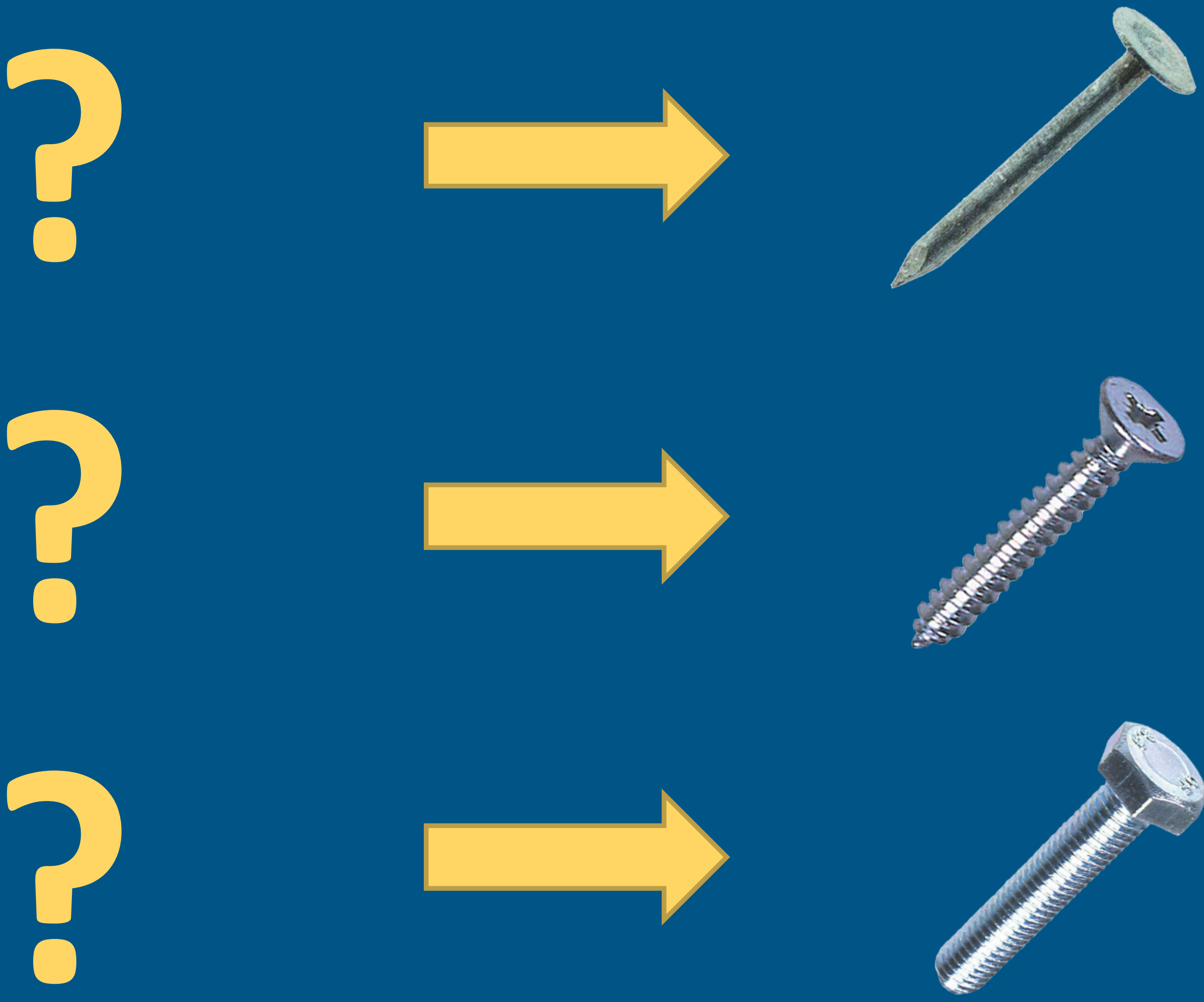
cloudera®

Next-Generation Apache Hadoop

Open problems in distributed storage and resource management

Karthik Kambatla | kasha@cloudera.com

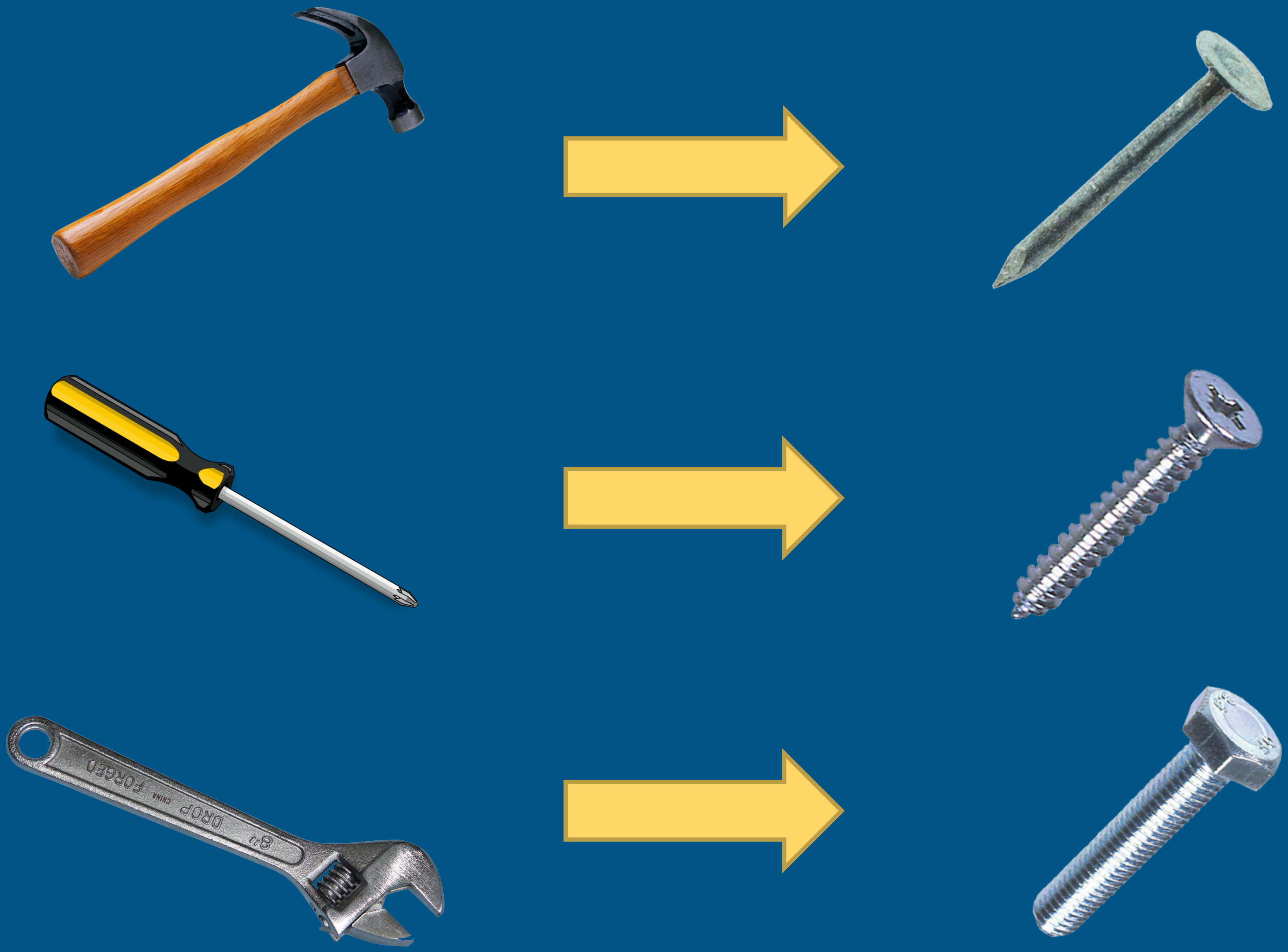
Andrew Wang | andrew.wang@cloudera.com



?

