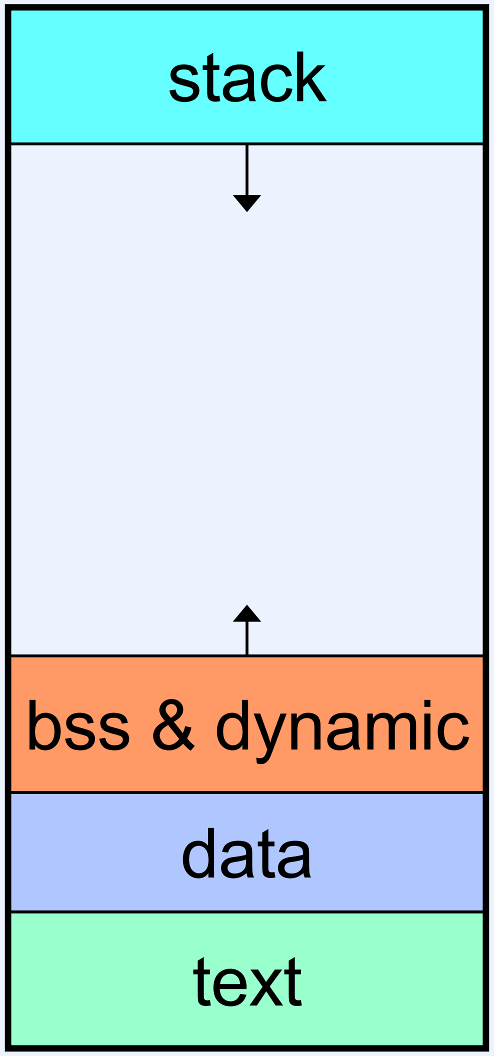
Unix Structure (2)

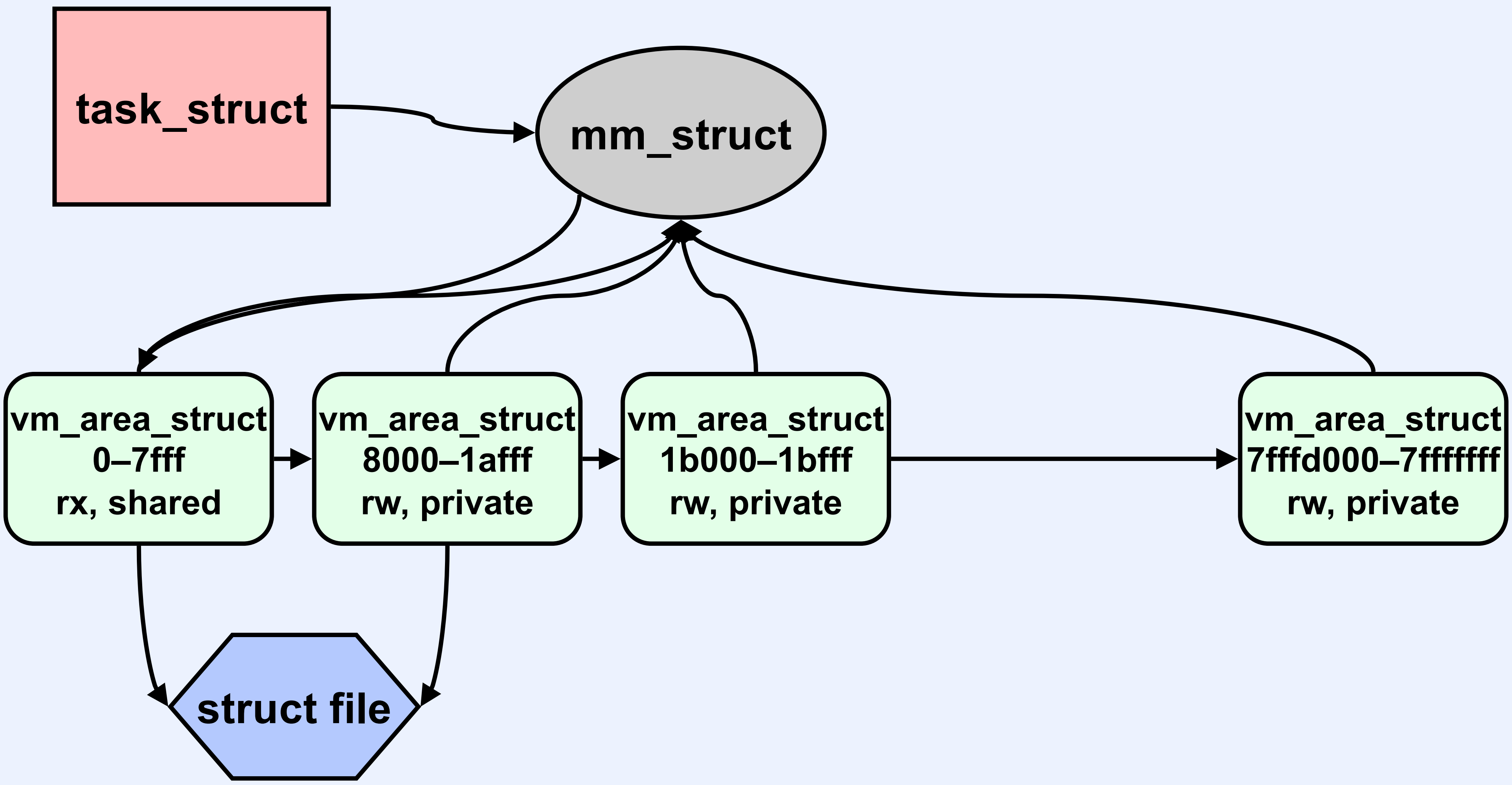
Representing the Address Space

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

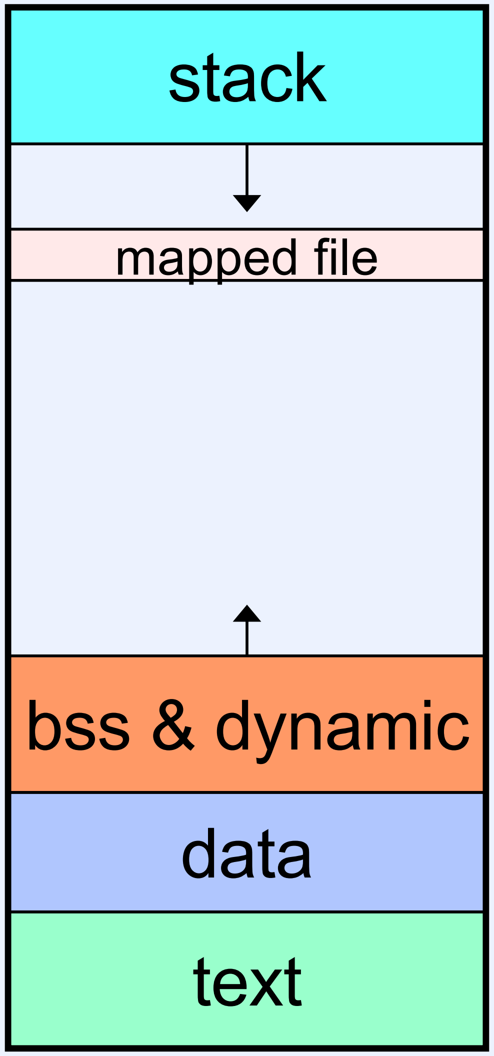
Simple User Address Space



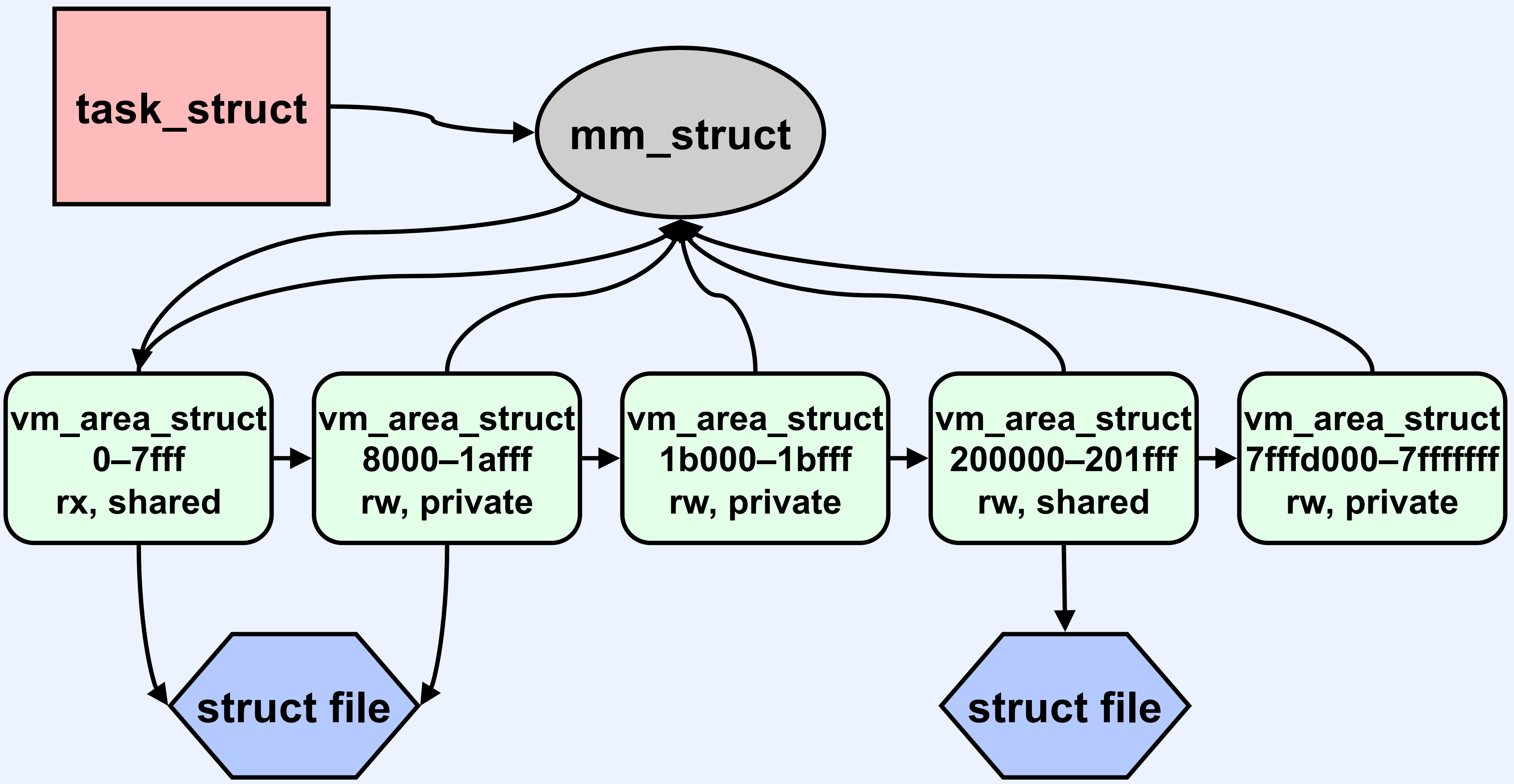
Address-Space Representation
Somewhat Simplified



