Big Data systems

From the streets of silicon valley to the 2013 World Tour

Carlo Curino

Agenda

History

- 2003 cluster as an embedded device
- 2013 the advent of general purpose computing

Cluster OS (new abstraction)

- Mesos / Omega / YARN

Supports for new applications

- Apps, services and Meta-frameworks

History

“From the dive bars of silicon valley to the World Tour”

The origin

Purpose-built technology

- Within large web companies
- Well targeted mission (process webcrawl)
- → scale and fault tolerance

Google leading the pack

- Google File System + MapReduce (2003/2004)

Open-source and parallel efforts

- Yahoo! Hadoop ecosystem HDFS + MR (2006/2007)
- Microsoft Scope/Cosmos (2008) (more than MR)

In-house growth

What was the key to success for Hadoop?

In-house growth (following Hadoop story)

Access, access, access…

- *All the data* sit in the DFS
- *Trivial to* use massive compute power
- → lots of new applications

But… everything has to be MR

- Cast any computation as map-only job
- MPI, graph processing, streaming, *launching web-servers!?!*

Popularization

Everybody wants Big Data

- Insight from raw data is cool
- Outside MS and Google, Big-Data == Hadoop
- Hadoop as catch-all big-data solution (and cluster manager)

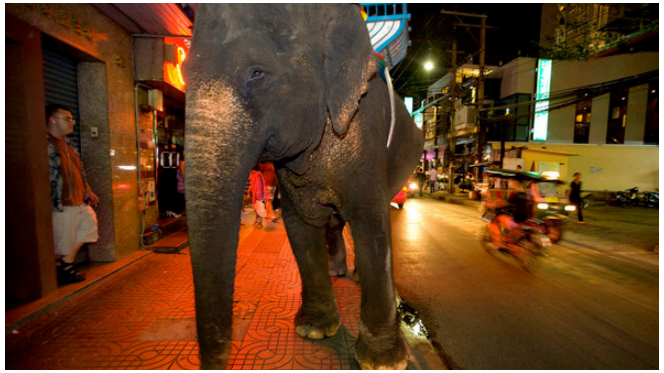
Not just massive in-house clusters

New deployment environments

- Small clusters (10s of machines)
- Public Cloud

New challenges?

New deployment challenges



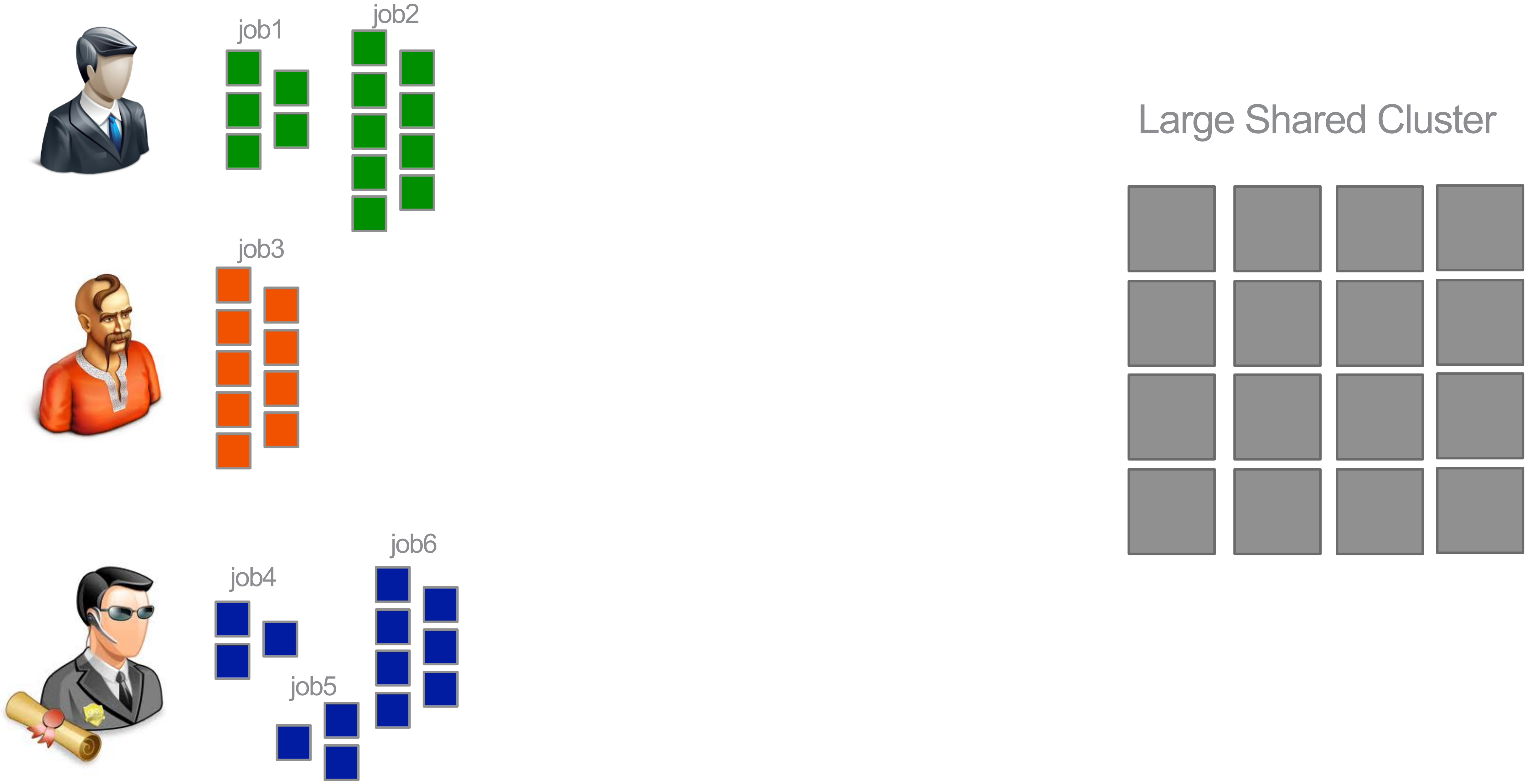
Small clusters

- Efficiency matter more than scalability
- Admin/tuning done by mere mortals

Cloud

- Untrusted users (security)
- Users are paying (availability, reliability)
- Users are unrelated to each other (performance isolation)

