



Chip Coldwell
Senior Software Engineer, Red Hat

Classical Storage Stack: the top layer

■ File system

- examples: ext3, NFS, GFS
- built on Linux VFS (“Virtual File System” abstraction)
- defines on-disk structure: how blocks are organized into files and directories.
- Important data structures: `dentry` (directory entry), `inode` (index node)
- Important concepts: “extents”, `fsck`, journaling
- remote file systems, like NFS, are based on RPC, not block devices, but consider iSCSI: a remote block device.
- we won't discuss file systems any more today.

Classical Storage Stack: the middle layer

■ Block devices

- A block device is distinct from a character device in that data are accessible one block at a time instead of one byte at a time.
 - Block sizes vary from device to device, but pages are architecture specific (4096 bytes on i386 and x86_64).
- Access is usually mediated by the page cache: we only go to magnetic media when we have to, and when we do, we cache the result in main memory.
 - The page cache will grow to consume all of available memory, and dirty page write back begins when allocations would otherwise fail.
 - You can bypass the page cache with direct I/O, which can be useful for backups, etc. that would unnecessarily dirty a lot of page cache.

Classical Storage Stack: the middle layer

■ Block devices

- Important concepts: request queues, I/O scheduler (no-op, anticipatory, deadline, CFQ), plugging and unplugging
- Important data structures: bio, request
 - A bio is submitted to the I/O scheduler, which either creates a new request or merges it into an existing request. Block devices see requests.
- This is the layer where device mapper (LVM, multipath) and multiple devices drivers are inserted, more on this soon.
- Familiar examples of block devices
 - named device nodes such as `/dev/sda`
 - `/dev/mapper/VolGroup00-LogVol100`
 - **symlinks** `/dev/disk/by-id/scsi-3600601601040190000c7da47f286dc11`

