Flash-Friendly File System (F2FS)

Presented at KLF 2012
October 12, 2012
Jaegeuk Kim
S/W Development Team
Memory Division, Samsung Electronics Co., Ltd.

SAMSUNG
TURN ON TOMORROW

Align with your imagination

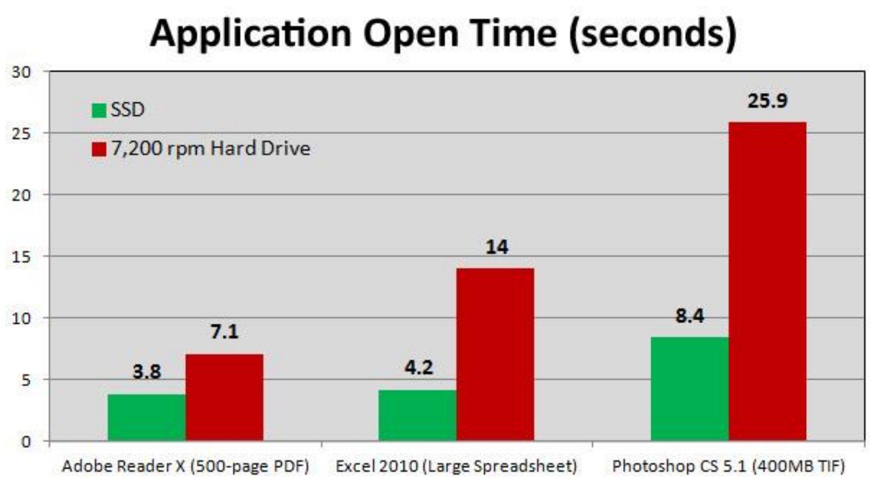
Agenda

- Introduction
- Log-structured File System
- Design Issues
- Design of F2FS
- Performance Evaluation
- Summary

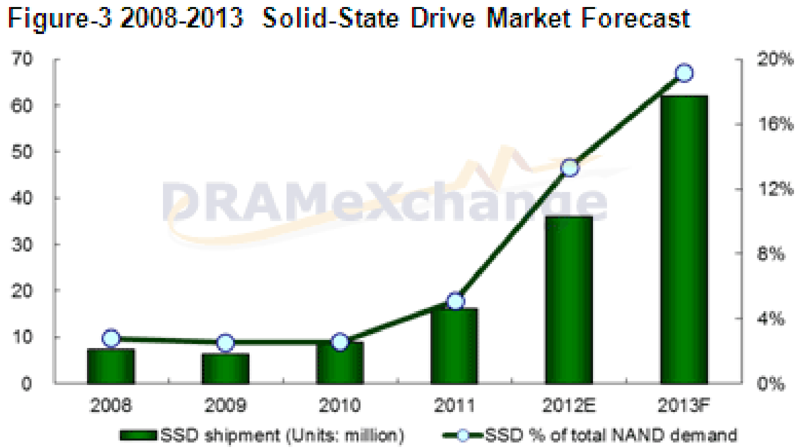
Introduction

- NAND Flash-based Storage Devices
  - SSD for PC and server systems
  - eMMC for mobile systems
  - SD card for consumer electronics

- The Rise of SSDs
  - Much faster than HDDs
  - Low power consumption



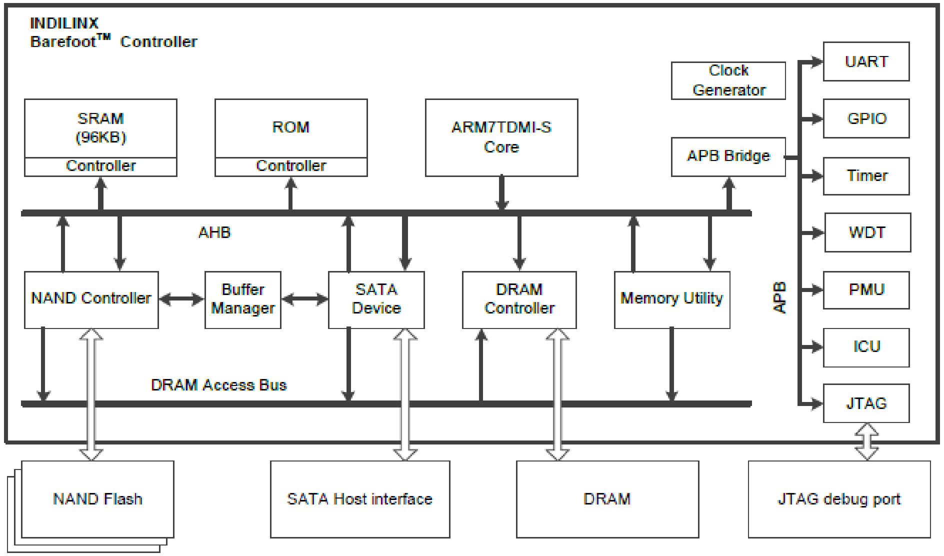
Source: March 30th, 2012 by Avram Piltch, LAPTOP Online Editorial Director



