# QCOW2 in the Linux kernel

Manuel Bentele

University of Freiburg

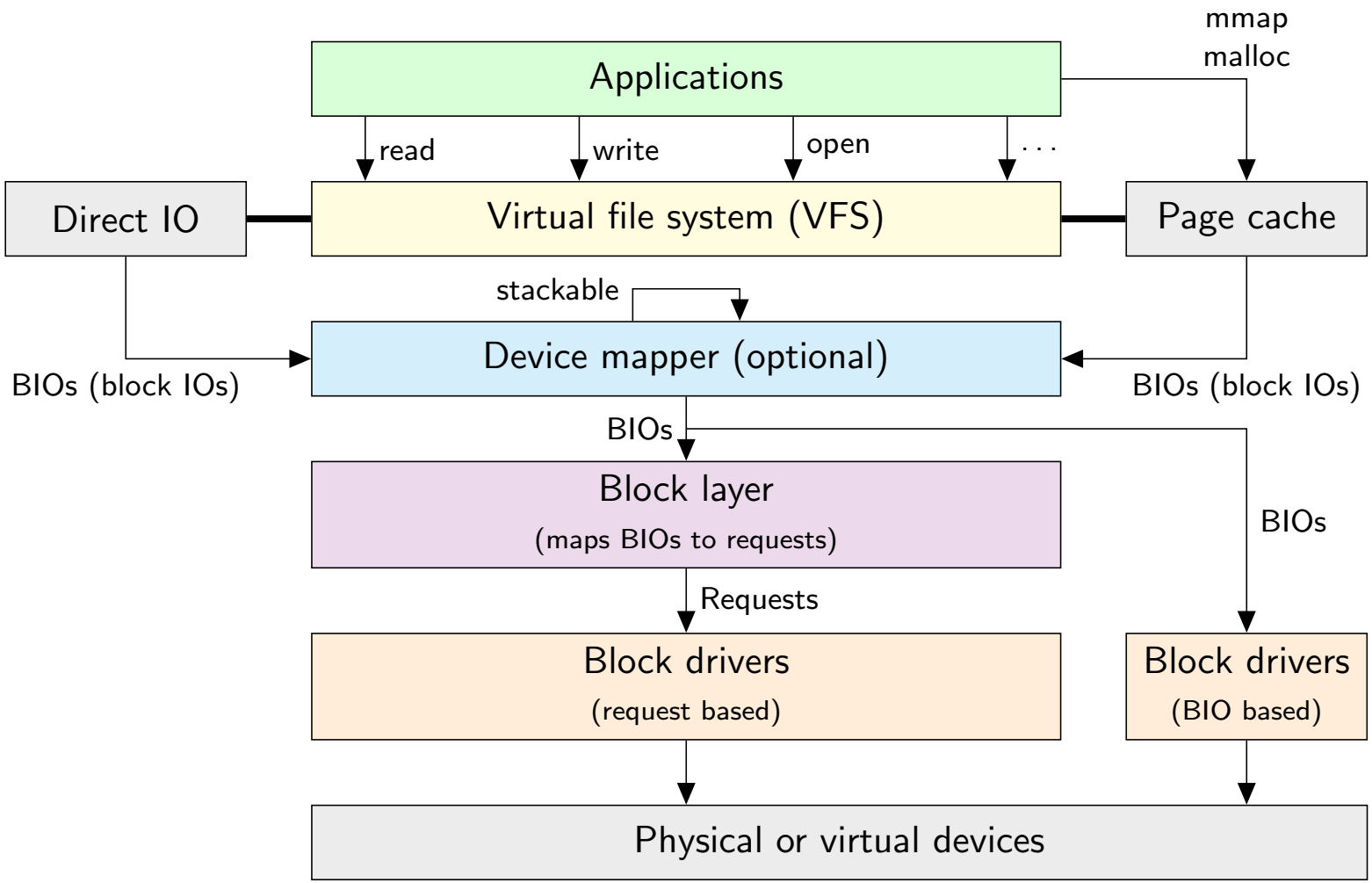
September 2, 2019

# What has to be done?

Implement the QCOW (QEMU Copy On Write) disk file format version 2 in the Linux kernel such that ...

