An examination of UBI

TOSHIBA CORPORATION
Core Technology Center
Embedded System Core Technology Development Dept.
Shinji Namihira

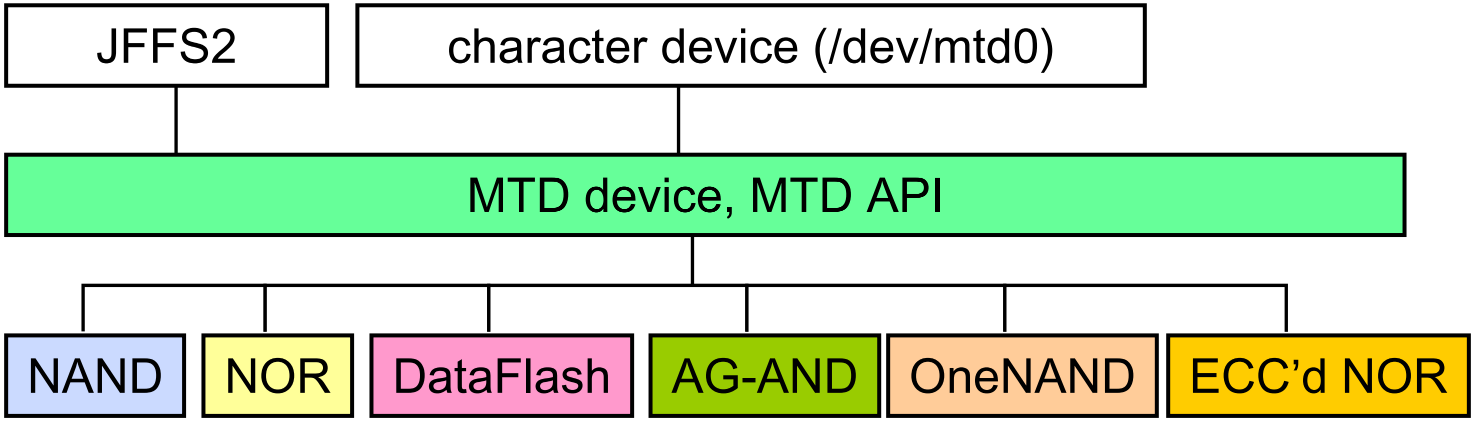
Aug 29, 2008

Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

Current Flash File systems & Driver : Structure

- **Bare Flash Chips**
- **MTD**
- **Flash File Systems**



Bare Flash Chips

Differences from other storage devices

- Erase operation is required before rewriting
- 2 Types of Technologies

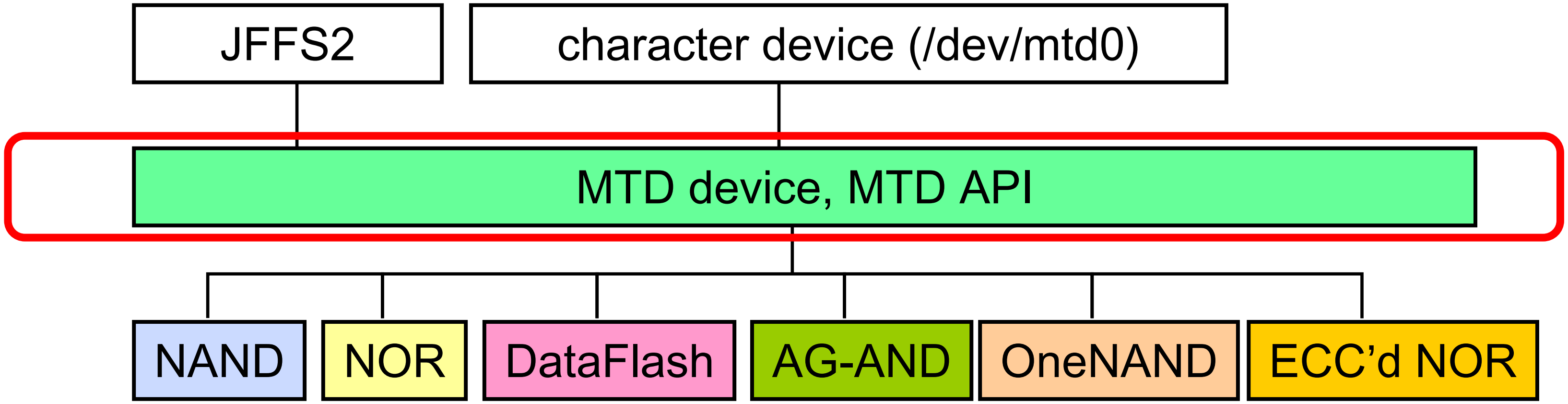
type	access		XIP	speed		
	Erase	R/W		Erase	Read	Write
NOR	sequential	random	OK	Poor	Good	Poor
NAND	sequential	sequential	N/A	Good	Fair	Good

Problems

- Life-Time ⇨ Wear Leveling
- Bit-Flips (NAND) ⇨ ECC
- Bad Block (NAND) ⇨ Management

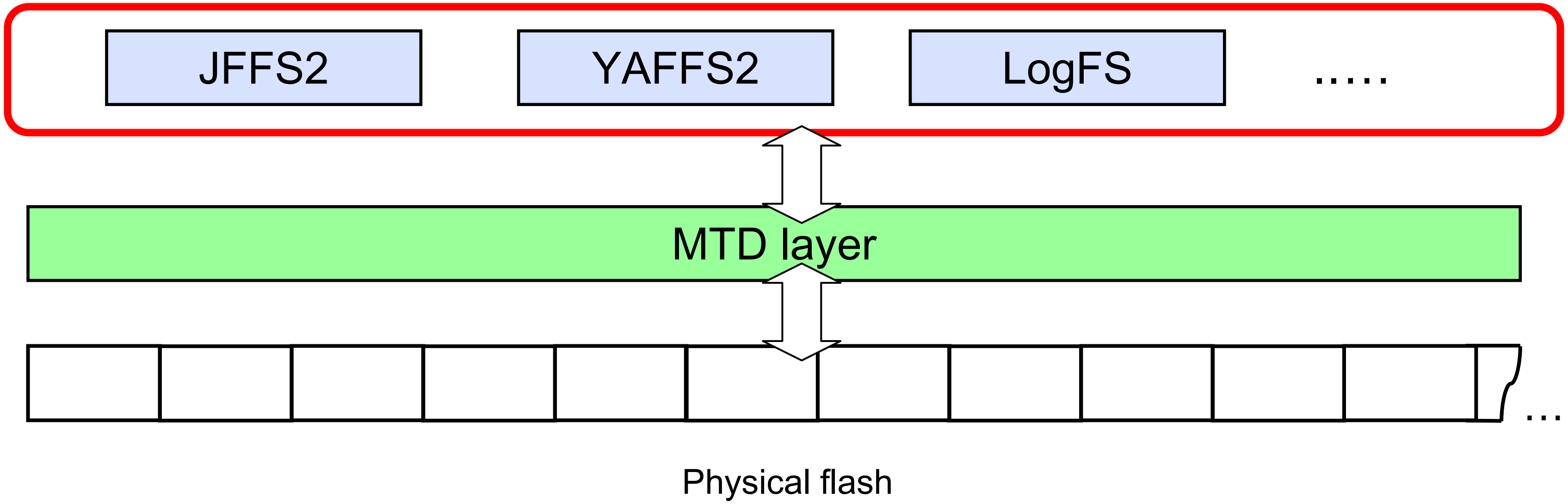
MTD

- **MTD stands for “Memory Technology Devices”**
- **MTD is a Linux subsystem (`drivers/mtd/`)**
- **MTD provides uniform access to various flash devices**
- **MTD provides a generic API for that**
- **MTD provides an “MTD device” abstraction**



