VDO, the transparent deduplication/compression layer

July 2018 @tlug.jp

Christian Horn / chorn@fluxcoil.net

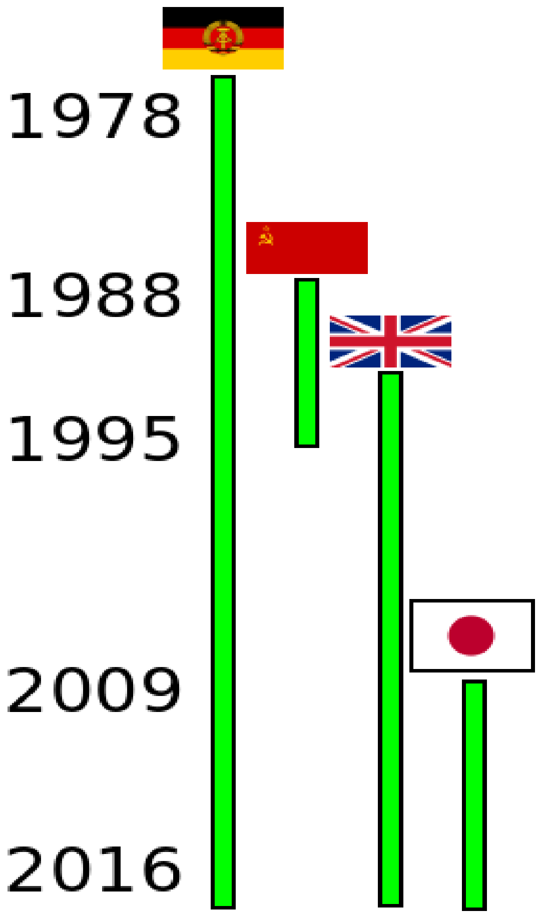
Agenda

- whoami
- The basics of VDO
- Use cases: where can VDO help?
- VDO setup and configuration
- How is VDO influencing read/write performance?
- How much storage space can I save for my use cases?

Let’s find out!

whoami

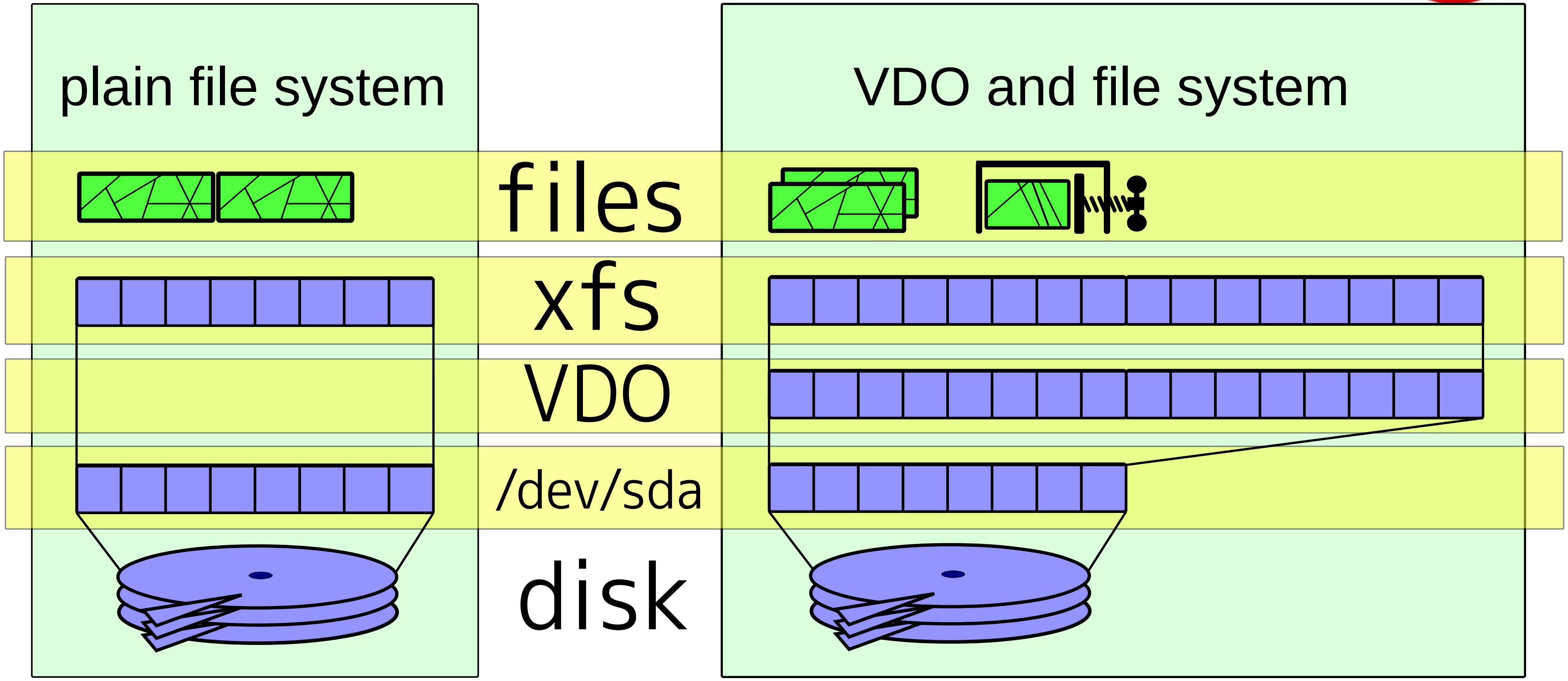
- Born in East Germany, 12 years before German unification
- 15 years in Munich, did many things around Linux. Worked 5 years as TAM at Red Hat Germany.
- Since 2016: AMC TAM at Red Hat Japan



