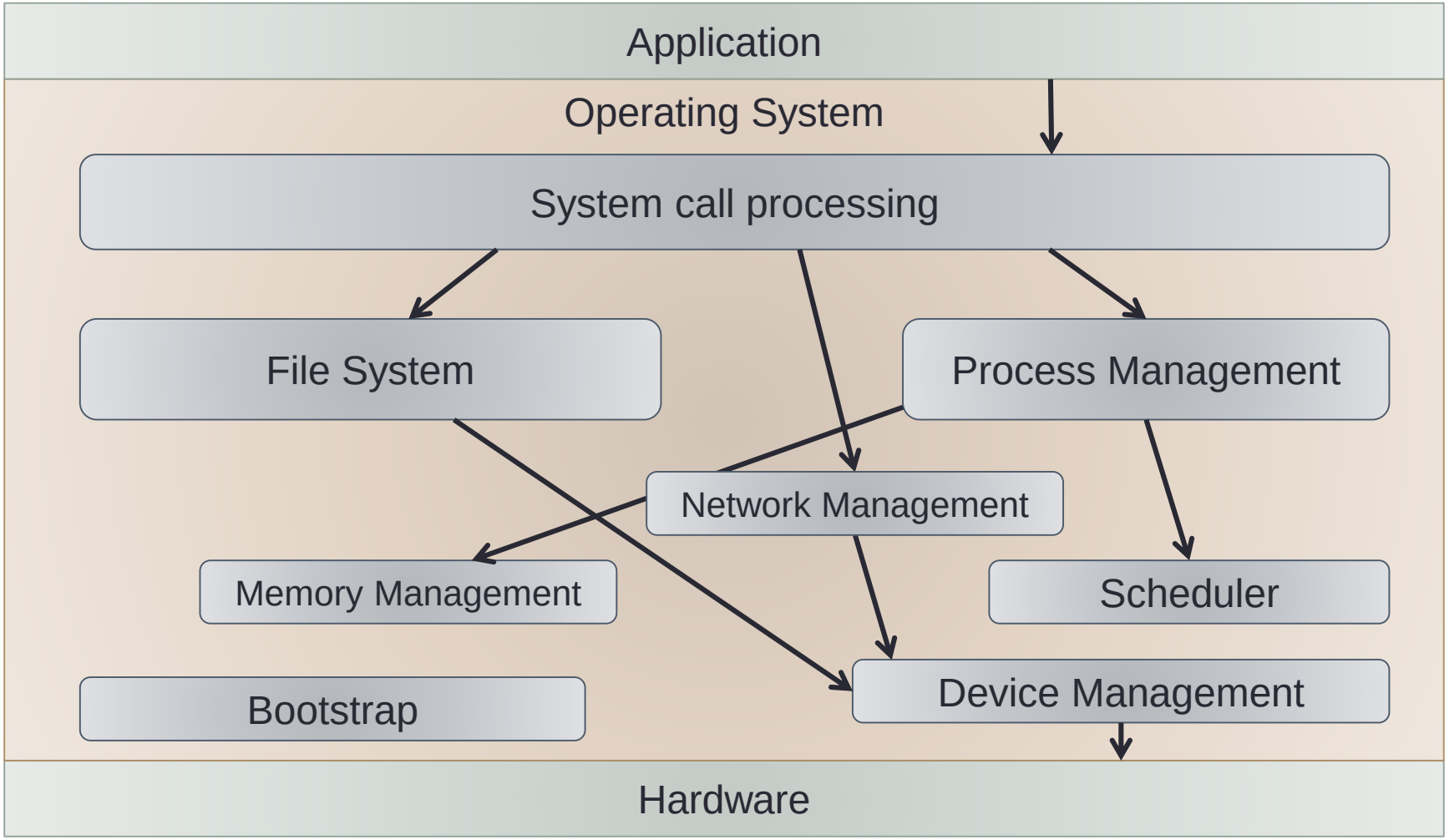
SOFTWARE ARCHITECTURE
2. FILE SYSTEM

Tatsuya Hagino
hagino@sfc.keio.ac.jp

lecture URL

https://vu5.sfc.keio.ac.jp/slide/

Operating System Structure



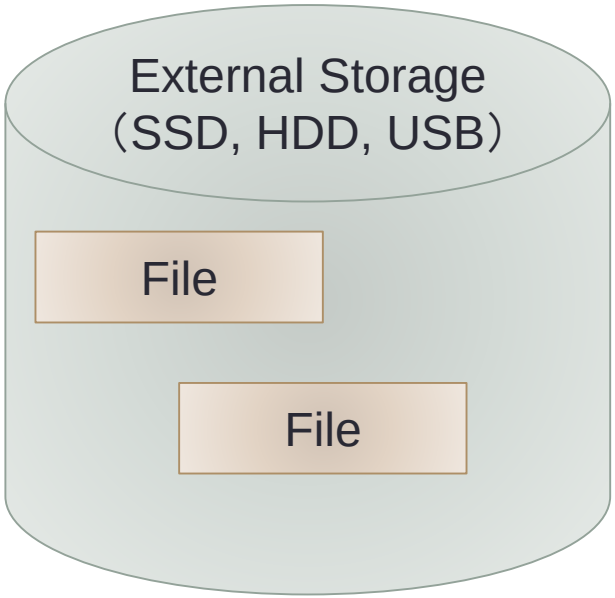
What is a File?

- A unit to store information in an external storage
  - sometimes called dataset

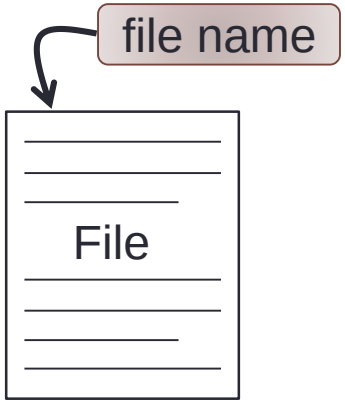
- Characteristics of a file
  - Information in a file is non-volatile.
    - Its content does not disappear when power is off.
  - Information in a file is persistent.
    - It exists forever.
    - You can start with what you have left.

- File System
  - Manage files on an external storage
  - Windows → NTFS
  - MacOS → HFS or APPLE FS
  - Unix → UFS

- File related conventions:
  - file name
  - file structure
  - file type
  - file access method



File Naming Convention



- Each file has a name.
  - File name
- Letters used for file name
  - case insensitive: lower and upper letters are not distinguished.
  - case sensitive: lower and upper letters are distinguished.
  - Encoding of non alphabet characters (e.g. kanji file name)
- Length of file name
  - MS-DOS limits 8+3 characters.
  - UNIX limits 255 characters.
- File extension
  - It may express its file type:
    - Windows: use it to find associated program.
    - Mac: uses to hold file type internally.
    - UNIX: depends on applications
      - Compilers may use it to select programming language.