Flash File Systems : Features

- **Bad Block Management**
- **Wear Leveling**
- **Journaling**



  - Flash File Systems call MTD I/F
  - Flash File Systems handle the physical MTD Partitions & Blocks

Flash File Systems : overview

- **JFFS2 (Journaling Flash File System Ver.2)**
  - License : GPL
  - How to get : Included in Vanilla Kernel since 2.4.10.
- **YAFFS / YAFFS2 (Yet Another Flash File System)**
  - License : GPL
  - How to get : http://www.yaffs.net
- **LogFS (Log File System)**
  - License : GPL
  - How to get : http://logfs.com/logfs/

▪ http://tree.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree19
“The Comparison of Flash File system performance”

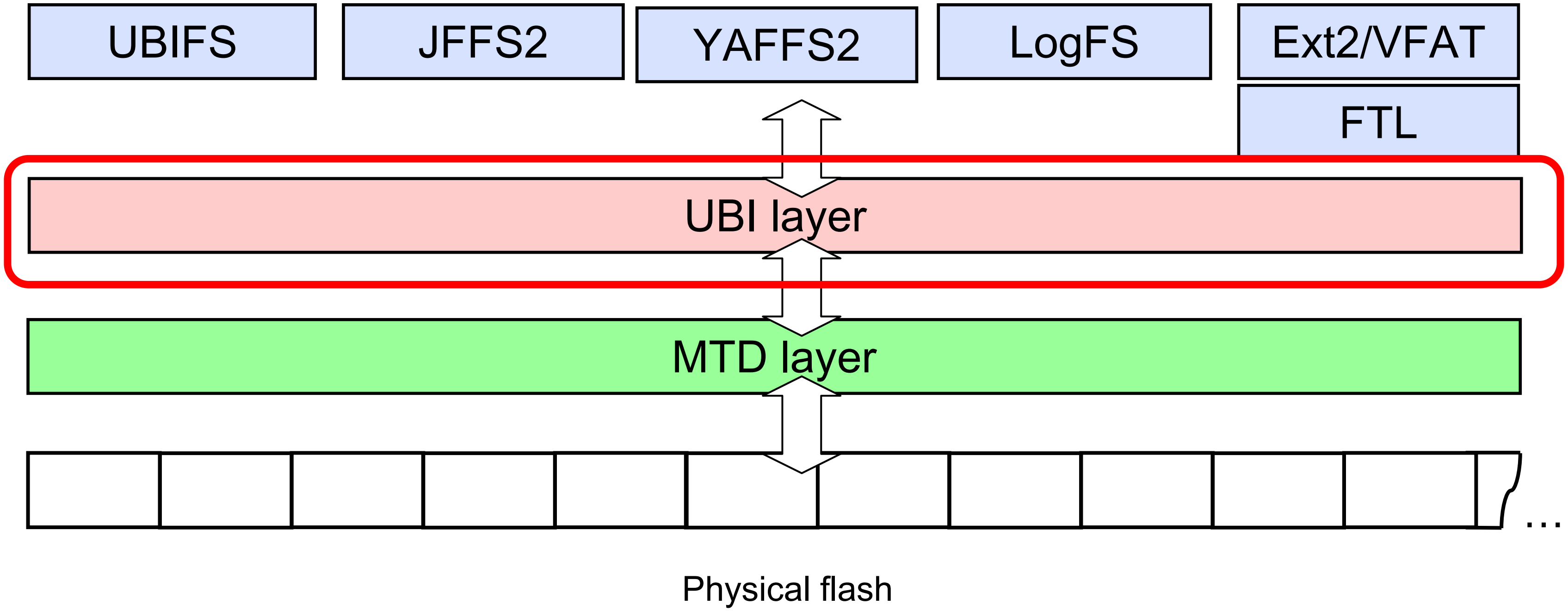
▪ http://tree.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree20
“Flash File system, current development status ”

Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

UBI overview : UBI Layer

- **UBI is built on top of the MTD devices**
- **UBI stands for "Unsorted Block Images"**
- **UBI provides sequential Logical Blocks to UBI Clients**



UBI overview : vs. MTD

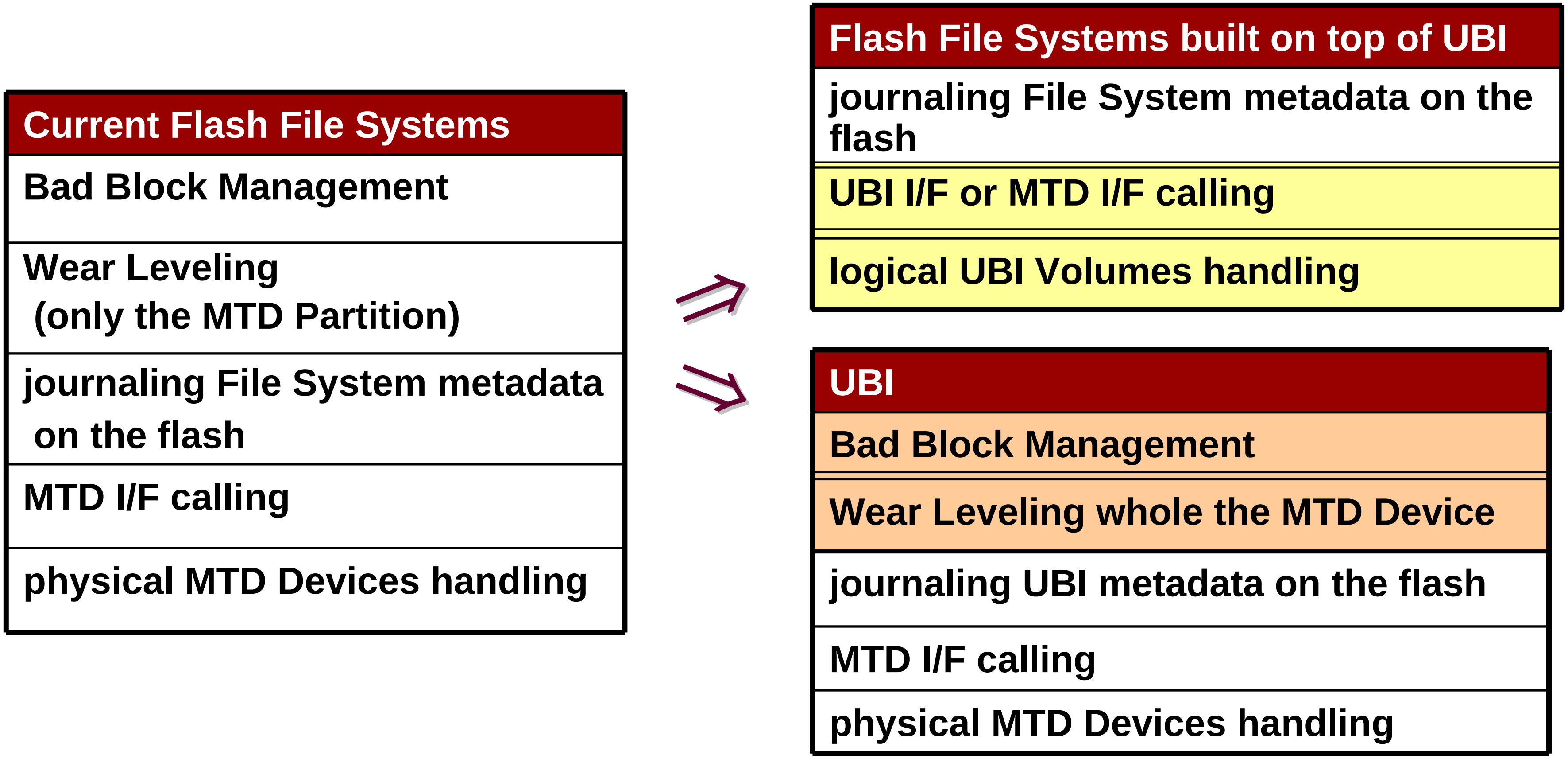
MTD partition	UBI Volume
Consists of physical eraseblocks (PEB)	Consists of logical eraseblocks (LEB)
Does not implement wear-leveling	Implements wear-leveling
Admits of bad PEBs	Devoid of bad LEBs

