Today

- Today's Learning Outcomes
  - Explain the different data structures that map between file descriptors and files.
  - Relate these different representations to what aspects of file management are shared among threads, processes, users, etc.
  - Identify when two file accesses share an offset and when they do not.
  - Define:
    - File descriptor table
    - Open file table
- Reading
  - 10.8

Recall:

- If you and I are both allowed to read a file in the file system, should we be able to share the file's data? **Yes!**
- If two processes are reading (writing) the same file, should they be using the same file offset? **No!**
- If a process opens the same file twice, should it have one file descriptor or two? In either case, should the two opens share an offset? **No!**
- If two threads are using the same file descriptor, should they be using the same file offset? **Yes!**
- If one process (the parent) creates another process (a child), should the child inherit the parent's file descriptors? Will they use the same offset? **Yes!** **Yes!**

Per-Process data Structures

- Facts:
  - When a process issues the open system call, the return value is a small integer called a file descriptor.
  - Every process starts with three open file descriptors:
    - Standard input: fd = 0 (STDIN_FILENO)
    - Standard output: fd = 1 (STDOUT_FILENO)
    - Standard error: fd = 2 (STDERR_FILENO)
    - *Note: These are not stdin, stdout, stderr – those have different types.*
      - What are stdin, stdout, stderr anyway?
- Given these facts: What data structure(s) do we need to retain for each process?

Per-Process data Structures

- Facts:
  - When a process issues the open system call, the return value is a small integer called a file descriptor.
  - Every process starts with three open file descriptors:
    - Standard input: fd = 0 (STDIN_FILENO)
    - Standard output: fd = 1 (STDOUT_FILENO)
    - Standard error: fd = 2 (STDERR_FILENO)
    - *Note: These are not stdin, stdout, stderr – those have different types.*
      - What are stdin, stdout, stderr anyway?
- Given these facts: What data structure(s) do we need to retain for each process? A file descriptor table.

(File) Descriptor Table

- A per-process data structure.
- The file descriptors returned by open correspond to indices into this table.
  - Example: If a process calls open and gets an `fd` of 3 returned, then the file is represented by the 4th (recall that C arrays are 0-based) entry in that table.
- The table is mostly going to contain a pointer to other structures.
- Question: Can we store the offset in this table?

(File) Descriptor Table

- A per-process data structure.
- The file descriptors returned by open correspond to indices into this table.
  - Example: If a process calls open and gets an `fd` of 3 returned, then the file is represented by the 4th (recall that C arrays are 0-based) entry in that table.
- The table is mostly going to contain a pointer to other structures.
- Question: Can we store the offset in this table?
  - No! In UNIX/Linux-style operating systems, we create processes by one process (a parent) calling `fork` to create a new process (the child):
    a. The parent and child are different processes, but:
    b. They **share the offset of any file descriptors open when the parent forked**.

