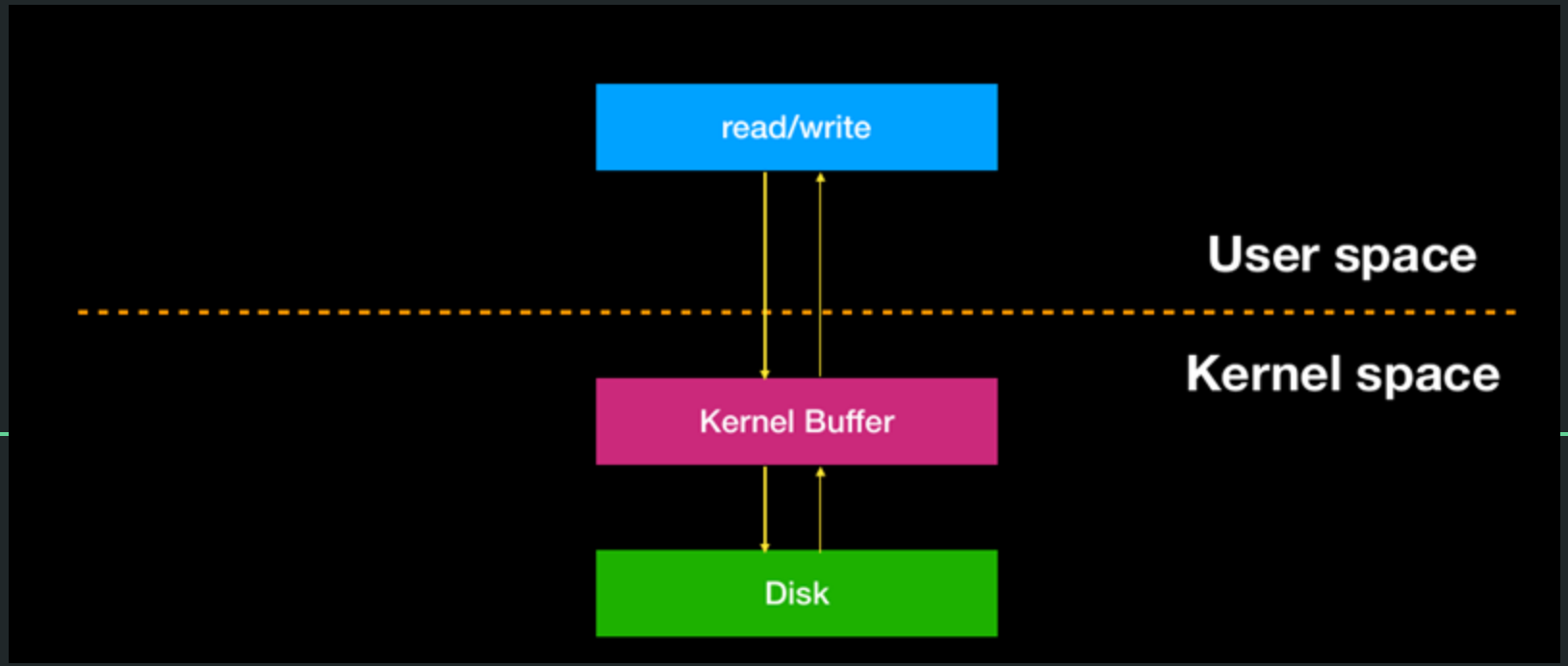
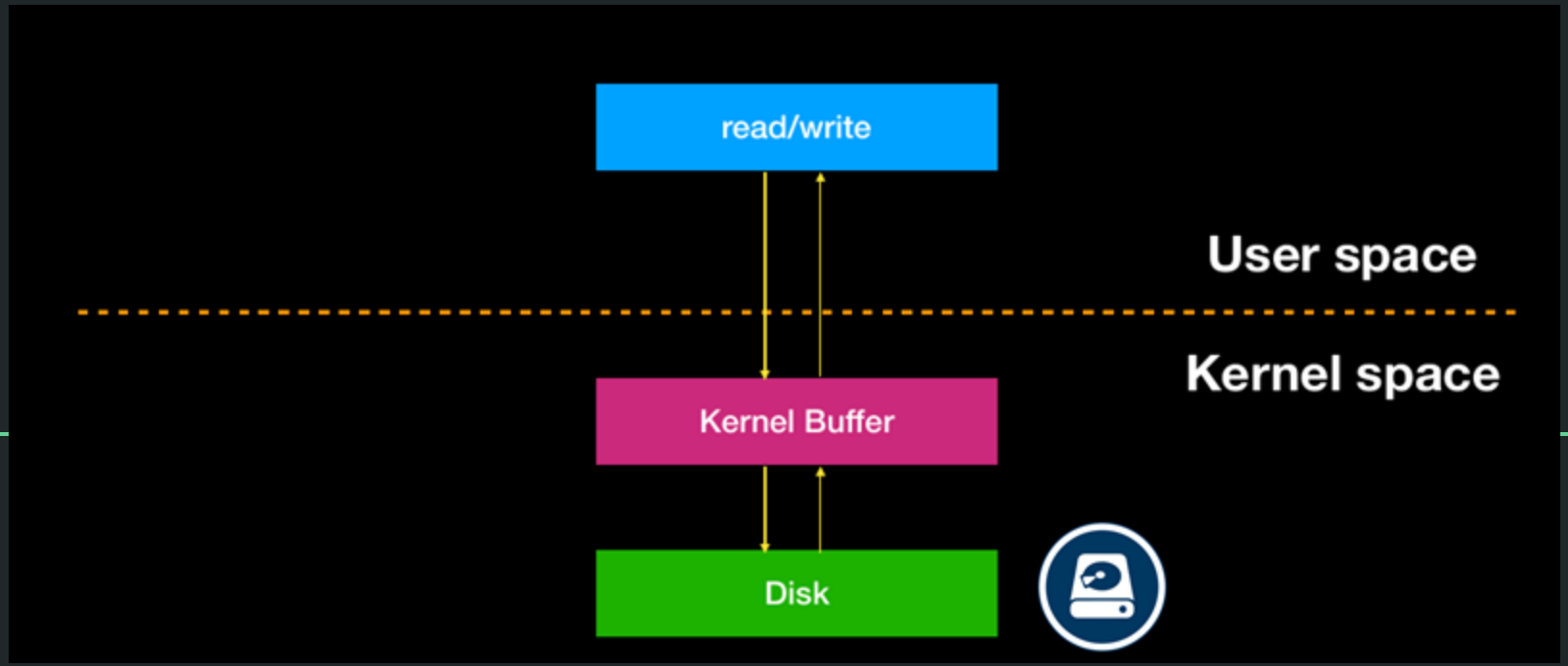