⇓

Advantages of UBI

- **Allows dynamic volume creation, deletion, and re-sizing**
  ⇒ more flexible
- **Eliminates the “wear” problem ⇒ simpler software**
- **Eliminates bad eraseblocks problem ⇒ simpler software**

UBI overview : vs. Current Flash File Systems



Current Flash File Systems

Bad Block Management

Wear Leveling
(only the MTD Partition)

journaling File System metadata on the flash

MTD I/F calling

physical MTD Devices handling

Flash File Systems built on top of UBI

journaling File System metadata on the flash

UBI I/F or MTD I/F calling

logical UBI Volumes handling

UBI

Bad Block Management

Wear Leveling whole the MTD Device

journaling UBI metadata on the flash

MTD I/F calling

physical MTD Devices handling

- UBI makes it simple to design a new Flash File System
- UBI provides MTD I/F emulator to Flash File Systems
  - Kernel Config : MTD_UBI_GLUEBI
    Emulate MTD Device for UBI Clients (default: off)

UBI overview : vs. Current similar Modules

- **vs. FTL (Flash Translation Layer)**

  FTL is similar as the layer between Device and File System but..

  – FTL is a block device emulation layer

  – FTL may be on the top of UBI for block device File Systems

  mailing list : “Block Device Emulation over UBI”

  http://lists.infradead.org/pipermail/linux-mtd/2008-January/020381.html

- **vs. LVM (Logical Volume Manager)**

  LVM is similar as the layer of providing logical & flexible but..

  – LVM is for block devices

  - LVM doesn’t support Wear Leveling
  - LVM doesn’t support Bad Block Management

UBI was designed for bare flashes which may be found in Embedded Systems.

by. http://www.linux-mtd.infradead.org/ubi.html

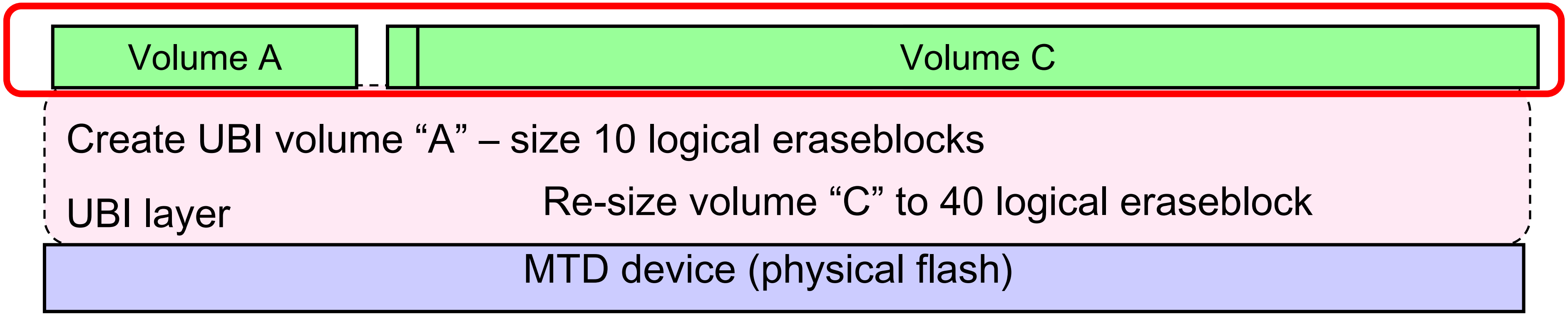
Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

Block Management : Logical Volume

- **UBI provides Logical Volumes instead of Physical MTD Partitions to UBI Clients**
- **User can create, delete & resize a UBI Volume on the fly**
- **Volume type**
  - Static : for Read Only data (protected by CRC checksum)
  - Dynamic : for Read / Write data (not protected in UBI)

→ for Wear Leveling

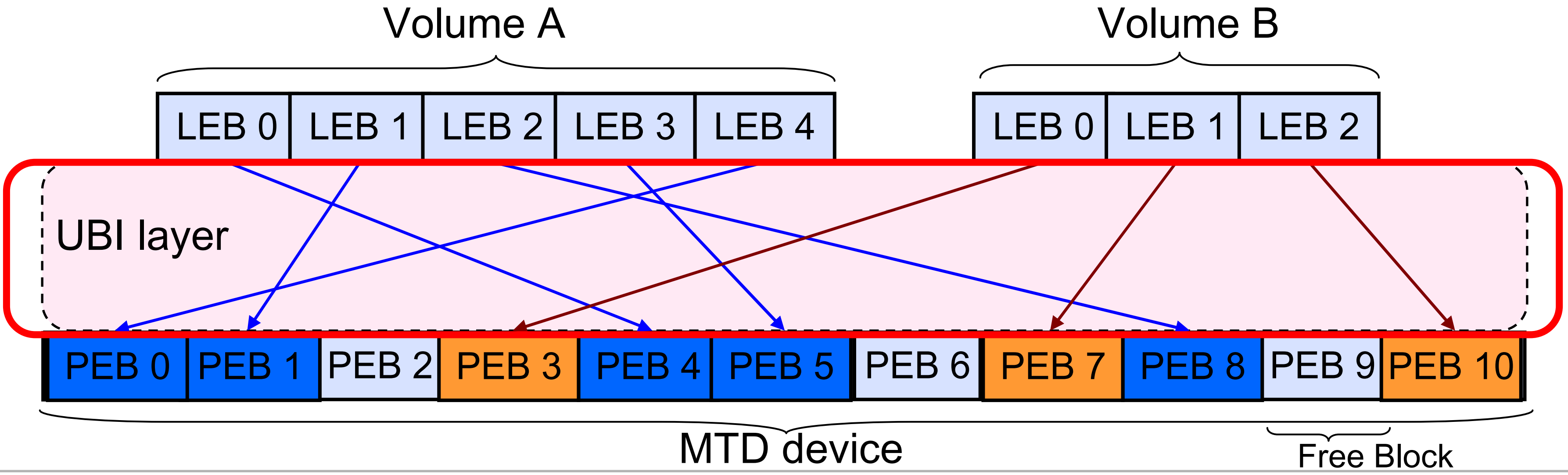

Volume A
Volume C
Create UBI volume "A" – size 10 logical eraseblocks
UBI layer
Re-size volume "C" to 40 logical eraseblock
MTD device (physical flash)

