CE 528 Cloud Computing

Lecture 9: Resource Sharing
Spring 2026

Prof. Yigong Hu

Slides courtesy of Chang Lou and Robert Morrios

Administrivia

Spring break!

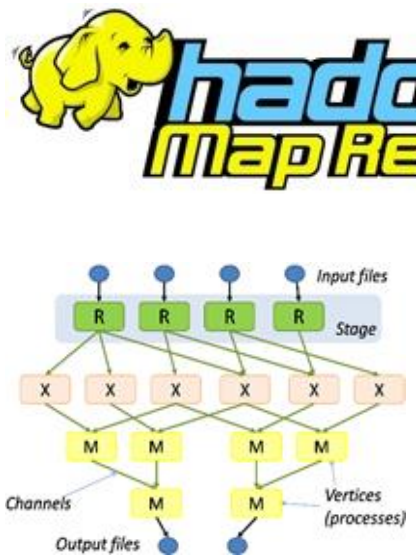
- No class
- Assignments
  - food, lots of it
  - sleep, lots of it

The answers to three questions are missing.

- Why/How does DataFusion Comet accelerate Spark SQL?
- What does “provenance” mean in the context of ML systems?
- What is the first step in the pipeline?

Background

Rapid innovation in cluster computing frameworks



Dryad

CIEL

Google Pregel

Pig

RAILS

S4 distributed stream computing platform

Google Percolator

MPI-2

Problem

Rapid innovation in cluster computing frameworks

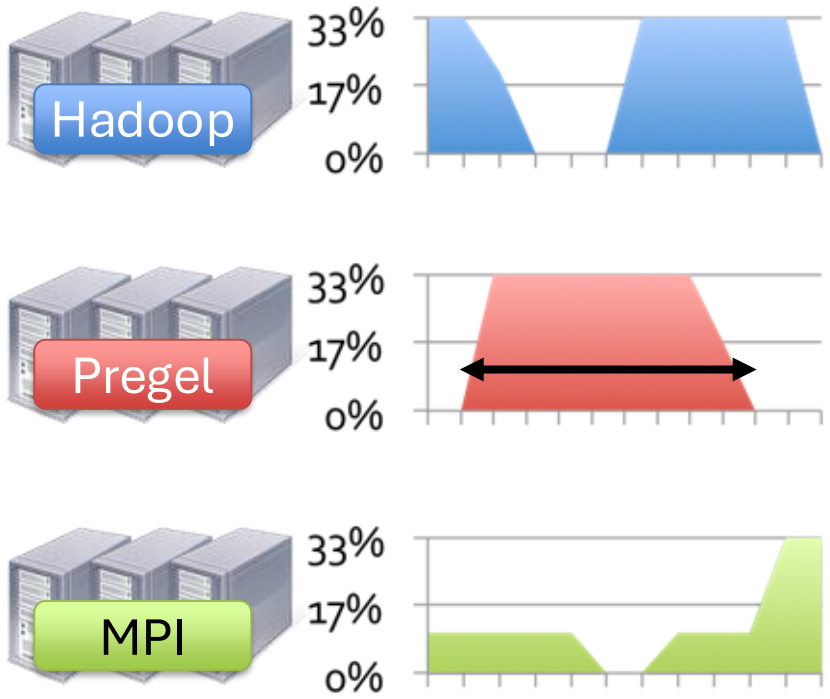
No single framework optimal for all applications

Want to run multiple frameworks in a single cluster

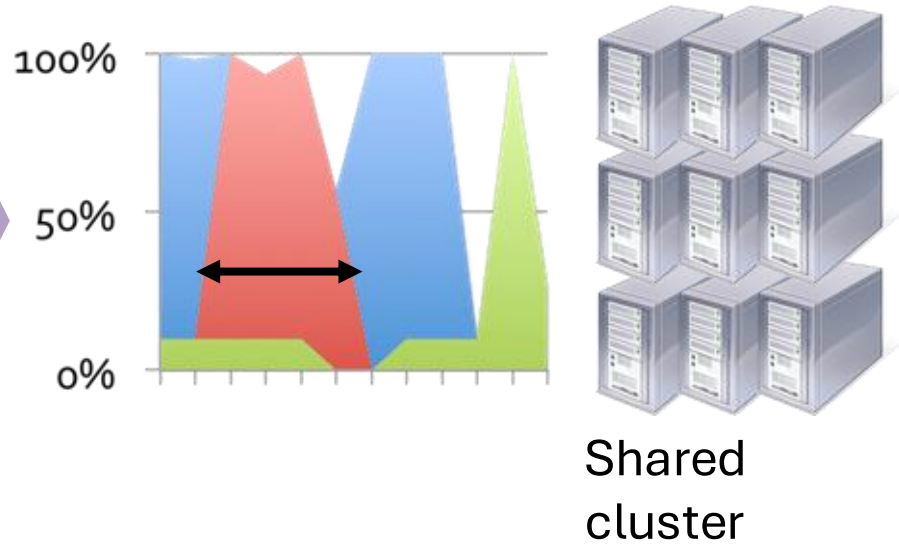
- …to maximize utilization
- …to share data between frameworks

