Intrinsyc Software

Linux on eMMC

Optimizing for Performance

Ken Tough
Principal Engineer
ktough@intrinsyc.com

Agenda

* What's unique for eMMC (the black box)
* Effect on performance (inside the box)
* Improving performance
  * MMC driver
  * Block I/O
  * Scheduler
  * File Systems
  * User Space
* Future, Q/A

What is eMMC?

* Solid state storage device on MMC bus
* Chip on PCB
* NAND flash based
* Block-based (sector) storage

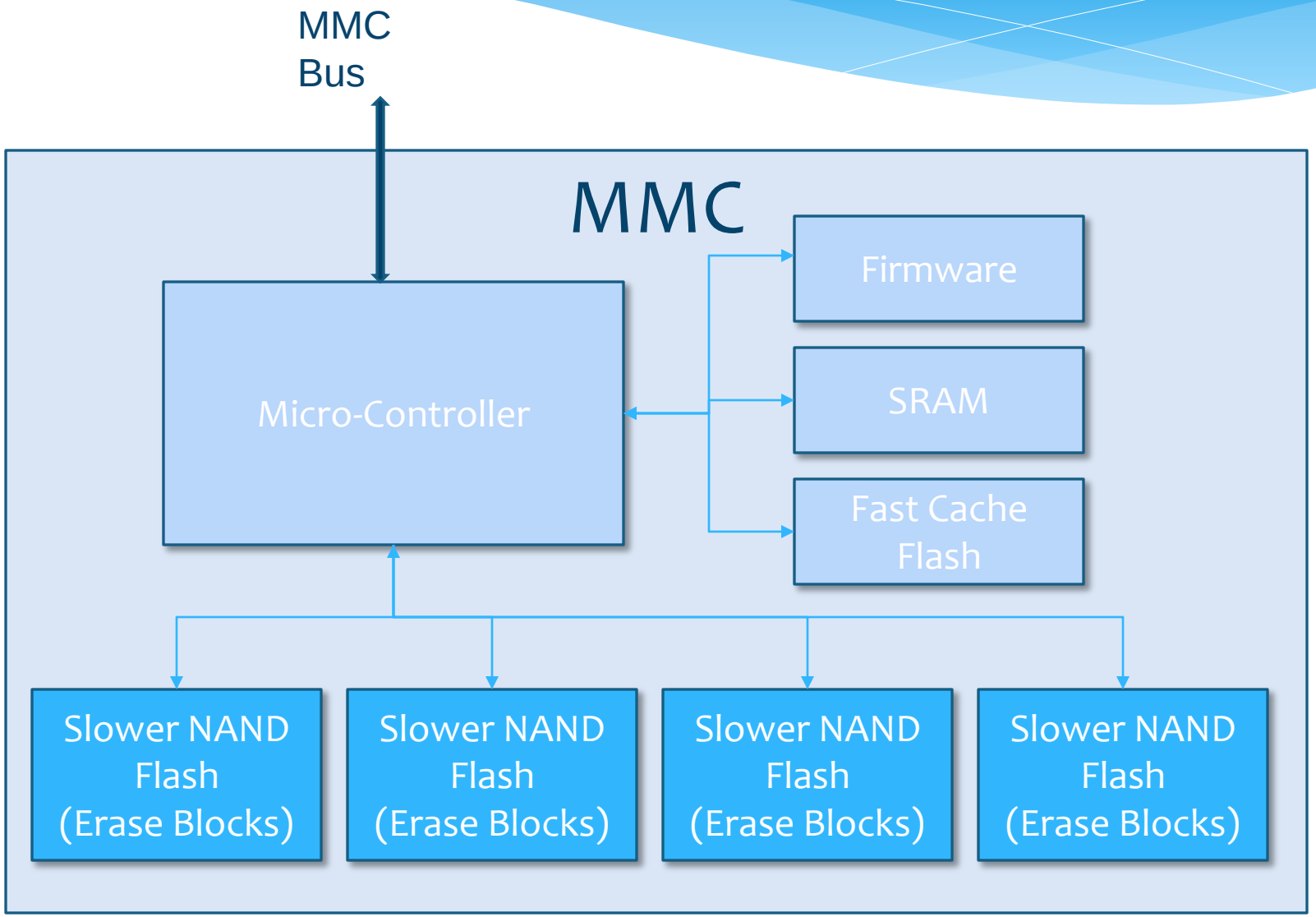
Why eMMC matters

* Popular on embedded devices
* Cheap
* Flexible
* In theory, more "generic"

eMMC scenarios

* Tablets, smart phones with lots of DRAM
* Netbooks with lots of DRAM
* Multimedia players, USB memory sticks

Inside



Inside the eMMC

* NAND flash arranged in pages
* Controller with temporary storage
* Wear levelling
* Free space management

eMMC characteristics

* Fast read access
* Fast read seek times
* Acceptable sequential write performance
* Poor random write performance

Discard (TRIM)

