TreeDisk and Testing

CS 4411

Spring 2020

Announcements

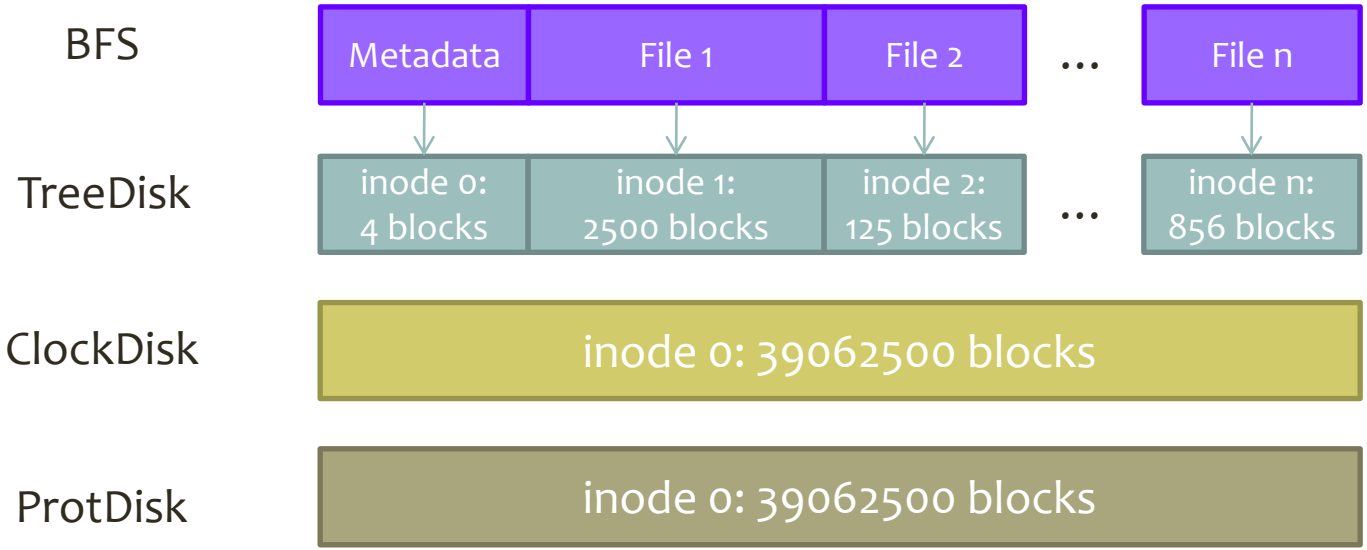
- Last lecture
- P5 due May 8th
- Office hours continue until May 8th (including in this time slot)

Outline for Today

- TreeDisk design
- TraceDisk and Traces
- Using TraceDisk to test and debug

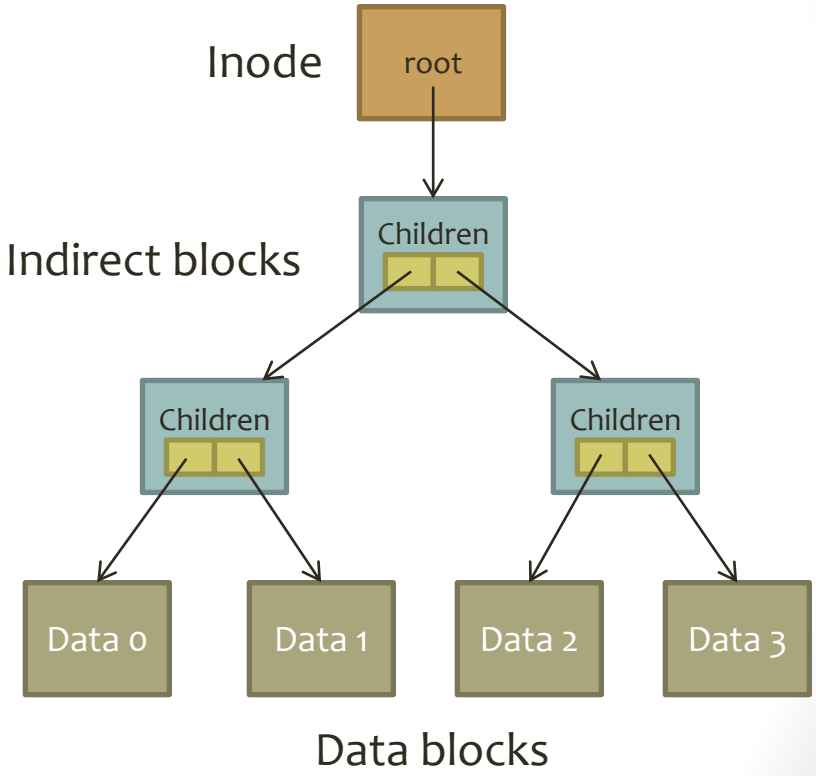
Recall: EGOS Filesystem Design

- Block File Server uses one Virtual Block Store per file
- Disks like TreeDisk and FatDisk provide VBSes