Block Management : Logical Block to Physical Block

- **UBI Clients handle Logical Volume & Blocks**
- **UBI managements the mapping of Logical to Physical**
- **User can reserve initial amount of Free Blocks for Bad Block & Wear Leveling**
  - Kernel Config : MTD_UBI_BEB_RESERVE

    Reserved Block % for Bad Block

    (default : 1 , range: 0 - 25)



Block Management : Physical Block Structure

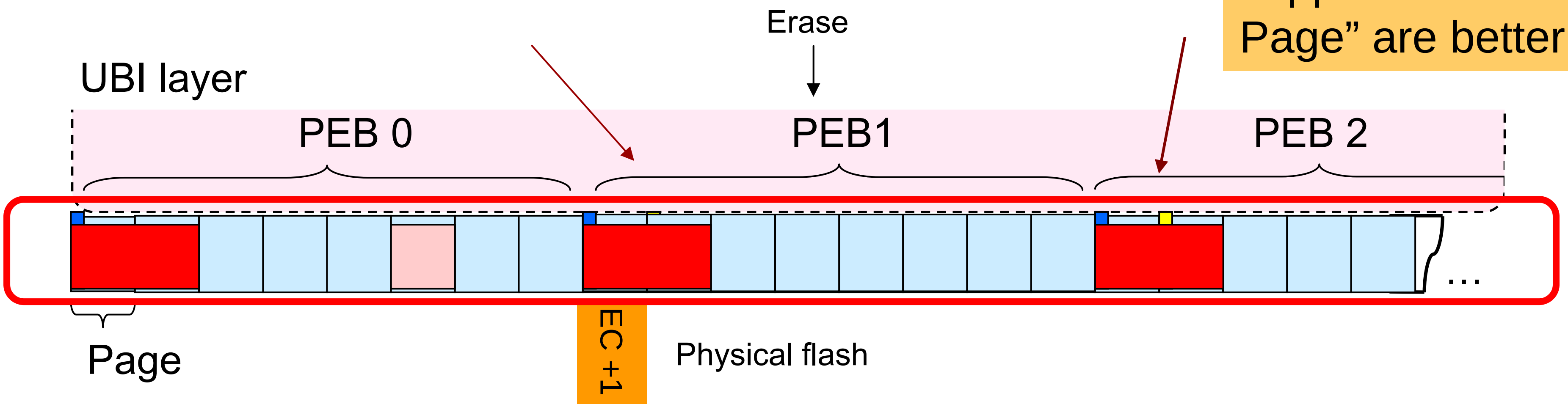
- **Erase Counter Header & Volume ID Header**

header	store	size	where	when
EC	Erase Counter	64 bytes	first page	format
Volume ID	mapped Volume & LEB	64 bytes	2nd page	mapped

- **Data Alignment**
  - Read / Write size : Page (typically 512B - 2KB)
  - Erase size : Block (typically 16KB - 128KB)

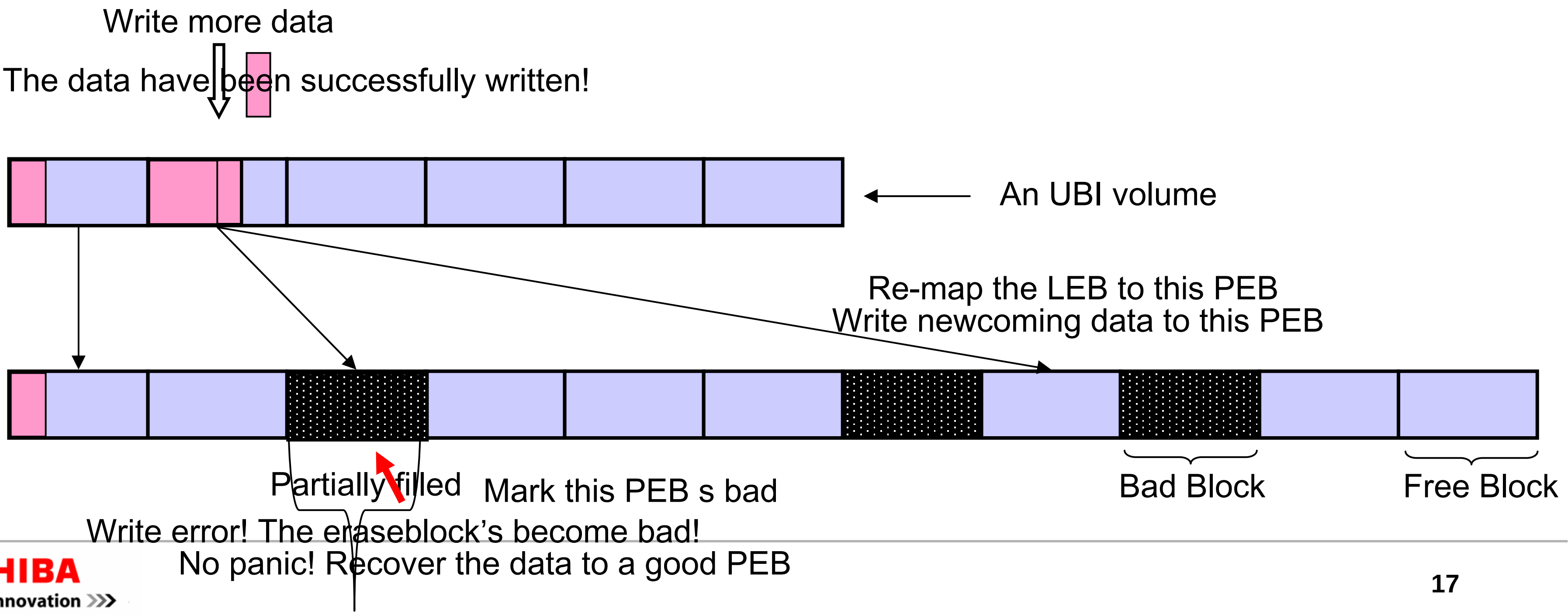
Scale Issue

Flash Chips that support "Sub-Page" are better



Block Management : Read & Write working

- **UBI Clients appoint ...**
  - Logical Volume Number
  - Logical Block Number
  - Offset from the Block Start point
  - Length
- **In Write operation, even if UBI fails in MTD Write I/F, it doesn’t return I/O error (it assigns another Free Block)**


Write more data
The data have been successfully written!
An UBI volume
Re-map the LEB to this PEB
Write newcoming data to this PEB
Partially filled
Mark this PEB s bad
Bad Block
Free Block
Write error! The eraseblock’s become bad!
No panic! Recover the data to a good PEB

