SNIA
Storage Networking Industry Association

Common RAID Disk Data Format Specification

Revision 1.2

This document has been released and approved by the SNIA. The SNIA believes that the ideas, methodologies and technologies described in this document accurately represent the SNIA goals and are appropriate for wide spread distribution. Suggestion for revision should be directed to dftwg@snia.org.

SNIA Technical Position

July 28, 2006

Revision History

Revision	Date	Sections	Originator:	Comments
1.0	12/14/2004		Bill Dawkins	Original Release
1.1			Bill Dawkins	
1.10.04	11/16/2005		Bill Dawkins	Draft submitted to SNIA Technical Council
1.10.05	1/12/2006		Bill Dawkins	Minor correction – resubmitted to SNIA Technical Council
1.2	4/10/2006		Arnold Jones	Changed final published version number to 1.2.
1.2	7/28/2006		Arnold Jones	Officially published as SNIA Technical Position.

Suggestions for changes or modifications to this document should be sent to the Disk Data Format TWG at ddftwg@snia.org.

Typographical Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC2119 [http://www.ietf.org/rfc/rfc2119.txt].

Usage

TABLE OF CONTENTS

1 Introduction

In today's IT environments, there are several reasons why system administrators would wish to change the internal RAID solutions they are using. For example, many servers are shipped with a RAID solution implemented on the motherboard (ROMB). ROMB solutions allow RAID formats to be applied to the disks internal to the server. As the server's data set grows, the administrator often finds s/he needs to move to a larger direct attached storage (DAS) solution with external JBODs. The system administrator would like to move the internal disks and their data to the DAS system's external JBODs. One method of migration is to backup a RAID group, transfer the disks to the new storage system, reconfigure the disks as a new RAID group behind the new RAID controller, and restore the data from the backup device. This time consuming procedure also carries some risk of data loss. A better method would be to move the disks with data-in-place from one RAID implementation to another. Unfortunately, the different methods for storing configuration information prohibit data-in-place migration between systems from different storage vendors.

The SNIA Common RAID Disk Data Format Technical Working Group was chartered define a standard data structure describing how data is formatted across the disks in a RAID group. This specification defines the Common RAID Disk Data Format (DDF) structure. The DDF structure allows a basic level of interoperability between different suppliers of RAID technology. The Common RAID DDF structure benefits storage users by enabling data-in-place migration among systems from different vendors.

Part of the specification defines how data is distributed for many basic RAID levels. This is necessary to precisely document how data is formatted for RAID levels indicated by the DDF structure. The DDF TWG recognizes that the formats described do not represent all methods for implementing the defined RAID levels. The SNIA does not imply that specification formats represent a preferred RAID implementation. Reviewers of this specification are encouraged to suggest alternate RAID level formats for inclusion into future revisions of the specification.

The DDF data structure also allows RAID groups with vendor unique RAID formats. While vendor unique RAID formats prohibit data-in-place migration between vendors, the Common RAID DDF will be a benefit in these situations. At a minimum, when a RAID group containing a unique format is moved to a different RAID solution that does not support the format, the new system will still be able to read the DDF structure. It can identify the RAID groups containing the unique format and notify the system administrator that these RAID groups contain valid data that is not accessible by the current storage system. Potential data loss is prevented because the new RAID system will not overwrite the data without administrator confirmation.

The document is divided into the following sections:

- Section 2 is the overview;
- Section 3 describes the definitions used in this specification;
- Section 4 describes the RAID levels defined in this specification;
- Section 5 details the DDF structure.

2 Overview

2.1 Purpose

This document provides requirements for the RAID configuration Disk Data Format (DDF) structure stored on physical disks attached to RAID controllers. Configuration on Disk (COD) and Metadata are also commonly used terms for the same type of data structure. This DDF structure allows storing RAID configuration information on physical disks by different vendor implementations in a common format so the user data on physical disks is accessible independent of the RAID controller being used. Controllers are not required to store this information in the same format in their internal memory.

In the terminology of the SNIA Shared Storage Model (http://www.snia.org/tech_activities/shared_storage_model), the technical scope of the DDF is limited to the interface between a *block aggregation* implementation and *storage devices.* The DDF is stored as data structures on the *storage devices*.

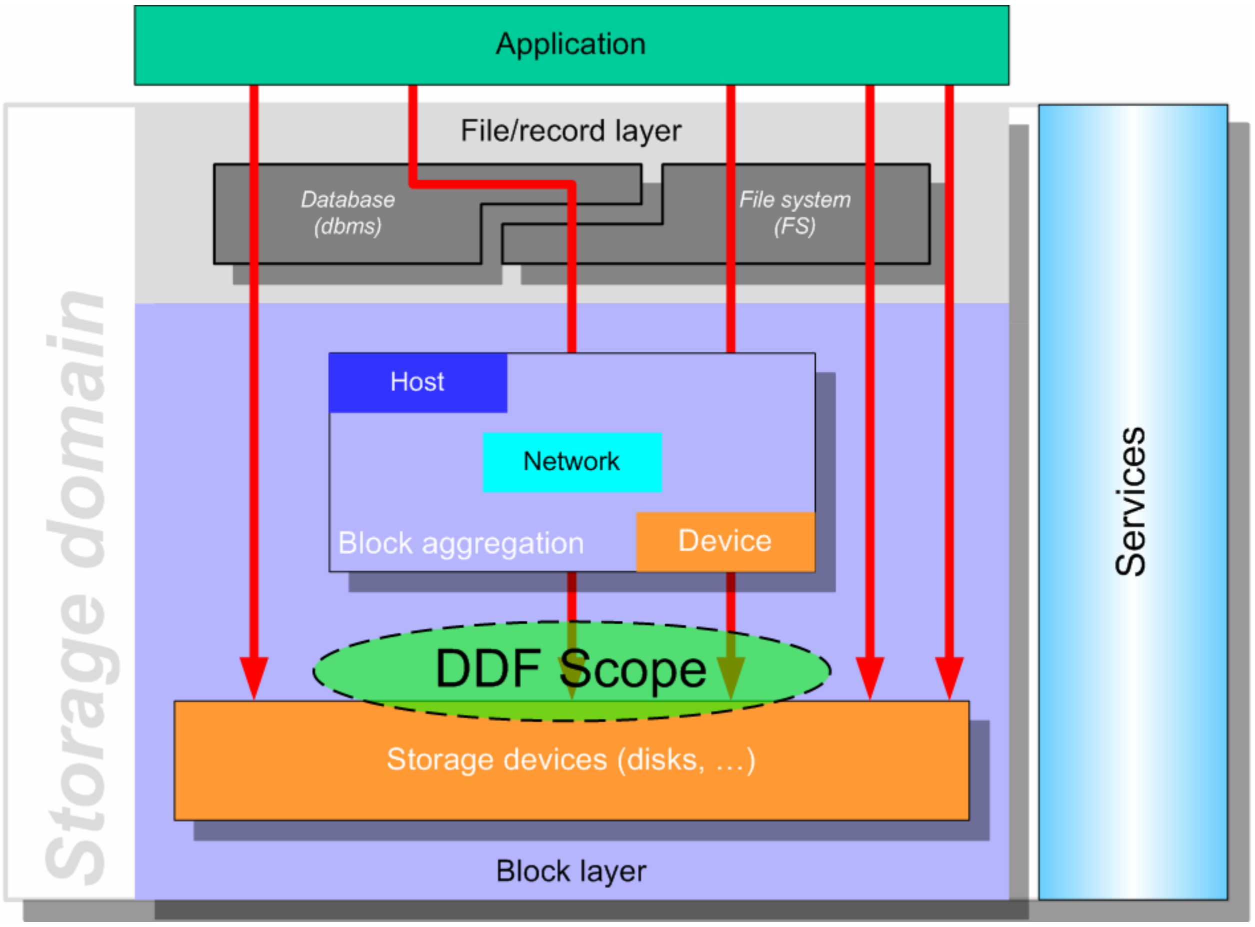

Figure 1: DDF Technical Scope

2.2 Design Considerations

Location, locality, size and contents are major considerations for the DDF structure. Details on the contents and format of the DDF structure can be found in Section 5.

2.2.1 *Location*

The Anchor Header (see Section 5.5) for the DDF structure MUST be stored at the last logical block returned by either the Identify Device or Read Capacity commands depending on the type of physical disk.

The DDF structure SHOULD be stored at the end of the physical disk next to the anchor header. Storing the DDF structure at the end of the physical disk allows the possibility of converting a single non-RAID physical disk to a RAID 1 configuration without shifting user data. Similarly, data on a member of a RAID 1 configuration with the DDF structure at the end can also be accessed using a non-RAID controller.

2.2.2 *Locality*

Locality is defined as the scope of the DDF structure. One approach is to store configuration information about all RAID groups attached to a controller on every physical disk connected to the controller. The second approach is to store configuration information of a RAID group (or virtual disk) only on the physical disks belonging to the participating in a RAID group. In other words, does the DDF structure on one RAID group have information about other RAID groups attached to the controller? This plays a role when a user wants to move a RAID group from one controller to another while keeping the RAID group intact and without causing ghost images on either controller. If the DDF structure on one RAID group contains no information about the other RAID groups and if an entire RAID group disappears due to power or cabling problems, the user should be notified about the missing RAID group. Configuration information about all RAID groups may be stored on NVRAM on the controller and provide a notification for the user. However, in case of a failed or replacement controller, the information on the RAID groups is not available.

The middle ground, chosen for the DDF structure, is to store the complete configuration information of a RAID group on each physical disk participating in the RAID group and to store a minimal amount of information about other the RAID groups and physical disks attached to the controller. This allows a controller to provide notification to the user about missing RAID groups or physical disks when the controller does not have complete information about the configuration.

2.2.3 *DDF Structure Size*

A large DDF structure size provides room for expansion in the future and still uses a negligible amount of storage. It is tempting to use a fixed large DDF structure size, however, low end solutions may not have the memory space to process large tables.

The DDF structure size is not fixed and depends on solution needs. This is done by using flexible structures where size is a function of the number of physical and virtual disks. A fixed space SHOULD be reserved on physical disks for the DDF structure to accommodate the largest DDF structure size for migration of configurations between different types and classes of solutions. Details on DDF structure size can be found in Section 5.

2.2.4 *DDF Structure Contents*

The DDF structure contains information about partitioning, RAID level, and cache parameters for each virtual disk defined. RAID group state, physical disk location information, and controller settings are among other information included in the DDF structure. The DDF structure contents are defined in detail in Section 5.

2.2.5 *DDF Structure Redundancy*

DDF structure redundancy allows recovery after configuration structure corruption or loss. Support for redundancy increases solution complexity and is considered OPTIONAL.

3 Definitions

Whenever possible, this specification uses the definitions for storage terminology provided by SNIA's "A Dictionary of Storage Networking Terminology." The dictionary can be found at http://www.snia.org/education/dictionary. This section defines terms that do not have entries in the SNIA dictionary. It also defines terms that use definitions that differ from the definition listed in the SNIA dictionary.

3.1 RAID Terms

3.1.1 *Virtual Disk (VD)*

A virtual disk is the object presented to the host level for user data storage. At least one physical disk is associated with a VD.

3.1.2 *Basic Virtual Disk (BVD)*

A basic virtual disk is a VD configured using only non-hybrid RAID levels like RAID-0, RAID5 or RAID5E. Its elements are physical disks.

3.1.3 *Secondary Virtual Disk (SVD)*

A secondary virtual disk is a VD configured using hybrid RAID levels like RAID10 or RAID50. Its elements are BVDs.

3.1.4 *Disk Grouping*

A number of physical disks can be combined into a disk group. The primary characteristic of a disk group is that all VDs created on the physical disks cannot extend to other physical disks that are not part of the group. Disk Grouping, when enforced, ensures that the contiguous address space of a VD does not extend beyond a disk group.

3.1.5 *Foreign configuration*

A configuration moved from one controller to another controller is considered a foreign configuration on the new controller unless new controller imports the configuration. Whenever a foreign configuration is detected by a controller, The Foreign_Flag MUST be set in the DDF header on the physical disks in the foreign configuration. Details of the Foreign_Flag can be found in Section 5.5.

3.1.6 *Legacy or Pass-through Disk*

Legacy (pass-through) physical disks are attached to a RAID controller and operate as if they were attached to a non-RAID controller. No DDF structure is stored on these physical disks. This feature is primarily targeted for users moving physical disks containing data from non-RAID controllers to RAID controllers.

4 RAID Levels and RAID Level Qualifiers

This section lists the RAID types and qualifiers for use in following fields used in the Configuration Record (Section 5.9):

- Primary RAID Level
- RAID Level Qualifier
- Secondary RAID Level

4.1 Primary RAID Level

Table 1 lists values used in the Primary_RAID_Level field of the Virtual Disk Configuration Record (Section 5.9.1) and the definitions of these values. The Primary_RAID_Level field MUST use the values listed in Table 1. The table defines the standard RAID levels, such as RAID 0, 1, 3, 5, etc. and some proprietary RAID types. Non-RAID types such as JBOD and concatenation are also included for completeness.

Table 1: Primary RAID Levels

Name	PRL Byte	Description
RAID-0	00	Striped array with no parity
RAID-1	01	Mirrored array
RAID-3	03	Striped array with typically non-rotating parity, optimized for long, single-threaded transfers
RAID-4	04	Striped array with typically non-rotating parity, optimized for short, multi-threaded transfers
RAID-5	05	Striped array with typically rotating parity, optimized for short, multi-threaded transfers
RAID-6	06	Similar to RAID-5, but with dual rotating parity physical disks, tolerating two physical disk failures
RAID-1E	11	>2 disk RAID-1, similar to RAID-10 but with striping integrated into array
Single Disk	0F	Single, non-arrayed disk
Concatenation	1F	Physical disks combined head to tail
RAID-5E	15	RAID-5 with hot space at end of array
RAID-5EE	25	RAID-5 with hot space integrated into array

4.2 RAID Level Qualifier

This section defines RAID Level Qualifiers (RLQ) for each Primary RAID Level as defined earlier. The RLQ field MUST be ignored for JBOD and Concatenations (PRL=0F and 1F). Examples are provided for each RLQ where columns show extents (e.g., member physical disks, partitions of member physical disks, etc.) of a VD.

4.2.1 RAID-0 Simple Striping (PRL=00, RLQ=00)

Figure 1 shows an example of simple striping (RAID-0). The standard SNIA dictionary definitions for stripe, strip, stripe depth, and extent are used by this example and following examples. For a Basic Virtual Disk, as defined by this specification, an extent MUST be a contiguous area of a physical disk. A BVD's extents MUST be of equal size but are not required to reside in the same location of each physical disk.

The example introduces the concept of *extent* and *stripe* indices for a strip. *strip* (*i, j*) represents the strip located on extent *i* in stripe *j*. The *data block* index *k* represents the offset of a given data block from the beginning of a strip. To represent a specific block in a specific extent the following notation is used:

extent_block (*k, i, j*).

To refer to a specific block in a virtual disk, the following notation is used:

virtual_block (*x*),

where *x* is the offset from the beginning of the VD. Table 2 summarizes the indices and constants used in the notation along with the any restrictions on legal values.

Table 2: Indices and Constants for Simple Striping

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	*N*-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR((*M*-1)/*L*)/*N*)
k	Offset of a data block from the beginning of a strip	0	*L*-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of blocks in a VD. *M* MUST be evenly divisible by *N.*	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	*M*-1

For RAID-0 (PRL=00, RLQ=00), the first strip of the virtual disk MUST be strip (0,0). The allocation of data MUST adhere to the following formula:

Eq. 1

virtual_block (*x*) = extent_block (MOD(*x,L*), MOD(FLOOR(*x/L*), *N*), FLOOR(FLOOR(*x/L*)/*N*)).

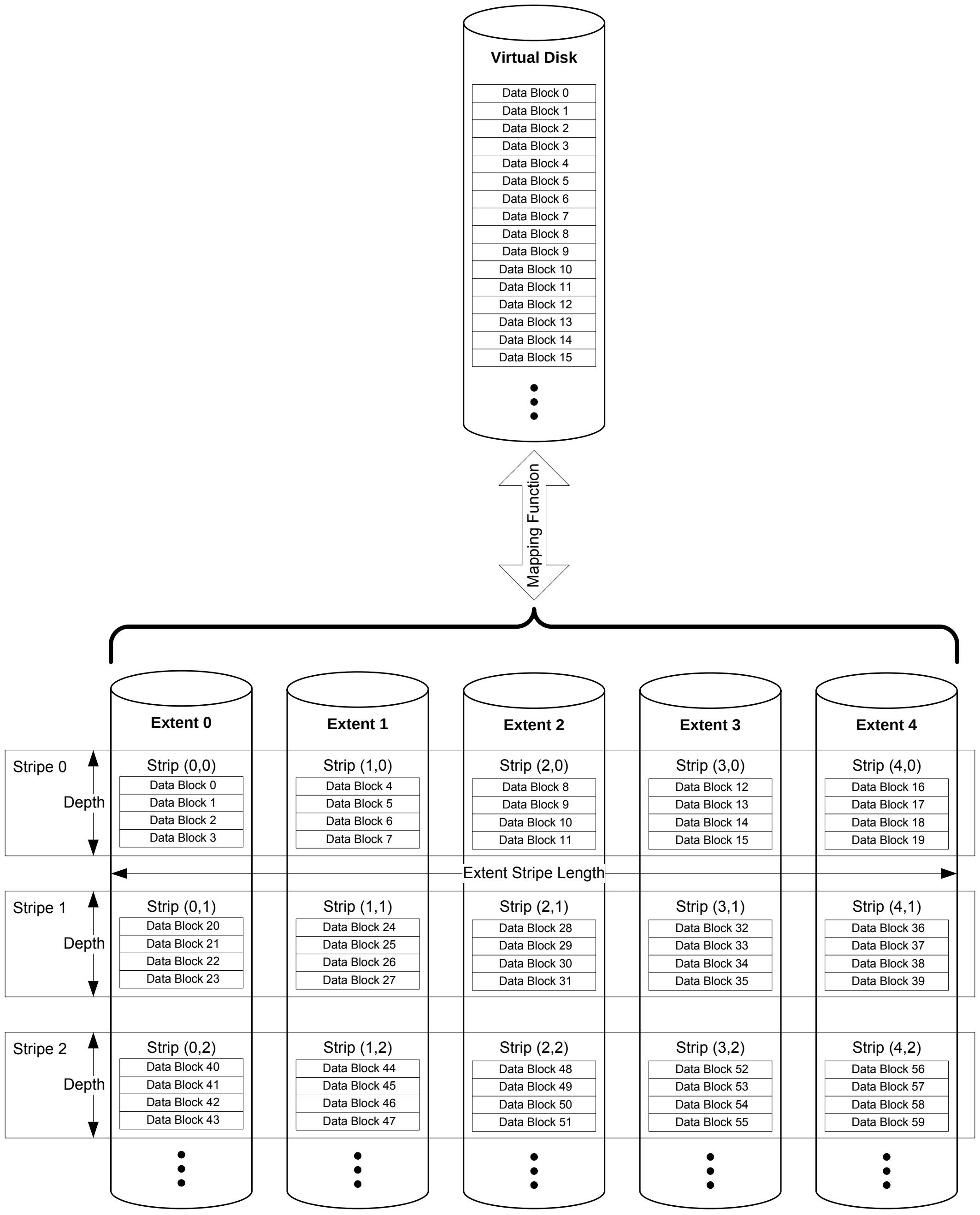

Figure 2: Simple Striping (PRL=00, RLQ=00) Example

4.2.2 RAID-1 Simple Mirroring (PRL=01, RLQ=00)

A VD with PRL=01 and RLQ=00 MUST have two and only two extents. Each extent MUST be equal to the size of the VD. Each block of the VD, virtual_block(*x*), MUST be duplicated on both extents at the same offset

Figure 3 gives an example of simple mirroring.

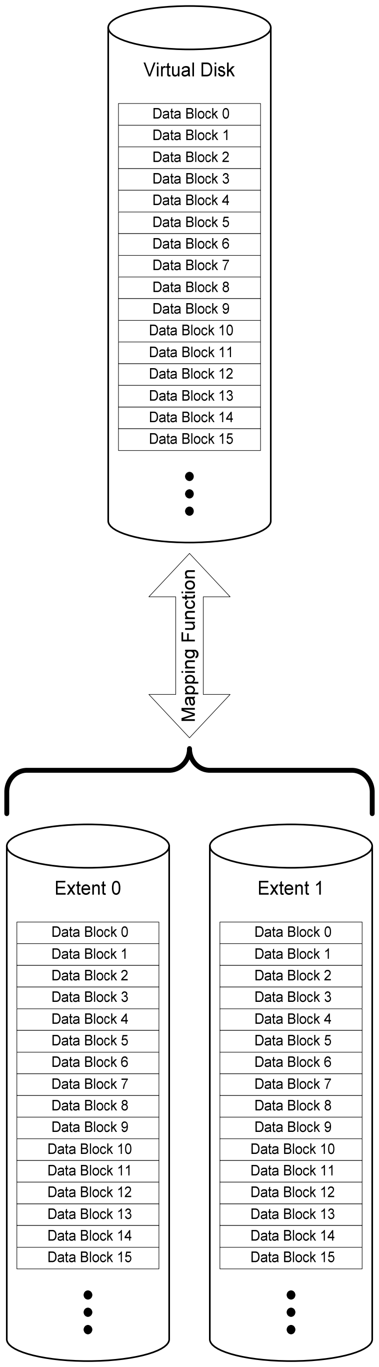

Figure 3: Simple Mirroring (PRL=01, RLQ=00) Example

4.2.3 RAID-1 Multi Mirroring (PRL=01, RLQ=01)

Multi Mirroring (PRL=01, RLQ=01) is a triple mirror. Data MUST be triple copied on three extents. Each virtual_block(x) MUST be duplicated on each extent in the VD. Figure 4 gives an example of multi mirroring.

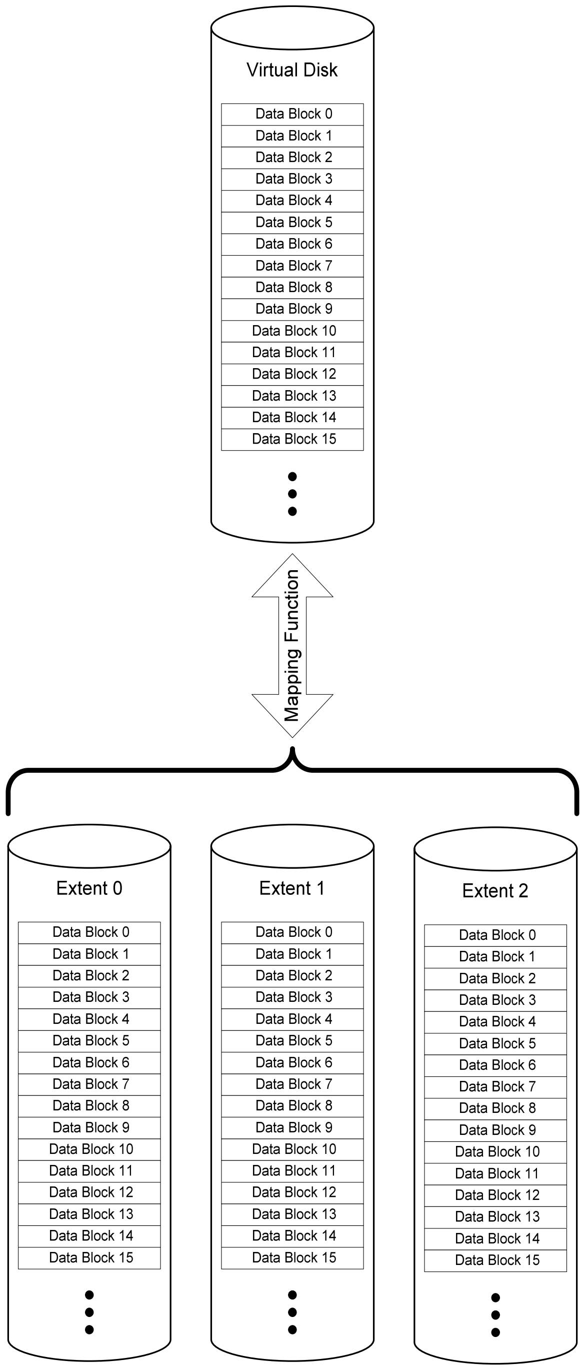

Figure 4: Multi Mirroring (PRL=01, RLQ=01) Example

4.2.4 RAID-3 Non-Rotating Parity 0 (PRL=03, RLQ=00)

Figure 5 gives an example of RAID-3 with parity contained on the first extent or Non-Rotating Parity 0. Table 3 defines the indices and constants used in the description of RAID-3.

Table 3: Indices and Constants for RAID-3 Non-Rotating Parity

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	*N*-1
j	Index of a stripe in a VD	0	*M*-1
p	Index of a data block portion	0	*N*-1
M	Number of data blocks in a VD	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	*M*-1

In a RAID-3 VD with *N* extents, a virtual data block MUST be segmented into *N*-1 block portions. The notation for data block portions is:

block_portion (*p, i, j*)

The allocation of data blocks in a RAID-3 Non-Rotating Parity 0 VD MUST adhere to the following formula:

Eq. 2

$$\text{virtual\_block}(x) = \mathop{\Big\|}_{p=0}^{N-2} \text{block\_portion}(p, p+1, x),$$

where || represents the concatenation operator.

