
  - Much higher than at Facebook, which is < 50%
- Mean pickiness: 0.11



Summary: Resource Offers

- Ramp-up time low under most scenarios
- Barely any performance differences between global and distributed schedulers in Facebook workload
- Optimizations
  - Master doesn't send an offer already rejected by a framework (negative caching)
  - Allow frameworks to specify white and black lists of nodes

Borg

Borg

Cluster management system at Google that achieves high utilization by:

- Admission control
- Efficient task-packing
- Over-commitment
- Machine sharing

The User Perspective

- Users: Google developers and system administrators mainly
- The workload: Production and batch, mainly
- Cells, around 10K nodes
- Jobs and tasks

The User Perspective

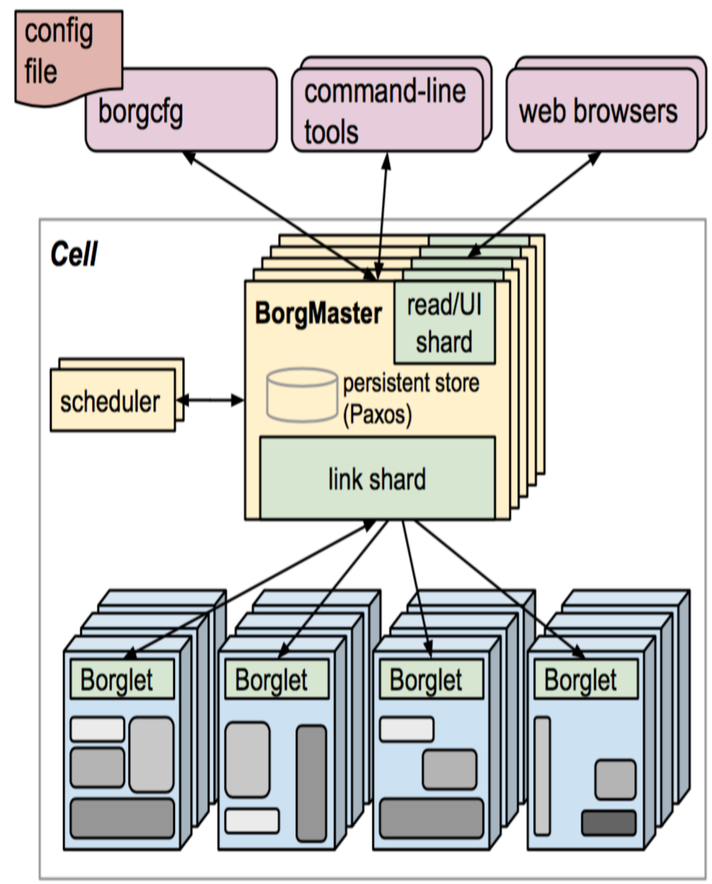
- Allocs
  - Reserved set of resources
- Priority, Quota, and Admission Control
  - Job has a priority (preempting)
  - Quota is used to decide which jobs to admit for scheduling
- Naming and Monitoring
  - 50.jfoo.ubar.cc.borg.google.com
  - Monitoring health of the task and thousands of performance metrics

Scheduling a Job

```
job hello_world = {
  runtime = { cell = “ic” } //what cell should run it in?
  binary = ‘../hello_world_webserver’ //what program to run?
  args = { port = ‘%port%’ }
  requirements = {
    RAM = 100M
    disk = 100M
    CPU = 0.1
  }
  replicas = 10000
}
```