Source: DRAMeXchange, Jan., 2012

Introduction

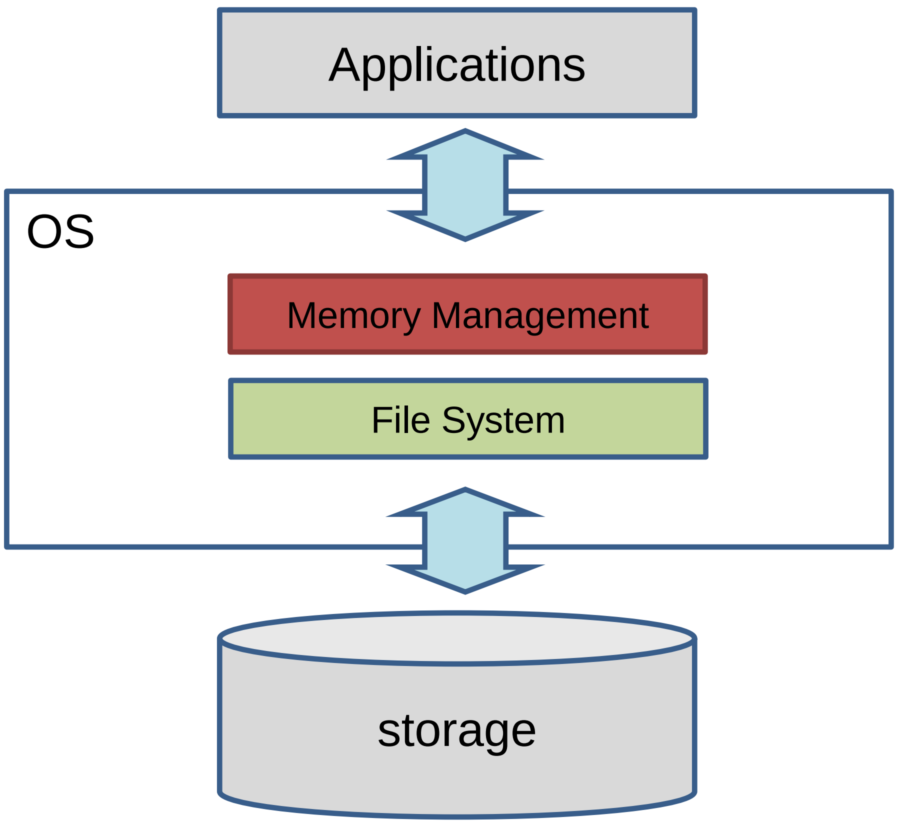
- NAND Flash Memory
  - Erase-before-write
  - Sequential writes inside the erase unit
  - Limited program/erase (P/E) cycle
- Flash Translation Layer (FTL)
  - Garbage collection
  - Wear-leveling
  - Bad block management
- Host-side Issues
  - Poor random write performance
  - Life span and reliability

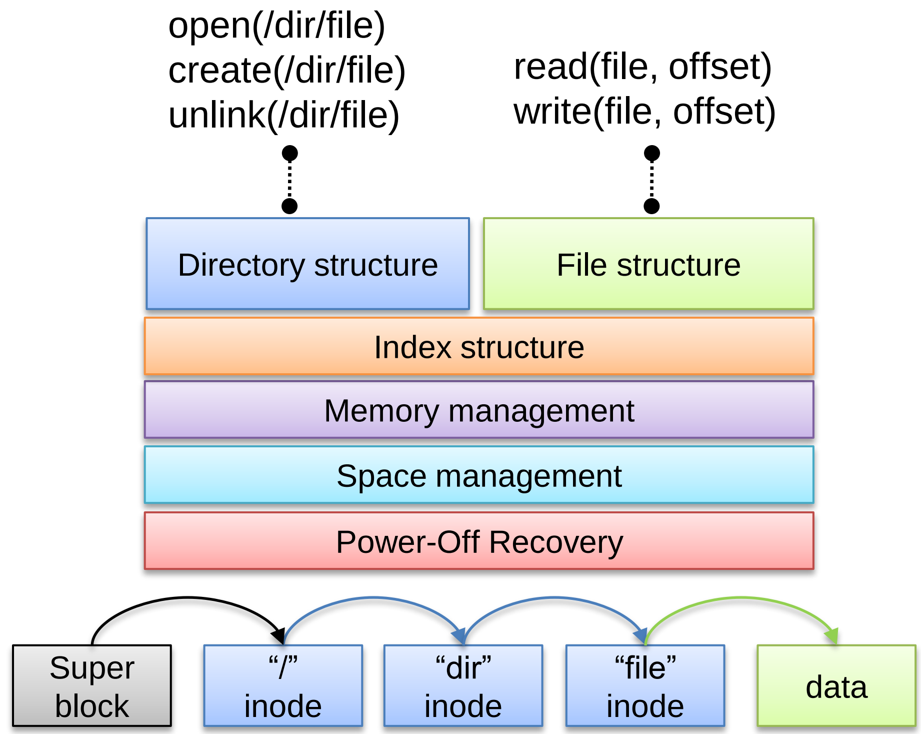


Source: INDILINX Barefoot controller

Introduction

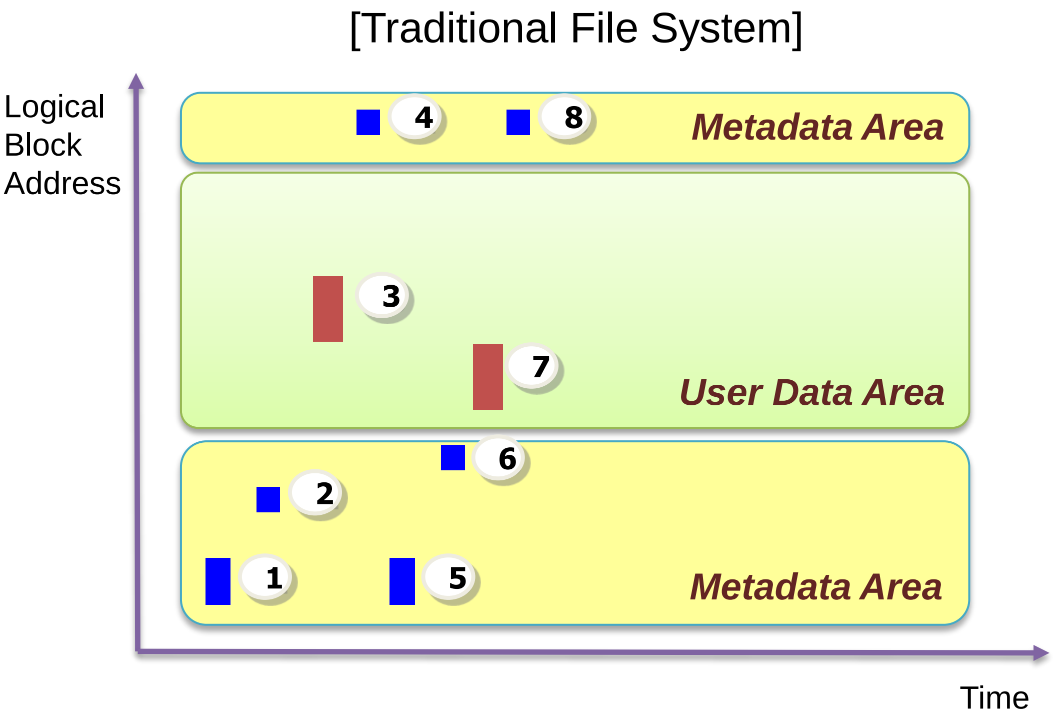
- File System
  - Serve directory and file operations to users
  - Manage the whole storage space

