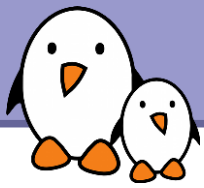




Update on filesystems for flash storage

Michael Opdenacker.
Bootlin
<https://bootlin.com/>





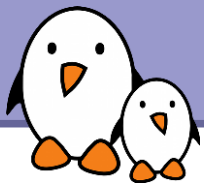
About this document

This document is released under the terms
of the Creative-Commons BY-SA 3.0 license:

<http://creativecommons.org/licenses/by-sa/3.0/>

Documents updates can be found or described on

<https://bootlin.com/pub/conferences/2008/elce/>



Contents

Introduction

Available flash filesystems

Our benchmarks

Best choices

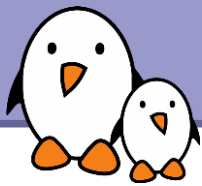
Experimental filesystems

Advice for flash-based block devices



Update on filesystems for flash storage

Introduction

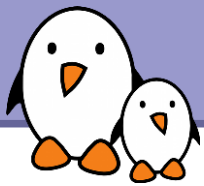


Flash storage

We are talking about flash chips, accessed by the Linux kernel as Memory Technology devices.

Compact Flash, MMC/SD, Memory Stick cards, together with USB flash drives and Solid State Drives (SSD), are interfaced as block storage, like regular hard disks.

At the end, we will say a few words about dealing with the second category.



Existing solutions

For the last years, only 2 filesystem choices for flash storage

jffs2

Wear leveling, ECC

Power down resistant

Compression

Huge mount times

Rather big memory usage

Mainstream support

yaffs2

Wear leveling, ECC

Power down resistant

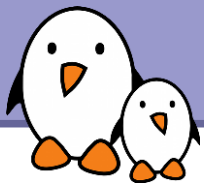
No compression

Very quick mount time

Programmed by Wookies
(at least 1)

Available as a Linux patch.

2 solutions, but far from being perfect!



Election time!

At last, new choices have been developed.

LogFS

New filesystem for MTD
storage

UBI

New layer managing erase
blocks and wear leveling

UBIFS

New filesystem taking
advantage of UBI's
capabilities

AXFS

Advanced XIP FileSystem

How do they compare
to existing solutions?

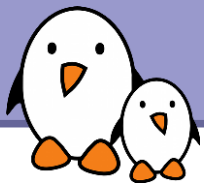
Mounting time

Access speed

Memory usage

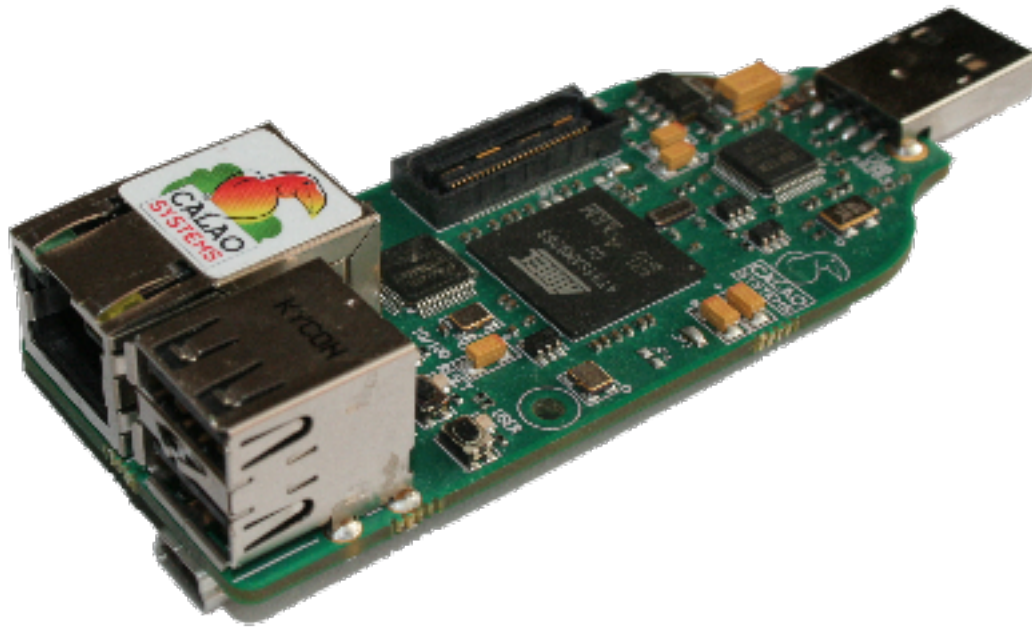
CPU usage

Size?



Test hardware

Calao Systems USB-A9263



AT91SAM9263 ARM CPU

64 MB RAM

256 MB flash

2 USB 2.0 host

1 USB device

100 Mbit Ethernet port

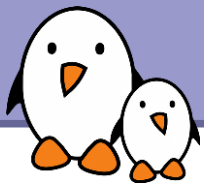
Powered by USB!

Serial and JTAG through
this USB port.

Multiple extension boards.

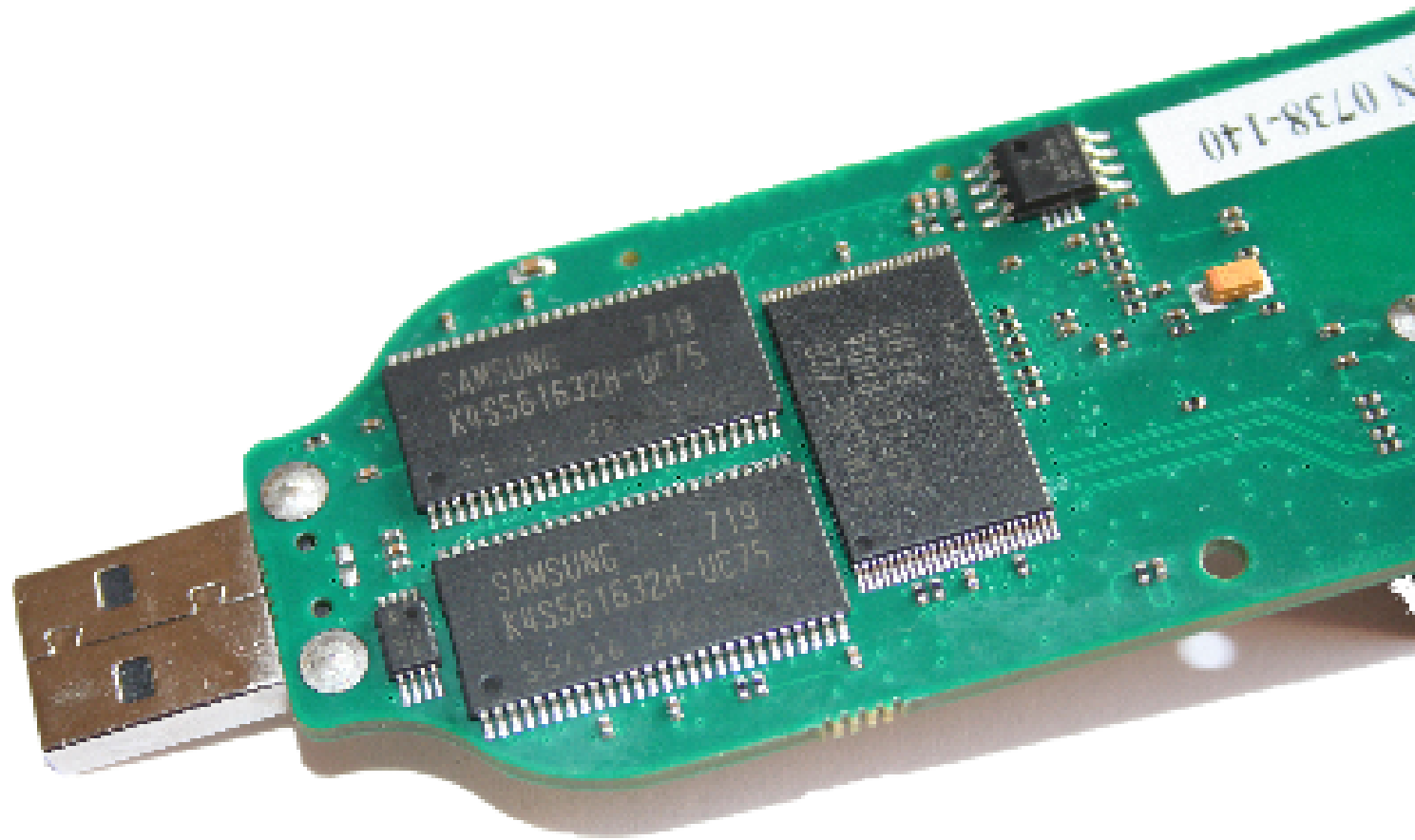
162 EUR

Supported by Linux 2.6.27!

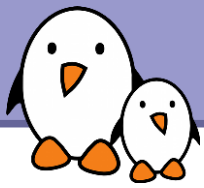


Flash chips

NAND device: Manufacturer ID: 0xec, Chip ID: 0xda
(Samsung NAND 256MiB 3,3V 8-bit)

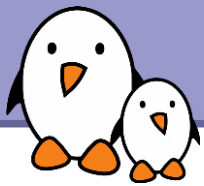


Samsung's reference: K4S561632H-UC75



Update on filesystems for flash storage

Available flash filesystems

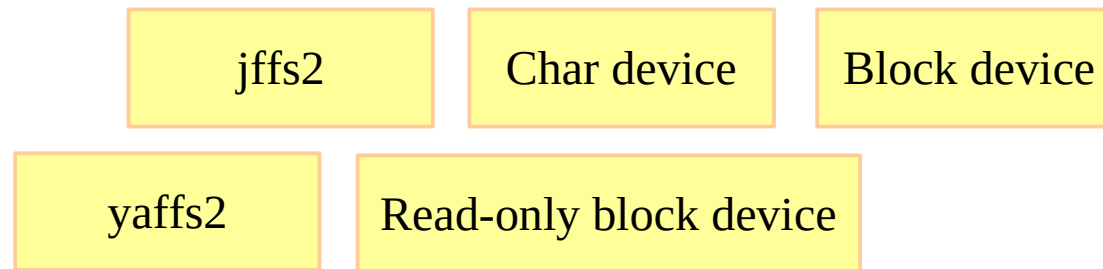


The MTD API

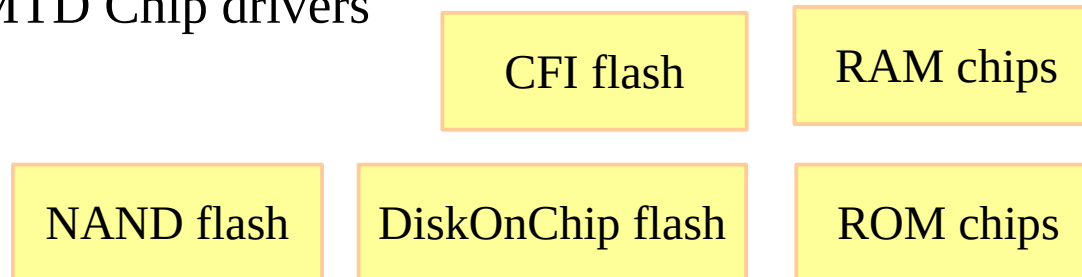
A Linux kernel API to access Memory Technology Devices
Abstracts the specifics of MTD devices: erase blocks, page size...

