Appendix B

NMO Buffer Layout

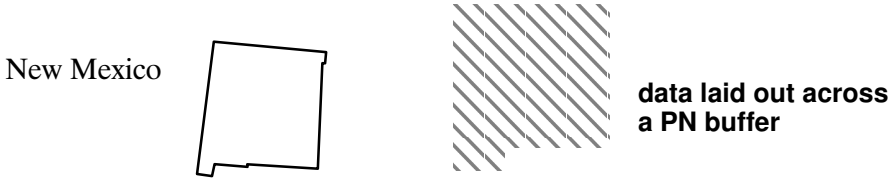
NOTE

This appendix applies only to SFS (SDA) file I/O. I/O on network connections is limited to full word quantities (multiples of 4 bytes), but it is not subject to the disk stripe caveats discussed here. The *CM-5 HIPPI User's Guide* contains information about I/O to and from network connections.

You should understand NMO (New Mexico order) if you are using the CMOST I/O interface (**`CM_read`**, **`CM_write`**). If you are using an I/O interface provided by a programming language (e.g., CM Fortran utility library, Fortran I/O statements) or the CMFS library, all issues that pertain to NMO are taken care of automatically. (You may wish to understand NMO to obtain best performance, though; see also Chapter 3.)

Copyright © 1994 Thinking Machines Corporation

NMO is so named because of the pattern in which bytes from a file are placed into the I/O buffer. The shape of the filled buffer often resembles the outline of the state of New Mexico:



B.1 SDA Files Are in Canonical (Serial) Order

Data from a file is stored on the SDA in canonical (serial) order, which both enables a CM-5 file to be portable across different SDA file system configurations and allows a CM-5 file to be used by a serial computer, and vice versa. See Figure 11.

There is yet another advantage to storing data in serial order: A file written from a partition of one size can be read by a partition of a different size, since the packets are always put back on disk in the same order.

Using the SDA, the smallest amount of data the operating system reads or writes is one *disk stripe*: 1 packet * the number of SDA data disks, where 1 packet is 16 bytes. (The term *disk stripe* is synonomous with *blocking factor*.) When a file system is equipped with a parity disk, parity is generated on writes for each disk stripe — 1 parity packet per disk stripe.

The SDA file system by default performs reads without parity checking, though the system administrator can set the file system to perform reads with parity checking. (Users cannot choose on a per-process basis whether to read with or without parity checking.)

NOTE: System administrators can use the **`sdacc`** command **`efrwp`** to set the file system to perform reads with parity checking. See the *CM-5 System Administrator's Guide*, Version 7.3, for more information.

