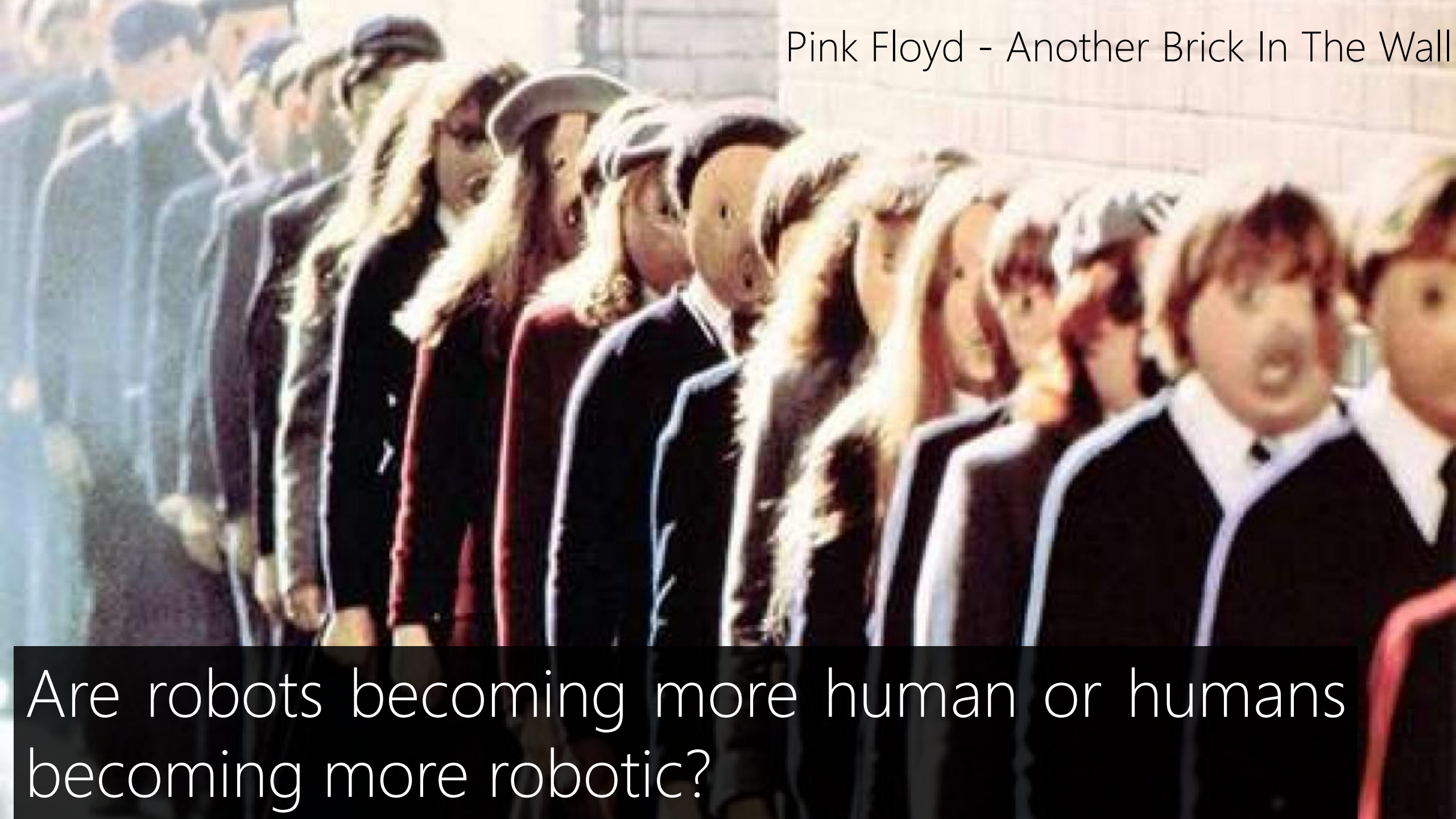
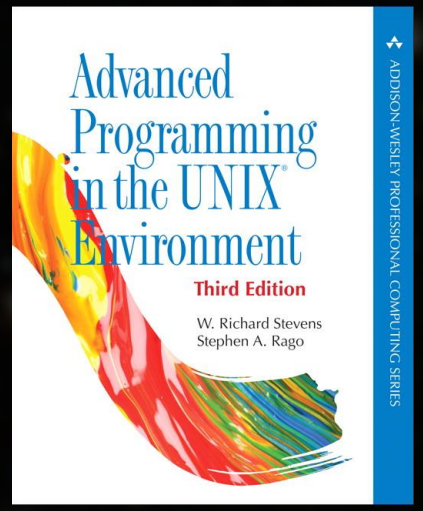
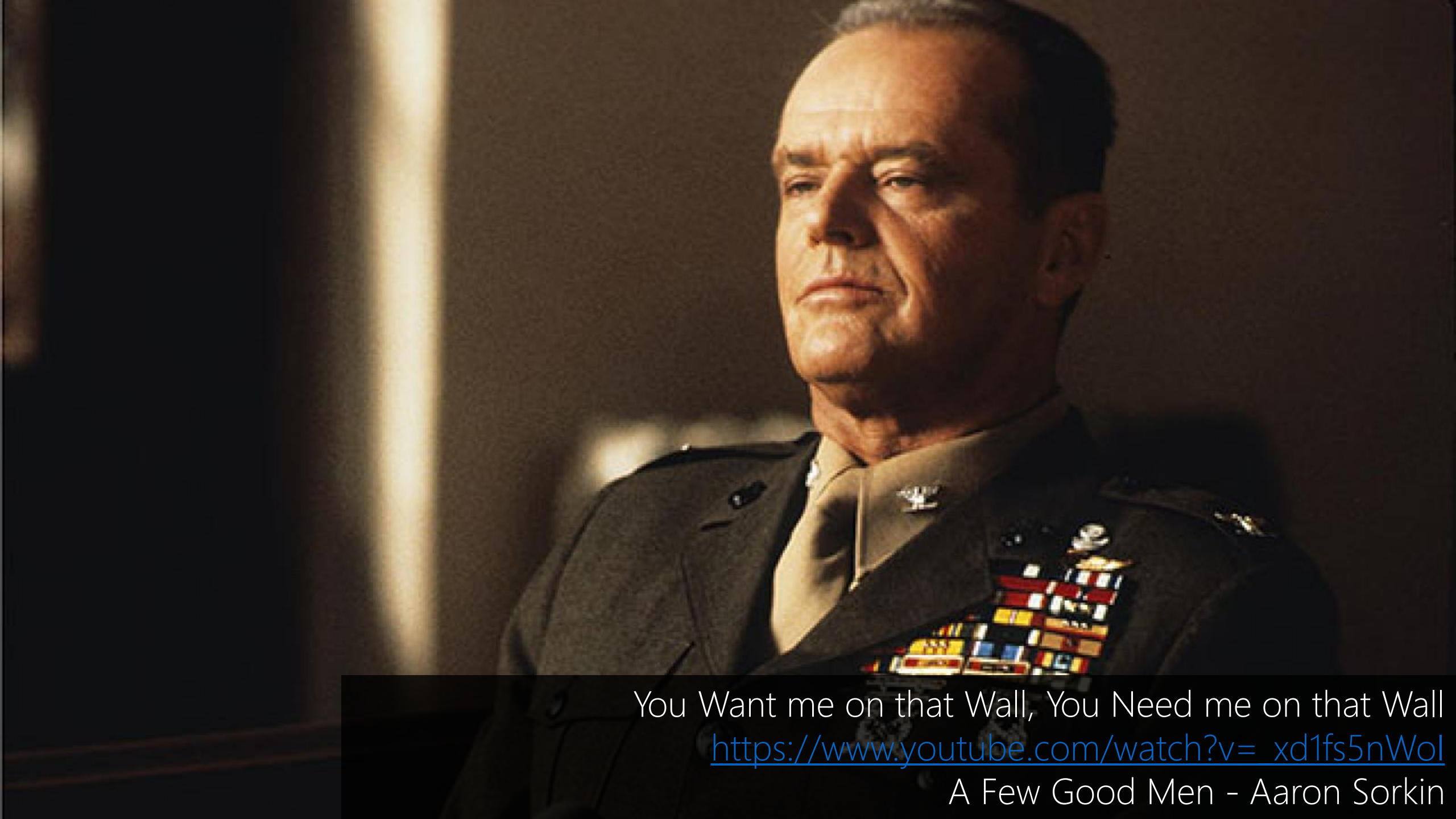
1-Week Extension to Lab06, Lec06



Are robots becoming more human or humans becoming more robotic?



Chapter 04: File I/O



You Want me on that Wall, You Need me on that Wall

https://www.youtube.com/watch?v=_xd1fs5nWoI

A Few Good Men - Aaron Sorkin

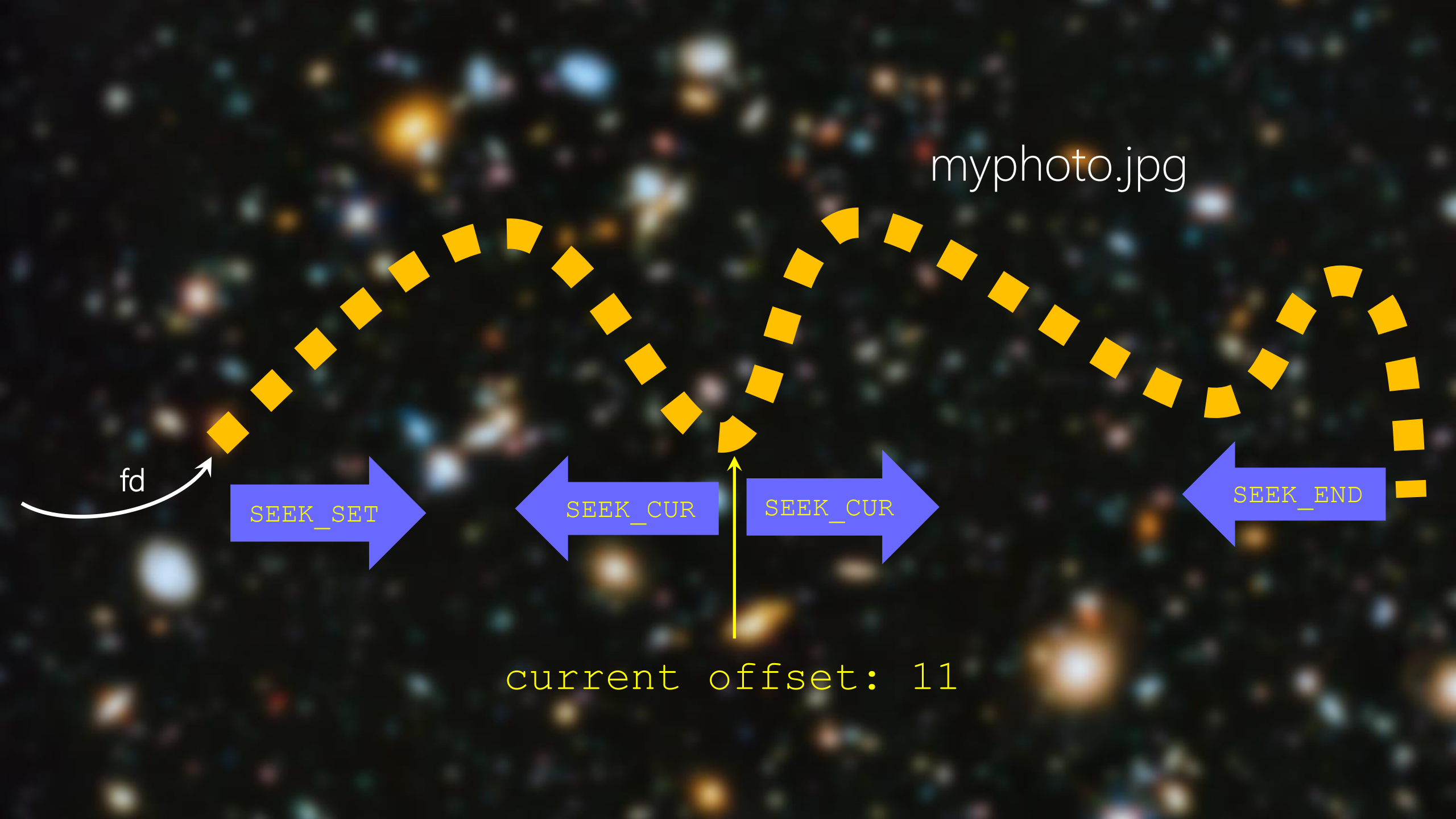
lseek

POSIX

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error


myphoto.jpg
fd
SEEK_SET
SEEK_CUR
SEEK_CUR
SEEK_END
current offset: 11

```
hfani@charlie:~$ vi hole.c
```

```
#include <fcntl.h>
#include <unistd.h>
void main(void){
        int fd = open("./hole_test.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
        int cur_offset = lseek(fd, 10, SEEK_SET);
        char buf[20] = "write after the hole.";
        write(fd, buf, 20);
}
~
```

current offset = 0

move it 10 bytes ahead from start

write new bytes

```
hfani@charlie:~$ cc hole.c -o hole
hole.c: In function ‘main’:
hole.c:6:17: warning: initializer-string for array of ‘char’ is too long
    6 |  char buf[20] = "write after the hole.";
      |                 ^~~~~~~~~~~~~~~~~~~~~~~
hfani@charlie:~$ ./hole
hfani@charlie:~$ vi hole_test.txt
```

