ECE 650
Systems Programming & Engineering

Spring 2018

File Systems

Tyler Bletsch
Duke University

Slides are adapted from Brian Rogers (Duke)

File Systems

- Disks can do two things: **read_block** and **write_block**
- We want better interface, e.g. files and directories:
  open, **read**, **write**, **close**, **mkdir**, **rm**, etc.

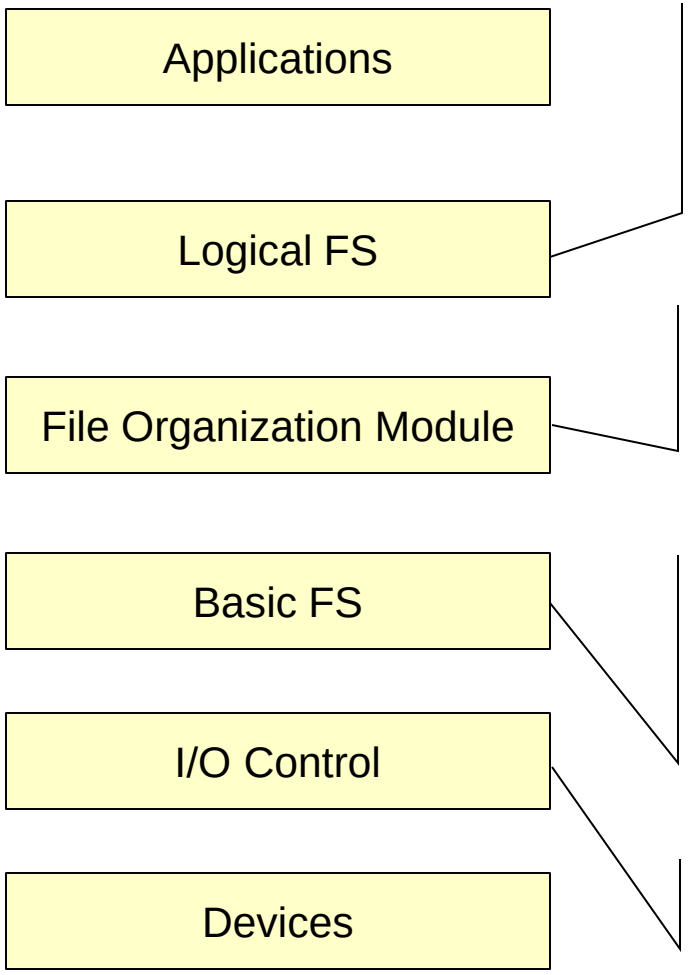
- Filesystem is what does this (abbreviated FS in these slides)

- FS allows easy access by applications to disk storage
  - Two main aspects of a FS:
    - What should the interface to the user be?
      - E.g. File attributes, allowed file operations, directory structure
    - What algorithms & data structures to map logical files to devices?

Hard Disk Properties

- We should understand conceptual basics for FS topics
- Can be rewritten in place
  - E.g. read, modify, write to update data at one location
  - Unlike, say, flash storage
- Easy access both sequentially and randomly
  - Rotate disks and move disk read/write heads to right location
- Addressed as single-dimension array of logical blocks
  - Usually 512B; unit of size for disk I/O transfers
- Disk organization
  - Multiple platters; disk arm has read/write heads above each platter
  - Platters divided into tracks; tracks into sectors
  - Set of tracks at a particular arm position form a cylinder
- Can convert logical block number into a physical disk location:
  - Cylinder #, track number within the cylinder, sector number within the track
  - In reality, this is complicated (e.g. by bad sectors)

FS Abstractions



- Manages FS meta-data
- Everything except for file contents
- Converts file name to logical block address
- Keeps file control block (e.g. inode) w/ file info

- Translates logical block address to physical
- Implements file allocation policy(ies)
- Tracks storage blocks & manages free space

- Can accept generic file commands
- Issues commands to appropriate device drivers
- Manages memory buffers that cache FS pieces
- E.g. directory & data blocks