Copyright © 1994 Thinking Machines Corporation

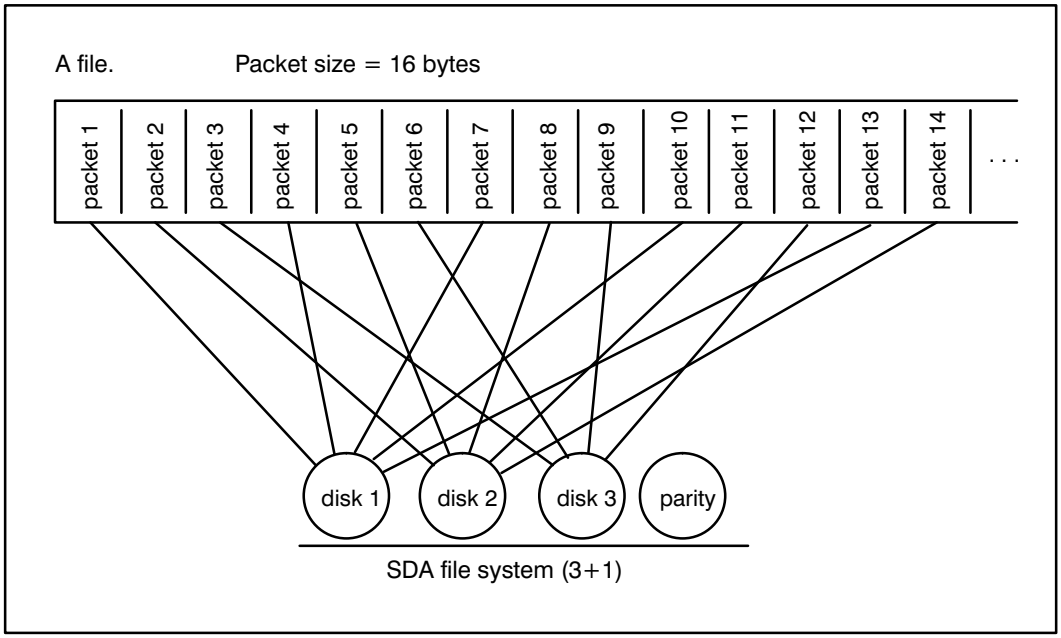


Figure 11. A file's packets are laid out on disk in canonical (serial) order.

B.2 Buffer Layout

The easiest way to understand NMO buffer layout is to understand how the **CM_read** and **CM_write** system calls work.

B.2.1 The CM_read() System Call

To read from a file into NMO buffer space, execute **CM_read**:

```
int = CM_read(fd, pbuf, count)
```

During a **CM_read** system call, the NMO buffer space of each PN receives, one after the other (wrapping back to the first PN), an entire disk stripe (or part of one, when there is not an entire disk stripe left), until *count* bytes are read. Depending on the size of *count,* relative to the size of the partition and the disk stripe, one or more PNs may receive more data than succeeding

Copyright © 1994 Thinking Machines Corporation

PNs, which would give the filled NMO buffer its New Mexico−shaped outline.

- If the total amount to be read is a multiple of the *partition stripe* factor — the disk stripe factor * number of PNs — each PN will have the same amount of data: one or more disk stripes. See Figure 12.

