Spark RDD partitions, tasks, and supporting infrastructure

Infrastructure support for RDD

A discussion about how tasks execute within Spark,

starting from reading the file to execution,

with consideration of the Spark / YARN (executor) configuration

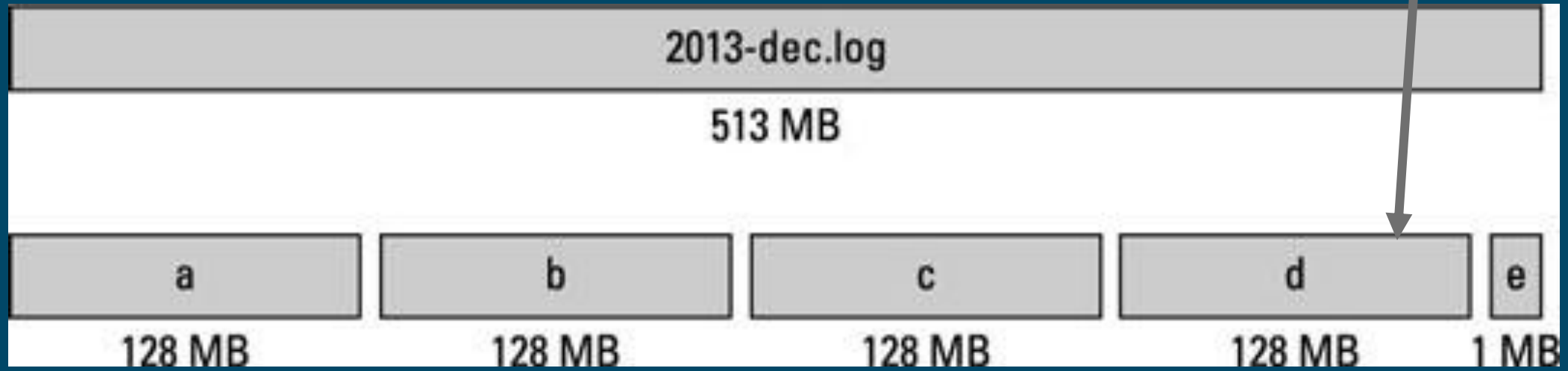
Overview

- From HDFS file, each block becomes a partition
- Each partition becomes a task in Spark
- Each task runs as a thread in a (JVM) container
- Containers are configured in YARN (memory and vCores)
  - Hardware memory and cores limits max container configuration
  - More containers allows for more parallel task execution
  - Max parallelism with max number of containers having little memory and few cores
  - Min parallelism with min number of containers having max memory and max cores

Partitions run in a container that executes the task on the partition (data)

From HDFS file, each block becomes a partition

Given a file of **data**, divide it into a **sequence of blocks**, and **store the blocks**



Partitions from reading a file in Databricks

Community Optimized 15.3 GB Memory, 2 Cores, 1 DBU

Native Linux file

```
1 path = "/databricks-datasets/README.md"
2 lines = sc.textFile(path) # use the sc context to read in a text file
3 print "file is from Linux file system"
4 print "parallelism = {} should be max(2, Number of HDFS blocks).".format(lines.getNumPartitions())
```

```
file is from Linux file system
parallelism = 2 should be max(2, Number of HDFS blocks).
```

```
# Set parallelism used in shuffle
sqlContext.setConf("spark.sql.shuffle.partitions", "200")
```

Partitions from reading a file in Databricks

Community Optimized 15.3 GB Memory, 2 Cores, 1 DBU

Cmd 2

S3 file

```
1 rdd = sc.textFile("dbfs:/databricks-datasets/amazon/README.md")
2 rdd.getNumPartitions()
```

```
Out[1]: 2
```

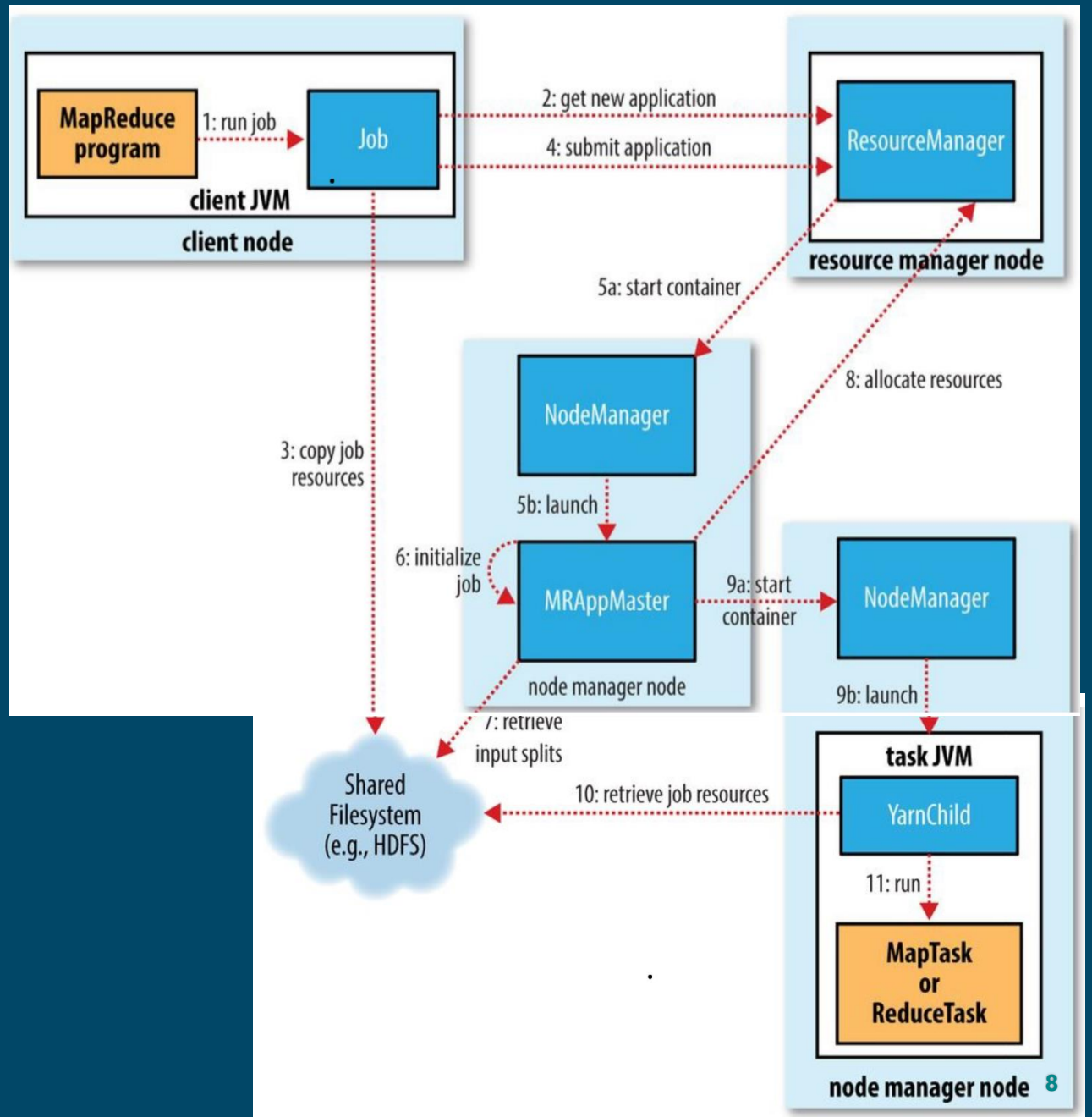
```
1 data = range(1,1000)
2 rdd = sc.parallelize(data)
3 print "parallelism = {} should be `spark.default.parallelism`, which defaults to the number of all cores on all
  machines".format(rdd.getNumPartitions())
4 data = range(1,2)
5 rdd = sc.parallelize(data)
6 print "parallelism = {} should be `spark.default.parallelism`, which defaults to the number of all cores on all
  machines".format(rdd.getNumPartitions())
7 data = range(1,1000)
8 rdd = sc.parallelize(data,20)
9 print "parallelism = {} should be the parameter value to parallelize".format(rdd.getNumPartitions())
```

```
parallelism = 8 should be `spark.default.parallelism`, which defaults to the number of all cores on all machines
parallelism = 8 should be `spark.default.parallelism`, which defaults to the number of all cores on all machines
parallelism = 20 should be the parameter value to parallelize
```

