Scheduling

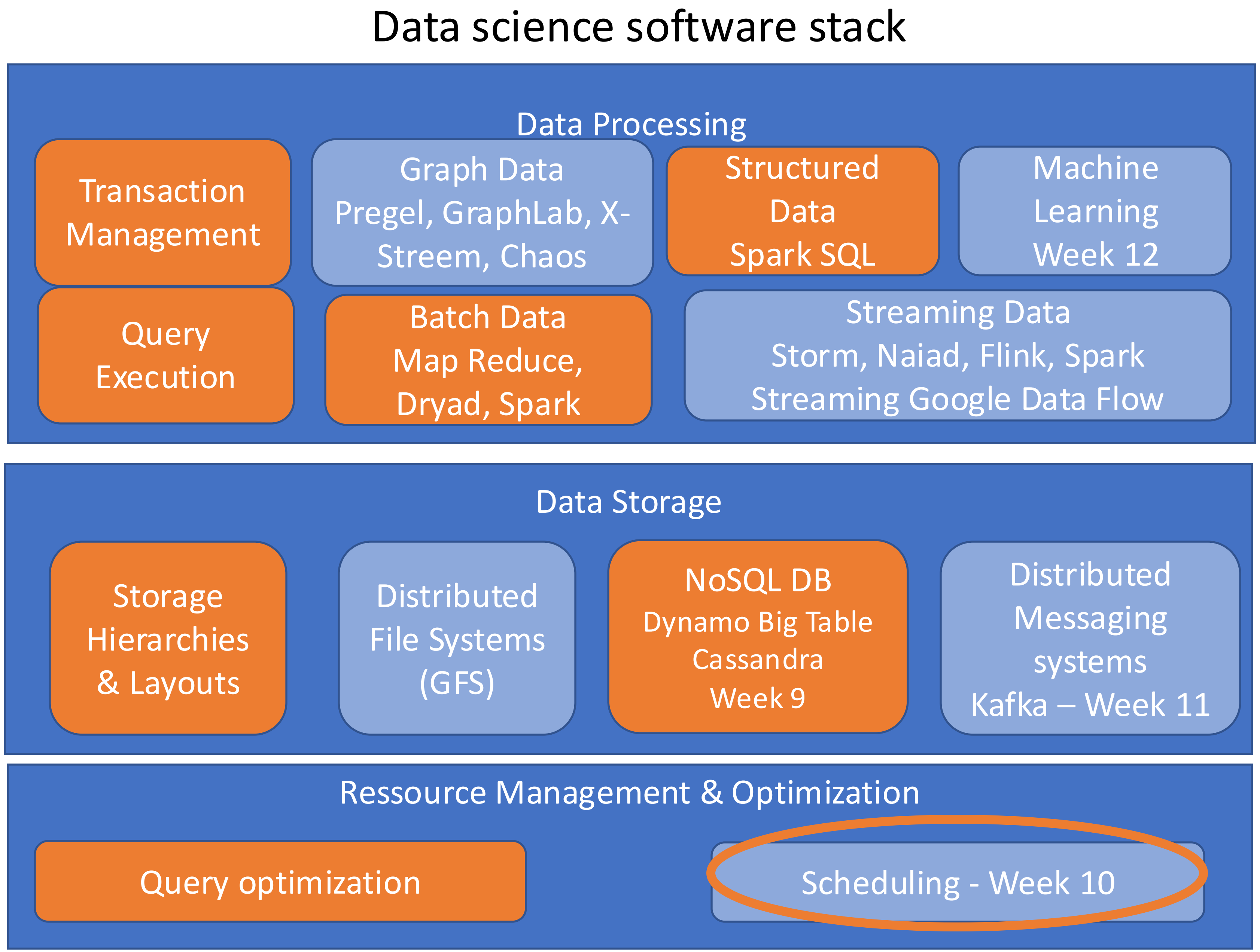
Anne-Marie Kermarrec

Where are we?

Gossip Protocols
Week 7

Consistency protocols
CAP Theorem
Week 9

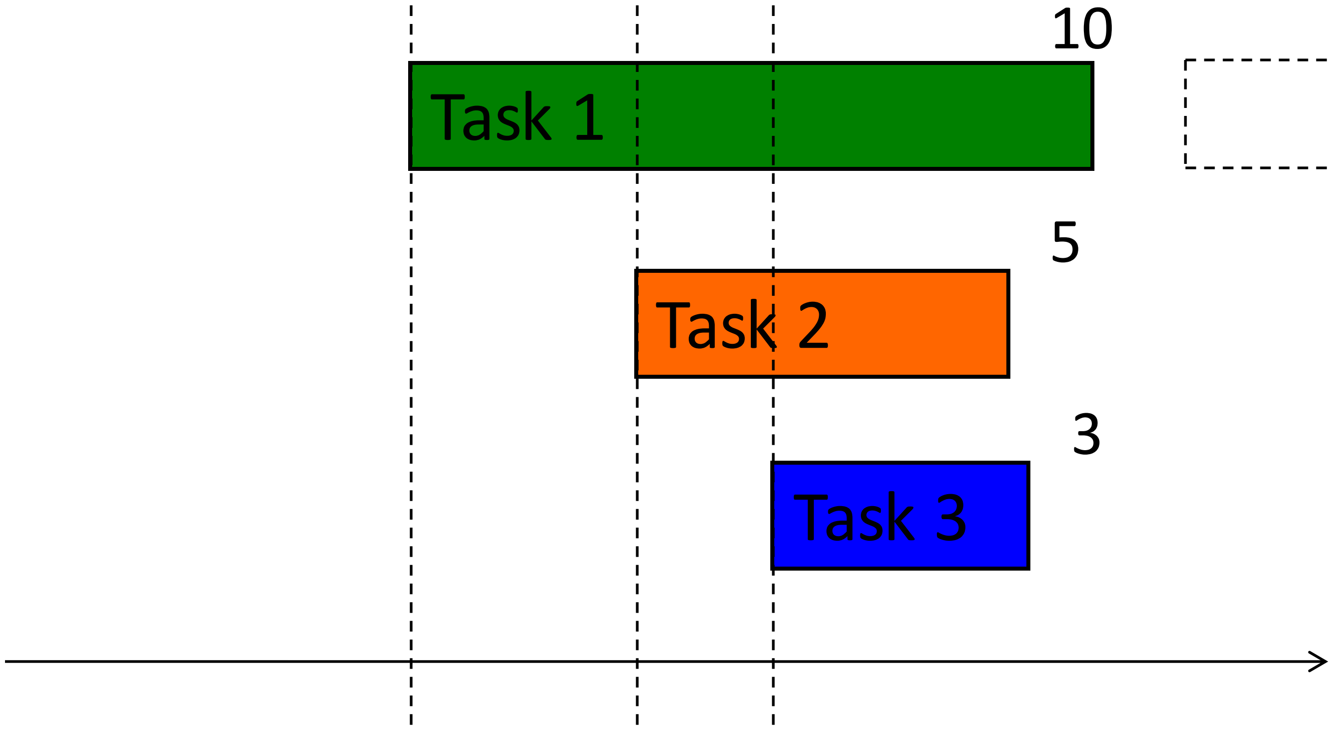
Distributed/decentralized systems
Week 8-12



Scheduling

- Multiple “tasks” to schedule
  - The processes on a single-core OS
  - The tasks of a Hadoop job
  - The tasks of multiple Hadoop jobs
  - The tasks of multiple frameworks
- Limited resources that these tasks require
  - Processor(s)
  - Memory
  - (Less contentious) disk, network
- Scheduling goals
  1. Good throughput or response time for tasks (or jobs)
  2. High utilization of resources
  3. Share resources

Single processor scheduling



Arrival Times → 0 6 8

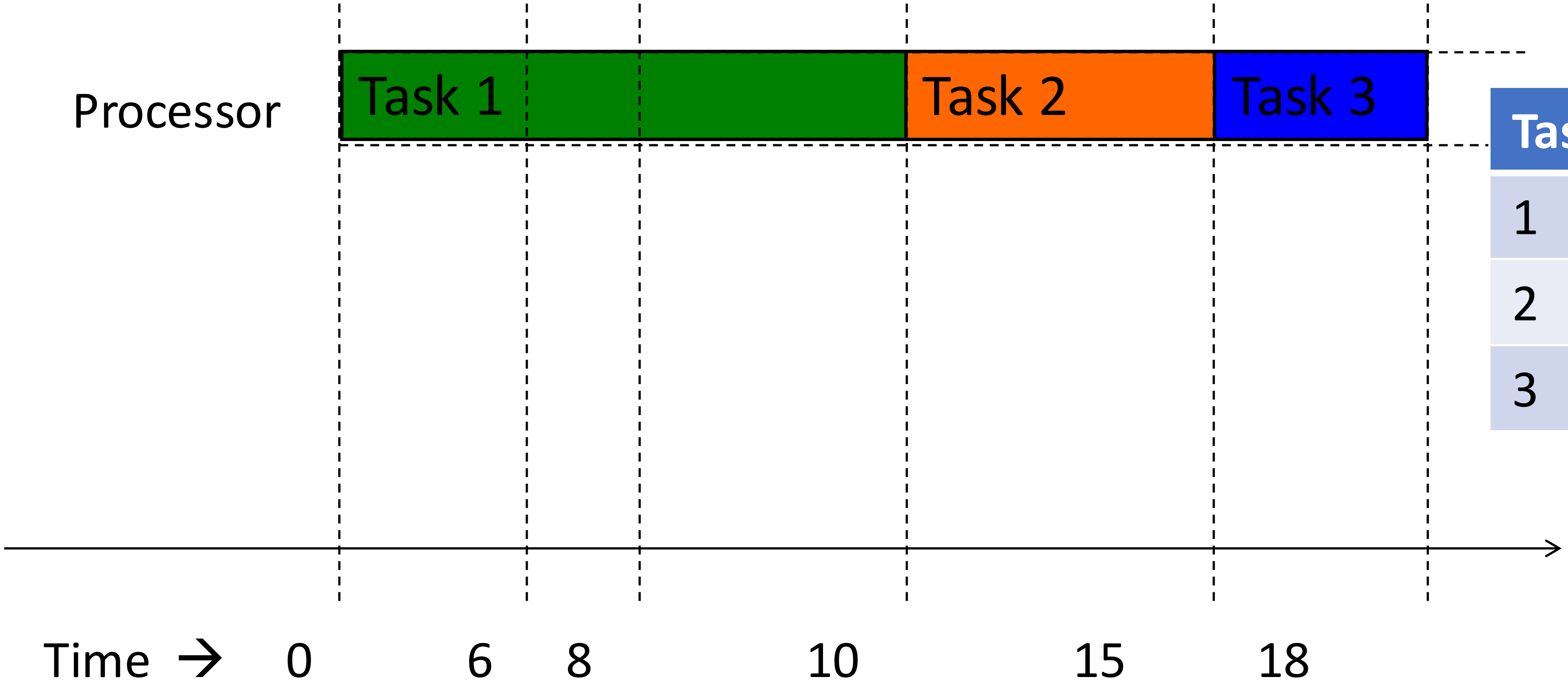
Which tasks run when?

Processor

Task	Length	Arrival
1	10	0
2	5	6
3	3	8

EPFL

FIFO Scheduling (First In First Out)



Task	Length	Arrival
1	10	0
2	5	6
3	3	8

- *Maintain tasks in a queue in order of arrival*
- *When processor free, dequeue head and schedule it*

FIFO/FCFS Performance

- Average completion time may be high
- For our example on previous slides,
  - Average completion time of FIFO/FCFS =

    (Task 1 + Task 2 + Task 3)/3

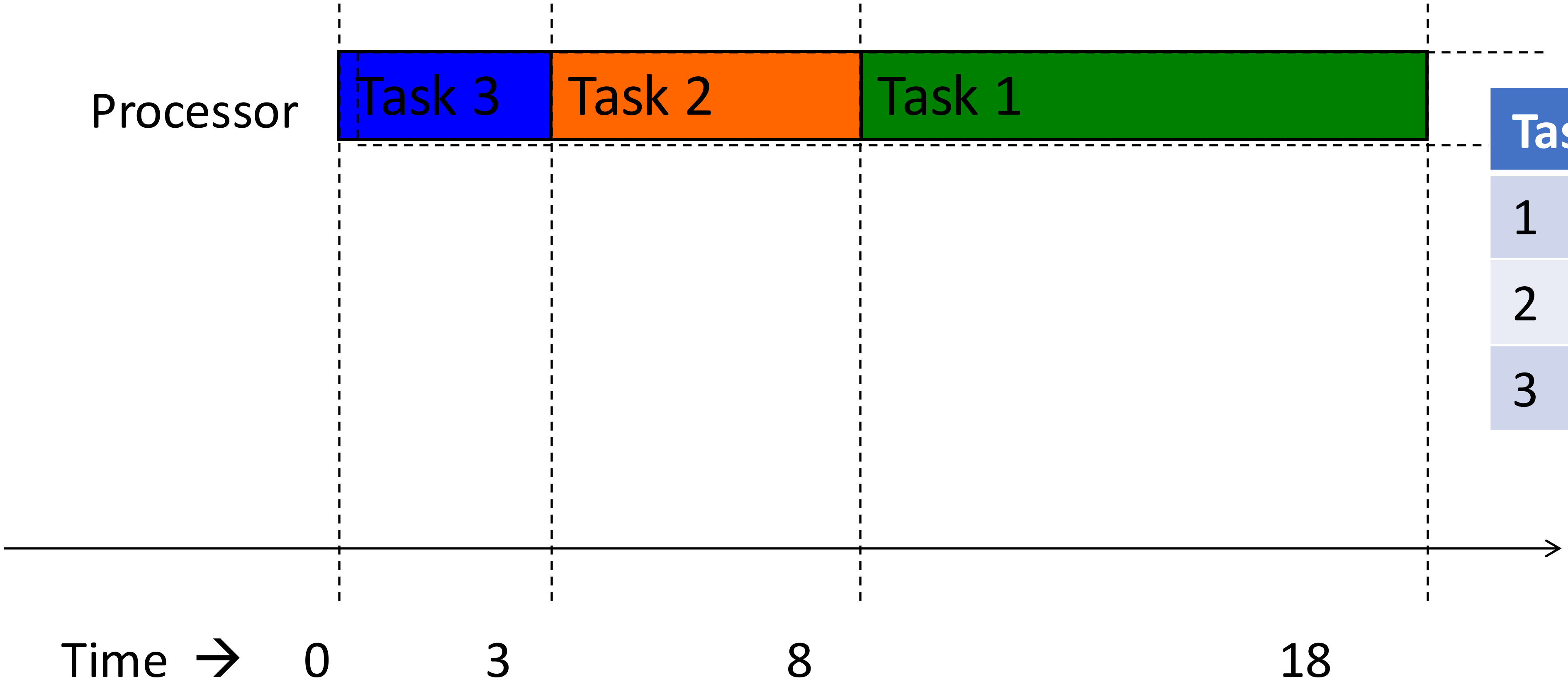
    = (10+15+18)/3

    = 43/3

    = 14.33

EPFL

STF Scheduling (Shortest Task First)