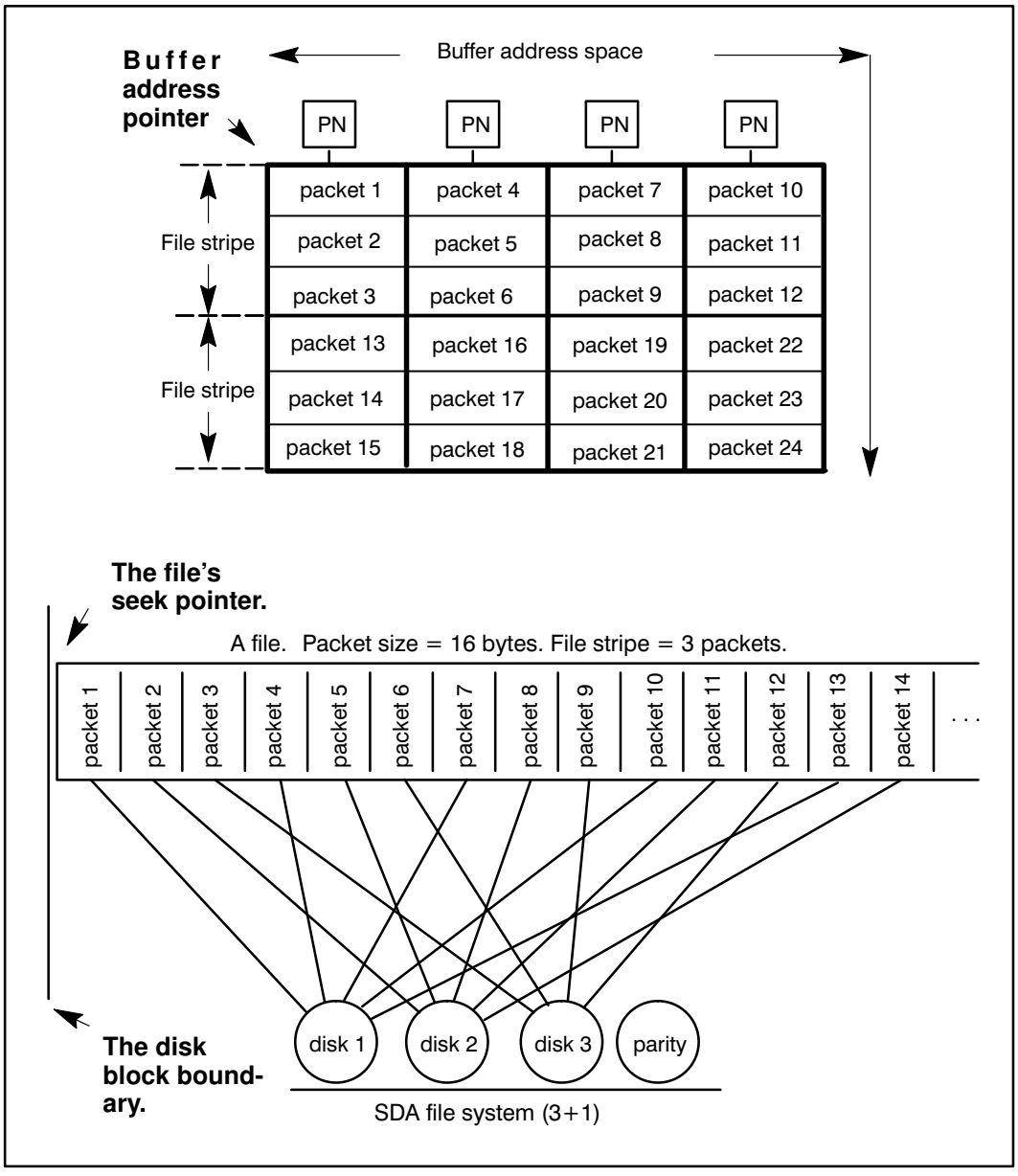


Figure 12. A file's packets are layed out in PN memory in NMO order. The upper half of the diagram shows the result of `CM_read`(*fd, pbuf*, `384`).

Copyright © 1994 Thinking Machines Corporation

- If the total amount to be read is not a multiple of the partition stripe factor, one or more PNs will have more data than do succeeding PNs:
  - If the amount to be read is a multiple of the disk stripe factor, one or more PNs will have one disk stripe more than succeeding PNs. See Figure 13.
  - If the amount to be read is not a multiple of a disk stripe, the PN that receives the last of the data will receive a partial disk stripe. See Figure 14. Note that partial packets are supported.

When allocating NMO buffer space, the user must take into account not only the projected size of the *count* argument, but also the SDA's file stripe factor. Since the system performs I/O in disk-stripe-sized chunks, the NMO buffer size must be at least as large as *count* rounded up to the next disk stripe boundary. If there is not enough buffer space to accommodate an entire disk stripe at the end of the read operation, **`CM_read`** will return **`EFAULT`**.

The file's seek pointer determines where in the file to initiate the read and where in the NMO buffer to begin placing data:

- If the pointer is on a disk stripe boundary, **`CM_read()`** initiates data placement at the beginning of the allocated NMO buffer.
- If the pointer is not on a disk stripe boundary, the offset data — the data that falls between the file's seek pointer and the preceding disk stripe boundary — is still placed into the NMO buffer, even though it is not needed ultimately. Most of the CM-5 interfaces (CMMD, C*, etc.) that provide routines for copying data from the NMO buffer to a parallel variable accept an argument that specifies the amount of the unneeded data — the "notch" — so it is not copied into the parallel variable; see Figure 15.