Linux filesystem interface

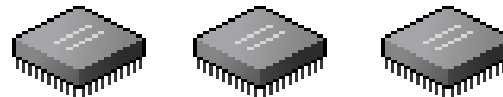
MTD “User” modules



MTD Chip drivers



Memory devices hardware





MTD - How to use

Creating the device nodes

Char device files

`mknod /dev/mtd0 c 90 0 (bad idea!!!)`

`mknod /dev/mtd1 c 90 2 (Caution!)`

`mknod /dev/mtd2 c 90 4 (Caution)`

Block device files

`mknod /dev/mtdblock0 b 31 0 (bad idea!!!)`

`mknod /dev/mtdblock0 b 31 2`

`mknod /dev/mtdblock0 b 31 2`



Today's standard filesystem for MTD flash

Nice features:

- On the fly compression. Saves storage space and reduces I/O.

- Power-down reliable.

- Implements wear-leveling

Drawbacks: doesn't scale well

- Mount time depending on filesystem size:
the kernel has to scan the whole filesystem at mount time, to read which block belongs to each file.

- Keeping this information in RAM is memory hungry too.

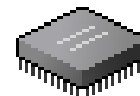
Standard file
API



JFFS2
filesystem



MTD driver



Flash chip



New jffs2 features

CONFIG_JFFS2_SUMMARY

Reduces boot time by storing summary information.

New jffs2 compression options:

- Now supports lzo compression, and not only zlib
(and also the rtime and rubin compressors)

- Can try all compressors and keep
the one giving the best results

- Can also give preference to lzo, to the expense of size,
because lzo has the fastest decompression times.



jffs2 - How to use

Compile mtd-tools if needed:

```
git-clone git://git.infradead.org/mtd-utils.git
```

Erase and format a partition with jffs2:

```
flash_eraseall -j /dev/mtd2
```

Mount the partition:

```
mount -t jffs2 /dev/mtdblock2 /mnt/flash
```

Fill the contents by writing

Or, use an image:

```
nandwrite -p /dev/mtd2 rootfs.jffs2
```



yaffs2

<http://www.yaffs.net/>

Supports both NAND and NOR flash

No compression

Wear leveling, ECC, power failure resistant

Fast boot time

Code available separately through CVS
(Dual GPL / Proprietary license
for non Linux operating systems)

Standard file
API

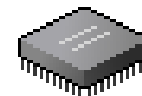
— — — —

YAFFS2
filesystem

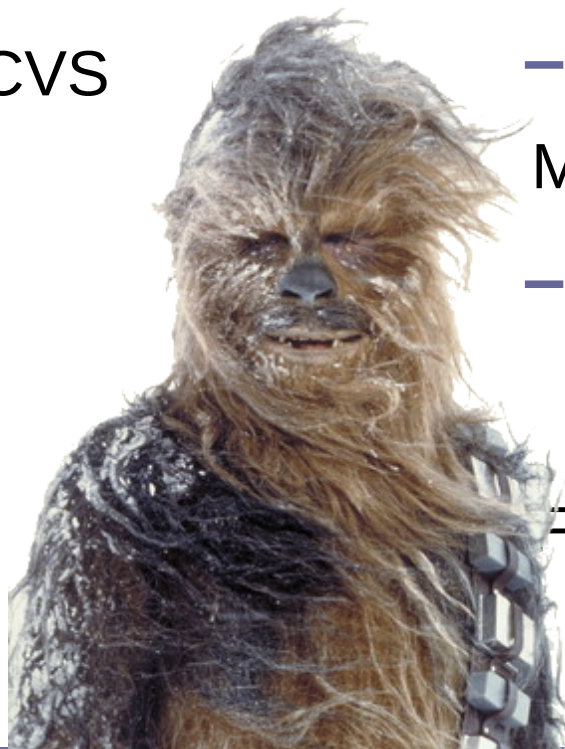
— — — —

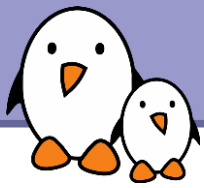
MTD driver

— — — —



Flash chip





yaffs2 - How to use

Erase a partition:

```
flash_eraseall /dev/mtd2
```

Format the partition:

```
sleep (any command can do!)
```

Mount the partition:

```
mount -t yaffs2 /dev/mtdblock2 /mnt/flash
```



UBI

Unsorted Block Images

<http://www.linux-mtd.infradead.org/doc/ubi.html>

Volume management system on top of MTD devices.

Allows to create multiple logical volumes
and spread writes across all physical blocks.

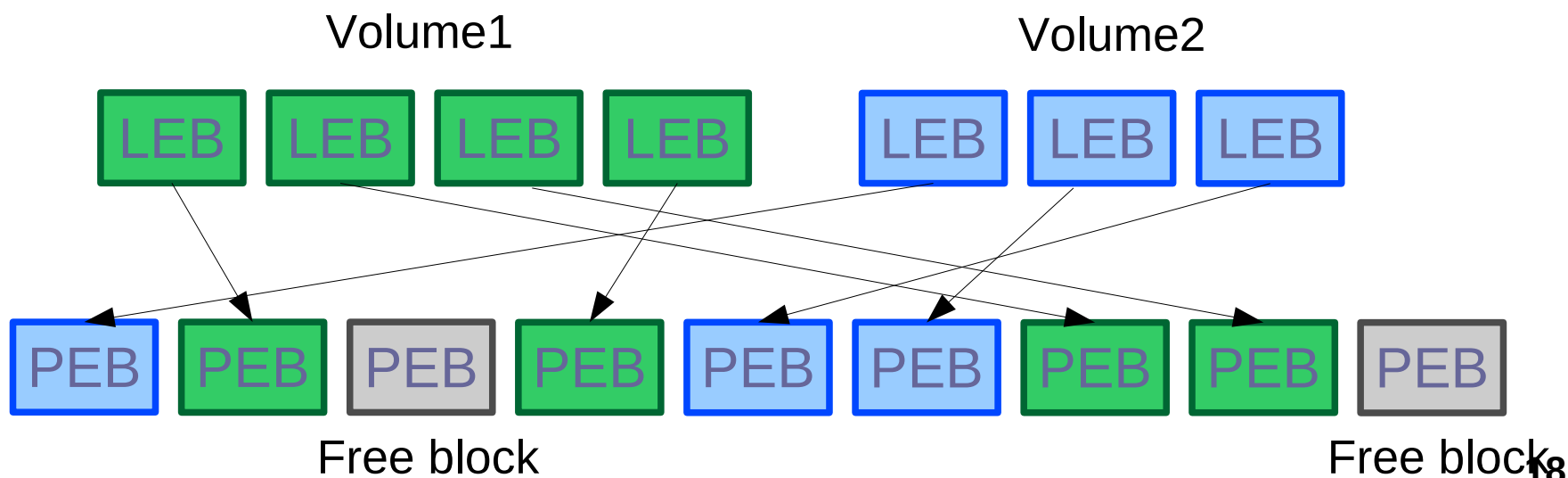
Takes care of managing the erase blocks and wear leveling.
Makes filesystem easier to implement.

UBI

Logical
Erase Blocks

MTD

Physical
Erase Blocks





UBI - How to use (1)

First, erase your partition (NEVER FORGET!)

```
flash_eraseall /dev/mtd1
```

First, format your partition:

```
ubiformat /dev/mtd1 -s 512
```

(possible to set an initial erase counter value)

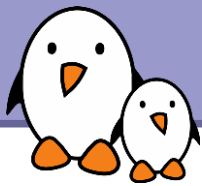
See <http://www.linux-mtd.infradead.org/faq/ubi.html> if you face problems

Need to create a `/dev/ubi_ctrl` device (if you don't have udev)

Major and minor number allocated in the kernel. Find these numbers in `/sys/class/misc/ubi_ctrl/dev/` (e.g.: 10:63)

Or run `ubinfo`:

```
UBI version: 1
Count of UBI devices: 1
UBI control device major/minor: 10:63
Present UBI devices: ubi0
```



UBI - How to use (2)

Attach UBI to one (of several) of the MTD partitions:

```
ubiattach /dev/ubi_ctrl -m 1
```

Find the major and minor numbers used by UBI:

```
cat /sys/class/ubi/ubi0/dev (e.g. 253:0)
```

Create the UBI device file:

```
mknod /dev/ubi0 c 253 0
```



UBIFS

<http://www.linux-mtd.infradead.org/doc/ubifs.html>

The next generation of the jffs2 filesystem,
from the same linux-mtd developers.

