Cloud Computing
Resource management

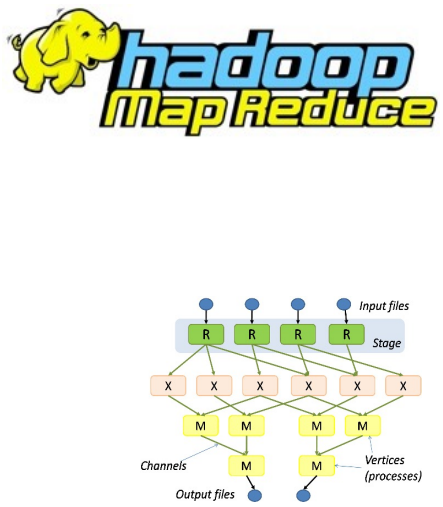
Minchen Yu
SDS@CUHK-SZ
Fall 2024

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

Background

Rapid innovation in cluster computing frameworks



Dryad

Ciel

S4 distributed stream computing platform

Google Percolator

Google Pregel

MPI-2

Background

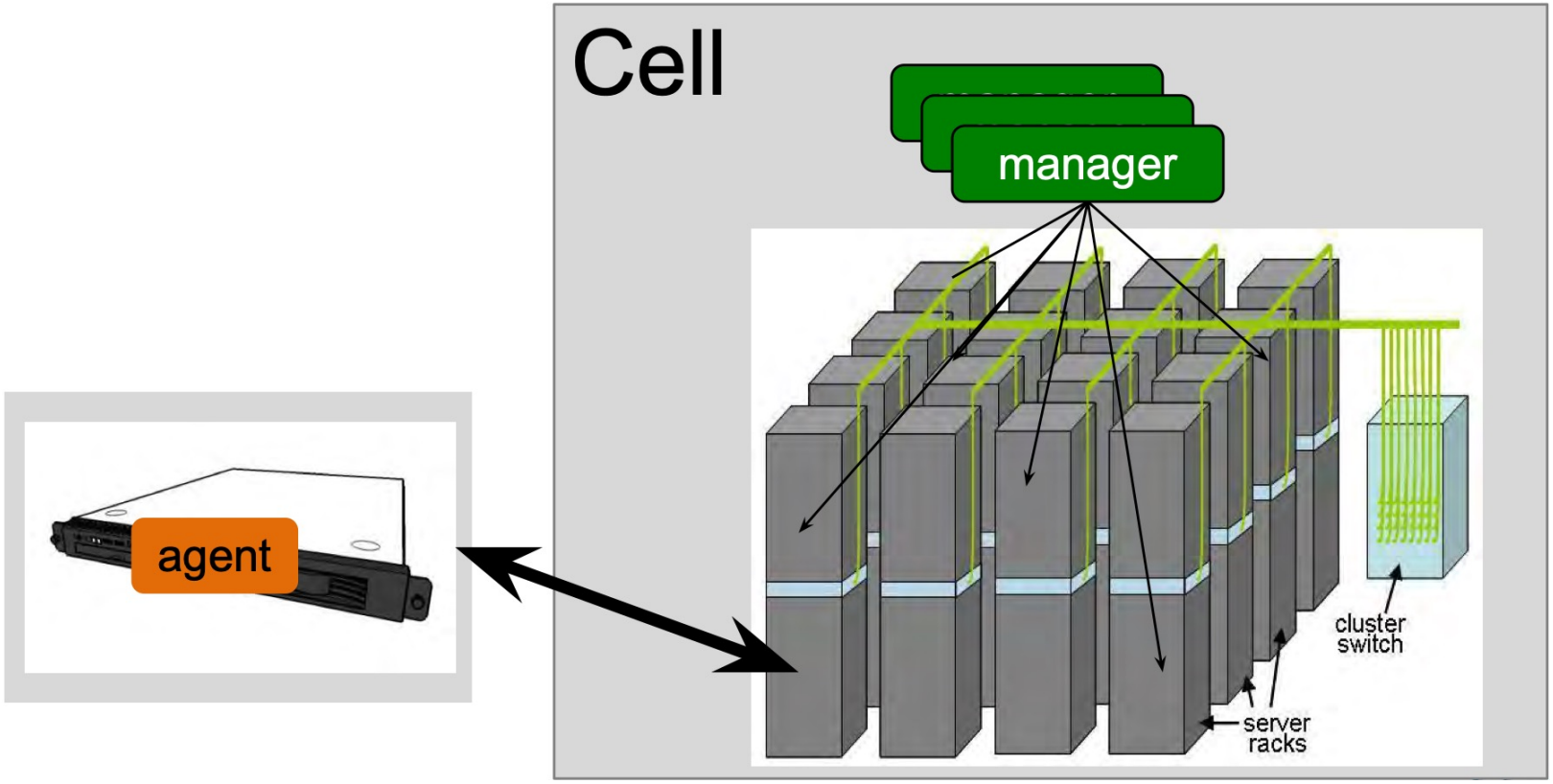
- Data centers are built from clusters of commodity hardware
- These clusters run a diverse set of applications: e.g., MapReduce, Spark, etc.
- Each application has its own execution framework
- Multiplexing a cluster between frameworks improves resources utilization and so reduces costs

Background

- It is very difficult and challenging for a single framework (application-specific) to efficiently manage the resources of clusters
- The aim of the cluster management is to be able to run multiple frameworks in a single cluster to:
  - maximize utilization
  - share data between frameworks

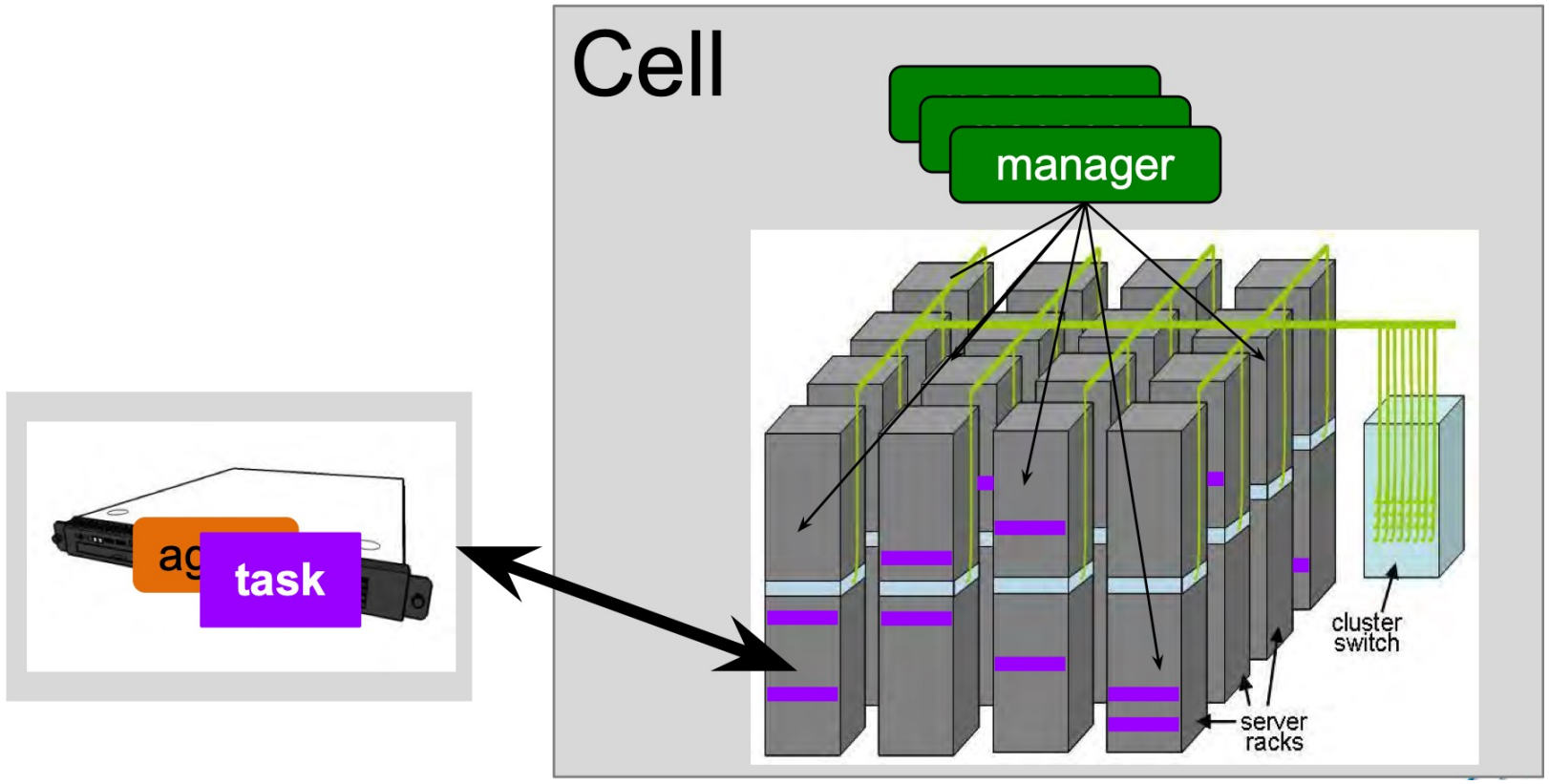
Cluster management: what is it?

A cluster is managed as 1 or more *cells*



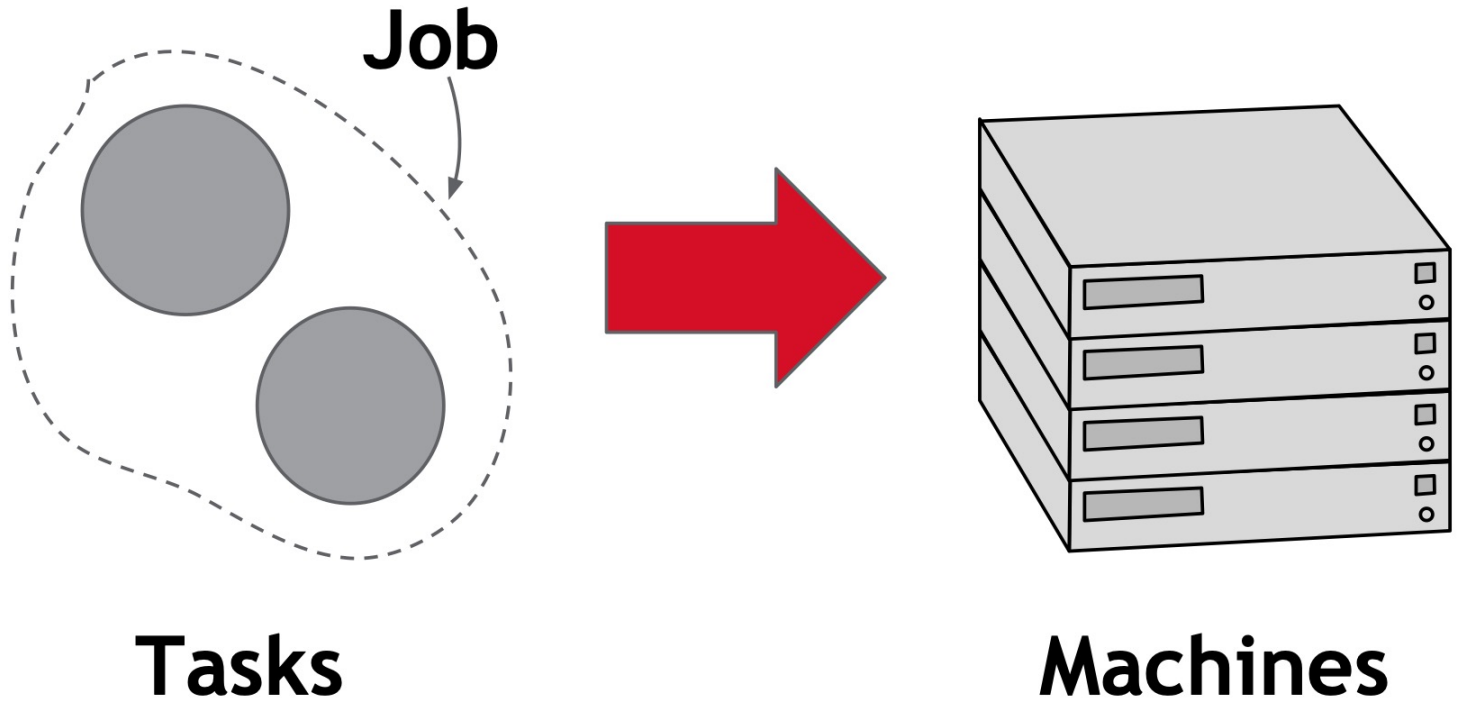
Cluster management: what is it?