file name

document.docx

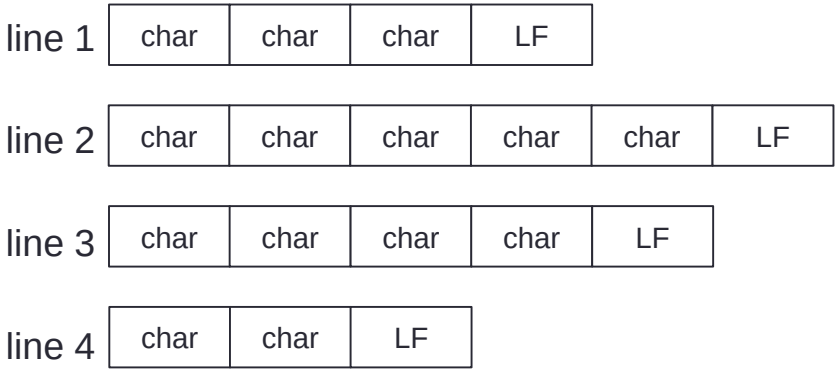
file extension

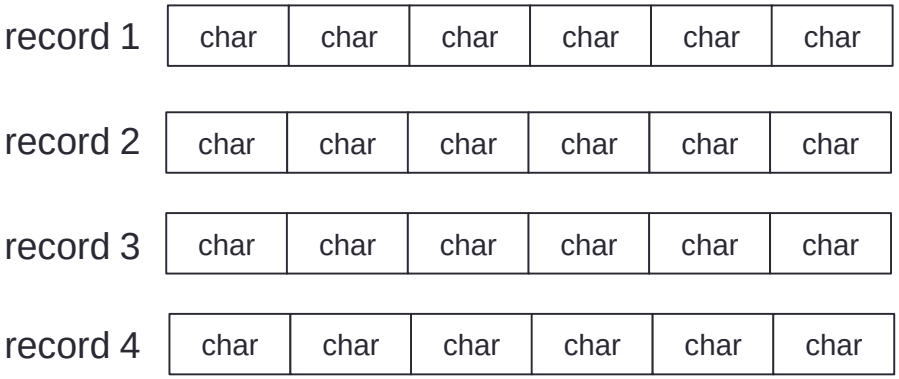
File Structure

- Sequence of bytes
  - No structure in a file
  - User can create arbitrary fields.
  - A text file uses LF or CR to separate lines.
  - May not be efficient when reading or writing.

vs

- Sequence of records
  - Consist of fixed length records.
  - For a punch card, each record consists of 80 characters. (i.e. one record = one line)
  - Efficiently read and write records.





File Type

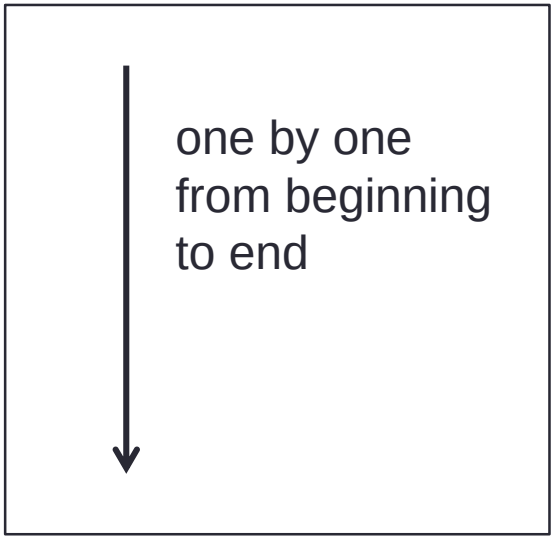
- Regular File:
  - text file or binary file
- Directory:
  - Folder
  - Manage a set of files
- Character Special File:
  - Input/output device
  - Serial devices like terminal, printer, network, etc.
- Block Special File:
  - Devices with block access (i.e. read/write blocks)
  - HDD, SSD, etc.
  - File system can be created on it.

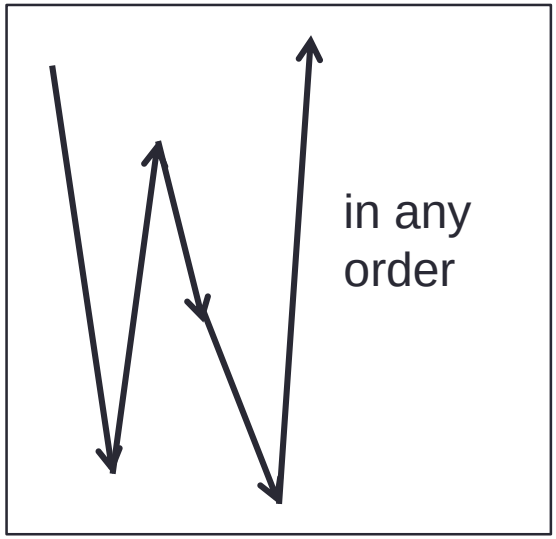
File Access Method

- Sequential access
  - Read/write a file one byte by one byte sequentially.
  - Cannot skip or change the order of reading/writing.
    - May rewind it and start from the beginning again.

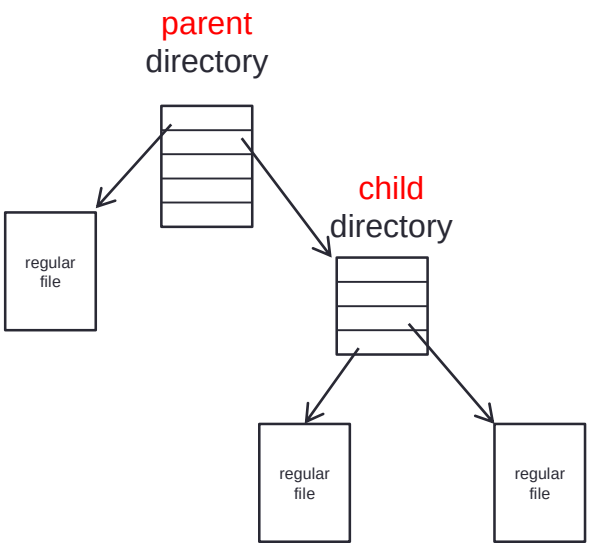
vs

- Random access
  - Read/write a file randomly in any order.
  - In UNIX, the next position can be specified using seek system call.



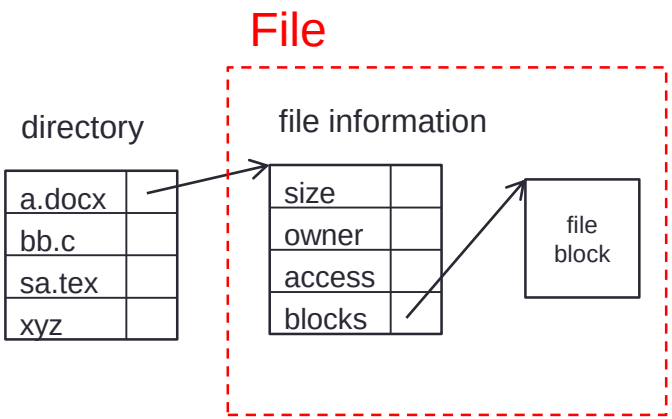


Hierarchical File System



- Implemented by allowing a directory to have other directories inside.

