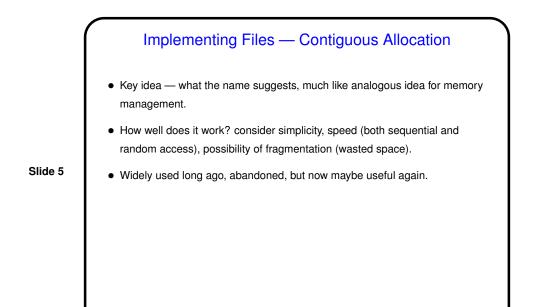
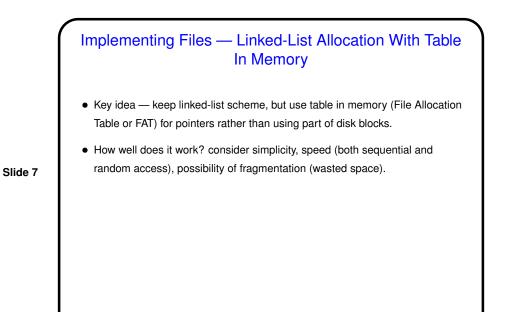


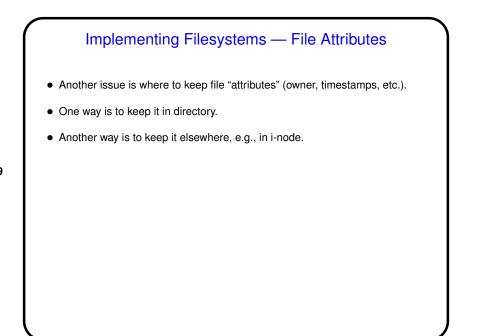
## Implementing Files One problem is keeping track of which disk blocks belong to which files. No surprise — there are several approaches. (All assume some outside "directory"-type structure with some information about each file — a starting block, e.g.)

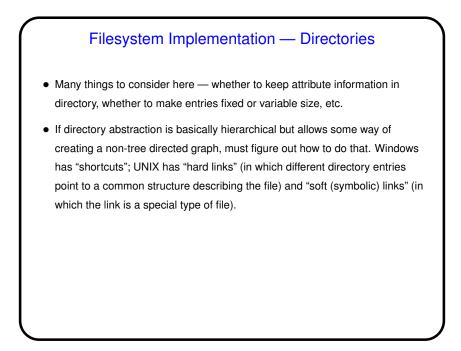


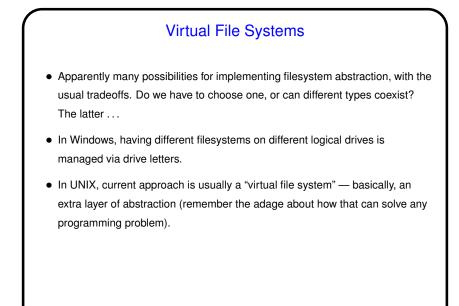
Implementing Files — Linked-List Allocation
Key idea — organize each file's blocks as a linked list, with pointer to next block stored within block.
How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).



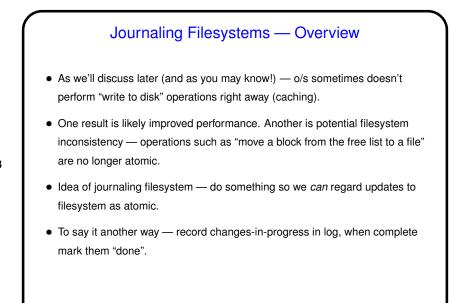
Implementing Files — I-Nodes • Key idea — associate with each file a data structure ("index node" or i-node) containing file attributes and disk block numbers, keep in memory for "open" files. • How well does it work? consider simplicity, speed (both sequential and random access), possibility of fragmentation (wasted space).







## Log-Structured Filesystems Log-structured filesystem — *everything* is written to log, and only to log. That sounds impractical, but ... Key idea is that these many disk reads are satified from cache anyway, and lots of small writes to disk give poor performance, so it makes more sense to just write (to cache) a log, and periodically save that to disk. Not used much, though, because incompatible with other file systems. Instead ...

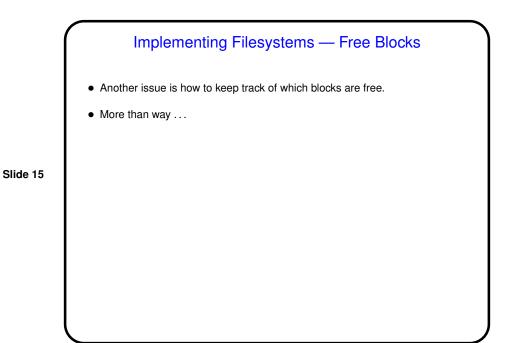


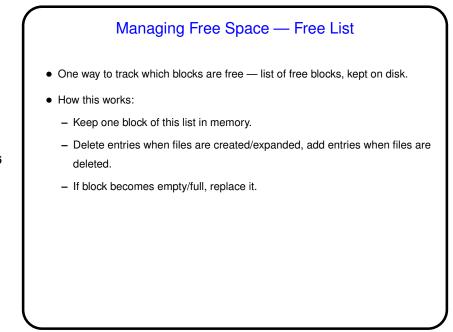
Slide 14

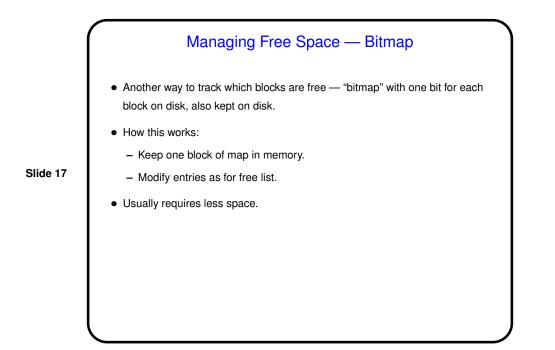
Journaling Filesystems, Continued
Can record "data", "metadata" (directory info, free list, etc.), or both.
"Undo logging" versus "redo logging":

