Rich placement constraints: Who said YARN cannot schedule services?

Konstantinos Karanasos, Wangda Tan

Dataworks Summit 2018

San Jose, June 20

Intros

- Konstantinos Karanasos
  Apache Hadoop PMC
  Senior Scientist at Microsoft

- Wangda Tan
  Apache Hadoop PMC
  Staff Software Engineer at Hortonworks

Agenda

- Support for long-running applications/services on YARN
- Scheduling with constraints
- Demo

Support for long-running applications / services in YARN

Long-Running Applications and Services (LRAs)

Diverse applications in compute clusters

> Short-running containers
  - MapReduce, Scope, Tez

Shift towards long running containers

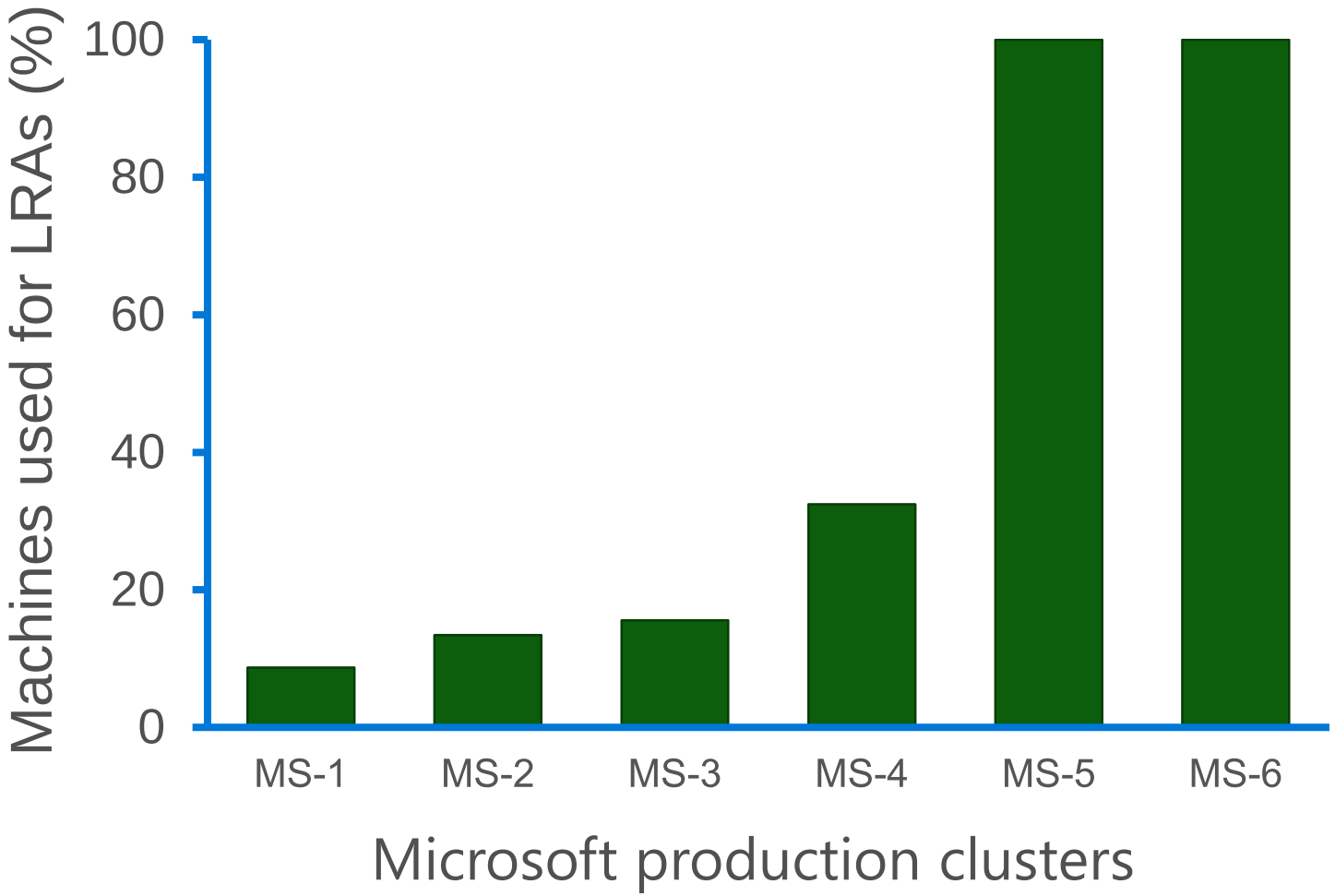
> Interactive data-intensive applications
  - Spark, Hive LLAP

> Streaming systems
  - Flink, Storm, Kafka Streams

> Latency-sensitive applications
  - HBase, Memcached

> ML frameworks
  - TensorFlow, Spark ML

LRAs in production analytics clusters



Microsoft:

> Each cluster comprises tens of thousands of machines

> 10-100% of each cluster's machines used for LRAs

Hortonworks:

> Internal clusters dedicated to LRAs

LRAs vs. classic batch jobs

Important

- Runs longer :D
- Uptime is important
- In-place upgrade
- Service discovery
- Dependency management
- Flexible deployment models
- Placement is important

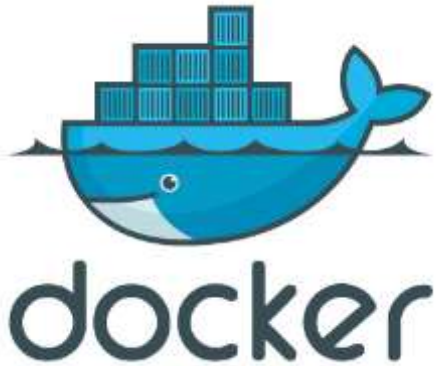
Less important

- Scheduling latency
- Container launch latency

So we introduced YARN service framework (Apache Hadoop 3.1.0)

Addressing these requirements

- Dependency management



  - **Lightweight** mechanism for packaging / isolation
  - Popularized and made accessible by Docker.
  - Can replace VMs in some cases

- Service discovery
  - DNS
- Flexible deployment models
  - Provide Easy-to-use spec
  - Let's look at spec

YARN service framework.

- Sample service spec (Yarnfile)

Docker →

DNS →

```
{
    "name": "tf-zeppelin-service",
    "version": "1.0.0",
    "components": [
        {
            "name": "tf-zeppelin",
            "number_of_containers": 1,
            "artifact": {
                "id": "wtan/zeppelin-tf-1.8.0-gpu:0.0.1",
                "type": "DOCKER"
            },
            "launch_command": "/zeppelin/bin/zeppelin.sh",
            "resource": {
                "cpus": 4,
                "memory": "16384",
                "additional": {
                    "yarn.io/gpu": {
                        "value": 2
                    }
                }
            }
        }
    ],
    "quicklinks": {
        "Tensorflow Zeppelin UI": "http://tf-zeppelin-0.${SERVICE_NAME}.${USER}.${DOMAIN}:8080"
    }
}
```

