YAFFS
A NAND-flash filesystem

Wookey
Aleph One Ltd, Embedded Debian
wookey@aleph1.co.uk

Intro

- Who’s who
- Project Genesis and history
- Flash primer - NOR vs NAND
- How it works (log structure, GC, ECC)
- Porting and Use
- Performance, and JFFS2 comparison
- Licencing
- YAFFS2

Project Genesis

- TCL needed a reliable FS for NAND
- Considered Smartmedia compatibility
- Considered JFFS2
  - Better than FTL
  - High RAM use
  - Slow boot times

History

- Decided to create YAFFS - Dec 2001
  - Working on RAM emulation - March 2002
  - Working on real NAND (Linux) - May 2002
  - WinCE version - Aug 2002
  - ucLinux use - Sept 2002
  - Linux rootfs - Nov 2002
  - pSOS version - Feb 2003
  - Shipping commercially - Early 2003

Flash primer - NOR vs NAND

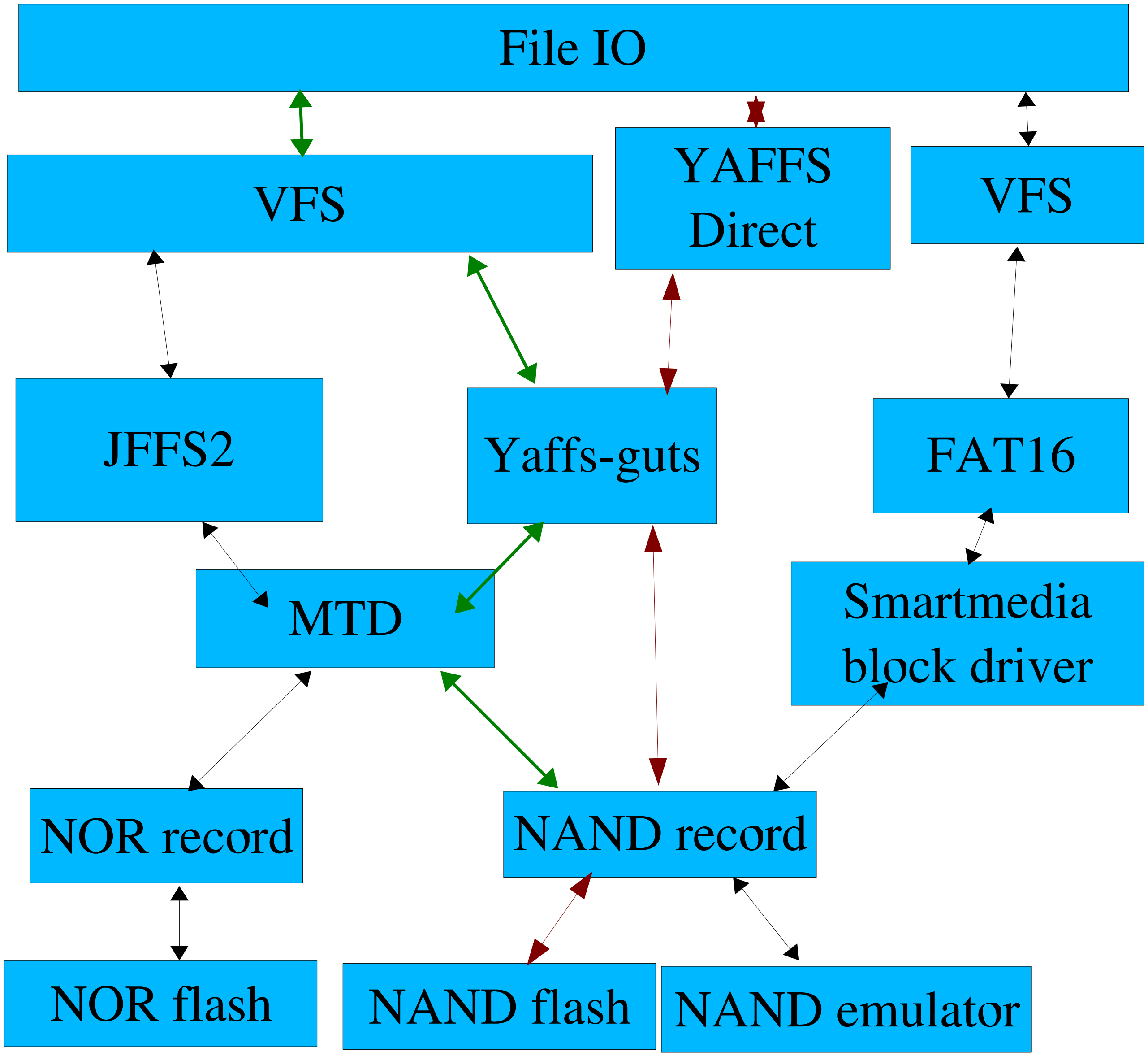
NOR	NAND
Linear random access Like RAM Can do XIP	Page access Almost like a disk. Can't be used for XIP Each page has 16-bytes of extra management data
$2/MByte	$0.5/MB
Low density (approx 8MB max device size)	Higher density (approx 128MB max device size)
Erasable blocks of 8kB to 128kB typical	Erasable blocks of 32x512-byte pages.
Endurance 100k to 1M erasures	Endurance 100k to 1M erasures
Erase time 1second/erasable block	Erase time 2ms
Designed as ROM replacements	Designed as mass storage replacements
Byte-by-byte programming. No limit on writes.	Page or partial-page programming. Limit on max number of page writes (3-10)before block must be erased.
Program byte to change 1s to 0s. Erase block to change 0s to 1s.	Program page to change 1s to 0s. Changing 0s to 1s requires an erasure.
Random access programming	Pages must be written sequentially within a block
No bad blocks when delivered, but the devices wear out. Thus file systems should be fault tolerant.	Bad blocks are expected when the devices are delivered. Further degradation is expected with use. Thus fault tolerance is an absolute necessity.

