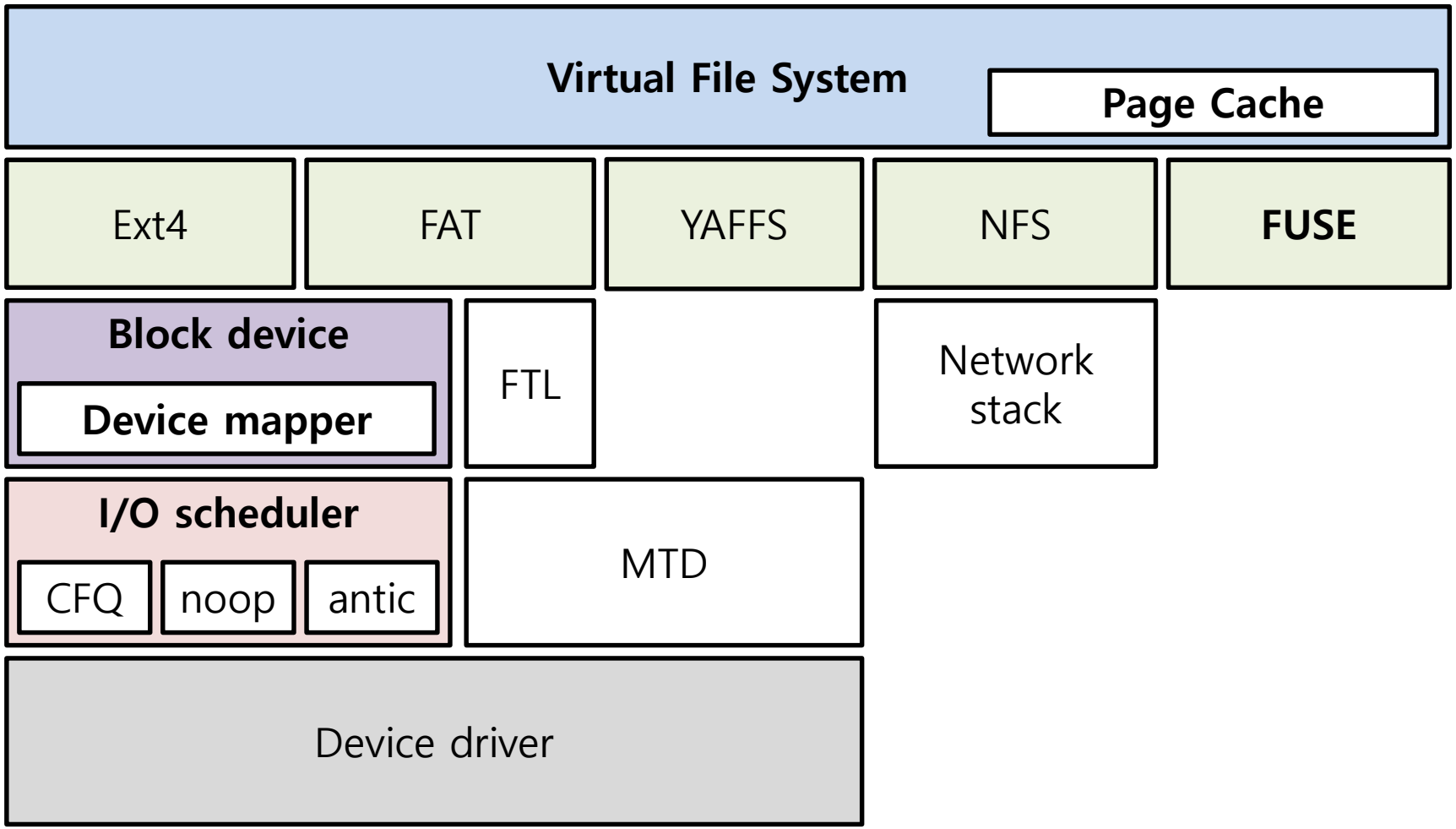
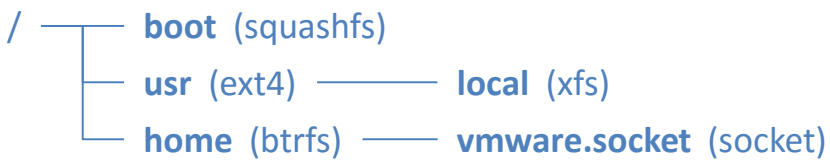
Linux virtual file system