Copyright © 1994 Thinking Machines Corporation

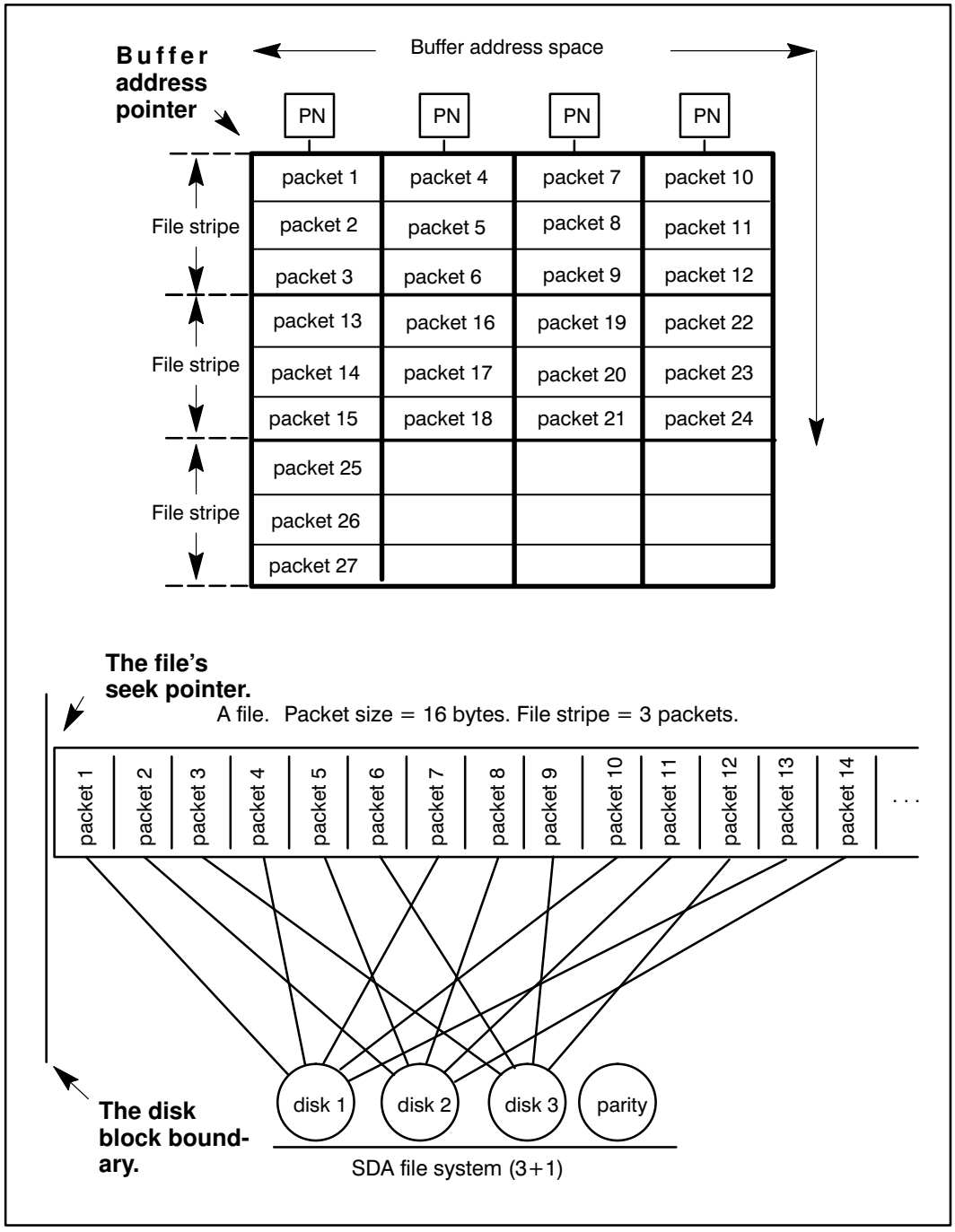


Figure 13. A file's packets are layed out in PN memory in NMO order. The upper half of the diagram shows the result of `CM_read`(*fd, pbuf*, `432`).

