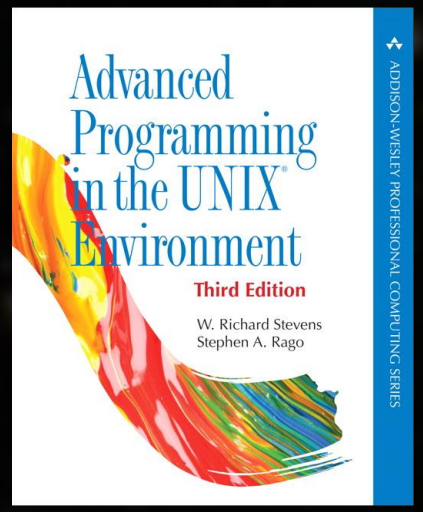
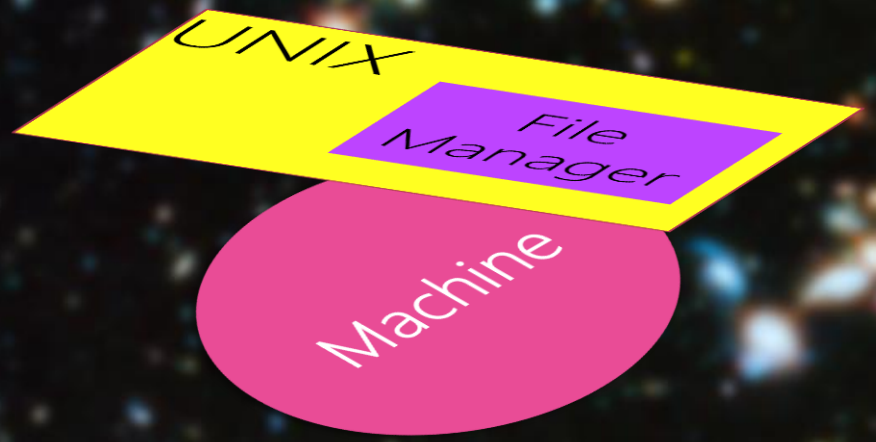
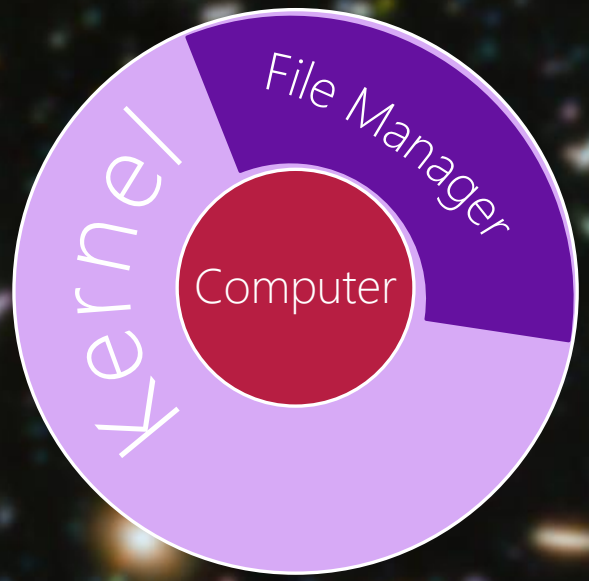
Midterm Exam Marks + Lab07 + Lec07

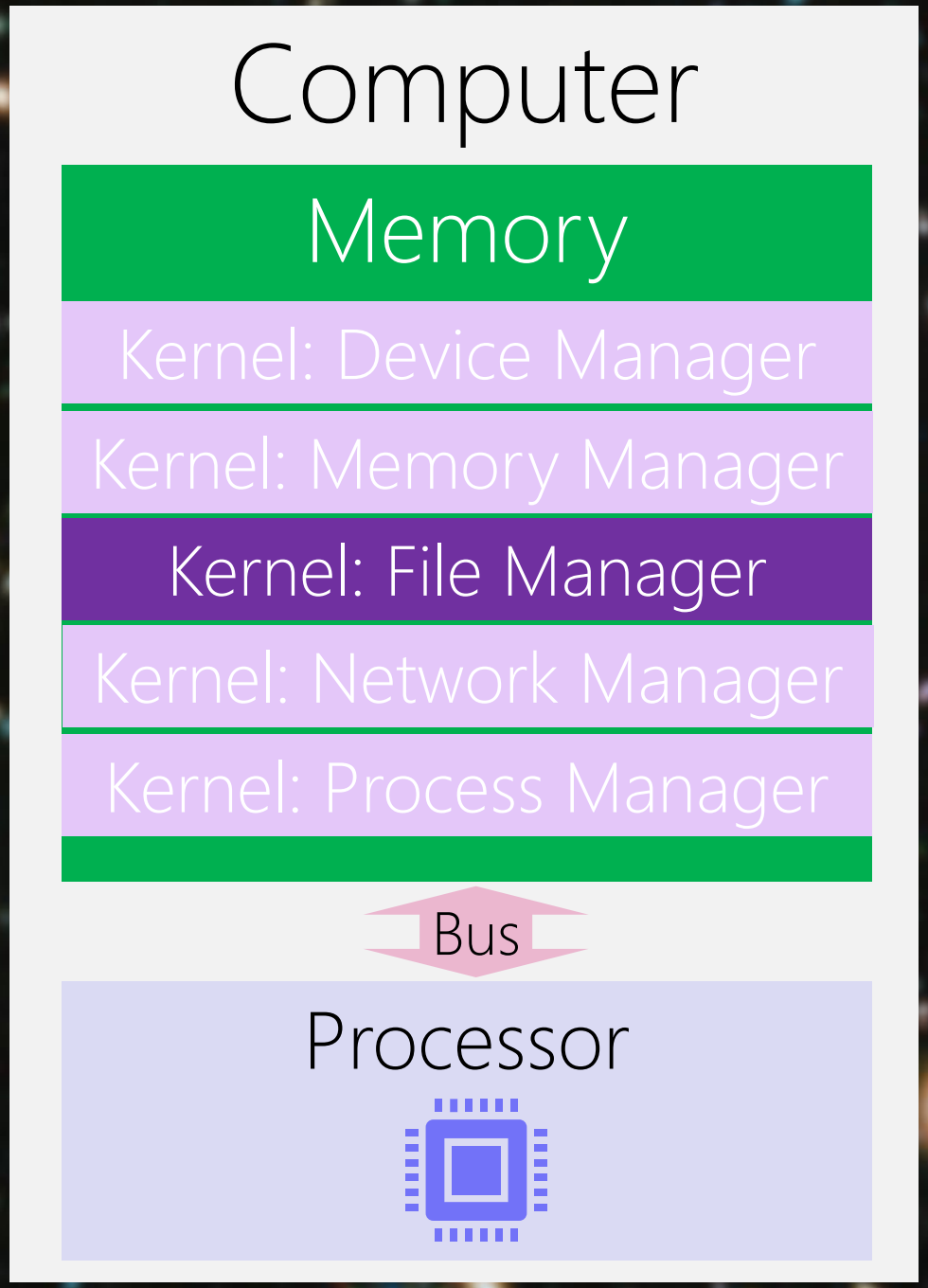
- Review File System `lseek`, `dup`, I/O Redirection
- File System for Storage Devices: `i-node`, Directories

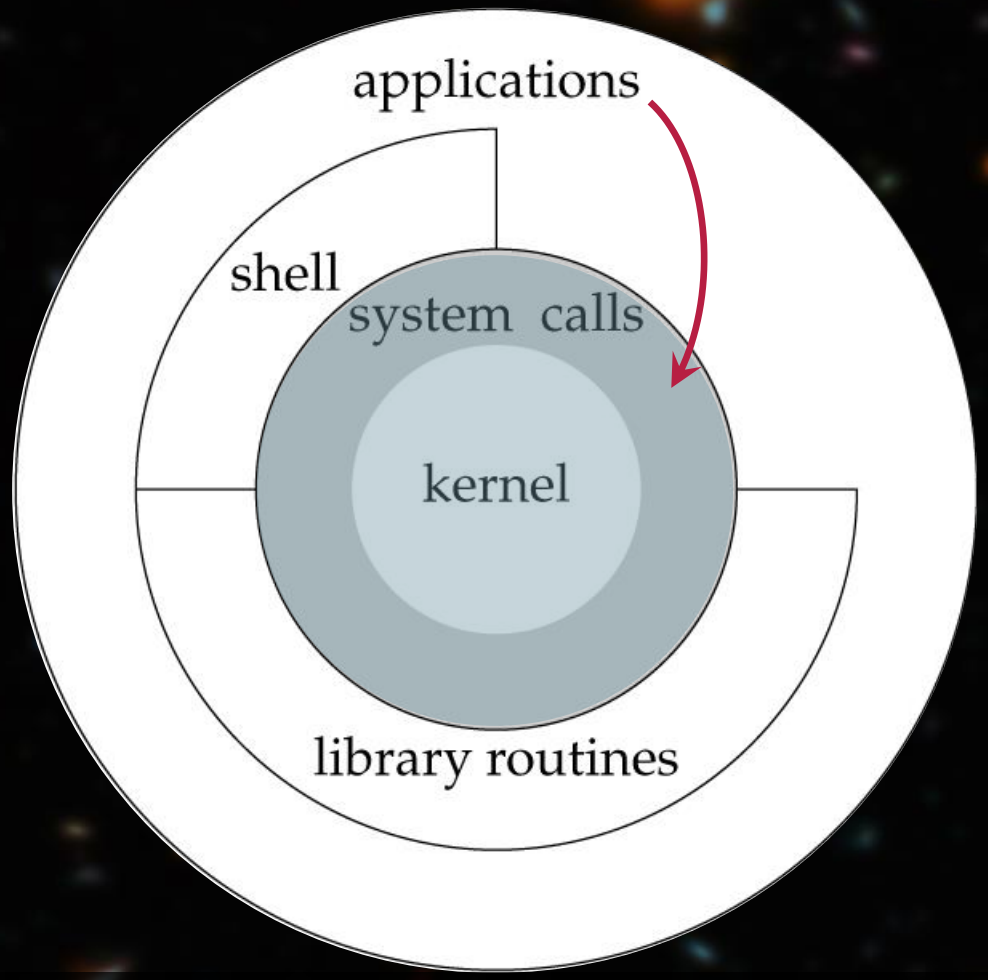


Chapter 03: File I/O
Chapter 04: File I/O


UNIX
File Manager
Machine


Kernel
File Manager
Computer


Computer
Memory
Kernel: Device Manager
Kernel: Memory Manager
Kernel: File Manager
Kernel: Network Manager
Kernel: Process Manager
Bus
Processor



Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries (Section 4.22)
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control (Section 3.14)
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file (Section 6.4)
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function (Section 14.4.2)
<pthread.h>	•	•	•	•	threads (Chapters 11 and 12)
<pwd.h>	•	•	•	•	password file (Section 6.2)
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O (Chapter 18)
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions
<arpa/inet.h>	•	•	•	•	Internet definitions (Chapter 16)
<net/if.h>	•	•	•	•	socket local interfaces (Chapter 16)
<netinet/in.h>	•	•	•	•	Internet address family (Section 16.3)
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function (Section 14.4.1)
<sys/socket.h>	•	•	•	•	sockets interface (Chapter 16)
<sys/stat.h>	•	•	•	•	file status (Chapter 4)
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times (Section 8.17)
<sys/types.h>	•	•	•	•	primitive system data types (Section 2.8)
<sys/un.h>	•	•	•	•	UNIX domain socket definitions (Section 17.2)
<sys/utsname.h>	•	•	•	•	system name (Section 6.9)
<sys/wait.h>	•	•	•	•	process control (Section 8.6)

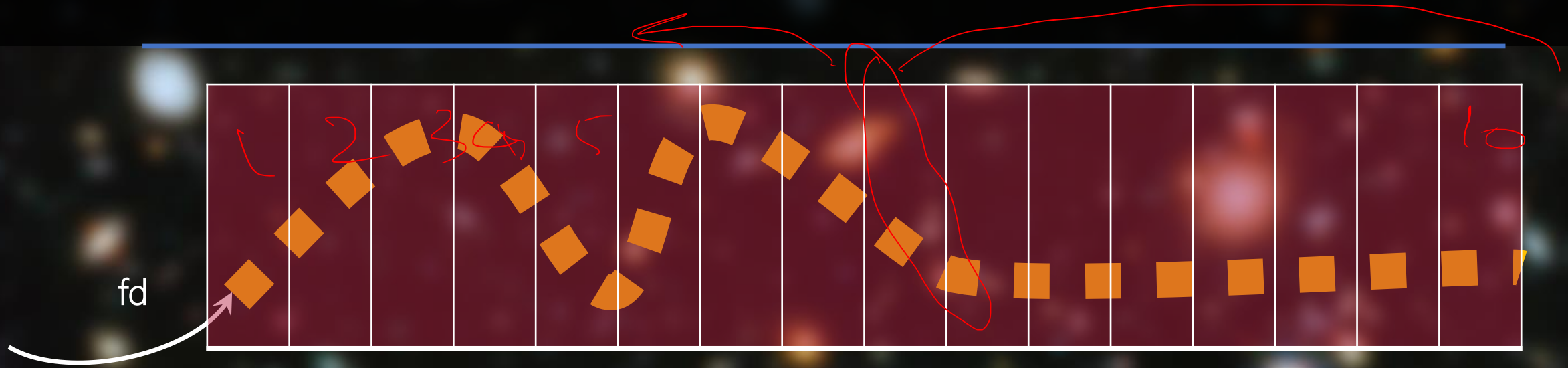
lseek

A Better Use Case: Binary Search in Sorted Elements

$x = 1M$

$O(n)$ los

$O(\log n)$



dup

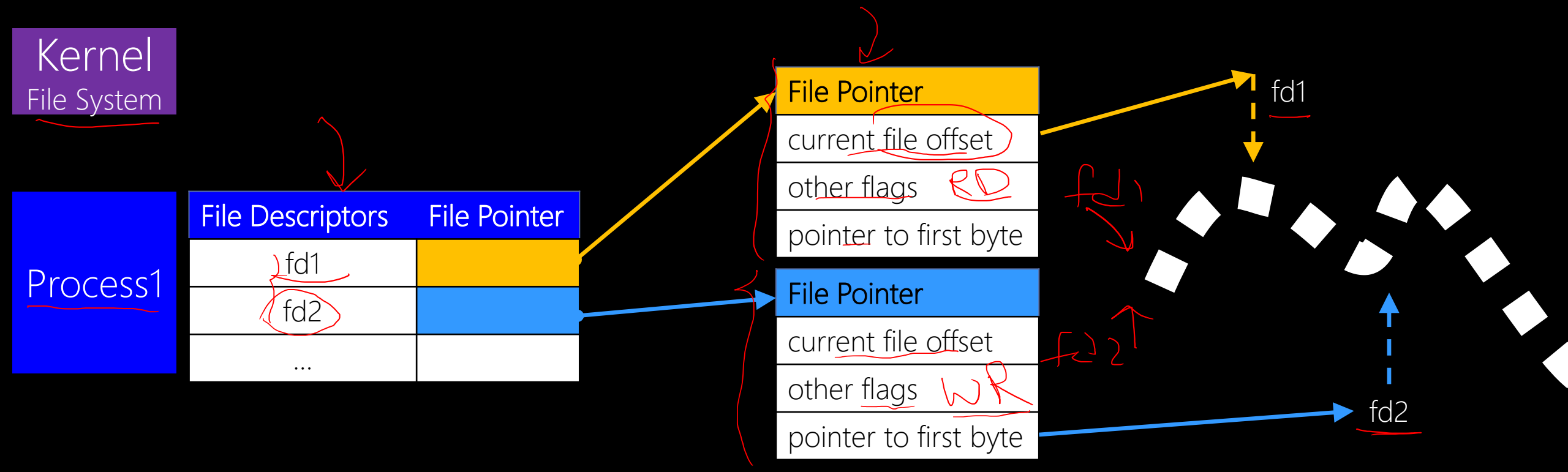
```
#include <unistd.h>
int dup(int fd);
```

the lowest-numbered available file descriptor to the same file if OK, –1 on error

dup vs. multiple open

open() : each returned fd has its own properties and current offset

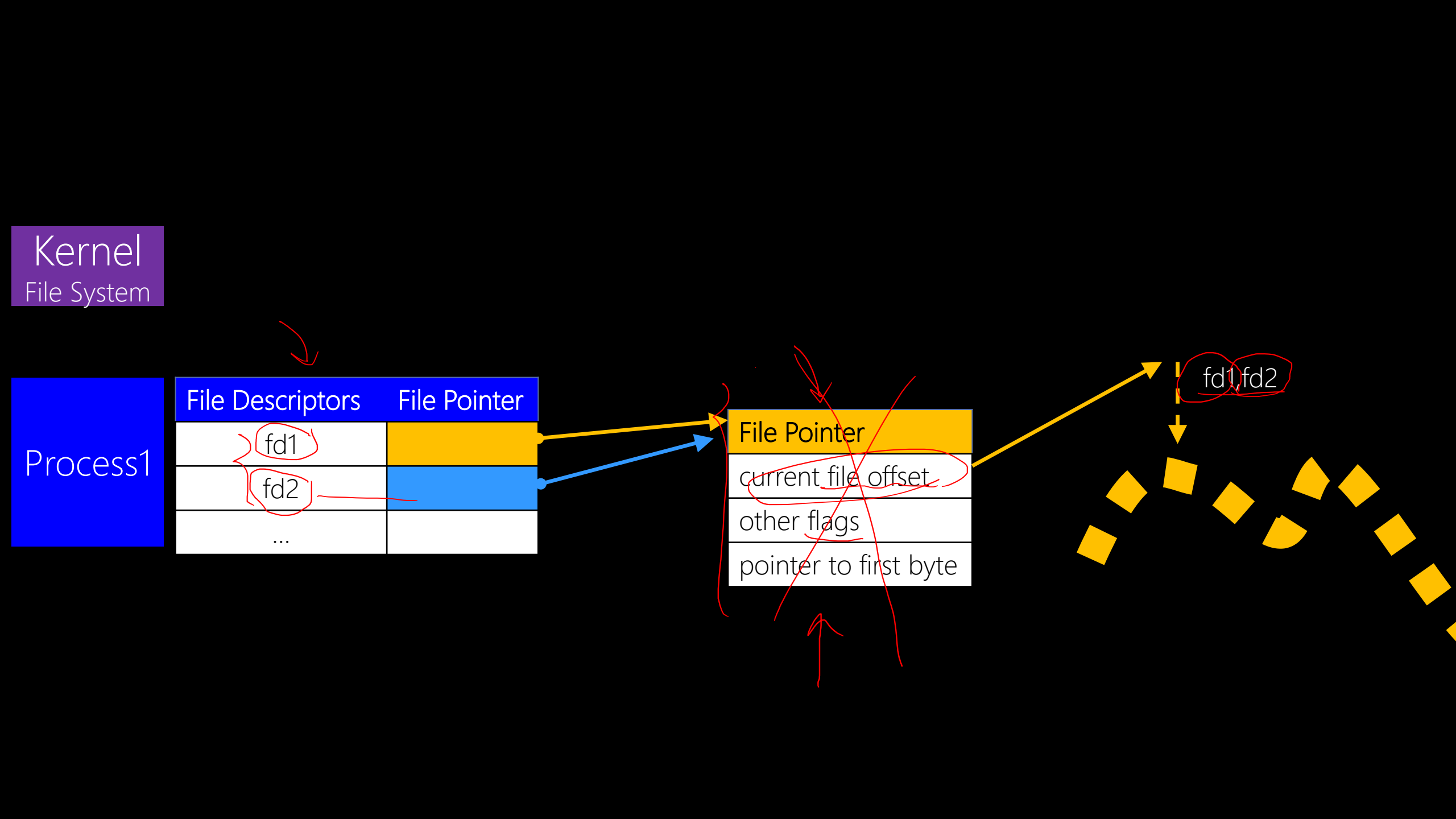
```
fd1 = open("test.txt", O_RDONLY)
fd2 = open("test.txt", O_WRONLY)
```