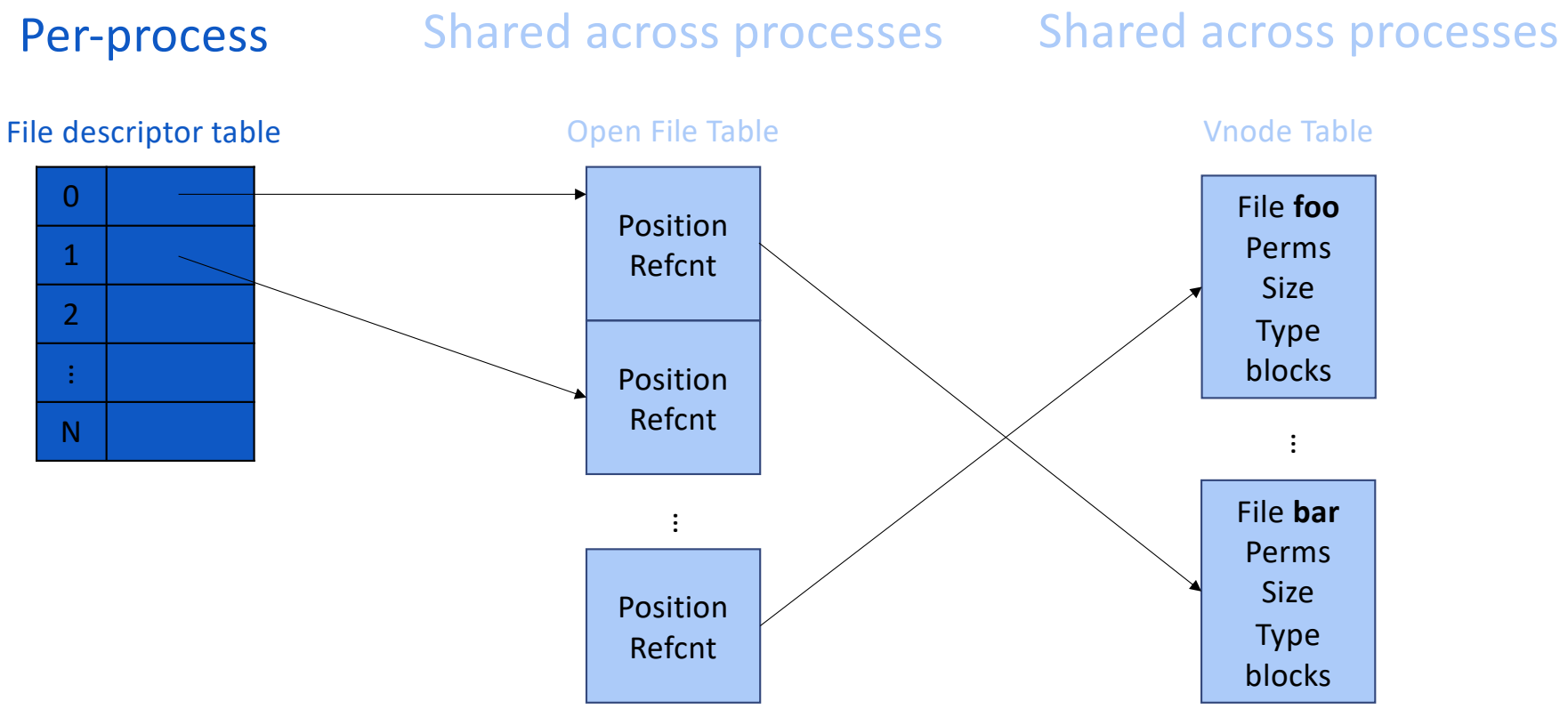
So, where do we store offsets?

- An offset corresponds to a call to open.
- We need a data a structure corresponding to each open.
- The **(open) file table**: Shared by all processes.
  - Maintains the file position (i.e., offset in the file).
  - Keeps a reference count (for when we have two fds referencing the same open file object (we'll see how that can happen).
  - Keeps a reference to an object that represents the actual file.

