Apache YARN 3.x in Alibaba

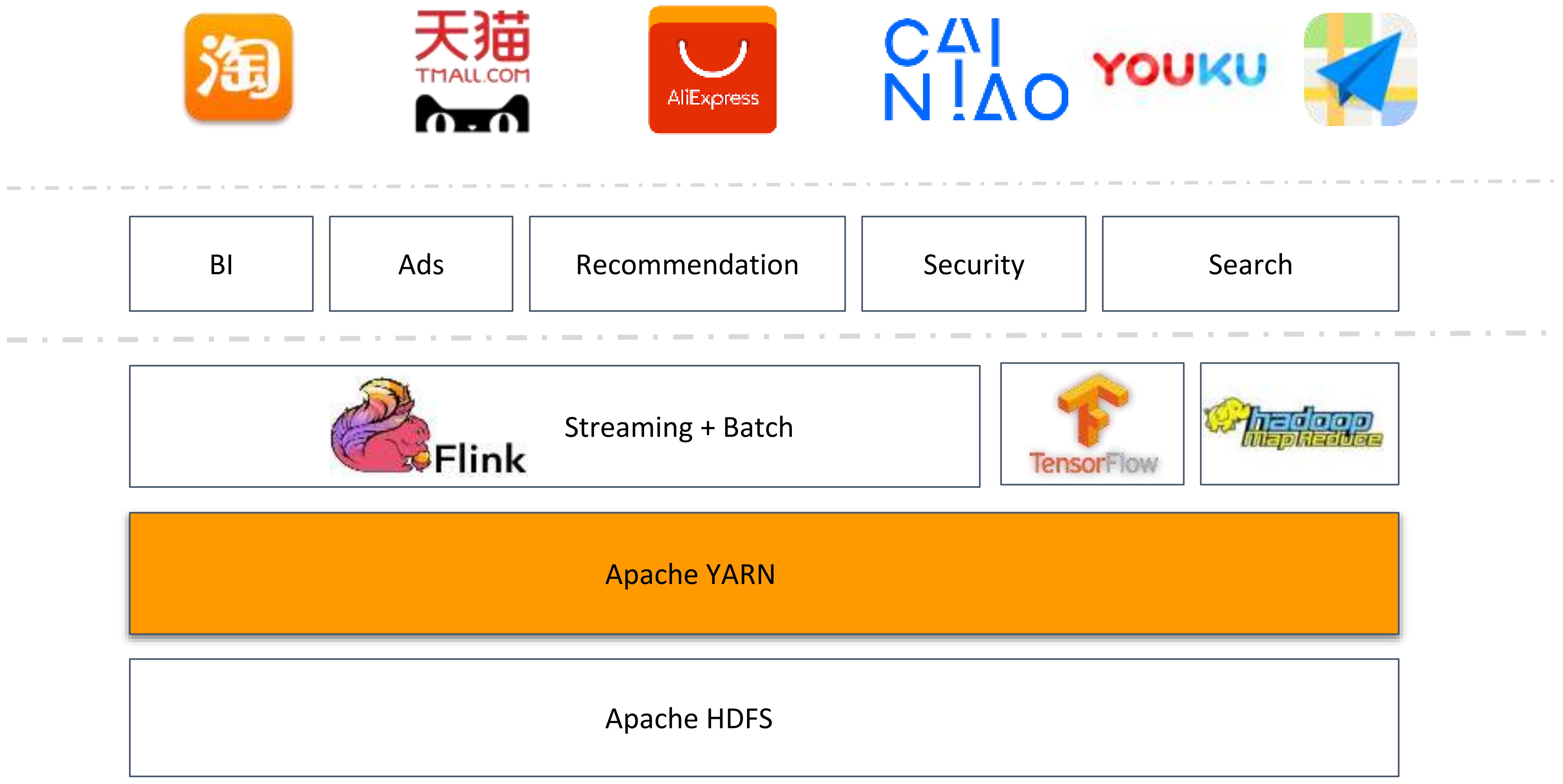
Weiwei Yang, Chunde Ren

Agenda

Apache YARN Ecosystem in Alibaba

Part I

Apache YARN Ecosystem in Alibaba



Challenges

Utilization

Service SLA

Omni Scheduler

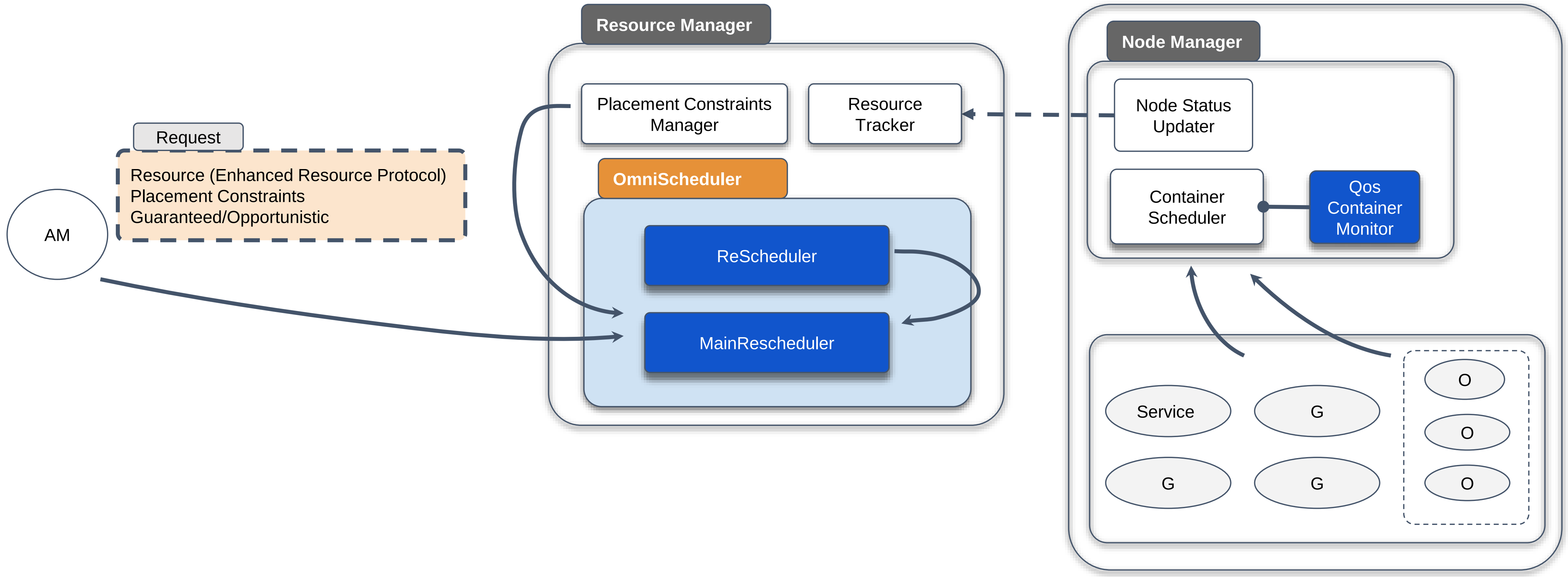
Part II

Motivation

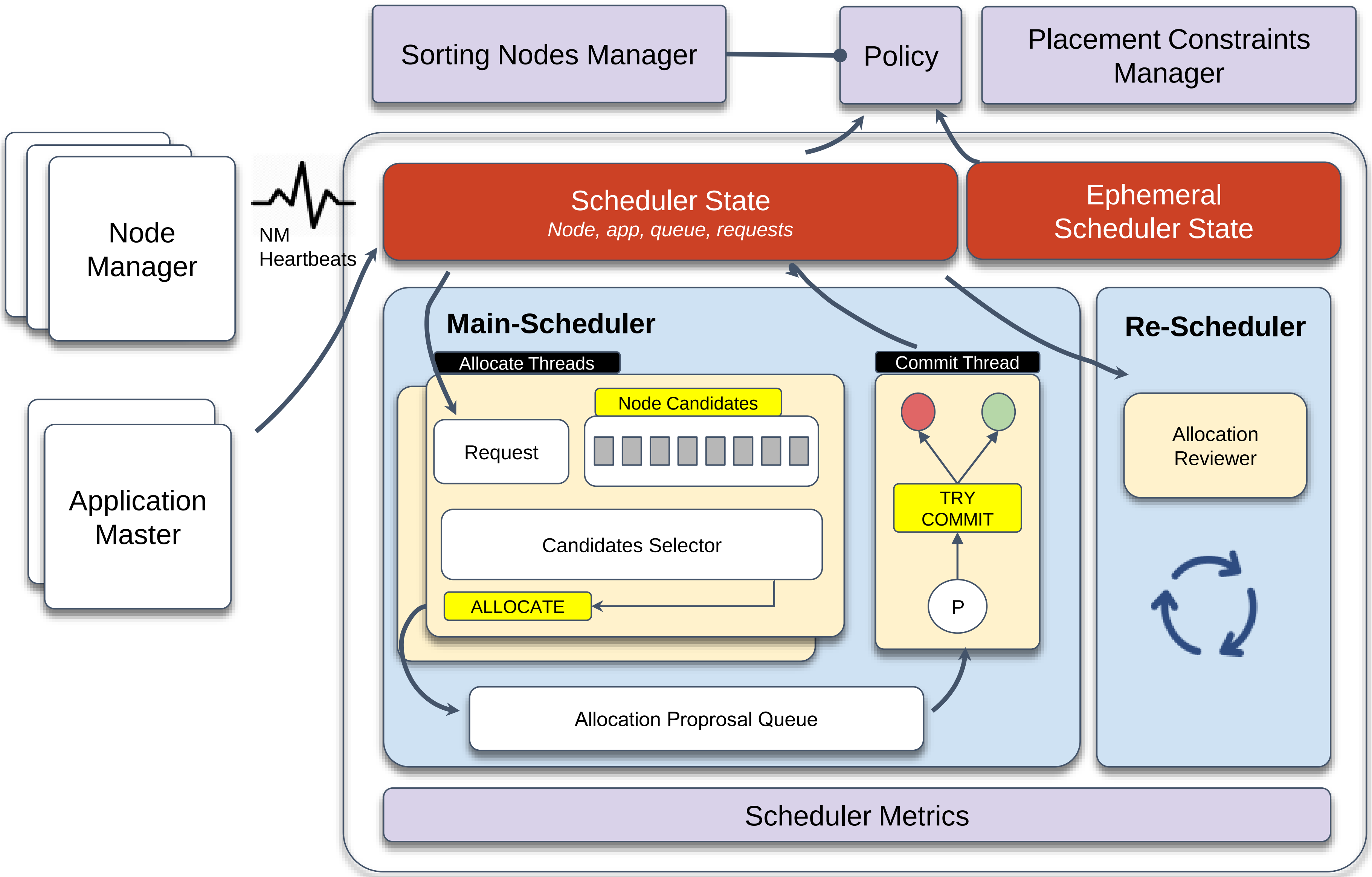
Existing Capacity/Fair Scheduler:

- Not ready to support resource over-subscription
- Not able to make overall good decisions
- Limitations to support online service
- Absent of dynamic scheduling ability

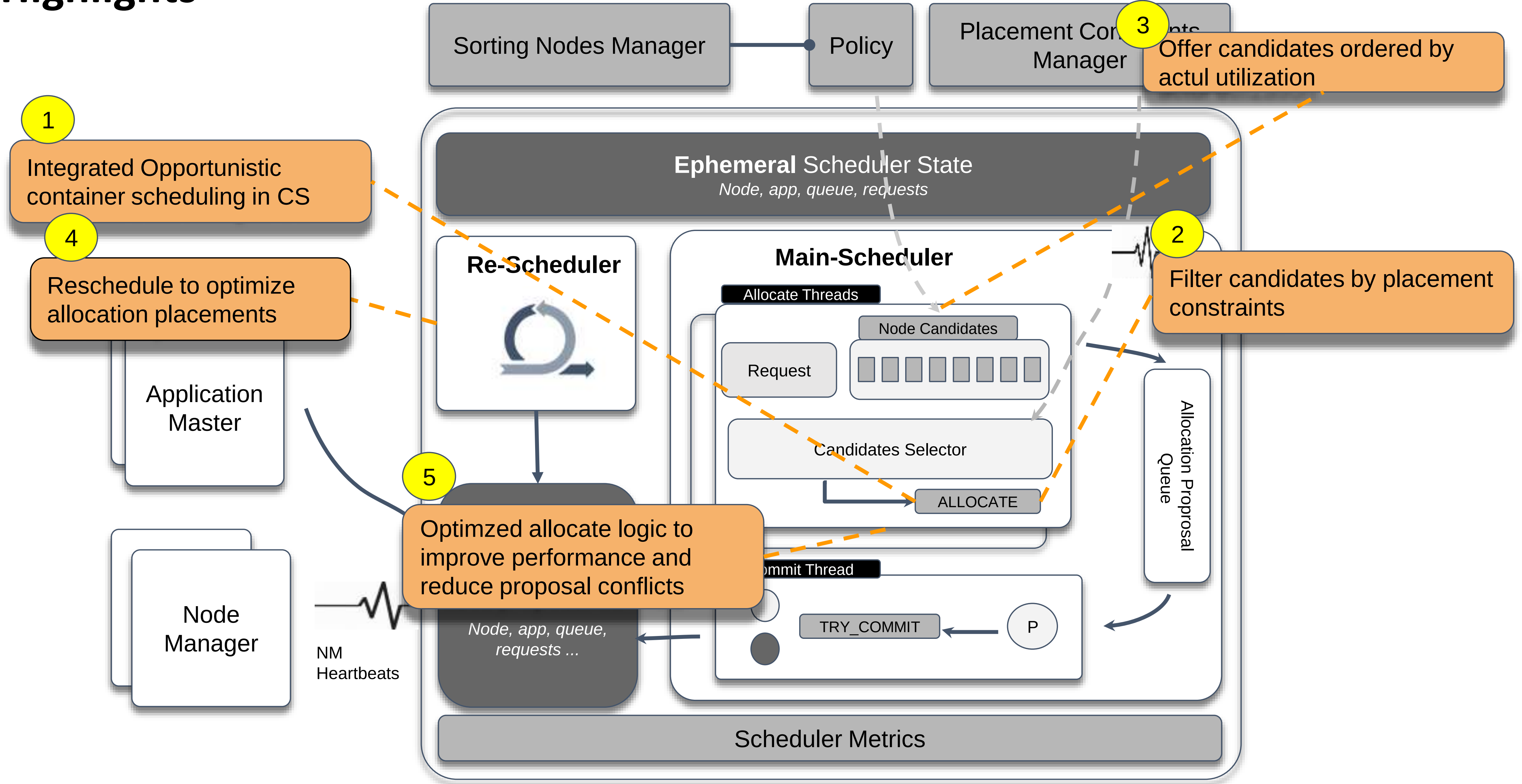
OmniScheduler Architecture



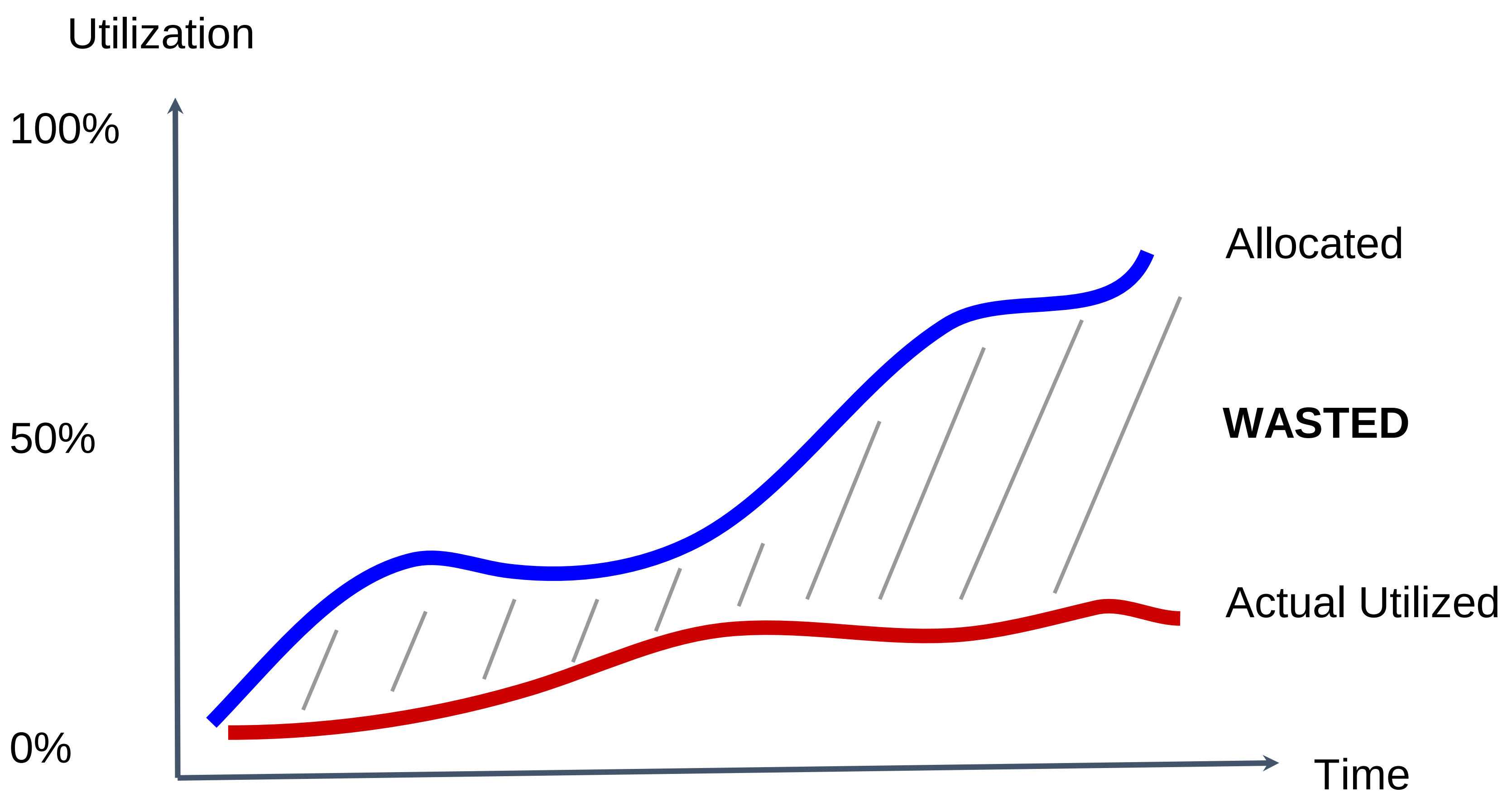
OmniScheduler - A Closer Look



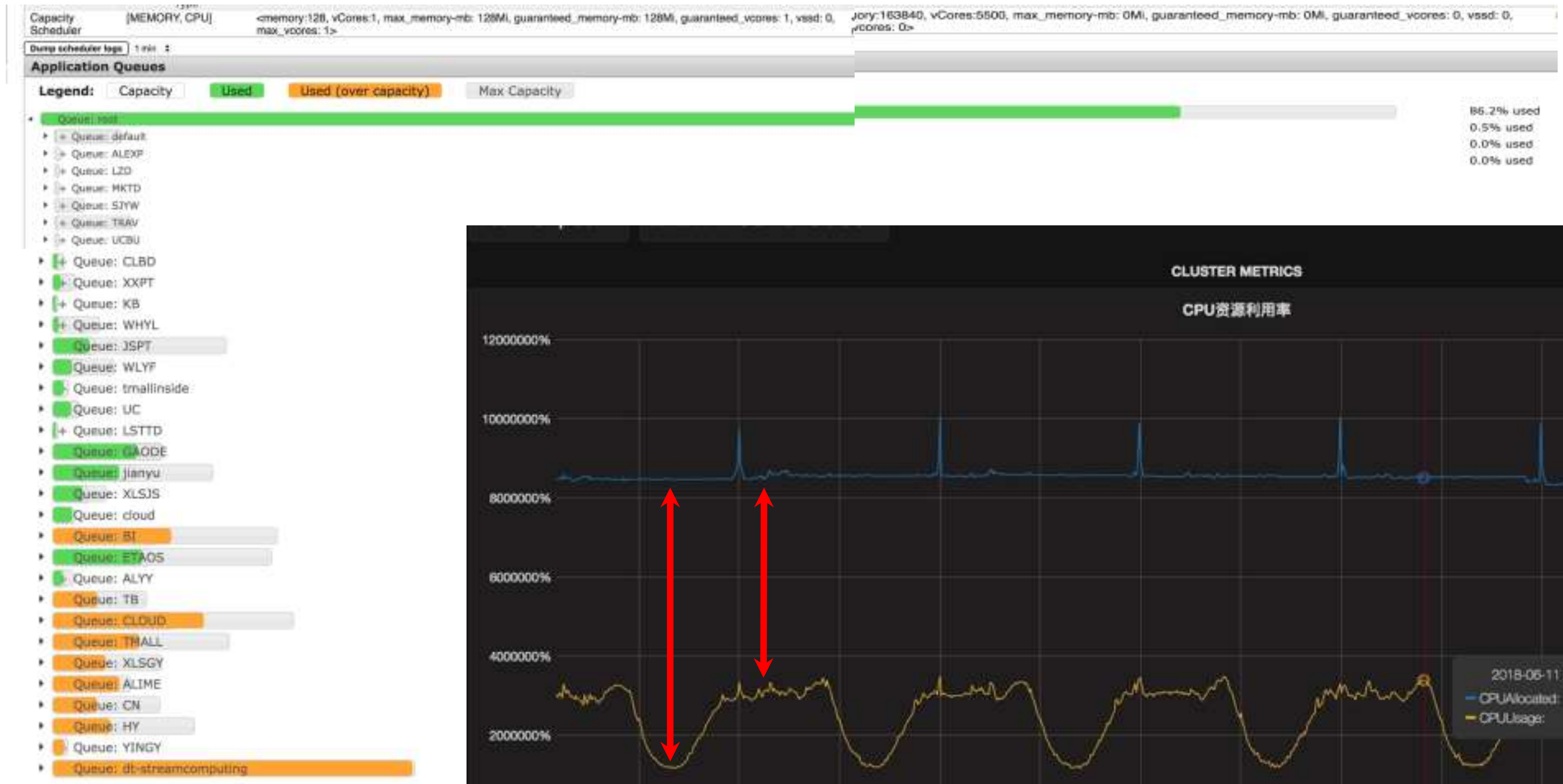
Highlights


Sorting Nodes Manager
Policy
Placement Constraints Manager
3
Offer candidates ordered by actul utilization
1
Integrated Opportunistic container scheduling in CS
Ephemeral Scheduler State
Node, app, queue, requests
4
Reschedule to optimize allocation placements
Re-Scheduler
Main-Scheduler
2
Filter candidates by placement constraints
Allocate Threads
Node Candidates
Request
Candidates Selector
ALLOCATE
Allocation Proposal Queue
Application Master
5
Optimzed allocate logic to improve performance and reduce proposal conflicts
Commit Thread
TRY_COMMIT
P
Node Manager
NM Heartbeats
Node, app, queue, requests ...
Scheduler Metrics

What is resource oversubscription



The Problem


Capacity Scheduler [MEMORY, CPU] <memory:128, vCores:1, max_memory-mb: 128Mi, guaranteed_memory-mb: 128Mi, guaranteed_vcores: 1, vssd: 0, max_vcores: 1> ory:163840, vCores:5500, max_memory-mb: 0Mi, guaranteed_memory-mb: 0Mi, guaranteed_vcores: 0, vssd: 0, vcores: 0>