Storage implementation layers



Introduction to VFS

- Hordes **different** file system implementations
  - IPC mechanisms (PIPE, FIFO, socket, …) too



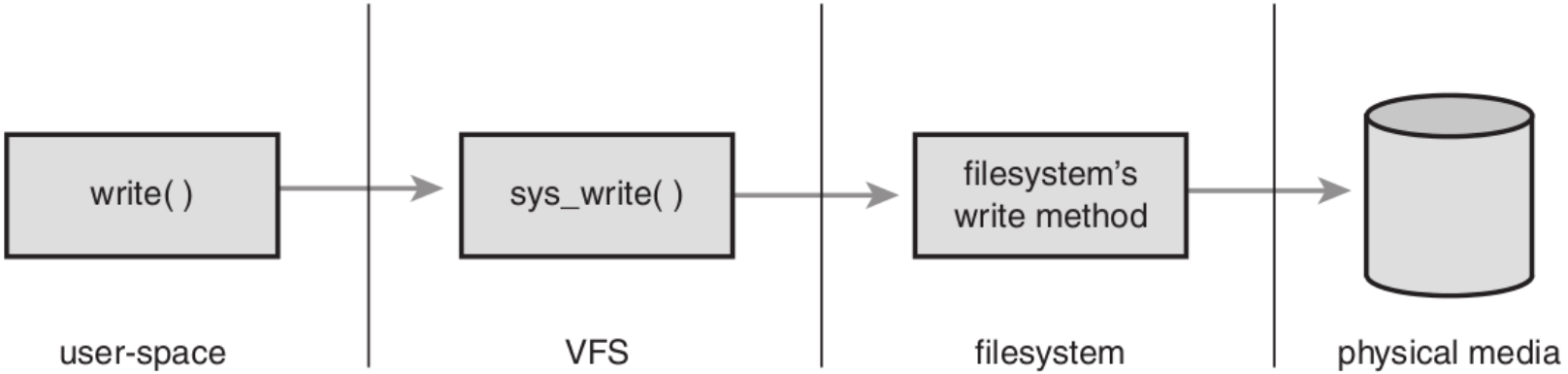
- Abstracts **generic** file system implementations
  - Directory traversal
  - Page cache
- Interfaces with **POSIX** system call APIs
  - File descriptor management

File system APIs

- **File operations**
  - open, creat, read, write, lseek, close, dup
  - pread, pwrite, readv, writev, preadv, pwritev
- **Directory operations**
  - chdir, mkdir, rmdir
  - link, unlink, rename, chmod, chown, stat
  - opendir, readdir, readdir_r, closedir, scandir
- **Special commands**
  - fcntl, ioctl, flock

VFS implementation

- System call to file system's specific methods



- **Generic objects**
  - *Superblock*: specific file system
  - *Inode*: a file meta-data
  - *Dentry*: a directory entry
  - *File*: an open file

VFS operations

- File system specific operations
  - Pseudo object oriented programming model
    - File system specific object: ex. `sb->s_fs_info`
    - File system specific operations: ex. `sb->s_op.sync_fs()`
  - Generic object + FS specific object + operations = VFS
- VFS internal objects
  - To handle file system status
    - struct `file_system_type` file system mounting
    - struct `vfsmount` file system mount point
    - struct `file_struct` file descriptor management
      - `struct file *fd_array[NR_OPEN_DEFAULT]`
    - struct `fs_struct` process status (working dir, …)
  - Dentry cache

VFS operations example

- sys_open (fs/open.c)
  - Main routine: do_sys_open()
  - Allocate fd: get_unused_fd_flags()
  - Walk path and open a file: do_filp_open()
    - lookup_fast(), __d_lookup() : *dentry cache lookup*
    - i_op->lookup(dir, dentry) : *repeat to target inode*
    - d_op->d_hash(dentry, name),
      d_op->d_compare(dentry, name1, name2)
    - f_op->open(inode, file)
    - i_op->create(dir, dentry, mode)
    - s_op->alloc_inode(sb)

