Linux flash file systems
JFFS2 vs UBIFS

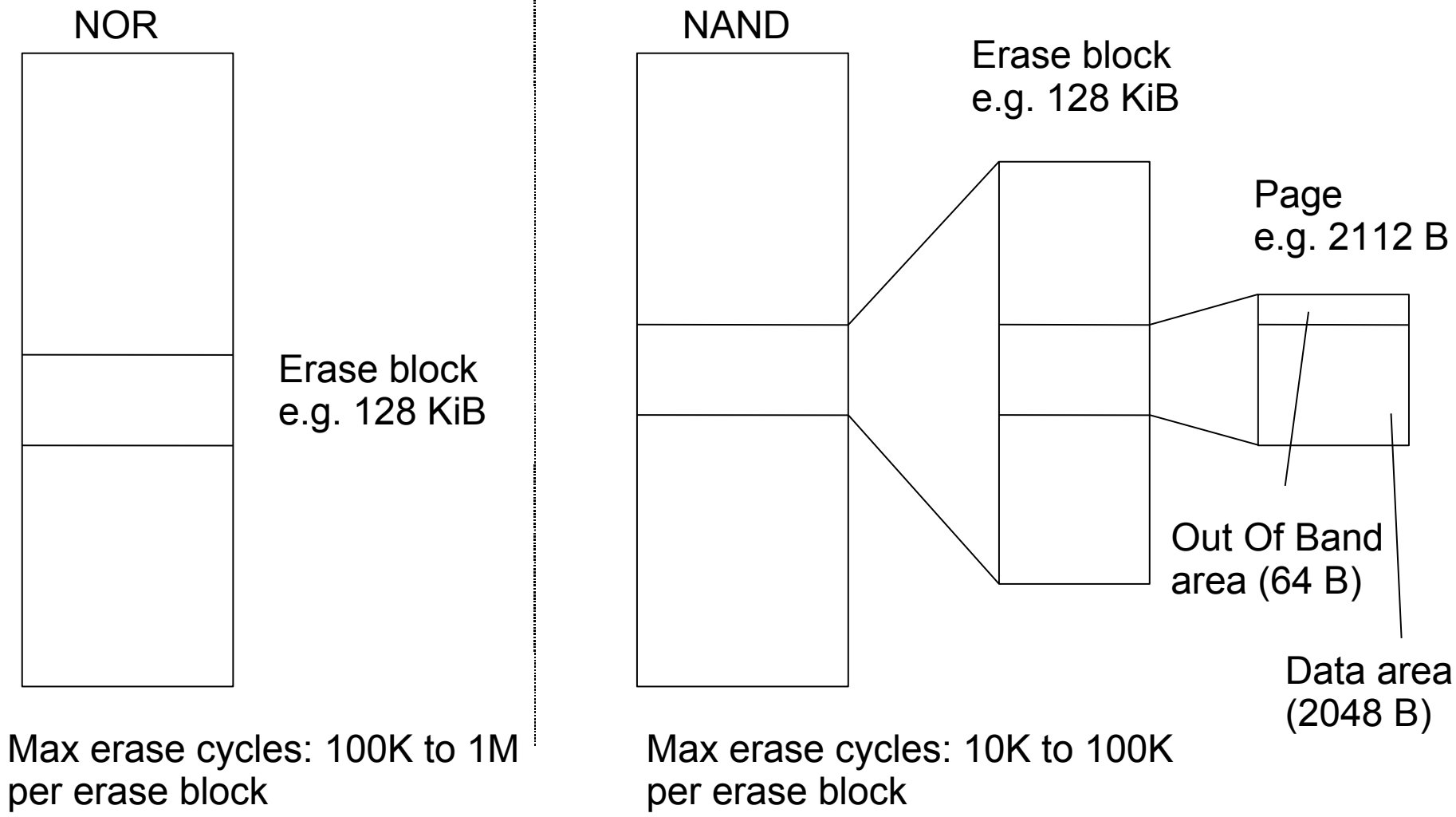
Chris Simmonds
2net Limited

Embedded Systems Conference UK. 2009

Overview

- Many embedded systems use raw flash chips
- JFFS2 has been the main choice for almost 10 years
- As flash sizes increase the scalability problems of JFFS2 become more obvious
- UBIFS is being talked about as the next flash file system
  - How does it compare?

