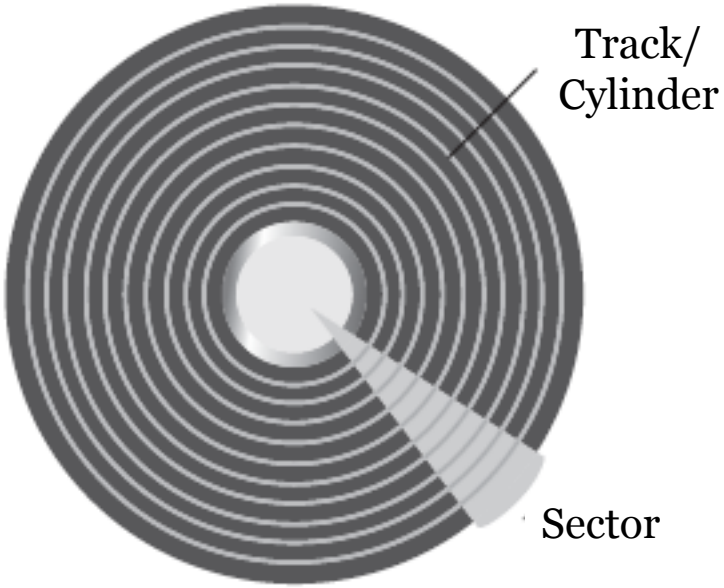
Block Device Programming

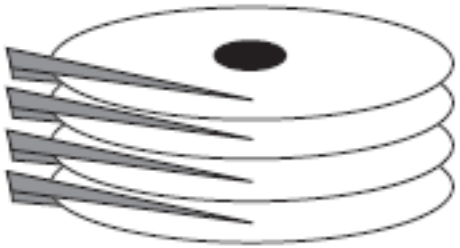
Agenda

- About Hard Disk
- Block Device Structure
- Data Structure about I/O
- Block Device Driver Programming

Hard Disk Structure/Performance

- Block device vs. Character device
  - Random Access vs. Stream
- Hard disk
  - 1Ghz CPU : 1 inst./10^{-9}sec
  - Total Disk Access Time = seek time + rotation time + data transfer time
  - Seek time = 8 ms (10^{-3}sec)
  - Rotation time = 8.3ms/cycle (7200 RPM)
  - Data transfer time : 5.1us/512B, 10.4ms/1MB (for 100Mbyte/s)

