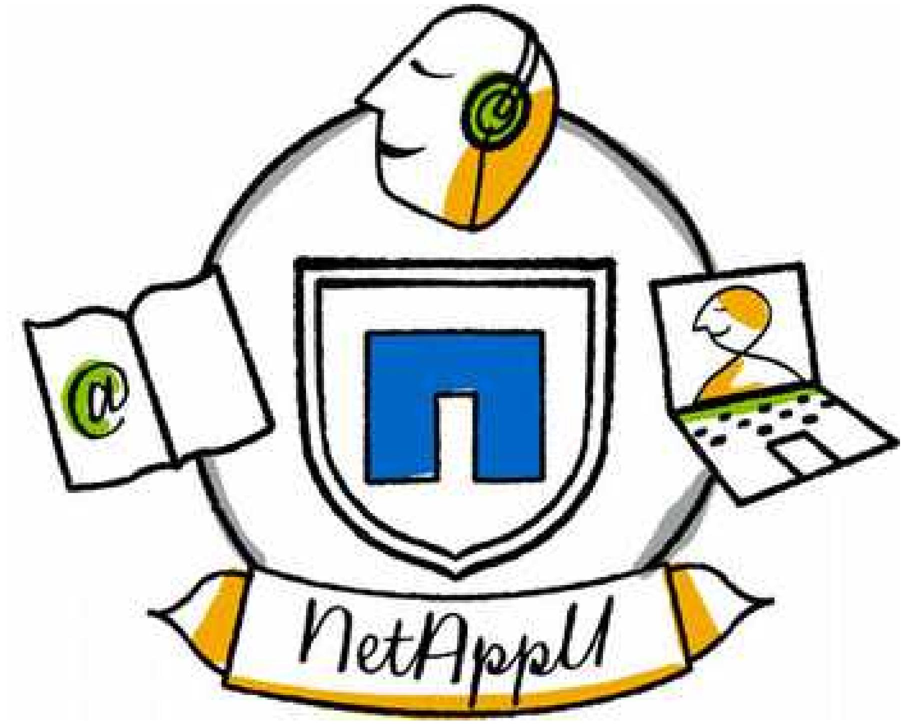
OnTap 8.x

Edmund J. Sutcliffe



NetApp

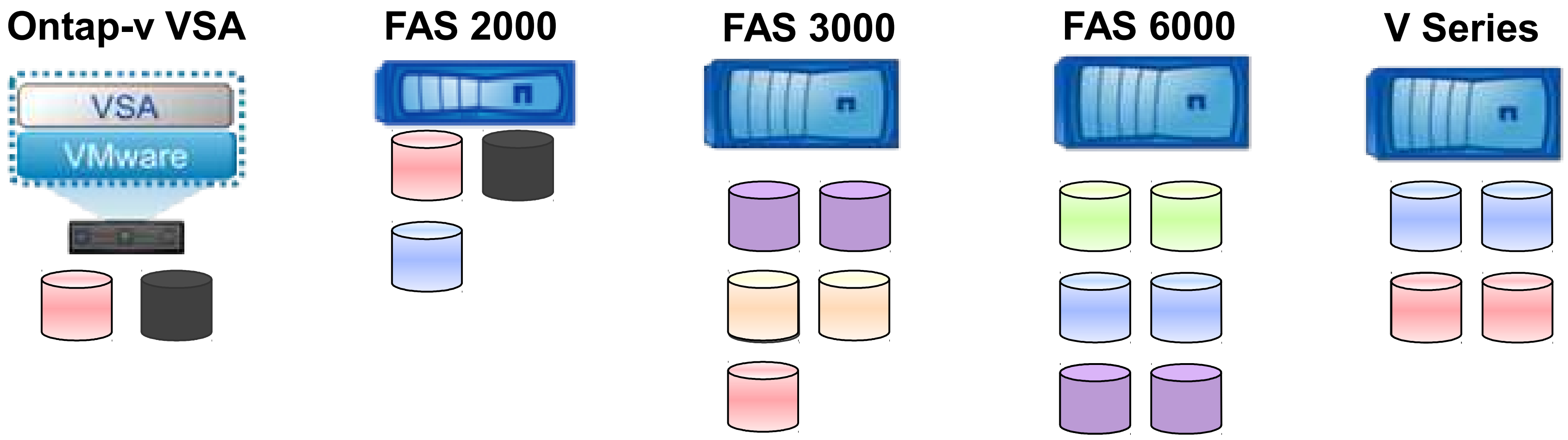
Storage Virtual Machine

- A memory-bound, cluster-wide, entity that fully abstracts all aspects of connecting backend storage from the physical array.

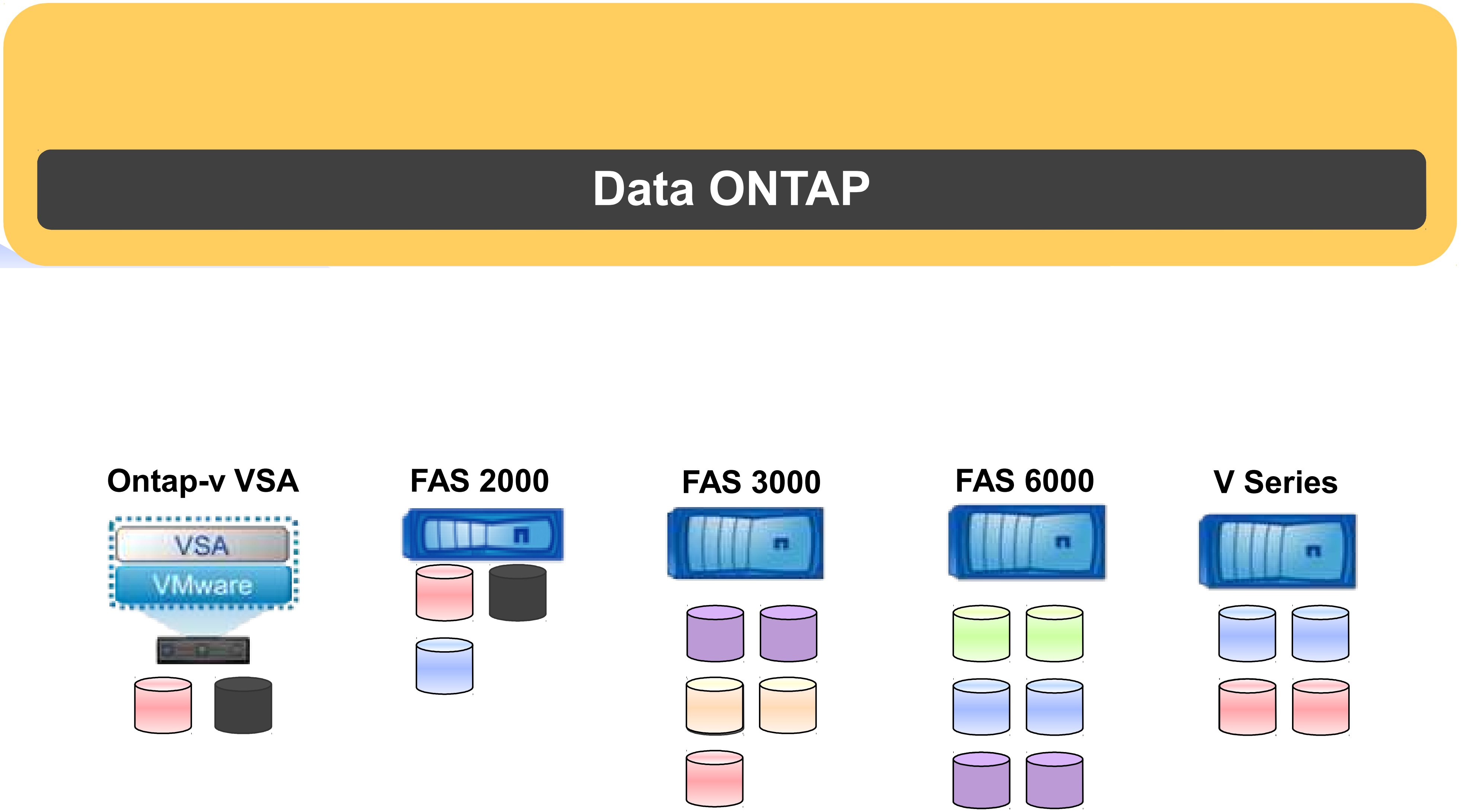
- Act as a “front-end” to the array, and virtualize access by way of a per-protocol, per-node interface known as a LIF, or logical interface.

