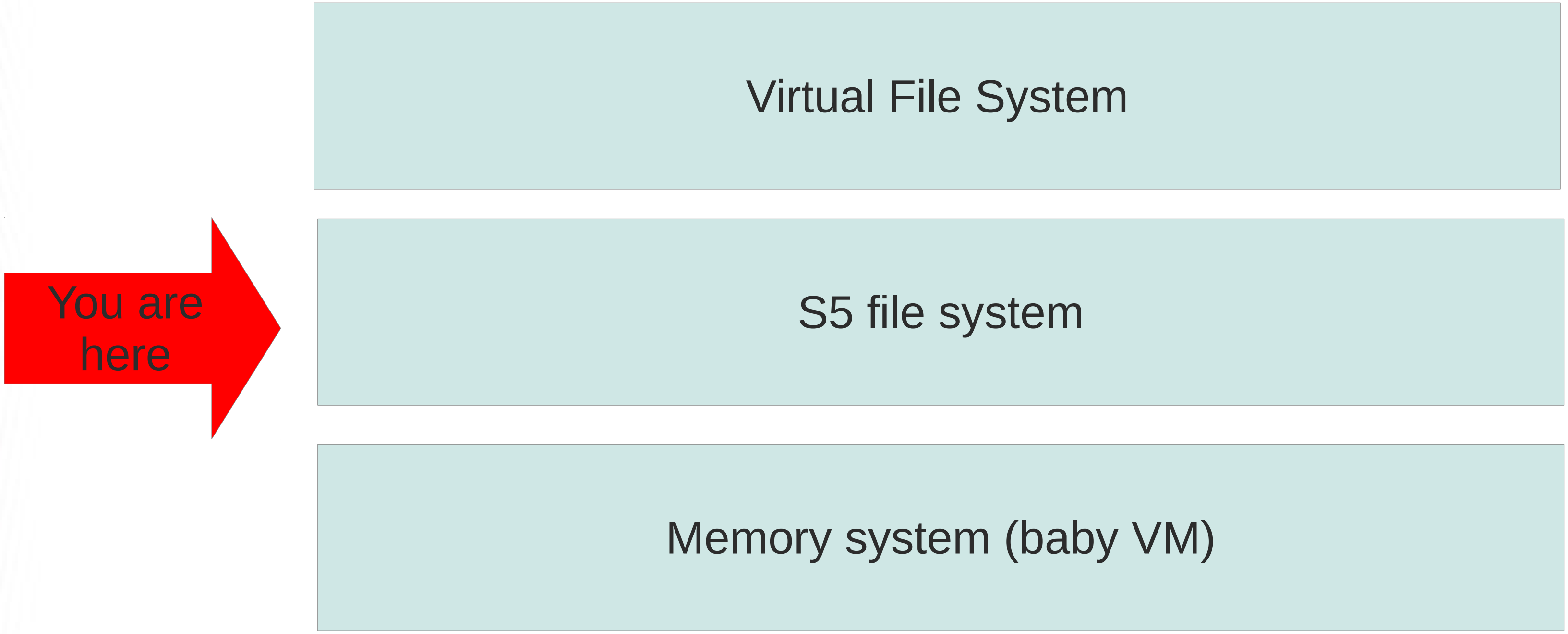
USC CSCI 402x
Assignment 3

Ted Faber
faber@isi.edu

Implement an i-node file system

- Manage disk data structures
  - Free list
  - I-nodes
  - Data blocks
- Implement system calls
  - Read/write
  - Directory ops: link, unlink, new files/devices/directories
- Debugging - kshell

