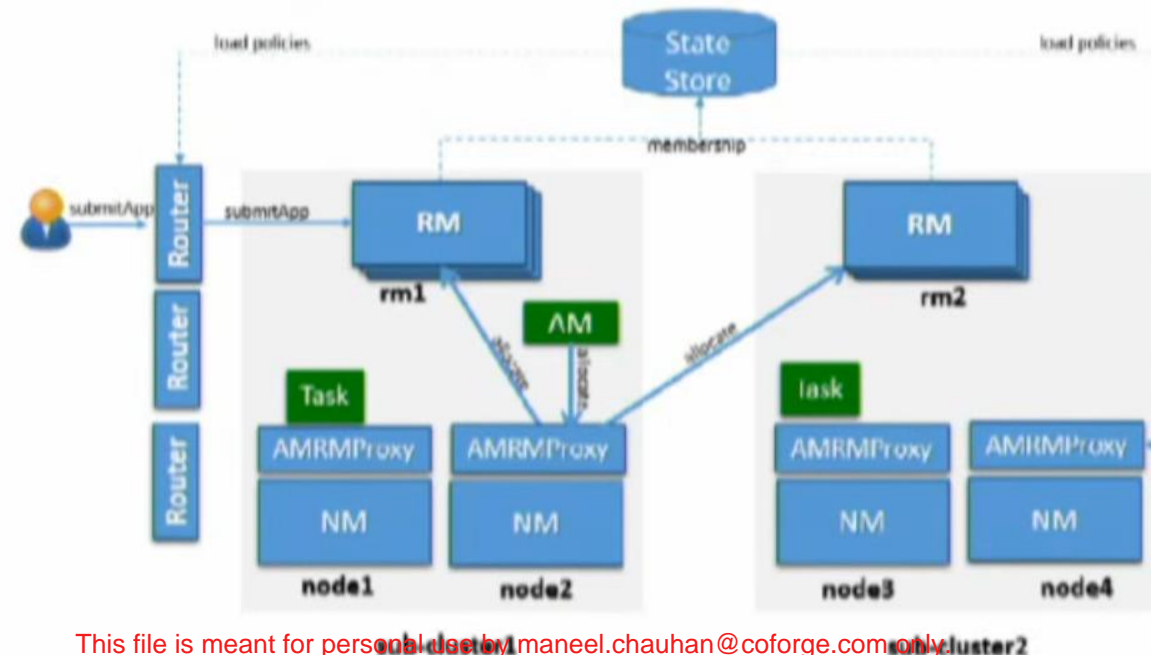


What's new in Hadoop 3?

YARN Federation

- Enables applications to scale to **100k** of thousands of nodes
- Federation divides a large (10-100k nodes) cluster into smaller units called sub-clusters
- Federation negotiates with sub-clusters RM's and provide resources to the application
- Applications can schedule tasks on any node



This file is meant for personal use by maneel.chauhan@coforce.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Moving towards Global & Fast Scheduling

- **YARN-5139**
- **Problems**
 - Current design of one-node-at-a-time allocation cycle can lead to suboptimal decisions.
 - Several coarse grained locks.
- **With this, we improved to**
 - Look at several nodes at a time
 - Fine grained locks
 - Multiple allocator threads
 - YARN scheduler can allocate **3k+ containers per second** \approx 10 mil allocations / hour!
 - **10X throughput gains**
 - Much better placement decisions

Better placement strategies (YARN-6592)

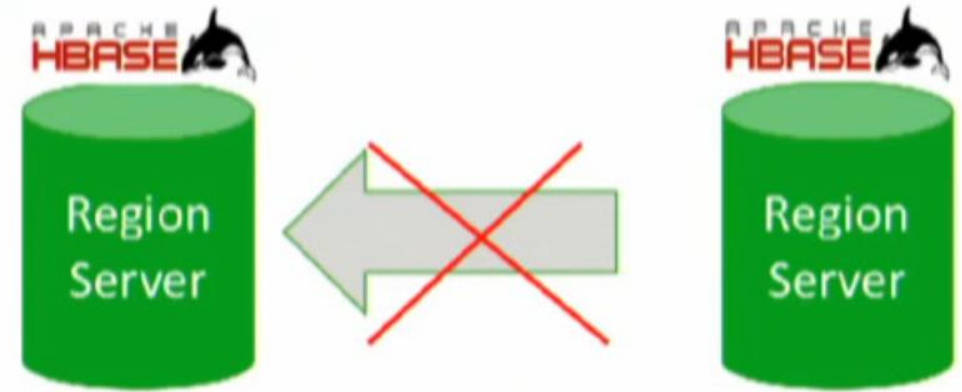
- **Past**
 - Supported constraints in form of Node Locality
- **Now YARN can support a lot more use cases**
 - Co-locate the allocations of a job on the same rack (**affinity**)
 - Spread allocations across machines (**anti-affinity**) to minimize resource interference
 - Allow up to a specific number of allocations in a node group (**cardinality**)

Better placement strategies (YARN-6592)

