



# Logical Volume Manager (LVM2)

Heinz Mauelshagen  
Consulting Development Engineer



# Top@sage

Project Motivation

Requirements

Software Architecture

User Interface

Metadata Locations and Backup/Restore

Logical Volume Snapshots

Project Status/Summary

URLs

# Motivation = Storage Virtualization

Virtualisation of mass-storage is (not only) mandatory in data centers to minimize systems downtime

Main areas covered by storage virtualisation:  
deploy logical rather than physical storage  
flexibility with regards to varying capacity needs  
fully resizable storage pools  
online data relocation to deploy newer, faster  
or more resilient storage subsystems  
optimized layouts (eg, mirrored, striped, ...)

Not available in Linux back in 1997 ->

LVM1 project start followed by the LVM2 project in 2001



## Requirements (1)

CLI close to HP-UX LVM because of the ease of use and quick learn curve

Virtualisation designed as a simple 3 Layer-Model

Storage devices (aka Physical Volumes or PVs) are grouped together in...

Volume Groups (VGs; “logical disks”); total capacity of a VG is slightly less than sum of all PV capacities because metadata needs to be stored on PVs

VG-Capacity allocated dynamically to Logical Volumes (LVs; “logical Partitions”; /dev/VGName/LVName)



## Requirements (2)

### Functionality

Create PVs, VGs and LVs (linear, striped, ...)