Superblock API

- Superblock **object**
  - Per mounted file system instance
- Superblock **operations**
  - **Superblock** management
    - write_super(), put_super()
  - **Inode** management
    - alloc_inode(), write_inode()
  - **File system** management
    - sync_fs(), free_fs()
- **Initialization:** get_sb() function

Type	Name
list_head	s_list
dev_t	s_dev
list_head	s_inodes
list_head	s_files
super_operations	**s_op**
void *	**s_fs_info**
...	...

Inode API

- Inode **object**

- Inode **operations**
  - **Inode** management
    - create, truncate, setattr, fallocate,
  - **Directory** management
    - lookup, link, unlink, symlink, mkdir, …

Type	Name
super_block	i_sb
list_head	**i_dentry**
unsigned long	**i_ino**
atomic_t	i_count
uid_t	i_uid
struct timespec	i_atime
loff_t	**i_size**
address_space	i_mapping
inode_operations	i_op
file_operations	i_fop
void *	i_private

File API

- File object

- File operations
  - llseek(), read(), write(), mmap()
  - open(), release(), flush()

Type	Name
struct path	f_path
int	f_flags
loff_t	**f_pos**
address_space	f_mapping
file_operations	**f_op**
void *	private_data

Directory API

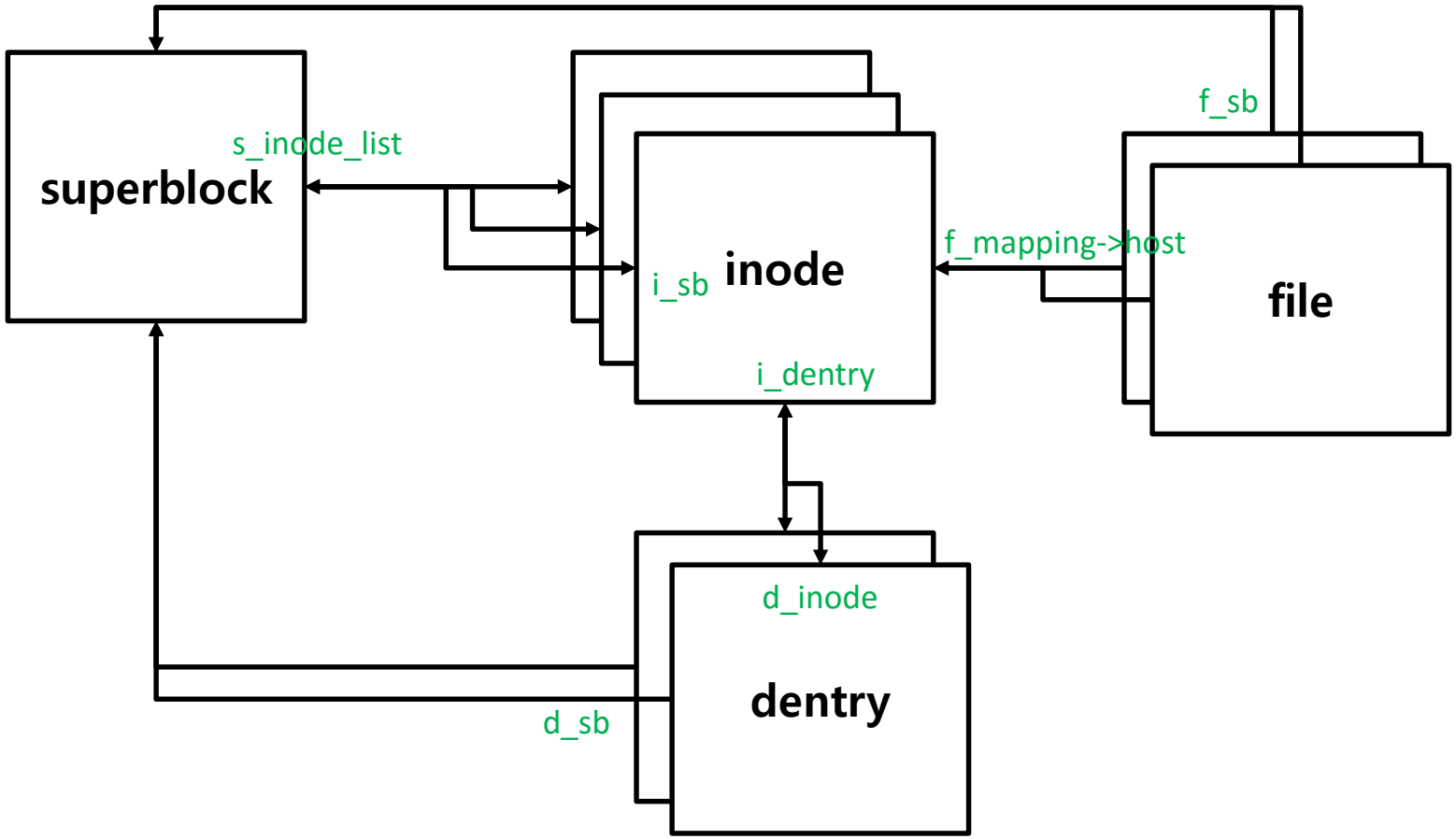
- Dentry **object**

- **Operations**
  - d_revalidate
  - d_hash
  - d_compare
  - d_delete, d_release
  - d_iput, d_dname

- Almost no need to implement
  - Exception: ***case insensitive*** file name

Type	Name
struct inode	d_inode
hlist_node	d_hash
struct dentry	d_parent
struct qstr	**d_name**
list_head	d_subdirs
list_head	d_alias
dentry_operations	d_op
void*	d_fsdata
char	**d_iname[]**

VFS objects relationships



Page cache implementation

- Integrated with process virtual memory module
- **Page** associated with inode
  - page->mapping, index
- **address_space** object
  - Target file
  - Page cache tree (radix)
- address_space **operations**
  - **I/O**: writepage, readpage, writepages, readpages
  - **Block allocation and mapping**: bmap()

Type	Name
struct inode	host
radix_tree_root	page_tree
long	nrpages
address_space_ operations	a_ops
…	…

