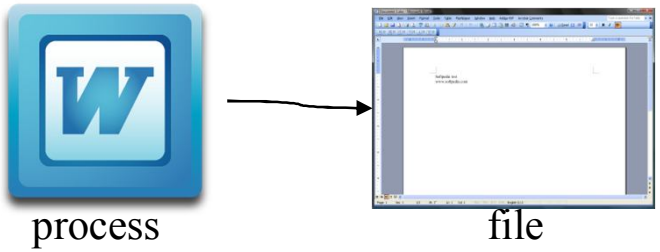
# File I/O

# 이 장의 기본 내용

- Process가 file을 사용하려면?
  - File system에서 file의 위치를 찾는다. ➔ open()
  - File의 data를 읽거나 쓴다. ➔ read()/write()
  - File 사용을 마친다. ➔ close()



# File Descriptor

- file descriptor
  - all open files are referred to by file descriptors.
  - how to obtain file descriptor
    - return value of open( ), creat( )
  - when we want to read or write a file,
    - we identify the file with the file descriptor
  - file descriptor is the index of user file descriptor table
  - STDIN_FILENO(0), STDOUT_FILENO(1), STDERR_FILENO(2)
    - defined in <unistd.h>
  - Ranged of file descriptor
    - 0 ~ OPEN_MAX (63 in many systems)

# open( )

```
#include <fcntl.h>

int open(const char *pathname, int oflag, ... /* mode_t mode   */ );
                                                    Returns: file descriptor if OK, -1 on error
```

- open/create a file and return a file descriptor.
  - What does "…" mean?
    - Third argument is used only when a new file is created.

# open( )

- pathname
  - The name of the file to open or create
- oflag
  - Access mode (One of three constants must be specified.)
  - O_RDONLY
    - Open for reading only
  - O_WRONLY
    - Open for writing only
  - O_RDWR
    - Open for reading and writing

# open( )

- oflag(cont.)
  - The followings are optional.
  - O_CREAT
    - Create the file if it doesn't exist.
    - Requires a third argument, mode.
  - O_EXCL
    - Generate an error if O_CREAT is also specified and the file already exists.
  - O_APPEND
    - Append to the end of file on each write.

# open( )

- oflag(cont.)
  - O_TRUNC
    - If the file exists and if it is successfully opened for either write-only or read-write, truncate its length to 0.
  - O_SYNC
    - Any writes on the resulting file descriptor will block the calling process until the data has been physically written to the underlying hardware

# open( )

- mode
  - specifies the permissions to use if a new file is created.
  - should always be specified when O_CREAT is in the flags, and is ignored otherwise.

- return value
  - return the new file descriptor, or -1 if an error occurred.
    - the lowest numbered unused descriptor

# open( )

- example

```
int fd;
fd = open(“/etc/passwd”, O_RDONLY);
fd = open(“/etc/passwd”, O_RDWR);

fd = open(“ap”, O_RDWR | O_APPEND);
fd = open(“ap”, O_RDWR | O_CREAT | O_EXCL, 0644);
```

# creat( )

```
#include <fcntl.h>

int creat(const char *pathname, mode_t mode);
                        Returns: file descriptor opened for write-only if OK, -1 on error
```

- Create a new file
  - It is equivalent to
    - open (pathname, O_CREAT|O_WRONLY|O_TRUNC, mode);
  - Note that the file is opened only for writing.

# close()

```
#include <unistd.h>

int close(int filedes);
                                        Returns: 0 if OK, -1 on error
```

- Close an open file
  - When a process terminates, all of its open files are closed automatically by the kernel.
  - ➔ Many program often do not explicitly close open files.

# read()

```
#include <unistd.h>

ssize_t read(int filedes, void *buf, size_t nbytes);
                        Returns: number of bytes read, 0 if end of file, -1 on error
```

- read up to *nbytes* from *filedes* into the buffer starting at *buf*
  - read() starts at the file's current offset.
  - Before a successful return, the offset is incremented by the number of bytes actually read.
- return value
  - On success, the number of bytes read is returned.
  - 0 indicates end of file.
  - On error, -1 is returned.

# read()

- the number of bytes actually read may be less than the amount requested.
  - If the end of regular file is reached before the requested number of bytes has been read.
  - When reading from a terminal device, up to one line is read at a time.
  - When reading from a network, buffering within the network may cause less than the requested amount to be returned.
  - When reading from a pipe, if the pipe contains fewer bytes than requested, read will return only what is available.
  - …

# write()

```
#include <unistd.h>

ssize_t write(int filedes, const void *buf, size_t nbytes);
                                Returns: number of bytes written if OK, -1 on error
```

- writes up to *nbytes* to the file referenced by *filedes* from the buffer starting at *buf*
  - write start at the file's current offset.
  - If O_APPEND was specified when the file was opened,
    - The file's offset is set to the end of file before write.
- return value
  - On success, the number of bytes written is returned.
  - On error, -1 is returned.

# read()/write()

- example

```
#include <unistd.h>
#include <stdio.h>
#define BUFFSIZE 8192

int main(void)
{
        int n;
        char buf[BUFFSIZE];

        while ((n=read(STDIN_FILENO, buf, BUFFSIZE))>0)
                if (write(STDOUT_FILENO, buf, n)!=n)
                  printf("write error\n");

        if (n<0)
                printf("read error\n");
        exit(0);
}
```

# read()/write()

- 실행 예

```
$ ./a.out
hello, world.
hello, world.
Are you enjoying this class?
Are you enjoying this class?
Ctrl + D
$
```

# lseek( )

```
#include <unistd.h>

off_t lseek(int filedes, off_t offset, int whence);
                                        Returns: new file offset if OK, -1 on error
```

- Explicitly repositions an open file's offset
  - The offset for regular files must be non-negative.
- return value
  - success: the resulting offset location as measured in bytes from the beginning of the file
  - error: -1

# lseek( )

- whence
  - SEEK_SET
    - The offset is set to offset bytes from the beginning of the file.
  - SEEK_CUR
    - The offset is set to its current location plus offset bytes.
  - SEEK_END
    - The offset is set to the size of the file plus offset bytes.

# lseek( )

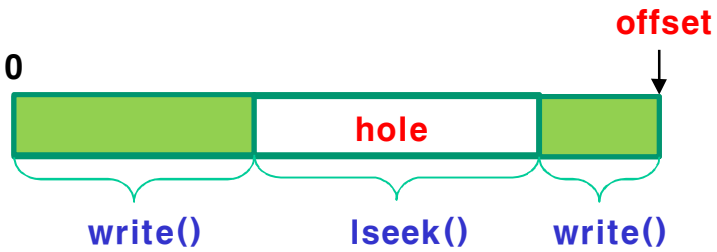
- example

```
off_t curpos;
curpos = lseek(fd, 0, SEEK_CUR);              // get the current offset

lseek(fd, 0, SEEK_SET);
lseek(fd, 0, SEEK_END);
lseek(fd, -10, SEEK_CUR);
lseek(fd, 100, SEEK_END);
```

# lseek( )

- hole
  - The file's offset can be greater than the file's size.
    - Next write to the file will extend the file.
  - It means that a hole in file is created and is allowed.
  - read from the data in hole returns 0.



# lseek( )

- example

```
#include "apue.h"
#include <fcntl.h>

char    buf1[] = "abcdefghij";
char    buf2[] = "ABCDEFGHIJ";

int
main(void)
{
    int     fd;

    if ((fd = creat("file.hole", FILE_MODE)) < 0)
        err_sys("creat error");
    /* FILE_MODE is defined as 644 in “apue.h”. */
```

# lseek( )

- example(cont.)

```
    if (write(fd, buf1, 10) != 10)
        err_sys("buf1 write error");
    /* offset now = 10 */

    if (lseek(fd, 16384, SEEK_SET) == -1)
        err_sys("lseek error");
    /* offset now = 16384 */

    if (write(fd, buf2, 10) != 10)
        err_sys("buf2 write error");
    /* offset now = 16394 */

    exit(0);
}
```

# lseek( )

## 실행 예

```
$ ./a.out
$ ls -l file.hole                check its size
-rw-r--r-- 1 sar          16394 Nov 25 01:01 file.hole
$ od -c file.hole                let's look at the actual contents
0000000   a  b  c  d  e  f  g  h  i  j \0 \0 \0 \0 \0 \0
0000020  \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
*
0040000   A  B  C  D  E  F  G  H  I  J
0040012
```

byte offset in octal

od utility: dump files in octal.
-c: print the contents as characters

# lseek( )

- Are the disk blocks allocated for hole?

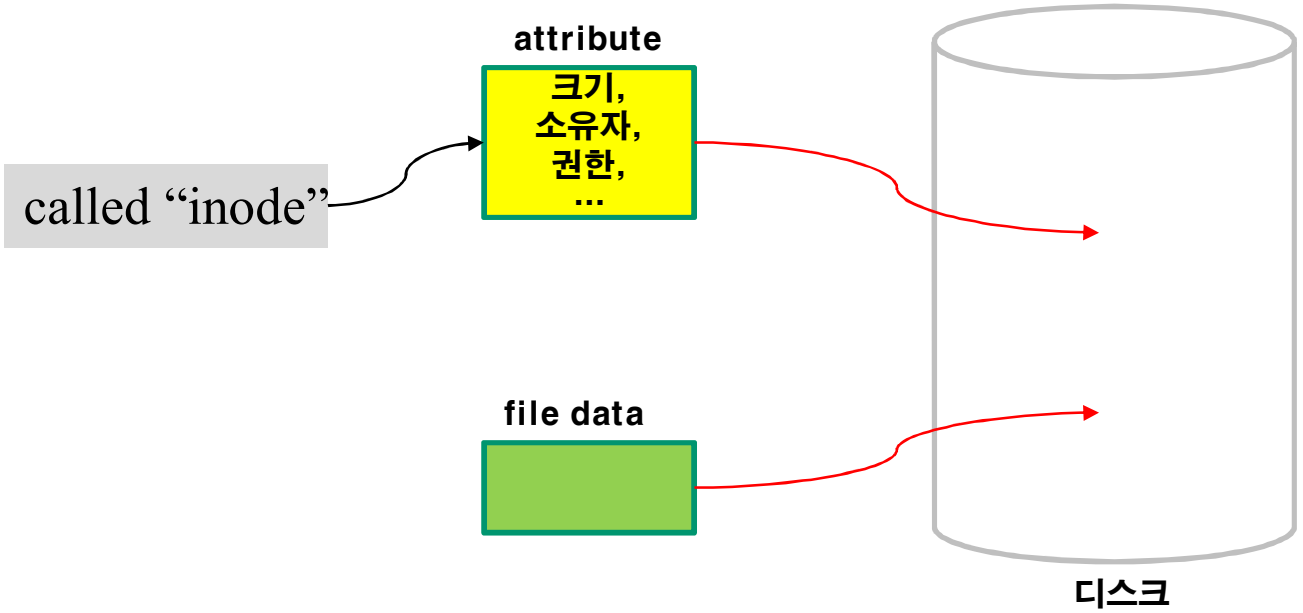
```
$ ls -ls file.hole file.nohole
    8 -rw-r--r-- 1 sar        16394 Nov 25 01:01 file.hole
   20 -rw-r--r-- 1 sar        16394 Nov 25 01:03 file.nohole
```

  - Compare the sizes of file.hole and file.nohole
    - file.hole: with hole
      - ➔ 8 blocks are allocated
    - file.nohole: a file of the same size, but without holes.
      - ➔ 20 blocks are allocated

ls utility
-s: with -l, print size of each file, in blocks.