Task	Length	Arrival
1	10	0
2	5	0
3	3	0

- *Maintain all tasks in a queue, in increasing order of running time*
- *When processor free, dequeue head and schedule*

STF is Optimal

- Average completion of STF is the shortest among *all* scheduling approaches
- Average completion time of STF =

  (Task 1 + Task 2 + Task 3)/3

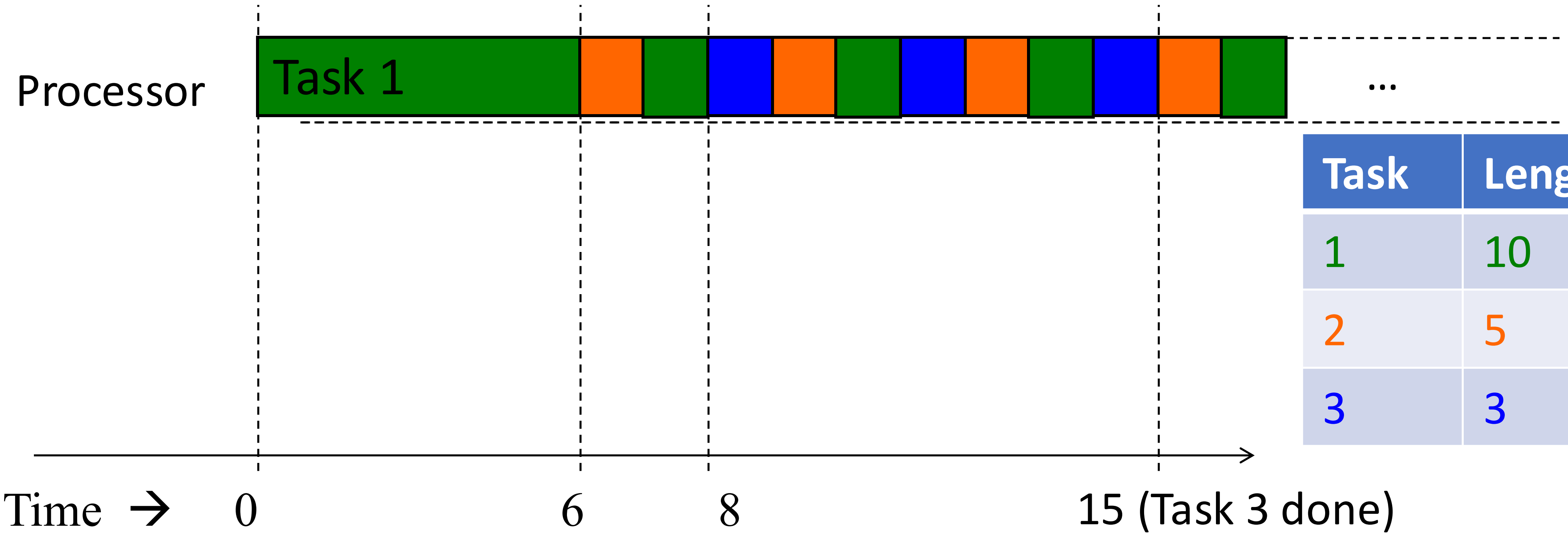
  = (18+8+3)/3

  = 29/3

  = 9.66

  (versus 14.33 for FIFO/FCFS)
- In general, STF is a special case of *priority scheduling*
  - Instead of using time as priority, scheduler could use user-provided priority

Round-Robin Scheduling



Task	Length	Arrival
1	10	0
2	5	6
3	3	8

- *Use a quantum (say 1 time unit) to run portion of task at queue head*
- *Pre-empts processes by saving their state, and resuming later*
- *After pre-empting, add to end of queue*

Round-Robin vs. STF/FIFO

- Round-Robin preferable for
  - Interactive applications
  - User needs quick responses from system
- FIFO/STF preferable for Batch applications
  - User submits jobs, goes away, comes back to get result

Summary

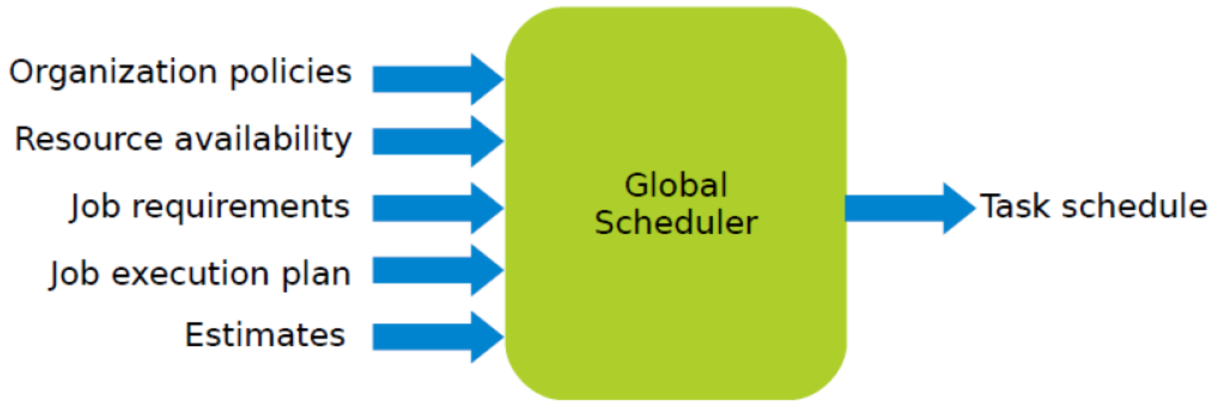
- Single processor scheduling algorithms
  - FIFO/FCFS
  - Shortest task first (optimal)
  - Priority
  - Round-robin
- What about cloud scheduling?

Goals of Cloud Computing Scheduling

- Running multiple frameworks on a single cluster.
- Maximize utilization and share data between frameworks.
- Two main resource management systems:
  - Yarn: cluster management system designed for Hadoop workloads
  - Mesos: manage a variety of different workloads, including Hadoop, Spark, and containerized applications

Schedule frameworks: Global scheduler

- Job requirements
  - Response time
  - Throughput
  - Availability
- Job execution plan
  - Task DAG
  - Inputs/outputs
- Estimates
  - Task duration
  - Input sizes
  - Transfer sizes



Global scheduler

Advantages

- Can achieve optimal schedule

Disadvantages

- Complexity: hard to scale and ensure resilience
- Hard to anticipate future frameworks requirements.
- Need to refactor existing frameworks.

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

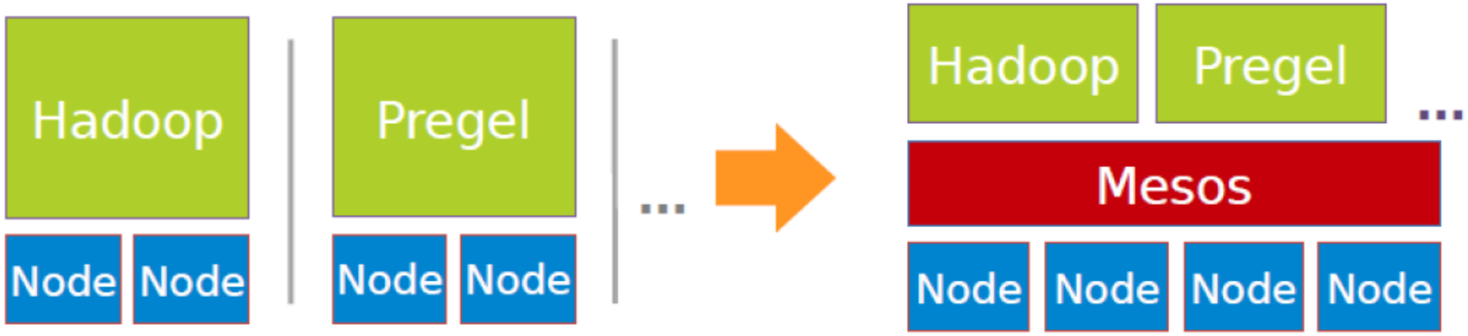
Usenix 2011

Mesos

Coexistence of multiple applications

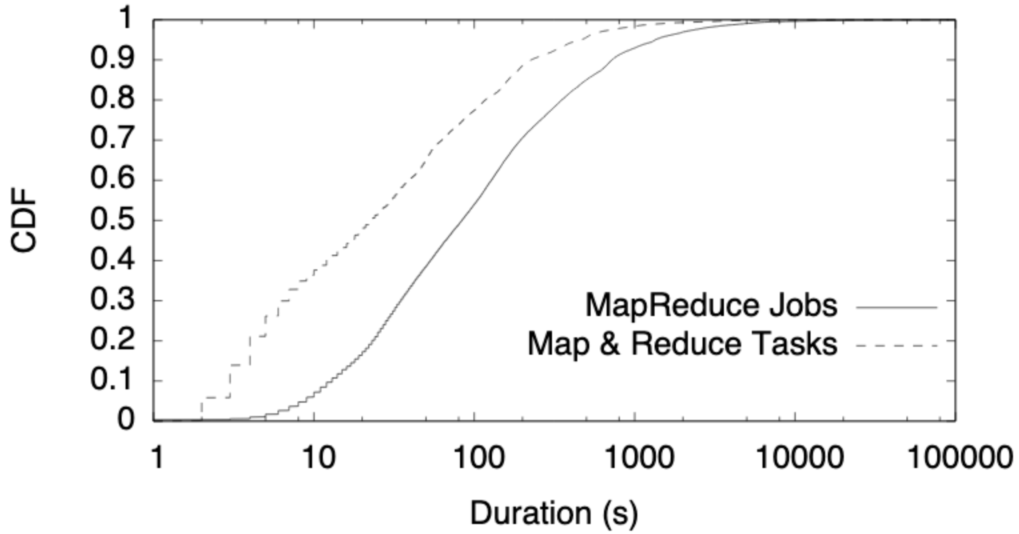
- Ex: FB->Business intelligence, spam detection, ad optimization
- Production job, machine learning ranging from multi-hour computation to 1 mn ad-hoc query

Platform for sharing resources of commodity clusters between multiple diverse frameworks



Mesos model

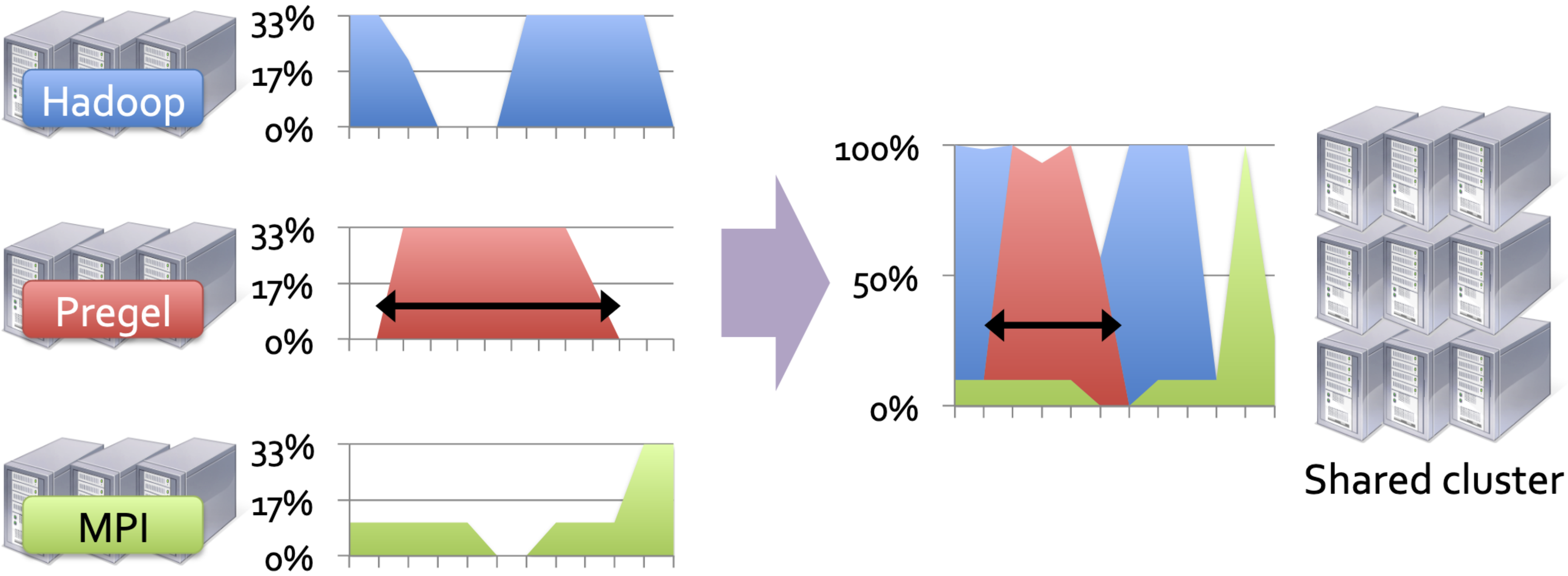
- A framework (e.g., Hadoop, Spark) manages and runs one or more jobs.
- A job consists of one or more tasks.
- A task (e.g., map, reduce) consists of one or more processes running on same machine.
- Short duration of tasks: exploit data locality



CDF of job and task durations in Facebook's Hadoop data warehouse

Challenges

- Various scheduling needs of frameworks
  - Programming model, scheduling needs, task dependencies, data placement, etc.
- Fault-tolerant & high availability
- Avoids the complexity of a central scheduler