Architecture of jobs & their tasks

- Each partition becomes a task in Spark

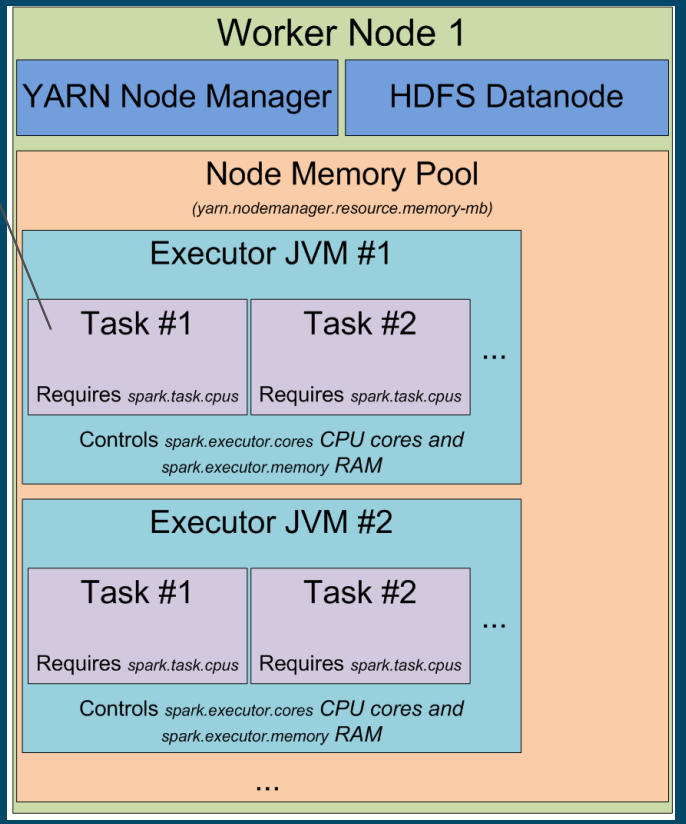
- The application master decides how to run the tasks of a given job



Job tasks execute within a JVM (container)

- The application master decides how to run the tasks of a given job
  - Each task runs as a thread in a (JVM) container
  - If the job is small, the application master may choose to run all tasks in the same JVM
  - Container here is a YARN or Kubernetes container

>1 task / executor = uberized
Otherwise, 1 task per executor



Spark manages tasks better than MapReduce

- MapReduce
  - Each task runs inside a JVM, sequentially
  - A new job causes a new JVM to be created for the tasks that run within
    - Slow to create new JVM
- Spark
  - Each task runs as a thread inside a JVM
  - New tasks are created by forking new threads in an existing JVM
    - Forking within a JVM is faster than a new JVM

Containers are configured in YARN
(memory and vCores)

Spark job config specifies executor (aka container) size

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

```
# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000
```

Spark2 (from Notebook) config specifies physical executor size (which runs in container)

Zeppelin

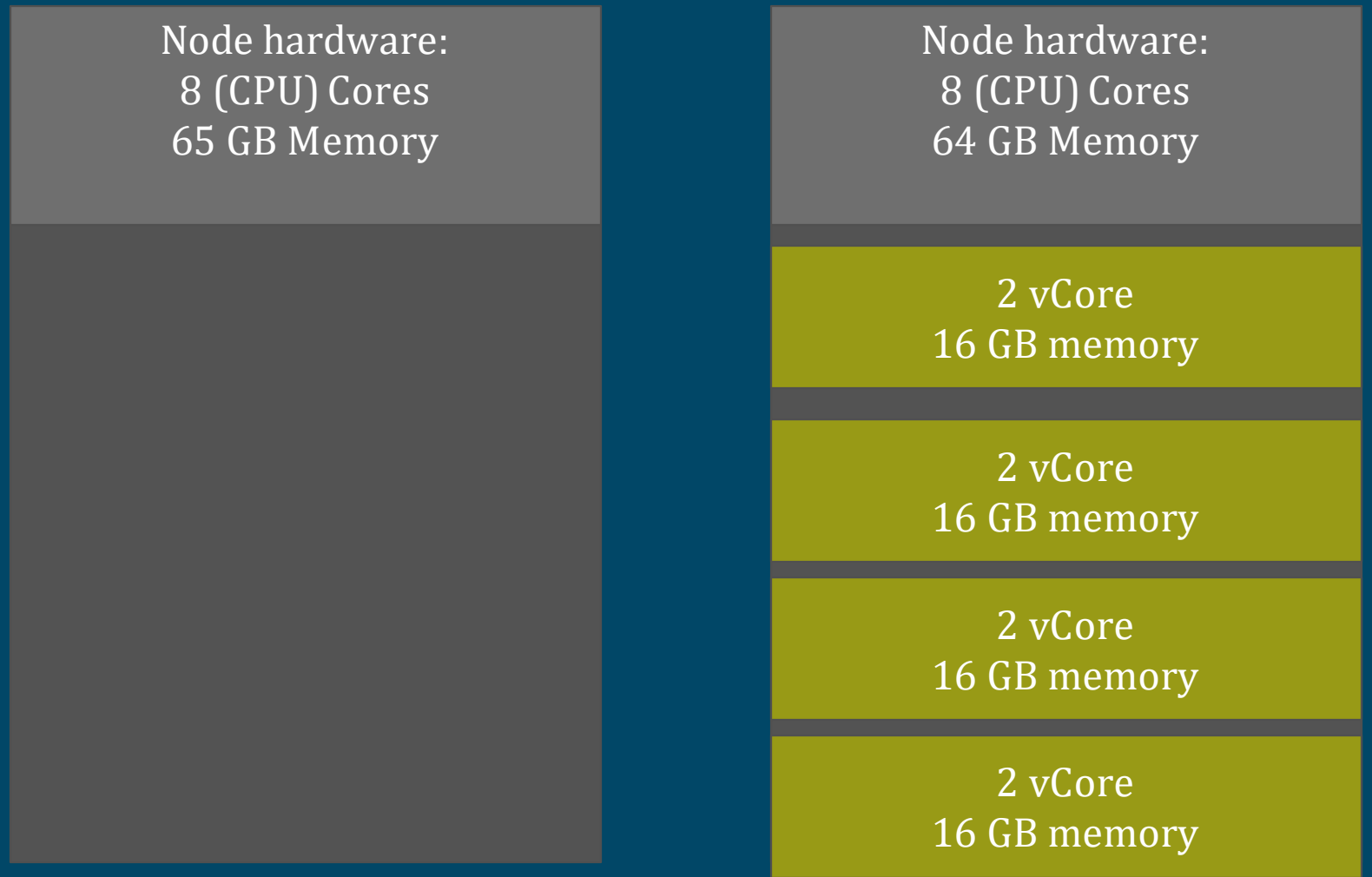
args	
master	yarn-client
spark.app.name	Zeppelin
spark.cores.max	
spark.executor.memory	9g

