Προγραμματισμός ΙΙ (ECE116)

#14 περισσότερα για τους περιγραφείς αρχείων και ανακατεύθυνση Ε/Ε

Σπύρος Λάλης
lalis@uth.gr

Εσωτερική δομή πρόσβασης αρχείου

- Όταν ανοίγει ένα αρχείο, το λειτουργικό δημιουργεί (εσωτερικά) μια ξεχωριστή **δομή πρόσβασης**
  - μεταξύ άλλων, περιέχει πληροφορία για την αντίστοιχη θέση ανάγνωσης/εγγραφής πάνω στο αρχείο

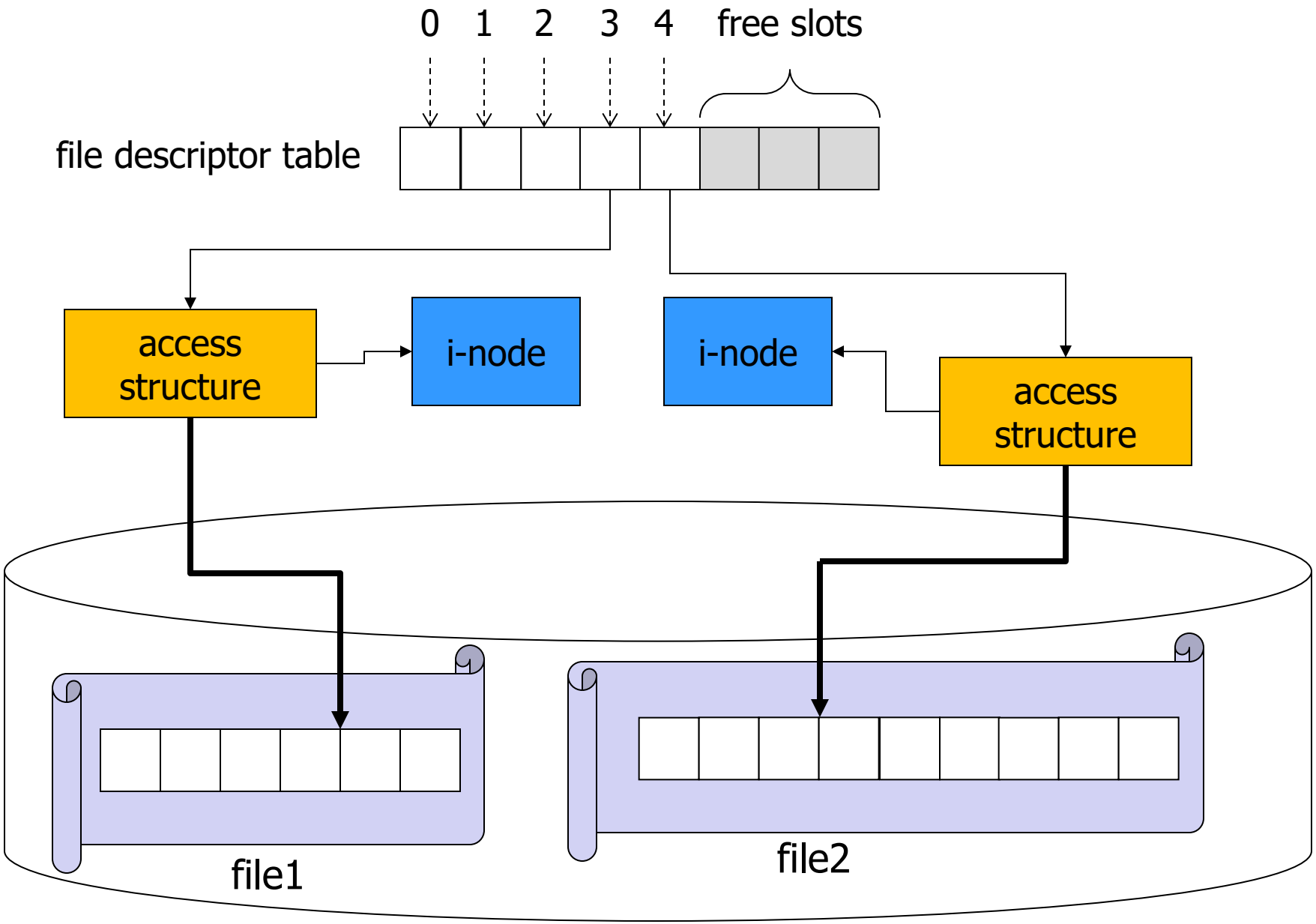
- Το λειτουργικό διατηρεί (εσωτερικά) έναν **πίνακα από δείκτες** στις δομές πρόσβασης που έχουν δημιουργηθεί στο πλαίσιο κάθε προγράμματος
  - αυτές οι δομές πρόσβασης **δεν** είναι άμεσα προσπελάσιμες από τον κώδικα της εφαρμογής – γιατί;

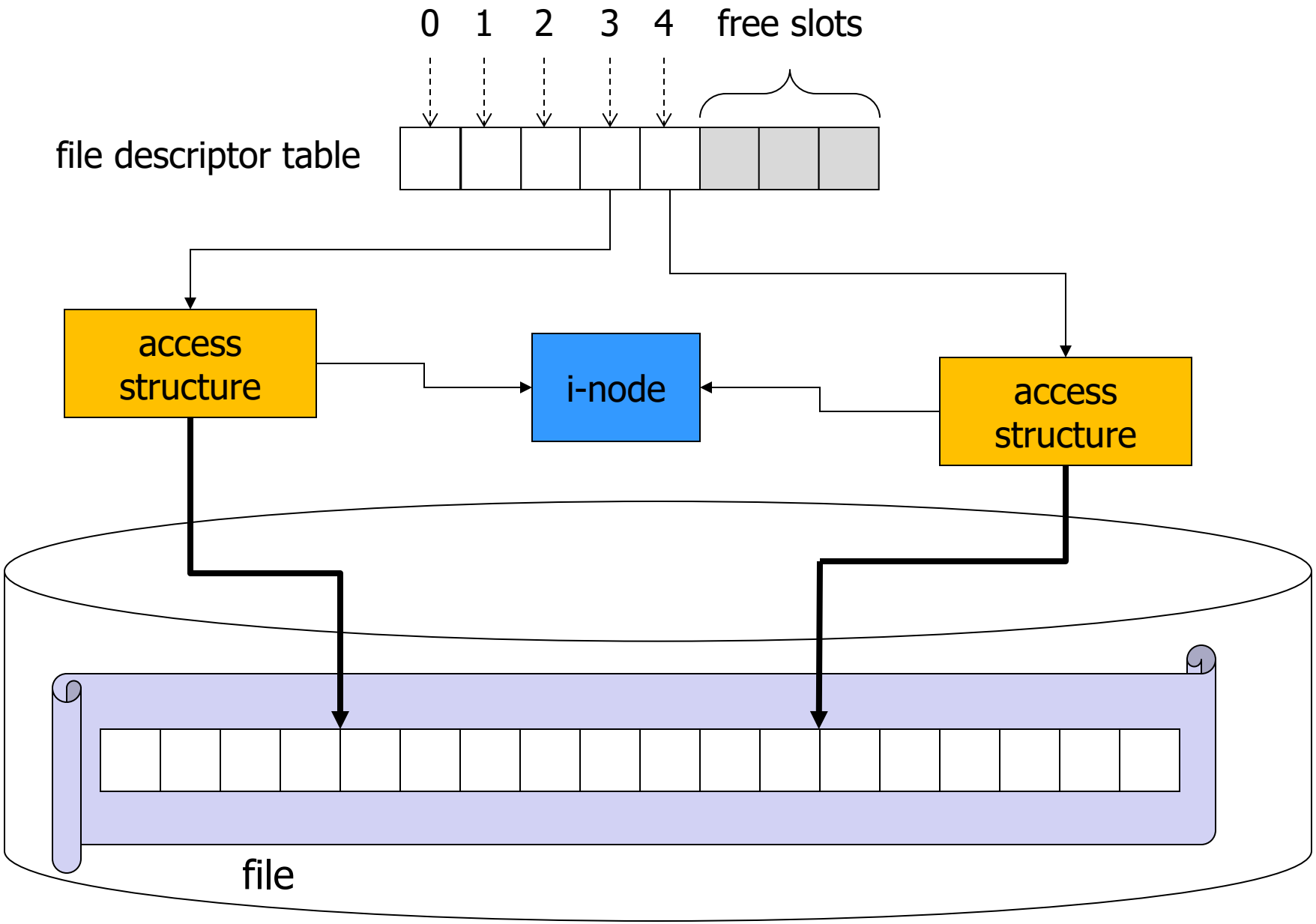
- Κάθε δομή πρόσβασης δείχνει στο inode του αρχείου

Τι είναι τελικά ο περιγραφέας αρχείου;

- Η **θέση στον πίνακα** όπου βρίσκεται ο δείκτης στην νέα δομή πρόσβασης που δημιουργήθηκε μέσω `open`
- Οι λειτουργίες `read`, `write`, `close` παίρνουν αυτόν τον ακέραιο ως παράμετρο, έτσι ώστε το λειτουργικό να χρησιμοποιήσει την (σωστή) δομή πρόσβασης που βρίσκεται στη συγκεκριμένη θέση του πίνακα

- Θεωρητικά, σε αυτές τις λειτουργίες μπορεί να δοθεί ως παράμετρος μια **οποιαδήποτε** τιμή ακεραίου
  - με ευθύνη του προγραμματιστή
- Θα χρησιμοποιηθεί η (όποια) δομή πρόσβασης «τυχαίνει» να βρίσκεται σε αυτή τη θέση του πίνακα


0
1
2
3
4
free slots
file descriptor table
access structure
i-node
i-node
access structure
file1
file2


0
1
2
3
4
free slots
file descriptor table
access structure
i-node
access structure
file

Γενικότητα περιγραφέων αρχείων

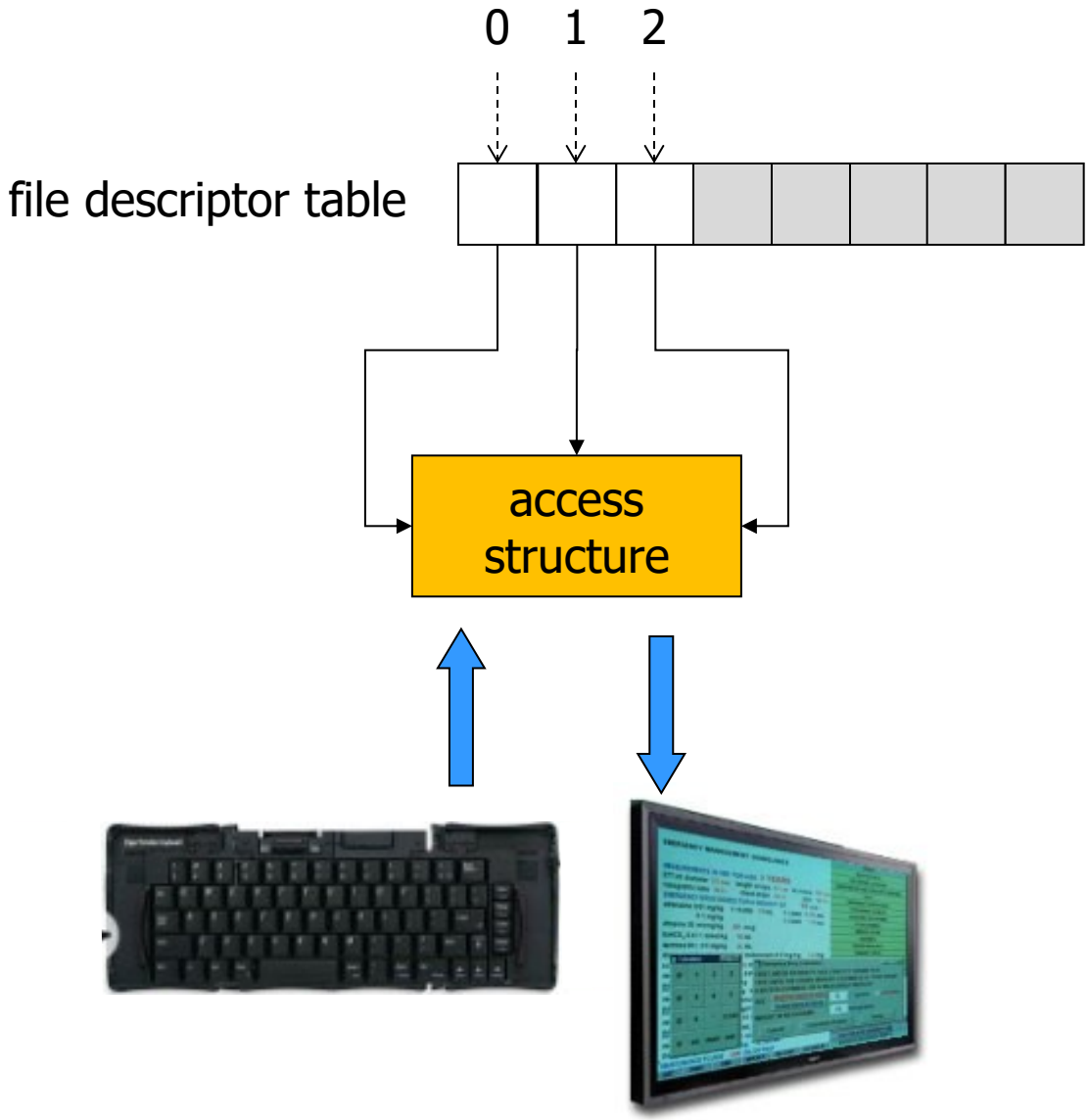
- Οι περιγραφείς αρχείων και οι βασικές λειτουργίες για διάβασμα και γράψιμο δεδομένων `read`, `write` είναι ένας **«γενικός» μηχανισμός πρόσβασης**