?

?

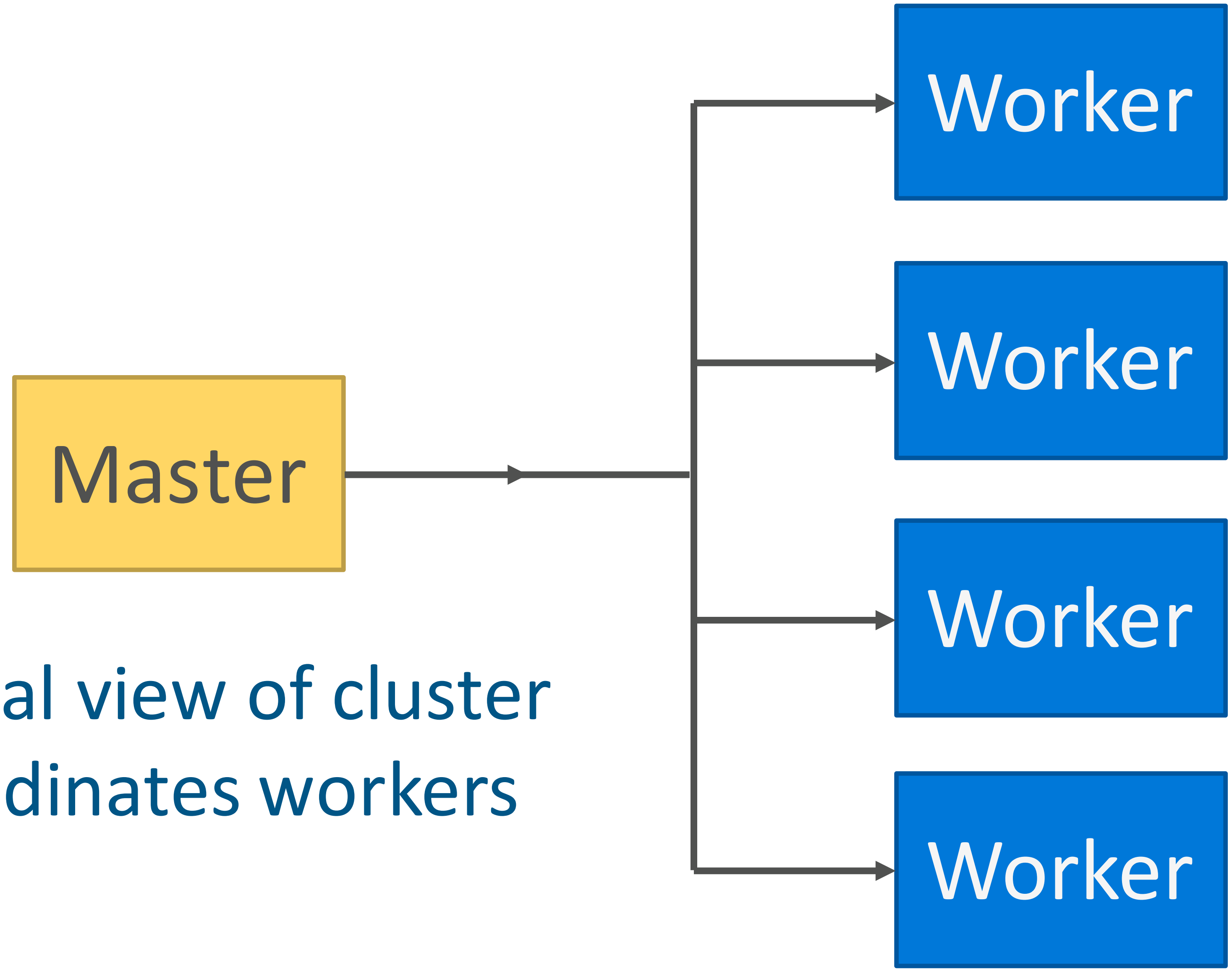

DROP FORGED
CHINA

Cloudera perspective

- Hadoop software stack is relatively mature
- Seen broad uptake in many industries
  - Wider variety of workloads
  - Larger and larger amounts of data
- New datacenter hardware trends on the horizon

- Good time to revisit original design assumptions
- Collaborate with academics on these problems