Living in a Real OS



Virtual File System

You are here

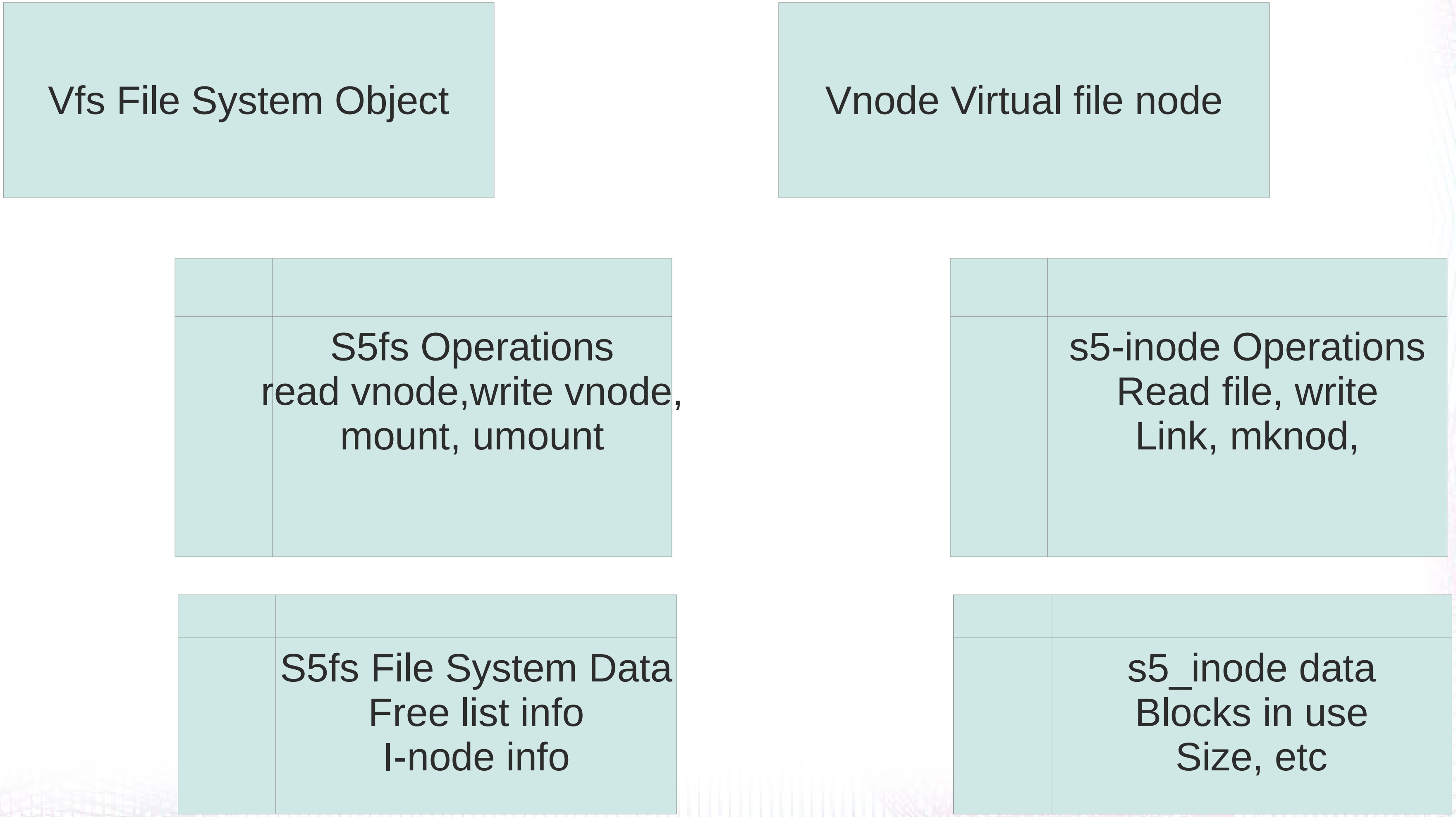
S5 file system

Memory system (baby VM)

Goal: build a working on-disk file system

- Integrate with VFS
  - Implement specializations for i-node disk-based file system
  - Called s5fs (System V File System)
- Integrate with paging system
  - Implement routines to manage disk-based data structures
- Again, we'll give you acceptance tests
  - Step 1: get the kernel shell up and running

Virtual File System



Vfs File System Object

Vnode Virtual file node

S5fs Operations
read vnode,write vnode,
mount, umount

s5-inode Operations
Read file, write
Link, mknod,

S5fs File System Data
Free list info
I-node info

s5_inode data
Blocks in use
Size, etc

Virtual File System

- Good news:
  - Existing code to make syscalls
    - kernel/fs/vfs_syscalls.c (and others)
- Bad news
  - Integration of new code with this somewhat complex system

Virtual File System

- Good news:
  - Existing code to make syscalls
    - kernel/fs/vfs_syscalls.c (and others)
  - We give you a valid disk to start
- Bad news
  - Integration of new code with this somewhat complex system

S5 file system superblock

```
/* The contents of the superblock, as stored on disk. */
typedef struct s5_super {
        uint32_t s5s_magic;             /* the magic number */
        uint32_t s5s_free_inode;          /* the free inode pointer */
        uint32_t s5s_nfree;             /* number of blocks currently in
                                         * s5s_free_blocks */
        /** First "node" of free block list */
        uint32_t s5s_free_blocks[S5_NBLKS_PER_FNODE];

        uint32_t s5s_root_inode;          /* root inode */
        uint32_t s5s_num_inodes;            /* number of inodes */
        uint32_t s5s_version;           /* version of this disk format */
} s5_super_t;
```

Where do I get a superblock?