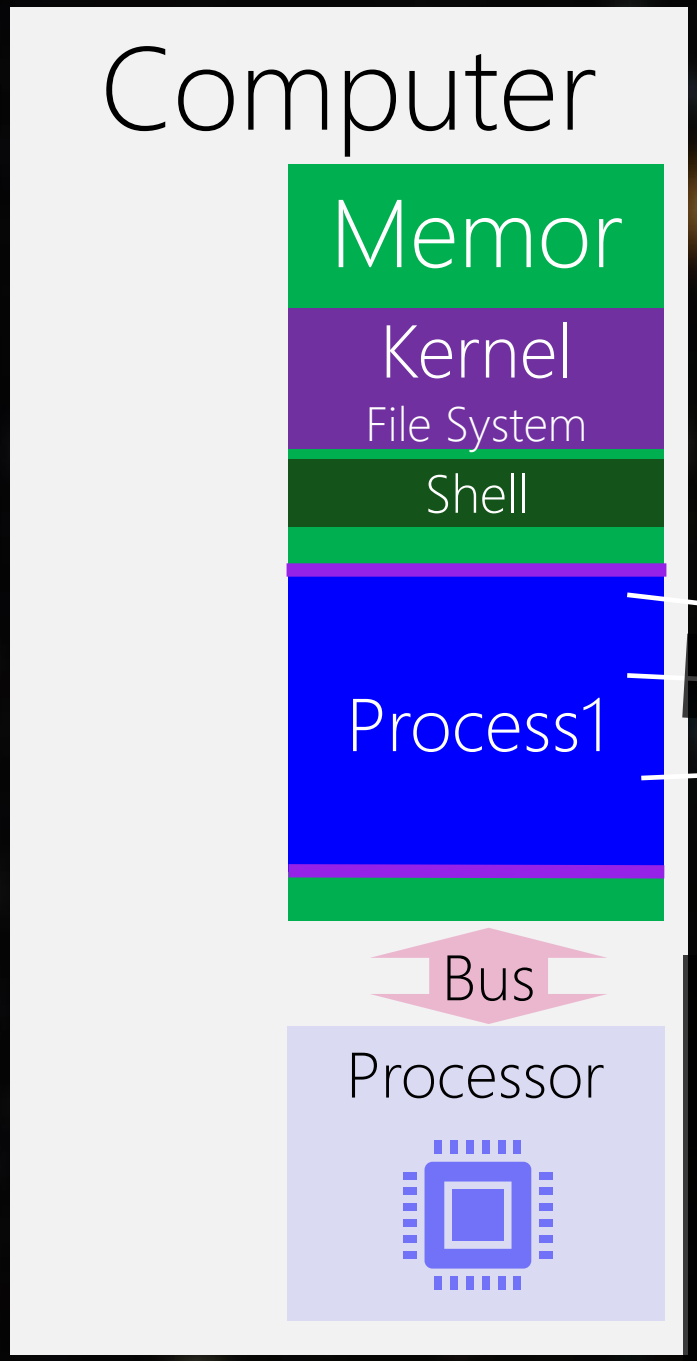

Kernel
File System
Process1
File Descriptors
File Pointer
fd1
fd2
...
File Pointer
current file offset
other flags
RD
pointer to first byte
File Pointer
current file offset
other flags
WR
pointer to first byte
fd1
fd2
fd1
fd2

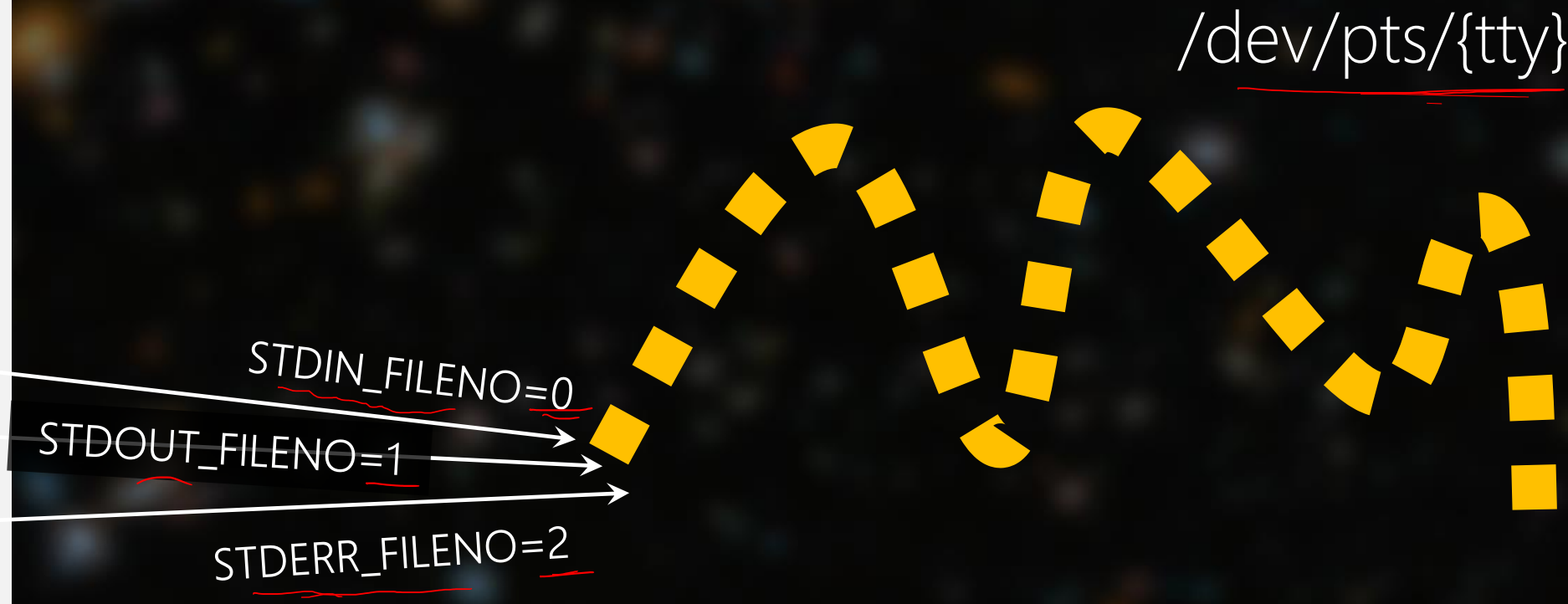
dup vs. multiple open

`dup()`: each returned fd has the same properties and current offset

```
fd2 = dup(fd1)
```


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





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

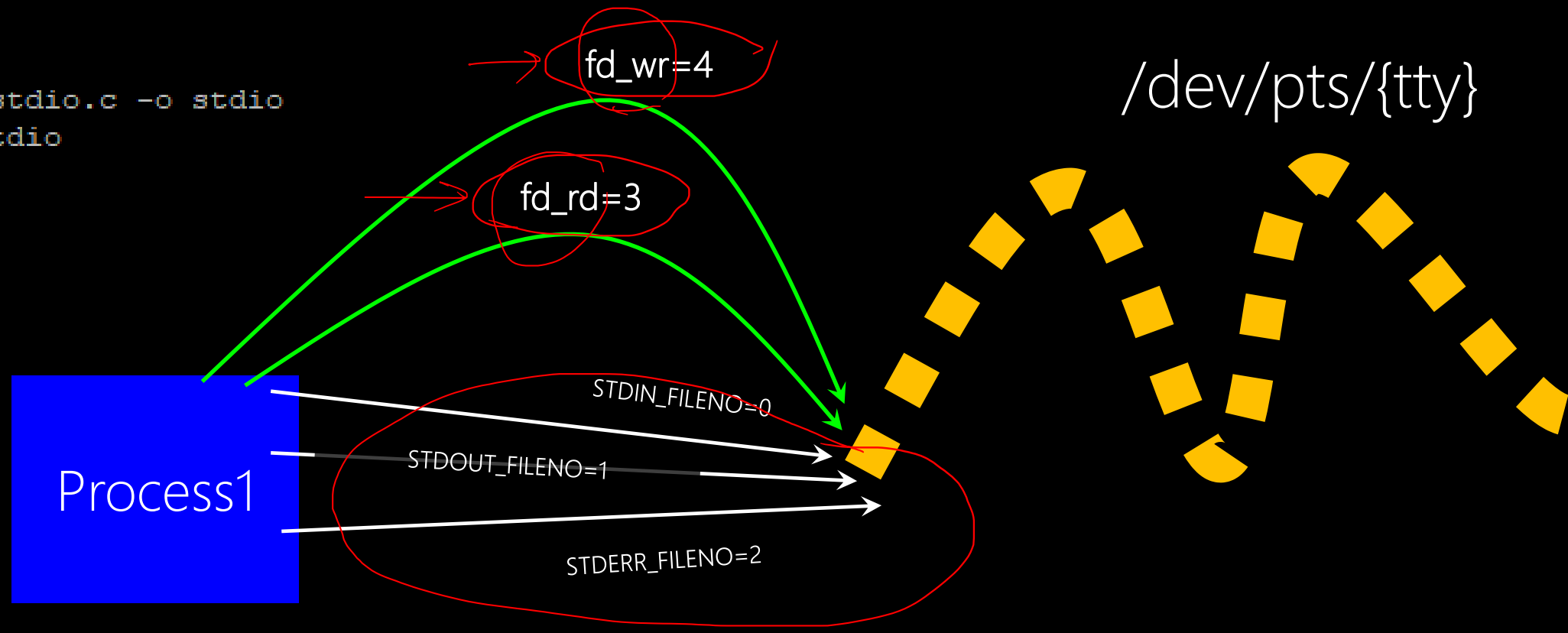
STDIN_FILENO = 0 : O_RDONLY
STDOUT_FILENO = 1 : O_WRONLY
STDERR_FILENO = 2 : O_WRONLY

```c
#include <fcntl.h>
#include <unistd.h>
void main(void){

 char buf_rd[20];

 int fd_rd = open("/dev/fd/0", O_RDONLY);
 int res = read(fd_rd, buf_rd, 10);
 int fd_wr = open("/dev/fd/1", O_WRONLY);
 write(fd_wr, buf_rd, 10);
```

```c
#include <fcntl.h>
#include <unistd.h>
void main(void){

 char buf_rd[20];

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

