Customer
Engineering
Conference
2003

February 28 - March 3

Colorado Convention Center
Denver, Colorado

Where's My Data

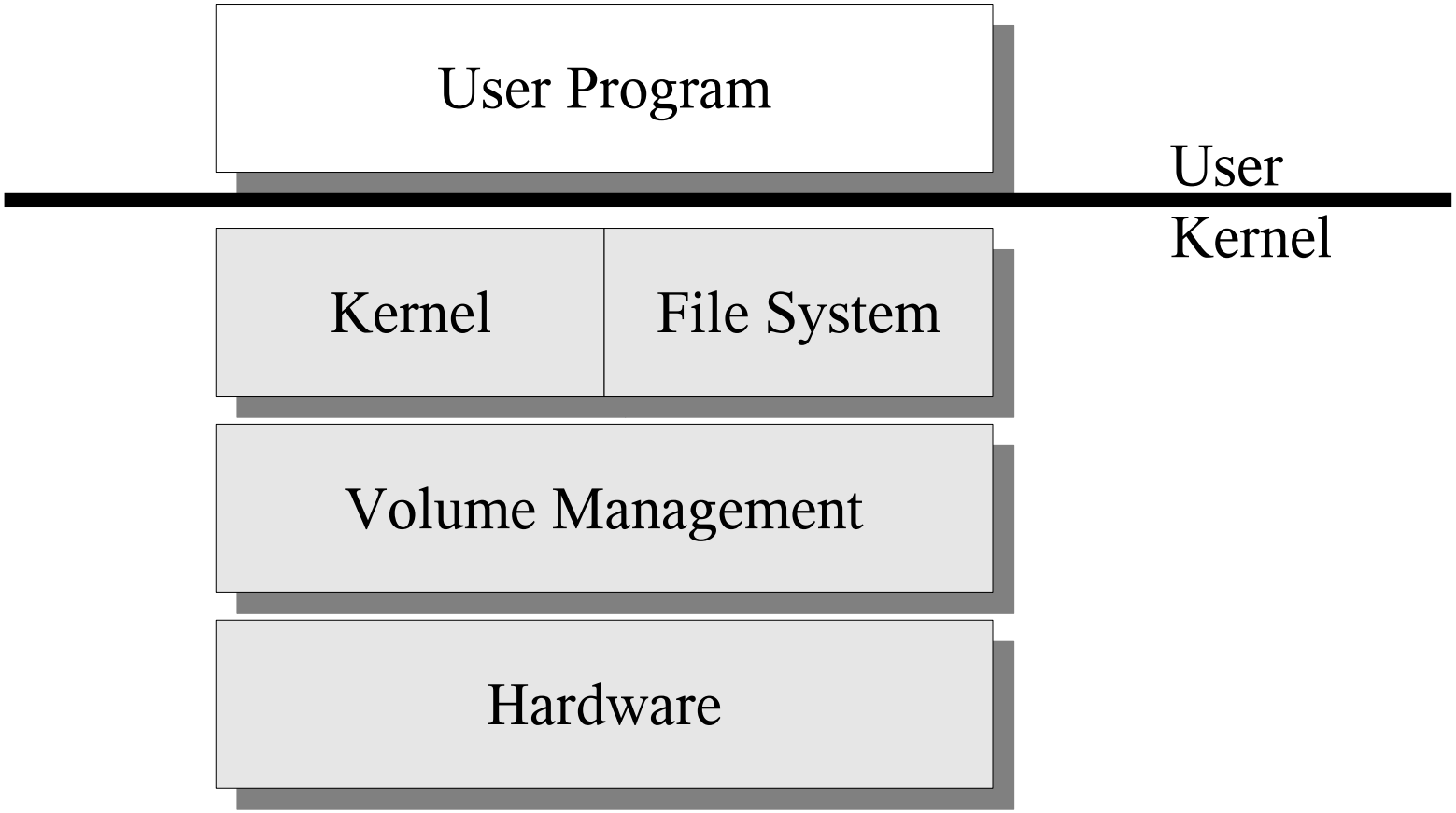
Where's My Data?

- Your data is either
  - In Memory (ie in your program)
  - In the page cache (ie in the kernel)
  - On Disk
  - Oh dear you didn't mirror it, then it's on your backup tape

How did it get there?

- Your program
- The File System (UFS)
- SVM and its brothers and sisters
- The disk drivers and hardware stuff
- The disk
- What about return codes?

Where's My Data?



