File Systems Control Structures

- A file is a named collection of information
- Primary roles of file system:
  - To store and retrieve data
  - To manage the media/space where data is stored
- Typical operations:
  - Where is the first block of this file?
  - Where is the next block of this file?
  - Where is block 35 of this file?
  - Allocate a new block to the end of this file
  - Free all blocks associated with this file

Finding Data On Disks

- Essentially a question of how you managed the space on your disk
- Space management on disk is complex
  - There are millions of blocks and thousands of files
  - Files are continuously created and destroyed
  - Files can be extended after they have been written
  - Data placement on disk has performance effects
  - Poor management leads to poor performance
- Must track the space assigned to each file
  - On-disk, master data structure for each file

On-Disk File Control Structures

- On-disk description of important attributes of a file
  - Particularly where its data is located
- Virtually all file systems have such data structures
  - Different implementations, performance & abilities
  - Implementation can have profound effects on what the file system can do (well or at all)
- A core design element of a file system
- Paired with some kind of in-memory representation of the same information

The Basic File Control Structure Problem

- A file typically consists of multiple data blocks
- The control structure must be able to find them
- Preferably able to find any of them quickly
  - I.e., shouldn't need to read the entire file to find a block near the end
- Blocks can be changed
- New data can be added to the file
  - Or old data deleted
- Files can be sparsely populated

The In-Memory Representation

- On file open, create an in-memory structure
- Not an exact copy of the disk version
  - The disk version points to disk blocks
  - The in-memory version points to RAM pages
    - Or indicates that the block isn't in memory
  - Also keeps track of which blocks are dirty and which aren't
- Handles issues of multiple processes sharing an open file simultaneously

File System Structure

- How do I organize a disk into a file system?
  - Linked extents
    - The DOS FAT file system
  - File index blocks
    - Unix System V file system

Basics of File System Structure

- Most file systems live on disks
- Disk volumes are divided into fixed-sized blocks
  - Many sizes are used: 512, 1024, 2048, 4096, 8192 ...
- Most blocks will be used to store user data
- Some will be used to store organizing "meta-data"
  - Description of the file system (e.g., layout and state)
  - File control blocks to describe individual files
  - Lists of free blocks (not yet allocated to any file)
- All operating systems have such data structures
  - Different OSes and file systems have very different goals
  - These result in very different implementations

The Boot Block

- The 0^{th} block of a disk is usually reserved for the boot block
  - Code allowing the machine to boot an OS
- Not usually under the control of a file system
  - It typically ignores the boot block entirely
- Not all disks are bootable
  - But the 0^{th} block is usually reserved, "just in case"
- So file systems start work at block 1

