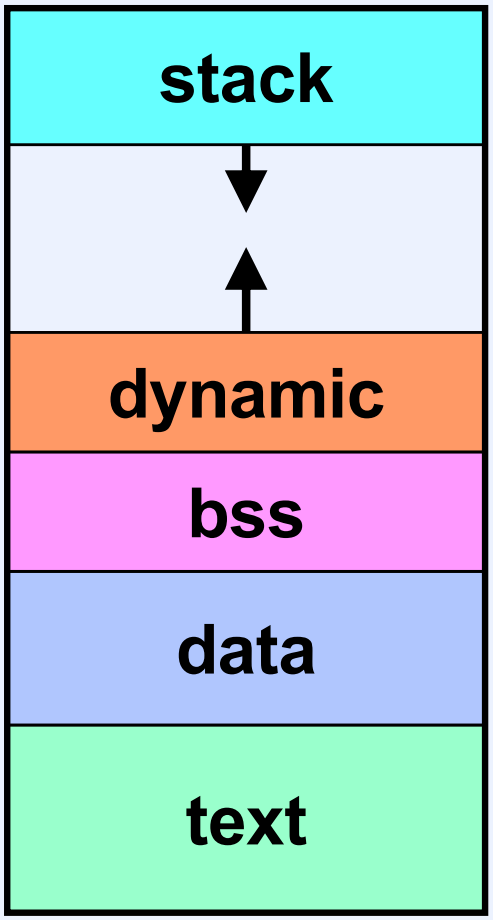
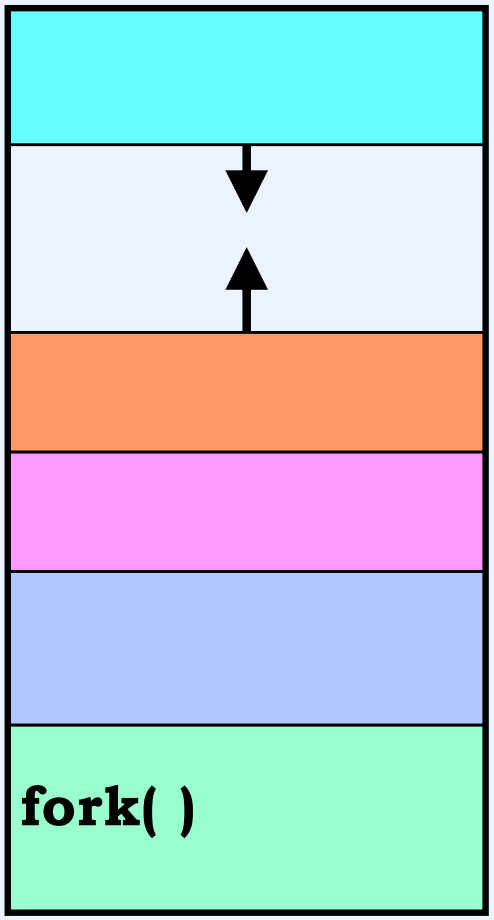
UNIX Structure

The Unix Address Space

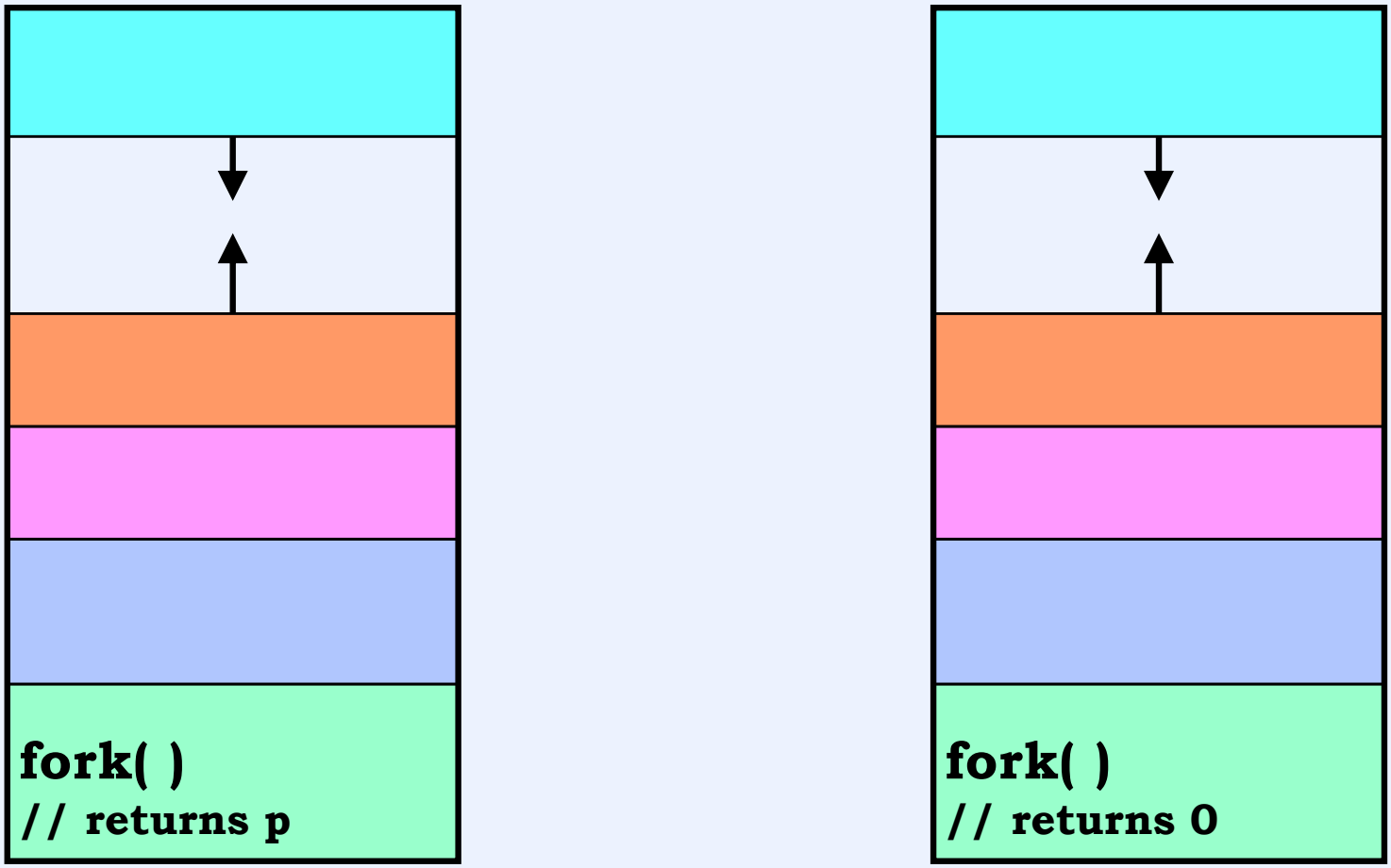


Creating a Process: Before



parent process

Creating a Process: After



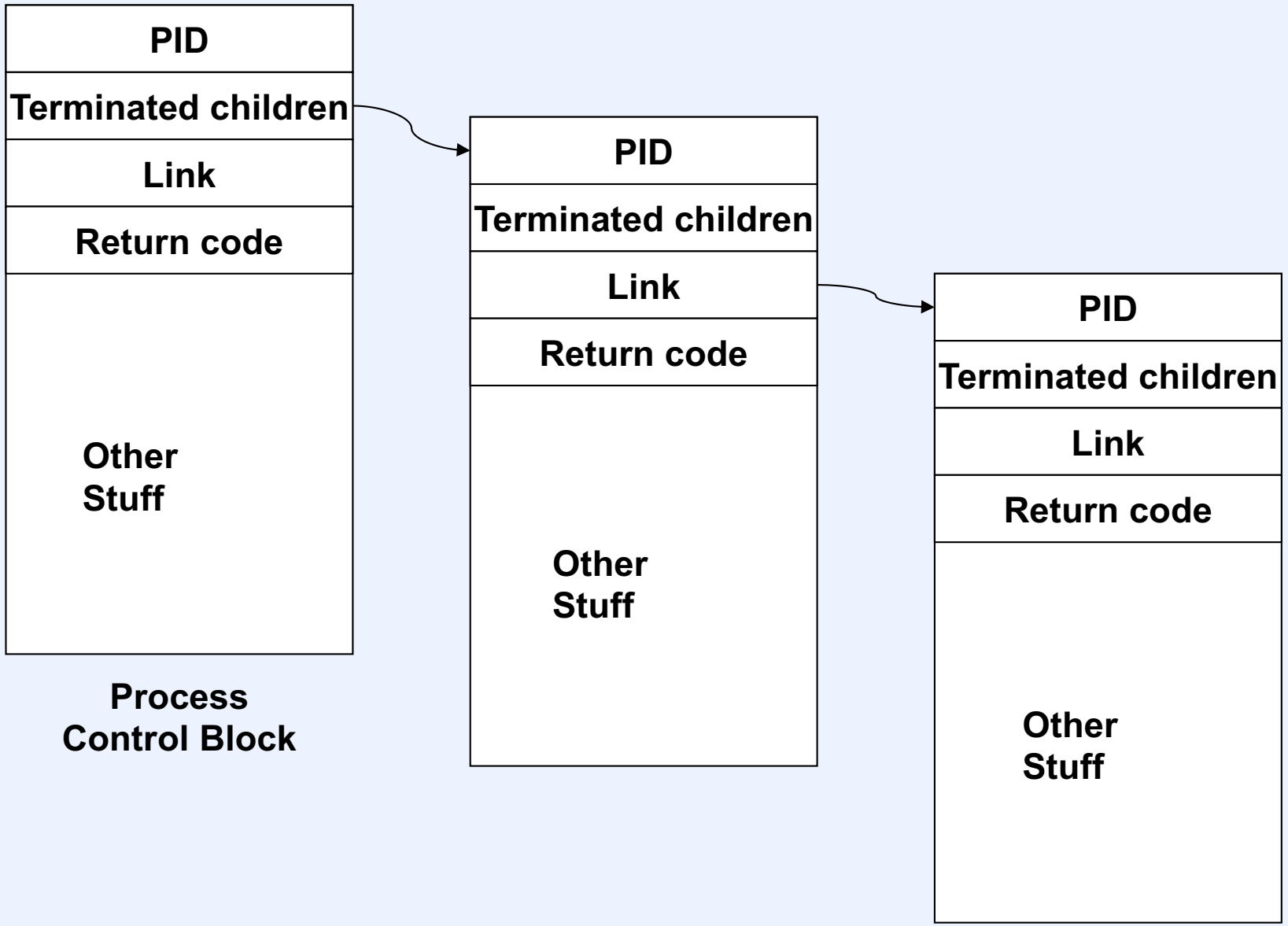
parent process

child process (pid = p)

Fork and Wait

```
short pid;
if ((pid = fork()) == 0) {
   /* some code is here for the child to execute */
   exit(n);
} else {
   int ReturnCode;
   while(pid != wait(&ReturnCode))
      ;
   /* the child has terminated with ReturnCode as its
      return code */
}
```