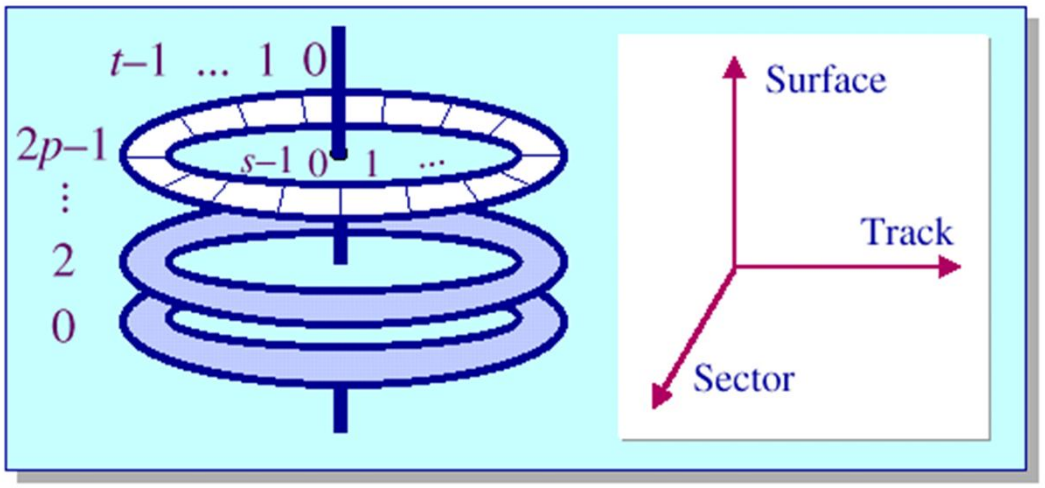
# File system

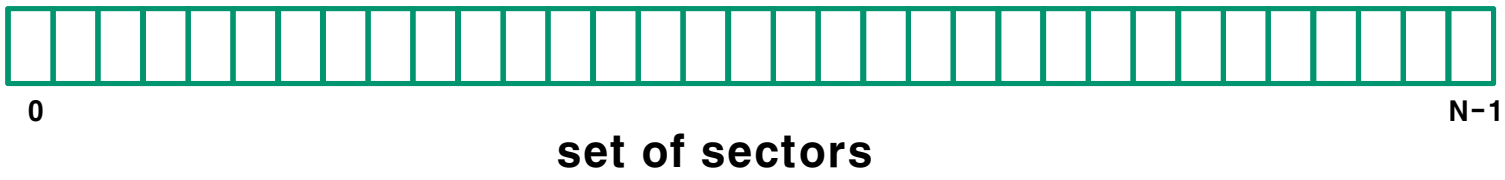
- File system
  - file data과 file's attribute의 저장, 검색



# File system

- 디스크 내 3D 구조에서 1D 구조로의 매핑

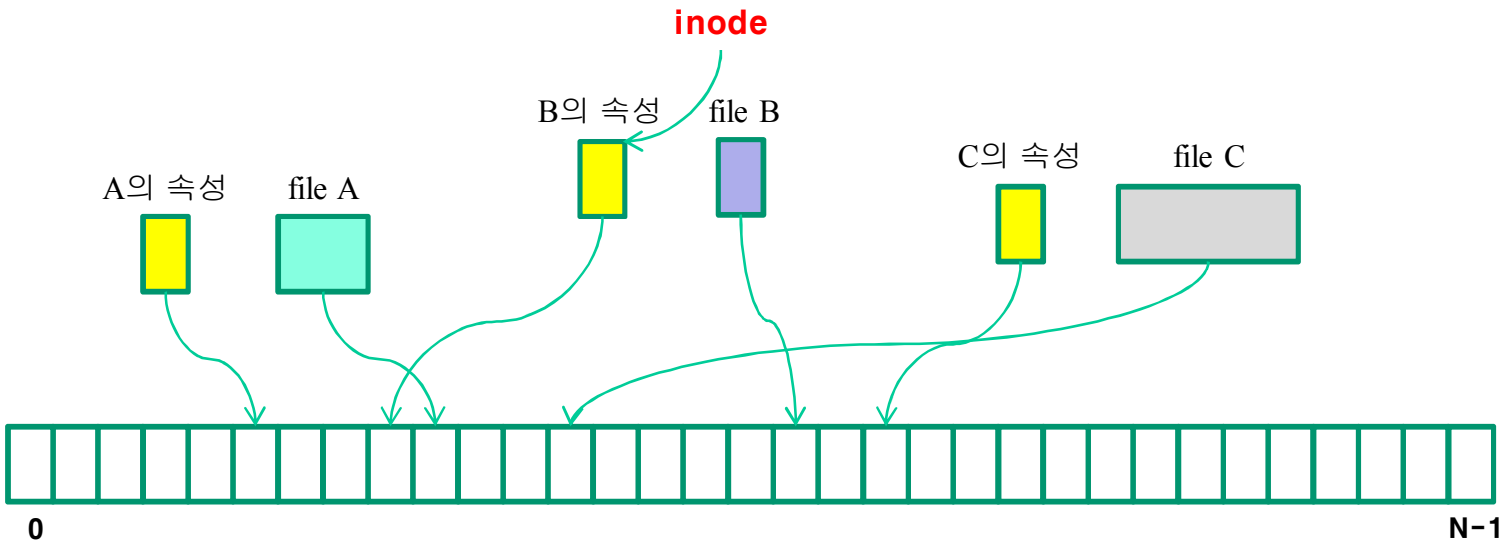


# File system

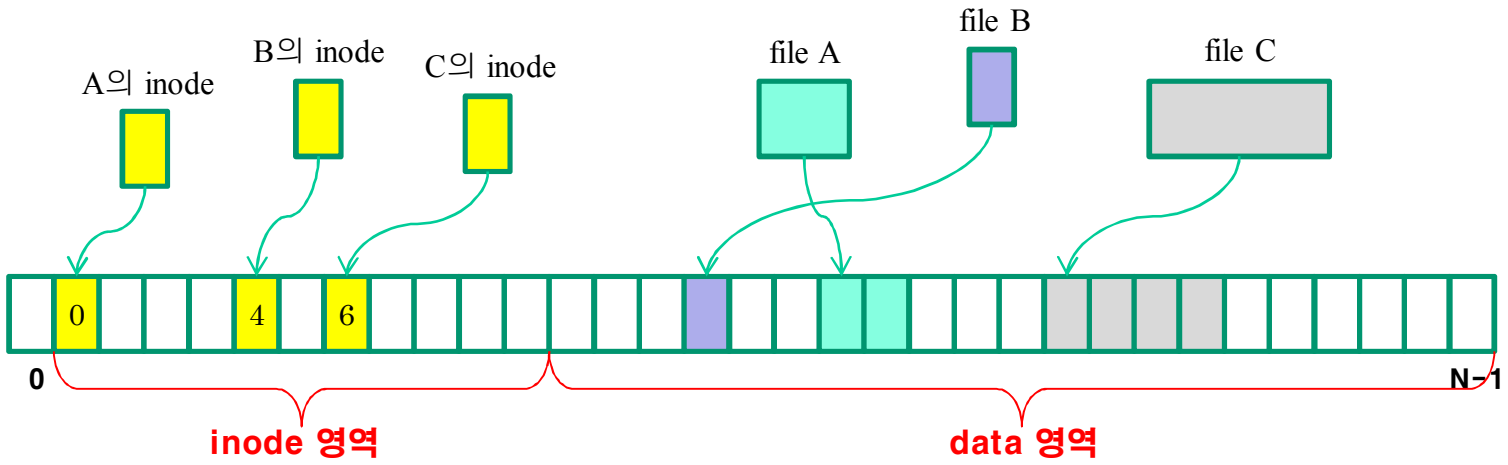
## File system

- inode는 file마다 하나씩 존재.
- 아래의 block array가 하나의 partition이라 가정.



