Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization

Wei Chen, Jia Rao*, and Xiaobo Zhou
University of Colorado, Colorado Springs
* University of Texas at Arlington

University of Colorado Colorado Springs

UNIVERSITY OF TEXAS ARLINGTON

Data Center Computing

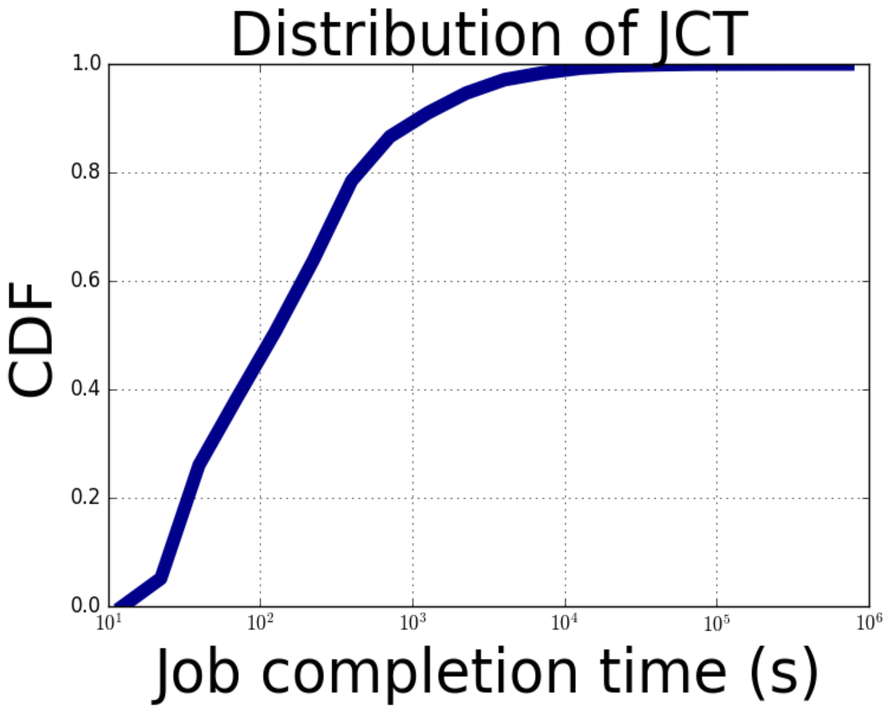
- Challenges
  - Increase hardware utilization and efficiency
  - Meet SLOs
- Heterogeneous workloads
  - Diverse resource demands
    - ✓ Short jobs v.s. long jobs
  - Different QoS requirements
    - ✓ Latency v.s. throughput

Data Center Computing

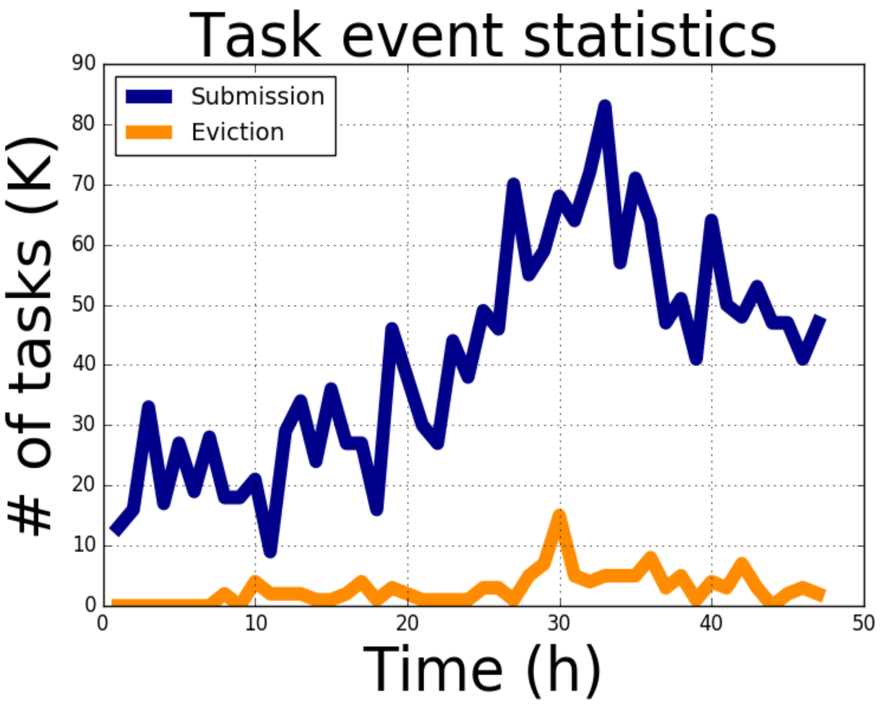
- Challenges
  - Increase hardware utilization and efficiency
  - Meet SLOs
- Heterogeneous workloads
  - Diverse resource demands
    - ✓ Short jobs v.s. long jobs
  - Different QoS requirements
    - ✓ Latency v.s. throughput

Long jobs help improve hardware utilization while short jobs are important to QoS

Data Center Trace Analysis



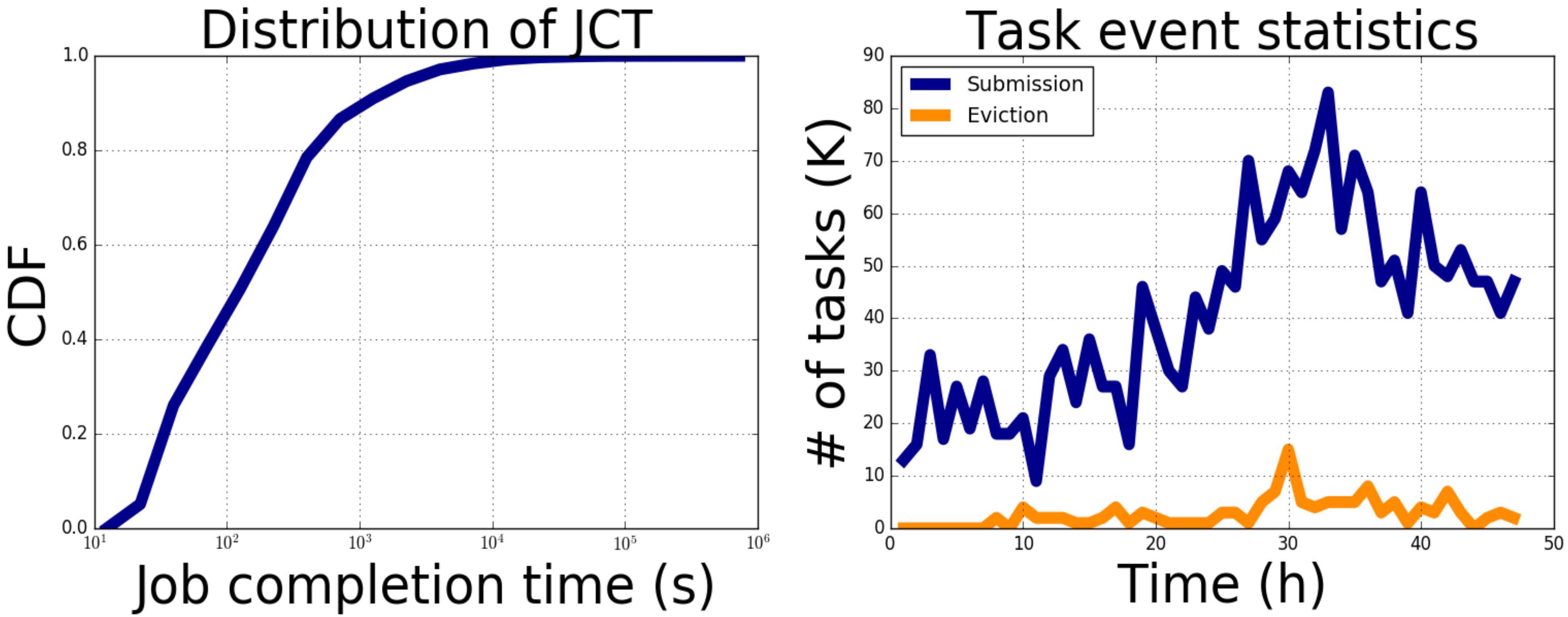
10% long jobs account
for 80% resource usage



Tasks are evicted if encountering
resource shortage

Google traces (https://github.com/google/cluster-data)

Data Center Trace Analysis



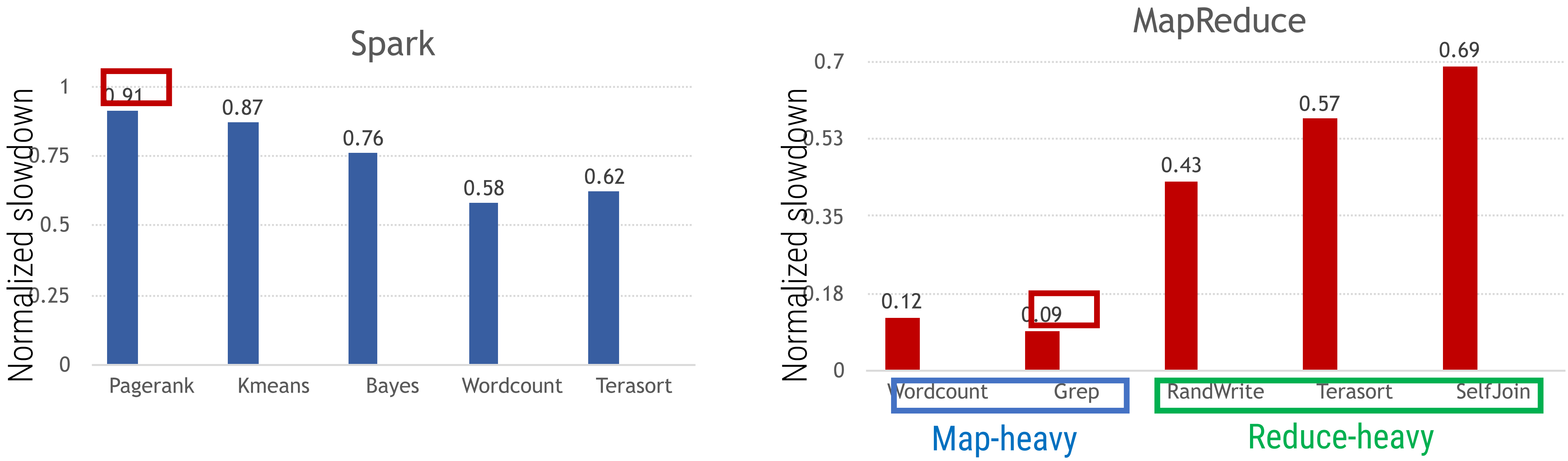
10% long jobs account for 80% resource usage

Tasks are evicted if encountering resource shortage

Short jobs have higher priority and most preempted (evicted) tasks belong to long jobs

Google traces (https://github.com/google/cluster-data)

Overhead of Kill-based Preemption



1. MapReduce jobs experience various degrees of slowdowns
2. Spark jobs suffer from more slowdowns due to frequent inter-task synchronization and the re-computation of failed RDDs

Our Approach

- **Container-based task preemption**
  - Containerize tasks using *docker* and control resource via `cgroup`
  - Task preemption without losing the execution progress
    - ✓ Suspension: reclaim resources from a preempted task
    - ✓ Resumption: re-activate a task by restoring its resource
- **Preemptive fair share scheduler**
  - Augment the capacity scheduler in YARN with preemptive task scheduling and fine-grained resource reclamation