Available in Linux 2.6.27

Works on top of UBI volumes

Standard file
API

— — — —

UBIFS

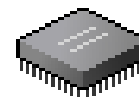
— — — —

UBI

— — — —

MTD driver

— — — —



Flash chip



UBIFS - How to use

Creating

```
ubimkvol /dev/ubi0 -N test -s 116MiB
```

```
mount -t ubifs ubi0:test /mnt/flash
```

Deleting

```
umount /mnt/flash
```

```
ubirmvol /dev/ubi0 -N test
```

Detach the MTD partition:

```
ubidetach /dev/ubi_ctrl -m 1
```



LogFS

<http://logfs.org/logfs/>

Also developed as a replacement for jffs2

We announced we would cover it, but its latest version only supports 2.6.25. Our board only supports 2.6.21, 2.6.27 and beyond, and the 2.6.25 LogFS patch doesn't compile in 2.6.27!

Anyway, LogFS is not ready yet for production.

Will it ever be, now that jffs2 has a valuable replacement?
Competition is useful though.



AXFS

Advanced XIP FileSystem for Linux

<http://axfs.sourceforge.net/>

Allows to execute code directly from flash,
instead of copying it to memory.

As XIP is not possible with NAND flash, works best when there
is a mix of NOR flash (for code) and NAND (for non XIP
sections).

Currently posted for review / inclusion in the mainstream Linux
kernel. To be accepted in 2.6.29 or later?

Not benchmarked here. We only have NAND flash anyway.



SquashFS

<http://squashfs.sourceforge.net/>

Filesystem for block storage!?

But read-only! No problem with managing erase blocks and wear-leveling. Fine to use with the mtdblock driver.

You can use it for the read-only sections in your filesystem.

Actively maintained. Releases for many kernel versions (recent and old).

Currently submitted by Philip Lougher for inclusion in mainline. Don't miss his talk tomorrow!



SquashFS - How to use

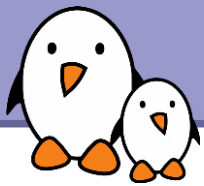
Very simple!

On your workstation, create your filesystem image
(example: `120m/` directory in our benchmarks)
`mfsquashfs 120m 120m.sqfs`

Erase your flash partition:
`flash_eraseall /dev/mtd2`

Make your filesystem image available to your device (NFS,
copy, etc.) and flash your partition:
`dd if=120m.sqfs of=/dev/mtdblock2`

Mount your filesystem:
`mount -t squashfs /dev/mtdblock2 /mnt/flash`



Update on filesystems for flash storage

Benchmarks



Benchmark overview

Compared filesystems:

jffs2, default options

jffs2, lzo compression only

yaffs2

ubifs, default options

ubifs, no compression

squashfs

Different MTD partitions

8M

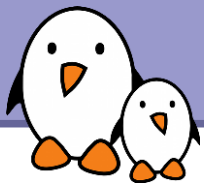
32M

120M

Corresponding to most
embedded device scenarios.

Partitions filled at about 85%

All tested with Linux 2.6.27.

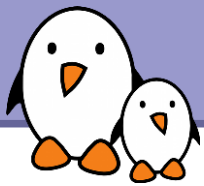


Read and mounting experiments

Mounting an arm Linux root filesystem,
taken from the OpenMoko project.

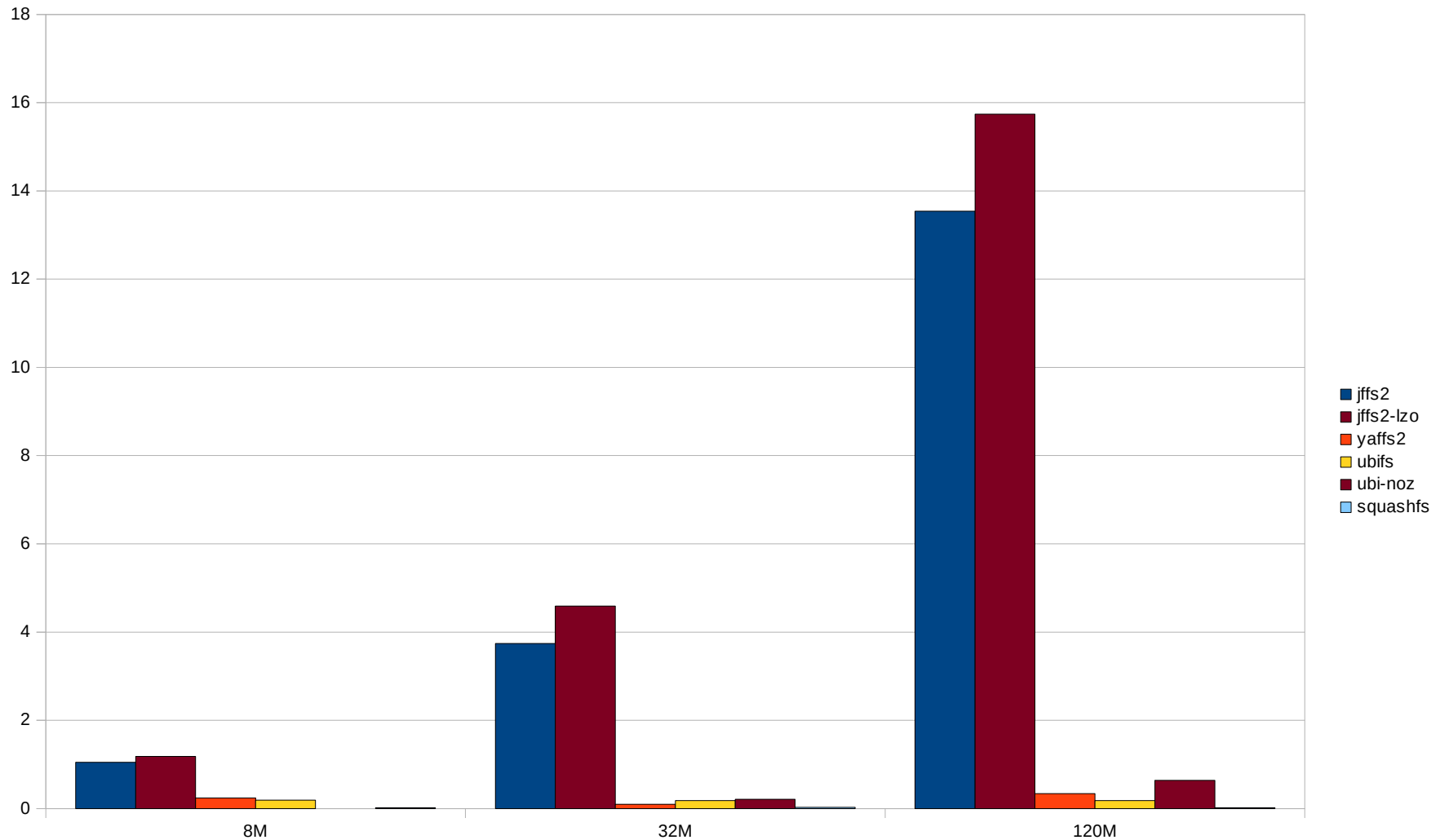
Advantages: mainly contains compressible files (executables
and shared libraries).

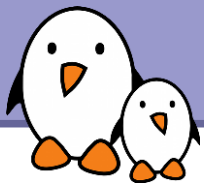
Represents a very important scenario: booting on a filesystem
in flash. Mounting and file access time are major
components of system boot time.



Mount time (seconds)