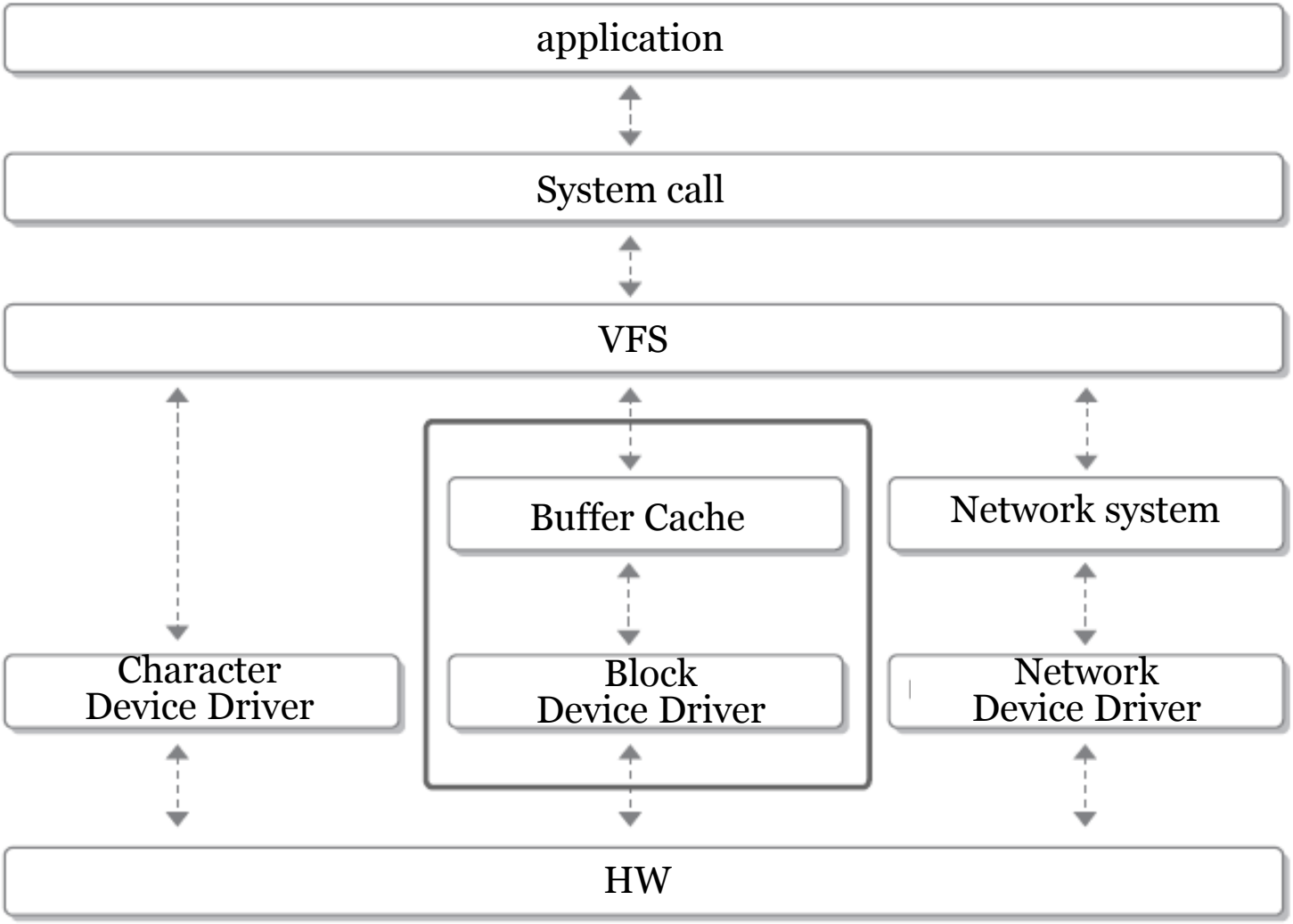




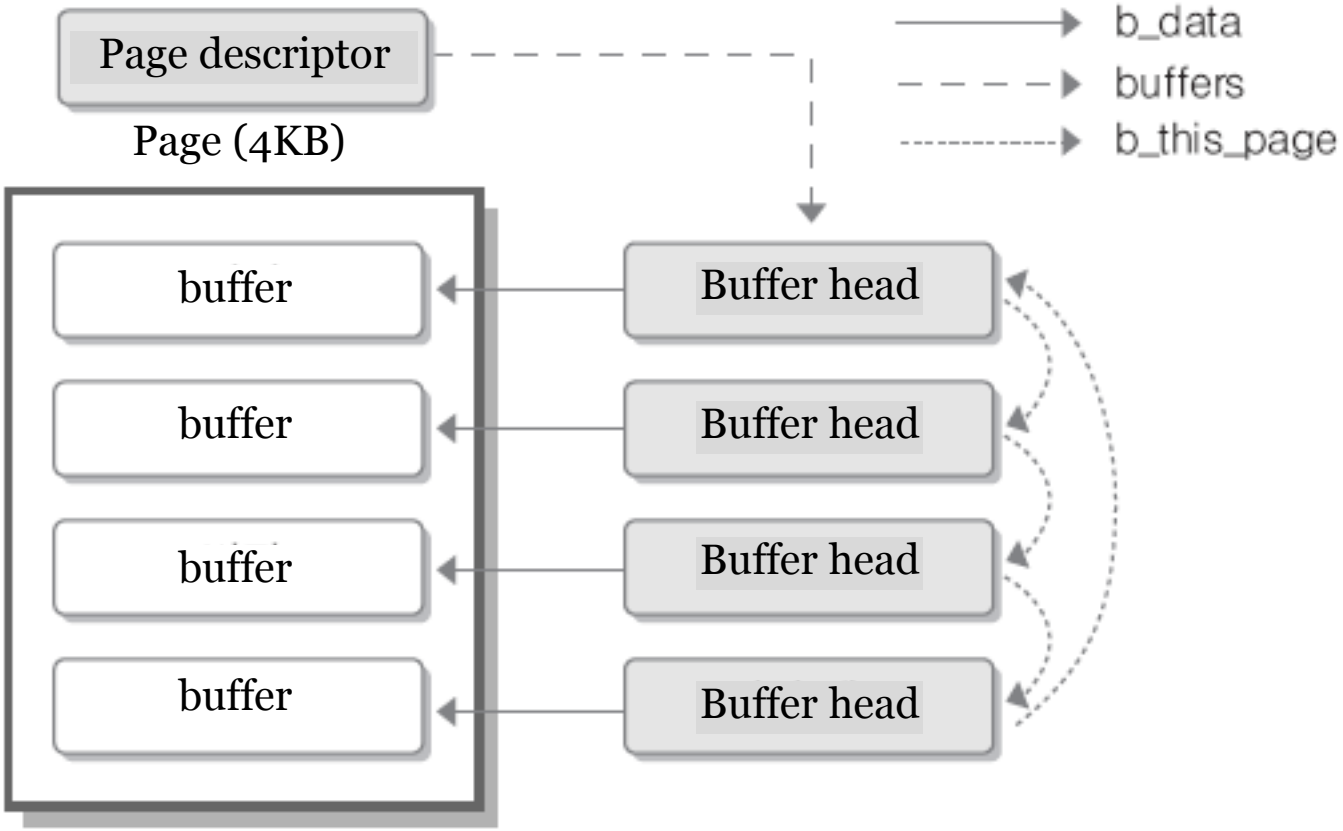
8 Heads
4 Platters

http://en.wikipedia.org/wiki/Cylinder-head-sector』

Block Device Structure



Buffer Head(Kernel 2.4)



buffer_head (kernel 2.6)