Ever felt like you have to much storage?

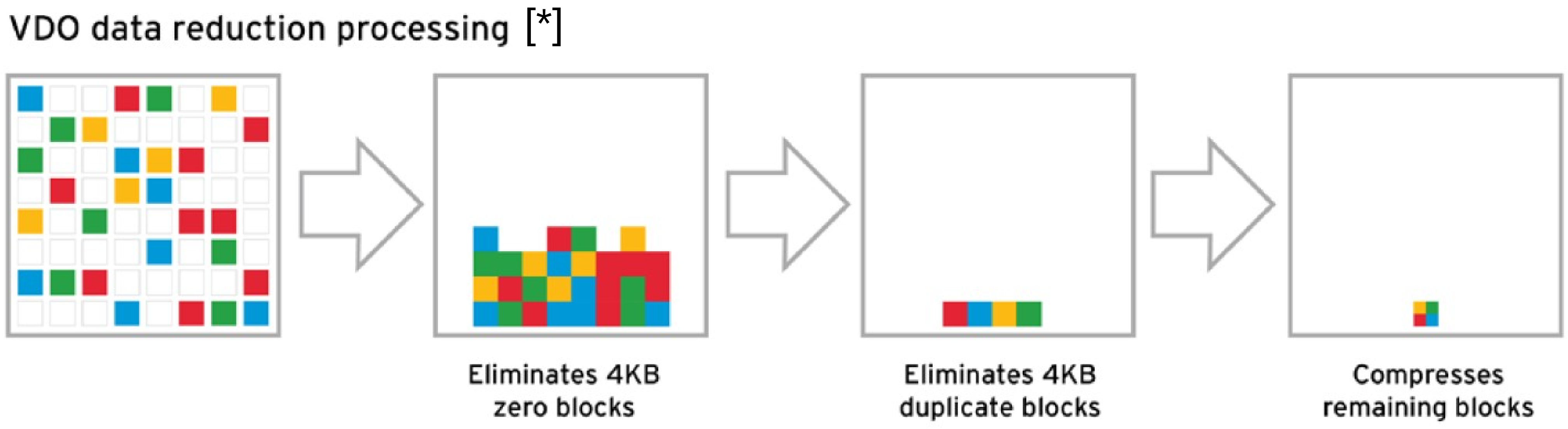
- Rather not, there is no ‘too much storage’
- Since a long time we use userland gzip and rar for compression – with Virtual Data Optimizer (VDO), RHEL7.5 got now a transparent compression/deduplication layer
- VDO comes from the Permabit acquisition 2017
  - code is available in source RPMs, but we are not in upstream → kernel gets tainted
  - upstream projects are getting established now (as per upstream first policy)
- PerformanceCoPilot (PCP)
  added VDO metrics in version 4.0.0 , in Feb. 2018.
  bz1565370 is open bringing it into RHEL7.6

The layers with VDO



- The plain system uses the disk directly as block device.

