LUND UNIVERSITY

Cluster Resource Management

JONAS DÜRANGO

Driving trends in cluster RM

Clusters of commodity servers becoming the primary computational platform.

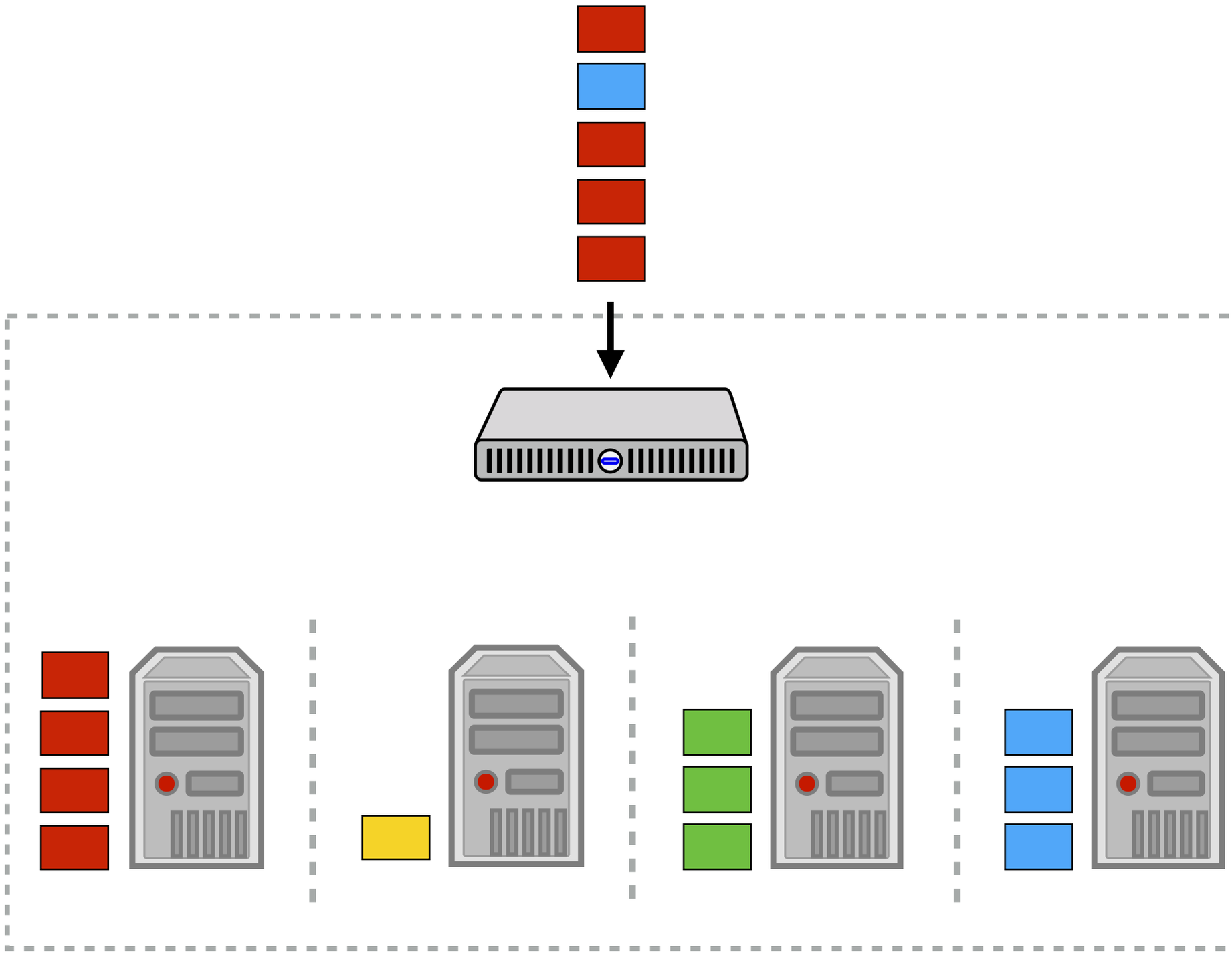
Emergence of frameworks that simplify cluster computing programming (MapReduce, Dryad etc).

No framework to rule them all: each framework has its own merits!

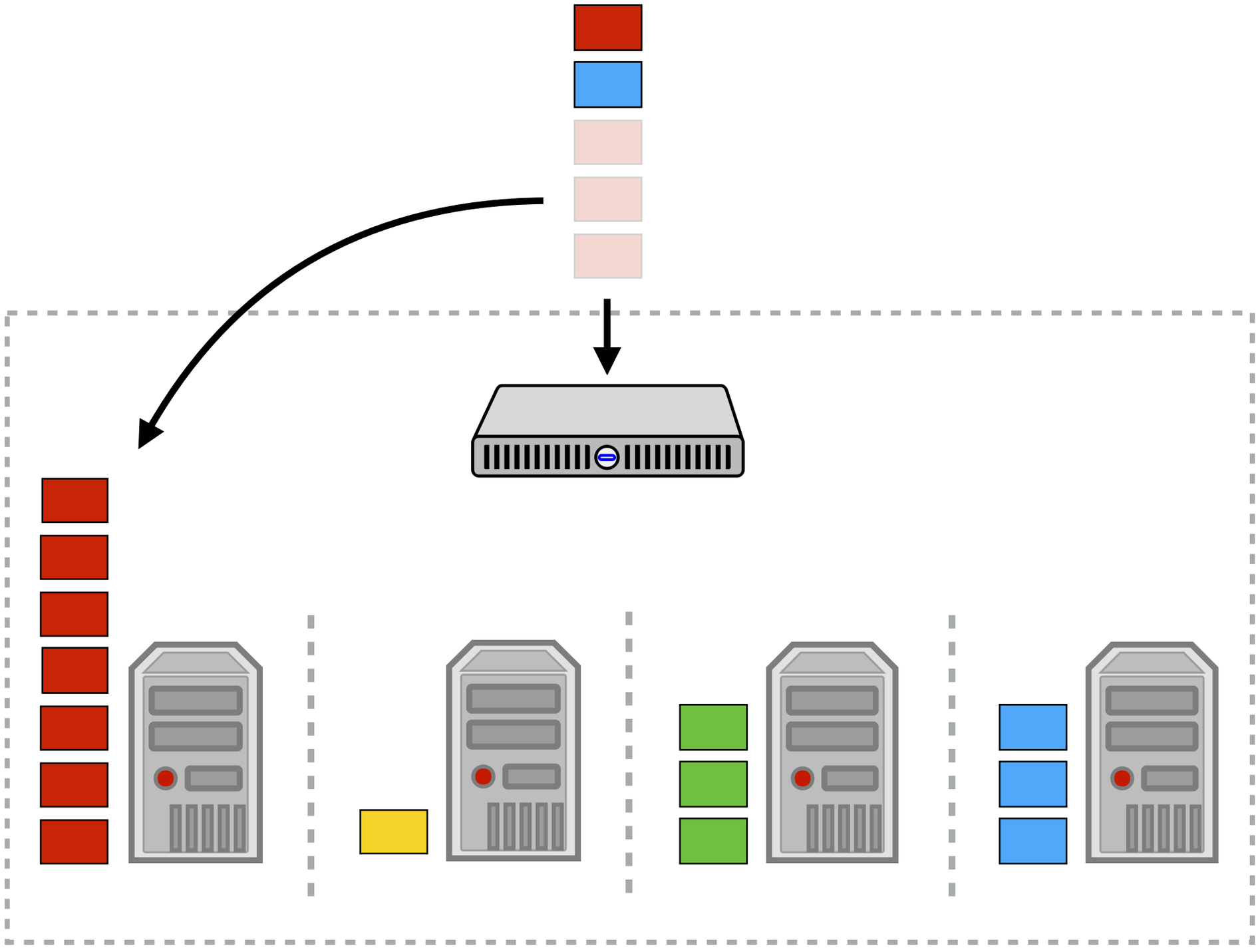
Running multiple frameworks on a cluster increases utilization and decreases unnecessary data replication!