- Affinity

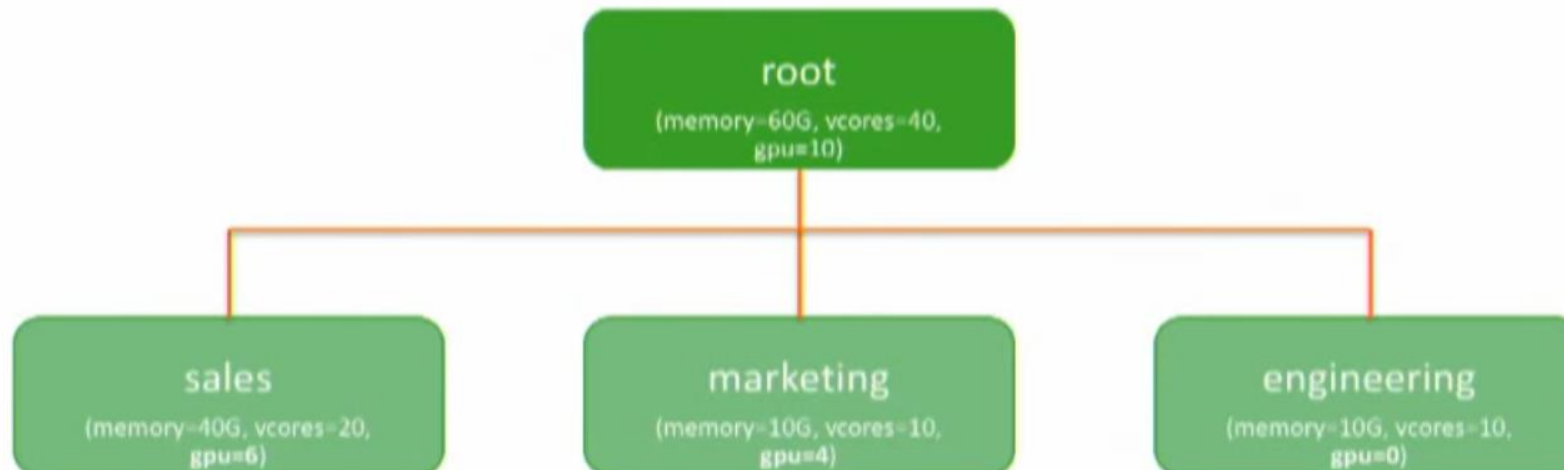


- Anti-affinity



Absolute Resources Configuration in CS – YARN-5881

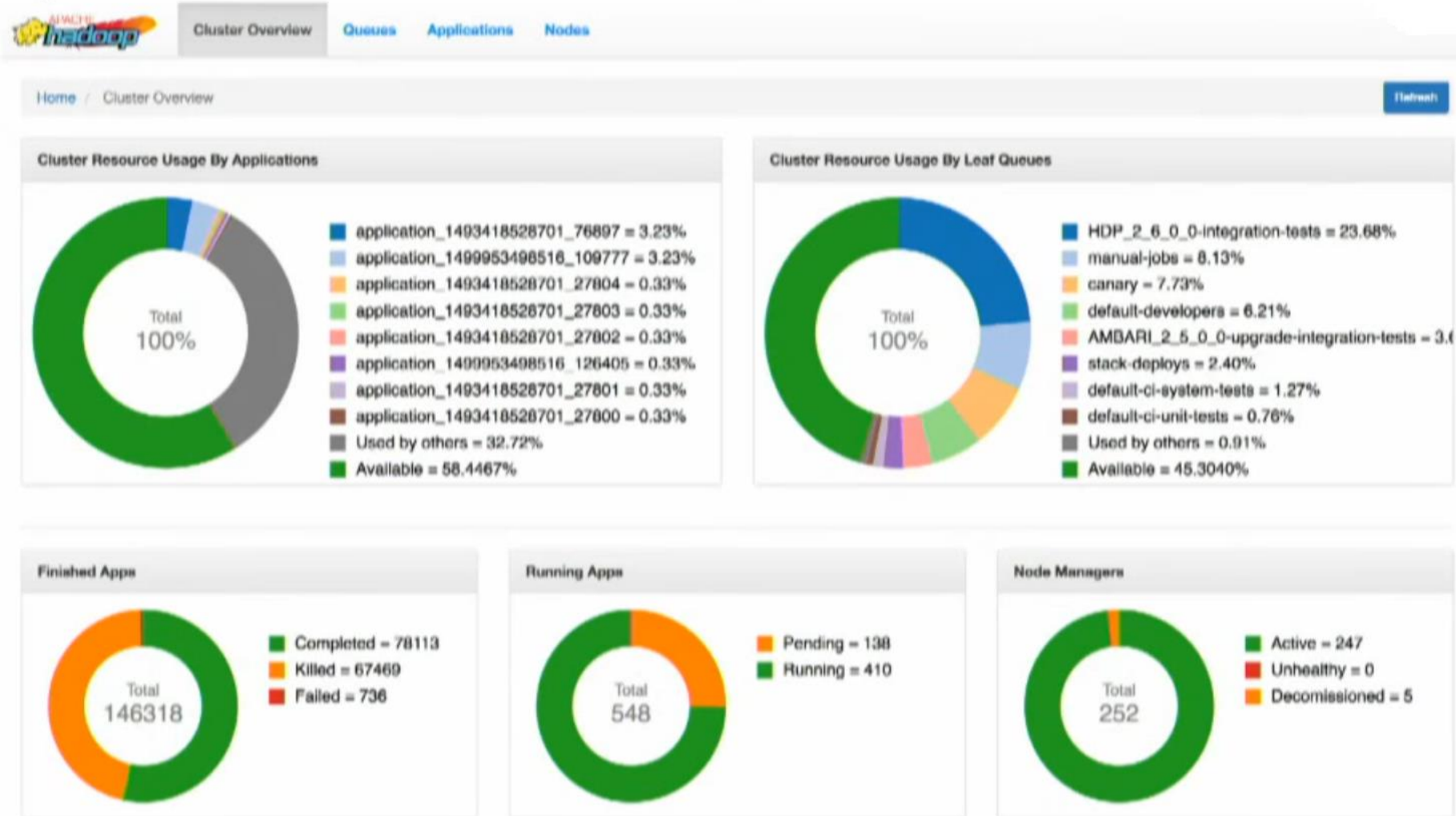
- The cloud model! “Give me X resources, not X%”
- Gives ability to configure Queue resources as below
`<memory=24GB, vcores=20, yarn.io/gpu=2>`
- Enables admins to assign different quotas of different resource-types
- No more **“Single percentage value limitation for all resource-types”**



Auto Creation of Leaf Queues - YARN-7117

- **Easily map a queue** explicitly to user or group with out additional configs
- For e.g, User X comes in, automatically create a queue for user X with a templated capacity requirements
- **Auto created Queues** will be
 - created runtime based on user mapping
 - cleaned up after use
 - adhering to ACLs

