Example

```bash
la13@en-cs-cisugcl10:~$ cd example
la13@en-cs-cisugcl10:~/example$ ls
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366 . 391230414 ..
la13@en-cs-cisugcl10:~/example$ echo ezra > cornell
la13@en-cs-cisugcl10:~/example$ cat cornell
ezra
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366 . 391230414 .. 392852368 cornell
la13@en-cs-cisugcl10:~/example$ ln cornell bigred
da13@en-cs-cisugcl10:~/example$ cat bigred
ezra
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred 392852368 cornell
```
Example

```
la13@en-cs-cisugcl10:~$ cd example
la13@en-cs-cisugcl10:~/example$ ls
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366 391230414 ..
```
Example
Example

```bash
~/$ cd example
~/example$ ls
~/example$ ls -ai
~/example$ echo ezra > cornell
~/example$ cat cornell ezra
~/example$ ls -ai
~/example$ ln cornell bigred
~/example$ cat bigred ezra
~/example$ ls -i
~/example$ ln bigred ../../../bestivy
~/example$ ls -i
~/example$ cd ...
~/example$ cd example
```
Example

```
la13@en-cs-cisugcl10:~$ cd example
la13@en-cs-cisugcl10:~/example$ ls
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366  391230414 ...
la13@en-cs-cisugcl10:~/example$ echo ezra > cornell
la13@en-cs-cisugcl10:~/example$ cat cornell ezra
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366  391230414  392852368 cornell
la13@en-cs-cisugcl10:~/example$ ln cornell bigred
la13@en-cs-cisugcl10:~/example$ cat bigred ezra
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred  392852368 cornell
la13@en-cs-cisugcl10:~/example$ ln bigred ../bestivy
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred  392852368 cornell
la13@en-cs-cisugcl10:~/example$ cd ..
la13@en-cs-cisugcl10:~/example$ cd ..
la13@en-cs-cisugcl10:~$ ls -i
392852368 bestivy  398842589 CS4410-2020sp-A4  392852366 example
la13@en-cs-cisugcl10:~$ cd example
la13@en-cs-cisugcl10:~/example$ cd ...
```
Example

```bash
la13@en-cs-cisugcl10:~$ cd example
la13@en-cs-cisugcl10:~/example$ ls
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366  391230414 ...
la13@en-cs-cisugcl10:~/example$ echo ezra > cornell
la13@en-cs-cisugcl10:~/example$ cat cornell ezra
la13@en-cs-cisugcl10:~/example$ ls -ai
392852366  391230414  392852368 cornell
la13@en-cs-cisugcl10:~/example$ ln cornell bigred
da13@en-cs-cisugcl10:~/example$ cat bigred ezra
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred 392852368 cornell
la13@en-cs-cisugcl10:~/example$ rm cornell
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred 392852368 cornell
la13@en-cs-cisugcl10:~/example$ ls -i
392852368 bigred 392852368 cornell
```
Symbolic (Soft) links

More flexible than hard links
- can link to a directory
- can link to files in another volume

A map between pathnames
- to link newpathname to existingpathname for file inode1:
  - create a hard link between newpathname and new file inode2
  - store in inode2 the existingpathname for inode1
- so, a symbolic link is really a file (inode2 in our example) of a third type
  - neither a regular file nor a directory

Created using `ln`, but with the `-s` flag
Example

```bash
la13@en-cs-cisugcl05:~$ cd example
la13@en-cs-cisugcl05:~/example$ echo ezra > cornell
la13@en-cs-cisugcl05:~/example$ ls -i
392852367 cornell
```
Example

```bash
la13@en-cs-cisugcl05:~$ cd example
la13@en-cs-cisugcl05:/example$ echo ezra > cornell
la13@en-cs-cisugcl05:/example$ ls -i
392852367 cornell
la13@en-cs-cisugcl05:/example$ ln cornell bigred
la13@en-cs-cisugcl05:/example$ ls -i
392852367 bigred  392852367 cornell
```
Example

```
la13@en-cs-cisugc105:~$ cd example
la13@en-cs-cisugc105:~/example$ echo ezra > cornell
la13@en-cs-cisugc105:~/example$ ls -i
392852367 cornell
la13@en-cs-cisugc105:~/example$ ln cornell bigred
la13@en-cs-cisugc105:~/example$ ls -i
392852367 bigred 392852367 cornell
la13@en-cs-cisugc105:~/example$ cd ..
