An introduction to flash memory in Linux

Ezequiel Garcia <ezequiel@collabora.com>

Linux Developer Conference Brazil 2018

Agenda

COLLABORA

- Flash memory: NAND and NOR
- Linux MTD subsystem
- Linux UBI/UBIFS systems

The hardware side

Intro to flash memory

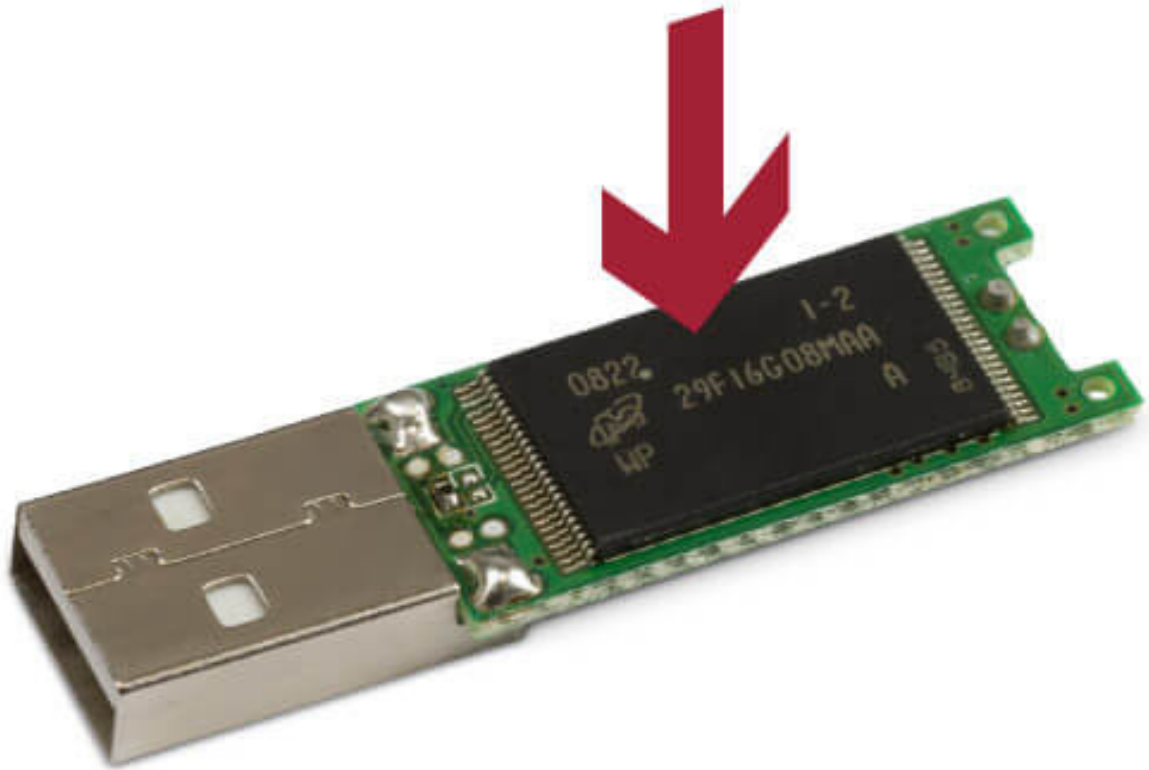
Fujio Masuoka, invented NAND and NOR memory circa 1980.

