
# Managing Data in Flash Memory: A Look Inside

Wanmo Wong
Director of NSG System Software Development
Micron Technology - CA


# Contributors

| Name | Company | Title |
|---|---|---|
| Mark Jahn | Micron | Principal Software Engineer |
| Ralph Gibson | Micron | Staff Firmware Engineer |

# World wide mobile phone, tablet, ultra book, and PC shipments and projected shipments in thousand of units

| | Mobile phone | Tablet | Ultra book | PC |
|---|---|---|---|---|
| 2012 | 1,746,176 | 116,113 | 9,822 | 341,263 |
| 2013 | 1,875,774 | 197,202 | 23,592 | 315,229 |
| 2014 | 1,949,722 | 267,731 | 38,687 | 302,315 |
| 2015 | 2,128,871 | 467,951 | 96,350 | 271,612 |

Source : Gartner (April 2013)

# Sample mobile phone, tablet, ultra book, and PC storage solutions

| | Mobile phone | Tablet | Ultra book | PC |
|---|---|---|---|---|
| Apple | Raw NAND | Raw NAND | SSD | SSD/HDD |
| Samsung | eMMC | eMMC | SSD | SSD/HDD |

Source : iSuppli

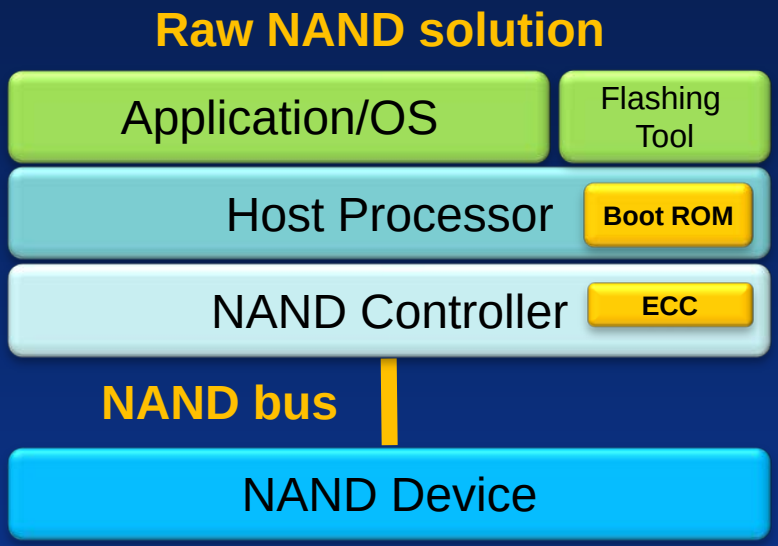
# Sample mobile phone, tablet, ultra book, and PC storage solutions

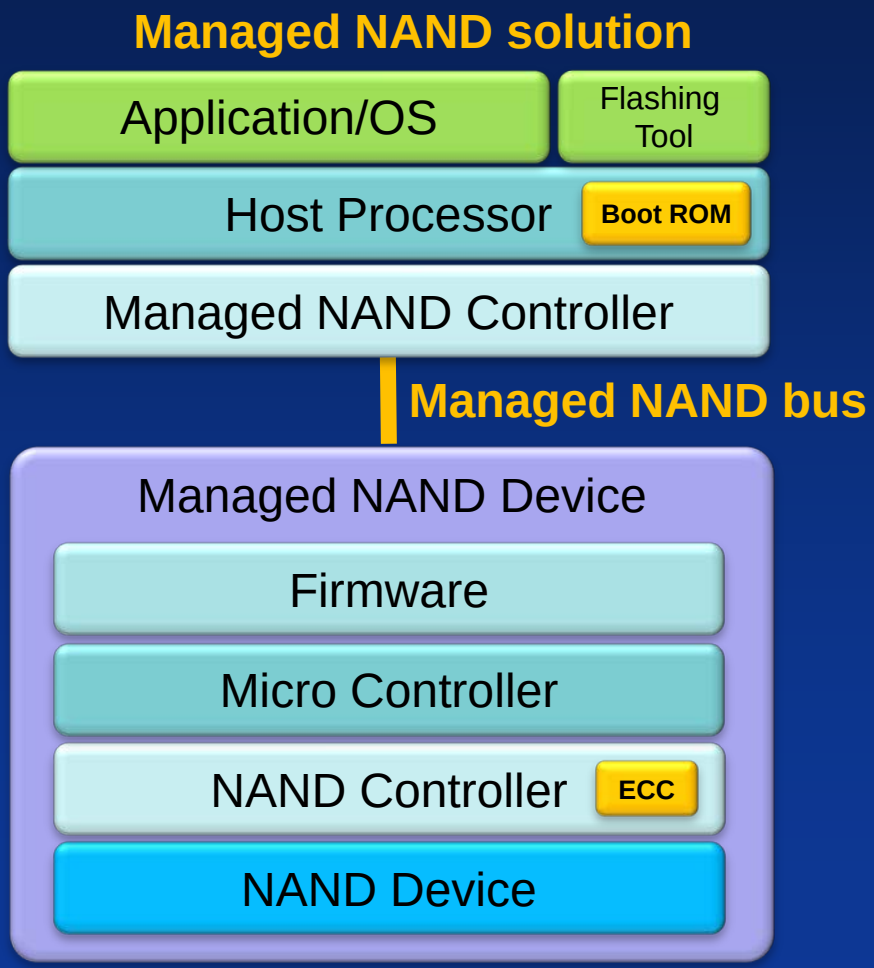
| | Mobile phone | Tablet | Ultra book | PC |
|---|---|---|---|---|
| Apple | Raw NAND | Raw NAND | **SSD** | **SSD**/HDD |
| Samsung | **eMMC** | **eMMC** | **SSD** | **SSD**/HDD |

Source : iSuppli

- **eMMC and SSD use raw NAND flash memory for data storage**
- **Raw NAND flash memory is the core storage solution today**

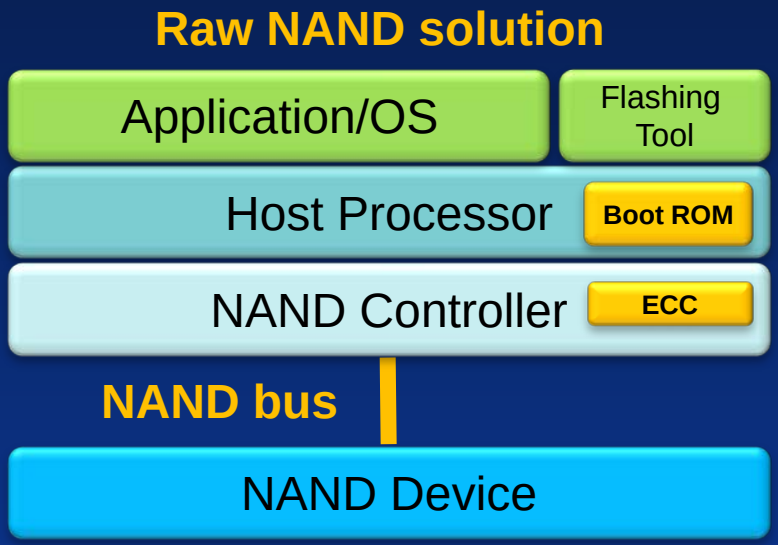
# Raw NAND and managed NAND solutions




# RAW NAND solution considerations

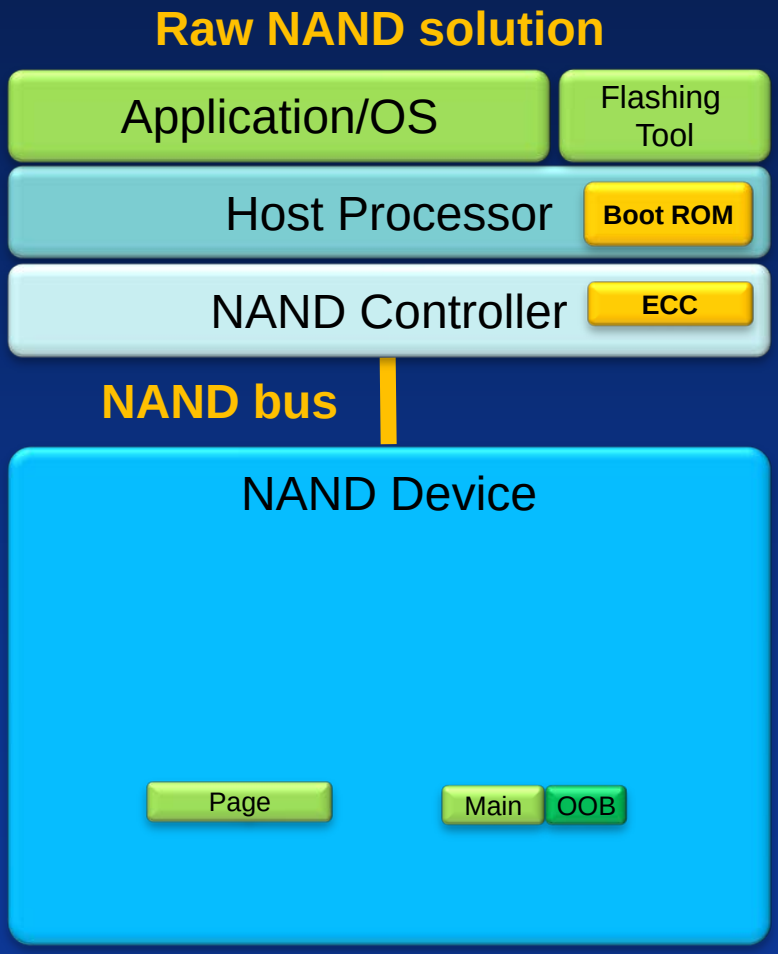
# RAW NAND solution




## Usage Consideration

- Raw NAND architecture
- Bad block management
- Raw NAND capacity
- ECC requirement
- Programming a page
- Reading a page
- Erasing a block
- Raw NAND bus interface
- Raw NAND base level performance
- Raw NAND enhanced performance
- Logical to physical mapping
- Block and Flash based file systems
- Wear leveling
- Power loss recovery
- Flashing and booting

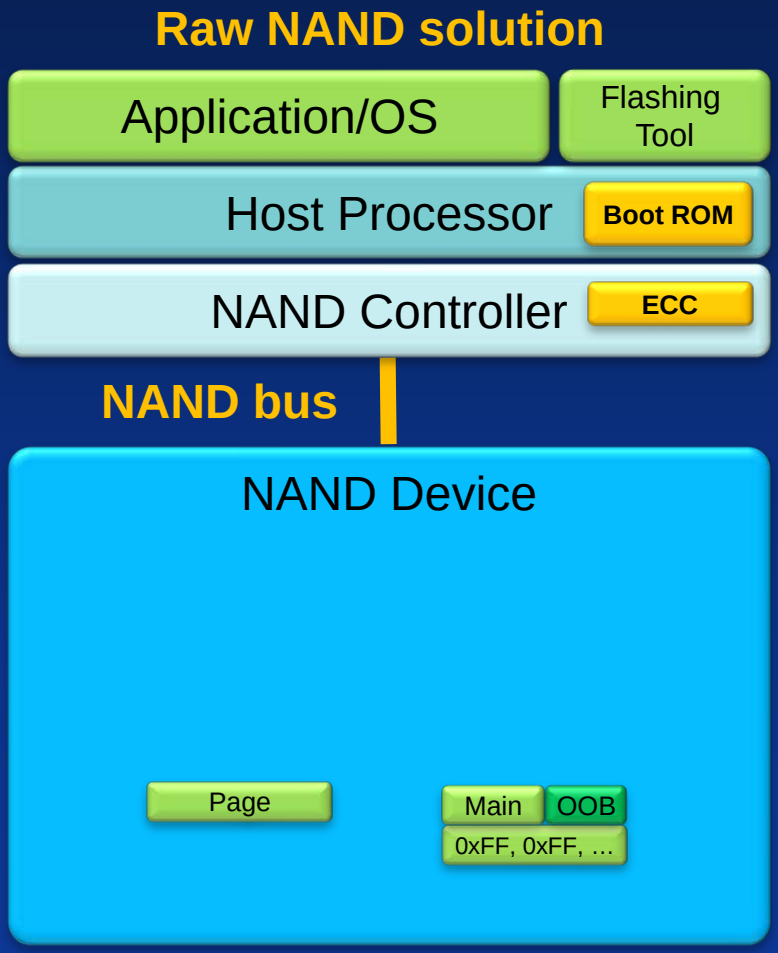
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data

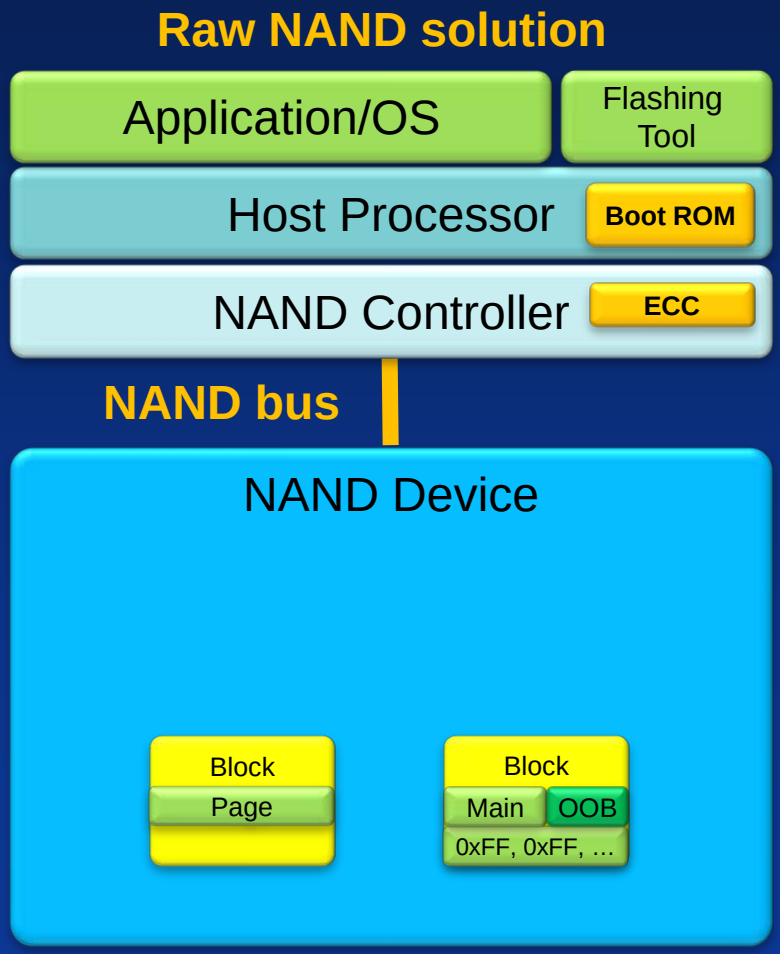
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data

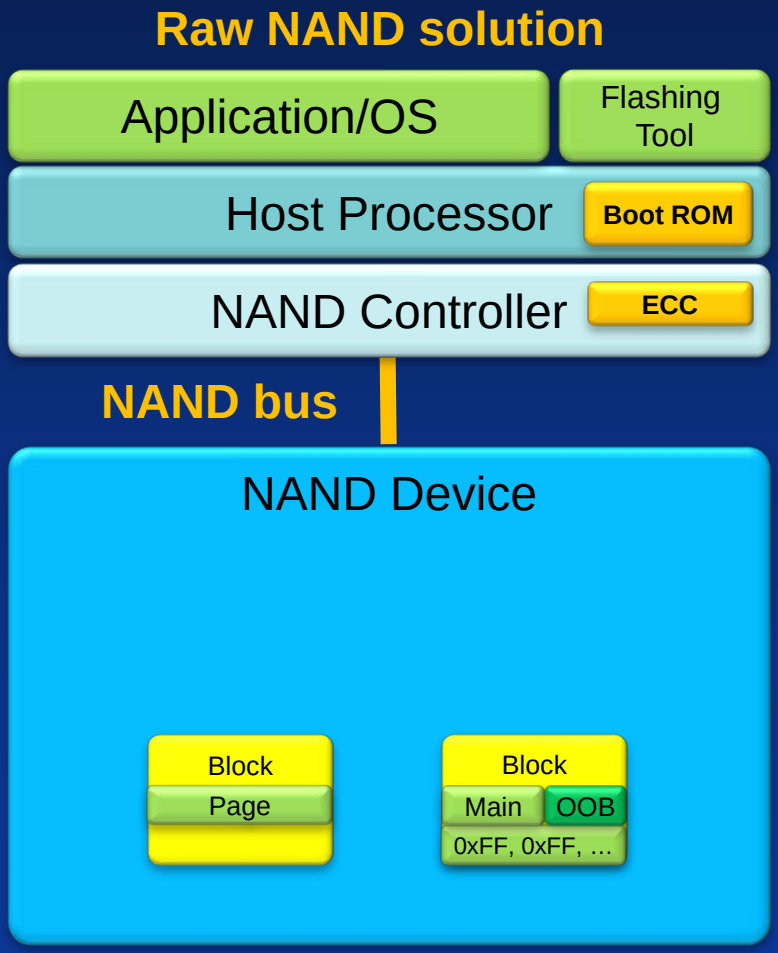
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data
- Group pages into block

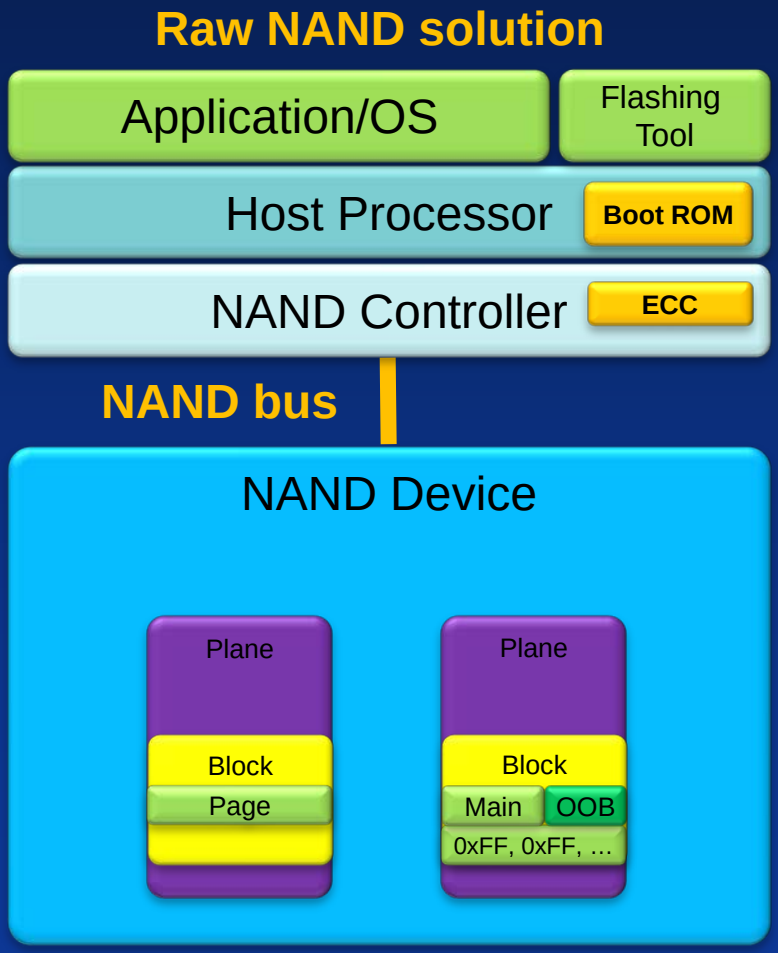
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data
- Group pages into block
  - Store data reliably in page using ECC for block with erase count less than maximum block erase count per data sheet

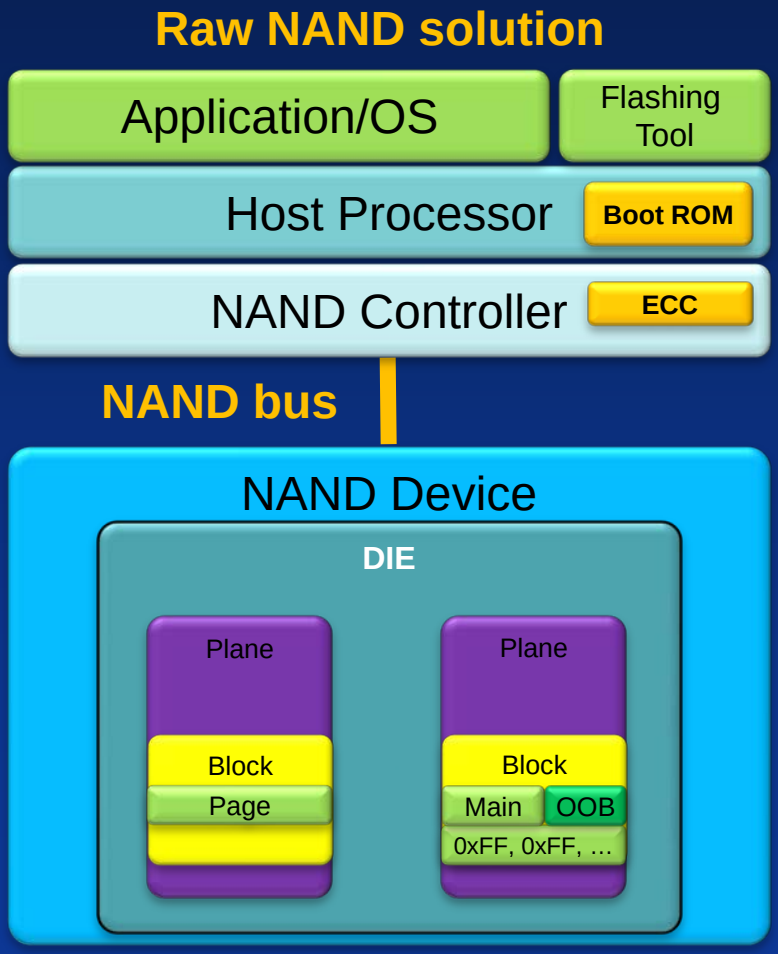
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data
- Group pages into block
  - Store data reliably in page using ECC for block with erase count less than maximum block erase count per data sheet
- Group blocks into plane