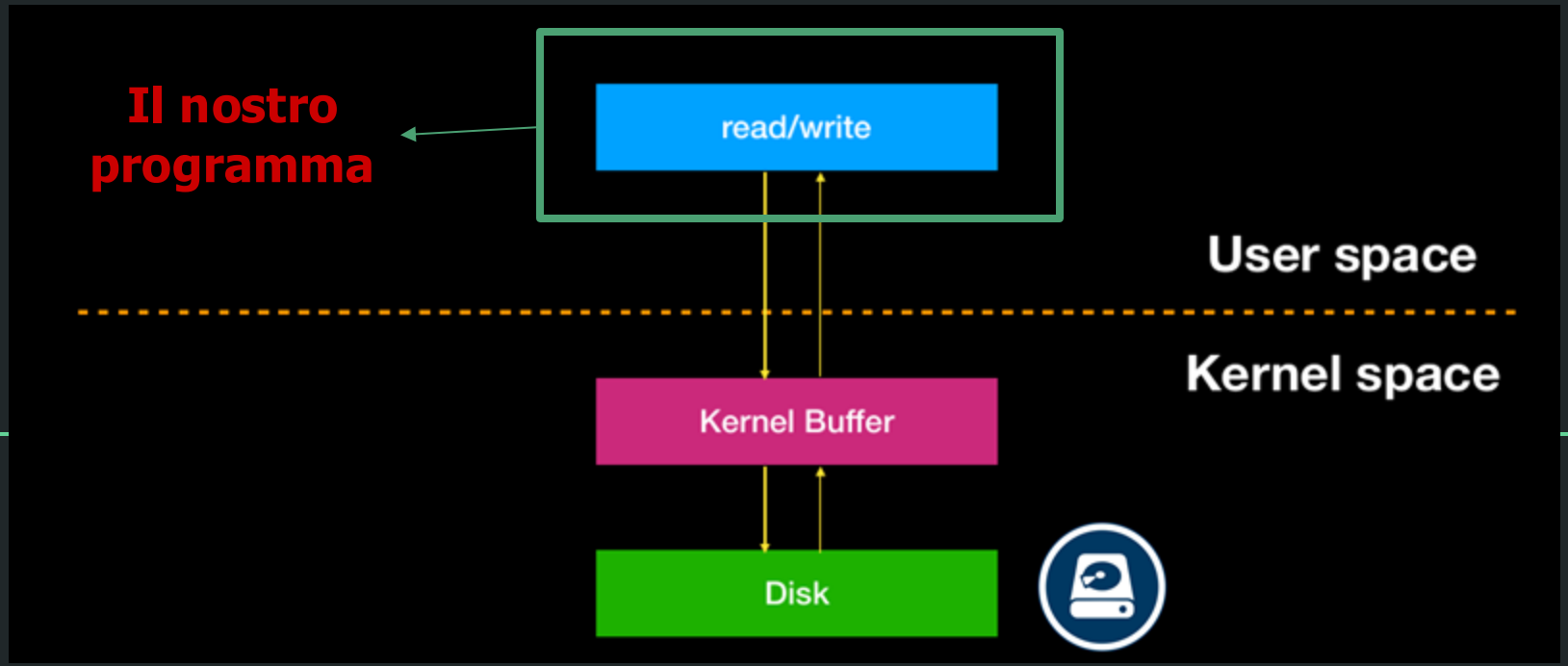
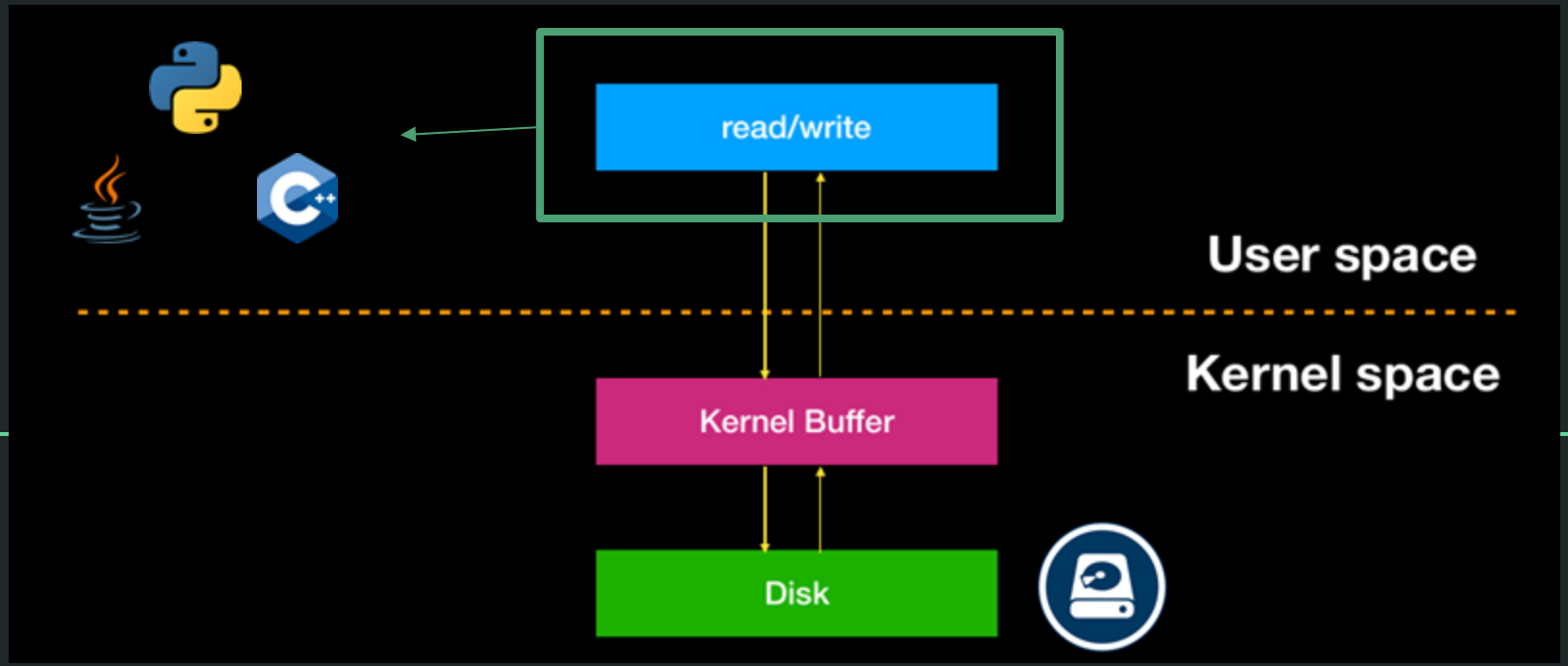
# I FILE