LUND UNIVERSITY

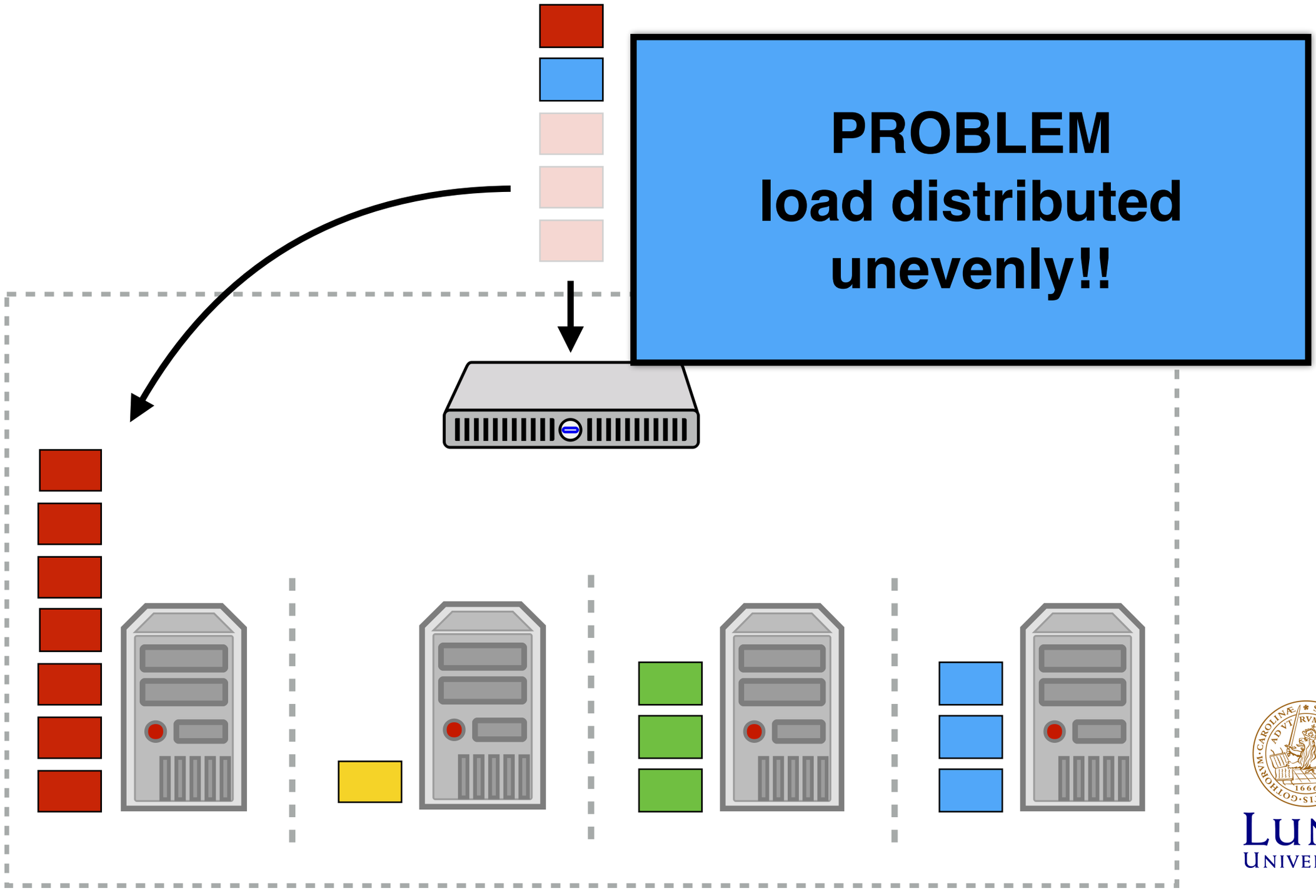
Static cluster partitioning



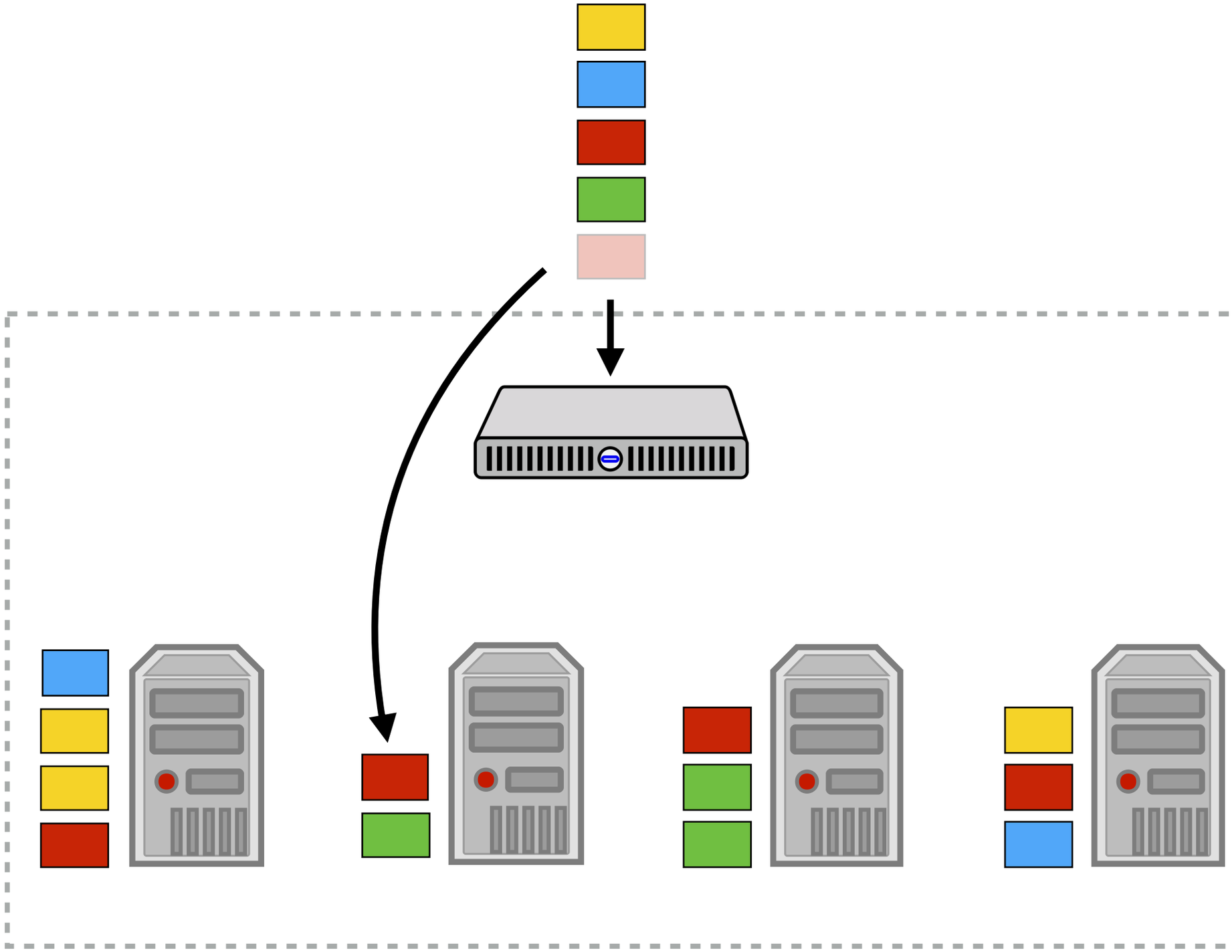
Static cluster partitioning



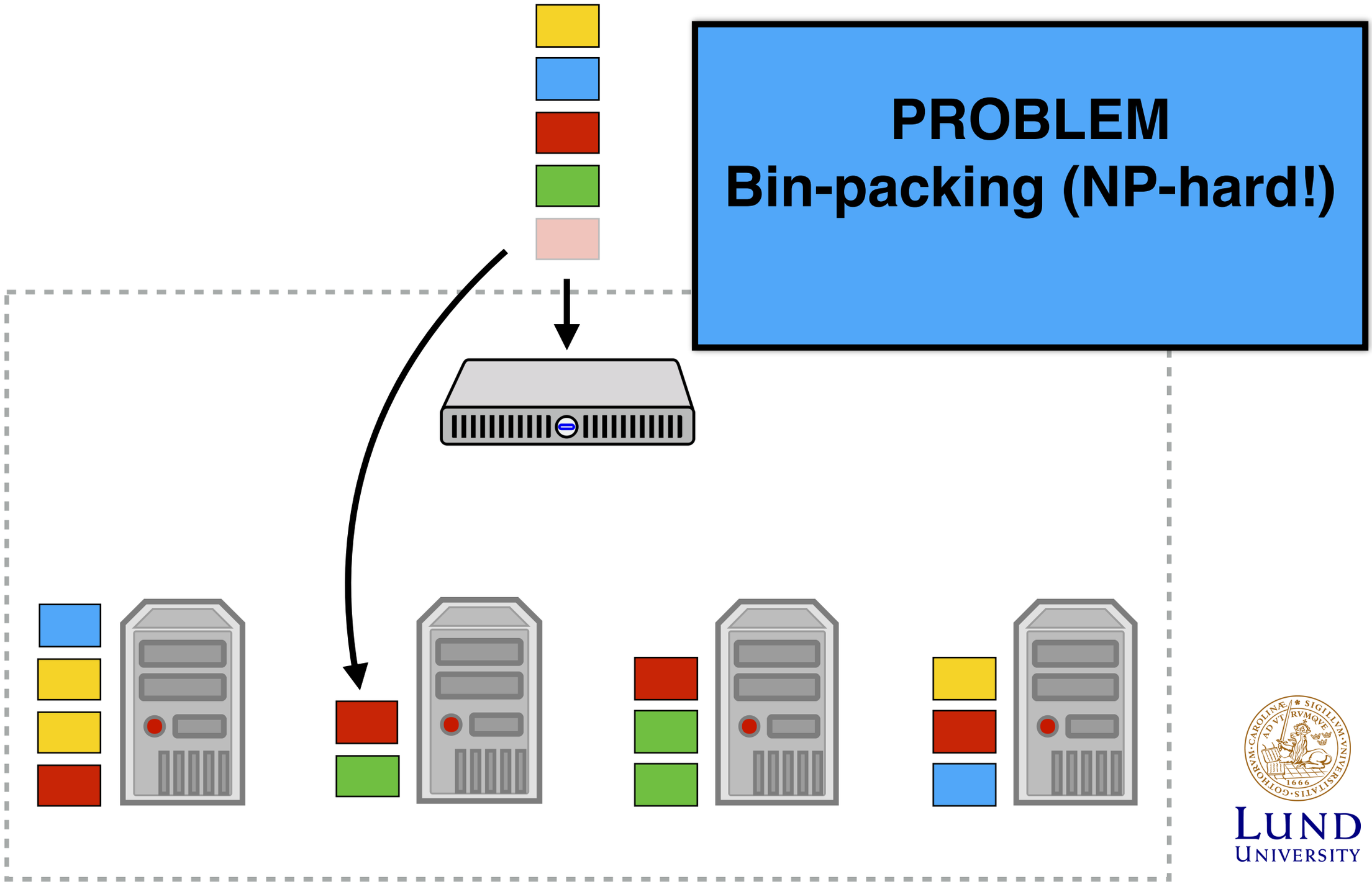
Static cluster partitioning



Alternative?



Alternative?



LUND UNIVERSITY

Requirements and challenges

Modern cluster RMs supporting multi-tenancy face the following…

Requirements:

high utilization, scalability, flexibility, user-supplied placement constraints, rapid decision making, data locality awareness, reliability.

Challenges:

complexity, hardware failures, cluster heterogeneity, workload heterogeneity, workload time-variability.

Not all frameworks are equal

Frameworks have different characteristics. Modern frameworks often are elastic, while some are rigid.

