Scheduling Work on Large Heterogeneous Compute Clusters.

Scheduling Work on Large Scale Heterogeneous Cloud Compute Clusters

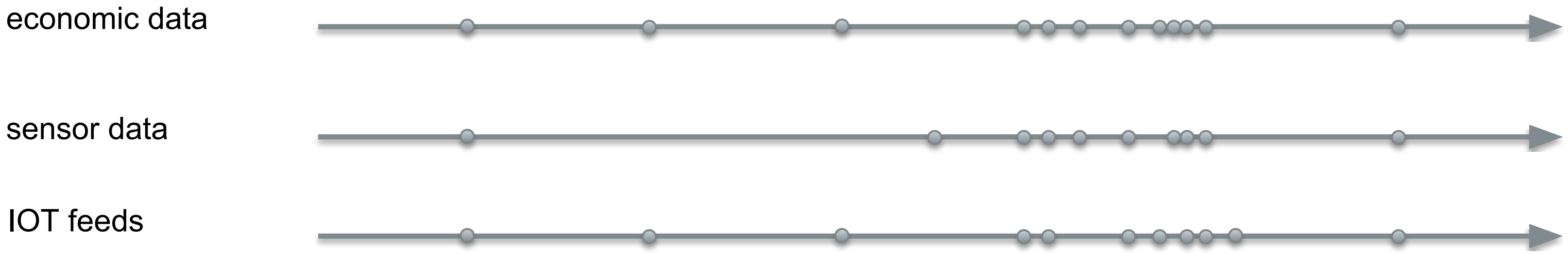
TWO SIGMA

Technology drives our business. We use machine learning, distributed computing and other technologies to find connections in the world’s data.

we use data.

Time Series Data

An ordered sequence of values of a variable

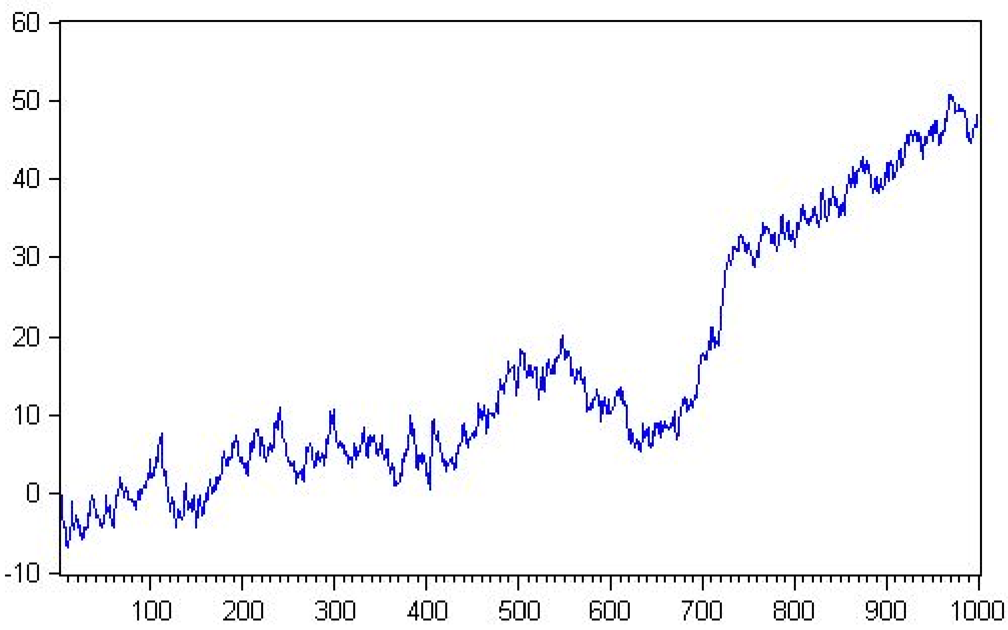


Time Series at Two Sigma

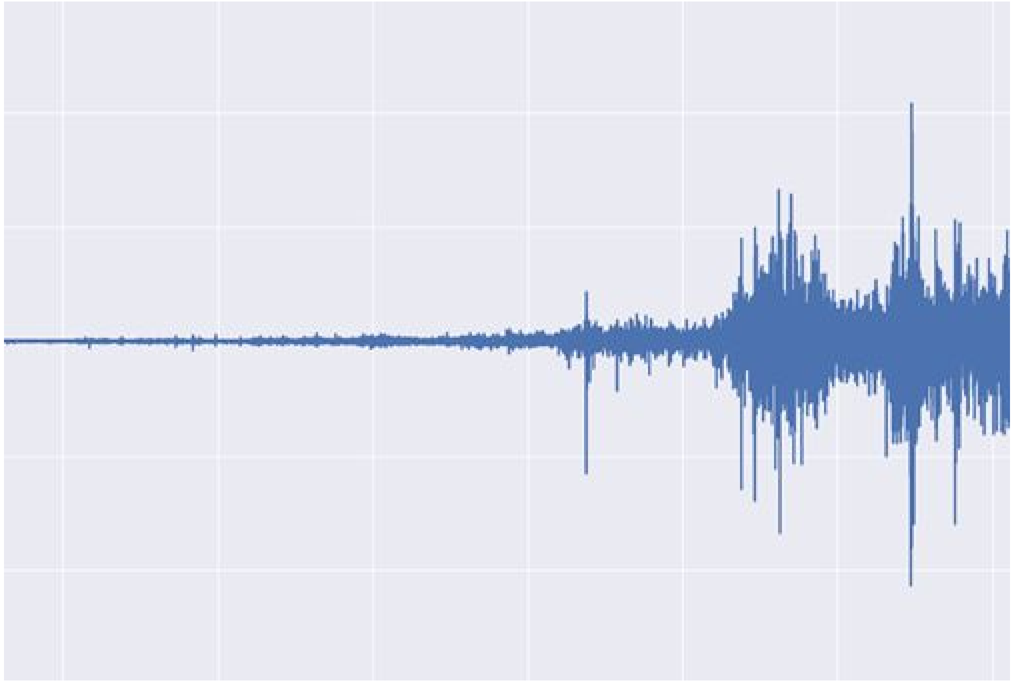
- Millions of Time Series
- Big and Small (1GB – 1PB)
- Narrow (10 columns) and Wide (1MM Columns)
- Evenly and Unevenly Spaced Observations

we find connections.