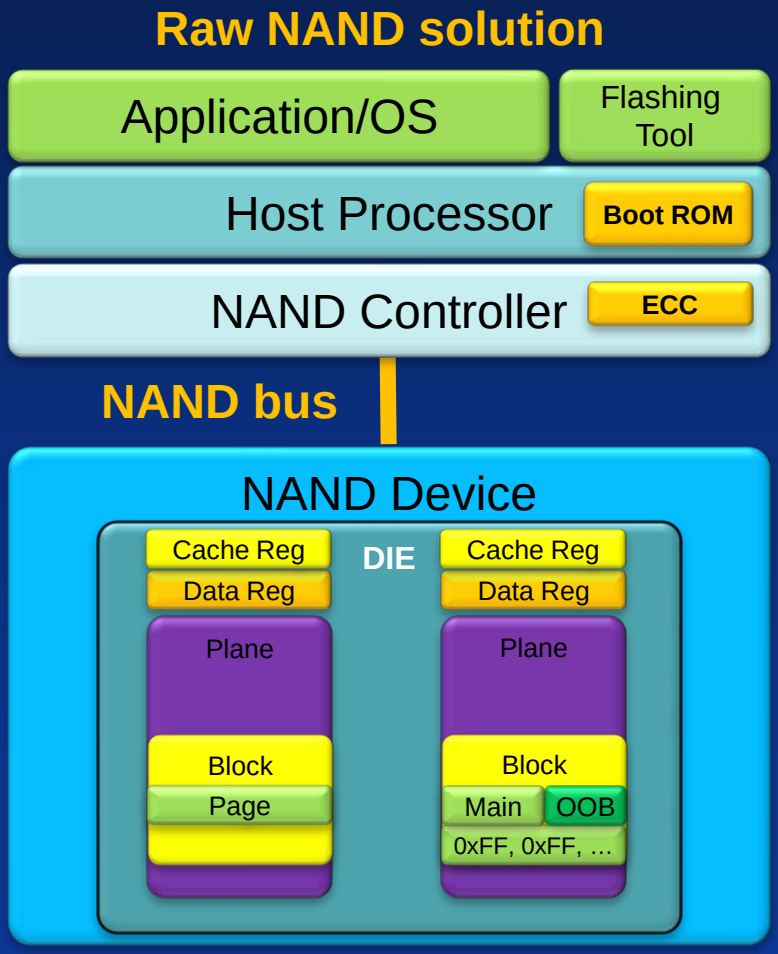
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data
- Group pages into block
  - Store data reliably in page using ECC for block with erase count less than maximum block erase count per data sheet
- Group blocks into plane
- Group planes into DIE (LUN – Logical Unit)

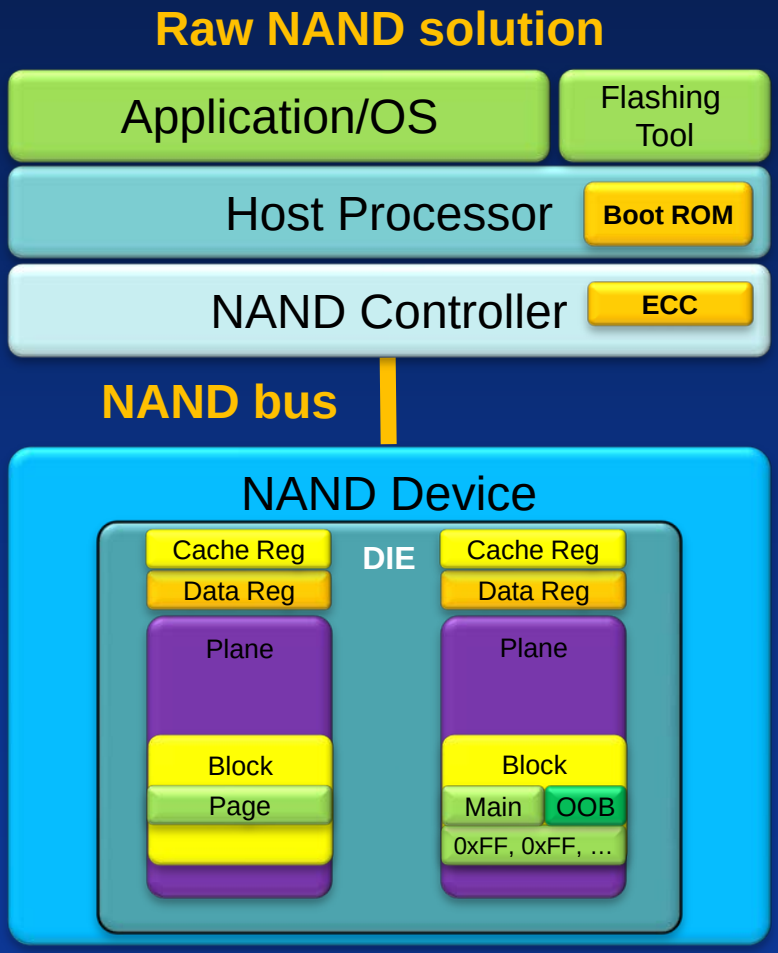
# RAW NAND solution




## Raw NAND architecture

- Store data by page with
  - Main area for user data
  - OOB area for ECC and meta data
- Factory ships pages in erased state
  - Erased state pages have all 0xFF data
- Group pages into block
  - Store data reliably in page using ECC for block with erase count less than maximum block erase count per data sheet
- Group blocks into plane
- Group planes into DIE (LUN – Logical Unit)
  - Each DIE has Cache & Data Reg
  - Reg has the same size as a page

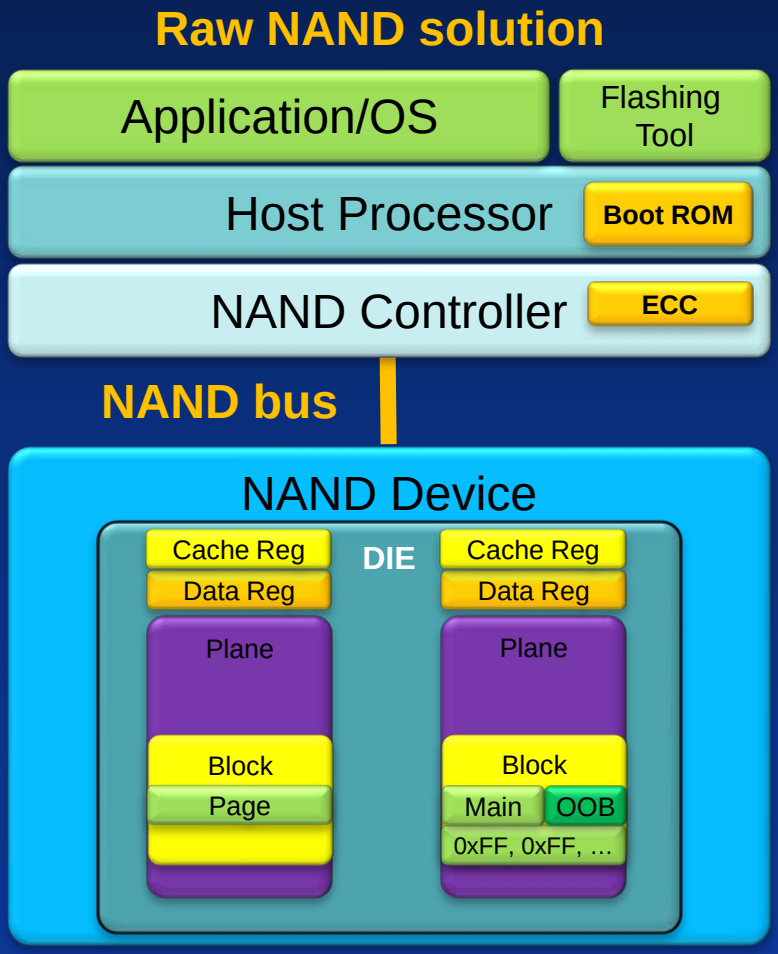
# RAW NAND solution




## Bad block management

- Factory may ship NAND with bad block
  - Actual bad block count is less than maximum bad block count per DIE defined in data sheet
- Detect bad block
  - A common bad block marker is a non 0xFF byte value at the start of OOB
  - Read bad block marker without ECC
  - Bad block marker is vendor unique

# RAW NAND solution




## Bad block management

- Handle run time bad block
  - Block with programming error
    - Save all valid pages before the page with programming error to another block
    - Mark block as bad block
  - Block with erasing error
    - Mark block as bad block
- Do not use bad block for data storage

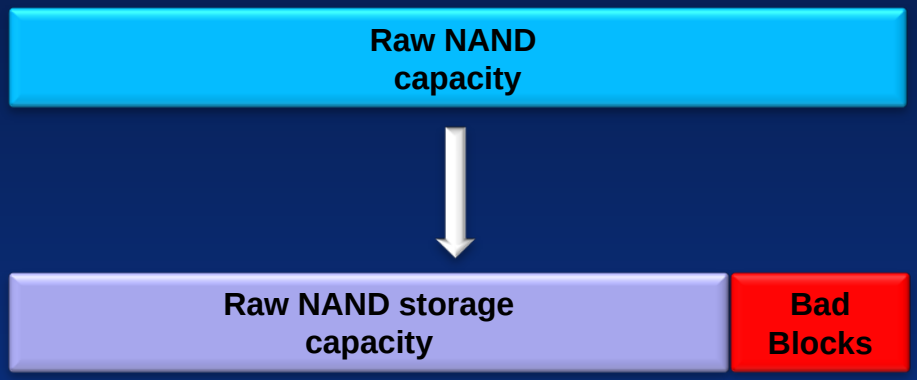
# RAW NAND solution

**Raw NAND capacity**

**Raw NAND capacity**

- **Raw NAND capacity is the total capacity of blocks in a raw NAND storage system including bad blocks**

# RAW NAND solution

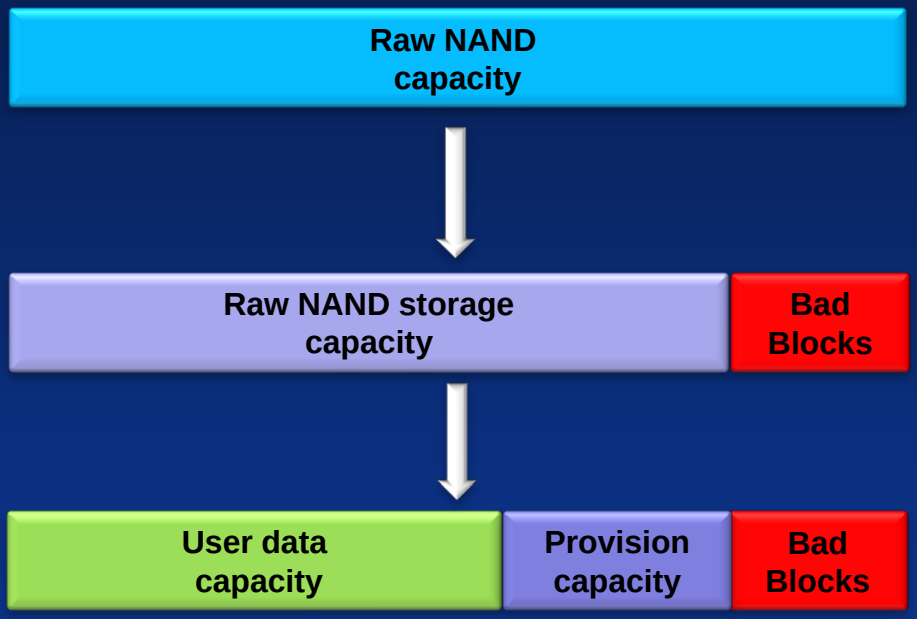



**Raw NAND capacity**

- **Raw NAND capacity is the total capacity of blocks in a raw NAND storage system including bad blocks**

- **Raw NAND storage capacity is the total capacity of good blocks in a raw NAND storage system**

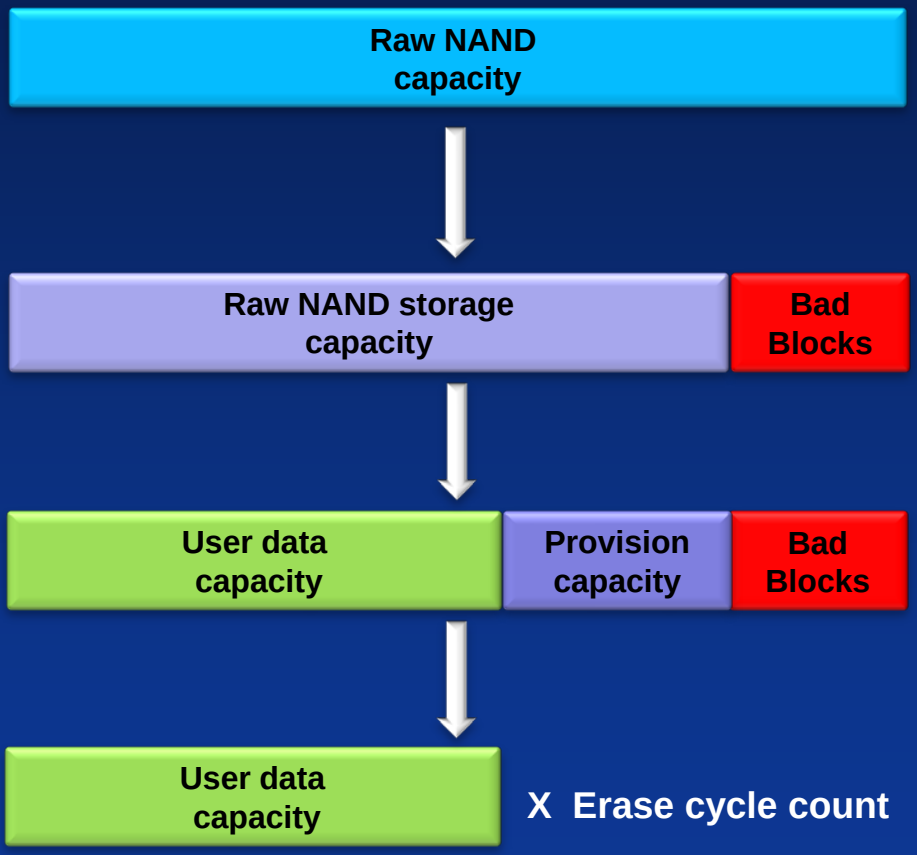
# RAW NAND solution




**Raw NAND capacity**

- **Raw NAND capacity is the total capacity of blocks in a raw NAND storage system including bad blocks**
- **Raw NAND storage capacity is the total capacity of good blocks in a raw NAND storage system**
- **Provision capacity is the storage for provisional user data and meta data**
- **User data capacity is the capacity of user data storage for the life of the storage system**

# RAW NAND solution

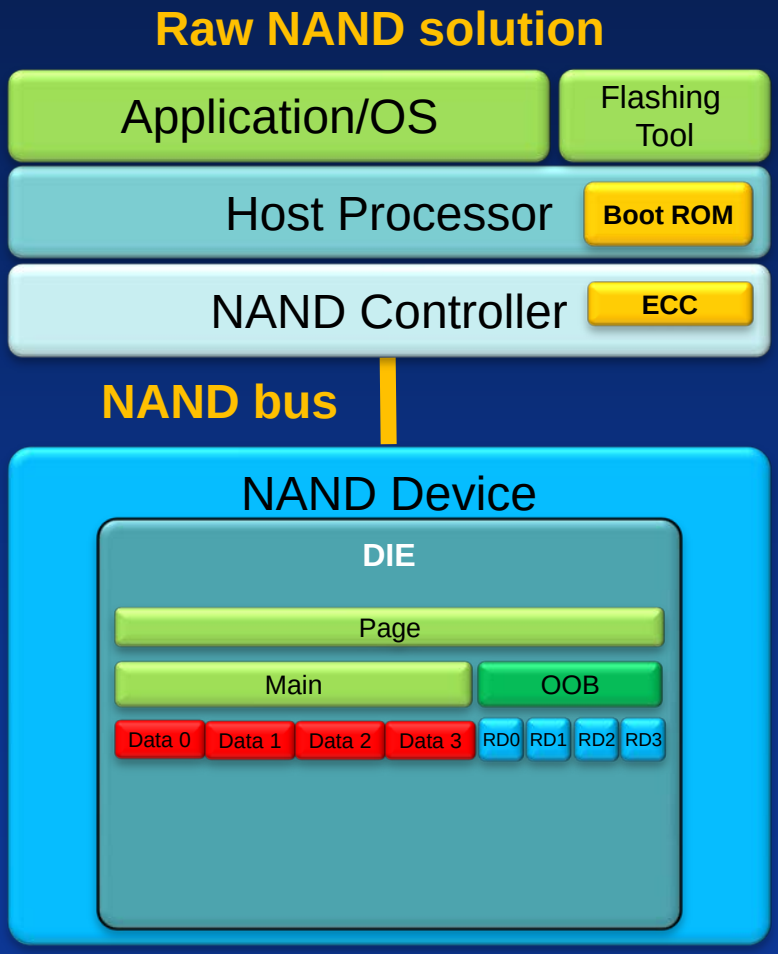



**Raw NAND capacity**

- **User data programming capacity is user data capacity X erase cycle count**

- **Life of storage system in days is user data programming capacity / number of bytes written per day**

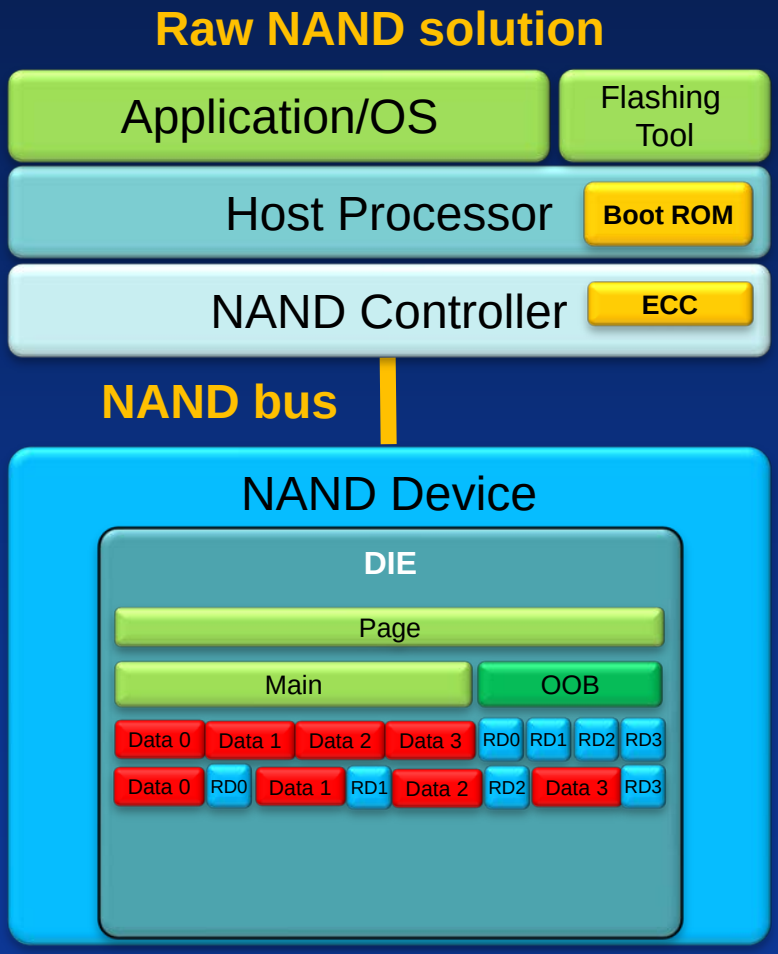
# RAW NAND solution




## ECC requirement

- NAND delivers reliable data storage in a page applying required ECC for block with erase cycle count less than or equal to maximum erased cycle count per data sheet
- Data sheet defines ECC requirement as a number of correctable bits per a number of bytes
- The number of bytes includes programming data and the ECC redundancy data (RD) for the programming data
- Software runs the ECC engine with the programming data to generate the ECC redundancy data

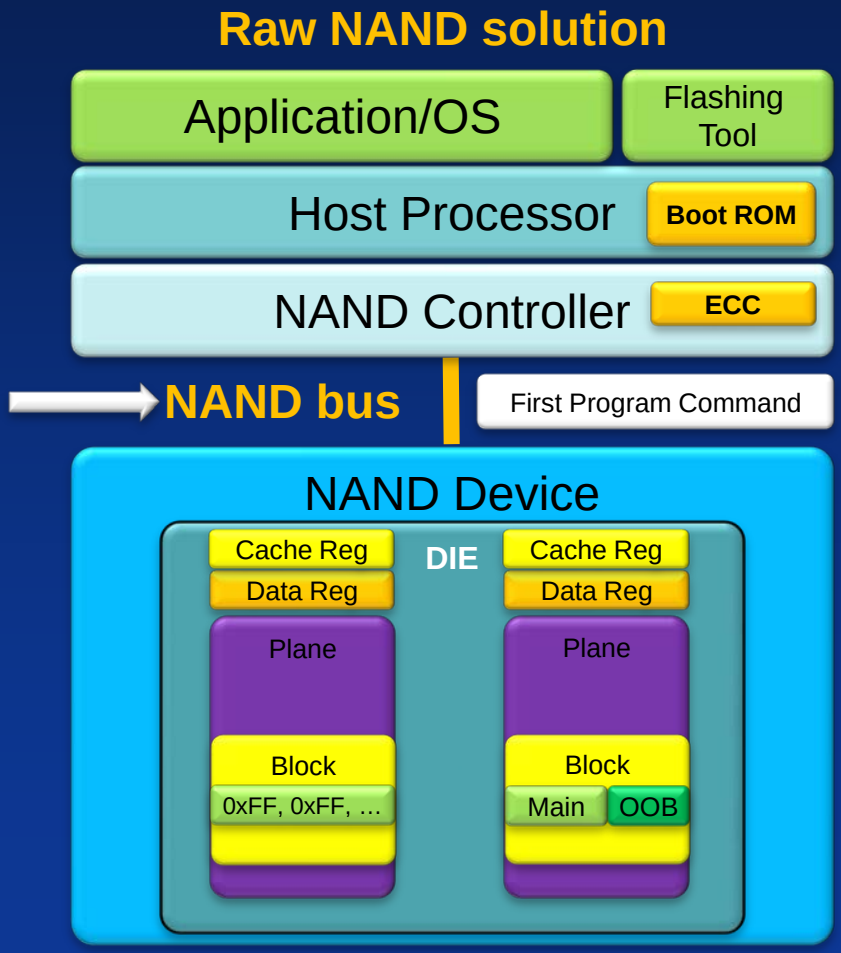
# RAW NAND solution




## ECC requirement

- **The combination of programming data and redundancy data must fit in a page**
- **Software programs data and redundancy data in the destination page**
- **Location of programming data and redundancy data are platform dependent**