⇨ Delayed allocation

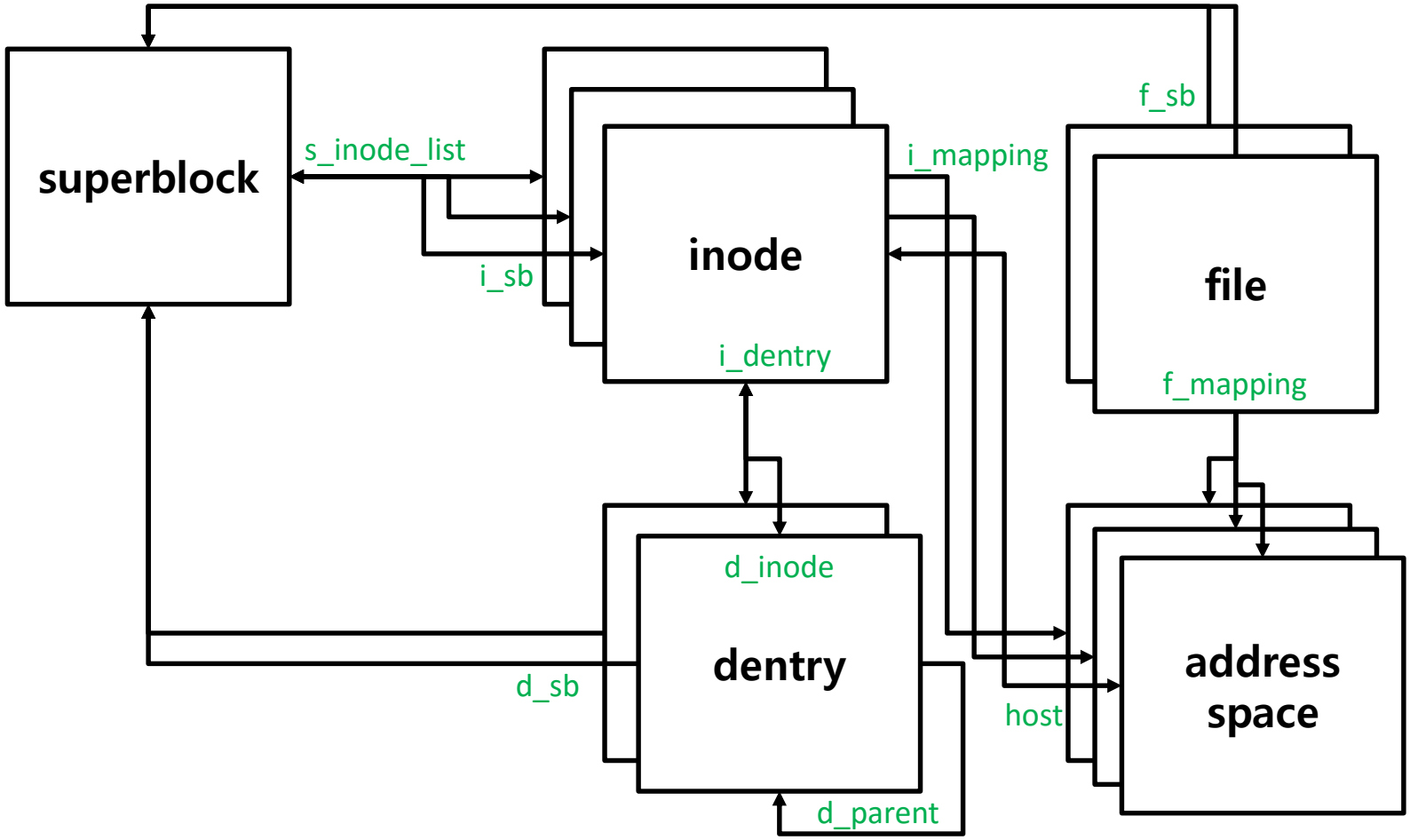
Page cache implementation

- **Useful API**
  - page = find_get_page(mapping, index)
  - SetPageDirty(page), ClearPageDirty(page)

- **Flush thread**
  - Write-back dirty pages
    - On-demand, dirty threshold, free threshold