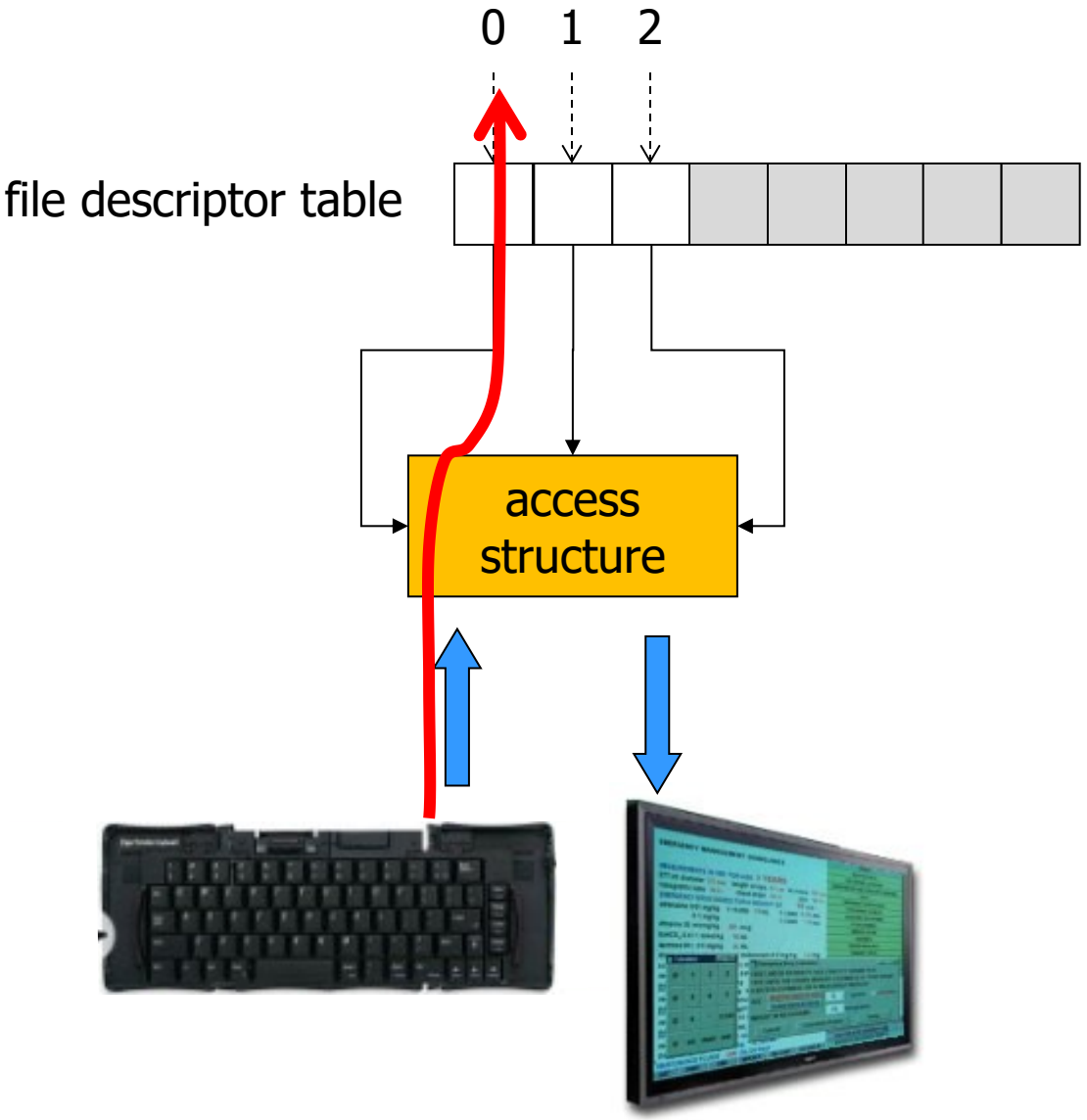
- Παρέχουν **ομοιόμορφη** προσπέλαση σε διαφορετικές «πηγές» δεδομένων
  - αρχεία δίσκου (files), συσκευή τερματικού (terminal), αγωγοί (pipes), υποδοχείς (sockets)

- Μπορεί να γραφτεί κώδικας που χρησιμοποιεί περιγραφείς αρχείων με **αφαιρετικό** τρόπο
  - χωρίς να ενδιαφέρει το «είδος αρχείου» που θα χρησιμοποιηθεί στην πραγματικότητα για διάβασμα/γράψιμο

Καθιερωμένοι περιγραφείς αρχείων

- Υπάρχουν **κατά σύμβαση** τρείς τιμές περιγραφεών αρχείων με καθιερωμένη χρήση
- **Καθιερωμένη είσοδος:** `STDIN_FILENO` (`0`)
- **Καθιερωμένη έξοδος:** `STDOUT_FILENO` (`1`)
- **Καθιερωμένη έξοδος λαθών:** `STDERR_FILENO` (`2`)
- Αντιστοιχούν (συνήθως) στην **συσκευή τερματικού** (tty) που χρησιμοποιεί ο χρήστης του προγράμματος
  - `0` -> τερματικό/πληκτρολόγιο, `1,2` -> τερματικό/οθόνη
  - αυτά **δεν** είναι πραγματικά αρχεία
- Οι αντίστοιχες δομές πρόσβασης δημιουργούνται **αυτόματα** από το λειτουργικό / περιβάλλον εκτέλεσης
  - χωρίς να κάνει κάτι η εφαρμογή/προγραμματιστής

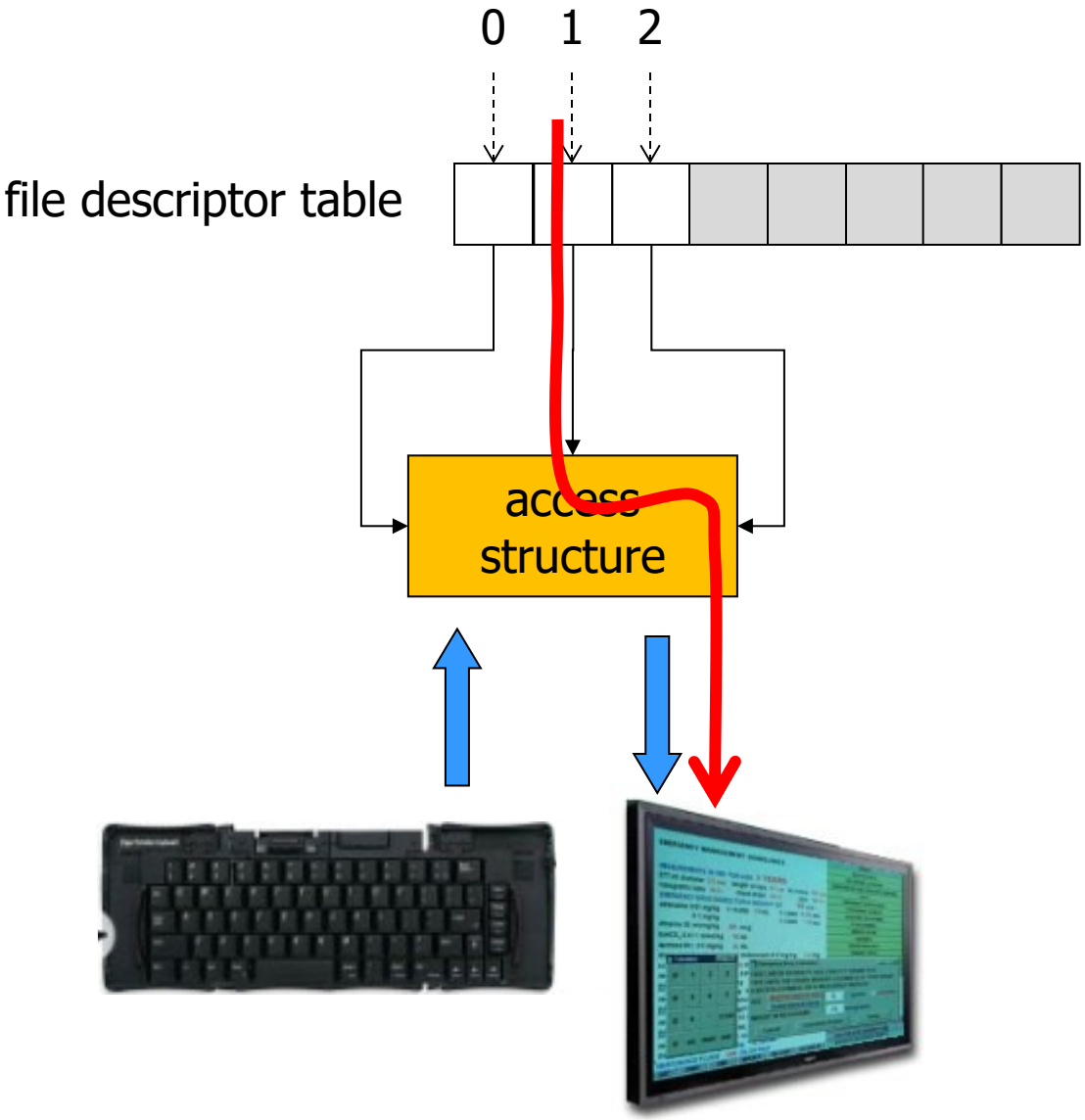

0
1
2
file descriptor table
access
structure




```
scanf(...);
```

↓

```
read(0,...);
```




```
printf(...);
```

↓

```
write(1,...);
```

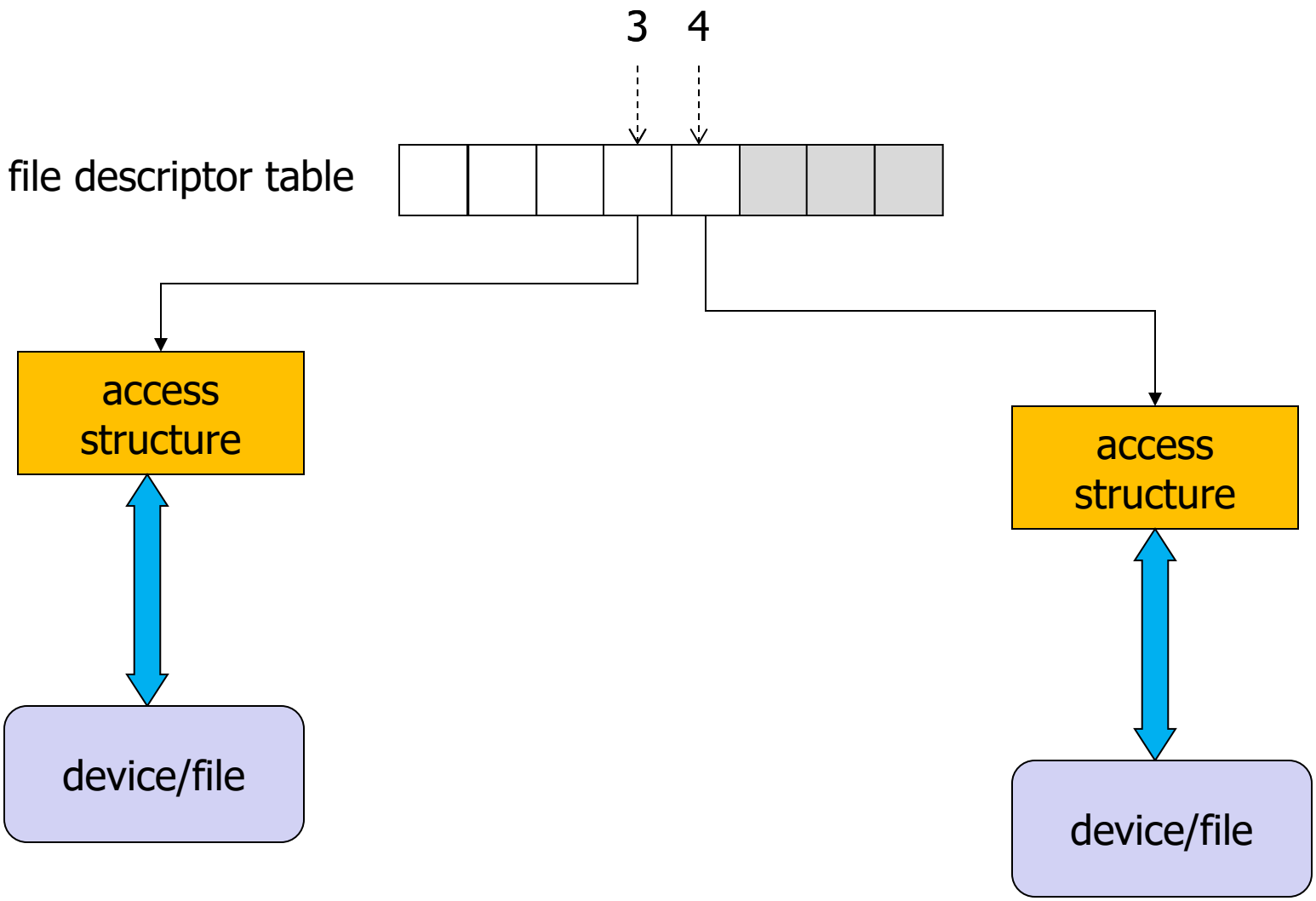
```
int n;
char str[STRLEN];

printf("Enter a few words\n");

do {
  n = read(STDIN_FILENO, str, STRLEN);
  // echo whatever was read
  write(STDOUT_FILENO, str, n);
} while (n > 0);

printf("End of input\n");
```

