Python FUSE

File-System in **USE**rspace

Beyond the Traditional File-Systems

http://mbertozzi.develer.com/python-fuse

Matteo Bertozzi (Th30z)
http://th30z.netsons.org

Talk Overview

- What is a File-System
- Brief File-Systems History
- What is FUSE
- Beyond the Traditional File-System
- API Overview
- Examples (Finally some code!!!)
  http://mbertozzi.develer.com/python-fuse
- Q&A

What is a File-System

Is a Method of storing and organizing data to make it easy to find and access.

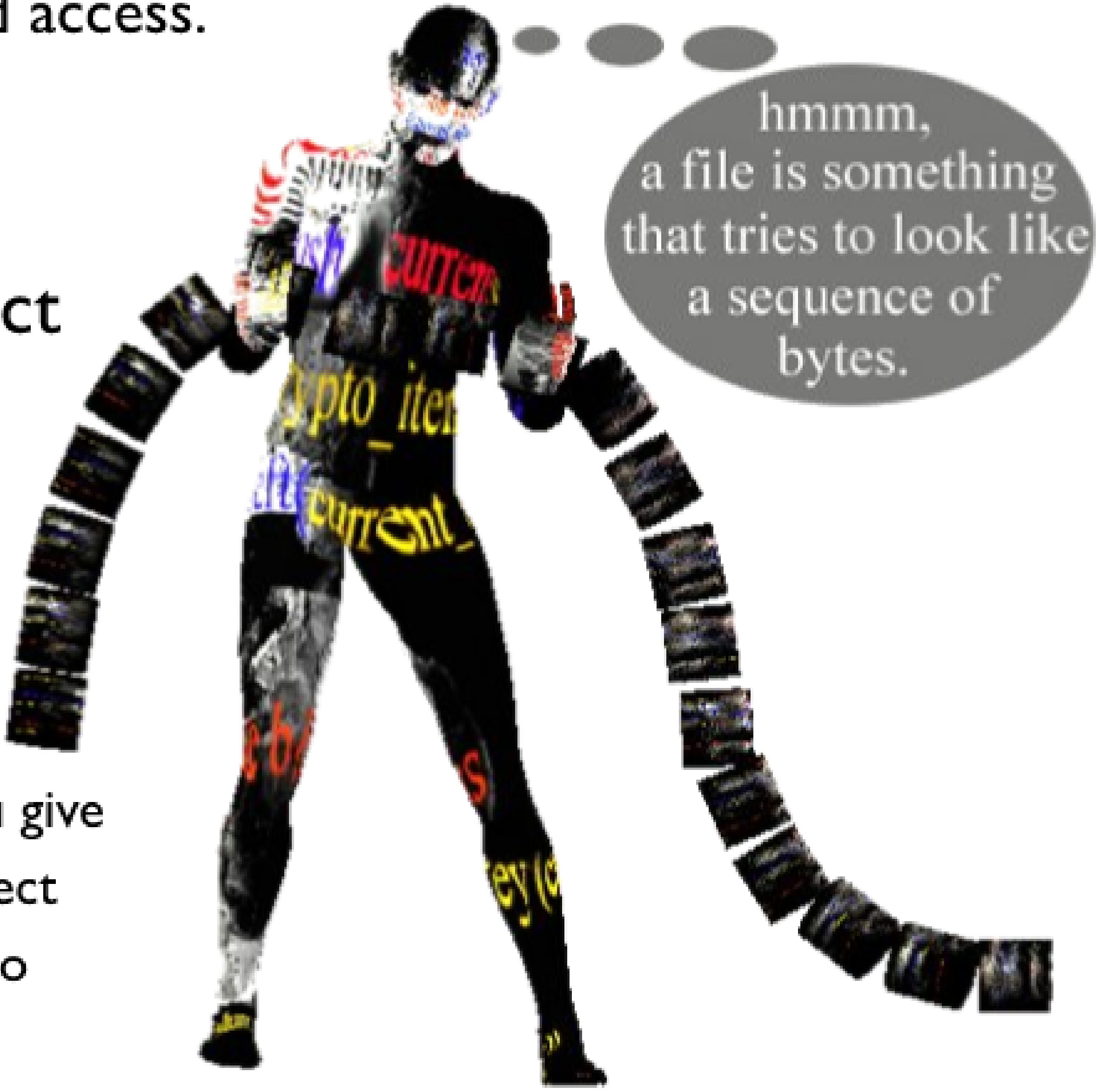
...to interact with an object

You name it, and you say what you want it do.

The Filesystem takes the name you give

Looks through disk to find the object

Gives the object your request to do something.



What is a File-System

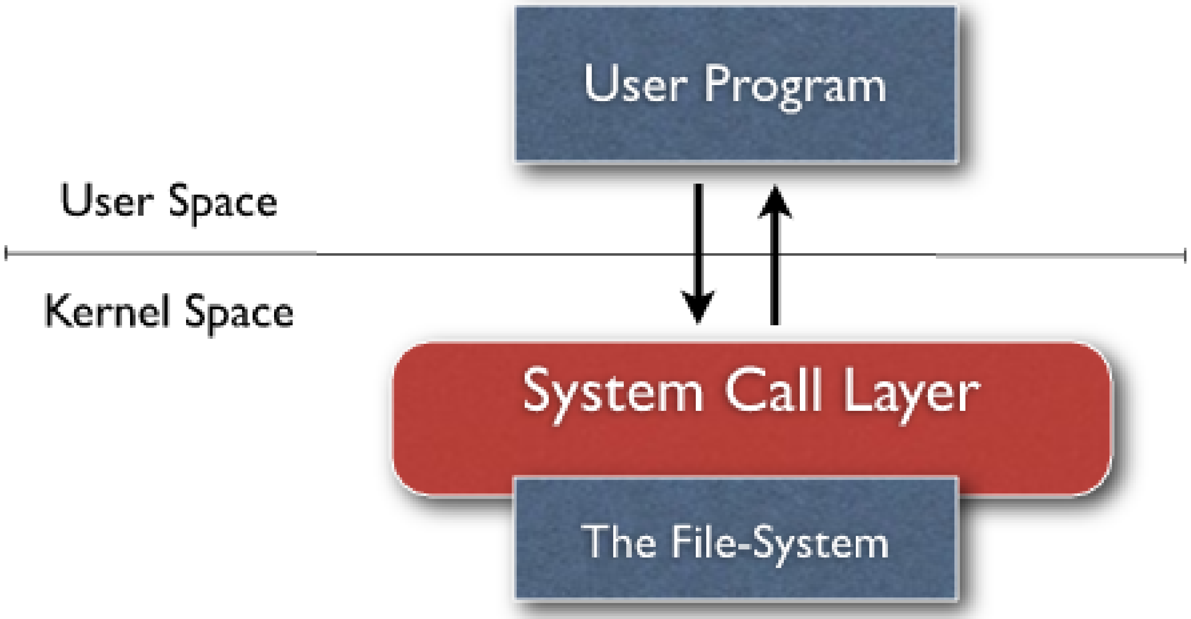
- On Disk Format (...serialized struct)
  ext2, ext3, reiserfs, btrfs...
- Namespace
  (Mapping between name and content)
  /home/th30z/, /usr/local/share/test.c, ...
- Runtime Service: open(), read(), write(), ...