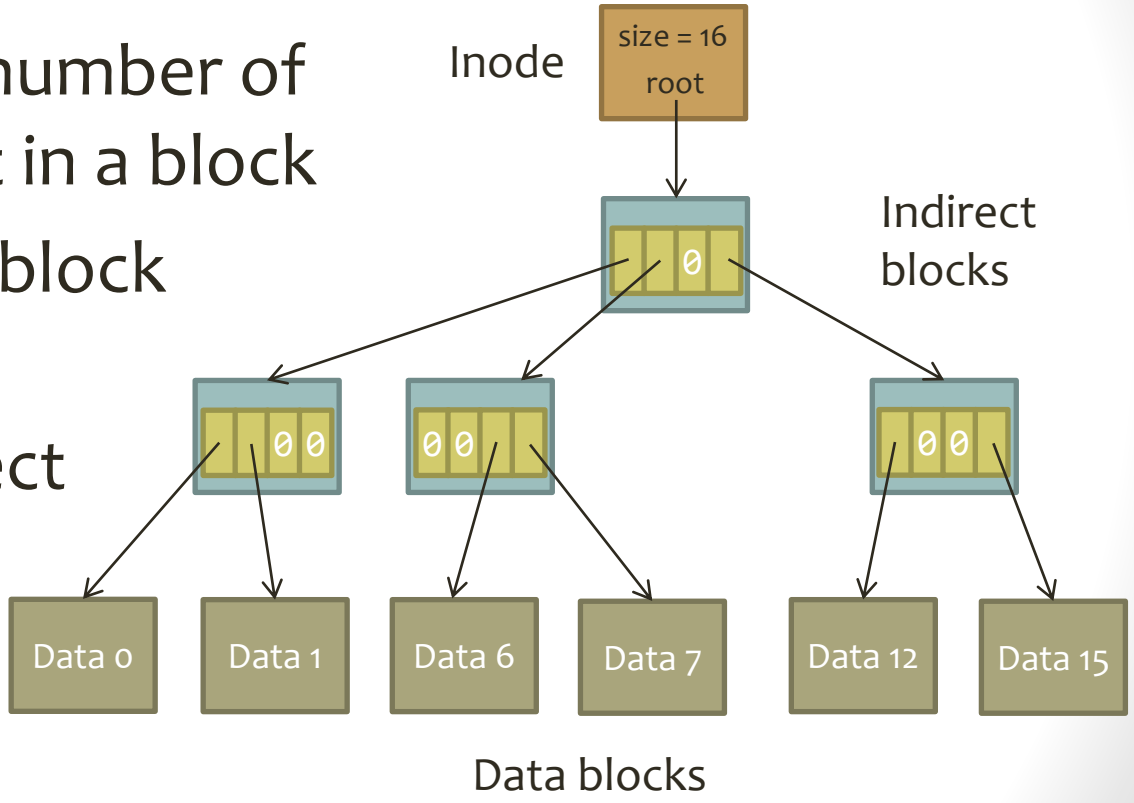
TreeDisk Overview

- Every VBS (file) is a tree of blocks
- Data only stored at leaves
- Indirect blocks store pointers to children (block numbers)
- Inode points to root