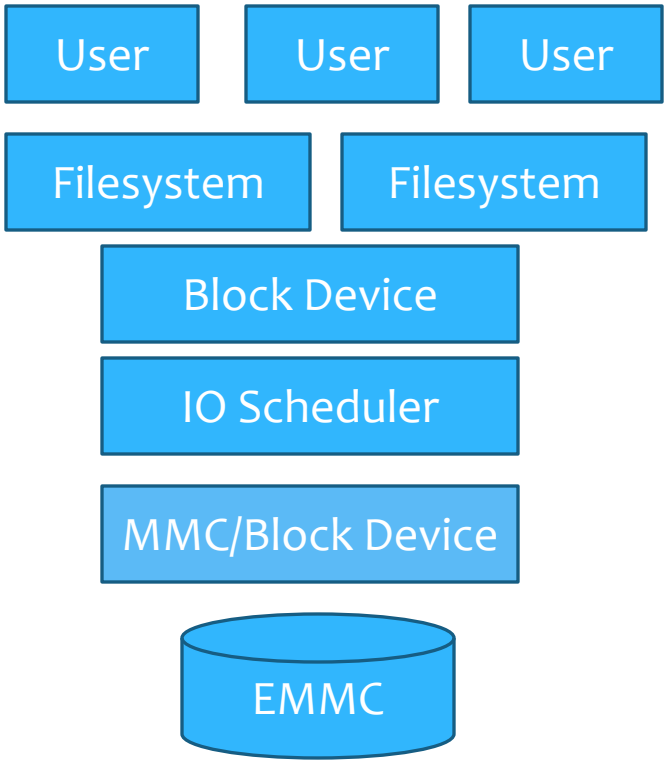
* eMMC TRIM command
* Tells controller what is free
* Frees up erase blocks & internal resources
* TRIM blocks on format
* Not same as "erase" command

eMMC spec performance

* Typically emphasizes sequential write performance
* Random accesses hit eMMCs internal pipelines
* Frequently limited by eMMC's Random IOPs limit
* Minimum OP time regardless of OP size
* Not often data BW limited
* <200 IOPs (e.g. 4kB per OP)
* Analyze application's eMMC read/writes patterns

Areas of Focus

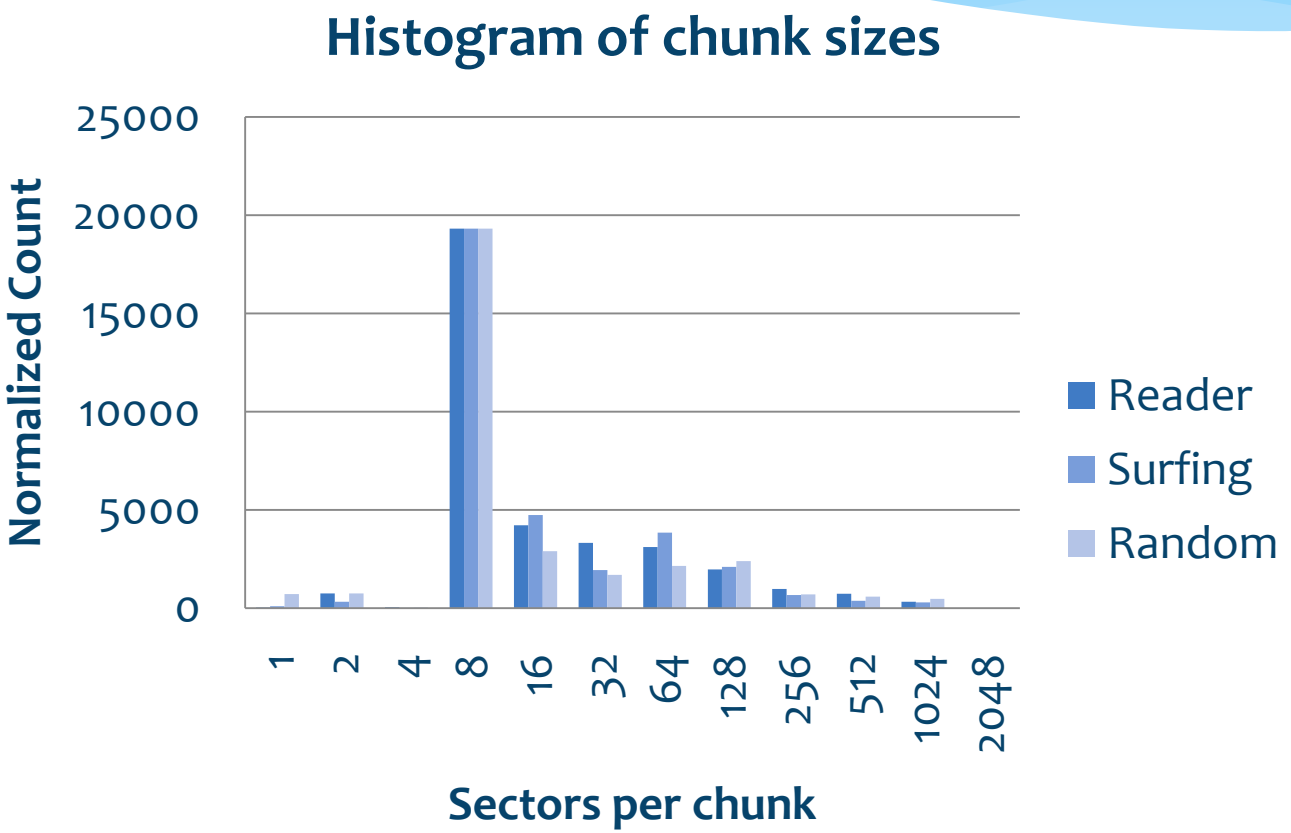
* User space
* Filesystem type
* Filesystem layout
* Block IO & Cache
* IO Scheduler
* MMC bus driver