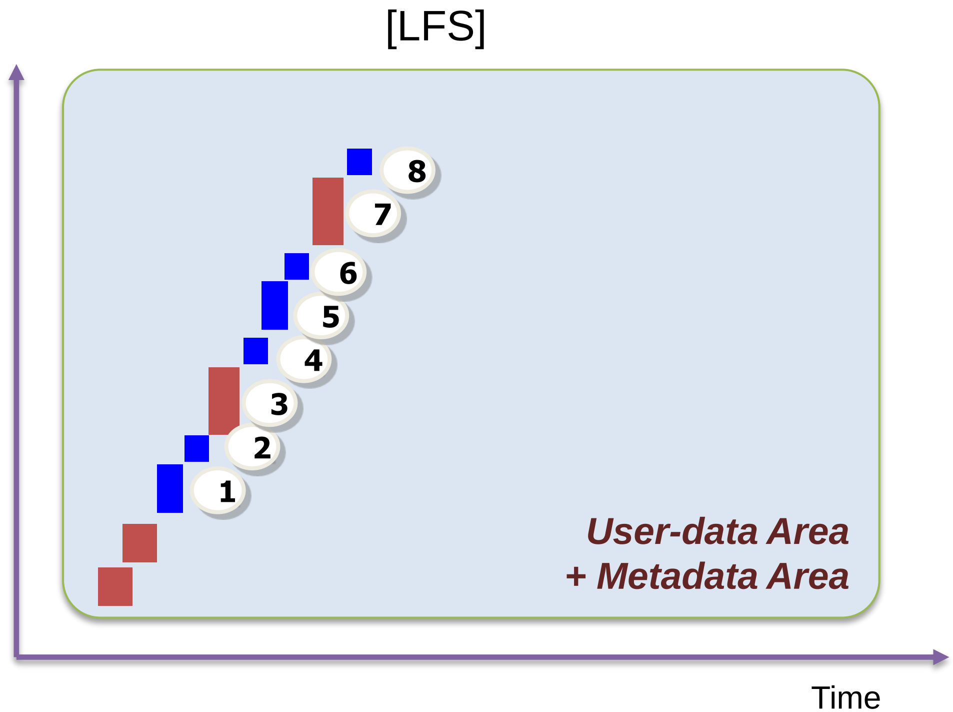




Log-structured File System

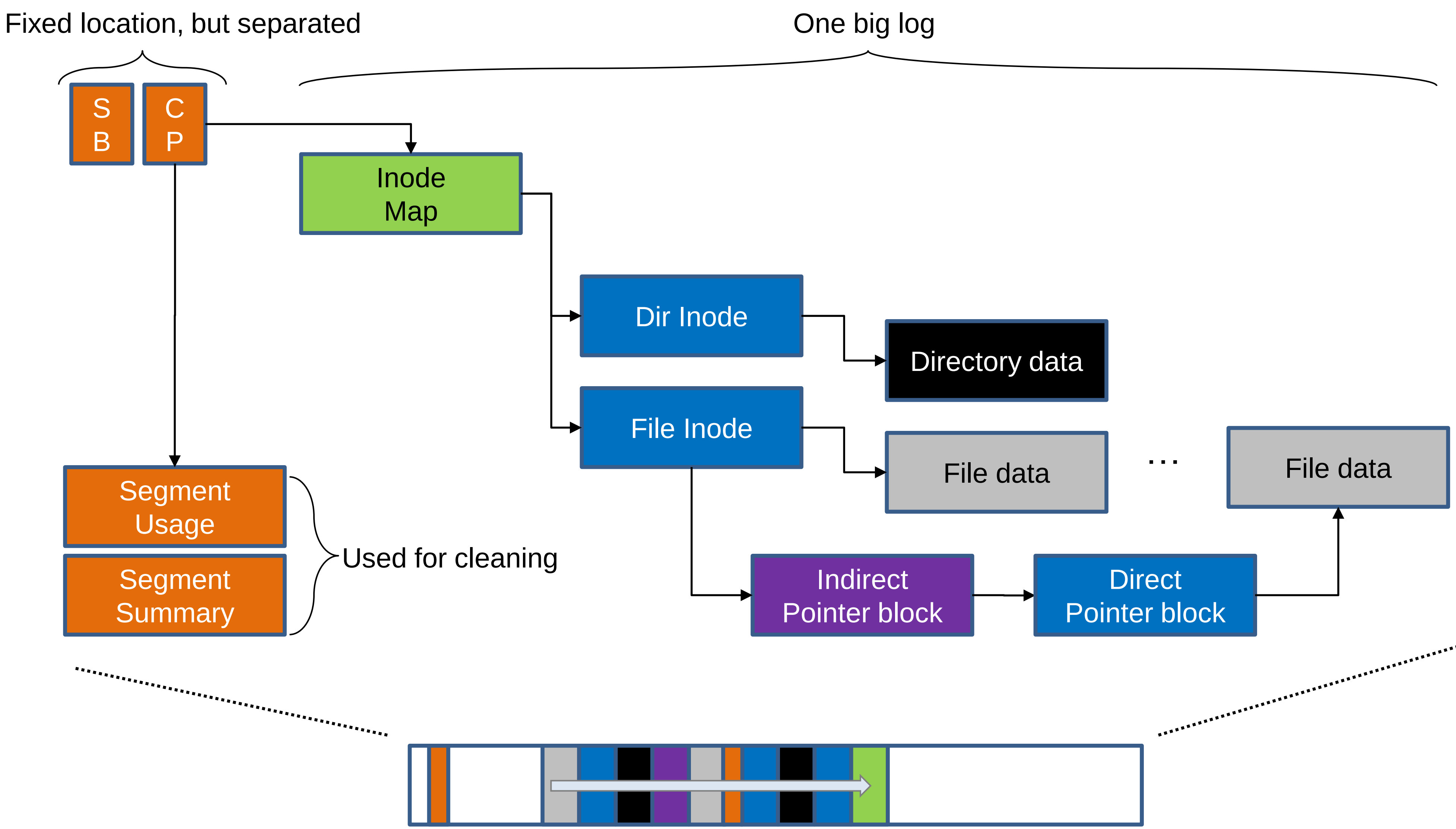
- Log-structured File System (LFS)[1]
  - Assume the whole disk space as a big contiguous area
  - Write all the data sequentially
  - Recover quickly with “checkpoint”