Scalability



Master

Worker

Worker

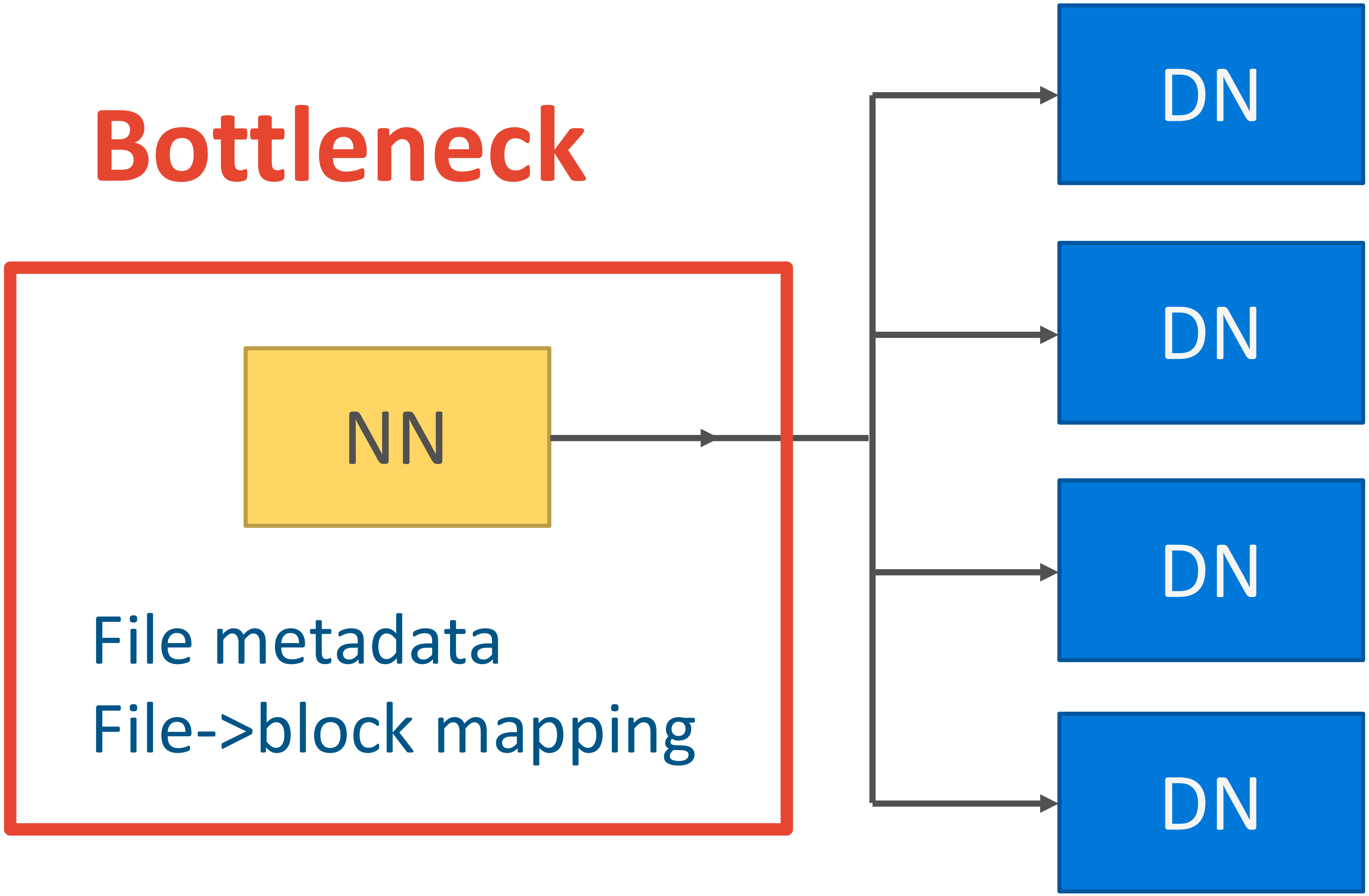
Worker

Worker

Global view of cluster
Coordinates workers

Simple
Do work
Scale-out

Bottleneck



NN

File metadata
File->block mapping

DN

DN

DN

DN

Store blocks
Serve data reads/writes

Vertically scaling HDFS

Project	Improvement	Cost (months)
Multiple volumes per NN	Operational	6
Split namespace and block management locking	2x RPC	12
Fine-grained locking of namespace	2x RPC	6
Pageable namespace	2x object count	6
Persistent block space	Operational	6
Block management as a service	2x object count	12+
Volume migration	Operational	12

Scary changes

Project	Improvement	Cost (months)
Multiple volumes per NN	Operational	6
Split namespace and block management locking	2x RPC	12
Fine-grained locking of namespace	2x RPC	6
Pageable namespace	2x object count	6
Persistent block space	Operational	6
Block management as a service	2x object count	12+
Volume migration	Operational	12

Incremental

Project	Improvement	Cost (months)
Multiple volumes per NN	Operational	6
Split namespace and block management locking	2x RPC	12
Fine-grained locking of namespace	2x RPC	6
Pageable namespace	2x object count	6
Persistent block space	Operational	6
Block management as a service	2x object count	12+
Volume migration	Operational	12

Years of work

Project	Improvement	Cost (months)
Multiple volumes per NN	Operational	6
Split namespace and block management locking	2x RPC	12
Fine-grained locking of namespace	2x RPC	6
Pageable namespace	2x object count	6
Persistent block space	Operational	6
Block management as a service	2x object count	12+
Volume migration	Operational	12

Hardware trends on the horizon

	2006	2016	2021
HDD capacity (TB)	0.2	2	20
HDD speed (MB/s)	90	110	140
Network speed (Gb/s)	0.1	10	40

Fewer IOPS/GB

HDD locality irrelevant

A fresh look

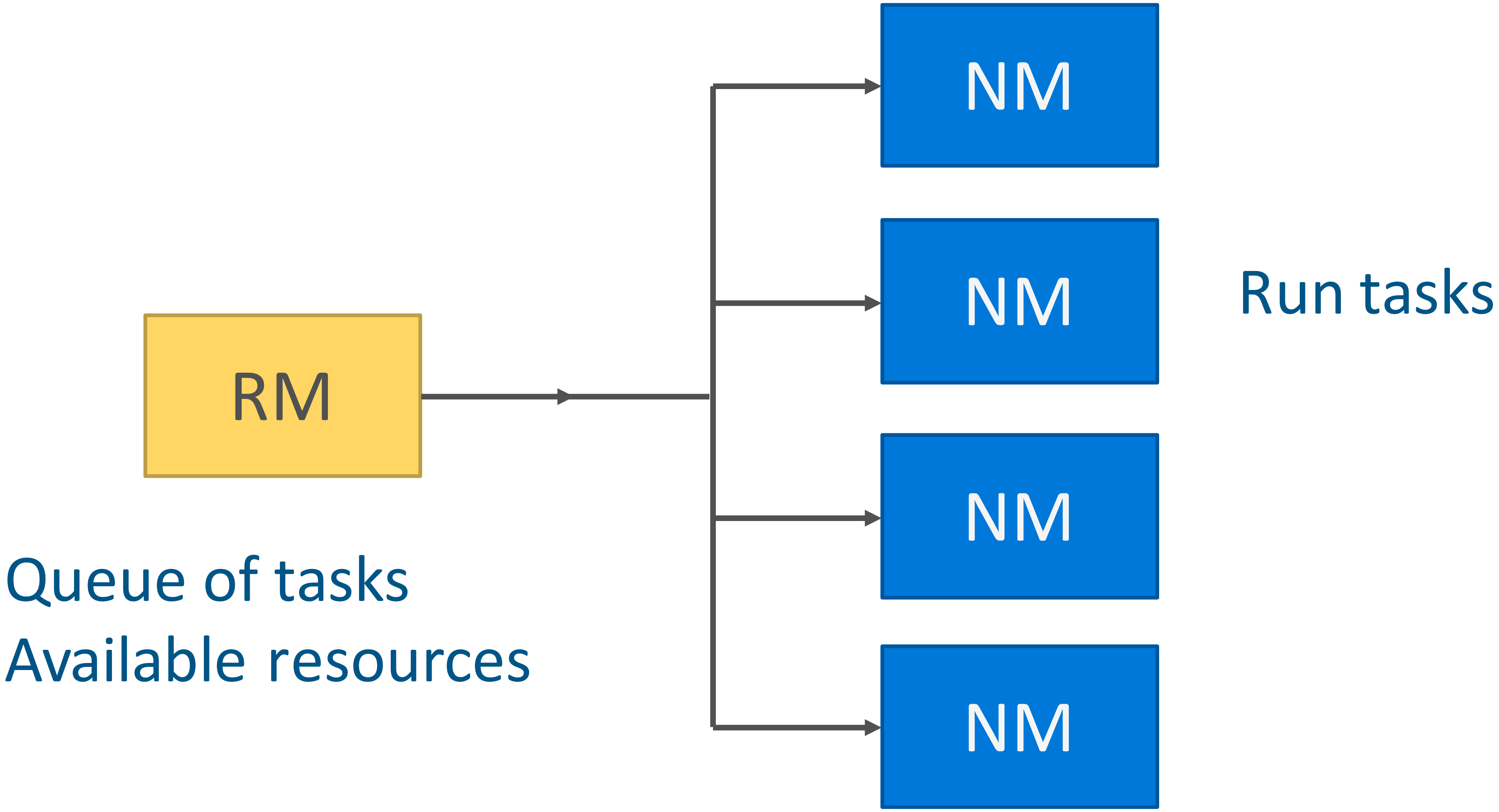
- Designed for analytic workloads
- Scales horizontally (exabyte scale)
- Operationally robust
- Designed for future hardware trends

Blobstore

- Users think in datasets, not directories and files
- Spectrum of blobstore vs. filesystem functionality
- What is the equivalent of the POSIX API for a scalable storage system?
  - What set of operations are required?
  - What are their semantics?
  - What can and cannot be supported scalably?