```c
#include <fcntl.h>
#include <unistd.h>
void main(void){

 char buf_rd[20];

 read(0, buf_rd, 10);
 write(1, buf_rd, 10);
```

```
hfani@charlie:~$ cc stdio.c -o stdio
hfani@charlie:~$ ./stdio
comp2560
comp2560
hfani@charlie:~$
```

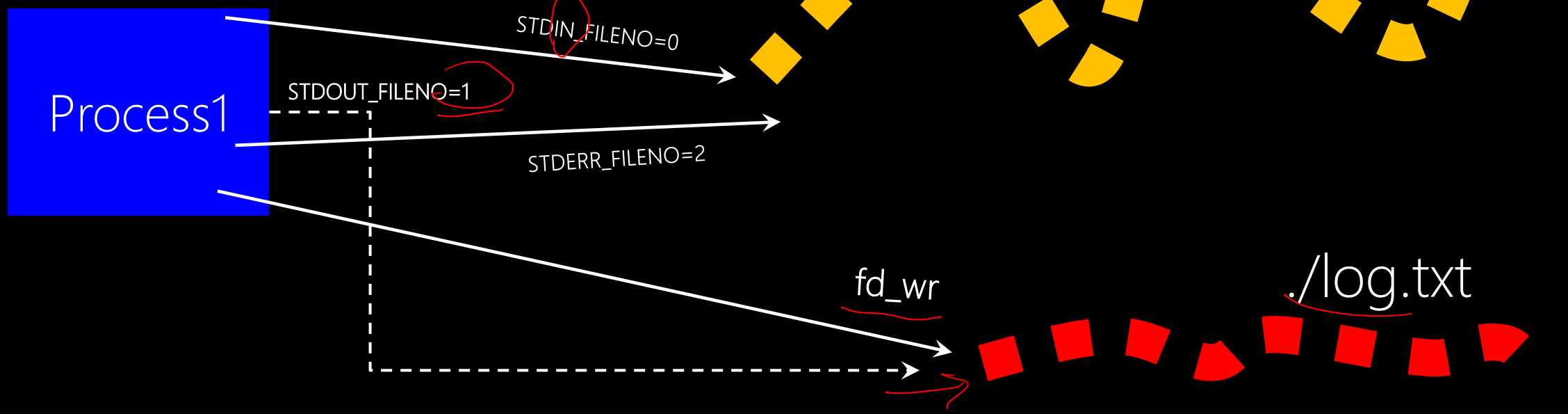


```c
#include <fcntl.h>
#include <unistd.h>
void main(void){
        char buf_rd[20];

        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
```

the lowest-numbered available file descriptor

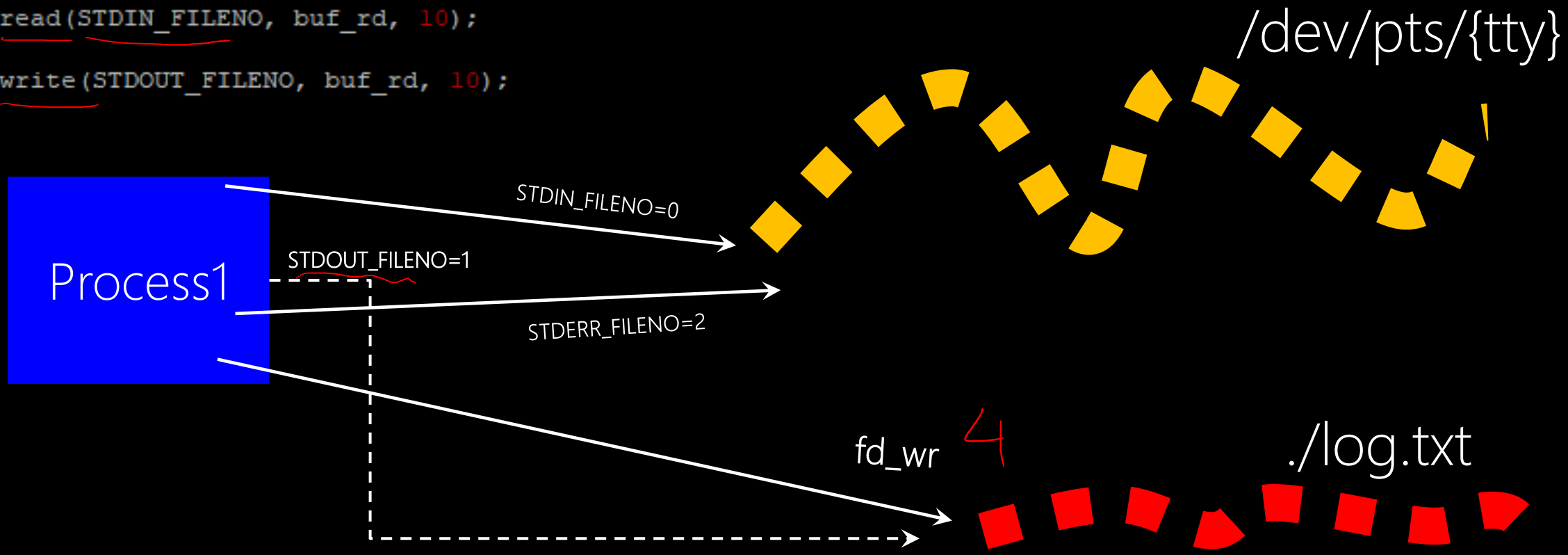
printf scanf

dup2



```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);
```

5



Shell

$ ./program > log.txt

We can ask the shell to do this redirection for us

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

Story

How about STDIN_FILENO? STDERR_FILENO?

Story

What does this mean and what is the benefit?