Elastic frameworks (e.g. Hadoop, Spark):

Jobs can start executing as soon as a small set of resources have been allocated. Can scale appropriately as resources are added or removed.

Rigid frameworks (e.g. MPI):

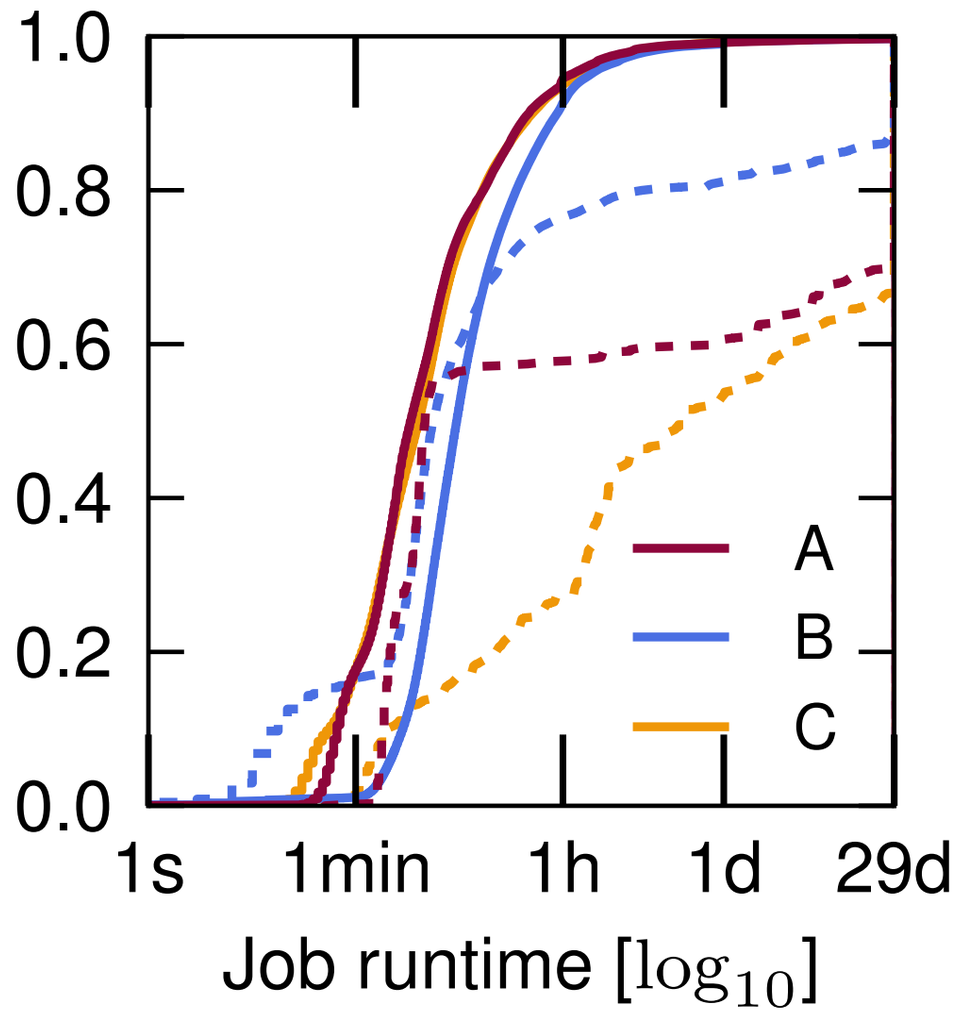
Must acquire a fixed set of resources before jobs can be executed. Not scalable!

Workload characteristics

Workloads are often *very* heterogeneous.

Rough division of jobs:

- **Batch**: ”once and done”, best effort, small!
- **Service**: large, long-running, provides end-user service!

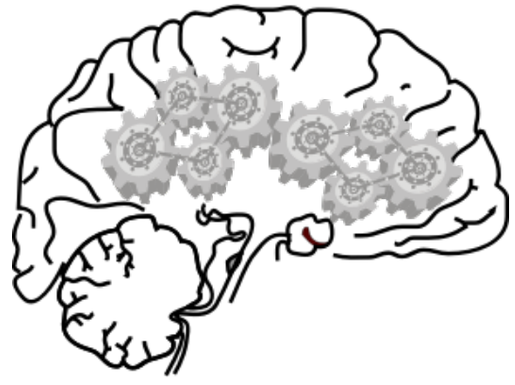


While the majority (>80%) of the jobs are batch jobs, service jobs consume most resources (50-80%).

Implications for scheduling?

