Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

File Systems

Ext2/3, squash and tmpfs

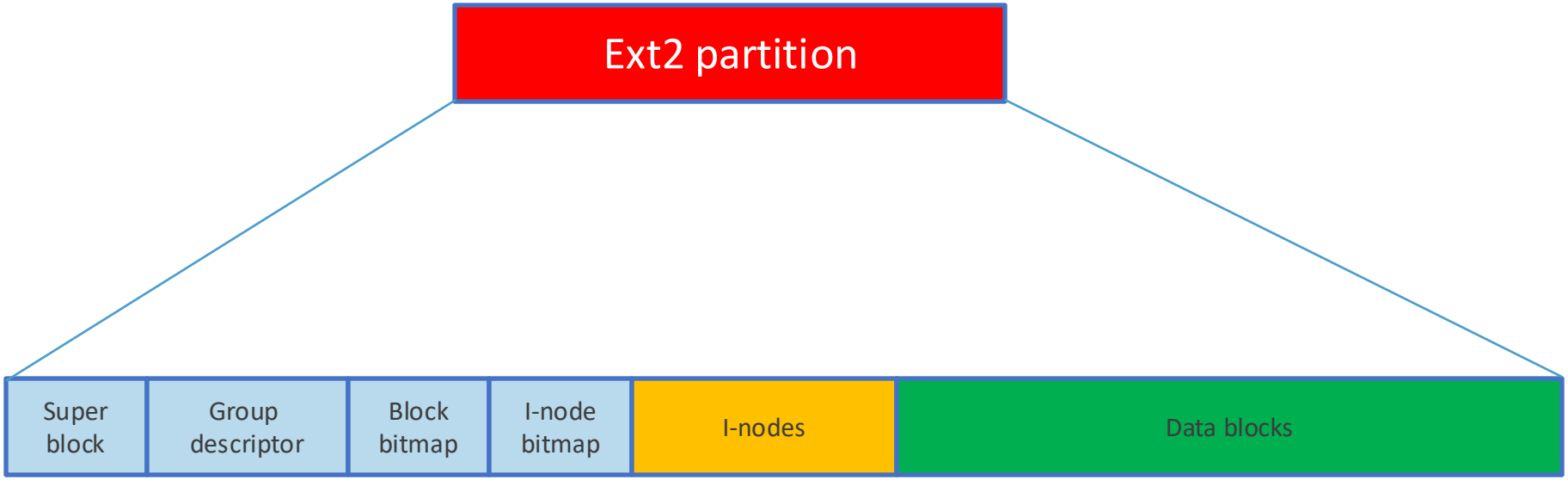
Hes·so FRIBOURG FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

Ext2 file system

- **Ext2** or second extended filesystem is a file system for the Linux kernel
  - Ext2 is **not** a journaled file system
  - Ext2 uses block mapping in order to reduce file fragmentation (it allocates several free blocks → reduce fragmentation).
  - After an unexpected power failure or system crash (also called an unclean system shutdown), each mounted ext2 file system on the machine must be checked for consistency with the **`e2fsck`** program.

Ext2 file system

An ext2 Linux disk partition contains a file system with this layout:



Ext2 file system

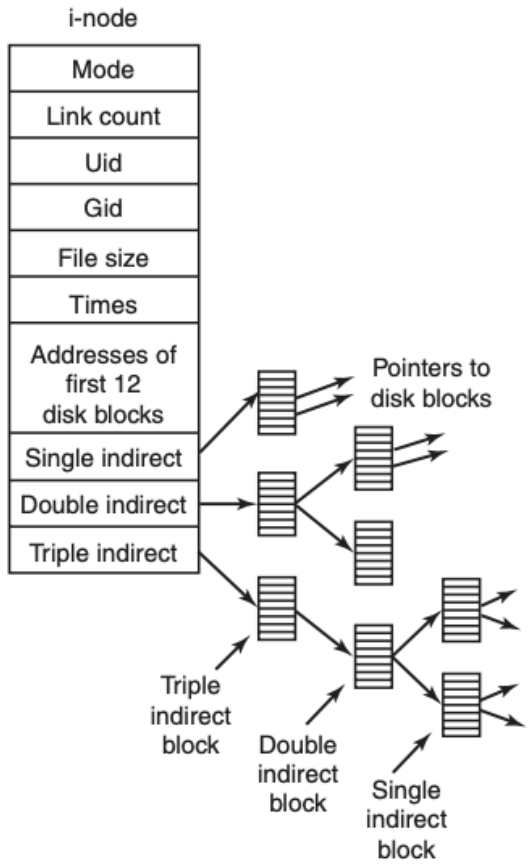
- The first block is the superblock. It contains information about the layout of the file system, including the number of i-nodes, the number of disk blocks, and the start of the list of free disk blocks.
- Next comes the group descriptor, which contains information about the
  - The location of the bitmaps (free blocks and free i-nodes).
  - The number of free blocks and i-nodes in the group, and the number of directories in the group.
- Next come block bitmap and I-node bitmap. These two bitmaps are used to keep track of the free blocks and free i-nodes.
- Next comes the i-nodes. Each file has an i-node. An i-node contains information to locate all the disk blocks that hold:
  - the file's data
  - or the name of files or directories
- And finally comes the data blocks. These blocks contain:
  - The file's data
  - Or the name of files or directories