```
fd = dup(0)
```

dup2(fd1, fd2)

At Home

I/O Efficiency
buffered vs. unbuffered

System Call `unistd.h` vs. `stdio.h` Library Routine

At Home.

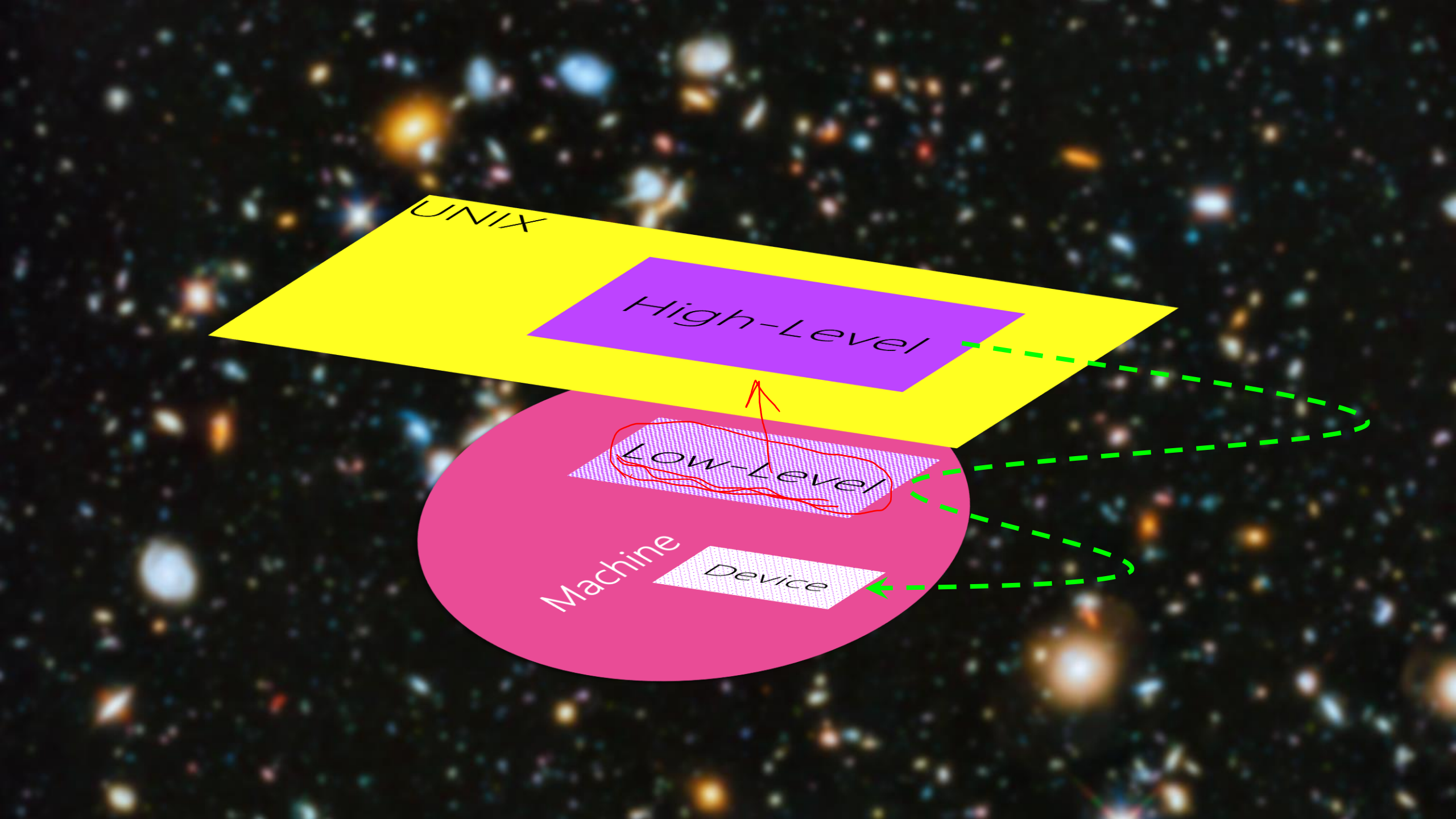
File Sharing

Advanced! We won’t cover it.

Atomic Operation

Advanced! We won’t cover it.

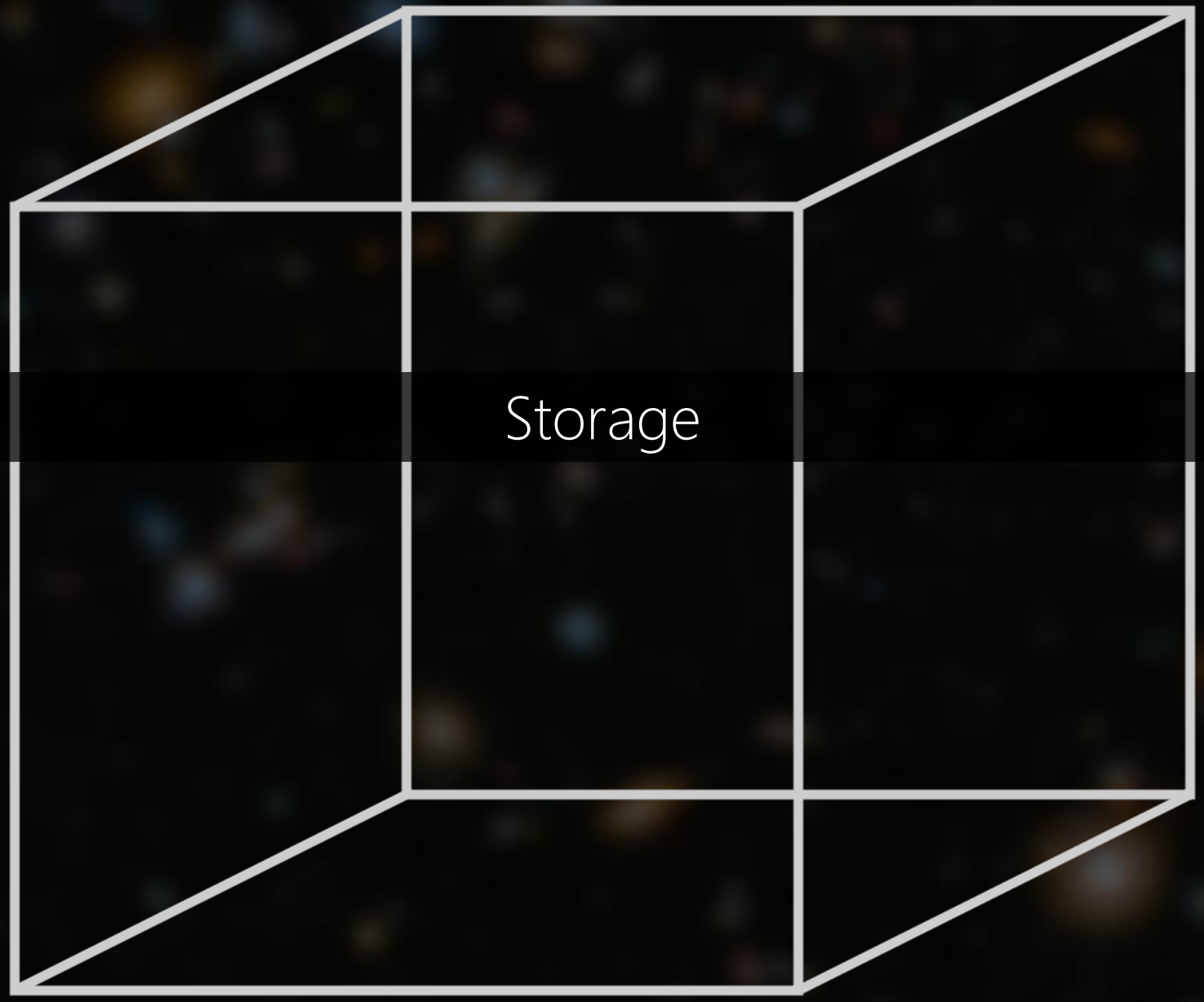
- Review File System `lseek`, `dup`, I/O Redirection