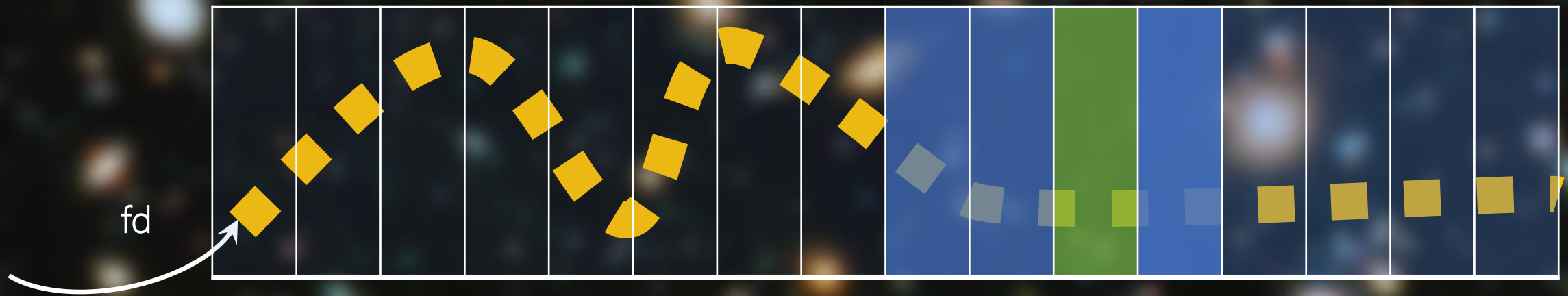
```
^@^@^@^@^@^@^@^@^@^@write after the hole
~
```

```
hfani@charlie:~$ hexdump hole_test.txt
0000000 0000 0000 0000 0000 0000 7277 7469 2065
0000010 6661 6574 2072 6874 2065 6f68 656c
000001e
```

```
hfani@charlie:~$ od -c hole_test.txt
0000000  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0   w   r   i   t   e
0000020   a   f   t   e   r       t   h   e       h   o   l   e
0000036
```

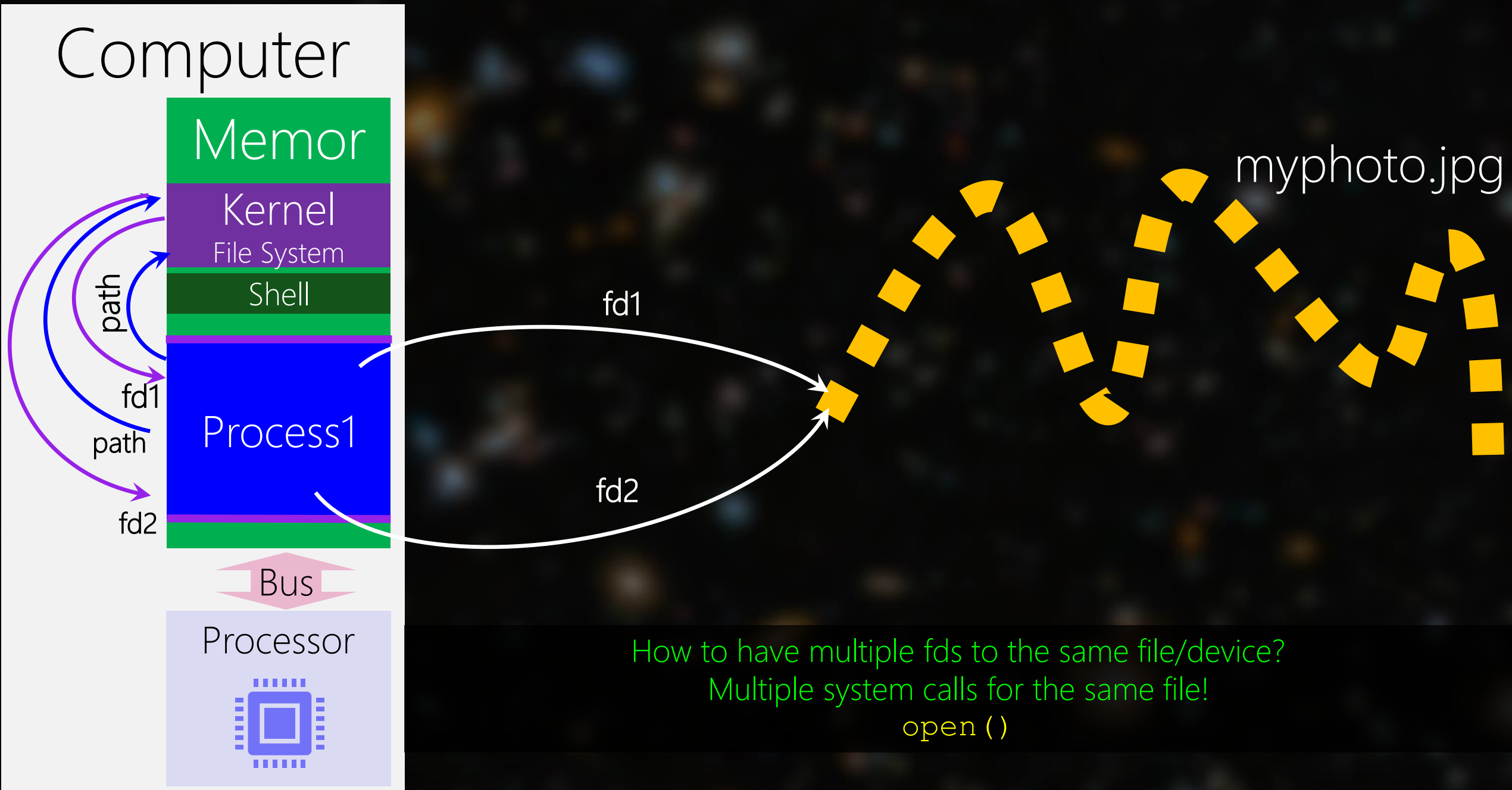
lseek

A Better User Case: Binary Search

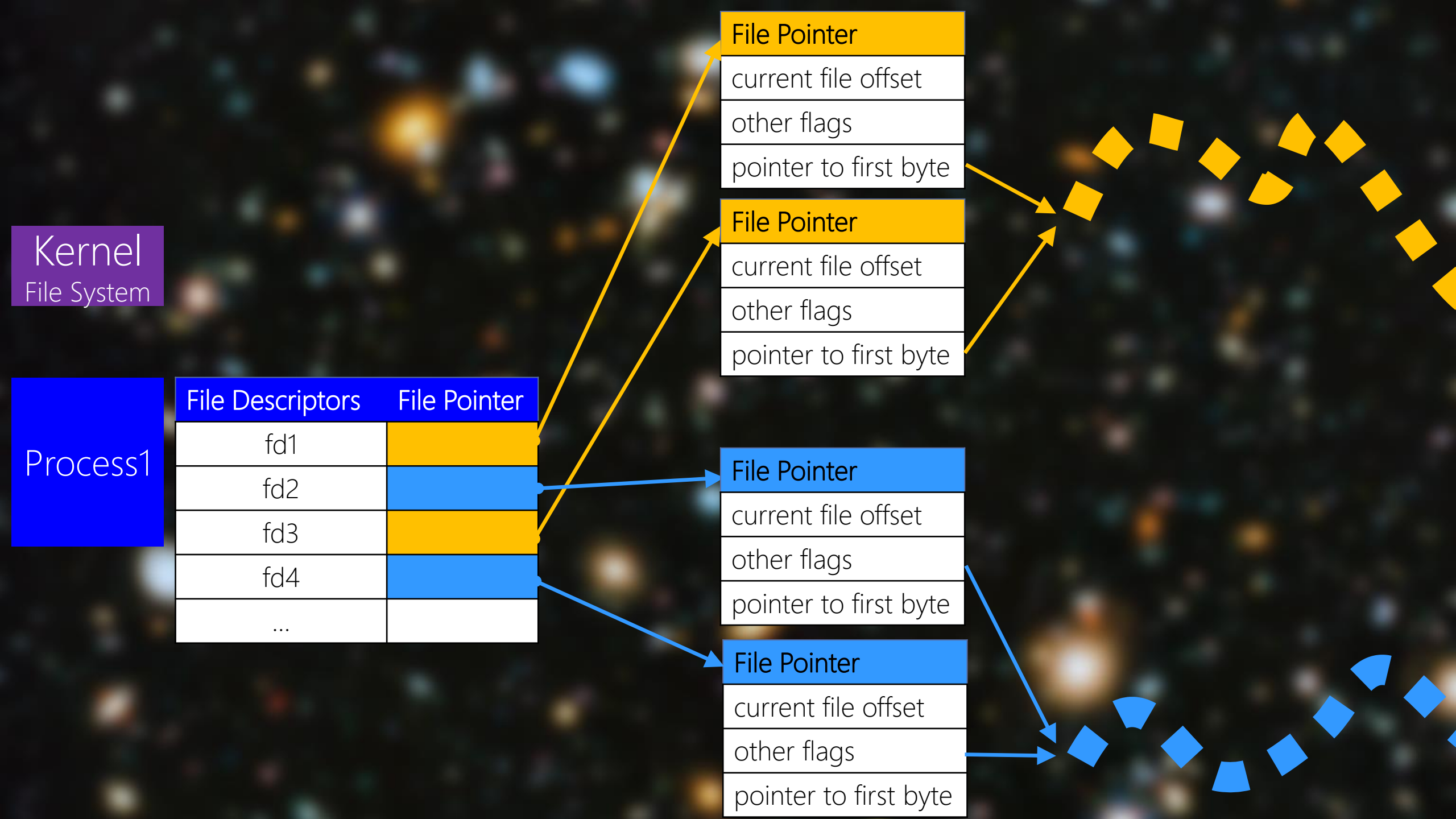


`dup` and `dup2`

File System: High-Level



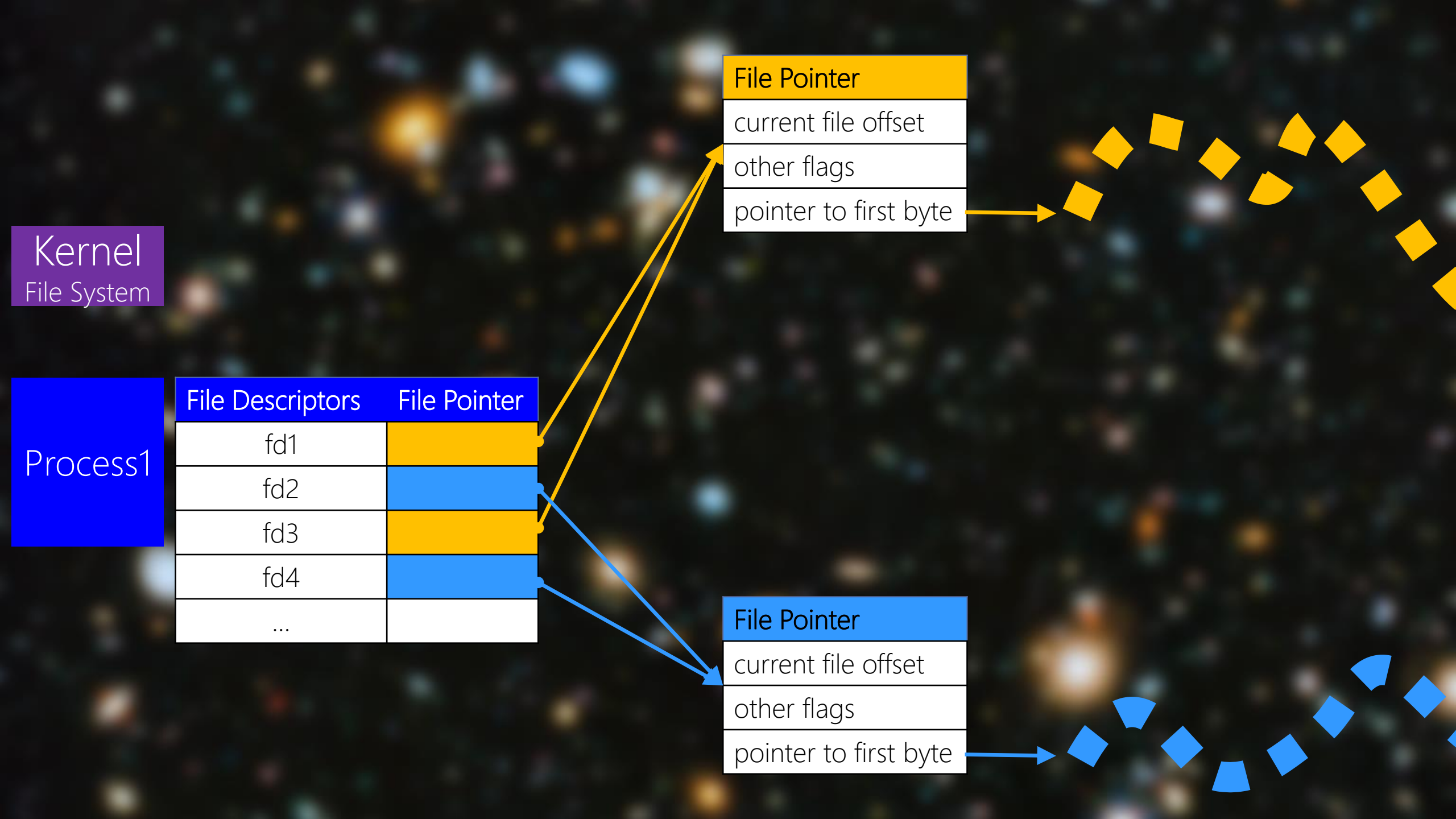
How to have multiple fds to the same file/device?
Multiple system calls for the same file!
`open()`