Usability: UI 1/2



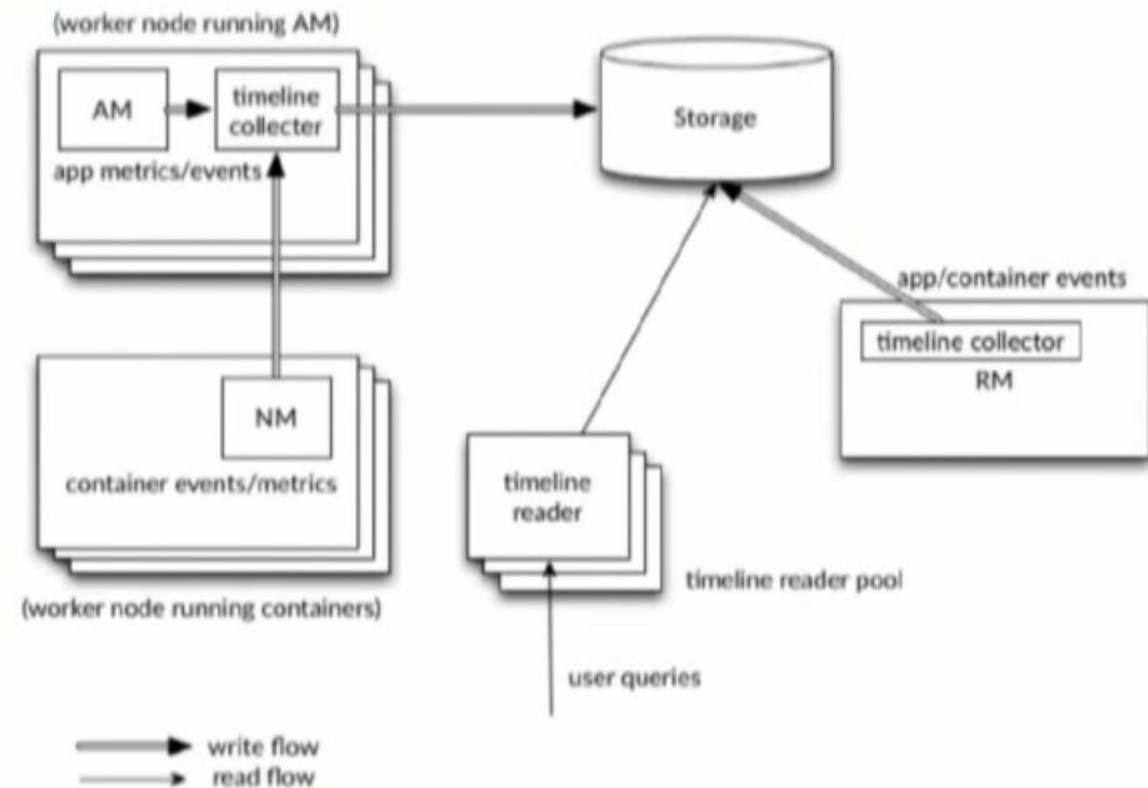
Proprietary content. ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

This file is meant for personal use by maneel.chauhan@coforge.com only.

Sharing or publishing the contents in part or full is liable for legal action.

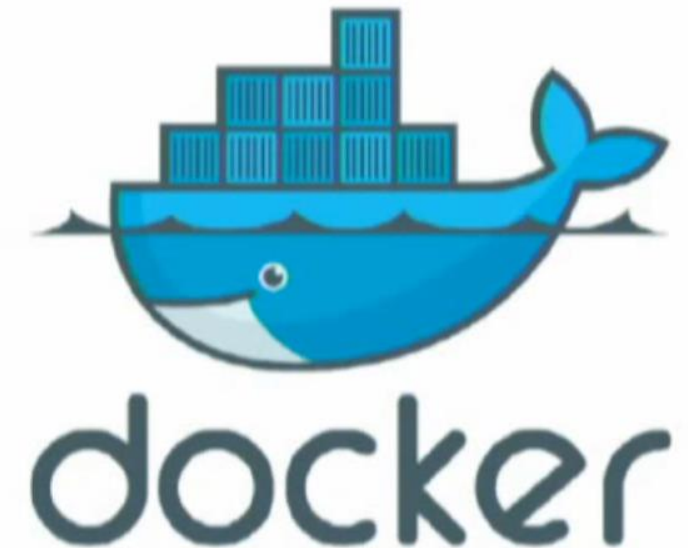
Timeline Service 2.0

- Understanding and Monitoring a Hadoop cluster itself is a BigData problem
 - Using **HBase as backend** for better scalability for read/write
 - More robust storage fault tolerance
 - Migration and compatibility with v.1.5



