A Comparative Analysis Between Embedded Linux Flash File Systems

Ahmed Lutful Sharif

lutfulsharif@yahoo.com

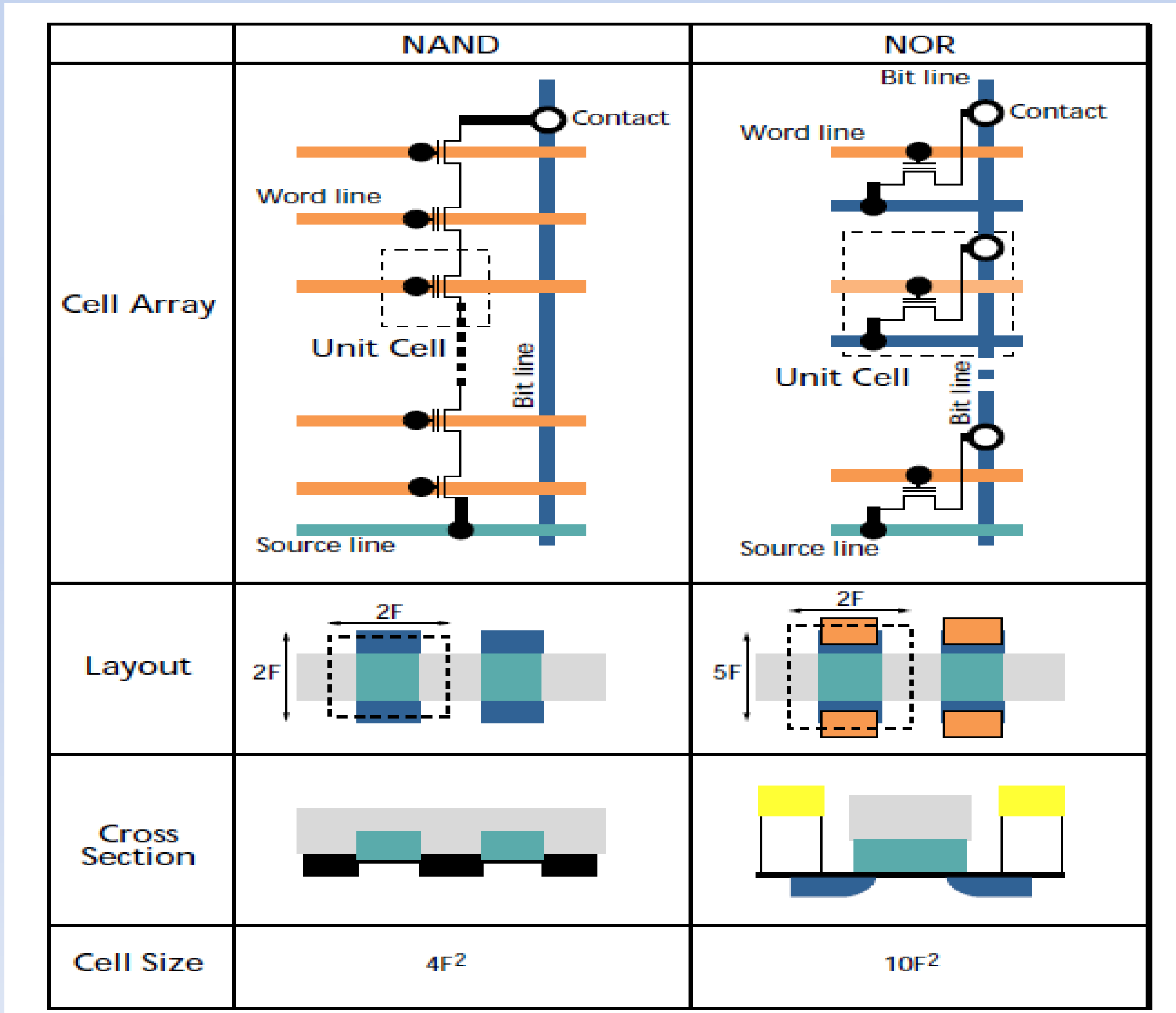
About me

- Oakland Alumni
- Finished MS Embedded Systems in 2008.
- Only working in Embedded Systems Field last 10 years.

