Flash Friendly File System (F2FS) Overview

Leon Romanovsky
leon@leon.nu

www.leon.nu

November 17, 2012

Disclaimer

Everything in this lecture shall not, under any circumstances, hold any legal liability whatsoever. Any usage of the data and information in this document shall be solely on the responsibility of the user. This lecture is not given on behalf of any company or organization.

Introduction: Flash Memory

Definition

Flash memory is a non-volatile storage device that can be electrically erased and reprogrammed.

Challenges

- block-level access
- wear leveling
- read disturb
- bad blocks management
- garbage collection
- different physics
- different interfaces

Introduction: Flash Memory

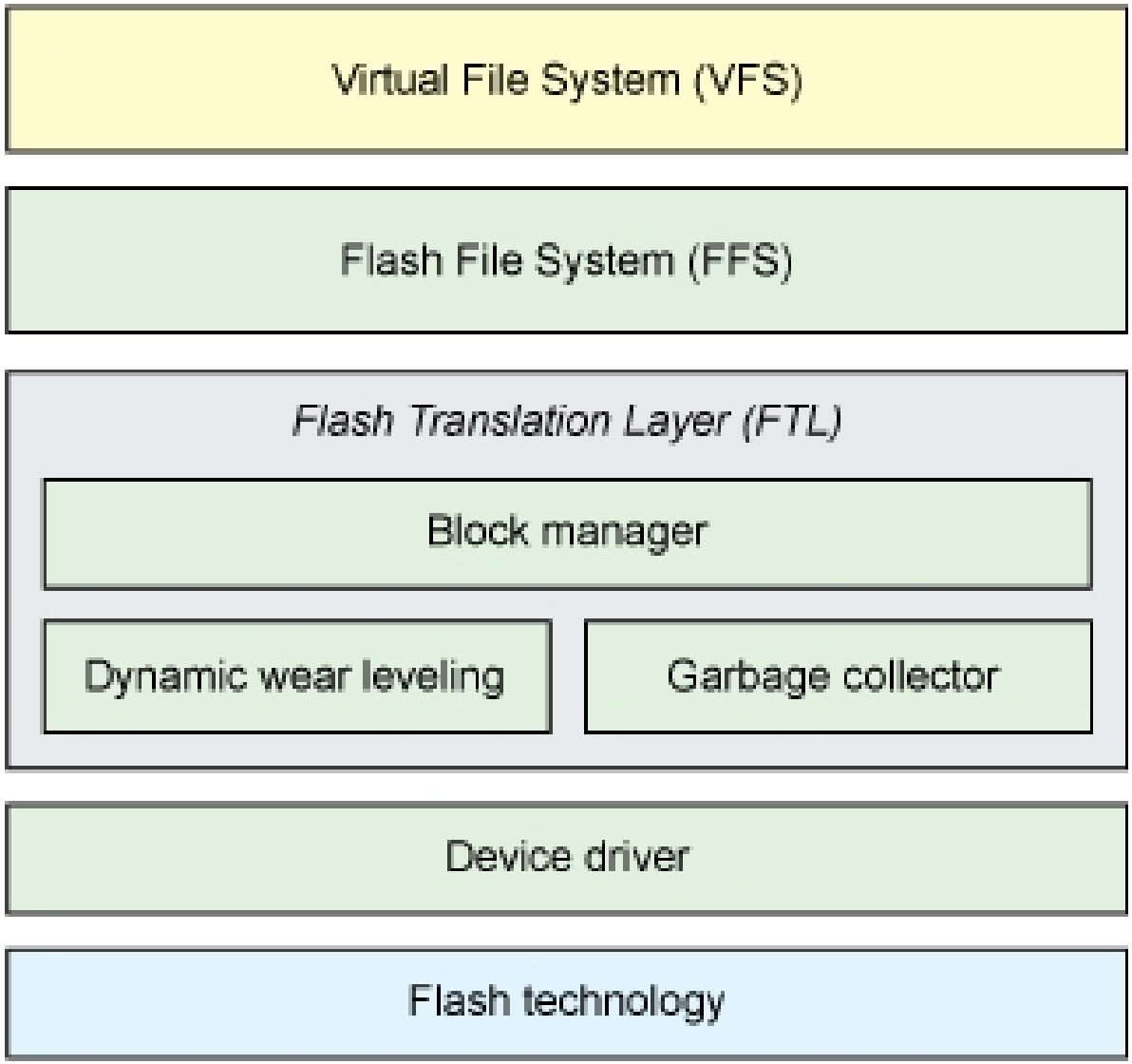
Definition

Flash memory is a non-volatile storage device that can be electrically erased and reprogrammed.