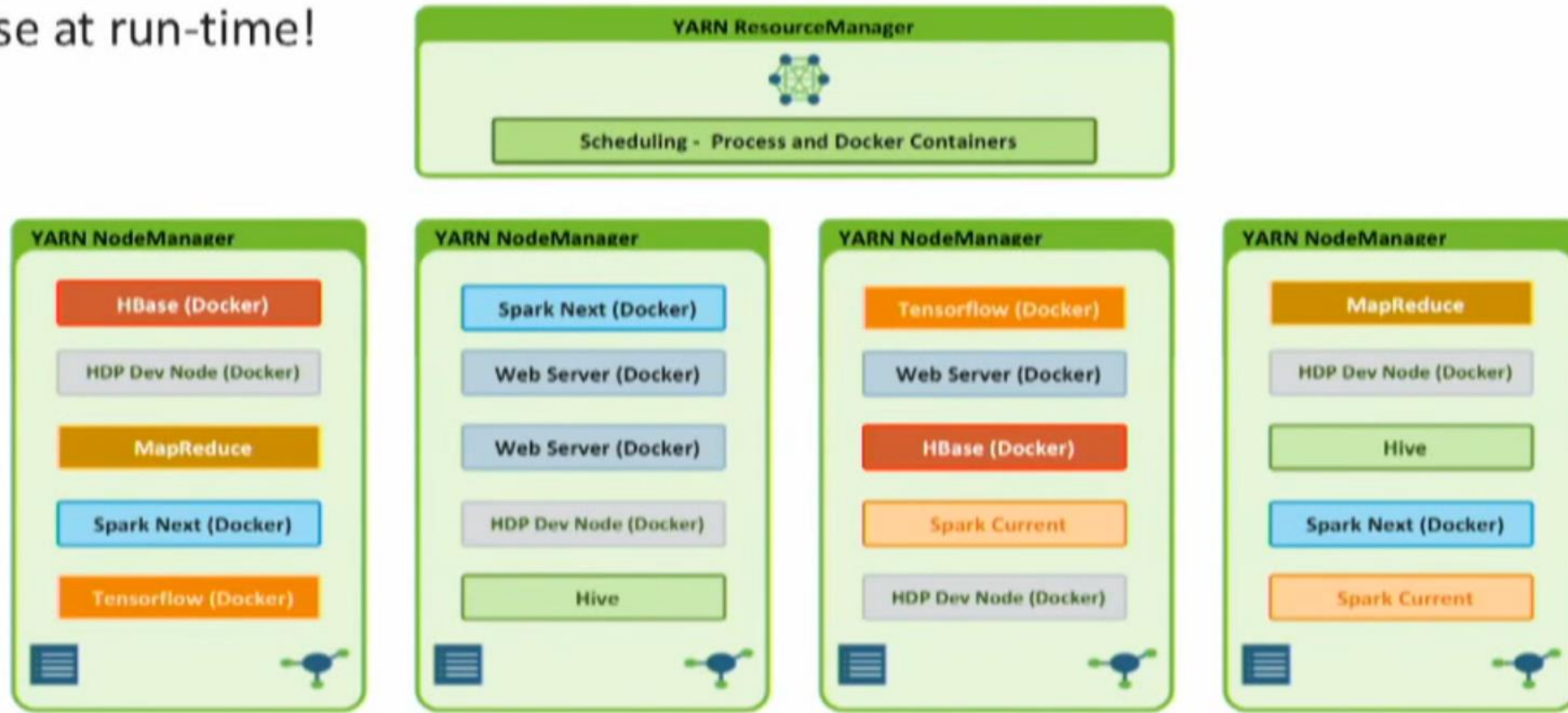
Containers

- Better Packaging model
 - Lightweight mechanism for packaging and resource isolation
 - Popularized and made accessible by Docker
- Native integration ++ in YARN
 - Support for “Container Runtimes” in LCE: YARN-3611
 - Process runtime
 - **Docker runtime**



Containers

- Run both with and without docker on the same cluster
- Choose at run-time!



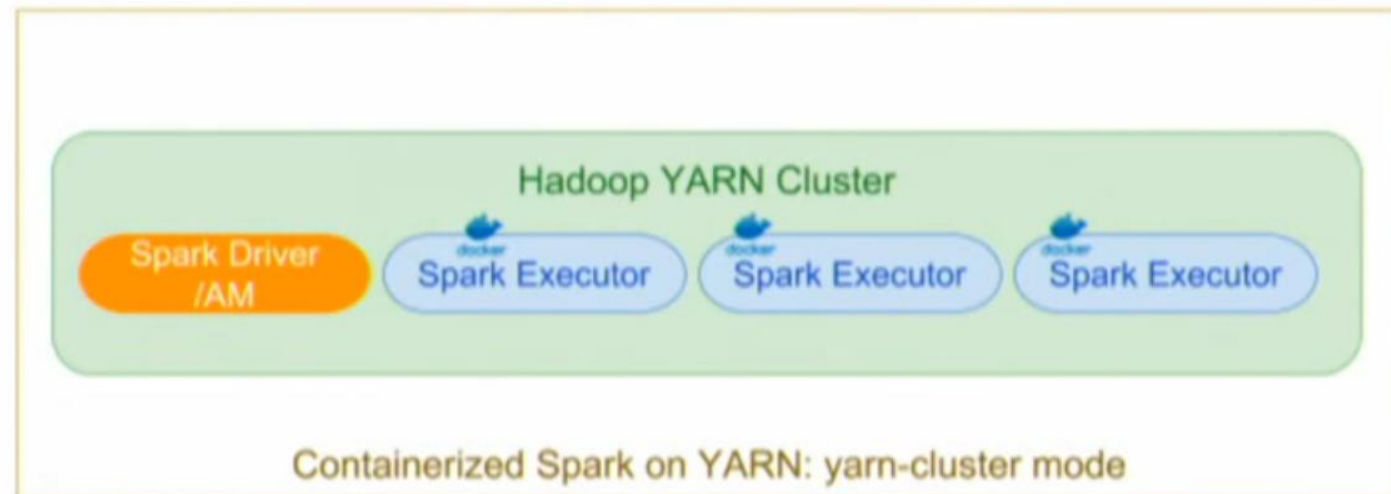
Proprietary content. ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

This file is meant for personal use by maneel.chauhan@coforge.com only.

Sharing or publishing the contents in part or full is liable for legal action.

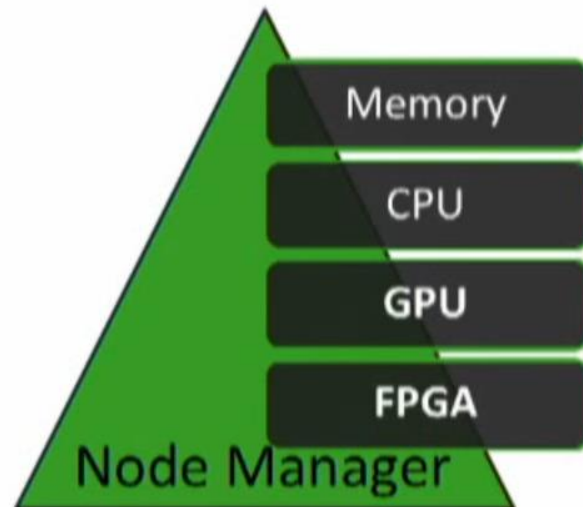
Spark on Docker in YARN

- **Apache Spark** applications have a complex set of required software dependencies
- Docker on YARN helps to solve Spark **package isolation issues with**
 - **PySpark** - Python versions, packages
 - **R** - packages



Resource profiles and custom resource types

- **YARN** supported only **Memory** and **CPU**
- **Now**
 - A generalized vector for all resources
 - Admin could add arbitrary resource types!



- Ease of resource requesting model using profiles for apps

Profile	Memory	CPU	GPU
Small	2 GB	4 Cores	0 Cores
Medium	4 GB	8 Cores	0 Cores
Large	16 GB	16 Cores	4 Cores

GPU support on YARN

- **Why?**
 - No need to setup separate clusters
 - Leverage shared compute!
- **Why need isolation?**
 - Multiple processes use the single GPU will be:
 - Serialized.
 - Cause OOM easily.
- **GPU isolation on YARN:**
 - Granularity is for per-GPU device.
 - Use cgroups / docker to enforce isolation.

