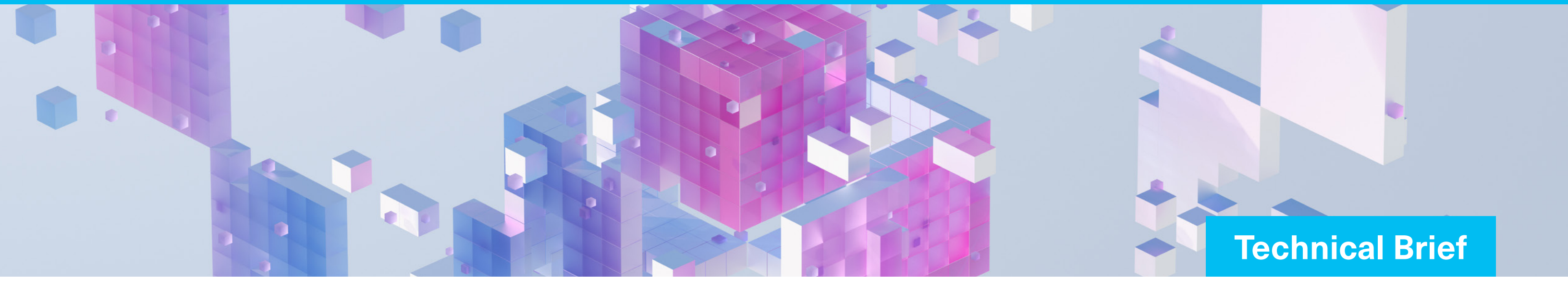


Managing Garbage Collection Operations Efficiently to Improve e-MMC[1] and UFS[2] Device Performance

NAND flash memory enables content mobility that is relied upon daily for personal and business use. When embedded in a managed flash[3] device (e-MMC/UFS) for smartphones, PCs and a wide variety of other products, its performance and Write Amplification Factor[4] (WAF) may worsen as the NAND flash memory reaches its storage capacity limit. If this occurs, the managed flash device, as well as access to files, may experience a performance slowdown and a reduction in lifecycles of the device itself. By understanding how the e-MMC or UFS device addresses these shortcomings and the actions it takes, users can avoid performance lags while improving device lifecycles.

How Does an e-MMC or UFS Device Address Garbage Collection?

A host (server, system, controller, etc.) communicates with e-MMC and UFS devices for data access and retrieval. The fundamental read and write operations are performed using logical block addresses (LBAs). Once the managed flash device receives data, it writes the data to a physical block address (PBA) using available free blocks in NAND flash memory. The device also develops the mapping between an LBA and PBA using logical-to-physical (L2P) tables for these purposes.

An e-MMC or UFS device writes data to available blocks depending on the amount of data it receives. However, the distinctive nature of flash storage introduces complexities. For example, once data in a block is written, flash storage does not allow the data to be overwritten. Instead, the managed flash device performs an erase operation to a large group of blocks, of which that block is included. All written data in that large group are erased, enabling the blocks to be written to again.

To modify data in a previously written block, the entire block must be erased for subsequent rewriting. When the host responds to an overwrite operation, the e-MMC or UFS device writes data in a free block. It then updates the L2P table that the status of the overwritten data has changed to invalid data[5], and the new data is now valid data[6]. At this time, the invalid data is stored in flash storage and not erased immediately.

When an e-MMC or UFS device is first used, write operations are very straightforward and easy to complete without much contention because there are typically plenty of free blocks available. However, as more data is written to available blocks, there will be less blocks available to write into. Or, in situations when there is an abundance of invalid data, flash storage may become fragmented. When these circumstances occur, the managed flash device performs garbage collection (GC) to enable blocks to become available, and utilizes the following process:

1. *Locates a group of blocks that need to be erased, of which some of the blocks contain invalid data*
2. *Copies all valid data in the group of blocks to free blocks*
3. *Updates the L2P table to make the new data valid and the old data invalid*
4. *Erases the group of blocks that became invalid which have now become new free blocks*

In general, the garbage collection process is similar for many flash storage devices, such as SSDs, embedded memory, USB sticks and SD memory cards. For detailed information on this process, refer to the KIOXIA technical brief entitled, “Understanding Garbage Collection in NAND Flash Memory” available here.