- Implementation of a directory:
  - Each directory consists of a list of entries.
  - Each entry consists of a file name and a pointer to a file.
    - A file can be pointed from more than one directory.
    - File sharing (hard link)



- Specifying a file
  - Use path name
  - Concatenate file names with separation characters
    - UNIX and Mac → /
    - Windows → \ or ¥
    - Old Mac OS → :

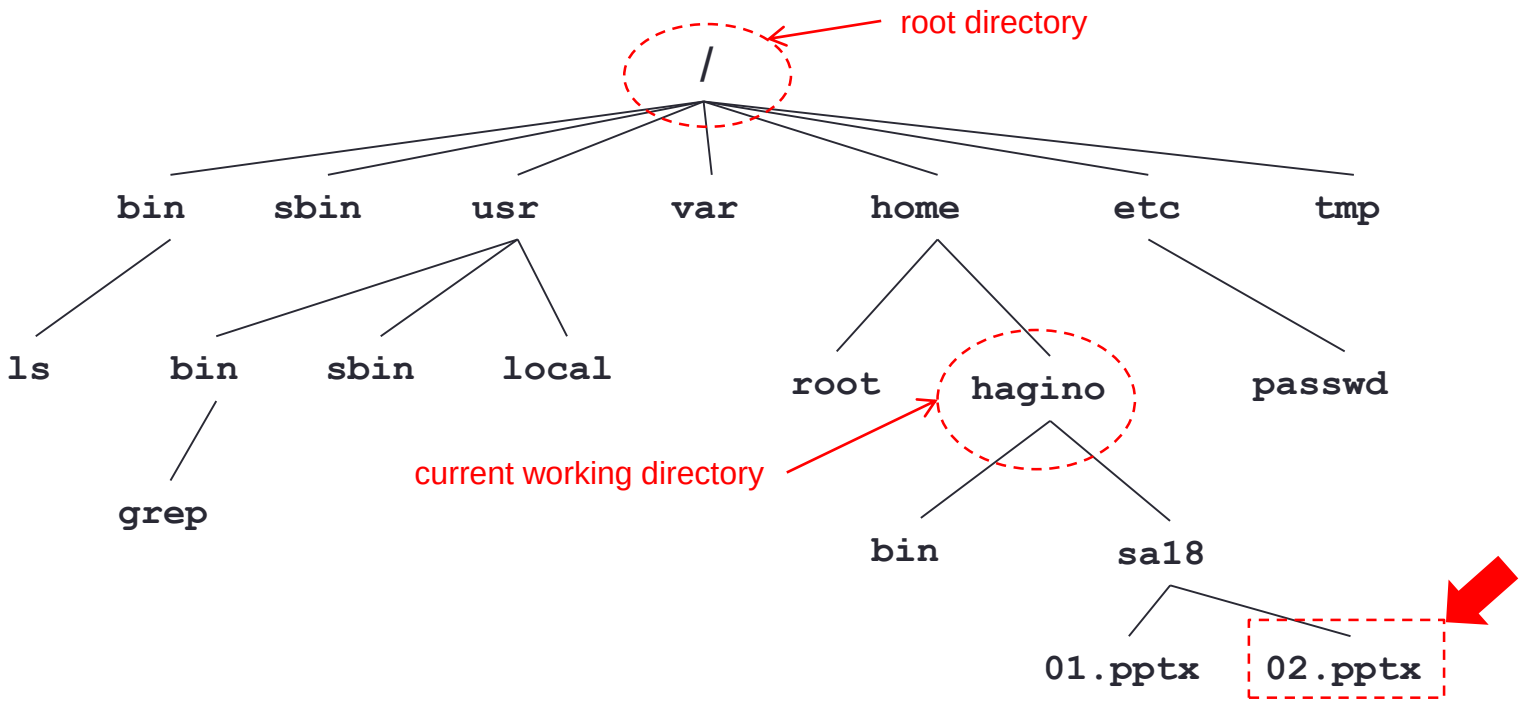
Path Name

- Absolute path name
  - Starting from the *root directory*, sub directory names are listed with / separators.

/home/hagino/sa18/02.pptx

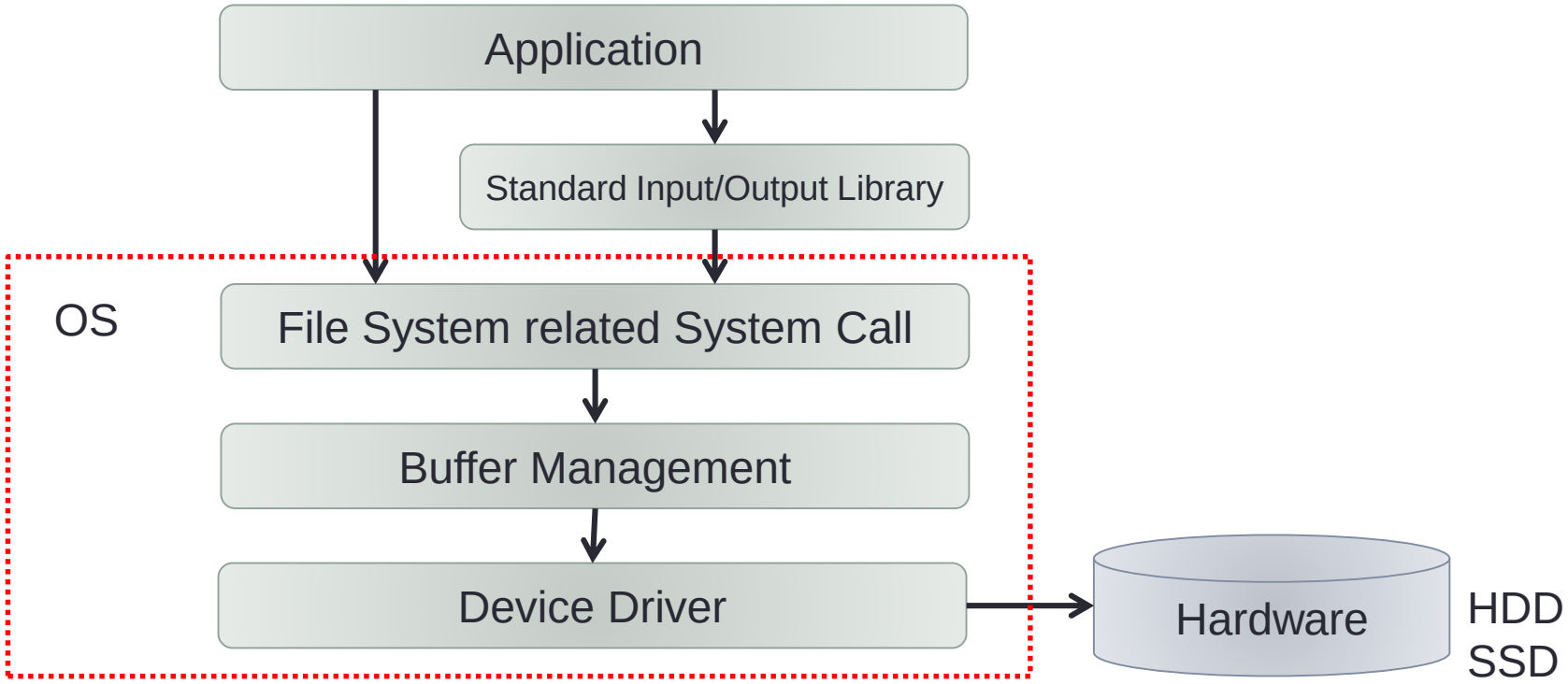
- Relative path name
  - Starting from the *current working directory*