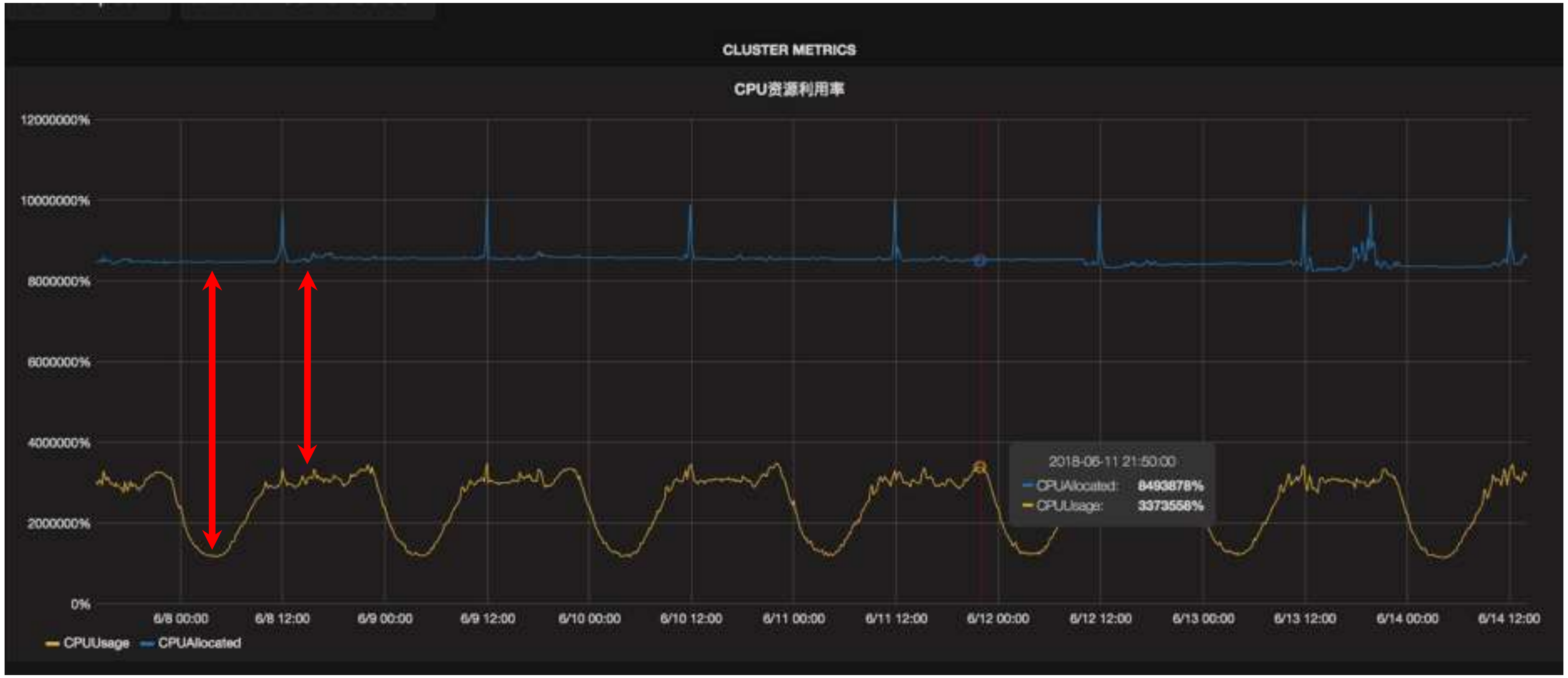
Dump scheduler logs 1 min

Application Queues

Legend: Capacity 🟩 Used 🟧 Used (over capacity) Max Capacity

- Queue: root 86.2% used
  - Queue: default 0.5% used
  - Queue: ALEXP 0.0% used
  - Queue: LZD 0.0% used
  - Queue: MKTD
  - Queue: SJYW
  - Queue: TRAV
  - Queue: UCBU
  - Queue: CLBD
  - Queue: XXPT
  - Queue: KB
  - Queue: WHYL
  - Queue: JSPT
  - Queue: WLYF
  - Queue: tmallinside
  - Queue: UC
  - Queue: LSTTD
  - Queue: GAODE
  - Queue: jianyu
  - Queue: XLSJS
  - Queue: cloud
  - Queue: BI
  - Queue: ETAOS
  - Queue: ALYY
  - Queue: TB
  - Queue: CLOUD
  - Queue: TMALL
  - Queue: XLSGY
  - Queue: ALIME
  - Queue: CN
  - Queue: HY
  - Queue: YINGY
  - Queue: dt-streamcomputing

86.2% Allocated



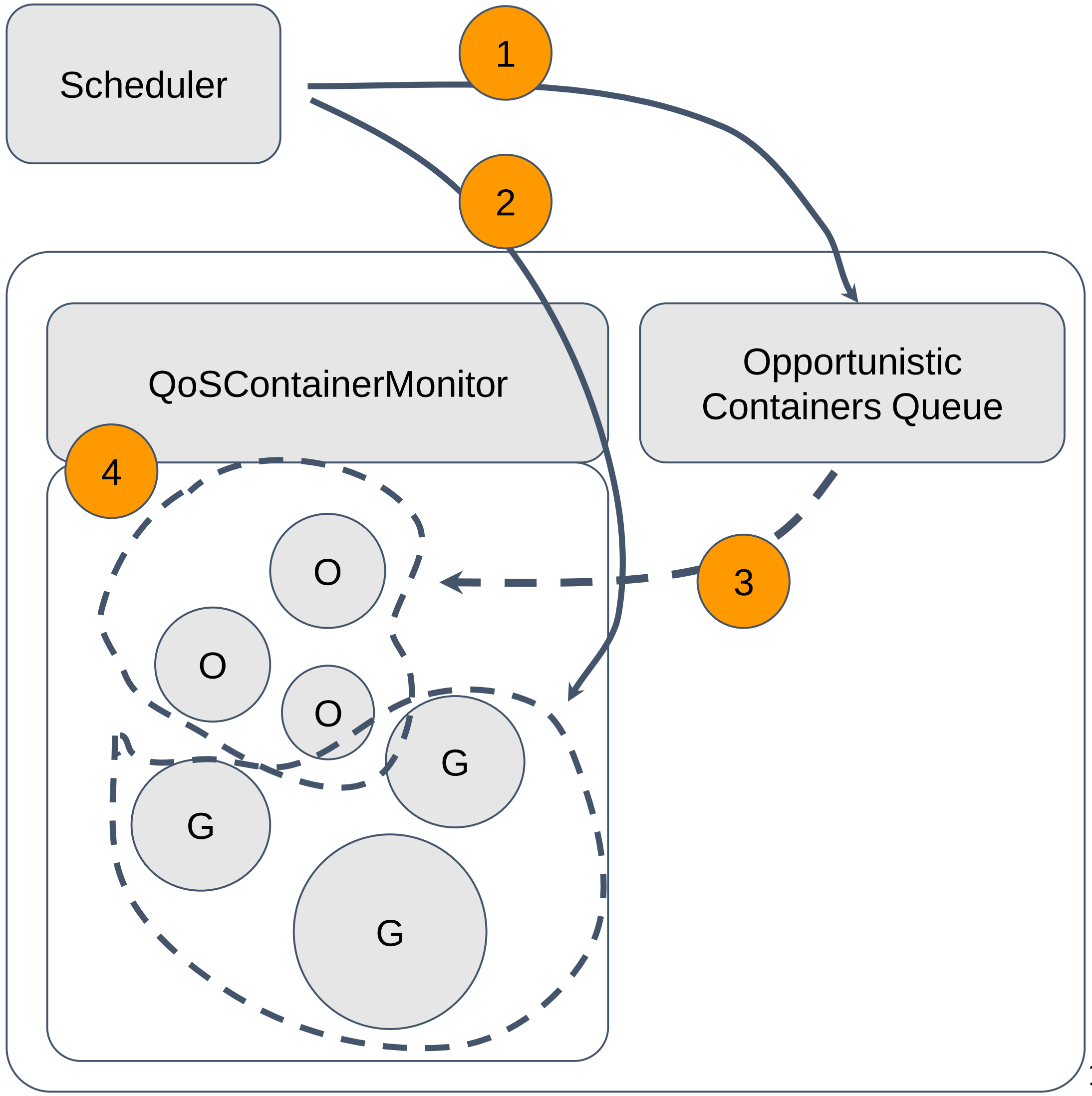
Allocated: 8.5 million cores
Actual Utilized Peak: 3.3 million cores
Actual Utilized Trough: 1.2 million cores

Resource Oversubscription

Integrated Opportunistic container scheduling into Capacity Scheduler, it leverages a dynamical over-allocation threshold in order to ensure a reasonable range of the oversubscription. We also added a QoS module on NM to manage the lifecycle of Opportunistic containers.