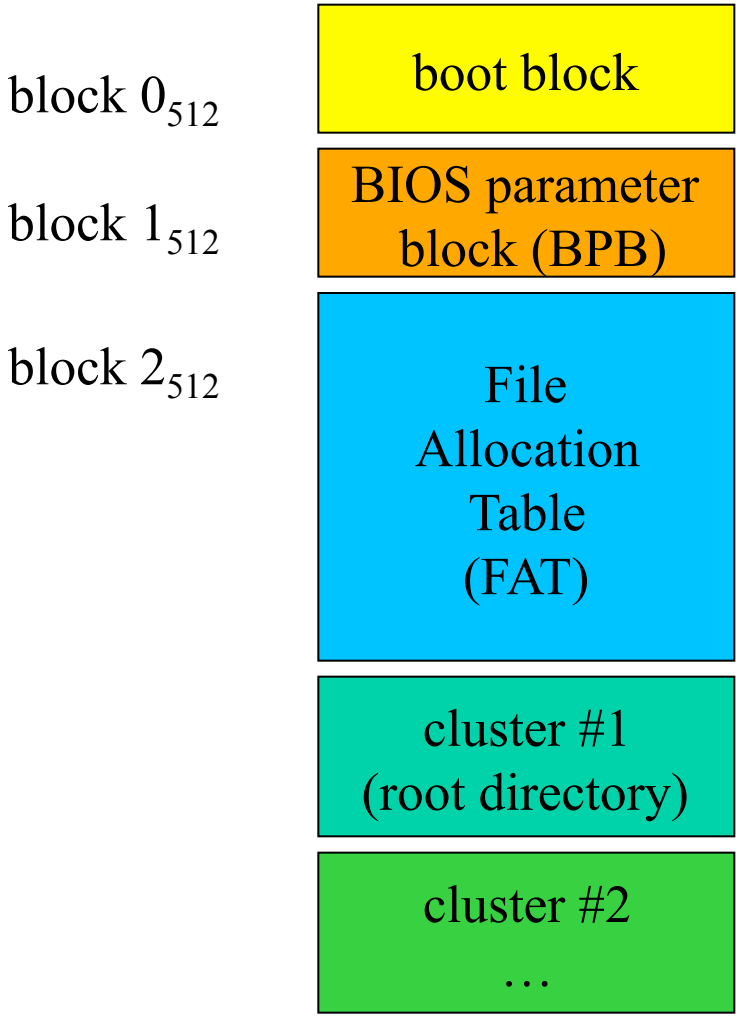
Managing Allocated Space

- A core activity for a file system, with various choices
- What if we give each file same amount of space?
  - Internal fragmentation ... just like memory
- What if we allocate just as much as file needs?
  - External fragmentation, compaction ... just like memory
- Perhaps we should allocate space in "pages"
  - How many chunks can a file contain?
- The file control data structure determines this
  - It only has room for so many pointers, then file is "full"
- So how do we want to organize the space in a file?

Linked Extents

- A simple answer
- File control block contains exactly one pointer
  - To the first chunk of the file
  - Each chunk contains a pointer to the next chunk
  - Allows us to add arbitrarily many chunks to each file
- Pointers can be in the chunks themselves
  - This takes away a little of every chunk
  - To find chunk N, you have to read the first N-1 chunks
- Pointers can be in auxiliary "chunk linkage" table
  - Faster searches, especially if table kept in memory

