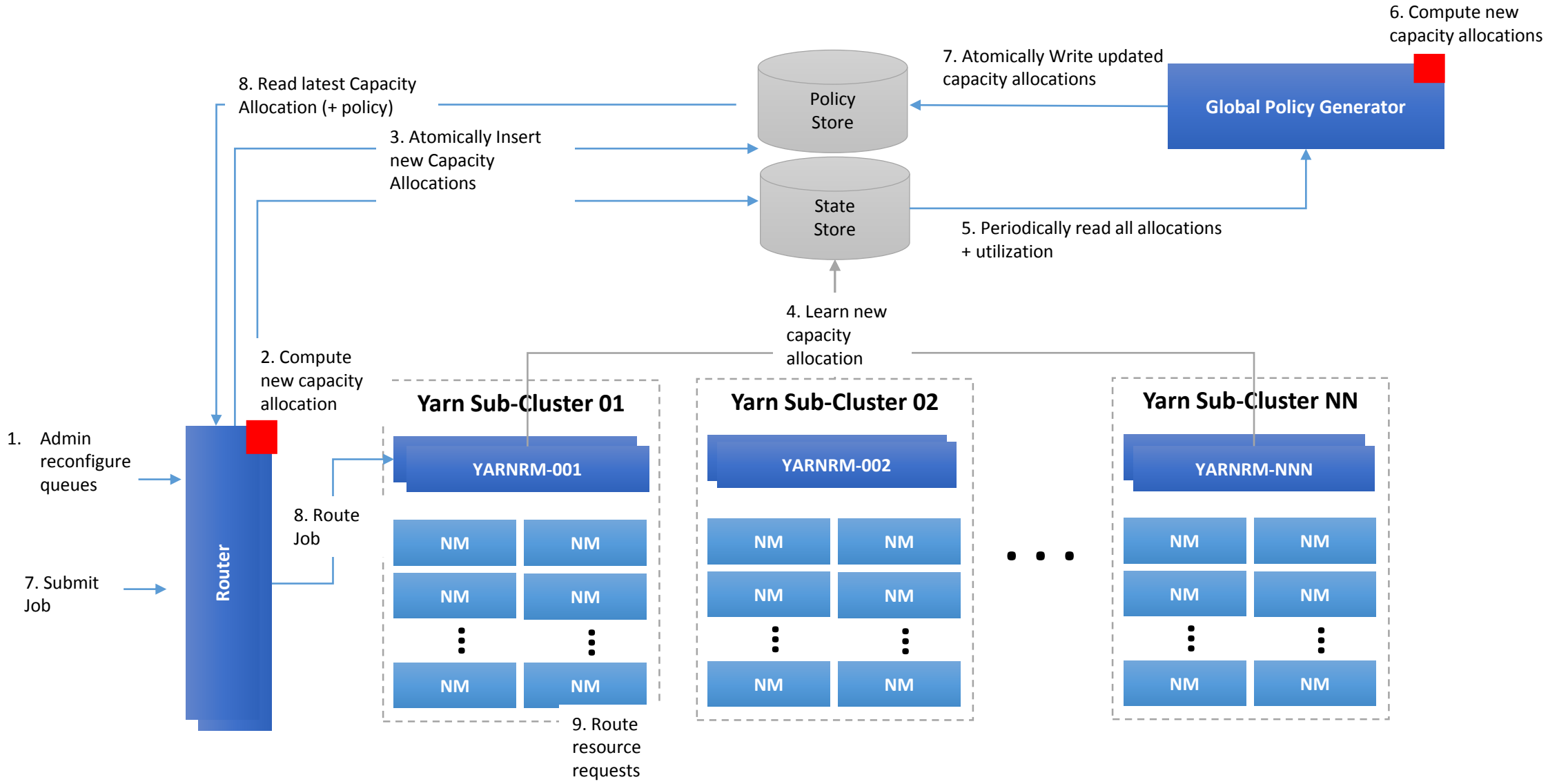
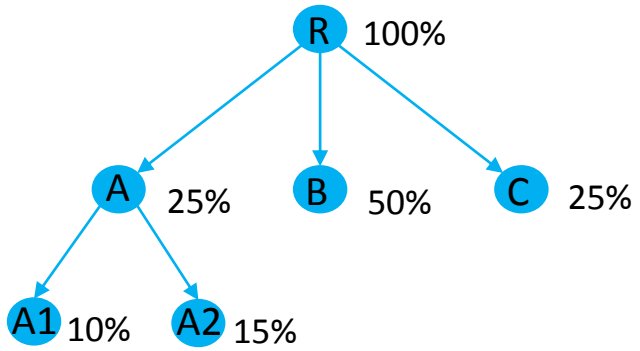
YARN Federation: Capacity Allocation and Admission Control