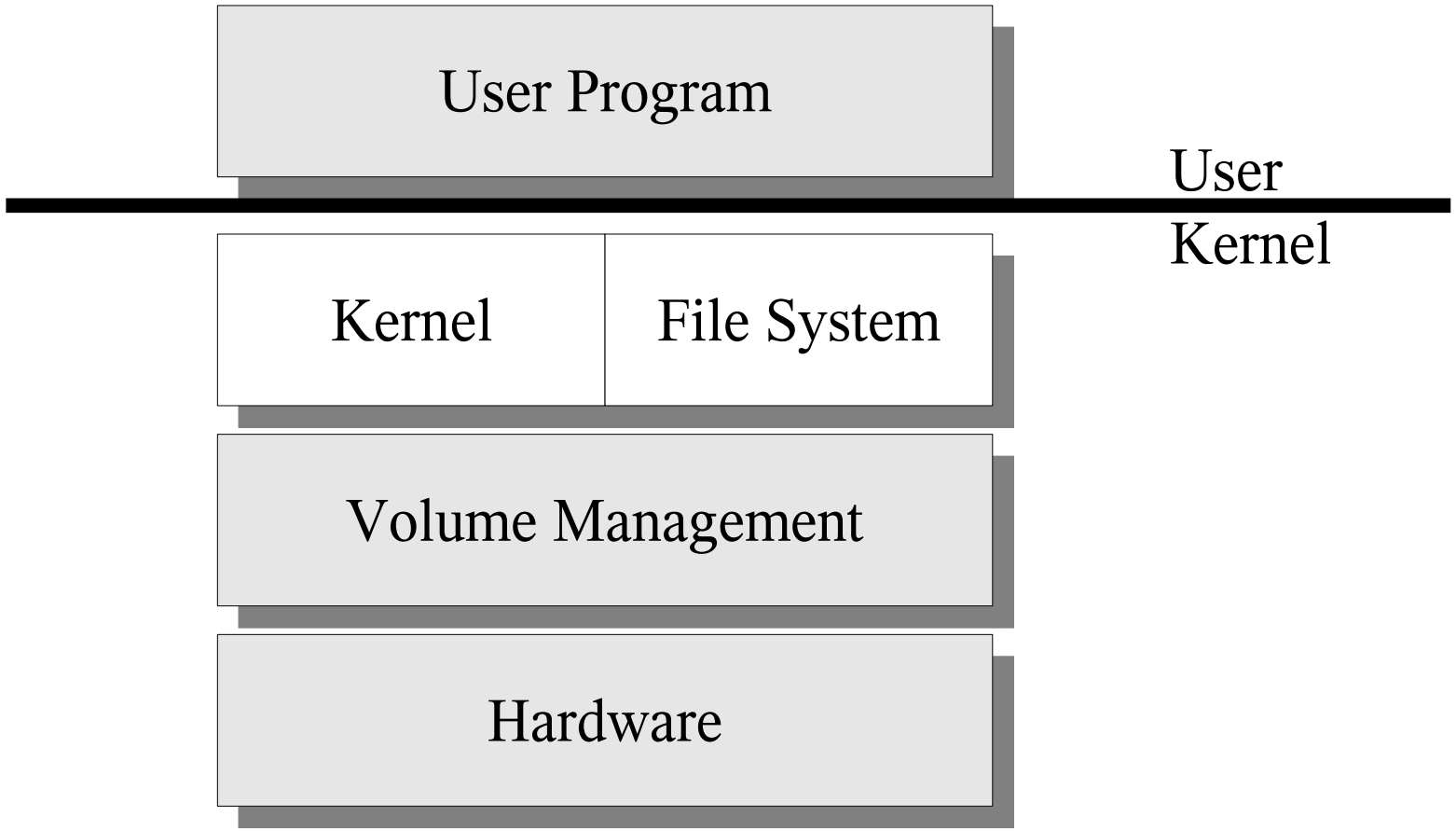
Your Program

- In its simplest form your program will
  - open() a file
  - read() or write() to the file
  - close() the file
- We'll concentrate on write()
  - read() is just the same in reverse
  - close() is trivial
  - open() is a little more complicated

Your Program

```
main()
{
        int fd;
        char[] mydata=”IMPORTANT”;
        fd = open(“mydatafile”, O_CREAT);
        nbytes = write(fd, mydata);
        if (nbytes != sizeof(mydata)){
             fprintf(stderr, “Oh dear didn't write the right number
   of bytes”);
             exit (1);
        }
        close (fd);
        exit (0);
}
```

Where's My Data?



The Filesystem

- Why have a filesystem?
- It provides a translation between the files and directories we like to use to manage the layout of blocks on a disk

The Filesystem

- Some important concepts/structures
  - The Inode
    - Describes the file including how it's laid out, when it was updated what device it was on
    - Struct inode
    - usr/src/uts/common/sys/fs/ufs_inode.h
  - The Vnode
    - In core version of the inode
    - Abstracts the file (makes the information fs independent)
    - Describes what operations you can perform on it
    - Struct vnode
    - usr/src/uts/common/sys/vnode.h

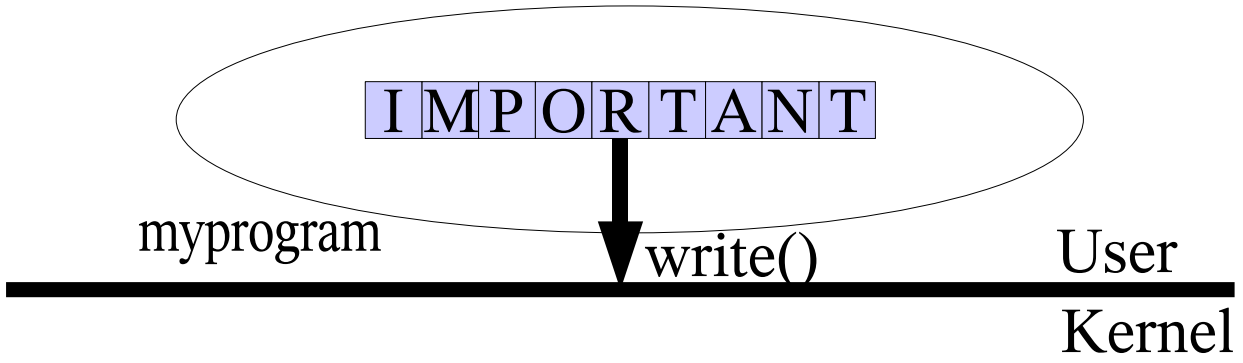
The Kernel

- Some more important concepts/structures
  - The buf
    - The basic unit of data by which I/Os are communicated
    - Device major and minor numbers
      - Used to distinguish which driver and target the buf is bound for
    - offset
    - iodone() function
    - Struct buf
    - usr/src/uts/common/sys/buf.h
    - Manipulated by the b* functions (e.g. bwrite_common())
    - usr/src/uts/common/os/bio.c

open()

- Checks the file permissions
- Allocates a file structure (falloc())
- Finds the vnode
  - (eventually through lookuppnvp())
  - Either by reading the dnlc or reading the directory
- Calls the specific open function
  - On ufs it's a NOP
- Populates the file structure

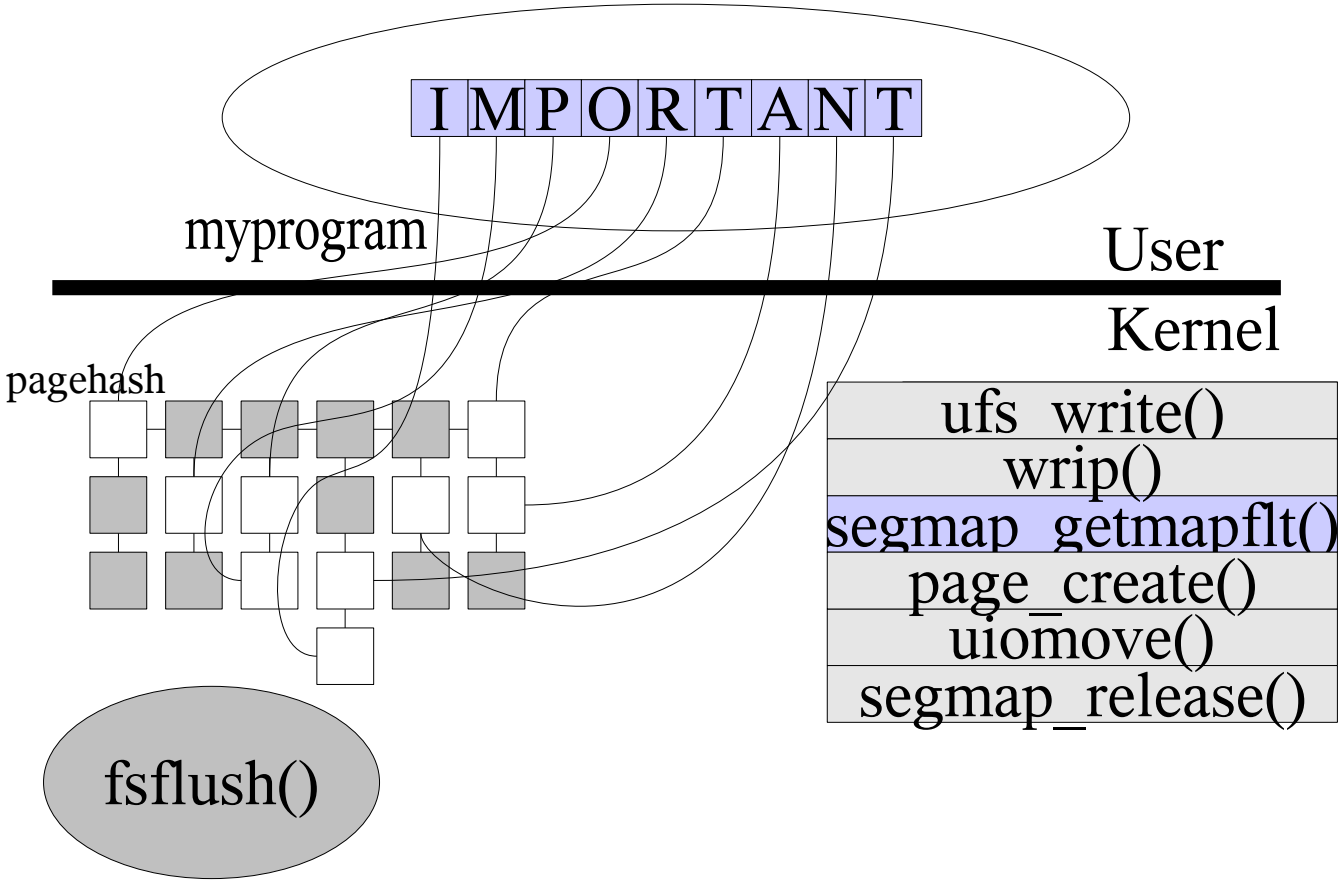
write()