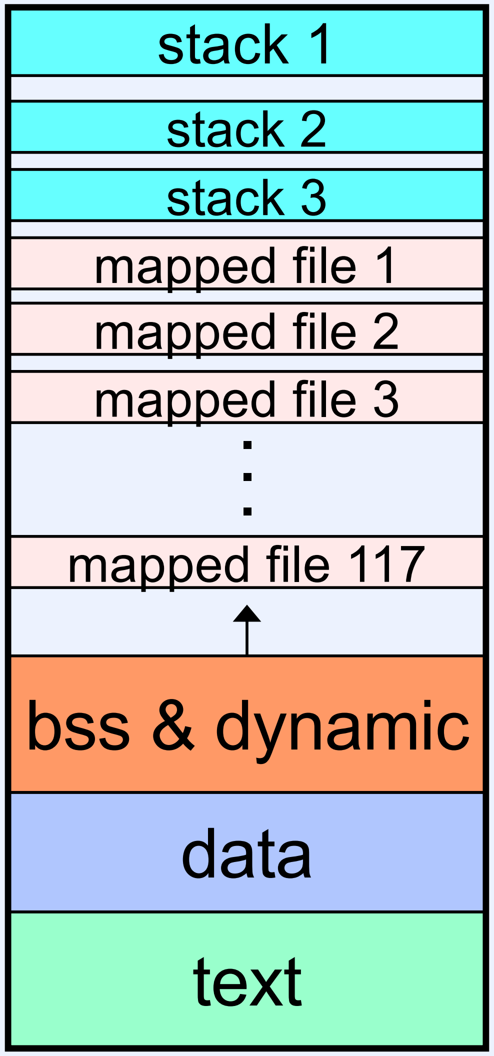
Adding a Mapped File



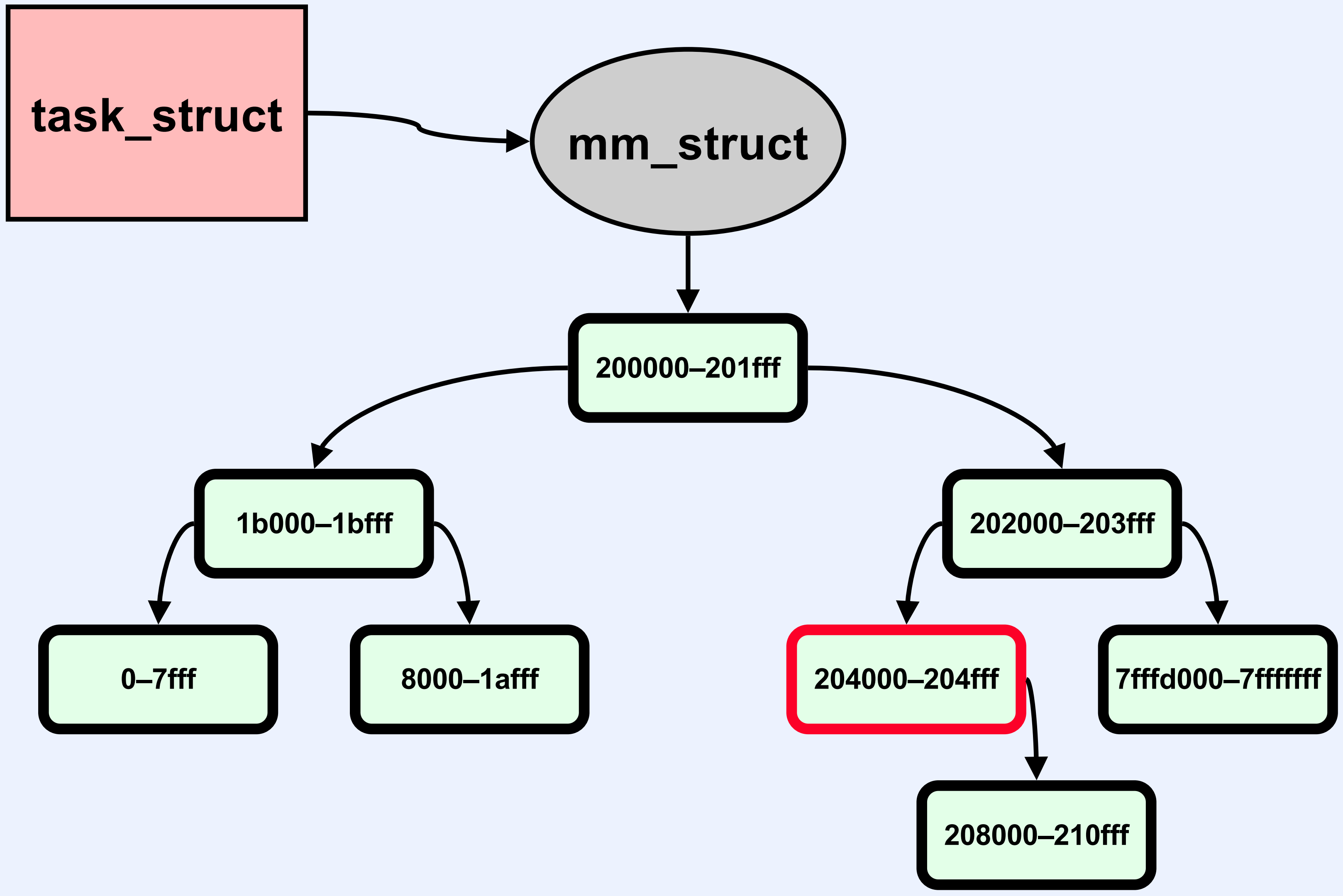
Address-Space Representation:
More Areas