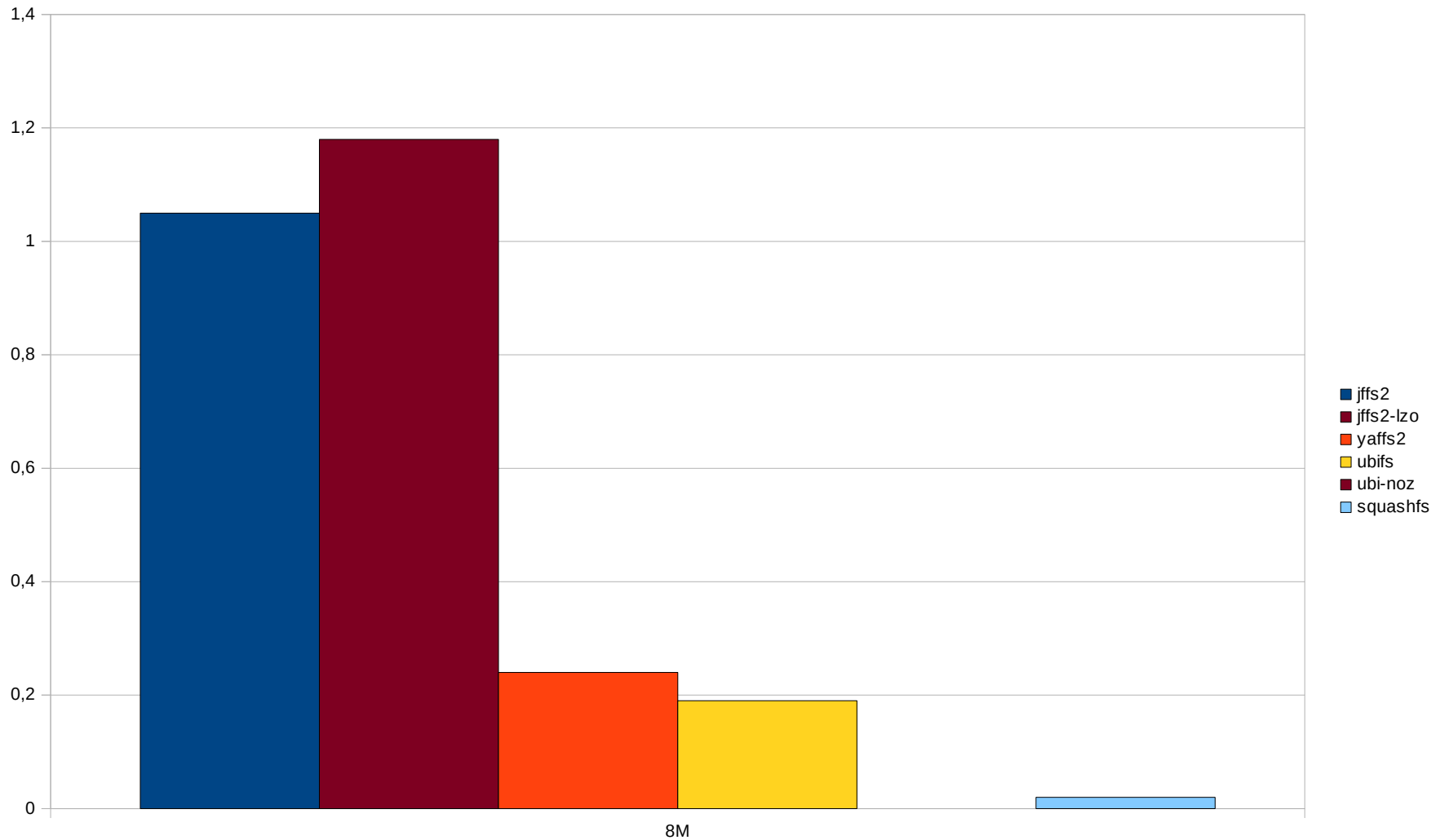
ubifs-noz / 8M: doesn't fit





Zoom - Mount time (seconds) - 8M

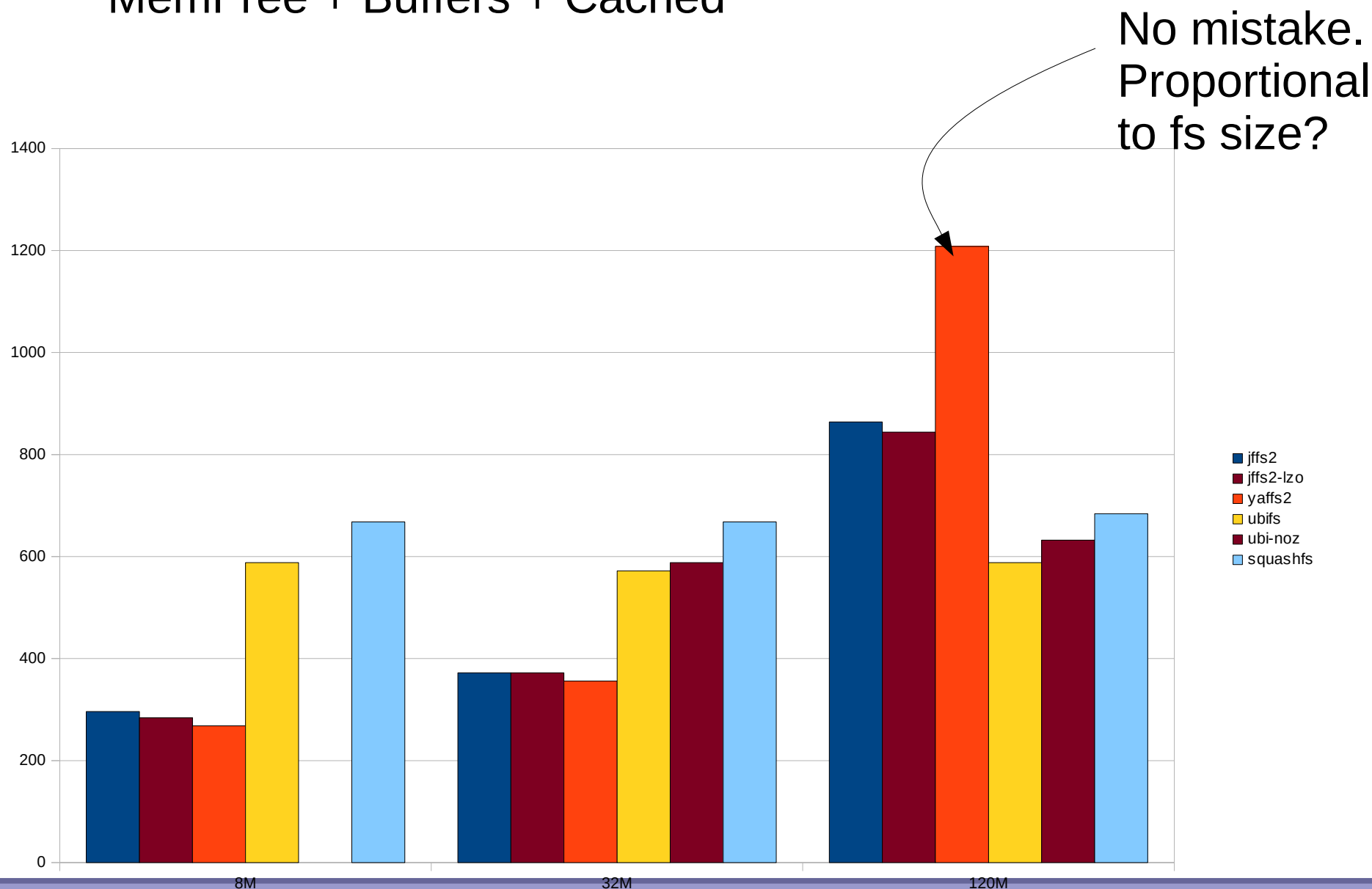
ubifs-noz / 8M: doesn't fit





Memory consumption after mounting (KB)

Free memory measured with /proc/meminfo:
MemFree + Buffers + Cached

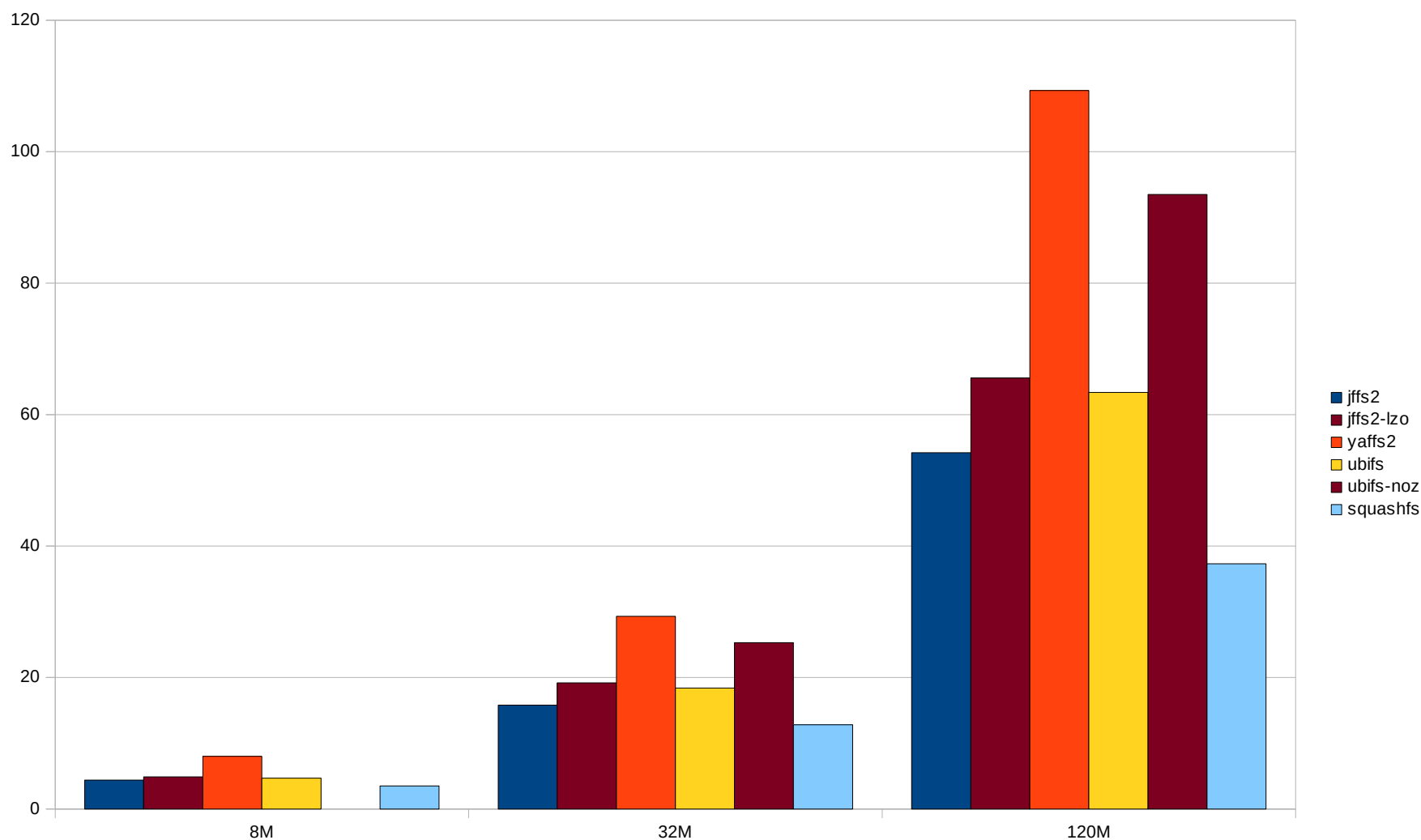


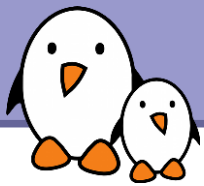


Used space (MB)

