TOSHIBA

Leading Innovation >>>

Evaluation of Flash File Systems for Large NAND Flash Memory

TOSHIBA CORPORATION
Embedded System Technology Development Dept.
Core Technology Center
Toru Homma

4/6/2009 CELF Embedded Linux Conference

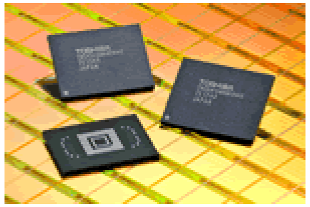
Agenda

- Background
- Purpose
- File system software block
- Overview of the different Flash file systems
- Testing environment
- Benchmark result
- Summary
- References

Background

- **NAND flash memory is commonly used in embedded systems.**
- **The falling price of NAND device encourages us to use large memories (e.g. Larger than 128MB).**

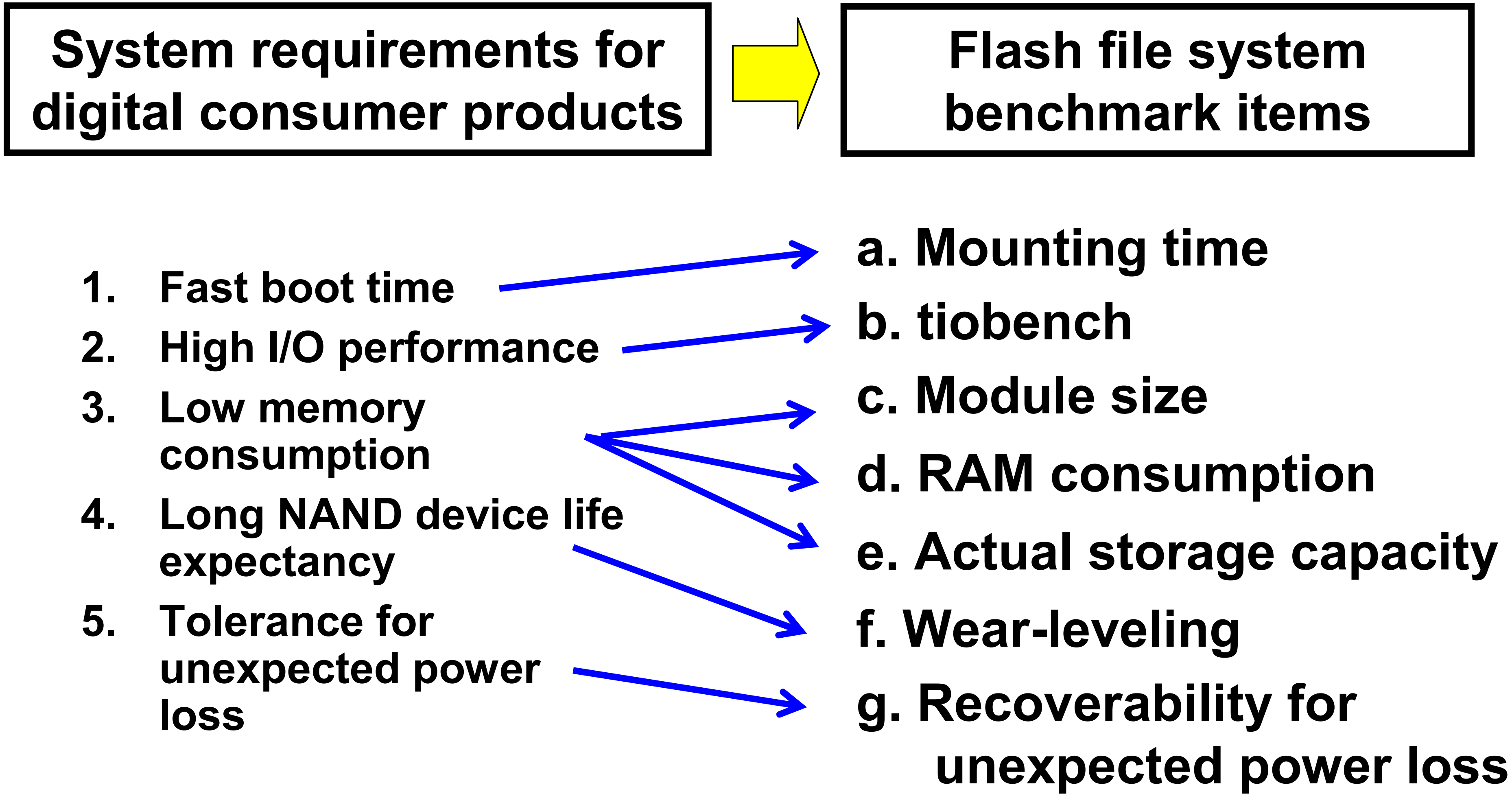
- **Limitations of bare NAND flash memory devices –**
  - Block erasure
    - finite number of erase-write cycles (~10K cycles and MLC is less)
  - Normal operations
    - Bit flip possibilities



➡ ***Important to use suitable file system***

- **There are some cases for previous file systems that do not fit large NAND systems.**
- **Defining system requirements and then breaking them down to individual benchmark items.**
- **Comparing each file system.**

Purpose



System requirements for digital consumer products

Flash file system benchmark items

1. **Fast boot time**
2. **High I/O performance**
3. **Low memory consumption**
4. **Long NAND device life expectancy**
5. **Tolerance for unexpected power loss**

a. Mounting time

b. tiobench

c. Module size

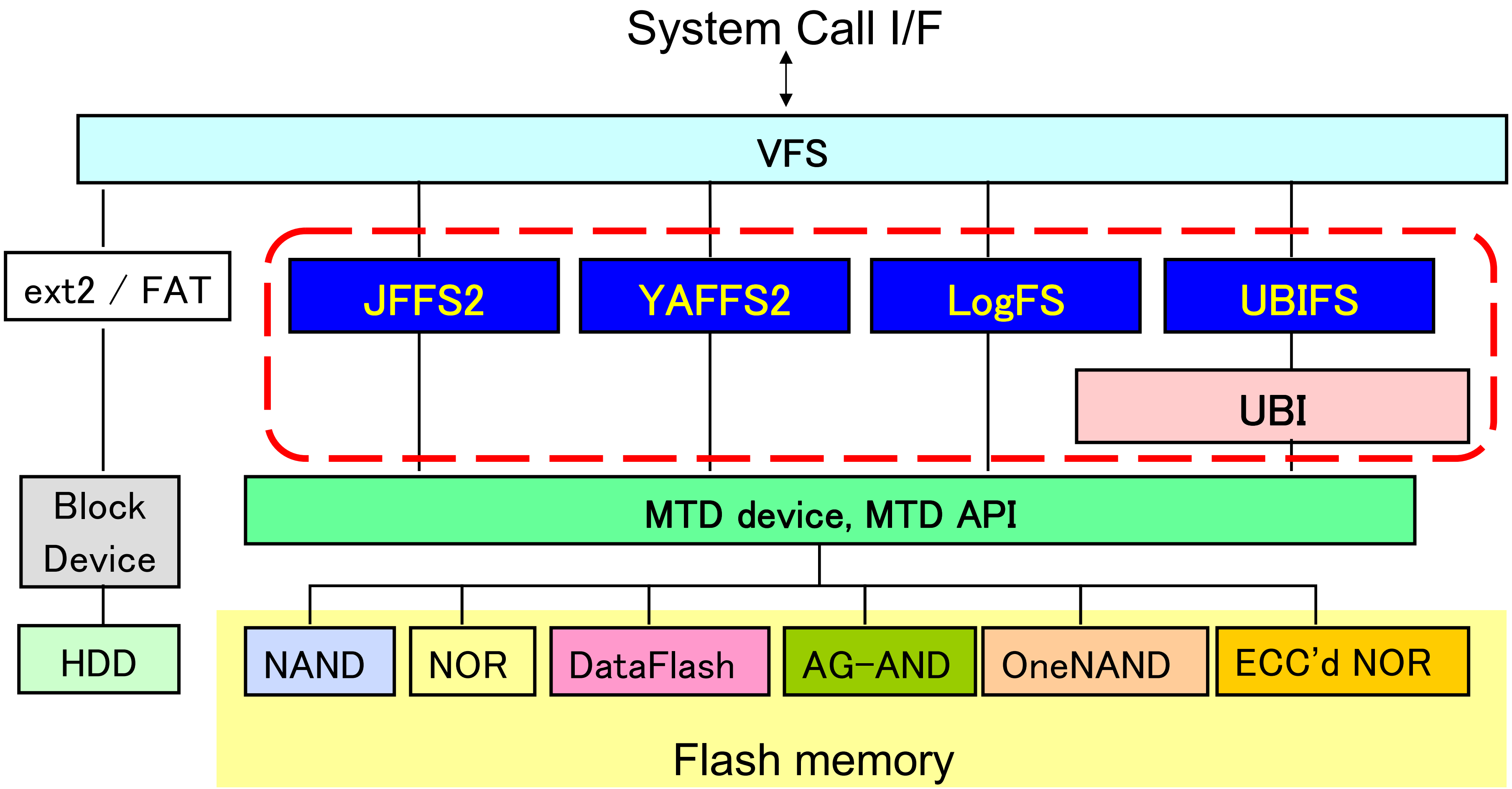
d. RAM consumption

e. Actual storage capacity

f. Wear-leveling

g. Recoverability for unexpected power loss

Flash file system software block



VFS: Virtual File System
MTD: Memory Technology Device

Overview of the different Flash file systems

- **JFFS2 : Journaling Flash File System version 2**

  (David Woodhouse)
  - **Has been integrated in Linux kernel since 2001.**
  - **Commonly used for low volume flash devices.**
  - **Compression is supported.**
- **YAFFS2 : Yet Another Flash File System version 2**

