Reminder

- **Midterm monday**
- **Covers:**
  - Mostly readings and lectures
  - May reference lab—e.g., assume you understand NFS3
- **Open Book**
  - Make sure you bring copies of all papers and handouts
  - You can print out and bring class notes, too

Socket I/O

- **By default socket I/O is synchronous**
  - Read will block until there is some data to return
  - Write will block if kernel socket buffer is full
- **Traditional servers get concurrency with processes**
  - For TCP: Parent in accept, fork, close loop
  - Okay for single-client child to block in read/write
- **Other option: Non-blocking I/O (`make_async`)**
  - Read and write return EAGAIN instead of blocking
  - But disk I/O still synchronous (unless you use NFS)

Disk scheduling

- **Sequential transfer speeds fast (>10MB/sec)**
- **Seek times slow**
  - Short seeks $\sim$1 ms (settle time), long seeks $\geq$15 ms
- **Schedule disk to improve seek time**
  - Closer logical block numbers often physically closer (truest at larger distances when head movement dominates)
  - OS should attempt multiple requests near each other (CSCAN)
  - Disk can optimally schedule requests (*command queuing*)
  - The more requests the OS has, the better it can do

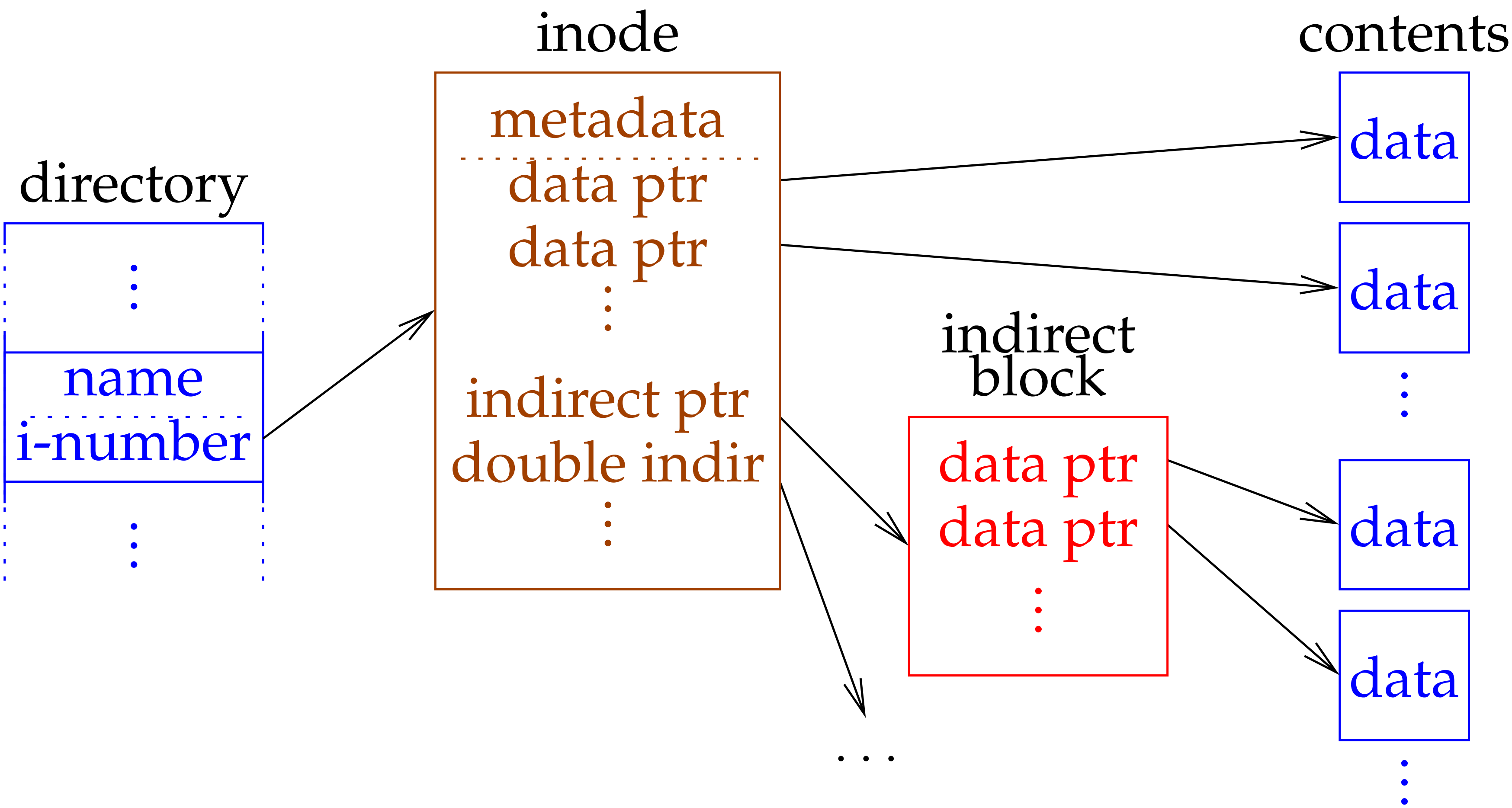
Deceptive idleness

- **Request locality often exists within a process**
  - Competing processes can cause much longer disk seeks
  - But each process doesn't always have a request outstanding (Takes short time after previous request to generate next)
- **Idea: Delay requests in case closer one generated**
  - Decision based on cost benefit analysis

UBM

- **Optimal: Evict block needed furthest in future**
- **LRU works well for most workloads, except**
  - Sequential (will never look at data again)
  - Looping (MRU is best to evict)
- **Detection: sequential, looping, or other**
- **Replacement: MRU for seq. & loop, LRU for other**
- **Allocator: Chose pool based on marginal gain**

FFS



- Cylinder groups: Put directories, inodes near files
- Write allocation: Try to write in big (64K) contiguous chunks

Shadow paging

- **Idea: Never overwrite blocks in place**
  - Keep page table mapping logical to actual block #s
  - Always write new copy of blocks (copy-on-write)