- Device drivers; interrupt mechanism
- Takes requests & writes control bits to devices

File Basics

- File is named collection of data on secondary storage
- Users only interact w/ secondary storage through files
- Can represent many different types of information
  - Executable programs
  - Databases
  - Spreadsheets, word processing documents, text files
- Organization of information in a file depends on its type
  - E.g. text file vs. object file vs. executable file

File Basics (2)

- Attributes
  - Name, ID (unique number within the file system), type, location on storage device, size, access control protection
- Operations
  - Create, read, write, seek, delete
- File operations require finding the file
  - Files typically found by searching a “directory” of file names
    - Directory entry for a file name will point to its disk location
  - OS optimizes this by keeping an open-file table
    - With information about all open files
  - After a file is opened, it can be reference by an ID
    - E.g. a file descriptor
    - Points to location in open file table

File System Directory

- Symbol table used to manage system files
  - Stores meta-data about the file
    - Name, disk location, file type, etc.
  - When files are opened, searched for, created, deleted, renamed, or directories are traversed, we use the directory
  - Directory organization:
    - Single-level: all files must have a distinct name
    - Two-level: e.g. a file directory per user, with user files inside
    - Tree:
      - What we are familiar with from most OSes
      - Real file name is file name + path through directory tree to the file

Directory Implementation

- Need to map from file location to device storage block
  - Has many implications
    - Device efficiency
    - Performance
    - Reliability
- Map a file name to pointers to the file data blocks
- What kind of data structure to use?
  - List
  - Hash Table

Directory List Implementation

- List of data structures
- Data structure contains at least:
  - File name, pointers to data blocks on disk
  - We will talk more about how to organize these pointers in a bit
- Simple, but inefficient
  - Finding a file requires a linear search of all list entries
  - Same for creating a file
    - If not found, add a new entry to end of list
  - Same for deleting a file
    - Can have an extra bit or marker file name for “free” list entries
    - Or keep a separate list of free list entries (a free list)

