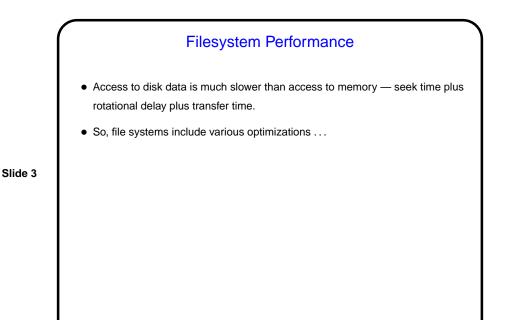


## Virtual File Systems Apparently many possibilities for implementing filesystem abstraction, with the usual tradeoffs. Do we have to choose one, or can different types coexist? The latter . . . In Windows, having different filesystems on different logical drives is managed via drive letters. In UNIX, current approach is usually a "virtual file system" — basically, an extra layer of abstraction (remember the adage about how that can solve any programming problem).

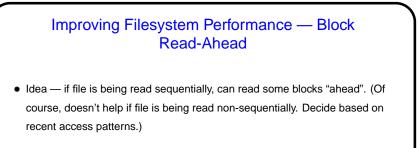


Improving Filesystem Performance — Caching
Idea — keep some disk blocks in memory; keep track of which ones are there using hash table (base hash code on device and disk address).
When cache is full and we must load a new block, which one to replace? Could use algorithms based on page replacement algorithms, could even do LRU accurately — though that might be wrong (e.g., want to keep data blocks being filled).
When should blocks be written out?

If block is needed for file system consistency, could write out right away. If block hasn't been written out in a while, also could write out, to avoid data

 Two approaches: "Write-through cache" (Windows) — always write out modified blocks right away. Periodic "sync" to write out (UNIX).

loss in long-running program.



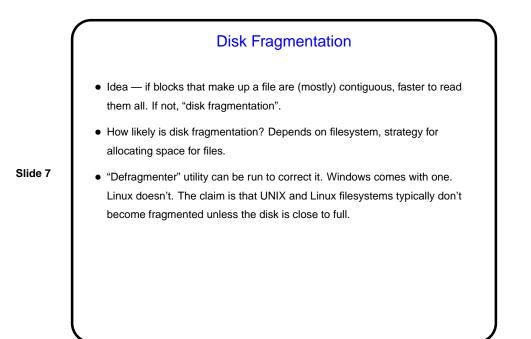
Slide 5

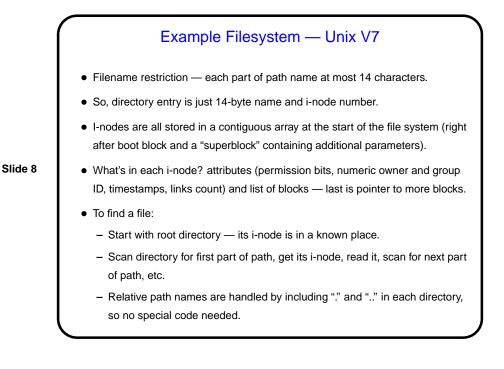
## Improving Filesystem Performance — Reducing Disk Arm Motion

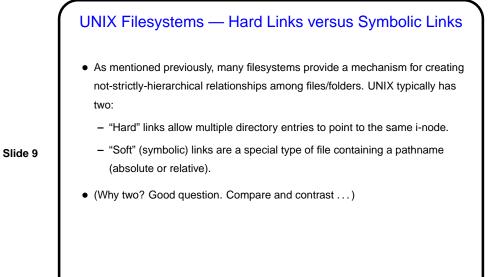
 Group blocks for each file together — easier if bitmap is used to keep track of free space. If not grouped together — "disk fragmentation" may affect performance.

Slide 6

• 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).







Minute Essay • What are the advantages/disadvantages of hard links versus symbolic links? are there things one can do that the other can't? are there ways in which one can go wrong and the other can't? • This wraps up the planned lectures on filesystems. Anything you'd like to hear more about?