- s5fs_mount()
  - In kernel/fs/s5fs/s5fs.c
  - Called from the new idle to mount the s5fs
  - Actual disk read isn't working yet
- That code also initializes
  - vfs data structures (including pointer to superblock)
  - vfs function table

S5 inodes

```
/* The contents of an inode, as stored on disk. */
typedef struct s5_inode {
        union {
                uint32_t s5_next_free; /* inode free list ptr */
                uint32_t s5_size;      /* file size */
        } s5_un;
#define       s5_next_free s5_un.s5_next_free
#define       s5_size      s5_un.s5_size
        uint32_t   s5_number;              /* this inode's number */
        uint16_t   s5_type;         /* one of S5_TYPE_{FREE,DATA,DIR} */
        int16_t    s5_linkcount;    /* link count of this inode */
        uint32_t   s5_direct_blocks[S5_NDIRECT_BLOCKS];
        uint32_t   s5_indirect_block;
} s5_inode_t;
```

Where do I get an i-node

- V-node routine vget calls s5fs_read_vnode
  - You get to write this
  - Read the i-node from disk and return pointer
- vget is called from s5fs_mount
  - Once you make s5fs_mount work
    - Superblock is read
    - I-node's are readable

Reading and Writing the disk through memory objects

- Manage set of pages (VM-like pages)
  - Pages associated with object
- Operations
  - Fillpage – put data into page (read)
  - Cleanpage – put data on disk (write)
  - Dirtypage - Allocate space for page in fs
- Examples:
  - Disk device implemented
  - V-node – you'll implement s5fs specializations

Pages

- Fixed size memory associated with memory object
  - All data from or going to disk is in a page
    - superblock
    - Data blocks
    - I-nodes (*not* v-nodes!)
- Sent to disk at shutdown or memory pressure
  - All data is write behind

Page frame

```
/* A pframe structure represents a page frame in physical memory available to the
 * kernel. pframes are managed by mmobjs */
typedef struct pframe {
        /* Public read: (do not modify outside pframe.c) */

        /* Object and page number, which together uniquely identify the page */
        struct mmobj       *pf_obj;
        uint32_t            pf_pagenum;

        /*   The address of the page frame. Note that this is NOT a
         *   physical address, but is a virtual address in the kernel's memory
         *   map (i.e., it will be higher than 0xc0000000) */
        void               *pf_addr;

        /* Private: */
        uint8_t             pf_flags;    /* PF_DIRTY, PF_BUSY */
        ktqueue_t           pf_waitq;    /* wait on this if page is busy */
        int                 pf_pincount;
        list_link_t         pf_link;     /* link on {free,allocated,pinned}_list */
        list_link_t         pf_hlink;    /* link on hash chain of resident page hash */
        list_link_t         pf_olink;    /* link on object's list of resident pages */
} pframe_t;
```

Page States

- Busy – moving to/from disk cannot use
- Dirty – memory copy changed
- Pinned – must stay in memory
  - This is different from general pinned description
  - Page may be pinned more than once
    - Keep pin count, unpinned at 0
  - You implement page pinning
    - kernel/mm/pframe.c

What To Implement: VM system

- kernel/mm/pframe.c
  - Page reading
  - Page pinning/unpinning
- kernel/fs/s5fs/s5fs.c – vnode ops for paging
  - Make a v-node a memory object
  - s5fs_fillpage, s5fs_cleanpage, s5fs_dirtypage

Paging: pframe_get

- Function: retrieve page given memory object and offset
  - Disk: logical disk block
  - v-node: file block
- Logic:
  - Look for copy in memory and return it
  - If low on memory ask pageoutd for more
  - Allocate page and fill it from memory object

Pframe_get: your toolchest

- pframe_get_resident() - returns the page if in memory
- pageoutd_needed() - true if low on mem
- pageoutd_wakeup() - start pageoutd running
  - Wait for it on alloc_waitq
- pframe_alloc() - get an unused page
- pframe_fill() - fill page from memory object
  - Works on disks now
  - You need to make it work for s5fs v-nodes

Page pinning

- 2 lists – alloc_list, pinned_list
  - Use same field in pframe_t so only on one list
- pf_pincount : pinned - unpinned
- pframe_pin()
  - Increment pf_pincount move to pinned_list if not there
- pframe_unpin()
  - Decrement pf_pincount, move to alloc_list is pf_pincount is 0

Making v-nodes into memory objects

- kernel/fs/s5fs/s5fs.c
  - s5fs_fillpage() - put data into page
  - s5fs_cleanpage() - put data onto disk
  - s5fs_dirtypage() - allocate space
- Note that s5fs v-nodes have their own memory objects too – a disk device

fillpage

- s5fs_fillpage(vnode_t *vnode, off_t offset, void *pagebuf)
  - Vnode is the node to read
  - Offset is where to start reading (page aligned)
  - Pagebuf is a page-sized destination
- Find disk block with data (s5_seek_to_block())
- If there is one, copy it out from disk (read_block)
- Otherwise copy zeroes out

Why the zeroes?

