8. Files and File Systems

- File Storage Structure
- File System Implementation
- Kernel Abstraction
- Communication Through a Pipe

Disk Operation

Disk Structure

- Plates, tracks, cylinders, sectors
- Multiple R/W heads
- Quantitative analysis
  - Moderate peak bandwidth in continuous data transfers
    E.g., up to 160MB/s on a modern SATA, 320MB/s on a modern SCSI
    Plus a read (and possibly write) cache in DRAM memory
  - Very high latency when moving to another track/cylinder
    A few milliseconds on average, slightly faster on SCSI

Block Granularity Access and Request Handling Algorithms

- Coarse-grain I/O: *block*-based I/O to amortize latency
- Scheduling
  - Queue pending requests and select them in a way that minimizes *head movement* and *idle plate rotation*
  - Variants of the *"elevator" algorithm*
  - Strong influence on process scheduling and preemption: *disk thrashing*

8. Files and File Systems

- File Storage Structure
- File System Implementation
- Kernel Abstraction
- Communication Through a Pipe

Storage Structure: Inode

Index Node

- UNIX distinguishes *file data* and *information about a file* (or meta-data)
- File information is stored in a structure called *inode*
- Attached to a particular device

Attributes

File Type
Number of hard links (they all share the same inode)
File length in bytes
Device identifier
User identifier (UID, file owner)
Group identifier (GID, user group of the file)
Timestamps: last status change (e.g., creation, modification, and access)
Access rights

Possibly more attributes, depending on the file system

Inode: Access Rights

- Classes of file accesses

  user: owner
  group: users who belong to the file's group, excluding the owner
  others: all remaining users

- Classes of access rights

  read: *directories: controls listing*
  write: *directories: controls file status changes*
  execute: *directories: controls searching (entering)*

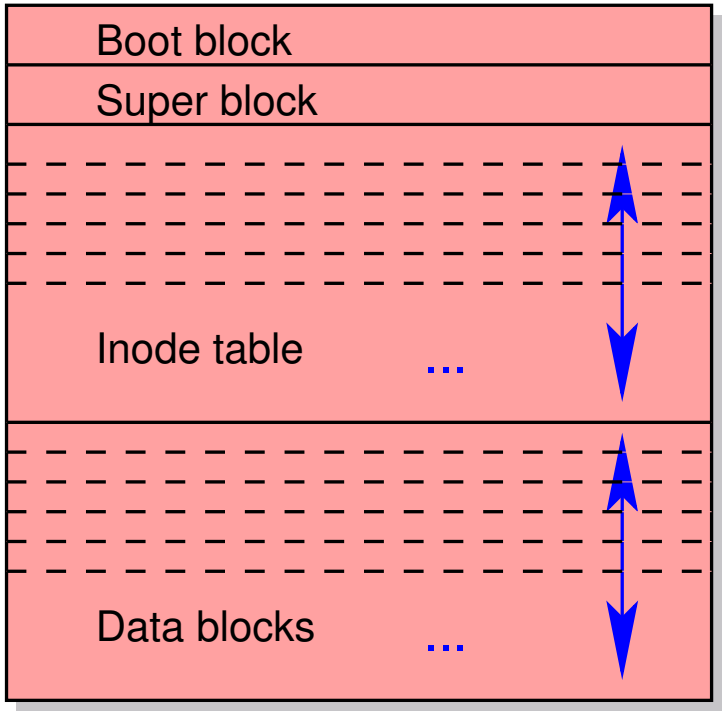
- Additional file modes

  suid: with **execute**, the process gets the file's UID
  directories: nothing
  sgid: with **execute**, the process gets the file's GID
  directories: created files inherit the creator process's GID
  sticky: (not portable)
  directories: files owned by others cannot be deleted or renamed