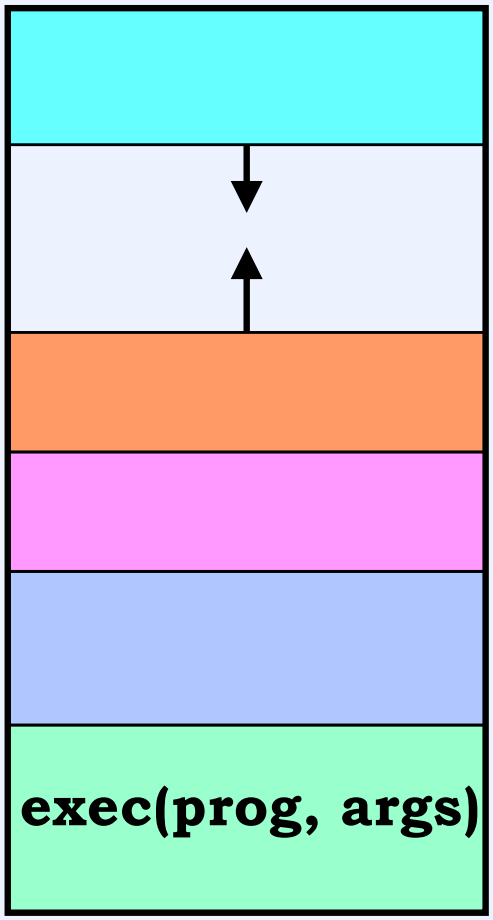
Process Control Blocks



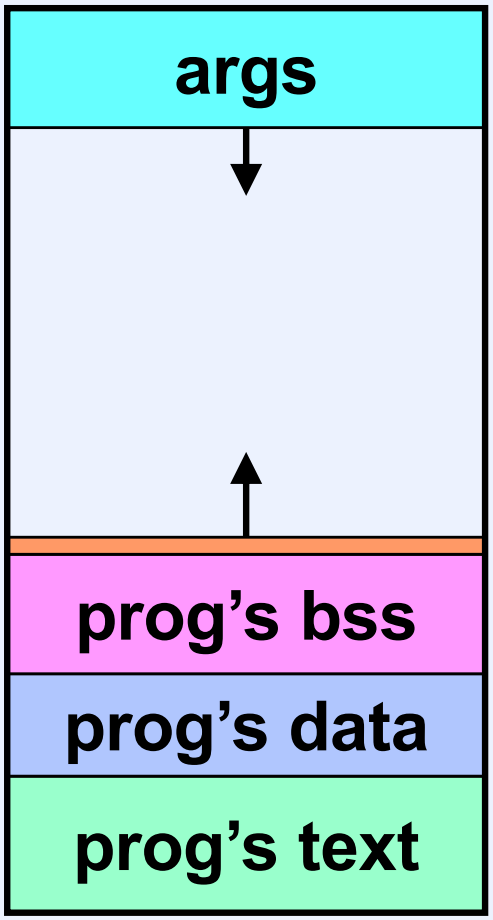
Exec

```
int pid;
if ((pid = fork()) == 0) {
   /* we'll soon discuss what might take place before exec
      is called */
   execl("/home/twd/bin/primes", "primes", "300", 0);
   exit(1);
}

/* parent continues here */

while(pid != wait(0))       /* ignore the return code */
   ;
```

Loading a New Image



Before



After

Quiz 1

```
int A=0, B=0, C=0, D=0;
A=1;
if (fork() > 0) {
  B=1;
  A=111;
} else {
  C=2;
  if (fork() > 0) {
    D=222;
  } else {
    D=A+B+C;
    // what value is now
    // in D for this process?
  }
}
exit(0);
```

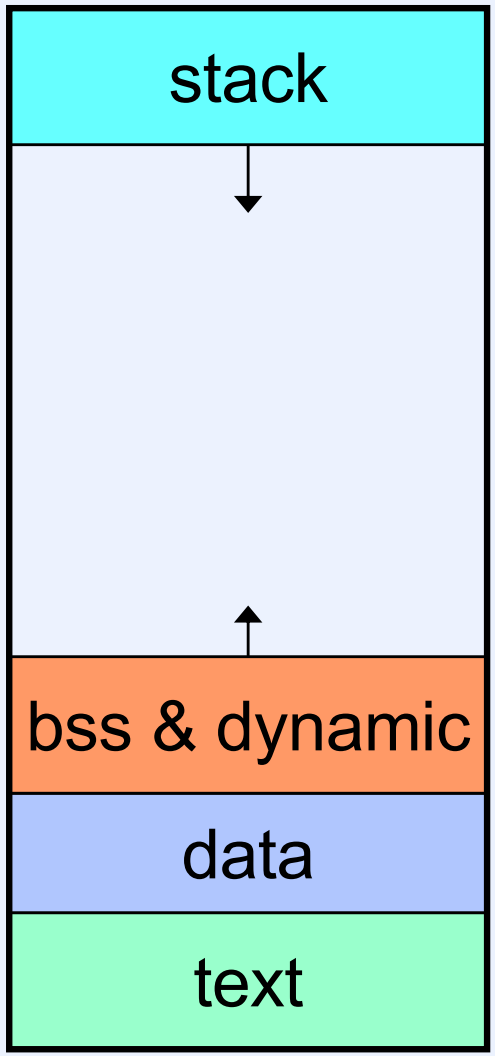
Answer:

a) 0
b) 3
c) 113
d) indeterminate

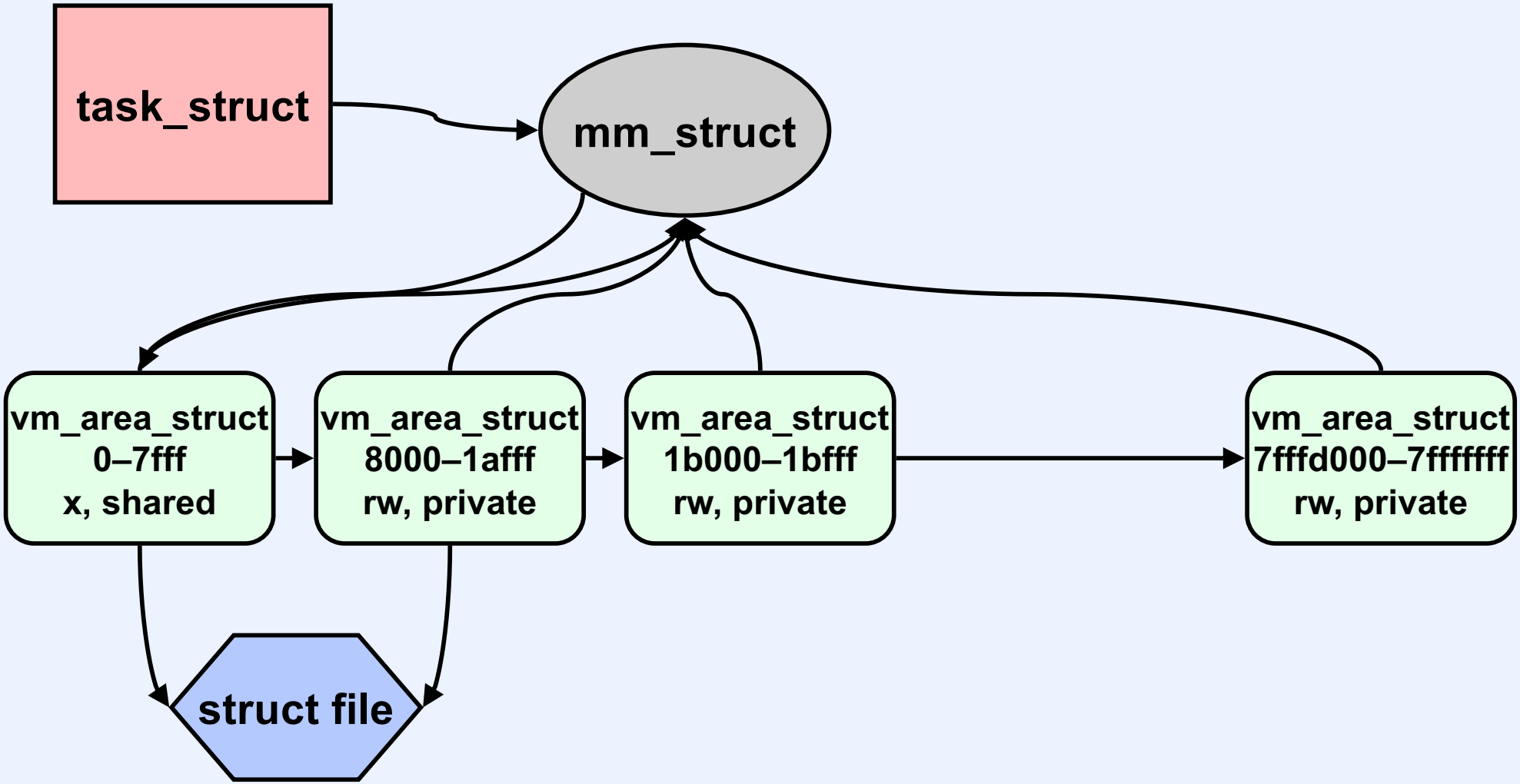
Representing the Address Space

- Important component of a process is its address space
  - how is it represented?
- Can page tables represent a process's address space?