Classic MapReduce



job1

job2

job3

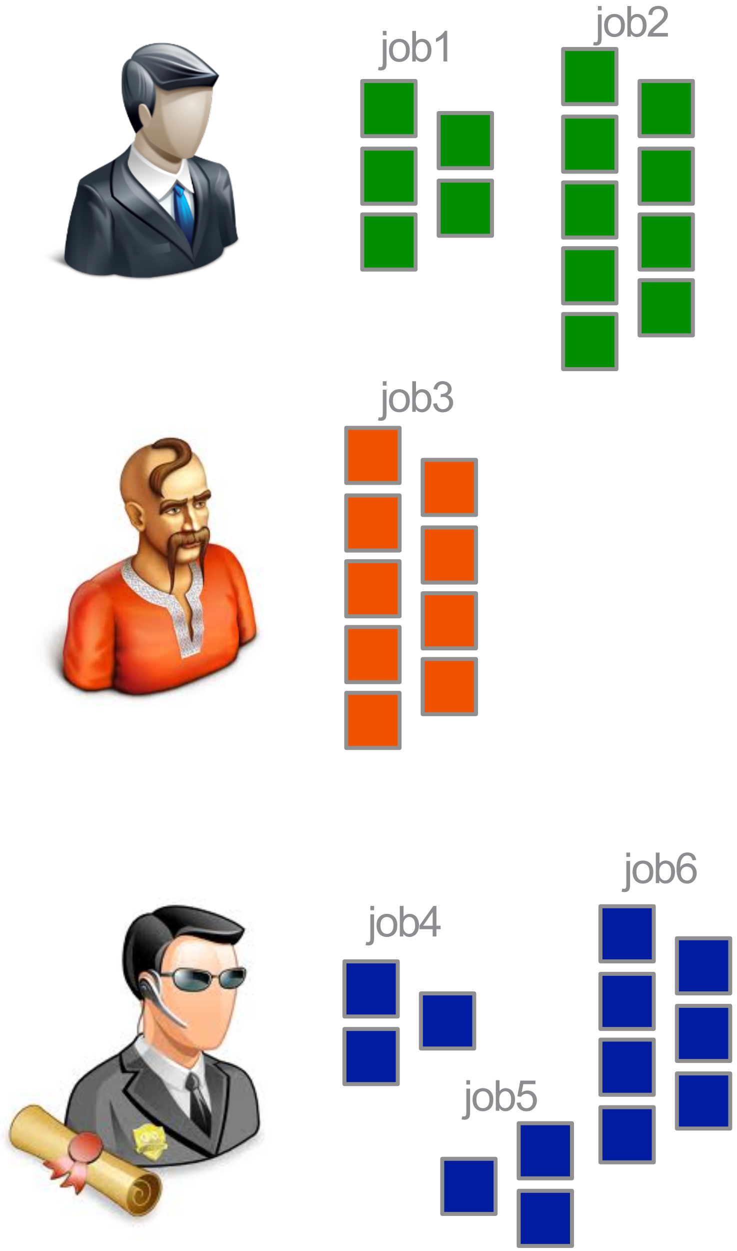
job4

job5

job6

Large Shared Cluster

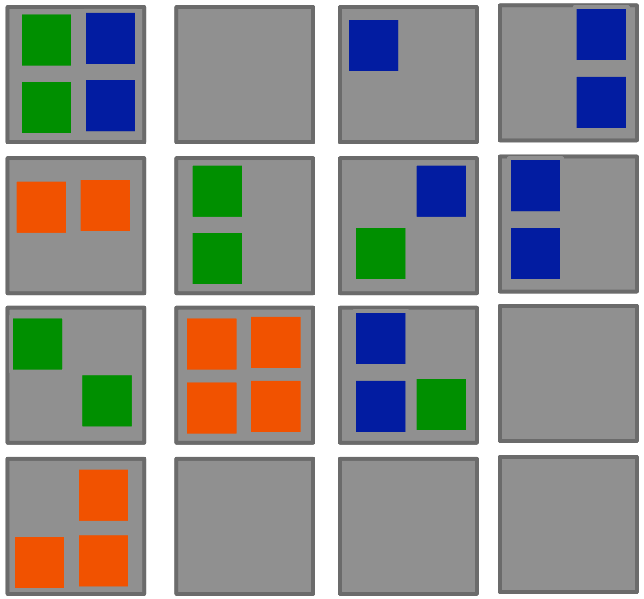
Classic MapReduce



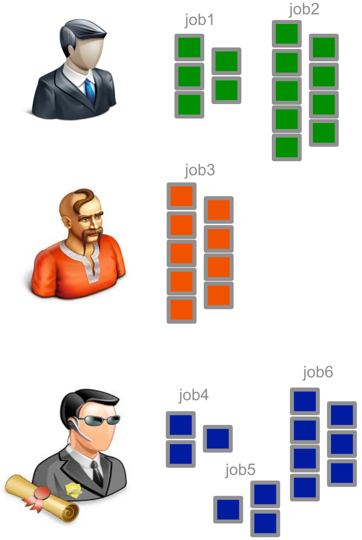
Problems to solve

- Who runs?
- Where?
- How much resources?
- Order of execution
- Monitor progress
- Handle failures

Large Shared Cluster



Classic MapReduce



Resource Management

- Who runs?
- Where?
- How much resources?