- Correlations can be drawn between Vservers and traditional virtual machines in the way that they take physical hardware and resources and abstract access to it, allowing for more efficient uses to be made of the physical investments.

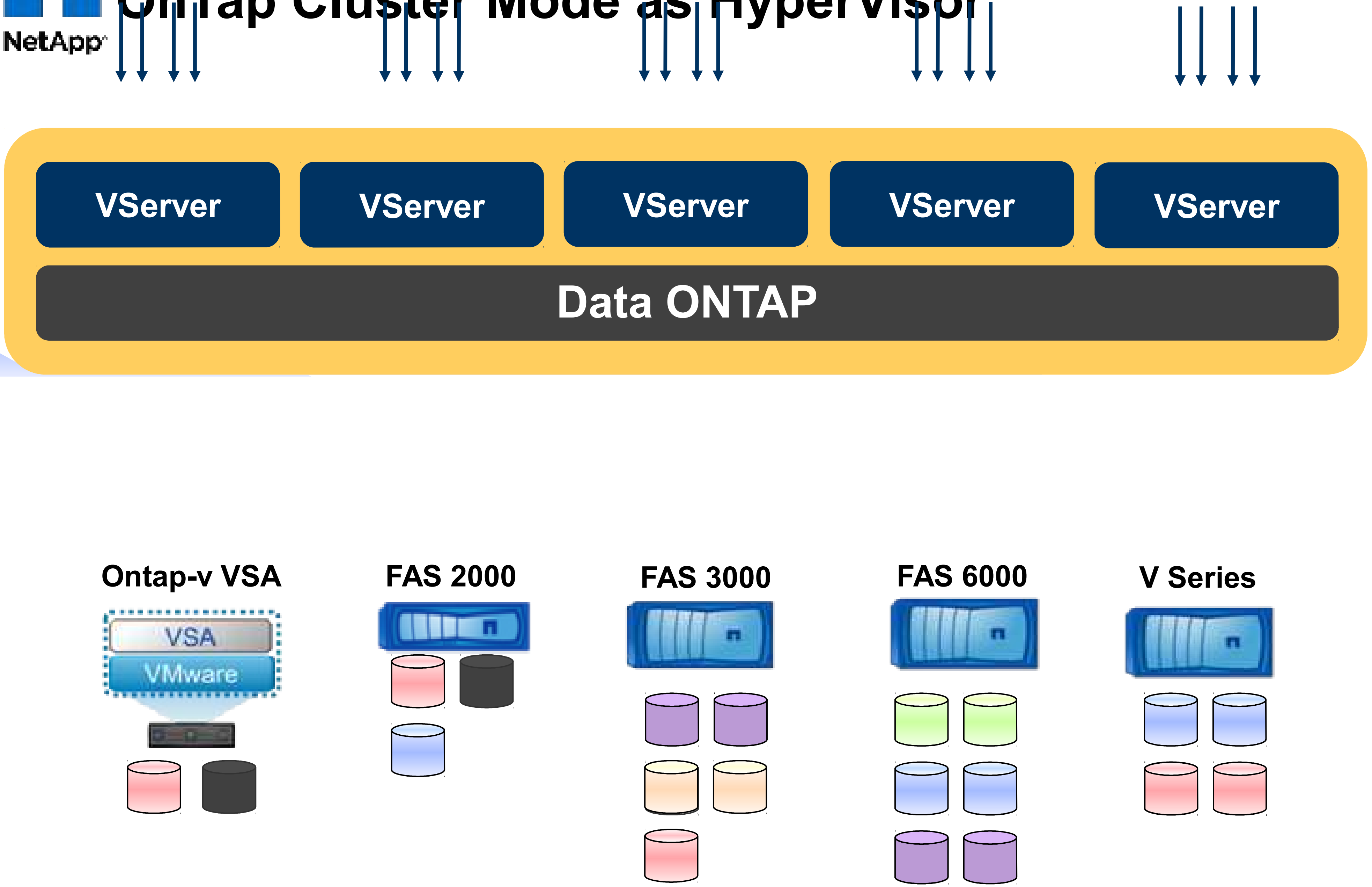
OnTap Cluster Mode



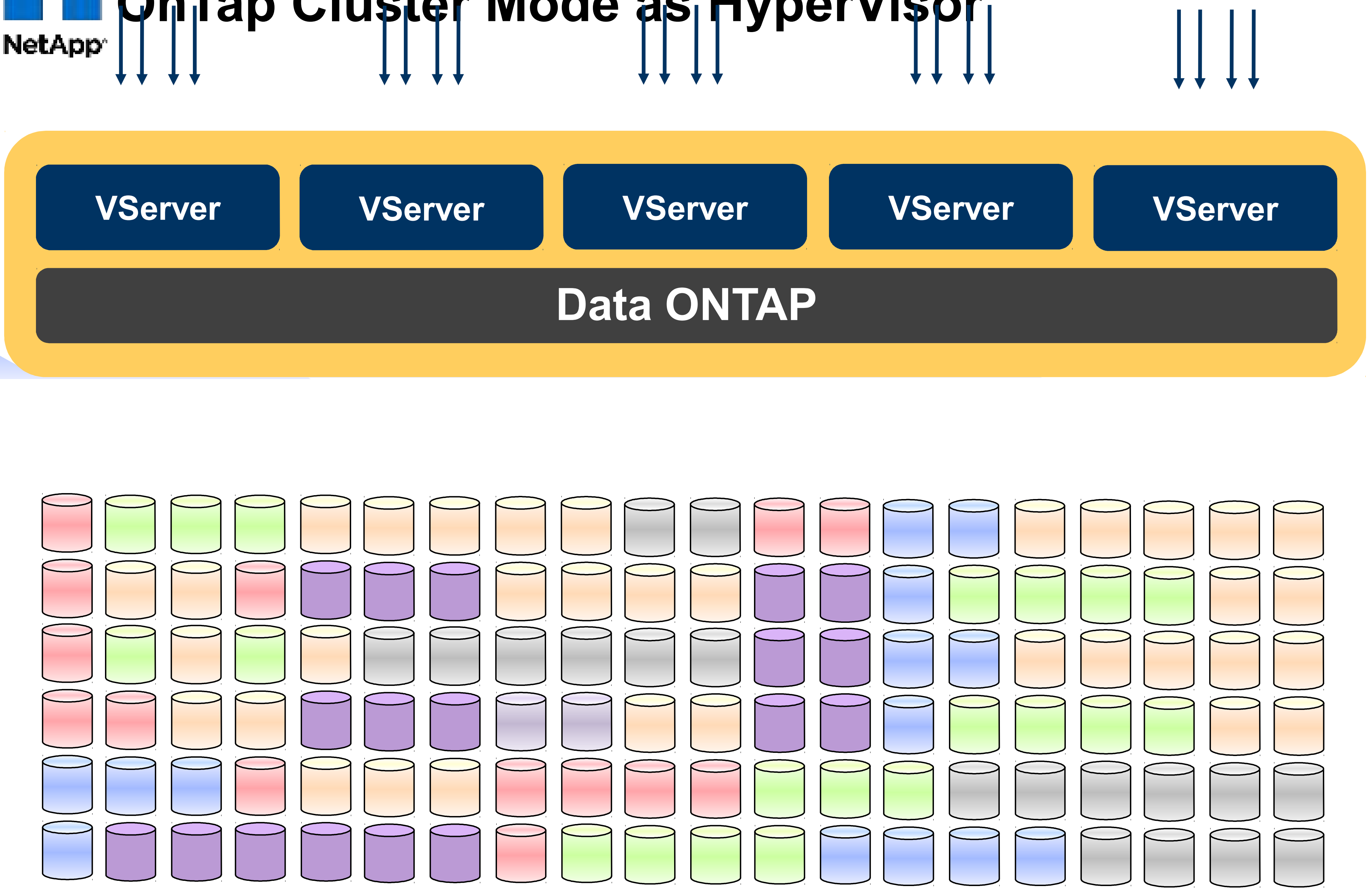
OnTap Cluster Mode as HyperVisor



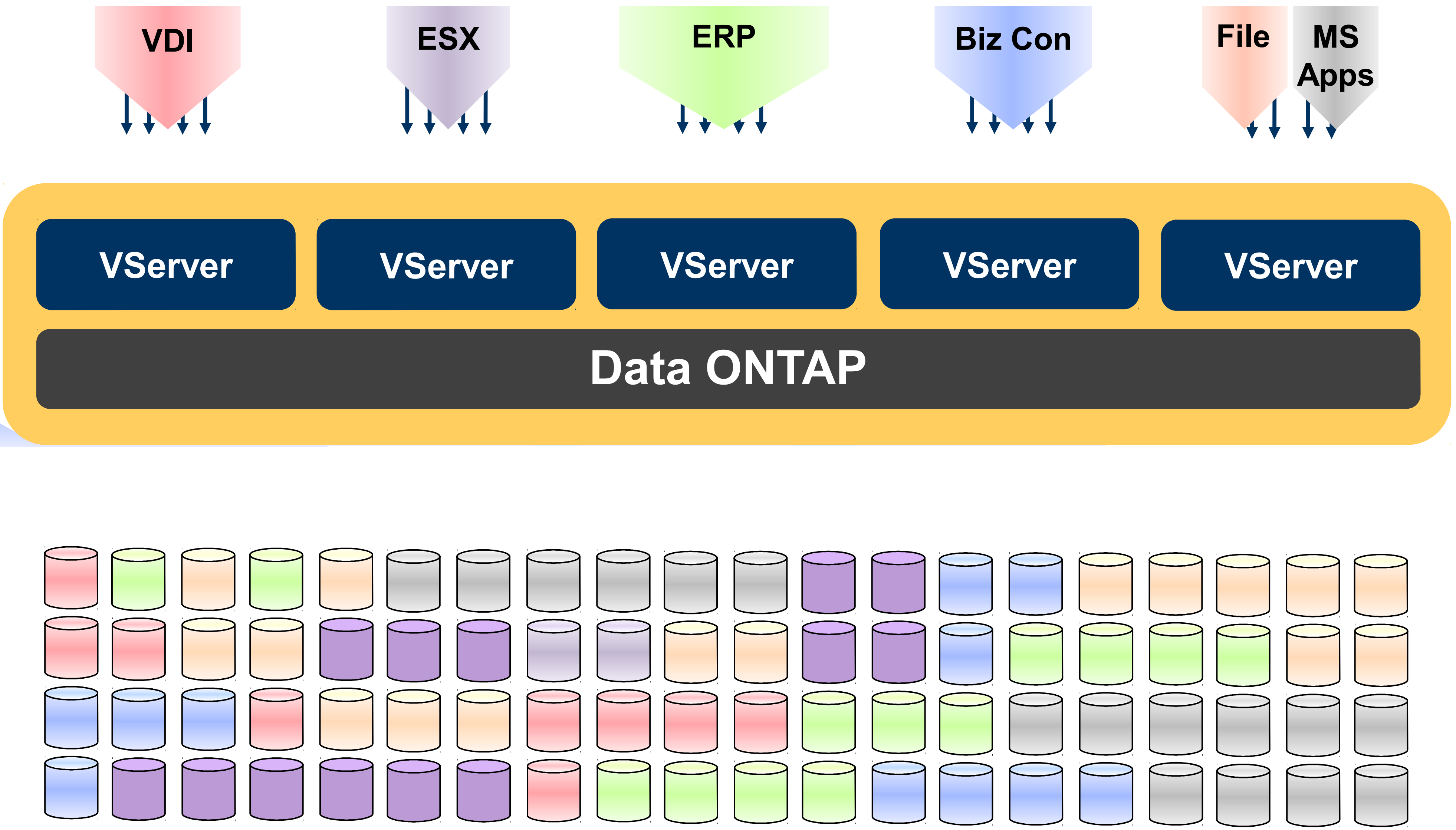