Other considerations

- Erasure coding
  - Required to be cost competitive
- Multi-datacenter replication
  - Important for business-critical analytics
- 3D Xpoint
  - New addition to storage hierarchy
  - Could change how we write software and think about persistence



Queue of tasks

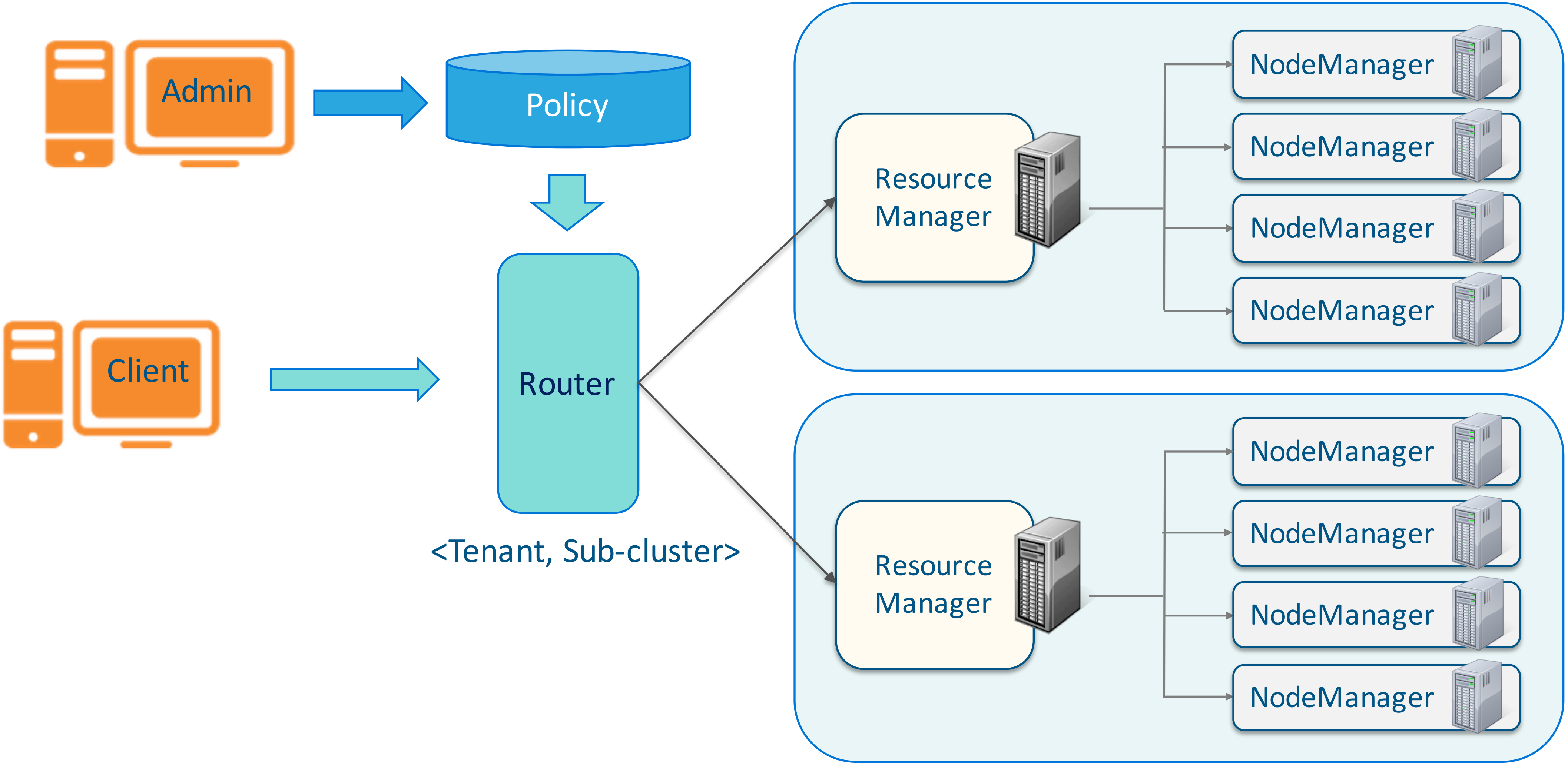
Available resources

Run tasks

One cluster to rule them all

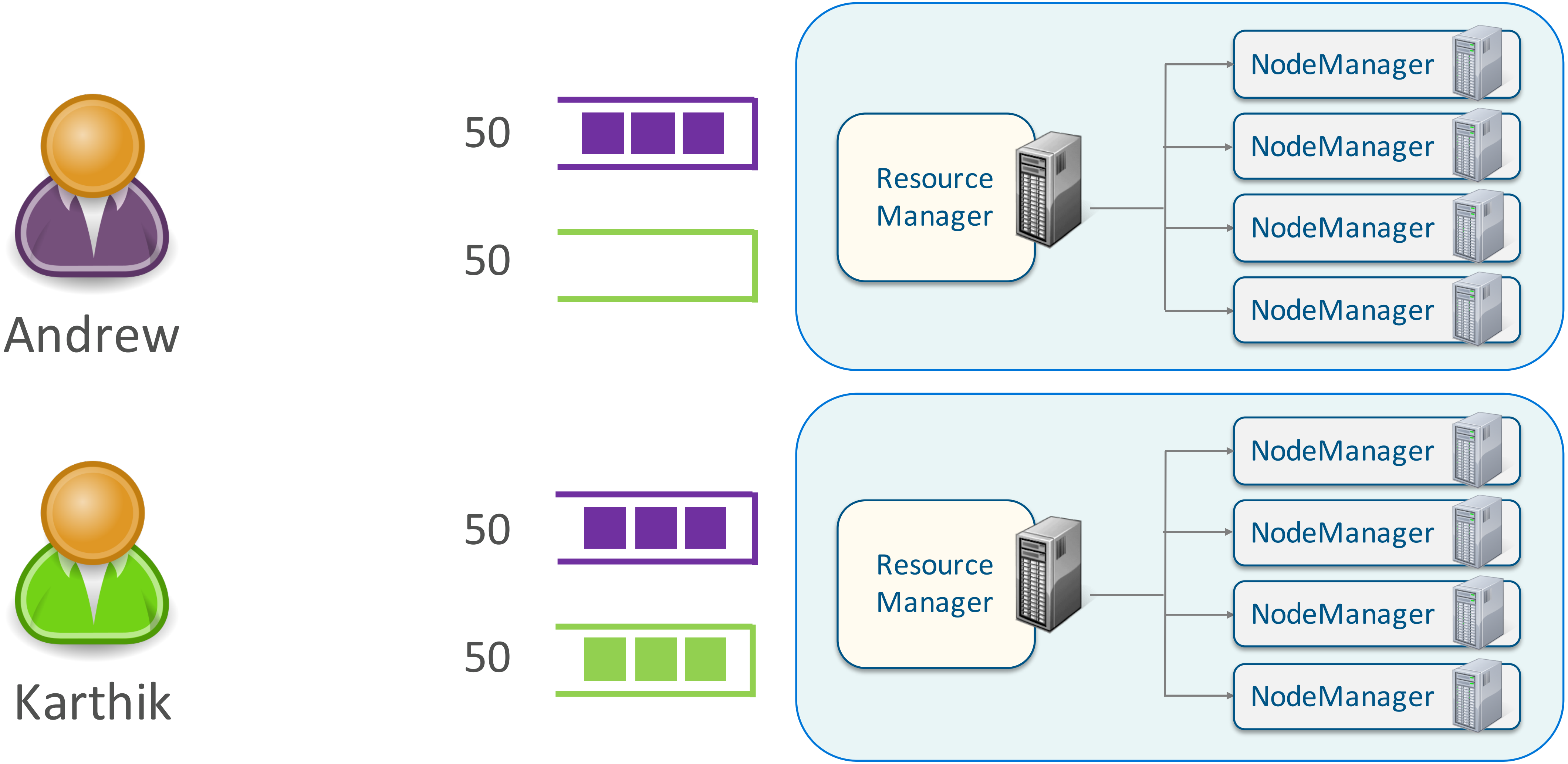
- Exabyte-scale storage means exabyte-scale processing
- Current: 10,000 node YARN clusters
- Goal: 1,000,000 nodes
    - One cluster for all compute at an internet-scale company
    - Think Microsoft or Twitter