  (Charles Manning)
  - **YAFFS is the first file system designed specifically for NAND (since 2001).**
  - **Version 2 supports 2KB large page NAND (since 2005).**
  - **Compression is not supported.**
- **LogFS : Log Flash File System**

  (Jörn Engel)
  - **Mounting time is short (since 2005)**
  - **Under development（Needs more testing on large devices）**
  - **User data is not compressed, while meta data is compressed.** *(Jörn said that user data is also compressed in ELC2009, but we could not see it in our testing. We used the default settings.)*
- **UBIFS : Unsorted Block Image File System**

  (Artem Bityutskiy, Adrian Hunter)
  - **Mainlined in 2.6.27 in Oct 2008.**
  - **Works on top of UBI volumes.**
  - **Compression is supported.**

Testing environment

- **Software**
  - Vanilla kernel + Original patch for embedded systems
  - Linux kernel : 2.6.27.9 (JFFS2, YAFFS2, UBIFS), 2.6.25.10 (LogFS)
  - NAND driver : ECC is done by software.
- **Hardware**
  - Board : Digital product development board

CPU	MIPS 327 MHz (I$/D$: 64 KB/64 KB)		
RAM (Kernel)	256 MB (32MB)		
NAND	Bus	8 bit	
	Regions	Data	Out of band
	Total size	256 MB	8 MB
	Erasing block	128 KB	4 KB
	Page	2 KB	64 B
	Sub-page	512 B	16 B

– NAND performance（MTD character device direct access）

Erase	Read	Write
10.61	2.50	2.00

[MB/s]

Benchmark result – Fast boot time

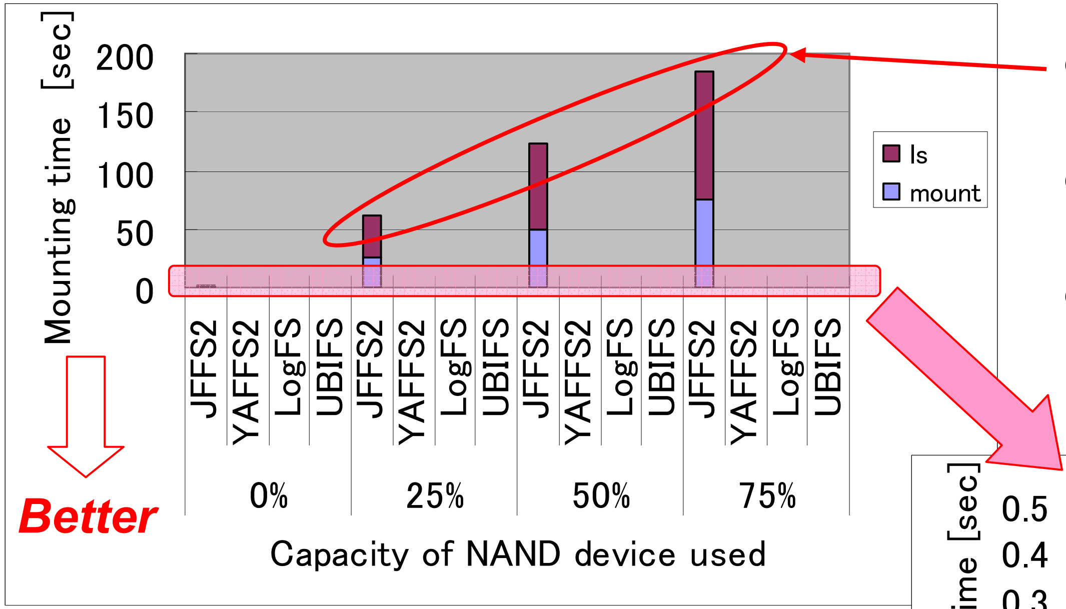
(a) Mounting time

- **Mounting time is important for boot time.**
- **Comparing the NAND device readiness**
  - Time taken from device mount to completion of “ls” command.
- **Comparing 4 patterns of NAND device used**
  - 0% (0MB), 25% (64MB), 50% (128MB), 75% (192MB)
  - One file is stored for each case.
- **Configurations**
  - Following settings are used for making the same conditions:

JFFS2	YAFFS2	LogFS	UBIFS
No compression	Default	Default	No compression

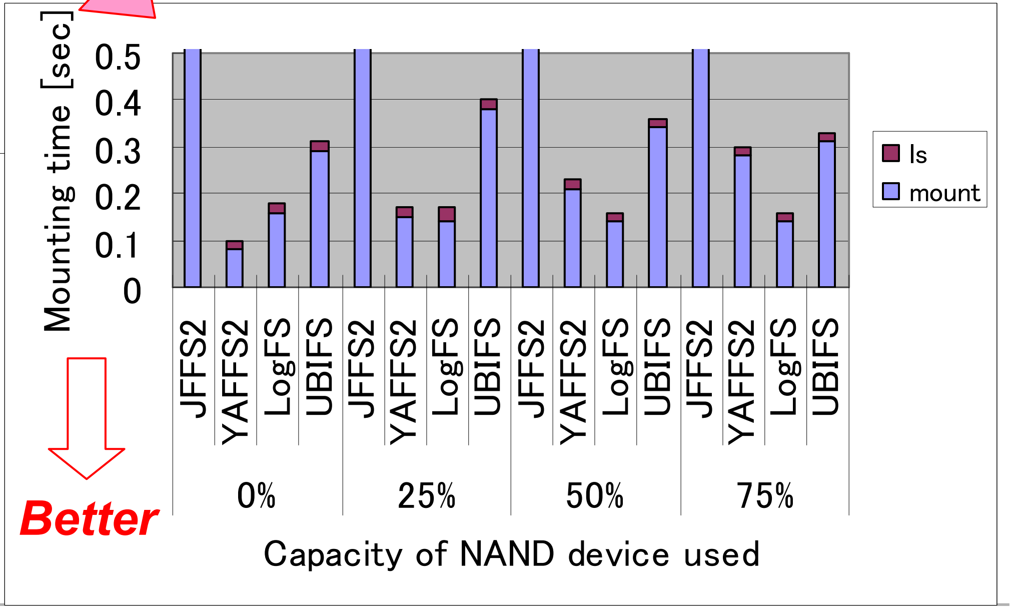
Benchmark result – Fast boot time (cnt'd)

(a) Mounting time (cnt'd)



- Scan takes time for **JFFS2** mounting time. It takes 180sec for the 75% case.
- **YAFFS2**, **LogFS**, and **UBIFS** are within 0.4 sec.
- **YAFFS2** mounting time increases linearly in terms of the capacity of NAND device used.

- **LogFS** stores the tree structure in the flash device so that mounting time does not depend on the used capacity.
- **UBIFS** mounting time is not affected by the used capacity. UBI initialization time linearly depends on the number of PEB, which does not affect on this testing.



