CS5460/6460: Operating Systems

Lecture 17: File systems

Anton Burtsev
March, 2014

The role of file systems

The role of file systems

- Sharing
  - Sharing of data across users and applications
- Persistence
  - Data is available after reboot

Architecture

- On-disk and in-memory data structures represent
  - The tree of named files and directories
  - Record identities of disk blocks which hold data for each file
  - Record which areas of the disk are free

Crash recovery

- File systems must support crash recovery
  - A power loss may interrupt a sequence of updates
  - Leave file system in inconsistent state
    - E.g. a block both marked free and used

Multiple users

- Multiple users operate on a file system concurrently
  - File system must maintain invariants

Speed

- Access to a block device is several orders of magnitude slower
  - Memory: 200 cycles
  - Disk: 20 000 000 cycles
- A file system must maintain a cache of disk blocks in memory

Block layer

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

- Read and write data
  - From a block device
  - Into a buffer cache
- Synchronize across multiple readers and writers

Transactions

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

- Group multiple writes into an atomic transaction

Files

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

- Unnamed files
  - An inode
  - Sequence of blocks holding file's data

Directories

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

- Special kind of inode
  - Sequence of directory entries
  - Each contains name and a pointer to an unnamed inode

Pathnames

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

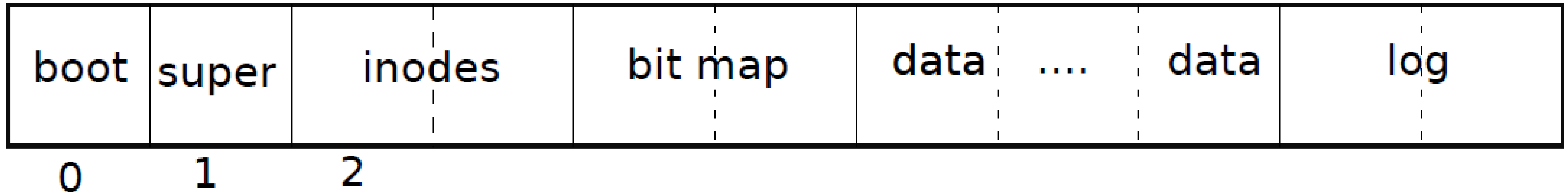
- Hierarchical path names
  - /usr/bin/sh
  - Recursive lookup

System call

System calls	File descriptors
Pathnames	Recursive lookup
Directories	Directory inodes
Files	Inodes and block allocator
Transactions	Logging
Blocks	Buffer cache

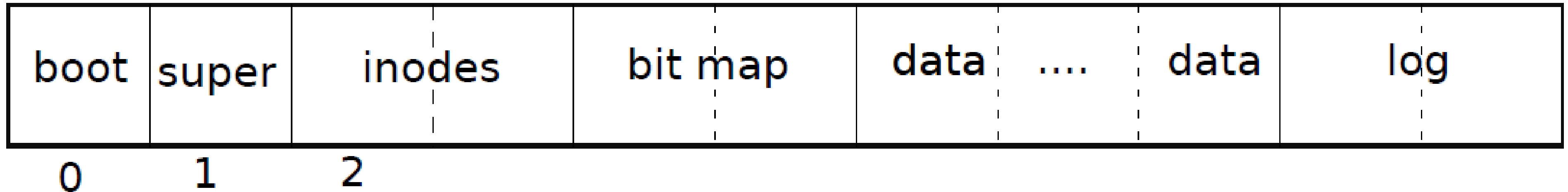
- Abstract UNIX resources as files
  - Files, sockets, devices, pipes, etc.
- Unified programming interface

File system layout on disk



- Block #0: Boot code
- Block #1: Metadata about the file system
  - Size (number of blocks)
  - Number of data blocks
  - Number of inodes
  - Number of blocks in log

File system layout on disk



- Block #2 (inode area)
- Bit map area: track which blocks are in use
- Data area: actual file data
- Log area: maintaining consistency in case of a power outage or system crash