- VDO has typically a thin provisioned device
- Copies are mostly deduplicated
- Plus compression, removing zeros


VDO data reduction processing [*]
Eliminates 4KB zero blocks
Eliminates 4KB duplicate blocks
Compresses remaining blocks

(1) Unmodified data

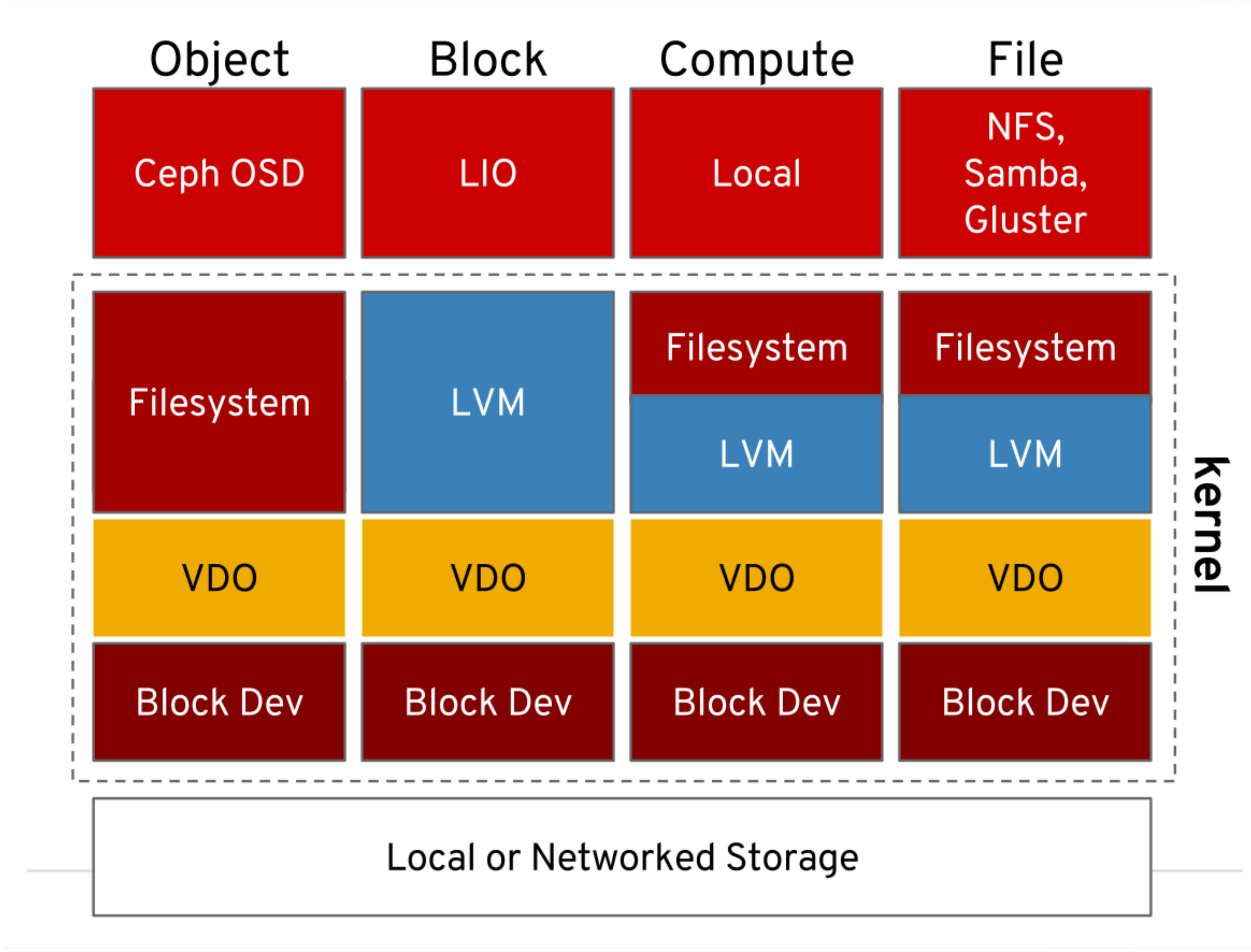
(2) Reduce zero blocks

(3) Deduplicate

(4) Compress with lz4

[*] graphic from http://permabit.com/cloud-economics-drive-the-it-infrastructure-of-tomorrow-2/

VDO and the system layers



Where is VDO useful?