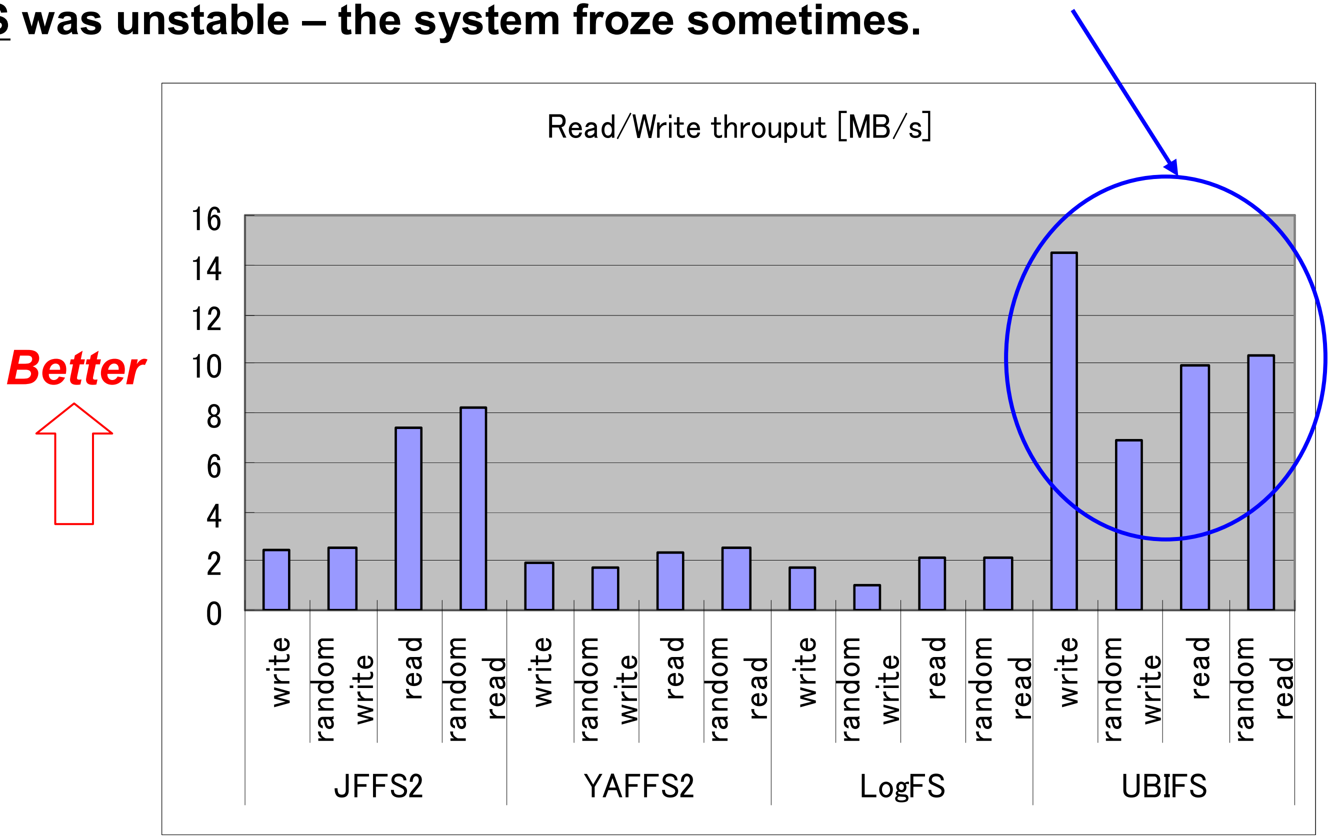
Benchmark result – I/O performance

(b) Tiobench – Read/write throughput w/ 128KB block size

Tiobench parameters : 1 thread, no sync, 192MB for sequential 64MB for random.

UBIFS has the highest throughput because of write-back caching support.

LogFS was unstable – the system froze sometimes.



configurations

JFFS2	YAFFS2	LogFS	UBIFS
Compression	Default	Default	Compression

Benchmark result – I/O performance (cnt'd)

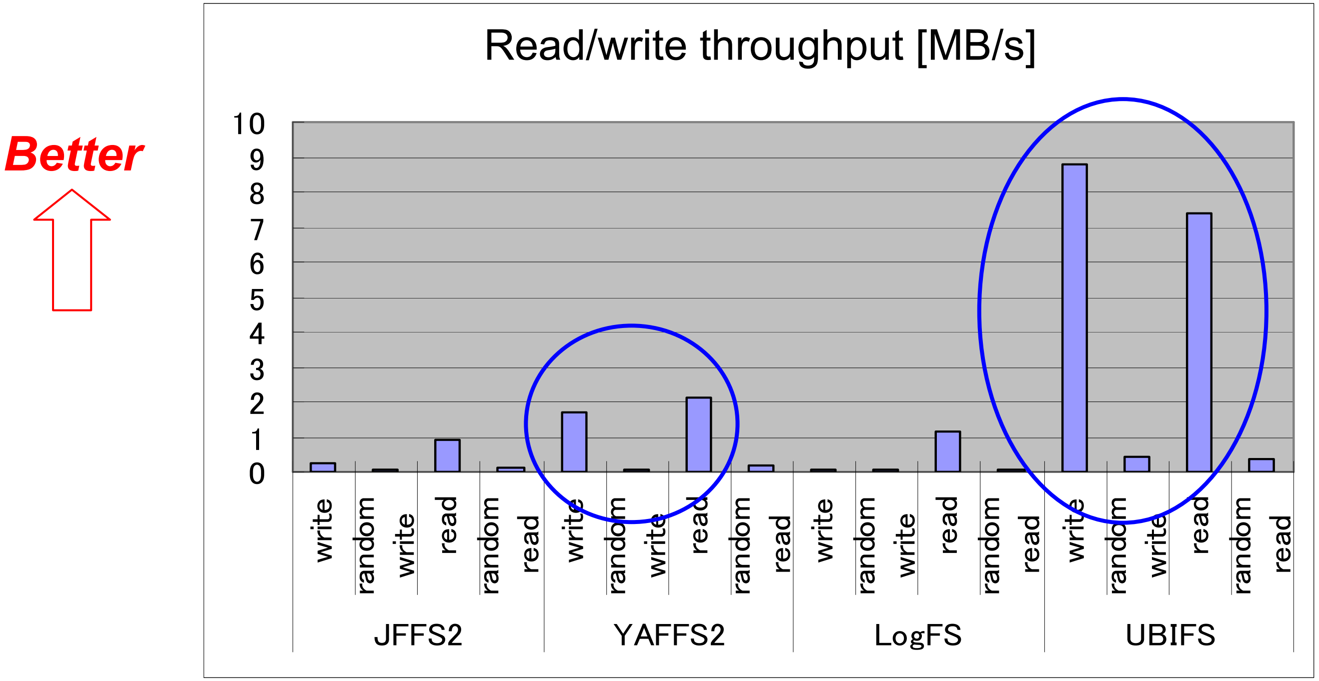
(b) Tiobench – Read/write throughput w/ 256B block size

Setting I/O block size to a half of NAND sub-page.

The throughput is lower in general.

UBIFS is good for sequential read/write due to write-back caching support.

YAFFS2 is good for sequential read/write because of the local buffer.



configurations	JFFS2	YAFFS2	LogFS	UBIFS
	Compression	Default	Default	Compression

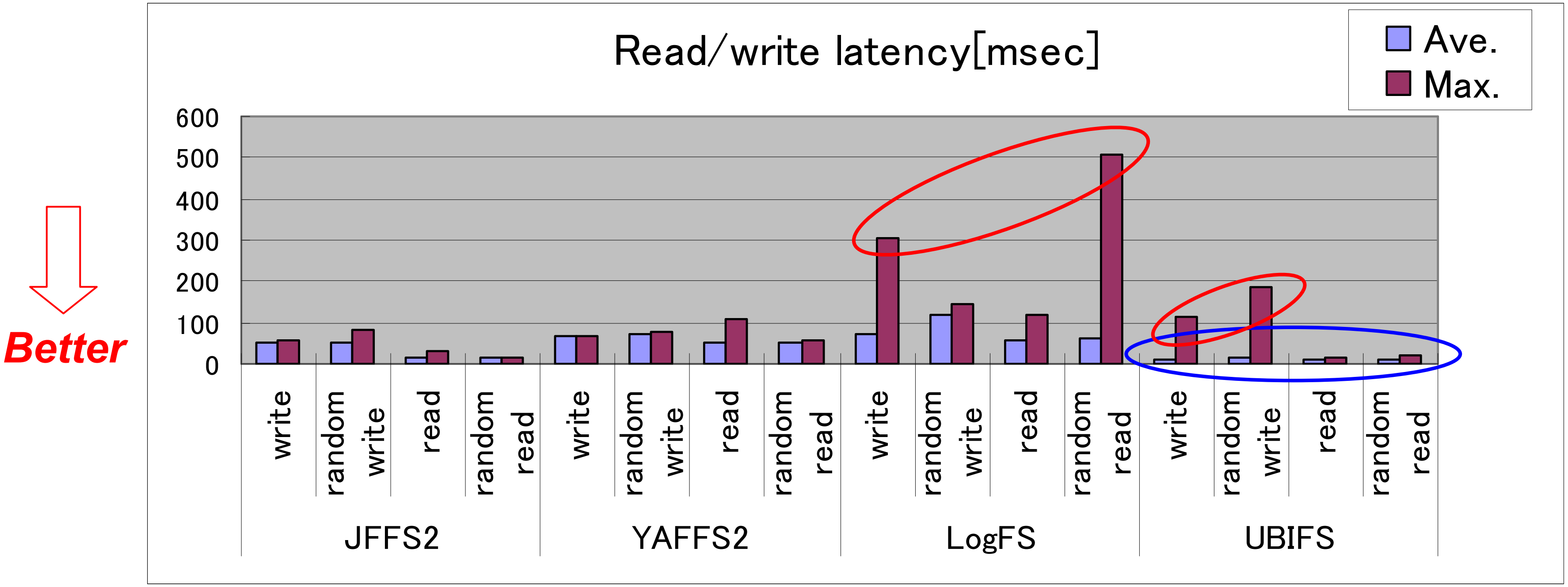
Benchmark result – I/O performance (cnt'd)

(b) Tiobench – Read/write latency w/ 128KB block size

UBIFS has the lowest latency for average case.

UBIFS has high latency for max case because of flushing cached data.

LogFS has the highest latency for max case because of error.