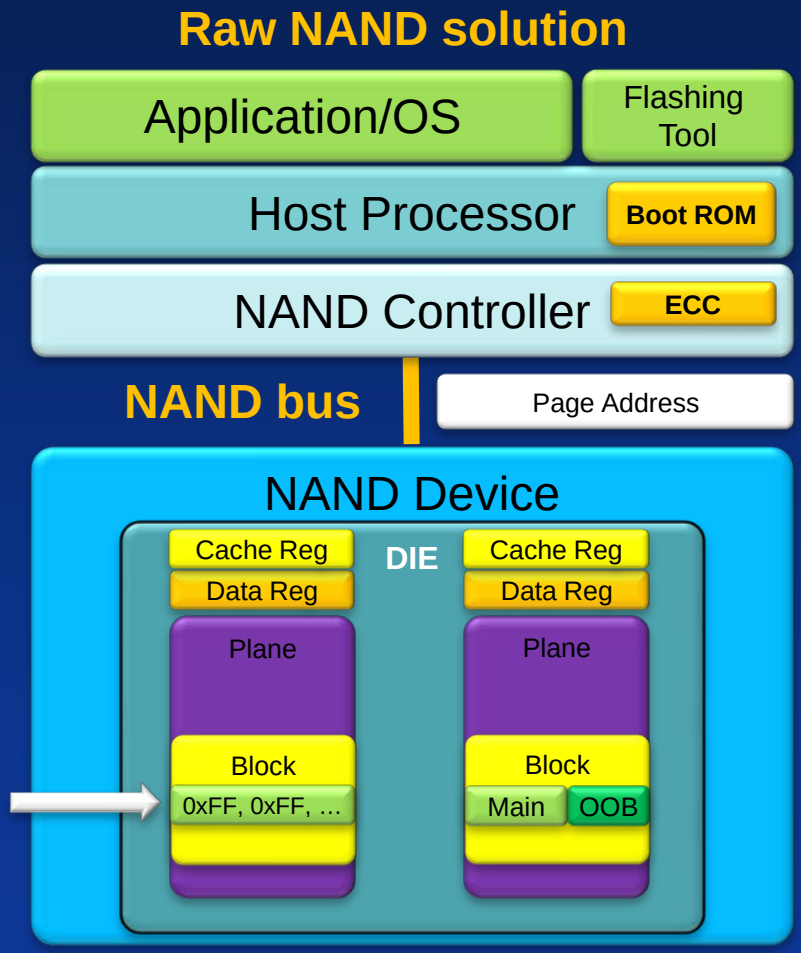
# RAW NAND solution




**Programming a page**

- **Send first program command**

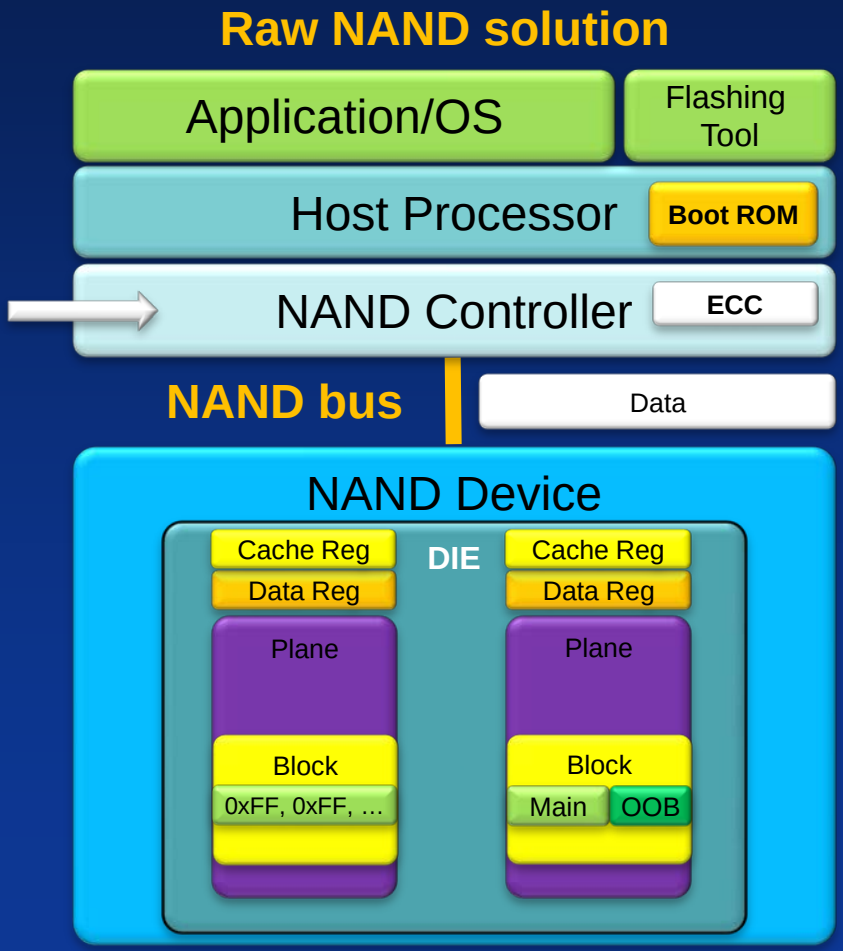
# RAW NAND solution




**Programming a page**

- **Send first program command**
- **Send page address to program**

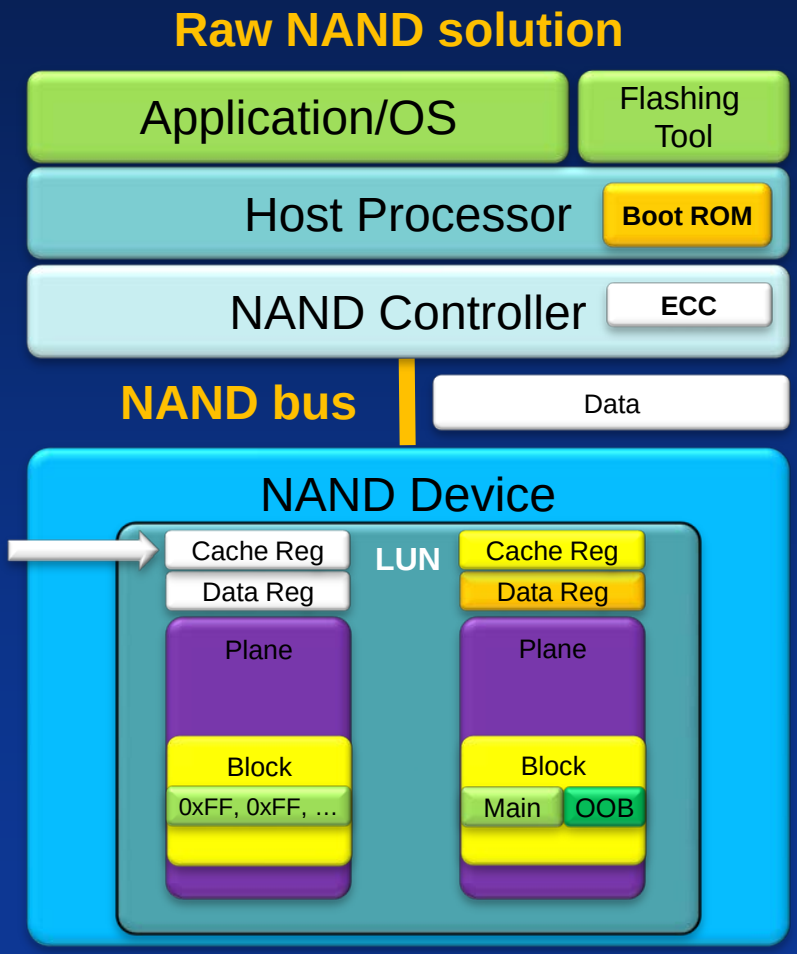
# RAW NAND solution




**Programming a page**

- **Send first program command**
- **Send page address to program**
- **Clock in data to program**

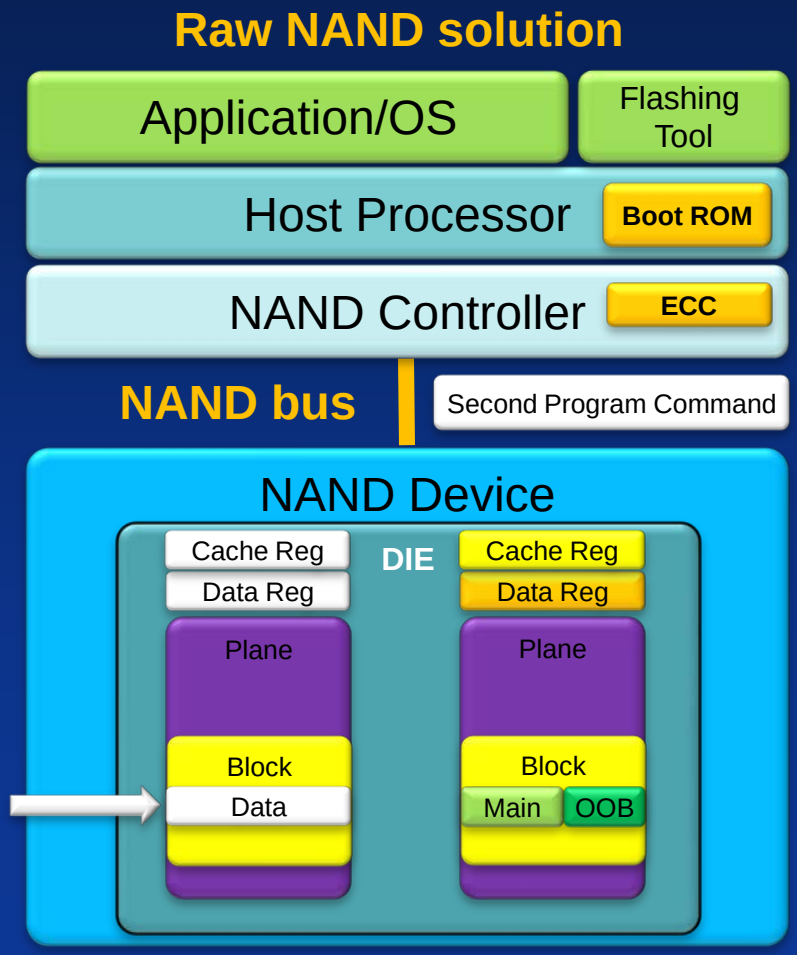
# RAW NAND solution




**Programming a page**

- **Send first program command**
- **Send page address to program**
- **Clock in data to program**
- **Clock in ECC RD of programmed data**
- **Data arrives at Cache and Data Reg**

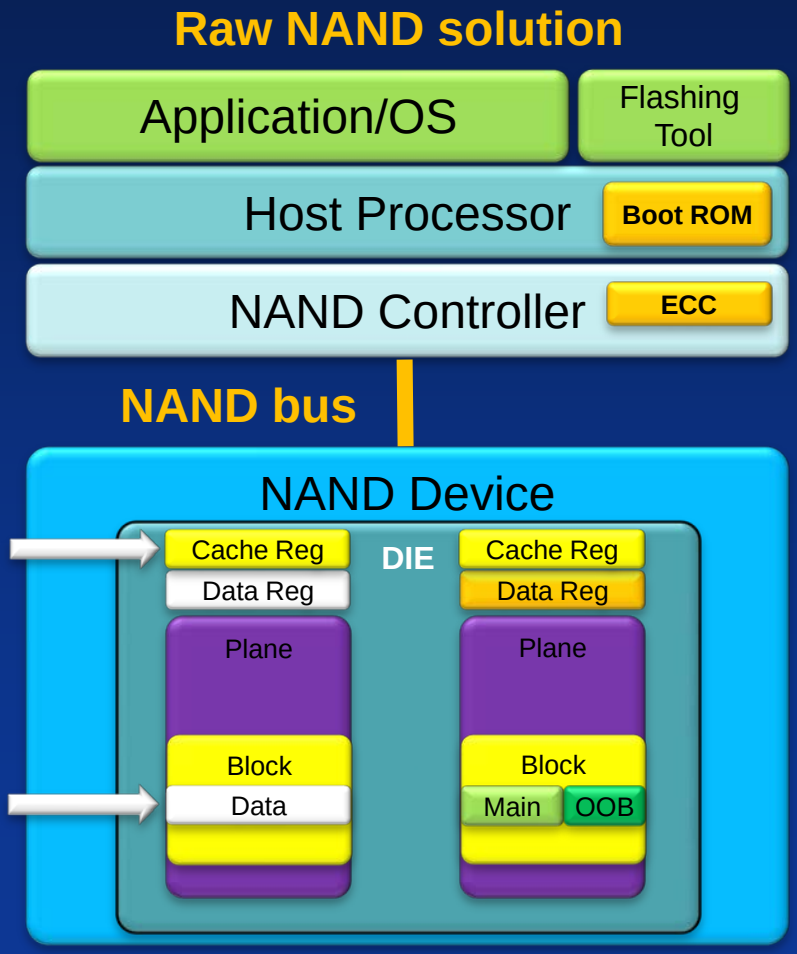
# RAW NAND solution




## Programming a page

- Send second program command
  - Program data from Data Reg to page
  - The device goes busy until programming is done
  - The busy time is the latency for programming a page

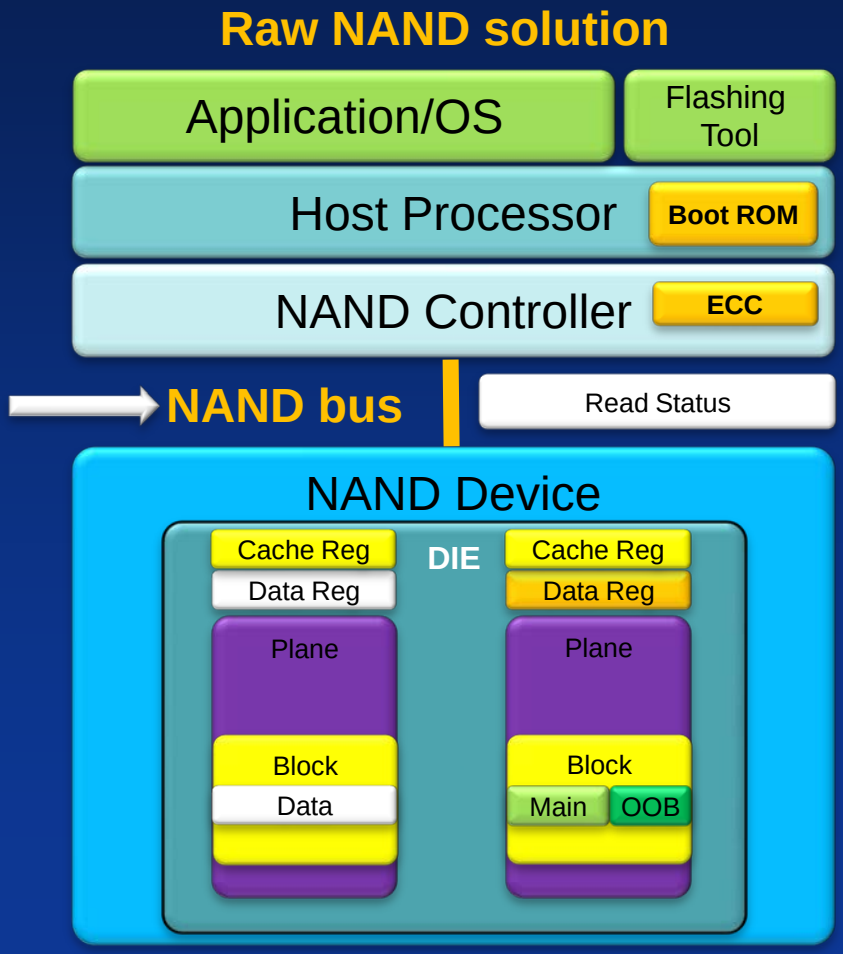
# RAW NAND solution




**Programming a page**

- **Send second program command**
  - **Program data from Data Reg to page**
  - **The device goes busy until programming is done**
  - **The busy time is the latency for programming a page**
  - **Cache Reg is available to clock in the next page of programming data overlapping the Data Reg to improve NAND page programming time**

# RAW NAND solution

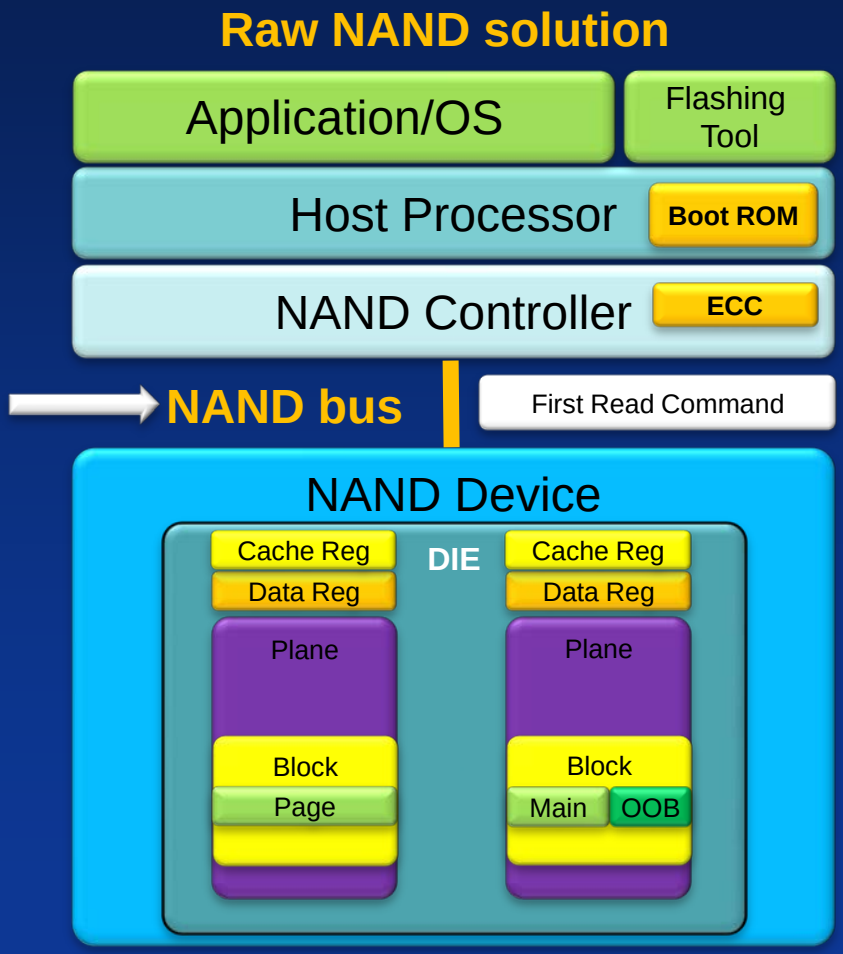



**Programming a page**

- **Send second program command**
  - **Program data from Data Reg to page**
  - **The device goes busy until programming is done**
  - **The busy time is the latency for programming a page**
  - **Cache Reg is available to clock in the next page of programming data overlapping the Data Reg to improve NAND page programming time**