I-node structure

```
                Mode    [0100644]
             User ID    [0]
            Group ID    [0]
                Size    [7]
       Creation time    [1690953428]
   Modification time    [1690953428]
         Access time    [1690958443]
       Deletion time    [0]
          Link count    [1]
    Block count high    [0]
         Block count    [8]
          File flags    [0x0]
          Generation    [0x83bbaed2]
            File acl    [0]
  High 32bits of size   [0]
     Fragment address   [0]
      Direct Block #0   [512]
      Direct Block #1   [0]
      Direct Block #2   [0]
      Direct Block #3   [0]
      Direct Block #4   [0]
      Direct Block #5   [0]
      Direct Block #6   [0]
      Direct Block #7   [0]
      Direct Block #8   [0]
      Direct Block #9   [0]
     Direct Block #10   [0]
     Direct Block #11   [0]
       Indirect Block   [0]
Double Indirect Block   [0]
Triple Indirect Block   [0]
```

File's attributes

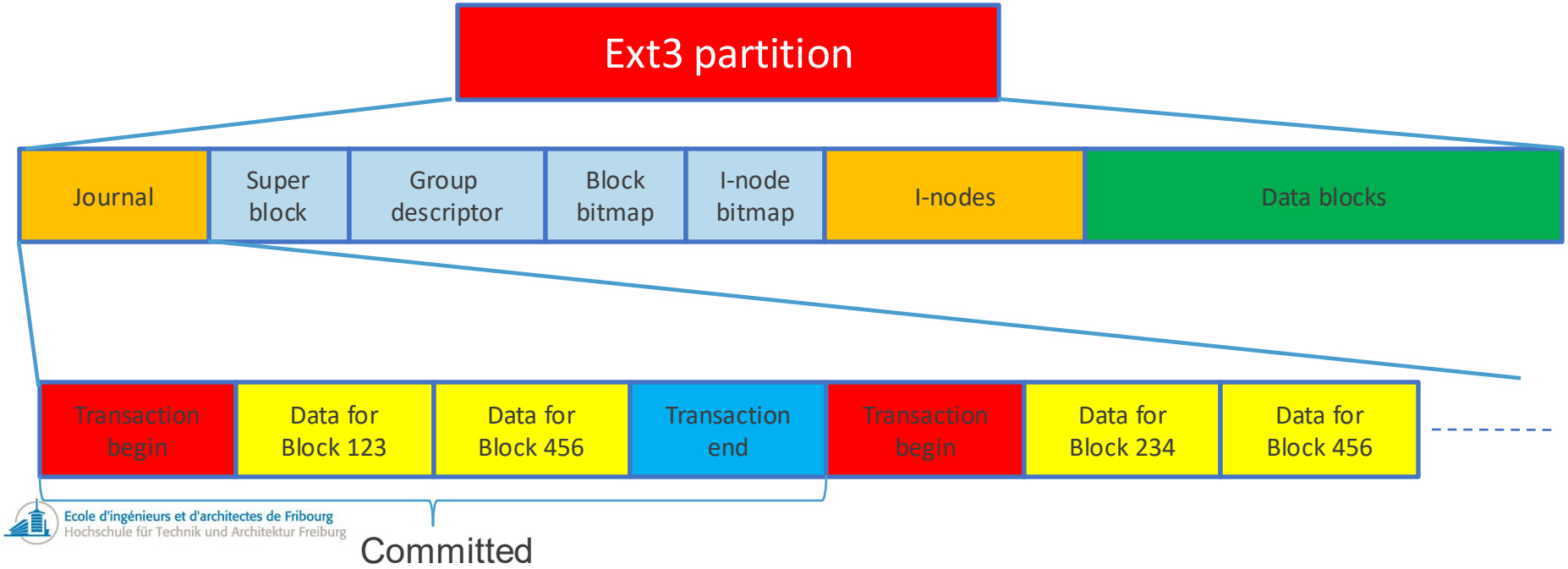
Data blocks where the content of files is or where the name of the files or directories are