- reading of the normal QCOW2 disk file format is possible
- compressed & sparse QCOW2 disk files are supported as well
- the disk file format is exposed as block device
- the implementation compiles & runs under Linux kernel 5 later
- the performance is better than using qemu-nbd

# How does the Linux storage stack looks like?



# How can the implementation be achieved?

## ✗ FUSE (Filesystem in Userspace) driver

- implement reading of QCOW2 file format as user space driver

## ✗ Device mapper target

- implement reading of QCOW2 file format as mapped target

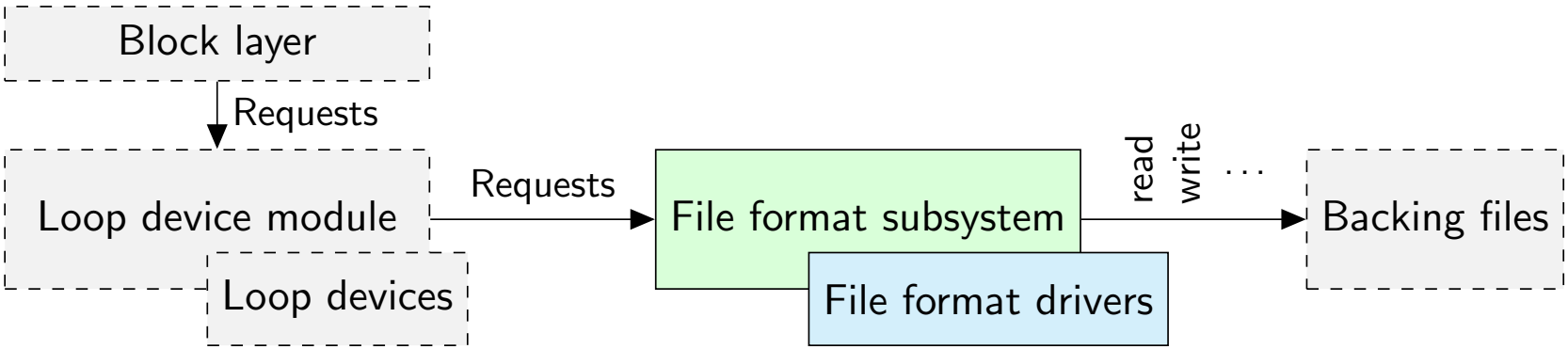
## ✗ Custom block driver

- create block driver & configuration utility for reading QCOW2

## ✓ Loop device module extension

- extend the loop device module & configuration utility by a file format subsystem to implement QCOW2 as additional file format driver

# How is the file format subsystem integrated?

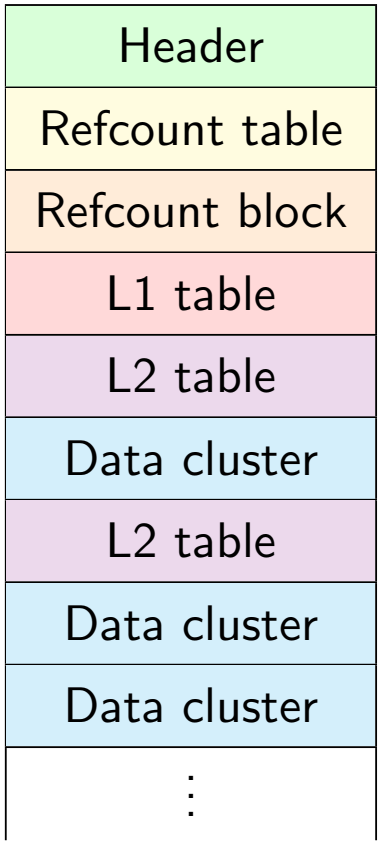



- file format subsystem abstracts the direct access to backing files to implement various file formats
- file formats are implemented as file format drivers
- drivers are registered at the subsystem
- subsystem supports (asynchronous) reads, (asynchronous) writes, flushes and virtual disk sizes

# How does a file format driver look like?

```
#include "loop_file_fmt.h"

static int drv_file_fmt_read(struct loop_file_fmt *lo_fmt,
                             struct request *rq) {
    /* TODO: implement reading of file format */
    return -EIO;
}

static struct loop_file_fmt_ops drv_file_fmt_ops = {
    .read = drv_file_fmt_read
};

static struct loop_file_fmt_driver drv_file_fmt = {
    .name = "DRV",
    .file_fmt_type = LO_FILE_FMT_RAW,
    .ops = &drv_file_fmt_ops,
    .owner = THIS_MODULE
};

// register driver with loop_file_fmt_register_driver(&drv_file_fmt)
// unregister driver with loop_file_fmt_unregister_driver(&drv_file_fmt)
```