A cell runs *jobs*, made up of *tasks*, for internal users



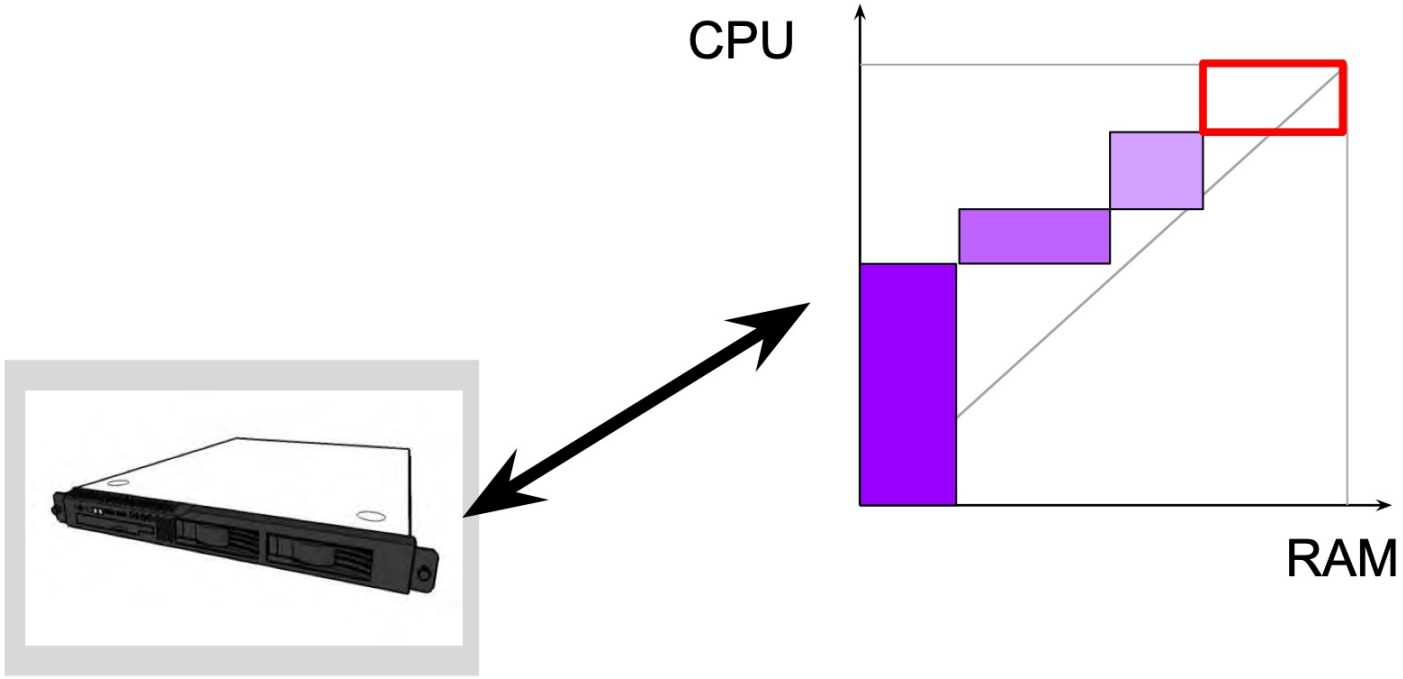
Cluster management: what is it?

The scheduling problem

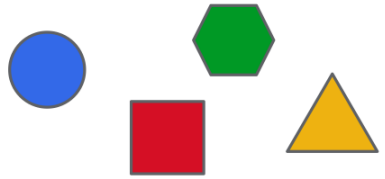


Cluster management: what is it?

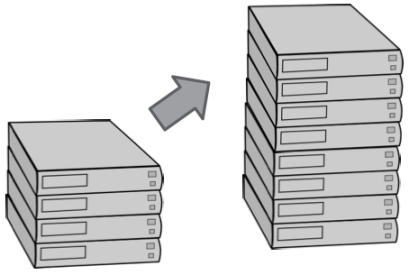
Placing tasks onto machines is just a ***knapsack problem***, with constraints



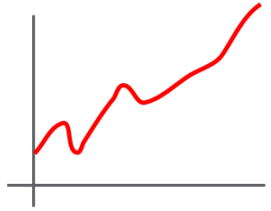
Cluster management is hard!



Diverse workloads



Increasing cluster sizes



Growing job arrival rates

Diverse workloads

Batch

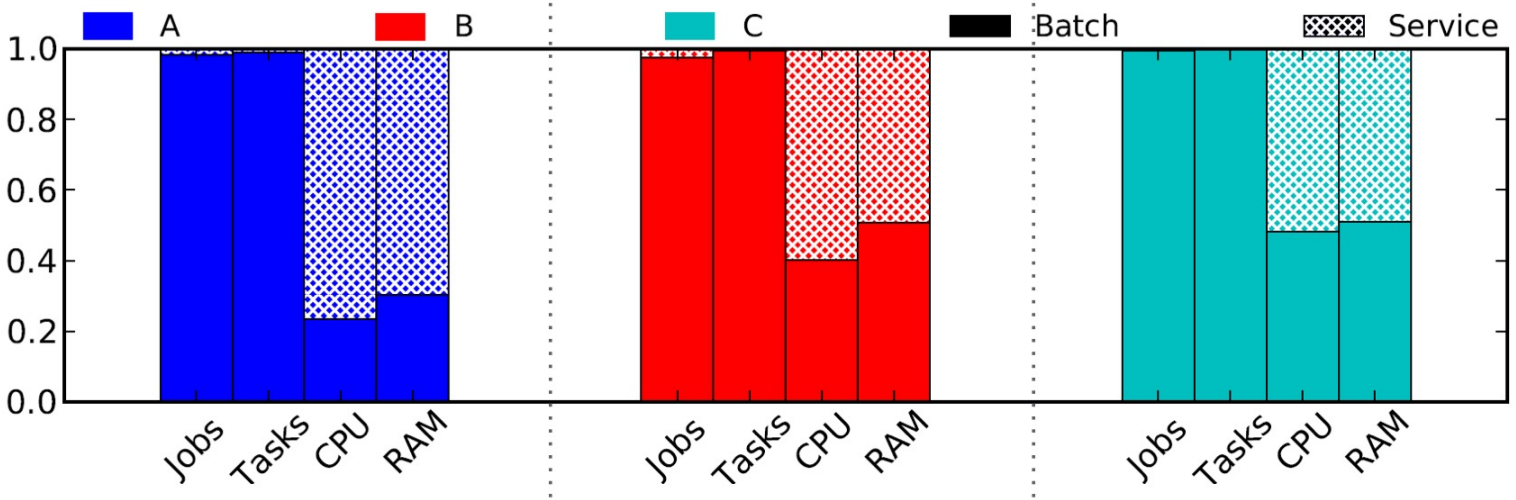
Service

Batch/service split

Cluster A
Medium size
Medium utilization

Cluster B
Large size
Medium utilization

Cluster C
Medium (12k mach.)
High utilization
Public trace



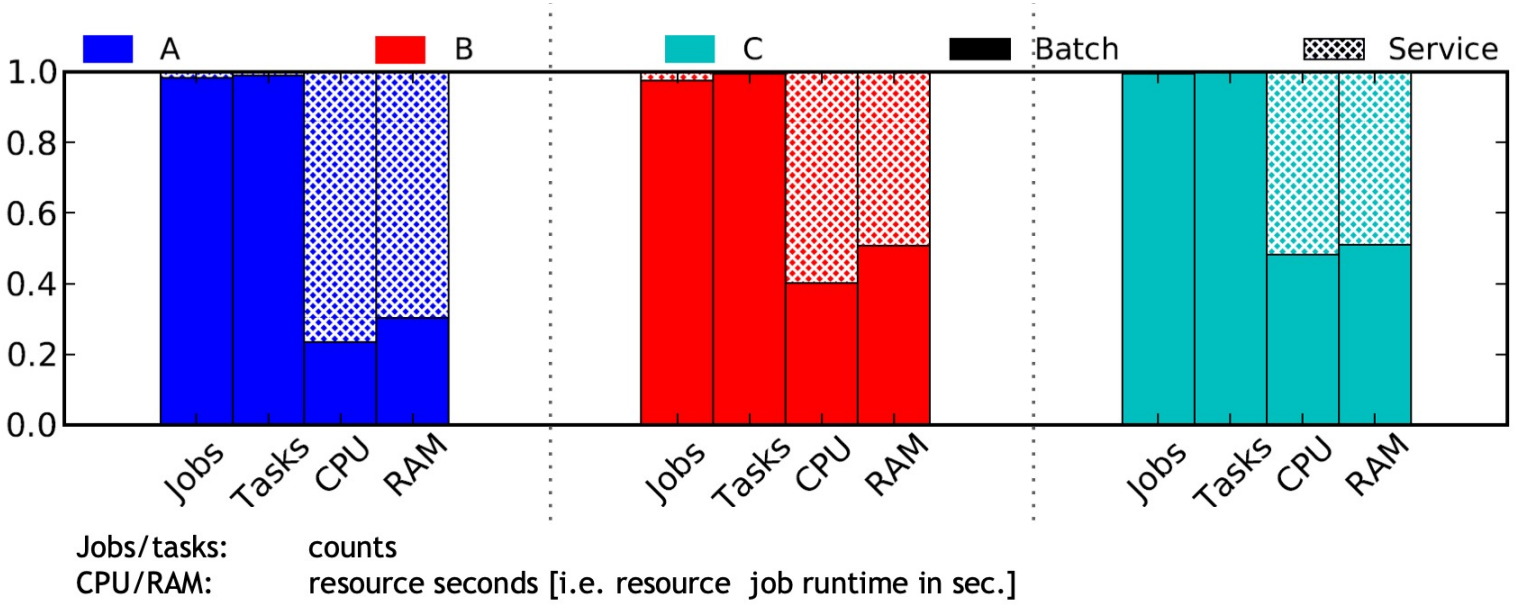
Jobs/tasks: counts
CPU/RAM: resource seconds [i.e. resource job runtime in sec.]

M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In ACM EuroSys, 2013.

Batch/service split

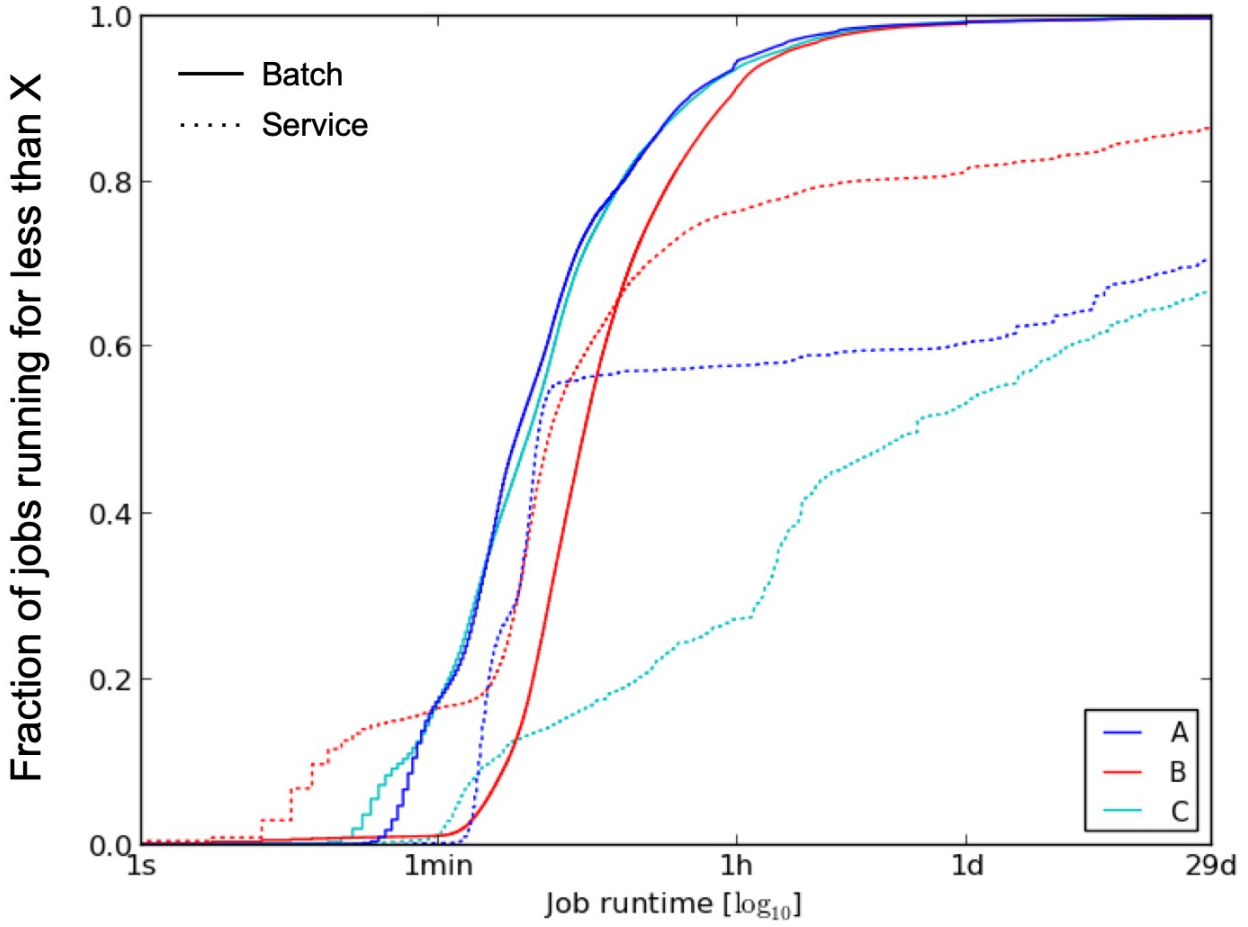
TAKEAWAY

Most jobs are batch, but most resources are consumed by service jobs.