Types of flash memory



NAND flash

- Bit errors
  - Need ECC stored in OOB area to detect & correct
  - ECC may be handled in hardware or software
- Bad blocks
  - Up to 2% erase blocks bad in new chips
  - Blocks may go bad during normal operation
  - Bad block marked with a flag on OOB
- Multi-Level Cell (MLC) NAND
  - High storage density; high bit error rate; few erase cycles (10 K)

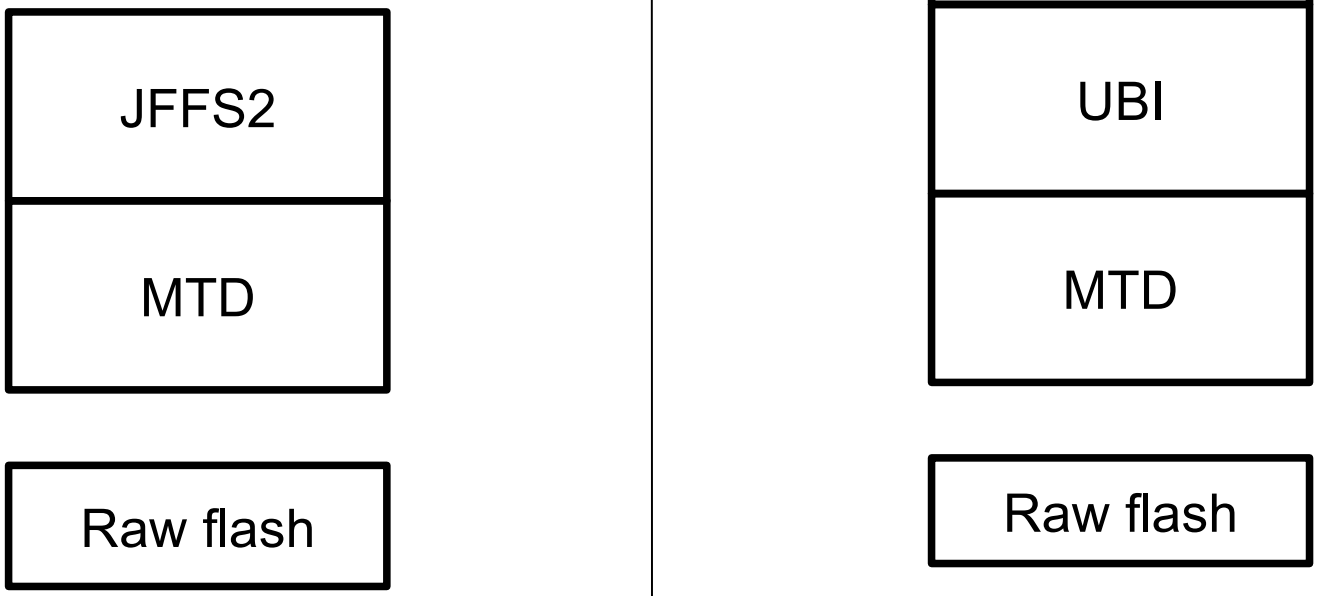
Flash translation layers

- Sub allocation within erase block
- Garbage collection to coalesce & free obsolete data
- Wear leveling
- Bad block handling (NAND)
  - Includes ECC generation & checking
- Avoid data corruption when powered down