Copyright © 1994 Thinking Machines Corporation

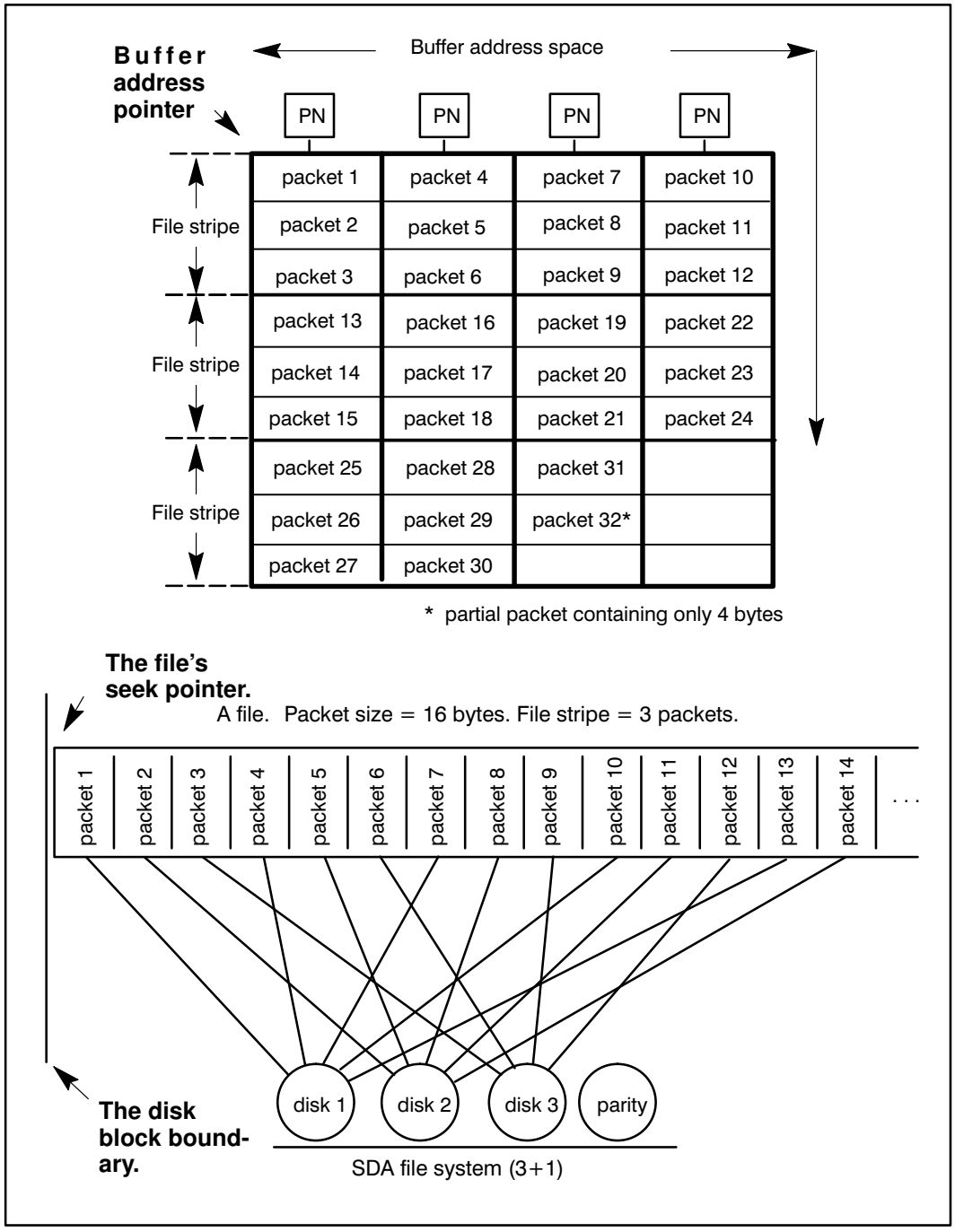


Figure 14. A file's packets are layed out in PN memory in NMO order. The upper half of the diagram shows the result of `CM_read`(*fd, pbuf*, `500`).

