

Storage Scale Technical Exchange – New User Group



IBM Storage Scale Principles of Operation

Chris Maestas, CTO Data and AI Storage Solutions
<https://linkedin.com/in/chieftroublemakingofficer>



Agenda

- [Basics of Storage Scale clustering](#)
 - What is on a **node**?
 - What is a **cluster**?
 - Quorum and why we need it
- [Storage and the cluster layer](#)
 - Storage Scale as a clustered file system
 - Working with NSDs
- [Storage and the file system layer](#)
 - Working with file systems
- [File system node roles](#)
 - Token and file system managers
 - Tuning basics
 - [Distributed lock management](#)

Appendices

- [Deployment topologies and building blocks](#)
- [Tiebreaker disk quorum](#)
- [Tuning disk leasing and cluster membership](#)



Philosophy of this session



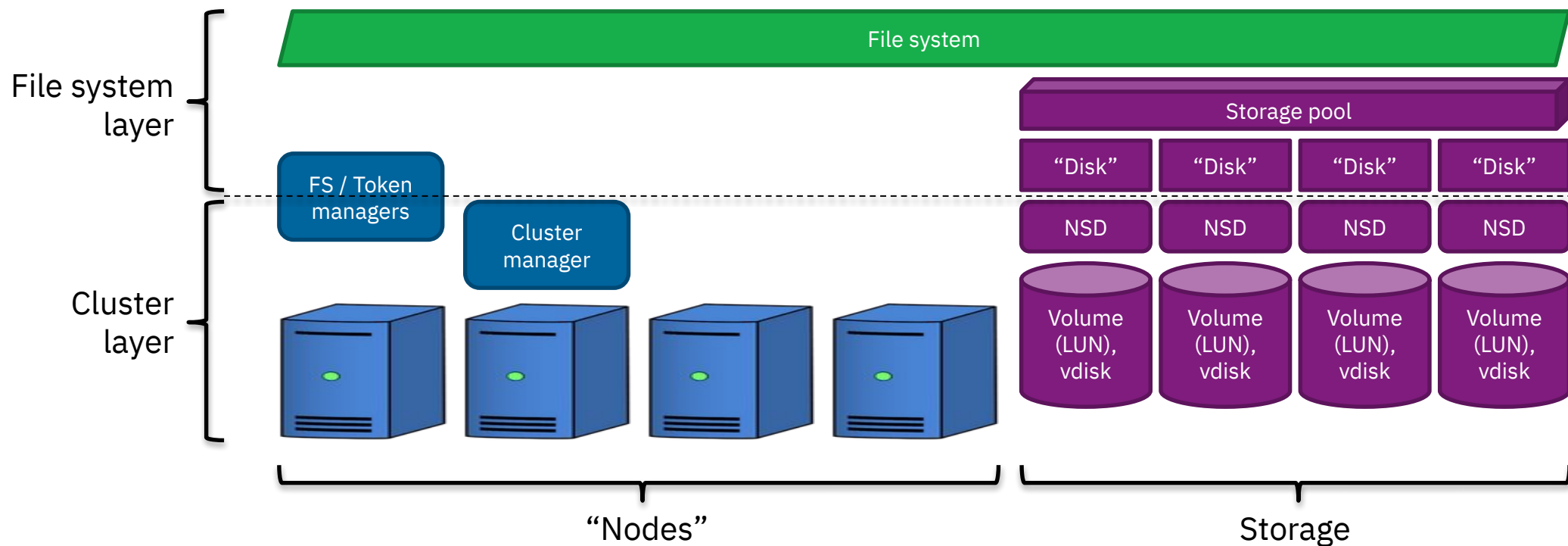
Richard Feynman

“He fixes radios by thinking!”

Overview: Cluster and file system layers in Storage Scale

Rough outline of our approach

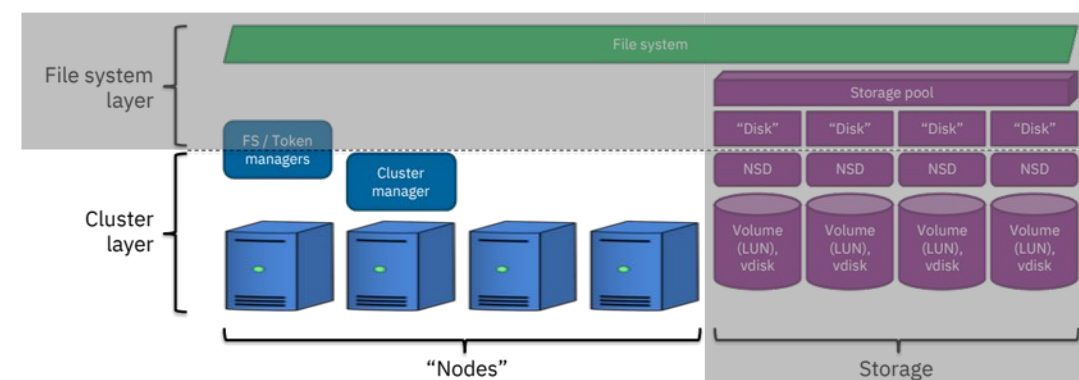
1. Cluster nodes
2. Cluster storage
3. File system storage
4. File system node roles



Chapter 1

Clusters and nodes

... or, where does Storage Scale run?



Storage Scale Node Architecture

Every system (or **node**) running Scale software has these components:

- *Administrative commands* (on Linux or AIX, these are found in /usr/lpp/mmfs/bin, so you probably want to put this directory on your PATH).
- Several *kernel extensions* (Linux nodes will have the Scale open-source portability layer, built into several kernel modules)
- A *multi-threaded daemon*, mmfsd.
- Most *local node state* is stored in the directory tree rooted at /var/mmfs.

Scale leverages the capabilities of the operating system for networking (e.g., bonding) and accessing block storage (e.g., multipath).

- Think of it as a systems-level software application.

```
# ls /usr/lpp/mmfs/bin
auditConsumerManager      mmdelnsd                  mmunmount
auth.snap.py              mmdelnsnapshot            mmuserauth
callhome.snap.py          mmddevdiscover            mmvaryonvg
ces.snap.py               mmdf                      mmwatch
cleanupkeystonedb         mmdiag                   mmxargs
core.snap.py              mmdsh                    mmyfields
daemonize                 mmdsm
msgqueueDaemonManager    mmdspmsg                  msgqueue.snap.py
gpfsrunlevel              mmdump                   nfs.snap.py
gpfs.snap                 mmdumpfilocks            object.snap.py
gui.snap.py               mmdumpkthreads           objmetric_count.py
hadoop.snap.py            mmdumpperfdata           perfmon.snap.py
kdump                     mmdynamicmap             perf.snap.py
kdump-3.10.0-693.el7.x86_64 mmeditac1                protect.snap.py
lxtrace
...

# ps -fc mmfsd
UID      PID  PPID  C  STIME TTY      TIME CMD
root     27473 27153  0  Oct09 ?        00:16:10 /usr/lpp/mmfs/bin/mmfsd

# lsmod | grep mmfs
mmfs26                2618329  2
mmfslinux              800204  25 mmfs26
tracedev               48579   3 mmfs26,mmfslinux

# ls /var/mmfs
ccr  ces  etc  gen  mmbackup  mmpmon  mmsysmon  ssl  tmp
```

Scale administrative commands

Commands and binaries are stored in
/usr/lpp/mmfs/bin

All commands intended for customer use have the prefix **mm-**.

- Generally, the form is:

mm{verb}{object}

{verb} is an action like create, add, del~~e~~te, change, list, ...

{object} is class of object of interest, like cluster, node, nsd, disk, file system, ...

Examples:

- **mmlscluster**
- **mmlsconfig**
- New is the Native REST API and its **scalectl** command, an emerging role-based replacement for many **mm-** commands.

```
# export PATH=/usr/lpp/mmfs/bin:$PATH
# mmlscluster
```

GPFS cluster information

=====

```
GPFS cluster name:      slsr.local
GPFS cluster id:        14559237835526968155
GPFS UID domain:        slsr.local
Remote shell command:   /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:        CCR
```

Node	Daemon node name	IP address	Admin node name	Designation
1	a-scale01	10.0.200.11	a-scale01	quorum-manager
2	b-scale01	10.0.200.21	b-scale01	quorum-manager
3	tiebreak	10.0.200.31	tiebreak	quorum
4	a-scale02	10.0.200.12	a-scale02	
5	a-scale03	10.0.200.13	a-scale03	
6	b-scale02	10.0.200.22	b-scale02	
7	b-scale03	10.0.200.23	b-scale03	

Did you know?

- The **mm-** prefix harkens to Scale's origin as the "Multi-Media File System".
- Many internal commands start with the prefix **ts-**, for an earlier Tigershark File System (based on even earlier Shark). Don't use the **ts-** commands unless directed by support!

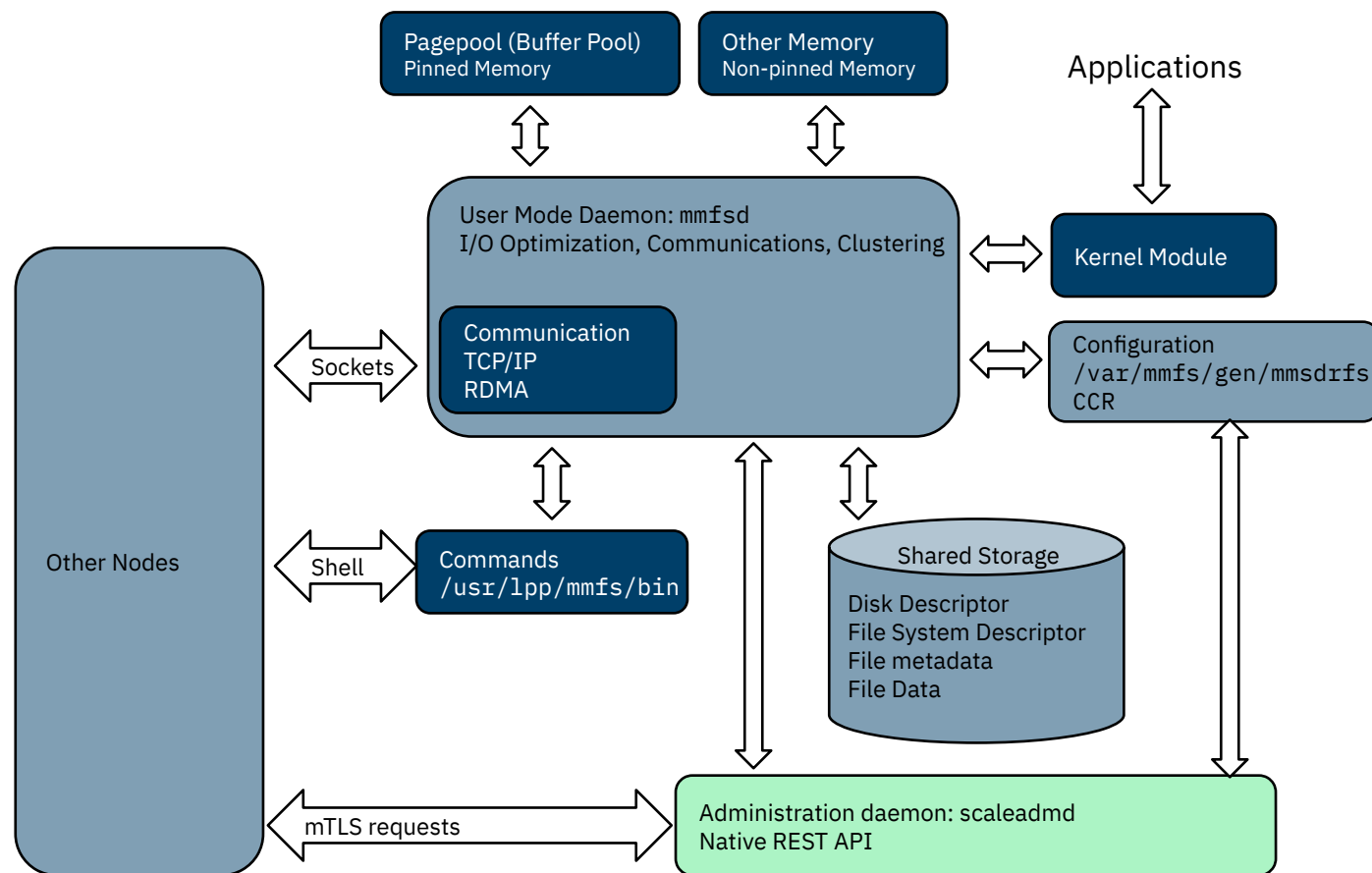
Scale kernel extensions

The Scale kernel extensions link to the operating system “vnode” and “VFS” (virtual file system) internal interfaces.

- Scale appears to the application as just another file system.

I/O Flow

- Applications make file system calls to the OS
- OS presents calls to the Scale kernel extension
- Scale kernel extensions will either satisfy requests using information already available or send a message to the Scale daemon `mmfsd` to complete the request.



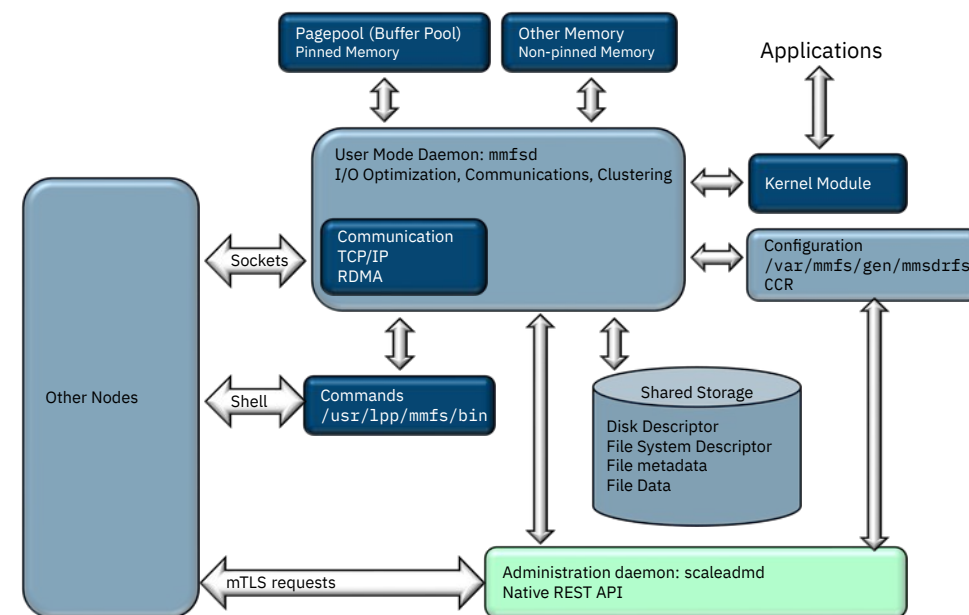
Storage Scale Daemon

The Scale daemon, mmfsd, performs:

- all I/O and buffer management
- reading ahead for sequential reads
- write behind for all writes not specified as synchronous.

All I/O is protected by a “distributed lock manager” to ensure file system consistency

The mmfsd is highly multi-threaded, with some threads dedicated to specific functions.



```
# ps -Lf -C mmfsd | head
UID      PID  PPID  LWP  C  NLWP STIME TTY      TIME  CMD
root    27473 27153 27473 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27491 0   504 Oct09 ?        00:00:01 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27493 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27494 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27495 0   504 Oct09 ?        00:00:21 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27496 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27497 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27498 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
root    27473 27153 27499 0   504 Oct09 ?        00:00:00 /usr/lpp/mmfs/bin/mmfsd
```

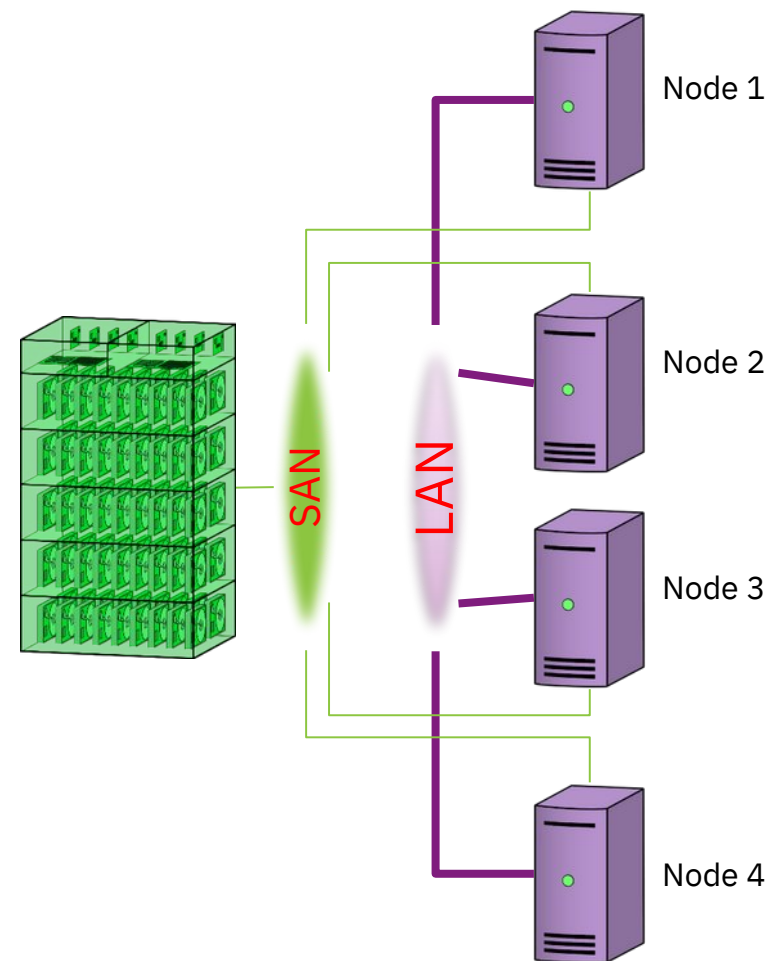
Storage Scale clusters

A Storage Scale **cluster** is a group of Scale systems, or **nodes**, configured into a single administrative grouping:

- All nodes have a common view of the data – one or more **clustered file systems**.
- The nodes are tightly coupled, trusting each others' authentication of users.
- A cluster can have several Storage Scale file systems.
- A cluster may share a file system with an authenticated remote cluster.