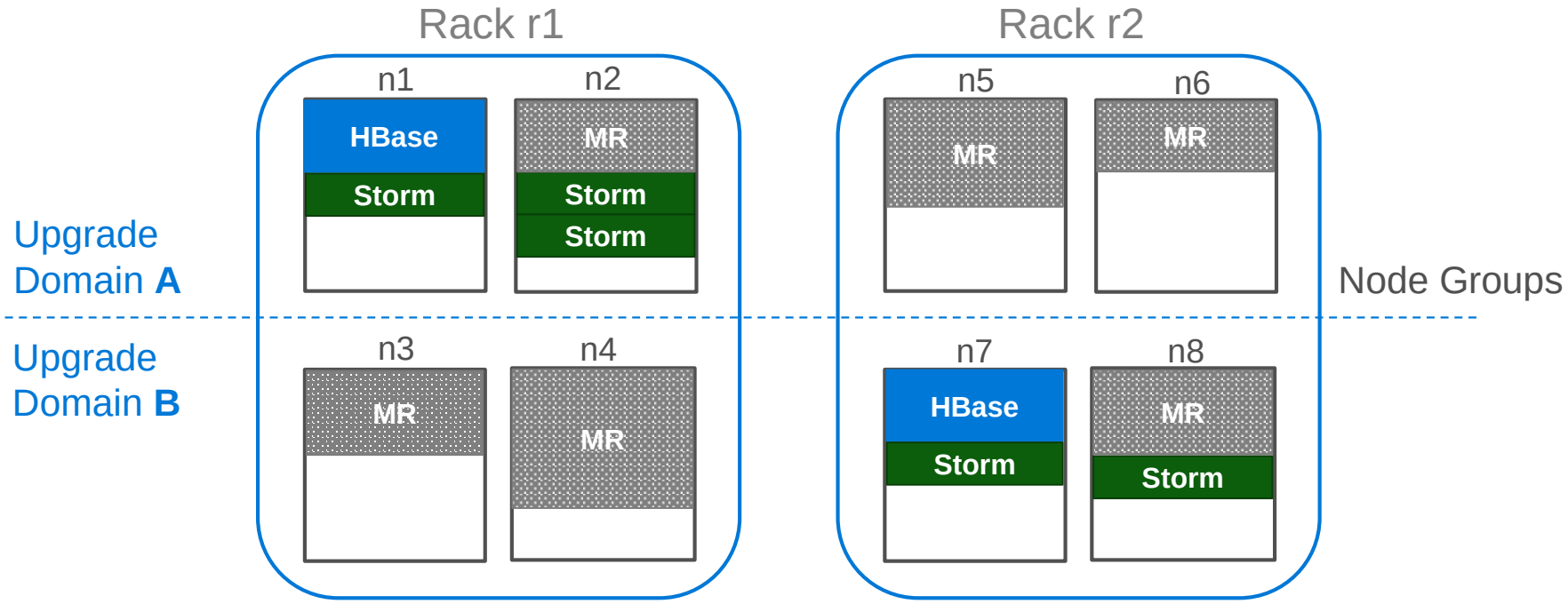
YARN service framework – Placement Policy

***Coming after Hadoop 3.2.0**

```
{
    "name": "hello-world",
    "version": "1.0.0",
    "description": "hello world example with anti-affinity",
    "components": [
        {
            "name": "hello",
            "number_of_containers": 3,
            "launch_command": "./start_nginx.sh",
            "placement_policy": {
                "constraints": [
                    {
                        "type": "ANTI_AFFINITY",
                        "scope": "NODE",
                        "node_attributes": {
                            "os": [
                                "centos6",
                                "centos7"
                            ],
                            "fault_domain": [
                                "fd1",
                                "fd2"
                            ]
                        },
                        "node_partitions": [
                            "gpu"
                        ]
                    }
                ]
            }
        }
```

Scheduling LRAs: Placement with constraints

LRA placement: example



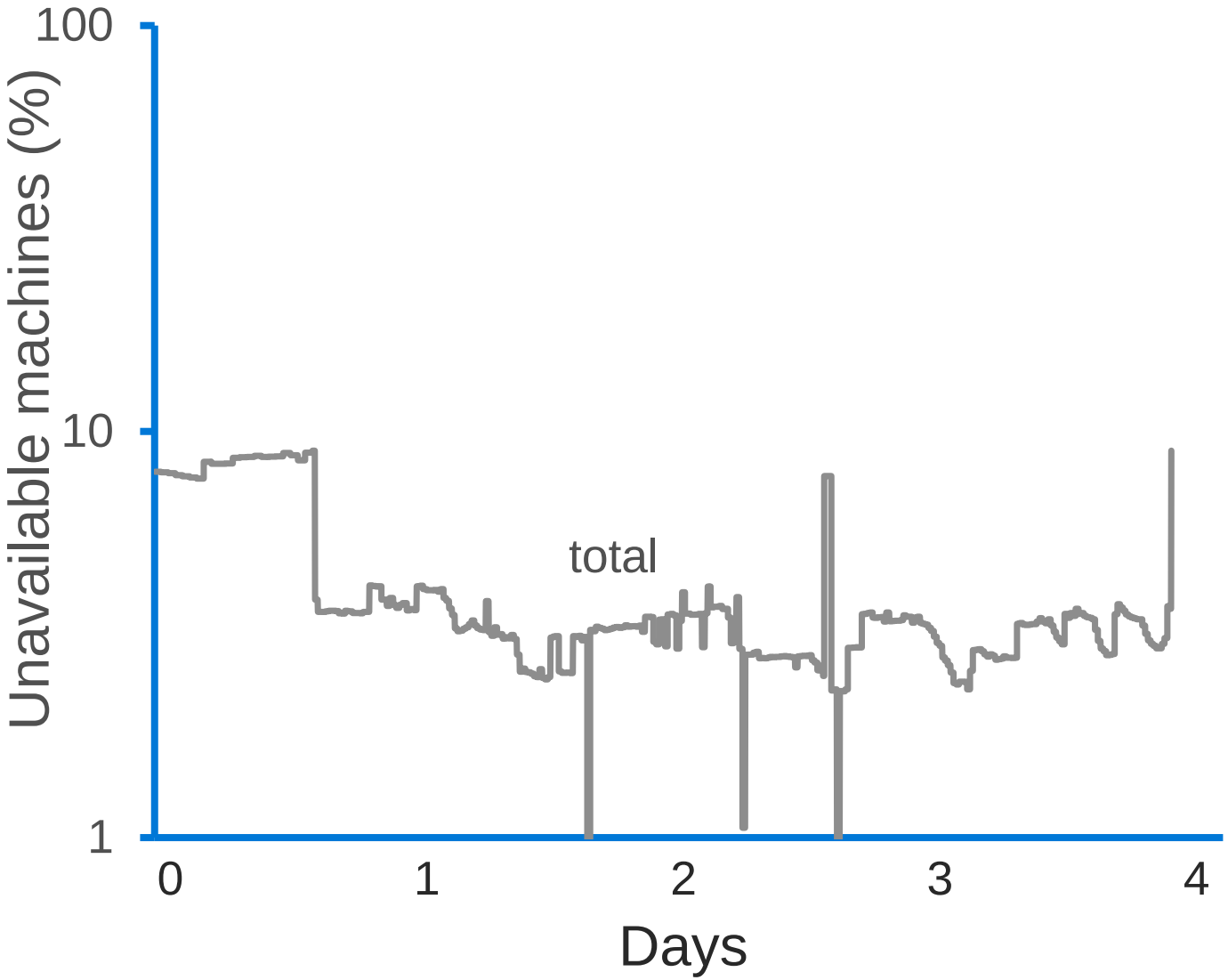
> Performance:
"Place Storm containers on the same rack as HBase"
Such constraints can improve TensorFlow performance by 40+%

> Resilience:
"Place HBase containers across upgrade domains"