Intro to flash memory

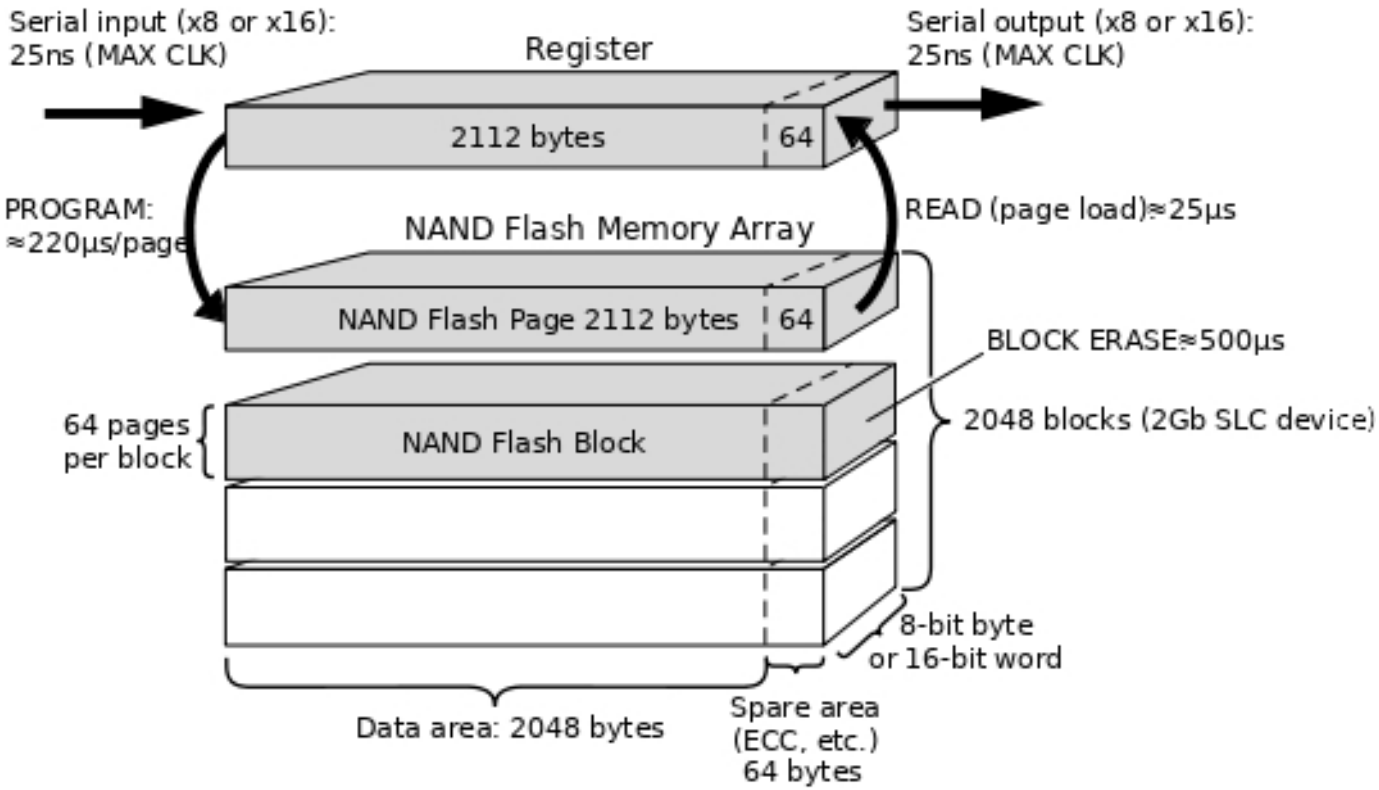
COLLABORA

Intro to flash memory

COLLABORA

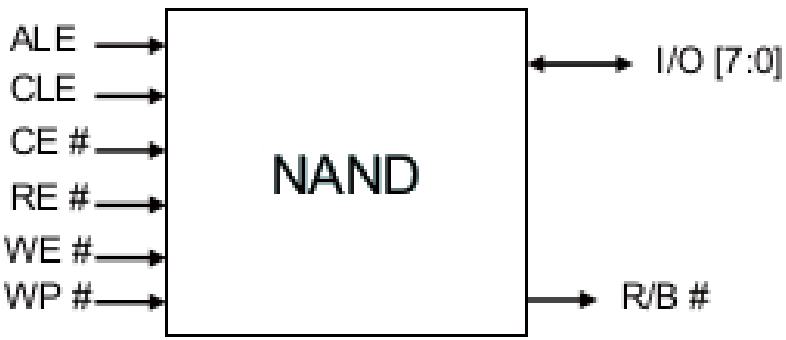


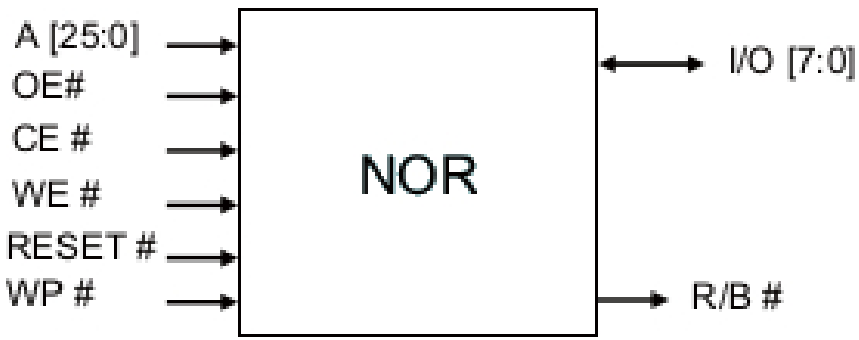
Memory organization



- Minimum I/O size
- Erase size usually larger than I/O size

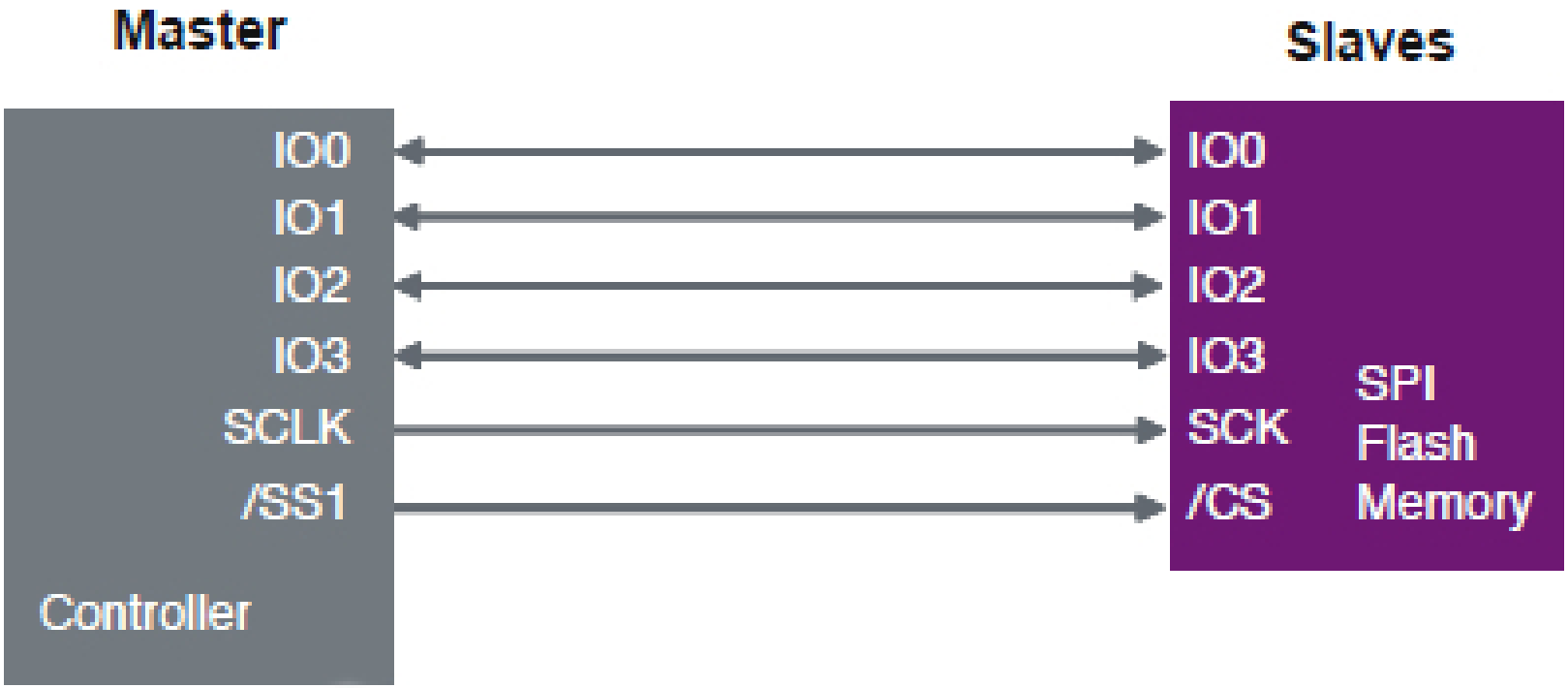
NAND and NOR interface