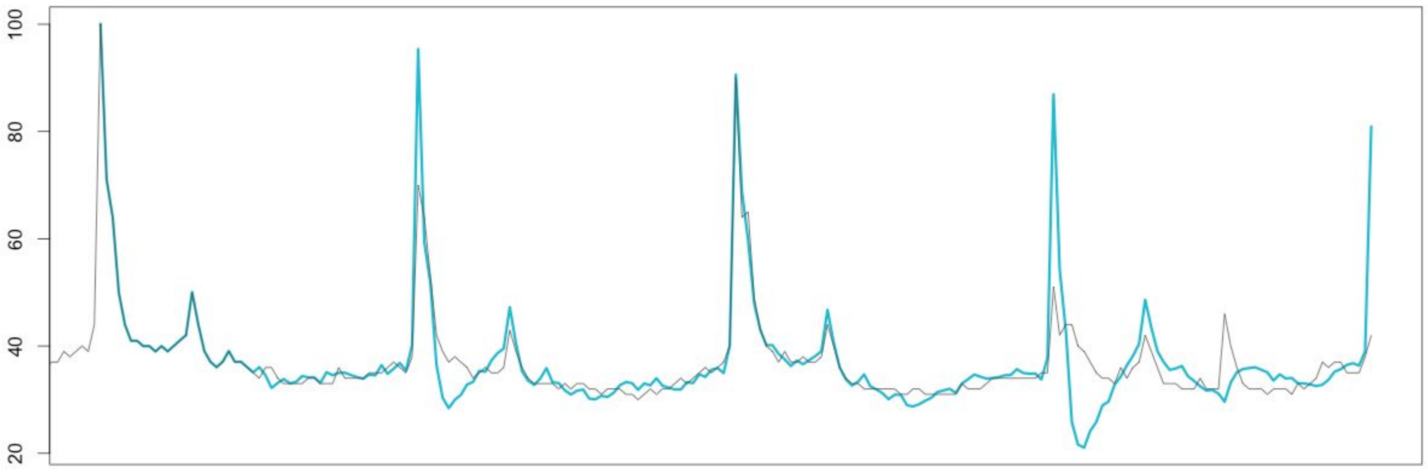
Time Series Analysis



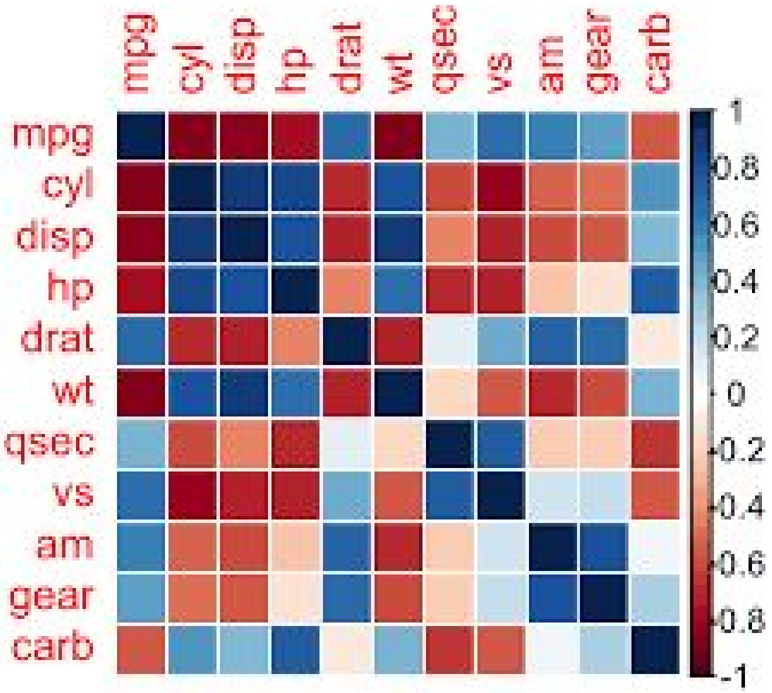
Time Series Analysis



Time Series Analysis



Time Series Analysis: Finding Connections



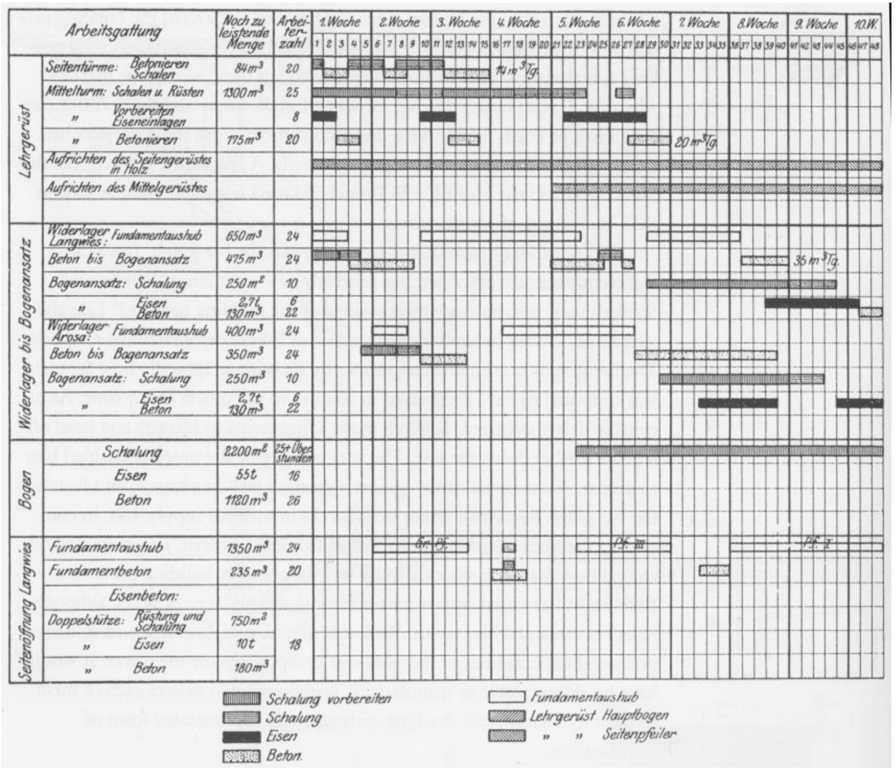
we use machine learning and distributed computing