Challenges

- block-level access
- wear leveling
- read disturb
- bad blocks management
- garbage collection
- different physics
- different interfaces

Introduction: General System Architecture



Introduction: File Systems

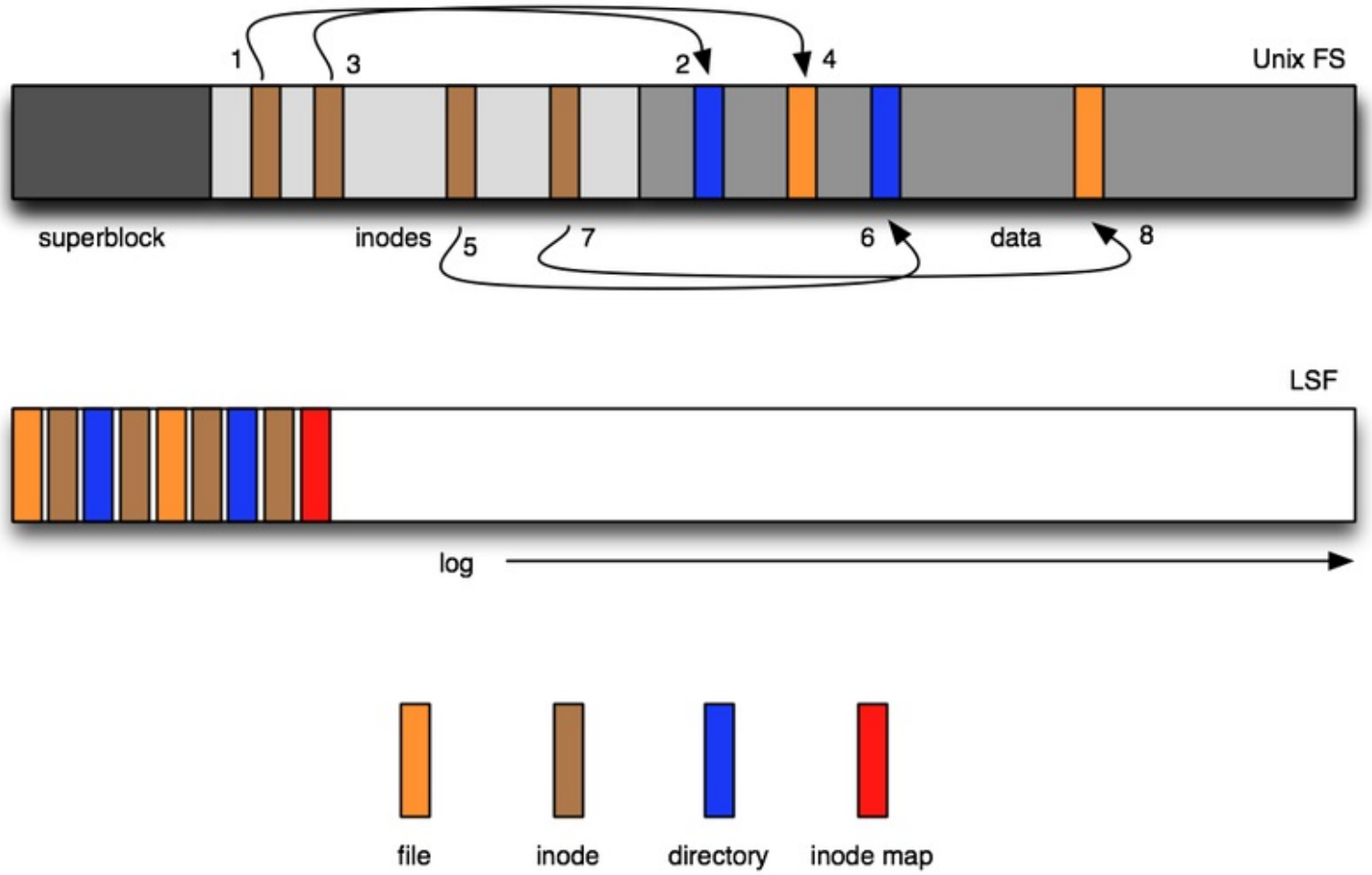
Optimized for disk storage

- EXT2/3/4
- BTRFS
- VFAT

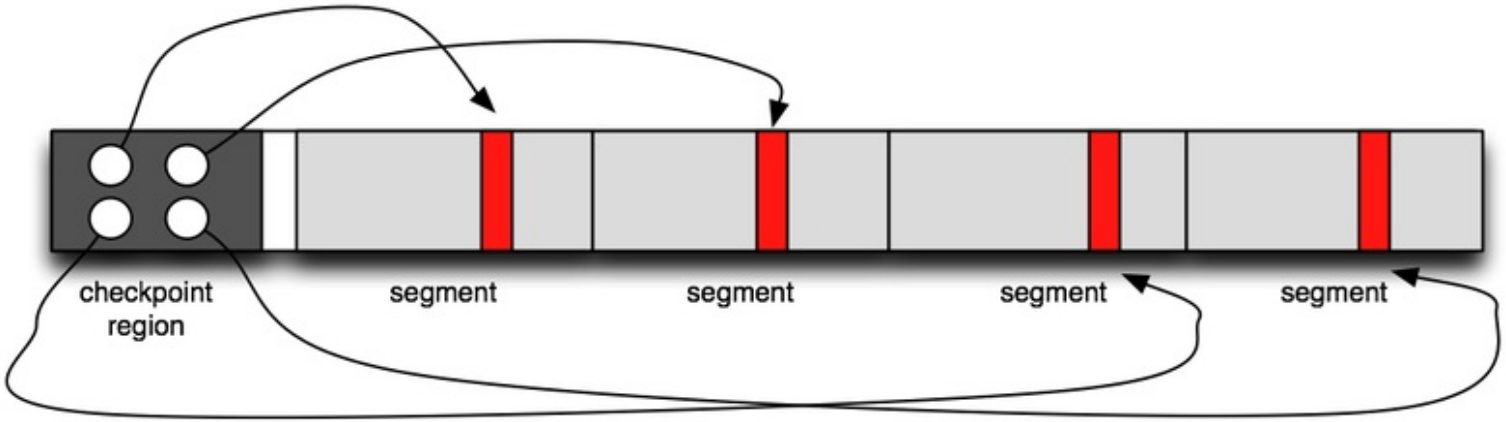
Optimized for flash, but not aware of FTL