...A bit of History (The Origins)

Multics 1965 (File-System Paper)

A General-Purpose File System For Secondary Storage

Unix Late 1969



Only One File-System

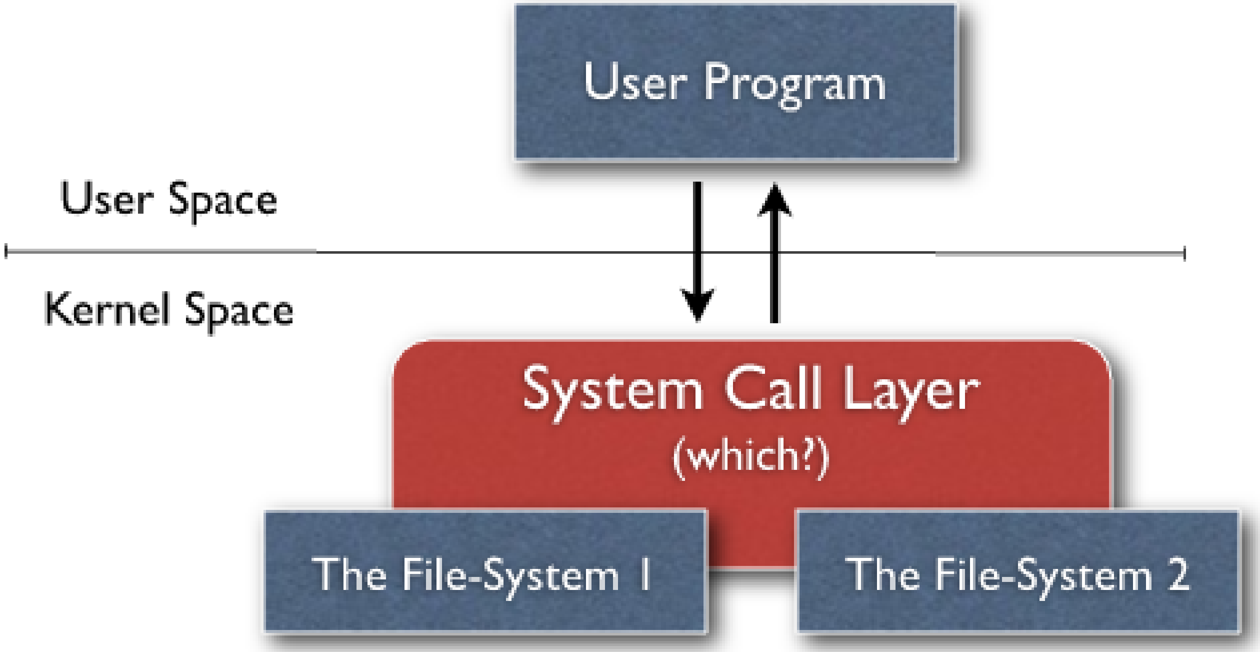
(The Evolution)

...A bit of History

Multics 1965 (File-System Paper)

A General-Purpose File System For Secondary Storage

Unix Late 1969



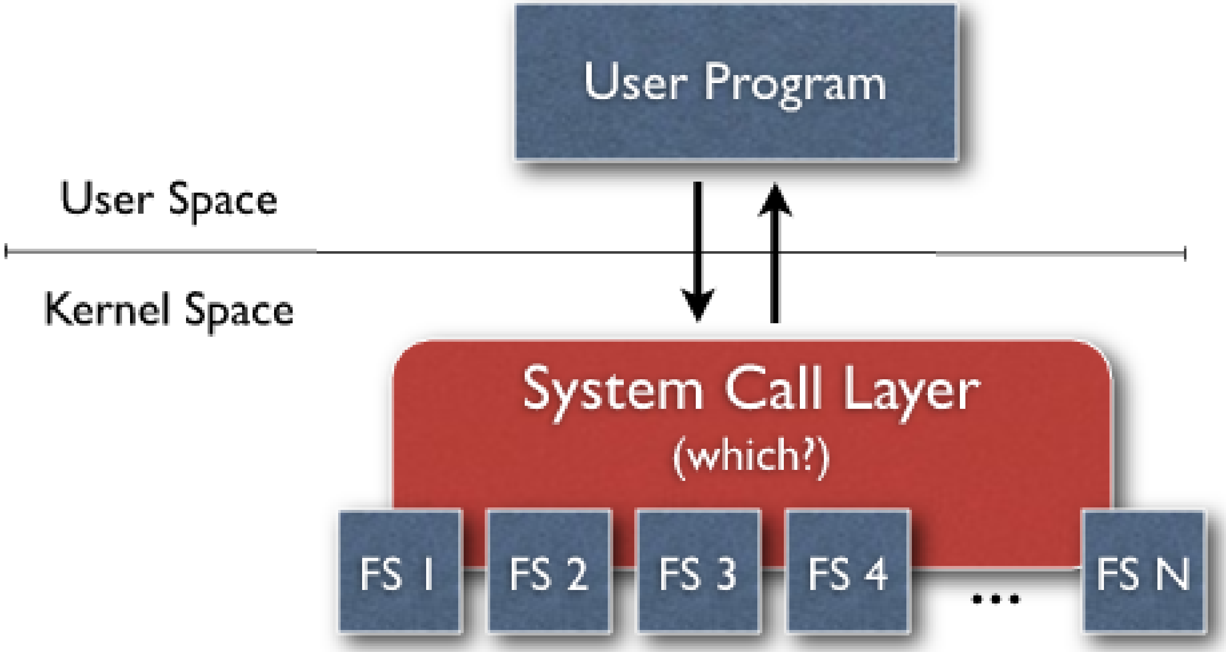
(The Evolution)

...A bit of History

Multics 1965 (File-System Paper)

A General-Purpose File System For Secondary Storage

Unix Late 1969



(The Solution)