Parity blocks MUST reside on extent 0. The values of the parity blocks must adhere to the following formula:

Eq. 3

$$\text{parity\_block}(0, 0, x) = \bigoplus_{p=0}^{N-2} \text{block\_portion}(p, p+1, x).$$

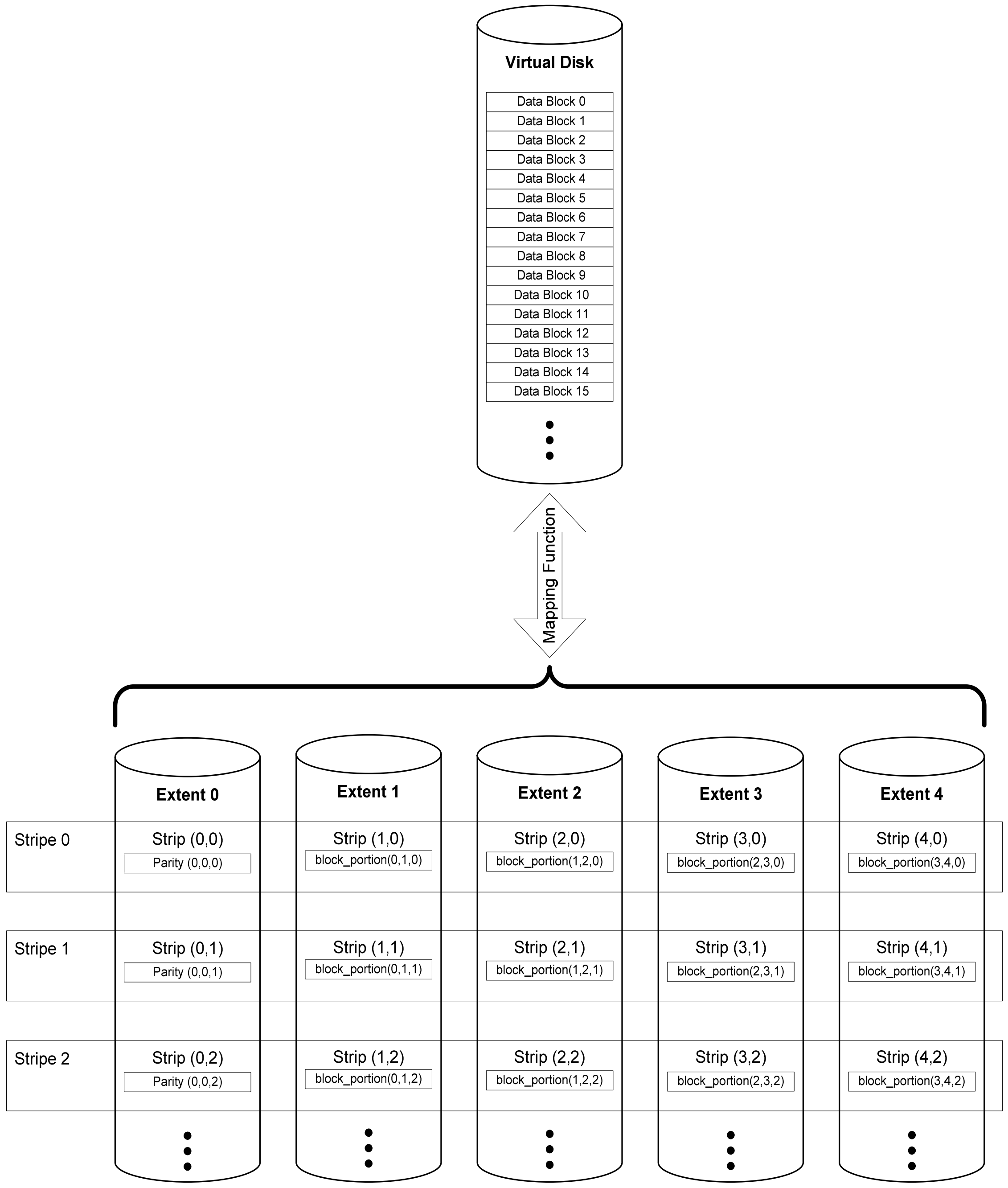

Figure 5: RAID-3 Non-Rotating Parity 0 (PRL=03, RLQ=00) Example

4.2.5 RAID-3 Non-Rotating Parity N (PRL=03, RLQ=01)

Figure 6 gives an example of a RAID-3 Non-Rotating Parity N VD. The indices and constants defined in Table 3 are valid for this type of VD.

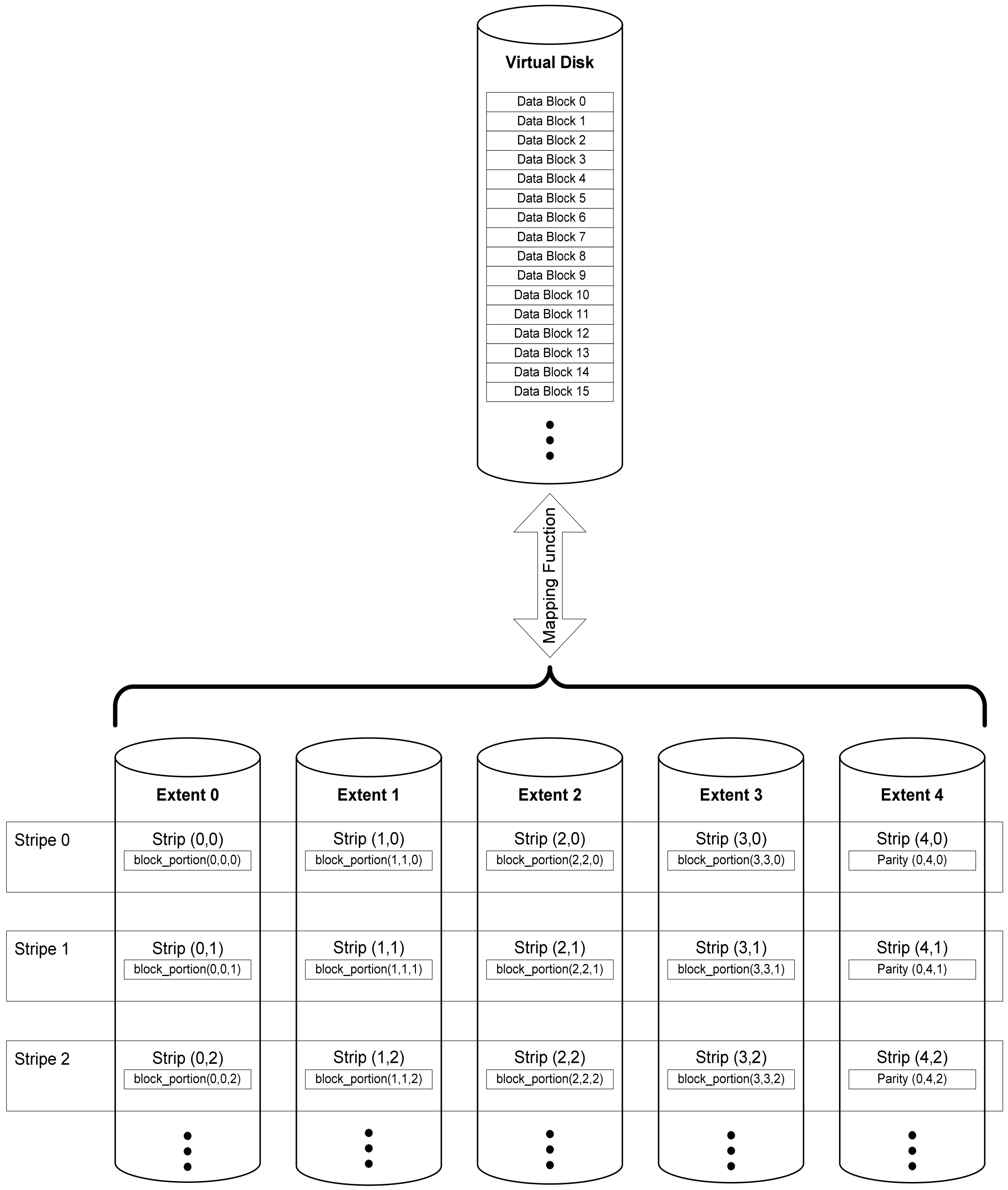

Figure 6: RAID-3 Non-Rotating Parity N (PRL=03, RLQ=01) Example

The allocation of data blocks in a RAID-3 Non-Rotating Parity N VD MUST adhere to the following formula:

Eq. 4

$$\text{virtual\_block}(x) = \mathop{\Big\|}_{p=0}^{N-2} \text{block\_portion}(p, p, x).$$

Parity Blocks MUST reside on extent *N*. The allocation of parity blocks MUST adhere to the following formula:

Eq. 5

$$\text{parity\_block}\ (0, N, x) = \bigoplus_{p=0}^{N-2} \text{block\_portion}(p, p, x).$$

4.2.6 RAID-4 Non-Rotating Parity 0 (PRL=04, RLQ=00)

Figure 7 gives an example of Non-Rotating Parity 0 (a.k.a. RAID-4).

Table 4: Indices and Constants for RAID 4 Non-Rotating Parity

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	*N*-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR((*M*-1)/*L*)/*N*-1)
k	Offset of a block from the beginning of a strip	0	*L*-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. *M* MUST be evenly divisible by *N-1.*	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	*M*-1

The allocation of data blocks in a Non-Rotating Parity 0 VD MUST adhere to the following equation:

Eq. 6

$$\text{virtual\_block}\ (x) = \text{extent\_block}\ (\text{MOD}(x,L), \text{MOD}(\text{FLOOR}(x/L), N-1) +1, \text{FLOOR}(\text{FLOOR}(x/L)/N-1)).$$

A parity block contains the parity calculated for *N*-1 data blocks. The following notation is used to represent a parity block:

$$\text{parity\_block}\ (k, i, j).$$

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 7

$$\text{parity\_block}(k, 0, j) = \bigoplus_{i=1}^{N-1} \text{extent\_block}(k, i, j).$$

For Non-Rotating Parity 0, all parity blocks MUST reside on extent 0. Thus, *i* MUST equal zero for all parity blocks.

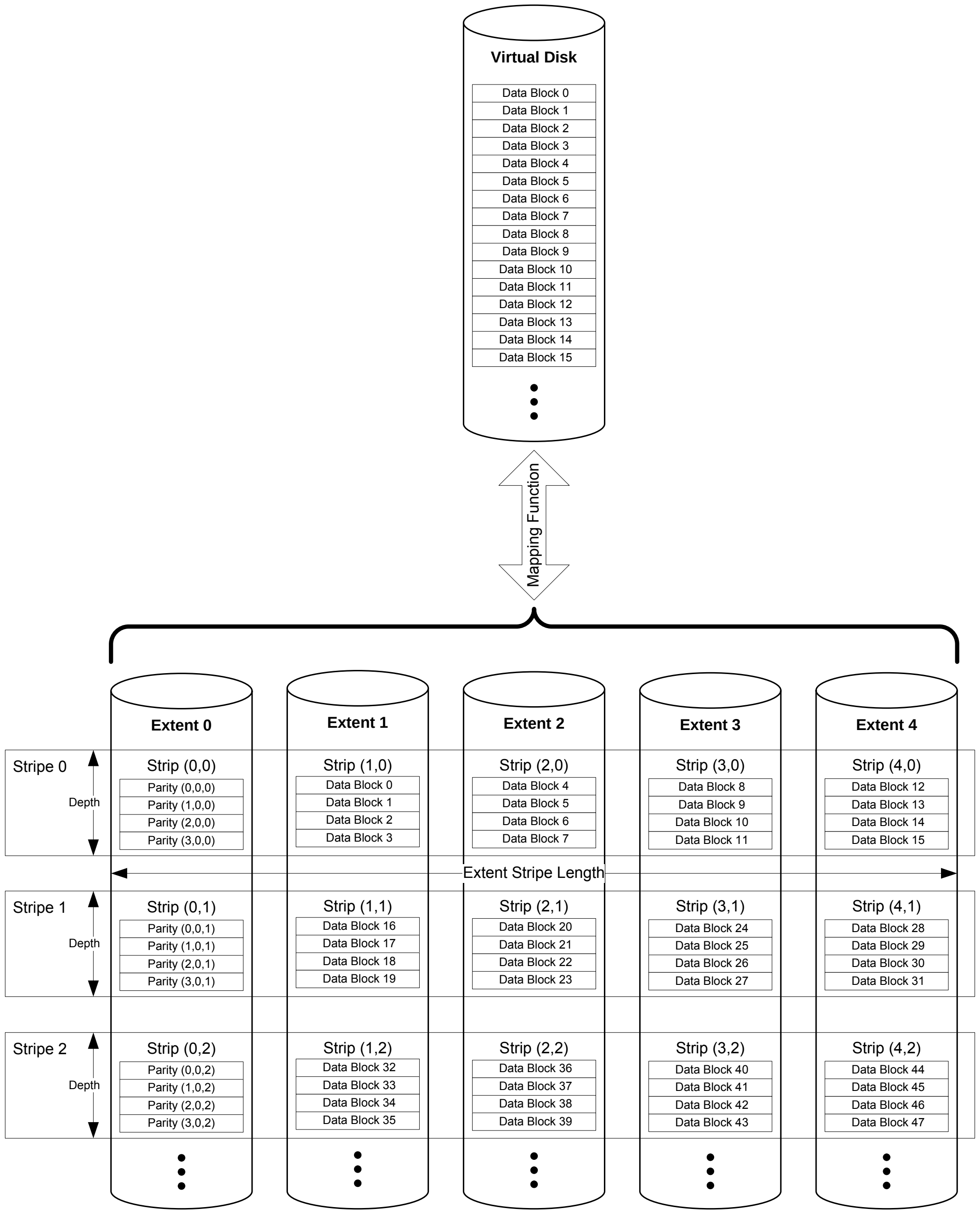

Figure 7: Non-Rotating Parity 0 (PRL=04, PRL=00) Example

4.2.7 RAID-4 Non-Rotating Parity N (PRL=04, RLQ=01)

Figure 8 gives an example on Non-Rotating Parity N (a.k.a. RAID-4). Non-Rotating Parity N differs from Non-Rotating Parity 0 in that the parity is stored in the last extent of a VD. The indices and constants of Table 4 are valid for Non-Rotating Parity N.

The allocation of data blocks in a Non-Rotating Parity N VD MUST adhere to the following formula:

Eq. 8

$$\text{virtual\_block}\,(x) = \text{extent\_block}\,(\text{MOD}(x,L),\ \text{MOD}(\text{FLOOR}(x/L),\ N\text{-}1),\ \text{FLOOR}(\text{FLOOR}(x/L)/N\text{-}1)).$$

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 9

$$\text{parity\_block}\,(k,\ N\text{-}1,\ j) = \bigoplus_{i=0}^{N-2} \text{extent\_block}(k,\ i,\ j).$$

All parity blocks MUST reside on extent *N-1*. Thus, *i* MUST equal *N-1* for all parity blocks.

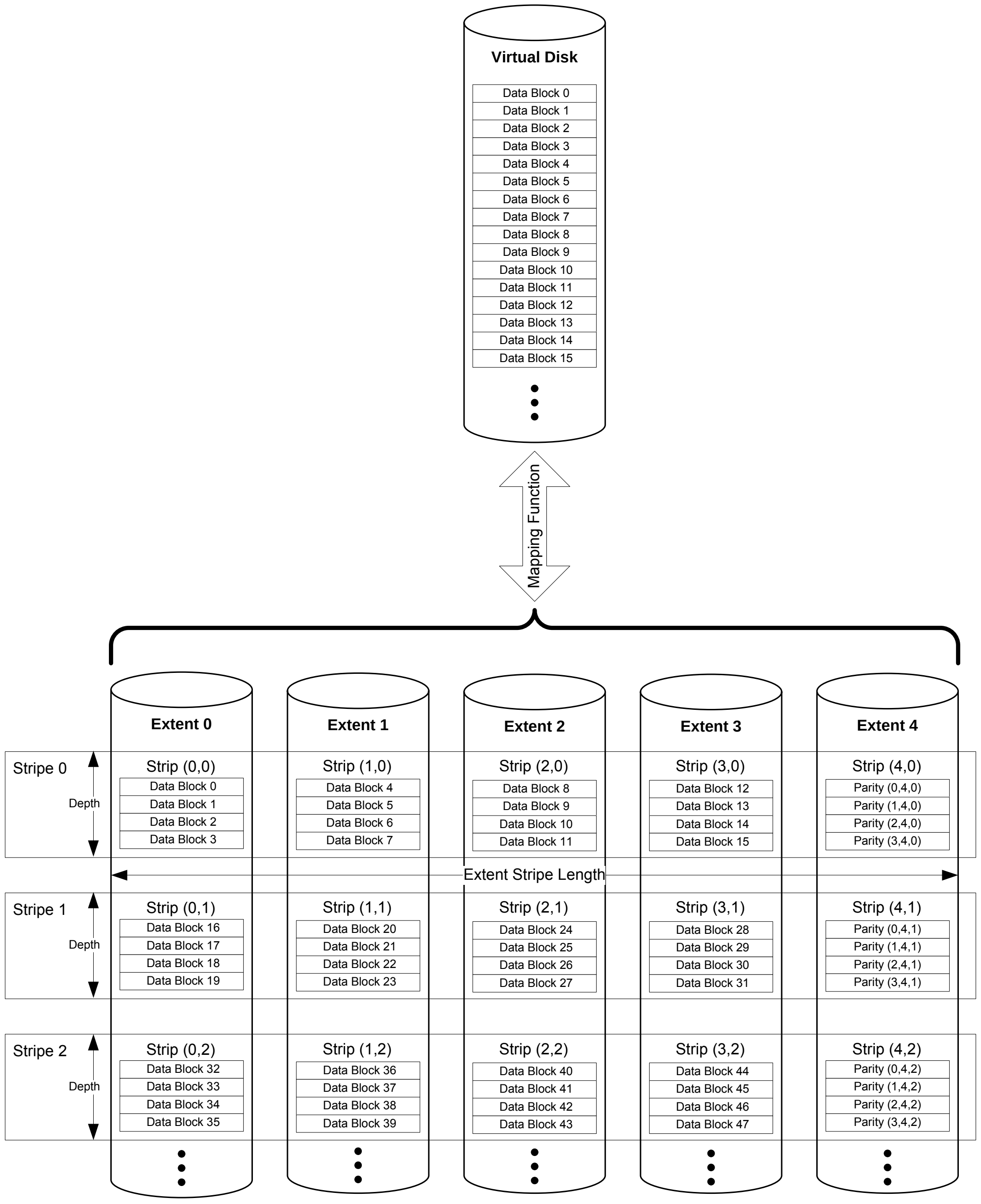

Figure 8: Non-Rotating Parity N (PRL=04, RLQ=01)

4.2.8 RAID-5 Rotating Parity 0 with Data Restart (PRL=05, RLQ=00)

Figure 9 gives an example of Rotating Parity 0 with Data Restart. Rotating Parity 0 with Data Restart is an implementation of RAID-5.

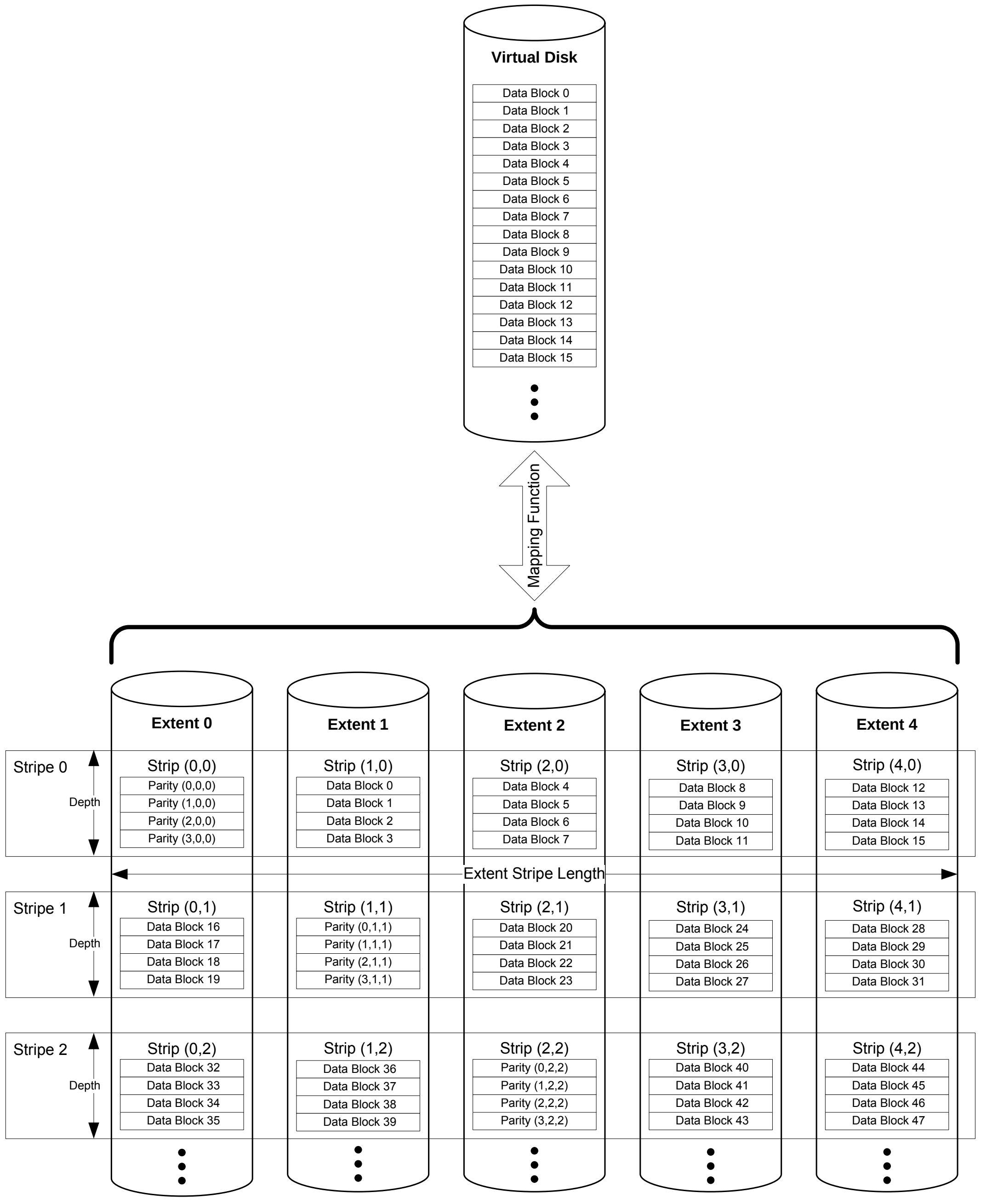

Figure 9: Rotating Parity 0 with Data Restart (PRL=05, RLQ=00)

Table 5: Indices and Constants for RAID-5 Rotating Parity 0 and N

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	N-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR((M-1)/L)/(N-1))
k	Offset of a block from the beginning of a strip	0	L-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. M MUST be evenly divisible by (N-1)*L .	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	M-1
p	Index of the extent on which the parity_blocks for a given stripe reside	0	N-1

The extent p on which the parity block for a given virtual block x resides MUST adhere to the following formula:

Eq. 10

$$p = \text{MOD}(\text{FLOOR}(\text{FLOOR}(x/L)/(N-1)),N).$$

The extent i on which a given virtual block x resides MUST adhere to the following formula:

Eq. 11

```
IF  $\text{MOD}(\text{FLOOR}(x/L),N-1) < p$
THEN  $i = \text{MOD}(\text{FLOOR}(x/L),N-1)$
ELSE  $i = \text{MOD}(\text{FLOOR}(x/L),N-1)+1$.
```

The allocation of data blocks in a Rotating Parity 0 with Data Restart VD MUST adhere to the following formula:

Eq. 12

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L),\ i,\ \text{FLOOR}(\text{FLOOR}(x/L)/N-1)$$

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 13

$$\text{parity\_block}(k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block}(k, i, j).$$

4.2.9 RAID-5 Rotating Parity N with Data Restart (PRL=05, RLQ=02)

Figure 10 gives an example of an implementation of RAID-5 called Rotating Parity N with Data Restart. The indices and constants listed in Table 5 are valid for this type of RAID.

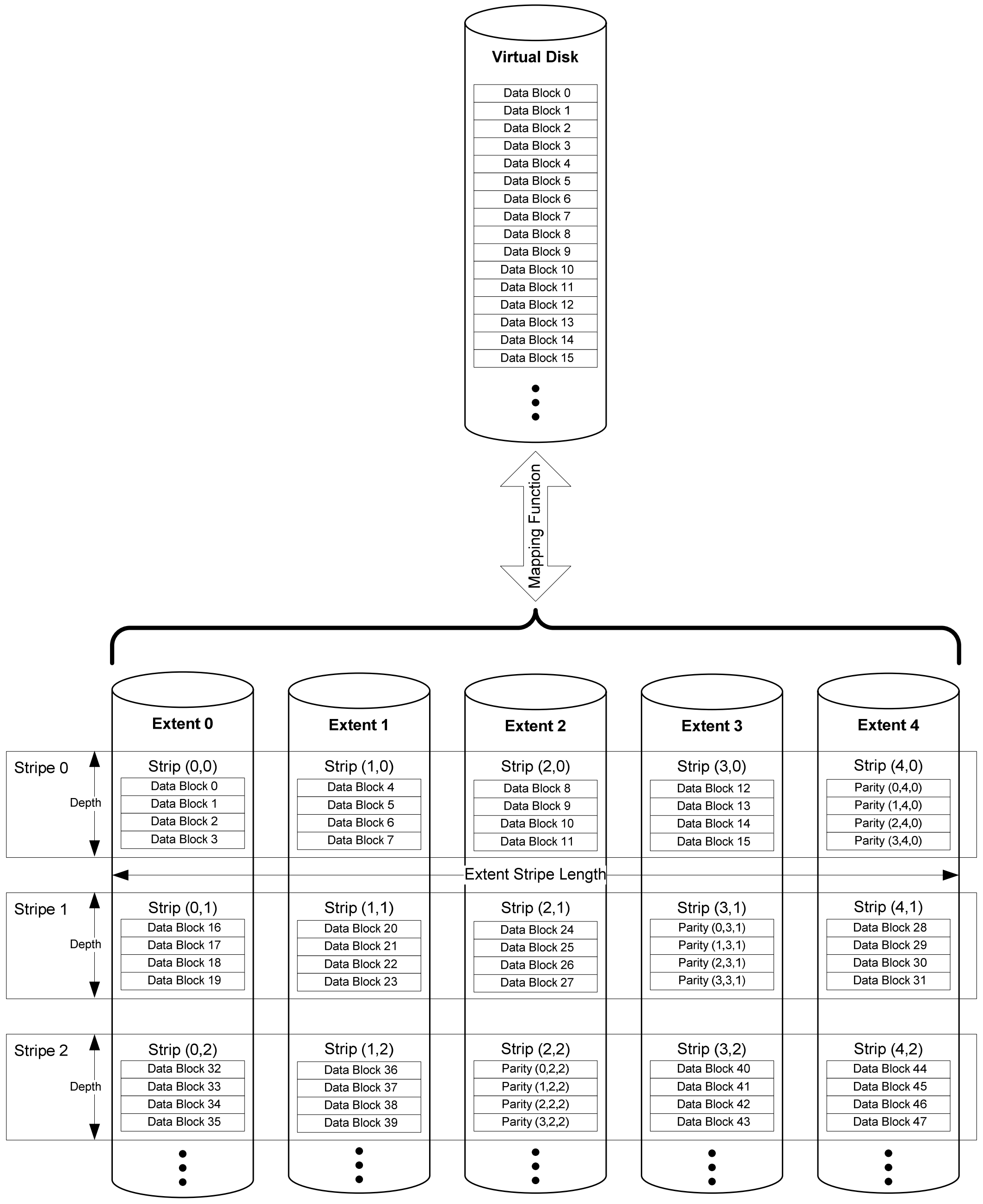

Figure 10: Rotating Parity N with Data Restart (PR=05, RLQ=02)

The extent *p* on which the parity block for a given virtual block *x* resides MUST adhere to the following formula:

Eq. 14

$$p = (N\text{-}1)\text{-MOD}(\text{FLOOR}(\text{FLOOR}(x/L)/(N\text{-}1)),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 15

IF MOD(FLOOR(x/L),N-1) < p

THEN i = MOD(FLOOR(x/L),N-1)

ELSE i = MOD(FLOOR(x/L),N-1)+1.