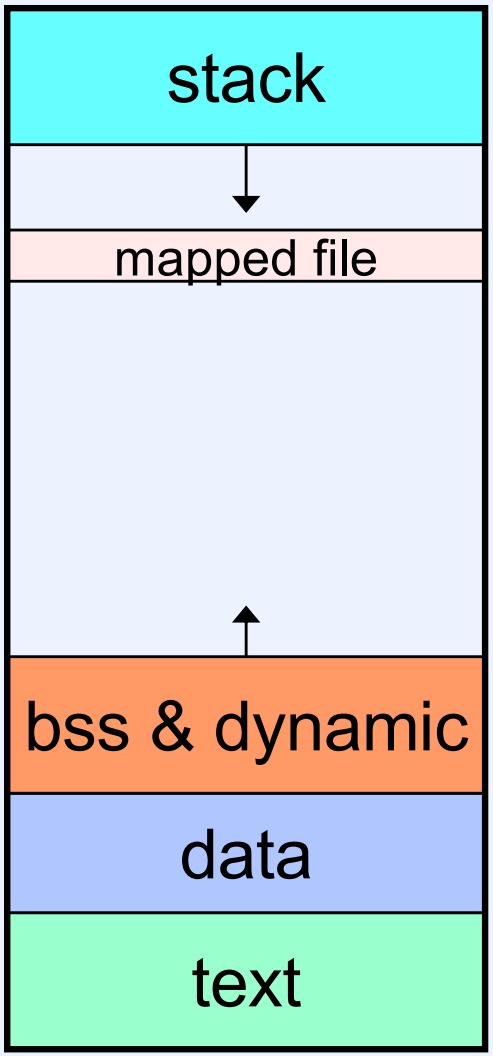
Simple User Address Space



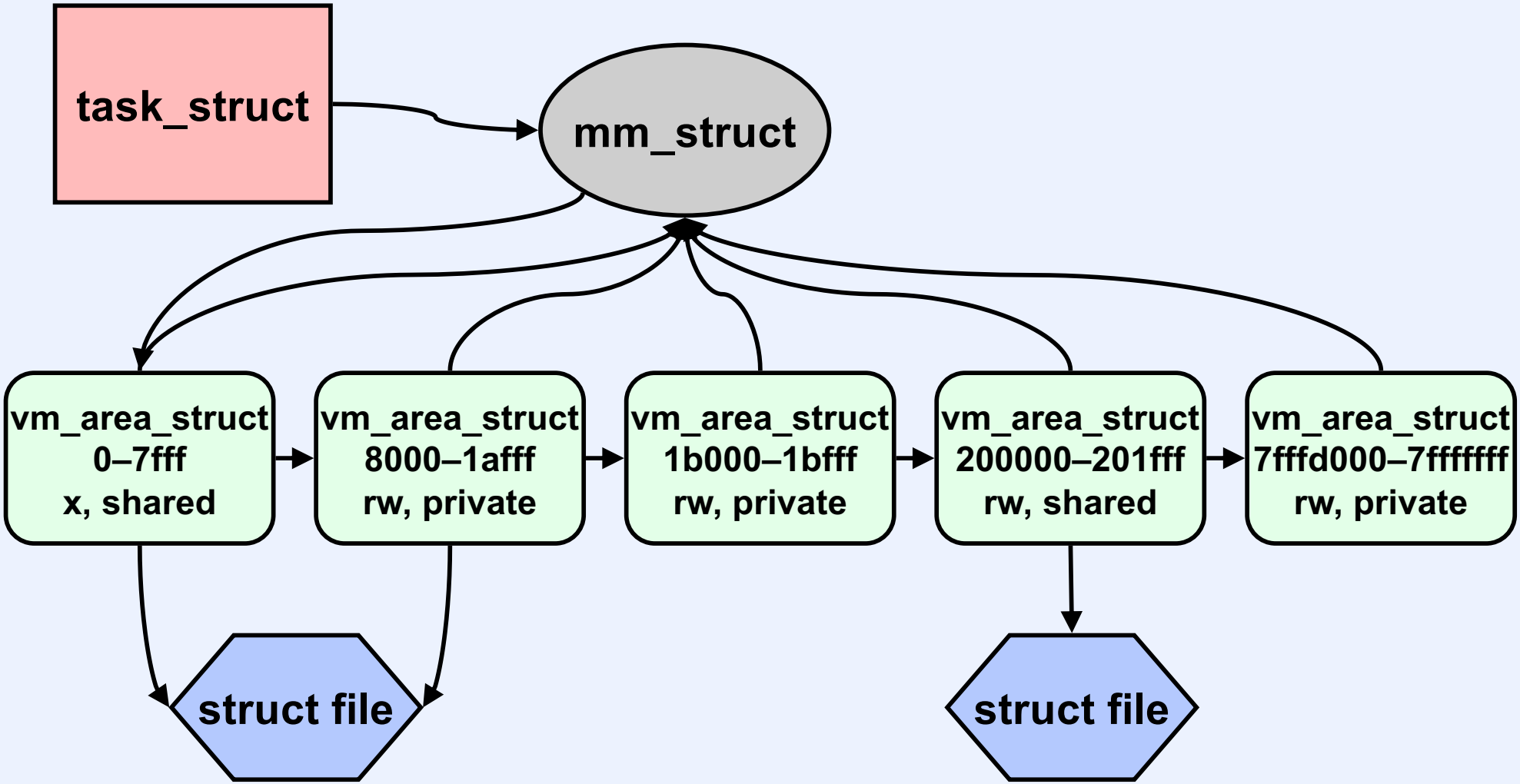
Address-Space Representation
Somewhat Simplified



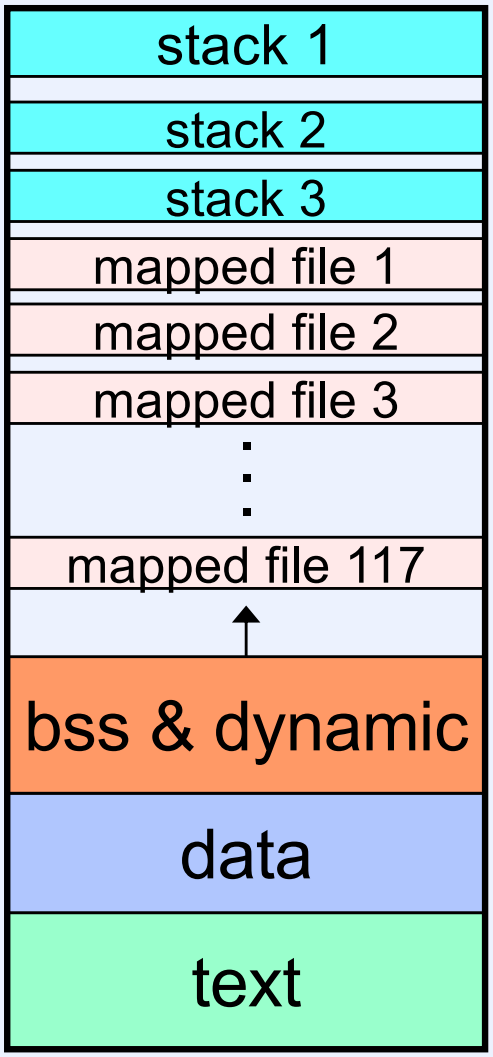
Adding a Mapped File



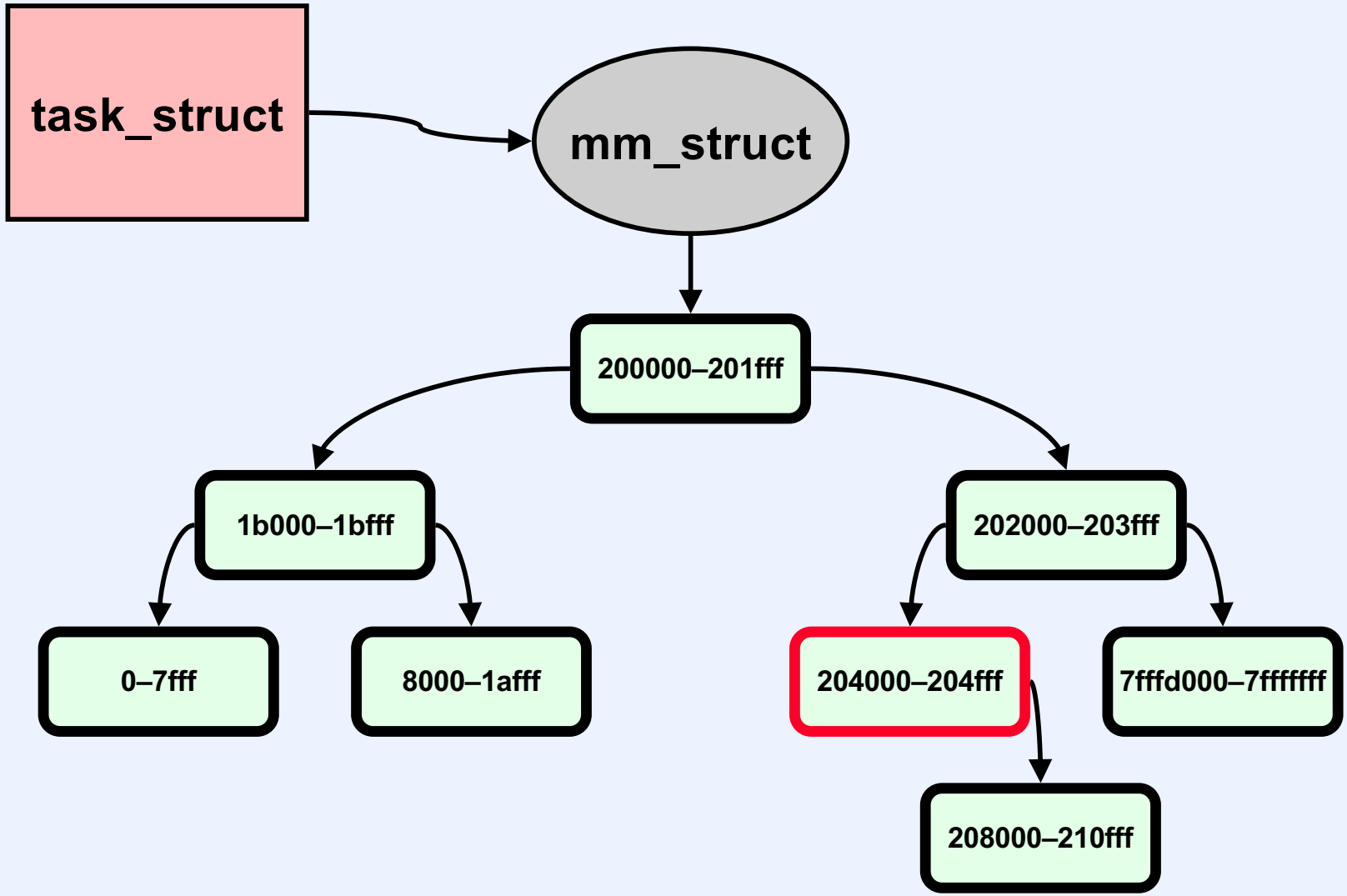
Address-Space Representation:
More Areas