→

Key Metrics

- Cluster throughput
- Scheduling Invariants
- Efficiency (locality)

Application Workflow

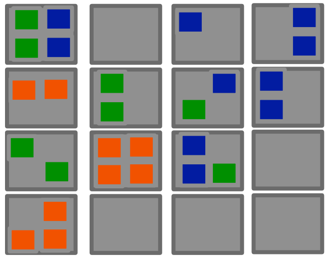
- Order of execution
- Monitor progress
- Handle failures

→

Key Metrics

- Fault tolerance
- App Semantics

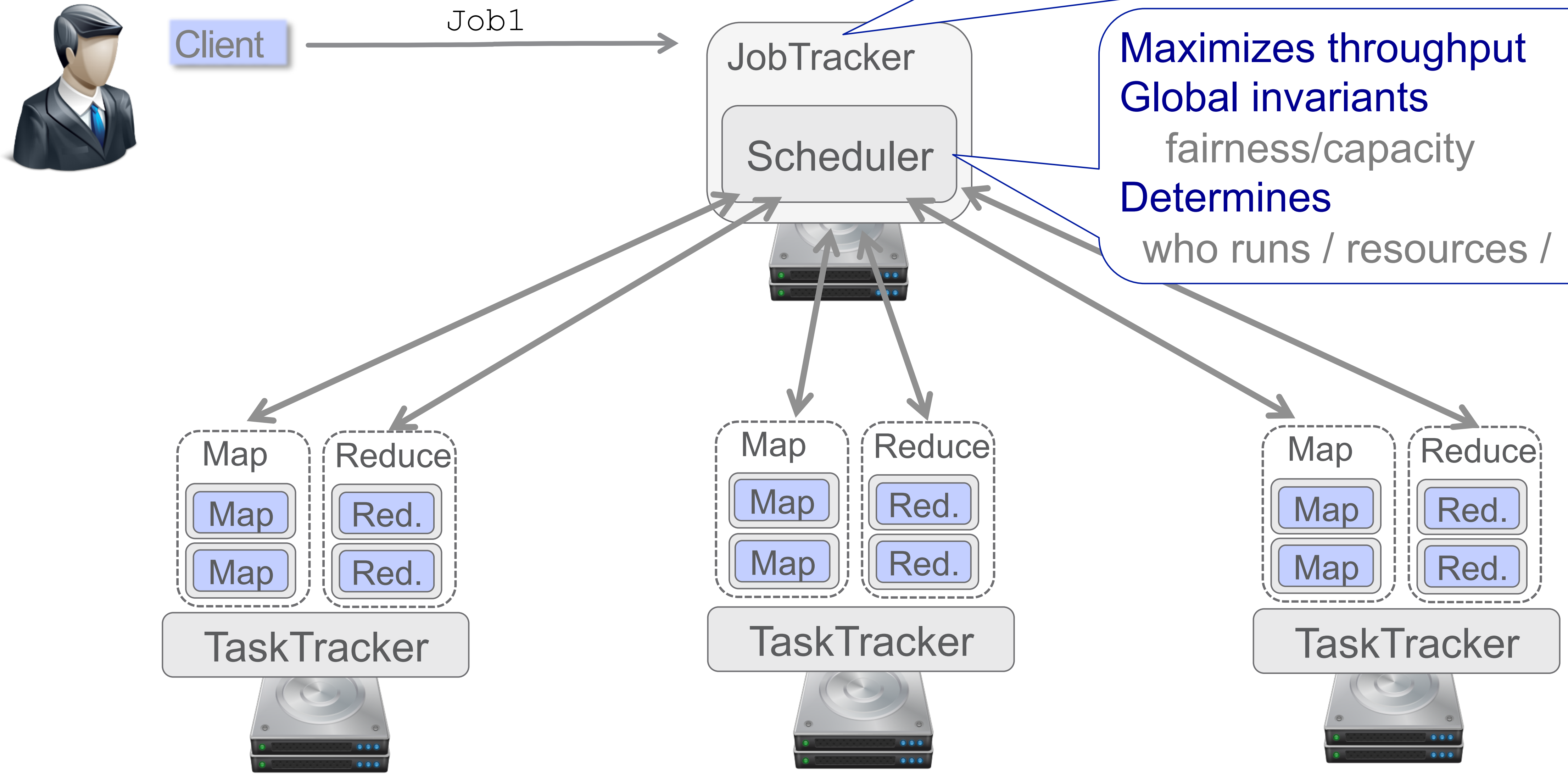
Large Shared Cluster



Classic Hadoop (1.0) Architecture

Manages MapReduce application flow
maps before reducers, re-run upon failure, etc..

Maximizes throughput
Global invariants
fairness/capacity
Determines
who runs / resources / where



Hadoop 1.0 Shortcomings

What are the key shortcomings of (old) Hadoop?

