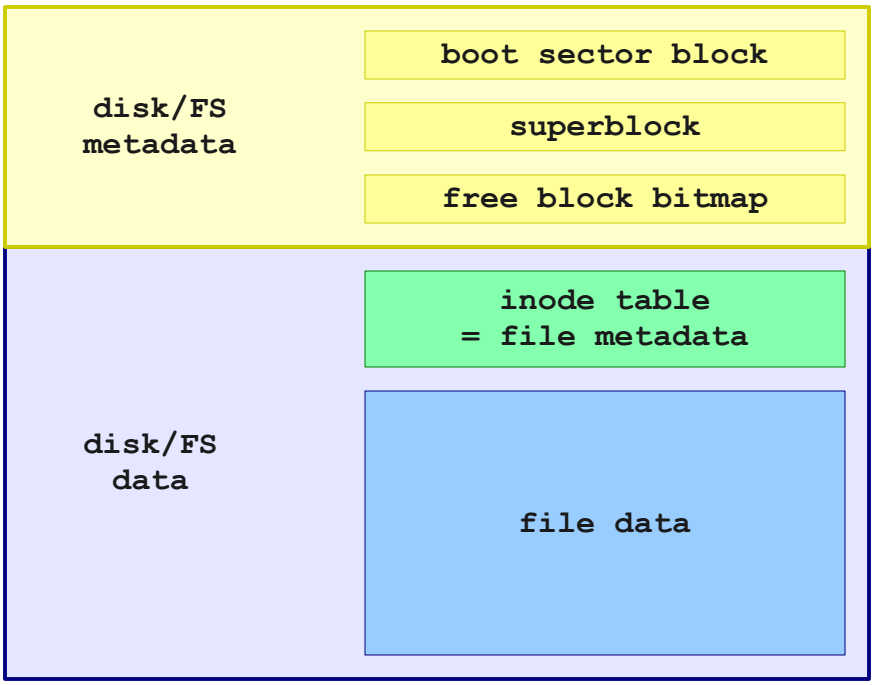
What do file systems do?

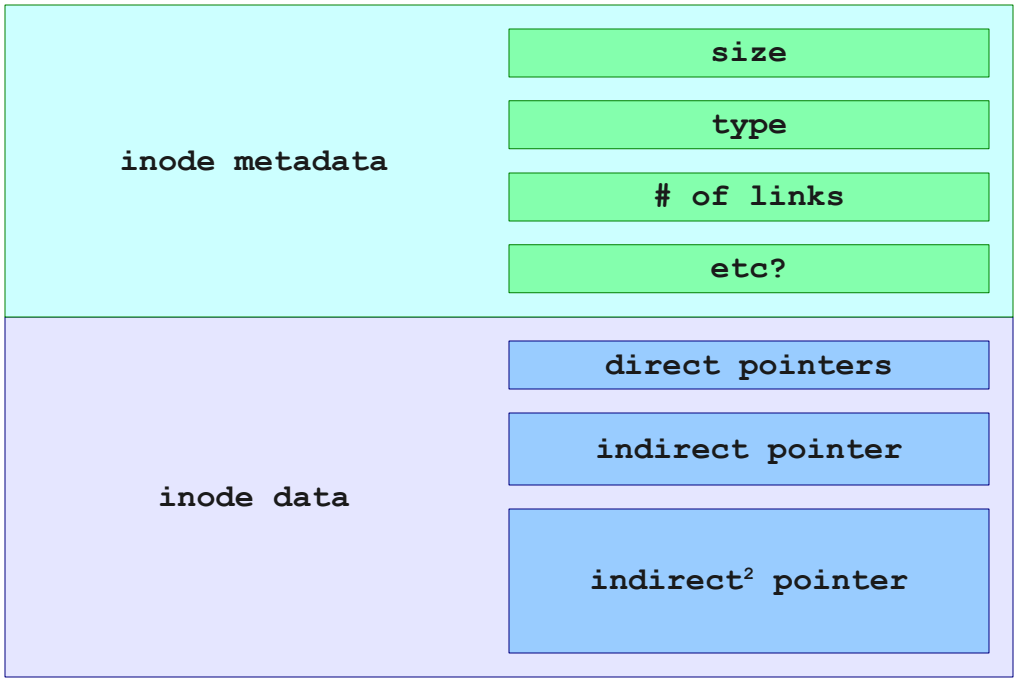
- **Organize storage (and devices)**
- **Manages storage to implement the file interface**
- **Gives names to files to be easy to access**