FPGA on YARN

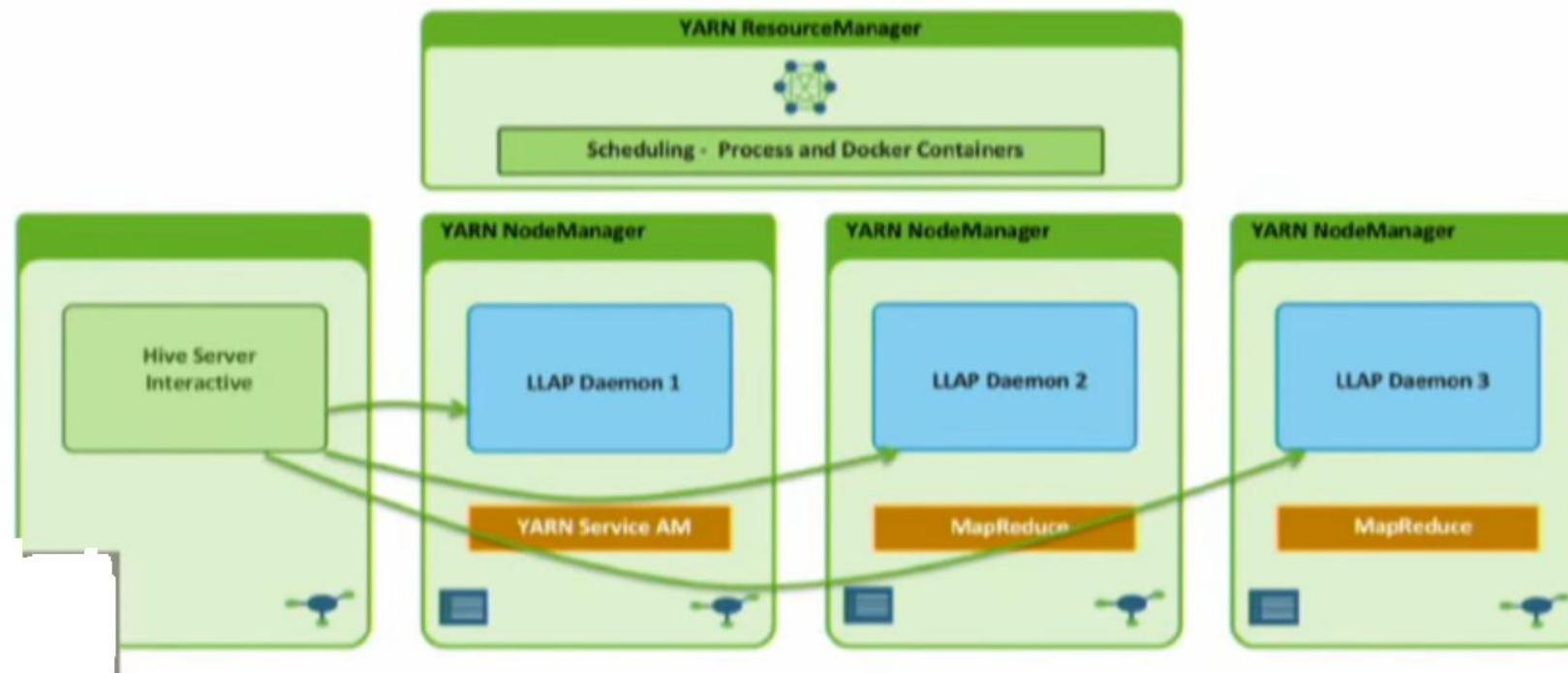
- FPGA isolation on YARN: .
 - Granularity is for per-FPGA device.
 - Use Cgroups to enforce the isolation.
- Currently, only Intel OpenCL SDK for FPGA is supported. But implementation is extensible to other FPGA SDK.

Services support in YARN

- A native **YARN services framework**
 - YARN-4692
 - [Umbrella] Native YARN framework layer for services and beyond
 - Apache Slider retired from Incubator – lessons and key code carried over to YARN
- Simplified discovery of services via **DNS mechanisms**: YARN-4757
 - *regionserver-0.hbase-app-3.hadoop.yarn.site*
- **Application & Services upgrades**
 - “Do an upgrade of my HBase app with minimal impact to end-users”
 - YARN-4726

LLAP on YARN

- **Apache Hive LLAP** is a key long running application
 - Used for query processing
 - Designed to run on a shared multi-tenant YARN cluster



Proprietary content. ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

This file is meant for personal use by maneel.chauhan@coforge.com only.

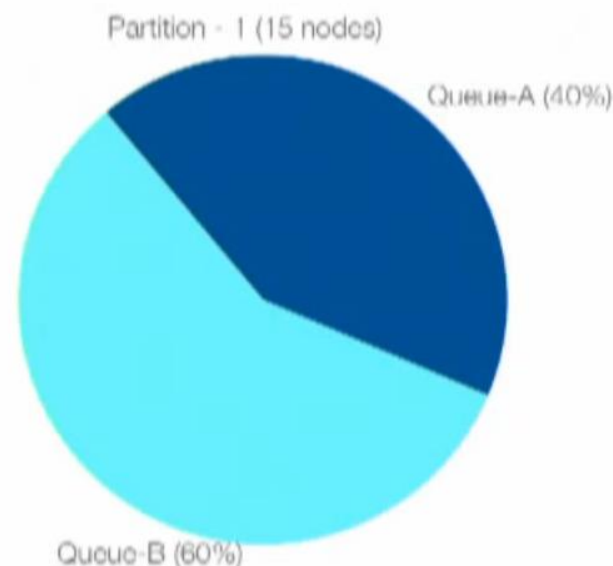
Sharing or publishing the contents in part or full is liable for legal action.

Application Timeout – YARN-3813

- Controlling running time of workloads in YARN
- Define lifetime for an application anytime for YARN to manage.
- “Give me resources for this app/service but kill it after 15 days”

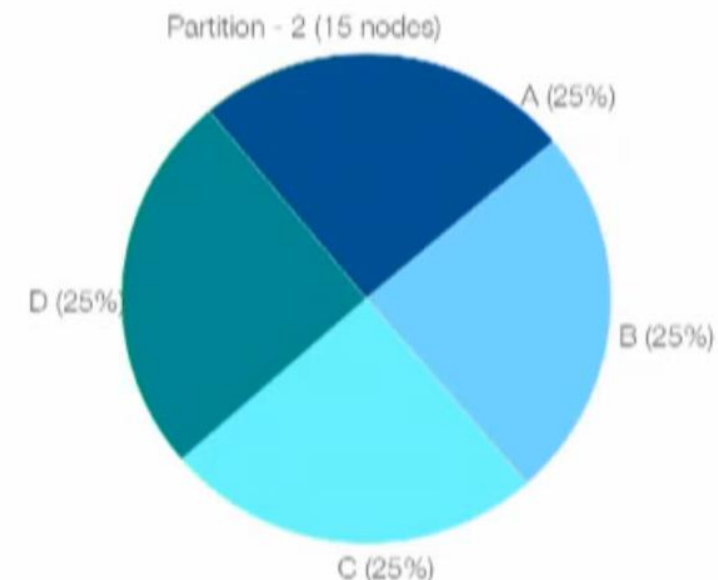
Node Attributes (YARN-3409)