Objective

1. Better Fairness
2. Scheduling with predicted resource utilization
3. 2 thresholds (G+O/O) and 2 factors (min, predict)
4. Qos (Isolation and Elastic...)
5. Future: optimized preemption decision with consideration of preemption cost

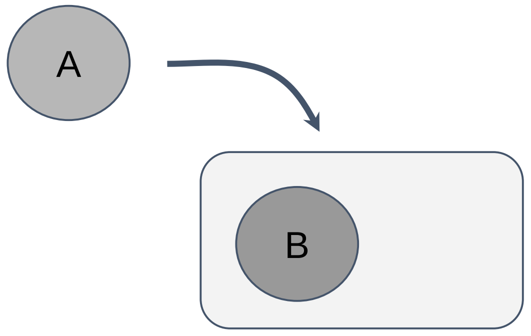


Placement Constraints

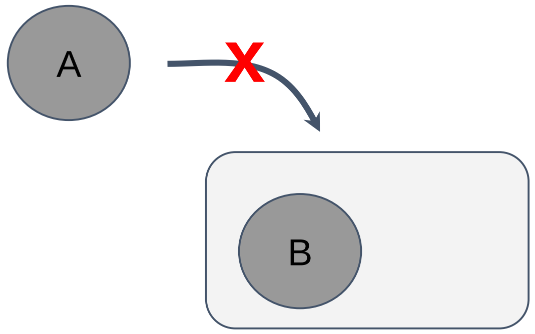
1 **A**: Don’t place me with **B** on same node (anti-affinity)

2 **A**: Do place me with **B** on same node (affinity)

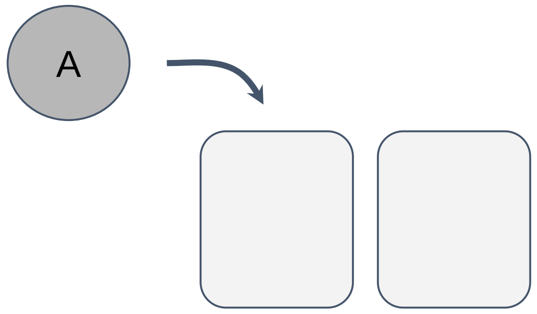
3 **A**: Do place me on node that has … (affinity with node)



(1)

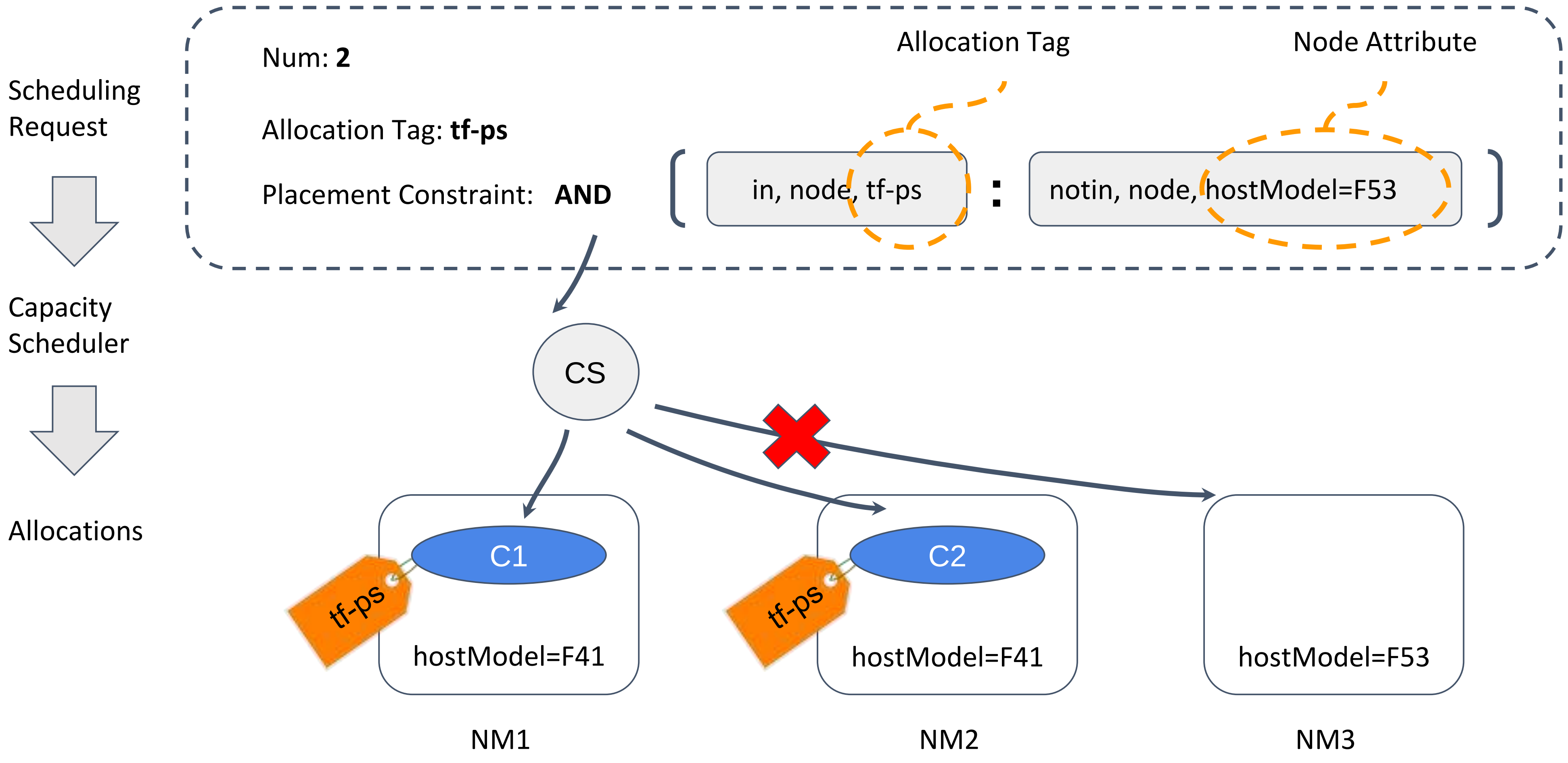


(2)



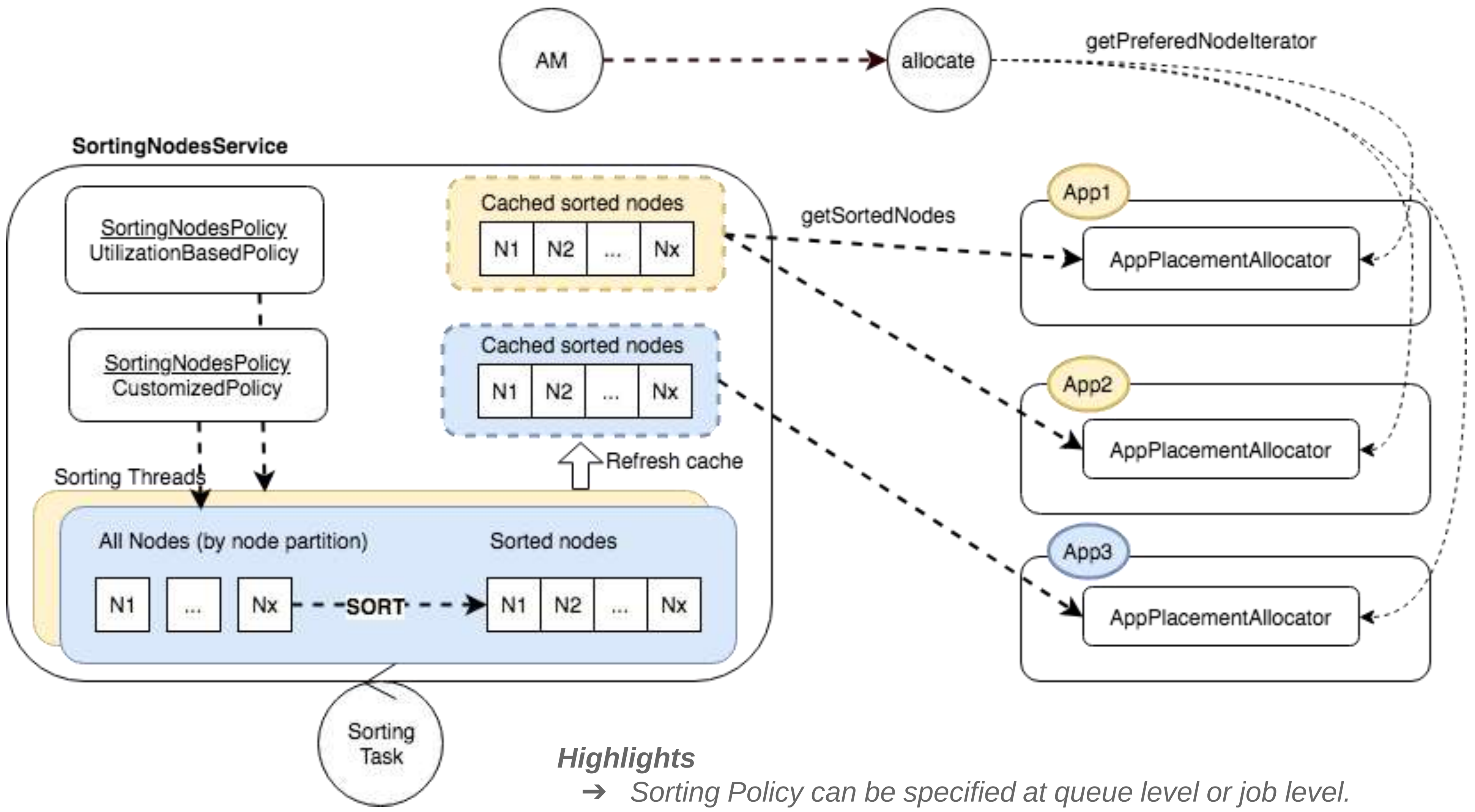
(3)

Placement Constraints



Advanced: Allocation tag Namespace, composite constraints, operators.
Related issues: YARN-6592, YARN-7812, YARN-3409.