- For example under local file systems, iSCSI or Ceph
- on file servers as base for local file systems, handing out NFS, CIFS or Gluster services
- Remember nfs-root? Dozens of Linux systems sharing read only NFS root file systems to save storage? You can now give each of these systems an own individual image via iSCSI, store then on a VDO backend, and have VDO deduplicate/compress the common parts of the images.

VDO installation.. easy!

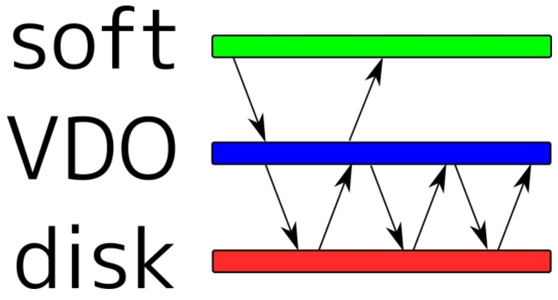
- Normal RHEL7.5 repos should be available (extras, optional channels not required)
- Installation:

```
[root@rhel7u5a ~]# yum install vdo kmod-kvdo
```

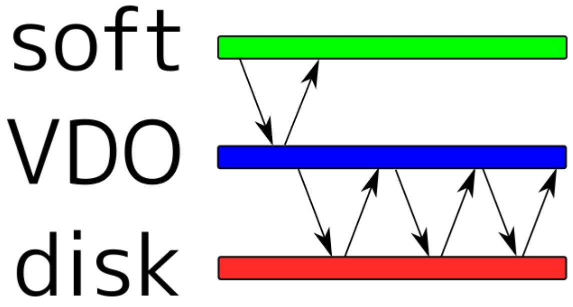
- Authoritative docs: The Storage Administration Guide

Configuring VDO devices: the 3 write modes

- **(1) ‘sync’ mode:** writes to the VDO device are acked when the underlying storage has written the data permanently. Data is here first written, then dedup/compression are done.



- **(2) ‘async’ mode:** writes are acknowledged before being written to persistent storage. VDO obeys flush requests from the layers above also in async mode. So also async mode can safely deal with your data - equivalent to other devices with volatile write back caches.

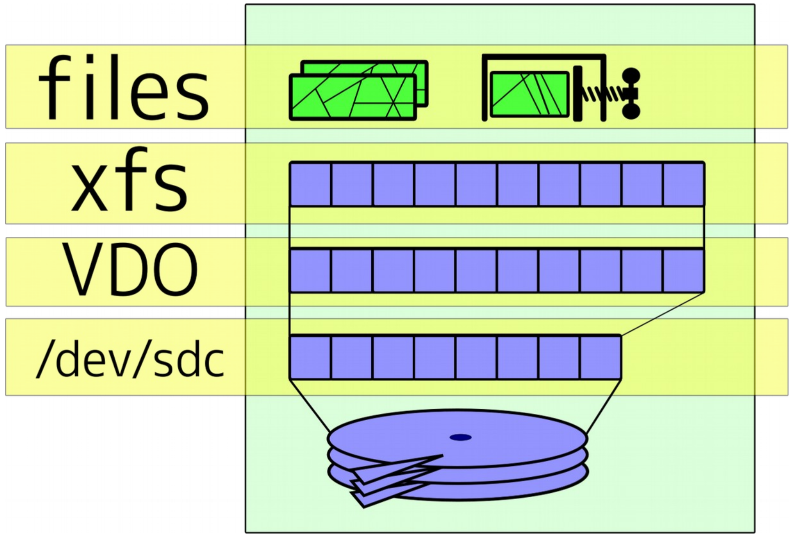


- **(3) ‘auto’ mode:** the default, selects ‘async’ or ‘sync’ based on capabilities of the underlying storage. If ‘auto’ puts you into ‘sync’ you are safe – unless your drive reports capabilities incorrectly, in that case you can manually choose ‘sync’. In all other cases, the only safe mode is ‘async’.