OnTap Cluster Mode as HyperVisor



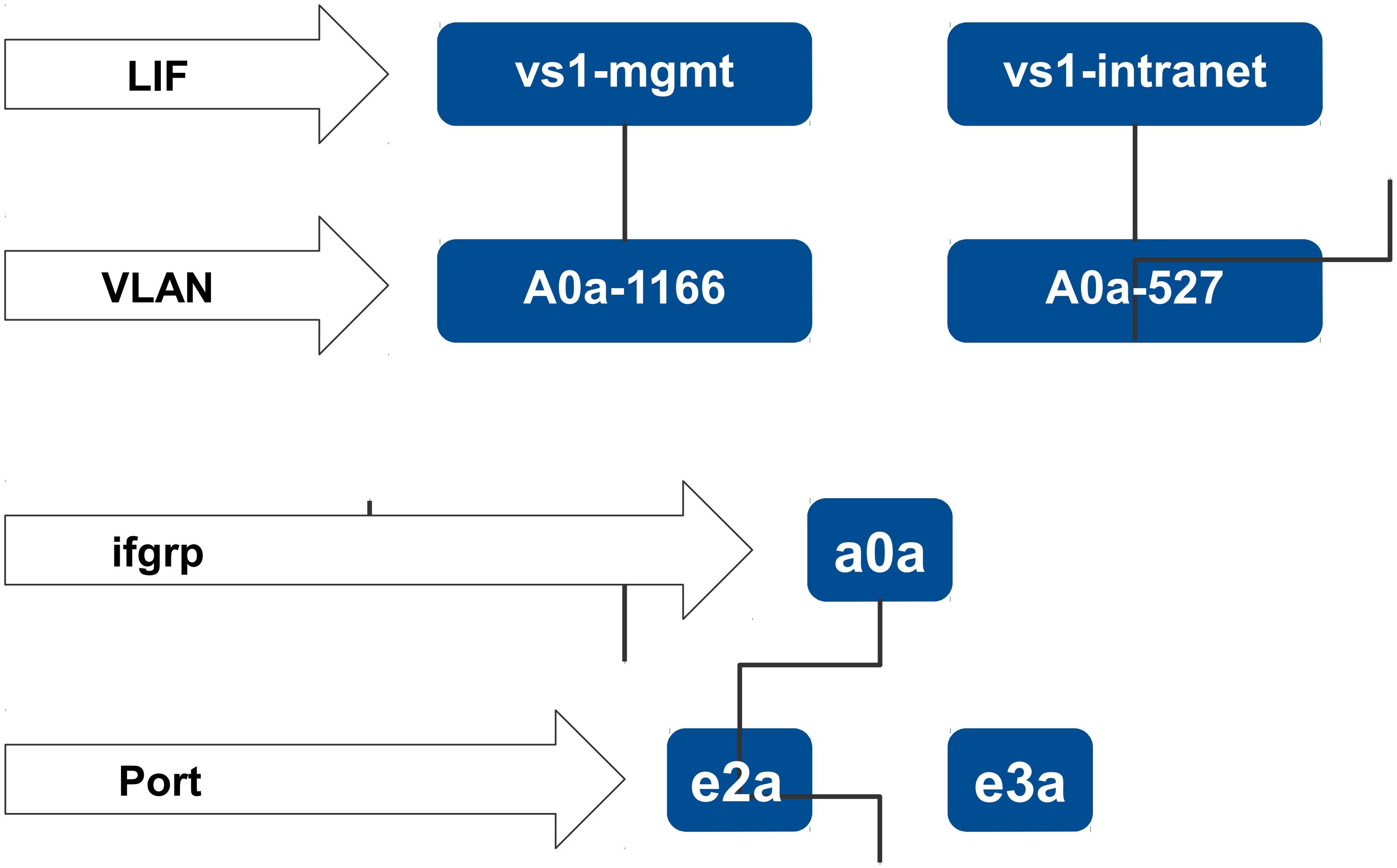
OnTap Cluster Mode as HyperVisor



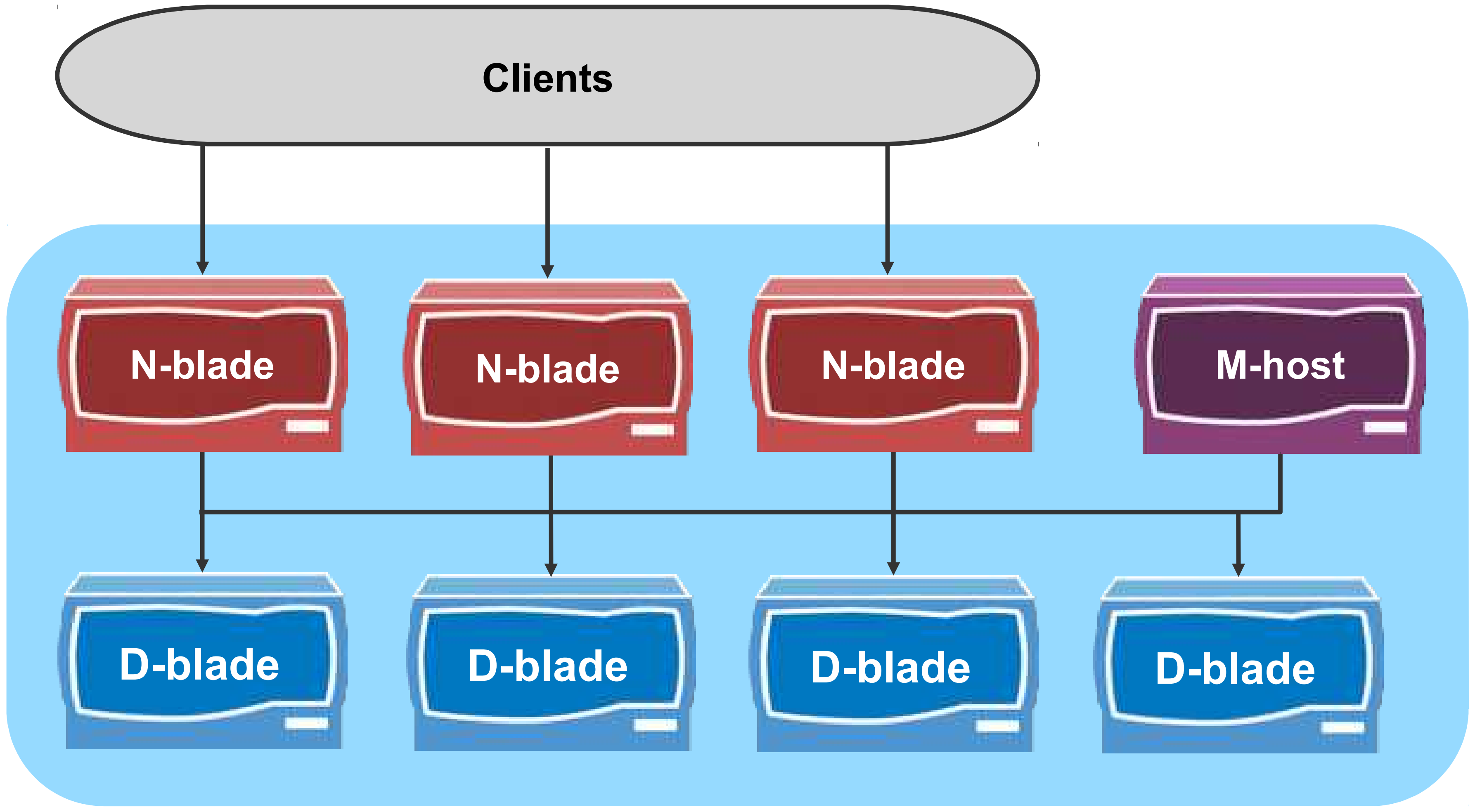
OnTap Cluster Mode as HyperVisor



How Network works?