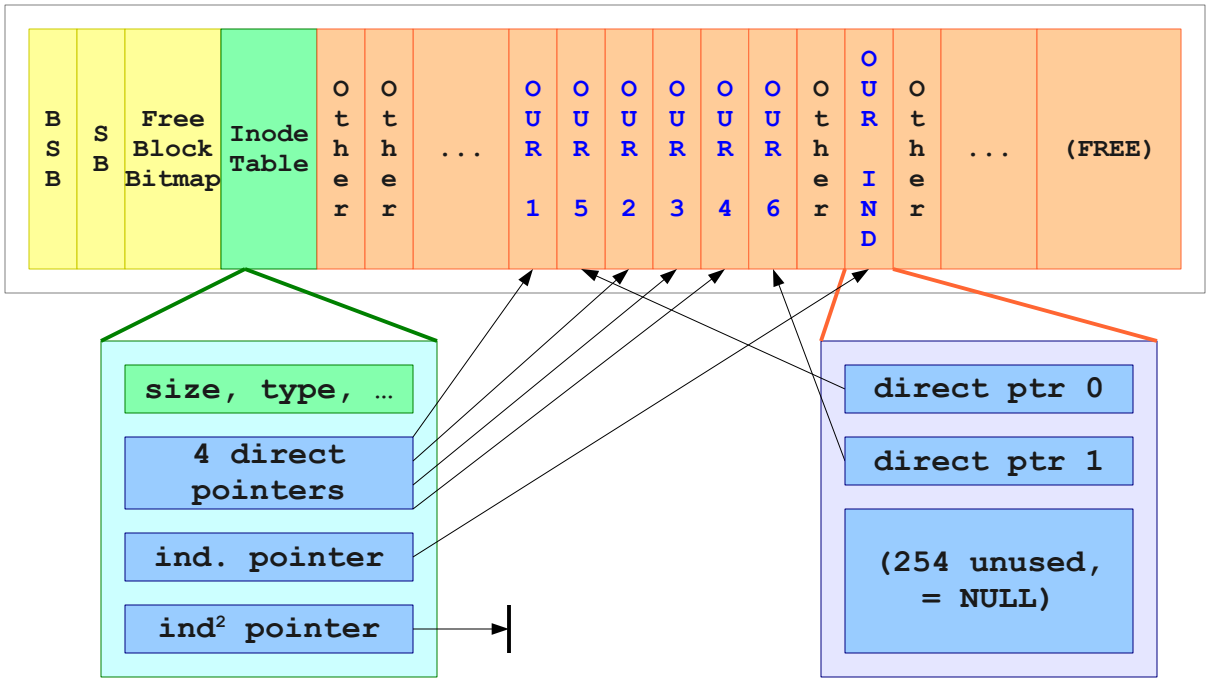
Why do we need them?

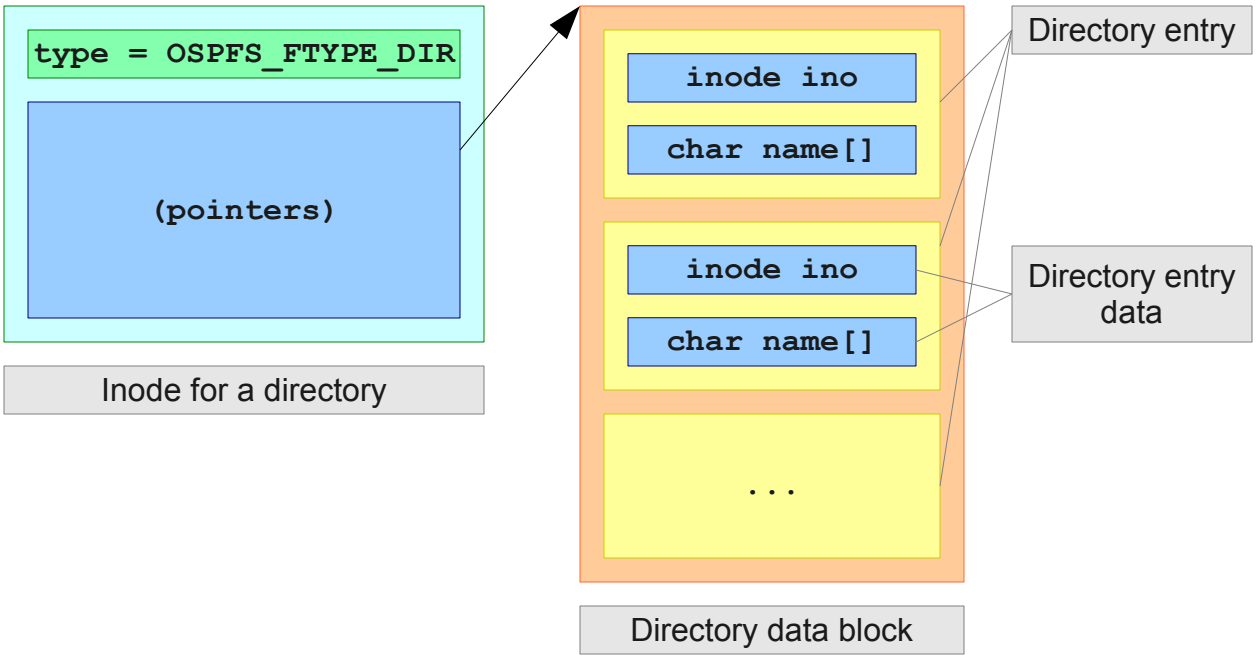
- **Difficult to deal with sectors directly or manually**
- **Difficult to grow and shrink space directly**
- **Difficult to remember sector names**