  - Per allocation-group
    - Per disk if device mapper is not used
    - Per top-level virtual device if device mapper is applied

Implementing file system

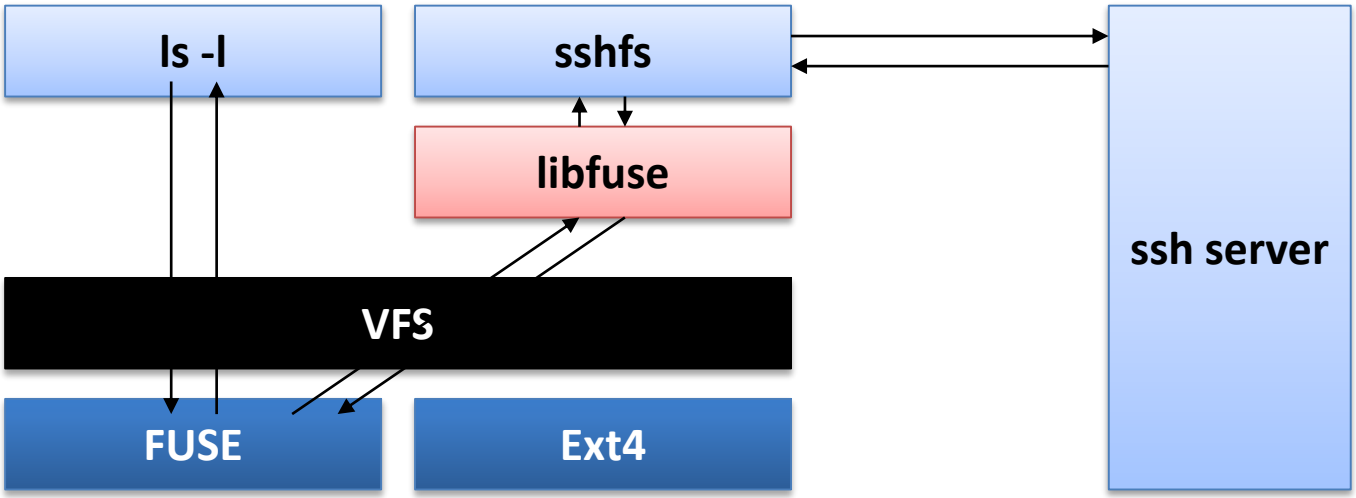
- superblock, inode, file, dentry, address space