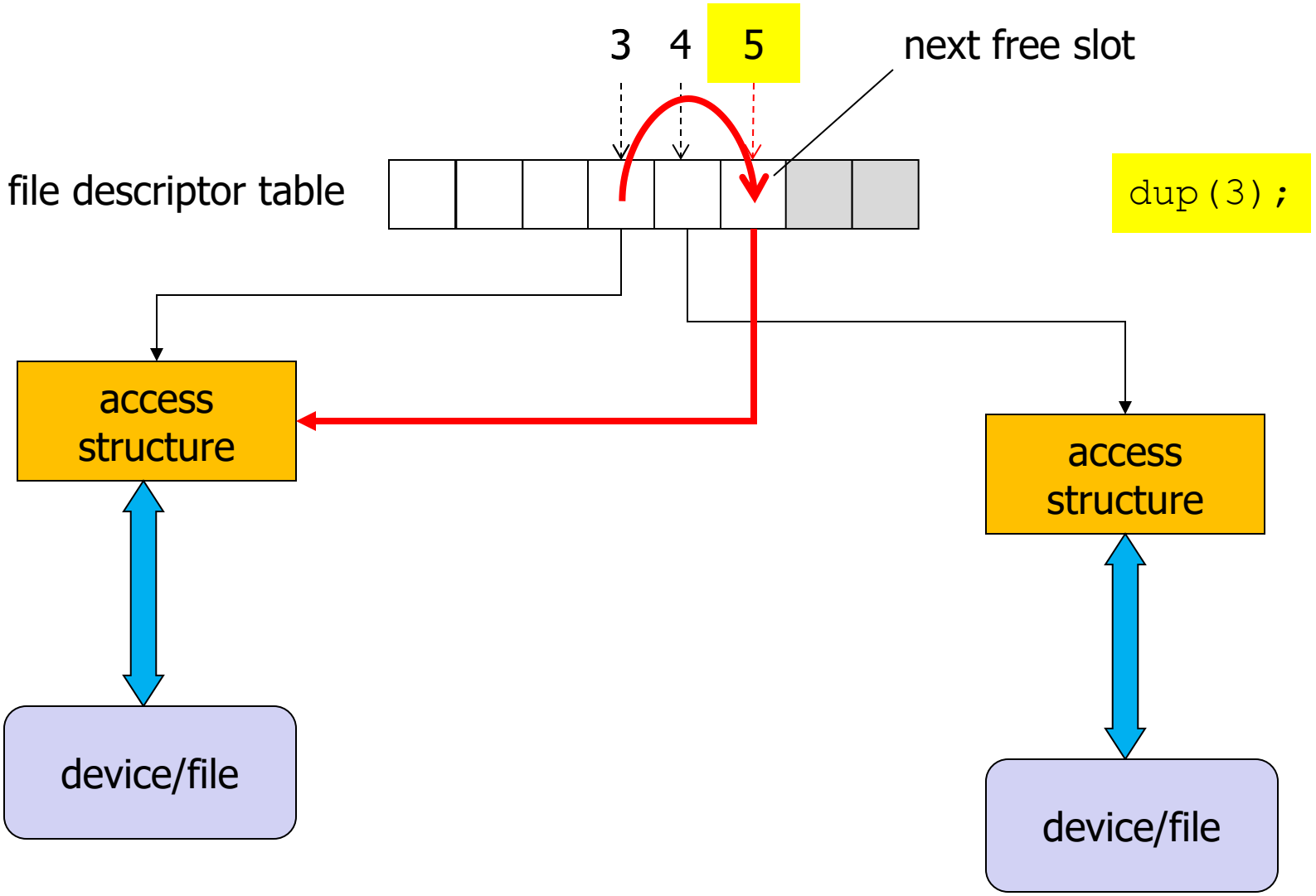
Αντίγραφα περιγραφέων

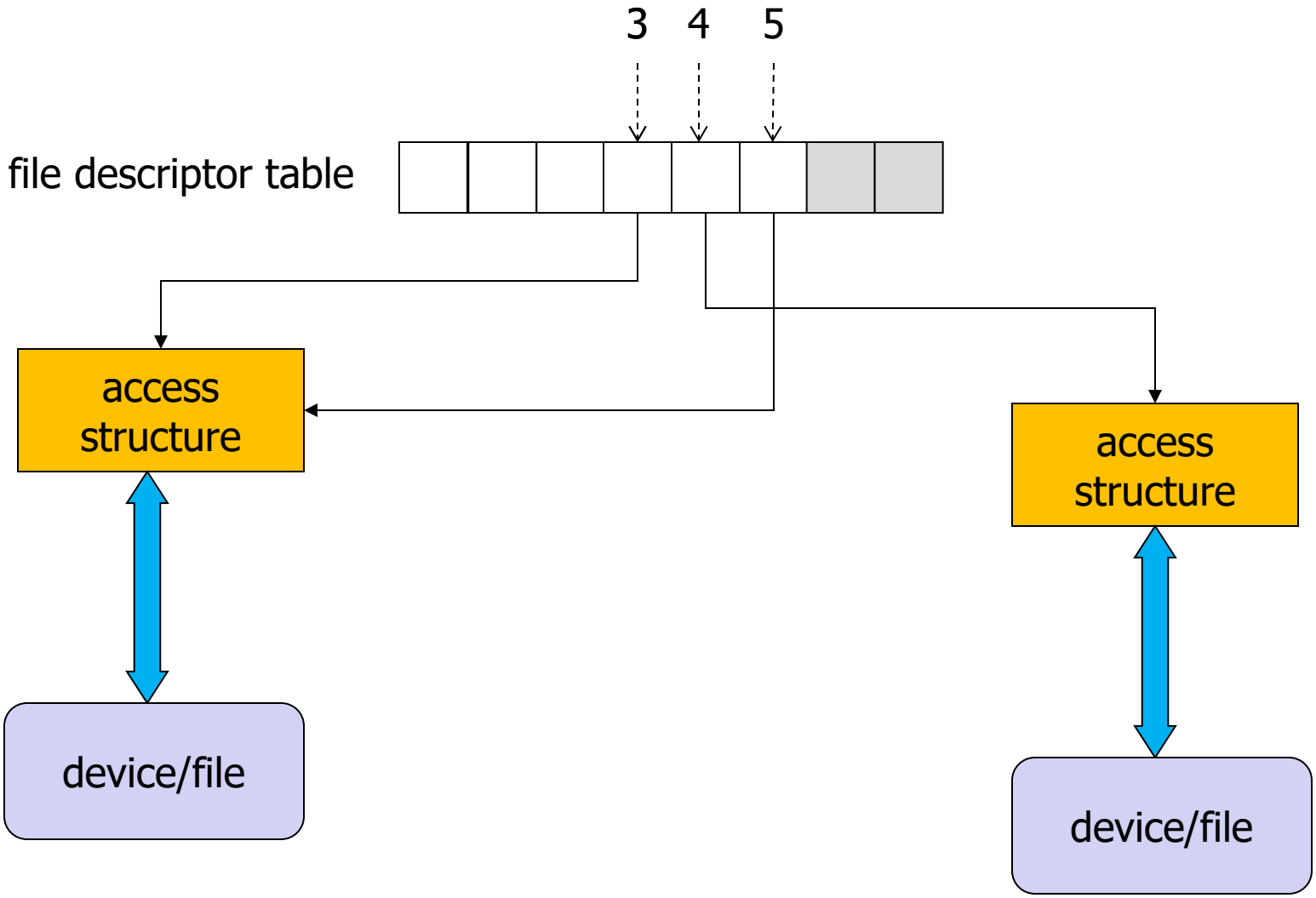
- `int dup(int fd)`: δημιουργεί ένα **αντίγραφο** του περιγραφέα `fd`, που δείχνει στην ίδια δομή πρόσβασης, και επιστρέφει την θέση του νέου περιγραφέα στον πίνακα των δομών πρόσβασης

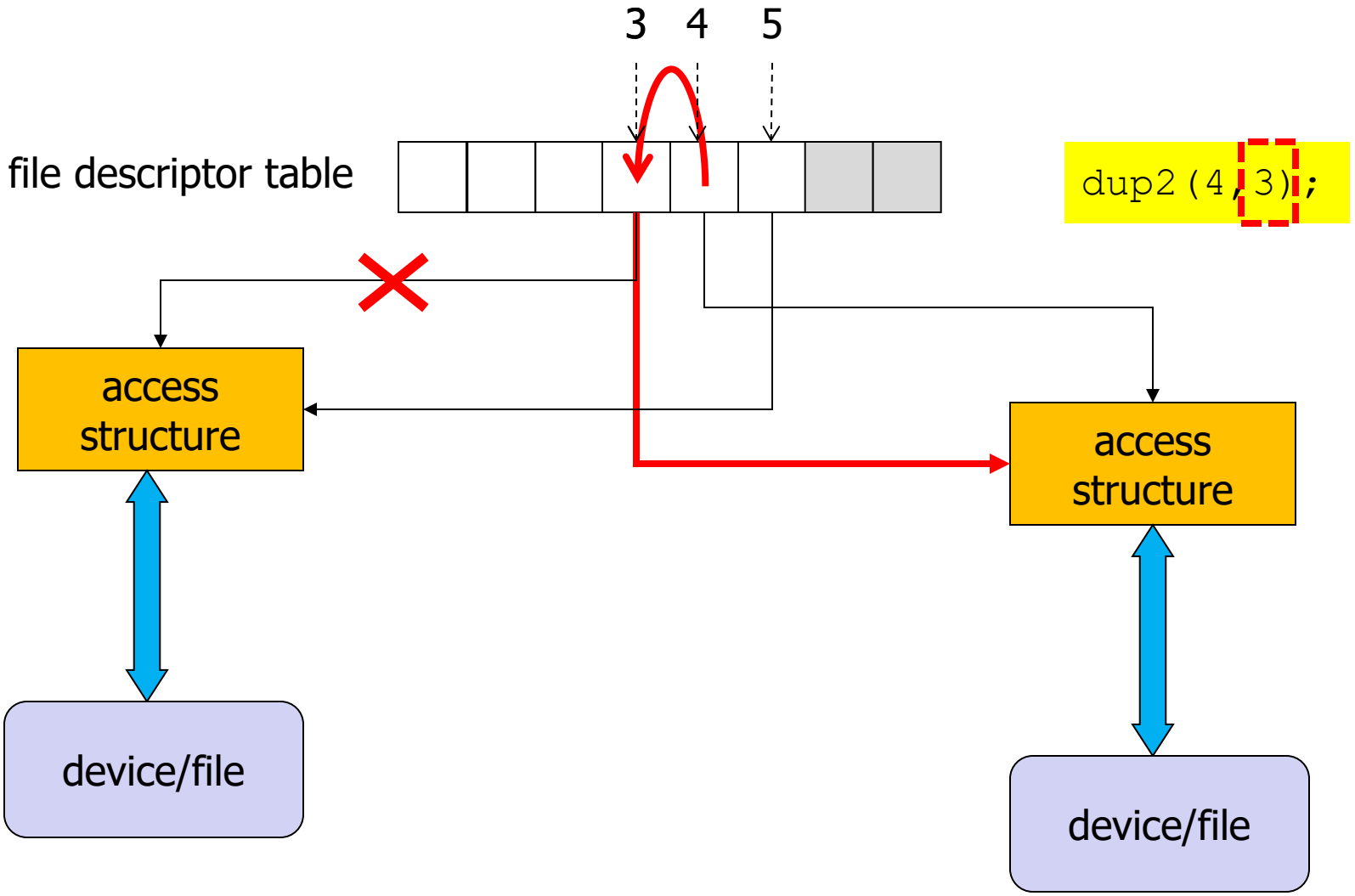
- `int dup2(int fd, int id)`: δημιουργεί ένα **αντίγραφο** του `fd`, στην θέση `id` του πίνακα
  - αν στην θέση `id` ήδη υπάρχει δείκτης σε κάποια άλλη δομή πρόσβασης, γίνεται **αντικατάσταση**

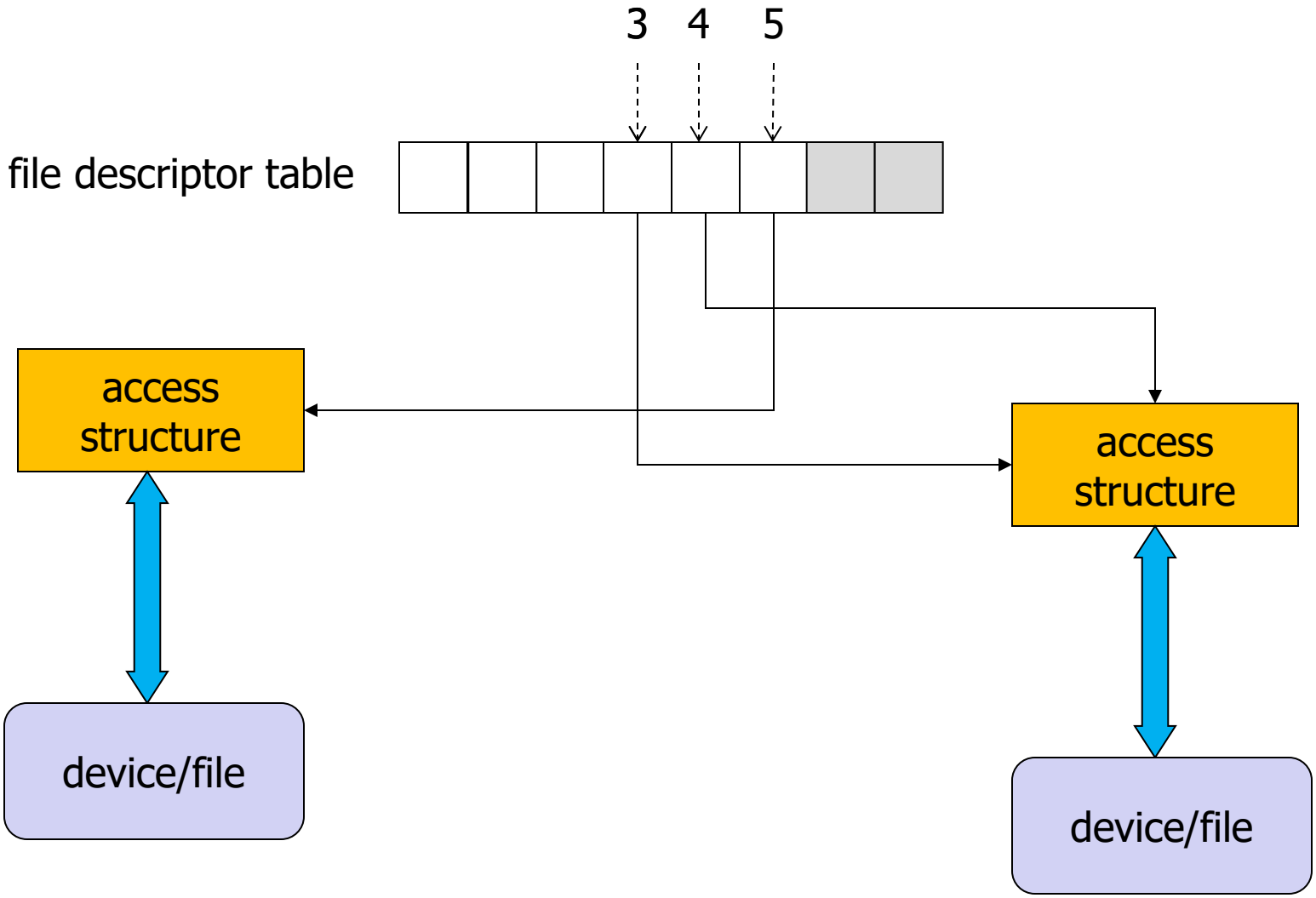
- Τα **αντίγραφα** που δημιουργούνται μέσω `dup/dup2` δείχνουν στην **ίδια** δομή πρόσβασης
  - έχουν την **ίδια** θέση ανάγνωσης/εγγραφής


3
4
file descriptor table
access structure
access structure
device/file
device/file


3
4
5
next free slot
file descriptor table
dup(3);
access structure
access structure
device/file
device/file

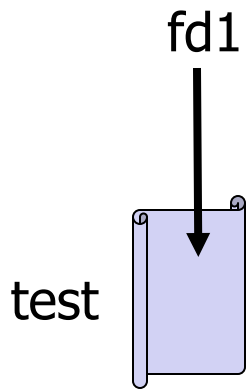

3
4
5
file descriptor table
access structure
access structure
device/file
device/file


3
4
5
file descriptor table
dup2(4,3);
access structure
access structure
device/file
device/file