Analysis at Two Sigma

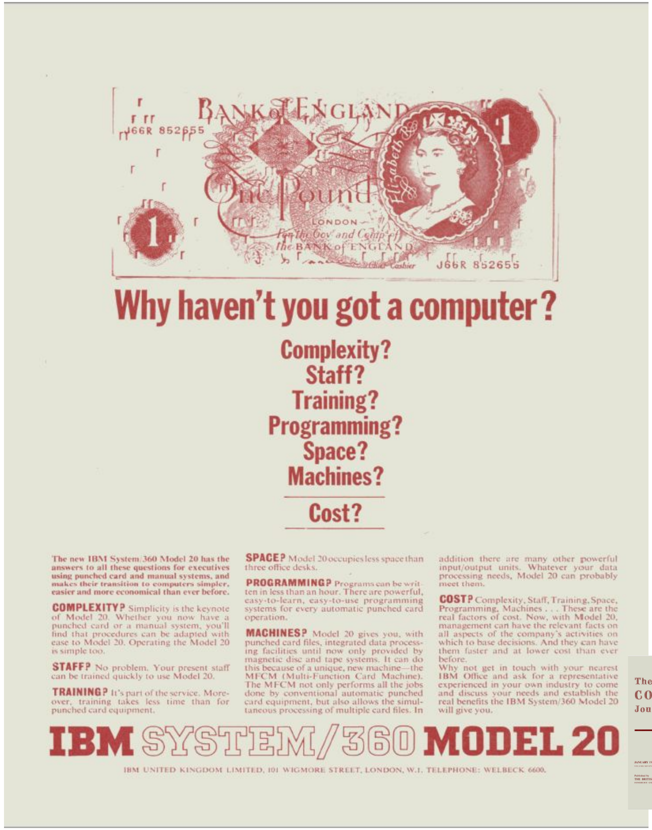
- Hundreds of Users
- Thousands of Computers
- Millions of Jobs
- Hundreds of Applications

Lets start from the beginning.

Early Job Scheduling

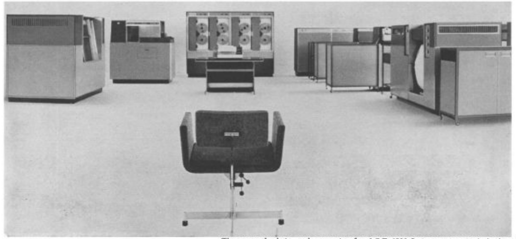


Enter the Computer.



Scheduling Compute.

EXECUTIVE AT WORK!



The operator's chair – and some units of an I.C.T. 1900 Series computer in the background.

Invisible mastermind speeds a mass of different jobs through an I.C.T. 1900 Series computer automatically

(and all at the same time!)

However big the computer's work load is, 'Executive' (a special I.C.T. master program) will find the quickest, most economic way of getting through it. In doing so it reduces human error to a minimum (and even gives the operator instructions!).

It notes the priorities and requirements of all the programs, and allocates time on the central processor and peripherals accordingly. It transforms control between peripherals, and between them and the central processor. And it does this so that no part of the machine needed by any of the programs is ever idle. One job may have top priority, but work on the less important ones can still go on.

* * *

I.C.T. has a distinct lead with master programs like 'Executive' and with multi-programming techniques. This lead was first established with the Atlas and Orion computers.

4 IMPORTANT POINTS FOR BUSINESSMEN

1. Why is the 1900 Series a sound long-term investment?

Whatever configuration you have, you can add more peripheral equipment to it as your needs expand. If the time comes when your work requires more peripherals than the central processor can cope with—you change it easily and quickly for the next largest in the series, *and go on using the same peripherals*. (The feature that makes it possible for *all* the peripherals to work with *all* the processors is called Standard Interface.)

2. Is getting started with the 1900 Series easy?

Yes—largely because programming is simple. There is a wide range of programming languages to choose from and the large I.C.T. library of commercial and scientific sub-routines is available to you. And, of course, with the 1900 Series—a program written for the smallest processor will work on all the others.

3. What does it cost?

Typical systems range from a price of about £40,000 to £750,000 or more. Attractive rental terms can be arranged.

4. When can I have one delivered?

Deliveries start within a year.

The new I.C.T. 1900 Series gives businessmen and scientists exactly what they want in a computer series—now, and for years ahead. It is fully competitive technically. It is I.C.T.'s firm belief that no comparable series has a higher productivity per £ invested. And it can be seen now.

I.C.T. are eager to answer any questions you may want to ask about the 1900 Series. To show you why in your case this new British computer series is a sound long-term investment. Why not get your secretary to fix an appointment, or ask for a brochure with full details and specifications?

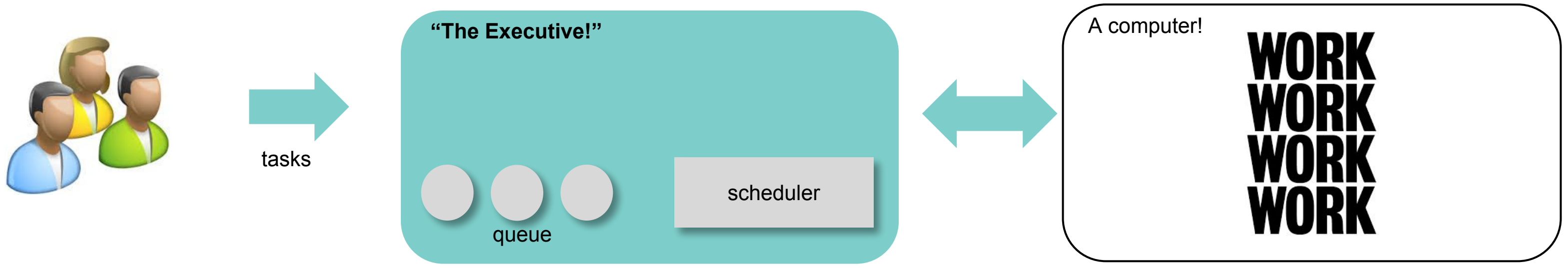
I·C·T

International Computers and Tabulators Limited

I.C.T. SALES HEADQUARTERS, BRIDGE HOUSE, PUTNEY BRIDGE, LONDON SW6. TEL: RENOWN 3322

First Generation Schedulers.