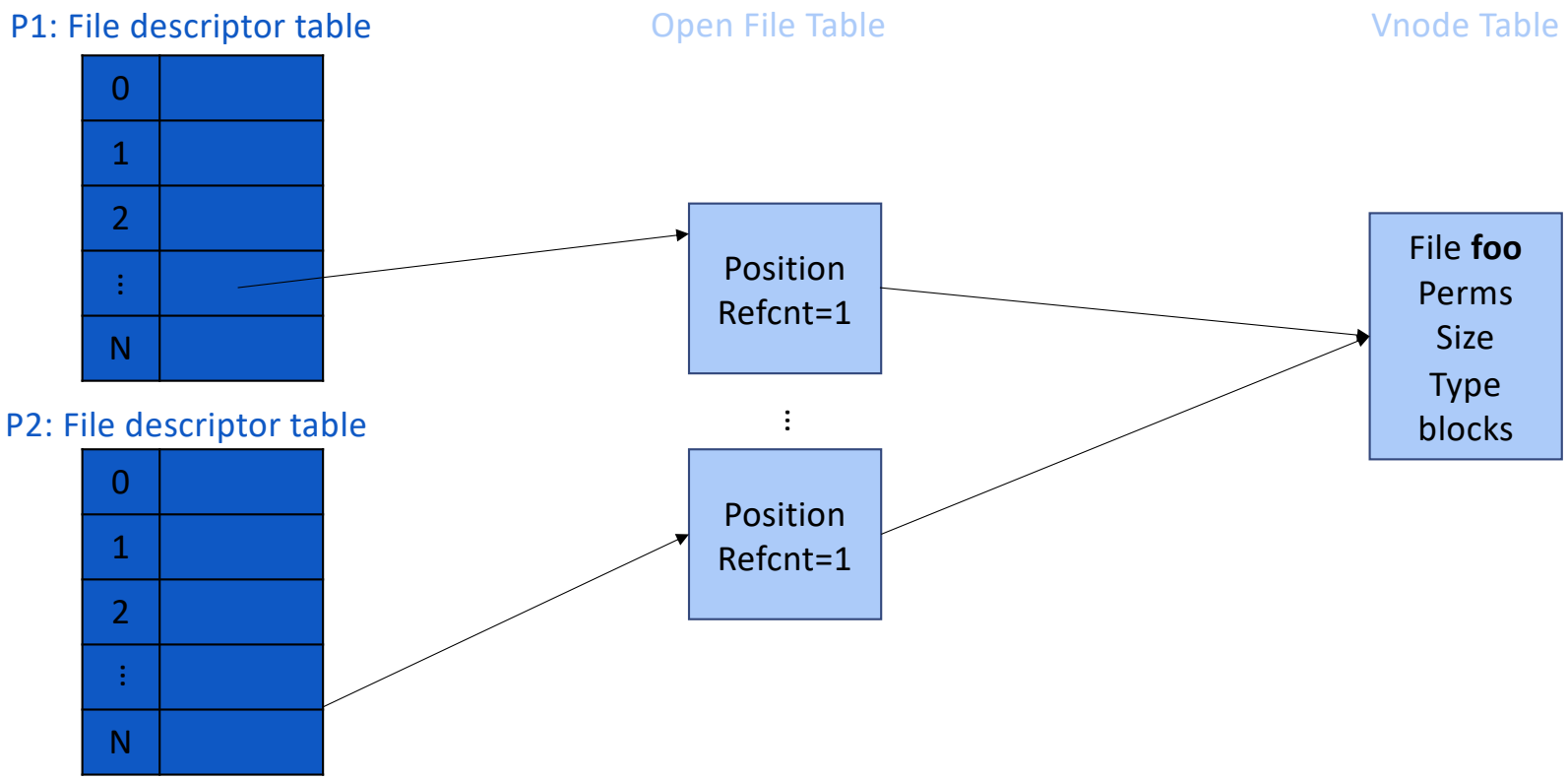
Finally: In-memory objects representing files

- Vnode: Virtual node: in-memory representation of a file
- Vnode table: collection of all vnodes; shared among all processes.
  - Contains a copy of the file's meta-data
  - Including a way to locate the file's blocks on disk (we'll talk more about precisely what this looks like when we talk about different ways to store files on disk).