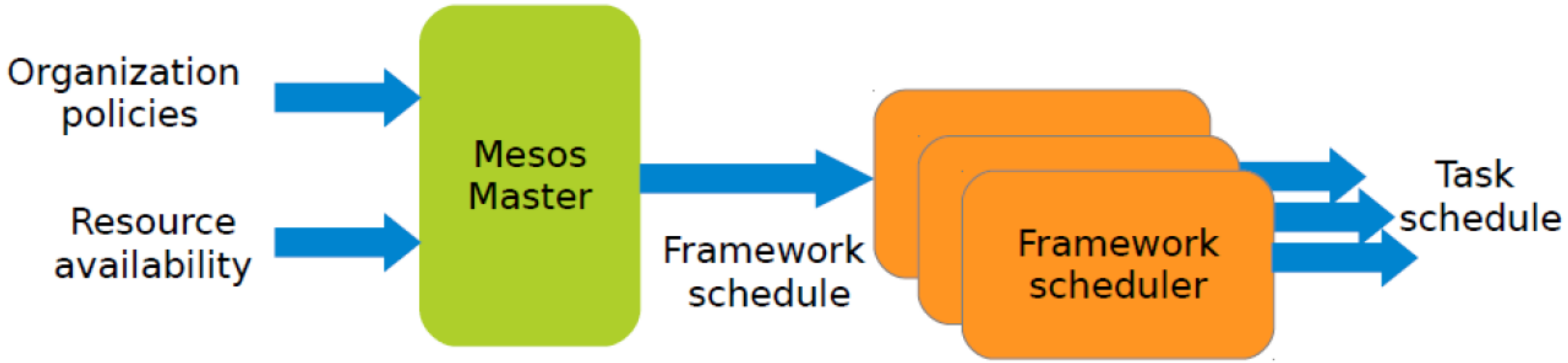
"A Platform for Fine--Grained Resource Sharing in the Data Center " Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony Joseph, Randy Katz, Scott Shenker, Ion Stoica. Usenix 2011

Ressource offers

- **Delegates control over scheduling to the frameworks**
- 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
  - 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

Distributed scheduler



Distributed scheduler

- Master sends *resource offers* to frameworks
- Frameworks select which offers to accept and which tasks to run
- Unit of allocation: resource offer
  - Vector of available resources on a node
  - For example, node1: (1CPU; 1GB), node2: (4CPU; 16GB)

Distributed scheduler

Advantages

- Simple: easier to scale and make resilient
- Easy to port existing frameworks, support new ones

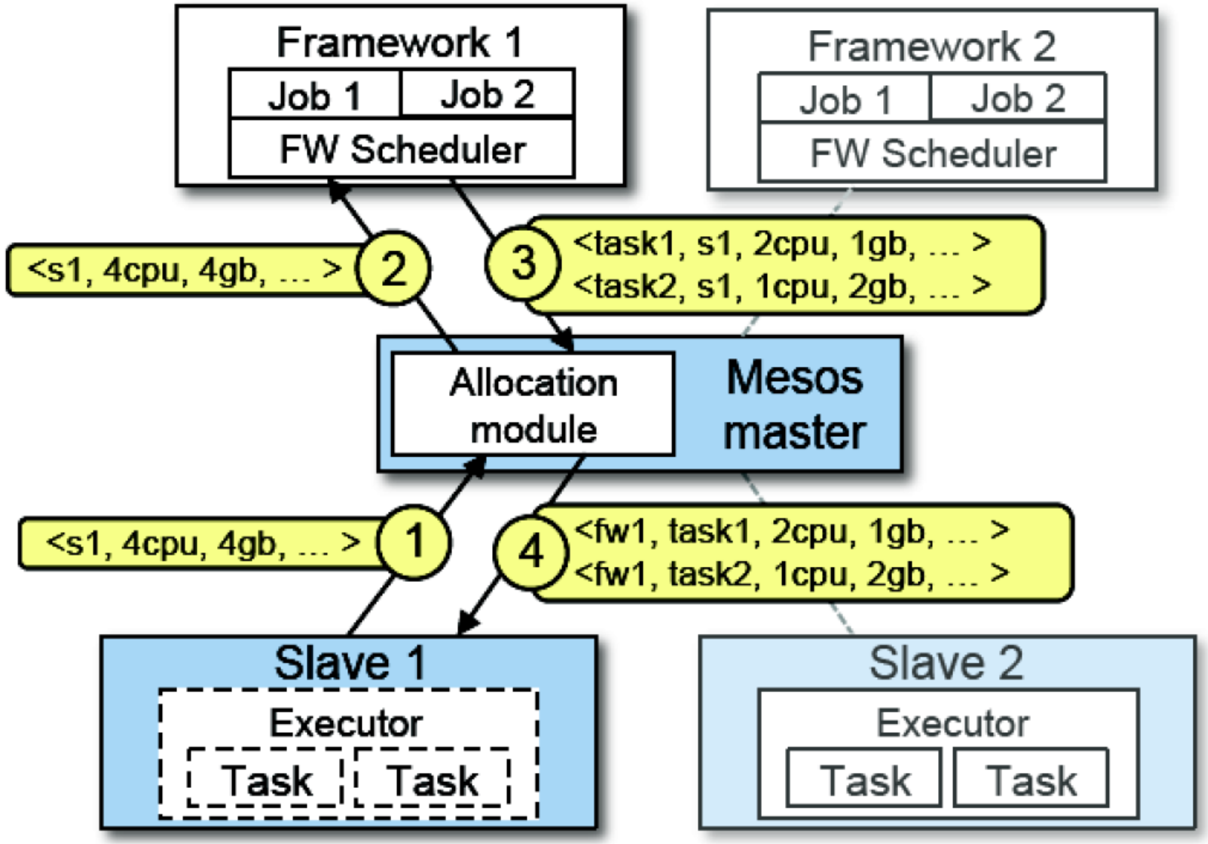
Disadvantages

- May not always lead to optimal
- In practice meet goals such as data locality almost perfectly

Mesos architecture

Pluggable scheduler picks framework to send an offer to.

Slaves continuously send status updates about resources to the Master



Framework scheduler selects resources and provides tasks.

Framework executors launch tasks.

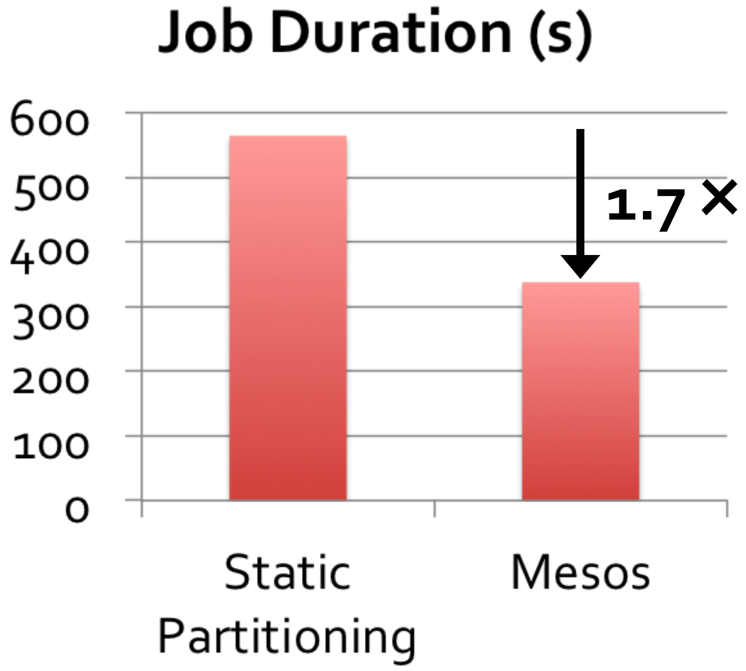
Mesos vs Static Partitioning

- 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 ×

Data Locality with Resource Offers

- Ran 16 instances of Hadoop on a shared HDFS cluster
- Used delay scheduling in Hadoop to get locality (wait a short time to acquire data-local nodes)



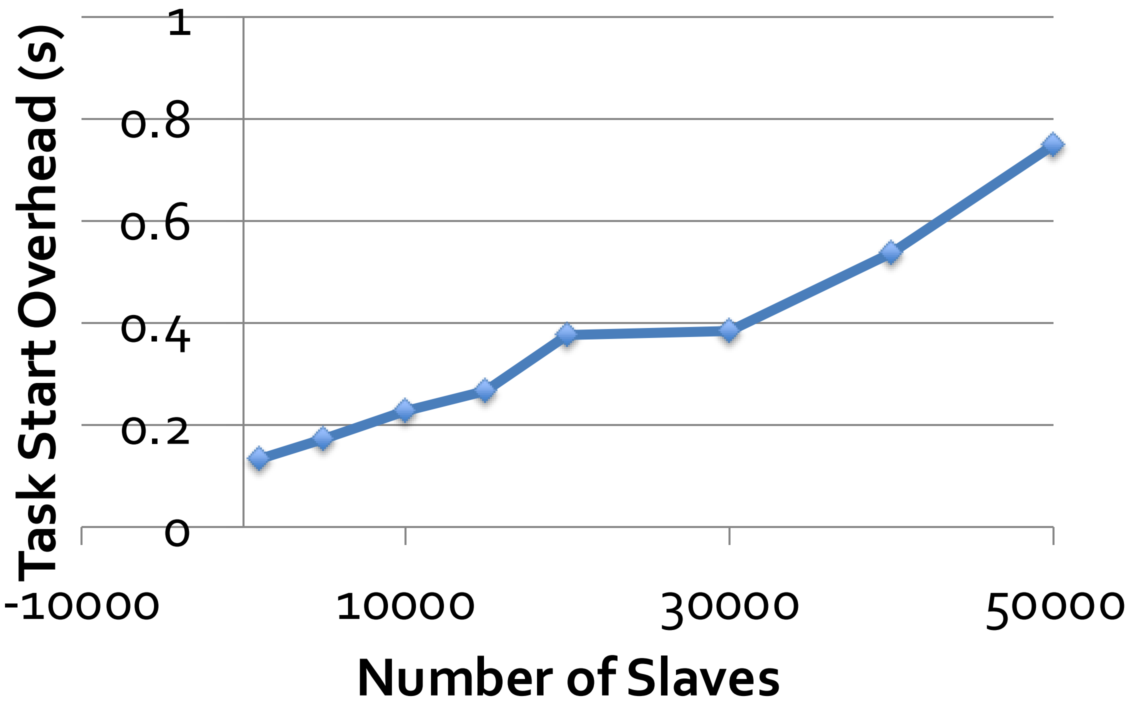
EPFL

Scalability

- Mesos only performs *inter-framework* scheduling (e.g. fair sharing), which is easier than intra-framework scheduling

Result:

Scaled to 50,000 emulated slaves, 200 frameworks, 100K tasks



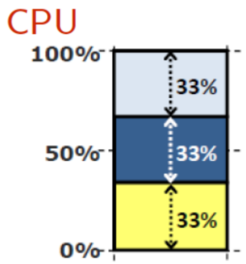
Who is using Mesos

- Apple uses it to power the back end of SIRI
- Netflix uses it for batch and stream processing, anomaly detection, machine learning
- Twitter uses it for analytics and ads

Resource allocation in Mesos

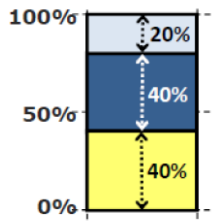
How to allocate resources of different types?

Single Resource: Fair Sharing



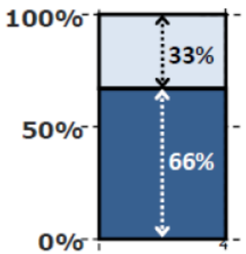
n users want to share a resource, e.g., CPU.

- Solution: allocate each 1/n of the shared resource.



Generalized by max-min fairness.

- Handles if a user wants less than its fair share.
- E.g., user A wants no more than 20%.



Generalized by weighted max-min fairness

- Give weights to users according to importance.
  - E.g., user A gets weight 1, user B weight 2.

Max-min fairness: example

- 1 resource: CPU
- Total resources: 20 CPU
- User A has *x* tasks and wants (1CPU) per task
- User B has *y* tasks and wants (2CPU) per task

max(x; y) (maximize allocation)

subject to

$$x + 2y = 20 \text{ (CPU constraint)}$$

$$x = 2y$$

So $x = 10, y = 5$

Properties of Max-Min Fairness

Share guarantee

- Each user can get at least 1/n of the resource.
- But will get less if her demand is less.

Strategy proof

- Users are not better off by asking for more than they need.
- Users have no reason to lie.

Max-Min fairness is the only reasonable mechanism with these two properties.

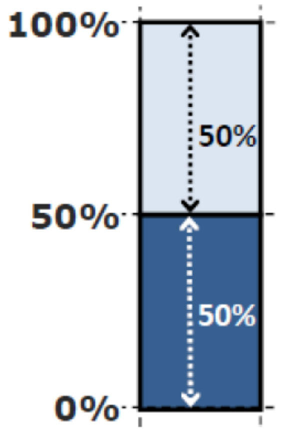
Widely used: OS, networking, datacenters, can be used in Mesos

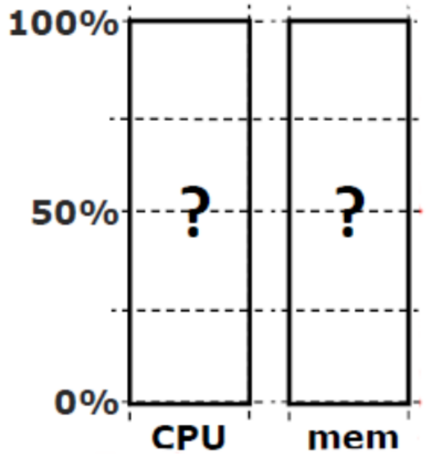
When is Max-Min Fairness NOT Enough?

Need to schedule multiple, heterogeneous resources, e.g.,
CPU, memory, etc.

Problem

- Single resource example
  - 1 resource: CPU
  - User A wants 1CPU per task
  - User B wants 2CPU per task
- Multi-resource example
  - 2 resources: CPUs and mem
  - User A wants 1CPU; 2GB per task
  - User B wants 2CPU; 4GB per task





A Natural Policy (1/2)

Fairness: give weights to resources (e.g., 1 CPU = 1 GB) and equalize total value given to each user.

- Total resources: 28 CPU and 56 GB RAM (e.g., 1 CPU = 2 GB = 1$)
  - User A has x tasks and wants 1CPU; 2GB per task
  - User B has y tasks and wants 1CPU; 4GB per task
- Asset fairness yields

  max(x; y)

  x + y <= 28 (CPU constraints)

  2x + 4y <= 56 (Memory constraint)

  2x = 3y (every user spends the same 1 CPU = 2 GB)

User A: x = 12: (43%CPU; 43%GB (86%))
User B: y = 8: (28%CPU; 57%GB (85%))

A Natural Policy (2/2)

- Problem: violates share guarantee.
  - User A: x = 12: (43%CPU; 43%GB (86%))
  - User B: y = 8: (28%CPU; 57%GB (85%))

- User A gets less than 50% of both CPU and RAM.
- Better off in a separate cluster with half the resources

Challenge: Can we find a fair sharing policy that provides

Share guarantee & Strategy-proofness

Can we generalize max-min fairness to multiple resources?