File System Storage

General Structure

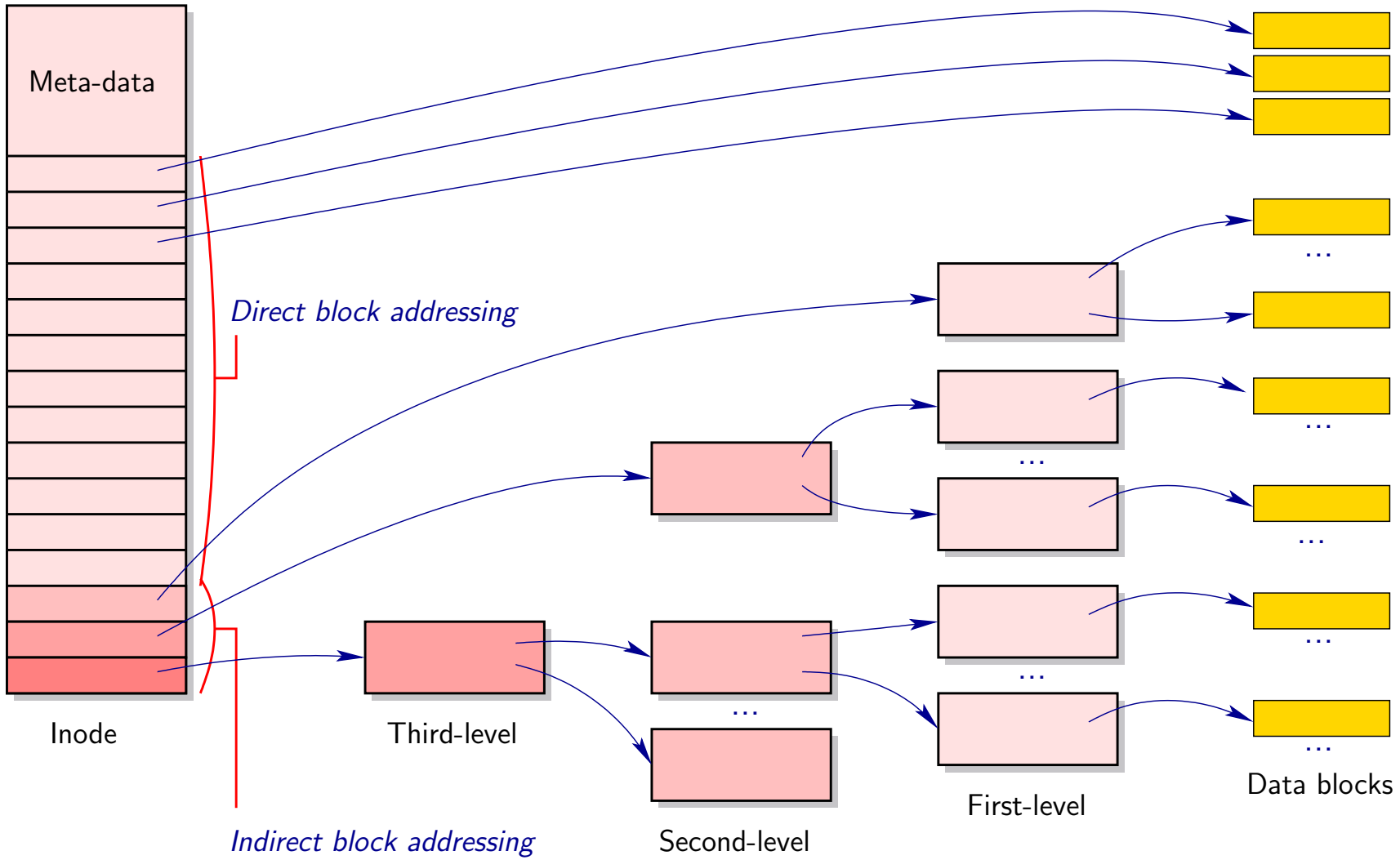
- Boot block
  - Boot flag (bootable or not)
  - Link to data blocks holding boot code
- Super block
  - File system status (mount point)
  - Number of allocated and free nodes
  - Link to lists of allocated and free nodes
- Inode table
- Data blocks
  - Note: directory = list of file names in a data block



Simplified file system layout

Inode: Data Block Addressing

- Every Inode has a table of block addresses
- Addressing: direct, one-level indirect, two-levels indirect, ...



File Meta-Data

Protection Modes for `chmod` Command

Mode	Octal value	Comment
u±rwx	00700	mask for file owner permissions
u±r	00400	owner has read permission
u±w	00200	owner has write permission
u±x	00100	owner has execute permission
g±rwx (or g±rwX)	00070	mask for group permissions
g±r	00040	group has read permission
g±w	00020	group has write permission
g±x (or g±X)	00010	group has execute permission
o±rwx	00007	mask for permissions for others
o±r	00004	others have read permission
o±w	00002	others have write permission
o±x (or o±X)	00001	others have execute permission
u±s	04000	SUID bit
g±s	02000	SGID bit
±t	01000	restricted deletion flag (sticky bit)

8. Files and File Systems

- File Storage Structure
- File System Implementation
- Kernel Abstraction
- Communication Through a Pipe

File Systems

Virtual File System

- Mounting multiple file systems under a common tree
  - `$ mount [options] device directory`
- Superset API for the features found in modern file systems
  - Software layer below system calls
  - Full support of UNIX file systems
  - Integration of pseudo file systems
    - `/proc`, `/sys`, `/dev`, `/dev/shm`, etc.
  - Supports "virtual" devices
    - Loopback devices: `/dev/loop0`, `/dev/loop1`... for encryption, compression, swap files, etc.
  - Support foreign and legacy file systems: FAT, NTFS, ISO9660, etc.

Modern File Systems

Features

- Transparent defragmentation
- Unbounded file name and size
- Minimize down-time with *journaling*
  - Maximal protection (default): support logging of all data and meta-data blocks
  - Minimal overhead: logging of meta-data blocks only
- Atomic (transactional) file operations
- Access control Lists (ACL)

Modern File Systems

Features

- Transparent defragmentation
- Unbounded file name and size
- Minimize down-time with *journaling*
  - Maximal protection (default): support logging of all data and meta-data blocks
  - Minimal overhead: logging of meta-data blocks only
- Atomic (transactional) file operations
- Access control Lists (ACL)

Notes About Linux EXT3

- Compatible with EXT2
- Journalization through a specific block device
- Use a hidden file for the log records

Modern File Systems

Features

- Transparent defragmentation
- Unbounded file name and size
- Minimize down-time with *journaling*
  - Maximal protection (default): support logging of all data and meta-data blocks
  - Minimal overhead: logging of meta-data blocks only
- Atomic (transactional) file operations
- Access control Lists (ACL)

