

# Hadoop Execution Environment

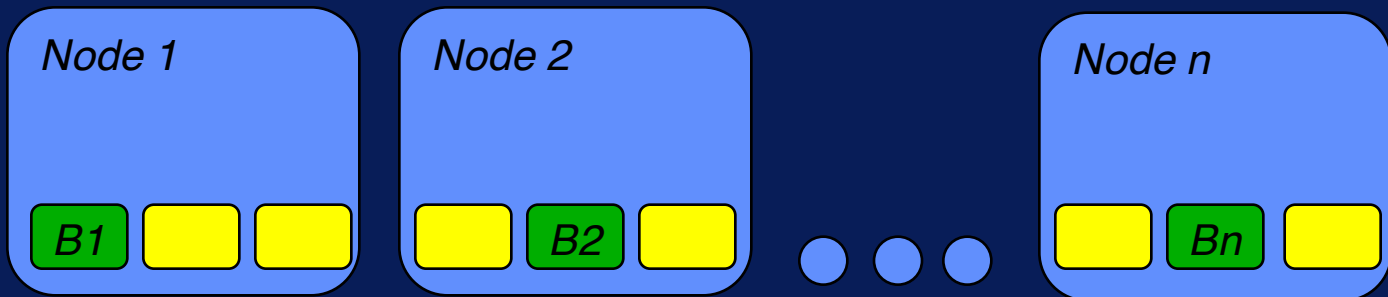


# Hadoop Execution Environment

- *Learn about execution environments in Hadoop.*
- *Limitations of classic MapReduce framework.*
- *New frameworks like YARN, Tez, Spark to compliment classic MapReduce.*

