Definition (**file system**)

- A **file** is a logical unit of (stored) data.
- A **file system** is an abstraction mechanism: it allows logical manipulation of files, without knowledge of their physical representation.

Definition (**file system**)

A **file system** provides a mapping

$$\text{identifier} \mapsto (\text{meta-data}, \text{data}),$$

plus a mechanism to manage (concurrent) manipulation of both

$$\begin{aligned} \text{data} &\simeq \text{content} \\ \text{meta-data} &\simeq \text{structure} \end{aligned}$$

Definition (**file system**)

A **file system** provides a mapping

$$\text{identifier} \mapsto (\text{meta-data}, \text{data}),$$

plus a mechanism to manage (concurrent) manipulation of both

$$\begin{aligned} \text{data} &\simeq \text{content} \\ \text{meta-data} &\simeq \text{structure} \end{aligned}$$

- Question: why be so abstract?

Definition (**file system**)

A **file system** provides a mapping

$$\text{identifier} \mapsto (\text{meta-data}, \text{data}),$$

plus a mechanism to manage (concurrent) manipulation of both

$$\begin{aligned} \text{data} &\simeq \text{content} \\ \text{meta-data} &\simeq \text{structure} \end{aligned}$$

- Question: why be so abstract?
- Answer: file systems support multiple use-cases, e.g.,

1. general-purpose data storage,
2. special-purpose data storage (e.g., swap space [4]), or
3. interface with kernel

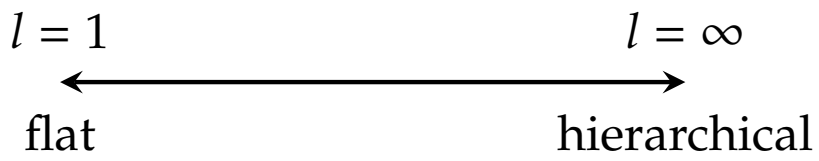
so saying "file" rather than "data" *may* be artificially limiting.

- Challenge(s): we need to consider
  - what an appropriate system call interface should be, *and*
  - how to map the semantics of said interface onto one or more concrete storage devices

  noting the former necessarily has a nod to user-friendliness.

Concept (1)

- Option: the mapping can range between

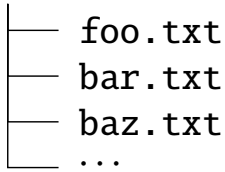


with $l > 1$ implying

- entries may be **directories**,
- the identifier specifying an entry includes a (potentially implicit) **path**,
- paths can be
  - **absolute** (from *root* directory) or
  - **relative** (from *some* directory)

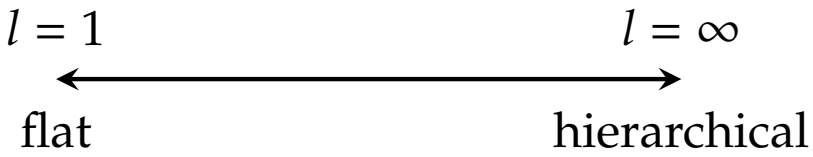
  and hence needn't be unique.

root directory



Concept (1)

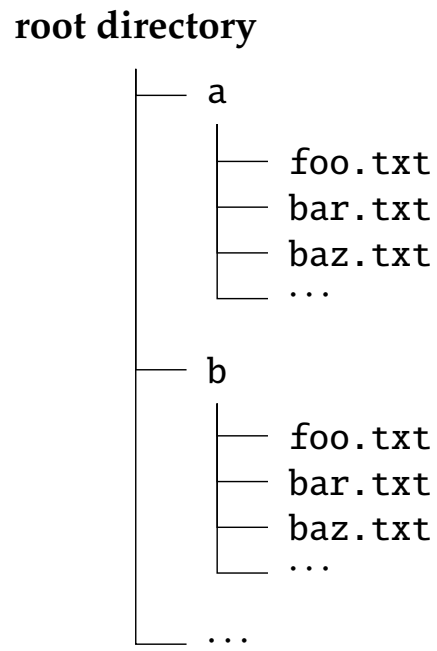
- Option: the mapping can range between



with $l > 1$ implying

- entries may be **directories**,
- the identifier specifying an entry includes a (potentially implicit) **path**,
- paths can be
  - **absolute** (from *root* directory) or
  - **relative** (from *some* directory)

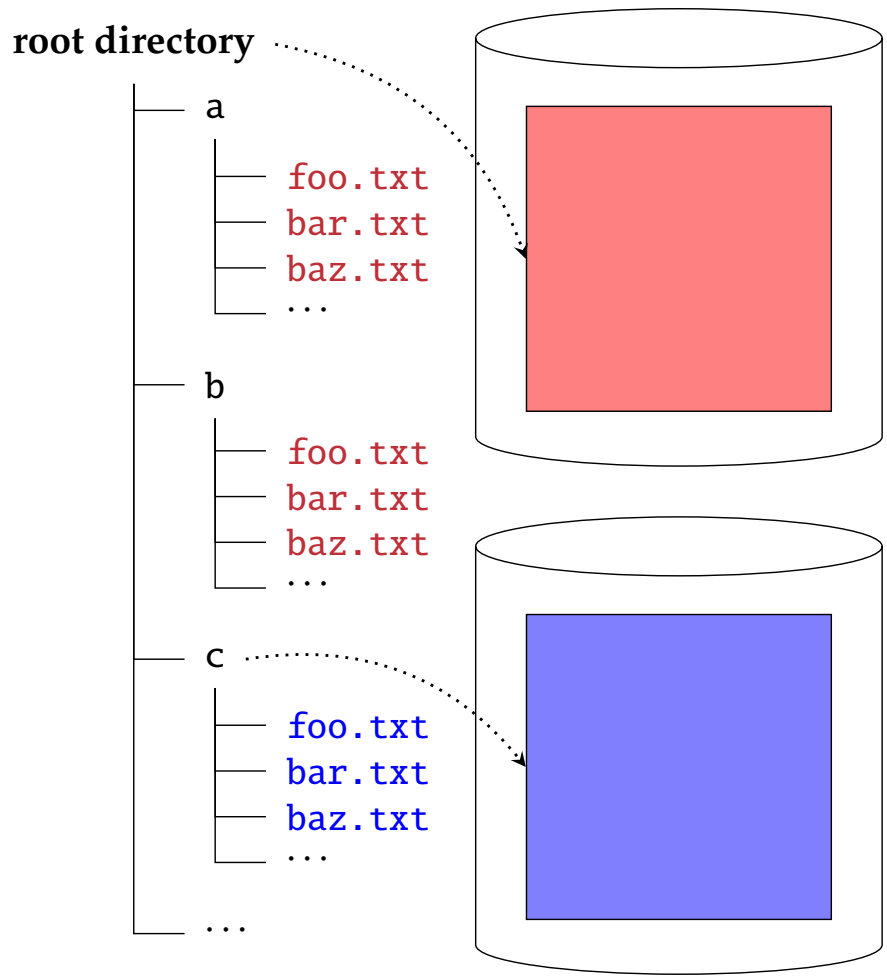
and hence needn't be unique.



Concept (1)

- Option: the root can be
  1. a **volume** identifier, or
  2. a directory

  where
  - the former implies multiple, segregated hierarchies,
  - the latter implies one, unified hierarchy ...
  - ... we **mount** a volume at a **mount point**.

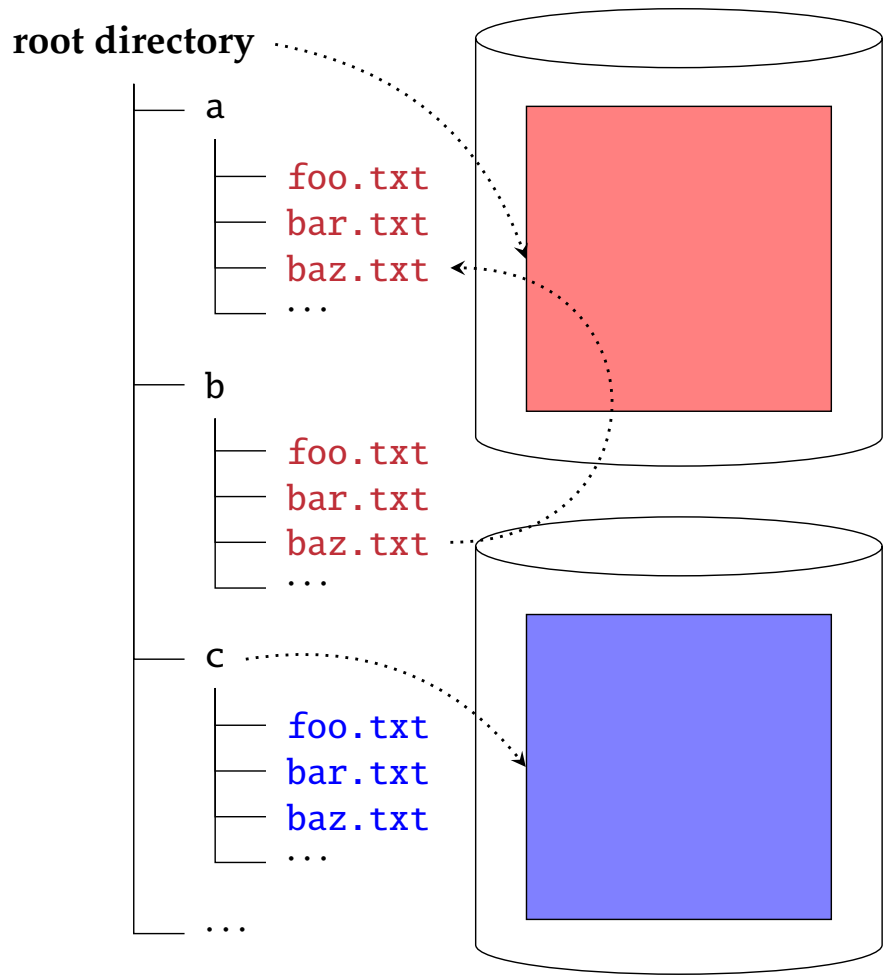


Concept (1)

- Option: entries in the hierarchy include
  - a file,
  - a directory,
  - a **symbolic link**, and
  - a **device node**

  where
  - entry types may be differentiated via meta-data or embedded magic numbers,
  - links are often categorised as either
    - **hard**, or
    - **soft**

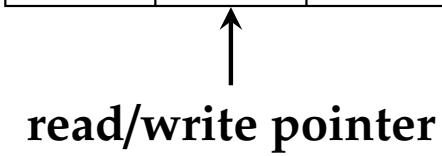
    but, either way, imply the hierarchy is now a (acyclic) *graph* (vs. a tree).



Mechanism: POSIX(ish) system call interface (1)

- Assumption: underlying file system allows us to view data as a byte sequence, i.e.,

$$\text{identifier} \mapsto (\text{meta-data}, \text{data}) = (\text{meta-data}, \boxed{d_0 \mid d_1 \mid \cdots \mid d_{l-1}})$$



that supports

1. automatic extensibility, and
2. random access (via **seek** operations).