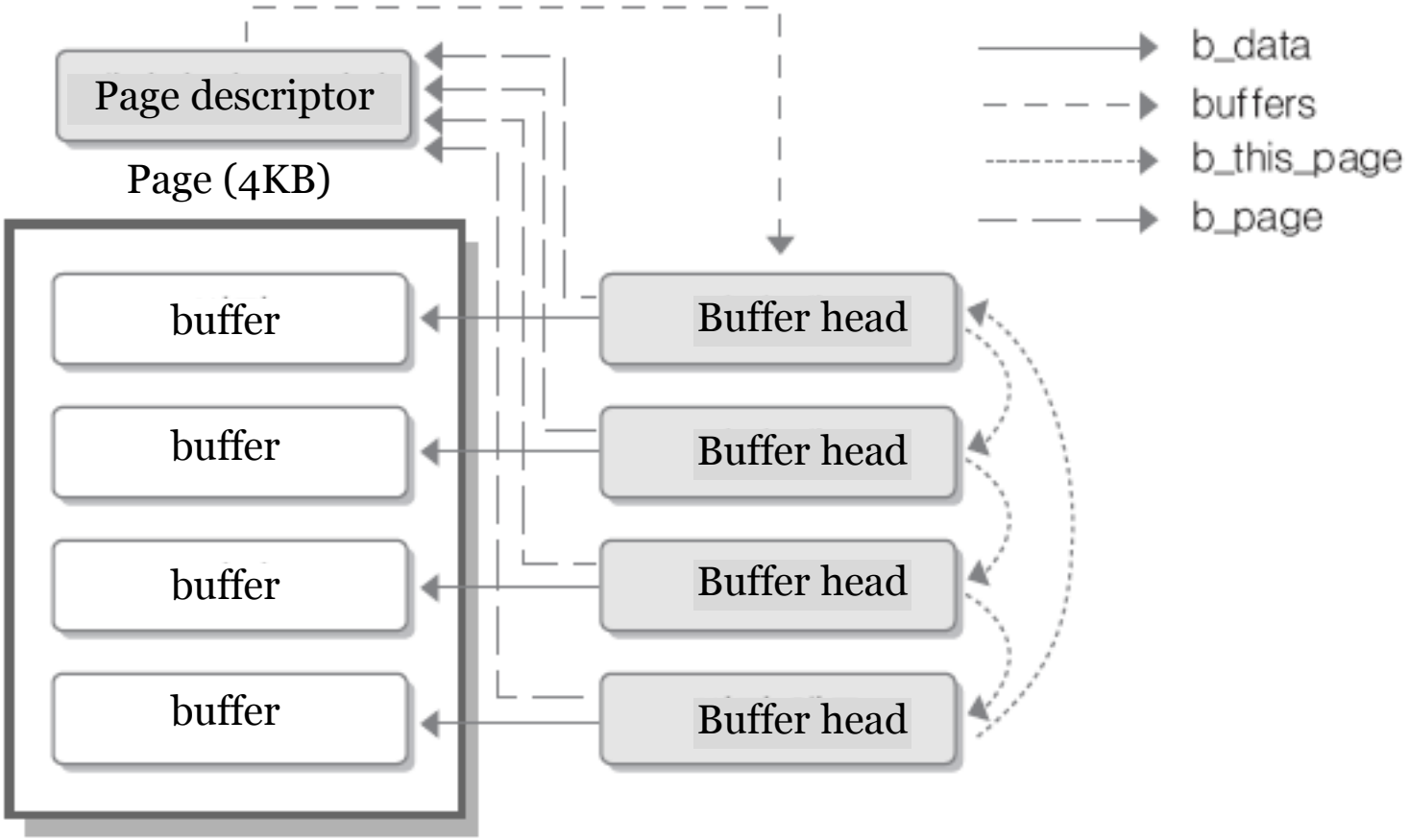
```
struct buffer_head
  unsigned long b_state;
  struct buffer_head *b_this_page;
  struct page *b_page;
  atomic_t b_count;
  u32 b_size;

  sector_t b_blocknr;
  char *b_data;

  struct block_device *b_bdev;
  bh_end_io_t *b_end_io;
  void *b_private;
  struct list_head b_assoc_buffers;

};
```

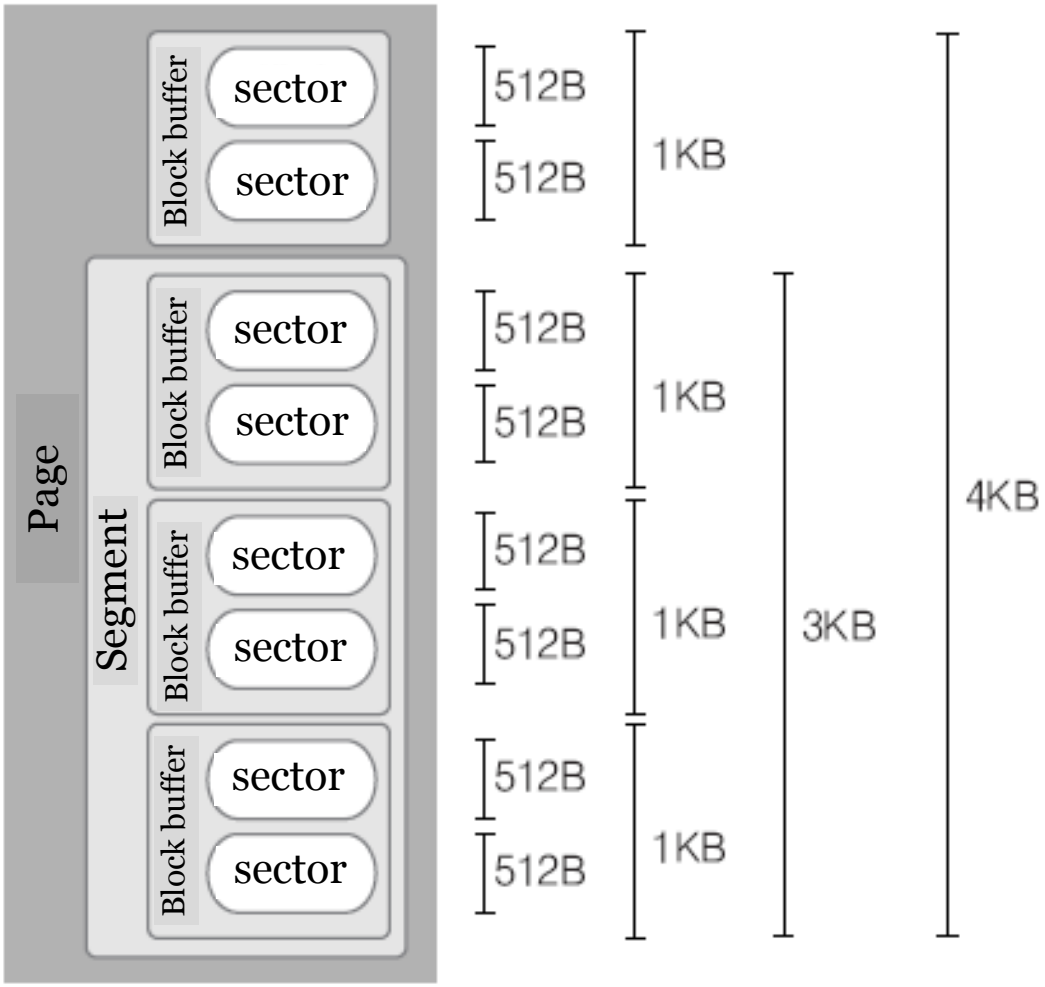
Buffer Head(Kernel 2.6)