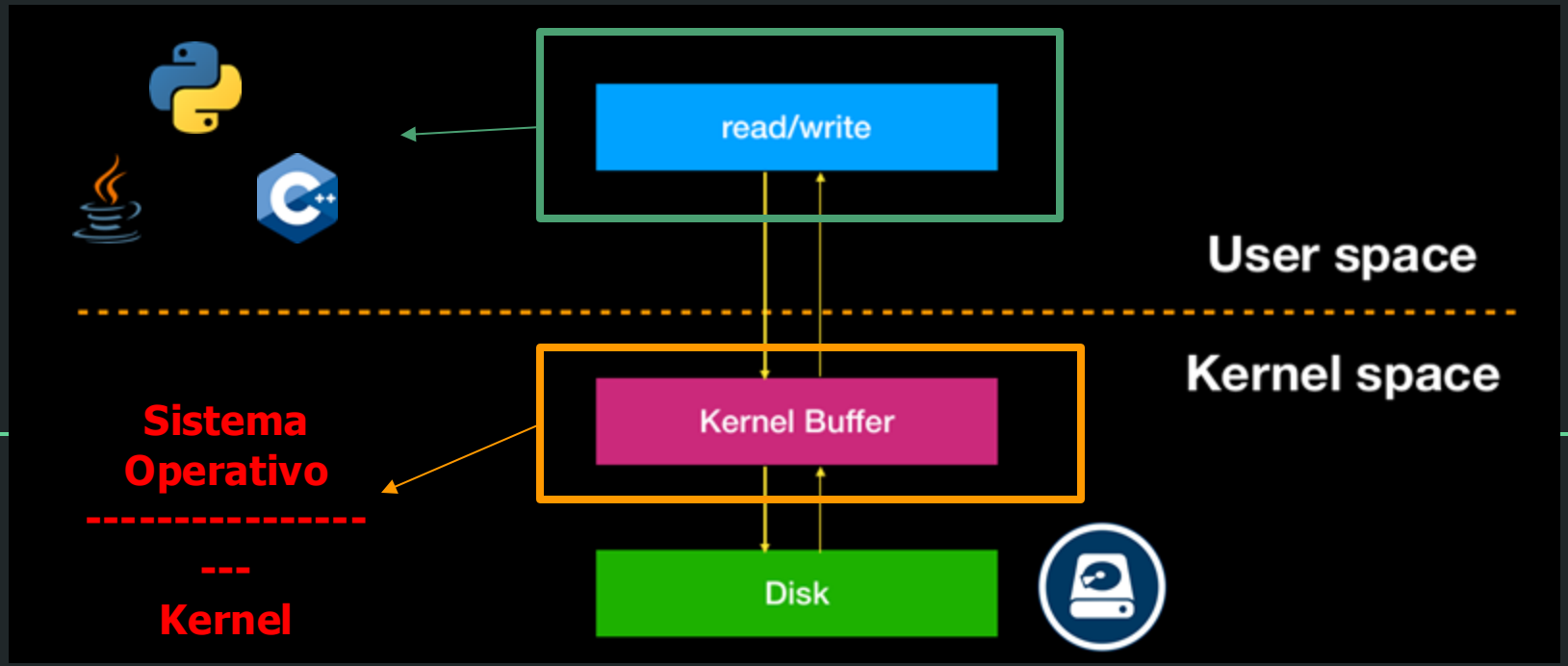
e come vengono gestiti.