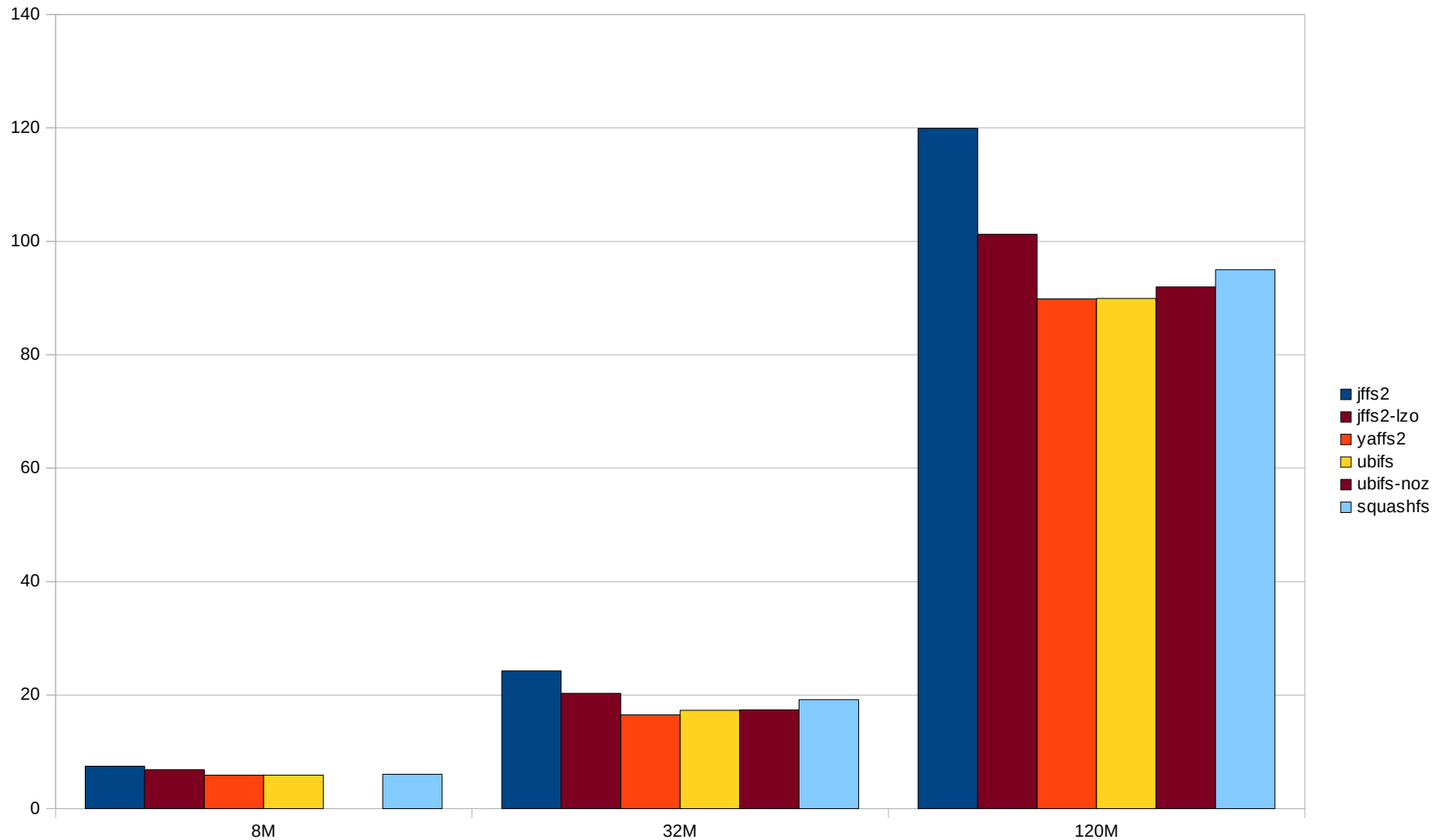
Measured with `df`

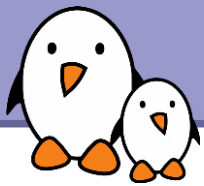
Add some space for UBIFS!
1 MB for 8 MB



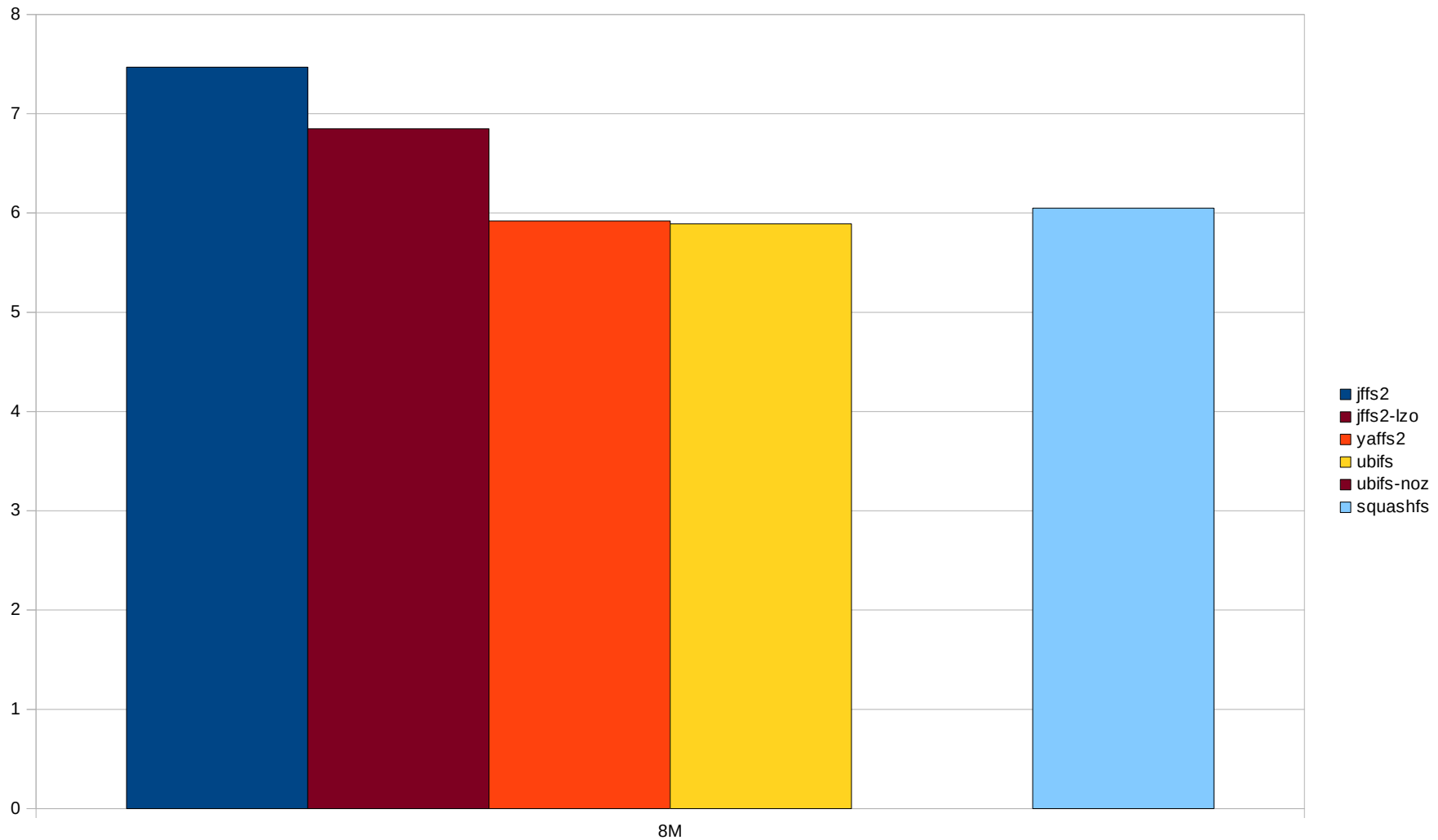


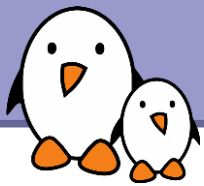
Read time (seconds)





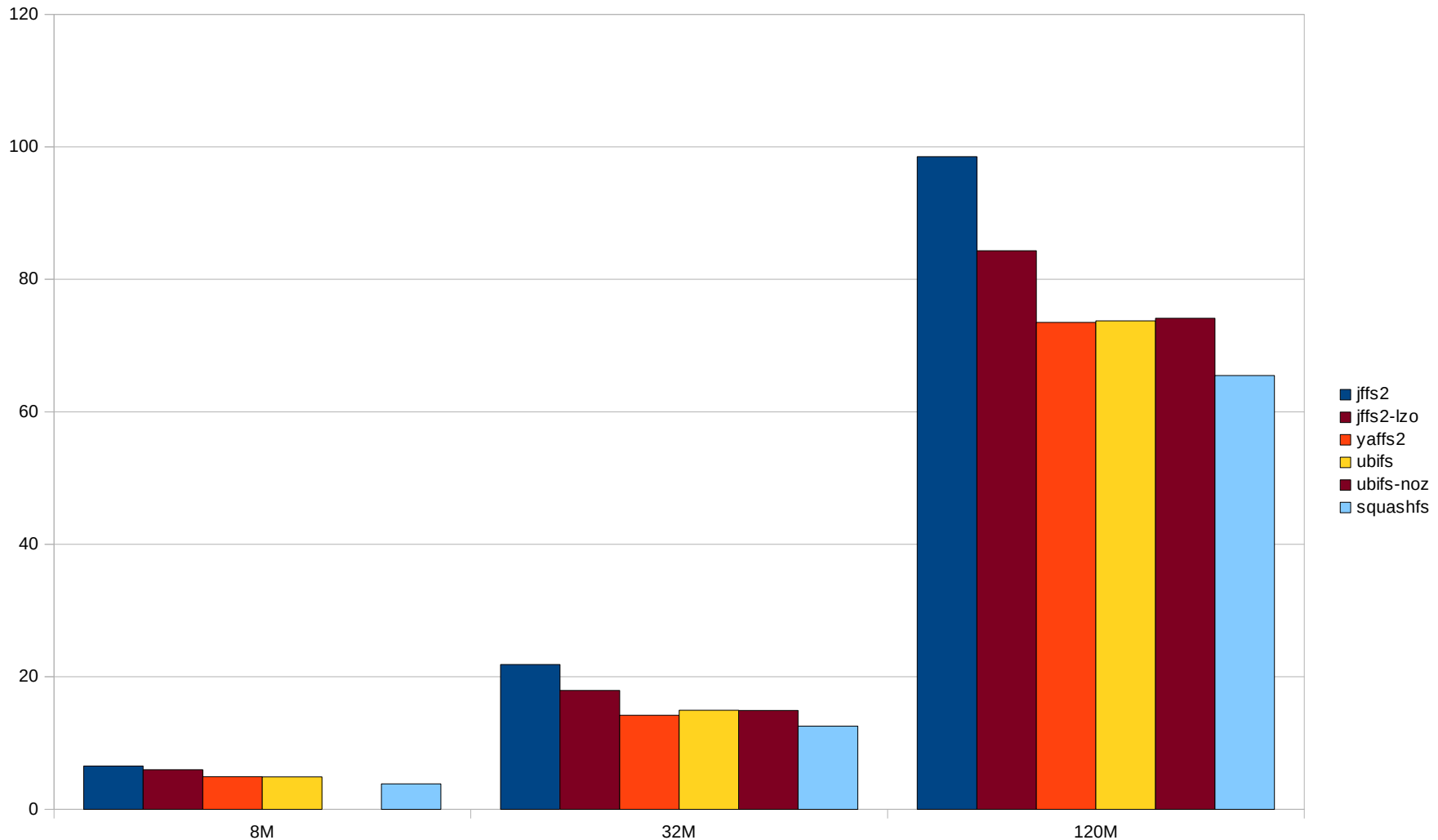
Zoom - Read time (seconds) - 8M

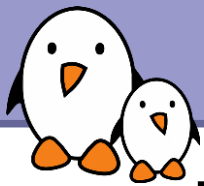




CPU usage during read (seconds)