Related Work

- Optimizations for heterogeneous workloads
  - YARN [SoCC'13]: kill long jobs if needed — Long job slowdown and resource waste ✗
  - Sparrow [SOSP'13]: decentralized scheduler for short jobs — No mechanism for preemption ✗
  - Hawk [ATC'15]: hybrid scheduler based on reservation — Hard to determine optimal reservation ✗
- Task preemption
  - Natjam [SoCC'13], Amoeba [SoCC'12]: proactive checkpointing — Hard to decide frequency ✗
  - CRIU [Middleware'15]: on-demand checkpointing — Application changes required ✗
- Task containerization
  - Google Borg [EuroSys'15]: mainly for task isolation — Still kill-based preemption ✗

Related Work

- Optimizations for heterogeneous workloads
  - YARN [SoCC'13]: kill long jobs if needed — Long job slowdown and resource waste ✗
  - Sparrow [SOSP'1 … chanism for preemption ✗
  - Hawk [ATC'15]: h … ine optimal reservation ✗
- Task preemptio…
  - Natjam [SoCC'1… … rd to decide frequency ✗
  - CRIU [Middleware'15]: on-demand checkpointing — Application changes required ✗
- Task containerization
  - Google Borg [EuroSys'15]: mainly for task isolation — Still kill-based preemption ✗

If short jobs can timely preempt long jobs

✓ **No need for cluster reservation**

✓ **Preserving long job's progress**

✓ **Application agnostic**

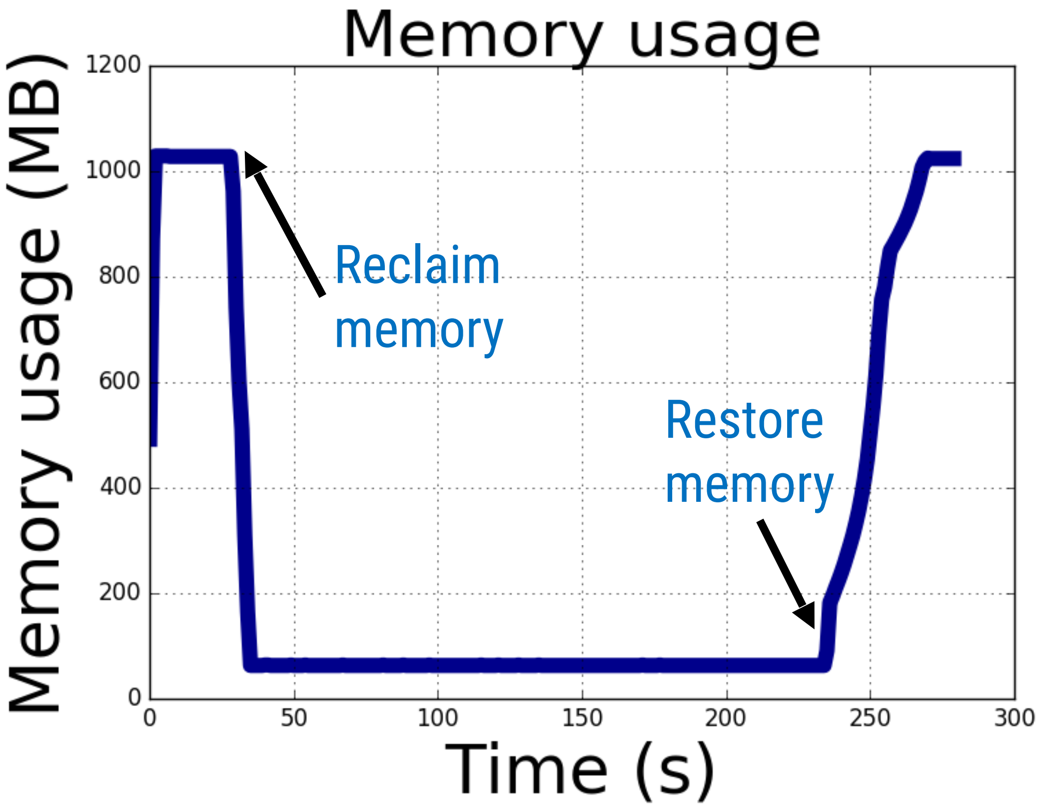
✓ **Fine-grained resource management**

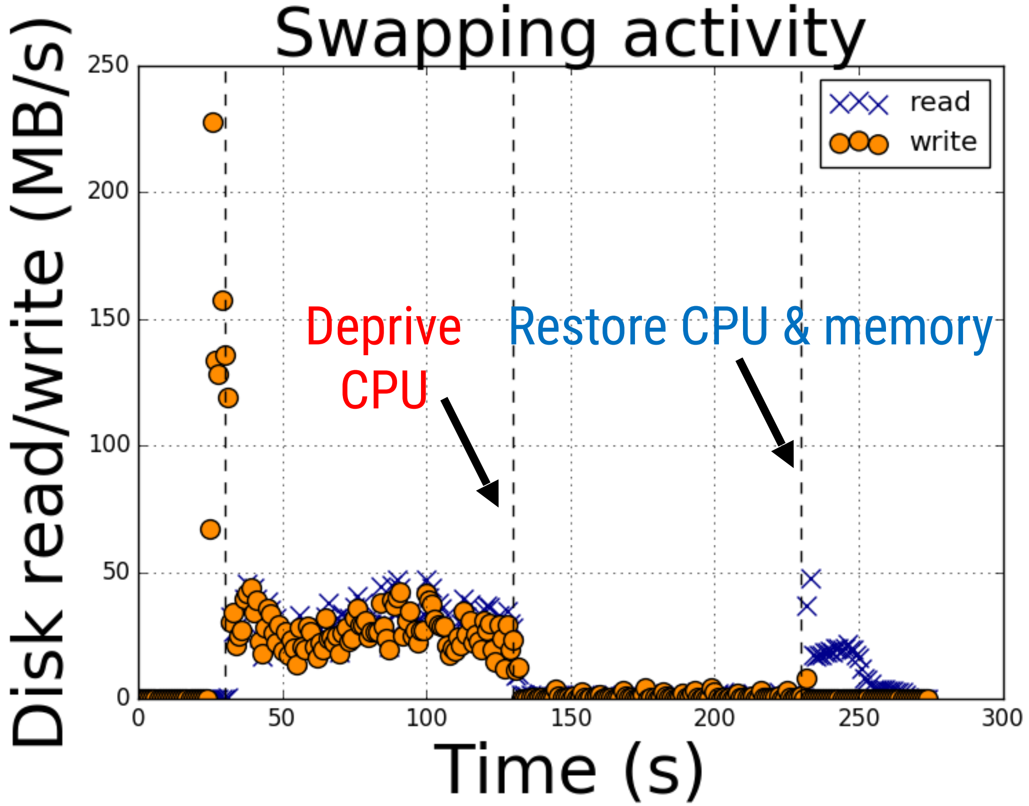
Container-based Task Preemption

- Task containerization
  - Launch tasks in *Docker* containers
  - Use `cgroup` to control resource allocation, i.e., CPU and memory
- Task suspension
  - Stop task execution: deprive task of CPU
  - Save task context: reclaim container memory and write dirty memory pages onto disk
- Task resumption
  - Restore task resources

Task Suspension and Resumption

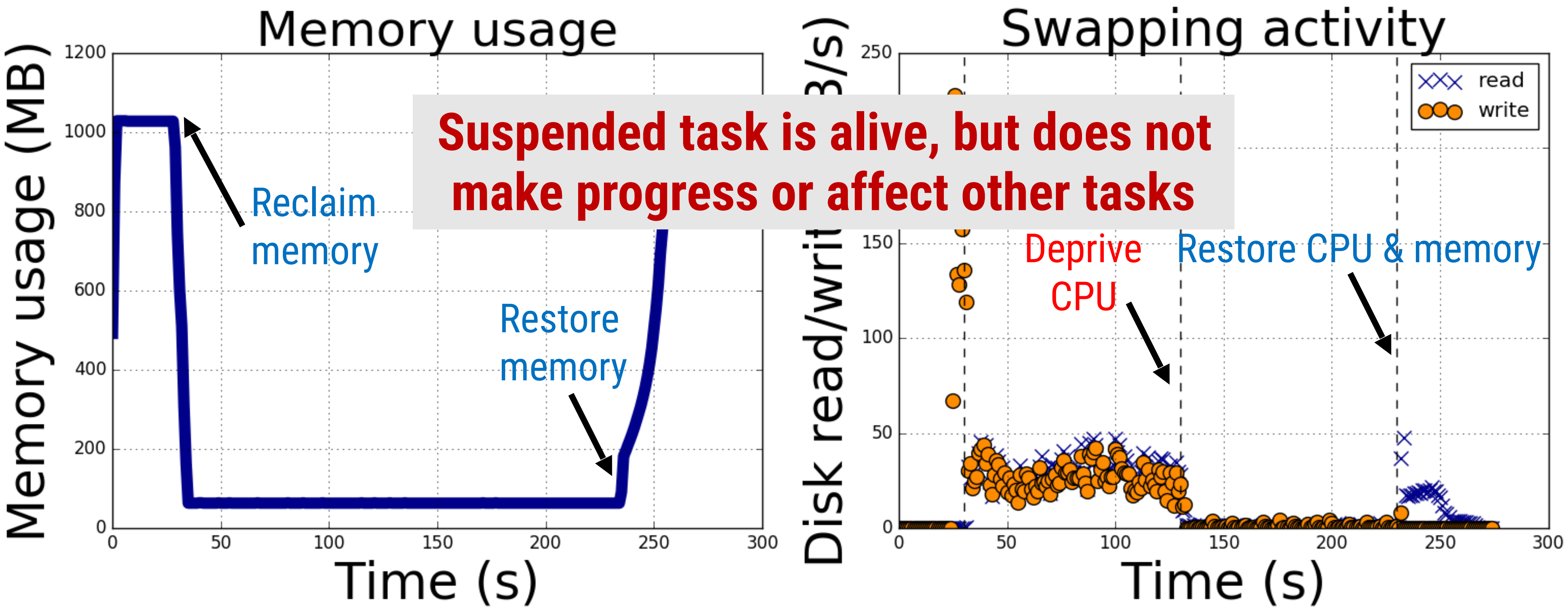
Keep a minimum footprint for a preempted task: 64MB memory and 1% CPU