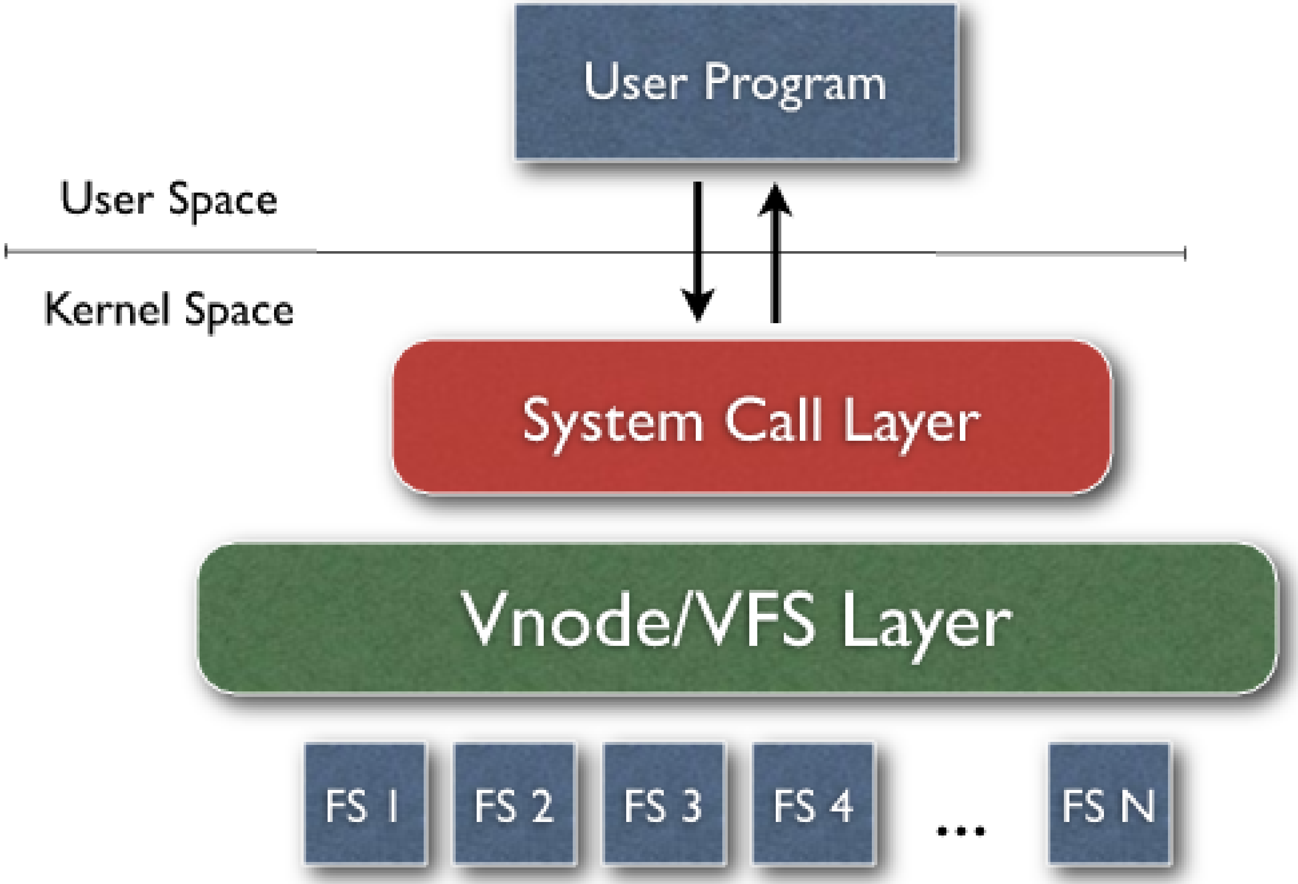
...A bit of History

Multics 1965 (File-System Paper)

A General-Purpose File System For Secondary Storage

Unix Late 1969

Sun Microsystem 1984



Virtual File-System

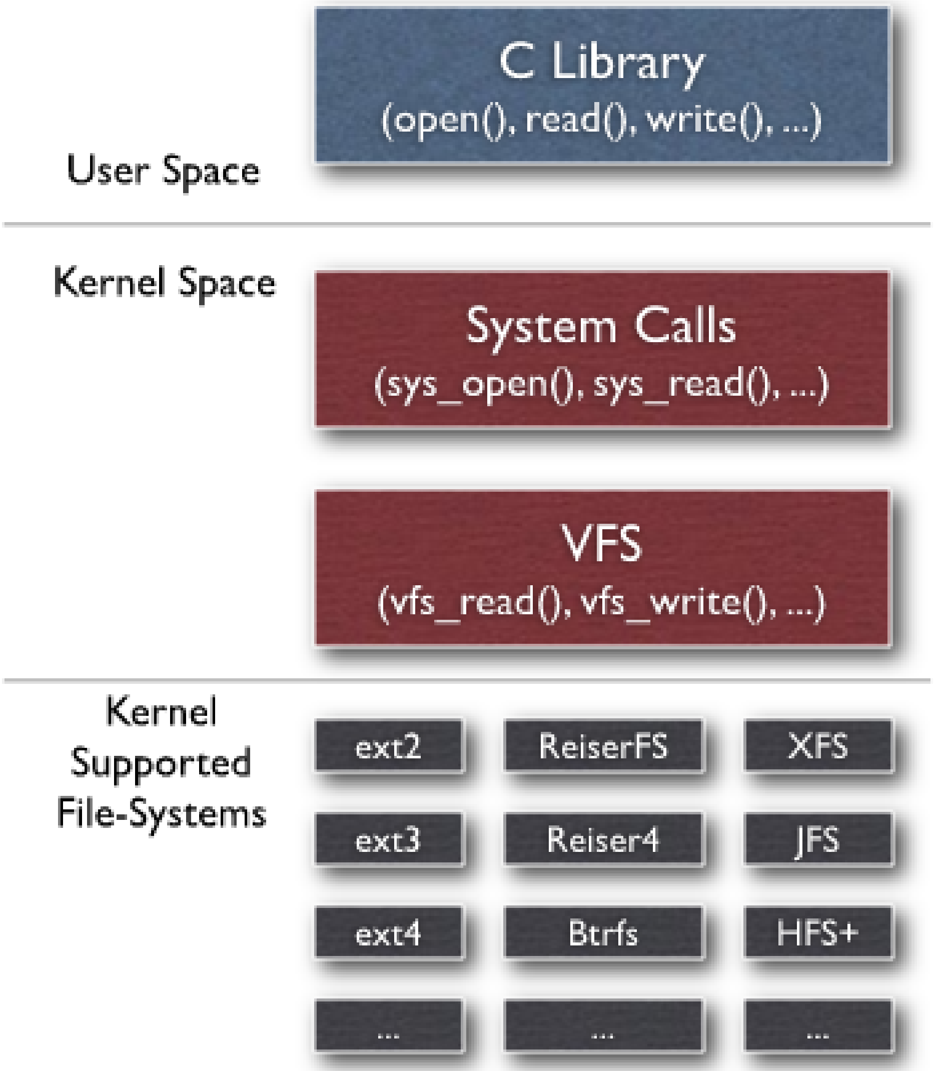
- Provides an abstraction within the kernel which allows different filesystem implementations to coexist.
- Provides the filesystem interface to userspace programs.

VFS Concepts

A super-block object represents a filesystem.

I-Nodes are filesystem objects such as regular files, directories, FIFOs, ...

A file object represents a file opened by a process.



User Space

C Library
(open(), read(), write(), ...)

Kernel Space

System Calls
(sys_open(), sys_read(), ...)

VFS
(vfs_read(), vfs_write(), ...)