- JFFS/JFFS2
- YAFFS
- LogFS
- UbiFS
- NILFS

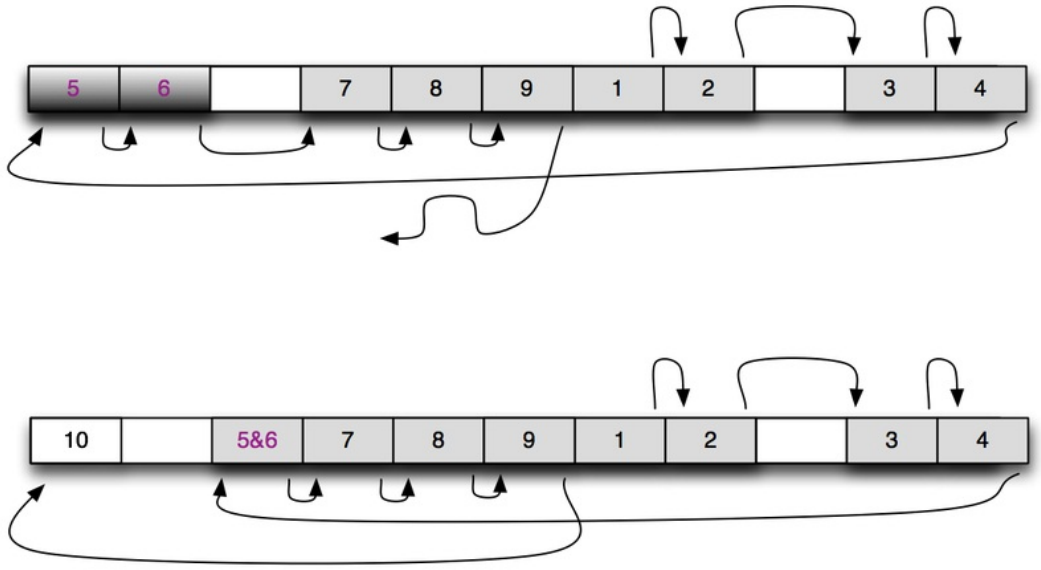
Background: LFS vs. Unix FS



Background: LFS Overview



Background: LFS Garbage Collection



segment cleaning (garbage collection)

1. A victim segment is selected through referencing segment usage table.
2. It loads parent index structures of all the data in the victim identified by segment summary blocks.
3. It checks the cross-reference between the data and its parent index structure.
4. It moves valid data selectively.

Background: LFS Summary

- maximized write throughput
- easy snapshotting
- easy recovery
- uneffective while storage has low empty space

f2fs: Introduction

- Assumes presence of FTL
  - no effort to distribute writes
  - random write area enlarged
  - grouping blocks with similiar life expectancies up to six in parallel
  - data structures aligned to the units of FTL
- Based on the log-structured design:
  - requires copy-on-write
  - free space is managed in large regions which are written to sequentially
- some metadata, and occasionaly some regular data is written via random single block writes
- support different garbage collection algorithms

f2fs: Introduction

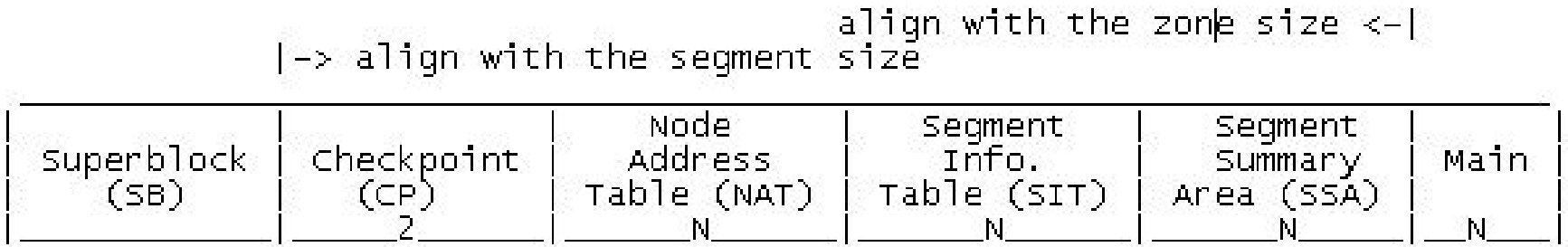
- Assumes presence of FTL
  - no effort to distribute writes
  - random write area enlarged
  - grouping blocks with similiar life expectancies up to six in parallel
  - data structures aligned to the units of FTL
- Based on the log-structured design:
  - requires copy-on-write
  - free space is managed in large regions which are written to sequentially
- some metadata, and occasionaly some regular data is written via random single block writes
- support different garbage collection algorithms