Hadoop 1.0 Shortcomings (similar to original MR)

Programming model rigidity

- JobTracker manages resources
- JobTracker manages application workflow (data dependencies)

Performance and Availability

- Map vs Reduce slots lead to *low cluster utilization* (~70%)
- JobTracker had too much to do: *scalability concern*
- JobTracker is a single point of failure

Cluster OS

“…towards general purpose computing…”

Three proposals

YARN (2008-2013, Hadoop 2.x, production at Yahoo!, GA)

Request-based central scheduler

Mesos (2011, UCB, open-sourced, tested at Twitter)

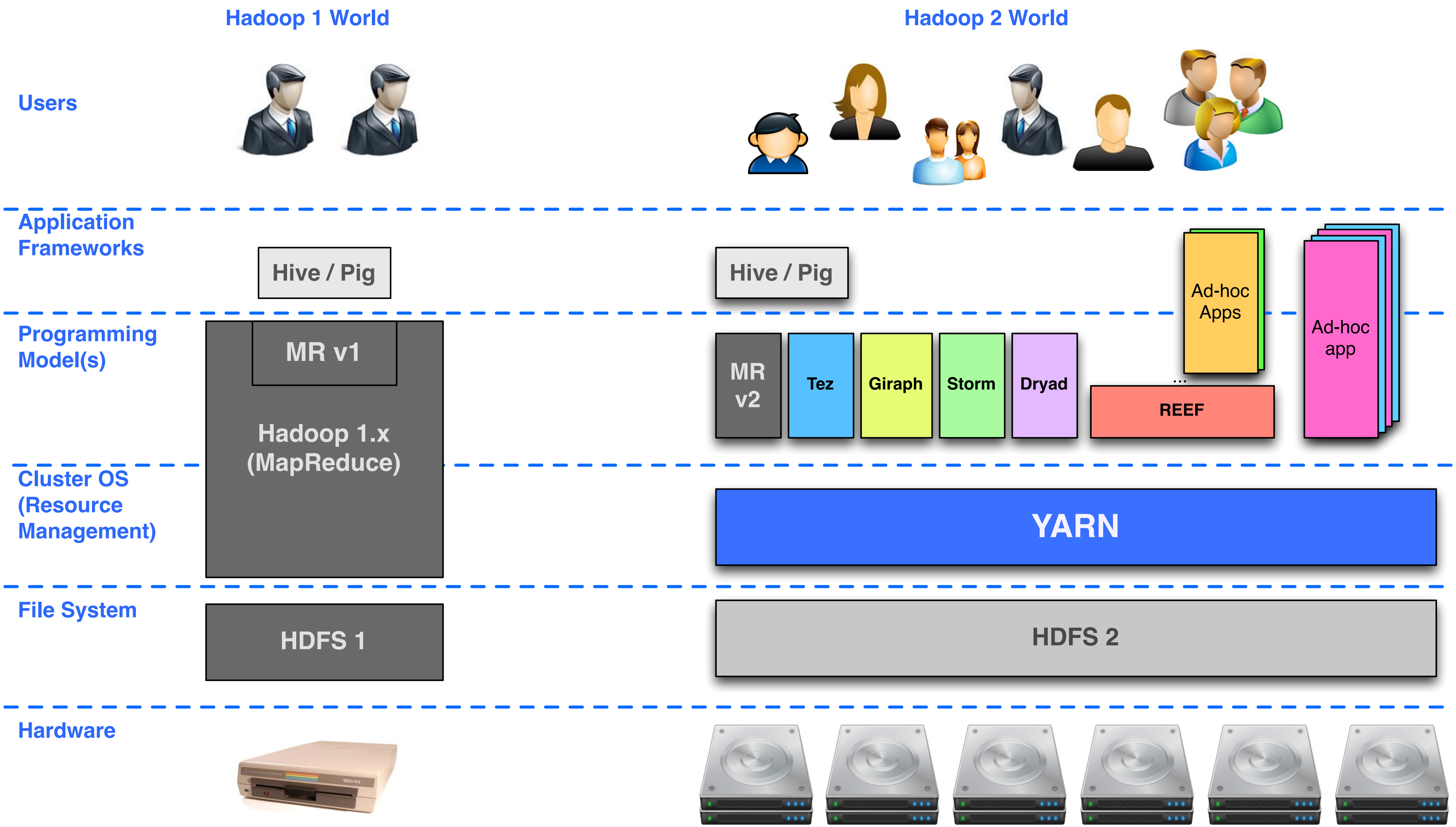
Offer-based two level scheduler

Omega (2013, Google, simulation)

Shared-state-based scheduling

Note: all three were best-papers or best-student-paper, I borrowed slides from all three talks

YARN



Hadoop 1 World

Hadoop 2 World

Users

Application Frameworks

Hive / Pig

Hive / Pig

Ad-hoc Apps

Ad-hoc app

Programming Model(s)

MR v1

Hadoop 1.x (MapReduce)

MR v2

Tez

Giraph

Storm

Dryad

…

REEF

Cluster OS (Resource Management)