How Garbage Collection Works in an e-MMC or UFS Device: Abundant Free Space or Full Capacity

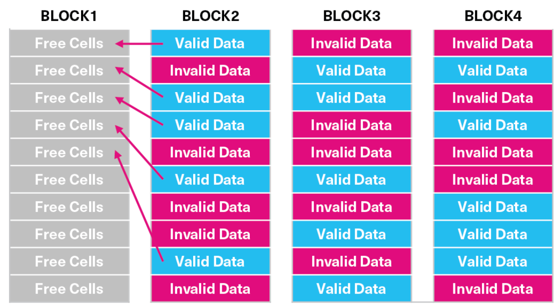
This section describes how garbage collection works based on the quantity of valid data in an e-MMC or UFS device by presenting two examples where valid data consumes either 50% or 90% of the total storage space. In both examples, it is assumed that the managed flash device will be used with a lot of free space to store data, so 50% represents one example. On the other hand, it is assumed that the managed flash device will be used close to a fully used state, so 90% represents the second example.

In the first example, Figure 1 shows how a free block is made with 50% valid data. Initially there is one free block with a goal of making an additional free block.

- *As shown in Step 1, all valid data in BLOCK2 is copied to BLOCK1.*
- *After the copy operation completes, all copied valid data becomes invalid, at which time the block can be erased as shown in Step 2.*
- *Similarly, all valid data in BLOCK3 is copied to the target block sequentially as shown in Step 3.*
- *All copied valid data in BLOCK3 becomes invalid and the block can be erased as shown in Step 4.*
- *At the end of the process, as shown in Step 5, the total number of free blocks is two, which are then copied, and the two blocks are erased.*

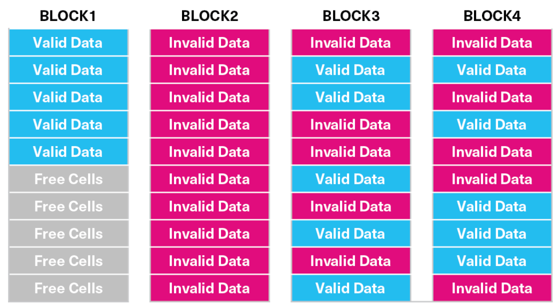
Figure 1 below shows a garbage collection operation made with 50% valid data – **Step 1 thru Step 5**:

Step 1



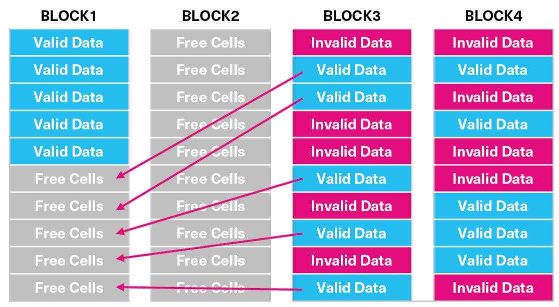
BLOCK1	BLOCK2	BLOCK3	BLOCK4
Free Cells	Valid Data	Invalid Data	Invalid Data
Free Cells	Invalid Data	Valid Data	Valid Data
Free Cells	Valid Data	Valid Data	Invalid Data
Free Cells	Valid Data	Invalid Data	Valid Data
Free Cells	Invalid Data	Invalid Data	Invalid Data
Free Cells	Valid Data	Valid Data	Invalid Data
Free Cells	Invalid Data	Invalid Data	Valid Data
Free Cells	Invalid Data	Valid Data	Valid Data
Free Cells	Valid Data	Invalid Data	Valid Data
Free Cells	Invalid Data	Valid Data	Invalid Data

Step 2



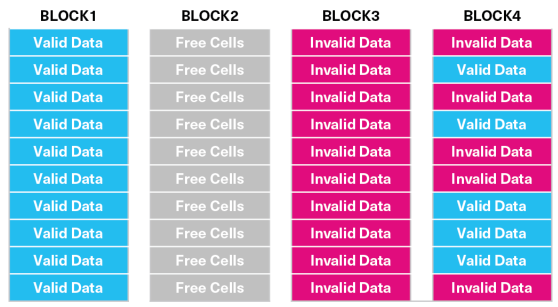
BLOCK1	BLOCK2	BLOCK3	BLOCK4
Valid Data	Invalid Data	Invalid Data	Invalid Data
Valid Data	Invalid Data	Valid Data	Valid Data
Valid Data	Invalid Data	Valid Data	Invalid Data
Valid Data	Invalid Data	Invalid Data	Valid Data
Valid Data	Invalid Data	Invalid Data	Invalid Data
Free Cells	Invalid Data	Valid Data	Invalid Data
Free Cells	Invalid Data	Invalid Data	Valid Data
Free Cells	Invalid Data	Valid Data	Valid Data
Free Cells	Invalid Data	Invalid Data	Valid Data
Free Cells	Invalid Data	Valid Data	Invalid Data