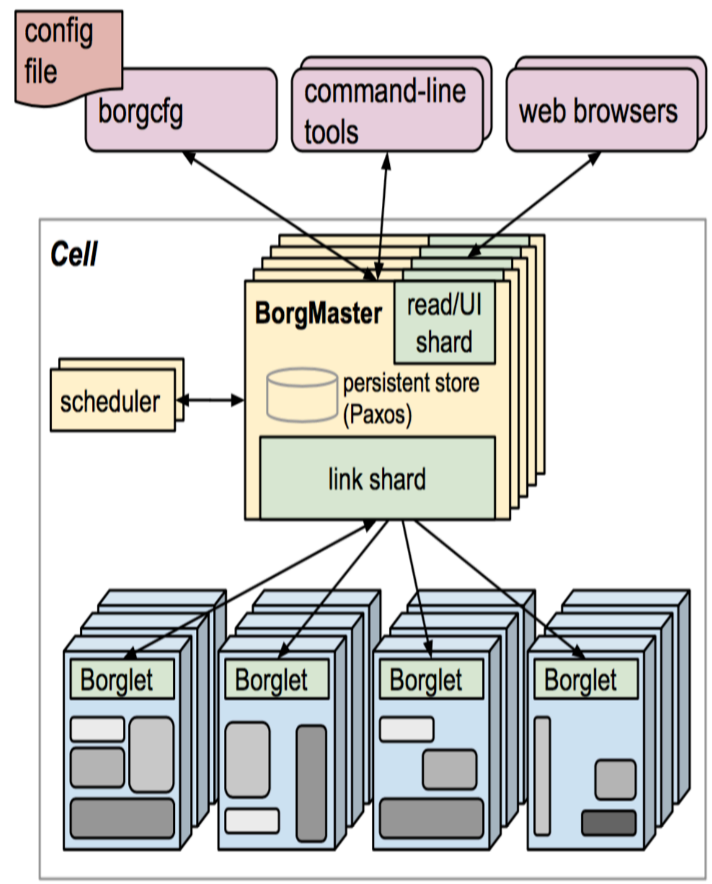
Borg Architecture

- Borgmaster
  - Main Borgmaster process & Scheduler
  - Five replicas
- Borglet
  - Manage and monitor tasks and resource
  - Borgmaster polls Borglet every few seconds



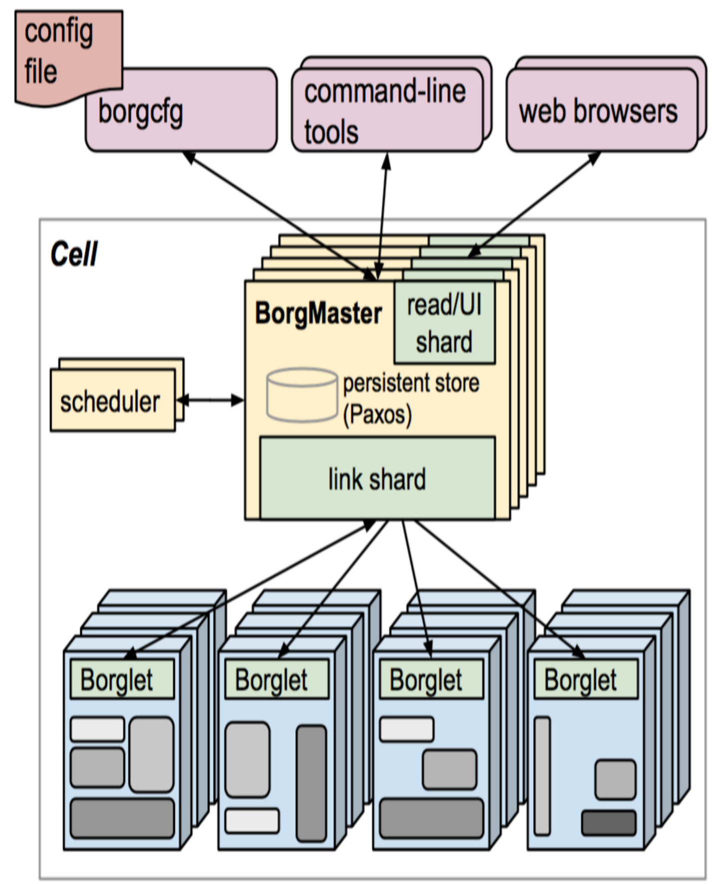
Borg Architecture

- Fauxmaster: high-fidelity Borgmaster simulator
  - Simulate previous runs from checkpoints
  - Contains full Borg code
- Used for debugging, capacity planning, evaluate new policies and algorithms