FUSE: File system in user space

Introduction

- Implement simple file system in userspace



- **Compatiblity**
  - Linux, FreeBSD, NetBSD, Mac OS X, OpenSolaris, …
- **Example file systems**
  - Gnome VFS2, unionfs, ftpfs, sshfs, Android sdcard, …

How to use

- **Installation**
  - apt-get install fuse fuse-utils libfuse2 libfuse-dev
  - Source: http://fuse.sourceforge.net/

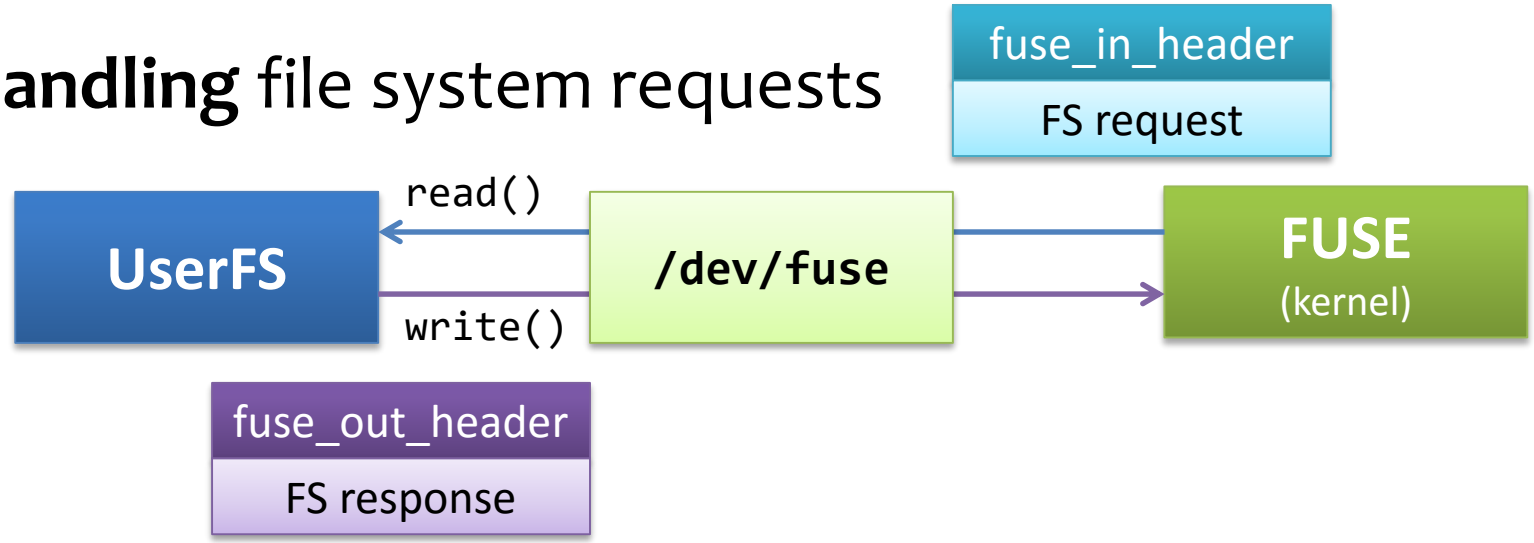
- **Mount / unmount**
  - ***sshfs*** user@host /tmp/fuse
  - ***fusermount*** –u /tmp/fuse