Objective

- Clarify design decisions behind choosing embedded linux file systems.
- There may not be a clear winner of a certain file-system over all others, with the performance parameters, it can be deduced what would be a suitable for a particular system.

Flash Type: NAND vs NOR



Flash Type: NAND vs NOR

- NAND is faster during erase/write than NOR.
- NAND less reliable and need ECC support for bit correction.
- NAND erase cycle (100k-1M) and NOR erase cycle (10k-100k). MLC NAND is much lower – max 10k.
- NOR can be memory mapped, NAND is I/O only.

Flash Type: NAND vs NOR

- NAND is more compact.
- NOR is for code storage, NAND can be used for both code and data. For code NAND usage, need to have ECC correction.

Flash Type: NAND vs NOR

NAND Flash issues:

- Bit-Flipping: Inconsistent read (a bit value is read randomly reversed), more happening in NOR – EDC/ECC can correct to some extent.
- BAD Block Management: NOR doesn’t need it. NAND comes with BAD blocks and in course of time develops more. Need BAD block handling.

Flash Type: NAND vs NOR

- Life Span/Endurance:

	Min Erase Cycles Allowed (per erase block)	Max Erase Cycles Allowed (per erase block)
NAND	100,000	1,000,000
NOR	10,000	100,000

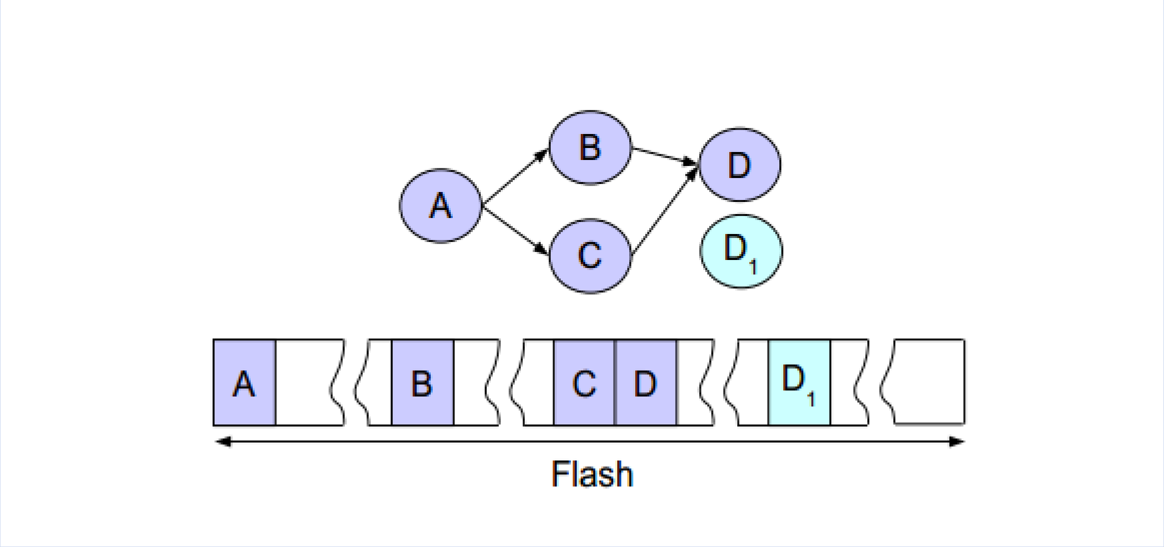
Flash Type: Other

- Some new options i.e. eMMC

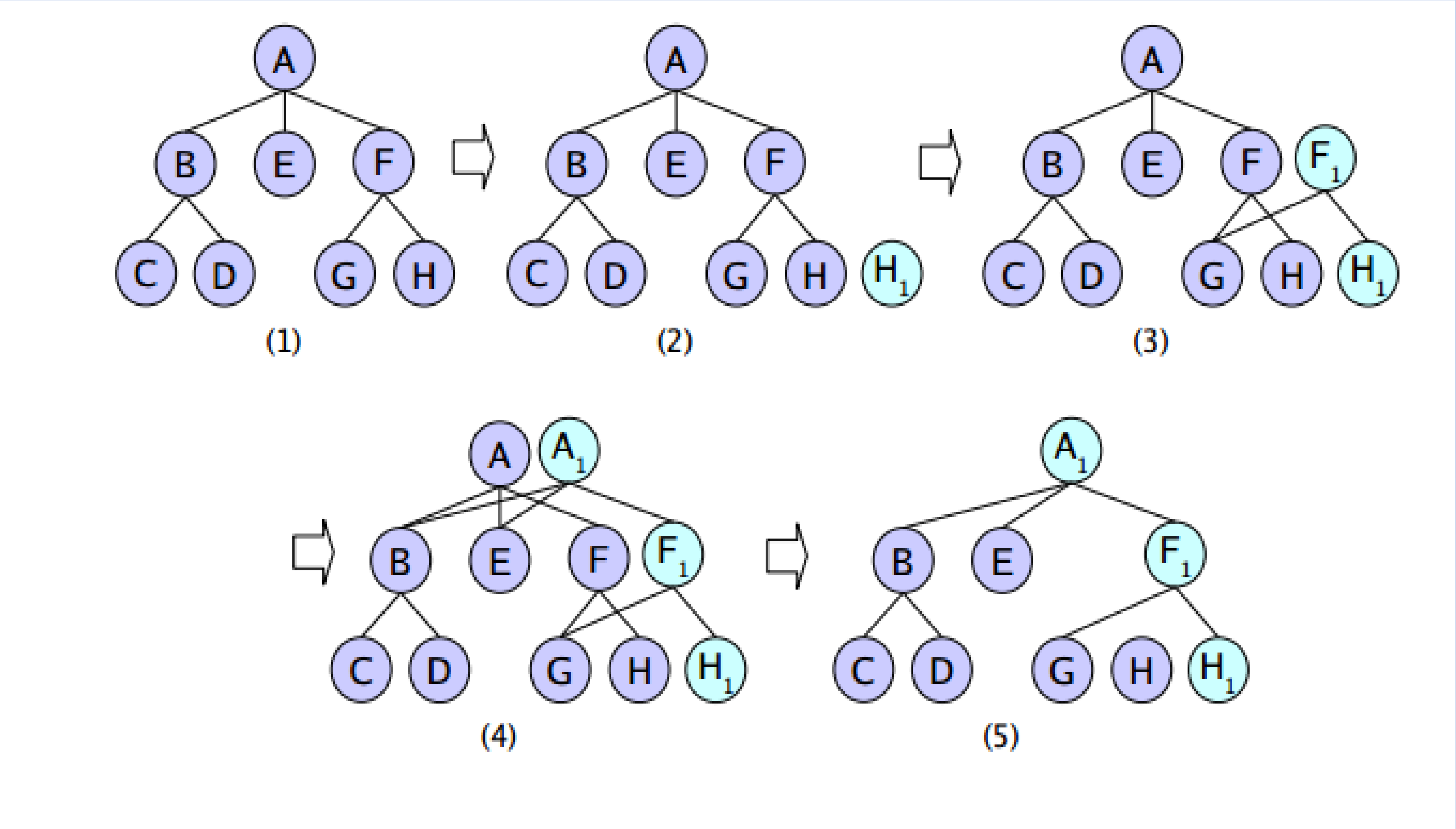
Flash Device

- Can’t do in-place update like a HDD device or RAM.
- Have to copy erase-block, update contents (to be written), erase the block and write again the whole erase block – impractical and will result slowness.
- More practical to adopt a log structure – whenever it’s time to update, find a fresh erase-block (already erased) and continue writing there.
- Need some special handling/mechanism in case the log structure is corrupt in any time of the update (i.e. power cut, user reboot etc.)

