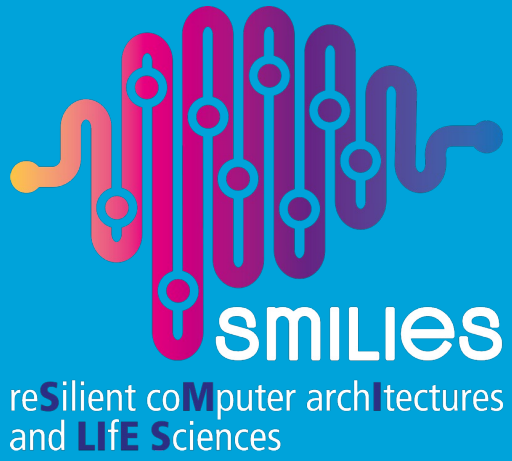


Politecnico
di Torino

Department of Control and
Computer Engineering

INTRODUCTION TO KERNEL MODULES

STEFANO DI CARLO

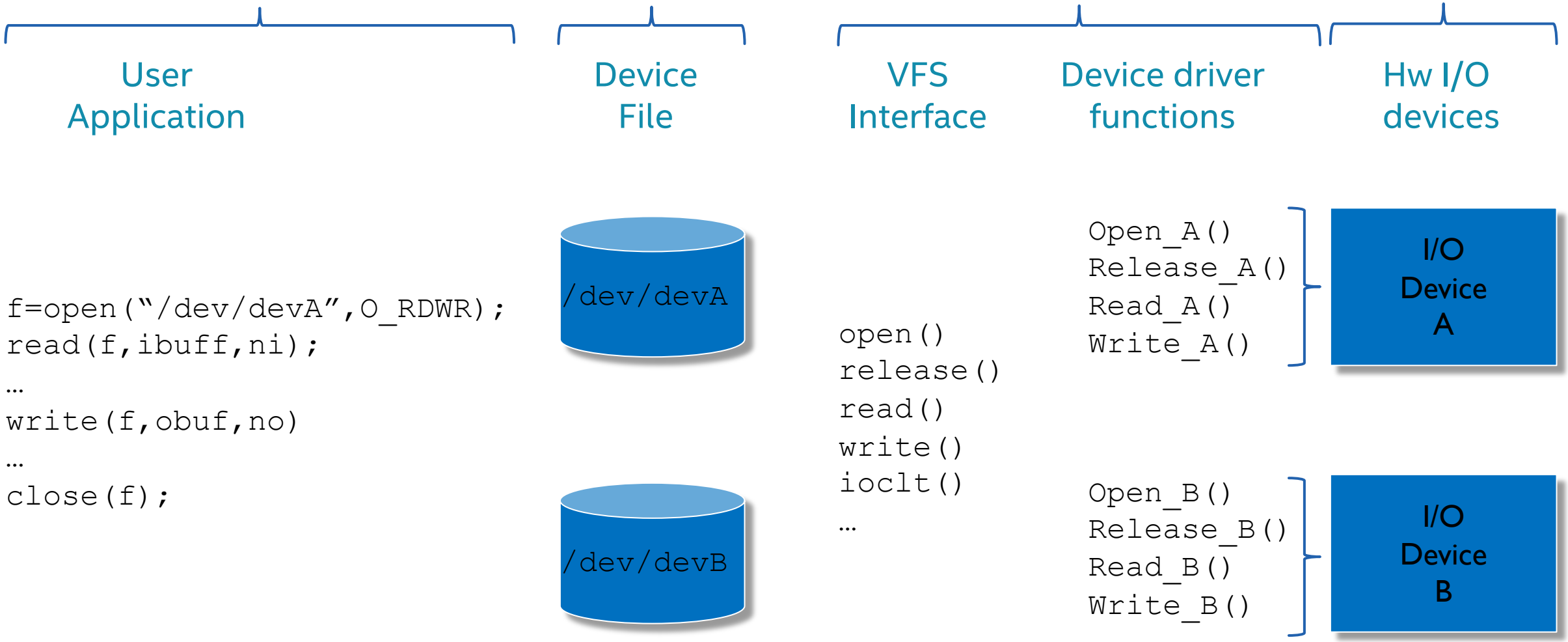
LINUX DEVICES

- Linux provides an abstraction to make communication with I/O easy.
  - Software developer does not need to know every detail of the physical device.
  - Portability can be increased by using the same abstraction for different I/O devices.
- Linux recognizes three classes of devices:
  - Character devices which are devices that can be accessed as stream of words (e.g., 8-/16-/32-/... bits) as in a file; reading word n requires reading all the preceding words from 0 to n-1.
  - Block devices, which are devices that can be accessed only as multiples of one block, where a block is 512 bytes of data or more. Typically, block devices host file systems.