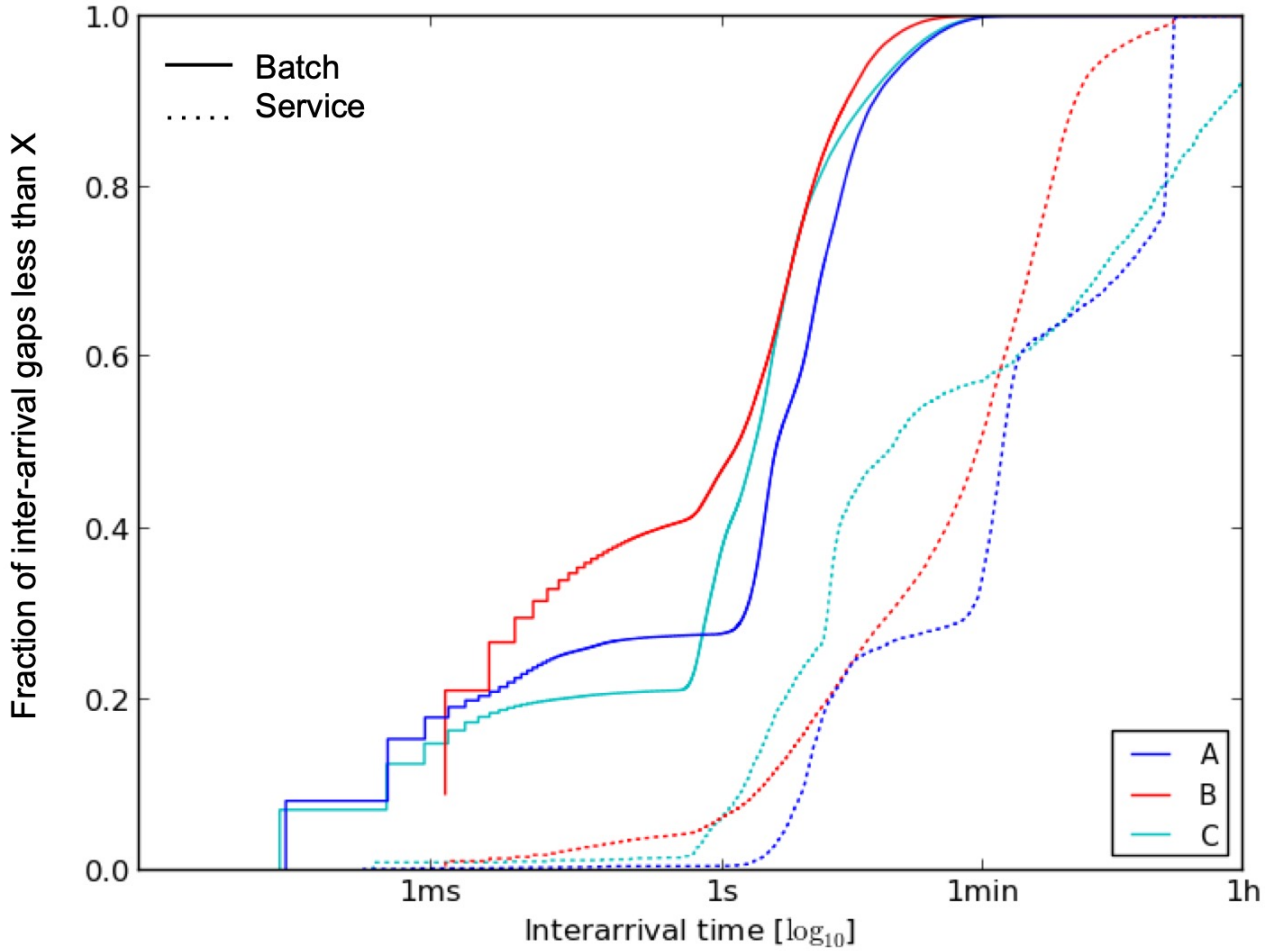
M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In ACM EuroSys, 2013.

Job runtime distributions



M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In ACM EuroSys, 2013.

Inter-arrival time distributions



M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In ACM EuroSys, 2013.

Batch/service split

Batch jobs	*Service jobs*
80th %ile **runtime**	
12-20 min.	**29 days**
80th %ile **inter-arrival time**	
4-7 sec.	**2-15 min.**

Scale matters

Big

- The storage system *holds* a Petabyte of data

Google

- The storage system pages you when there is only a few Petabytes of space left

Scale matters: faults

> 1%	Rate of uncorrectable DRAM errors per machine year
2-10%	Annual failure rate of disk drives
~2	Crashes/machine year
2-6	OS upgrades/machine year
> 1	Power utility events per year

Consequence: a 2000-machine service will have >10 machine crashes per day!

Scale matters: stragglers

High tail latency due to variability of response time

A toy example: a task that

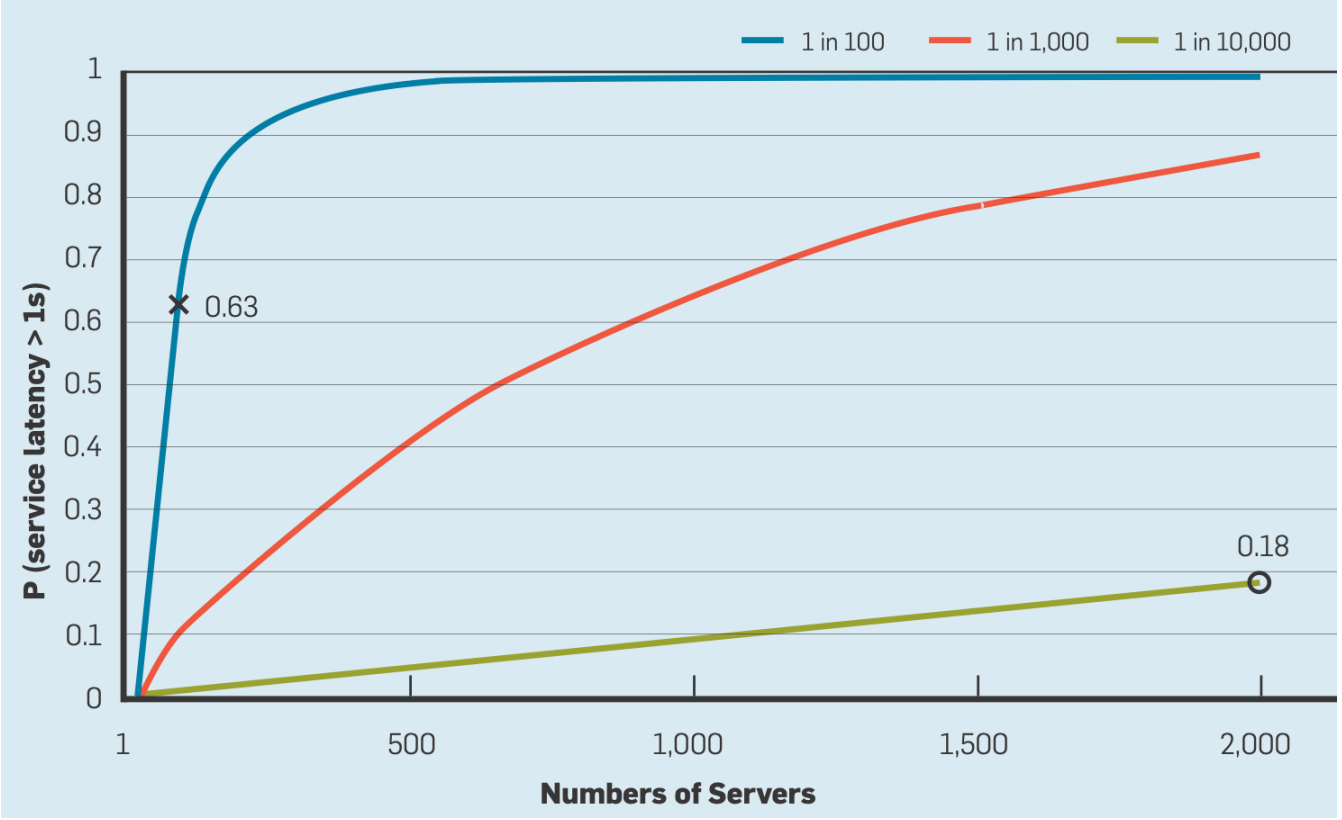
- completes in 1s w/ probability 0.9
- completes in 100s w/ probability 0.1

For a job with ***N*** tasks, what's the distribution of the job completion time?

- ***N*** = 10, 100, 1000

Stragglers: A non-toy

Probability of one-second Google service-level response time as the system scales and frequency of server-level high-latency outliers varies.



J. Dean and L. A. Barroso. The tail at scale. Commun. ACM, 56(2):74–80, 2013.

Cluster management: goals

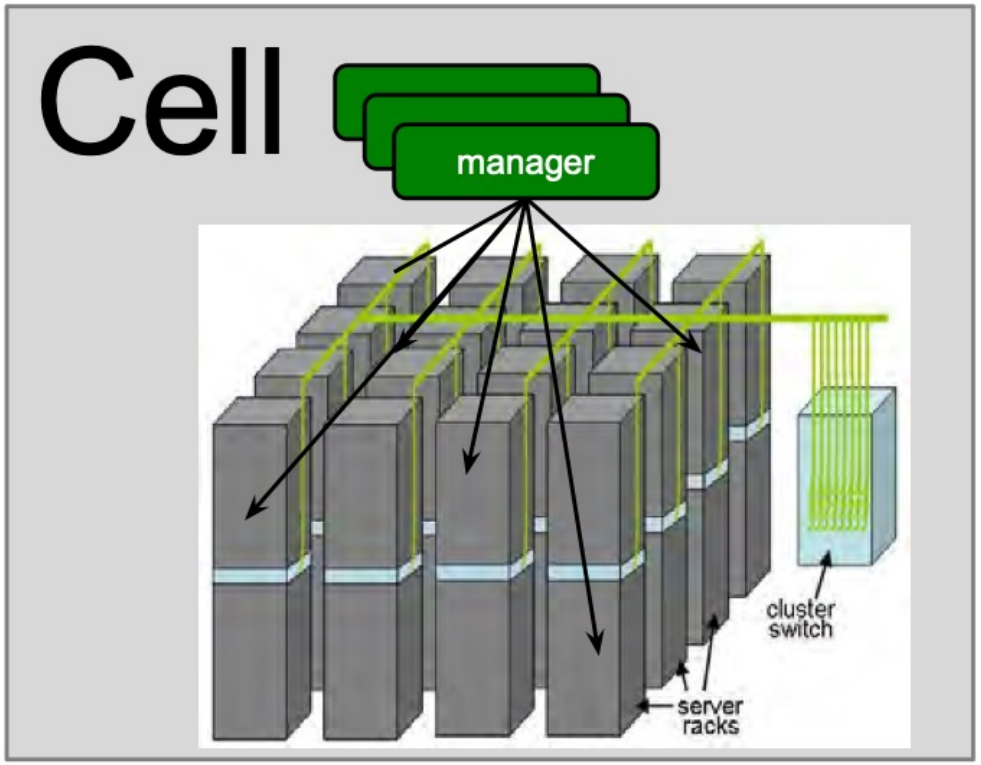
Run everything

Tolerate failures

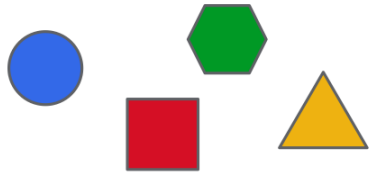
Predictable behavior

High utilization

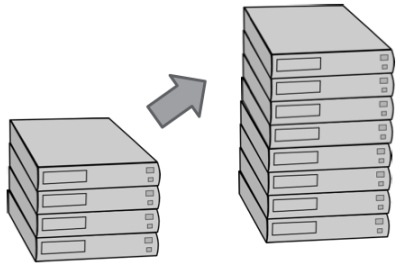
With very few people!



