'FUSE'ing Python for rapid development of storage efficient file-system

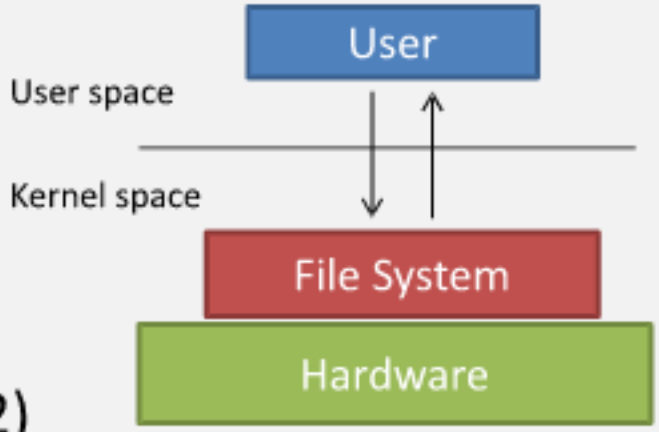
PyCon APAC '12

Singapore, Jun 07-09, 2012

Chetan Giridhar, Vishal Kanaujia

File Systems

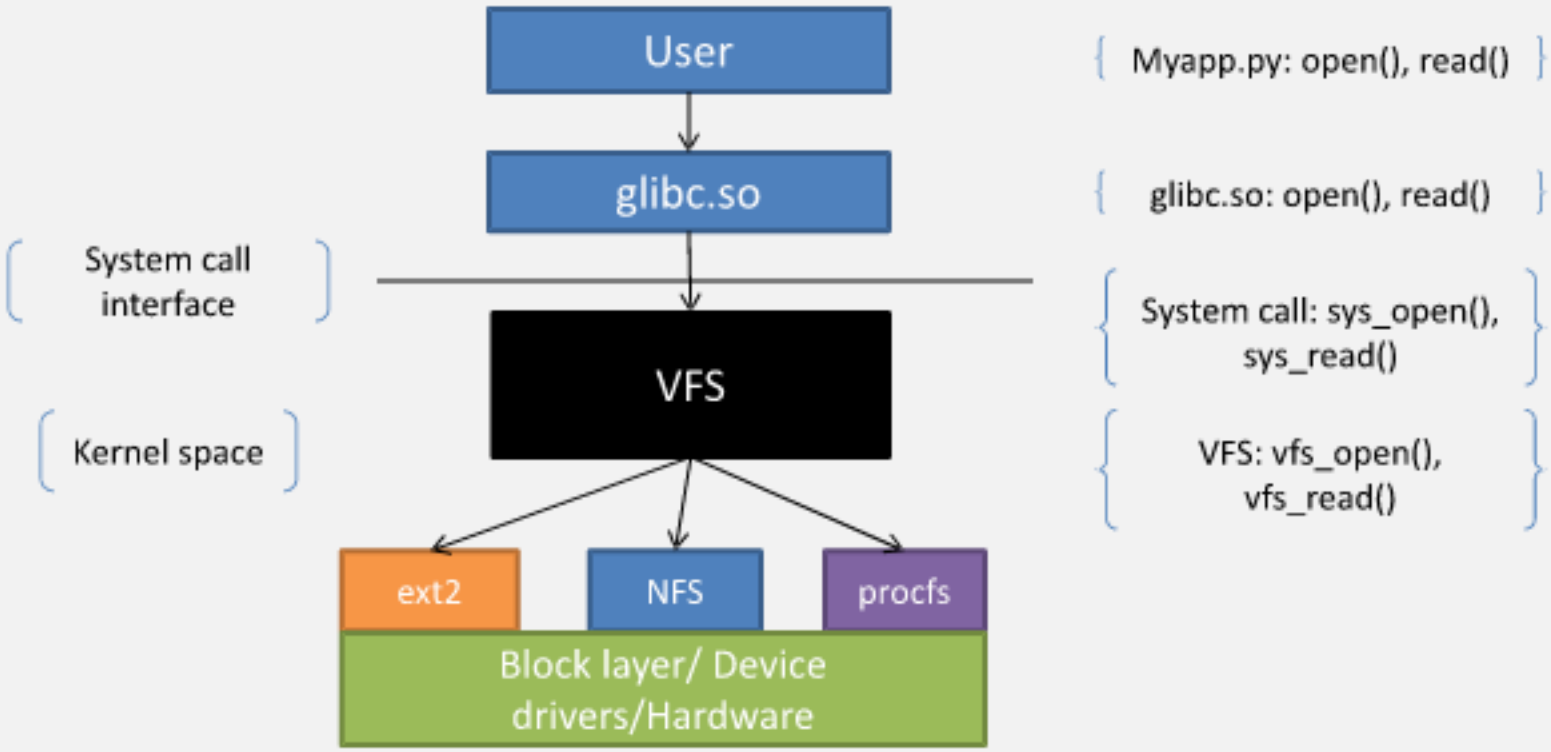
- Provides way to organize, store, retrieve and manage information
- Abstraction layer
- File system:
  - Maps name to an object
  - Objects to file contents
- File system types (format):
  - Media based file systems (FAT, ext2)
  - Network file systems (NFS)
  - Special-purpose file systems (procfs)
- Services
  - open(), read(), write(), close()…



Virtual File-system

- To support multiple FS in *NIX
- VFS is an abstract layer in kernel
- Decouples file system implementation from the interface (POSIX API)
  - Common API serving different file system types
- Handles user calls related to file systems.
  - Implements generic FS actions
  - Directs request to specific code to handle the request
- Associate (and disassociate) devices with instances of the appropriate file system.

