# Downscaling: The Achilles heel of Autoscaling Apache Spark Clusters

Prakhar Jain
Sourabh Goyal

# Agenda

- Why Autoscaling on cloud?
- How nodes in spark cluster are used?
- Easy upscale, Difficult downscale
- Optimizations

# Autoscaling on cloud

- Cloud for compute provides elasticity
  - Launch nodes when required
  - Take them away when you are done
  - Pay-as-you-go model. No long term commitments.

- Autoscaling clusters are needed to use this elastic nature of the cloud
  - Add nodes to the cluster when required
  - Remove nodes from the cluster when the cluster utilization is low

- Use **Cloud object stores** to store the actual data and just use the elastic clusters on the cloud for data processing/ML etc

# How are nodes used in a spark cluster?

Nodes/Instances in a Spark cluster are used for

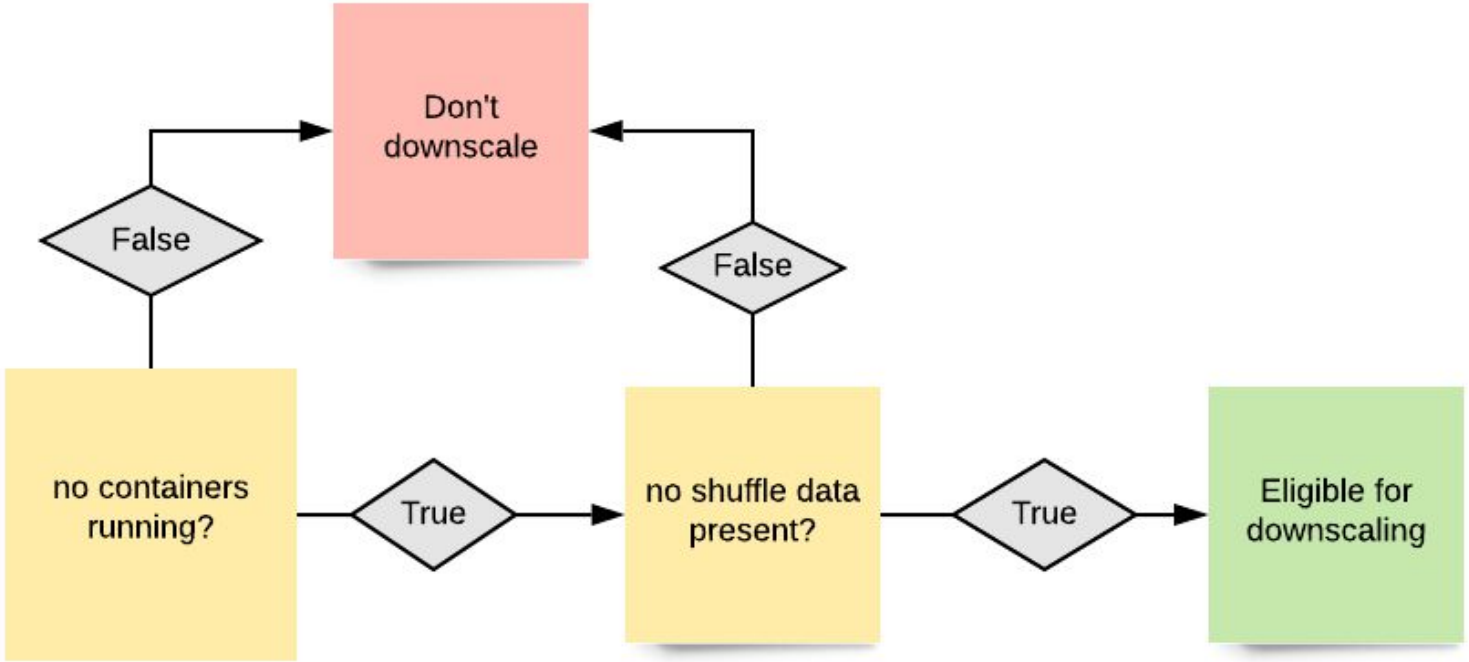
- Compute
  - Executors are launched on these nodes which do the actual processing of Data
- Intermediate temporary data
  - nodes are also used as temporary storage e.g. for storing temporary application related shuffle/cache data
  - Writing temporary data to object store (like s3 etc) deteriorates the overall performance of the application

# Upscale easy, downscale difficult

- Upscaling a cluster on cloud is easy
  - When the workload on the cluster is high, simply add more nodes
  - Can be achieved using simple Load balancer

- Downscaling nodes are difficult
  - No running containers
  - No shuffle/cache data stored on disks
  - Container fragmentation within cluster nodes
  - Some nodes have no containers running but are used for storage and vice versa

# Factors affecting downscaling of a node



# Terminology

Any cluster generally comprises of following entities:

- Resource Manager
  - Administrator for allocating and managing resources in a cluster. e.g. YARN/Mesos etc
- Application Driver
  - Brain of the application
  - Interacts with Resource Scheduler and negotiates for resources
    - Ask for executors when needed
    - Release executors when not needed
  - e.g. Spark/Tez/MR etc
- Executor
  - Actual worker responsible for running smallest unit of execution - task

# Current resource allocation strategy

Problem: Executors fragmentation

Current allocation strategy
allocates on emptier nodes first

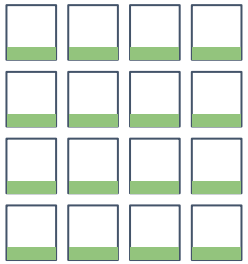
# Can we improve?