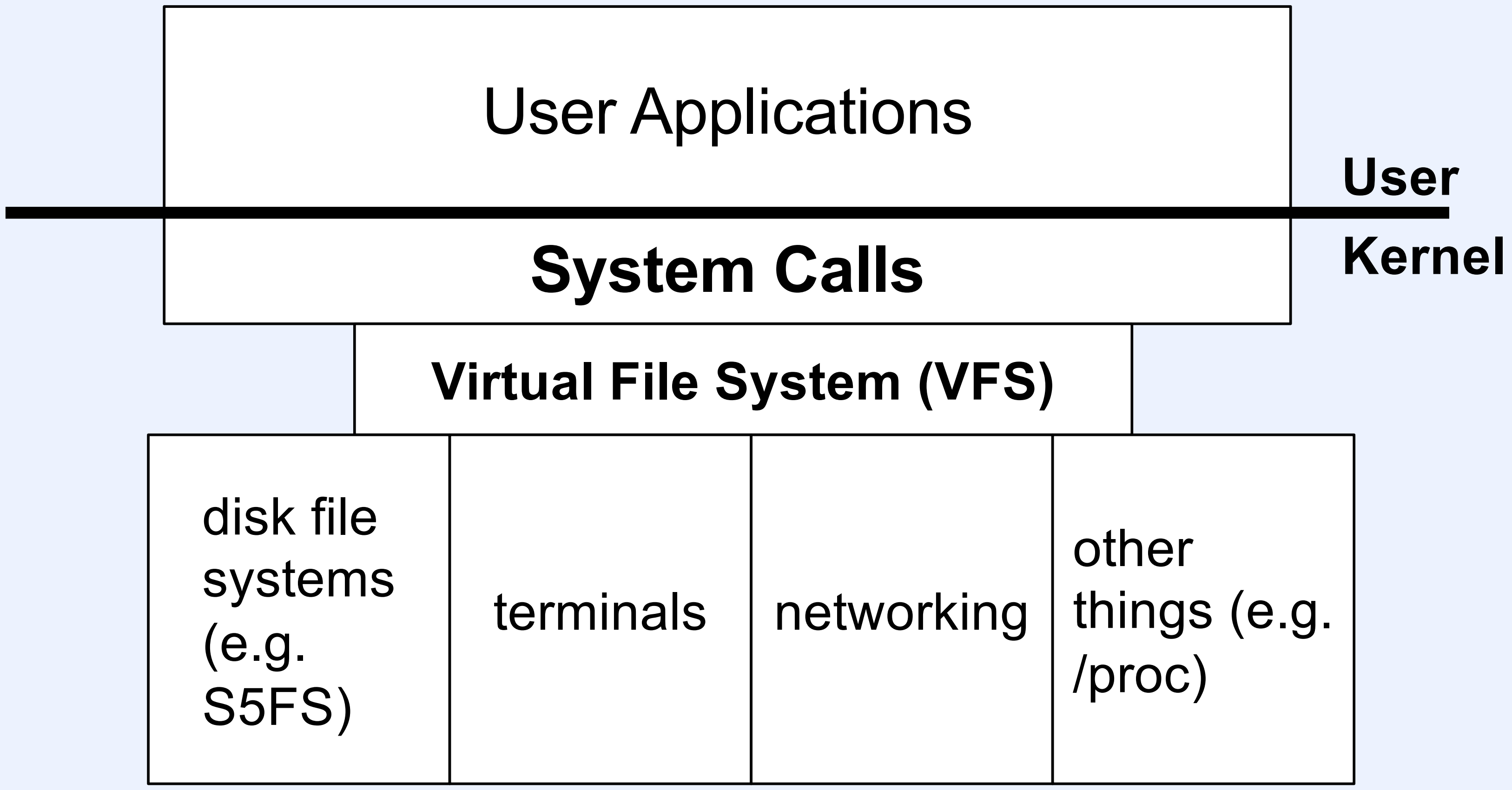
Adding More Stuff



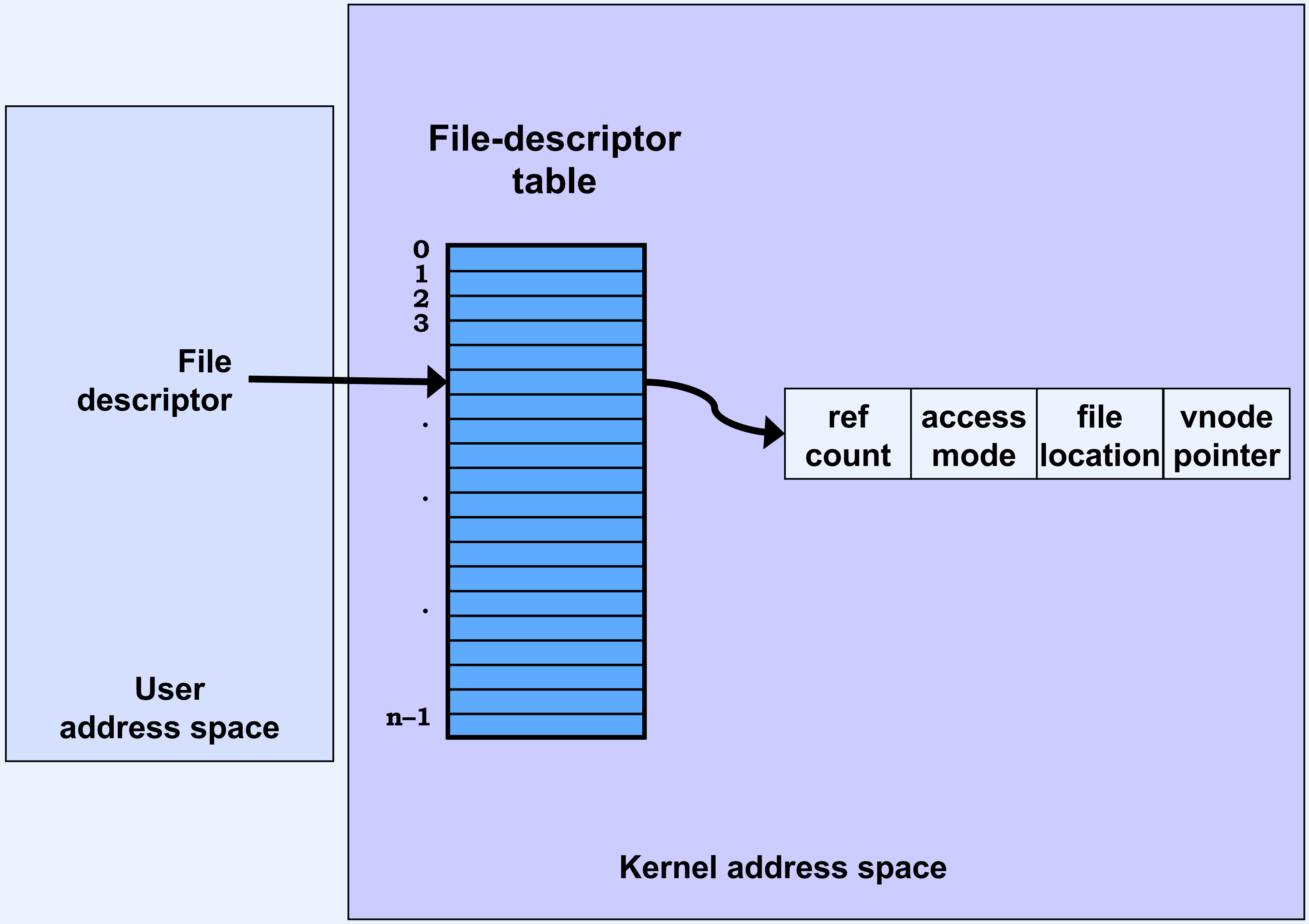
Address-Space Representation:
Reality



Layering


User Applications
User
Kernel
System Calls
Virtual File System (VFS)
disk file systems (e.g. S5FS)
terminals
networking
other things (e.g. /proc)