> Cluster
"Reduce

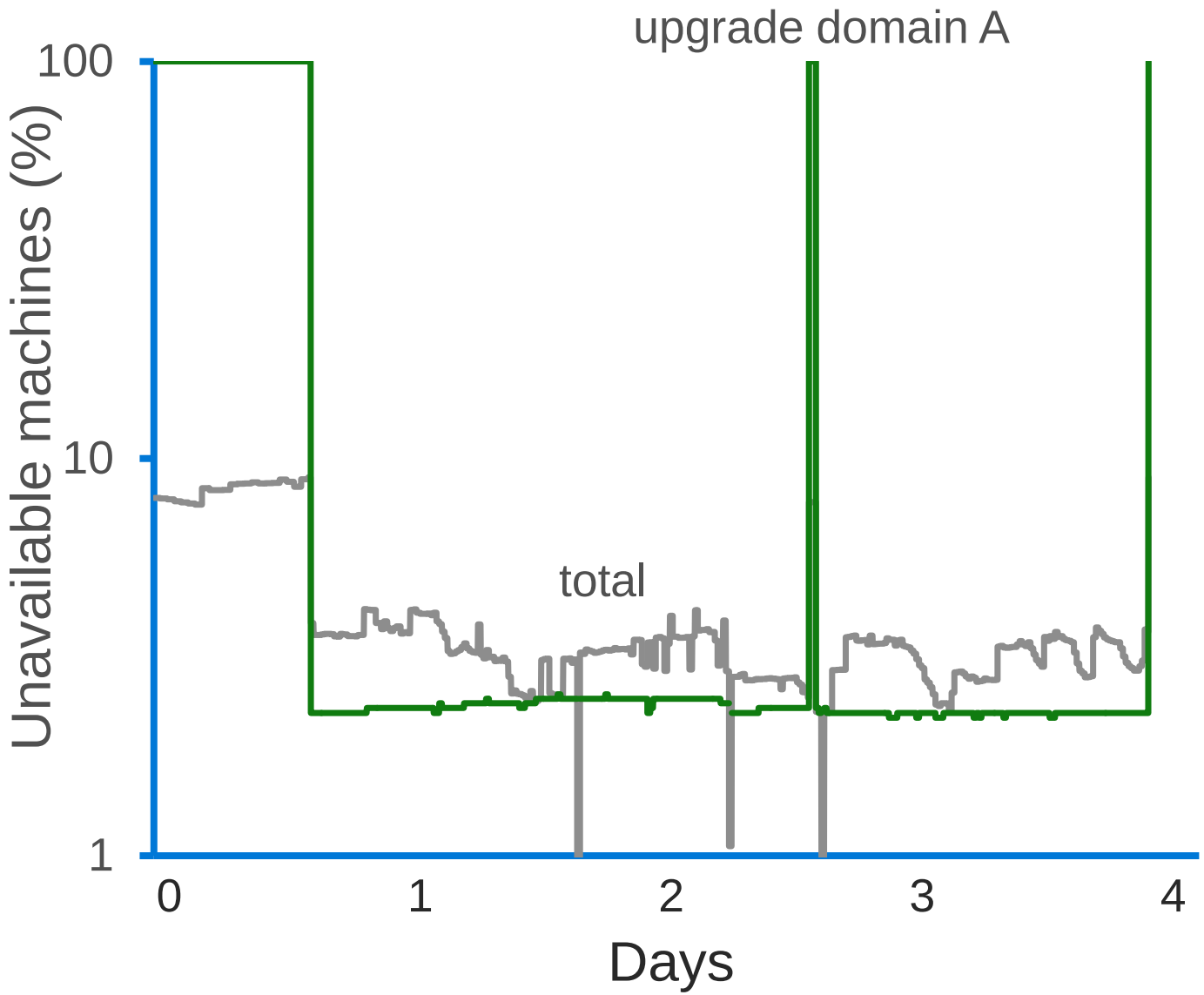
It is all about placement with constraints!

Machine unavailability in a Microsoft cluster



- Less than 10% of cluster nodes unavailable
- Cluster is organized in node groups

Machine unavailability in a Microsoft cluster



- Less than 10% of cluster nodes unavailable
- Cluster is organized in node groups
- Machines become unavailable in groups
- With random placement, an LRA might lose all its containers at once

Existing solutions for placement with constraints

- Random placement
  Good performance… only if you are *very* lucky
- Specify target nodes
  "Place containers c1 and c2 on node n1"
- Based on static machine attributes
  "Place containers on GPU nodes"
- Kubernetes supports affinity and anti-affinity across containers
  Places one container at a time → many constraint violations

Challenges

- How to refer to container groups and node groups?
  - Container tags and node groups
- How to express constraints related to LRA containers?
  - Expressive constraints within and across LRAs
- How to achieve high quality placement without affecting task-based jobs?
  - Placement constraint processor

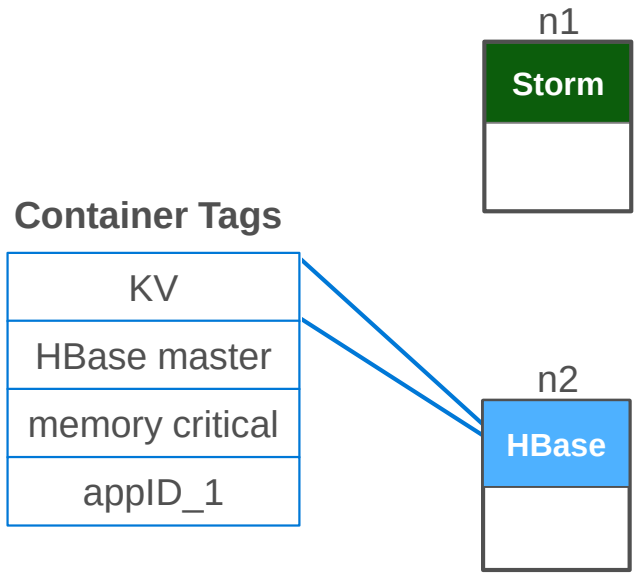
Container tags

> Idea: use tags to refer to groups of containers

Describe:

- application type
- application role
- resource specification
- global application ID

Can refer to any current or future LRA container



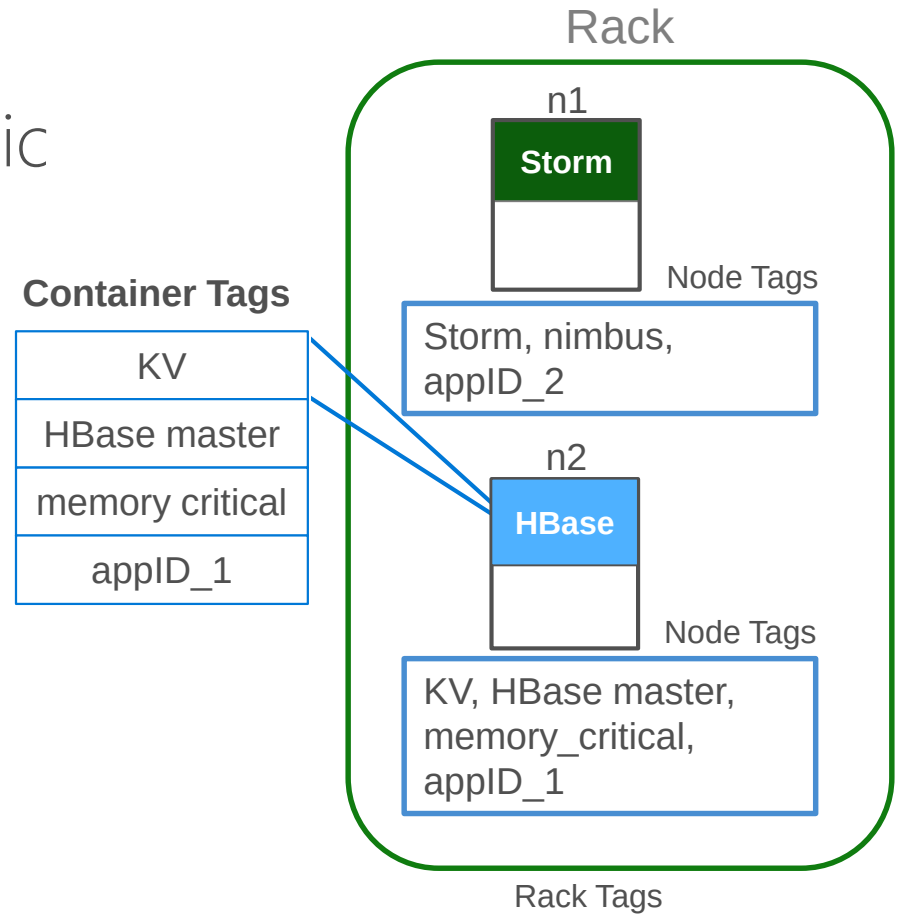
Node groups

> **Idea:** logical node groups to refer to dynamic node sets

Examples: node, rack, upgrade domain

Associate nodes with all the container tags that live there

Hide infrastructure
"spread across upgrade domains"