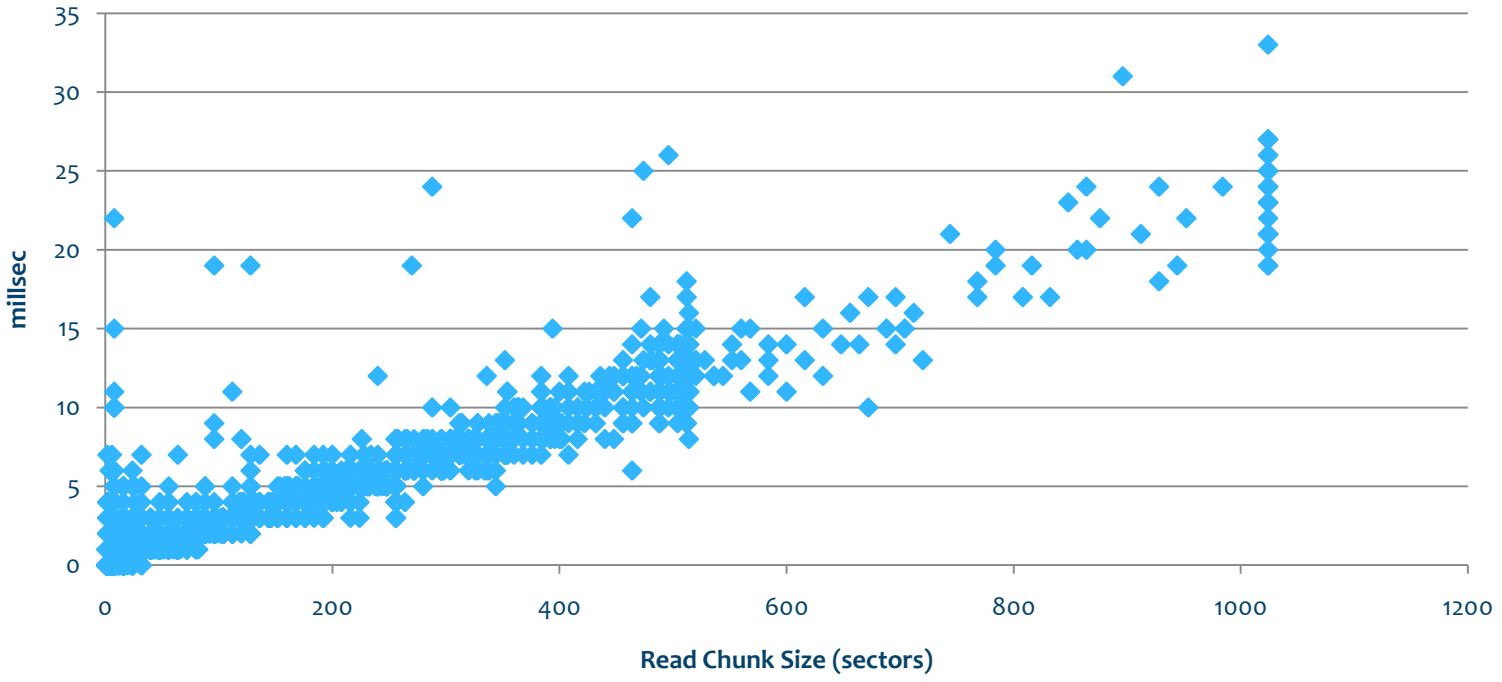
MMC driver

* Maximum bandwidth enabled (8-bit, 50MHz)
* Enable DMA if option
* Power management
* Trim / vendor command support
* Benchmarking Log Device