- “Take me to a node with JDK 10”
- Node Partition vs. Node Attribute
- Partition:
 - One partition for one node
 - ACL
 - Shares between queues
 - Preemption enforced.
- Attribute:
 - For container placement
 - No ACL/Shares on attributes
 - First-come-first-serve



Node 1
os.type=ubuntu
os.version=14.10
glibc.version=2.20
JDK.version=8u20

Node 2
os.type=RHEL
os.version=5.1
GPU.type=x86_64
JDK.version=7u20

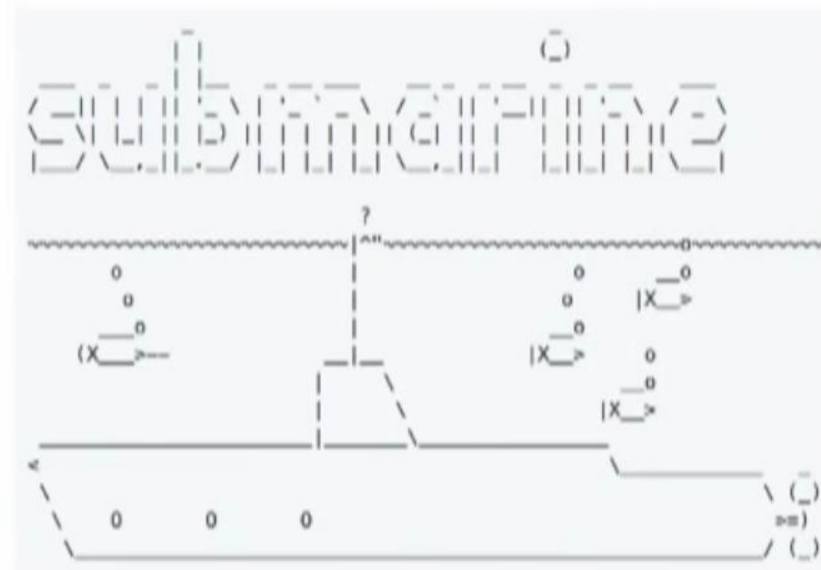


Node 16
os.type=windows
os.version=7
JDK.version=8u20

Node 17
os.type=SUSE
os.version=12
GPU.type=i686
JDK.version=7u20

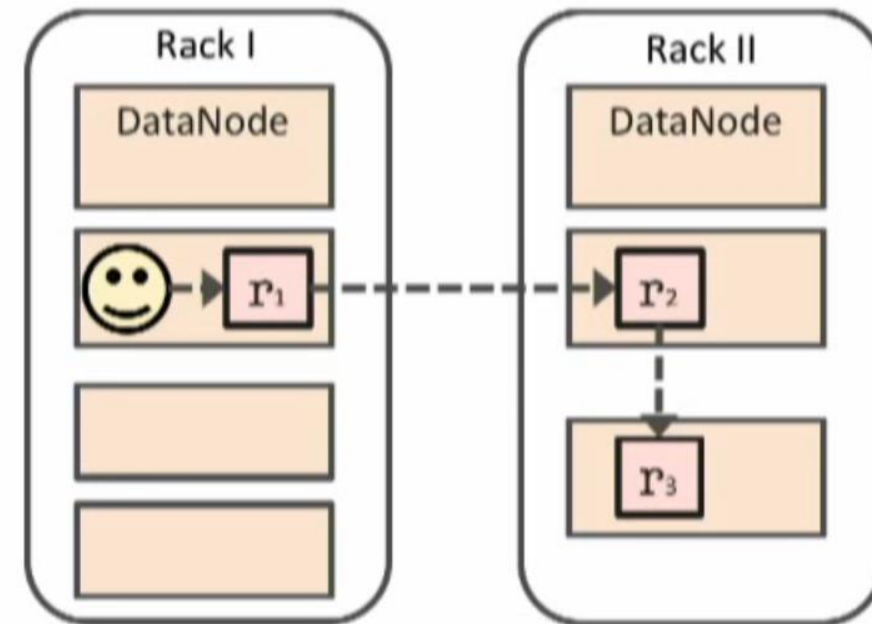
TensorFlow on YARN: {Submarine} Project

- Run deep learning workloads on the same cluster as analytics, stream processing etc!
 - It allows jobs easy access data/models in HDFS and other storages.
 - Support run distributed Tensorflow jobs with simple configs.
 - Support run user-specified Docker images.
 - Support specify GPU and other resources.
 - Support launch tensorboard for training jobs if user specified.



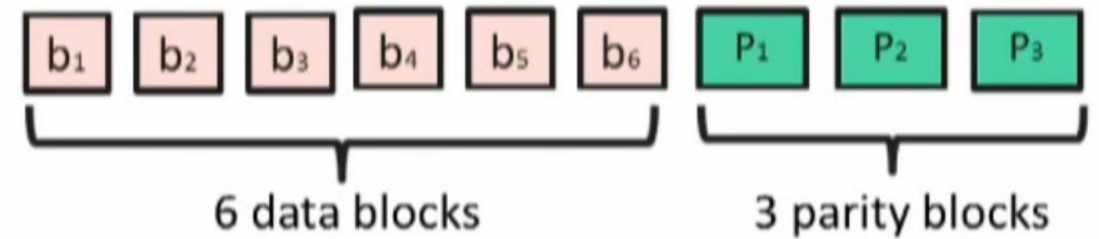
Current HDFS Replication Strategy

- Three replicas by default
 - 1st replica on local node, local rack or random node
 - 2nd and 3rd replicas on the same remote rack
 - 3x storage overhead
- Reliability: tolerate 2 failures
- Good data locality
- Fast block recovery
- Expensive for
 - Massive data size e.g. cold data