Yarn Federation

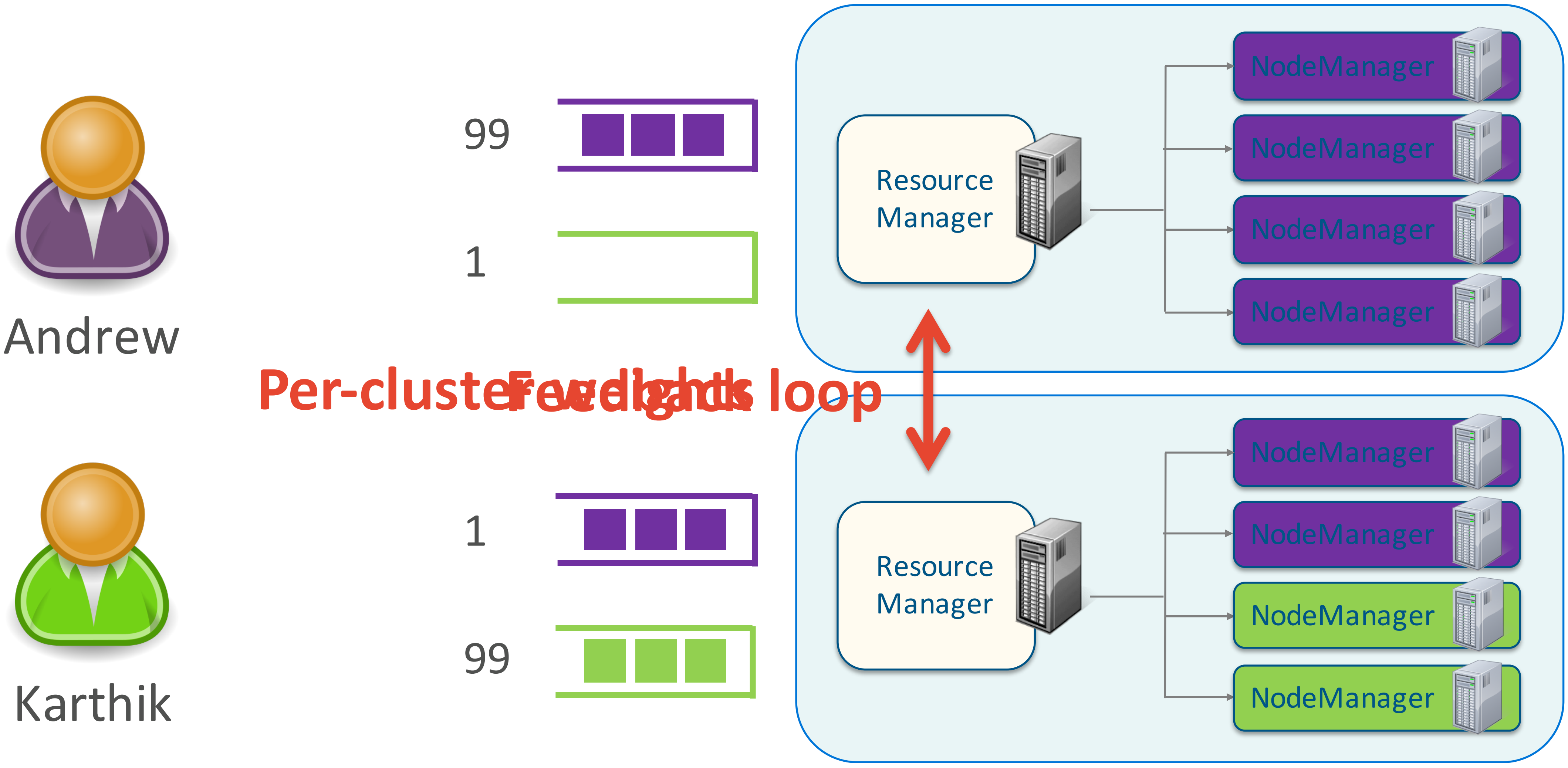


cloudera

Fair-Sharing and Federation



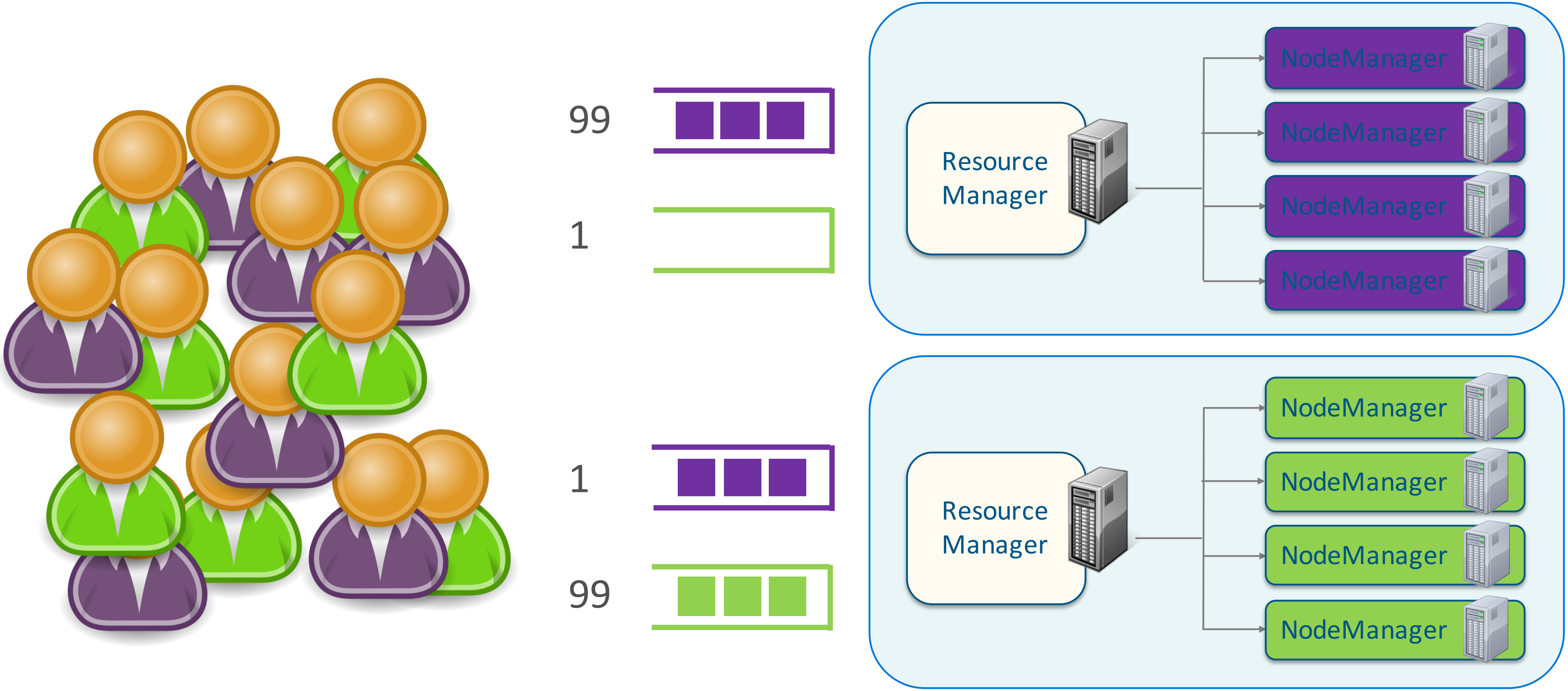
Fair-Sharing and Federation



Per-cluster weights

Feedback loop

Fair-Sharing and Federation



Scheduling

Variety of workloads

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Batch processing	Mins - hours	Seconds	**< 400,000**	**Jobs (10Ks)**	Low
Interactive SQL	Seconds	Milliseconds	100s	Users (100s)	Medium
Stream processing	Months	Minutes	10s	Jobs (10s)	High
Long-running services	Months	Minutes	# Nodes	Services (10s)	High

cloudera

Scheduling latency

	Duration	Scheduling Latency	"Tasks"	Tenant Scale	Placement Quality
Batch processing	Mins - hours	**Seconds**	< 400,000	Jobs (10Ks)	Low
Interactive SQL	Seconds	**Milliseconds**	100s	Users (100s)	Medium
Stream processing	Months	**Minutes**	10s	Jobs (10s)	High
Long-running services	Months	**Minutes**	# Nodes	Services (10s)	High

cloudera

Low latency scheduling for distributed systems

- State of the art
  - Low-latency scheduling: Sparrow
    - Second-level scheduler that needs pre-allocated resources
  - Operational
    - Static partitioning: set aside resources
    - Semi-static: Maintain a per-user cache of resources
    - Downside: low utilization

Can we design scalable algorithms for low-latency scheduling?

Scheduling latency

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Batch processing	Mins - hours	**Seconds**	< 400,000	Jobs (10Ks)	Low
Interactive SQL	Seconds	**Milliseconds**	100s	Users (100s)	Medium