Ext3 file system

Ext3 file system replaces ext2

- It was merged in the 2.4.15 kernel (November 2001)
- Ext3 is compatible with ext2
- An Ext3 partition has a Journal where all written data is memorized in the Journal until the data is committed
- Journal is logically a fixed-size, circular array:
  - Implemented as a special file with a hard-coded i-node number
  - Each journal transaction is composed of a begin marker, log, and end marker



Ext4 file system

Ext4 file system

- ext4 is backward compatible with ext3 and ext2, making it possible to mount ext3 and ext2 as ext4
- ext4 is included in the kernel 2.6.28 on 11 October 2008
- ext4 supports Large file system:
  - volume max: 2^{60} bytes
  - File max: 2^{40} bytes
- ext4 uses extents (as opposed to the traditional block mapping scheme used by ext2 and ext3), which improves performance when using large files and reduces metadata overhead for large files.
- Extents represent contiguous blocks of storage, for instance 128 MB of contiguous 4-KB blocks vs. individual storage blocks, as referenced in ext2

Ext4 commands

Create a partition (rootfs), start 64MB, length 256MB

```
sudo parted /dev/sdb mkpart primary ext4 131072s  655359s # 's' after the number means sectors
```

Format the partition with the volume label = rootfs

```
sudo mkfs.ext4 /dev/sdb1 -L rootfs
```

Modify (on the fly) the ext4 configuration

```
sudo tune2fs <options> /dev/sdb1
```

check the ext4 configuration

```
mount
sudo tune2fs -l /dev/sdb1
sudo dumpe2fs /dev/sdb1
```

mount an ext4 file system

```
mount -t ext4 /dev/sdb1  /mnt/test    // with default options
mount -t ext4 -o defaults,noatime,discard,nodiratime,data=writeback,acl,user_xattr
/dev/sdb1    /mnt/test
```

Ext4, ext2 – buildroot, busybox

In order to have mkfs.ext2 and tune2fs program on the NanoPi, it is necessary to configure busybox

```
cd /buildroot

make busybox-menuconfig
   Go to "Linux Ext2 FS Progs" → [*] tune2fs
   Go to "Linux System Utilities" → [*] mkfs.ext2
```

Ext4 mount options and MMC/SD-Card

- filesystem options can be activated with the mount command (or via /etc/fstab file).
- These options can be modified with tune2fs command
- Journaling: the journaling guarantees the data consistency, but it reduces the file system performances
- MMC/SD-Card constraints: In order to improve the longevity of MMC/SD-Card, it is necessary to **reduce the unnecessary writes**
- Mount options to reduce the unnecessary writes `(man mount)`:
  - noatime: Do not update inode access times on this filesystem
  - nodiratime: Do not update directory inode access times on this filesystem
  - relatime: this option can replace the noatime and nodiratime if an application needs the access time information (like mutt)

Ext4 mount options and MMC/SD-Card

Mount options for the journaling (`man ext4`):

- Data=journal: All data is committed into the journal prior to being written into the main filesystem (It is the safest option in terms of data integrity and reliability, though maybe not so much for performance)
- Data=ordered: This is the default mode. All data is forced directly out to the main file system before the metadata being committed to the journal
- Data=writeback: Data ordering is not preserved - data may be written into the main filesystem after its metadata has been committed to the journal.

Ext4 mount options and MMC/SD_SDCard

- Discard: Use discard requests to inform the storage that a given range of blocks is no longer in use. A MMC/SD-Card can use this information to free up space internally, using the free blocks for wear-levelling.