Kernel
File System
Process1
File Descriptors
File Pointer
fd1
fd2
fd3
fd4
...
File Pointer
current file offset
other flags
pointer to first byte
File Pointer
current file offset
other flags
pointer to first byte
File Pointer
current file offset
other flags
pointer to first byte
File Pointer
current file offset
other flags
pointer to first byte

dup

```
#include <unistd.h>
int dup(int fd);
```

new file descriptor to the same file if OK, –1 on error


Kernel
File System
Process1
File Descriptors
File Pointer
fd1
fd2
fd3
fd4
...
File Pointer
current file offset
other flags
pointer to first byte
File Pointer
current file offset
other flags
pointer to first byte

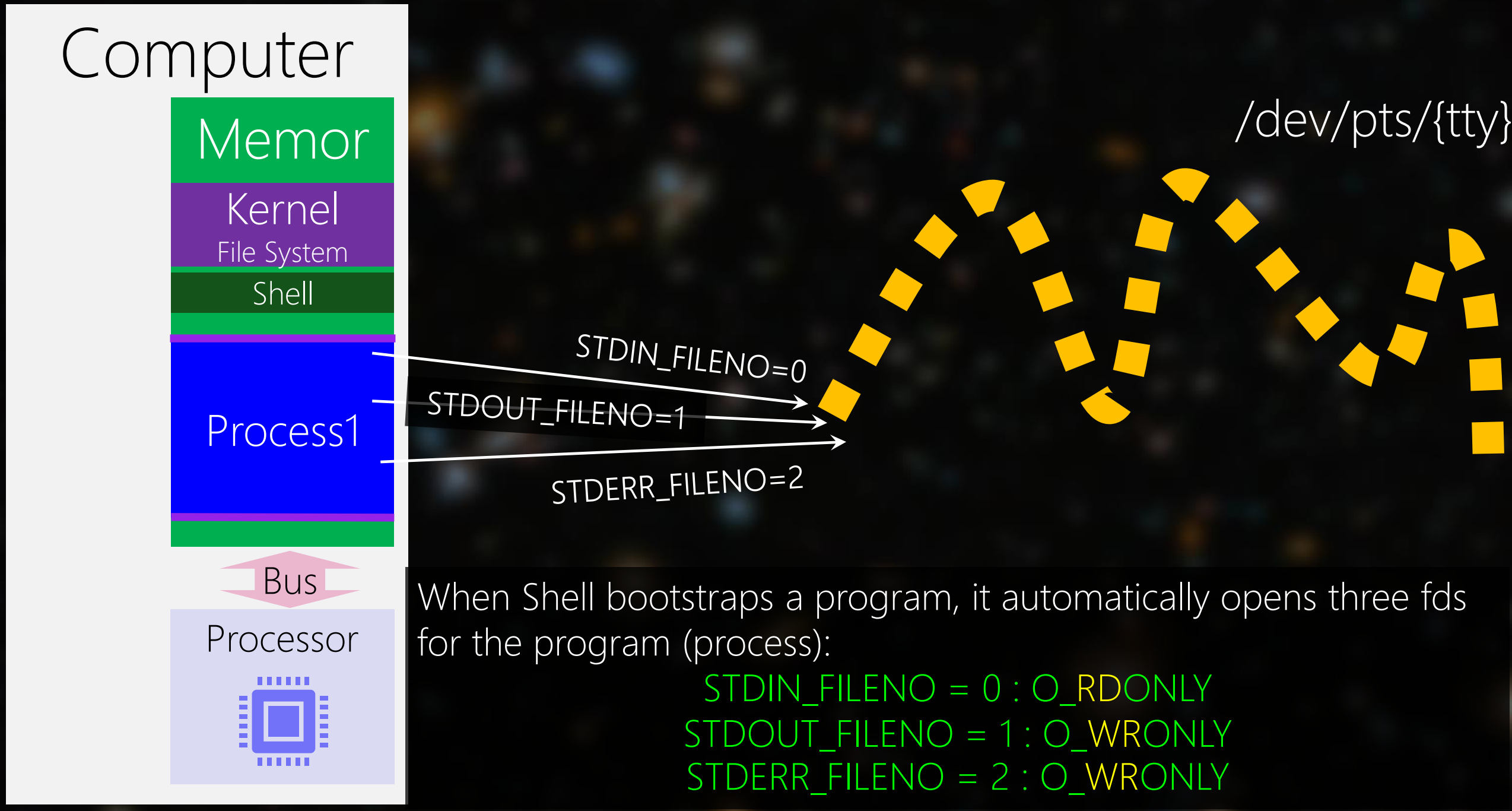
dup

Why do we duplicate fd?

I/O Redirection

STDIO ↔ File

STDERR → STDOUT



When Shell bootstraps a program, it automatically opens three fds for the program (process):

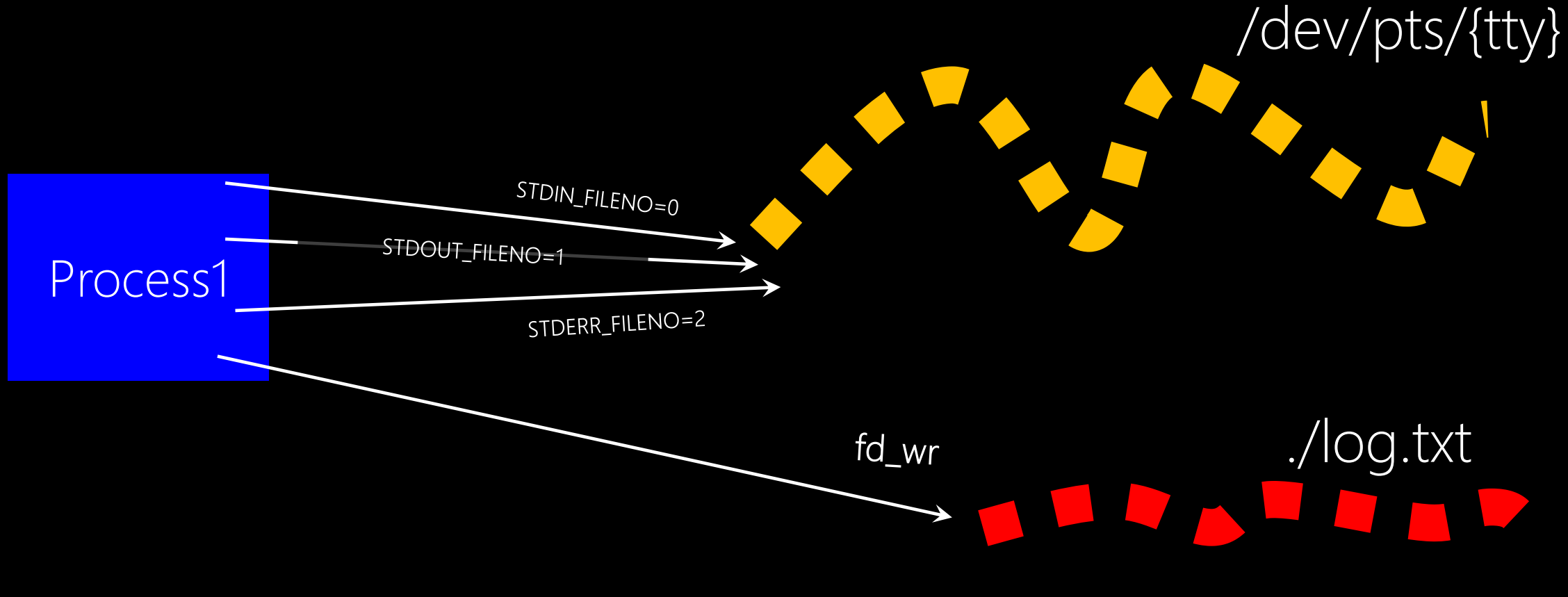
STDIN_FILENO = 0 : O_RDONLY

STDOUT_FILENO = 1 : O_WRONLY

STDERR_FILENO = 2 : O_WRONLY

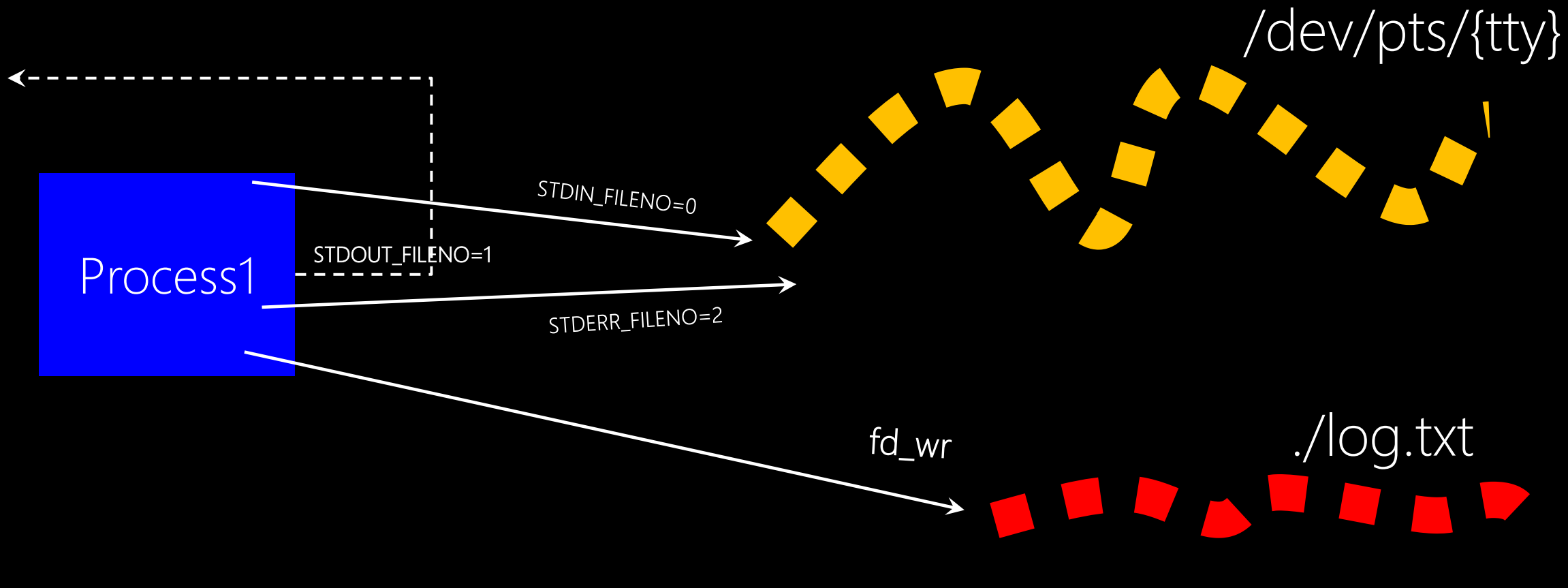
```c
#include <fcntl.h>
#include <unistd.h>
void main(void){
        char buf_rd[20];

        int fd_wr = open("./log.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
```