Task Suspension and Resumption

Keep a minimum footprint for a preempted task: 64MB memory and 1% CPU



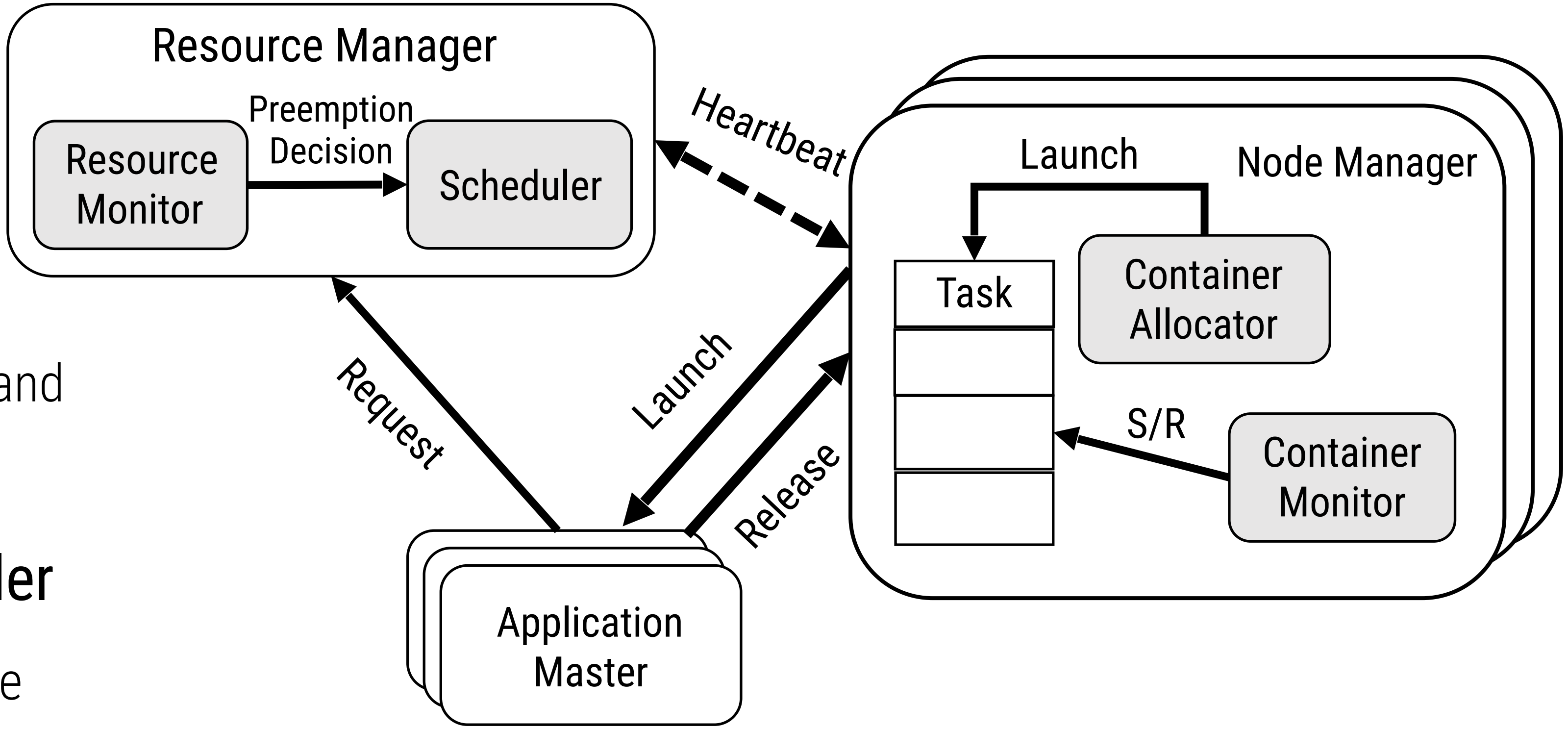
Suspended task is alive, but does not make progress or affect other tasks

Two Types of Preemption

- Immediate preemption (IP)
  - Reclaims all resources of a preempted task in **one** pass
  - **Pros:** simple, fast reclamation
  - **Cons:** may reclaim more than needed, incur swapping, and cause long reclamation
- Graceful preemption (GP)
  - Shrinks a preempted task and reclaims its resources in **multiple** passes, at a step of $\vec{r} = (c, m)$
  - **Pros:** fine-grained reclamation, avoid swapping
  - **Cons:** complicated, slow reclamation, tuning of step r needed

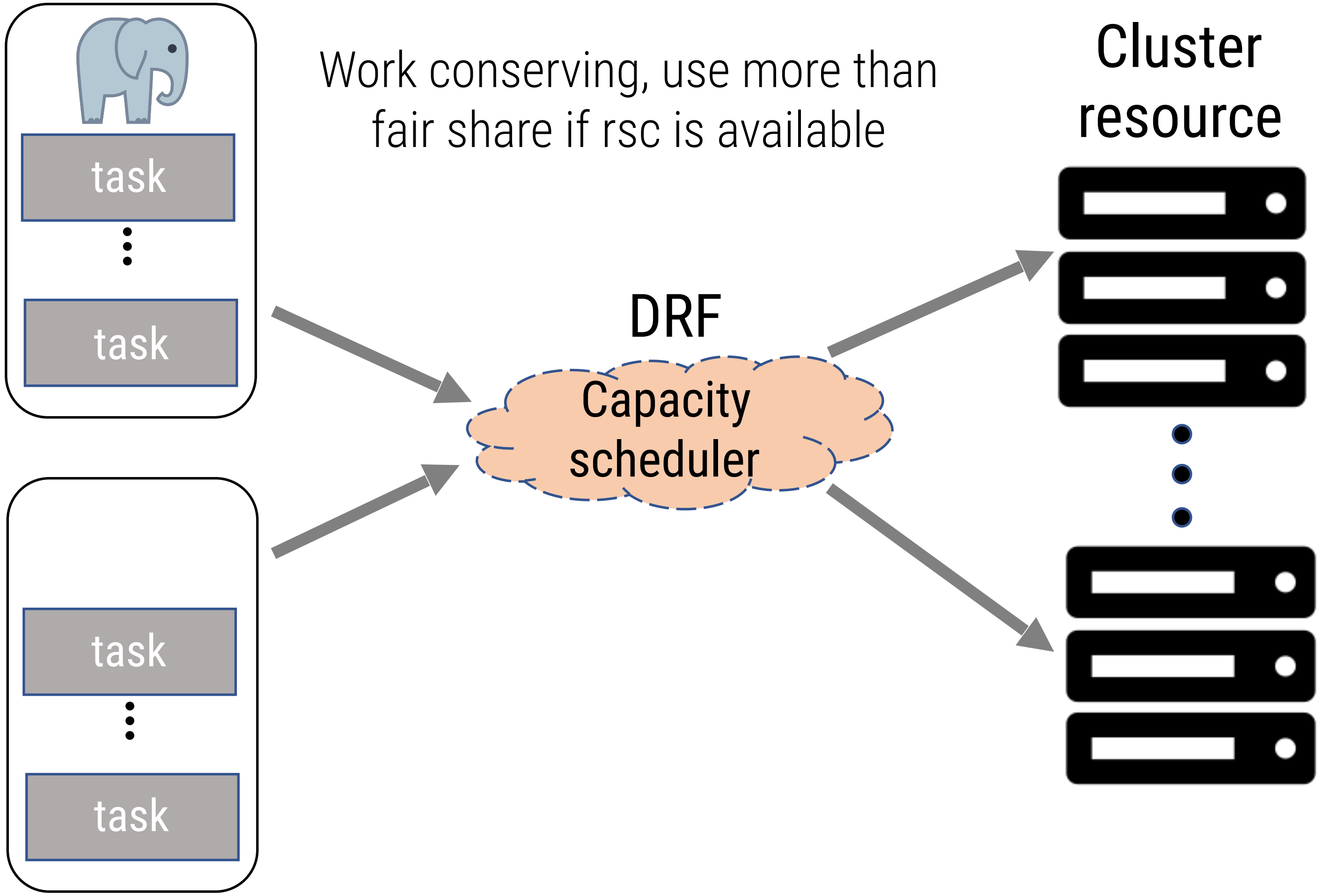
BIG-C: Preemptive Cluster Scheduling

- Container allocator
  - Replaces YARN's nominal container with docker
- Container monitor
  - Performs container suspend and resume (S/R) operations
- Resource monitor & Scheduler
  - Determine how much resource and which container to preempt