YARN

File System

HDFS 1

HDFS 2

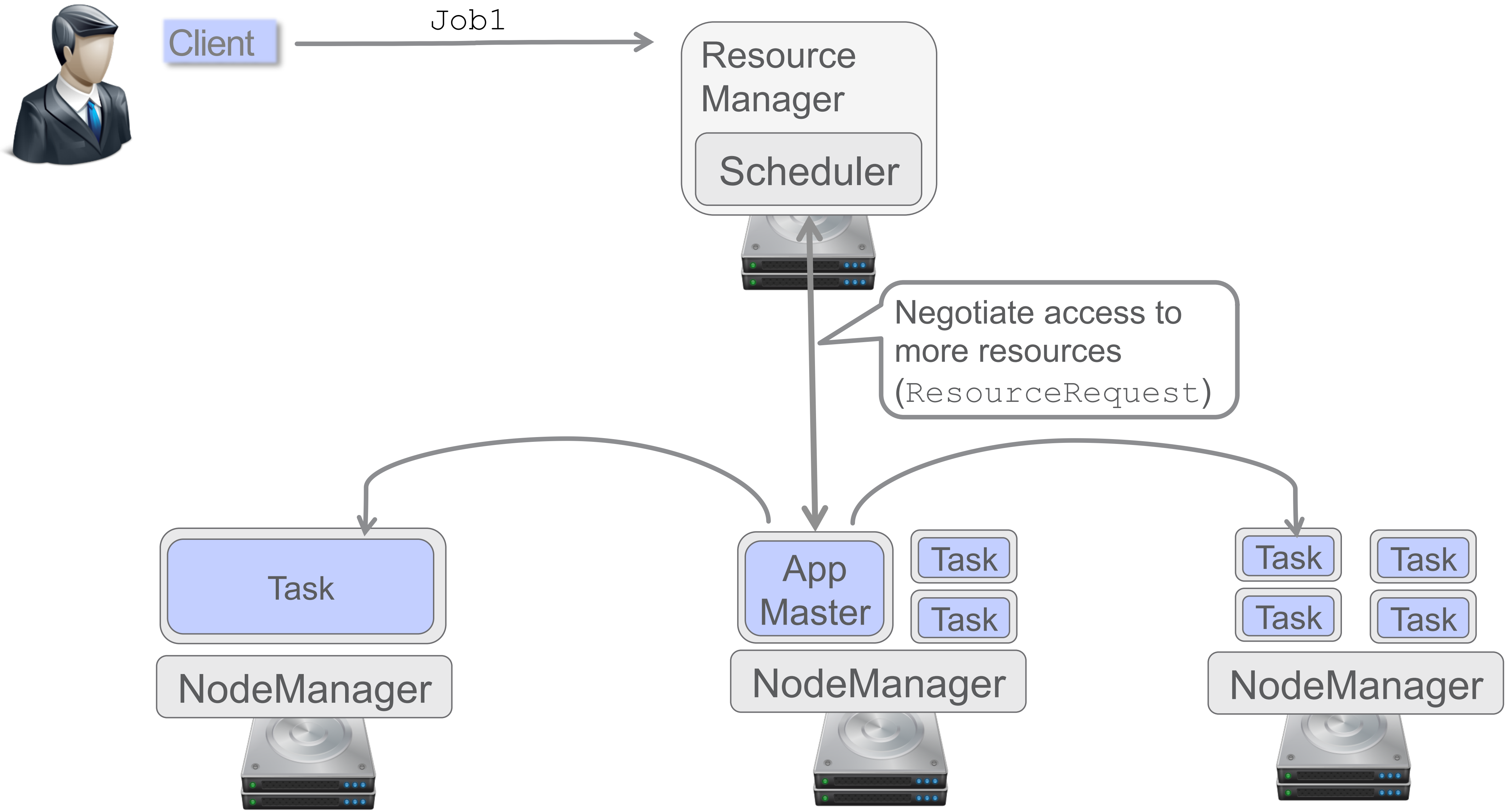
Hardware

YARN (or Hadoop 2.x)

A new architecture for Hadoop

- Decouples resource management from programming model
- (MapReduce is an "application" running on YARN)

