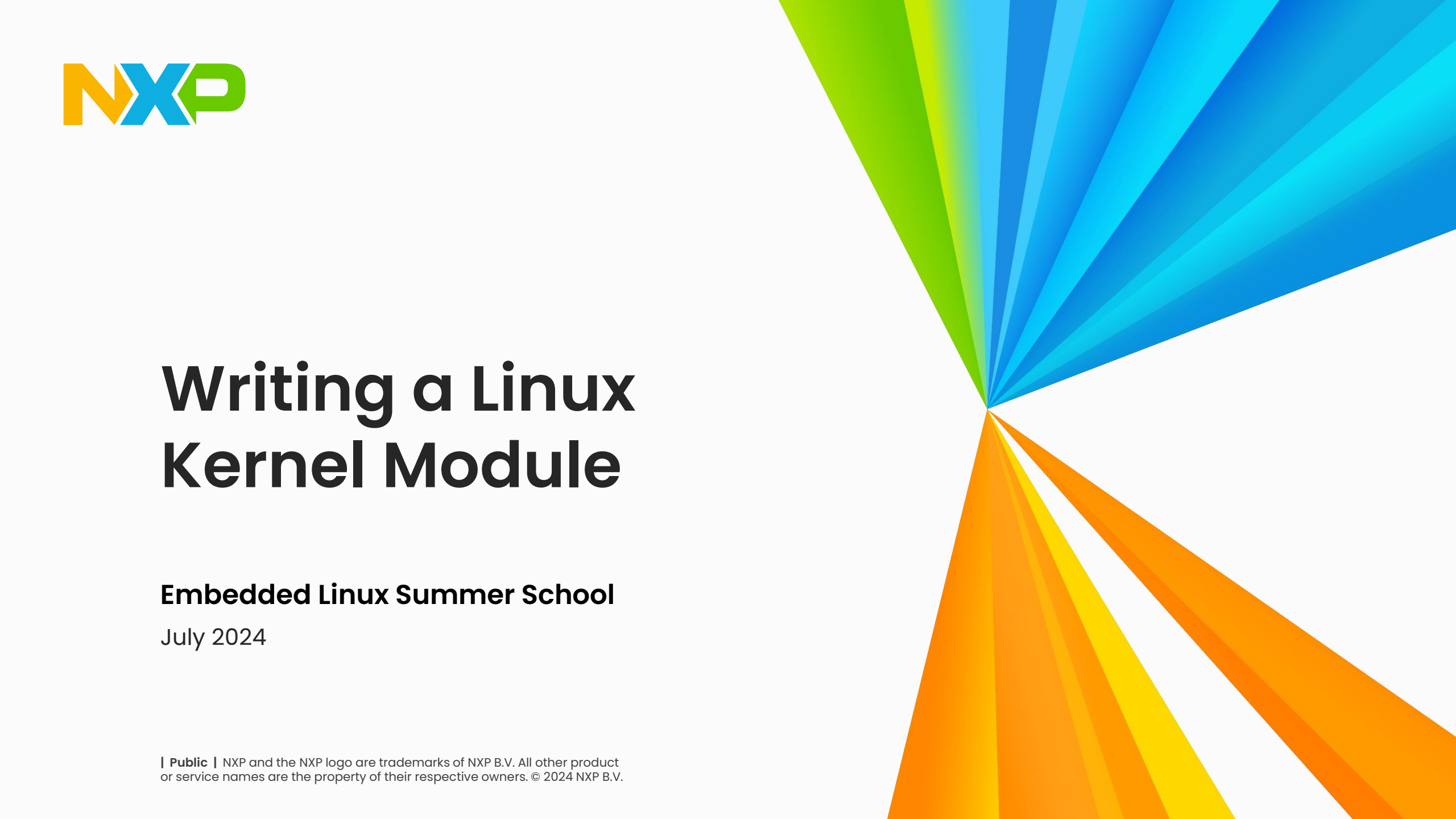


Writing a Linux Kernel Module

Embedded Linux Summer School

July 2024