  - Network interfaces, which are in charge of sending and receiving data packets through the network subsystem of the kernel

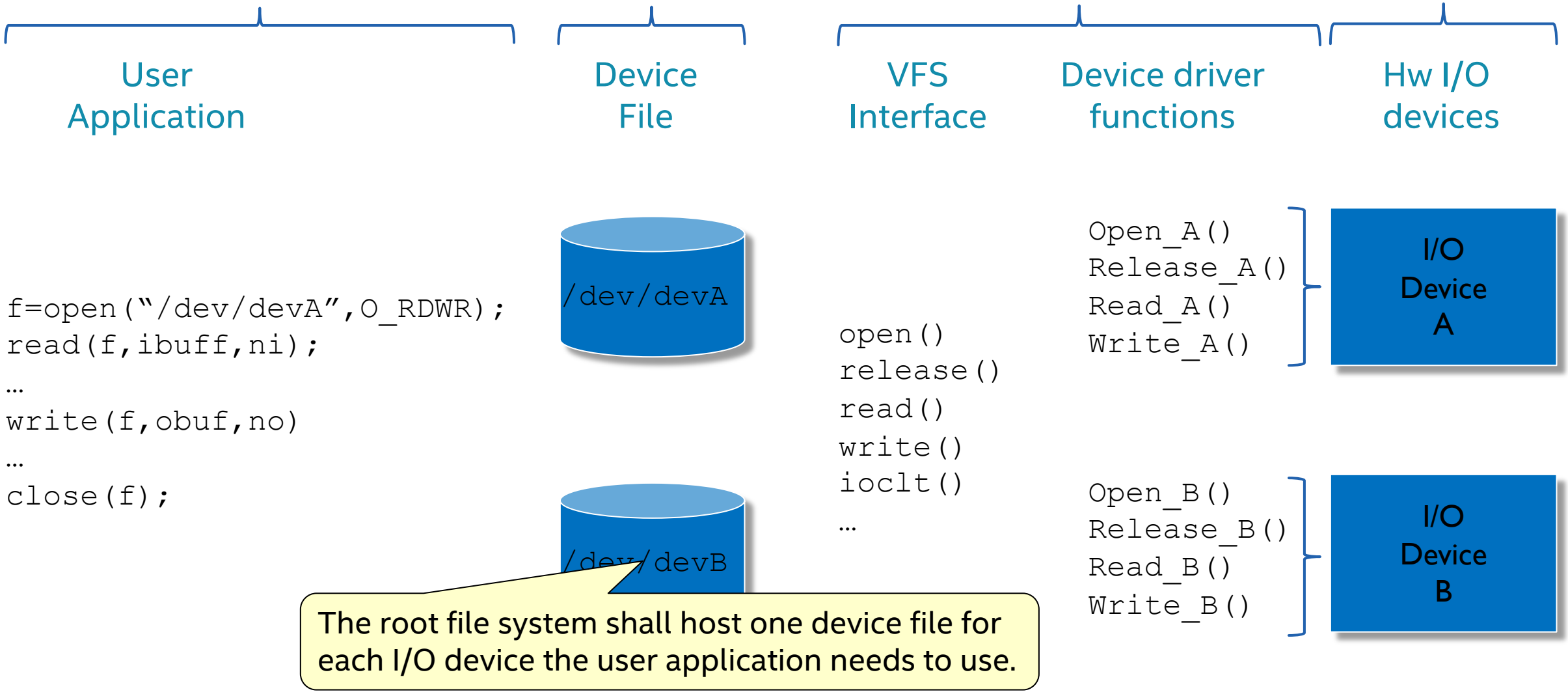
THE VIRTUAL FILE SYSTEM (VFS) ABSTRACTION

- Character/block devices are accessed as files stored in the file system, as each device is associated with a device file
- Typical usage:
  - Open the device file
  - Read/Write data from/to device file
  - Close the device file
- Linux forwards the open/read/write/close operations to the I/O device associated to the device file.
- The operations for each I/O device are implemented by a custom piece of software in the Linux kernel: the device driver.
- In the following, we will consider only character (char) devices.