First Generation Schedulers



The Goals of a Scheduler

Be Fair

Maximize Throughput

Minimize Response Time

Be Predictable

Minimize Overhead

Maximize Resource Utilization

Enforce Priorities

Scale Horizontally

Second Generation Schedulers.

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

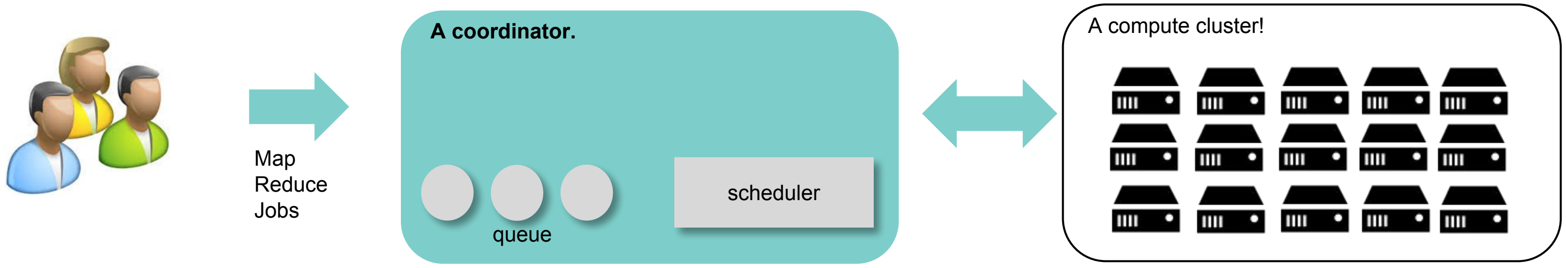
Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

Second Generation Schedulers.



Scheduling Work on Large Heterogeneous Cloud Compute Clusters

Third Generation Schedulers.

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
University of California, Berkeley

Thursday 30th September, 2010, 12:57

Abstract

We present Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. Sharing improves cluster utilization and avoids per-framework data replication. Mesos shares resources in a fine-grained manner, allowing frameworks to achieve data locality by taking turns reading data stored on each machine. To support the sophisticated schedulers of today's frameworks, Mesos introduces a distributed two-level scheduling mechanism called resource offers. Mesos decides *how many* resources to offer each framework, while frameworks decide *which* resources to accept and which computations to run on them. Our results show that Mesos can achieve near-optimal data locality when sharing the cluster among diverse frameworks, can scale to 50,000 (emulated) nodes, and is resilient to failures.

1 Introduction

Clusters of commodity servers have become a major computing platform, powering both large Internet services and a growing number of data-intensive scientific applications. Driven by these applications, researchers and practitioners have been developing a diverse array of cluster computing frameworks to simplify programming the cluster. Prominent examples include MapReduce [23], Dryad [30], MapReduce Online [22] (which supports streaming jobs), Pregel [34] (a specialized framework for graph computations), and others [33, 18, 28].

It seems clear that new cluster computing frameworks[1] will continue to emerge, and that no framework will be optimal for all applications. Therefore, organizations will want to run *multiple frameworks in the same cluster*, picking the best one for each application. Sharing a cluster between frameworks improves utilization and allows applications to share access to large datasets that may be too costly to replicate.

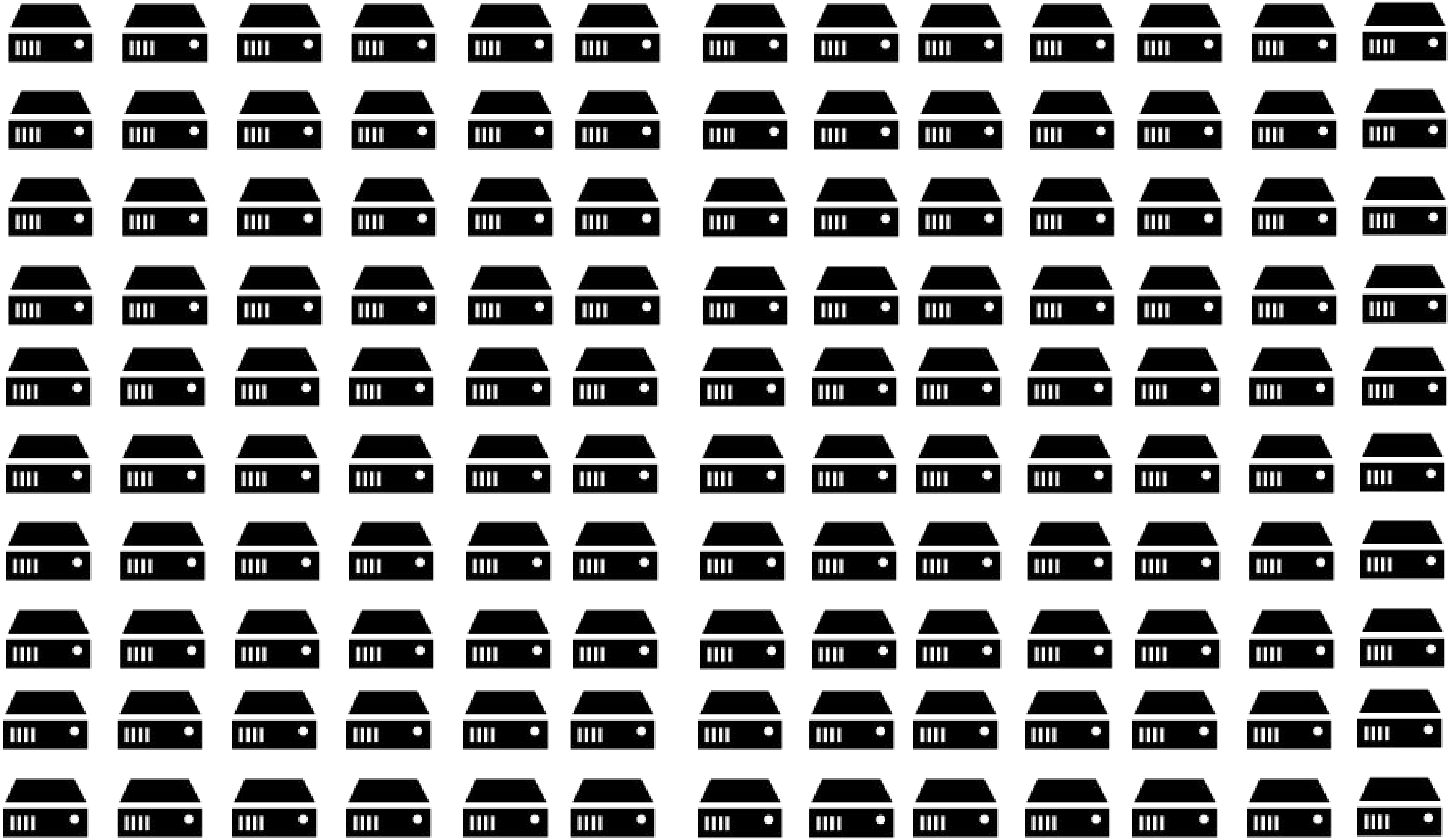
The solutions of choice to share a cluster today are either to statically partition the cluster and run one framework per partition, or allocate a set of VMs to each framework. Unfortunately, these solutions achieve neither high utilization nor efficient data sharing. The main problem is the mismatch between the allocation granularities of these solutions and of existing frameworks. Many frameworks, such as Hadoop and Dryad, employ a fine-grained resource sharing model, where nodes are subdivided into "slots" and jobs are composed of short *tasks* that are matched to slots [31, 44]. The short duration of tasks and the ability to run multiple tasks per node allow jobs to achieve high data locality, as each job will quickly get a chance to run on nodes storing its input data. Short tasks also allow frameworks to achieve high utilization, as jobs can rapidly scale when new nodes become available. Unfortunately, because these frameworks are developed independently, there is no way to perform fine-grained sharing *across* frameworks, making it difficult to share clusters and data efficiently between them.