Mechanism: POSIX(ish) system call interface (2)

- Based on this assumption, the kernel must (at least)
  1. maintain a global **mount table** that captures the hierarchy,

Mechanism: POSIX(ish) system call interface (2)

- Based on this assumption, the kernel must (at least)

  2. maintain a per process **file descriptor table** that captures
     - a **File Control Block (FCB)** of physical addressing information,
     - the mode the entry was opened in (e.g., read, write, or read/write),
     - the current read/write pointer

     that indexes into a global **file table** tracking open entries,

Mechanism: POSIX(ish) system call interface (2)

- Based on this assumption, the kernel must (at least)

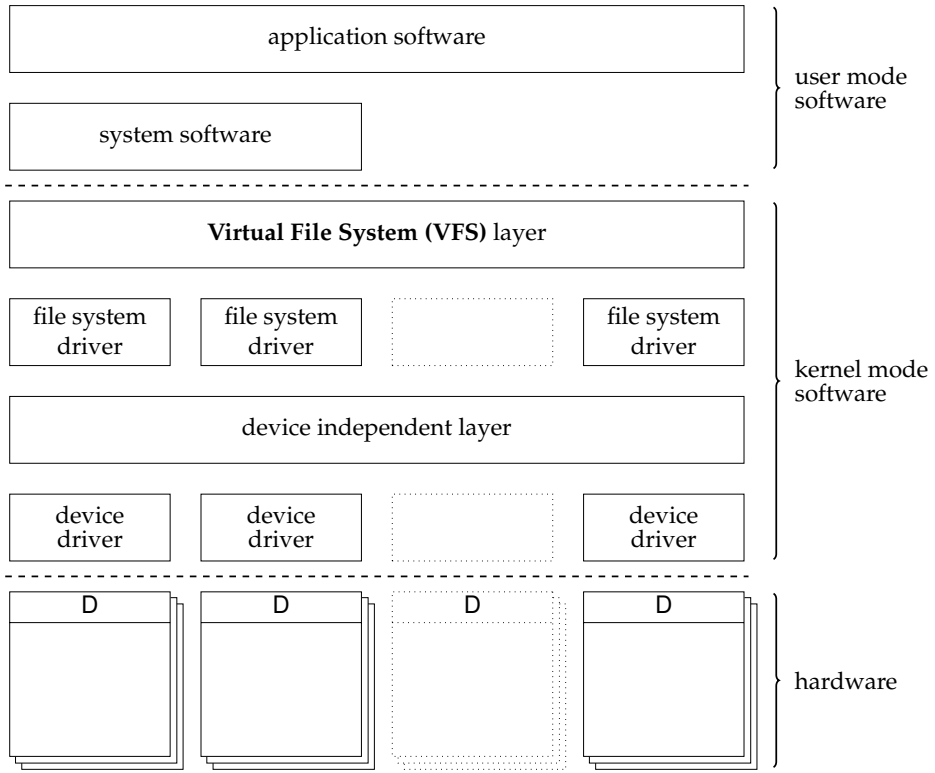
3. support a suite of system calls, e.g.,

Function	Reference	Purpose
`creat`	[3, Page 702]	create a file
`open`	[3, Page 1379]	open a file
`close`	[3, Page 676]	close a file
`unlink`	[3, Page 2154]	delete a file
`write`	[3, Page 2263]	write to a file
`read`	[3, Page 1737]	read from a file
`lseek`	[3, Page 1265]	move read/write pointer

which are ...

Mechanism: POSIX(ish) system call interface (3)

- ... (typically) exposed via a **Virtual File System (VFS)** layer



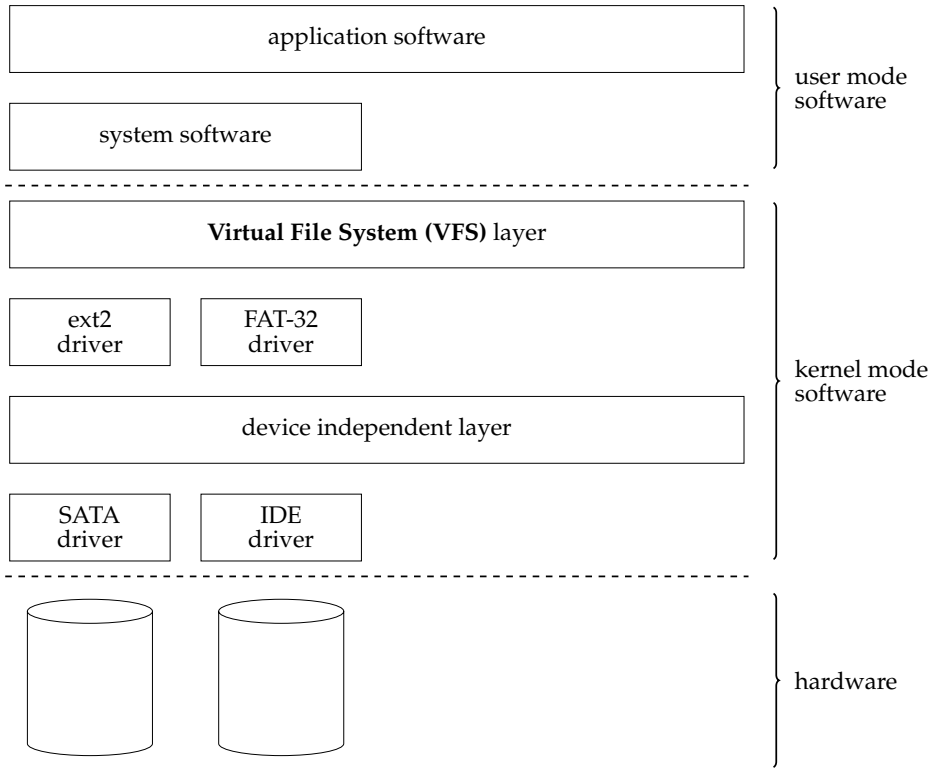
offering

1. a uniform interface to
   - multiple heterogeneous concrete file systems, and
   - "device-less" pseudo-files

   plus
2. various optimisation and translation operations.

Mechanism: POSIX(ish) system call interface (3)

- ... (typically) exposed via a **Virtual File System (VFS)** layer



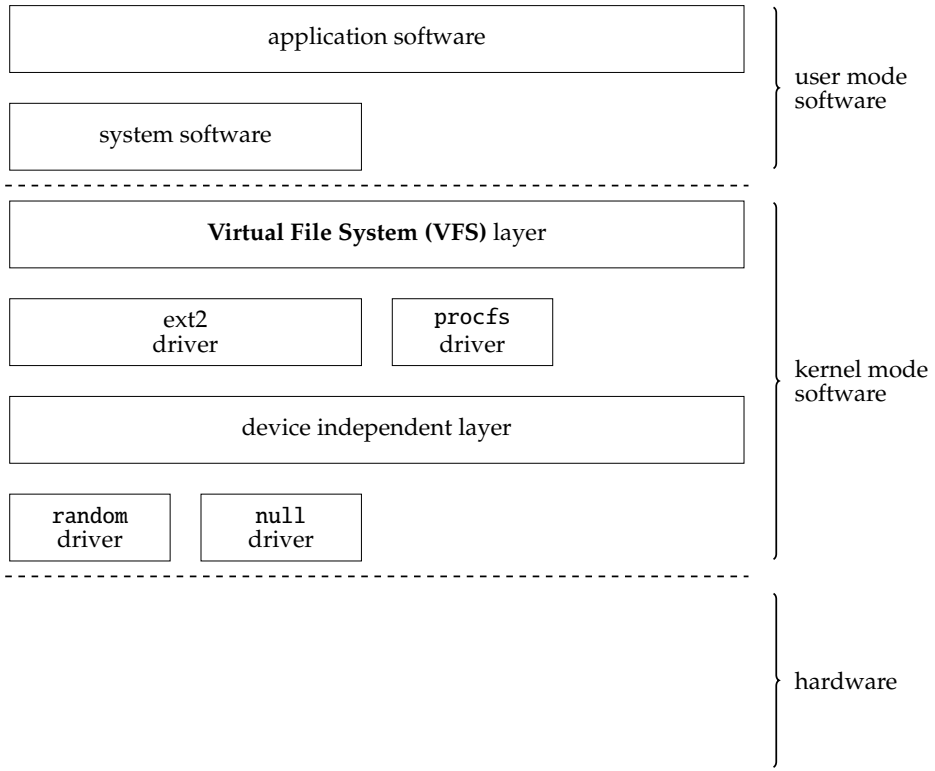
offering

1. a uniform interface to
   - multiple heterogeneous concrete file systems, and
   - "device-less" pseudo-files

   plus
2. various optimisation and translation operations.

Mechanism: POSIX(ish) system call interface (3)

- ... (typically) exposed via a **Virtual File System (VFS)** layer



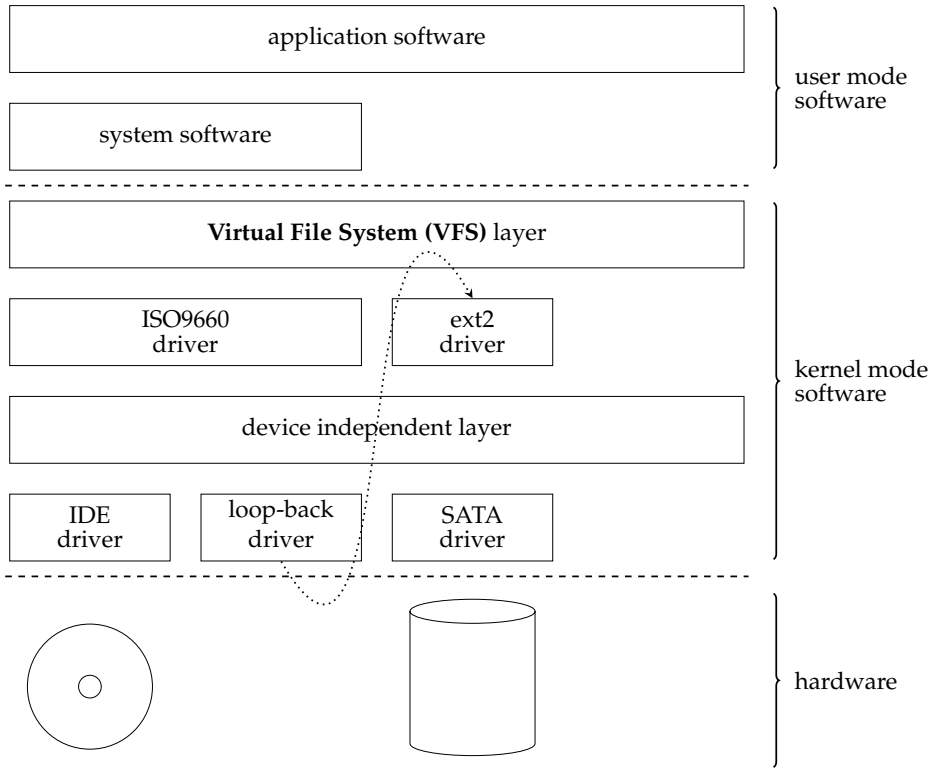
offering

1. a uniform interface to
   - multiple heterogeneous concrete file systems, and
   - "device-less" pseudo-files

   plus
2. various optimisation and translation operations.