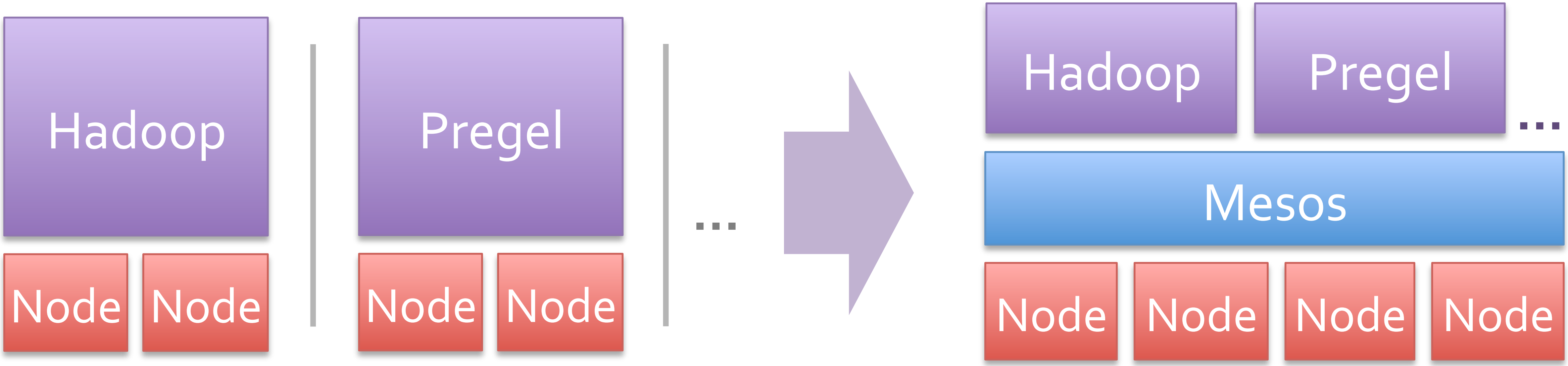
YARN (Hadoop 2) Architecture



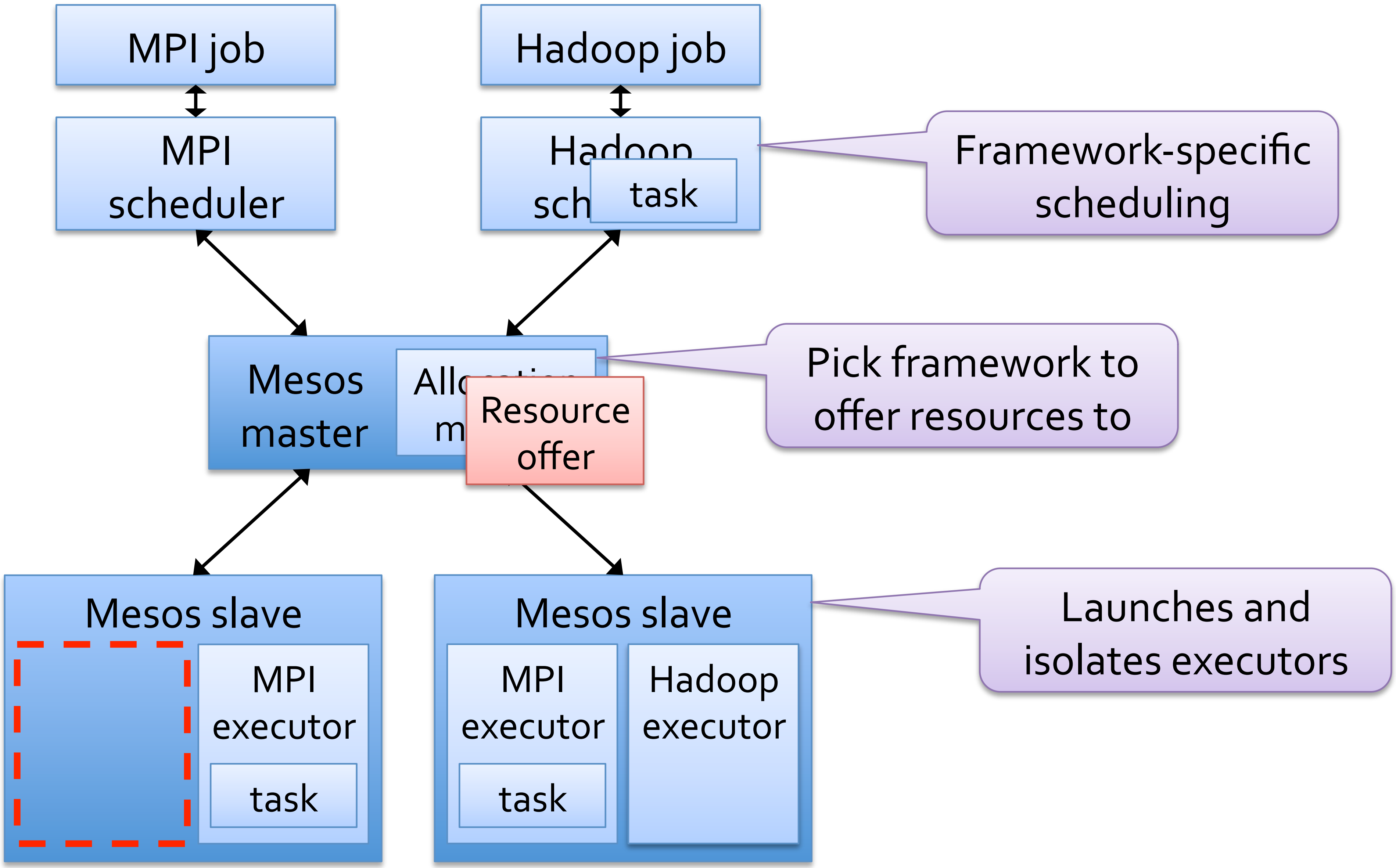
Negotiate access to more resources (ResourceRequest)