Buffer head stats

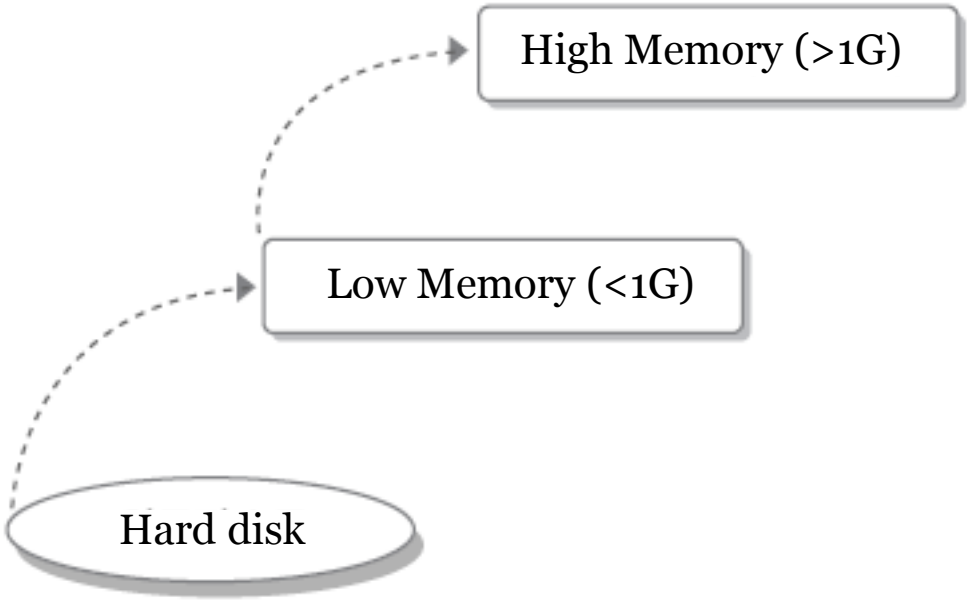
```
enum bh_state_bits {
  BH_Uptodate,      /* 블록과 버퍼 헤드가 동일한 데이터인 경우 */
  BH_Dirty,         /* 블록과 버퍼 헤드가 동일하지 않은 경우 */
  BH_Lock,          /* 프로세스가 버퍼를 사용 중이어서 잠겨있는 경우 */
  BH_Req,           /* I/O 요청을 보낸 경우 */
  BH_Uptodate_Lock, /* 페이지의 첫 번째 버퍼 헤드로 사용된 경우
                       페이지에 다른 버퍼들의 I/O 완료를 저장하기 위한 용도*/
  BH_Mapped,        /* 버퍼와 디스크가 매핑되어 있음 */
  BH_New,           /* get_block( )으로 새로 생성한 디스크 매핑 */
  BH_Async_Read,    /* end_buffer_async_read I/O를 수행중인 경우 */
  BH_Async_Write,   /* end_buffer_async_write I/O를 수행중인 경우 */
  BH_Delay,         /* 버퍼에 할당된 디스크 블록이 없는 경우 */
  BH_Boundary,      /* 비연속적인 블록 다음에 경계를 표시하기 위한 용도 */
  BH_Write_EIO,     /* 쓰기 오류 */
  BH_Ordered,       /* 정렬된 쓰기(ordered write) */
  BH_Eopnotsupp,    /* 지원하지 않는 연산(barrier) */
  BH_PrivateStart,  /* 사용하지 않음 */
};
```