Stream processing	Months	**Minutes**	10s	Jobs (10s)	High
Long-running services	Months	**Minutes**	# Nodes	Services (10s)	High

Scheduling latency

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Batch processing	Mins - hours	**Seconds**	< 400,000	Jobs (10Ks)	Low
Interactive SQL	Seconds	**Minutes**	100s	Users (100s)	Medium
Stream processing	Months	**Minutes**	10s	Jobs (10s)	High
Long-running services	Months	**Minutes**	# Nodes	Services (10s)	High

Jobs vs Services

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Batch processing	Mins - hours	**Seconds**	**< 400,000**	**Jobs (10Ks)**	**Low**
Interactive SQL	Seconds	**Minutes**	**100s**	**Users (100s)**	**Medium**
Stream processing	Months	**Minutes**	**10s**	**Jobs (10s)**	**High**
Long-running services	Months	**Minutes**	**# Nodes**	**Services (10s)**	**High**

Jobs vs Services

	Duration	Scheduling Latency	"Tasks"	Tenant Scale	Placement Quality
Jobs	Mins - hours	**Seconds**	**< 400,000**	**~ 10,000**	**Low**
Services	Seconds	**Minutes**	**#Nodes**	**< 100**	**High**

Scalability - Tenants

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Jobs	Mins - hours	**Seconds**	**< 400,000**	**~ 10,000**	**Low**
Services	Seconds	**Minutes**	**#Nodes**	**< 100**	**High**

cloudera

Scalability – Tenants vs Nodes

- Scheduling is allocating resources for tenants on cluster nodes
  - Matching/join between two sets
  - Scheduling latency = |Tenants| x |Nodes|

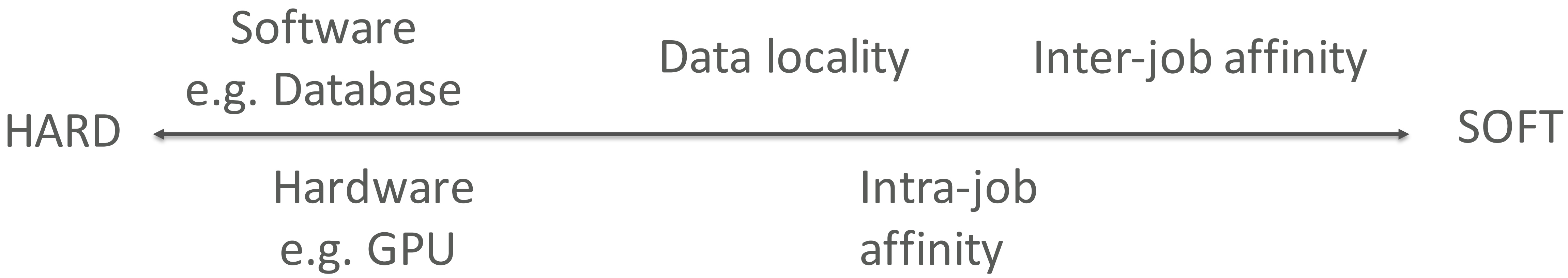
Can we lower the bound on scheduling latency?