The DOS File System



Cluster size and FAT length are specified in the BPB

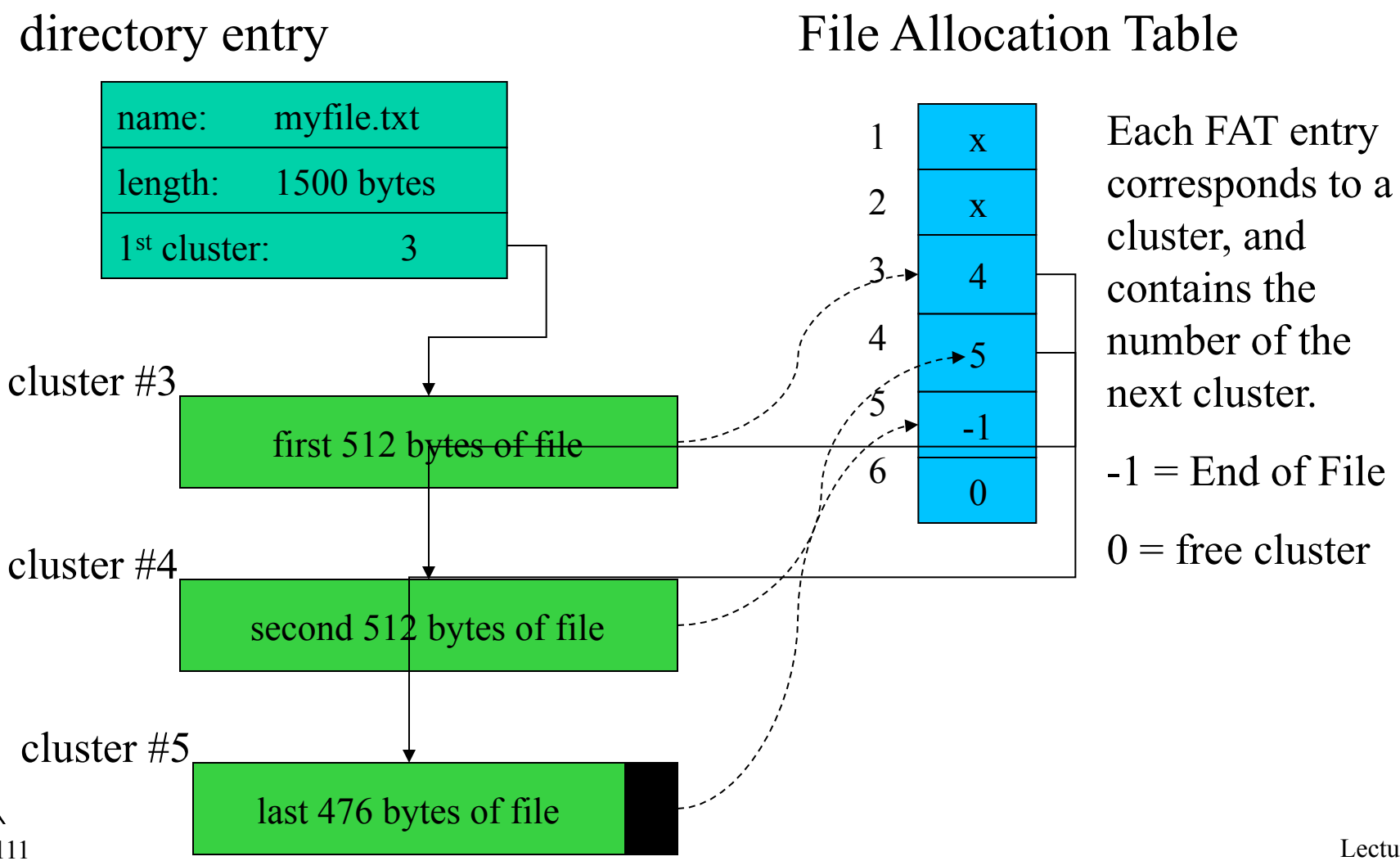
Data clusters begin immediately after the end of the FAT

Root directory begins in the first data cluster

DOS File System Overview

- DOS file systems divide space into "clusters"
  - Cluster size (multiple of 512) fixed for each file system
  - Clusters are numbered 1 though N
- File control structure points to first cluster of a file
- File Allocation Table (FAT), one entry per cluster
  - Contains the number of the next cluster in file
  - A 0 entry means that the cluster is not allocated
  - A -1 entry means "end of file"
- File system is sometimes called "FAT," after the name of this key data structure

DOS FAT Clusters



DOS File System Characteristics

- To find a particular block of a file
  - Get number of first cluster from directory entry
  - Follow chain of pointers through File Allocation Table
- Entire File Allocation Table is kept in memory
  - No disk I/O is required to find a cluster
  - For very large files the search can still be long
- No support for "sparse" files
  - Of a file has a block *n*, it must have all blocks < *n*
- Width of FAT determines max file system size
  - How many bits describe a cluster address
  - Originally 8 bits, eventually expanded to 32