What Clients Access

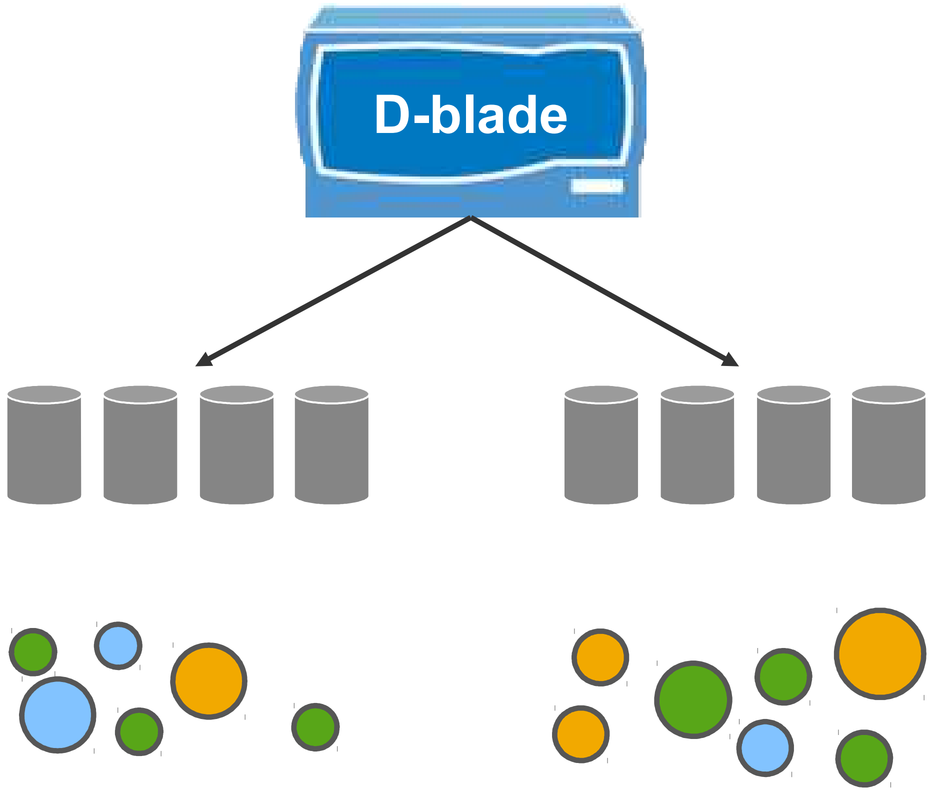


GX Cluster

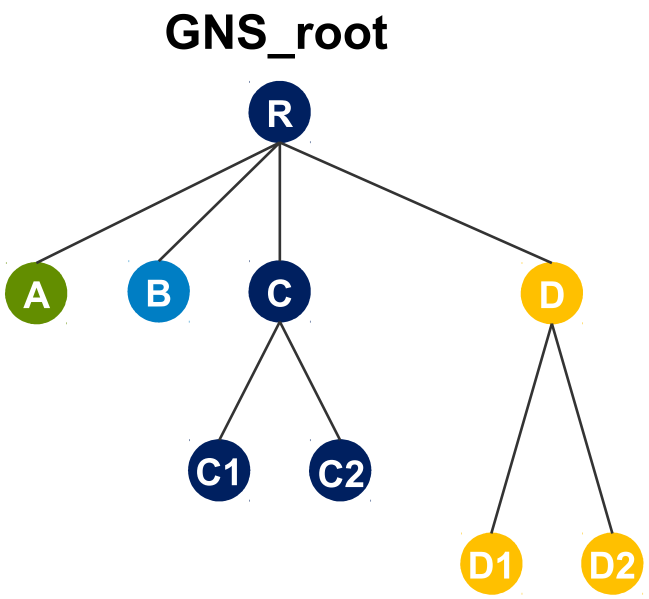
D-Blades

Aggregates

Volumes

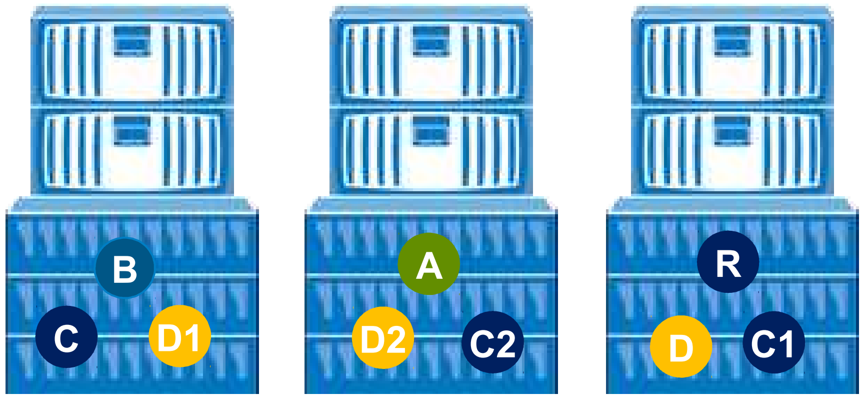


How the Namespace works?

