csci 3411: Operating Systems

File System API/Filesystems

Gabriel Parmer

Slides evolved from Silberschatz

Operating Systems – Abstraction

- Make the system easier to use
- Abstract many of the details of the physical hardware away
  - Scheduling – illusion of exclusive execution
  - Virtual Memory – illusion of exclusive memory access
- Need abstractions/interfaces for
  - Interacting with devices (disks, terminals, keyboards)
  - Interacting with other processes

- Emphasis: *simple is beautiful, K.I.S.S.*
  - UNIX syscalls ~ 20-60, Windows syscalls ~ thousands

Abstraction of Disk

- Disk is a large array of bits – like memory
  - Partitioned into *sectors* (usually 512 bytes)
    - Unit of transfer to and from disk
  - Non-volatile – data persists across reboots/powerdowns
    - vs. memory?
- How do we share the disk between processes?
  - Protection/Security
- How do we make the disk easy to use?
  - While maintaining efficiency

Files/Directories

- Abstraction: Files
  - Logical storage unit, collection of related data
  - Generic storage – untyped sequence of bytes
    - Binaries (executables, media, etc...)
    - Ascii (text documents)
    - Structured documents (HTML/XML)
  - More structure == good, right?
    - Why not store everything as XML?
- Abstraction: Directories
  - Give an *address/location* to a file and group files

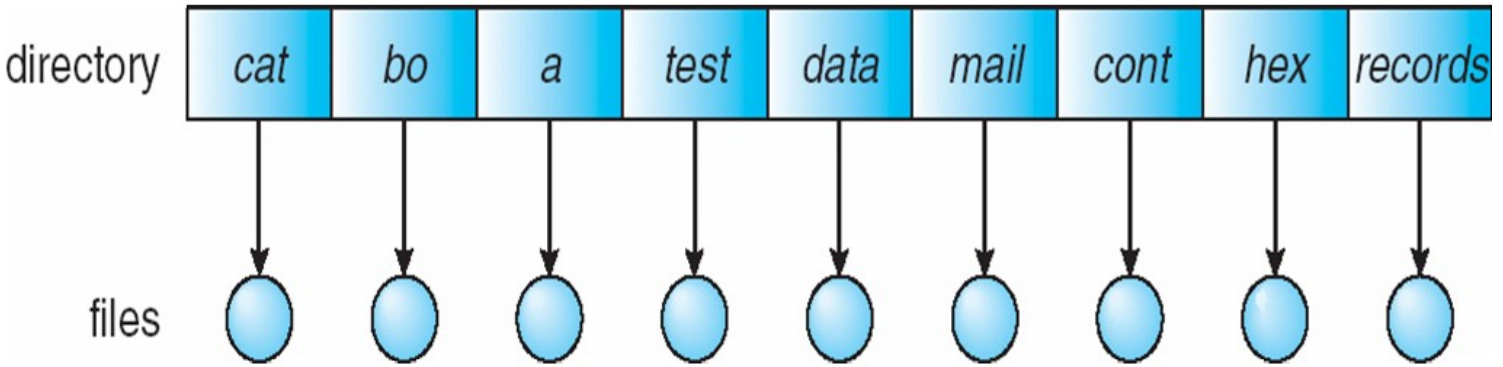
Files Conceptually Include...

- Name – human usable name
- Identifier (inode number) – unique identification
- Size – size in bytes
  - fragmentation?
- Date/Time – creation time, access time
- Protection (UNIX)
  - Owning user and group
  - List of access rights (read, write, execute)
    - For owner, other users in *group*, and all others

Naming of Files

- Naming: How do we refer to or *address* resources
  - Files, devices, information about processes, …
    - Also named: memory, threads, semaphores, etc...
- Directory hierarchy/namespace – goals:
  - Efficiently locate and identify files
  - Allow intuitive *grouping* of like files together
  - Useful for both OS and users