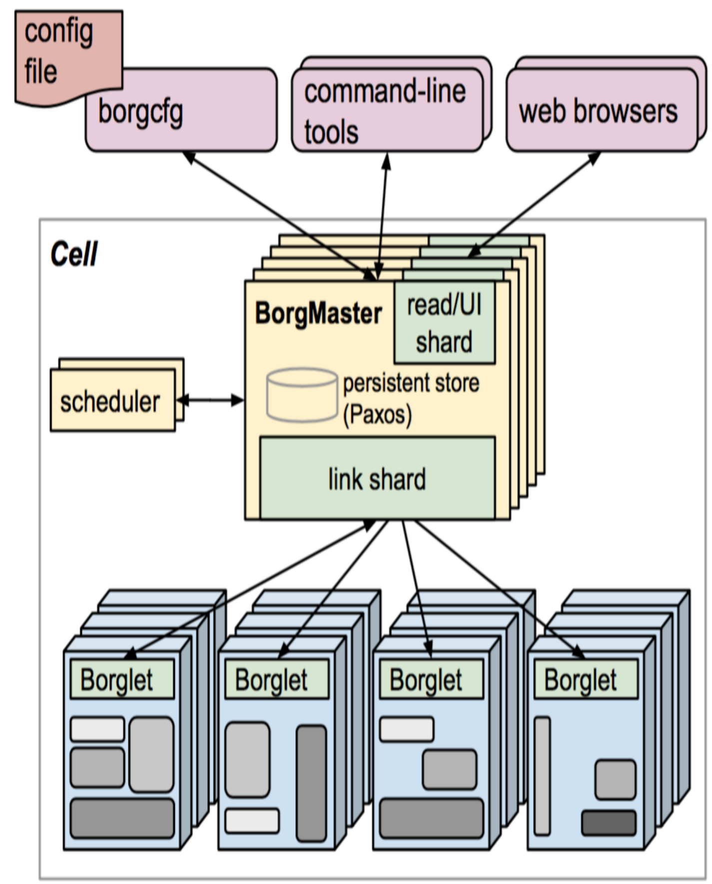
Scalability

- Separate scheduler
- Separate threads to poll the Borglets
- Partition functions across the five replicas
- Score caching
- Equivalence classes
- Relaxed randomization



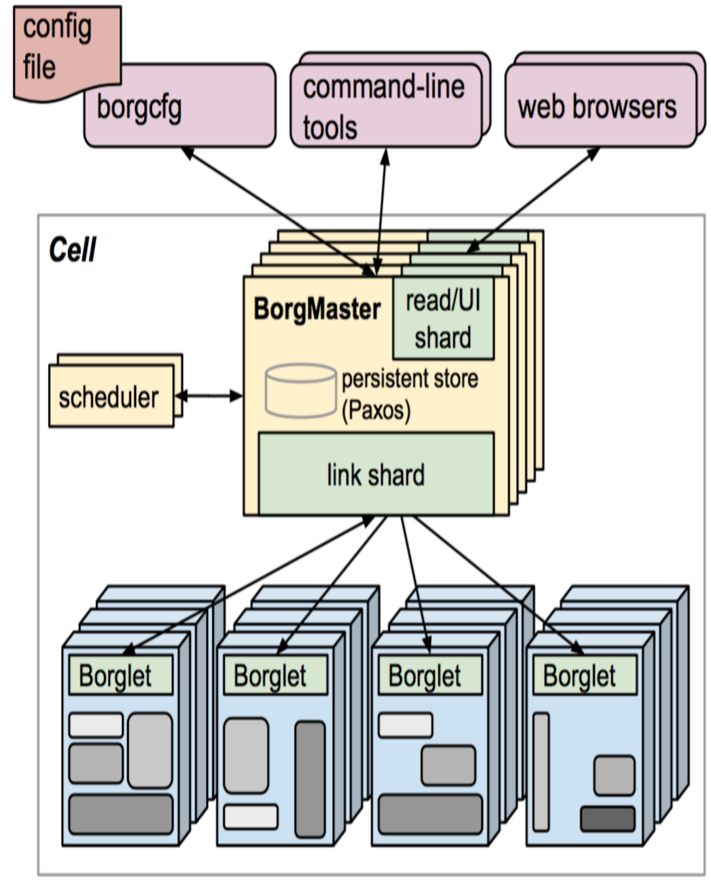
Scheduling

- feasibility checking: find machines for a given job
- Scoring: pick one machines
  - User prefs & build-in criteria
    - Minimize the number and priority of the preempted tasks
    - Picking machines that already have a copy of the task's packages
    - spreading tasks across power and failure domains
    - Packing by mixing high and low priority tasks