sa18/02.pptx

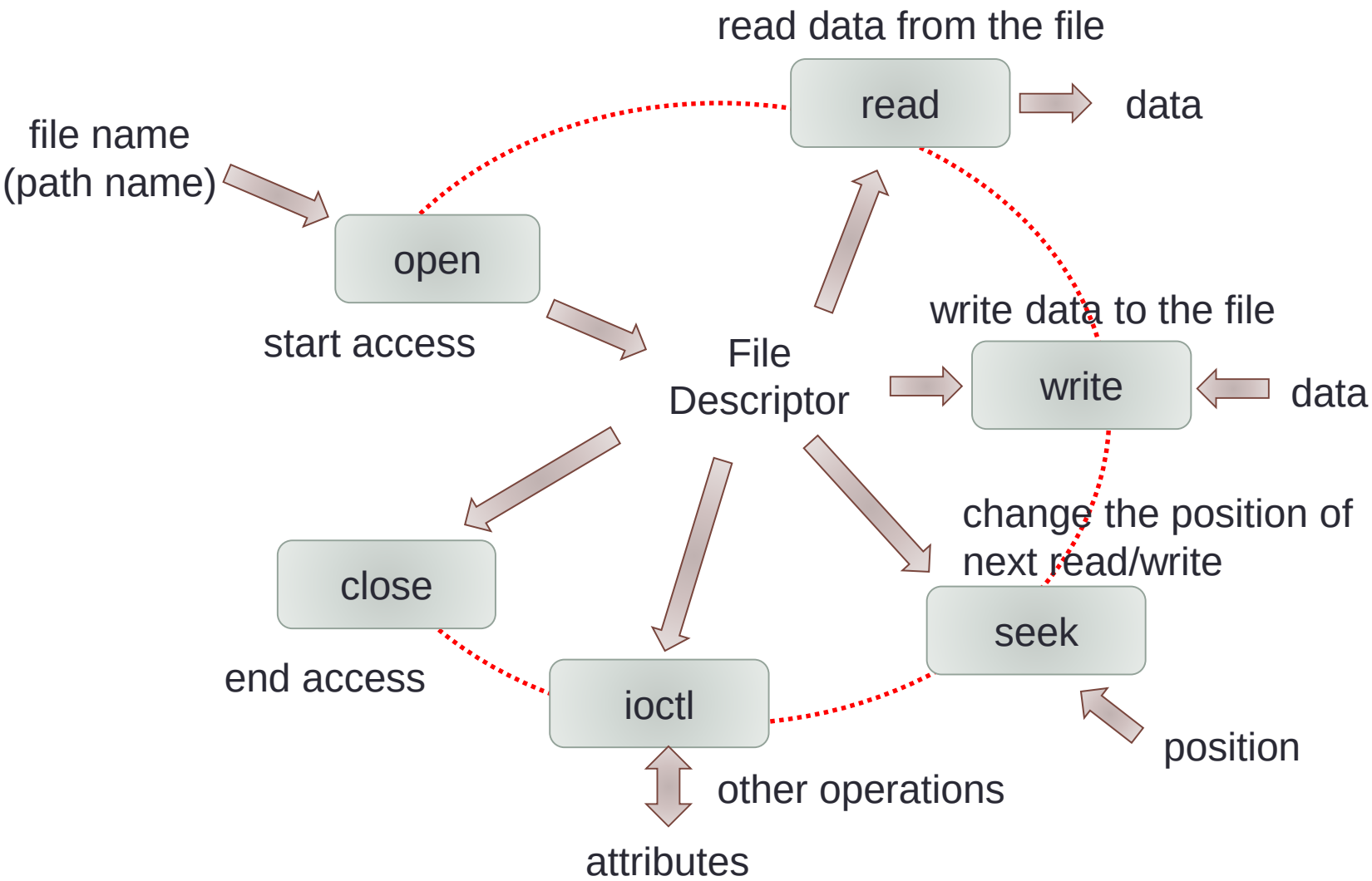


File Read/Write Mechanism

- UNIX as an example
  - Directly use system calls to read/write a file
  - Use standard input/output libraries



File System related System Calls



File Descriptor

- A small number returned by open system call
  - specifies an opened file
  - used by read/write
  - remembers a position of read/write

- Managed by each process
  - The meaning of file descriptors is local to each process.
  - Even in a single process, if the same file is opened more than once, different file descriptors are assigned.

- Special file descriptors:
  - standard input → 0
  - standard output → 1
  - standard error output → 2

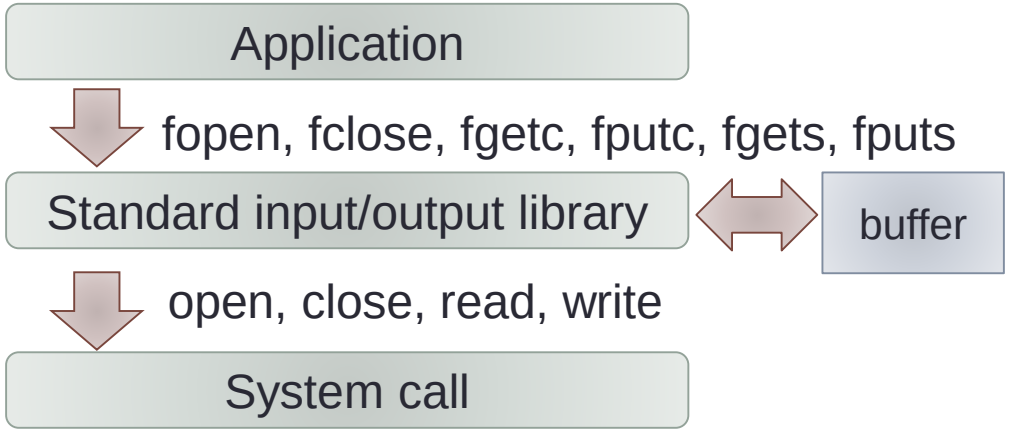
Example program of reading a file using system calls

```
#include <fcntl.h>

int main(argc, char *argv[])
{
  int fd, n;
  char buf[512];

  fd = open(argv[1], O_RDONLY);
  while ((n = read(fd, buf, 512)) > 0) {
    write(1, buf, n);
  }
  close(fd);
}
```