Kernel Supported File-Systems

ext2 ReiserFS XFS

ext3 Reiser4 JFS

ext4 Btrfs HFS+

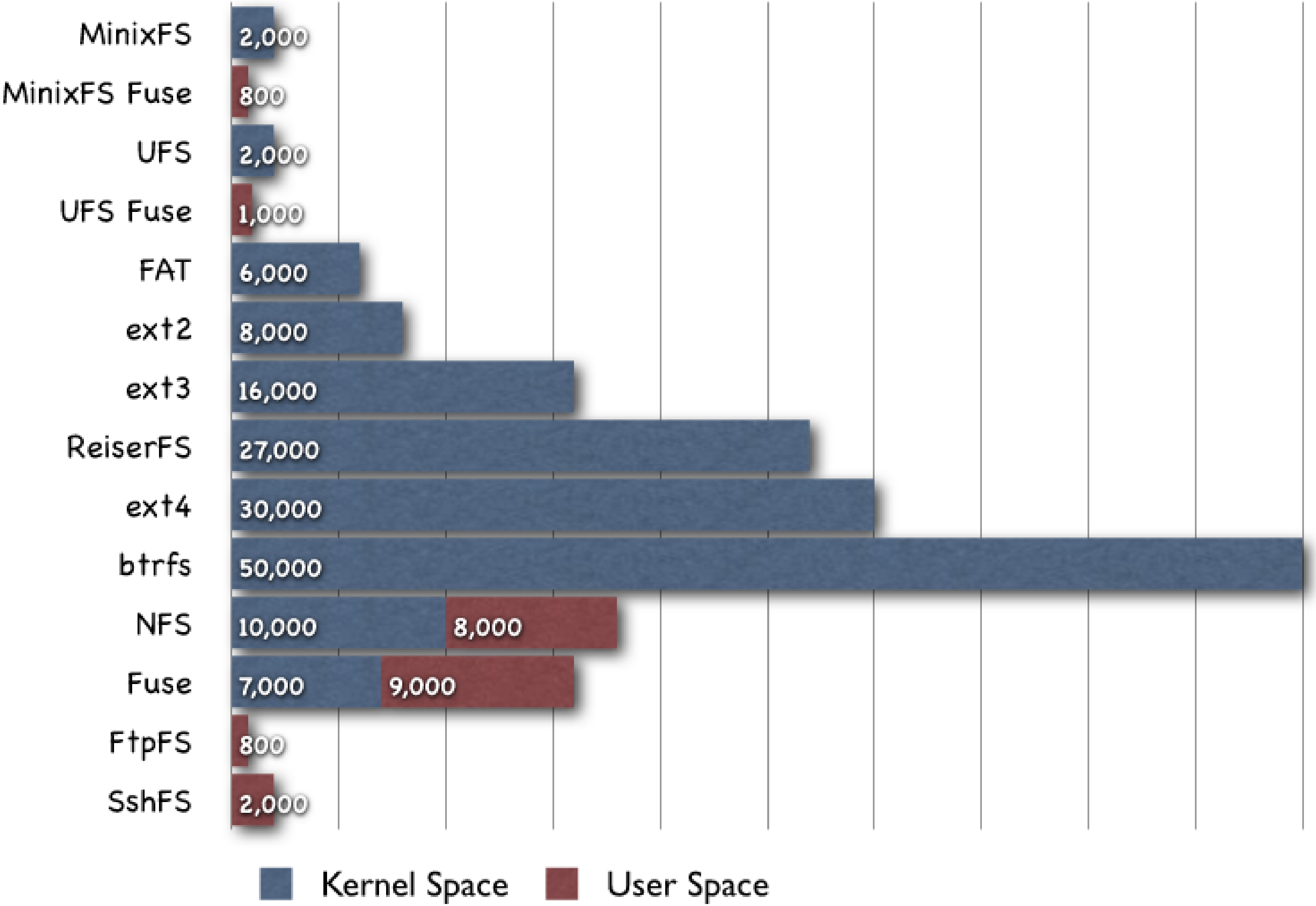
... ...

Wow, It seems not much difficult writing a filesystem

Why File-System are Complex

- You need to know the Kernel (No helper libraries: Qt, Glib, ...)
- Reduce Disk Seeks / SSD Block limited write cycles
- Be consistent, Power Down, Unfinished Write... (Journal, Soft-Updates, Copy-on-Write, ...)
- Bad Blocks, Disk Error
- Don't waste to much space for Metadata
- Extra Features: Deduplication, Compression, Cryptography, Snapshots…

File-Systems Lines of Code



Building a File-System is Difficult

- Writing good code is not easy (Bugs, Typo, ...)
- Writing good code in Kernel Space
  Is much more difficult!
- Too many reboots during the development
- Too many Kernel Panic during Reboot
- We need more flexibility and Speedups!

FUSE, develop your file-system
with your favorite language and library
in user space

What is FUSE

- Kernel module! (like ext2, ReiserFS, XFS, ...)
- Allows non-privileged user to create their own file-system without editing the kernel code. (User Space)
- FUSE is particularly useful for writing "virtual file systems", that act as a view or translation of an existing file-system storage device. (Facilitate Disk-Based, Network-Based and Pseudo File-System)
- Bindings: Python, Objective-C, Ruby, Java, C#, ...

File-Systems in User Space?

...Make File Systems Development Super Easy

- All UserSpace Libraries are Available
- ...Debugging Tools
- No Kernel Recompilation
- No Machine Reboot!
  ...File-System upgrade/fix
  2 sec downtime, app restart!

Yeah, ...but what’s FUSE?