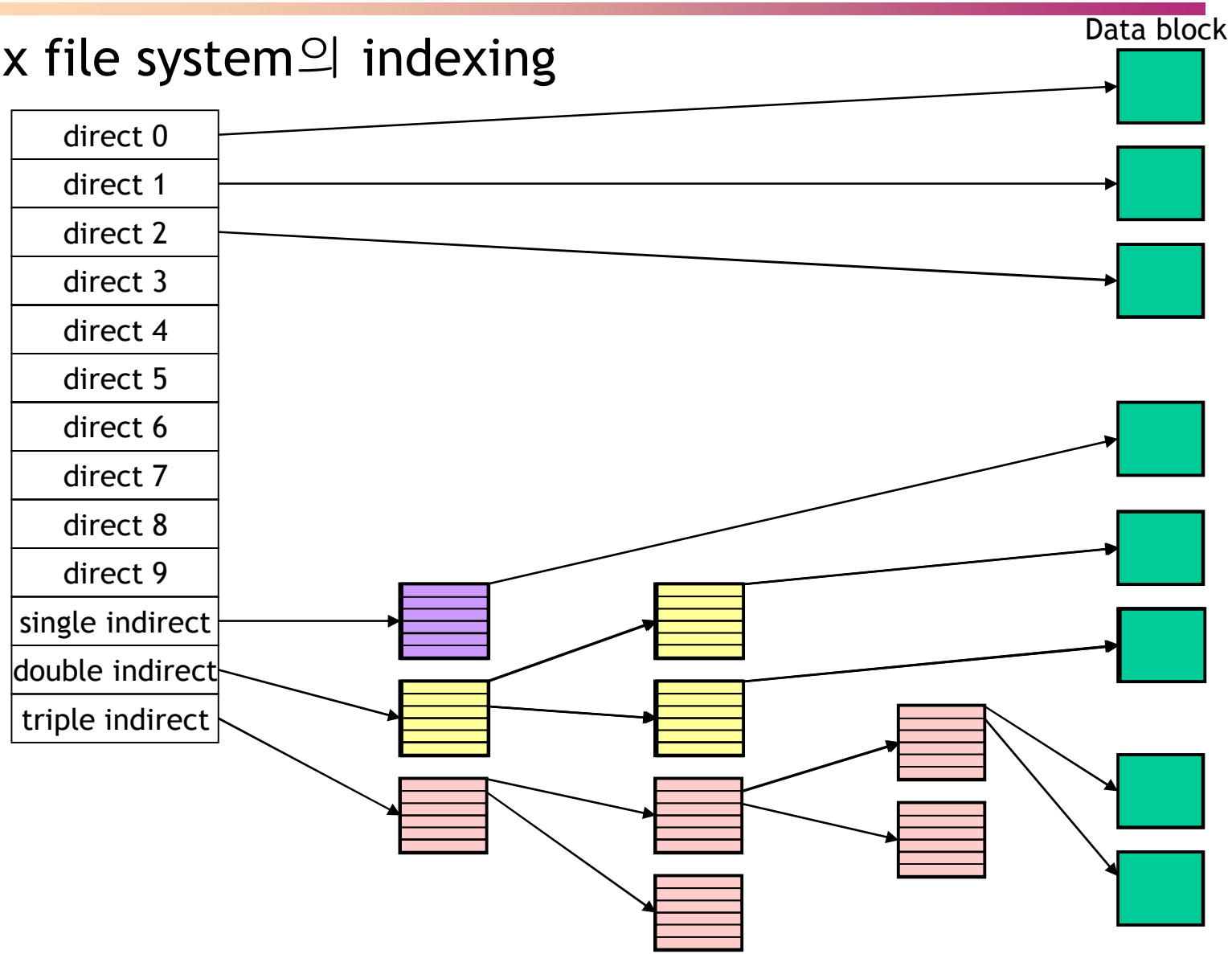
# File system

- Unix file system의 예
  - inode 저장 영역과 data block 저장 영역의 분리
  - inode 크기는 일정 (attribute 정보이므로)
    - ➔ i-number로 접근 가능 (아래 그림에서 0, 4, 6)



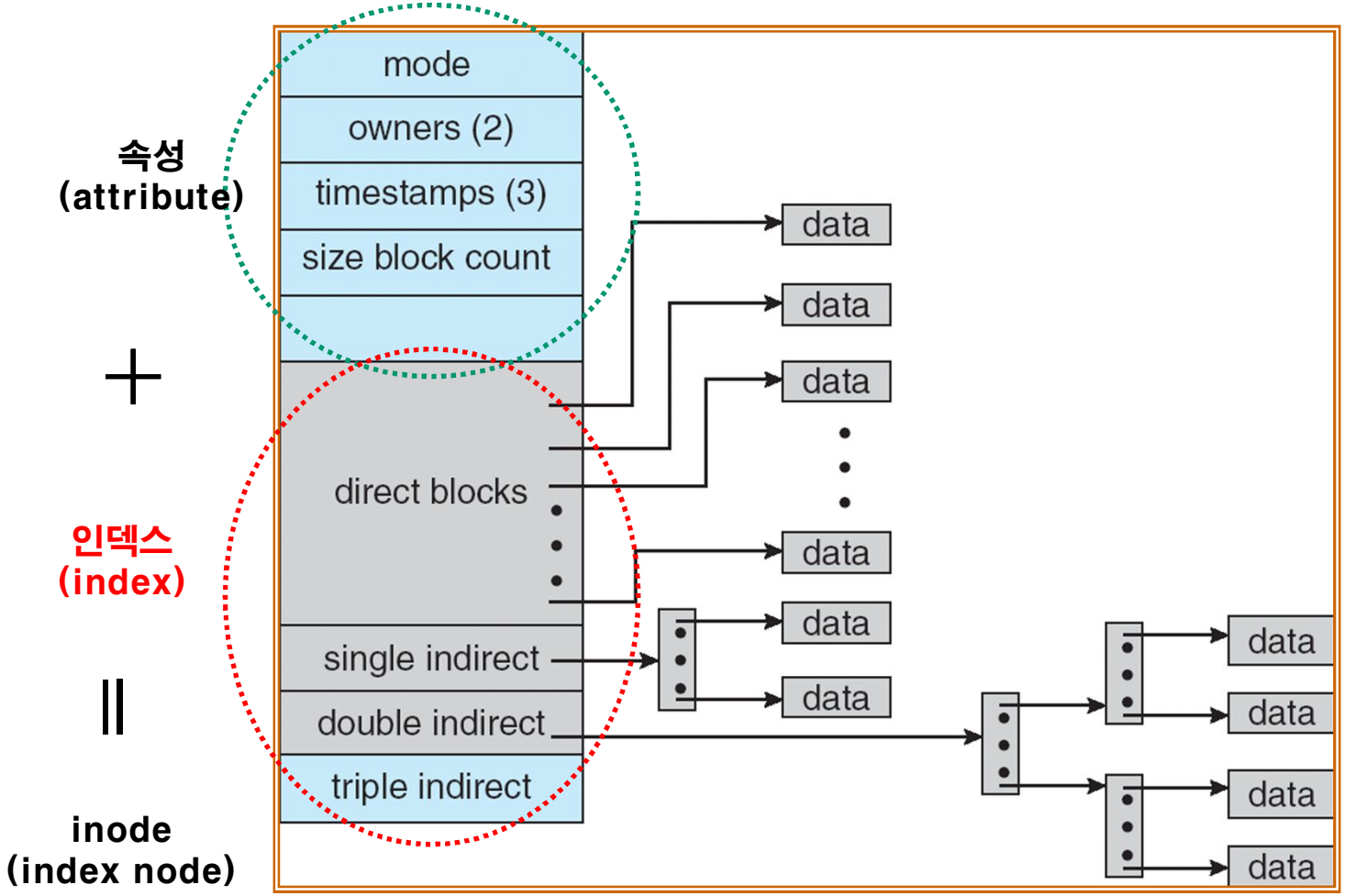
# File system

- Unix file system의 indexing



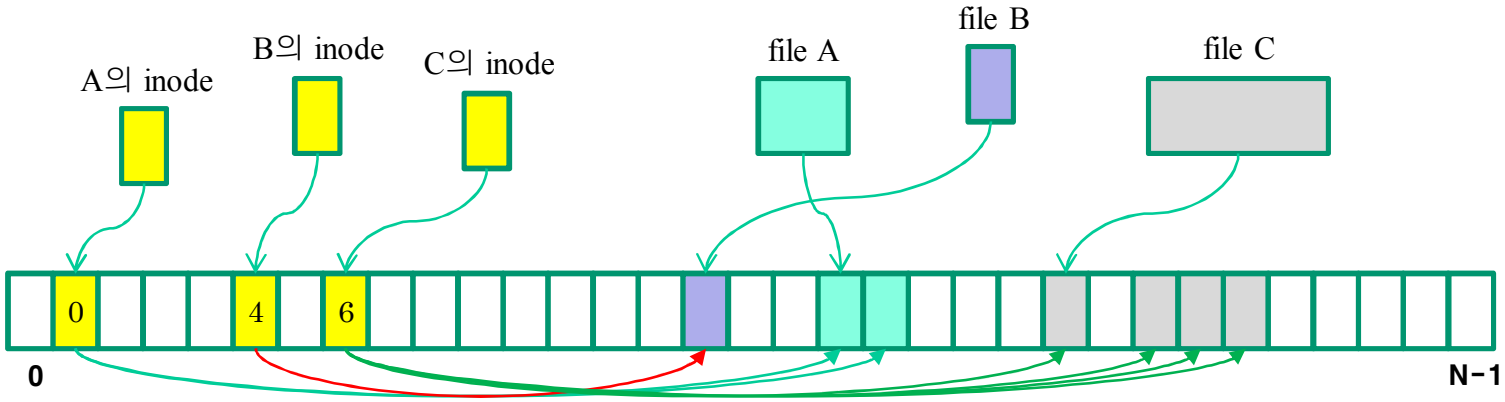
# File system

- Unix file system의 inode
  - attribute외에 data block을 찾을 수 있는 pointer정보를 포함.



# File system

- inode에서의 data block 연결



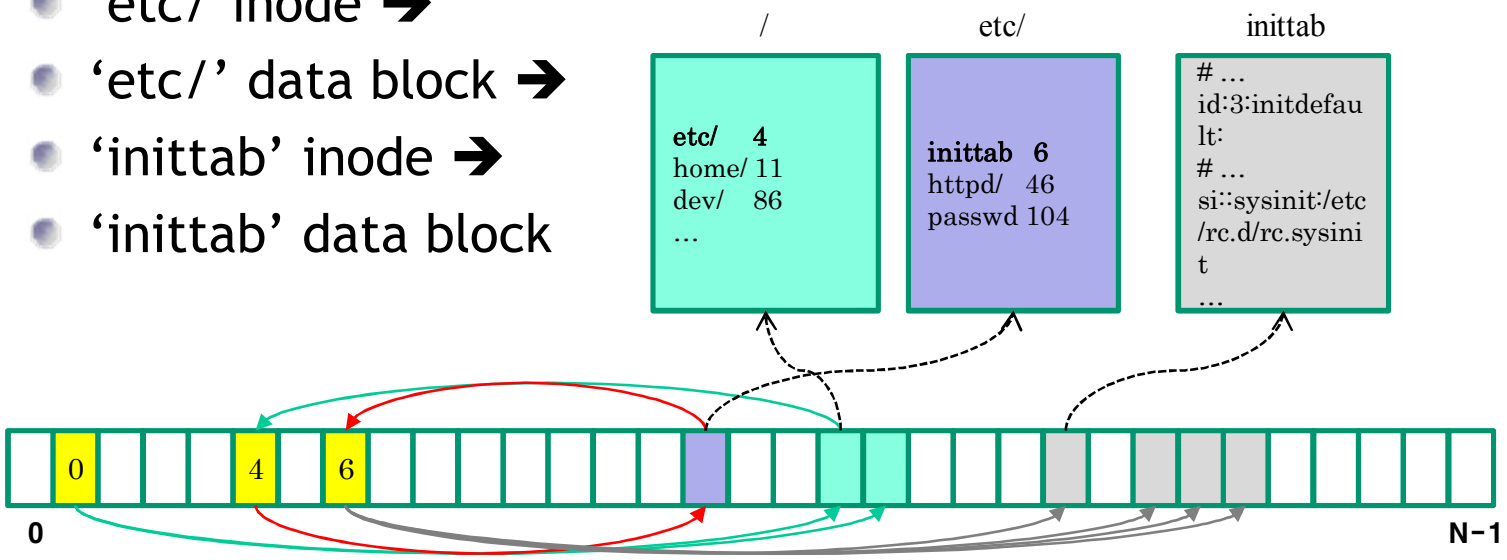

  - 그렇다면 정작 inode는 어떻게 찾을까?
    - Directory file에 file name 및 inode number를 저장.