Copyright © 1994 Thinking Machines Corporation

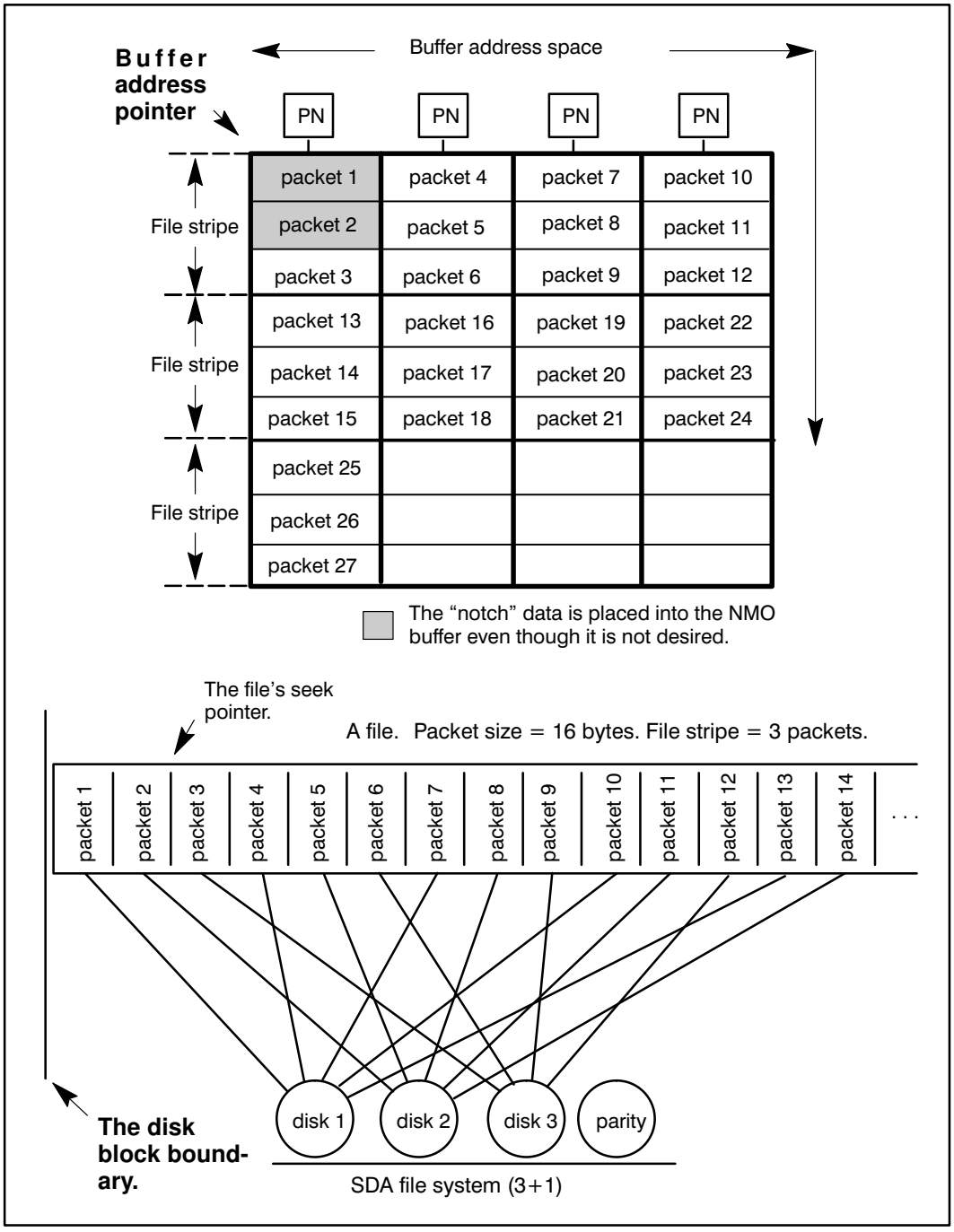


Figure 15. A notched NMO buffer (result of `CM_read(`*fd, pbuf*`, 400)`).

Copyright © 1994 Thinking Machines Corporation

B.2.2 The CM_write() System Call

To write from NMO buffer space to a file, execute **CM_write**:

```
int = CM_write(fd, pbuf, count)
```

During a **CM_write** system call, the NMO buffer is written to the specified file in the order shown in Figure 16: Starting at the buffer pointer, one disk stripe per PN is written, wrapping back to the first PN, until *count* bytes have been written.

Note that if the buffer pointer is not pointing to a byte that is *s* bytes from the beginning of the buffer, where *s* is 0 or a multiple of the file stripe, the intial file stripe will, in fact, be a partial stripe. And if *count* is not a multiple of the partition stripe, the final file stripe will, in fact, be a partial stripe. If **CM_write** must operate on partial stripes, a read-modify-write operation — which reads the balance of the file stripe into the NMO buffer — must be performed, for parity-calculation purposes. (Parity generation requires an entire stripe's data.) Each read-modify-write causes **CM_write** to take longer than it would otherwise.