It’s a File-System with user-space callbacks

ntfs-3g

sshfs

ifuse

ChrionFS

zfs-fuse

YouTubeFS

gnome-vfs2

ftpfs

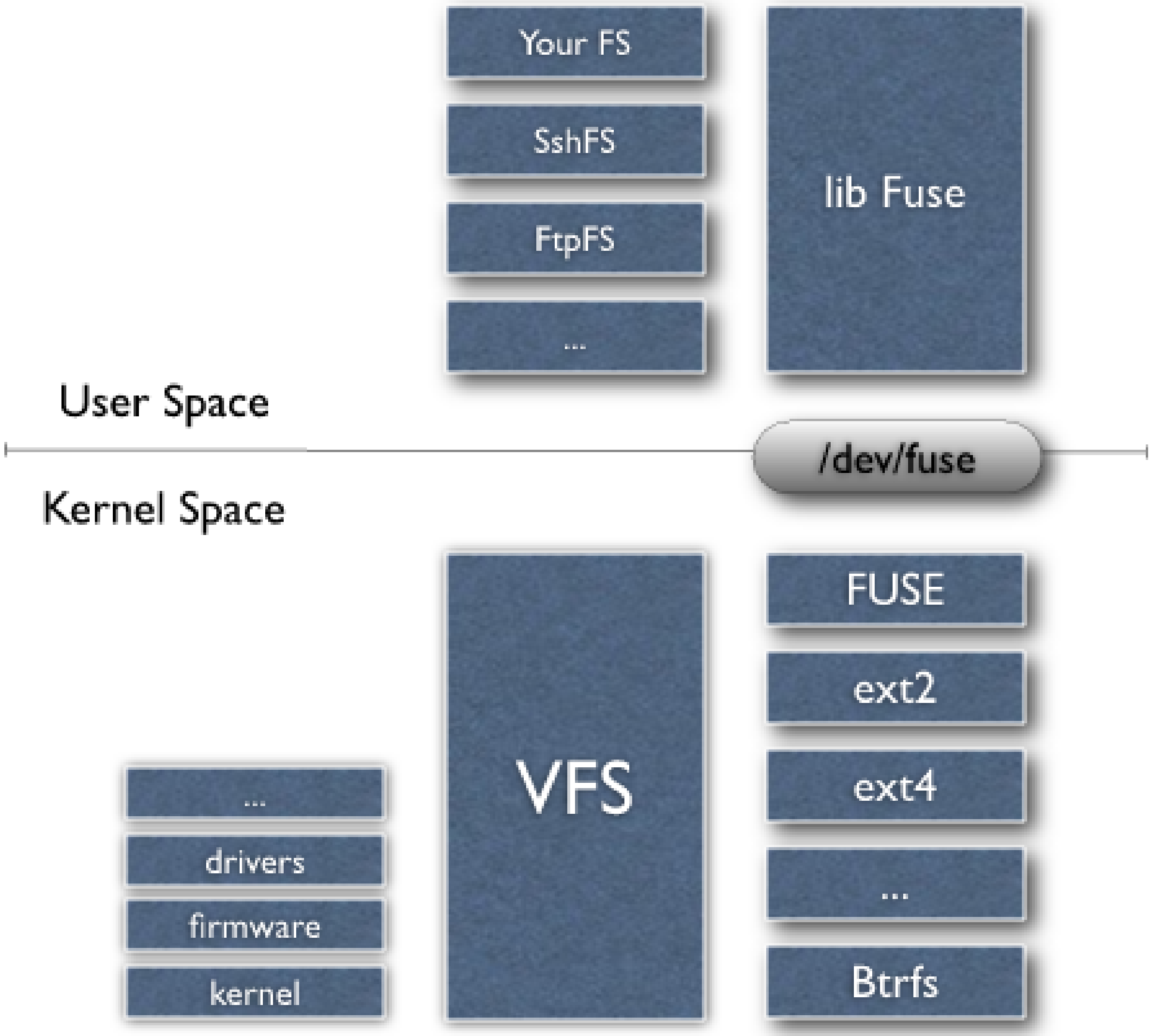
cryptoFS

RaleighFS

Unix

FUSE Kernel Space and User Space

The FUSE kernel module and the FUSE library communicate via a special file descriptor which is obtained by opening /dev/fuse



Your FS

SshFS

FtpFS

...

lib Fuse

User Space

/dev/fuse

Kernel Space

...

drivers

firmware

kernel

VFS

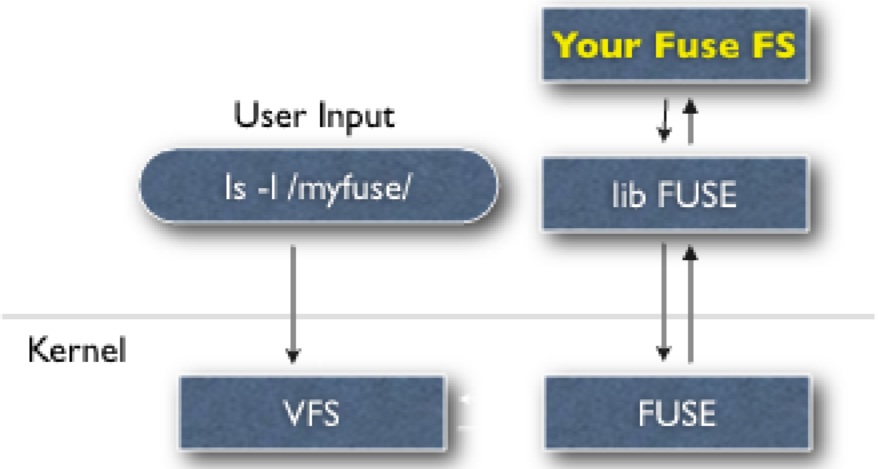
FUSE

ext2

ext4

...

Btrfs



Your Fuse FS

User Input

ls -l /myfuse/

lib FUSE

Kernel

VFS

FUSE

...be creative

Beyond the Traditional File-Systems

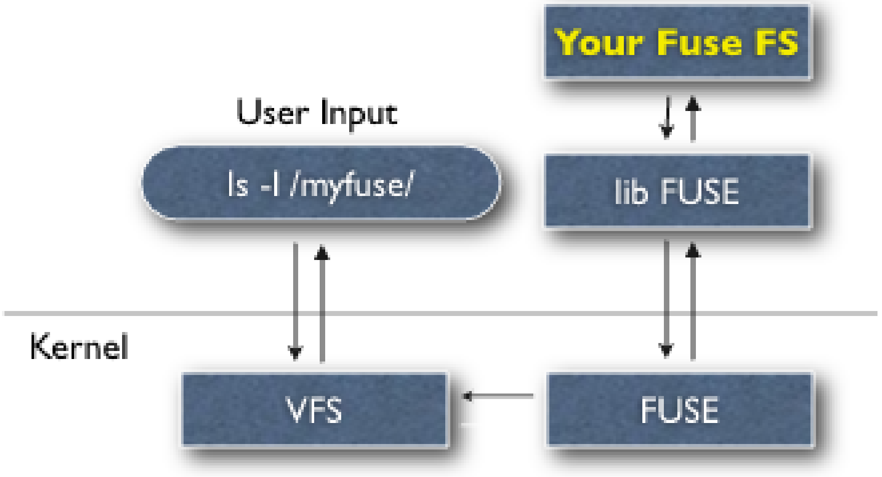
- ImapFS: Access to your mail with grep.
- SocialFS: Log all your social network to collect news/jokes and other social things.
- YouTubeFS: Watch YouTube video as on your disk.
- GMailFS: Use your mailbox as backup disk.