f2fs: Introduction

- Assumes presence of FTL
  - no effort to distribute writes
  - random write area enlarged
  - grouping blocks with similiar life expectancies up to six in parallel
  - data structures aligned to the units of FTL
- Based on the log-structured design:
  - requires copy-on-write
  - free space is managed in large regions which are written to sequentially
- some metadata, and occasionaly some regular data is written via random single block writes
- support different garbage collection algorithms

f2fs: On-disk Layout

- Block (4K in size)
- Segment (2MB in size)
- Section (consecutive segments)
- Zone (set of sections)
- Area (multiple sections)
- Volume (six areas)



f2fs: Name Conventions

Superblock (SB)

- located at the beginning of the partitions
- two copies exist
- basic partition information
- some default parameters of f2fs

Checkpoint (CP)

- file system information, bitmaps for valid NAT/SIT sets, orphan inode lists, and summary entries of current active segments.

Node Address Table (NAT)

- block address table for all the node blocks stored in Main area.

f2fs: Name Conventions

Segment Information Table (SIT)

- segment information such as valid block count and bitmap for the validity of all the blocks
- 74 bytes per entry (segment)
- keep track of active blocks

Segment Summary Area (SSA)

- summary entries which contains the owner information of all the data and node blocks stored in Main area

Main Area

- file and directory data including their indices

f2fs: Design

Blocks

- 4K in size
- 32 bits for address space ($2^{(32+12)}$ available blocks)
- limited to 16 terabytes

Segments

- 512 blocks
- 2MB in size
- each segment has segment summary block (file plus offset of each block in the segment)

f2fs: Design

Blocks

- 4K in size
- 32 bits for address space ($2^{(32+12)}$ available blocks)
- limited to 16 terabytes

Segments

- 512 blocks
- 2MB in size
- each segment has segment summary block (file plus offset of each block in the segment)

f2fs: Design

Sections

- flexible size (power of two)
- filled from start to end
- clean one section at a time
- default size is one segment per sector
- 6 sections “open” for writting
- allows hot/warm/cold data segregation

Zones

- any (integer) number of sections
- default one sector per zone
- intended to distribute “open” sections over different devices for parallel processing

f2fs: Design

Sections

- flexible size (power of two)
- filled from start to end
- clean one section at a time
- default size is one segment per sector
- 6 sections “open” for writting
- allows hot/warm/cold data segregation

Zones

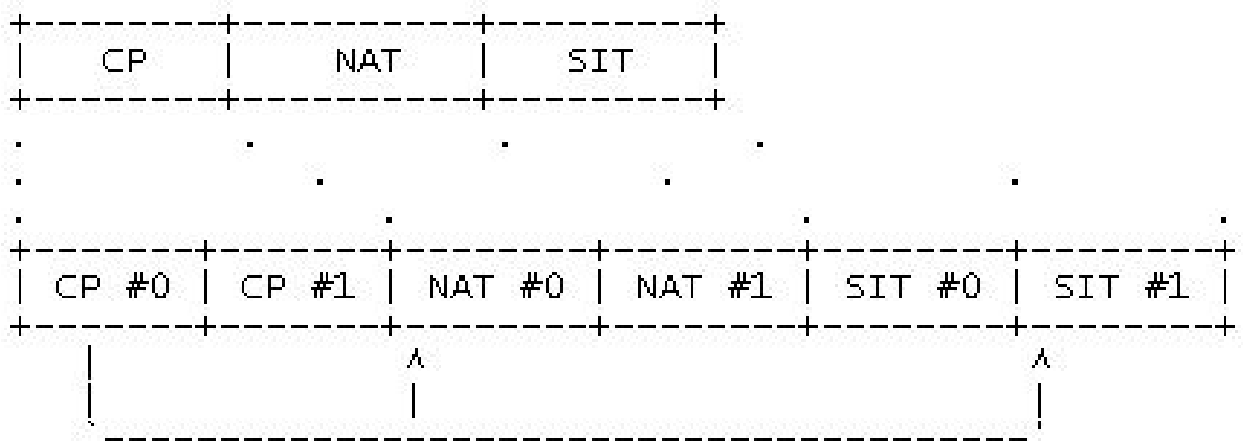
- any (integer) number of sections
- default one sector per zone
- intended to distribute “open” sections over different devices for parallel processing

f2fs: Design

- writes to NAT and SIT are cached till write to CP
- minimize random block updates
- packs random access to flash
- out of spaces causes to random writes
- read-only data (superblock) - never changes
- segment summary blocks - updated in-place
- double space allocation - primary/secondary location