Design approach

- OS neutral and developed in user space
- Developed on NAND emulator
- Portable - OS interface, guts, hardware interface (diag)

- Single threaded (don’t need separate GC thread like NOR)
- Journalling - Tags break down dependence on physical location
- Avoid in-place re-writing - expensive due to erase size

Architecture



Terminology

Flash-defined

- Page - 512 byte flash page
- Block - Erasable set of pages (usually 32)

YAFFS-defined

- Chunk - YAFFS tracking unit. ==page (for YAFFS1)

Filesystem Design

- Each file has an id - equivalent to inode. id 0 indicates ’deleted’
- File data stored in chunks, same size as flash pages (512 bytes)
- Chunks numbered 1,2,3,4 etc - 0 is header.
- Each flash page is marked with file id and chunk number
- These tags are stored in the OOB - 64bits: including file id, chunk number, write serial number, tag ECC and bytes-in-page-used
- On overwriting the relevant chunks are replaced by writing new pages with new data but same tags - the old page is marked ’discarded’
- File headers (mode, uid, length etc) get a page of their own (chunk 0)
- Pages also have a 2-bit serial number - incremented on write
  - Allows crash-recovery when two pages have same tags (because old page has not yet been marked ’discarded’).
- Discarded blocks are garbage-collected.

Filesystem Limits

YAFFS

- 2^18 files (>260,000)
- 2^20 max file size (512MB)
- 1GB max filesystem size

YAFFS2

- 8GB max filesystem size

Devices, hardlinks, softlinks, pipes supported

OOB data

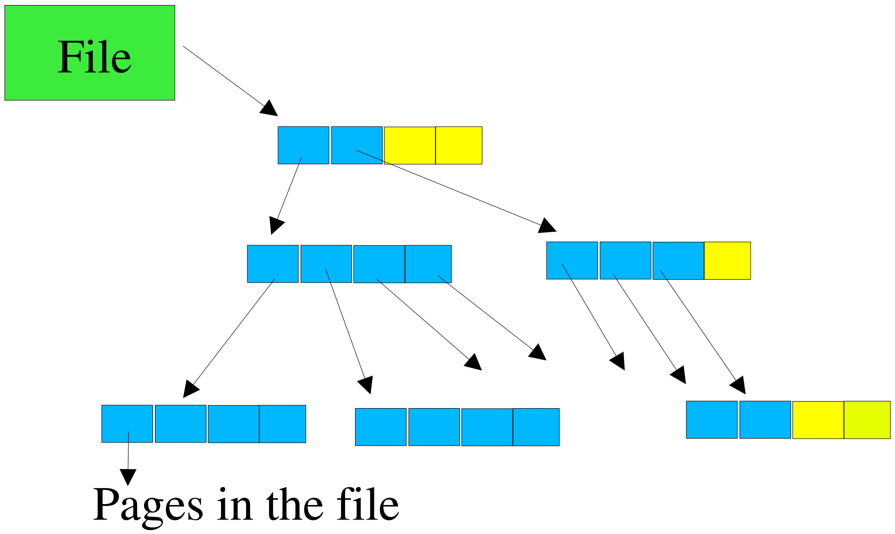
16 bytes. YAFFS/Smartmedia or JFFS2 ECC

Byte	Smartmedia	YAFFS
0-511	Data	**Data**
512-515	Reserved	**Tags**
516	Data status	**Data status**
517	Block status	**Block status**
518-519	Block address	**Tags**
520-522	ECC on 2nd 256 bytes	**ECC on 2nd 256 bytes**
523-524	Block address	**Tags**
525-527	ECC on 1st 256 bytes	**ECC on 1st 256 bytes**

Filesystem RAM Data Structures (1)

- Not fundamental - needed for speed
- 16-bit page address for each file
  - 1-1 mapping on 32Mb NAND -
  - block of 4 pages on 128Mb NAND - scan
- yaffs_Object - file, dir, hardlink, softlink
- yaffs_ObjectHeader - in OOB, corresponds to yaffs_object. Each Object has parent, siblings (linked list), children if directory

Filesystem RAM Data Structures (2)



- yaffs_Tnode tree of data chunks in a file. As the file grows in size, the levels increase. The Tnodes are a constant size (32 bytes). Level 0 (ie the lowest level) comprise 16 2-byte entries giving an index used to search for the chunkId. Tnodes at other levels comprise 8 4-byte pointer entries to other tnodes lower in the tree.

data structures are allocated in groups to reduce allocation/freeing overhead

Partitioning

- Internal - give start and end block
- MTD partitioning (partition appears as device)

Garbage Collection and threads

- When a block (1-4 pages) contains only discarded pages - collect it
- If only one valid page than can copy it to release block for collection

- Single threaded
- Gross locking, matches NAND
- Soft Background deletion -
  - Delete/Resize large files can take up to 0.5s
    Incorporated with GC
  - Spread over several writes

ECC

- NAND is unreliable - bad blocks, data errors
- Needs Error Correction Codes for reliable use
- ECC on Tags and data
- 22bits per 256 bytes, 1-bit correction
- CPU/RAM intensive
- Lots of options:
  - Hardware or software
  - YAFFS (slightly faster) or MTD
  - JFFS2 or YAFFS/Smartmedia positioning (if using MTD)
- Make sure bootloader, OS and FS generation all match!
- Can be disabled - not recommended!