Defaults to max allowed

Container size specified, here for notebook jobs

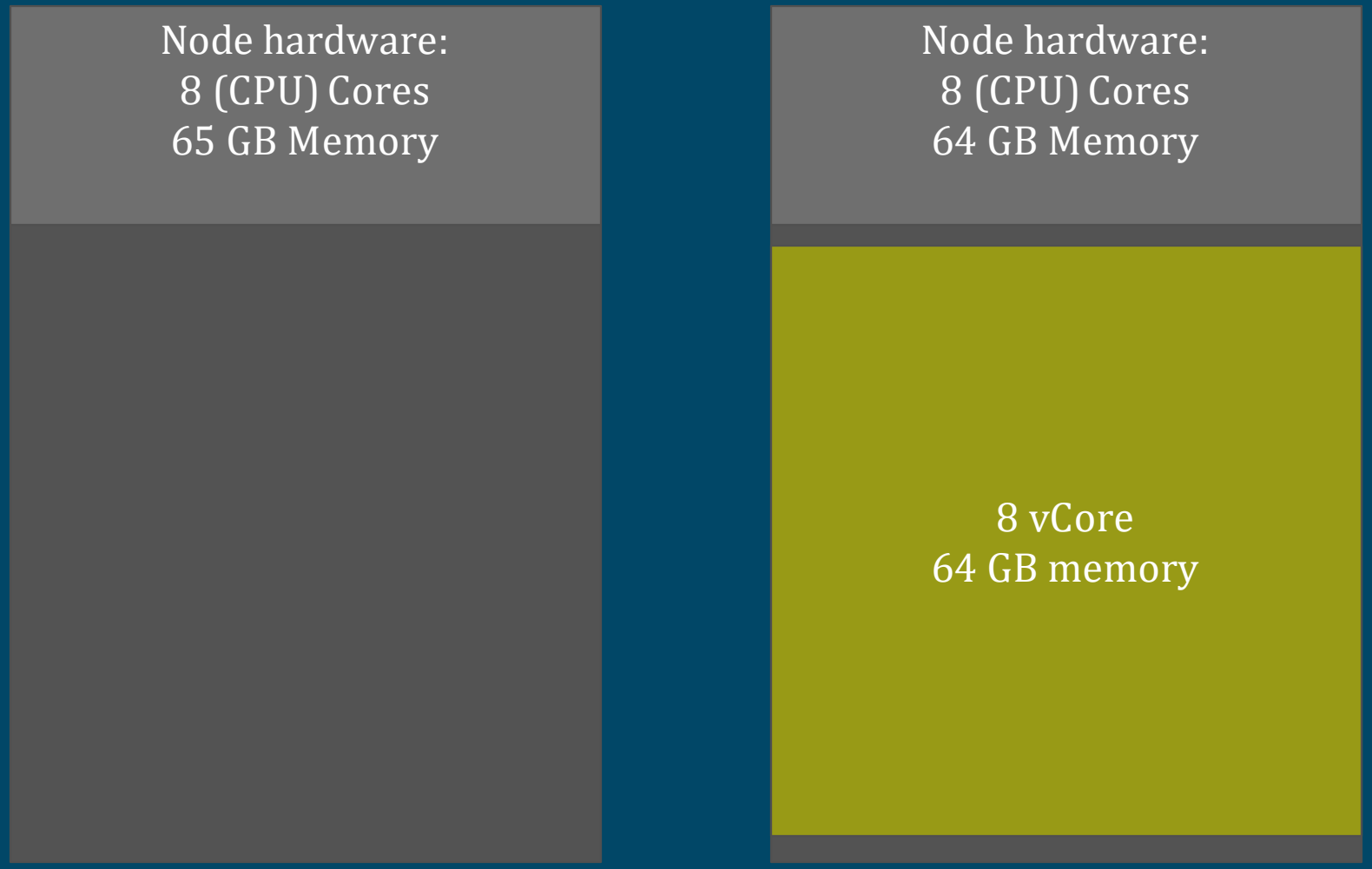
Containers allocate physical data node memory & cores