f2fs: File System Metadata Structure

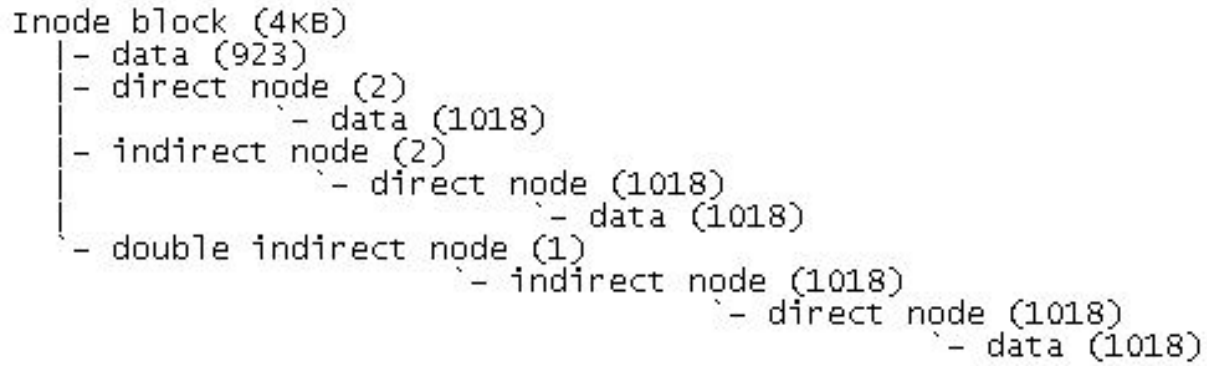
- shadow copy mechanism



f2fs: Index Structure

Nodes

- inodes
- direct node
- indirect node



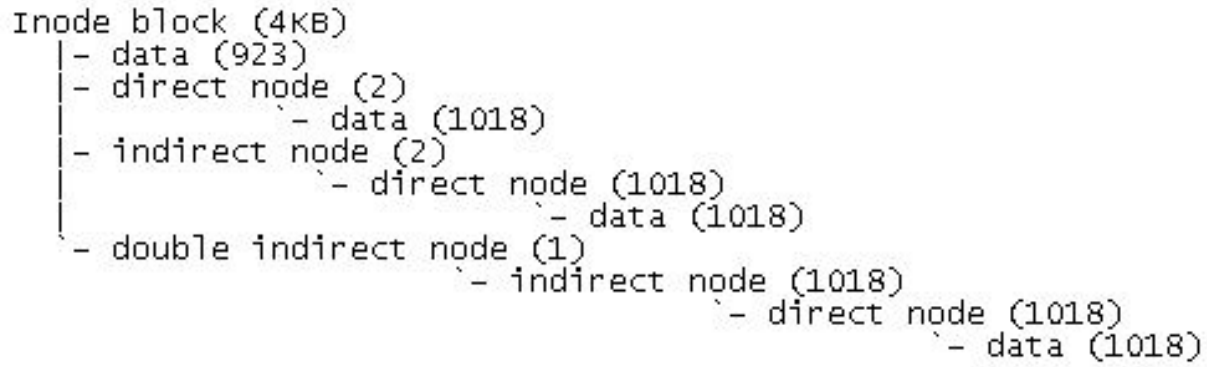
$$4KB*(927+2*1018+2*1018*1018+1018*1018*1018) := 3.94TB$$

- index tree for a given file has a fixed and known size
- block updates done via NAT (node address table)

f2fs: Index Structure

Nodes

- inodes
- direct node
- indirect node



$$4KB*(927+2*1018+2*1018*1018+1018*1018*1018) := 3.94TB$$

- index tree for a given file has a fixed and known size
- block updates done via NAT (node address table)

f2fs: Index Structure

Nodes

- inodes
- direct node
- indirect node

```
Inode block (4KB)
  |- data (923)
  |- direct node (2)
  |            `- data (1018)
  |- indirect node (2)
  |              `- direct node (1018)
  |                            `- data (1018)
  `- double indirect node (1)
                          `- indirect node (1018)
                                        `- direct node (1018)
                                                      `- data (1018)
```

$$4KB*(927+2*1018+2*1018*1018+1018*1018*1018) := 3.94TB$$

- index tree for a given file has a fixed and known size
- block updates done via NAT (node address table)