- **Send read status command and keep reading status until device is ready**

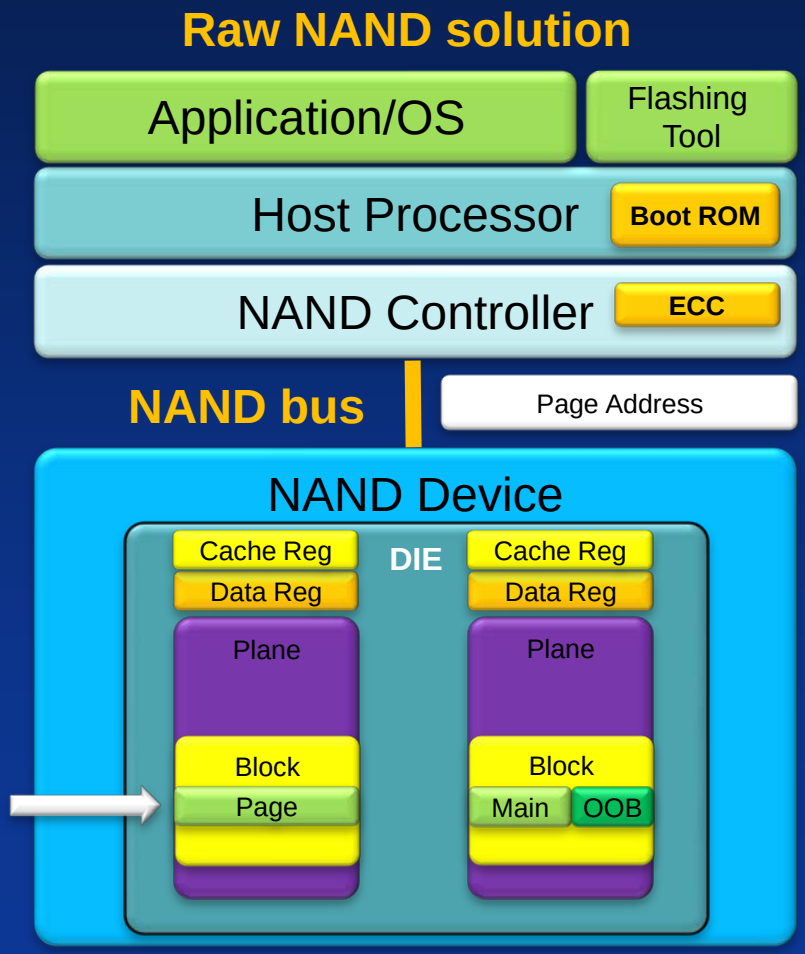
# RAW NAND solution




## Reading a page

- **Send first read command**

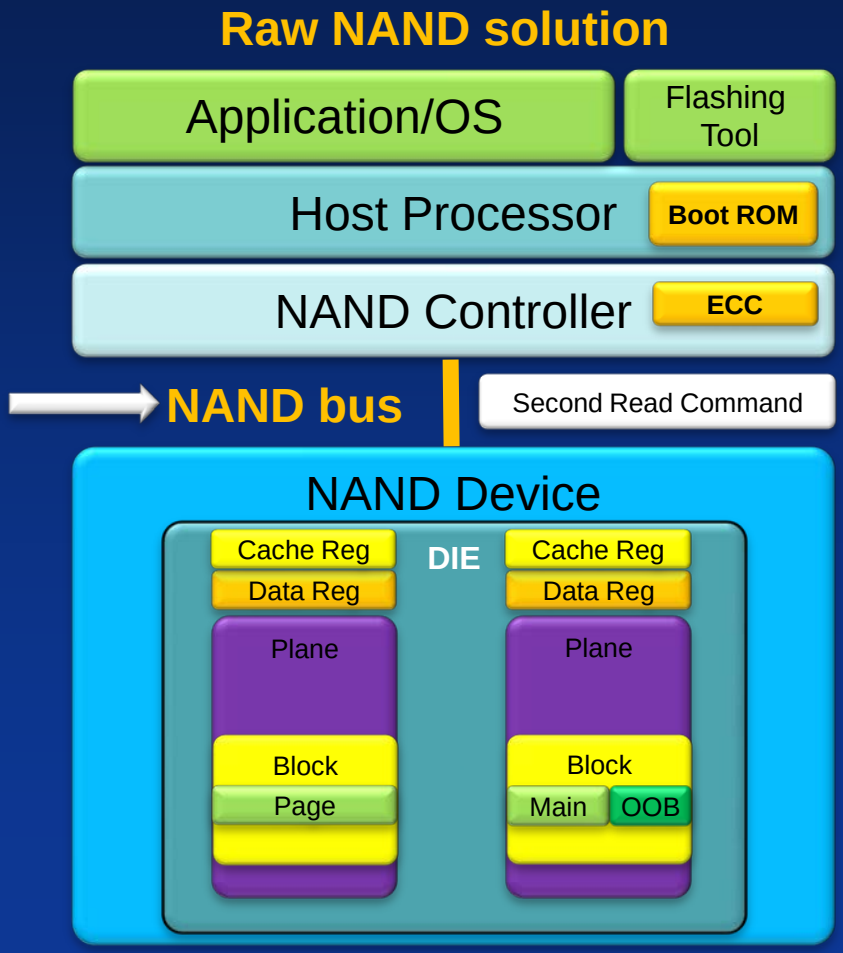
# RAW NAND solution




**Reading a page**

- **Send first read command**
- **Send page address to read**

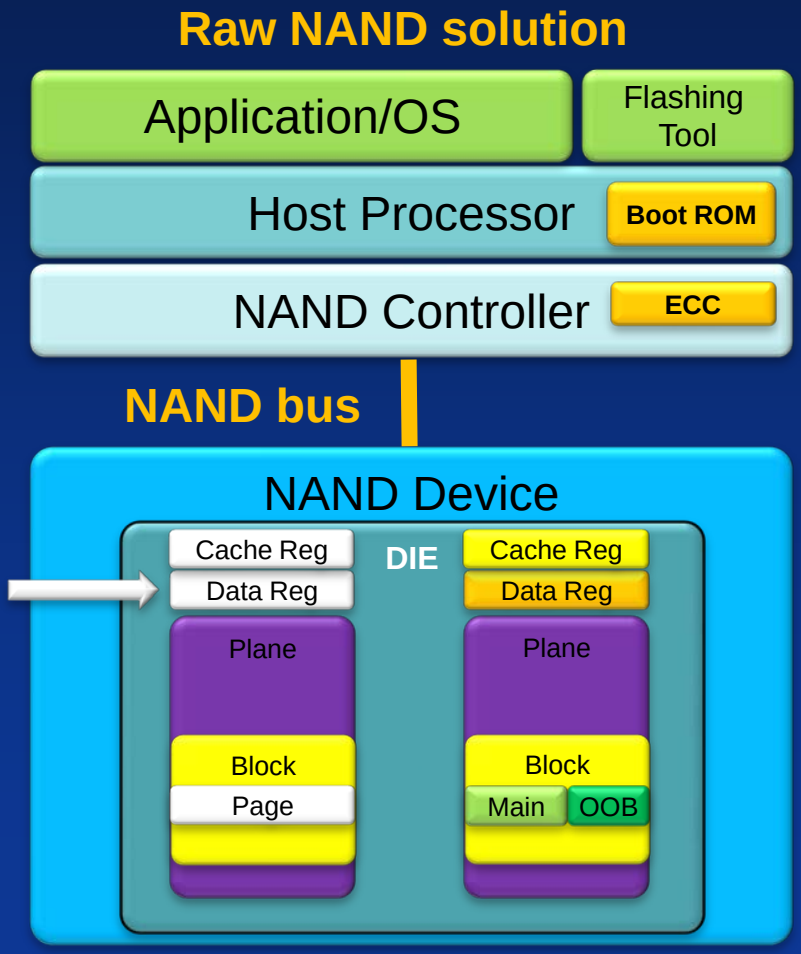
# RAW NAND solution




## Reading a page

- **Send first read command**
- **Send page address to read**
- **Send second read command**

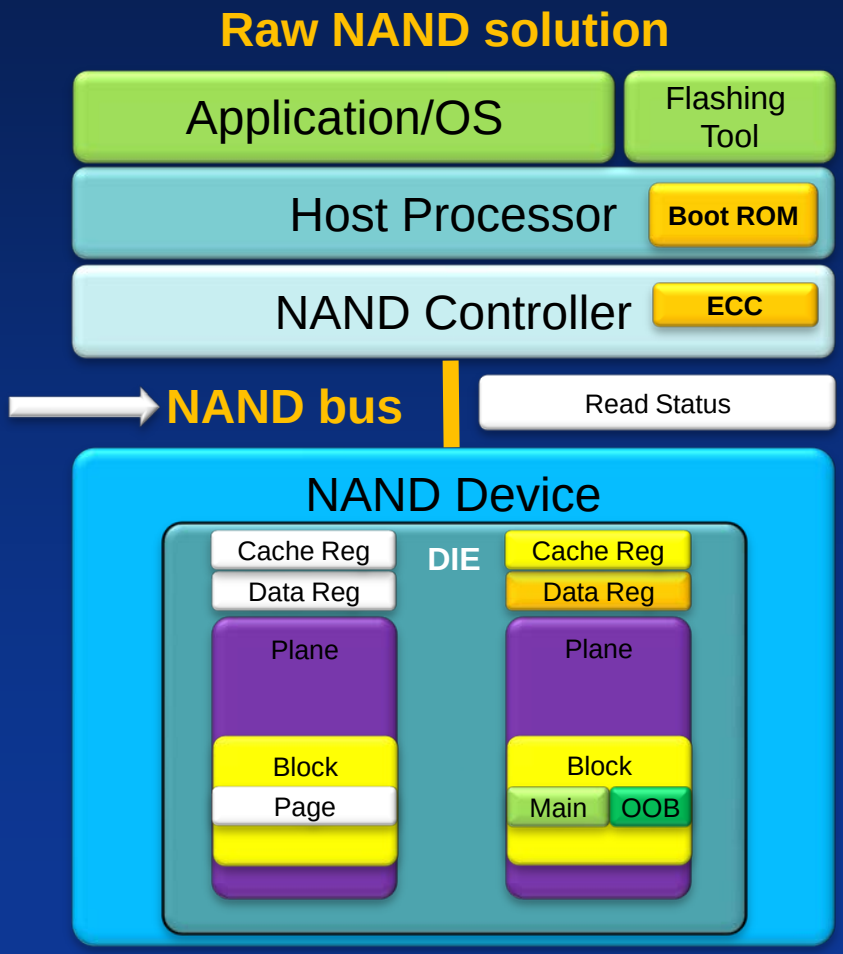
# RAW NAND solution




## Reading a page

- Send first read command
- Send page address to read
- Send second read command
  - NAND copies data from page to Data Reg
  - NAND copies data from Data Reg to Cache Reg
  - The device goes busy until copy is done
  - The busy time is the latency for reading a page

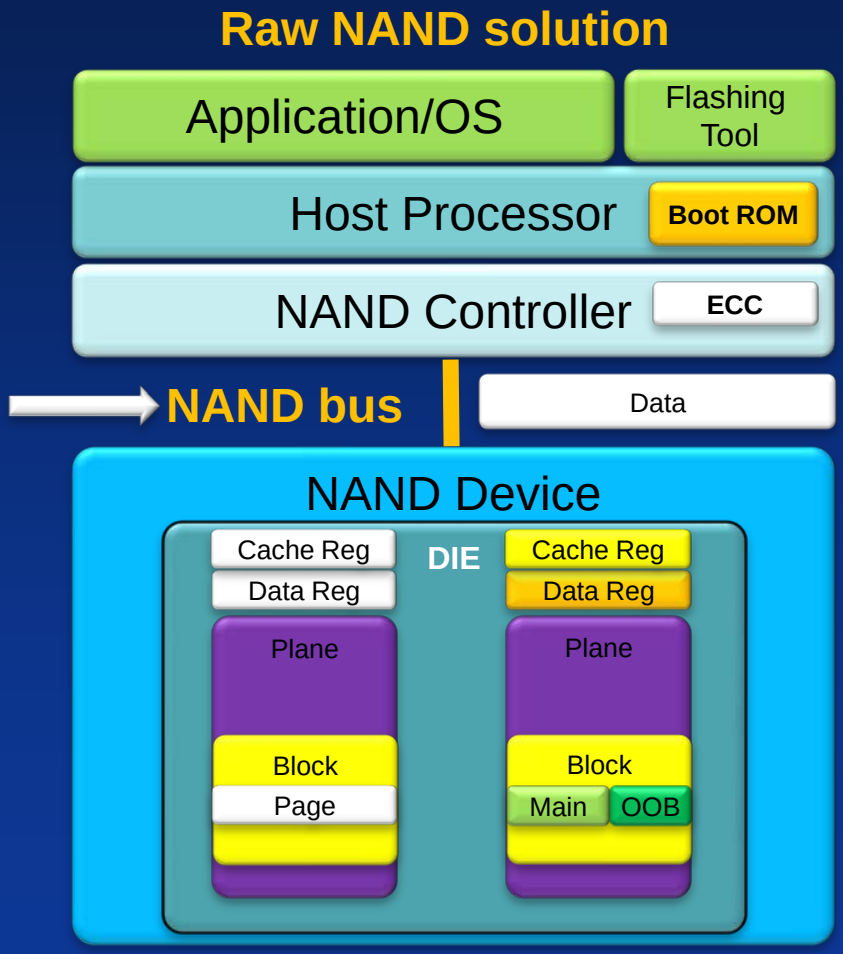
# RAW NAND solution




**Reading a page**

- **Send read status command and read status until device is ready**

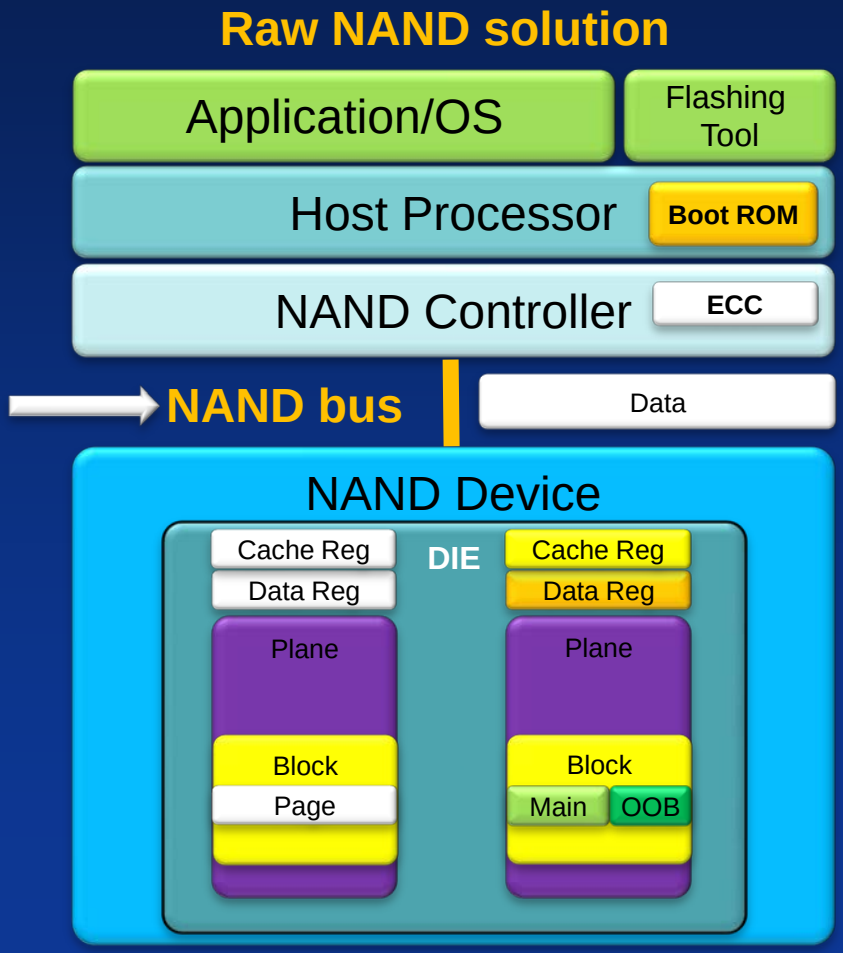
# RAW NAND solution




**Reading a page**

- **Send read status command and read status until device is ready**
- **When status is ready, clock data out from Cache Reg**
- **Use ECC engine to detect, report, and fix errors if necessary**
- **If there is no ECC error, return the read data to the caller**
- **If ECC was able to correct all errors, return data with ECC correction to the caller**

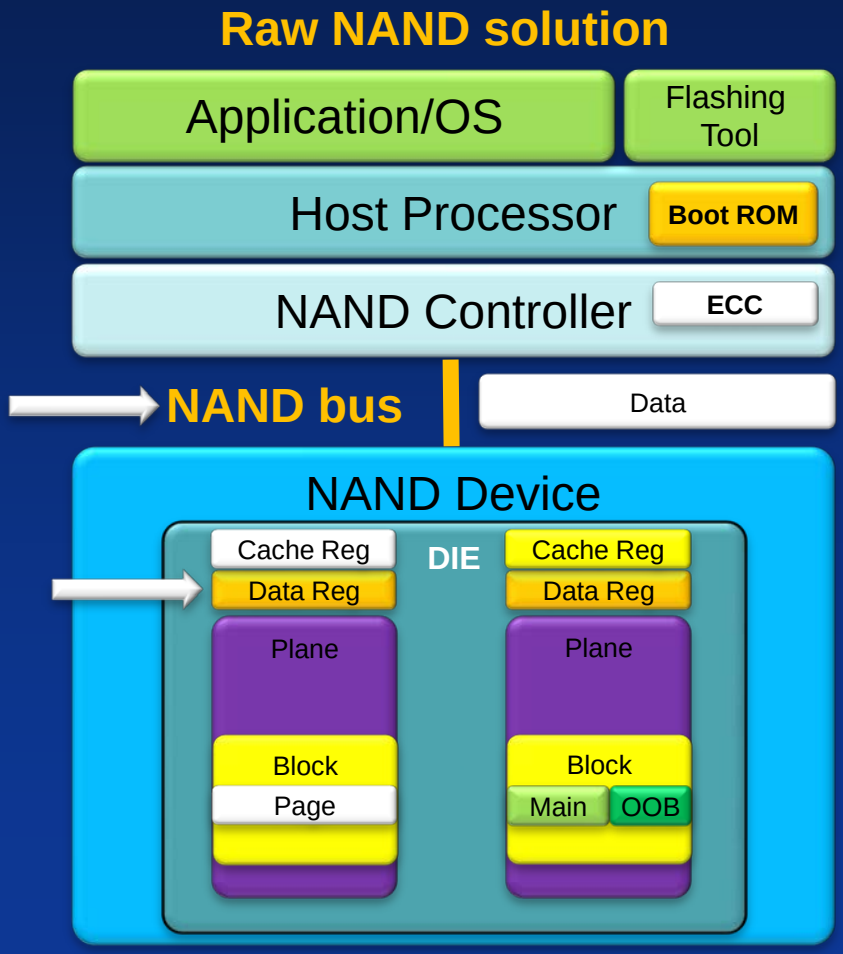
# RAW NAND solution




## Reading a page

- If ECC was unable to fix all errors
  - Return error to caller if there is no read retry support or no next read retry option
  - If there is next read retry option
    - Enable next read retry option
    - Retry the read request

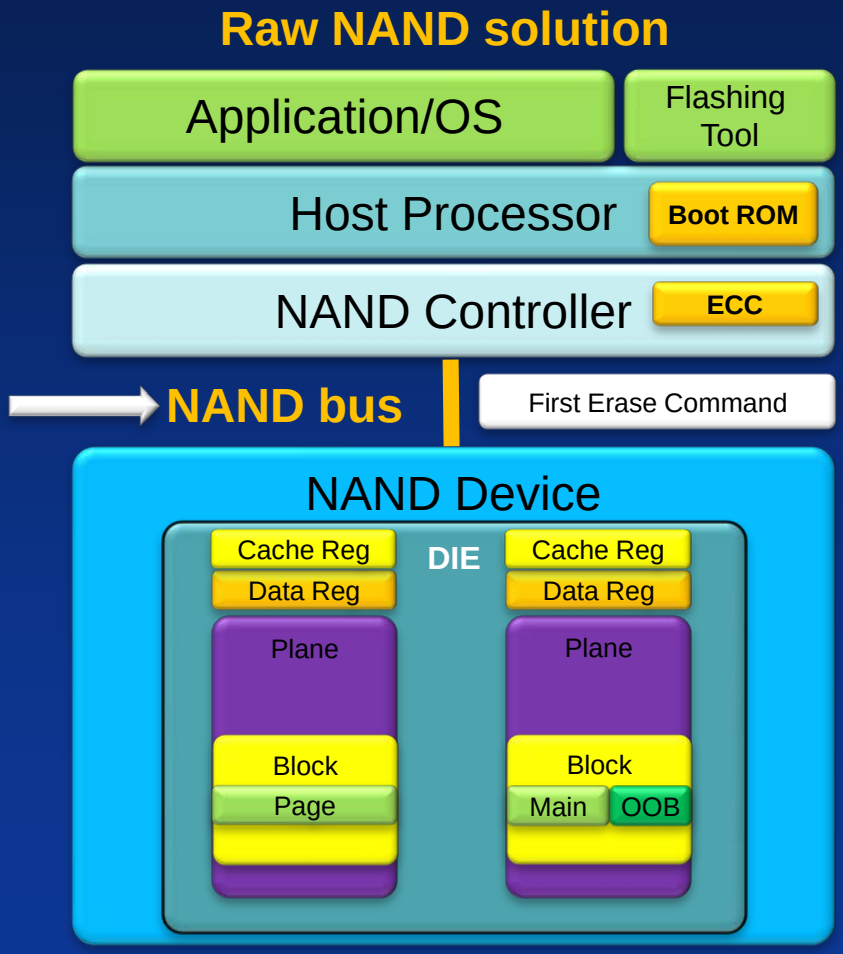
# RAW NAND solution




## Reading a page

- As the application is clocking out the data from the Cache Reg, Data Reg is available for the next read operation