Scheduling

- Feasibility checking: find machines for a given job
- Scoring: pick one machines
  - User prefs & build-in criteria
  - E-PVM (Enhanced-Parallel Virtua Machine) vs best-fit
    - Hybrid approach



Borg's Allocation Algorithms and Policies

Advanced Bin-Packing algorithms:

- Avoid stranding of resources

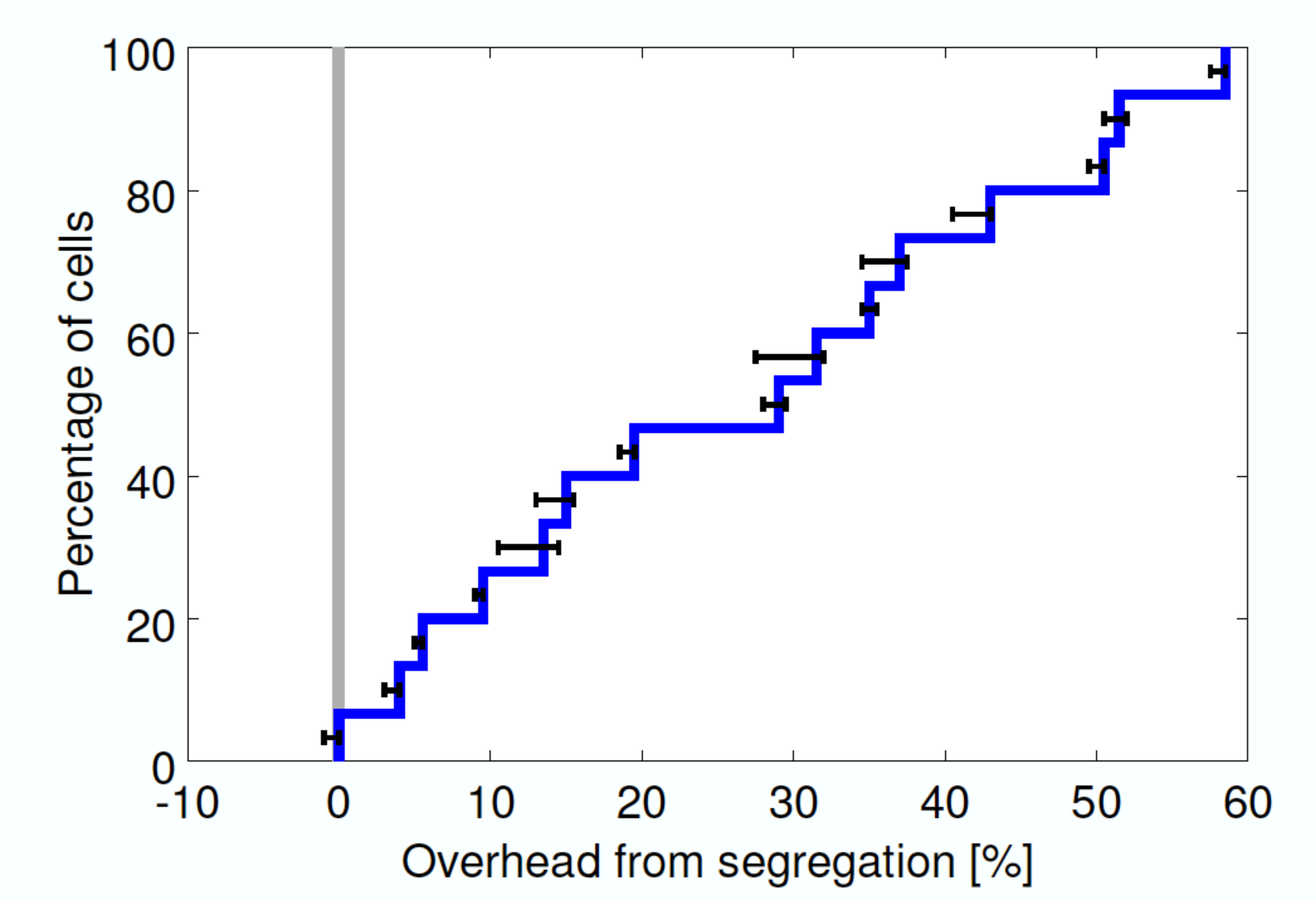
Evaluation metric: Cell-compaction

- Find smallest cell that we can pack the workload into…
- Remove machines randomly from a cell to maintain cell heterogeneity