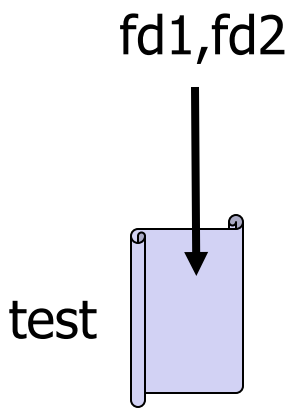

3
4
5
file descriptor table
access structure
access structure
device/file
device/file

```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```

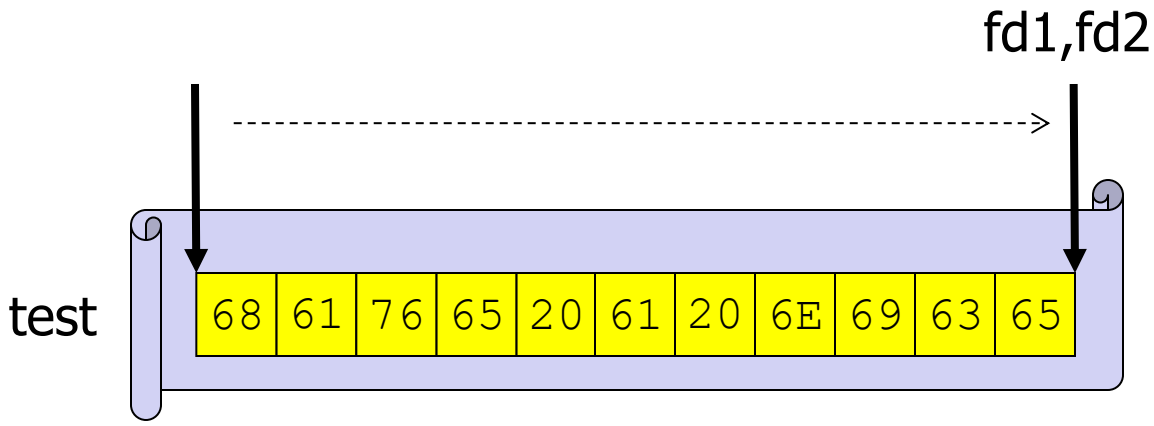
```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



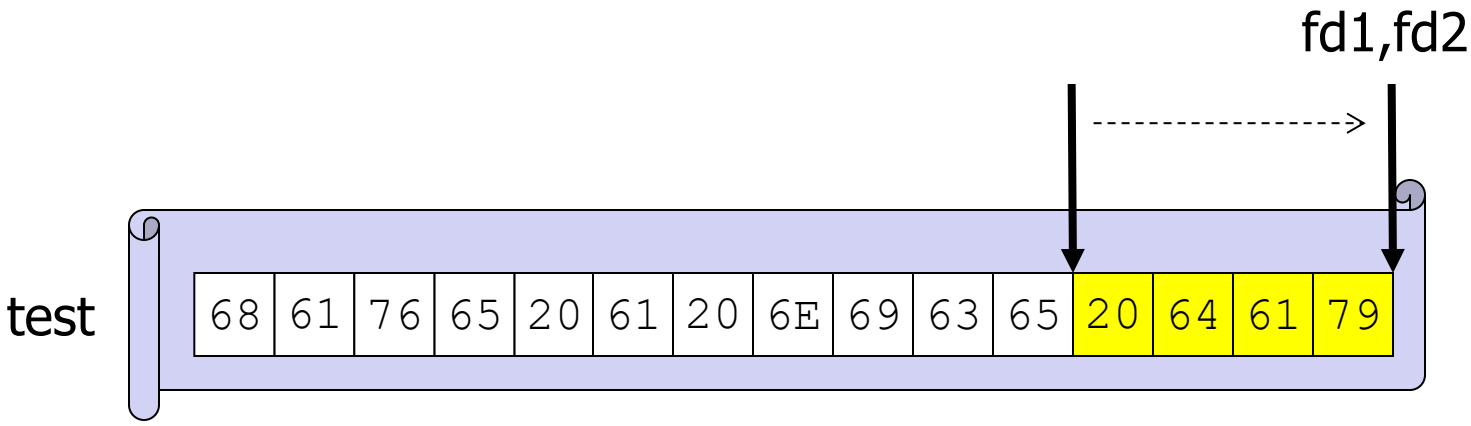
```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



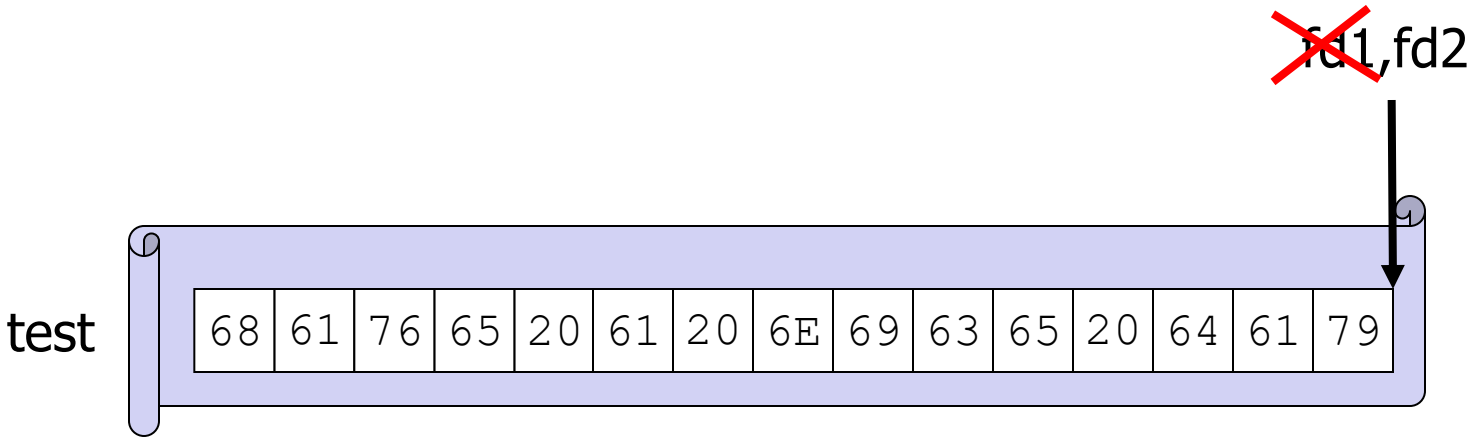
```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



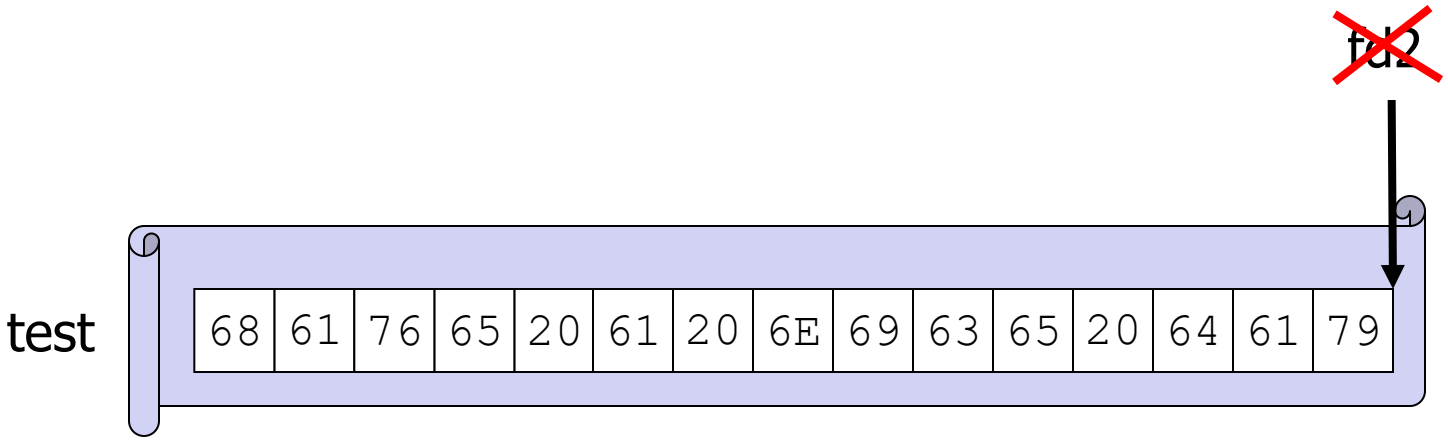
```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```



```
int fd1,fd2;