Classical Storage Stack: the bottom layer

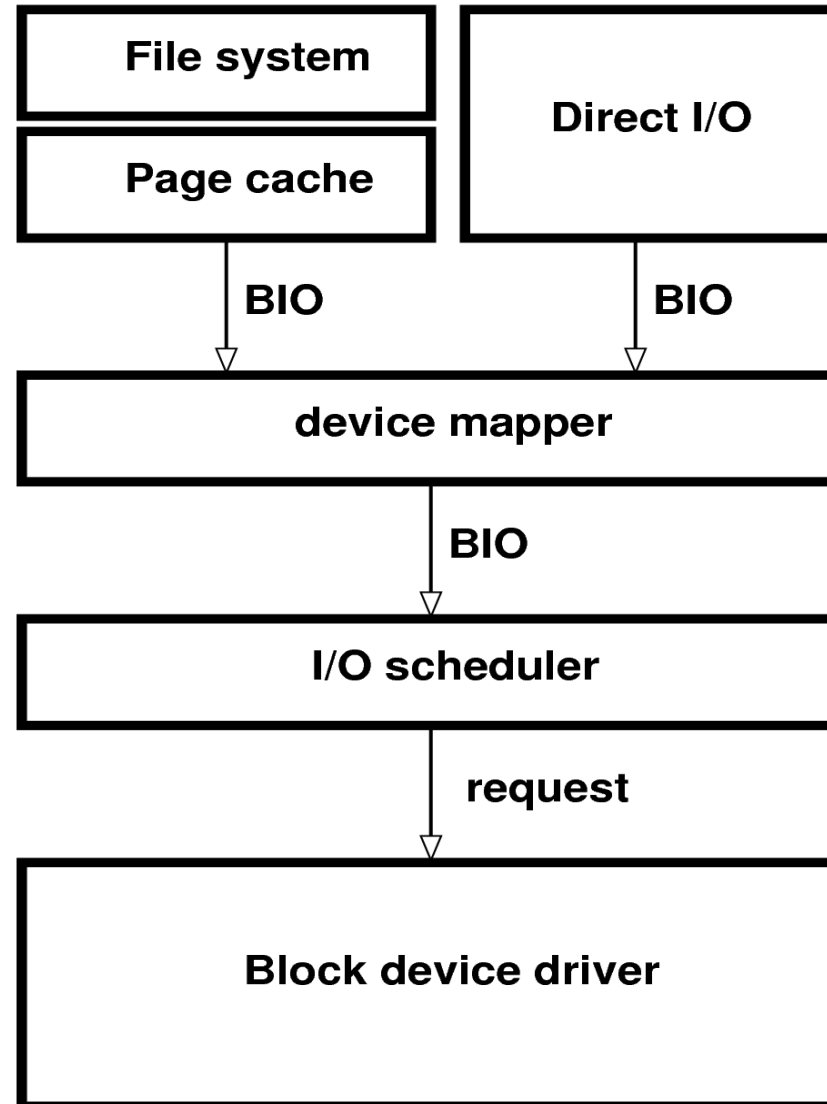
■ Device drivers

- This is where the bits hit the medium
- Usually factored into a mid-layer and low-level device driver (LLDD)
 - mid-layer: `scsi_mod`, `scsi_transport_fc`
 - LLDD: `qla2xxx`, `mptsas`, `lpfc`, etc.
- Very hardware specific, usually locked to a set of PCI IDs
- `/sys/class/scsi_host/hostN`: contents are very driver dependent, unfortunately

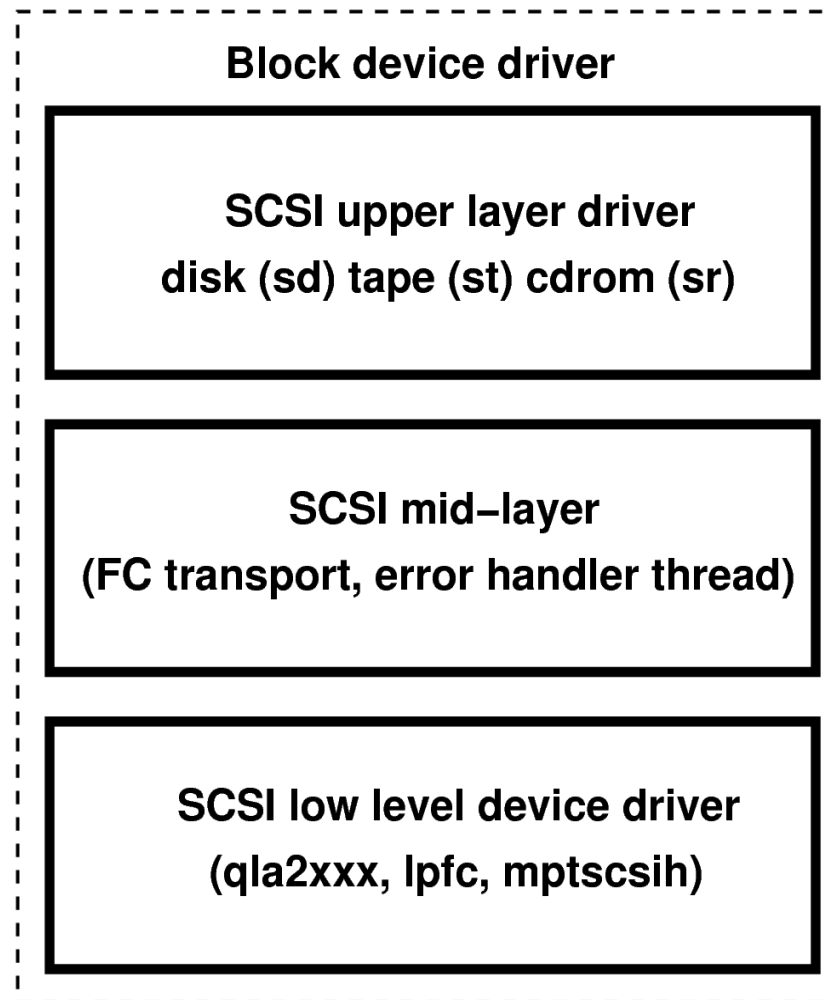
Device-mapper and MD-RAID

- These are inserted in the middle layer (block devices)
 - Both device-mapper and MD-RAID can do RAID1 (mirroring)
 - Device-mapper can also do LVM
 - md-RAID can also do RAID-0, RAID-5, etc.
- Both are virtual block devices
 - Bypass the elevator algorithm and I/O scheduler by replacing `__make_request` with their own make-request functions.
 - Remap block I/Os from the virtual device to the physical devices.
 - Unfortunately, because these operate at the block device level, some error information is lost, e.g. fine grained SCSI error codes get converted to -EIO by the SCSI midlayer. This is especially a problem for dm-multipath due to the “fail fast” flag.

- A simplified diagram of the block subsystem



- Layers within a SCSI block device driver



On line reconfiguration of virtual block devices

- This has always been a goal of the design of LVM
- snapshots, adding, removing and extending physical volumes, volume groups and logical volumes
- LVM interfaces with the kernel via device-mapper, and is a virtual block device like MD-RAID. This provides abstraction layers between LVM and the physical device.