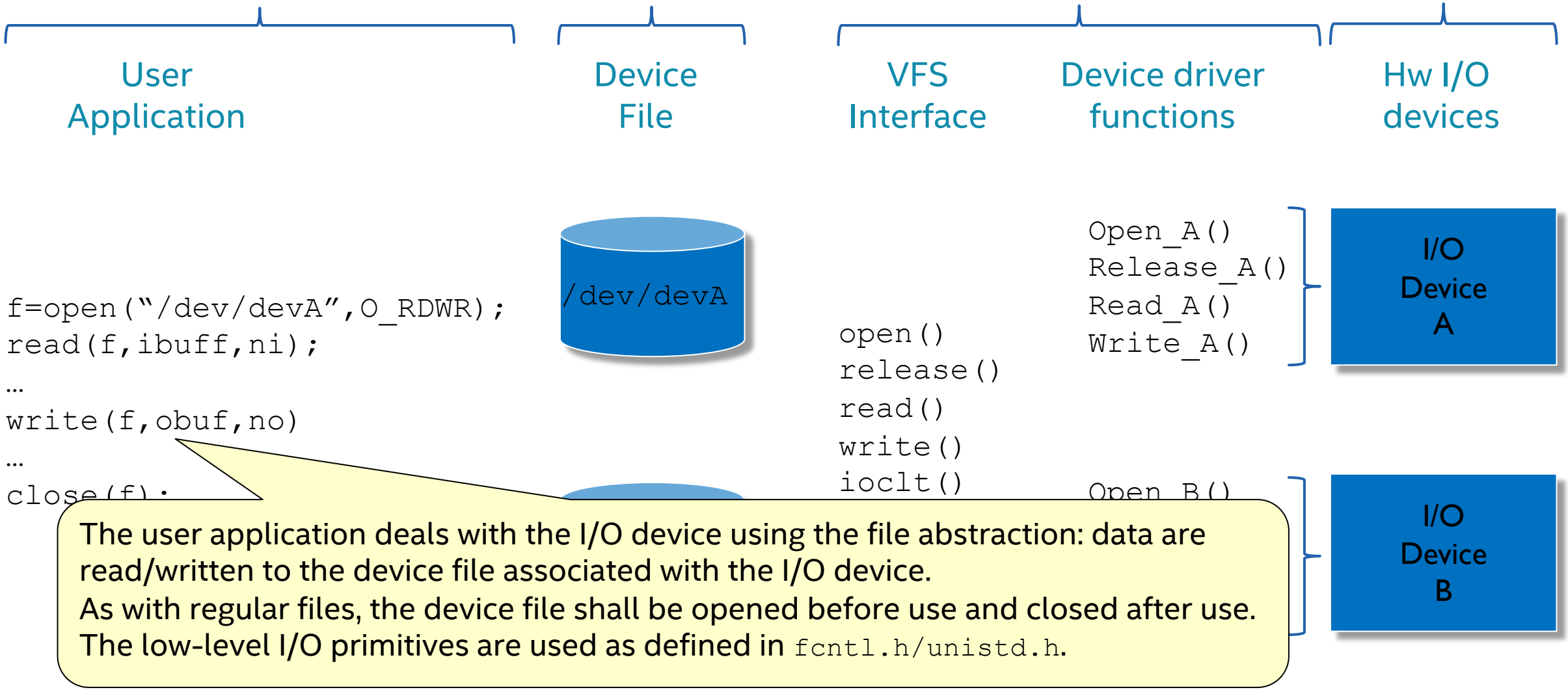
VFS: AN EXAMPLE



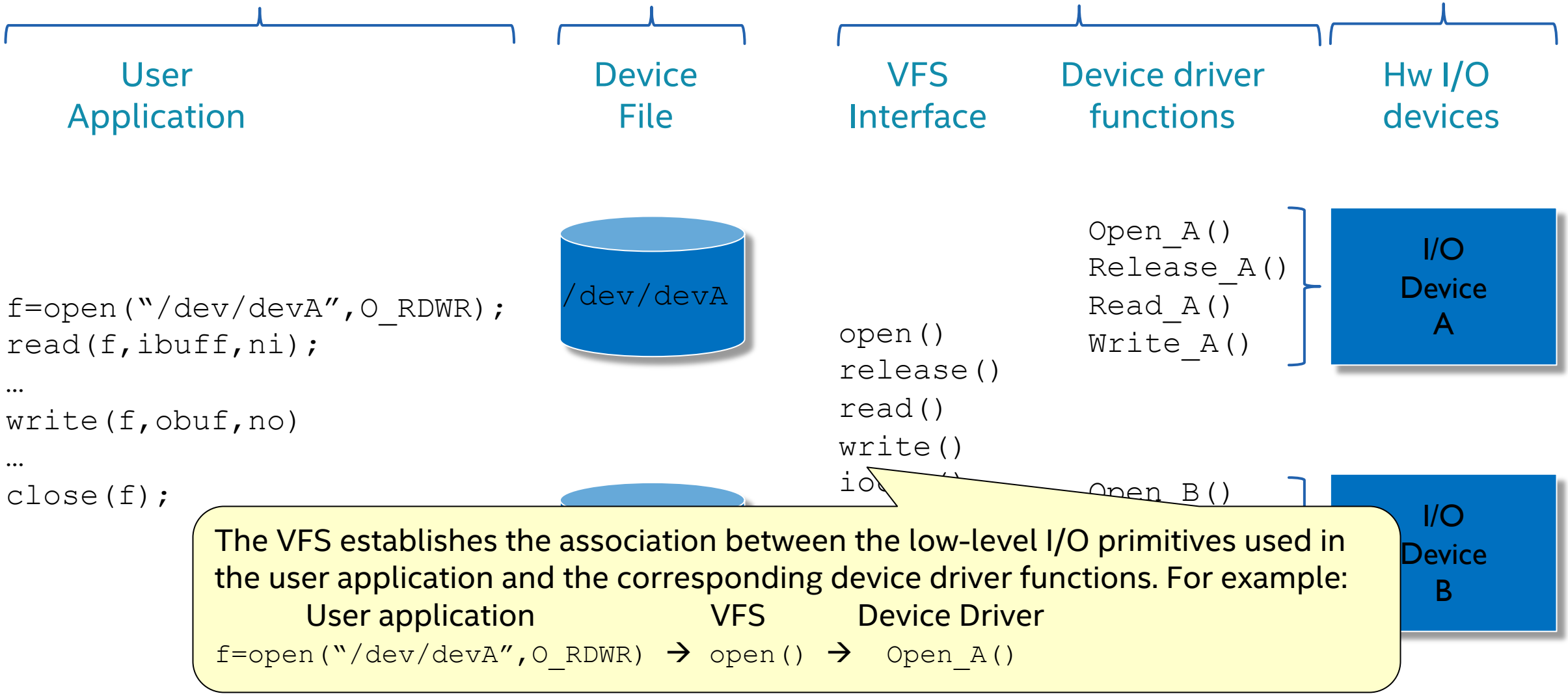
VFS: AN EXAMPLE



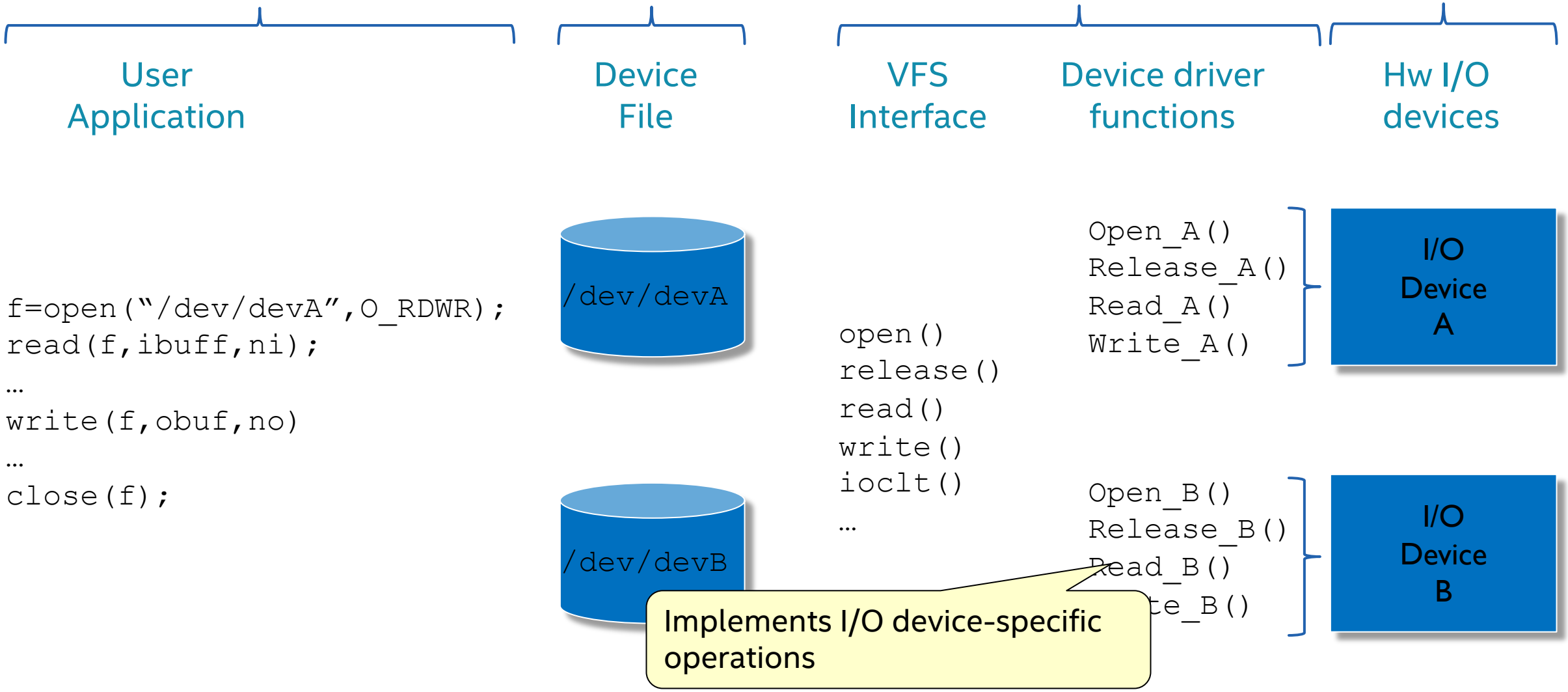
VFS: AN EXAMPLE



VFS: AN EXAMPLE



VFS: AN EXAMPLE



VFS FUNCTIONS: INCLUDE/LINUX/FS.H

```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
  ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
  ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
  ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
  int (*iterate) (struct file *, struct dir_context *);
  int (*iterate_shared) (struct file *, struct dir_context *);
  unsigned int (*poll) (struct file *, struct poll_table_struct *);
  long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
  long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
  int (*mmap) (struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *, fl_owner_t id);
  int (*release) (struct inode *, struct file *);
  int (*fsync) (struct file *, loff_t, loff_t, int datasync);
  int (*aio_fsync) (struct kiocb *, int datasync);
  int (*fasync) (int, struct file *, int);
  int (*lock) (struct file *, int, struct file_lock *);
  ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
  unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long, unsigned long);
  int (*check_flags)(int);
  int (*flock) (struct file *, int, struct file_lock *);
  ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);
  ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);
  int (*setlease)(struct file *, long, struct file_lock **, void **);
  long (*fallocate)(struct file *file, int mode, loff_t offset,
  loff_t len);
  void (*show_fdinfo)(struct seq_file *m, struct file *f);
  #ifndef CONFIG_MMU
    unsigned (*mmap_capabilities)(struct file *);
  #endif
  ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,
  loff_t, size_t, unsigned int);
  int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,
  u64);
  ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,
  u64);
 };
```

VFS functions: prototypes of the functions Linux sets available for accessing a file. In case of device files, the actions each function performs are defined by the corresponding device driver.