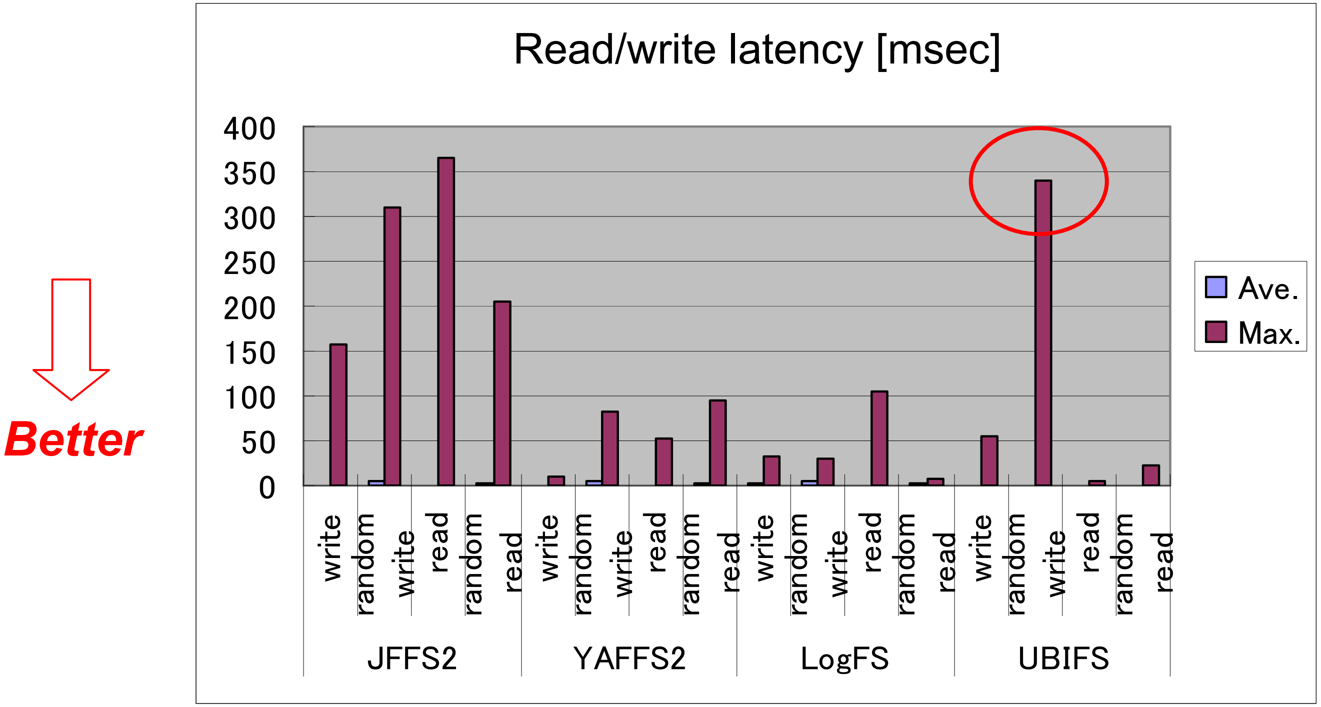
configurations	JFFS2	YAFFS2	LogFS	UBIFS
	Compression	Default	Default	Compression

Benchmark result – I/O performance (cnt'd)

(b) Tiobench – Read/write latency w/ 256B block size

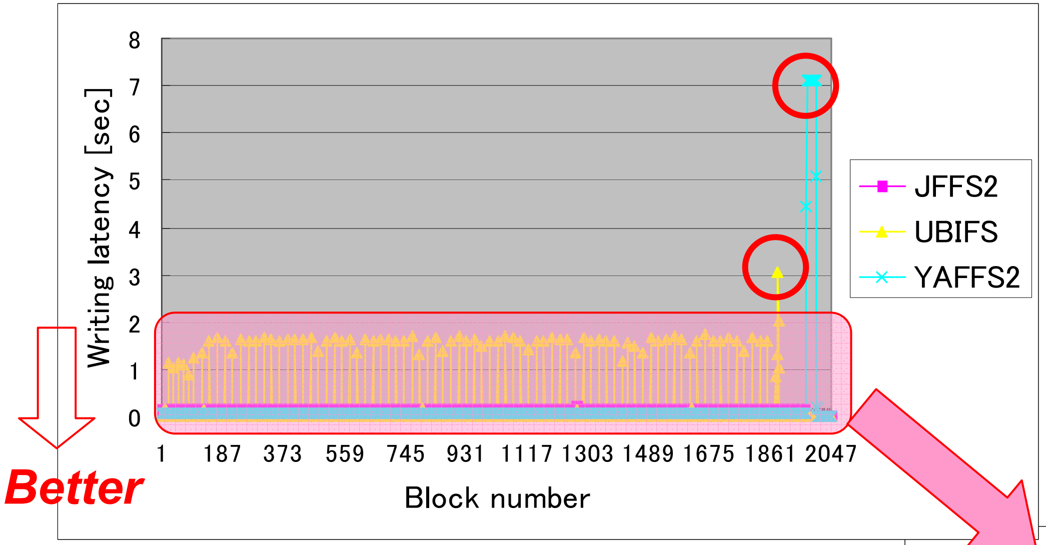
Moving PEB before writing needs more time

in case the writing block is smaller than sub-page.



Benchmark result – I/O performance

(b) Write latency in terms of time and left space

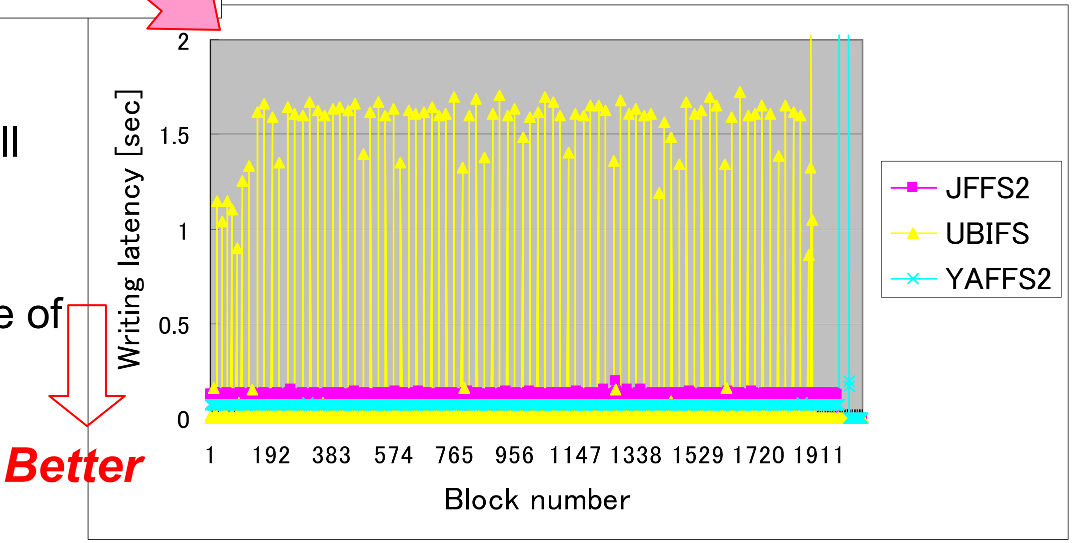


Writing 128KB data up to the capacity limit.

- **YAFFS2**and **UBIFS** have peaks of write latency when the left space becomes less.
- One of the reasons is the garbage collection.

- **UBIFS** supports write-back, thus the cached data has to be flushed. This will cause some latency periodically.

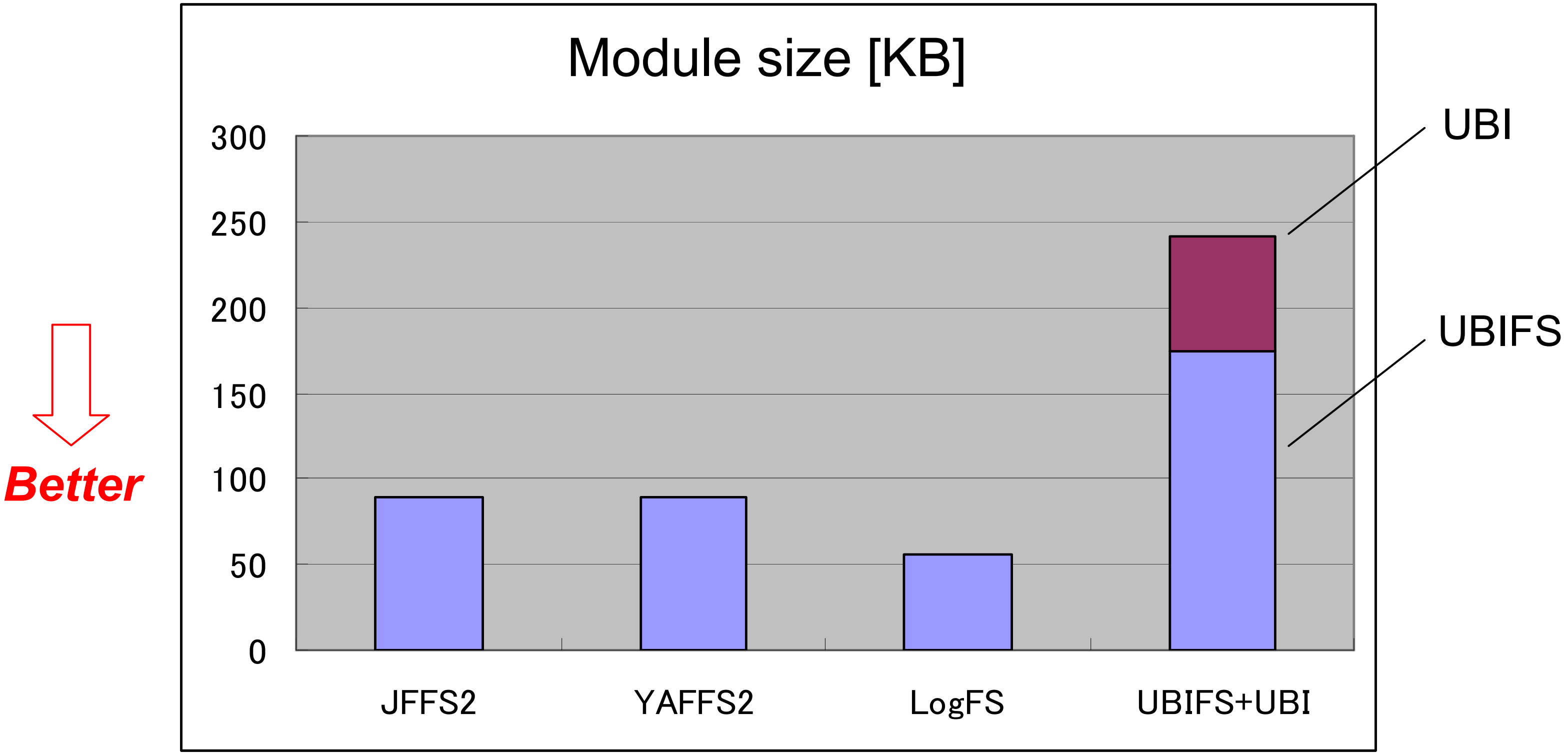
- **LogFS** could not be measured because of error.