Node Scorer



Highlights

- ➔ *Sorting Policy can be specified at queue level or job level.*
- ➔ *The sorting interval of each policy is configurable, if set to zero, it runs live-sorting.*

Rescheduler

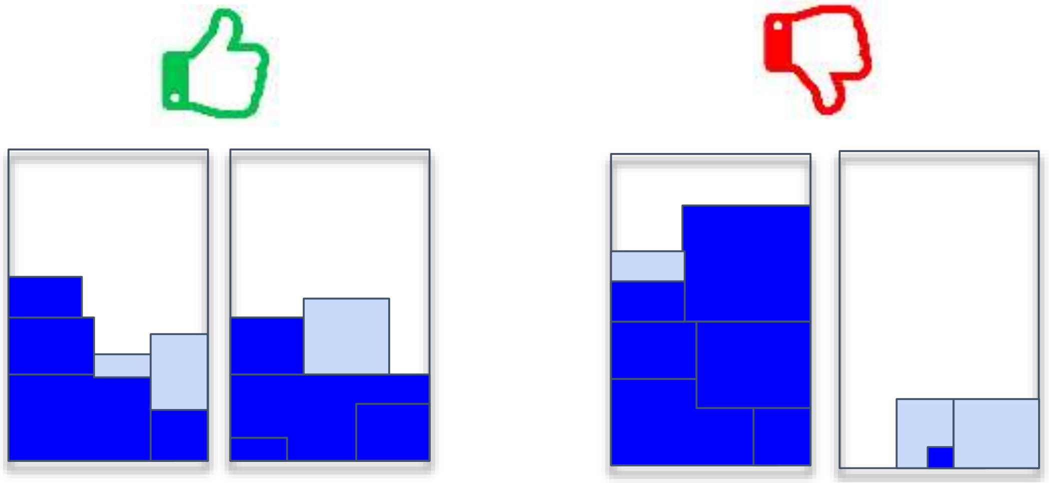
We not only concern about allocations at “scheduling” phase!

Objective

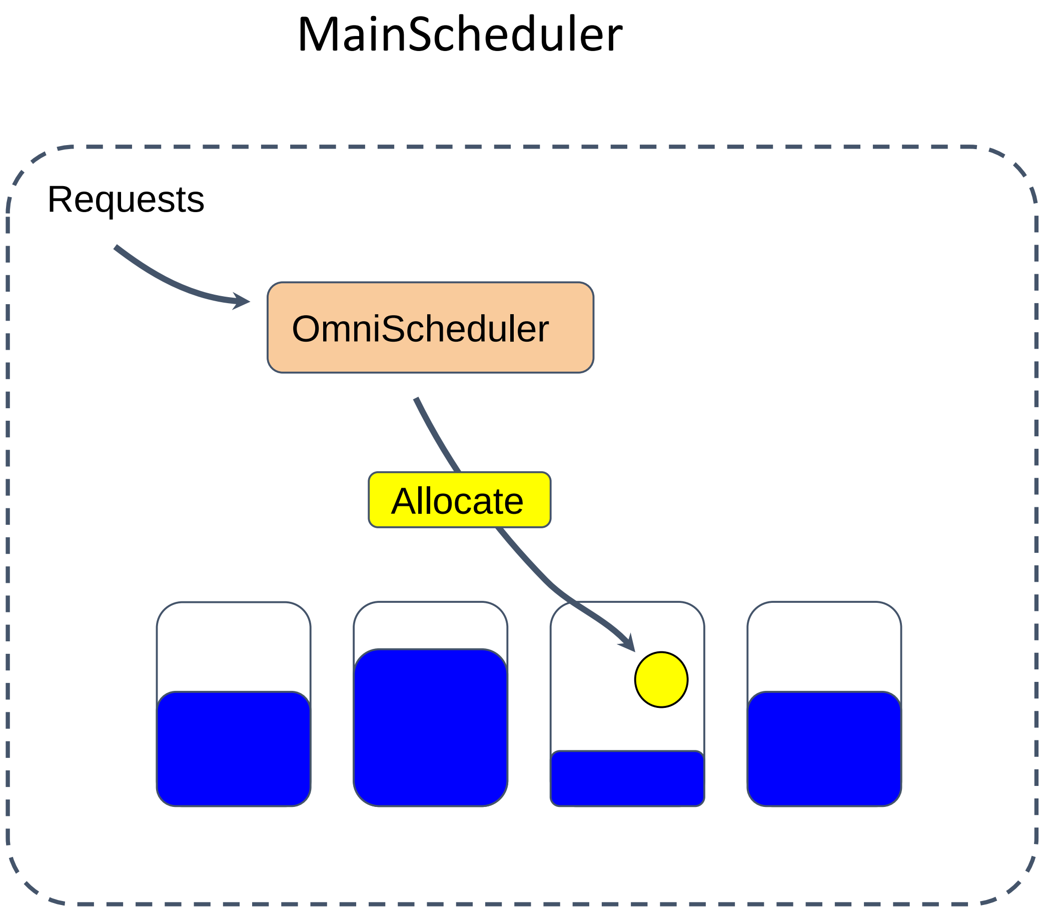
- ✓ Dynamically opmize container distributions on the cluster

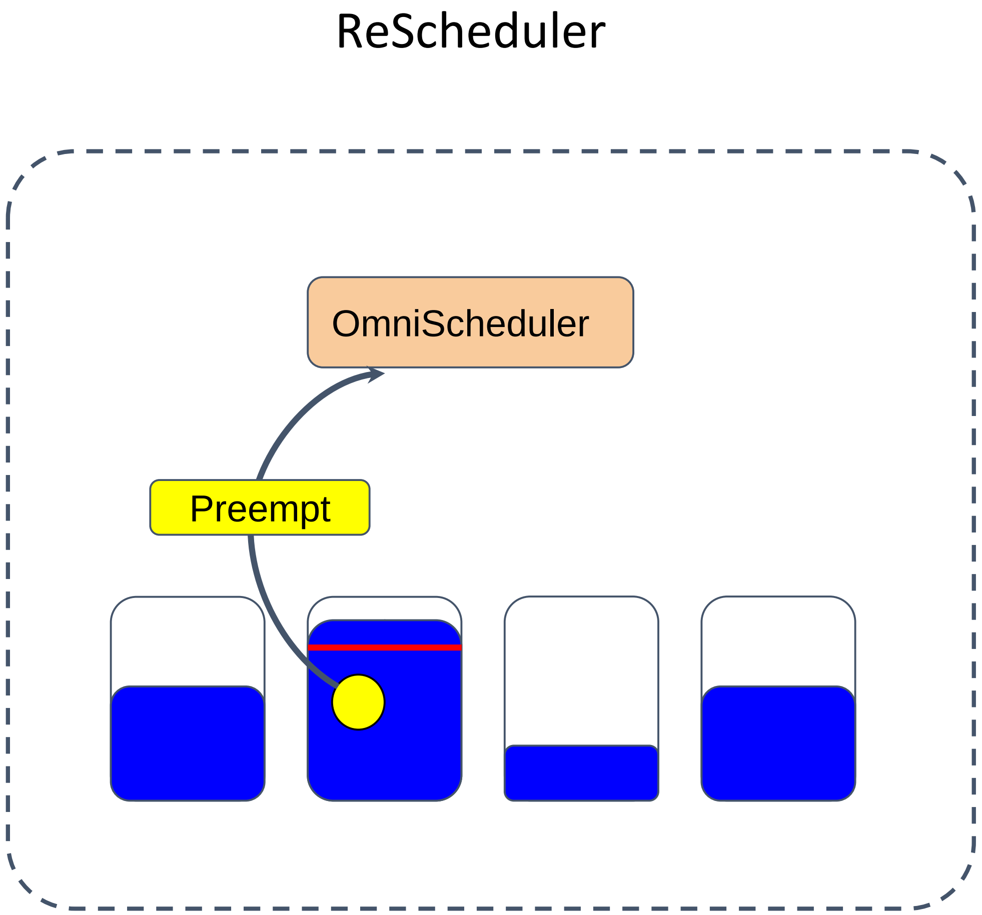
Our Use case:

- ✓ Eliminate Hotspot
- ✓ Eliminate Fragmentation (future)