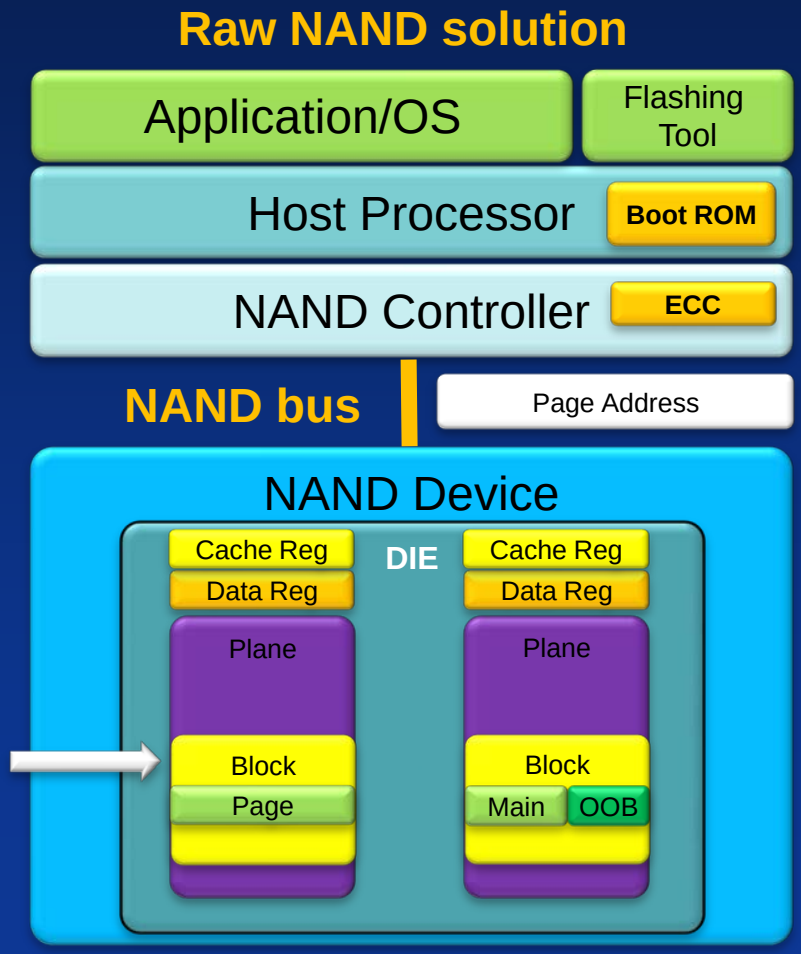
# RAW NAND solution




## Erasing a block

- **Send first erase command**

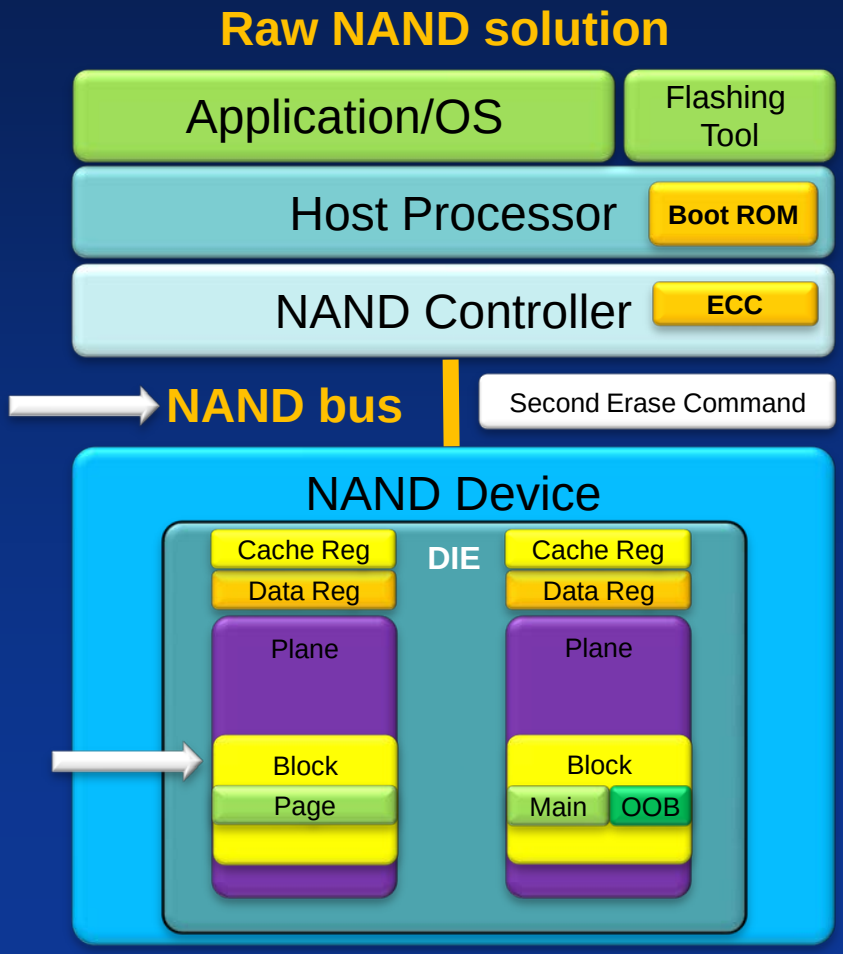
# RAW NAND solution




## Erasing a block

- **Send first erase command**
- **Send page address of block to erase**

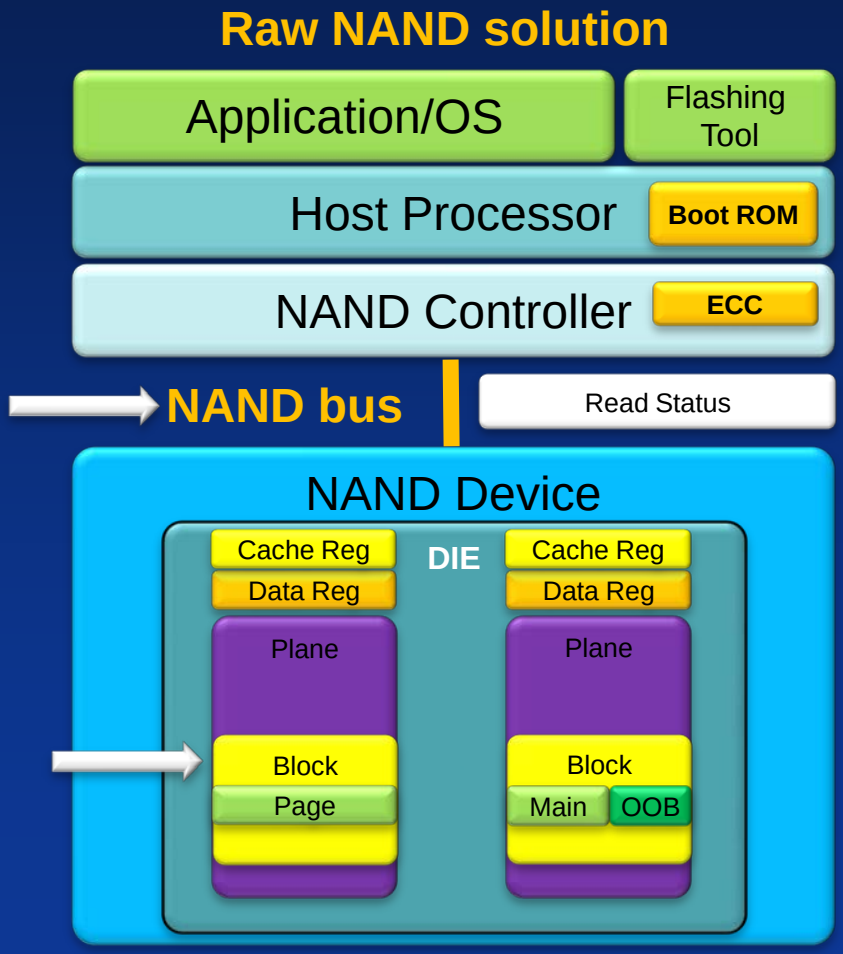
# RAW NAND solution




## Erasing a block

- Send first erase command
- Send page address of block to erase
- Send second erase command
  - This command starts erasing the selected block
  - The device will go busy until the erase operation is done
  - The busy time is the block erase latency time

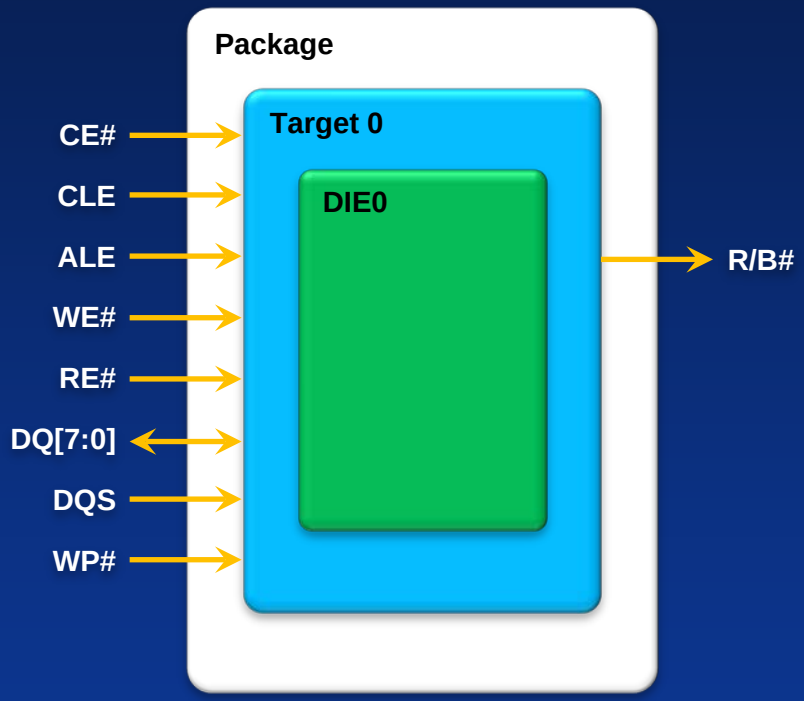
# RAW NAND solution




## Erasing a block

- Send first erase command
- Send page address of block to erase
- Send second erase command
  - This command starts erasing the selected block
  - The device will go busy until the erase operation is done
  - The busy time is the block erase latency time
- Send read status command and read status until device is ready

# RAW NAND solution

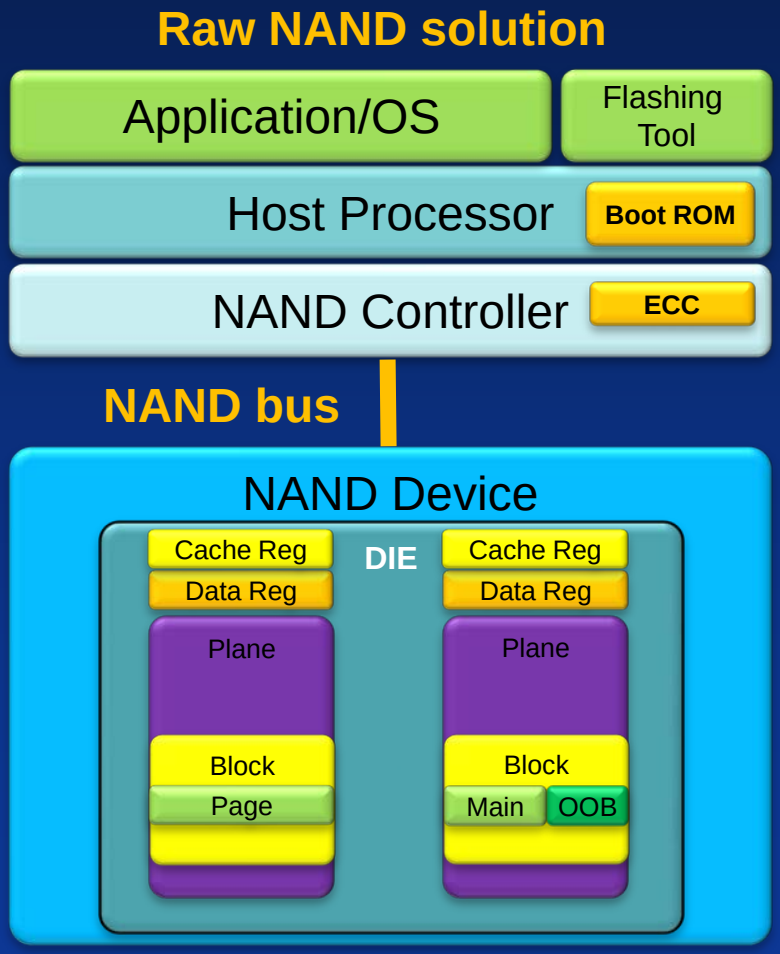



## NAND bus interface

- Runs in asynchronous and/or synchronous mode
- Synchronous mode supports Double Data Rate (DDR) operation
- Multiplexed bus with command, data, and address for low pin count solution
- Supports hardware and software ready/busy detection

CE# = Chip Enable, CLE = Command Latch Enable, ALE = Address Latch Enable, WE# = Write Enable, RE# = Read Enable, DQ = Data Inputs / Outputs, DQS = Data Strobe Signal, WP# = Write Protect, R/B# = Ready Busy

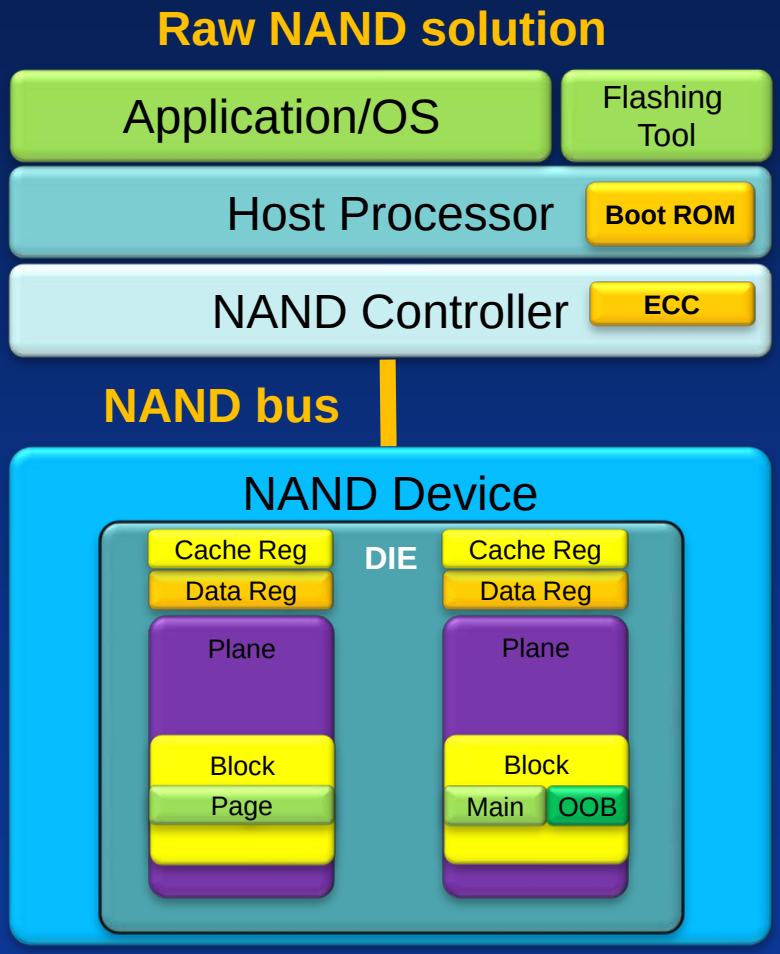
# RAW NAND solution




**Raw NAND base level performance**

- **Based on IO time + Latency**
  - **IO time depends on bus speed**
- **Programming**
  - **IO time is the clocking in time**
  - **Latency is the programming time**
- **Reading**
  - **IO time is the clocking out time**
  - **Latency is the reading time**
- **Erasing**
  - **IO time is none**
  - **Latency is the erasing time**

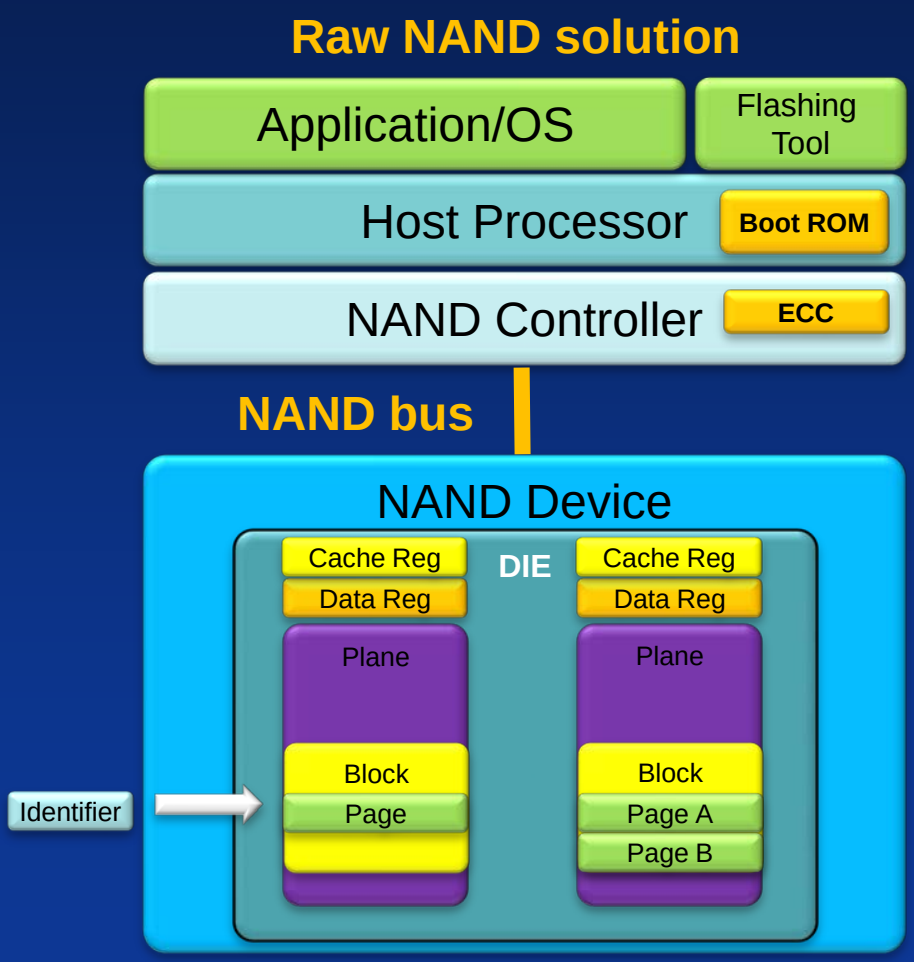
# RAW NAND solution




**Raw NAND enhanced performance**

- **Cache operation**
  - **Overlap IO time for programming**
  - **Overlap latency for reading**
- **Multiple plane operation**
  - **Overlap latency for programming, reading, and erasing**
- **Synchronous bus interface**
  - **Reduce IO time**
- **Multiple channel**
  - **Overlap IO time and latency**

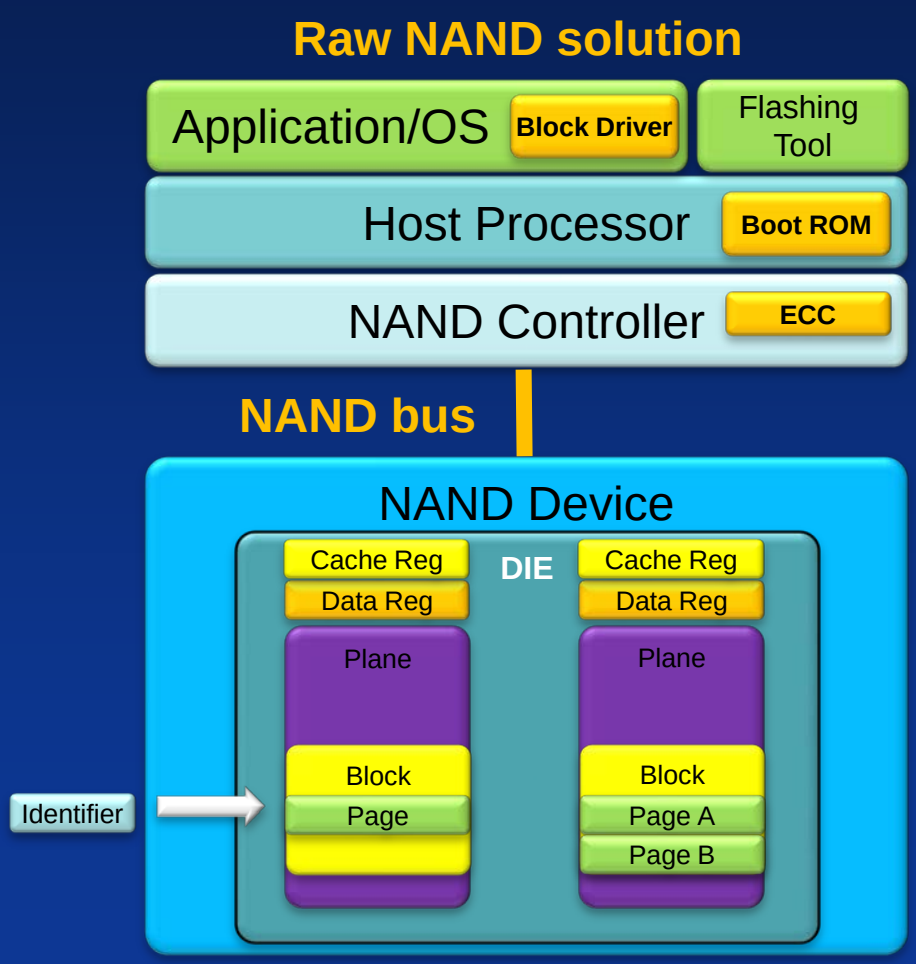
# RAW NAND solution




**Logical to physical (L2P) mapping**

- **Application uses unique identifier to find stored data**
- **The unique identifier is a logical key**
- **The process of mapping data storage address with a logical key is logical to physical (L2P) mapping**

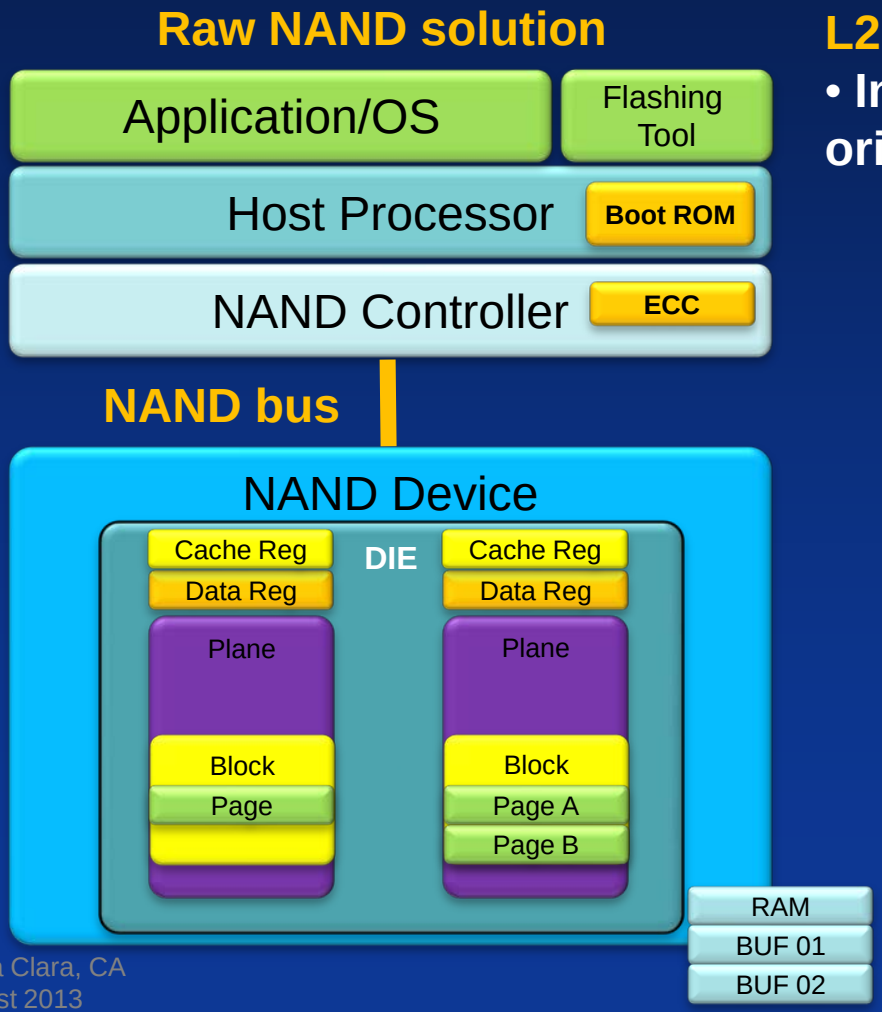
# RAW NAND solution




**Logical to physical (L2P) mapping**

- **Application uses unique identifier to find stored data**
- **The unique identifier is a logical key**
- **The process of mapping data storage address with a logical key is logical to physical (L2P) mapping**
- **L2P mapping is usually implemented in a block driver**
- **Block driver provides application with logical sector reading and writing functions with raw NAND device**