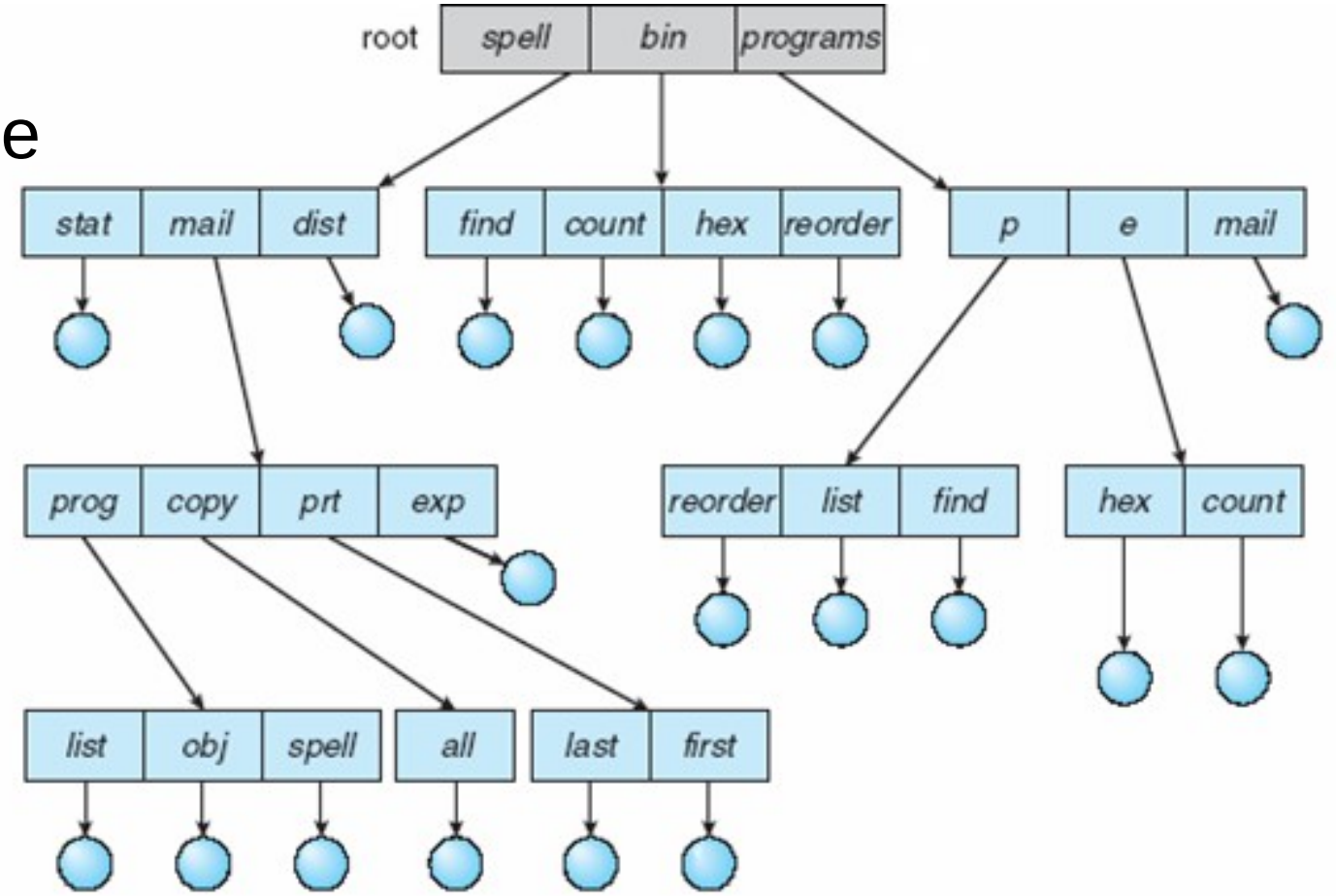
Single-Level Directory



- Imagine if all files were in a *single* directory!
- No abstraction within the namespace
  - No grouping of like files together
  - Cannot have two files with the same name
    - Only one index.html!
- Fast!!!
  - Must only access a single directory entry (get file location, check permissions)
- Gmail, memcached, cell phones, itunes, ...