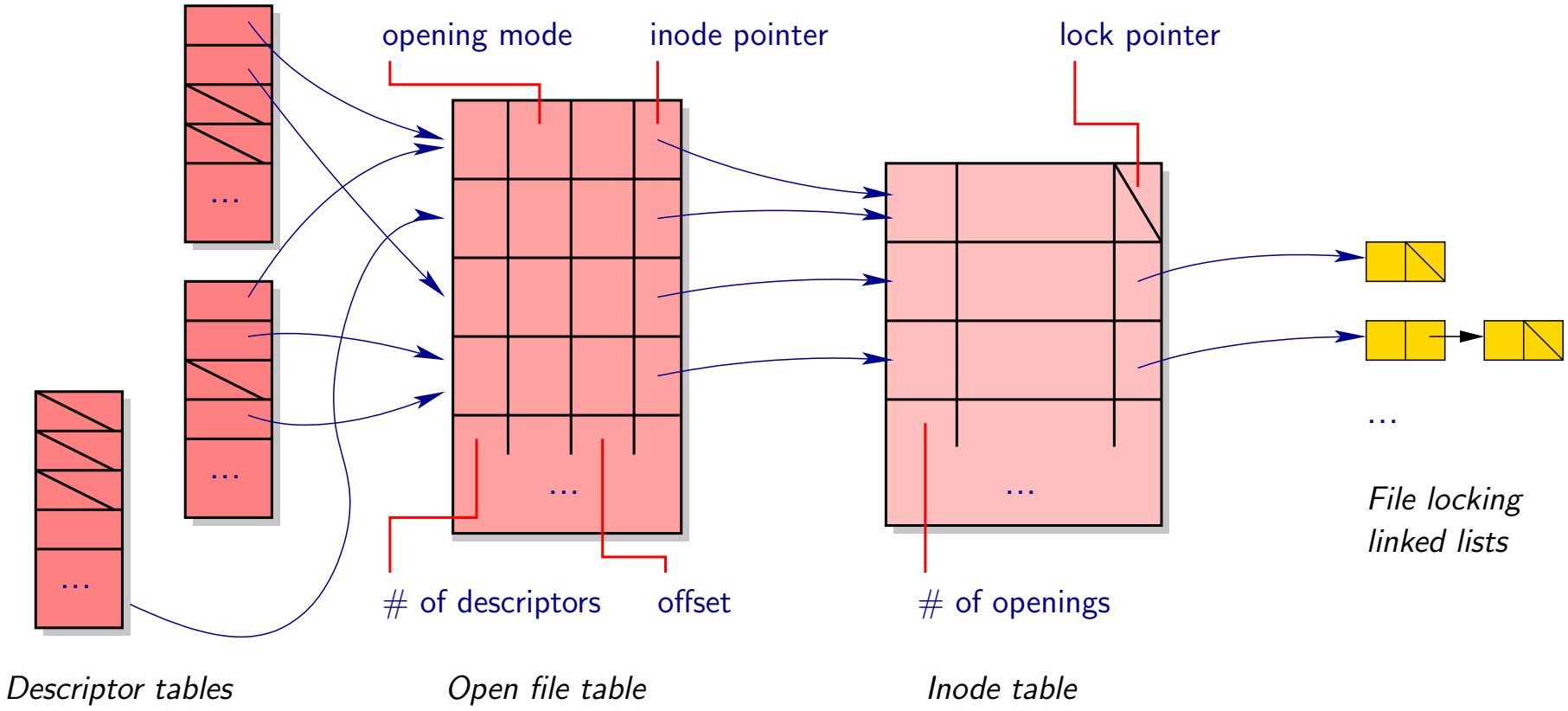
Notes About Windows NTFS

- Optimization for small files: "resident" data
- Direct integration of compression and encryption