Benchmark result – Memory consumption

(c) Module size

UBIFS plus UBI is the largest – 250KB.
LogFS is the smallest – 50KB.
This difference is not a big deal for some systems.



Benchmark result – Memory consumption

(d) RAM consumption

Measuring the RAM consumption in terms of the following cases:

- 3 patterns of the file size
  (0, 1MB, 10MB)
- 3 patterns of the number of files
  (0, 1024 of 1KB files (1MB), 10240 of 1KB files (10MB))

Conditions:

JFFS2	YAFFS2	LogFS	UBIFS
No compression	Default	Default	No compression

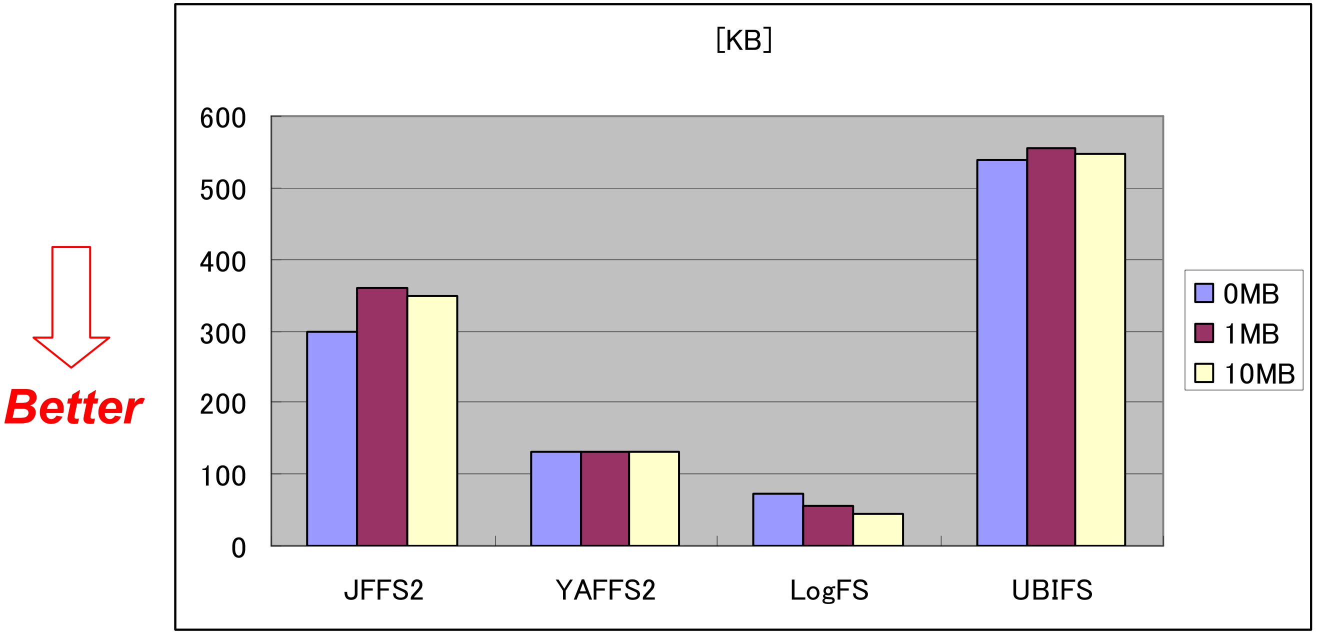
Benchmark result – Memory consumption

(d) RAM consumption

Measuring the RAM consumption in terms of the following cases:

- 3 patterns of the file size
  (0, 1MB, 10MB)

RAM consumption does not depend on the file size.
UBIFS > JFFS2 > YAFFS2 > LogFS



Benchmark result – Memory consumption

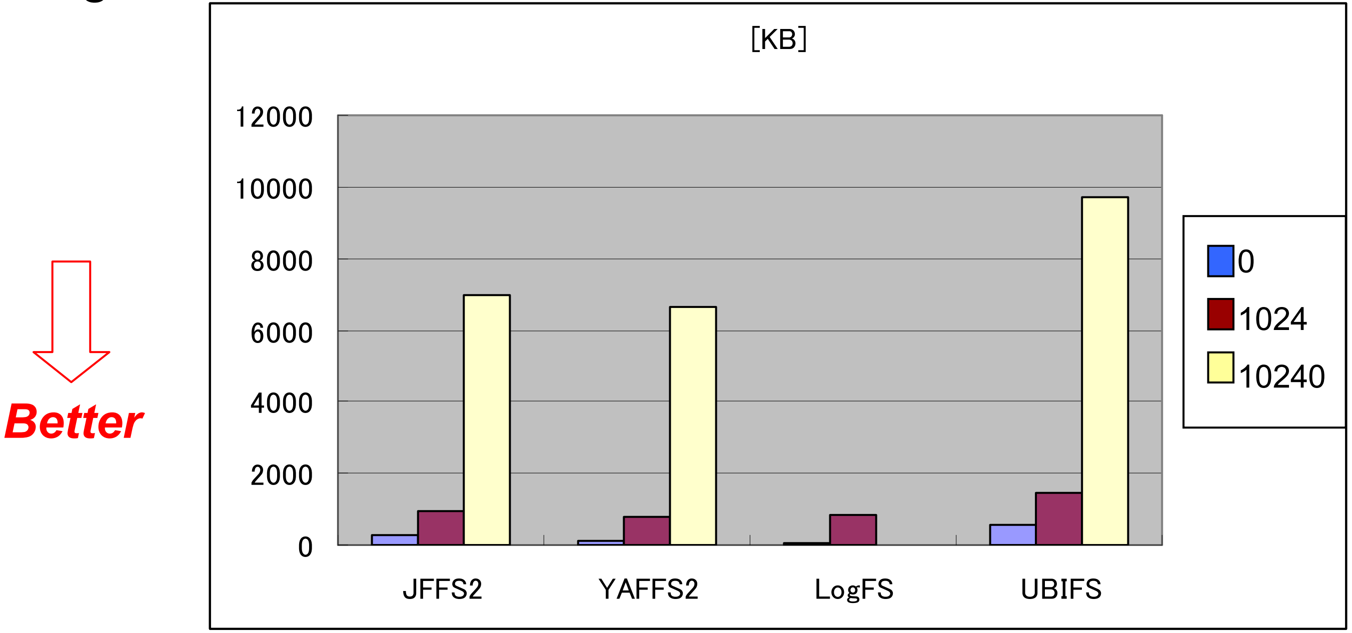
(d) RAM consumption

Measured the RAM consumption in terms of the following cases:

- 3 patterns of the <u>number of files</u>

(0, 1024 of 1KB files (1MB), 10240 of 1KB files (10MB))

RAM consumption increases linearly in terms of the number of files.
Memory usage per one file : UBIFS > YAFFS2 > JFFS2
LogFS could not be measured due to the error.