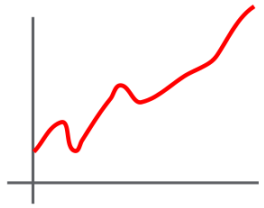
Let’s build a cluster scheduler



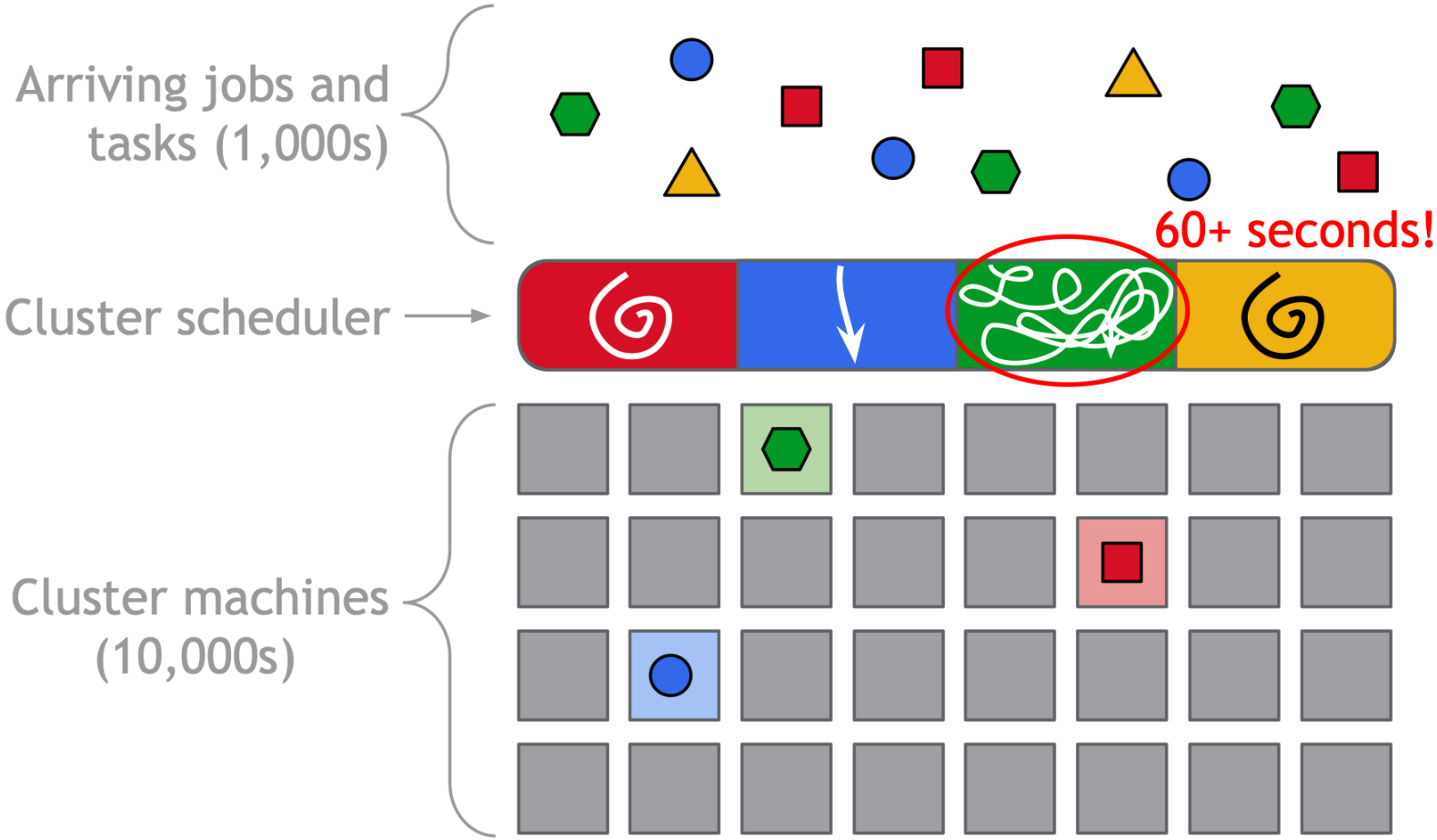
Diverse workloads



Increasing cluster sizes



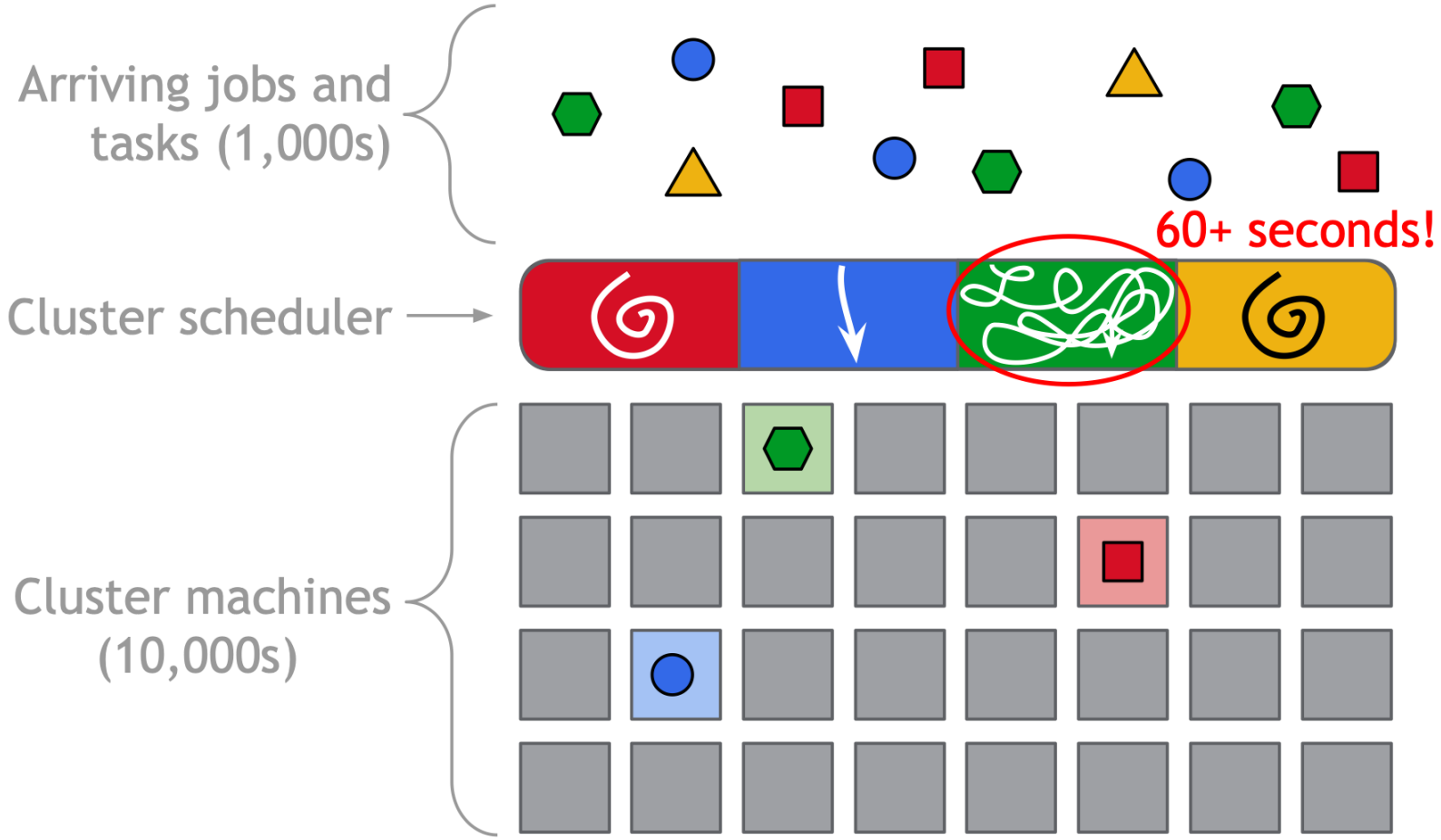
Growing job arrival rates


Arriving jobs and tasks (1,000s)
60+ seconds!
Cluster scheduler
Cluster machines (10,000s)

Why might scheduling take 60+ s?

Large jobs

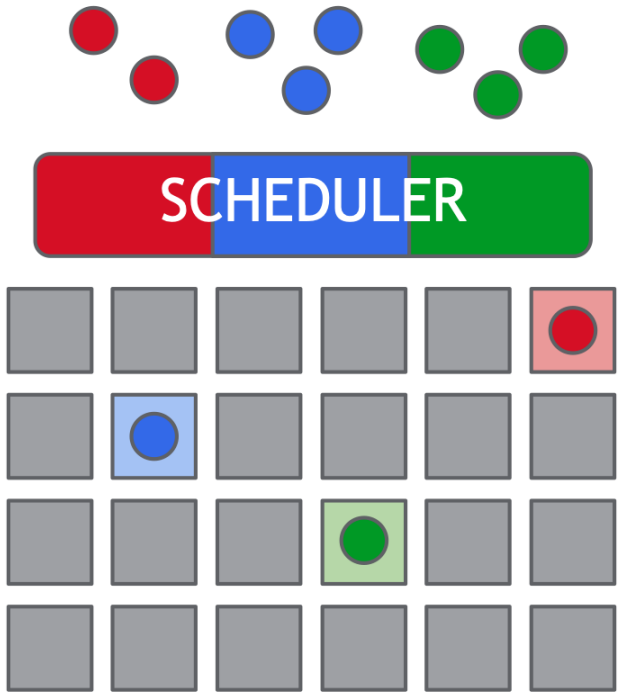
- tens of thousands of tasks
- optimization algorithms (constraints, bin packing w/ knock-on preemption)
- picky jobs in a full cluster



Hence: break up into independent schedulers, and arbitrate resources between them

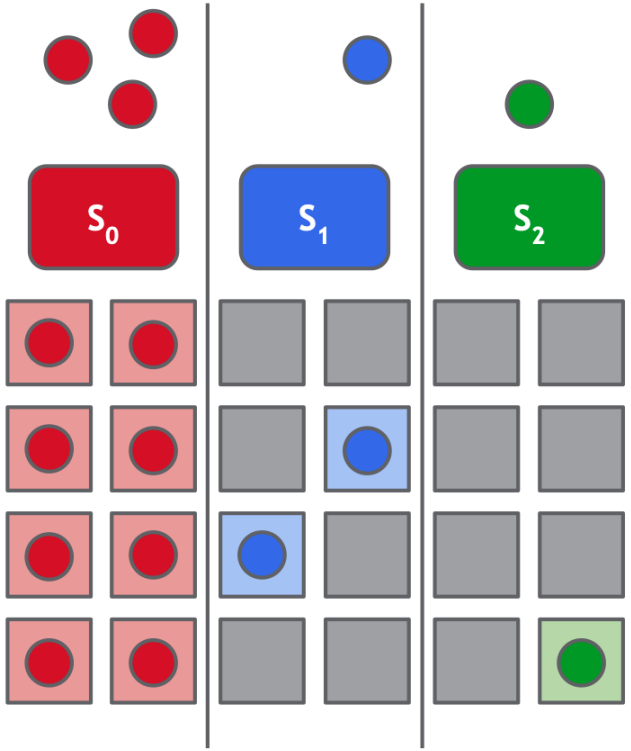
How to arbitrate resources?