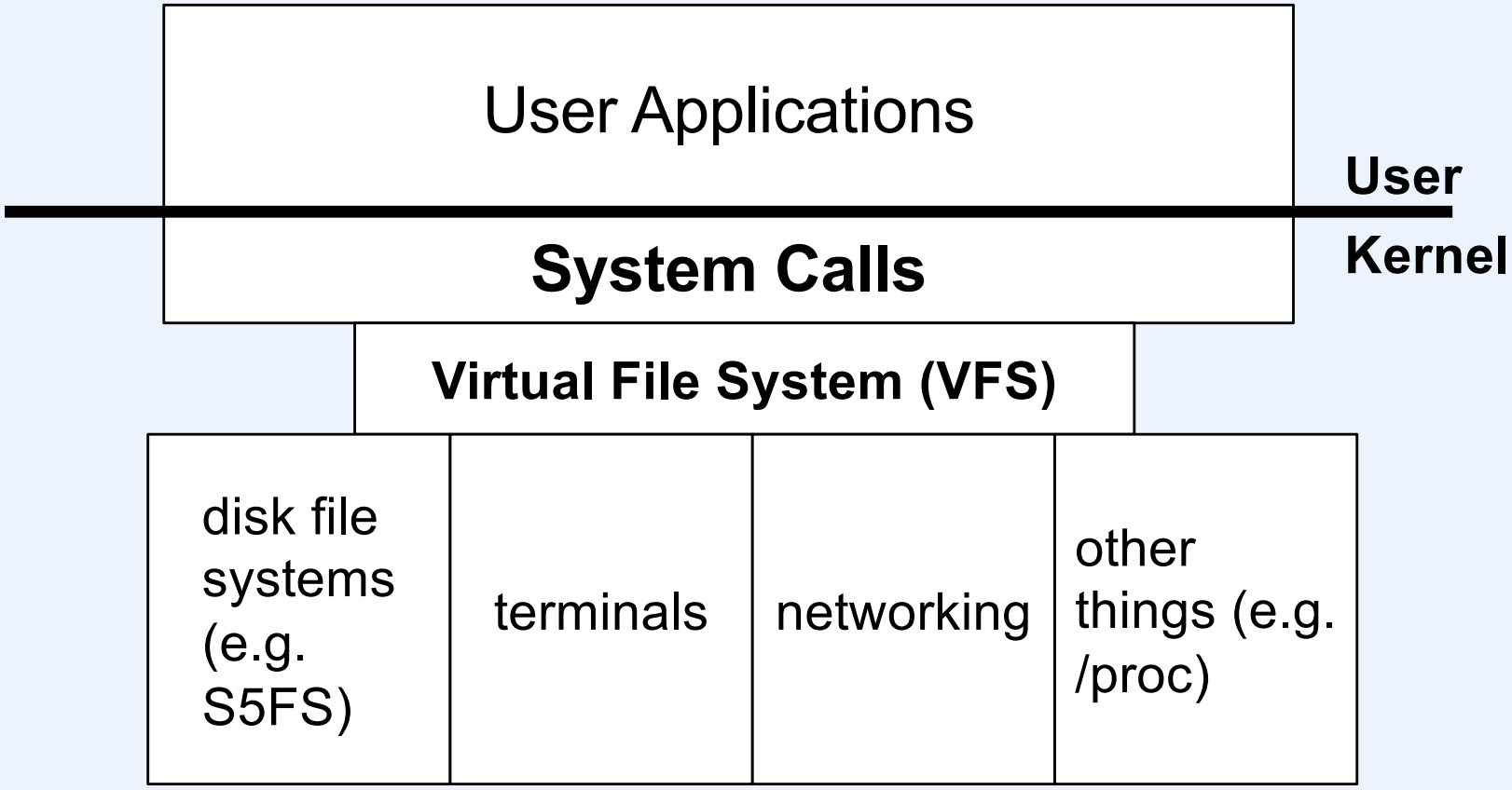
Adding More Stuff



Address-Space Representation:
Reality



Layering



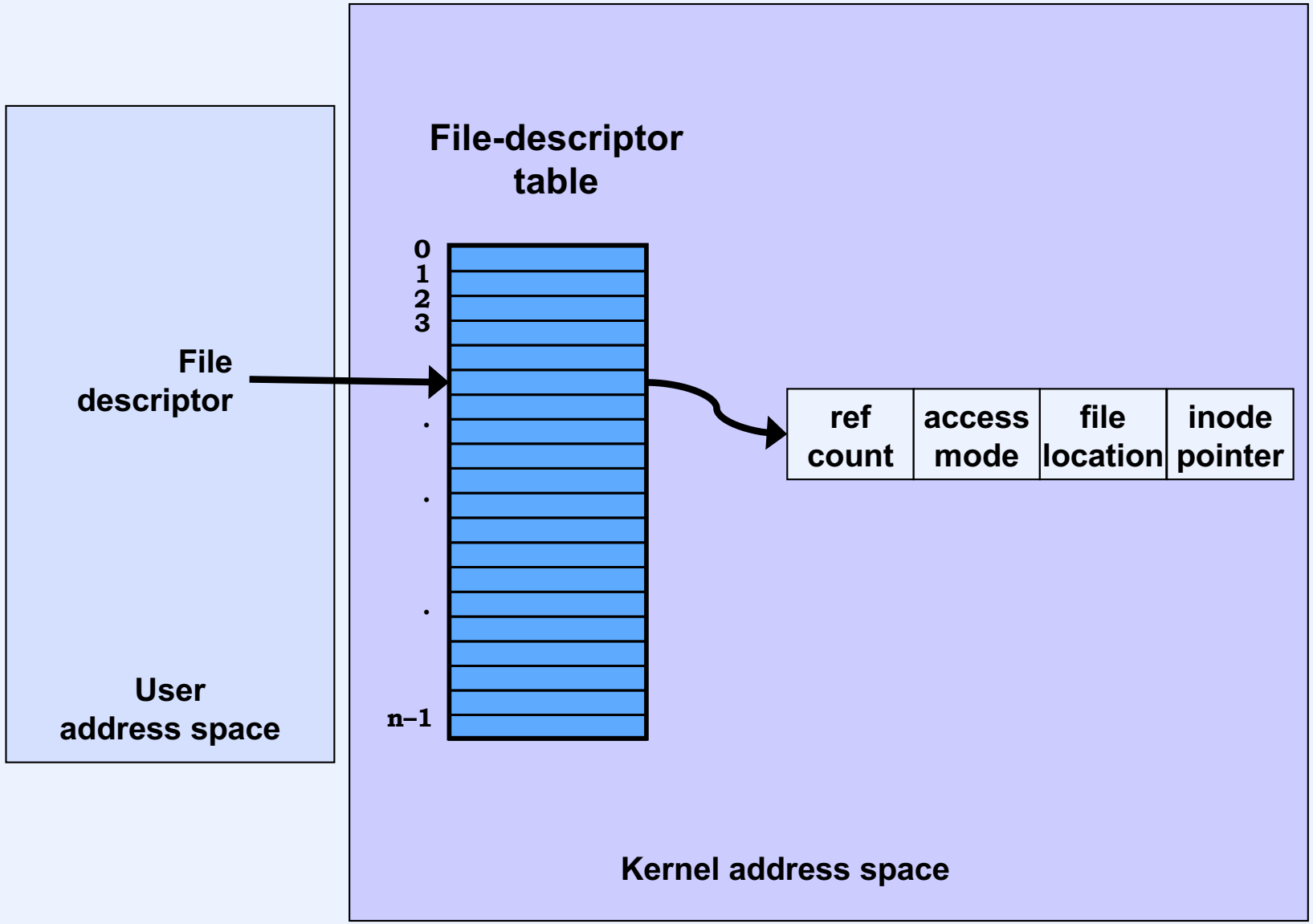
Quiz 2

```
int fd1 = open("file", O_CREAT|O_RDWR, 0666);
unlink("file");
write(fd1, "123", 3);
int fd2 = open("file", O_CREAT|O_RDWR, 0666);
write(fd2, "4", 1);
if (fork() == 0) {
  write(fd1, "5", 1);
}
exit(0);
```

The final contents of file are:

a) 12345
b) 453
c) 45
d) 4

File-Descriptor Table



Allocation of File Descriptors

- **Whenever a process requests a new file descriptor, the lowest numbered file descriptor not already associated with an open file is selected; thus**

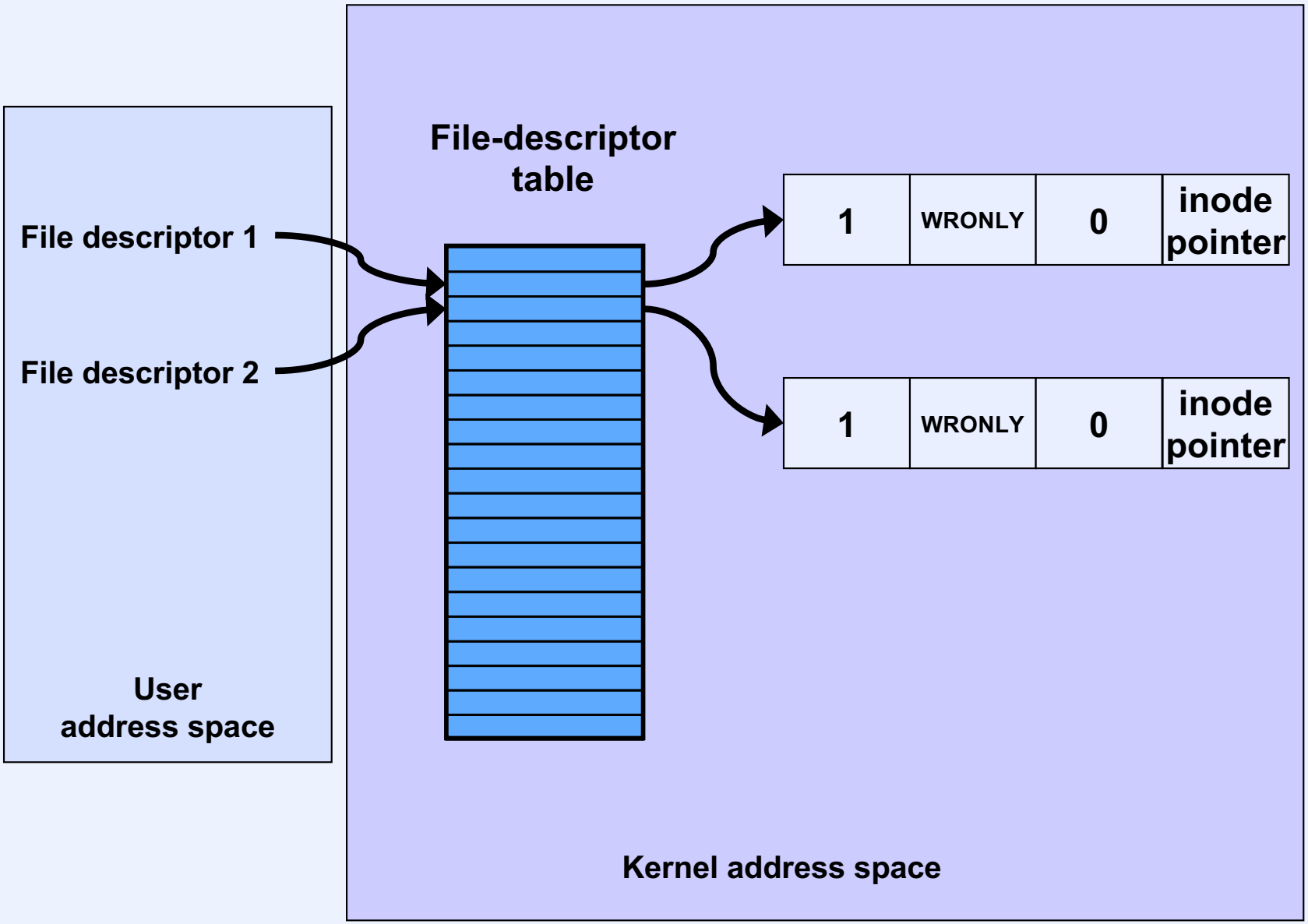
```
#include <fcntl.h>
#include <unistd.h>

close(0);
fd = open("file", O_RDONLY);
```

  - **will always associate *file* with file descriptor 0 (assuming that the *open* succeeds)**