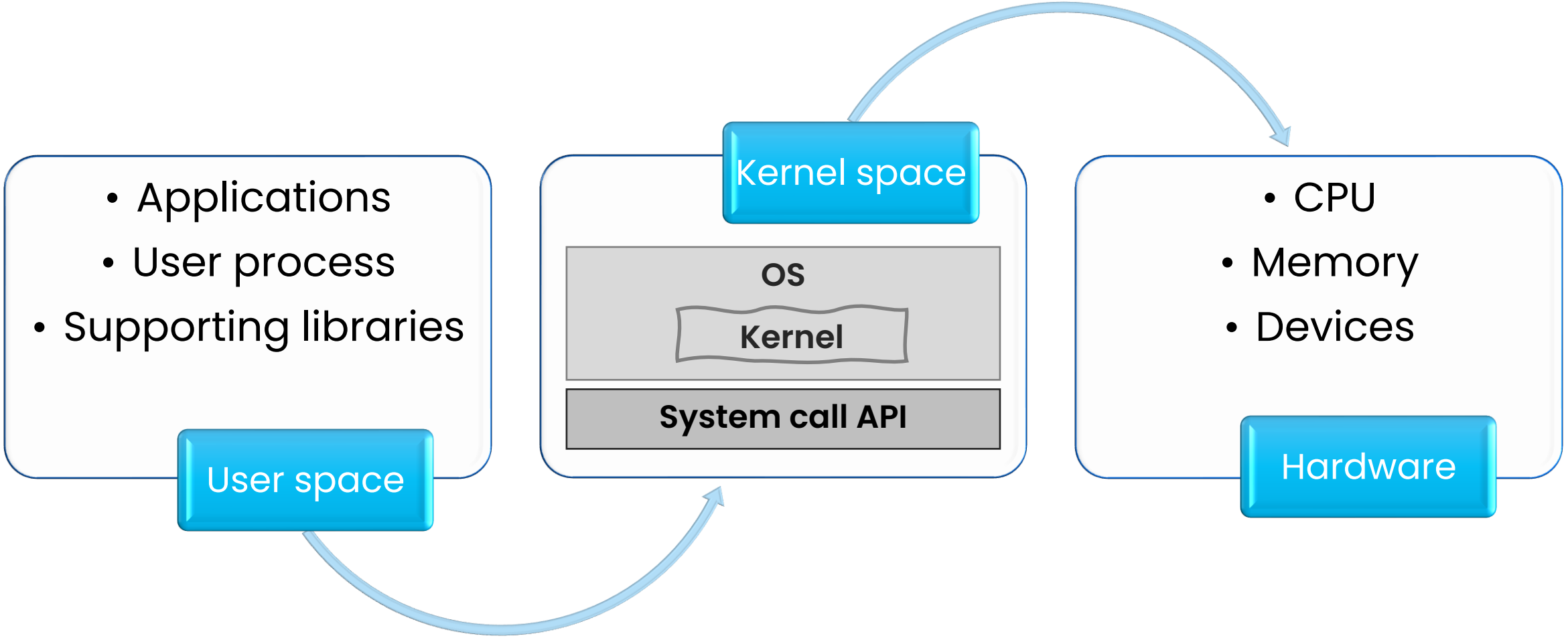
Content

Kernel space vs user space

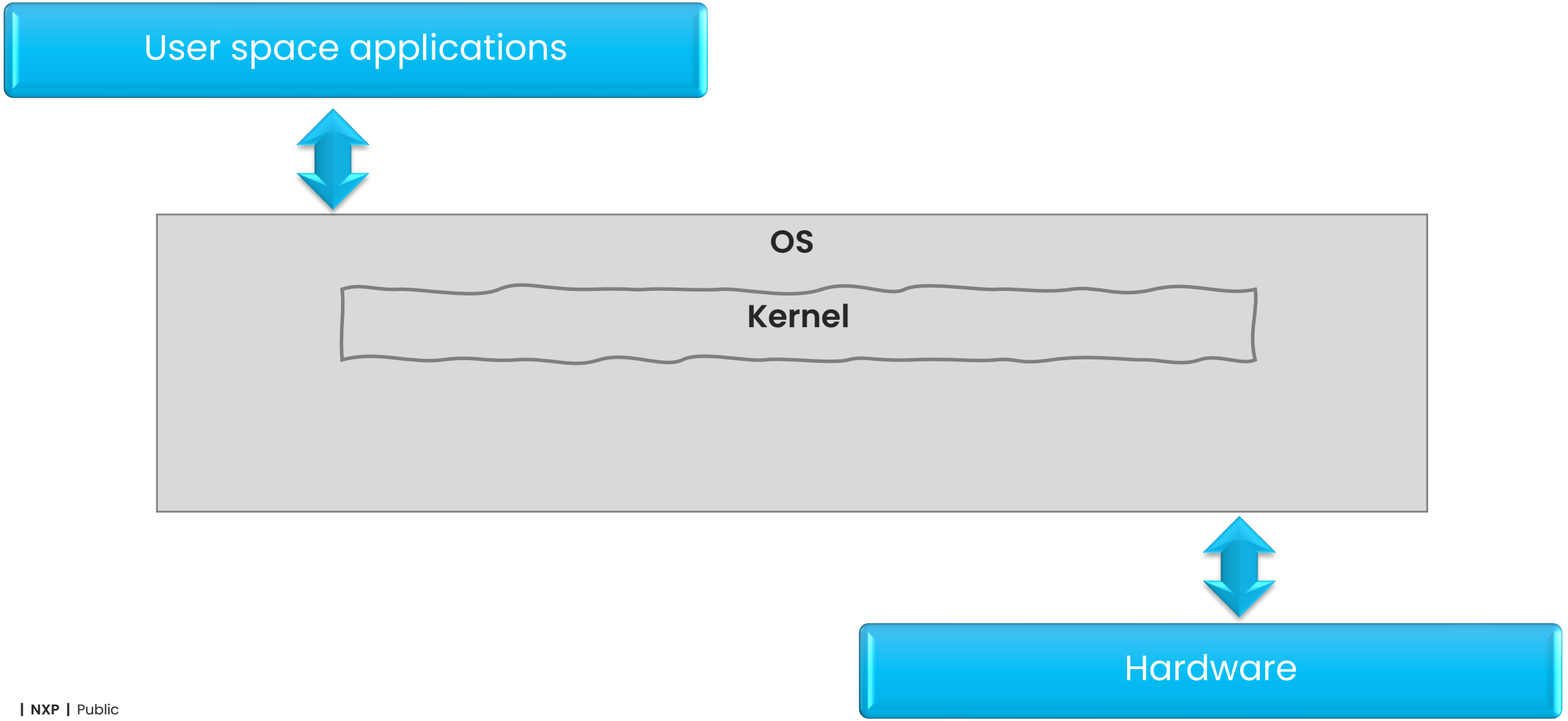