monolithic scheduler



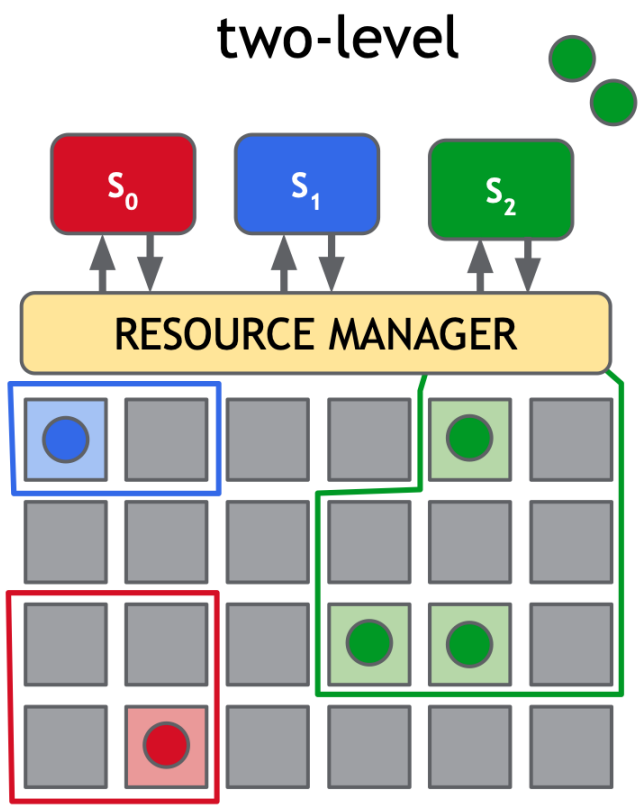
- hard to diversify
- code growth
- scalability bottleneck

static partitioning

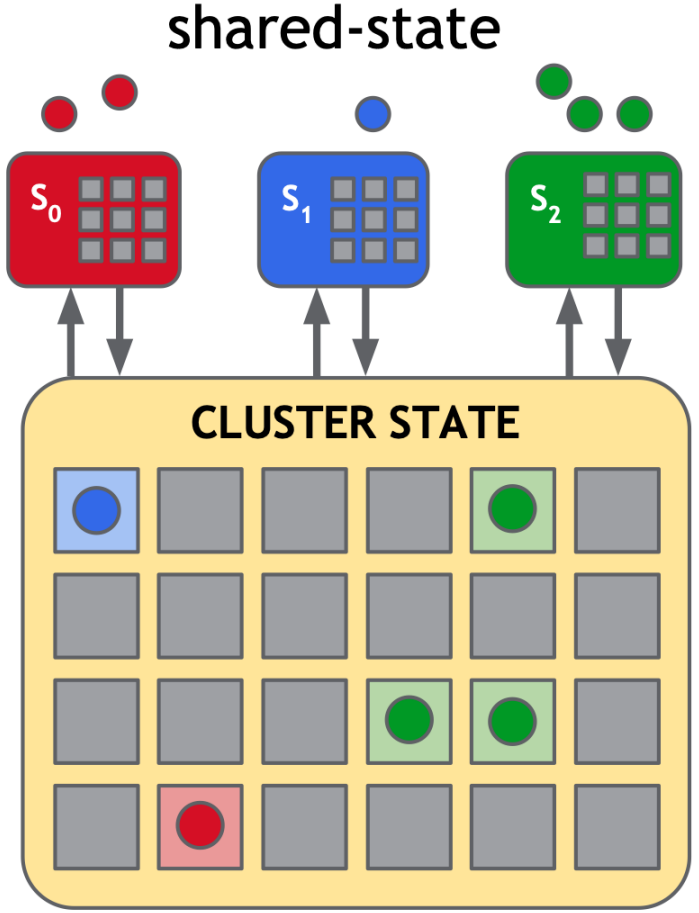


- poor utilization
- inflexible

Existing approaches



Mesos, YARN



Google's Omega

Mesos

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

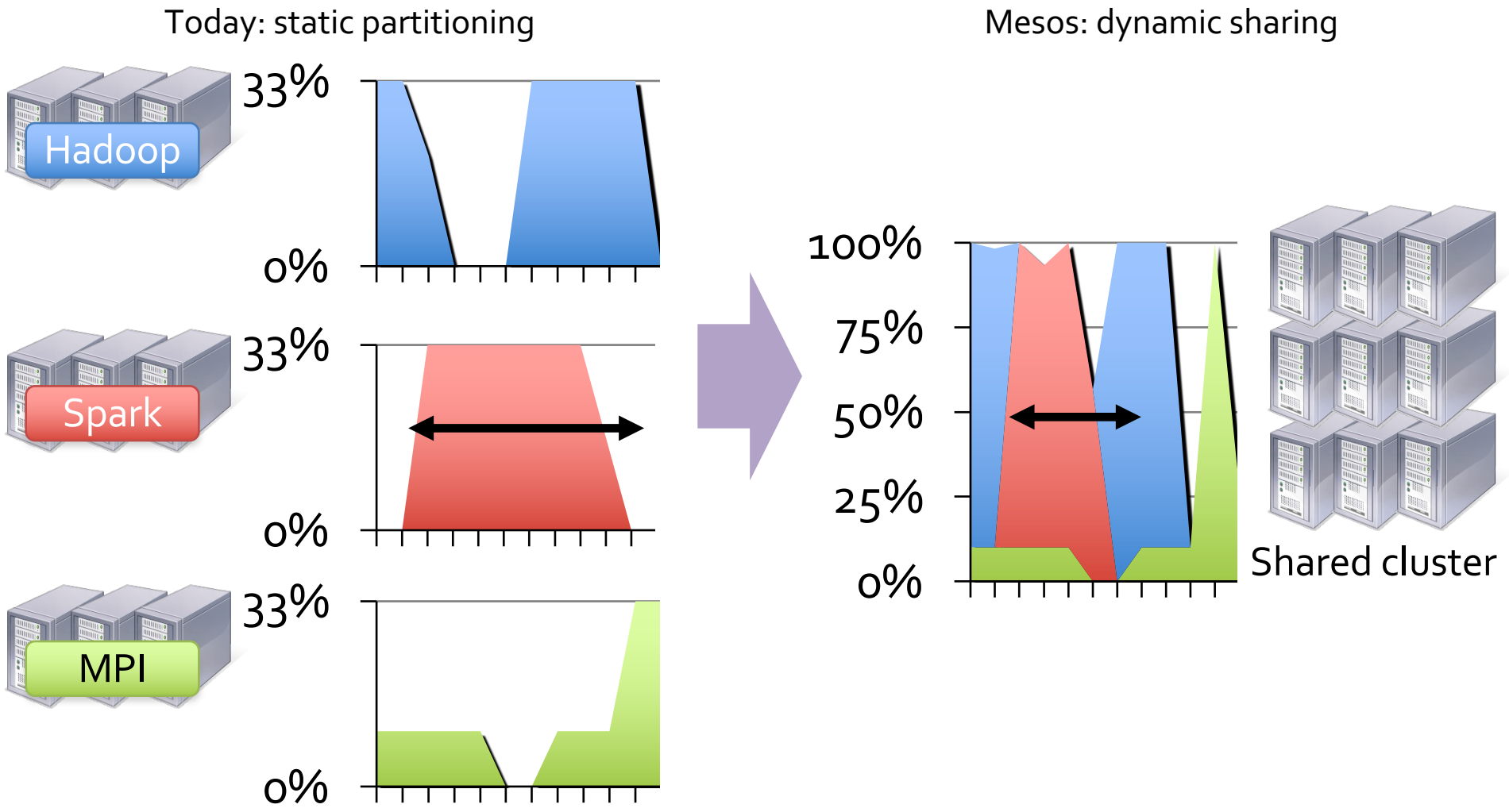
Benjamin Hindman, Andy Konwinski, Matei Zaharia,
Ali Ghodsi, Anthony Joseph, Randy Katz, Scott Shenker, Ion Stoica

University of California, Berkeley

Challenges and goals

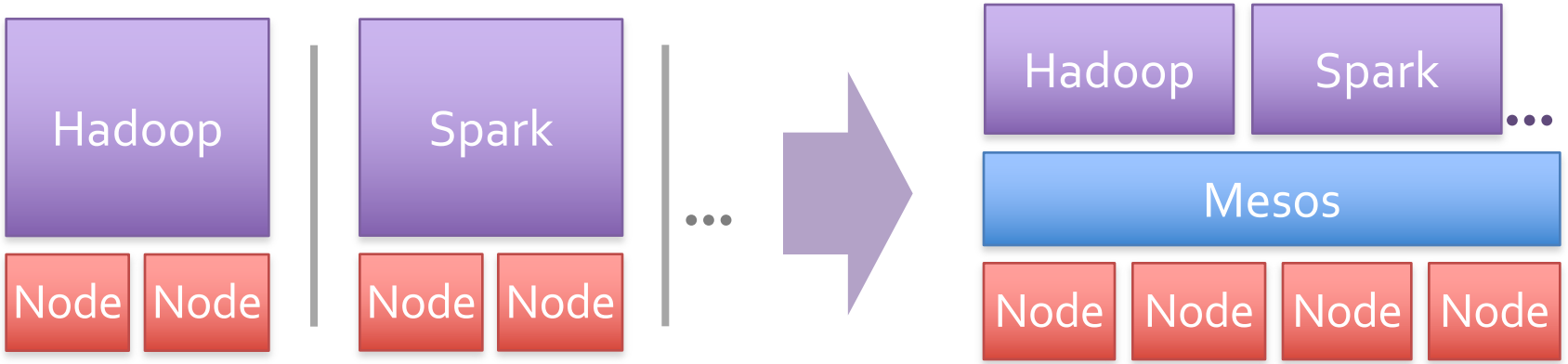
- High utilization
- Support diverse frameworks: each framework will have different scheduling needs
- Scalability: the scheduling system must scale to clusters of 1,000s of nodes running 100s of jobs with 1,000,000s of tasks
- Reliability: because all applications would depend on Mesos, the system must be **fault-tolerant** and **highly available.**

Where We Want to Go



Mesos

A thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common interface for accessing cluster resources.



Design Elements

Fine-grained sharing

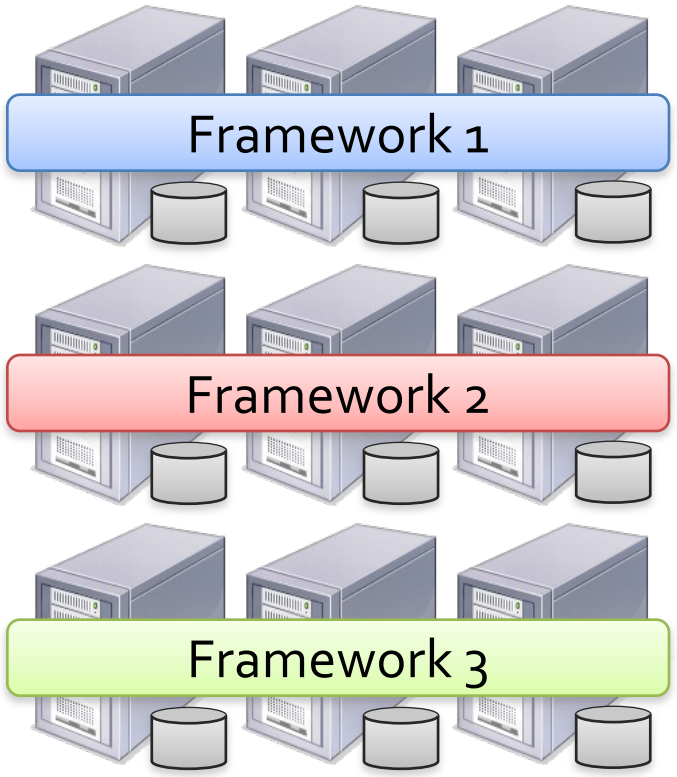
- Allocation at the level of tasks within a job
- Improves utilization, latency, and data locality

Resource offers

- Simple, scalable application-controlled scheduling mechanism

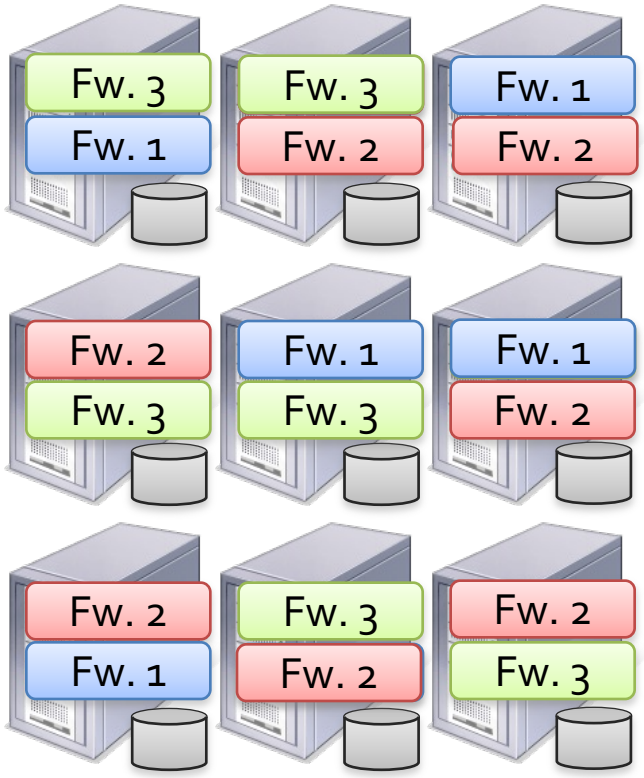
Element 1: Fine-Grained Sharing

Coarse-Grained Sharing (HPC):



Storage System (e.g. HDFS)

Fine-Grained Sharing (Mesos):



Storage System (e.g. HDFS)