Mechanism: POSIX(ish) system call interface (3)

- ... (typically) exposed via a **Virtual File System (VFS)** layer



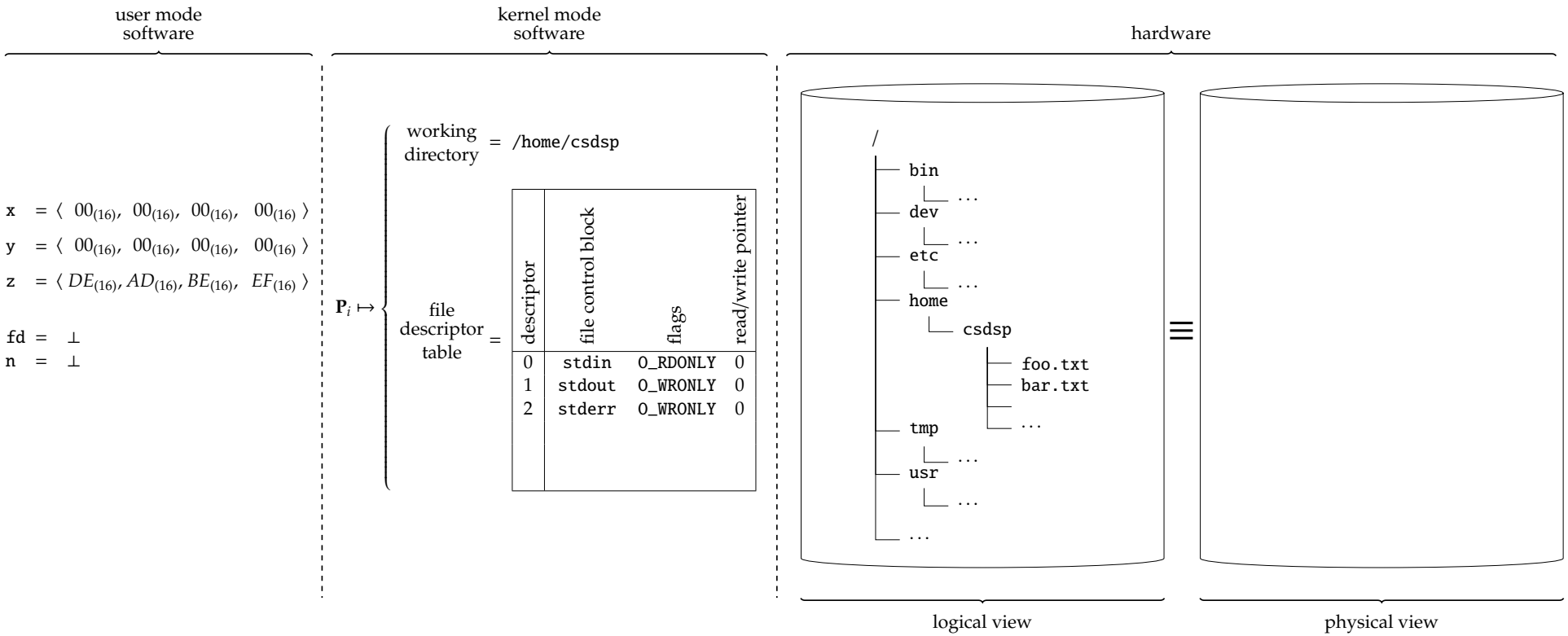
offering

1. a uniform interface to
   - multiple heterogeneous concrete file systems, and
   - "device-less" pseudo-files

   plus
2. various optimisation and translation operations.

Mechanism: POSIX(ish) system call interface (4)

- Example:

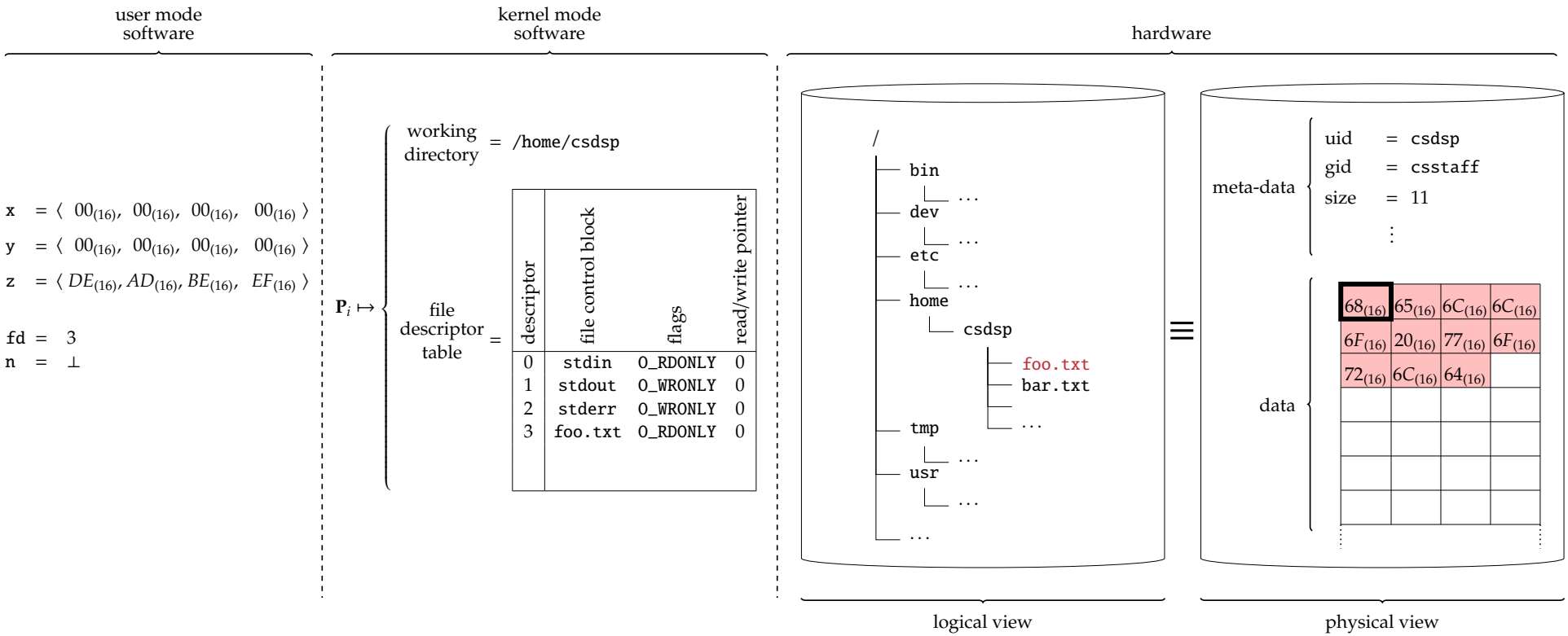


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
fd = open( "foo.txt", O_RDONLY )
```

then the result is described by

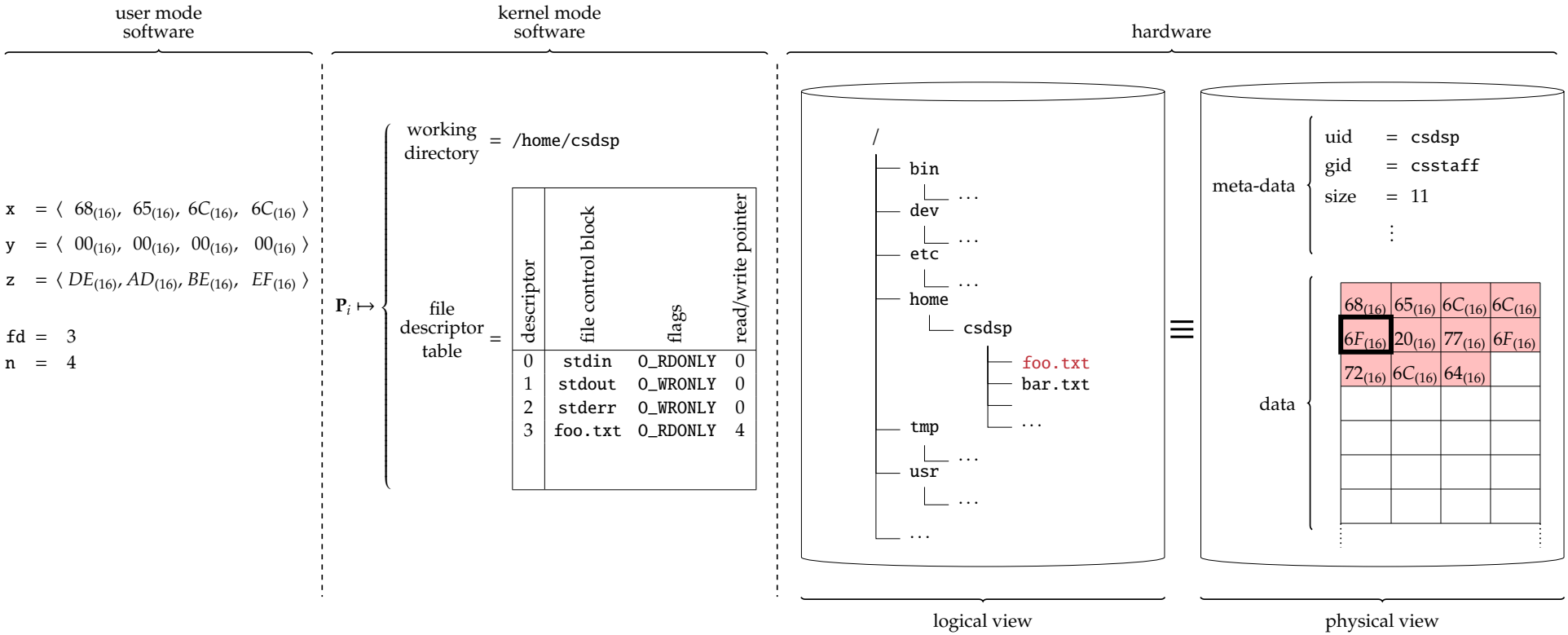


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = read( fd, x, 4 )
```