Thousand of tools available cat/grep/sed

open() is the most used function in our applications

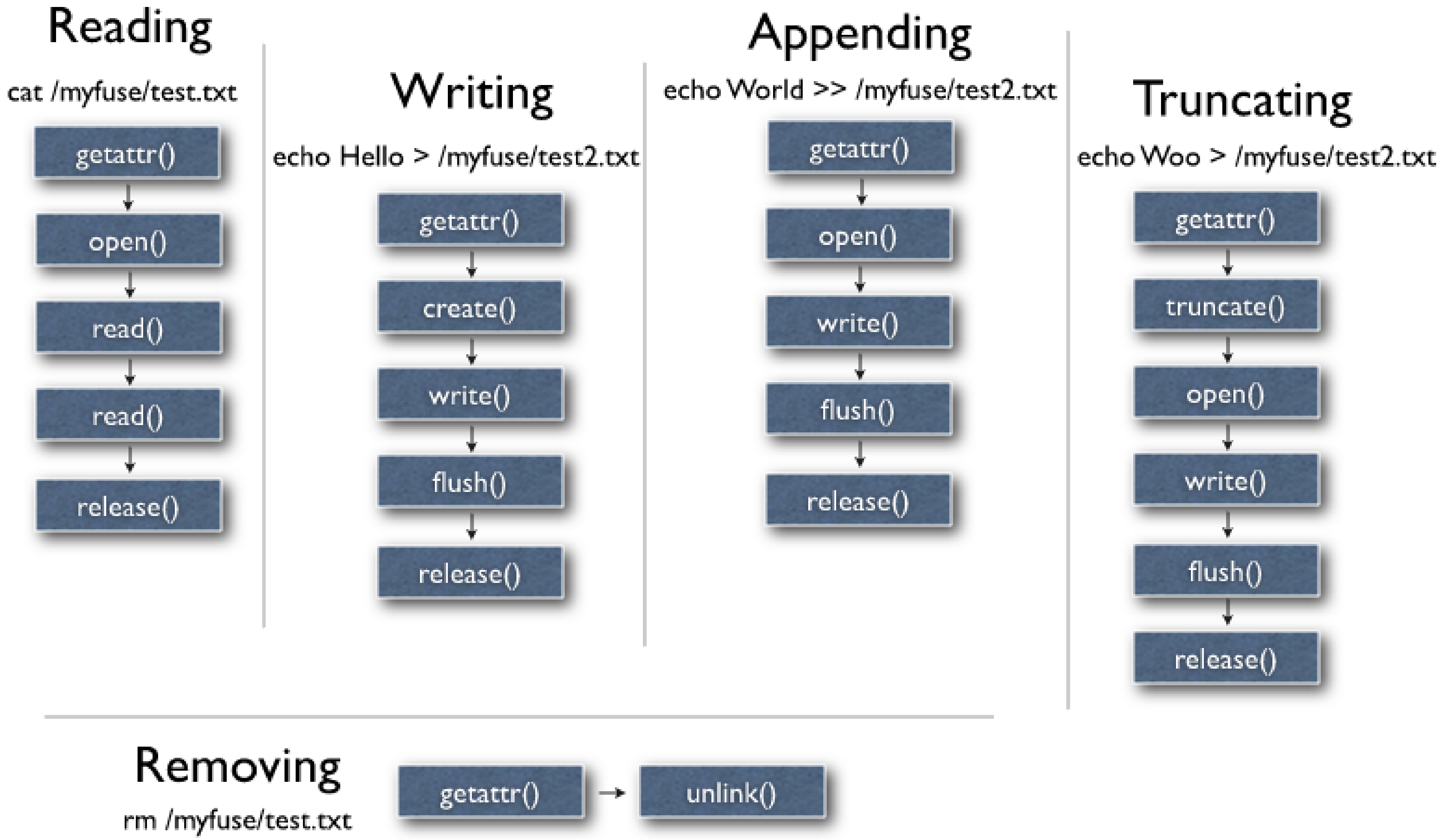
FUSE API Overview

- create(path, mode)
- truncate(path, size)
- mknod(path, mode, dev)
- open(path, mode)
- write(path, data, offset)
- read(path, length, offset)
- release(path)
- fsync(path)
- chmod(path, mode)
- chown(path, oid, gid)
- mkdir(path, mode)
- unlink(path)
- readdir(path)
- rmdir(path)
- rename(opath, npath)
- link(srcpath, dstpath)



FUSE API Overview

(File Operations)



Reading

cat /myfuse/test.txt

getattr()
open()
read()
read()
release()

Writing

echo Hello > /myfuse/test2.txt

getattr()
create()
write()
flush()
release()

Appending

echo World >> /myfuse/test2.txt

getattr()
open()
write()
flush()
release()

Truncating

echo Woo > /myfuse/test2.txt

getattr()
truncate()
open()
write()
flush()
release()

Removing

rm /myfuse/test.txt

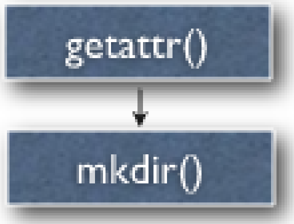
getattr()
unlink()

(Directory Operations)

FUSE API Overview

Creating

mkdir /myfuse/folder

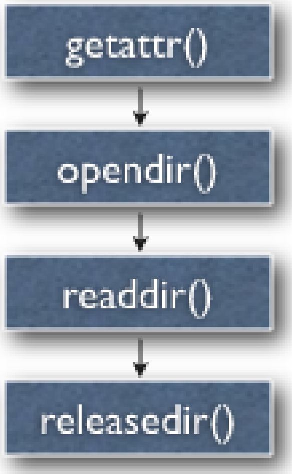


getattr()

mkdir()

Reading

ls /myfuse/folder/



getattr()

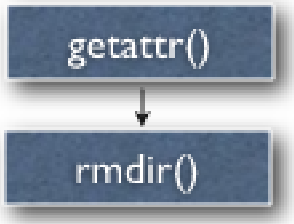
opendir()

readdir()

releasedir()

Removing

rmdir /myfuse/folder



getattr()

rmdir()

Other Methods (getattr() is always called)

chown th30z:develer /myfuse/test.txt	getattr() -> chown()
chmod 755 /myfuse/test.txt	getattr() -> chmod()
ln -s /myfuse/test.txt /myfuse/test-link.txt	getattr() -> symlink()
mv /myfuse/folder /myfuse/fancy-folder	getattr() -> rename()

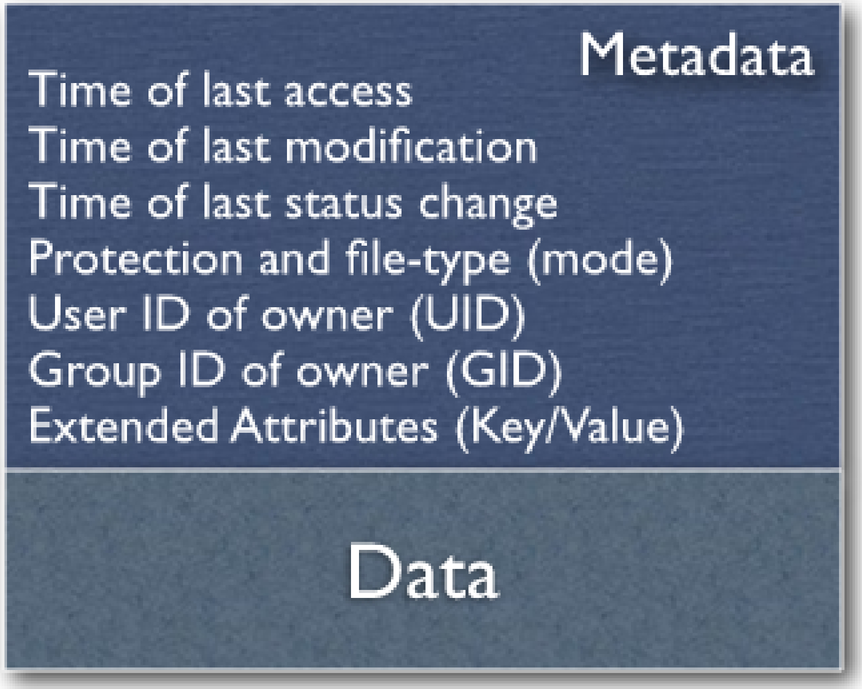
First Code Example!
HTFS

(HashTable File-System)

HTFS Overview

- Traditional Filesystem Object with Metadata (mode, uid, gid, ...)
- HashTable (dict) keys are paths values are Items.