- acl: Support POSIX Access Control Lists

- default: rw, suid, dev, exec, auto, nouser, and async
  - rw: read-write
  - suid: Allow set-user-identifier or set-group-identifier bits
  - dev: Interpret character or block special devices on the filesystem
  - exec: Permit execution of binaries
  - auto: Can be mounted with the -a option (mount –a)
  - nouser: Forbid an ordinary (i.e., non-root) user to mount the filesystem
  - async: All I/O to the filesystem should be done asynchronously

/etc/fstab file

- File /etc/fstab contains descriptive information about the filesystems the system can mount

- **NanoPi example: /etc/fstab**

```
# cat /etc/fstab
# /etc/fstab: static file system information.
#
# <file system> <mount pt>     <type>   <options>                    <dump> <pass>
/dev/root       /              ext4     rw,noauto                         0    1
proc            /proc          proc     defaults                          0    0
devpts          /dev/pts       devpts   defaults,gid=5,mode=620           0    0
tmpfs           /dev/shm       tmpfs    mode=0777                         0    0
tmpfs           /tmp           tmpfs    defaults,nosuid,noexec,nodev,rw   0    0
sysfs           /sys           sysfs    defaults                          0    0
```

/etc/fstab file

<file system>: block special device or remote filesystem to be mounted

<mount pt>: mount point for the filesystem

<type>: the filesystem type

<options>: mount options associated with the filesystem

<dump>: used by the dump (backup filesystem) command to determine which filesystems need to be dumped (0 -> no backup).

<pass>: used by the fsck (8) program to determine the order in which filesystem checks are done at reboot time. The root filesystem should be specified with 1, and other filesystems should have a 2. if <pass> is not present or equal 0 -> fsck will assume that the filesystem is not checked.

Field options: It contains at least the type of mount plus any additional options appropriate to the filesystem type.

Common for all types of file system are the options (man mount):

auto	Can be mounted with the -a option (mount -a)
defaults	Use default options: rw, suid, dev, exec, auto, nouser, and async.
nosuid	Do not allow set-user-identifier or set-group-identifier bits to take effect.
noexec	Do not allow direct execution of any binaries on the mounted file system
nodev	Do not interpret character or block special devices on the file system.

Squash File System [1], [2], [3], [4]

- Squashfs is a compressed read-only filesystem for Linux.
- Squashfs versions:
  - Squashfs 2.0 and squashfs 2.1: 2004, kernel 2.2
  - Squashfs 3.0: 2006, kernel 2.6.12
  - Squashfs 4.2: 2011, kernel 2.6.29

- It uses gzip, lzma, lzo, lz4 and xz compression to compress files, i-nodes and directories.

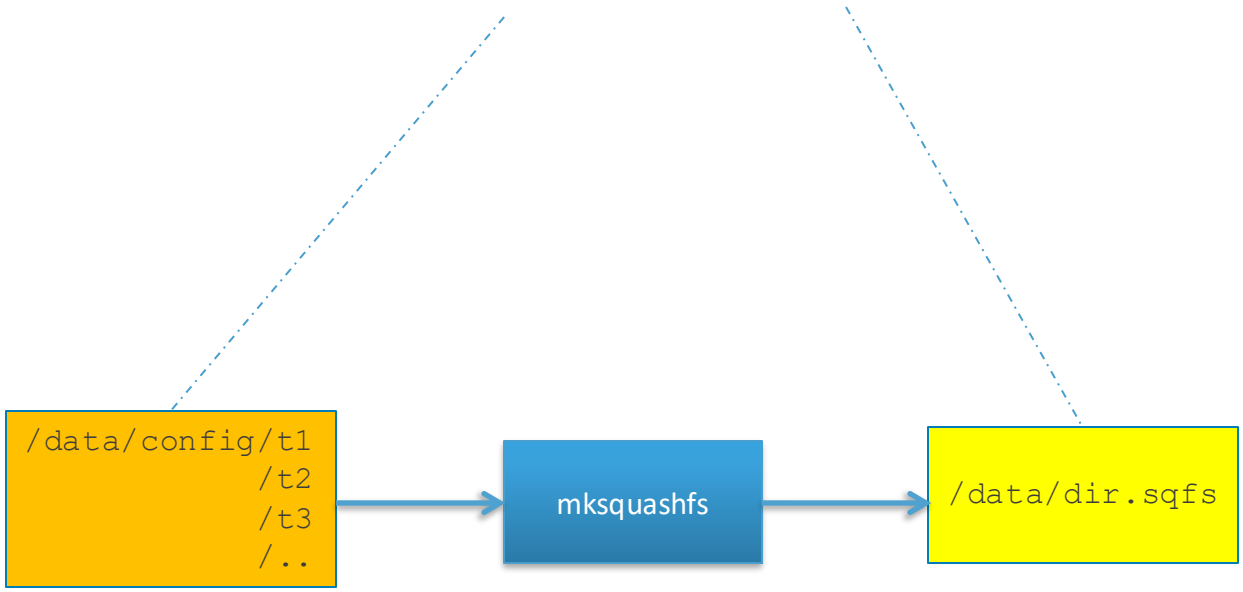
- SquashFS 4.0 supports 64 bits filesystems and files (larger than 4GB), full uid/gid information, hard links and timestamps.