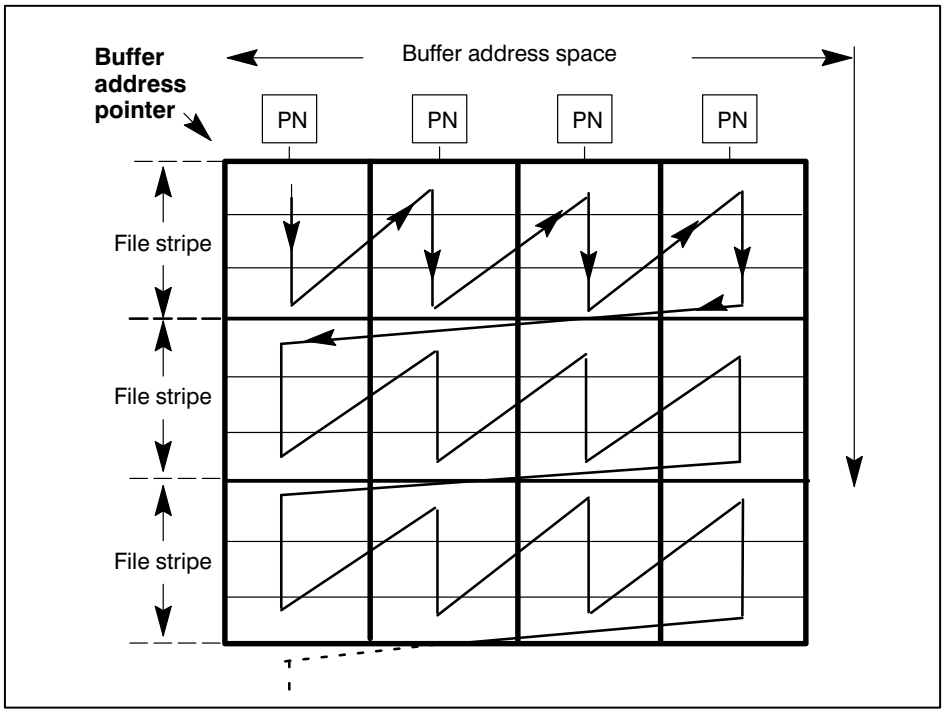


Figure 16. The order in which packets are written to a file.

Copyright © 1994 Thinking Machines Corporation

The NMO buffer must be able to accommodate read-modify-write operations; if there is not enough buffer space allocated to perform a required read-modify-write, **`CM_write`** will return **`EFAULT`**.

B.3 NMO on Vector Unit NMO Buffers

Each PN may be equipped with a high-performance arithmetic accelerator, which consists of four vector units (VUs), each with its own memory bank, as shown in Figure 17. On a PN so equipped, the NMO buffer space encompasses the memory of all four vector units.

As such, although the principle is the same, a VU-equipped NMO buffer fills a bit differently than it does on a system without VUs. On a VU-equipped system, the NMO buffer space of the first PN is filled by placing one packet per VU, wrapping back to the first VU, until an entire disk stripe (or part of one, when there is not an entire disk stripe left), is placed. Subsequent PNs are filled in the same manner. The process continues, wrapping back to the first PN, until *count* bytes are read.

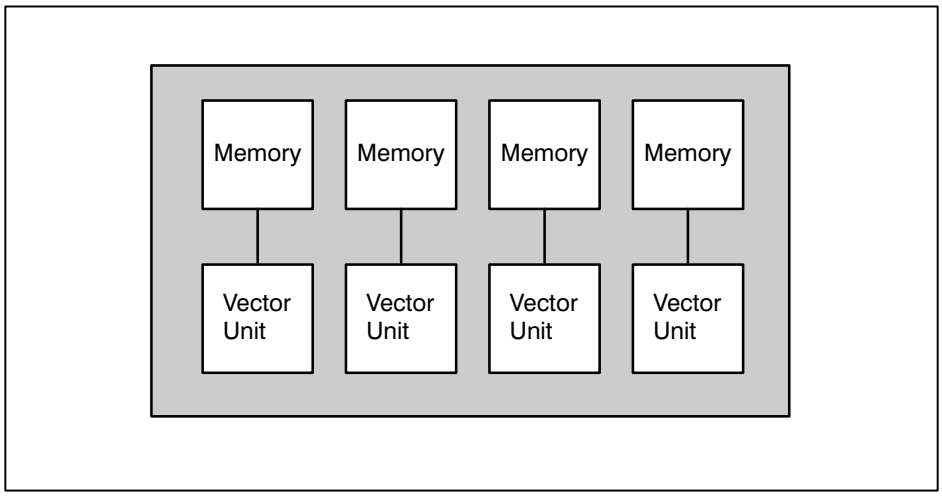


Figure 17. The high-performance arithmetic accelerator hardware in a PN. For more information about the arithmetic accelerator and the vector units themselves, please refer to the *VU Programmer's Handbook* or the *Connection Machine CM-5 Technical Summary.*

Copyright © 1994 Thinking Machines Corporation

Figure 18 shows what the NMO buffer space of a 4-PN system looks like after a call to **`CM_read`**`(`*fd, pbuf*, 384`)`. Compare Figure 18 with Figure 12, which shows a non-VU-equipped NMO buffer filled by the same system call.