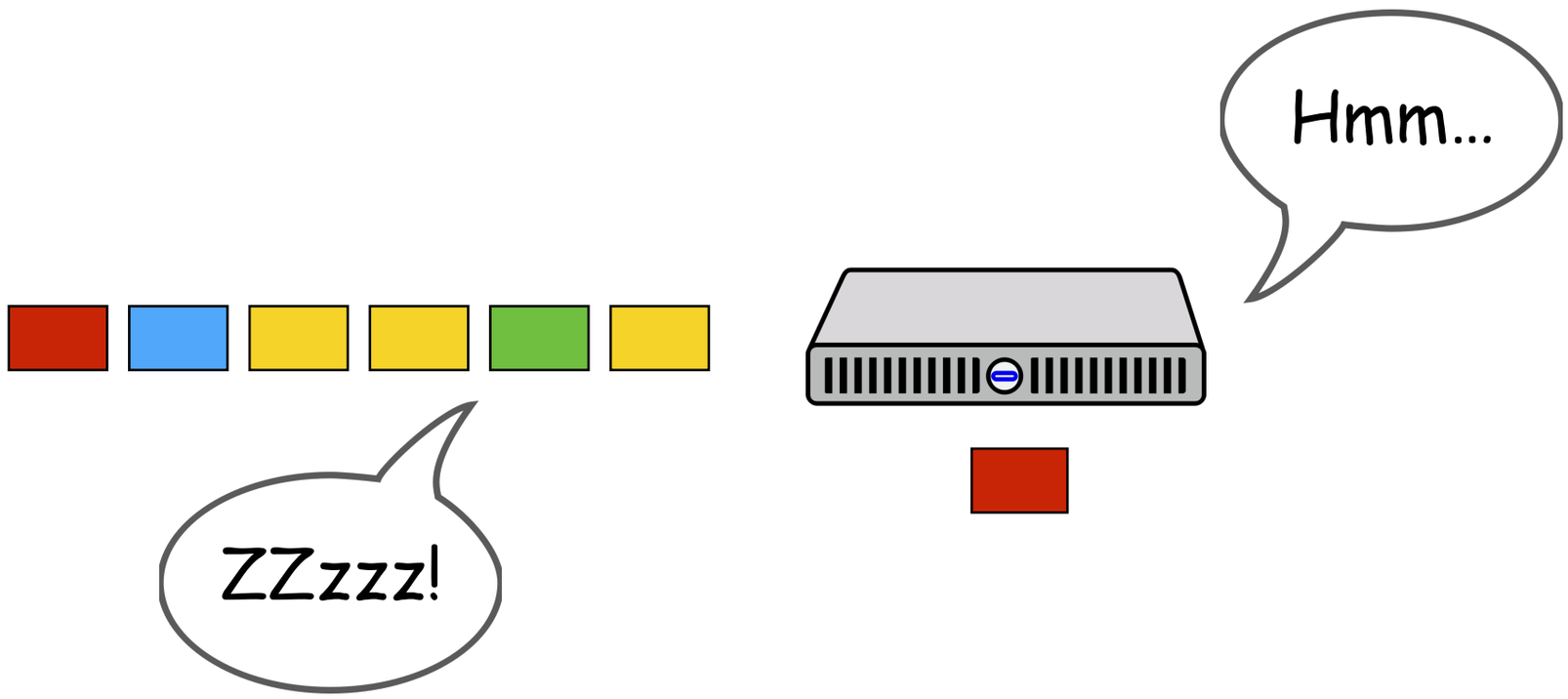
Workload characteristics

It can make sense to spend some time on ”clever” scheduling of long-running service jobs.



For batch jobs, this takes too much time!

Creates ”head of line blocking”!



LUND UNIVERSITY

Some existing solutions

Focus here will be on some existing cluster resource management systems.

Some of them have developed significantly since their publication.

Monolithic
Borg

Two-level
Mesos, YARN

Shared state
Omega

Degree of de-centralization

LUND UNIVERSITY

Monolithic scheduling

Mature technology, been around a long time, common in HPC.

Single resource manager and scheduler.

Some issues:

- Head of line blocking (adressed by multi path scheduling).
- Limited scalability.
- Limited flexibility.

Prominent example: Borg, used internally at Google.

Two-level (dynamic) scheduling

Prominent example: Mesos.

No central scheduler: each framework keeps its own scheduler and keep track of arriving jobs. Instead, framework schedulers are offered resources by a central process and may choose to accept or reject the offer.

The RM dynamically partitions the cluster, deciding the allocation of resources to each framework.

Issues: suitable for short-lived jobs, not for service jobs. Schedulers are unaware of each other. Gang scheduling by hoarding -> potential deadlock!

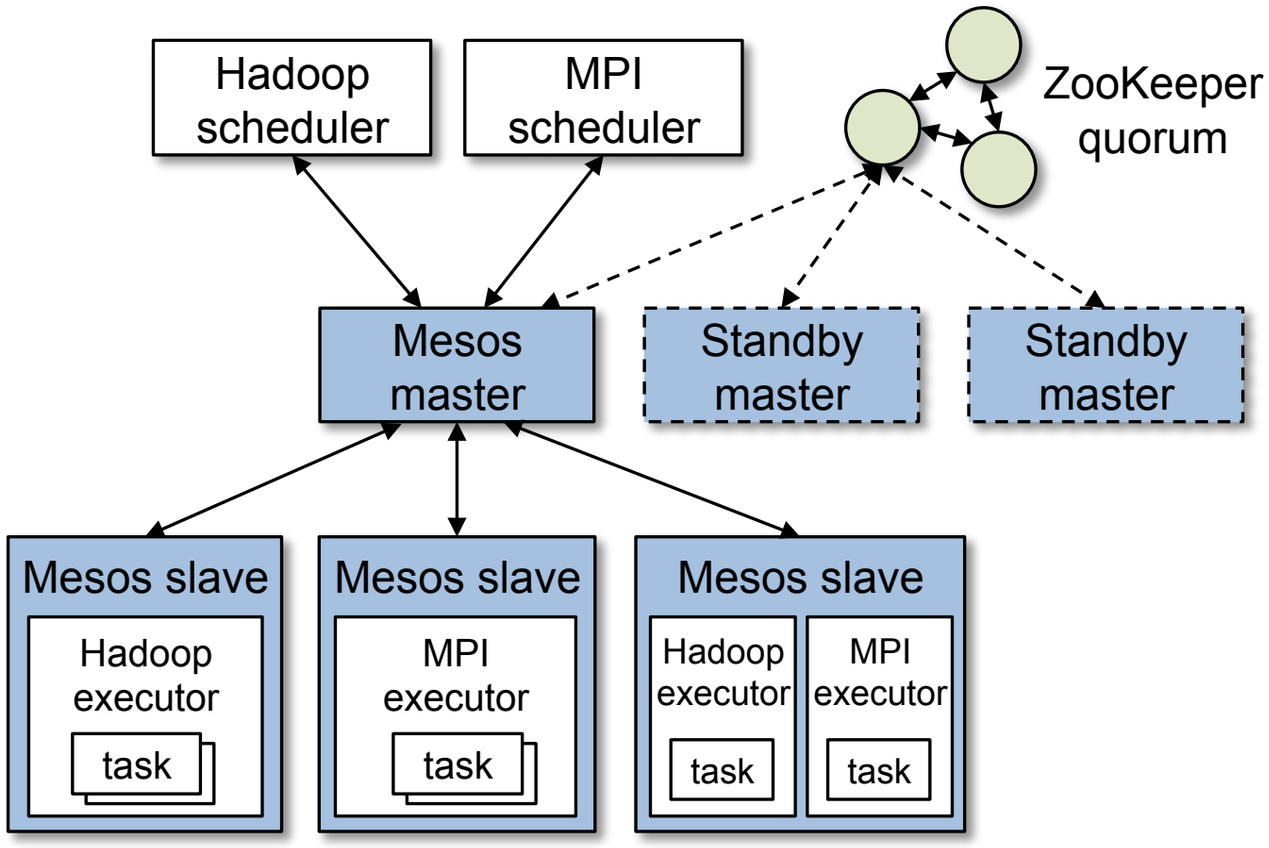
Architecture of Mesos

Add new FW:

- scheduler registers with Mesos master
- executors launched on slave nodes to do the actual work.