Hierarchical Directory

- Intuitive grouping of files
- Serialized access of directories to access files
  - permissions
- Files can share names
- Can also be used to address devices, etc...
  - As you've seen



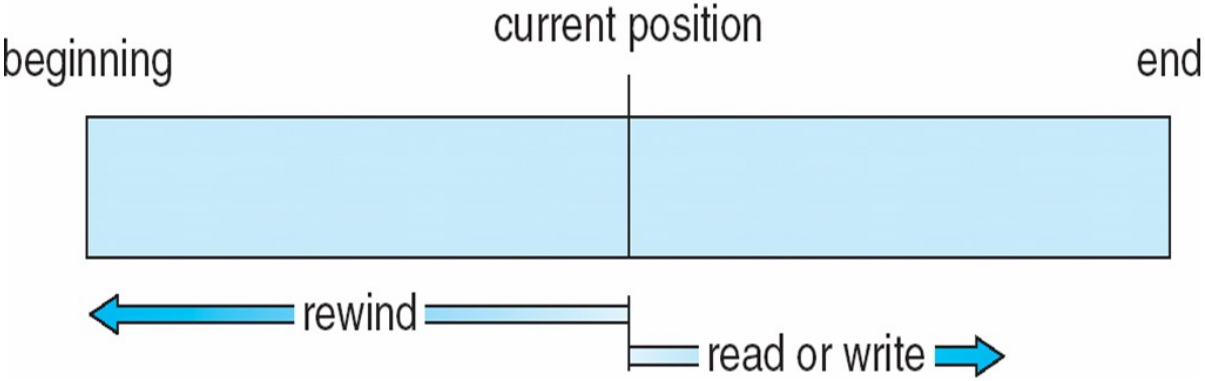
Hierarchical Directory Interface

- How can we use/manipulate this namespace?
  - *mkdir(p)/rmdir(p)* – create/remove a directory entry
  - *readdir(dir_id)* – read a directory entry (retrieve the files/directories it contains)
    - *dir_id = opendir(path)/closedir(dir_id)*
  - *rename(p1, p2)*

  - *creat(p)* – create a file
  - *unlink(p)* – remove/delete a file
- *p* is a path through the directory hierarchy (e.g. /a/b/c.txt)

Access/Manipulate Files

- Second namespace – per-proc *File Descriptors*
  - *read(fd,...)* – retrieve contents of file sequentially
  - *write(fd,...)* – modify contents of file sequentially



  - *seek(fd, off)* – Change current file's position/offset
    - Enables random access
  - *new_fd = dup(fd)* – duplicate a file descriptor
  - *close(fd)* – deallocate fd, signal we aren't using file