Directory List Example

Free List

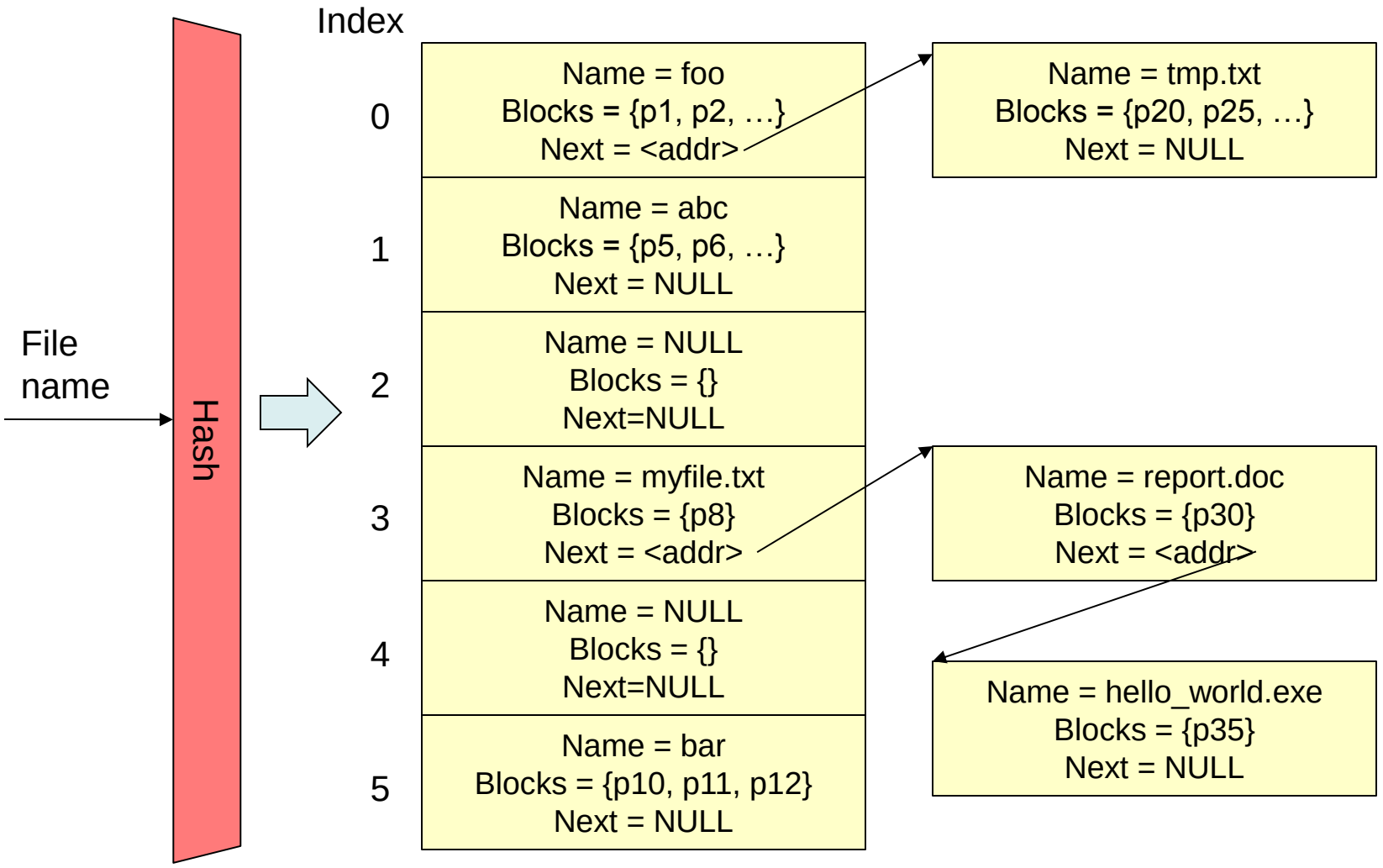
2
4

Index	
0	Name = foo Blocks = {p1, p2, …}
1	Name = abc Blocks = {p5, p6, …}
2	Name = NULL Blocks = {}
3	Name = myfile.txt Blocks = {p8}
4	Name = NULL Blocks = {}
5	Name = bar Blocks = {p10, p11, p12}

Hash Table Implementation

- Again, a list (table) of directory entries
  - But list index for a file is determined via a hash of the file name
- Improves efficiency
  - Finding a file is straightforward
  - Creating and deleting a file are constant time
- Extra complexity for handling collisions
  - What if we only have a list of 64 entries, but 65 files?
  - Multiple file names may hash to same entry
  - Can utilize a chain of directory entries at each entry of the table
    - Hybrid of List + Hash Table implementations
    - Finding a file requires: 1) hash calculation + 2) small list search