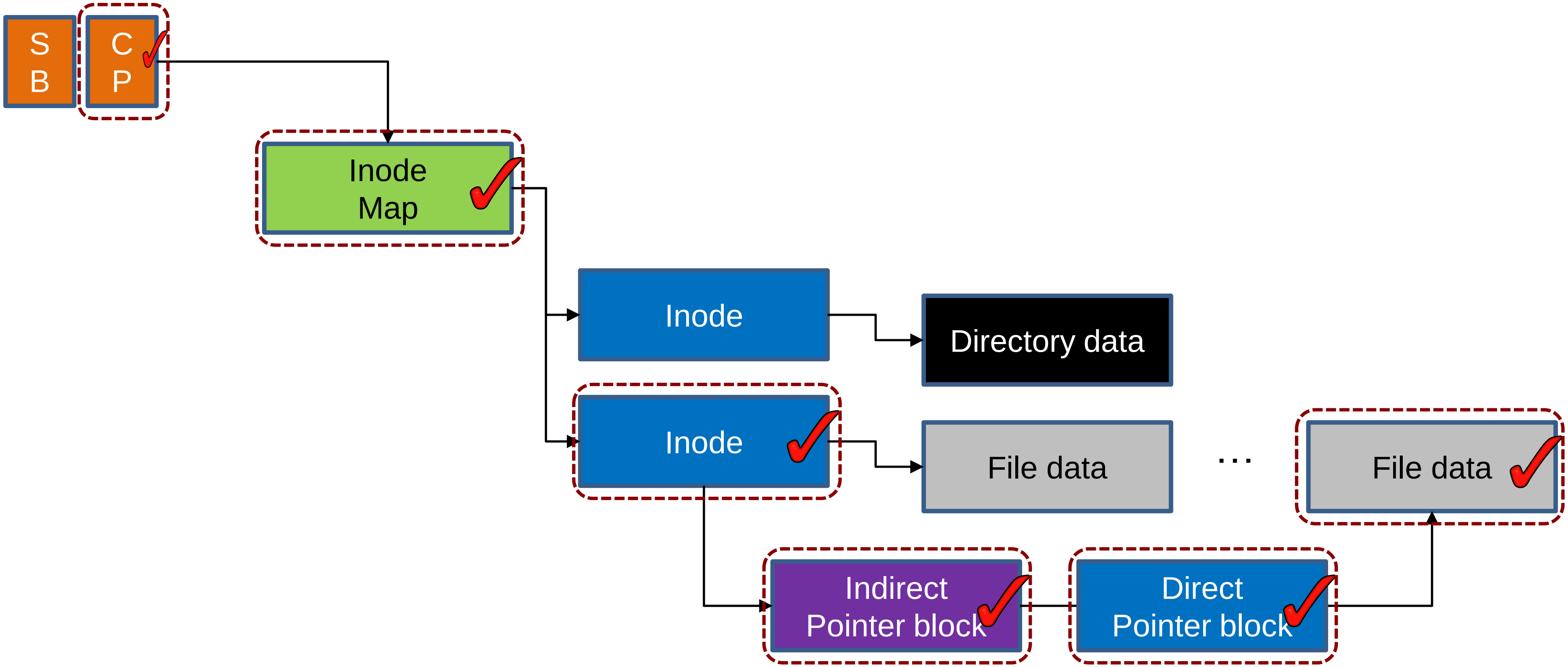
[1] Mendel Rosenblum and John K. Ousterhout. 1992. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.* 10, 1 (February 1992), 26-52.

Log-structured File System (Index Structure)



Design Issues I

- Wandering tree problem [2]
  - Propagates index updates recursively
- Goal
  - Eliminate or relax the update propagation



[2] BITYUTSKIY, A. 2005. JFFS3 design issues. http://www.linux-mtd.infradead.org/.

Design Issues II

- Cleaning Process
  - Reclaim obsolete data scattered across the whole storage for new empty log space
  - Get victim segments through referencing segment usage table
  - Load parent index structures of there-in data identified from segment summary blocks
  - Move valid data by checking their cross-reference

- Goal
  - Hide cleaning latencies to users
  - Reduce the amount of valid data to be moved
  - Move data quickly

- Specific Issues
  - Cleaning in the background
  - Victim selection policy
  - Hot and cold data separation
  - Instant valid data identification

Design of F2FS

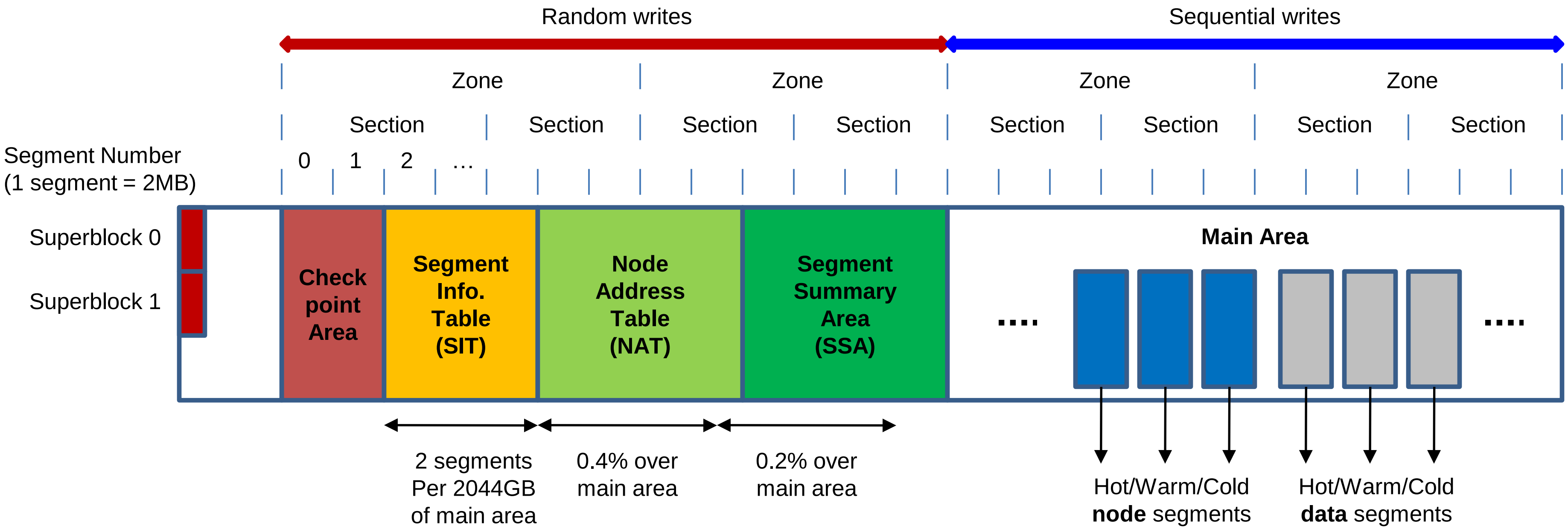
- Flash Awareness
  - Enlarge the random write area for performance, but provide the high spatial locality
  - Align FS data structures to the operational units in FTL