Example: resize an ext3 file system

- Create and mount a 20G ext3 file system
 - Multipath SAN with three LUNs, device IDs
 - 3600601601040190000c7da47f286dc11
 - 360060160104019008e275c5af286dc11
 - 36006016010401900be062253f286dc11
 - In `/etc/multipath.conf`, `user_friendly_names no`
 - `pvcreate /dev/mapper/36*`
 - `vgcreate san0 /dev/mapper/36*`
 - `lvcreate -L 20G san0`
 - `mke2fs -j -L SAN0 /dev/san0/lvol0`
 - `mount LABEL=SAN0 /test`

Example: resize an ext3 file system

- Now unmount the filesystem and resize the underlying logical volume
 - `umount /test`
 - `e2fsck -f /dev/san0/lvol0`
 - `lvextend -L 30G /dev/san0/lvol0`
 - `resize2fs /dev/san0/lvol0`
 - `mount LABEL=SAN0 /test`

Example: resize an ext3 file system

■ We can also shrink the filesystem

- `umount /test`
- `e2fsck -f /dev/san0/lvol0`
- `resize2fs /dev/san0/lvol0 20G`
- `lvreduce -L 20G /dev/san0/lvol0`
- `mount LABEL=SAN0 /test`

■ Some bad news here

- had to unmount the filesystem!
- had to fsck!
 - Feeling lucky? run `resize2fs` without `umount` or `fsck`: works for extending but not for shrinking the file system.

Example: resize an ext3 file system

■ Postmortem

- The example used an ext3 file system, but would work for any file system that provides a utility analogous to `resize2fs`.
- The important point is that because LVM lives entirely in the middle layer of the storage stack, it is agnostic about both the file system above it and the block device below it.
- In fact, in this particular example, the logical volume is a virtual block device (managed by device-mapper in the kernel) sitting on top of another virtual block device (the multipath device to the SAN, also managed by device-mapper).
 - Holy nested abstraction barriers, Batman!

Adding a SCSI LUN in a fibre channel SAN

- A quick dip into sysfs
- `/sys/class/fc_host/hostH/(port_name|node_name)`
 - 64 bit port WWN
 - shows up in SANsurfer, Navisphere, vendor utilities
 - (sometimes as 128 bit node_name:port_name)
 - can be used for zoning, allocating LUNs, etc
 - If a LUN is allocated to a port_name:node_name, then Linux will show the associated SCSI disk device as being attached to that host
 - In the H:B:T:L nexus, we now know “H”
 - `lsscsi` can show us all the devices connected to that host.

Adding a SCSI LUN in a fibre channel SAN

- `echo yes >/sys/class/fc_host/hostH/issue_lip`
 - Forces the HBA to issue a LIP (Loop Initialization Protocol)
 - Re-scans the storage processor and adds new SCSI devices
 - In a multipath SAN, must repeat for all HBAs with paths to the LUN
- In a multipath environment, run `multipath -ll` to verify that device mapper has recognized the new device
- `pvcreate /dev/mapper/3600601601040190002e7ff73e23bdd11`
- `vgextend san0 3600601601040190002e7ff73e23bdd11`
- `lvextend -L +50G /dev/san0/lvol0`
- `resize2fs /dev/san0/lvol0`

Removing a SCSI LUN in a fibre channel SAN

- `umount /test`
- `e2fsck -f /dev/san0/lvol0`
- `resize2fs /dev/san0/lvol0 29G`
- `lvreduce -L -50G /dev/san0/lvol0`
- `vgreduce san0
/dev/mapper/3600601601040190002e7ff73e23bdd11`
- `echo yes >/sys/class/scsi_disk/H:B:T:L/device/delete`
 - **In a multipath environment, repeat for all paths**
- `multipath -f 3600601601040190002e7ff73e23bdd11`
- `mount LABEL=SAN0 /test`

Adding/Removing a SCSI LUN in a fibre channel SAN

- Always remember, in terms of the storage stack
 - Add from the bottom up
 - LUN, physical volume, volume group, logical volume, then filesystem
 - Remove from the top down
 - filesystem, logical volume, volume group, then LUN
- LVM is your friend: adding and removing LUNs is much less useful if you can't grow and shrink your file systems.

Device Names

- **Never, ever use** `/dev/sdX`
 - `/dev/sdX` names are generated dynamically when devices are scanned by the SCSI mid-layer, in the order in which devices are discovered
 - device discovery order is unpredictable, can be done in parallel.
 - If you add one device and remove another, it is unlikely that on the next reboot the device you added will be assigned the same name.
 - In a multipath environment, each `/dev/sdX` refers to one path, not one logical device.