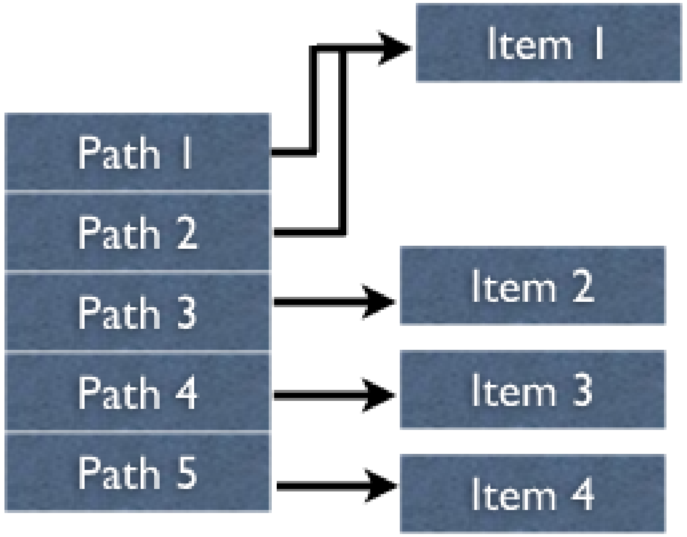
FS Item/Object



Metadata

Time of last access
Time of last modification
Time of last status change
Protection and file-type (mode)
User ID of owner (UID)
Group ID of owner (GID)
Extended Attributes (Key/Value)

Data



Path 1
Path 2
Path 3
Path 4
Path 5

Item 1
Item 2
Item 3
Item 4

(Disk - Storage HashTable)

Item can be a Regular File or Directory or FIFO...

Data is raw data or filename list if item is a directory.

HTFS Item

```
class Item(object):
  def __init__(self, mode, uid, gid):
     # ------------------------------------ Metadata --
     self.atime = time.time()   # time of last acces
     self.mtime = self.atime    # time of last modification
     self.ctime = self.atime    # time of last status change

     self.mode = mode       # protection and file-type
     self.uid  = uid          # user ID of owner
     self.gid  = gid         # group ID of owner

     # Extended Attributes
     self.xattr = {}

     # --- Data -----------
     if stat.S_ISDIR(mode):
         self.data = set()
     else:
         self.data = ''
```

This is a File!
we've metadata
data and even xattr

(Data Helper)

HTFS Item

```
def read(self, offset, length):
    return self.data[offset:offset+length]

def write(self, offset, data):
    length = len(data)
    self.data = self.data[:offset] + data + self.data[offset+length:]
    return length

def truncate(self, length):
    if len(self.data) > length:
        self.data = self.data[:length]
    else:
        self.data += '\x00' * (length - len(self.data))
```

...a couple of utility methods to read/write and interact with data.

HTFS Fuse Operations

```
class HTFS(fuse.Fuse):
  def __init__(self, *args, **kwargs):
   fuse.Fuse.__init__(self, *args, **kwargs)

   self.uid = os.getuid()
   self.gid = os.getgid()

   root_dir = Item(0755 | stat.S_IFDIR, self.uid, self.gid)
   self._storage = {'/': root_dir}
```

File-System must be initialized with the / directory

```
def main():
    server = HTFS()
    server.main()
```

Your FUSE File-System is like a Server...

getattr() is called before any operation. Tells to the VFS if you can access to the specified file and the “State”.

```
def getattr(self, path):
  if not path in self._storage:
     return -errno.ENOENT

   # Lookup Item and fill the stat struct
   item = self._storage[path]
   st = zstat(fuse.Stat())
   st.st_mode  = item.mode
   st.st_uid   = item.uid
   st.st_gid   = item.gid
   st.st_atime = item.atime
   st.st_mtime = item.mtime
   st.st_ctime = item.ctime
   st.st_size  = len(item.data)
   return st
```

(File Operations)

HTFS Fuse Operations

```
def create(self, path, flags, mode):
   self._storage[path] = Item(mode | stat.S_IFREG, self.uid, self.gid)
   self._add_to_parent_dir(path)

def truncate(self, path, len):
     self._storage[path].truncate(len)

def read(self, path, size, offset):
   return self._storage[path].read(offset, size)

def write(self, path, buf, offset):
   return self._storage[path].write(offset, buf)
```

Disk is just a big dictionary...

...and files are items

key = name

value = data

```
def unlink(self, path):
  self._remove_from_parent_dir(path)
  del self._storage[path]

def rename(self, oldpath, newpath):
  item = self._storage.pop(oldpath)
  self._storage[newpath] = item
```

(Directory Operations)

HTFS Fuse Operations

```
def mkdir(self, path, mode):
    self._storage[path] = Item(mode | stat.S_IFDIR, self.uid, self.gid)
    self._add_to_parent_dir(path)

def rmdir(self, path):
    self._remove_from_parent_dir(path)
    del self._storage[path]

def readdir(self, path, offset):
    dir_items = self._storage[path].data
    for item in dir_items:
        yield fuse.Direntry(item)

def _add_to_parent_dir(self, path):
     parent_path = os.path.dirname(path)
     filename = os.path.basename(path)
     self._storage[parent_path].data.add(filename)
```

Directory is a File
that contains
File names
as data!

(XAttr Operations)

HTFS Fuse Operations

```
def setxattr(self, path, name, value, flags):
   self._storage[path].xattr[name] = value

def getxattr(self, path, name, size):
   value = self._storage[path].xattr.get(name, '')
   if size == 0:   # We are asked for size of the value
     return len(value)
   return value

def listxattr(self, path, size):
   attrs = self._storage[path].xattr.keys()
   if size == 0:
      return len(attrs) + len(''.join(attrs))
   return attrs

def removexattr(self, path, name):
   if name in self._storage[path].xattr:
       del self._storage[path].xattr[name]
```

Extended attributes extend the basic attributes associated with files and directories in the file system. They are stored as name:data pairs associated with file system objects

(Other Operations)

HTFS Fuse Operations

Lookup Item,
Access to its
information/data return or
write it.
This is the
File-System's Job

```
def chmod(self, path, mode):
    item = self._storage[path]
    item.mode = mode

def chown(self, path, uid, gid):
   item = self._storage[path]
   item.uid = uid
   item.gid = gid
```

```
def symlink(self, path, newpath):
   item = Item(0644 | stat.S_IFLNK, self.uid, self.gid)
   item.data = path
   self._storage[newpath] = item
   self._add_to_parent_dir(newpath)

def readlink(self, path):
   return self._storage[path].data
```

Symlinks contains just
pointed file path.