Where We Want to Go

Today: static partitioning



Mesos: dynamic sharing



Shared cluster

Mesos

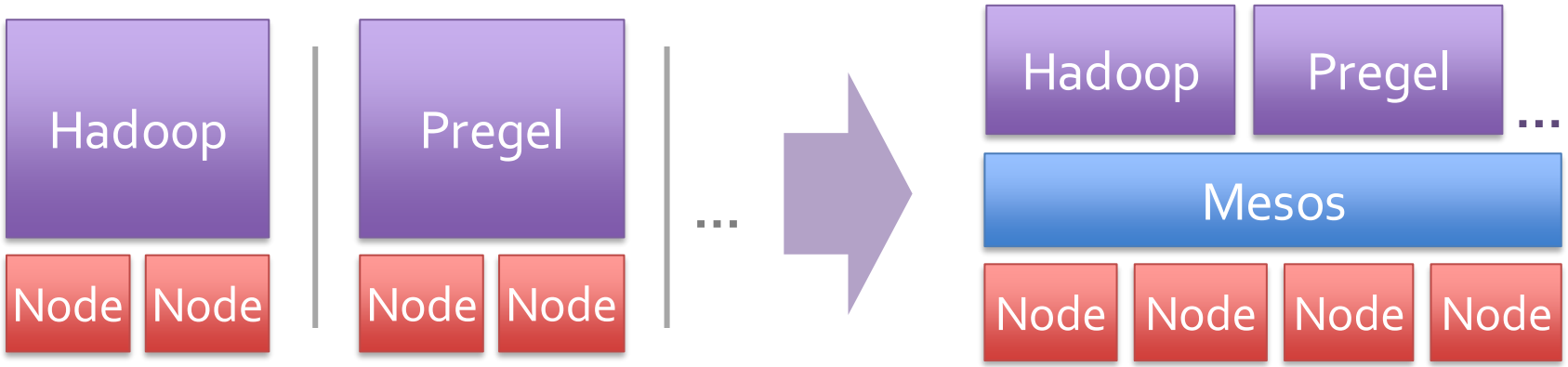
A Platform for Fine-Grained Resource Sharing in the Data Center

Benjamin Hindman, Andy Konwinski, **Matei Zaharia**,
Ali Ghodsi, Anthony Joseph, Randy Katz, Scott Shenker, Ion Stoica

University of California, Berkeley

Solution

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



Other Benefits of Mesos

Run multiple instances of the same framework

- Isolate production and experimental jobs
- Run multiple versions of the framework concurrently

Build specialized frameworks targeting particular problem domains

- Better performance than general-purpose abstractions

Outline

Mesos Goals and Architecture

Implementation

Results

Related Work

What are the key goals Mesos?

Mesos Goals

High utilization of resources

Support diverse frameworks (current & future)

Scalability to 10,000's of nodes

Reliability in face of failures

Resulting design: Small microkernel-like core that pushes scheduling logic to frameworks

What Are the Two Key Design Elements?

Design Elements

Fine-grained sharing:

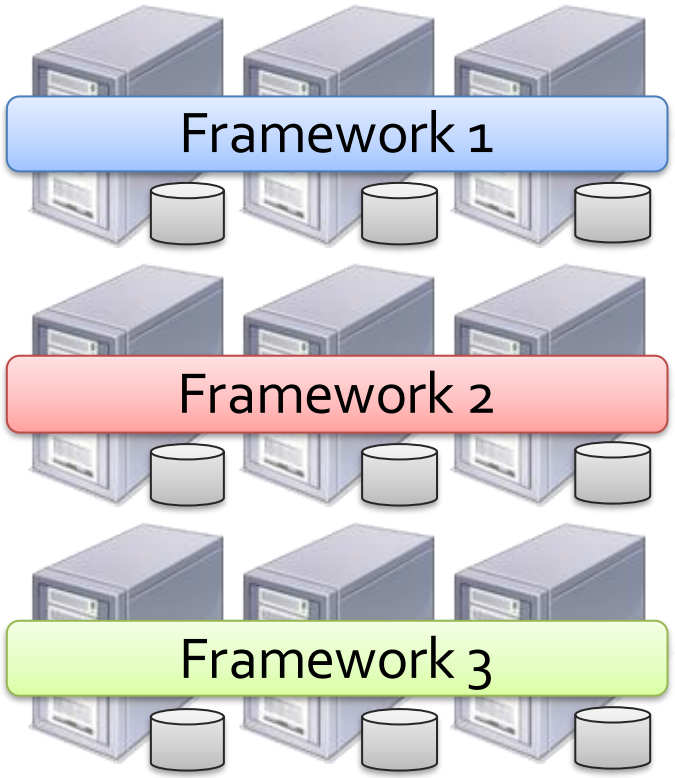
- Allocation at the level of tasks within a job
- Improves utilization, latency, and data locality

Resource offers:

- Simple, scalable application-controlled scheduling mechanism

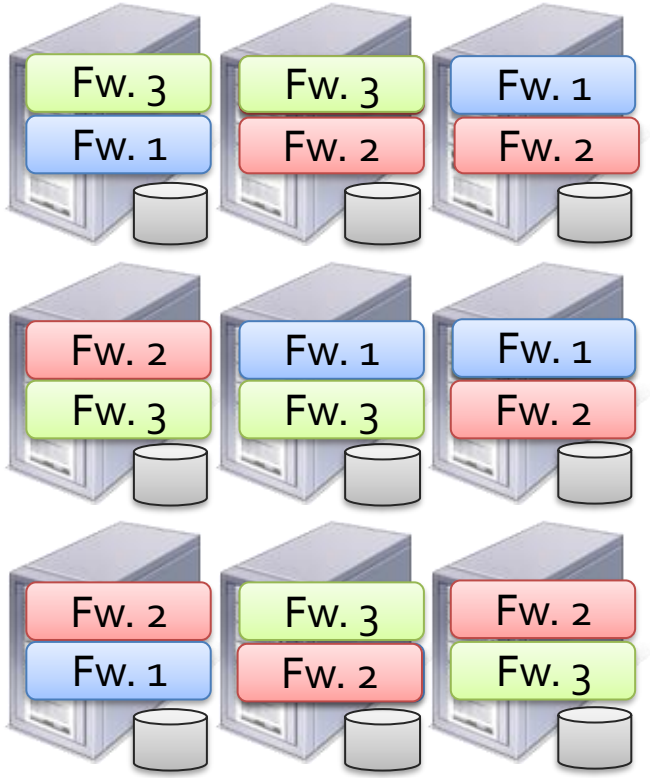
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):



Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):



Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality

Element 2: Resource Offers

Option: Global scheduler

- Frameworks express needs in a specification language, global scheduler matches them to resources

+ Can make optimal decisions

– Complex: language must support all framework needs

– Difficult to scale and to make robust

– Future frameworks may have unanticipated needs

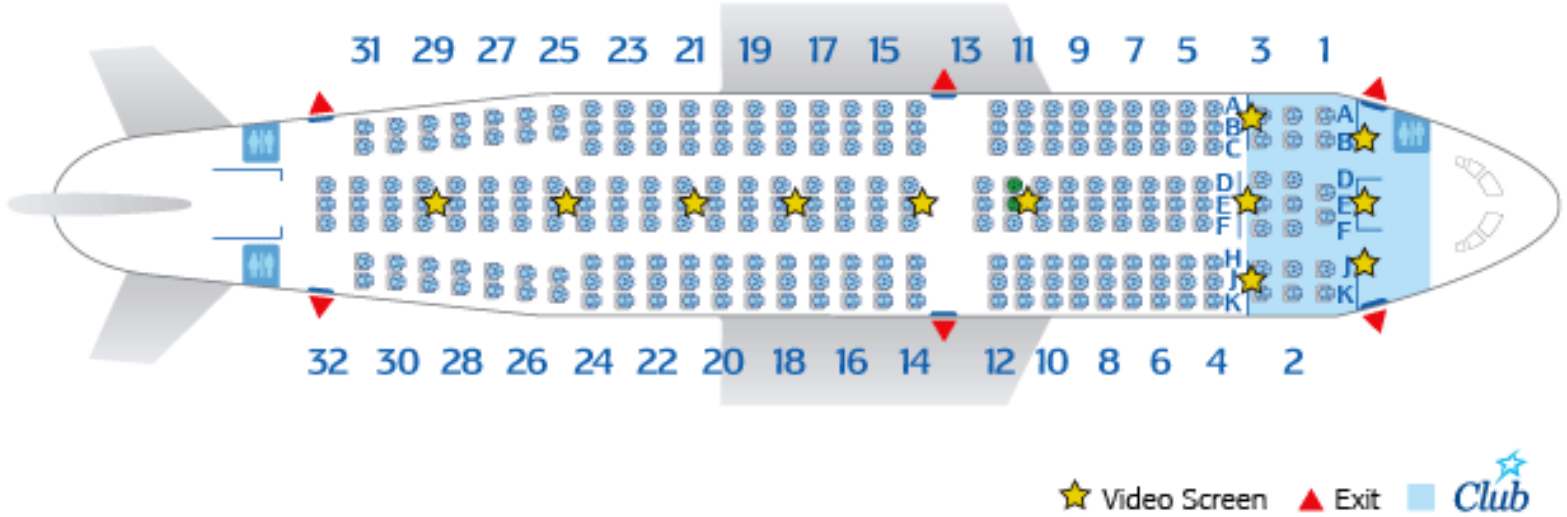
Element 2: Resource Offers

Mesos: Resource offers

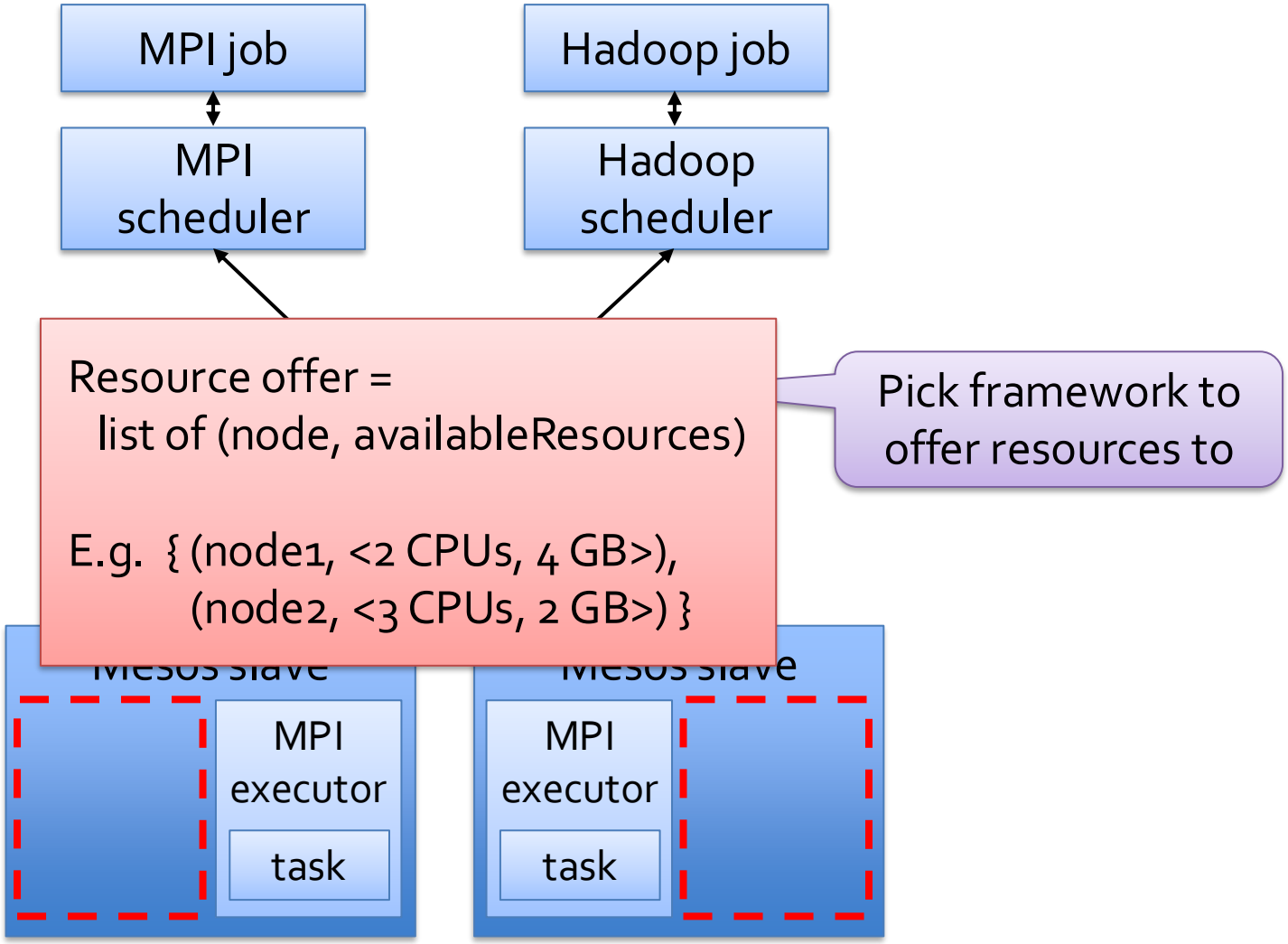
- Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