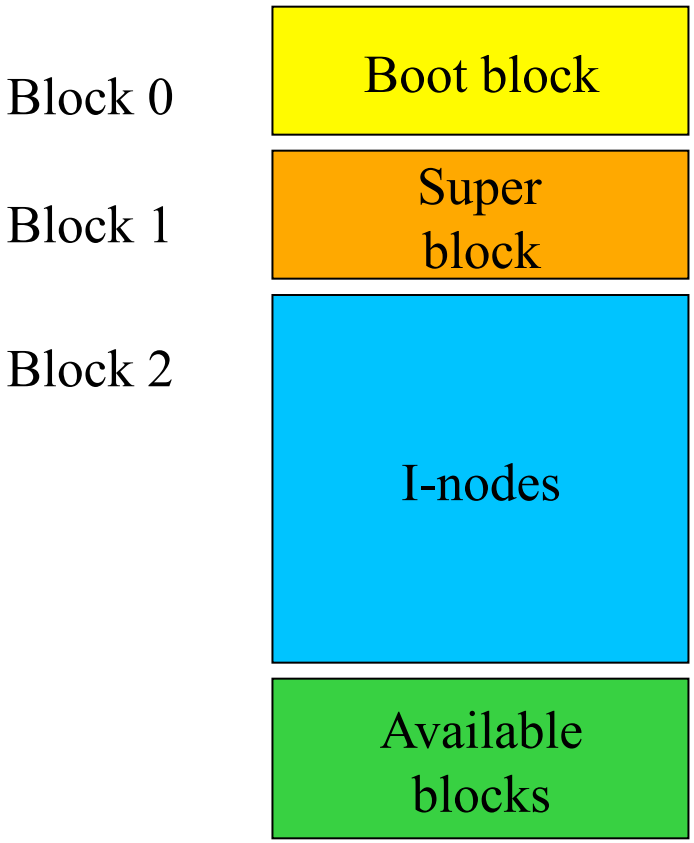
File Index Blocks

- A different way to keep track of where a file's data blocks are on the disk
- A file control block points to all blocks in file
  - Very fast access to any desired block
  - But how many pointers can the file control block hold?
- File control block could point at extent descriptors
  - But this still gives us a fixed number of extents

Hierarchically Structured File Index Blocks

- To solve the problem of file size being limited by entries in file index block
- The basic file index block points to blocks
- Some of those contain pointers which in turn point to blocks
- Can point to many extents, but still a limit to how many
  - But that limit might be a very large number
  - Has potential to adapt to wide range of file sizes