f2fs: Directory Structure

directory entry == 11 bytes

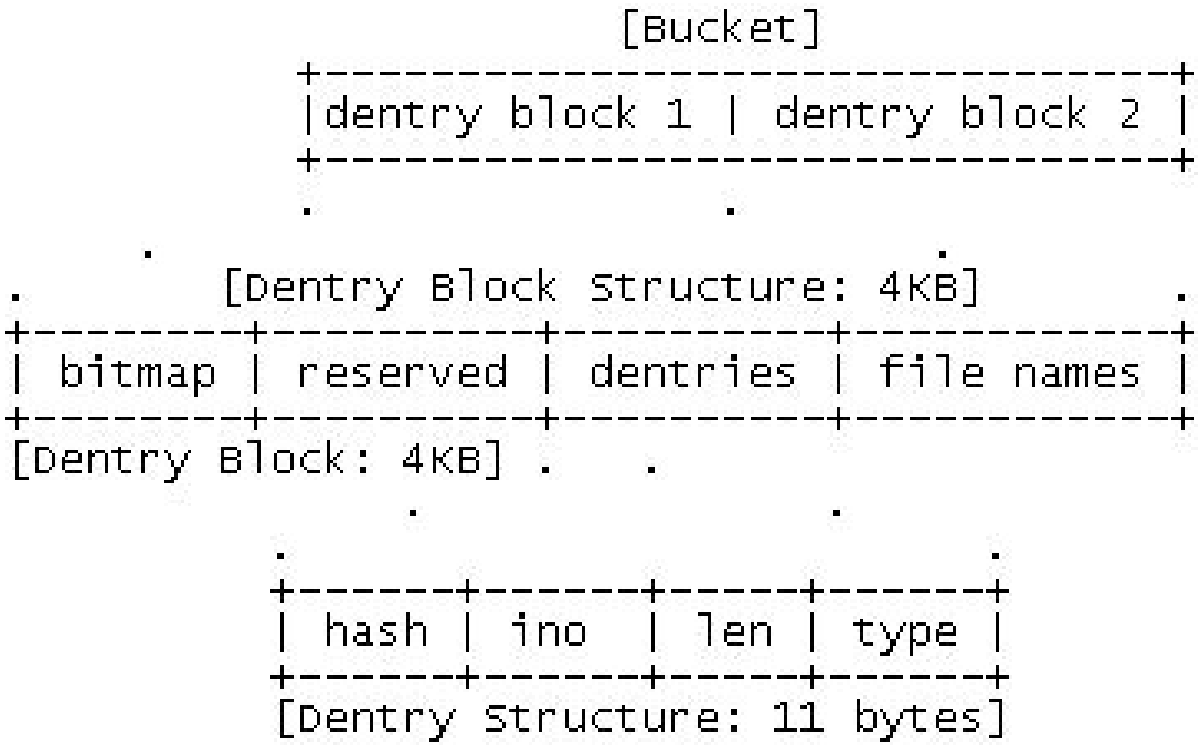
- hash - hash value of the file name
- ino - inode number
- len - the length of file name
- type - file type such as directory, symlink, etc

directory block

- 214 dentry slots
- dentry validity bitmap
- 4KB in size

f2fs: Directory Structure

$$DentryBlock(4K) = bitmap(27bytes) + reserved(3bytes) + dentries(11 * 214bytes) + filename(8 * 214bytes)$$



f2fs: File Lookup Complexity

The number of blocks and buckets are determined by,

$$\#\text{ of blocks in level \#n} = \begin{cases} 2, & \text{if n} < \text{MAX\_DIR\_HASH\_DEPTH} / 2, \\ 4, & \text{otherwise} \end{cases}$$

$$\#\text{ of buckets in level \#n} = \begin{cases} 2\text{\^{}n}, & \text{if n} < \text{MAX\_DIR\_HASH\_DEPTH} / 2, \\ 2\text{\^{}((MAX\_DIR\_HASH\_DEPTH} / 2) - 1), & \text{otherwise} \end{cases}$$

bucket number to scan in level #n = (hash value) % (# of buckets in level #n)

$\Longrightarrow$ O(log(# of files)) complexity

f2fs: Directory Structure Summary

- combination of hash and linear search
- global seed (possible vulnerable to hash collision attack)
- stable address for telldir()

f2fs: Default Block Allocation

Node/Data Type	Contains
hot node	direct node blocks of directories
warm node	direct node blocks except hot node blocks
cold node	indirect node blocks
hot data	dentry blocks
warm data	data blocks except hot and cold data blocks
cold data	multimedia data or migrated data blocks

f2fs: Free Space Management

Combination of two schemes, depends on file system status:

copy-and-compactions scheme

- good for sequential write performance
- suffers from cleaning overhead under high utilization

threaded log scheme

- no cleaning process is needed
- suffers from random write

f2fs: Cleaning process

on demand

triggered when there are not enough free segments to serve VFS calls

background(kernel thread)

triggerred the cleaning job when the system is idle

greedy policy (on-demand cleaner)

F2FS selects a victim segment having the smallest number of valid blocks

cost-benefit policy (background cleaner)

F2FS selects a victim segment according to the segment age and the number of valid blocks in order to address log block thrashing problem in the greedy algorithm

Victim segment list is managed in bit stream bitmap

f2fs: Cleaning process

on demand

triggered when there are not enough free segments to serve VFS calls

background(kernel thread)

triggerred the cleaning job when the system is idle

greedy policy (on-demand cleaner)

F2FS selects a victim segment having the smallest number of valid blocks

cost-benefit policy (background cleaner)

F2FS selects a victim segment according to the segment age and the number of valid blocks in order to address log block thrashing problem in the greedy algorithm

Victim segment list is managed in bit stream bitmap

f2fs: Cleaning process

on demand

triggered when there are not enough free segments to serve VFS calls

background(kernel thread)

triggerred the cleaning job when the system is idle

greedy policy (on-demand cleaner)

F2FS selects a victim segment having the smallest number of valid blocks

cost-benefit policy (background cleaner)

F2FS selects a victim segment according to the segment age and the number of valid blocks in order to address log block thrashing problem in the greedy algorithm