Buffer cache layer

Buffer cache layer

- Two goals:
  - Synchronization:
    - Only one copy of a data block exist in the kernel
    - Only one writer updates this copy at a time
  - Caching
    - Frequently used copies are cached for efficient reads and writes

Buffer cache layer: interface

- bread()
- bwrite() - obtain a copy for writing
  - Owned until brelse()
  - Other threads will be blocked on bwrite() until brelse()

Logging layer

Logging layer

- Consistency
  - File system operations involve multiple writes to disk
  - During the crash, subset of writes might leave the file system in an inconsistent state
  - E.g. file delete can crash leaving:
    - Directory entry pointing to a free inode
    - Allocated but unlinked inode

Logging

- Writes don't directly go to disk
  - Instead they are logged in a journal
  - Once all writes are logged, the system writes a special commit record
    - Indicating that log contains a complete operation
- At this point file system copies writes to the on-disk data structures
  - After copy completes, log record is erased

Recovery

- After reboot, copy the log
  - For operations marked as complete
    - Copy blocks to disk
  - For operations partially complete
    - Discard all writes
    - Information might be lost (output consistency, e.g. can launch the rocket twice)

Block allocator

- Bitmap of free blocks
  - balloc()/bfree()
- Read the bitmap block by block
  - Scan for a “free” bit
- Access to the bitmap is synchronized with bread()/bwrite()/brelse() operations