⇒ Abstracts storage to be usable by humans


disk/FS
metadata
boot sector block
superblock
free block bitmap
disk/FS
data
inode table
= file metadata
file data


inode metadata
size
type
# of links
etc?
inode data
direct pointers
indirect pointer
indirect² pointer


BSB
SB
Free Block Bitmap
Inode Table
Other
Other
...
OUR 1
OUR 5
OUR 2
OUR 3
OUR 4
OUR 6
Other
OUR IND
Other
...
(FREE)
size, type, …
4 direct pointers
ind. pointer
ind² pointer
direct ptr 0
direct ptr 1
(254 unused, = NULL)


type = OSPFS_FTYPE_DIR
(pointers)
Inode for a directory
inode ino
char name[]
inode ino
char name[]
...
Directory data block
Directory entry
Directory entry data

File system level: superblock

- os_magic # marker
- os_nblocks # blocks
- os_ninodes # inodes
- os_firstinob # offset first inode block

File level: inode (regular files, dirs)

- oi_size file size
- oi_ftype file, dir, symlink
- oi_nlink # of hardlinks
- oi_mode file permissions

(dir, ind, ind^2 block pointers)

– Often useful to make a file available in multiple directories

  – For example, I have my .emacs in a dir that I back up

  – Also useful for sharing code between projects

    – For example,
    project FOO has write access to com/example/FOO
    and BAR wants to use com/example/FOO/FooClass.java
    that project FOO maintains

Issues with duplication:

1. Wasted space
2. Updates of multiple copies

Solution: links

Hard links: every inode has an nlinks counter

- **When file is first created, set to 1**
- **Increment whenever we link from another location**
- **Decrement whenever we delete a filename for that inode**
  - **When nlinks = 0, can free disk space**

Q: can't (in ext4, anyway) hard link a directory—why not?

symlinks:

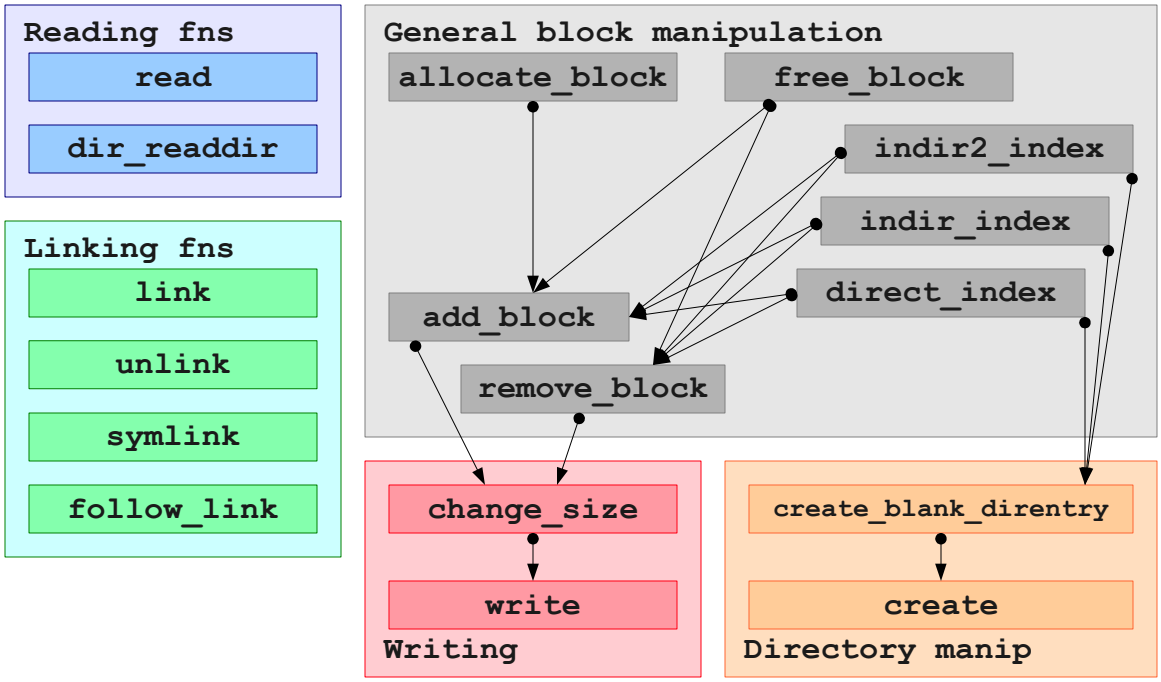
- **Contains a pathname**
- **When we open the symlink:**
  - **The OS looks up the pathname**
  - **If valid, it opens the file at that pathname**
  - **If we move, delete, rename the referenced file, we must manually update the symlink**