Page, segment, and block

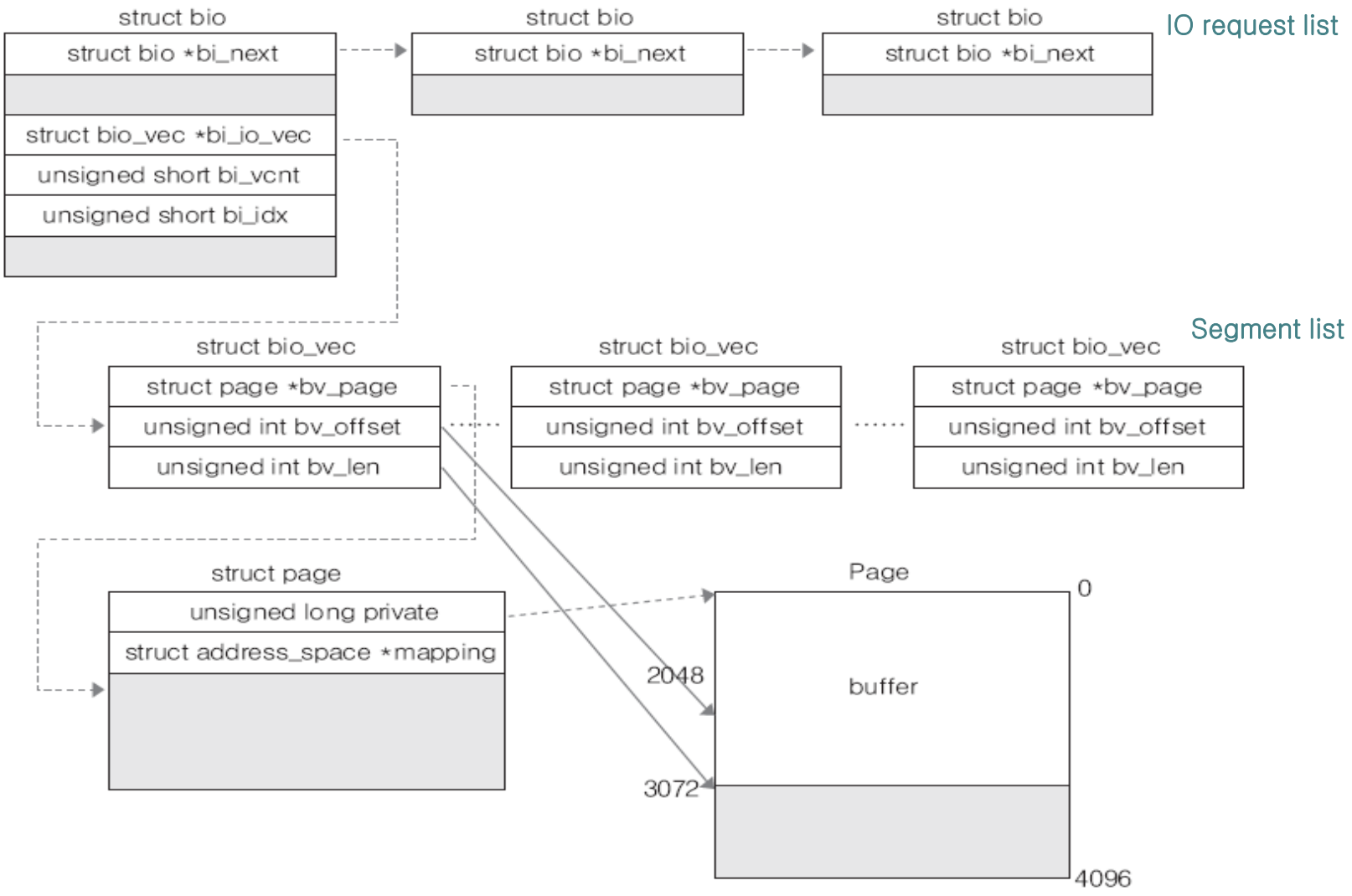


Problem in Buffer Head (Kernel 2.4)

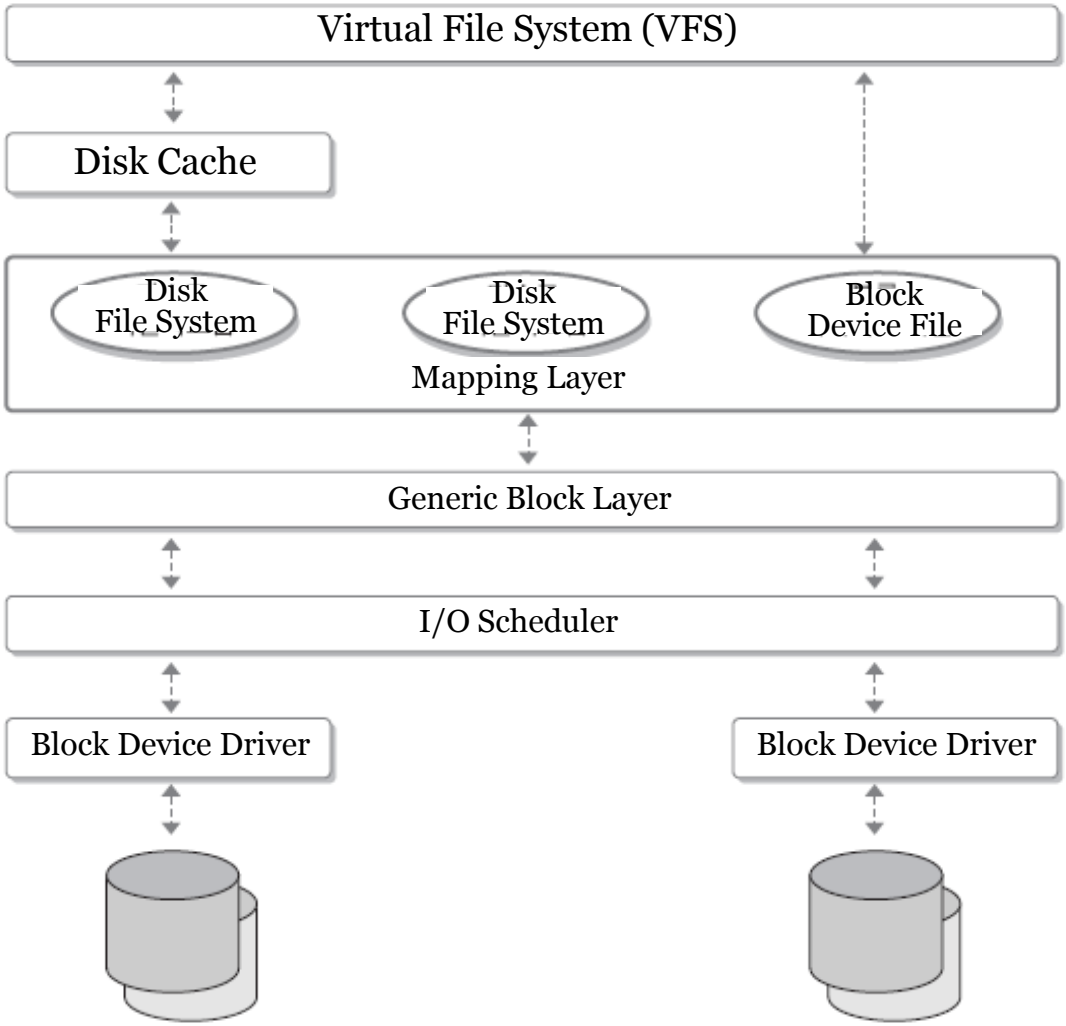
- Kernel handles page-base memory (4KB)
- Block device handles block-base data (512B)
- Thus, proper handling of such a mismatch is needed
- struct buffer_head is too large!
- buffer bounce