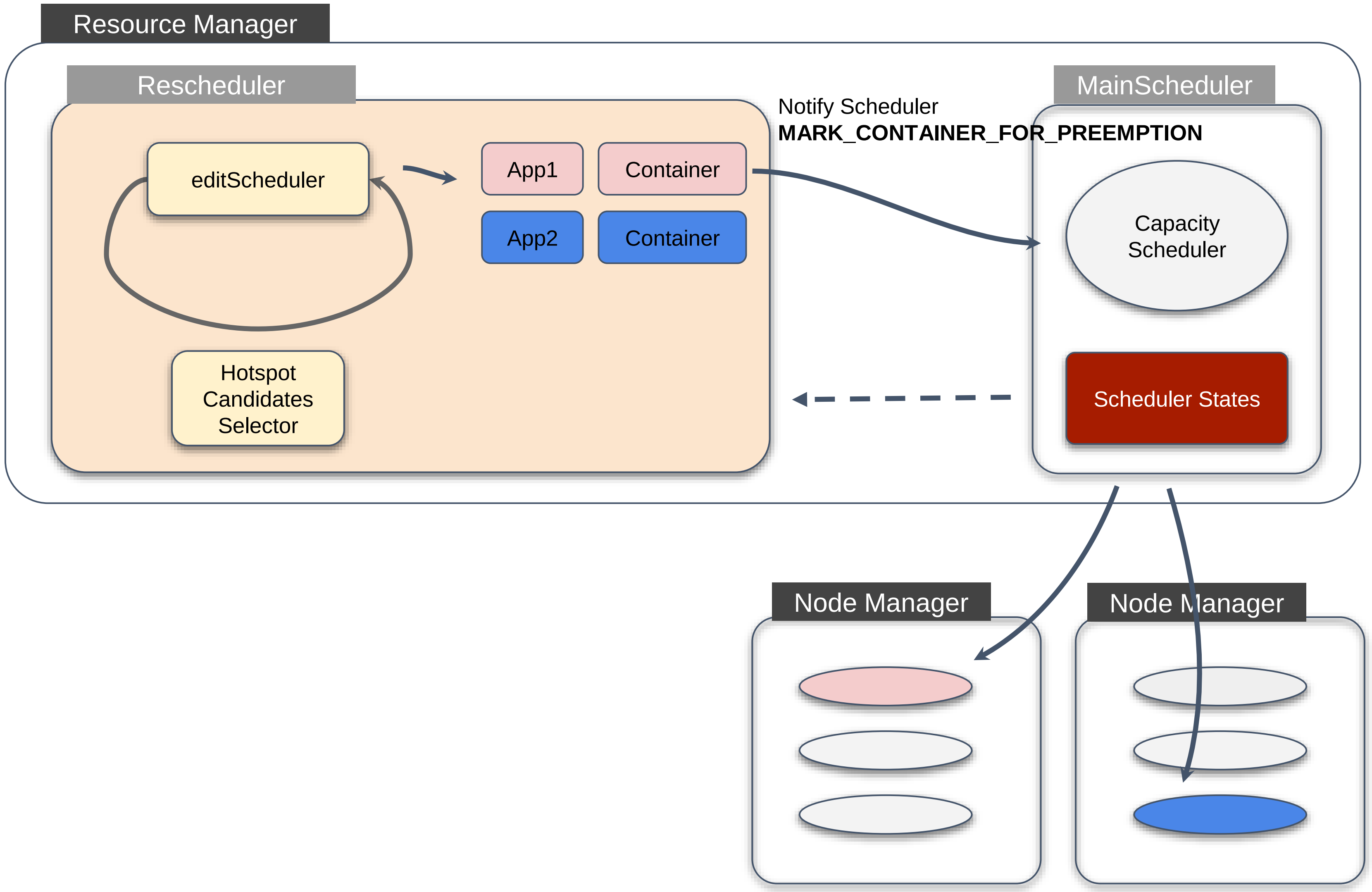


Elinimate Hotspots





Rescheduler - Eliminate Hotspot

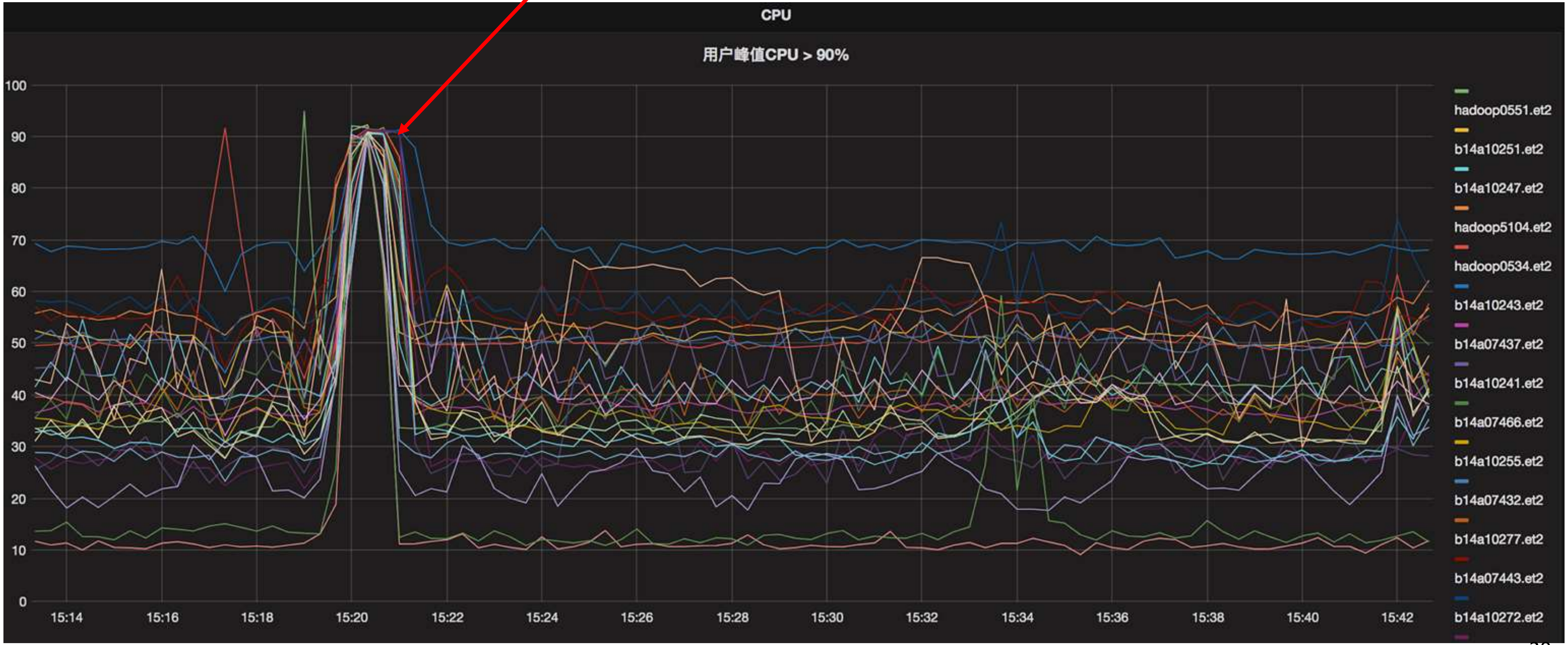


Highlights

1. High/Low water mark
2. Lazy marker: weaken the impact of momentary utilization
3. Observe mode

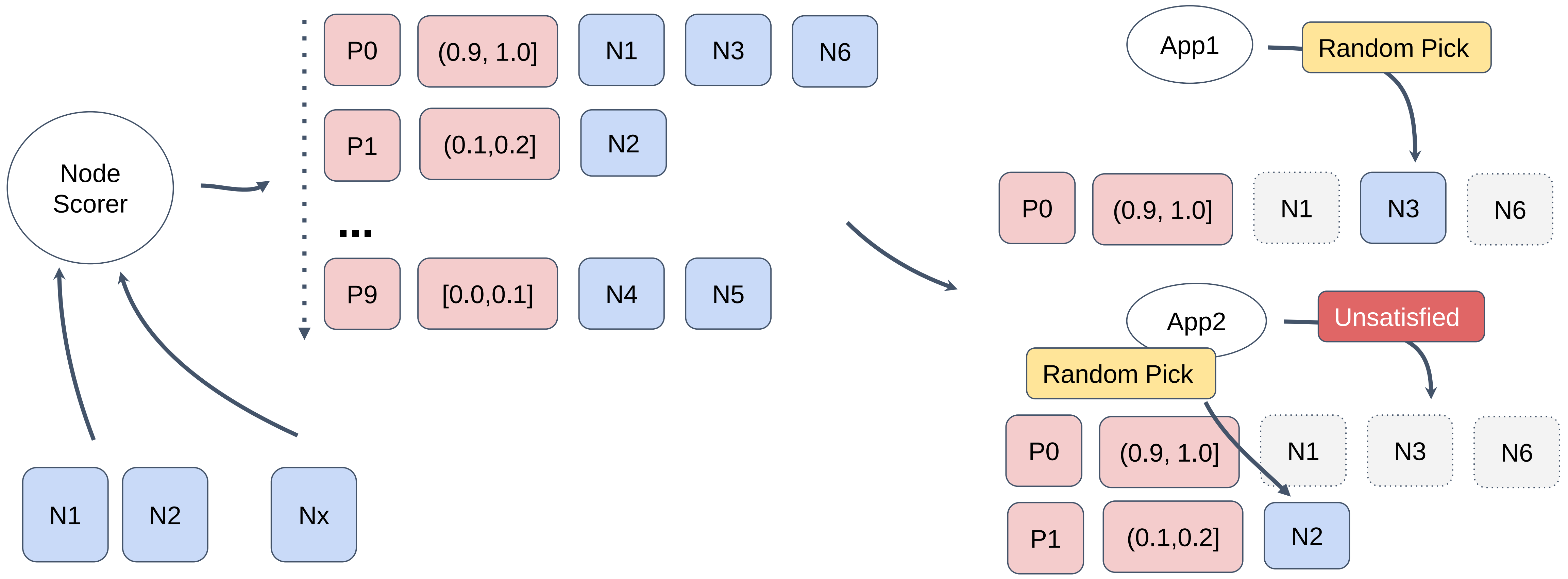
Result

Enable Node Scorer



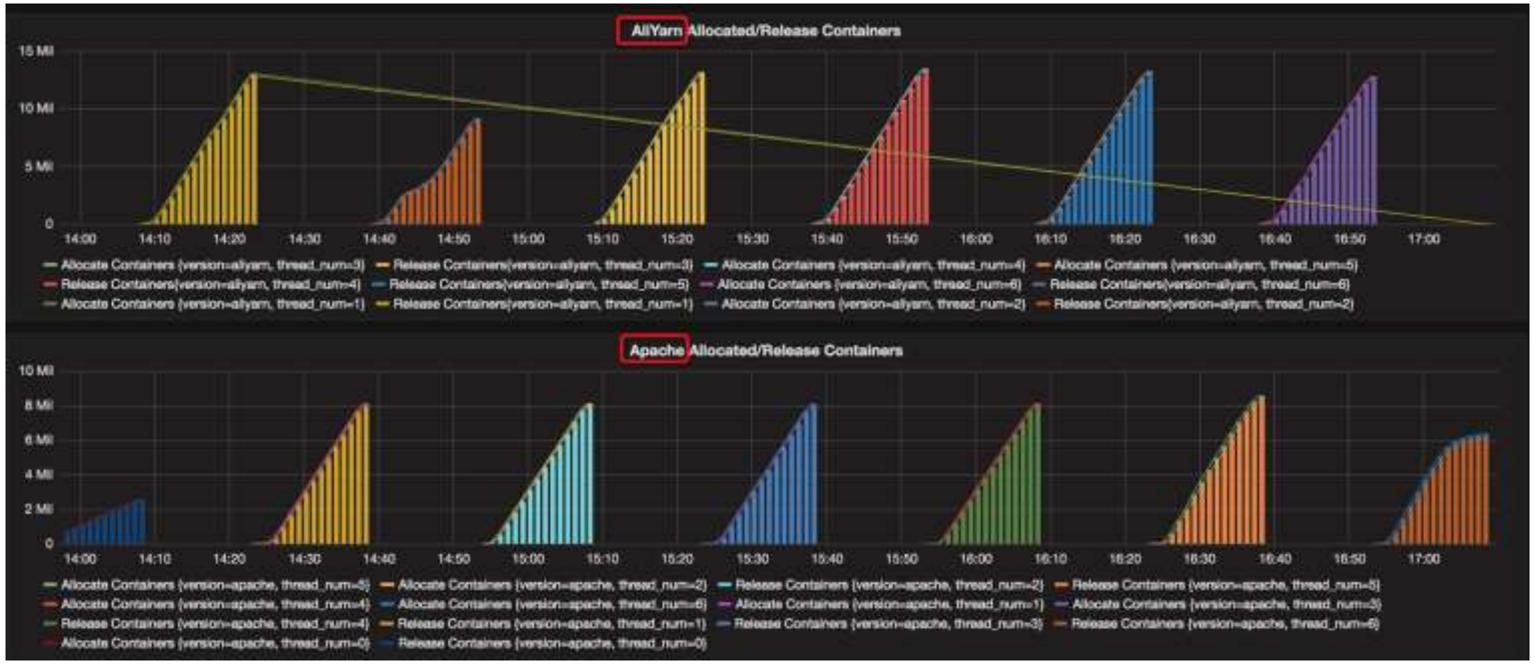
Reduce Allocation Proposal Conflicts

Allocate thread is fast, often tens of milliseconds, sorting is slow (depending on the number of nodes), possibly a large number of allocate threads will see same sorting result. If they both do allocate in order, that creates a lot of conflicts. Therefore, we optimized the sorting strategy to “**Partition Score Sort**”.



Performance Improvements

- Throughput improvement
  - Allocate a batch of request in each allocate iteration
- Optimize RESERVE container behavior
  - Lazy Reservation: Do not reserve container until all candidates cannot satisfy the request
  - Accelerate reservation allocation: attempt to allocate reserved container when heartbeat arrives



Cluster size: 10K nodes

Node capability: 128gb, 128vcore

Workload: 23.5K Job, 1000 task per job

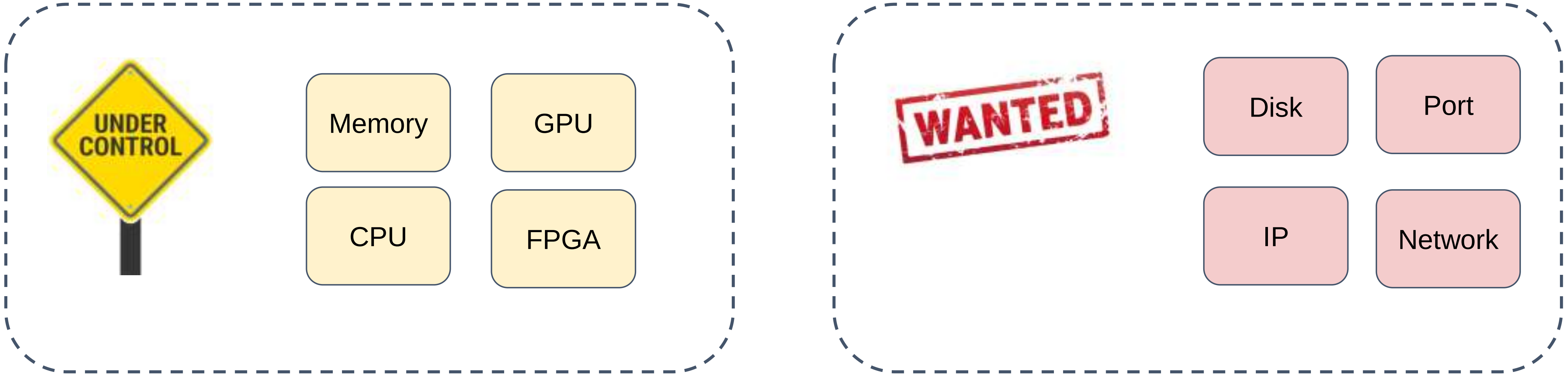