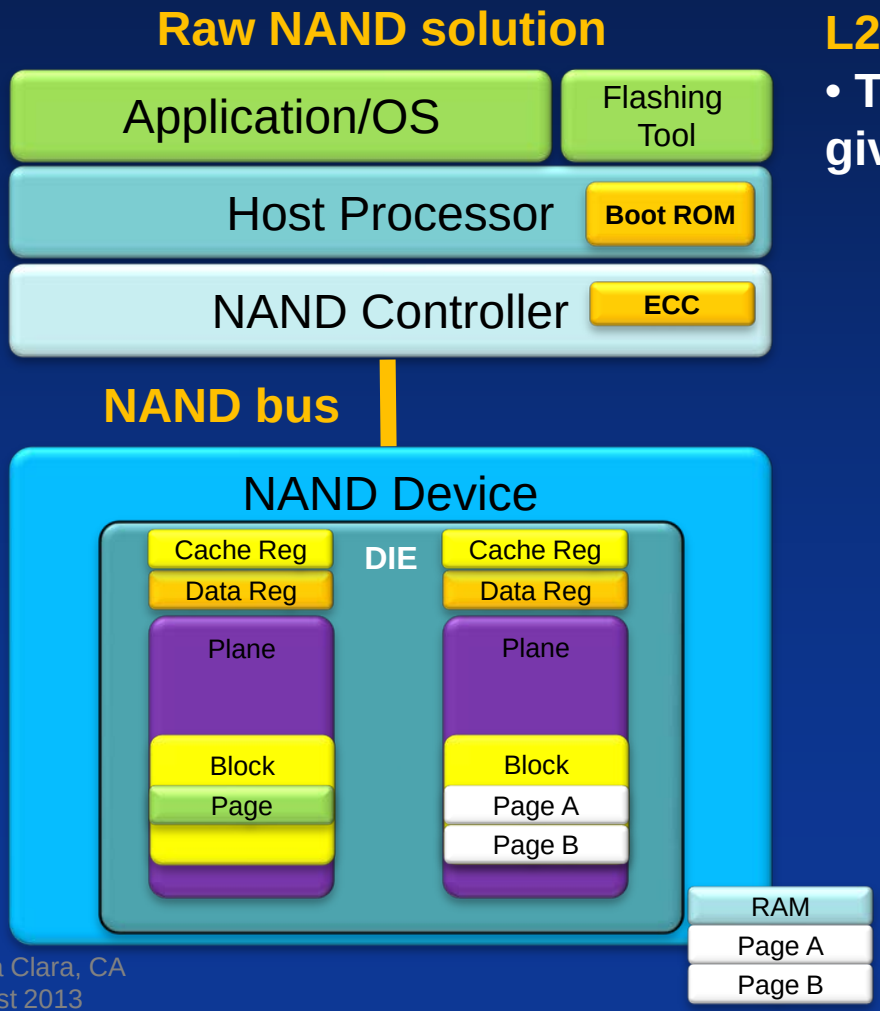
# RAW NAND solution




**L2P mapping - in place update**

- **In place update updates data at the original storage location**
- **RAM and HDD storage media perform in place update**
- **An example is to change the value of a memory location in RAM**

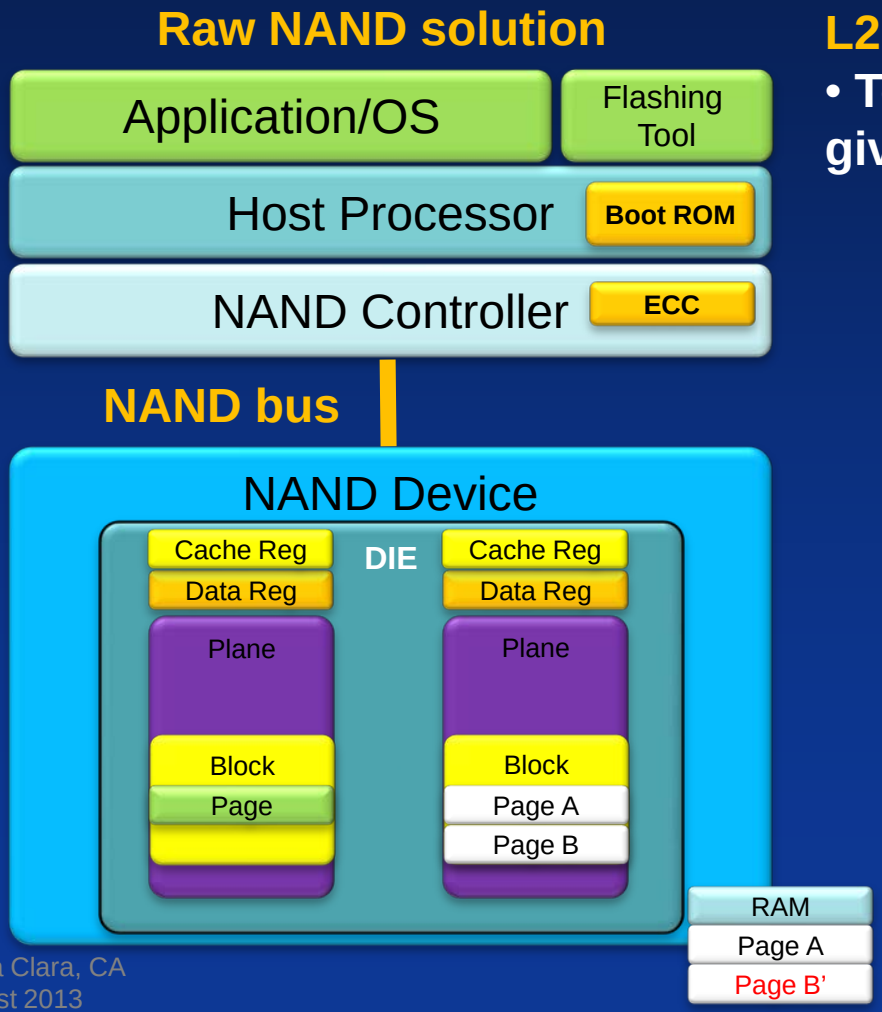
# RAW NAND solution




**L2P mapping - in place update**

- **To do NAND in place update for a given page :**
  - **Copy all valid pages from the block of the given page to RAM**

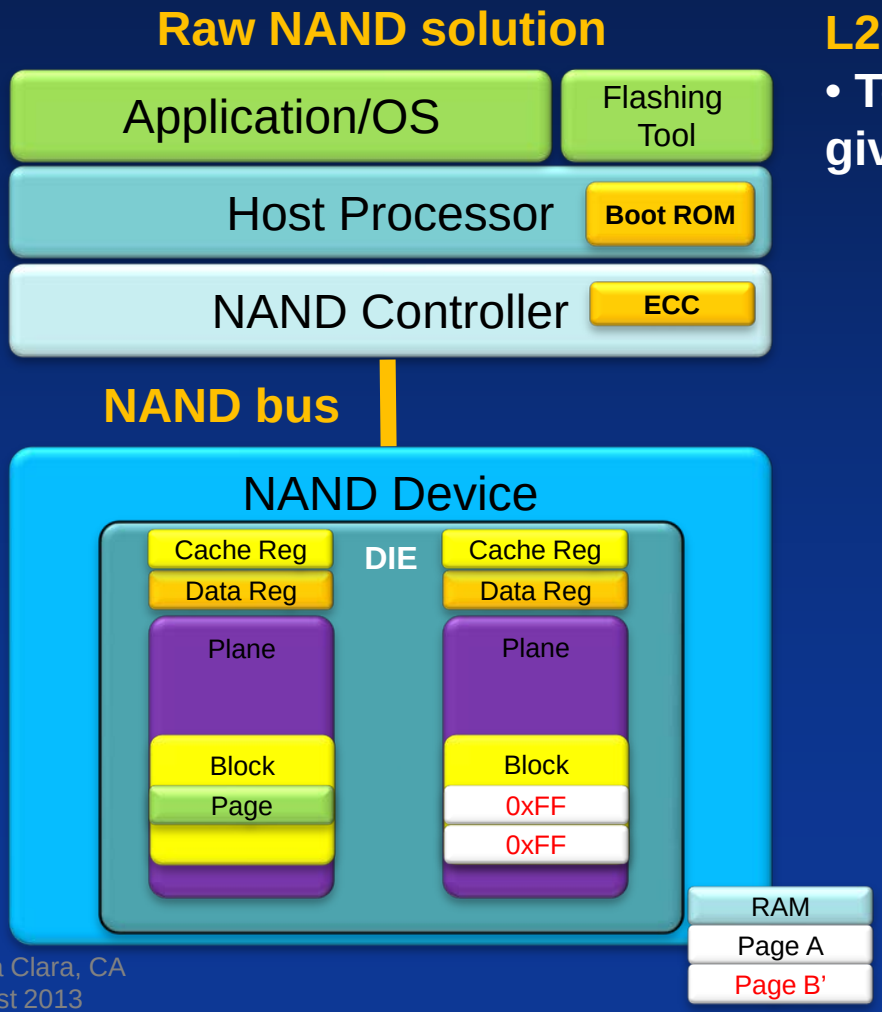
# RAW NAND solution




**L2P mapping - in place update**

- **To do NAND in place update for a given page :**
  - **Copy all valid pages from the block of the given page to RAM**
  - Apply the update in RAM

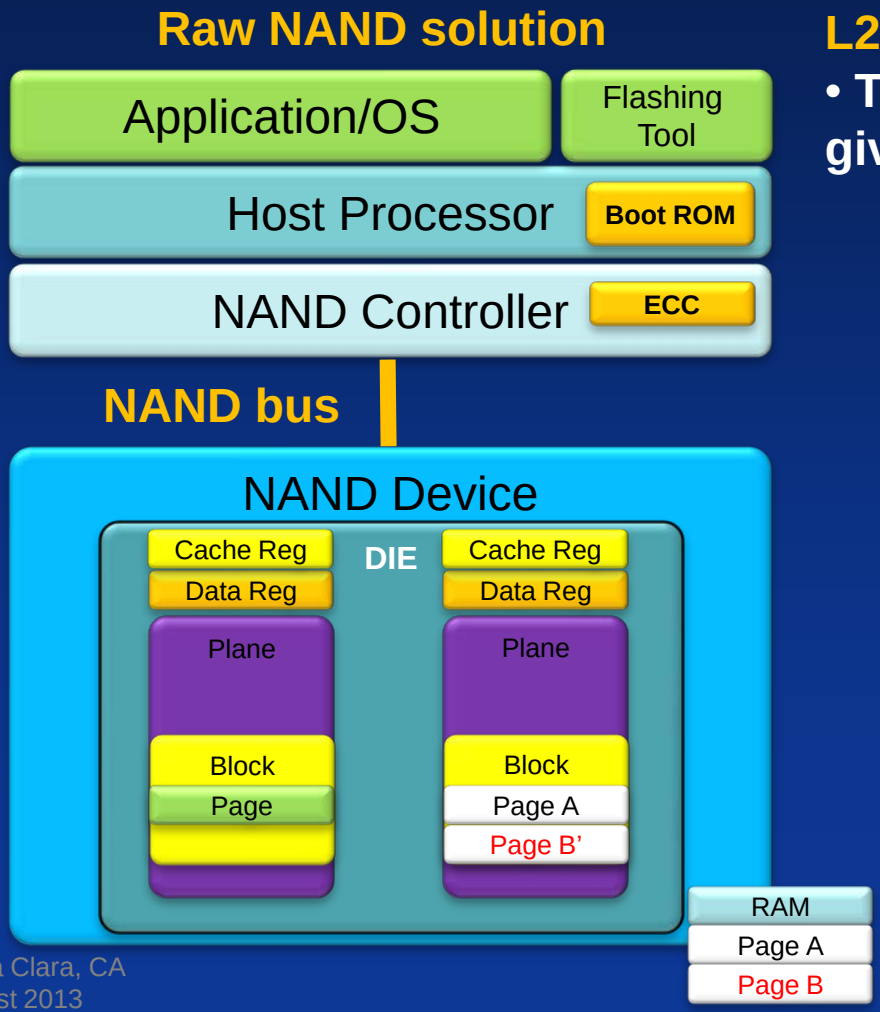
# RAW NAND solution




**L2P mapping - in place update**

- **To do NAND in place update for a given page :**
  - **Copy all valid pages from the block of the given page to RAM**
  - **Apply the update in RAM**
  - **Erase the block**

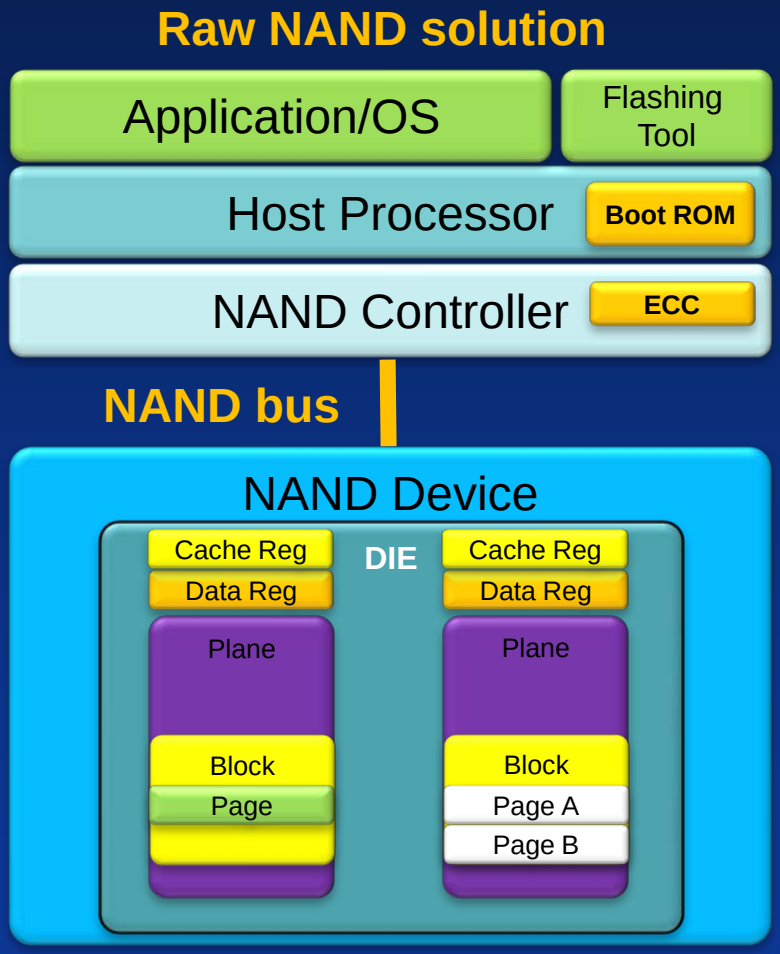
# RAW NAND solution




**L2P mapping - in place update**

- **To do NAND in place update for a given page :**
  - **Copy all valid pages from the block of the given page to RAM**
  - **Apply the update in RAM**
  - **Erase the block**
  - **Program pages from RAM to NAND**

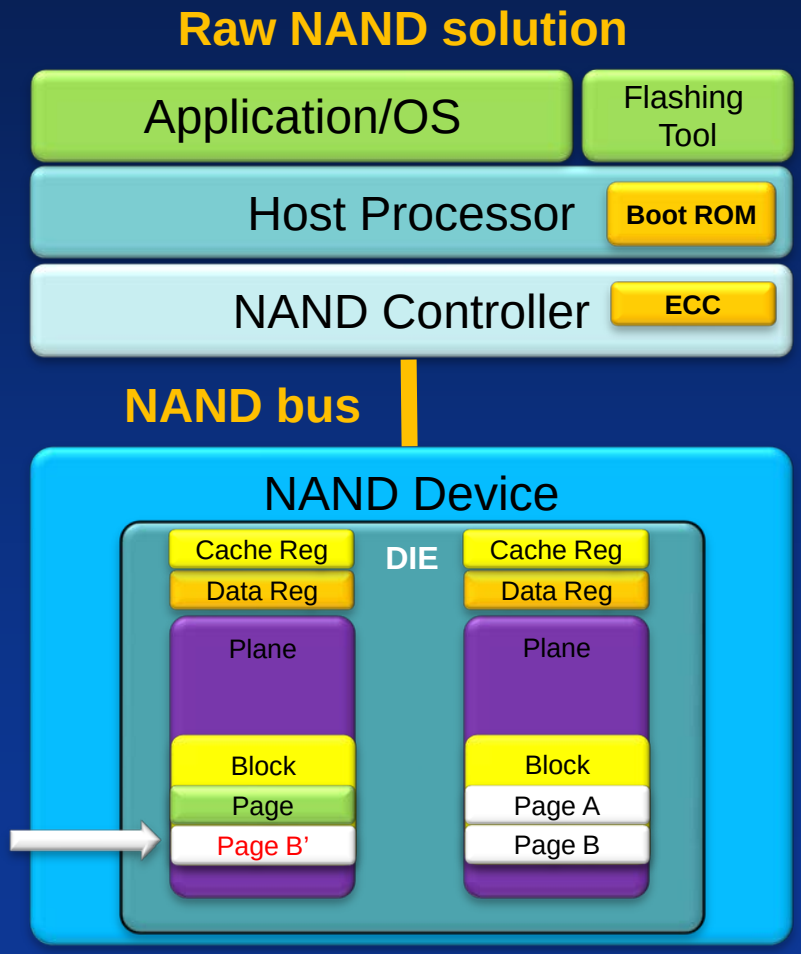
# RAW NAND solution




**L2P mapping - in place update**

- **In place update for NAND implementation is not practical :**
  - **Copying valid pages and erasing block are time consuming**
  - **Valid page my be lost in the event of power loss**

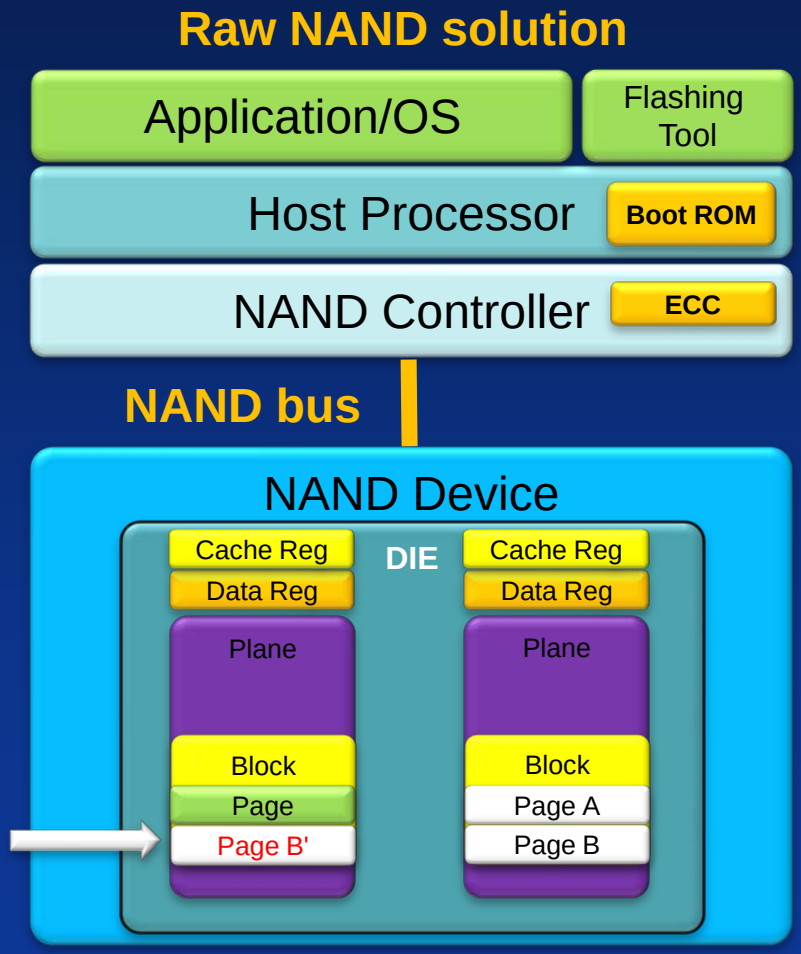
# RAW NAND solution




**L2P mapping - Out of place update**

- **Out of place update (OPU) stores updated page in a new location**

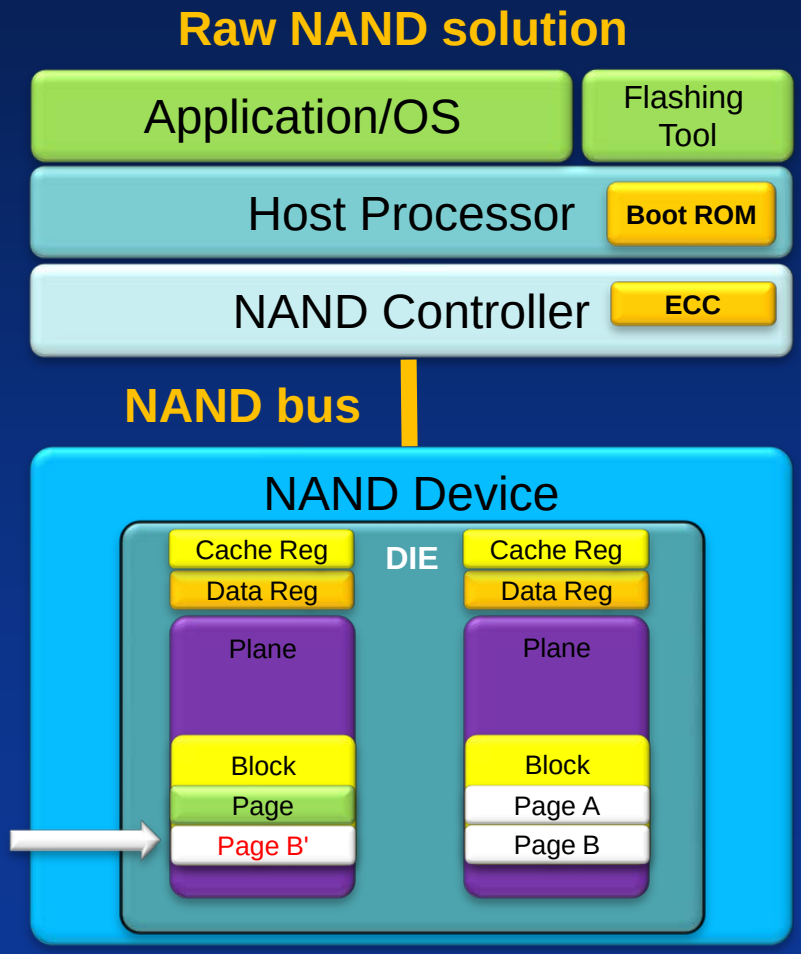
# RAW NAND solution




**L2P mapping - Out of place update**

- **Out of place update (OPU) stores updated page in a new location**
- **OPU implementation requires :**
  - **Logical to physical mapping function to find and track new location for updating data page**
  - **Garbage collection function to remove obsolete pages making room for future update**