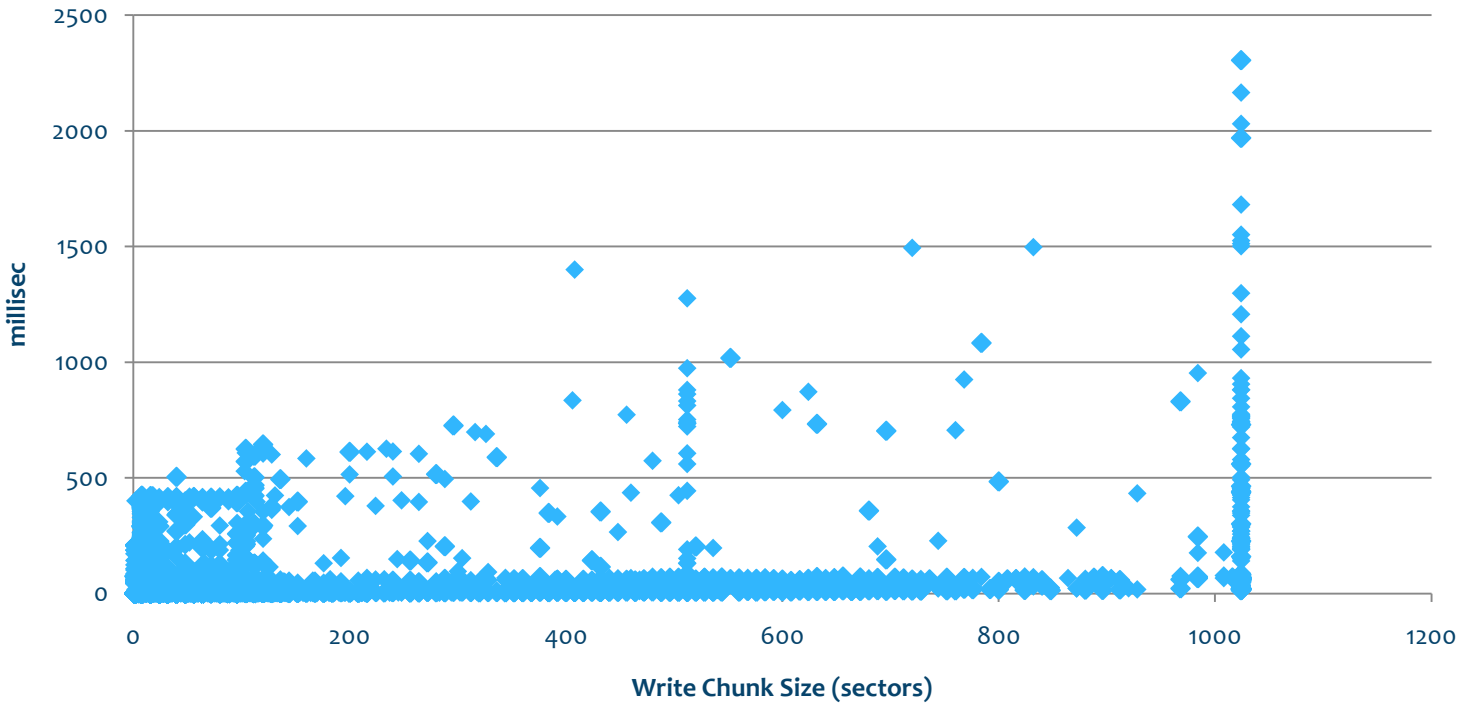
Analysis at MMC/Block Level



eMMC Read Times



eMMC Write Times



Vendor Performance

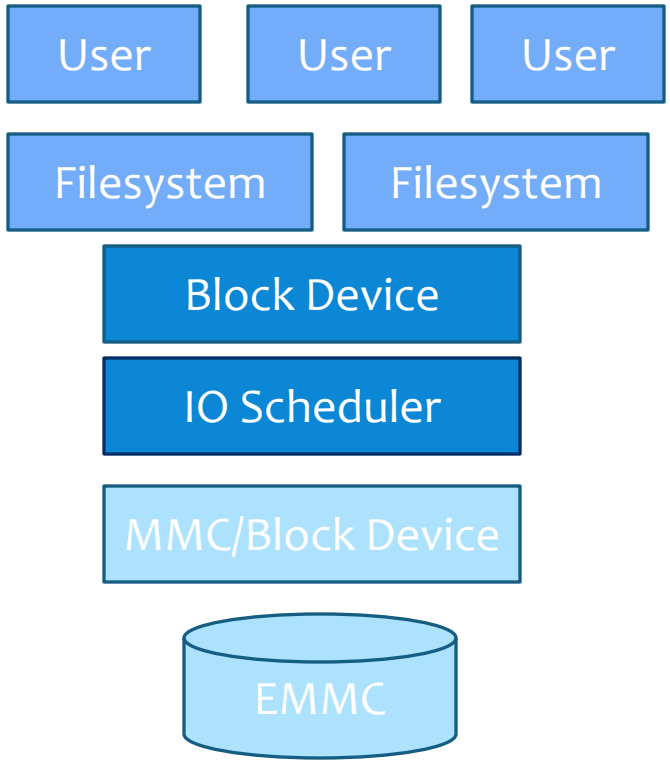
* Wide variation in read/write times
* Big dependency on internal eMMC firmware
* Power Class support
* Geometry / technology
* Trim support

MMC v4 High Priority Interrupt

* Allows reads to bypass long writes
* Useful in very specific applications
* Small RAM
* Page/Block cache and IO Scheduler
* Internal eMMC Pipelines blocked anyway
* Multimedia apps and “long” buffering

Areas of Focus

* User space
* Filesystem type
* Filesystem layout
* **Block IO & Cache**
* **IO Scheduler**
* MMC bus driver



Cache is King

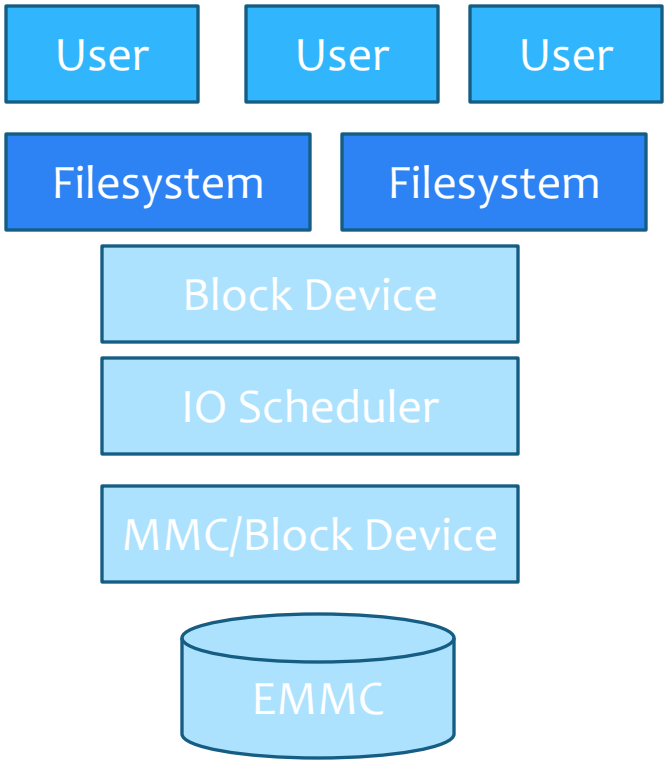
* Alleviates write performance issues
* Improves read times even further
* Reduces NAND wear

I/O schedulers

* CFQ, noop, deadline
* Results are similar within ~10% range
* QOS considerations are more important than throughput