Task: mem 1gb - 8gb, exec 5s - 10s

Improvements for Online Service

Part III

YARN - Resource Management System

A Resource Management System but not managing all resources ?!



Current resource types is not able to support these!

Multi-dimensional Resources

COUNTABLE

```
{
 "name" : "memory",
 "units" : "mb"
 "value" : "1024"
}
```

New

SET

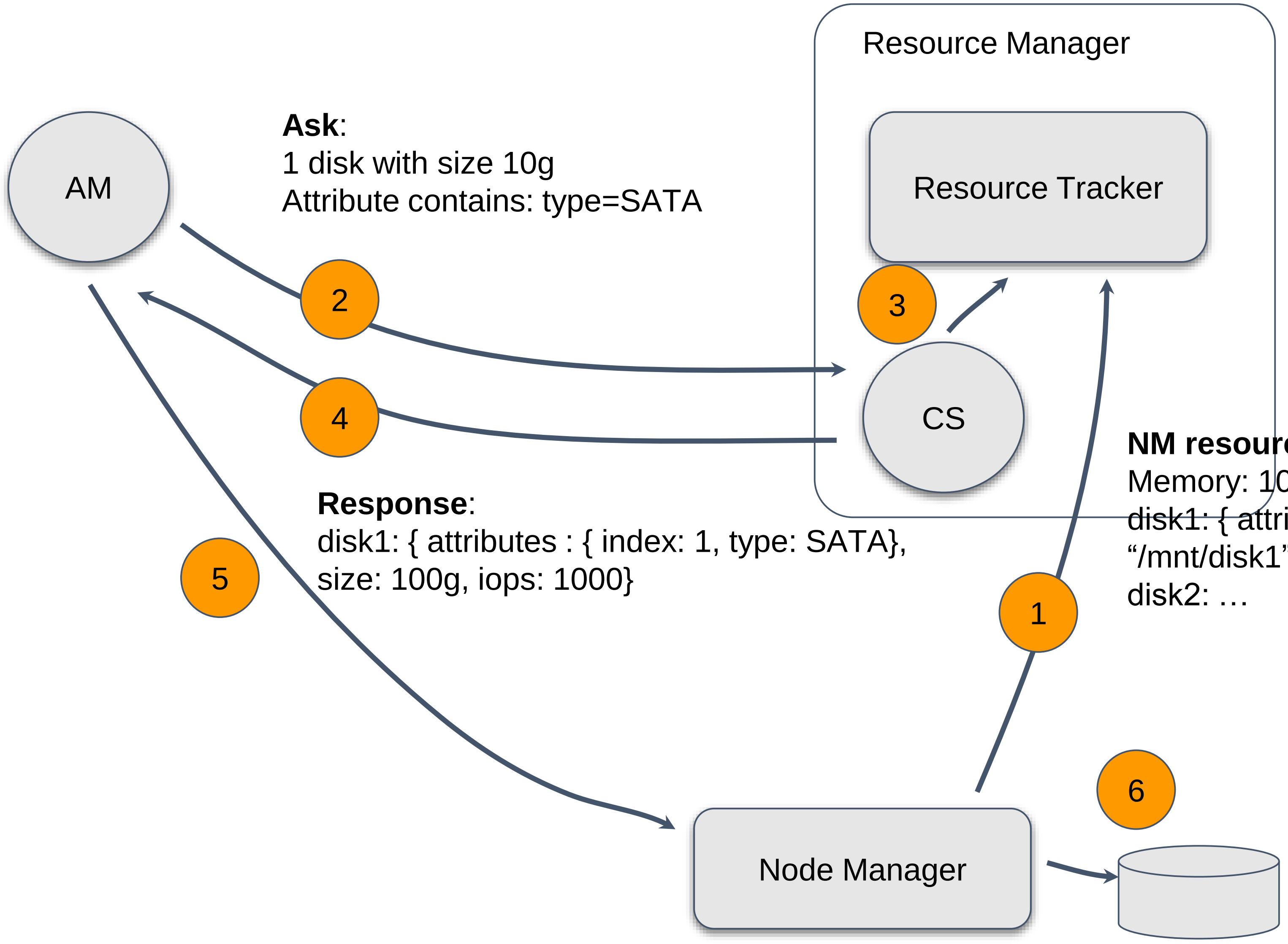
```
{
 "name": "IP",
 "values": ["10.100.0.1", "0.100.0.2", "100.100.0.3"]
}
```

New

RESOURCE SET

```
disks : [ {attributes : {"type":"sata", "index":"1"}, size : 100, iops : 100, ratio : 100},
          {attributes : {"type":"ssd", "index":"2"}, size : 100, iops : 100, ratio : 100},
          {attributes : {"type":"ssd", "index":"9999"}, size : 40, iops : 40, ratio : 40}]
```

Disk Resource - Workflow



Ask:
1 disk with size 10g
Attribute contains: type=SATA

Response:
disk1: { attributes : { index: 1, type: SATA},
size: 100g, iops: 1000}

NM resource:
Memory: 10g, Vcore: 100
disk1: { attributes : { index: 1, type: SATA, localDir: "/mnt/disk1"}, size: 100g, iops: 1000}
disk2: …

Container Launch:
disk1: { attributes : { index: 1, type: SATA},
size: 100g, iops: 1000}

Workdir: /mnt/disk1/application_xx/...