The allocation of data blocks in a Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 16

virtual_block (x) = extent_block (MOD(x/L), i, FLOOR(FLOOR(x/L)/N-1)

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 17

$$\text{parity\_block}(k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block}(k, i, j).$$

4.2.10 RAID-5 Rotating Parity N with Data Continuation (PRL=05, RLQ=03)

Figure 11 gives an example of RAID-5 implemented with Rotating Parity N with Data Continuation. The indices and constants given in Table 5 also apply to the formulas given below for Rotating Parity N with Data Continuation.

The extent p on which the parity block for a given virtual block x resides MUST adhere to the following formula:

Eq. 18

p = (N-1)-MOD(FLOOR(FLOOR(x/L)/(N-1)),N).

The extent i on which a given virtual block x resides MUST adhere to the following formula:

Eq. 19

i = MOD(MOD(FLOOR(x/L),(N-1))+p+1),N).

The allocation of data blocks in a Rotating Parity N with Data Continuation VD MUST adhere to the following formula:

Eq. 20

virtual_block (x) = extent_block (MOD(x/L), i, FLOOR(FLOOR(x/L)/N-1)

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 21

$$\text{parity\_block}\ (k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block}\ (k, i, j).$$

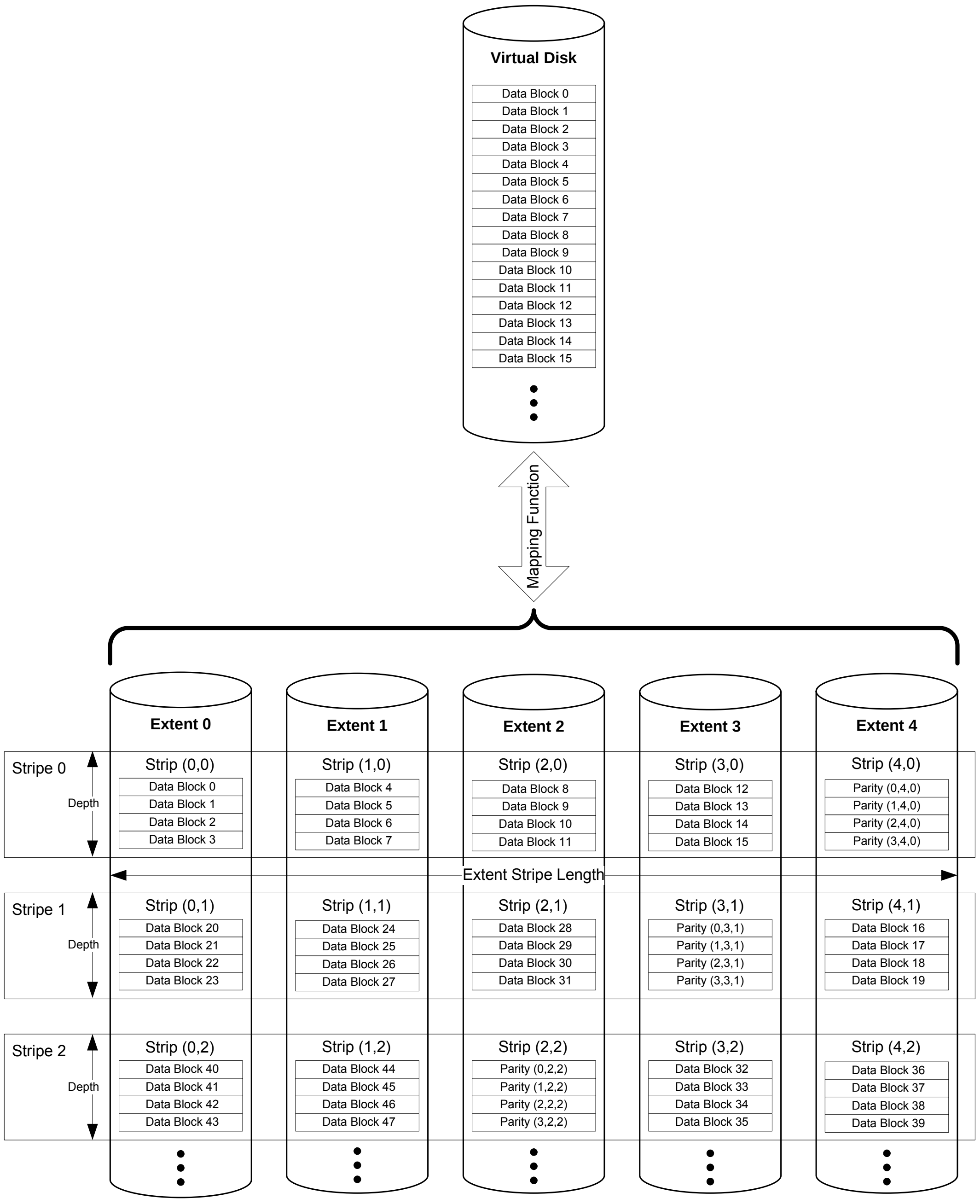

Figure 11: Rotating Parity N with Data Continuation (PRL=05, RLQ=03)

4.2.11 RAID-5E Rotating Parity 0 with Data Restart (PRL=15, RLQ=00)

Figure 12 gives an example of RAID-5E implemented with Rotating Parity 0 with Data Restart. RAID-5E has hot space at the end of each extent. In the event of an extent failure, the hot space on the remaining extents is used to rebuild and re-stripe the data in a manner that the remaining extents become a RAID-5 VD. Table 6 gives the indices and constants used to describe the data layout of this type of RAID in the following formulas.

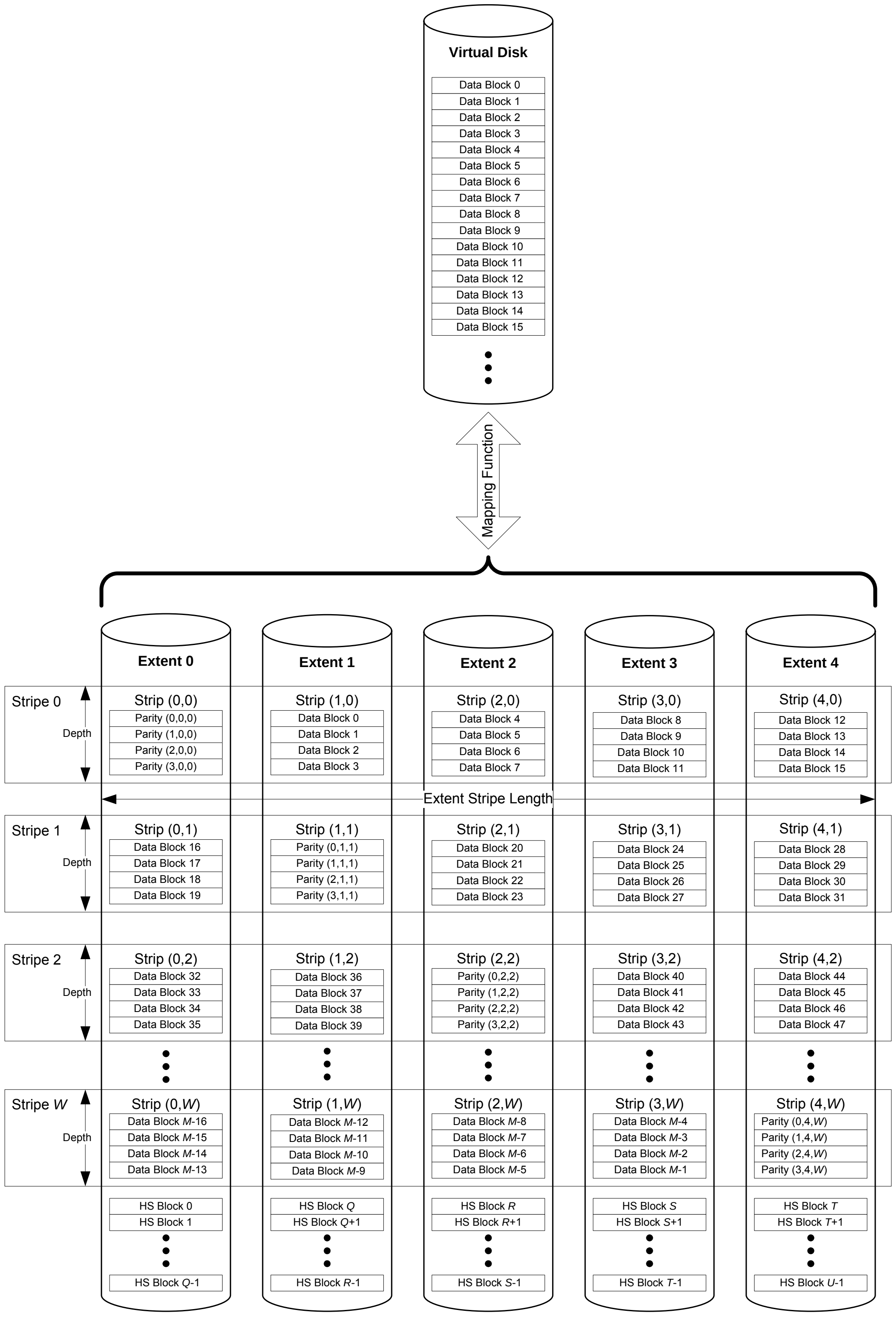

Figure 12: RAID-5E Rotating Parity 0 with Data Restart (PRL-15, RLQ=00)

Table 6: Indices and Constants for RAID-5E

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	*N*-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR((*M*-1)/L)/(*N*-1))
k	Offset of a block from the beginning of a strip	0	*L*-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. *M* MUST be evenly divisible by (*N-1)*L*	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	*M*-1
p	Index of the extent on which the parity_blocks for a given stripe reside	0	*N*-1
U	Number of hot space blocks	≥ ((*M*/(*N*-1)) * *N*) - *M*	≥ ((*M*/(*N*-1)) * *N*) - *M*
W	Index of the last stripe containing data blocks	FLOOR(FLOOR((*M*-1)/L)/(*N*-1))	FLOOR(FLOOR((*M*-1)/L)/(*N*-1))

The extent *p* on which the parity block for a given virtual block *x* resides MUST adhere to the following formula:

Eq. 22

p = MOD(FLOOR(FLOOR(x/L)/(*N*-1)),*N*).

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 23

IF MOD(FLOOR(*x/L*),*N*-1) < *p*

THEN *i* = MOD(FLOOR(*x/L*),*N*-1)

ELSE *i* = MOD(FLOOR(*x/L*),*N*-1)+1.

The allocation of data blocks in a RAID-5E Rotating Parity 0 with Data Restart VD MUST adhere to the following formula:

Eq. 24

virtual_block (*x*) = extent_block (MOD(*x/L*), *i*, FLOOR(FLOOR(*x/L*)/*N*-1)

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 25

$$\text{parity\_block}(k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block}(k, i, j).$$

The number of hot space blocks *U* MUST adhere to the following formula:

Eq. 26

$$U \geq ((M/(N\text{-}1)) * N) - M$$

Hot space blocks MUST begin at offset *M* from the beginning of the VD. The total number of blocks in the VD MUST equal *M+U*. The hot space blocks MUST be evenly distributed across all extents. All hot space blocks on an extent MUST reside at the end of the extent.

In the event of an extent failure, the controller MUST reallocate the data as described in Section 4.2.8. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5E VD reduced by one.

4.2.12 RAID-5E Rotating Parity N with Data Restart (PRL=15, RLQ=02)

Figure 13 gives an example of a RAID-5E implementation utilizing Rotating Parity N with Data Restart. The indices and constants listed in Table 6 are valid for this type of RAID.

The extent *p* on which the parity block for a given virtual block *x* resides MUST adhere to the following formula:

Eq. 27

$$p = (N\text{-}1)\text{-MOD}(\text{FLOOR}(\text{FLOOR}(x/L)/(N\text{-}1)),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 28

```
IF  $\text{MOD}(\text{FLOOR}(x/L),N\text{-}1) < p$
THEN  $i = \text{MOD}(\text{FLOOR}(x/L),N\text{-}1)$
ELSE  $i = \text{MOD}(\text{FLOOR}(x/L),N\text{-}1)+1$.
```

The allocation of data blocks in a RAID-5E Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 29

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L), i, \text{FLOOR}(\text{FLOOR}(x/L)/N\text{-}1)$$

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 30

$$\text{parity\_block}(k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block}(k, i, j).$$

The number of hot space blocks *U* MUST adhere to the following formula:

Eq. 31

$$U \geq ((M/(N-1)) * N) - M$$

Hot space blocks MUST begin at offset *M* from the beginning of the VD. The total number of blocks in the VD MUST equal *M+U*. The hot space blocks MUST be evenly distributed across all extents. All hot space blocks on an extent MUST reside at the end of the extent.

In the event of an extent failure, the controller MUST reallocate the data as described in Section 4.2.9. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5E VD reduced by one.

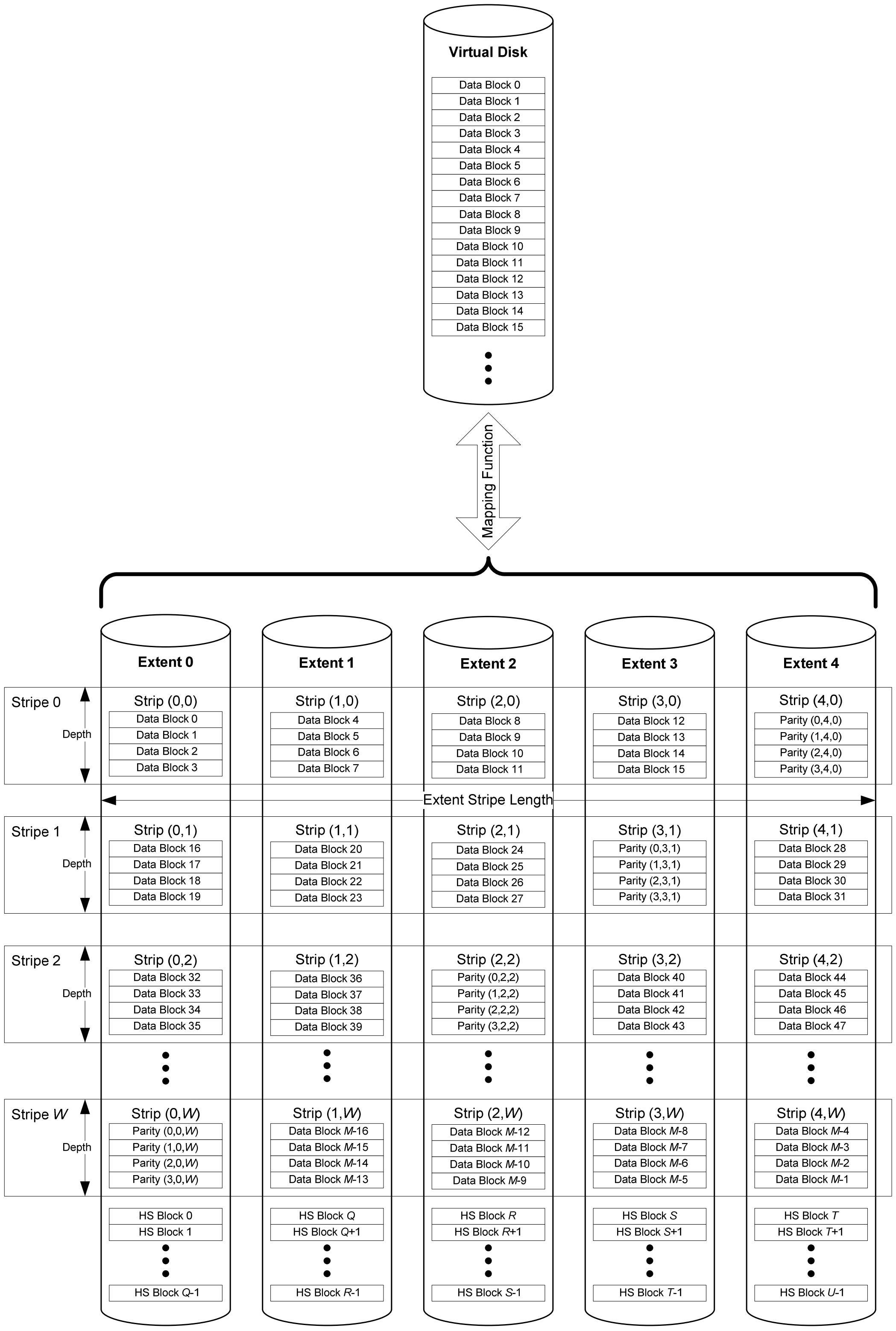

Figure 13: RAID-5E Rotating Parity N with Data Restart (PRL=15, RLQ=02)

4.2.13 RAID-5E Rotating Parity N with Data Continuation (PRL=15, RLQ=03)

Figure 14 given an example of a RAID-5E volume implemented utilizing Rotating Party N with Data Continuation. The indices and constants listed in Table 6 are valid for this type of RAID.

The extent *p* on which the parity block for a given virtual block *x* resides MUST adhere to the following formula:

Eq. 32

$$p = (N\text{-}1)\text{-MOD(FLOOR(FLOOR(}x/L)/(N\text{-}1)),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 33

$$i = \text{MOD(MOD(FLOOR(}x/L),(N\text{-}1))+p+1),N).$$

The allocation of data blocks in a RAID-5E Rotating Parity N with Data Continuation VD MUST adhere to the following formula:

Eq. 34

$$\text{virtual\_block }(x) = \text{extent\_block (MOD(}x/L), i, \text{FLOOR(FLOOR(}x/L)/N\text{-}1)$$

The values of the parity blocks MUST be calculated according to the following formula:

Eq. 35

$$\text{parity\_block }(k, p, j) = \bigoplus_{i=0}^{N-1, i \neq p} \text{extent\_block }(k, i, j).$$

The number of hot space blocks *U* MUST adhere to the following formula:

Eq. 36

$$U \geq ((M/(N\text{-}1)) * N) - M$$

Hot space blocks MUST begin at offset *M* from the beginning of the VD. The total number of blocks in the VD MUST equal *M+U*. The hot space blocks MUST be evenly distributed across all extents. All hot space blocks on an extent MUST reside at the end of the extent.

In the event of an extent failure, the controller MUST reallocate the data as described in Section 4.2.10. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5E VD reduced by one.

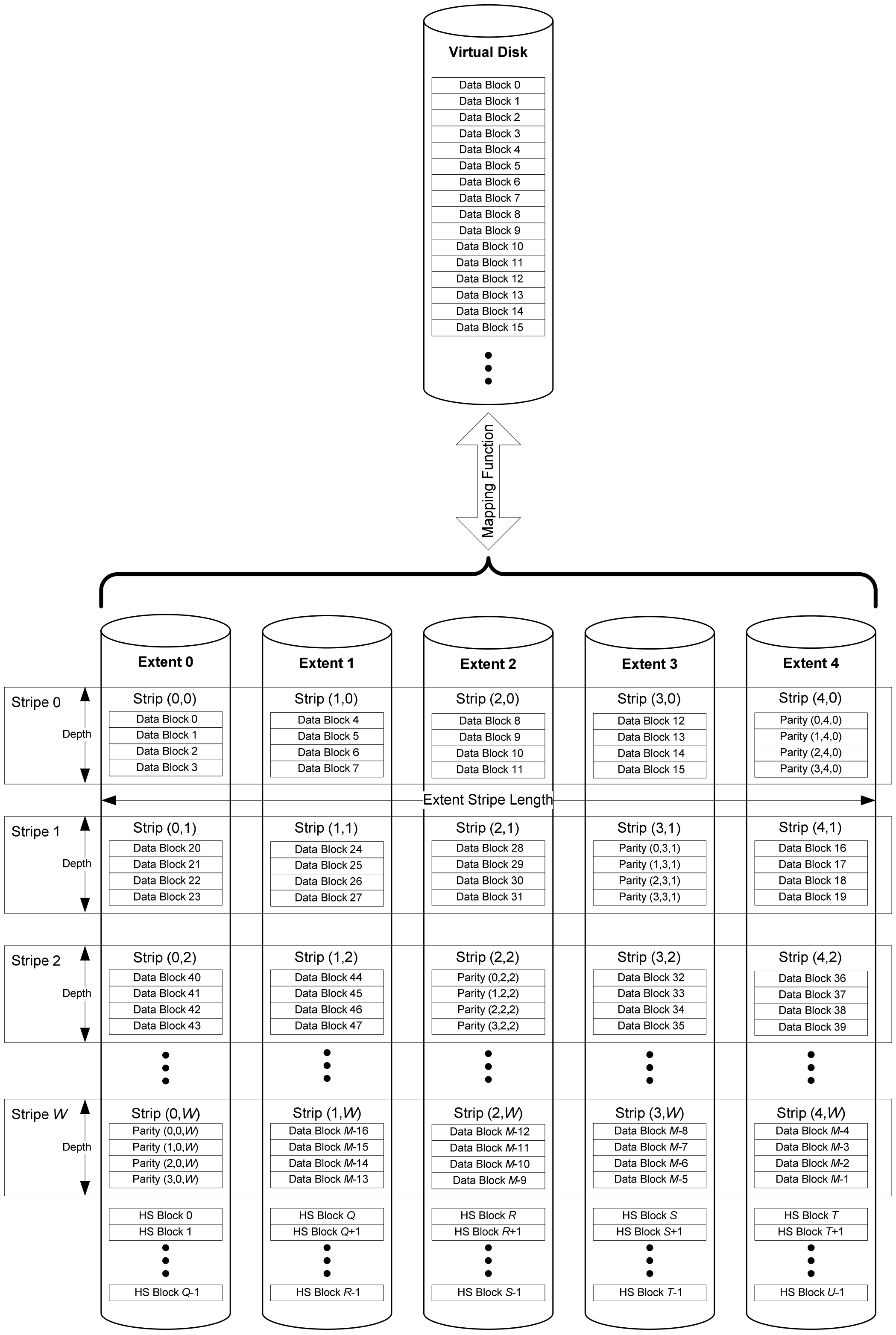

Figure 14: RAID-5E Rotating Parity N with Data Continuation (PRL=15, RLQ=03)

4.2.14 RAID-5EE Rotating Parity 0 with Data Restart (PRL=25, RLQ=00)

Figure 15 gives an example of RAID-5EE implemented with Rotating Parity 0 with Data Restart. RAID-5EE has hot space distributed across the extents. In the event of an extent failure, the hot space on the remaining extents is used to rebuild and re-stripe the data in a manner that the remaining extents become a RAID-5 VD.

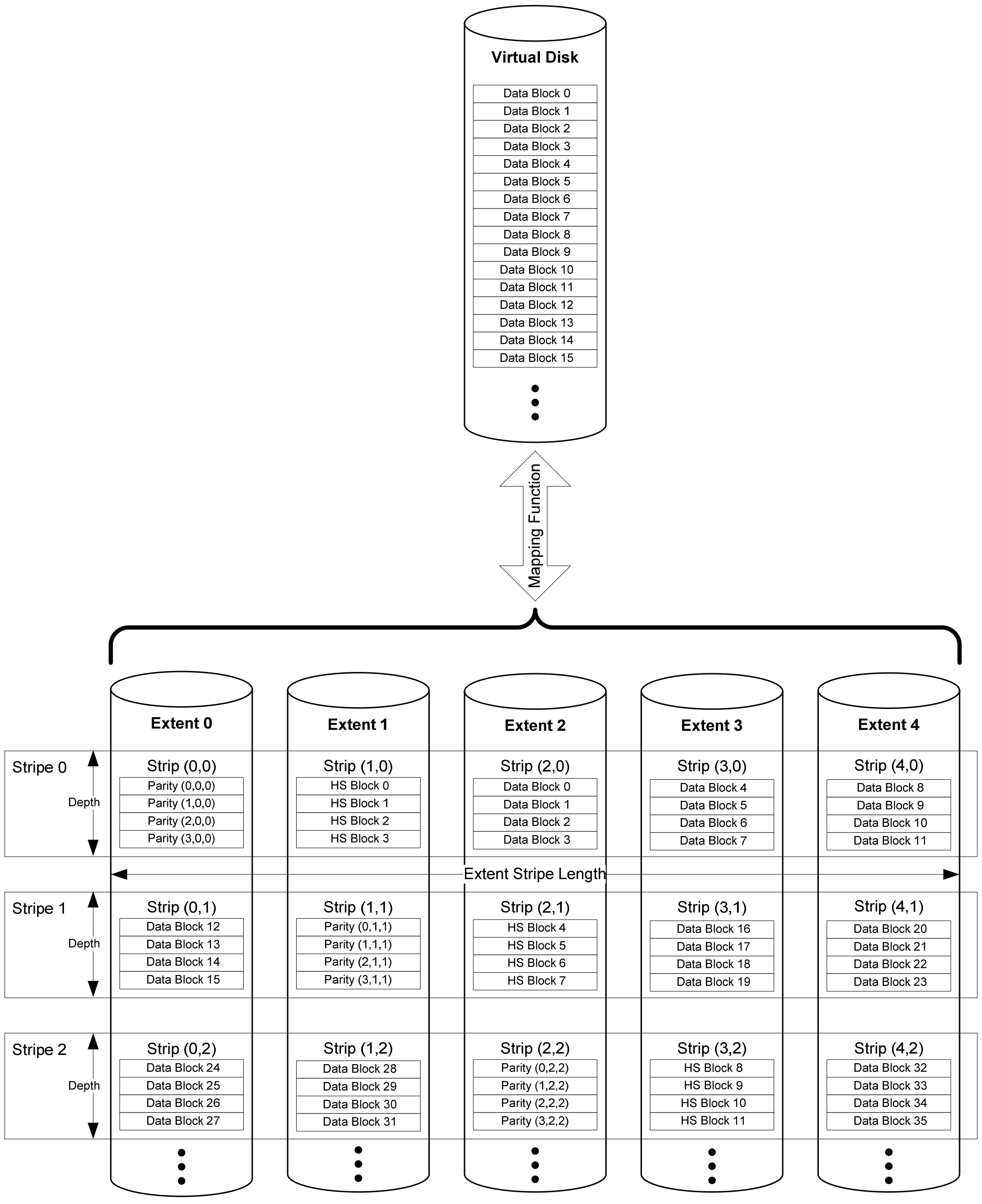

Figure 15: RAID-5EE Rotating Parity 0 with Data Restart (PRL=25, RLQ=00)

Table 7 gives the indices and constants used to describe the data layout of this type of RAID in the following formulas.