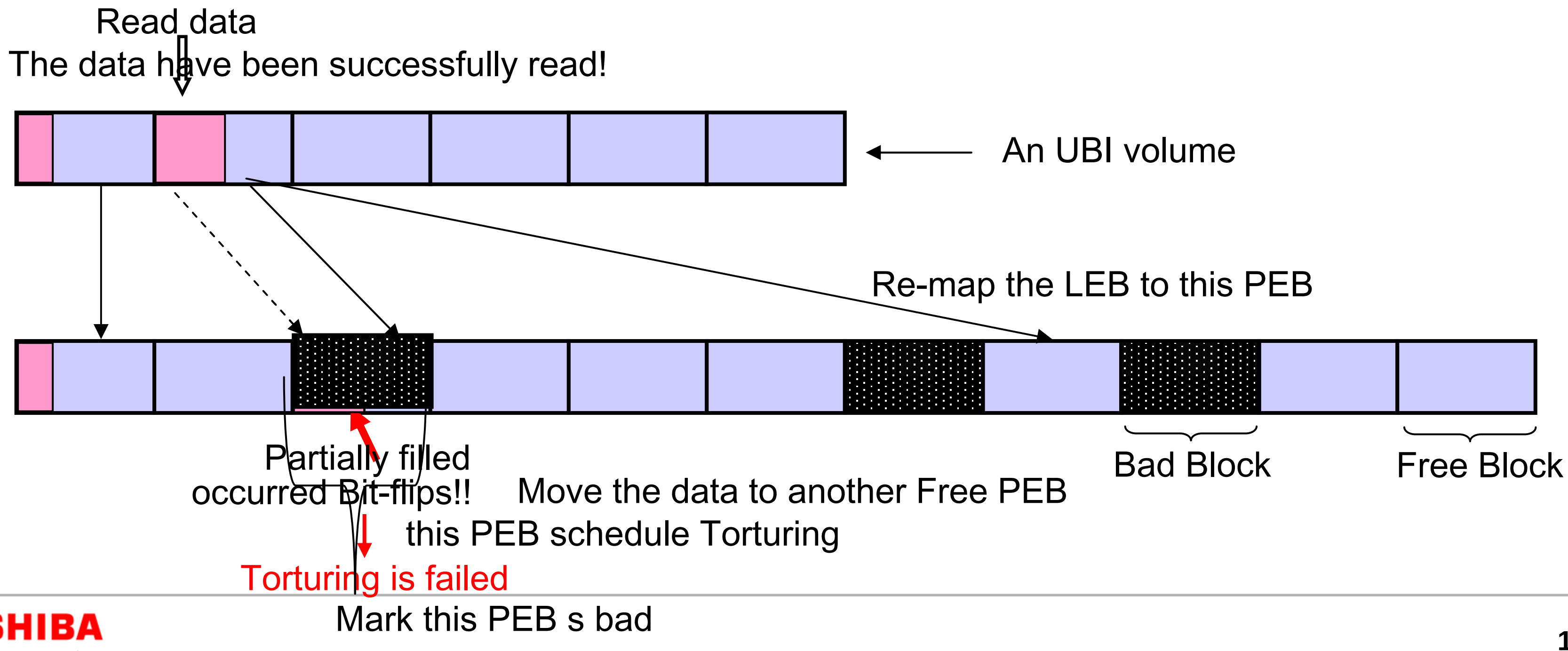
Block Management : Bit-Flip handling

- **In Read operation, if UBI detects corrected Bit-Flip, it doesn't keep using the Physical Block for reliability**

"Scrubbing"

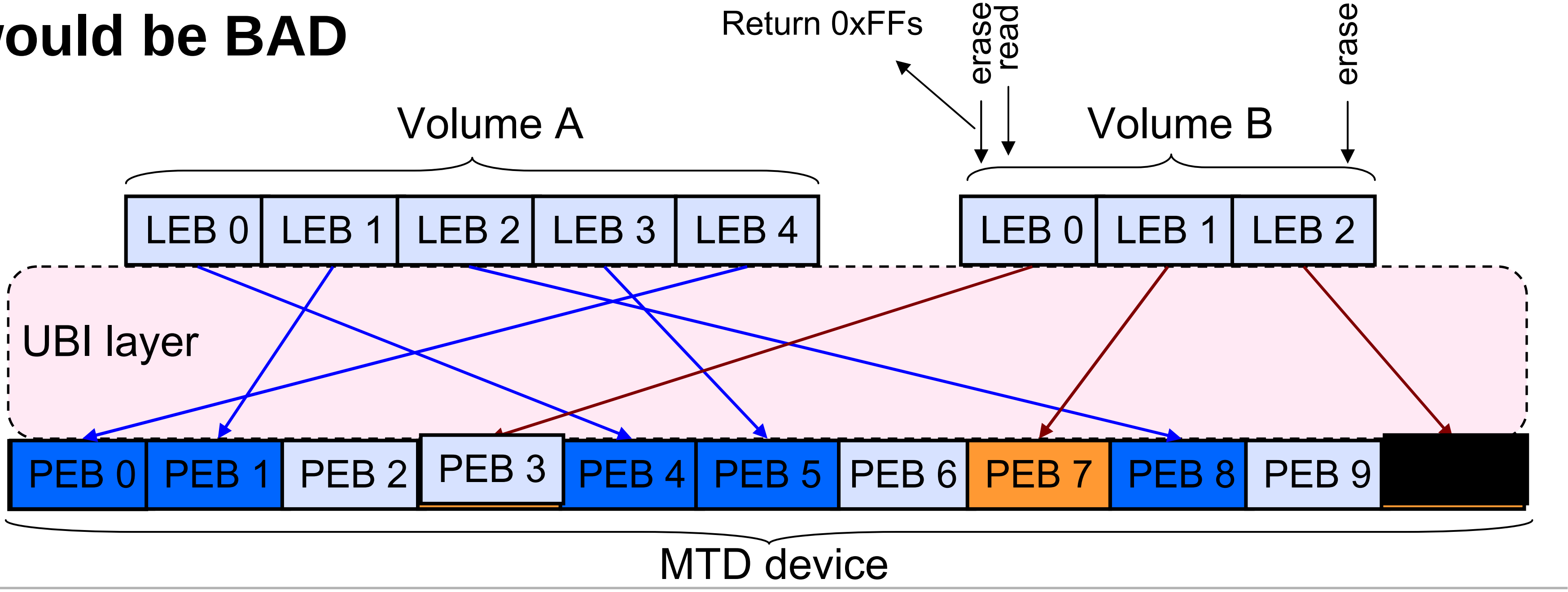
– UBI moves the correct data to another Free Block & do "Torturing"

– "Torturing" is done asynchronously by a Background Thread


Read data
The data have been successfully read!
An UBI volume
Re-map the LEB to this PEB
Partially filled
occurred Bit-flips!!
Move the data to another Free PEB
this PEB schedule Torturing
Torturing is failed
Mark this PEB s bad
Bad Block
Free Block

Block Management : Erase working

- **UBI supports a “unmap” operation that just releases the mapping & returns success immediately**
- **UBI erases the Physical Block asynchronously by a Background Thread**
- **UBI handles the erased Logical Block & return 0xFFs immediately in Read operation**
- **If UBI failed to do MTD Erase I/F, the Physical Block would be BAD**



Block Management : Wear Leveling

- **UBI can do Wear Leveling across whole the flash chip (MTD Device)**

  ➡ This will extend the Life Time

- **Wear Leveling Type**
  - Dynamic : applies to Blocks whose Erase Counter increases frequently
  - Static : applies to Blocks whose Erase Counter doesn’t increase for a long time

- **User gives information of the Data Term in Write operation to UBI**
  - long
  - short
  - unknown