- Writing an empty file
  - Get page
  - Write data
- Get page will be confused if we don't fill with zeroes...
- When does that page get allocated in the i-node?

dirtypage

- Same signature as fillpage
- If no space allocated in i-node, get some because that page will be written
  - s5_alloc_block() - you'll write it
  - Whenever you manipulate i-node use s5_dirty_inode() to tell the paging system

cleanpage

- s5fs_cleanpage(vnode_t *vnode, off_t offset, void *pagebuf)
  - Vnode is the node to write
  - Offset is where to start writing (page aligned)
  - Pagebuf is a page-sized source
- Find destination disk block (s5_seek_to_block())
- Copy data out to disk (write_block)

What to implement: VFS

- vfs operations
  - s5fs_vnode_read (passed in a vnode)
    - Read inode data into page (and pin it)
    - Initialize vnode data structures
  - s5fs_delete_vnode
    - If linkcount is 0 free the i-node
    - Unpin the page

What to implement: v-nodes

- s5fs_read()
  - Move data from pages to buffer
- s5fs_write()
  - Move data from buffer into pages
  - Allocate pages if needed
  - Adjust size of needed (files only grow)

What to implement: v-nodes

- s5fs_create() - make an empty file
  - Allocate an i-node (pull one from disk and pin)
  - Add a directory entry
  - Make modified pages dirty
- s5fs_mknod() - Make a device
  - Do all of create and lookup device to link
  - You'll need this for the kernel_shell

Directories

```
/* The contents of a directory entry, as stored on disk. */
typedef struct s5_dirent {
        uint32_t   s5d_inode;
        char       s5d_name[S5_NAME_LEN];
} s5_dirent_t;
```

What to implement: v-nodes

- s5fs_lookup() - find a directory entry
  - If name is in the directory, return the vnode associated with it
- s5fs_link() - Make a link
  - Add a directory entry linking to the v-node
  - Dirty the file blocks (and maybe the i-node)
- s5fs_unlink() - remove a link (not a directory)
  - Remove the directory entry linking to the v-node
  - Reduce the link count
  - Dirty the file blocks (and maybe the i-node)

What to implement: v-nodes

- s5fs_mkdir() - make a directory entry
  - Allocate an i-node (pull one from disk and pin)
  - Add a directory entries
    - New directory, . and ..
  - Get the link counts right
  - Make modified pages dirty
- s5fs_rmdir() - Unlink an empty directory
  - Remove the directory entry linking to the v-node
  - Reduce the link count
  - Dirty the file blocks (and maybe the i-node)

What to implement: v-nodes

- s5fs_readdir() - read a directory entry
  - Read an s5fs directory entry
  - Translate to generic format
- s5fs_stat() - file information
  - Read an i-node info
  - Translate to generic format

What to implement: helpers

- All these in s5fs_subr.c
- s5_seek_to_block() - find and allocate data blocks
  - Translate offset to block containing it
  - If requesting allocation
    - get a data block
    - Connect to i-node
    - Dirty both
  - Used by fillpage, cleanpage, dirtypage

What to implement: helpers

- s5_write_file(), s5_read_file()
  - Implementations of read and write earlier
  - Useful as subroutines in all directory implementations
- s5_alloc_block()
  - Get a new data block
  - Manipulates free list (more in a sec)
  - Dirty new block and superblock

What to implement: helpers

- s5_find_dirent()
  - Read directory and match name
  - Return i-node number
- s5_remove_dirent()
  - Find entry to remove
  - In not last, write last over it
    - Use s5_read_file and s5_write_file to dirty blocks
  - Adjust size

What to implement: helpers

- s5_link()
  - Add a new directory entry
    - Dirty the files – use read/write
  - Used by all the creation calls