then the result is described by

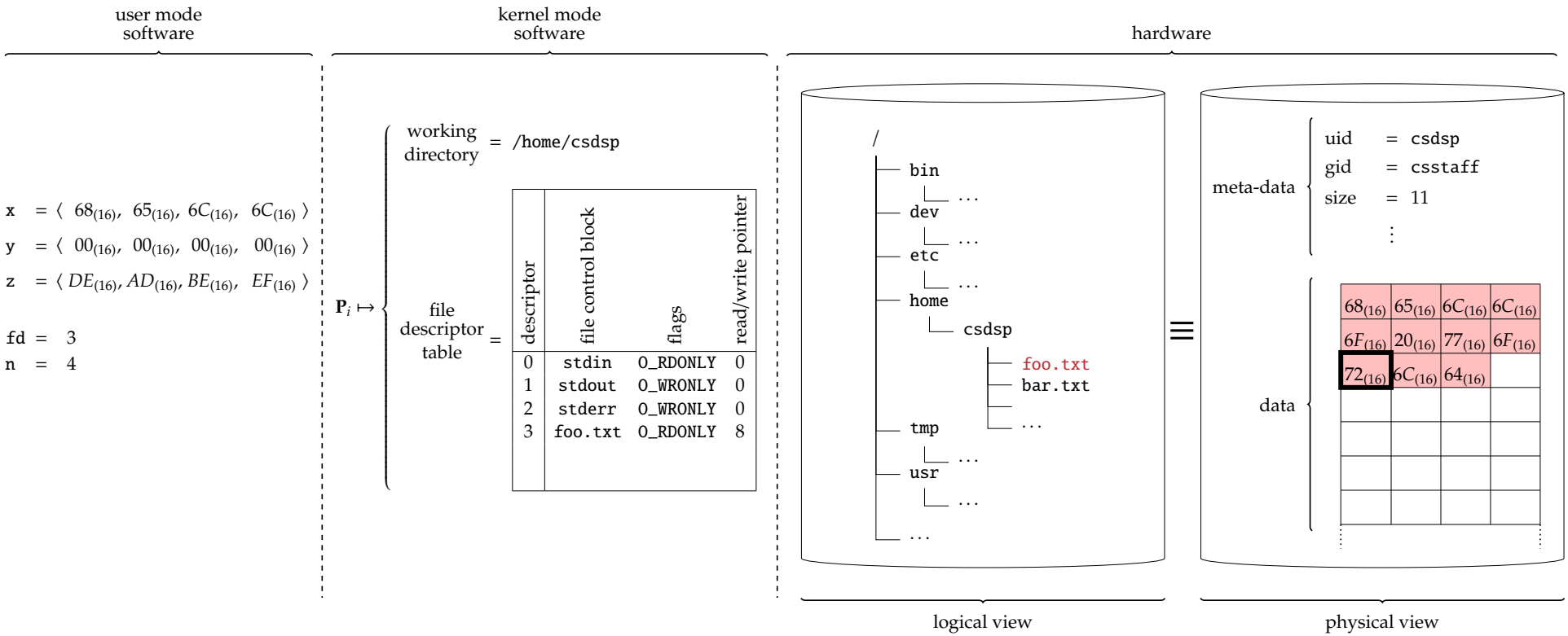


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
lseek( fd, +8, SEEK_SET )
```

then the result is described by

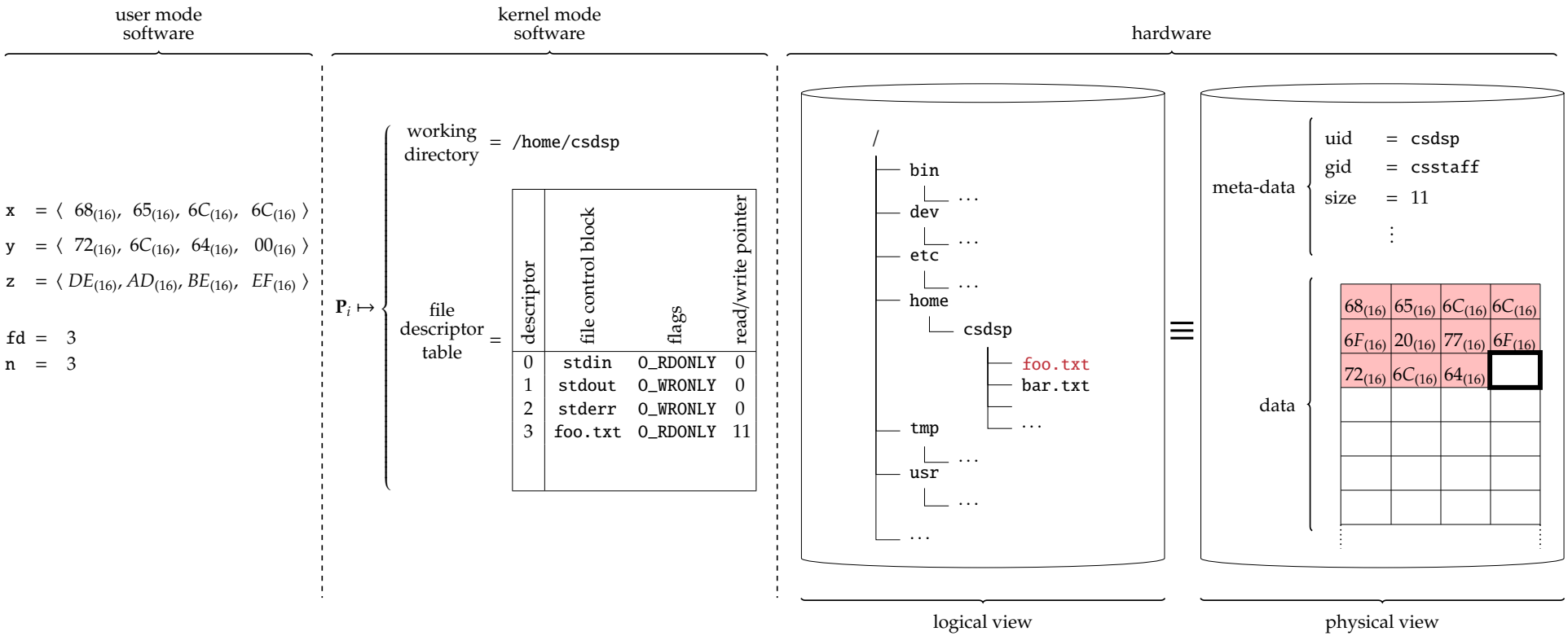


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = read( fd, y, 4 )
```

then the result is described by

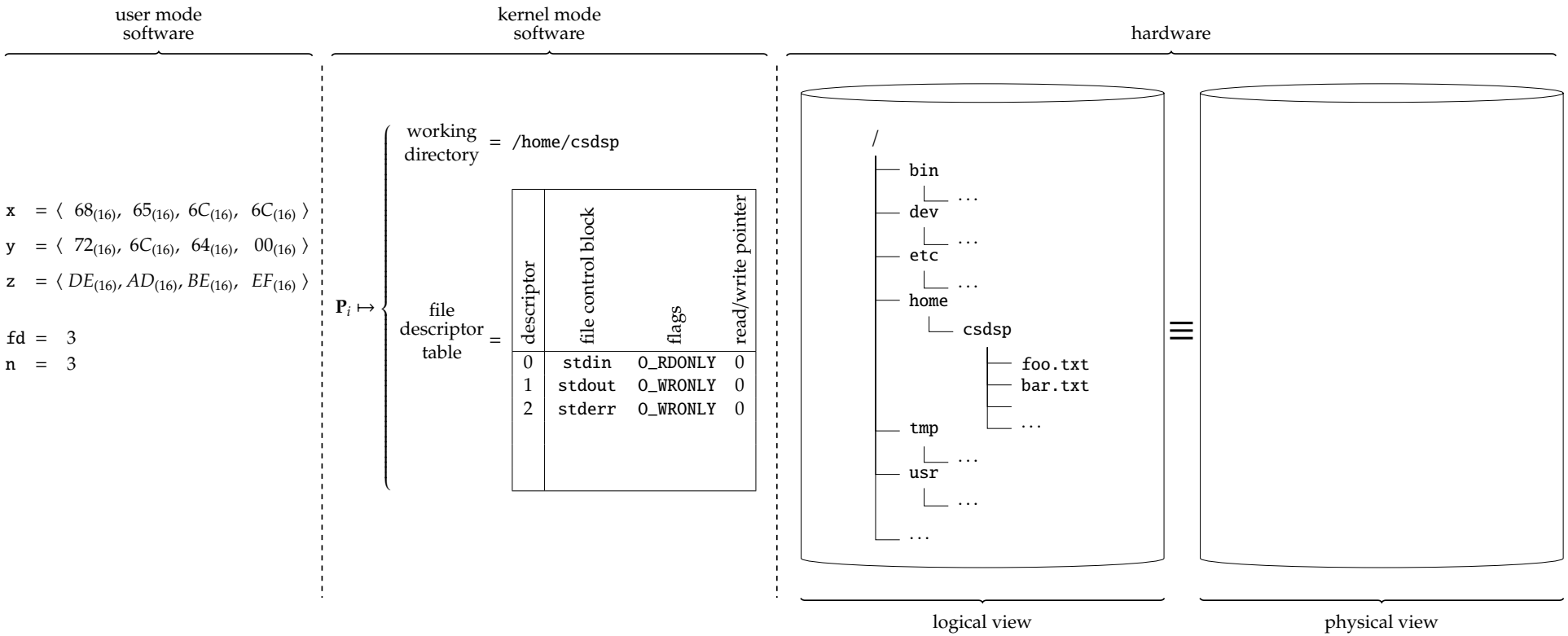


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
close( fd )
```

then the result is described by

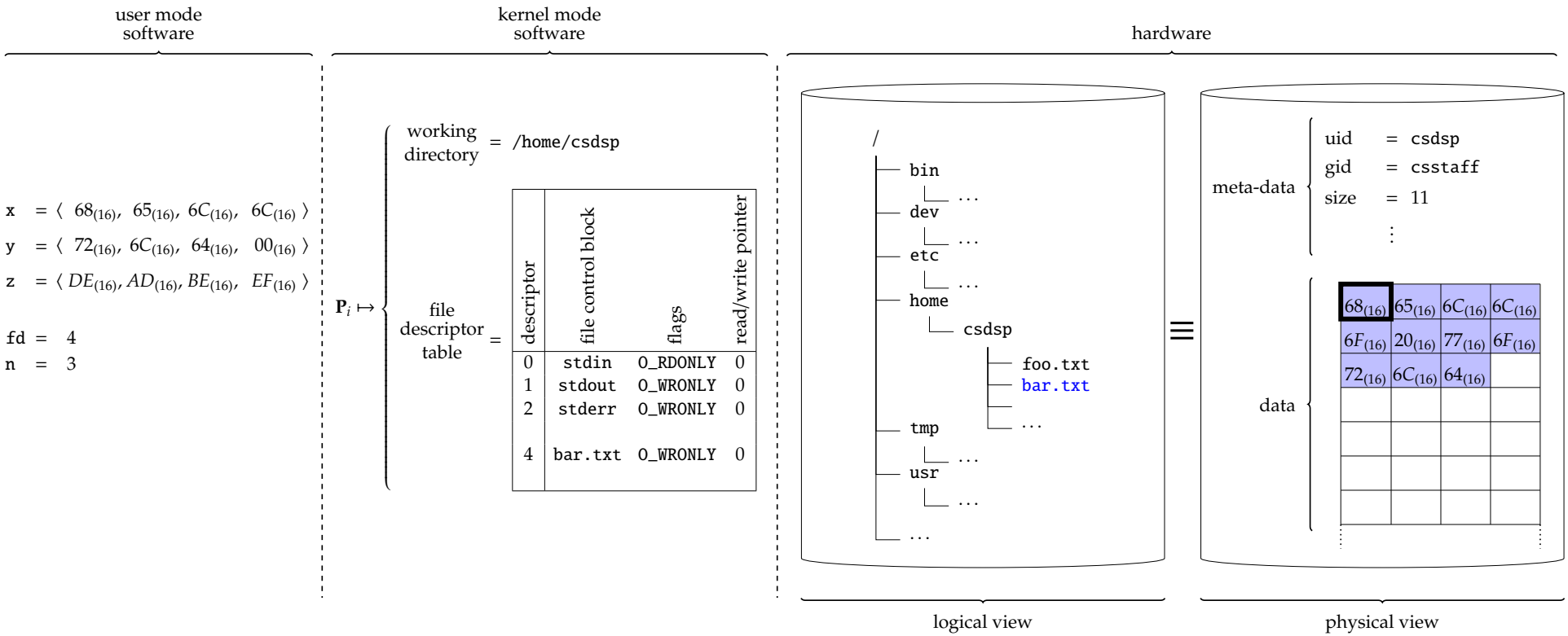


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
fd = open( "bar.txt", O_WRONLY )
```