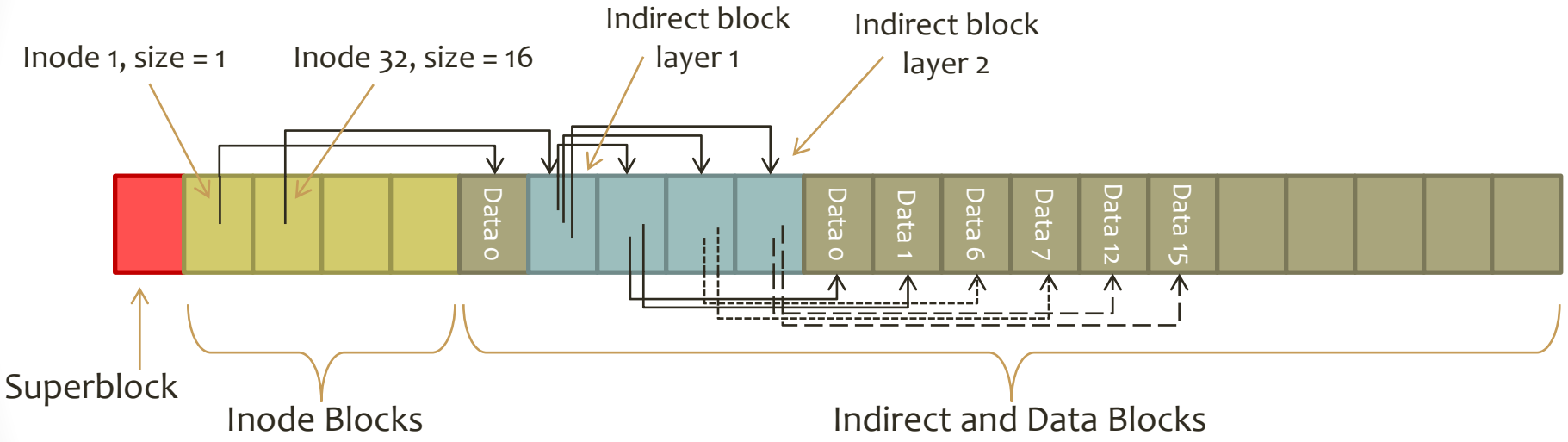


More Details

- Branching factor: number of `block_nos` that fit in a block
- Empty blocks: use block number 0
- If size = 1, no indirect blocks: root points to single block