+ Improved utilization, responsiveness, data locality

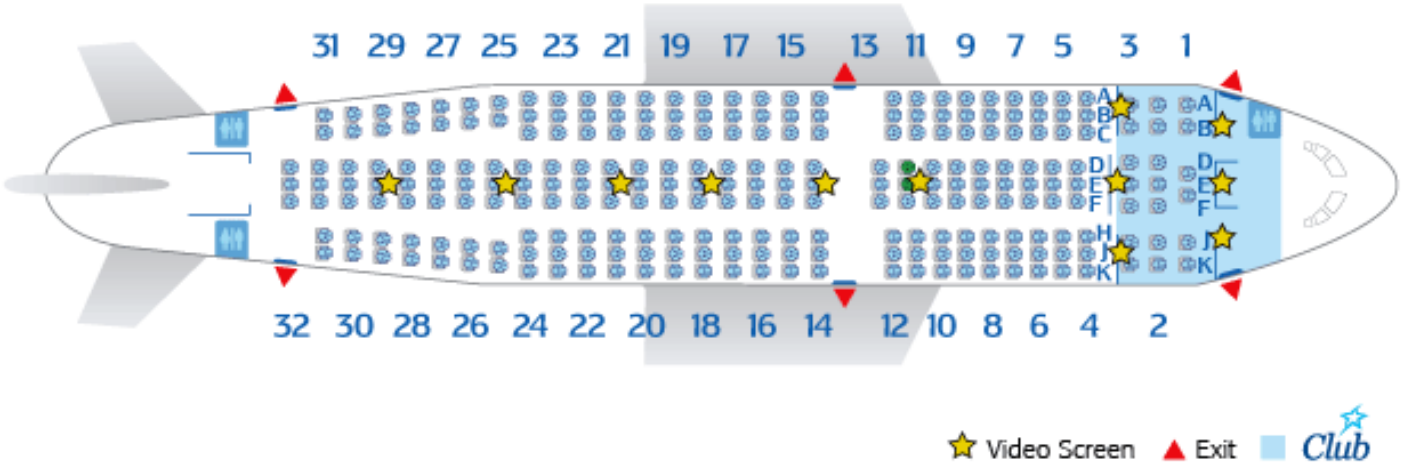
Element 2: Resource Offers

Mesos: Resource offers

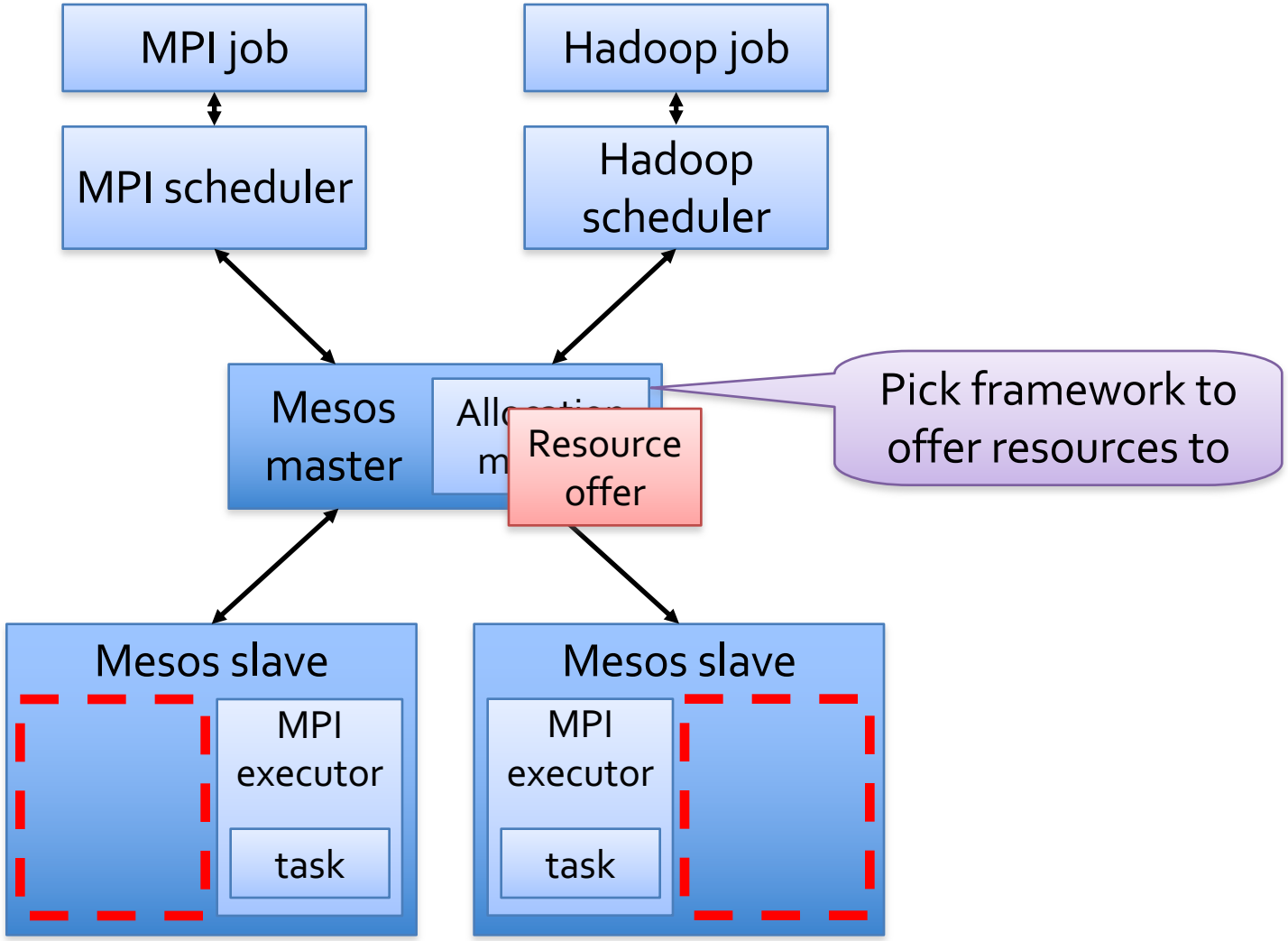
- Offer available resources to frameworks, let them pick which resources to use and which tasks to launch

\+ Keeps Mesos simple, lets it support future frameworks

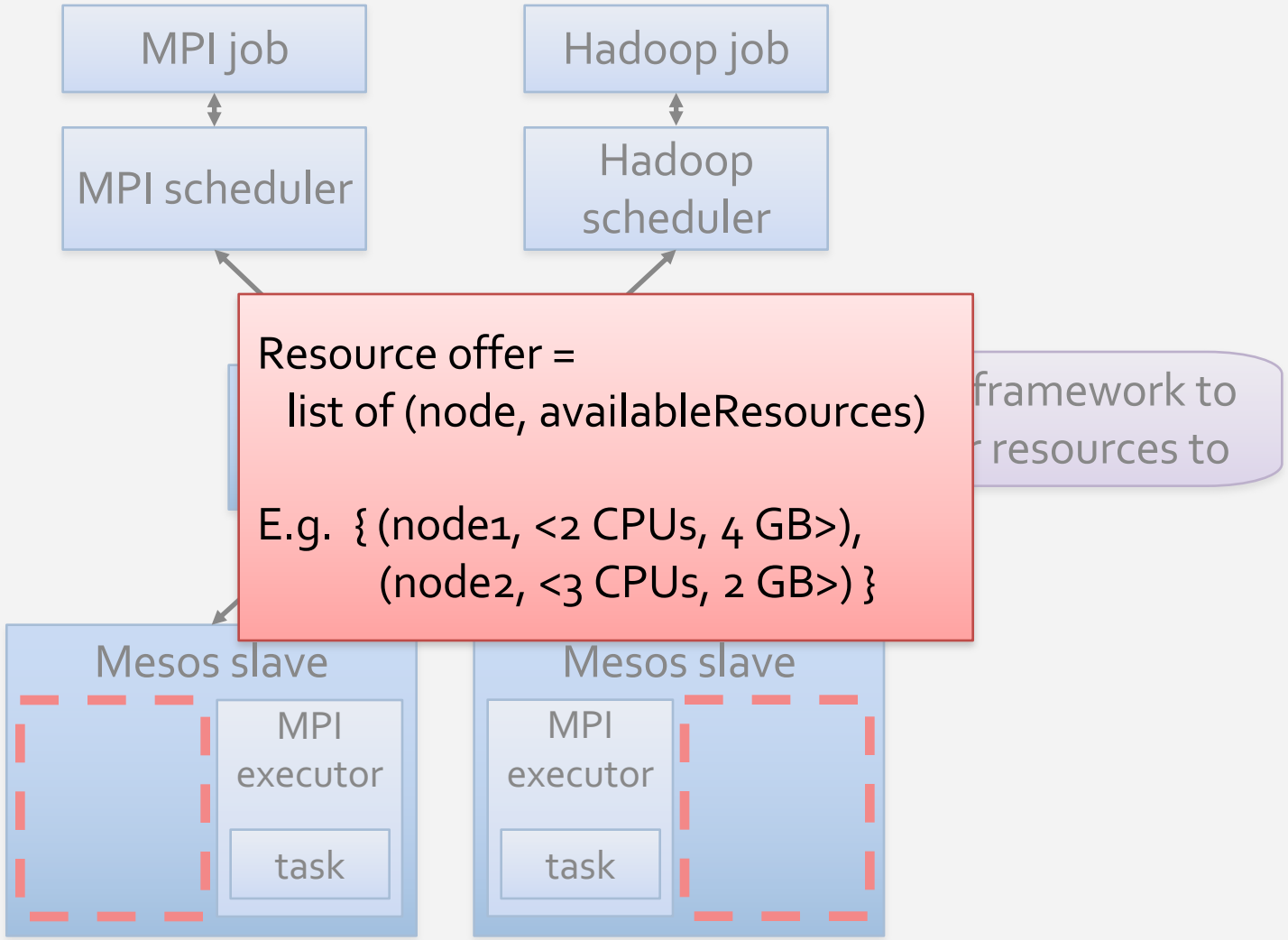
– Decentralized decisions might not be optimal



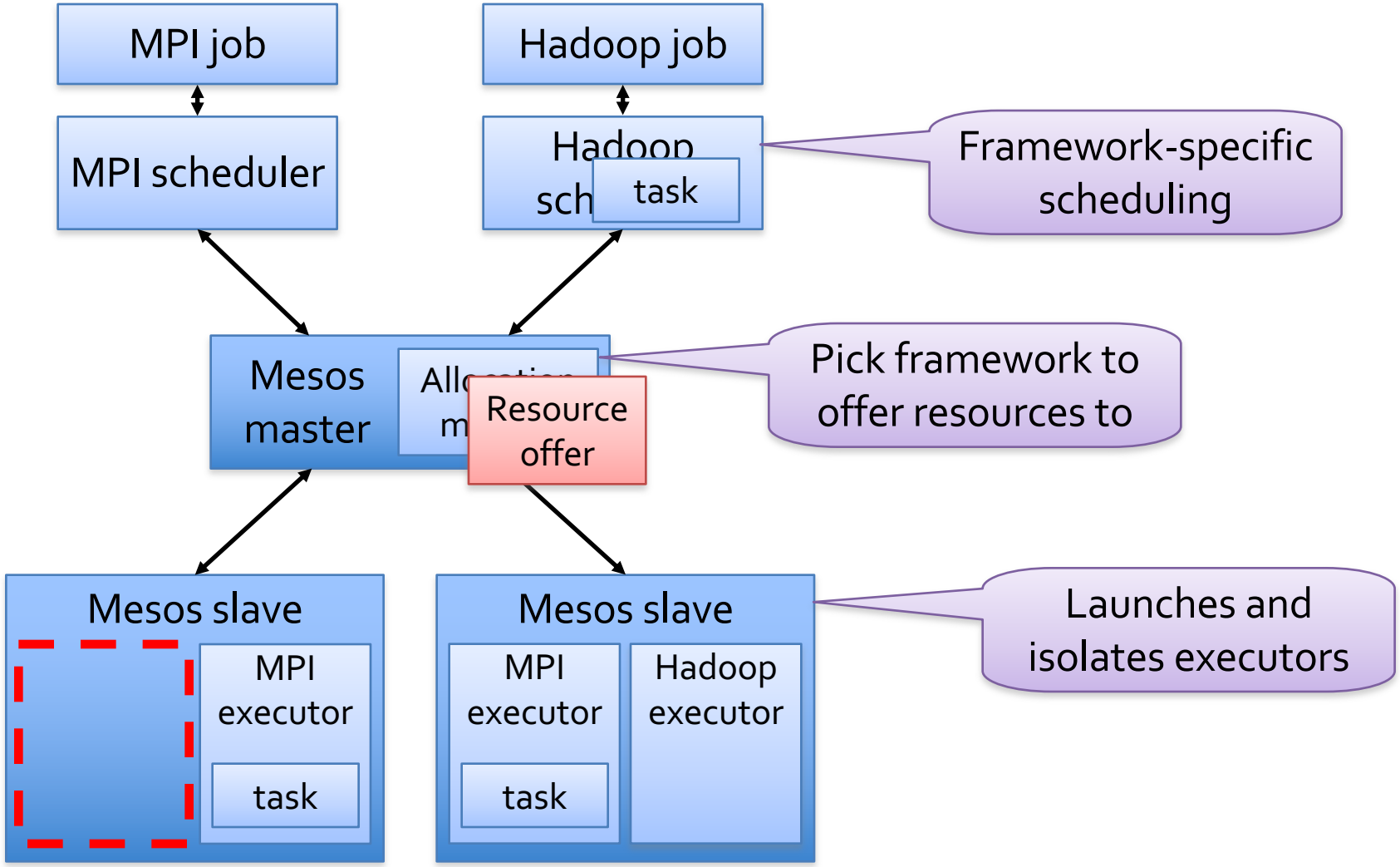
Mesos Architecture



Mesos Architecture



Mesos Architecture



Mesos Architecture

- Mesos master: fine-grained sharing across frameworks
- Mesos slave on each node
- Frameworks that run tasks on slave nodes
- Framework schedulers

“While the Mesos master determines how many resources to offer to each framework, the frameworks’ schedulers select which of the offered resources to use”.