8. Files and File Systems

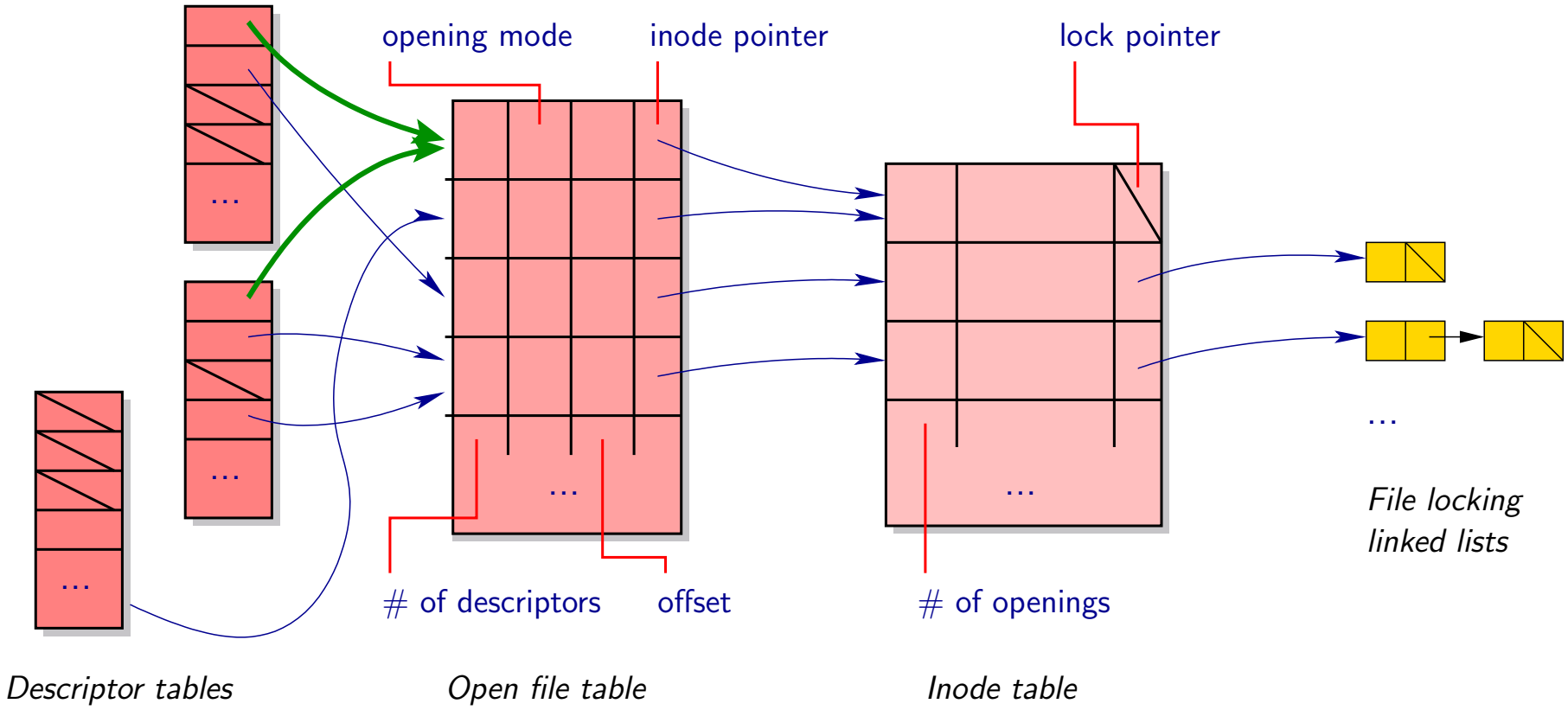
I/O Kernel Structures

- One table of *file descriptor*s per process:
  `System.in`, `System.out`, `System.err`
- Table of *open file*s (status, including opening mode and offset)
- Inode table (for all open files)
- File locks



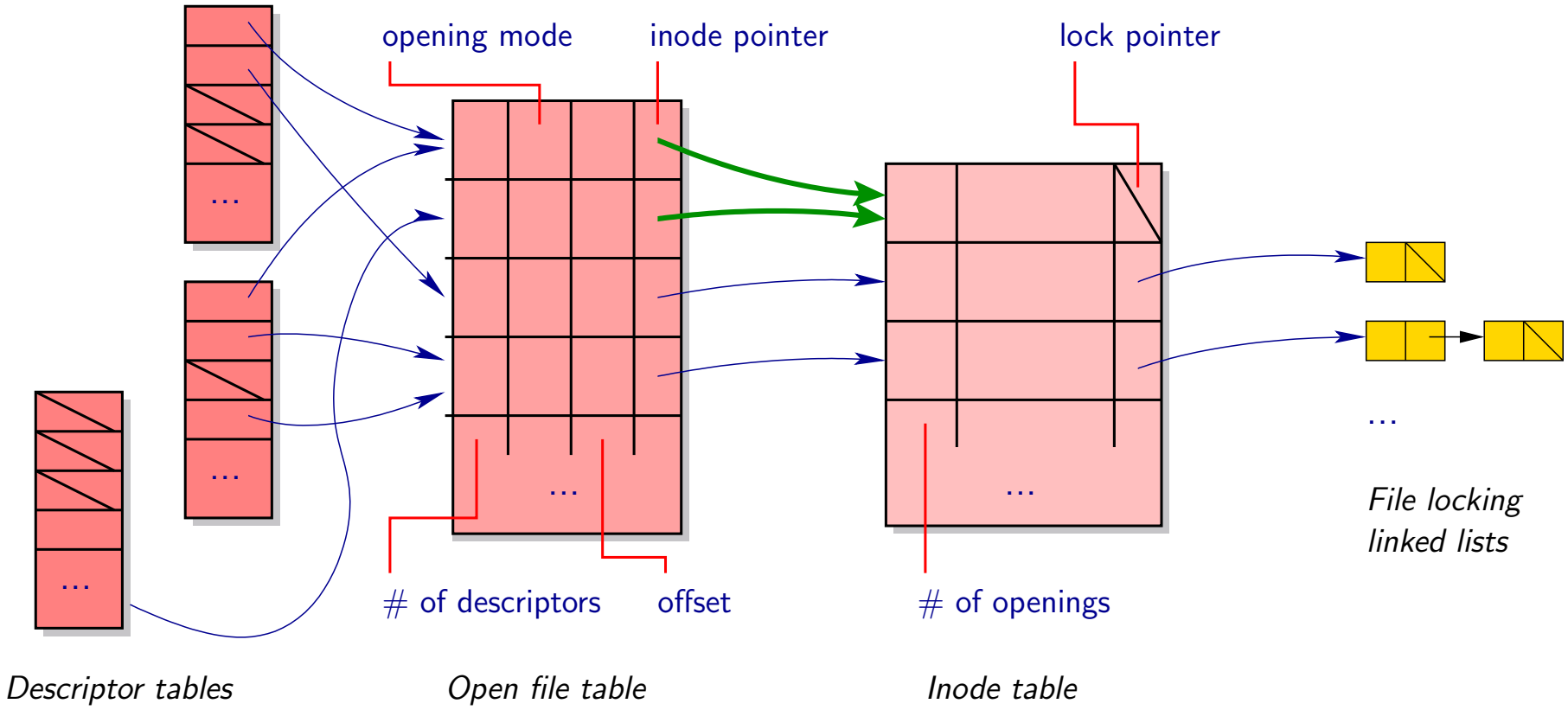
I/O Kernel Structures

- Example: file descriptor aliasing
  - E.g., process creation (e.g., via the `ProcessBuilder` class in Java)