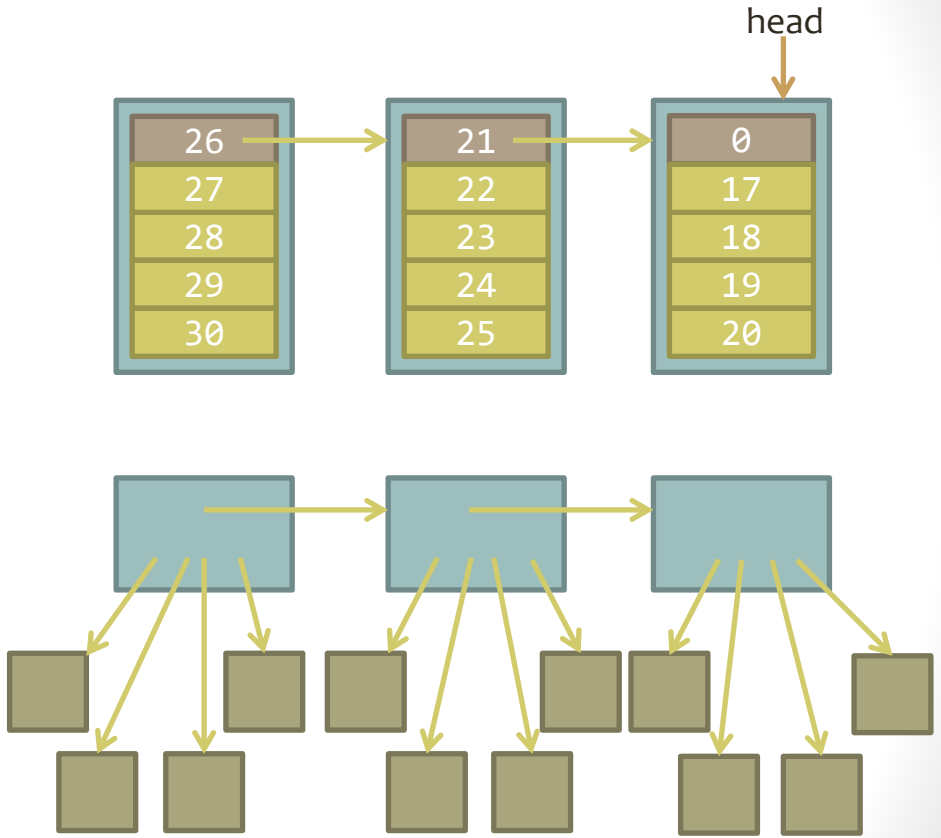


Tree Blocks on Disk



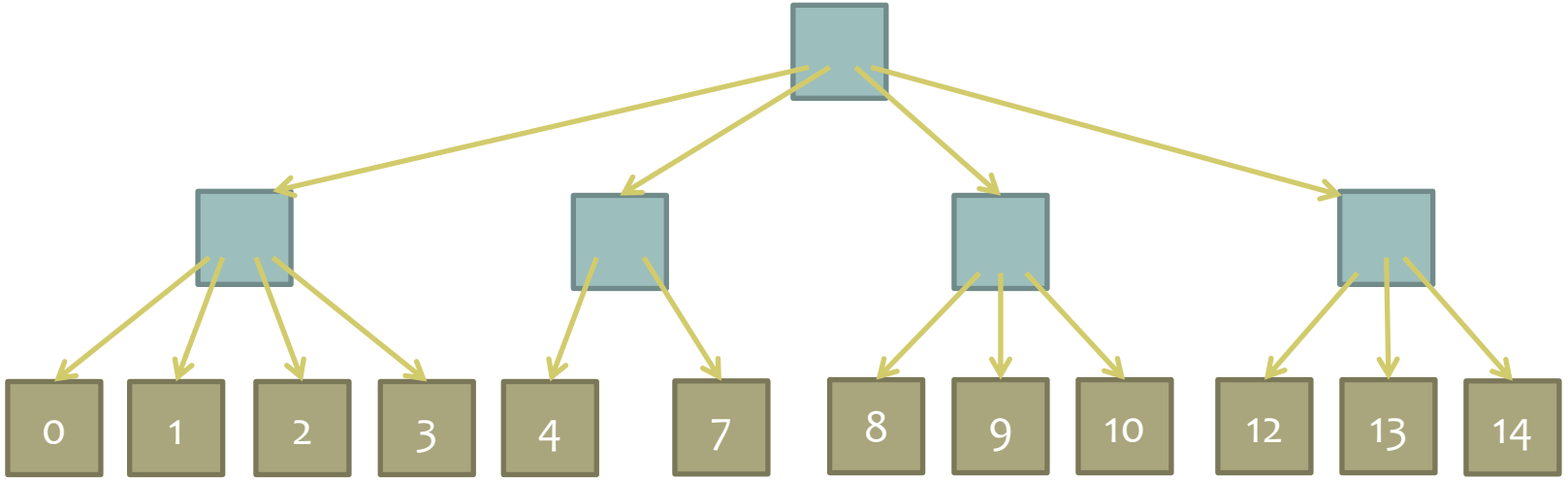
The Free List

- Linked list of “indirect blocks”
- Superblock stores head
- Used as a stack: take & add free blocks from head
- Initially all data blocks are on free list



TreeDisk Read

- Get root indirect block by reading inode
- Search down tree to find block with specified index



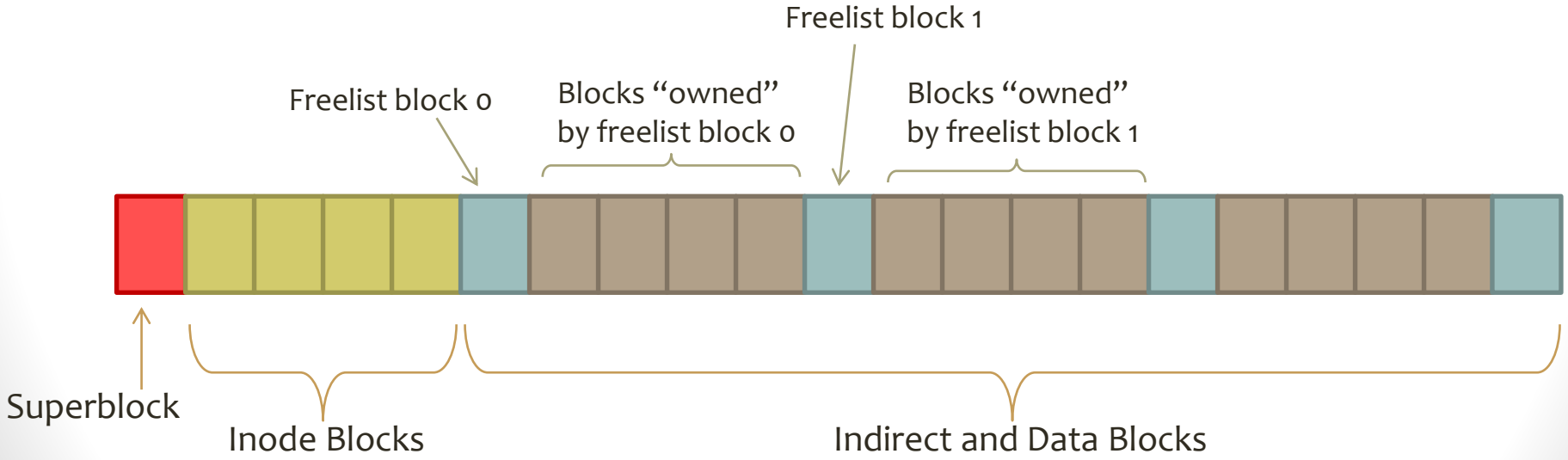
- Since data blocks are always "sorted," a simple *n*-ary search

TreeDisk Write

- Determine if write will require tree to grow
  - A VBS with *b* blocks and *n*-ary trees will require $\log_n b$ levels, rounded up
- Add a new level if necessary by adding indirect block at root
  - Former root becomes leftmost child of new root, all other children are empty
- Traverse the tree to the specified block number, creating new indirect blocks if necessary to fill holes
- When adding new blocks, take them from the free list

TreeDisk Create

- Compute number of inode blocks for # of inodes
- Initialize superblock and inode blocks
- Put all data blocks on free list