Containers for isolation.

Fault tolerance by making the master *soft state*.

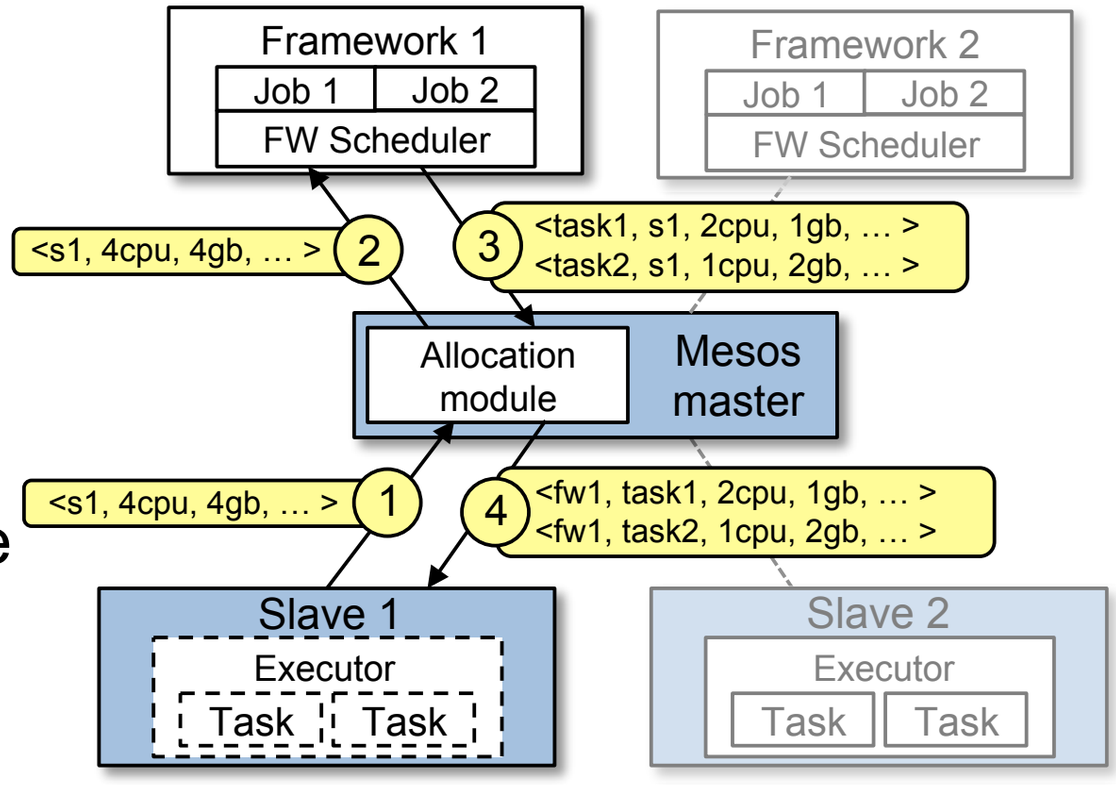


Example: resource offers in Mesos

1. Slave 1 reports that it has free resources.
2. The master offers the newly freed resources to FW1.
3. FW1 responds with two tasks it wants launched, along with resources needed.
4. The master sends the tasks to the slave.

Still free resources may be offered to FW2.

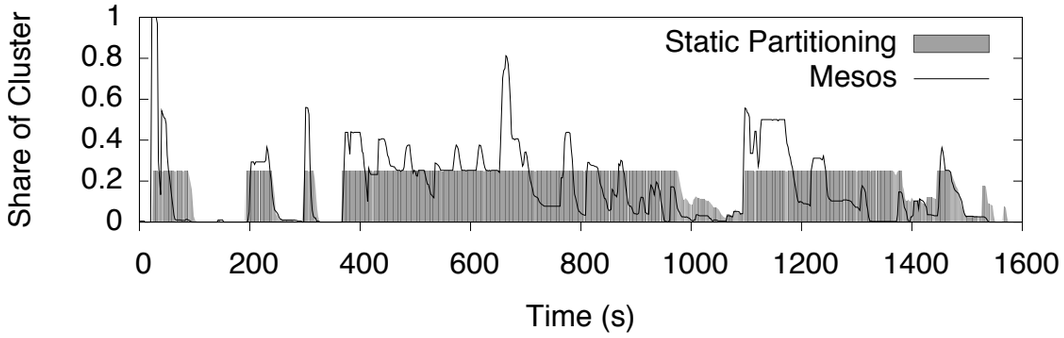
Data locality handled using delay scheduling.



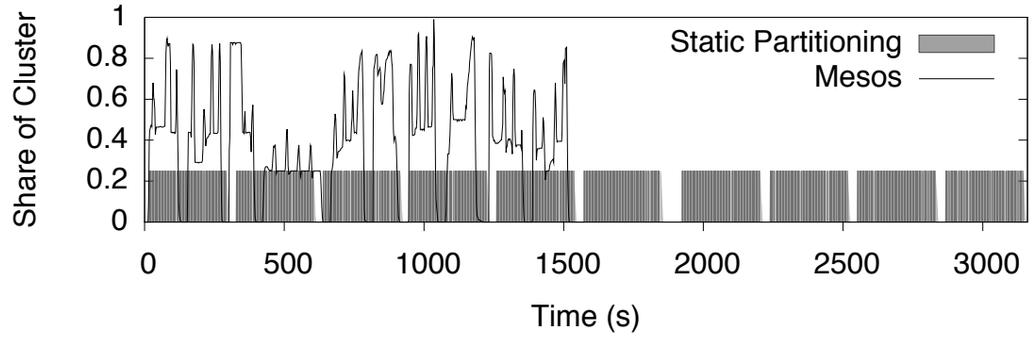
(*) adopted from Hindman et al: *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*