la13@en-cs-cisugc105:~/example$ ln example/cornell bestivy
la13@en-cs-cisugc105:~/example$ ln -s example/cornell highabove
la13@en-cs-cisugc105:~/example$ ls -i
392852367 bestivy 398842589 CS4410-2020sp-A4 392852366 example 392971138 highabove
```
Example
Example

```
la13@en-cs-cisugcl05:~$ cd example
la13@en-cs-cisugcl05:~/example$ echo ezra > cornell
la13@en-cs-cisugcl05:~/example$ ls -i
392852367 cornell
la13@en-cs-cisugcl05:~/example$ ln cornell bigred
la13@en-cs-cisugcl05:~/example$ ls -i
392852367 bigred 392852367 cornell
la13@en-cs-cisugcl05:~/example$ cd ..
la13@en-cs-cisugcl05:~$ ln example/cornell bestivy
la13@en-cs-cisugcl05:~$ ln -s example/cornell highabove
la13@en-cs-cisugcl05:~$ ls -i
392852367 bestivy 398842589 CS4410-2020sp-A4 392852366 example 392971138 highabove
la13@en-cs-cisugcl05:~$ ls -l
total 8
-rw-r--r-- 3 la13 pug-la13 5 Apr 28 23:03 bestivy
drwxr-sr-x 4 la13 pug-la13 4096 Apr 27 11:55 CS4410-2020sp-A4
drwxr-sr-x 2 la13 pug-la13 4096 Apr 28 23:03 example
lrwxrwxrwx 1 la13 pug-la13 15 Apr 28 23:04 highabove -> example/cornell
la13@en-cs-cisugcl05:~$ cat bestivy
ezra
la13@en-cs-cisugcl05:~$ cat highabove
ezra
```
Example

```
la13@en-cs-cisugcl05:~$ cd example
la13@en-cs-cisugcl05:~/example$ echo ezra > cornell
la13@en-cs-cisugcl05:~/example$ ls -i
392852367 cornell
la13@en-cs-cisugcl05:~/example$ ln cornell bigred
la13@en-cs-cisugcl05:~/example$ ls -i
392852367 bigred  392852367 cornell
la13@en-cs-cisugcl05:~/example$ cd ..
la13@en-cs-cisugcl05:~$ ln example/cornell bestivy
la13@en-cs-cisugcl05:~$ ln -s example/cornell highabove
la13@en-cs-cisugcl05:~$ ls -l
392852367 bestivy  398842589 CS4410-2020sp-A4  392852366 example  392971138 highabove
la13@en-cs-cisugcl05:~$ ls -l
```

```
-rw-r--r-- 3 la13 pug-la13  5 Apr 28 23:03 bestivy
drwxr-xr-x 4 la13 pug-la13  4096 Apr 27 11:55 CS4410-2020sp-A4
drwxr-xr-x 2 la13 pug-la13  4096 Apr 28 23:03 example
lwxrwxrwx 1 la13 pug-la13  15 Apr 28 23:04 highabove -> example/cornell
la13@en-cs-cisugcl05:~$ cat bestivy
ezra
la13@en-cs-cisugcl05:~$ cat highabove
ezra
la13@en-cs-cisugcl05:~$ rm example/cornell
```
Example

```
lal3@en-cs-cisugcl05:~$ cd example
lal3@en-cs-cisugcl05:/example$ echo ezra > cornell
lal3@en-cs-cisugcl05:/example$ ls -i
392852367 cornell
lal3@en-cs-cisugcl05:/example$ ln cornell bigred
lal3@en-cs-cisugcl05:/example$ ls -i
392852367 bigred 392852367 cornell
lal3@en-cs-cisugcl05:/example$ cd ..
lal3@en-cs-cisugcl05:~$ ls -i
392852367 bestivy 398842589 CS4410-2020sp-A4 392852366 example 392971138 highabove
lal3@en-cs-cisugcl05:~$ ls -l
total 8
-rw-r--r--  3 lal13  pug-lal13  5 Apr 28 23:03 bestivy
drwxr-xr-x  4 lal13  pug-lal13  4096 Apr 27 11:55 CS4410-2020sp-A4
drwxr-xr-x  2 lal13  pug-lal13  4096 Apr 28 23:03 example
lrwxrwxrwx  1 lal13  pug-lal13  15 Apr 28 23:04 highabove -> example/cornell
lal3@en-cs-cisugcl05:~$ cat bestivy
ezra
lal3@en-cs-cisugcl05:~$ cat highabove
ezra
lal3@en-cs-cisugcl05:~$ rm example/cornell
lal3@en-cs-cisugcl05:~$ cat bestivy
```
Example

```
la13@en-cs-cisugc105:~$ cd example
la13@en-cs-cisugc105:/example$ echo ezra > cornell
la13@en-cs-cisugc105:/example$ ls -i
392852367 cornell
la13@en-cs-cisugc105:/example$ ln cornell bigred
la13@en-cs-cisugc105:/example$ ls -i
392852367 bigred 392852367 cornell
la13@en-cs-cisugc105:/example$ cd ..