Outline

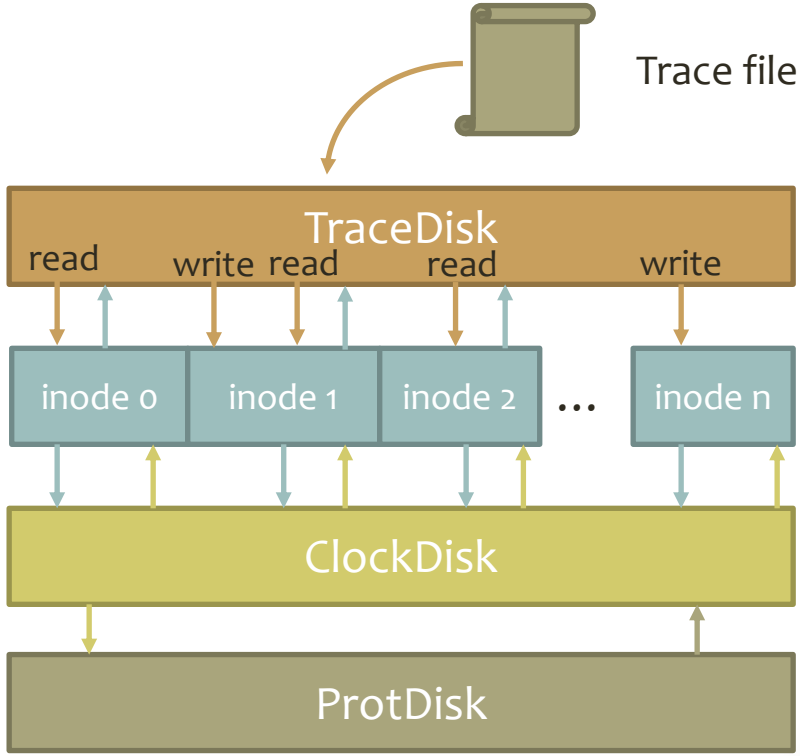
- TreeDisk design
- **TraceDisk and Traces**
- Using TraceDisk to test and debug

Testing a Block Store

- How do you know if a block store works?
- Throw the whole OS at it and see if it boots?
- Block store API:
  - `read(ino, offset)`
  - `write(ino, offset)`
  - `getsize(ino)`
  - `setsize(ino, size)`
  - `sync(ino)`
- What if we could test these operations individually?

TraceDisk: Scriptable Block Store

- TraceDisk is top layer instead of BFS
- Calls read, write, etc. operations on layer below based on script in an input file
- Can run outside EGOS



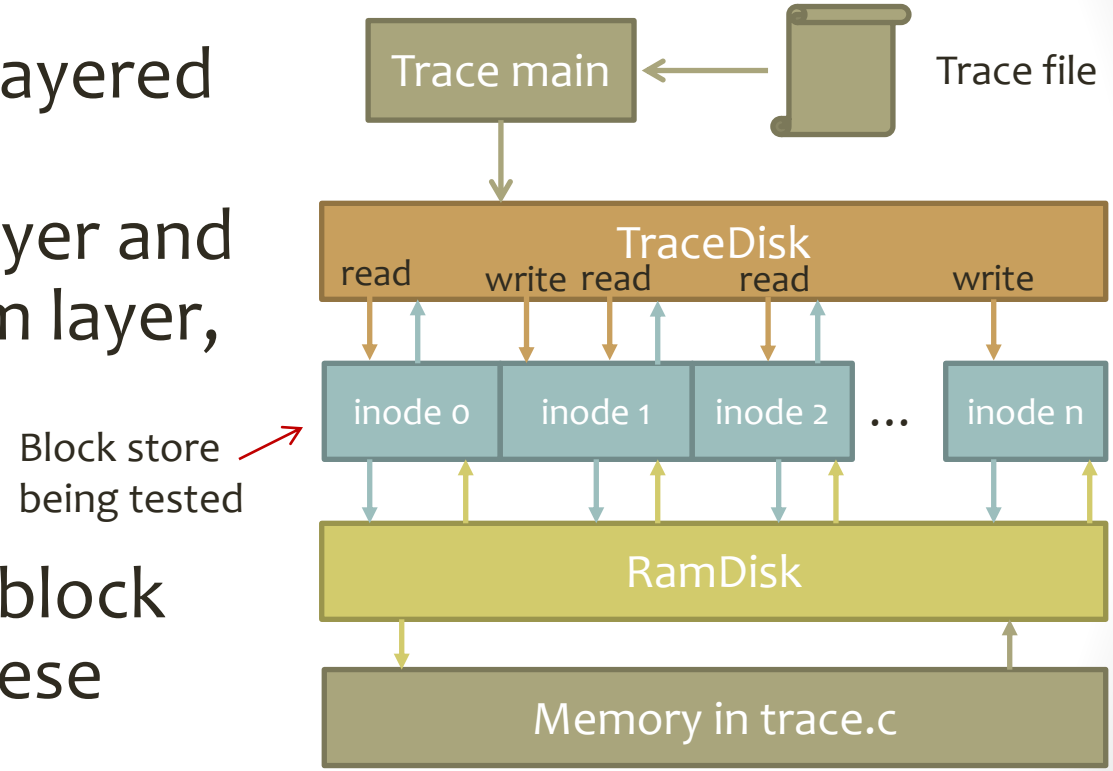
Trace File Format

- command:inode:block:[data]
- W – write integer value to the block
- R – read the block, compare its data to the expected value
- S – call setsize on the inode, setting it to the specified number of blocks
- G – call getsize on the inode, compare the result to the expected number of blocks
- F – call sync on the inode

```
W:1:0:666
W:2:0:999
W:2:1:3768
R:1:0:666
R:2:0:999
W:2:0:8765
W:2:1:1342
G:2:2
S:2:0
F:-1
```