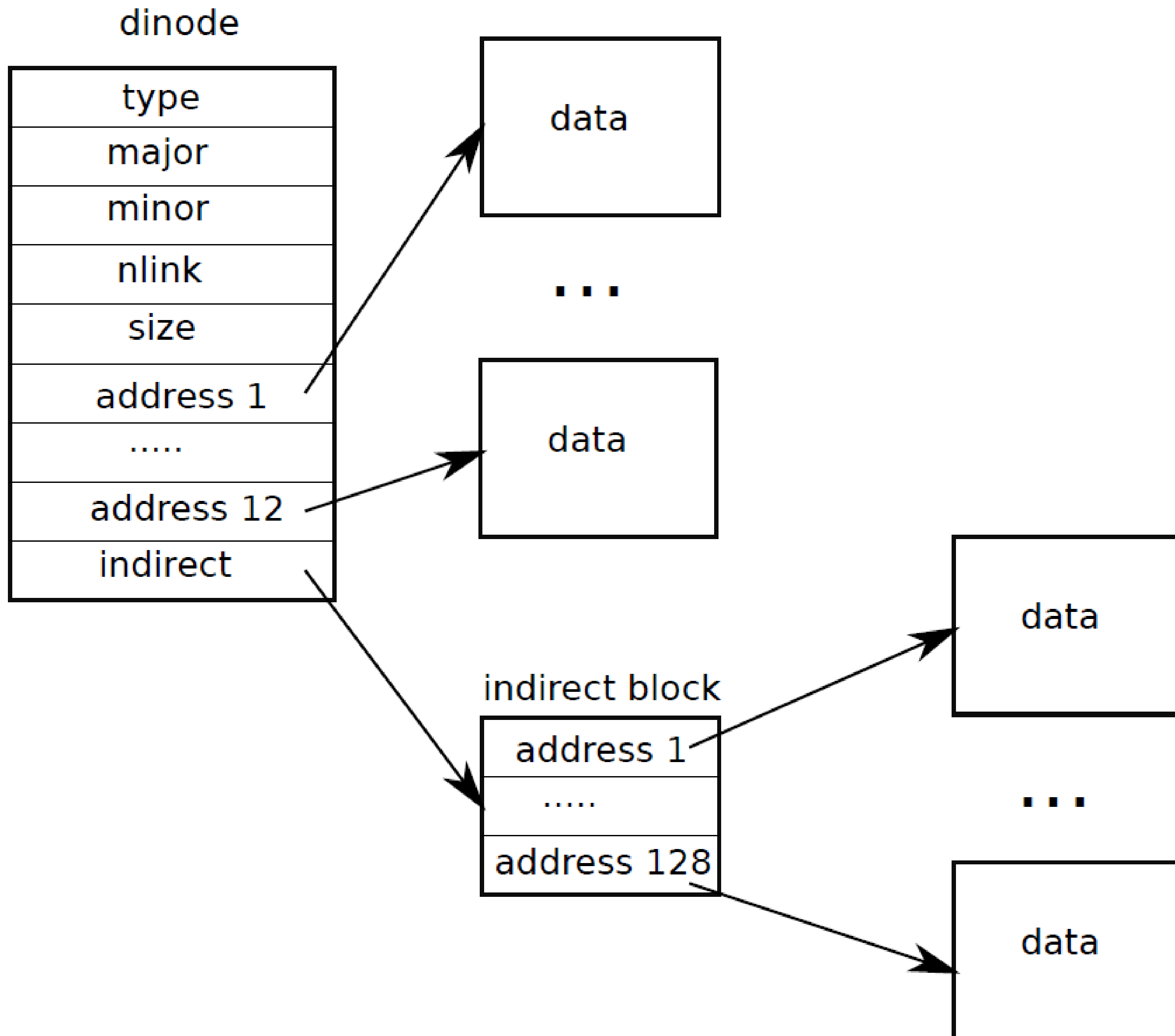
Inode layer

Inodes

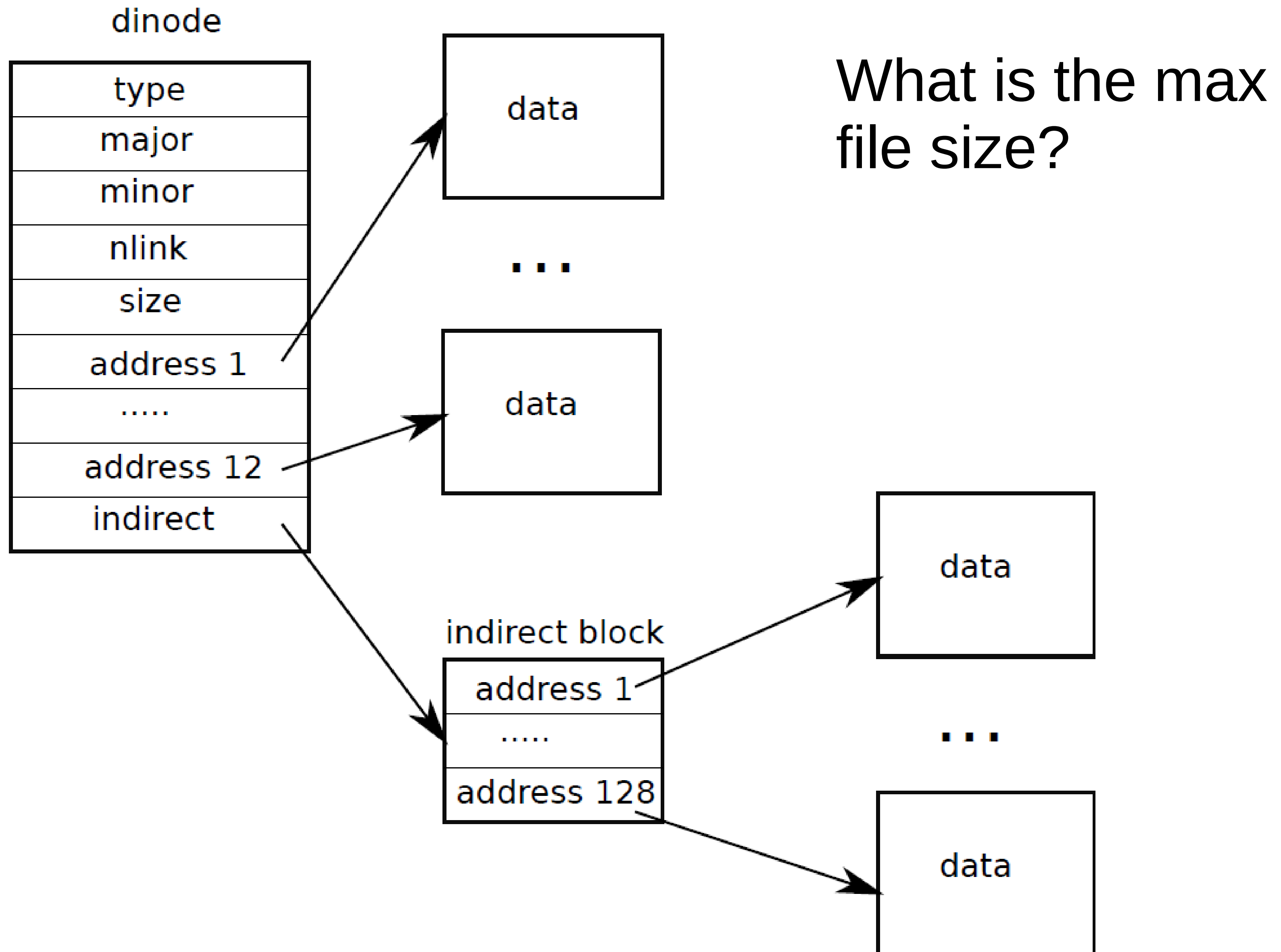
- Two meanings
  - On disk
    - File size + list of blocks with data
  - In memory
    - A copy of an on-disk inode + some additional kernel information
-