- Packing of executors

## Priority in which jobs are allocated to nodes in Qubole Model

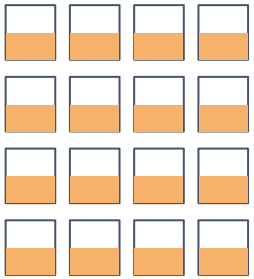
2

Low Usage



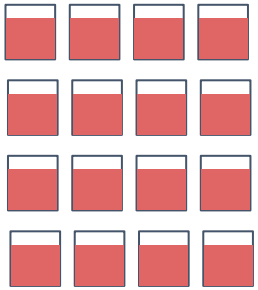
1

Medium Usage



3

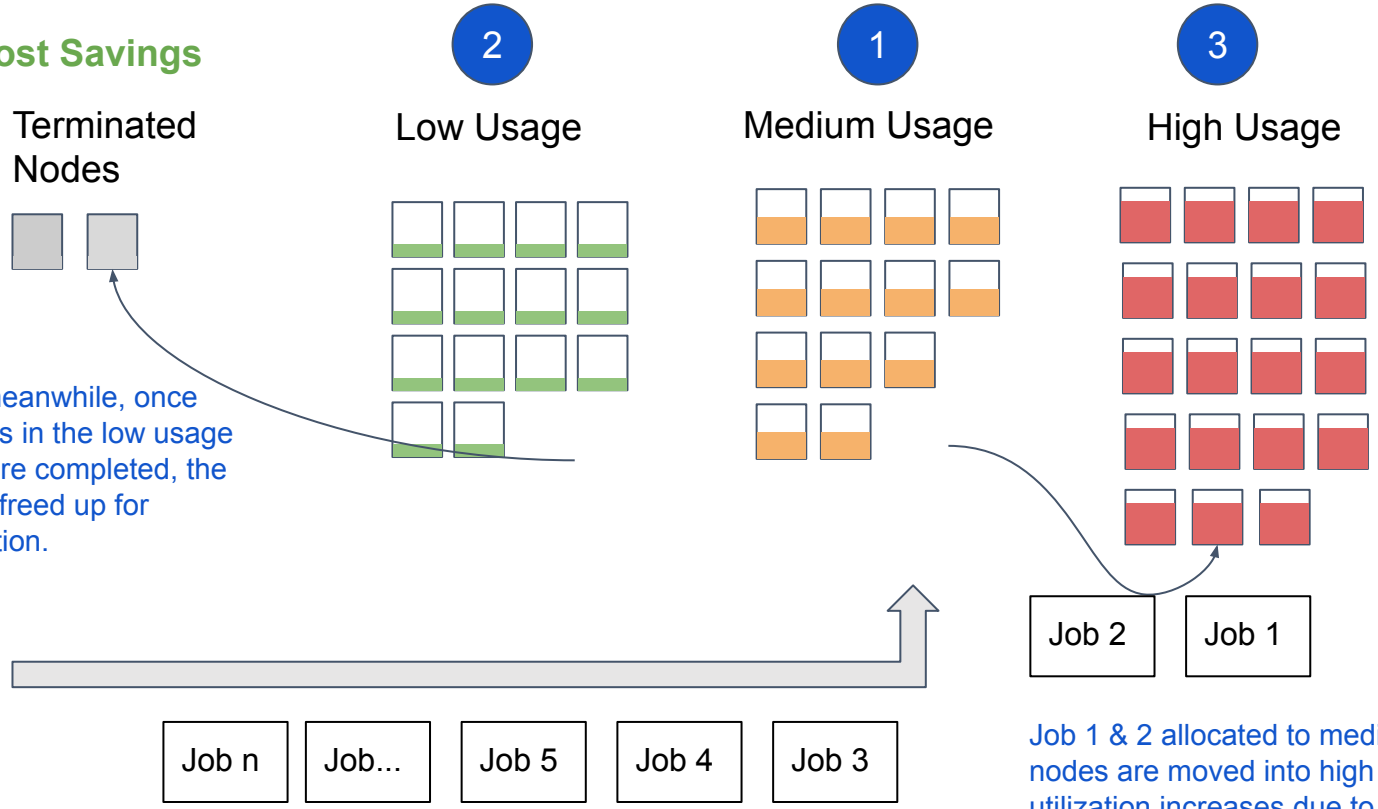
High Usage



Jobs are prevented from being assigned first to low usage nodes, instead priority is given to medium usage nodes.This ensures that low usage nodes can be downscaled.

Job n | Job... | Job 5 | Job 4 | Job 3 | Job 2 | Job 1

## Cost Savings



## Cost Savings

Downscaled Nodes

2

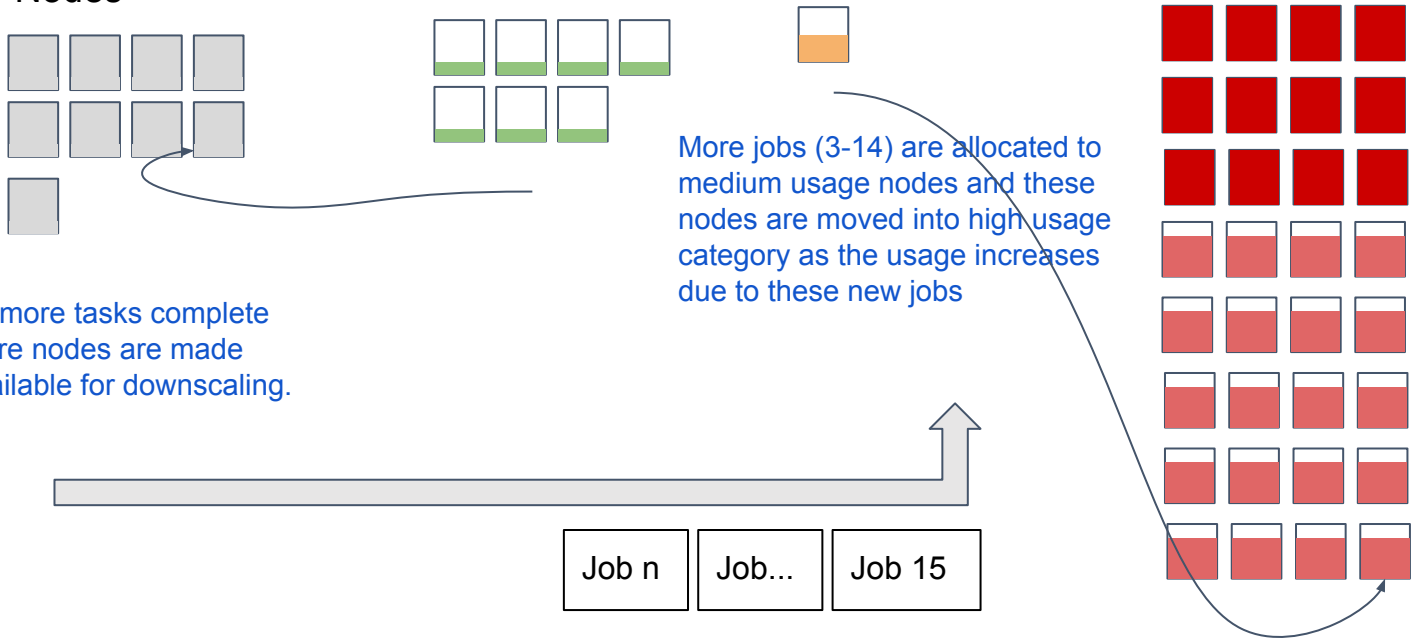
Low Usage

1

Medium Usage

3

High Usage



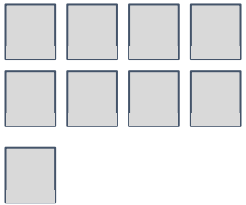
More jobs (3-14) are allocated to medium usage nodes and these nodes are moved into high usage category as the usage increases due to these new jobs

As more tasks complete more nodes are made available for downscaling.