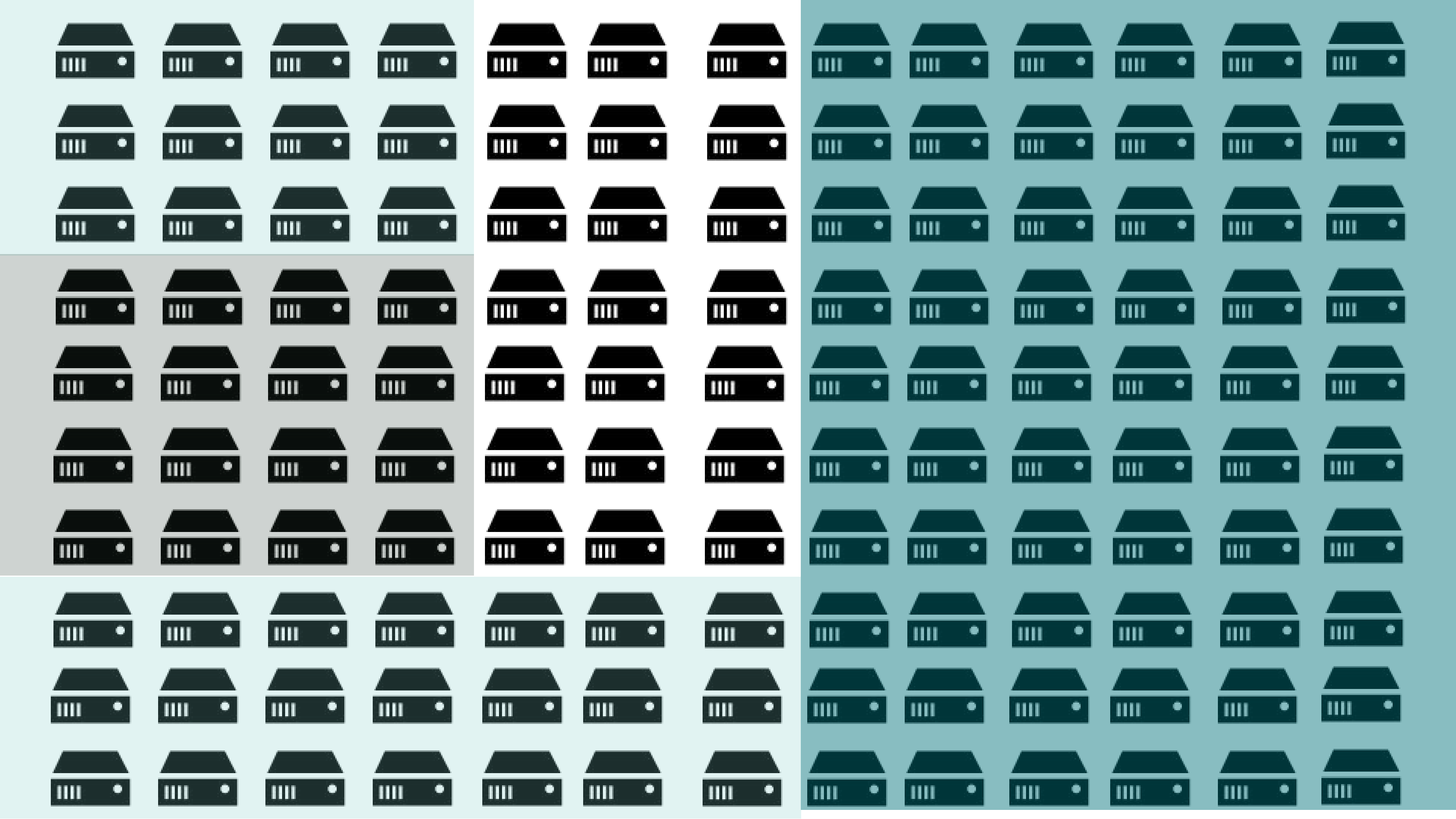
In this paper, we propose Mesos, a thin resource sharing layer that enables fine-grained sharing across diverse cluster computing frameworks, by giving frameworks a common interface for accessing cluster resources.

The main design question that Mesos must address is how to match resources with tasks. This is challenging for several reasons. First, a solution will need to support a wide array of both current and future frameworks, each of which will have different scheduling needs based on its programming model, communication pattern, task dependencies, and data placement. Second, the solution must be highly scalable, as modern clusters contain tens of thousands of nodes and have hundreds of jobs with millions of tasks active at a time. Third, the scheduling system must be fault-tolerant and highly available, as all the applications in the cluster depend on it.