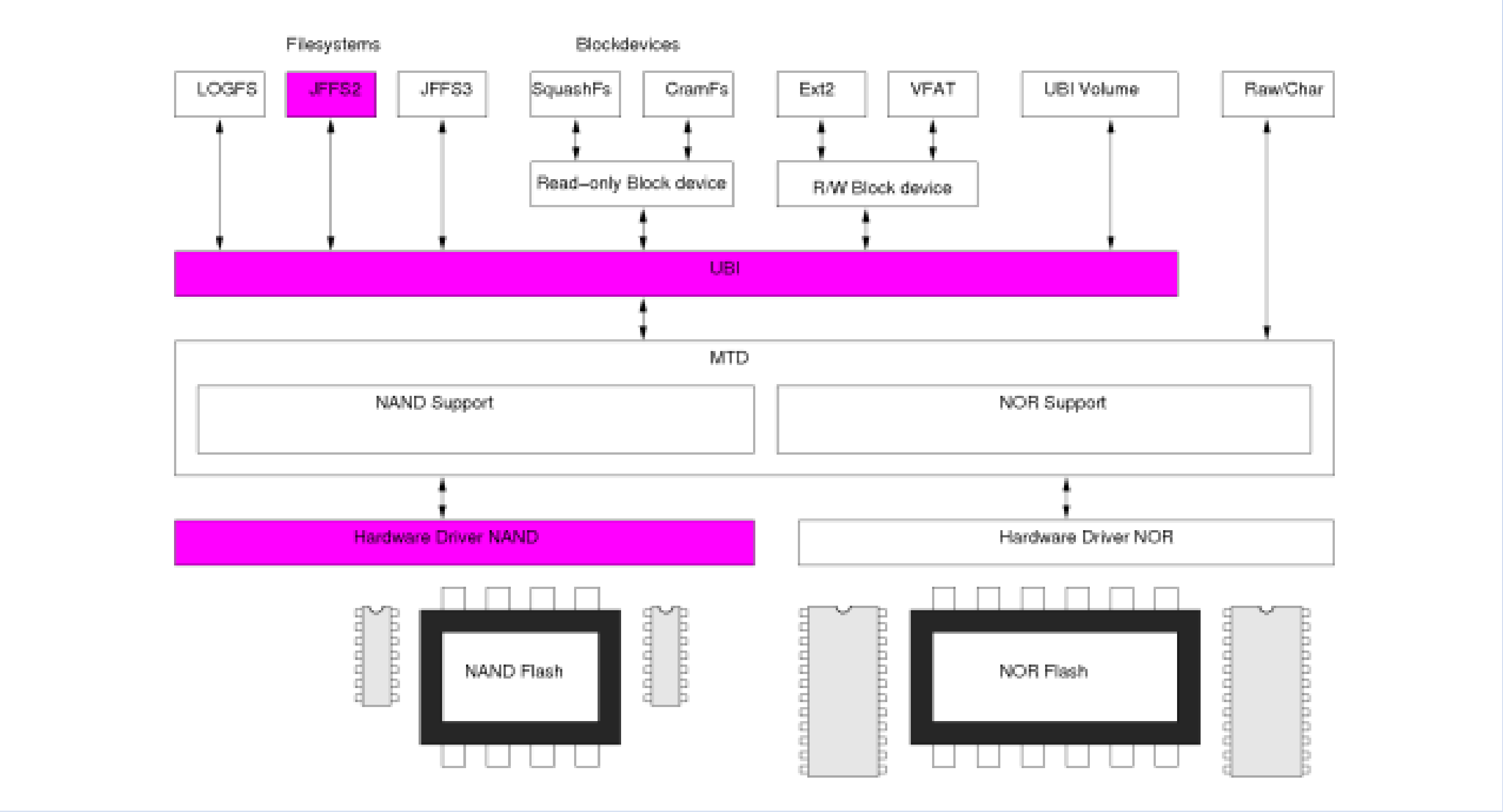
Flash Device Write