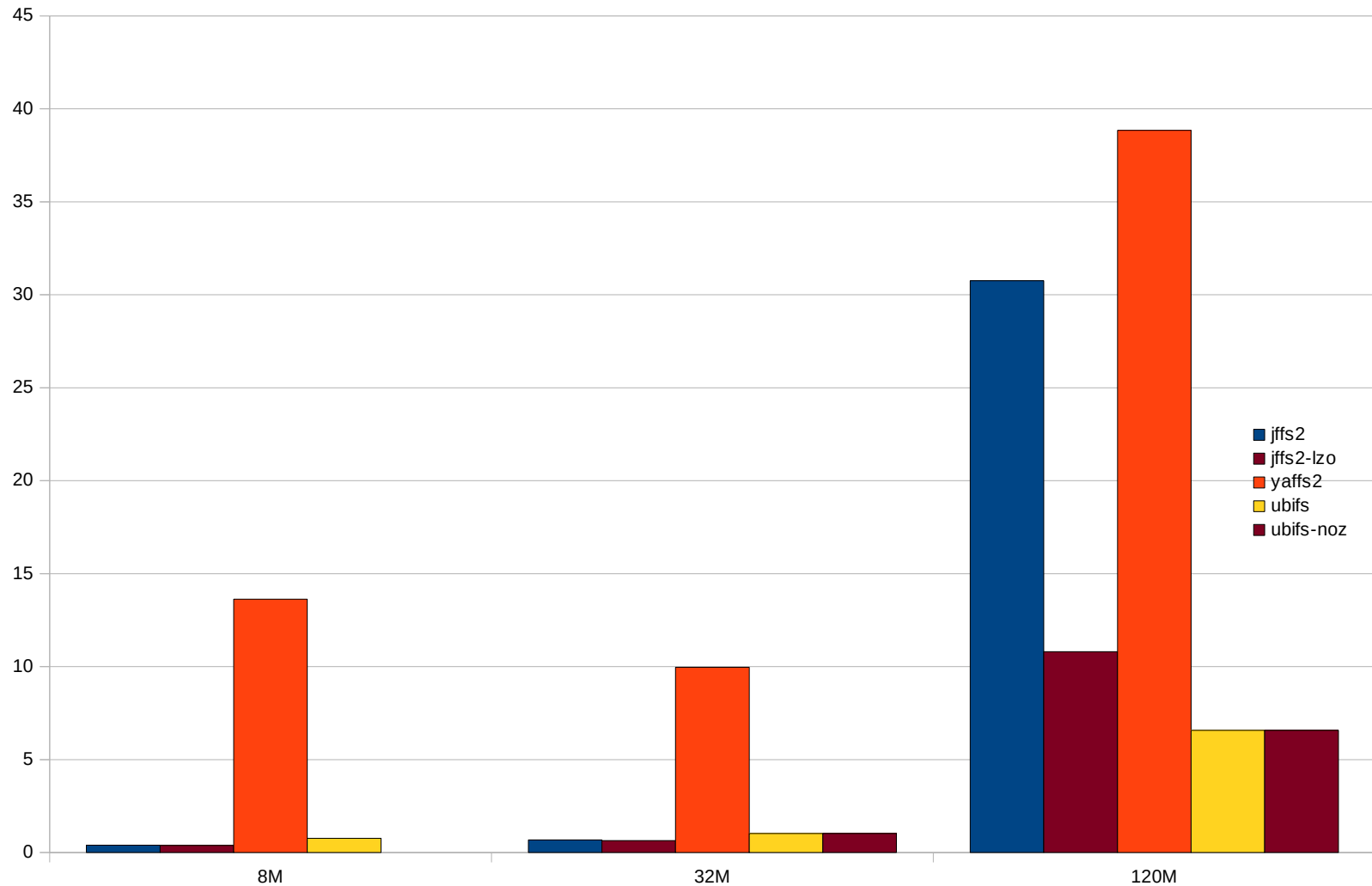
During the experiments in the previous slide
(using the sys measure from the time command)





File removal time (seconds)

Removing all the files in the partition
(after the read experiment)





Write experiment

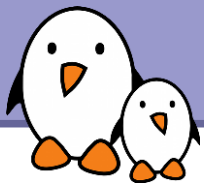
Writing 8M directory contents multiple times
(less in the 8M case)

Data copied from a tmpfs filesystem,
for no overhead reading the files.

Contents: arm Linux root filesystem.

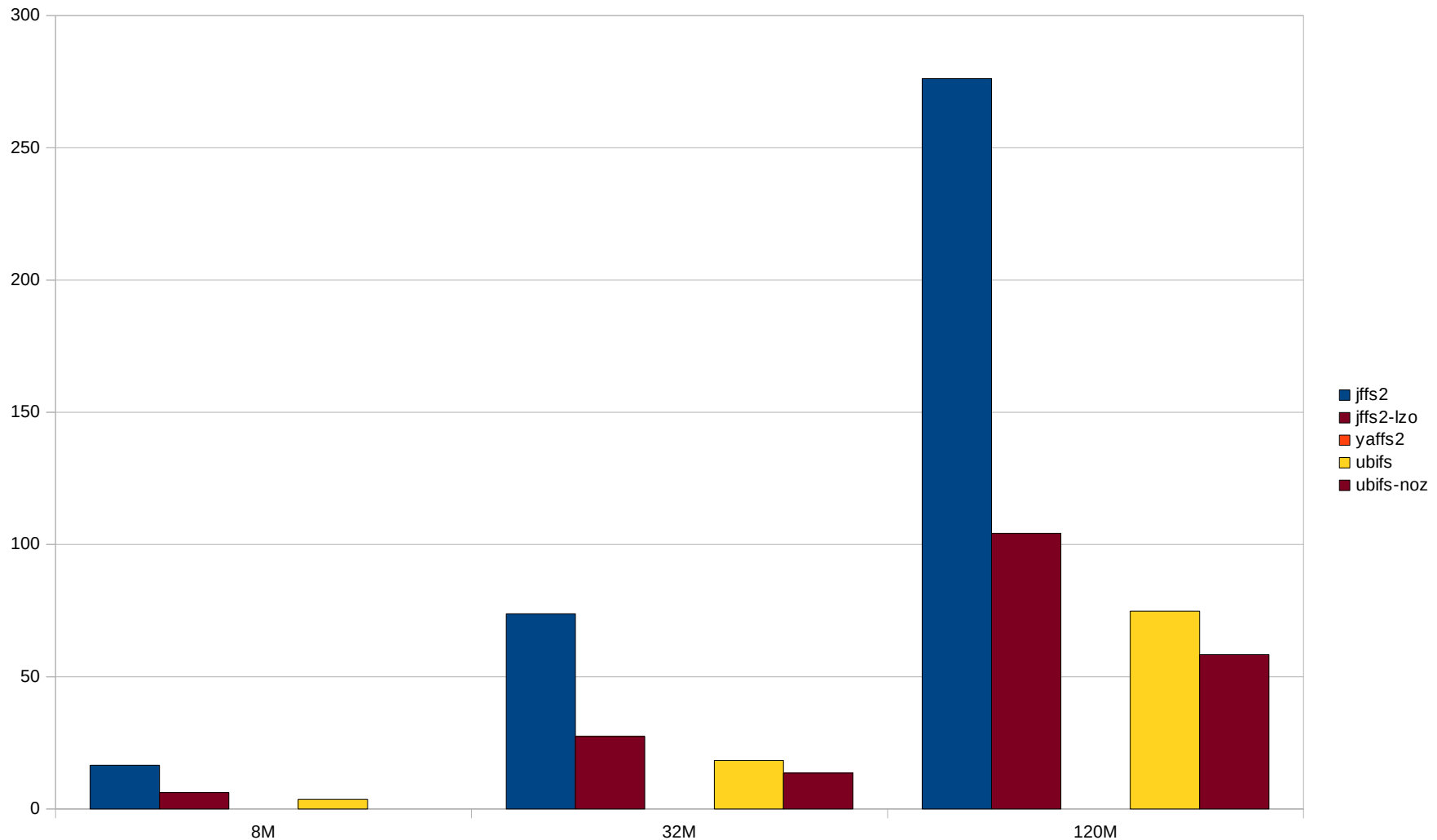
Small to medium size files, mainly executables and shared
libraries.

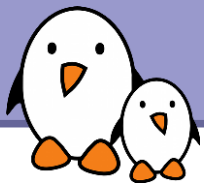
Not many files that can't be compressed.