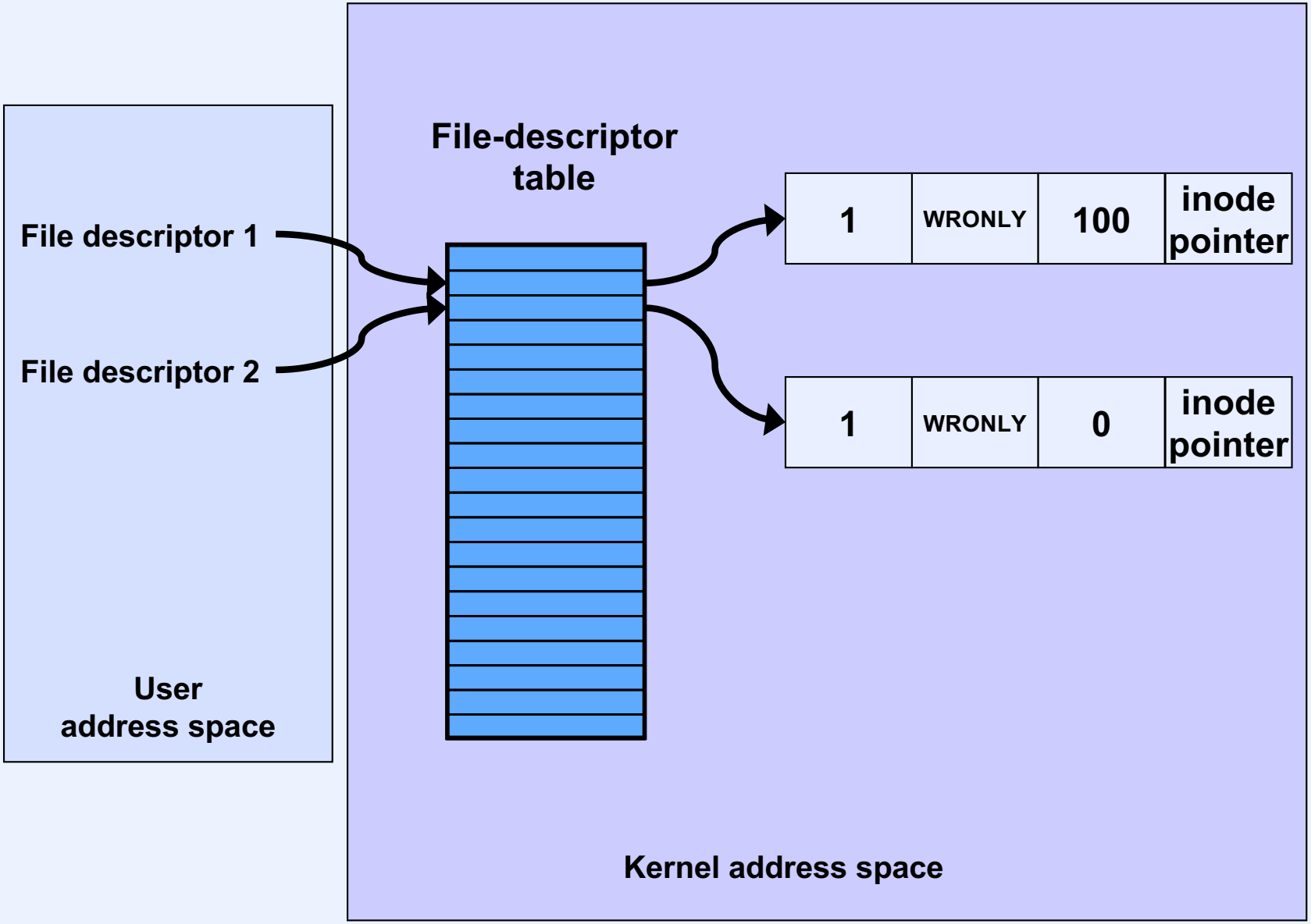
Redirecting Output … Twice

```
if (fork() == 0) {
   /* set up file descriptors 1 and 2 in the child process */
   close(1);
   close(2);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   execl("/home/twd/bin/program", "program", 0);
   exit(1);
}

/* parent continues here */
```

Redirected Output



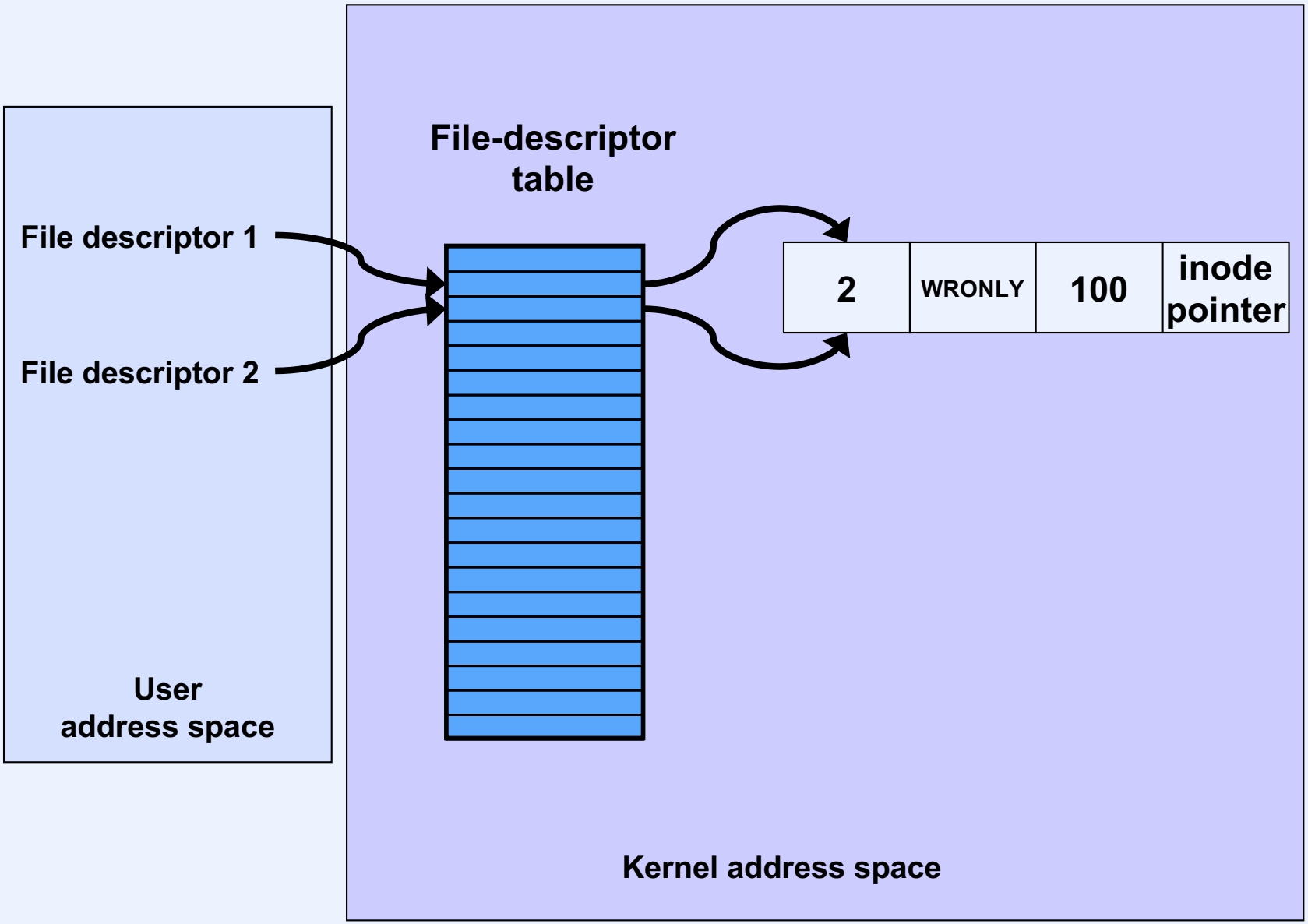
Redirected Output After Write



Sharing Context Information

```
if (fork() == 0) {
   /* set up file descriptors 1 and 2 in the child process */
   close(1);
   close(2);
   if (open("/home/twd/Output", O_WRONLY) == -1) {
      exit(1);
   }
   dup(1); /* set up file descriptor 2 as a duplicate of 1 */
   execl("/home/twd/bin/program", "program", 0);
   exit(1);
}
/* parent continues here */
```