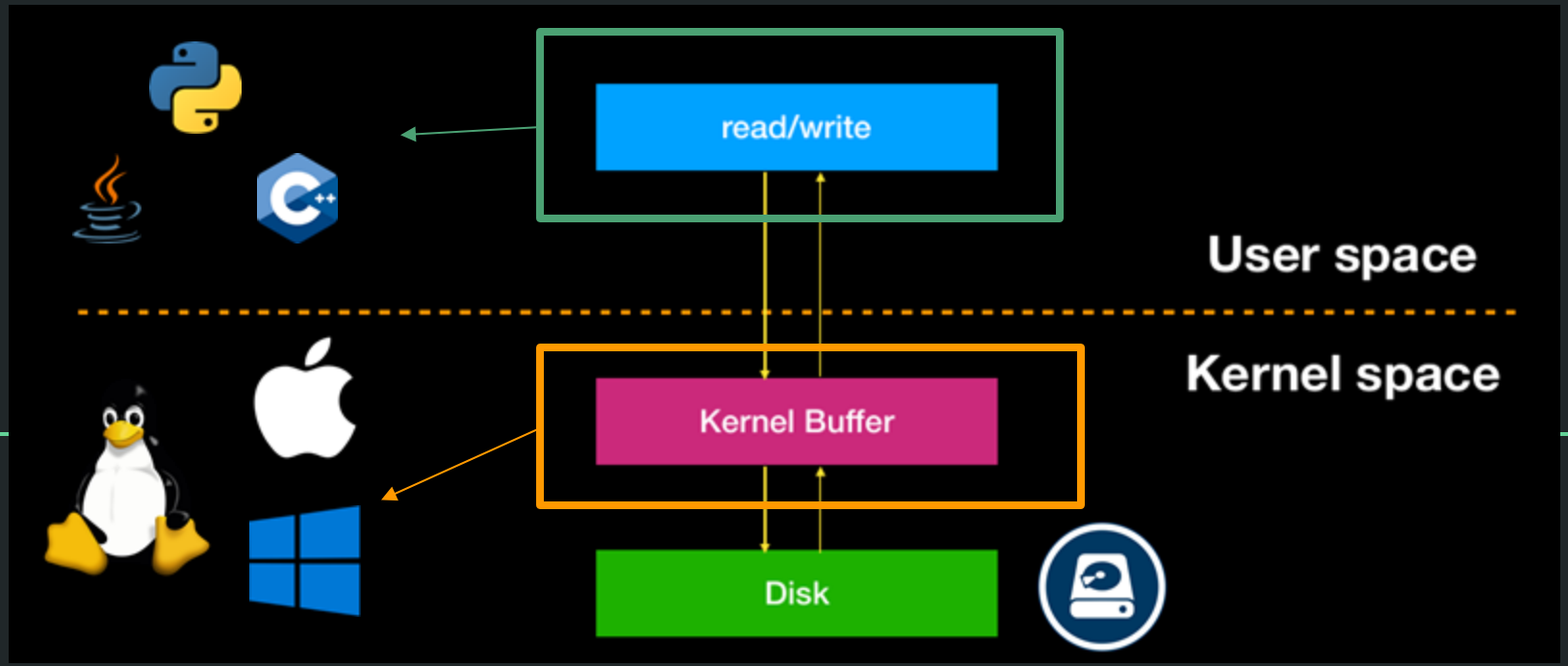

read/write
User space
Kernel space
Kernel Buffer
Disk

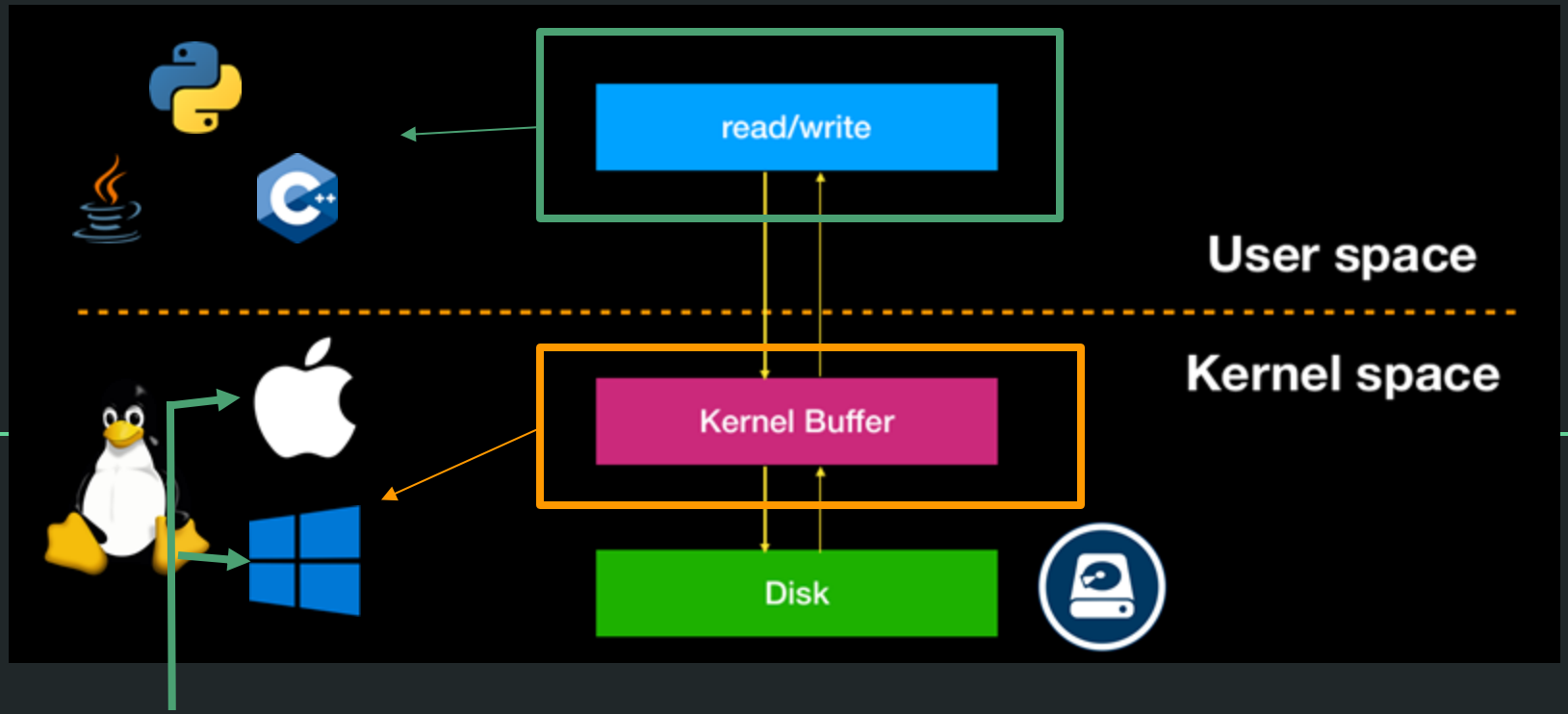

read/write
User space
Kernel space
Kernel Buffer
Disk


Il nostro programma
read/write
User space
Kernel space
Kernel Buffer
Disk

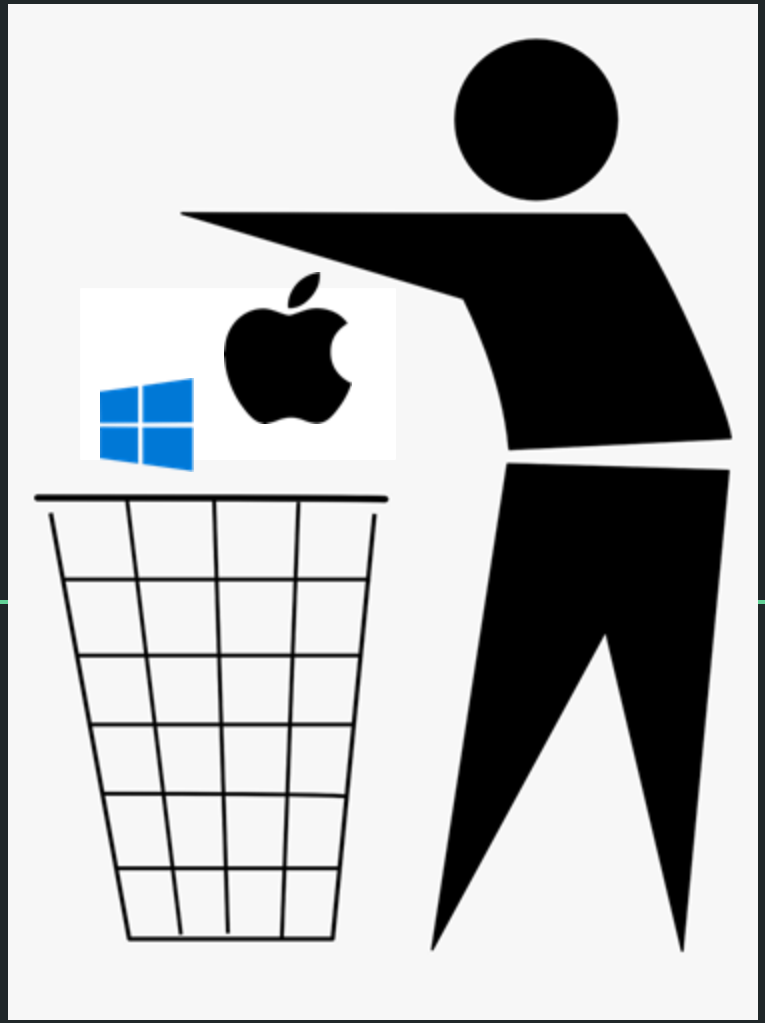

read/write
User space
Kernel space
Kernel Buffer
Disk

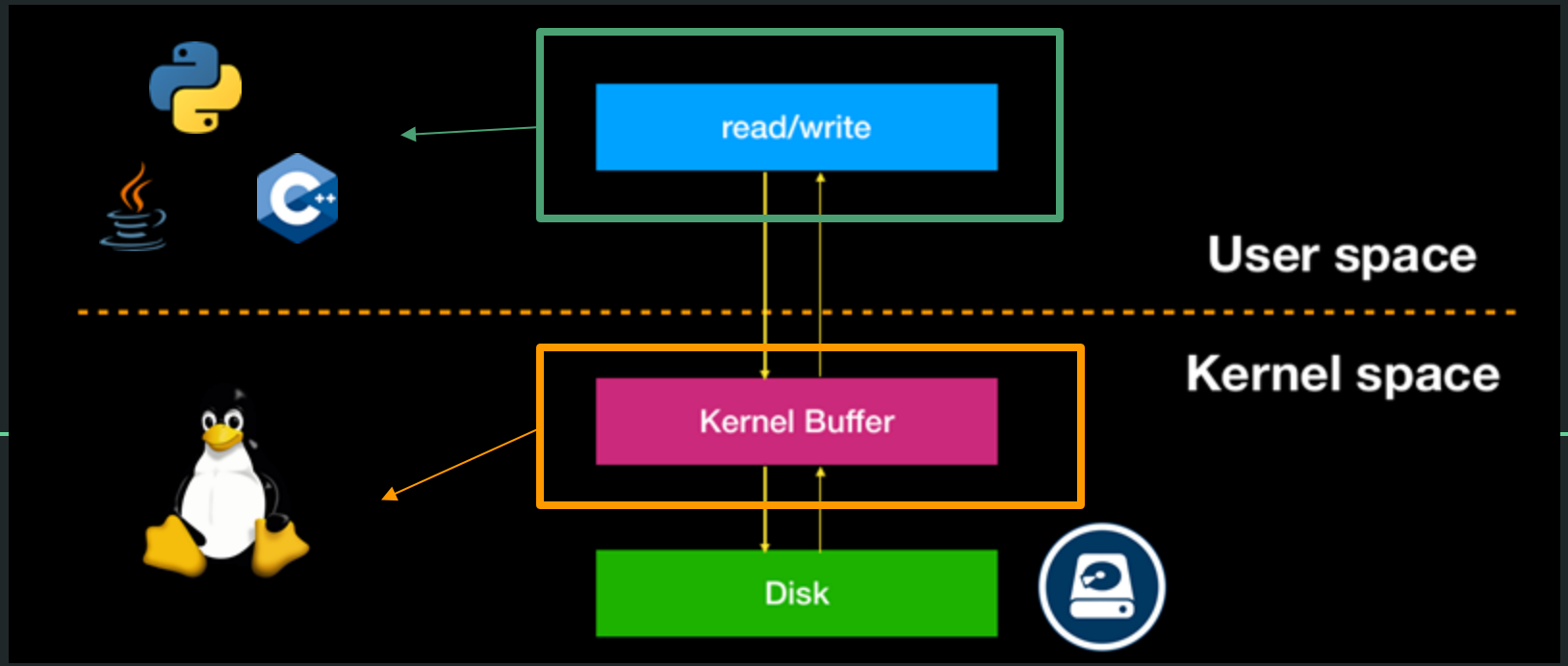

read/write
User space
Kernel space
Kernel Buffer
Disk
Sistema Operativo
---
Kernel