Carlo Curino, Alexey Tumanov, Subru Krishnan, Giovanni Fumarola
Cloud and Information Services Lab (CISL) -- Microsoft

Federation Architecture



Mapping to local queue: spectrum



SC1 ?

SC2 ?

SC3 ?

SC4 ?

Full Partitioning

fairness - +

utilization - +

scalability + -

tenant isolation + -

Full Replication

Federation Scalability Reminder

YARN Scalability:

Bottleneck on RM *scheduling bandwidth*

(consumed) scheduling bandwidth is proportional to #nodes, #jobs, heartbeats rates

#nodes_per_rm = #nodes / #sub_clusters

#jobs_per_rm = #jobs / #sub_clusters

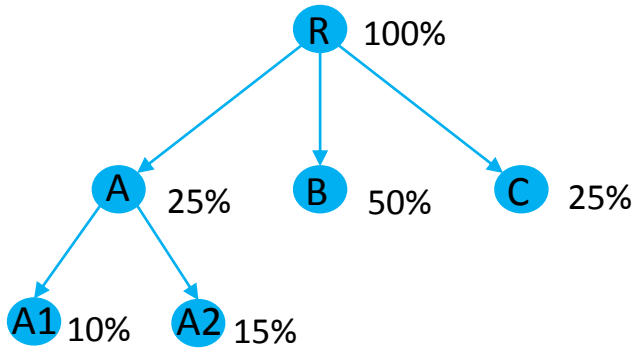
Heartbeat rate = constant

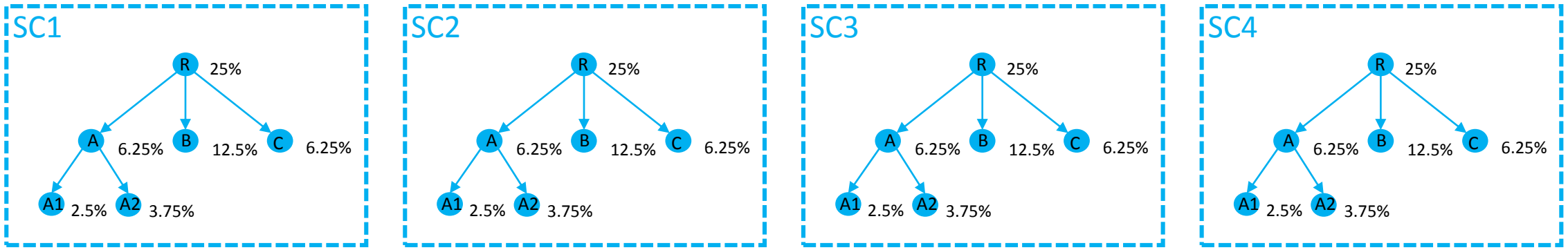
#nodes_per_rm = #nodes / #sub_clusters

#jobs_per_rm = #jobs

Heartbeat rate = constant

Spectrum of options: Full Replication



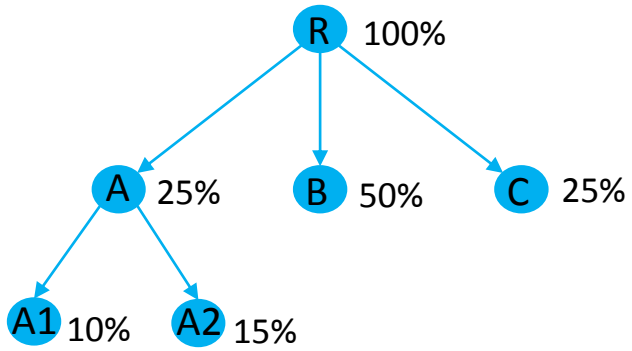


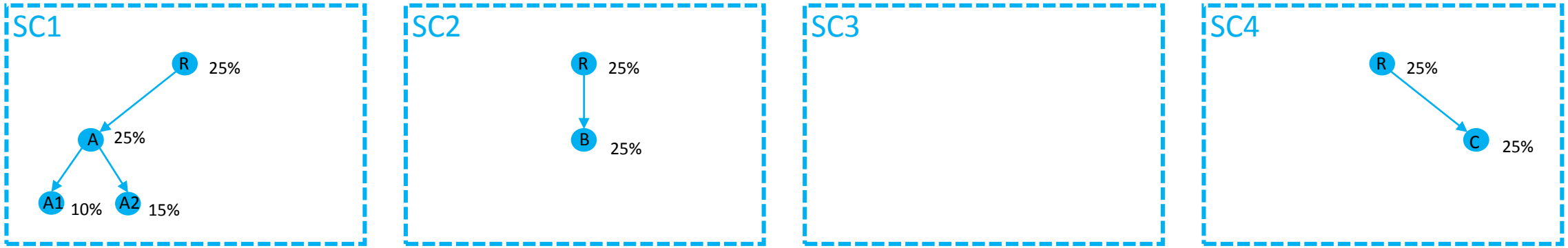
Pros: Simple/symmetric, provably fair (if all jobs broadcast demand), resilient to impairments

Cons: scalability in #jobs, locality

Heuristic improvements: 1) RM data structures, 2) variable heartbeat rate, 3) broadcast only for large jobs

