Announcements

- **Reading Chapter 11 (11.1-11.5)**
  - suggested problems: 11.1, 11.2, 11.6, 11.8
- **Midterm #1 was returned on Th. before spring break**
  - all request for re-grade must be submitted in writing by April 2 at 10:45 AM.
- **Program #3 is due on Wed. in discussion section**

Filesystems

- **Raw Disks can be viewed as:**
  - a linear array of fixed sized units of allocation, called blocks
    - assume that blocks are error free (for now)
    - typical block size is 512 to 4096 bytes
  - can update a block in place, but must write the entire block
  - can access any block in any desired order
    - blocks must be read as a unit
    - for performance reasons may care about "near" vs. "far" blocks (but that is covered in a future lecture)

- **A Filesystem:**
  - provides a hierarchical namespace via directories
  - permits files of variable size to be stored
  - provides disk protection by restricting access to files based on permissions

File System Implementation

Application Programs

⬇

Logical file system:
Knows about directories, application view of file names

⬇

File Organization Module:
Can translate logical block addresses to physical block addresses

⬇

Basic File System:
Issues physical block read/write commands

⬇

Low Level I/O Control
Interfaces to hardware

Allocation Methods

- How do we select a free disk block to use?

- Contiguous allocation

  - allocate a contiguous chunk of space to a file

  - directory entry indicates the starting block and the length of the file

  - easy to implement, but

    - how to satisfy a given sized request from a list of free holes?

    - two options

      - first fit (find the first gap that fits)

      - best fit (find the smallest gaps that is large enough)

    - What happens if one wants to append to file?

  - from time to time, one will need to repack files

copyright 1996 Jeffrey K. Hollingsworth

Linked Allocation

- Each file is a linked list of disk blocks, blocks can be located anywhere
  - Directory contains a pointer to the first and last block of a file
  - Each block contains a pointer to the next block
  - This is essentially a linked-list data structure

- Problems:
  - Best for sequential access data structures
    - requires sequential access whether you want to or not!
  - Reliability - one bad sector and all portions of your file downstream are lost

- Useful fix:
  - Maintain a separate data structure just to keep track of linked lists
  - Data-structure includes pointers to actual blocks

Indexed Allocation

- **Bring all pointers together in an *index block***
  - Each file has its own index block - *i*th entry of index block points to *i*th block making up the file

- **How large to make an index block?**
  - unless one only wants to support fixed size files, index block scheme needs to be extensible

- **Linked scheme:**
  - maintain a linked list of indexed blocks

- **Multilevel index:**
  - Index block can point to other index blocks (which point to index blocks ....), which point to files

- **Hybrid multi-level index**
  - first n blocks are from a fixed index
  - next m blocks from an indirect index
 - next o blocks from a double indirect index

Hybrid Multi-level Index (UNIX)