Using SSHFS

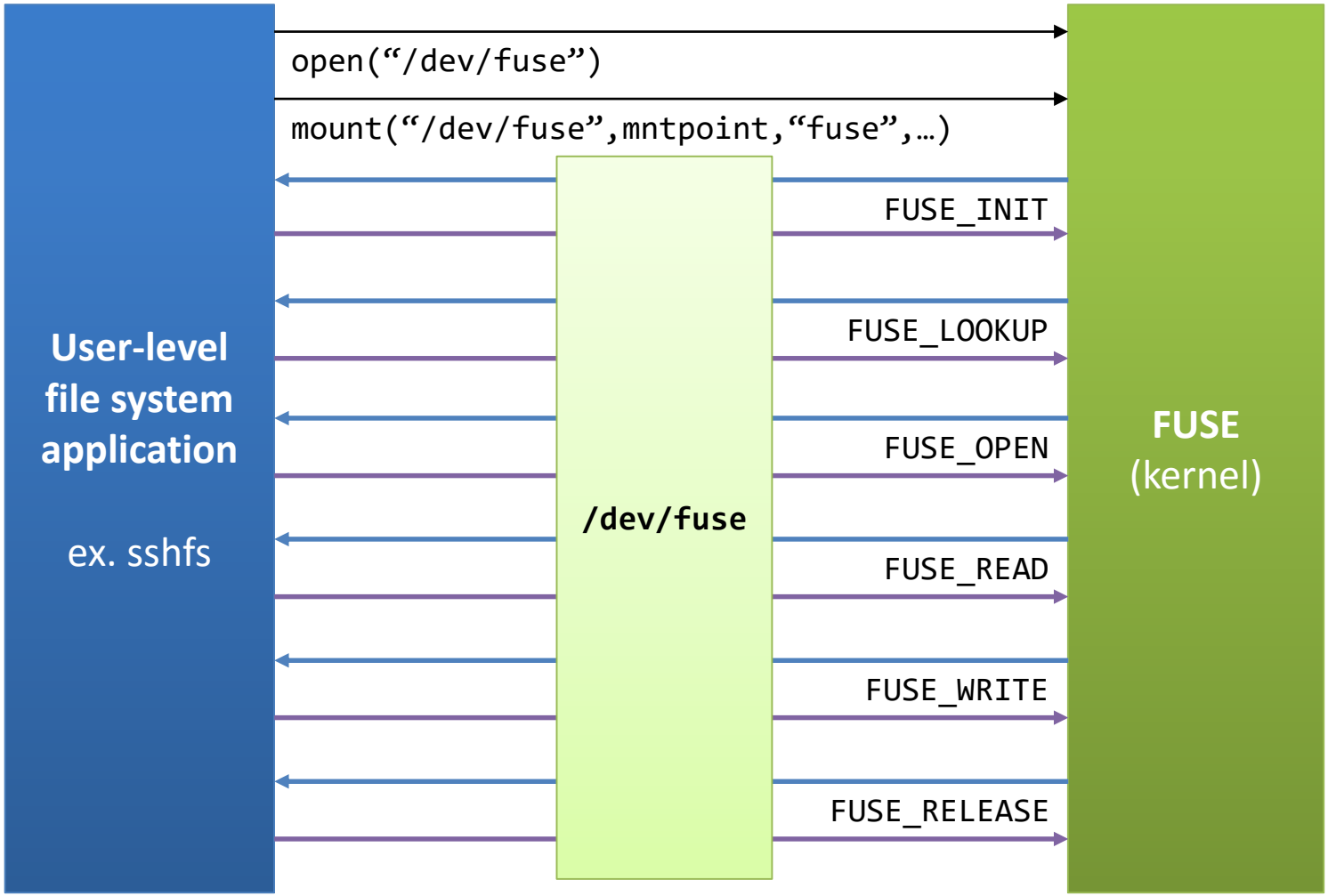
- **Setup ssh server**
  - apt-get install ssh
- **Install** sshfs
  - apt-get install sshfs
- **Mount** sshfs
  - mkdir /home/user/fusemnt
  - sshfs user@localhost:/home/user /home/user/fusemnt
- **Unmount** sshfs
  - fusermount –u /home/user/fusemnt

Low-level FUSE interface

- **FUSE communication channel**
  - */dev/fuse* `fd = open("/dev/fuse", O_RDWR)`
- **Mounting** a file system
  - mount("/dev/fuse", mount_point, "fuse", flags, *opts*)
  - opts: "fd=n,…"
- **Handling** file system requests