Victim segment list is managed in bit stream bitmap

f2fs: Linux Samsung Test

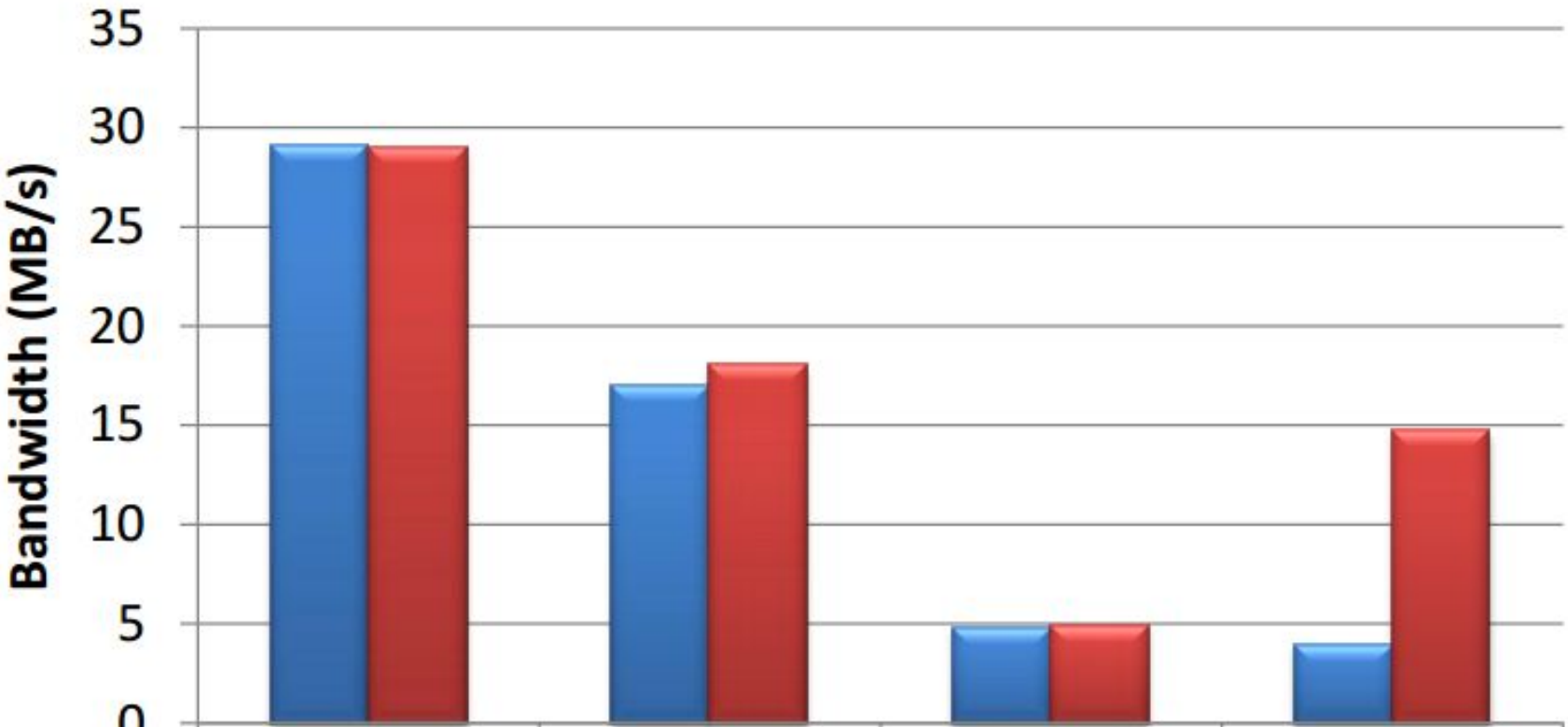
Panda board

- Kernel : Linaro 3.3
- DRAM : 1GB
- Partition : Samsung eMMC 4.5, 12GB

Test

- ./iozone -i0 -s 2g -r 4k -f $MNT -e -w -n
- ./iozone -i2 -s 2g -r 4k -f $MNT -e -w -R

f2fs: Linux Samsung Test



	seq. Read	seq. Write	rand. Read	rand. Write
EXT4	29.058	17.018	4.786	3.943
F2FS	29.029	18	4.95	14.668