Importantly, the cluster ensures all nodes in the cluster have a *consistent* view of the Scale file systems, even when more than one node is actively accessing a file system (or even the same file).

Careful! “Cluster” is a word used in many different ways in different contexts. Here we mean not merely a group of systems, but rather systems configured to work together as a single unit by running Scale.



Cluster manager and the active cluster

Many nodes may be **configured** to be part of the cluster.

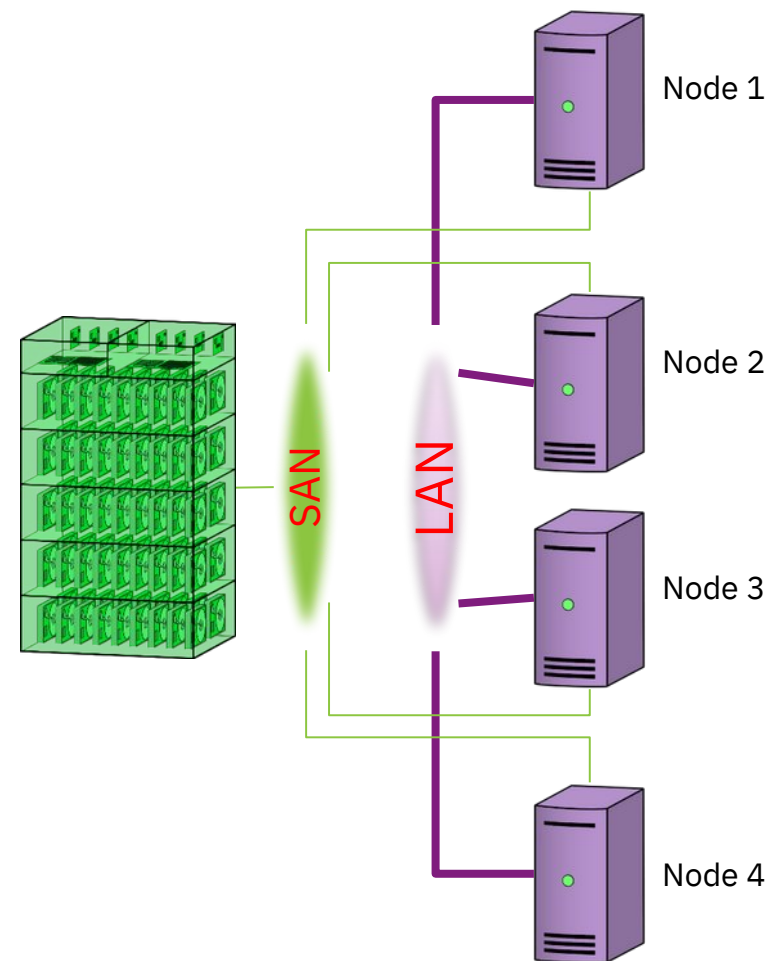
The **active cluster** is the set of nodes currently communicating with each other and sharing resources.

A **cluster manager** keeps track of which nodes are currently part of the active cluster, controls access to managed cluster resources, and maintains the configuration of the cluster.

To avoid the cluster manager from being a single point of failure, this is a **role** that may run on one of several cluster nodes.

Because failures happen, the cluster manager may need to **expel** “dead” nodes from the active cluster and **fence** them from resources like shared disks:

- Perhaps a node has failed
- Perhaps a node's network has failed

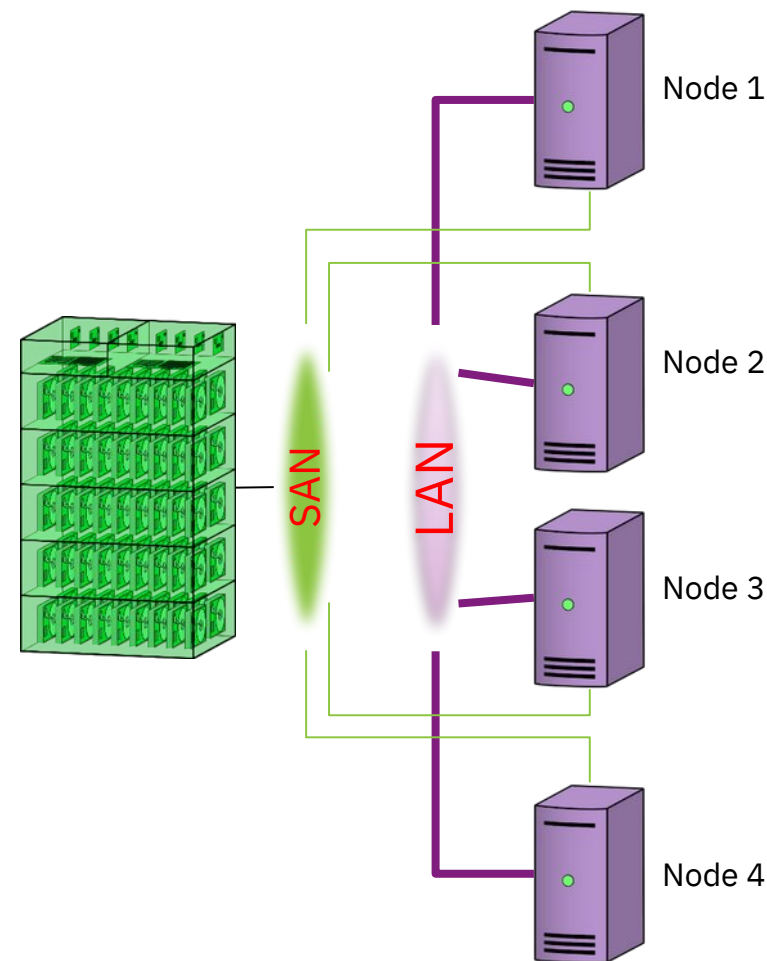


Disk leasing

Disk leasing is the fencing mechanism usually used to coordinate nodes' access to disks.

- A node must hold a valid **disk lease** to shared disk volumes in order to do any I/O on those volumes.
- The cluster manager coordinates issuing **disk leases** for specified durations, as well as handling renewals and active cluster membership.
- Disk leases expire, and nodes are supposed to renew their lease before they expire.
- Lease requests also function as **heartbeats** – a node failing to renew its disk lease will eventually be expelled by the cluster manager as an active cluster member.

SCSI persistent reserve, if enabled and supported on all disks in the cluster, can be used by Scale to fence off disks from non-responsive nodes as soon as they are declared dead. Otherwise wait for `leaseRecoveryTime` seconds (default is 35) to allow for in-flight I/O. After, the cluster may recover.



Guarding against “split brain”

To keep the cluster resources consistent and prevent data loss, *there must be one and only one cluster manager*.

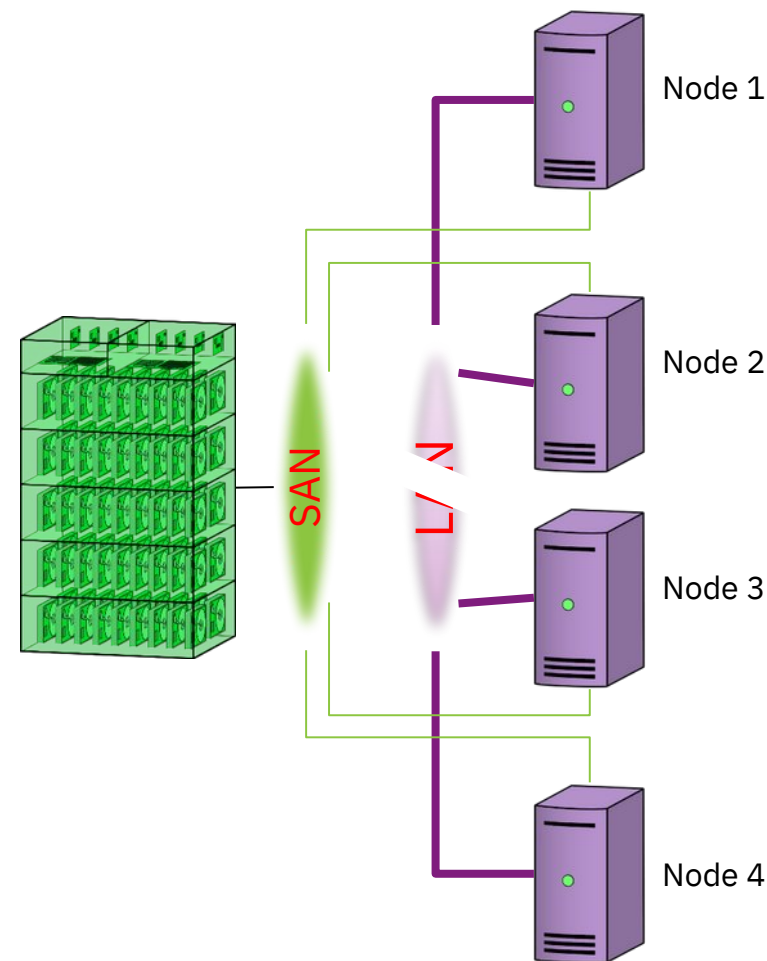
Scenario: Nodes 3 and 4 can't communicate with Nodes 1 and 2.

- Are Nodes 1 and 2 dead, so Nodes 3 and 4 may keep working?
- Or is it that they are alive but can't communicate – so maybe Nodes 3 and 4 should stop?
- If Nodes 1 and 2 were active (with a cluster manager), and Nodes 3 and 4 were active (with their own cluster manager), we would have a **split-brain** condition, leading to *disaster*.



CAP theorem – Brewer 1998,
Proven Gilbert, Lynch 2002

See https://en.wikipedia.org/wiki/CAP_theorem



Scale never has split-brain: quorum

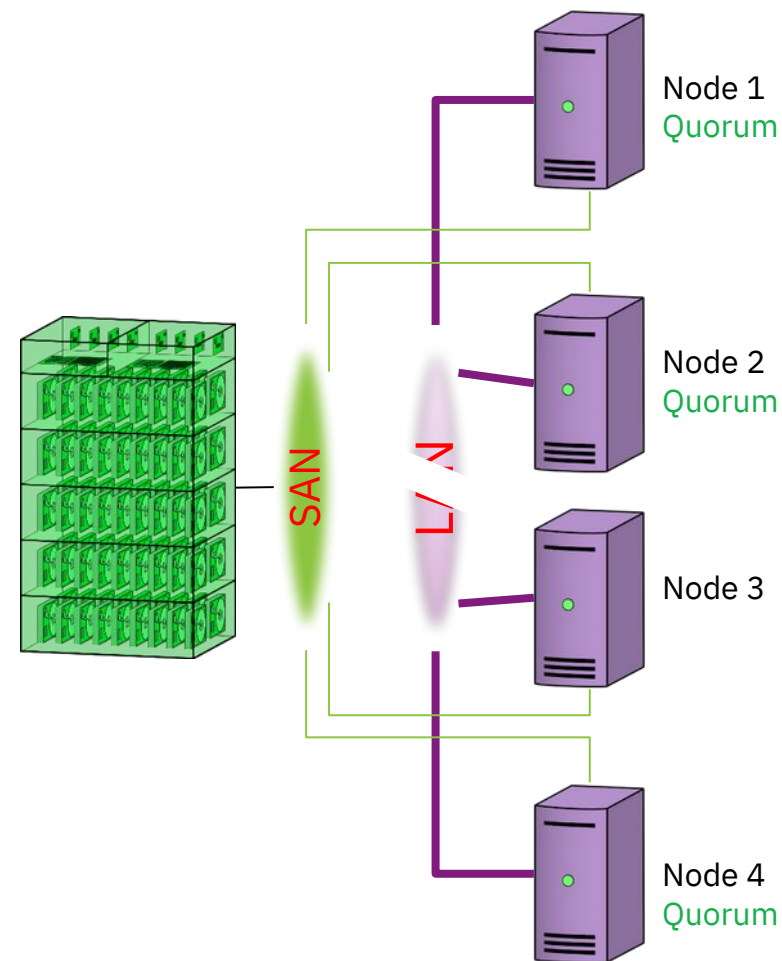
Several nodes are designated as *quorum nodes*, used to elect the cluster manager from amongst themselves.

With **node quorum**, a simple majority of the quorum nodes must be active and communicating to choose a cluster manager.

- Generally, choose a small odd number of quorum nodes, like 3. More quorum nodes lengthens recovery time.
- Quorum nodes also maintain cluster configuration files and settings through the “CCR configuration” mechanism.

Clusters with shared storage accessible to all quorum nodes can use [tiebreaker disk quorum](#) to choose the cluster manager.

- This allows the cluster to be active with as few as one active quorum node.
- The quorum rules are very different; *access to the tiebreaker disks is crucial*.
- Generally, tiebreaker disk quorum is only used with small clusters.



Working with clusters

Create a cluster with:

```
mmcrcluster
```

See what nodes are in a cluster, and their roles, with:

```
mmlscluster
```

Bring up a cluster with:

```
mmstartup -a
```

Check the quorum state with:

```
mmgetstate -aLs
```

See which node is the cluster manager with:

```
mmlsmgr -c
```

Shutdown a cluster with:

```
mmshutdown -a
```

```
# mmlscluster
```

```
GPFS cluster information
```

```
=====
```

```
GPFS cluster name:      slsr.local
GPFS cluster id:        14559237835526968155
GPFS UID domain:        slsr.local
Remote shell command:   /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:        CCR
```

Node	Daemon node name	IP address	Admin node name	Designation
1	a-scale01	10.0.200.11	a-scale01	quorum-manager
2	b-scale01	10.0.200.21	b-scale01	quorum-manager
3	tiebreak	10.0.200.31	tiebreak	quorum
4	a-scale02	10.0.200.12	a-scale02	
5	a-scale03	10.0.200.13	a-scale03	
6	b-scale02	10.0.200.22	b-scale02	
7	b-scale03	10.0.200.23	b-scale03	

```
# mmlsmgr -c
```

```
Cluster manager node: 10.0.200.11 (a-scale01)
```

Working with nodes

- Add a node to a cluster with:

```
mmaddnode -N NODENAME
```

- Change a node to a quorum node with:

```
mmchnode --quorum -N NODENAME
```

Shut down Scale on a node with:

```
mmshutdown -N NODENAME
```

Remove a node with:

```
mmdelnode -N NODENAME
```

Scale allows us to add and remove nodes from a running cluster.

```
# mmchnode --quorum -N a-scale03
Tue Oct 23 21:07:58 EDT 2018: mmchnode: Processing node a-scale03
mmchnode: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

```
# mmgetstate -aLs
```

Node number	Node name	Quorum	Nodes up	Total nodes	GPFS state	Remarks
1	a-scale01	3	4	7	active	quorum node
2	b-scale01	3	4	7	active	quorum node
3	tiebreak	3	4	7	active	quorum node
4	a-scale02	3	4	7	active	
5	a-scale03	3	4	7	active	quorum node
6	b-scale02	3	4	7	active	
7	b-scale03	3	4	7	active	

Summary information

```
-----
Number of nodes defined in the cluster:      7
Number of local nodes active in the cluster: 7
Number of remote nodes joined in this cluster: 0
Number of quorum nodes defined in the cluster: 4
Number of quorum nodes active in the cluster: 4
Quorum = 3, Quorum achieved
```

```
# mmshutdown -N a-scale03
Tue Oct 23 21:13:34 EDT 2018: mmshutdown: Starting force unmount of GPFS file systems
Tue Oct 23 21:13:39 EDT 2018: mmshutdown: Shutting down GPFS daemons
Tue Oct 23 21:13:47 EDT 2018: mmshutdown: Finished
```

Working with node classes

- *Node classes* allow logical groupings of nodes to be treated as a unit.
- Most commands that use the `-N` flag to specify nodes will also accept a node class.
 - Some node classes are automatically created: `all`, `cesNodes`, `nsdNodes`, `quorumNodes`, etc.
 - See the complete list of automatic node classes:

```
mm1snodeclass --system
```

- Create a node class with `mmcrnodeclass`
- Modify a node class with `mmchnodeclass`
- Delete a node class with `mmdelnodeclass`
- List a node class, or nodes in a class, with `mm1snodeclass`

```
# mmcrnodeclass hub -N hub1,hub2
mmcrnodeclass: Propagating the cluster configuration data to all
affected nodes. This is an asynchronous process.
```

```
# mm1snodeclass hub
Node Class Name      Members
-----
hub                  hub2,hub1
```

```
# mmgetstate -N hub
```

Node number	Node name	GPFS state
2	hub2-dmz	active
3	hub1-dmz	active

```
# mm1snodeclass
Node Class Name      Members
-----
GUI_MGMT_SERVERS    jumpserver
GUI_SERVERS         hub2,hub1,jumpserver
hub                  hub2,hub1
```

Administrative nodes and mmdsh

Some “administrative commands” like `mmstartup` use `ssh` to issue helper commands on other cluster nodes.

- We will call these **administrative nodes**.
- Administrative nodes require passwordless `ssh` access as `root` to every other node in the cluster.
 - Administrative nodes are simply the nodes from which you will issue administrative commands, as well as GUI nodes, protocol nodes, HDFS NameNodes, perhaps a few others.
- The `sudo-wrappers` feature can remove this requirement: `root` needs passwordless `ssh` access to a *designated user* listed in the `Sudoers` file. (A few features don’t work with `sudo-wrappers`.)

The `mmdsh` command may be used to issue the same command to multiple cluster nodes.

- Node classes can be used here.