- Inodes are kept as an array in the inode area on disk
- In memory inodes
    - Cached

Representing files on disk

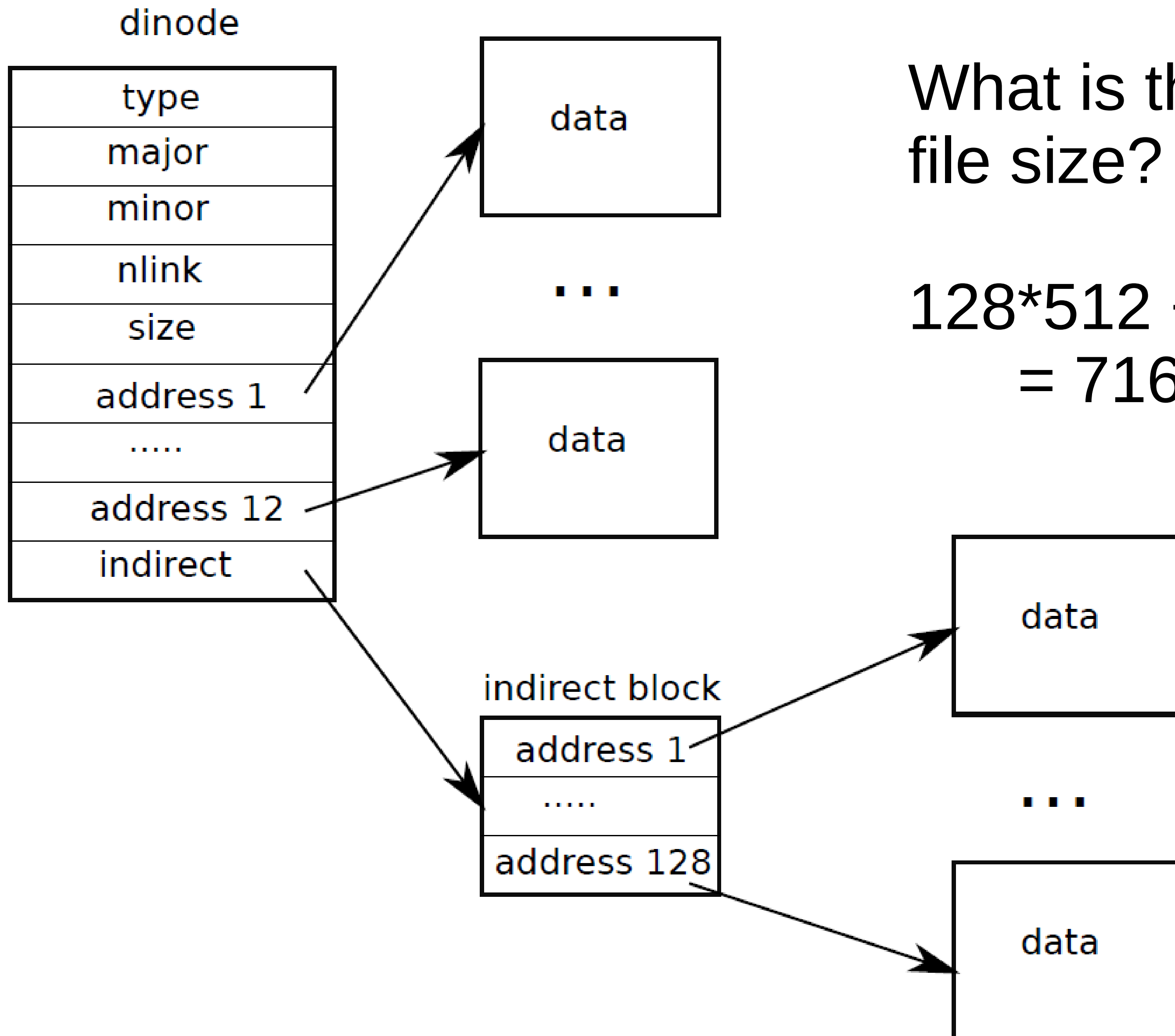


Representing files on disk



What is the max file size?

Representing files on disk



What is the max file size?

128*512 + 12*512 = 71680

Directory layer

Directory inodes

- A directory inode is a sequence of directory entries and inode numbers
  - Each name is max of 14 characters
  - Has a special inode type T_DIR
- dirlookup() - searches for a directory with a given name
- dirlink() - adds new file to a directory

Path names layer

- Series of directory lookups to resolve a path
  - E.g. /usr/bin/sh
- Namei() - resolves a path into an inode
  - If path starts with “/” evaluation starts at the root
  - Otherwise current directory

File descriptor layer

System call layer

System calls

- sys_link()/sys_unlink() - edit directories
  - Add/remove names of existing inodes
- Create() – creates a name for a new inode

Example: write system call

Write() syscall

```
5476 int
5477 sys_write(void)
5478 {
5479   struct file *f;
5480   int n;
5481   char *p;
5482
5483   if(argfd(0, 0, &f) < 0
       || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
5484     return -1;
5485   return filewrite(f, p, n);
5486 }
```

```
5352 filewrite(struct file *f, char *addr, int n)
5353 {
...
5360   if(f->type == FD_INODE){
5367     int max = ((LOGSIZE-1-1-2) / 2) * 512;
5368     int i = 0;
5369     while(i < n){
5370       int n1 = n - i;
5371       if(n1 > max)
5372         n1 = max;
5373
5374       begin_trans();
5375       ilock(f->ip);
5376       if ((r = writei(f->ip, addr + i, f->off, n1)) > 0)
5377         f->off += r;
5378       iunlock(f->ip);
5379       commit_trans();
5386   }
5390 }
```

Write several blocks at a time

Start a transaction

```
4277 begin_trans(void)
4278 {
4279   acquire(&log.lock);
4280   while (log.busy) {
4281     sleep(&log, &log.lock);
4282   }
4283   log.busy = 1;
4284   release(&log.lock);
4285 }
```

```
4952 writei(struct inode *ip, char *src, uint off, uint n)
4953 {

4965   if(off + n > MAXFILE*BSIZE)
4966     return -1;
4967
4968   for(tot=0; tot<n; tot+=m, off+=m, src+=m){
4969     bp = bread(ip->dev, bmap(ip, off/BSIZE));
4970     m = min(n - tot, BSIZE - off%BSIZE);
4971     memmove(bp->data + off%BSIZE, src, m);
4972     log_write(bp);
4973     brelse(bp);
4974   }
4975
4976   if(n > 0 && off > ip->size){
4977     ip->size = off;
4978     iupdate(ip);
4979   }
4980   return n;
4981 }
```