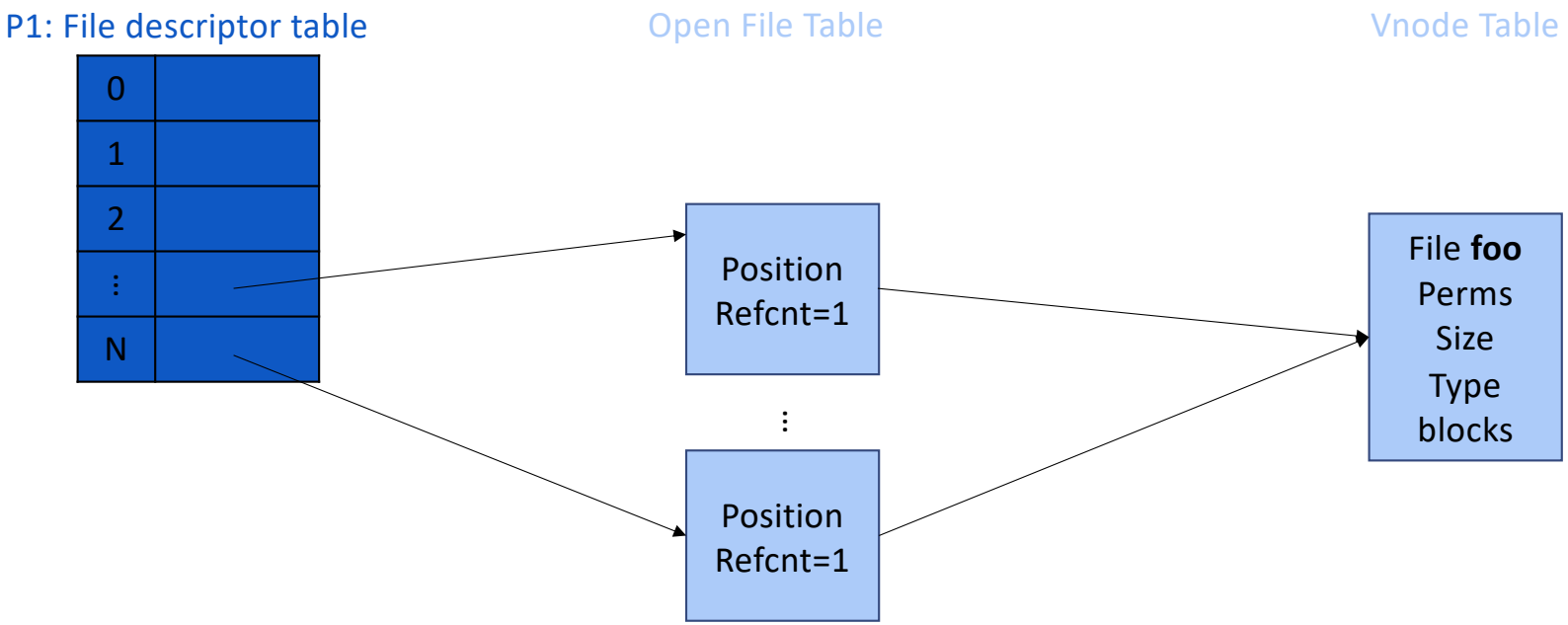
Putting this all together



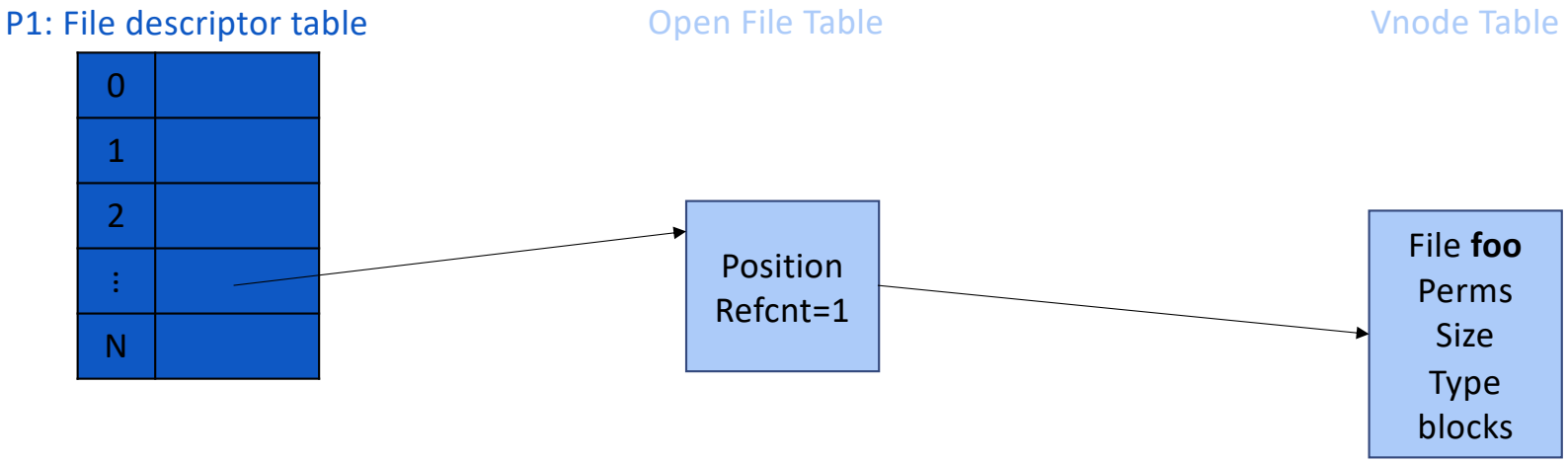
P1 and P2 read the same file