```
# mmdsh -N cesNodes uptime
hub1: 16:27:40 up 139 days, 14 min, 2 users, load average: 0.43, 0.42, 0.52
hub2: 16:27:40 up 139 days, 1 min, 1 user, load average: 0.33, 0.17, 0.13

# mmdsh -N all date
hub2: Wed Sep 4 16:28:52 EDT 2019
hub1: Wed Sep 4 16:28:52 EDT 2019
jumpserver: Wed Sep 4 16:28:52 EDT 2019
```

A word about licensing

Each node needs a license type assigned to it:

- **server** – Permits all Scale operations
- **client** – May not be a quorum node, manager node, AFM gateway, protocol node, TCT gateway, NSD server
 - Additionally, `client` nodes are not permitted to run servers to export data, stored in Scale, via FTP, HTTP(s), NFS, SMB, etc. unmodified, but no mechanism enforces this.
- **fpo** – Let's ignore this

With an ancient “socket” or “PVU” license, server licenses were much more expensive than client licenses.

- Nothing enforces not allocating too many licenses besides fear of someone finding out – honor system.

With a modern capacity licenses, these are all free (once you've paid for capacity).

```
# mmchlicense server --accept -N hub1,hub2
```

```
# mmlslicense -L
```

Node name	Required license	Designated license
hub1	server	server
hub2	server	server
jumpserver	client	client

```
Summary information
```

```
-----
Number of nodes defined in the cluster:          3
Number of nodes with server license designation: 2
Number of nodes with FPO license designation:    0
Number of nodes with client license designation: 1
Number of nodes still requiring server license designation: 0
Number of nodes still requiring client license designation: 0
This node runs IBM Spectrum Scale Data Management Edition
```

Some additional node roles in the cluster layer

NSD server

- Makes storage available over the LAN to nodes
- As we'll see, a node is an NSD server simply if it is assigned NSDs to serve.

Manager

- Runs token managers, and a candidate for the file system manager

Graphical User Interface (GUI)

- Visually monitor performance
- GUI control over many cluster and file system functions
- RESTful API for managing the cluster

Performance monitoring

- Monitor performance of nodes, storage, network through CLI or GUI

SNMP manager

- Collect alerts and generate traps

AFM gateway

- Used to move data when using AFM

Cluster Export Services ("protocol node")

- Exports data from file systems through SMB, NFS, Swift

Cloud gateway

- Transfers data between file systems and underlying cloud storage systems

```
# mmlcluster
GPFS cluster information
=====
GPFS cluster name:      Plato.
GPFS cluster id:       8052019643637662953
GPFS UID domain:      Plato.
Remote shell command:  /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:      CCR
```

Node	Daemon node name	IP address	Admin node name	Designation
1	plato01-dmz	192.168.62.30	plato01	quorum-manager-
perfmon				
2	plato02-dmz	192.168.62.39	plato02	quorum-manager-
perfmon				
12	newton05-dmz	192.168.62.44	newton05	perfmon
15	newton04-dmz	192.168.62.43	newton04	perfmon
16	gauss01-dmz	192.168.62.99	gauss01	perfmon
17	gauss02-dmz	192.168.62.98	gauss02	perfmon
19	plato08-dmz	192.168.62.42	plato08	perfmon
22	gauss03-dmz	192.168.62.97	gauss03	perfmon
23	gauss04-dmz	192.168.62.96	gauss04	perfmon
24	plato04-dmz	192.168.62.37	plato04	perfmon
25	plato05-dmz	192.168.62.36	plato05	perfmon
26	plato03-dmz	192.168.62.38	plato03	quorum-manager-
perfmon				
30	ashe	192.168.62.14	ashe	
31	billie	192.168.62.15	billie	

```
# mmchnode --gateway --perfmon --snmp-agent -N ashe,billie
```

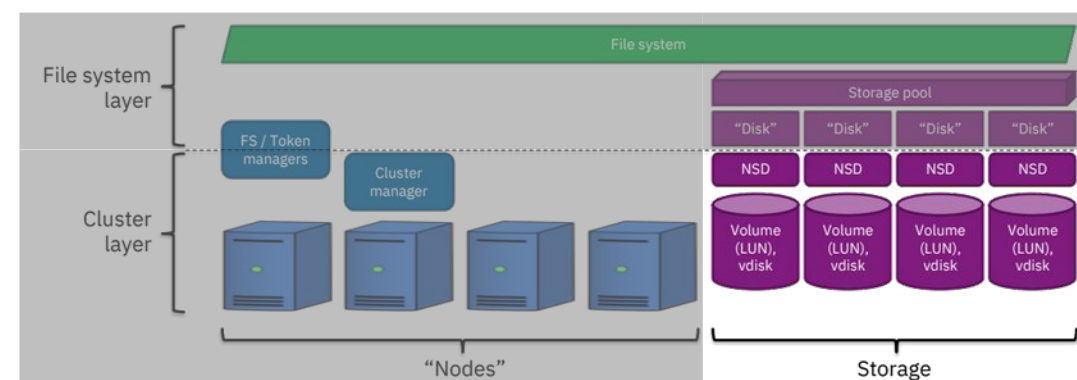
Pause: Got questions about chapter 1?



Chapter 2

Clustering and storage

... or, gotta put that data somewhere!



Clustered file systems

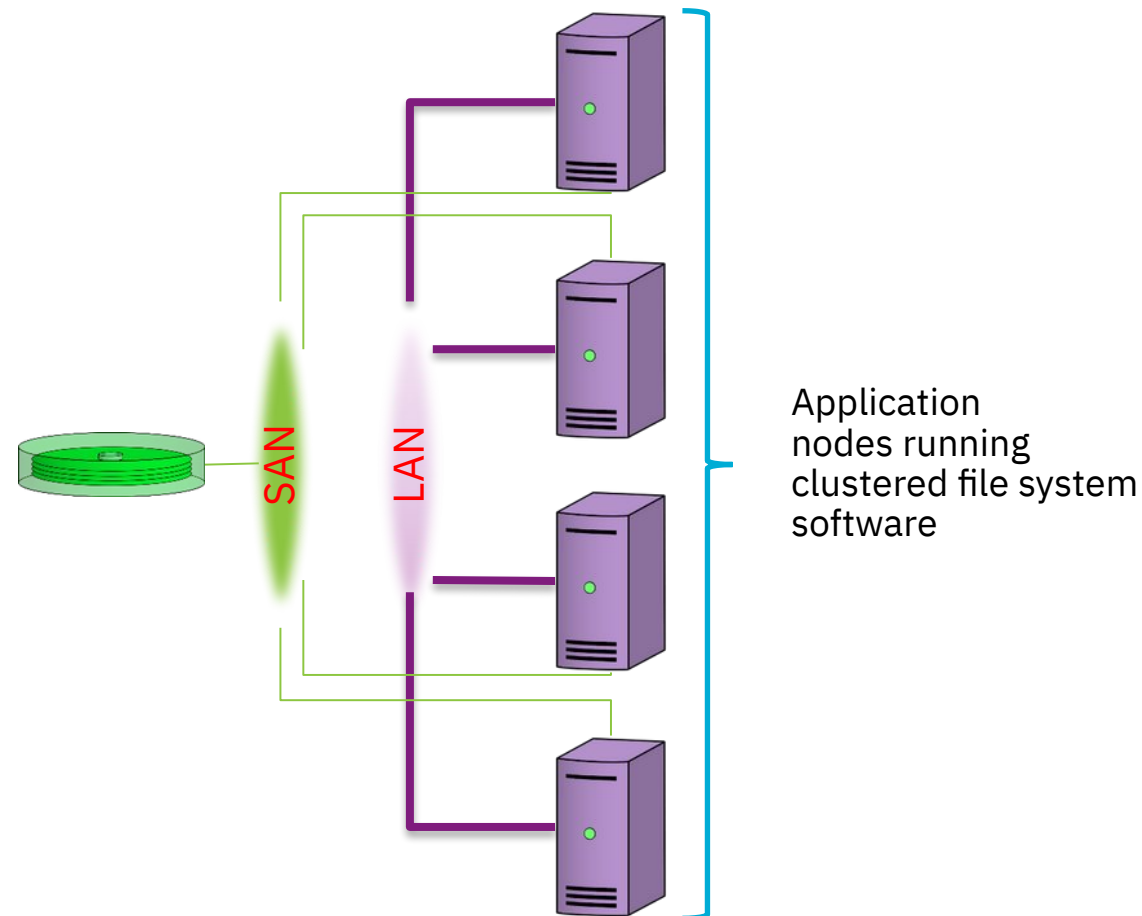
A **clustered file system** may be mounted on multiple file system nodes simultaneously, sharing access to the same disk or volume through the **SAN**.

- Software implementing the clustered file system must be running on each node.
- A logical volume manager (LVM) might allow multiple disk volumes to be federated into a single logical volume.
- This clustering software coordinates access to the actual (logical) disk volume to maintain disk integrity.
- This coordination is done through messages on the LAN.
- There are no “clients” and “servers” – everything participates.
- All nodes in the cluster are able to be actively accessing files (even multiple nodes accessing the same file).

Some common clustered file systems:

- SanFS
- Oracle OCFS2
- Redhat GFS2
- Veritas

Typically, the clustered file system software is not bundled as part of the operating system, so it needs to be installed on each node.



Storage Scale: a parallel clustered file system

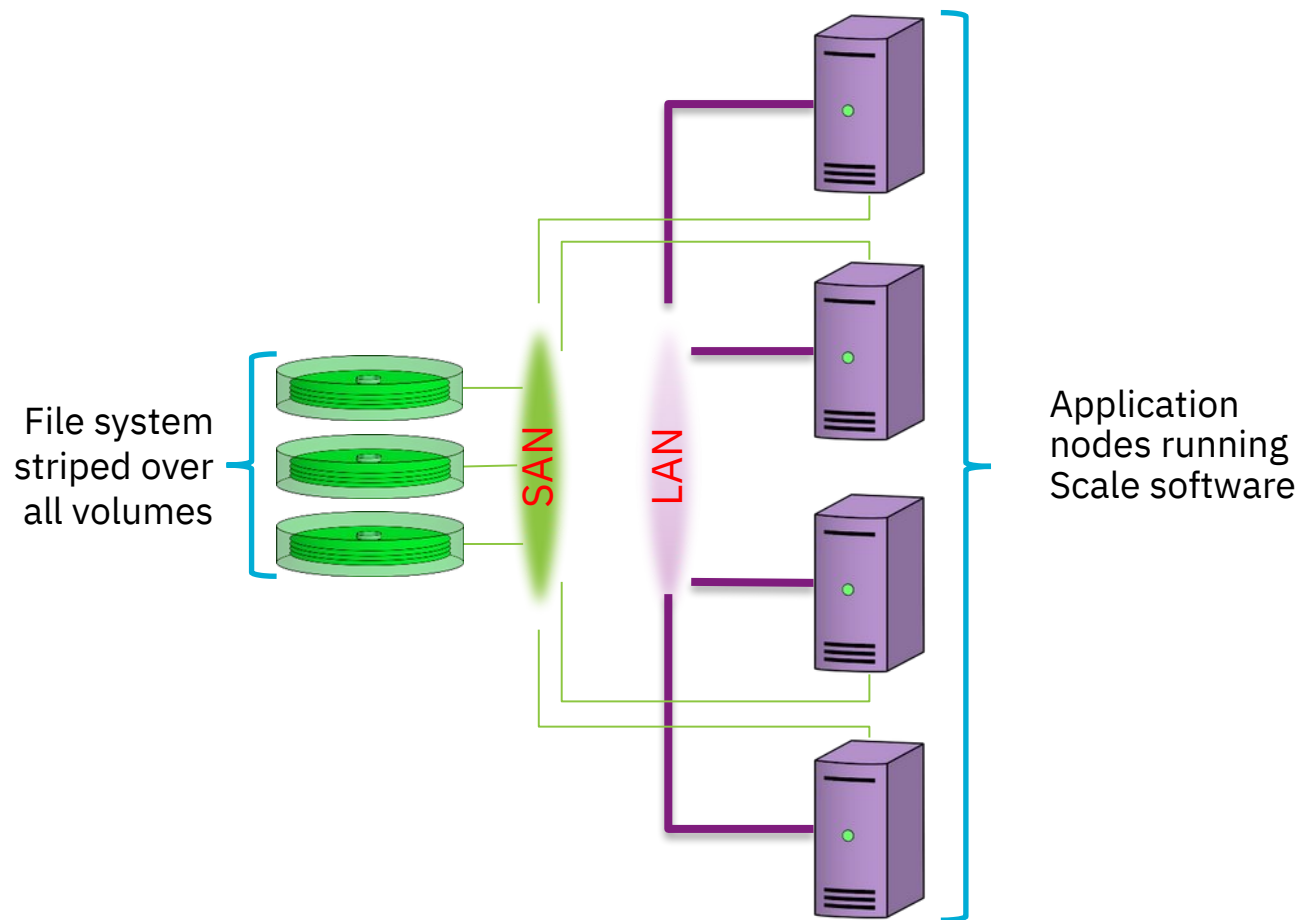
Scale is fundamentally a clustered file system:

- All nodes in the cluster share in the responsibility of accessing the disk volumes to manage the file system and the data stored in it.
 - There are no “client” and “server” roles, as with distributed file systems like NFS and SMB.

Scale is a **parallel** file system:

- The file system is striped over potentially many disk volumes.
- Even individual files (once large enough) are striped over multiple volumes.
- Uniquely, Scale even stripes file system metadata over many volumes.

All disk volumes controlled by a Scale cluster are called **NSDs**.



Distributing Scale over the LAN with NSD servers

Rather than accessing disk volumes through the SAN, Scale can access them through the normal LAN network.

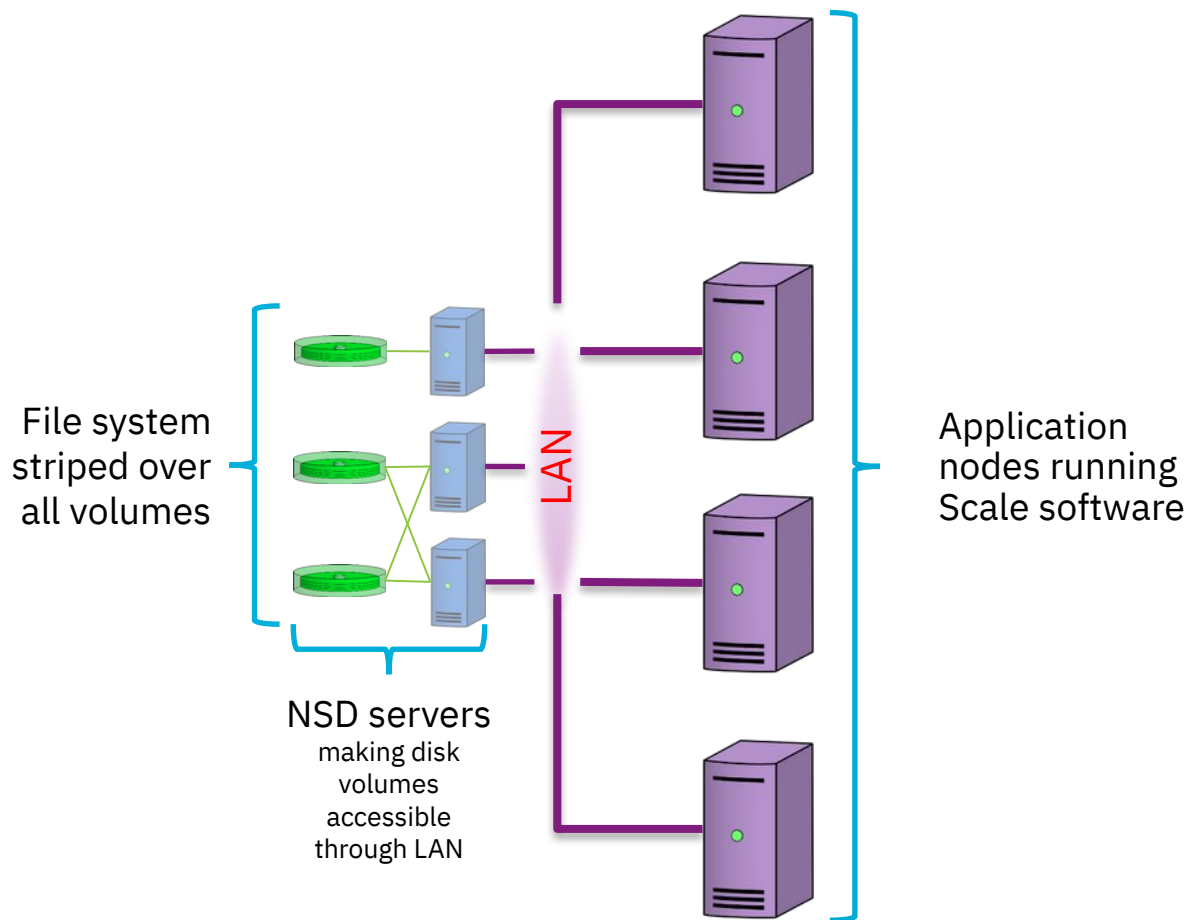
One obvious approach would be to use iSCSI, NVMe-of, or FCoE targets for the volumes. But there is a better way...

Scale also provides its own protocol for making disk volumes accessible through the LAN – the NSD protocol.

- Any Scale node is potentially an NSD server, since this capability is provided through mmfsd.

Every Scale node needs at least one path to every NSD, whether through the SAN, or through an NSD server.

NSD servers are *not* file servers. (Even if it is called a Scale System.)



Aside: What kind of storage can be used for an NSD?

Scale can use any persistent storage presented to the operating system as a block device under `/dev`.

- SAS and SATA disks (internal or directly attached)
- Fibre-channel (directly attached or through SAN)
- iSCSI, SRP, iSER
- Storage Scale RAID vdisks
- TRIM is supported with SSD/NVMe volumes

Multipathing may be used – set `no_path_retry`. Use the `/dev/dm-*` name for the device, not the underlying devices.

- The multipath name may change on reboot – it only needs to be correct when the NSD is first formatted.

Thin-provisioned NSDs containing metadata or replicated data are supported via RPQ using `thinDiskType` NSD clauses. The `thinDiskType` clause is also used with NVMe disks to enable TRIM. (Only Linux servers may be attached to these `thinDiskType` disks, but they may serve as NSD servers.)



With Linux multipath, Scale expects these parameter settings:

- `features "0"`
- `failback immediate`
- `no_path_retry fail`

Working with NSDs

To add NSDs to a running Scale cluster:

```
mmcrnsd -F StanzaFile
```

The StanzaFile lists names the NSD, associates it with a block device, and lists the servers that can access it.

- If no nsd= clause is listed, mmcrnsd will edit/create one into the file.
- The servers= clause turns these nodes into NSD servers. The first one listed will be “primary” for that NSD. This line can be omitted if the volume is directly connected to every node.
- The device= clause lists the Unix block device associated with the volume. It must be correct on the primary NSD server when mmcrnsd is run (or on the node where mmcrnsd is run, if there are no NSD servers listed).
 - After NSDs are defined, Scale can find them even if the device mapping has changed.

To remove NSDs from a running cluster:

```
mmdelnsd nsdName
```

- You can list multiple NSDs, separated with semicolons, on the command line.
- Only NSDs that are not part of a file system may be deleted.

On an ESS, or with an ECE cluster, NSDs and “disks” are managed using the mmvdisk command, without stanza files. But they still support the underlying attributes seen here.

%nsd:

```
nsd=a1
device=/dev/dm-16
servers=scale01,scale02
failureGroup=1
pool=system
usage=dataAndMetadata
```

The device path /dev/dm-16 must exist on scale01 and be the desired device.
The path name may or may not be same on scale02!

%nsd:

```
nsd=b1
device=/dev/dm-12
servers=scale02,scale03
failureGroup=2
pool=system
usage=dataAndMetadata
```

Pause: Got questions about chapter 2?

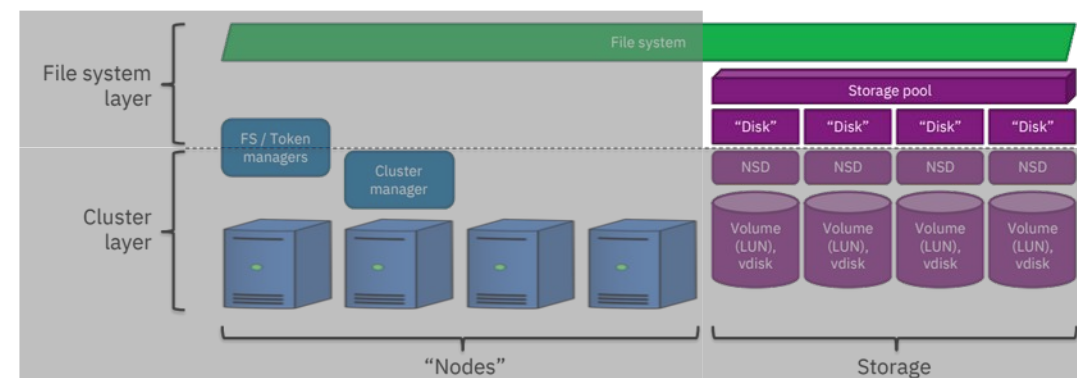


Chapter 3

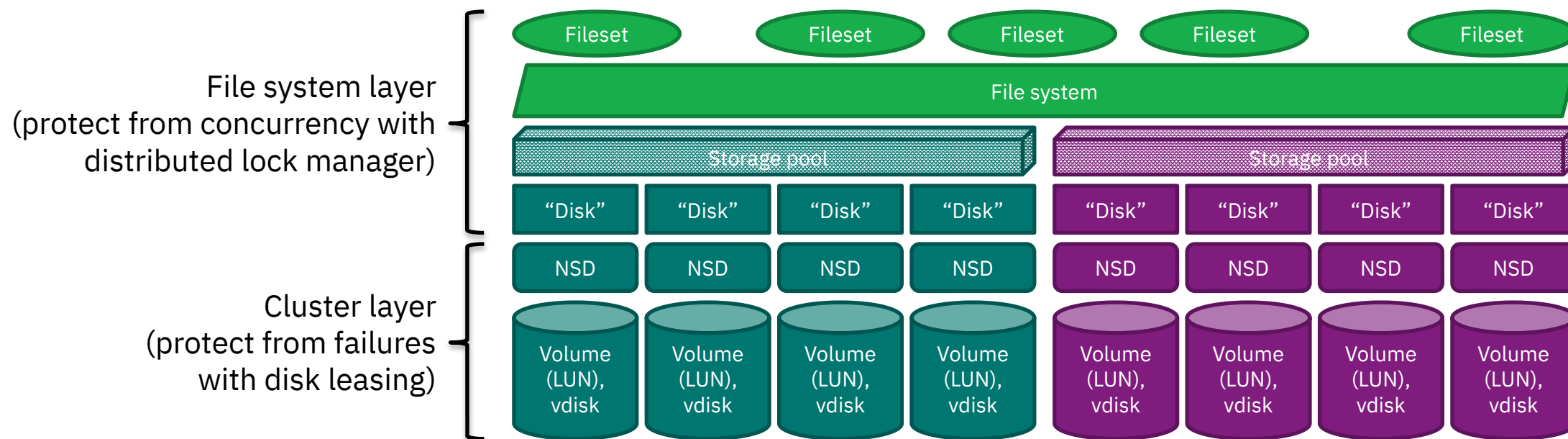
Storage Scale storage:

File system layer

... or, what does Storage Scale look like?

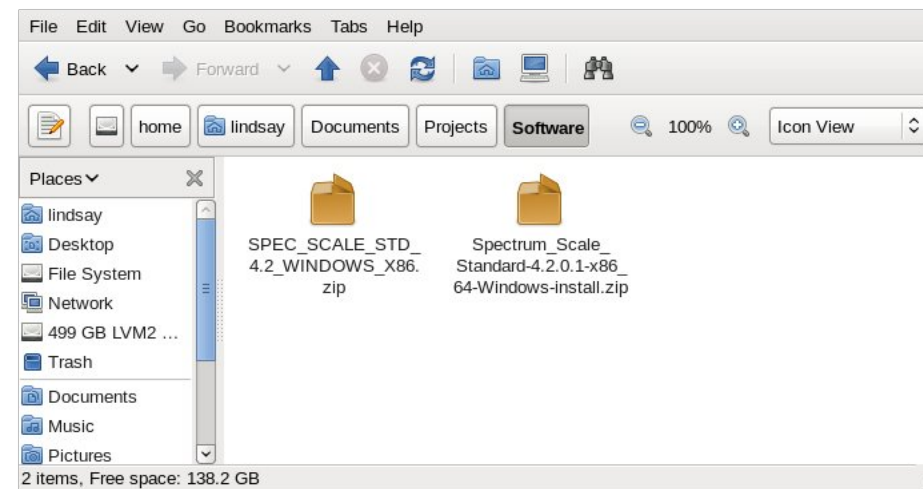


A conceptual overview of Scale storage



Storage Scale – a POSIX file system

- POSIX file systems are organized as *directories* (folders) containing *files* (documents).
- POSIX defines programming APIs for manipulating files, such as:
 - mount, umount the file system
 - open, close file descriptor
 - write, read to/from file descriptor
 - mkdir, rmdir, creat, unlink, link, symlink
 - fcntl (byte range locks, etc.)
 - stat, utimes, chmod, chown, chgrp
- Path names are case sensitive; components are separated with forward slash (/).
- Essentially, this is the API that evolved for Unix and adopted by Linux.



```
$ cd /home/lindsay/Documents/Projects/Software
$ ls
SPEC_SCALE_STD_4.2_WINDOWS_X86.zip
Spectrum_Scale_Standard-4.2.0.1-x86_64-Windows-install.zip
$
```

Some basic file system concepts – data blocks

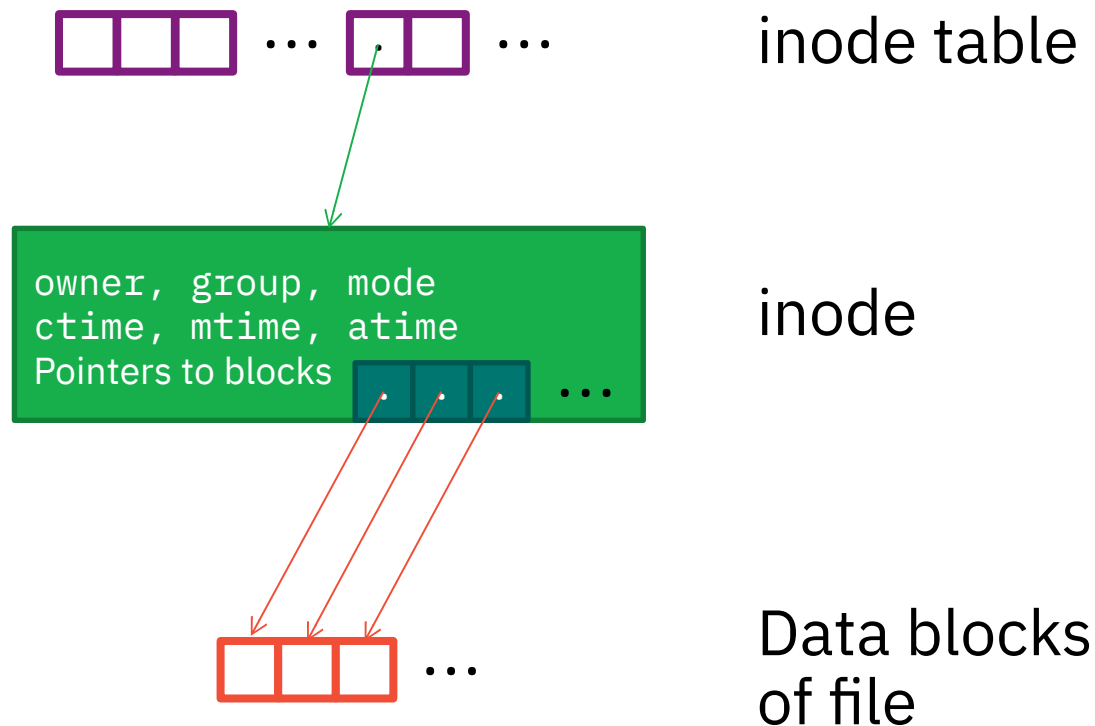
The data content of a file is stored as logical sequence of file system **blocks**.

- Each block is a fixed **block size**
- The last block of a file might be a fragment (not a full block).
 - A block could contain the fragments from the ends of several different files – `mmdefragfs` can do this to reduce *fragmentation*.

The actual blocks usually do not reside sequentially on the disk volume.

- Scale will usually put sequential blocks of a file on different disk volumes, so they can be processed in parallel.

Ideally file system blocks are aligned with the storage system's blocks.



Some basic file system concepts – metadata

File systems also contain *metadata*, i.e., “data about the data”.

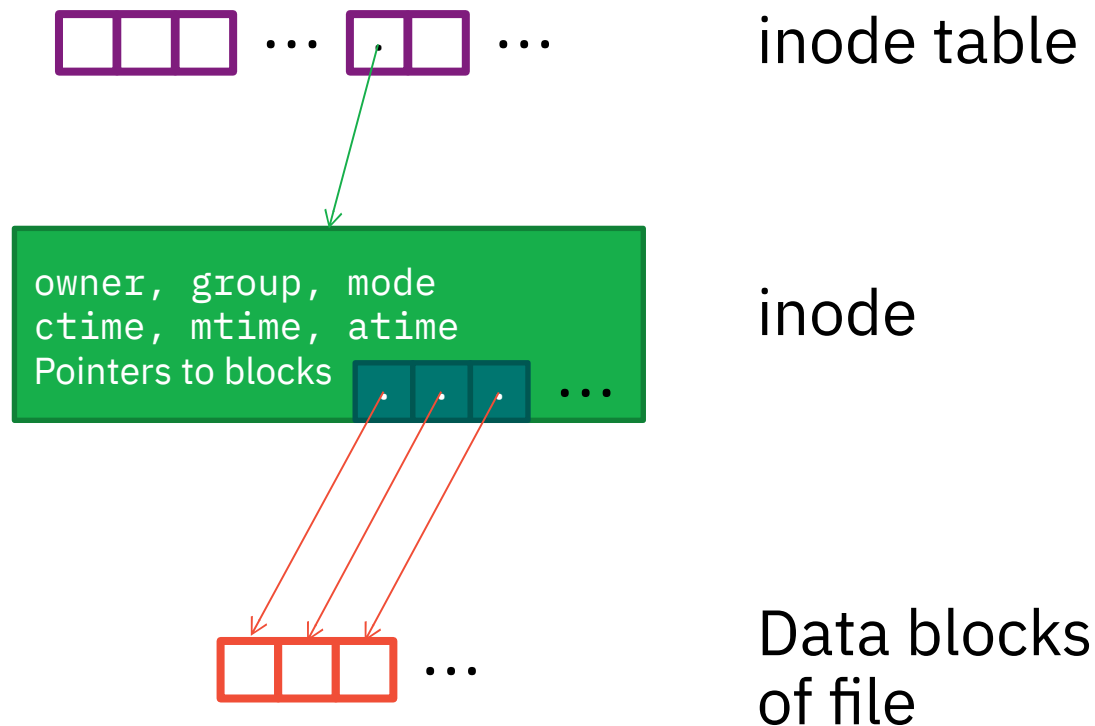
There is an *inode* for each file or directory, providing:

- Locations of the data blocks for the file – large files may need indirect inodes to address every block.
- Attributes about the file, like “time last accessed” or “owner of the file”

The *inode table* records where each inode is located, indexed by number.

Directories are special files listing the names and inode numbers of files under the folder.

Metadata updates (and small writes if HAWC is enabled) are first committed to the **file system log** or **journal**.



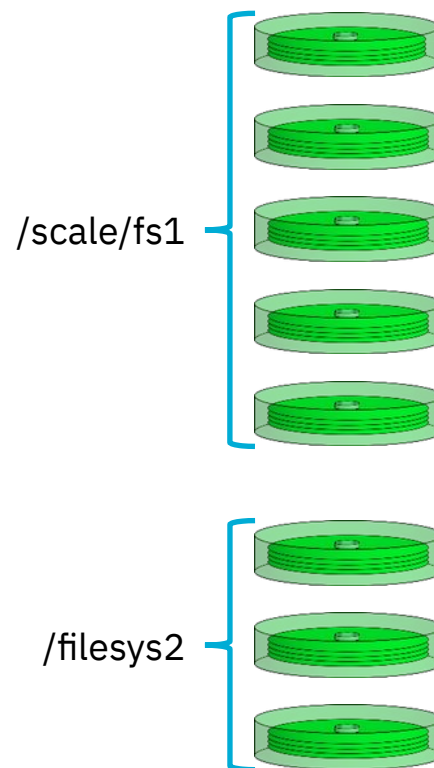
Scale file systems

Within a cluster, up to 256 Scale file systems may be concurrently mounted (including remote-cluster mounted file systems).

A file system (sometimes called a *stripe group*) is striped over one or more NSDs.

File systems have a short name (or “device name”), e.g., `fs1`, and a mount point, e.g., `/scale/fs1`

- In the past, an entry under `/dev` was created for the file system device name, e.g., `/dev/fs1`. This is no longer done, although not all documentation reflects this.



Creating a file system

To create a file system, use:

```
mmcrfs FSDEV -F StanzaFile -B BlockSize ...
```

- On a Scale System or ECE cluster, use `mmvdisk` instead. But these attributes are still relevant.
- Many flags allow tuning other attributes besides `BlockSize`. Some can be changed later; others cannot be changed later! See the table for important examples.

-B	BlockSize	Block size of the file system, 4M or 8M are usually good.
-j	MapType	Block allocation type, set as scatter
-n	NumNodes	Best estimate for number of nodes that will mount the file system (including remote) – allow for growth.
-M	MaxMetadataRep	Maximum allowed replication for metadata; generally, set to 2.
-R	MaxDataRep	Maximum allowed replication for data. If you will ever use replication, set to 2.

%nsd:

```
nsd=a1
device=/dev/dm-16
servers=scale01,scale02
failureGroup=1
pool=system
usage=dataAndMetadata
```

%nsd:

```
nsd=b1
device=/dev/dm-12
servers=scale02,scale03
failureGroup=2
pool=system
usage=dataAndMetadata
```

To use a file system with a block size larger than 4M, you must set the `maxblocksize` configuration parameter to 8M or 16M. (Both Scale System and the install toolkit will do this automatically for their own clusters.)

More on the stanza files

The stanza files are used as input to the commands and are not needed after.

- But they are good documentation of your file system!
- It is possible to have a more compact representation of a single line for each stanza.

The `mmcrfs` (and `mmadddisk`) commands ignore the `device=` and `servers=` clauses – those are only used by `mmcrnsd` and `mmchnsd`.

The `usage=` clause may be `dataOnly`, `metadataOnly`, or `dataAndMetadata`, depending on what will be allowed on the disk.

The `pool=` clause sets the storage pool (for ILM).

The `failureGroup=` clause is used along with replication.