Step 3



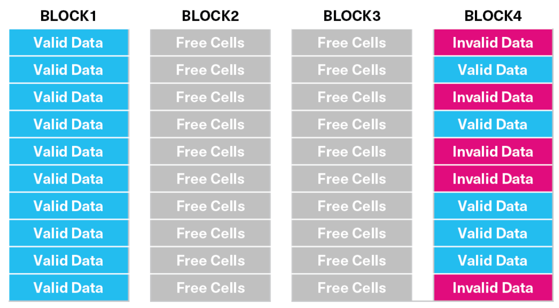
BLOCK1	BLOCK2	BLOCK3	BLOCK4
Valid Data	Free Cells	Invalid Data	Invalid Data
Valid Data	Free Cells	Valid Data	Valid Data
Valid Data	Free Cells	Valid Data	Invalid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Invalid Data
Free Cells	Free Cells	Valid Data	Invalid Data
Free Cells	Free Cells	Invalid Data	Valid Data
Free Cells	Free Cells	Valid Data	Valid Data
Free Cells	Free Cells	Invalid Data	Valid Data
Free Cells	Free Cells	Valid Data	Invalid Data

Step 4



BLOCK1	BLOCK2	BLOCK3	BLOCK4
Valid Data	Free Cells	Invalid Data	Invalid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Invalid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Invalid Data
Valid Data	Free Cells	Invalid Data	Invalid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Valid Data
Valid Data	Free Cells	Invalid Data	Invalid Data

Step 5



BLOCK1	BLOCK2	BLOCK3	BLOCK4
Valid Data	Free Cells	Free Cells	Invalid Data
Valid Data	Free Cells	Free Cells	Valid Data
Valid Data	Free Cells	Free Cells	Invalid Data
Valid Data	Free Cells	Free Cells	Valid Data
Valid Data	Free Cells	Free Cells	Invalid Data
Valid Data	Free Cells	Free Cells	Invalid Data
Valid Data	Free Cells	Free Cells	Valid Data
Valid Data	Free Cells	Free Cells	Valid Data
Valid Data	Free Cells	Free Cells	Valid Data
Valid Data	Free Cells	Free Cells	Invalid Data

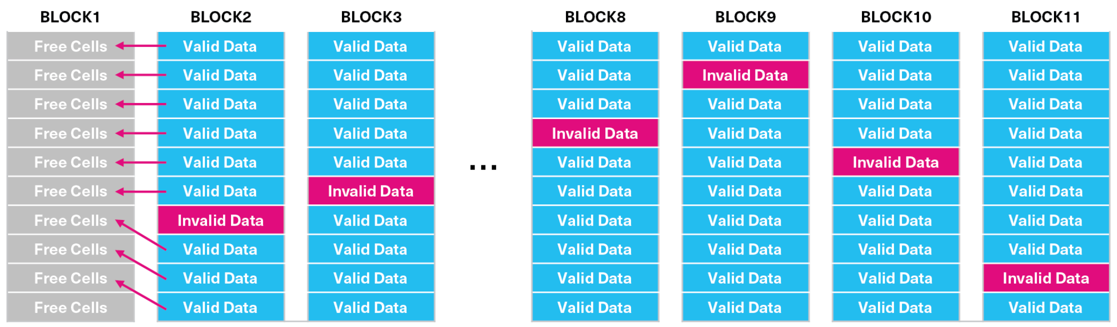
In another example of a garbage collection operation, Figure 2 shows how a free block is made with 90% valid data. Initially, the process is similar to Figure 1 that uses 50% valid data as there is also one free block with the intention of creating an additional free block. The steps to make an additional free block include:

- *As shown in Step 1, all valid data in BLOCK2 is copied to BLOCK1*
- *After the copy operation completes, all data in BLOCK2 becomes invalid and BLOCK2 would be erased as shown in Step 2*
- *All valid data in BLOCK3 is copied to the remaining free cells of BLOCK1 and BLOCK2, as shown in Step 3*
- *All data in BLOCK3 become invalid and BLOCK3 can be erased, as shown in Step 4*
- *After the erase completes, all invalid data in BLOCK3 becomes free cells, as shown in Step 5*
- *For BLOCK4 through BLOCK11, the Step 1 and Step 2 processes are performed sequentially and all valid data in those blocks are copied to free cells, until another block is created to accept free cells*
- *As shown in Step 17, all valid data is copied in BLOCK10 to BLOCK8 and BLOCK9*
- *All valid data in BLOCK10 becomes invalid, as shown in Step 18*
- *As shown in Step 19, all valid data is copied in BLOCK11 to BLOCK9*
- *All valid data in BLOCK11 becomes invalid as shown in Step 20*
- *At the end of the process, all data in BLOCK11 is erased and two free blocks, BLOCK10 and BLOCK11 become available, as shown in Step 21*

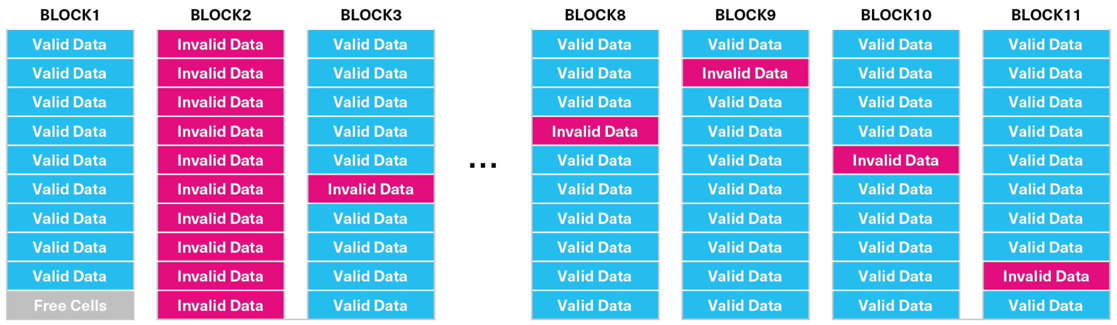
As this process demonstrates, there is less invalid data which requires more garbage collection to make an additional free block. As a result, ten blocks needed to be copied and then erased.

Figure 2 below shows a garbage collection operation made with 90% valid data – **Step 1 thru Step 5 and Step 17 thru Step 21**:

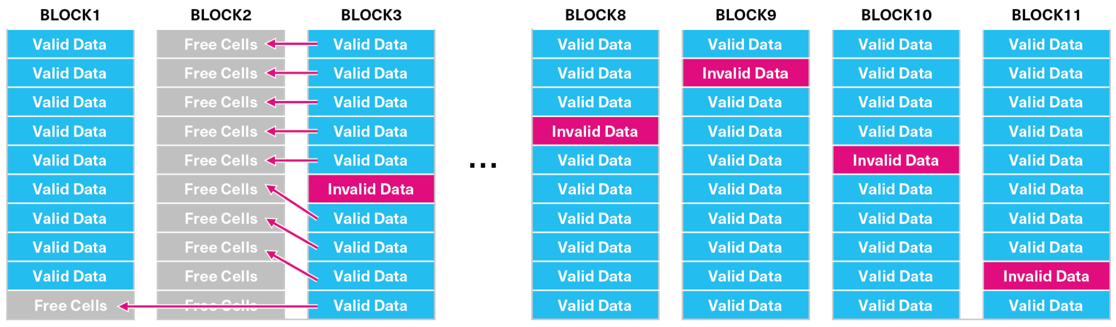
Step 1


BLOCK1
BLOCK2
BLOCK3
BLOCK8
BLOCK9
BLOCK10
BLOCK11
BLOCK1: Free Cells, Free Cells, Free Cells, Free Cells, Free Cells, Free Cells, Free Cells, Free Cells, Free Cells, Free Cells
BLOCK2: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data
BLOCK3: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data
...
BLOCK8: Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK9: Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK10: Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK11: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data