Defining constraints

> Placement Constraints API

Static methods for LRAs to specify constraints

> Affinity "Place 3 Storm containers in the same rack as an HBase container"

storm=3, IN, RACK, hbase

> Anti-affinity "Place 5 Storm containers in different nodes than Spark"

storm=5, NOTIN, NODE, spark

> Cardinality "Place 7 Storm containers with no more than 5 containers per node"

storm=7, CARDINALITY, NODE, storm, 0, 5

Defining constraints cont'd

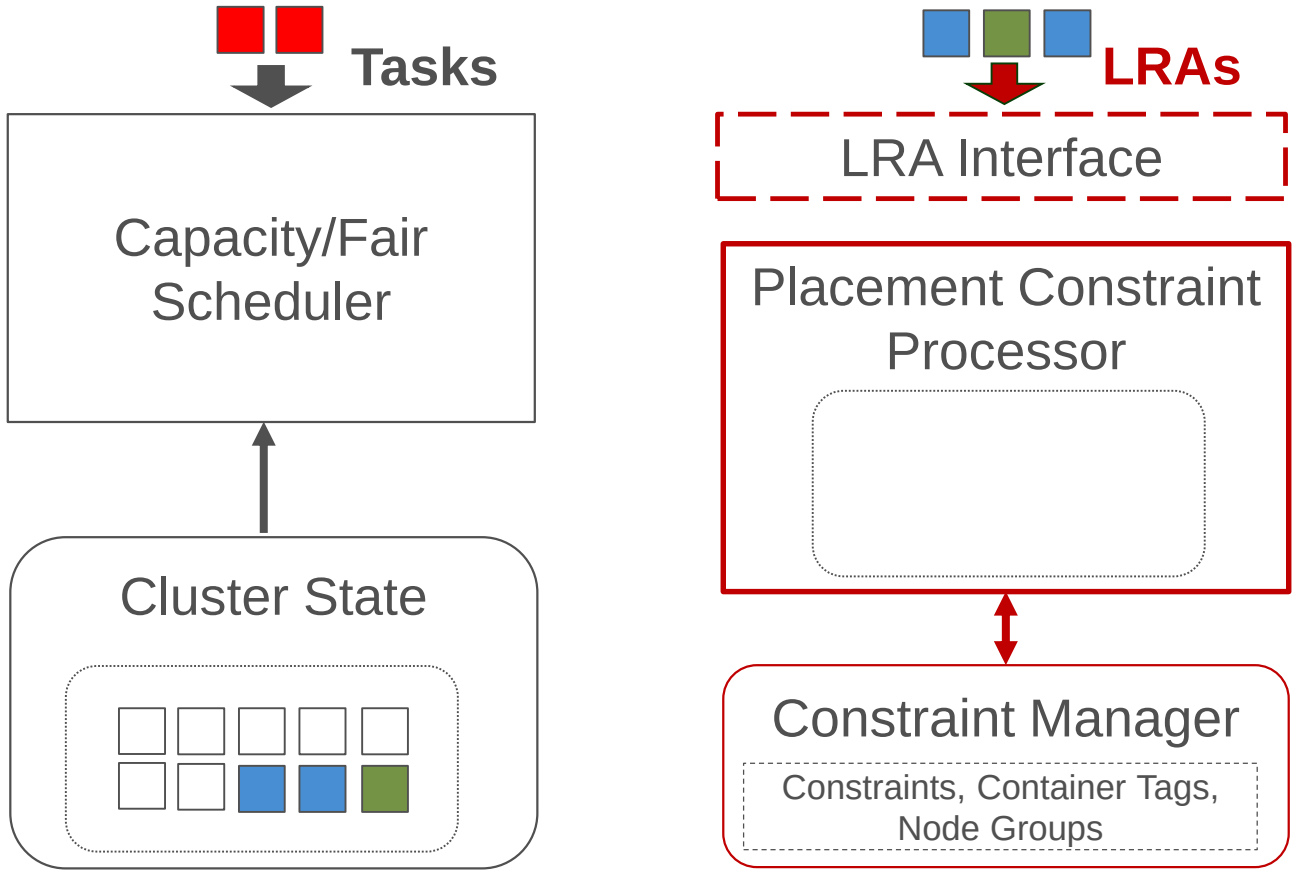
- Intra- and inter-application constraints
  zk=3, NOTIN, NODE, **not-self/**zk
  hbase=5, IN, RACK, **all/**zk

- Composite constraints (AND, OR)
  zk=5, **AND**(IN, RACK, hbase : NOTIN, NODE, zk)

- Constraints can be defined at different levels
  Scheduling request
  Application
  Cluster-wide

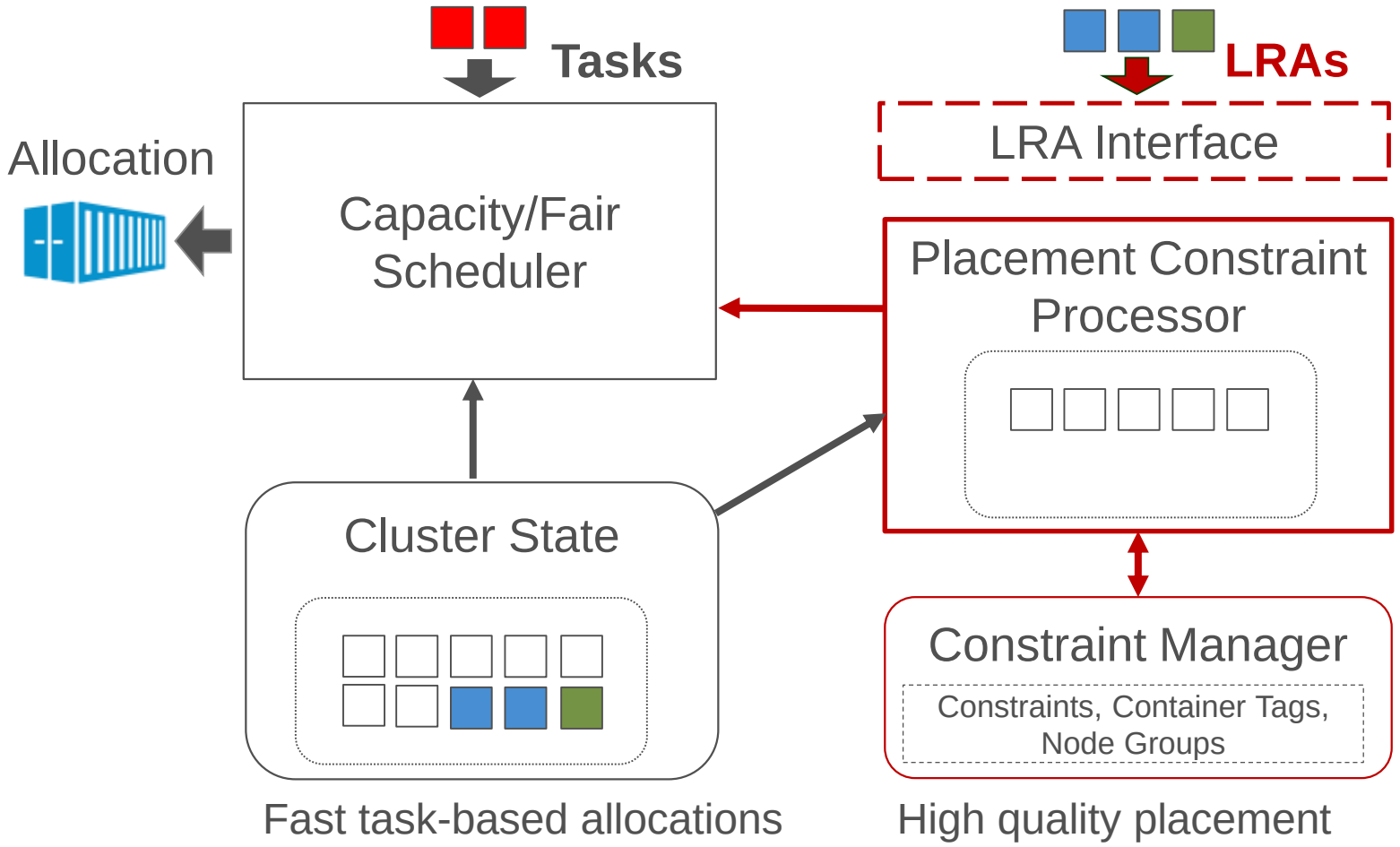
Scheduling with constraints

> Idea: introduce Placement Constraint Processor for satisfying constraints of LRA requests