VFS FUNCTIONS: `INCLUDE/LINUX/FS.H`

- For character devices the most commonly used VFS functions are the following:
  - `ssize_t (*read) (struct file *, char *__user, size_t, loff_t *)`: it reads data from a file.
  - `ssize_t (*write) (struct file *,  const char *__user, size_t, loff_t *)`: it writes data to a file.
  - `int (*ioctl) (struct *inode, struct file *,  unsigned int, unsigned long)`: it performs custom operations to the file.
  - `int (*open) ( struct *inode, struct file * )`: it prepares a file for use.
  - `int (*release) ( struct inode *, struct file * )`: it indicates the file is no longer in use.

THE DEVICE FILE CONCEPT

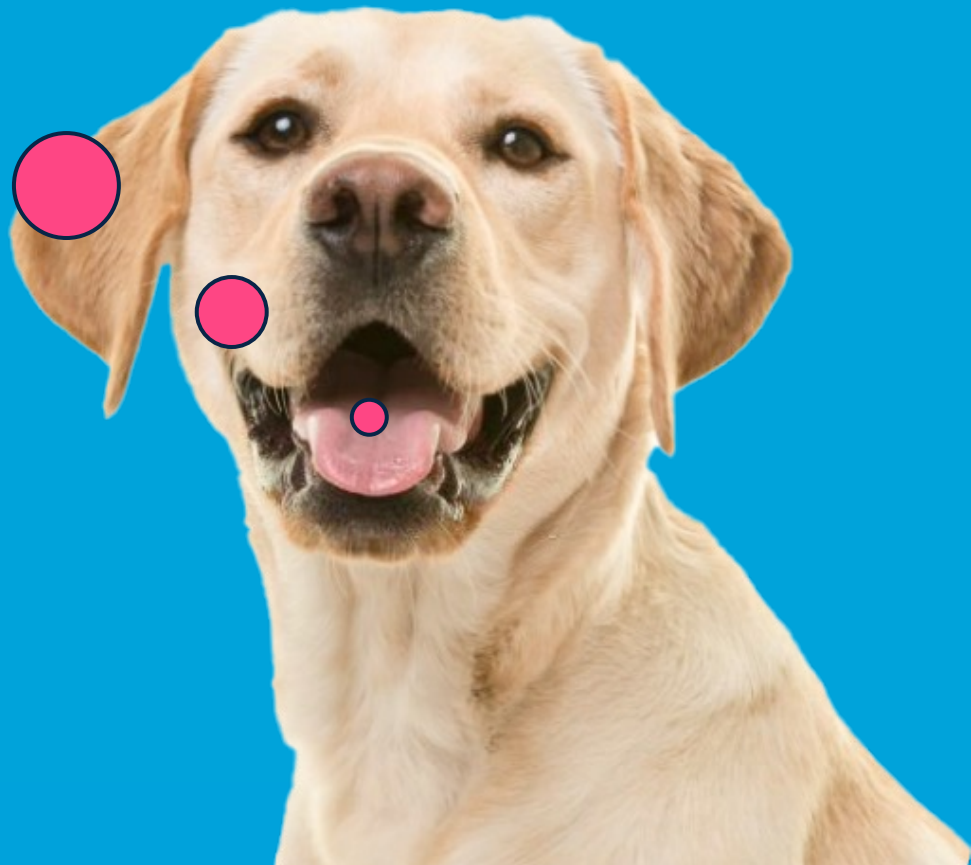
- The device file is the intermediator through which a user application can exchange data with a device driver.
- The device file does not contain any data, while its descriptor contains the relevant information to identify the corresponding driver:
  - The device file type which could be either c = character device, b = block device, or p = named pipe (inter-process communication mechanism)
  - The major number, which is an integer number that identifies univocally a device driver in the Linux kernel
  - The minor number, which is used to discriminate among multiple instances of I/O devices handled by the same device driver

SMILIES
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



THANK YOU!