  ➡ This will provide more effective Wear Leveling

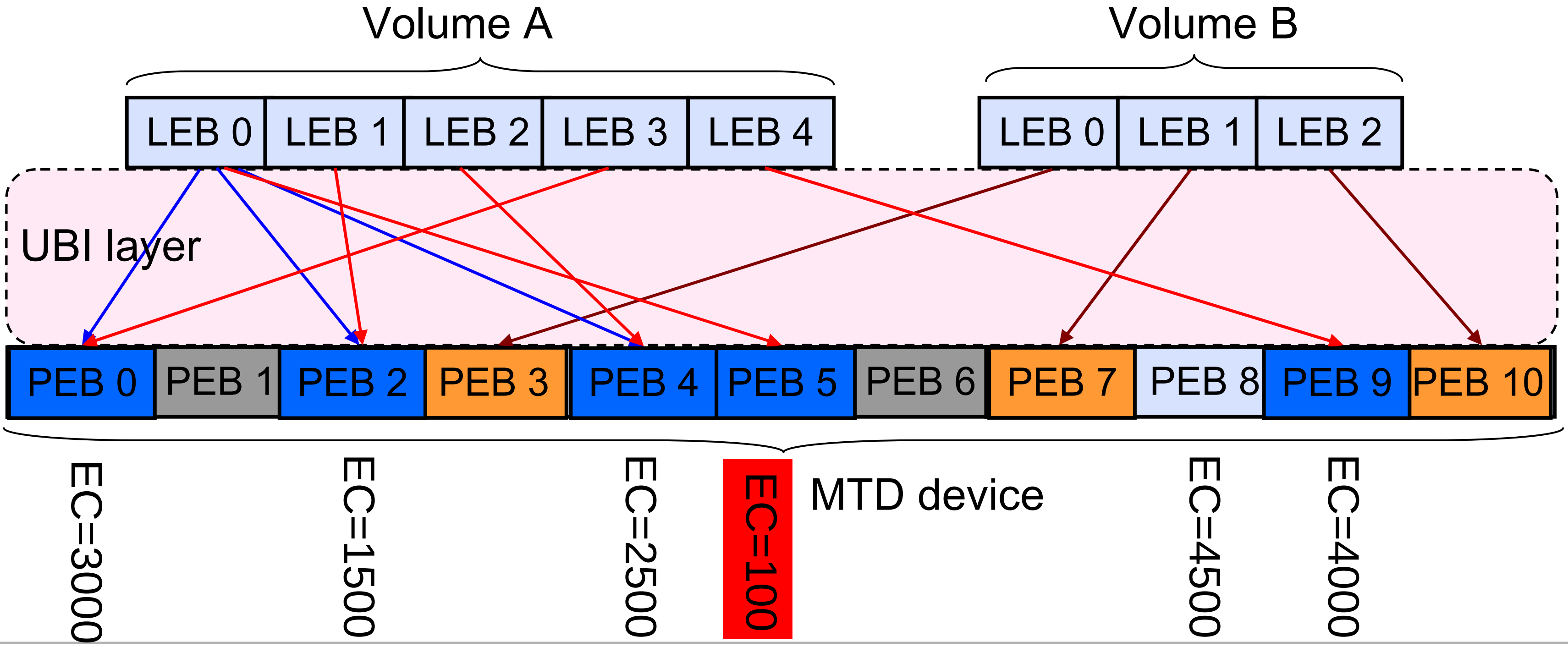
Block Management : Dynamic Wear Leveling

- **What?**

  UBI selects a Free Block that has lower Erase Counter

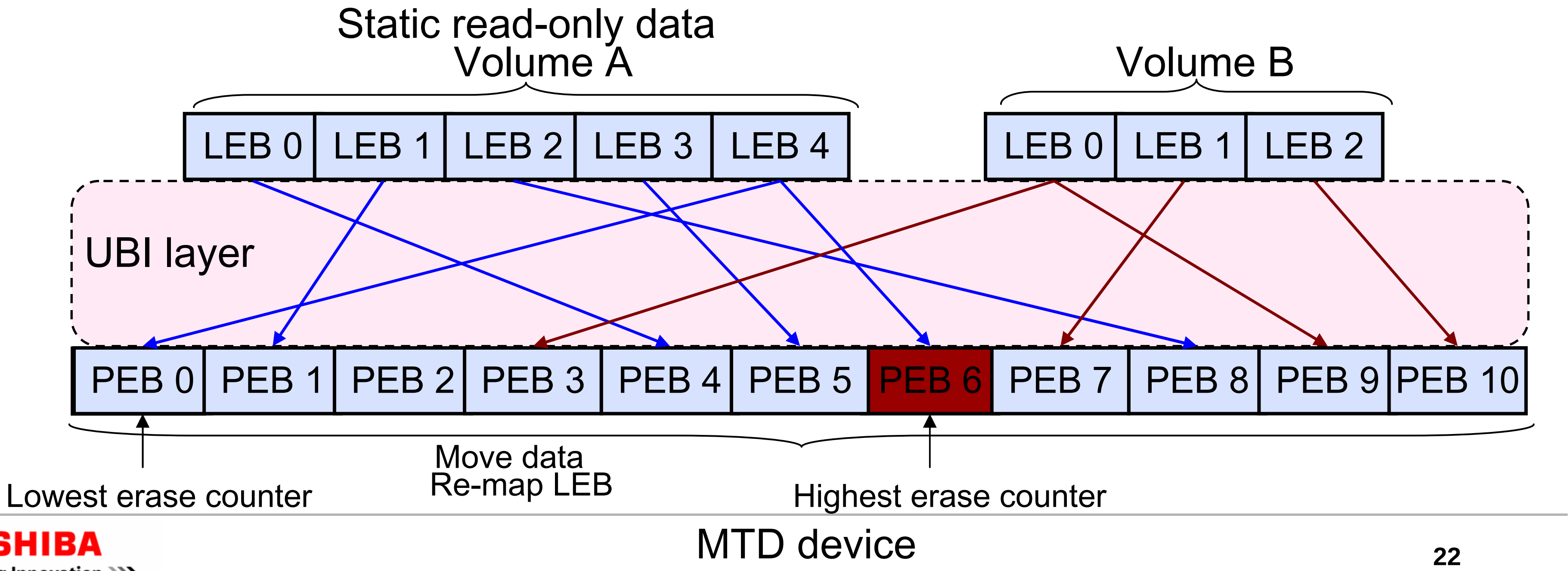
- **When?**

  In Volume create, resize, Write operation, or Scrubbing



Block Management : Static Wear Leveling

- **What?**
  - ➡ **UBI unmaps the lowest Erase Counter Block & makes it Free Block**
- **When?**
  - ➡ **Erase Counter MAX – MIN exceeds “Threshold”**
    in Volume delete or Erase operation
    (Wear Leveling is done by a Background Thread asynchronously)
    - **Kernel Config :** MTD_UBI_WL_THRESHOLD
      (default: 4096, Range: 2 – 65536)