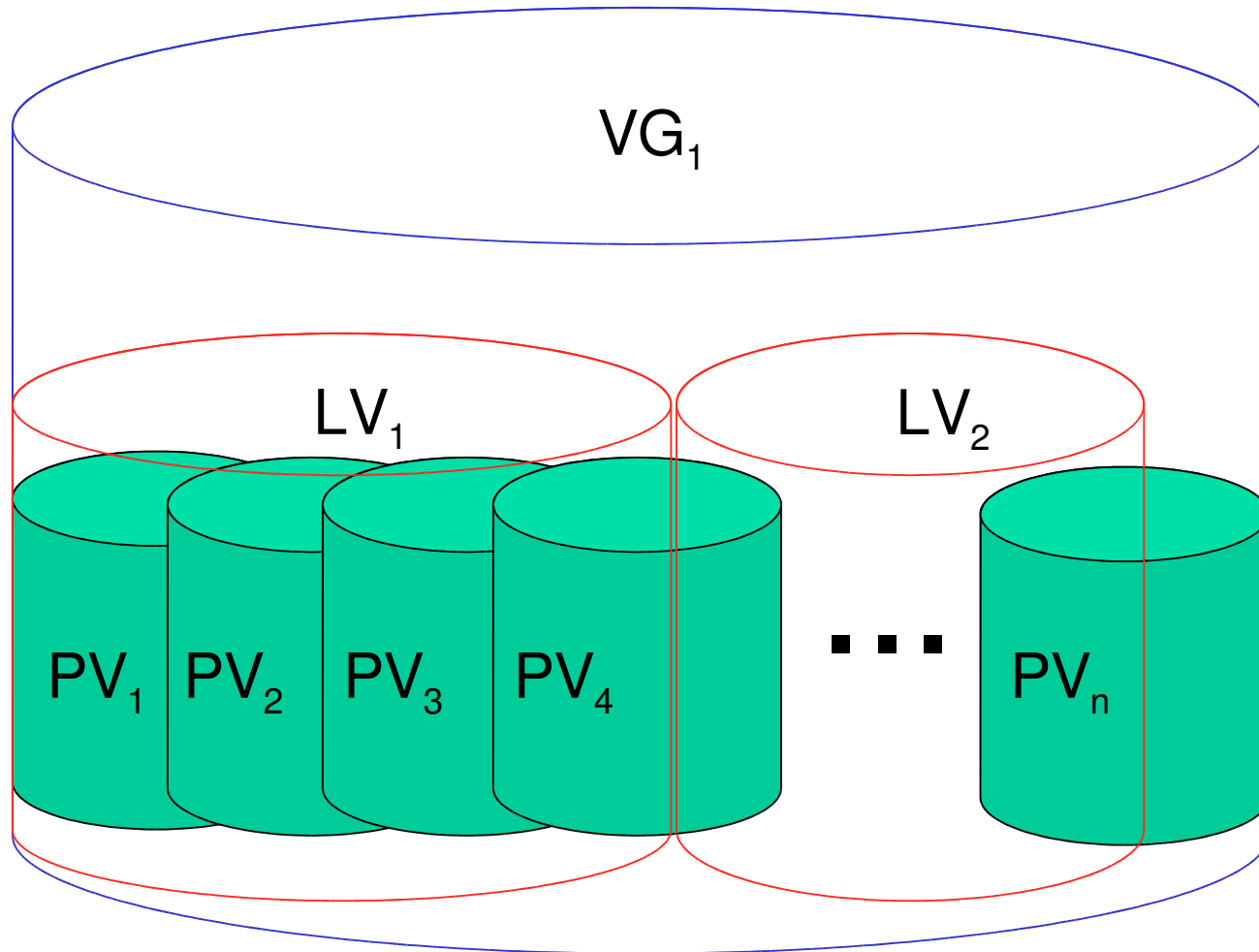
Grow and shrink VGs and LVs online

Display attributes of PVs, VGs und LVs

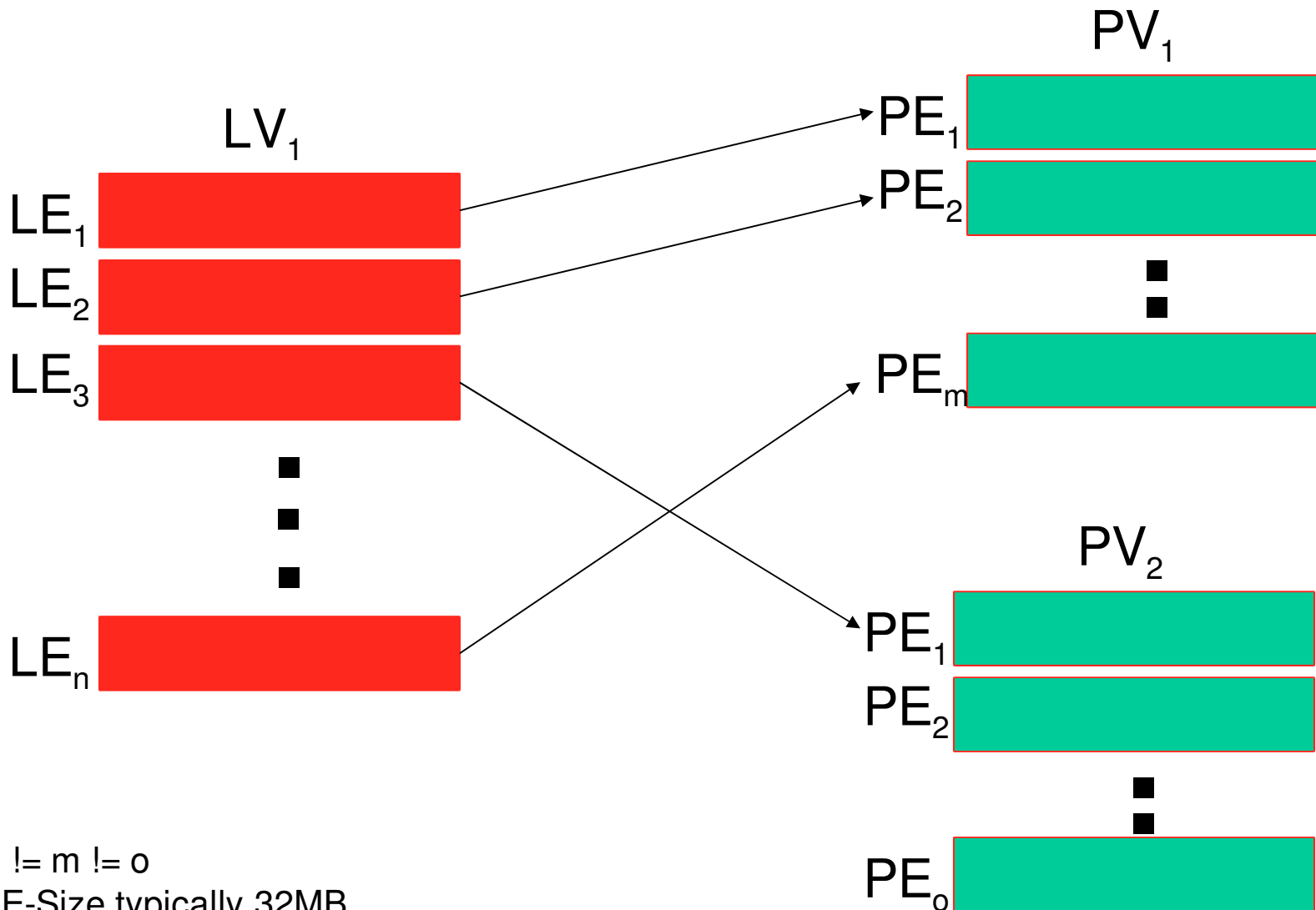
Relocate (parts of) LVs online

-> **Large flexibility to adjust to varying storage capacity and quality requirements!**

# Virtualisation: 3 Layer-Model



# Mapping: logical -> physical



$n \neq m \neq 0$   
LE-Size typically 32MB

# LVM2 Software-Architecture (1)

Device-Mapper as the generic mapping runtime platform in the Linux kernel (since 2.5)

can be used by multiple applications which need block device mapping (eg, dmraid, EVMS or Multipath Tools) manages Mapped Devices (MDs) (create, remove, ...) handles text formatted Mapping-Tables (load, unload, reload) all table loading actions happen online to reflect changed layouts (eg, a Mapped-Device size change) while the device is suspended for data integrity reasons code to handle mappings factored out into Mapping-Targets, which handle linear, striped, mirrored, snapshot, multipath, zero, error and crypt layouts (interface: constructor, destructor, map, ...)



## LVM2 Software-Architecture (2)

Device-Mapper continued...

Mapping-Targets are dynamically loadable plugins and register with the Device-Mapper core

Mapped Devices can be stacked in order to build complex mappings (eg, to enable pvmove via a temporary mirror)

no 256 Limit on PVs, VGs or LVs any more

(tested with thousands of LVs)

theoretically up to 8 EB per LV in Linux 2.6 (CONFIG\_LBD);

max size on 32 bit archs limited to 16 TB because of VM constraints

comes with a user space library to be interfaced by

Volume Management applications and a test tool dmsetup

lib creates nodes to access Mapped Devices in /dev/mapper/



## LVM2 Software-Architecture (3)

Examples of Device-Mapper tables which can be activated using the dmsetup tool:

```
0 1024 linear /dev/hde1 40
```

```