Spectrum of options: Full Partitioning

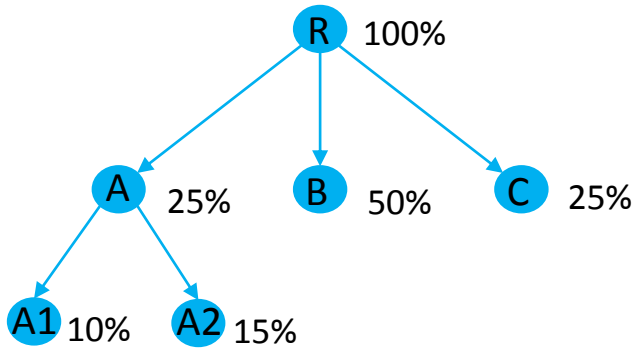


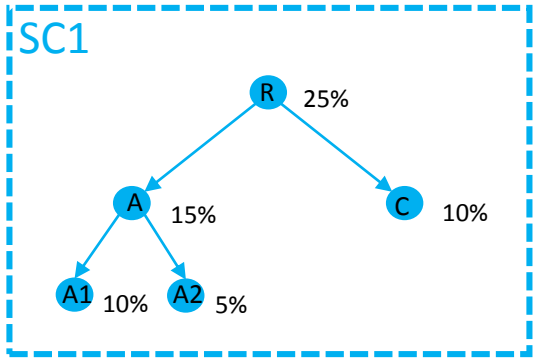


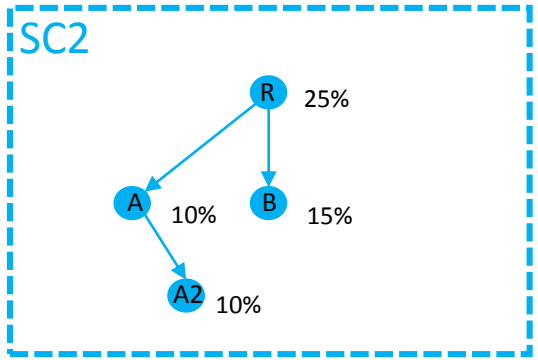
Pros: perfect scale-out

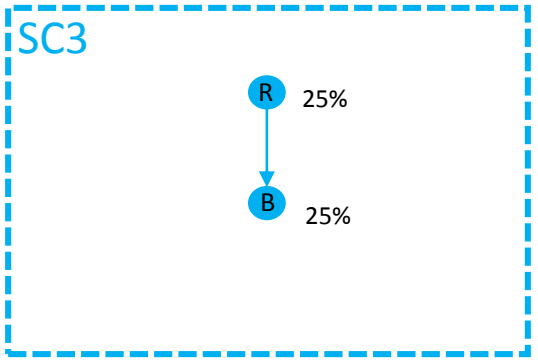
Cons: fragmentation/utilization issues, uneven impact of localized capacity impairment, largest tenant must fit in one sub-cluster

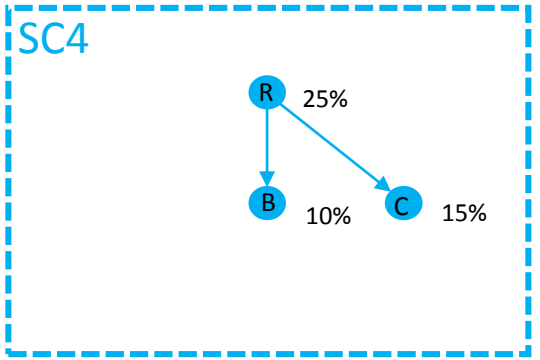
Spectrum of options: Partial Replication











Pros: Trade-off between advantages of two extremes

Cons: Complexity of rebalancing

Reasons to “rebalance” local queues

Changing of input (global queue structure):

operators update of queues, might trigger rebalancing

Capacity impairments:

sub-clusters being un-evenly affected by loss of capacity

Utilization/Fairness:

demand imposed by jobs is “uneven” w.r.t. allocation

Intuition for "rebalancing" algorithms

Linear Programming Formulation:

Constraints: CPU, RAM, disk, net on each sub-cluster

scheduling bandwidth on each sub-cluster

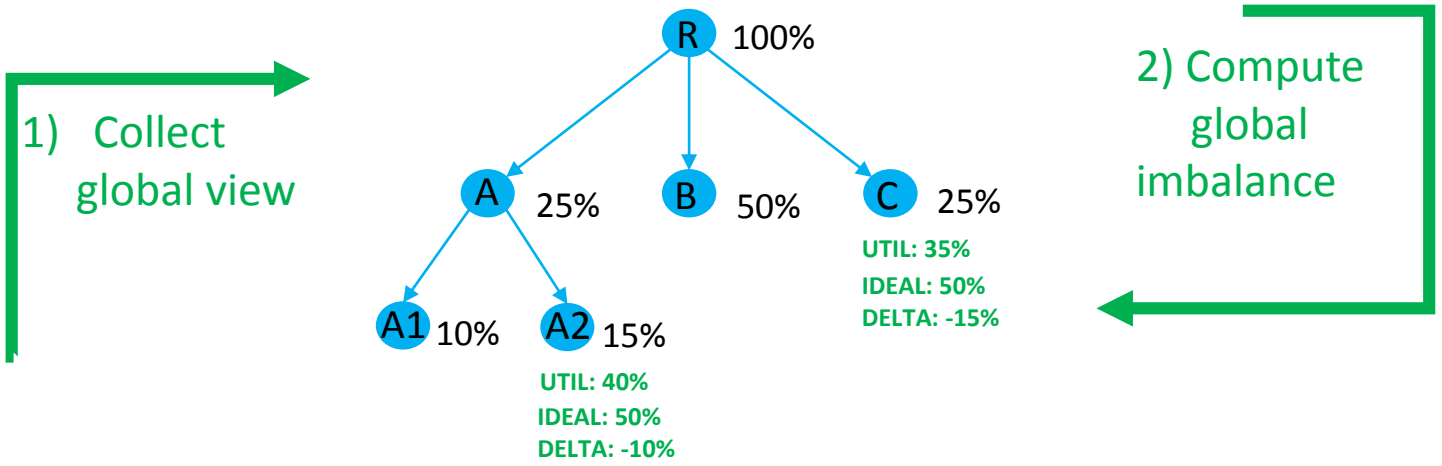
all queues are allocated

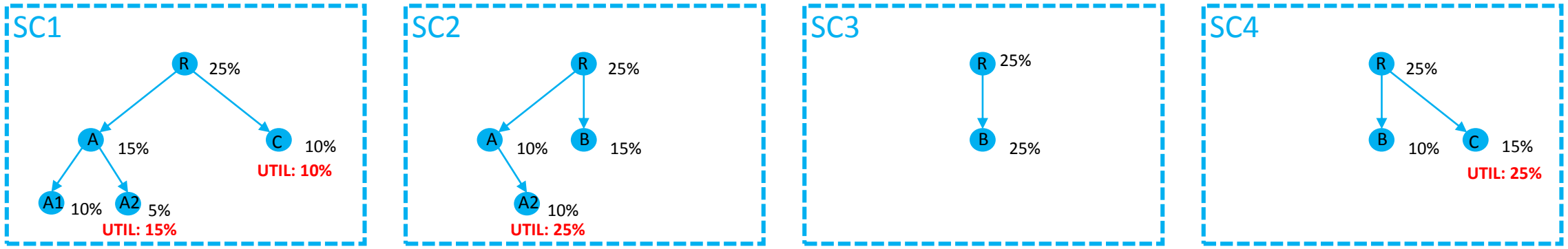
Opt-Function: max-min fairness + (locality, stability, headroom biases)

Heuristic rebalancing:

1) Collect load on each sub-cluster and "add-up" in global view
2) Find "global" imbalance (use the CS preemption algo)
3) Do local edits (add/remove/resize) queues to address imbalance
4) Eliminate/merge queues to fit within sub-cluster scheduling bandwidth

Heuristic intuition (by example)