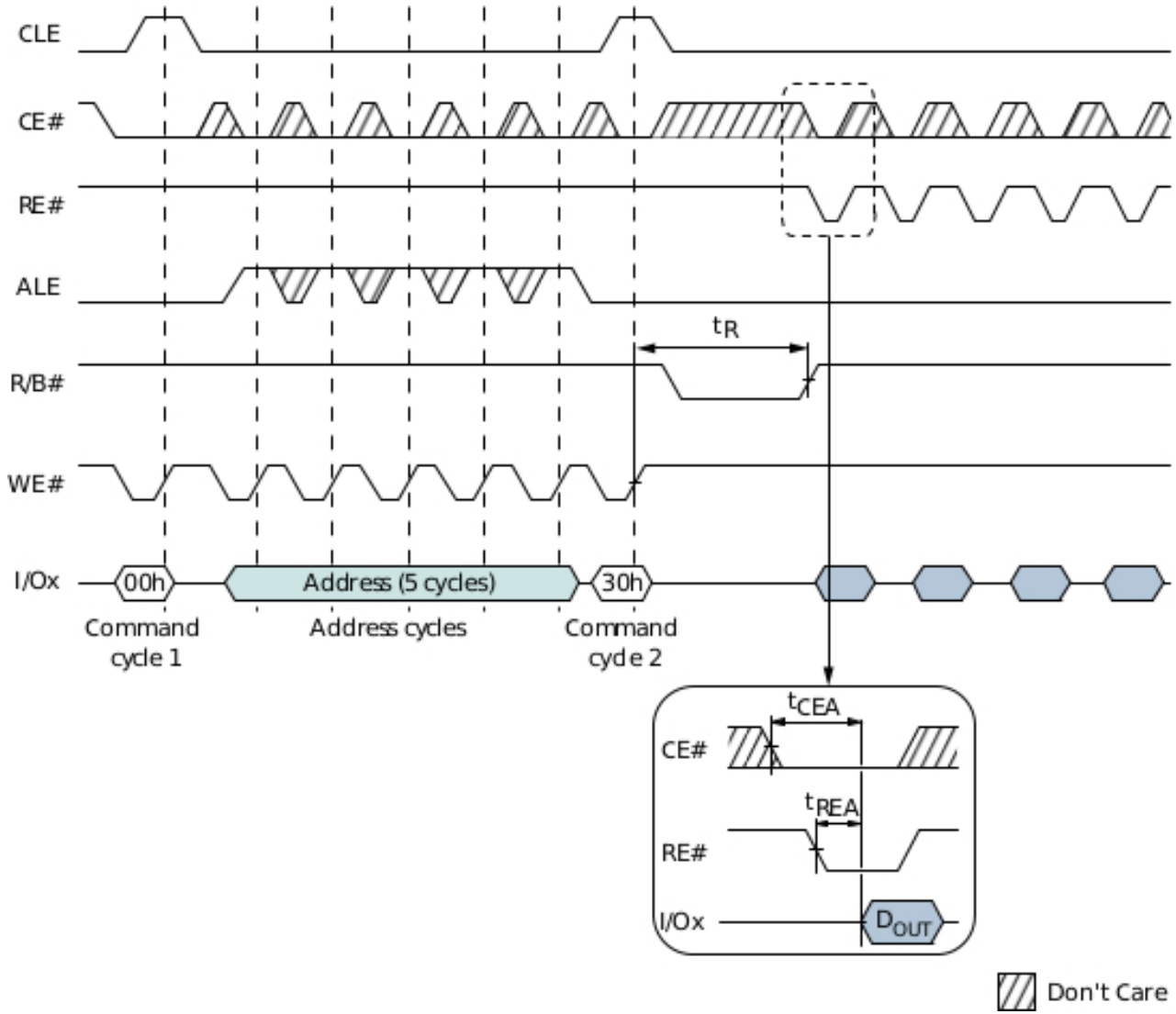


SPI NOR interface

COLLABORA



NAND command example



Constraints of flash memory

- On NAND devices, there is no byte I/O
- Erase blocks go bad after **X** number of erases
- NAND devices also have factory shipped bad blocks
- Random bit-flips appear as a result of stress

Constraints of flash memory

These constraints mean that...

- On NAND devices, you can't eXecute-in-place (without additional hardware)
- Wear leveling is needed to maximize flash life
- Bad block management is needed on the kernel
- Bad block management will be needed on the bootloader too!

The software side

What is our goal?