read/write
User space
Kernel space
Kernel Buffer
Disk

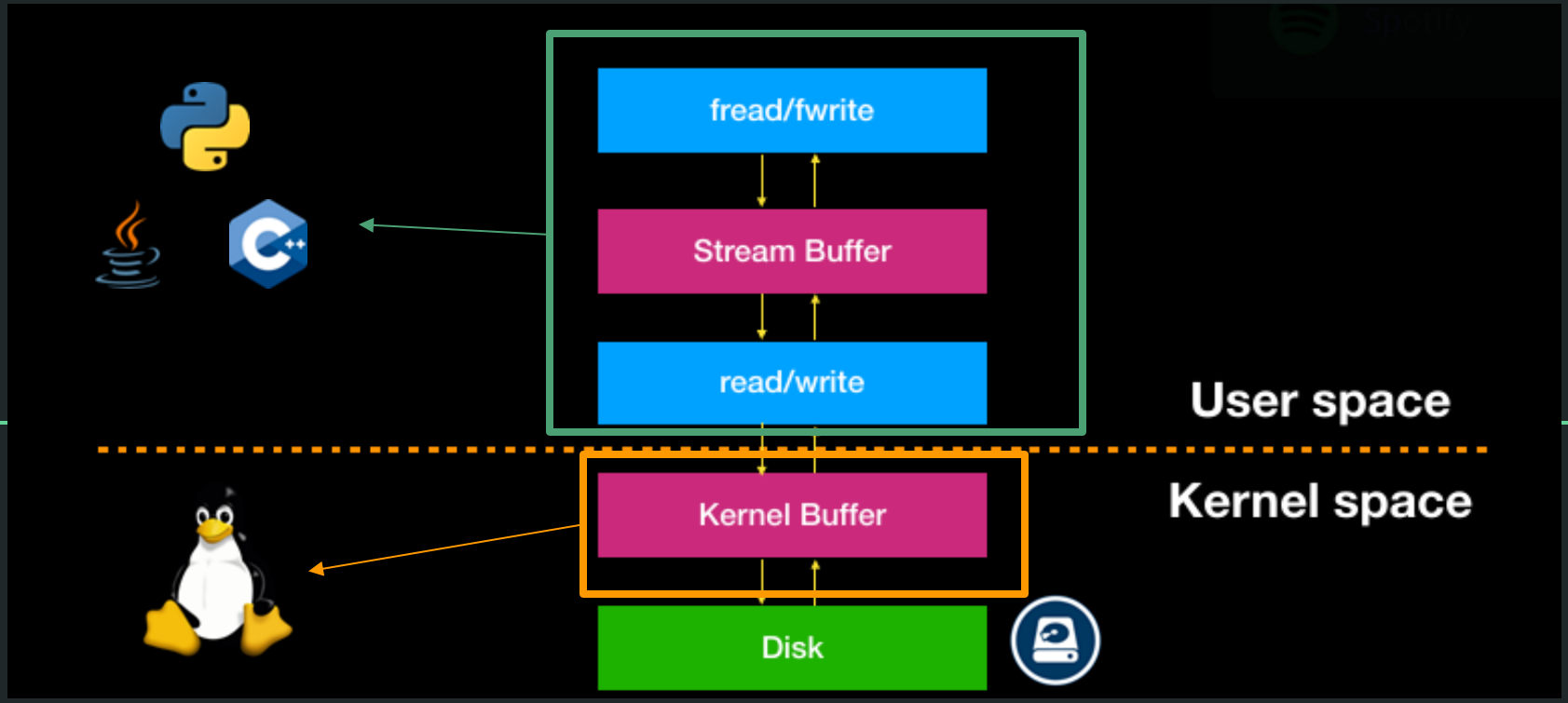



**Non conosciamo bene il funzionamento del kerner**




read/write
User space
Kernel space
Kernel Buffer
Disk

# ENTRIAMO NEL DETTAGLIO

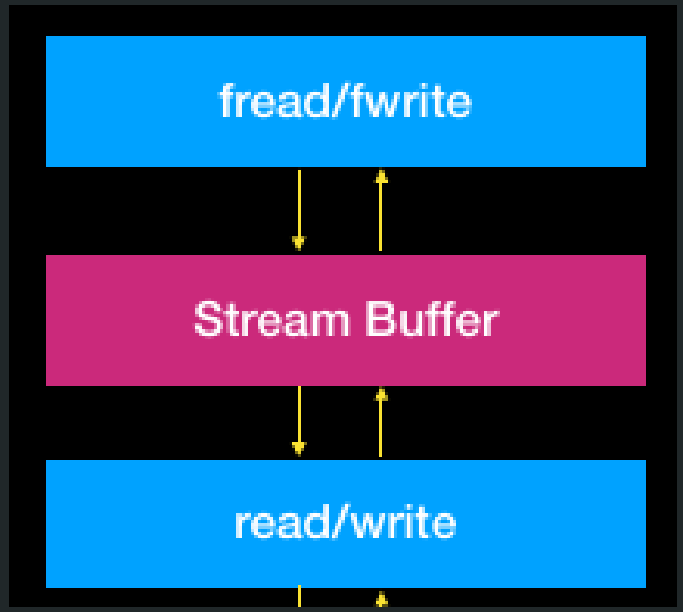

fread/fwrite
Stream Buffer
read/write
User space
Kernel space
Kernel Buffer
Disk

# USER SPACE

**I file nei vari linguaggi**

```
f =
open("file.txt","wt")
f.write("Hello world!")
f.flush()
f.close()
```

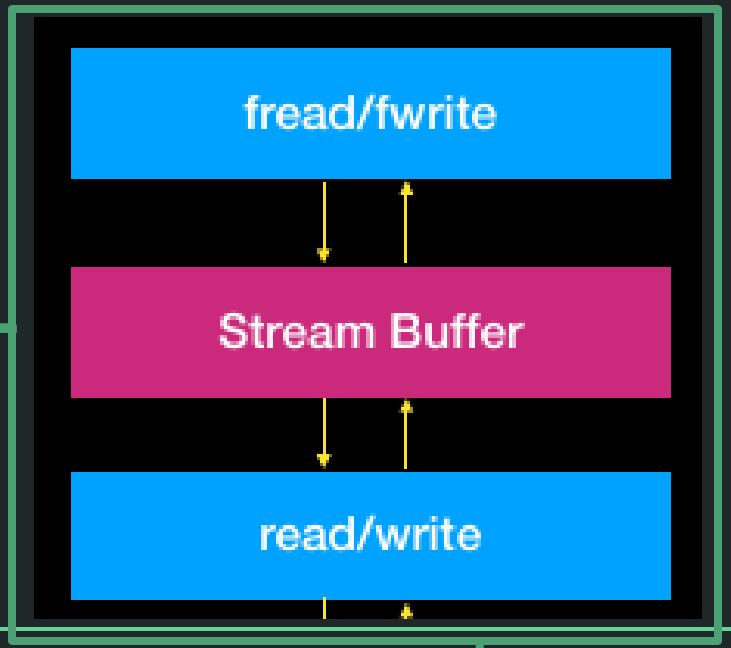
```
FILE * f;
f =
fopen("file.txt","wt");
fputs(f,"Hello World!");
fflush( f );
fclose( f );
f = NULL;
```




```
FileWriter f = new FileWriter("file.txt");
f.write("Hello World!");
f.flush();
f.close();
```

```
f =
open("file.txt","wt")
f.write("Hello world!")
f.flush()
f.close()
```

```
FILE * f;
f =
fopen("file.txt","wt");
fputs(f,"Hello World!");
fflush( f );
fclose( f );
f = NULL;
```

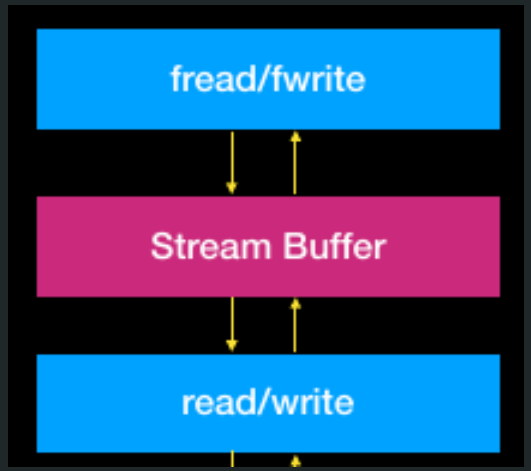



```
FileWriter f = new FileWriter("file.txt");
f.write("Hello World!");
f.flush();
f.close();
```