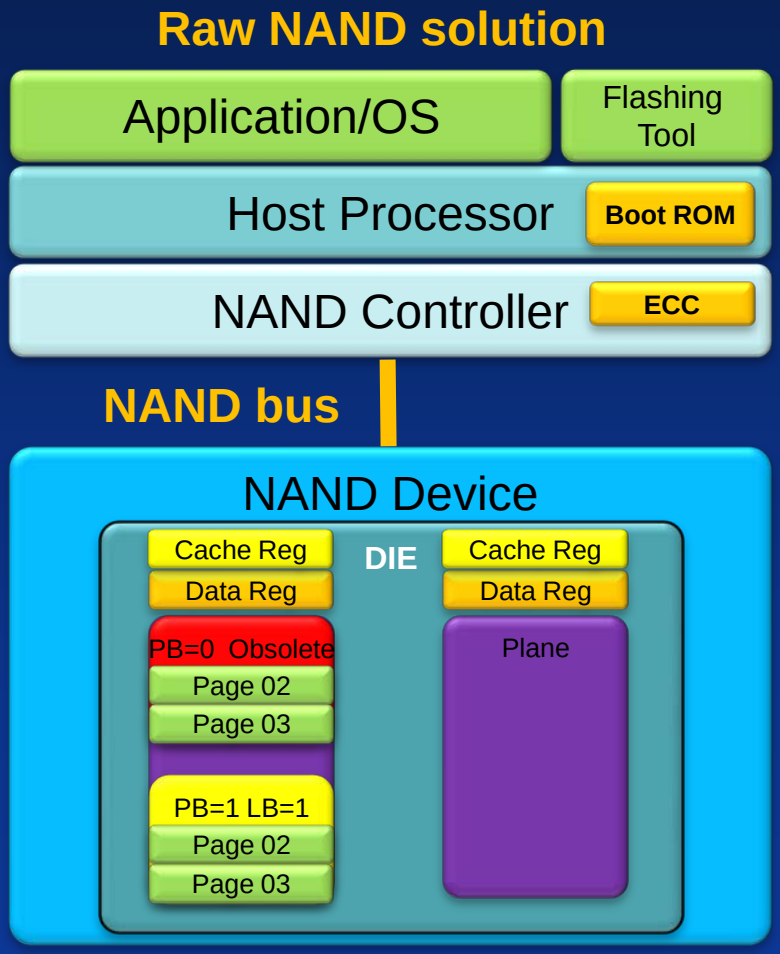
# RAW NAND solution




**L2P mapping - Out of place update**

- **Out of place update (OPU) stores updated page in a new location**
- **OPU implementation requires :**
  - **Logical to physical mapping function to find and track new location for updating data page**
  - **Garbage collection function to remove obsolete data pages making room for future update**
- **OPU block and page based L2P mappings are popular today**

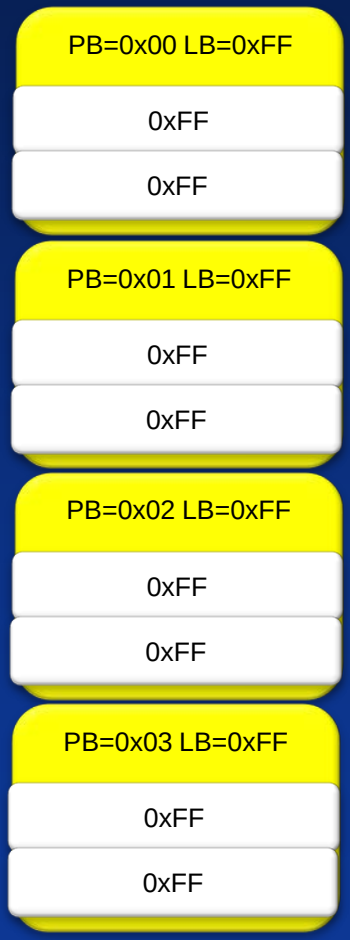
# RAW NAND solution




**OPU Block base L2P mapping**

- **Group logical pages in logical block**
- **Keep track of logical block in DRAM**
- **Perform logical block garbage collection when there is no space for new update**

# RAW NAND solution

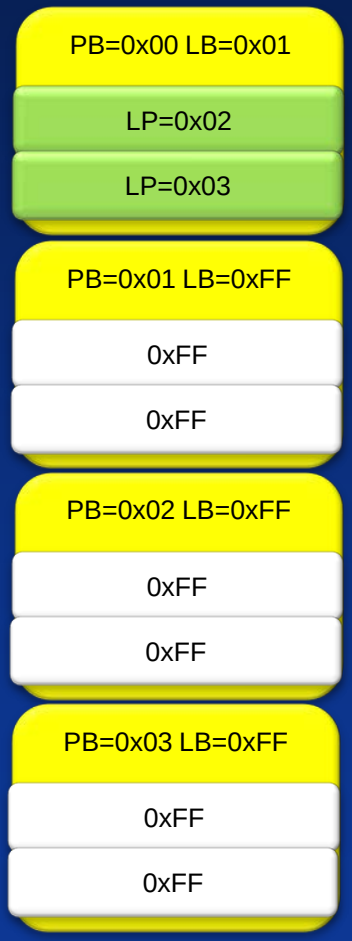



**OPU Block base L2P mapping**

- **An example**
  - **4 erased blocks with 2 pages per block**
  - **Logical blocks hold 2 consecutive logical pages per block**

**PB = Physical block LP = Logical page**
**LB = Logical block**

# RAW NAND solution

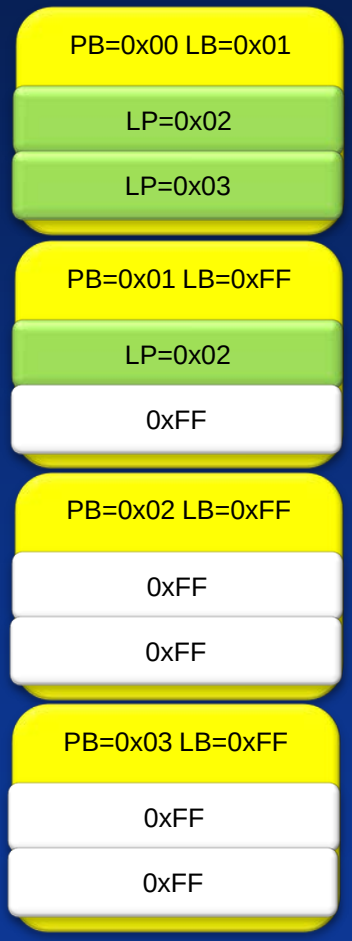



**OPU Block base L2P mapping**

- **An example**
  - **4 erased blocks with 2 pages per block**
  - **Logical blocks hold 2 consecutive logical pages per block**
  - **Write logical pages 2 and 3**
    - **Physical block 0 holds data for logical block 1**

**PB = Physical block LP = Logical page**
**LB = Logical block**

# RAW NAND solution




**OPU Block base L2P mapping**

- **An example**
  - **Write page 2 again**
    - **Write new logical page 2 into physical block 1 page 0**

**PB = Physical block LP = Logical page**
**LB = Logical block**

# RAW NAND solution


**OPU Block base L2P mapping**

- **An example**
  - **Write page 2 again**
    - **Write new logical page 2 into physical block 1 page 0**
  - **Copy logical page 3 to physical block 1 page 1**

**PB = Physical block LP = Logical page**
**LB = Logical block**

# RAW NAND solution


PB = Physical block LP = Logical page
LB = Logical block = obsolete data

**OPU Block base L2P mapping**

- An example
  - Write page 2 again
    - Write new logical page 2 into physical block 1 page 0
  - Copy logical page 3 to physical block 1 page 1
  - Mark physical block 0 as obsolete
    - Physical block 0 is a candidate for garbage collection
  - Physical block 01 holds data for logical block 01

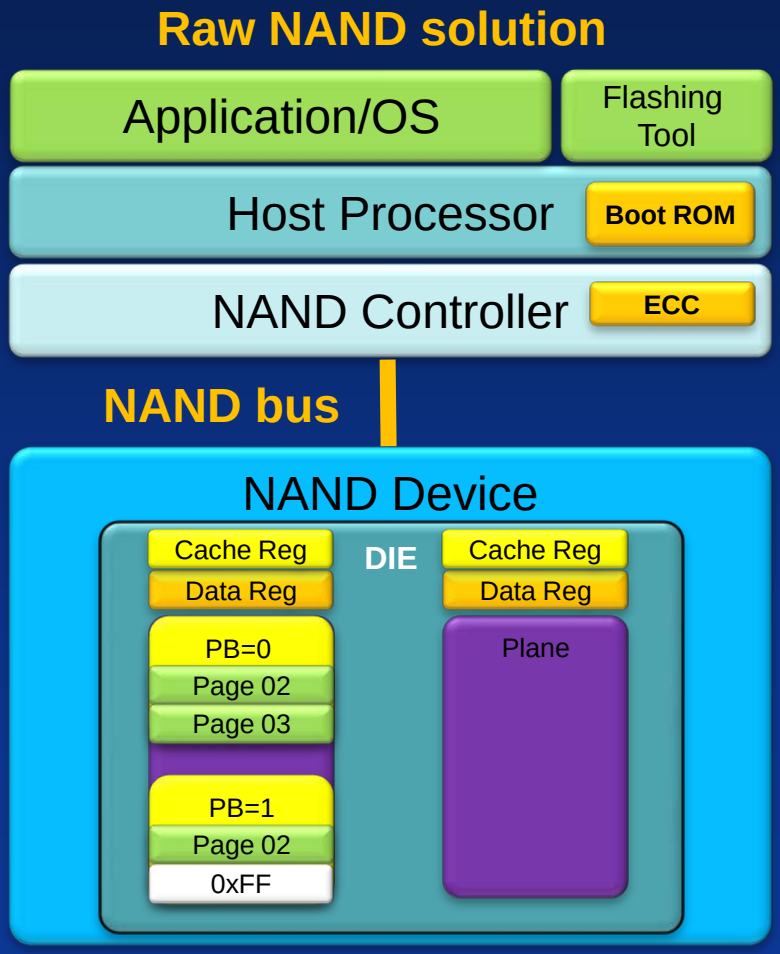
# RAW NAND solution


**OPU Block base L2P mapping**

- **Advantage:**
  - **Uses small amount of RAM mainly for logical to physical block mapping table storage**
- **Disadvantage:**
  - **Copy data even when free space is available for programming**

**PB = Physical block LP = Logical page**
**LB = Logical block = obsolete data**

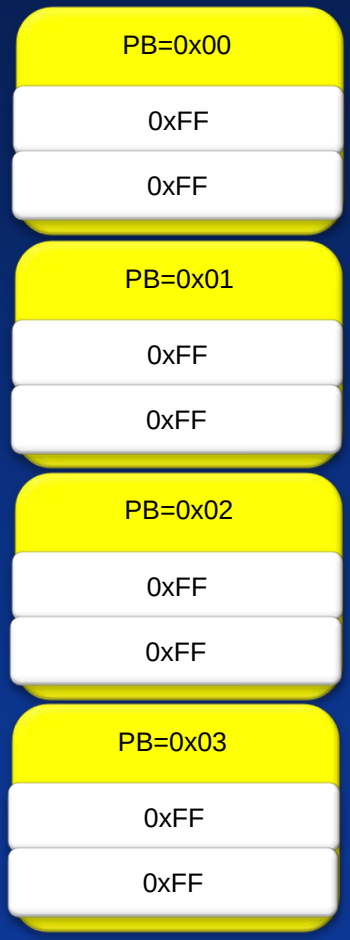
# RAW NAND solution




**OPU page base L2P mapping**

- **Program logical page into next available physical page**
- **Keep track of logical page to physical page mapping**
- **Perform physical block garbage collection when there is no space for new update**

# RAW NAND solution

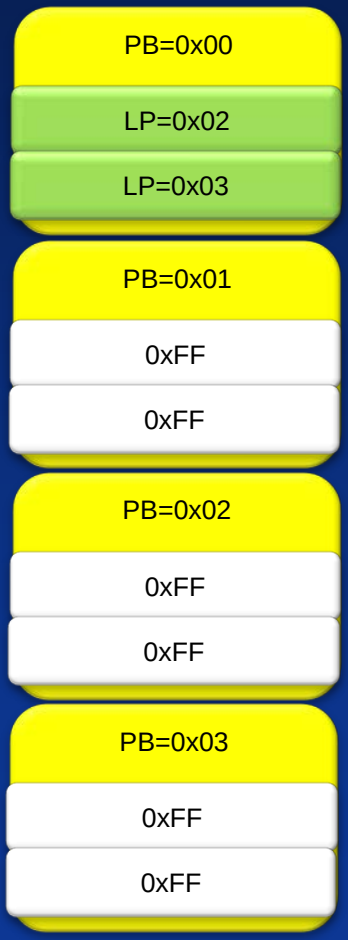



**OPU page base L2P mapping**

- **An example**
  - **4 erased blocks with 2 pages per block**
  - **Physical block holds any 2 logical pages per block**

**PB = Physical block LP = Logical page**

# RAW NAND solution

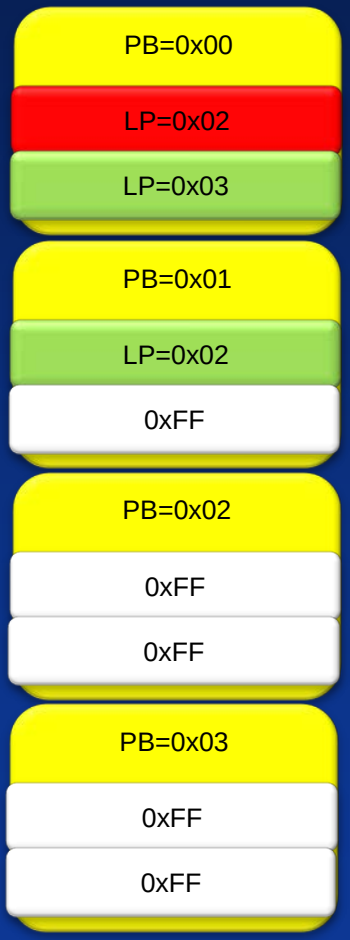



**OPU page base L2P mapping**

- **An example**
  - **4 erased blocks with 2 pages per block**
  - **Physical block holds any 2 logical pages per block**
  - **Write logical page 2 and 3**
    - **Physical block 0 holds the two written pages**

**PB = Physical block LP = Logical page**

# RAW NAND solution

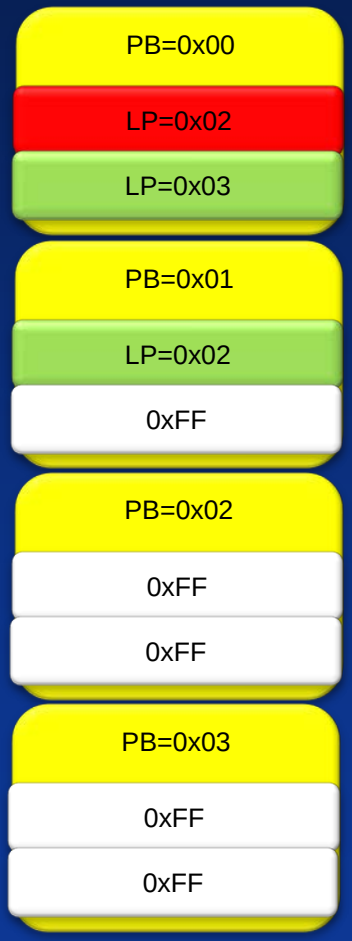



**OPU page base L2P mapping**

- **An example**
  - **Write page 2 again**
    - **Write new logical page 2 into physical block 1 page 0**
    - **Mark LP 0x02 in PB 0x00 as obsolete**

**PB = Physical block LP = Logical page**
**= obsolete data**

# RAW NAND solution

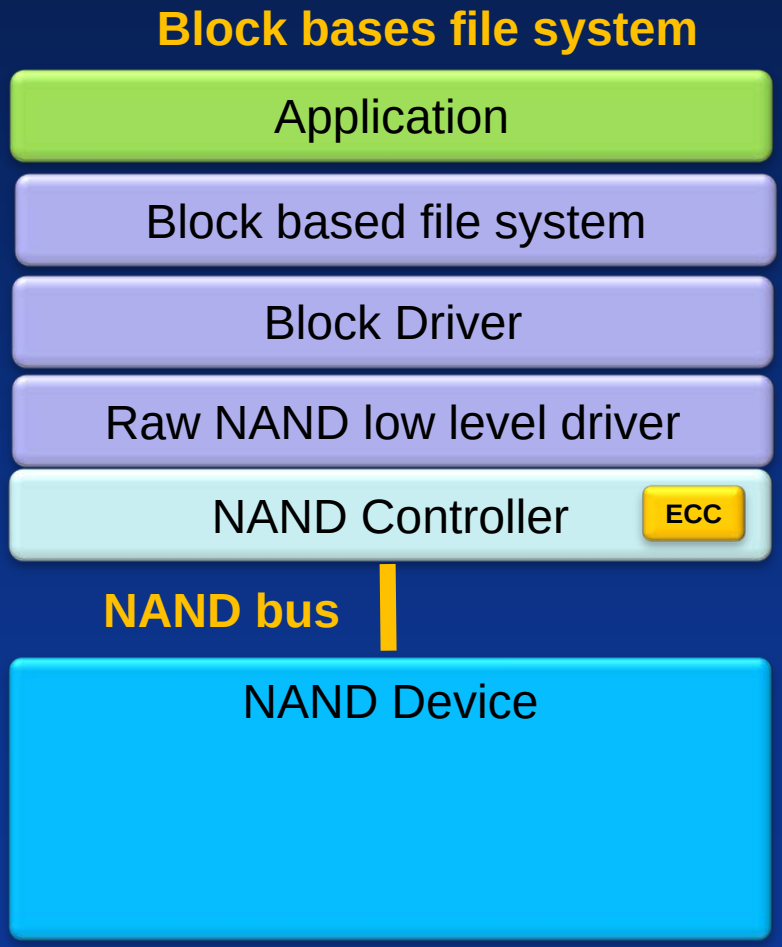



**OPU page base L2P mapping**

- **Advantage:**
  - **No copying logical page for programming a new page**
- **Disadvantage:**
  - **Need RAM and sophisticated algorithm to track logical to physical page mapping table**

**PB = Physical block LP = Logical page**
**= obsolete data**

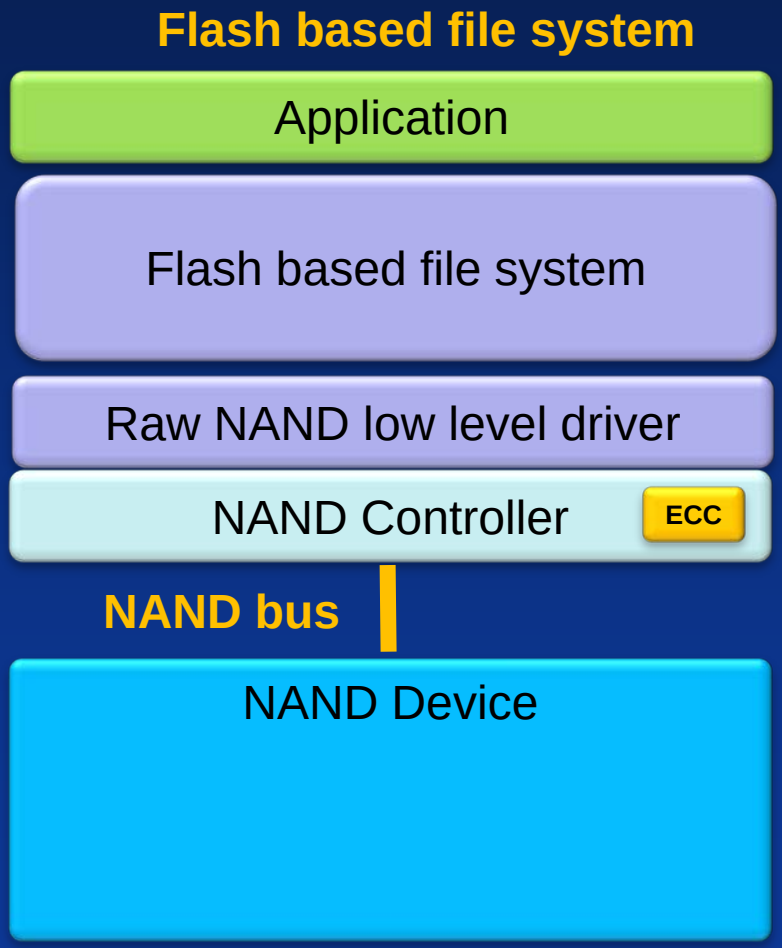
# RAW NAND solution




**Block and flash based file systems**

- **Block device uses L2P mapping to provide logical sector reading and writing access with RAW NAND storage**
- **Application uses functions of file system to perform data storage**
- **Block based file system calls block driver to perform application data storage requests with the RAW NAND storage media**
- **FAT and EXT4 are popular block based file systems**