File-Descriptor Table



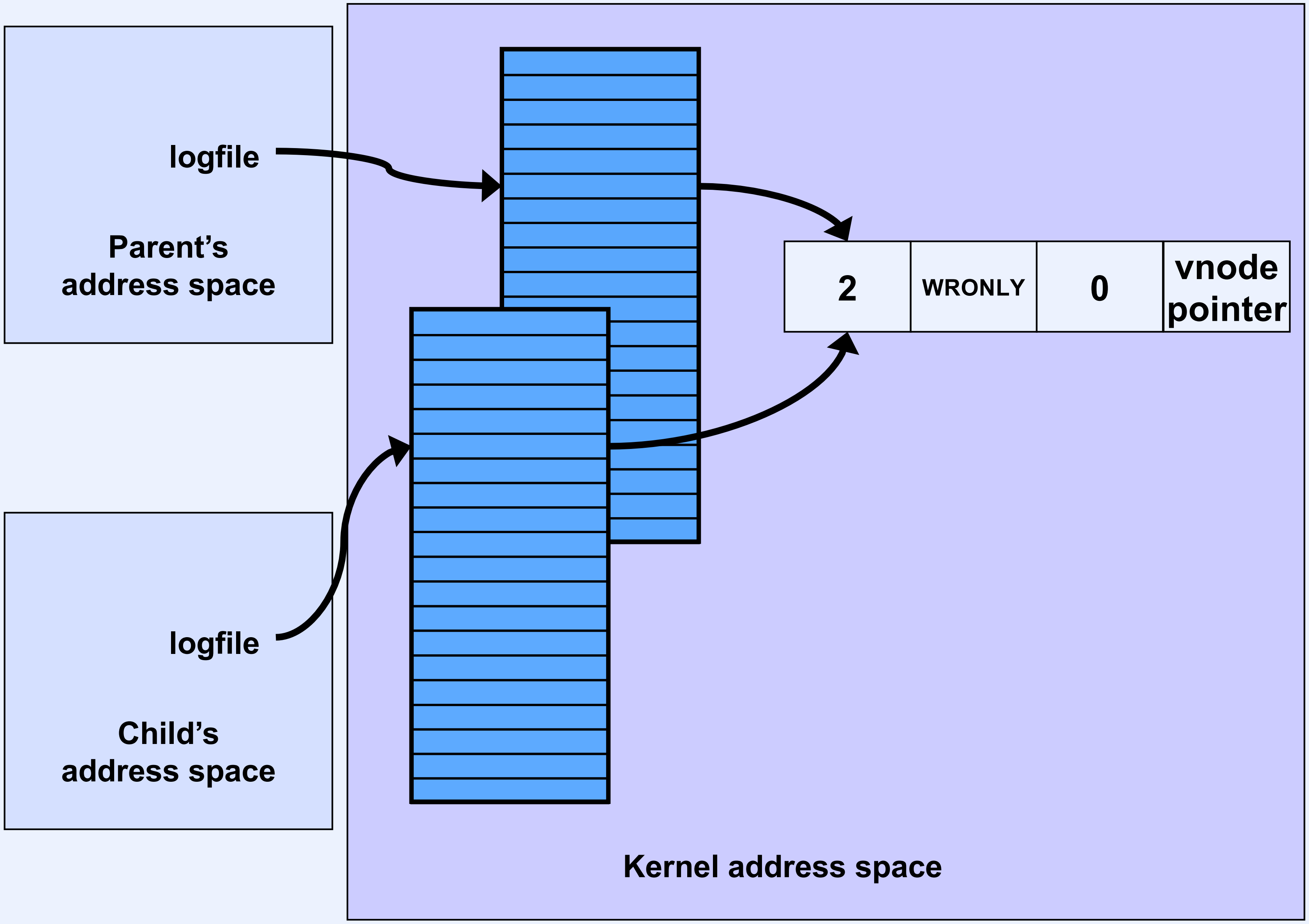
Fork and File Descriptors

```
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



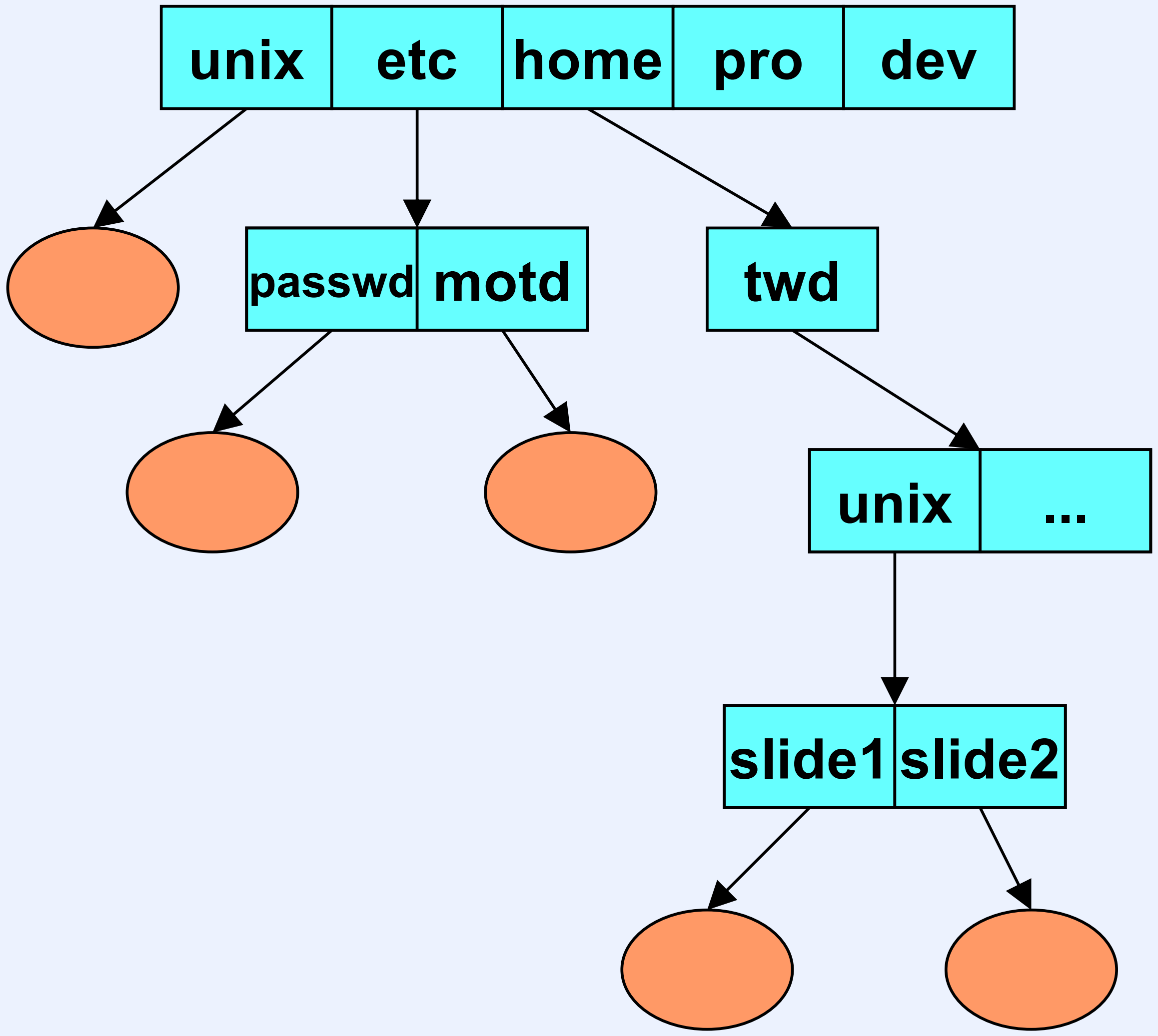
Quiz 1

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

Directories



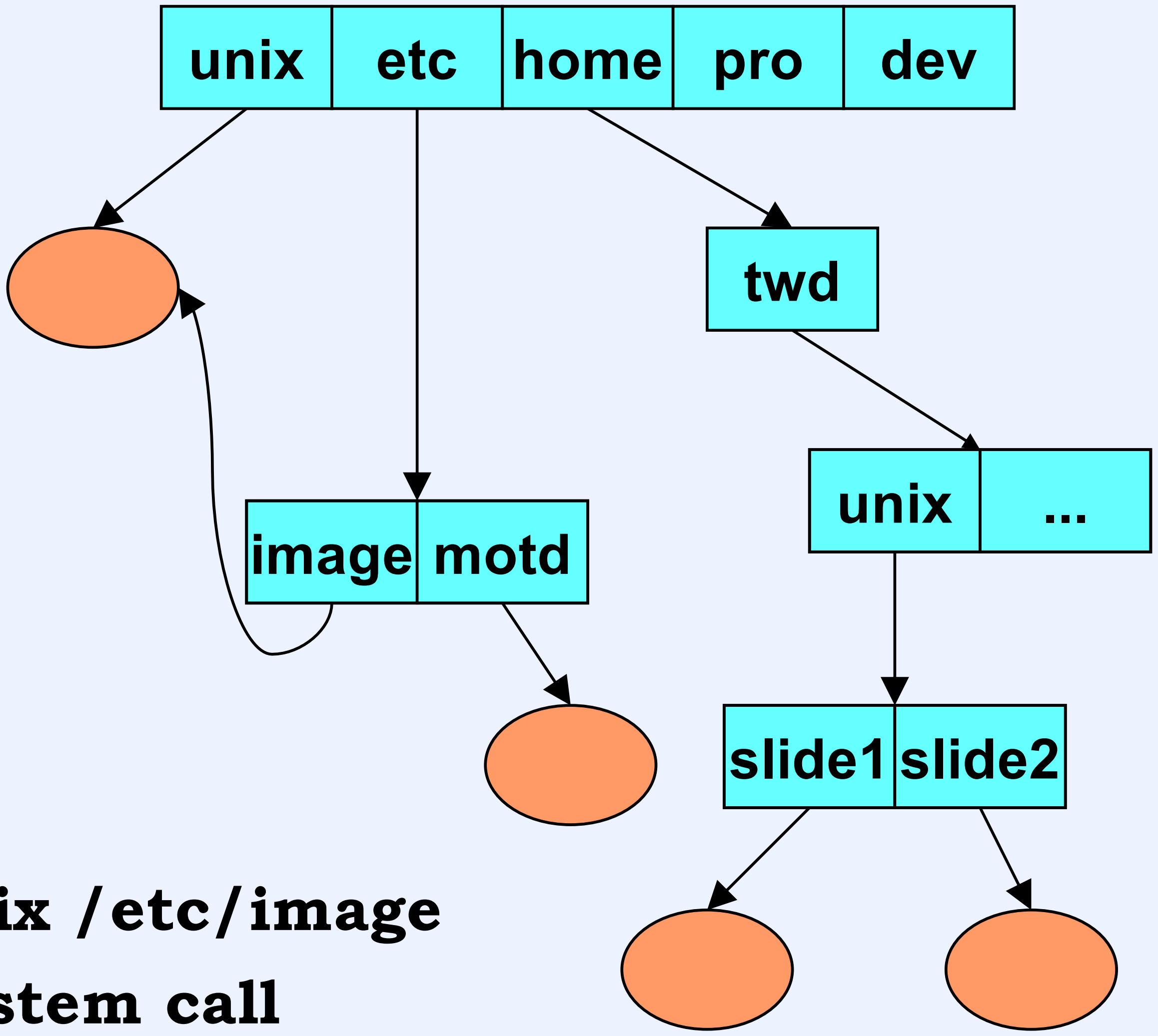
Directory Representation

Component Name	Inode Number

directory entry

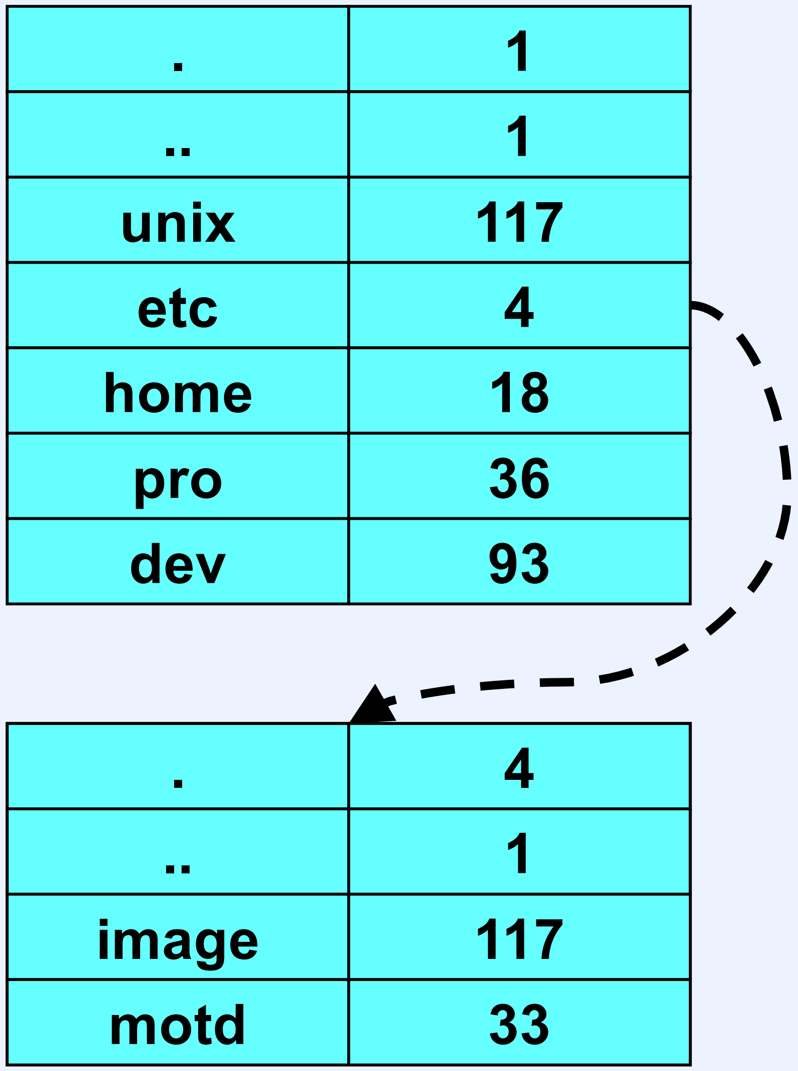
.	1
..	1
unix	117
etc	4
home	18
pro	36
dev	93

Hard Links



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

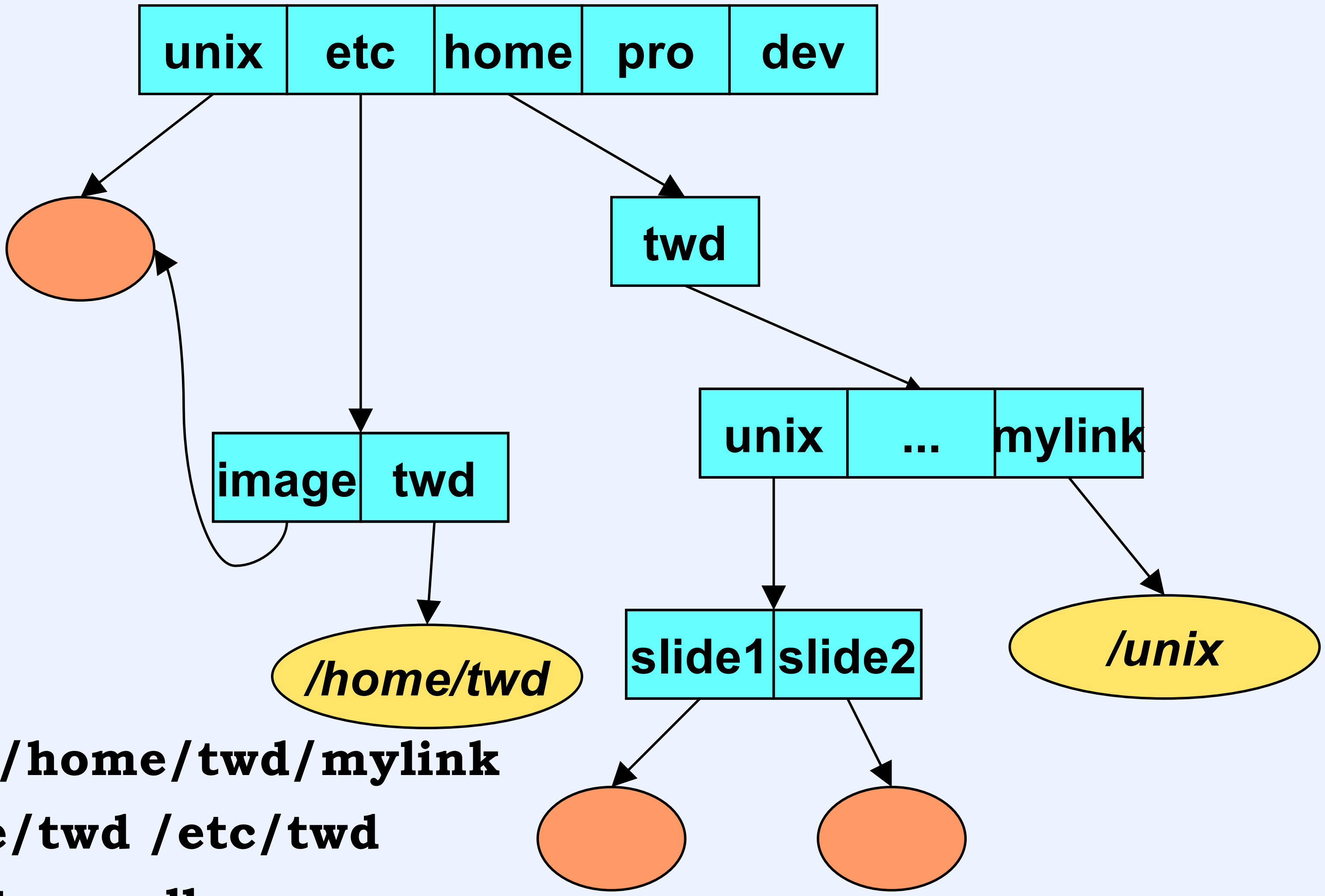
Directory Representation



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

.	4
..	1
image	117