**Tutte le strutture offerte nei vari linguaggi offrono tutte le funzionalità descritte nell'immagine.**

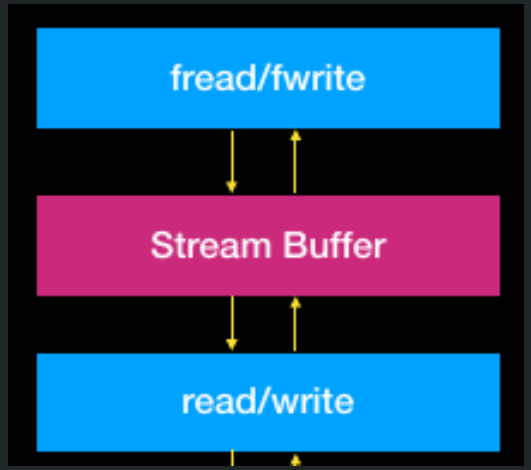
# USER SPACE

**Cosa succede?**



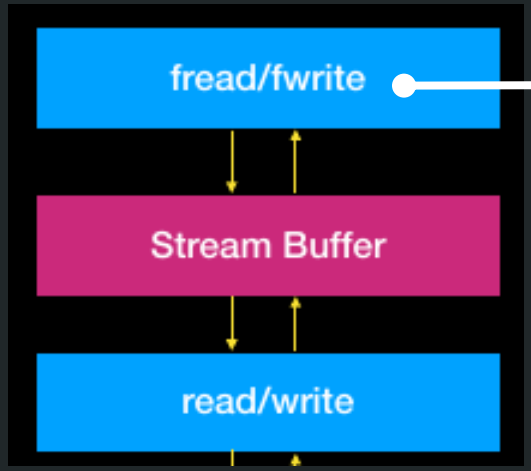

**KERNEL**

**Consideriamo che il file sia stato già aperto, quindi abbiamo un "mezzo trasmissivo" che ci permette di leggere e scrivere sul file tramite il kernel, che ci offre accesso alla memoria di massa**




**Ora vogliamo scrivere “Hello World!” sul file aperto…**

**KERNEL**

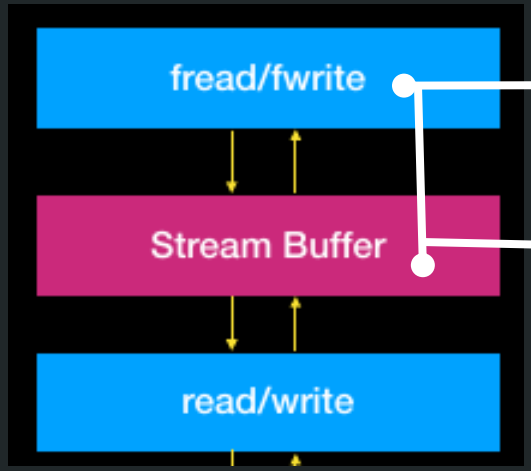



**SCRIVI “Hello World!”**

**Stream Buffer**
“”

# KERNEL

**File Aperto sul kernel**
“”

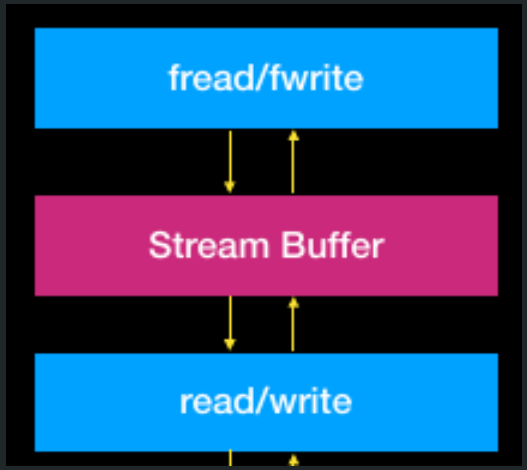



**SCRIVI “Hello World!”**

**Stream Buffer**
**“Hello World!”**

# KERNEL

**File Aperto sul kernel**
**“”**




Stream Buffer
“Hello World!”

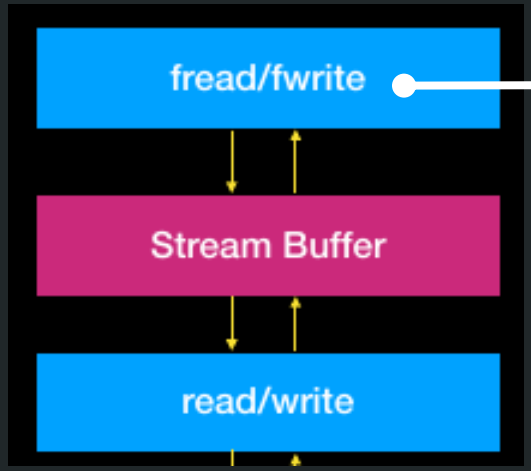
**L’operazione si è conclusa ma il file non è stato scritto!**

## Come mai?

**Non viene richiesta subito la scrittura su file per evitare di fare troppe chiamate al kernel: la chiamata al kernel (tecnicamente chiamata syscall) è un’azione molto dispendiosa in termini di tempo, quindi si cerca di fare il minimo numero di chiamate. Infatti se dobbiamo nuovamente scrivere il file avremo un vantaggio.**

## KERNEL

File Aperto sul kernel
“”

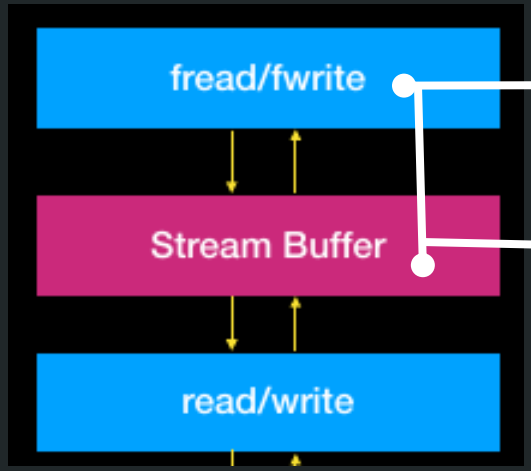



**SCRIVI “\nHow are you?”**

**Stream Buffer**
**“Hello World!”**

# KERNEL

**File Aperto sul kernel**
**“”**

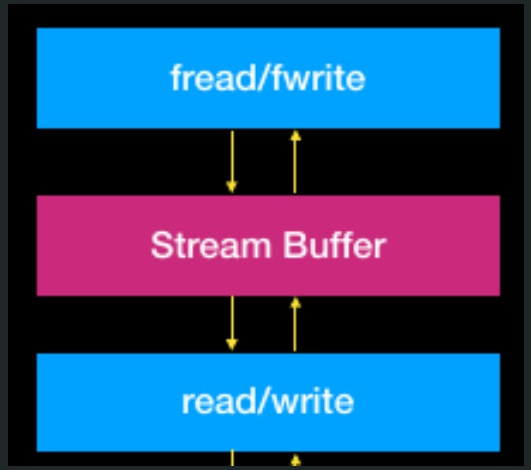



**SCRIVI "\nHow are you?"**

**Stream Buffer**
**"Hello World!\nHow are you?"**

# KERNEL

**File Aperto sul kernel**
**""**




Stream Buffer
“Hello World!\nHow are you?”

**Ecco che qui si rendono evidenti i vantaggi…**
**Al posto di chiamare 2 volte il kernel per scrivere 1 volta**

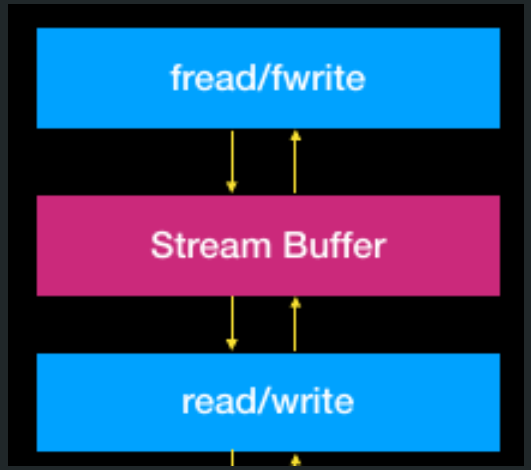
**“Hello World!” e “\nHow are you?”**

**siamo riusciti a compattare le informazioni da scrivere per poi poter fare 1 syscall al posto di 2.**

**KERNEL**

## Ma ora quando viene scritto il file?

File Aperto sul kernel
“”




Stream Buffer
“Hello World!\nHow are you?”