Mesos

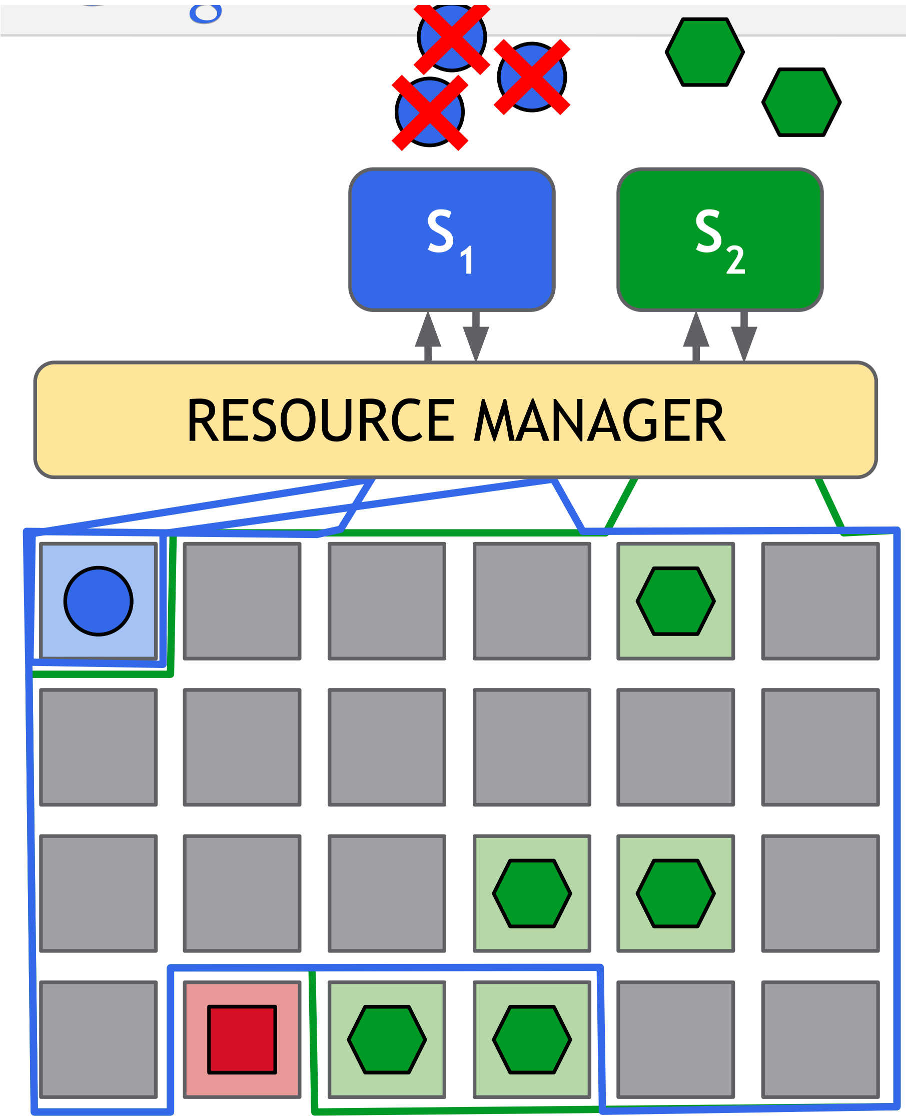
Layer between framework and hardware



Mesos


MPI job
Hadoop job
MPI scheduler
Hadoop sch
task
Framework-specific scheduling
Mesos master
Allo
m
Resource offer
Pick framework to offer resources to
Mesos slave
MPI executor
task
Mesos slave
MPI executor
task
Hadoop executor
Launches and isolates executors

Omega critique of Mesos



1. Green receives offer of all available resources.

2. Blue's task finishes.

3. Blue receives tiny offer.

4. Blue cannot use it.

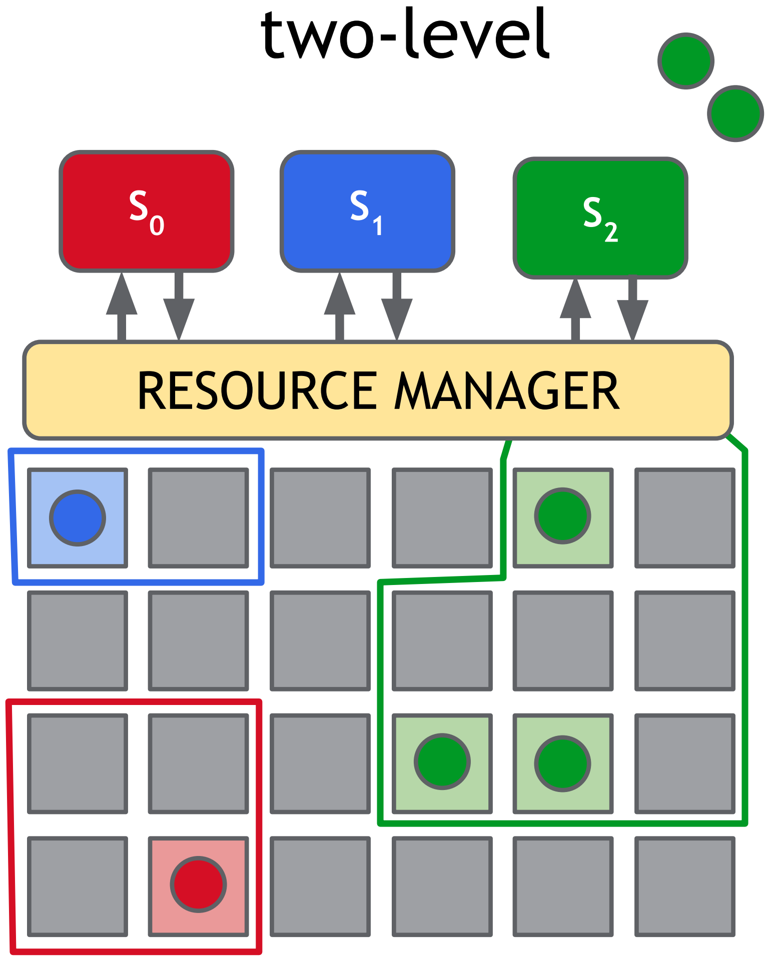
[repeat many times]

5. Green finishes scheduling.

6. Blue receives large offer.

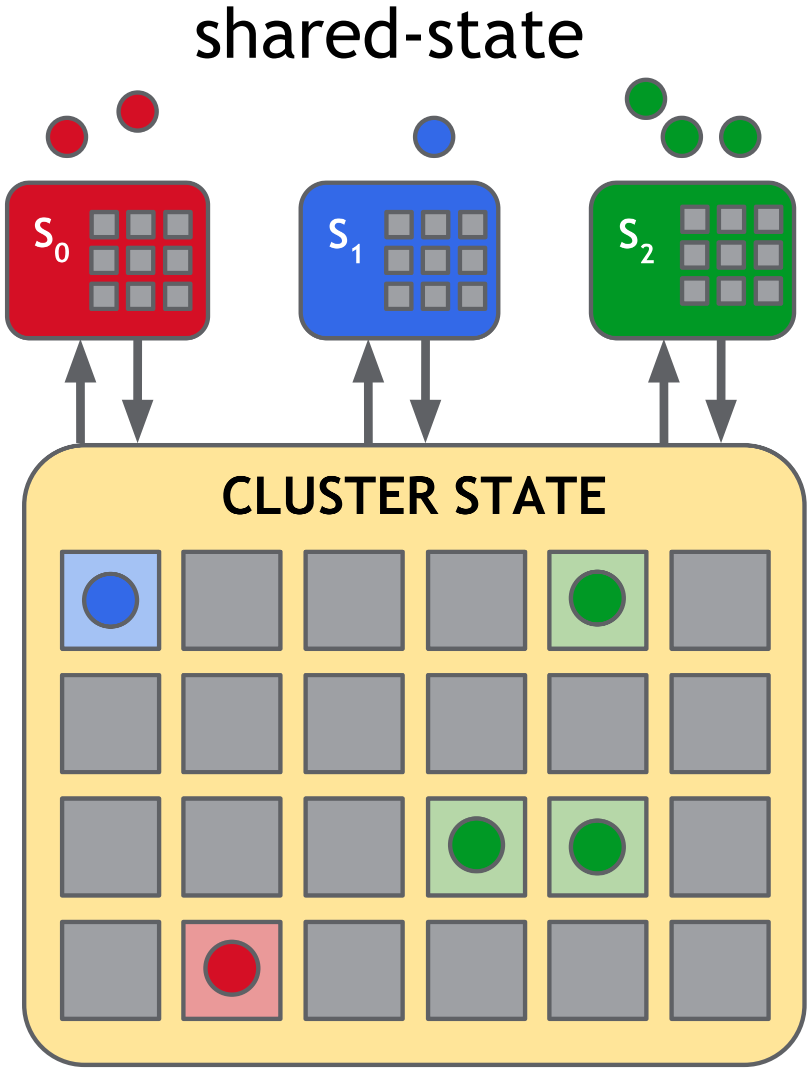
By now, it has given up.