Quality of placement

	Duration	Scheduling Latency	“Tasks”	Tenant Scale	Placement Quality
Jobs	Mins - hours	**Seconds**	**< 400,000**	**~ 10,000**	**Low**
Services	Seconds	**Minutes**	**#Nodes**	**< 100**	**High**

cloudera

Placement requirements



HARD

Software
e.g. Database

Hardware
e.g. GPU

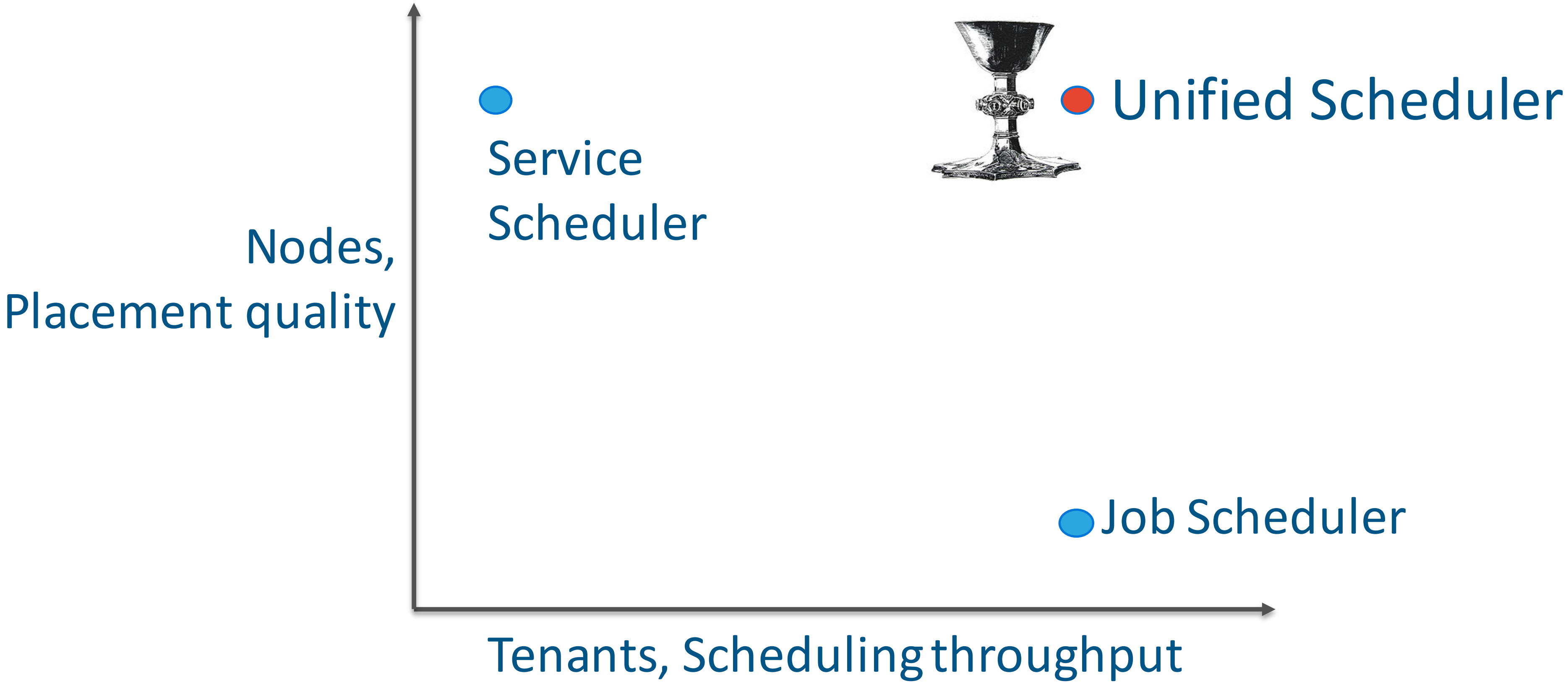
Data locality

Intra-job
affinity

Inter-job affinity

SOFT

Multi-tenancy and scalability



Utilization

Production clusters

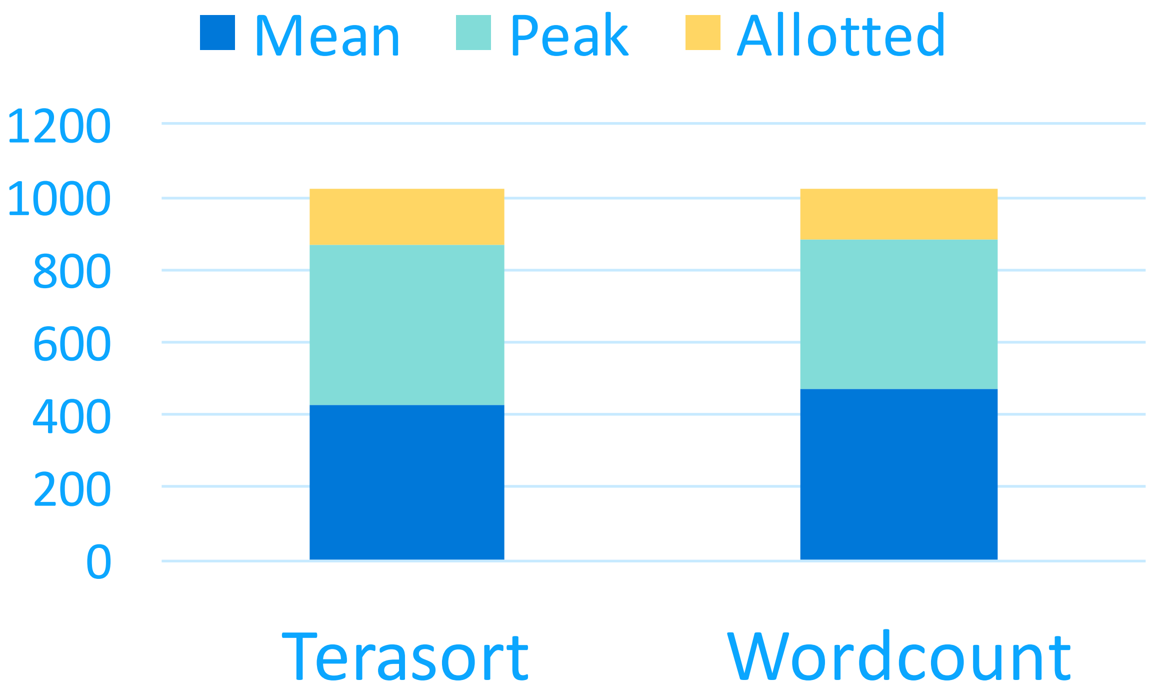
	CPU Utilization %	Memory Utilization %
MapReduce v1	< 20 [1]	< 20 [1]
YARN / MapReduce v2	50 [1]	30 [2]

[1] Apache YARN at SOCC ‘13

[2] Anecdotal from the community

Potential for improvement

- A task's resource usage varies over time.
- Resource usage varies across tasks of the same job



Over-subscribing nodes

- Allocate unused resources to pending tasks
- Challenges
  - Handle sudden spikes in resource usage gracefully
  - Performance of tasks can not deteriorate
    - Contention on non-isolated resources

cloudera

Conclusion

Apache Hadoop is mature and very widely deployed.

The underlying assumptions are 10 years old and need revisiting.

Lots of interesting and hard research problems in the space.