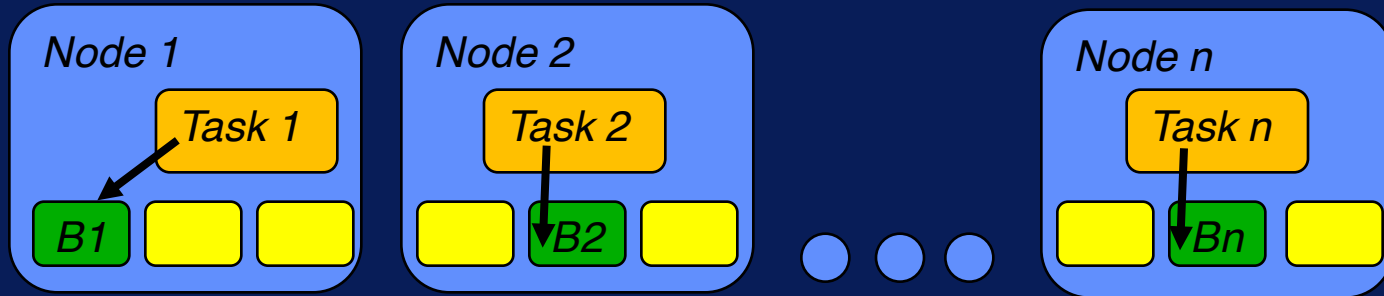
# Recall Hadoop Architecture

- *Data distributed across nodes*



# Recall Hadoop Architecture

- *Data distributed across nodes*
- *Keep compute task on the node with data.*

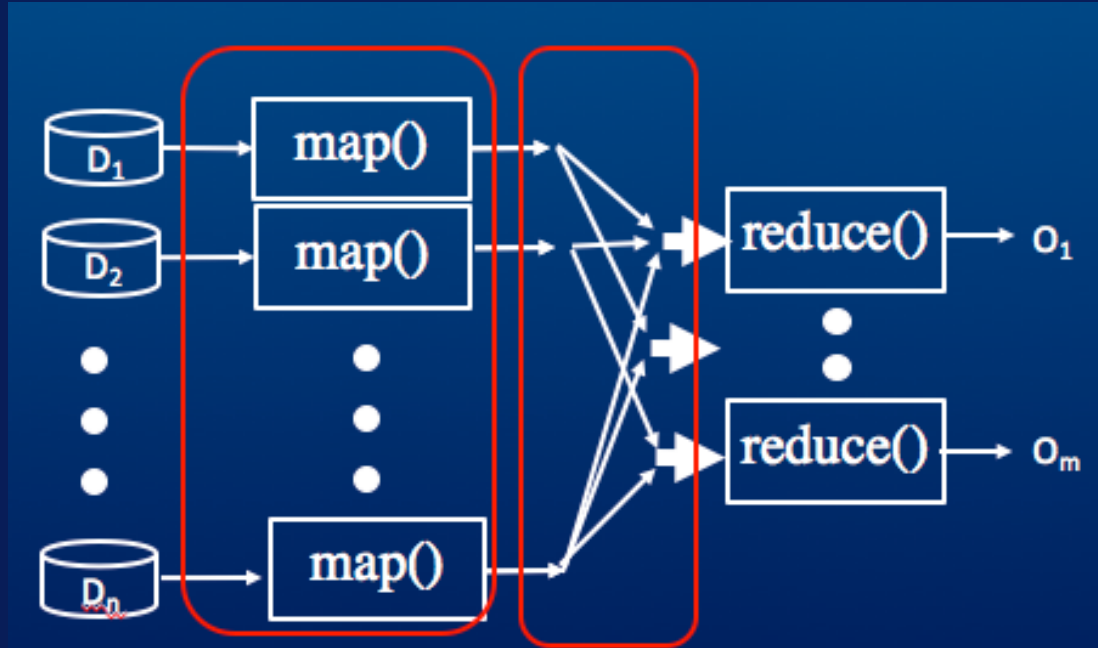


# MapReduce Execution Framework

- Software framework
- *Schedules, monitors, and manages tasks*

# MapReduce Execution Framework

- Works for Applications that fit MapReduce paradigm.*



# NextGen Execution Frameworks

- *What if Application doesn't fit or is not efficient in MapReduce Paradigm?*

# NextGen Execution Frameworks

- *What if Application doesn't fit or is not efficient in MapReduce Paradigm?*
  - *Interactive data exploration*
  - *Iterative data processing*



# NextGen Execution Frameworks

- *Enter: Execution frameworks like YARN, Tez, Spark*
- *Support complex directed acyclic graph (DAG) of tasks.*
- *In memory caching of data*