- **Observations**
  - most files are small
  - most of the space on the disk is consumed by large files
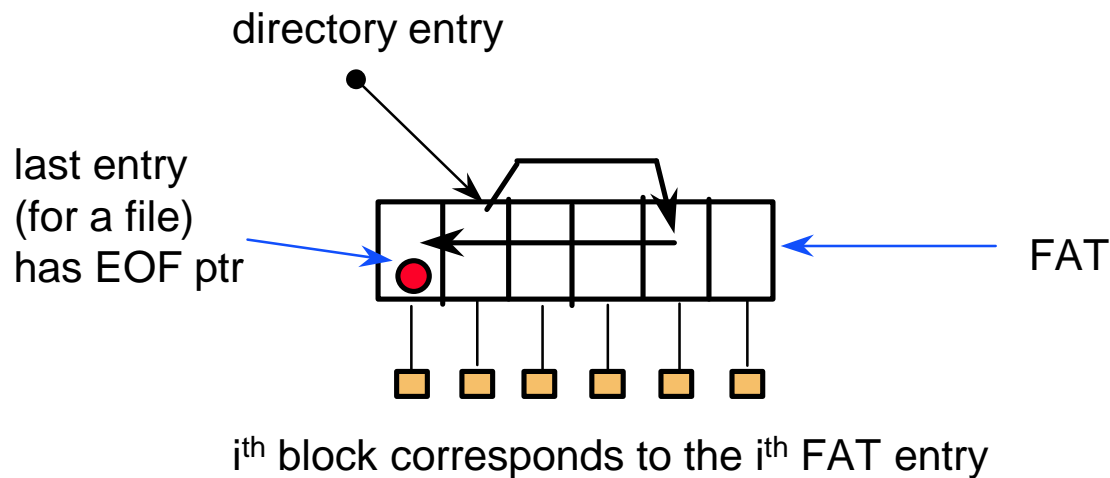
- **Want a flexible way to support different sized**
  - assume 4096 byte block
  - first 12 blocks (48 KB) are from a fixed index
  - next 1024 blocks (4 MB) from an indirect index
  - next 1024^2 blocks (16 GB) from a double indirect index
  - final 1024^3 blocks (64 TB) from a triple indirect index
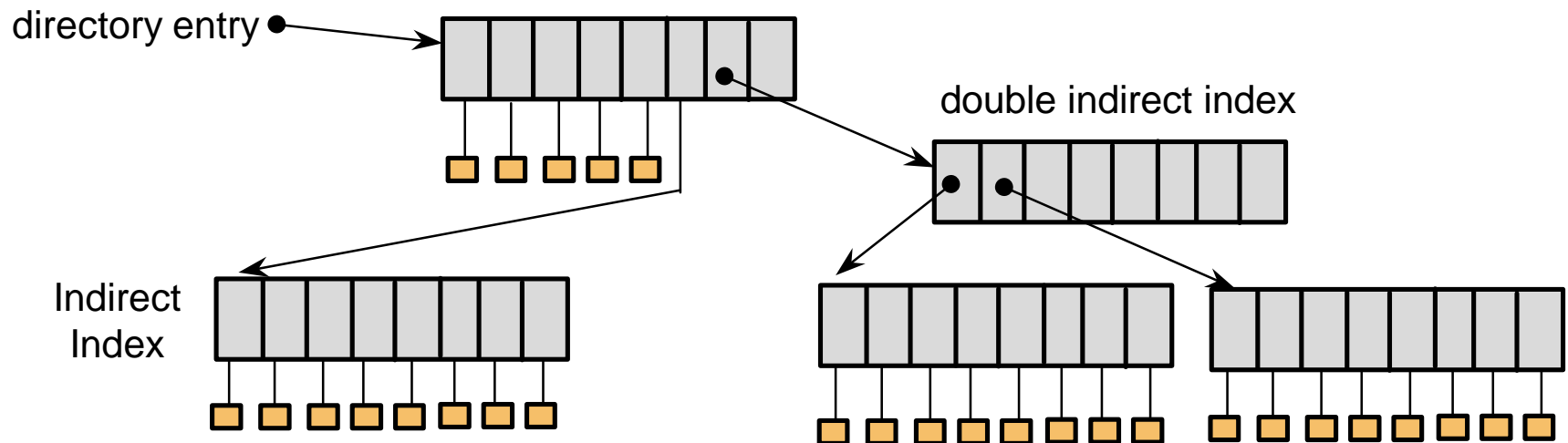
Modified Linked Allocation (FAT)

- **Section of disk contains a table**
  - called the file allocate table (FAT)
  - used in MS-DOS
- **Directory entry contains the block number of the first block in the file**
- **Table entry contains the number of the next block in the file**
- **Last block has a end-of-file value as a table entry**

directory entry

last entry
(for a file)
has EOF ptr

FAT

i^{th} block corresponds to the ith FAT entry

copyright 1996 Jeffrey K. Hollingsworth

Performance Issues

- ## FAT
  - 🗐 simple, easy to implement
  - 🗐 faster to traverse than linked allocation
  - – random access requires following links

- ## Hybrid indirect
  - 🗐 fast access to any part of the file
  - – more complex

Free Space Management

- How do we find a disk block to allocate?
- Bit Vectors
  - array of bits (one per block) that indicates if a block is free
  - compact so can keep in memory
    - 1.3 GB disk, 4K blocks -> 78K per disk
  - easy to find long runs of free blocks
- Linked lists
  - each disk block contains the pointer to the next free block
  - pointer to first free block is keep in a special location on disk
- Run length encoding (called counting in book)
  - pointer to first free block is keep in a special location on disk
  - each free block also includes a count of the number of consecutive blocks that are free

Implementing Directories

- ● **Linear List**
  - – array of names for files
  - – must search entire list to find or allocate a filename
  - – sorting can improve search performance, but adds complexity

- ● **Hash table**
  - – use hash function to find filenames in directory
  - – needs a good hash function
  - – need to resolve collisions
  - – must keep table small and expand on demand since many directories are mostly empty

DOS Directories

- ## Root directory
  - immediately follows the FAT

- ## Directory is a table of 32 byte entries
  - 8 byte file name, 3 byte filename extension
  - size of file, data and time stamp, starting cluster number of the file, file attribute codes
  - Fixed size and capacity

- ## Subdirectory
  - This is just a file
  - Record of where the subdirectory is located is stored in the FAT