  - Update pointer to page table atomically
- **LFS**
  - Checkpoint & reset root every 30 seconds (consistent)
  - Can roll forward to lose less than 30 seconds
- **System R**
  - Transactions may be in progress at file update (inconsistent)
  - File root block also points to last checkpoint in log
  - Why use shadow paging?

Logging

- **Log-structured storage (LFS)—Data lives in log**
- **Write-ahead logging**
  - Separate log and "permanent" location for data
  - Always write log before permanent location
  - After crash, replay log—updates must be idempotent
- **Do/undo/redo**
  - Updates possibly not idempotent (e.g., external interactions)
  - Must undo "losers", redo "winners"

XFS

- **Tricks for good contiguous allocation**
  - Delay allocation until write-back (more to coalesce)
  - Use (position, length) extents instead of block pointers
  - B-tree free map lets you find blocks near desired location
- **Journaling for fast crash recovery**
  - Also allows metadata write-behind

Soft updates

- **Three rules of crash recovery**
  1. Never write pointer before initializing structure
  2. Never reuse block before nullifying pointers to it
  3. Set new pointer to live resource before clearing old
- **Undo any violations of rules before writing block**
- **Delay some updates until other blocks written**

AFS

- **Whole file caching w. close-to-open consistency**
  - Client can keep large on-disk cache
- **Designed to scale to many clients**
  - Callbacks – many opens don't go to server
  - Directory caching, w. component-by-component lookup
- **Break cell up into a number volumes**
  - Like a mini-file system that can be resized, moved
  - Copy on write mechanism helpful for backup, migration
  - Volume location server replicated at AFS servers, tells clients where to find a particular volume

SFS

- **Goal, access any server from any client, securely**
- **Idea: Make file system security independent of key management**
- **Specify server keys in *self-certifying pathnames*:**

  `/sfs/@sfs.mit.edu,bzcc5hder7cuc86kf6qswyx6yuemnw69/dm/`
- **Allows multiple key management schemes to coexist**
  - Local symbolic link place by system administrator
  - Use password to download path securely from server
  - Remote symbolic link offers delegation