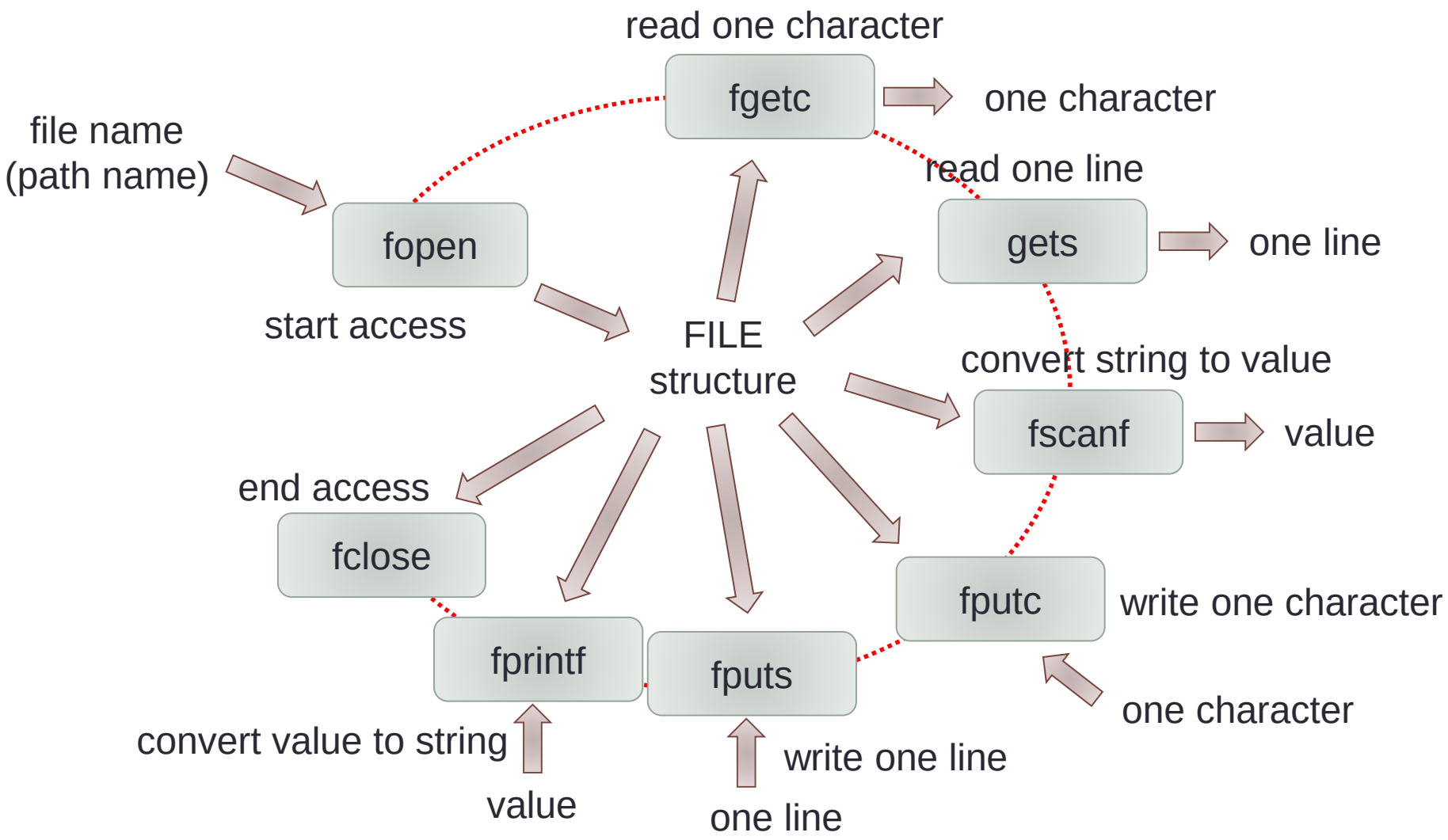
Standard Input/Output Library

- System call
  - Not easy to use
    - e.g. There is no system call to read one line.
  - Inefficient for small usage
    - System call needs to go though OS protection boundary (user mode to supervise mode).
    - Costs much more than calling a library.

- Standard input/output library
  - Useful interface
    - read one line
    - read one character
  - Optimize system call access
    - Use buffer



How to Use Standard Input/Output Libraries



Example program of reading a file using standard input/output libraries

```
#include <stdio.h>

int main(argc, char *argv[])
{
  FILE fp;
  int ch;

  fp = fopen(argv[1], "r");
  while ((ch = fgetc(fp)) >= 0) {
    fputc(ch, stdout);
  }
  fclose(fp);
}
```

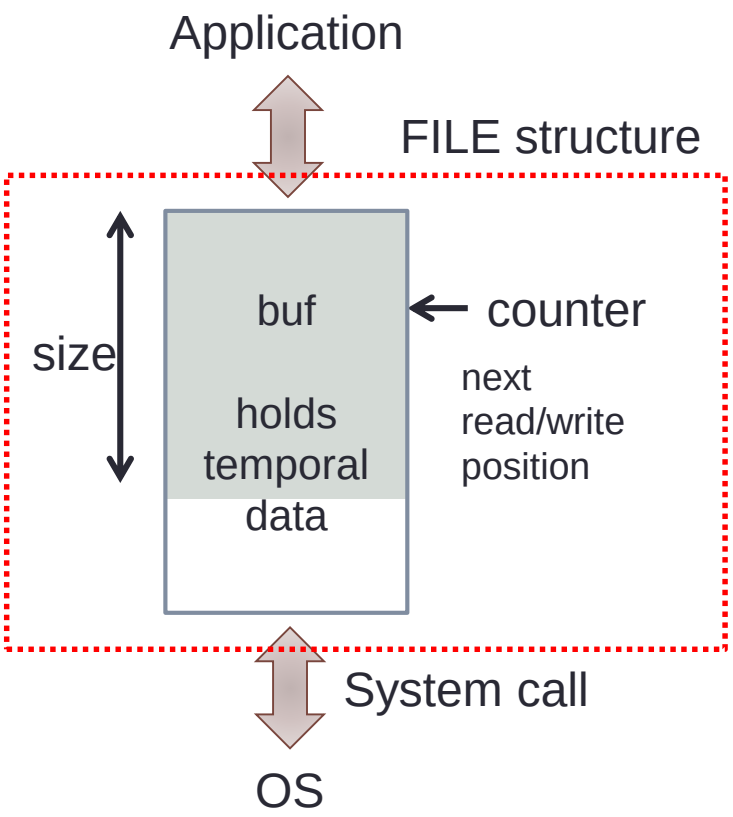
Implementation of Standard Input/Output Library

fopen

```
FILE *fopen(char *path, char *mode) {
  FILE *fp;
  fp = (FILE *)malloc(sizeof(struct FILE));
  fp->fd = open(path, ...);
  fp->size = 0;
  fp->counter = 0;
  return fp;
}
```

FILE structure

```
struct FILE {
  int fd;
  char buf[BUFSIZE];
  int size;
  int counter;
};
typedef struct FILE FILE;
```