Interaction Between Namespaces

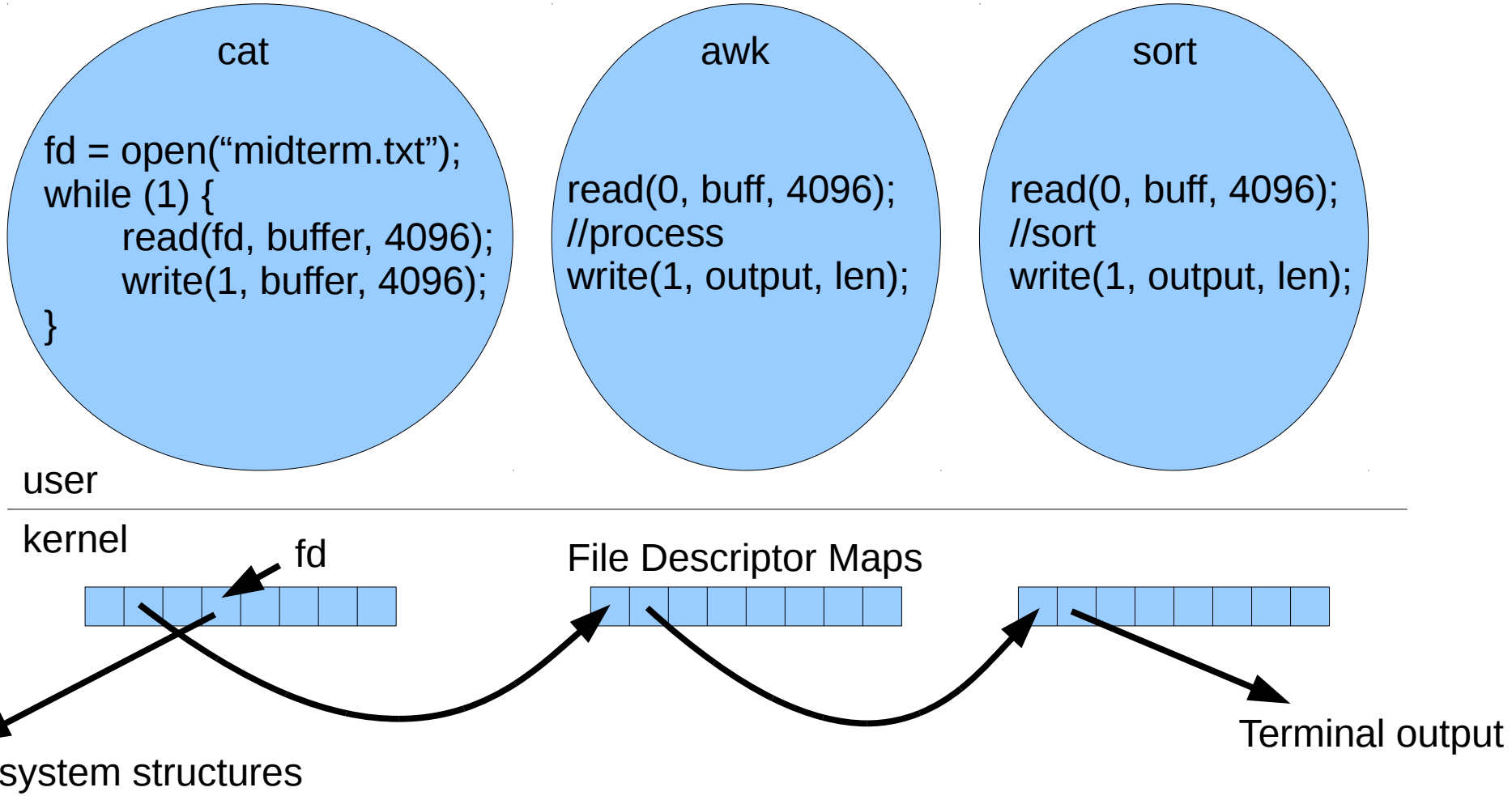
- Directory namespace must interact somehow with file descriptor namespace
  - Must be able to *read/write* files in the directory namespace!
- *fd* = *open(p)* – allocate fd associated with a path in the directory namespace
  - Locate *p* in the directory hierarchy
  - Check user's permissions to *p*
- ***Why file descriptors at all!?***
  - *read(p, ...)/write(p, …)* instead?

File Descriptors: Unified Interface for Accessing System Resources

- Polymorphic
  - The same functions (read/write) used to access/modify many different types of resources
  - Large versatility of a small number of functions
    - A single user program can behave differently depending on what its file descriptors access

File Descriptors/File Operations

- **cat midterm.txt | awk '{print $4;}' | sort**
  - Common language spoken between them all: text/ascii



More File Descriptor Justifications

- Race conditions:

```
read(“/blah”, ...);
unlink(“/blah”);
read(“/blah”, ...);
// FAIL
```