1024 2048 striped 2 64 /dev/hde1 1064 /dev/hdf1 0
```

```
0 1024 zero
```

```
1024 1000 error
```

```
0 83886080 mirror core 1 64 2 /dev/sda 0 /dev/sdb 0
```



## LVM2 Software-Architecture (4)

LVM2 Application to manage ondisk metadata of the PVs, VGs und LVs

user configurable with regards to device nodes to use/ignore (Device Name Filter), logging of runtime information and more in `/etc/lvm/lvm.conf` (`/etc/lvm/` can be changed in environment variable `LVM_SYSTEM_DIR`)

keeps list of used (filtered) block device names in a cache file `/etc/lvm/.cache` for performance reasons

transforms between ondisk metadata and internal unified representation of PVs, VGs and LVs

reads+writes LVM1 and new LVM2 (redundant metadata copies, transaction oriented updates) formats other metadata format handlers can be added



## LVM2 Software-Architecture (5)

LVM2 Application to manage ondisk metadata of the PVs, VGs und LVs

bidirectional migration of LVM1 <-> LVM2 metadata  
(subject to LVM1 metadata constraints)

stores one metadata backup per VG automatically in  
`/etc/lvm/backup/`

stores multiple metadata archives per VG automatically in  
`/etc/lvm/archive/` (metadata history of changes)

## LVM2 Software-Architecture (6)

other LVM2 Application functional areas

- create the Device-Mapper tables for the Mapped Devices

- create symlinks from `/dev/VGName/LVName` to the

- Device-Mapper nodes in `/dev/mapper/`

- log information

- interactive LVM2 shell (eg, optimization of bulk updates)

## LVM2 Software-Architecture (7)

Summary of directories and files used by LVM2:

/etc/lvm/lvm.conf:

central config file read by the tools

/etc/lvm/.cache:

device name filter cache file (configurable)

/etc/lvm/backup/:

directory for automatic VG metadata backups (configurable)

/etc/lvm/archive/:

directory for automatic VG metadata archives  
(configurable WRT directory path and archive history depth)