e.g., 4 containers consume all node resources



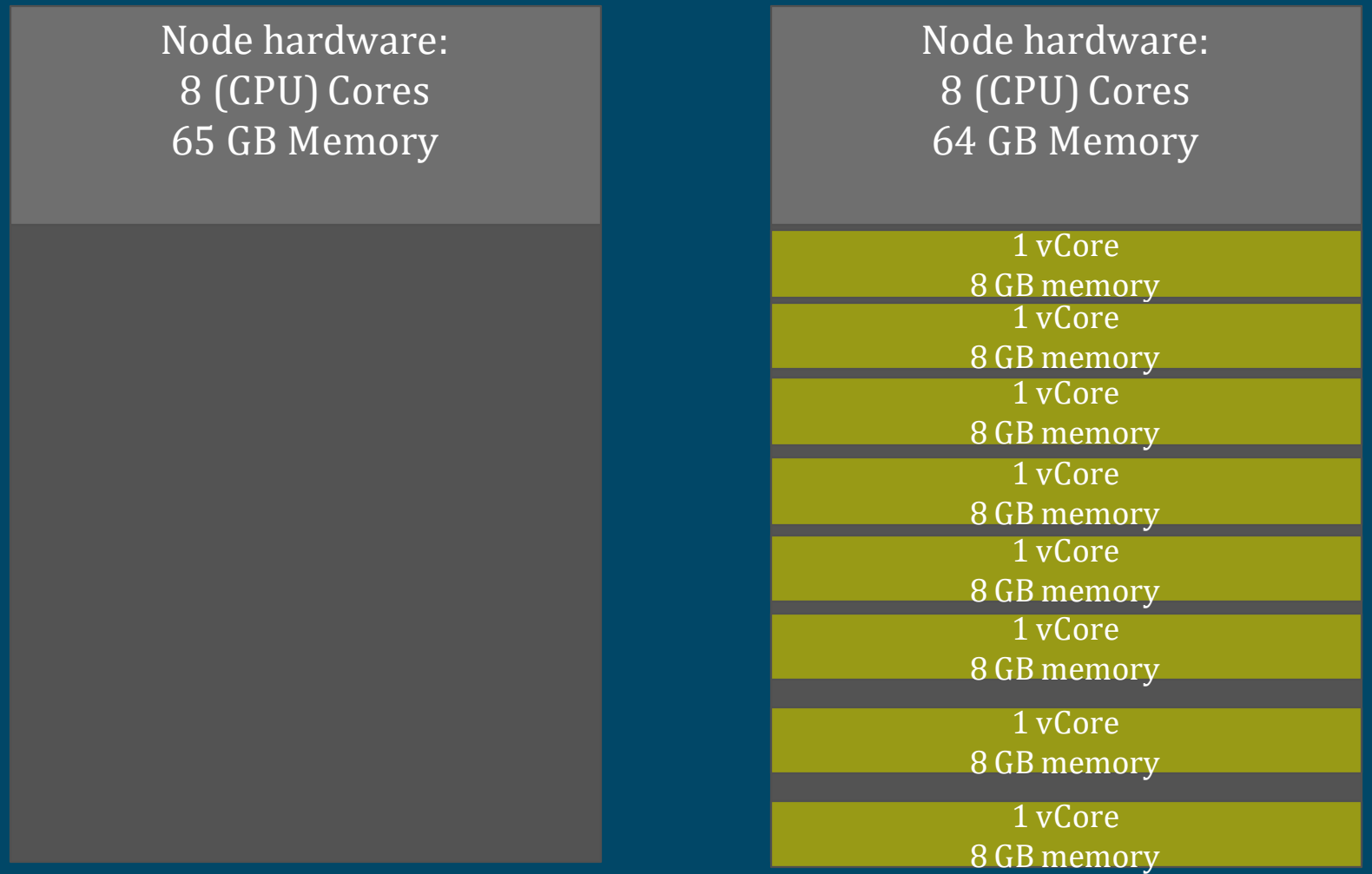
Containers allocate node memory & cores

e.g., 1 containers consumes all node resources



Containers allocate node memory & cores

e.g., 8 containers consume all node resources



Containers allocate node memory & cores

e.g., 8 containers consume all node resources

Node hardware:
8 (CPU) Cores
65 GB Memory

Node hardware:
8 (CPU) Cores
64 GB Memory

1 vCore
8 GB memory

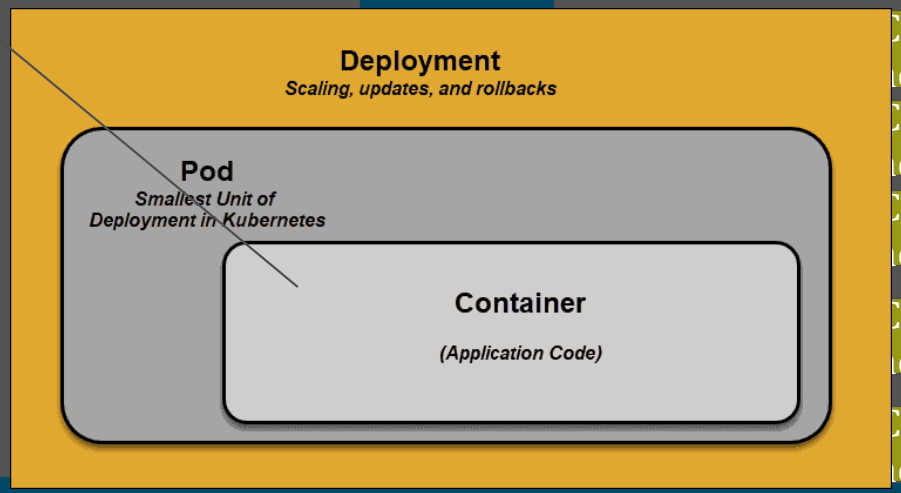
1 vCore
8 GB memory

1 vCore
8 GB memory

Spark submit communicates with the cluster manager

YARN manages its containers (running Spark task)

Kubernetes manages its containers (running Spark tasks)



Resource problem example

- If your GCP GKE deployment is Pending for a long time, you may have an issue with insufficient resources
  - To view pods:
    - kubectl get pods
  - To view a pods description:
    - kubectl describe pod POD_NAME_FROM_GET_PODS_ABOVE
  - If the above indicates insufficient resources in log, then delete old pods:
    - kubectl delete pods -l app=airbnb
  - Typically, delete pods after each run
    - Otherwise, your node memory may be filled with unused containers

Container size affects parallelism

- Smaller containers result in more total containers
  - More containers generally means more tasks executing in parallel
    - Container (JVM) parallelism
- Larger containers result in fewer total containers
  - Fewer containers may mean fewer tasks executing in parallel
  - When a container is given multiple tasks, then (Spark) tasks run as threads within container
    - Note: a container has a limited number of thread (e.g., default 5)
    - Depends on thread parallelism

Containers affect shuffle & sort

- Partition is executed by a task, which runs in a container
- If tasks/containers on only one node, then the data transfer of shuffle & sort (wide transformation) is within a node
- Otherwise, data travels across nodes (which is slow)
- That's why wide transformation (reduceByKey, groupByKey, repartition) are slower