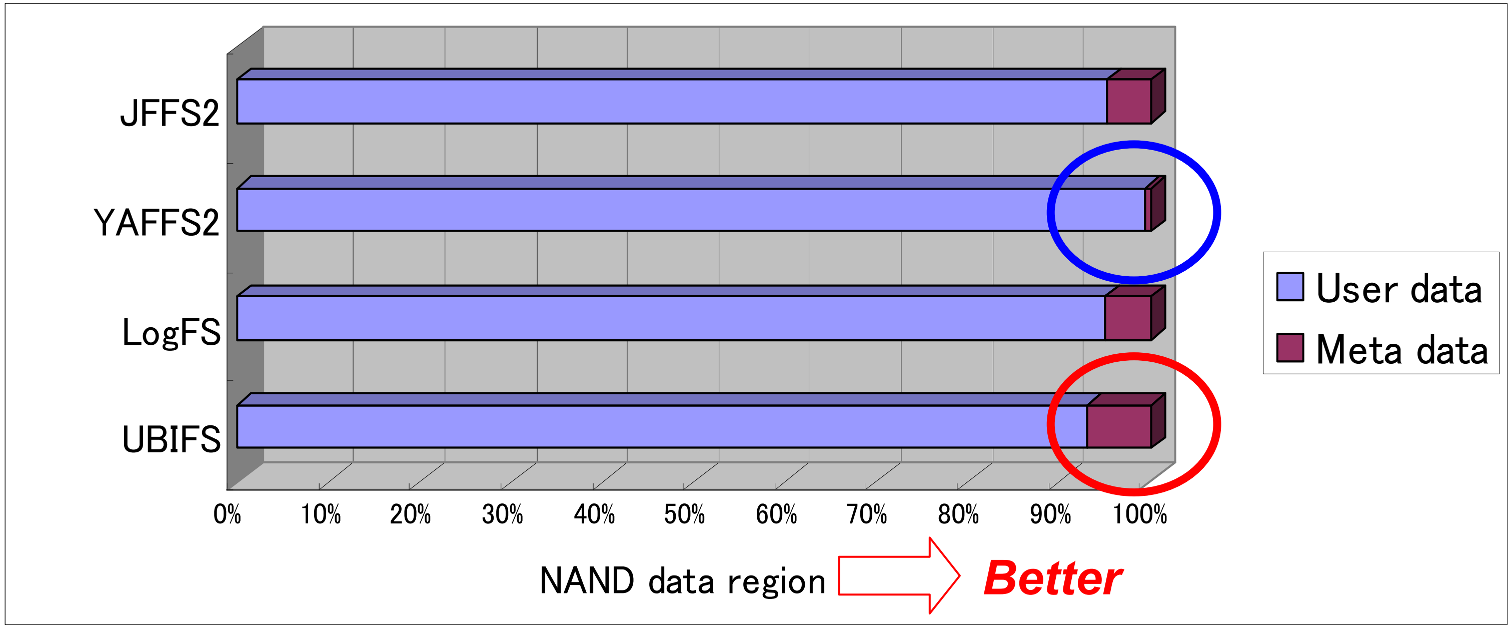
Benchmark result – Memory consumption

(e) Actual storage capacity

Writing a single file to see how much data could be written.

YAFFS2 can have the largest user data region.

UBIFS needs the most meta data region.



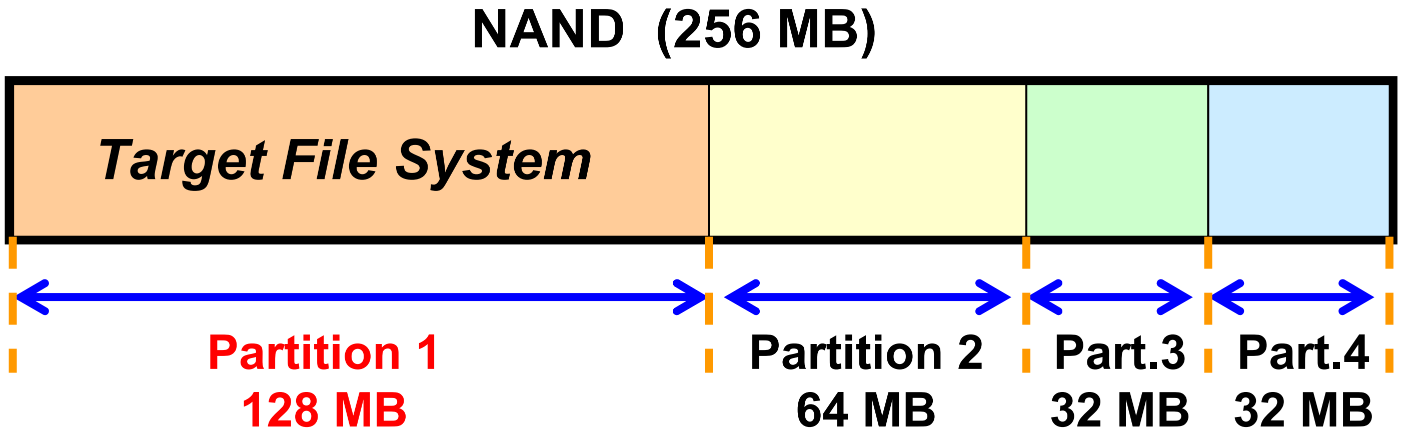
JFFS2	YAFFS2	LogFS	UBIFS
No compression	Default	Default	No compression

Benchmark result – NAND chip life expectancy

(f) Wear-leveling

Testing scenario :

- No compress options for JFFS2 and UBIFS.
- Partition 1 (128MB) is used for the given file system.
- Read-only data is stored in partition 1.
- Test tool to write 50MB data and erase it continuously.
- Counting how many each PEB was erased.



LogFS could not be tested because of error.

Benchmark result – NAND chip life expectancy

– Changed source code in JFFS2 for wear leveling test

[fs/jffs2/erase.c]

```
static void jffs2_erase_succeeded(struct jffs2_sb_info *c, struct jffs2_eraseblock *jeb)
{
                D1(printk(KERN_DEBUG "Erase completed successfully at 0x%08x¥n", jeb->offset));

+ #ifdef JFFS2_DEBUG_WL_COUNT
+ {
+               unsigned int eraseblock_number = (unsigned int)(jeb->offset/JFFS2_DEBUG_WL_EB_SIZE);
+               jffs2_wl_log.erase_count[eraseblock_number]++;
+ }
+ #endif

                mutex_lock(&c->erase_free_sem);
                spin_lock(&c->erase_completion_lock);
                list_move_tail(&jeb->list, &c->erase_complete_list);
                spin_unlock(&c->erase_completion_lock);
                mutex_unlock(&c->erase_free_sem);
                /* Ensure that kupdated calls us again to mark them clean */
                jffs2_erase_pending_trigger(c);
}
```

Benchmark result – NAND chip life expectancy

– **Changed source code in <u>YAFFS2</u> for wear leveling test**

[fs/yaffs2/yaffs_mtdif.c]

```
int nandmtd_EraseBlockInNAND(yaffs_Device * dev, int blockNumber)
{
        struct mtd_info *mtd = (struct mtd_info *)(dev->genericDevice);
        __u32 addr =
            ((loff_t) blockNumber) * dev->nDataBytesPerChunk
                * dev->nChunksPerBlock;
        struct erase_info ei;
        int retval = 0;

 :
 :
        /* Todo finish off the ei if required */

        sema_init(&dev->sem, 0);

        retval = mtd->erase(mtd, &ei);

        if (retval == 0)
+ {
+ #ifdef YAFFS2_DEBUG_WL_COUNT
+       yaffs2_wl_log.erase_count[blockNumber]++;
+ #endif
                return YAFFS_OK;
+ }
        else
                return YAFFS_FAIL;
}
```

Benchmark result – NAND chip life expectancy

– **Changed source code in LogFS for wear leveling test.**

[fs/logfs/dev_mtd.c]

```
static int mtd_erase(struct super_block *sb, loff_t ofs, size_t len)
{
                struct mtd_inode *mi = logfs_super(sb)->s_mtd;
                struct mtd_info *mtd = mi->mtd;
                struct erase_info ei;
                DECLARE_COMPLETION_ONSTACK(complete);
                int ret;

                BUG_ON(len % mtd->erasesize);

                if (logfs_super(sb)->s_flags & LOGFS_SB_FLAG_RO)
                                return -EROFS;
                   :
                   :
                   :
                wait_for_completion(&complete);
                if (ei.state != MTD_ERASE_DONE)
                                return -EIO;

+ #ifdef LOGFS_DEBUG_WL_COUNT
+ {
+               u_int32_t eraseblock_number = ((u_int32_t)ofs / mtd->erasesize);
+               logfs_wl_log.erase_count[eraseblock_number]++;
+ }
+ #endif

                return 0;
}
```