Write time (seconds)

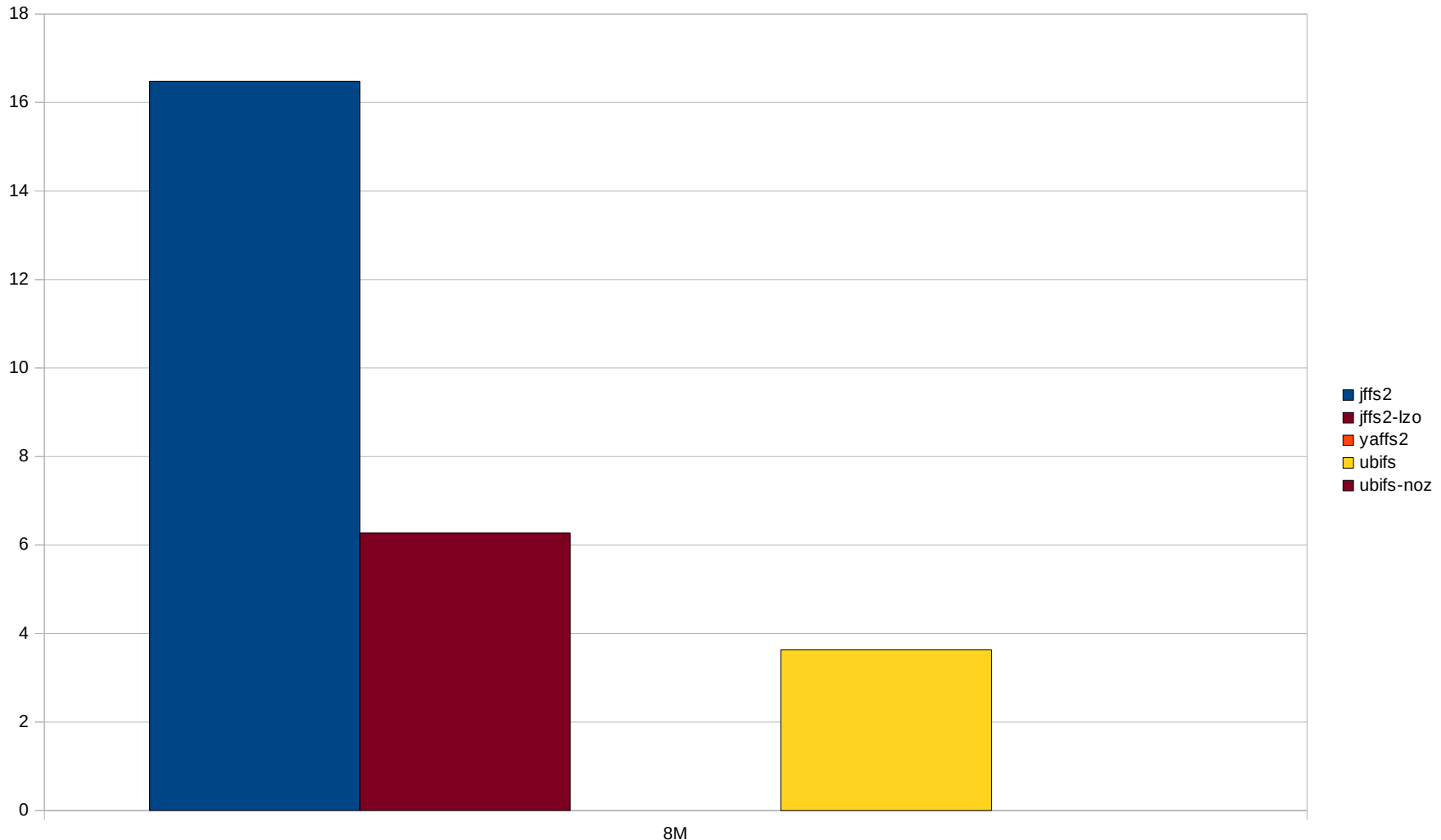
yaffs2 / 8M-32M-120M: doesn't fit
ubifs-noz / 8M: doesn't fit





Zoom - Write time (seconds) - 8M

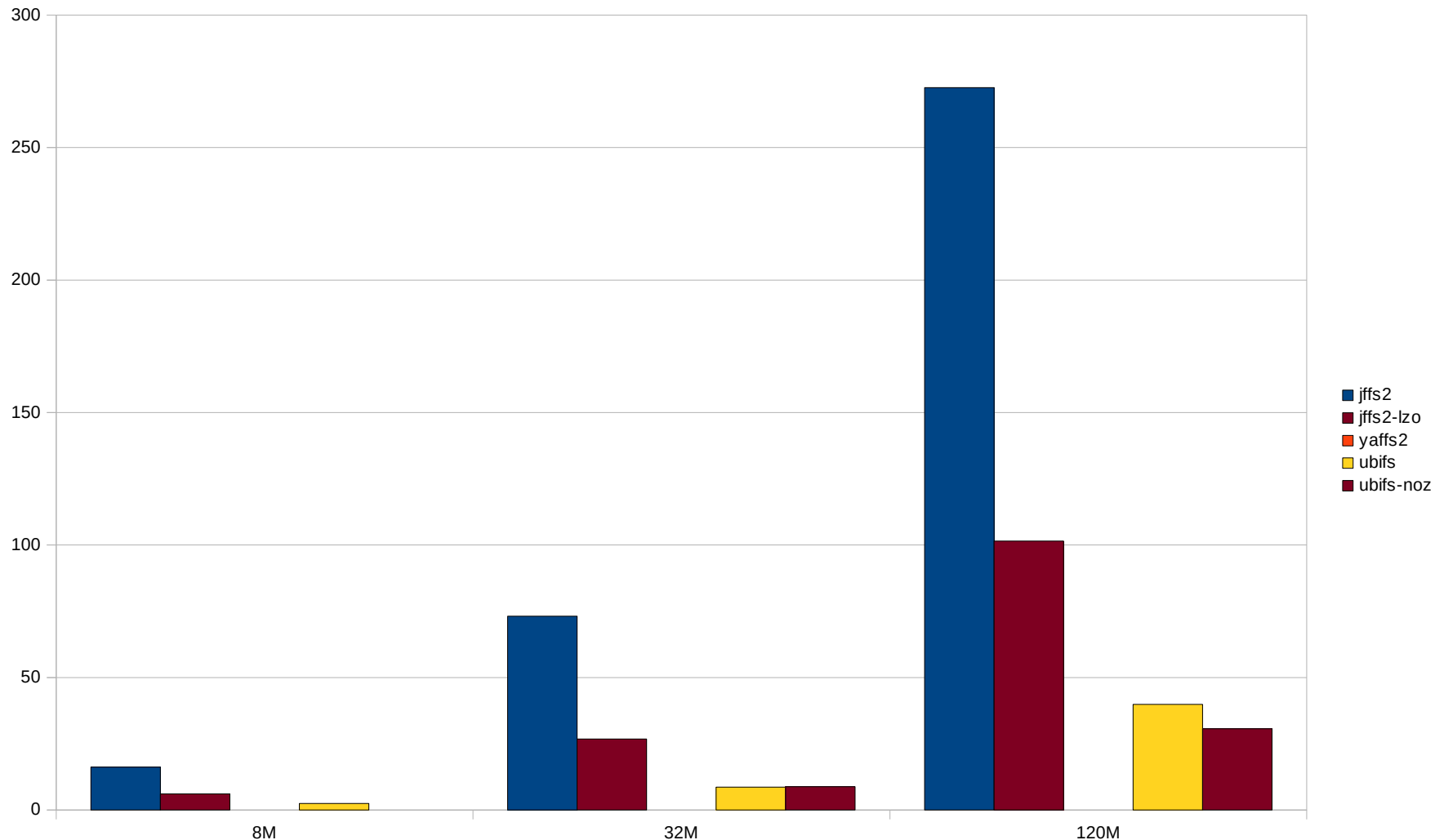
yaffs2 / 8M: doesn't fit
ubifs-noz / 8M: doesn't fit

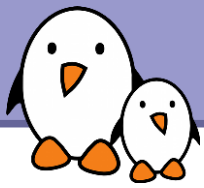




CPU usage during write (seconds)

During the experiments in the previous slide
(using the sys measure from the time command)

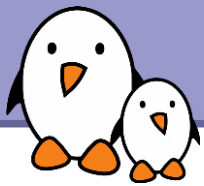




Random write experiment

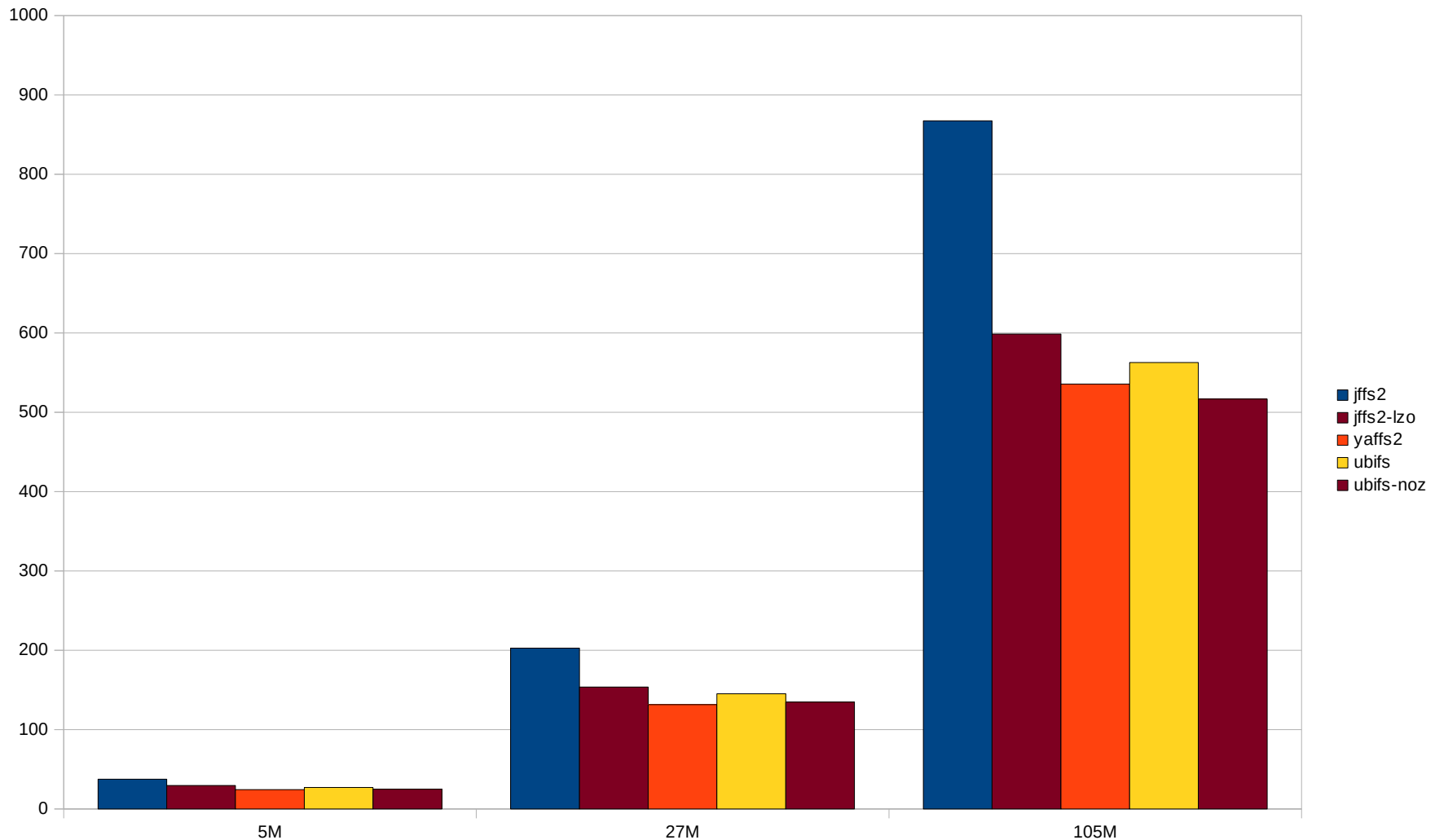
Writing 1 MB chunks of random data
(copied from `/dev/urandom`).