One approach would be for Mesos to implement a centralized scheduler that takes as input framework requirements, resource availability, and organizational policies,

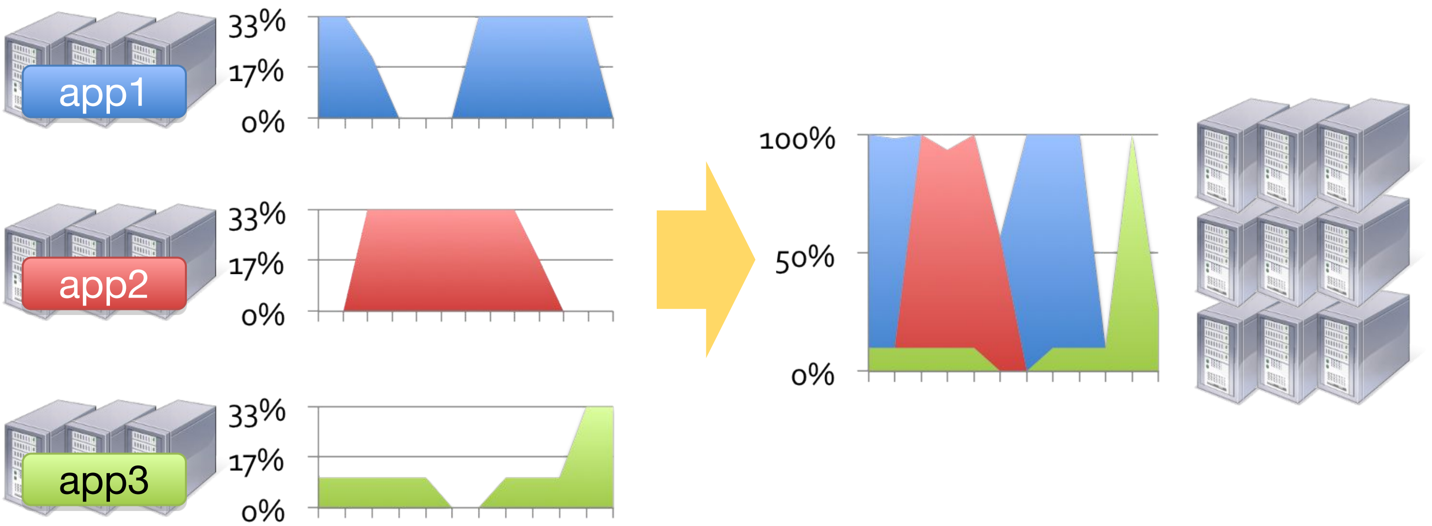
[1] By framework we mean a software system that manages and executes one or more jobs on a cluster.





Static partitioning considered harmful

Static Partitioning vs. Elastic Allocation



Like most problems in computer science, we manage complexity by providing layers of abstraction.

3rd Generation: Two Tiered Schedulers.

Mesos: an abstraction

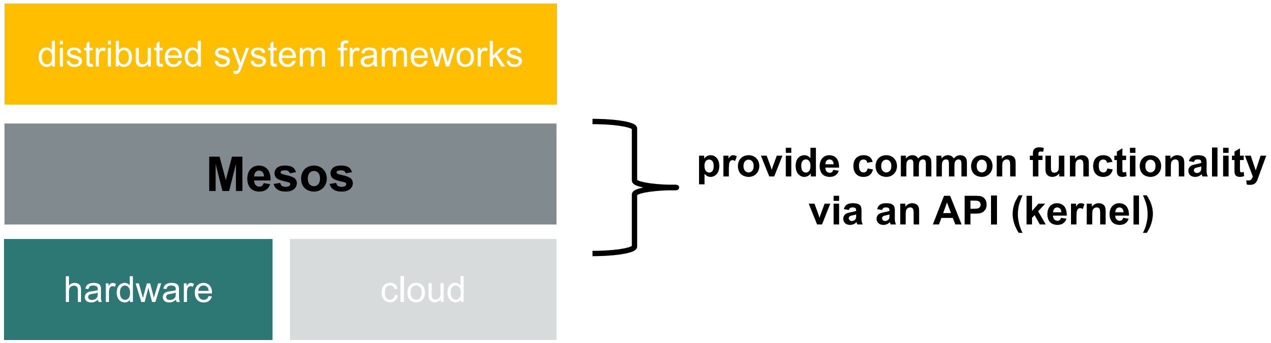
Mesos

build and run
distributed systems
that use *resources* instead
of *machines*

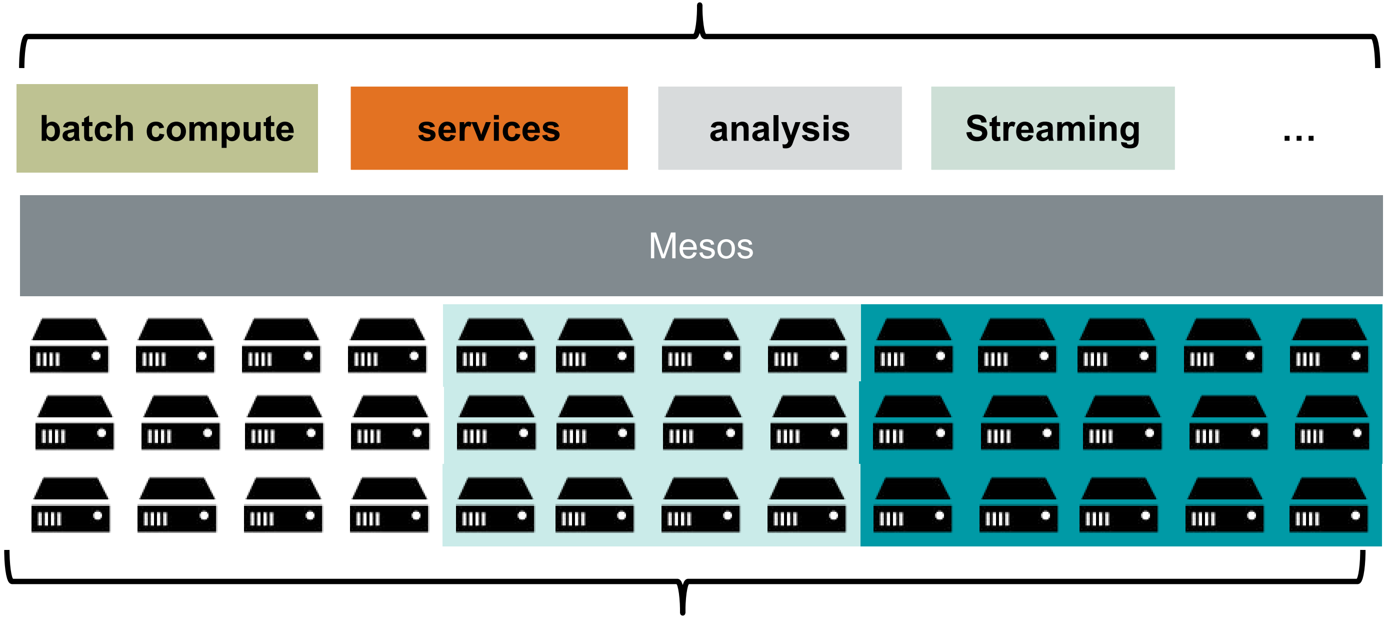
Mesos abstracts compute.