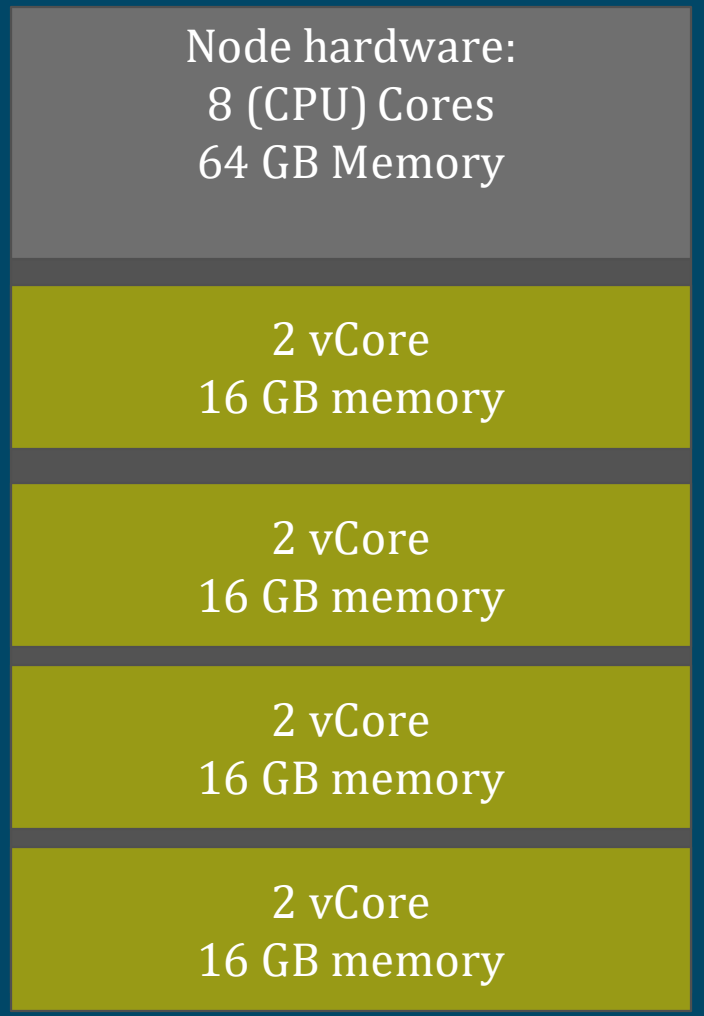
Tasks are run in JVM containers,

so...

how big should a Spark JVM container be?

Container size goals

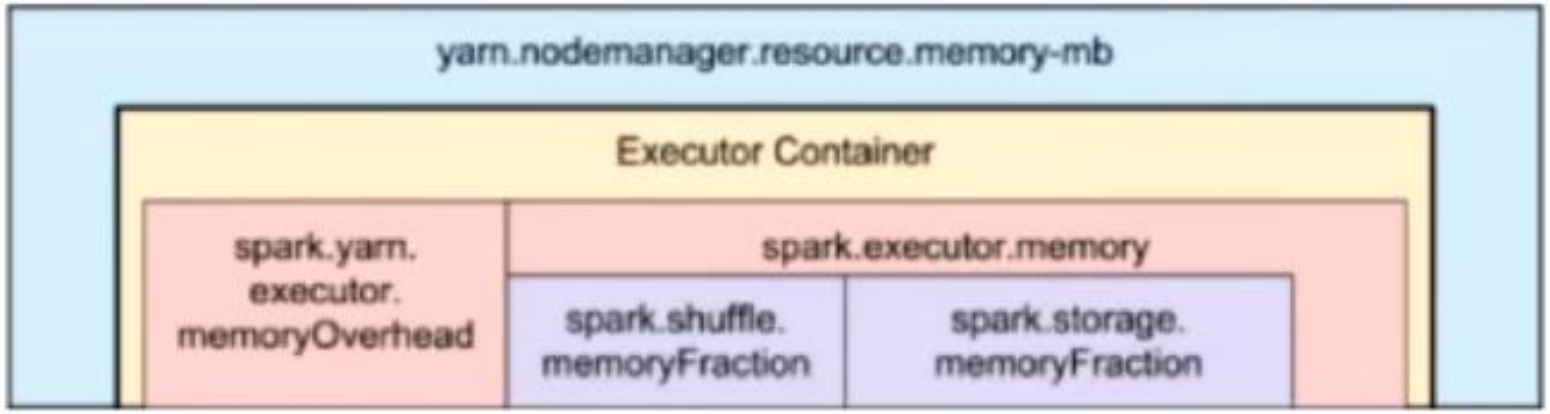
- Uses all of cluster memory (& CPUs), leaving no remainder
  - Executor size + overhead = container size
- Maximum partition fits within executor memory
- Generally, prefer larger executor with thread parallelism because container parallelism has more overhead than thread



Container does the work

- Imagine 4 computers each with
  - 25g of memory, for a total of 100g
  - 10 CPUs, for a total of 40 CPUs
- Let's say our standard container is given 1G of memory and 1 CPU
  - How many containers per computer?
    - Only 10, because the CPU is the limitation
- In configuring YARN (the resource manager), you set the min/max of both
  - memory and
  - virtual CPUs

Container has memory overhead (~7%) in addition to the executor's memory



- --executor-memory controls the heap size
- Need some overhead (controlled by spark.yarn.executor.memory.overhead) for off heap memory
  - Default is max(384MB, .07 * spark.executor.memory)

Exam question

- An analysis of data partitions reveals that the largest partition is about 4G.
  Currently, Spark executor container is configured to be 32G of the 64G memory in each node
  - How can possessing of the data be improved?
  - Explain why your change reduces processing time