Table 7: Indices and Constants for RAID-5EE

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	*N*-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR(*M*-1/*L*)/(*N*-2))
k	Offset of a block from the beginning of a strip	0	*L*-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. *M* MUST be evenly divisible by (*N-2)*L*	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of a VD	0	*M*-1
p	Index of the extent on which the parity_blocks for a given stripe reside	0	*N*-1
h	Index of the extent on which the hot space for a given strip resides	0	*N-1*

The stripe *j* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 37

$$j = (\text{FLOOR}(\text{FLOOR}(x/L)/(N\text{-}2))$$

The extent *p* on which the parity for a given stripe *j* resides MUST adhere to the following formula:

Eq. 38

$$p = \text{MOD}(j,N).$$

The extent *h* on which hot space for a given stripe *j* resides MUST adhere to the following formula:

Eq. 39

$$h = \text{MOD}((p+1),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 40

```
IF MOD(FLOOR(x/L),N-2) < p
THEN
    IF MOD(FLOOR(x/L),N-2) < h
```

```
            THEN
                    $i$ = MOD(FLOOR($x/L$),$N$-2)
            ELSE
                    i = MOD(FLOOR($x/L$),$N$-2)+1
    ELSE
            $i$ = MOD(FLOOR($x/L$,1),$N$-2)+2
```

The allocation of data blocks in a RAID-5EE Rotating Parity 0 with Data Restart VD MUST adhere to the following formula:

Eq. 41

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L), i, j)$$

The values of the party blocks MUST be calculated according to the following formula:

Eq. 42

$$\text{parity\_block}(k, p, j) = \bigotimes_{i=0}^{N-1, i\neq p, i\neq h} \text{extent\_block}(k, i, j).$$

In the event of an extent failure, the controller MUST reallocate the data as describe in Section 4.2.8. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5EE volume reduced by one.

4.2.15 RAID-5EE Rotating Parity N with Data Restart (PRL=25, RLQ=02)

Figure 16 gives an example of RAID-5EE implemented with Rotating Parity N with Data Restart. The indices and constants given in Table 7 are used in the description below.

The stripe *j* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 43

$$j = (\text{FLOOR}(\text{FLOOR}(x/L)/(N-2)).$$

The extent *p* on which the parity for a given stripe *j* resides MUST adhere to the following formula:

Eq. 44

$$p = (N-1) - \text{MOD}(j,N).$$

The extent *h* on which hot space for a given stripe *j* resides MUST adhere to the following formula:

Eq. 45

$$h = \text{MOD}((p-1),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 46

```
IF $MOD(FLOOR(x/L),N-2) < h$
THEN
    IF $MOD(FLOOR(x/L),N-2) < p$
    THEN
        $i = MOD(FLOOR(x/L),N-2)$
    ELSE
        $i = MOD(FLOOR(x/L),N-2)+1$
ELSE
    $i = MOD(FLOOR(x/L,1),N-2)+2$
```

The allocation of data blocks in a RAID-5EE Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 47

$$\text{virtual\_block}(x) = \text{extent\_block}(MOD(x/L), i, j).$$

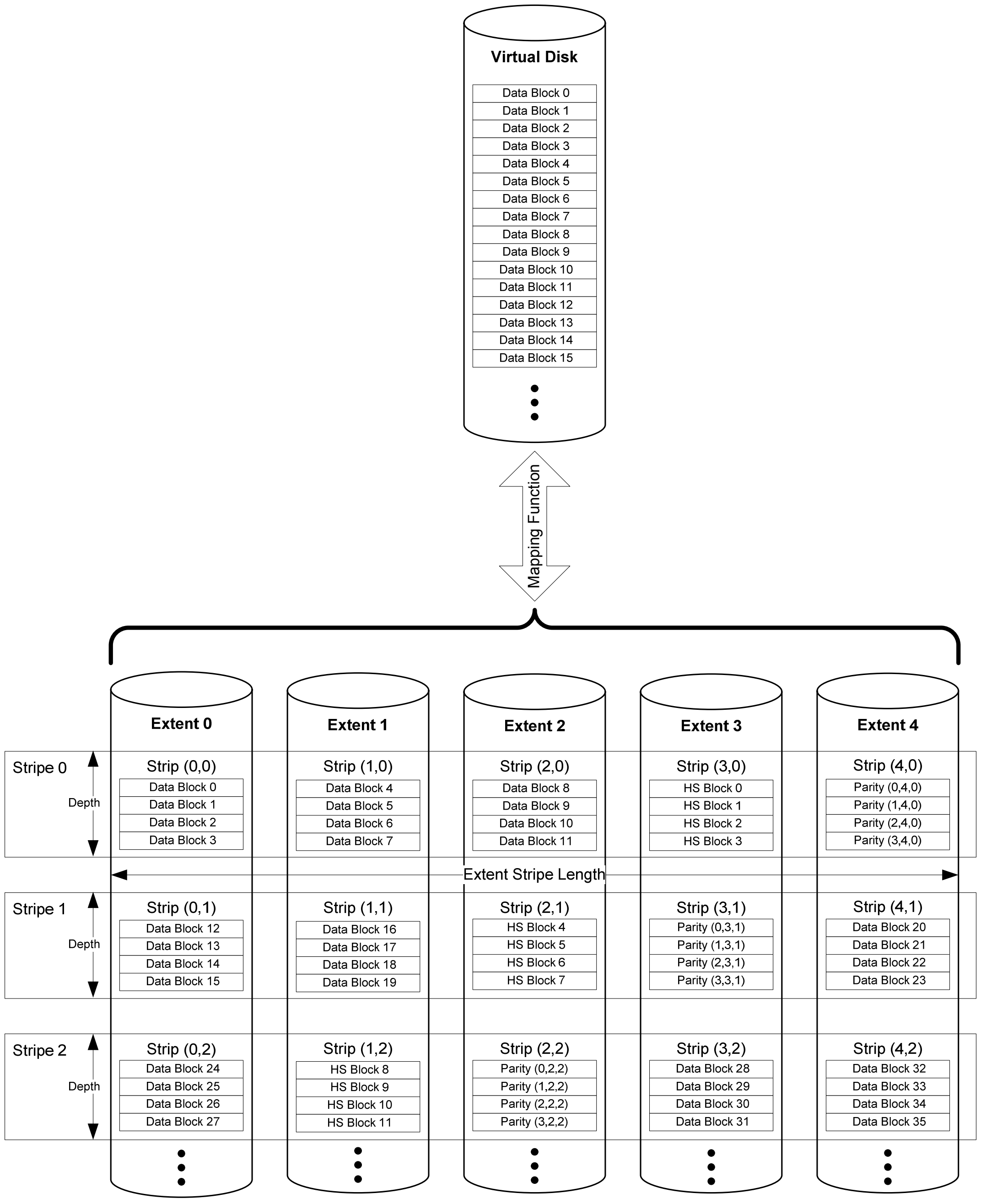

Figure 16: RAID-5EE Rotating Parity N with Data Restart (PRL=25, RLQ=02)

The values of the party blocks MUST be calculated according to the following formula:

Eq. 48

$$\text{parity\_block}\ (k,\ p,\ j) = \bigotimes_{i=0}^{N-1, i \neq p, i \neq h} \text{extent\_block}\ (k,\ i,\ j).$$

In the event of an extent failure, the controller MUST reallocate the data as describe in Section 4.2.9. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5EE volume reduced by one.

4.2.16 RAID-5EE Rotating Parity N with Data Continuation (PRL=25, RLQ=03)

Figure 17 gives an example of RAID-5EE implemented with Rotating Party N with Data Continuation. The indices and constants given in Table 7 are used in the description below.

The strip *j* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 49

$$j = (\text{FLOOR}(\text{FLOOR}(x/L)/(N\text{-}2)).$$

The extent *p* on which the parity for a given stripe *j* resides MUST adhere to the following formula:

Eq. 50

$$p = (N\text{-}1) - \text{MOD}(j,N).$$

The extent *h* on which hot space for a given stripe *j* resides MUST adhere to the following formula:

Eq. 51

$$h = \text{MOD}((p\text{-}1),N).$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 52

$$i = \text{MOD}(\text{MOD}(\text{FLOOR}(x/L),N\text{-}2)+p+1,\text{N}).$$

The allocation of data blocks in a RAID-5EE Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 53

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L), i, j).$$

The values of the party blocks MUST be calculated according to the following formula:

Eq. 54

$$\text{parity\_block}\ (k, p, j) = \bigotimes_{i=0}^{N-1, i\neq p, i\neq h} \text{extent\_block}\ (k, i, j).$$

In the event of an extent failure, the controller MUST reallocate the data as describe in Section 4.2.10. The number of extents used in the resulting RAID-5 VD MUST be equal to the number of extents used in the RAID-5EE volume reduced by one.

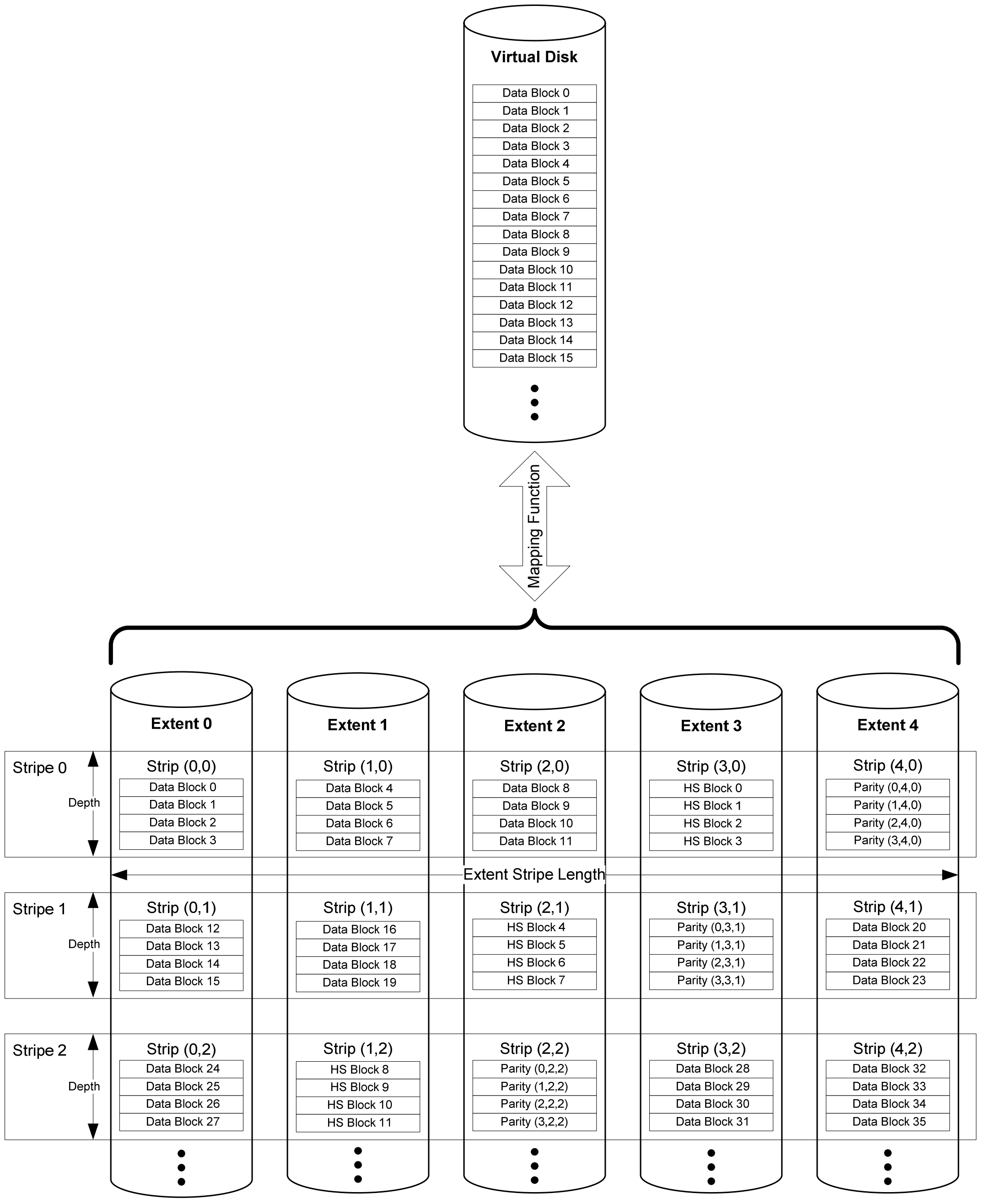

Figure 17: RAID-5EE Rotating Parity N with Data Continuation (PRL=25, RLQ=03)

4.2.17 Integrated Adjacent Stripe Mirroring (PRL= 11, RLQ=00)

Figure 18 gives an example of Integrated Adjacent Strip Mirroring. Table 8 gives the indices and constants used to describe the data layout of this type of RAID in the following formulas.

Table 8: Indices and Constants for Integrated Adjacent Strip Mirroring

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD where the primary copy of a virtual_block is stored.	0	*N*-1
j	Index of a stripe in a VD	0	*(M/N)*-1
k	Offset of a block from the beginning of a strip	0	*L*-1
L	Size of a strip in blocks	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. *M* MUST be evenly divisible by *N-1.*	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD	N/A (a fixed value)	N/A (a fixed value)
q	Index of an extent of a VD where the mirrored copy of a virtual_block is stored.	0	*N*-1
r	Index of the stripe where the mirrored copy of a virtual_block is stored	0	*(M/N)*-1
x	Offset of a data block from the beginning of a VD	0	*M*-1

In an Integrated Adjacent Stripe Mirroring VD, each virtual_block(*x*) MUST be stored in two locations. These two locations are referred to as extent_block (*k, i, j*) and extent_block' (*k, q, r*). The allocation of data in an Integrated Adjacent Stripe Mirroring VD MUST adhere to the following formulas:

Eq. 55

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L), \text{MOD}(\text{FLOOR}(x/L)*2, N), \text{FLOOR}(\text{FLOOR}(x/L)*2/N))$$

Eq. 56

$$\text{virtual\_block}(x) = \text{extent\_block}'(\text{MOD}(x/L), \text{MOD}((\text{FLOOR}(x/L)*2)+1, N), \text{FLOOR}(((\text{FLOOR}(x/L)*2)+1)/N)$$

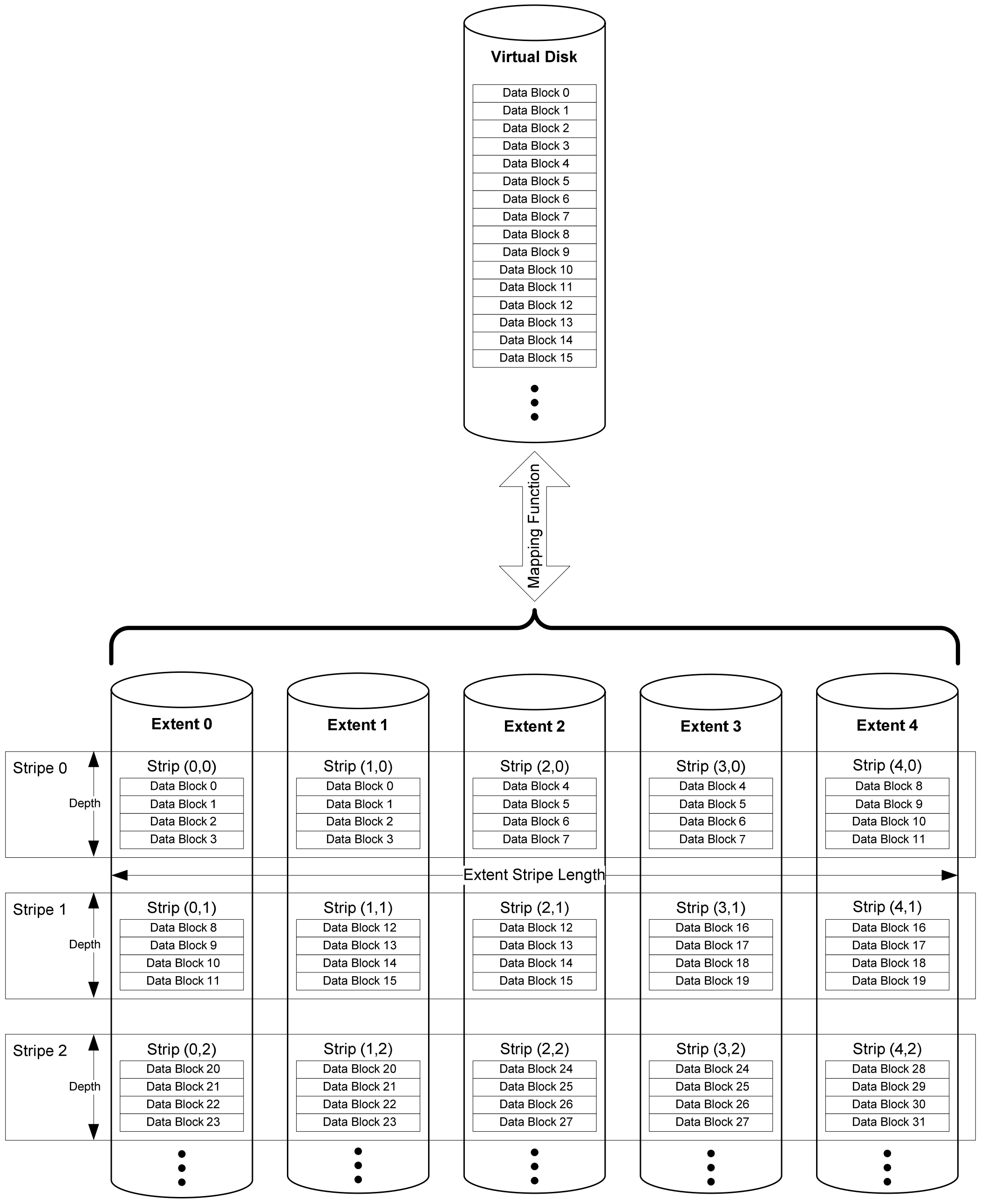

Figure 18: Integrated Adjacent Stripe Mirroring (PRL= 11, RLQ-00)

4.2.18 Integrated Offset Stripe Mirroring (PRL=11, RLQ=01)

Figure 19 gives an example of Integrated Offset Stripe Mirroring. The constants and indices given in Table 8 also apply for the following formulas that describe the data layout of an Integrated Offset Stripe Mirroring VD. Each virtual_block (*x*) MUST be stored in two locations. These two locations are referred to as extent_block (*k, i, j*) and extent_block' (*k, q, r*). The allocation of data in an Integrated Offset Stripe Mirroring VD MUST adhere to the following formulas:

Eq. 57

virtual_block (x) = extent_block (MOD(x/L), MOD(FLOOR(x/L), N), FLOOR(FLOOR(x/L)/N))*2)

Eq. 58

virtual_block (x) = extent_block’ (MOD(x/L), MOD(FLOOR(x/L)+1, N), (FLOOR(FLOOR(x/L)/N)*2)+1)

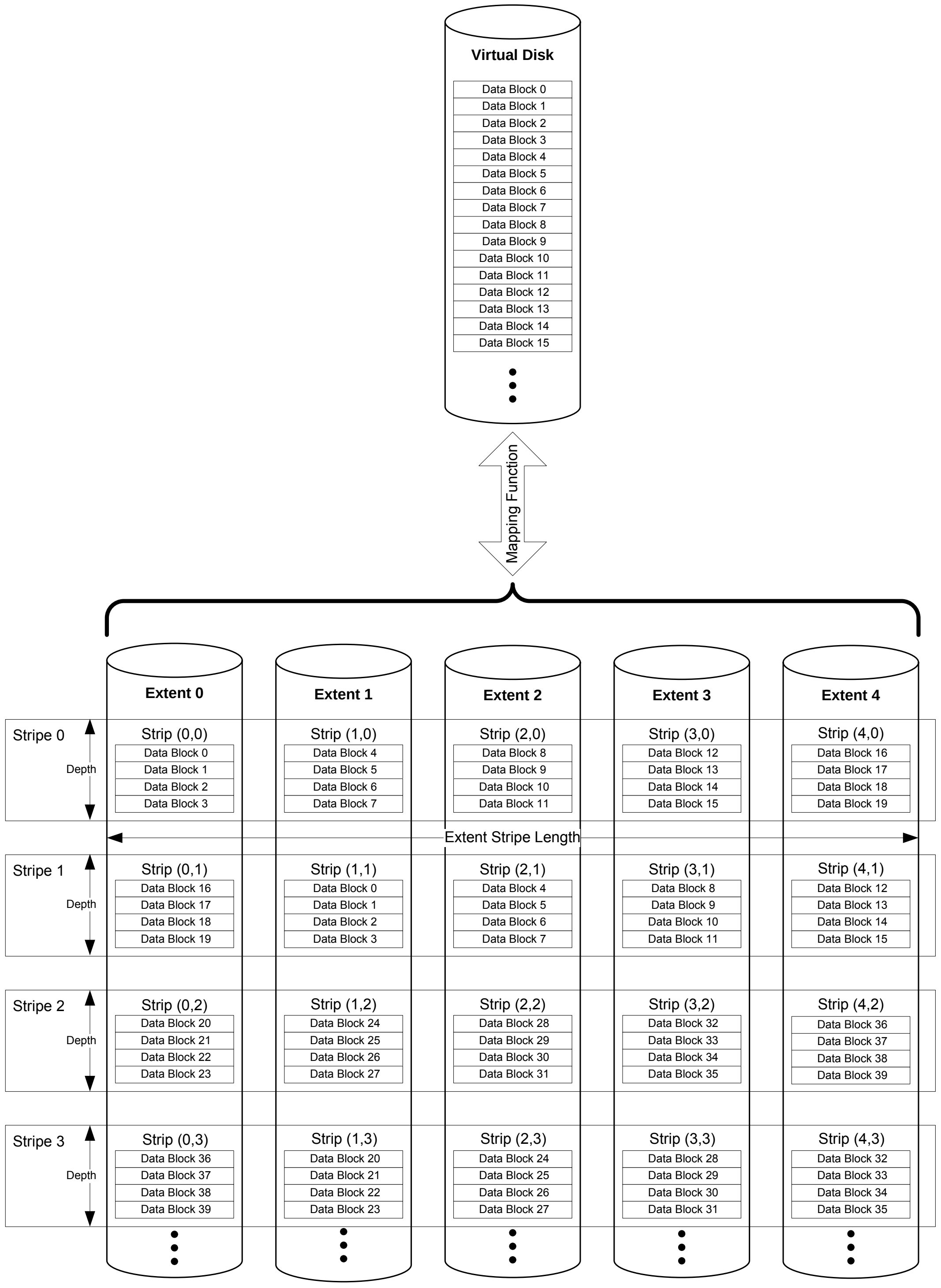

Figure 19: Integrated Offset Stripe Mirroring (PRL= 11, RLQ=01)

4.2.19 RAID 6 Rotating Parity 0 with Data Restart (PRL=06, RLQ=01)

The following table defines the variables and indices that are required to describe the RAID-6 data layout and parity computation.

Table 9: Indices and Constants for RAID-6 Rotating Parity 0 and N

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of an extent of a VD	0	N-1
j	Index of a stripe in a VD	0	FLOOR(FLOOR((M-1)/L)/(N-2))
k	Offset of a block from the beginning of a strip.	0	L-1
L	Size of a strip in blocks.	N/A (a fixed value)	N/A (a fixed value)
M	Number of data blocks in a VD. M MUST be evenly divisible by (N-2)*L.	N/A (a fixed value)	N/A (a fixed value)
N	Number of extents in a VD.	N/A (a fixed value)	N/A (maximum of 255)
b	Index of a byte in an extent block.	0	Block Size - 1 (a fixed value of 511).
F	Maximum number of simultaneous disk failures tolerated.	2	2
p	Index of the extent on which parity P for a given stripe resides.	0	N-1
q	Index of the extent on which parity Q for a given byte stripe resides.	0	N-1
Z	The polynomial used to generate the Galois field elements (required for parity computation).	N/A (a constant value)	N/A (a constant value)
K_i	The ith Galois field element enumerated in Table 10.	See Table 10	See Table 10

The number of simultaneous disk failures supported (F) determines the number of parity elements that must be computed. For example, a RAID 6 sub-system that tolerates 2 simultaneous disk failures MUST compute two parity elements: P and Q.

Figure 20 gives an example of RAID-6 called Rotating Parity 0 with Data Restart. The indices and constants provided in Table 9 are valid for this type of RAID.