fd1 = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
fd2 = dup(fd1);
write(fd1,"have a nice",11);
write(fd2," day",4);
close(fd1);
close(fd2);
```

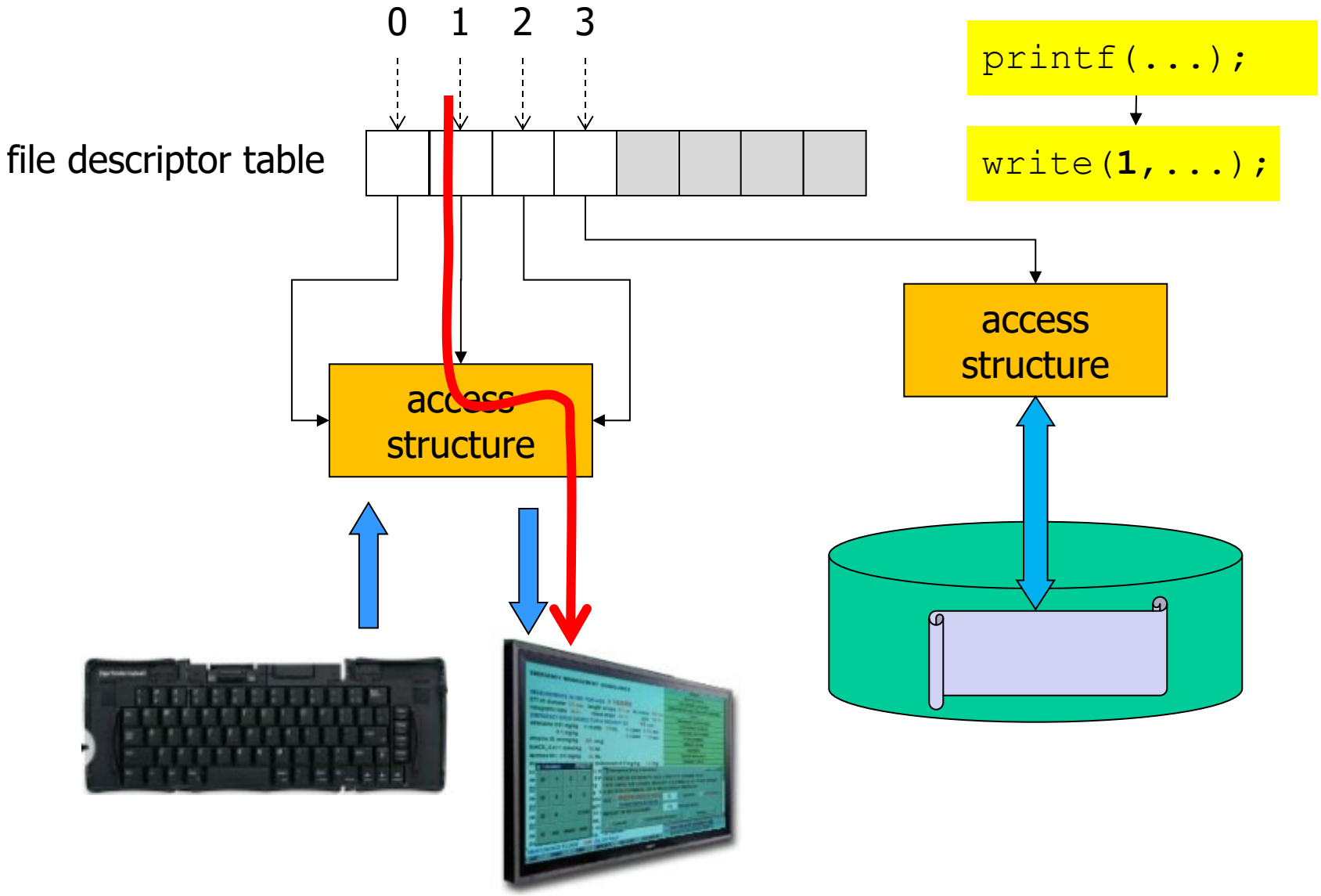


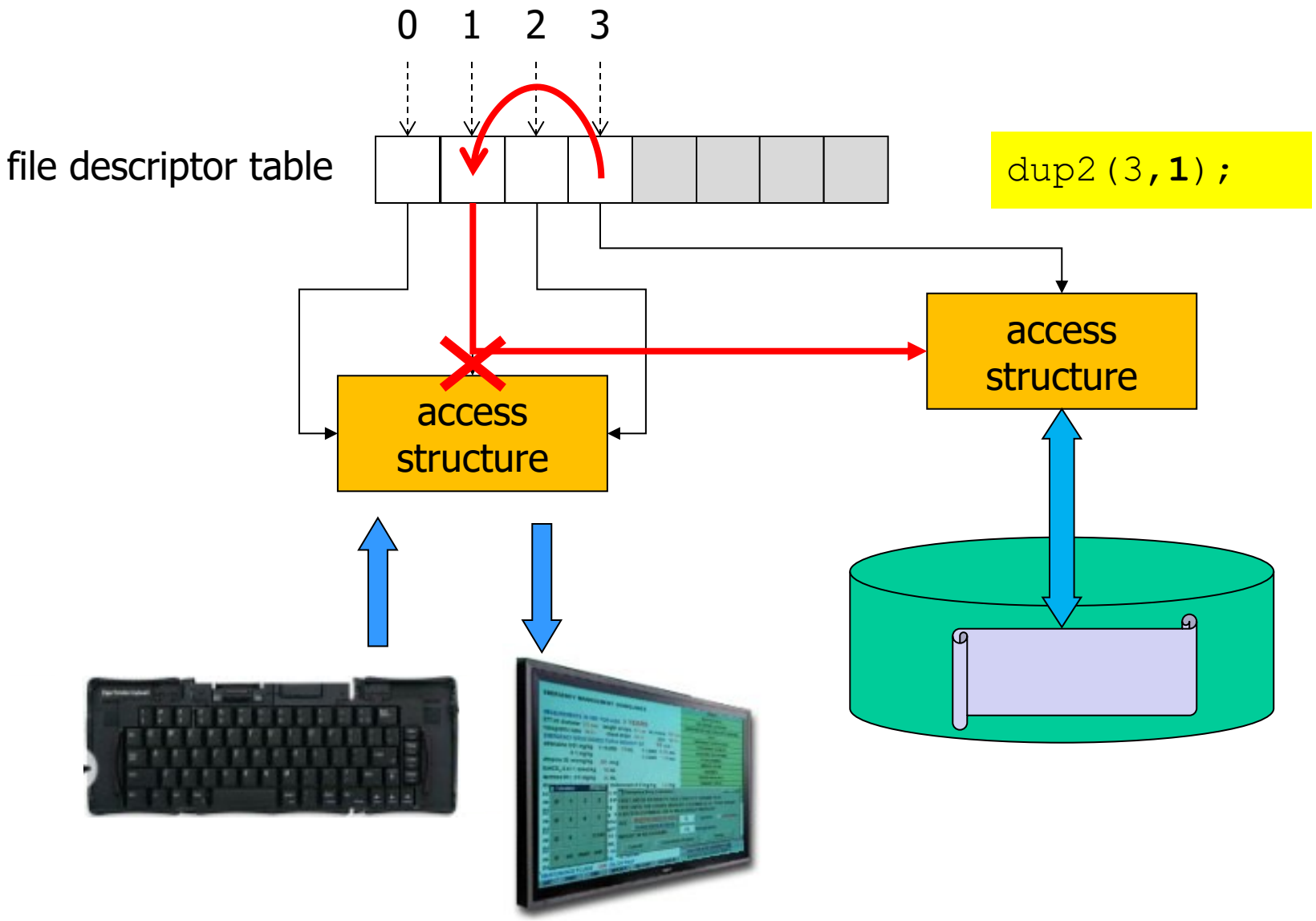
Ανακατεύθυνση εισόδου/εξόδου

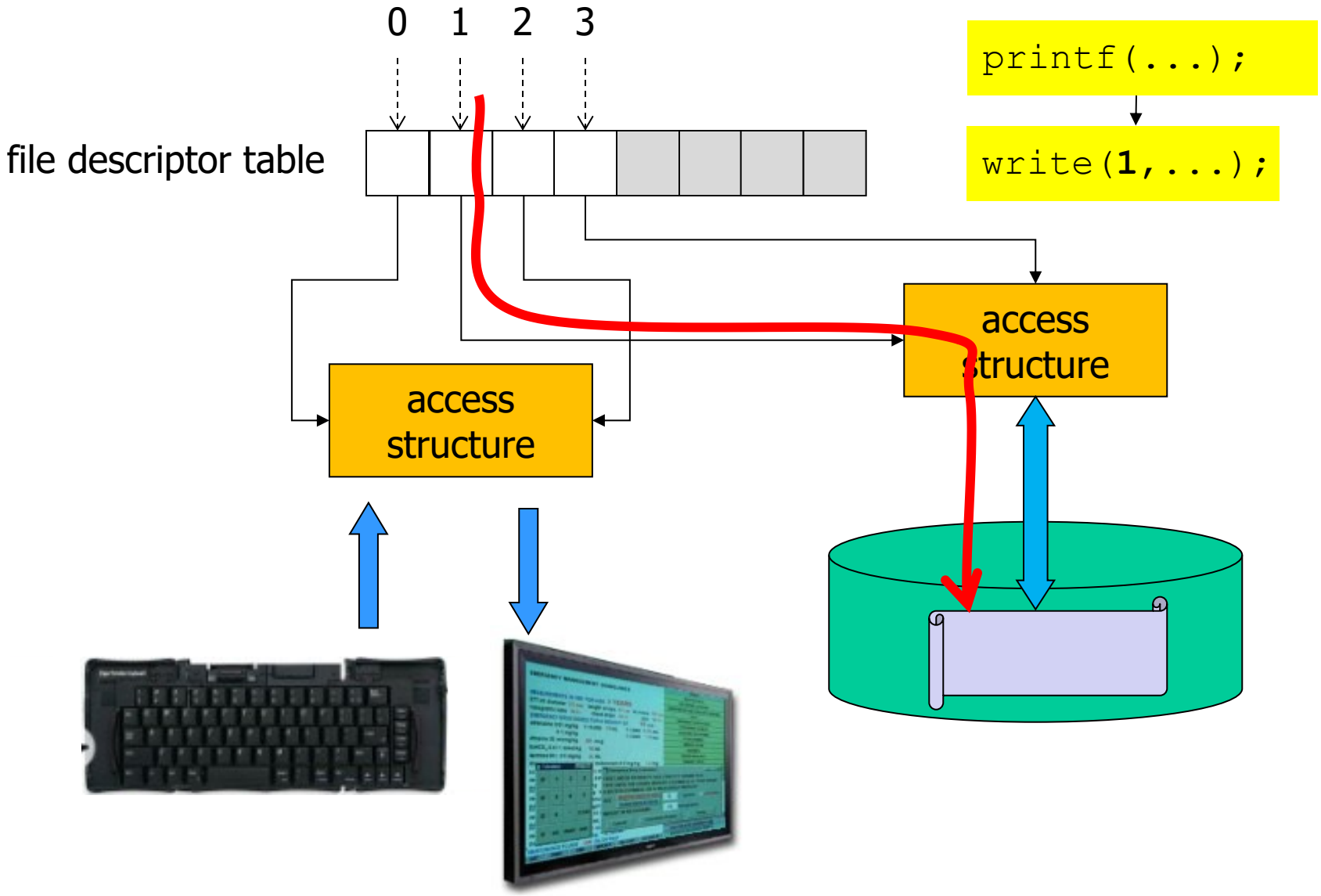
- Οι περιγραφείς `0`, `1`, `2` για την καθιερωμένη είσοδο, έξοδο και έξοδο λαθών, είναι **αρχικοποιημένοι** να δείχνουν στο τερματικό
- Όμως **δεν** είναι μόνιμα «καρφωμένοι» στο τερματικό

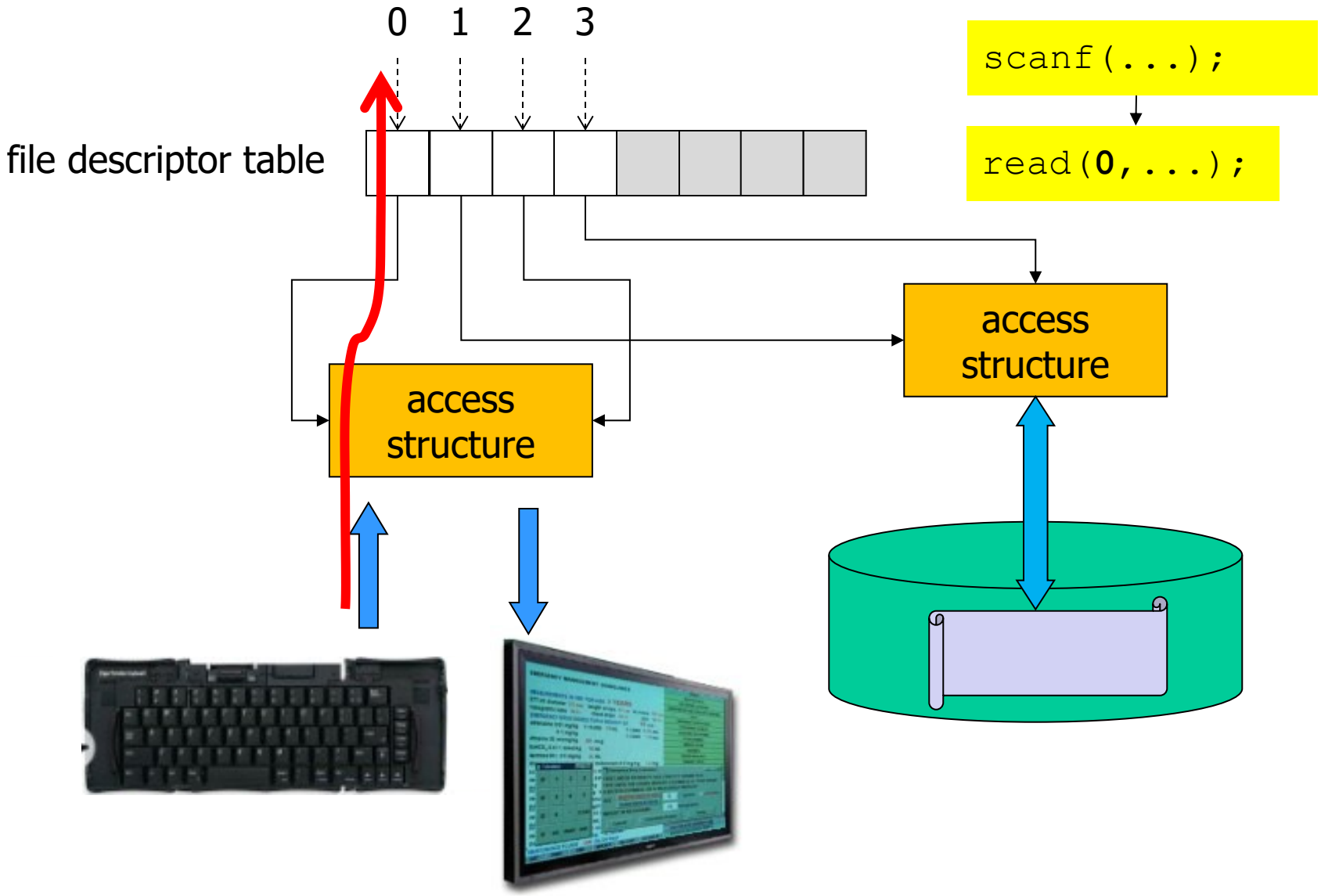
- Μπορεί να γίνει **αντικατάσταση** των δομών πρόσβασης που βρίσκονται σε αυτές τις θέσεις
  - ακόμα και κατά την διάρκεια της εκτέλεσης
- Χρησιμοποιώντας το `dup2`

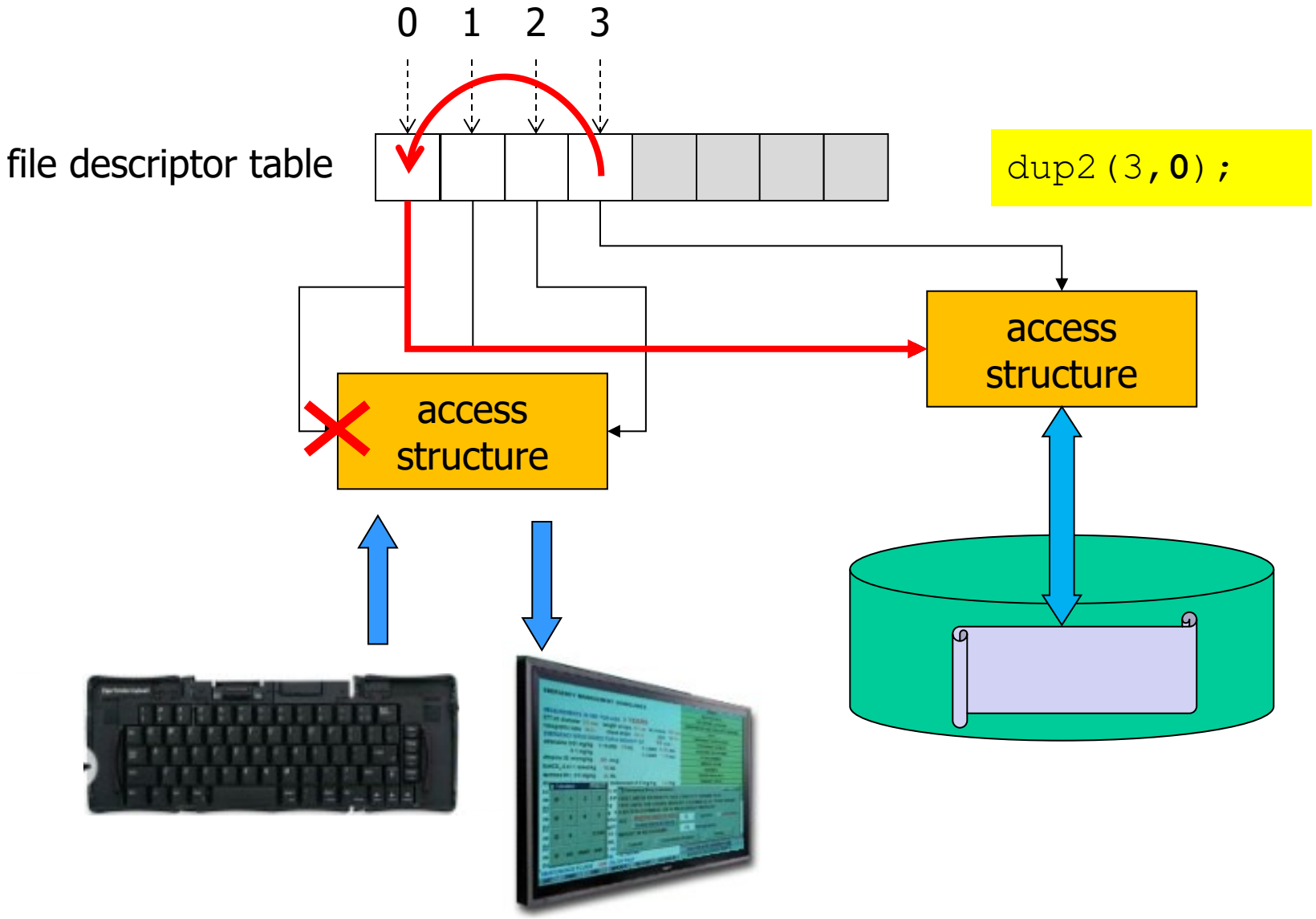
- Αυτό γίνεται **εν αγνοία** του όποιου κώδικα ήδη χρησιμοποιεί τους περιγραφείς `0`, `1`, `2`
  - π.χ., λειτουργίες `scanf`/`printf`

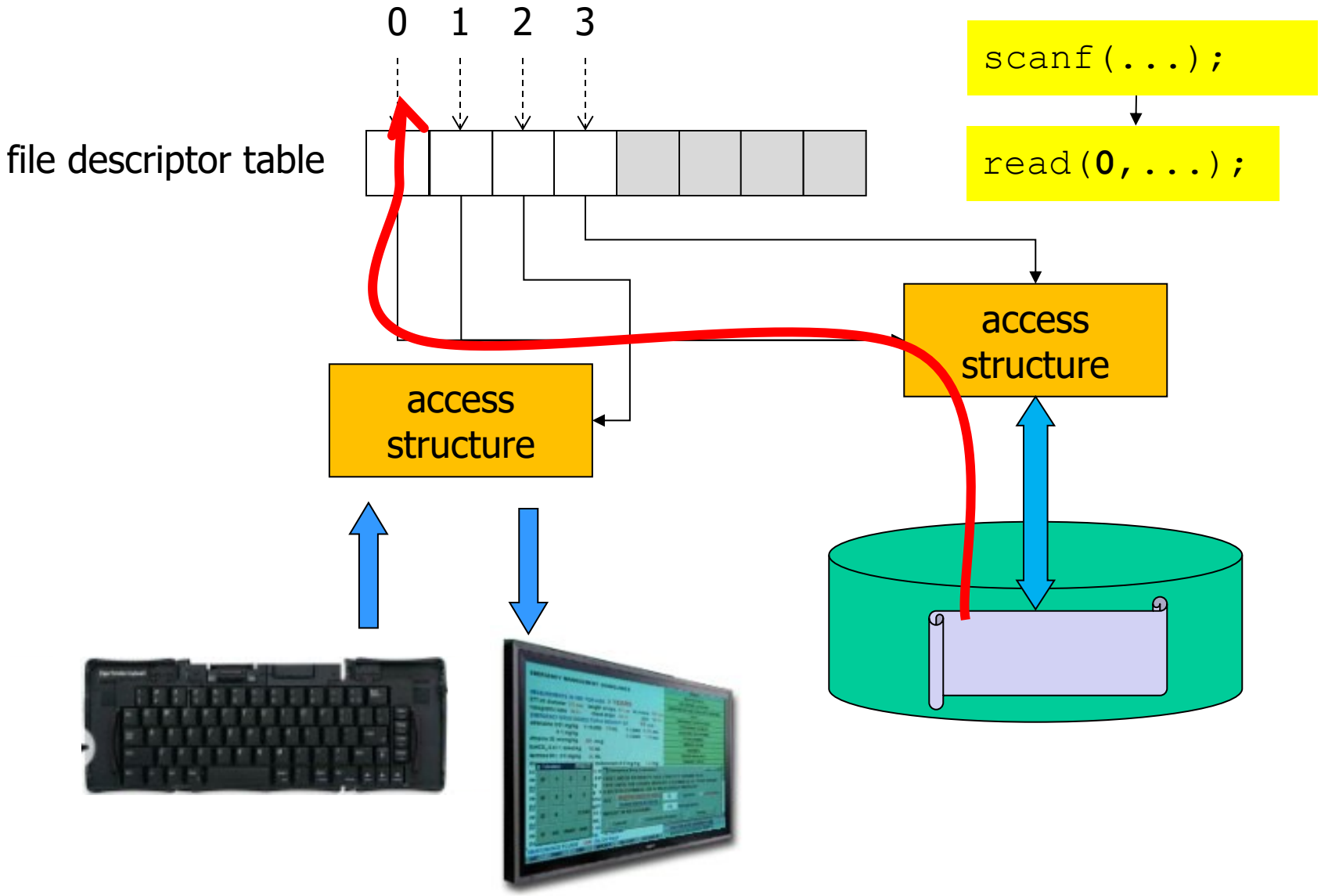

0
1
2
3
printf(...);
write(1,...);
file descriptor table
access structure
access structure


0
1
2
3
file descriptor table
dup2(3,1);
access structure
access structure


0
1
2
3
printf(...);
write(1,...);
file descriptor table
access structure
access structure


0 1 2 3
file descriptor table
scanf(...);
read(0,...);
access structure
access structure


0 1 2 3
file descriptor table
dup2(3,0);
access structure
access structure


0 1 2 3
file descriptor table
scanf(...);
read(0,...);
access structure
access structure