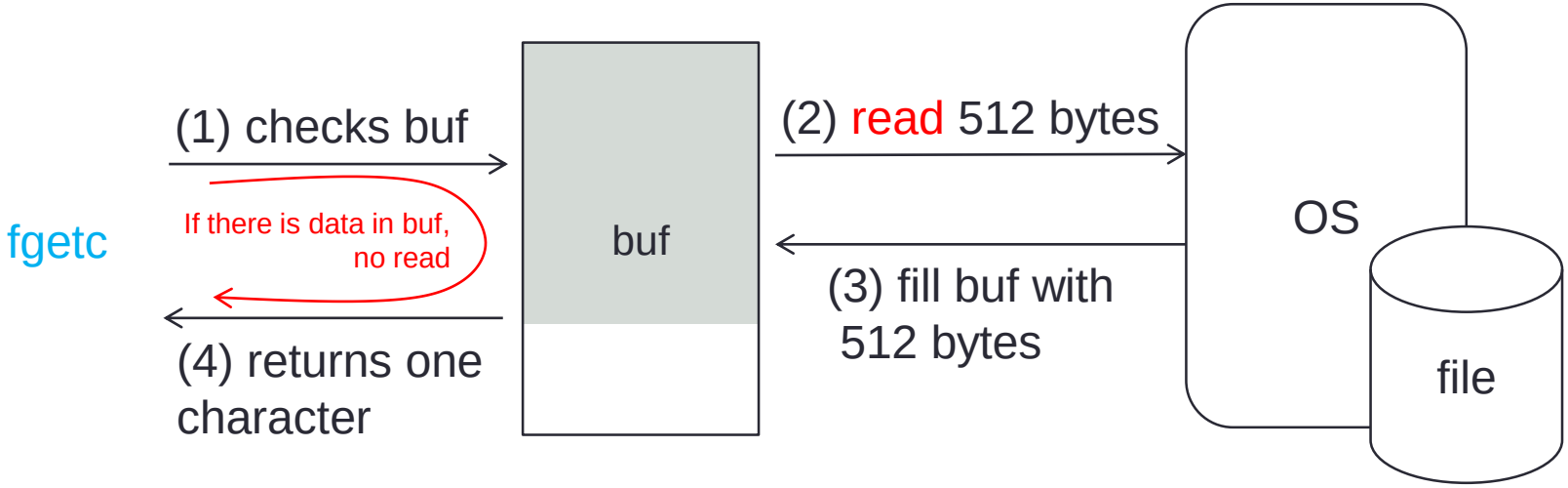


Implementation of fgetc

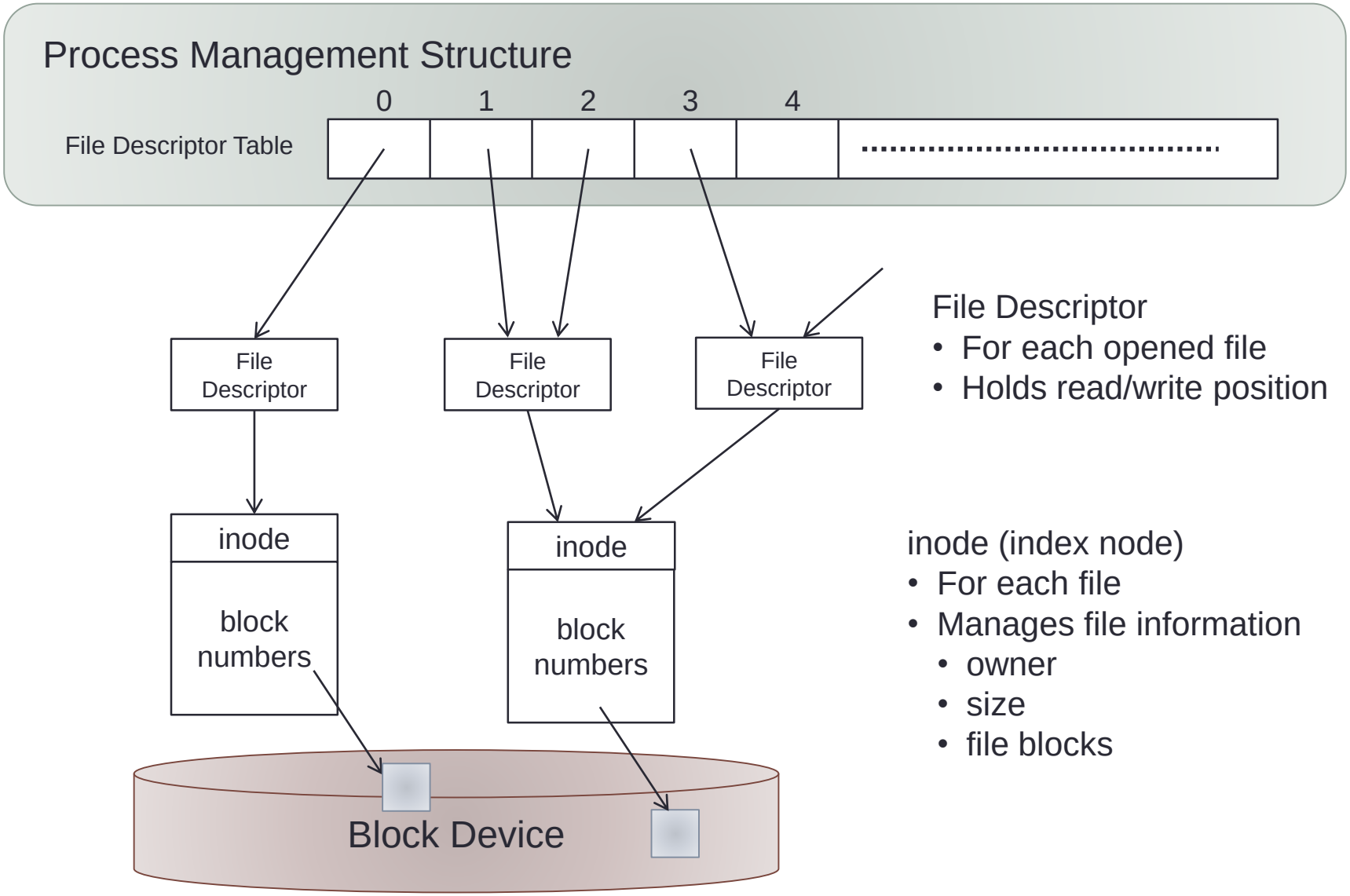
fgetc

```
int fgetc(FILE *fp) {
  if (fp->counter >= fp->size) {
    fp->size = read(fp->fd, fp->buf, BUFSIZE);
    if (fp->size <= 0) return -1;
    fp->counter = 0;
  }
  return fp->buf[fp->counter++];
}
```

assume
BUFSIZE = 512



Implementation of System Call



File Descriptor
- For each opened file
- Holds read/write position

inode (index node)
- For each file
- Manages file information
  - owner
  - size
  - file blocks

Implementation of open

- Returns a file descriptor for a given file
- Calls namei to find its inode.
- Finds an empty file descriptor slot and sets a new file structure.

```
int open(char *path, int flags, ...) {
  struct inode *ip;
  int fd;
  struct file *fp;
  ip = namei(path);
  if (ip) {
    fp = create a new file structure;
    fp->inode = ip;
    fp->offset = 0;
    fp->refcount = 1;
    fd = unused file descriptor of proc;
    proc->file[fd] = fp;
    return fd;
  }
  else return -1;
}
```

file structure

```
struct file {
  struct inode *inode;
  long offset;
  int refcount;
};
```

namei

- Following the path, checks each directory to fined the named file.

```
struct inode *namei(char *path) {
  struct inode *dp;
  if (*path == '/') {
    dp = proc->root_directory;
    path++;
  } else dp = proc->current_working_directory;
  while (*path) {
    name = select next name from path;
    dp = lookup(dp, name);
  }
  return dp;
}
```