Write an inode

```
4810 bmap(struct inode *ip, uint bn) {

4815   if(bn < NDIRECT){
4816     if((addr = ip->addrs[bn]) == 0)
4817       ip->addrs[bn] = addr = balloc(ip->dev);
4818     return addr;
4819   }

4820   bn -= NDIRECT;
4822   if(bn < NINDIRECT){
4824     if((addr = ip->addrs[NDIRECT]) == 0)
4825       ip->addrs[NDIRECT] = addr = balloc(ip->dev);
4826     bp = bread(ip->dev, addr);
4827     a = (uint*)bp->data;
4828     if((addr = a[bn]) == 0){
4829       a[bn] = addr = balloc(ip->dev);
4830       log_write(bp);
4831     }
4832     brelse(bp);
4833     return addr;
4834   }
4837 }
```

Find a block on disk

Write to a log

```
4324 void
4325 log_write(struct buf *b)
4326 {
4327   int i;
4334   for (i = 0; i < log.lh.n; i++) {
4335     if (log.lh.sector[i] == b->sector) // log absorbtion?
4336       break;
4337   }
4338   log.lh.sector[i] = b->sector;
4339   struct buf *lbuf = bread(b->dev, log.start+i+1);
4340   memmove(lbuf->data, b->data, BSIZE);
4341   bwrite(lbuf);
4342   brelse(lbuf);
4343   if (i == log.lh.n)
4344     log.lh.n++;
4345   b->flags |= B_DIRTY; // XXX prevent eviction
4346 }
```

```
5352 filewrite(struct file *f, char *addr, int n)
5353 {
...
5360   if(f->type == FD_INODE){
5367     int max = ((LOGSIZE-1-1-2) / 2) * 512;
5368     int i = 0;
5369     while(i < n){
5370       int n1 = n - i;
5371       if(n1 > max)
5372         n1 = max;
5373
5374       begin_trans();
5375       ilock(f->ip);
5376       if ((r = writei(f->ip, addr + i, f->off, n1)) > 0)
5377         f->off += r;
5378       iunlock(f->ip);
5379       commit_trans();
5386   }
5390 }
```

Commit each writei

```
4301 commit_trans(void)
4302 {
4303   if (log.lh.n > 0) {
4304     write_head(); // Write header to disk -- the real commit
4305     install_trans(); // Now install writes to home locations
4306     log.lh.n = 0;
4307     write_head(); // Erase the transaction from the log
4308   }
4309
4310   acquire(&log.lock);
4311   log.busy = 0;
4312   wakeup(&log);
4313   release(&log.lock);
4314 }
```

Commit

```
4221 install_trans(void)
4222 {
4223   int tail;
4224
4225   for (tail = 0; tail < log.lh.n; tail++) {
         // read log block
4226     struct buf *lbuf = bread(log.dev, log.start+tail+1);
         // read dst
4227     struct buf *dbuf = bread(log.dev, log.lh.sector[tail]);
4228     memmove(dbuf->data, lbuf->data, BSIZE); // copy block to dst
4229     bwrite(dbuf); // write dst to disk
4230     brelse(lbuf);
4231     brelse(dbuf);
4232   }
4233 }
```

Write committed blocks to disk

```
4267 static void
4268 recover_from_log(void)
4269 {
4270   read_head();
4271   install_trans();
4272   log.lh.n = 0;
4273   write_head(); // clear the log
4274 }
```

Recover after reboot

Conclusion

Thank you!