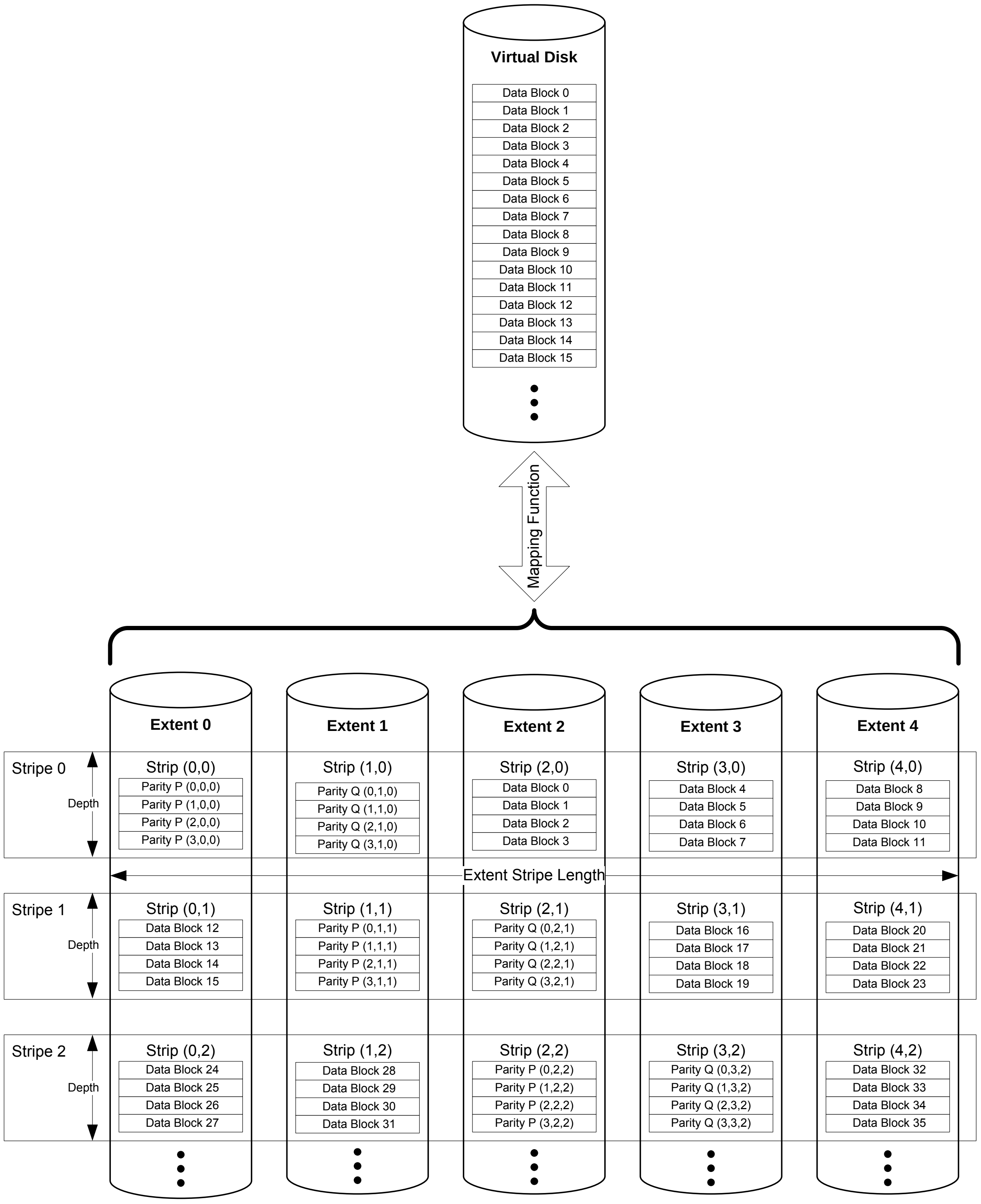

Figure 20: RAID 6 Rotating Parity 0 with Data Restart (PRL=06, RLQ=01)

The stripe *j* on which a virtual block *x* resides MUST adhere to the following formula:

Eq. 59

$$j = (\text{FLOOR}\ ((\text{FLOOR}(x/L))/(N\text{-}2))$$

The extent *p* on which the parity P for a given stripe *j* resides MUST adhere to the following formula:

Eq. 60

$$p = \text{MOD}\ (\text{FLOOR}\ ((\text{FLOOR}\ (x/\text{L}))/(N\text{-}2)),\ \text{N})$$

The extent *q* on which the parity Q for a given stripe *j* resides must adhere to the following formula:

Eq. 61

$$q = \text{MOD}\ (1+p,\ N)$$

The extent *i* on which a given virtual block *x* resides MUST adhere to the following formula:

Eq. 62

```
IF MOD (FLOOR ($x/L$), $N$-2) < $p$
THEN {
        $i$ = MOD (FLOOR ($x/L$), $N$-2)
        IF $p$+2 > $N$
        THEN   $i = i$ + MOD ($p$+2, $N$)
}
ELSE
        $i$ =  MOD (FLOOR ($x/L$), $N$-2) + 2
```

The allocation of data blocks in a Rotating Parity 0 with Data Restart VD MUST adhere to the following formula:

Eq. 63

$$\text{virtual\_block}\ (x) = \text{extent\_block}\ (\text{MOD}(x,\ L),\ i,\ j)$$

The values of the parity P and Q MUST be computed according to the following formula:

Eq. 64

$$\text{Parity P}\ (k,\ p,\ j) = \bigoplus_{i=0}^{N-1, i\neq p, i\neq q} \text{extent\_block}\ (k, i,\ j).$$

The operator $\oplus$ refers to bit-wise XOR of the operands.

Eq. 65

$$\text{Parity\_Byte Q } (b,k,q,j) = \sum_{i=0}^{i=N-1, i\neq p, i\neq q} K_i \otimes \text{extent\_block\_byte } (b,k,q,j)$$

The operators $\sum$ and $\otimes$ refers to Galois field addition and Galois field multiplication respectively. The value for constant K_i MUST adhere to the following formula:

Eq. 66

$$K_i = \text{GFILOG}(i)$$

Eq. 67

$$\text{Parity Q } (k,q,j) = \underset{b=0}{\overset{b<512}{\forall}} \text{Parity\_Byte Q } (b,k,q,j)$$

Notes:

1. The notation $\forall$ implies that Parity Q (k,q,j) is a sequential enumeration of the Parity_Byte Q (b,k,q,j), where b ranges from 0 to the block size -1 (i.e., 511).

2. Function extent_block_byte (b,k,i,j) returns the b^{th} byte of an extent_block (k,i,j).

4.2.19.1 Parity Re-computation on Block Update

If a data block is updated at byte index b of an extent u, the equations for P and Q parity re-computation MUST be as follows:

Eq. 68

$$\text{Parity } P_{new} = \text{extent\_block\_byte}(b,k,p,j) \oplus \text{old\_extent\_block\_byte}(b,k,u,j) \oplus \text{new\_extent\_block\_byte}(b,k,u,j)$$

Eq. 69

$$\text{Parity } Q_{new} = \text{extent\_block\_byte}(b,k,q,j) \oplus K_u \otimes \text{old\_extent\_block\_byte}(b,k,u,j) \oplus K_u \otimes \text{new\_extent\_block\_byte}(b,k,u,j)$$

Where K_u = GFILOG(u) and $\otimes$ refers to Galois multiplication operation. P_{new} and Q_{new} MUST be re-computed for every byte index b updated in the extent u.

4.2.19.2 Galois Field Operations

This section describes how the Galois field operations MUST be defined to calculate Parity Q for the RAID 6 algorithms described in Sections 4.2.19, 4.2.20, and 4.2.21.

4.2.19.2.1 GFILOG () Function (Z = 0x11D)

The following table lists the Galois field elements generated from the polynomial 0x11D.

Table 10: Galois Field Elements for Polynomial 0x11D

GFILOG(0xRS)		S															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R	0	1	2	4	8	10	20	40	80	1D	3A	74	E8	CD	87	13	26
	1	4C	98	2D	5A	B4	75	EA	C9	8F	03	06	0C	18	30	60	C0
	2	9D	27	4E	9C	25	4A	94	35	6A	D4	B5	77	EE	C1	9f	23
	3	46	8C	05	0A	14	28	50	A0	5D	BA	69	D2	B9	6F	DE	A1
	4	5F	BE	61	C2	99	2F	5E	BC	65	CA	89	0F	1E	3C	78	F0
	5	FD	E7	D3	BB	6B	D6	B1	7F	FE	E1	DF	A3	5B	B6	71	E2
	6	D9	AF	43	86	11	22	44	88	0D	1A	34	68	D0	BD	67	CE
	7	81	1F	3E	7C	F8	ED	C7	93	3B	76	EC	c5	97	33	66	CC
	8	85	17	2E	5C	B8	6D	DA	A9	4F	9E	21	42	84	15	2A	54
	9	A8	4D	9A	29	52	A4	55	AA	49	92	39	72	E4	D5	B7	73
	A	E6	D1	BF	63	C6	91	3F	7E	FC	E5	D7	B3	7B	F6	F1	FF
	B	E3	DB	AB	4B	96	31	62	C4	95	37	6E	DC	A5	57	AE	41
	C	82	19	32	64	C8	8D	07	0E	1C	38	70	E0	DD	A7	53	A6
	D	51	A2	59	B2	79	F2	F9	EF	C3	9B	2B	56	AC	45	8A	09
	E	12	24	48	90	3D	7A	F4	F5	F7	F3	FB	EB	CB	8B	0B	16
	F	2C	58	B0	7D	FA	E9	CF	83	1B	36	6C	D8	AD	47	8E	XX

Note: The values within Table 10 are hexadecimal values.

GFILOG(0xFF) is undefined and represented as XX. Here are two examples of how to apply Table 10:

GFILOG(0x8C) = 0x84,

GFILOG(0x21) = 0x27.

4.2.19.2.2 GFLOG() Function (Z = 0x11D)

The following table lists the logarithmic value of the Galois field elements.

Table 11: Log of Galois Field Elements for Polynomial 0x11D

GFLOG(0xRS)		S															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R	0	XX	00	01	19	02	32	1A	C6	03	DF	33	EE	1B	68	C7	4B
	1	04	64	E0	0E	34	8D	EF	81	1C	C1	69	F8	C8	08	4C	71
	2	05	8A	65	2F	E1	24	0F	21	35	93	8E	DA	F0	12	82	45
	3	1D	B5	C2	7D	6A	27	F9	B9	C9	9A	09	78	4D	E4	72	A6
	4	06	BF	8B	62	66	DD	30	FD	E2	98	25	B3	10	91	22	88
	5	36	D0	94	CE	8F	96	DB	BD	F1	D2	13	5C	83	38	46	40
	6	1E	42	B6	A3	C3	48	7E	6E	6B	3A	28	54	FA	85	BA	3D
	7	CA	5E	9B	9F	0A	15	79	2B	4E	D4	E5	AC	73	F3	A7	57
	8	07	70	C0	F7	8C	80	63	0D	67	4A	DE	ED	31	C5	FE	18
	9	E3	A5	99	77	26	B8	B4	7C	11	44	92	D9	23	20	89	2E
	A	37	3F	D1	5B	95	BC	CF	CD	90	87	97	B2	DC	FC	BE	61
	B	F2	56	D3	AB	14	2A	5D	9E	84	3C	39	53	47	6D	41	A2
	C	1F	2D	43	D8	B7	7B	A4	76	C4	17	49	EC	7F	0C	6F	F6
	D	6C	A1	3B	52	29	9D	55	AA	FB	60	86	B1	BB	CC	3E	5A
	E	CB	59	5F	B0	9C	A9	A0	51	0B	F5	16	EB	7A	75	2C	D7
	F	4F	AE	D5	E9	E6	E7	AD	E8	74	D6	F4	EA	A8	50	58	AF

Note: The values within Table 11 are hexadecimal values.

GFLOG(0x00) is undefined and represented as XX. Here are three examples of how to apply Table 11.

GFLOG(0x01) = 0x00,

GFLOG(0xF9) = 0xD6,

GFLOG(0x94) = 0x26.

4.2.19.2.3 Galois Field Addition (∑)

The Galois field addition operation is defined as the bitwise XOR ($\oplus$) of the operands. For example,

Eq. 70

$$\sum (0x22, 0x33, 0x44) = (0x22 \oplus 0x33 \oplus 0x44) = 0x55.$$

4.2.19.2.4 Galois Field Multiplication ($\otimes$)

The Galois field multiplication ($a \otimes b$) is defined as:

Eq. 71

$$\text{GFILOG (MOD (GFLOG } (a) + \text{GFLOG } (b), \text{0xFF)).}$$

Note: Integer addition applies within Eq. 71.

Refer Section 4.2.19.2.2 and Section 4.2.19.2.1 for the definitions of functions GFLOG() and GFILOG() respectively. For example,

a) 0x33 $\otimes$ 0x44 = GFILOG (MOD (0x7D + 0x66, 0xFF)) = GFILOG (0xE3) = 0x90

b) 0x08 $\otimes$ 0x54 = GFILOG (MOD (0x03 + 0x8F, 0xFF)) = GFILOG (0x92) = 0x9A

4.2.20 RAID 6 Rotating Parity N with Data Restart (PRL=06, RLQ=02)

Figure 21 gives an example of an implementation of RAID-6 called Rotating Parity N with Data Restart. The indices and constants provided in Table 9 are valid for this type of RAID.

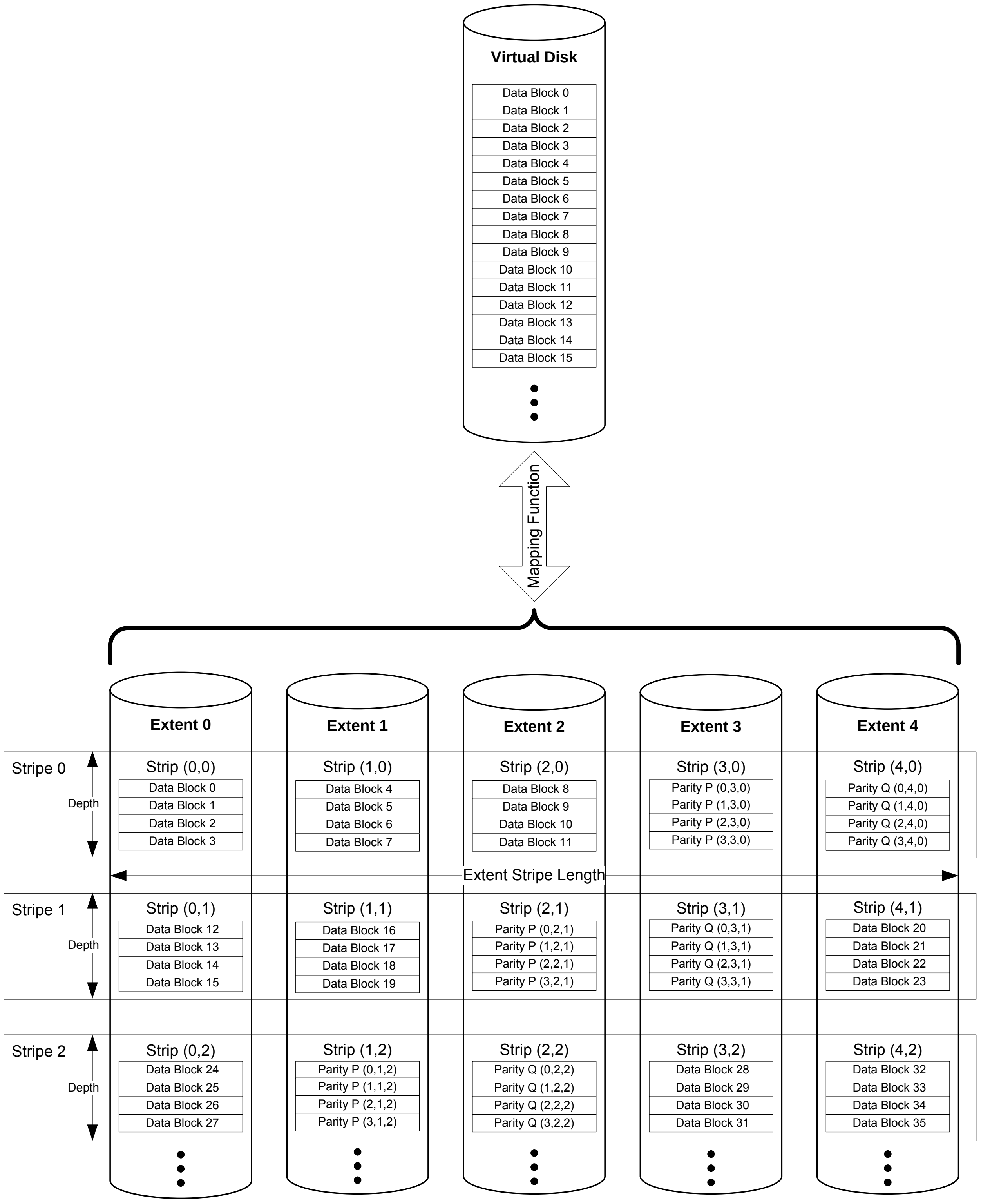

Figure 21: Rotating Parity N with Data Restart (PRL=06, RLQ=02)

The stripe *j* on which a virtual block *x* resides MUST adhere to the following formula:

Eq. 72

$$j = (\text{FLOOR}\ ((\text{FLOOR}(x/L)) \ /\ (N\text{-}2))$$

The extent *p* on which the parity P for a given stripe *j* resides must adhere to the following formula:

Eq. 73

$$p = (N\text{-}1) - (\text{MOD}\ (\text{FLOOR}\ ((\text{FLOOR}(x/L)) \ /\ (N\text{-}2) + 1),\ N))$$

The extent *q* on which the parity Q for a given stripe *j* resides must adhere to the following formula:

$$q = \text{MOD}\ (1+p,\ N)$$

The extent i on which a given block x resides MUST adhere to the following formula:

Eq. 74

```
IF MOD (FLOOR ($x/L$), $N$-2) < $p$
 THEN {
        i = MOD (FLOOR (x/L), N-2)
        IF $p$+2 > $N$
        THEN $i$ = $i$ + MOD ($p$+2, $N$)
}
ELSE
        $i$ =  MOD (FLOOR ($x/L$), $N$-2) + 2
```

The allocation of data blocks in a Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 75

$$\text{virtual\_block}\ (x) = \text{extent\_block}\ (\text{MOD}(x,\ L),\ i,\ j)$$

The values of the parity P and Q MUST be computed according to the following formula:

Eq. 76

$$\text{Parity P}\ (k,p,j) = \bigoplus_{i=0}^{N-1, i \neq p, i \neq q} \text{extent\_block}\ (k,i,j)$$

and the operator $\oplus$ refers to bit-wise XOR of the operands.

Eq. 77

$$\text{Parity\_Byte Q}\ (b,k,q,j) = \sum_{i=0}^{i=N-1, i \neq p, i \neq q} K_i \otimes \text{extent\_block\_byte}\ (b,k,i,j)$$

The operators $\sum$ and $\otimes$ refer to Galois field addition and Galois field multiplication respectively. The value for constant K_i is arrived as:

Eq. 78

$$K_i = \text{GFILOG}(i).$$

Eq. 79

$$\text{Parity Q } (k,q,j) = \overset{b<512}{\underset{b=0}{\forall}} \text{Parity\_Byte Q } (b,k,q,j)$$

Notes:

1. The notation $\forall$ implies that Parity Q (*k,q,j*) is a sequential enumeration of the Parity_Byte Q (*b,k,q,j*), where *b* ranges from 0 to the block size -1 (i.e., 511).

2. Function extent_block_byte (*b,k,i,j*) returns the *b*th byte of an extent_block (*k,i,j*).

4.2.20.1 Parity Re-computation on Block Update

If a data block is updated at byte index b of an extent u, the equations for P and Q parity re-computation MUST be as follows:

Eq. 80

$$\text{Parity } P_{new} = \text{extent\_block\_byte}(b,k,p,j) \oplus \text{old\_extent\_block\_byte}(b,k,u,j) \oplus \text{new\_extent\_block\_byte}(b,k,u,j)$$

Eq. 81

$$\text{Parity } Q_{new} = \text{extent\_block\_byte}(b,k,q,j) \oplus K_u \otimes \text{old\_extent\_block\_byte}(b,k,u,j) \oplus K_u \otimes \text{new\_extent\_block\_byte}(b,k,u,j)$$

Where K_u = GFILOG(*u*) and $\otimes$ refers to Galois multiplication operation. P_{new} and Q_{new} MUST be re-computed for every byte index *b* updated in the extent *u*.

4.2.20.2 Galois Field Operations

The Galois field operations that MUST be used to calculate Parity Q for the RAID 6 algorithm described in Section 4.2.20 are the same as the Galois field operations described in 4.2.19.2.

4.2.21 RAID 6 Rotating Parity N with Data Continuation (PRL=06, RLQ=03)

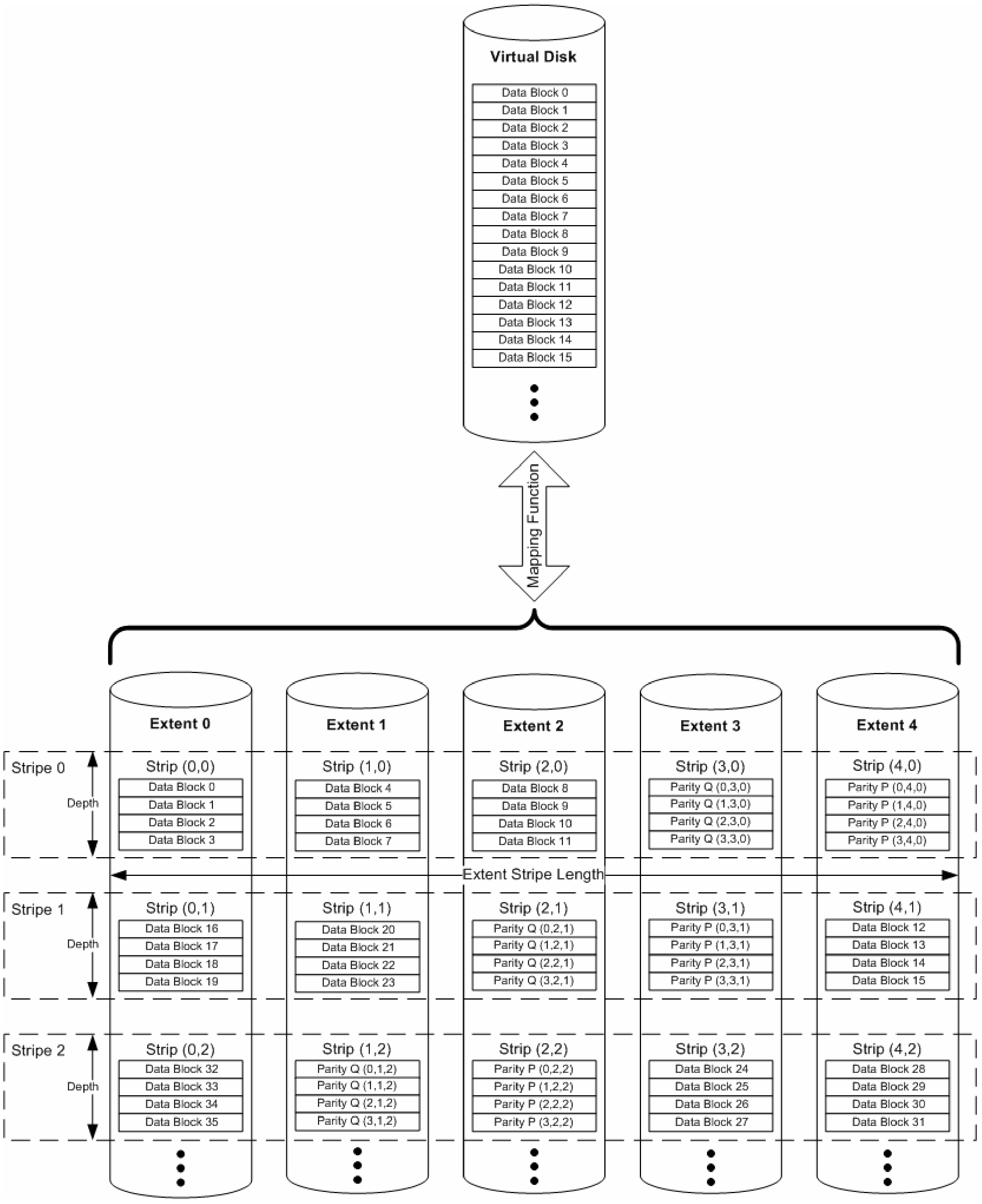

Figure 22: Rotating Parity N with Data Continuation (PRL=06, RLQ=03)

Figure 22 gives an example of an implementation of RAID-6 called Rotating Parity N with Data Continuation. The indices and constants provided in Table 9 are valid for this type of RAID.

The stripe j on which a virtual block x resides MUST adhere to the following formula:

Eq. 82

$$j = (\text{FLOOR}((\text{FLOOR}(x/L)) / (N\text{-}2))$$

The extent p on which the parity P for a given stripe j resides must adhere to the following formula:

Eq. 83

$$p = (N\text{-}1) - \text{MOD}(j,N).$$

The extent q on which the parity Q for a given stripe j resides must adhere to the following formula:

$$q = \text{MOD}((p\text{-}1),N).$$

The extent i on which a given virtual block x resides MUST adhere to the following formula:

Eq. 84

$$i = \text{MOD}(\text{MOD}(\text{FLOOR}(x/L),N\text{-}2)+p+1,\text{N}).$$

The allocation of data blocks in a RAID-6 Rotating Parity N with Data Restart VD MUST adhere to the following formula:

Eq. 85

$$\text{virtual\_block}(x) = \text{extent\_block}(\text{MOD}(x/L), i, j).$$

The values of the parity P and Q MUST be computed according to the following formula:

Eq. 86

$$\text{Parity P }(k,p,j) = \bigoplus_{i=0}^{N-1, i\neq p, i\neq q} \text{extent\_block }(k,i,j)$$

and the operator $\oplus$ refers to bit-wise XOR of the operands.

Eq. 87

$$\text{Parity\_Byte Q }(b,k,q,j) = \sum_{i=0}^{i=N-1, i\neq p, i\neq q} K_i \otimes \text{extent\_block\_byte }(b,k,i,j)$$

The operators $\sum$ and $\otimes$ refer to Galois field addition and Galois field multiplication respectively. The value for constant K_i is arrived as:

Eq. 88

$$K_i = \text{GFILOG}(i).$$

Eq. 89

$$\text{Parity Q }(k,q,j) = \underset{b=0}{\overset{b<512}{\forall}} \text{ Parity\_Byte Q }(b,k,q,j)$$

Notes:

1. The notation $\forall$ implies that Parity Q (k,q,j) is a sequential enumeration of the Parity_Byte Q (b,k,q,j), where b ranges from 0 to the block size -1 (i.e., 511).

2. Function extent_block_byte (b,k,i,j) returns the b^{th} byte of an extent_block (k,i,j).

4.2.21.1 Parity Re-computation on Block Update

If a data block is updated at byte index b of an extent u, the equations for P and Q parity re-computation MUST be as follows:

Eq. 90

$$\text{Parity } P_{new} = \text{extent\_block\_byte}(b,k,p,j) \oplus \text{old\_extent\_block\_byte}(b,k,u,j) \oplus \text{new\_extent\_block\_byte}(b,k,u,j)$$

Eq. 91

$$\text{Parity } Q_{new} = \text{extent\_block\_byte}(b,k,q,j) \oplus K_u \otimes \text{old\_extent\_block\_byte}(b,k,u,j) \oplus K_u \otimes \text{new\_extent\_block\_byte}(b,k,u,j)$$

Where K_u = GFILOG(u) and $\otimes$ refers to Galois multiplication operation. P_{new} and Q_{new} MUST be re-computed for every byte index b updated in the extent u.

4.2.21.2 Galois Field Operations

The Galois field operations that MUST be used to calculate Parity Q for the RAID 6 algorithm described in Section 4.2.21 are the same as the Galois field operations described in 4.2.19.2.

4.3 Secondary RAID Level

Table 12 lists values used in the Secondary_RAID_Level field of the Virtual Disk Configuration Record (Section 5.9.1) and their definitions. The table defines secondary RAID levels such as Striped, Volume Concatenation, Spanned, and Mirrored for hybrid or multilevel virtual disks. The Secondary_RAID_Level field in the Virtual Disk Configuration Record MUST use the values defined in Table 12.

Table 12: Secondary RAID Levels

Name	SRL Byte	Description
Striped	00	Data is striped across Basic VDs. First strip stored on first BVD and next on next BVD. **NOTE:** BVD sequence is determined by the Secondary_Element_Seq field in the Virtual Disk Configuration Record (Section 5.9.1).
Mirrored	01	Data is mirrored across Basic VDs.
Concatenated	02	Basic VDs combined head to tail.
Spanned	03	A combination of stripping and concatenations involving Basic VDs of different sizes. **NOTE:** BVD sequence is determined by the Secondary_Element_Seq field in the Virtual Disk Configuration Record (Section 5.9.1).