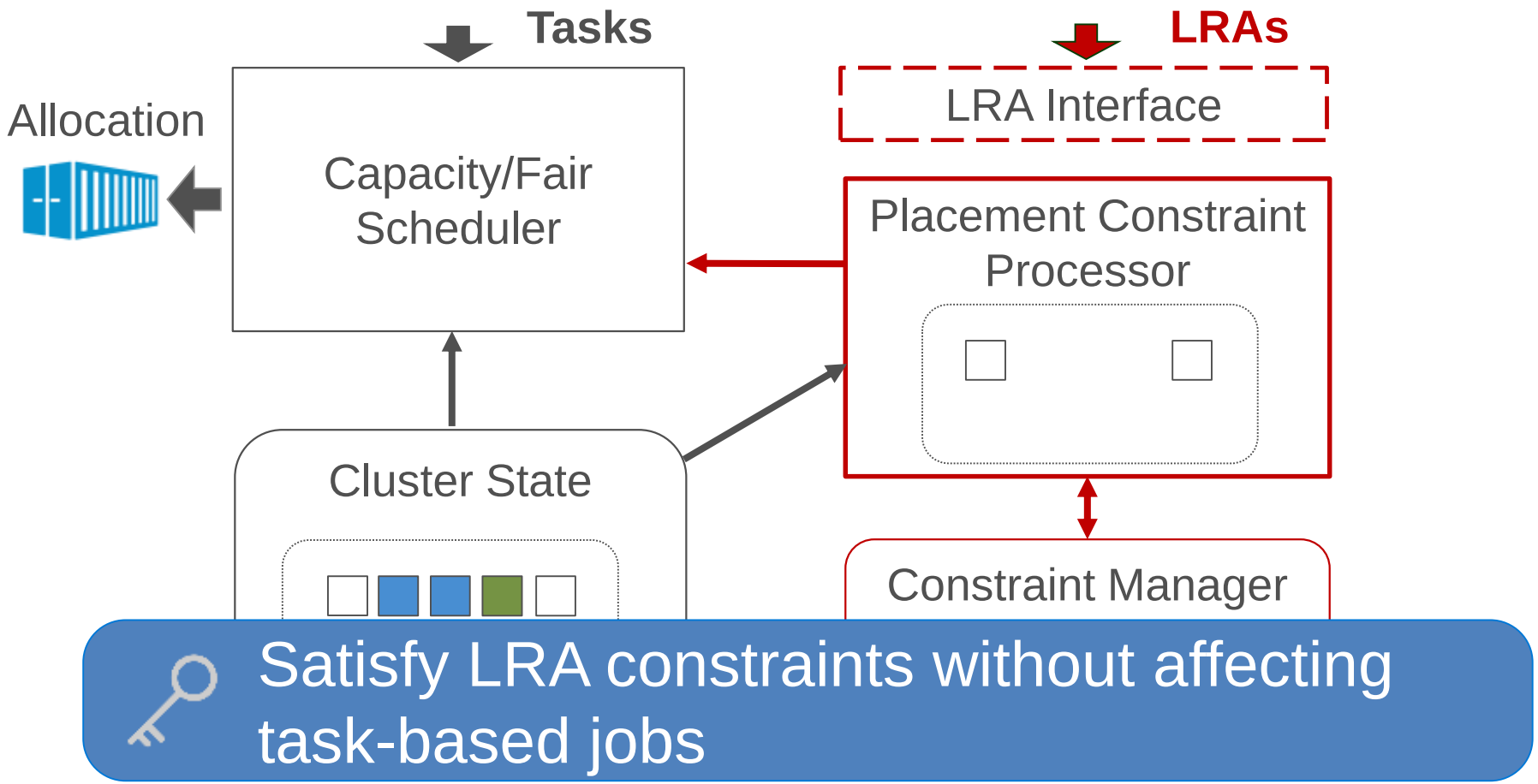
Scheduling with constraints

> LRA scheduling algorithm in the Placement Constraints Processor
> Invoked when an LRA is submitted, considers multiple containers



Scheduling with constraints

- LRA scheduling algorithm in the Placement Constraint Processor
  Invoked when an LRA is submitted, considers multiple containers



Implementation

- Part of Apache Hadoop as of release 3.1
  Umbrella JIRA: YARN-6592

- Main additions:
  Placement Constraints API
  Tag Manager, Constraint Manager, Placement Constraint Processor

- Additional implementation of placement constraints inside the Capacity Scheduler (useful for non-LRA constraints)

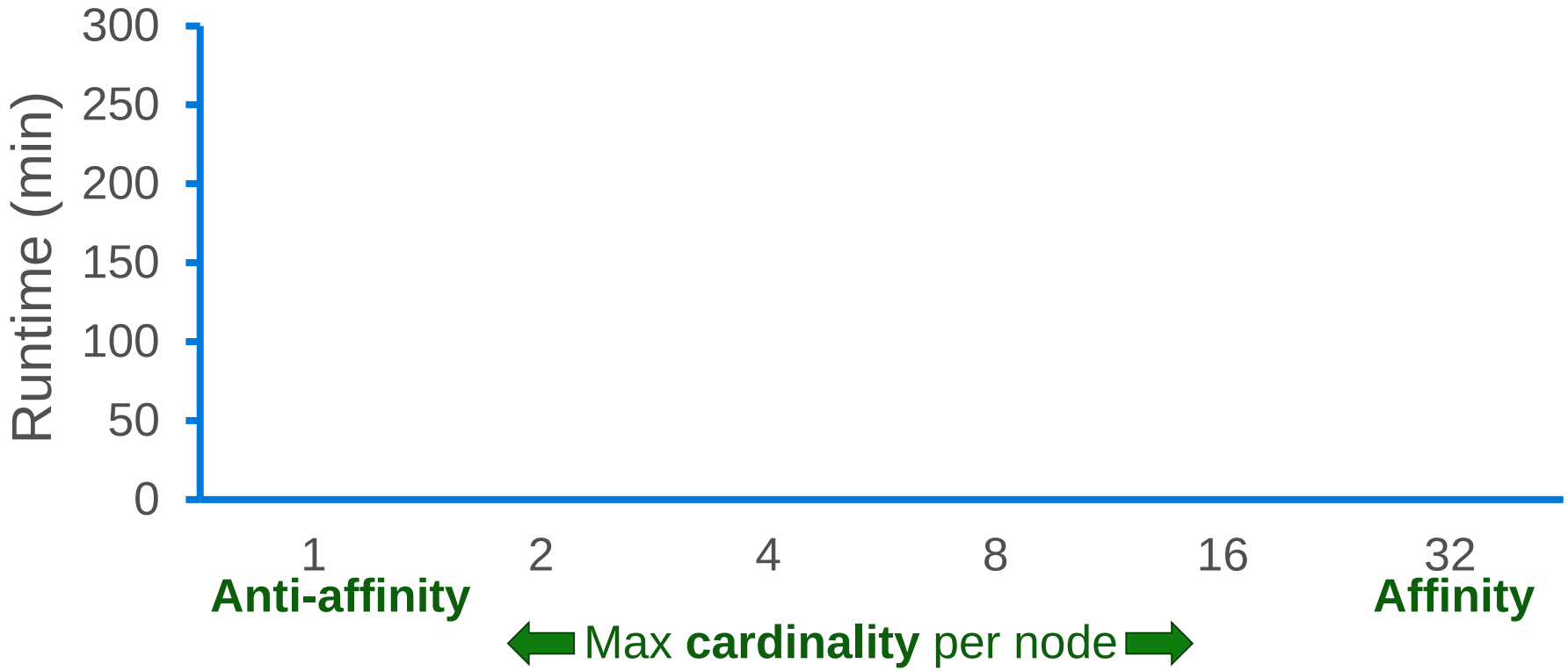
- Special thanks also to: Arun Suresh (Microsoft), Weiwei Tan (Alibaba), Panagiotis Garefalakis (Imperial College)

Evaluation

> Do we need expressive constraints?