f2fs: Android Samsung Test

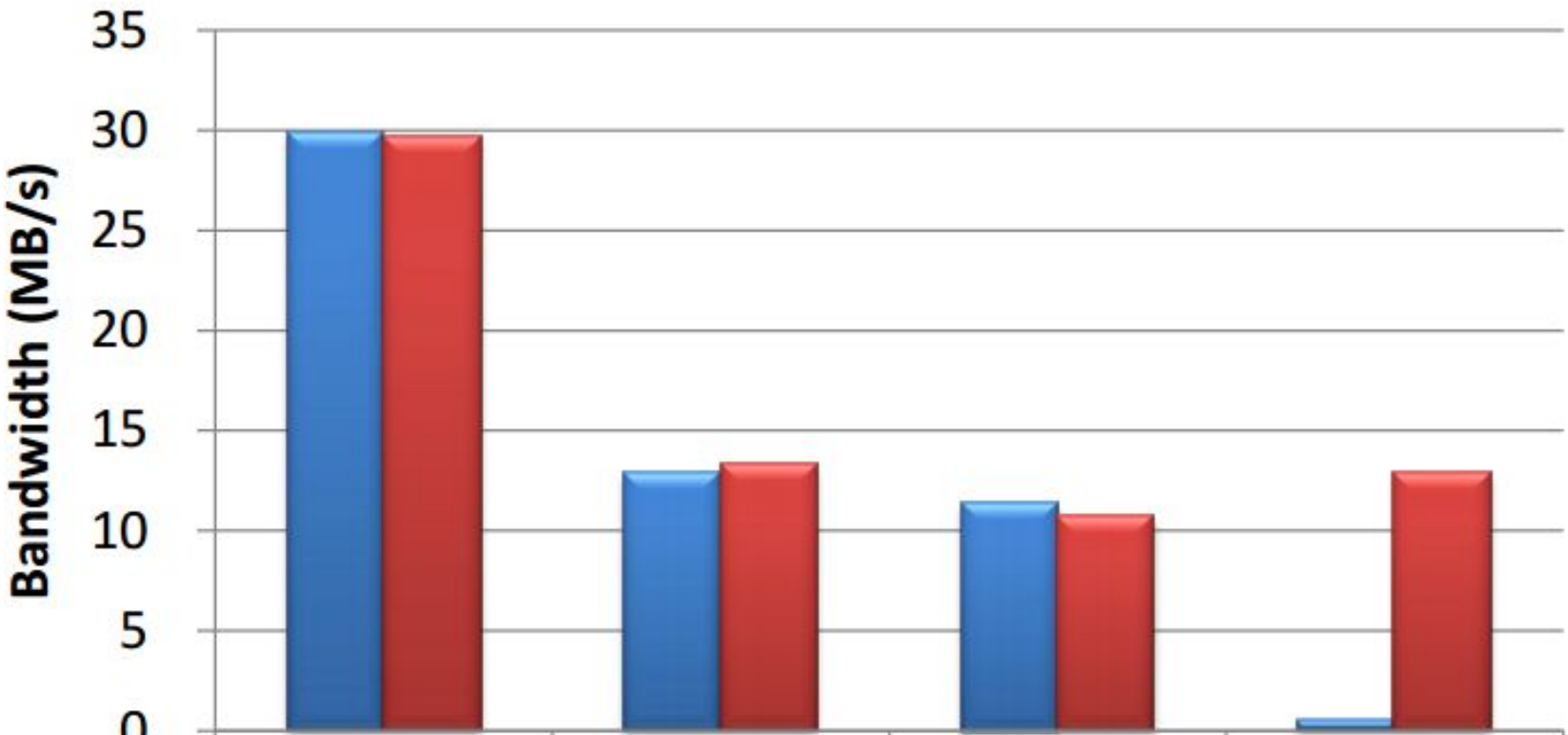
Galaxy Nexus

- Android : 4.0.4_r1.2
- Kernel : OMAP 3.0.8
- DRAM : 1GB
- Partition : /data, 12GB

Test

- ./iozone -i0 -i1 -i2 -s 512m -r 4k -f $MNT

f2fs: Android Samsung Test



	seq. Read	seq. Write	rand. Read	rand. Write
EXT4	29.88	12.83	11.43	0.56
F2FS	29.70	13.34	10.79	12.82

Timeline

- 2 Jun. 2012 - Storage Summit at Linaro Connect Q2.12
- 5 Oct 2012 - [PATCH v1] introduce flash-friendly file system
- 23 Oct. 2012 - [PATCH v2] introduce flash-friendly file system
- 29 Oct. 2012 - f2fs review at Linaro Connect Q3.12
- 31 Oct. 2012 - [PATCH v3] introduce flash-friendly file system
- Q3 2013 - Expected at the "real world"

Links

- Rosenblum, M. and Ousterhout, J. K., 1992, "The design and implementation of a log-structured file system", ACM Trans. Computer Systems 10, 1, 2652.
- Wikipedia: http://en.wikipedia.org/wiki/Flash_memory
- Anatomy of Linux flash file systems: http://www.ibm.com/developerworks/linux/library/l-flash-filesystems/
- Next-generation Linux file systems: NiLFS(2) and exofs http://www.ibm.com/developerworks/linux/library/l-nilfs-exofs/
- Log structured file system for dummies http://work.tinou.com/2012/03/log-structured-file-system-for-dummies.html
- Flash memory card design https://wiki.linaro.org/WorkingGroups/Kernel/Projects/FlashCardSurvey

Links

- LKML: [PATCH v1] introduce flash-friendly file system
  `https://lkml.org/lkml/2012/10/5/205`
- LKML: [PATCH v2] introduce flash-friendly file system
  `https://lkml.org/lkml/2012/10/22/664`
- LKML: [PATCH v3] introduce flash-friendly file system
  `https://lkml.org/lkml/2012/10/31/156`
- LKML: initial report on F2FS filesystem performance
  `https://lkml.org/lkml/2012/10/16/3`
- LWN: An f2fs teardown
  `http://lwn.net/Articles/518988/`