Linux kernel modules

Character device drivers

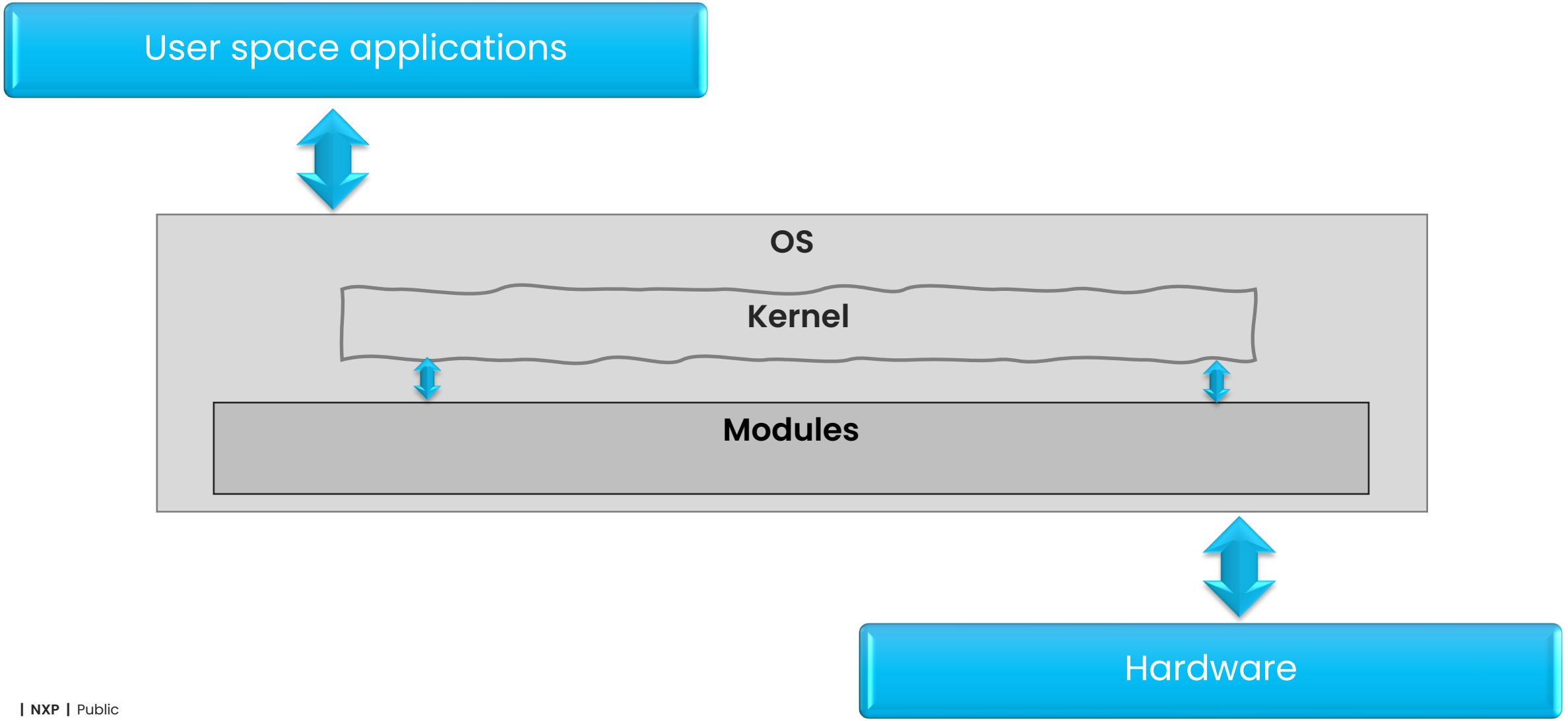
Kernel space vs User space