Areas of Focus

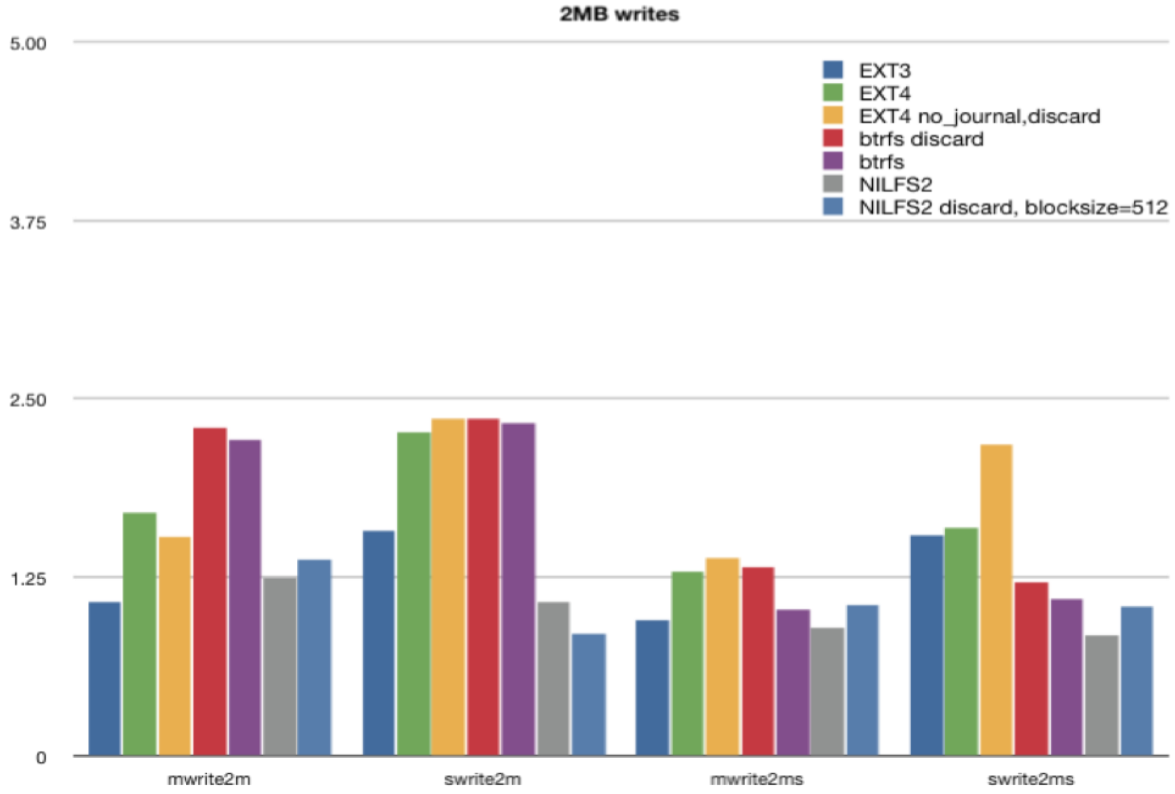
* User space
* **Filesystem type**
* **Filesystem layout**
* IO Scheduler
* Block IO & Cache
* MMC bus driver