then the result is described by

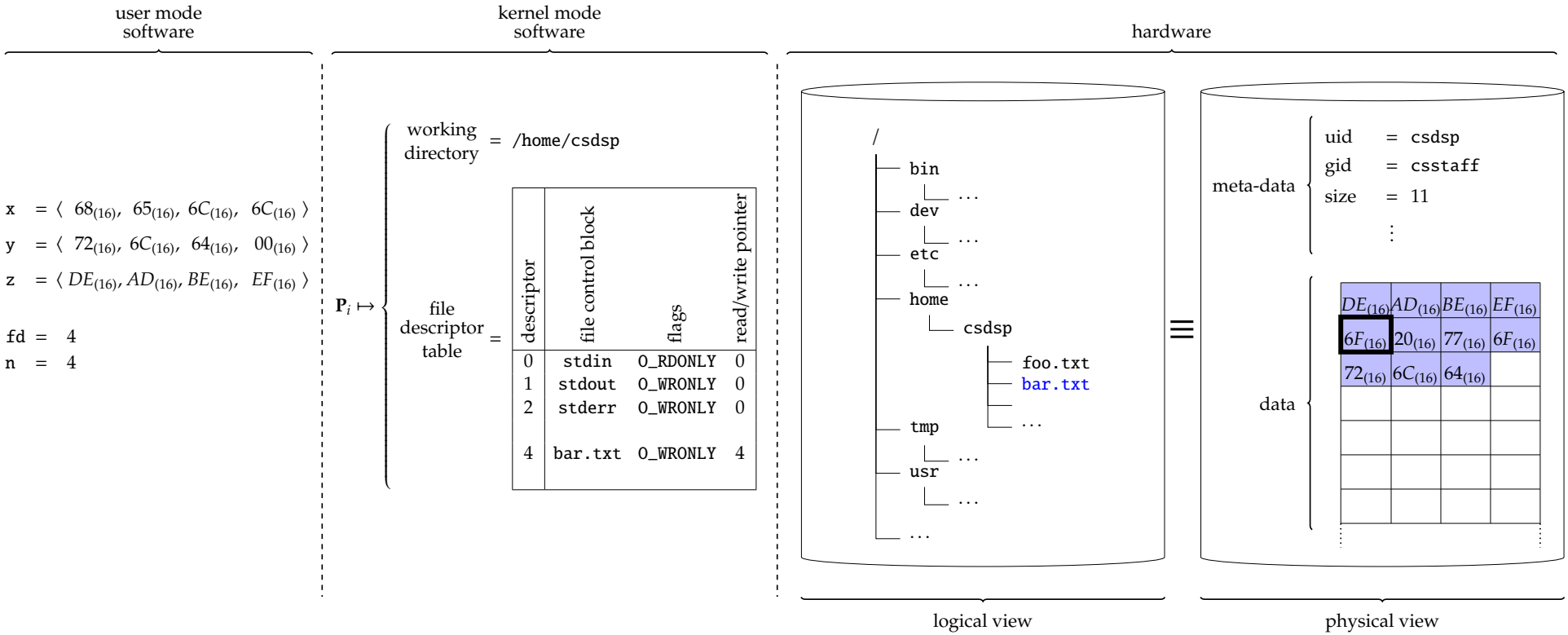


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = write( fd, z, 4 )
```

then the result is described by

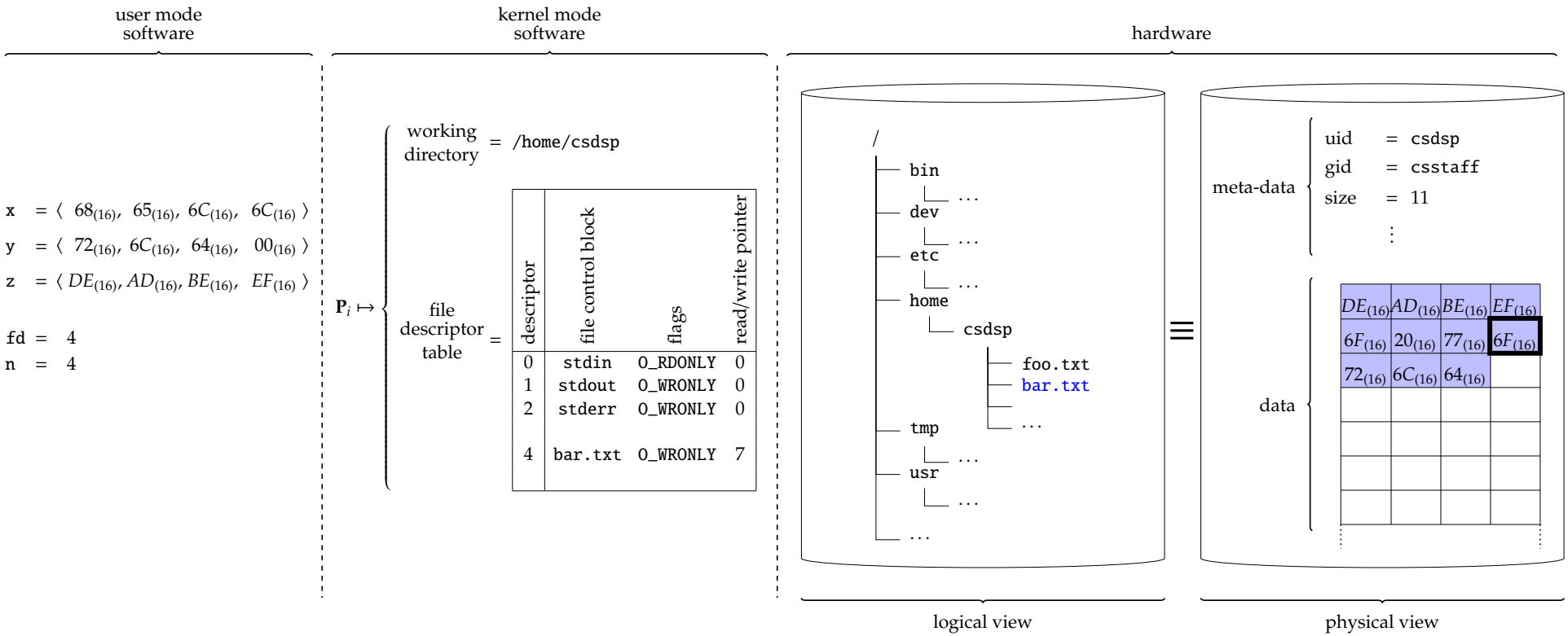


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
lseek( fd, -4, SEEK_END )
```

then the result is described by

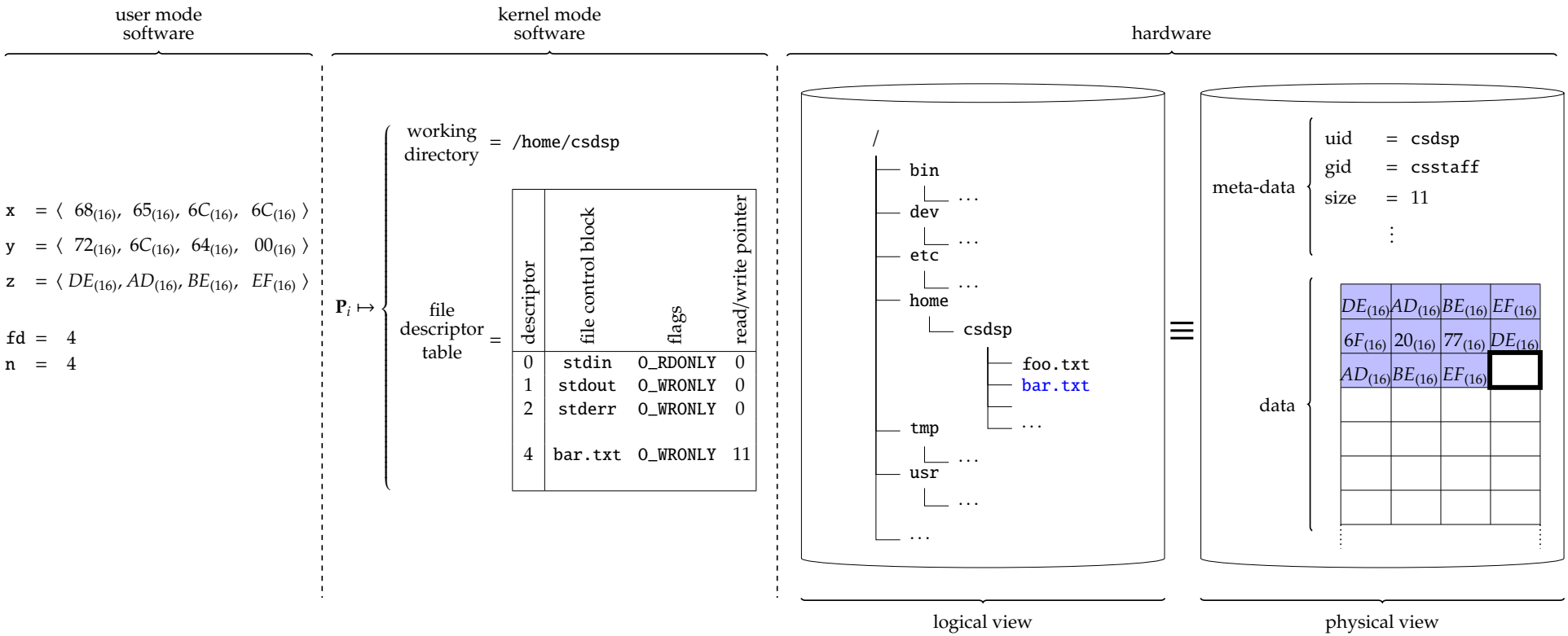


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = write( fd, z, 4 )
```

then the result is described by

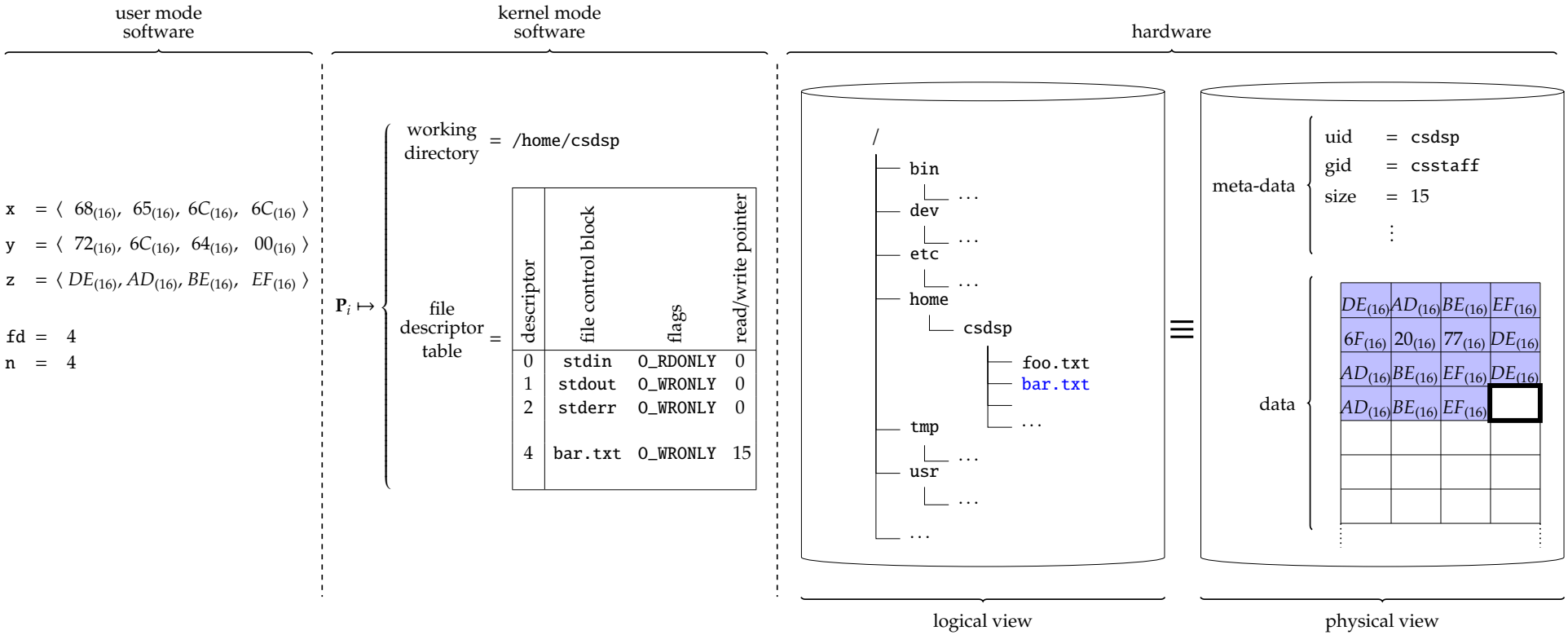


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = write( fd, z, 4 )
```