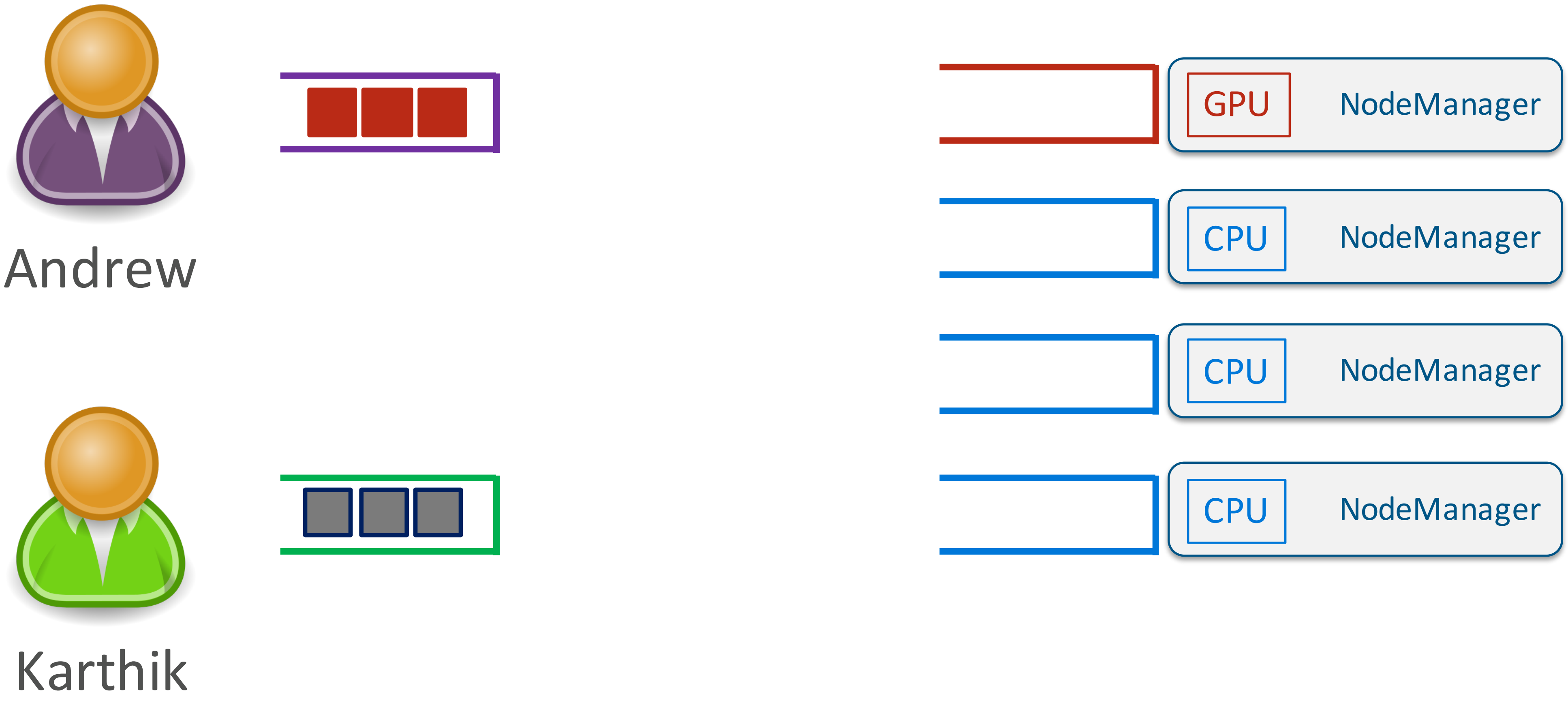
cloudera

Thank you

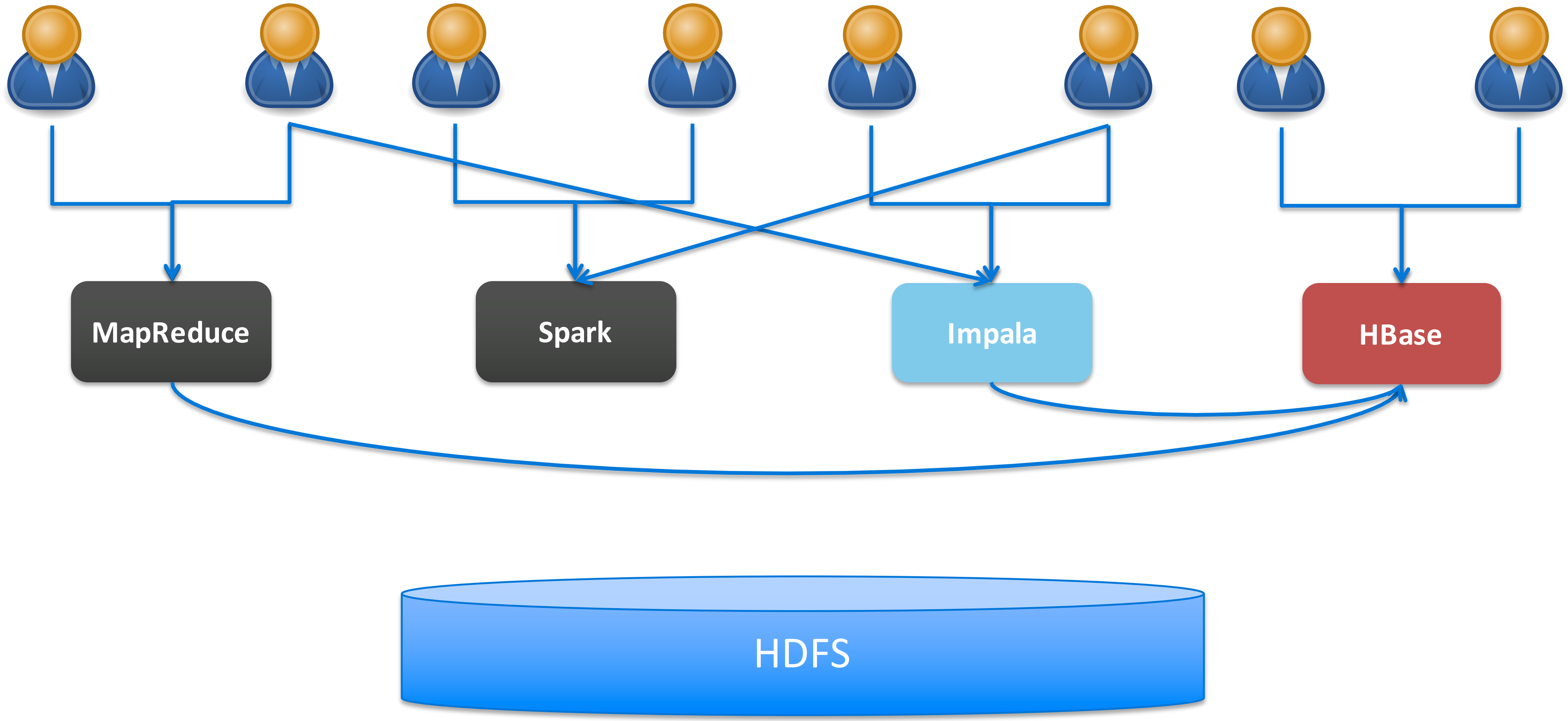
Open Problems

- Storage scalability
- Blobstore API for analytic workloads
- Global fairness in a federated YARN cluster
- Low-latency scheduling
- Jobs and services on the same cluster
    - Scheduler scalability in tenants and nodes
    - Improving quality of placement with a latency upper bound
- Cluster utilization improvements
- I/O scheduling for predictability and QoS

Greedy placement is not optimal



Multi-tenancy


MapReduce
Spark
Impala
HBase
HDFS