la13@en-cs-cisugc105:~/example/cornell$ ln example/cornell bestivy
la13@en-cs-cisugc105:~/example/cornell$ ls -s example/cornell highabove
la13@en-cs-cisugc105:~/example/cornell$ ls -l
392852367 bestivy 398842589 CS4410-2020sp-A4 392852366 example 392971138 highabove
la13@en-cs-cisugc105:~/example/cornell$ ls -l
```

```
total 8
-rw-r--r--  3 la13  pug-la13   5 Apr 28 23:03 bestivy
drwxr-xr-x  4 la13  pug-la13  4096 Apr 27 11:55 CS4410-2020sp-A4
drwxr-xr-x  2 la13  pug-la13  4096 Apr 28 23:03 example
-drwxr-xr-x  1 la13  pug-la13   15 Apr 28 23:04 highabove -> example/cornell
la13@en-cs-cisugc105:~/example/cornell$ cat bestivy
exra
la13@en-cs-cisugc105:~/example/cornell$ cat highabove
exra
la13@en-cs-cisugc105:~/example/cornell$ rm example/cornell
la13@en-cs-cisugc105:~/example/cornell$ cat bestivy
exra
la13@en-cs-cisugc105:~/example/cornell$ cat highabove
```

```
cat: highabove: No such file or directory
la13@en-cs-cisugc105:~/example/cornell$ ```
Permission Bits

File bestivy

- leading - says bestivy is a regular file
  - d is for directory; l is for soft link

- Next nine characters are permission bits
  - rwx for owner, group, everyone
    - owner can read and write; group and others can just read
    - x set in a regular file means file can be executed
    - x set in a directory that user/group/everybody is allow to cd to that directory
  - can be set using chmod
File System Layout

- File System is stored on disks
  - disk can be divided into one or more partitions
  - Sector 0 of disk: Master Boot Record (MBR). It contains:
    - bootstrap code (loaded and executed by firmware)
    - partition table (addresses of where partitions start & end)
  - First block of each partition has boot block
    - loaded by executing code in MBR and executed on boot

```
<table>
<thead>
<tr>
<th>MBR</th>
<th>Partition Table</th>
<th>PARTITION 1</th>
<th>PARTITION 2</th>
<th>PARTITION 3</th>
<th>PARTITION 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BOOT BLOCK</td>
<td>SUPERBLOCK</td>
<td>Free Space Mgmt</td>
<td>I-Nodes</td>
</tr>
</tbody>
</table>
```
Peeking Inside

Persistent storage modeled as a sequence of N blocks
- from 0 to N-1
  - in this example, 64 blocks, each 4KB
- some blocks store data
Peeking Inside

Persistent storage modeled as a sequence of N blocks
- from 0 to N-1
  - in this example, 64 blocks, each 4KB
- some blocks store data
- other blocks store metadata
  - an array of inodes
    - if an inode is 256 bytes, then 16 inodes per block.
      With 5 blocks for inodes, file system can have up to 80 files
Peeking Inside

- Persistent storage modeled as a sequence of $N$ blocks
  - from 0 to $N-1$
    - in this example, 64 blocks, each 4KB
  - some blocks store data
  - other blocks store metadata
    - an array of inodes
      - if an inode is 256 bytes, then 16 inodes per block.
        With 5 blocks for inodes, file system can have up to 80 files
Peeking Inside

Persistent storage modeled as a sequence of N blocks

- from 0 to N-1
  - in this example, 64 blocks, each 4KB
- some blocks store data
- other blocks store metadata (remember `stat()`?)
  - an array of inodes
    - if an inode is 256 bytes, then 16 inodes per block.
      With 5 blocks for inodes, file system can have up to 80 files
  - bitmaps tracking free inodes and data blocks;
Peeking Inside

- **Persistent storage modeled as a sequence of N blocks**
  - from 0 to N-1
  - in this example, 64 blocks, each 4KB
- some blocks store data
- other blocks store metadata (remember stat()?)