motd	33

Soft Links



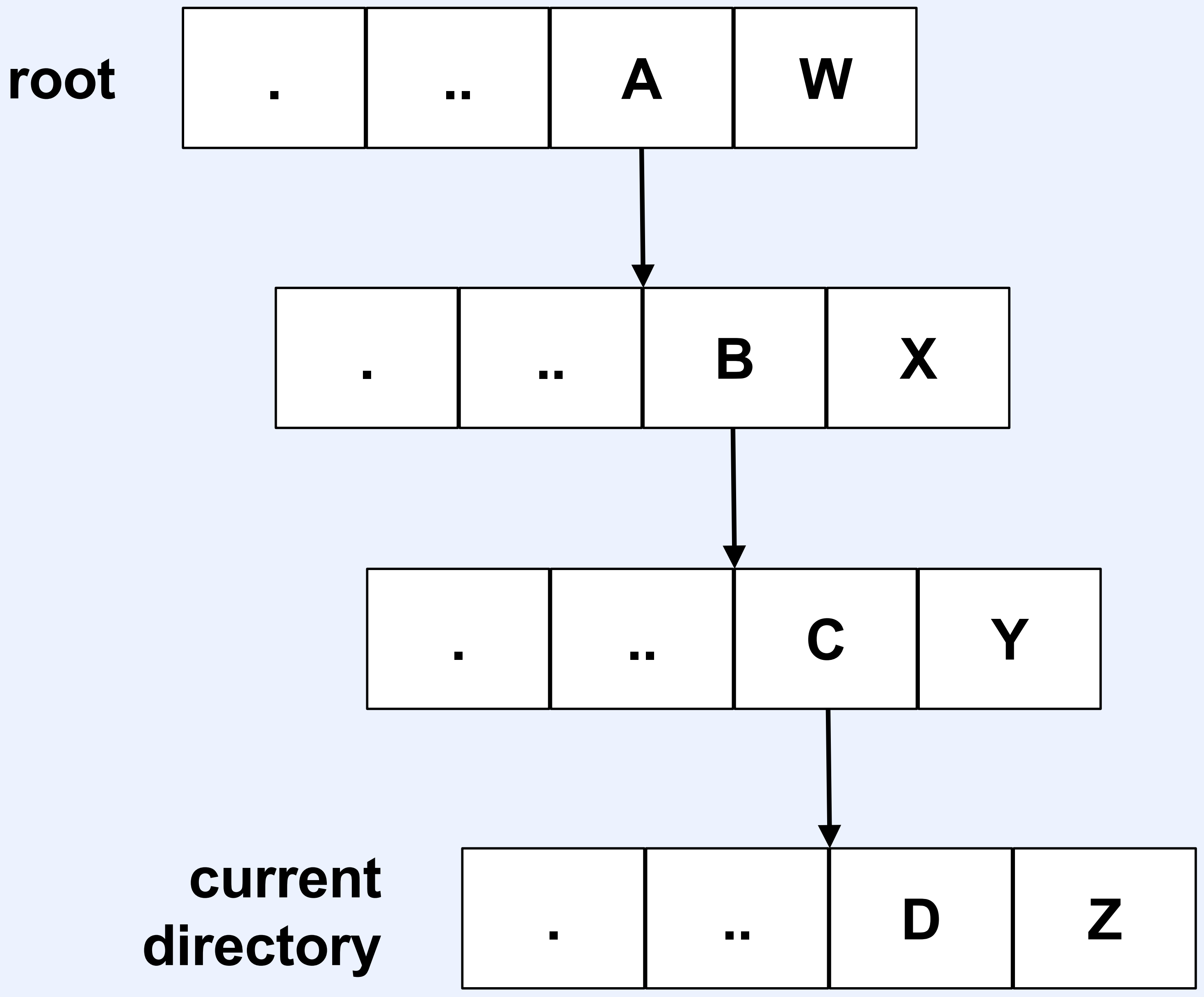
```
% ln –s /unix /home/twd/mylink
% ln –s /home/twd /etc/twd
# symlink system call
```

Working Directory

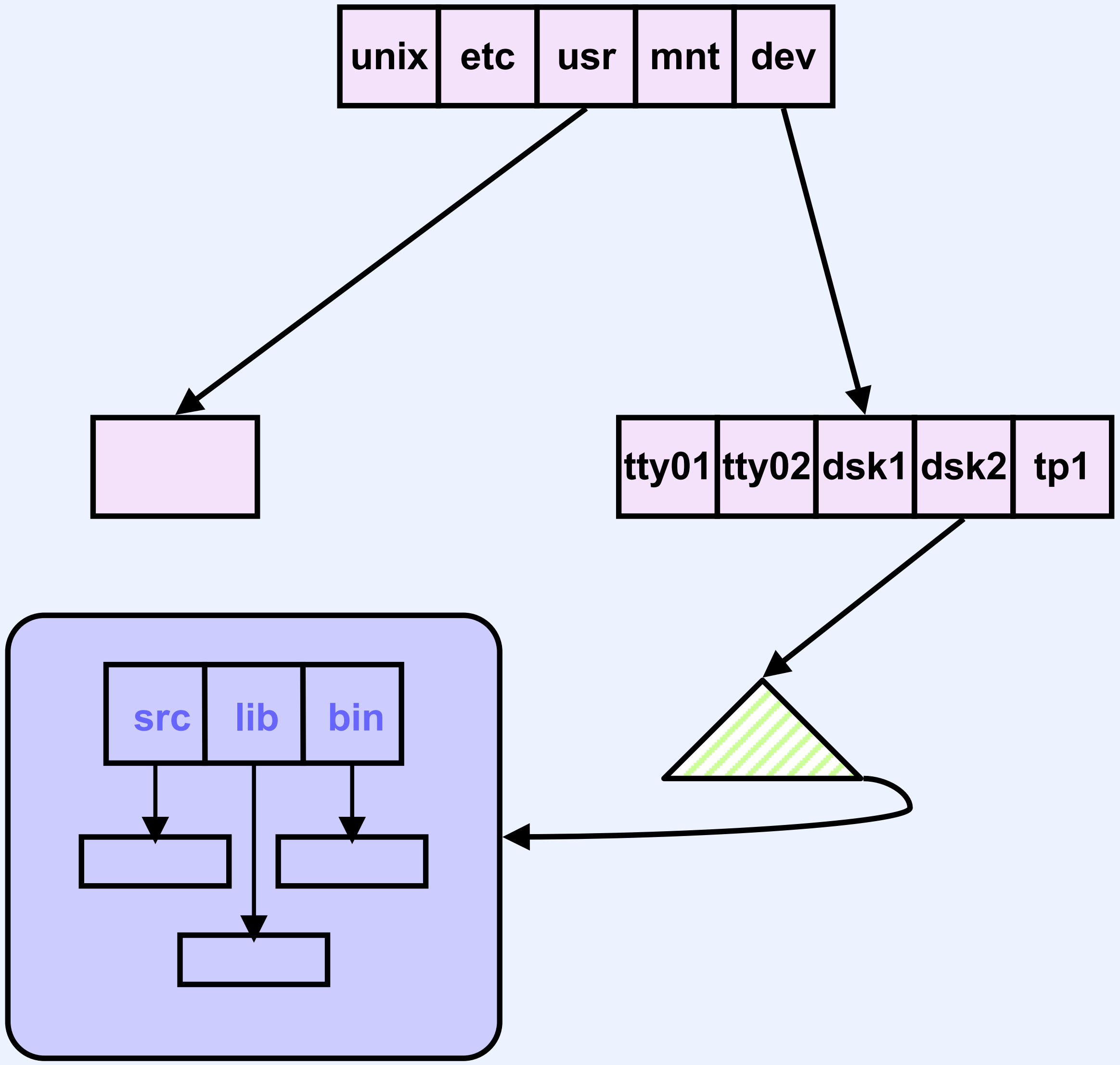
- Maintained in kernel for each process
  - paths not starting with “/” start with the working directory
  - changed by use of the *chdir* system call
  - displayed (via shell) using “pwd”
    - how is this done?

pwd

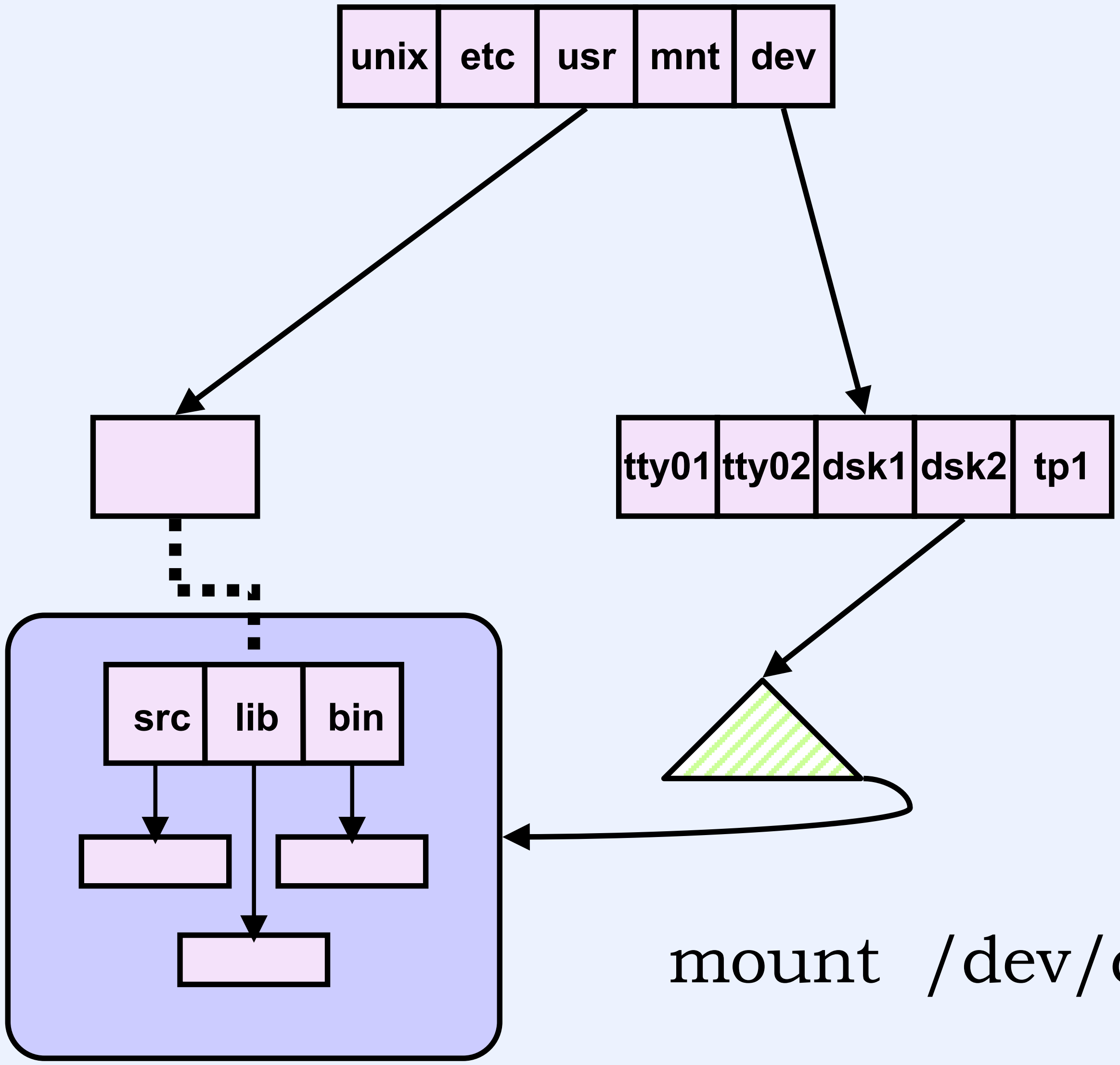
- Print Working Directory
  - suppose the current working directory is /A/B/C
  - how can a program determine it?


root
.
..
A
W
.
..
B
X
.
..
C
Y
current directory
.
..
D
Z

Mount Points (1)



Mount Points (2)