Implementation of File

- Each file consists of a list of blocks on a disk (or a block device).
- UNIX uses inode (index node) to manage block numbers
  - FAT file system uses FAT to manage

inode structure

```
struct inode {
  short mode;
  int uid
  int gid;
  long size;
  ...
  int atime;
  int mtime;
  int ctime;
  ...
  int direct[12];
  int indirect[3];
};
```

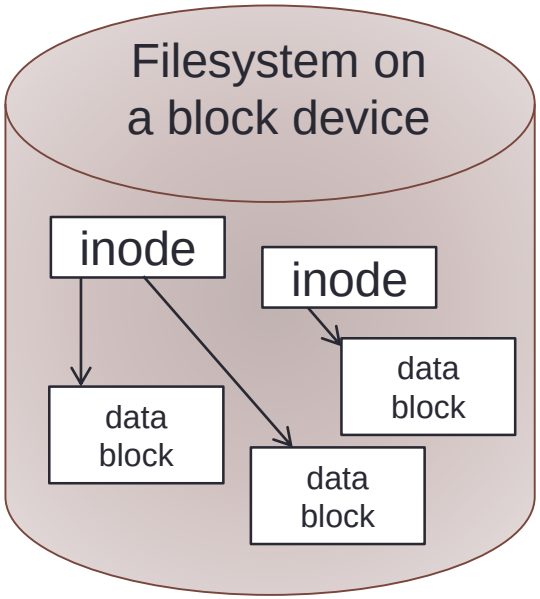
owner

size

modification time

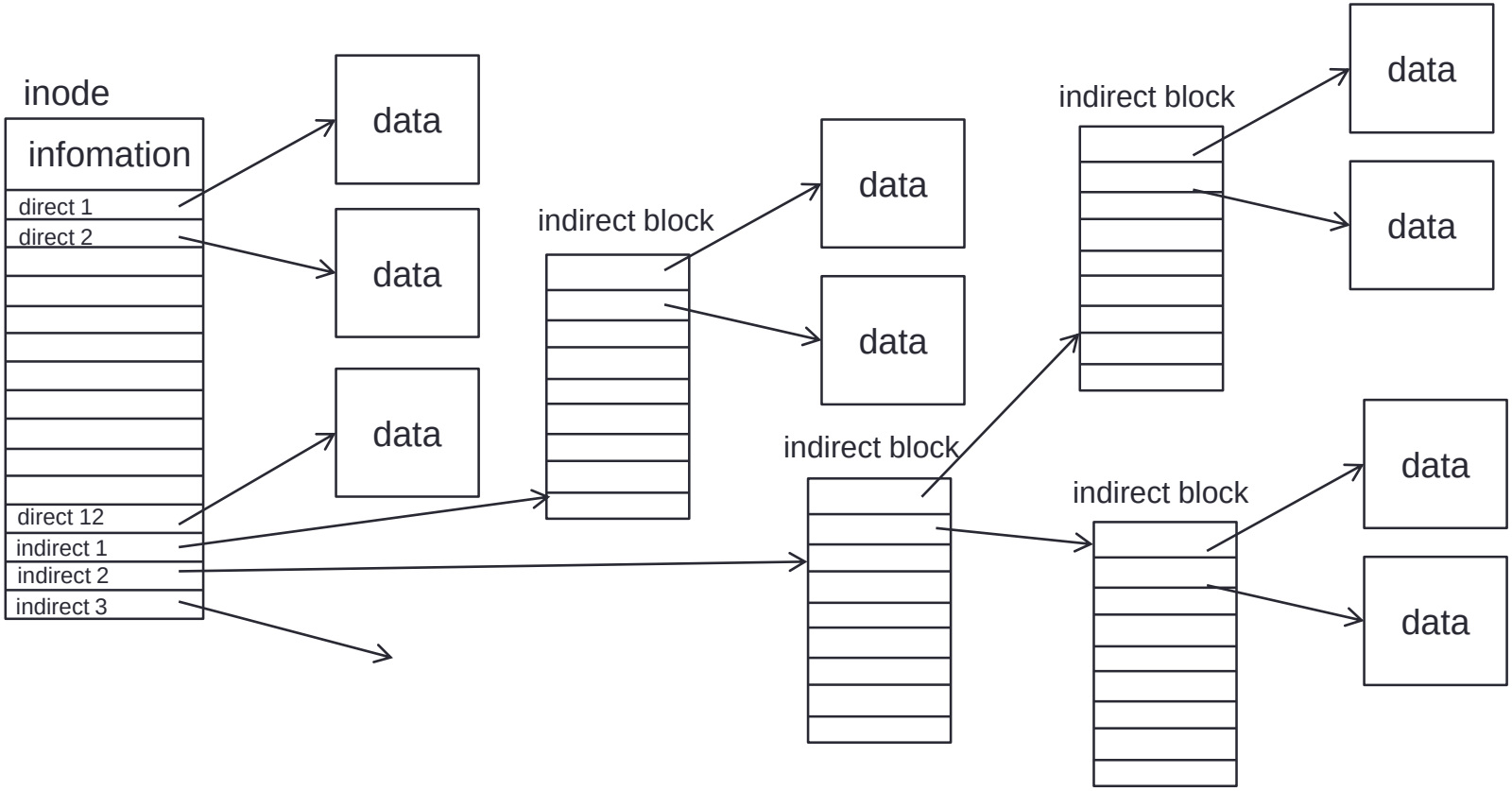
direct reference block numbers（12 blocks）

indirect block numbers（3 blocks）



Direct and Indirect Blocks

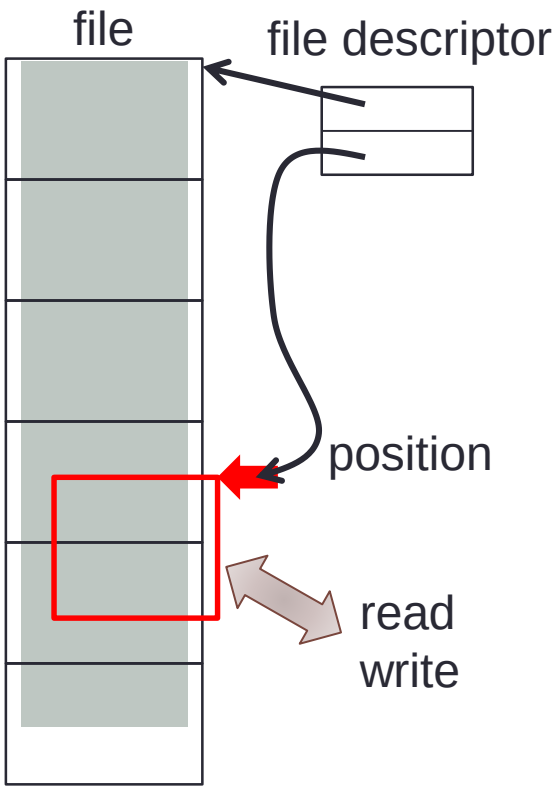
- Each inode points 12 direct blocks.
  - When one block is 512 bytes, it can hold a file less than 512 × 12=6KB.
- An indirect block contains a list of blocks.