```
%nsd:
  nsd=a1
  device=/dev/dm-16
  servers=scale01,scale02
  failureGroup=1
  pool=system
  usage=dataAndMetadata
```

```
%nsd:
  nsd=b1
  device=/dev/dm-12
  servers=scale02,scale03
  failureGroup=2
  pool=system
  usage=dataAndMetadata
```

Working with file systems

Mount all file systems everywhere with:

```
mmm mount all -a
```

Unmount file systems on current system with:

```
mmu mount all
```

To see where file systems are mounted:

```
mm ls mount all -L
```

```
mm ls mount all --report all
```

Restripe and rebalance a file system with:

```
mm restripefs FSDEV -b
```

```
# mmlsmount all --report all
```

File system fs1 is mounted on 6 nodes:

```
10.0.200.11    a-scale01
10.0.200.22    b-scale02
10.0.200.23    b-scale03
10.0.200.31    tiebreak
10.0.200.12    a-scale02
10.0.200.21    b-scale01
```

File system fs1 is not mounted on 1 nodes:

```
10.0.200.13    a-scale03
```

```
# mmrestripefs fs1 -b
```

Scanning file system metadata, phase 1 ...

```
100 % complete on Thu Oct 25 14:03:35 2018
```

Scan completed successfully.

Scanning file system metadata, phase 2 ...

```
100 % complete on Thu Oct 25 14:03:35 2018
```

Scan completed successfully.

Scanning file system metadata, phase 3 ...

```
100 % complete on Thu Oct 25 14:03:35 2018
```

Scan completed successfully.

Scanning file system metadata, phase 4 ...

```
100 % complete on Thu Oct 25 14:03:35 2018
```

Scan completed successfully.

Scanning user file metadata ...

```
100.00 % complete on Thu Oct 25 14:03:36 2018 ( 92160
```

inodes with total 3020 MB data processed)

Scan completed successfully.

Working with disks in a file system

To add one or more disks to a file system (which may be mounted), use:

```
mmadddisk FSDEV -F StanzaFile
```

- Additional options control whether data should be restriped over the new disks or wait for a more opportune time.

To remove one or more disks from a file system (which may be mounted), use:

```
mmdeldisk FSDEV "DiskName[;DiskName...]"
```

- Data on these disks will be copied to other disks, before actually removing the disk from the file system.
- Additional options control whether data on remain disks should be restriped or wait for a more opportune time.

Scale System and ECE provide equivalent `mmvdisk` commands to add or remove storage vdisks.

```
# mmadddisk fs1 -F newdisks-stanza.txt
The following disks of fs1 will be formatted on node a-scale01:
  aspare01: size 300 MB
Extending Allocation Map
Checking Allocation Map for storage pool system
Completed adding disks to file system fs1.
mmadddisk: Propagating the cluster configuration data to all
  affected nodes. This is an asynchronous process.

# mmdeldisk fs1 aspare01
Deleting disks ...
Scanning file system metadata, phase 1 ...
  100 % complete on Wed Oct 24 15:42:48 2018
Scan completed successfully.
Scanning file system metadata, phase 2 ...
  100 % complete on Wed Oct 24 15:42:48 2018
Scan completed successfully.
Scanning file system metadata, phase 3 ...
  100 % complete on Wed Oct 24 15:42:48 2018
Scan completed successfully.
Scanning file system metadata, phase 4 ...
  100 % complete on Wed Oct 24 15:42:48 2018
Scan completed successfully.
Scanning user file metadata ...
  100.00 % complete on Wed Oct 24 15:42:49 2018 (      92160 inodes with
total      3020 MB data processed)
Scan completed successfully.
Checking Allocation Map for storage pool system
tsdeidisk completed.
mmdeldisk: Propagating the cluster configuration data to all
  affected nodes. This is an asynchronous process.
```

Changing the state of a disk

To list the disks in a file system and see their state, use:

```
mmlsdisk FSDEV
```

To change the state of disks, use mmchdisk:

```
mmchdisk FSDEV OP -d "DiskDesc[;DiskDesc...]"
```

Where *OP* is one of:

suspend – Stop allocating space for new data on this disk, but it can still be read or updated.

resume – Start using a disk fully again, if it had been suspended. This does not change stopped disks.

stop – Stop the disk from being used. A disk may be automatically stopped if a failure is detected. Stopped disks remain stopped until explicitly started.

start – Start a stopped disk. (If more than one disk is down, they must all be simultaneously restarted using the **-a** flag.) This will ensure file system consistency in the face of replication.

change – Change an attribute of a disk (its disk usage or failure group). This may not change the storage pool.

```
# mmlsdisk fs1
disk      driver  sector  failure holds  holds  status  availability  storage
name      type    size    group metadata data      pool
-----
a01a      nsd       512      1 Yes    Yes    ready    up    system
b01a      nsd       512      2 Yes    Yes    ready    up    system
a02a      nsd       512      1 Yes    Yes    ready    up    system
b02a      nsd       512      2 Yes    Yes    ready    up    system
a03a      nsd       512      1 Yes    Yes    ready    down  system
b03a      nsd       512      2 Yes    Yes    ready    up    system
a01b      nsd       512      1 Yes    Yes    ready    up    system
b01b      nsd       512      2 Yes    Yes    ready    up    system
a02b      nsd       512      1 Yes    Yes    ready    up    system
b02b      nsd       512      2 Yes    Yes    ready    up    system
a03b      nsd       512      1 Yes    Yes    ready    down  system
b03b      nsd       512      2 Yes    Yes    ready    up    system
a01c      nsd       512      1 Yes    Yes    ready    up    system
b01c      nsd       512      2 Yes    Yes    ready    up    system
a02c      nsd       512      1 Yes    Yes    ready    up    system
b02c      nsd       512      2 Yes    Yes    ready    up    system
a03c      nsd       512      1 Yes    Yes    ready    down  system
b03c      nsd       512      2 Yes    Yes    ready    up    system
tiebreak  nsd       512      3 No     No     ready    up    system

# mmchdisk fs1 start -a
mmsddiscover: Attempting to rediscover the disks. This may take a while ...
mmsddiscover: Finished.
b-scale03: Rediscovered nsd server access to b03a.
...
Scanning user file metadata ...
100.00 % complete on Thu Oct 25 14:13:03 2018 ( 92160 inodes with total 3044 MB
data processed)
```

Blocks and subblocks

A file system block the largest contiguous single I/O size to the disk (so optimize with underlying storage system). The smallest I/O size is to a single **sector** (usually 512 or 4096 bytes).

The smallest allocatable unit of storage is the **subblock**. The number of subblocks in a block depends on block size (in the past, it was always 32).

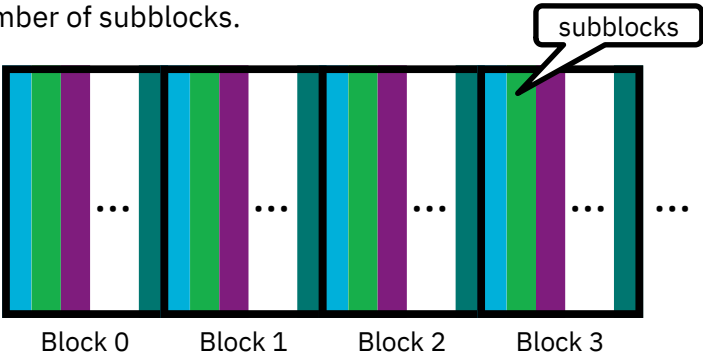
Ends of files are usually fragments, consuming several subblocks. These can be combined into a single block (thus containing fragments of multiple files), using `mmdefragfs`.

Large record I/O performance requires large blocks. Small record I/O is usually not dependent on block size (low latency matters more). A block is also the smallest granularity of the distributed lock manager for byte range locks.

The deprecated `--metadata-block-size` file system parameter, when smaller than data block size, will determine the number of subblocks per block for both data and metadata.

Block size	Subblocks per full block	Calculated subblock size*
64KiB, 128KiB, 256KiB	32	2KiB, 4KiB, 8KiB
512KiB	64	8KiB
1MiB	128	8KiB
2MiB	256	8KiB
4MiB	512	8KiB
8MiB	512	16KiB
16MiB	1024	16KiB
32MiB	2048	16KiB

* Metadata block sizes can be set differently than data block sizes, but that may change the calculate subblock size and number of subblocks.



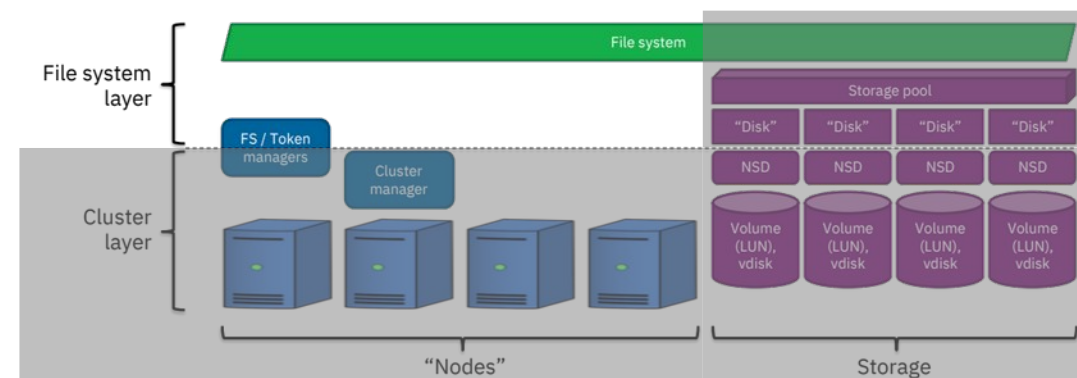
Pause: Got questions about chapter 3?



Chapter 4

Storage Scale nodes: the file system layer

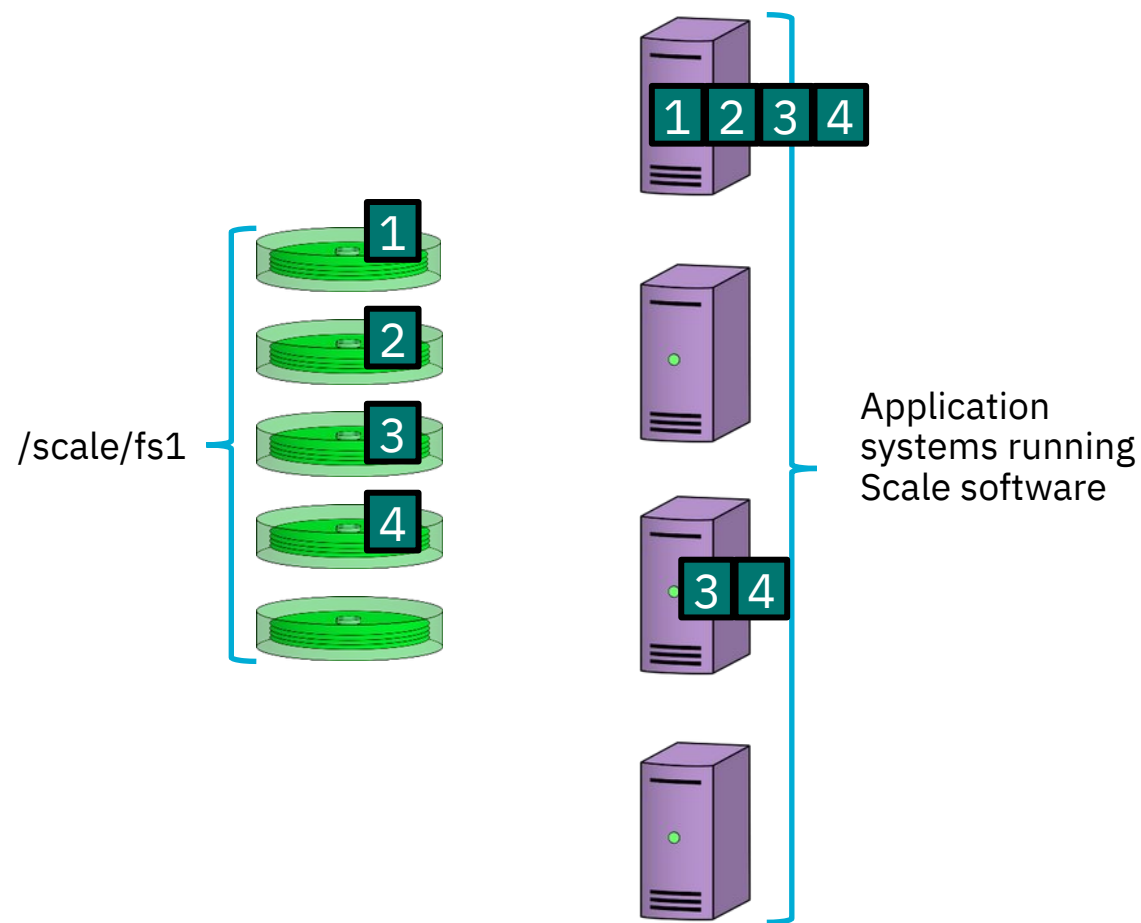
... careful so we don't step on ourselves...



Parallel I/O – the secret to performance with Storage Scale

Striping a file system over multiple disks is the key to Scale's performance.

- Large files are written over multiple disks – and those writes are sent in **parallel**.
- The **cluster manager** ensures that application nodes are coordinating their access to the disks.
- Reads can also happen in parallel – Scale's prefetch mechanism leverages this.
- Different nodes may access the same file – the **Distributed Lock Manager (DLM)** will make ensure that the nodes have a consistent view of the file system and the data in the files.
- This scales: Adding disks (which can be done online) adds both *capacity and performance* to the original file system, allowing multitudes of systems to safely access huge amounts of data (and metadata) quickly.



What do we mean by “consistent view of the file system”?

For performance, modern operating systems buffer file I/O in memory caches.

POSIX operating systems, like Linux, mandate consistency between different threads (on the same system) reading and writing the same file.

- If one thread writes to a file, another thread can read that part of the file and see what was written.
- This must work even without using `lockf` or `fsync`, even if the *operating system* internally buffers file I/O in memory caches.

Scale maintains this same POSIX-guaranteed consistency *even when the threads are running on different nodes of the cluster*.

- Scale also caches file data, for performance.
- *Cache consistency* is maintained via the **distributed lock management** system, using **locks** and **tokens**.

Thread 1	Thread 2
<pre>x = os.open('/scale/aFile', os.O_RDWR os.O_CREAT) os.write(x, 100*b'A')</pre>	
	<pre>y = os.open('/scale/aFile', os.O_RDWR) os.seek(y, 100, os.SEEK_SET) os.write(y, 100*b'B')</pre>
<pre>os.lseek(x, 0, os.SEEK_SET) os.write(x, 100*b'C') os.lseek(x, 80*(2**20), os.SEEK_SET) os.write(x, 100*b'D')</pre>	
	<pre>os.lseek(y, 95, os.SEEK_SET) print(os.read(y, 10)) → b'CCCCBBBBB'</pre>

With Scale, it doesn’t matter if threads 1 and 2 are on the same node, or on different nodes of the active cluster.

NFS does not guarantee this consistency.

Aside: Buffered vs Direct I/O

POSIX file system semantics require **buffered I/O**, with consistency handled through the distributed lock manager (DLM) system.

- Most applications expect POSIX semantics.

Direct I/O bypasses operating system buffers and avoids the DLM overhead by writing directly to NSDs.

- With direct-attached (SAN) NSDs, direct I/O can bypass the `mmfsd` daemon altogether.

Generally, only databases and similar programs *designed for Direct I/O* can use this feature without corrupting data.

Direct I/O is enabled by one of these:

- Set Direct I/O attribute on a file: `mmchattr -D yes`
- Open the file with the `O_DIRECT` flag to `open()`

Limitations for Direct I/O

Files opened for Direct I/O will only use Direct I/O:

- Bytes of data transferred must be a multiple of the sector size (512 or 4096), aligned to the start of a sector.
- The write must not cause the file to increase in length
- The write must not be to a hole or preallocated region (not yet written).
- When these conditions are not met, buffered I/O is used but immediately flushed.

Some operations require `mmfsd`:

- The NSD protocol is needed (not direct-attached)
- Data replication is used
- The operation is happening on Windows

Direct I/O is most likely to benefit programs that randomly access files with no locality.

- The application takes on the burden of file consistency.

File system manager

Several of the nodes in a cluster (frequently quorum nodes) are designated as **manager nodes**.

For each file system, one manager node is chosen by the cluster manager to be the **file system manager**. The remaining manager nodes are additional **token managers** for that file system.

The file system manager is responsible for:

- Managing the internal file system configuration
 - Adding and removing disks
 - Changes to disk status
 - Repairs to the file system
- Managing disk space allocation
- Quota controls and management
- **Token management** (along with the other token managers)

To see the file system manager, use: `mm1smgr FSDEV`

To move the file system manager, use: `mmchmgr FSDEV NODE`