Exam answer

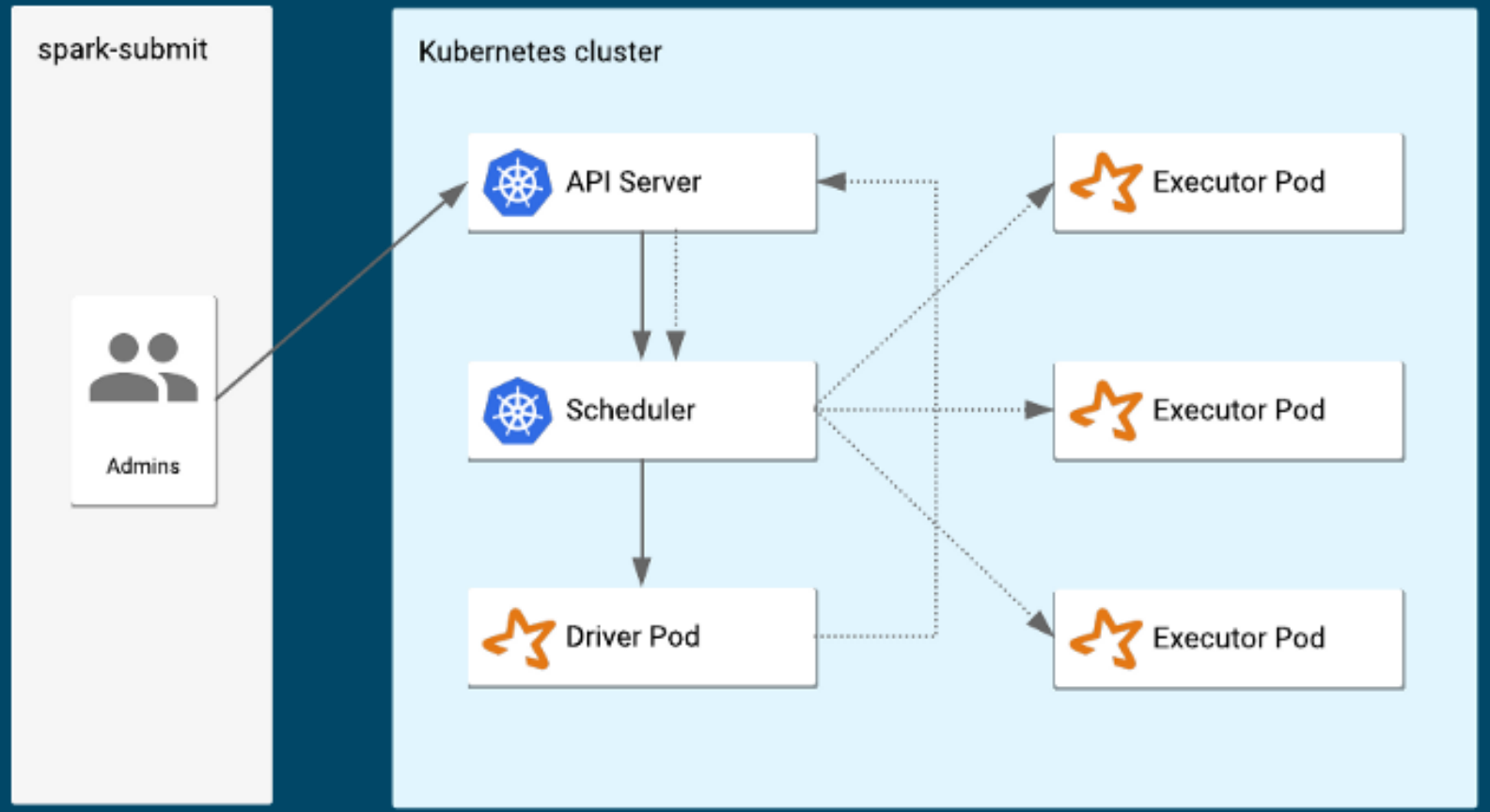
- An analysis of data partitions reveals that the largest partition is about 4G. Currently, Spark executor container is configured to be 32G of the 64G memory in each node
  - How can possessing of the data be improved?
    - Reduce the size of the container from 32g to 8g
  - Explain why your change reduces processing time
    - 8g will provide sufficient memory for the 4g of data and the additional (~7%) overhead required by the container (YARN processes)
    - 8g container will increase the total containers from 2 (@ 32g) to 8 (@ 8g)
    - Increasing the number of containers, increases the parallelism (generally)

Example of spark_submit (for GKE)

spark_submit

- Spark contains a cluster manager than runs spark jobs over Kubernetes clusters
- spark_submit provides the means to run a spark program from a Docker image over a Kubernetes cluster

spark_submit



https://medium.com/@toddrosner/spark-on-kubernetes-b18f16ac95ad

```
#!/bin/bash

# Run on Docker for Windows with data and source in Docker image

# Spark on Kubernetes doesn't support submitting locally stored files with spark-submit
# This means many submit parameters will not work unless the path is http, including py-files, files, archives
# * https://stackoverflow.com/questions/61711673/how-to-submit-pyspark-job-on-kubernetes-minikube-using-spark-submit
# Rely on run_env.py to load and run modules, which must be built into the Docker image (or mounted)

# Local spark to run spark-submit
# TODO: Check that this is the path to your spark.
# TODO: If running in WSL2 Linux, then ensure that this is the path to spark install in WSL2 Linux
SPARK_HOME=/opt/spark

# URL of Kubernetes master on Docker desktop
MASTER=k8s://https://localhost:6443

# Module here is source zip (not egg file). Must create a zip file from the source
PYSPARK_APP_MODULE=airbnb

# Image execution, values may be overridden with Kubernetes: spark.kubernetes.driver.limit.cores, spark.kubernetes.dr
EXECUTORS=1
EXECUTOR_MEMORY=2g
DRIVER_MEMORY=1g

# Image which includes run_env.py in working directory
# TODO: Ensure that this is the name of the image you built
IMAGE=wrobinson/airbnb:1.0

# working directory on image (default set by spark image creation, docker-image-tool.sh)
# https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b
WORKING_DIR=/opt/spark/work-dir

SCRIPT=local://${WORKING_DIR}/run_env.py  # Run your module
#SCRIPT=local://${WORKING_DIR}/sleep.py   # For debugging (e.g., checking the mount)
# Shell into container; kubectl exec <CONTAINER ID> -ti /bin/bash

SPARK_CMD="$SPARK_HOME/bin/spark-submit \
  --master ${MASTER} \
  --deploy-mode cluster \
  --driver-memory ${DRIVER_MEMORY}
  --executor-memory ${EXECUTOR_MEMORY} \
  --conf spark.executor.instances=${EXECUTORS} \
  --conf spark.dynamicAllocation.enabled=false \
  --conf spark.kubernetes.container.image=${IMAGE} \
  --name ${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.executor.label.app=${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.driver.label.app=${PYSPARK_APP_MODULE} \
  --conf spark.kubernetes.driverEnv.PYSPARK_MAJOR_PYTHON_VERSION=3 \
  --conf spark.kubernetes.driverEnv.PYSPARK_APP_MODULE=${PYSPARK_APP_MODULE} \
  ${SCRIPT} \
  --master=${MASTER} --py_files=${WORKING_DIR}/${PYSPARK_APP_MODULE}.zip"

echo ${SPARK_CMD}
echo
eval ${SPARK_CMD}
```

```
#!/bin/bash

# Run on Docker for Windows with data and source in Docker image

# Spark on Kubernetes doesn't support submitting locally stored files with spark-s
# This means many submit parameters will not work unless the path is http, includi
# * https://stackoverflow.com/questions/61711673/how-to-submit-pyspark-job-on-kubernetes-minikube-using-spark-submit
# Rely on run_env.py to load and run modules, which must be built into the Docker image (or mounted)

# Local spark to run spark-submit
# TODO: Check that this is the path to your spark.
# TODO: If running in WSL2 Linux, then ensure that this is the path to spark install in WSL2 Linux
SPARK_HOME=/opt/spark

# URL of Kubernetes master on Docker desktop
MASTER=k8s://https://localhost:6443

# Module here is source zip (not egg file). Must create a zip file from the source
PYSPARK_APP_MODULE=airbnb
```

Path to local install of Spark

Path to Kubernetes server

Name of our Python module
Place it in environment variable (of the image)
run_env.py Script will read this name to start the module