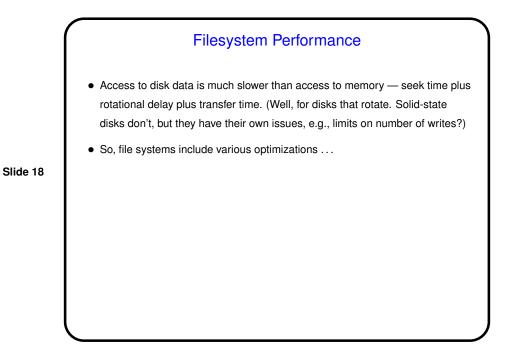
Undo logging: First copy old data to log, then write new data (possibly many blocks) to disk. If something goes wrong during update, "roll back" by copying old data from log.
Redo logging: First write new data to log (i.e., record changes we're going to make), then write new data to disk. If something goes wrong during update, complete the update using data in log.

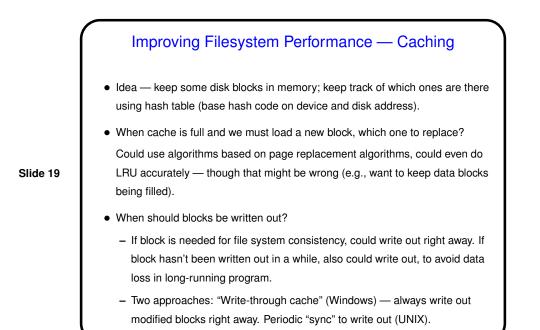
A key benefit — after a system crash, we should only have to look at the log for incomplete updates, rather than doing a full filesystem consistency check.











 Idea — if file is being read sequentially, can read some blocks "ahead". (Of course, doesn't help if file is being read non-sequentially. Decide based on recent access patterns.)

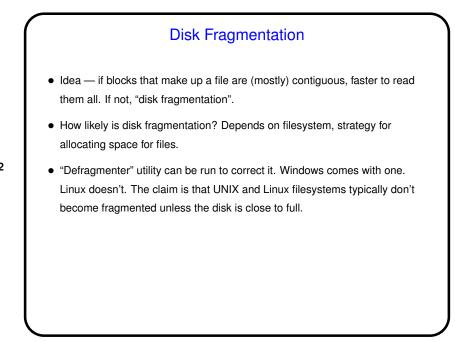
- Improving Filesystem Performance Block Read-Ahead

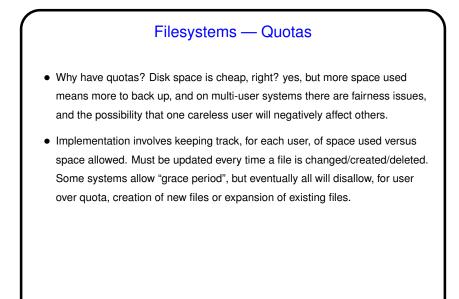


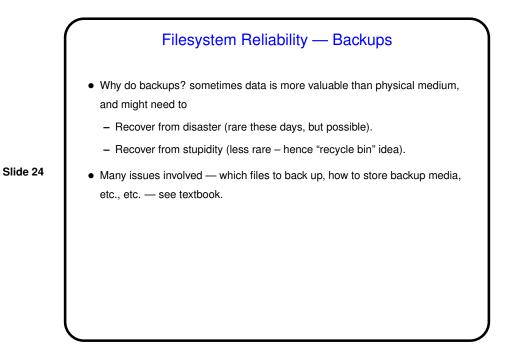
 Group blocks for each file together — easier if bitmap is used to keep track of free space. If not grouped together — "disk fragmentation" may affect performance.

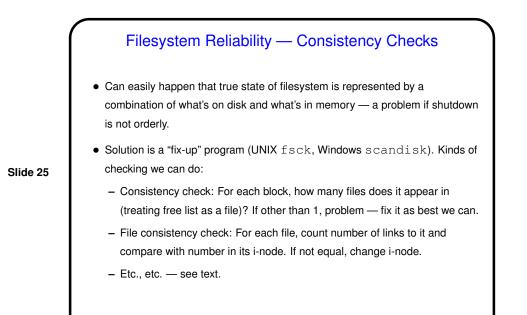
Slide 21

• If i-nodes are being used, place them so they're fast to get to (and so maybe we can read an i-node and associated file block together).









6
If you have a system that supports multiple different file systems (such as Linux with Samba to access Windows files), what problems might arise in copying files between different file systems?
(We had an interesting problem many years ago with backing up /users to an OS X machine because the default for OS X is case-insensitive.)
Also — anything noteworthy about Homework 5?