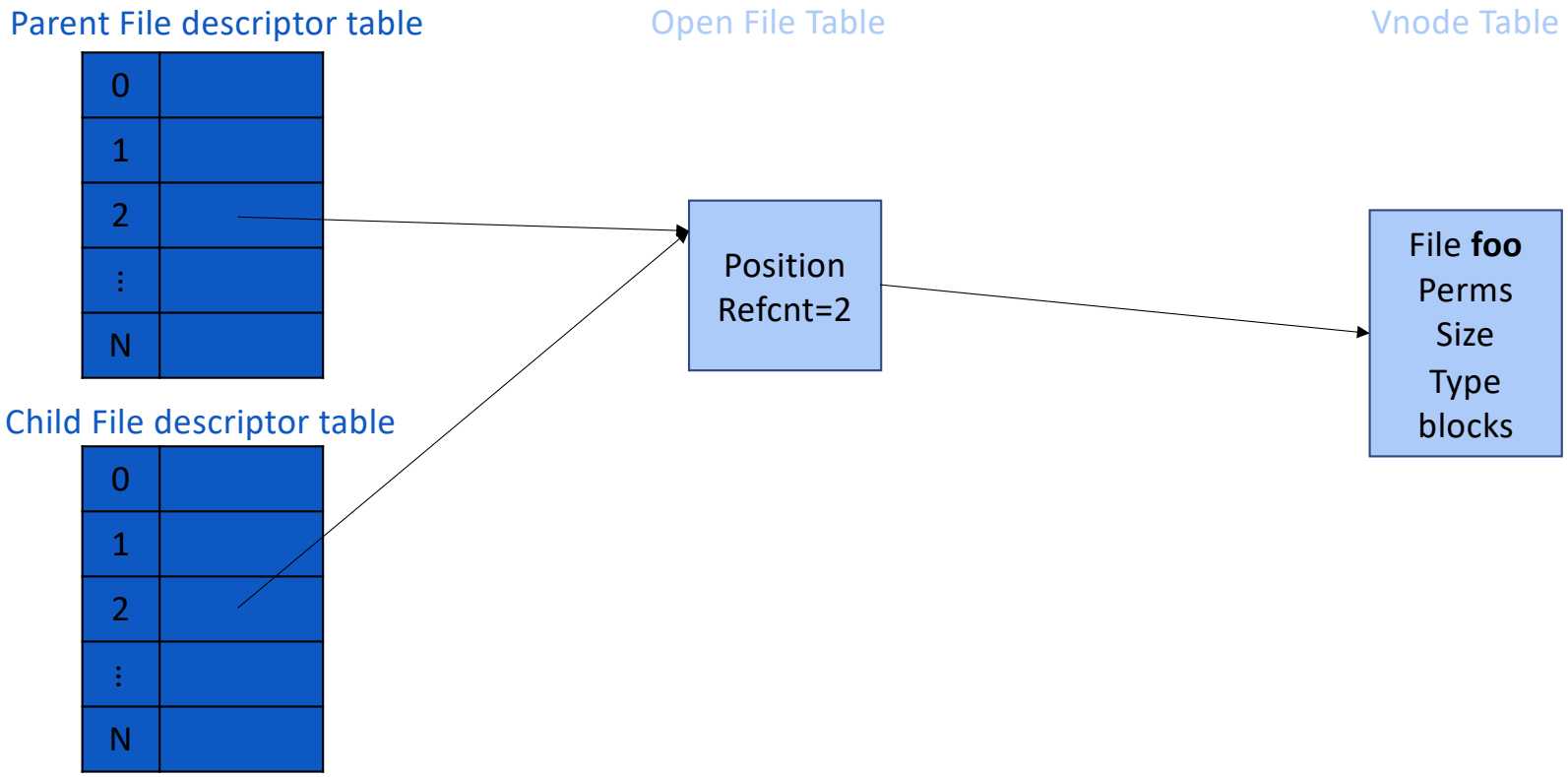
P1 opens the same file twice



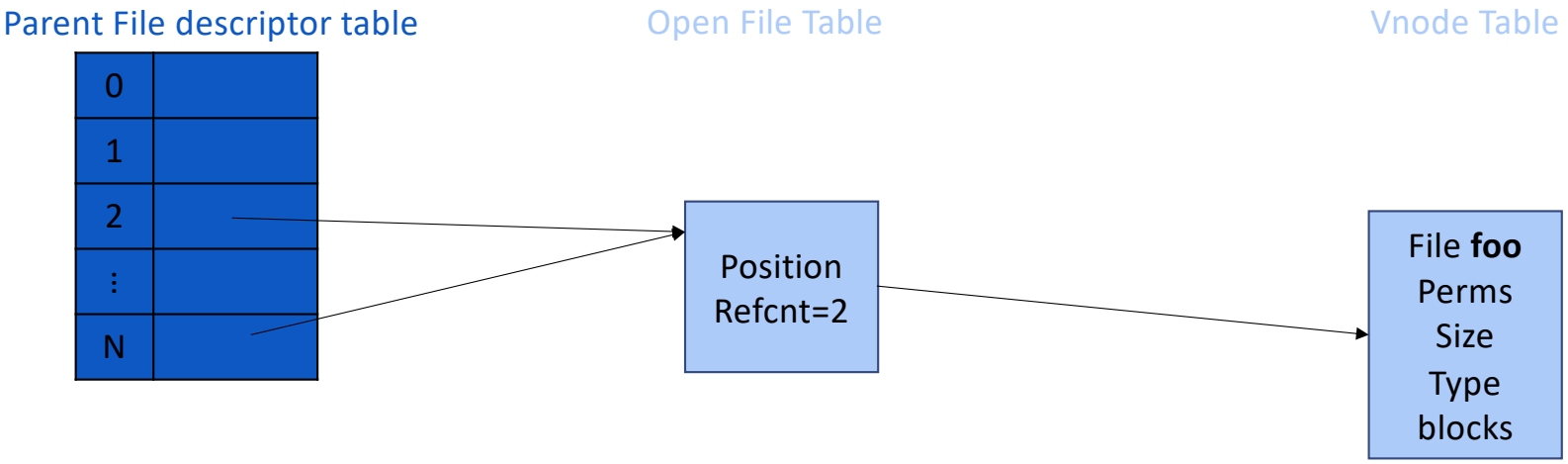
Two threads in P1 share an fd



Parent and Child have the same FD



Another way to get a refcount of 2



The dup (dup2) system call(s) "duplicates a file descriptor"

```
int dup(int fd);
int dup2(int fd1, int fd2);
```