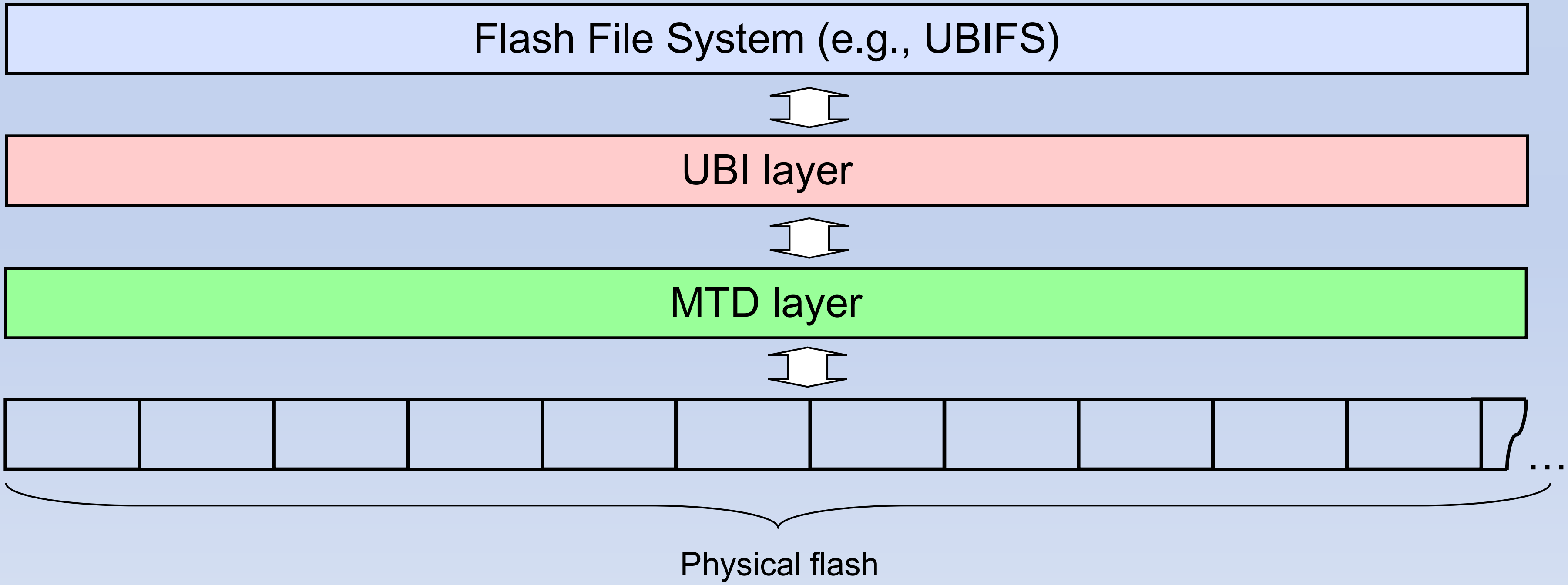
Flash Device Write (Wandering Tree)