Dominant-Resource Fair Scheduling

Dominant Resource Fairness (DRF)

- Proposed by researchers from U. California Berkeley
- Proposes notion of fairness across jobs with multi-resource requirements
- They showed that DRF is
  - Fair for multi-tenant systems
  - Strategy-proof: tenant cannot benefit by lying
  - Envy-free: tenant cannot envy another tenant’s allocations

Where is DRF Useful?

- DRF is
  - Usable in scheduling VMs in a cluster
  - Usable in scheduling Hadoop in a cluster
- DRF used in Mesos
- DRF-like strategies also used some cloud computing company's distributed OS's

Dominant Resource Fairness (DRF) (1/2)

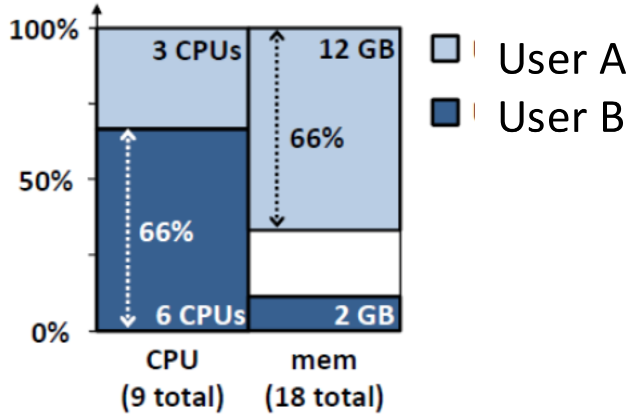
- Dominant resource of a user: the resource that user has the biggest share of.
  - Total resources: 8CPU; 5GB
  - User A allocation: 2CPU; 1GB
    - 2/8 = 25% CPU and 1/5 = 20% RAM
    - Dominant resource of User A is CPU (25% > 20%)

- Dominant share of a user: the fraction of the dominant resource she is allocated.
  - User A dominant share is 25%.

Dominant Resource Fairness (DRF) (2/2)

- Apply max-min fairness to dominant shares: give every user an equal share of her dominant resource.
- Equalize the dominant share of the users.
- Total resources: (9CPU; 18GB)
- User A wants (1CPU; 4GB) for each task; Dominant resource: RAM (1/9 < 4/18) 22% RAM
- User B wants (3CPU; 1GB) for each task; Dominant resource: CPU (3/9 > 1/18) 33% CPU
- x is the number of tasks allocated to User A, y to User B

max(x; y) subject to
x + 3y <= 9 (CPU constraints)
4x + y <=18 (Memory constraints)
4x/18 = 3y/9 (equalize dominant shares)
User A: x = 3: (33%CPU; 66%GB)
User B: y = 2: (66%CPU; 16%GB)



Algorithm

Algorithm 1 DRF pseudo-code

$R = \langle r_1, \cdots, r_m \rangle$ ▷ total resource capacities
$C = \langle c_1, \cdots, c_m \rangle$ ▷ consumed resources, initially 0
$s_i \ \ (i = 1..n)$ ▷ user i's dominant shares, initially 0
$U_i = \langle u_{i,1}, \cdots, u_{i,m} \rangle \ \ (i = 1..n)$ ▷ resources given to user i, initially 0

pick user i with lowest dominant share s_i
$D_i \leftarrow$ demand of user i's next task
if $C + D_i \leq R$ **then**
    $C = C + D_i$ ▷ update consumed vector
    $U_i = U_i + D_i$ ▷ update i's allocation vector
    $s_i = \max_{j=1}^{m}\{u_{i,j}/r_j\}$
else
    return ▷ the cluster is full
end if

EPFL

User A wants (1CPU; 4GB)
User B wants (3CPU; 1GB)

Total resources: (9CPU; 18GB)

Step 0: No tasks assigned.
- Dominant shares: A = 0%, B = 0%

Step 1: Assign 1 task to User A (lowest dominant share)
- A: 1 CPU, 4 GB → dominant share = 4/18 = **22.2%**
- B: 0 → 0%
- Next: assign to **User B**

Step 2: Assign 1 task to User B
- A: 1 CPU, 4 GB → 22.2%
- B: 3 CPU, 1 GB → 3/9 = **33.3%**
- Next: A (smaller dominant share)

Step 3: A gets 2nd task
- A: 2 CPU, 8 GB → 8/18 = **44.4%**
- B: 3 CPU, 1 GB → 33.3%
- Next: B

Step 4: B gets 2nd task
- A: 2 CPU, 8 GB → 44.4%
- B: 6 CPU, 2 GB → 6/9 = **66.6%**
- Next: A

Step 5: A gets 3rd task
- A: 3 CPU, 12 GB → 12/18 = **66.6%**
- B: 6 CPU, 2 GB → 66.6%
- Equal! Can't go further without exceeding total resources.

Example

- At the end of the schedule
  - User A gets (3CPU,12GB)
  - User B gets (6CPU, 2GB)

- Corresponds to the solution
  - User A: $x = 3$: (33%CPU; 66%GB)
  - User B: $y = 2$: (66%CPU; 16%GB)

DRF Fairness

- For a given job, the % of its dominant resource type that it gets cluster-wide, is the same for all jobs
  - Job 1's % of RAM = Job 2's % of CPU
- Can be written as linear equations, and solved

Other DRF Details

- DRF generalizes to multiple jobs
- DRF also generalizes to more than 2 resource types
  - CPU, RAM, Network, Disk, etc.
- DRF ensures that each job gets a fair share of that type of resource which the job desires the most
  - Hence fairness

Summary: Scheduling

- Scheduling very important problem in cloud computing
  - Limited resources, lots of jobs requiring access to these resources
- Single-processor scheduling
  - FIFO/FCFS, STF, Priority, Round-Robin
- Centralized Scheduler (Hadoop)
- Two-level Scheduler (Mesos, Yarn)
- Distributed Scheduler (Sparrow)
- Hybrid Scheduling (Omega, Hawk)

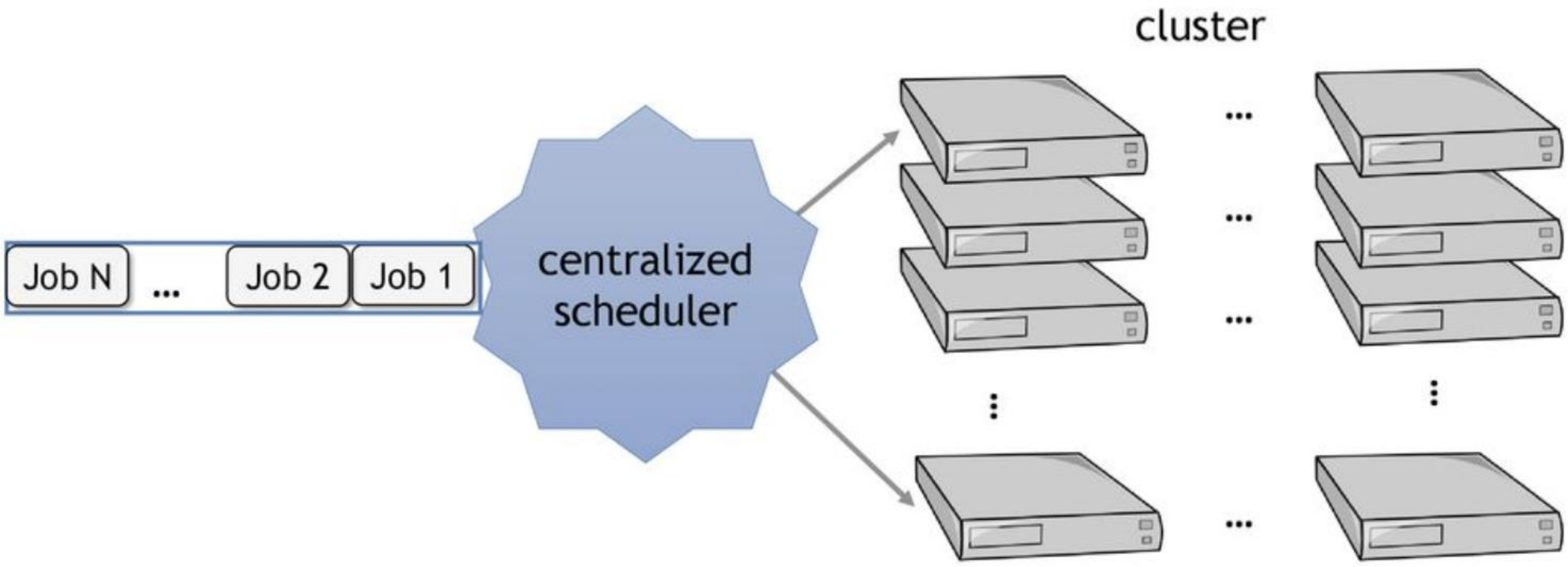
References

- B. Hindman et al., “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center", USENIX 2011
- A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types”. NSDI 2011
- V. Vavilapalli et al., “Apache hadoop yarn: Yet another resource negotiator", ACM Cloud Computing 2013
- P Delgado, F Dinu, AM Kermarrec, W Zwaenepoel, “Hawk: Hybrid datacenter scheduling”, USENIX ATC, 2015