"/" directory

| | |
|---|---|
| **etc/** | **4** |
| home/ | 11 |
| dev/ | 86 |
| … | |

# File system

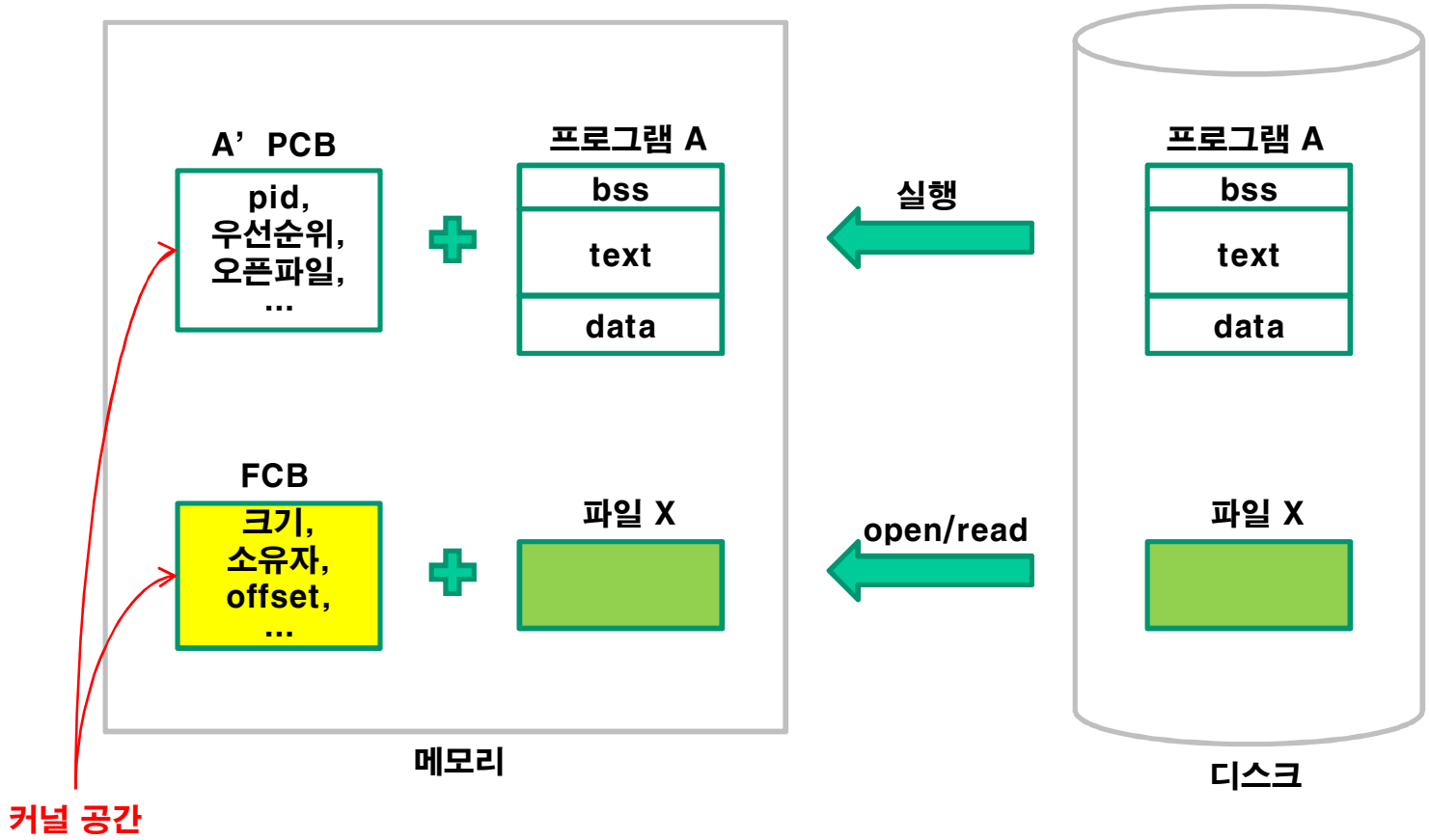
- “/etc/inittab” 파일을 접근하려면?
  - ‘/’ inode ➔
  - ‘/’ data block ➔
  - ‘etc/’inode ➔
  - ‘etc/’ data block ➔
  - ‘inittab’ inode ➔
  - ‘inittab’ data block




  - ‘/’ inode는 어떻게 찾는가?
    - 일반적으로 ‘/’의 i-number는 0임.

# Process가 file을 사용하려면?

- file의 metadata를 kernel 공간에서 관리해야 함.



# Process가 file을 사용하려면?

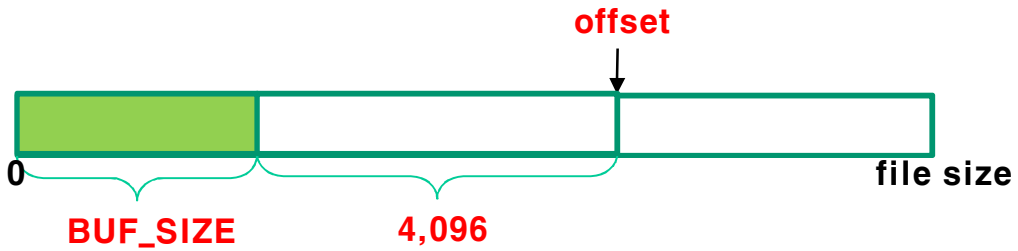
- file을 관리하기 위한 커널 내 metadata(FCB)
  - 크기 (e.g. 16KB)
  - 유형 (e.g. regular file)
  - 소유자 (e.g. obama)
  - 접근 권한 (e.g. rwxr--r--)
  - 데이터 블록 인덱스 (e.g. sector address)
  - 장치 (e.g. /dev/hda0)
  - 접근 위치 (e.g. offset)
  - …

# File I/O system call review

- fd = open("/etc/inittab", O_RDONLY);

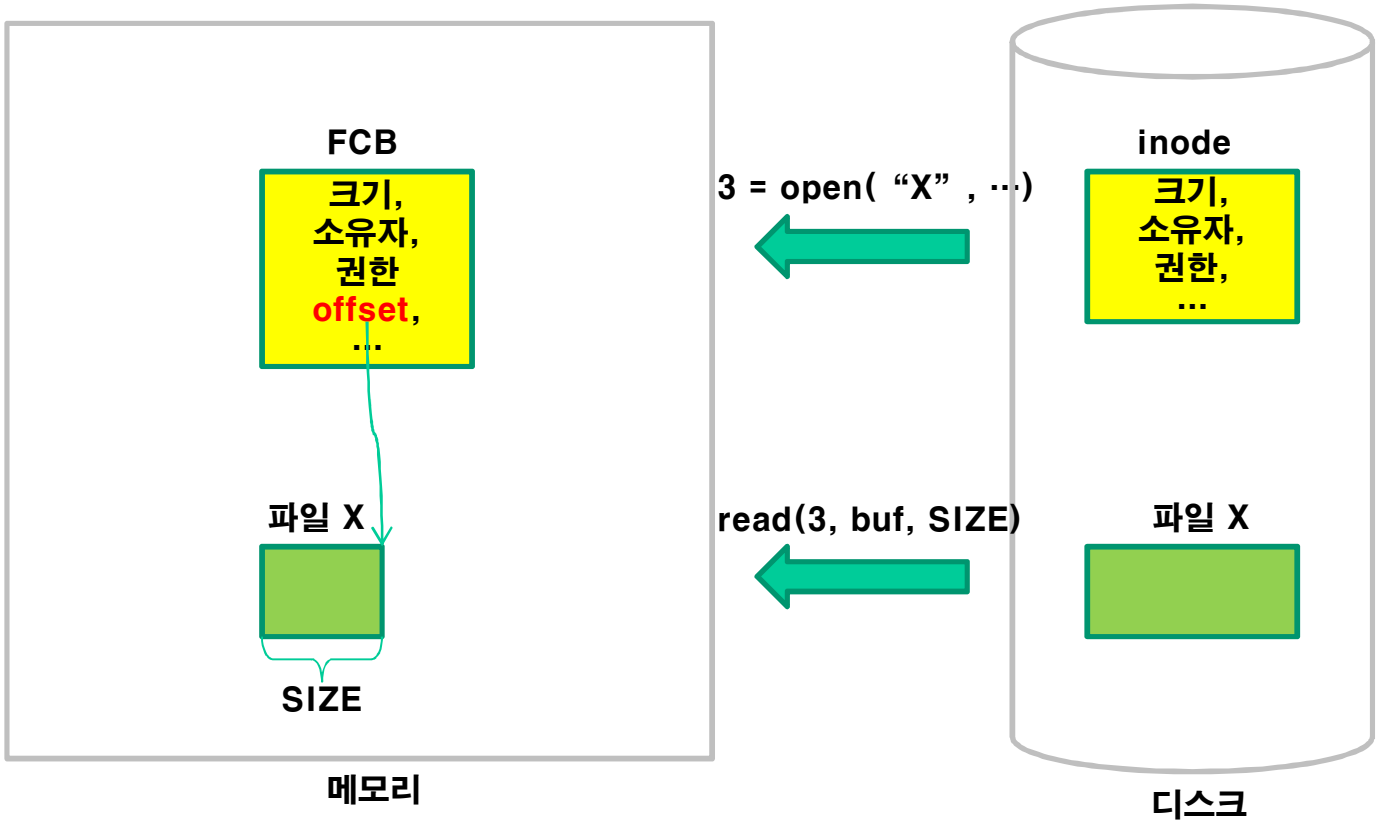

- nread = read(3, buffer, BUF_SIZE);


- npos = lseek(3, 4096, SEEK_CUR);