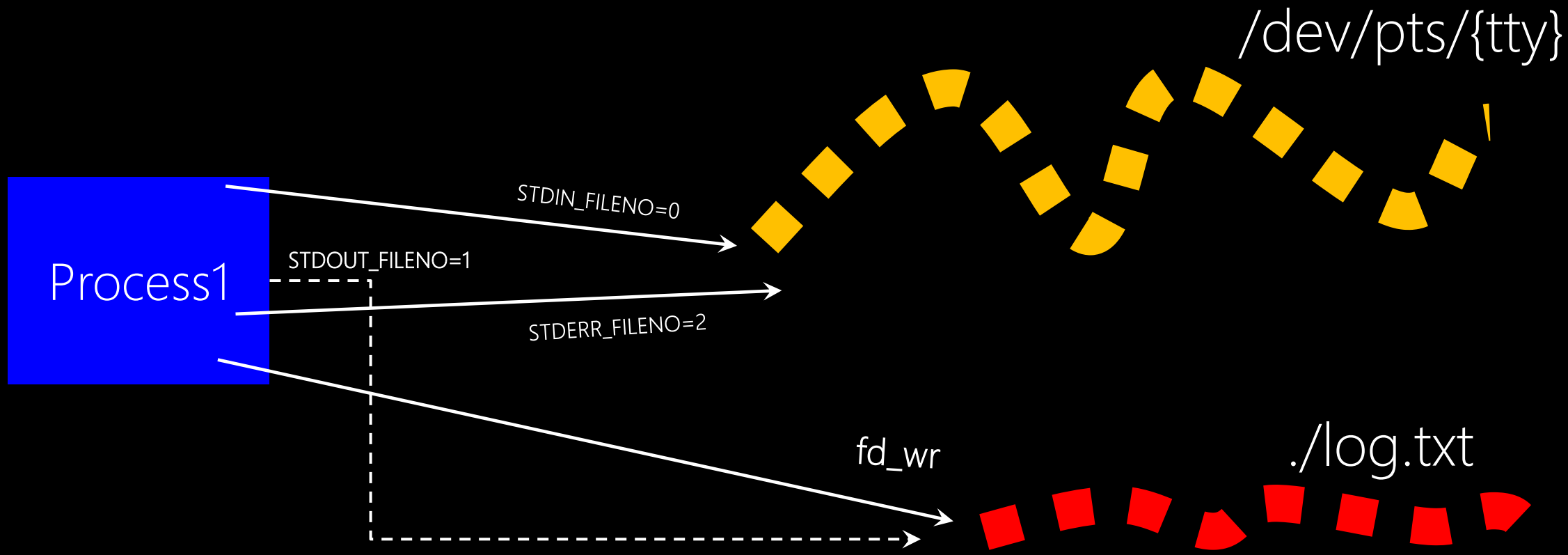
```c
#include <fcntl.h>
#include <unistd.h>
void main(void){
        char buf_rd[20];

        int fd_wr = open("./log.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
        close(1);
        //now the fd with value 1 is free
```



```c
#include <fcntl.h>
#include <unistd.h>
void main(void){
        char buf_rd[20];

        int fd_wr = open("./log.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
        close(1);
        //now the fd with value 1 is free
        //let's get it for our log file
        int new_fd = dup(fd_wr);
        //now the value of new_fd is 1
        //both fd_wr and new_fd are pointing to log.txt
```

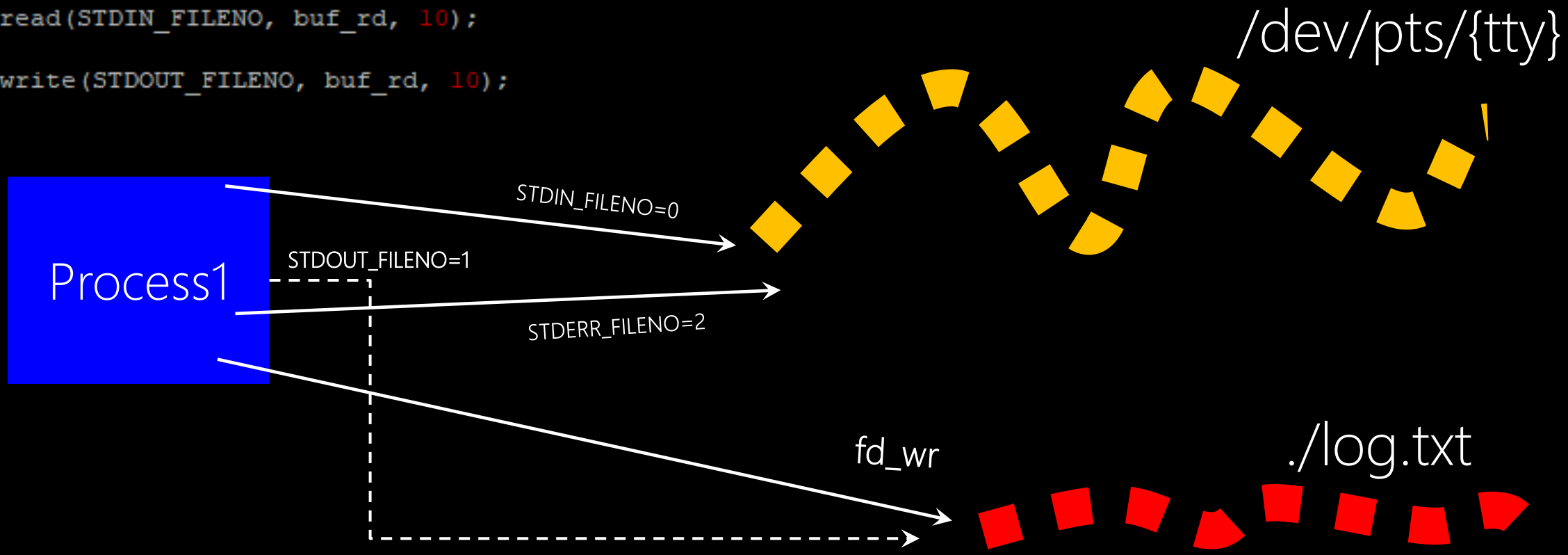


```c
#include <fcntl.h>
#include <unistd.h>
void main(void){
        char buf_rd[20];

        int fd_wr = open("./log.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
        close(1);
        //now the fd with value 1 is free
        //let's get it for our log file
        int new_fd = dup(fd_wr);
        //now the value of new_fd is 1
        //both fd_wr and new_fd are pointing to log.txt

        read(STDIN_FILENO, buf_rd, 10);

        write(STDOUT_FILENO, buf_rd, 10);
```



Shell

$ ./program > log.txt

We can ask the shell to do this redirection for us

{program file} > *{new destination for* STDOUT_FILENO*}*

`dup2`

At Home

I/O Efficiency
buffered vs. unbuffered

`unistd.h vs. stdio.h`

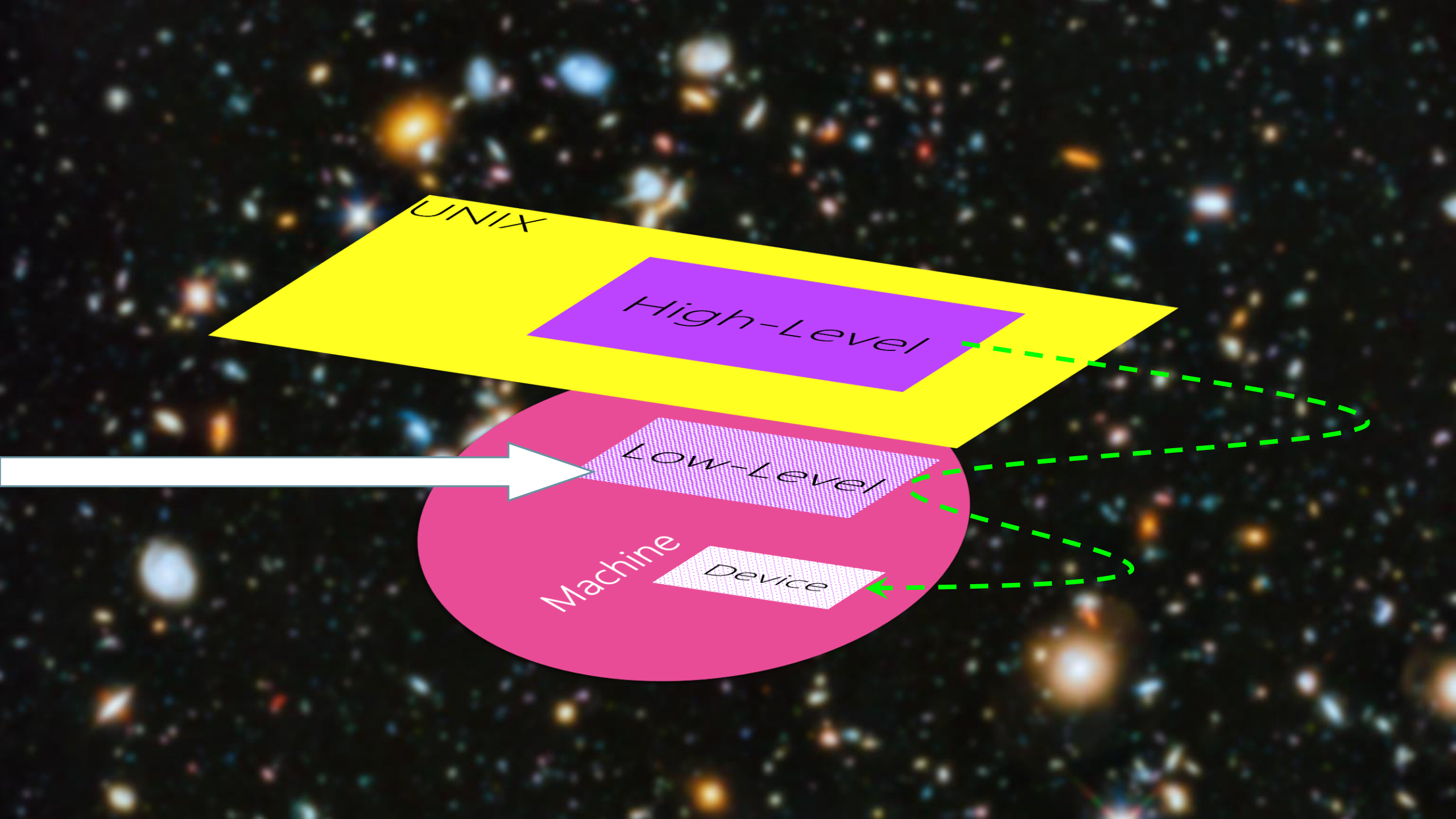
At Home.

File Sharing

Advanced! We won’t cover it.

Atomic Operation

Advanced! We won’t cover it.

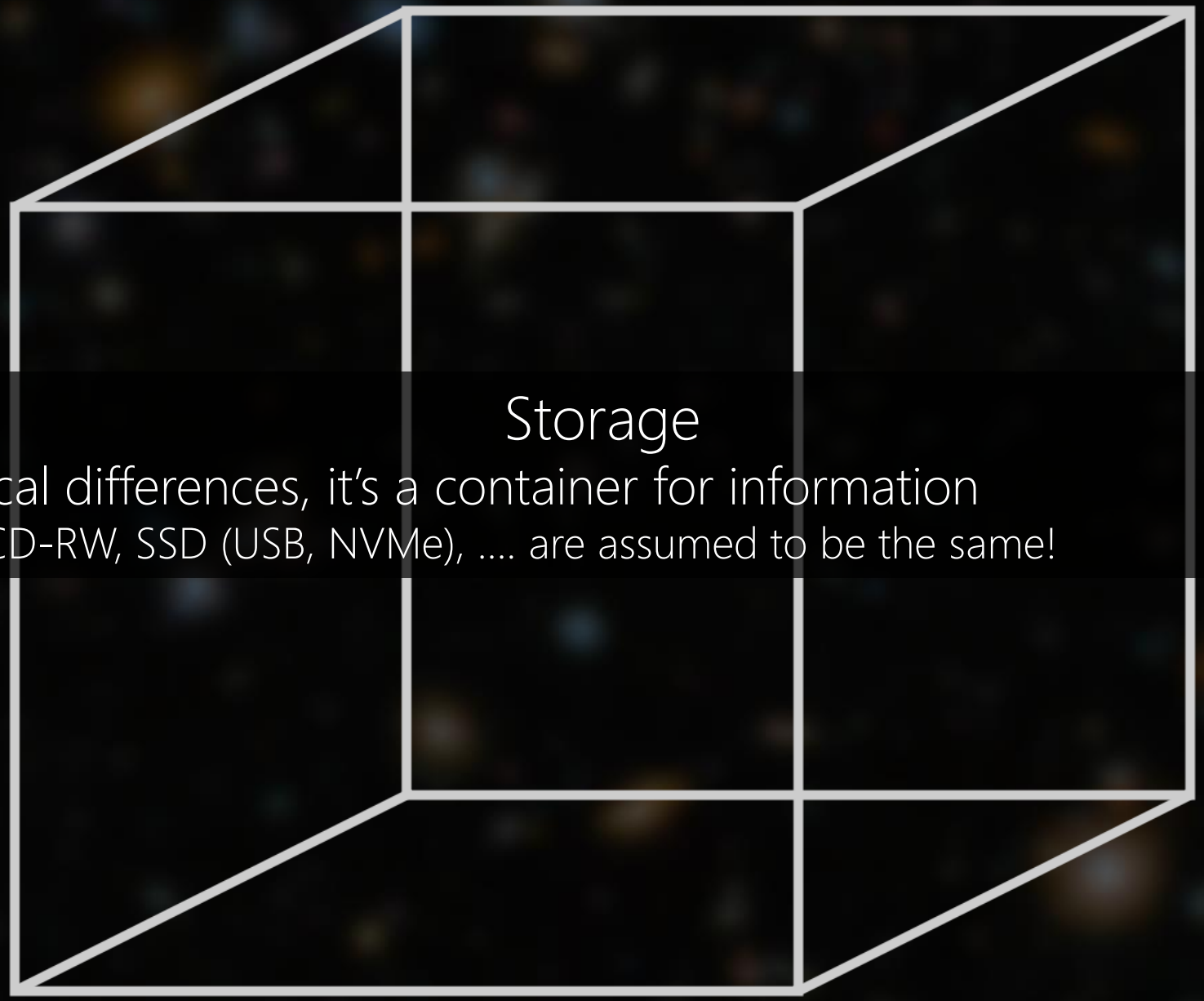

UNIX
High-Level
Low-Level
Machine
Device

Storage File System

aka. Disk File System

UNIX File System (UFS) vs. NTFS, ext4, ...