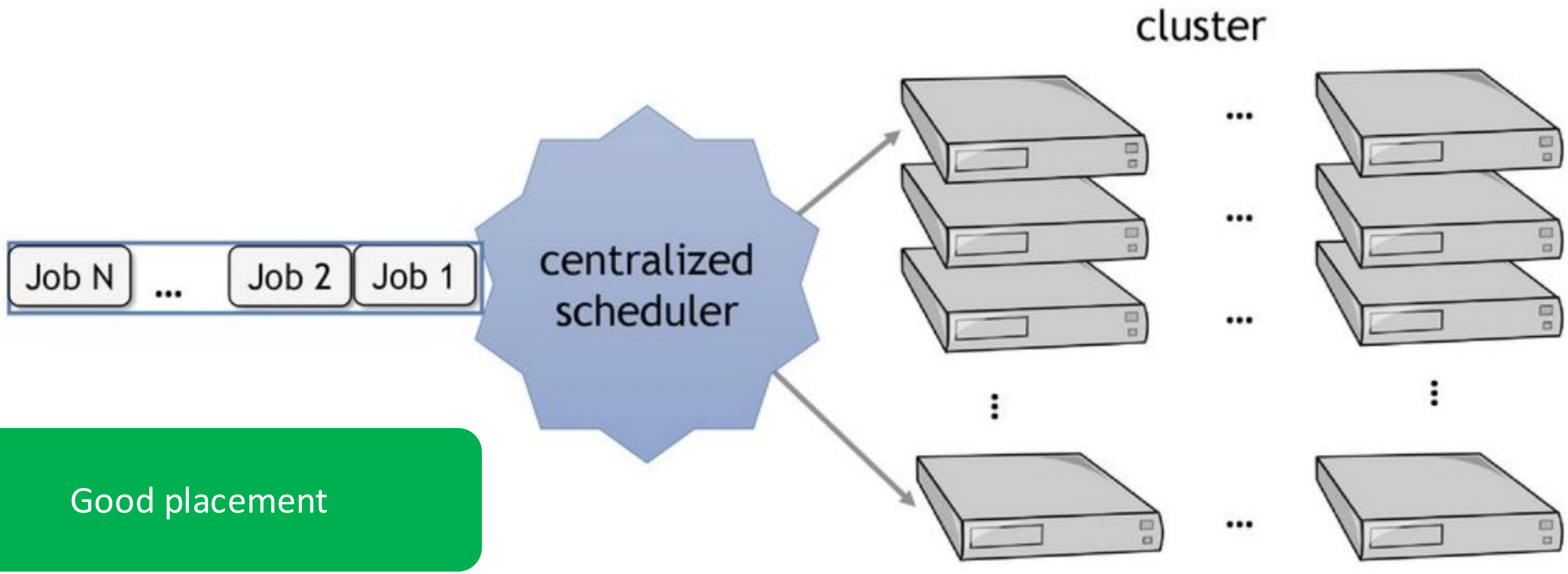
Hawk: Hybrid Datacenter Scheduling

Usenix, ATC 2015

Centralized Schedulers



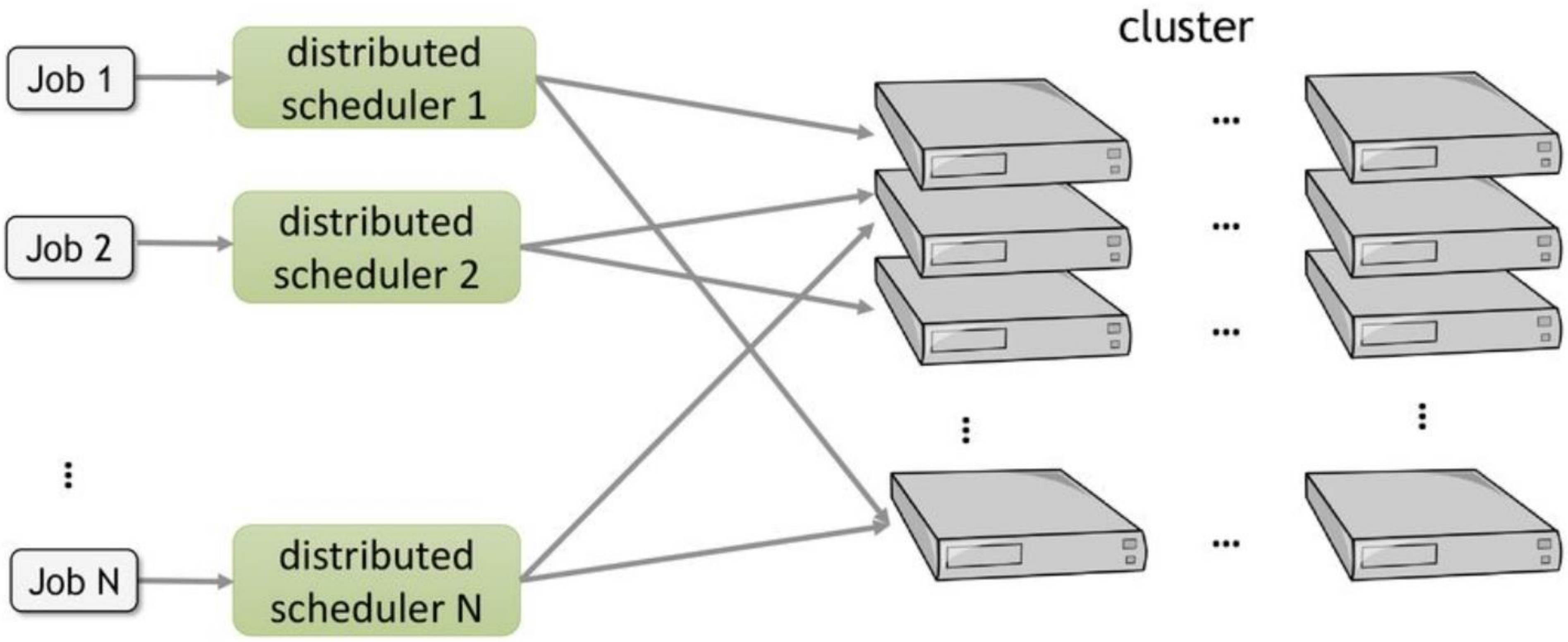
Centralized Schedulers



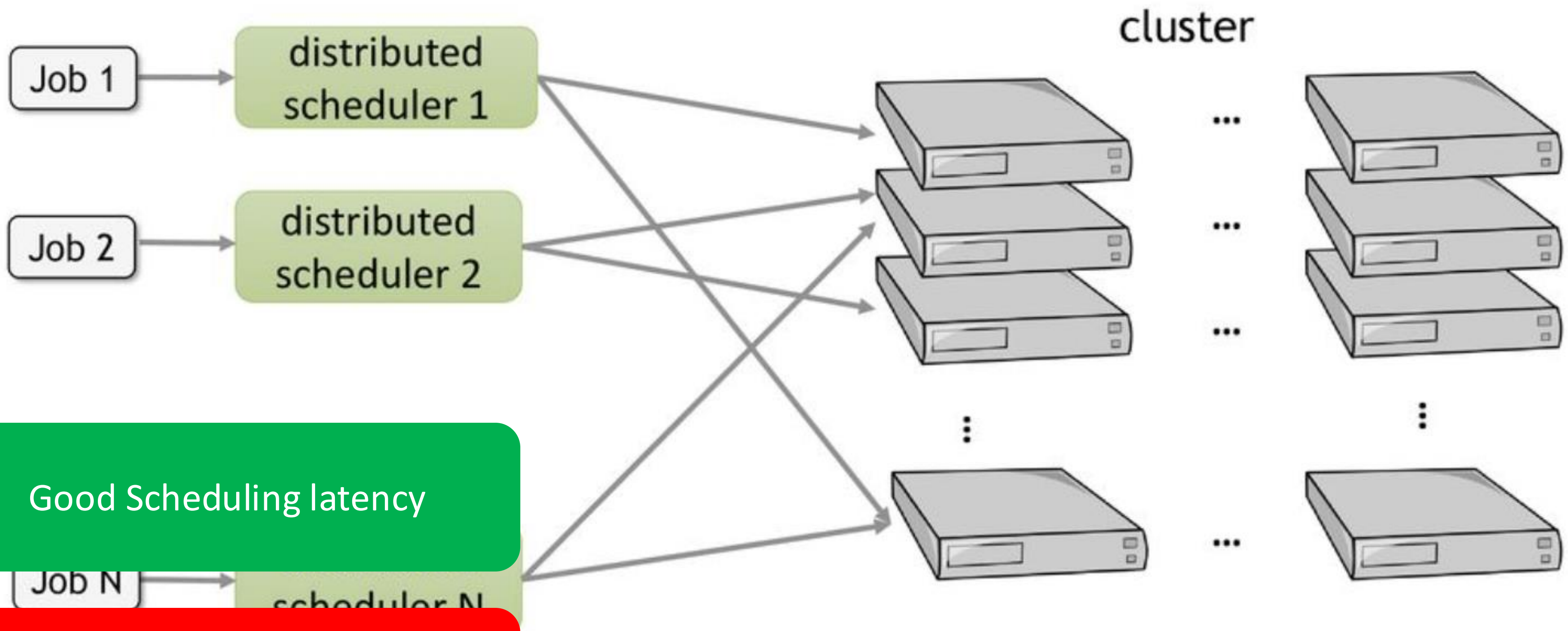
Good placement

High scheduling latency

Distributed Scheduling



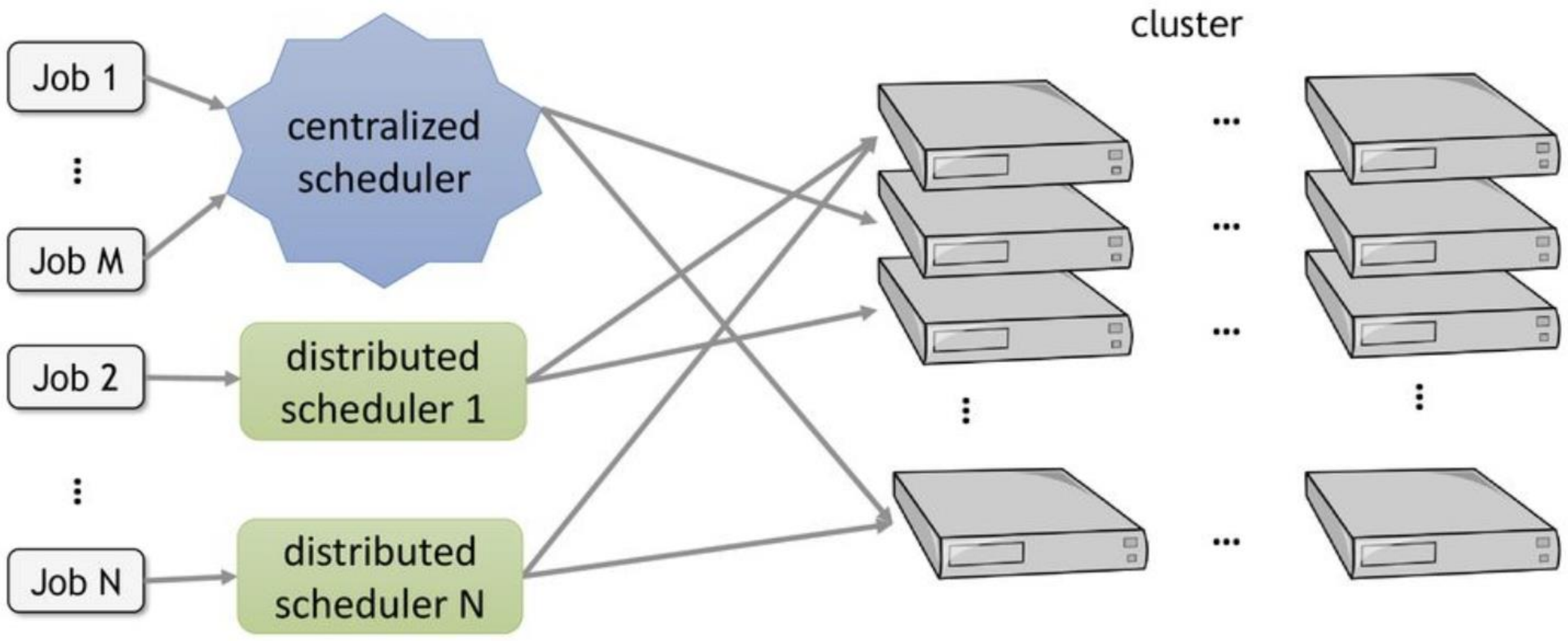
Distributed Scheduling



Good Scheduling latency

Sub-optimal placement

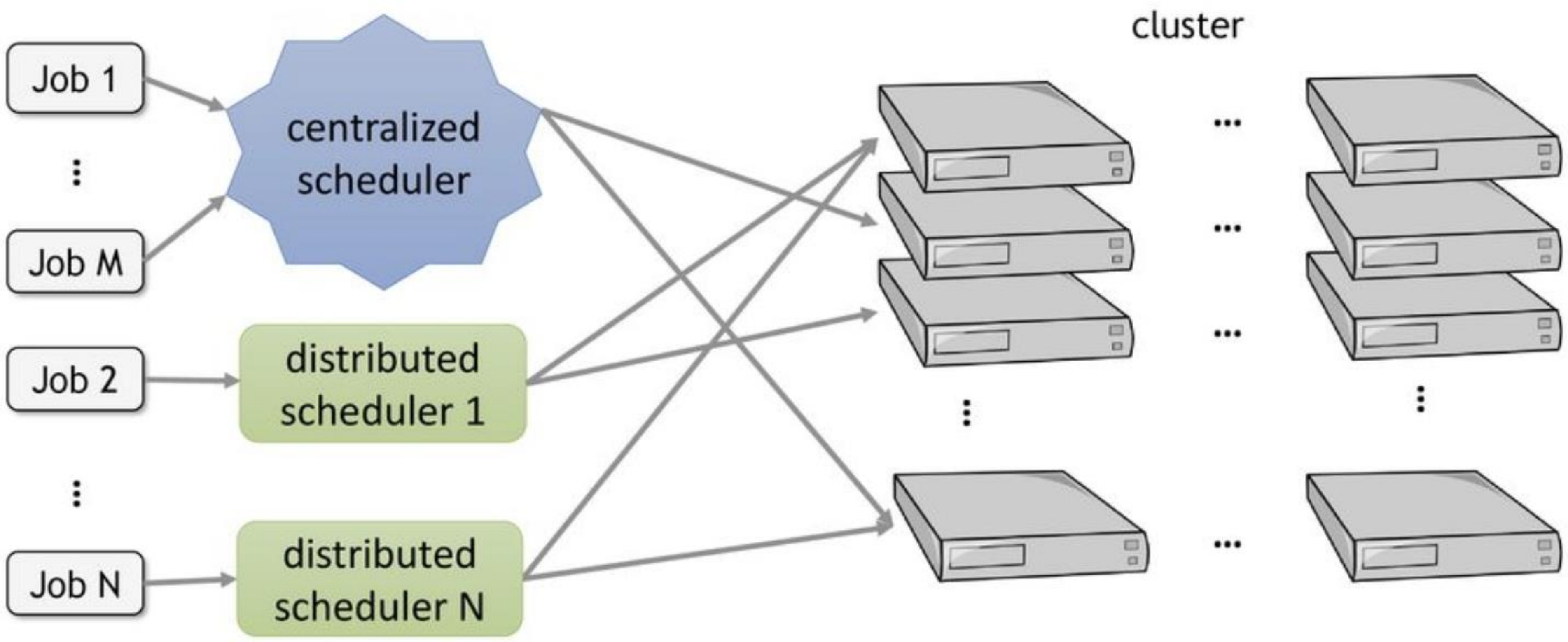
Hybrid Scheduling



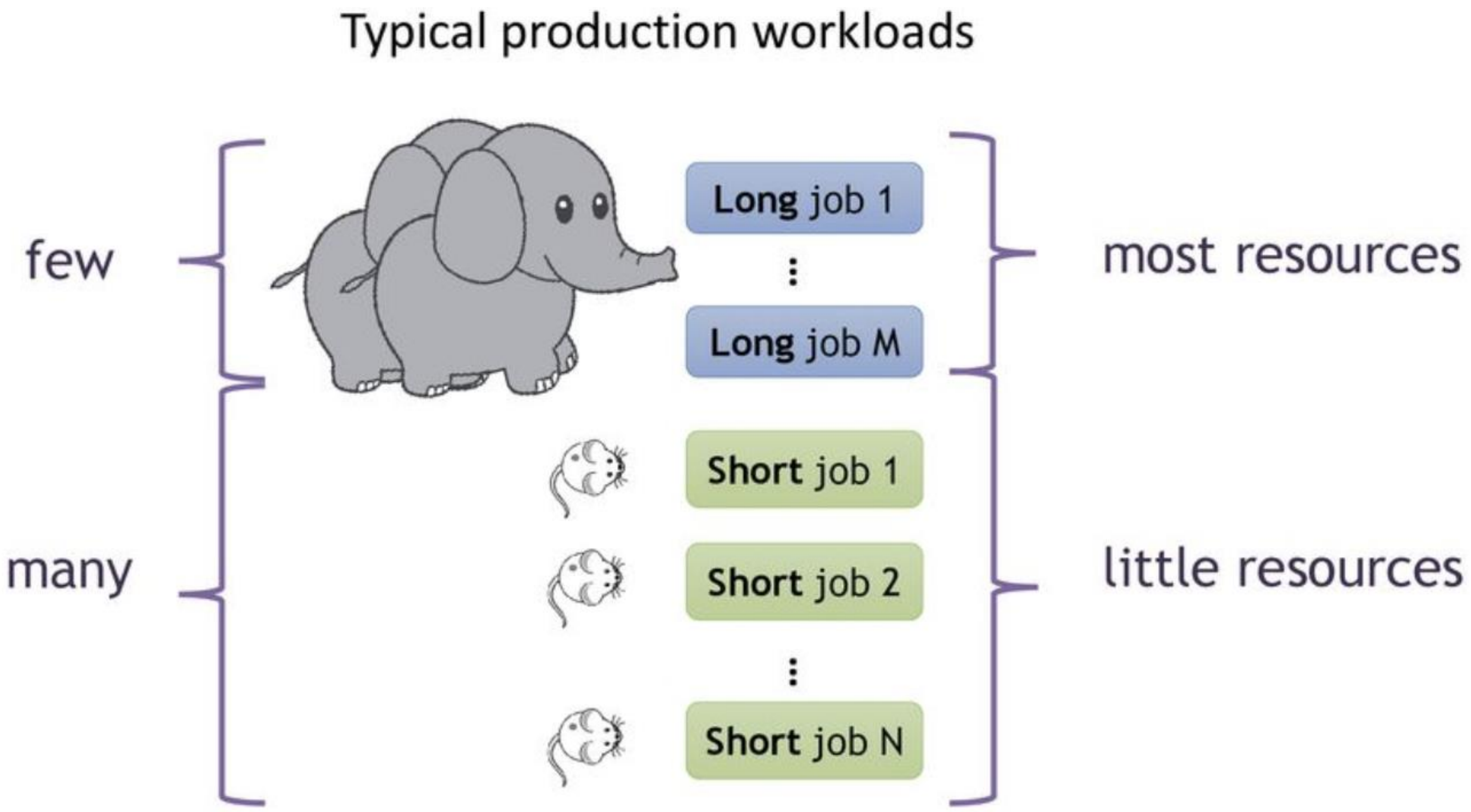