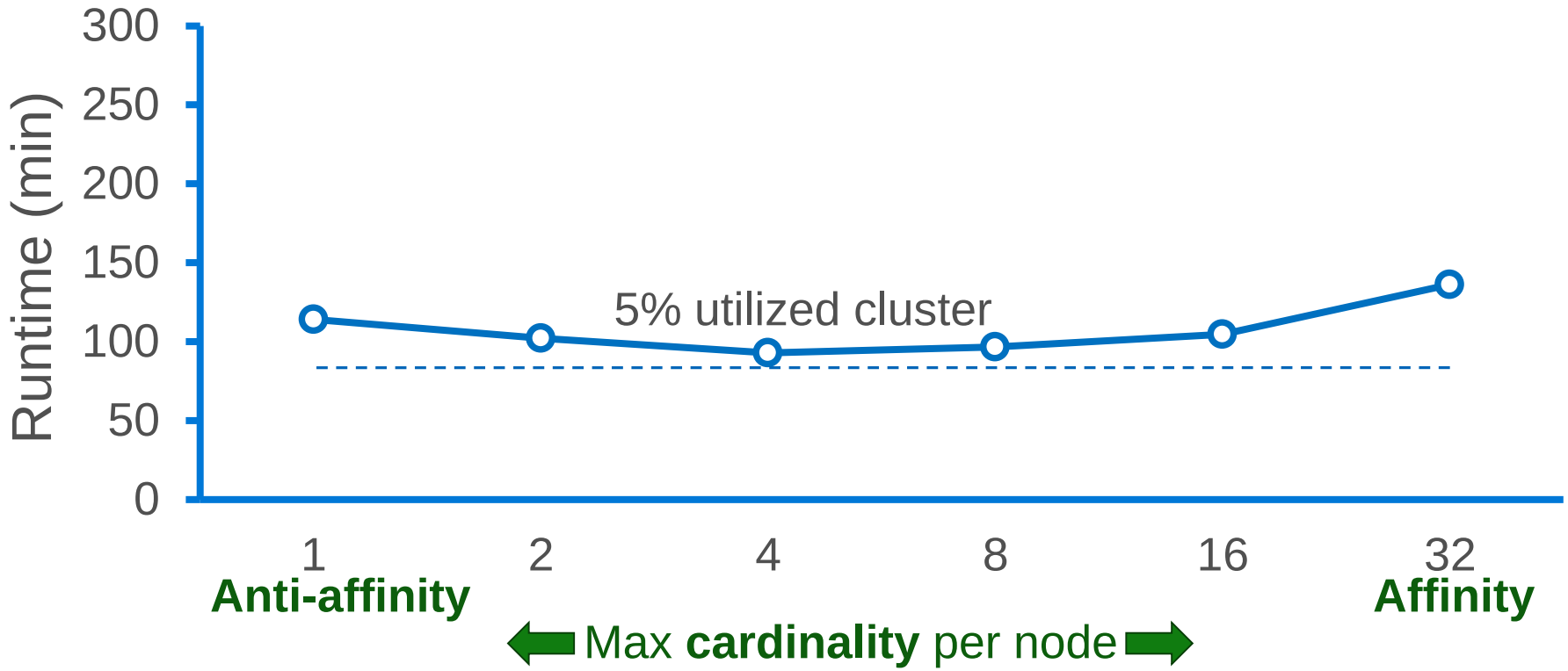
> What are the performance benefits of Hadoop 3.1 for LRAs?

Importance of expressive constraints



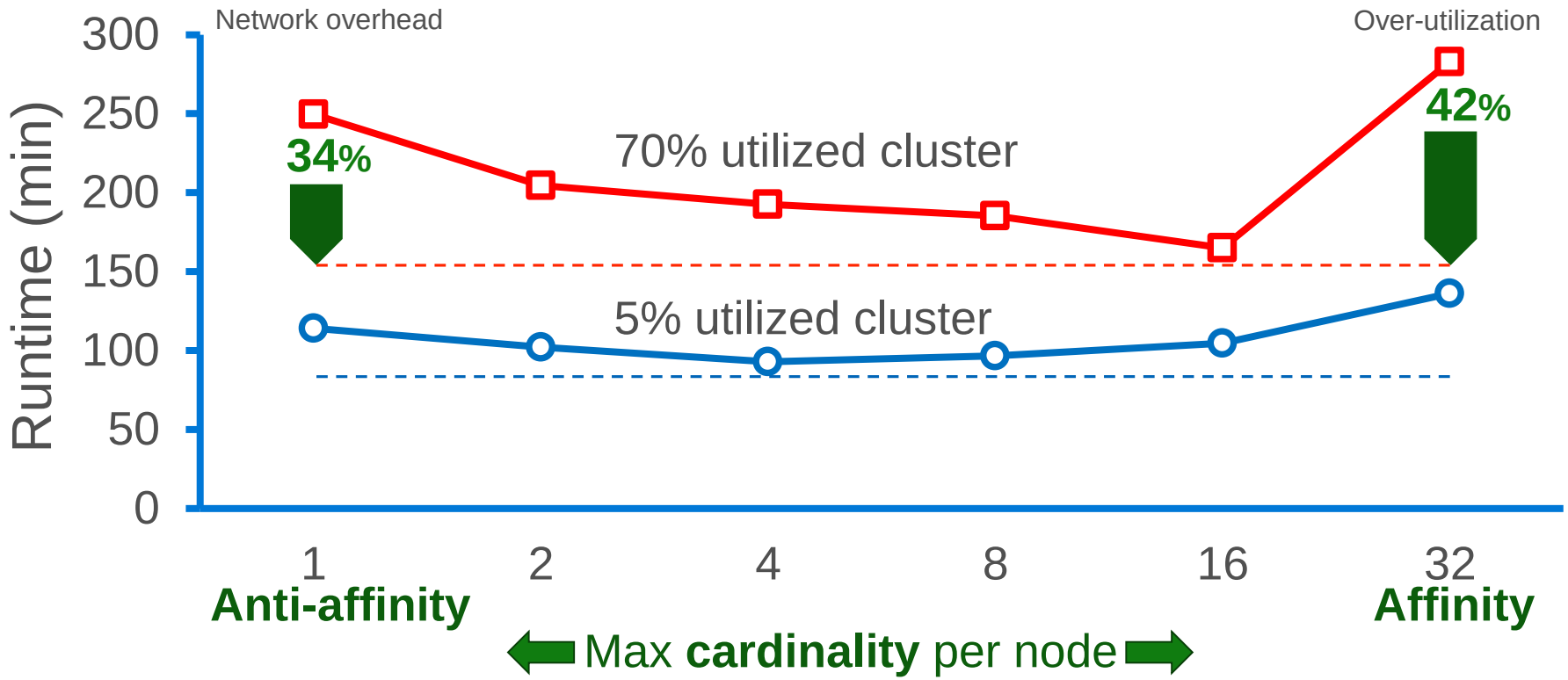
TensorFlow ML workflow with 1M iterations using 32 workers with varying workers per node