- Wandering tree problem
  - Use a term, “node”, that represents inodes as well as various pointer blocks
  - Introduce Node Address Table (NAT) containing the locations of all the “node” blocks

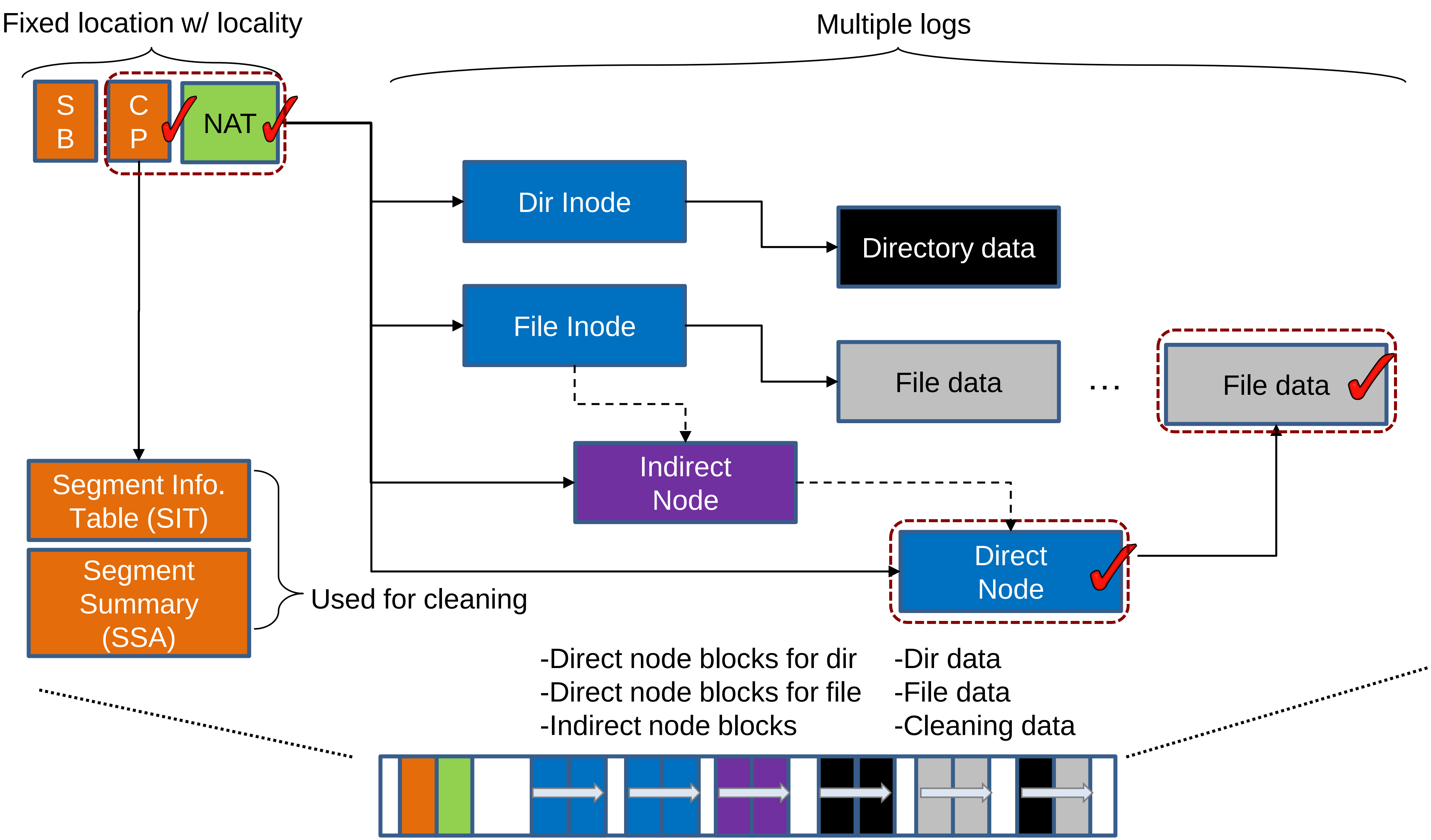
- Cleaning overhead
  - Support a background cleaning process
  - Support greedy and cost-benefit algorithms for victim selection policies
  - Support multi-head logs for static hot and cold data separation
  - Introduce adaptive logging for efficient block allocation

Design of F2FS (On-disk Layout)

- Flash Awareness
  - All the FS metadata are located together for locality
  - Start address of main area is aligned to the zone size
  - Cleaning operation is done in a unit of section
- Cleaning overhead
  - Six active logs for static hot and cold data separation



Design of F2FS (Index Structure)



Design of F2FS (Cleaning Process)

- Background cleaning process
  - A kernel thread doing the cleaning job periodically at idle time