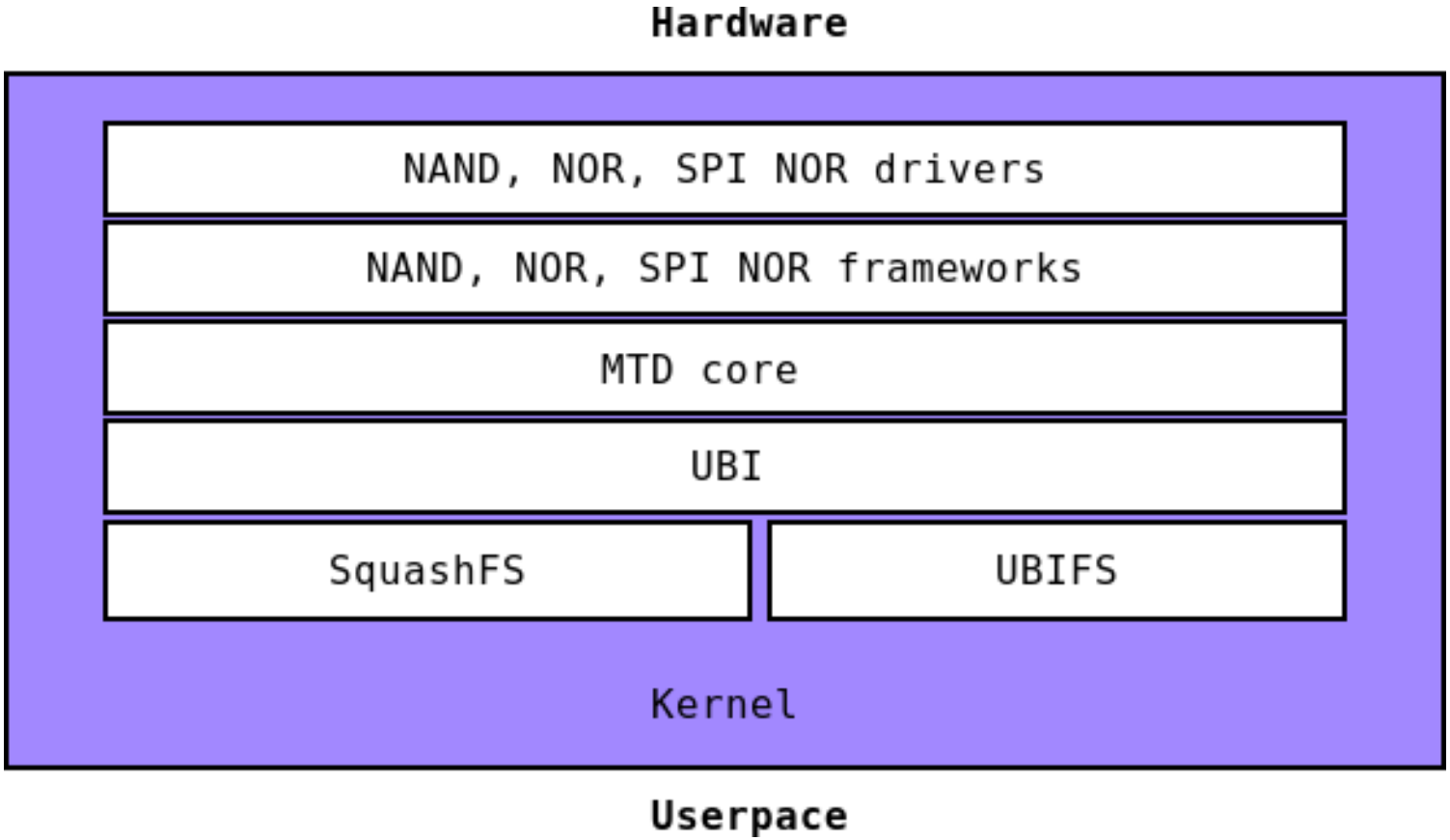
COLLABORA

We want simple UNIX file semantics:

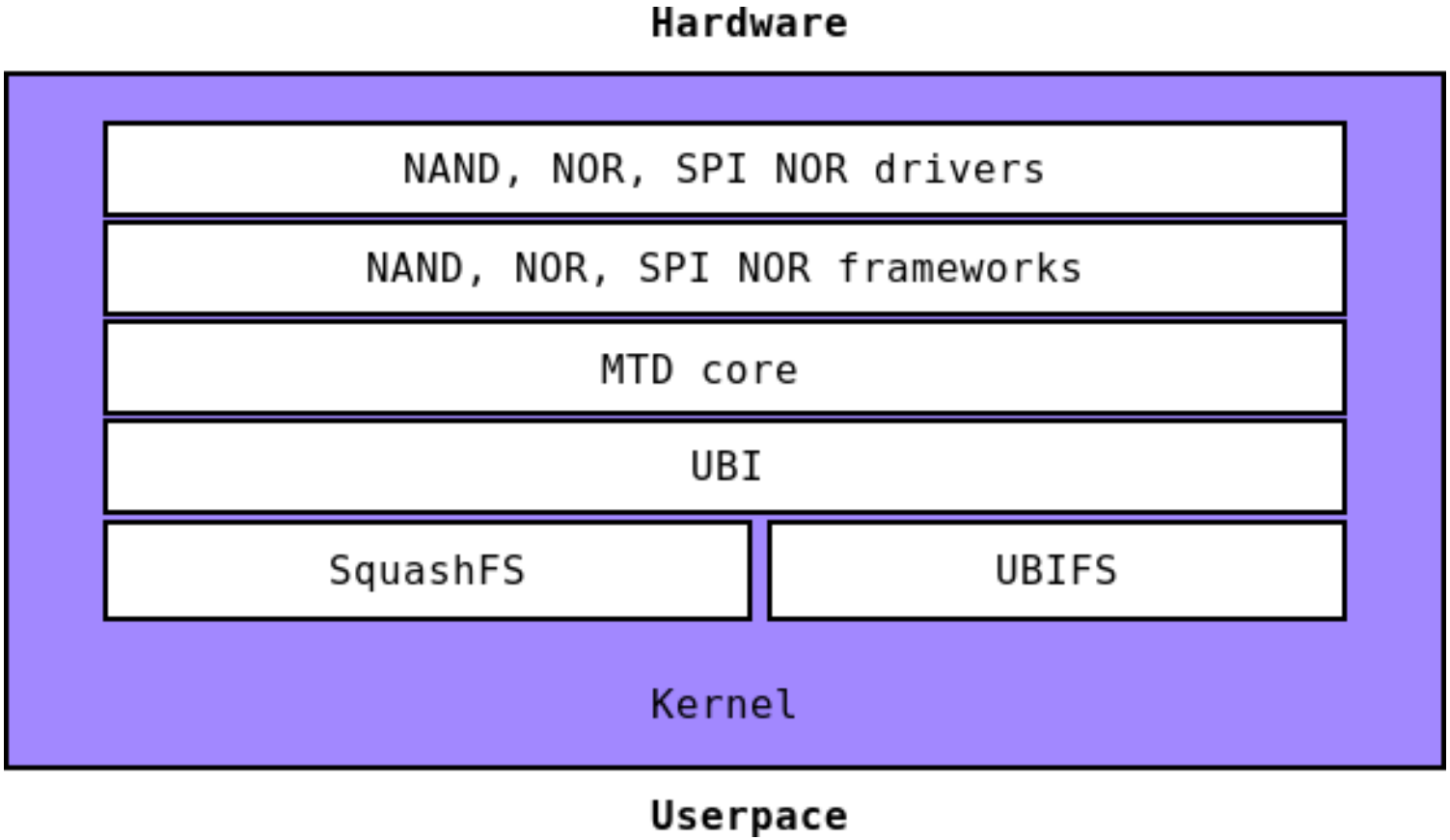
```
fd = open("/foo/some_file");
read(fd, buf, size);
write(fd, buf, size);
close(fd);
```

The stack

COLLABORA



The stack


Hardware
NAND, NOR, SPI NOR drivers
NAND, NOR, SPI NOR frameworks
MTD core
UBI
SquashFS
UBIFS
Kernel
Userpace

The MTD Linux subsystem

The MTD subsystem

- Handles NAND, NOR, SPI NOR and more.
- Devices can be partitioned, and labeled
- Each partitioned exposes a char device interface
- Offers a uniform API for different types of devices
- MTD offers a mechanism for upper layers to manage the bad block table

The MTD subsystem

The userspace API consists in POSIX read/write semantics, plus some ioctls to erase memory, create partitions, get the device geometry, etc.

```
struct erase_info_user erase;

fd = open("/dev/mtd0");
buf = malloc(size);

read(fd, buf, size);
ioctl(fd, MEMERASE, &erase);
write(fd, buf, size);
```

The MTD subsystem

Kernel API, simplified:

linux/mtd/mtd.h:

```
int mtd_erase(struct mtd_info *mtd,
              struct erase_info *instr);

int mtd_read(struct mtd_info *mtd, loff_t from,
             size_t len, size_t *retlen,
             u_char *buf);

int mtd_write(struct mtd_info *mtd, loff_t to,
              size_t len, size_t *retlen,
              const u_char *buf);
```

The MTD subsystem

Defining partitions can be done in several ways.
For instance, via devicetree:

```
partitions {
        compatible = "fixed-partitions";
        partition@0 {
                label = "U-Boot";
                reg = <0 0x200000>;
        };
        partition@400000 {
                label = "Filesystem";
                reg = <0x200000 0xce0000>;
        };
};
```

Also possible to define via kernel parameters and add/removed via ioctls too.