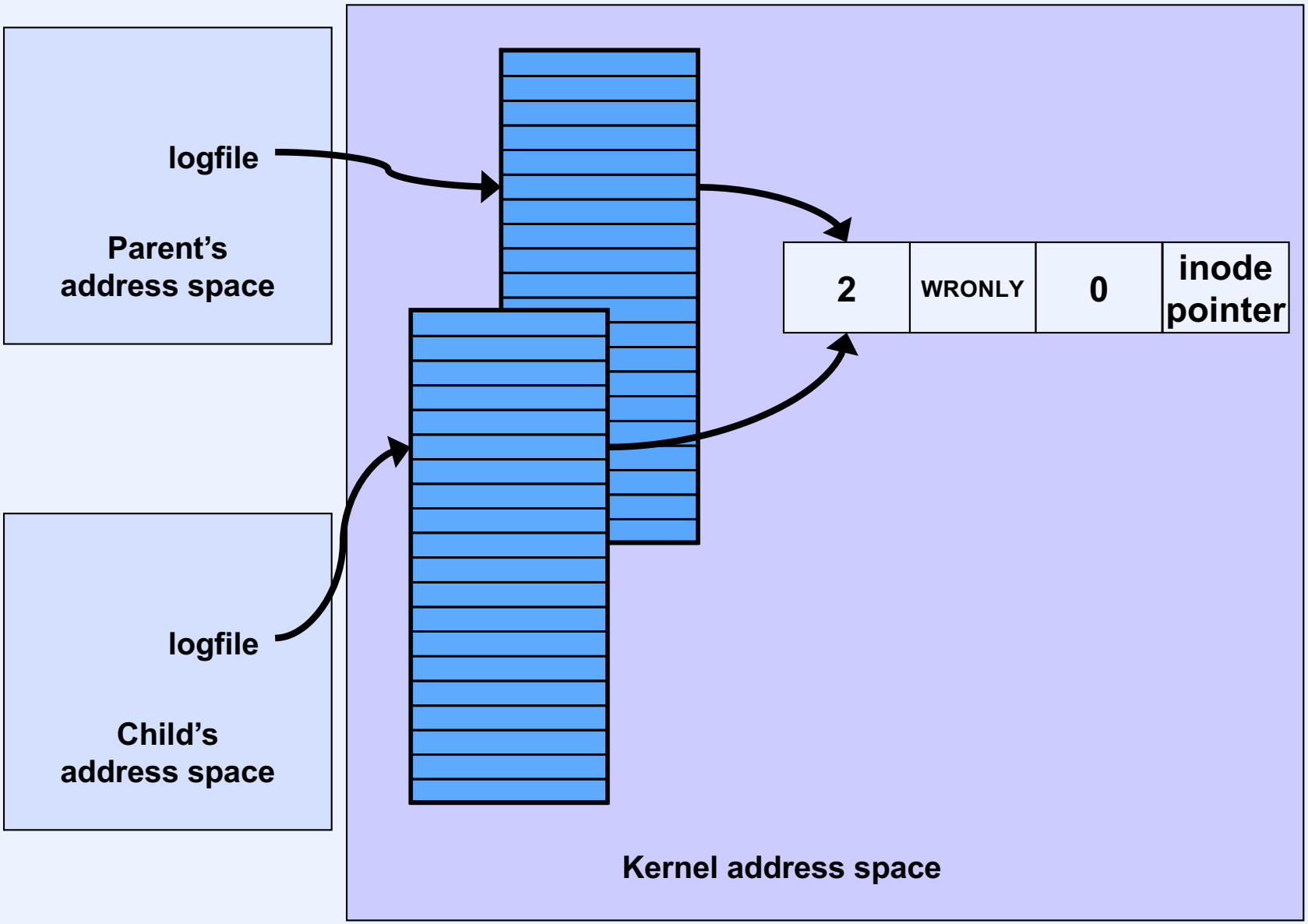
Redirected Output After Dup



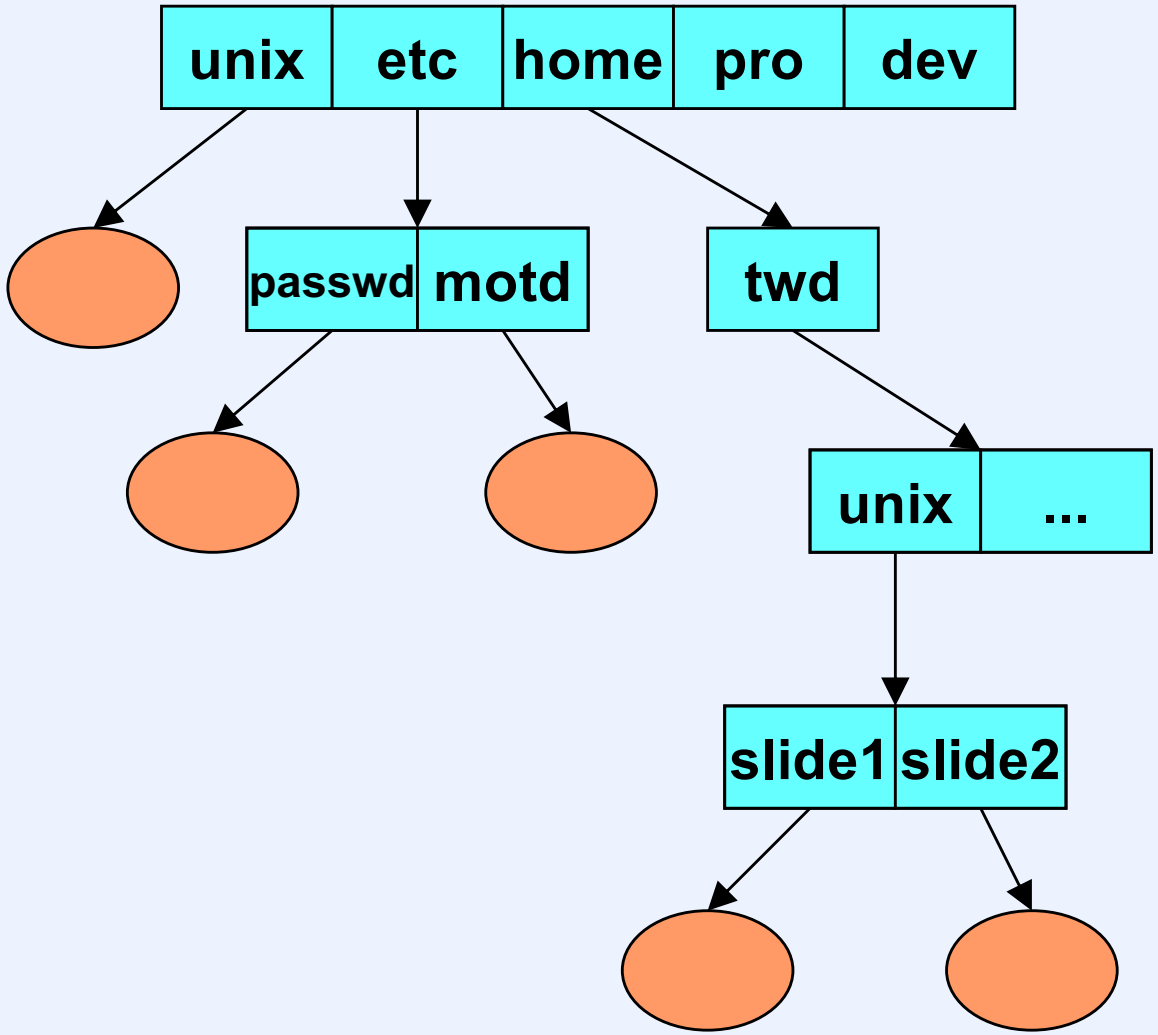
Fork and File Descriptors

```
int logfile = open("log", O_WRONLY);
if (fork() == 0) {
   /* child process computes something, then does: */
   write(logfile, LogEntry, strlen(LogEntry));
   …
   exit(0);
}

/* parent process computes something, then does: */

write(logfile, LogEntry, strlen(LogEntry));
…
```

File Descriptors After Fork



Directories



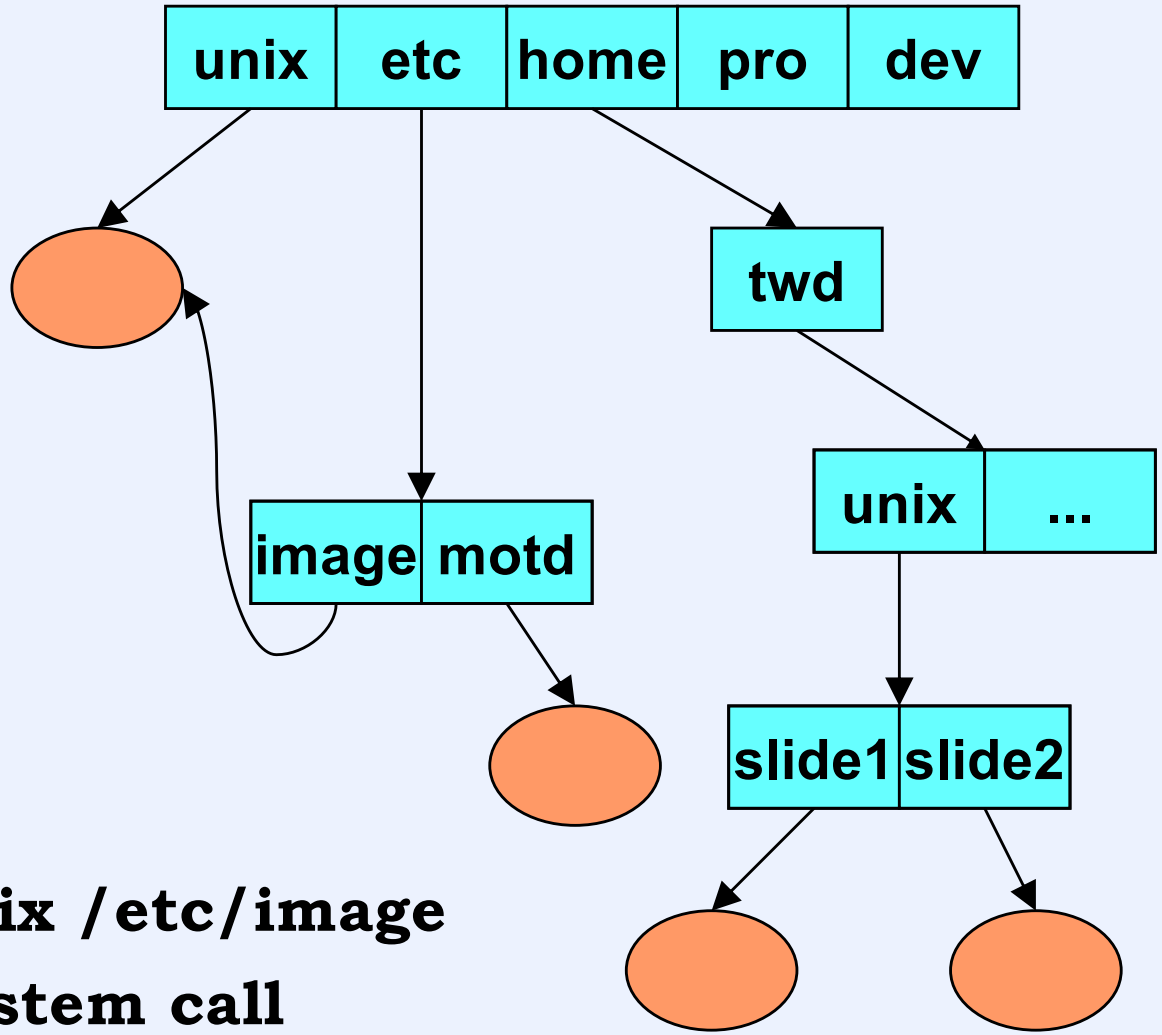
Directory Representation

Component Name	Inode Number

directory entry

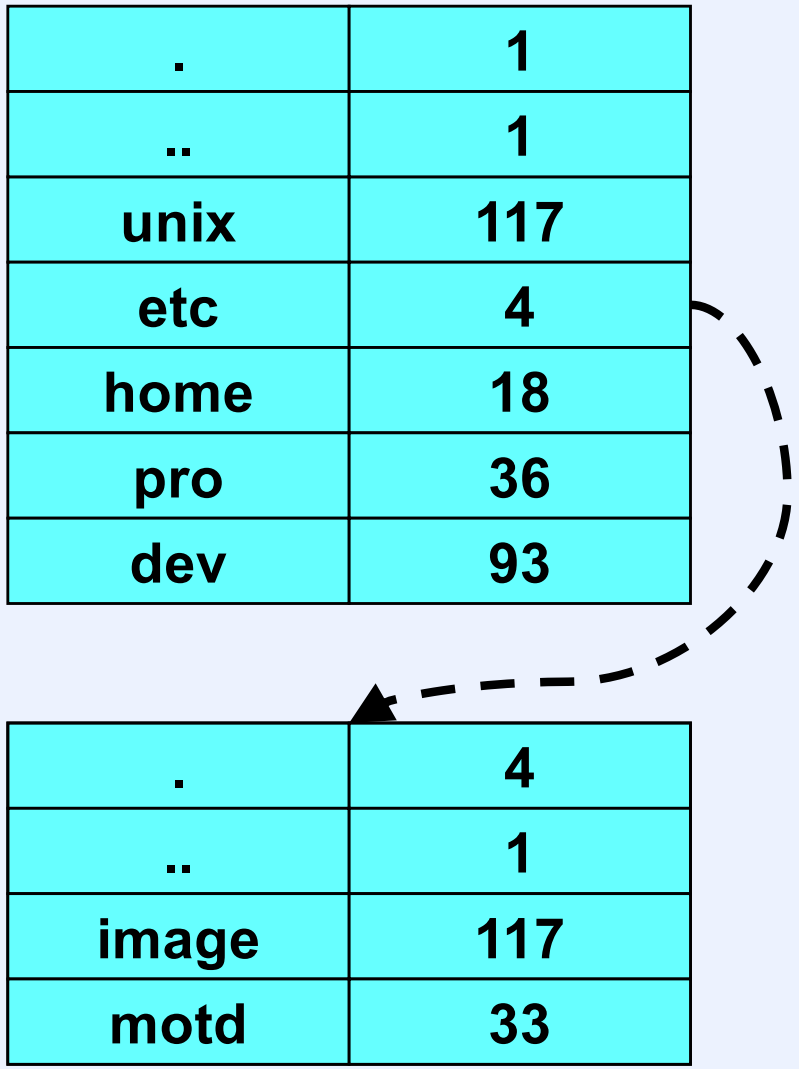
.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