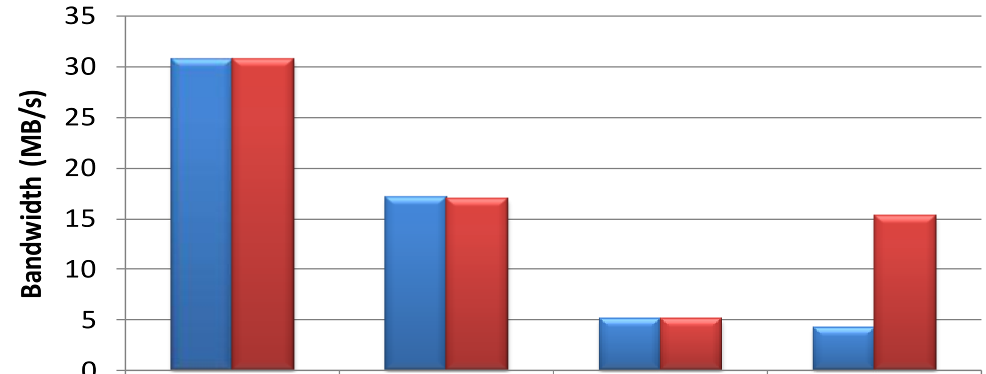
- Victim selection policies
  - Greedy algorithm for foreground cleaning job
  - Cost-benefit algorithm for background cleaning job

- Block allocation policy
  - Threaded logging
    - Reuse obsolete blocks without cleaning operations
    - Cause random writes
  - Copy-and-compaction
    - Need cleaning operations with some latency
    - Cause no random writes
  - Adaptive logging
    - Normally, copy-and-compaction is adopted
    - If there is not enough free space, the policy is dynamically changed to threaded logging

Performance Evaluation (micro benchmark)

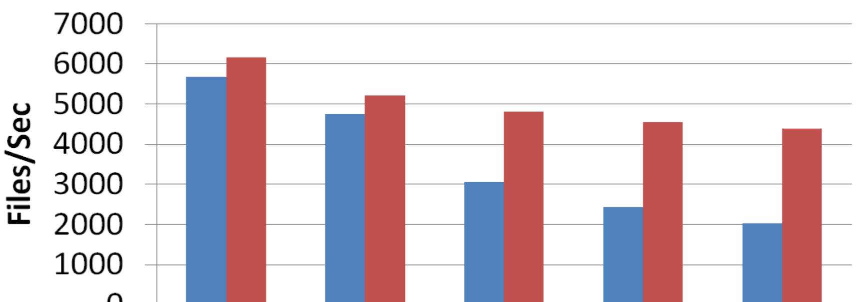
[System Specification]

CPU	ARM Coretex-A9 1.2GHz
DRAM	1GB
Storage	Samsung eMMC 64GB
Kernel	***Linux 3.3***
Partition Size	***12 GB***



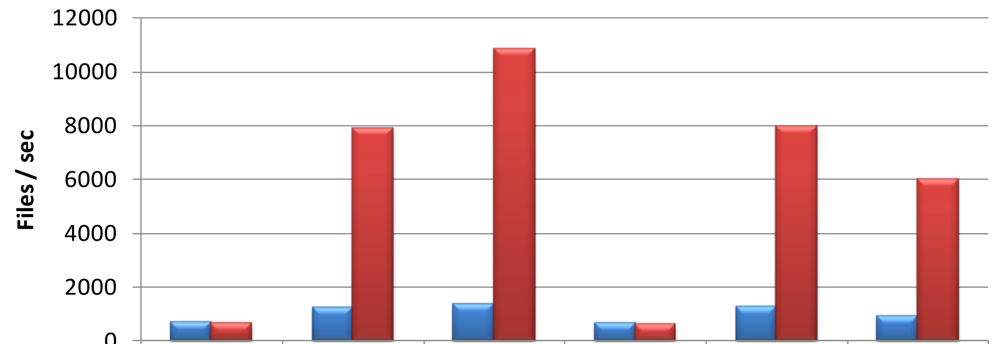
	seq. Read	seq. Write	rand. Read	rand. Write
EXT4	30.753	17.066	5.06	4.15
F2FS	30.71	16.906	5.073	15.204

[iozone]



	5000	10000	15000	20000	25000
ext4	5675.9	4741.6	3065	2439.5	2038.6
f2fs	6167.3	5207.9	4815.9	4544.4	4382.9

[fs_mark]



	seq.create	seq.stat	seq.delete	rand.create	rand.stat	rand.delete
EXT4	692	1238	1370	663	1250	915
F2FS	631	7871	10832	620	7962	5992

[bonnie++]

Performance Evaluation (Galaxy S2)

Quadrant	ext4	f2fs	
I/O Performance	**3476**	**3724**	**248(7.13%)**

RLBench	ext4	f2fs	
Overall(sec)	**40.76 (0.98)**	**33.57 (2.01)**	**-7.18(17.62%)**
1000 INSERTs(sec)	**20.59 (0.9)**	**11.96 (1.82)**	**-8.63(41.93%)**
25000 INSERTs in a transaction(sec)	1.79 (0.08)	1.71 (0.01)	-0.08(4.48%)
25000 INSERTs into an indexable table in a transaction(sec)	1.79 (0.08)	1.75 (0.03)	-0.03(1.9%)
100 SELECTs without an index(sec)	0.08 (0.04)	0.05 (0.02)	-0.03(34.21%)
100 SELECTs on a string comparison(sec)	0.07 (0.02)	0.15 (0.21)	0.08(-108.33%)
Creating an index(sec)	0.82 (0.04)	0.94 (0.09)	0.12(-14.08%)
5000 SELECTs with an index(sec)	1.47 (0.11)	1.54 (0.06)	0.07(-4.75%)
1000 UPDATEs without an index(sec)	4.48 (0.04)	4.48 (0.12)	0(0%)
25000 UPDATEs with an index(sec)	3.99 (0.08)	4.14 (0.18)	0.15(-3.81%)
INSERTs from a SELECT(sec)	1.62 (0.15)	1.81 (0.27)	0.19(-11.7%)
DELETE without an index(sec)	1.47 (0.25)	2.02 (0.43)	0.55(-37.41%)
DELETE with an index(sec)	1.43 (0.26)	1.64 (0.3)	0.21(-14.85%)
DROP TABLE(sec)	1.16 (0.11)	1.48 (0.2)	0.31(-26.98%)