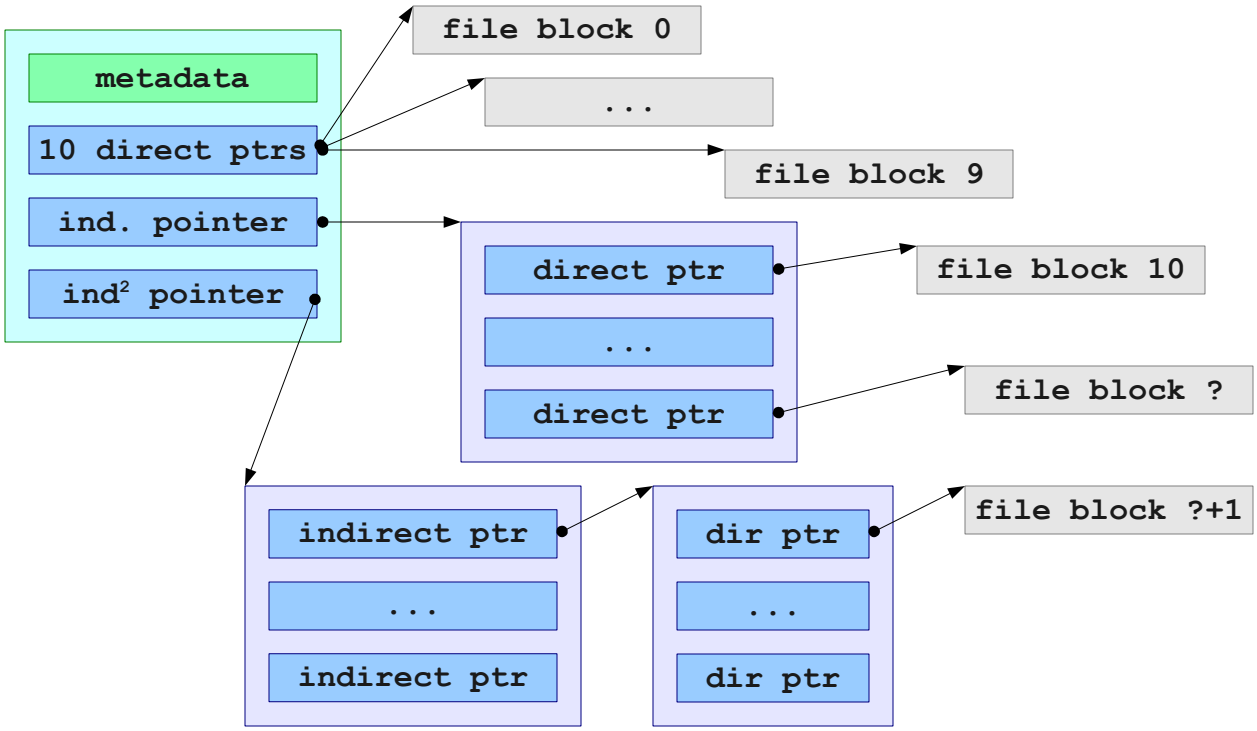
    Q: It's okay to symlink directories—why?
- **When we delete a symlink, we delete the link, but not the linked file!**

Loose C++ analogy: hard links = references, symlinks = pointers

Filesystems don't need to be just interfaces to disks:

- `/dev`
  - devices on the system
  - `/dev/urandom`: (pseudo) random byte pool
  - `/dev/stdout`: the standard output
    - symlinked to `/proc/self/fd/1`
- `/proc`
  - interface to kernel objects
  - `/proc/cpuinfo`
    - info about the CPUs, including bogomips


Reading fns
read
dir_readdir
Linking fns
link
unlink
symlink
follow_link
General block manipulation
allocate_block
free_block
indir2_index
indir_index
direct_index
add_block
remove_block
change_size
write
Writing
create_blank_direntry
create
Directory manip


metadata
10 direct ptrs
ind. pointer
ind² pointer
file block 0
...
file block 9
direct ptr
...
direct ptr
file block 10
file block ?
indirect ptr
...
indirect ptr
dir ptr
...
dir ptr
file block ?+1