Unix System V File System

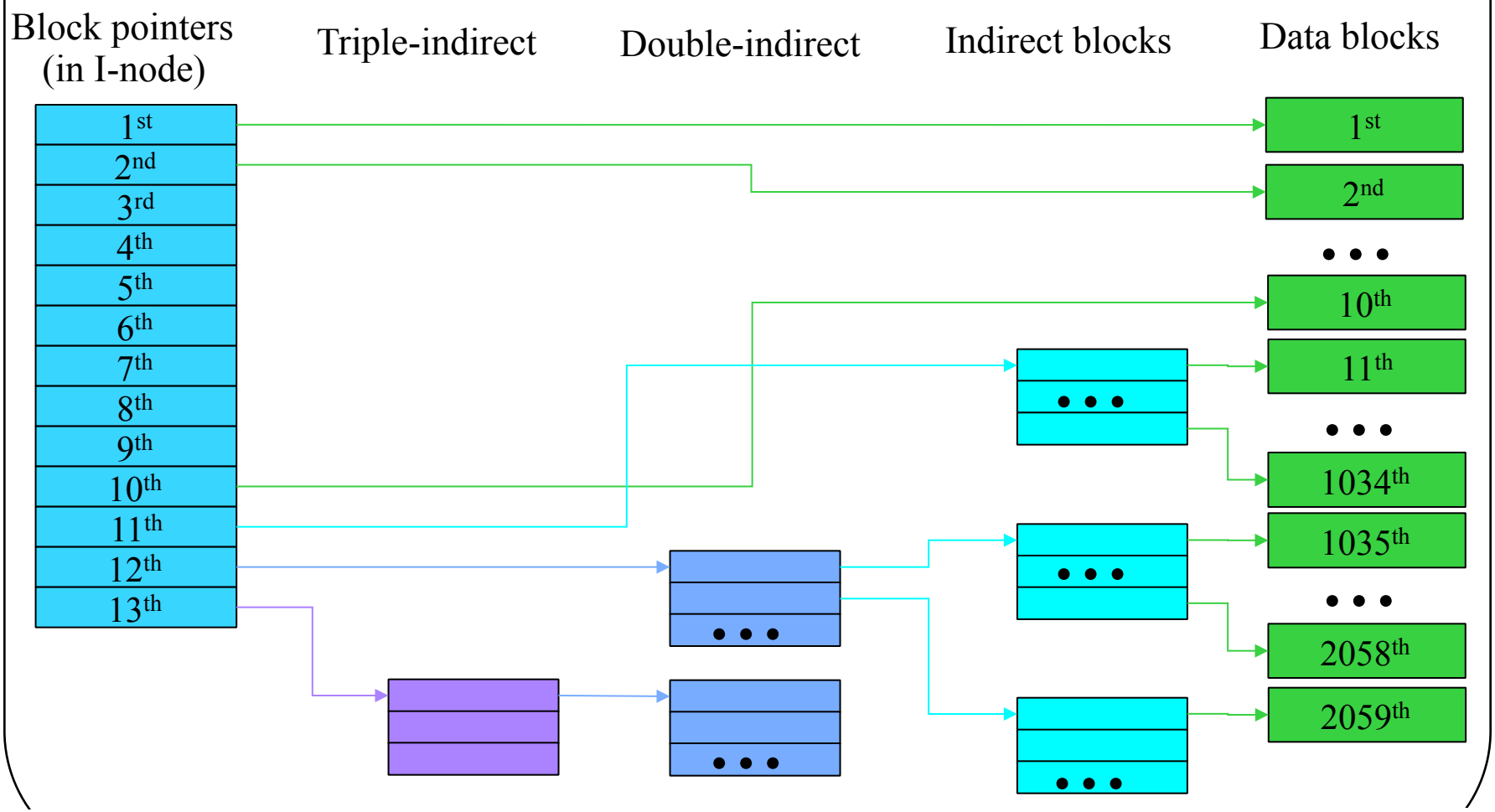


Block size and number of I-nodes are specified in super block

I-node #1 (traditionally) describes the root directory

Data blocks begin immediately after the end of the I-nodes.

Unix Inodes and Block Pointers



Why Is This a Good Idea?

- The UNIX pointer structure seems ad hoc and complicated
- Why not something simpler?
  - E.g., all block pointers are triple indirect
- File sizes are not random
  - The majority of files are only a few thousand bytes long
- Unix approach allows us to access up to 40Kbytes (assuming 4K blocks) without extra I/Os
  - Remember, the double and triple indirect blocks must themselves be fetched off disk

How Big a File Can Unix Handle?

- The on-disk inode contains 13 block pointers
  - First 10 point to first 10 blocks of file
  - 11th points to an indirect block (which contains pointers to 1024 blocks)
  - 12th points to a double indirect block (pointing to 1024 indirect blocks)
  - 13th points to a triple indirect block (pointing to 1024 double indirect blocks)
- Assuming 4k bytes per block and 4-bytes per pointer
  - 10 direct blocks = 10 * 4K bytes = 40K bytes
  - Indirect block = 1K * 4K = 4M bytes
  - Double indirect = 1K * 4M = 4G bytes
  - Triple indirect = 1K * 4G = 4T bytes
  - At the time system was designed, that seemed impossibly large
  - But . . .

Unix Inode Performance Issues

- The inode is in memory whenever file is open
- So the first ten blocks can be found with no extra I/O
- After that, we must read indirect blocks
  - The real pointers are in the indirect blocks
  - Sequential file processing will keep referencing it
  - Block I/O will keep it in the buffer cache
- 1-3 extra I/O operations per thousand pages
  - Any block can be found with 3 or fewer reads
- Index blocks can support "sparse" files
  - Not unlike page tables for sparse address spaces