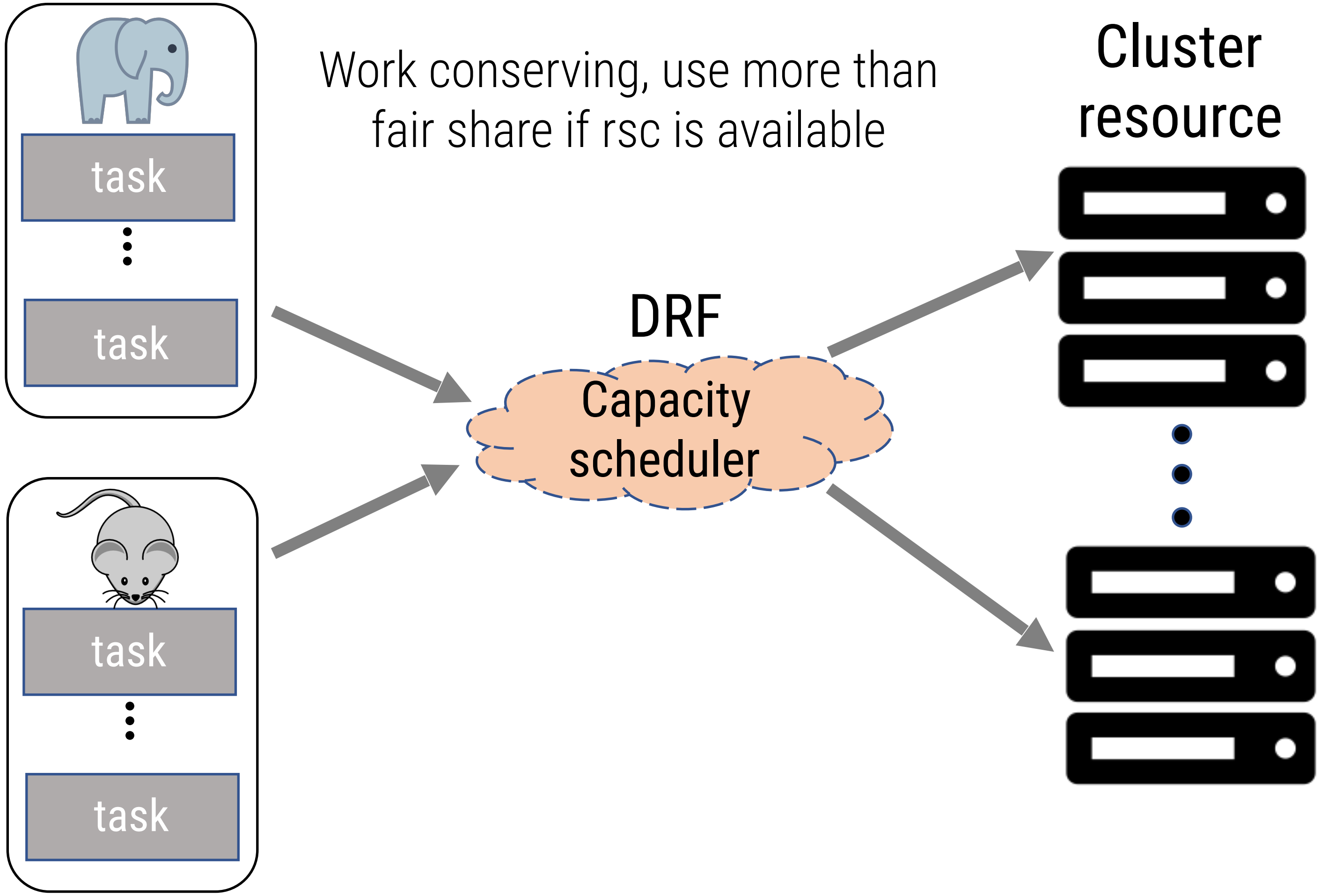


Source code available at https://github.com/yncxcw/big-c

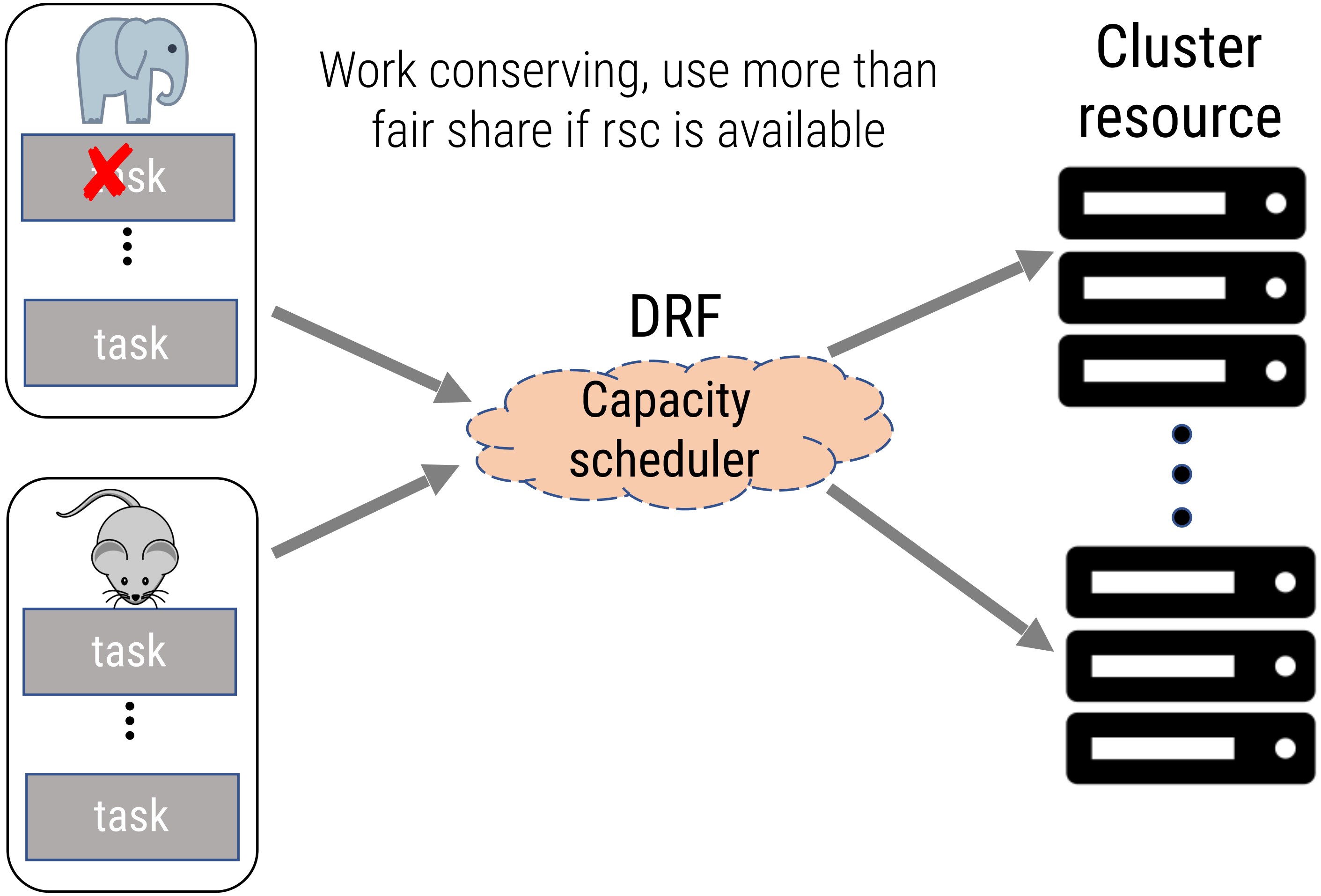
YARN's Capacity Scheduler



YARN's Capacity Scheduler



YARN's Capacity Scheduler



YARN's Capacity Scheduler

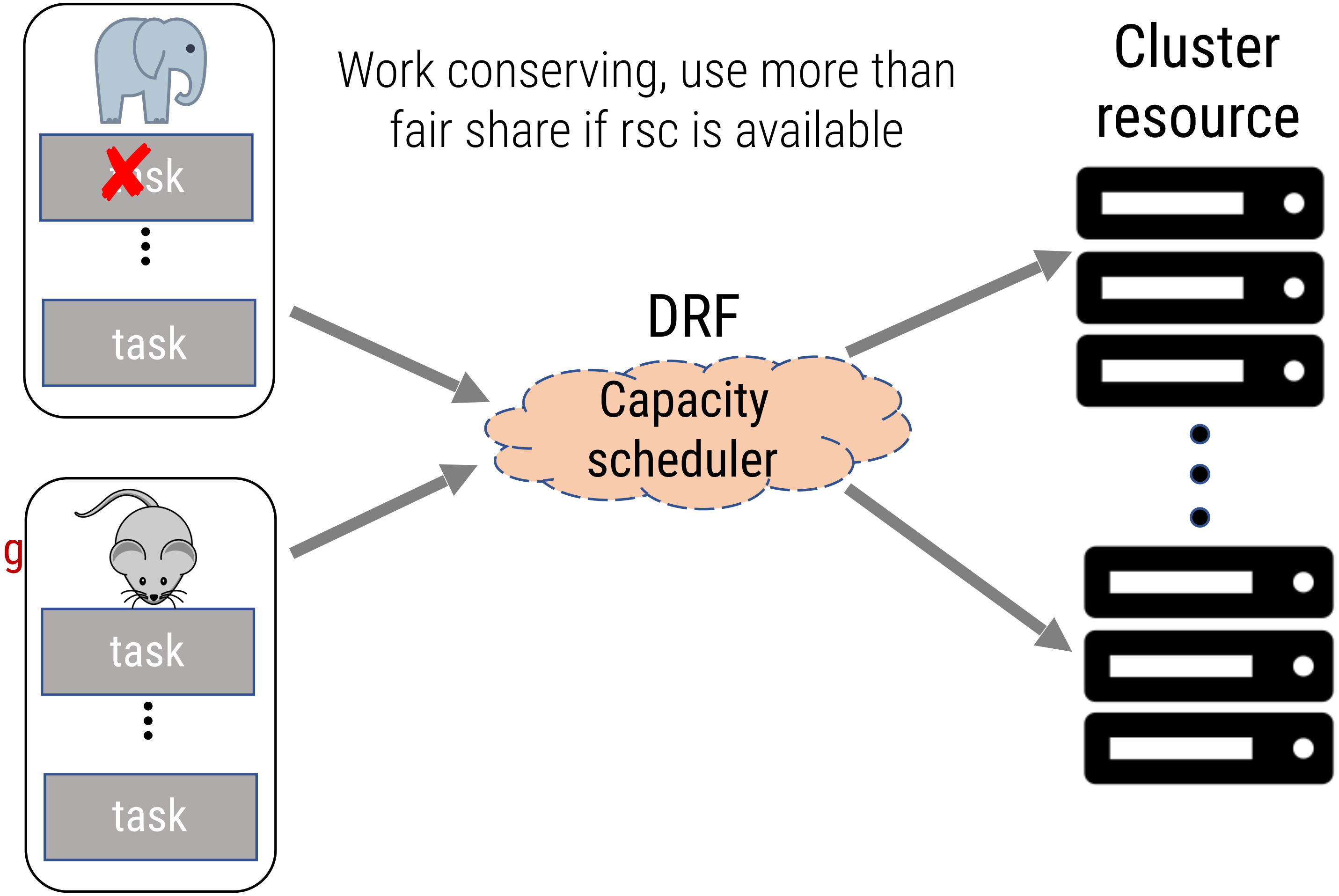
$\vec{r_l}$: long job demand
$\vec{f_l}$: long job fair share
$\vec{a}$: over-provisioned rsc
$\vec{r_s}$: short job demand
$\vec{p}$: rsc to preempt

$$\vec{a} = \vec{r_l} - \vec{f_l}$$

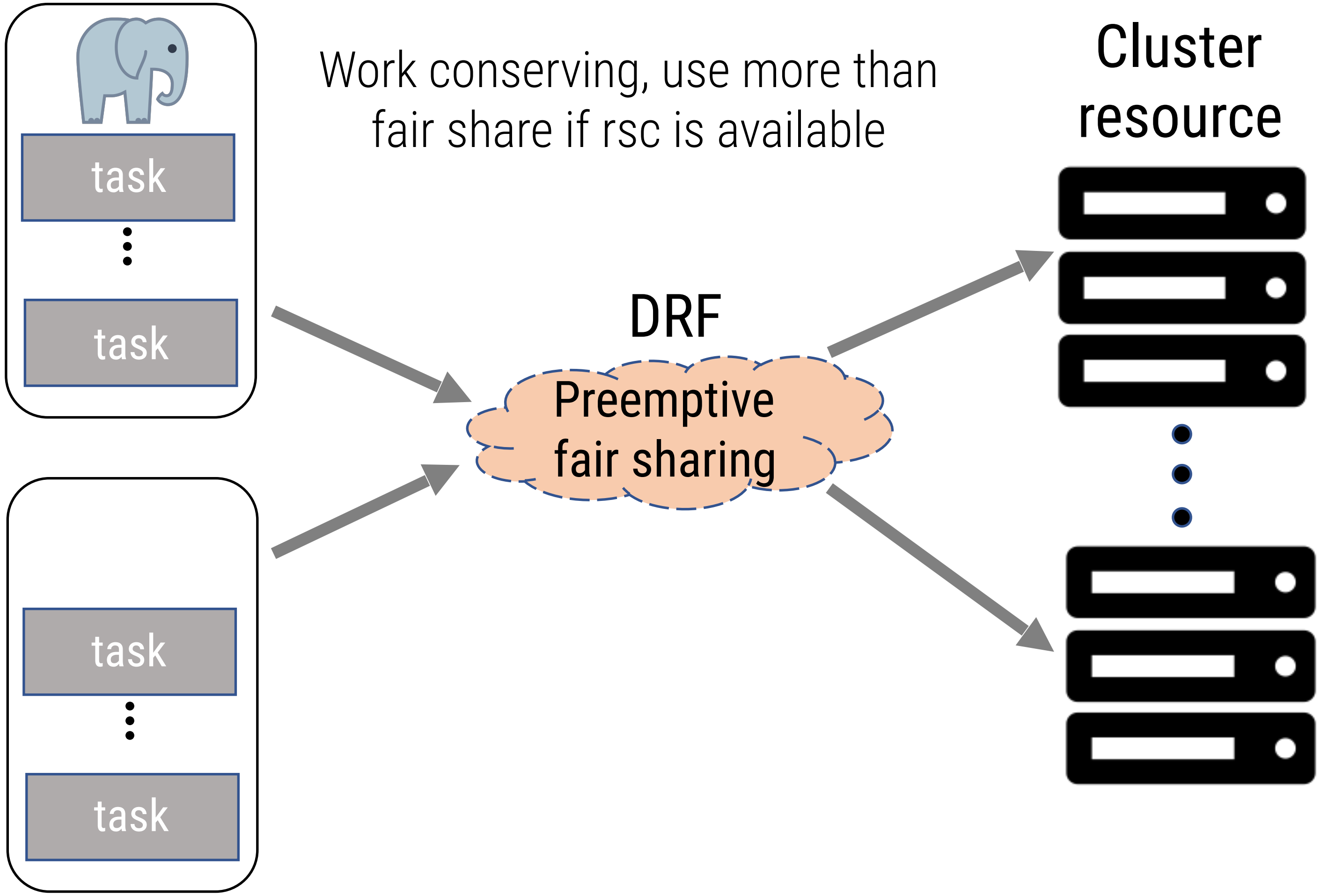
If $\vec{r_s} < \vec{a}$
$\quad \vec{p} = \vec{r_s}$
else
$\quad \vec{p} = \vec{a}$