Job n | Job... | Job 15

## Cost Savings

Terminated Nodes

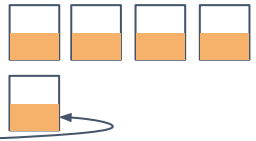


2

Low Usage

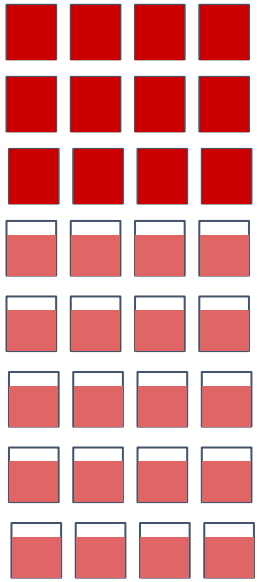
1

Medium Usage



3

High Usage

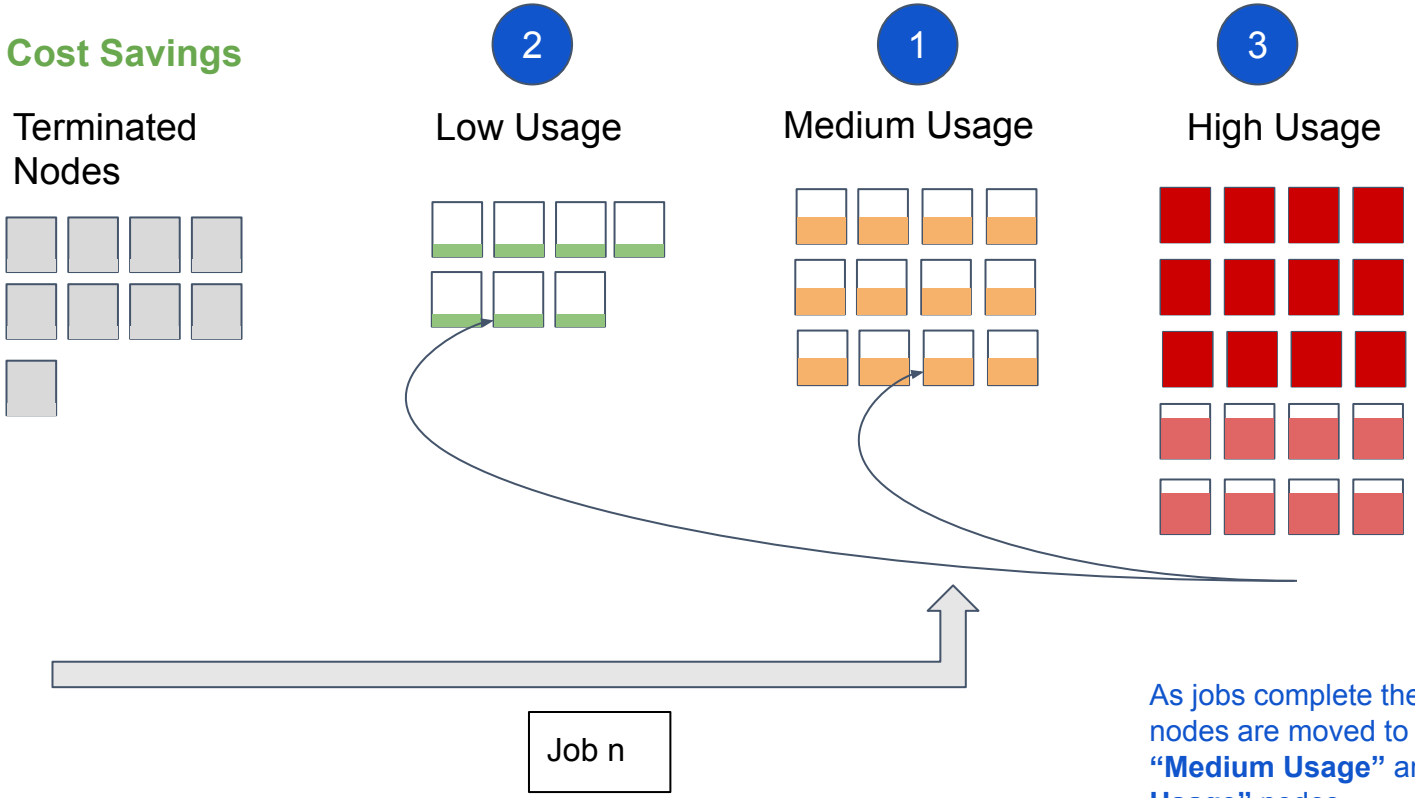


As medium usage nodes are reduced, jobs are allocated to **“Low Usage”** nodes and these nodes are moved into the **“Medium Usage”** Nodes