Benchmark result – NAND chip life expectancy

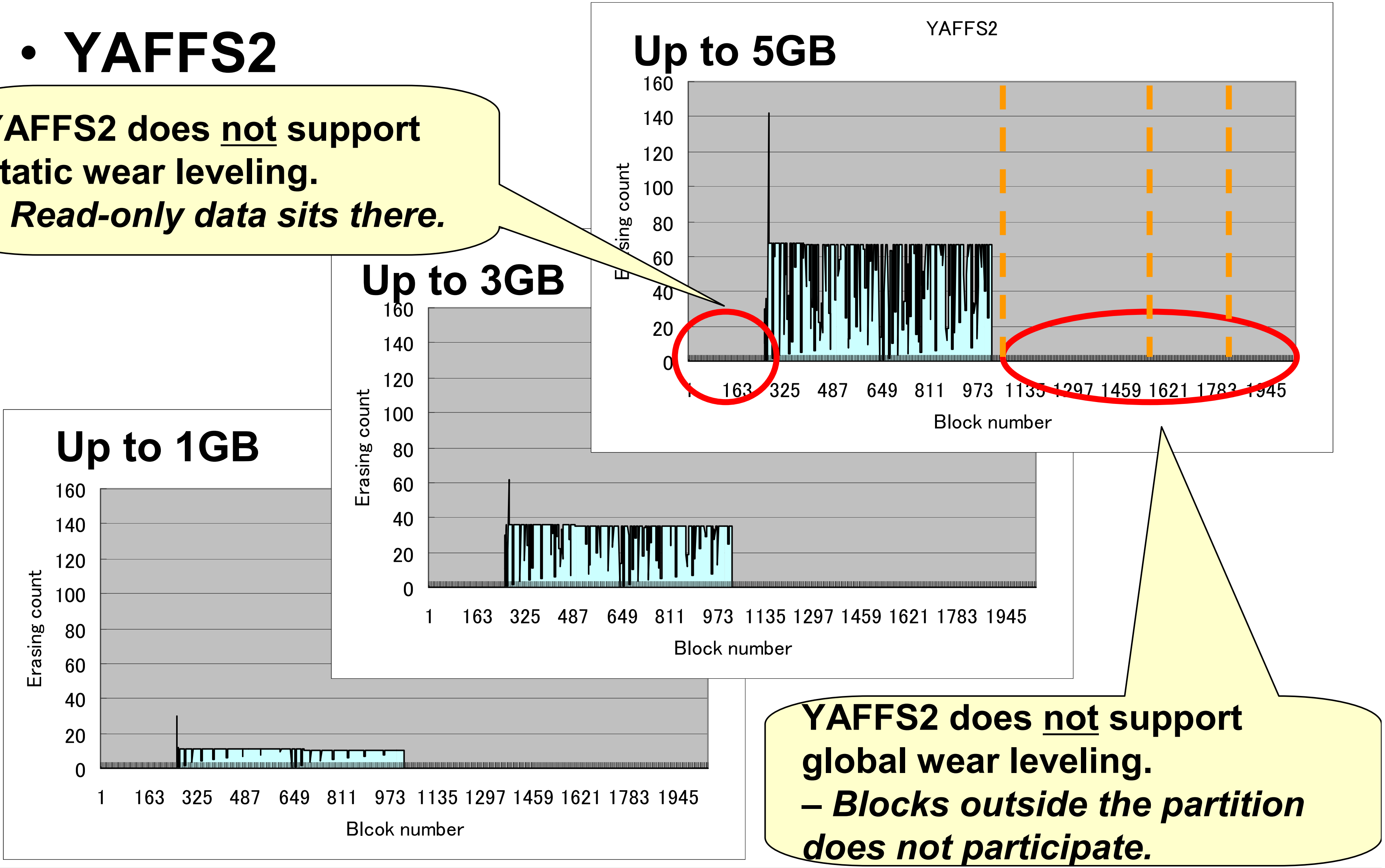
– **Changed source code in UBIFS for wear leveling test.**

[drivers/mtd/ubi/wl.c]

```
static int sync_erase(struct ubi_device *ubi, struct ubi_wl_entry *e,
                                    int torture)
{
                int err;
                struct ubi_ec_hdr *ec_hdr;
                unsigned long long ec = e->ec;
                     :
                     :
                ec += err;
                if (ec > UBI_MAX_ERASECOUNTER) {
                                /*
                                 * Erase counter overflow. Upgrade UBI and use 64-bit
                                 * erase counters internally.
                                 */
                                ubi_err("erase counter overflow at PEB %d, EC %llu",
                                                e->pnum, ec);
                                err = -EINVAL;
                                goto out_free;
                }

                dbg_wl("erased PEB %d, new EC %llu", e->pnum, ec);

+ #ifdef UBI_DEBUG_WL_COUNT
+               ubi_wl_log.erase_count[e->pnum]++;
+ #endif

                ec_hdr->ec = cpu_to_be64(ec);
                      :
                      :
```

Benchmark result – NAND chip life expectancy (cont'd)

- **YAFFS2**

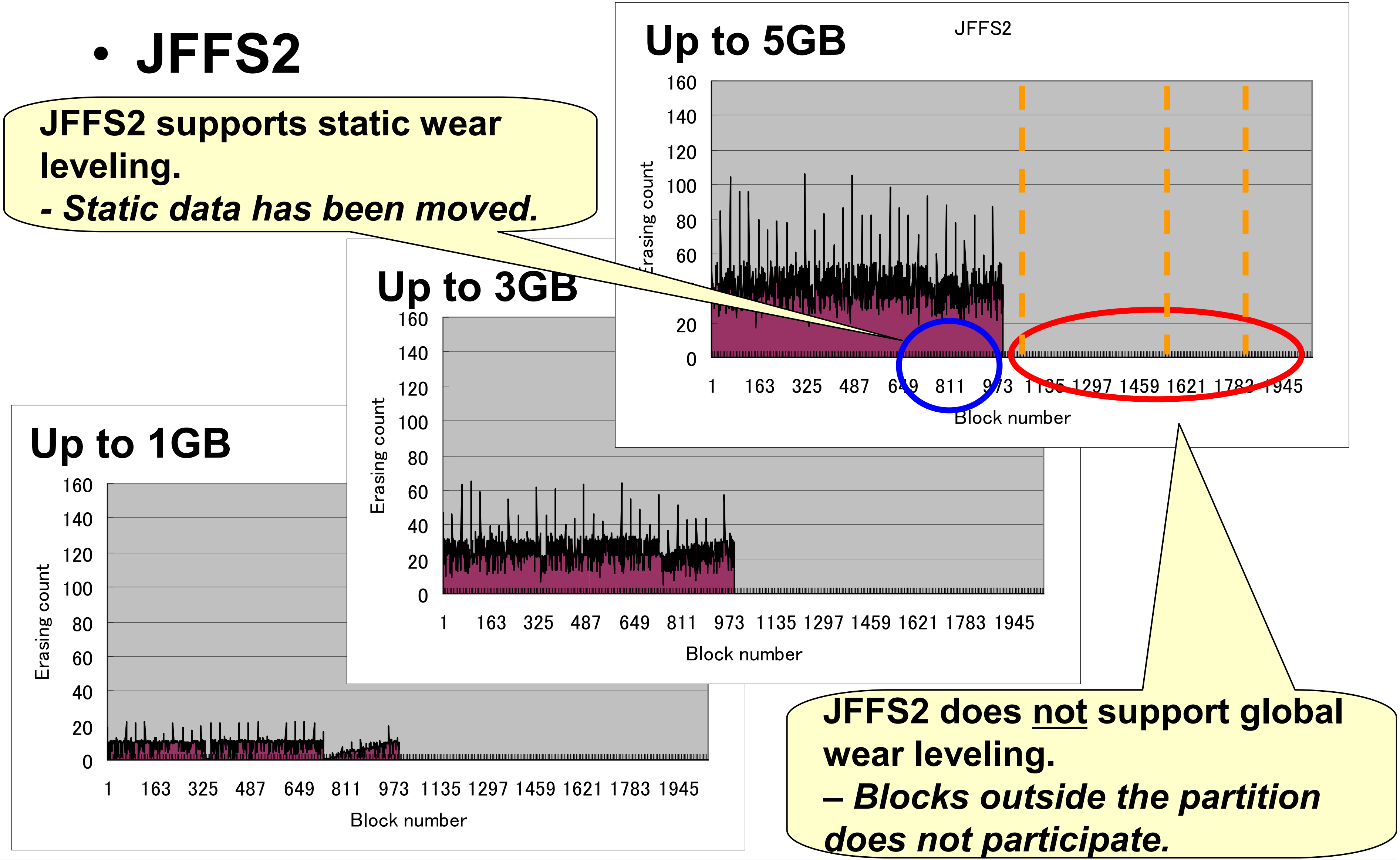


YAFFS2 does <u>not</u> support static wear leveling.
– Read-only data sits there.

YAFFS2 does <u>not</u> support global wear leveling.
– Blocks outside the partition does not participate.