Hard Links

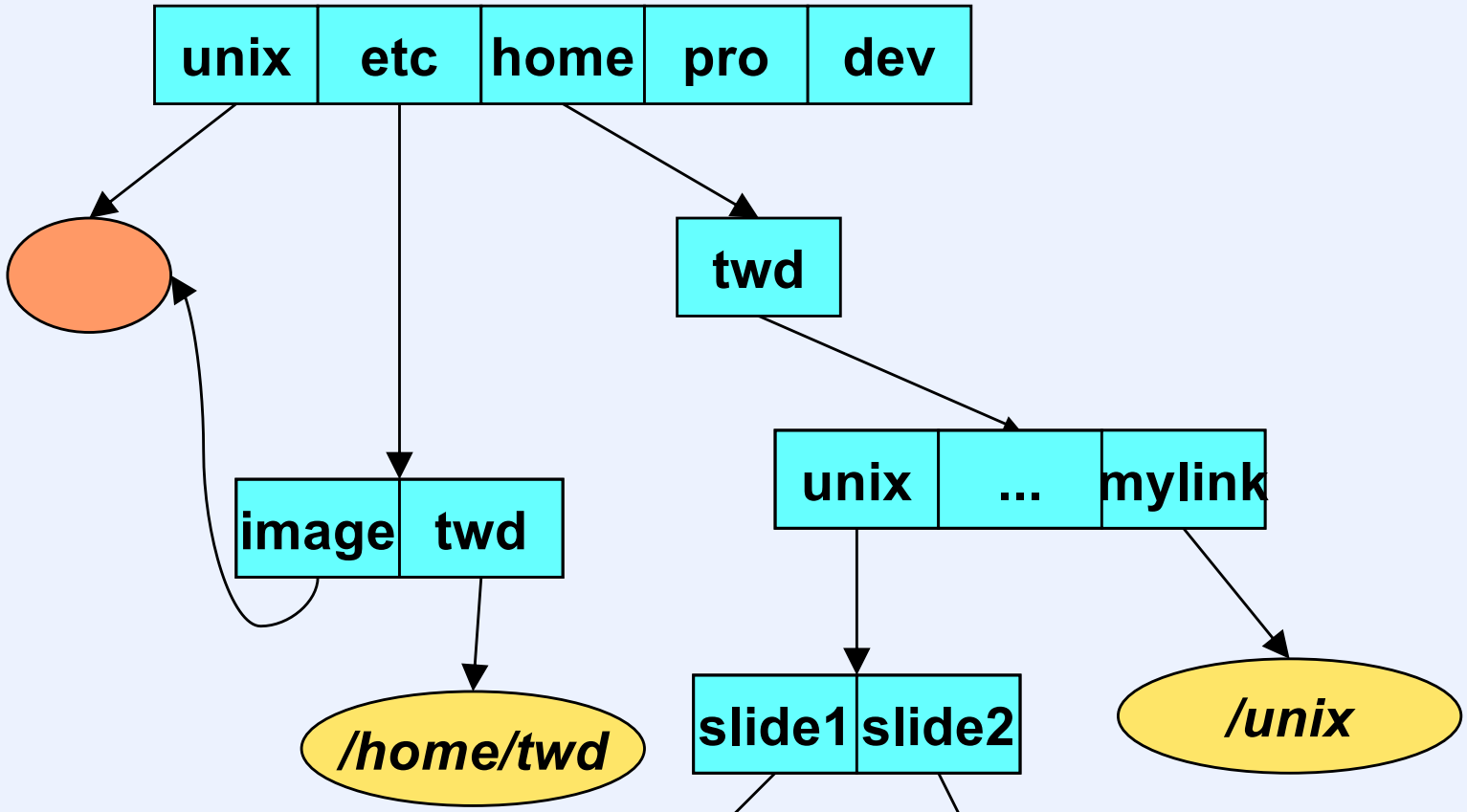



```
% ln /unix /etc/image
# link system call
```

Directory Representation



Soft Links

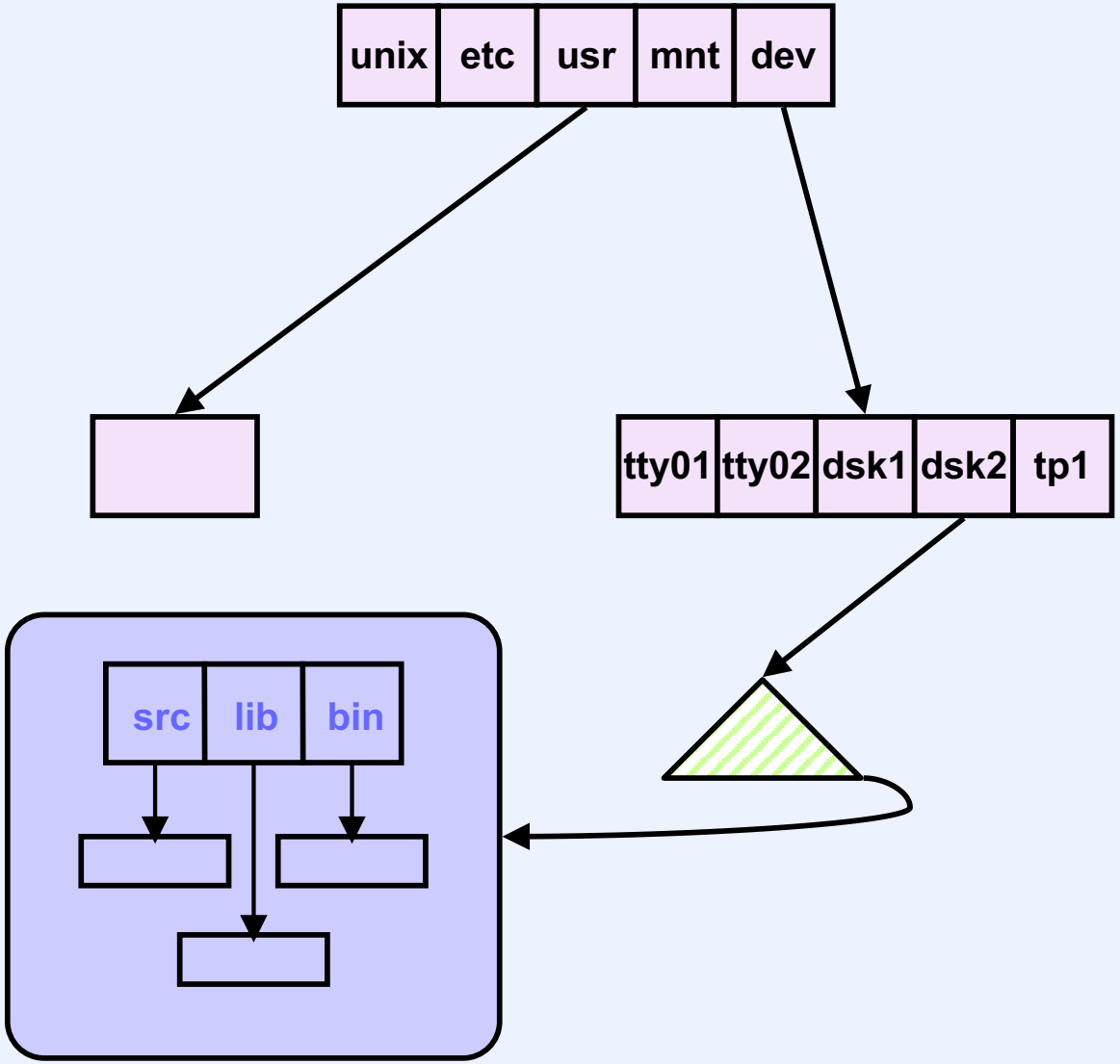



```
% ln –s /unix /home/twd/mylink
% ln –s /home/twd /etc/twd
# symlink system call
```

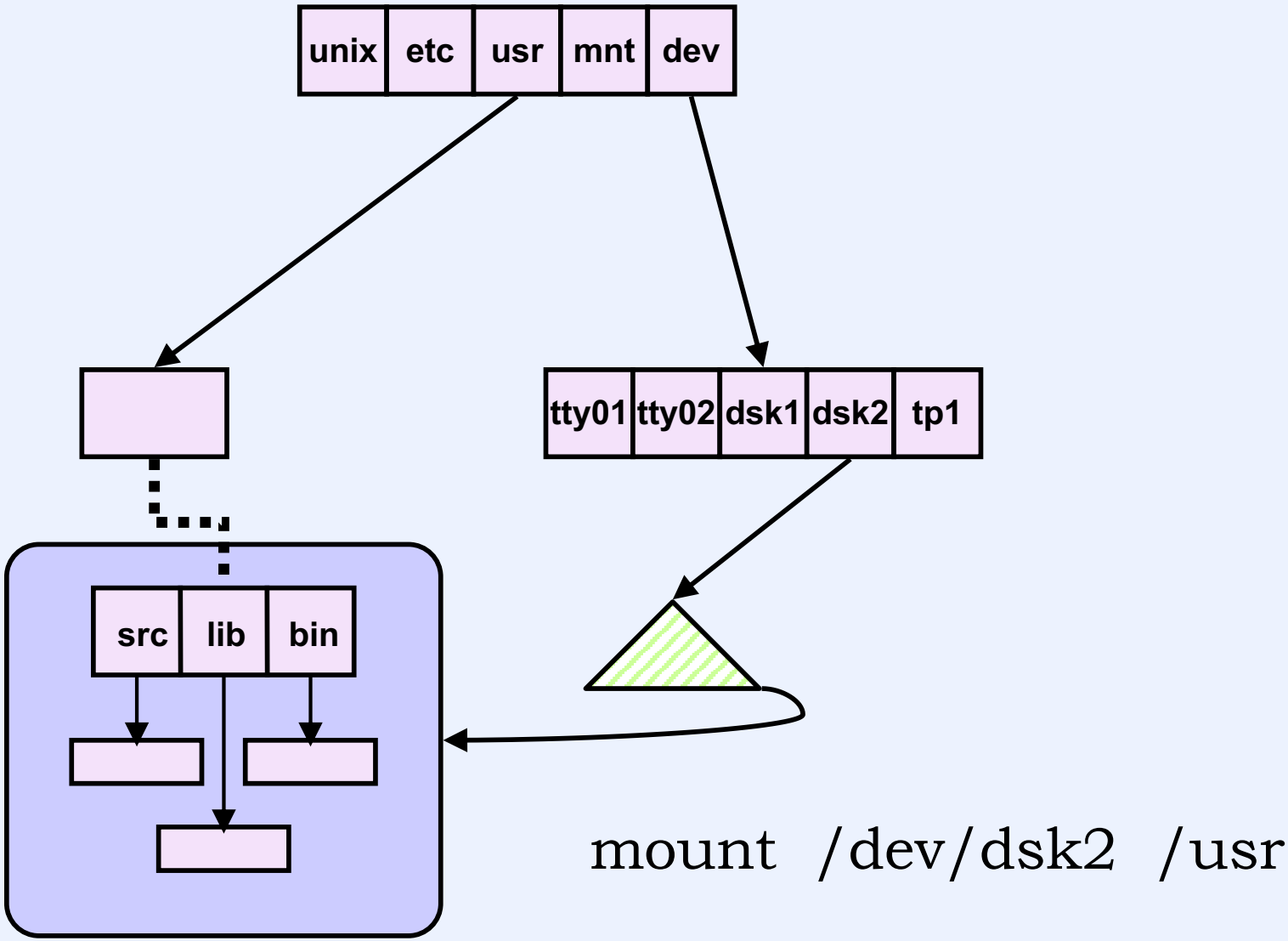
Working Directory

- Maintained in kernel for each process
  - paths not starting from "/" start with the working directory
  - changed by use of the *chdir* system call
  - displayed (via shell) using "pwd"
    - how is this done?