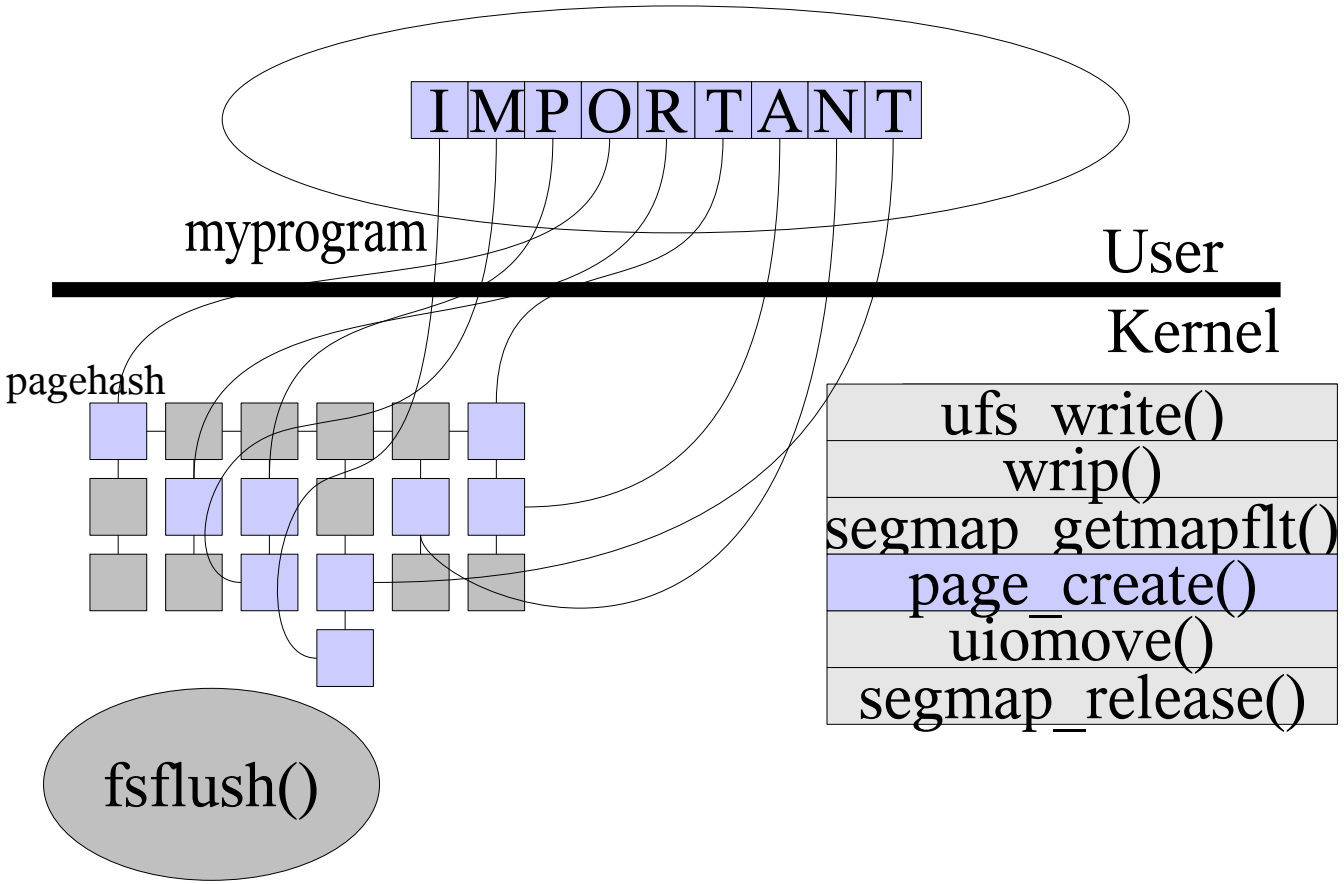
ufs_write()
wrip()
segmap_getmapflt()
page_create()
uiomove()
segmap_release()

fsflush()