The Trace Program

- Basic idea: Set up layered block storage with TraceDisk as top layer and RamDisk as bottom layer, run test, exit
- Can put any other block stores between these



Other Helpful Block Stores

- CheckDisk: Saves a copy of every block written
- On read, compares result of below->read to cached copy of block

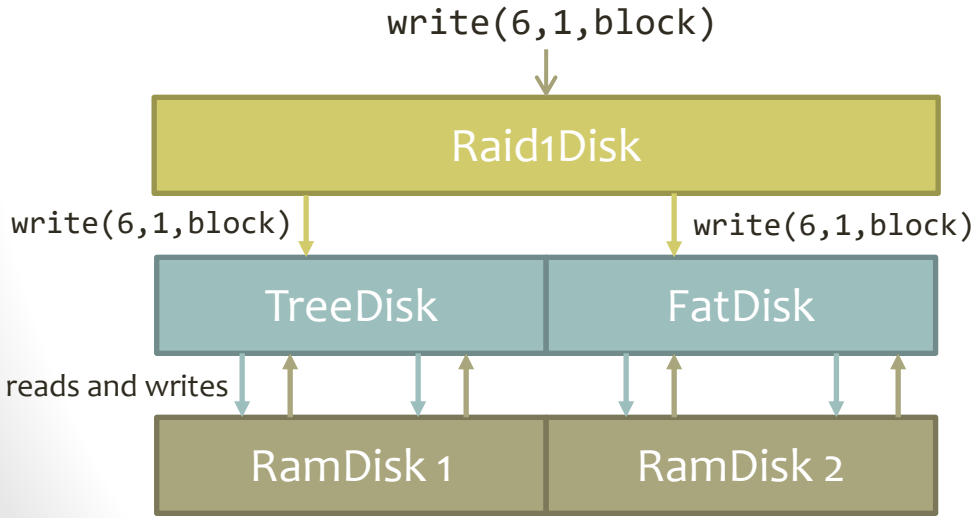
```
struct block_list {
  struct block_list *next;
  unsigned int ino;
  block_no offset;
  block_t block;
};
```

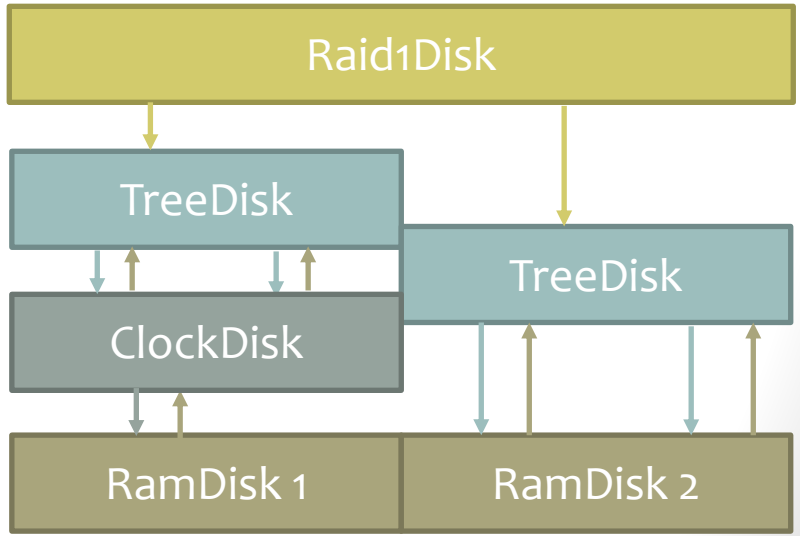
```
struct checkdisk_state {
  block_store_t *below;
  const char *descr;
  struct block_list *bl;
};
```

```
//in checkdisk_read…
(*cs->below->read)(cs->below, ino, offset, block)
struct block_list *bl;
for (bl = cs->bl; bl != 0; bl = bl->next) {
  if (bl->ino == ino && bl->offset == offset) {
    if (memcmp(&bl->block, block, BLOCK_SIZE) != 0) {
      fprintf(stderr, "!!CHKDISK %s: checkdisk_read:
corrupted\n\r", cs->descr);
      exit(1);
    }
    return 0;
  }
}
```