I/O Kernel Structures

- Example: open file aliasing
  - E.g., multiple processes opening the same file



File Permissions

Process and File Permissions

- Every file I/O is conditioned to read/write/execute access rights, relative to UID (user) and GID (group)

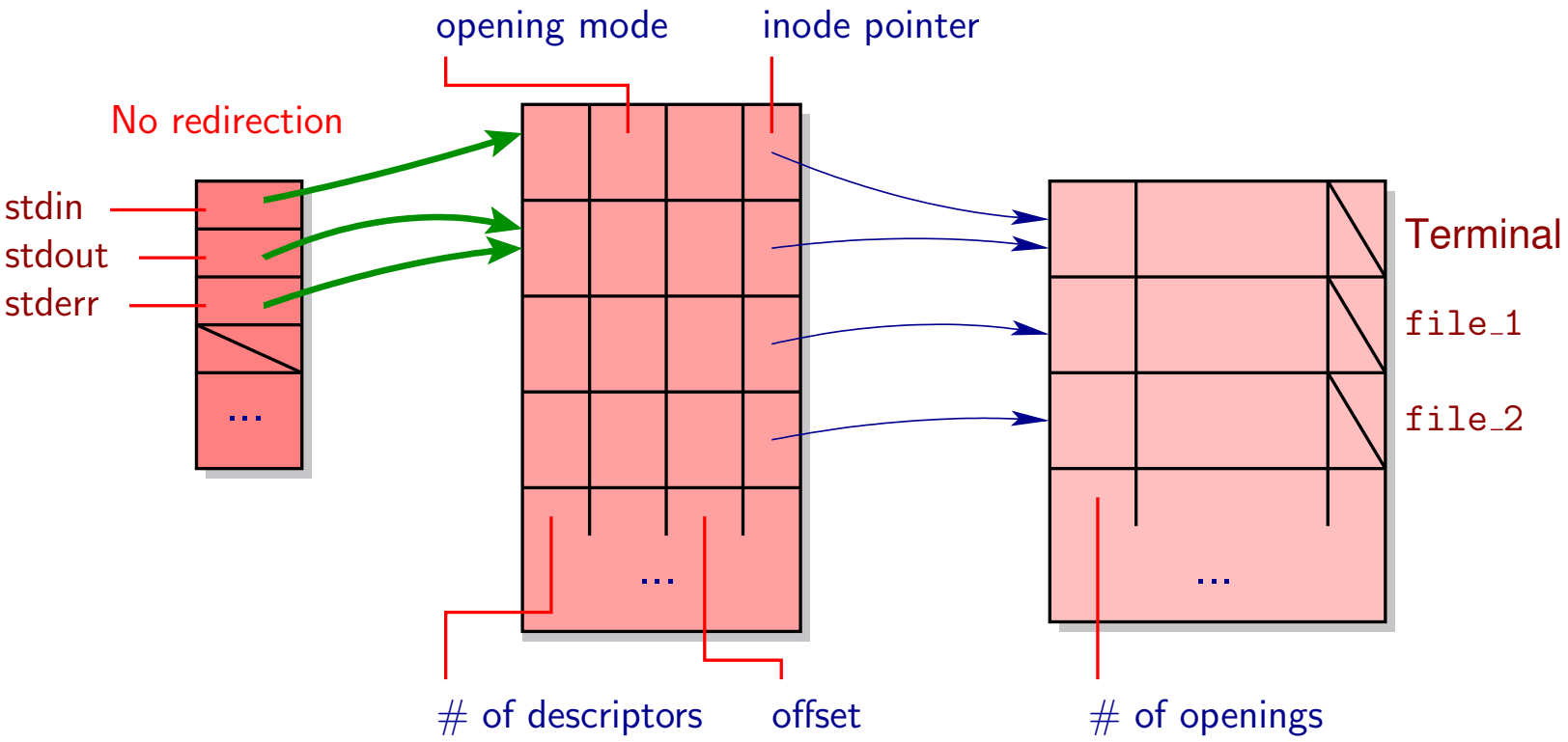
Java Interface

- `File` class
  - `checkRead()`, `checkWrite()`, `checkExec()`, etc.
  - `checkPermission()`
- `FilePermission` class
  - More precise control and interaction with security policies