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

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

Revocation

Mesos allocation modules can revoke (kill) tasks to meet organizational SLOs

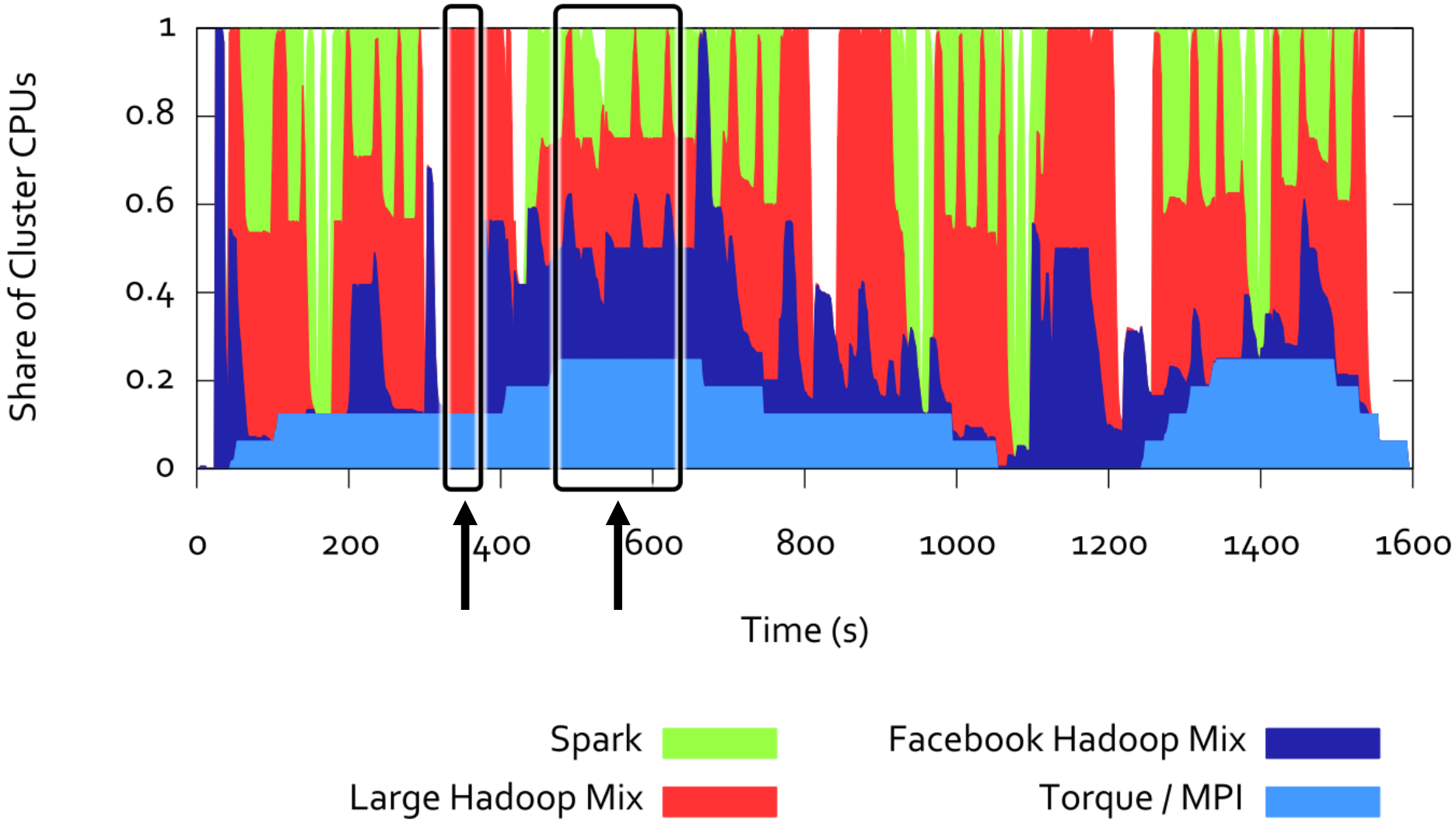
Framework given a grace period to clean up

“Guaranteed share” API lets frameworks avoid revocation by staying below a certain share

Results

- Utilization and performance vs static partitioning
- Framework placement goals: data locality
- Scalability
- Fault recovery

Dynamic Resource Sharing



Mesos vs Static Partitioning

Compared performance with statically partitioned cluster where each framework gets 25% of nodes

Framework	Speedup on Mesos
Facebook Hadoop Mix	1.14×
Large Hadoop Mix	2.10×
Spark	1.26×
Torque / MPI	0.96×

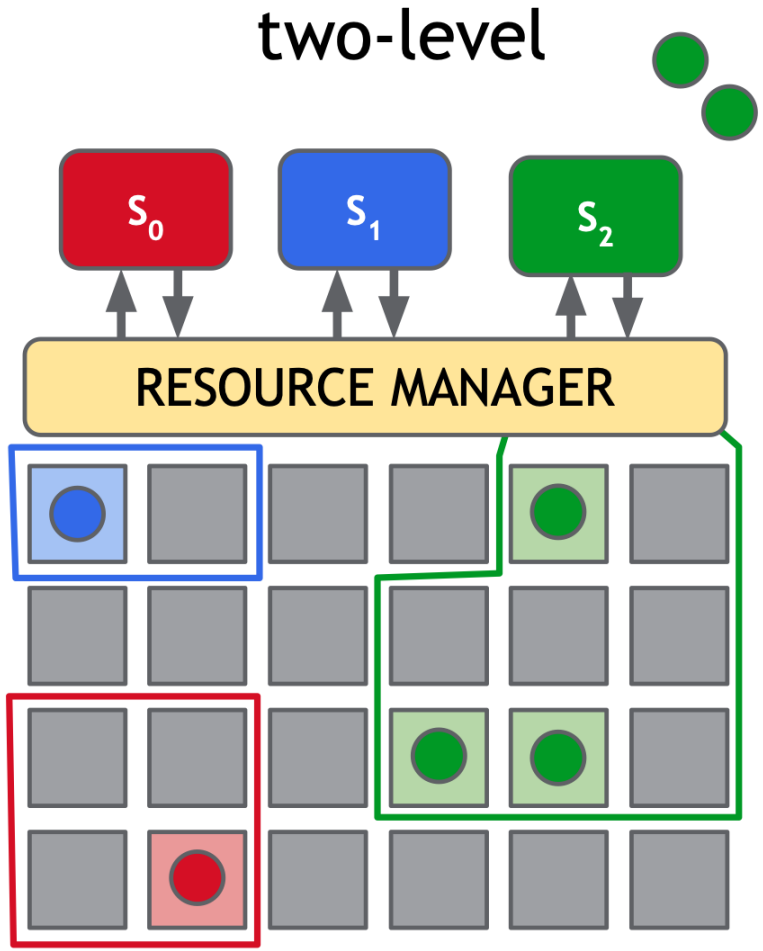
Mesos's problem

Hoarding

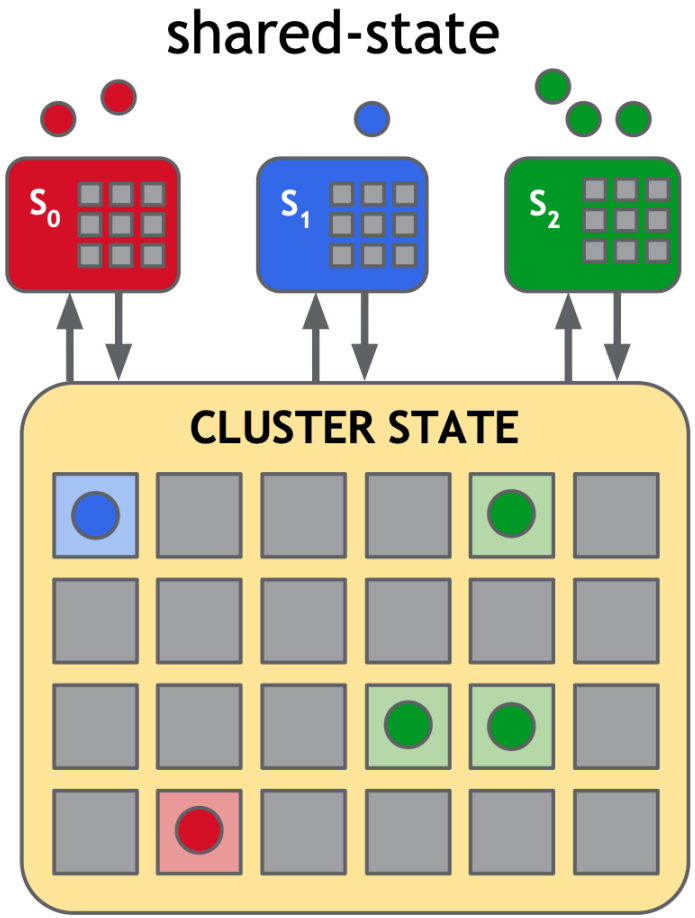
- Mesos avoids conflicts by offering a given resource to one framework at-a-time. Mesos chooses the order. One framework holds the lock of the resources for the duration of the decision → slow

Information hiding

- A framework does not have access to all the cluster state. Hard for preemption or getting all resources.



Omega: shared-state



Cell state

- a resilient master copy of the resource allocations of the cluster

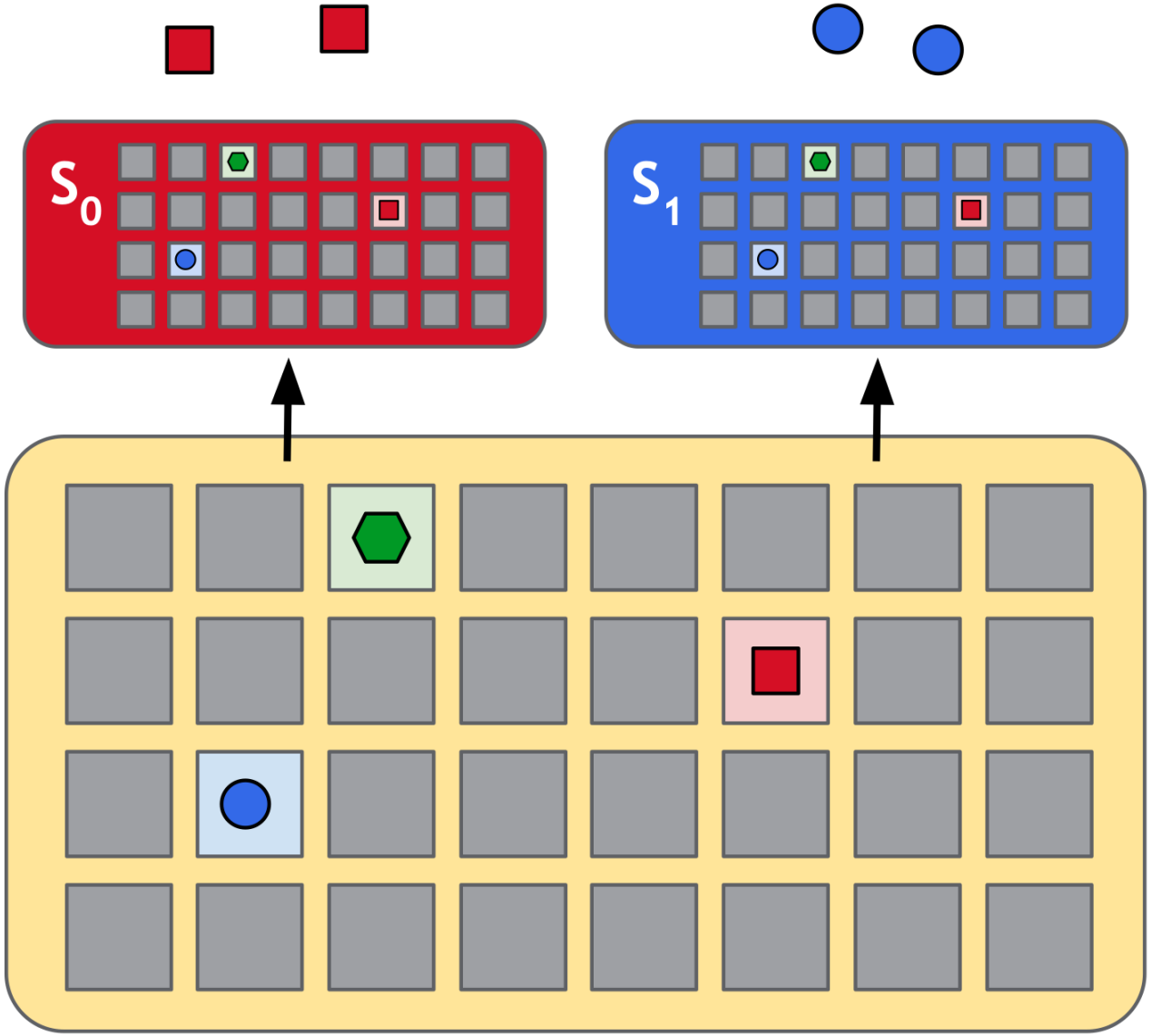
No centralized scheduler

- Each framework maintains its own scheduler. Each scheduler has its own private, local, frequently-updated copy of the cell state which uses to make decisions