Provide *resources* as
physical machines, or
ts/cloud or even public cloud

A kernel for the datacenter.



support multiple frameworks

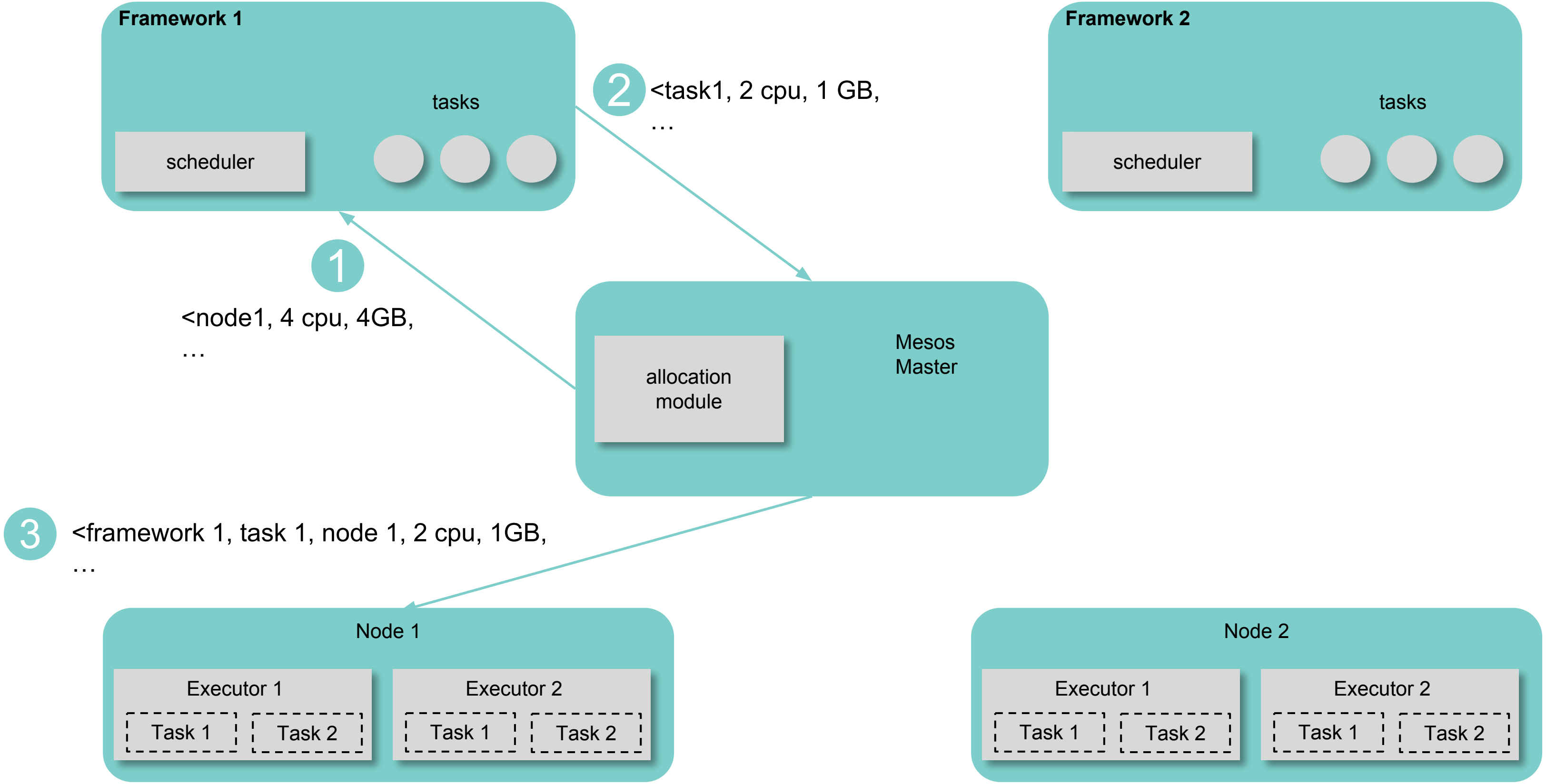


run on a range of compute resources.