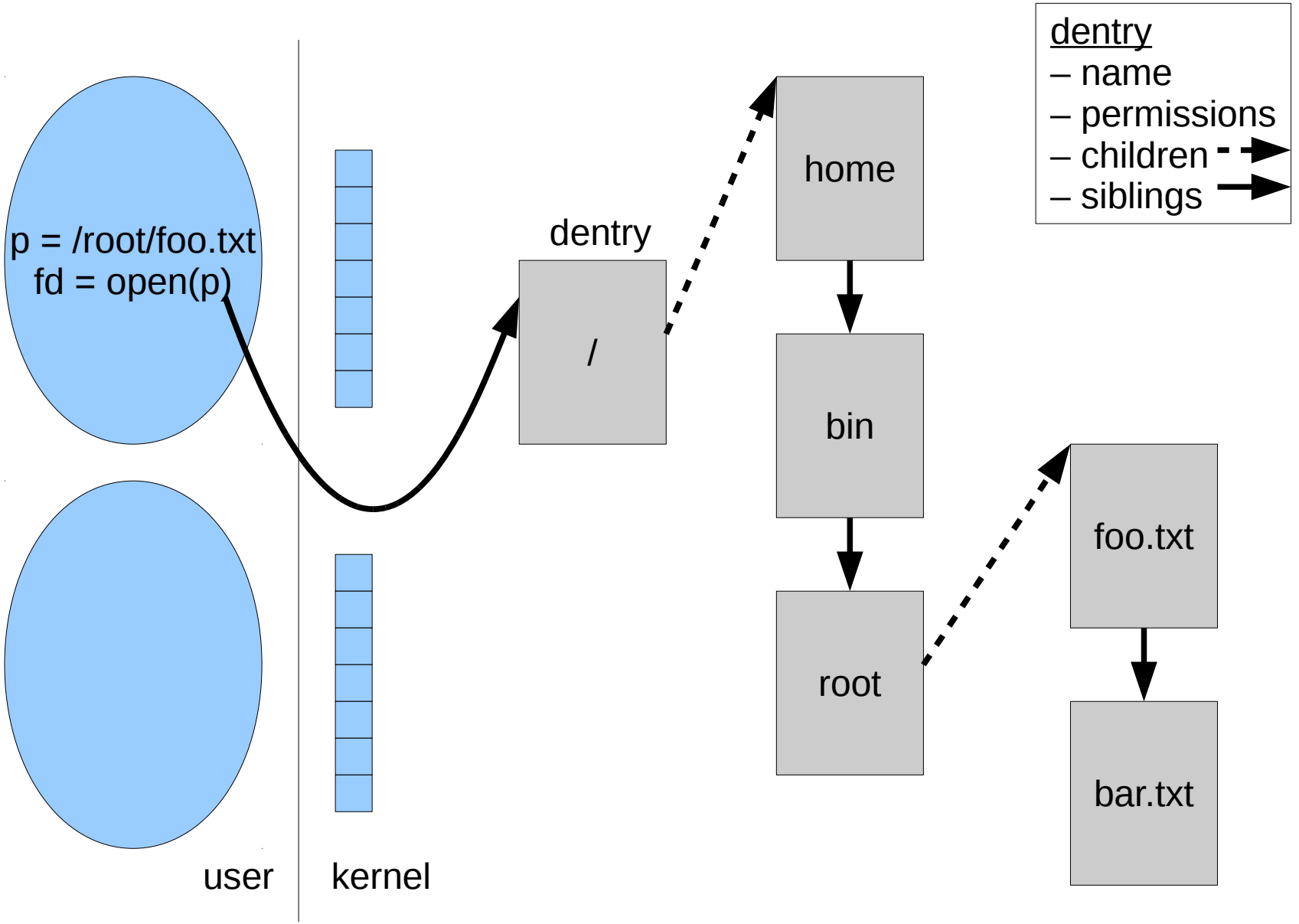
vs.

```
fd = open(“/blah”, ...);
read(fd, ...);
unlink(“/blah”);
read(fd, ...);
```