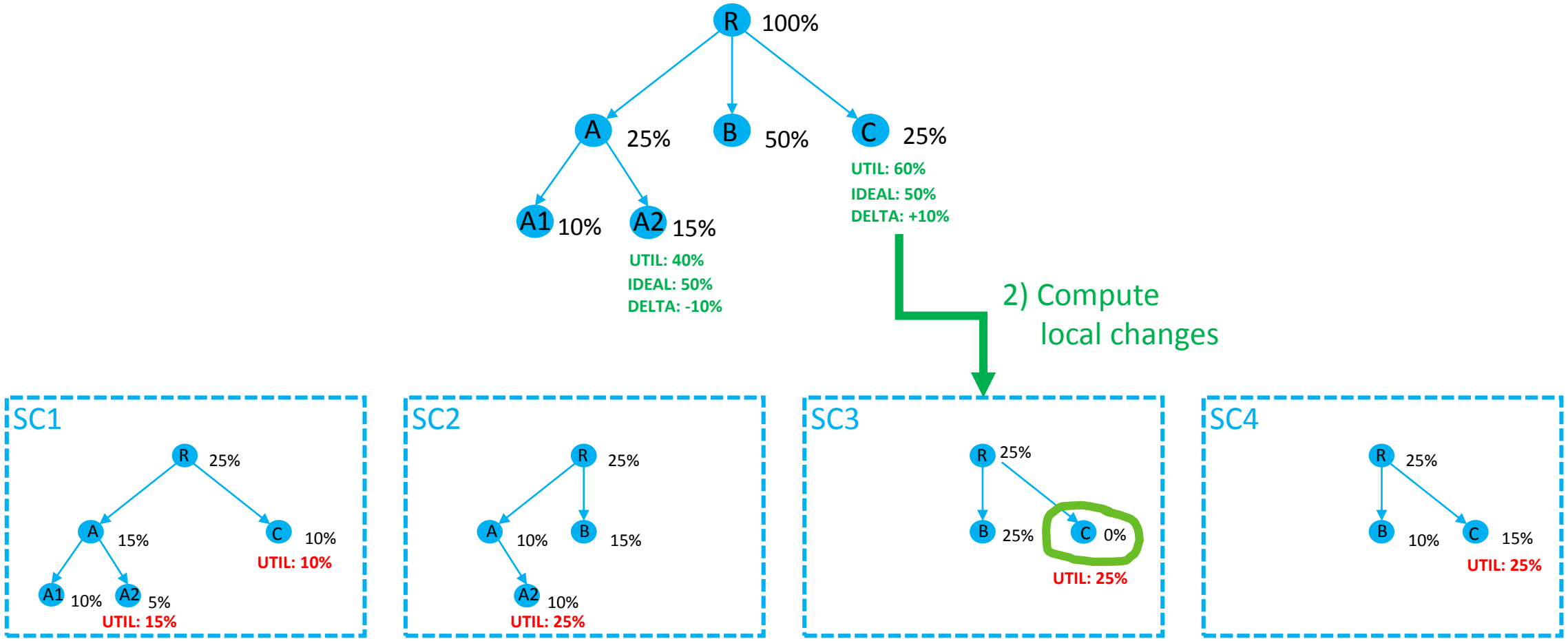




Input: B,A1 has zero demand, A2 and C has huge demand

Problems: SC3 is underutilized, A2 