\+ Keeps Mesos simple, lets it support future frameworks

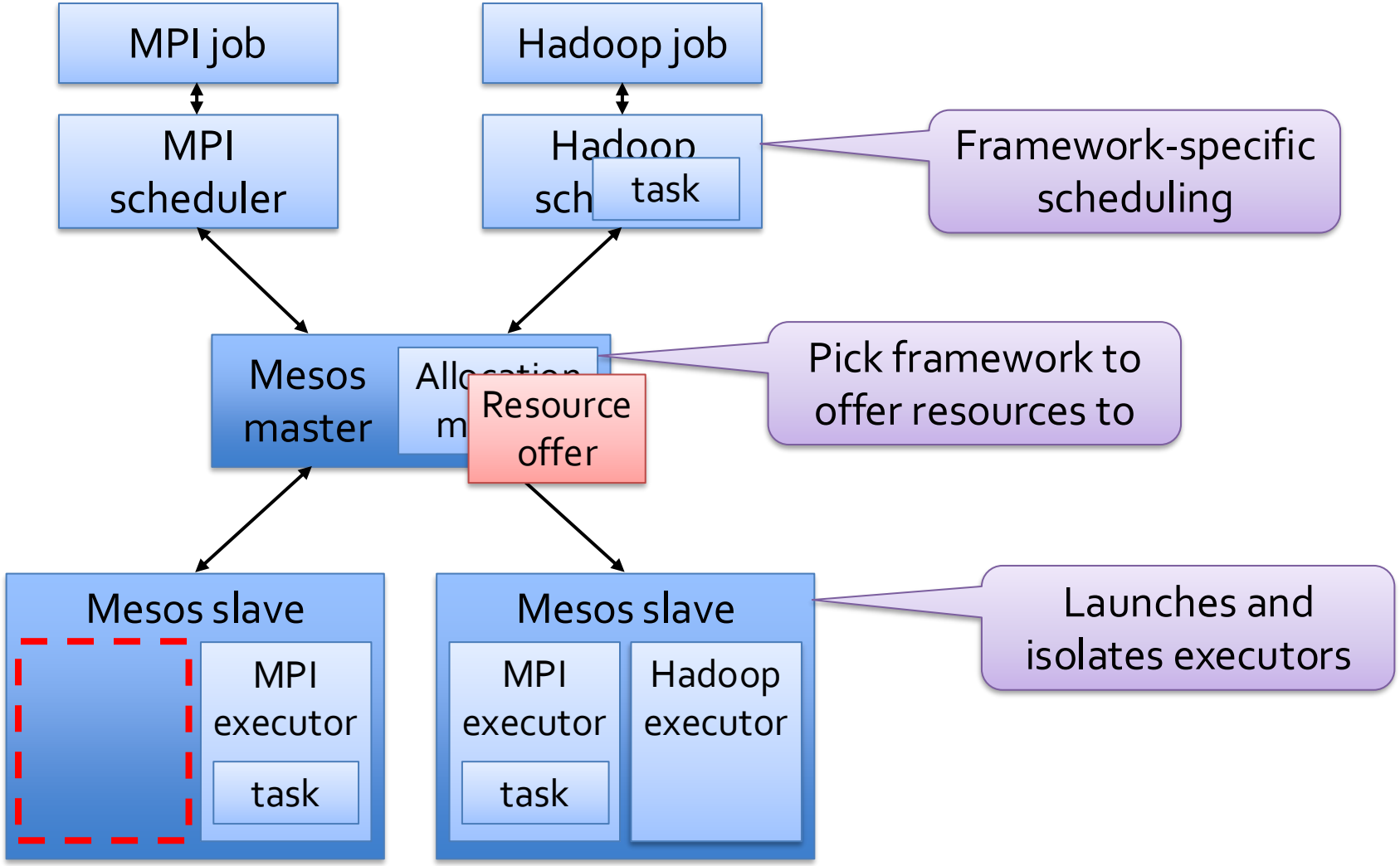
− Decentralized decisions might not be optimal



Mesos Architecture



Mesos Architecture



Optimization: Filters

Let frameworks short-circuit rejection by providing a predicate on resources to be offered

- E.g. “nodes from list L” or “nodes with > 8 GB RAM”
- Could generalize to other hints as well

Ability to reject still ensures correctness when needs cannot be expressed using filters

Users

Twitter uses Mesos on > 100 nodes to run ~12 production services (mostly stream processing)

Berkeley machine learning researchers are running several algorithms at scale on Spark

Conviva is using Spark for data analytics

UCSF medical researchers are using Mesos to run Hadoop and eventually non-Hadoop apps

Framework Isolation

Mesos uses OS isolation mechanisms, such as Linux containers and Solaris projects