Experimental results with Mesos

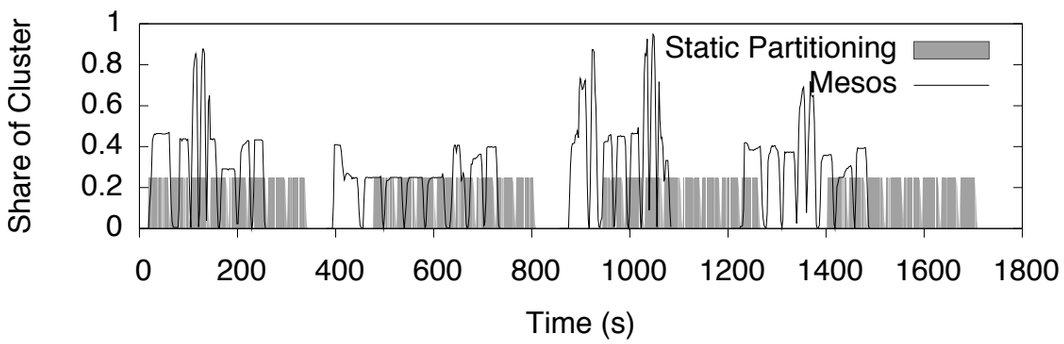
(a) Facebook Hadoop Mix



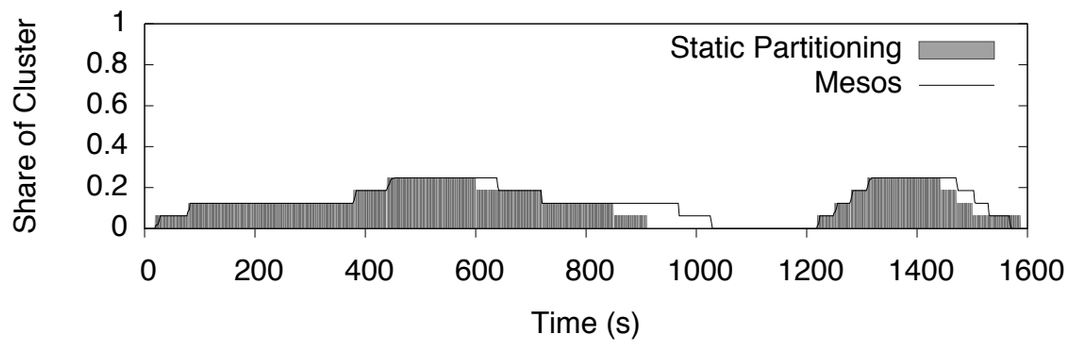
(b) Large Hadoop Mix



(c) Spark



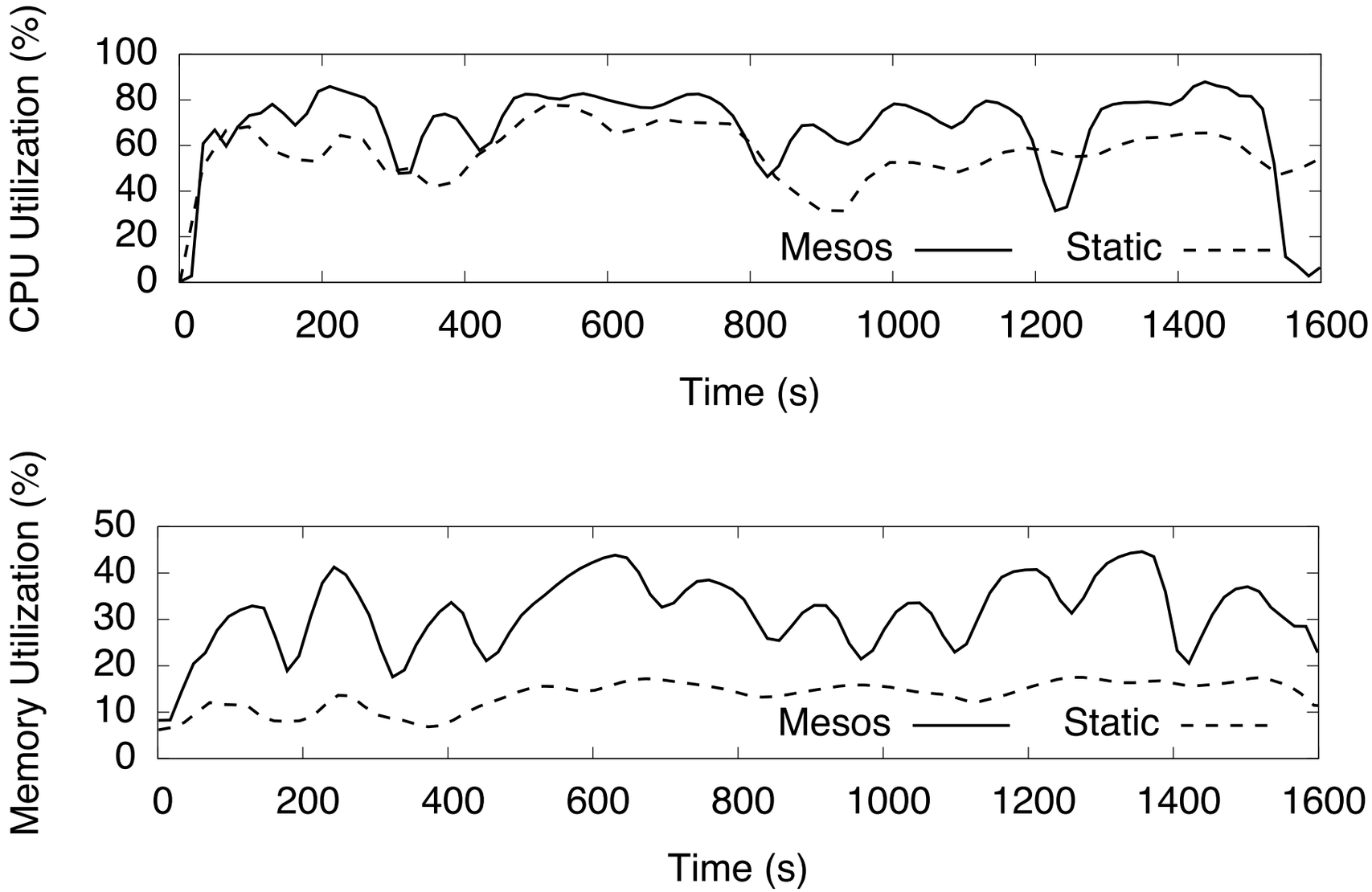
(d) Torque / MPI



(*) adopted from Hindman et al: *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*

LUND UNIVERSITY

Experimental results with Mesos



(*) adopted from Hindman et al: *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*

LUND UNIVERSITY

Shared-state scheduling

Prominent example: Omega.

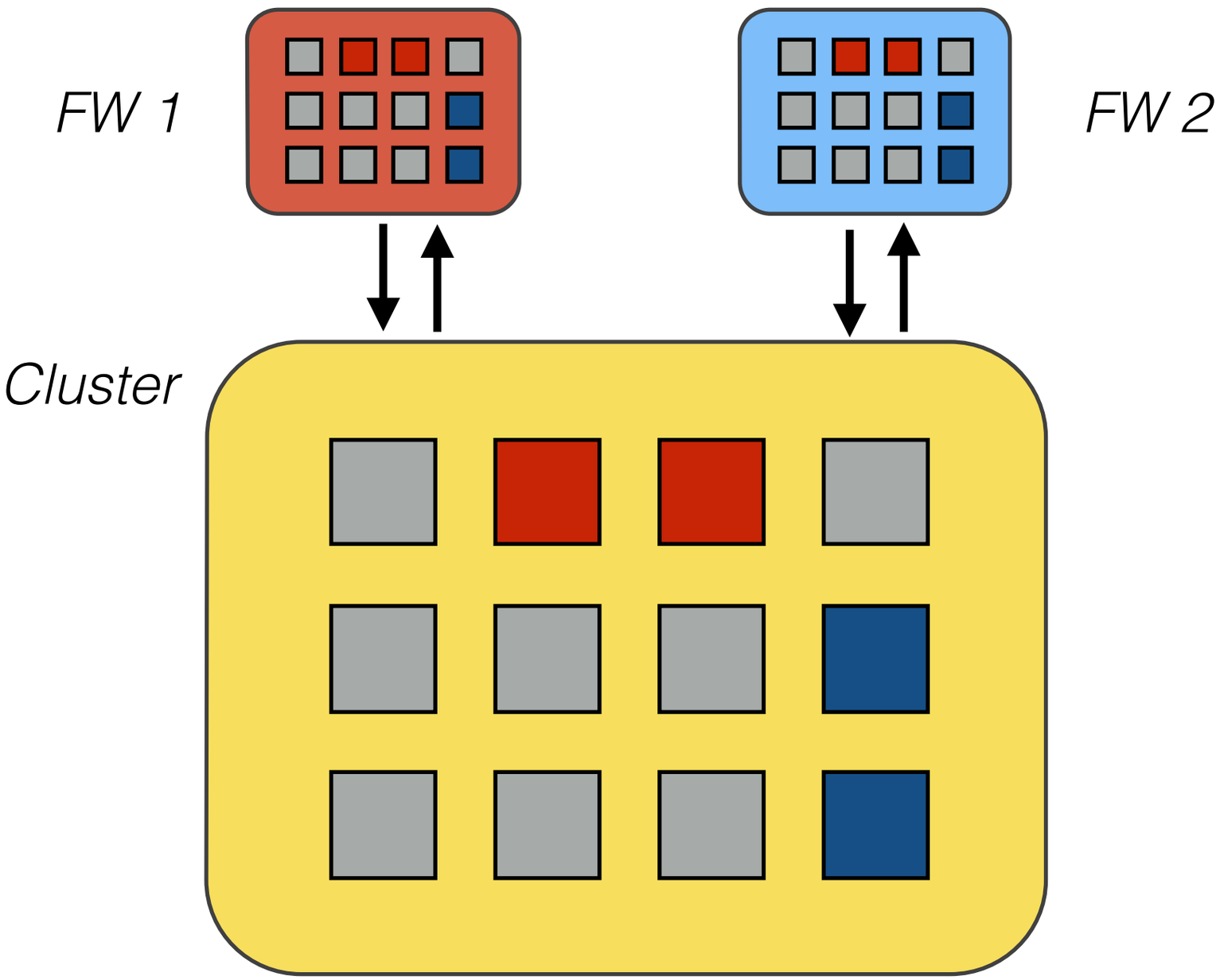
No central scheduler NOR resource manager! Instead, a master copy of the cluster state.