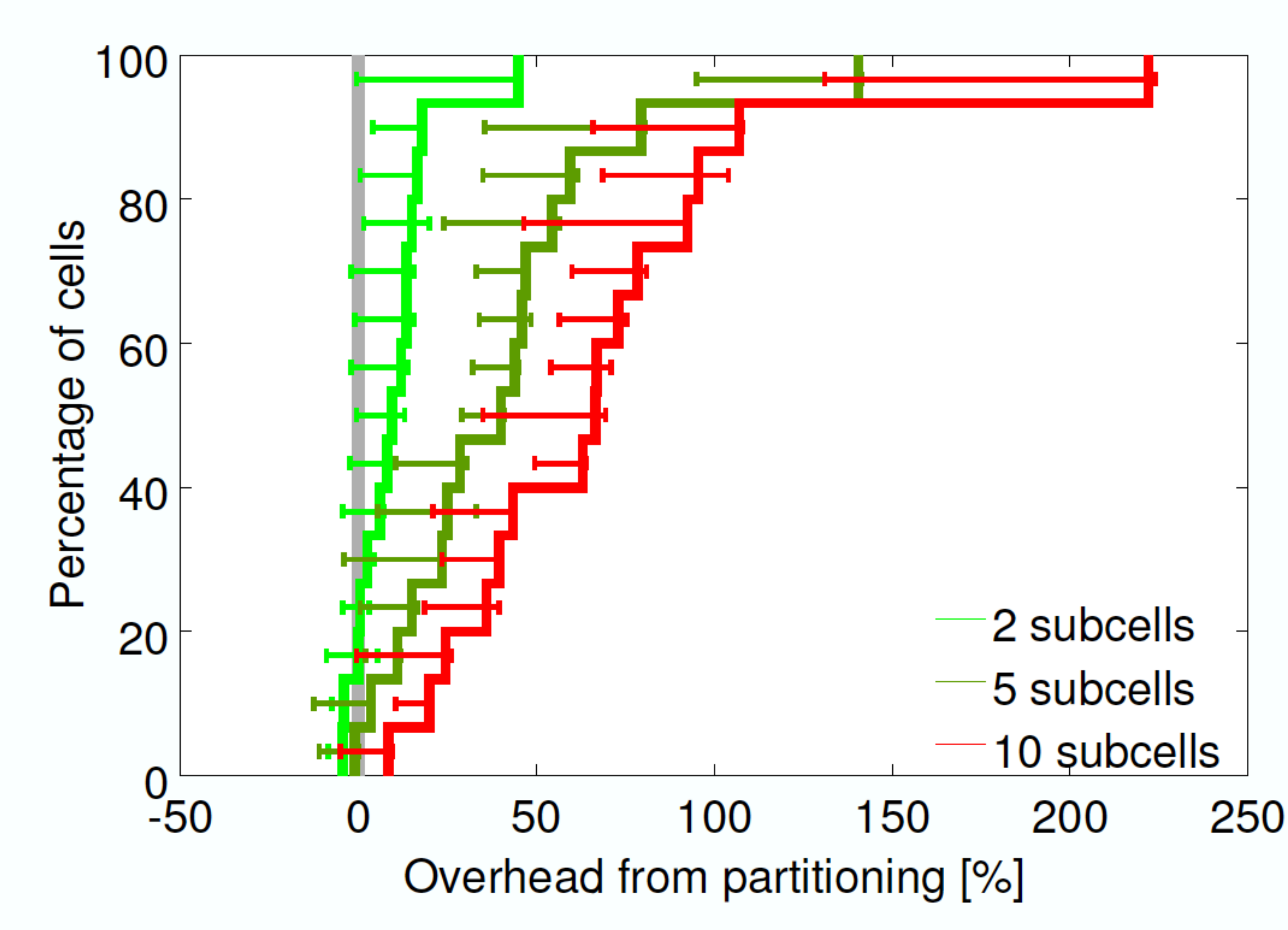
Evaluated various policies to understand the cost, in terms of extra machines needed for packing the same workload