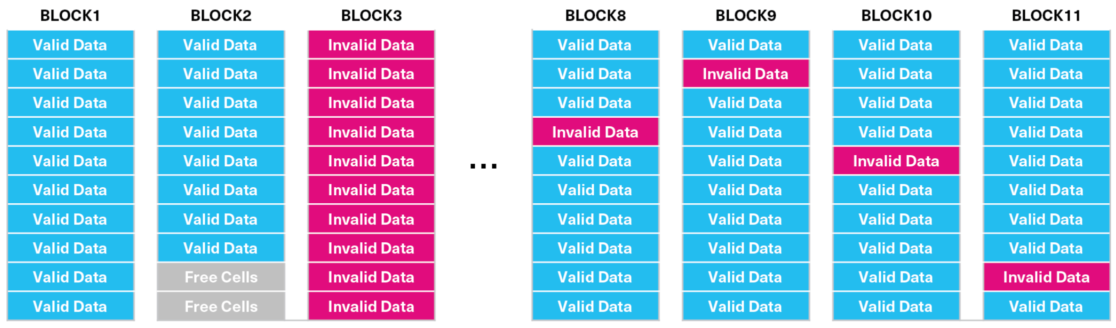
Step 2


BLOCK1
BLOCK2
BLOCK3
BLOCK8
BLOCK9
BLOCK10
BLOCK11
BLOCK1: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Free Cells
BLOCK2: Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data, Invalid Data
BLOCK3: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data
...
BLOCK8: Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK9: Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK10: Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data
BLOCK11: Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Valid Data, Invalid Data, Valid Data

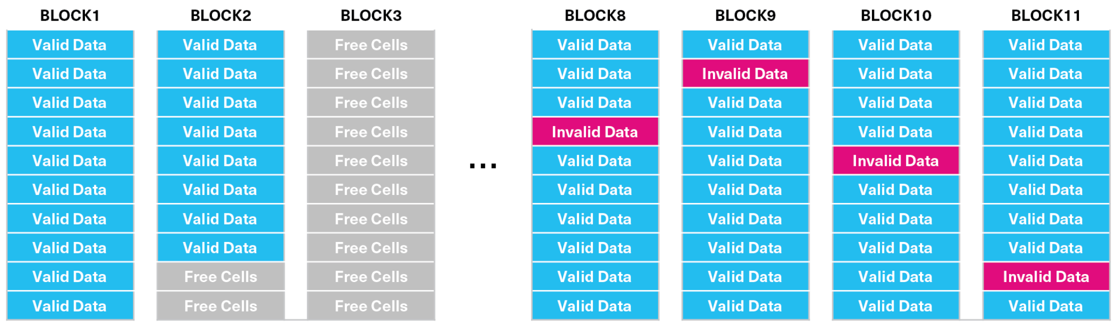
Step 3


BLOCK1 BLOCK2 BLOCK3 … BLOCK8 BLOCK9 BLOCK10 BLOCK11
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Invalid Data | Valid Data
Valid Data | Free Cells | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Invalid Data
Free Cells | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data | Valid Data

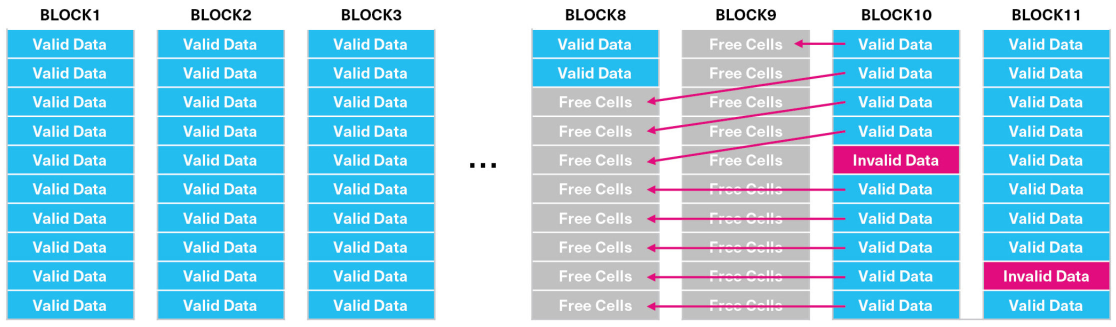
Step 4


BLOCK1 BLOCK2 BLOCK3 … BLOCK8 BLOCK9 BLOCK10 BLOCK11
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Invalid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Invalid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Invalid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Invalid Data | Valid Data | Valid Data | Valid Data | Invalid Data
Valid Data | Free Cells | Invalid Data | Valid Data | Valid Data | Valid Data | Valid Data