Parameters for a Kubernetes cluster

```
# Image execution, values may be overridden with Kubernetes: spark.kubernetes.driver.limit.cores, spark.kubernetes.dr
EXECUTORS=1
EXECUTOR_MEMORY=2g
DRIVER_MEMORY=1g

# Image which includes run_env.py in working directory
# TODO: Ensure that this is the name of the image you built
IMAGE=wrobinson/airbnb:1.0

# working directory on image (default set by spark image creation, docker-image-too
# https://levelup.gitconnected.com/spark-on-kubernetes-3d822969f85b
WORKING_DIR=/opt/spark/work-dir

SCRIPT=local://${WORKING_DIR}/run_env.py  # Run your module
#SCRIPT=local://${WORKING_DIR}/sleep.py   # For debugging (e.g., checking the mount)
# Shell into container; kubectl exec <CONTAINER ID> -ti /bin/bash
```

Image to run
(same on all nodes)

Spark working directory (on image)

Script to run to start our program

Spark Submit
spark knows how to start image in Kubernetes

```
38  SPARK_CMD="$SPARK_HOME/bin/spark-submit \
39    --master ${MASTER} \
40    --deploy-mode cluster \
41    --driver-memory ${DRIVER_MEMORY}
42    --executor-memory ${EXECUTOR_MEMORY} \
43    --conf spark.executor.instances=${EXECUTORS} \
44    --conf spark.dynamicAllocation.enabled=false \
45    --conf spark.kubernetes.container.image=${IMAGE} \
46    --name ${PYSPARK_APP_MODULE} \
47    --conf spark.kubernetes.executor.label.app=${PYSPARK_APP_MODULE} \
48    --conf spark.kubernetes.driver.label.app=${PYSPARK_APP_MODULE} \
49    --conf spark.kubernetes.driverEnv.PYSPARK_MAJOR_PYTHON_VERSION=3 \
50    --conf spark.kubernetes.driverEnv.PYSPARK_APP_MODULE=${PYSPARK_APP_MODULE} \
51    ${SCRIPT} \
52    --master=${MASTER} --py_files=${WORKING_DIR}/${PYSPARK_APP_MODULE}.zip"
```

Parameters defined above

```
SCRIPT=local://${WORKING_DIR}/run_env.py
```

Notice we'll start the program with the ${SCRIPT}

py_files points to our program, which is local to the image

Resource problem example

- If your GCP GKE deployment is Pending for a long time, you may have an issue with insufficient resources
  - To view pods:
    - kubectl get pods
  - To view a pods description:
    - kubectl describe pod POD_NAME_FROM_GET_PODS_ABOVE
  - If the above indicates insufficient resources in log, then delete old pods:
    - kubectl delete pods -l app=airbnb
  - Typically, delete pods after each run
    - Otherwise, your node memory may be filled with unused containers

Hortonworks/Cloudera YARN configuration example

Spark2 (from Notebook) config specifies physical executor size (which determines container size)

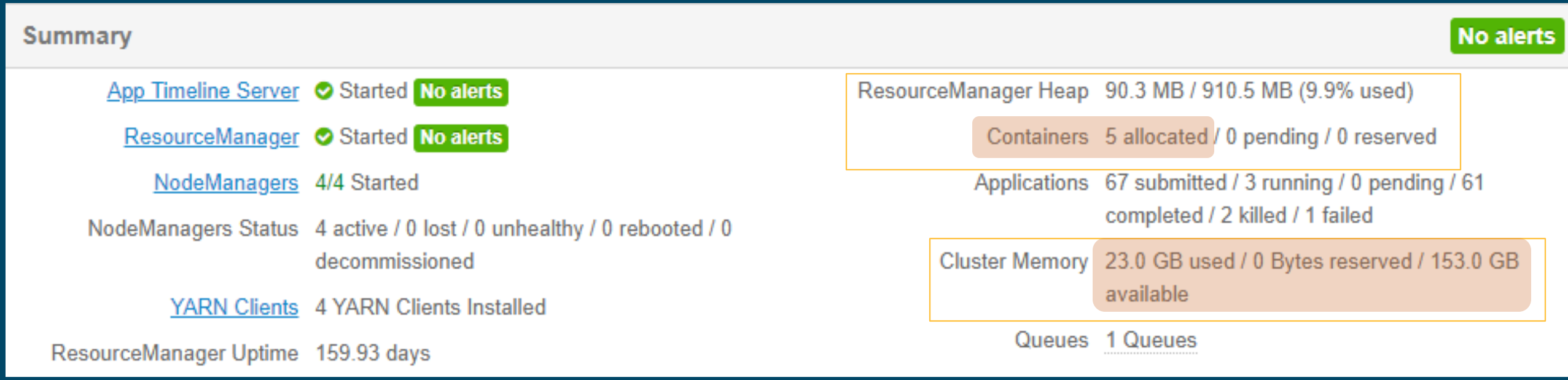
Zeppelin

args	
master	yarn-client
spark.app.name	Zeppelin
spark.cores.max	
spark.executor.memory	9g

Defaults to max allowed

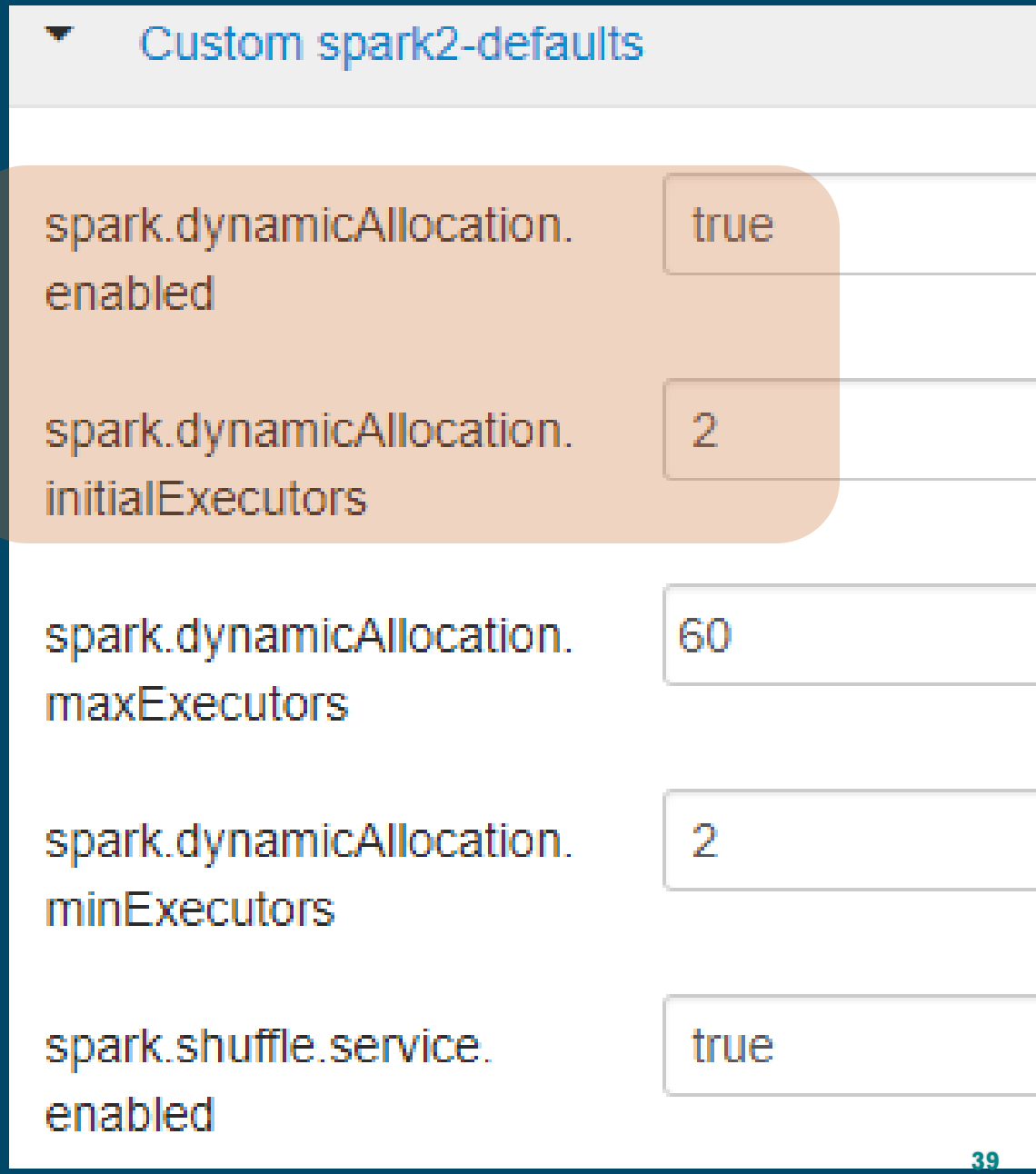
Container size specified, here for notebook jobs