Commodity flash devices

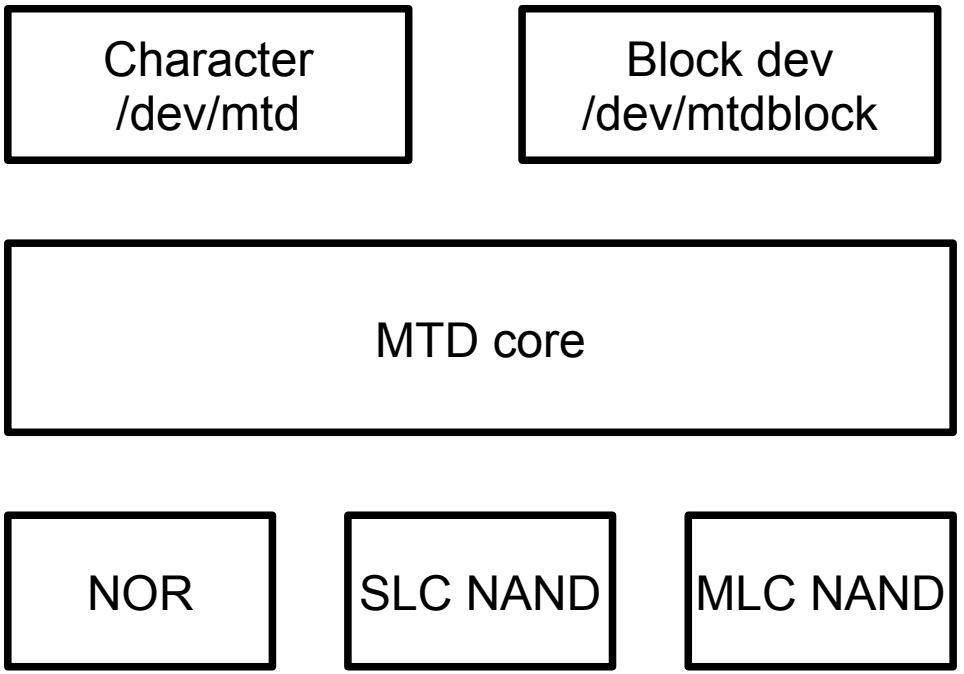
- For example SD, Compact Flash, USB storage
- Flash translation layer implemented in firmware on the device
  - Appears to operating system like a hard drive
- Very limited reliability data from manufacturers
  - Some have known problems with wear leveling and corruption at power off
- Alternative: use raw flash with translation in the file system
  - That is what JFFS2 and UBIFS do!