Containers currently support CPU, memory, IO and network bandwidth isolation

Not perfect, but much better than no isolation

Analysis

Resource offers work well when:

- Frameworks can scale up and down elastically
- Task durations are homogeneous
- Frameworks have many preferred nodes

These conditions hold in current data analytics frameworks (MapReduce, Dryad, …)

- Work divided into short tasks to facilitate load balancing and fault recovery
- Data replicated across multiple nodes

Large-scale cluster management at Google with Borg

By: Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, John Wilkes

Google Inc

Example

```
job hello_world = {
   runtime = { cell = 'ic' }                  // What cluster should we run in?
   binary = '.../hello_world_webserver'       // What program are we to run?
   args = { port = '%port%' }                 // Command line parameters
   requirements = {      // Resource requirements
     ram = 100M
     disk = 100M
     cpu = 0.1
   }
   replicas =  10000    // Number of tasks
}
```

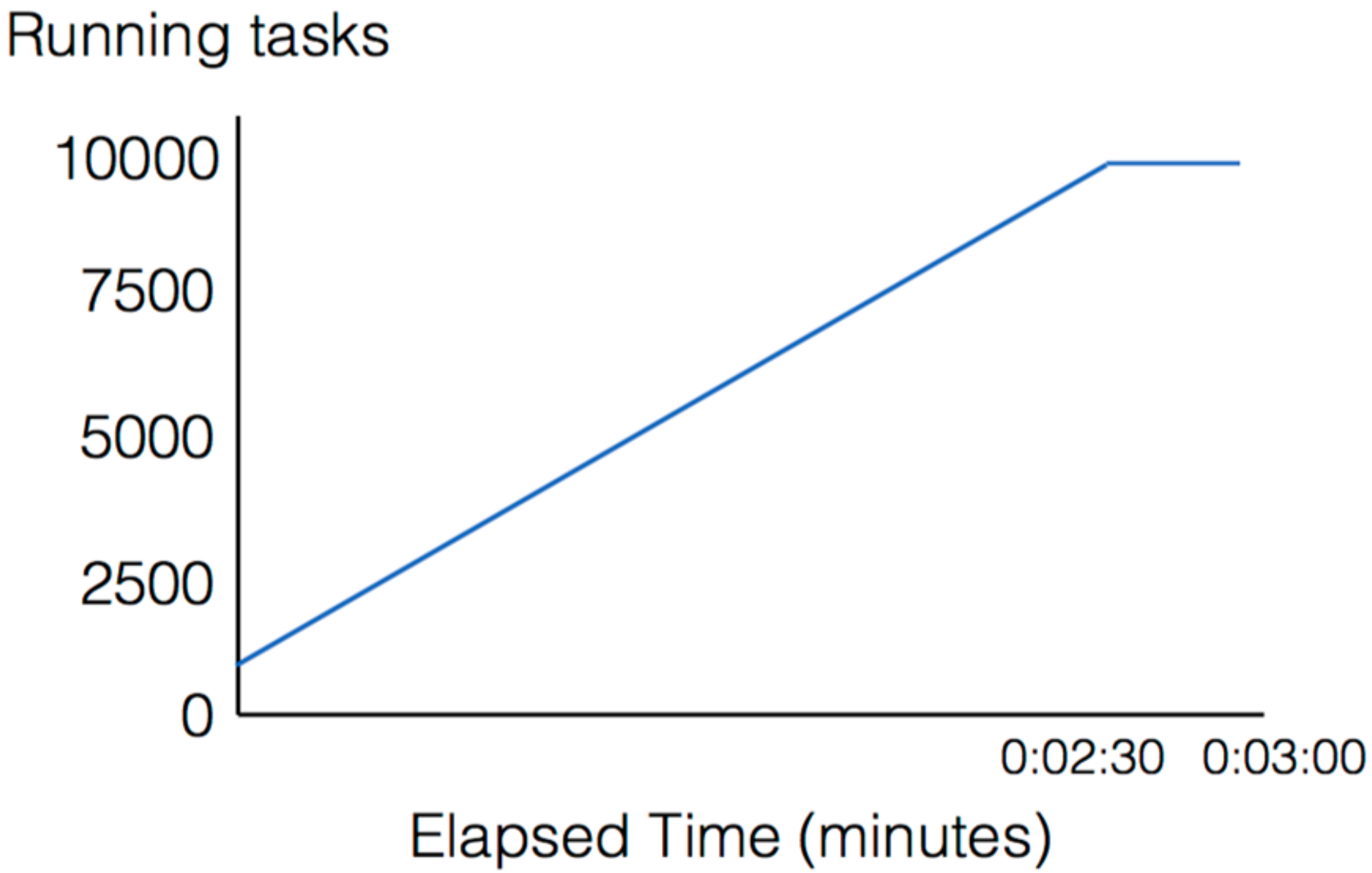
Abstractions

Cell – group of tightly coupled nodes

Job – name, owner, collection of identical tasks

Task – set of linux processes running in a container

Allocs and Alloc sets - allocation of resources on one machine, or for one job


Running tasks
10000
7500
5000
2500
0
0:02:30
0:03:00
Elapsed Time (minutes)