FUSE communication example



FUSE message headers

- struct ***fuse_in_header***
  - Describes a file system request
  - enum ***fuse_opcode*** opcode:
    - init
    - open, read, write
    - mknod, mkdir, readlink, symlink, unlink
    - rmdir, rename
    - lookup, forget, getattr, setattr
    - statfs, release, flush
    - opendir, readdir, releasedir
- struct ***fuse_out_header***
  - Return execution result

fuse_in_header	
u32	len
u32	opcode
u64	unique
u64	nodeid
u32	uid
u32	gid
u32	pid
u32	padding

fuse_out_header	
u32	len
s32	error
u64	unique

Directory access

- **FUSE_LOOKUP**
  - Find file "***data***" in directory ***fuse->nodeid***
  - Returns `struct` **`fuse_entry_out`**
- struct ***fuse_open_in***
  - u32 flags
  - Open file identified by `fuse->nodeid`
  - User defines file handle `fh`
- struct ***fuse_release_in***
  - Close file `fh`
  - No return

fuse_entry_out	
u64	nodeid
u64	generation
u64	entry_valid
u64	attr_valid
	fuse_attr

fuse_open_out	
u64	fh
u32	open_flags

fuse_release_in	
u64	fh
u32	flags
u32	release_flags
u64	lock_owner

Read and write

- struct ***fuse_read_in***
  - Next to fuse_in_header
  - Read file (offset, size)
  - Send result to */dev/fuse*
- struct ***fuse_write_in***
  - Identical to *fuse_read_in*
  - File data comes next to *fuse_write_in*
  - Return *fuse_write_out* to */dev/fuse*

fuse_read_in	
u64	fh
u64	offset
u32	size
u32	read_flags
u64	lock_owner
u32	flags
u32	padding

fuse_write_out	
u32	size
u32	padding

Other primitives

- Refer to ***include/fuse_kernel.h***
  - struct fuse_(opcode)_in/out

- **FUSE for Python**
  - http://sourceforge.net/apps/mediawiki/fuse/index.php?title=SimpleFilesystemHowto

- **Other language bindings**
  - C++, Java, C#, Go, Haskell, TCL, Perl, Ocaml, Ruby, Lua, …
  - http://sourceforge.net/apps/mediawiki/fuse/index.php?title=LanguageBindings

libfuse: C binding

- **High-level API**
  - Implements common fuse ⇔ kernel interactions
    - Mount command-line parsing
    - Interpret fuse_in_header and build fuse_out_header
  - Simplifies file system implementation

- **Developing with libfuse**
  - **Include:** <fuse.h>
  - **Compile:** `gcc -Wall `pkg-config fuse --cflags --libs``
  - **Example:** `/usr/share/doc/libfuse-dev/examples/hello.c`
  - **Document:** http://fuse.sourceforge.net/doxygen/index.html

Implementing FS with libfuse

- `struct` **`fuse_operations`**
  - Define and override file system operations
  - Example
    - **`.open`**`(char *path, struct fuse_file_info *fi)`
    - **`.read`**`(char *path,`
      `char *buf, size_t size, off_t offset,`
      `struct fuse_file_info *fi)`
- **`fuse_main`**`(argc, argv, &op, user_data)`
  - Mounts the file system
  - Fetches file system requests and calls defined functions
  - Returns when unmounted: `fusermount -u`

Project 3
FUSE file system

Overview

- **Project goal**
  - Implement a simple FUSE file system using libfuse
- **Due**: 17th June, 23:59
- **Baseline**
  - Implement root directory (no subdirectory)
  - **File read/write**: run IOzone to prove it working
  - **File allocation**: sequential (no fragmented) block allocation
- **You need to implement**
  - getattr, readdir, open, read, write

File system layout

- **Use a 1 GB file as a storage media**

```
$ truncate -s 1G disk.img
```

- **Layout**
  - 4 KB blocks
  - Block bitmap size: 32 KB (8 blocks)
  - Inode bitmap size: 1 block ← you define # of inodes
  - Array-based directory (max length: 100 bytes)