- At least kill one long task

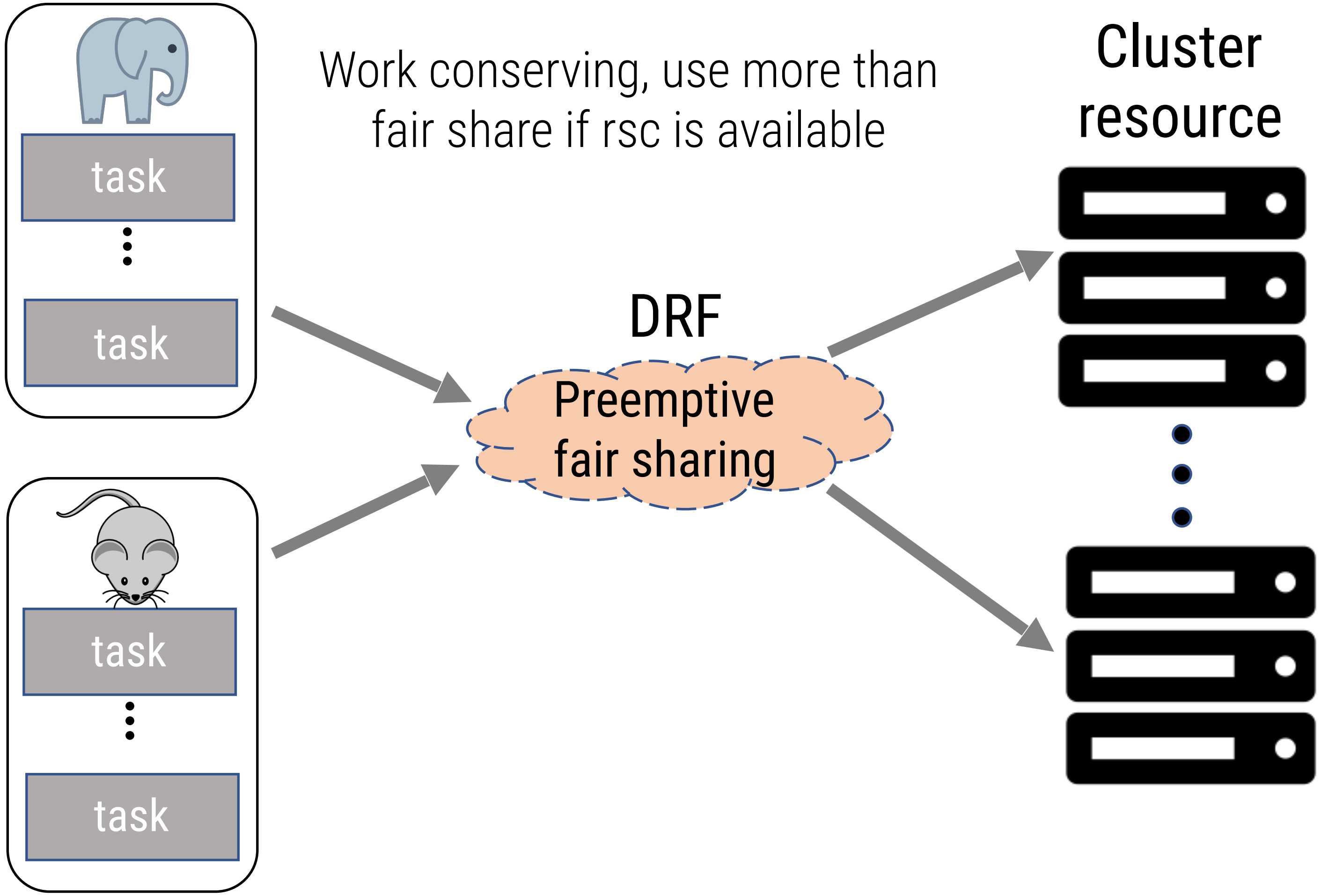
- Rsc reclamation does not enforce DRF



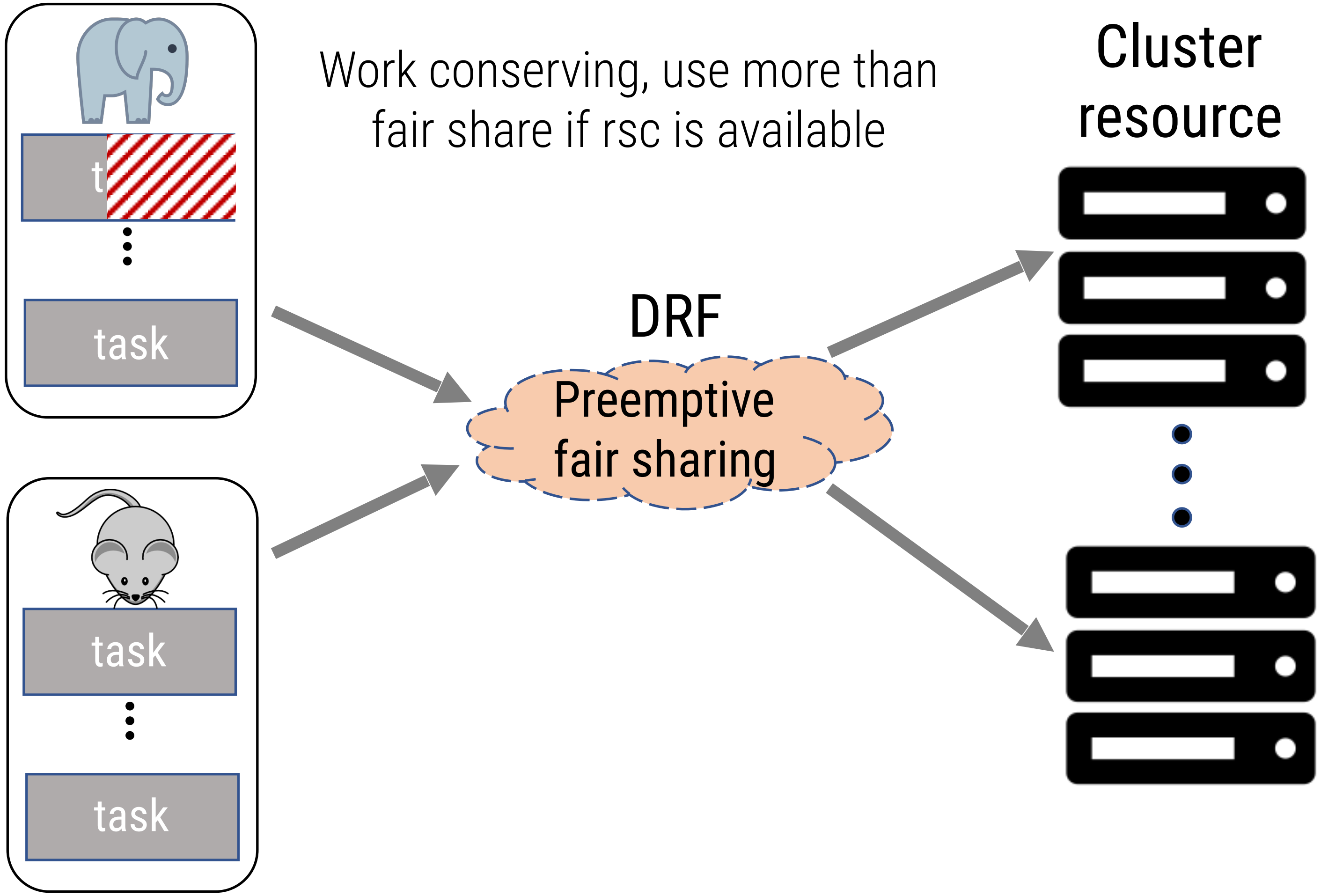
Preemptive Fair Share Scheduler



Preemptive Fair Share Scheduler



Preemptive Fair Share Scheduler



Capacity scheduler

VOID

Preemptive Fair Share Scheduler

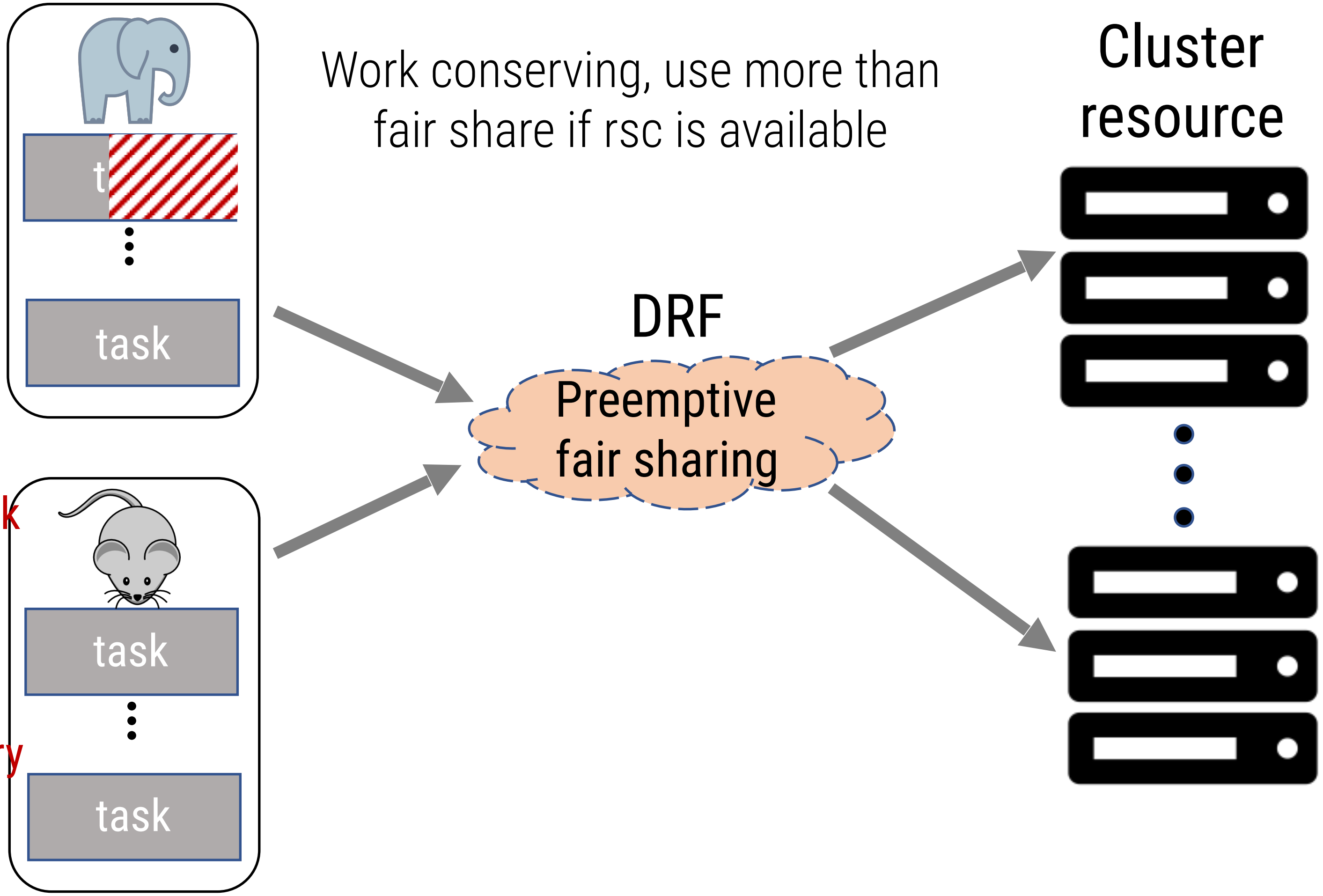
$\vec{r_l}$: long job demand
$\vec{f_l}$: long job fair share
$\vec{a}$: over-provisioned rsc
$\vec{r_s}$: short job demand
$\vec{p}$: rsc to preempt

$$\vec{a} = \vec{r_l} - \vec{f_l}$$

If $\vec{r_s} < \vec{a}$
$\quad \vec{p} = \vec{r_s}$
else
$\quad \vec{p} =$ **ComputeDR**$(\vec{r_l}, \vec{a})$

- Preempt part of task rsc

- Enforce DRF, avoid unnecessary reclamation



Compute DR at Task Preemption

If $\vec{r_s}$ = ⟨20CPU,10GB⟩ and $\vec{a}$ = ⟨10CPU,15GB⟩ , what is $\vec{p}$?