OS portability

- Linux
- WinCE
- pSOS
- ThreadX
- DSP-BIOS

- Reliable FS on NAND flash (un-corruptible, ECC, wear leveling, bad blocks)
- Good for battery devices - (power fail)

Yaffs in Use

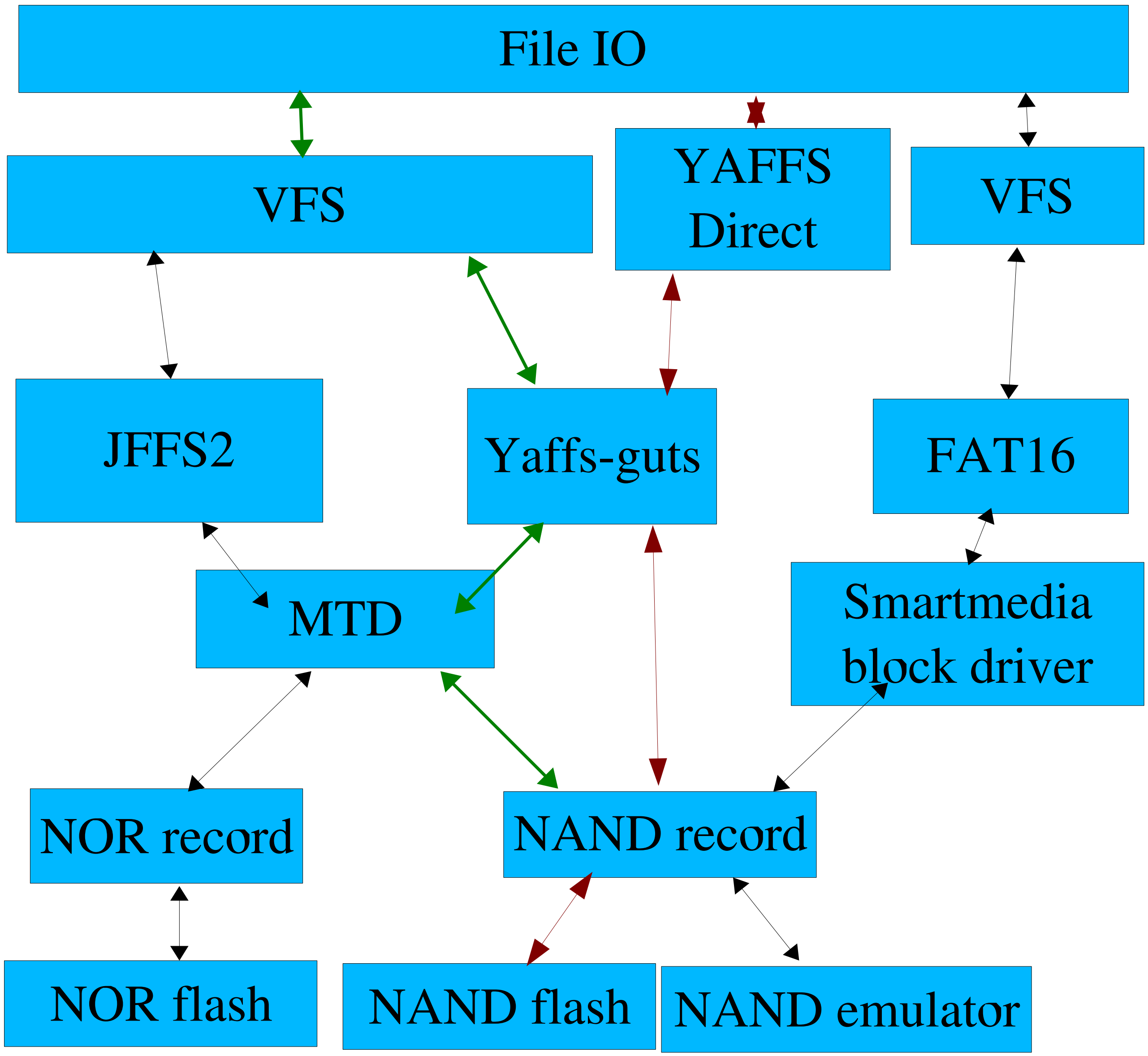
- Formatting a device/partition for Yaffs simpy consists of blanking it
- Creating a filesystem image needs to generate OOB data
  - mkyaffsimage tool - generates images
  - mkyaffs - makes partition, and can fill with fs image
- In Linux - kernel module
- YAFFS Direct - supply 7 functions

Embedded system use - YAFFS Direct Interface (1)

- YDI replaces Linux VFS/WinCE FSD layer
- open, close, stat, read, write, rename, mount etc
- Caching of unaligned accesses

- Port needs 7 functions:
  - Lock and Unlock (mutex)
  - current time (for time stamping)
  - NAND access (read, write, init, erase).