Trying to mimic the behavior of digital cameras and
camcorders, recording already compressed data.



Random write time (seconds)

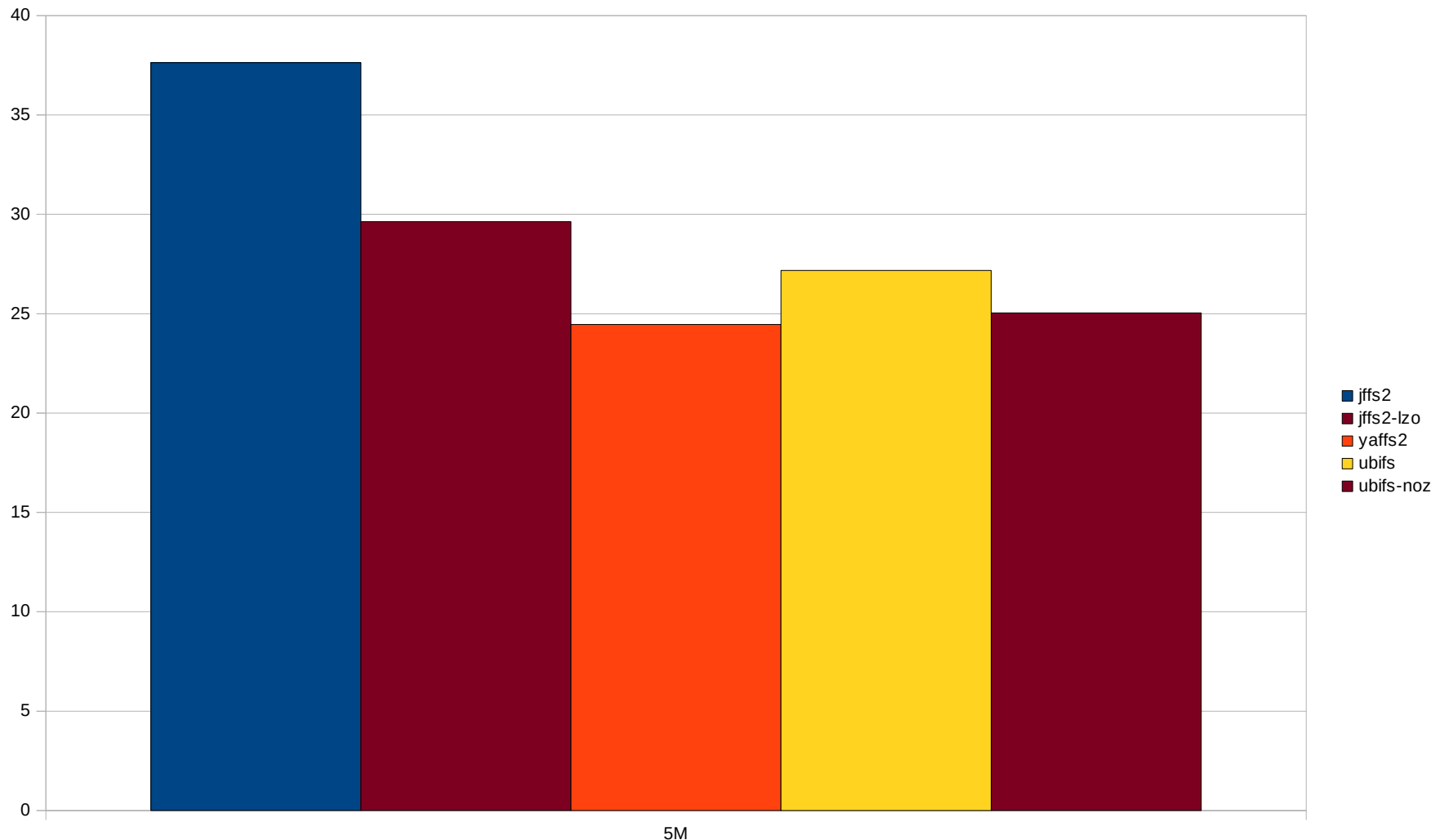
Caution: includes CPU time generating random numbers!

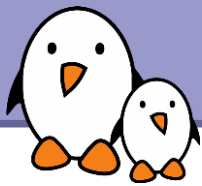




Zoom - Random write time (seconds) - 8M

Caution: includes CPU time generating random numbers!





Other experiments

UBIFS with only lzo support

UBIFS supports both lzo (faster to compress and uncompress) and zlib (slower, but compresses better), and tries to find the best speed / size compromise.

We tried UBIFS with only lzo support, hoping that having only one compressor would reduce runtime.

Results: tiny differences in all benchmarks, even in CPU usage. (roughly between 0.1 and 1%).

Conclusion: don't try to be too smart.
The filesystem is already fine tuned to work great in most cases.



Suitability for very small partitions

8M MTD partition

jffs2 fits 13 MB of files

But probably doesn't leave
enough free blocks

UBI consumes 0.9 MB

ubifs fits 6.6 MB of files

4M MTD partition

jffs2 fits 5.1 MB of files

UBI consumes 0.8 MB

ubifs fits only 1.6 MB of
files!

Bigger sizes: UBI overhead can be neglected:

32 MB: consumes 1.2 MB

128 MB: consumes 3.6 MB



What we observed

jffs2

Dramatically outperformed by
ubifs in most aspects.

Huge mount / boot time.

yaffs2

Also outperformed by ubifs.

May not fit all your data

Ugly file removal time
(poor directory update
performance?)

Memory usage not scaling

ubifs leaves no reason
to stick to yaffs2.

ubifs

Great performance in all
corner cases.

SquashFS

Best or near best
performance
in all read-only scenarios.



Conclusions

Convert your jffs2 partitions to ubifs!

It may only make sense to keep jffs2
for MTD partitions smaller than 10 MB,
in case size is critical.

No reason left to use yaffs2 instead of jffs2?

You may also use SquashFS to squeeze more stuff
on your flash storage. Advisable to use it on top of
UBI, to let all flash sectors participate to wear
leveling.

SquashFS

— — — —

MTD block

— — — —

MTD API

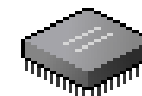
— — — —

UBI

— — — —

MTD driver

— — — —



Flash chip



Experimental filesystems (1)

A look at possible future solutions?

wikifs

A CELF sponsored project.

A Wiki structured filesystem
(today's flash filesystems
are log structured).

Already used in Sony digital
cameras and camcorders.

Pros: direct / easy export of
device functionality
description to elinux.org.

The author is in the room!

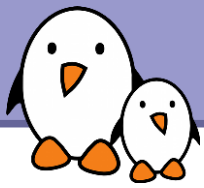
linuxtinyfs

Targets small embedded
systems.

Negative memory
consumption: achieved by
compiling out the kernel file
cache.

Pros: very fast mount time

Cons: a mount-only
filesystem. Way to
implement read and write
not found yet.



Experimental filesystems (2)

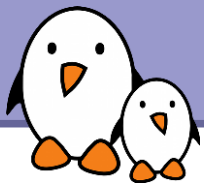
fsckfs

An innovative filesystem rebuilding itself at each reboot.

Pros: no user space tools are needed.

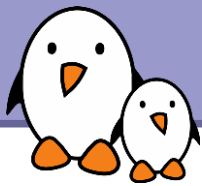
No `fsck`. `fsckfs` utility needed.

Cons: mount time still needs improving.



Update on filesystems for flash storage

Advice for flash-based block storage



Issues with flash-based block storage

Flash storage made available only through a block interface.

Hence, no way to access a low level flash interface
and use the Linux filesystems doing wear leveling.

No details about the layer (Flash Translation Layer) they use.
Details are kept as trade secrets, and may hide poor
implementations.

Hence, it is highly recommended to limit the number of writes
to these devices.



Reducing the number of writes

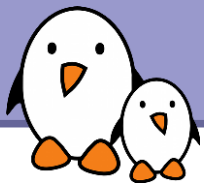
Mount your filesystems as read-only, or use read-only filesystems (SquashFS), whenever possible.

Keep volatile files in RAM (tmpfs)

Use the `noatime` mount option, to avoid updating the filesystem every time you access a file. Or at least, if you need to know whether files were read after their last change, use the `relatime` option.

Don't use the `sync` mount option (commits writes immediately). No optimizations possible.

You may decide to do without journaled filesystems. They cause more writes, but are also much more power down resistant.



Useful reading

Introduction to JFFS2 and LogFS:

<http://lwn.net/Articles/234441/>

Documentation on the linux-mtd website:

<http://www.linux-mtd.infradead.org/>



Other talks

During this ELCE 2008 conference

Thursday

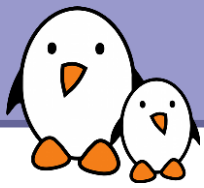
11:50 - Managing NAND longevity in a product
Matthew Porter, Embedded Alley (too late!)

Friday

11:15 - Using the appropriate wear leveling to extend product
lifespan. Bill Roman, Datalight

14:10 - Overview of SquashFS filesystem
Philip Lougher (independent)

15:25 - NAND chip driver optimization and tuning
Vitaly Wool, Embedded Alley



Thank you!

Questions?

New filesystem
suggestions?