Kernel space vs User space

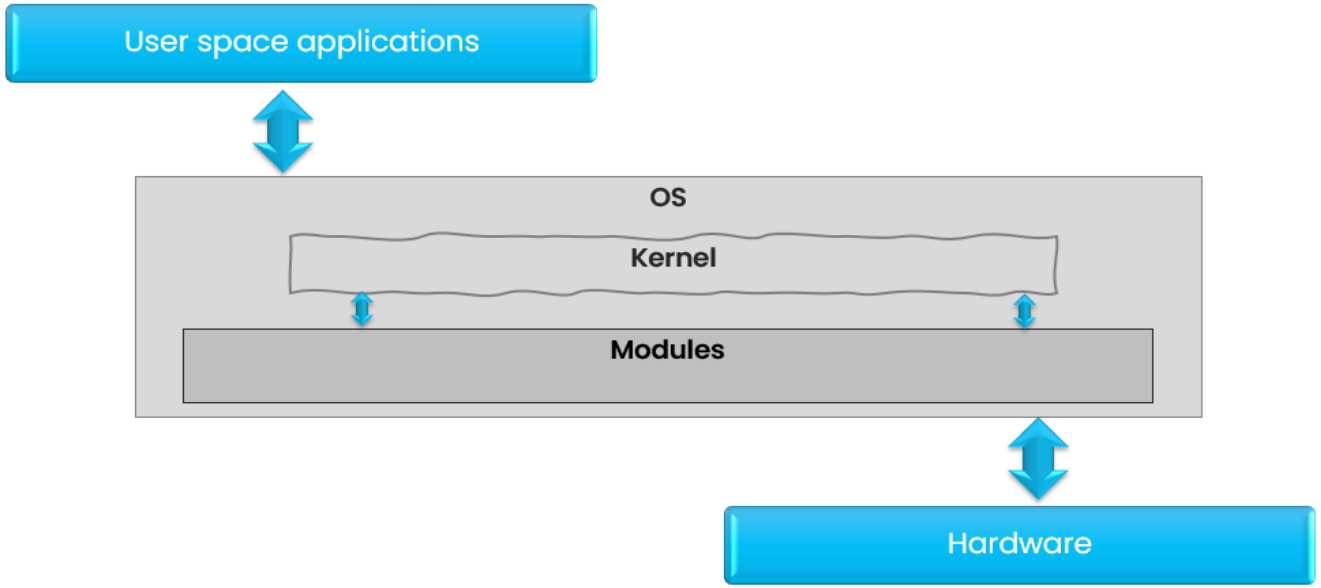


Kernel space vs User space



What is a Linux Kernel Module?