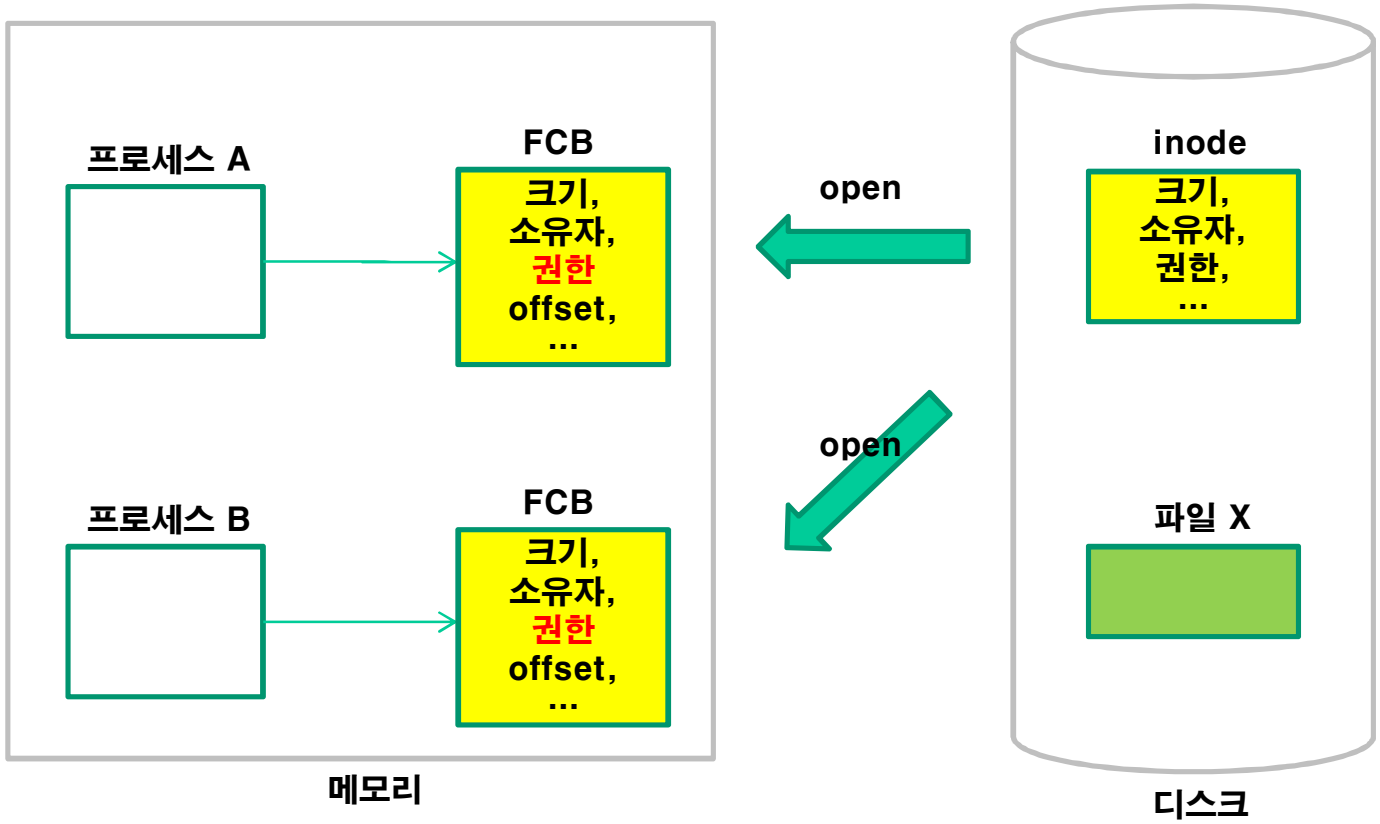
# File I/O system call review

- open과 read



# Process가 file을 사용하려면?

- 동일파일을 두 process가 사용하려면 2개의 FCB 필요.
  - process A가 권한 정보를 수정한다면?
    - $ chmod a+w X



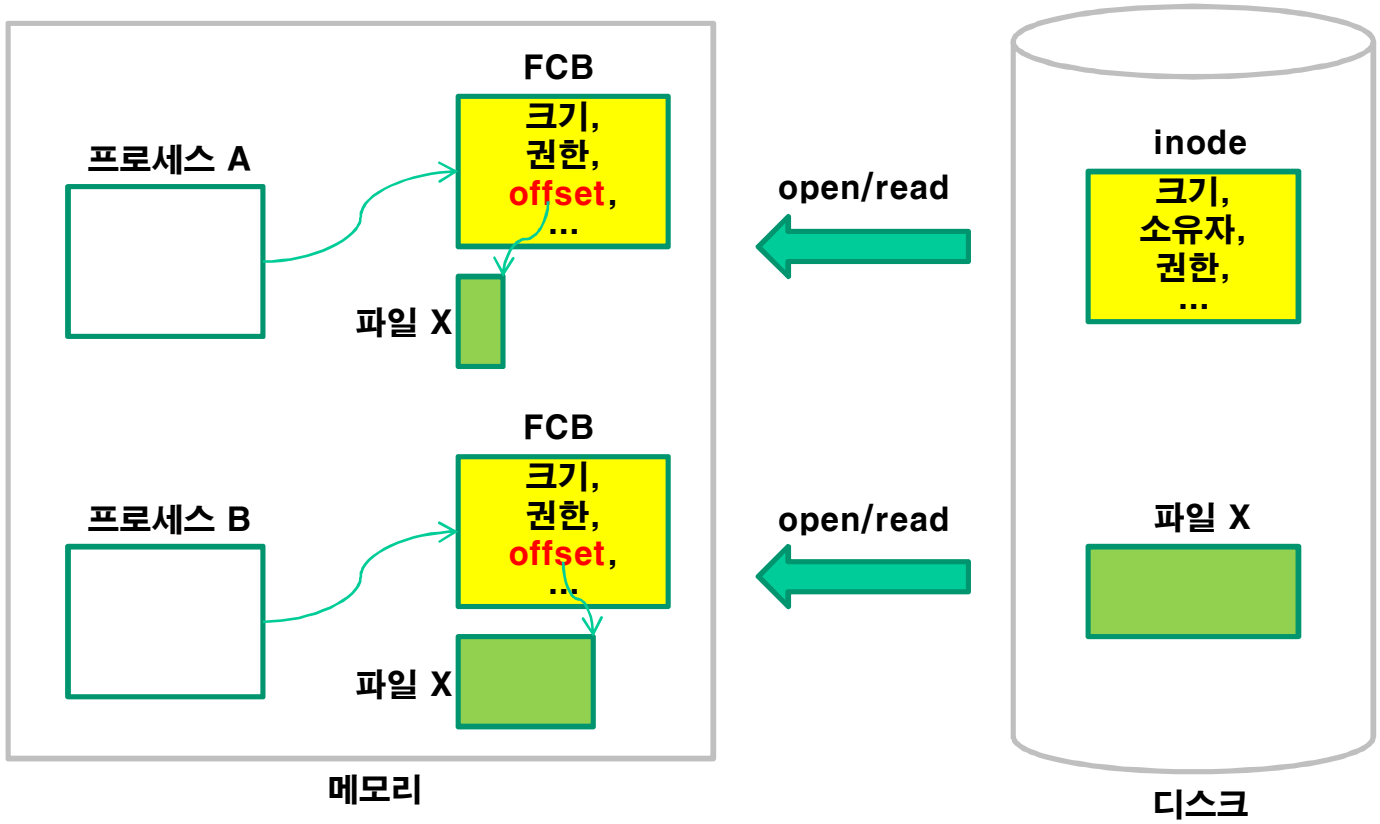
# Process가 file을 사용하려면?

- metadata를 수정한 후, 모두 복사한다면?
  - 불일치 (inconsistency) 발생 가능성
  - 비효율적(inefficient)

- 가능하면 metadata를 공유
  - 접근 권한, 파일의 크기, 유형 등은 process 간 공유 가능.
  - 하지만, offset은?
    - 모든 process가 다른 위치에서 read/write 중.
    - 이 정보는 process마다 별도로 관리해야 함.

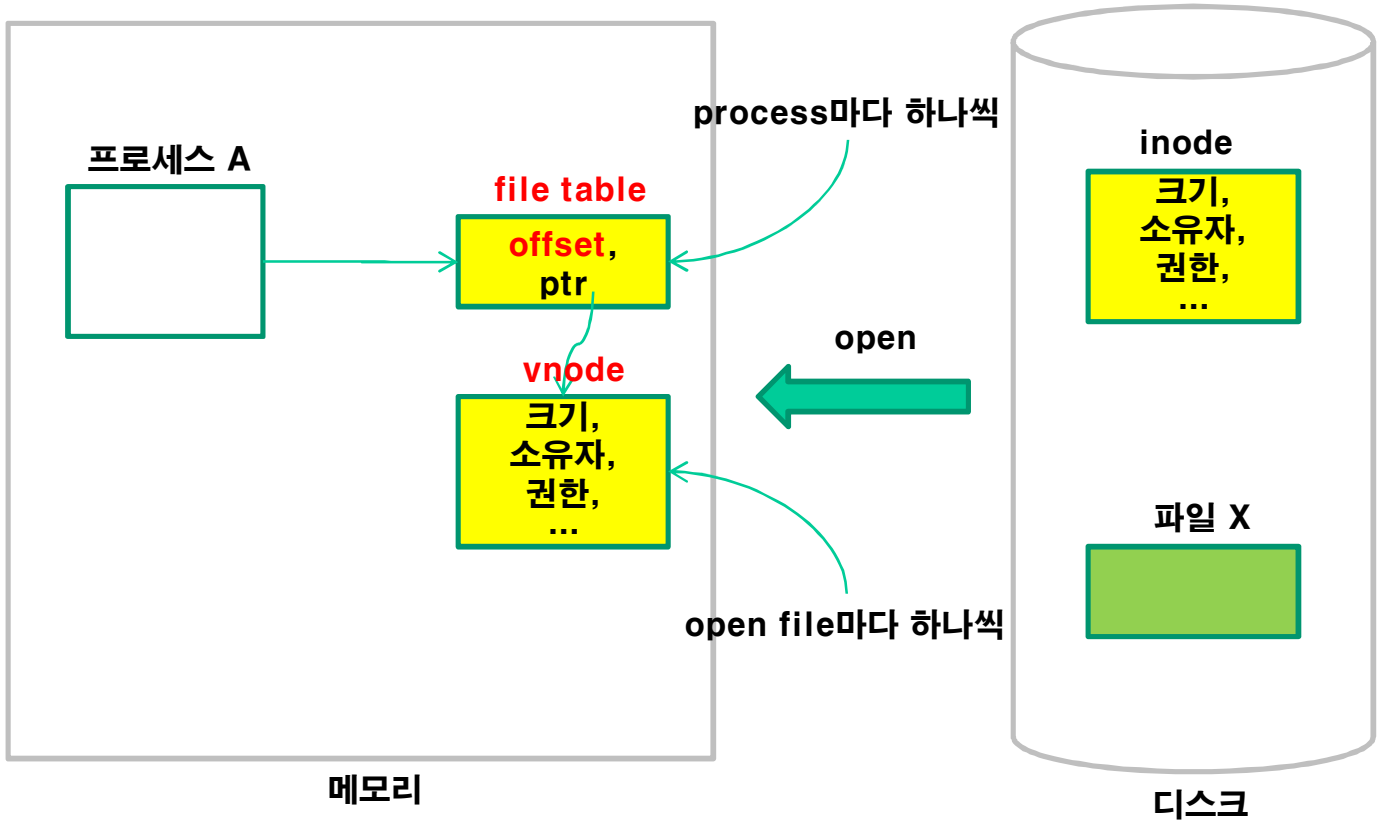
# Process가 file을 사용하려면?