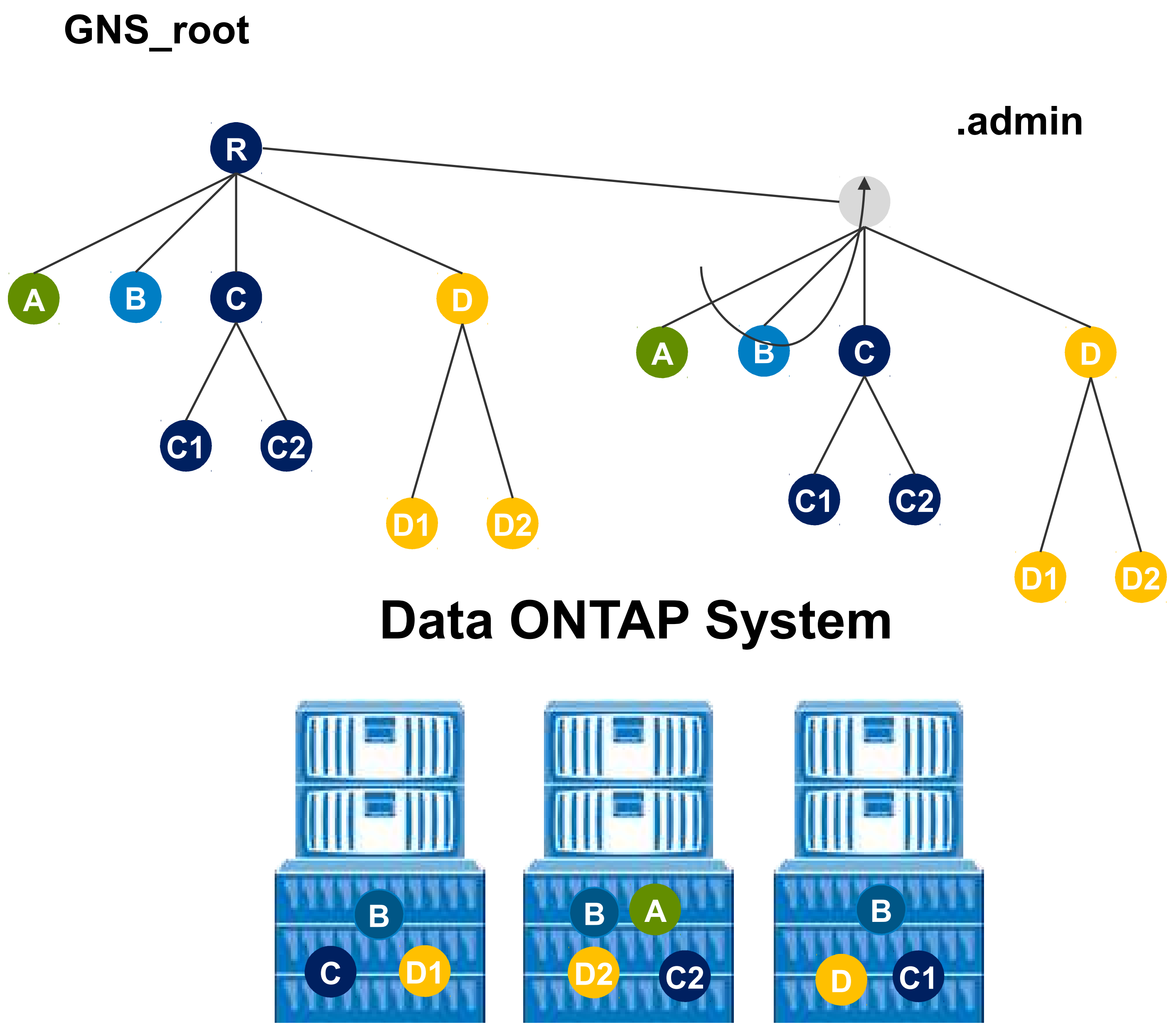


- NetApp in HA Pairs
- Partners within a pair use Infiniband interconnect
- Pairs connect to each other over Ethernet.