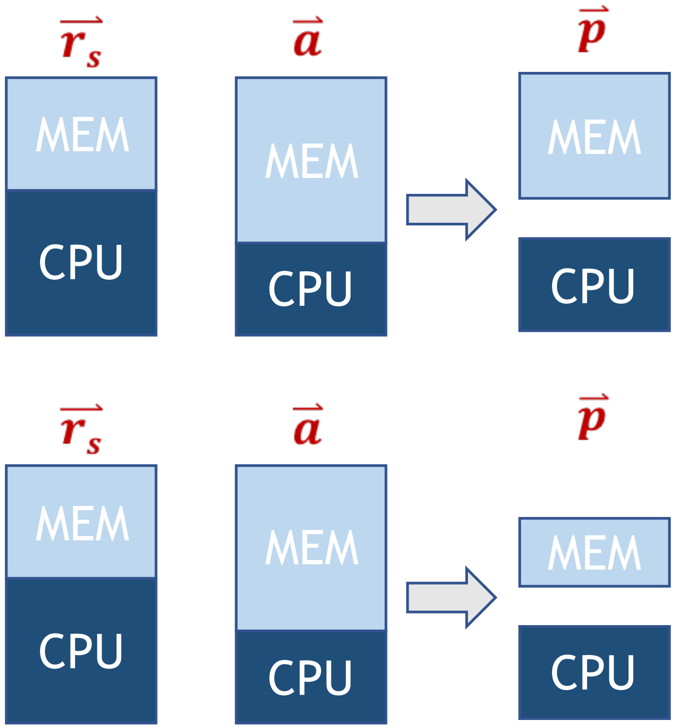
- Capacity scheduler

  $\vec{p}$= ⟨10CPU,10GB⟩

- Preemptive fair sharing

  $\vec{p}= \langle 10\text{CPU}, \frac{10\text{GB}}{20\text{CPU}} \times 10\text{GB} \rangle$

  = ⟨10CPU,5GB⟩



Compute DR at Task Preemption

If $\vec{r_s}$ = ⟨20CPU,10GB⟩ and $\vec{a}$ = ⟨10CPU,15GB⟩ , what is $\vec{p}$?

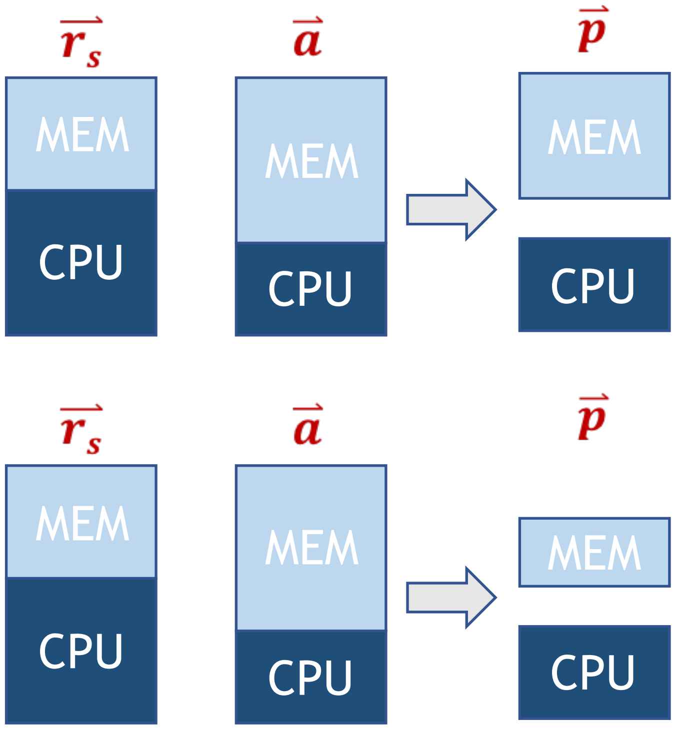
- Capacity scheduler

  $\vec{p}$= ⟨10CPU,10GB⟩

- Preemptive fair sharing

  $\vec{p}$= $\langle 10\text{CPU}, \frac{10\text{GB}}{20\text{CPU}} \times 10\text{GB} \rangle$

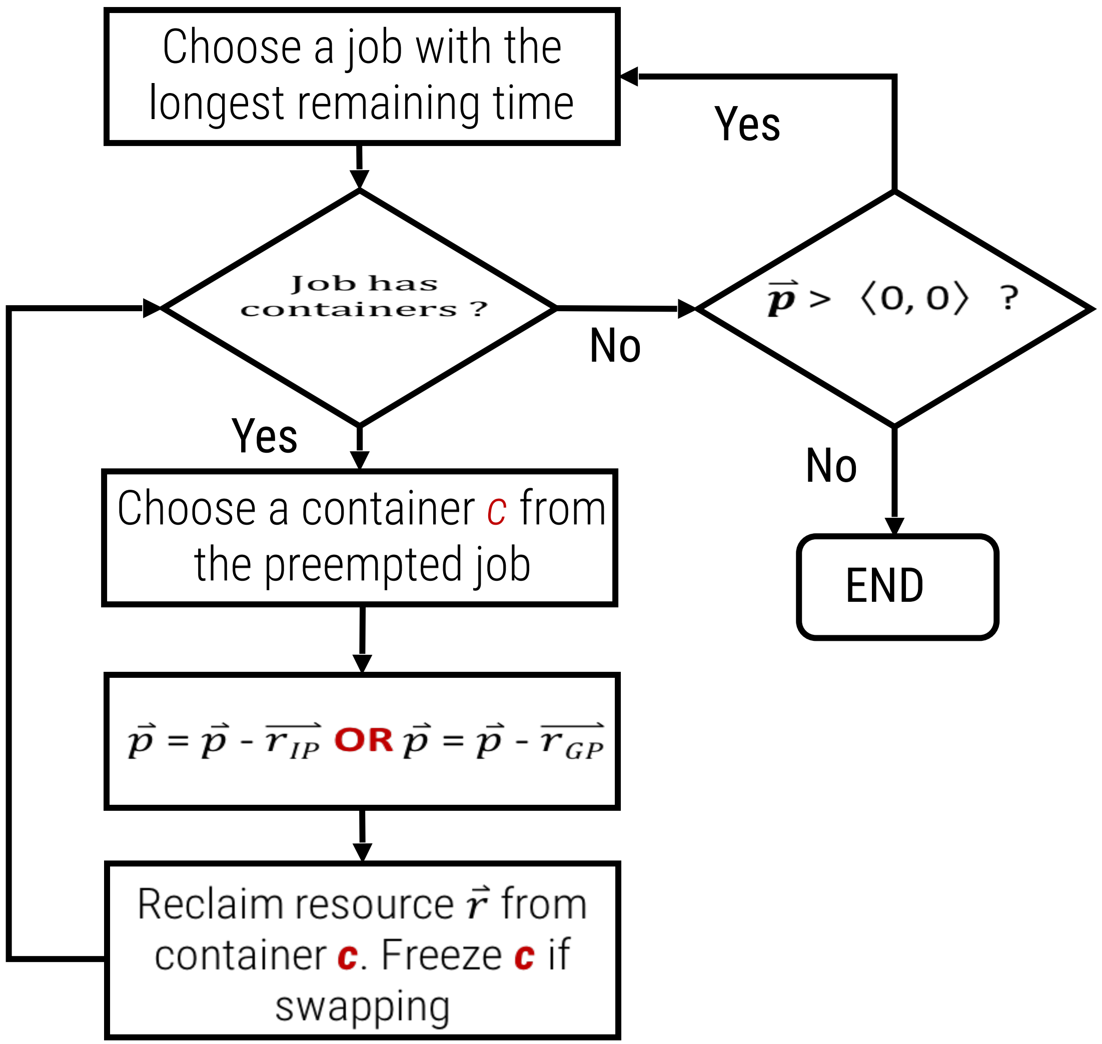
  $=$ ⟨10CPU,5GB⟩



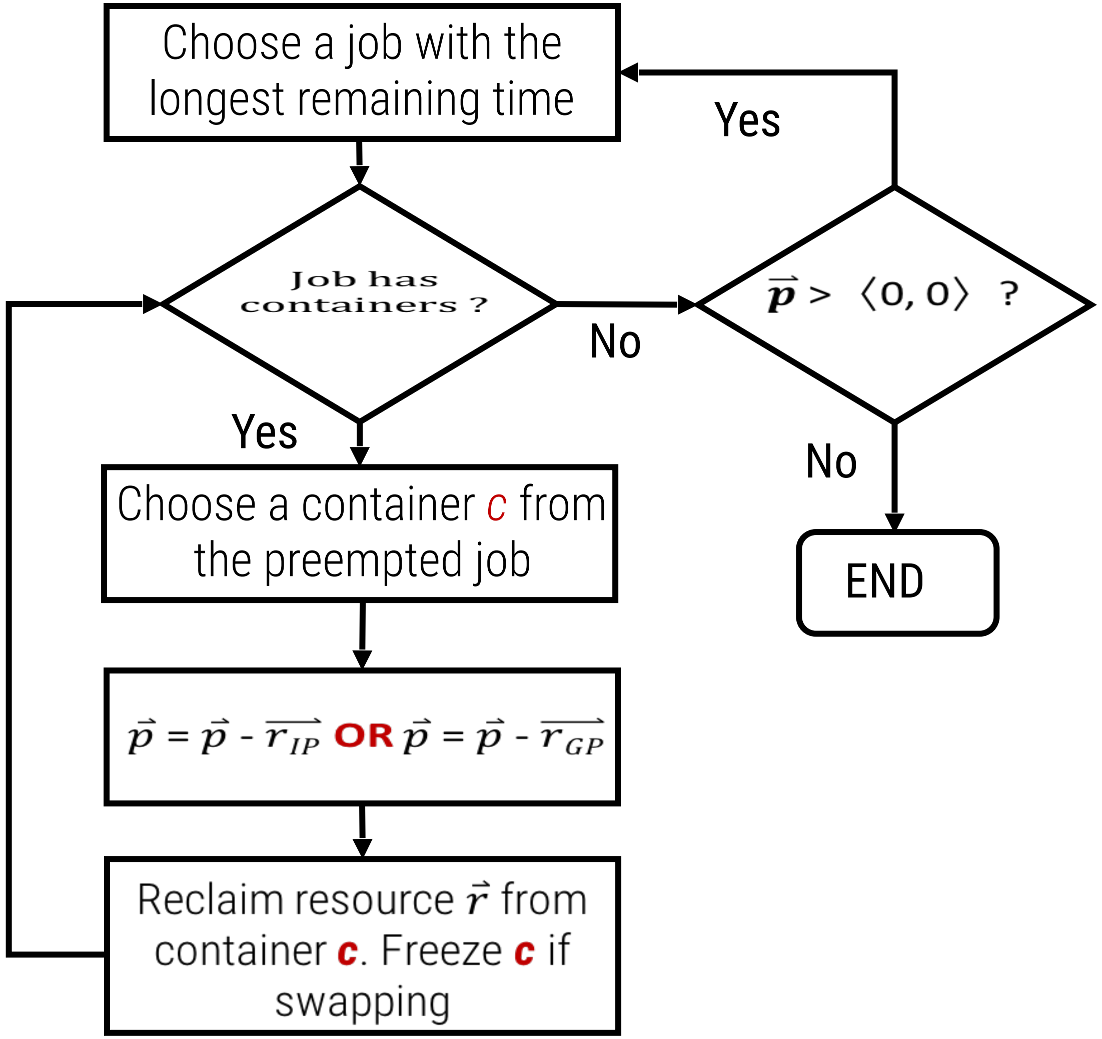
$\vec{r_s}$ is the total demand of many small tasks, which may not be able to fully use 10GB mem since CPU is not fully satisfied

Memory reclamation is in proportion to the reclaimed CPU according to $\vec{r_s}$

Container Preemption Algorithm



Container Preemption Algorithm



Immediate preemption (IP) suspends a container and reclaims its entire resource $\overrightarrow{r_{IP}}$

Graceful preemption (GP) shrinks a container and reclaims its resource at a step of $\overrightarrow{r_{GP}}$. GP reclaims resources from multiple tasks (containers) and jobs.