Flash file systems



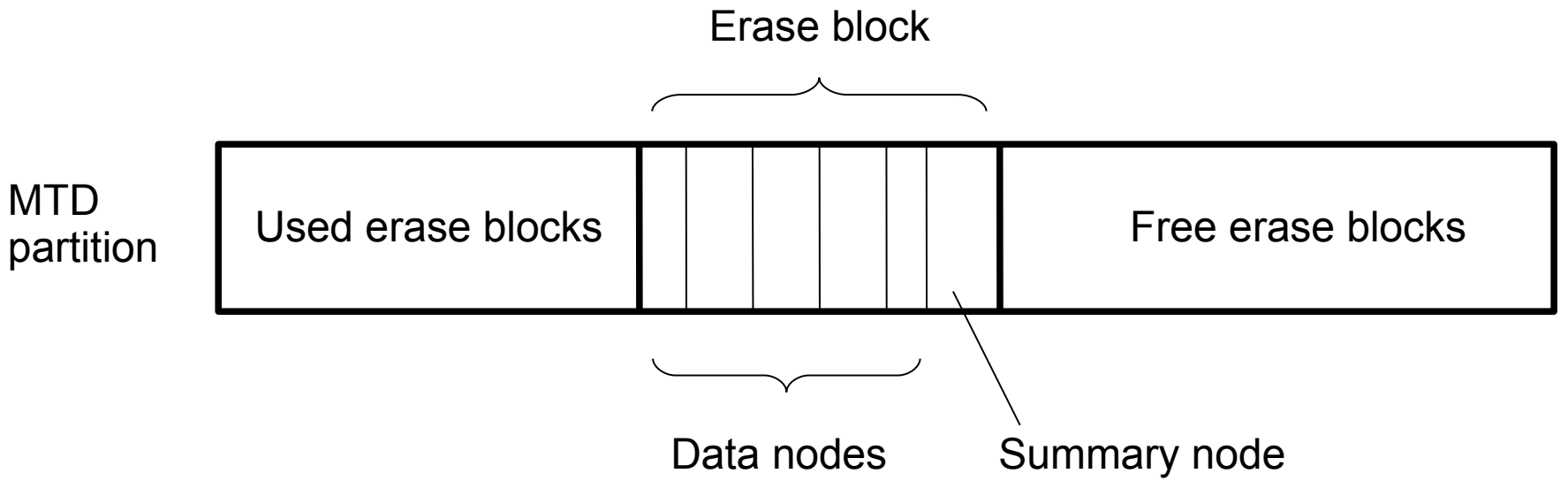
Memory Technology Device layer

- MTD is the lowest level for accessing flash chips
- Presents flash as one or more partitions of erase blocks



JFFS2

- File data and meta data stored as nodes
- No index stored on-chip: have to re-create from summary nodes at mount: mount is slow
- Bad block handling (NAND)
- Optional data compression - zlib default