```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```

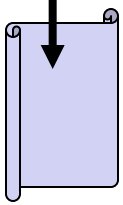
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```

0

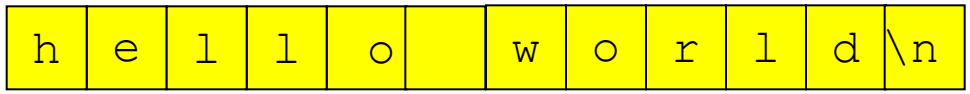
1

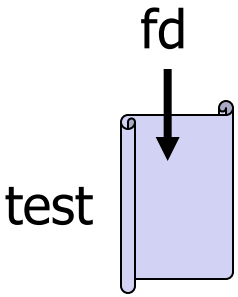
fd

test

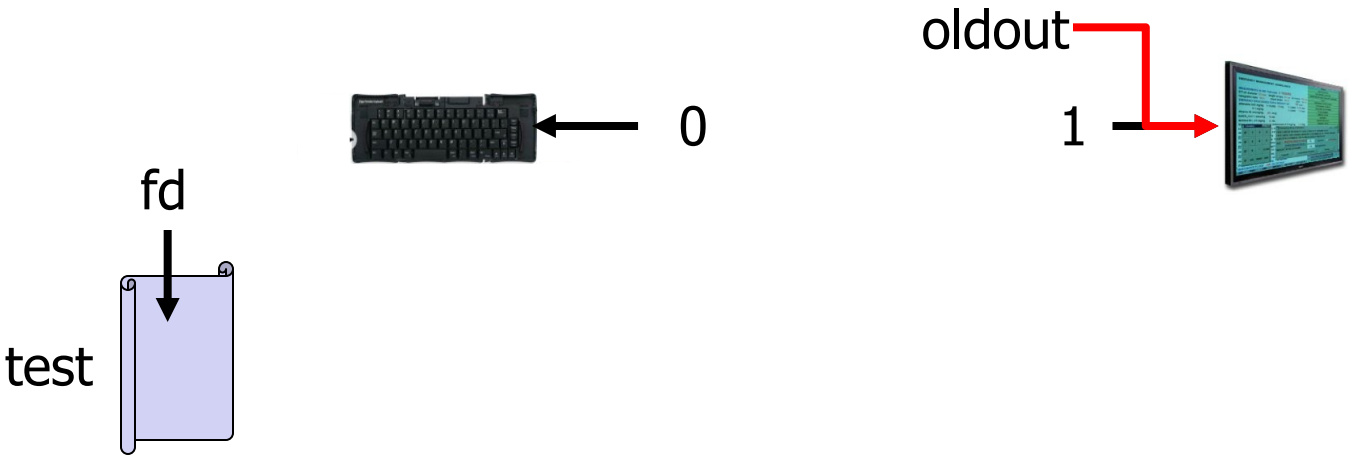


```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```

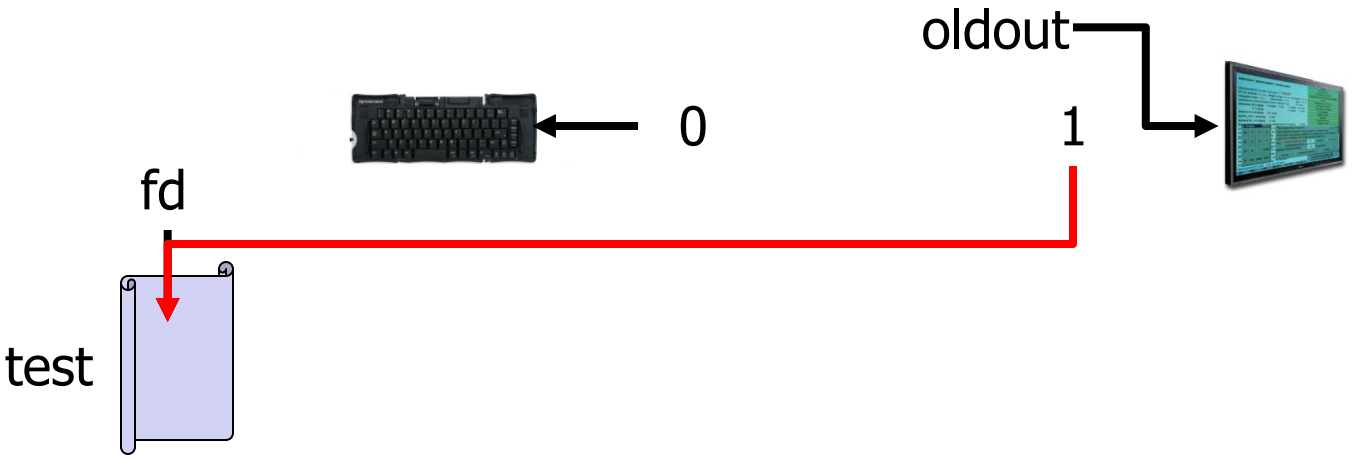




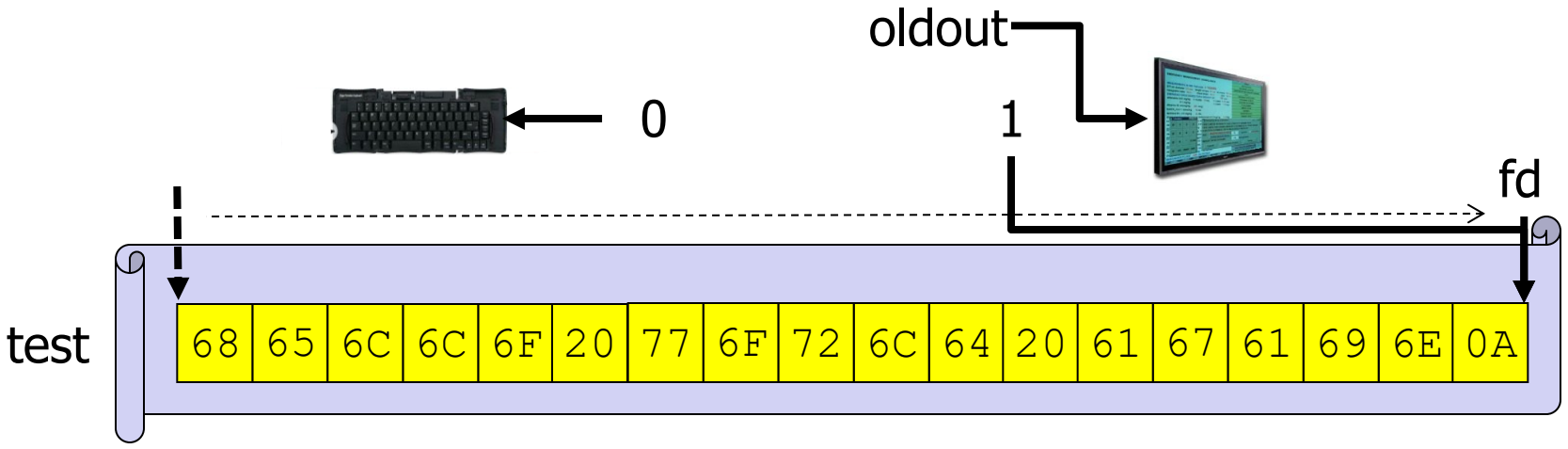
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```



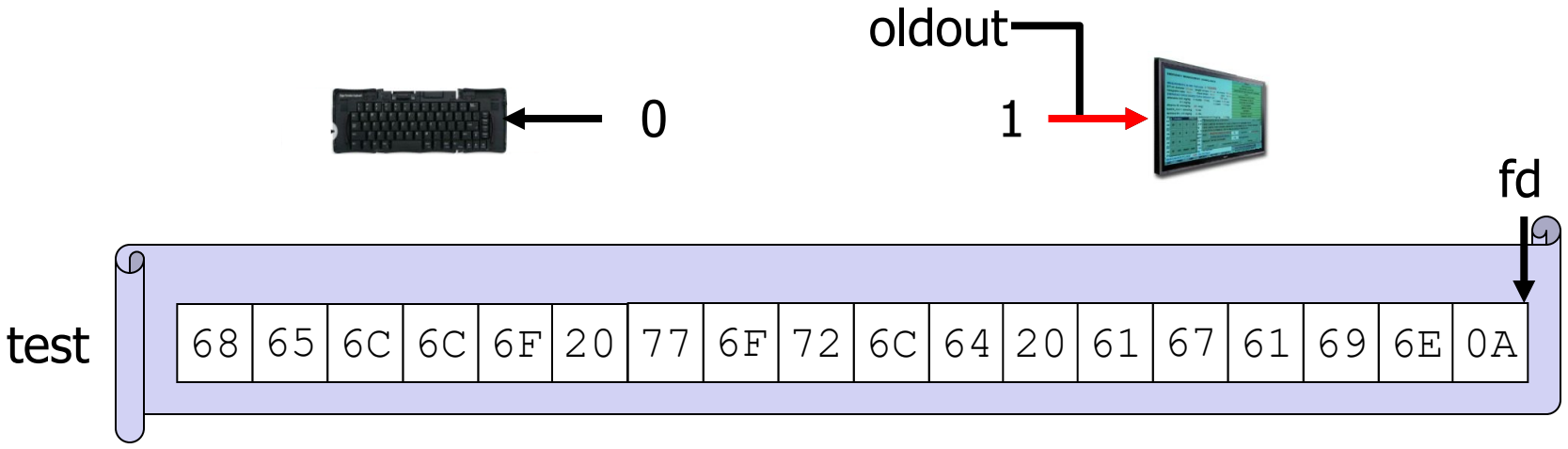
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```



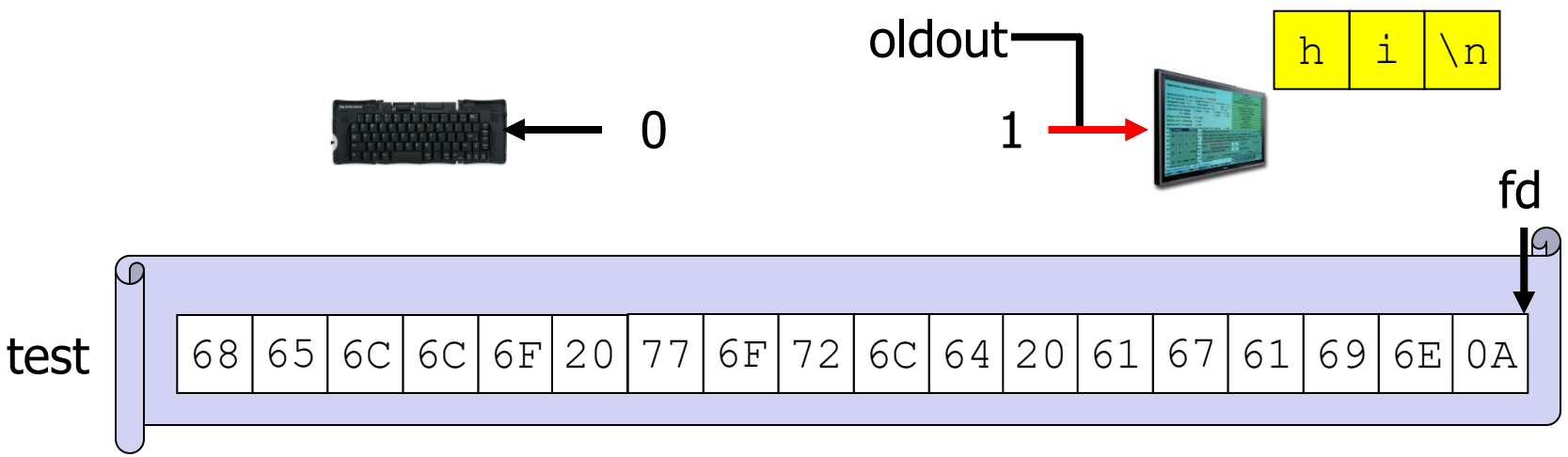
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```



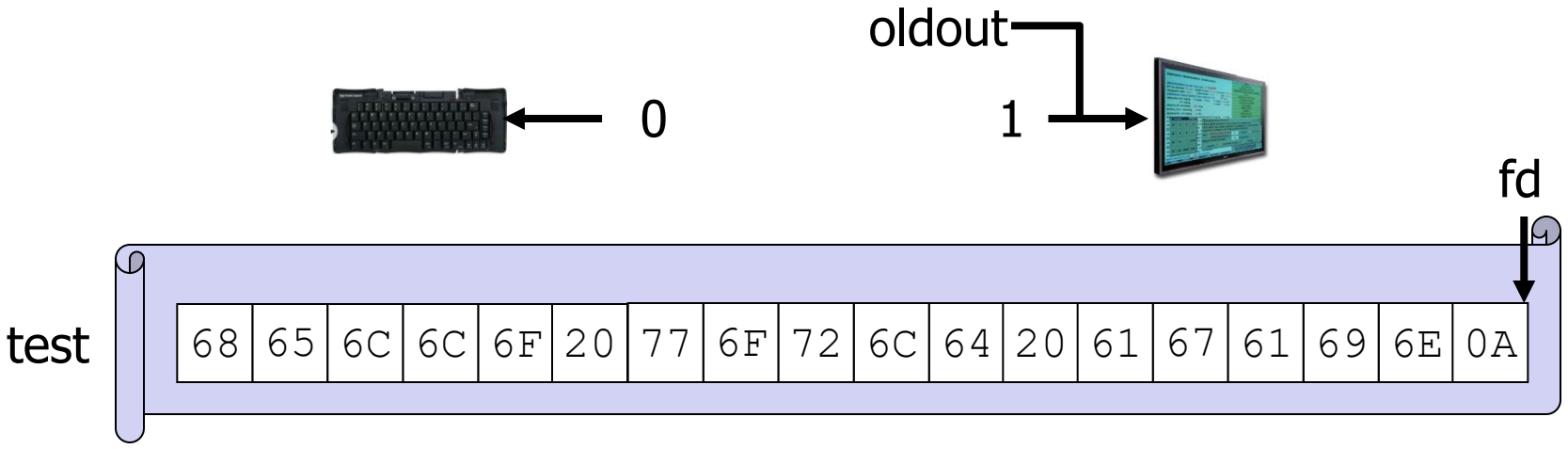
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```



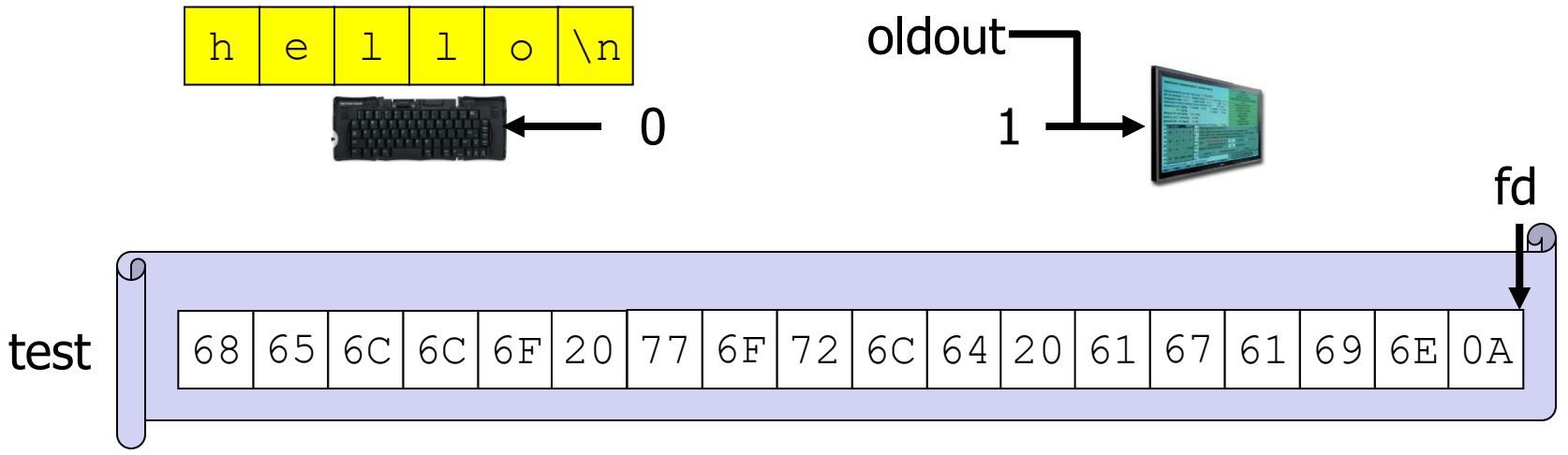
```
int fd, oldout, oldin;
char buf[16];