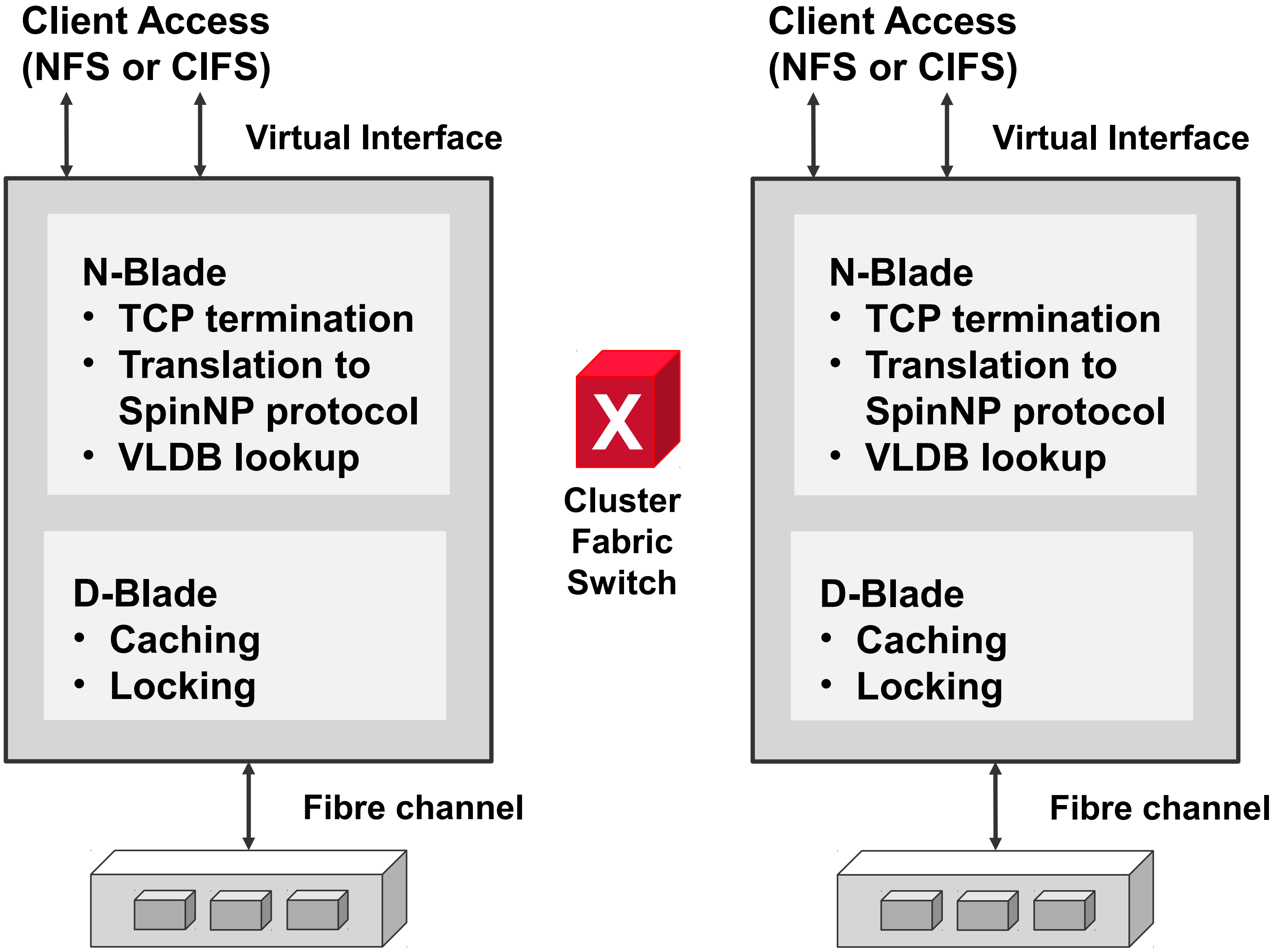
Data ONTAP System



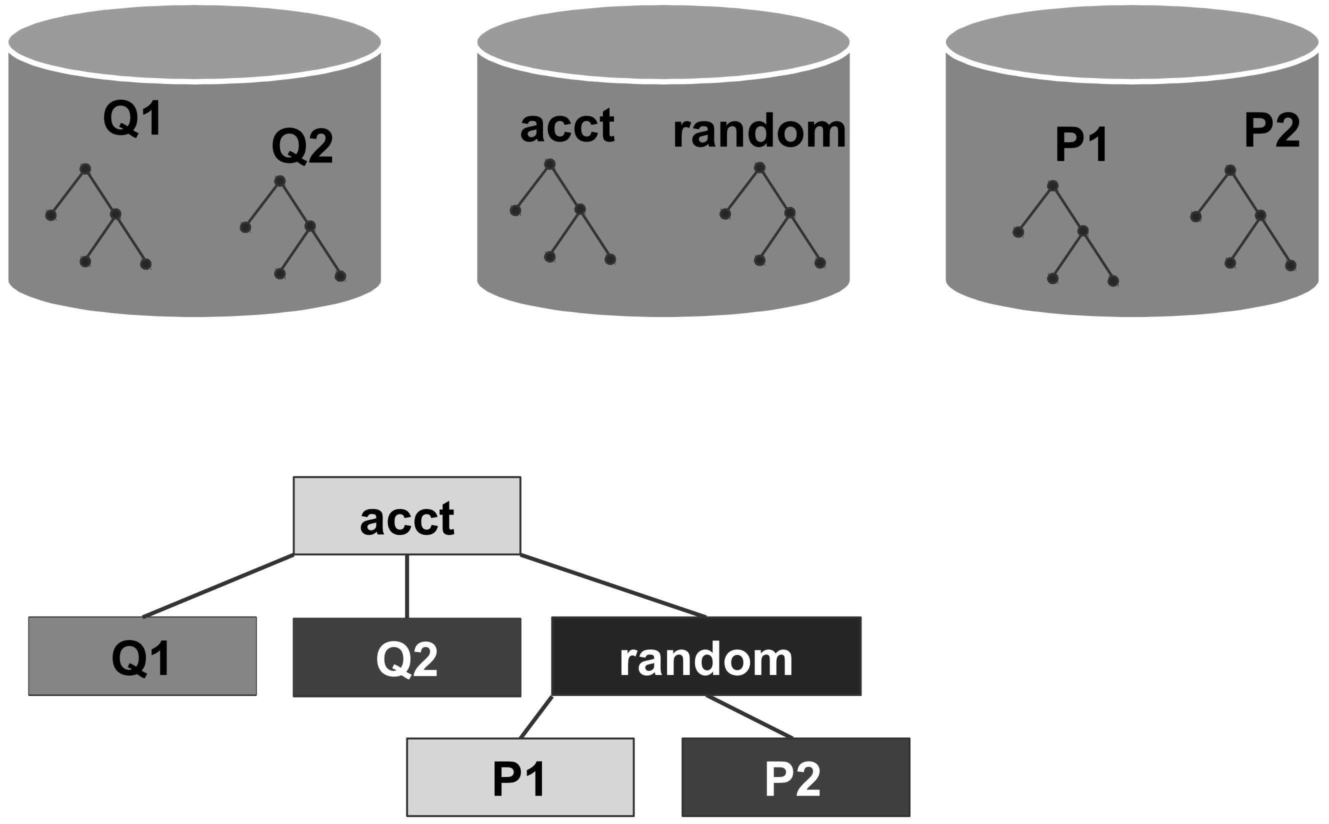
How Load Sharing Replication Works



Basic Switched Architecture



Namespace Construction



This deck was developed from information found here

http://datacenterdude.com/netapp/storage-dawn-part-2

I’d like to give this a name. Let’s call DataONTAP the Storage Hypervisor.

DataONTAP has enabled the ability to centralize on a platform. Standardize across your entire storage infrastructure on one skillset, commandset, and if you’re using our OnCommand Unified Manager and accompanying suite of tools, all arrays can be managed from a single extensible interface, similar, again, to VMware’s vSphere Client.

So, this is where it starts getting interesting.

Let’s take what we know so far about this unified operating system, and apply some virtualization techniques to it.

Glad you asked. Think about constructs within a virtual datacenter. Think about Virtual Machines. Think about vNetworks. Think about vMotion, HA, and DRS. Cool stuff, right?

What if we could do the same thing in storage?   If we continue to expand on the previous images, it would look something like this

Not quite the whole story but showing the access points to the varrays

So let's talk about these vServers for a minute. vServers are the theoretical equivalent of a virtual machine. Each is running what is a semblance of DataONTAP code, but it's not a full install. It's simply a construct that lives in memory of all controllers in the cluster, and it has logical network interfaces ("LIFs") that connect to the underlying hardware on the backend, up to the app/infrastructure on the front end, and can even have their own mgmt interfaces for delegated control. It is safe to think of them as logical storage controllers. Commands can be either executed directly on them, or from the cluster context itself. It's all about RBAC and delegation. If you're a one-man wrecking crew, you'll likely spend most of your time at the cluster interface. But for you storage admins that delegate control to storage resources to other departments, you can now delegate control of an entire vServer (with pre-appropriated storage) to an individual or group.