bio Structure

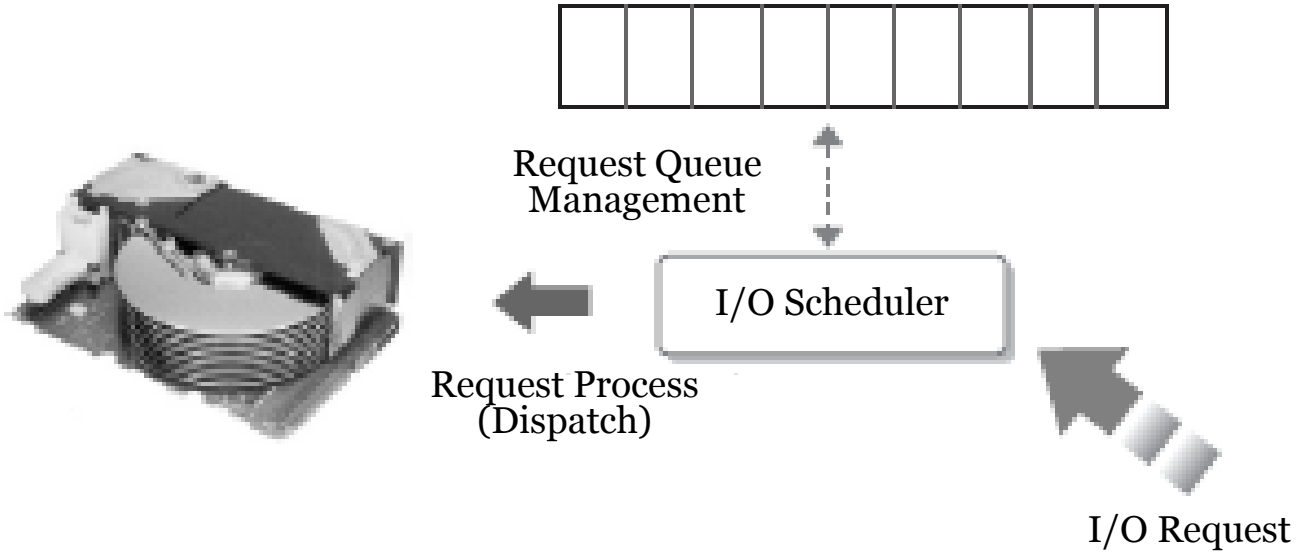


I/O Scheduler



『출처: Understanding the Linux Kernel 3rd, Figure 14-1』

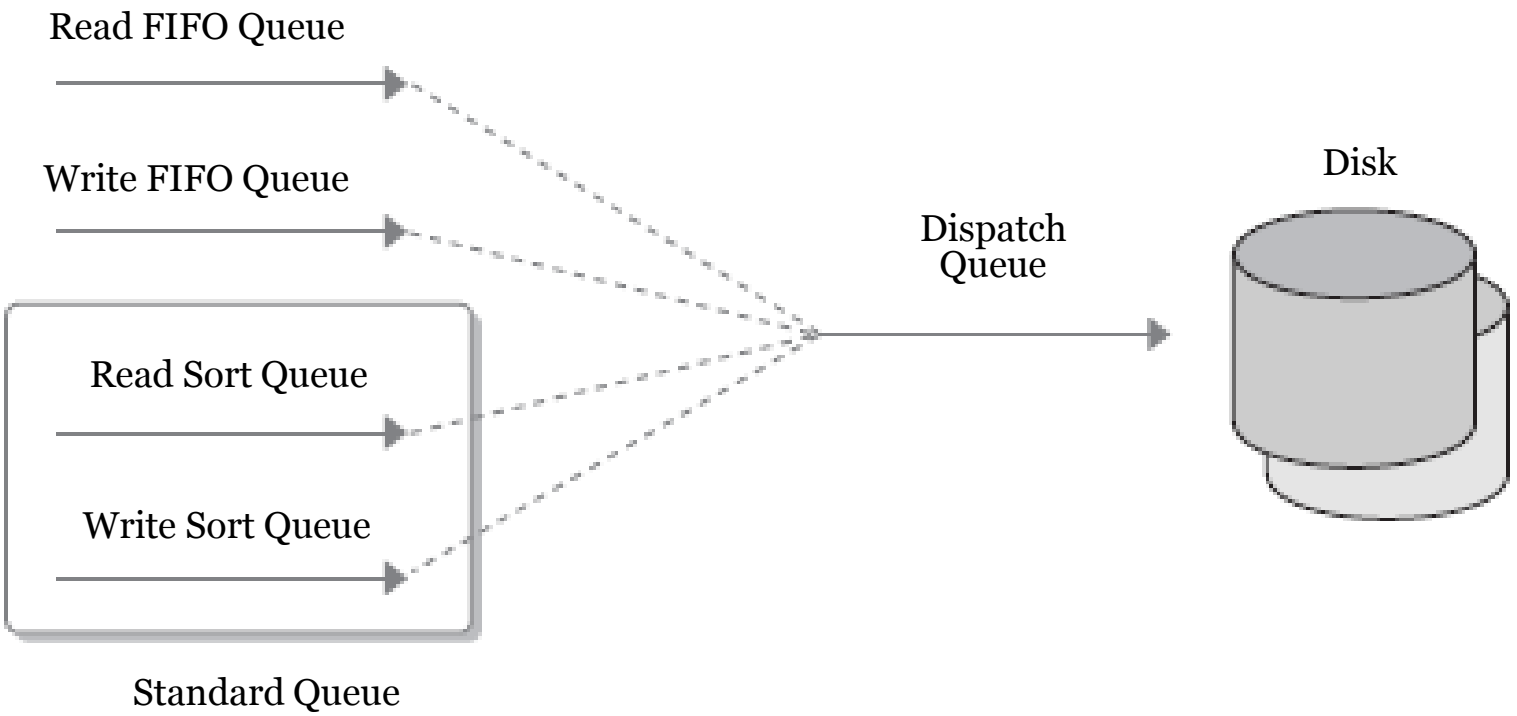
I/O Scheduler



Linus' Elevator

- Used in Kernel 2.4
- Front Merge & Back Merge
- Starvation in case of write bomb
  - elevator_sequence
  - Aging

Deadline Elevator (Kernel 2.6)



Elevators

- Anticipatory Elevator
- Complete Fairness Queuing (CFQ) Elevator
- Noop (No Operation)

- Currently used scheduler
  - cat /sys/block/had/queue/scheduler

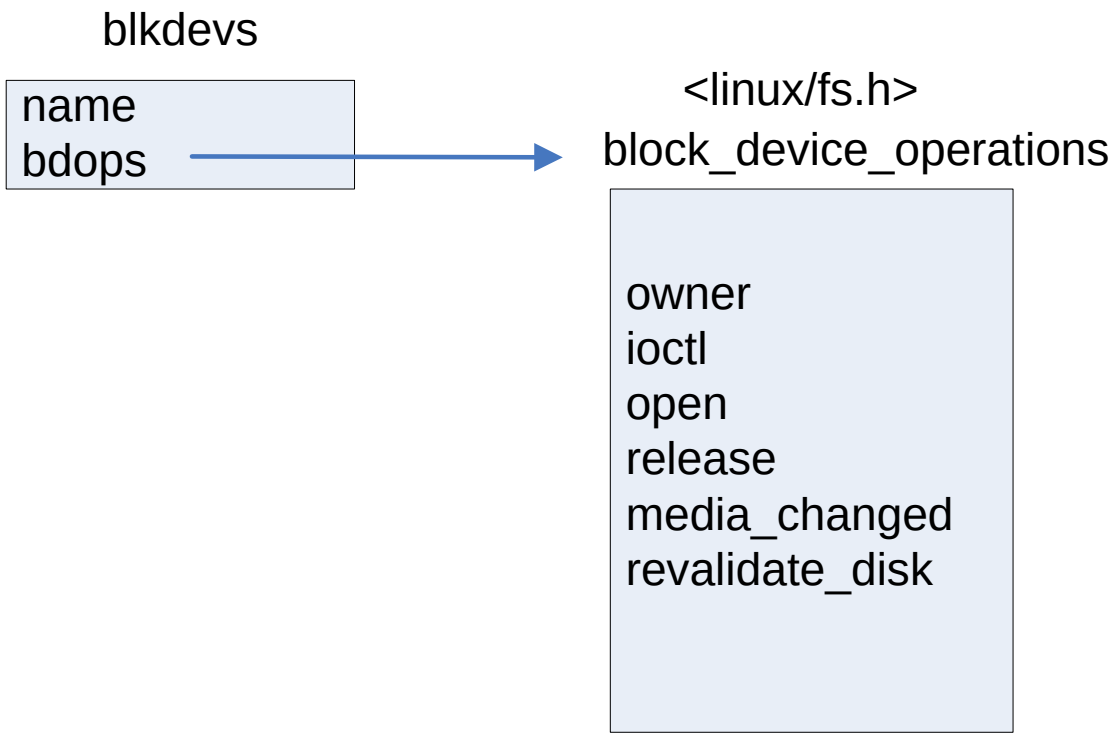
Data Structure for Block Device Driver

Kernel 2.4

- int register_blkdev(unsigned int major, const char *name, struct block_device_operations *bdops);
- int unregister_blkdev(unsigned int major, const char *name);

Kernel 2.6

- int register_blkdev(unsigned int major, const char *name);



Data Structure for Block Device Driver

Kernel 2.4
(divers/block/ll_rw_block.c)

- void blk_queue_make_request(request_queue_t *q, make_request_fn * mfn)

- #define BLK_DEFAULT_QUEUE(_MAJOR) &blk_dev[_MAJOR].request_queue
- typedef int (make_request_fn) (request_queue_t *q, int rw, struct buffer_head *bh)