The MTD subsystem

The MTD subsystem is very simple, its goal is to abstract the hardware differences and expose a uniform API to upper layers.
It won't do anything smart, in particular:

- No bad block hiding
- No wear leveling

The UBI and UBIFS Linux subsystems

UBI

Solves almost all the constraints problems! Let's take a look at its features:

- UBI exposes logical volumes
- UBI implements wear-leveling on MTD devices
- Volumes are composed of consecutive **logical** erase blocks
- Volumes may be created, removed, or re-sized
- UBI can also expose a **read-only** block interface
- UBI fastmap (experimental) can improve attach time
- UBI is power-cut tolerant

Although the block interface may be used to mount a read-only filesystem. It's not enough for a read-write filesystem.

UBI API

COLLABORA

```
int ubi_leb_erase(struct ubi_volume_desc *,
                  int lnum);
int ubi_leb_read(struct ubi_volume_desc *,
                 int lnum, char *buf, int off,
                 int len, int check);
int ubi_leb_write(struct ubi_volume_desc *,
                  int lnum, const void *buf,
                  int offset, int len);
int ubi_leb_change(struct ubi_volume_desc *,
                   int lnum, const void *buf,
                   int len);
```

UBIFS

Journal filesystem for UBI volumes,
designed to meet special requirements such as power-cut tolerance.
Its features are:

- scalability
- fast mount (when UBI fastmap is used)
- write-back support
- power-cut tolerance
- fast I/O
- on-the-flight compression
- recoverability
- integrity

See `http://www.linux-mtd.infradead.org/doc/ubifs.html` for detailed information about UBIFS implementation and features.

Some examples to put all of these together

UBI and UBIFS examples

Let's start by preparing a virtual MTD device, backed by a regular file. The commands will need root permission:

```
$ fallocate -l 100M a_dummy_file

$ losetup --show -f a_dummy_file

$ modprobe block2mtd block2mtd=/dev/loop0
```

(Yes, it's a bit hacky, but it does the job pretty well).

UBI and UBIFS examples

Taa-daa!

```
$ mtdinfo /dev/mtd0
mtd0
Name:                            block2mtd: /dev/loop0
Type:                                       ram
Eraseblock size:                            524288 bytes
Amount of eraseblocks:                      200
Minimum input/output unit size:             1 byte
Sub-page size:                              1 byte
Character device major/minor:               90:0
Bad blocks are allowed:                     false
Device is writable:                         true
```

UBI and UBIFS examples

Now let's format the MTD as UBI and create a UBI volume.

```
$ ubiformat /dev/mtd0

$ ubiattach --mtdn 0

$ ubimkvol /dev/ubi0 -s 60MiB -N voltest
```

UBIFS example

COLLABORA

Finally, we create an UBIFS image, write it and mount it.

```
$ mkfs.ubifs -r ~/testree -m 1 \
  -e 524160 -c 200 \
  -o rootfs.ubifs

$ ubiupdatevol /dev/ubi0_0 rootfs.ubifs

$ mount -t ubifs ubi0:voltest /mnt/
```

UBI read-only filesystem examples

COLLABORA

What if we just want a read-only filesystem on top of UBI?

```
fallocate -l 50M rootfs.ext4
mkfs.ext4 rootfs.ext4

ubiupdatevol /dev/ubi0_0 rootfs.ext4

ubiblock --create /dev/ubi0_0

mount /dev/ubiblock0_0 /mnt

touch /mnt/some_new_stuff
```

References

- Micron NAND Flash 101: An Introduction to NAND
  `https://www.ece.umd.edu/~blj/CS-590.26/micron-tn2919.pdf`
- Linux MTD documentation
  `http://www.linux-mtd.infradead.org/doc/general.html`
- An Introduction to SPI-NOR Subsystem
  `https://youtu.be/GJay1MCUvfE`
- the internet

Thanks for attending!

Q & A

ezequiel@collabora.com