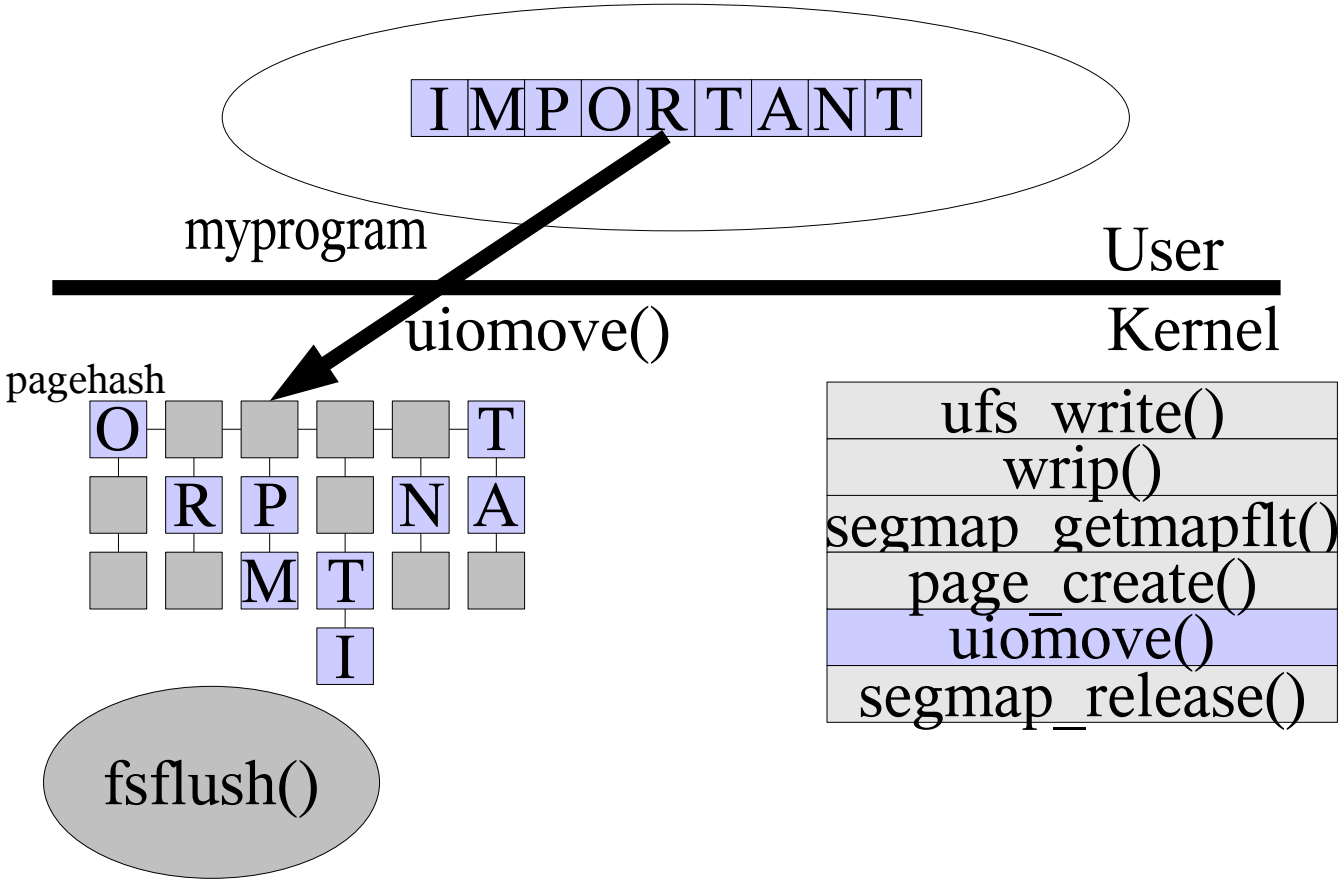
write()



write()



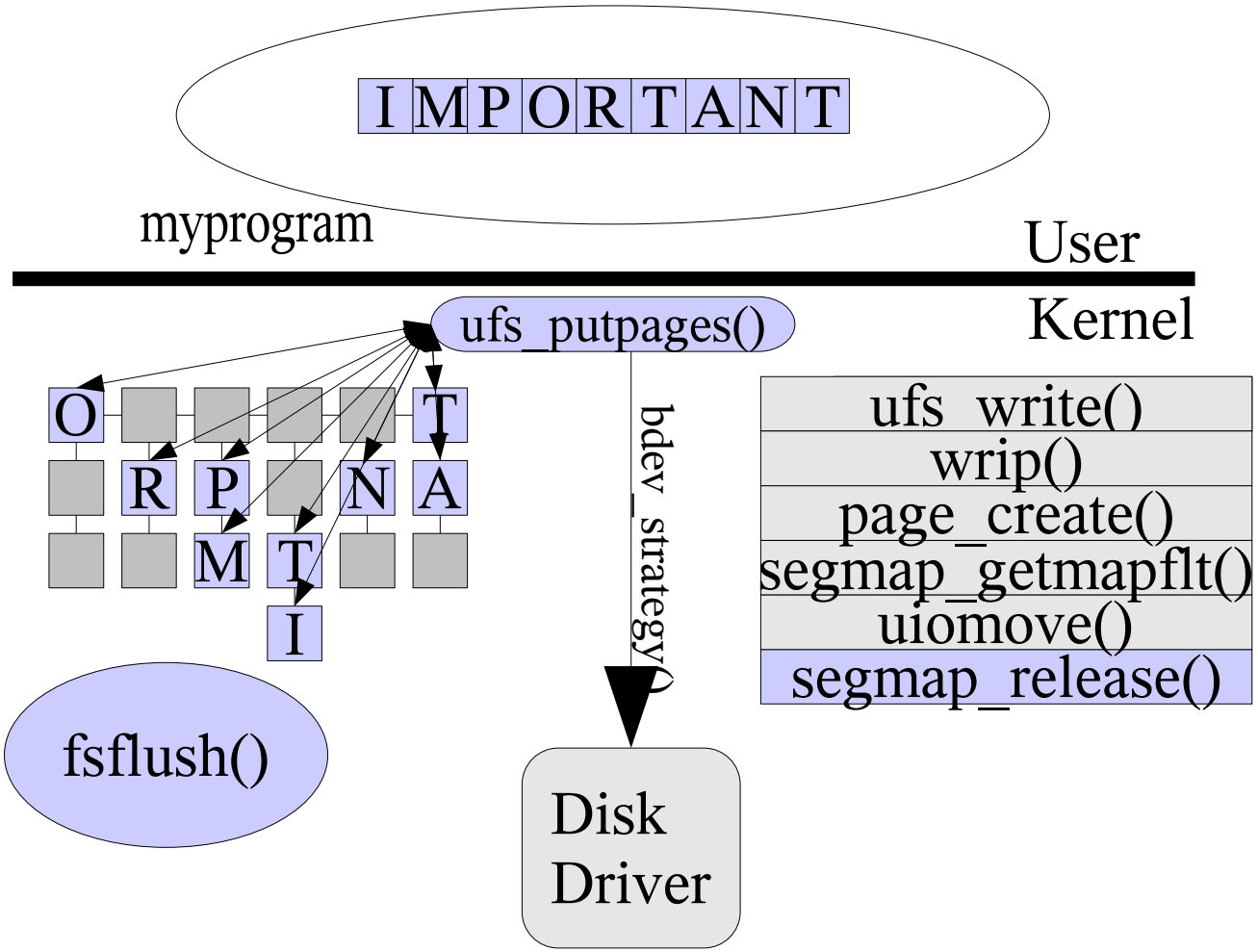
write()



The Filesystem

- Where's my data now?
  - The uiomove() call copied the data from the userland pages to the kernel pages set up by the segmap_getmapflt()
  - But how does it get to disk?

write()