```
# mm1smgr fs1
file system      manager node      [from 10.0.200.11 (a-scale01)]
-----
fs1              10.0.200.11 (a-scale01)

# mmchmgr fs1 b-scale01
Sending migrate request to current manager node 10.0.200.11 (a-scale01).
Node 10.0.200.11 (a-scale01) resigned as manager for fs1.
Node 10.0.200.21 (b-scale01) appointed as manager for fs1.
```

The metanode

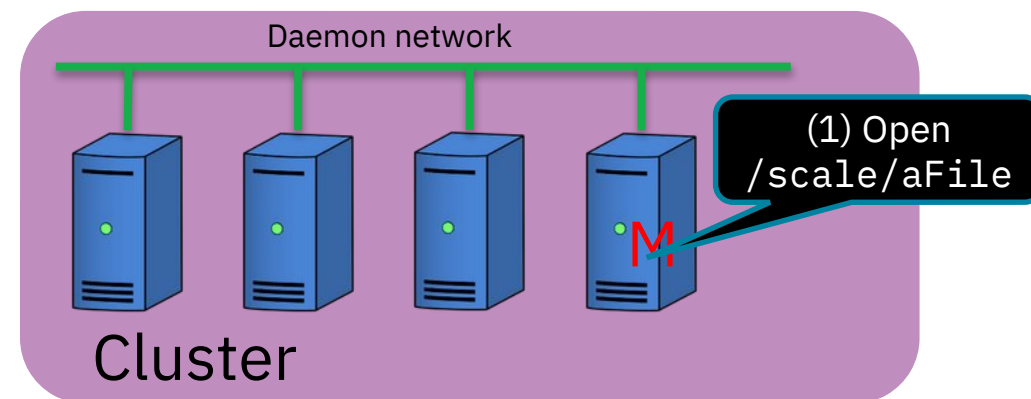
For every open file, some node (usually the one that first opened the file) is designated the **metanode** for that file.

The metanode oversees all metadata operations relating to that file, such as file length, indirect blocks, atime, etc.

Nodes needing to change metadata related to a file ask the file's designated metanode to do it for them.

Open directories (think current working directory) will also have a metanode.

The metanode is often a client.



Memory regions used by Scale

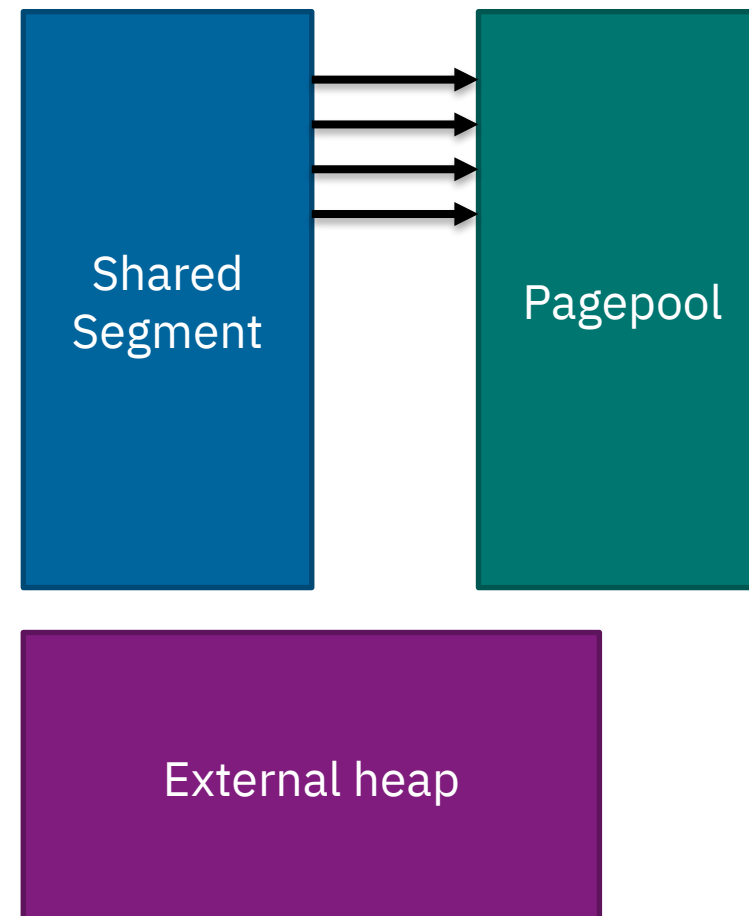
- The **pagepool** is pinned memory holding cached file data, cached inodes, etc.

The **sharedSegment** is shared memory holding references to data in the pagepool.

- References to data in the pagepool, and some other internal “metadata”
- Dominant cost is about 10kB per maxFilesToCache entry, 480B per maxStatCache entry.
- On token managers, the sharedSegment also includes token state.
- Total size depends on configuration of nodes mounting file systems – roughly $0.5kB * \frac{(\text{total cached objects in cluster})}{(\text{number of token managers})}$ then monitor actual usage!

The **external heap** is normal memory allocated to processes.

Use `mmcachectl` to explore cache usage.



Set the Linux kernel tunable `vm.min_free_kbytes` to 5-6% RAM, but no more than 2 GiB – use `sysctl` for this.

Dynamic pagepool – new with Scale 5.1.9

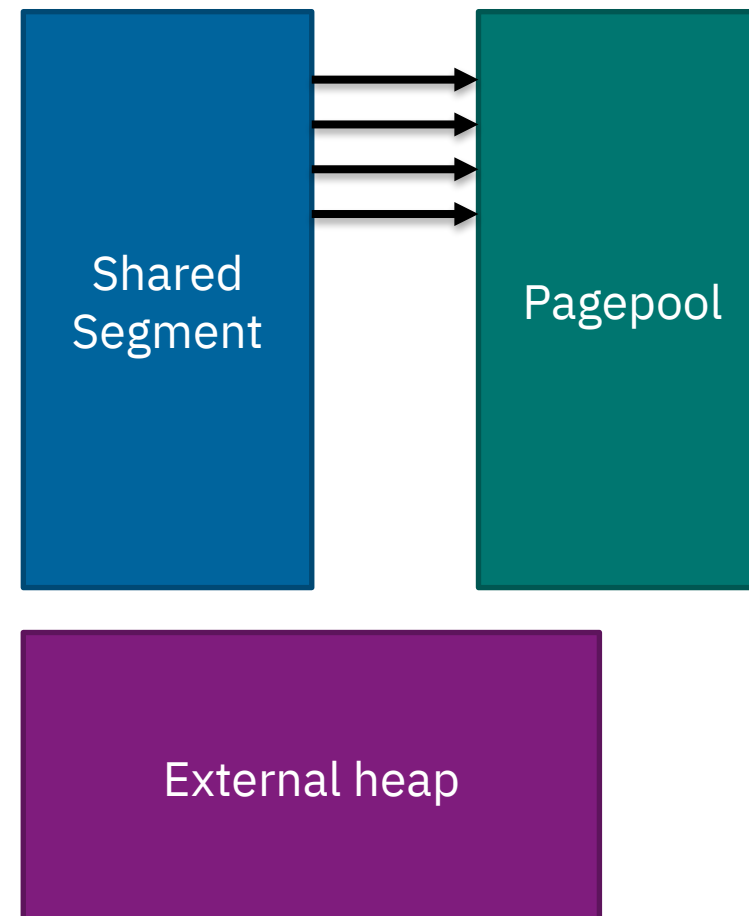
Dynamic pagepool helps balance memory usage between applications and Scale's buffers:

- When needed, pagepool will increase, up to `pagepoolMaxPhysMemPct` of memory.
- Application memory pressure will reduce pagepool usage, down to `pagepoolMinPhysMemPct` of memory.

Requirements:

- ~~Node cannot be an NSD server.~~
- Node must run Linux and have at least 6 GiB RAM.