4.3.1 Striped Secondary RAID Level (SRL=00)

In a VD with a Striped Secondary RAID Level, the data MUST be striped across basic virtual disks. The BVDs MUST have an equal number of user addressable data strips. The BVDs are not required to have the same Primary RAID Level (Section 4.1), RAID Level Qualifier (Section 4.2), or capacity. Table 13 gives the indices and constants used in the description of a Striped Secondary RAID Level (Striped SRL) VD.

Table 13: Indices and Constants for Striped Secondary RAID Level

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of a BVD of in a striped VD	0	N-1
j	Index of a stripe in a VD	0	J-1
q	Offset of a data block from the beginning of a stripe	0	S-1
L_i	Size of a strip in blocks on BVD i	Equal to the stripe depth of BVD i's Primary RAID Level	Equal to the stripe depth of a BVD i's Primary RAID Level
M	Number of blocks in a VD	N/A (a fixed value)	N/A (a fixed value)
N	Number of BVDs in a striped VD	N/A (a fixed value)	N/A (a fixed value)
J	Number of stripes	M/S	M/S
S	Size of a stripe in a striped VD	$S = \sum_{i=0}^{N-1} L_i$	$S = \sum_{i=0}^{N-1} L_i$
x	Offset of a data block from the beginning of a VD	0	M-1

The Striped Secondary RAID level introduces the concept of a *stripe_block*. Since BVDs in a Striped SRL VD can have different strip sizes, it is more economical to represent a block's offset from the beginning of a stripe rather than from the beginning of a strip on a specific BVD. A stripe_block uses the indices *q* and *j* to identify its position (i.e., stripe_block(*q*,*j*))

The size of a stripe on a Striped SRL VD MUST adhere to the following formula:

Eq. 92

$$S = \sum_{i=0}^{N-1} L_i$$

The number of stripes *J* is equal to *M*/*S*. The user addressable data MUST be distributed according to the following formula:

Eq. 93

virtual_block(*x*) = stripe_block(MOD(*x*, *S*), FLOOR(*x, S*))

Figure 23 gives an example of a VD with a Striped Secondary RAID Level. In this example L_0 = 4, L_1 = 2, L_2 = 3, L_3 = 4, and L_4 = 3. Thus, *S* = 16.

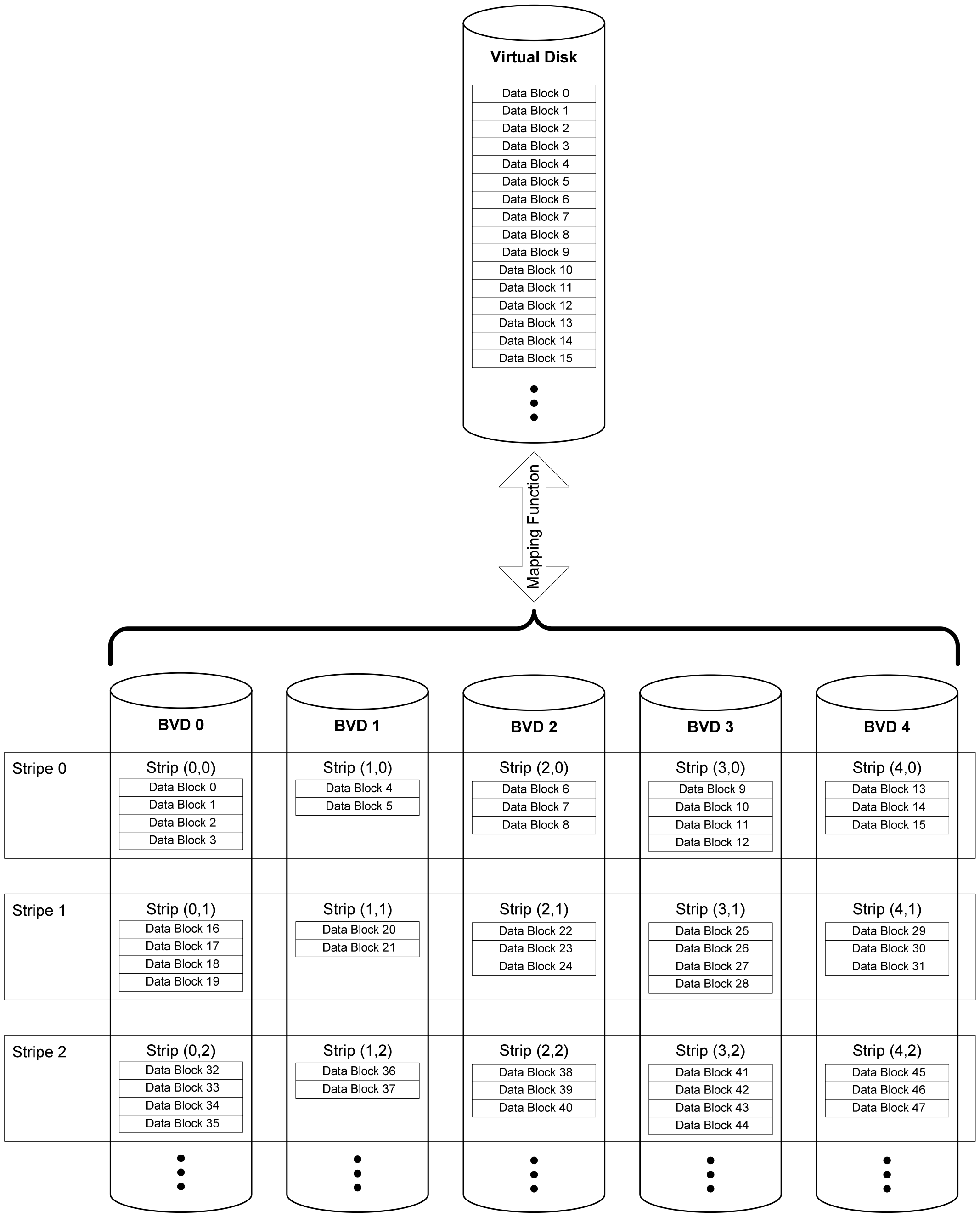

Figure 23: Striped Secondary RAID Level (SRL=00)

4.3.2 Mirrored Secondary RAID Level (SRL=01)

In a VD with a Mirrored Secondary RAID Level, the data MUST be mirrored across basic virtual disks. Figure 24 gives an example of a VD with a Mirrored Secondary RAID Level. Each data block of the virtual disk (virtual_block(*x*)) MUST be stored at the same user addressable data offset in each BVD.

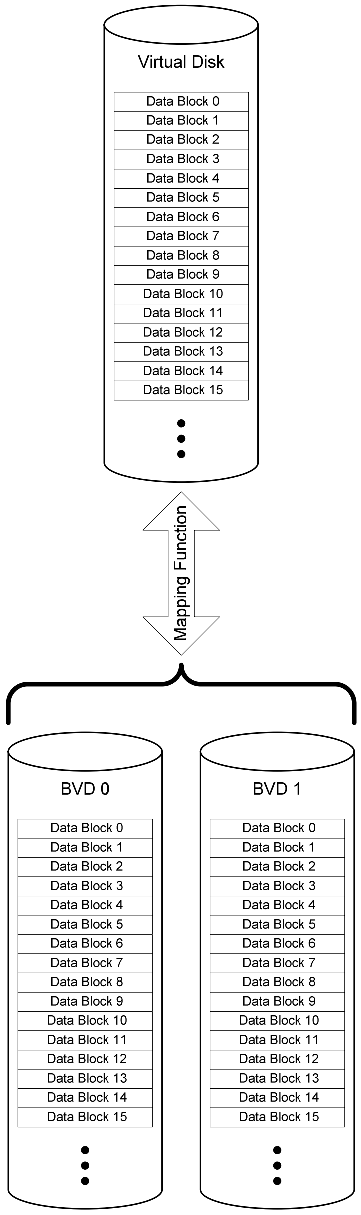

Figure 24: Mirrored Secondary RAID Level (SRL=01)

4.3.3 Concatenated Secondary RAID Level (SLR=02)

In a VD with a Concatenated Secondary RAID Level, the user addressable data space is created by combining the user addressable data spaces of the BVDs making up the VD. The BVDs act as extents for the VD and are called *BVD extents*. The BVDs are combined in a head to tail fashion. The BVDs are not required to have the same addressable data space.

Figure 25 gives an example of a Concatenated VD with three BVD extents. Table 14 gives the indices and constants used in the formulas below

To represent a specific block in a specific BVD the following notation is used:

bvd_extent_block (*y*, *i*).

The allocation of data MUST adhere to the following equations.

Eq. 94

```
IF x = 0
THEN virtual_block ($0$) = bvd_extent_block ($0, 0$)
ELSE IF x = $M-1$
THEN virtual_block ($x$) = bvd_extent_block ($C_{N-1}$-1, $N$-1)
ELSE IF virtual_block ($x-1$) = bvd_extent_block ($C_i$-1, $i$)
THEN virtual_block ($x$) = bvd_extent_block (0, $i$+1)
ELSE IF virtual_block ($x-1$) = bvd_extent_block ($y-1$, $i$)
THEN virtual_block ($x$) = bvd_extent_block ($y$, $i$)
```

Table 14: Indices and Constants for Concatenated Secondary RAID Level VDs

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of a BVD extent of a VD	0	N-1
M	Number of data blocks in a VD	N/A (a fixed value)	N/A (a fixed value)
N	Number of BVDs acting as extents in a VD with a Concatenated Secondary RAID Level	N/A (a fixed value)	N/A (a fixed value)
C_i	Number of data blocks in BVD i	N/A (a fixed value)	N/A (a fixed value)
x	Offset of a data block from the beginning of the VD	0	M-1
y	Offset of a data block from the beginning of a BVD extent	0	M-1

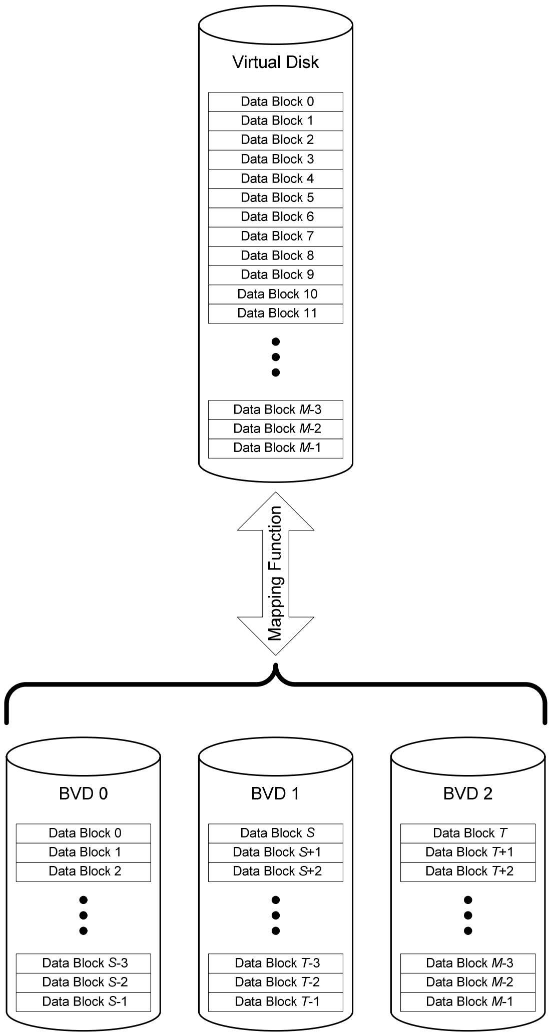

Figure 25: Concatenated Secondary RAID Level (SRL=02)

4.3.4 Spanned Secondary RAID Level (SRL=03)

In VD with a Spanned Secondary RAID Level, the user addressable data space is created by a combination of stripping and concatenation of the user addressable data spaces of the BVDs making up the VD. The BVDs can be in any order. They can have different capacities, different RAID levels and different strip sizes. Unlike the Striped Secondary RAID Level, the BVDs do not have to have the same number of user addressable data strips. Table 15 gives the indices and constants used in the description of a Spanned Secondary RAID Level (Spanned SRL) VD.

The size of stripe *j* MUST be determined using the following formula:

Eq. 95

$$S_j = \sum_{i=0}^{N-1} \{\text{IF } J_i \geq j \text{ THEN } L_i \text{ ELSE } 0\}.$$

The total number of user addressable data blocks included in stripes 0 through *j* is:

Eq. 96

$$T_j = \sum_{v=0}^{j} S_v$$

The Spanned SRL VD is also described using the stripe_block(*q, j*) concept (Section 4.3.1). The user addressable data in a Spanned SRL VD MUST be distributed according to the following formula.

Eq. 97

$$\text{virtual\_block}(x) = \text{stripe\_block}(x - T_u, u + 1),$$

where T_u is the largest value of T_j less than or equal to x.

Table 15: Indices and Constants for Spanned Secondary RAID Level

Variable or Index	Description	Minimum Value	Maximum Value
i	Index of a BVD of in a spanned VD	0	N-1
j	Index of a stripe in a VD	0	J-1
q	Offset of a data block from the beginning of a stripe	0	S-1
L_i	Size of a strip in blocks on BVD *i*	Equal to the stripe depth of BVD *i*'s Primary RAID Level	Equal to the stripe depth of a BVD *i*'s Primary RAID Level
M	Number of blocks in a VD	N/A (a fixed value)	N/A (a fixed value)
N	Number of BVDs in a striped VD	N/A (a fixed value)	N/A (a fixed value)
J_i	Number of stripes in on BVD *i*	N/A (a fixed value determined by BVD *i*'s Primary RAID Level parameters)	N/A (a fixed value determined by BVD *i*'s Primary RAID Level parameters)
S_j	Size of a stripe *j* on a Spanned VD	$S_j = \sum_{i=0}^{N-1}$ {IF $J_i \geq j$ THEN L_i ELSE 0 }	$S_j = \sum_{i=0}^{N-1}$ {IF $J_i \geq j$ THEN L_i ELSE 0 }
T_j	Number of blocks in stripes 0 through *j*	$T_j = \sum_{v=0}^{j} S_v$	$T_j = \sum_{v=0}^{j} S_v$
x	Offset of a data block from the beginning of a VD	0	M-1

Figure 26 gives an example of a Spanned SRL VD with five BVDs.

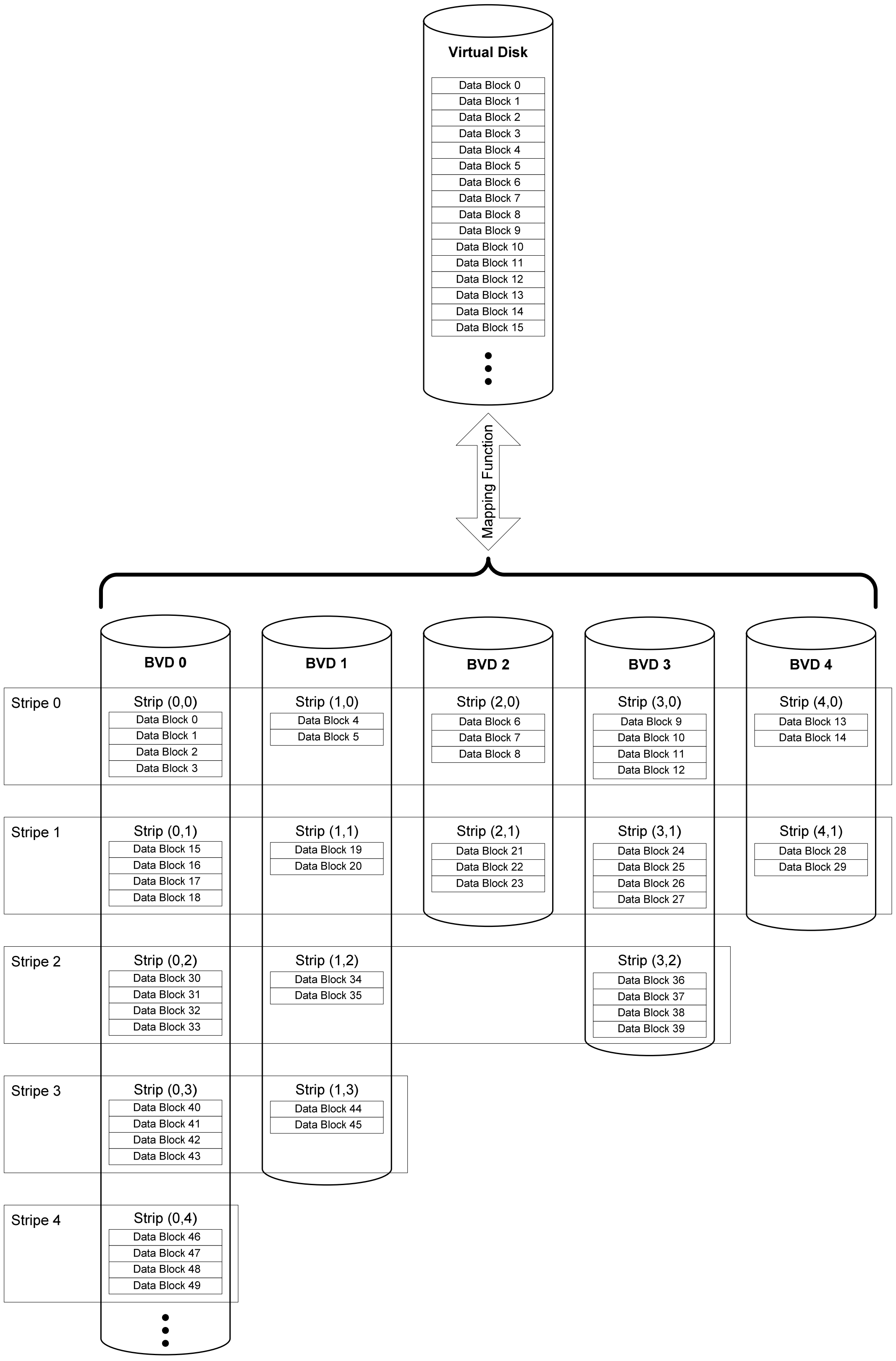

Figure 26: Spanned Secondary RAID Level (SRL=03)

5 DDF Structure

5.1 DDF Structure Overview

A DDF structure MUST reside on every physical disk that participates in a RAID configuration in a RAID storage subsystem. A minimum of 32MB MUST be reserved on each physical disk for a DDF structure. The last block of the reserved space MUST be the last addressable block of the physical disk.

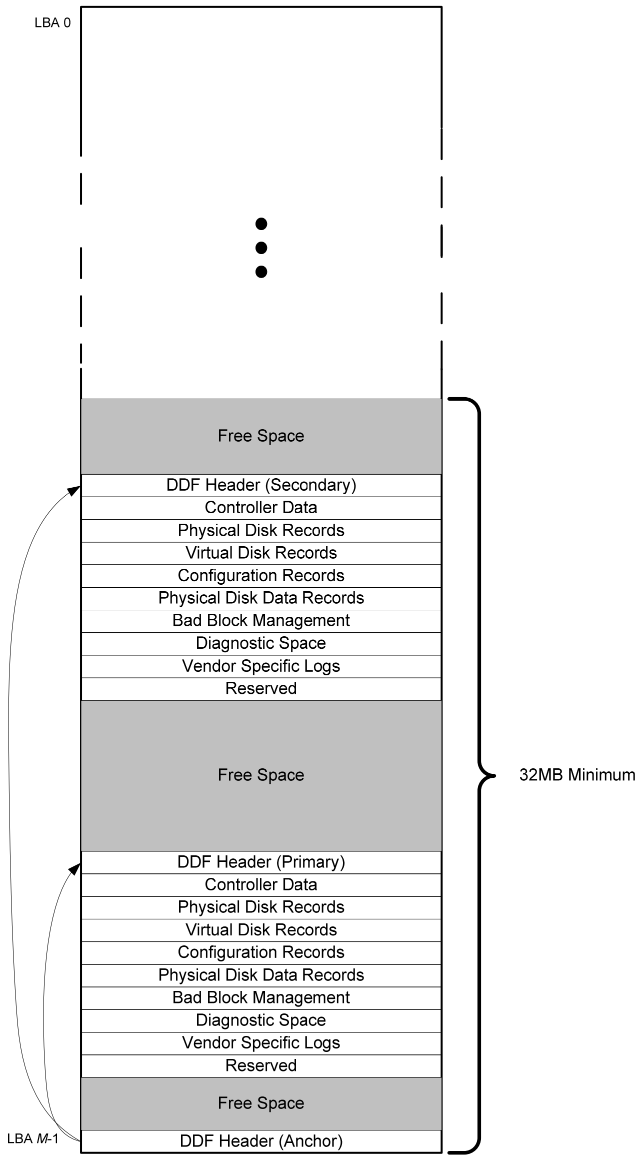

Figure 27: DDF Structure

Figure 27 illustrates the conceptual format of the DDF Structure on a physical disk with *M* addressable blocks. The DDF structure contains nine section types that are listed in Table 16.

Table 16: DDF Sections

Contents	Context
DDF Header	Global
Controller Data	Global
Physical Disk Records	Global
Virtual Disk Records	Global
Configuration Records	Local
Physical Disk Data	Local
Bad Block Management Log	Local
Diagnostic Space	Local
Vendor Specific Logs	Vendor specific

The DDF Header section has three types. The Anchor DDF Header MUST be stored at the end of the physical disk. The last block of the Anchor DDF Header MUST be the last addressable block on a physical disk. The Anchor DDF Header contains a pointer to the Primary DDF Header. If DDF structure redundancy is implemented, the Anchor DDF Header also contains a pointer to the Secondary DDF Header.

Figure 27 shows a DDF structure with redundant entries. Redundancy is OPTIONAL but if it is implemented the values of all fields in all section types except DDF Header sections MUST be equal in both the primary and secondary locations. Furthermore, the offset of the beginning of secondary sections from the start of the Secondary DDF Header MUST be equal to the offset of the beginning of the primary sections from the start of the Primary DDF Header. There MUST be only one DDF structure stored on a disk. The disk MUST not be partitioned using a partition table or another mechanism. The partitioning of capacity is handled by the DDF structure using the concept of Virtual Disks. The term DDF structure includes the Anchor DDF Header, the Primary DDF Header, the Secondary DDF Header, and all DDF Sections described in Table 16.

Signatures and CRCs are used to delineate the sections of the DDF structure. Sequence numbers are used to manage transitions during configuration and the change/update process. Timestamps are used to provide chronological hints. Configuration groups, Controllers, Physical Disks and Virtual Disks are uniquely identified by using "Globally Unique IDs" (GUIDs).

The size of sections like Physical Disk Records, Virtual Disk Records and Configuration Records can vary based on the maximum number of physical disks (PDs), virtual disks and the maximum number of physical disks configurable within a virtual disk (PD per VD) a given implementation supports. This makes the DDF structure size variable.

A minimum of 32MB (65536 blocks) MUST be reserved on each physical disk to accommodate redundant DDF sections, optional spare blocks, and vendor specific logs or work space. It is NOT RECOMMENDED that redundant copies of DDF be placed adjacent to each other.

5.2 Byte Ordering

Each section of the DDF MUST be stored in big-endian format (i.e., the more significant bytes of the section are stored in lower addresses in relation to bytes of lesser significance). Figure 28 gives examples of the ordering of bytes in multi-byte fields.

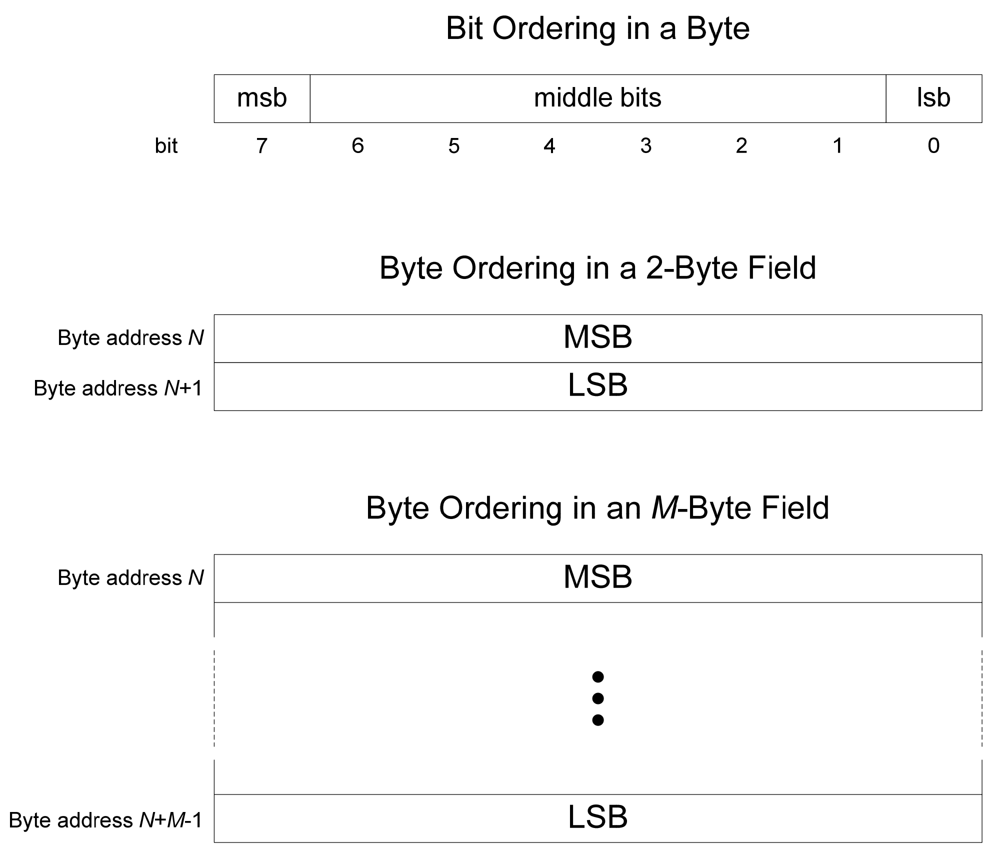

Figure 28: Byte Ordering in Multi-Byte Fields

Multiple character ASCII strings are left justified when stored in multi-byte fields (i.e., the first character in the string is stored in the MSB of a multi-byte field). If an ASCII string has fewer characters than maximum allowed by the field in which it is to be stored, all unused bytes in the field MUST be filled with the ASCII character for "space." Figure 29 gives the example of the 4-byte ASCII string "ABCD" stored in a 6-byte field.