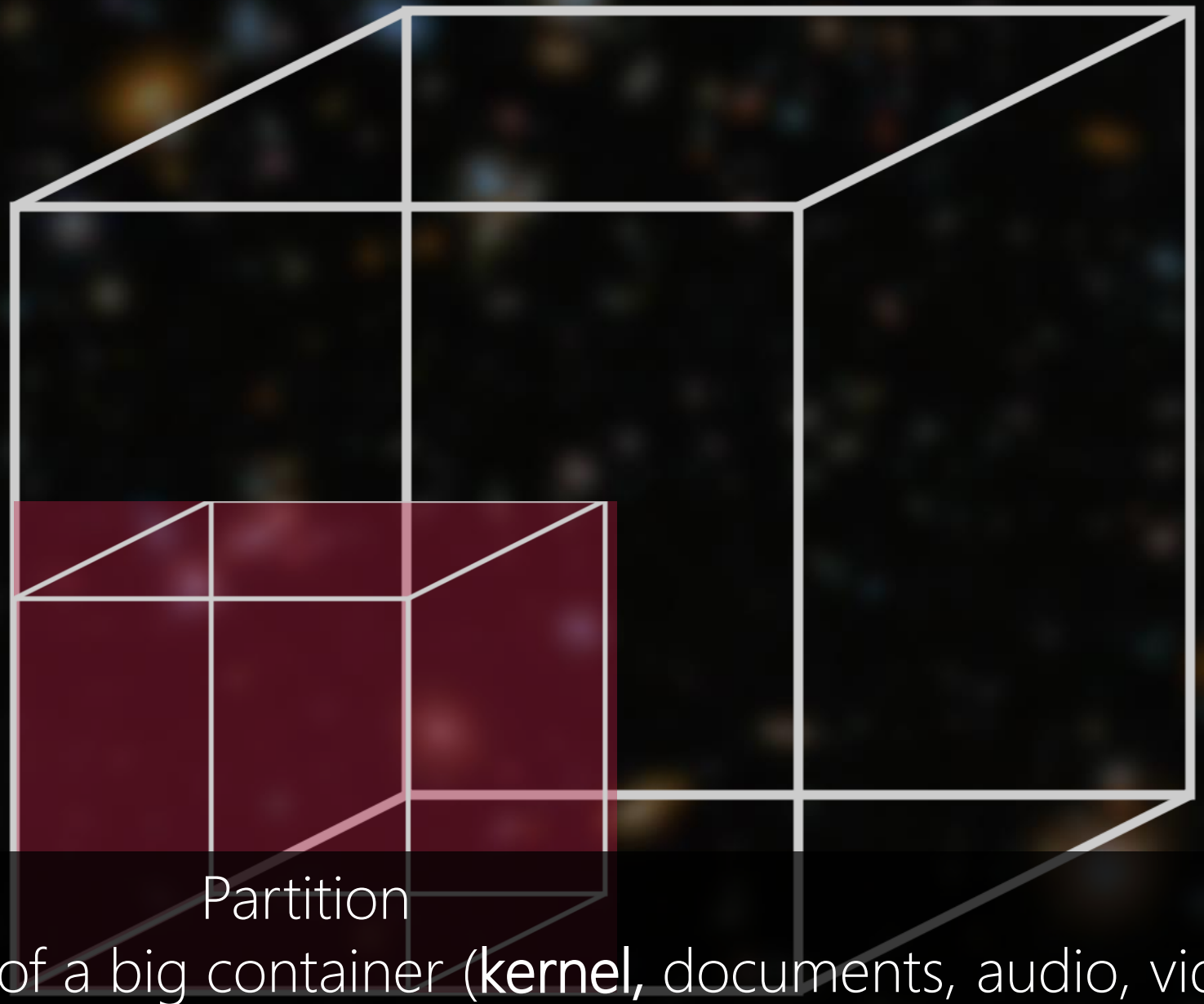
File System: Low Level



Storage

Regardless of physical differences, it's a container for information

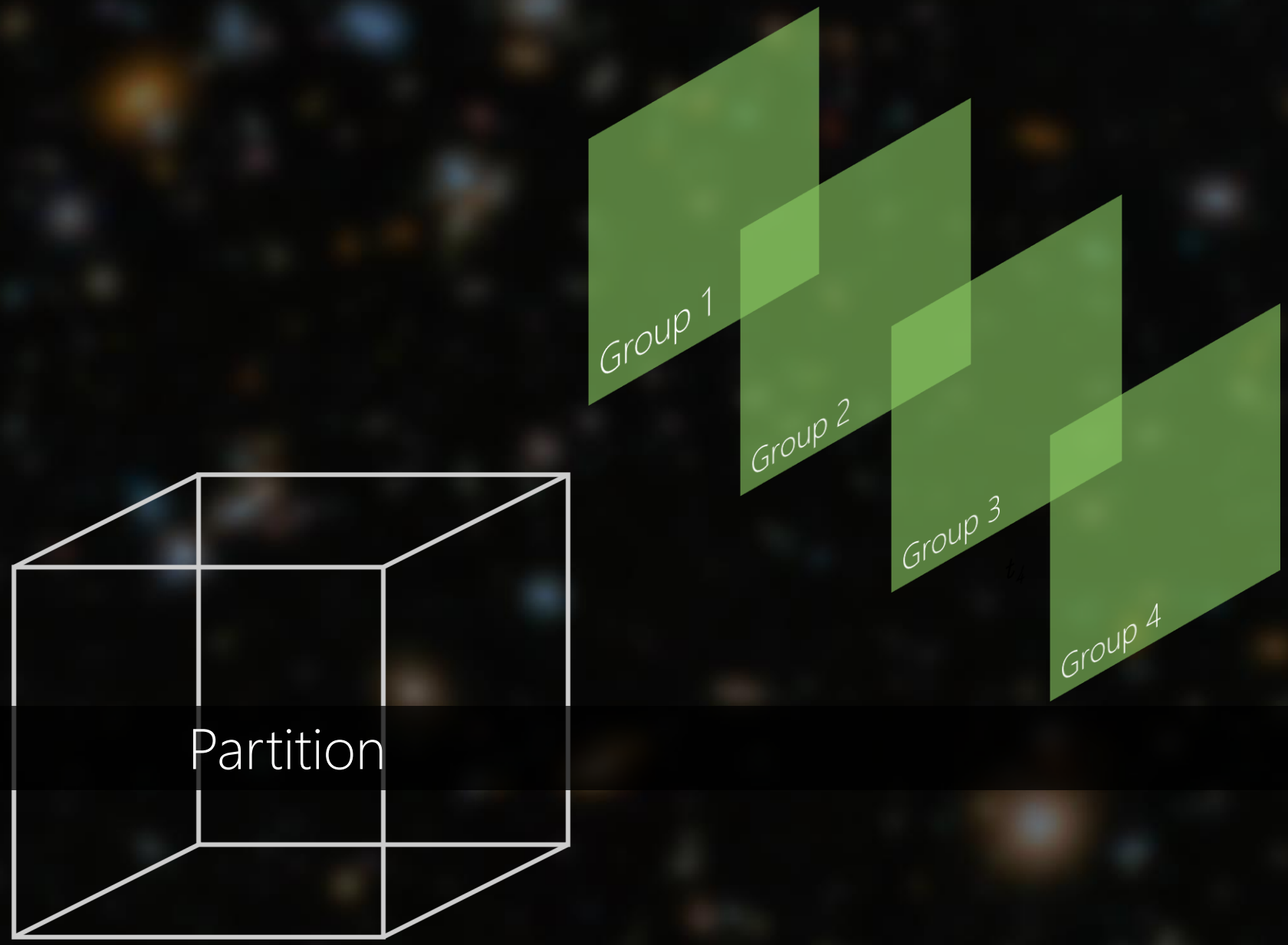
HDD-IDE, HDD-SATA, CD-RW, SSD (USB, NVMe), …. are assumed to be the same!