  - an array of inodes
    - if an inode is 256 bytes, then 16 inodes per block.
    - With 5 blocks for inodes, file system can have up to 80 files
  - bitmaps tracking free inodes and data blocks; Superblock; Boot block

![Diagram of file system storage with blocks labeled as free lists, inodes, data blocks, and other data structures.](image-url)
The Superblock

- One logical superblock per file system
  - at a well-known location
  - contains metadata about the file system, including
    - how many inodes
    - how many data blocks
    - where the inode table begins
    - may contain info to manage free inodes/data blocks
  - read first when mounting a file system
Files can be allocated in different ways

- **Contiguous allocation**
  - All bytes together, in order
- **Linked Structure**
  - Each points to the next block
- **Indexed Structure**
  - Index block, pointing to many other blocks

**Which is best?**

- For sequential access? Random access?
- Large files? Small files? Mixed?
Contiguous Allocation

- All bytes together, in order
  - **Simple:** only need start block and size
  - **Efficient:** one seek to read entire file
  - **Fragmentation:** external, and can be serious
  - **Usability:** User need to know file’s size at time of creation

Used in CD-ROM, DVDs
Linked List Allocation

- Each file is stored as a linked list of blocks
  - first word of each block points to next block
  - the rest of the block is data

- Space utilization: no external fragmentation
- Simplicity: only need to find first block of each file
- Performance: random access is slow
- Implementation: blocks mix data and metadata
Decouple data and metadata
- reduces disk seeks (and enables caching!)

Microsoft, late 70s
- still widely used today
  - thumb drives, camera cards, CD ROMs

File Allocation Table (FAT) FS

Metadata

Data

not to scale!
FAT File system

Index Structures
File Allocation Table (FAT)
- array of 32-bit entries
- one entry per block
- file represented as a linked list of FAT entries
- file # = index of first FAT entry

Free space map
- If data block i is free, then FAT[i] = 0
- find free blocks by scanning FAT

Locality heuristics
- As simple as next fit:
  □ scan sequentially from last allocated entry and return next free entry
- Can be improved through defragmentation

Directory
- Maps file name to FAT index

<table>
<thead>
<tr>
<th>Directory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>jack.txt</td>
<td>12</td>
</tr>
<tr>
<td>jill.txt</td>
<td>9</td>
</tr>
</tbody>
</table>

Data blocks
- file 9 block 3
- file 9 block 0
- file 9 block 1
- file 9 block 2
- file 12 block 0
- file 12 block 1
- file 9 block 4
FAT File system

Advantages
- simple!
  - per file, needs only start block
  - widely supported
  - no external fragmentation
  - no conflating data and metadata in the same block

Disadvantages
- Poor locality
  - many file seeks unless entire FAT in memory
  - 1 TB \(2^{40}\) bytes disk, 4kb \(2^{12}\) bytes block, \(2^{28}\) FAT entries; at 4B/entry, 1 GB (!)
- Poor random access
  - needs sequential traversal
- Limited access control
  - no file owner or group ID
  - any user can read/write any file
- No support for hard links
- Volume and file size are limited
  - FAT entry is 32 bits, but top 4 are reserved
  - no more than \(2^{28}\) blocks
  - with 4kB blocks, at most 1TB FS
  - file no bigger than 4GB
- No support for advanced reliability techniques
File System Layout
Tree-based Multi-level Index

- UFS (Unix File System) (Ken Thompson, 1969)
- 4.2 BSD FFS (Fast File System) (McKusick, Joy, Leffler, Fabry, 1983)

Diagram:
- Super block
- Includes location of free data blocks, free inodes
- i-node blocks storing an array of i-nodes
- Data blocks
Multilevel index

Inode Array

- at known location on disk
- file number = inode number = index in the array
File structure

Each file is a **fixed, asymmetric tree**, with fixed size data blocks (e.g. 4KB) as its leaves.

The root of the tree is the file’s **inode**, containing:

- metadata (more about it later)
- a set of 15 pointers
  - first 12 point to data blocks
  - last three point to intermediate blocks, themselves containing pointers...
    - #13: pointer to a block containing pointers to data blocks
    - #14: double indirect pointer
    - #15: triple indirect pointer (!)