Androbench	ext4	f2fs	
Sequential Read(MB/s)	41.58 (2.72)	41.78 (2.05)	0.2(0.48%)
Sequential Write(MB/s)	4.81 (1.19)	5.63 (1.15)	0.82(17.05%)
Random Read(MB/s)	3.39 (0.06)	3.46 (0.07)	0.07(2.12%)
Random Write(MB/s)	0.25 (0.01)	0.48 (0.01)	0.23(93.5%)
SQLite Insert(s)	15.05 (0.37)	16.63 (0.39)	1.58(-10.5%)
SQLite Update(s)	6.28 (0.27)	3.51 (0.31)	-2.77(44.16%)
SQLite Delete(s)	6.49 (0.19)	3.89 (0.56)	-2.59(39.96%)

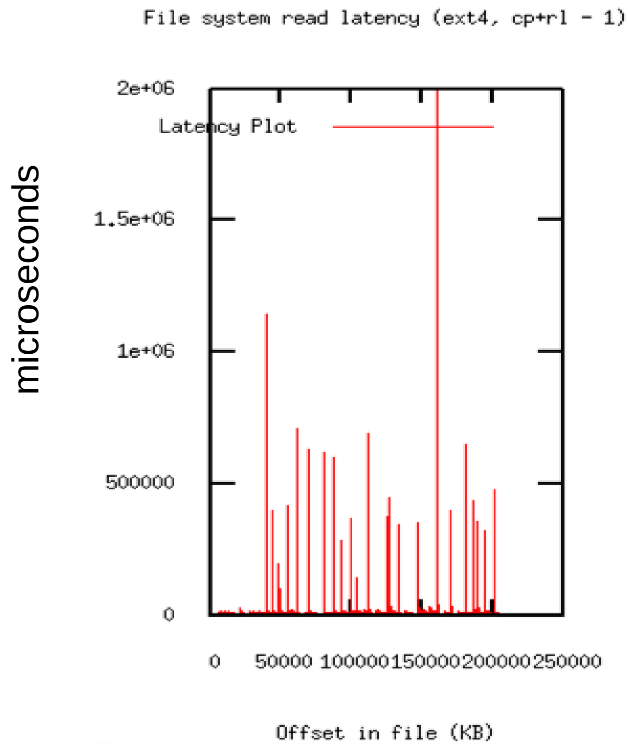
Reduce total execution time by 18% over ext4
Reduce DB insertion time by 42% over ext4

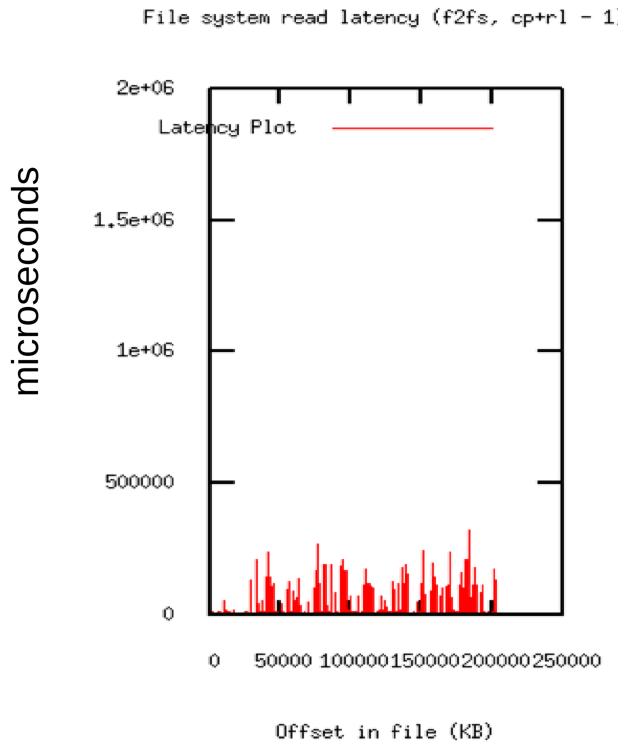
Improve random write performance by 94% over ext4

Reduce DB update time by 44% over ext4

Performance Evaluation (Galaxy S2)

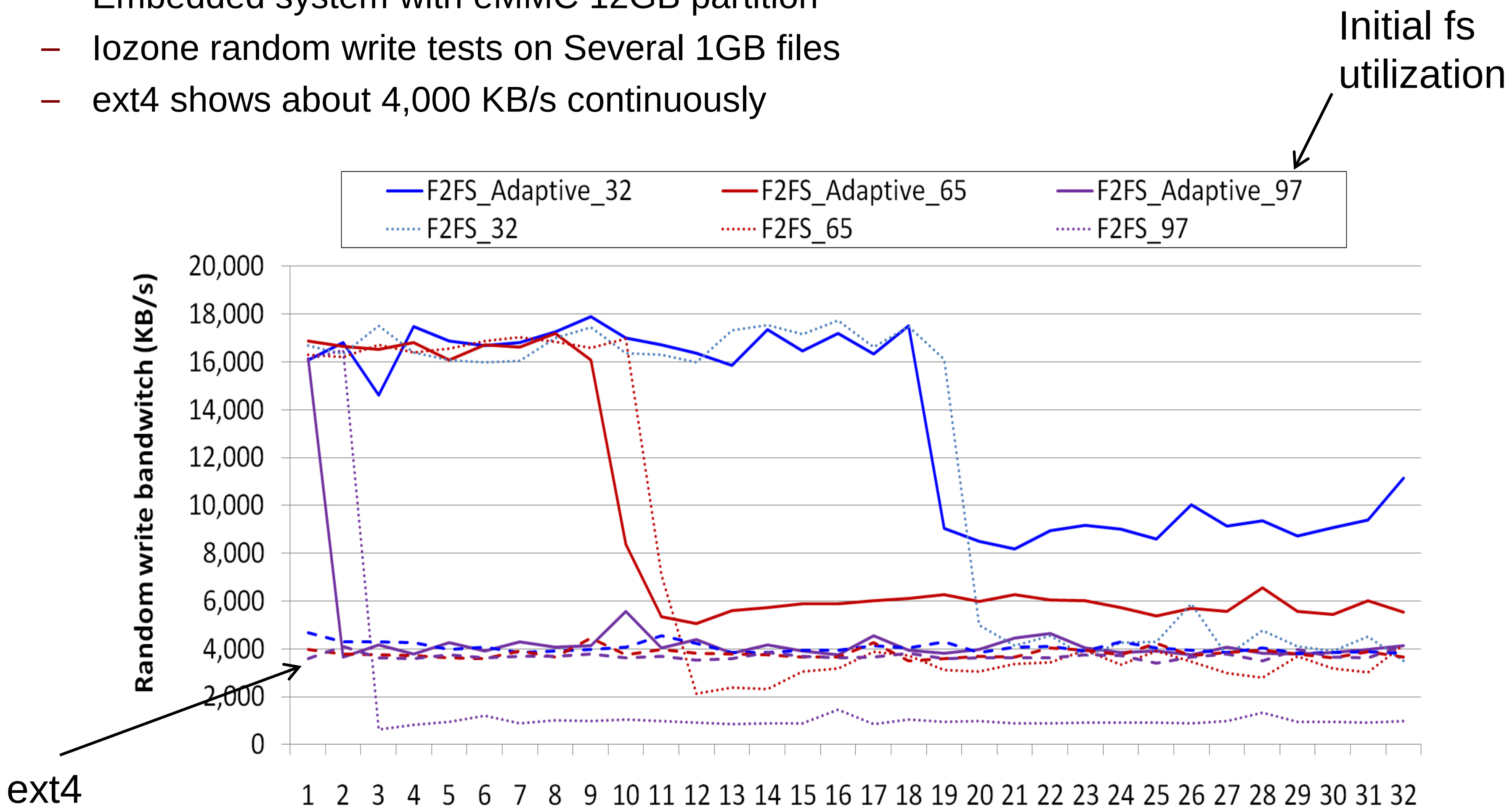
- Multi-tasking Scenario
  - Run RLBench and file copying in background
  - Measure video streaming latencies in foreground
  - High latency may cause cracks in videos
- Results
  - Ext4 shows a latency by up to 2.5s, and over 0.5s of latencies more than 10 times
  - F2FS shows a latency by up to 0.35s