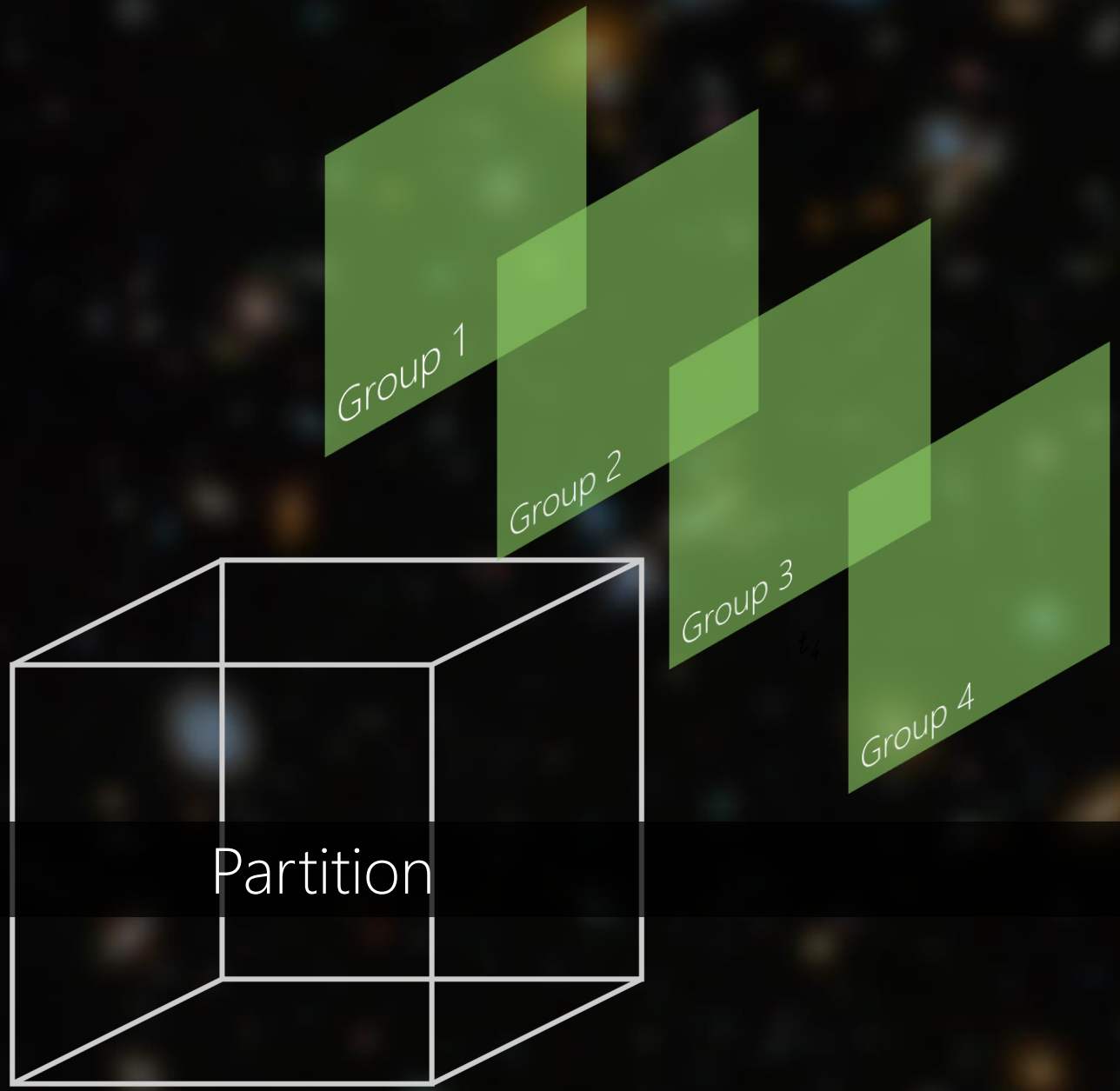


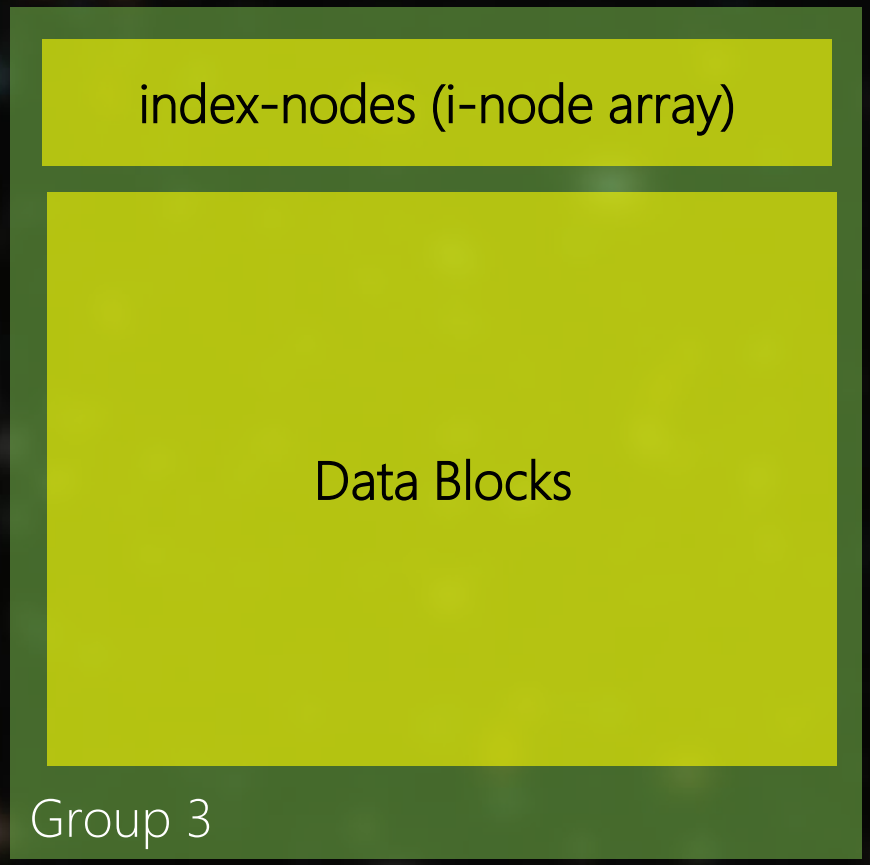
Partition

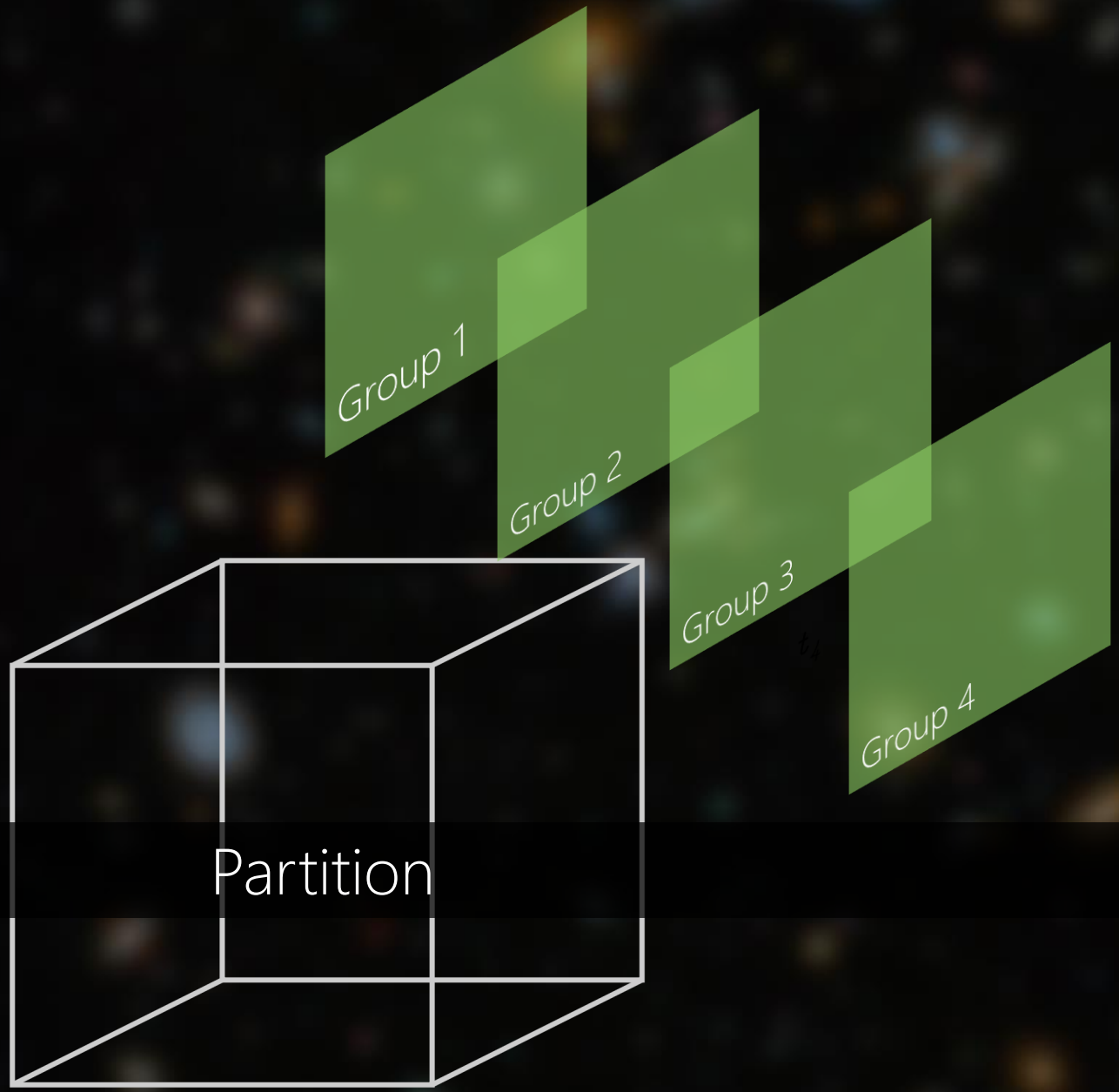
Physical division of a big container (**kernel,** documents, audio, video, …)

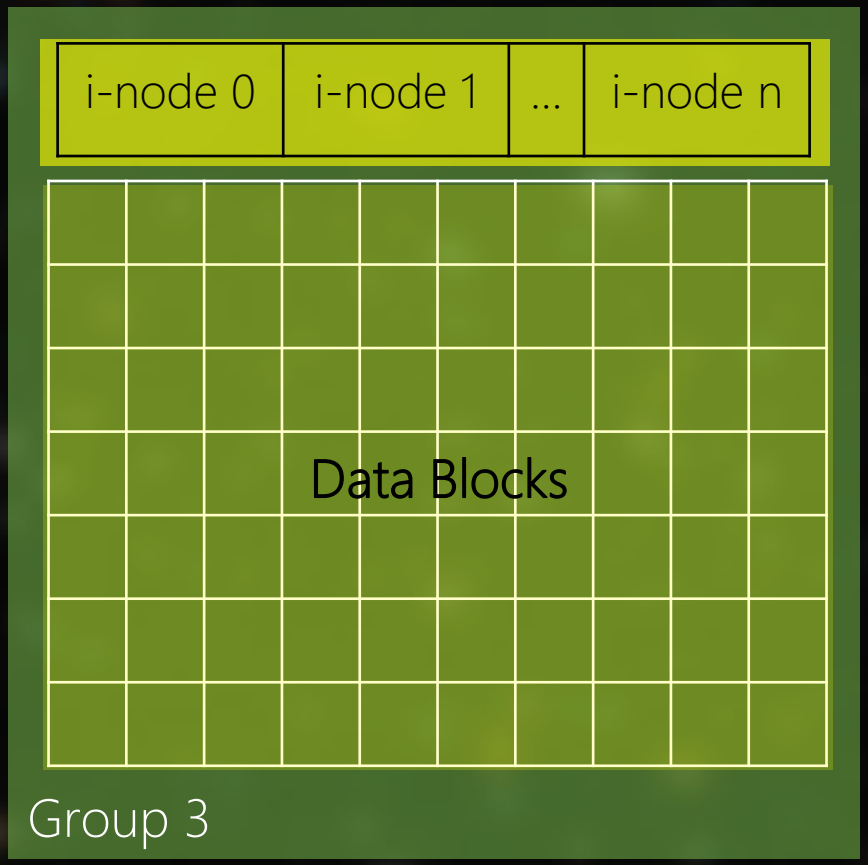
Some special treatment (customization) for each partition