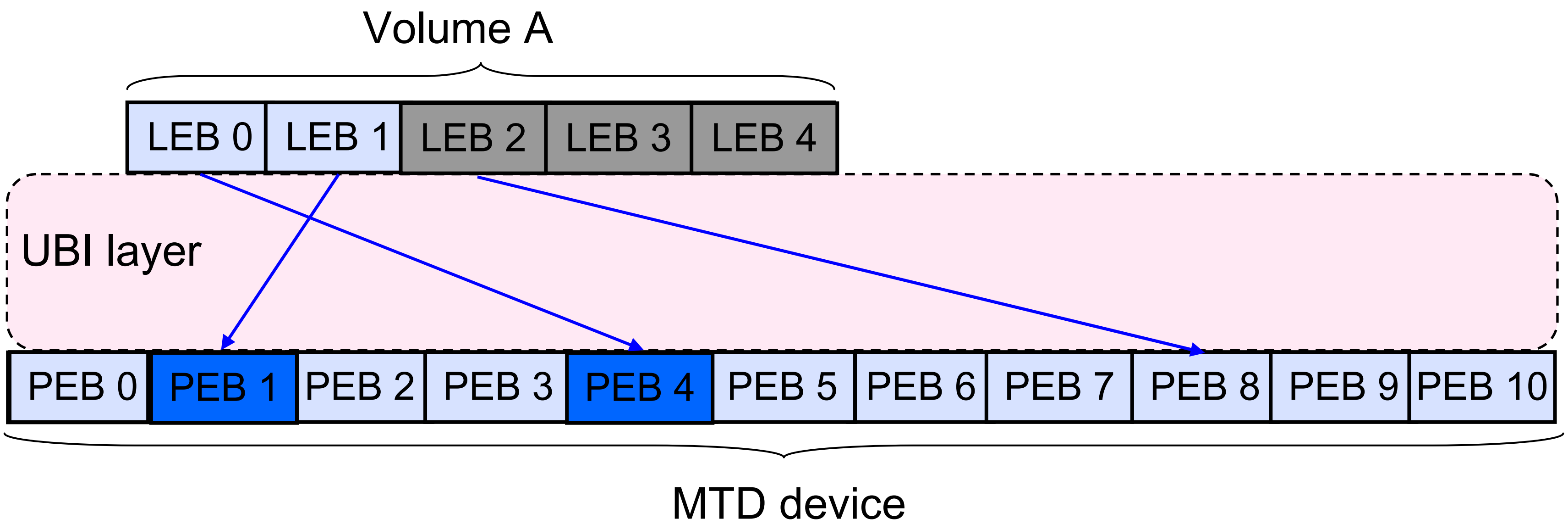
Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

Unclean Reboot : Issue of Corrupt Volume

(Case.1)

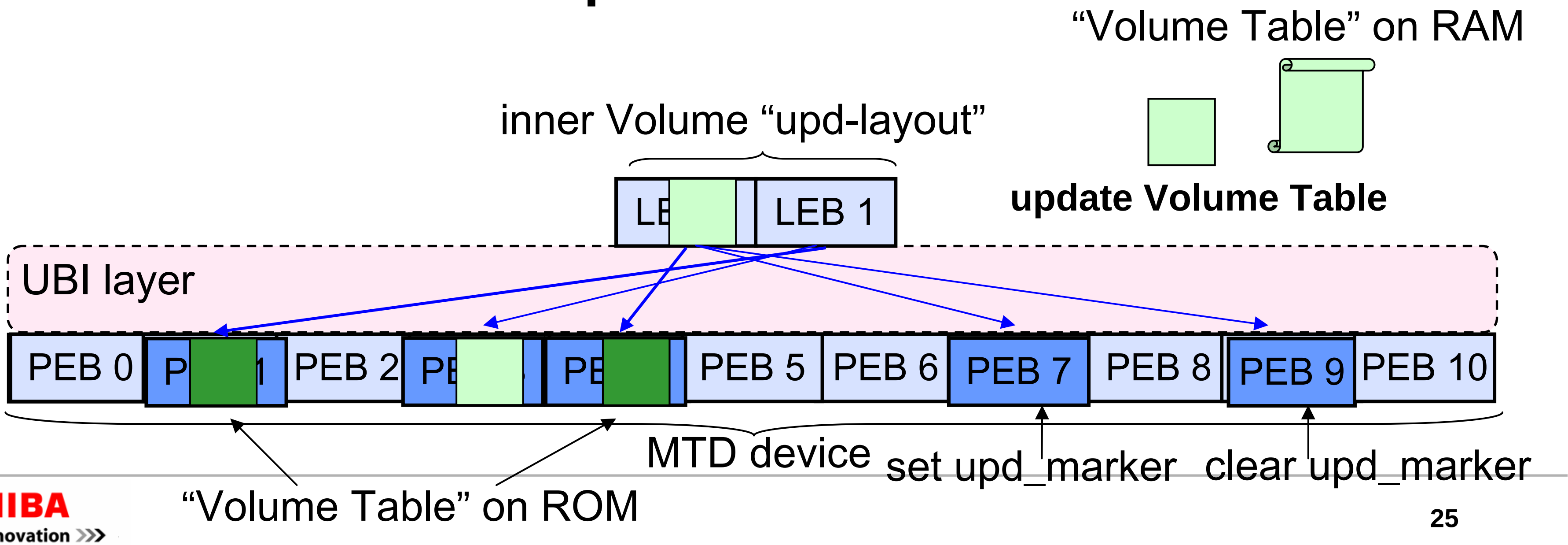
If an Unclean Reboot (like a power down) happened while creating a Volume ...



Unclean Reboot : Solution to Corrupt Volume

(Solution.1)

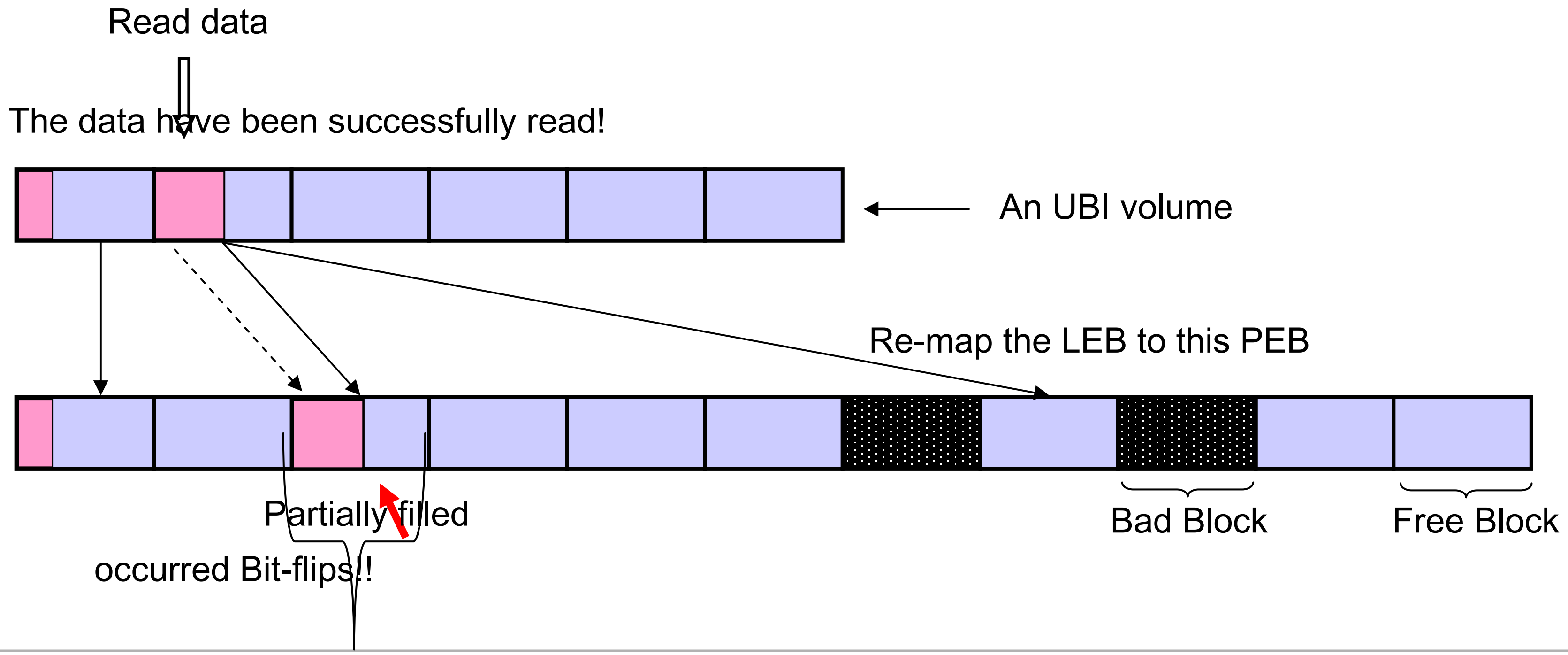
- **UBI stores duplicated “Volume Table” on the flash (2 Blocks)**
- **Even if LEB0 is interrupted on updating the new Volume Table, LEB1 keeps a normal previous Table**
- **UBI knows the interruption by “upd_marker” in the Table**
- **UBI reports the interruption to UBI Clients & wait for the clients to do Volume Update**



Unclean Reboot : Issue of Corrupt Mapping

(Case.2)

If an Unclean Reboot (like a power down) happened while moving a Physical Block in Wear Leveling or Scrubbing ...