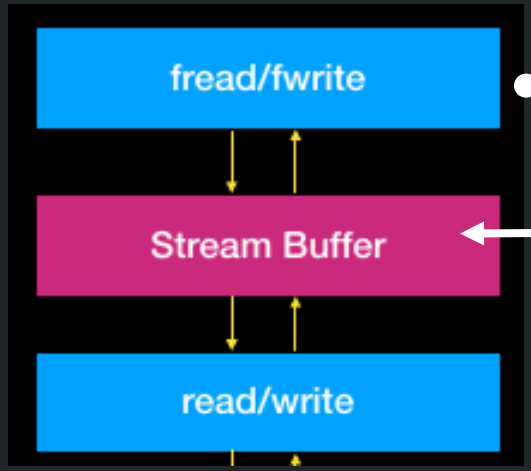
**ABBIAMO 2 CASI:**

**- Lo spazio sul buffer è finito**
**- Viene chiamata la funzione flush**

**In questi 2 casi, il linguaggio di programmazione procede a scrivere ciò che è scritto nel buffer sul file, chiedendo al kernel di farlo**

# KERNEL

File Aperto sul kernel
“”

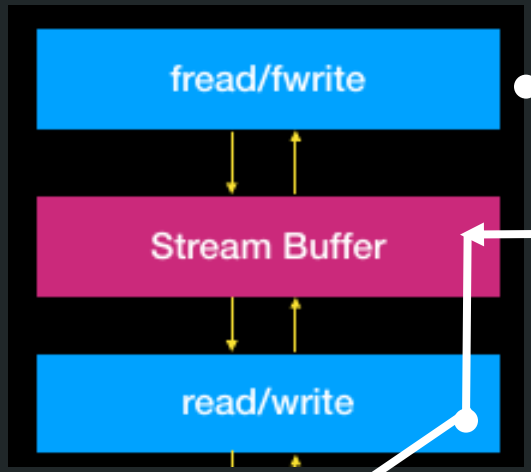



**FLUSH BUFFER!**

**Stream Buffer**
**“Hello World!\nHow are you?”**

# KERNEL

**File Aperto sul kernel**
**“”**



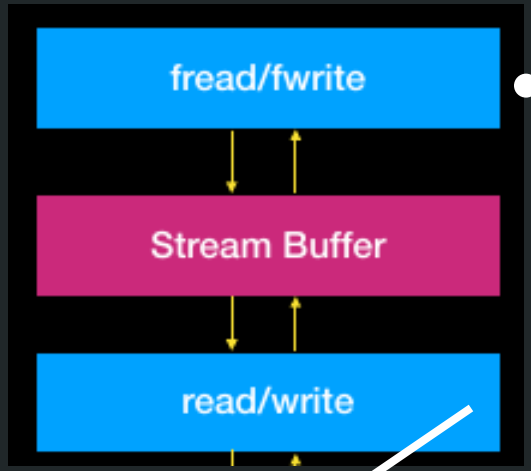

**FLUSH BUFFER!**

**Stream Buffer**
**“Hello World!\nHow are you?”**

**SYSCALL**

# KERNEL

**File Aperto sul kernel**
**“”**



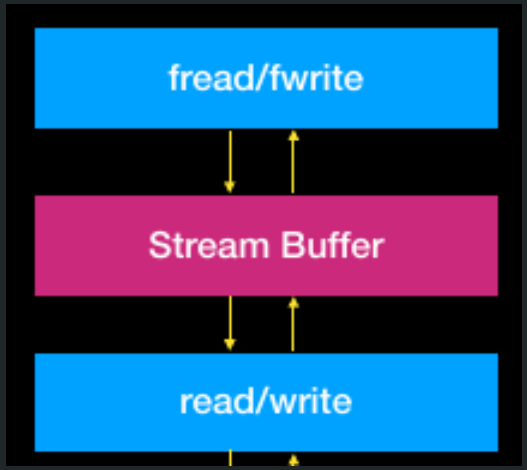

**FLUSH BUFFER!**

**Stream Buffer**
**“”**

**SYSCALL**

# KERNEL

**File Aperto sul kernel**
**“Hello World!\nHow are you?”**




Stream Buffer
“”

**Come risultato finale abbiamo che siamo riusciti a fare 1 sola chiamata al kernel e che abbiamo nuovamente svuotato lo stream Buffer**