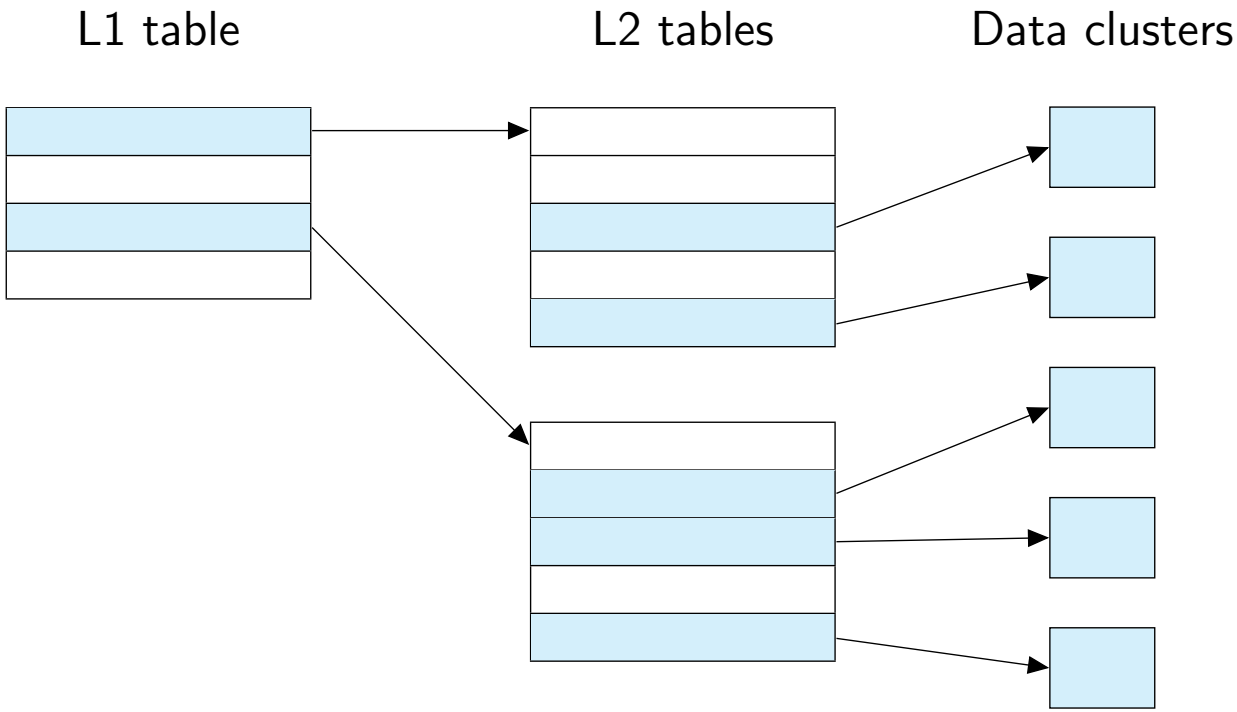
# How is the QCOW2 disk file format structured?



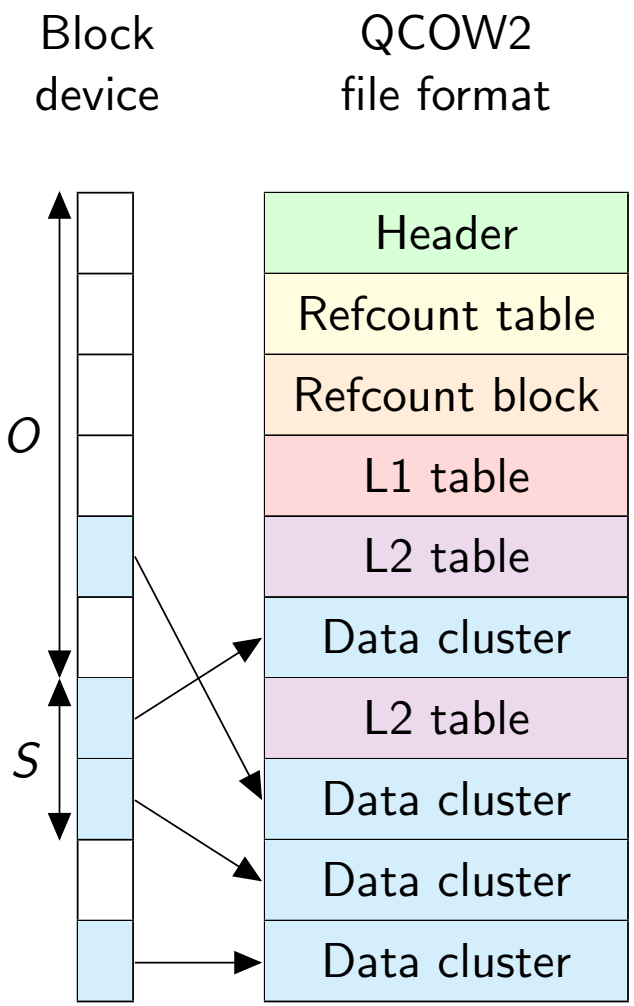

- data is saved in data clusters of equal size (512 B – 2 MB)
- header provides offsets to 1st level tables
- two-level lookup of data clusters (L1 & L2 tables)
- two-level reference count for copy on write (Refcount & Refcount block tables)
- numbers are stored in big-endian order
- data clusters can be compressed or encrypted
- support of embedded snapshots by use of internal copy on write

# How does QCOW2 addresses data clusters?

- QCOW2 header stores an offset in the file to the L1 table
- L1 table stores offsets in the file to L2 tables
- L2 tables stores offsets in the file to the data clusters