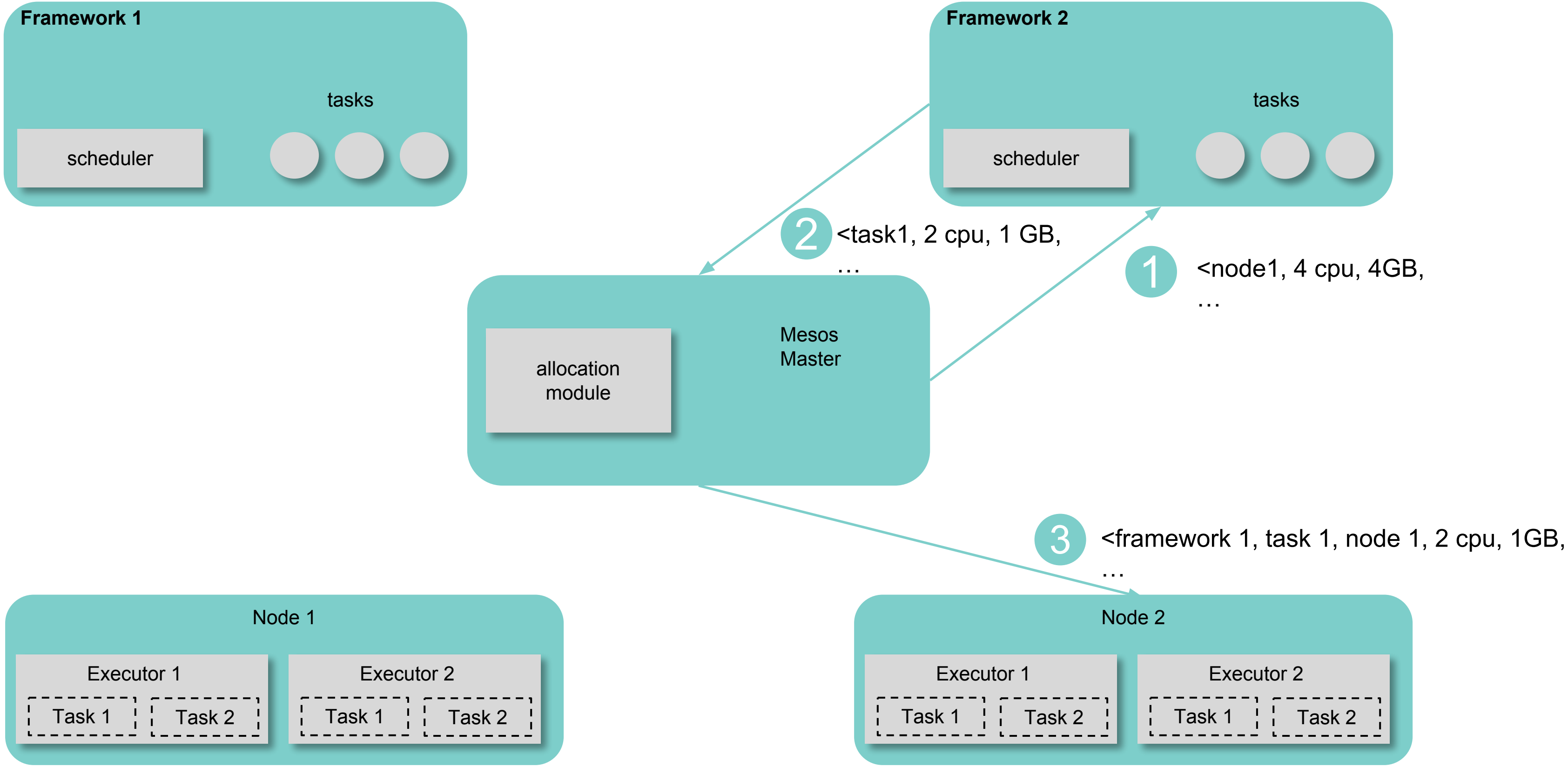
Mesos: Run Everything in Containers

Resource Management and Resource Offers

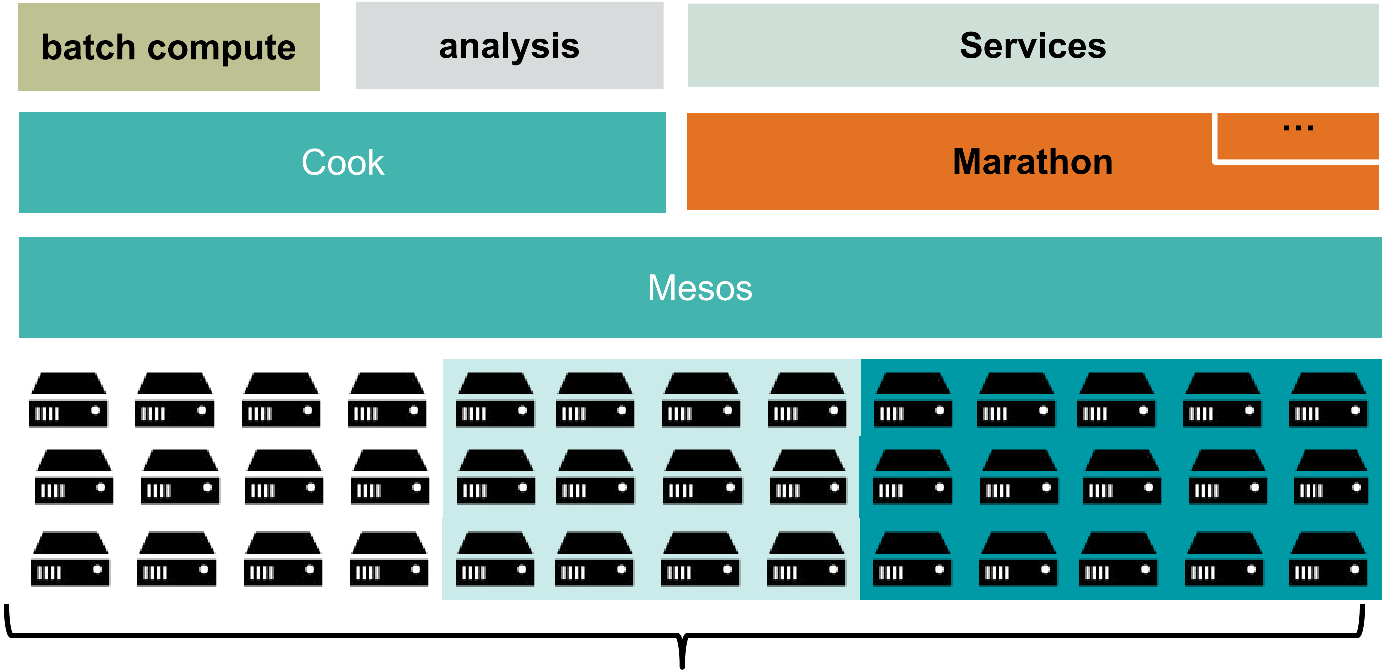
Mesos Scheduling …



Mesos Scheduling ...



The Cloud Operating System



run on a range of cloud compute providers.

Cook: Two Sigma’s open-source resource scheduler for compute clusters, uses pre-emption to achieve low latency and high throughput.

https://www.twosigma.com/insights/cook-a-fair-preemptive-resource-scheduler-for-compute-clusters

Cook: What is Fairness?

Cook: About Pre-emption

Cook: Autoscaling

Challenges and the Future: Discussion