- 자료구조 분리의 필요성
  - 소유자, 권한은 process간 공유 가능
  - offset은 process간 공유 불가능



# Process가 file을 사용하려면?

- 자료구조의 분리
  - offset만 따로 file table에 저장
  - 나머지 정보는 vnode에 저장



# Process가 file을 사용하려면?

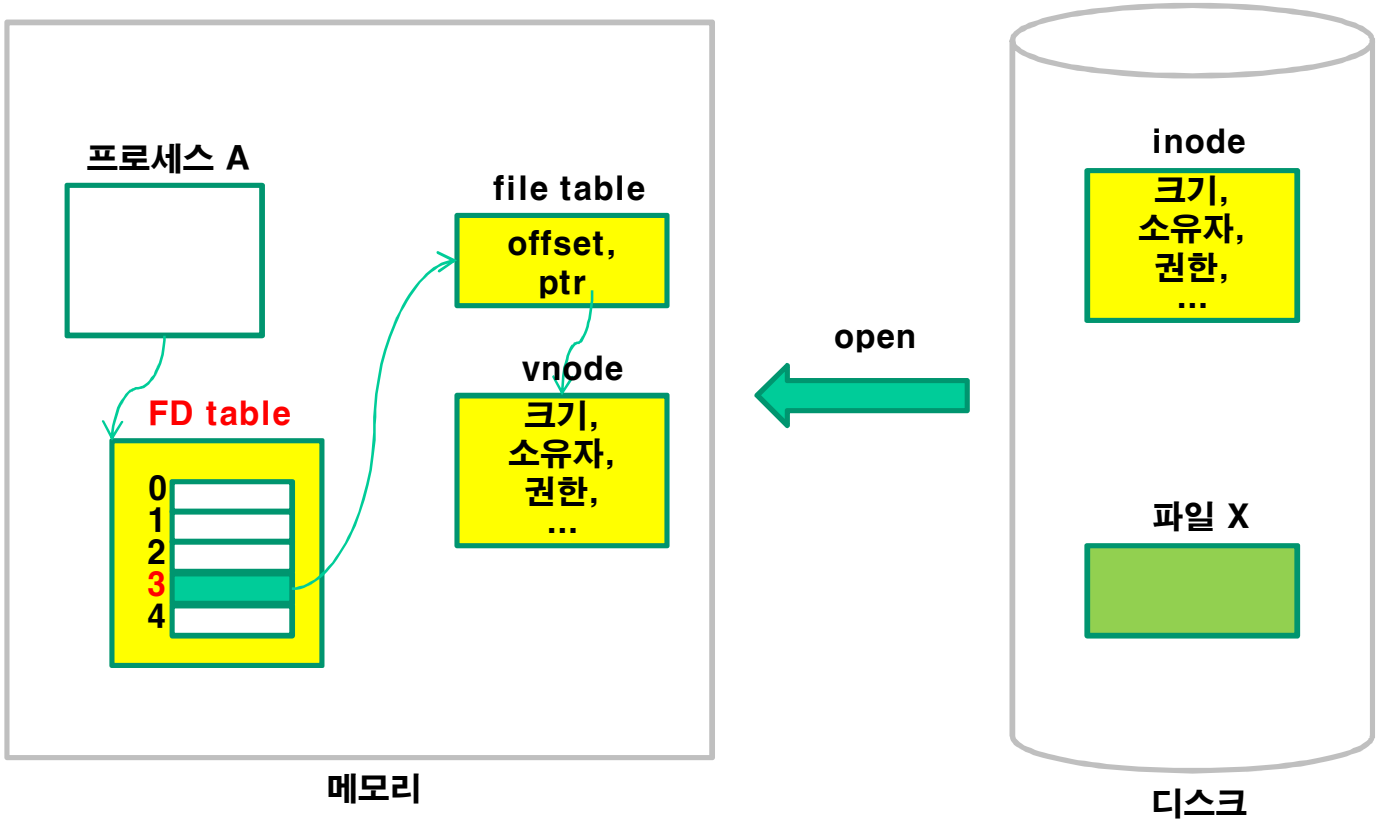
- file table
  - Open할 때마다 하나씩 생성.
  - 내용
    - offset
    - vnode에 대한 포인터

# Process가 file을 사용하려면?

- vnode
  - offset보다 덜 자주 바뀌는 정보들.
  - 대부분의 정보를 디스크의 inode에서 그대로 읽어 옴.
    - 접근 권한(protection mode)
    - 소유자(owner)
    - 크기(size)
    - 시간(time)
    - 디스크 상의 data block 위치

# Process가 file을 사용하려면?

- 자료구조의 분리 - file descriptor table의 추가
  - open() 후 자료 구조를 용이하게 접근하기 위해

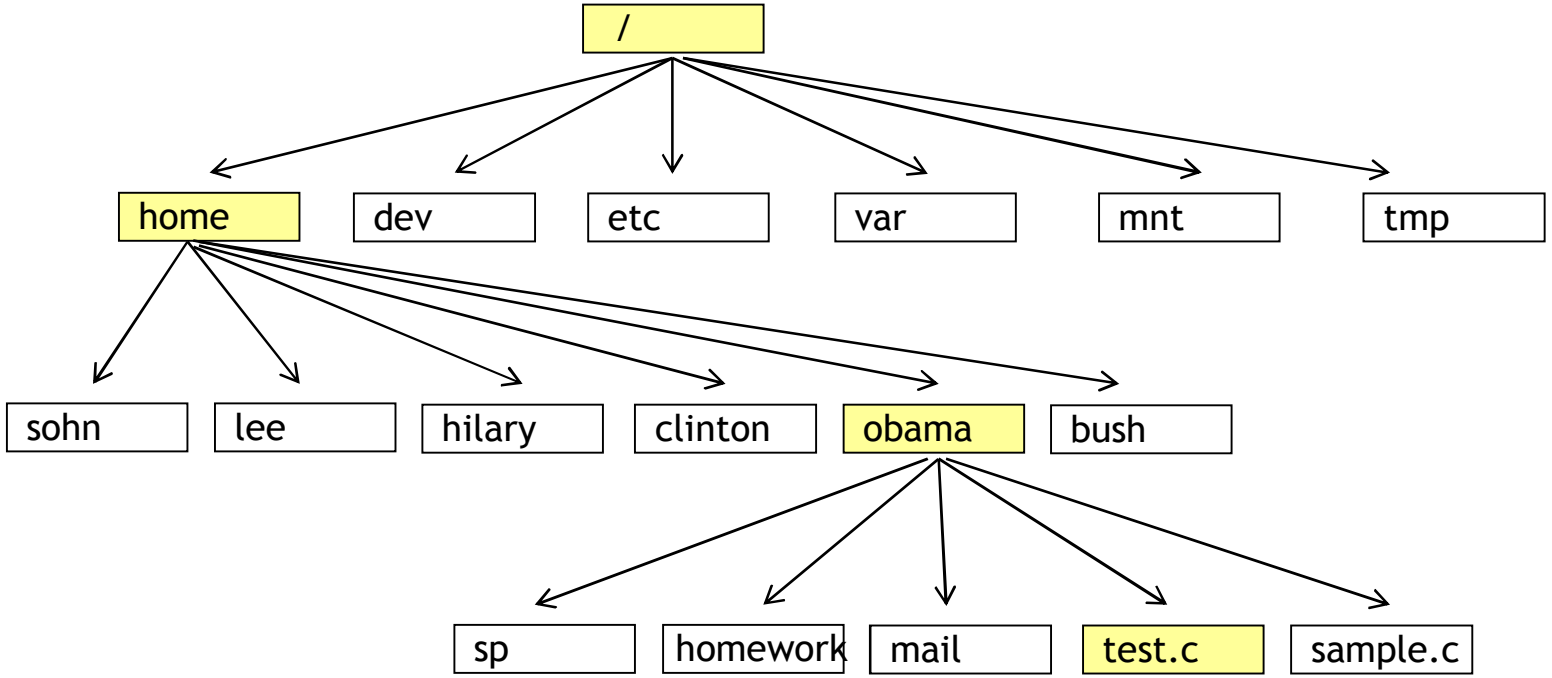


# Process가 file을 사용하려면?

- file descriptor table
  - Per process data structure
  - fd = open(“/a/b”, …)
  - fd는 file을 접근하기 위한 index
  - fd는 0부터 시작하는 정수(file descriptor)
    - 0, 1, 2는 standard input/output/error로 예약됨.