fd = open("test",O_RDWR|O_CREAT|O_TRUNC,S_IRWXU);
printf("hello world\n");
oldout = dup(STDOUT_FILENO); // copy stdout
dup2(fd,STDOUT_FILENO); // redirect stdout to fd
printf("hello world again\n");
dup2(oldout,STDOUT_FILENO); // restore stdout
printf("hi\n");
…
```



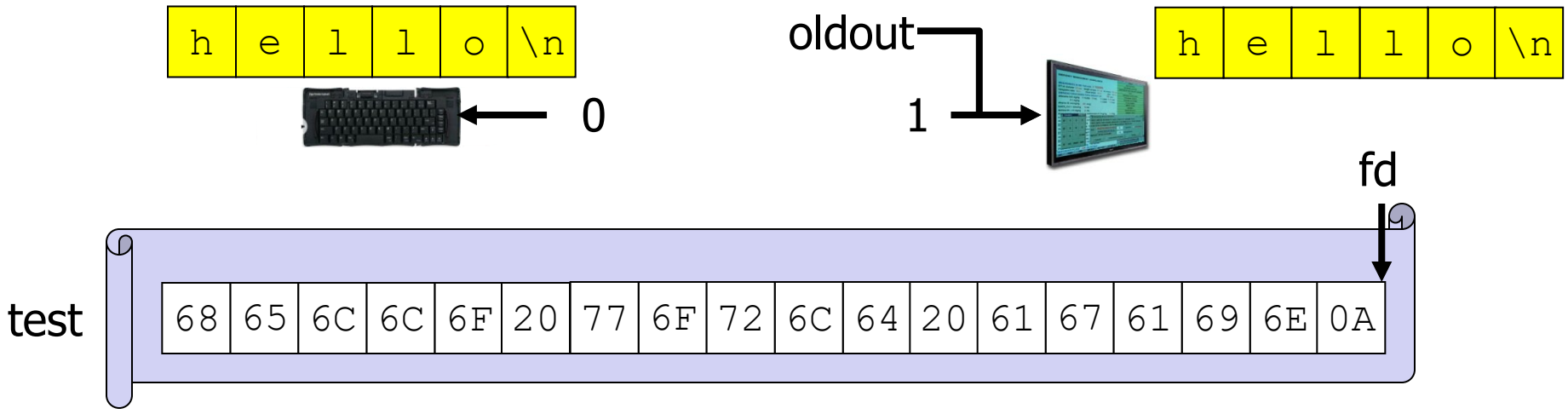
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



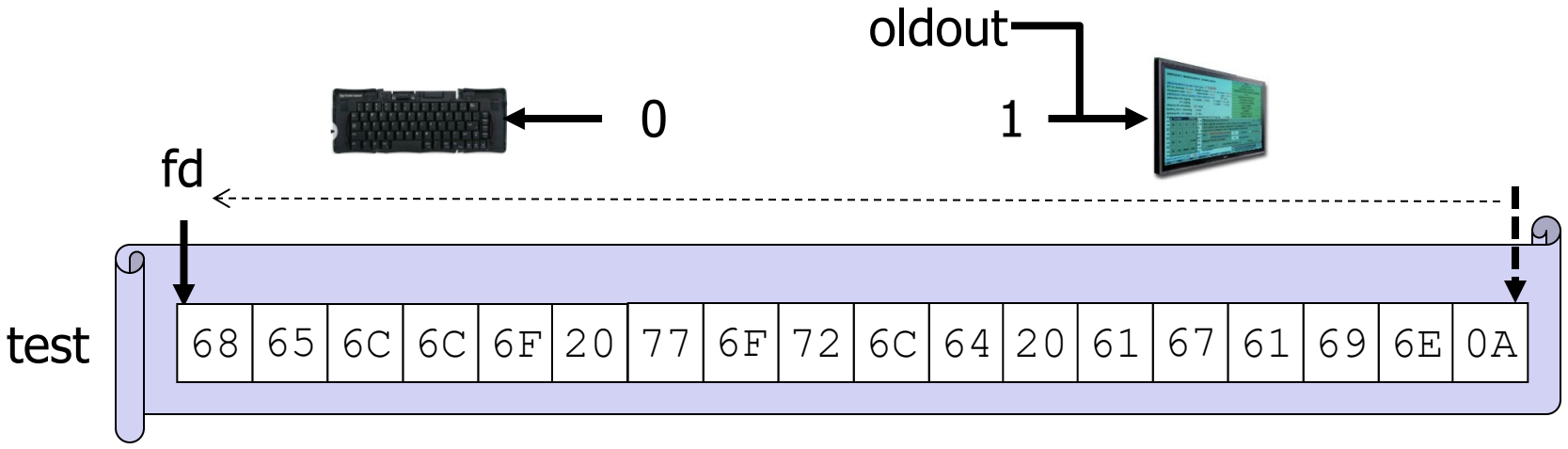
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



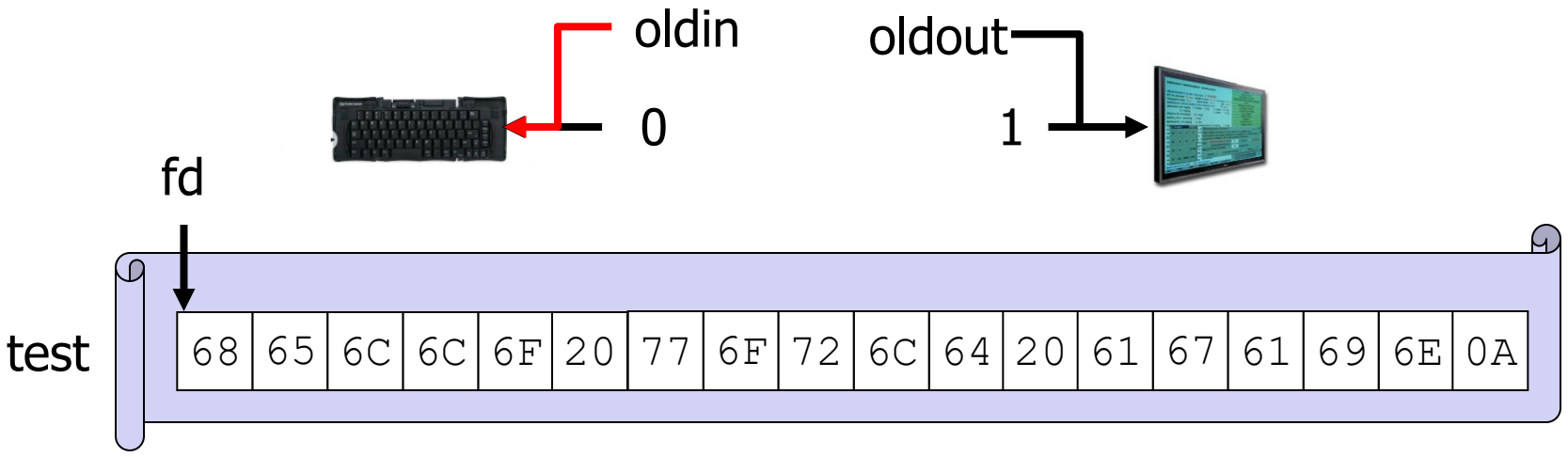
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



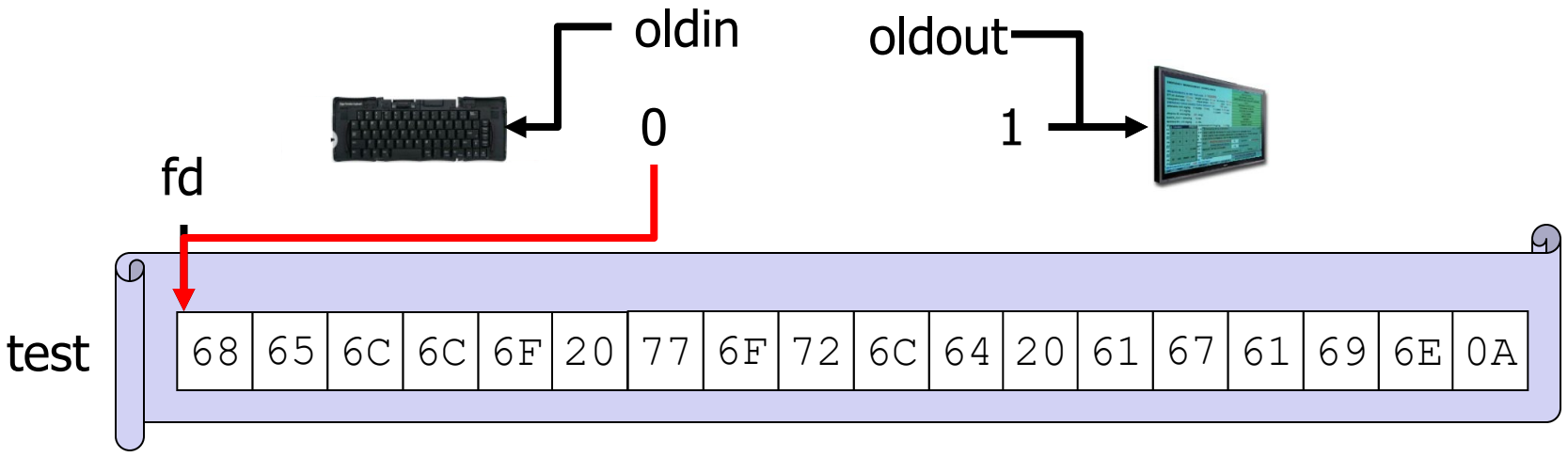
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



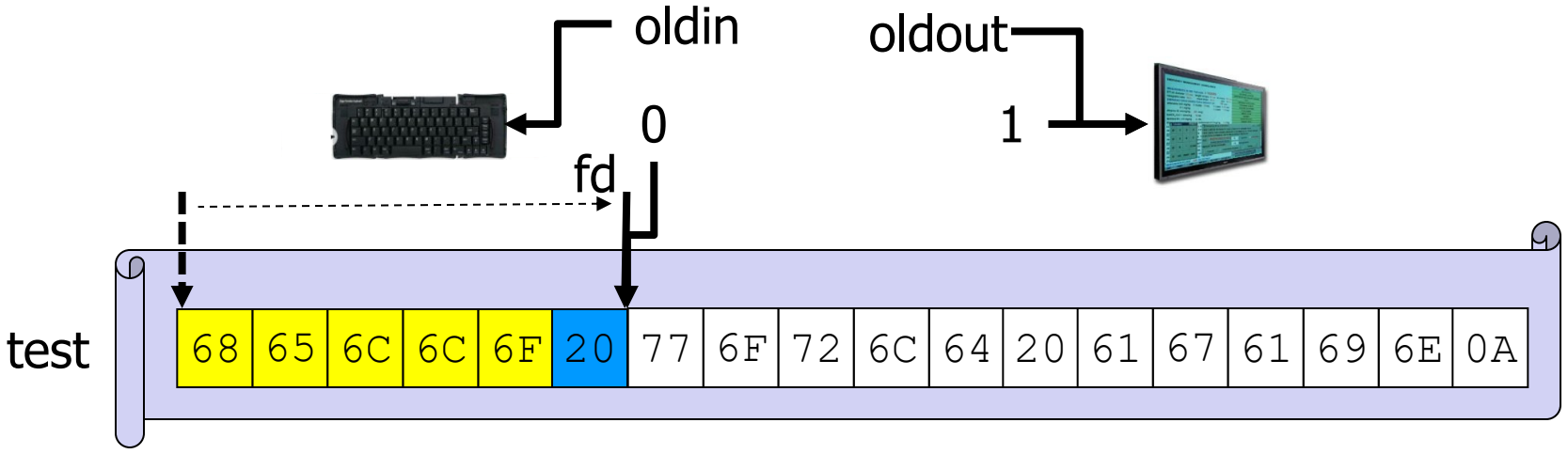
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



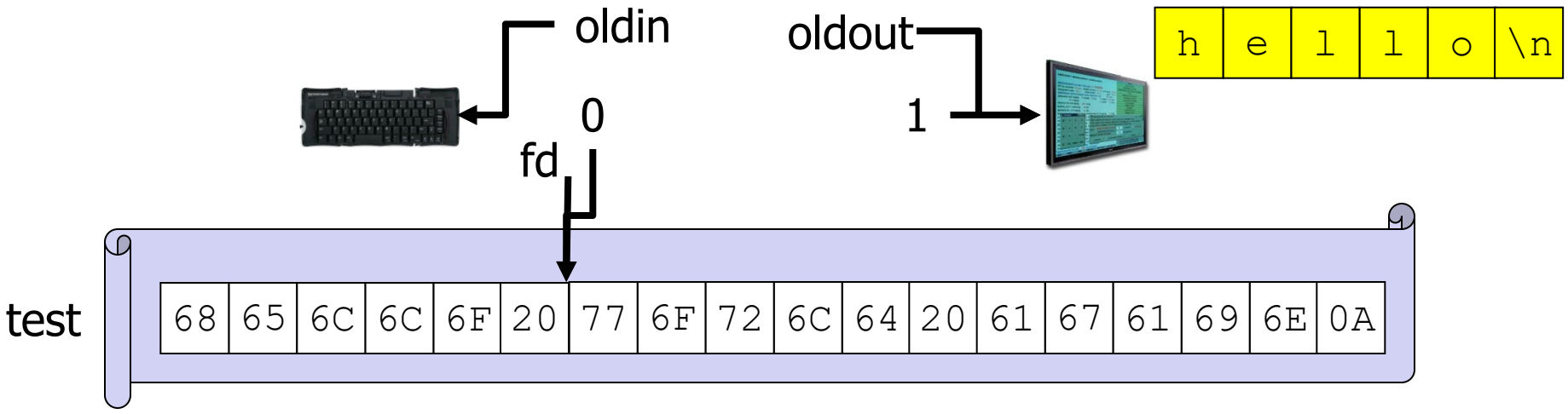
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



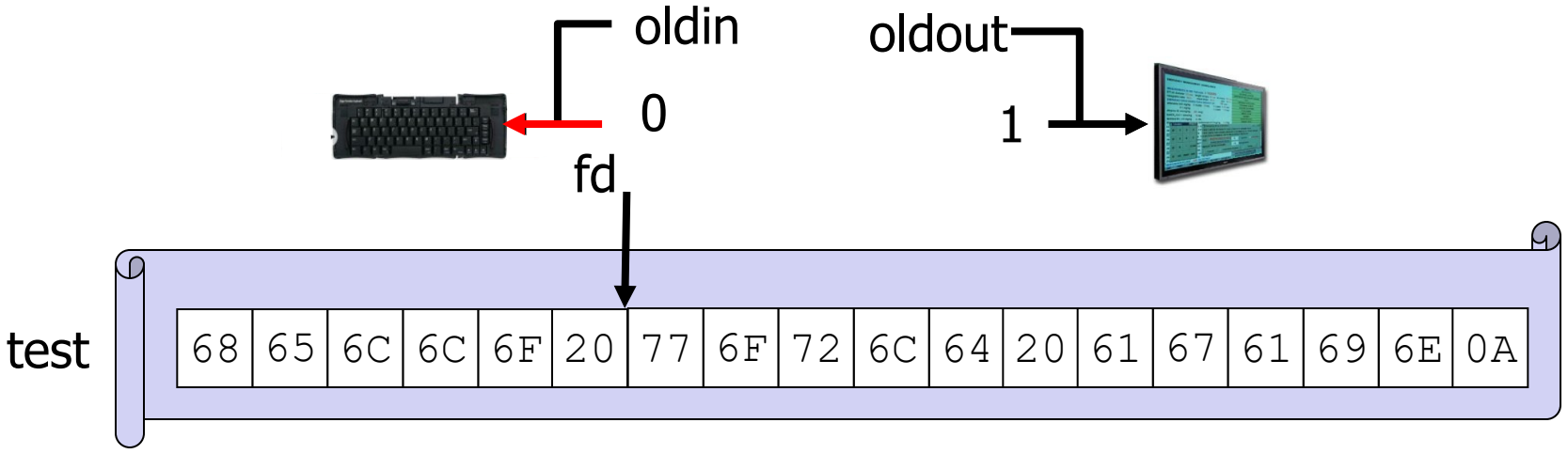
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



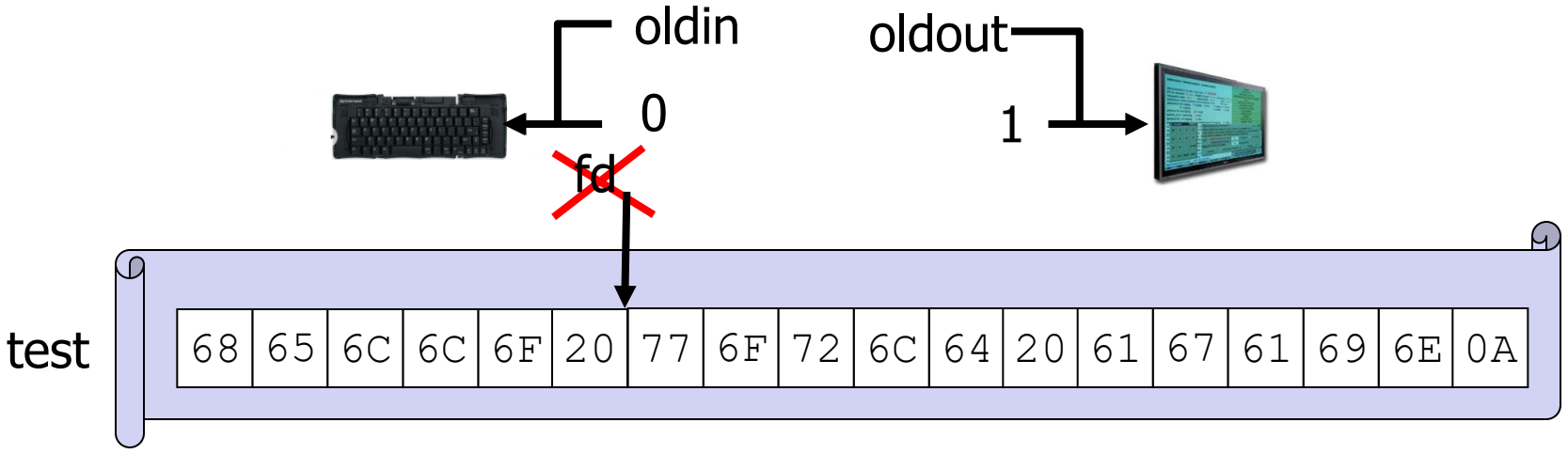
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



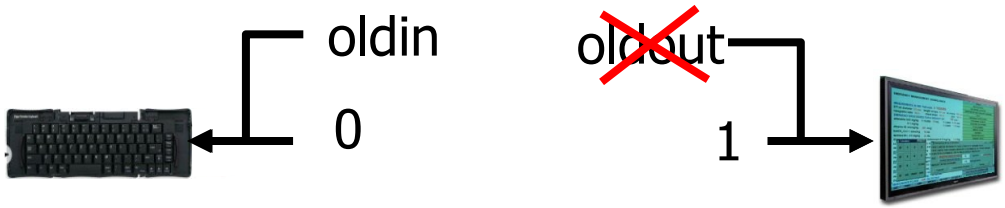
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```

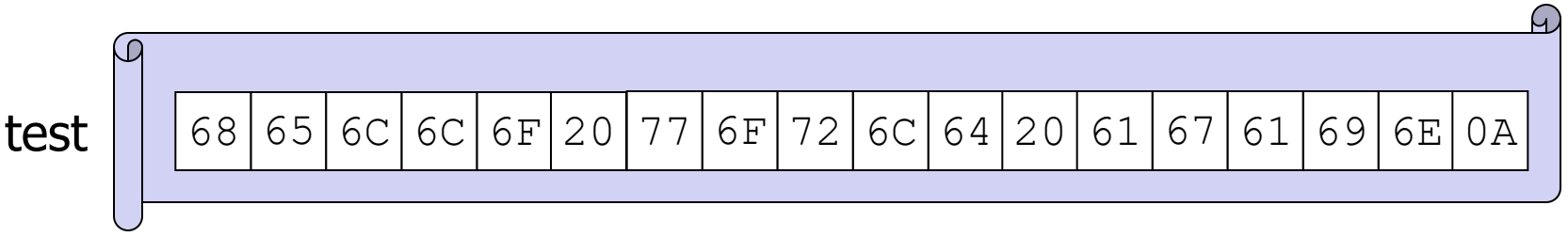