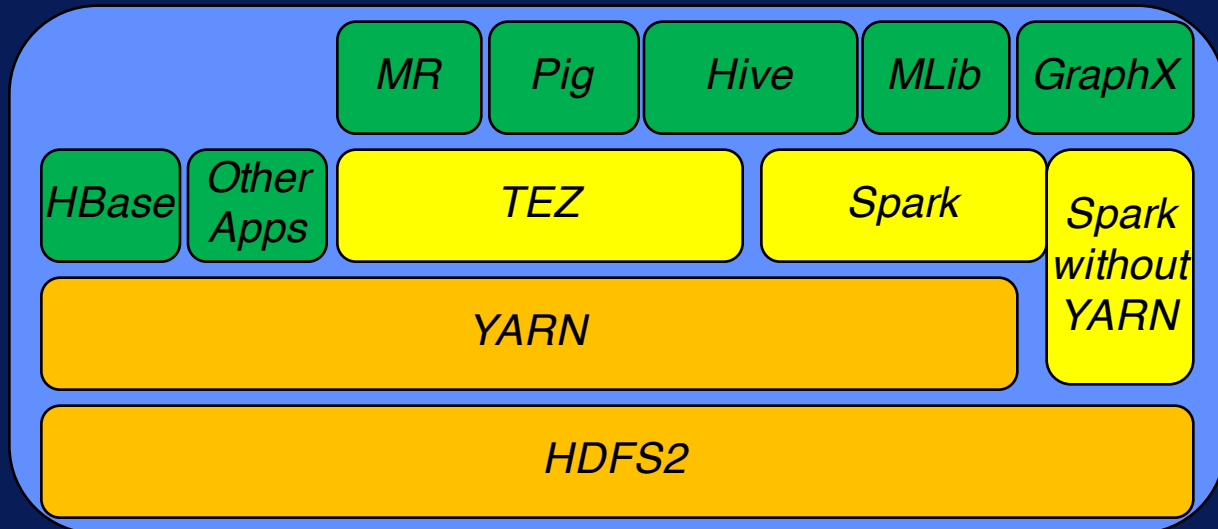
# Lesson 2, Video #2

# Hadoop Execution Environment

- *Layout of new frameworks (YARN, Tez, Spark) in Hadoop environment.*
- *Optimization strategies used in new frameworks.*
- *Examples illustrating use of Tez, Spark.*