Job n

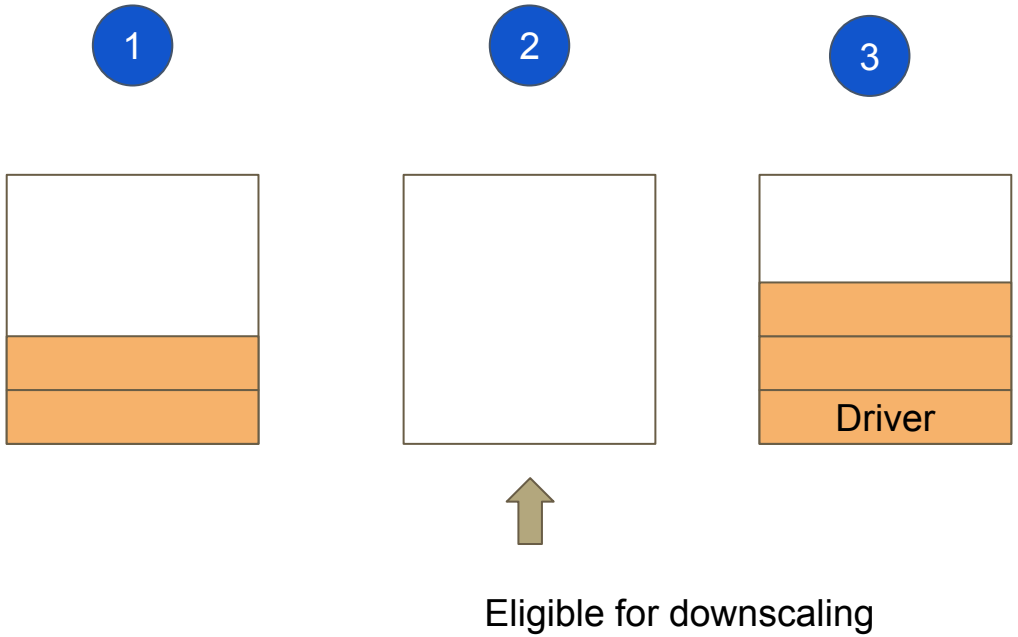
Job 21

## Cost Savings

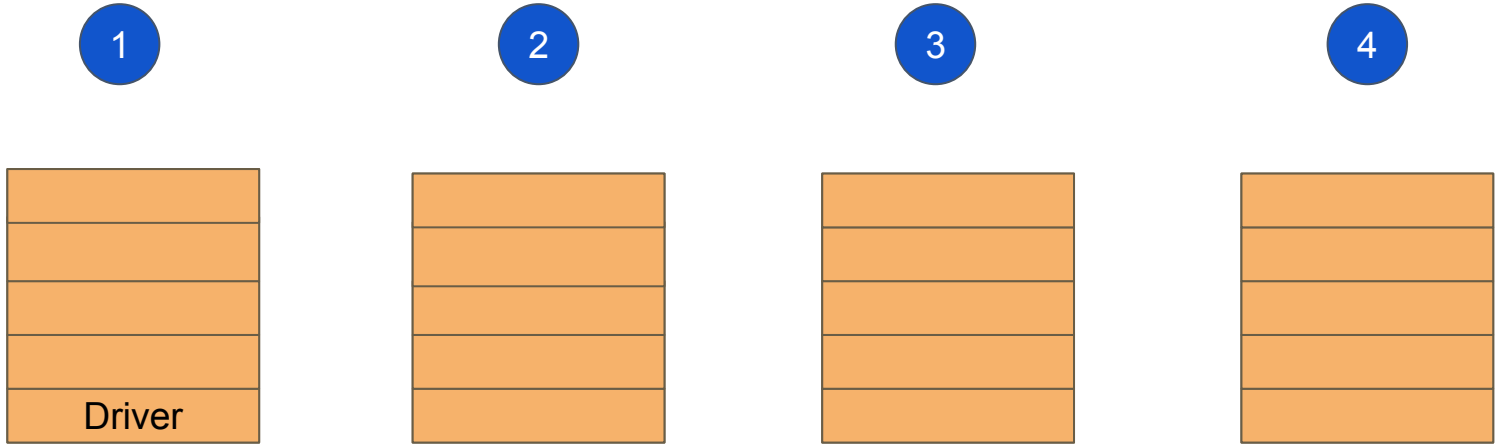



As jobs complete these nodes are moved to **“Medium Usage”** and **“Low Usage”** nodes.

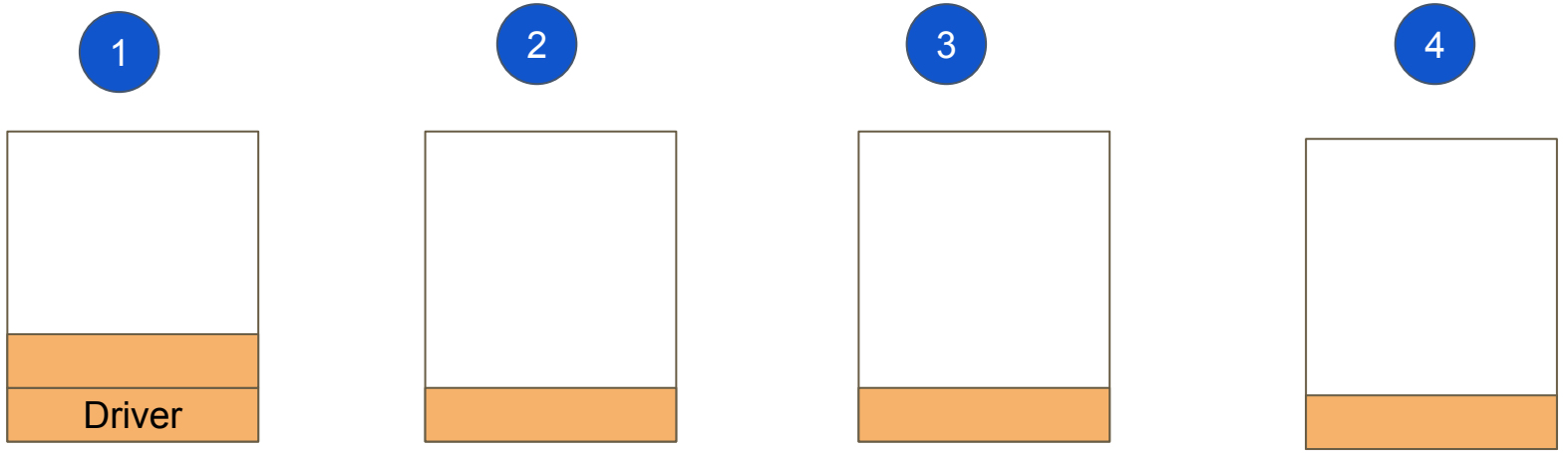
# Example revisited with new allocation strategy