```
mount /dev/dsk2 /usr
```

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);
    …
    class mobj mobj;
};
```

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 *);
};
```

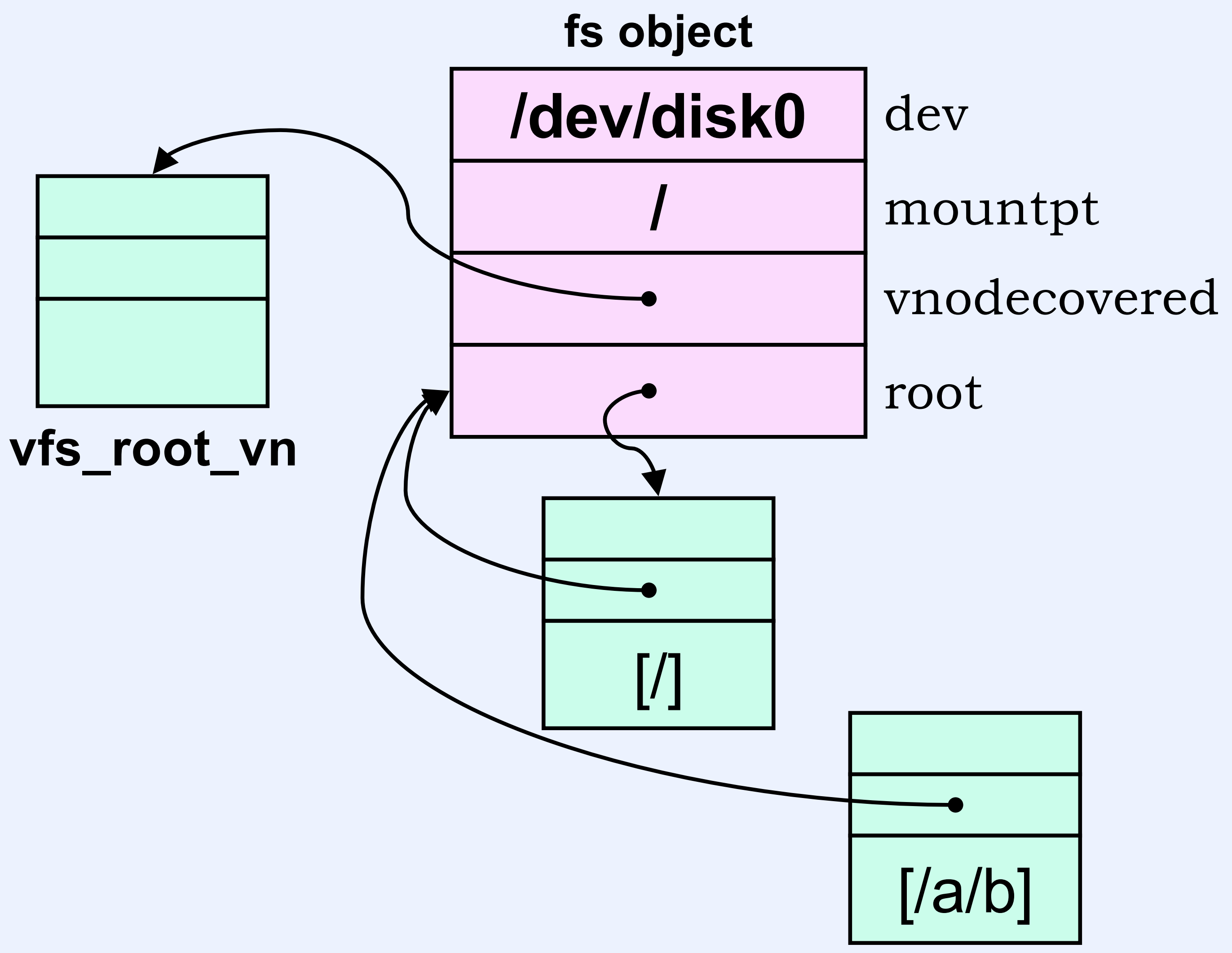
```
typedef struct fs {
  char fs_dev[STR_MAX];
  char fs_mountpt[STR_MAX];
  struct vnode *fs_vnodecovered;
  struct vnode *fs_root;
  fs_ops_t *fs_op;
      /* function pointers */
  void *fs_i;
      /* extra stuff in subclasses */
} fs_t;
```

vnode

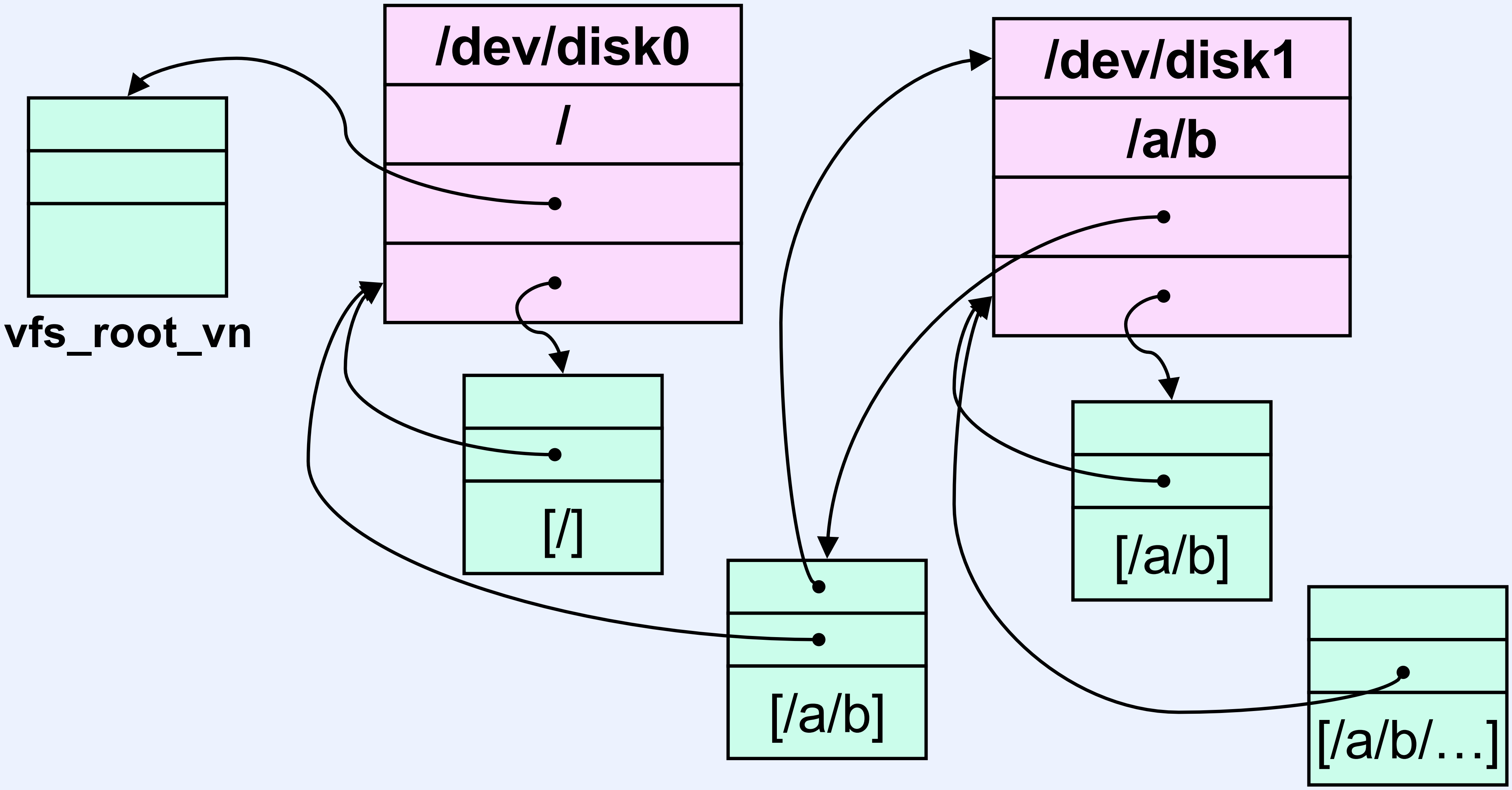
```
class vnode {
   unsigned short refcount;
   fs *vfsmounted;
   fs *vfs;
   unsigned long vno;
   int mode;
   int len;
   link_list_t link;
   kmutex_t mutex;
   virtual int create(const char *, int,
    vnode **);
   virtual int read(int, void *, int);
   virtual int write(int, const void *,
 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

```
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 */
   struct mobj mobj;
   void *vn_i;
      /* extra stuff in subclasses */
} vnode_t;
```