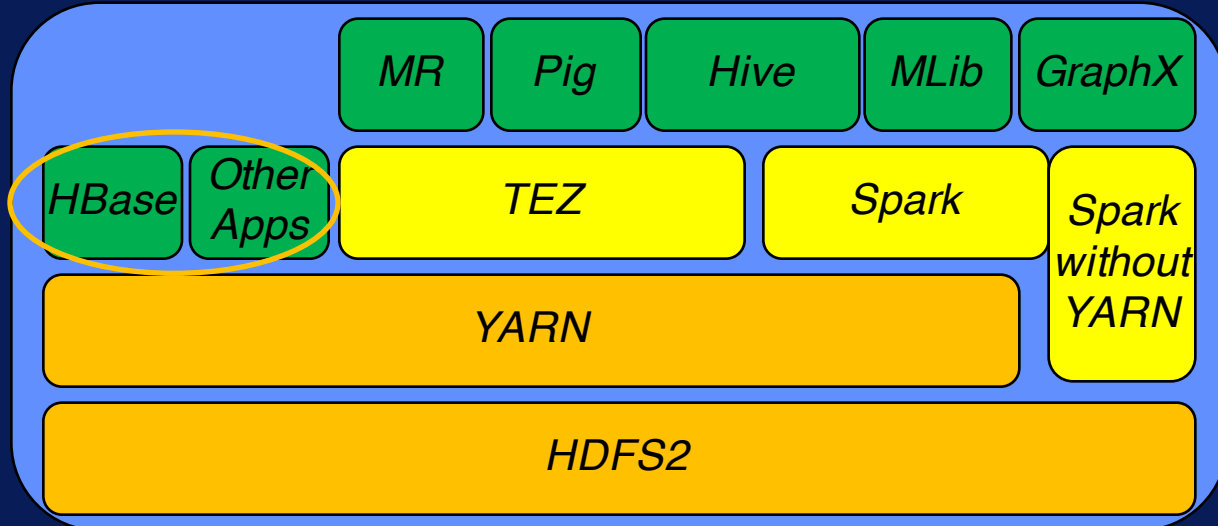
# YARN, Tez, Spark

- Execution frameworks: YARN, Tez, and Spark



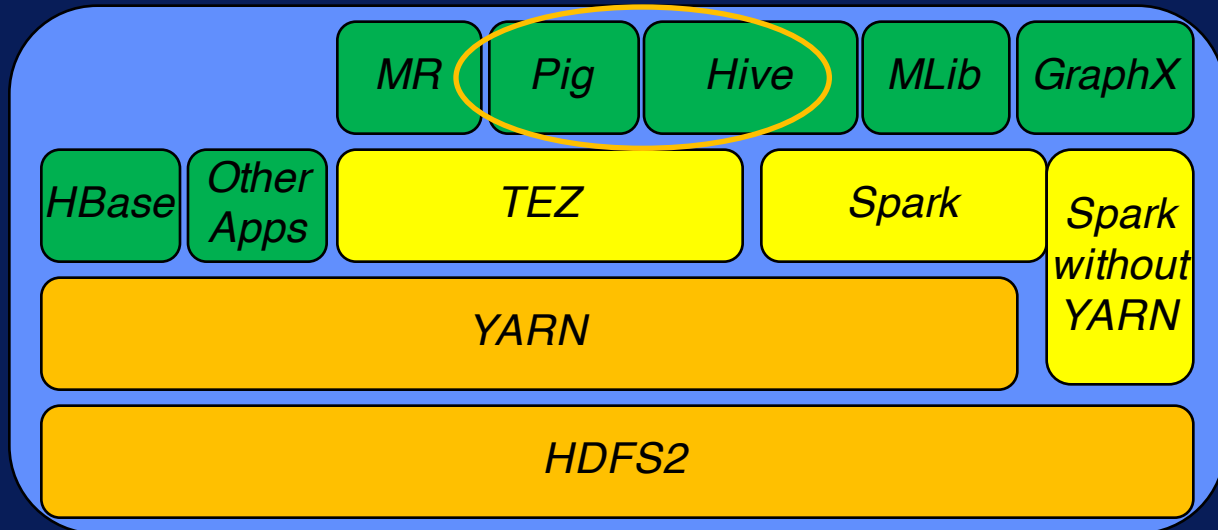
# YARN, Tez, Spark

- Execution frameworks: YARN, Tez, and Spark



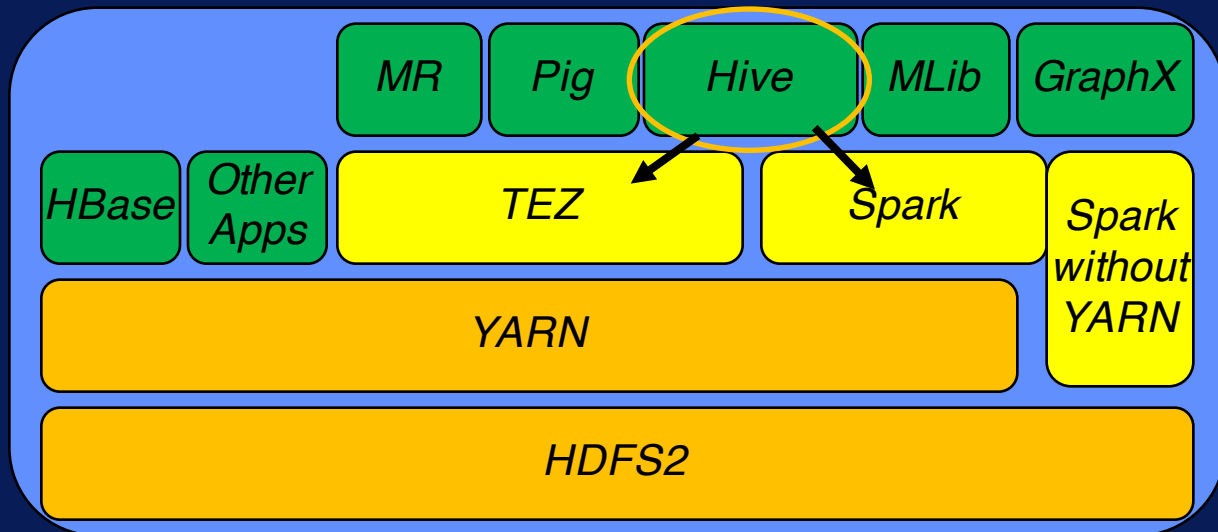
# YARN, Tez, Spark

- Execution frameworks: YARN, Tez, and Spark



# YARN, Tez, Spark

- Execution frameworks: YARN, Tez, and Spark



# YARN

- *MapReduce*
- *Open source/commercial applications*
- *User developed applications*
- *Frameworks like Tez, Spark*