Omega

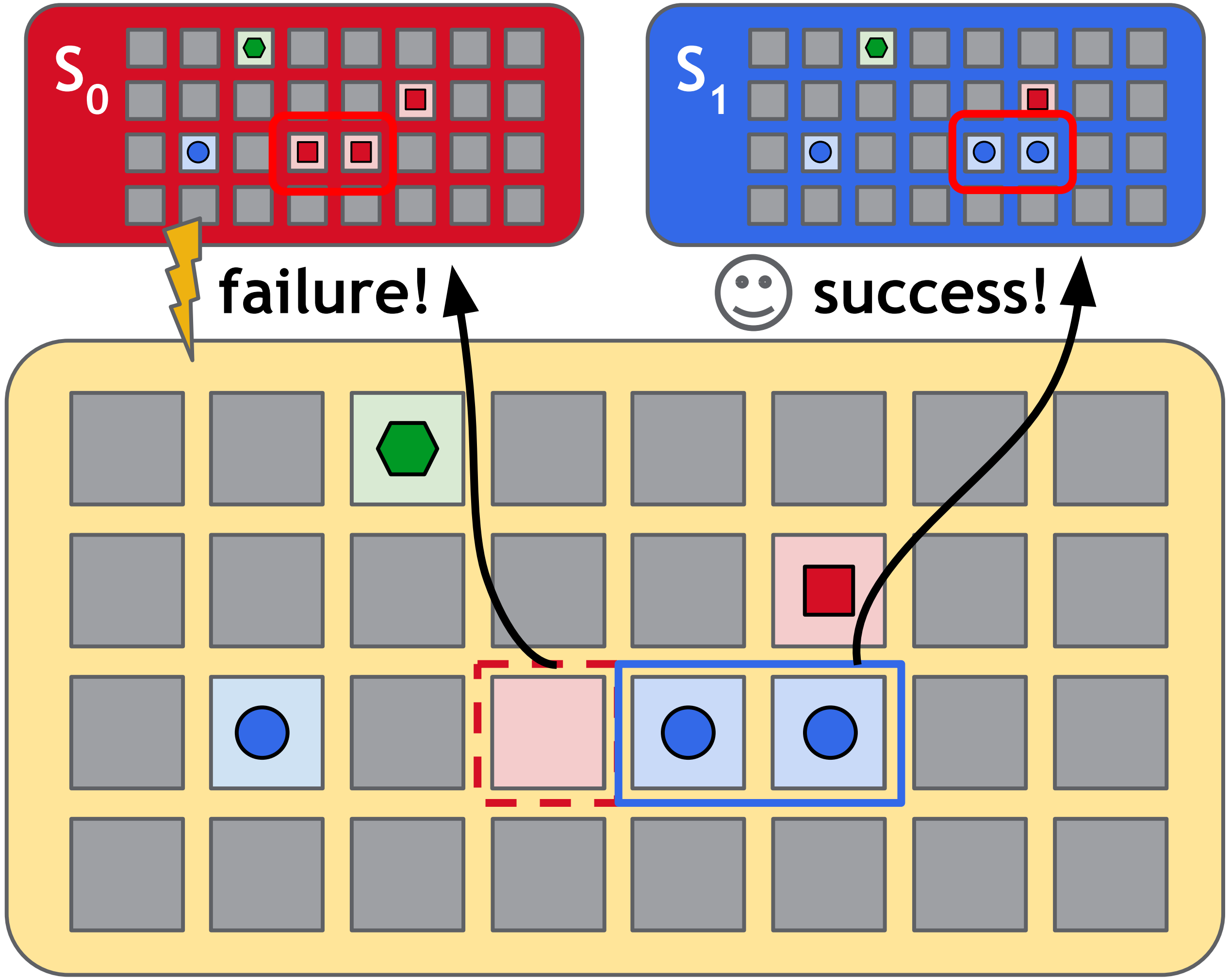


- hoarding
- information hiding

e.g. UCB Mesos [NSDI 2011]



Omega



Why does this matter?

Flexibility, Performance and Availability

- Multiple Programming Models
- Central components do less → *scale better*
- Easier High-Availability (e.g., RM vs AM)

Anything else you can think?

Anything else you can think?

Maintenance, Upgrade, and Experimentation

- Run with multiple framework versions (at one time)
- Trying out a new ideas is as is as launching a job

Summarizing (technical considerations)

ALL improve scalability, flexibility, availability

YARN focus on “jobs”, untrusted apps, no faith sharing, global invariants are king, does not show cluster state

Mesos good upgrade, high-availability story, works for services,

expose cluster state

Omega even more focus on scale, major trust in frameworks, global invariants afterthought, services are key, expose cluster state

Summarizing (non technical considerations)

YARN	open-source, GA (since yesterday), used in-production
Mesos	open-source, tested in real companies
Omega	prototype, Google-internal