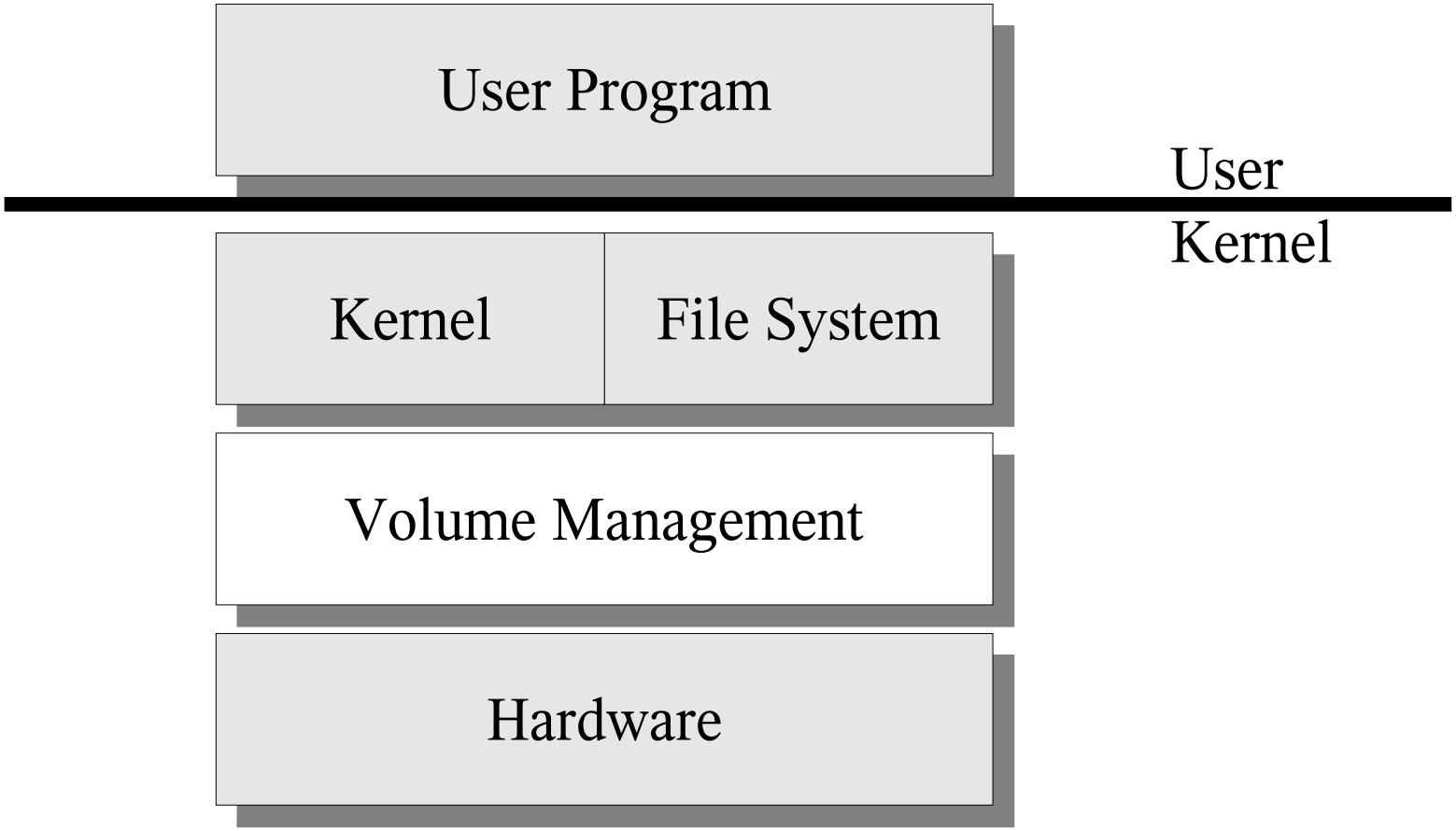


The Filesystem

- When you do a write() what happens?

➔write()->VOP_WRITE() -> ufs_write()
  - ➔ wrip()
    - ➔ Calculates offsets to write to
    - ➔ segmap_getmapflt()
    - ➔ Creates any needed pages
    - ➔ uiomove()
    - ➔ segmap_release()
      - ➔ VOP_PUTPAGE() ->ufs_putapage()
        - ➔ bdev_strategy()
        - ➔ biowait()

Where's My Data?



Volume Management Layer

Why Have a Volume Manager ?

- Availability
  - Provide RAID protection to our data
- Performance
  - Spread data over many spindles
- Disk Management
  - Split very large LUN's into smaller sizes
  - Join lots of small spindles into bigger sizes