Hash Table Example



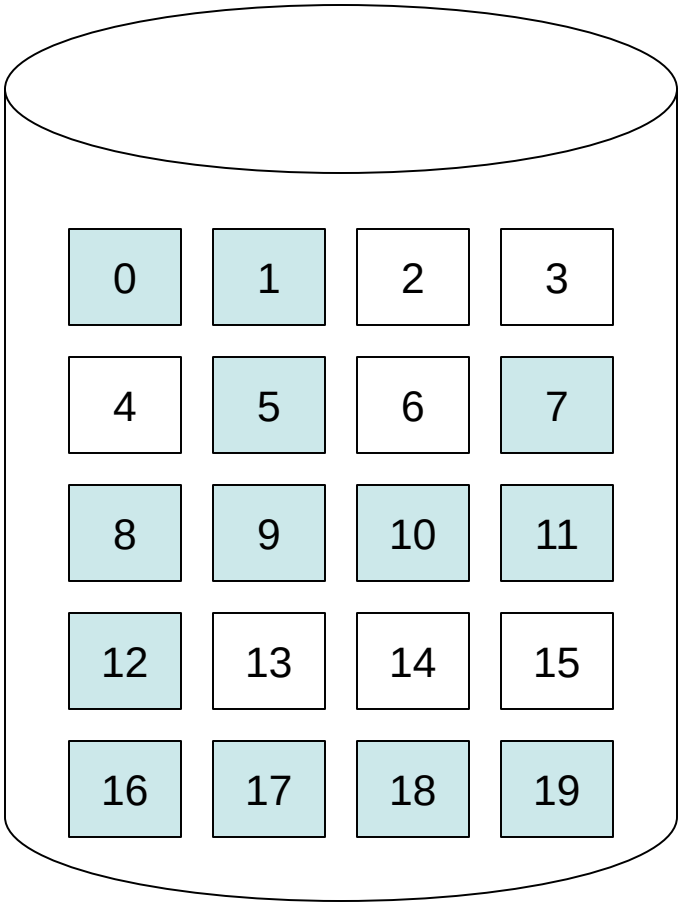
Disk Allocation

- Need to allocate space for files on disk
- Want to utilize the disk effectively
  - E.g. minimize fragmentation, minimize seek times for reading files
- Common approaches
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation
- Different approaches may be used by different FS'es
- Thus, OS may support multiple approaches for different FS types

Contiguous Allocation

- Each file occupies a sequential set of blocks on disk
  - For file requiring N blocks, its blocks are:
    - j,j+1, j+2, j+3, … , j+N
- Requires minimal disk activity for reading the file
  - Disk rotation to read blocks from sectors within a track
  - Read/write head only moves to next track after reading last sector of current track
- Directory entry for each file is very simple:
  - Starting block number on disk + length of file
- Both sequential and random access is easy:
  - FS remembers current location in file and advances automatically
  - To access block “b”, can compute j+b

Contiguous Allocation Example



File Name	Start Block	Size
foo	0	2
notes.txt	5	1
report.doc	7	6
hello_world	16	4

Drawbacks of Contiguous Allocation

- Finding free blocks for a new file is complicated
  - Described in detail in later charts
  - We've studied a similar problem already (dynamic memory)
    - Search "free" blocks: first fit, best fit, worst fit
    - **External fragmentation** as blocks are alloc'd & free'd
  - Often, some form of defragmentation is done
    - Either periodically off-line, or regularly on-line
- Not easy to deal with growing / shrinking files
  - When creating a file, how much space to request on disk?
    - Too little? File runs out of space; Too much? **Internal fragmentation**
  - Some OSes use mechanism known as **extent** to handle this
    - If a file fills up its space, an extent (new set of blocks) is allocated
    - File directory stores location + size, as well as pointer to extent