Importance of expressive constraints



Cardinality constraints are important
Affinity and anti-affinity are not enough

Importance of expressive constraints



Cardinality constraints are important
Affinity and anti-affinity are not enough

Larger-scale deployment

> Pre-production cluster

400 machines on 10 racks

> Workloads

50 HBase instances (10 workers each)

45 TensorFlow instances (8 workers and 2 PS each)

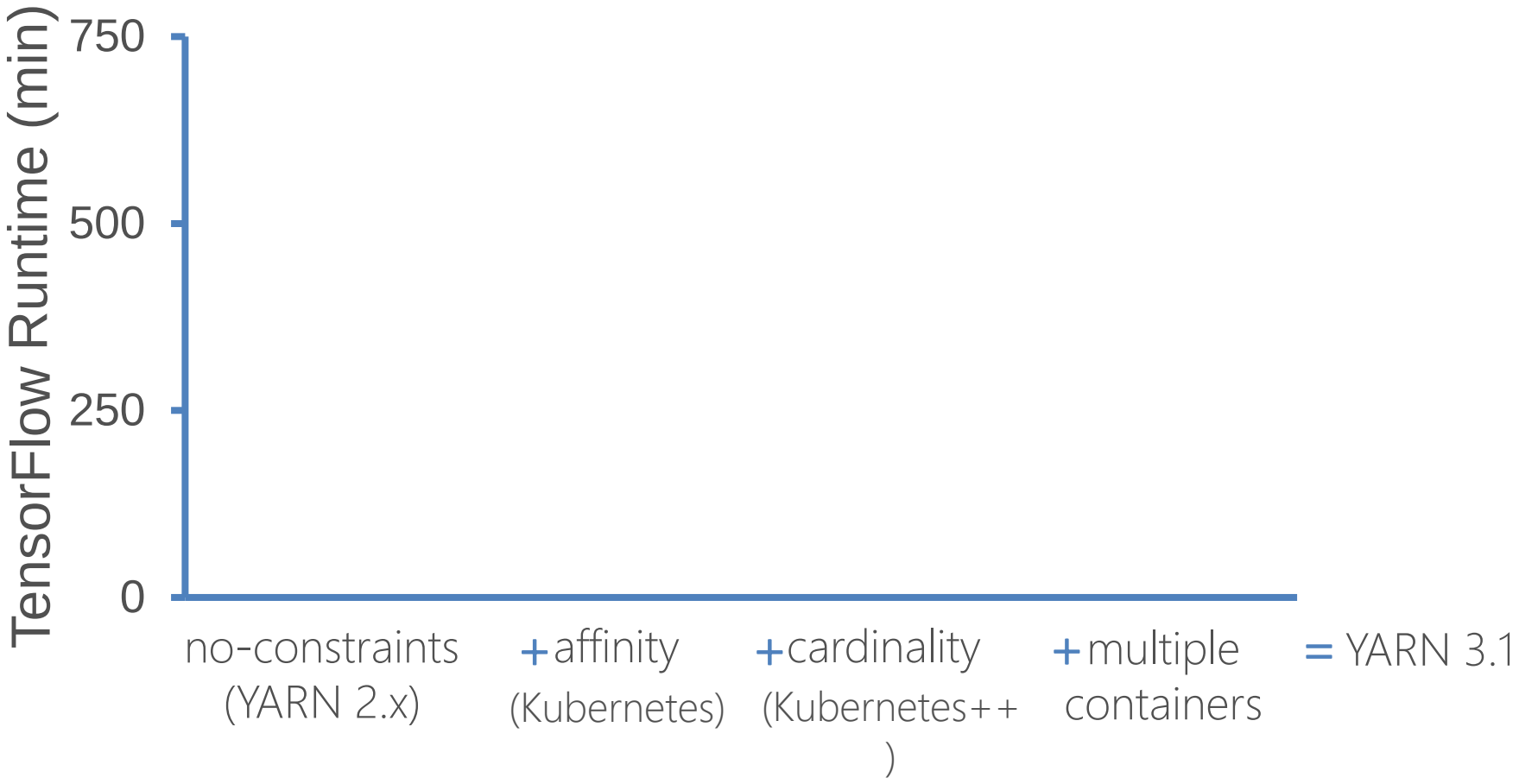
Batch production workload (50% of cluster resources)

> Constraints

Containers of each LRA instance on the same rack

No more than 2 HBase (4 for TensorFlow) containers on same node

LRA performance in YARN 3.1



Significant performance and predictability improvement!

How can I try it out?

- Only one parameter in yarn-conf.xml **yarn.resourcemanager.placement-constraints.handler** set to **placement-processor** or **scheduler**

- Applications should use the PlacementConstraints API at their AM

- Services configuration has support for placement policies

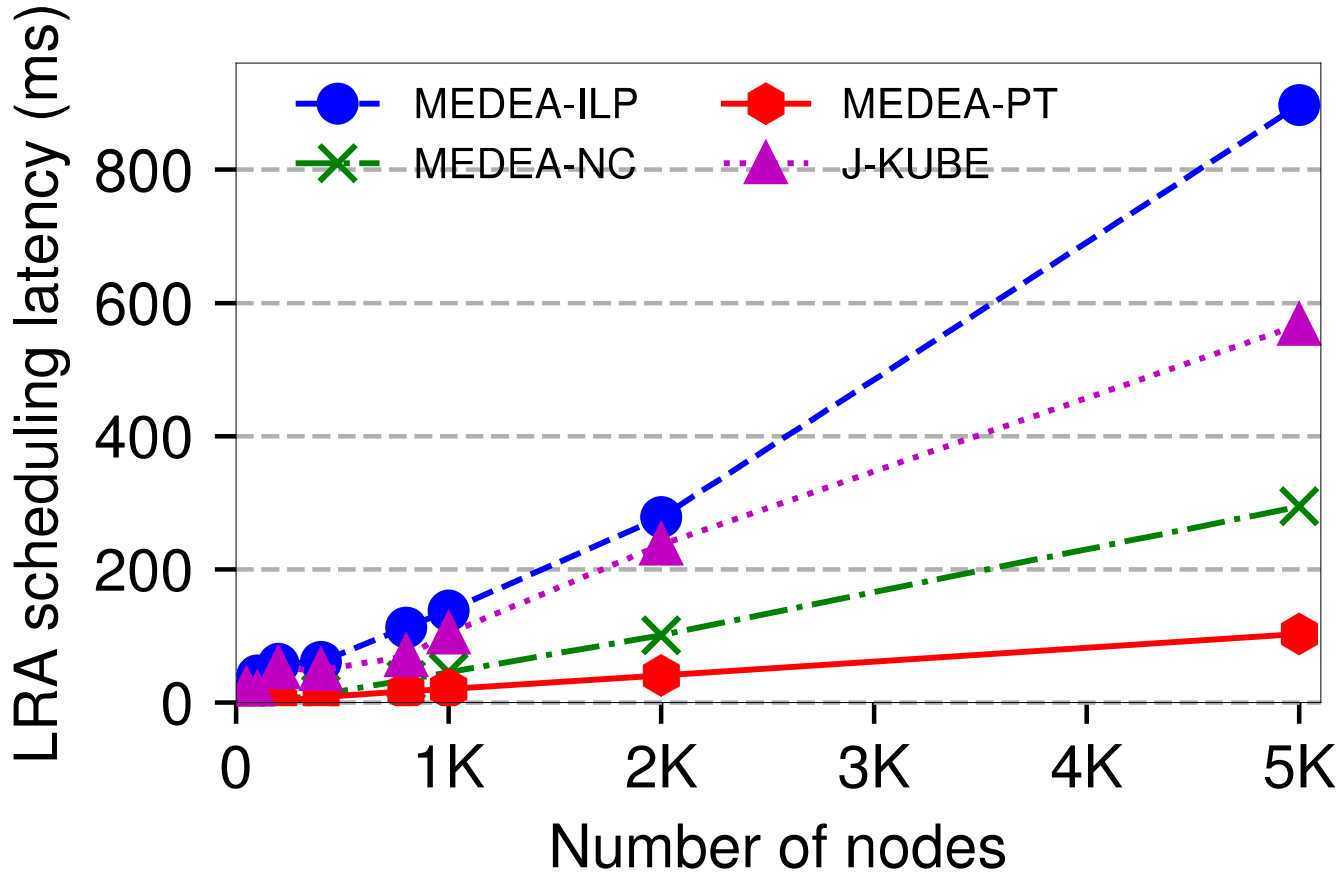
- Documentation with examples at: http://hadoop.apache.org/docs/r3.1.0/hadoop-yarn/hadoop-yarn-site/PlacementConstraints.html

Wrap-up

- Important additions for long-running applications/services in Hadoop 3.1
- Deployment, packaging, upgrade, discovery of LRAs
- Scheduling of LRAs
  Expressive constraints (affinity, anti-affinity, cardinality)
  High quality placement via constraint processor
- Many more things to be done: come help us!
- Demo time!