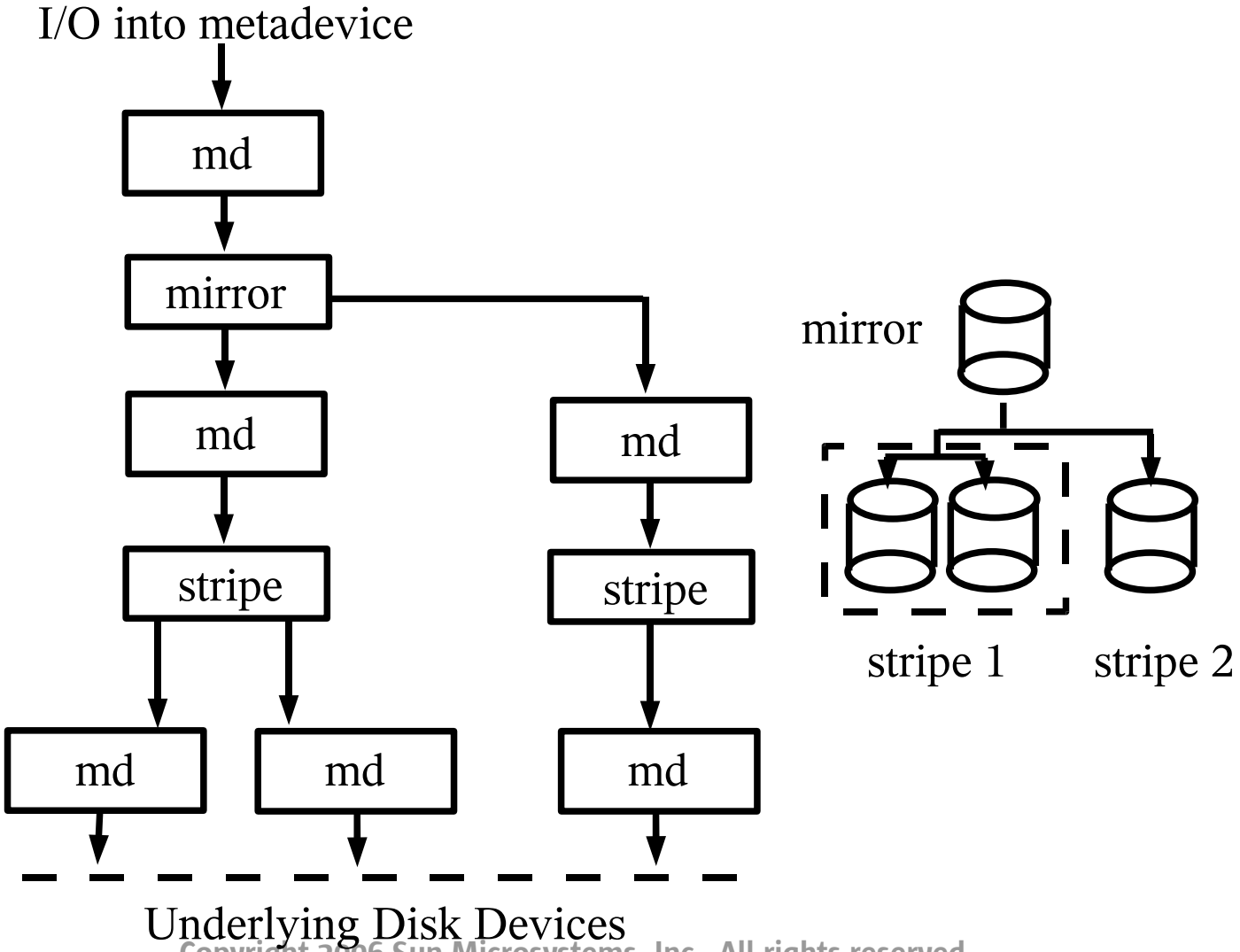
Volume Management Layer

Solaris Volume Manager Architecture

- Modular design
  - md – core driver that directs I/O to lower layers
  - Plugin driver modules for :
    - Mirroring – md_mirror
    - RAID 5 – md_raid
    - Stripes – md_stripe
    - Soft Partitions – md_sp
    - Hot Spares – md_hotspares

Volume Management Layer

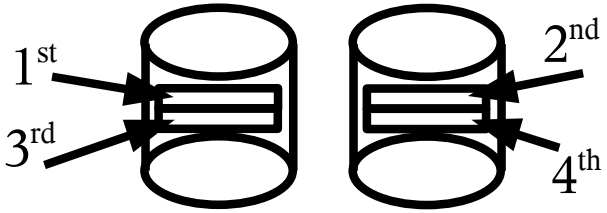
Solaris Volume Manager Architecture



Volume Management Layer

Example – writing to a stripe

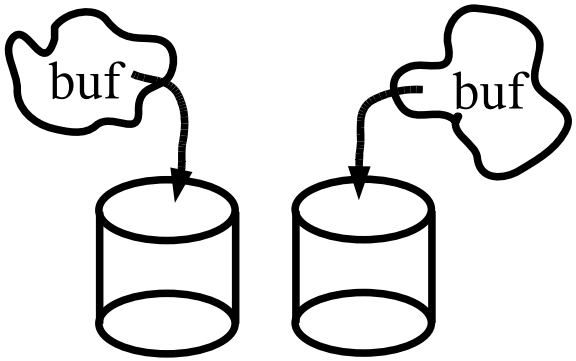
- Strategy routine gets called
  - stripe_strategy()
- Receives buf from the mirror layer
- Looks up underlying devices
  - Major / minor number from metadevice data
  - Accounts for interlace factor



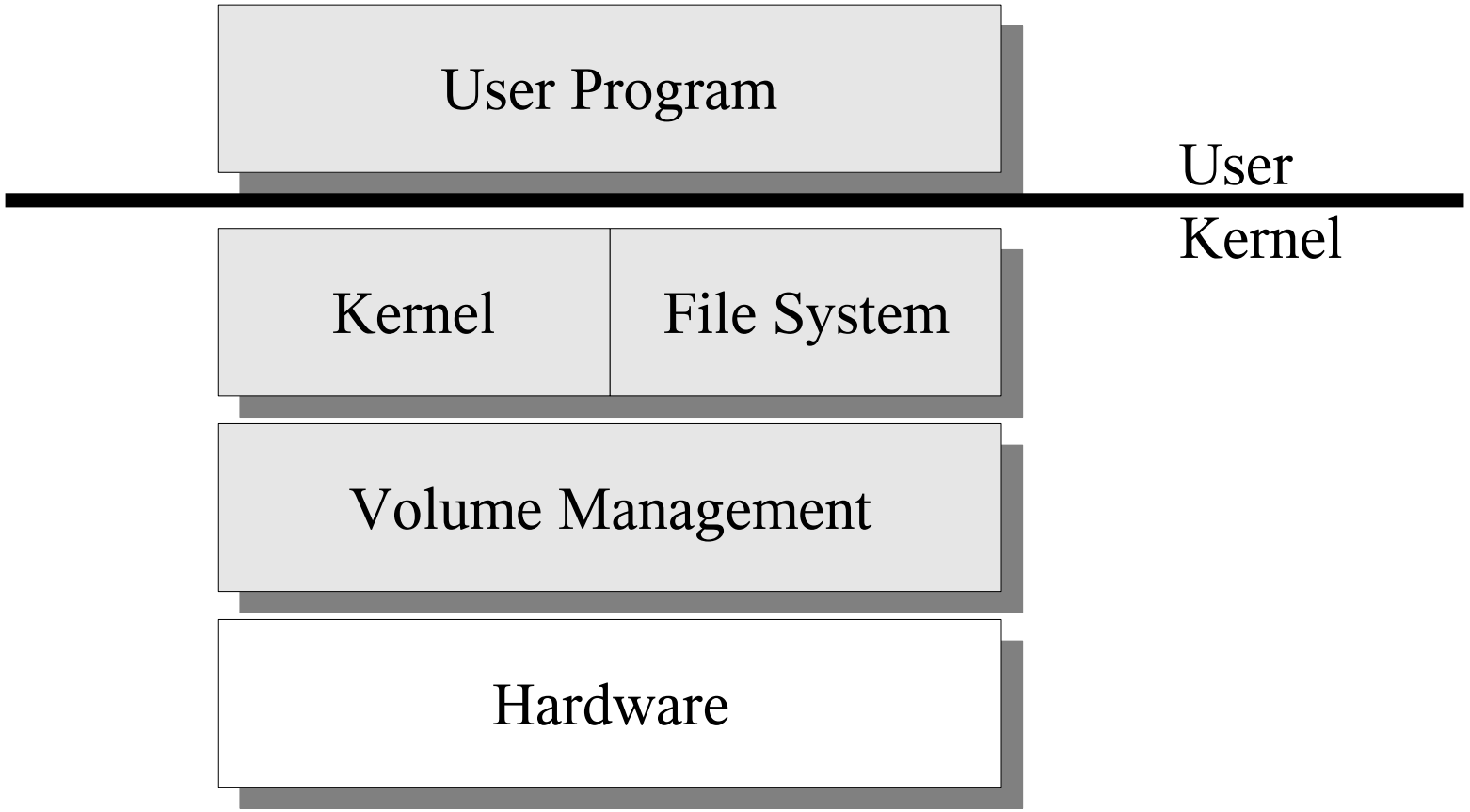
Volume Management Layer

Example – writing to a stripe (cont.)

- Copy the parent buf into new child buf's
  - Single buf created for each underlying device
  - done using md_bioclone()
- Call md_strategy() to progress further
  - Passes buf's on to the target driver layer



Where's My Data?