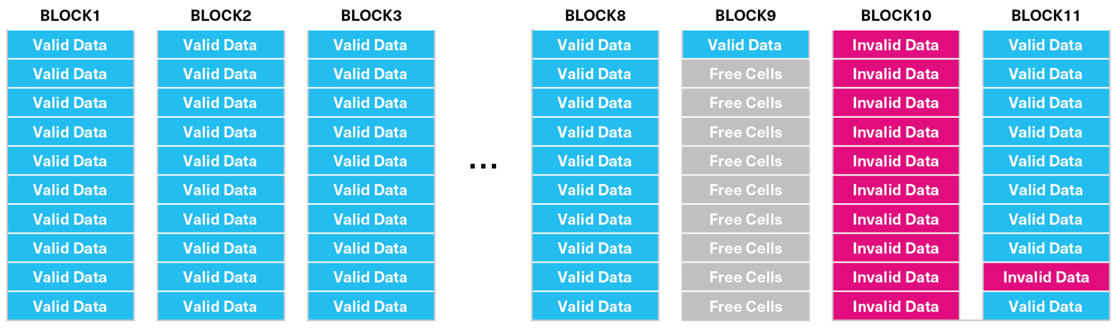
Step 5


BLOCK1 BLOCK2 BLOCK3 … BLOCK8 BLOCK9 BLOCK10 BLOCK11
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Invalid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Invalid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Invalid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Valid Data | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data
Valid Data | Free Cells | Free Cells | Valid Data | Valid Data | Valid Data | Invalid Data
Valid Data | Free Cells | Free Cells | Valid Data | Valid Data | Valid Data | Valid Data

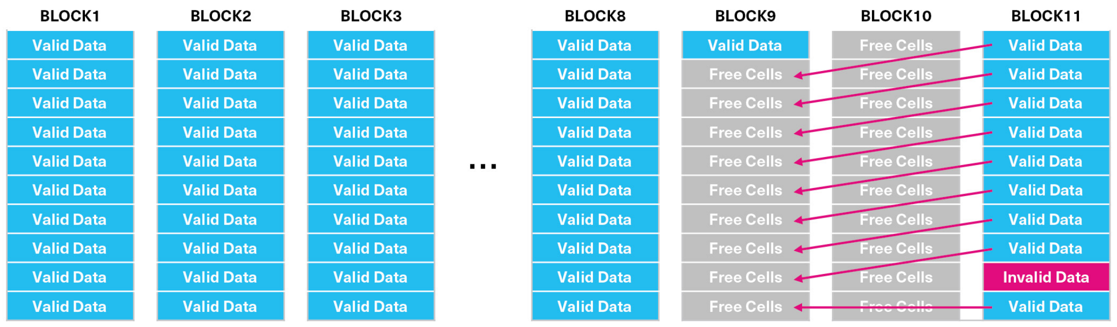
Step 17


BLOCK1 BLOCK2 BLOCK3 … BLOCK8 BLOCK9 BLOCK10 BLOCK11
Valid Data | Valid Data | Valid Data | Valid Data | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Valid Data | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Invalid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Invalid Data
Valid Data | Valid Data | Valid Data | Free Cells | Free Cells | Valid Data | Valid Data