Application: I/O Redirection

Example

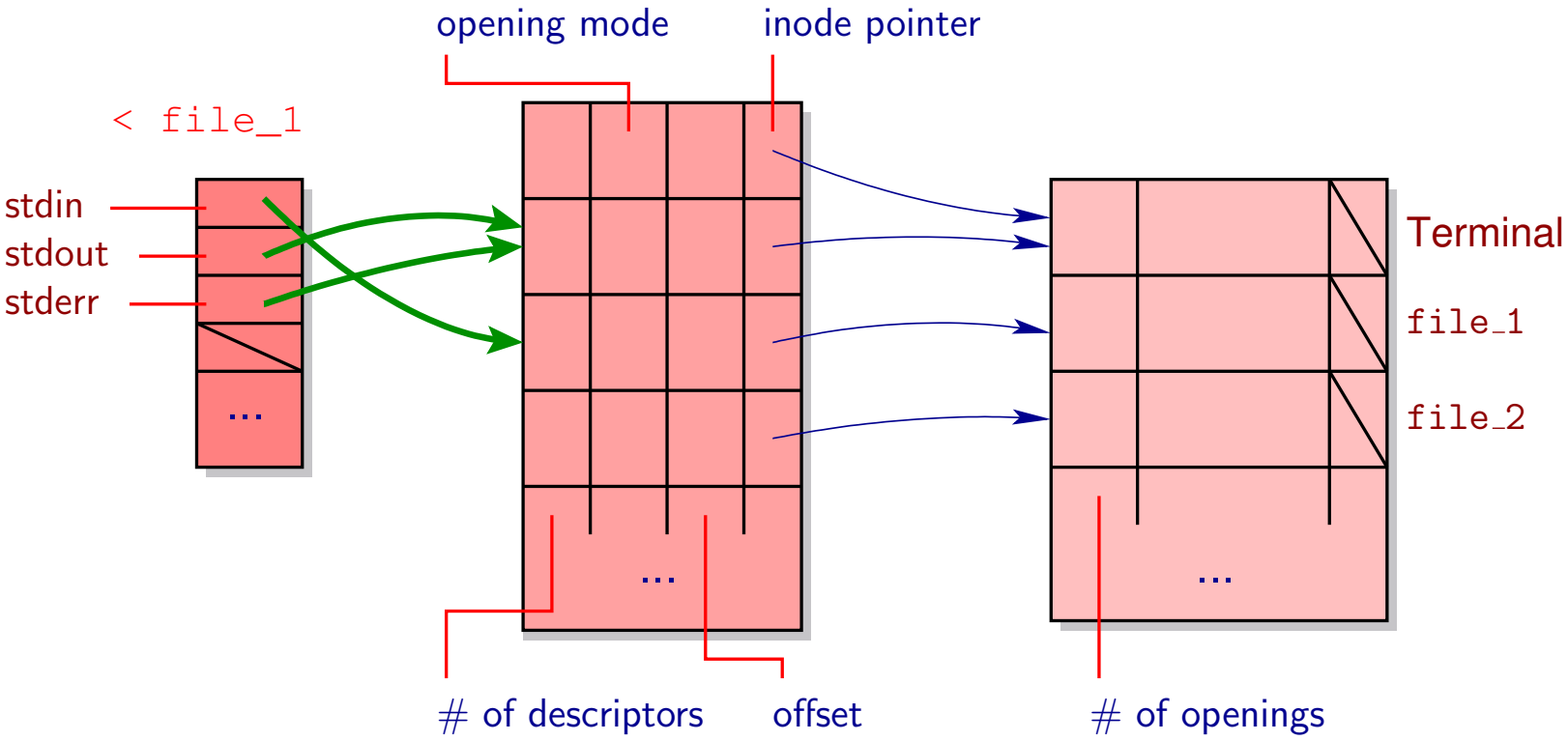
No redirection



Application: I/O Redirection

Example

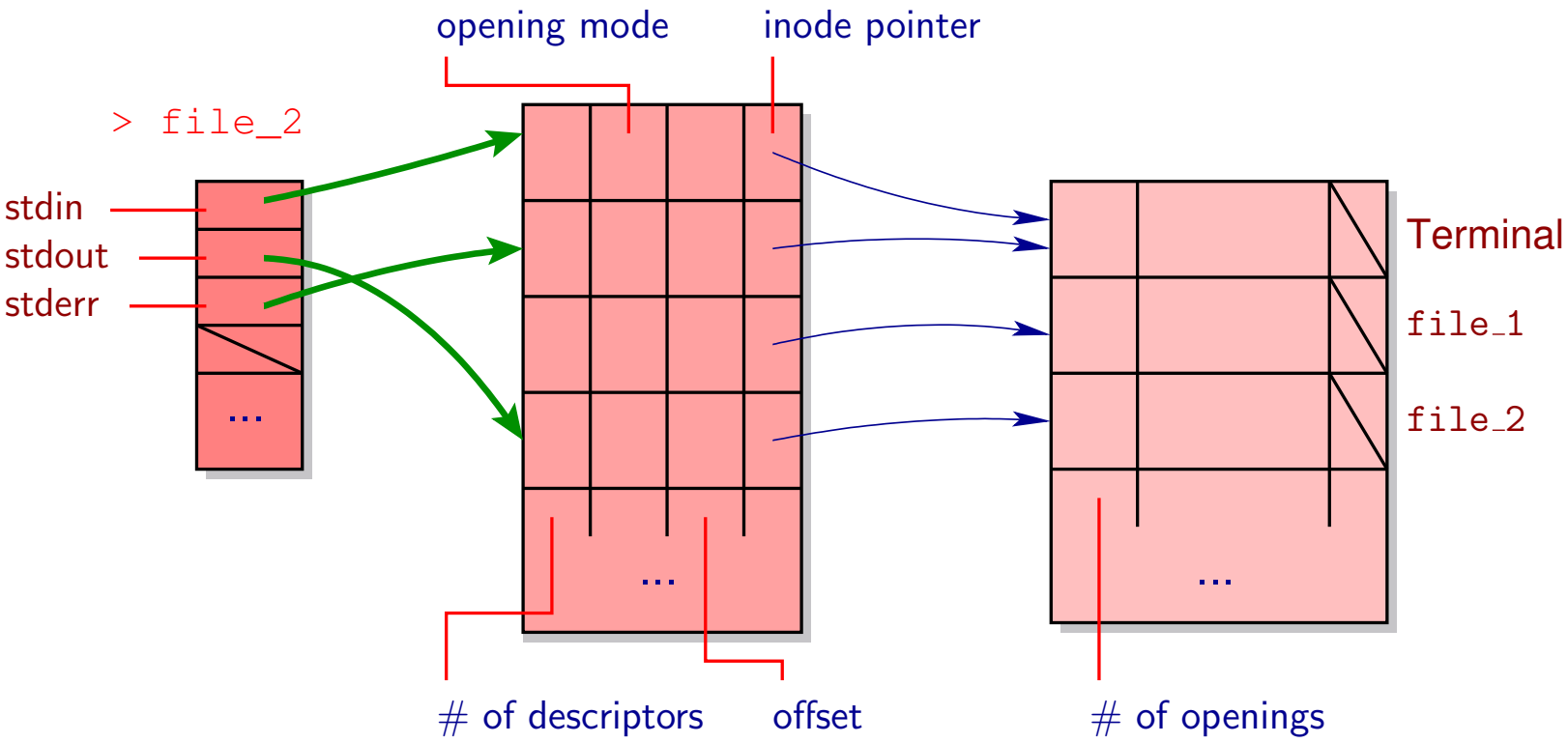