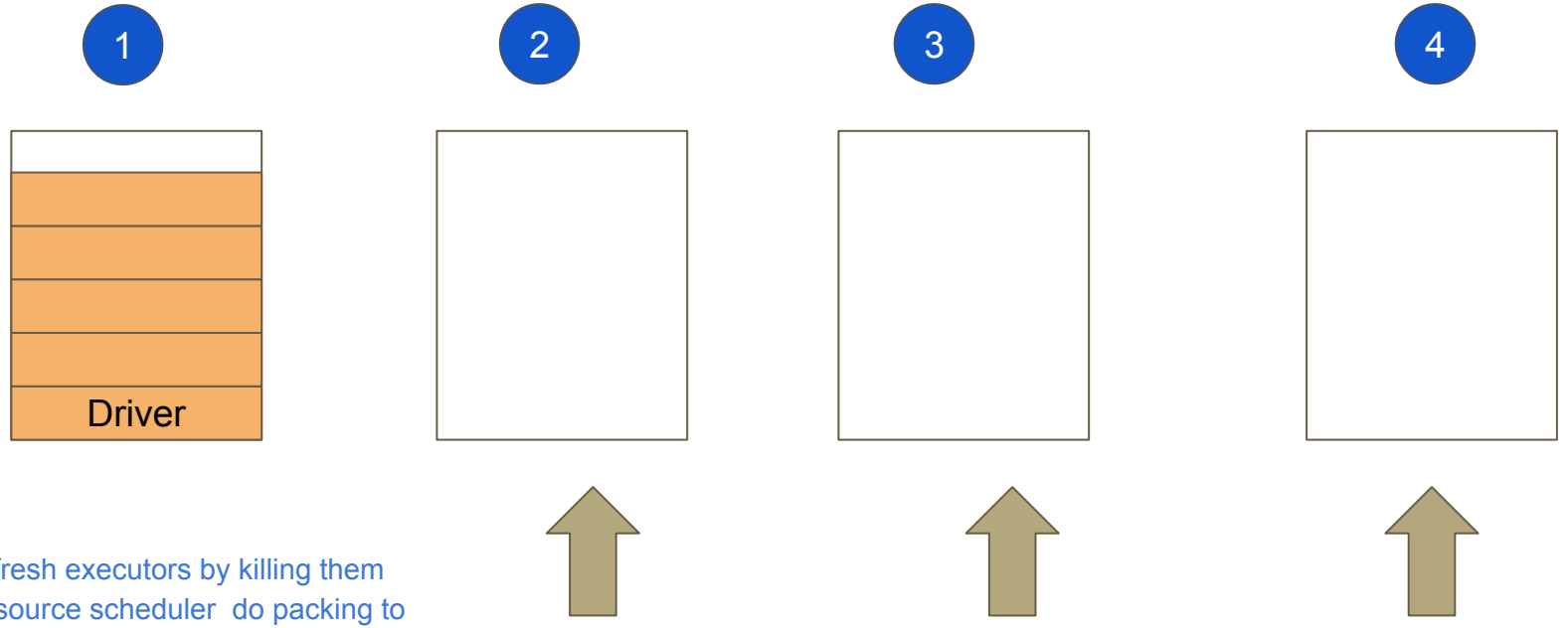
# Downscale issues with min Executors



# Min executors distribution without packing



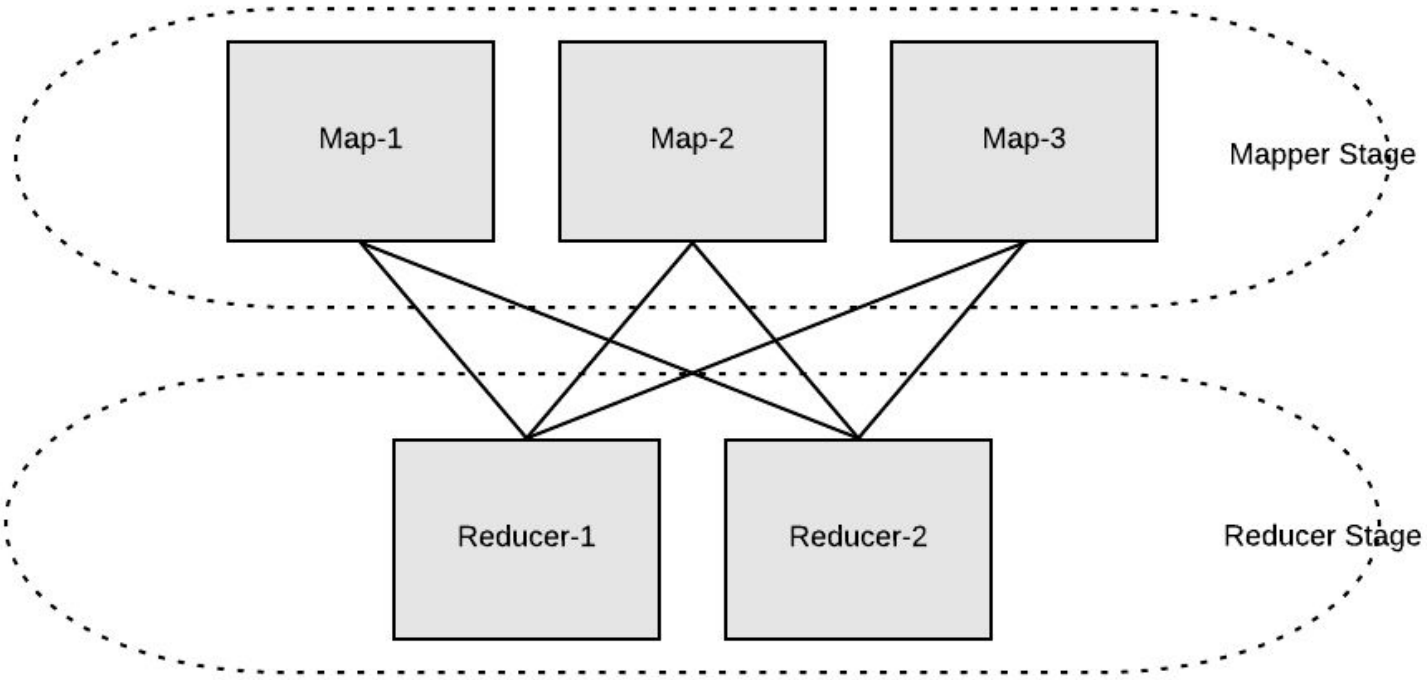
# Min executors distribution with packing




Rotate/refresh executors by killing them and let resource scheduler do packing to defragment the cluster

Nodes eligible for downscaling

# How Shuffle data is produced / consumed?

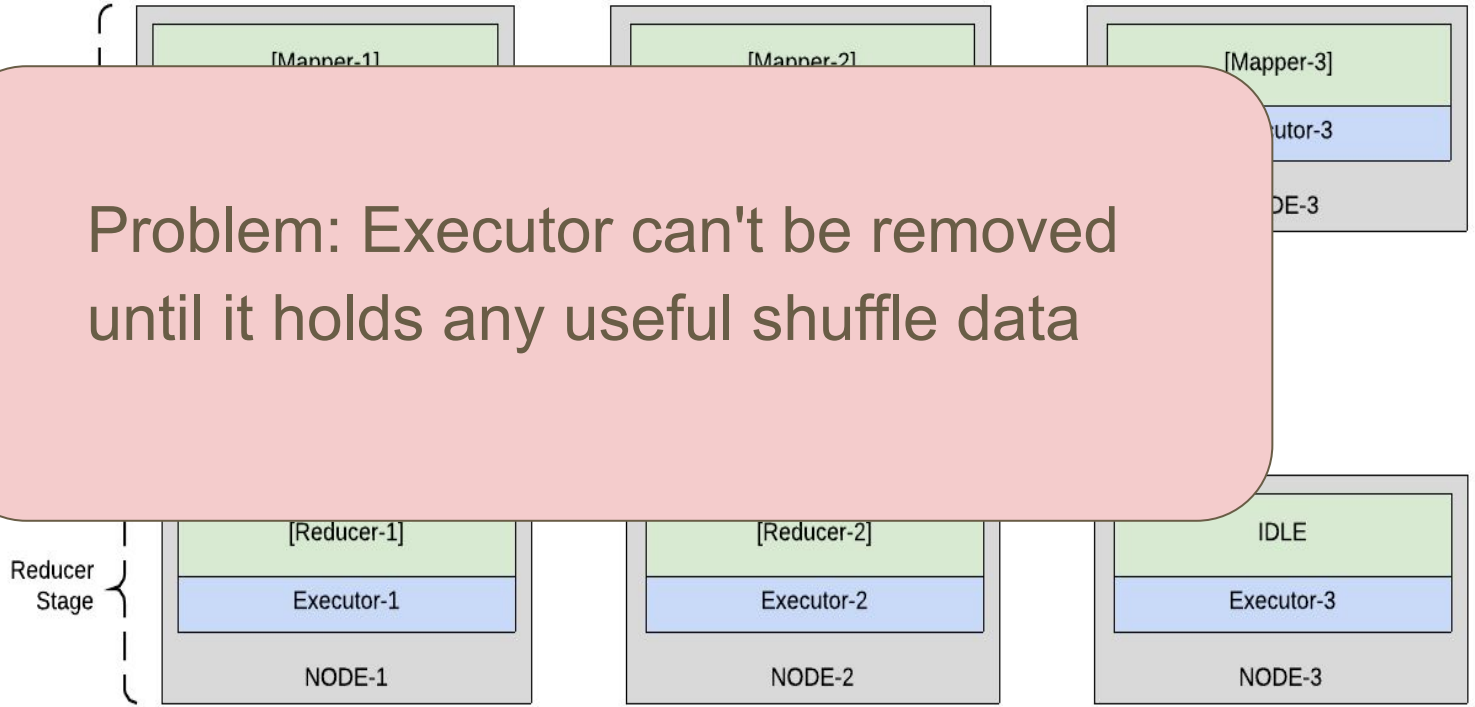


# How Shuffle data is produced / consumed?

**Can't downscale executor 3**

Since reducer stage needs shuffle data generated by all mappers, so corresponding executors needs to be UP.