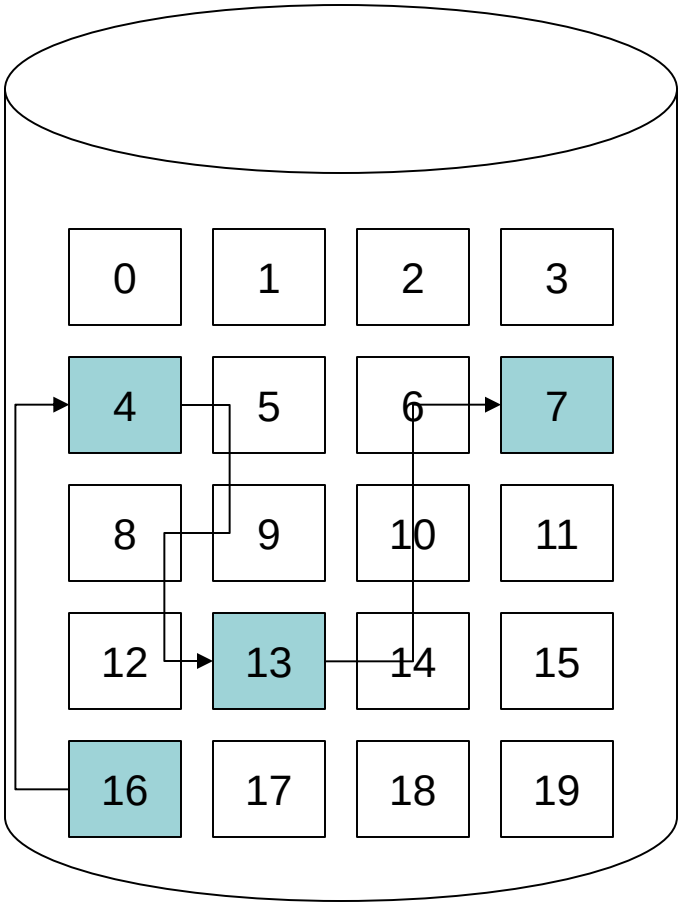
Linked Allocation

- Addresses drawbacks of contiguous allocation
- File occupies a linked list of disk blocks
- Blocks of a single file may be located anywhere on disk
- Data Structures
  - Directory stores block pointer to first and last blocks
  - Each block stores a pointer to next block location
    - Pointer is not available to user

Linked Allocation Operation

- Create file
  - Create a new directory entry
    - Pointer to first block of file; size set to 0
    - File writes allocate a new block; add block to end of file list
- Advantages
  - No external fragmentation (no need to compact disk space)
  - No need to know file size at file creation time

Linked Allocation Example



File Name	Start Block	End block
hello_world	16	7

Drawbacks of Linked Allocation

- Random file access is inefficient
  - To read data from “i”th block:
    - Must always start at beginning and read from “i” blocks
- Sequential file access is “ok”
  - But more disk seeks usually required as file is read
- Some disk space overhead is required for the pointers
  - One pointer (e.g. 4 or 8 bytes) per 512 byte block
  - Can group multiple blocks into a cluster and allocate clusters
    - Improves overhead and sequential access performance

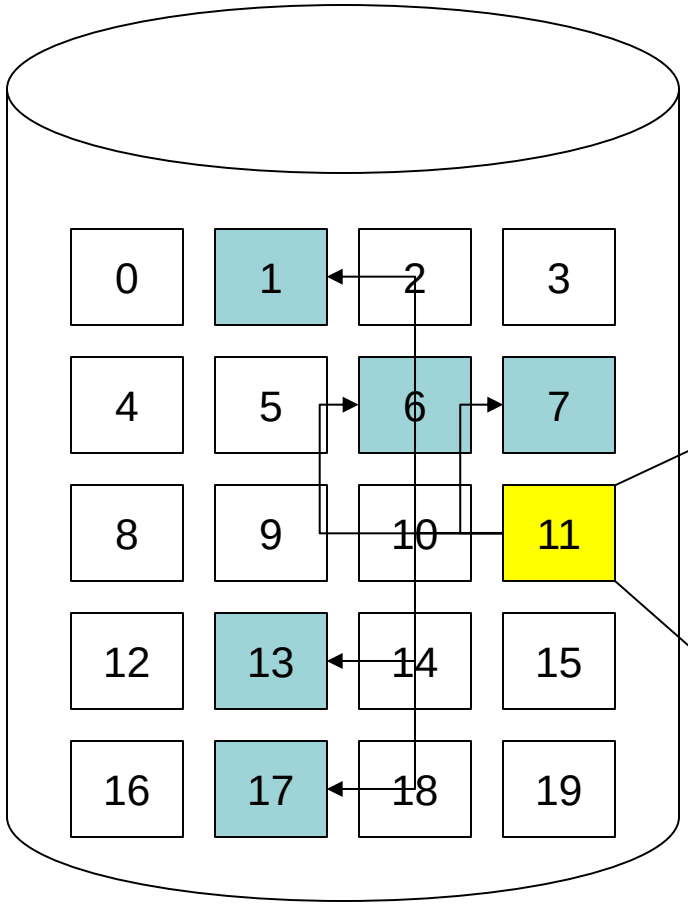
Example of Linked Allocation Variant

- File-allocation Table (FAT) file system
  - Used by MS-DOS and OS/2
- Disk space at the beginning of a volume is reserved
  - Used to store a file allocation table (FAT)
  - One entry per disk block (indexed by block number)
  - Directory entry for a file contains pointer to start block in FAT
    - Each entry in the FAT stores a pointer to the next entry for the file
  - Essentially, group all of the block pointers together
- Cache the FAT (or parts of it) in memory
  - Can improve random file access behavior
    - Eliminates disk accesses to identify file blocks