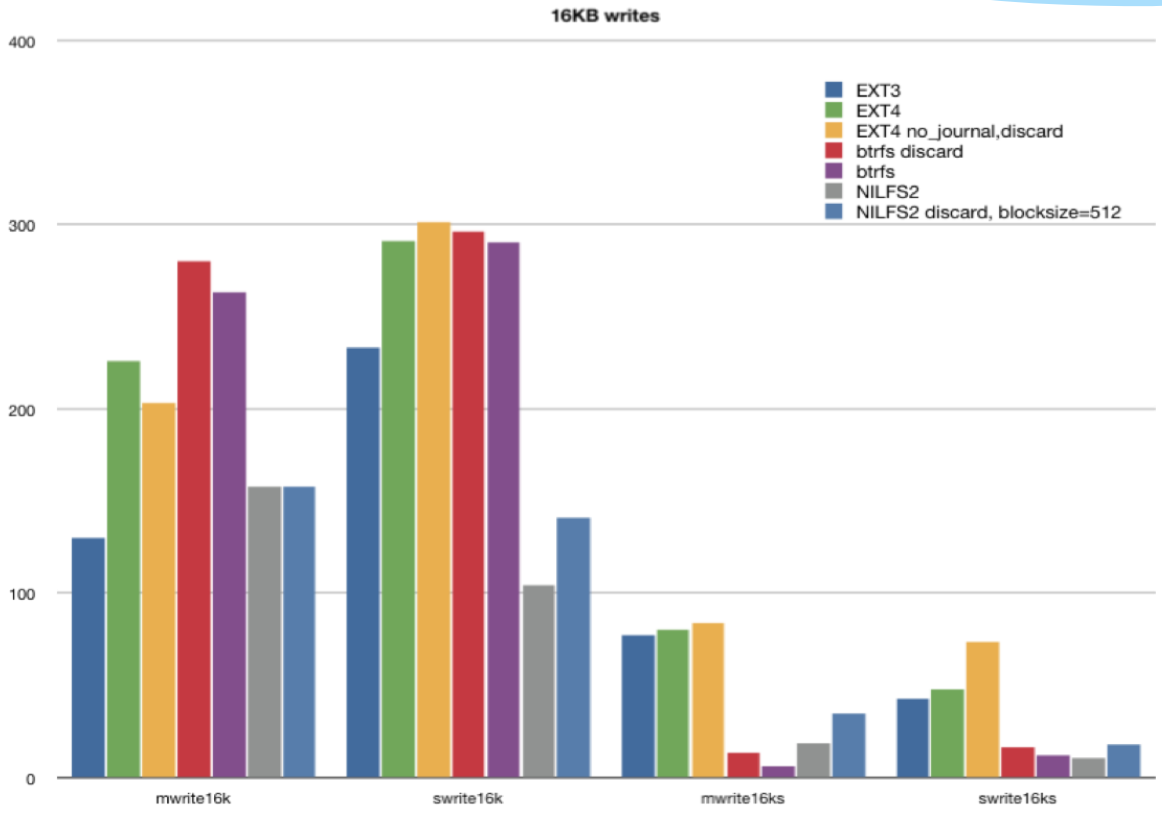


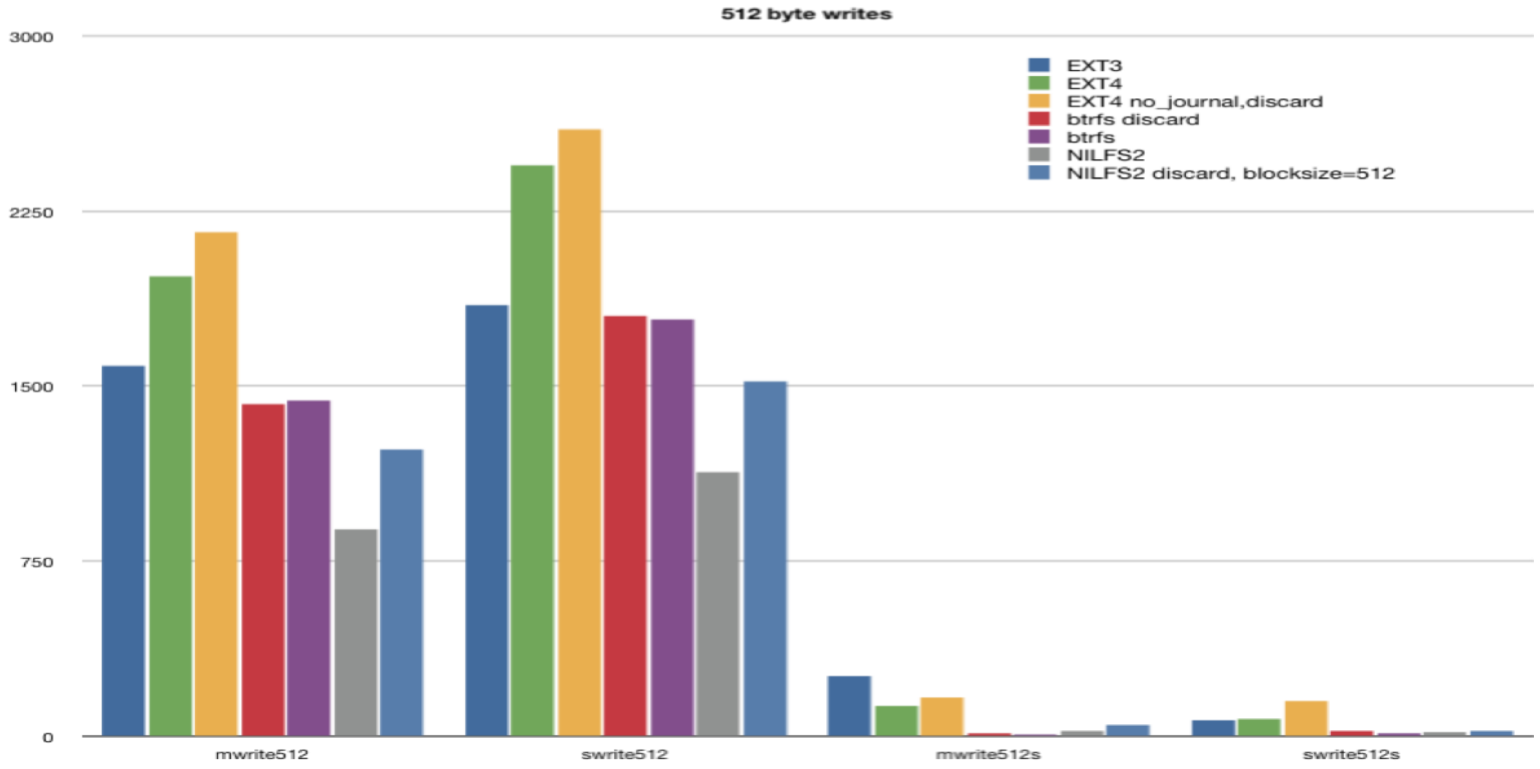
Filesystems

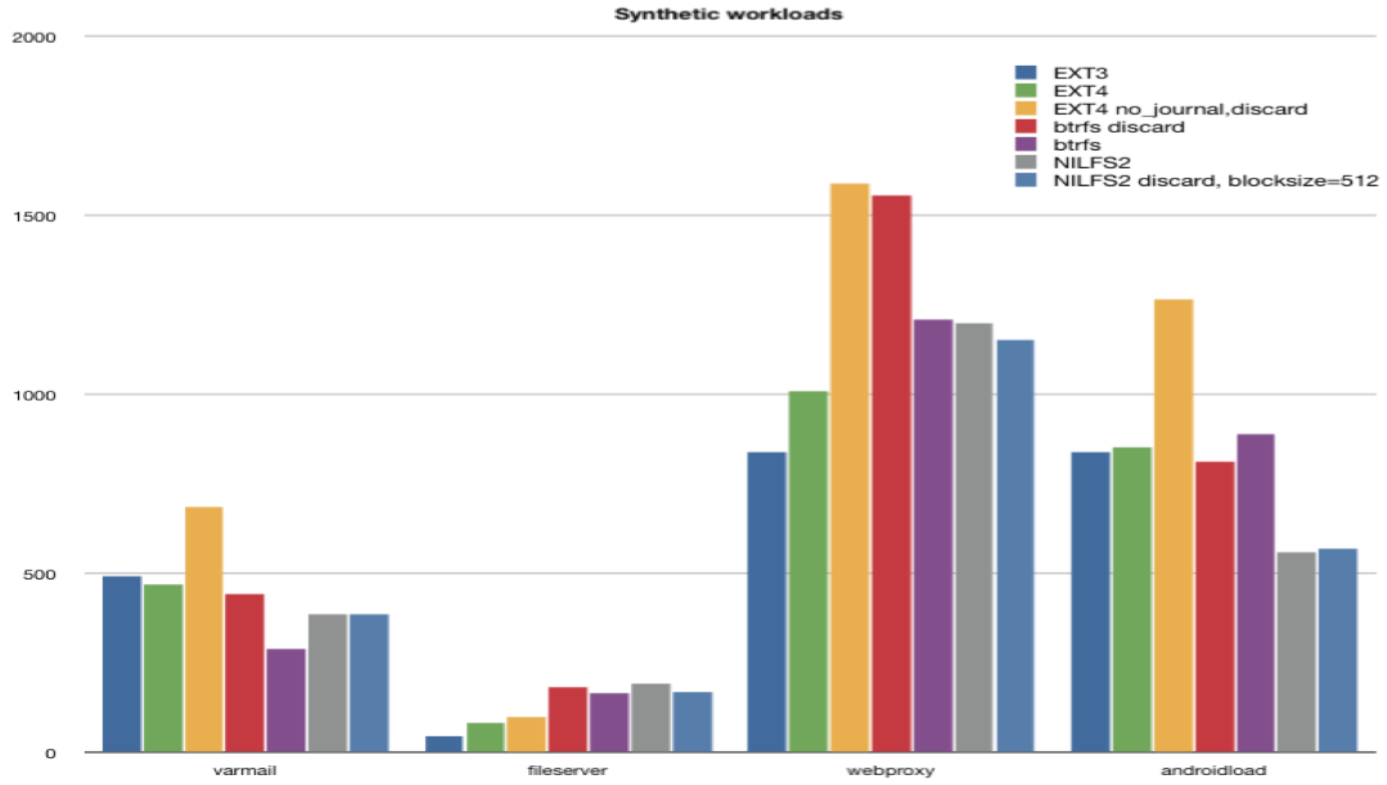
* Focus on write performance
* Tests run using fsbench (3.0 kernel, OMAP3 eReader tablet, 8GB MMC, 512 MB RAM)
* Various low-level and high-level scenarios modelled
* EXT4, BTRFS, NILFS2 tested