Group 1
Group 2
Group 3
Group 4
Partition


Group 1
Group 2
Group 3
Group 4
Partition


index-nodes (i-node array)
Data Blocks
Group 3


Group 1
Group 2
Group 3
Group 4
Partition


i-node 0
i-node 1
...
i-node n
Data Blocks
Group 3

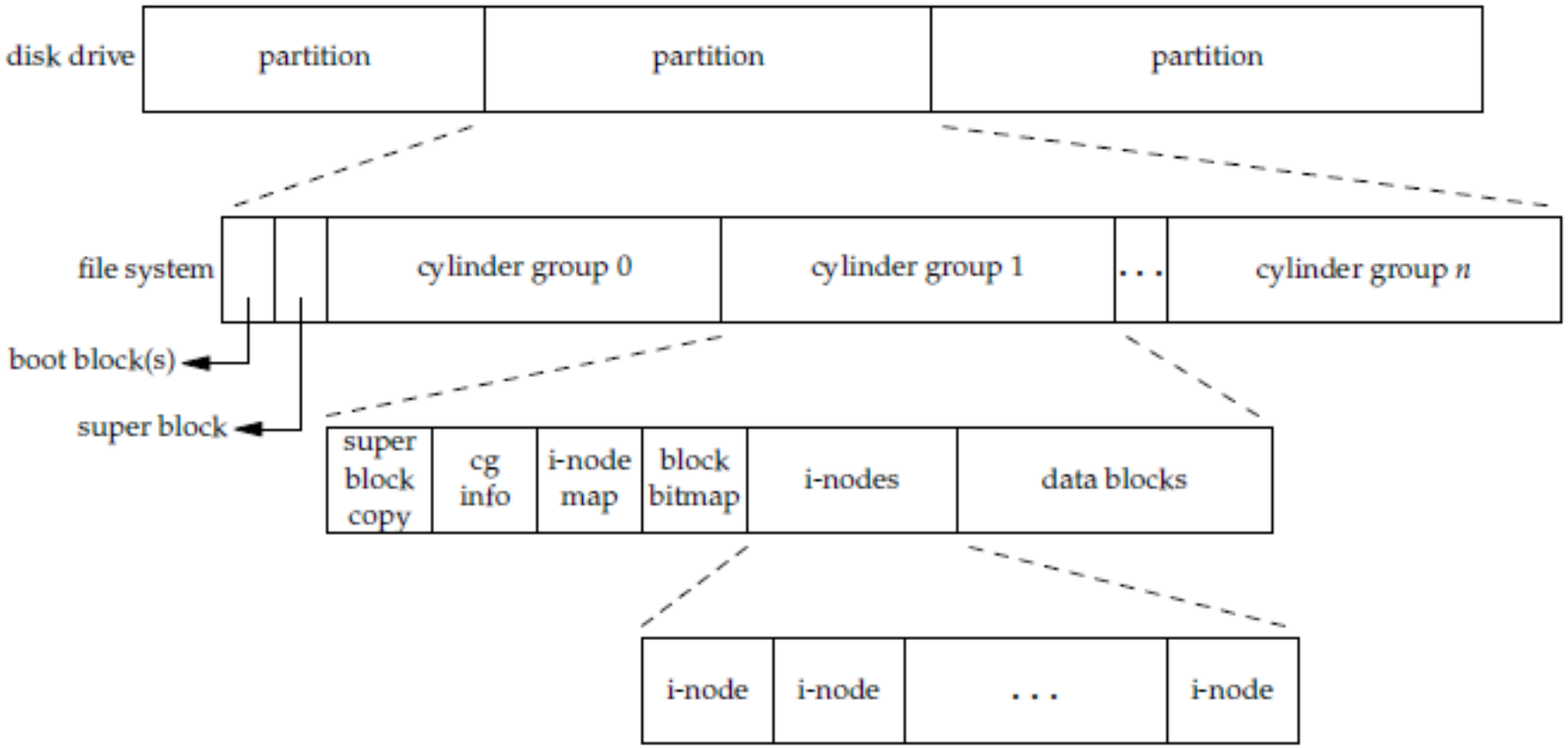
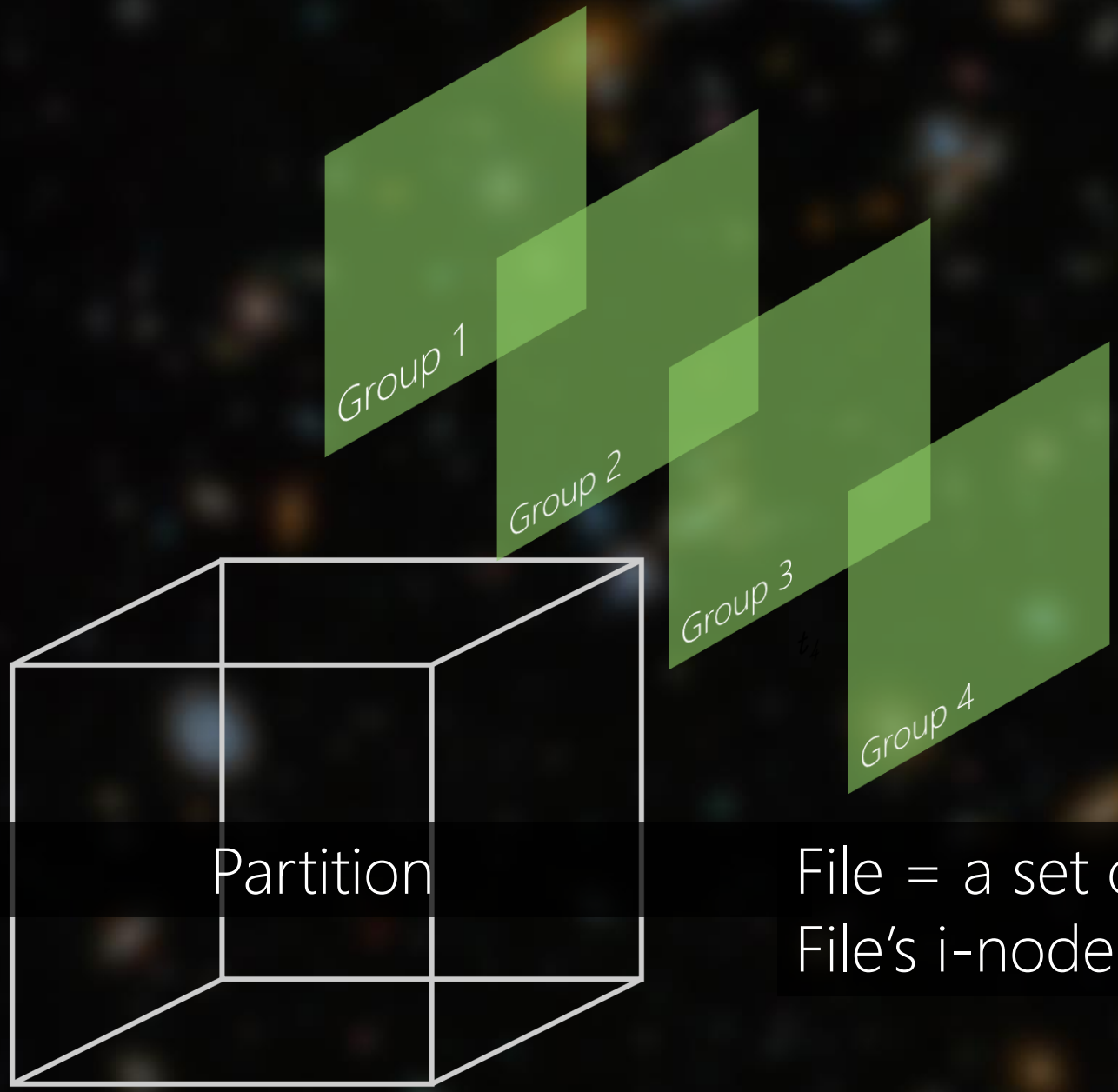
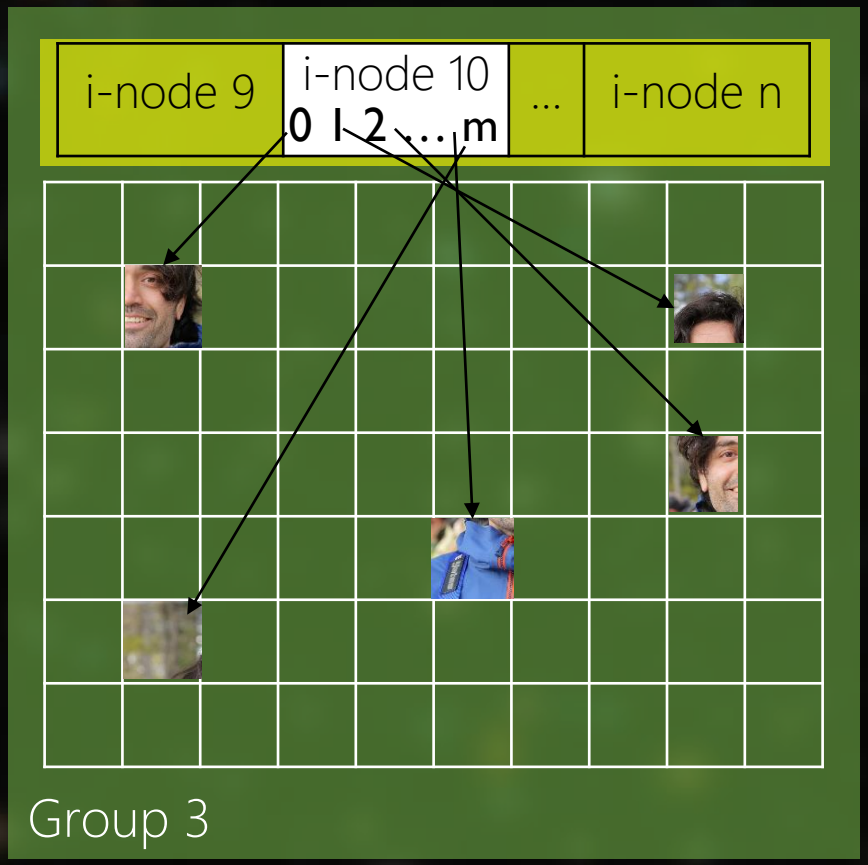


Figure 4.13 Disk drive, partitions, and a file system





File = a set or heap (not list) of data blocks

File's i-node knows the order!

Each File has one index-node (i-node)

```
$ ls -i {filename}
```

```
hfani@charlie:~$ ls -i ./hfani.jpeg
94552308 ./hfani.jpeg
```

Each File has many data blocks

Shell → stat {filename}

```
hfani@charlie:~$ stat ./hfani.jpeg
  File: ./hfani.jpeg
  Size: 113327          Blocks: 223        IO Block: 1048576 regular file
Device: 29h/41d Inode: 94552308    Links: 1
Access: (0644/-rw-r--r--)  Uid: (239080/    hfani)   Gid: (  400/    temp)
Access: 2021-07-05 22:48:41.000000000 -0400
Modify: 2021-07-05 22:48:41.000000000 -0400
Change: 2021-10-27 05:15:35.999746375 -0400
 Birth: -
```

Each File has many data blocks

System Call → stat, fstat

```
#include <sys/stat.h>
int stat(const char *restrict pathname, struct stat *restrict buf);
int fstat(int fd, struct stat *buf);
```

Return 0 if OK –1 on error

```
struct stat {
    mode_t st_mode;          /* file type & mode (permissions) */
    ino_t st_ino;            /* i-node number (serial number) */
    dev_t st_dev;            /* device number (file system) */
    dev_t st_rdev;           /* device number for special files */
    nlink_t st_nlink;        /* number of links */
    uid_t st_uid;            /* user ID of owner */
    gid_t st_gid;            /* group ID of owner */
    off_t st_size;           /* size in bytes, for regular files */
    struct timespec st_atim;/* time of last access */
    struct timespec st_mtim;/* time of last modification */
    struct timespec st_ctim;/* time of last file status change */
    blksize_t st_blksize;    /* best I/O block size */
    blkcnt_t st_blocks;      /* number of disk blocks allocated */
};
```

```c
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>

void main(void)
{
        struct stat sb;

        char *file = "./hfani.jpeg";
        int fd = open(file, O_RDONLY);

        if (fstat(fd, &sb) == -1) {
                printf("error in fetching stat on file %s", file);
                return;
        }

        printf("I-node number: %d\n", sb.st_ino);
        printf("Preferred I/O block size: %d bytes\n",  sb.st_blksize);
        printf("File size: %d bytes\n",  sb.st_size);
        printf("Blocks allocated: %d\n", sb.st_blocks);
}
```

```
hfani@charlie:~$ cc stat.c -o stat
hfani@charlie:~$ ./stat
I-node number: 94552308
Preferred I/O block size: 1048576 bytes
File size: 113327 bytes
Blocks allocated: 223
```

The size of data blocks are determined at the time of partition creation (aka partitioning & formatting) and is fixed afterward.

Large (10MB) vs. Small (1MB)

Fragmentation

```
hfani@charlie:~$ stat ./hfani.jpeg
  File: ./hfani.jpeg
  Size: 113327          Blocks: 223        IO Block: 1048576 regular file
Device: 29h/41d Inode: 94552308    Links: 1
Access: (0644/-rw-r--r--)  Uid: (239080/    hfani)   Gid: (  400/    temp)
Access: 2021-07-05 22:48:41.000000000 -0400
Modify: 2021-07-05 22:48:41.000000000 -0400
Change: 2021-10-27 05:15:35.999746375 -0400
 Birth: -
```

/Path/Filename ⟶ i-node ⟶ Data Blocks

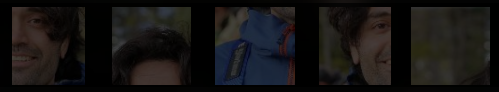
`/home/hfani/hfani.jpeg → 94552308 →` 

/Path/Filename ⟶ i-node ⟶ Data Blocks

/home/hfani/hfani.jpeg → 94552308 → 

/Path/Filename ⟶ i-node ⟶ Data Blocks

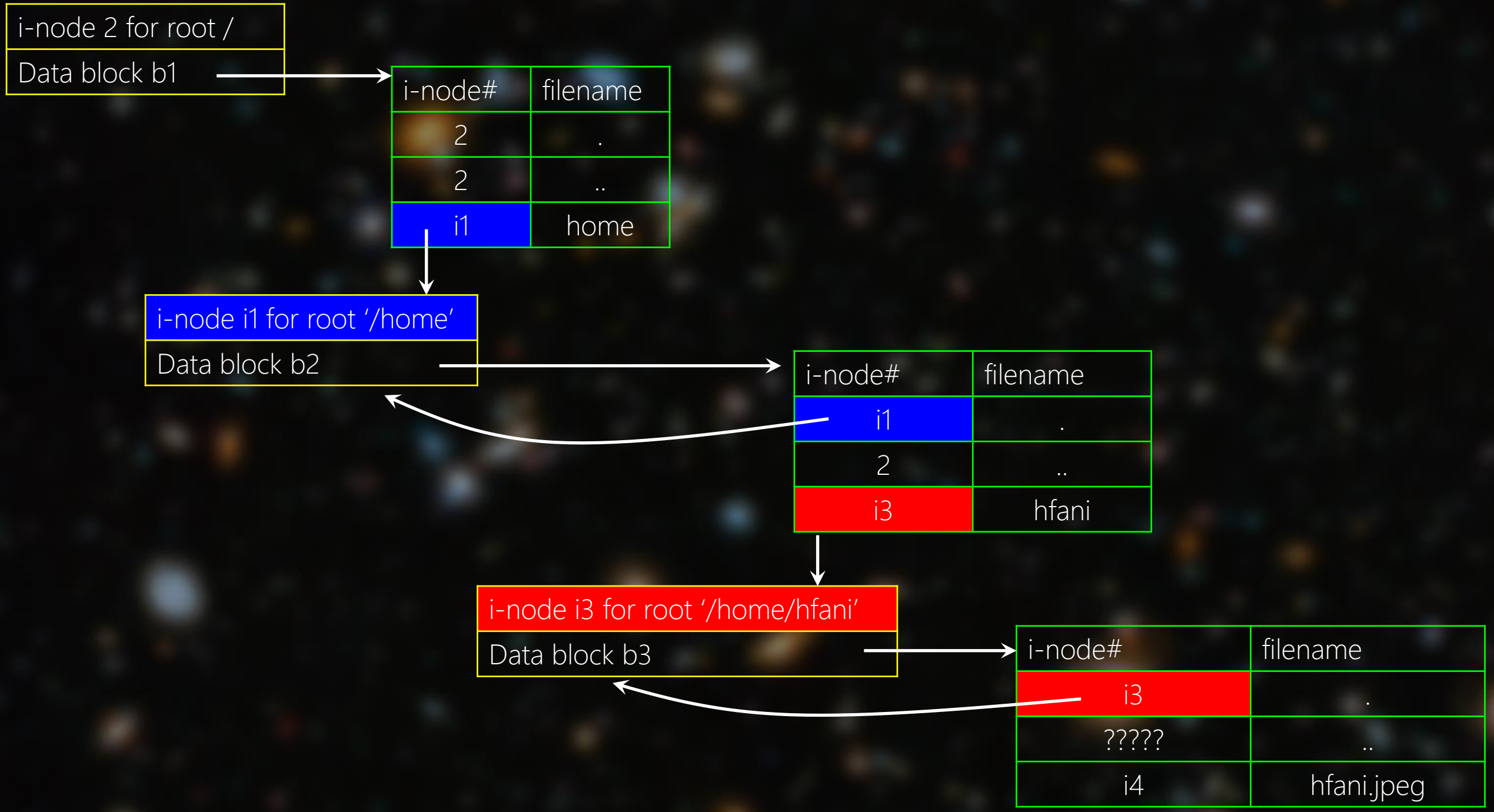
`/home/hfani/hfani.jpeg → 94552308 →`



Directories

They are files. So, each of them has one i-node.
The content is not an image, audio, … but mapping between filenames and their i-nodes

Logical (not physical) division of files


i-node 2 for root /
Data block b1
i-node#
filename
2
.
2
..
i1
home
i-node i1 for root '/home'
Data block b2
i-node#
filename
i1
.
2
..
i3
hfani
i-node i3 for root '/home/hfani'
Data block b3
i-node#
filename
i3
.
?????
..
i4
hfani.jpeg