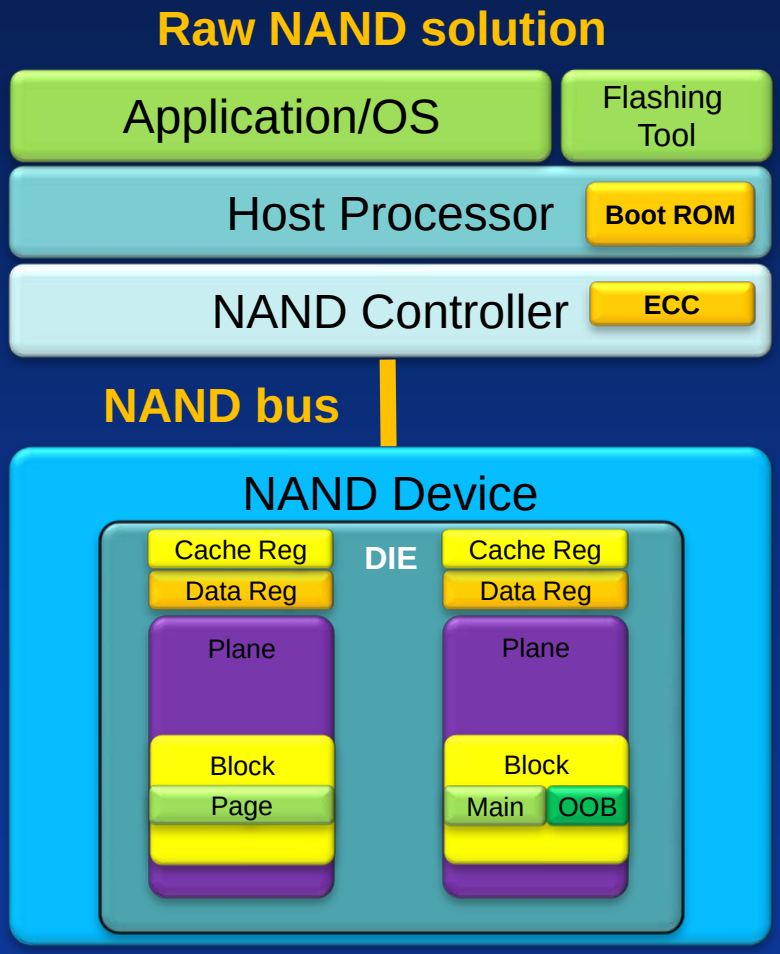
# RAW NAND solution




**Block and flash based file systems**

- **Flash based file systems handle file system data to NAND flash page storage mapping directly**
- **Flash based file system does not need block driver**
- **Most of the flash based file systems are from the open source community**
- **YAFFS2 and JFFS2 are popular flash based file systems**

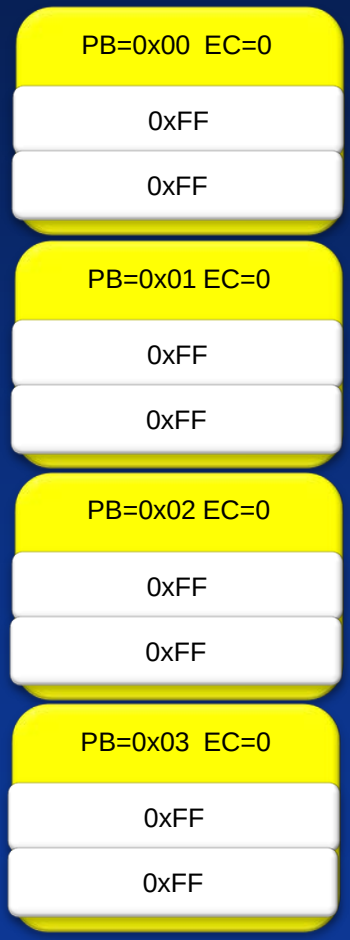
# RAW NAND solution




## Wear leveling

- **Wear leveling tries to maintain similar erase counts for all blocks managed by L2P mapping**
- **Similar erase counts mean the maximum erase count minus the minimum erase count of managed blocks must be less than or equal to the defined maximum erase count gap value**
- **A common erase count gap value is the number of good blocks in the system divided by two**

# RAW NAND solution

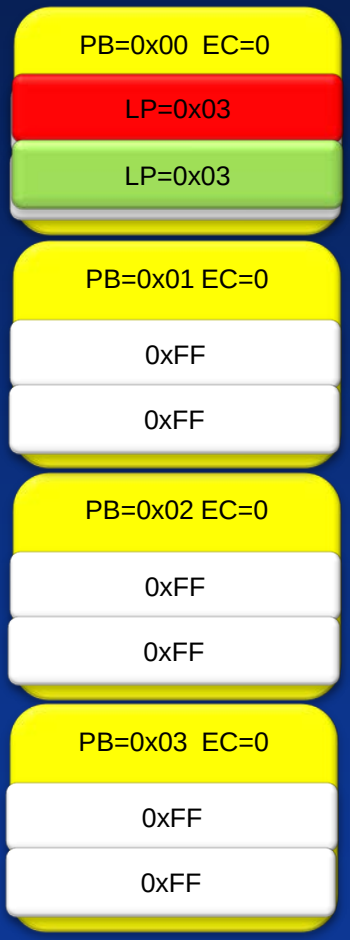



## Wear leveling

- An example without wear leveling
  - 4 erased blocks with 2 pages per block
  - Max erase count per block is 2
  - OPU page based L2P mapping requires two provision blocks for writing

PB = Physical block LP = Logical page
EC = Erase count

# RAW NAND solution

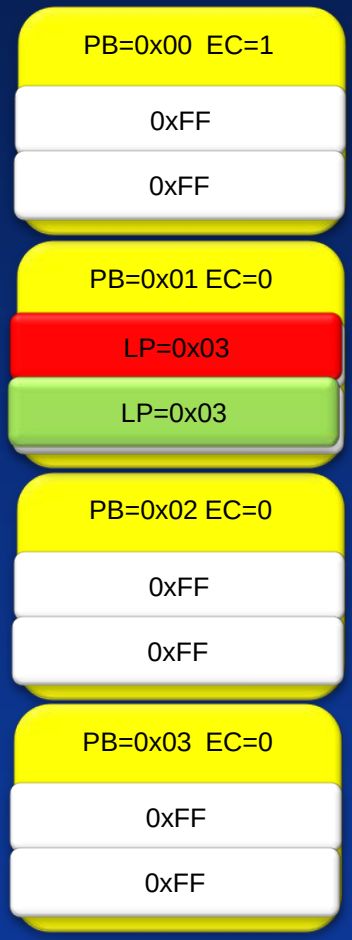



**Wear leveling**

- **An example without wear leveling**
  - **Write logical page 3 two times to PB 0x00**

**PB = Physical block LP = Logical page**
**EC = Erase count = Obsolete data**

# RAW NAND solution

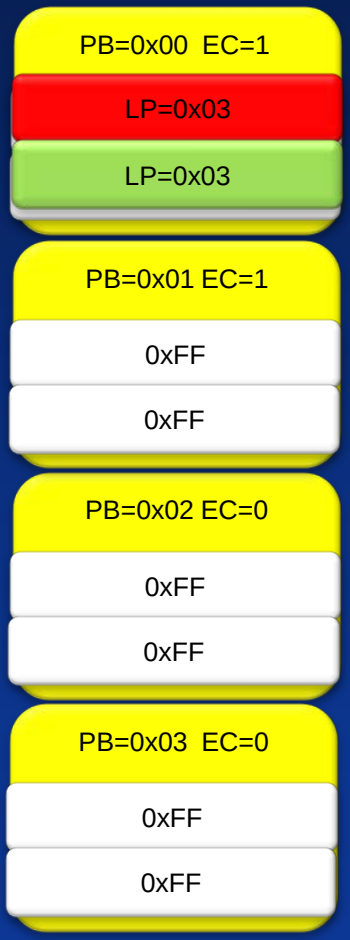



**Wear leveling**

- **An example without wear leveling**
  - **Write logical page 3 two times to PB 0x01**
  - **Erase PB 0x00**

**PB = Physical block LP = Logical page**
**EC = Erase count = Obsolete data**

# RAW NAND solution

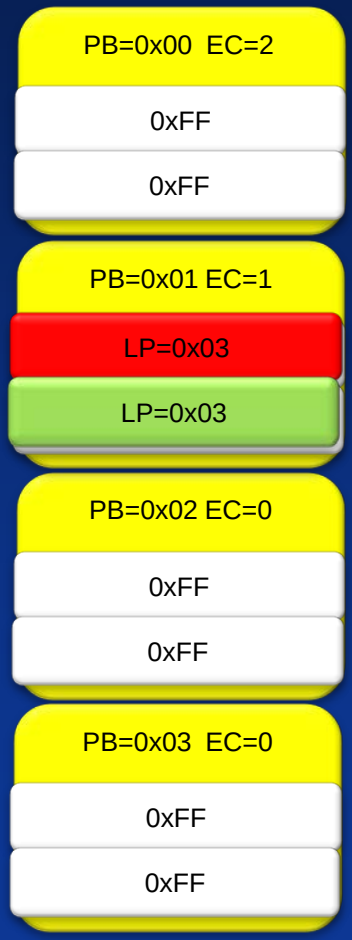



**Wear leveling**

- **An example**
  - **Write logical page 3 two times to PB 0x00**
  - **Erase PB 0x01**

**PB = Physical block LP = Logical page**
**EC = Erase count = Obsolete data**

# RAW NAND solution

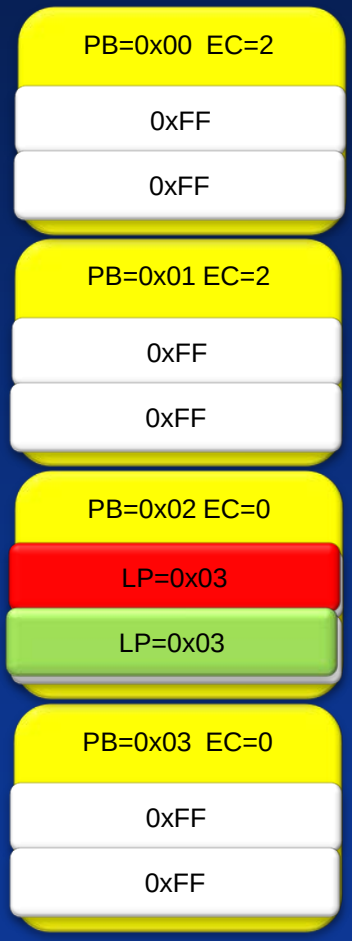



**Wear leveling**

- **An example without wear leveling**
  - **Write logical page 3 two times to PB 0x01**
  - **Erase PB 0x00**

**PB = Physical block LP = Logical page**
**EC = Erase count = Obsolete data**

# RAW NAND solution

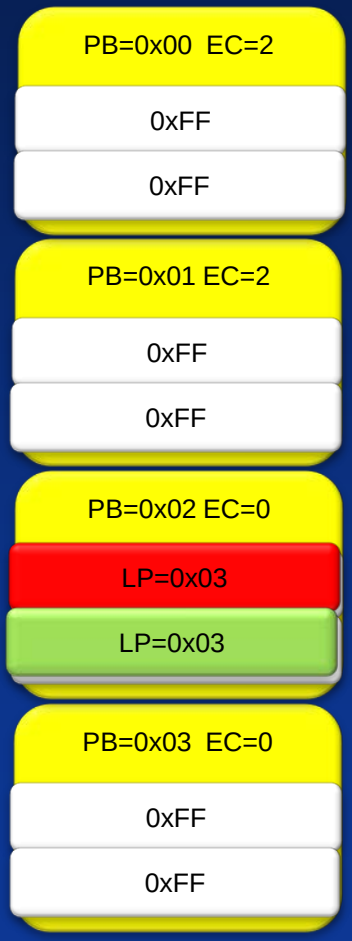



**Wear leveling**

- **An example without wear leveling**
  - **Write logical page 3 two times to PB 0x01**
  - **Erase PB 0x00**

**PB = Physical block LP = Logical page**
**EC = Erase count = Obsolete data**

# RAW NAND solution



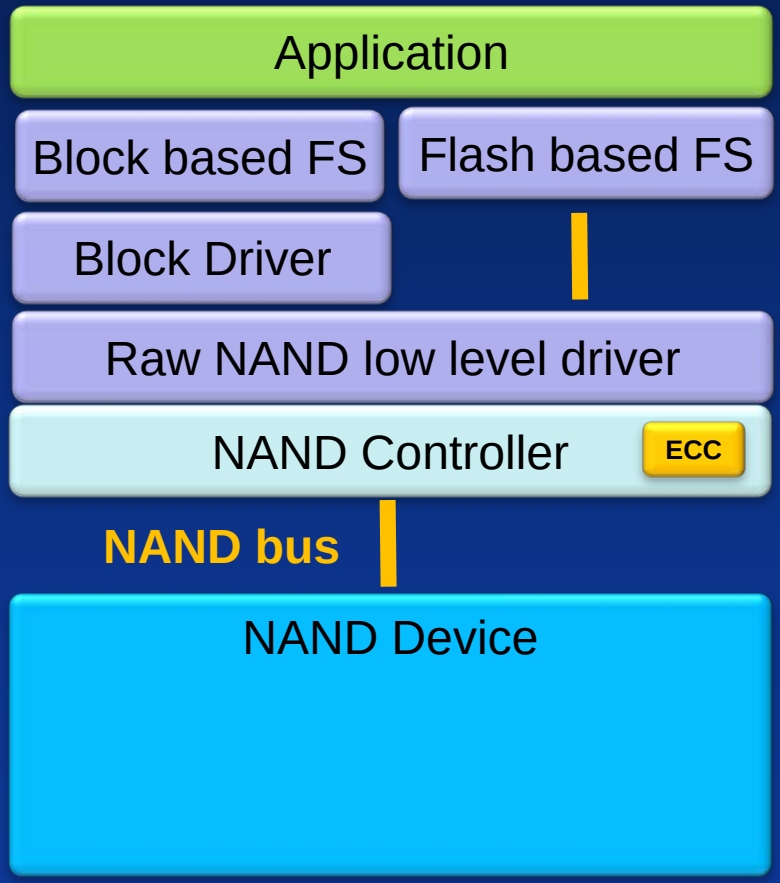

## Wear leveling

- An example without wear leveling
  - Can not write any more for less than two provision blocks available
  - PB 0x00 and 0x01 have reached maximum erase count of 2

PB = Physical block LP = Logical page
EC = Erase count = Obsolete data

# RAW NAND solution

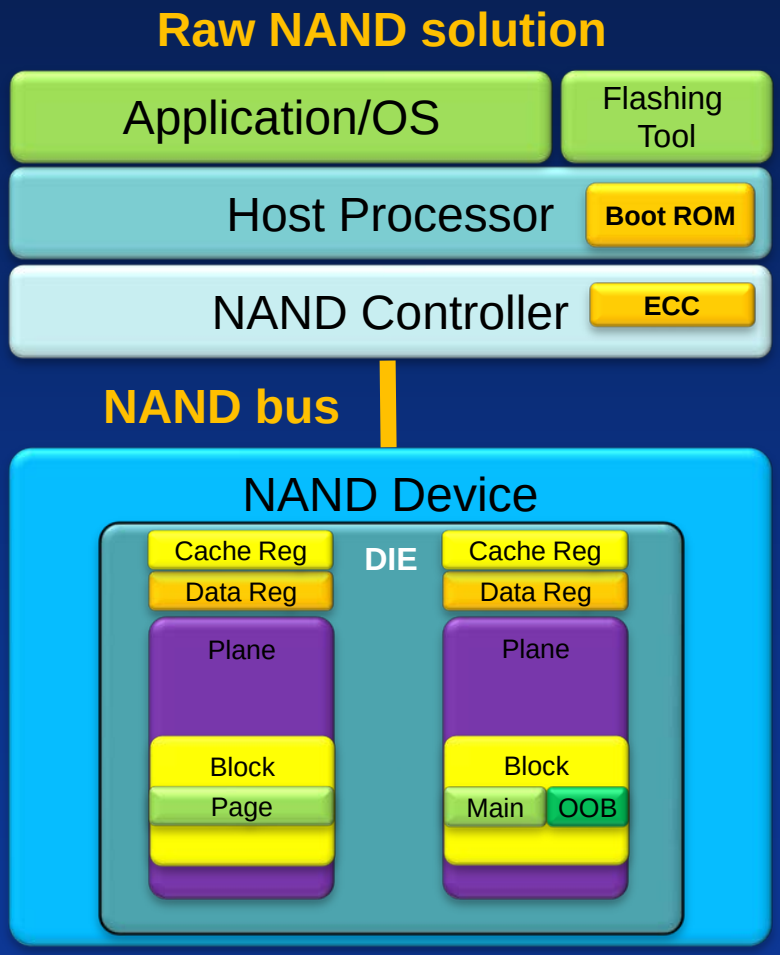
**Block and flash based file system**




**Power loss recovery**

- **Power loss recovery means the raw NAND storage system can be functional after an event of power loss**
- **Programming data for a page in the event of power loss may be lost**
- **Erasing a block in the event of power loss may result with unknown data in the block**
- **No existing data in the raw NAND storage system may be lost in the event of the power loss**

# RAW NAND solution



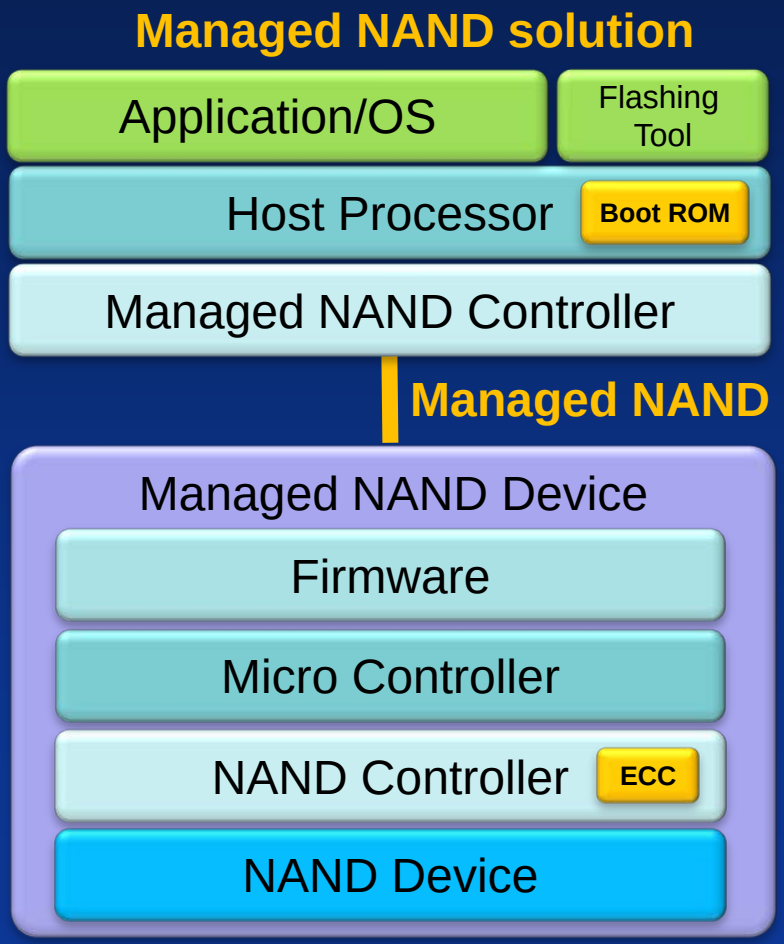

## Flashing and booting

- **Need to make sure the flashing tool for the target system can flash the selected NAND device**

- **Need to make sure the boot ROM code for the target system can boot reliably with the selected NAND device and meet the application life expectancy requirement**


# Managed NAND solution considerations

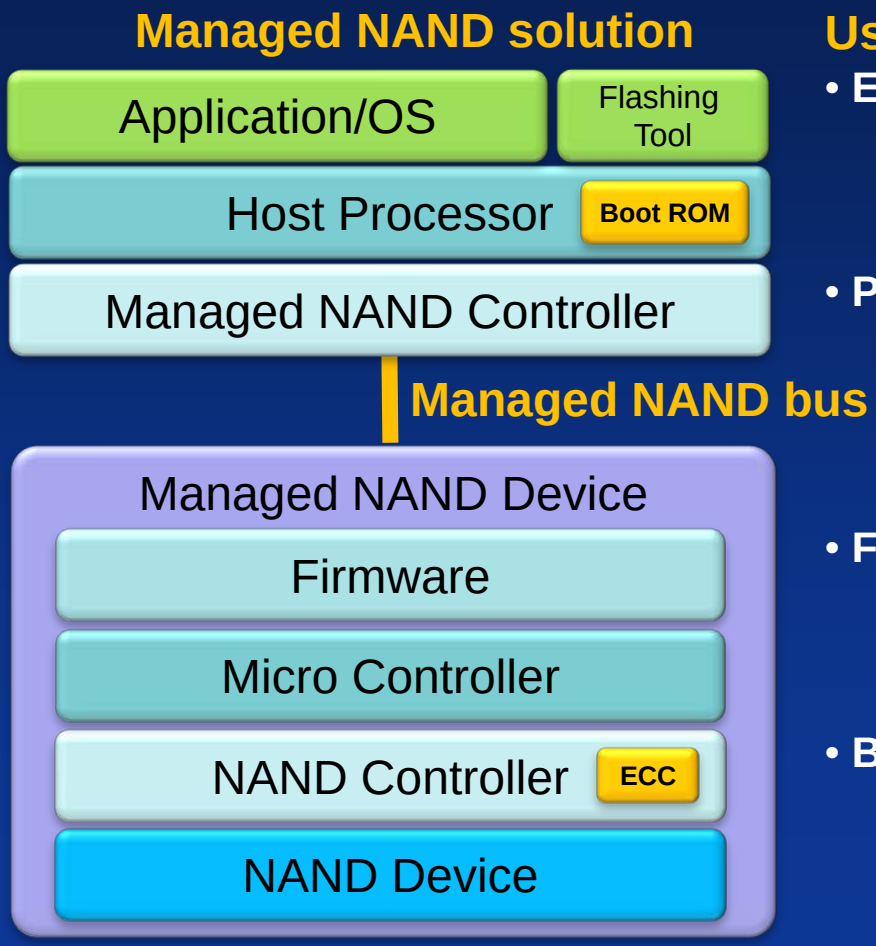
# Managed NAND solution




## Usage Consideration

- **Host platform hardware support**
  - **Ensure the host platform bus can support selected device**
  - **Ensure the host platform controller can support the version of the selected device**
- **Host platform software support**
  - **Ensure the host platform software can support the version of the selected device**
- **Performance**
  - **Ensure the host platform performance meets application requirements**

# Managed NAND solution




## Usage Consideration

- **Endurance**
  - **Ensure endurance meets application requirements**
- **Power loss recovery**
  - **Ensure reading and writing sectors in the event of power loss do not impact existing data stored in the device**
- **Flashing**
  - **Ensure target platform tool can flash the device**
- **Booting**
  - **Ensure target platform boot ROM code can boot from the device**


# Summary

- Managed NAND storage solution
  - Simple integration
  - Larger density
  - No customization for performance and endurance
- RAW NAND storage solution
  - Complex integration
  - Lower density
  - Customization for performance and endurance
  - Deliver higher performance for 16GB or less device