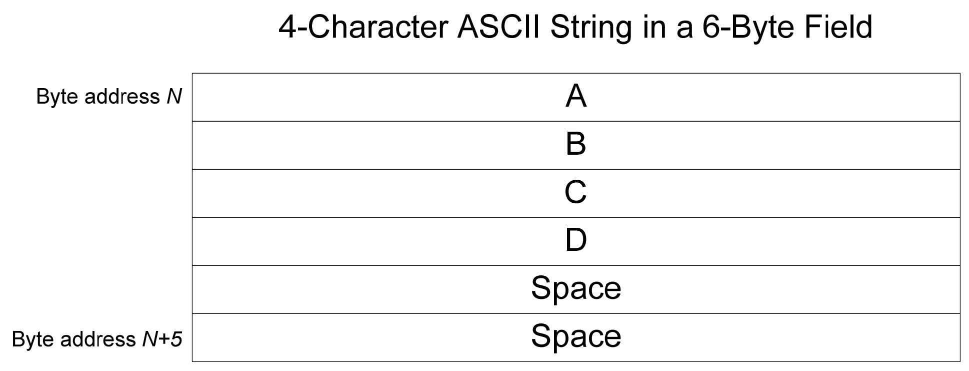

Figure 29: ASCII String Justification

Finally, Figure 30 gives an example of the byte ordering in the Controller Data Section of a DDF structure. The Controller Data Section is defined in Section 5.6 of this document.

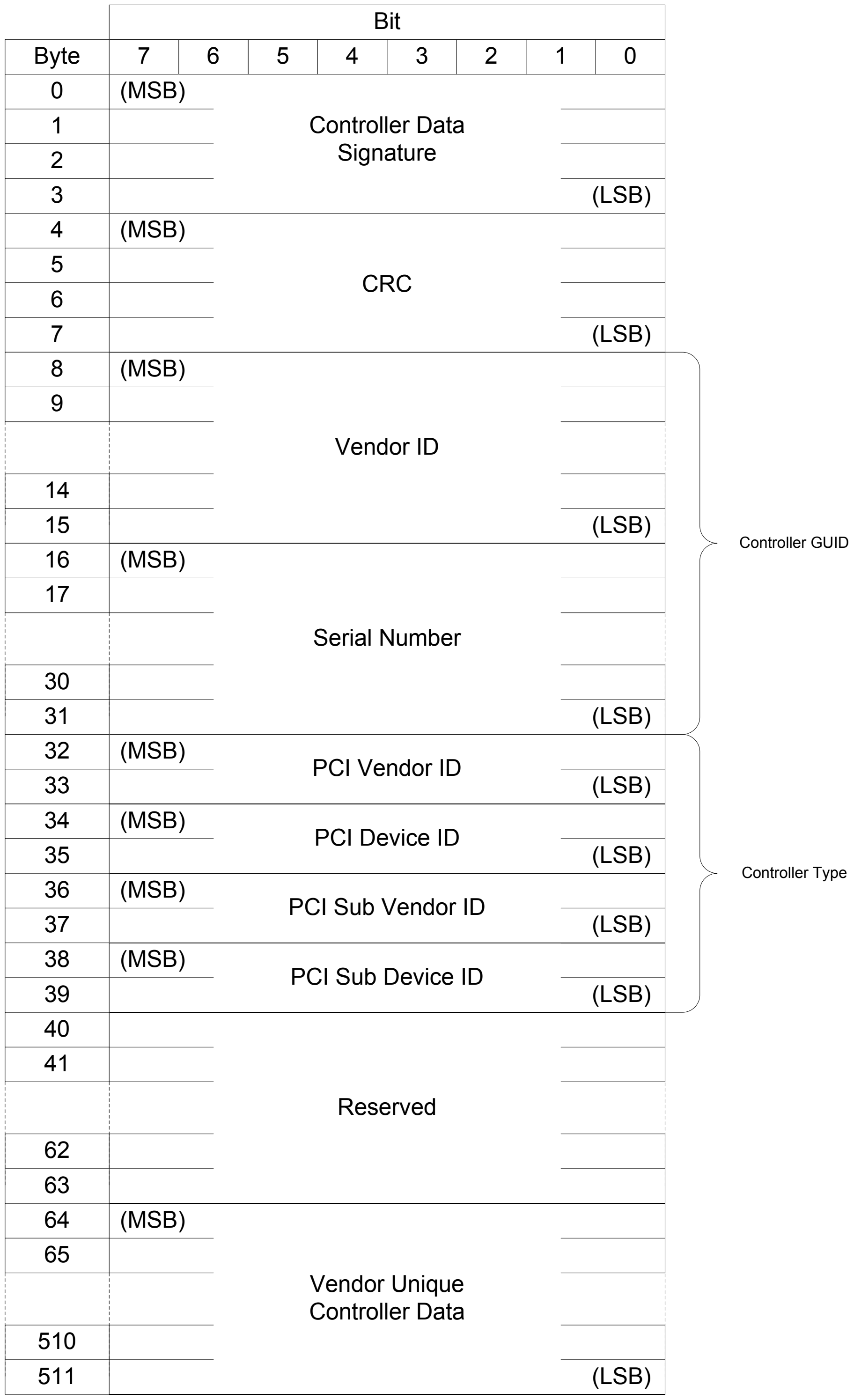

Figure 30: Controller Data Byte Ordering Example

5.3 Signatures, Timestamps and CRCs

Signatures are unique 32-bit fields indicating the start of different sections in the DDF structure. Table 17 lists the signatures used by the DDF structure.

Table 17: DDF Signatures

Signature Name	Signature Value	Notes
`DDF_Header`	`0xDE11DE11`	
`Controller_Data`	`0xAD111111`	
`Physical_Disk_Records`	`0x22222222`	
`Physical_Disk_Data`	`0x33333333`	
`Virtual_Disk_Records`	`0xDDDDDDDD`	
`VD_Configuration_Record`	`0xEEEEEEEE`	
`Spare_Assignment_Record`	`0x55555555`	
`VU_Configuration_Record`	`0x88888888`	
`Vendor_Specific_Log`	`0x01DBEEF0`	
`Bad_Block_Management_Log`	`0xABADB10C`	

Timestamps are used to provide chronological hints for a number of DDF structures. A timestamp MUST be the number of seconds that have occurred since midnight GMT of January 1, 1980 and the time the timestamp is created.

The CRC MUST be calculated for a given structure with 0xFFFFFFFF as the initial and default value for the CRC field in the structure. The CRC MUST calculated according to the CRC-32 algorithm described in ISO 3309 and in section 8.1.1.6.2 of ITU-T V.42. The CRC MUST be calculated using following polynomial:

```
X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X+1.
```

Once the CRC is calculated, the calculated value MUST replace the 0xFFFFFFFF stored in the CRC field during calculation.

5.4 GUIDs

The Globally Unique Identifier (GUID) is a structure used to uniquely identify the Controllers, Physical and Virtual Disks. It MUST be 24 bytes in length. A valid GUID MUST NOT use the values 0x20, 0x00 and 0xFF in Byte0 (MSB). These values in Byte 0 are reserved.

5.4.1 *Controller GUID*

The Controller GUID MUST be an ASCII string built by combining the T10 Vendor ID and the last 16 characters from the controller serial number. Information on T10 Vendor IDs and how to obtain one can be found at the T10 website (www.t10.org). Following is an example of a controller GUID, where the controller serial number is “11-10800BE10318” and the vendor name is “VendorID”,

V	E	N	D	O	R	I	D		1	1	-	1	0	8	0	0	B	E	1	0	3	1	8
0							7	8															23

If the T10 Vendor ID is less than eight characters or the controller serial number is less than 16 characters, the fields MUST be padded by ‘space’ (0x20) to achieve the required length. Padding MUST be placed between the vendor name and serial number.

NOTE: If there is no serial number defined for the controller, then the 16 byte serial number field MUST be built by concatenating an 8-byte ASCII representation of a 4-byte timestamp (bytes 8-15) and an 8-byte ASCII representation of a 4-byte random number (bytes 16-23). This number SHOULD be created on the first configuration and stored in the controller's non-volatile memory. This field SHOULD NOT be user modifiable so that it remains constant for the controller's life from user's perspective.

5.4.2 Physical Disk GUID

The Physical Disk (PD) GUID MUST be 24 byte field.

For physical disks that are accessed using SCSI commands (e.g., Parallel SCSI, Serial Attached SCSI, and Fibre Channel physical disks), the PD GUID MUST be built by combining the T10 Vendor ID of the disk vendor with the identifier returned by INQUIRY page 83h (Association=0 Identifier Type=1h, 2h, 3h or 8h) or the serial number returned in EVPD page 80h. If the identifier returned by INQUIRY page 83 or the disk serial number in EVPD page 80h is longer than 16 bytes, then the 16 least significant bytes MUST be used and the higher bytes discarded. If the serial number returned by EVPD page 80h is 'left justified' with spaces in the least significant bytes, the serial number MUST be 'right justified' before discarding the higher bytes and using the 16 least significant bytes. If the vendor name is less than eight characters or the disk serial number/identifier is less than 16 characters, the fields MUST be padded by 'space' (0x20) to achieve the required length. Padding MUST be placed between the vendor name and serial number/identifier. The following is an example of a PD_GUID for a SCSI disk, where the serial number/identifier is "5G45B673" and the T10 Vendor ID is "HDD."

H	D	D														5	G	4	5	B	6	7	3
0							7	8															23

When a serial number or INQUIRY page 83h identifier is not available for physical disks accessed using SCSI commands or the PD_GUID generated is not unique among the disks accessed by the controller, the controller MUST generate a forced serial number by concatenating an eight byte current date in "yyyymmdd" ASCII format and an eight byte hexadecimal ASCII representation of a four byte random number. The GUID generated using this serial number is considered a forced PD GUID and the Forced_PD_GUID_Flag in the PD_Type field of the Physical Disk Entry (Section 5.7.1) for the physical disk MUST be set. In addition, the Forced_PD_GUID_Flag in the disk's Physical_Disk_Data section MUST also be set (Section 5.10). The controller MUST guarantee that a Forced PD GUID is unique among the drives accessed by the controller. The following is an example of a forced PD GUID, where the date is February 1, 2004, the eight byte hexadecimal ASCII representation of the four byte random number is "AABBCCDD" and the disk's T10 Vendor ID is "HDD."

H	D	D						2	0	0	4	0	2	0	1	A	A	B	B	C	C	D	D
0							7	8															23

For ATA or SATA physical disks, PD_GUID MUST be built by combining the three character ASCII string "ATA", with the 20-byte disk serial number as reported in response to Identify Drive ATA command (bytes 20-39). A 'space' (0x20) MUST separate the string "ATA" from the disk serial number. The following is an example of PD_GUID for an ATA disk where the serial number is "BB583GX3389103443379."

A	T	A		B	B	5	8	3	G	X	3	3	8	9	1	0	3	4	4	3	3	7	9
0			3	4																			23

When a serial number is not available for ATA or SATA physical disks, the controller MUST generate one by concatenating an eight character current date in “yyyymmdd” ASCII format and a twelve byte hexadecimal ASCII representation of a six byte random number. The GUID generated using this serial number is considered a forced PD_GUID and the Force_PD_GUID_Flag in the PD_Type field of the Physical Disk Entry (Section 5.7.1) for the physical disk MUST be set. In addition, the Forced_PD_GUID_Flag in the disk’s Physical_Disk_Data section MUST also be set (Section 5.10). The following is an example of a forced PD GUID, where the date is February 1, 2004 and the twelve byte hexadecimal ASCII representation of the six byte random number is “AABBCCDDEEFF”

A	T	A		2	0	0	4	0	2	0	1	A	A	B	B	C	C	D	D	E	E	F	F
0			3	4																			23

5.4.3 Virtual Disk GUID

The VD GUID MUST be built by concatenating the creating controller’s T10 Vendor ID with a 16 byte unique identifier. The 16 byte identifier MAY be created by concatenating the Controller_Type field from Controller Data (Section 5.6), a 4-byte timestamp, and a 4-byte random number. Using this method, the VD_GUID provides data about age and the creating controller type. The following is an example of a VD_GUID with a vendor ID of “VENDORID”, a Controller_Type field of “AAAAAAAA”, a time stamp of “BBBB”, and a random number of “CCCC.”

V	E	N	D	O	R	I	D	A	A	A	A	A	A	A	A	B	B	B	B	C	C	C	C
0							7	8							15	16			19	20			23

Alternatively, the unique identifier MAY be created by the controller using a vendor specific method. If this method is used, the controller MUST guarantee that the identifier is unique for all virtual disks in the system.

5.4.4 DDF Header GUID

The DDF Header GUID MUST be built by concatenating the creating controller’s T10 Vendor ID, the Controller_Type field from Controller Data (Section 5.6), a 4-byte timestamp, and a 4-byte random number. The DDF_GUID provides data about age and the creating controller type. The following is an example of a DDF_GUID with a vendor ID of “VENDORID”, a Controller_Type field of “AAAAAAAA”, a time stamp of “BBBB”, and a random number of “CCCC.”

V	E	N	D	O	R	I	D	A	A	A	A	A	A	A	A	B	B	B	B	C	C	C	C
0							7	8							15	16			19	20			23

5.5 DDF Header

DDF headers are global in context with respect to disk groups or related physical disks. Three types of DDF header are defined: Anchor, Primary, and Secondary. The fields used in the DDF Headers are defined in Table 18. The Anchor, Primary and Secondary headers in the DDF structure MUST have the same value in all fields except the CRC, Sequence_Number, Open_Flag and Header_Type. The Anchor header is NOT REQUIRED to track configuration changes and does not use the Sequence_Number and Open_Flag fields. All DDF Headers MUST adhere to the field definitions and uses described in Table 18.

Table 18: DDF Header Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for DDF_Header Section (See Table 17)
CRC	4	CRC for the section (See Section 5.3). The CRC is covers the entire DDF Header section.
DDF_Header_GUID	24	GUID for the configuration or group (See Section 0)
DDF_rev	8	This field contains the revision number of the DDF specification to which this DDF structure adheres. If the DDF structure adheres to an official version of the DDF specification, this field MUST contain an ASCII string in xx.yy.zz format, where x, y, and z are integer values 0-9. xx is the major revision number. yy is the minor revision number and zz is the development revision number. Minor DDF revisions are backward compatible to other minor revisions within the same major revision category. This field MAY also be used to state that the structure does not adhere to an official version of the DDF specification but is a vendor unique implementation of the structure. If the DDF structure uses a vendor unique specification, this field MUST be an ASCII string in CCxxxxxx format, where C is any non-numeric character. x can be any character. NOTE: The specification allows vendor unique implementations to be marked as such to allow a controller to notify the system administrator before importing or erasing configurations created using a vendor unique implementation of a DDF structure.
Sequence_Number	4	Sequence number for configuration changes. Initial value MUST be 0x00000001. Wraparound is indicated by 0x00000000. Updating the Sequence_Number is OPTIONAL when the Vendor Specific Logs section, the Bad Block Management section or the Diagnostic Space section is updated. MUST be set to 0xFFFFFFFF in the Anchor header to indicate that configuration changes are not tracked in the Anchor Header.
Timestamp	4	Header update timestamp. MUST be set when the DDF header is updated. Used for a chronological hint only as this may not be consistent across controllers in the configuration.

Open_Flag	1	DDF open/close flag. The DDF Open_Flag MUST be set on a physical disk before starting configuration change writes to the physical disk. The flag MUST be cleared (i.e., set to 0x00) after a successful write of configuration data to the physical disk. Open_Flag MUST NOT set during log entry updates. This field MUST be set to 0xFF in the Anchor Header. The valid values for the flag are:
		
		0x00 = DDF closed;
		0x01-0x0F = DDF Opened;
		0x10-0xFE = Reserved;
		0xFF = Not tracking DDF open/close. Used in the Anchor Header.
		
		NOTE: Any value of 0x01 through 0x0F means that the DDF structure is open. An implementation MAY use different values for different reasons the DDF structure is open. However, these reasons are not defined in the specification and may differ between different implementations.

Foreign_Flag	1	Foreign_Flag MUST be set when the controller has not imported the configuration defined by this DDF structure. With this flag set (and CRC recalculated/updated), nothing else is changed in the DDF structure and this physical disk MUST be marked as foreign until configuration is imported, cleared or the physical disk is physically removed. The valid values for this field are: 0x00 = Not Foreign 0x01 = Foreign **NOTE:** This is to ensure that foreign configurations are imported with a user's explicit acknowledgement. If a user has not acknowledged such an action then this flag is set and foreign configuration is not presented to OS level. This bit is persistent to tolerate controller swaps, denoting original foreign physical disks when the new controller sees all physical disks as foreign.
Disk_Grouping	1	Disk Grouping enforced/not enforced. When this field is set Disk Grouping MUST be enforced. The valid values for this field are: 0x00 = Not Enforced 0x01 = Enforced
Reserved	13	Filled with 0xFF
Header_Ext	32	Filled with 0xFF – Reserved for future use.
Primary_Header_LBA	8	LBA of the DDF Header of Primary DDF Structure.
Secondary_Header_LBA	8	LBA of the DDF Header of the Secondary DDF structure. This field MUST be filled with 0xFF if redundant DDF structures are not stored.

Header_Type	1	0x00 = Anchor DDF Header; Stored on the last block of the physical disks 0x01 = Primary DDF Header 0x02 = Secondary DDF Header
Reserved	3	Filled with 0xFF
Workspace_Length	4	Block count for reserved workspace that MAY be used for optional vendor specific functions. At least 16MB MUST be reserved for workspace. Thus, this field MUST have a value greater than or equal to 32,768.
Workspace_LBA	8	LBA of the first block of the reserved workspace.
Max_PD_Entries	2	Maximum number of Physical Disk Entries. This is implementation dependent. Actual number of Physical Disks supported by a controller MAY be less than number of entries allowed. DDF structure implementations MUST only allow values of 15 (0x000F), 63 (0x003F), 255 (0x00FF), 1023 (0x03FF), and 4095 (0x0FFF).
Max_VD_Entries	2	Maximum number of Virtual Disk Entries. This is implementation dependent. Actual number of Virtual Disks supported by a controller MAY be less than number of entries allowed. DDF structure implementations MUST only allow values of 15 (0x000F), 63 (0x003F), 255 (0x00FF), 1023 (0x03FF), and 4095 (0x0FFF).
Max_Partitions	2	Maximum number of Configuration Record Entries (partitions) allowed per disk.
Configuration_Record_Length	2	Virtual Disk Configuration Record (Section 5.9.1) length in blocks. Depends on the value of Max_Primary_Element_Entries field and calculated using the following formula: 1+ROUNDUP(Max_Primary_Element_Entries*(4+8)/512)
Max_Primary_ Element_Entries	2	The maximum number of physical disks configurable within a basic VD. This field determines value of Configuration_Record_Length. DDF structure implementations MUST only allow values of 16 (0x0010), 64 (0x0040), 256 (0x0100), 1024 (0x0400), and 4096 (0x1000). **NOTE:** The value of this field determines value Configuration_Record_Length and size of two fields in the Virtual Disk Configuration Record: Physical_Disk_Sequence and Starting_Block.
Reserved	54	Filled with 0xFF
Controller_Data_ Section	4	Offset of the start of the Controller Data (Section 5.6) section from the Primary or Secondary DDF Header LBA
Controller_Data_ Section_Length	4	Length of the Controller Data section in number of blocks
Physical_Disk_Records_Section	4	Offset of the start of the Physical Disk Records (Section 5.7) section from the Primary or Secondary Header LBA

Physical_Disk_Records_Section_Length	4	Length of the Physical Disk Records section in number of blocks. This field is dependant on the entry in the Max_PD_Entries field. Values allowed are 2, 8, 32, 128 and 512
Virtual_Disk_Records_Section	4	Offset of the start of the Virtual Disk Records (Section 5.8) section from the Primary or Secondary DDF Header LBA
Virtual_Disk_Records_Section_Length	4	Length of the Virtual Disk Records section in number of blocks. This value depends on the entry in the Max_VD_Entries field. Values allowed are 2, 8, 32, 128 and 512
Configuration_Records_Section	4	Offset of the start of the Configuration Records (Section 5.9) section from the Primary or Secondary DDF Header LBA
Configuration_Records_Section_Length	4	Length of the section in Configuration Records section in number of blocks. The value of this field MUST equal: Configuration_Record_Length*(Max_Partitions+1).
Physical_Disk_Data_Section	4	Offset from of the start of the Physical Disk Data (Section 5.10)section from the start of the Primary or Secondary DDF Header LBA
Physical_Disk_Data_Section_Length	4	Length of the Physical Disk Data section in number of blocks
BBM_Log_Section	4	Offset of the start of the Bad Block Management Log (Section 5.11) from the start of the Primary or Secondary Header LBA. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0xFFFFFFFF.
BBM_Log_Section_Length	4	Length of the Bad Block Management Log section in number of blocks. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0x00000000.
Diagnostic_Space	4	Offset of the start of the Diagnostic Space (Section 5.11.1) section from the Primary or Secondary DDF Header LBA. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0xFFFFFFFF.
Diagnostic_Space_Length	4	Length of the Diagnostic Space section in number of blocks. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0x00000000.
Vendor_Specific_Logs_Section	4	Offset of the start of the Vendor Specific Logs (Section 5.13) from the Primary or Secondary DDF Header LBA. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0xFFFFFFFF.
Vendor_Specific_Logs_Section_Length	4	Length of the Vendor Specific Logs section in number of blocks. This is an OPTIONAL section and if the section is not implemented this field MUST be set to 0x00000000.
Reserved	256	Filled with 0xFF

5.6 Controller Data

This section is global in context and provides information about the last controller that operated on an attached RAID configuration. The Controller Data section MUST adhere to the field definitions and uses described in Table 19. When a new controller accesses a DDF structure previously operated on by another controller produced by the same vendor, it is up to the vendor implementation to determine whether or not the information stored in the Vendor_Unique_Controller_Data field is used by the new controller. When a new controller accesses a DDF Structure previously operated on by another controller produced by a different vendor, the new controller SHOULD erase the data contained in the Vendor_Unique_Controller_Data field and replace it with data specific to the new controller.

Table 19: Controller Data Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for Controller_Data Section (See Table 17)
CRC	4	CRC for the section (See Section 5.3). The CRC covers the entire Controller_Data section.
Controller_GUID	24	Controller GUID (See Section 5.4.1)
Controller_Type	8	For controllers with a PCI host interface, this field MUST contain the PCI ID of the controller. Byte 0-1: Vendor ID Byte 2-3: Device ID Byte 4-5: Sub Vendor ID Byte 6-7: Sub Device ID For controllers with a non-PCI host interface, bytes 6 and 7 of this field MUST be set to 0xFFFF. Bytes 0 through 5 MAY be set to any value at the discretion of the vendor.
Product_ID	16	This field is an ASCII field. This field MUST contain the Product ID of the controller that last operated on the RAID configuration. For controllers that operate using the SCSI protocol, this field MUST be set to the value of the PRODUCT IDENTIFICATION field returned by the controller in the standard Inquiry data. For ATA controllers, this field MUST be set to the value the controller would return for the PRODUCT IDENTIFICATION field in the standard Inquiry data as described in the SCSI / ATA Translation (SAT) draft specification. For controllers that use neither the SCSI nor ATA protocols, the controller MUST create a Product_ID.
Reserved	8	Filled with 0xFF; Reserved for future use.
Vendor_Unique _Controller_Data	448	Vendor unique controller data.

5.7 Physical Disk Records

The Physical Disk Records section MUST list all configured physical disks attached to the controller. A physical disk is considered configured when it belongs to a Virtual Disk, is marked as a Spare, is configured as a Legacy (pass-through) disk or is a physical disk that needs to be tracked by the controller for a vendor-specific reason. This section is global in context (i.e., every configured physical disk has information about all configured physical disks participating in the system described by the DDF structure). This allows the detection of missing physical disks after a controller swap.

The length of this section is variable and is a function of maximum number of physical disks supported by the implementation. It is RECOMMENDED that the maximum number of physical disks result in block (512 byte) aligned sizes of the Physical Disk Records section. All Physical Disk Records sections MUST adhere to the field definitions and uses described in Table 20.

Table 20: Physical Disk Records Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for Physical_Disk_Records Section (see Table 17).
CRC	4	CRC for the section (Section 5.3). The CRC covers the entire Physical_Disk_Records section.
Populated_PDEs	2	Number of Physical Disk Entries (PDE) used/populated
Max_PDE_Supported	2	Number of maximum PDE supported by vendor
Reserved	52	Filled with 0xFF
Physical_Disk_Entries	vari-able	64-byte PDEs. The number of PDE entries in this section is equal to Max_PD_Entries.

5.7.1 *Physical Disk Entries*

Physical Disk Entries (PDE) give minimal information about all physical disks attached to the controller. PDEs are stored in the Physical Disk Records section (Section 5.7) Each PDE is a 64 byte structure. Each PDE MUST adhere to the definitions and uses of the fields described in Table 21.

Table 21: Physical Disk Entry Fields

Element	Size (Bytes)	Description
PD_GUID	24	Physical Disk GUID (Section 5.4.2) for the physical disk. If this entry refers to the physical disk on which it is currently stored, the value of this field MUST contain the same value as the PD_GUID field in the Physical Disk Data section (Section 5.10). For unused entries, this field MUST be filled with 0xFF.
PD_Reference	4	Reference number to be used in VD Configuration Records (Section 5.9.1). This field MUST equal the value of the PD_Reference field of a physical disk's corresponding Physical Disk Data (Section 5.10)
PD_Type	2	Bit map describing a physical disk's type. Bit 0: 0 – Not using a forced PD_GUID (Section 5.4.2) 1 - Using a forced PD_GUID (also called the Forced_PD_GUID_Flag) Bit 1: 0 – Not participating in a VD

		1 – Participating in a VD Bit 2: 0 – Not a global spare 1 – Global spare (VD Configuration Records are ignored) Bit 3: 0 – Not a spare disk 1 – Spare disk (Bit2 and Bit3 are exclusive. Bit3 MUST have precedence over Bit2) Bit 4: 0 – Not foreign 1 – Foreign (This is a Foreign disk and the Foreign_Flag in the DDF Header on this disk must be set. See Section 5.5) Bit 5: 0 – Not a Pass-through/Legacy disk 1 - Pass-through/Legacy disk (No DDF structure is stored on this physical disk as DDF structures are only stored on configured physical disks and in the controller. If there are no other configured physical disks attached to the controller, then this information would only be stored in controller NVRAM. An implementation MAY restrict Pass-through/Legacy physical disks to systems with at least one configured disk attached to the RAID controller) Bit 6: Reserved Bit 7: Reserved Bits 8-11: Reserved Bits 15-12: Interface Type 0x0 = Unknown 0x1 = SCSI (parallel) 0x2 = SAS 0x3 = SATA 0x4 = FC 0x5-0xF = Reserved All reserved bits MUST be set to 0.
PD_State	2	State of the physical disk as part of one or more Virtual Disks. Bit 0: 0 – Offline 1 - Online Bit 1: 0 – OK 1 – Failed Bit 2: 0 – Not Rebuilding 1 – Rebuilding (Physical disk rebuilding for a failed physical disk) Bit 3: 0 – Not in Transition 1 - Transition (e.g., replacing a member physical disk through a copy operation) Bit 4: 0 – No PFA/SMART error alerts 1 - PFA/SMART error alerts Bit 5: 0 - No Un-recovered Read Errors 1 - Un-recovered Read Errors Bit 6: 0 - Not missing 1 - Missing (a physical disk may be Failed and Missing) Bit 7: Reserved

		Bits 8-15: Reserved All reserved bits MUST be set to 0. **NOTE-1:** Bit 1 MUST have precedence among Bits 0-3. If Bit 0 and Bit 1 are both set, then the physical disk’s status is Failed and was probably Online before status change. **NOTE-2:** For a physical disk participating in multiple VDs, the disk status Bits 1-3 MUST be set if any part of the physical disk is going through the corresponding process.
Configured_Size	8	Configured size of the physical disk in terms of highest addressable LBA during normal operation. The DDF structure MUST be stored at LBAs greater than or equal to the value in this field.