- Code that executes as part of the Linux kernel
  - Pieces of code that can be loaded and unloaded into the kernel upon demand
- Extends the capabilities and sometimes might modify the behavior of the kernel
- Without modules, one would have to build monolithic kernels and add new functionality directly into the kernel image
- Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality



Anatomy of a Kernel Module

- Several typical components:
  - MODULE_AUTHOR("Jane Doe")
  - MODULE_LICENSE("GPL")
    - The license **must** be an open-source license (GPL,BSD, etc.) or you will "taint" your kernel

- `int init_module(void)`
  - Called when the kernel loads the module
  - Initialize all stuff here
  - Return 0 if all went well, negative if something wrong

- `void cleanup_module(void)`
  - Called when the kernel unloads the module
  - Free all resources here

Compiling a Kernel Module

- Accompany the kernel module with a 1-line GNU Makefile:

  `obj-m += hello.o`

  - Assumes file name is “hello.c”
- Run the magic make command:

  `make -C <kernel-src> M=`pwd` modules`

  - Produces: hello.ko

  - Assumes current directory is the module source

Kernel Module utilities

- lsmod - Show all loaded modules
- insmod - Insert a Module (excludes dependencies)

  `$sudo insmod <module_name>`
- modprobe - Load the kernel module plus any module dependencies