Filesystem Benchmarks

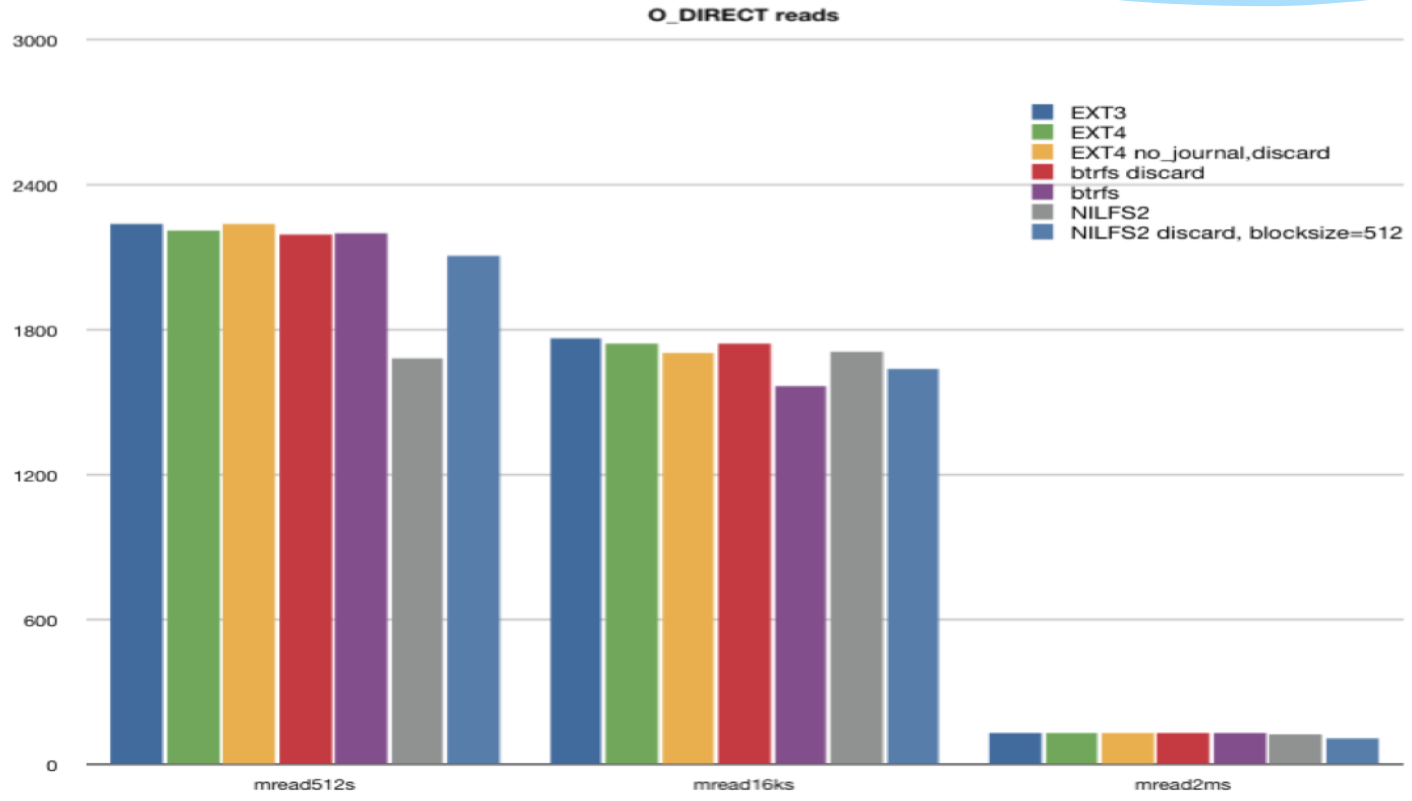



16KB writes
400
300
200
100
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
mwrite16k
swrite16k
mwrite16ks
swrite16ks