then the result is described by

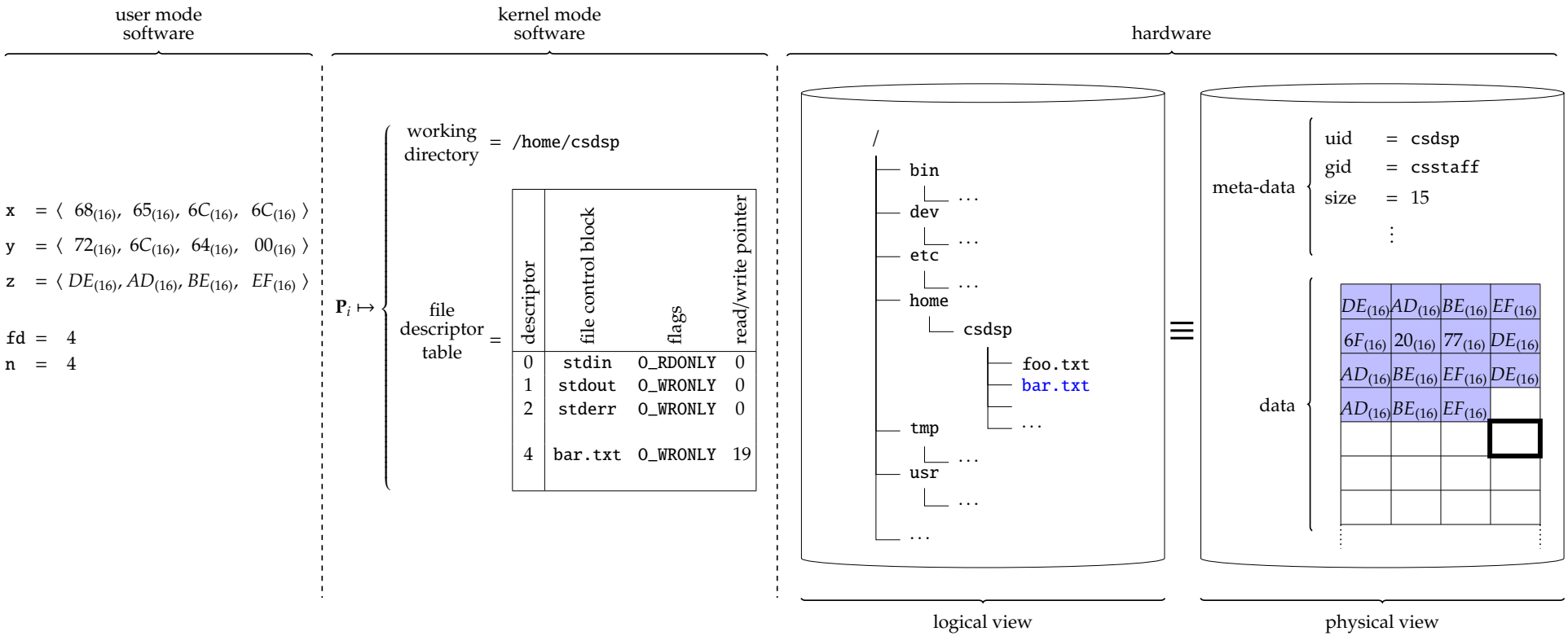


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
lseek( fd, +4, SEEK_END )
```

then the result is described by

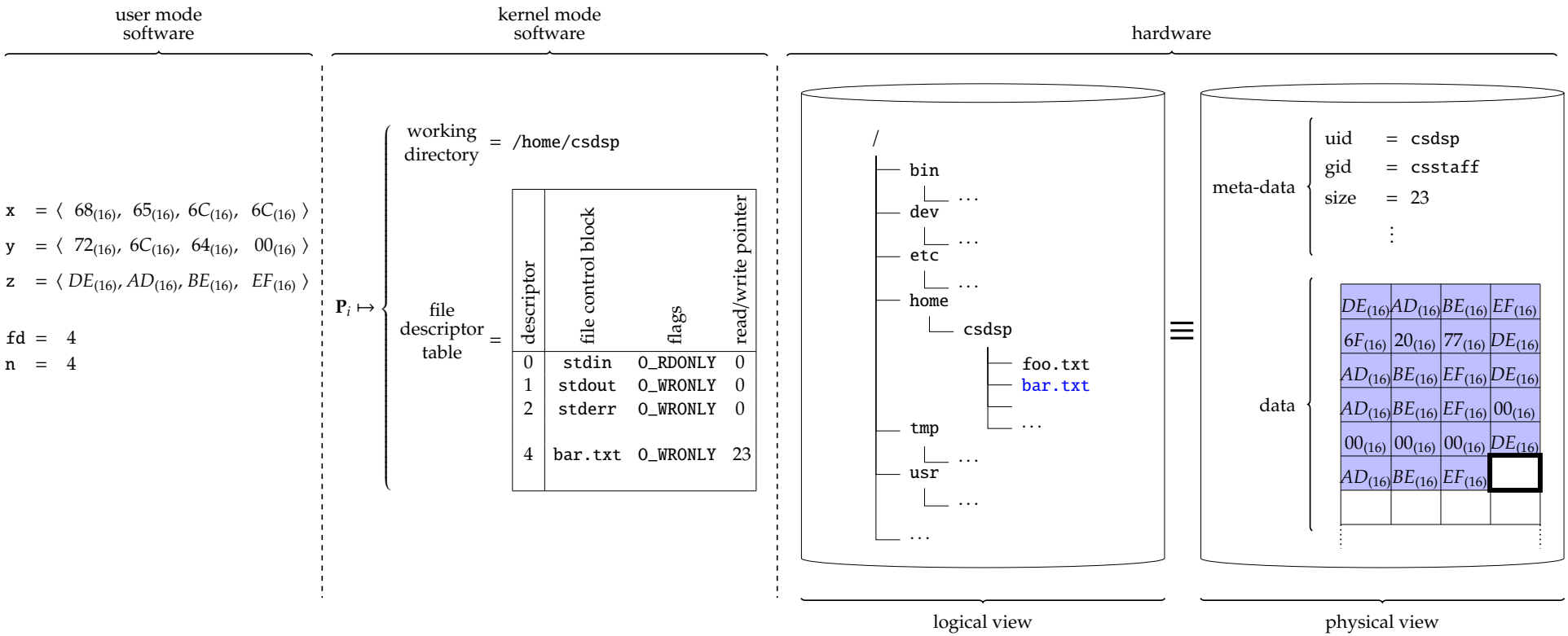


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = write( fd, z, 4 )
```