Problem: Executor can't be removed until it holds any useful shuffle data

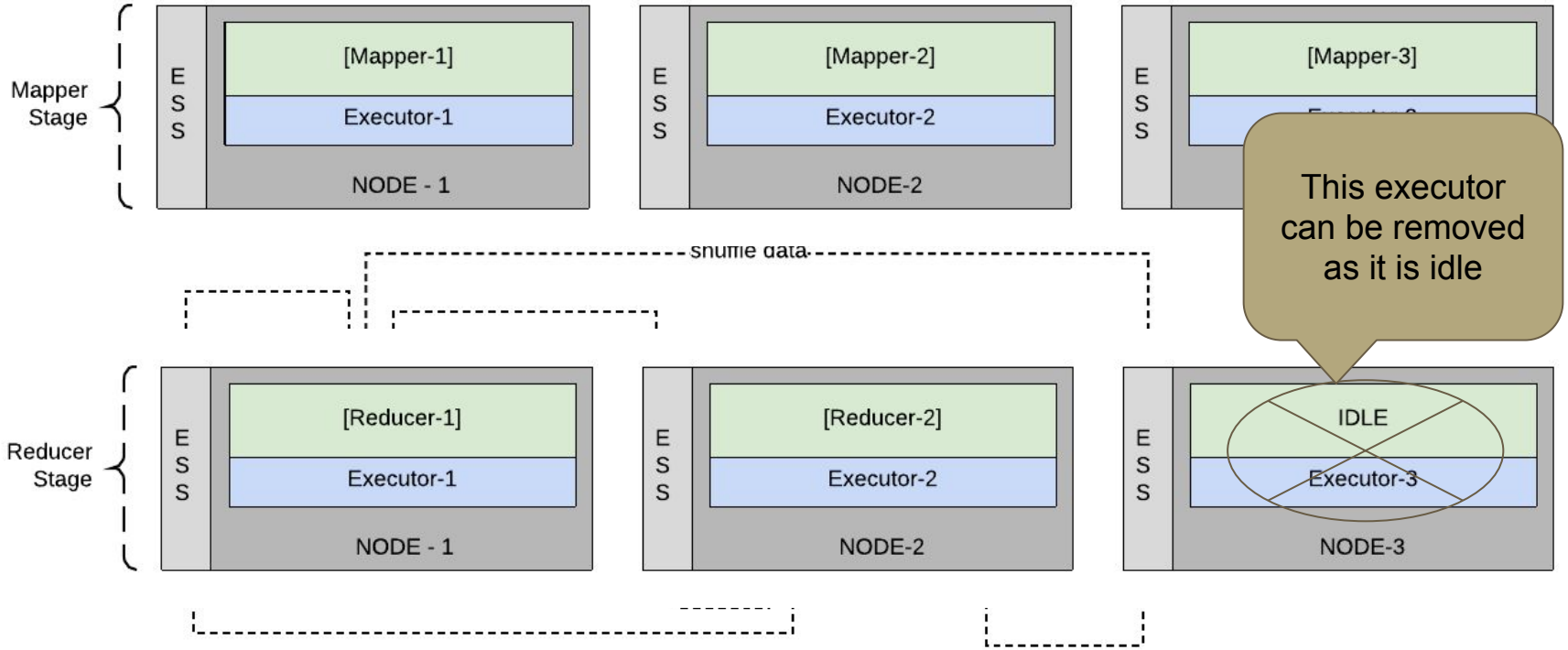


# External Shuffle Service

- Root cause of problem: Executor which generated shuffle data is also responsible for serving it. This ties shuffle data with executor

- Solution: Offload the responsibility of serving shuffle data to external service