  `$sudo modprobe <module_name>`
- modinfo - Show information about a module

  `$modinfo <module_name>`
- depmod - Build module dependency database

  `$/lib/modules/$(uname -r)/modules.dep`
- rmmod - Remove a module

  `$rmmod <module_name>`
- Show the log

  `$dmesg` or `$cat /var/log/syslog`

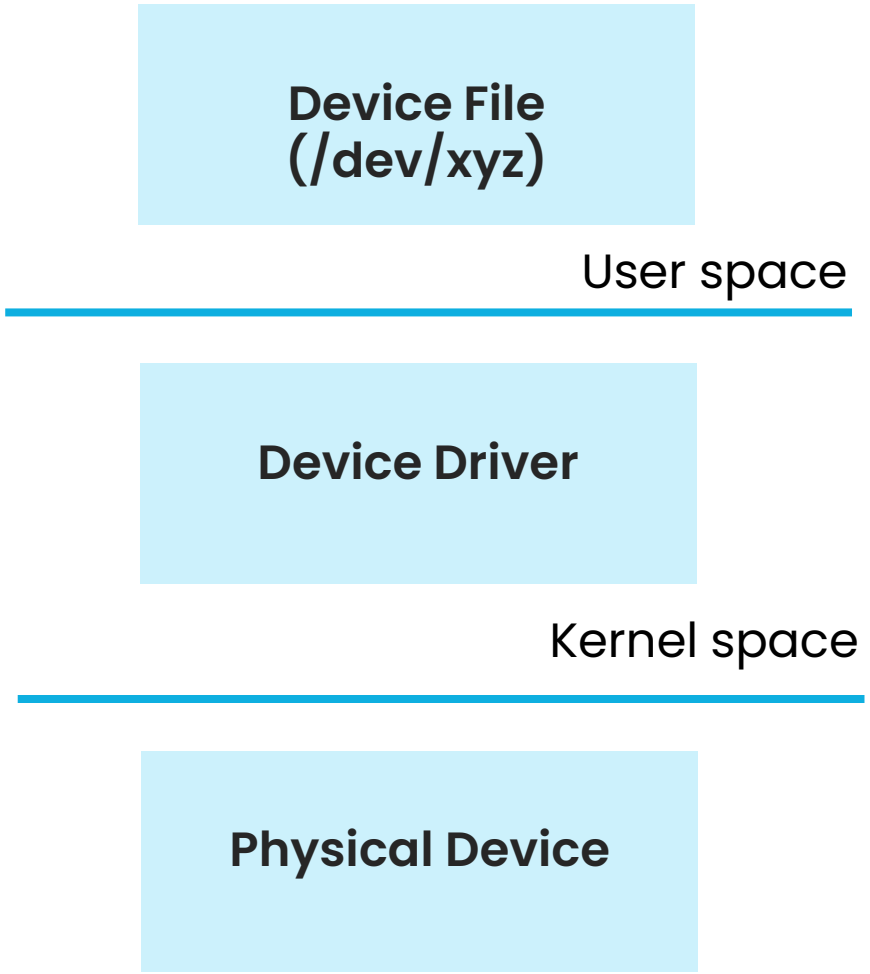
printk

- Log at kernel
- 8 priority levels (see: include/linux/kern_levels.h)
  - KERN_EMERG 0 system is unusable
  - KERN_ALERT 1 action must be taken immediately
  - KERN_CRIT 2 critical conditions
  - KERN_ERR 3 error conditions
  - KERN_WARNING 4 warning conditions
  - KERN_NOTICE 5 normal but significant condition
  - KERN_INFO 6 informational
  - KERN_DEBUG 7 debug-level messages
- `cat /proc/sys/kernel/printk`

Device drivers vs device files

- Everything is a file or a directory
- Every device is represented by a file in `/dev/`
- Device Driver: Kernel Module that controls a device
- Device File:
  - Interface for the Device Driver to
    - read from or write to a physical device
  - Also known as Device Nodes
  - Created with `mknod` system call

```
mknod [name] <c/b> <major> <minor>
```



Device files

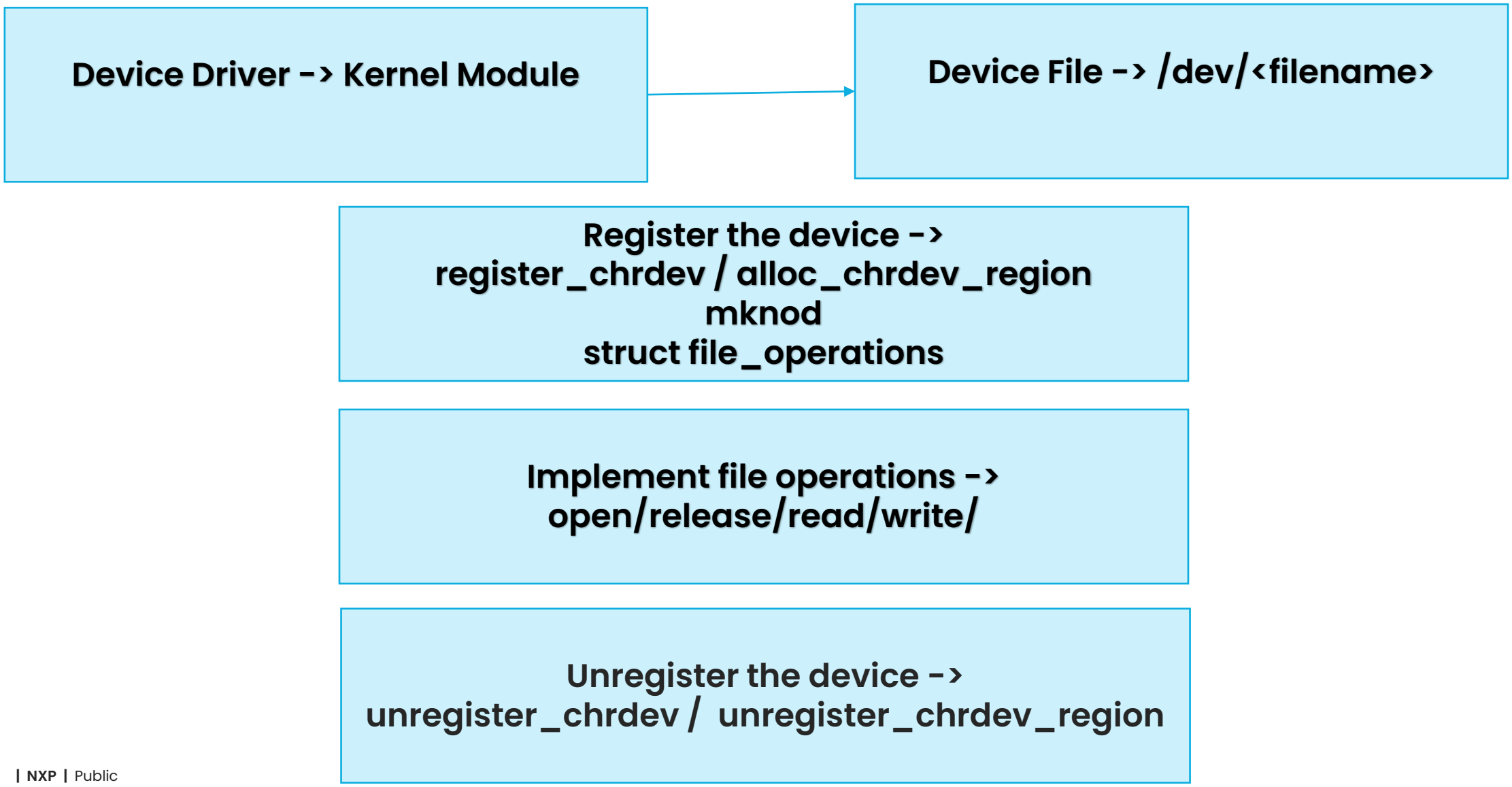
- Character device
  - Stream of data one character at a time
  - No restriction on number of bytes
- Block device
  - Random access to block of data
  - Can buffer and schedule the requests

```
$ ls -l /dev/
...
crw-------   1 root root     10,  60 Dec 15  2023 cpu_dma_latency
crw-------   1 root root     10, 203 Dec 15  2023 cuse
brw-rw----   1 root disk    253,   0 Dec 15  2023 dm-0
brw-rw----   1 root disk    253,   1 Dec 15  2023 dm-1
```