Configuring VDO devices, simple example

- Let’s create a VDO device on top of disk /dev/sdc. For a 10GB disk, depending on workload, one could decide to have VDO offer 100GB to the upper layers:

```
$ vdo create --name=vdoas --device=/dev/sdc \
    --vdoLogicalSize=100G --writePolicy=async
Creating VDO vdoas
Starting VDO vdoas
Starting compression on VDO vdoas
VDO instance 0 volume is ready at /dev/mapper/vdoas
$ mkfs.xfs -K /dev/mapper/vdoas
[..]
$ mount /dev/mapper/vdoas /mnt
$ cp -r /tmp/data /mnt/file
```



Configuring VDO devices, considerations

- Don’t just stuff VDO ‘somewhere’, read Storage Admin Guide: VDO requirements first.
  - For example, placing VDO below encryption layers like LUKS makes no sense: if you can deduplicate and compress that, it means your crypto has issues..
- For playing, 2GB RAM KVM guest is a good start. Production RAM requirements depend on the size of your blockdevice below VDO.
- Some part of the block device gets reserved and used for VDO: usually 3-4GB. Negligible in enterprise environments.

VDO performance impact?

The work flow:

- Create file system on VDO devices, and on plain LVM volumes:

```
$ mkfs.xfs -K -f /dev/mapper/vdo
$ mkfs.xfs -K -f /dev/vg0/lvplain
```

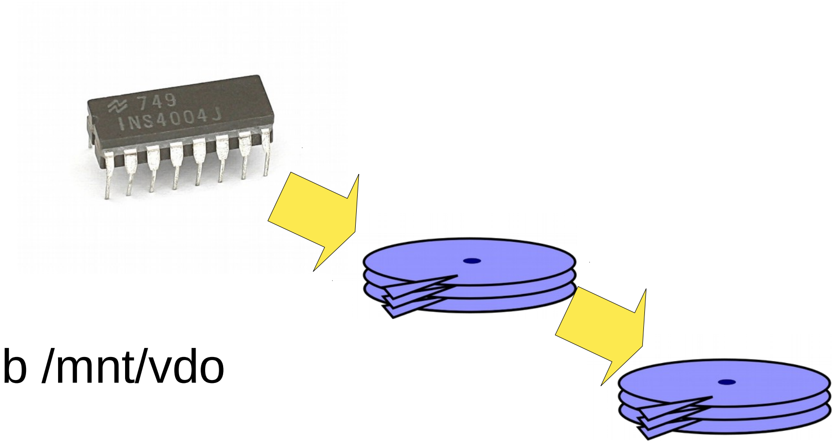
- Mount:

```
$ mount /dev/mapper/vdo /mnt/vdo
$ mount /dev/vg0/plain /mnt/plain
```

- Measure time of deployment, and copy:

```
$ /usr/bin/time -f '%e' cp -r /dev/shm/dir_5gb /mnt/vdo
$ /usr/bin/time -f '%e' sync
$ /usr/bin/time -f '%e' cp -r /mnt/vdo/dir_5gb /mnt/vdo/dir_5gb_copy
$ /usr/bin/time -f '%e' sync
```

- Results on the next slide..



VDO performance impact?

File system backend	Deploy to file system	Copy on file system
XFS onto of normal LVM volume	28 sec	35 sec
XFS on VDO device, async mode	55 sec	58 sec
XFS on VDO device, sync mode	71 sec	92 sec

- Writes to VDO are slower than to plain backend. Backend here was harddisk, with for example SSD as backend, the impact is lower.
- Same for copies on VDO: that data is duplicate, first gets written and then recognized as duplicate. VDO works in kernel land, unaware of above layers. So userland ‘cp’ is not telling it ‘this is a duplicate’.
- ‘tar’ has interesting features: ‘tar cf /dev/null /dir’ is not doing what one might expect

How much storage can I save?