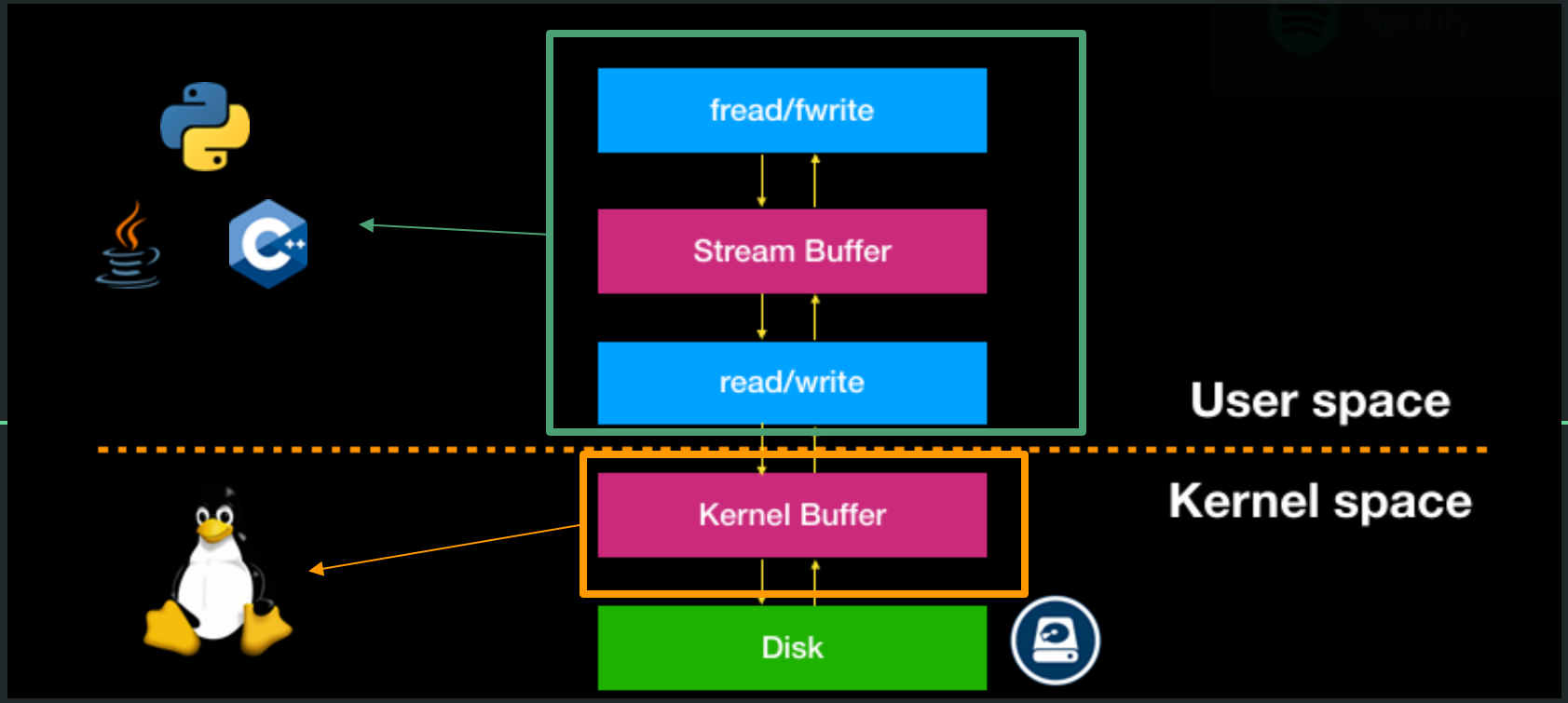
**KERNEL**

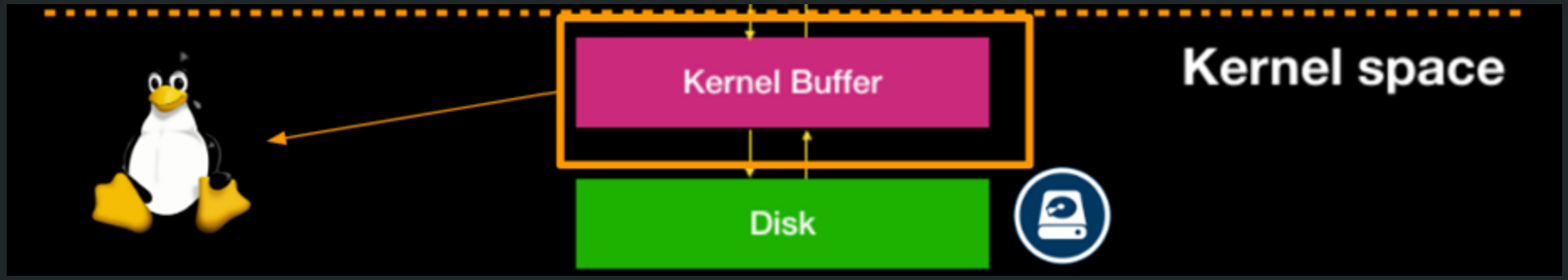
**Ora anche andando a controllare tramite un editor di testo, vediamo che il file è effettivamente visibile,**
**Infatti riusciamo a leggere ciò che abbiamo scritto tramite il programma.**

File Aperto sul kernel
“Hello World!\nHow are you?”

# KERNEL SPACE

**Cosa succede?**

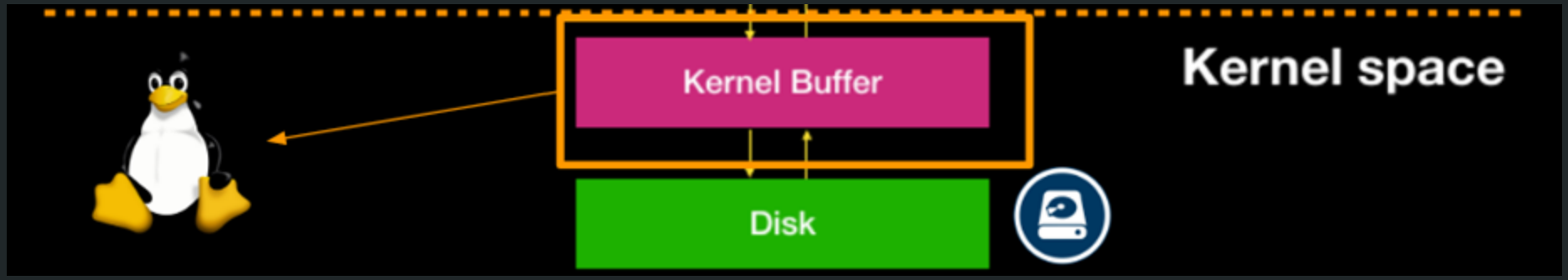

fread/fwrite
Stream Buffer
read/write
User space
Kernel space
Kernel Buffer
Disk




**Ebbene come abbiamo visto dallo schema generale, dopo la chiamata al kernel non siamo in realtà andati realmente a scrivere sul disco! Le nostre informazioni in realtà sono ancora in RAM.**

**Perchè il kernel inserisce un ulteriore buffer prima di scrivere effettivamente in memoria?**

**Come il programma cerca di evitare di fare molte syscall, il kernel si preoccupa di evitare di effettuare molte operazioni di lettura/scrittura che sono operazioni che richiedono un tempo maggiore, e vanno assolutamente ridotte!**

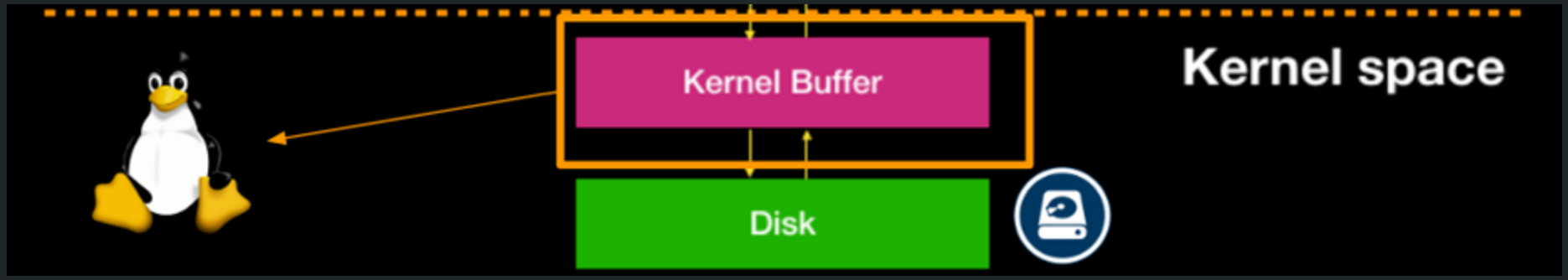



**Il kernel però si preoccupa di mostrare la risorsa del file come se già stata scritta, sia se a chiedere una lettura sia lo stesso programma, sia che la lettura sia effettuata da un'altro programma!**

**In generale la scrittura effettiva sulla memoria viene effettuata in alcuni casi dal kernel**

- **Se il buffer è pieno**
- **Se si richiede una lettura del file che abbiamo scritto**
- **viene effettuata la chiusura del programma che ha richiesto la scrittura.**

**A noi utenti però questo meccanismo è invisibile!**




## Scenario improbabile!

**Possiamo renderci conto dell'effettiva presenza di questo buffer nel caso (improbabile ma possibile) in cui il kernel non abbia scritto ancora i dati sulla memoria di massa, e improvvisamente il computer cessasse di avere corrente.**
**In questo caso avremmo una corruzione di memoria, poiché di fatto i dati erano ancora in RAM e sono andati persi!**

**Questo è uno dei tanti motivi per cui si consiglia sempre di evitare di spegnere brutalmente il computer, ma di farlo tramite il Sistema Operativo che eseguirà diverse operazioni essenziali affinché nessun**

# Dettaglio Aggiuntivo

**Ecco come poi il kernel si occupa della scrittura effettiva su file**


*** Il file system si occupa proprio di gestire la memoria di massa in unità logiche chiamate file. È proprio questo sistema che ci permette di salvare le informazioni in oggetto che in realtà non esiste che noi chiamiamo proprio file. In realtà i file non esistono, e i dati non sono necessariamente contenuti in un blocco unico di memoria. Infatti il filesystem si occupa anche della frammentazione e non solo….**
**Se il file system supporta la cifratura del disco, permette anche di creare un canale di comunicazione che sembra essere non cifrato da parte del kernel e dell'utente, ma che in realtà sulla memoria fisica è effettivamente cifrato. Il filesystem ci nasconde la complessità delle operazione di cifratura e decifratura e permette la gestione di un disco cifrato, pari a**