```
mmchconfig dynamicPagepoolEnabled=yes -N NODES...
mmchconfig pagepoolMinPhysMemPct=MIN -N NODES...
mmchconfig pagepoolMaxPhysMemPct=MAX -N NODES...
mmchconfig pagepool=DEFAULT -N NODES...
#
# Monitor usage:
mmdiag --pagepool
```



For systems with > 16 GiB RAM, set the Linux kernel tunable `vm.max_map_count` to $((\text{RAM in bytes})/262144)$. You may use `sysctl` to do this.

Tuning nodes

Scale has many parameters to control behavior – although only a few are normally needed.

Set parameters with `mmchconfig` (some parameters require restarting Scale).

Check parameters using `mmisconfig`.

Generally, very few parameters should be adjusted – others are computed from these or have sensible defaults. (ESS has many special values – don't change those!)

Be careful not to inadvertently change parameters on systems you didn't intend to – node classes help!

Affecting memory usage

- `numaMemoryInterleave`
- `pagepool`
- `maxFilesToCache`
- `maxStatCache`
- `maxBufferDescs`

Affecting I/O scheduling

- `maxMBps`
- `ignorePrefetchLUNCount`
- `workerThreads`

Affecting communication

- `maxTcpConnsPerNodeConn`
- `maxReceiverThreads`

Distributed Lock Management and those mysterious “tokens”

Toto, I’ve a feeling we’re not in Kansas anymore.

Locks and tokens

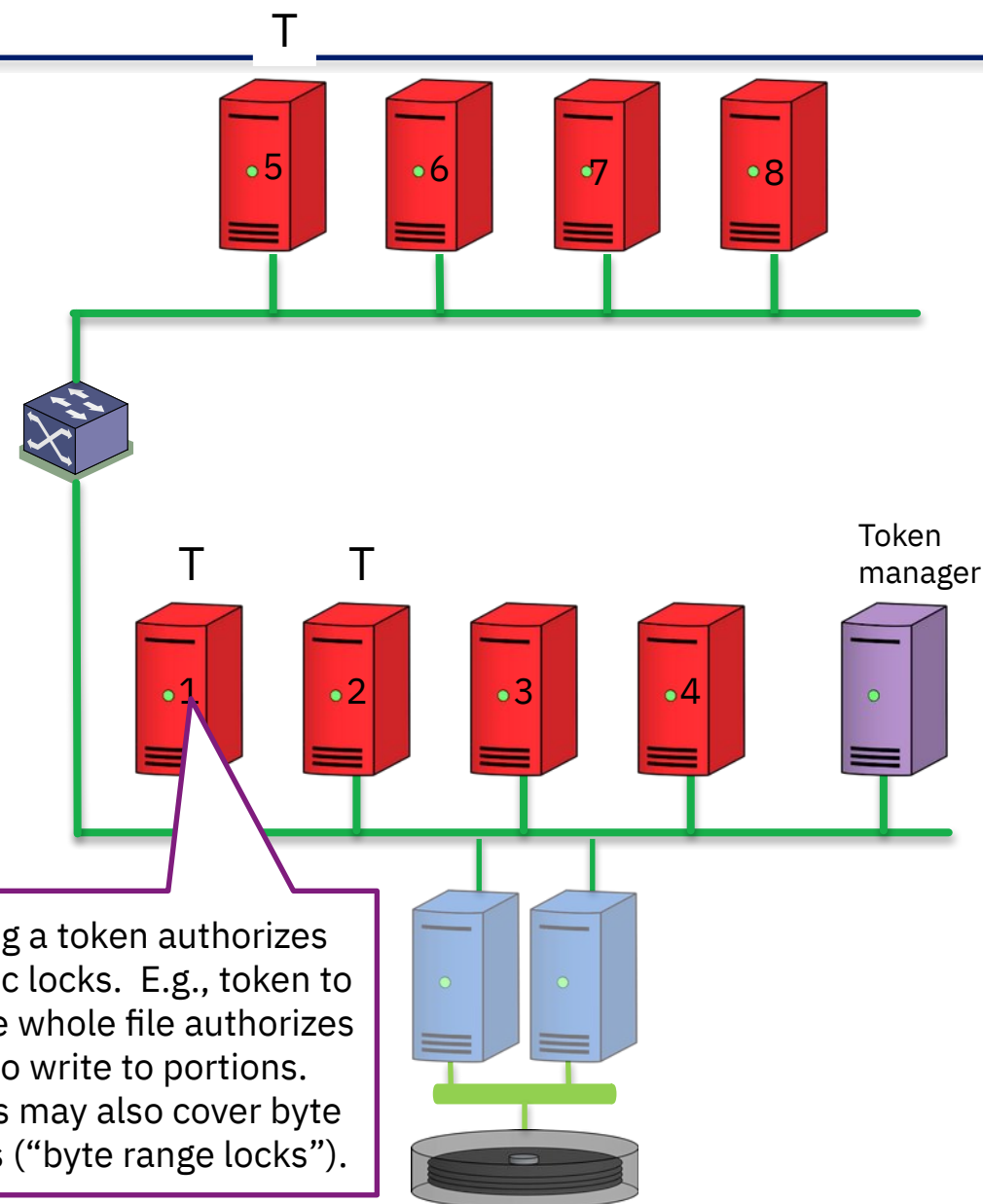
Locks (often called **mutexes**) efficiently control access to *caches* of files and metadata structures on a single node.

- The mmfsd process serves as a local lock manager.
- Locks will mediate access between threads on the node.
- Every file system operation will take out an appropriate lock, do the operation, and release the lock when done.

Tokens control which locks can be used on a particular node.

- All cached data must be protected with a token.
- When a lock is requested, the system will check if there is a token permitting that lock.
 - If the operation is permitted, the lock is granted.
 - If no token is held authorizing that lock, the node will need to request an appropriate token.

Tokens extend locks into a **distributed lock management** system.



Cache consistency with byte ranges (conceptual)

Node 1	Node 2	(Possible) token activity
<pre>x = os.open('/scale/aFile', os.O_RDWR os.O_CREAT) os.write(x, 100*b'A')</pre>		1. Node 1 gets token for entire file. 2. Node 1 updates cache (0,99)
	<pre>y = os.open('/scale/aFile', os.O_RDWR) os.seek(y, 100, os.SEEK_SET) os.write(y, 100*b'B')</pre>	1. Node 1 relinquishes token for range (100,inf), but keeps token for (0,99) 2. Node 2 receives token for (100,inf) 3. Node 2 writes to its own cache for (100,199)
<pre>os.lseek(x, 0, os.SEEK_SET) os.write(x, 100*b'C') os.lseek(x, 80*(2**20), os.SEEK_SET) os.write(x, 100*b'D')</pre>		1. Node 1 can freely change cache of (0,99) 2. Node 2 relinquishes token on (80M, inf) 3. Node 1 gets additional token on (80M, inf) 4. Node 1 updates cache of region (80M,80M+100)
	<pre>os.lseek(y, 95, os.SEEK_SET) print(os.read(y, 10)) → b'CCCCBBBBB'</pre>	1. Node 1 must relinquish token for (0,99) forcing a write 2. Node 2 extends token to (100, 80M) 3. Node 2 must read misses

Tokens give permission to maintain and use an internal cache on a byte range of a file system object.

- When a token is revoked, dirty data must be written before the next node can proceed.
- Eventually all dirty cached data gets written (sync thread)

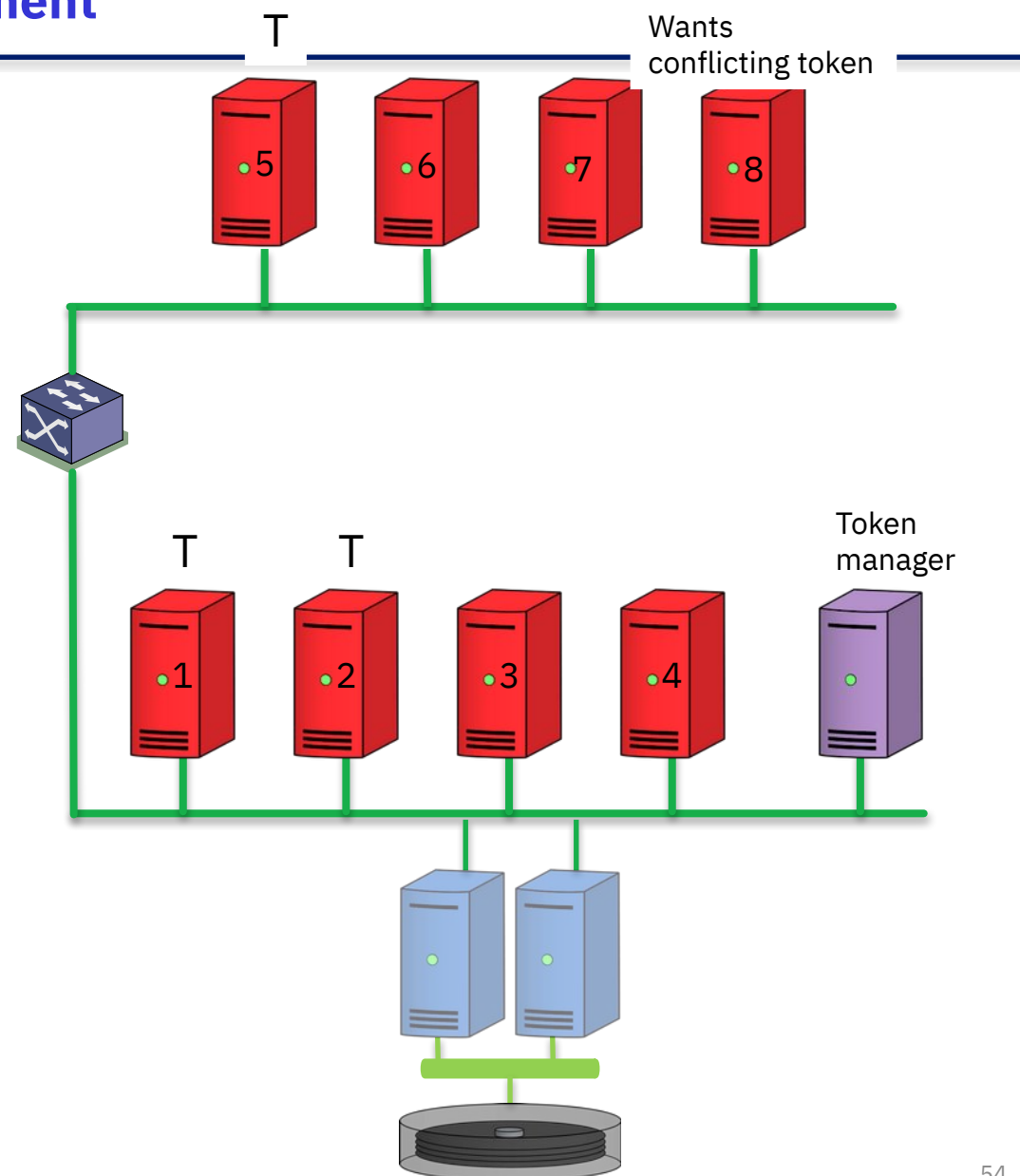
In reality, **byte-range locks have a granularity of a file system block.**

Distributed lock management (DLM): Token management

Tokens give a node permission to maintain and use an internal cache of part of a file system object (file, directory, or metadata) using distributed byte-range locks (at the granularity of a file system block) so it can *efficiently* perform file system operations.

Nodes request tokens from the **token managers** (including the file system manager).

- If granted, a node may operate on the object until the token is revoked.
 - Dirty data and metadata are flushed to disk when a token is revoked.
- If denied:
 1. The token manager tells the requesting node which other nodes have conflicting tokens.
 2. The node requesting the token communicates with those other nodes to revoke their tokens, collecting their responses.
 3. The requesting node updates the token manager with the current status.

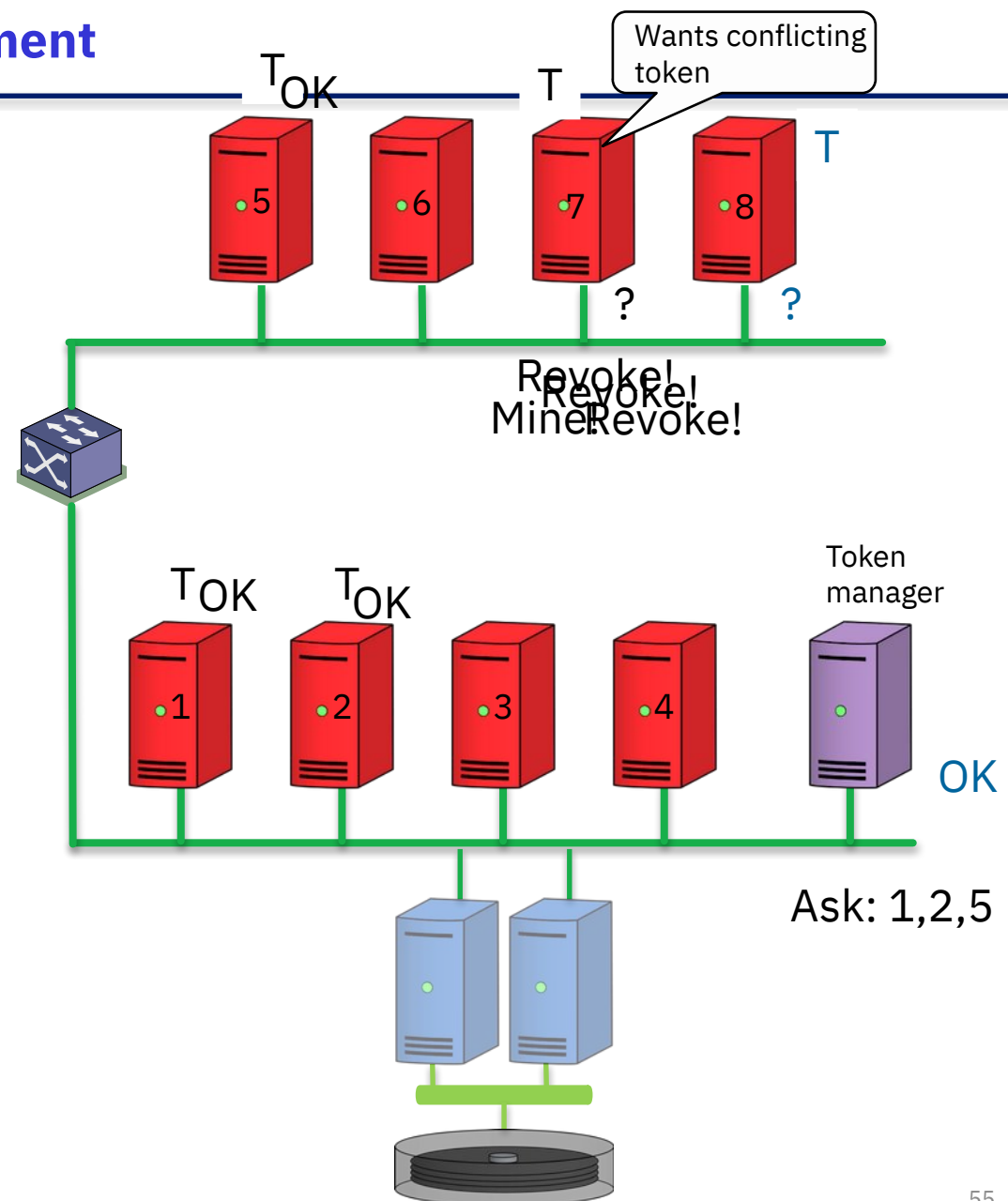


Distributed lock management (DLM): Token management

Tokens give a node permission to maintain and use an internal cache of part of a file system object (file, directory, or metadata) using distributed byte-range locks (at the granularity of a file system block) so it can *efficiently* perform file system operations.

Nodes request tokens from the **token managers** (including the file system manager).

- If granted, a node may operate on the object until the token is revoked.
 - Dirty data and metadata are flushed to disk when a token is revoked.
- If denied:
 1. The token manager tells the requesting node which other nodes have conflicting tokens.
 2. The node requesting the token communicates with those other nodes to revoke their tokens, collecting their responses.
 3. The requesting node updates the token manager with the current status.



Beware of “false sharing”

Byte-range locks allow threads on a node to use parallel writes to non-overlapping regions of a file but serialize overlapping writes.

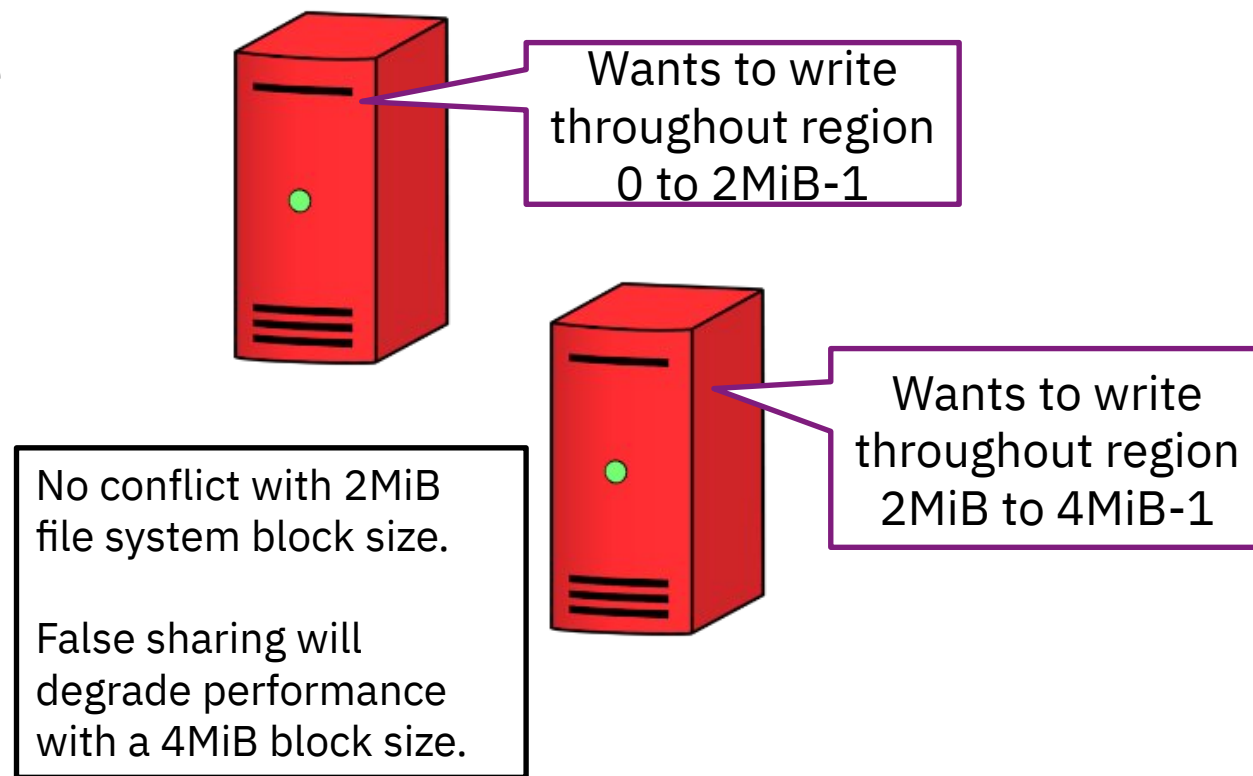
- Since the smallest I/O transfer size is a sector, byte-range locks round up the protected region to full sectors.
- Threads might not strictly overlap byte ranges within a sector – but will need the same lock. This is called **false sharing**.

Byte-range tokens allow nodes to create compatible byte-range locks on a region of a file.

- Byte-range tokens round the protected region to a full file system block.
- Threads on different nodes might not overlap byte ranges within a block – but only one node can have a token. This is **false sharing between nodes**.

Especially watch out for false sharing with:

- Databases
- Using mmap
- Log files opened by multiple threads/nodes (might try enabling `appendShipEnabled`)



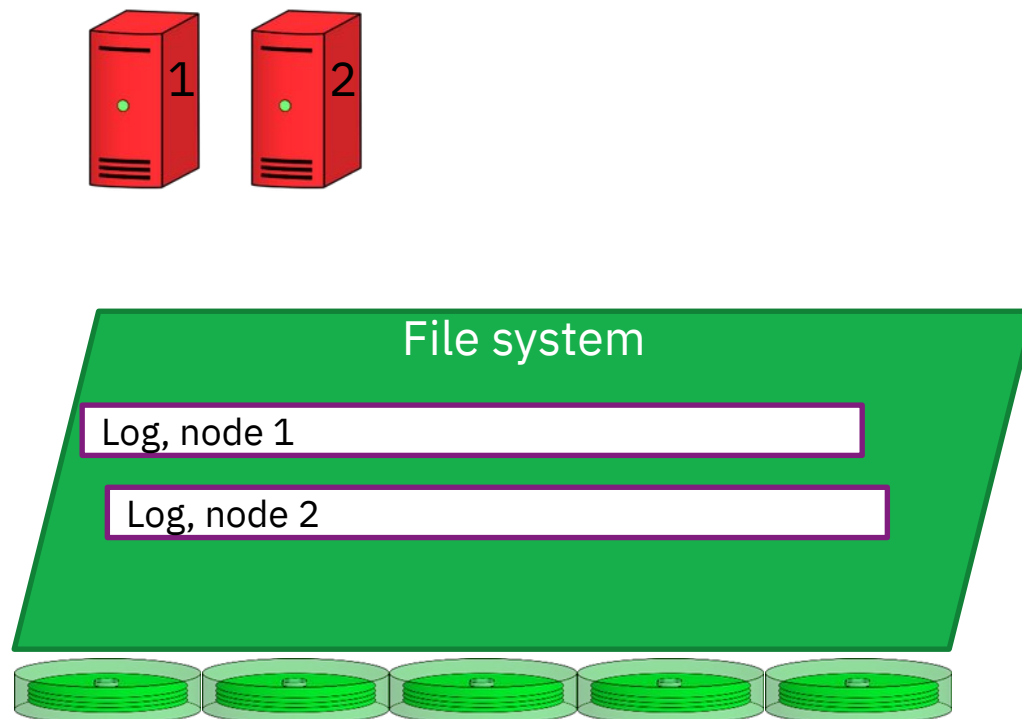
File system recovery log

Certain metadata operations (and with the HAWC feature, small data writes) are first described in a file system **log (journal)** – this is itself a metadata structure (but can be moved to `system.log pool`).

- Each node is assigned a log for each file system it mounted, stored in that file system.
- Log entries and token management are strictly ordered to maintain file system consistency, even in the face of a node failure.
- When a node fails, the file system manager will replay its log (after disk leases expire), then tell the token manager to release the node's tokens.

The `-L` flag to `mmcrfs` (`mmchfs`) changes log size – it can be changed from default (32MiB) up to 1024MiB.

- Default is usually good...
- Raise it for HAWC, or with high write rates, or with replication.
- But increased log size will make file system recovery from a node failure take longer.



Highly-available write cache (HAWC)

Small synchronous writes are common with VMs and databases, but horribly inefficient:

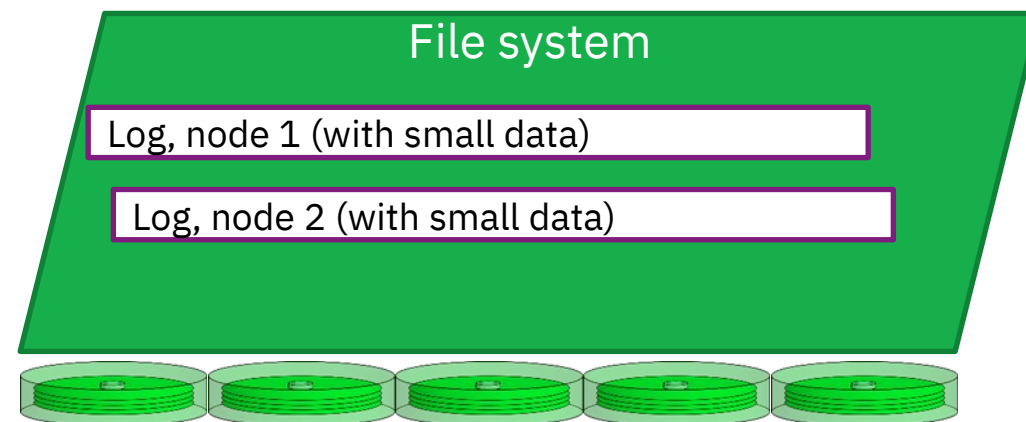
- Seek time of transaction dominates total time.
- Initial write to a block requires rest of block to be zeroed, amplifying the write.
- Usually, multiple writes can coalesce in buffer into a single larger write, but `fsync` prevents this.

The “Highly-available write cache” (HAWC) feature allows small (up to 64KiB) writes to be hardened into the recovery log so pagepool buffer can effectively and safely coalesce many small writes into fewer, larger writes.

HAWC consists of:

1. Ability to separate recovery log as `system.log` pool (if there is small amount of very fast storage).
2. Set a write cache threshold for the log.