Target Driver Layer

- Why have a target driver ?
  - Translate the write into the correct protocol
    - SCSI e.g. “sd”
    - FC-AL e.g. “ssd”
    - IDE / ATAPI e.g. “dad”
    - IPI e.g. “id”
    - Other ...
  - Must convert the address
    - From major / minor / offset
    - To bus / target / LUN / block

Some Concepts

- Sun Common SCSI Architecture - SCSA
  - Device independent interface between target driver and host adaptor
- SCSI Command Descriptor Block - CDB
  - Specifies command, LUN, length etc.
  - 6, 10, 12 or 16 bytes in length
- SCSI Protocol
  - Host Adaptor selects target device & sends CDB
  - Target device performs command
  - Target device tells Host Adaptor when finished
  - All performed by HBA & Target firmware

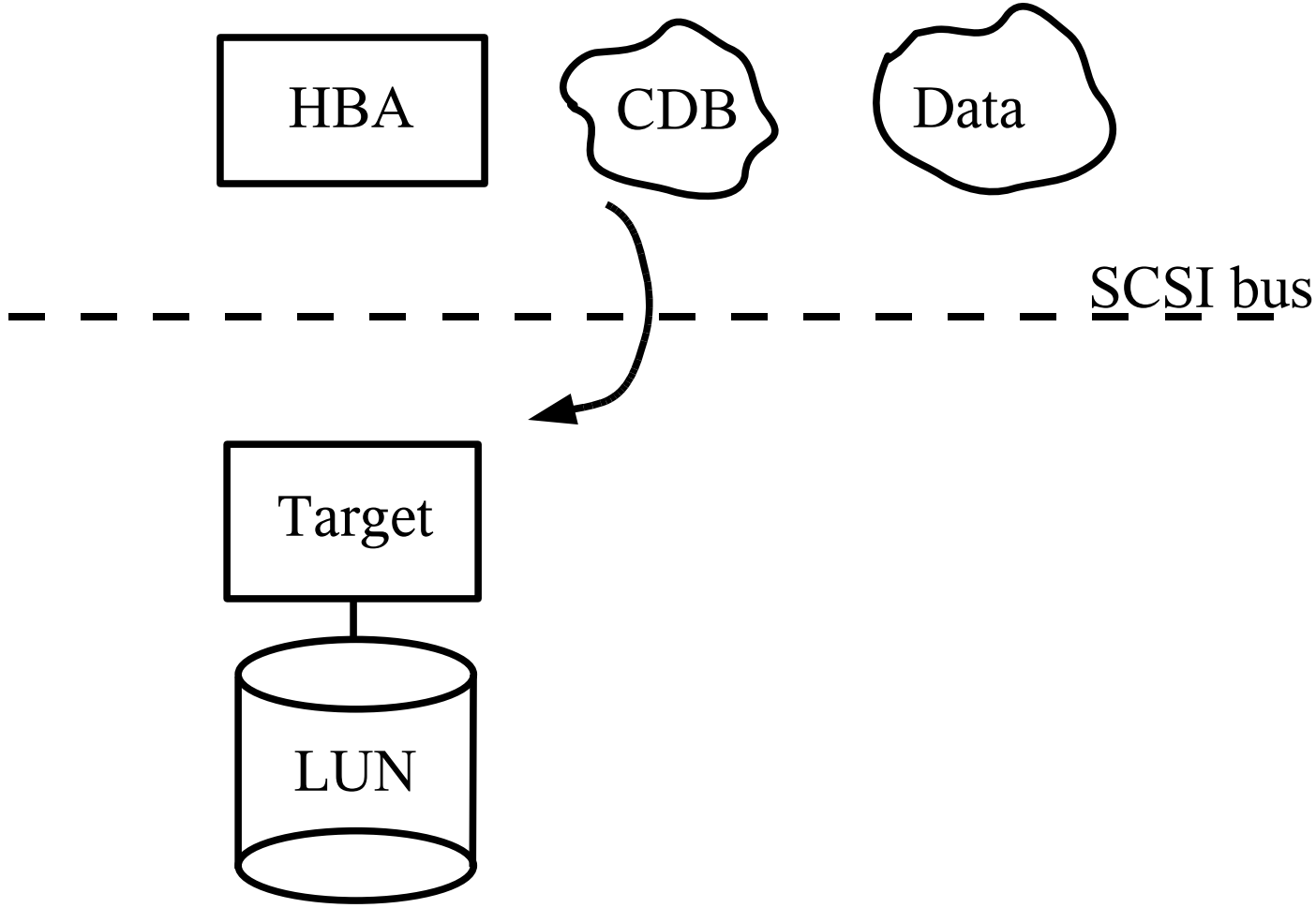
Target Driver Preparations

- sdstrategy() called for write operation
  - Basic checks performed
    - Does the device exist ?
    - Does it have valid geometry?
    - Is it a CDROM ?
  - Builds scsi_pkt structure
  - Populates CDB in scsi_pkt
  - Sets up timeout
  - Provides callback routine for completion
- Data still in our kernel buffer