then the result is described by

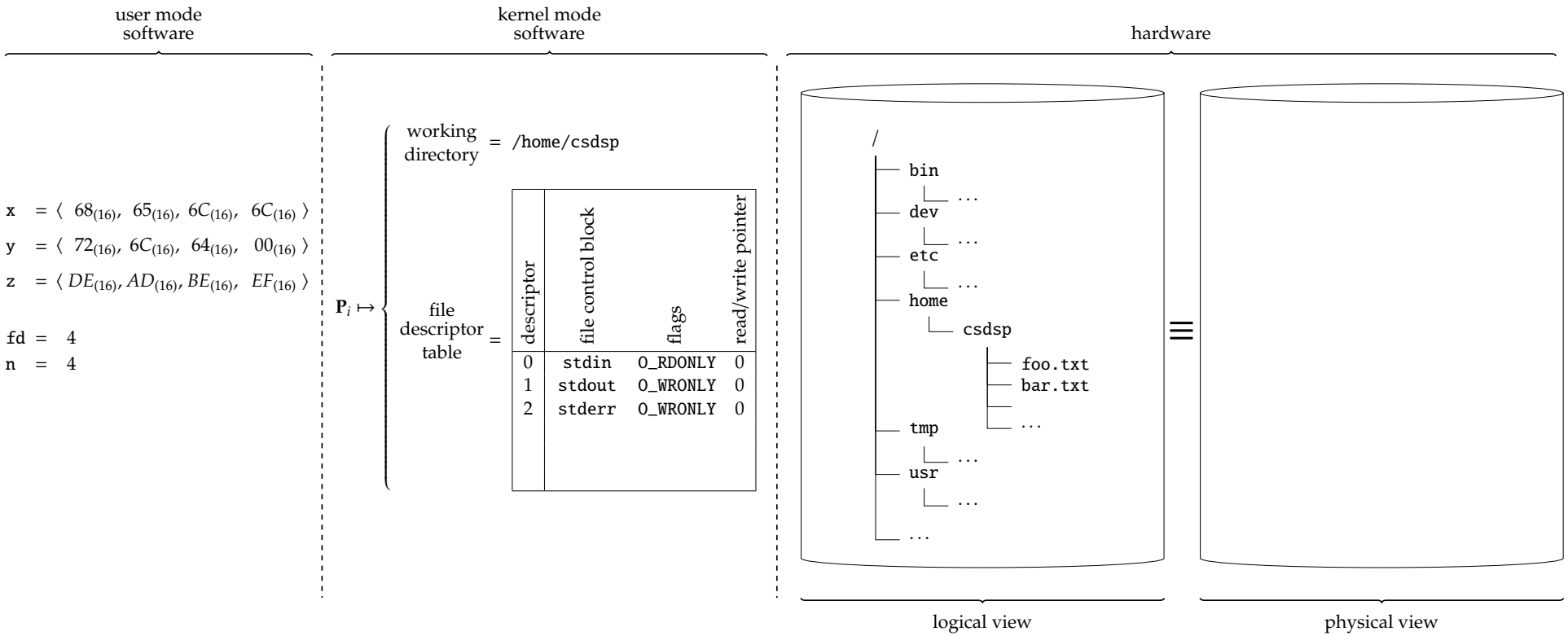


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
close( fd )
```

then the result is described by

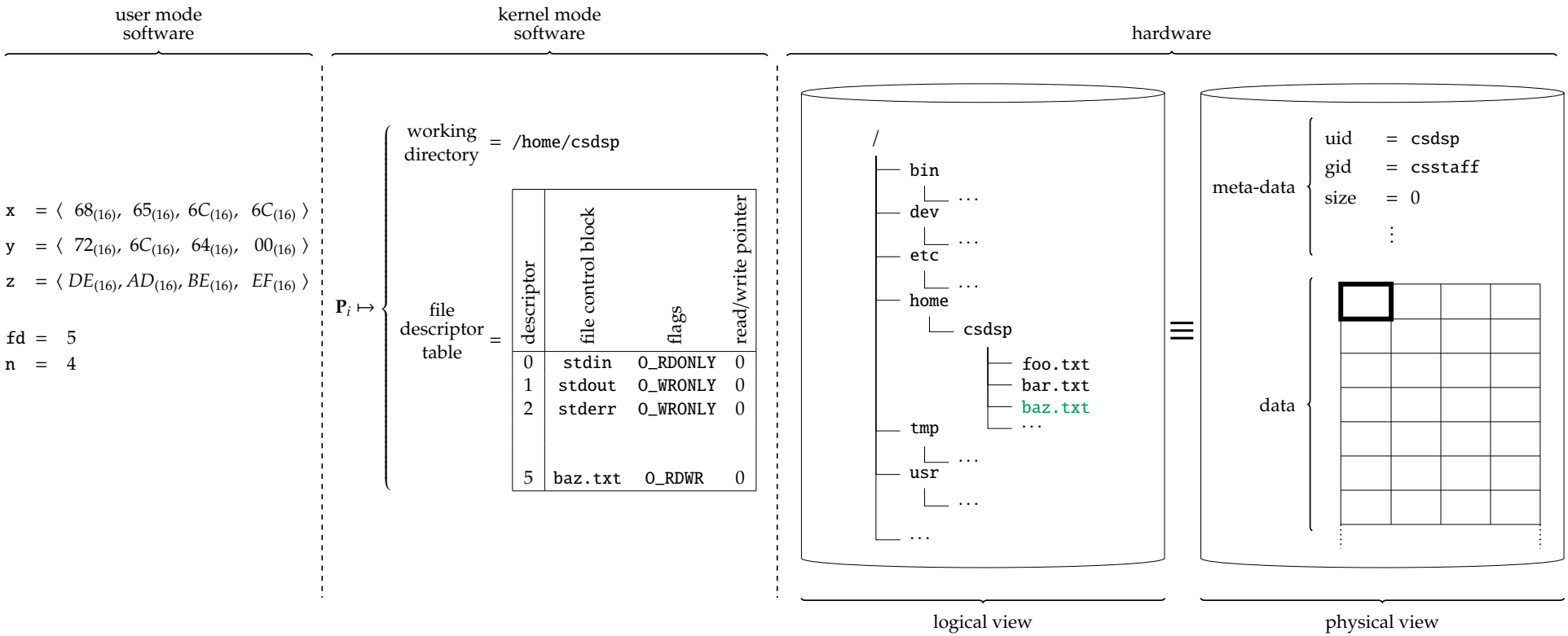


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
fd = creat( "baz.txt", O_WRONLY )
```

then the result is described by

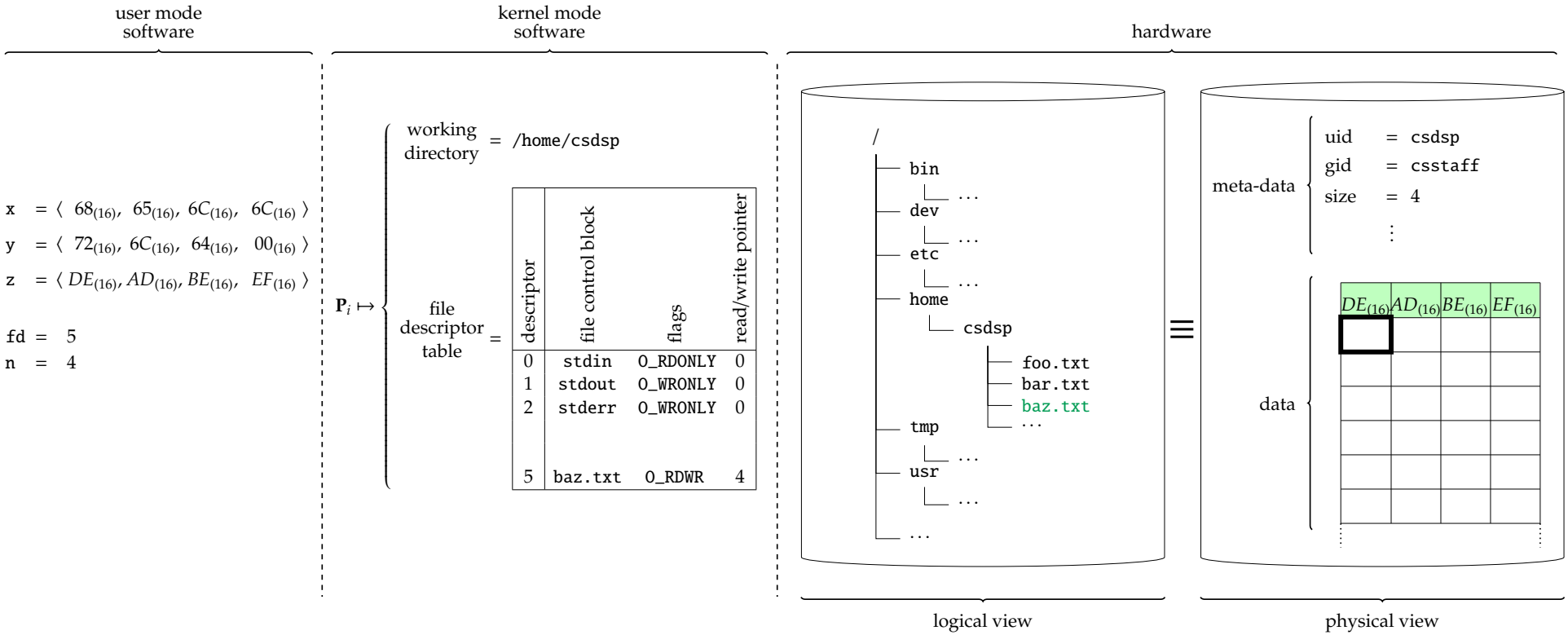


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = write( fd, z, 4 )
```

then the result is described by

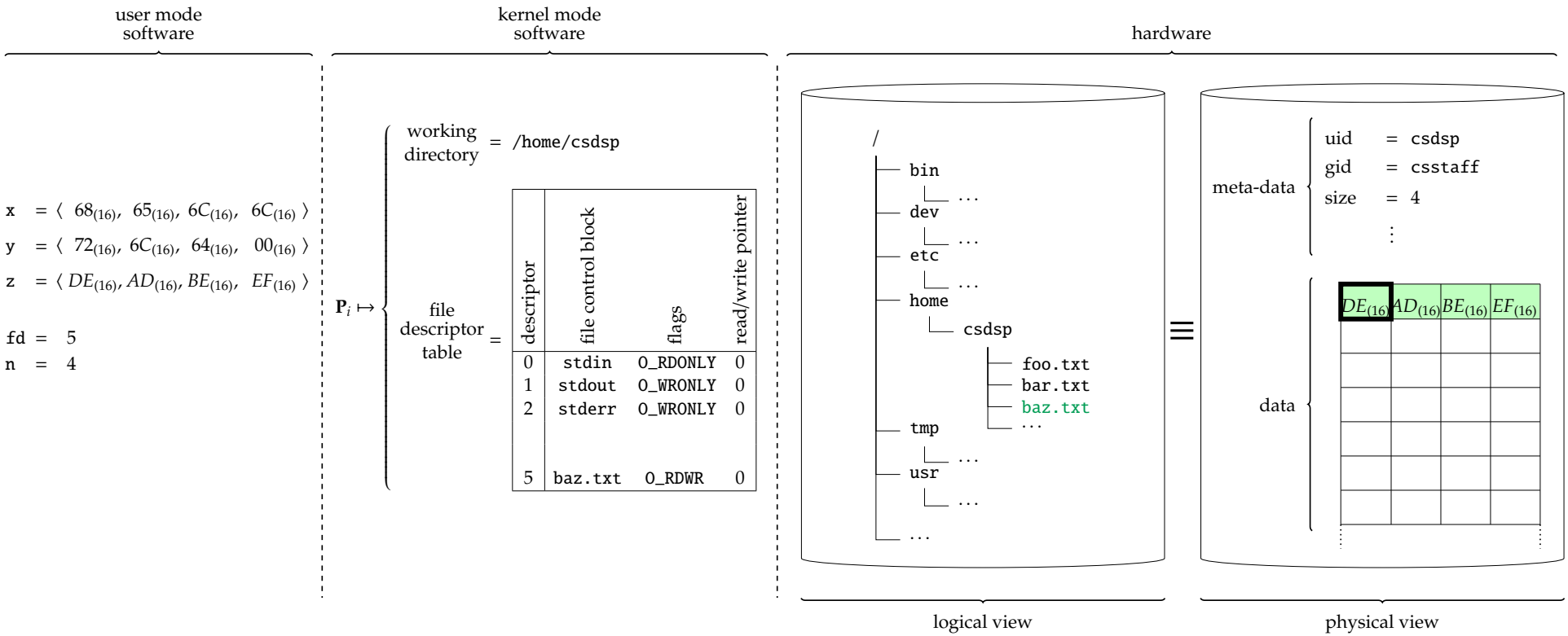


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
lseek( fd, +0, SEEK_SET )
```