```
# mmchfs fs1 -L 1024 \  
                --write-cache-threshold 64K  
  
# # With separate system.log, can also set:  
# mmchfs fs1 --log-replicas 3
```



Block allocation maps and token management traffic

The **block allocation map** is logically a bit vector representing each subblock's allocation status.

- The map is divided into n lockable regions, where n depends on the *node count* parameter of the file system.
- Each region represents a portion of each disk.
- The allocation manager thread attempts to give each node a region with free space; otherwise allocating blocks will require stealing tokens on to share regions.
- For file deletion, the allocation manager delegates work to any nodes already holding regions representing blocks in the file (maybe increase `maxActiveIallocSegs` to 8, especially with old file systems).
- Insufficient regions leads to token contention, increased network traffic, and latency.
- When creating file systems with `mmcrfs`, pay attention to the `-n` parameter (number of clients mounting the file system). This affects how many regions are created and *cannot be fully changed later*.

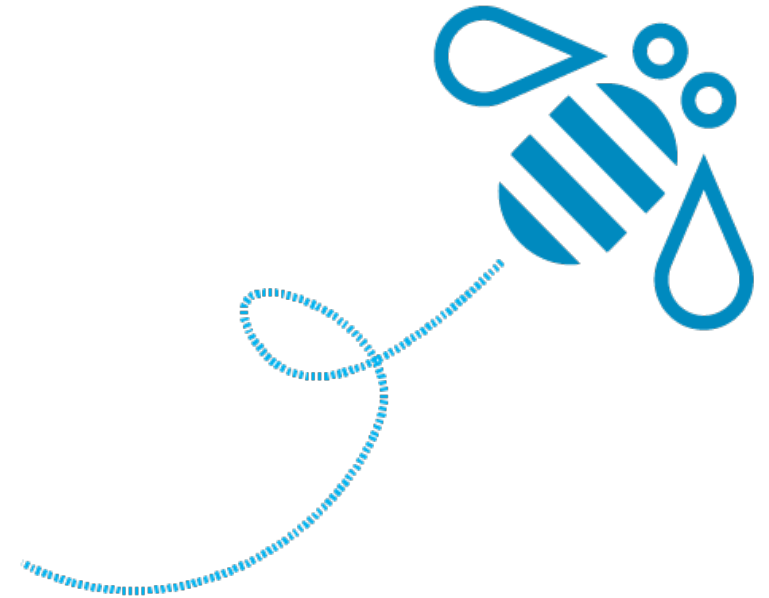
- These logical regions are written serially as a metadata file.
- If disks are added, additional “segments” of the block map are created, laid out just like the first segment.



Questions about chapter 4, or anything else?



Thank you!

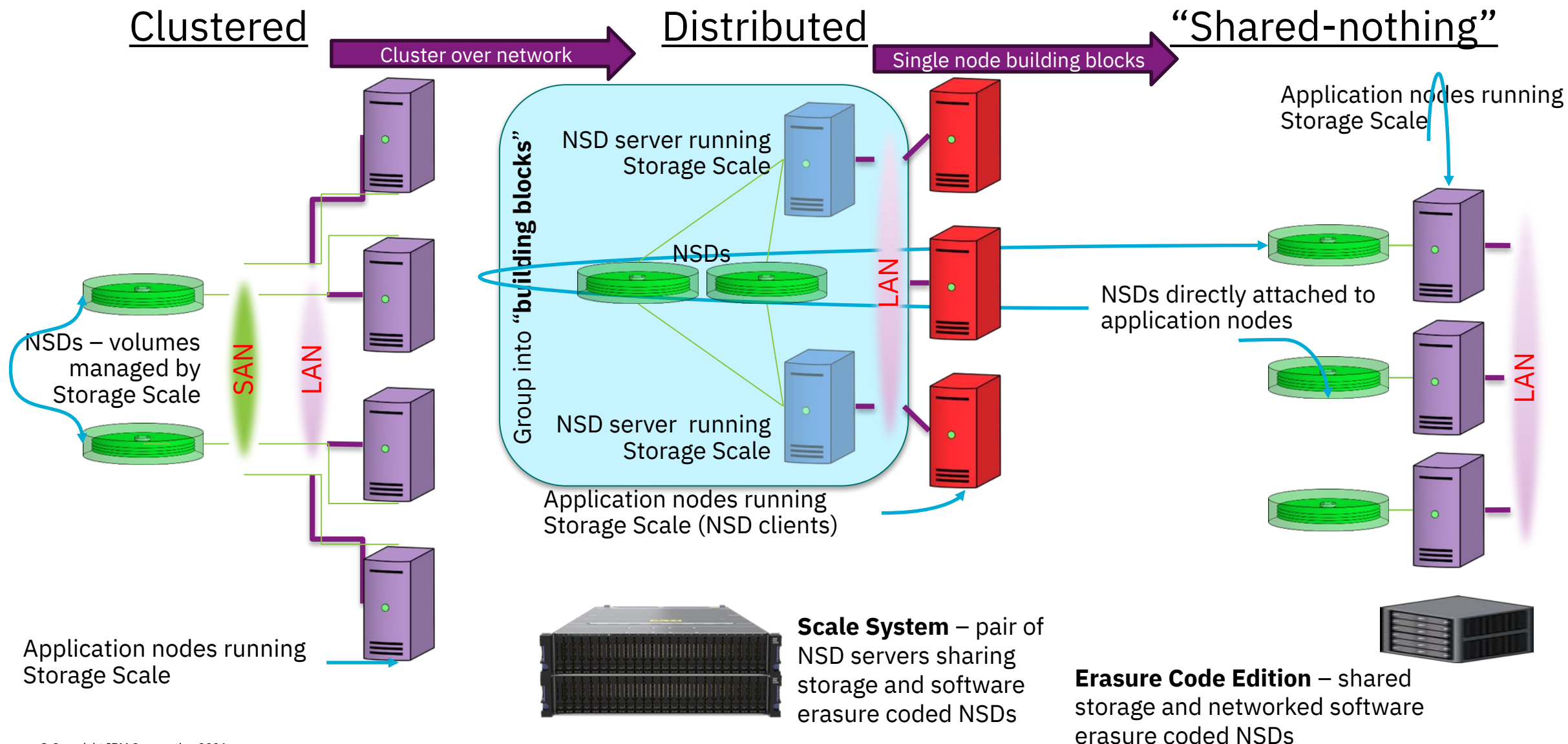


Appendix

Deployment topologies

... so many options

High-level deployment strategies



When to use the clustered topology

Small and cheap

- Can use existing storage
- Can use existing SAN
- Familiar to many storage administrators

Fast and reliable

- Direct to SAN performance – especially important if enterprise network can not handle the load.

Contraindication: All nodes connected to the SAN must use the same operating system platform (since OS services are used for multipath control). If this is not true, then must combine with the distributed topology.

When to use the distributed topology

Accelerate existing NAS applications

- Use existing network
- NSD protocol is faster than NAS protocols (NFS, SMB).

Large and scaleable

- Pool many types of storage
- Heterogenous operating systems
- Plan to use ESS

When to use the shared-nothing topology

Hadoop (FPO)

- Hadoop compatible (replaces HDFS)
- POSIX interface
- Enterprise features
- **(But consider if common storage and the Hadoop connector will work better.)**

“Cheap storage”

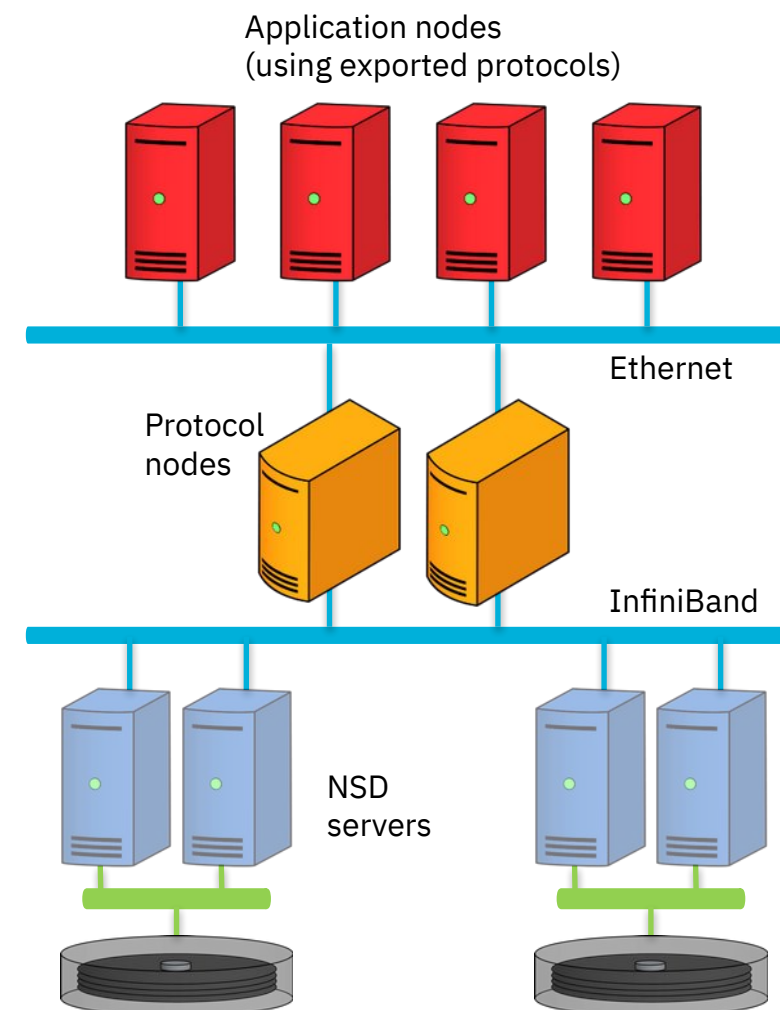
- In-server disk
- Perceived as “cheap” storage
- Can be basis for hyperconverged deployment.

Protocol node topologies

Protocol nodes should not be introduced gratuitously.

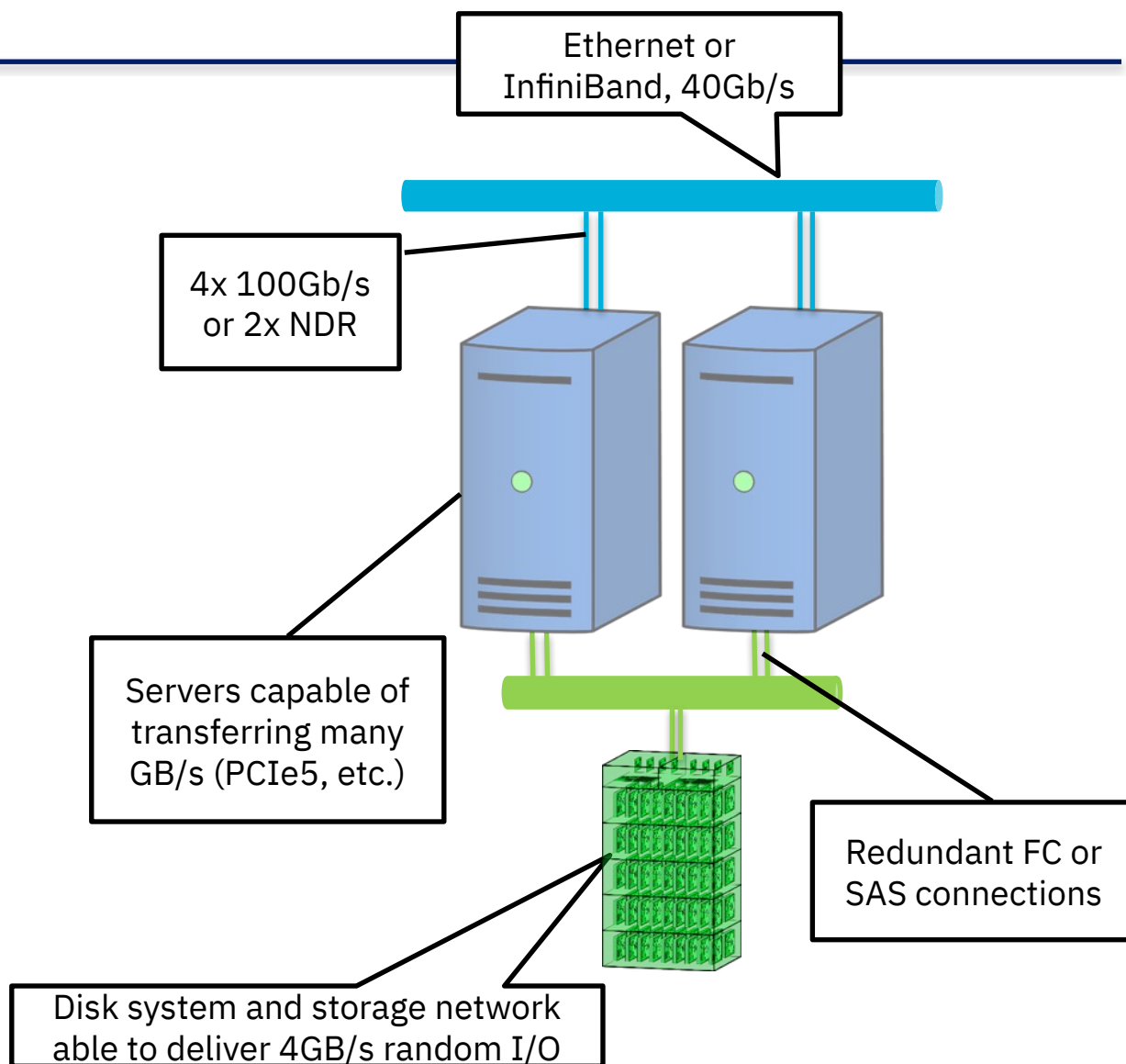
However, protocol nodes are essential when the application node can not directly run Storage Scale:

- Application needs the Object protocol
- The operating system supports NFS, but can't run Storage Scale – e.g., FreeBSD or OpenIndiana.
- In general, when Windows needs access to Storage Scale, it is simpler to use SMB through protocol nodes.
- MacOS is best supported using SMB through protocol nodes.



Building blocks

- A Storage Scale building block is a performance-matched set of NSD servers, storage shared amongst those servers, and networking.
- Each NSD server in the building block provides performance, as well as redundancy for its partner.
- Application nodes are on the Ethernet or InfiniBand network, accessing storage through NSD servers.
- Capacity and performance are increased by adding complete building blocks.



Appendix

Tiebreaker Disk quorum

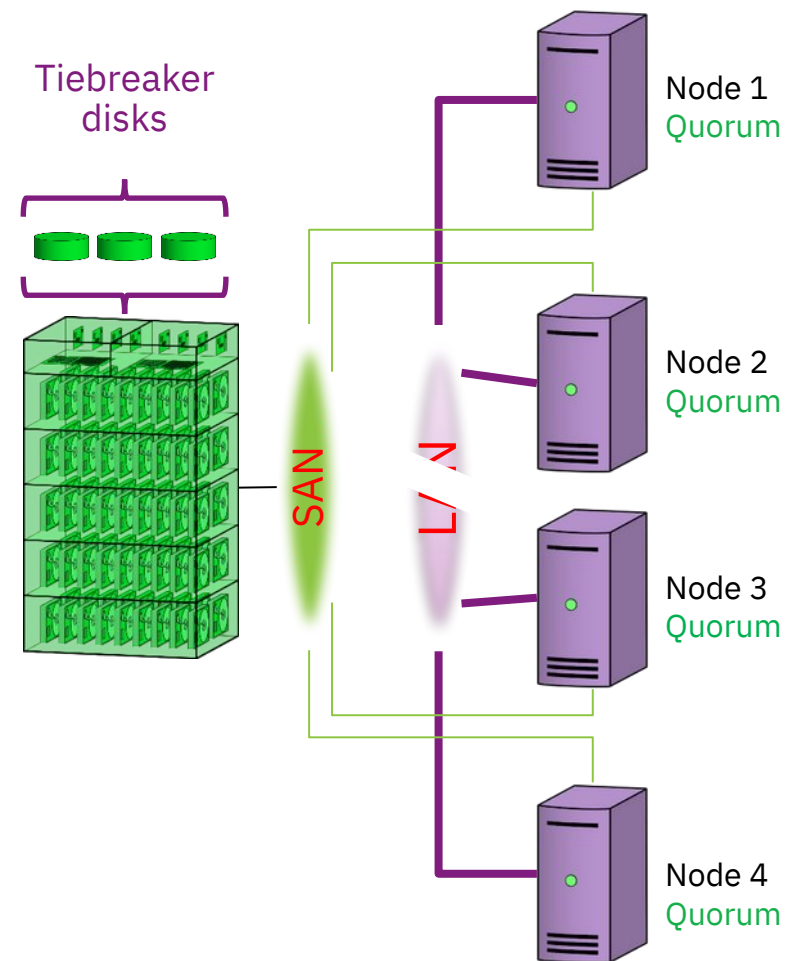
... because sometimes you trust the SAN more than the LAN

Tiebreaker disk quorum

Instead of [node quorum](#), tiebreaker disks may be used to help quorum nodes choose the cluster manager in small clusters.

- Tiebreaker disks are 1-3 specially designated NSDs.
 - These may also be used as normal NSDs, as parts of file systems.
- All quorum nodes must be directly- or SAN-connected to all the tiebreaker disks.
- A candidate node for being the cluster manager must be able to access a majority of the tiebreaker disks.
- A candidate must be able to connect to at least `minQuorumNodes` quorum nodes (including itself). The default for `minQuorumNodes` is 1.
- There is a maximum of 8 quorum nodes when tiebreaker disks are used.
- Elections require disk leases on tiebreaker disks to expire. SCSI persistent reserve can make quorum election faster.

If an election is triggered, the node that is currently the cluster manager will remain the cluster manager if it remains a candidate.



Tuning tiebreaker disk quorum

- **tiebreakerDisks** – Set to a semicolon-delimited set of 1-3 NSD names – these will be tiebreaker quorum disks.
 - This should be odd – so typically use 3 NSDs.
 - To revert to node quorum, set this to **no**.
 - This setting can be changed in running clusters.
- **minQuorumNodes** – When tiebreaker quorum is used, this sets the minimum number of quorum nodes that must be in a partition containing the cluster manager (default is 1).
- The **tiebreakerCheck** event is triggered when a quorum node detects a lack of network connectivity, and periodically on the cluster manager – a callback can be used to validate node membership.
- The CCR mechanism has some timeouts for different functions ranging from 45 seconds to 2 minutes; these may not be changed.

Appendix

Tuning disk leasing and cluster membership

... when you must push the limits

Tuning disk leasing

- **failureDetectionTime** – How many seconds it takes to detect a node is down (default is 35 seconds, same duration as a disk lease).
- **leaseRecoveryWait** – When a node fails, wait until known leases expire, then wait this many seconds, before starting recovery. Default is 35 seconds. Intent is to give “in-flight” I/O time to get through controllers on to disks.
- **usePersistentReserve** – enables SCSI persistent reserve for disk fencing. Please check documentation for guidance.
- **minMissedPingTimeout**, **maxMissedPingTimeout** – Sets the range in which calculated “missed ping timeout” (MPT) may fall (default between 3 and 60).
 - Default MPT is `leaseRecoveryWait - 5`.
 - After a lease expires, the cluster manager will ping a node, waiting MPT seconds for a response. After that, the node is expelled.
- **totalPingTimeout** – Nodes responding to ICMP pings but not sending heartbeats will be declared dead after this timeout (default 120 seconds).