- File System for Storage Devices: `i-node`, Directories


UNIX
High-Level
Low-Level
Machine
Device

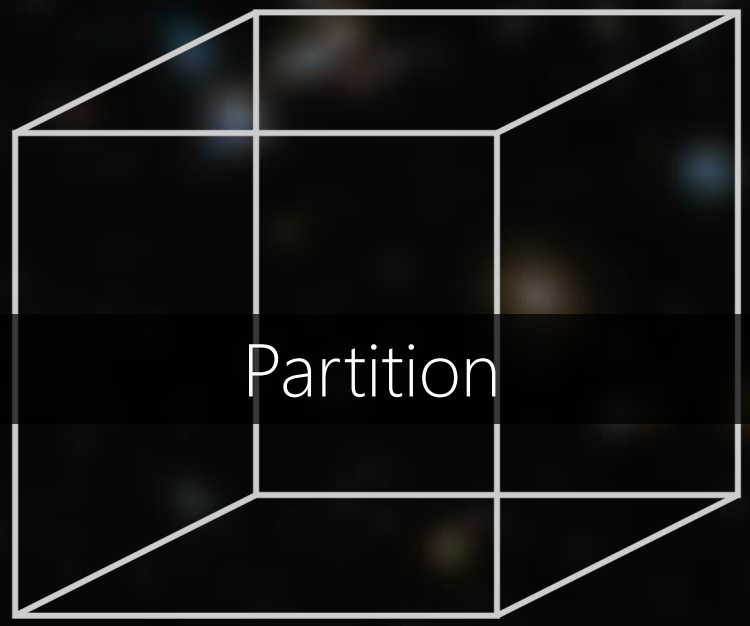
Storage File System

Disk File System

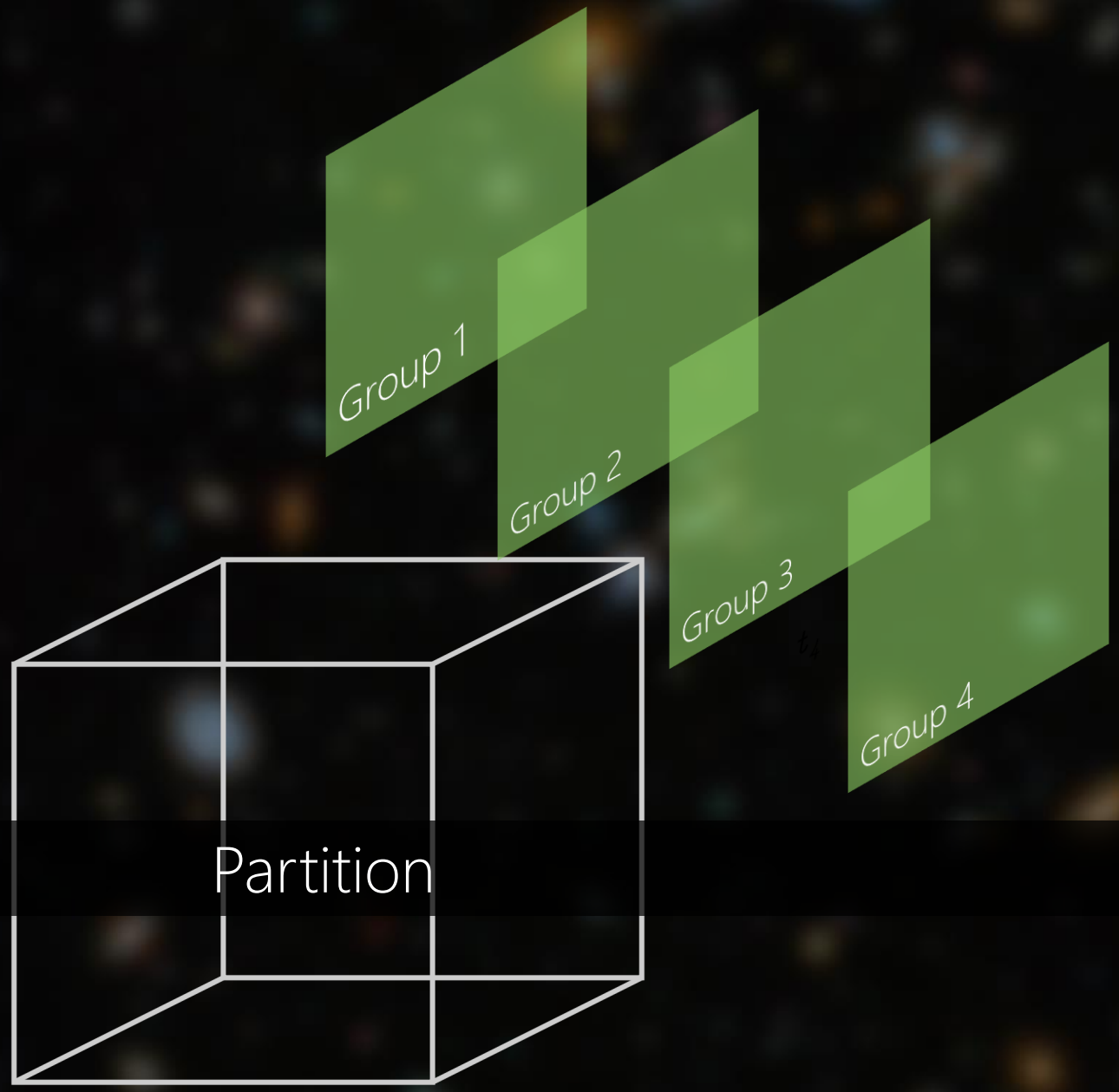
File System: Low Level

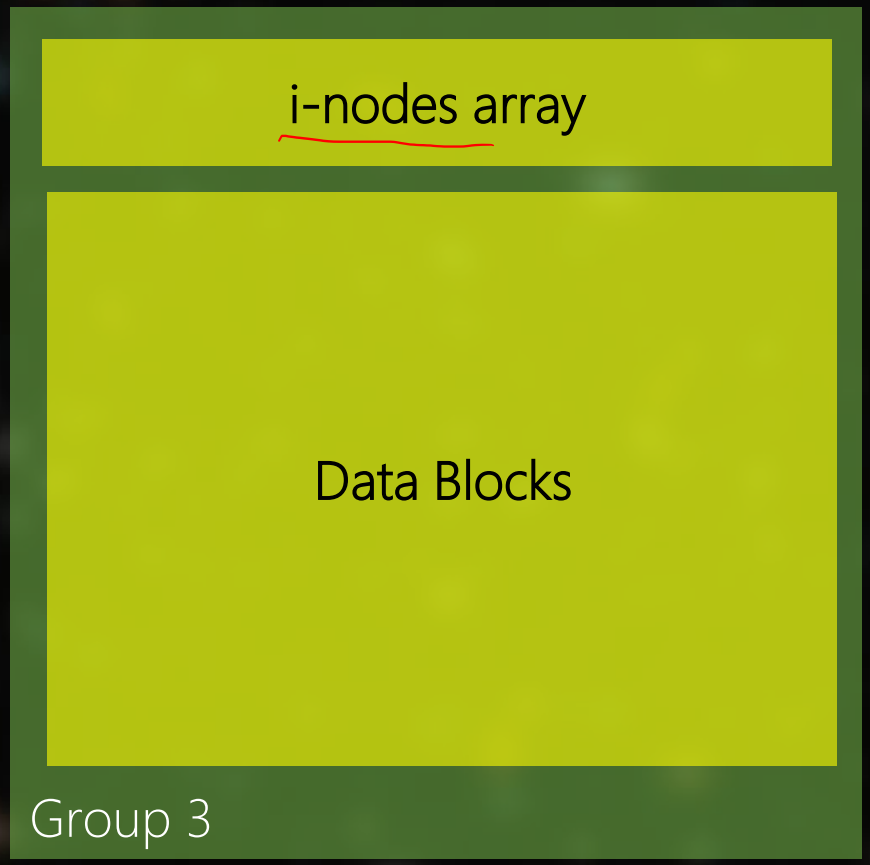

Storage

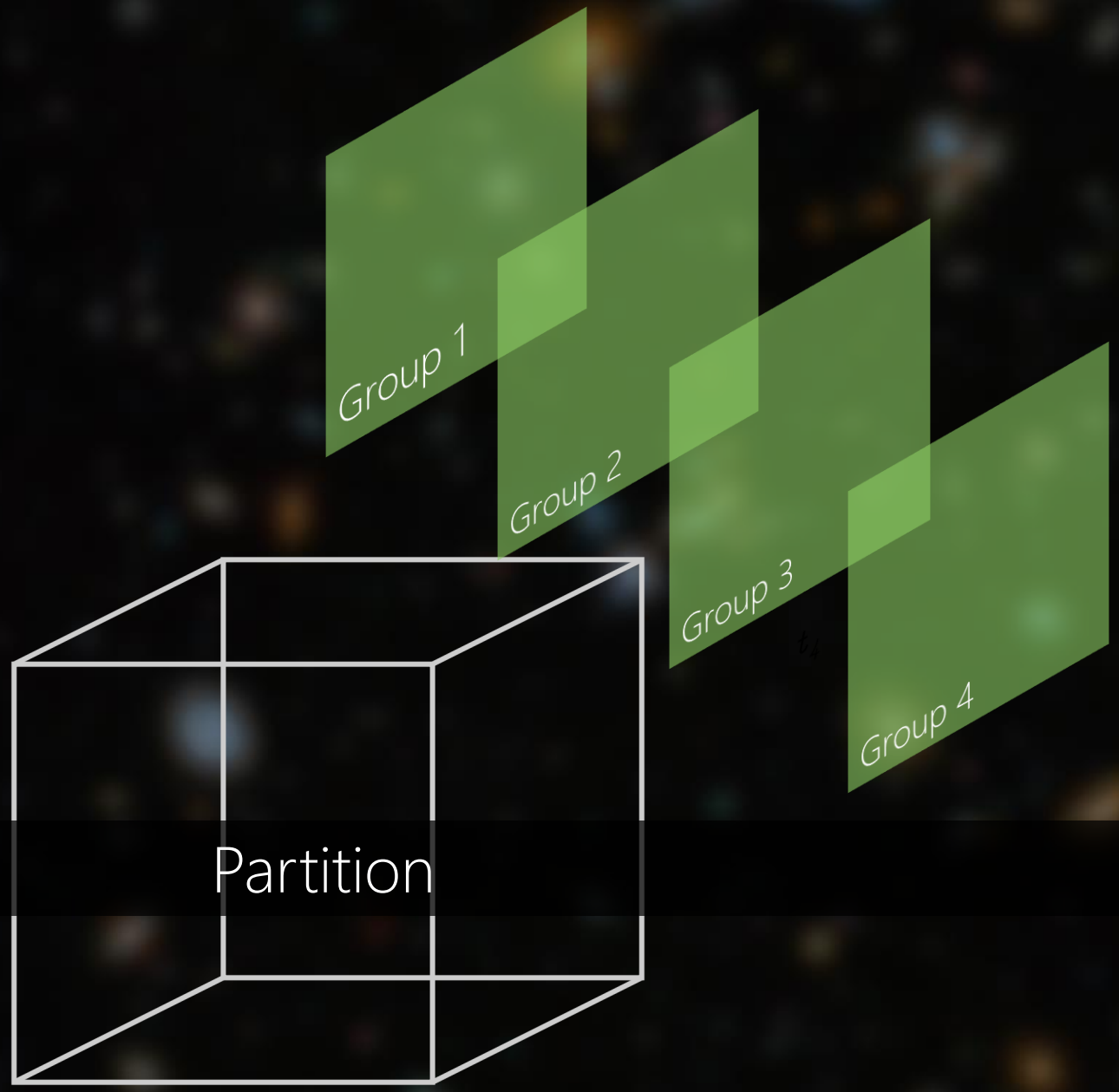
Partition

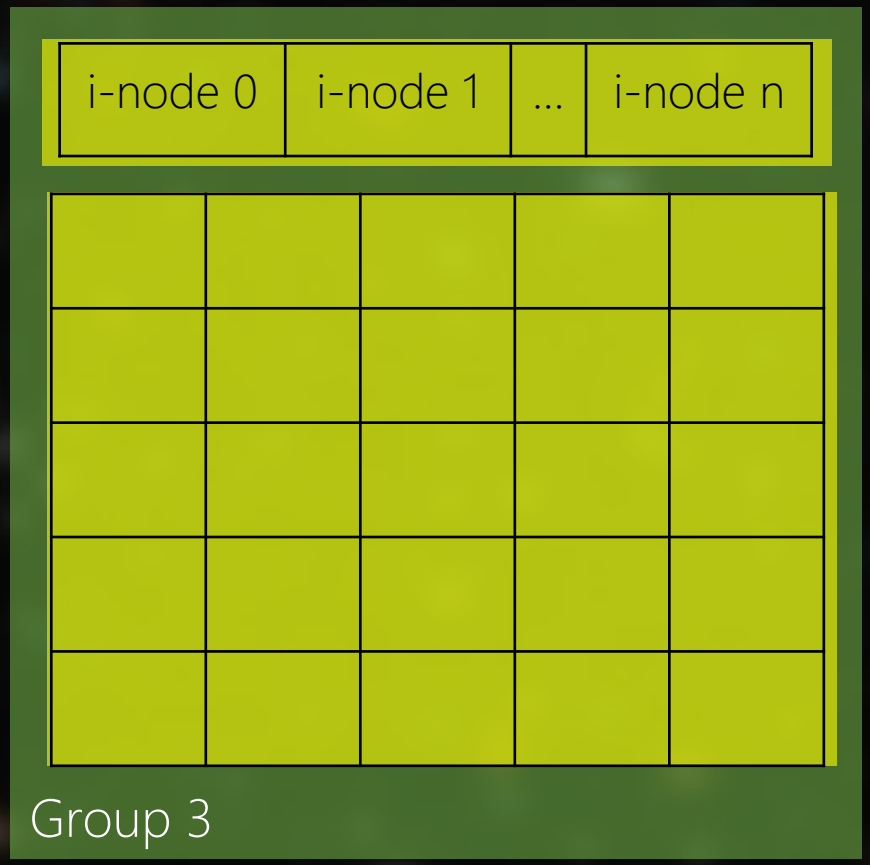


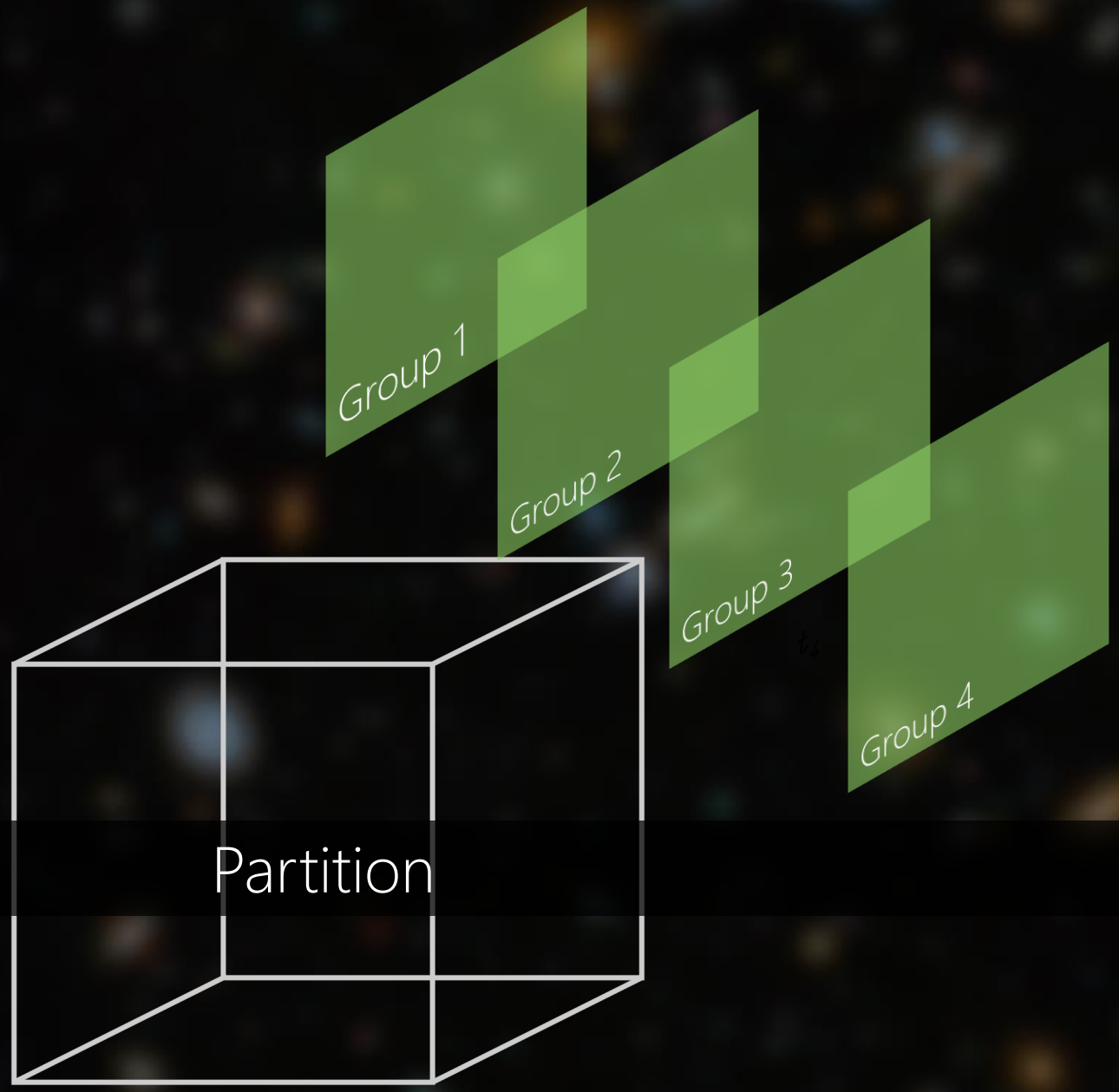