Hawk: Hybrid Scheduling

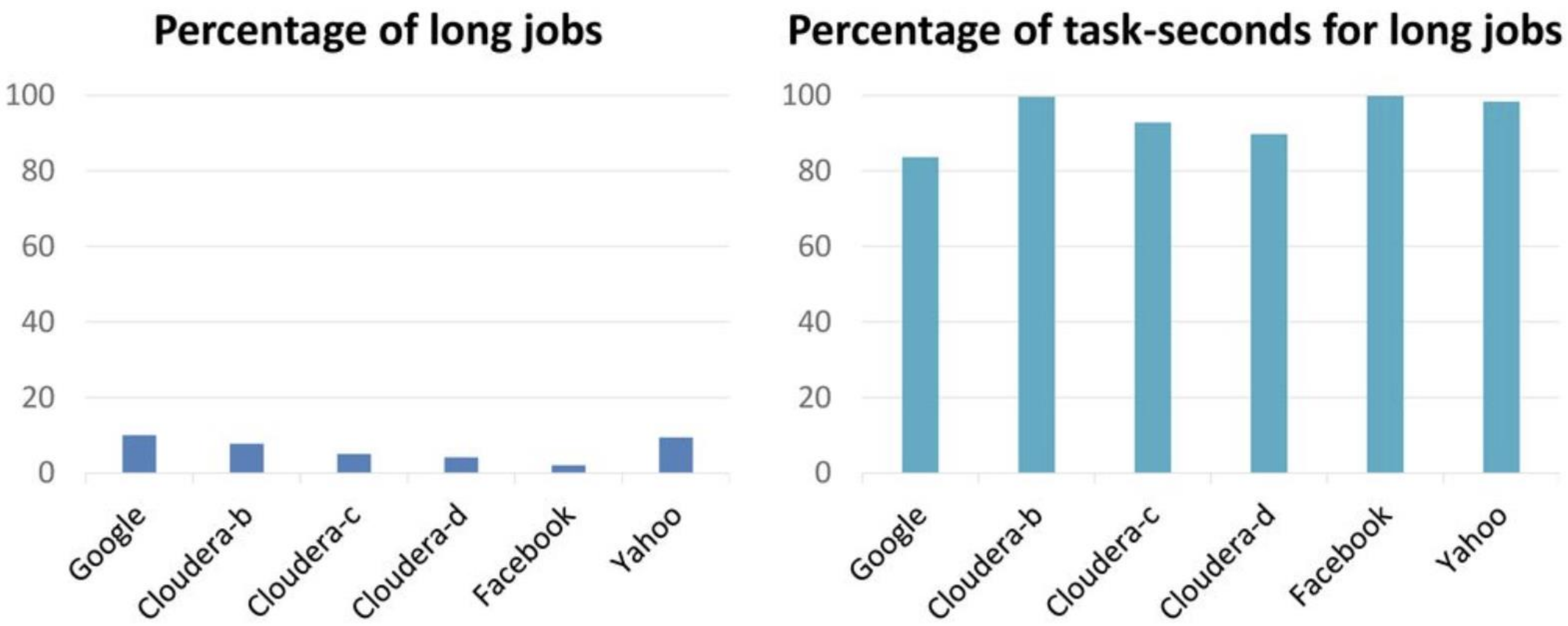
- Long jobs -> centralized
- Short jobs -> distributed

Hawk: Hybrid Scheduling

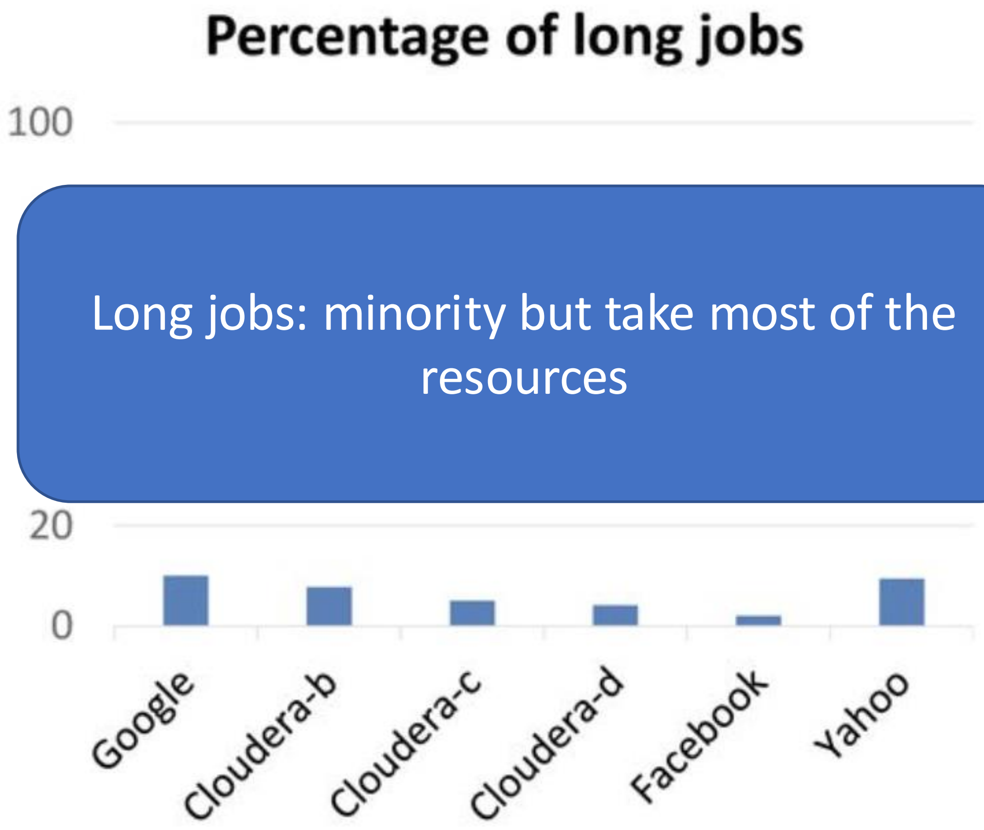


Hawk: Rationale

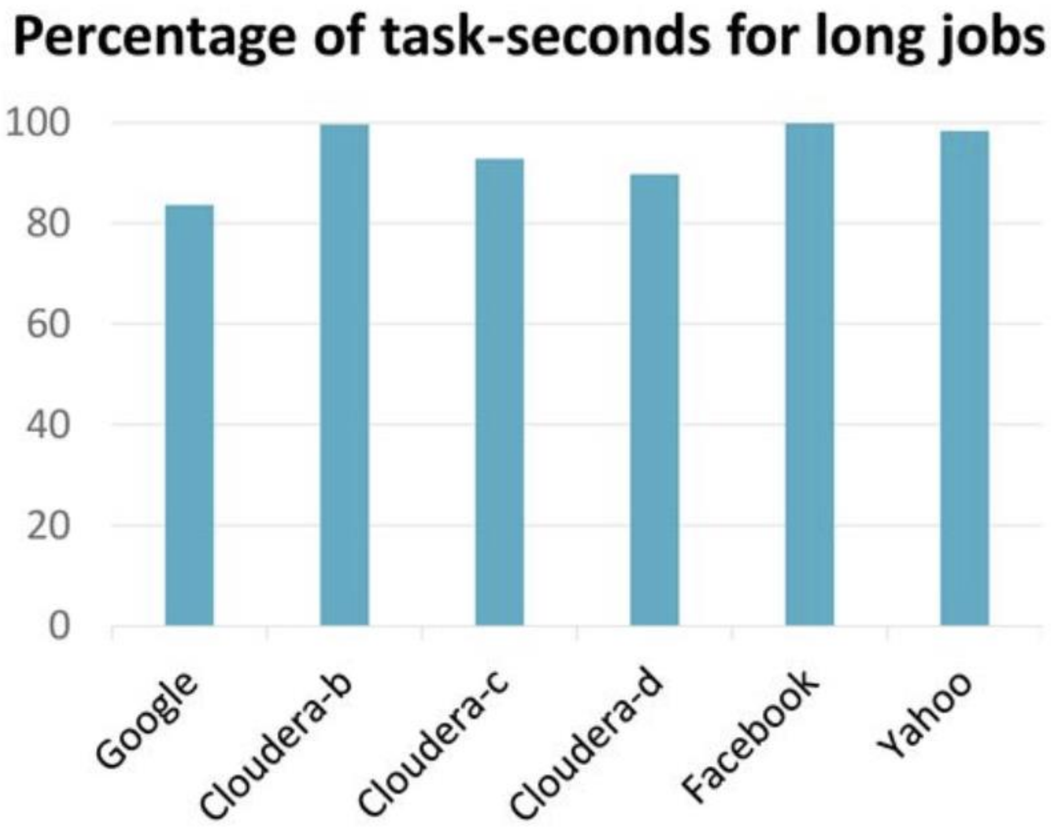




Source: Design Insights for MapReduce from Diverse Production Workloads, Chen et al 2012

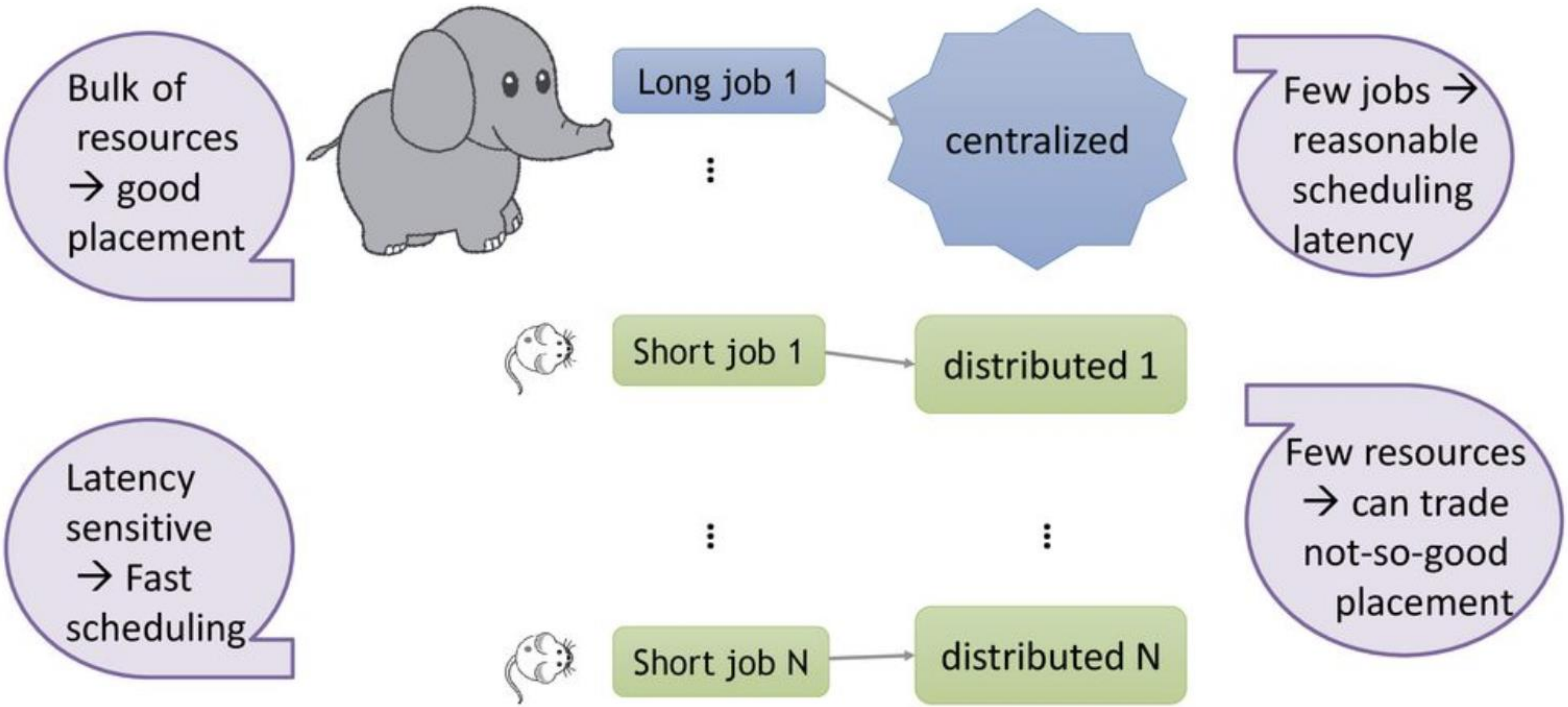


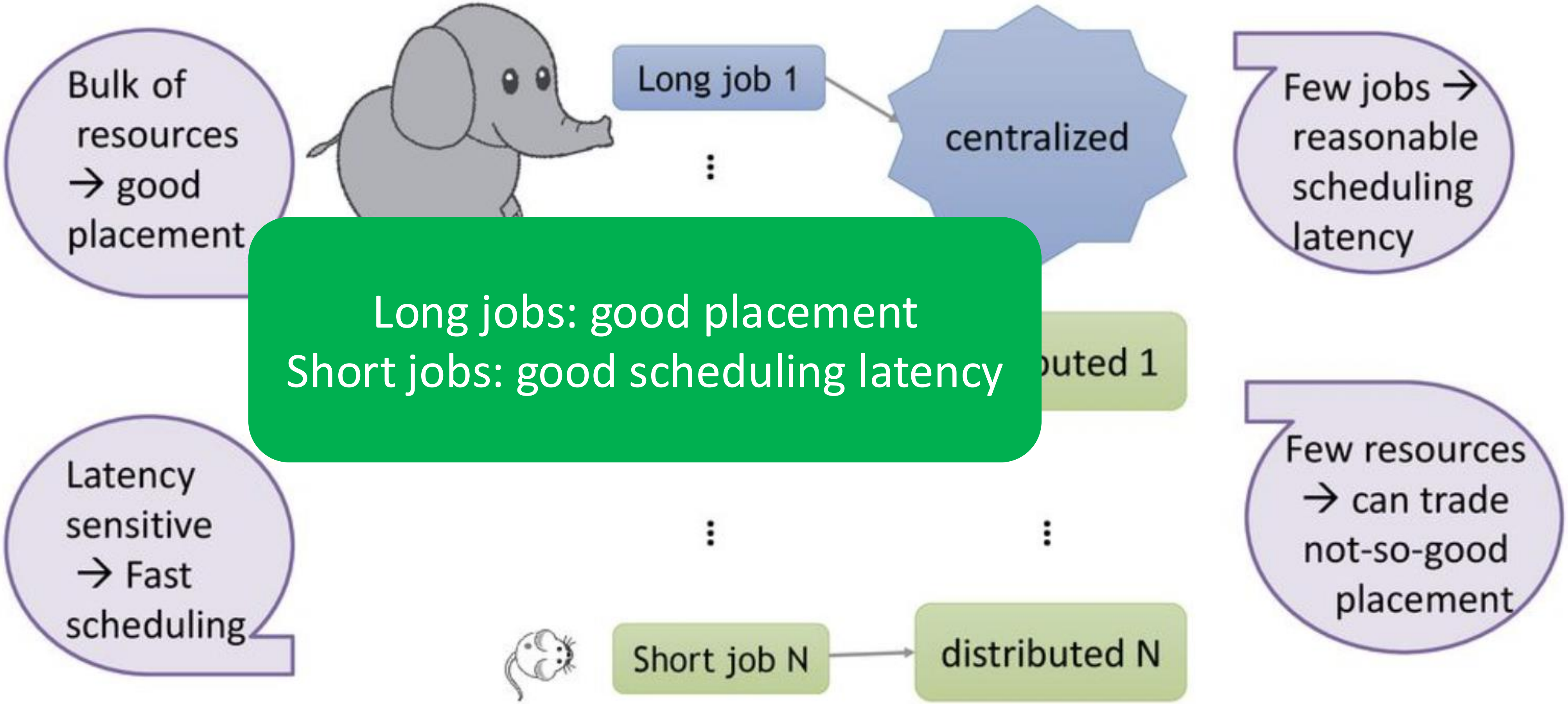
Long jobs: minority but take most of the resources