Mount Points (1)



Mount Points (2)



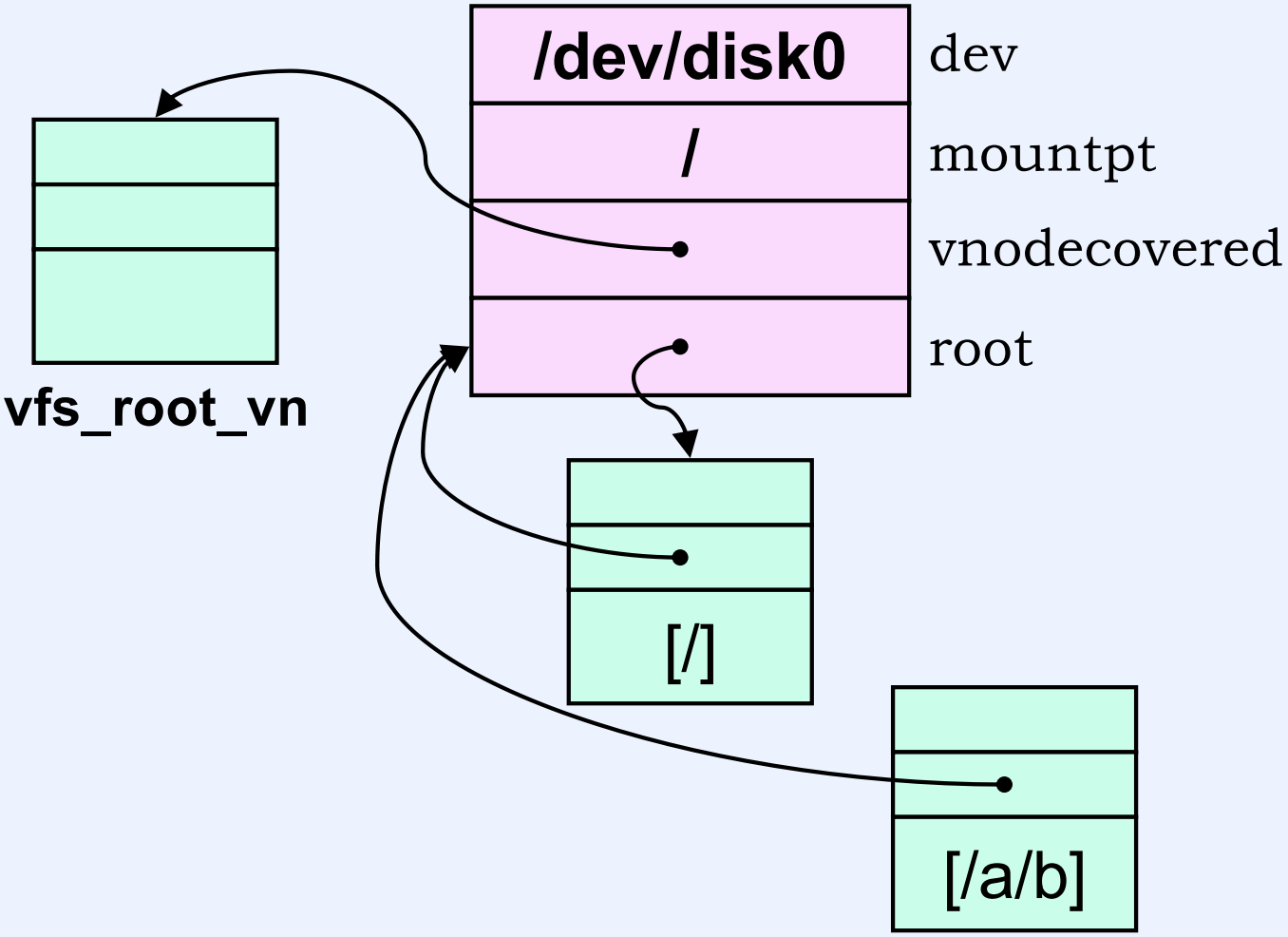
Representing File Systems

```
class fs {
    char dev[STR_MAX];          // device containing the f.s.
    char mountpt[STR_MAX];      // where the f.s. is mounted
    vnode *vnodecovered;        // file on which f.s. is mounted
    vnode *root;                // root of the f.s.
    virtual void read_vnode(vnode *);
    virtual void delete_vnode(vnode *);
};
```

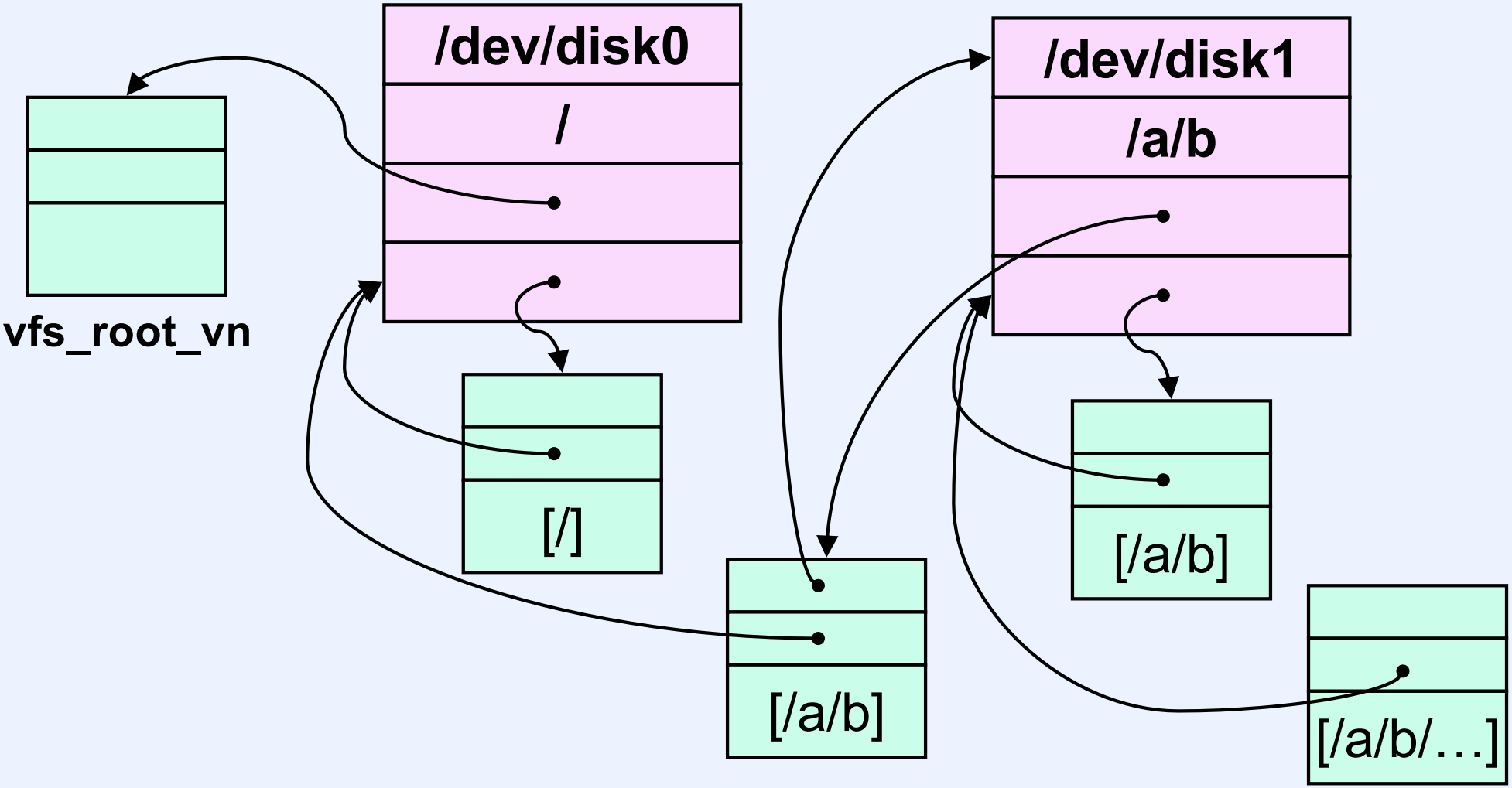
Representing Files

```
class vnode {
    unsigned short refcount;
    fs *vfsmounted;
    fs *vfs;
    unsigned long vno;
    int mode;
    int len;
    link_list_t link;
    kmutex_t mutex;
    virtual int create(const char *, int, vnode **);
    virtual int read(int, void *, int);
    virtual int write(int, const void *, int);
    …
};
```

Mounting a File System (1)



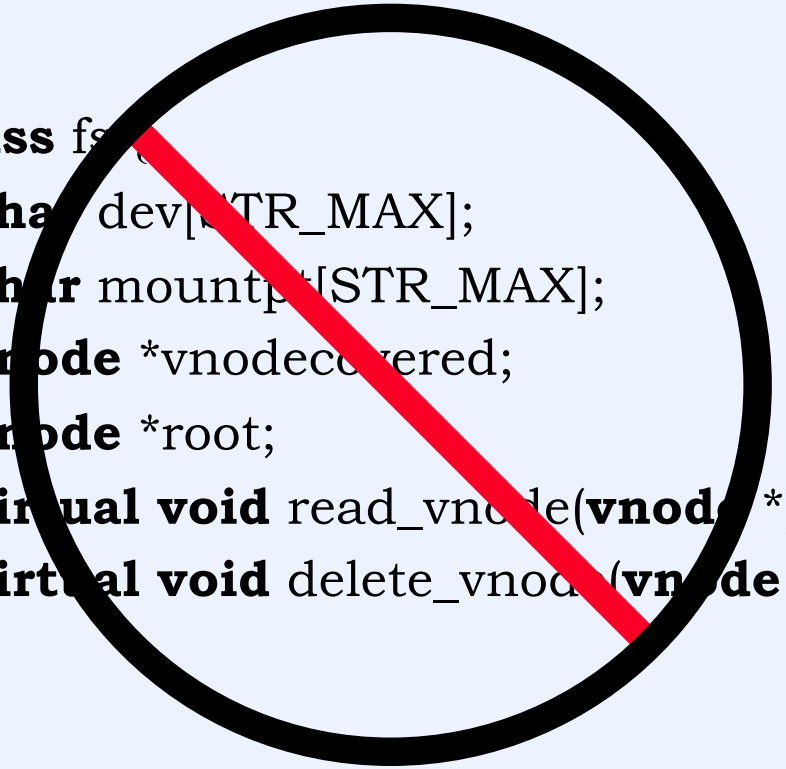
Mounting a File System (2)



But Wait …

- What's this about C++?
  - real operating systems are written in C …

fs



```
class fs {
  char dev[STR_MAX];
  char mountpt[STR_MAX];
  vnode *vnodecovered;
  vnode *root;
  virtual void read_vnode(vnode *);
  virtual void delete_vnode(vnode *);
};
```

```
typedef struct fs {
  char fs_dev[STR_MAX];
  char fs_mountpt[STR_MAX];
  struct vnode *fs_vnodecovered;
  struct vnode *fs_root;
  fs_ops_t *fs_op;
     /* function pointers */
  void *fs_i;
     /* extra stuff in subclasses */
} fs_t;
```

vnode

```
class vnode {
   unsigned short refcount;
   fs *vfsmounted;
   fs *vfs;
   unsigned long vno;
   int mode;
   int len;
   link_list_t link;
   kmutex_t mutex;
   virtual int create(const char *,
    int, vnode **);
   virtual int read(int, void *, int);
   virtual int write(int, const void *,
    int);
   …
};
```

```
typedef struct vnode {
   unsigned short vn_refcount;
   struct fs *vn_vfsmounted;
   struct fs *vn_vfs;
   unsigned long vn_vno;
   int vn_mode;
   int vn_len;
   link_list_t vn_link;
   kmutex_t vn_mutex;
   struct vnode_ops *vn_op;
      /* function pointers */
   void *vn_i;
      /* extra stuff in subclasses */
   }
```