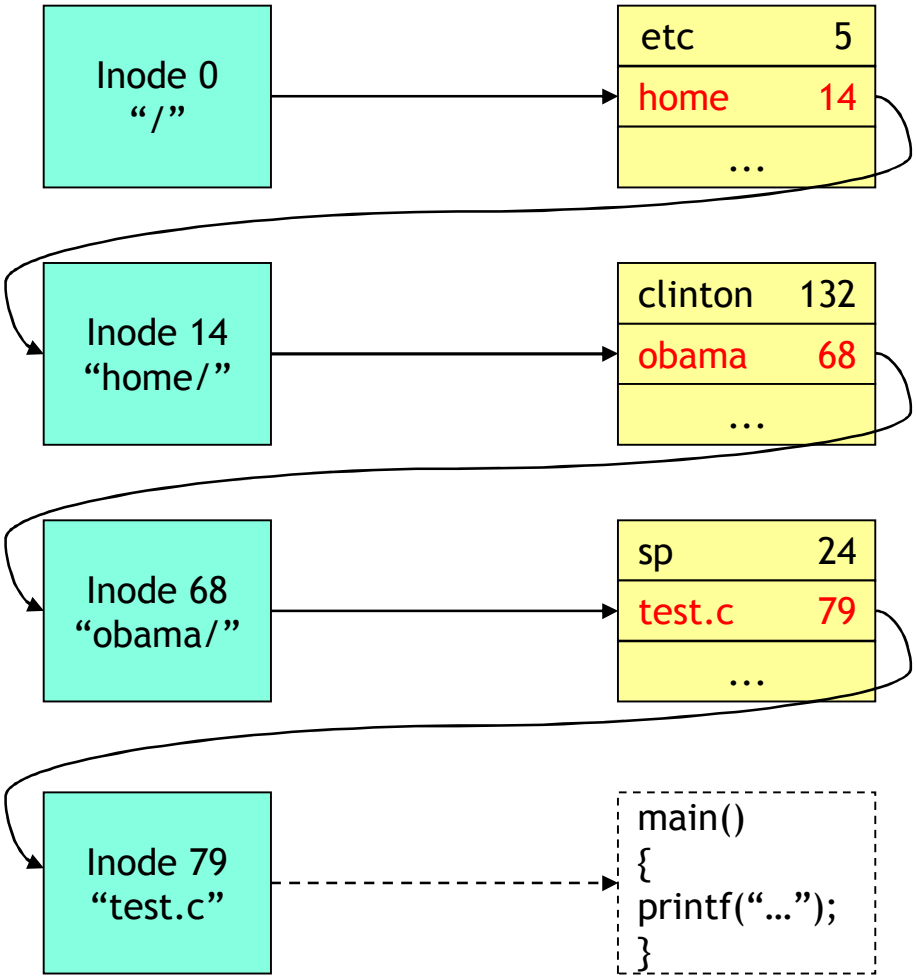
# open() 정리

- $ cat /home/obama/test.c
  - open("/home/obama/test.c", O_RDONLY)



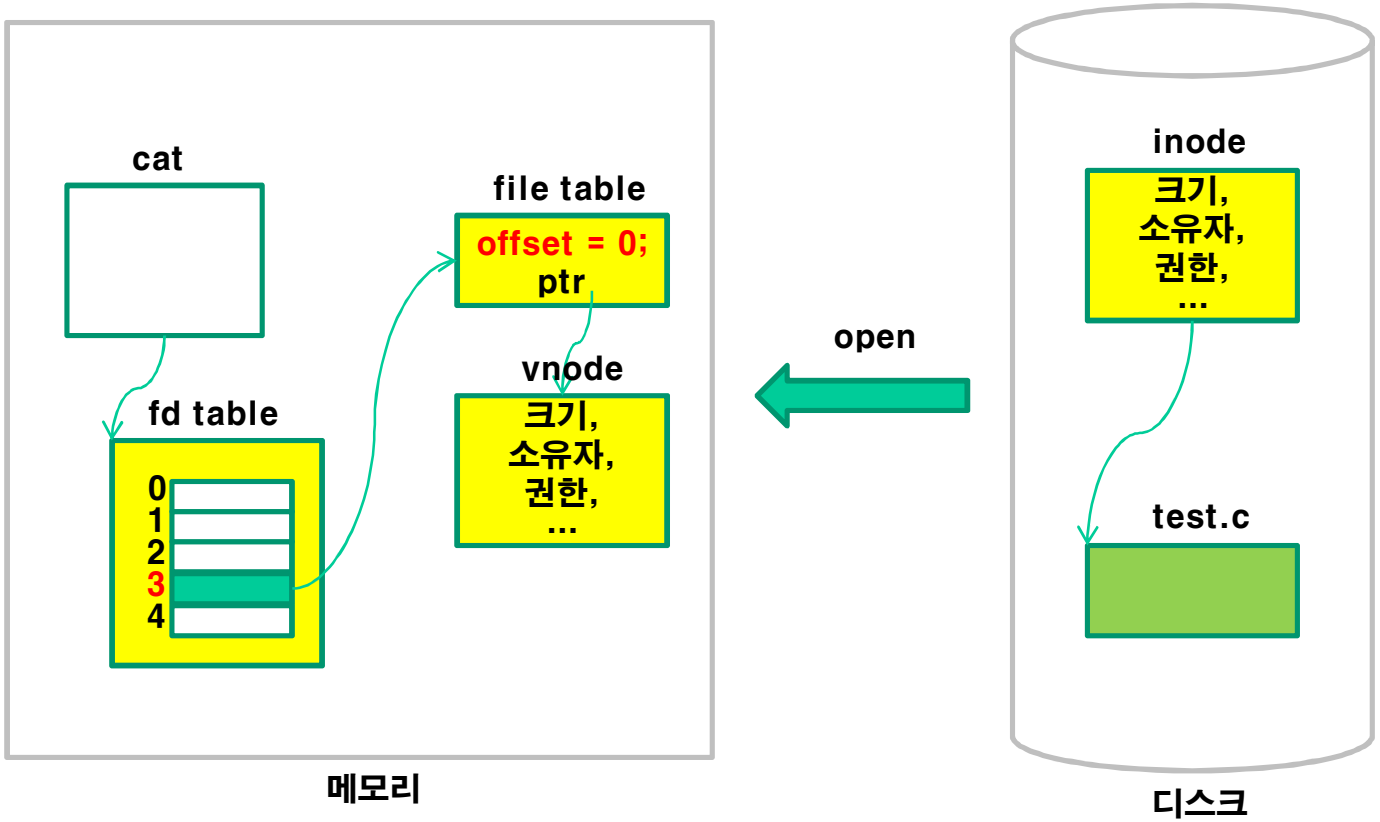
# open() 정리

- 1. Pathname lookup



# open() 정리

- 2. “test.c”를 위한 커널 내 자료 구조 완성
  - 메모리 내 vnode 생성.
  - file table 생성.(offset을 0으로 설정)
  - File descriptor table에 entry 생성하고, file descriptor 리턴.

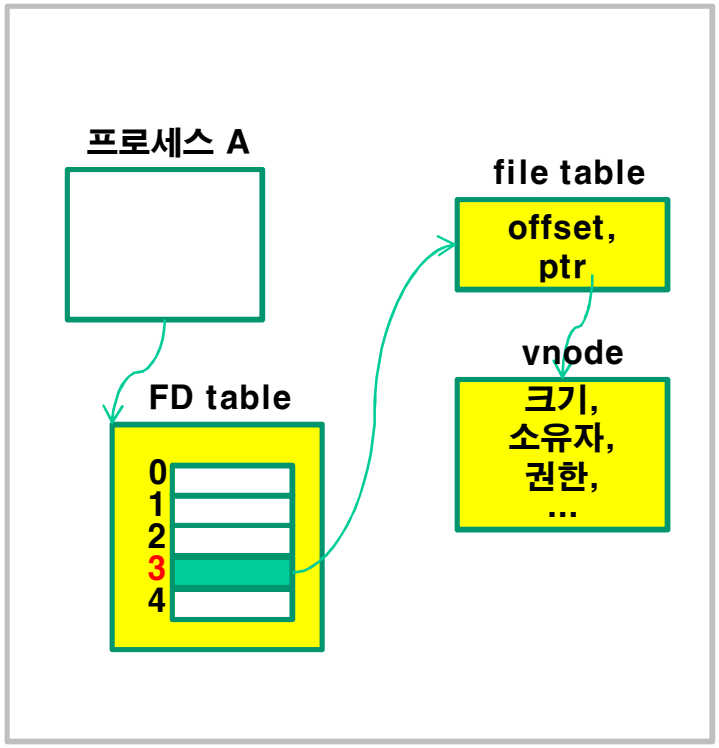


# open() 정리

- Pathname lookup 오버헤드
  - open(“/a/b”, …)는 여러 차례의 I/O연산 요구.
  - Pathname lookup은 한 번만 수행!
  - (pathname ➔ file descriptor) 변환 후 저장.
    - fd = open(“/a/b”, …)
  - 계속되는 시스템 콜 사용시 path 대신 file descriptor 사용.
    - read(fd, …), write(fd, …), …

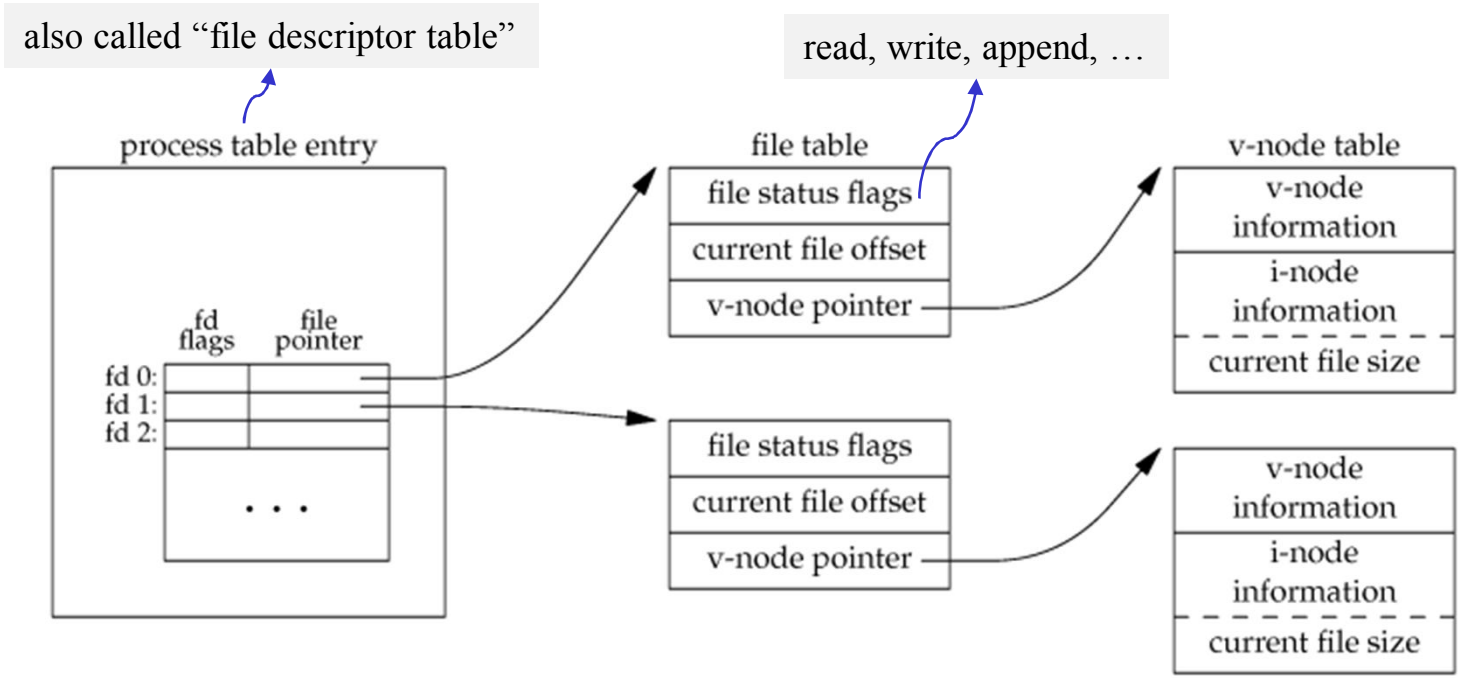
# File sharing

- Process가 file을 사용하기 위해서 kernel은 세 가지 data structure를 가진다.
  - File descriptor table
  - File table
  - Vnode table
- ➔ file sharing이 가능.