Other Helpful Block Stores

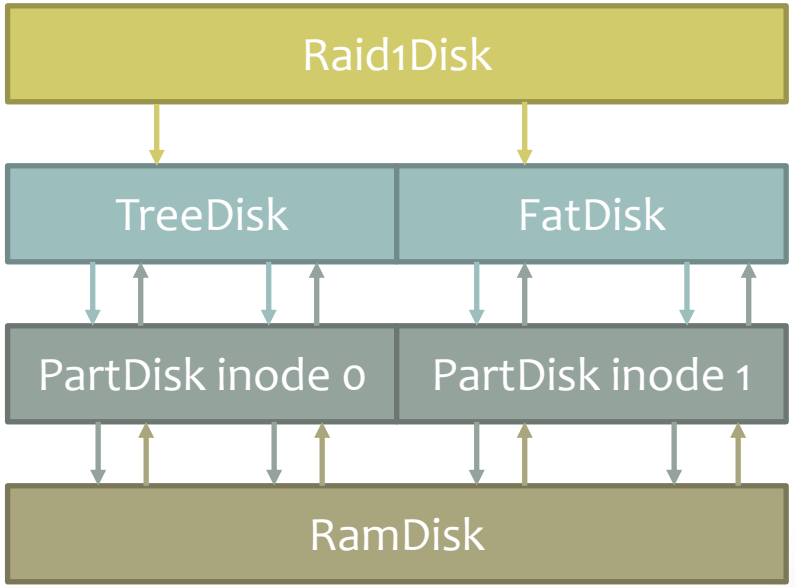
- Raid1Disk: mirrors operations onto two lower disks – run same trace on two block stores in parallel
- Compare filesystems, or with/without cache





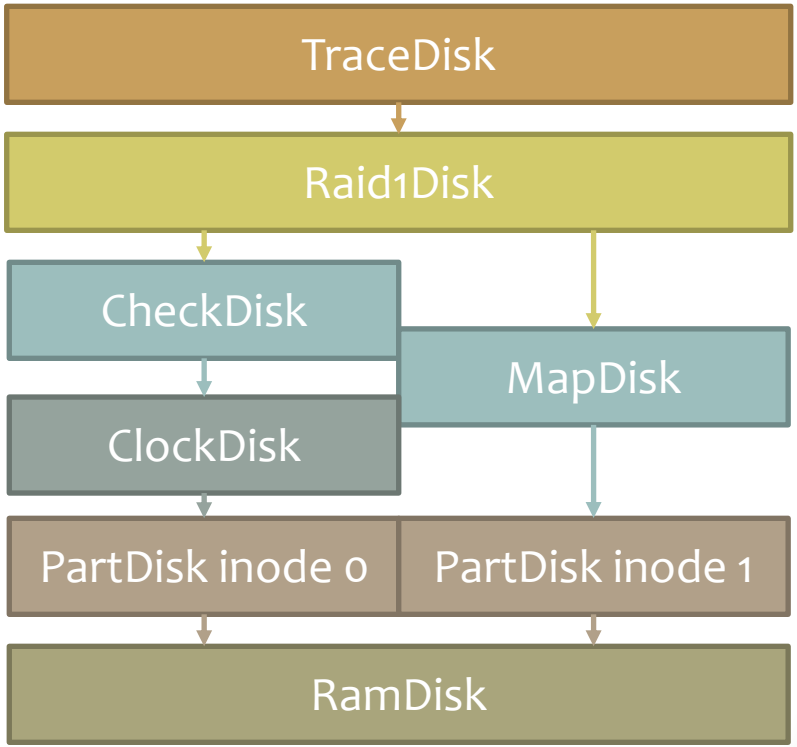
Other Helpful Block Stores

- PartDisk: Adds partitions to a disk
- Splits up a single-inode block store (e.g. RamDisk, ProtDisk) into several fixed-size inodes