Source: Design Insights for MapReduce from Diverse Production Workloads, Chen et al 2012

EPFL


Bulk of resources → good placement
Long job 1
centralized
Few jobs → reasonable scheduling latency
Short job 1
distributed 1
Latency sensitive → Fast scheduling
Few resources → can trade not-so-good placement
Short job N
distributed N



Long jobs: good placement
Short jobs: good scheduling latency

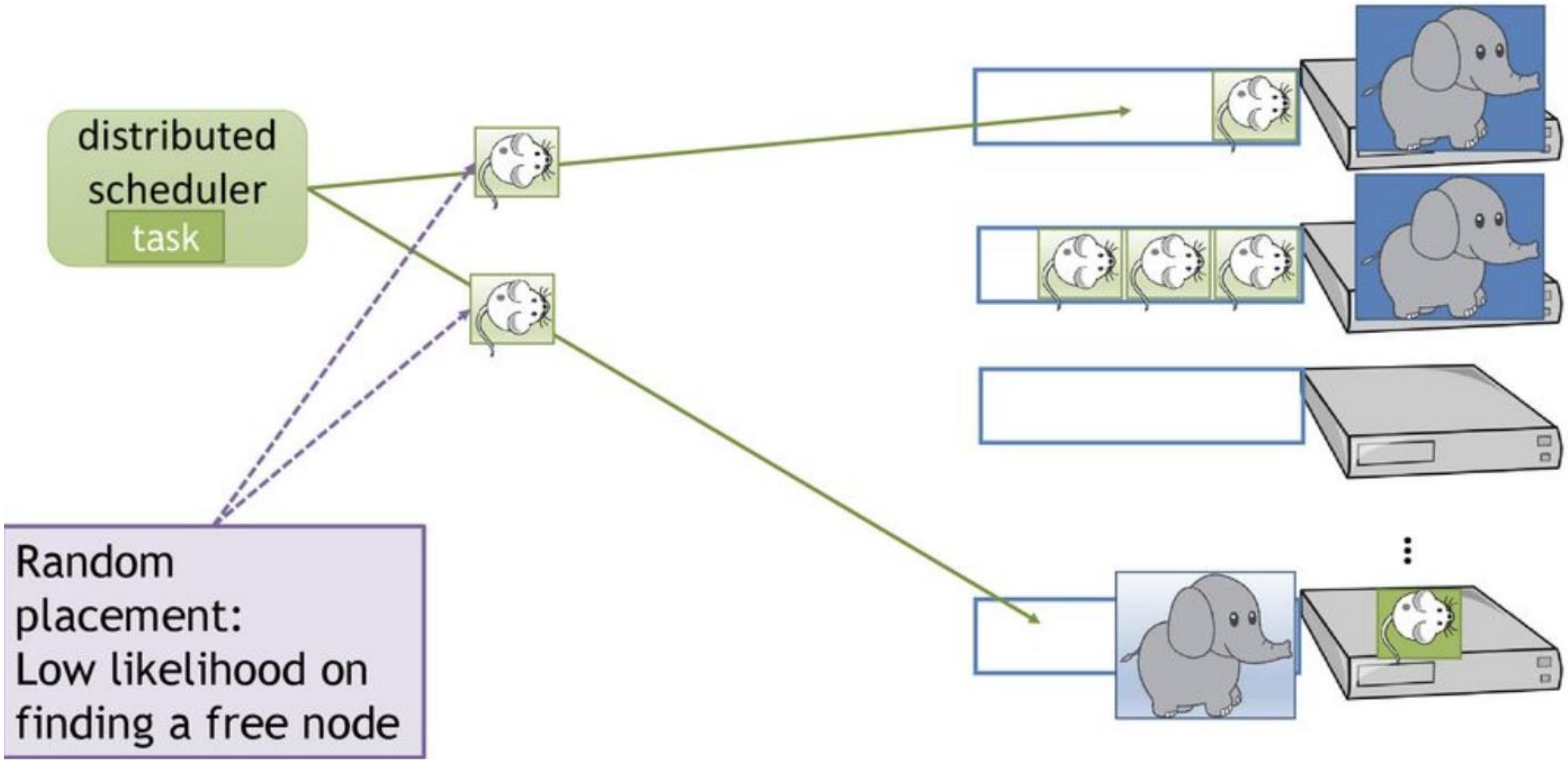
Hawk

- Sparrow: random placement

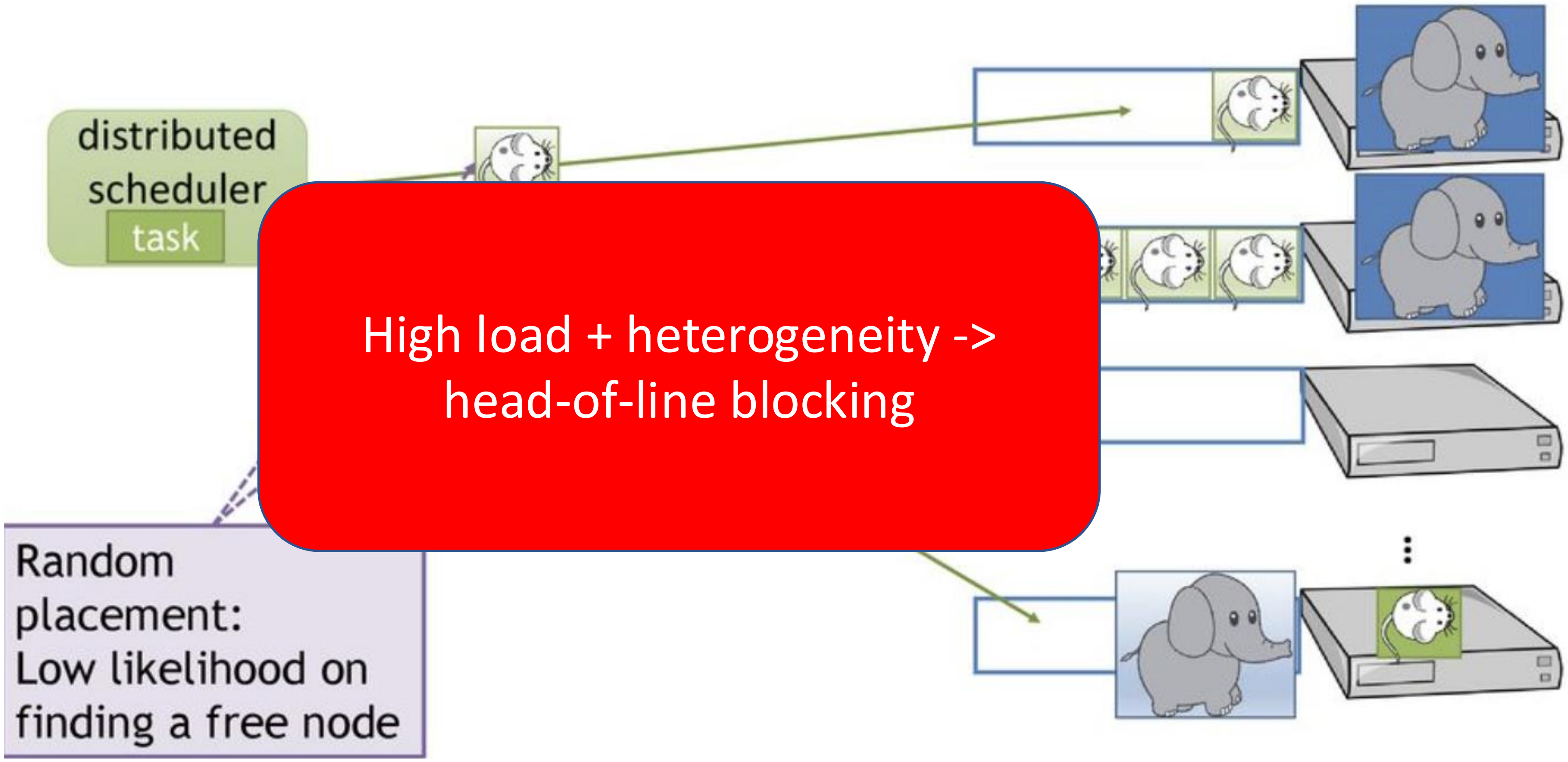
[Sparrow: Distributed, Low Latency Scheduling. Kay Ousterhout, Patrick Wendell, Matei Zaharia, Ion Stoica, University of California, Berkeley, SOSP 2013]

- Randomized work Stealing
- Cluster partitioning

Sparrow

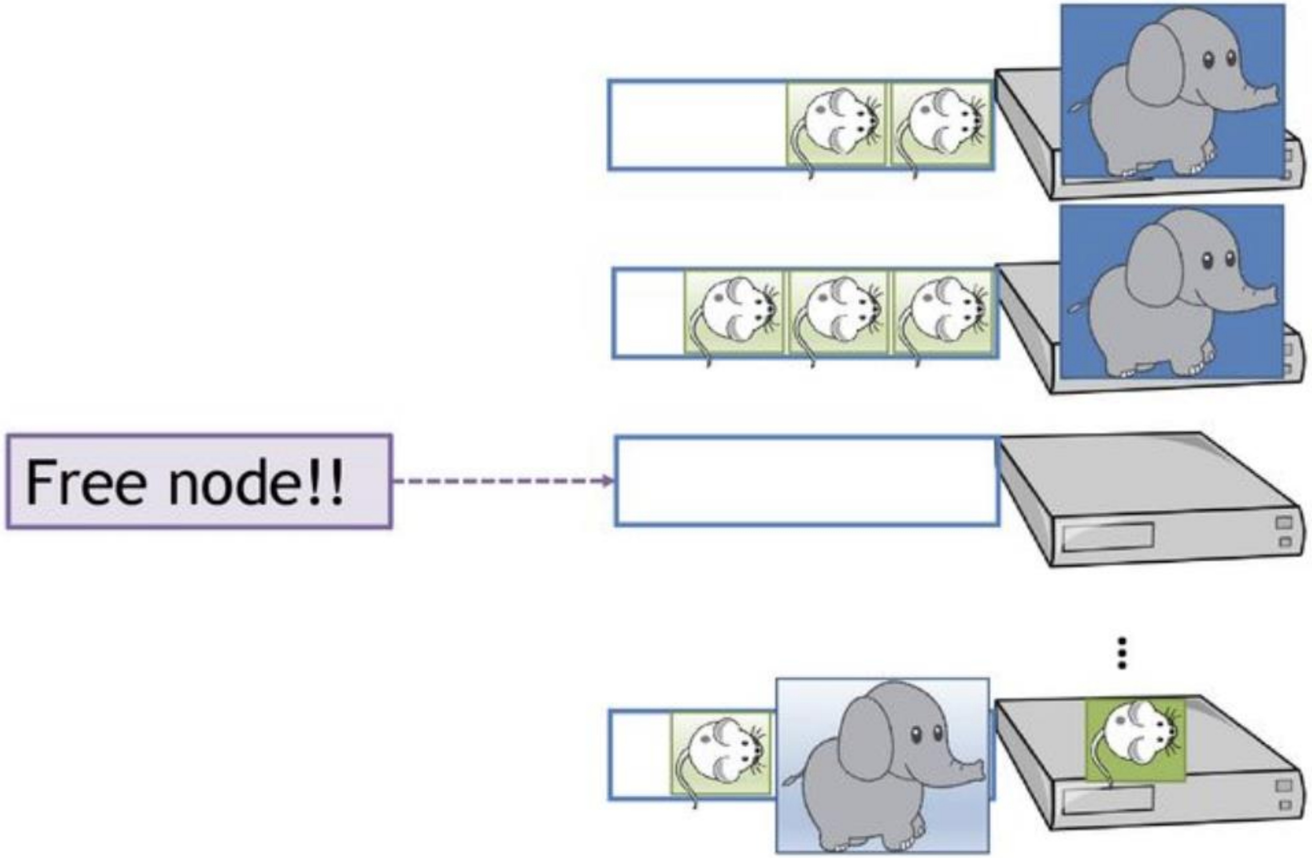

distributed scheduler
task
Random placement:
Low likelihood on finding a free node