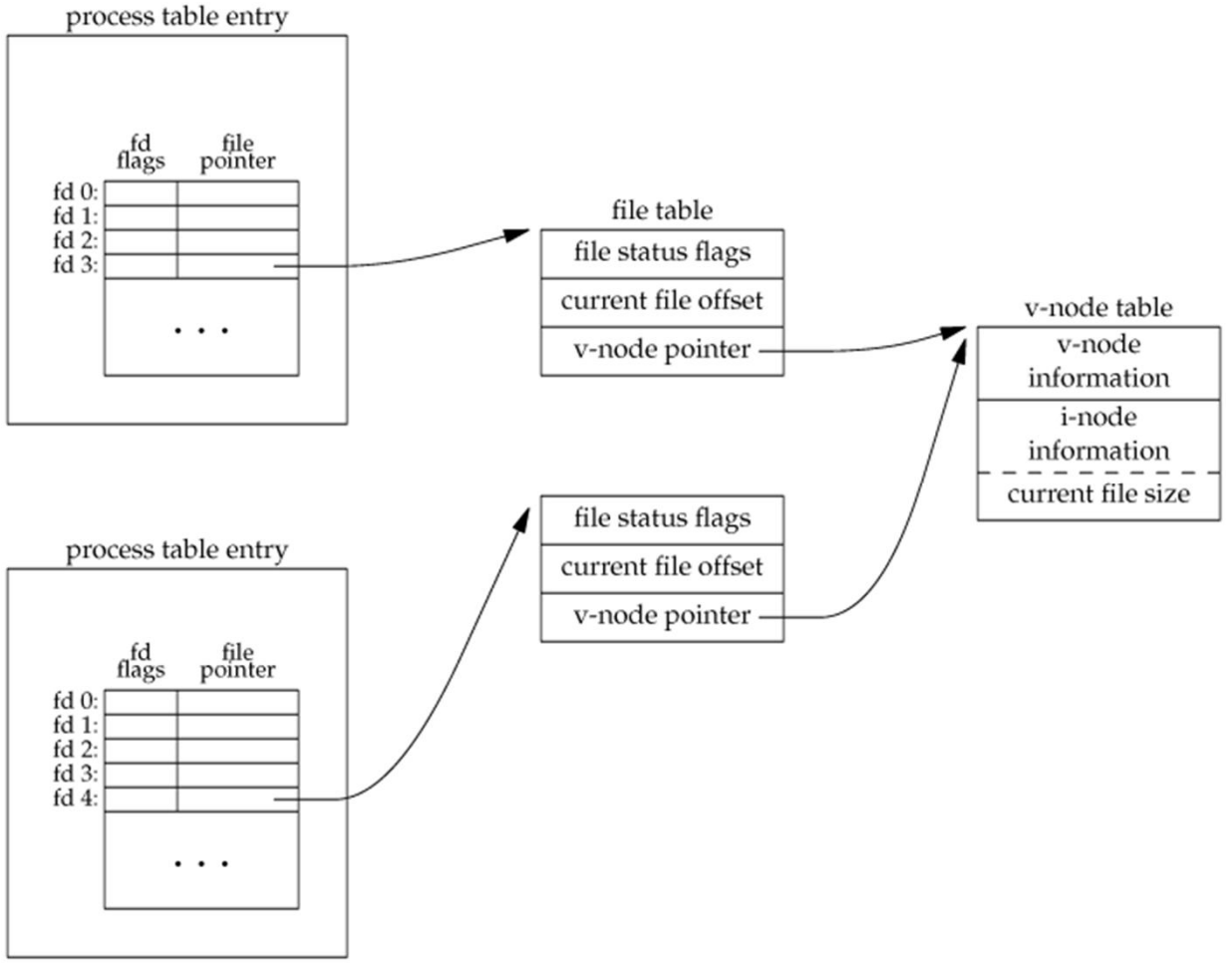
# File sharing

- kernel data structures for a single process that has two different files open.



# File sharing

- Two independent processes with the same file open



# dup( ) and dup2( )

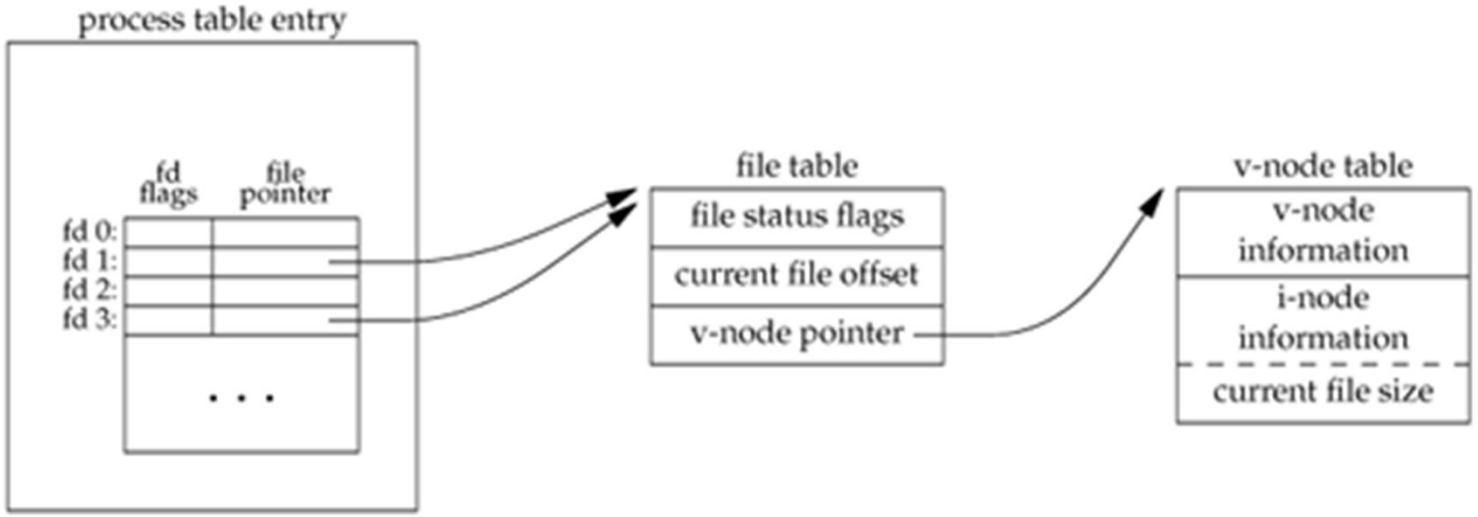
```
#include <unistd.h>

int dup(int filedes);
int dup2(int filedes, int filedes2);
                                    Both return: new file descriptor if OK, -1 on error
```

- dup
  - create a copy of *filedes* and returns a new file descriptor.
- dup2
  - makes *filedes2* be the copy of *filedes*, closing *filedes2* first if necessary.
- Return values
  - dup: the lowest numbered available file descriptor
  - dup2: the new file descriptor with the filedes2 argument

# dup( ) and dup2( )

- Kernel data structures after “dup(1)”
  - The next available descriptor is 3.

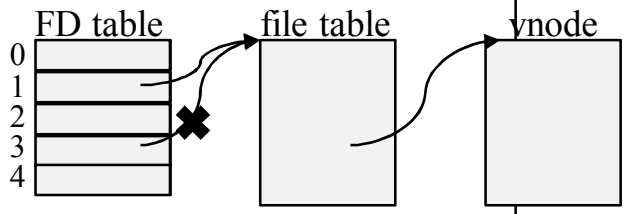


# dup( ) and dup2( )

- Example

```
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
        int fd;
        fd = creat("dup_result", 0644);
        dup2(fd, STDOUT_FILENO);
        close(fd);
        printf("hello world\n");
        return 0;
}
```




- 실 행

```
$ cat dup_result
hello world
```

# sync(), fsync(), and fdatasync()

- Delayed write
  - When write data to a file, the data is copied into buffers.
  - The data is physically written to disk at some later time.
  - ➔ 동일한 데이터에 대한 연속적인 read/write시 성능향상.

- When the delayed-write blocks are written to disk?
  - Buffer is filled with the delayed-write blocks or
  - Periodically by update daemon (usually every 30 seconds)

# sync(), fsync(), and fdatasync()

```
#include <unistd.h>

int fsync(int filedes);
int fdatasync(int filedes);
                                        Returns: 0 if OK, -1 on error
void sync(void);
```

- sync
  - Write all the modified buffer blocks to disk.
- fsync
  - Write only the modified (data + attribute) buffer blocks of a single file.
- fdatasync
  - Write only the modified data buffer blocks of a single file.

# fcntl()

```
#include <fcntl.h>

int fcntl(int filedes, int cmd, ... /* int arg */ );
                        Returns: depends on cmd if OK (see following), -1 on error
```

- Change the properties of a file that is already open
  - Duplicate an existing descriptor (cmd = F_DUPFD)
  - Get/set file descriptor flags (cmd = F_GETFD or F_SETFD)
  - Get/set file status flags (cmd = F_GETFL or F_SETFL)
  - Get/set asynchronous I/O ownership (cmd = F_GETOWN or F_SETOWN)
  - Get/set record locks (cmd = F_GETLK, F_SETLK, or F_SETLKW)

# fcntl()

- example

```
/* include header files 생략 */

int main()
{
   int mode, fd, value;

   fd = open("test.sh", O_RDONLY|O_CREAT);
   value = fcntl(fd, F_GETFL, 0);

   mode = value & O_ACCMODE;
   if (mode == O_RDONLY)
      printf("O_RDONLY setting\n");
   else if (mode == O_WRONLY)
      printf("O_WRONLY setting\n");
   else if (mode == O_RDWR)
      printf("O_RDWR setting\n");

}
```

# fcntl()

- 실 행

```
$ ./fgetfl_test
O_RDONLY setting
$
```