Standard input redirection: `System.setIn()`



Application: I/O Redirection

Example

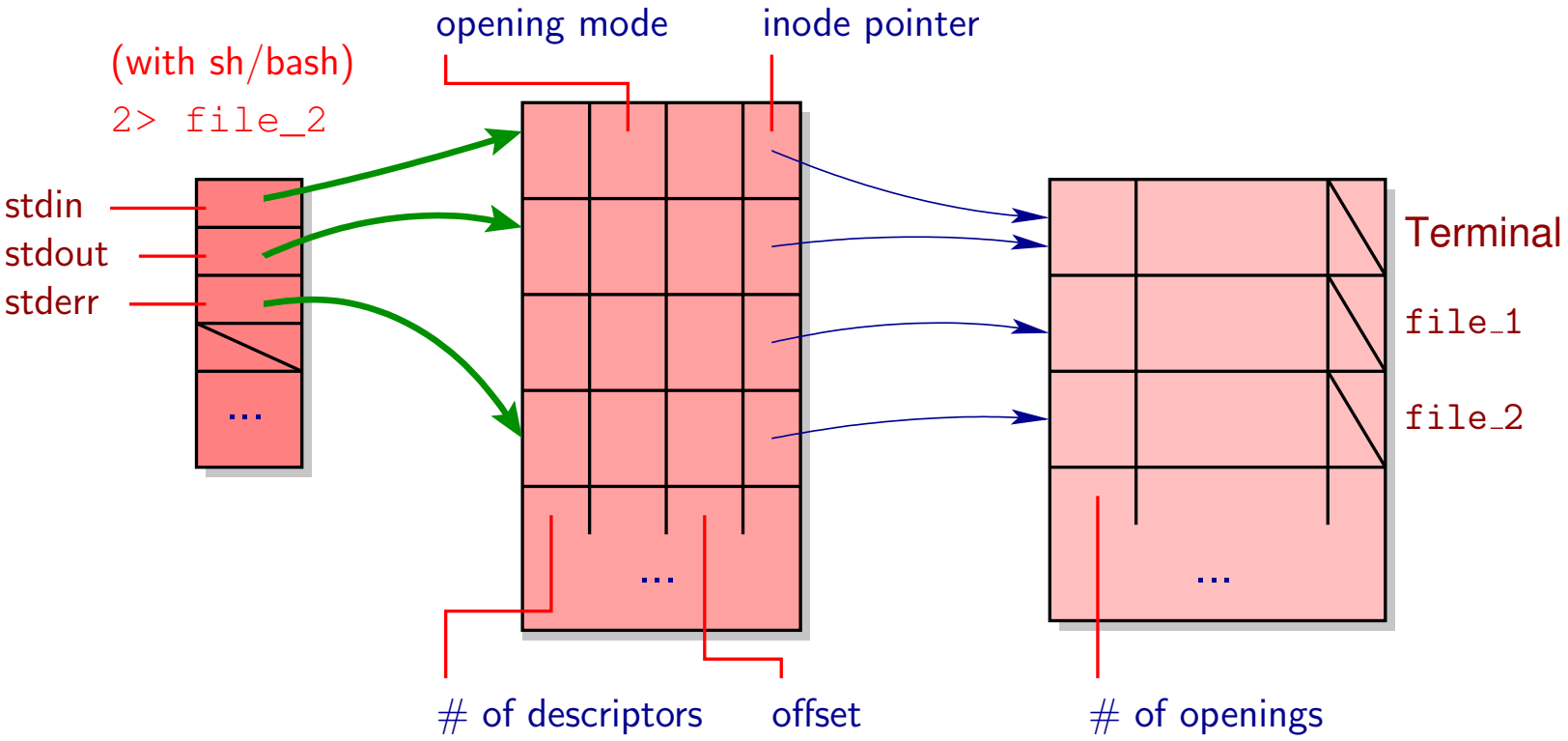
Standard output redirection: `System.setOut()`