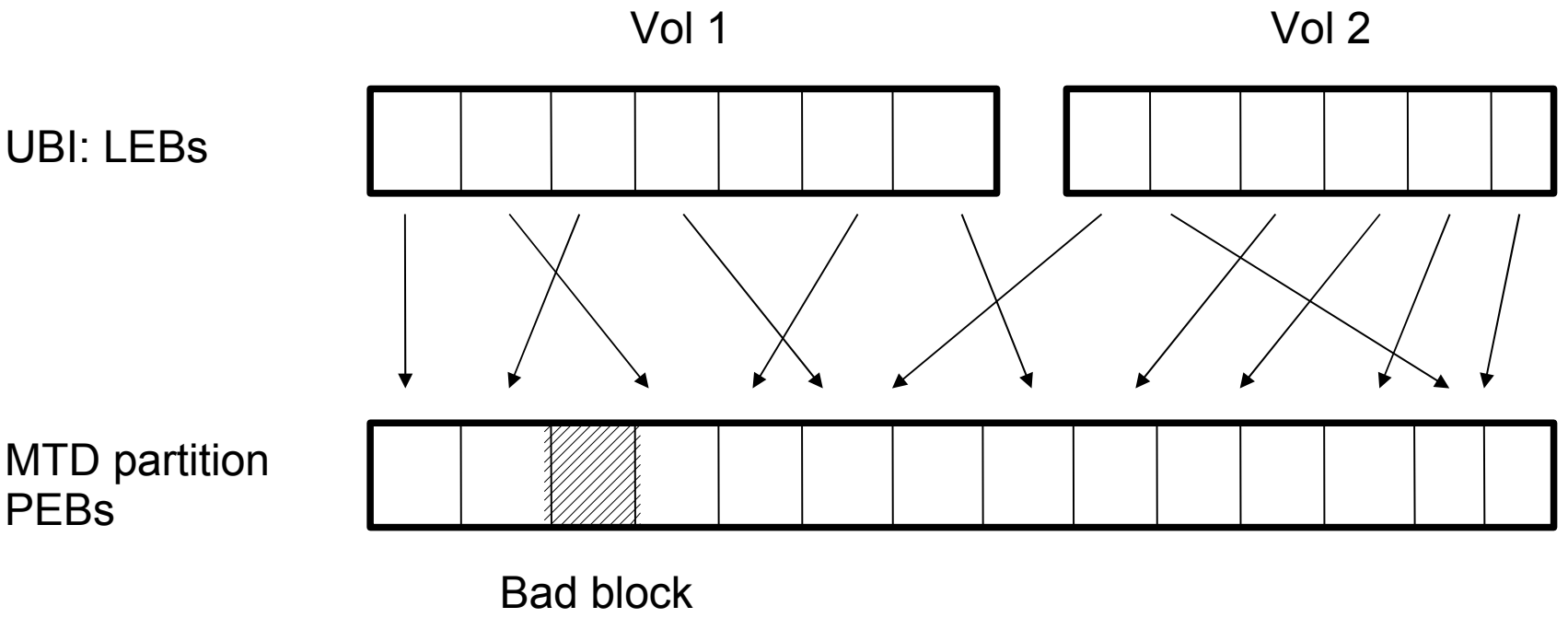


UBI

- UBI = Unsorted Block Image
- Maps *Physical Erase Blocks* in an MTD partition to *Logical Erase Blocks*
- Adds
  - Bad block handling
  - Volumes
  - Wear leveling within a volume
- Introduced in Linux 2.6.22

UBI - erase block mapping

PEB = Physical Erase Block
LEB = Logical Erase Block



UBIFS

- Journal
  - Robust on power fail
- Write-back cache
  - Faster writes (see next slide)
- On-chip index
  - Fast mount
- Compression: lzo or zlib
  - More data on your chip!
- Introduced in Linux 2.6.27

Consequences of write-back cache

- Write-through cache (e.g. JFFS2)
  - All writes are synchronous
- Write-back cache
  - Writes are completed later by pdflush daemon
- To avoid loss of data need to do one of
  - Call `fsync()` after critical writes
  - Open files with `O_SYNC` flag
  - Mount ubifs with `-o sync`