BACKUP SLIDES

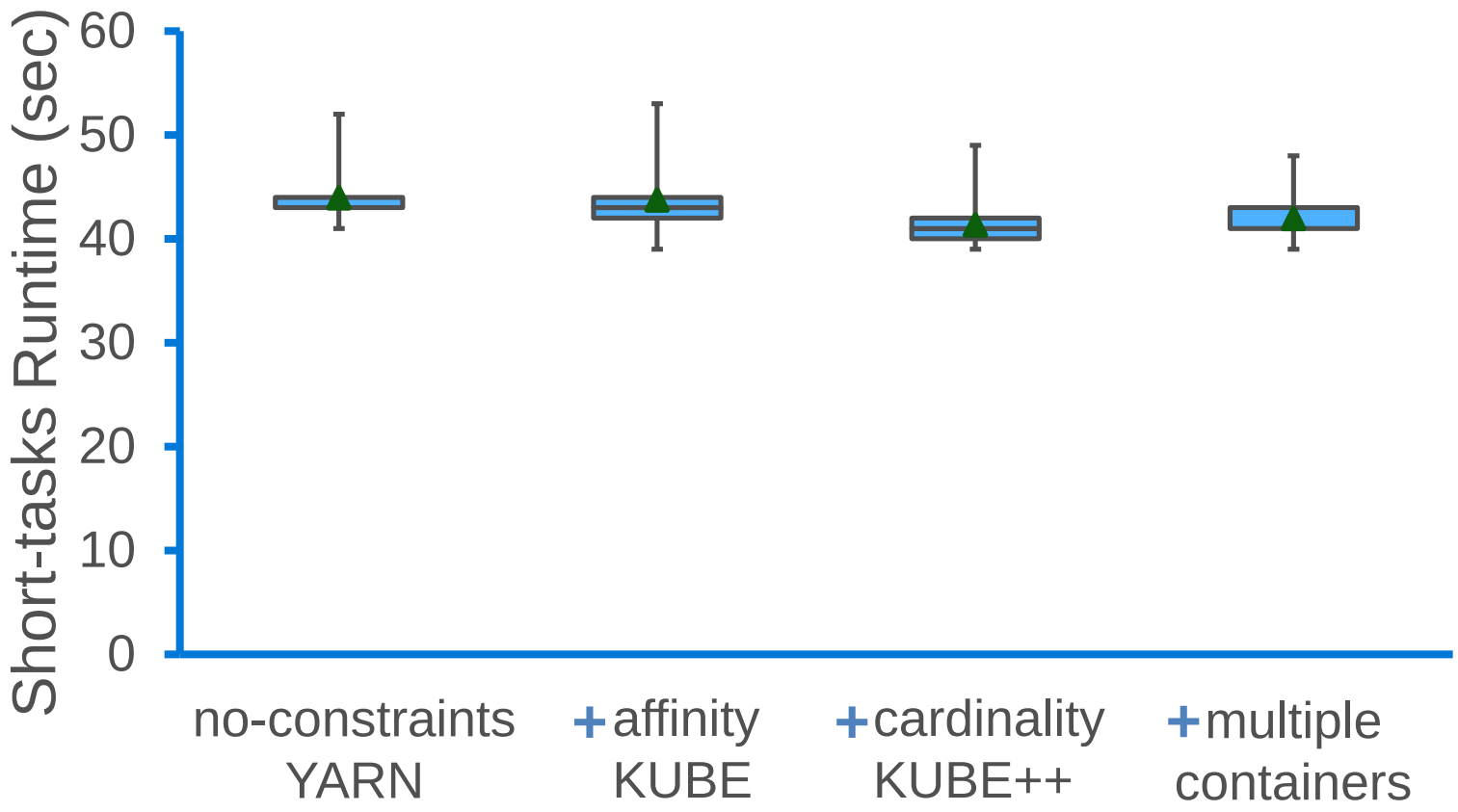
Scheduling scalability



> Latency for placing all containers of an LRA

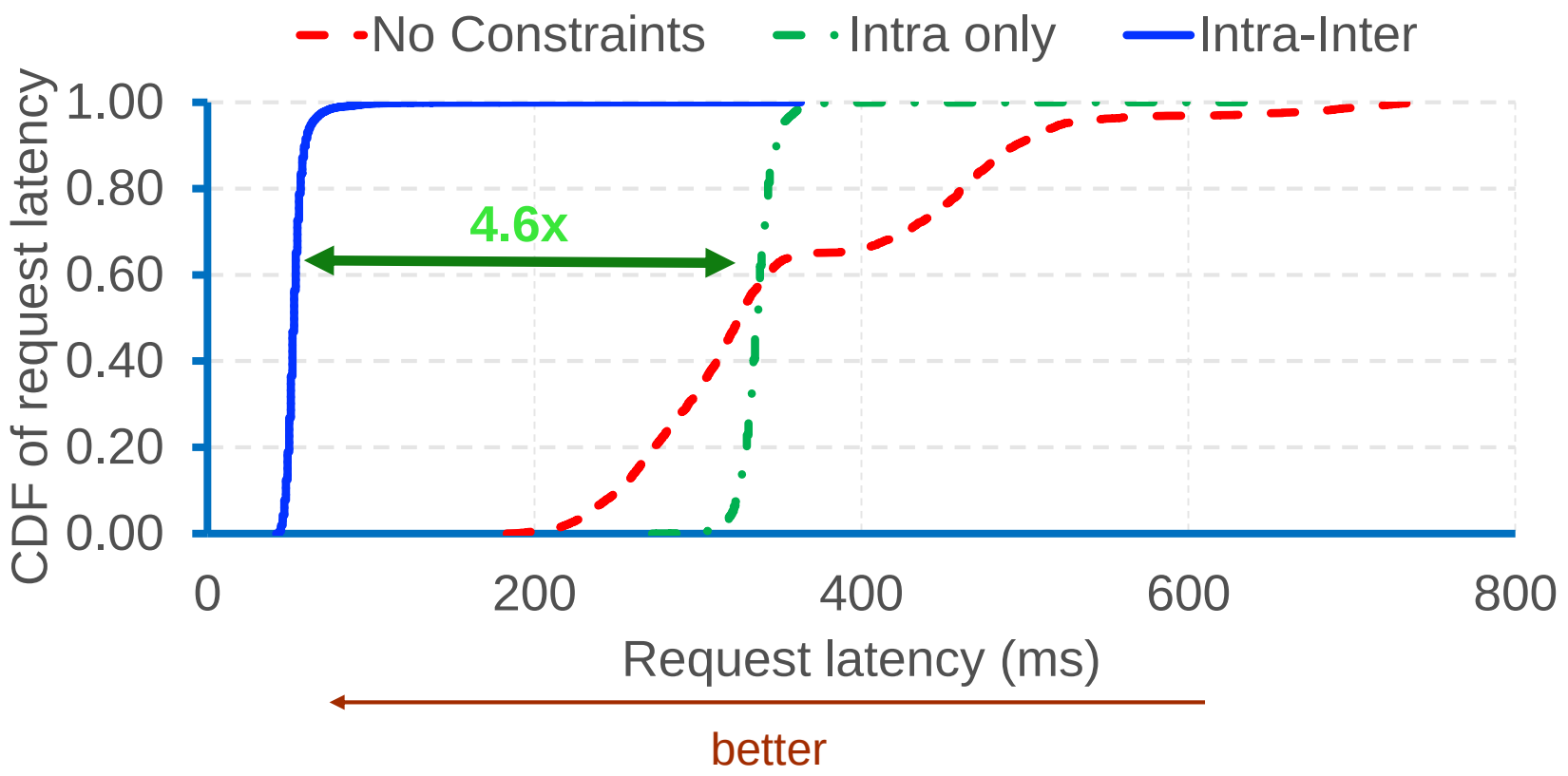
- 20% of cluster resources for LRAs
- Sum of scheduling latencies for all LRAs

Impact of MEDEA in Task performance



Task-based job runtimes are not affected

LRA performance: inter-app



Both intra- and inter-application constraints are crucial to application performance