Application: I/O Redirection

Example

Standard error redirection: `System.setErr()`



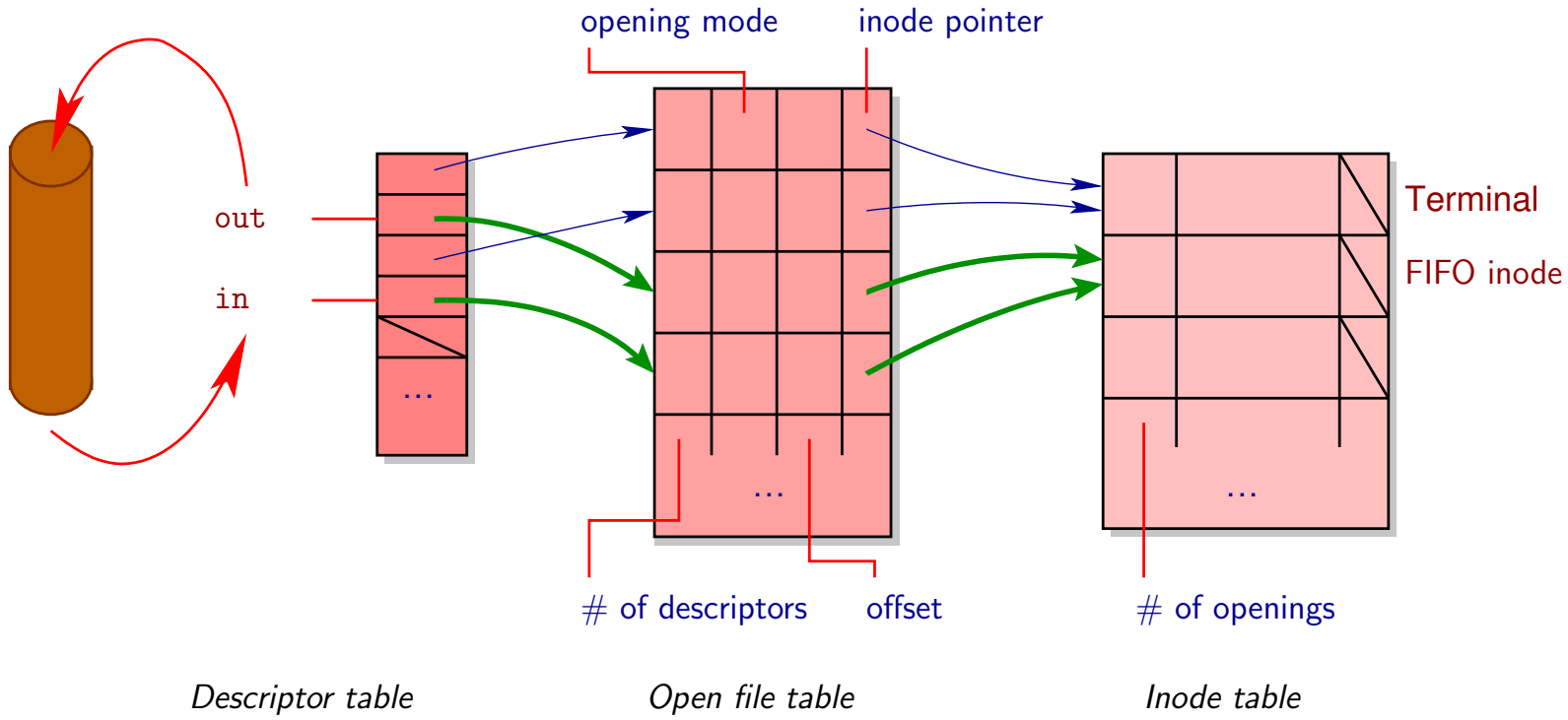
8. Files and File Systems

- File Storage Structure
- File System Implementation
- Kernel Abstraction
- Communication Through a Pipe

FIFO (Pipe)

Principles

- Channel to stream data among processes
  - Data traverses the pipe *first-in* (write) *first-out* (read)
  - Blocking read and write by default (bounded capacity)
  - Illegal to write into a pipe without reader (delivers UNIX signal `PIPE` = **13**)
  - A pipe without writer simulates *end-of-file*



FIFOs and I/O Redirection

Question

Implement `$ ls | more`

Solution

- `new PipeInputStream()`
- `new PipeOutputStream()`
- Process to become `ls`
  - `System.setOut()`
  - `new ProcessBuilder().start()` (which calls `getRuntime().exec()` on `"ls"`)
- Process to become `more`
  - `System.setIn()`
  - `new ProcessBuilder().start()` (which calls `getRuntime().exec()` on `"more"`)