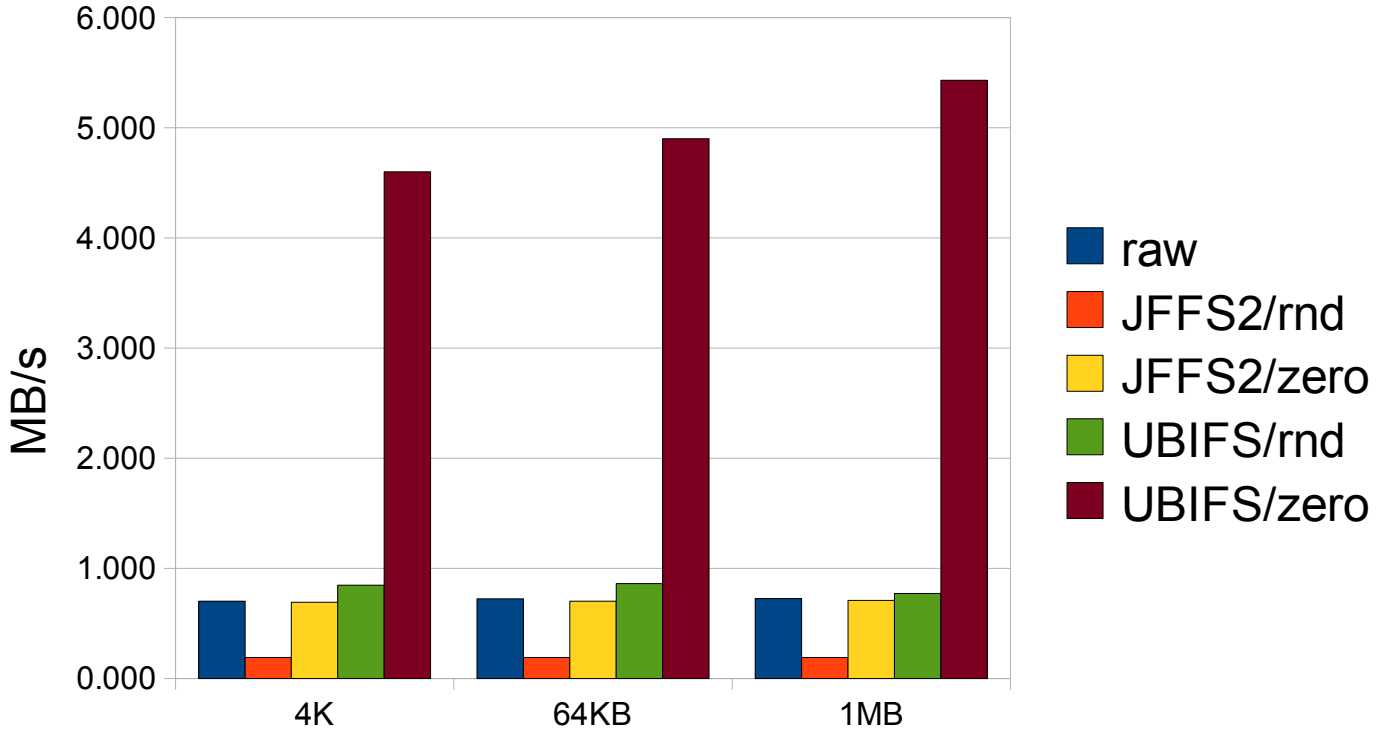
Device used for testing

- ARM 926 SoC @ 155 Mhz
- 64 MiB RAM
- 1 x 1Gib (128 MiB) ST/Numonyx NAND flash
  - 128 KiB erase block
  - 2 KiB page
  - Software ECC
  - Programmed i/o
- 2.6.27 kernel

Write test

- Write 10 MiB random data in block sizes 4KiB, 64KiB and 1MiB to
  - Raw device: /dev/mtdblock5
  - JFFS2 file
  - UBIFS file
- Write 10 MiB zeros to
  - JFFS2 file
  - UBIFS file

Write speed


6.000
5.000
4.000
3.000
2.000
1.000
0.000
MB/s
4K
64KB
1MB
raw
JFFS2/rnd
JFFS2/zero
UBIFS/rnd
UBIFS/zero