then the result is described by

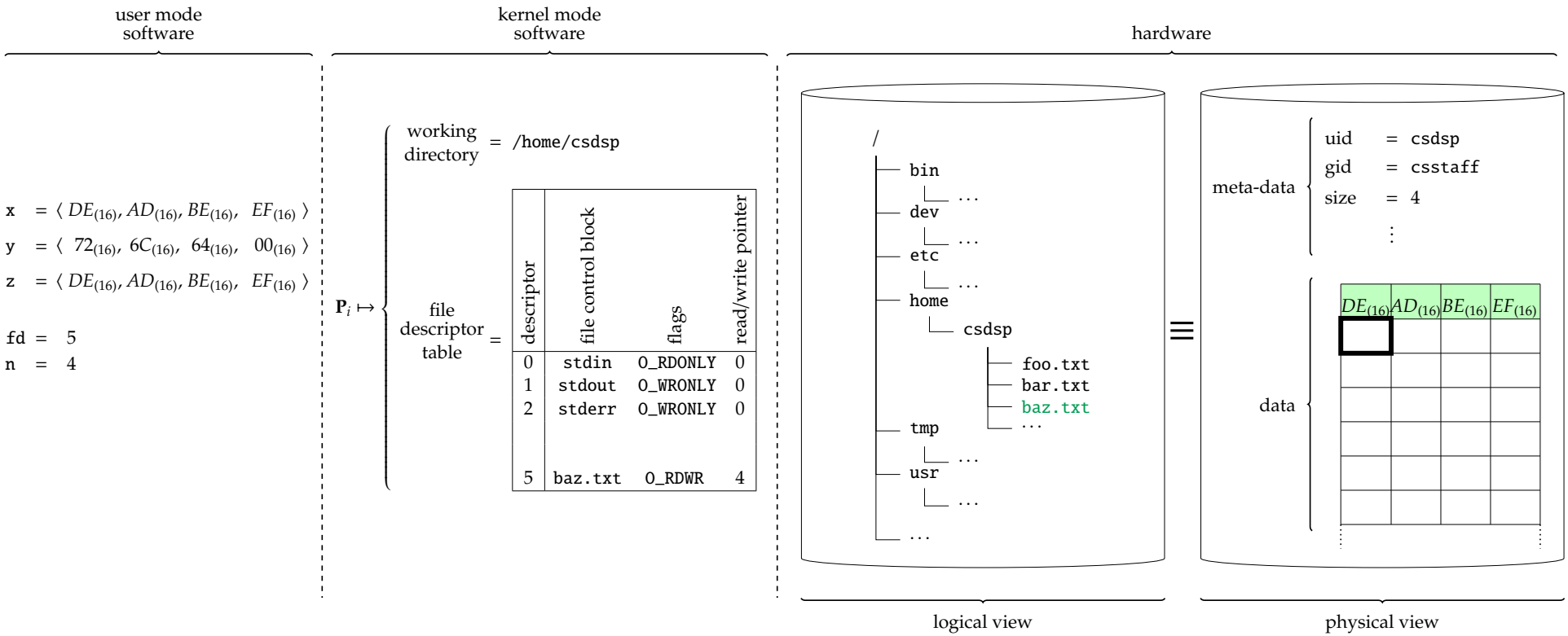


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
n = read( fd, x, 4 )
```

then the result is described by

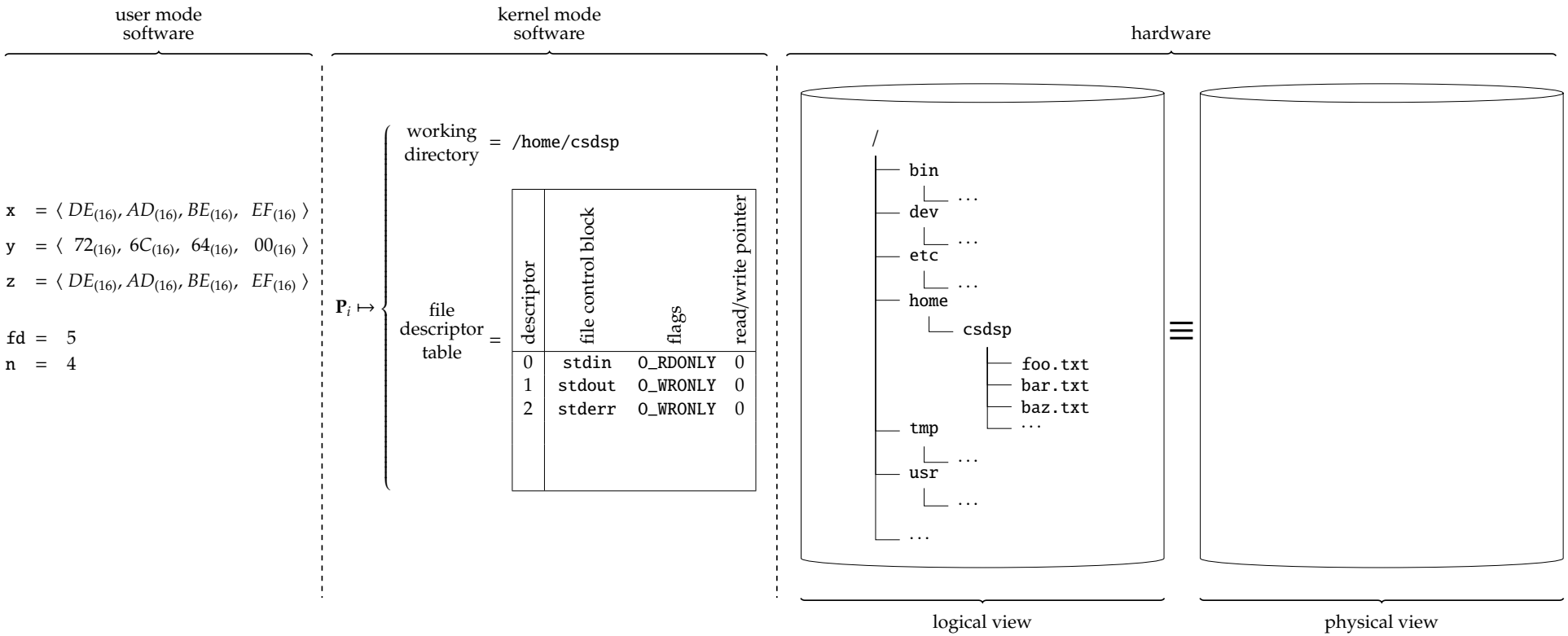


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

```
close( fd )
```

then the result is described by

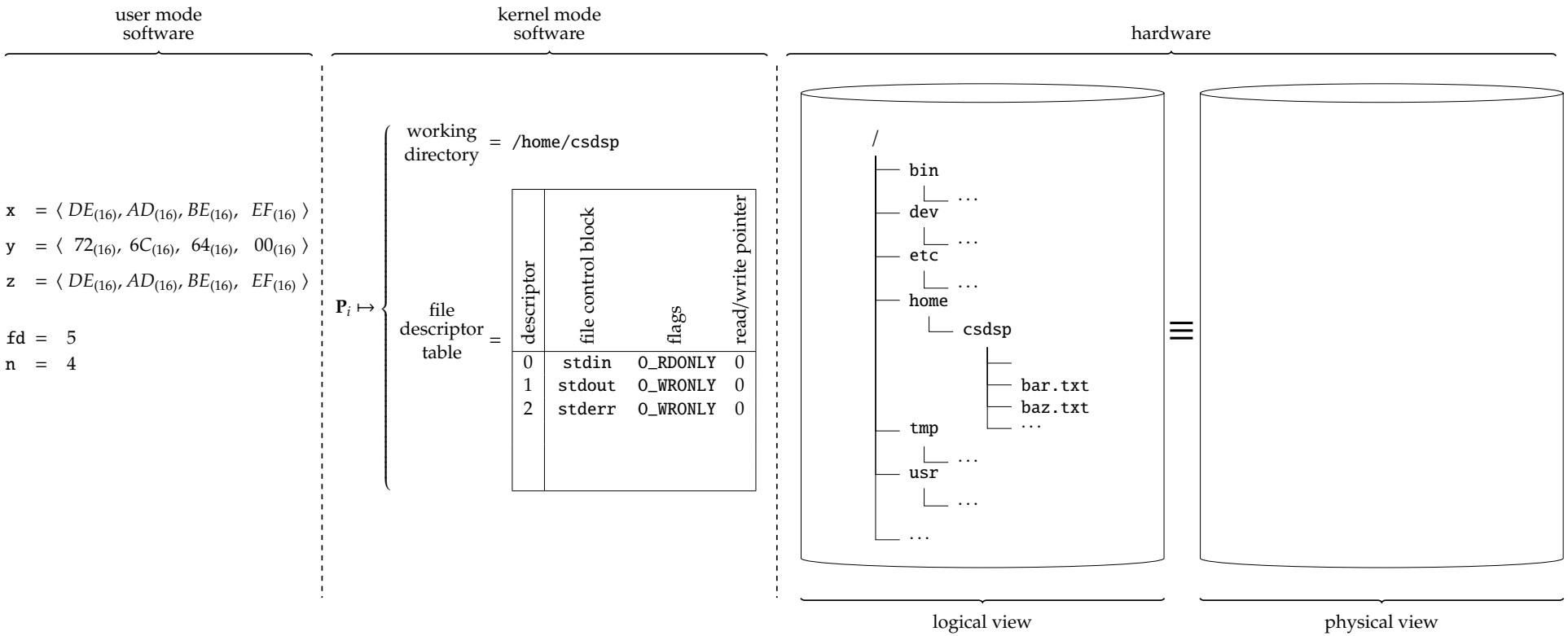


Mechanism: POSIX(ish) system call interface (4)

- Example: if the user mode process executes

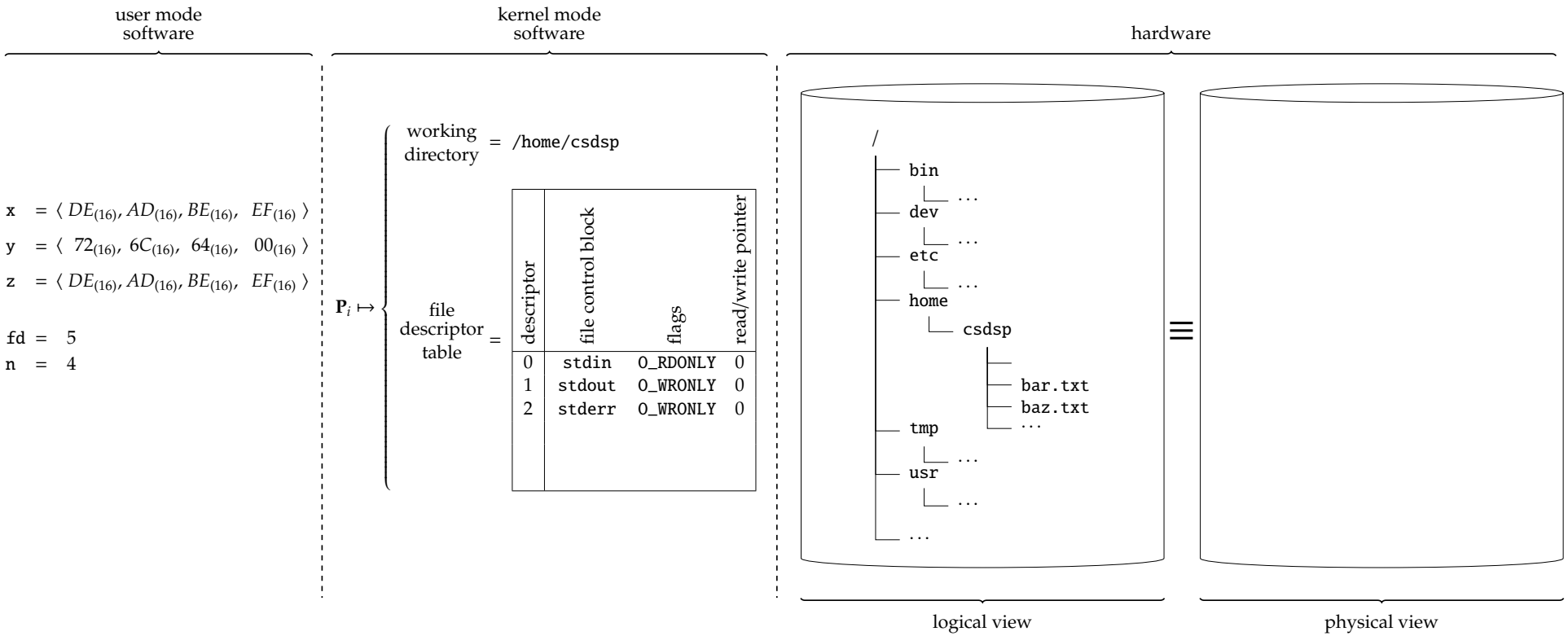
```
unlink( "foo.txt" )
```

then the result is described by



Mechanism: POSIX(ish) system call interface (4)

- Example:



Continued in next lecture ...

References

[1] Wikipedia: File system.
https://en.wikipedia.org/wiki/File_system.

[2] Wikipedia: Path.
https://en.wikipedia.org/wiki/Path_(computing).

[3] Standard for information technology - portable operating system interface (POSIX).
Institute of Electrical and Electronics Engineers (IEEE) 1003.1, 2008.
http://standards.ieee.org/findstds/standard/1003.1-2008.html.

[4] M. Gorman.
Chapter 11: Swap management.
In *Understanding the Linux Virtual Memory Manager*. Prentice Hall, 2004.
http://www.kernel.org/doc/gorman/.

[5] A. Silberschatz, P.B. Galvin, and G. Gagne.
Chapter 10: File system.
In *Operating System Concepts*. Wiley, 9th edition, 2014.

[6] A.S. Tanenbaum and H. Bos.
Chapter 4: File systems.
In *Modern Operating Systems*. Pearson, 4th edition, 2015.