Kernel 2.6
- typedef int (make_request_fn) (request_queue_t *q, int rw, struct bio *bio)

Block Device Driver

- Difference against Character Device Driver
  - Block-base process
  - Buffer management
  - Indirect access

Ram Disk Programming-1

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/fcntl.h>
#include <linux/vmalloc.h>
#include <linux/hdreg.h>
#include <asm/uaccess.h>
#include <linux/blk.h>
#include <linux/blkpg.h>

#define RD_DEV_NAME          "vrd"
#define RD_DEV_MAJOR         240
#define RD_MAX_DEVICES       2
#define MAJOR_NR             RD_DEV_MAJOR


#define RD_SECTOR_SIZE       512
#define RD_SIZE              (4*1024*1024)
#define RD_SIZE_KB           (RD_SIZE/1024)
#define RD_SECTOR_TOTAL      (RD_SIZE/RD_SECTOR_SIZE)
#define RD_AHEAD             2
```

Ram Disk Programming-2

```
static char *vdisk[RD_MAX_DEVICES] = {NULL,};
static int RD_size[RD_MAX_DEVICES] = {0 , };

static int RD_make_request( request_queue_t *q, int rw, struct buffer_head *sbh )
{
  unsigned int minor;
  char *pData;

  minor = MINOR(sbh->b_rdev);

  if( minor >= RD_MAX_DEVICES ) goto fail;
  if( ( ( sbh->b_rsector * RD_SECTOR_SIZE ) + sbh->b_size ) >= RD_SIZE ) goto fail;

  pData = vdisk[ minor ] + (sbh->b_rsector * RD_SECTOR_SIZE );
```

Ram Disk Programming-3

```
switch( rw ) {
    case READA:
    case READ:
      memcpy( sbh->b_data, pData, sbh->b_size );
      break;

    case WRITE:
      refile_buffer( sbh );
      memcpy( pData, sbh->b_data, sbh->b_size );
      mark_buffer_uptodate( sbh, 1 );
      break;
    default: goto fail;
  }
  sbh->b_end_io( sbh, 1 );
  return 0;
fail:
  buffer_IO_error( sbh );
  return 0;
}
```

Read Request Processing

Write Request Processing

Notify end of I/O request

Ram Disk Programming-4

```
int RD_open( struct inode *inode, struct file *filp ) {    MOD_INC_USE_COUNT;    return 0; }
int RD_release( struct inode *inode, struct file *filp ) {  MOD_DEC_USE_COUNT;   return 0; }
int RD_ioctl( struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg ) {
  int err, size;
  switch( cmd )  {
    case BLKGETSIZE:
      err = !access_ok( VERIFY_WRITE, arg, sizeof( long ) );
      if( err ) return -EFAULT;
      size = RD_SECTOR_TOTAL;
      if( copy_to_user( (long*)arg, &size, sizeof( long ) ) ) return -EFAULT;
      return 0;
    default:
      return blk_ioctl( inode->i_rdev, cmd, arg );
  }
  return -ENOTTY;
}
```

Return ths size of block device

Transfer it to blk_ioctl() in kernel

Ram Disk Programming-5

Block Device Operations
Declare & Function Specification

```
static struct block_device_operations RD_fops = {
   .open = RD_open,
   .release = RD_release,
   .ioctl = RD_ioctl
};
```

Ram Disk Programming-6

```
int init_module( void ) {
  int lp;
  register_blkdev( MAJOR_NR, RD_DEV_NAME, &RD_fops );
  blk_queue_make_request( BLK_DEFAULT_QUEUE(MAJOR_NR), &RD_make_request );
  read_ahead[MAJOR_NR] = RD_AHEAD;
  for( lp = 0; lp < RD_MAX_DEVICES; lp++ ) {
    RD_size[lp] = RD_SIZE_KB;
  }
  blk_size[MAJOR_NR] = RD_size;
  vdisk[0] = vmalloc( RD_SIZE );
  vdisk[1] = vmalloc( RD_SIZE );
  for( lp = 0; lp < RD_MAX_DEVICES; lp++ ) {
    register_disk( NULL,
        MKDEV(MAJOR_NR, lp ),
        1,
        &RD_fops,
        RD_SECTOR_TOTAL );
  }
  return 0;
}
```

Register Block Device

Register make_request()

Specify the number of Read Ahead

Virtual memory allocation

Register a disk

Ram Disk Programming-7

```
void clenaup_module(void)
{
  int lp;
  unregister_blkdev( MAJOR_NR, RD_DEV_NAME );
  vfree( vdisk[0] );
  vfree( vdisk[1] );

  read_ahead[ MAJOR_NR ] = 0;
  blk_size[ MAJOR_NR ] = NULL;
}

MODULE_LICENSE( "GPL" );
```

Unregister the block device and free the ram disk

RAM Disk

```
redhat92:/works/vdisk# mknod /dev/vrd0 b 240 0
redhat92:/works/vdisk# mknod /dev/vrd1 b 240 1
redhat92:/works/vdisk# insmod vrd.o
redhat92:/works/vdisk# lsmod
Module                  Size  Used by    Tainted: PF
vrd                     1128   0  (unused)
vmhgfs                 34528   4
debian24:/works/vdisk# mke2fs /dev/vrd0
mke2fs 1.37 (21-Mar-2005)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
1024 inodes, 4096 blocks
204 blocks (4.98%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
1024 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
redhat92:/works/vdisk#
```

Block Device Created

Format the ram disk