and C is are below fair-share (un-evenly)

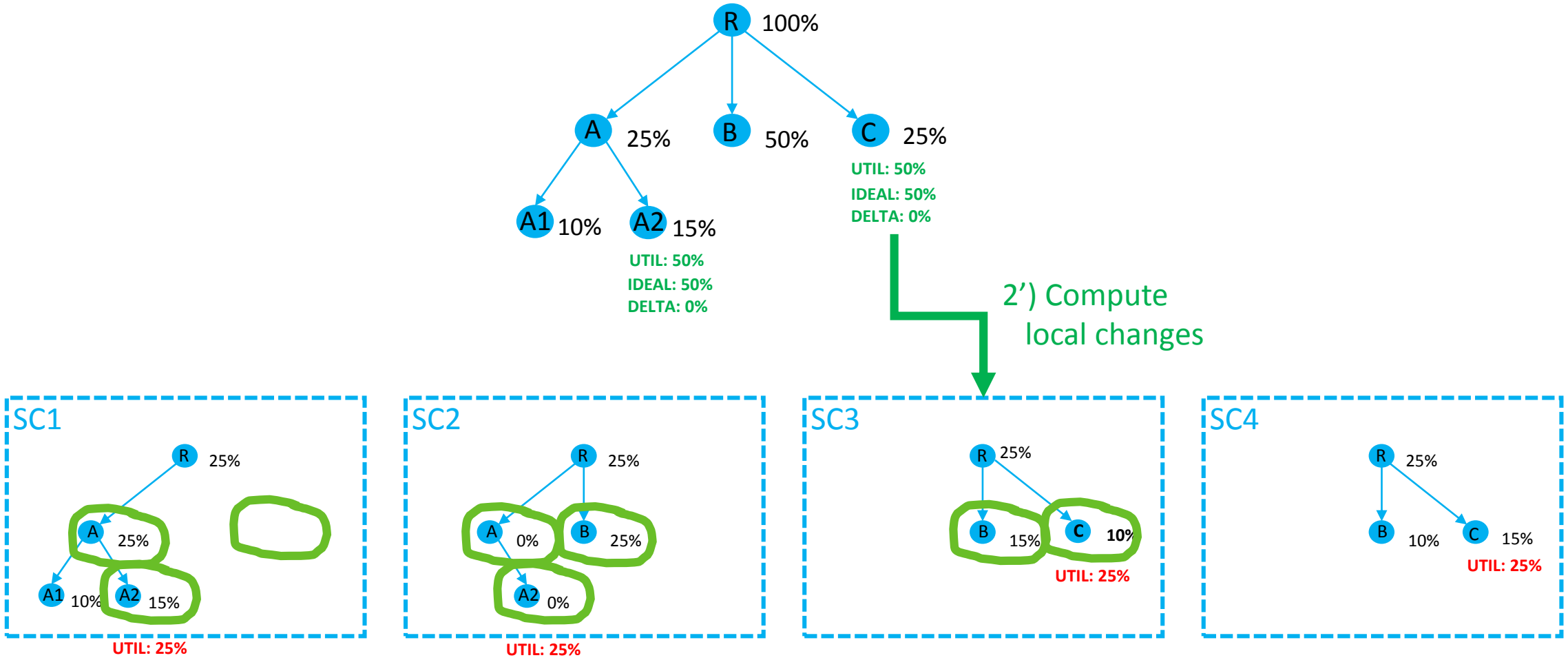
Heuristic intuition (by example)