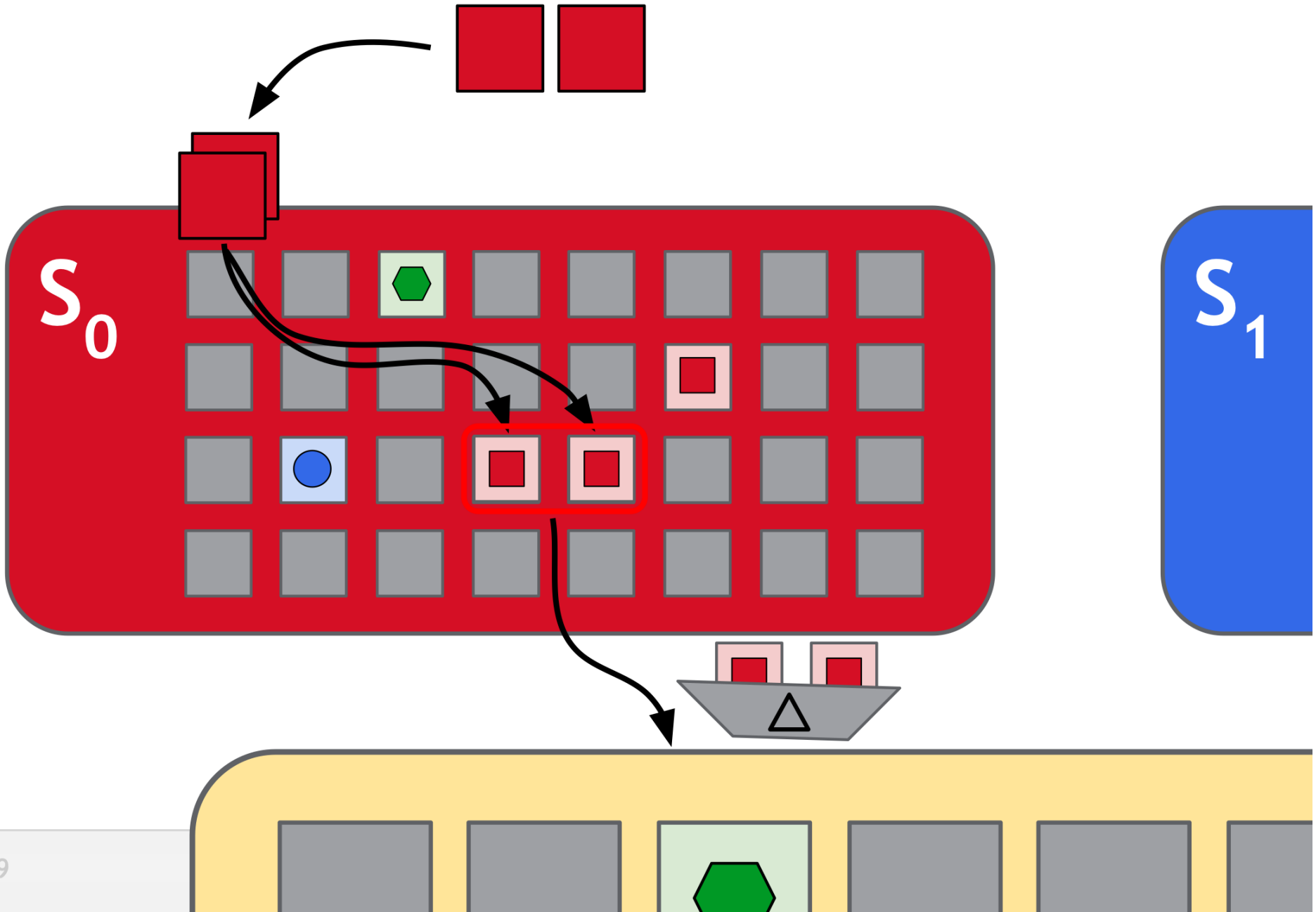
Omega shared-state scheduling

- A framework's scheduler makes a decision
- Updates the shared cell state copy in an atomic commit
- At most one such commit will succeed
- The scheduler resyncs its local copy and if needed re-schedules

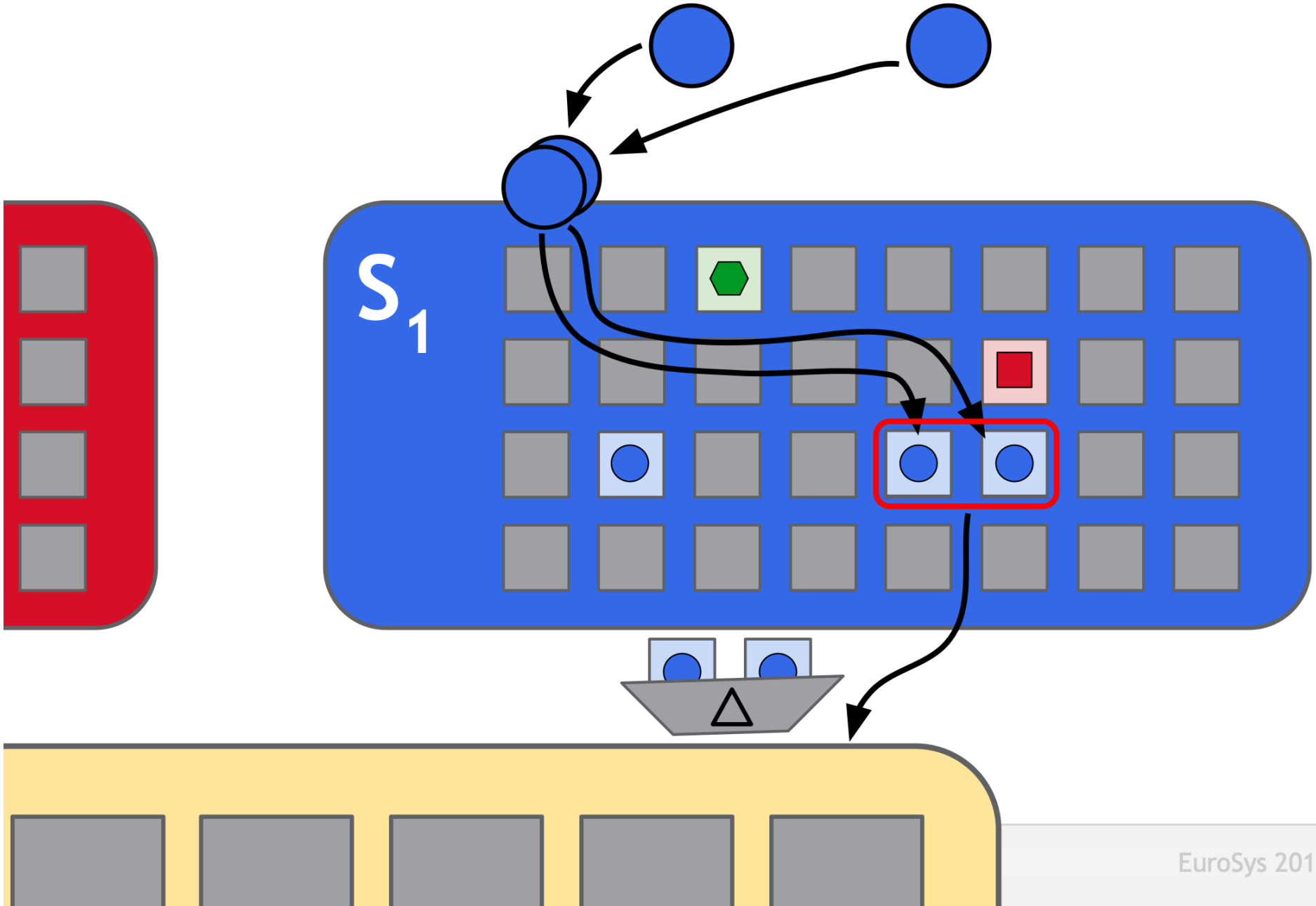
how does omega work?



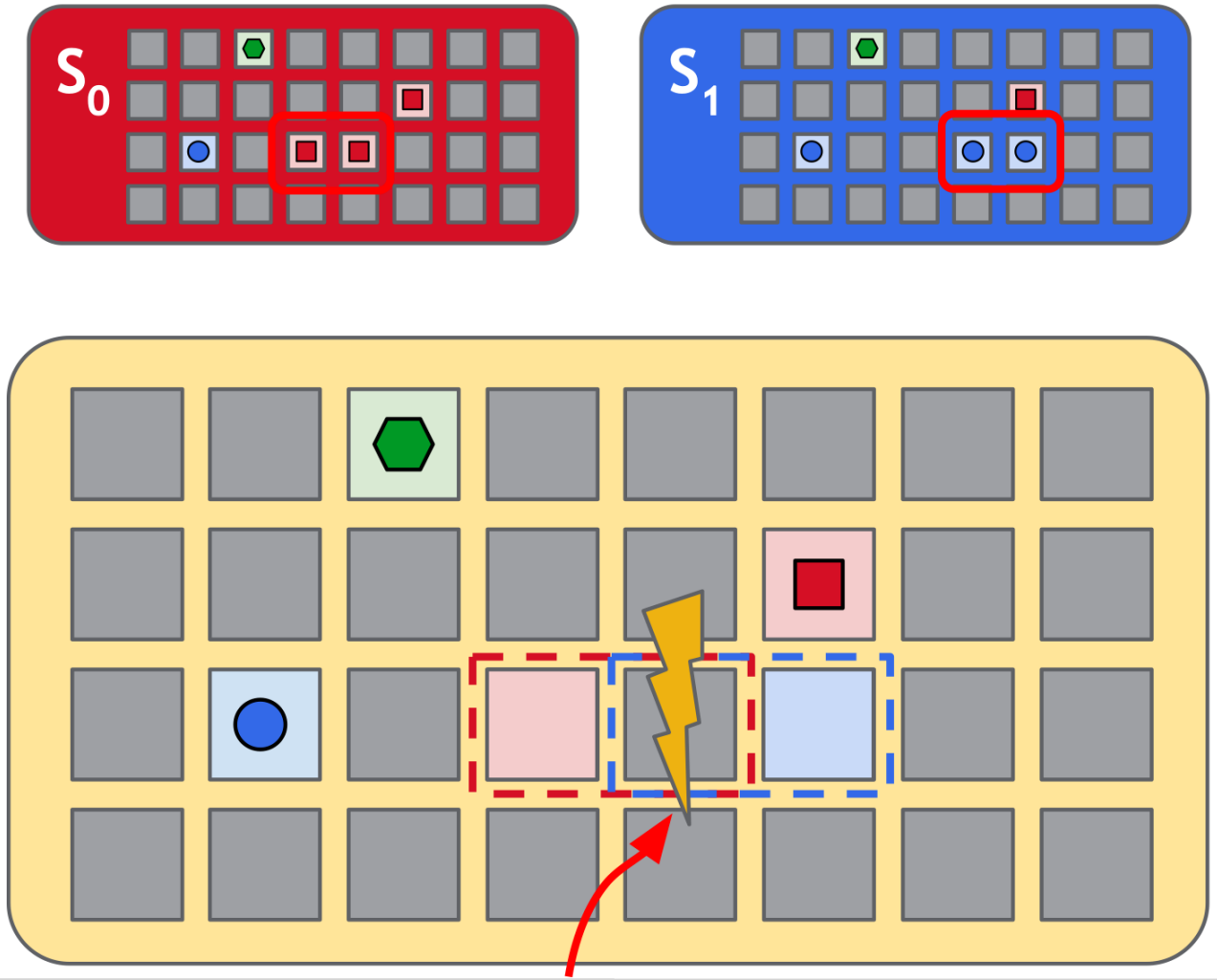
how does omega work?



how does omega work?



how does omega work?



Omega's Takeaways

- Flexibility and scale require parallelism
- Parallel scheduling works if you do it right
- Using shared state is the way to do it right

Credits

John Wilkes, Google

- M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In ACM EuroSys, 2013.

Matei Zaharia, Stanford & Databricks

- B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In USENIX NSDI, 2011.