Device names

■ So how to name devices?

- Names exist at every layer of the storage stack. Be careful which one you choose!
- File system labels
 - `mke2fs -L LABEL /dev/san0/lvol0`
- File system UUIDs
 - `/dev/disk/by-uuid` these are file system UUIDs
 - Can be read with `/lib/udev/vol_id /dev/san0/lvol0`
 - Physical volumes, volume groups and logical volumes also have UUIDs
- LVM names `/dev/volume-group/logical-volume`
 - symlinks into `/dev/mapper`

Device names

- So how to name devices? (continued)
 - SCSI VPD device identification (code page 0x83)
 - `scsi_id -g -p 0x83 -s /block/sda`
 - `3600601601040190000c7da47f286dc11`
 - alternatively: `sg_vpd --ident /dev/sda`
 - `sg_inq /dev/sda` will do a standard inquiry, possibly returning a device serial number
 - VPD page 0x83 is preferred, IEEE issued OUI
 - Note that symlinks in `/dev/disk/by-id/scsi-3600601601040190000c7da47f286dc11` are broken in multipath environments: point to most recently discovered `/dev/sdX` path to the LUN.

Device names

- So how to name devices? (continued)
 - In a multipath environment, you really want to use the VPD page 0x83 device identification. It's ugly, but
 - both storage and host agree on the ID
 - persistent across reboots
 - `/etc/multipath.conf` with “`user_friendly_names no`”
 - For mounting file systems, use file system labels or UUIDs
 - Never, ever put `/dev/sdX` in `/etc/fstab`

Introducing scsitol

- command-line utility somewhat similar to fcinfo on Solaris
- currently supports fibre channel, working on iSCSI
- Can query on several properties
 - host:bus:target:lun nexus, and regular expressions
 - device name (e.g. sda, sdx, etc)
 - WWID
 - target/host WWNN/WWPN
 - vendor or model
 - HBA device driver

Introducing scsitol (continued)

- similar to Doug Gilbert's lsscsi and sg3_utils, but
 - lsscsi runs unprivileged, hence can't query VPD page 0x83 to get disk WWID
 - scsitol always sorts its output by key, in the order the keys were given on the command line
 - scsitol allows the user to select output based on device properties, and only show matching devices

Introducing scsitol (continued)

```
root@dl585-03:~ (on dl585-03.lab.boston.redhat.com)
[root@dl585-03 ~]# scsitol --hbt1 --devname --vendor --model
1:0:0:0 sda DGC RAID 5
1:0:0:1 sdb DGC RAID 5
1:0:0:2 sdc DGC RAID 5
1:0:1:0 sdd DGC RAID 5
1:0:1:1 sde DGC RAID 5
1:0:1:2 sdf DGC RAID 5
1:0:2:0 sdg DGC RAID 5
1:0:2:1 sdh DGC RAID 5
1:0:2:2 sdi DGC RAID 5
1:0:3:0 sdj DGC RAID 5
1:0:3:1 sdk DGC RAID 5
1:0:3:2 sdl DGC RAID 5
3:0:0:0 sdm DGC RAID 5
3:0:0:1 sdn DGC RAID 5
3:0:0:2 sdo DGC RAID 5
3:0:1:0 sdp DGC RAID 5
3:0:1:1 sdq DGC RAID 5
3:0:1:2 sdr DGC RAID 5
3:0:2:0 sds DGC RAID 5
3:0:2:1 sdt DGC RAID 5
3:0:2:2 sdu DGC RAID 5
3:0:3:0 sdv DGC RAID 5
3:0:3:1 sdw DGC RAID 5
```

Introducing scsitol (continued)

```
root@dl585-03:~ (on dl585-03.lab.boston.redhat.com) [root@dl585-03 ~]# scsitol --wwid=3600601601040190000c7da47f286dc11 --devname 3600601601040190000c7da47f286dc11 sda 3600601601040190000c7da47f286dc11 sdd 3600601601040190000c7da47f286dc11 sdg 3600601601040190000c7da47f286dc11 sdj 3600601601040190000c7da47f286dc11 sdm 3600601601040190000c7da47f286dc11 sdp 3600601601040190000c7da47f286dc11 sds 3600601601040190000c7da47f286dc11 sdv [root@dl585-03 ~]#
```

Getting scsitol (alpha)

- `rpm -ivh`
<http://people.redhat.com/coldwell/coldwell-release-0-1.noarch.rpm>
- `yum install scsitol`
- report bugs to me (coldwell@redhat.com) not the Red Hat bugzilla!