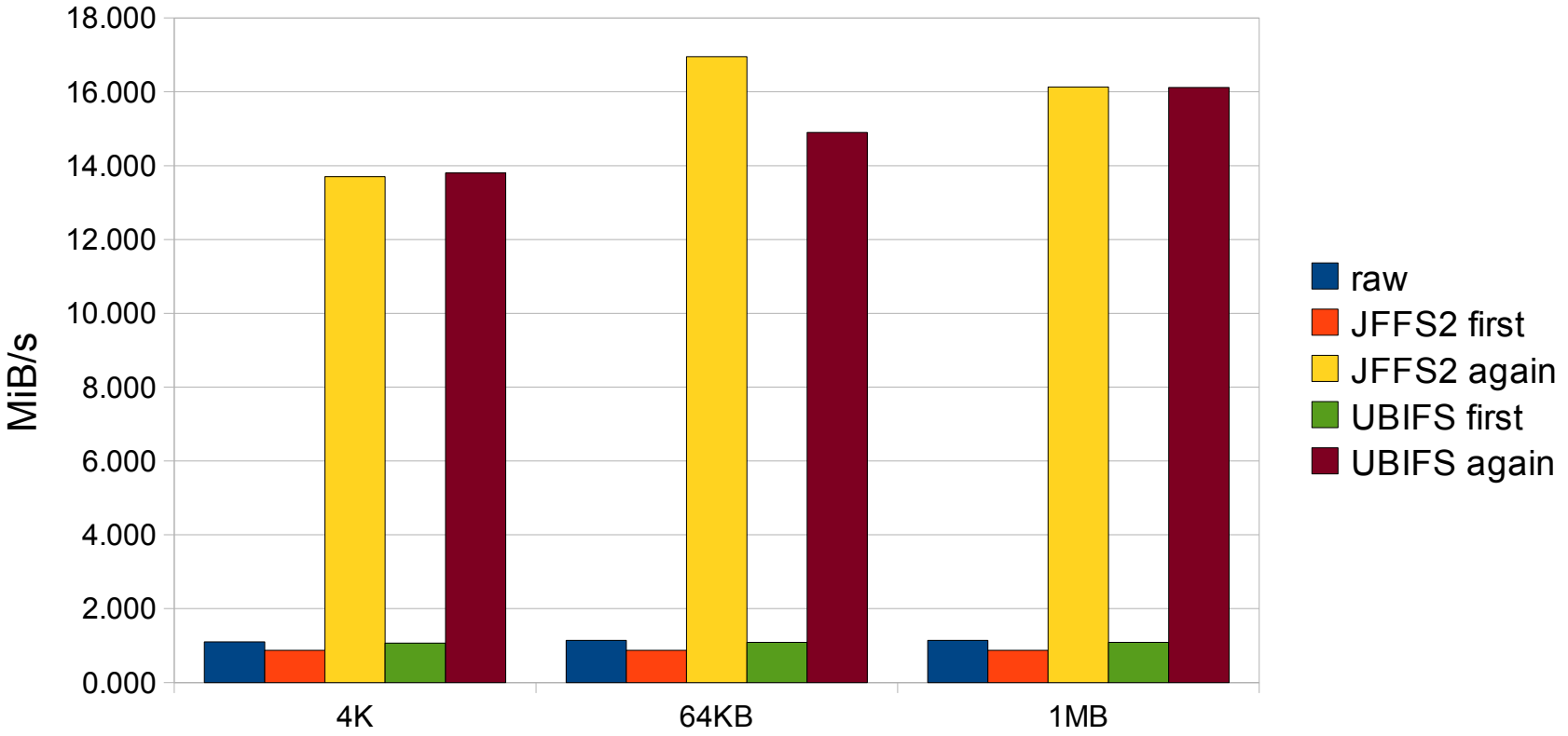
Write speed conclusions

- Raw speed is 0.7 MiB/s
- JFFS2
  - Random data: 0.2 MiB/s
    - Compression slows it down
  - Zeros: 0.7 MiB/s
    - Compression fast, approaches raw speed
- UBIFS
  - Random data: 0.8 MiB/s
  - Zeros: 5 MiB/s
    - Write-back cache speeds up in both cases

Read speed test

- Read 10 MiB random data in block sizes 4KiB, 64KiB and 1MiB from
  - Raw device: /dev/mtdblock5
  - JFFS2 file
  - UBIFS file
- Measure JFFS2 and UBIFS times
  - Immediately after mount (no data cached)
  - Again, with cache fully primed