Erasure Coding

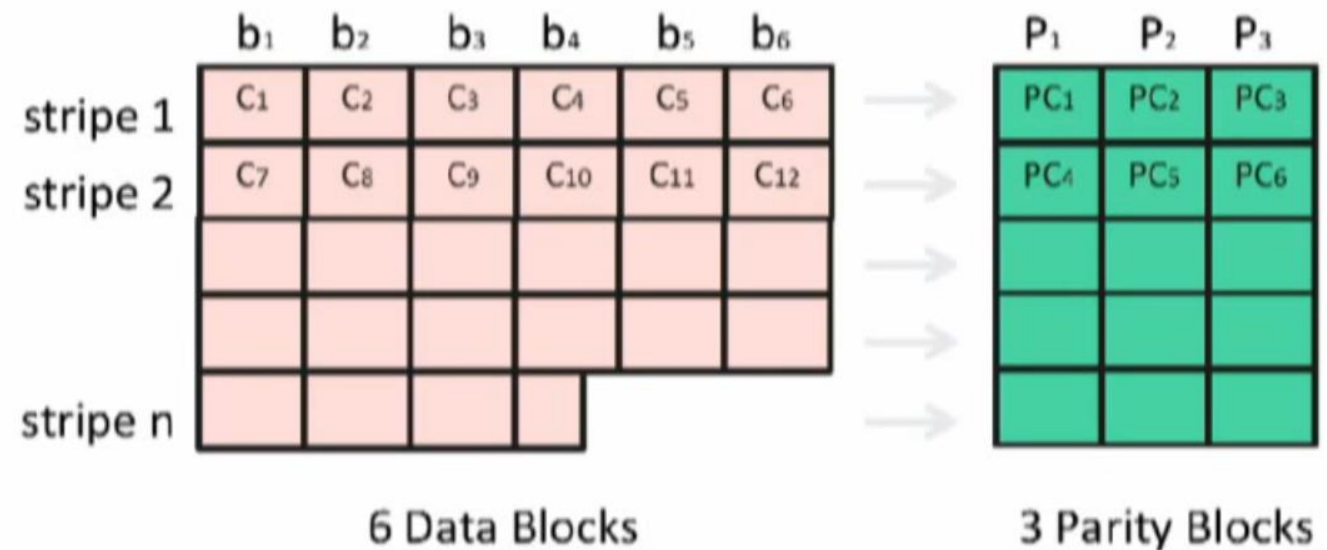
- k data blocks + m parity blocks ($k + m$)
 - Example: Reed-Solomon 6+3
- Reliability: tolerate m failures
- Save disk space
- Save I/O bandwidth on the write path



- 1.5x storage overhead
- Tolerate any 3 failures

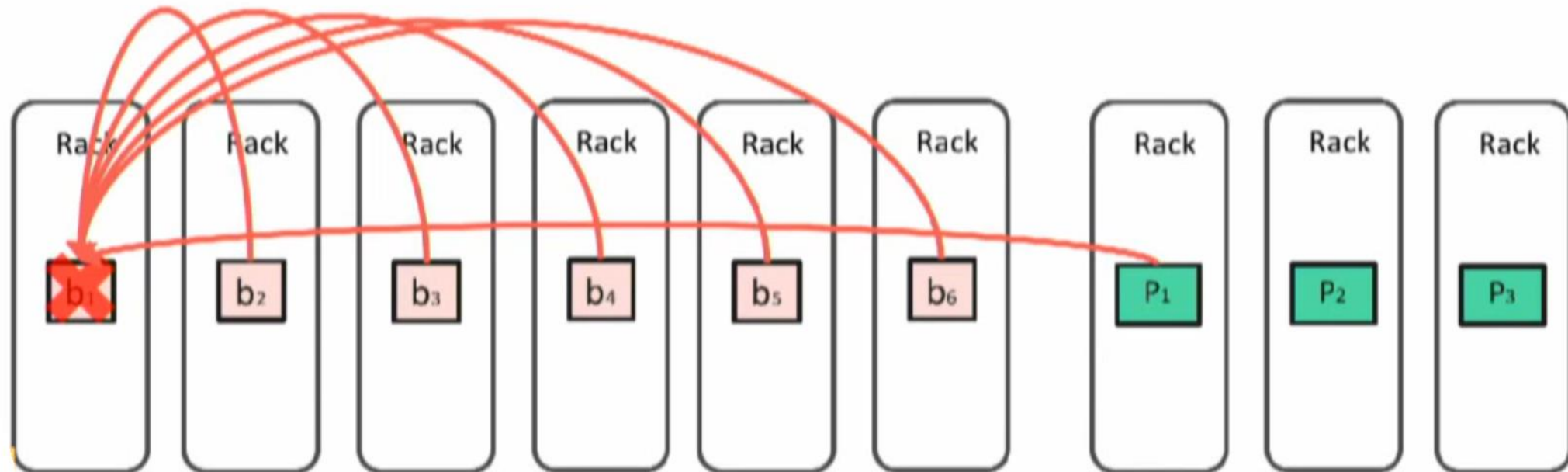
Erasure Coding on Striped Blocks

- Typical cell size – 1MB



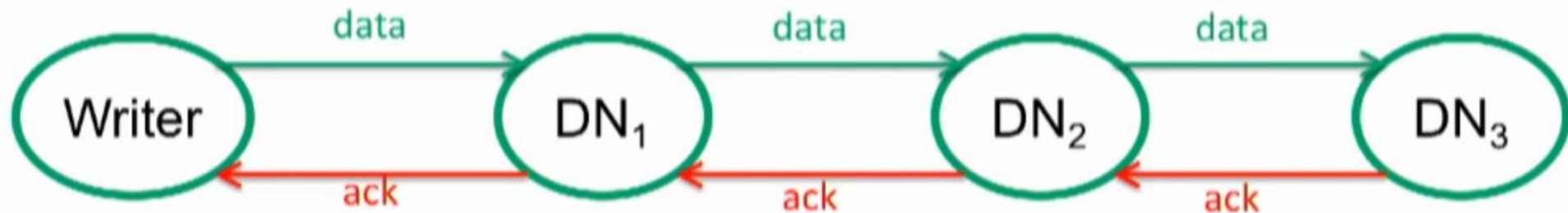
Block Reconstruction

- Block reconstruction overhead
 - Higher network bandwidth cost
 - Extra CPU overhead



Write Pipeline for Replicated Files

- Write to a datanode pipeline
- Durability: Use 3 replicas to tolerate maximum 2 failures
- Read is supported for being written files
- Consistency: Client can start reading from any replica and failover to any other replica to read the same data