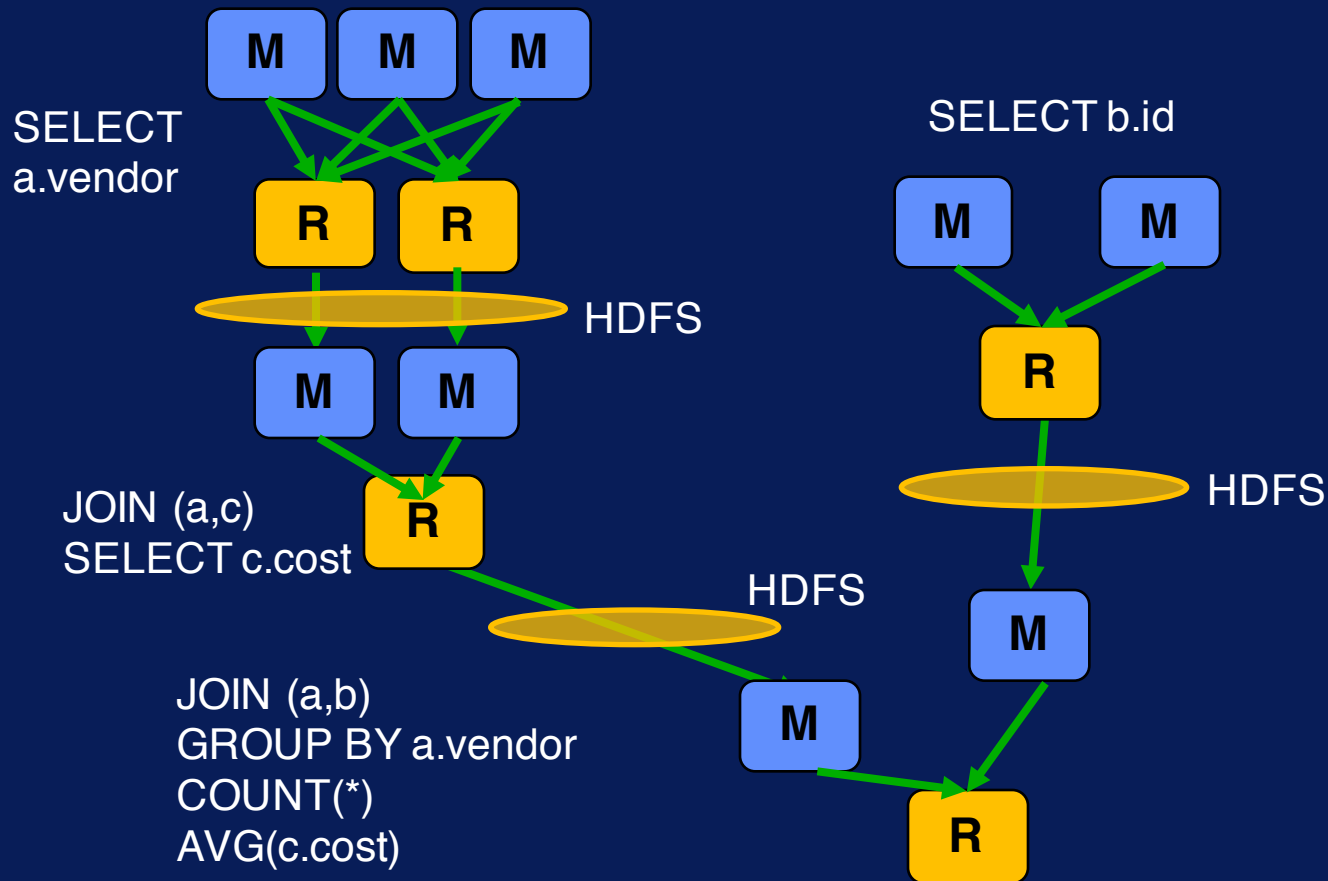
# Tez

- *Dataflow graphs*
- *Custom data types*
- *Can run complex DAG of tasks*
- *Dynamic DAG changes*
- *Resource usage efficiency*