- Squashfs is intended for general read-only filesystem use, for archival use, and in embedded systems with small processors where low overhead is needed

Create and use squashed file systems [2]

1. Create the squashed file system `dir.sqsh` from the regular directory `/data/config/`

```
bash# mksquashfs /data/config/ /data/dir.sqsh
```



Create and use squashed file systems [2]

2. The mount command is used with a loopback device in order to read the squashed file system `dir.sqsh`

```
bash# mkdir /mnt/dir
bash# mount -o loop -t squashfs /data/dir.sqsh /mnt/dir
bash# ls /mnt/dir
```

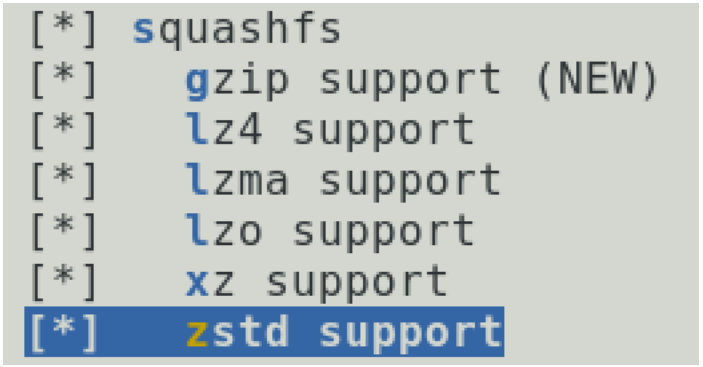
3. It is possible to copy the `dir.sqsh` to an unmounted partition (e.g. /dev/sdb2) with the `dd` command and next to mount the partition as squashfs file system

```
bash# umount /dev/sdb2
bash# dd if=dir.sqsh of=/dev/sdb2
bash# mount /dev/sdb2 /mnt/dir -t squashfs
bash# ls /mnt/dir
```

Squashfs - NanoPi - Buildroot

```
cd workspace/nanopi/buildroot
make menuconfig
```

- Go to: Target Packages → Filesystem and Flash utilities : choose squashfs



Tmpfs [1]

- tmpfs is a file system which keeps all files in virtual memory.
- Everything in tmpfs is temporary in the sense that no files will be created on your hard drive. If you unmount a tmpfs instance, everything stored therein is lost.
- tmpfs puts everything into the kernel internal caches and grows and shrinks to accommodate the files it contains and is able to swap unneeded pages out to swap space. It has maximum size limits which can be adjusted on the fly via 'mount -o remount ...'
- If you compare it to ramfs you gain swapping and limit checking. Ramdisks cannot swap and you do not have the possibility to resize them

Tmpfs [1]

- glibc 2.2 and above expects tmpfs to be mounted at /dev/shm for POSIX shared memory (shm_open, shm_unlink). Adding the following line to /etc/fstab should take care of this:

```
tmpfs    /dev/shm        tmpfs    defaults        0 0
```

- It is very convenient to mount /tmp, /var/tmp as tmpfs file system. Add this line to /etc/fstab

```
tmpfs    /tmp            tmpfs    mode=1777       0 0
tmpfs    /var/tmp        tmpfs    mode=1777       0 0
```

Devtmpfs [1]

- devtmpfs is a file system with automatically populates nodes files (/dev/…) known by the kernel.
- This means, it is not necessary to have udev running nor to create a static /dev layout with additional, unneeded and not present device nodes.
- Instead, the kernel populates the appropriate information based on the known devices.
- The kernel executes this command: `mount -n -t devtmpfs devtmpfs /dev`

- `/dev` is automatically populated by the kernel with its known devices

```
# ls /dev
autofs              ptypf               tty47
btrfs-control       random              tty48
bus                 rtc0                tty49
console             shm                 tty5
cpu_dma_latency     snapshot            tty50
…
```