Should we Share Clusters…

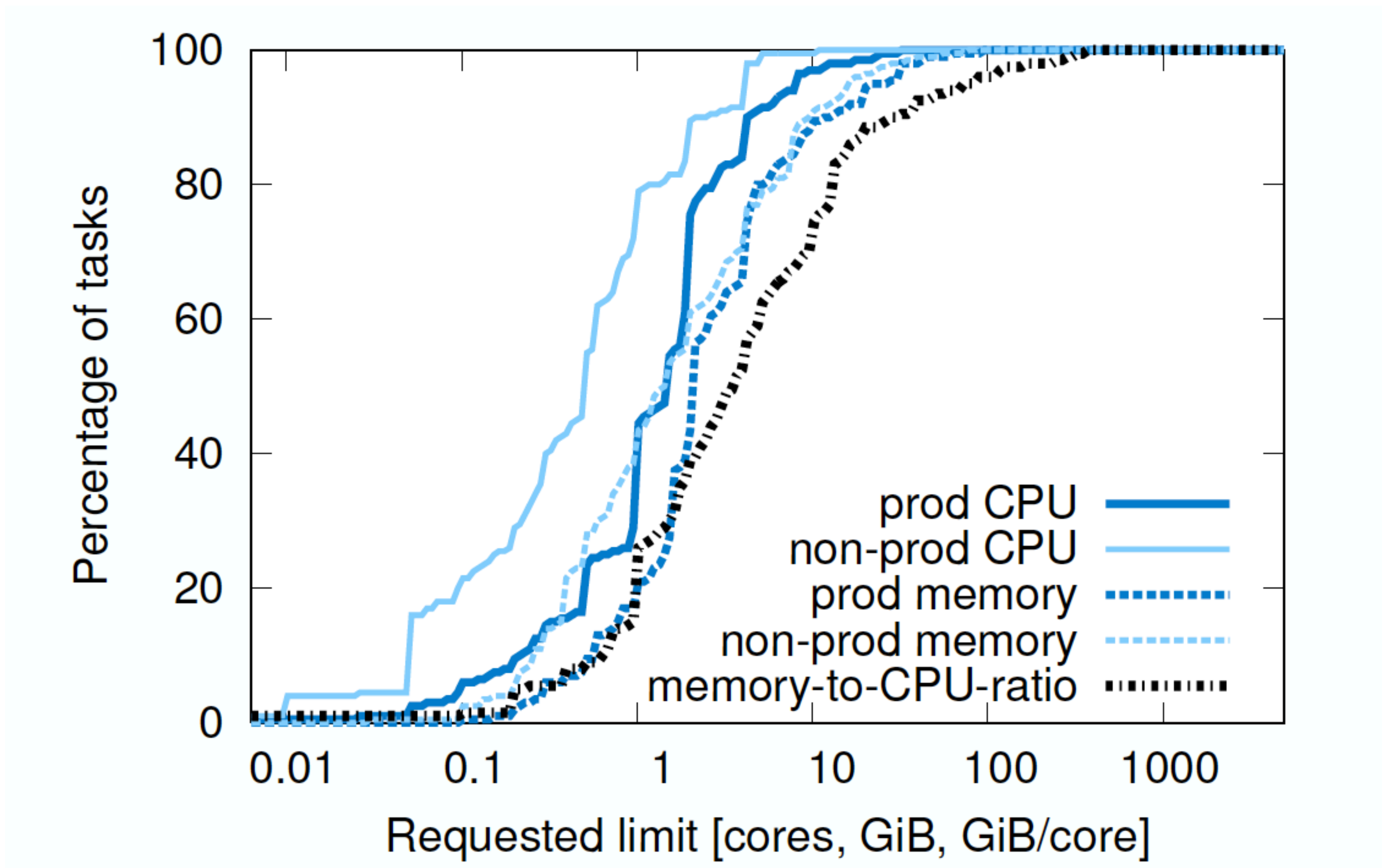
- … between production and non-production jobs?



Should we use Smaller Cells?



Would fixed resource bucket sizes be better?



Kubernetes

Google open source project loosely inspired by Borg

Directly derived

- Borglet => Kubelet
- alloc => pod
- Borg containers => docker
- Declarative specifications

Improved

- Job => labels
- managed ports => IP per pod
- Monolithic master => micro-services