Each framework scheduler has FULL access to the cluster and keeps a local copy of the cluster state.

FWs operate in a FFA manner and can grab any free resources according to their view of the cluster state.

Issues: Inherent risk for conflicts! Solved using optimistic concurrency control. Fault tolerance?

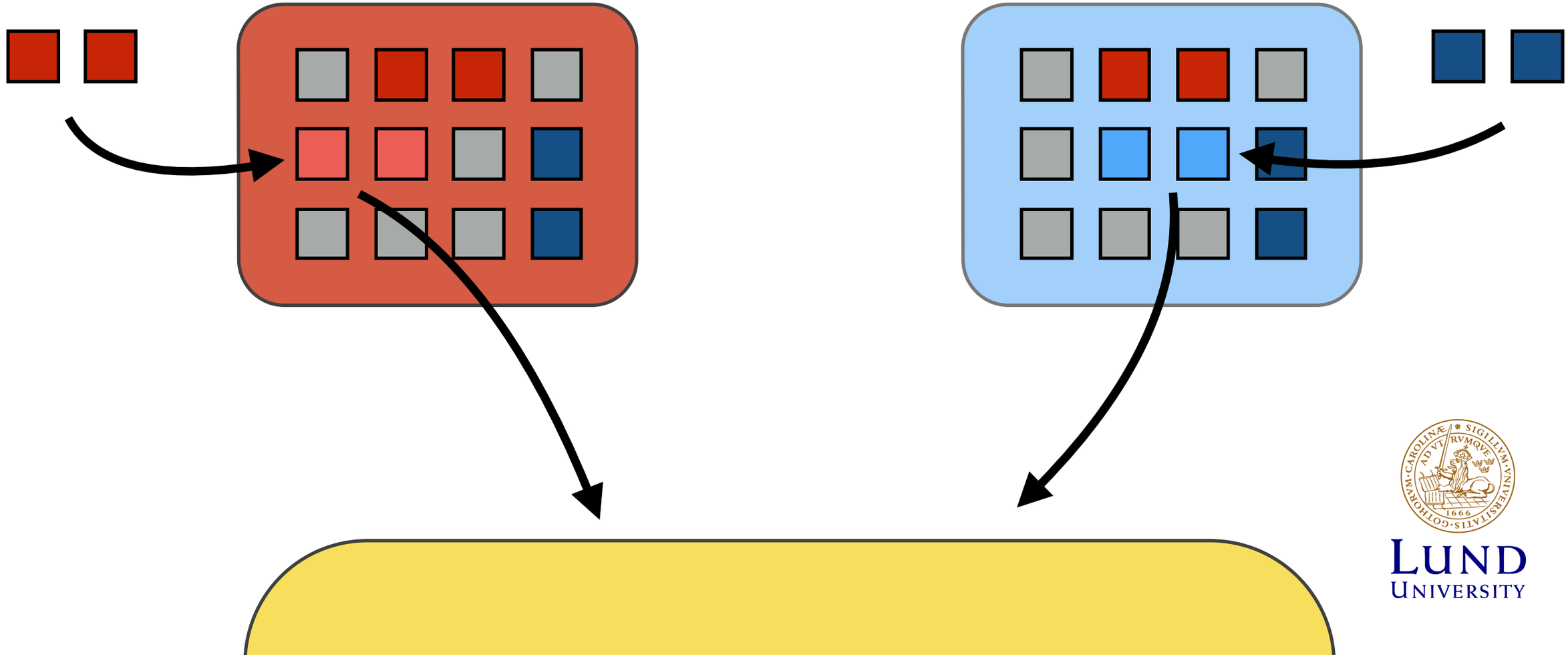
Resource reservation in Omega



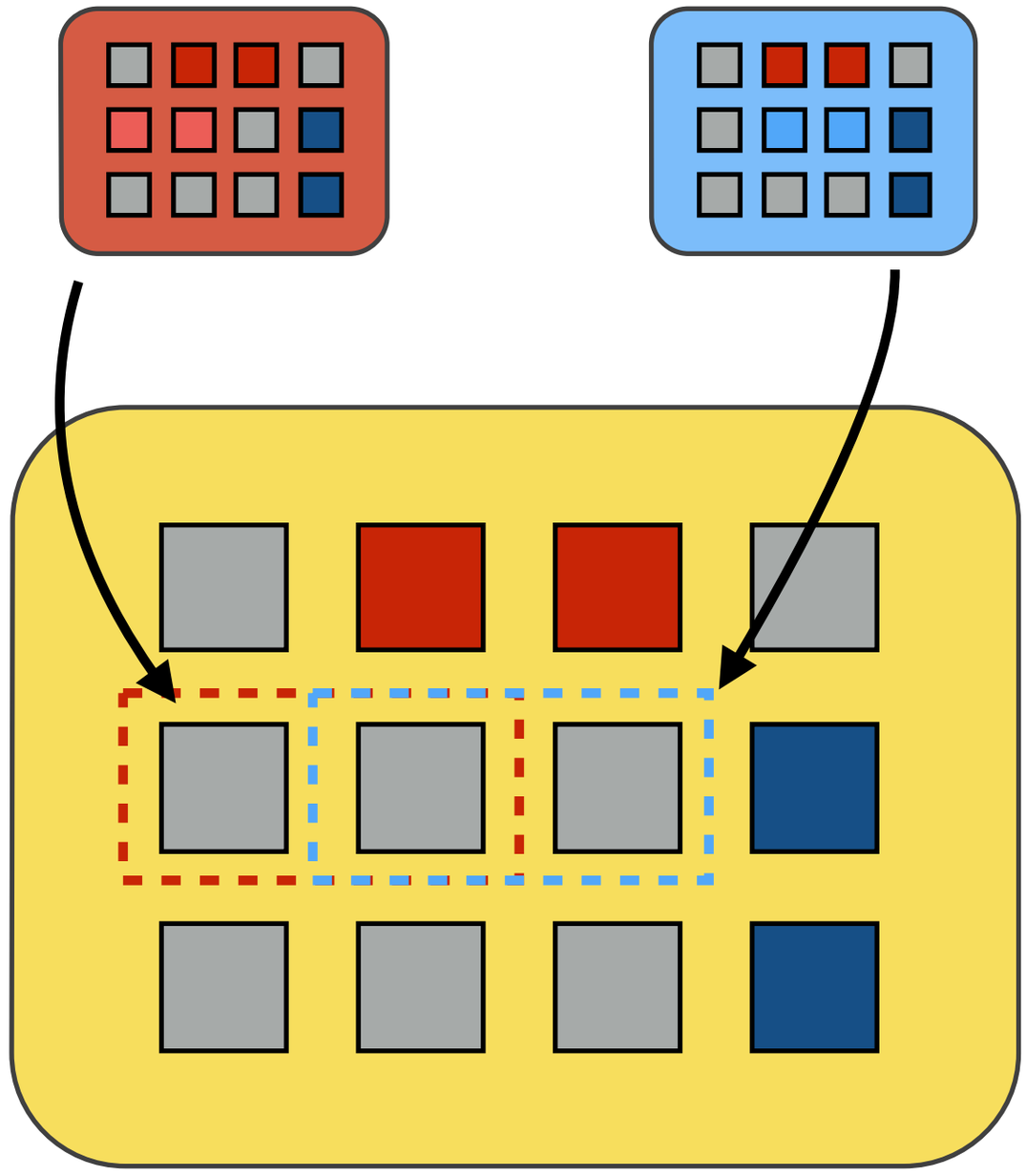
Resource reservation in Omega

Assume that jobs arrive to both FW1 and FW2.