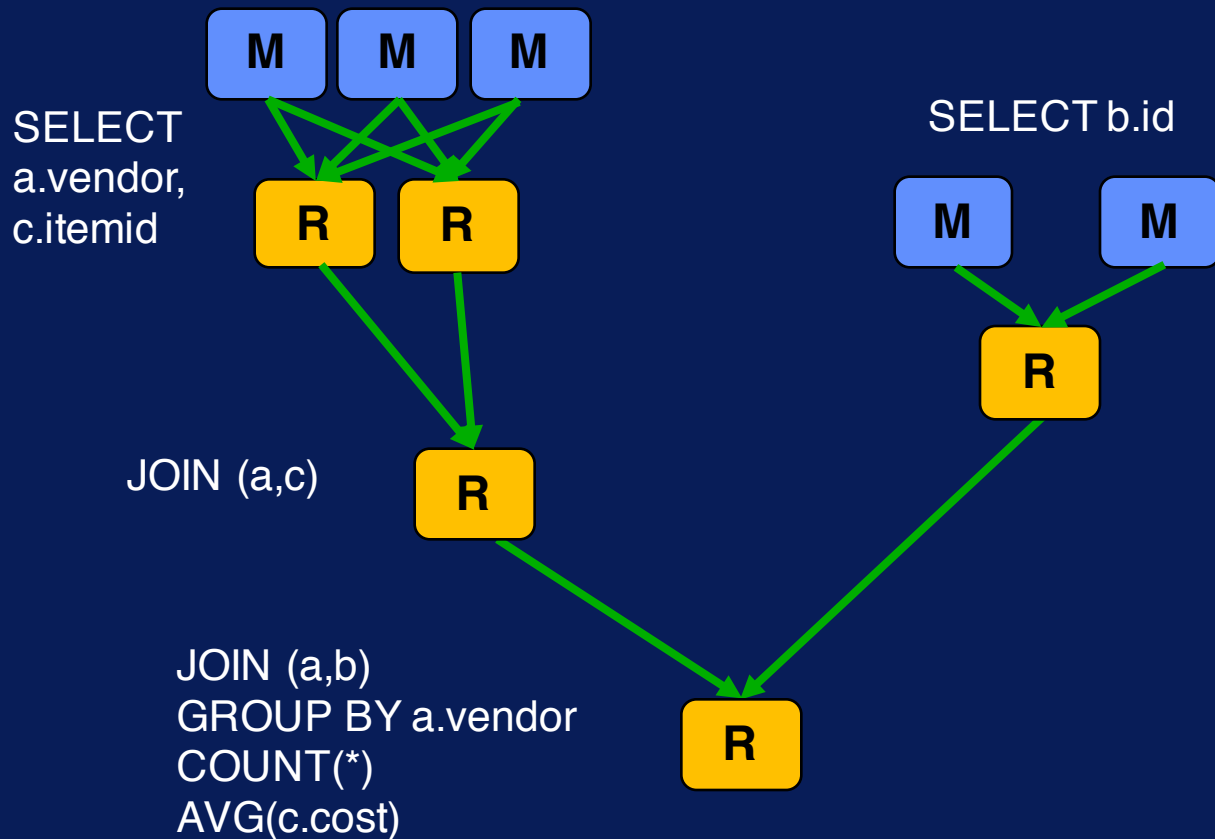
# HIVE on Tez example

```
SELECT a.vendor, COUNT(*), AVG(c.cost) FROM a  
JOIN b ON (a.id = b.id)  
JOIN c ON (a.itemid = c.itemid)  
GROUP BY a.vendor
```

# HIVE Example - MapReduce



# HIVE Example - Tez



# Spark

- *Advanced DAG execution engine*
- *Supports cyclic data flow*
- *In-memory computing*
- *Java, Scala, Python, R*
- *Existing optimized libraries*

# Spark Example

- Logistic Regression example

```
points = spark.textFile(...).map(parsePoint).cache()
w = numpy.random.randn(size = D) # current separating plane
for i in range(ITERATIONS):
    gradient = points.map(
        lambda p: (1 / (1 + exp(-p.y*(w.dot(p.x)))) - 1) * p.y * p.x
    ).reduce(lambda a, b: a + b)
    w -= gradient
print "Final separating plane: %s" % w
```

# Spark Example

- Logistic Regression example

```
points = spark.textFile(...).map(parsePoint).cache()
w = numpy.random.randn(size = D) # current separating plane
for i in range(ITERATIONS):
    gradient = points.map(
        lambda p: (1 / (1 + exp(-p.y*(w.dot(p.x)))) - 1) * p.y * p.x
    ).reduce(lambda a, b: a + b)
    w -= gradient
print "Final separating plane: %s" % w
```