512 byte writes
3000
2250
1500
750
0
EXT3
EXT4
EXT4 no_journal,discard
btrfs discard
btrfs
NILFS2
NILFS2 discard, blocksize=512
mwrite512
swrite512
mwrite512s
swrite512s



INTRINSYC ENABLING INTELLIGENT CONNECTIVITY



INTRINSYC ENABLING INTELLIGENT CONNECTIVITY

EXT4 - a write

* Journal write (usually ~16K)
* inode update (usually 4K)
* Data goes into page cache

BTRFS - a write

* Update non-sync very fast
* Sync write puts tree leaves on eMMC
* Sync write is 4 non-sequential writes

NILFS2 - a write

* Log structured filesystem
* Stores the ‘update’
* One large (40K+) write
* Eventually “snapshot” needs flushing
* Initialization
* Recovery

EXT4 w/o journal

* Not too dangerous on embedded systems with fixed battery
* Good performance due to improved sequentiality

BTRFS

* If not using a lot of fsync/fdatasync
* Great large write performance
* Terrible on small/medium sync writes
* Good performance on multiple writes

NILFS2

* Consistent performance
* Potentially much faster if eMMC part has fast sequential performance
* Should theoretically be the fastest :-)

EXT4 with journal

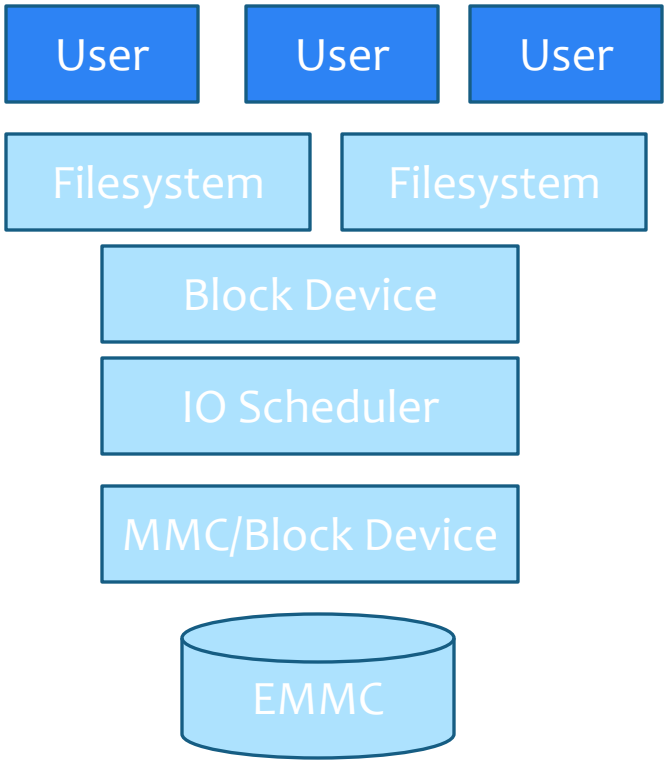
* Consider RAM journal (warm DRAM reset)
* Tune journal flush timers
* Better than BTRFS on small/medium sync writes

Filesystem layout

* No swap
* Align partitions to erase block boundaries
* Extents match erase blocks
* System design (multiple storage devices)

Areas of Focus

* **User space**
* Filesystem type
* Filesystem layout
* IO Scheduler
* Block IO & Cache
* MMC bus driver



User space

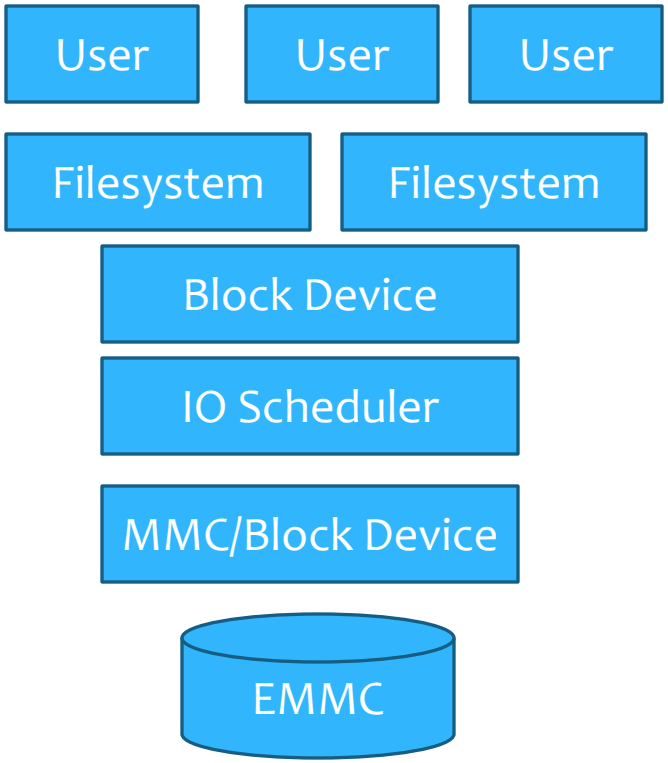
* Avoid synchronization on files (e.g. SQLite)
* Avoid sync/fsync/fdatasync/etc (batch transactions)
* Avoid small writes to files, better to buffer
* Don’t be afraid to read, be afraid to write!

Future

* Linaro project (www.linaro.org) working on improving eMMC experience
* eMMC 4.5 brings METADATA
* Effect of dirty throttling?

Summary

* User space
* Filesystem type
* Filesystem layout
* IO Scheduler
* Block IO & Cache
* MMC bus driver



Conclusion

* EXT4 (discard, ram/no journal) is probably your best bet
* Try out a couple of configurations for the eMMC you are targeting
* Benchmark per Vendor
* Evaluate vendor commands
* Avoid writes! :-)

Questions?