The difference between a traditional virtual machine and a vServer mostly has to do with the fact that a traditional virtual machine is a set of files. vServers in DataONTAP live in memory, and have a 20MB "root" volume on the storage controller they were created on. This basically holds core configuration information, such as metadata, network/LIFs, etc. That's why it's only 20MB.

So, we've now got our vServer's, commoditized storage clusters, LIF's or logical networking.

Now we’re talkin. A true imagery of the storage hypervisor. In the image, the hardware platforms are technically represented by the yellow rectangle in the background. Why all the different color disks? Well, it’s a correlation of two things:

1) Differing drive types & speeds
2) The apps being housed on those drives

The reality is, this is how customers will see it. That’s what is important to the customer.

“Where is my data?”
“What volumes are they on?”
“What arrays are those volumes on?”

With that said, let’s keep building out this big picture

There’s those colours we were looking at before. Now it all starts to come together and make sense, doesn’t it?

What you end up with is one giant logical pool of “cloud storage,” that is only segregated by drive type. This logical pool (or pools) is used to carve out volumes that will be used to serve data to your apps/infrastructure, and those volumes can seamlessly be moved around between the drive types, or between hardware controllers. Live! Hot! On-the-fly!

Sound familiar, VMware admins? [HINT: vMotion]

We call it DataMotion. I tried to get them to change it to “nMotion” at the last minute, but it didn’t take. :)

The stage is now set.

LIFs. Let's talk about these LIFs a little more. This is one of the hardest concepts for most people to grasp. But the reality is, it's just another logically abstracted layer of management and segregation. As I mentioned earlier in my definition, a LIF is a per-node, per-protocol logical interface. It's just software. Traffic cops, if you will. Their sole purpose in life is to connect clients to their data, regardless of where it is currently residing on the cluster.

If VMware's VMs were to span entire clusters of ESXi hosts, we could draw some closer correlations between the two, but essentially, we're doing for storage what VMware has done for servers [and more]. We could equate the volumes and LUNs, and their mobility to an svMotion, but we don't actually move the Vservers, we just re-home the LIF ports of the data paths.

In the next post, we'll continue the conversation to discuss Junction Paths and Export Policies. Export Policies?! Curious, isn't it. :)

As always, comments welcome, and it gives me things to add into followup posts as well! Thanks for reading! Let me know what you're excited about, and if there's anything that isn't clear enough, or that you need further detail on, please don't hesitate to speak up!

-Nick

Source: Images designed by my teammate, Peter Learmonth of NetApp, and were used with his permission. Images may not be re-used without express written consent.

Request processes divided between the network facing N-Blades and disk facing D-Blades
N & D Blades are both software kernel modules which may run on the same hardware nodes
M-Hosts provide management for the Cluster

N-Blades
- Terminate client transport connections & Sessions
- Authenticate and authorise clinets and users
- Route requests to correct D-Blade
- Route responses to correct client
- Are nearly stateless and cacheless to ease moving of Vservers between N-Blades

D-Blades
- Store all persistent filesystem state
- Manage lock state and enforce ACL
- Maintain local single instance data and metadata cahces
- Manage local filesystem instances as shared nothing stores
- RAID storage stack

Volumes
- Stored in aggregate

D-blades each store data volumes
– D-blades store all persistent file system state
– Manage lock state
– Enforce ACLs
– Maintain local single-instance data and metadata caches
– Manage local filesystem instances as shared-nothing data stores
Multiple volumes are stored in an aggregate
– Aggregate is a collection of one or more RAID groups
– Volumes are virtualized within an aggregate
Each D-blade can control multiple aggregates at any
time
-blade also handles RAID and storage stacks

MSIDs (Master Data Set Identifiers) identify a
group of mirrored volumes
–MSIDs are present in file handles handed to clients
–Uniquely specify a version (current or snapshot) of
a set of mirrored volumes
DSIDs (Data Set Identifiers) identify a single volume
–DSIDs are present in internal file handles presented
by N-blades to D-blades
–Uniquely specify a version (current or snapshot) of
a single volume

Each volume is a virtualized container storing a portion of file system namespace that descends from a single root directory

Volumes are linked together through junctions

Junctions

–May appear anywhere in a volume

–Link to the root of another volume

–May point to a volume on a different D-blade in the cluster

–Look like directories to the client

• Client does not see a referral across a junction

A GX cluster exports one or more VServers

–Often many more

Each VServer presents its own independent namespace

–Rooted at a separate root volume

Each Vserver has its own virtual interfaces (vifs)

–A vif is a network endpoint (IP address)

Vifs can migrate among N-blades

Volume B has read-only load sharing mirrors

MSIDs (Master Data Set Identifiers) identify a
group of mirrored volumes
–MSIDs are present in file handles handed to clients
–Uniquely specify a version (current or snapshot) of
a set of mirrored volumes

DSIDs (Data Set Identifiers) identify a single volume
–DSIDs are present in internal file handles presented
by N-blades to D-blades
–Uniquely specify a version (current or snapshot) of
a single volume

Each mirror has the same MSID, but different DSID

N-blade maps MSID to one of DSIDs and routes to D-blade

Volume B has one read/write master
– Same MSID, but different DSID

Master accessible through /.admin

Clients are distributed across a set of virtual interfaces (VIFs) at the front end of the storage "nodes" in the scale-out configuration (i.e. scale-out "cluster").

A VIF is the Cluster Mode representation of an IP/MAC address combination that provides client network access. At time of system configuration,

VIFs are "bound" to physical storage controller networking ports. That binding can be dynamically updated, either manually, or automatically, in response to a network port or controller failure, or as part of standard storage cluster expansions.

Such port migrations are nondisruptive to the client applications. A client connects to the storage system through a single VIF at time of mount.

Mapping of clients to VIFs can be done by any of a number of means, including static (partitioned) assignments, DNS roundrobin, or through Level4 load balancing switches.

The volumes that are physically distributed across the individual D-Blades are “stitched together” into a global namespace by connecting them in to a directory in a “parent” volume at time of creation.

When a client request is made to the N-Blade, a quick, in-memory lookup is done (in the VLDB, or volume location database) to determine which node has the data for the requested file, and the request is “routed” (through the cluster fabric switch) to the appropriate D-Blade.

(Think of the VLDB as a “volume router” — analogous to a network router, which caches its maps in memory for *very rapid* switching).

The target D-Blade makes the appropriate storage request, and returns the necessary information to the N-Blade, which then returns it to the client. This all occurs over the SpinNP protocol, a lightweight RPC protocol engineered from scratch by the Spinnaker team to ensure low latency and efficiency of storage network operations