Calculation of Block Number

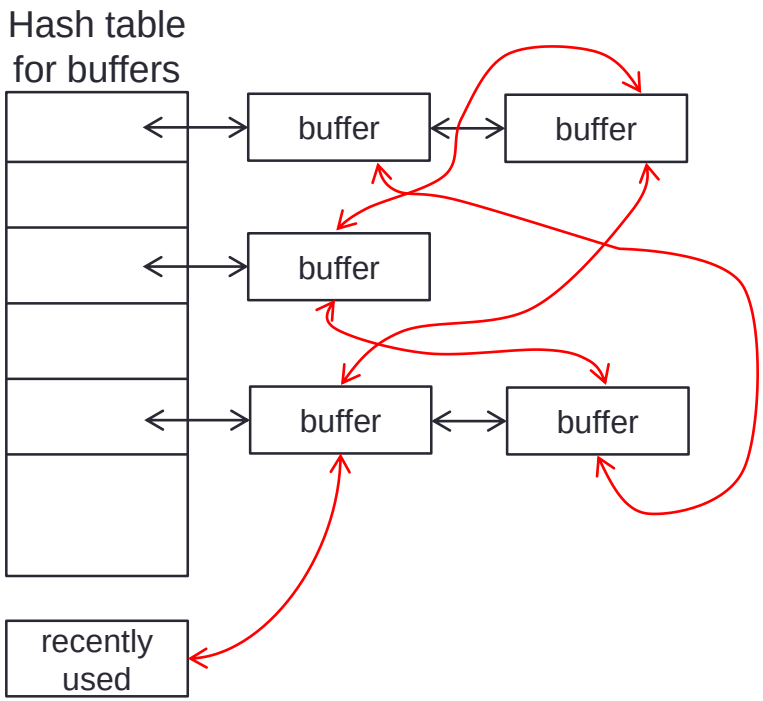
- From file position to calculate corresponding block number
  - A file descriptor holds a position of read/write and the next read/write access the corresponding block.

```
int balloc(struct inode *ip, long offset) {
  struct buf *bp;
  int i;
  blk = (offset + BLKSIZE - 1) / BLKSIZE;
  if (blk < 12) return ip->direct[blk];
  blk -= 12;
  blocks = BLKSIZE/sizeof(int);
  for (i = 0; i < 3; i++) {
    if (blk < blocks) break;
    blk -= blocks;
    blocks *= BLKSIZE/sizeof(int);
  }
  bp = getblock(ip->indirect[i])
  while (i-- > 0) {
    blocks /= BLKSIZE/sizeof(int);
    bp = getblock(bp->buf[blk/blocks]);
    blk %= blocks;
  }
  return bp->buf[blk];
}
```



Buffering

- Disk blocks are cached in OS memory.
- Read:
  - First time: reads data from the disk and put it in the cache.
  - Second time: do not access disk but use data in the cache.
- Write:
  - Do not write data to the disk immediately.
  - Write them out all later.

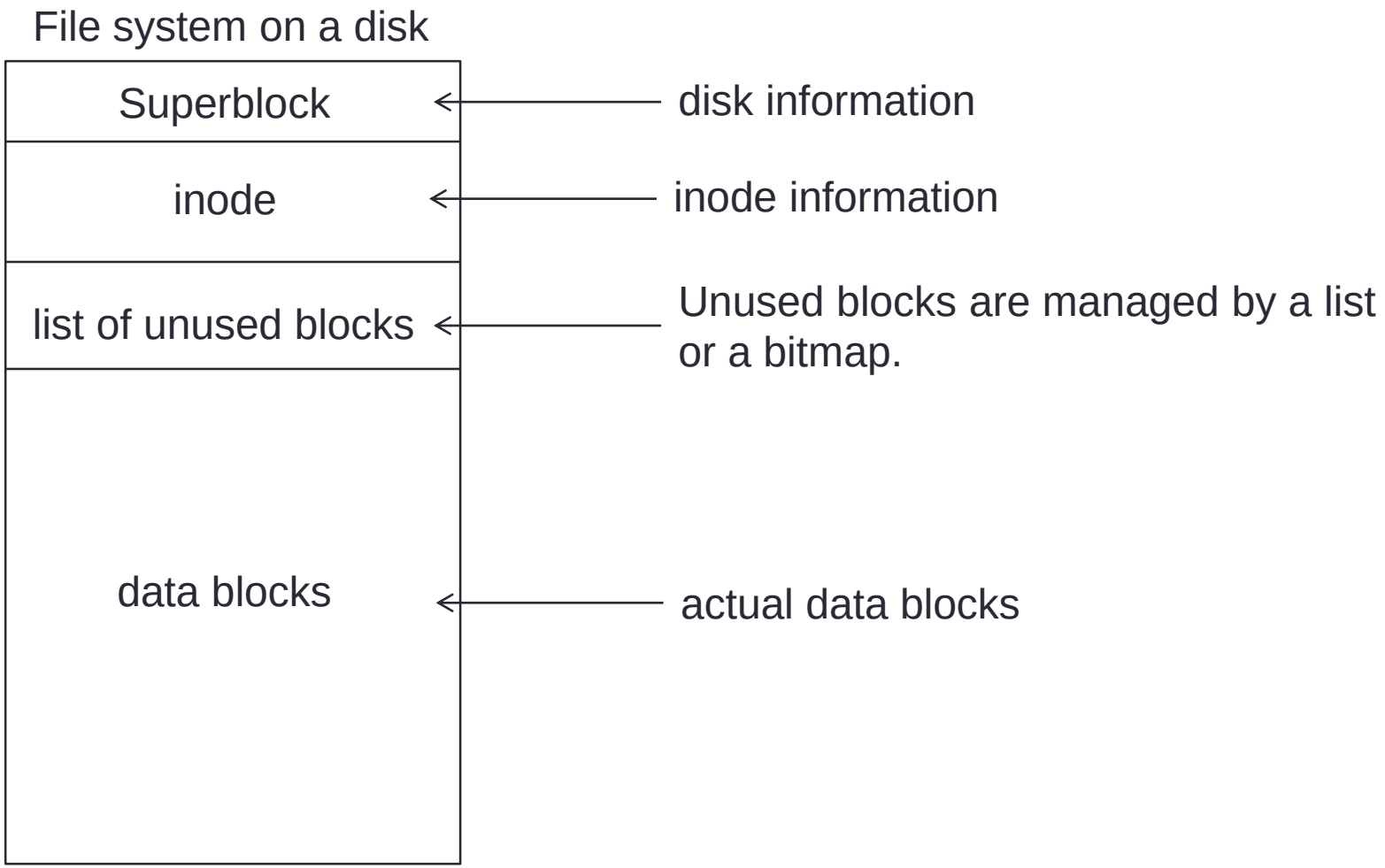


buf structure

```
struct buf {
  struct buf *next, *prev;
  struct buf *queue_next, *queue_prev;
  struct inode *inode;
  int dirty;
  int blkno;
  char buf[BLKSIZE];
};
struct buf buffers[HASH];
```

Disk Format

- A new disk needs to be formatted.



Summary

- File System
  - UNIX file system as an example.
- System call vs Standard Input/Output Library
  - Implementation of the library
  - File Descriptor
- Implementation of File System
  - inode
  - Direct and indirect blocks