Resource Isolation

Quota

Share

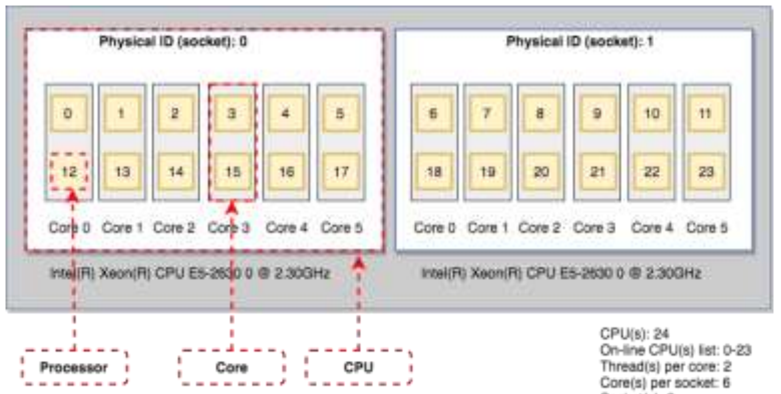
Set

Quota

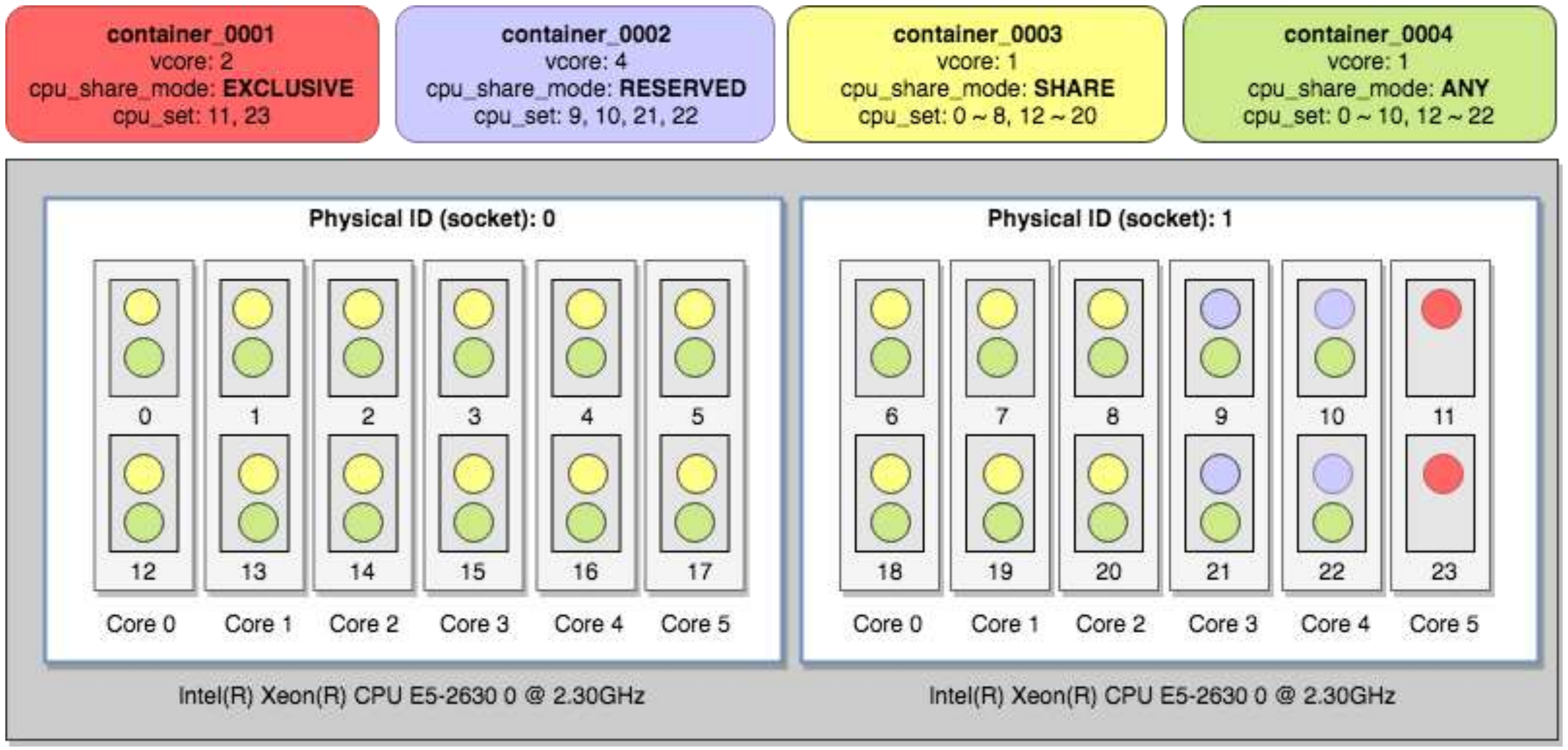
Priority

Resource Isolation - Cpuset

CPU context switch impacts the latency of a (Latency Sensitive) LS task.
Our solution: support **cpu_share_mode** via **cgroups cpuset.**



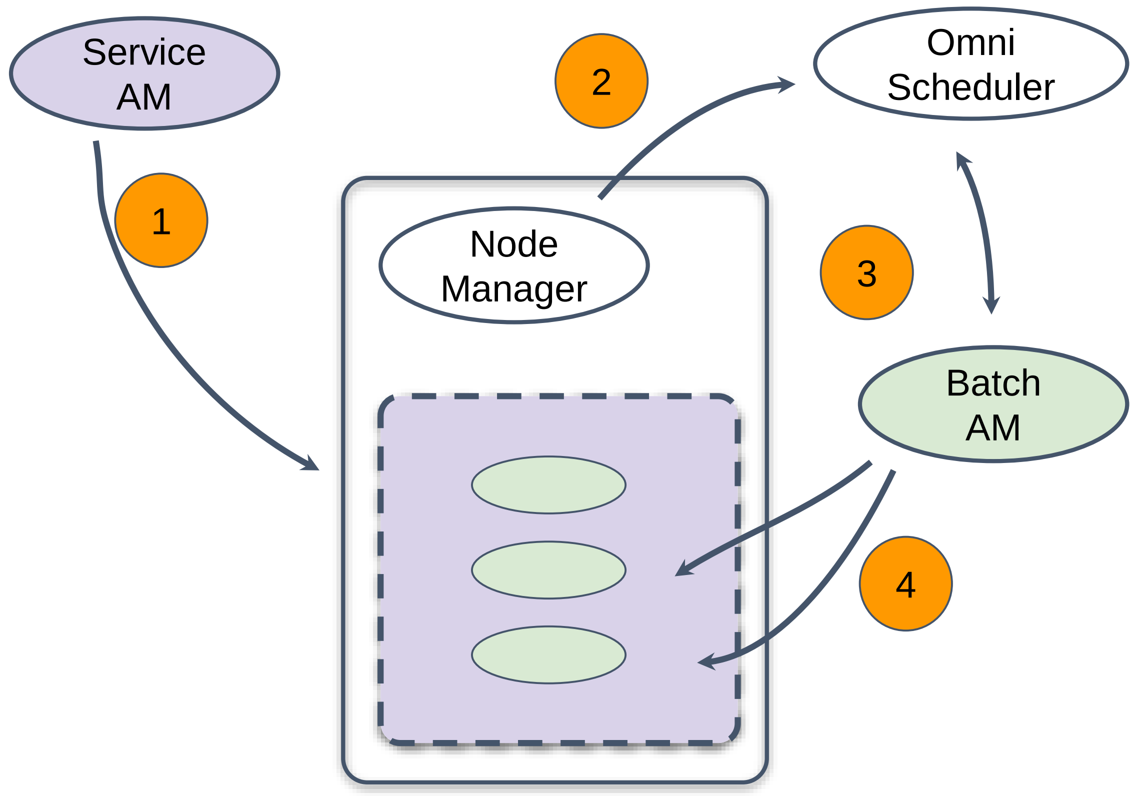
```
CPU(s): 24
On-line CPU(s) list: 0-23
Thread(s) per core: 2
Core(s) per socket: 6
Socket(s): 2
```

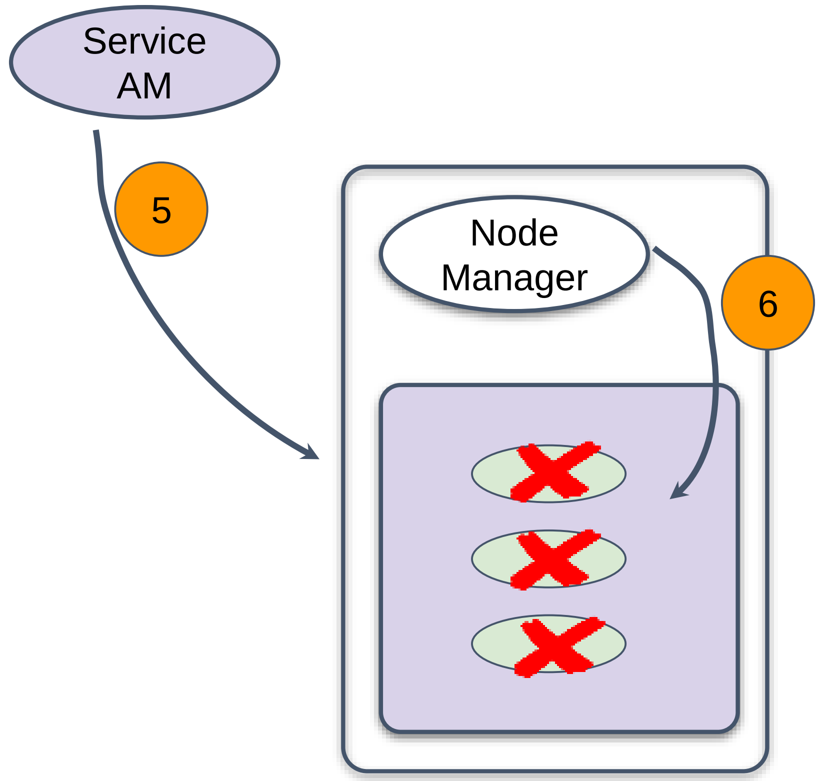


Related issue: YARN-8320

Resource Oversubscription - A Step Forward

Reserve allocation for me, Share resources to O containers





Future Work

Part IV

Future Work

- Rescheduler
  - Leverage ML to minimize the cost of movements
- Comprehensive Preemption
- Performance
  - High throughput & low latency
- Online service features
  - Volumn
  - Pod

THANKS/感谢聆听

--------- Q&A Section --------