Partition

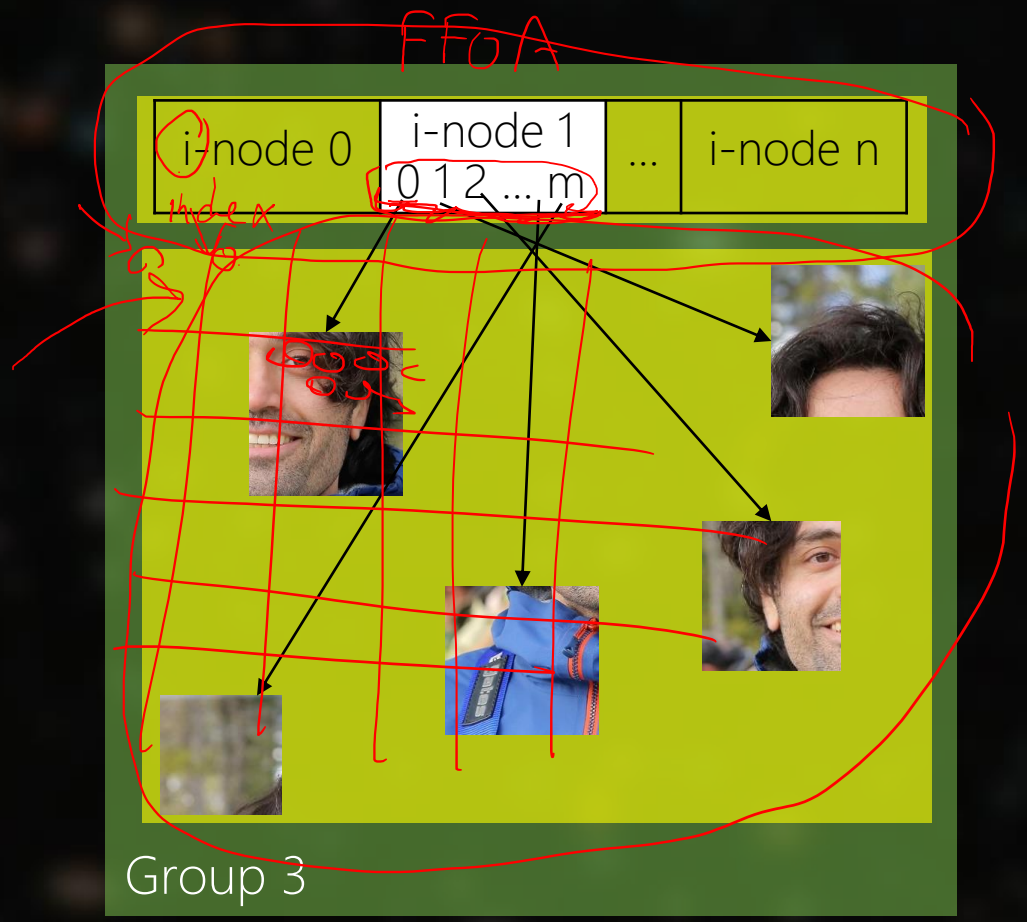

Group 1
Group 2
Group 3
Group 4
Partition


i-nodes array
Data Blocks
Group 3


Group 1
Group 2
Group 3
Group 4
Partition


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


Group 1
Group 2
Group 3
Group 4
Partition


FFOA
i-node 0
i-node 1
0 1 2 ... m
...
i-node n
Group 3

Each File has one i-node

$ ls -i {filename}