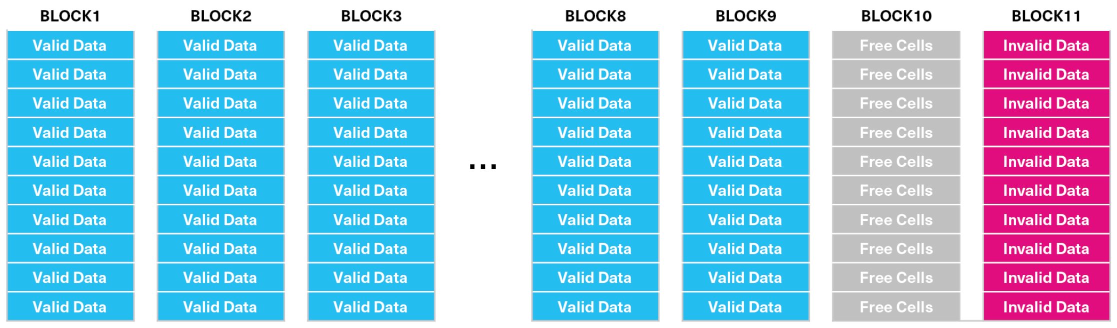
Step 18



Step 19



Step 20



Step 21

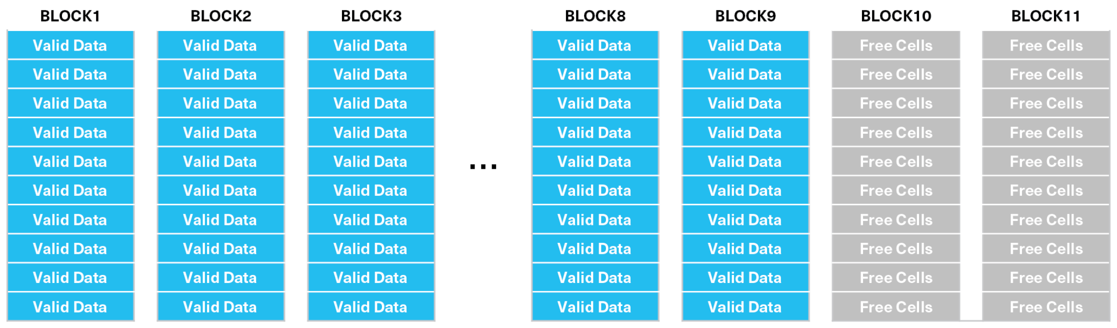


Table 1 presents a comparison of the garbage collection operation when there is 50% valid data and 90% valid data. In the case of 90% valid data, there is less invalid data in blocks, meaning more data needs to be copied and more blocks need to be erased to create an additional free block when compared with 50% valid data.

Percentage of Valid Data	Number of Copied Blocks	Number of Erased Blocks
50%	2	2
90%	10	10

Table 1 shows a comparison when garbage collection is made with 50% and 90% valid data

In these examples, it is assumed that ratio of valid to invalid data is either 50% or 90% in every block and only one free block is required. These examples are intended to provide a clear understanding regarding how garbage collection works - although such scenarios may be unlikely to occur as typically used in an e-MMC or UFS device. In reality, the valid to invalid ratio can vary from block to block whereby the e-MMC or UFS device can select blocks for more efficiency. The more free blocks that are available means a lower percentage of valid data and the more efficient the garbage collection operation will be.

The key point is that whenever there are not a lot of free blocks, additional operations are needed to free up blocks which can cause WAF deterioration and performance degradation in the foreground. In order to avoid these circumstances in advance, the host can issue an 'unmap/discard' command to erase invalid blocks resulting in more free blocks.

Summary

This technical brief describes the challenges faced by NAND flash memory when used in smartphones and PCs that can lead to reduced performance and lifespans as they approach storage capacity limits. The way in which an e-MMC or UFS device addresses garbage collection was discussed in detail, as well as the write, erase and overwrite processes. Garbage collection is essential, as free blocks are reduced and e-MMC or UFS device performance can be impacted.

To prevent performance degradation, data must be managed efficiently and the 'unmap/discard' command can be used to notify the e-MMC or UFS device of invalid data areas, which in turn can improve the efficiency of garbage collection operations.

In conclusion, it is recommended that unnecessary data be regularly deleted to maintain sufficient memory space. Other suggestions include taking advantage of managed flash device data organization features and increasing the memory capacity of the device for optimal performance (or external storage options).

General information about KIOXIA memory products is available here.

FOOTNOTES:

[1] Embedded MultiMediaCard (e-MMC) is a specification developed by JEDEC for mobile applications. The current release is v5.1, published in February 2015.

[2] Universal Flash Storage (UFS) devices are based on the UFS specification. The current release is v4.0 and published by JEDEC in July 2022.

[3] A managed flash device combines raw NAND flash memory and an intelligent controller in one integrated package, enabling internal memory management.

[4] Write Amplification Factor (WAF) is a value that represents the amount of data written by the managed flash controller in relation to the amount of data that the host controller writes.

[5] Data considered invalid is no longer of use and may be removed from the managed flash device.

[6] Data considered valid (or still needed data) needs to be stored in the managed flash device.