# Lesson 2, Video #3



# Hadoop Resource Scheduling

- *Learn about resource management*
- *Different kinds of scheduling algorithms*
- *Types of parameters that can be controlled.*

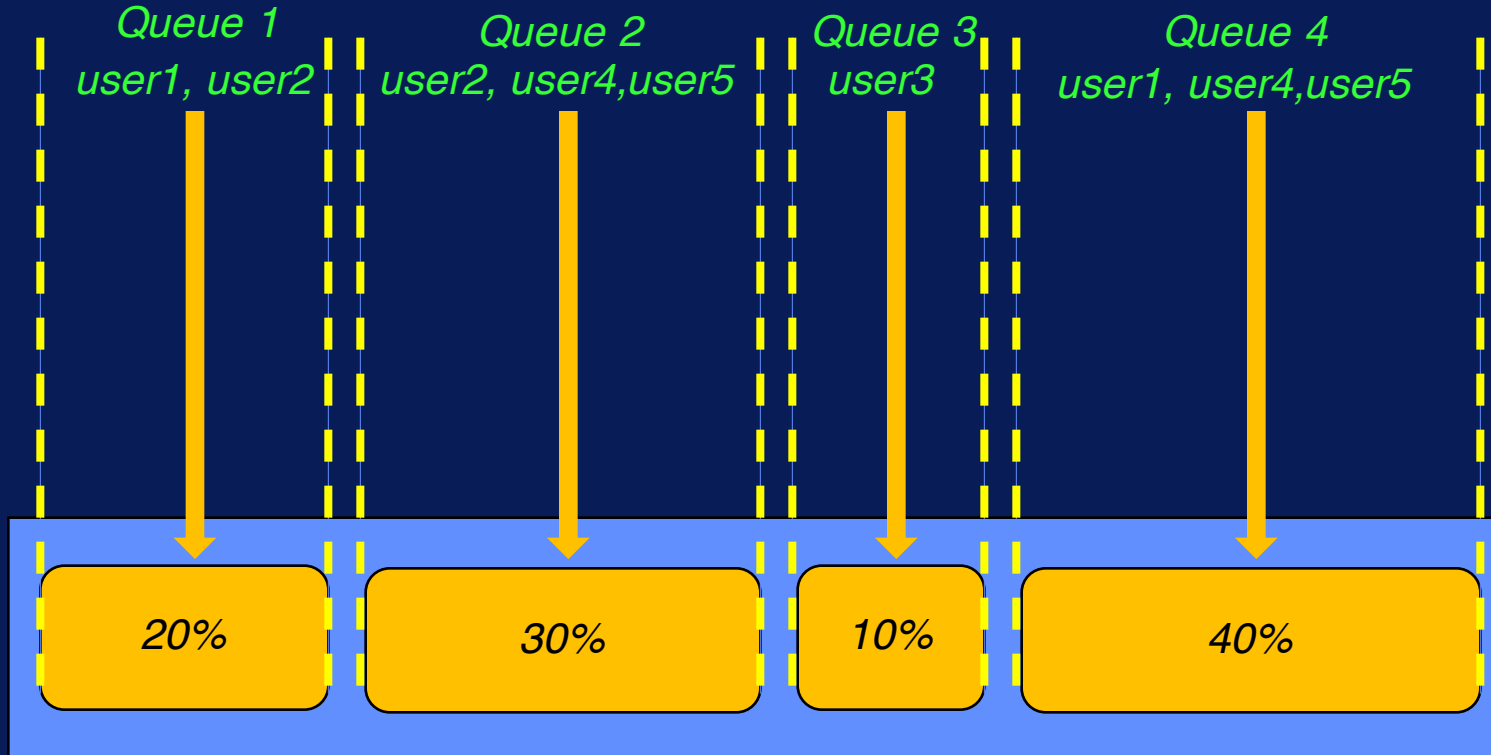
# Motivation for Schedulers

- *Various execution engines/options*
- *Scheduling, Performance*
- *Control of resources between components*