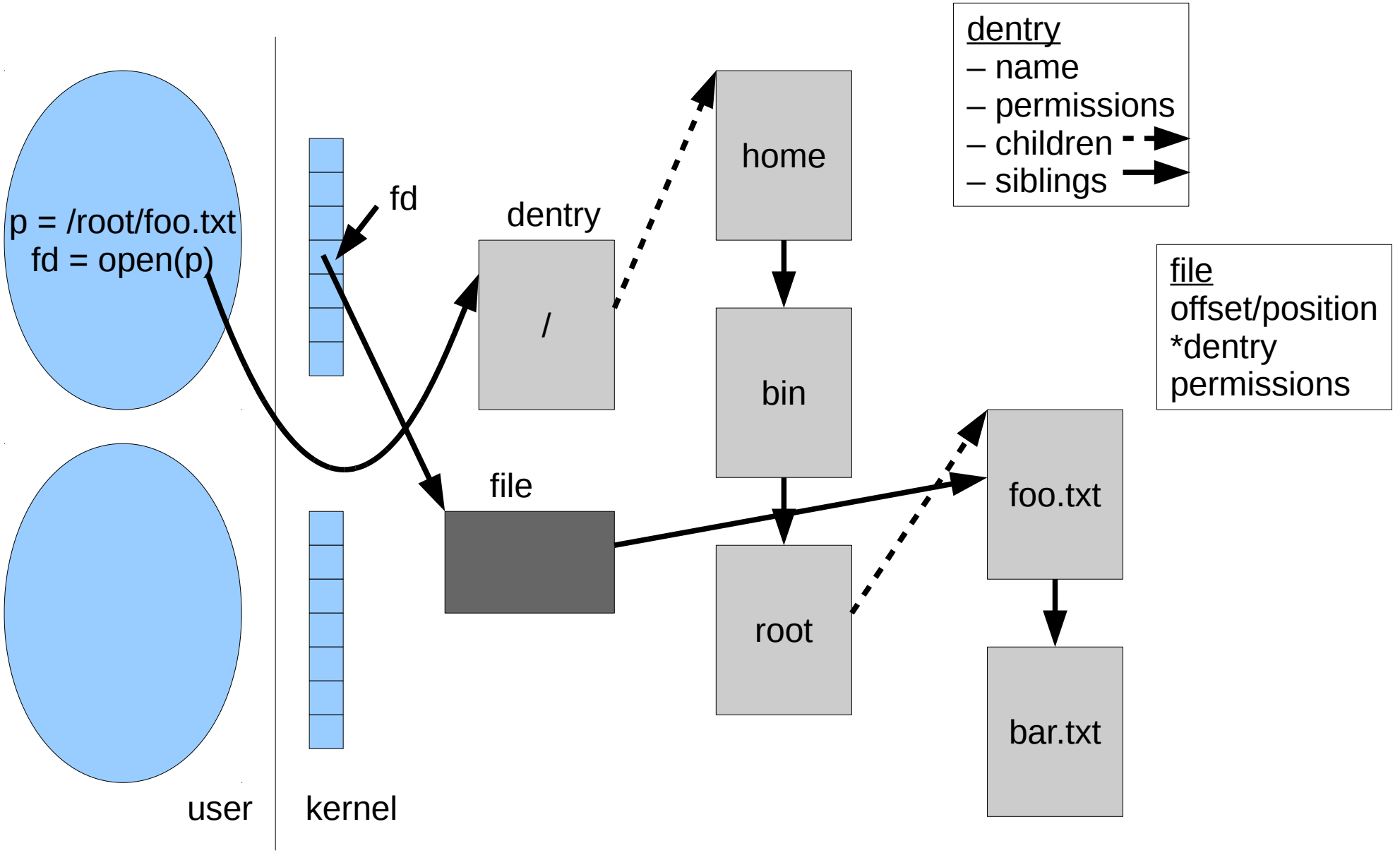
- Keep track of file offset

- Efficiency!
  - Don't have to go through entire directory path every time we *read/write*

OS Datastructures for FS & fds



OS Datastructures for FS & Fds



OS Datastructures for FS & Fds