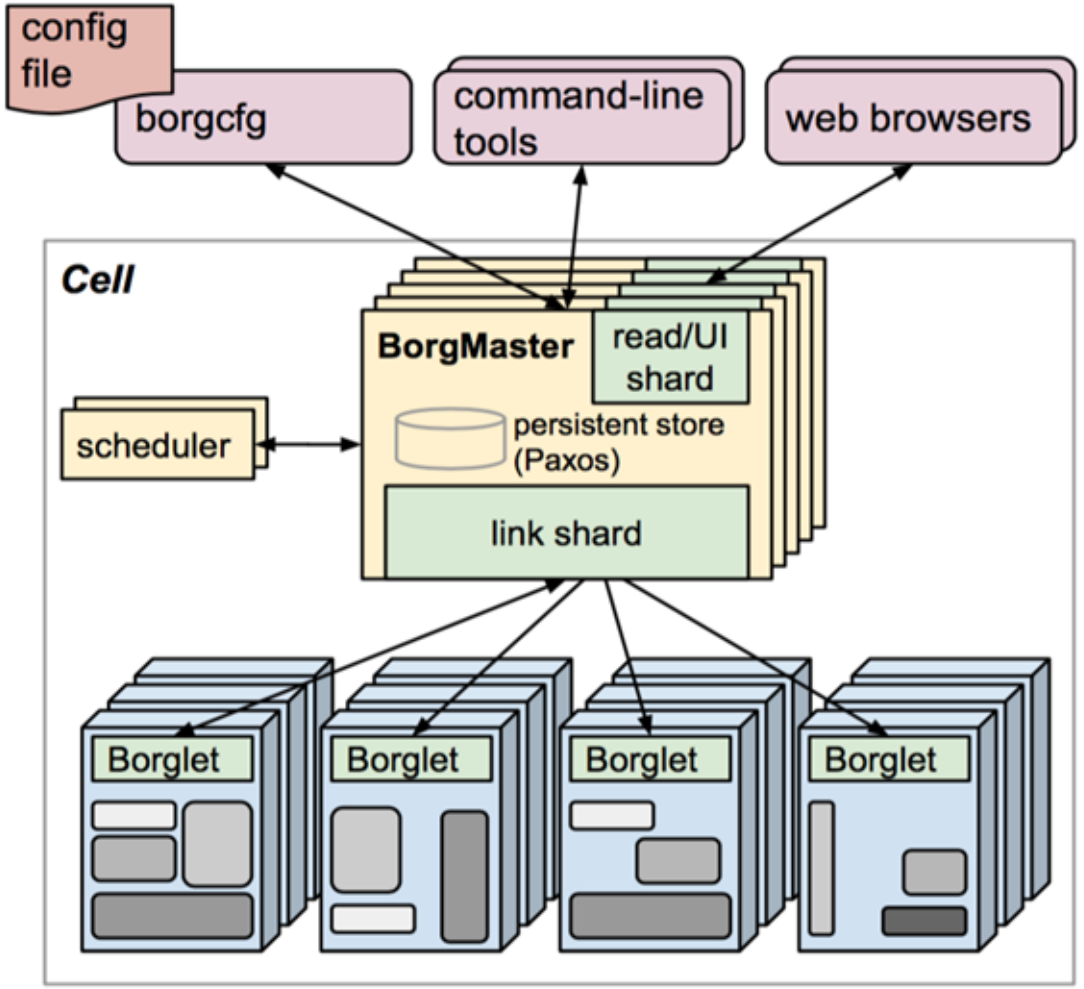
Benefits

Scalability to thousands of machines, efficiently shares the machines

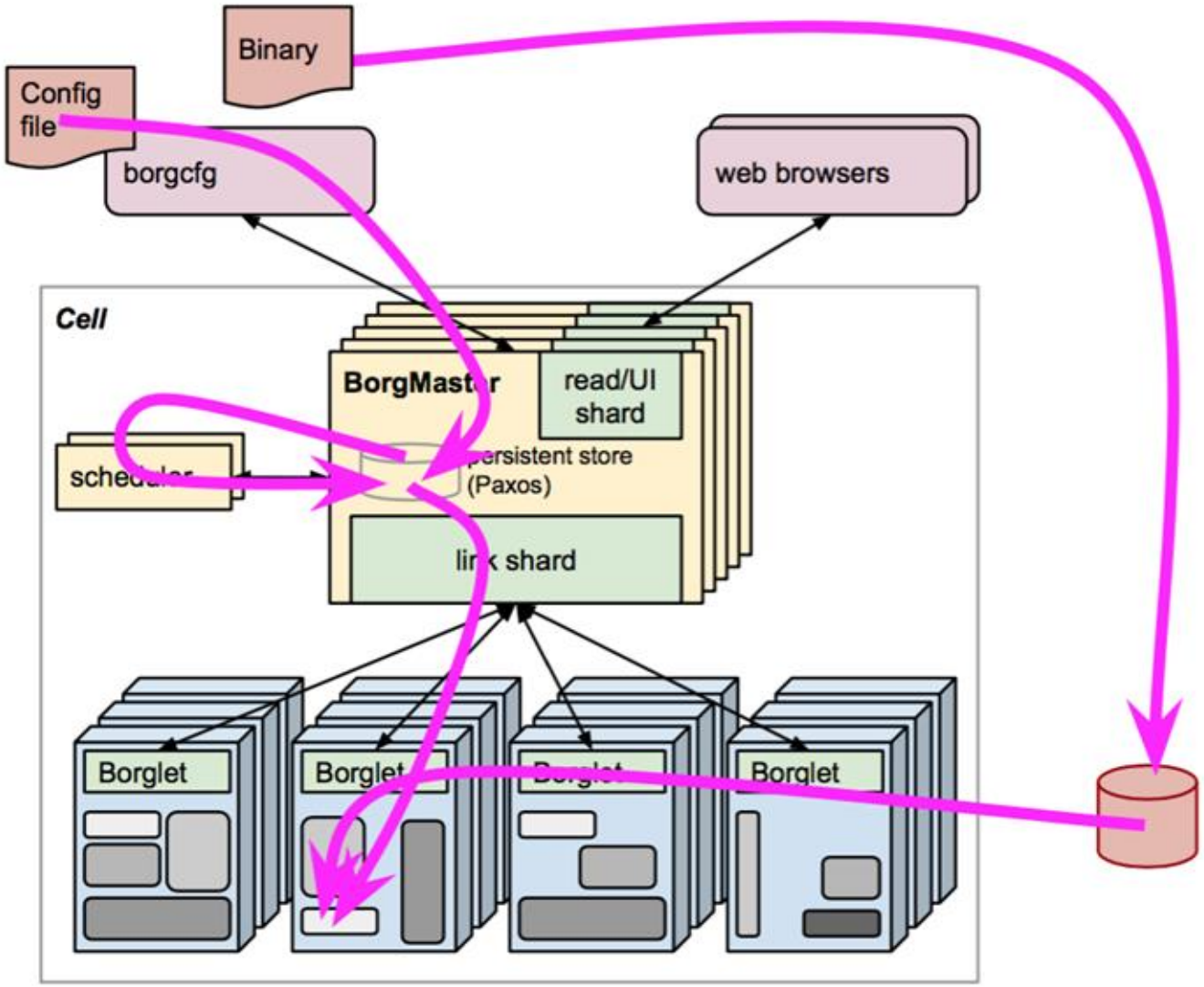
Abstracts away details of resource management, monitoring, fault handing from users