Indexed Allocation

- Solves the random access problem of linked allocation
- Aggregates block pointers together in an **index block**
- File has an index block
  - List of pointers to the file blocks
  - “i”th index block pointer points to the “i”th block of the file
- On file creation
  - Index block is allocated; all pointers set to NULL
- On file write (if a new block is needed)
  - Obtain block from free space manager
  - Stores block address in the next NULL index block entry

Indexed Allocation Example



File Name	Index Block
hello_world	11

11

1
6
7
13
17
-1
-1
-1

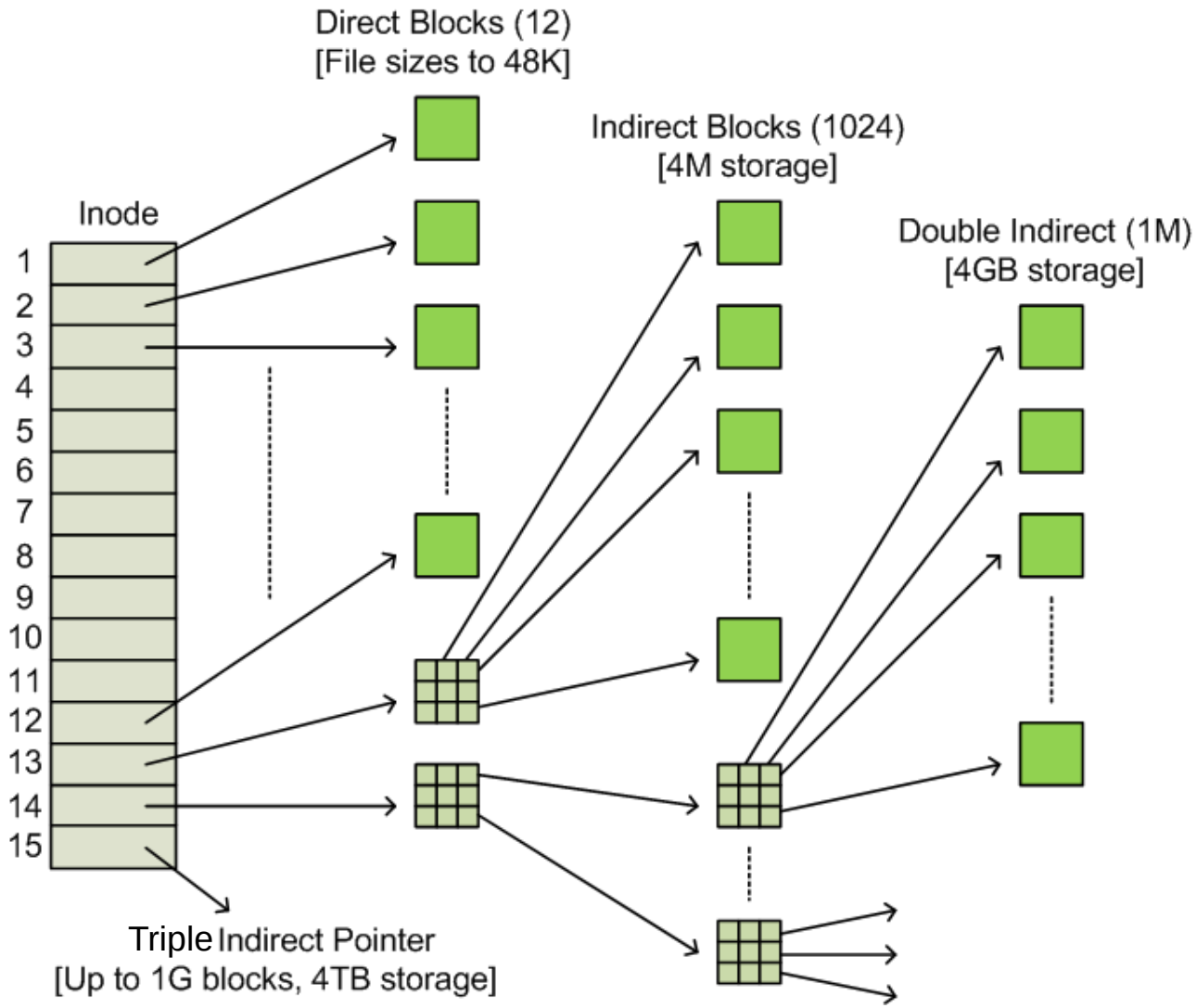
Drawbacks of Indexed Allocation

- A bit of extra wasted space compared to Linked
  - What if file does not use as many blocks as index node holds?
- Size of index block (what if file becomes too big)?
  - Linked index blocks
    - Last pointer of index blocks points to next index block
  - Multi-level index
    - First level index block points to second-level index blocks
    - Second-level index blocks point to file data disk blocks

Hybrid Indexed Allocation Scheme

- UNIX uses a combined implementation
  - And ext3 file system used frequently in Linux
- Directory entry data structure is called an inode
- inode has several fields
  - File mode, owners, timestamps, size (block count)
  - Index block of 15 pointers
    - First 12 point directly to file data blocks
    - One singly-indirect pointer
    - One doubly-indirect pointer
    - One triply-indirect pointer
- Advantages:
  - Small files fit in the direct indexed pointers
  - Larger files increasingly utilize more indirect index lists

Inode indirection



From “File Systems Indirect Blocks and Extents” by Cory Xie (link)

Management of Free Space

- Parallels to memory management
  - Need to reuse disk space new files as other files are deleted
- System maintains some type of list to track free blocks
  - Creating a file removes some blocks from free list
  - Deleting a file adds some blocks to free list