Mounting a File System (1)



Mounting a File System (2)



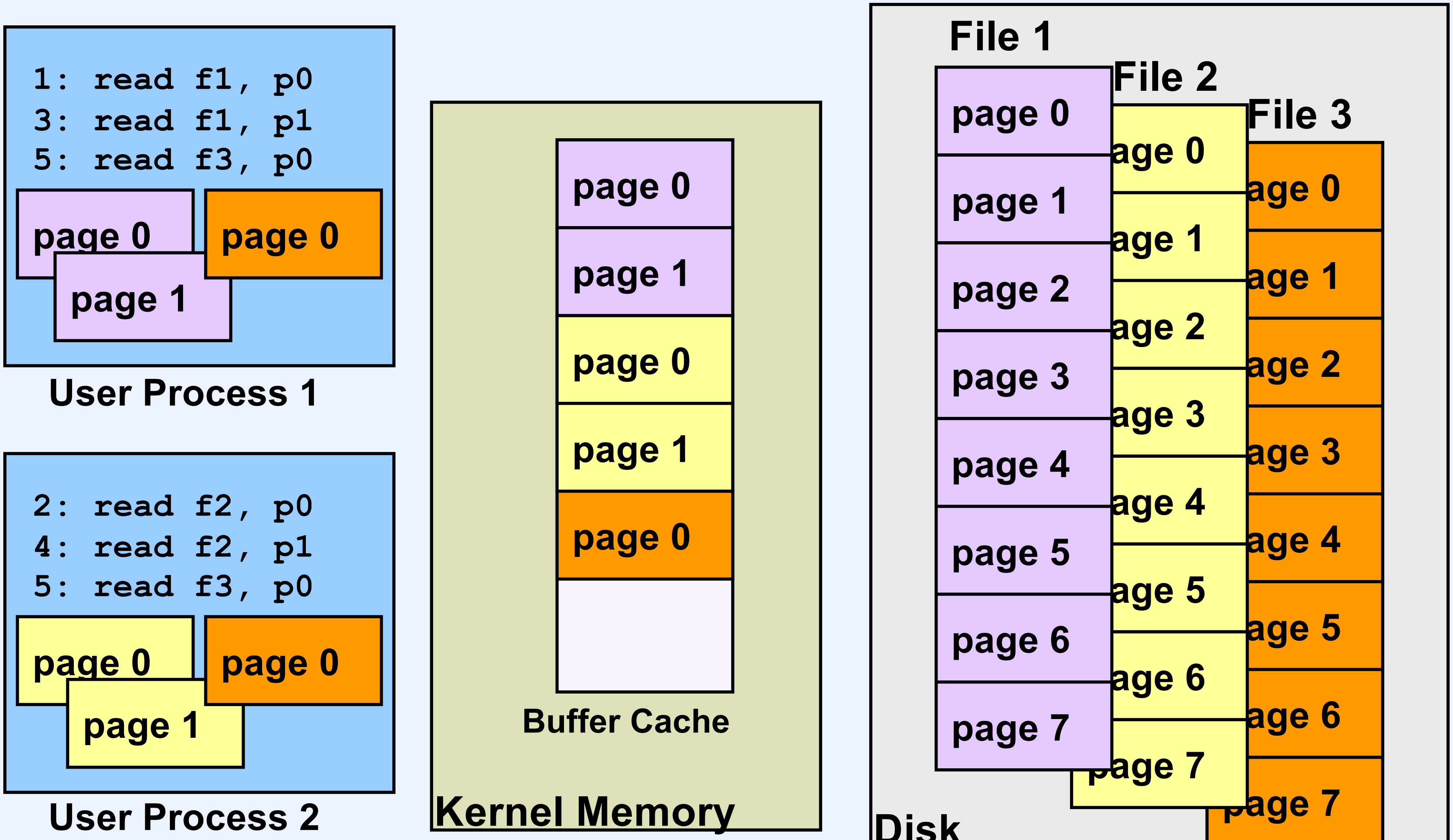
Quiz 2

- Suppose the current working directory is /A/B/M. /A/B is a path in file system 1. /M is a directory in file system 2. File system 2 is mounted on file system 1 at /A/B. How does the pwd command handle this case?
  a) it's easy: the ".." entry in /A/B/M refers to the directory /A/B and one simply continues backwards towards the root as if there were no mount point
  b) a special system call is required to determine the path
  c) one must start at the root and continue forwards towards the working directory