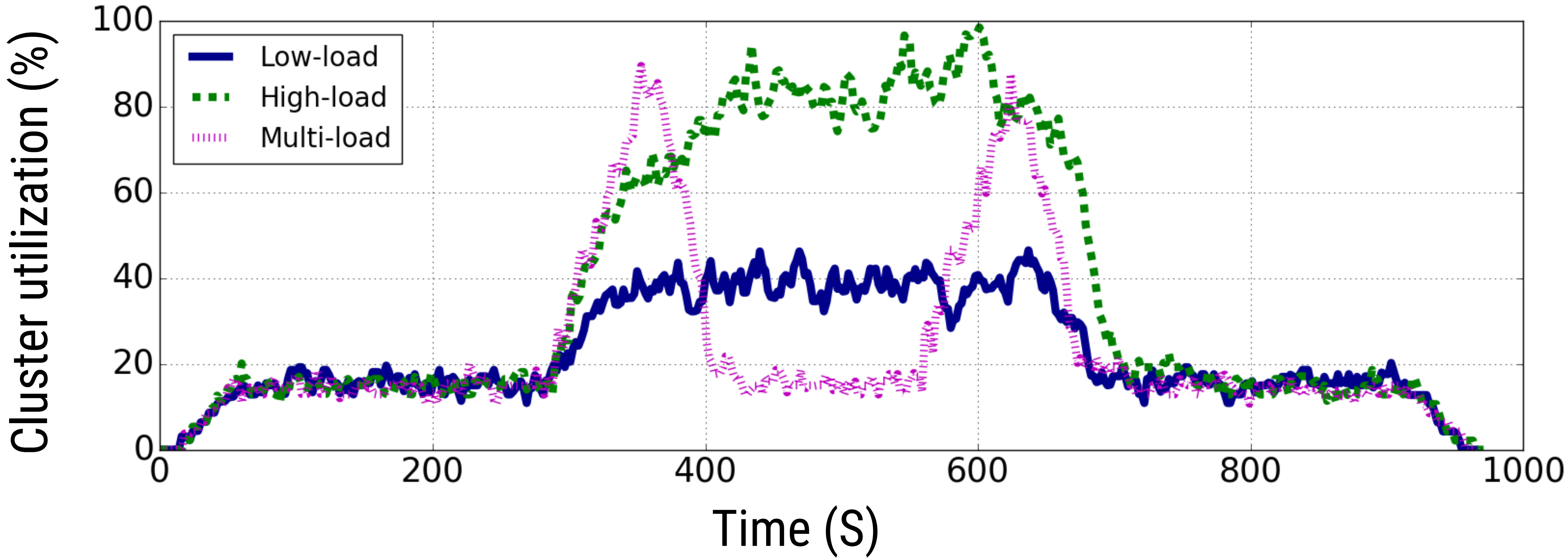
Optimizations

- Disable speculative execution of preempted tasks
  - Suspended tasks appear to be slow to the cluster manager and will likely trigger futile speculative execution
- Delayed task resubmission
  - Tasks may be resubmitted immediately after preemption, causing them to be suspended again. A suspended task is required to perform *D* attempts before it is re-admitted

Experimental Settings

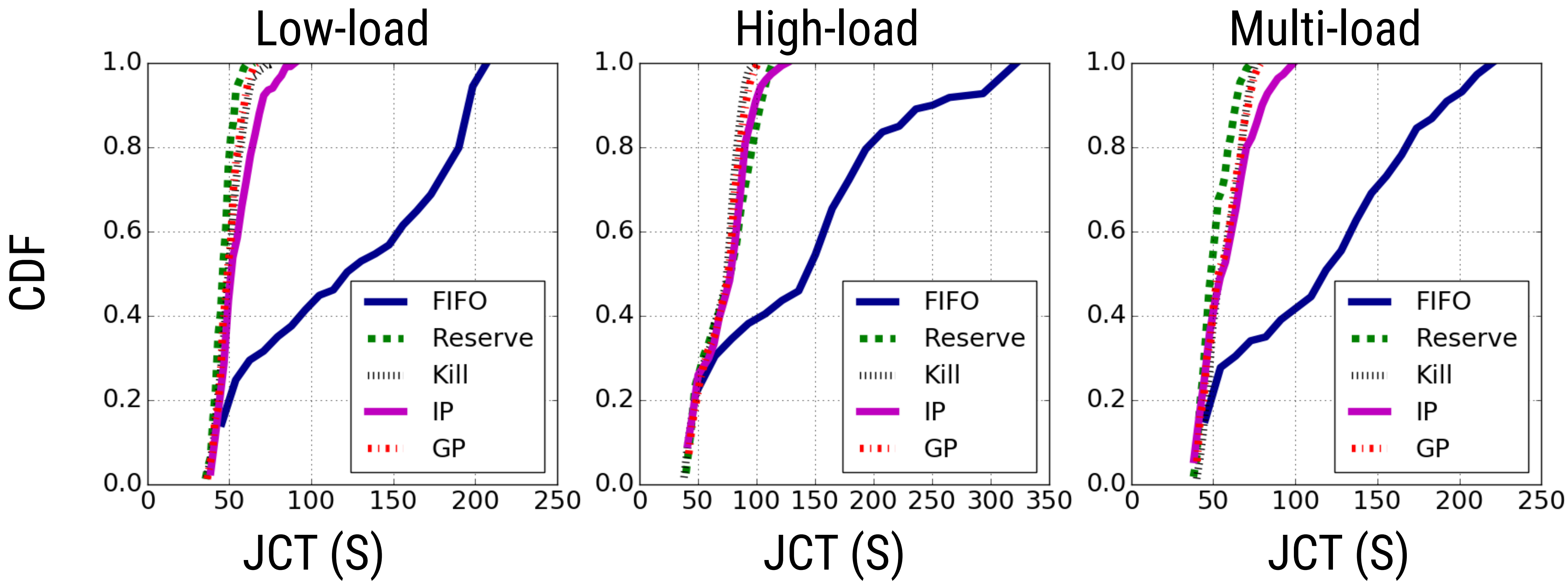
- Hardware
  - 26-node cluster; 32 cores, 128GB on each node; 10Gbps Ethernet, RAID-5 HDDs
- Software
  - Hadoop-2.7.1, Docker-1.12.1
- Cluster configuration
  - Two queues: 95% and 5% shares for short and long jobs queues, respectively
  - Schedulers: FIFO (no preemption), Reserve (60% capacity for short jobs), Kill, IP and GP
  - Workloads: Spark-SQL as short jobs and HiBench benchmarks as long jobs

Synthetic Workloads



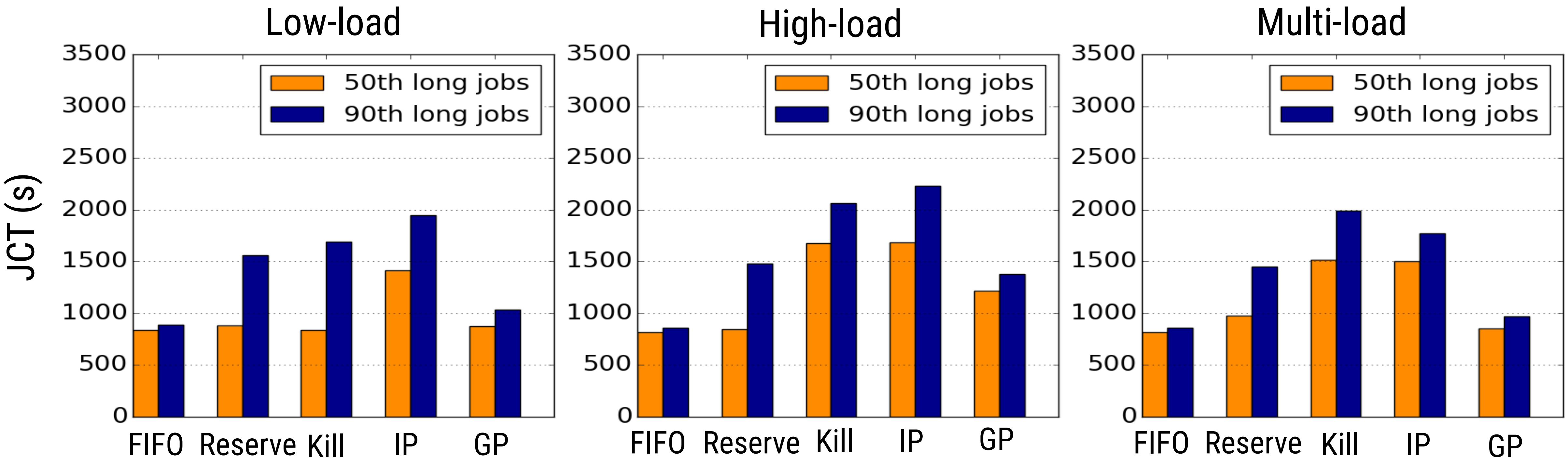
High, low, and multiple bursts of short jobs.
Long jobs persistently utilize 80% of cluster capacity

Short Job Latency with Spark



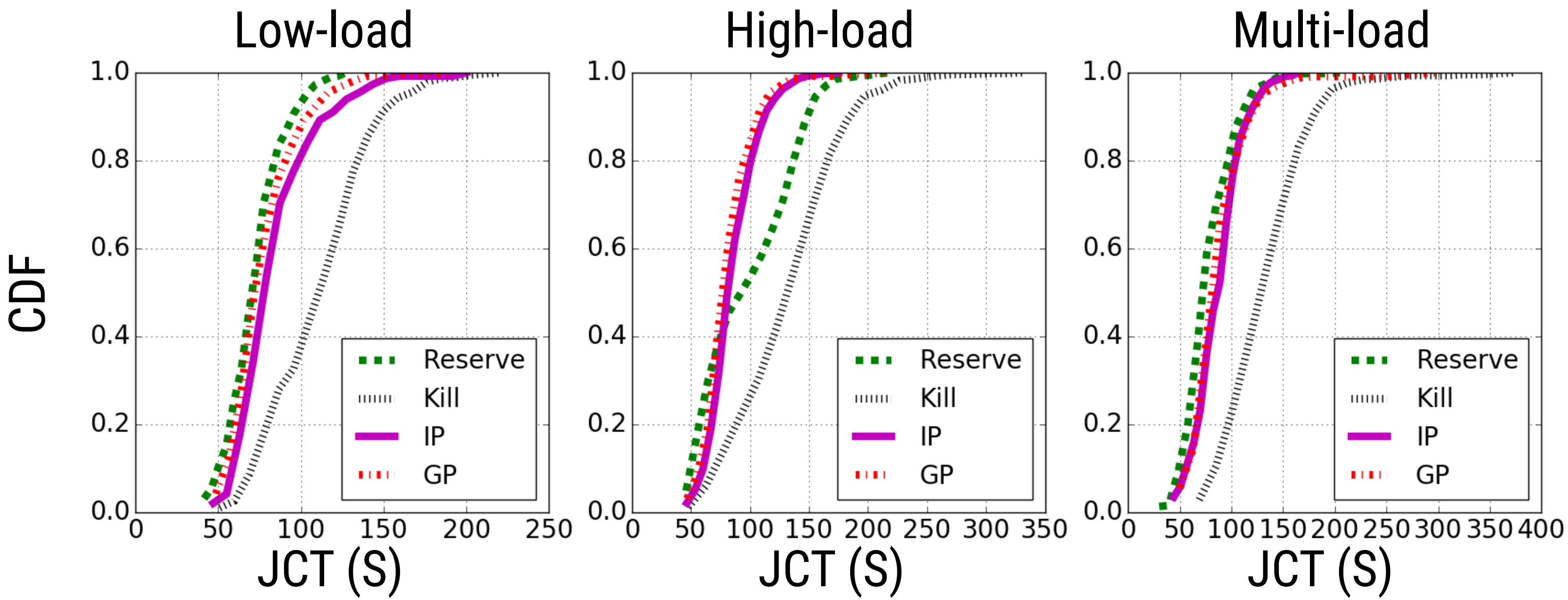
- FIFO is the worst due to the inability to preempt long jobs
- Reserve underperforms due to lack of reserved capacity under high-load
- GP is better than IP due to less resource reclamation time or swapping