- Monitor actual fill state: ‘vdostats –verbose’. Example for a 50GB volume:

.

```
[root@rhel7u5 ~]# vdostats \
>  --verbose /dev/mapper/vdoasync | \
>  grep -B6 'saving percent'
      physical blocks              : 13107200
      logical blocks               : 26214400
      1K-blocks                    : 5242880
      1K-blocks used               : 4227396
      1K-blocks available          : 48201404
      used percent                 : 8
      saving percent               : 99
```

Size of the backend blockdevice, 13.107.200 blocks * 4k byte = 50GB

How much do we show to upper layers, 26.214.400 blocks * 4k byte = 100GB → thin provisioned VDO volume

1k blocks the VDO volume can use

1k blocks internally used. Right after VDO creation ~4GB are in use.

1k blocks available, here 45.9GB

Volume fill state

- We are dealing with compression/deduplication here. So while we have 45.9GB available in VDO, if we store nicely deduplicatable data, this is more data on the file system layer.

How much storage can I save?

Let's make a copy of 13GB of data on top of VDO/XFS.

```
$ df -h /mnt/vdo0/
Filesystem          Size  Used  Avail  Use%  Mounted on
/dev/mapper/vdoas  100G    13G  88G     13%     /mnt/vdo0
$ vdostats --human /dev/mapper/vdoas
Device                   Size    Used   Available  Use% Space saving%
/dev/mapper/vdoas    50.0G  16.3G        33.7G     32%                  4%
```

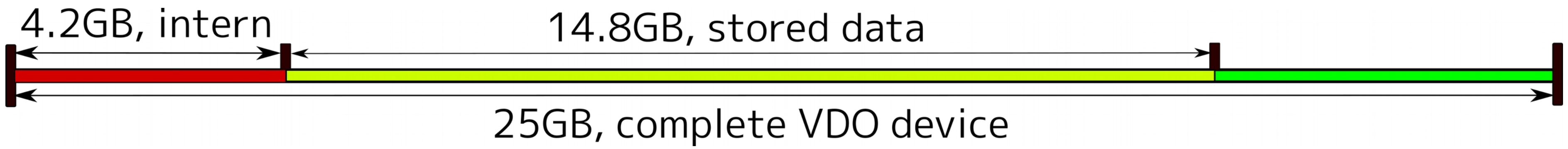
```
# cp -r /mnt/vdo0/dir13gb /mnt/vdo0/copy
```

```
$ df -h /mnt/vdo0/
Filesystem          Size  Used  Avail  Use%  Mounted on
/dev/mapper/vdoas  100G    26G  75G     26%     /mnt/vdo0
$ vdostats --human /dev/mapper/vdoas
Device                   Size     Used  Available  Use%  Space saving%
/dev/mapper/vdoas   50.0G   16.3G       33.7G    32%                  52%
```

13GB of data on file system layer, but occupies just ~120MB for VDO. Thanks, dedup! :)

Give me compression numbers!