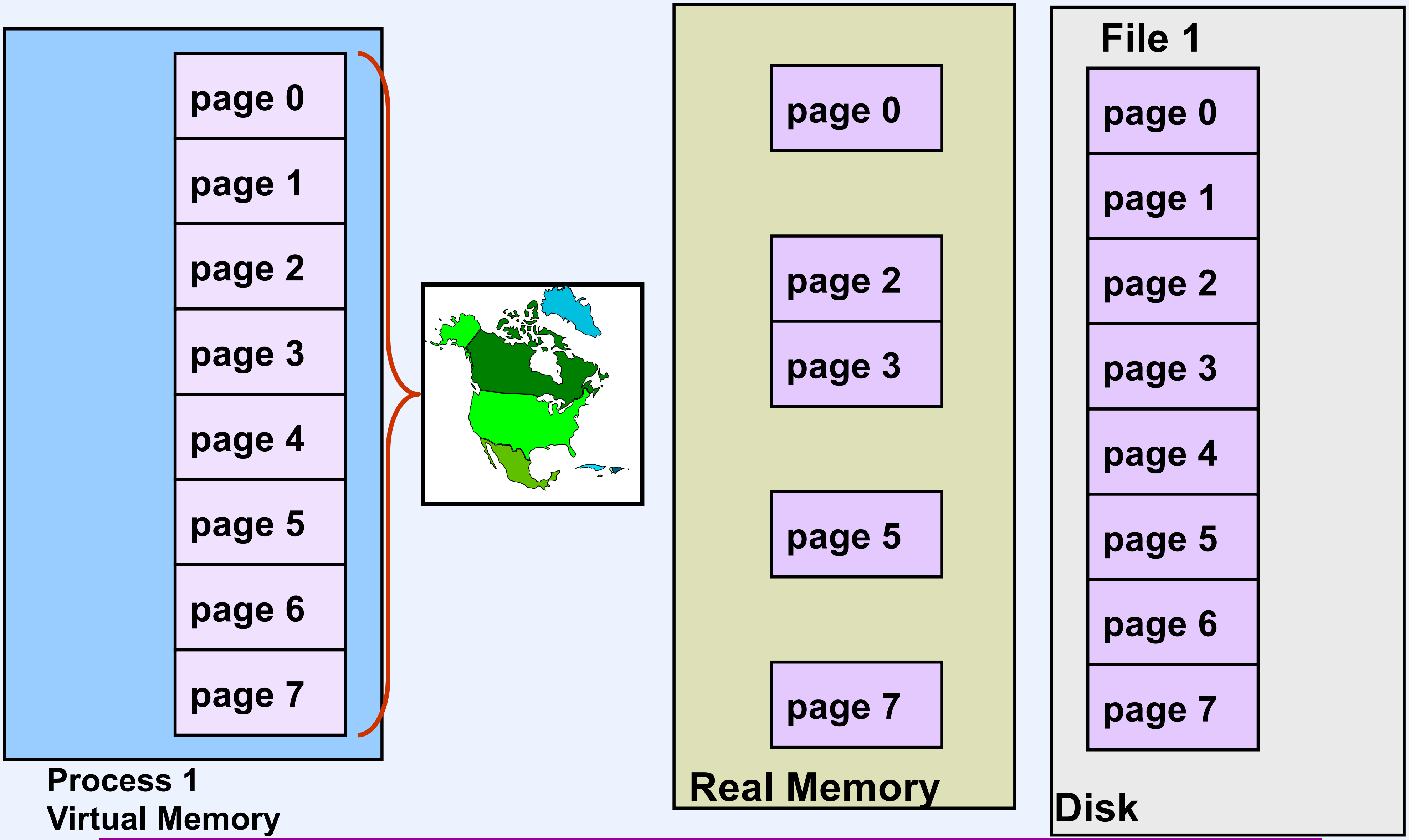
Digression (Sort of)

- Weenix has two file systems
  - ramfs
    - trivial in-memory file system for testing purposes
    - caching of blocks isn’t necessary
  - s5fs
    - non-trivial file system that you implement
    - caching of blocks is important
    - caching uses virtual-memory subsystem so that *mmap* works right

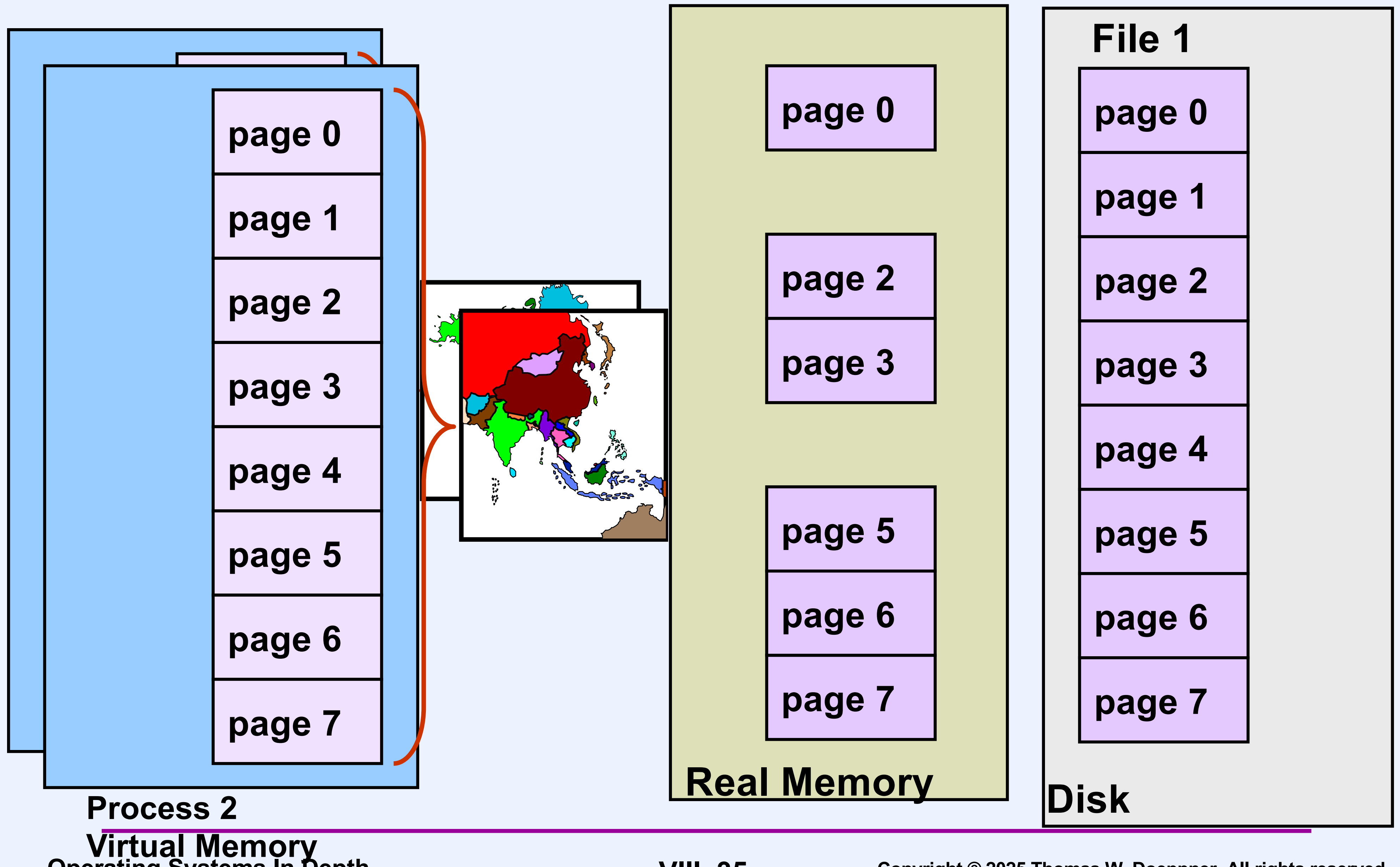
Traditional I/O



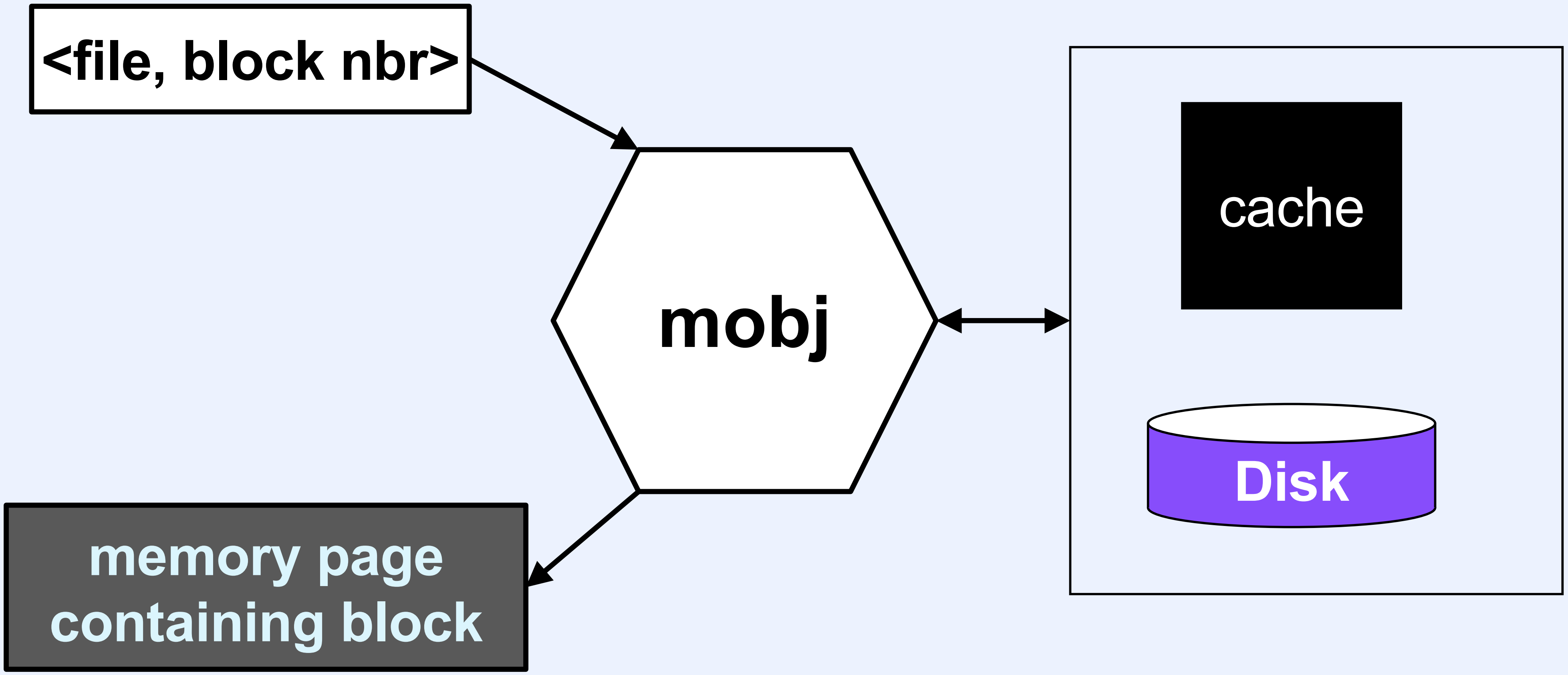
Mapped File I/O



Multi-Process Mapped File I/O



Memory Objects


<file, block nbr>
mobj
cache
Disk
memory page containing block