/var/lock/lvm:

lock files to prevent parallel tool runs from corrupting  
the metadata



## User Interface

CLI directly derived from the 3 layer-model (PV, VG, LV) and the actions to perform (change, create, remove, extend, reduce, rename, scan, s(how), display, move, split, merge)

A lot of commands are simply named by prefixing the actions to perform with pv, vg or lv:

pvchange, pvcreate, pvremove, pvdisplay, pvmove, pvscan, pvs  
vgchange, vgcreate, vgrename, vgdisplay,  
vgextend, vgreduce, vgscan, vgimport, vgexport,  
vgsplit, vgmerge, vgs  
lvchange, lvcreate, lvremove, lvdisplay,  
lvextend, lvreduce, lvscan, lvs

# User Interface

Other supporting commands are:

fsadm

lvmdiskscan

lvm dumpconfig

lvresize

vgcfgbackup, vgcfgrestore, vgck, vgconvert, vgmknodes

All commands take `-v`, which can be entered multiple times to increase the output verbosity

'`commandname --help`', '`man lvm`' and

'`man commandname`' are your friends :-)



## Commands (Physical Volume)

pvcreate:

initializes a block device to be used as a PV, which is a prerequisite to get included in a VG

pvremove:

zeroes the LVM2 metadata area on an **empty** PV

pvdisplay:

display PV properties (eg. size, extents, VG) in a fixed format

pvchange:

change attributes of the PV (eg, allocation)

## Commands (Physical Volume)

`pvmove`:

moves data off a PV to (an)other PV(s) online; uses temporary mirror to be restartable after a system failure

`pvscan`:

scans all filtered (`lvm.conf`) devices for PV metadata and lists hits

`pvs (new)`:

format selectable, flexible output of PV properties;

powerful for scripting (eg, `pvs -o pv_name,pv_size -Opv_free`)



## Commands (Volume Group)

`vgcreate:`

creates a new VG by name and adds at least one PV to it

`vgremove:`

removes an empty VG (with no LVs in it)

`vgdisplay:`

display VG properties (eg. size, extents, # of PVs) in a fixed format

`vgchange:`

(de)activates (partial) VG(s) and changes various attributes (eg, maximum # of PVs/LVs allowed, resizing)

`vgextend:`

grows a VG's capacity by adding one or more free PV(s)



## Commands (Volume Group)

`vgreduce`:

shrinks a VG's capacity by removing one or more empty PV(s) (eg, to use those in different VGs or remove them from the system)

`vgscan`:

scans all filtered (`lvm.conf`) devices for LVM metadata and stores all hits in the cache file, which is used by the other tools

`vgimport`:

makes a VG accessible on a machine after its PVs got attached

`vgexport`:

makes an inactive VG inaccessible In order to detach its PVs

## Commands (Volume Group)

`vgsplit:`

splits PV(s) of a VG creating a new VG; all LVs allocated to the PV(s) can't have extents allocated outside those PVs and no other LVs may have extents allocated on those (optionally use `pvmove` to force this)

`vgmerge:`

combine 2 VGs into one

`vgs (new):`

format selectable, flexible output of VG properties;  
powerful! (eg, `vgs -o vg_name,vg_uuid -Ovg_size`)



## Commands (Logical Volume)

lvcreate:

creates a new LV with layout (eg, striped) and size

lvremove:

removes an inactive LV

lvdisplay:

display LV properties (eg. size, layout, mapping) in a fixed format

lvchange:

change attributes of an LV (eg, allocation)

lvextend/lvreduce:

grow and shrink an LV; shrinking an LV makes VG capacity free to be allocated to other LVs in the same VG



## Commands (Logical Volume)

lvscan:

scan for all LVs in the system and list them

lvs (new):

format selectable, flexible output of LV properties  
(eg, `lvs -o lv_name,lv_attr -O-lv_name`)

## Commands (Supporting)

fsadm:

resizing of an LV **and** the filesystem therein in one go

lvmdiskscan:

scan for block devices that may be used as PVs

lvm dumpconfig:

displays the life config on stdout which can be redirected to a config file (eg, /etc/lvm/lvm.conf to pretty print it)

lvresize:

can be used instead of lvextend/lvreduce

vgcfgbackup:

creates metadata archives which are automatically created by default on every VG configuration change

## Commands (Supporting)

`vgcfgrestore:`

restores metadata of a VG from archive to all PVs.