Sparrow

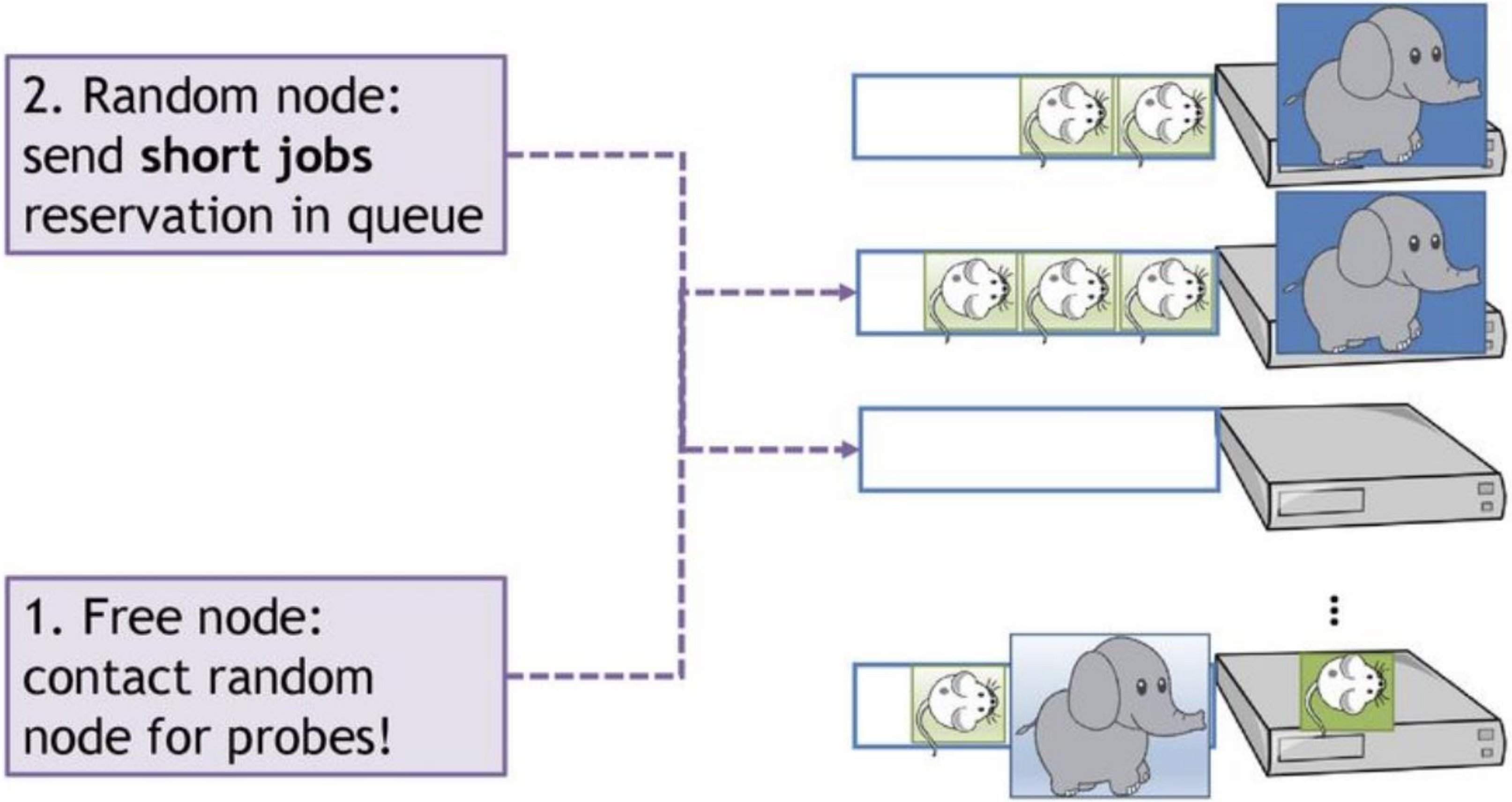


High load + heterogeneity -> head-of-line blocking

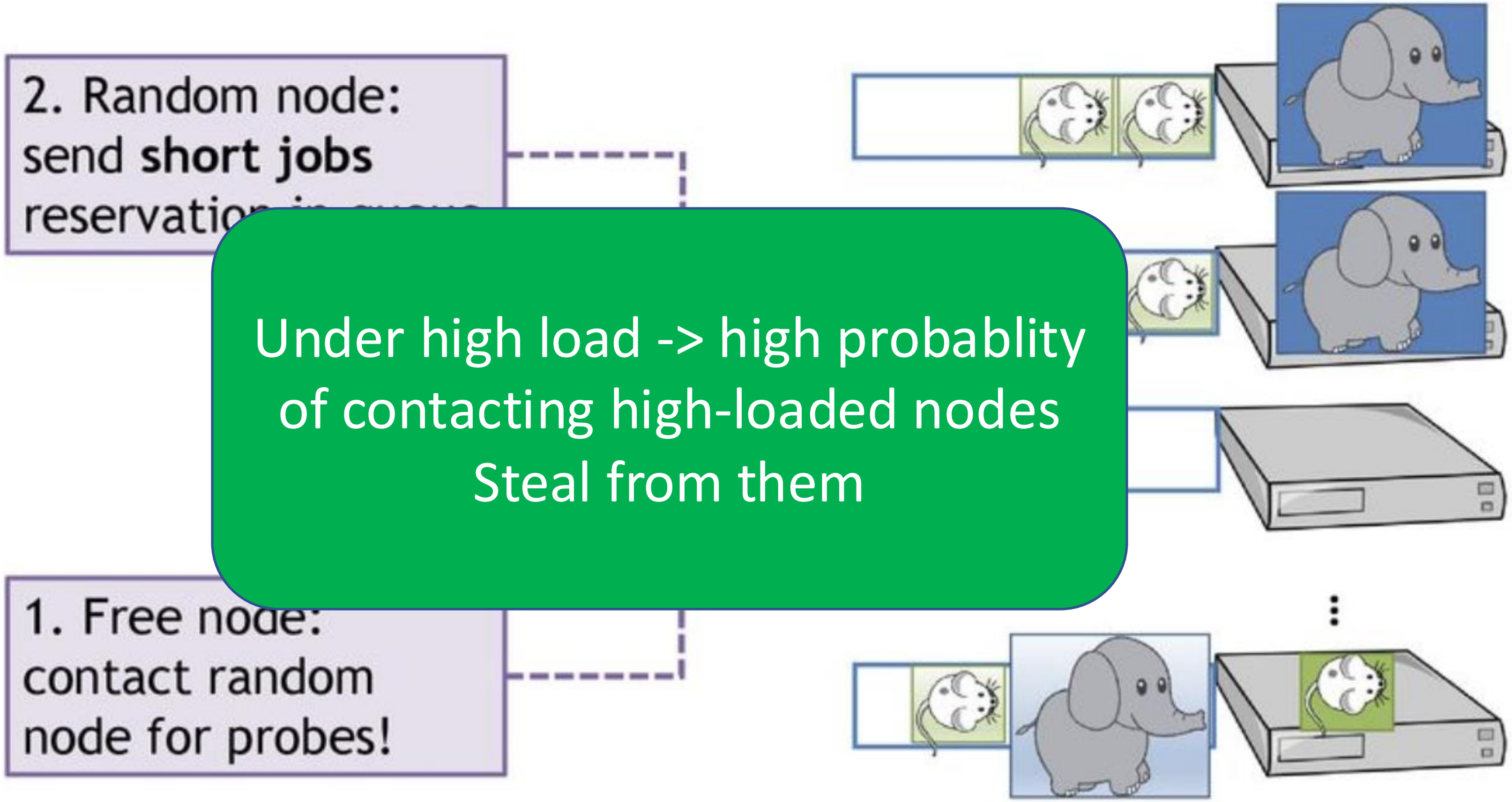
Hawk: work stealing



Hawk: work stealing

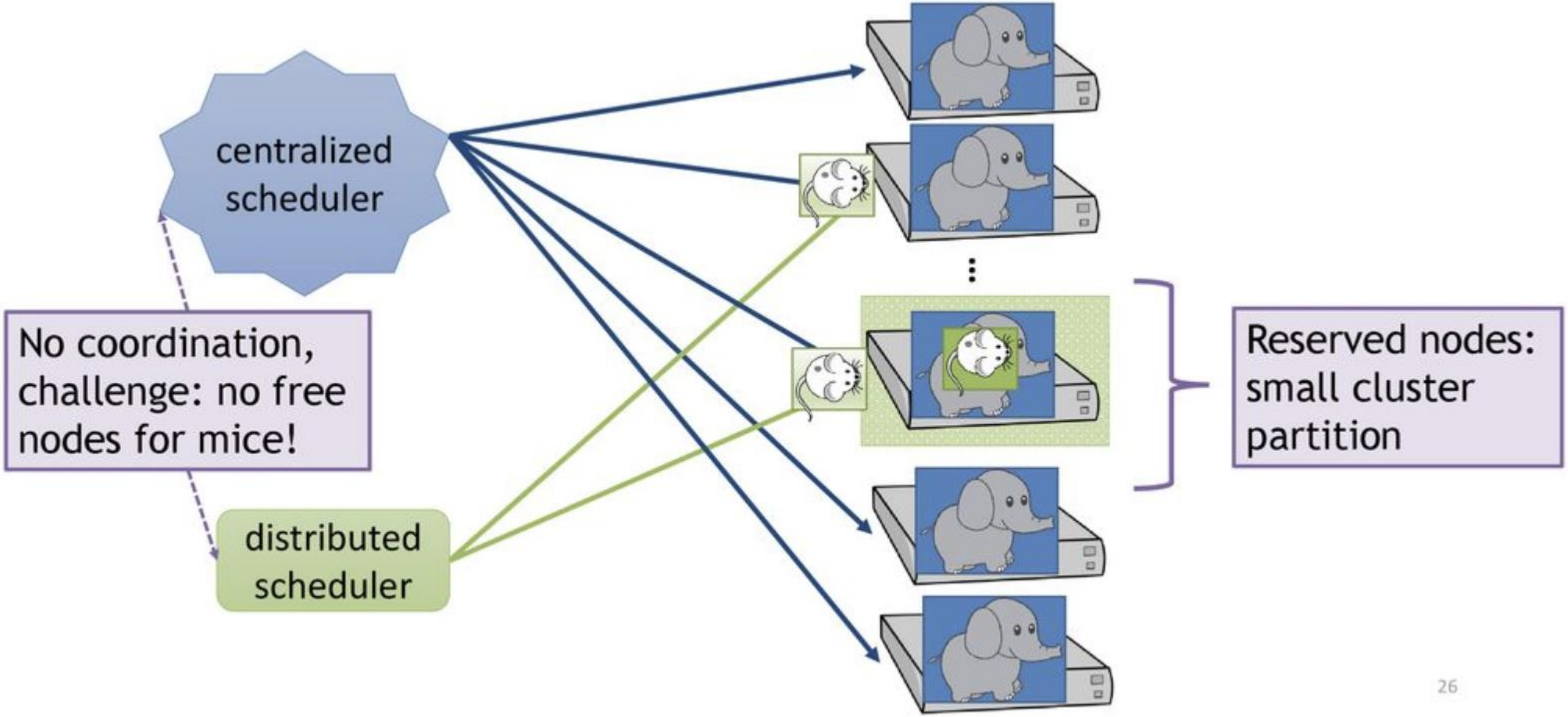


Hawk: work stealing

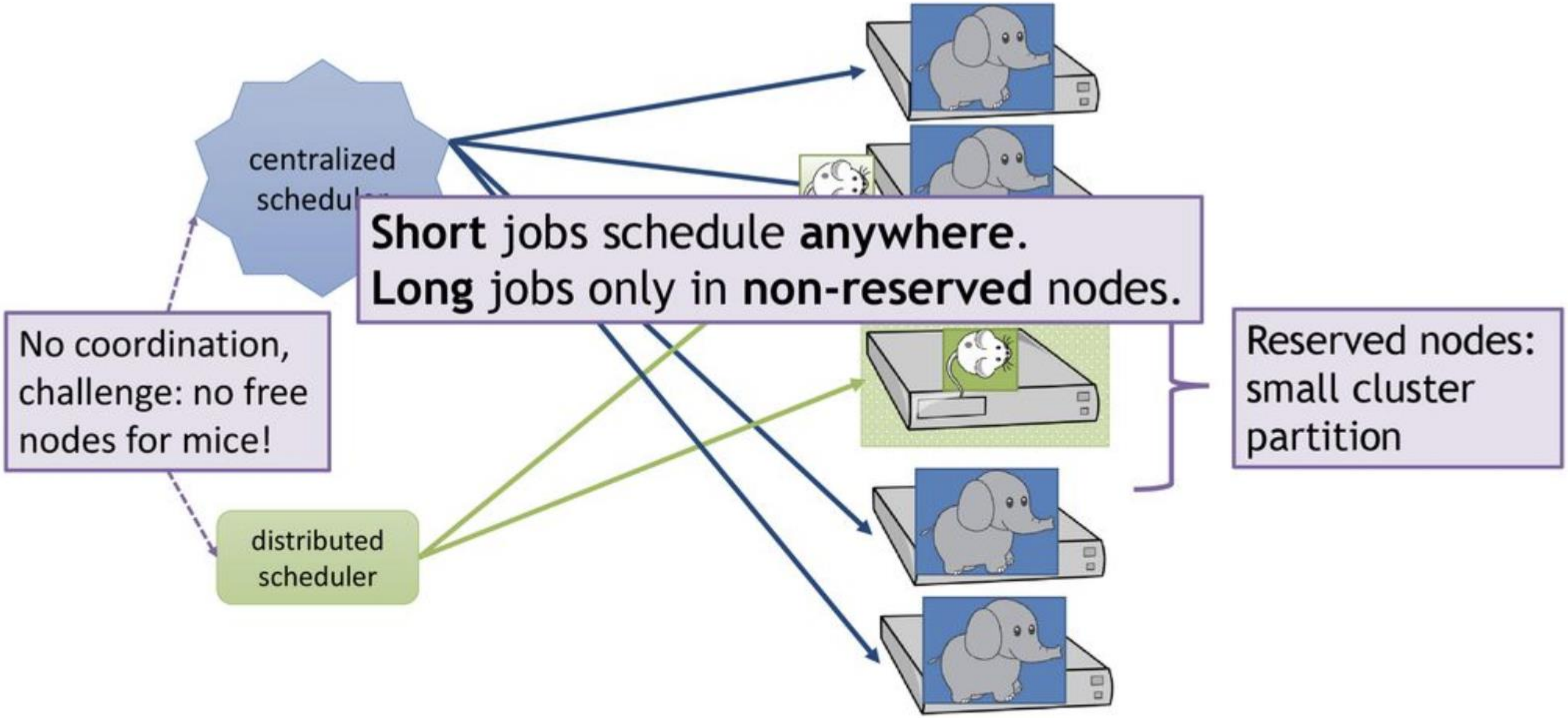


Under high load -> high probablity of contacting high-loaded nodes
Steal from them

Hawk: cluster partitioning



Hawk: cluster partitioning



References

- B. Hindman et al., “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center", USENIX 2011
- A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, I. Stoica. “Dominant Resource Fairness: Fair Allocation of Multiple Resource Types”. NSDI 2011
- V. Vavilapalli et al., “Apache hadoop yarn: Yet another resource negotiator", ACM Cloud Computing 2013
- P Delgado, F Dinu, AM Kermarrec, W Zwaenepoel, “Hawk: Hybrid datacenter scheduling”, USENIX ATC, 2015

Thanks to Indranil Gupta and to Amir H. Payberah