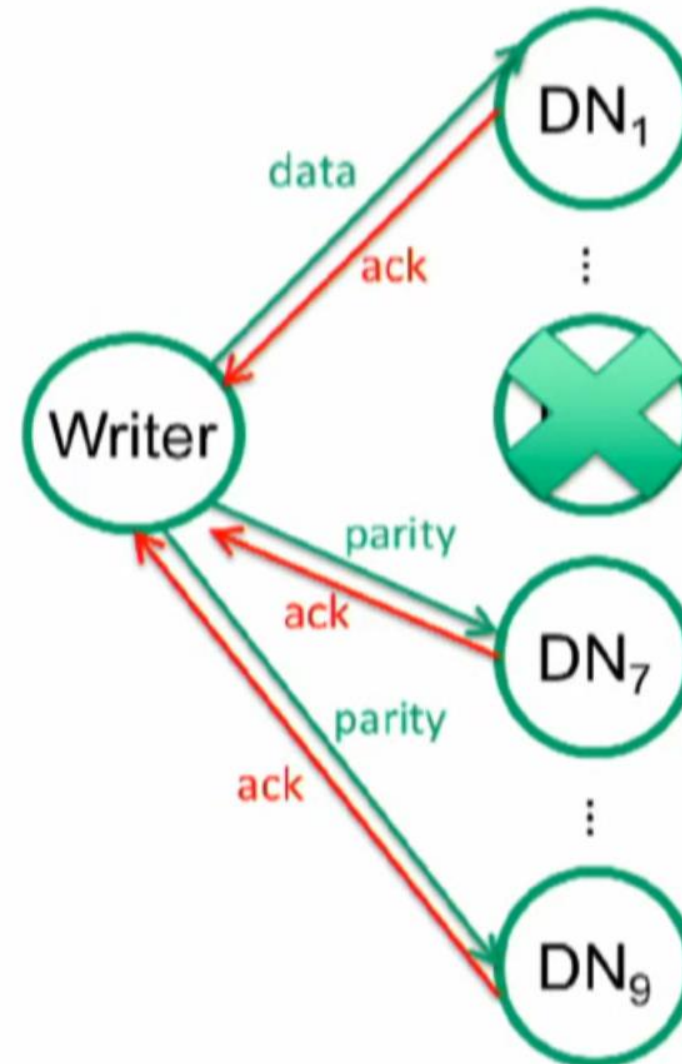
Proprietary content. ©Great Learning. All Rights Reserved. Unauthorized use or distribution prohibited.

This file is meant for personal use by maneel.chauhan@coforge.com only.

Sharing or publishing the contents in part or full is liable for legal action.

Write Failure Handling

- Data Node failure
 - Client ignores the failed datanode and continues writing.
 - Able to tolerate up to 3 failures.
 - Need at least 6 datanodes.
 - Missing blocks will be reconstructed later.
 - Implications for slow writers



EC policies

- Policy Naming Scheme
 - Codec – numDataBlocks(k) – NumParityBlocks(m) - CellSize
- Minimum cluster size is (k+m) nodes
- Some system supported policies
 - RS-6-3-1024k
 - RS-3-2-1024k
 - RS-10-4-1024k

Erasure Coding - Caveats

- Loss of read locality (all reads use the network, no short-circuit reads)
- Not appropriate for smaller clusters
 - Recommend 9 or more racks for the RS-6-3-1024K policy
- Not appropriate for slow writers
 - Long write time increases probability of DataNode failure
 - EC write path cannot replace failed DataNodes (unlike the 3-replica write pipeline)
- Reads and writes are off-rack operations
 - Placement tries to provide tolerance from rack failures

Erasure Coding – Current Technical Limitations

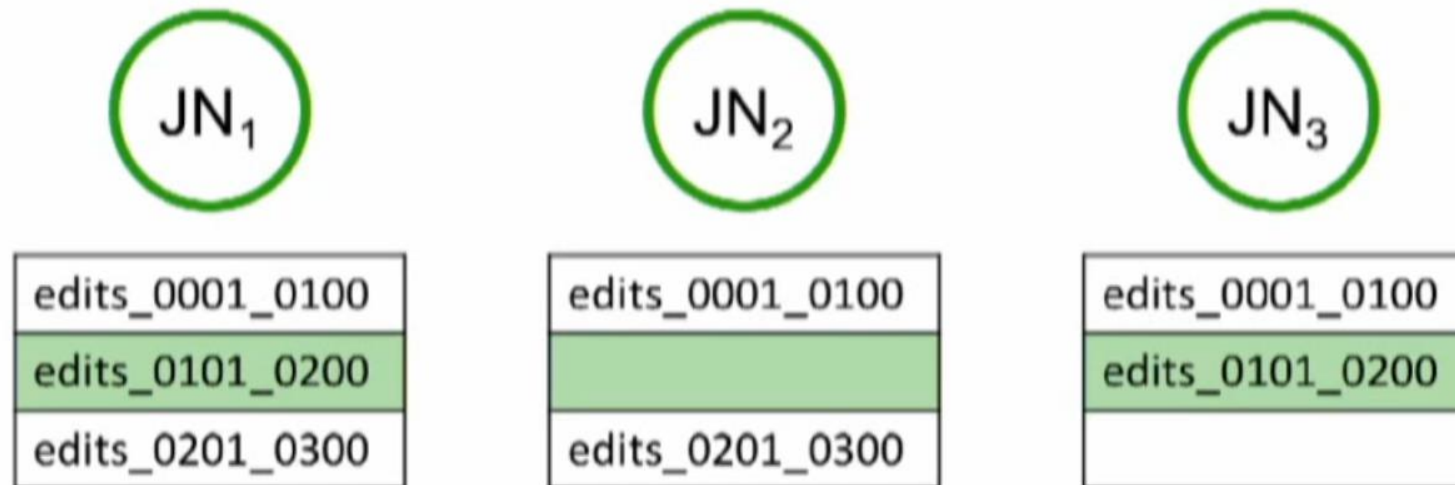
- No in-place encoding
- hflush/hsync are not supported
 - Cannot be used by HBase
- Append is not supported

So when is Erasure Coding recommended?

- Larger clusters
- Lots of cold data
- Medium to Large files
 - At least as large as the stripe width - 9MB for **RS-6-3-1024K**

Why Journal Node Sync?

- JNs can miss log segments and have gaps in the transaction history



3 NameNodes

- 1 Active NN, 1 Standby NN and 3 JNs can tolerate any one node failure
- 1 Active Namenode, 2 Standby Namenodes and 5 JNs
 - Node failure tolerance increases to two
- Configuration similar to 2 Namenode setup
- *dfs.ha.namenodes.mycluster = nn1, nn2, nn3*