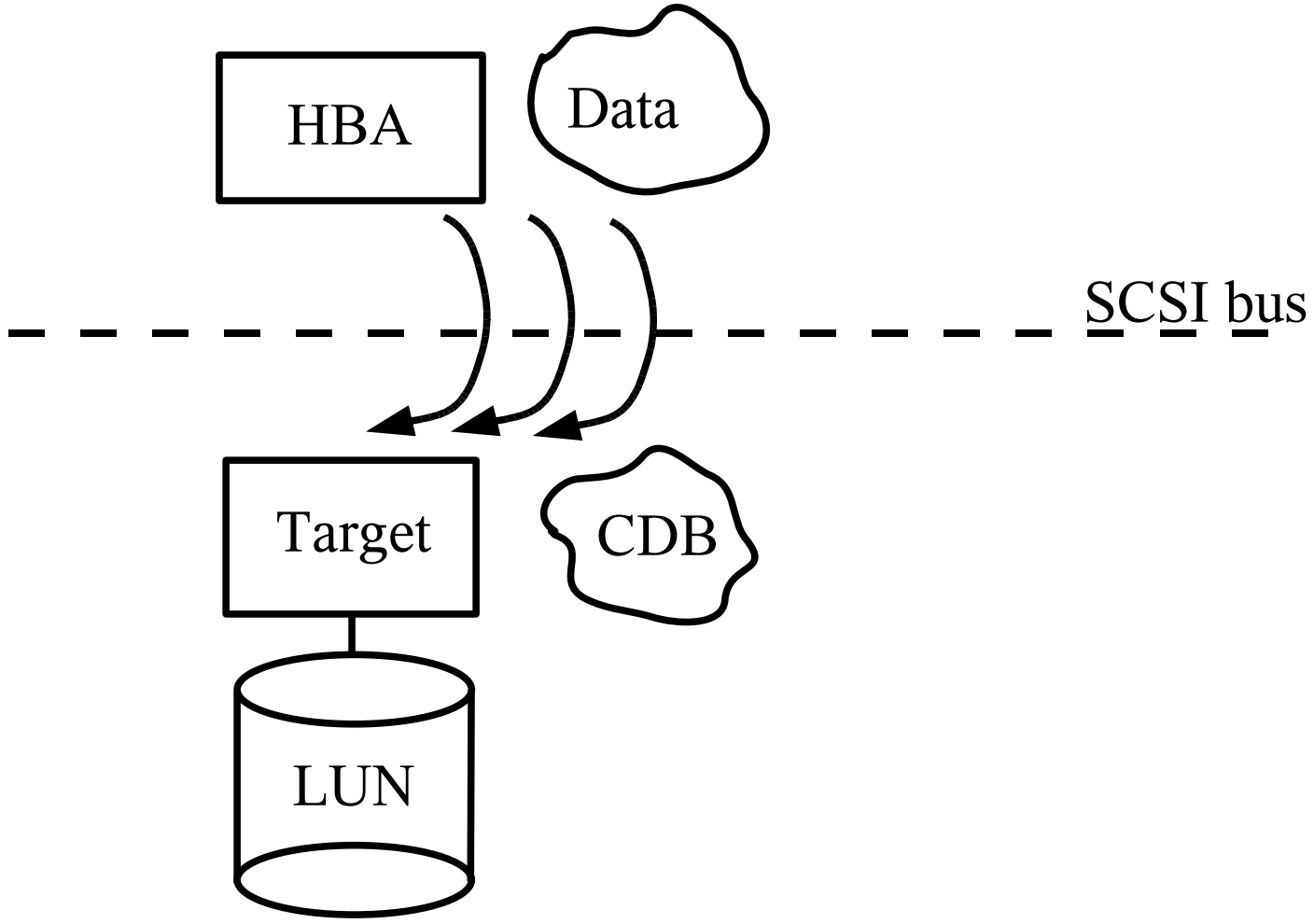
Host Bus Adaptor Layer

- Target Driver passes scsi_pkt
  - Passes in the prepared CDB
  - Receives back acknowledgment
- Queues command to SCSI BUS
  - Multiple commands in queue at once
  - Per target & per LUN queues
- Handles Interrupts
- Manages Timeouts

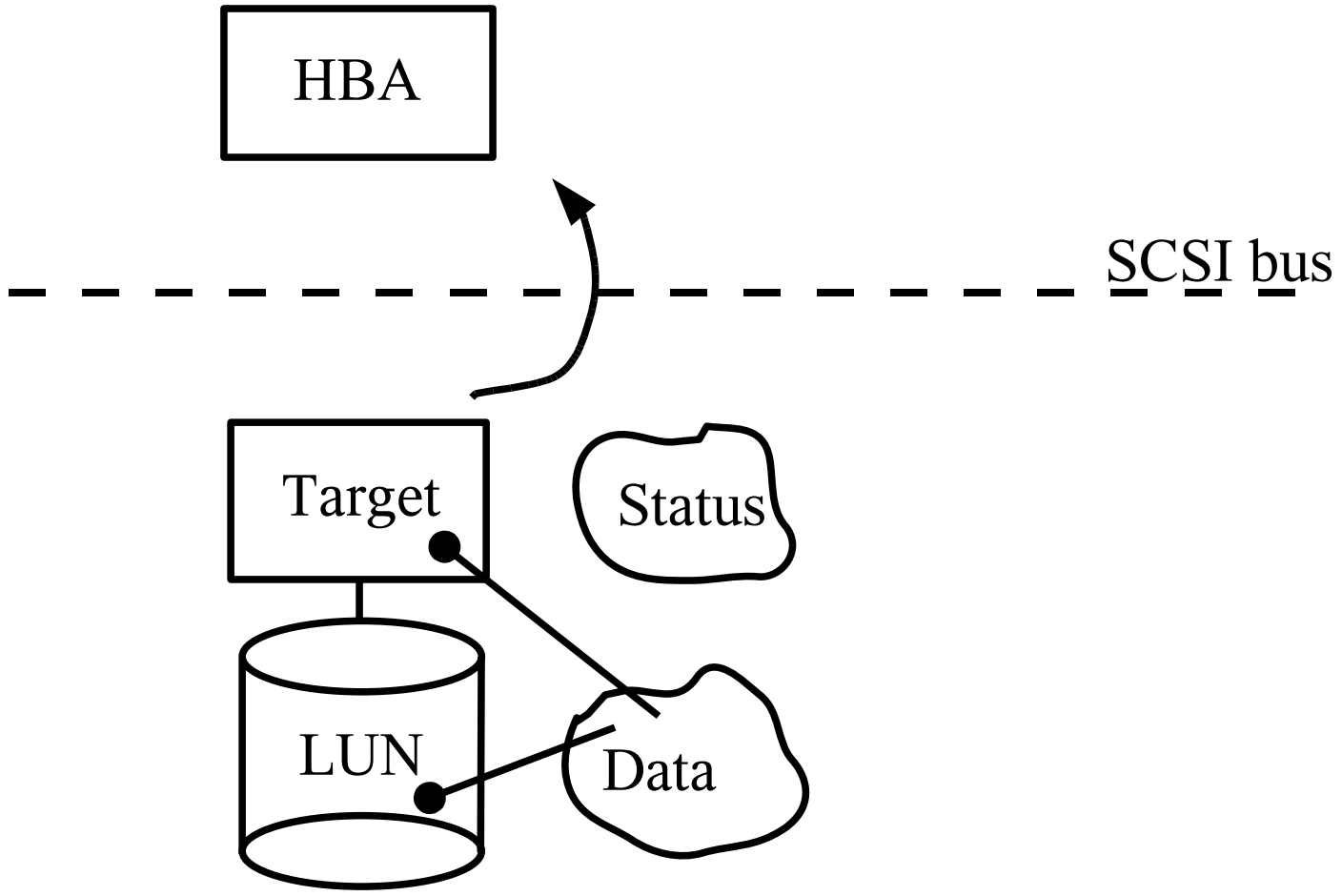
SCSI Transfer - Part 1



SCSI Transfer - Part 2



SCSI Transfer - Part 3



The Target Disk Drive

- Manages I/O operations itself
- May queue & reorder requests
- May cache reads & writes
- Controls SCSI bus protocol
- May translate geometry
- Multiple LUNs per target
- You're talking to the controller

Return Status

- Disk drive signals HBA Chipset
- HBA Chipset interrupts HBA Driver
- HBA Driver calls Target Driver callback routine
- Target Driver calls biodone()
- Stripe layer calls biodone() on the parent buf
- Mirror layer calls biodone() on its parent buf
- MD calls biodone() on the buf it was passed
  - So ufs_iodone() gets called
  - biowait() from ufs_putpage completes
- The write() system call returns

Further Information

- ANSI SCSI Specifications
- Writing Device Drivers, 805-7378-10

scsi_pkt structure