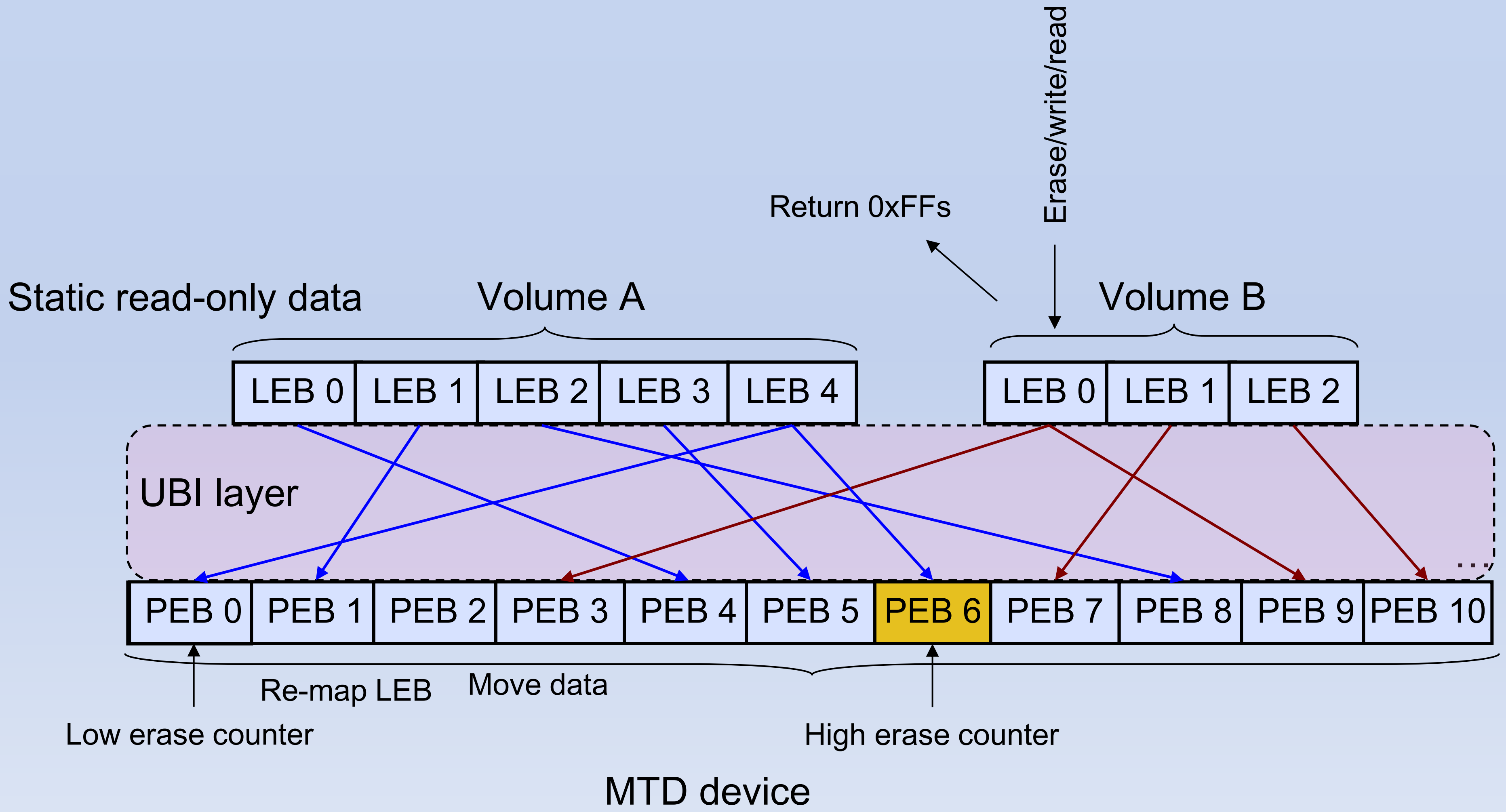
Linux Filesystems



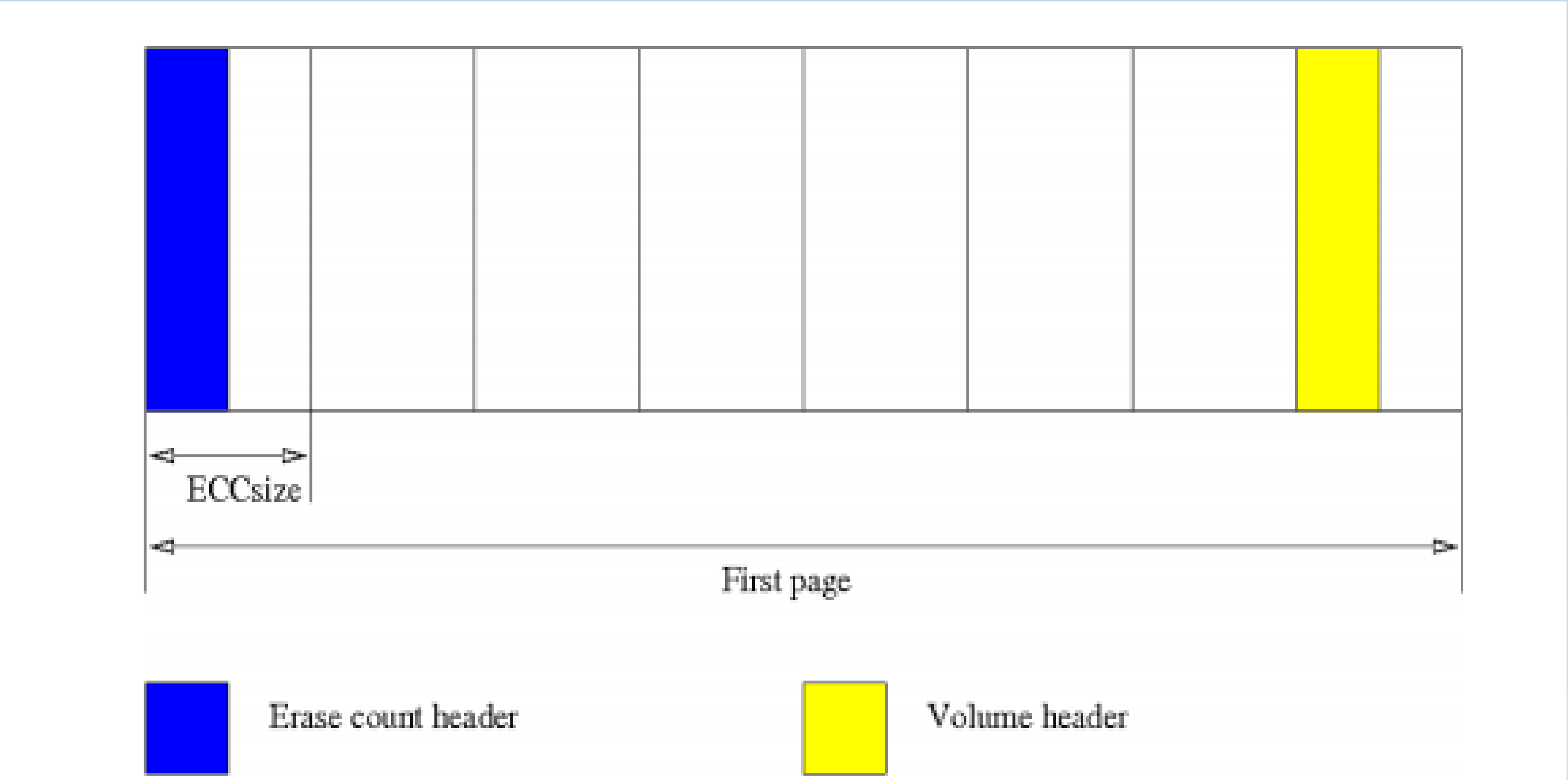
Linux Filesystems:Unsorted Block Image (UBI)



Linux Filesystems:UBI



Linux Filesystems:UBI



When a erase block is to be erased, the current erase count is kept in RAM and after the erase has completed, the incremented erase count is written back to FLASH. When the operation is interrupted, the erase counter is lost. Later after discovering this the affected block is set to the average erase count of all blocks.

Journaling Flash File System Version 2 (JFFS2)

-Economical Flash usage

-On-flight flash compression

-Unclean reboot robustness

-Good enough wear-leveling

Has scalability issues

-Needs to scan whole flash/partition to mount

-JFFS2 index is maintained in RAM – larger flash, larger RAM usage.