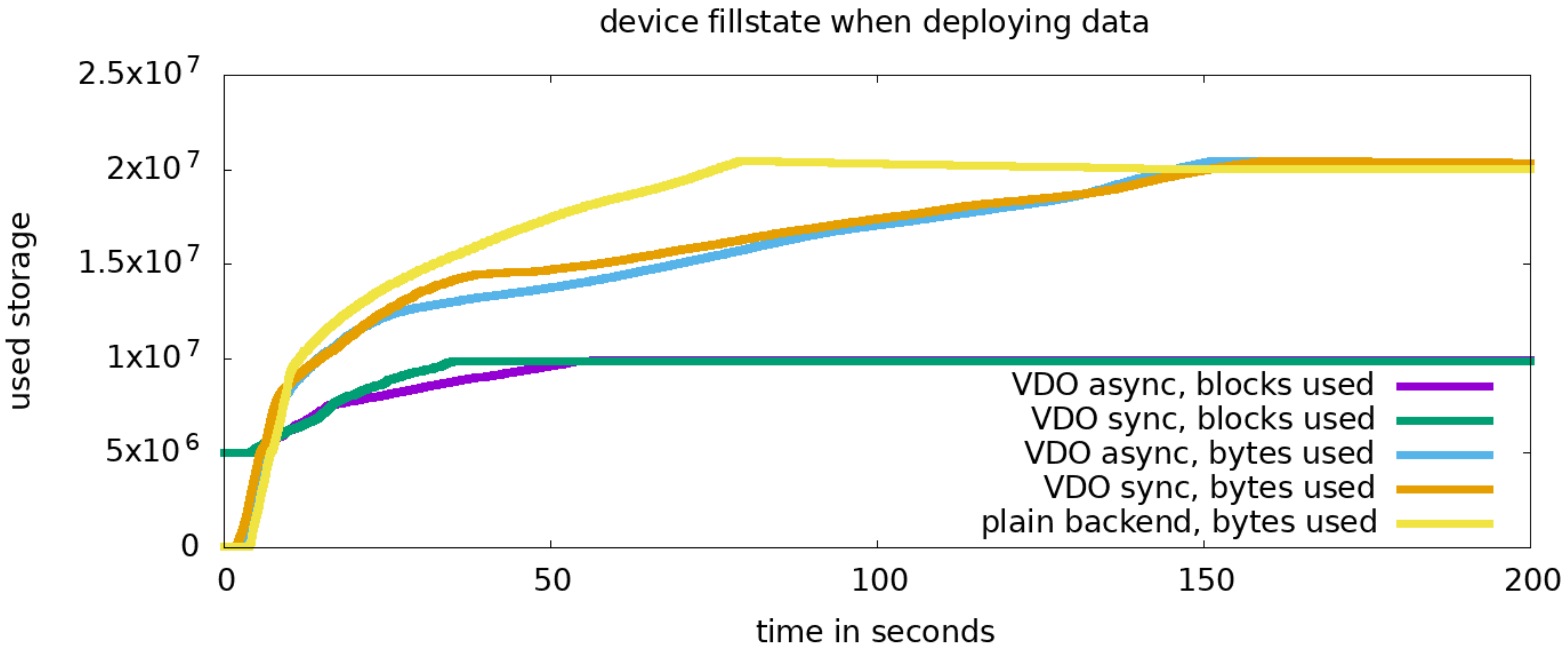
- **Note:** this is for example data, not data from your environments.
- We created a 25GB sparse file, and VDO/XFS on top. Right after creation, VDO uses 4.2GB. Let's then copy the data in:



data.tar on a normal file system	16.8GB	gutenberg.org text files
datasize on VDO (according to vdostats)	14.8GB	Took 608 sec on 4 cores/10GB RAM KVM guest, loopback
size of VDO-sparsefile	15.9GB	
size gzip(data.tar)	8.0GB	Took 992 sec on 1 core/10GB RAM KVM guest

Saving storage, illustrated

- Let's copy data to a device backed by harddisk, and then create copies:



- Yellow: plain backend finishes first – we already know that from earlier tests.
- Violet and green lines are the blocks used by our data on VDO devices. Async and sync mode are similar in this aspect.
- Both VDO volumes start with reporting '0 bytes occupied' via the 'df' command, but right from the start some blocks are used internally. For the VDO backends, the initial copy takes ~50 seconds, then the copies on top of VDO start. Due to deduplication, almost no further blocks get used at that time, but 'used bytes' as reported by the file system layer grows

Takeaways

- Deduplication is very impressive, if applicable to your data. If compression does not help with your data, it can be disabled. Compression rates are lower than when using gzip/xv.
- I/O is not improving from the applications point of view. When VDO sees a potential duplicate, it does a read verification to be sure – this takes time.
- VDO is designed for high performance in environments with random I/O, so using VDO as shared storage with multiple tasks on top doing I/O. Especially use cases like running multiple VMs on a single VDO volume let VDO shine.
- Use ‘vdostats’ for monitoring VDO device fill state: they should not fill up
- When benchmarking: carefully consider whether loopback devices and KVM change results. They are fine for comparing compression rates, but not for comparing I/O.

Conclusion and links

- Video
  Block Deduplication and Compression with VDO
  from Devconf 2018 is highly recommended.
- ‘man vdo’ has details regarding many tuning options like read caches. Extra tuning recommended for SSD and Nvram backends. The
  VDO section in the Storage Admin guide got recently extended with more details and a tuning section.

Thank you!

どうもありがとうございました！

Спасибо!

Danke!

Christian Horn / chorn@fluxcoil.net