```
hfani@bravo:~$ ls -i hfani.jpeg
99306609 hfani.jpeg
```

Each File has many data blocks

$ stat {filename}

```
hfani@bravo:~$ stat hfani.jpeg
  File: hfani.jpeg
  Size: 1552          Blocks: 5          IO Block: 1048576 regular file
Device: 2ch/44d Inode: 99306609    Links: 1
Access: (0644/-rw-r--r--)  Uid: (239080/   hfani)   Gid: (  400/    temp)
Access: 2022-09-14 11:17:45.909419913 -0400
Modify: 2022-09-14 11:17:45.927970184 -0400
Change: 2022-10-19 00:16:57.563217031 -0400
 Birth: -
```

Byte

Can we `vi` an `i-node`?

NO

/Path/Filename → i-node → Data Blocks

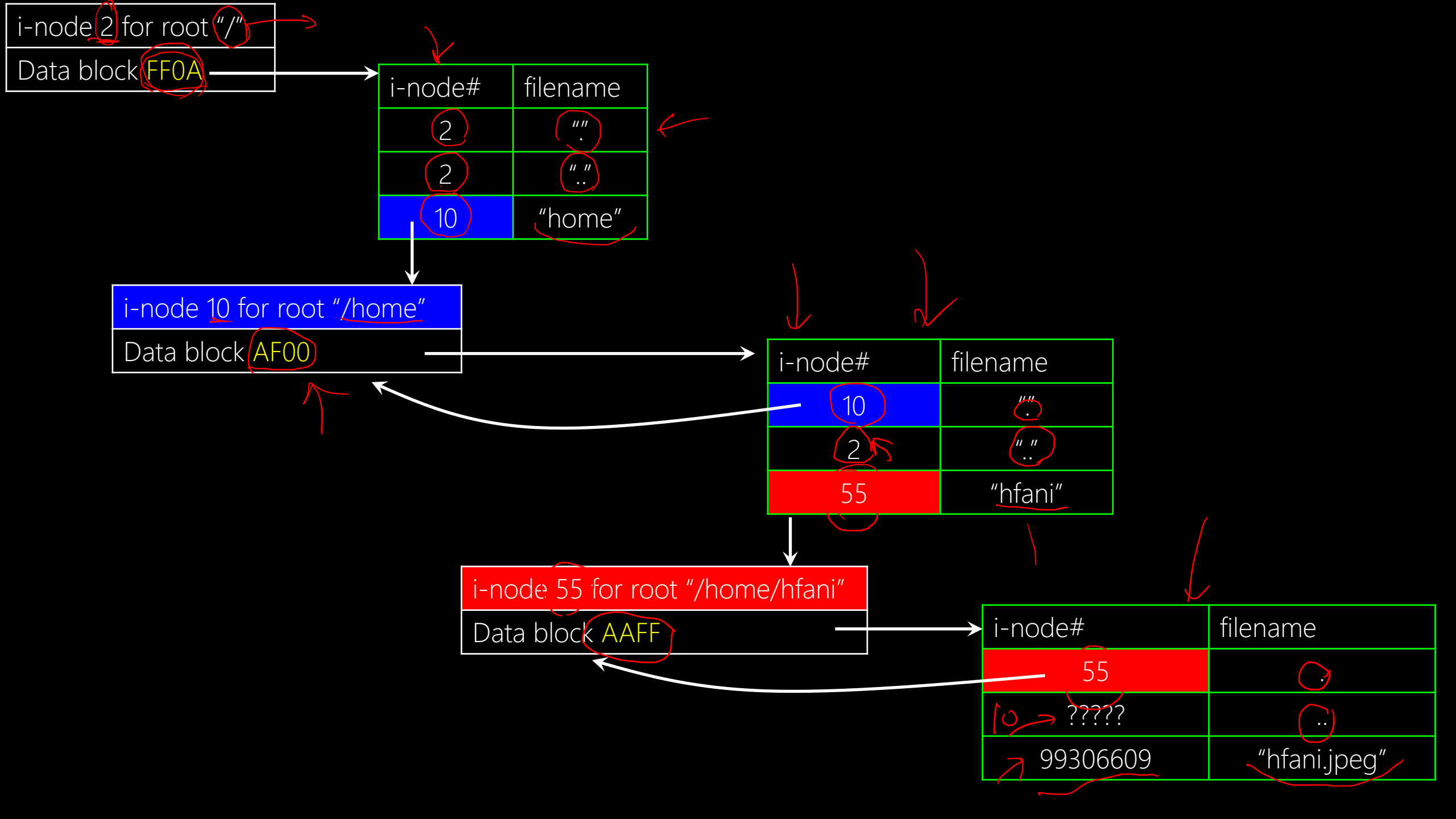
"/home/hfani/hfani.jpeg" → 99306609 → 

creat() or open()

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


i-node 2 for root "/"
Data block FF0A
i-node#
filename
2
"."
2
".."
10
"home"
i-node 10 for root "/home"
Data block AF00
i-node#
filename
10
"."
2
".."
55
"hfani"
i-node 55 for root "/home/hfani"
Data block AAFF
i-node#
filename
55
.
10
?????
..
99306609
"hfani.jpeg"

Directories

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

"/home/hfani/hfani.jpeg" → 1407374883666816 →

"/nosseinfani.png"

No Duplication!
Still one single file.
Two or more links.
hardlink, shortcut, ...

i-node 2 for root /
Data block FF0A

i-node#	filename
2	.
2	..
99306609	hosseinfani.png
10	

/home/fa

i-node 10 for root '/home'
Data block AF00

/home

i-node#	filename
10	.
2	..
55	hfani

993 hfa

i-node 55 for root '/home/hfani'
Data block AAFF

i-node#	filename
55	.
?????	..
99306609	hfani.jpeg

What happens when you delete a file?

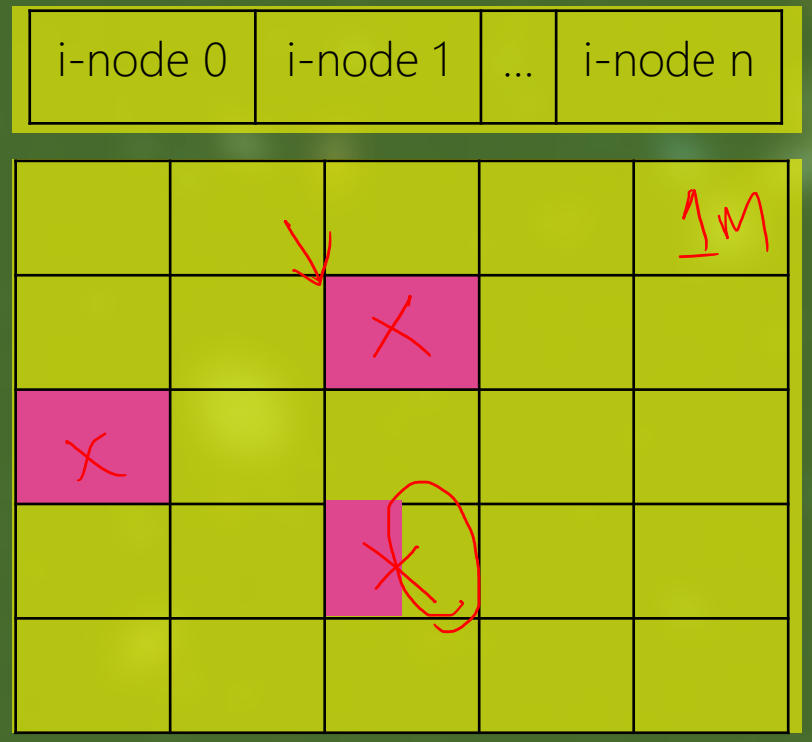
What happens when you move a file?

What happens when you copy a file?

Who determines the #partitions, #groups, data block size?

Partitioning (Formatting)

Large Data Blocks vs Small Data Blocks

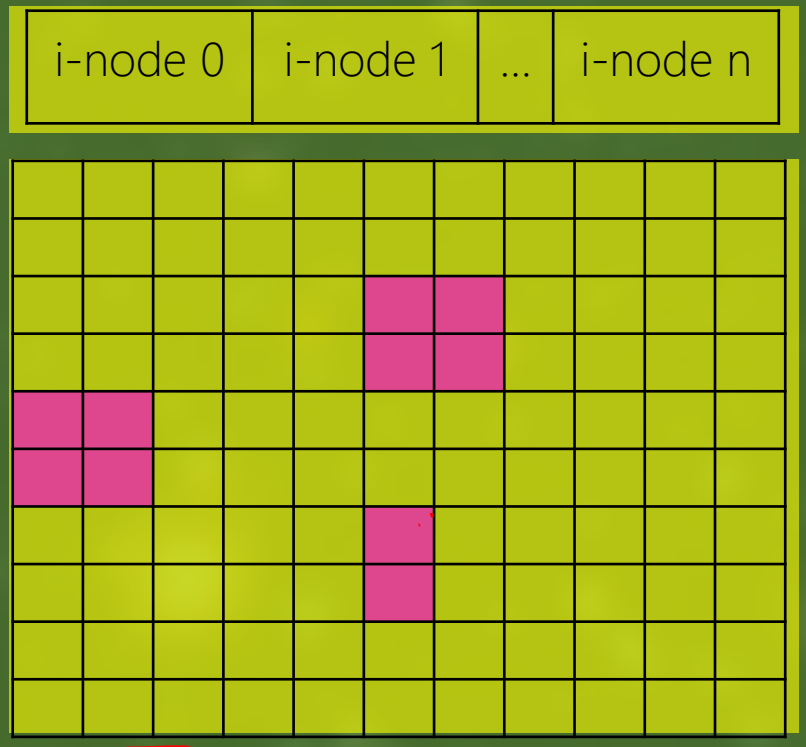


3 × 1M Blocks!

2.5M

3M

2.5M

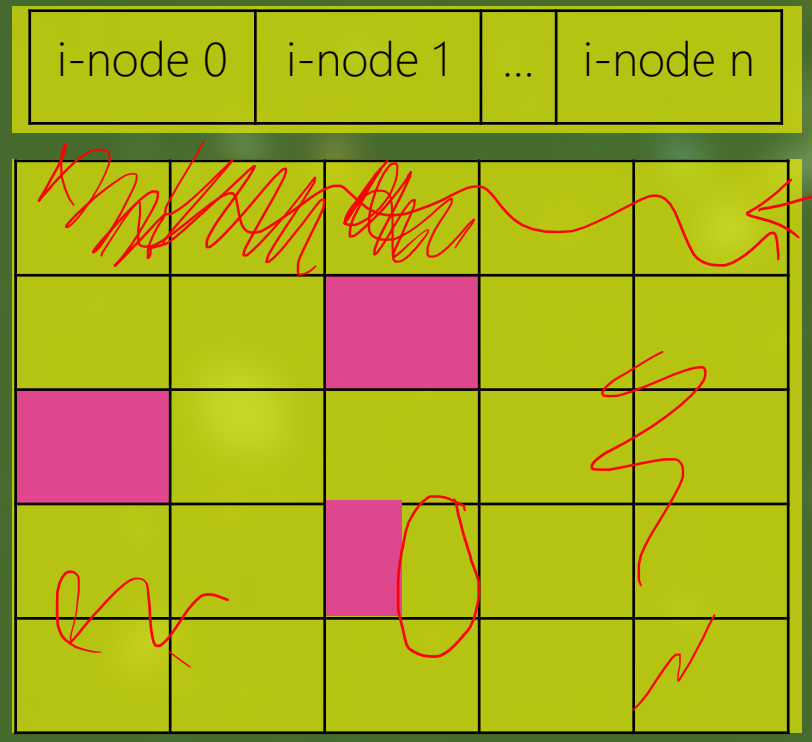


10 × 256K Blocks.

Large Data Blocks vs Small Data Blocks

Internal vs. External Fragmentation

Small vs. Large i-node



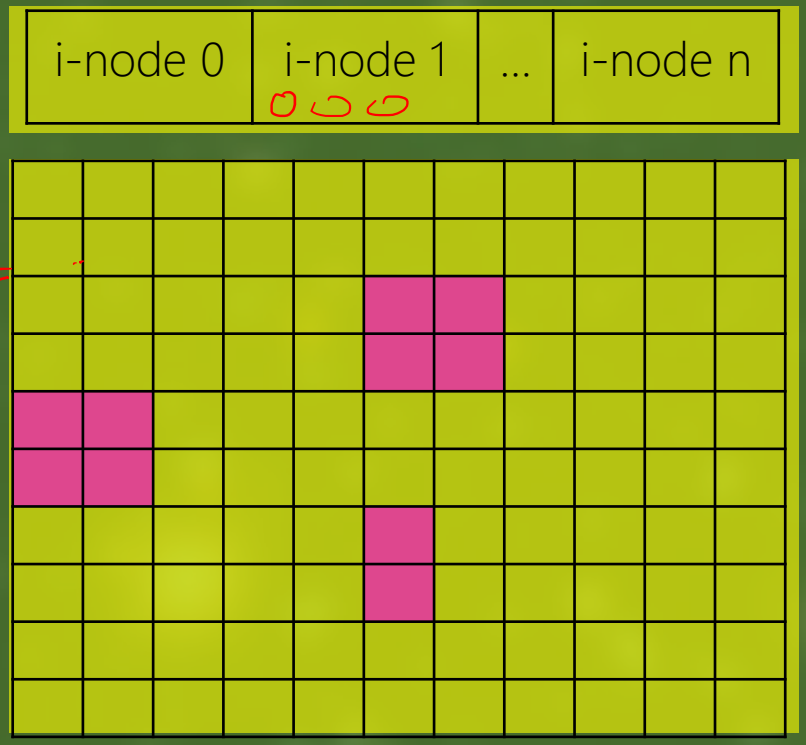
3 × 1M Blocks!

Movies?

Documents?

2.3 GB

256K



10 × 256K Blocks.

File Types (High-Level to Low-Level)

- 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

cat sda1