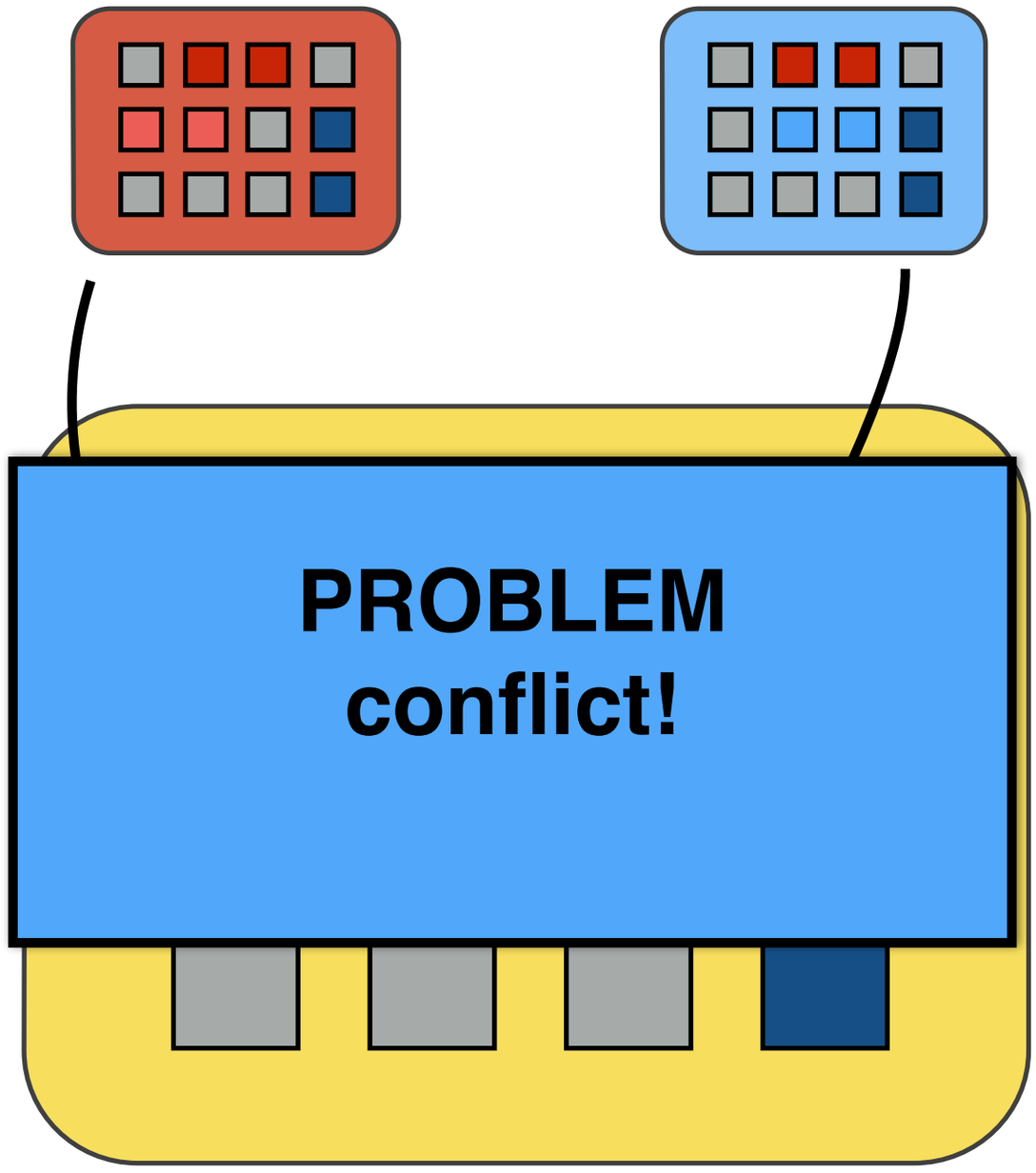
Both FWs try to grab resources that are *free from their POV!*



Resource reservation in Omega



Resource reservation in Omega

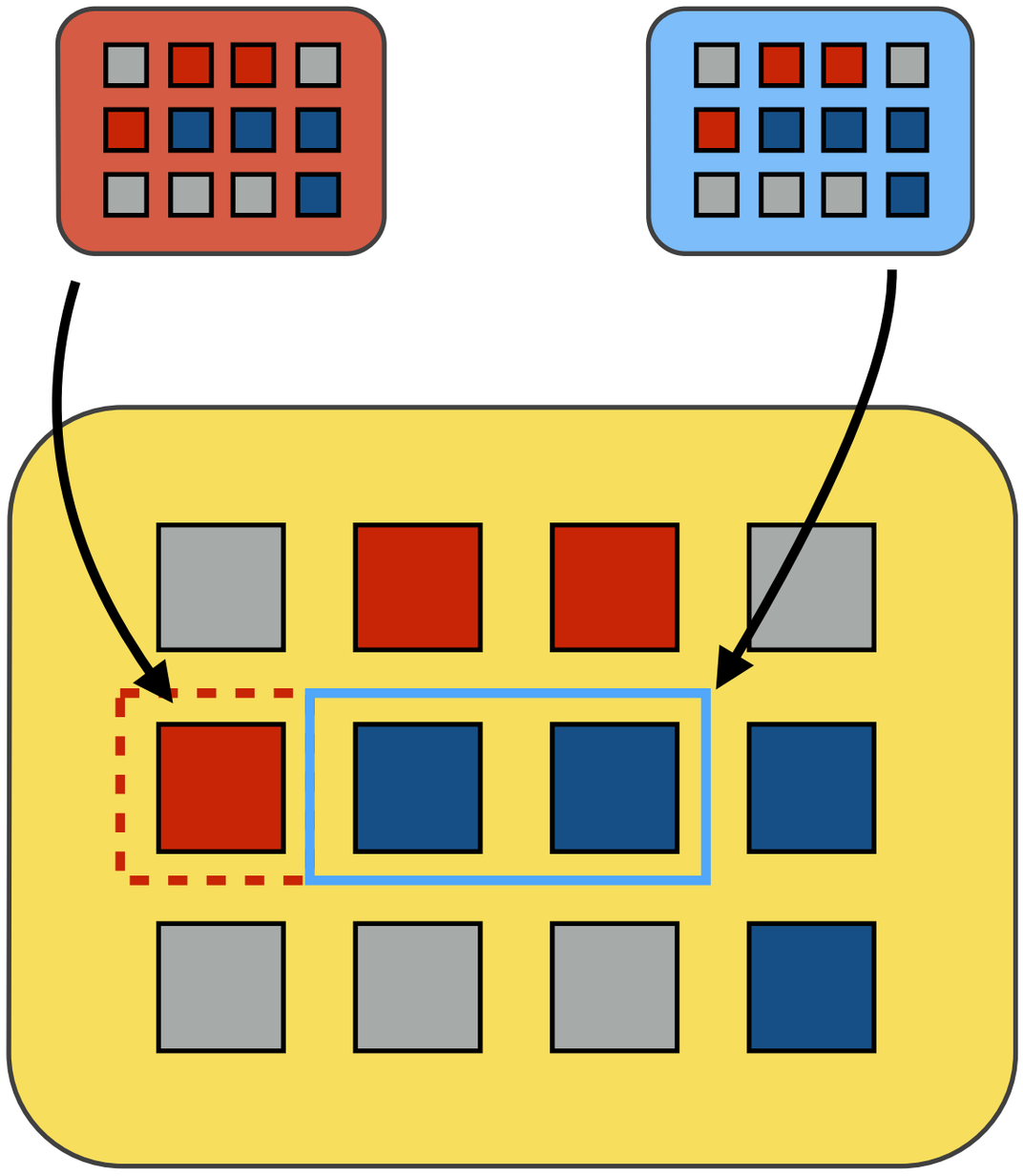


LUND UNIVERSITY

Resource reservation in Omega

Only one transaction can succeed. **Conflict resolved!**

Incremental reservation for FW 1.



LUND UNIVERSITY

To conclude…

Multi-tenancy is key for modern cluster RM systems.

Several of the solutions here are in active development, open-sourced and widely adopted.

Still, surprisingly little math behind… (I’m biased on this)

Ironically, several cases are built on assumed issues of monolithic schedulers. Recent Borg paper shows it might not be so…

Reference list

Hindman et al: *Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center*

Schwarzkopf et al: *Omega: flexible, scalable schedulers for large compute clusters*

Vavilapalli et al: *Apache Hadoop YARN: Yet Another Resource Negotiator*

Verma et al: *Large-scale cluster management at Google with Borg*

LUND
UNIVERSITY