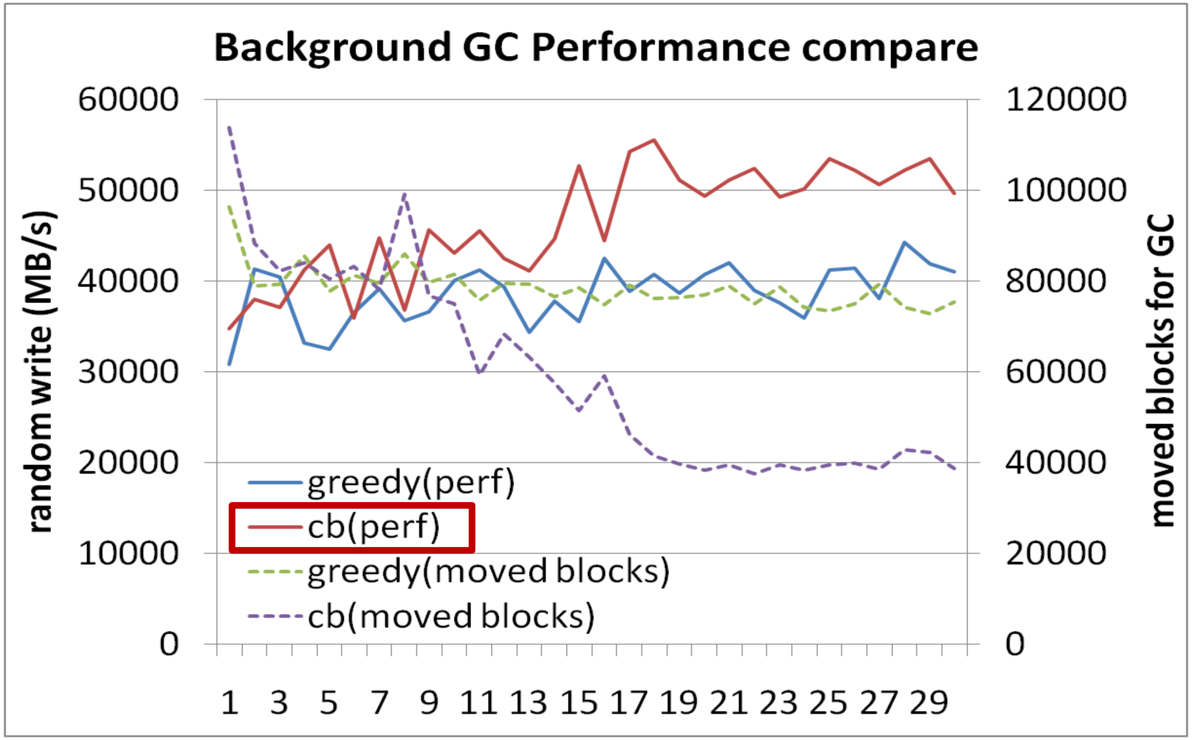
Adaptive Logging

- Setup
  - Embedded system with eMMC 12GB partition
  - Iozone random write tests on Several 1GB files
  - ext4 shows about 4,000 KB/s continuously



Victim Selection Policies

- Setup
  - In x86, set 3.7 GB partition
  - Create three files having 1GB data
  - Write 256MB data randomly to the three files
  - Write 256MB data randomly to one of them, 30 times



Galuxy Nesus

CPU	ARM Coretex-A9 1.2GHz
DRAM	1GB
Storage	Samsung eMMC (VFX) 16GB
Kernel	***3.0.8***
Android ver.	***Ice Cream Sandwich***

< Clean >

Items		Ext4	F2FS	Improv.
Contact sync time (seconds)		431	358	20%
App install time (seconds)		459	457	0%
RLBench (seconds)		92.6	78.9	17%
IOZoneWith AppInstall (MB/s)	Write	8.9	9.9	11%
	Read	18.1	18.4	2%

< Aged >

Items		Ext4	F2FS	Improv.
Contact sync time (seconds)		437	375	17%
App install time (seconds)		362	370	-2%
RLBench (seconds)		99.4	85.1	17%
IOZone With AppInstall (MB/s)	Write	7.3	7.8	7%
	Read	16.2	18.1	12%

Summary

- Flash-Friendly File System
  - Focused on Performance and Reliability
  - Not, on new fancy functionalities

- Ubuntu 12.04 LTS
  - Format “/” as F2FS
  - Install & compile kernel & run several applications

- Galaxy S2, S3, and Nexus
  - Format “/data” as F2FS
  - Factory reset & run android apps

- Further Optimization
  - Together!

Thank you!