Operates with high reliability and availability

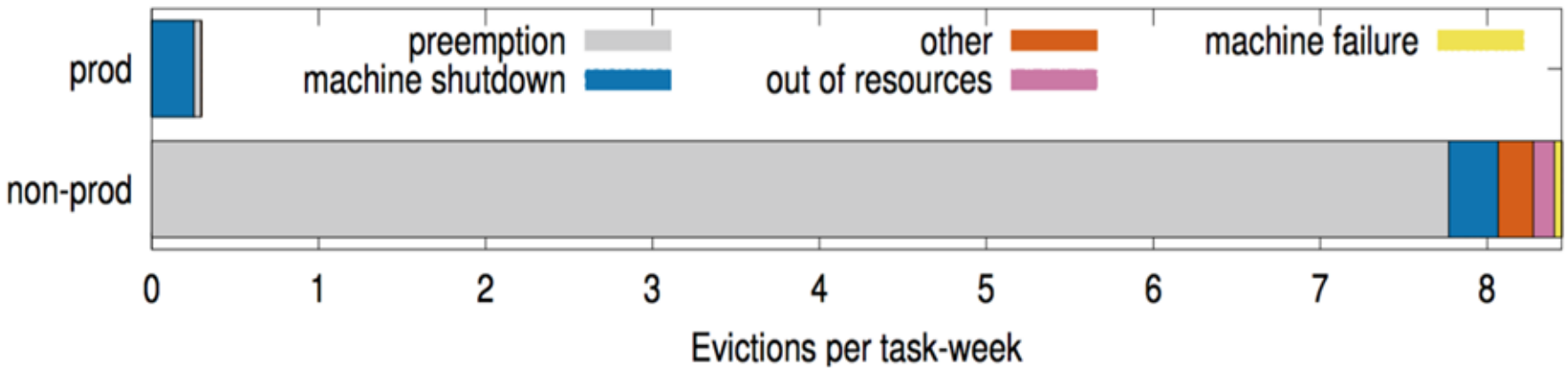
Borg Architecture

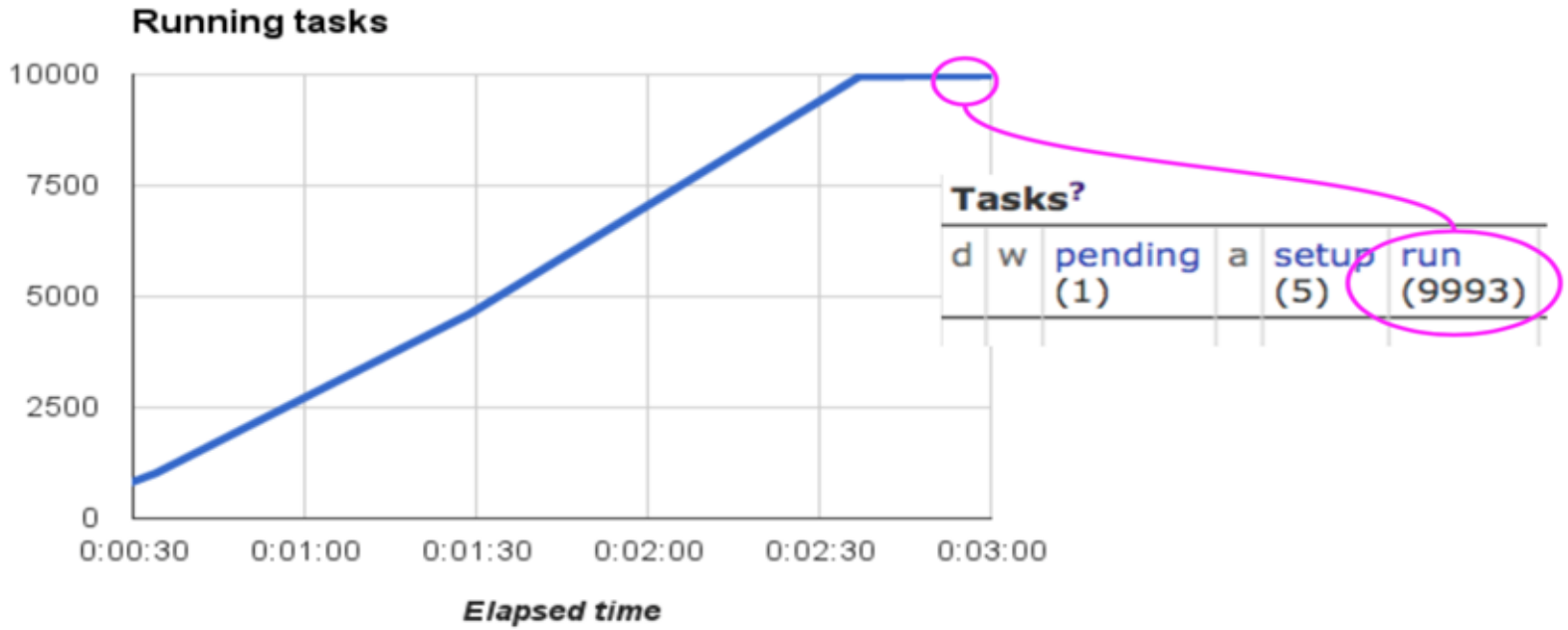


Control Flow



Availability





Characteristic of Workload

Long running services

- Latency sensitive, e.g., gmail, google docs, search, BigTable
- Many frameworks (MapReduce, Pregal…)
- Diurnal usage pattern
- Usually controller with master job, one or more worker job