Unsorted Block Image FS (UBIFS)

- UBIFS must work on top of UBI volumes MTD->UBI->UBIFS
- Scalability – Scales well w.r.t. flash size and mount time, memory consumption doesn’t depend on flash size.
- UBIFS doesn’t need to need scan the whole media for mounting, it takes msecs to mount UBIFS.
- UBIFS has write-back support.

UBIFS

- UBIFS has tolerance against unclean reboots.
- UBIFS can do on-flight compression during writing.
- UBIFS can recover itself if the indexing information got corrupted.
- UBIFS checksums everything it writes to flash to guarantee data integrity.

Compressed ROM FS (CRAMFS)

- Read-only filesystem.
- Free GPL Linux FS.
- Simple and Space-Efficient.
- Suitable for small/embedded systems.
- zlib-compressed one page at a time to allow random read access. (metadata not compressed)
- Filesize limited to 16MB. (max filesystem size 272MB).

SquashFS

- Read-only filesystem.
- SquashFS
  compresses files, inodes and directories.
- Supports block size upto 1MB for greater compression.
- Very Suitable for small/embedded systems.
- Supports gzip, lzma, lzo and xz (lzma2).
- No Filesize or rootfs size limitation.

Test Platform

- ARM9 S3C2440 FriendlyARM board with 64MB RAM and 64MB NAND Flash.
- Initramfs is used for ease of deducing different performance parameters.
- Different Filesystems are mounted and switch rooted to the corresponding filesystems.

FileSystem Comparison (Boot Time)

	JFFS2 Raw	JFFS2 over UBI	UBIFS	Cramfs (var JFFS2)	Squashfs (var UBIFS) LZO Compression	Squashfs (var UBIFS) XZ Compression
Mount Time	5.962330794	4.736091375	0.098302627	0.019719791	0.020023394	0.02244997
Rootfs Load Time	8.540637016	8.755834818	7.227636385	8.354395413	8.08851521	9.751157379
Total Boot Time	14.50296781	13.49192619	7.325939012	8.374115205	8.108538604	9.773607349

Write-back vs Write-through

Write-back:

- File changes do not go to the flash media straight away.
- They are cached and go to the flash later, when it is absolutely necessary.
- Helps to greatly reduce the amount of I/O which results in better

Write-through:

- File system changes go the flash synchronously.
- Sometimes a small buffer is maintained as a cache but once the buffer is full, it’s flashed immediately.

Write-back vs Write-through

System calls **fsync** and **API fsync()** can provide a file-specific write-through for a filesystem that supports write-back. (i.e. UBIFS)

Also, during mount time, a write-back system can be converted to write-through by changing options in the mount command

i.e. For UBIFS

mount –t ubifs –o sync ubi0:rootfs /mnt

Write Performance (one 10MB file)

	JFFS2	UBIFS	UBIFS with sync
Mount time	5.908775	0.123646021	0.146992922
Big File(10MB) copy	28.20541	25.52733696	29.05287504
Unmount Time	0.164276	0.962749958	0.102820992
Total	34.27846	26.61373293	29.30268896

Write Performance (small files)

	JFFS2	UBIFS	UBIFS with sync
Mount time	5.912980914	0.136153936	0.155074
Copy Small Files (35 files total 5.4MB)	16.87158704	6.717770934	18.912269
Unmount Time	0.052031994	12.38501	0.192489982
Total	22.83659995	19.23893487	19.25983298

Conclusion

- Small embedded systems (low RAM and ROM space): We can use cramfs or squashfs. Squashfs is better as xz compression is supported.

i.e. Small automotive telematics module.

- Full blown embedded systems: All UBIFS or Squashfs for the read-only part and UBIFS for the writable part.

i.e. Automotive Media player, infotainment systems.

- For read-only systems, in system init time, some tmps or RAMFS folder can be mounted for temporary files.

Reference

- Wikipedia
- http://linux-mtd.infradead.org
- FriendlyARM.net
- Linux Kernel Docs.
- http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/Documentation/filesystems/ubifs.txt
- www.embedded-linux.co.uk/