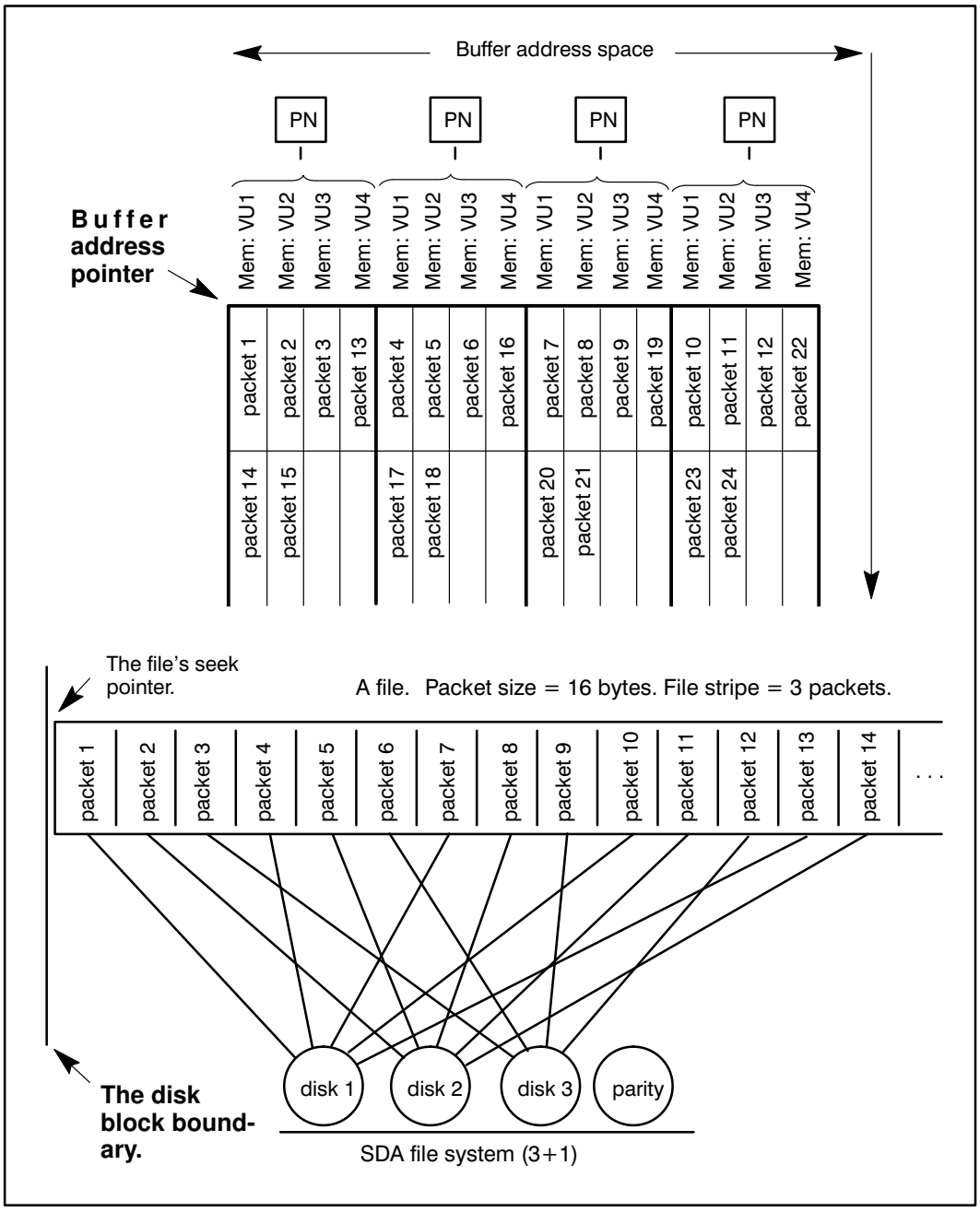


Figure 18. NMO buffer layout for a `CM_read`(*fd*, *pbuf*, `384`) to a 4-PN system with high-performance arithmetic accelerator hardware.

Copyright © 1994 Thinking Machines Corporation