Embedded system use - YAFFS Direct Interface (1b)



Embedded system use - YAFFS Direct Interface (1)

- YDI replaces Linux VFS/WinCE FSD layer
- open, close, stat, read, write, rename, mount etc
- Caching of unaligned accesses

- Port needs 7 functions:
  - Lock and Unlock (mutex)
  - current time (for time stamping)
  - NAND access (read, write, init, erase).

Embedded system use - YAFFS Direct Interface (2)

- No CSD - all filenames in full
- Case sensitive
- No UID/GIDS
- Flat 32-bit time
- Thread safe - one mutex
- Multiple devices - eg /ram /boot /flash

Performance - Examples

- 200Mhz StrongARM, 100Mhz SDRAM, 2*64MB NAND (winCE)
  - 1.5-200x comercial FAT-based FS
  - Flat-out write speed (1MB writes). 1.2MB/s.
  - YAFFS-to-YAFFS copy using the WinCE file explorer. 64kB chunks. 500kB/s.
  - Copy 11MB file from host over USB ActiveSync. Approx 26 seconds - most of which is USB transfer time.
  - Delete an 11MB file: 5 seconds.
- 24Mhz ARM720 prototype, 16bit 12MHz RAM bus, SW ECC.
  - Read: 185Kb/s
  - Write: 85Kb/s   Write (no verify): 175Kb/s

Performance - Caching

- Linux VFS has cache, WinCE and RTOS don’t
- YAFFS internal cache
  - 15x speed-up for short writes on WinCE
- Choose generic read/write (VFS) or direct read/write (MTD)
  - Generic is cached (_usually_ reads much faster ~10x, writes 5% slower)
  - Direct is more robust on power fail

JFFS2 and YAFFS comparison

- JFFS2 - 16 bytes per node (48 in JFFS) (4MB for 128MB)
  YAFFS - 2 bytes per page (512K for 128MB)
- 128MB NAND scan on boot - 25s Boot scan only reads OOB - 3s
- compressed uncompressed
- complex (especially GC) simple
- Linux, ecos Lots of OSes, easy to port
- boot: JFFS2 4s for 4MB(8MB), YAFFS:7s for 30MB (twice as fast)

Licensing

- GPL - patents, Good Thing (TM)
- Bootloader LGPL to allow incorporation

- YAFFS in proprietary OSes - WinCE4, pSOS
  - Wider use
  - Aleph One Licence - MySQL-style: ’If you don’t want to play then you can pay’

YAFFS2

- Specced Dec 2002
- For new NAND chips.
  - 2k pages
  - no re-writing
  - simultaneous page programming
  - 16-bit bus on some parts
- Main difference is ’discarded’ status tracking

  zero re-writes means can’t use ’discarded’ flag

  - Instead track block allocation order (with sequence number)
    - Delete by making chunks available for GC and move file to special ’unlinked’ directory until all chunks in it are ’stale’
    - GC gets more complex to keep ’sense of history’

YAFFS2 comparison with YAFFS1

- write: 1-3x faster (1.5-4.5MB/s vs 1.5MB/s
- read: 1-2x faster (7.6-16.7MB/s vs. 7.6MB/s)
- delete: 4-34x faster (7.8-62.5MB/s vs. 1.8MB/s)
- Garbage collection: 2-7x faster (2.1-7.7 vs. 1.1MB/s)
- RAM footprint 25%-50% less
  - Slowest figures are for 512b pages (like YAFFS1)
  - Faster ones for 2K pages and 2K pages+16bit bus
- Development sponsorship would help a lot

YAFFS
A NAND-flash filesystem

Questions?

Wookey
Aleph One Ltd, Embedded Debian, Balloon Project
wookey@aleph1.co.uk