Multilevel index

Inode Array

- at known location on disk
- file number = inode number = index in the array

I-node

- File Metadata
- 4 Bytes entries
- 12 x 4KB = 48KB

Data blocks

- indirect block contains pointers to data blocks
- double indirect block contains pointers to indirect blocks
- triple indirect block contains pointers to double indirect blocks

Data blocks

- 1K x 4KB = 4MB
- 1K x 1K x 4KB = 4GB
- 1K x 1K x 1K x 4KB = 4TB
Multilevel index: key ideas

- Tree structure
  - efficient in finding blocks

- High degree
  - efficient in sequential reads
    - once an indirect block is read, can read 100s of data block

- Fixed structure
  - simple to implement

- Asymmetric
  - supports large files
  - small files don’t pay large overheads
Good for small files...

If instead all blocks were accessed through a 3-level index, a file occupying a single 4KB block would require 16 KB:

- a triple indirect block
- a double indirect block
- an indirect block
- the 4KB data block
- reading would require reading 5 blocks to traverse the tree
...and for sparse files

Consider file `sparse.dat` with two 4K blocks: one at offset 0; the other at offset $2^{30}$

What is the file's size? `ls -lgGh sparse.dat`: 1.1 GB

What about disk usage? `du -hs sparse.dat`: 16 KB (!)

Reading from hole? FS creates a 0-filled buffer

Writing to hole? FS allocates storage blocks for data + any required indirect block
What else is in an i-node?

- **Type**
  - ordinary file
  - directory
  - symbolic link
  - special device

- **Size of the file (in bytes)**

- **No. of links to the i-node**

- **Owner (user id & group id)**

- **Protection bits**

- **Times: creation, last accessed, last modified**
A file that contains a collection of mapping from file name to file inode

To look up a file, find the directory that contains the mapping to the file’s inode

To find that directory, find the parent directory that contains the mapping to that directory’s inode.

Good news: root directory has well-known number (2)
Looking up a file

Find file /Users/lorenzo/griso.jpg

file 2 "/

bin 438
ger 782
Users 256

file 256 "/Users"

file 1061 "/Users/lorenzo"

Documents 394
Music 416
griso.jpg 864

file 864 "/Users/lorenzo/griso.jpg"
Directory Layout

- Directory stored as a file
  - Linear search to find filename (small directories)

File 1061
/Users/lorenzo

Larger directories use B trees
  - searched by hash of file name
Reading a File

First, must open the file
- open("/CS4410/roster", O_RDONLY)
- Follow the directory tree, until we get to the inode for “roster”
- Read that inode
  - do a permission check
  - return a file descriptor fd

Then, for each read() that is issued:
- read inode
- read appropriate data block (depending on offset)
- update last access time in inode
- update file offset in in-memory open file table for fd
### Read first 3 data blocks from `/CS4410/roster`

<table>
<thead>
<tr>
<th></th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>CS4410 inode</th>
<th>roster inode</th>
<th>root data</th>
<th>CS4410 data</th>
<th>roster data[0]</th>
<th>roster data[1]</th>
<th>roster data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>open(CS4410)</td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
</tr>
<tr>
<td>write()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
<td></td>
</tr>
<tr>
<td>write()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>write()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>read()</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>read()</td>
</tr>
</tbody>
</table>
Writing a File

Must open the file, like before

But now may have to allocate a new data block
  - each logical write can generate up to five I/O ops
    - reading the free data block bitmap
    - writing the free data block bitmap
    - reading the file’s inode
    - writing the file’s inode to include pointer to the new block
    - writing the new data block

Creating a file is even worse!
  - read and write free inode bitmap
  - write inode
  - (read) and write directory data
  - write directory inode
### Create `/CS4410/roster` & Write first 3 Data Blocks

<table>
<thead>
<tr>
<th>create (/CS4410/roster)</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>CS4410 inode</th>
<th>roster inode</th>
<th>root data</th>
<th>CS4410 data</th>
<th>roster data[0]</th>
<th>roster data[1]</th>
<th>roster data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>create (/CS4410/roster)</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>CS4410 inode</th>
<th>roster inode</th>
<th>root data</th>
<th>CS4410 data</th>
<th>roster data[0]</th>
<th>roster data[1]</th>
<th>roster data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>create (/CS4410/roster)</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>CS4410 inode</th>
<th>roster inode</th>
<th>root data</th>
<th>CS4410 data</th>
<th>roster data[0]</th>
<th>roster data[1]</th>
<th>roster data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>create (/CS4410/roster)</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>CS4410 inode</th>
<th>roster inode</th>
<th>root data</th>
<th>CS4410 data</th>
<th>roster data[0]</th>
<th>roster data[1]</th>
<th>roster data[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Code Snippet

```python
create('/CS4410/roster')
write()  # First Data Block
write()  # Second Data Block
write()  # Third Data Block
```
Caching

Reading a long path can cause a lot of I/O ops!