# Schedulers

- *Default – First in First out (FIFO)*
- *Fairshare*
- *Capacity*

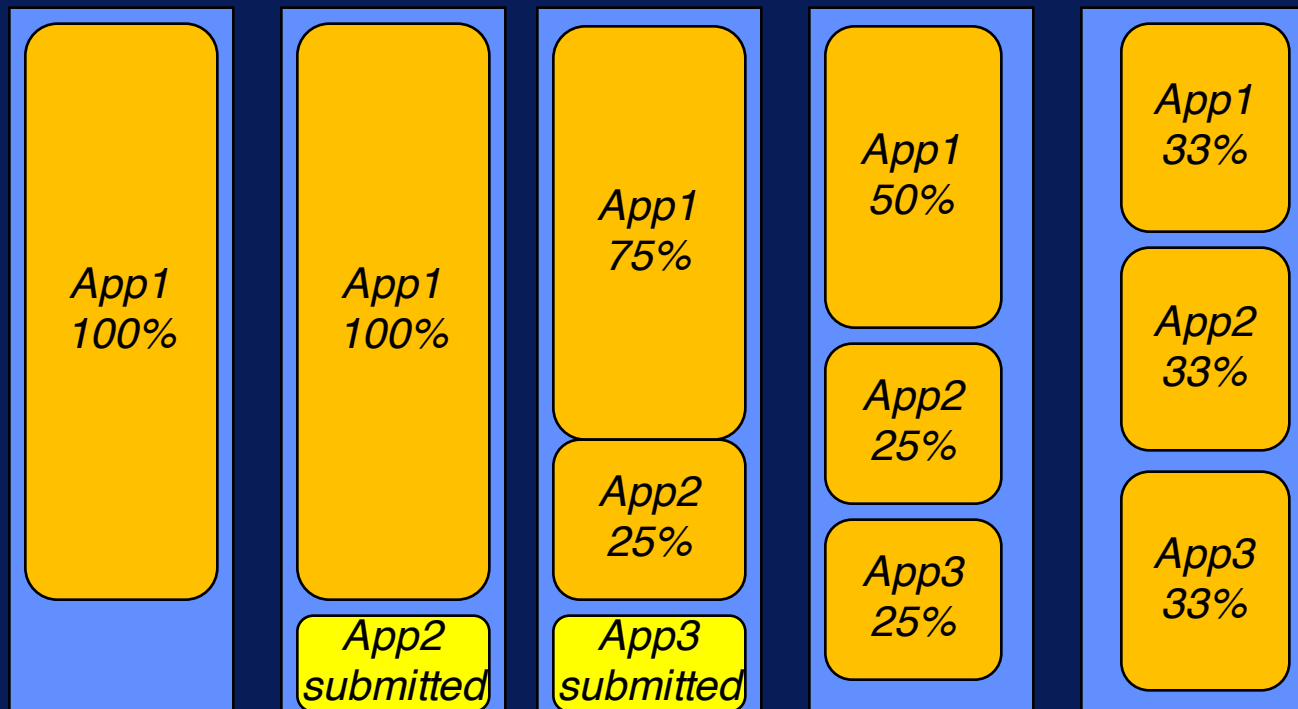
# Capacity Scheduler



# Capacity Scheduler

- *Queues and sub-queues*
- *Capacity Guarantee with elasticity*
- *ACLs for security*
- *Runtime changes/draining apps*
- *Resource based scheduling*

# Fairshare Scheduler



Time



# Fairshare Scheduler

- *Balances out resource allocation among apps over time.*
- *Can organize into queues/sub-queues*
- *Guarantee minimum shares*
- *Limits per user/app*
- *Weighted app priorities*

# Summary of resource scheduling

- *Default is FIFO*
- *Fairshare and Capacity schedulers*
- *Queues/sub-queues possible*
- *User/App based limits*
- *Resource limits*
- *Vendors usually provide additional mechanisms to allocate resources*