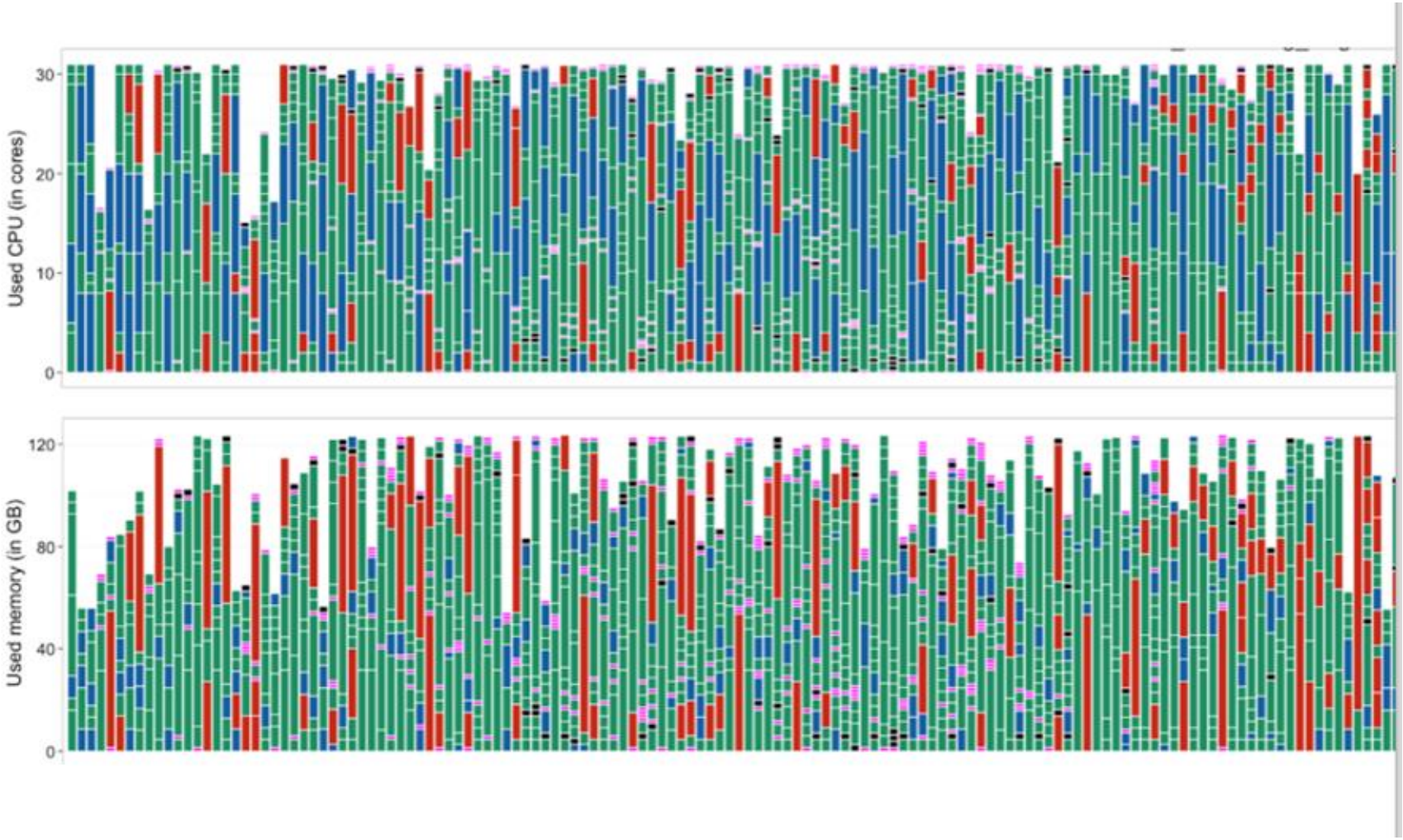
Batch jobs

- Take from seconds to days
- Insensitive to performance fluxuations

Priority – relative priority of jobs running or waiting to be run

- Higher will preempt/kill lower
- Production do not preempt each other
- Bands: 1) monitoring, 2) production, 3) batch, 4) best effort

Efficiency: Multiple Resources



Overcommit

Quota – used to decide which jobs to admit for scheduling; max resources job can ask for

- Users tend to buy more than they need
- System sells more lower-priority than it has
- Systems tends to be oversubscibed

Characteristic of App Execution Environment

Everything run in container

Binaries are statically compiled: dependencies

Tasks have built in HTTP server

- Health, and thousands of performance metrics
- Borg can monitor, and restart tasks don’t respond

Tasks assumed to handle failures

Borg/Mesos Comparison

Common:

- Infrastructure for scale, packaging…
- Containers
- Master that distributes work

Differences:

- Borg/Kubernetis
  - more prescriptive monitoring…
  - Scheduler that looks at all constraints, more efficient
  - Can schedule arbitrary jobs
- Mesos
  - exokernel like model, exploits/works with frameworks; not arbitrary jobs
  - Doesn’t talk about jobs that require more resources than are available
  - Much much simpler
  - Arguably more general
  - Probably more scalable