# External Shuffle Service

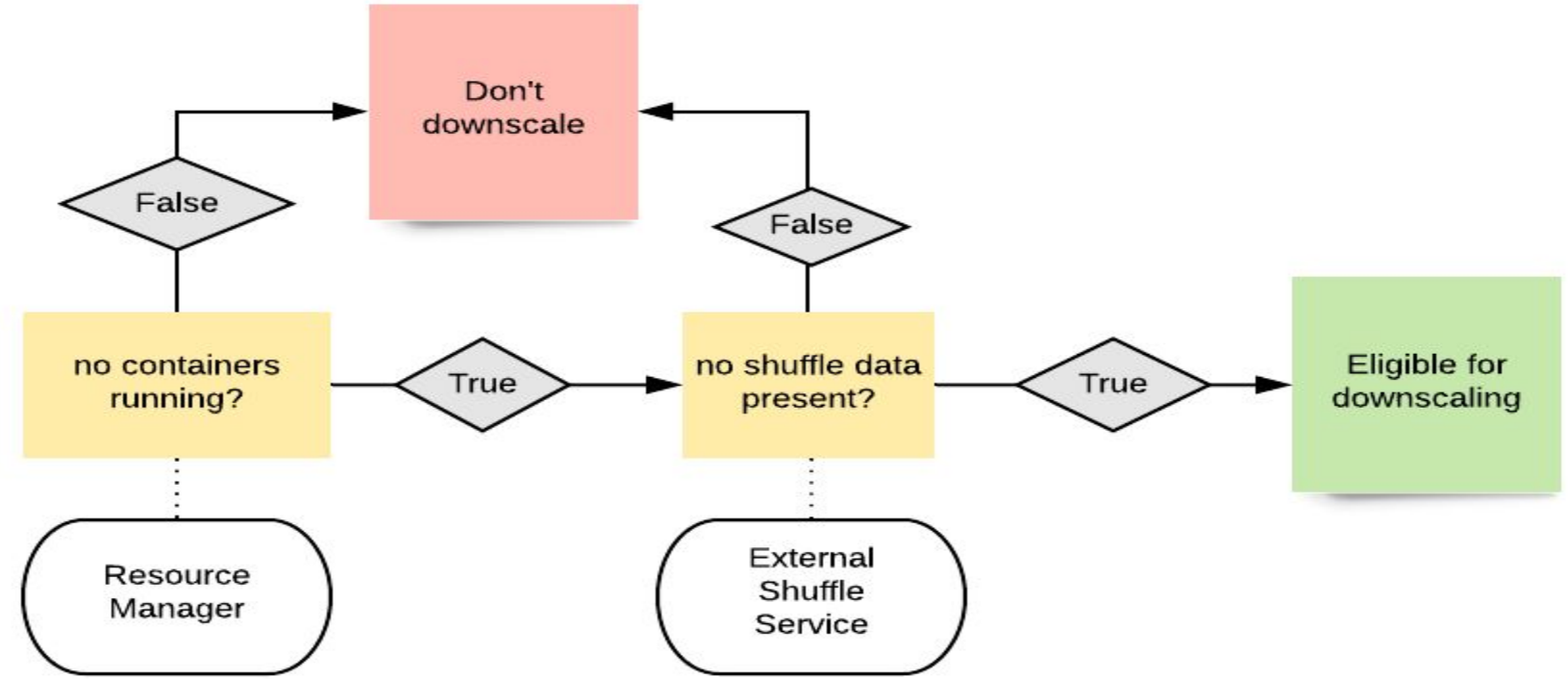


# External Shuffle Service

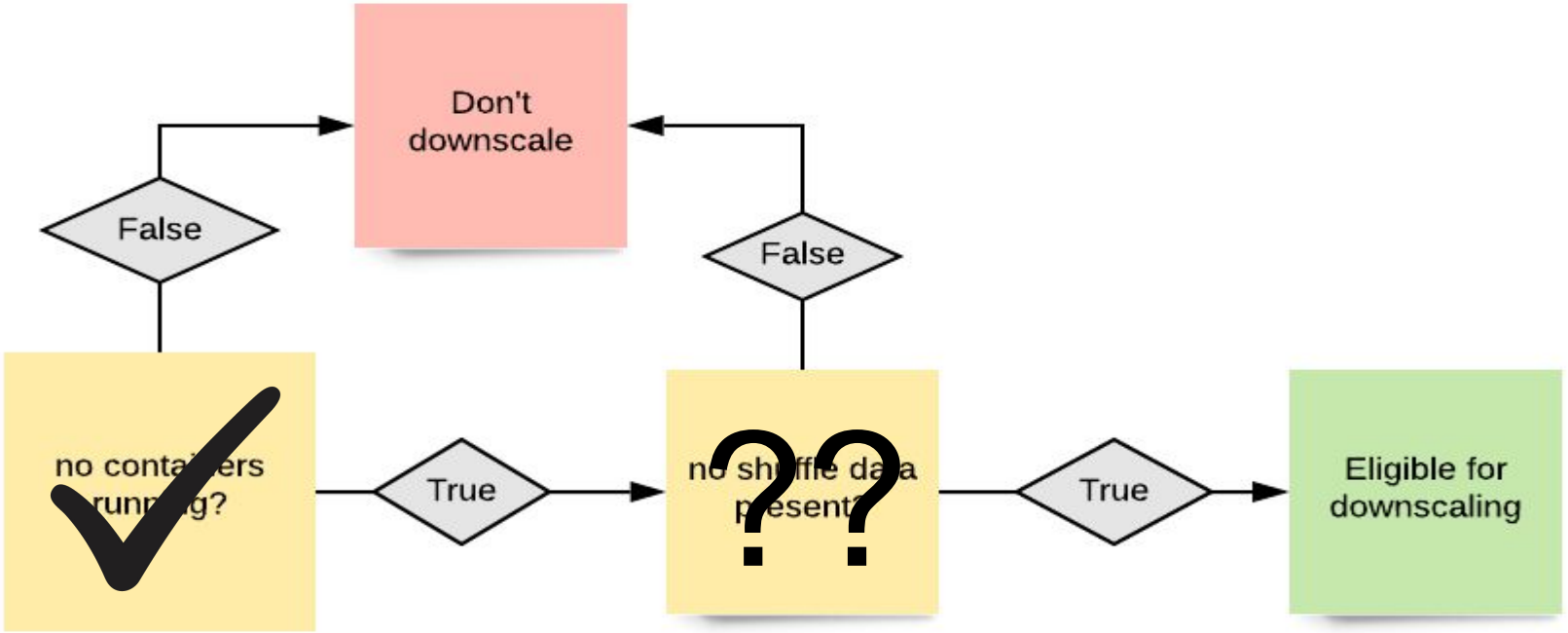
- One ESS per node
  - Responsible for serving shuffle data generated by any executor on that node
  - Once the executor is idle, it can be taken away

- At Qubole:
  - Once the node doesn't have any containers and ESS reports no shuffle data => node is downscaled

