## Announcements

- Reading Chapter 12
- Project #4 is Due Monday at 5:00 PM
- Midterm #2 is next Thursday in class

## File Consistency semantics

- How do multiple processes see updates to files
- UNIX
  - writes are visible immediately
  - have a mode to permit processes to share file pointers (“dup”)
- AFS
  - open/close semantics (keep a local copy)
    - “copy” the file on open
    - write-back on close
- Immutable files
  - once made visible to the world, the file never changes
    - usually done by attaching a version # to the filename
  - new versions of the file must be given a new name

## 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

## 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

## Linked Allocation

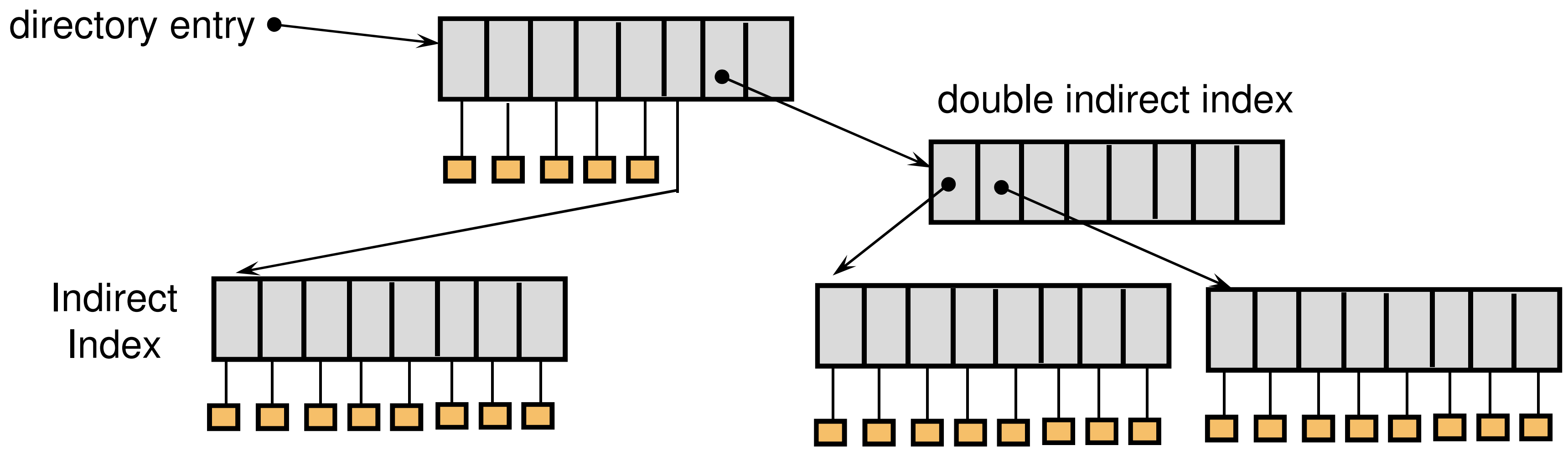
- Each file is a linked list of disk blocks, blocks can be located anywhere
  - Directory contains a pointer to the first and last block of a file
  - Each block contains a pointer to the next block
  - This is essentially a linked-list data structure
- Problems:
  - Best for sequential access data structures
    - 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?
  - To avoid a fixed maximum file size, index block must 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 (4 GB) from a double indirect index
  - final $1024^3$ blocks (4 TB) from a triple indirect index


directory entry
double indirect index
Indirect Index