- Free list implementations
  - Bit map
  - Linked list
  - Grouping
  - Counting

Bit Map

- A bit per block is allocated to store block status
- Advantages
  - Simplicity
  - Easy to find first free block or N consecutive free blocks
    - Many architectures have instructions to efficiently find first set bit
- Disadvantages
  - Efficient only if bit map can be kept in main memory
  - Size overhead becomes too large for large disks

Linked List

- Link all free blocks together in a list
- Head of list stored in special disk location
  - Also cached in CPU memory
- Operations are not efficient
  - Searching for free blocks requires many disk reads
  - But traversal is infrequent
  - Most often, the OS simply wants 1 free block
    - E.g. to allocate via an indexed allocation scheme
  - Can use first free block in list

Grouping

- Tweak on the free list approach
- Store address of N free blocks in first free block
  - N-1 of them are free blocks for use for files
  - Last one points to block containing pointers to N more free blocks
- Advantage
  - Multiple free blocks can now be found quickly
    - With few disk read operations

Counting

- Generally there are clusters of free blocks
  - When files are deleted that span multiple blocks
- Keep address of a free block + # of subsequent free blocks
  - Entry of free list is a block address + count
  - Each list entry requires a little more state
  - But number of free list entries may be significantly reduced

Conclusion

- File system design is a major contributor to overall performance
- Abstracts block device (read_block/write_block) into files and directories (open/read/write/close/mkdir/rm/...)
- Key design questions we looked at:
  - **Directory implementation**: list, hash table, *other options!*
  - **Block allocation**: contiguous, linked list, index, multi-level hybrid, *other options!*
  - **Free space management**: bitmap, list (+grouping?, +counting?), *others options!*