Read speed results



Read speed conclusions

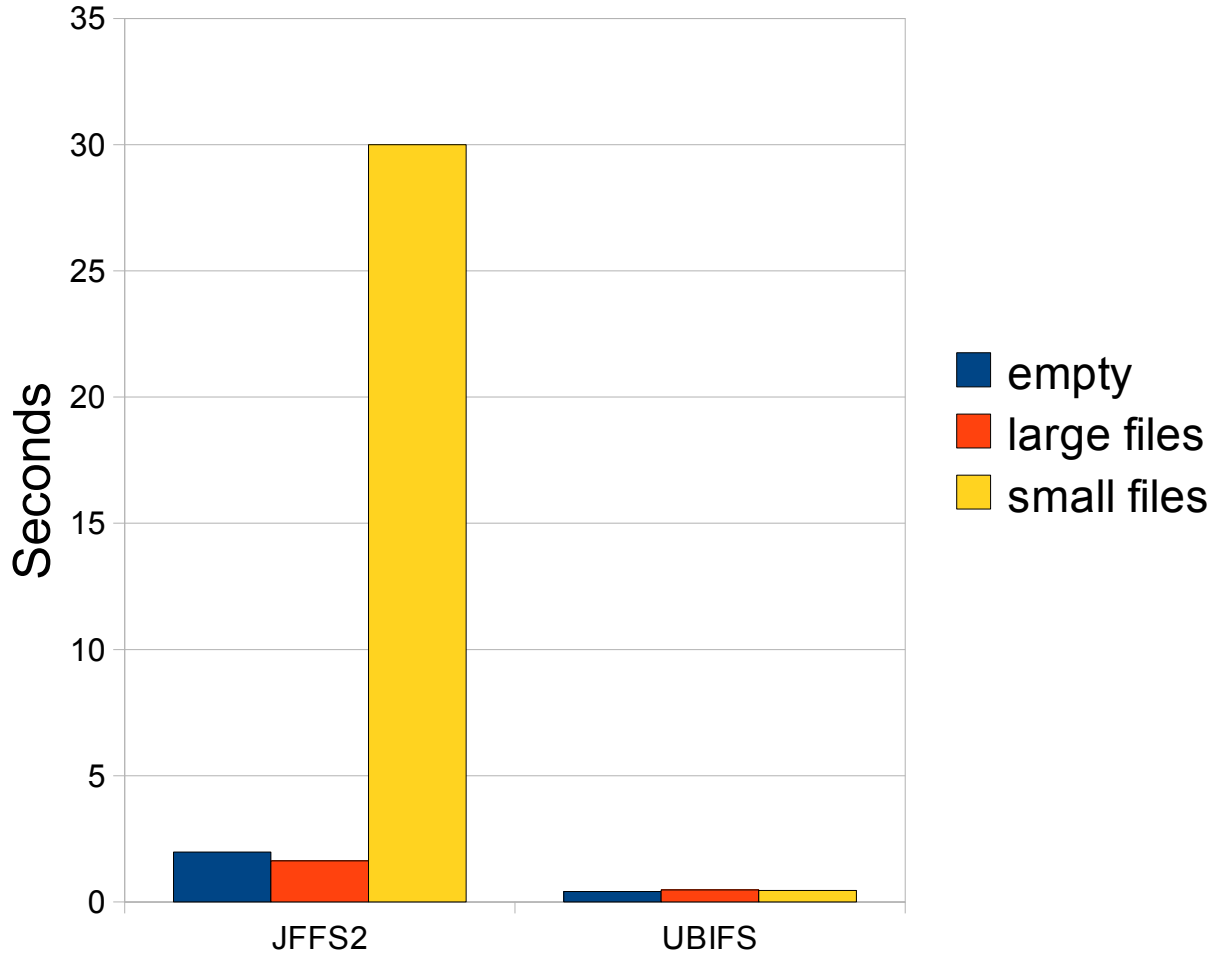
- Raw speed: 1.1 MiB/s
- Immediately after mount
  - JFFS2: 0.87 MiB/s
  - UBIFS: 1.0 MiB/s
- Subsequently
  - Both ~15 MiB/s
- Not much difference between JFFS2 and UBIFS

Mount time

- Mount a file system containing
  - No files
  - 10 files of 8MiB (partition 80% full)
  - 10,000 files of 8KiB (partition 80% full)

Mount speed

Mount time



Mount time conclusions

- UBIFS mount time is constant at 0.5s
- JFFS2 mount time increases dramatically
  - Empty: 1.98s
  - 10K small files: 30s
- The JFFS2 garbage collector thread runs for up to 90s after mount
  - Some file operations (e.g. ls *) will be blocked until it completes

Space efficiency

Empty partition with initial size 109312 blocks of 1 KiB

JFFS2	% overhead	UBIFS	% overhead
106624	2.46%	98004	11.54%

Space taken by a file containing 1 MiB random data when written many small pieces and one large piece

	JFFS2		UBIFS	
Write size	Blocks used	% overhead	Blocks used	% overhead
16 bytes	1468	43.36%	1364	33.20%
1MiB	1048	2.34%	1365	33.30%

Summary

- UBIFS is faster than JFFS2 in all cases
  - Overwhelmingly so during mount
- JFFS2 makes more efficient use of space
- Conclusion:
  - Use JFFS2 on small partitions (< 16 MiB)
  - Use UBIFS in other cases

References

- The Linux MTD, JFFS2 and UBI project
  - http://www.linux-mtd.infradead.org/index.html