Pros: SC3 utilization is high, minimal queue edit

Cons: A2 is still under fair-share, (C is over fair-share)

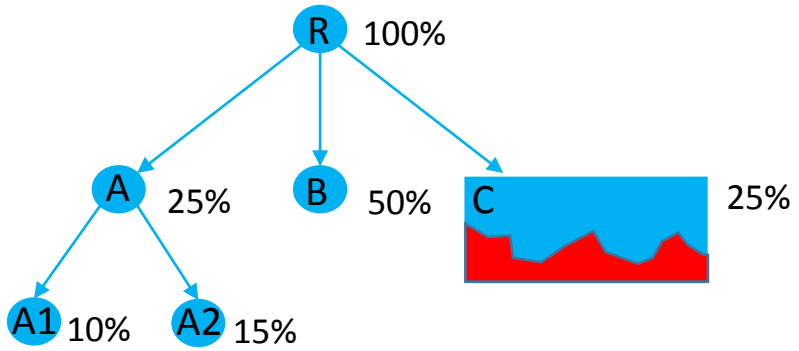
More global "rebalancing"

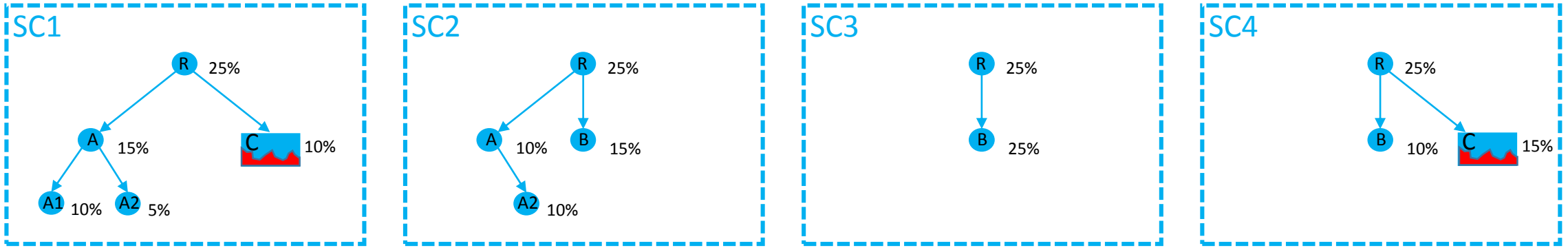


Pros: Perfect utilization, Perfect fair-share

Cons: Expensive "edit" of queues (preemption, locality, etc..)

Reservations





Intuition:

Natural match with (ongoing) extension to “node-labels” (SC1, SC4)

When “rebalancing” queues, we will need to invoke “replanner”

Planning

Start from Fully Replicated

Best practical trade-off to start (might be enough forever)

Work on RM scalability in #jobs + AMRMProxy heuristics + variable heartbeat rate

Explore Heuristics/Global rebalancing

As needed based on our scalability findings