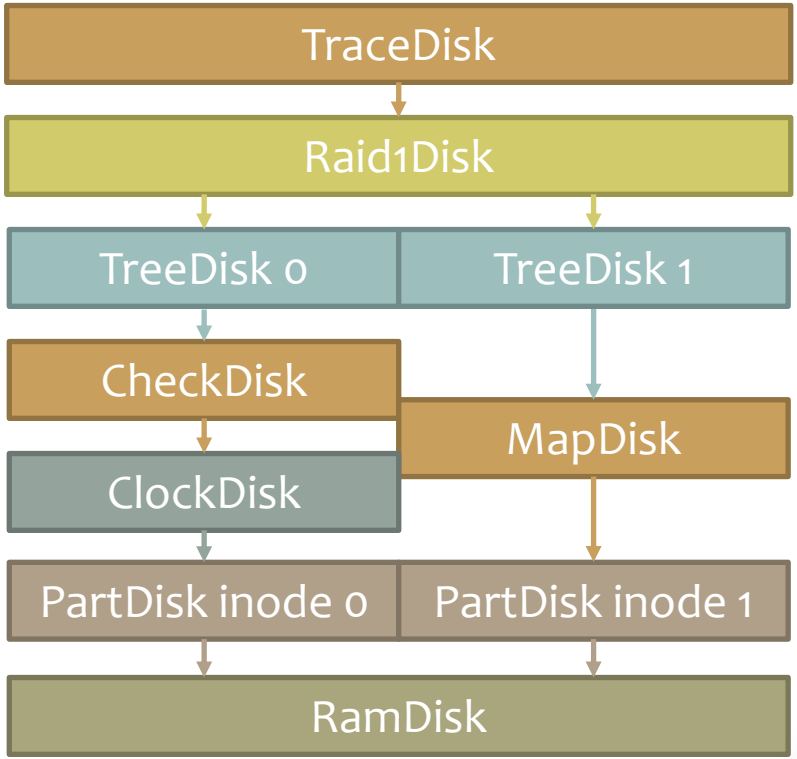
Trace Configurations

- "raid1": No filesystem, just 1 disk with cache and another disk without
- MapDisk: remaps inode 0 (in incoming commands) to inode x (on blockstore below)



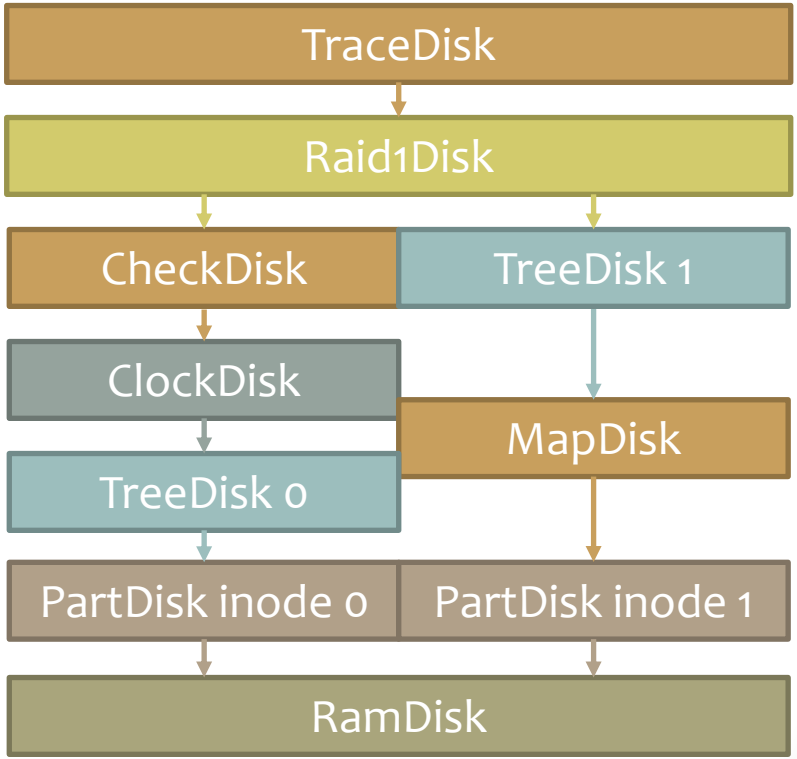
Trace Configurations

- "raid1+tree+cache": 2 copies of TreeDisk, 1 with cache, 1 without
- Now that TreeDisk can use an inode other than 0, could get rid of MapDisk here



Trace Configurations

- "raid1+cache+tree": almost the same, except cache goes *above* TreeDisk
- This is why cache can't make assumptions about what's above/below



Outline

- TreeDisk design
- TraceDisk and Traces
- **Using TraceDisk to test and debug**

Building and Running Trace

- Make directory test/cache_test/ parallel to src/
- Put trace.c and tracedisk.c in test/cache_test/
- Call `make cache_test`
- Program `trace` will now be in your root directory
- Run it:
  `./trace [-wt] config-string path/to/trace.txt`

Testing Your FatDisk

- Add fatdisk.c to the sources in src/make/Makefile.cache_test:

```
SRC = test/cache_test/tracedisk.c src/block/checkdisk.c
src/block/clockdisk.c … src/block/fatdisk.c
```

- Create a new "configuration" in trace.c that uses FatDisk, or edit an existing one to use FatDisk instead of TreeDisk:

```
if (fatdisk_create(xcdisk, 0, MAX_INODES) < 0) {
  panic("trace: can't create fatdisk file system");
}
block_store_t *fdisk = fatdisk_init(xcdisk, 0);
```

Debugging with Trace

- Trace is a “normal” C program, so you can debug it with GDB
- Remember to comment out this line in trace.c:
  `alarm(5)`
  - This sets an alarm to interrupt the program if it gets “stuck,” but debugging will intentionally freeze the program
- Set a breakpoint on fatdisk functions to stop when control reaches your “layer”

TraceDisk Demo