- Internally the kernel identifies each device by a triplet of information
  - Type - character or block
  - Major number - typically the category of devices
  - Minor number - typically the identifier of the device

```
crw-rw-rw- 1 root   root    1, 3   Feb 23 1999  null
crw------- 1 root   root   10, 1   Feb 23 1999  psaux
crw------- 1 rubini tty     4, 1   Aug 16 22:22 tty1
crw-rw-rw- 1 root   dialout 4, 64  Jun 30 11:19 ttyS0
crw-rw-rw- 1 root   dialout 4, 65  Aug 16 00:00 ttyS1
crw------- 1 root   sys     7, 1   Feb 23 1999  vcs1
crw------- 1 root   sys     7, 129 Feb 23 1999  vcsa1
crw-rw-rw- 1 root   root    1, 5   Feb 23 1999  zero
```

Character device driver implementation



Register device numbers

- `#include <linux/fs.h>`
- `int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count, const char *name)`
  - Allocates a range of char device numbers. The major number will be chosen dynamically, and returned (along with the first minor number) in @dev
- Registered devices are visible in /proc/devices:

```
$ cat /dev/devices
```

```
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
 10 misc
...
```

```
Block devices:
  7 loop
  8 sd
 31 mtdblock
 65 sd
 66 sd
 67 sd
 68 sd
...
```

Character device registration

- `#include <linux/cdev.h>`
- `static struct cdev char_cdev;`

- `void cdev_init(struct cdev *cdev, struct file_operations *fops);`

- `int cdev_add(struct cdev *p, dev_t dev, unsigned count);`

- The kernel knows the association between the major/minor numbers and the file operations.
  - Device is ready to be used

Character device unregistration

- `#include <linux/cdev.h>`
- `void cdev_del(struct cdev *p);`

- `void unregister_chrdev_region(dev_t from, unsigned count);`

Reading / Writing from character device

`$ cat /dev/imx8mq_chardev`

- calls `read()` function

- `unsigned long copy_to_user (void __user * to, const void * from, unsigned long n);`

`$ echo "hello" > /dev/imx8mq_chardev`

- calls `write()` function

- `unsigned long copy_from_user (void * to, const void __user * from, unsigned long n);`

nxp.com