You would like to maximize the number of tasks and use all of memory



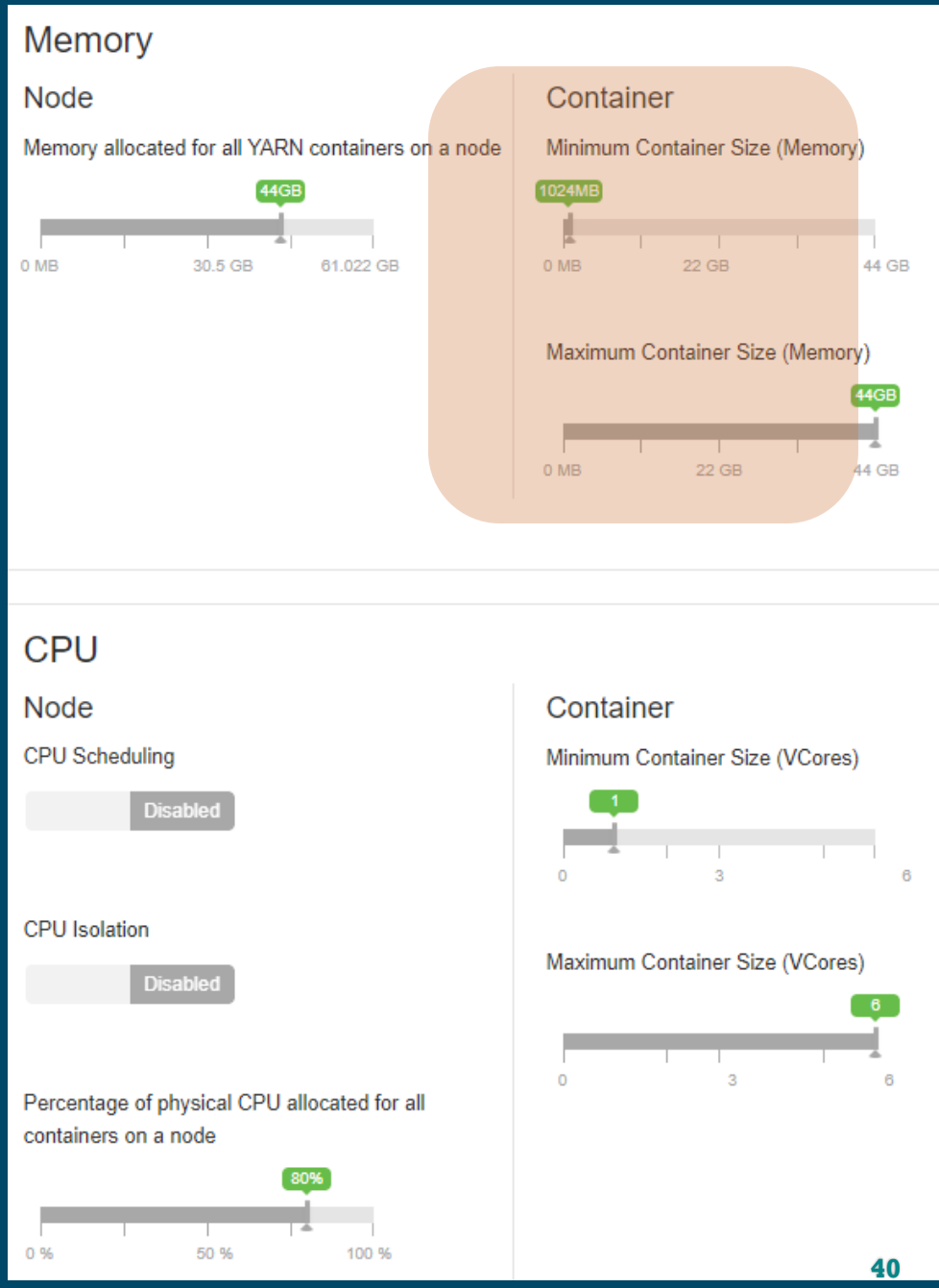
Spark2 dynamic configuration

- Newer Spark configuration will dynamically create containers as needed
  - Works in Kubernetes & YARN
- Allow dynamic allocation of
  - Tasks (executors)
- Initial 2 can become 60 executors (containers)



YARN Configuration

- Allow dynamic management of
  - container memory



Important to remember

- RDD tasks
  - Each file block becomes a partition
  - Each partition becomes a task in Spark
  - Each task runs as a thread in a (JVM) container
  - Spark code processes partitions
    - 10 partitions → 10 tasks (1 to 1)
- An executor (container) processes tasks, in threads
  - 100 partitions generates 100 tasks
  - When processed by 10 nodes, each having only room for 2 executors (as containers), then there are 20 executors
  - 20 executors can process 20 tasks in parallel (ignoring threads)
    - Each will get (ideally) 5 tasks; 100 / 20 = 5 per node
  - If 2 threads per executor and 20 executors, then 40 tasks in parallel
- Container = JVM running Spark tasks, managed by cluster manager