# ESS at Qubole



# Recap

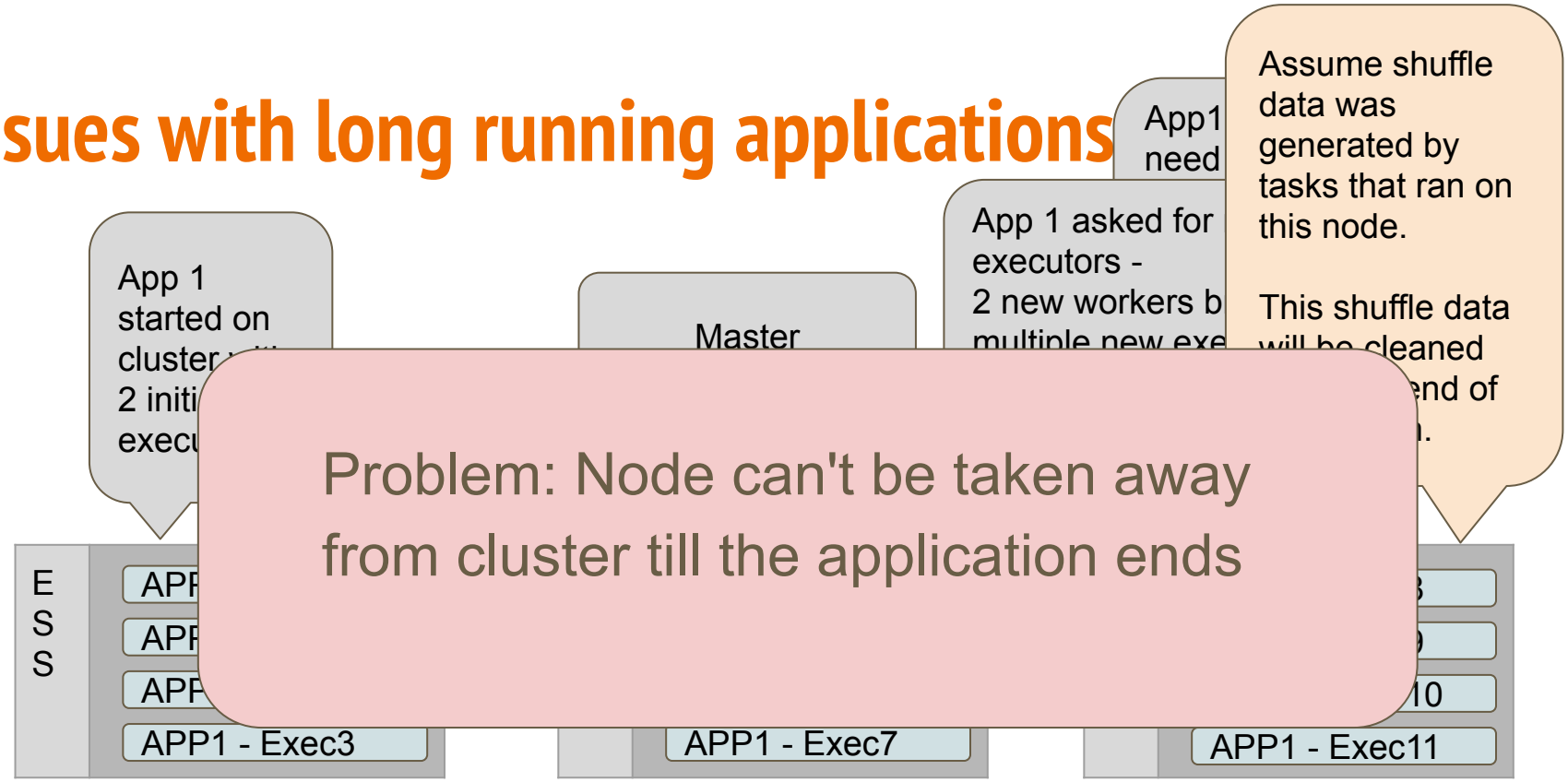

Don't downscale
False
False
no containers running?
True
no shuffle data present?
??
True
Eligible for downscaling

# Shuffle Cleanup

- Shuffle data is deleted at the end of application by ESS
  - In long running Spark applications (ex. interactive notebooks), it keeps on accumulating
  - Results in poor node downscaling

- Can it be deleted before end of application?
  - What shuffle files are useful at a point of time?

# Issues with long running applications



# Shuffle reuse in Spark

```
val df = spark.sql("select * from customer join customer_address on customer.c_current_addr_sk = customer_address.ca_address_sk")
```

```
df.collect()
```

▼ Spark Jobs (1)

▼ Job 1 Success

Stage 1 4000/4000 [Completed]

Stage 2 1000/1000 [Completed]

Stage 3 200/200 [Completed]

```
df.collect()
```

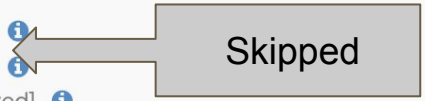
▼ Spark Jobs (1)

▼ Job 2 Success

Stage 4 0/4000 [Skipped]

Stage 5 0/1000 [Skipped]

Stage 6 200/200 [Completed]



# Shuffle Cleanup

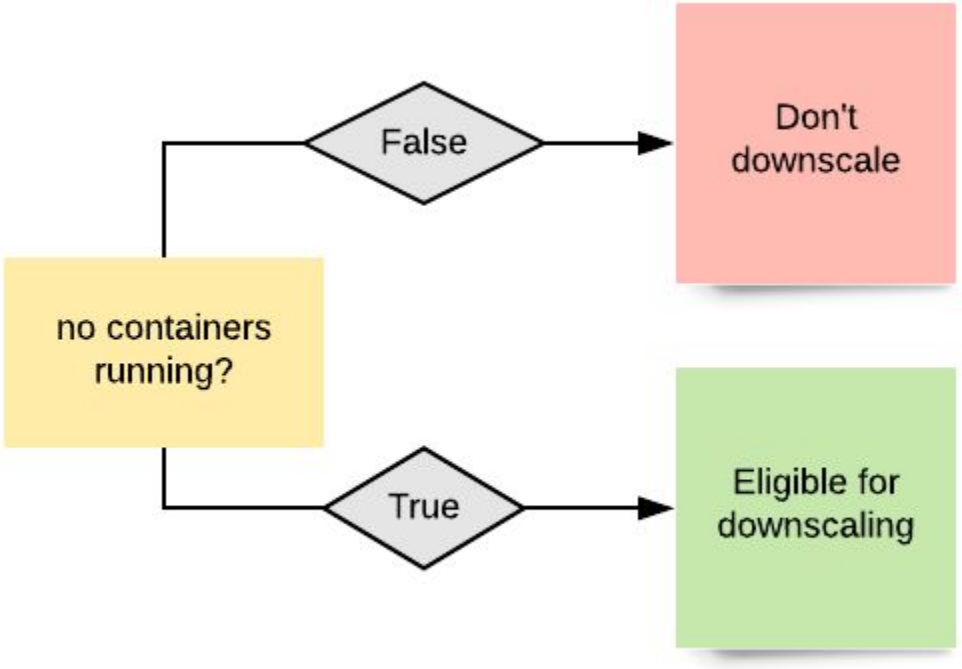
- If a DataFrame which generated the shuffle data goes out of scope in the underlying scala application, then there is no way that shuffle data can be accessed/reused
  - Delete shuffle files when that dataframe goes out of scope
- Helps us in downscaling by making sure that unnecessary shuffle data is deleted
  - Saw 30-40% downscaling improvements
- Related OS Jira: SPARK-4287

# Disaggregation of Compute and Storage

- To utilize full elasticity of the cloud, We have to disaggregate the compute (executors running) and the storage (shuffle data stored)

- Move shuffle data somewhere else?
  - Requirement: Highly available shared storage service
  - Use "*Amazon FSx for Lustre*" or similar services on other clouds

# Downscaling a Node



# Spark - Disaggregation of Compute and Storage

- Mount some NFS endpoint on all the nodes of cluster

- Change shuffle manager in Spark to something which can read/write shuffle from NFS mountpoint
  - Splash (Opensource Apache 2.0 project) provides shuffle manager implementation for shared filesystem
  - Spark can be configured to use Splash using config *spark.shuffle.manager*
  - All mappers will write shuffle data to NFS and all reducers will read shuffle data from splash

- *SPARK-25299 [Use remote storage for persisting shuffle data]* in progress.

# Summary and Future Work

- Different ways to improve downscaling
  - Executor packing strategy and periodic executor refresh
  - Use External Shuffle Service
  - Faster Shuffle cleanup
  - Disaggregate compute and storage

- Future Work: Offload shuffle data only when needed
  - By default use local disk to read/write shuffle data
  - When node is not used for compute, shift shuffle data to NFS
  - Better downscaling without comprising much on performance

# Thank You!