May need a “`pvcreeate --uuid ...`” in case a PV can’t be found

`vgck:`

checks VG consistency

`vgconvert:`

converts LVM1 metadata format <-> LVM2

(backward convertability subject to LVM1 restrictions like  
maximum # of LVs < 256 in LVM1)

`vgmknodes:`

creates all devnodes for VGs and LVs

(very useful in case of accidental deletion)

## Metadata locations

LVM1 format is binary only and one copy is at the very beginning of every PV

LVM2 format is an ASCII text format which is at the beginning, after a disk label, of every PV in 2 copies by default

in large configurations, LVM2 metadata copies can optionally be avoided on PVs (`pvcreate --metadatascopies 0 /dev/..`) to have faster metadata updates on configuration changes

`vgconvert` is capable to inplace migrate the binary LVM1 format to LVM2 text format and back

## Metadata backup and restore

metadata needs to be included in your daily backup!  
metadata backups and archives are automatically created on every VG/LV configuration change unless disabled (unwise but space saving!) in `lvm.conf`  
manual backups of metadata to `/etc/lvm/backup/` can be enforced with `vgcfgbackup`  
restores of metadata backups or archives can be done with `vgcfgrestore`, which looks at all devices in the device name cache for the PV-UUIDs stored in the backup/archive and restores to those found  
if `vgcfgbackup` fails to find one or more appropriate PVs by UUID, a changed device name filter or an overwritten PV-UUID can be the reasons (adjust device name filter or `pvcreate -u ...` the respective device)



## Online data relocation

uses a temporary mirror which gets reactivated in case the system reboots/crashes

to move all allocated space (extents) off /dev/sdc to some other free PV in the VG:

```
pvmove /dev/sdc
```

to move just those extents of a particular LV off /dev/sdc:

```
pvmove -n MyLV /dev/sdc
```

to move all extents allocated to /dev/sdc over to /dev/sdf1 in the background:

```
pvmove -b /dev/sdc /dev/sdf1
```

# Logical Volume Snapshots

LV snapshots are references to the data image of another LV (called the origin) at snapshot creation time.

IOW: snapshots are (read+write) 'frozen' images of origins  
full read+write access to the origin stays possible

Snapshots don't need the amount of storage allocated to the origins additionally (space optimization).

Eg, with a rarely updated origin, 3-5% of the origins capacity is sufficient; in case a snapshot runs full, it gets dropped in order to avoid starvation of the origin

Snapshots are fully resizable to avoid that they get dropped or to free space which is not needed to be used by other LVs



# Logical Volume Snapshots

are most valuable for taking consistent backups, fsck'ing mounted filesystems in order to check if the origin is in urgent need of an fsck (filesystem journalling doesn't preclude full fscks!) and testing with production data without corrupting the origin preserve changing content of origins using a copy-on-write mechanism in their private storage called 'cow store' for performance reasons don't preserve every single changing block but preserve whole multiple block chunks at once on writes to the origin



# Status

LVM2 compatible to LVM1 with enhancements (eg, atomic metadata updates) in Fedora Core > 1 and RHEL4 (Spring 2005)

Multipathing with Device-Mapper in Linux 2.6  
(Ex-)Sistina GFS, CLVM and cluster infrastructure open-sourced end June 2004

# Summary

## Linux Logical Volume Manager

virtualizes mass storage; no restrictions to physical disk sizes any more  
enables online adjustments to varying storage requirements  
supports live data migration  
(ie. migrate data to new disk-subsystems)

**-> LVM shrinks the downtime of Linux systems by avoiding offline times for storage management which reduces operational costs**



## URLs

<http://sources.redhat.com/LVM2> (LVM2 tools+library)

<http://sources.redhat.com/dm> (Device-Mapper tool+library)

<http://christophe.varoqui.free.fr> (Multipath Tools)

<http://www.redhat.com/mailman/listinfo/linux-lvm>  
to subscribe for [linux-lvm@redhat.com](mailto:linux-lvm@redhat.com)

<http://www.redhat.com/mailman/listinfo/dm-devel>  
to subscribe for [dm-devel@redhat.com](mailto:dm-devel@redhat.com)

<http://www.tldp.org/HOWTO/LVM-HOWTO>

[mauelshagen@redhat.com](mailto:mauelshagen@redhat.com)