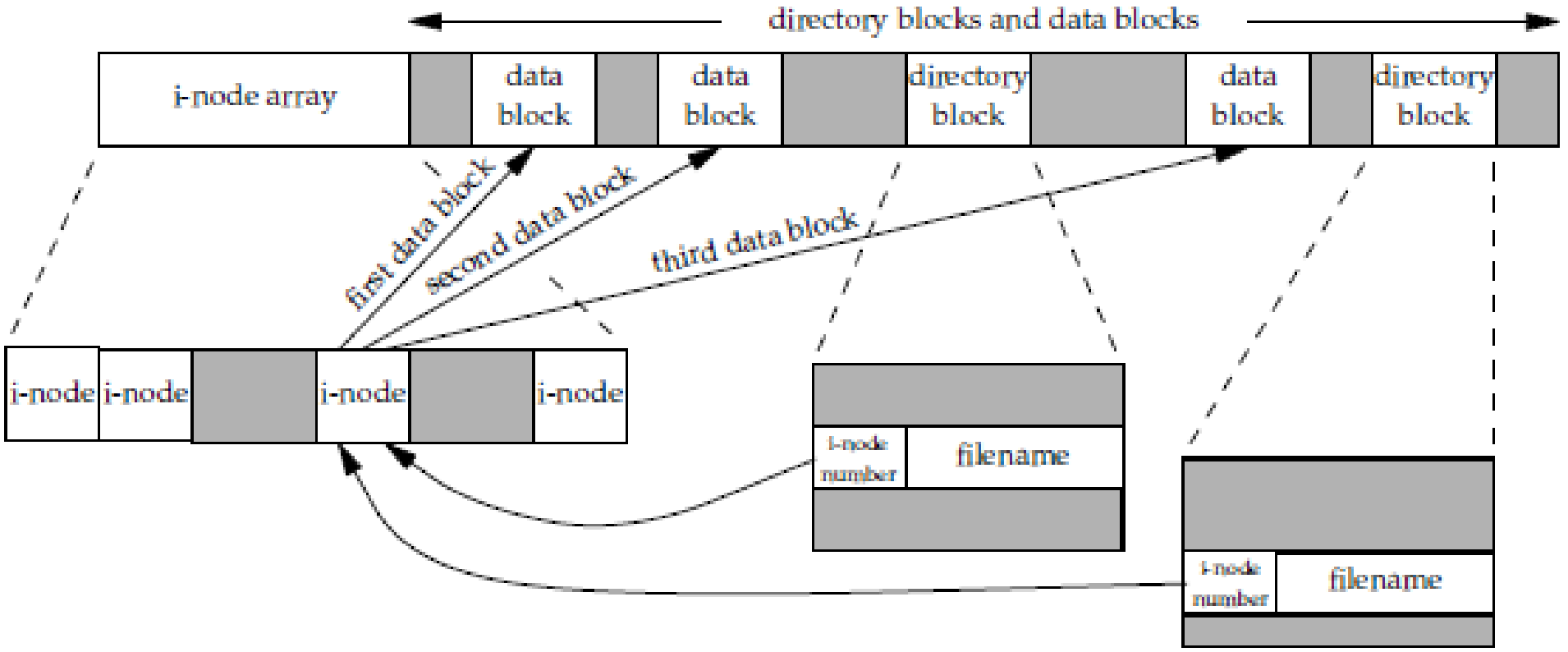
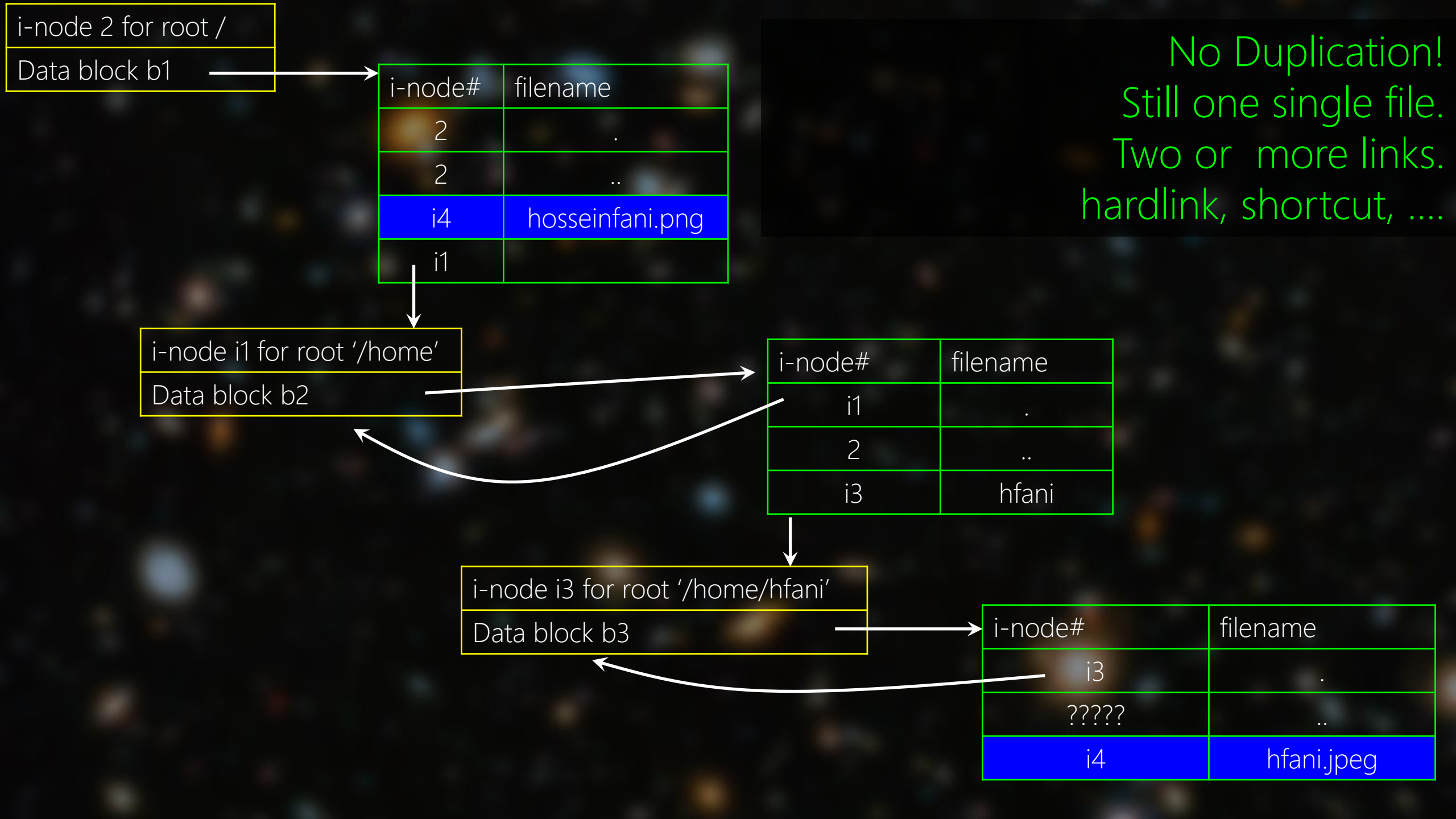


Figure 4.14 Cylinder group's i-nodes and data blocks in more detail

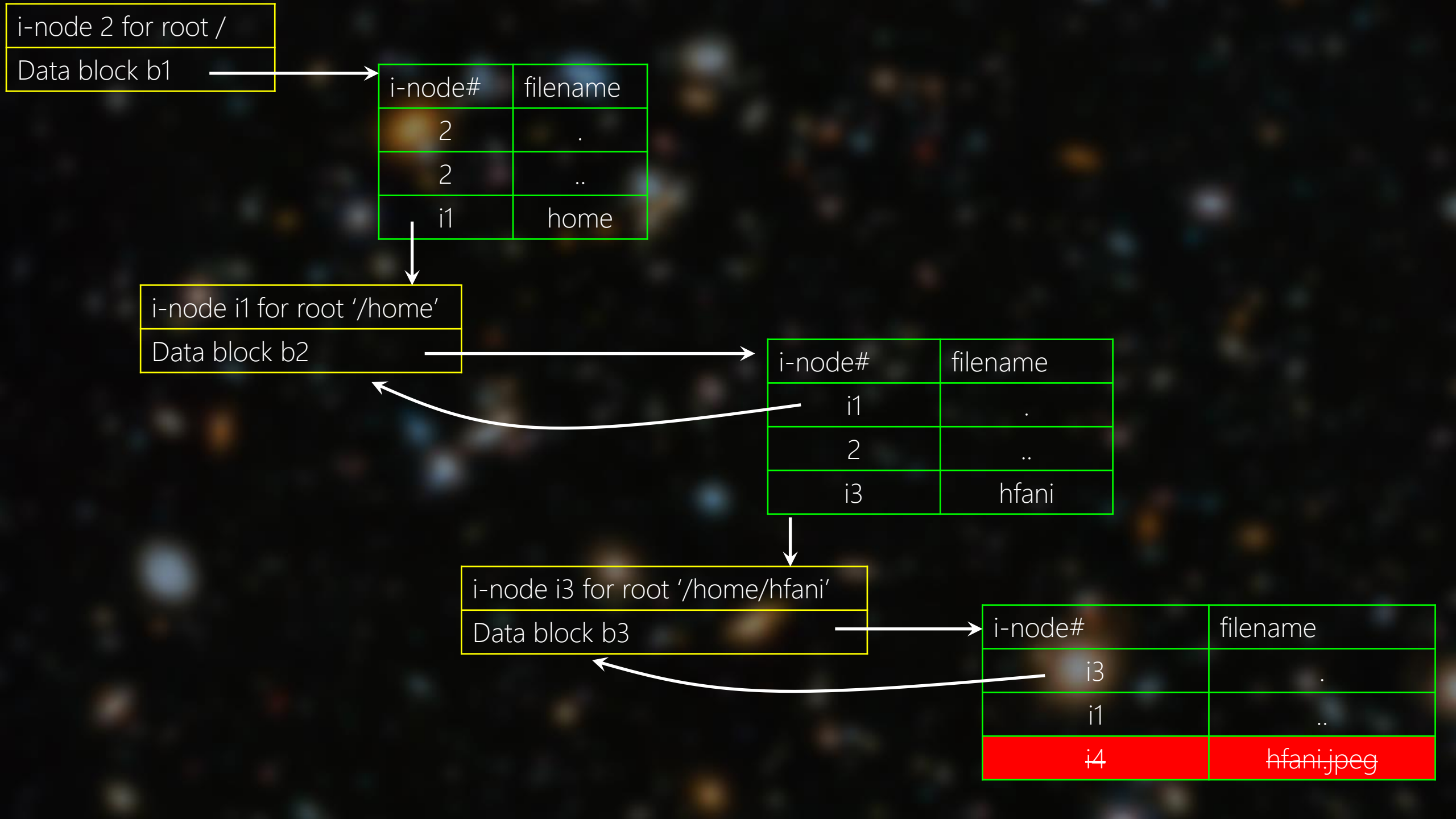
Directories

Is it possible to have multiple links to the same file? Yes.
How?


i-node 2 for root /
Data block b1
i-node#
filename
2
.
2
..
i4
hosseinfani.png
i1
i-node i1 for root '/home'
Data block b2
i-node#
filename
i1
.
2
..
i3
hfani
i-node i3 for root '/home/hfani'
Data block b3
i-node#
filename
i3
.
?????
..
i4
hfani.jpeg
No Duplication!
Still one single file.
Two or more links.
hardlink, shortcut, ...

What happens when you delete a file?

Zero all data blocks of the file?
Zero all data blocks of the file and Zero its i-node pointers?
Zero its entry (i-node, name) in the parent directory?


i-node 2 for root /
Data block b1
i-node#
filename
2
.
2
..
i1
home
i-node i1 for root '/home'
Data block b2
i-node#
filename
i1
.
2
..
i3
hfani
i-node i3 for root '/home/hfani'
Data block b3
i-node#
filename
i3
.
i1
..
i4
hfani.jpeg

What happens when you delete a file?

Is it able to recover a deleted file?

What happens when you move a file?

What happens when you copy a file?

File Types

- Regular Files → text or binary
- Directory Files → (i-node, filename) pairs
- Special Files (Devices)
  - Block → HDD, SSD, CD, …
  - Character (Stream) → TTY, Keyboard, Mouse, Printer, …
- Socket (Networking)
- FIFO (Pipes)
- Symbolic LinK

RD/WR on a Regular File: `open("./myfile.txt")`

vs.

RD/WR on a Storage (Lab06): `open("/dev/sda1")`

- Regular Files → text or binary
- Directory Files → (i-node, filename) pairs
- Special Files (Devices)
  - Block → HDD, SSD, CD, …
  - Character (Stream) → TTY, Keyboard, Mouse, Printer, …
- Socket (Networking)
- FIFO (Pipes)
- Symbolic LinK

Process Manager

aka. Process Control