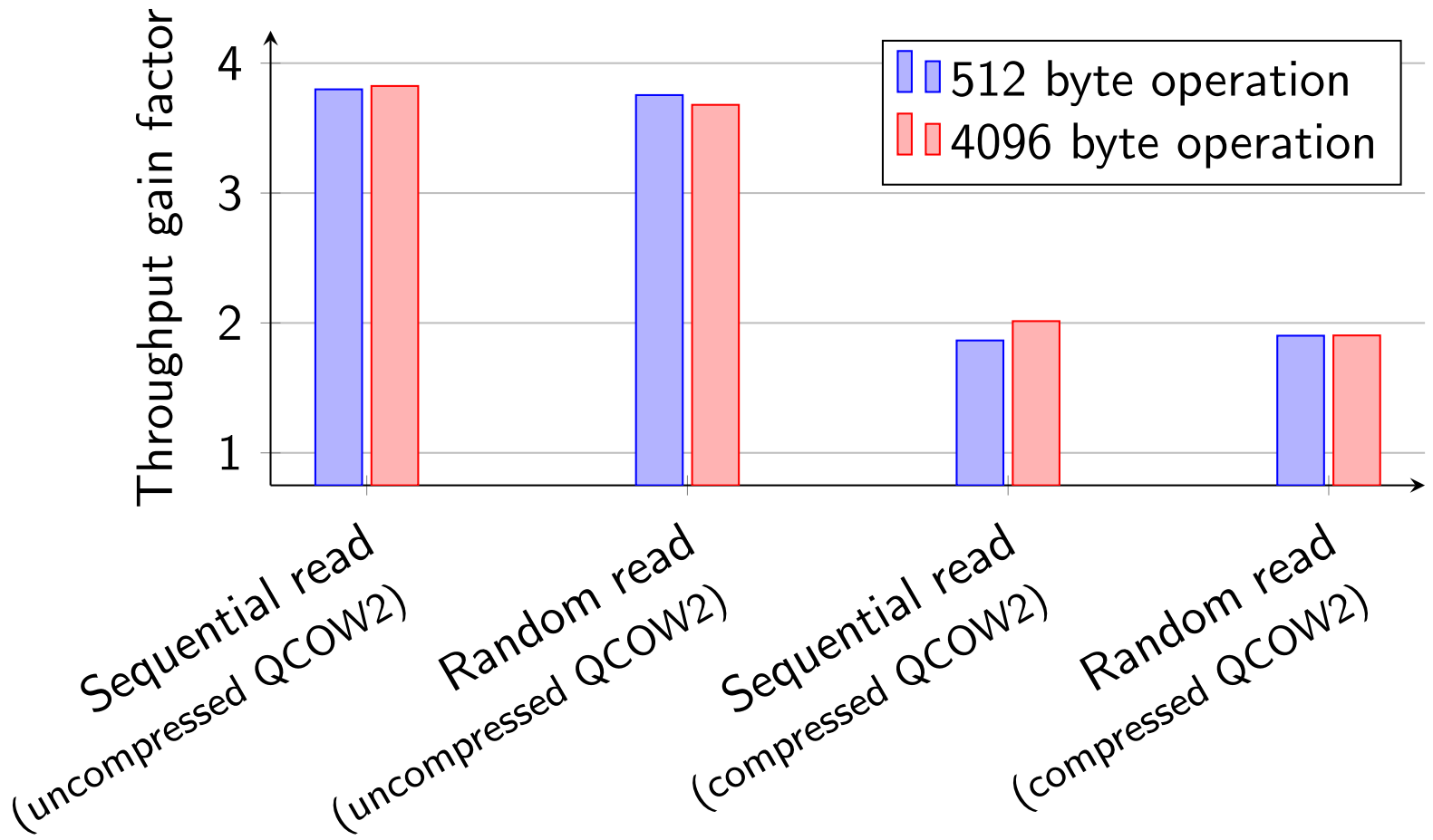
# How does the QCOW2 driver read data?




Given a Linux IO read request with size $S$ and block device offset $O$:

1. calculate cluster $C$ and position $P$ for $O$ using cached L1 & L2 tables
2. decompress the data of $C$ if $C$ is compressed
3. read data from $P$ into IO read request until $S$ bytes or the end of $C$ is reached
4. repeat steps 1 – 3 until IO read request is filled with $S$ bytes

# How does the implementation perform compared to qemu-nbd?



# What can be done in the future?

## File format subsystem

- implement other file formats, e.g. VDI, VMDK, ...
- extend the API to support snapshots & encryption

## QCOW2 file format driver

- implement write operations
- implement encryption & snapshot support
- improve performance by hardware aligned cache allocation
- add a QCOW2 L2 cache clean interval