```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```

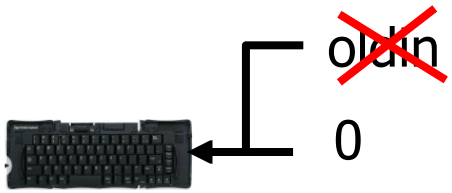


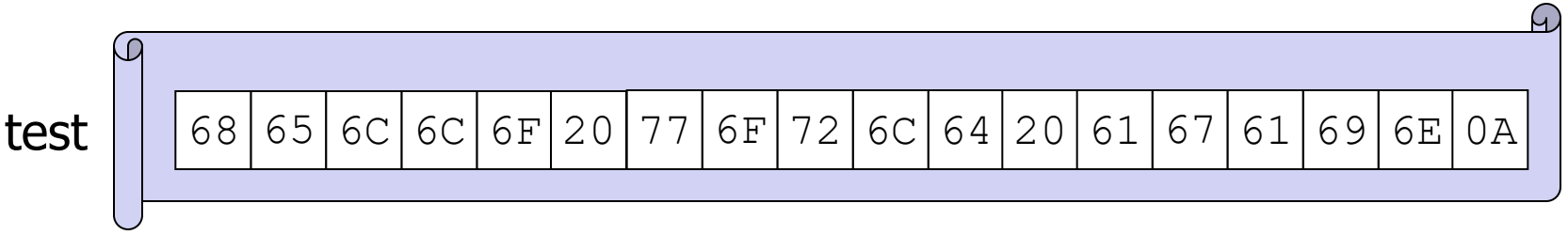
```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



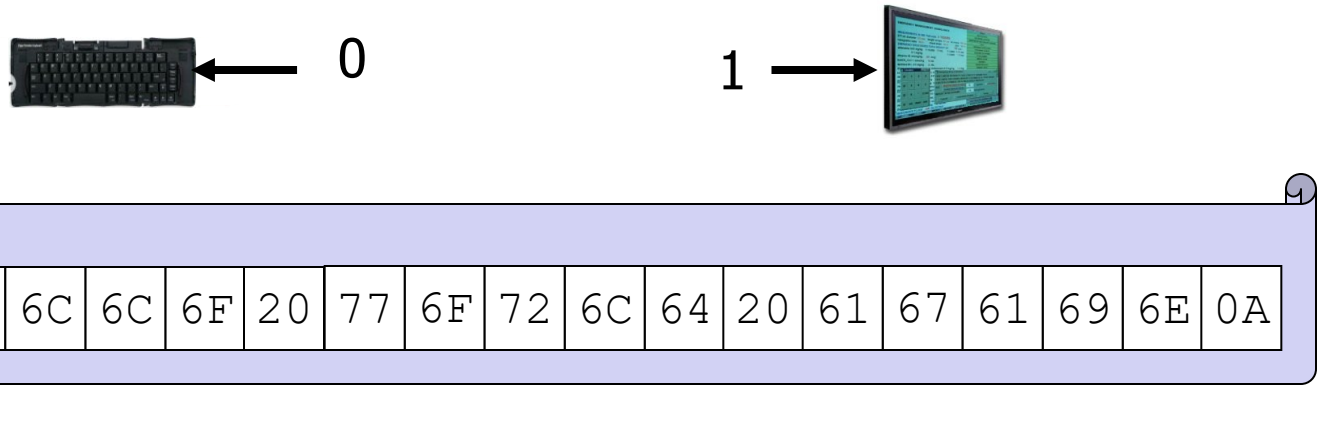


```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```





```
…
scanf("%s",buf); printf("%s\n",buf);
lseek(fd,0,SEEK_SET);
oldin = dup(STDIN_FILENO); // copy stdin
dup2(fd,STDIN_FILENO); // redirect stdin to fd
scanf("%s",buf); printf("%s\n",buf);
dup2(oldin,STDIN_FILENO); // restore stdin
close(fd);
close(oldout);
close(oldin);
```



Προσοχή!

- Η `stdio` εισάγει «δικό της» τύπο αρχείου μέσω του οποίου μπορεί κανείς να διαβάσει/γράψει δεδομένα
- Η `stdio` χρησιμοποιεί εσωτερικές αποθήκες
- Η χρήση της `stdio` ταυτόχρονα με κλήσεις συστήματος `read/write` πάνω στα ίδια αρχεία (περιγραφείς αρχείων) πρέπει να **αποφεύγεται**