Performance of Long Spark Jobs



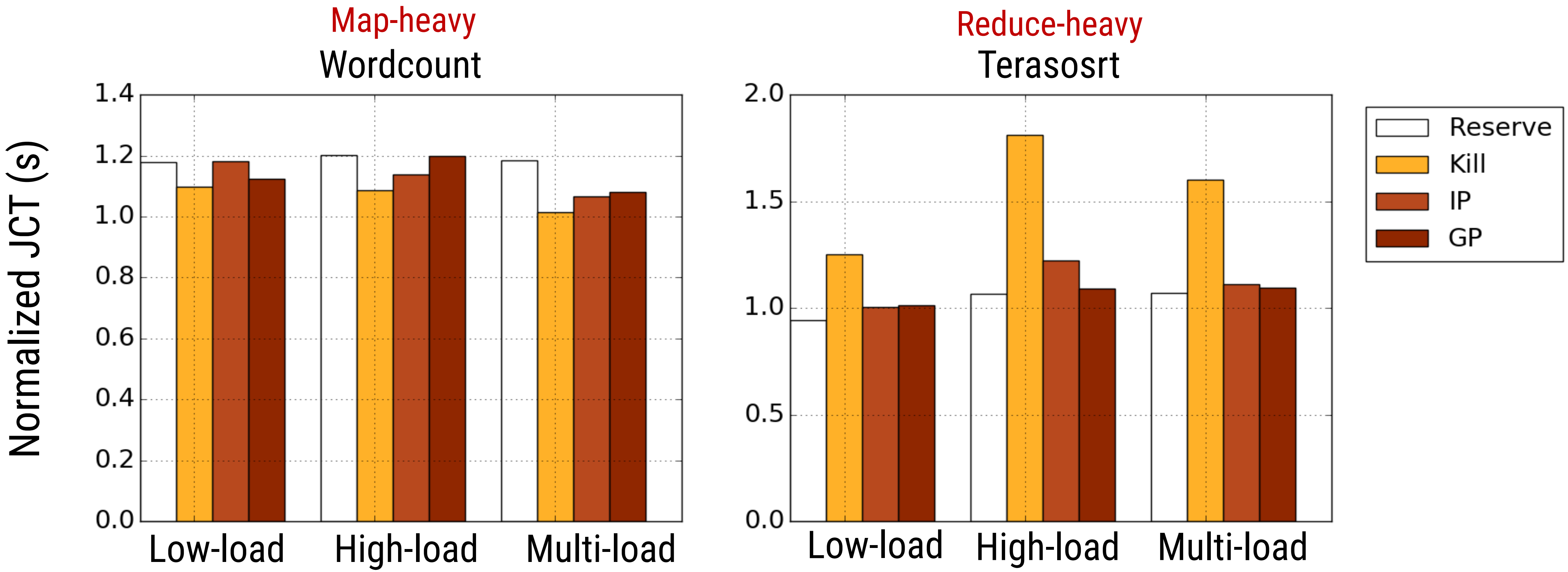
- FIFO is the reference performance for long jobs
- GP achieves on average 60% improvement over Kill.
- IP incurs significant overhead to Spark jobs:
  - aggressive resource reclamation causes system-wide swapping
  - completely suspended tasks impede overall job progress

Short Job Latency with MapReduce



- FIFO (not shown) incurs 15-20 mins slowdown to short jobs
- Re-submissions of killed MapReduce jobs block short jobs
- IP and GP achieve similar performance

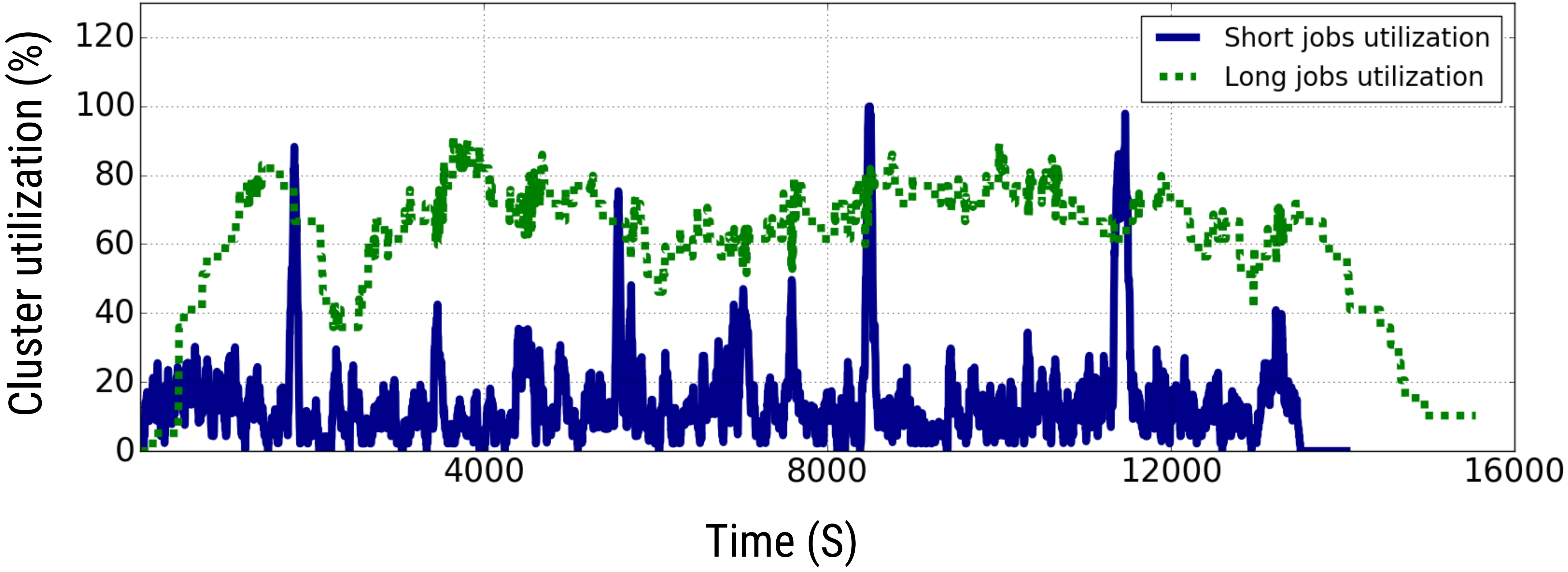
Performance of Long MapReduce Jobs



- Kill performs well for map-heavy workloads
- IP and GP show similar performance for MapReduce workloads
  - MapReduce tasks are loosely coupled
  - A suspended task does not stop the entire job

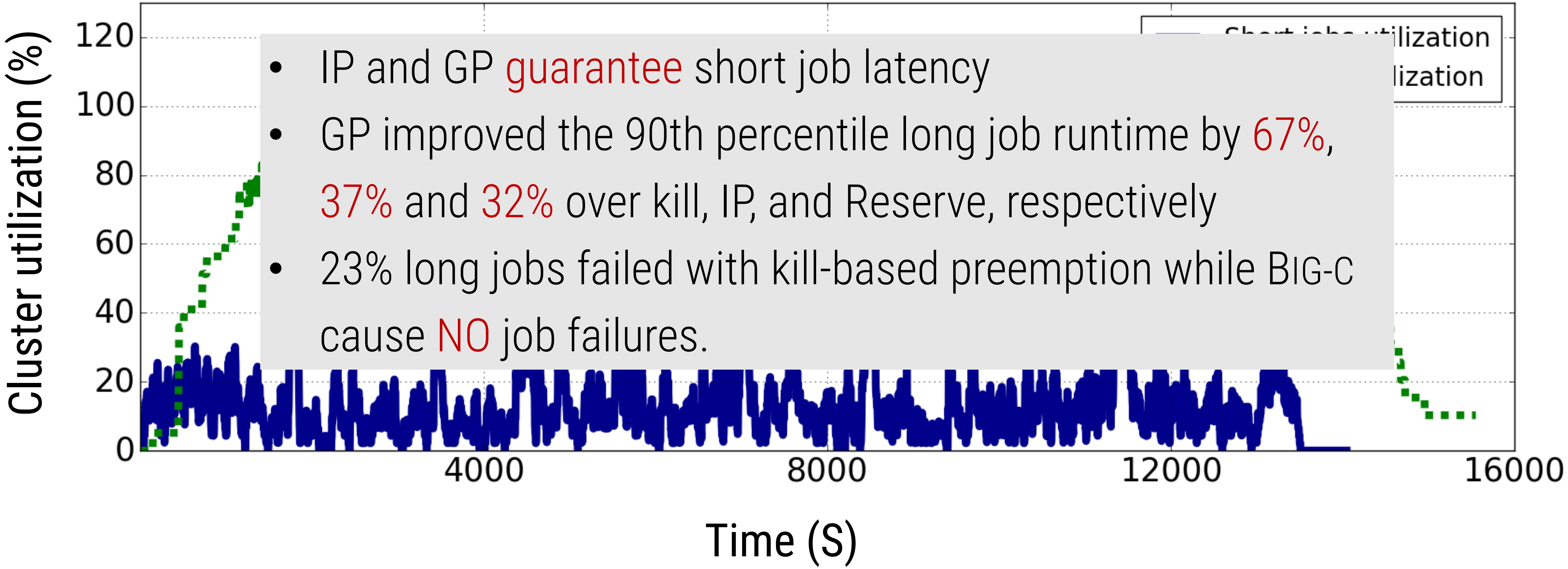
Google Trace

Contains 2202 jobs, of which 2020 are classified as short jobs and 182 as long jobs.



Google Trace

Contains 2202 jobs, of which 2020 are classified as short jobs and 182 as long jobs.



- IP and GP guarantee short job latency
- GP improved the 90th percentile long job runtime by 67%, 37% and 32% over kill, IP, and Reserve, respectively
- 23% long jobs failed with kill-based preemption while BIG-C cause NO job failures.

Summary

- Data-intensive cluster computing lacks an efficient mechanism for task preemption
  - Task killing incurs significant slowdowns or failures to preempted jobs
- **BIG-C** is a simple yet effective approach to enable preemptive cluster scheduling
  - lightweight virtualization helps to containerize tasks
  - Task preemption is achieved through precise resource management
- Results:
  - BIG-C maintains short job latency close to reservation-based scheduling while achieving similar long job performance compared to FIFO scheduling