Cache aggressively!
- early days: fixed sized cache for popular blocks
  - static partitioning can be wasteful
- current: dynamic partitioning via unified page cache
  - virtual memory pages and file system blocks in a single cache

Caching can significantly reduce disk I/O for reads

Buffering can reduce cost of writes
- some blocks may be overwritten
- batching helps with scheduling disk accesses
BSD FFS: Fast File System

- UFS treats disks as if they were RAM
  - files grab first free data block: seeks and fragmentation
- FFS optimizes file system layout for how disks work
- Smart locality heuristics
  - block group placement
    - optimizes placement for when a file data and metadata, and other files within same directory, are accessed together
  - reserved space
    - gives up about 10% of storage to allow flexibility needed to achieve locality
Locality heuristics: block group placement

- **Divide disk in block groups**
  - sets of nearby tracks

- **Distribute metadata**
  - old design: free space bitmap and inode map in a single contiguous region
    - lots of seeks when going from reading metadata to reading data
  - FFS: distribute free space bitmap and inode array among block groups. Keep a superblock copy in each block group

- **File Placement**
  - when a new regular file is created, FFS looks for inodes in the same block as the file’s directory
  - when a new directory is created, FFS places it in a different block from the parent’s directory

- **Data Placement**
  - first free heuristics
  - trade short term for long term locality
Locality heuristics: block group placement

- Divide disk in **block groups**
  - sets of nearby tracks

- Distribute metadata
  - old design: free space bitmap and inode map in a single contiguous region
    - lots of seeks when going from reading metadata to reading data
  - FFS: distribute free space bitmap and inode array among block groups. Keep a superblock copy in each block group

- File Placement
  - when a new regular file is created, FFS looks for inodes in the same block as the file’s directory
  - when a new directory is created, FFS places it in a different block from the parent’s directory

- Data Placement
  - first free heuristics
  - trade short term for long term locality
Locality heuristics: block group placement

- Divide disk in **block groups**
  - sets of nearby tracks

- Distribute metadata
  - old design: free space bitmap and inode map in a single contiguous region
    - lots of seeks when going from reading metadata to reading data
  - FFS: distribute free space bitmap and inode array among block groups. Keep a superblock copy in each block group

- File Placement
  - when a new regular file is created, FFS looks for inodes in the same block as the file’s directory
  - when a new directory is created, FFS places it in a different block from the parent’s directory

- Data Placement
  - first free heuristics
  - trade short term for long term locality
Locality heuristics: block group placement

- Divide disk in block groups
  - sets of nearby tracks

- Distribute metadata
  - old design: free space bitmap and inode map in a single contiguous region
    - lots of seeks when going from reading metadata to reading data
  - FFS: distribute free space bitmap and inode array among block groups. Keep a superblock copy in each block group

- File Placement
  - when a new regular file is created, FFS looks for inodes in the same block as the file's directory
  - when a new directory is created, FFS places it in a different block from the parent's directory

- Data Placement
  - first free heuristics
  - trade short term for long term locality
Locality heuristics: reserved space

- When a disk is full, hard to optimize locality
  - file may end up scattered through disk

- FFS presents applications with a smaller disk
  - about 10%-20% smaller
  - user’s write that encroaches on reserved space fails
  - super user still able to allocate inodes to clean things up
Long File Exception

Blocks of a huge file not all in the same block group
- or they will eat up all the blocks in the group!
- Instead, 12 blocks in a group (direct index)
- others divided in “chunks”

Locality lost when moving between chunks
- choose chunk size to amortize cost of seeks

Say we want 90% of peak transfer, and transfer rate is 40MB/s
- if positioning time (seek+rotation) is 10ms, we need a chunk large enough that transfer takes 90ms

\[
\text{chunk size} = \frac{40\text{MB}}{s} \times \frac{1\text{s}}{1000\text{ms}} \times 90\text{ms} = 3.6\text{ MB}
\]

In practice, FFS uses 4 MB chunks