Other small Examples

Simulate Tera Byte Files

```
class TBFS(fuse.Fuse):
    def getattr(self, path):
        st = zstat(fuse.Stat())
        if path == '/':
            st.st_mode = 0644 | stat.S_IFDIR
            st.st_size = 1
            return st
        elif path == '/tera.data':
            st.st_mode = 0644 | stat.S_IFREG
            st.st_size = 128 * (2 ** 40)
            return st
        return -errno.ENOENT

    def read(self, path, size, offset):
        return '0' * size

    def readdir(self, path, offset):
        if path == '/':
            yield fuse.Direntry('tera.data')
```

Read-Only FS
with 1 file
of 128TiB

No
Disk/RAM Space
Required!

read()
Send data only
when is requested

X^OR File-System

```
def _xorData(data):
    data = [chr(ord(c) ^ 10) for c in data]
    return string.join(data,“”)

class XorFS(fuse.Fuse):
    ...
    def write(self, path, buf, offset):
        data = _xorData(buf)
        return _writeData(path, offset, data)

    def read(self, path, length, offset):
        data = _readData(path, offset, length)
        return _xorData(data)
    ...
```

```
10101010 ^
01010101 =
---------
11111111 ^
01010101 =
---------
10101010
```

```
res = _xorData(“xor”)
print res // “rex”
res2 = _xorData(res)
print res // “xor”
```

Dup Write File-System

```
class DupFS(fuse.Fuse):
    def __init__(self, *args, **kwargs):
        ...
        fd_disk1 = open(‘/dev/hda1’, ...)
        fd_disk2 = open(‘/dev/hdb5’, ...)
        fd_log = open(‘/home/th30z/testfs.log’, ...)
        fd_net = socket.socket(...)
        ...
    ...
    def write(self, path, buf, offset):
        ...
        disk_write(fd_disk1, path, offset, buf)
        disk_write(fd_disk2, path, offset, buf)
        net_write(fd_net, path, offset, buf)
        log_write(fd_log, path, offset, buf)
        ...
```

Write on your Disk partition 1 and 2.

Send data over Network

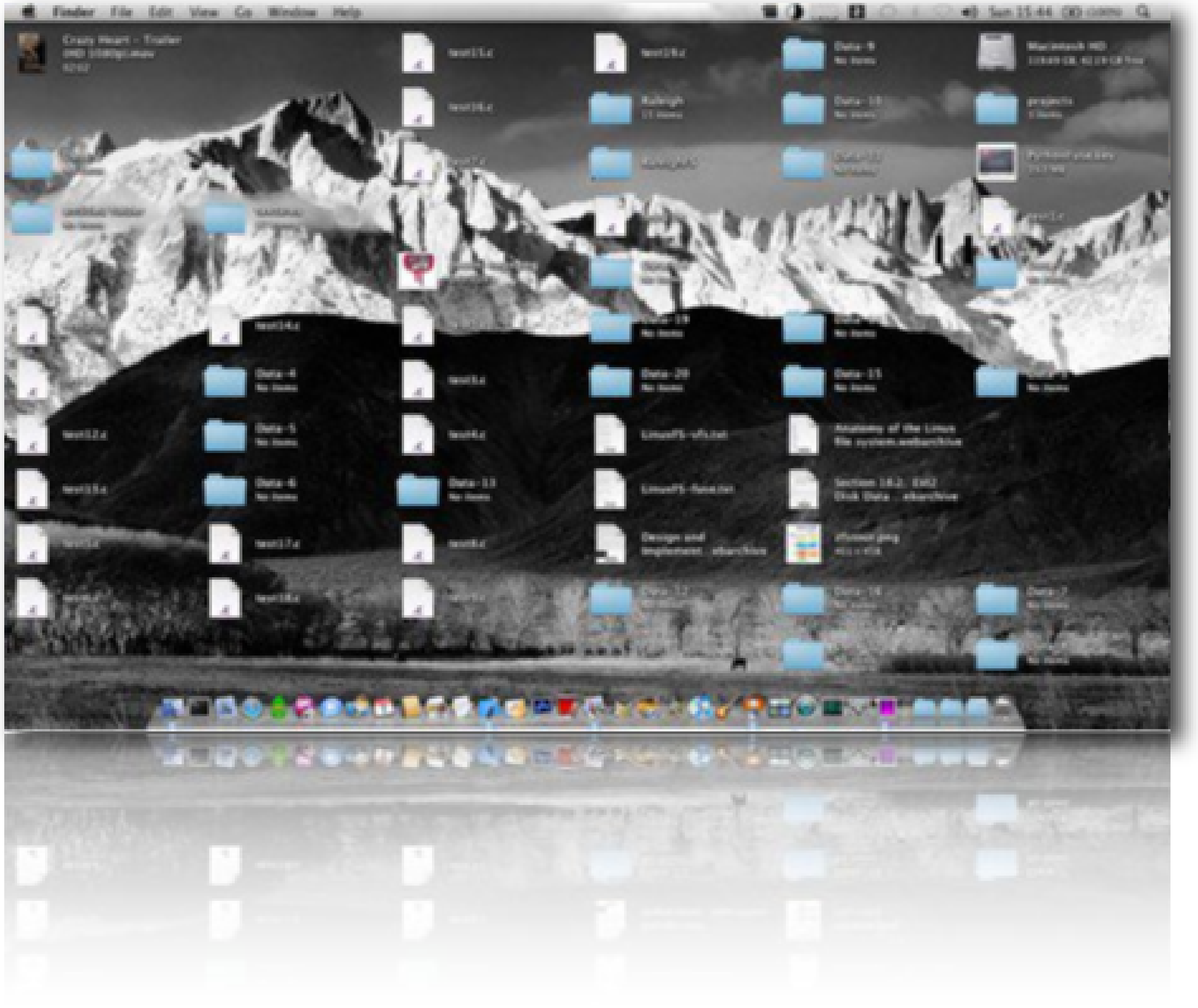
Log your file-system operations

...do other fancy stuff

One more thing

(File and Folders doesn't fit)

Rethink the File-System



I dont't know where I've to place this file...

...Ok, for now Desktop is a good place...

(Mobile/Home Devices)

Rethink the File-System

Small Devices
Small Files
EMails, Text...

We need to lookup
quickly our data.
Tags, Full-Text
Search...

...Encourage people
to view their content
as objects.

(Large Clusters, The Cloud...)

Rethink the File-System

Distributed data

Scalability

Fail over

Cluster Rebalancing

Q&A

Python FUSE

http://mbertozzi.develer.com/python-fuse