Unclean Reboot : Solution to Corrupt Mapping

(Solution.2)

- **UBI knows which is newer by a sequence number, "sqnum" in the Volume ID Header**
- **UBI tries to use the newer one at first & if it is corrupted, UBI uses another one**

UBI provides "atomic logical eraseblock change" operation to UBI Clients (since 2.6.25)

and..

"UBI is designed to be tolerant of power failures and unclean reboots."

by. http://www.linux-mtd.infradead.org/faq/ubi.html

Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

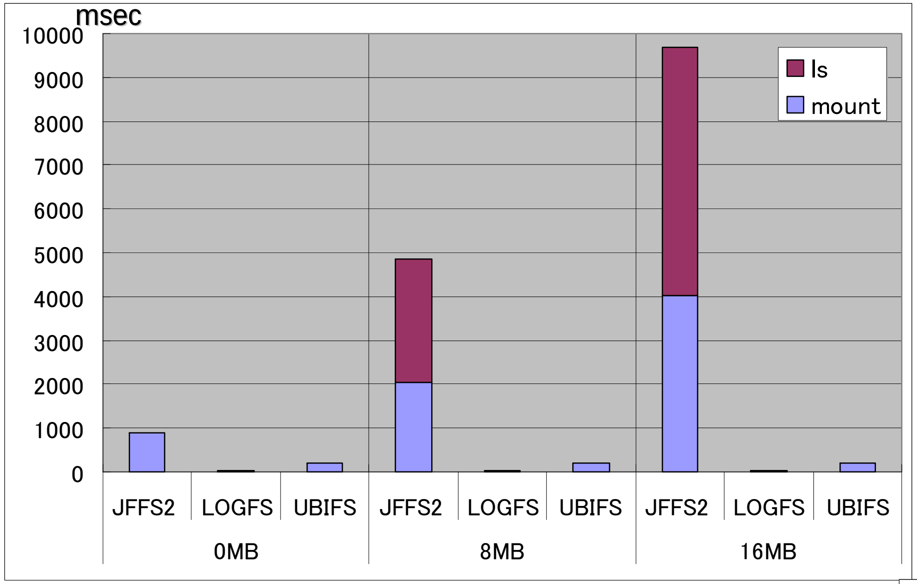
Boot Time : Issue

When UBI attached the MTD Device ..

- **UBI needs to scan all Physical Blocks to re-build EAT & ECT on RAM**
  - EAT : Eraseblock Association Table
  - ECT : Erase Counter Table
- **In addition, UBI has to calculate CRC32 checksum of each headers, too**

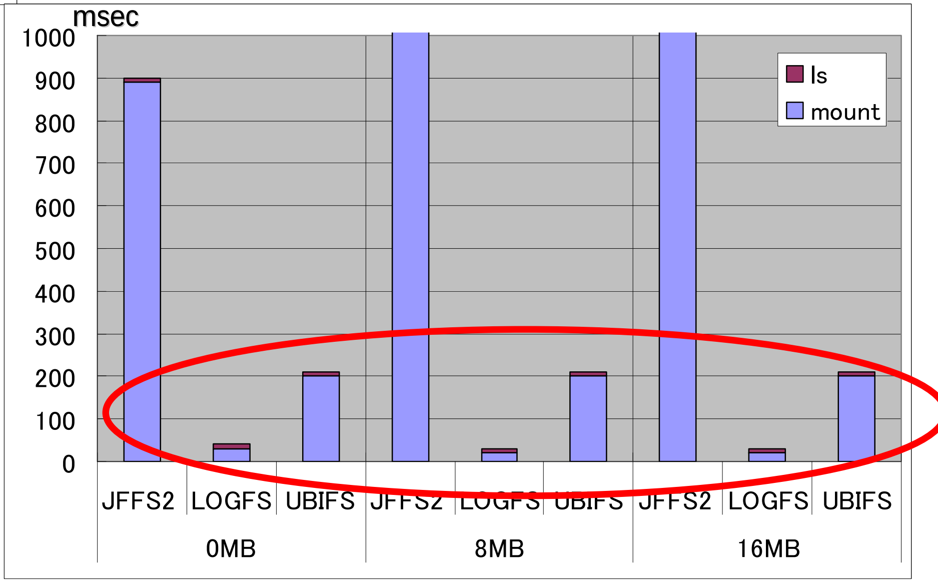
Boot Time linearly depends on the flash size

Boot Time : Performance of UBI+UBIFS (2.6.24)



- The mounting time of LogFS and UBIFS are independent of the size of a including file.

- The time of JFFS2 is much longer than LogFS/UBIFS.



by http://tree.celinuxforum.org/CelfPubWiki/JapanTechnicalJamboree20
"Flash File system, current development status "

Boot Time : Solution

(Solution)

UBI may store EAT & ECT on the flash, too. But..
those Tables are updated very frequently. So..
UBI doesn’t do that for simplicity & robustness.

↓

- **Select a flash chip which supports “Sub-Page”**
  (access only 1 Page per Block to re-build Tables)

- **“UBI2” is in Patch-2.6.27-rc4 & may be included in 2.6.27**

Nontheless, it is always possible to create UBI2 which would maintain the table in separate flash areas.

by. http://www.linux-mtd.infradead.org/ubi.html

Contents

- **Current Flash File Systems & Driver**
  - Bare Flash Chips
  - MTD
  - Flash File Systems
- **UBI**
  - UBI overview
  - Block Management
  - Unclean Reboot
  - Boot Time
- **Summary**

Summary :

item	point	UBI is..
Block Management	flexible & more effective	Good
Unclean Reboot	tolerant	Excellent
Boot Time	scan all physical blocks	No Good

⇩

depend on User
but..
expect UBI2

UBI information

- **License : GPL**
- **How to get : Include in Vanilla Kernel since 2.6.22**
- **Documents : http://www.linux-mtd.infradead.org/doc/**

  This PPT quoted from below:
  - ubi.html : basic information
  - faq/ubi.html : FAQ
  - ubi.ppt : guidance
  - ubidesign/ubidesign.pdf : detailed information
- **Source Code : git://git.infradead.org/~dedekind/ubi-2.6.git**
- **Mailing list : *linux-mtd@lists.infradead.org***
  - Archive : http://lists.infradead.org/mailman/listinfo/linux-mtd/

UBI had been incorporated into the Vanilla Kernel already (since 2.6.22) & UBIFS will be soon (on 2.6.27)

TOSHIBA

Leading Innovation