Already implemented: helpers

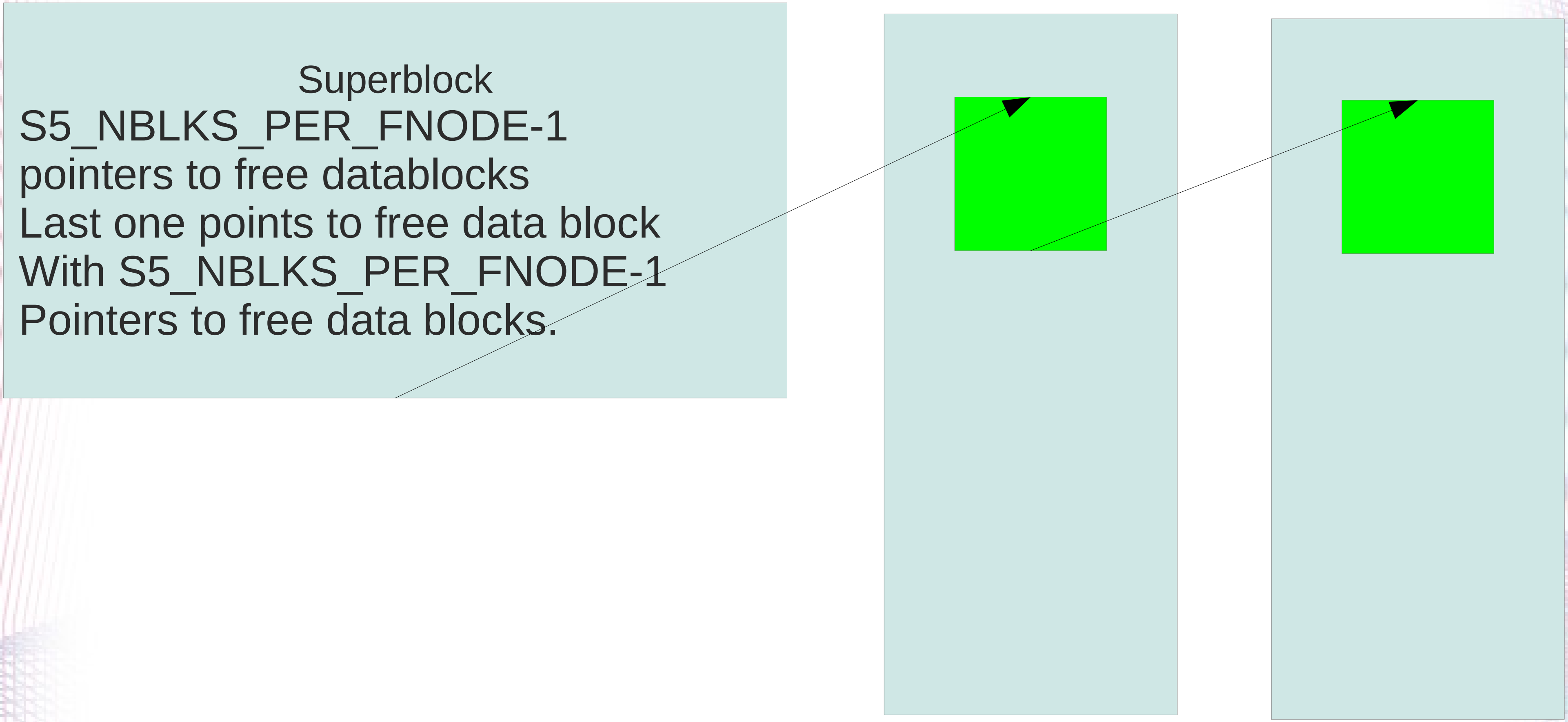
- s5_free_block() - free a data block
- s5_alloc_inode() - allocate an i-node
- s5_free_inode() - free an i-node and all blocks

- Useful to have – useful to read!

Data Structures: Indirect Blocks

- S5_NDIRECT direct blocks
- After that 1 indrect block
  - A data block that holds S5_NIDIRECT pointers
- Take into account in
  - s5_read_file, s5_write_file, s5_seek_to_block

Data Structures: Free List



Superblock

S5_NBLKS_PER_FNODE-1
pointers to free datablocks
Last one points to free data block
With S5_NBLKS_PER_FNODE-1
Pointers to free data blocks.

Data Structures: Free List

- When all pointers in superblock are used
  - Copy the pointers from the “next” free block into superblock
  - Allocate the “next” free block
- s5_free_block() does the reverse and is implemented....

Data Structures: Sparse Files

- Try this
  - Create File
  - Seek out 50KB
  - Write a block
- A sparse file will have one block allocated and connected to the i-node
- Make sparse files
  - Think about how s5fs_dirtypage interacts...