Path_Information	18	This field contains information on the path from a controller to a physical disk or the attach point of the physical disk. This field is used for information purposes only.
		
		The Path_Information structure depends on the physical disk Interface Type as defined in Bits 12-15 of the PD_Type field.
		
		For Interface Type = 0x01 (SCSI), the Path_Information structure MUST be interpreted as follows:
		
		Bytes 0-3: Path 0 information
		  Bits 7-0: LUN
		  Bits 15-8: SCSI Target ID
		  Bits 23-16: SCSI Channel
		  Bits 30-24: Reserved
		  Bit 31: 0 - Path 0 OK
		            1 – Path 0 Broken
		
		Bytes 4-7: Path 1 information (for systems with multiple paths for redundancy)
		  Bits 7-0: LUN
		  Bits 15-8: SCSI Target ID
		  Bits 23-16: SCSI Channel
		  Bits 30-24: Reserved
		  Bit 31: 0 - Path 1 OK
		            1 – Path 1 Broken
		
		Bytes 8-17: Reserved
		
		For Interface Type = 0x2 (SAS), the Path_Information structure MUST be interpreted as follows:
		
		Bytes 0-7: Path 0 - SAS address of the end device (edge expander or controller) where the physical disk is attached.
		Bytes 8-15: Path 1 - SAS address of the end device

		(edge expander or controller) where the physical disk is attached.
		
		Byte 16:
		  Bit 0-6: Path 0 PHY identifier
		  Bit 7: 0 – Path 0 OK
		          1 – Path 0 Broken
		Byte 17:
		  Bit 0-6: Path 1 PHY identifier
		  Bit 7: 0 – Path 1 OK
		          1 – Path 1 Broken
		
		All reserved bytes or bits MUST be set to 0.
		
		NOTE-1: Path 1 information bytes MUST be set to 0xFF when dual path support is not implemented for this physical disk.
		
		NOTE-2: For FC and SATA physical disks, this field is undefined and all bytes MUST be set to 0x00.

Reserved	6	Filled with 0xFF

5.8 Virtual Disk Records

The Virtual Disk Records section lists all the configured VDs tracked by the DDF structure. It has global context and must be stored on all configured physical disks. The length of this section is variable and is a function of the maximum number of configurable virtual disks supported by the system implementation. It is RECOMMENDED that the maximum number of configurable VDs supported result in a block (512 byte) aligned Virtual Disk Records section. The Virtual Disk Records section MUST adhere to the field definitions and uses described in Table 22.

Table 22: Virtual Disk Records Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for Virtual_Disk_Records Section (See Table 17)
CRC	4	CRC for the section (See Section 5.3). The CRC covers the entire Virtual Disk Records section.
Populated_VDEs	2	Number of Virtual Disk Entries used/configured
Max_VDE_Supported	2	Maximum number of VDEs supported by the implementation
Reserved	52	Filled with 0xFF
Virtual_Disk_Entries	variable	64 Byte VDEs. The number of VDEs in this field is equal to Max_VD_Entries.

5.8.1 Virtual Disk Entries

Virtual Disk Entries (VDE) contain information about the configured VDs. VDEs are stored in the Virtual Disk Records (Section 5.8). Each VDE MUST adhere to the definitions and uses described in Table 23.

Table 23: Virtual Disk Entry Fields

Element	Size	Description

	(Bytes)	
VD_GUID	24	Virtual Disk GUID (Section 5.4.3). For unused Virtual Disk Entries, this field MUST be filled with 0xFF.
VD_Number	2	The VD_Number MAY be used to map directly to the IDs (e.g., Logical Unit Numbers) presented to the OS as the result of an enumerate targets type command (e.g., SCSI Report LUNs command). For internal SCSI RAID controllers, the VD_Number MAY correspond directly to the LUN number reported to the OS. However, for external RAID controllers that have LUN masking and LUN mapping, the VD_Number cannot be used for this purpose and the RAID controller will have to implement its own method for maintaining this information. The VD_Number field needs to be managed during import MUST be consistent across reboots. May be the Valid range is 0-0x7FFF. 0x8000-0xFFFF is reserved.
Reserved	2	Reserved Filled with 0xFF
VD_Type	4	Bitmap describing the VD’s type. Bit 0: 0 – Private 1 - Shared (Disk Grouping MUST be enforced when this bit is set) Bit 1: 0 – No Disk Grouping 1 – Disk Grouping Enforced Bit 2: 0 – VD_Name in ASCII format 1 – VD_Name in Unicode format Bit 3: 0 – Owner ID Not Valid 1 – Owner ID Valid Bits 15-4: Reserved Bits 31-16: Primary Controller GUID CRC. Used as a hint for the last controller that owned a VD in a clustered environment. The value stored in these bits MUST be the 16-bit CRC of the Controller GUID. All reserved bits MUST be set to 0. **NOTE-1:** Bits 16-31 MUST be ignored when Bit 0 is clear. **NOTE-2:** If Bit 0 is set, the value of Bit 1 MUST be ignored since Disk Grouping is automatically enforced. **NOTE-3:** If the Disk_Grouping field in the DDF Header is set to 0x01, Bit 1 MUST be ignored since Disk Grouping is automatically enforced for all disks. **NOTE-4:** If Bit 1 is set, disk grouping MUST be enforced for all disks participating in the virtual disk. This bit allows disk grouping to be enforced on some drives in a DDF configuration and not enforced on other drives. **NOTE-5:** If Bit 2 is set, the value stored in the VD_Name field MUST be compliant with Version 4.0 of the Unicode standard.

		NOTE-6: If Bit 3 is clear, bits 16-31 are invalid. If Bit 3 is set, bits 16-31 are valid.
VD_State	1	Bitmap describing the state of the virtual disk. Bits 2-0: 0x0 - Optimal (The VD is operating and has experienced no failures of the disks that comprise VD.) 0x1 = Degraded (The VD has experienced at least one failure of the disks that comprise the VD. One more disk failure could result in the VD being placed in the “Failed” state indicating data loss has occurred.) 0x2 = Deleted (The VD has been marked as deleted by the system.) 0x3 = Missing (The VD has been marked as missing by the system.) 0x4 = Failed (The VD has experienced enough failures of the disks that comprise the VD for unrecoverable data loss to occur.) 0x5 = Partially Optimal (The VD has experienced disk failures. The VD can experience at least one more disk failure before it is placed in the “Degraded” state.) 0x6 = Reserved 0x7 = Reserved Bit 3: 0 – Not Morphing 1 – Morphing (The VD is performing a morphing activity: RAID level migration, online capacity expansion, shrinking, defragmenting, stripe size migration, etc.) Bit 4: 0 – VD Consistent 1 – VD Not Consistent Bits 7-5: Reserved All reserved bits MUST be set to 0. **NOTE-1:** When the Morphing bit (Bit 3) is set, this VD cannot be imported to another controller from a different vendor as the new controller will not be able to determine the state or type of morphing activity. It is RECOMMENDED that no voluntary migrations of physical disks between controllers from different vendors be allowed when VDs associated with these physical disks are in a Morphing state. **NOTE-2:** The VD Not Consistent bit (Bit 4) MUST be set when a controller cannot guarantee a VD is consistent. The term consistent designates the state when all writes to a VD by client computer system, that are acknowledged as successfully completed by the controller, have been correctly

		written to the VD, including any redundancy information (e.g., parity). This bit SHOULD be cleared for a VD on a clean shutdown of the system. The bit MAY also be cleared during idle periods after a controller has determined that all writes have been consistently written to the VD. The bit SHOULD NOT be used as an actual cache synchronization flag.
Init_State	1	Bits 1-0: Initialization State 0x00 = Not Initialized 0x01 = Quick Initialization in Progress 0x02 = Fully Initialized 0x03 = Reserved Bits 5-2: Reserved Bits 7-6: User Access Mode 0x00 = Read/Write 0x01 = Reserved 0x02 = Read Only 0x03 = Blocked (User reads and writes denied) **NOTE:** The Quick Initialization in Progress state MAY be used to indicate that the VD is being initialized but is still available for read and/or write access. Some controllers provide this capability.
Reserved	14	Reserved Filled with 0xFF
VD_Name	16	This field MAY be used to contain a 16 byte ASCII or Unicode string that is the name of the virtual disk. Bit 2 of the VD_Type field MUST be used to determine the Unicode or ASCII format of this field. This field MAY match the volume name used by some operating systems. If this field is not used, all bytes MUST be set to zero.

5.9 Configuration Records

The Configuration Records section is local in context (i.e., relevant to physical disks where it is stored). It is variable in length and the length is stored in the Configuration_Record_Section_Length field in the DDF Header (Section 5.5). The Configuration_Record_Section_Length is implementation dependent and MUST be managed during import/merge operations between different controller implementations. A physical disk MAY have multiple partitions. Thus, a physical disk MAY be simultaneously listed in multiple configuration records. These configuration records MUST only be stored on the relevant (participating) physical disks. This section contains multiple records and these records MUST be either:

a. No Virtual Disk Configuration Records;

b. All Virtual Disk Configuration Records;

c. One Spare Assignment Record;

d. All Virtual Disk Configuration Records and one a Spare Assignment Record; or

e. All Vendor Unique Configuration Records.

The Spare Assignment Record MUST only be stored on the relevant physical disk. The number of records stored on a physical disk MUST be one plus the value of the Max_Partitions field in the DDF header (Section 5.5). One extra record is provided to allow writing a modified entry before invalidating original. In the event that a Configuration Record section contains Max_Partitions Virtual Disk Configuration Records and One Spare Assignment Record, there will not be enough space to allow writing a modified entry before invalidating the original. It is up to the implementation to determine how to handle this situation. In certain situations, an implementation MAY convert a revertible spare to invalidate the Spare Assignment Record to provide the extra record for writing the new configuration. Another method would be to copy the Spare Assignment Record to Workspace temporarily and to copy it back after the new configuration record is written. To invalidate a Virtual Disk Configuration Record, a Spare Assignment Record, or a Vendor Unique Configuration Record, 0xFFFFFFFF MUST be stored in the Signature field of the record.

5.9.1 *Virtual Disk Configuration Record*

The size of VD configuration record MUST be the same as Configuration_Record_Length indicated in the DDF header (Section 5.5). Each VD configuration record MUST adhere to the definitions and uses described in Table 24.

Table 24: Virtual Disk Configuration Record Fields

Element	**Size (Bytes)**	**Description**
Signature	4	For used entries, this field MUST be set to the Unique Signature for VD_Configuration_Record (Table 17). For unused entries, this field MUST be set to 0xFFFFFFFF. **NOTE:** Invalidating a used entry MAY be accomplished by only writing 0xFFFFFFFF to this field. This allows the possibility of un-deleting a configuration entry.
CRC	4	CRC of the VD Configuration Record (Section 5.3)
VD_GUID	24	VD GUID (Section 5.4.3)
Timestamp	4	Configuration or reconfiguration timestamp. If this field is different than the timestamp in the VD's VD GUID, the configuration has changed since the VD was created. New timestamps MUST be greater than existing timestamps. Timestamps SHOULD only be used as chronological hints since they may not be consistent across controllers
Sequence_Number	4	Sequence number for configuration record changes. The initial value of this field MUST be 0x00000001. Subsequent configuration changes MUST increment this field by one. A wraparound of the sequence number MUST be indicated by the value 0x00000000.
Reserved	24	Filled with 0Xff
Primary_Element_Count	2	The number of physical disks used in a basic VD.

Strip_Size	1	Stripe depth in (2^n)*512 format where n is the value of the field.
		
		Examples:
		n=0 – 512B stripe depth
		n=1 – 1KB stripe depth
		n=2 – 2KB stripe depth
		n=3 – 4KB stripe depth
		n=7 – 64KB stripe depth
		n=11 – 1MB stripe depth
		etc.

Primary_RAID_Level	1	RAID level of the BVD as defined in Table 1.
RAID_Level_Qualifier	1	The RAID level qualifier as defined in Section 4.2.
Secondary_Element_Count	1	The number of BVDs in a VD with a secondary RAID level as defined in Section 4.2.19 (e.g., RAID 50). For VDs without a secondary RAID level, this field MUST be set to 1.
Secondary_Element_Seq	1	Position of current basic VD in secondary VD. Valid only if secondary element count > 1.
Secondary_RAID_Level	1	The secondary RAID level as defined in Table 12. Valid only if Secondary_Element_Count > 1.
Block_Count	8	This field applies to the physical disk on which the configuration record is stored. The field states the size in blocks of the partition on the physical disk that is participating in the VD described by this configuration record.
VD_Size	8	The size of the user addressable space in the virtual disk. The size is stated in number of blocks.
Reserved	8	Filled with 0xFF
Associated_Spares	32	This field contains eight 4-byte entries for associated spare physical disks. Each used entry MUST contain the PD_Reference defined in the Physical Disk Entry (Section 5.7.1) for the associated spare physical disk. Unused entries MUST be set to 0xFFFFFFFF. Bytes 0-3: Spare Entry 0 Bytes 4-7: Spare Entry 1 Bytes 8-11: Spare Entry 2 Bytes 12-15: Spare Entry 3 Bytes 16-19: Spare Entry 4 Bytes 20-23: Spare Entry 5 Bytes 24-27: Spare Entry 6 Bytes 28-31: Spare Entry 7 **NOTE:** This field is used to detect missing dedicated spares as Spares Assignment Records are local.

Cache Policies & Parameters	8	Cache policies for the VD. Cache policies, algorithms and parameters are implementation dependent. Therefore, bytes 1 through 7 are vendor specific. Byte 0 is a bit field where the bits are defined as: Bit0: 0-WriteThrough 1-Writeback Bit1: 0-Always (ignored if Bit0=0) 1-Adaptive (ignored if Bit0=0) Bit2: 0-No Read Ahead 1-Read Ahead Bit3: 0-Always (ignored if Bit2=0) 1-Adaptive (ignored if Bit2=0) Bit4: 0-No write caching if battery low or not present 1-Write caching allowed if battery low or not present Bit5: 0-No write caching allowed 1-Write caching allowed Bit6: 0-No read caching allowed 1-Read caching allowed Bit7: 0-No vendor specific caching algorithm 1-Vendor specific caching algorithm **NOTE-1:** Bits 4-6 are master enable/disable settings for cache parameters. **NOTE-2:** If bit7 is set, then bits 0-3 SHOULD left clear and ignored. Bits 4-6 SHOULD still be used. **NOTE-3:** During Vendor migration, if bit7 is found set, then the user SHOULD be notified and bit7 SHOULD be cleared. Bits 0-3 SHOULD be set to controller default and bits 4-6 SHOULD be preserved.
BG_Rate	1	This field is used to assign background task priorities to individual VDs. Examples of background tasks are: initialization, RAID level migration, expansion, etc. If the field is set to 0xFF, no background task rate has been set for this VD and the controller defaults SHOULD be used. If the field has a value of 0x00 through 0xFA, the VD SHOULD be assigned this relative priority for all background tasks. 0x00 is the lowest priority and 0xFA is the highest. **NOTE:** The actual weighting of the background task priorities is implementation dependent.
Reserved	3	Filled with 0xFF
Reserved	52	Filled with 0xFF
Reserved	192	Filled with 0xFF
V0	32	Filled with 0xFF – Reserved for future use

V1	32	Filled with 0xFF – Reserved for future use
V2	16	Filled with 0xFF – Reserved for future use
V3	16	Filled with 0xFF – Reserved for future use
Vendor Specific Scratch Space	32	Vendor unique information. All bytes in this field MUST be set to 0xFF when not used by the implementation.
Physical_Disk_Sequence	vari-able	This field gives the sequence of physical disks in the BVD. This field contains multiple 4-byte entries. Each used entry MUST be a PD_Reference from a Physical_Disk_Entry (Section 5.7.1). Unused entries MUST be filled with 0xFFFFFFFF. Used entries provide the primary element sequence in ascending order. For the BVD, the number of used entries MUST equal the Primary_Element_Count in the Virtual_Disk_Configuration_Record for the BVD. If a physical disk that was part of this VD has been removed from the VD, reconfigured as part of another VD, and then returned to the system containing the original VD, the PD_Reference entry for the returned physical disk MUST be set to 0x00000000. This is to prevent the controller from erroneously reading the data from the replaced drive as part of the current VD. **NOTE:** This is a variable size field. Its size is determined by the following formula: Max_Primary Element_Entries * 4
Starting_Block	vari-able	This field gives the starting LBAs for the partitions of the physical disks participating in the BVD. This field is variable in size and is comprised of 8-byte entries. Each entry contains an 8-byte LBA. Each entry corresponds to the physical drive with the PD_Reference stored in the same entry index in Physical_Disk_Sequence. Each entry MUST contain the starting LBA of its corresponding physical disk's partition that is part of the BVD. Unused entries MUST be filled with 0xFFFFFFFFFFFFFFFF. **NOTE:** This is a variable size field. Its size is determined by the following formula: Max_Primary_Element_Entries * 8

5.9.2 Vendor Unique Configuration Record

The vendor unique configuration record allows proprietary implementations within DDF structures. Its size MUST be the same as the Configuration_Record_Length field of the DDF header (Section 5.5). This structure is vendor specific except for the first 32-bytes which MUST adhere to the definitions and uses described in Table 25.

Table 25: Vendor Unique Configuration Record Mandatory Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for VU_Configuration_Record (Table 17)
CRC	4	CRC of the Vendor_Unique_Configuration_Record (Section 5.3)
VD_GUID	24	VD GUID (5.4.3)

5.9.3 Spare Assignment Record

A Spare Assignment Record MUST be present on a physical disk only if the physical disk is a spare. The length of this record MUST be equal to the value in the Configuration_Record_Length field of the DDF Header (Section 5.5). Spare Assignment Records MUST adhere to the filed definitions and uses described in Table 26.

Table 26: Spare Assignment Record Fields

Element	Size (Bytes)	Description
Signature	4	Unique Signature for Spare_Assignment_Record (Table 17)
CRC	4	CRC of the Spare_Assignment_Record (Section 5.3)
Timestamp	4	
Reserved	7	0xFFFFFFFFFFFFFF

Spare_Type	1	Bit0: 0 – Global 1 – Dedicated Bit1: 0 – Committable 1 – Revertible Bit2: 0 – Not Active 1 – Active Bit3: 0 – No Enclosure Affinity 1 – Enclosure Affinity Bits 7-4: Reserved The reserved bits MUST be set to 0. **NOTE-1:** Committable spares (Bit1 = 0) become permanent members of VDs after a rebuild. Revertible (Bit1 = 1) spares revert back to Spare status after the replacement of the original failed physical disk. An import, merge, or roaming action MAY require commitment of an active (failed-over) revertible spare. **NOTE-2:** An active spare is currently host user data on a VD for failed physical disk. **NOTE-3:** A spare with Enclosure Affinity MAY only be used as a spare for VDs that reside on disks in the same enclosure as the spare. Keeping track of which disks are associated with a particular enclosure is implementation dependent.
Populated_SAEs	2	Number of Spare Assignment Entries used
Max_SAE_Supported	2	Maximum number of SAE supported by vendor.
Reserved	8	Filled with 0xFF
Spare_Assignment_Entries	vari-able	Packed 32-byte spare assignment entries for dedicated spare physical disks (See Section 5.9.3.1). All bytes of unused entries MUST be set to 0xFF. For active revertible global spares (Spare_Type: Bit0 = 0, Bit1 = 1, Bit2 = 1), a Spare Assignment Entry MUST be present for each VD that the Global spare is currently hosting user data for a failed physical disk. For Non-Active Global spares (Spare_Type: Bit0 = 0, Bit2 = 0), all entries MUST be unused. For dedicated spares (Spare_Type: Bit0 = 1), a Spare Assignment Entry MUST be present for each VD that the dedicated spare may serve as a replacement. If a dedicated spare is active (Spare_Type: Bit2 = 1), the spare is replacing failed disks on any VDs that have VD_Configuration_Records on the spare and also have Spare Assignment Entries in this field. **NOTE:** This is a variable size field which depends on Configuration_Record_Length. It is equal to Configuration_Record_Length minus 32 bytes.

5.9.3.1 Spare Assignment Entry

A Spare Assignment Entry identifies the Virtual Disks to which a dedicated spare physical disk is assigned. The Spare Assignment Entry also allows a dedicated spare to be assigned to a particular Secondary Element of a hybrid VD (e.g., RAID50). Spare Assignment Entries MUST adhere to the field definitions and uses described in Table 27.

Table 27: Spare Assignment Entry Fields

Element	Size (Bytes)	Description
VD GUID	24	VD GUID of a VD to which this spare physical disk is assigned
Secondary_Element	2	VD Secondary Element number to which this spare physical disk is assigned. If the spare is not assigned to a particular secondary element, this field MUST be set to 0xFFFF.
Reserved	6	Filled with 0xFF

5.10 Physical Disk Data

This section is local in context (i.e., only relevant to the physical disk on which it is stored). It stores the Physical Disk GUID and the Reference Number that is used in the Physical Disk Entry (see Section 5.7.1) for this physical disk. If the Physical Disk GUID is not available (possibly due to a missing or inaccessible disk serial number) or not usable then a GUID MUST be generated (see Section 5.4.2) and the Forced_PD_GUID_Flag set to 1. Physical Disk Data sections MUST adhere to the field definitions and uses described in Table 28.

Table 28: Physical Disk Data Fields

Element	Size (Bytes)	Description
Signature	4	Unique signature for Physical_Disk_Data (Table 17)
CRC	4	CRC of the Physical_Disk_Data section (Section 5.3)
PD_GUID	24	Physical Disk GUID for this physical disk (Section 5.4.2)
PD_Reference	4	Physical Disk Reference Number. For disks accessed by SCSI commands, the reference number MUST be built by computing a 32-bit CRC of the data contained in EVPD page 80. For ATA and SATA disks, the reference number MUST be built by computing a 32-bit CRC of the data returned by the Identify Drive command. The reference number is global across the controller configuration and each disk on the controller MUST have a unique reference number. In case of conflict, a new value MUST be calculated. The method is implementation dependent but MUST insure that all disks in the configuration have

		unique PD_Reference values. This field MUST be managed when importing configured disk groups to insure the PD_Reference values remain unique. The values 0x00000000 and 0xFFFFFFFF are reserved and MUST not be used as PD_Reference values.
Forced_Ref_Flag	1	0x00 = No 0x01 = Yes The Forced_Ref_Flag is set when the PD_Reference is generated in a manner different than described in the PD_Reference field description. This situation occurs in the event two or more PD_References for different disks have the same value.
Forced_PD_GUID_Flag	1	0x00 = No 0x01 = Yes Once a Forced PD GUID is created for a physical disk (see Section 5.4.2), the Forced PD GUID MUST be associated with the physical disk as long as the disk participates in the configuration.
Vendor Specific Scratch Space	32	Vendor unique information. All bytes in this field MUST be set to 0xFF when not used by the implementation.
Reserved	442	Filled with 0xFF

5.11 Bad Block Management Log

This section is local in context (i.e., only relevant to the physical disk where it is stored). It is an OPTIONAL section that allows block mapping and reassignment for defective blocks or the marking of blocks that cannot be recovered during a rebuild or recovery operation due to a media error. The Bad Block Management (BBM) Log section must adhere to the field definitions and uses described in Table 29.

Table 29: Bad Block Management Log Fields

Element	**Size (Bytes)**	**Description**
Signature	4	Unique Signature for Bad_Block_Management_Log (Table 17)
CRC	4	CRC for the BBM Log (Section 5.3). The CRC covers the entire Bad_Block_Management_Log section.
Entry_Count	2	Count for valid mapped/marked block entries. When there are no mapped/marked blocks, this field MUST be set to 0x0000. The maximum entry for this field is the smaller of 254 or Reserved_Spare_Block_Count.
Reserved_Spare_Block_Count	4	Up to 0xFFFFFFFF. The value of this field is implementation dependent.
Reserved	10	Filled with 0xFF
First_Spare_LBA	8	LBA of first spare block.
Mapped_Block_Entries	4064	Up to 254 packed 16-byte entries for mapped LBAs

		(see Section 5.11.1). The entries MUST be sequentially populated. Unused entries must be filed with 0xFF.

5.11.1 *Mapped/Marked Block Entry*

A Mapped/Marked Block Entry identifies a contiguous group of blocks that are remapped to another section on the physical disk or marked as unrecoverable due to a media error. Mapped/Marked Block Entries MUST adhere to the field definitions and uses described in Table 30.

Table 30: Mapped/Marked Block Entry Fields

Element	Size (Bytes)	Description
Defective_Block_Start	8	The LBA of the first defective block in a contiguous group of defective blocks that are remapped or marked as unrecoverable.
Spare_Block_Offset	4	If this field is set to 0xFFFFFFFF, the blocks indicated by this Mapped/Marked Block Entry are marked as unrecoverable due to a media error. If this field is not set to 0xFFFFFFFF, the blocks MUST be remapped to a new location. The field contains the offset from the First_Spare_LBA where the group of defect blocks MUST be remapped.
Remapped_Marked_Count	2	Number of blocks to be remapped or marked as unrecoverable.
Reserved	2	Filled with 0xFF

5.12 Diagnostic Space

The Diagnostic Space section is local in context. This section is OPTIONAL. If implemented, the size of this section is stated in the Diagnostic_Space_Length field of the DDF Header (Section 5.5). The section MAY be used for destructive read/write tests on the physical disk.

5.13 Vendor Specific Logs

The Vendor Specific Logs section is OPTIONAL. The structure of this section is implementation/vendor specific except for the first 32 bytes which are used for mandatory fields. The first 32 bytes MUST adhere to the field definitions and uses described in Table 31.

Table 31: Vendor Specific Logs Mandatory Fields

Element	(Size Bytes)	Description
Signature	4	Unique signature for Vendor_Specific_Logs section (Table 17)
CRC	4	CRC for the Vendor Specific Logs section (Section 0) The CRC is covers entire Vendor_Specific_Logs section. Any unused bytes in the section MUST be filled with 0xFF.
Log_Owner	8	T10 vendor ID of the log owner. If the vendor specific logs are used, this field MUST be

		populated with the T10 Vendor ID of the vendor that created the log. It MUST be set to 0xFFFFFFFFFFFFFFFF when the logs are empty or not used.
Reserved	16	Filled with 0xFF