Benchmark result – NAND chip life expectancy (cont’d)

- **JFFS2**



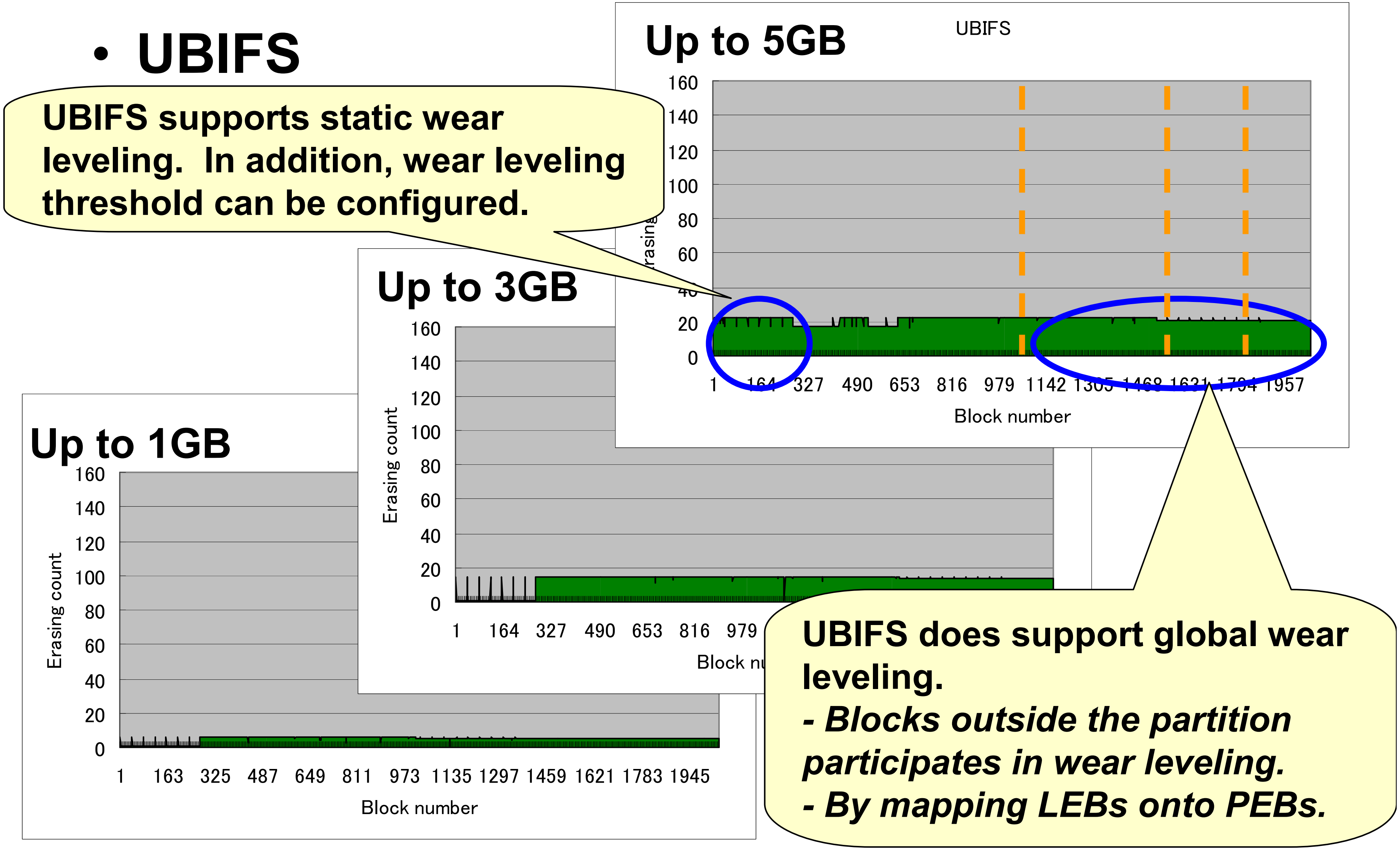
JFFS2 supports static wear leveling.
- Static data has been moved.

JFFS2 does <u>not</u> support global wear leveling.
– Blocks outside the partition does not participate.

Benchmark result – NAND chip life expectancy (cont'd)

- **UBIFS**

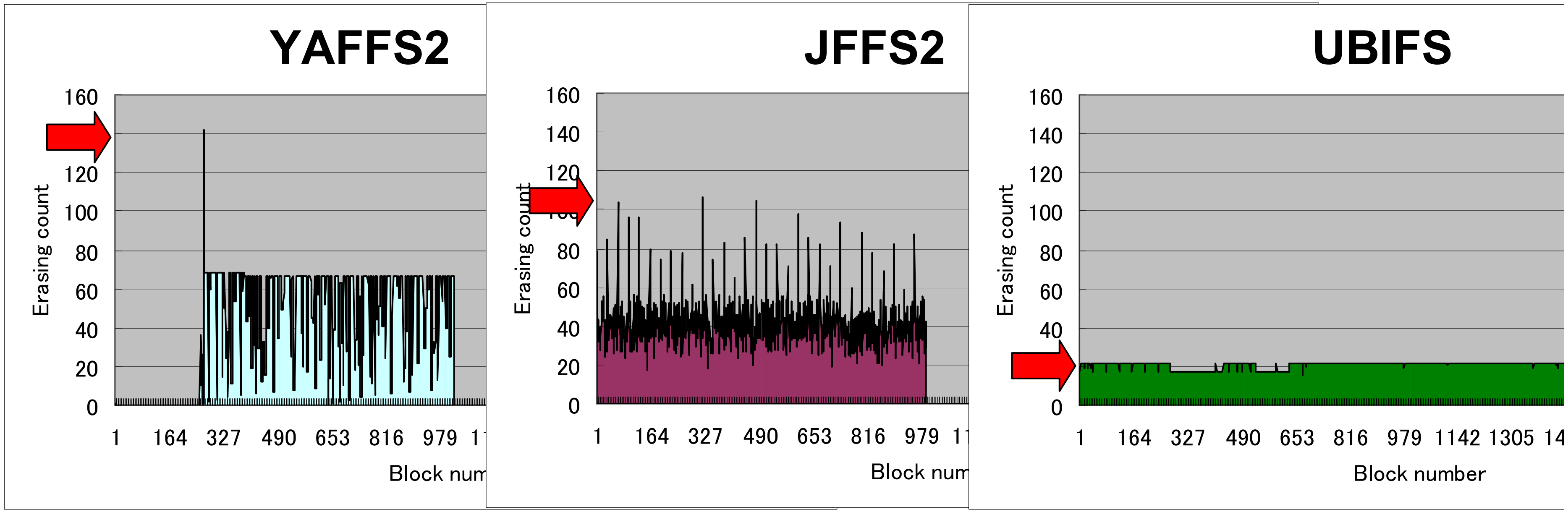
UBIFS supports static wear leveling. In addition, wear leveling threshold can be configured.



UBIFS does support global wear leveling.
- Blocks outside the partition participates in wear leveling.
- By mapping LEBs onto PEBs.

Benchmark result – NAND chip life expectancy (cont’d)

- **Wear leveling details – Erasing count per PEB**
  - UBIFS erasing count is distributed evenly in terms of the blocks.
  - JFFS2 varies more than the other file systems.



Benchmark result – Tolerance for unexpected power loss

(g) Recoverability for unexpected power loss

Counting mounting failure after unexpected power loss during the NAND device access.

Configurations:

JFFS2	YAFFS2	LogFS	UBIFS
Compression	Default	Default	Compression

LogFS failed about 20% of trials.
Needs more testing than 100 times trial.

Mounting failure after power off during the NAND device access

JFFS2	UBIFS	YAFFS2	LogFS
0	0	0	20

(100 times trial)

Summary – Criteria for large NAND flash integrated systems

- **UBIFS and YAFFS2 are good in general.**
- **UBIFS is in the mainline, which makes the maintenance cost lower.**
- **LogFS is under development and needs more work.**

	System requirement	JFFS2	YAFFS2	LogFS	UBIFS
1	Boot time	Poor	Good	Excellent	Good
2	I/O performance	Good	Good	Fair	Excellent
3	Resource usage	Fair	Excellent	Good	Fair
4	NAND device life expectancy	Good	Fair	N/A	Excellent
5	Tolerance for unexpected power-off	Good	Good	Poor	Good
6	Integrated in mainline	Yes	No	No	Yes

Summary – System models to fit each file system

- **System requirements for each file system**
  - Appropriate type of system
  - Improvements that will adapt your system to a particular file system

JFFS2	**Not dedicated to fast boot.** **To make small partitions.**
YAFFS2	**Little room for RAM or flash devices.** **To not write data often. To make the static data less. To make applications to handle static wear leveling.**
LogFS	**Dedicated to fast boot.** **Not dedicated to high I/O performance.**
UBIFS	**Having applications to write frequently on lifetime sensitive flash memories (e.g. MLC).** **Dedicated to high I/O performance.** **To have more room for RAM and flash.** **To not write data continuously until the cache overflow.**

MLC: Multi Level Cell

Summary

- **NAND flash device capacity is getting larger in consumer products.**
- **Showing which file system is to fit which system.**
- **Showing how to adapt your system to a particular file system.**
- **Improvement possibilities :**
  - YAFFS2 : to support static wear leveling.
  - LogFS : to make it more stable in case of large NAND.
  - UBIFS : to arrange the flushing of data to control write latency.

References

- **This presentation is based on**
  Shinji Namihira (Toshiba), “Examination of Linux Flash Filesystems for large NAND”, the 71st National Convention of IPSJ, 2009
- **MTD, JFFS2, UBIFS, UBI**
  http://www.linux-mtd.infradead.org/
- **YAFFS2**
  http://www.yaffs.net/
- **LogFS**
  http://www.logfs.com/logfs/
- **CE Linux Forum presentations**
  - Yutaka Araki, “Flash File system, current development status”
    http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree20?action=AttachFile&do=view&target=celf_flashfs.pdf
  - Katsuki Uwatoko, “The comparison of Flash File system performance”
    http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree19?action=AttachFile&do=get&target=celf_flash2.pdf
  - Keijiro Yano, “JFFS2 / YAFFS“
    http://www.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree17?action=AttachFile&do=view&target=celf_flashfs.pdf

TOSHIBA

Leading Innovation >>>