File System: VFS



Developing FS in *NIX

- In-kernel file-systems (traditionally)
- It is a complex task
  - Understanding kernel libraries and modules
  - Development experience in kernel space
  - Managing disk i/o
  - Time consuming and tedious development
    - Frequent kernel panic
    - Higher testing efforts
  - Kernel bloating and side effects like security

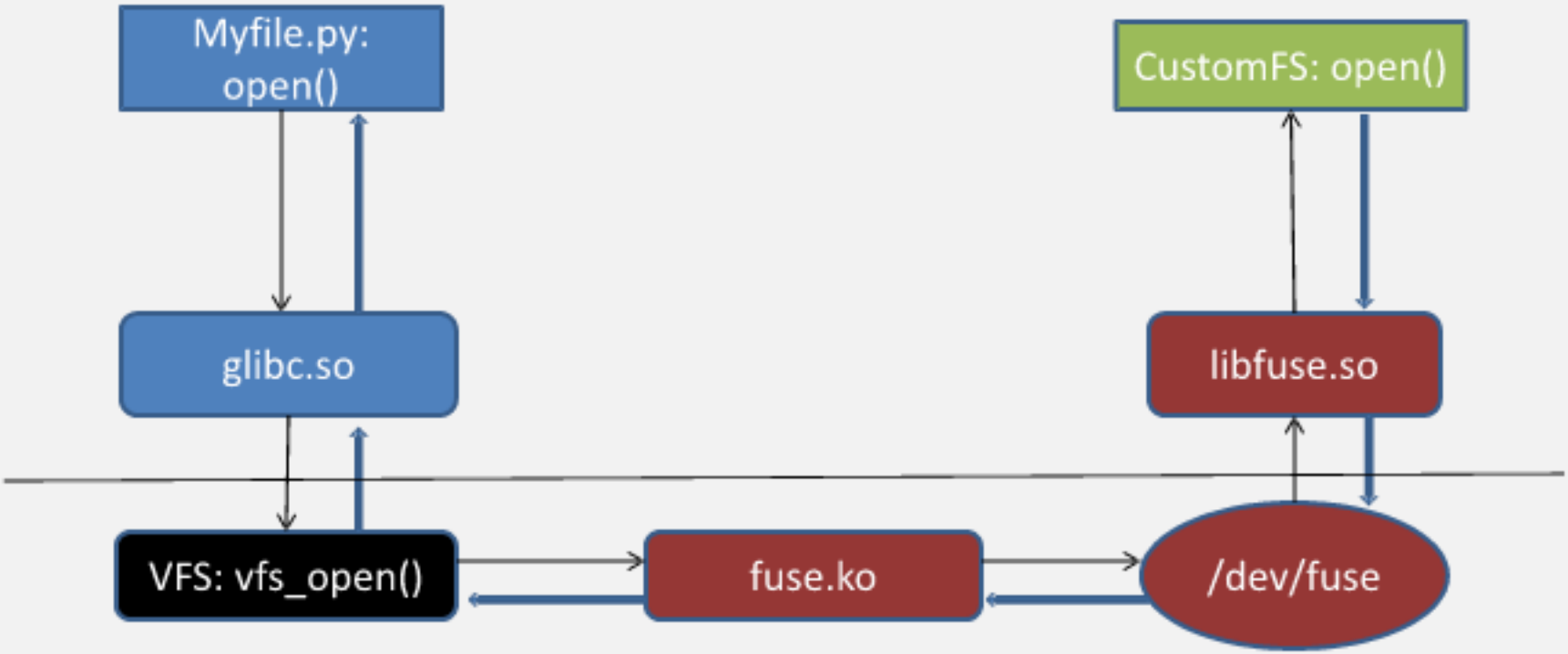
Solution: User space

- In user space:
  - Shorter development cycle
  - Easy to update fixes, test and distribute
  - More flexibility
    - Programming tools, debuggers, and libraries as you have if you were developing standard *NIX applications
- User-space file-systems
  - File systems become regular applications (as opposed to kernel extensions)

FUSE (**F**ilesystem in **USE**rspace)

- Implement a file system in user-space
  - no kernel code required!
- Secure, non-privileged mounts
- User operates on a mounted instance of FS:
  - Unix utilities
  - POSIX libraries
- Useful to develop “virtual” file-systems
  - Allows you to imagine “anything” as a file ☺
  - local disk, across the network, from memory, or any other combination

FUSE: Diving deeper



FUSE | develop

- Choices of development in C, C++, Java, … and of course Python!
- Python interface for FUSE
  - (*FusePython: most popularly used)*
- Open source project
  - http://fuse.sourceforge.net/
- For ubuntu systems:

  $sudo apt-get instatall python-fuse

  $mkdir ./mount_point

  $python myfuse.py ./mount_point

  $fusermount -u ./mount_point

FUSE API Overview

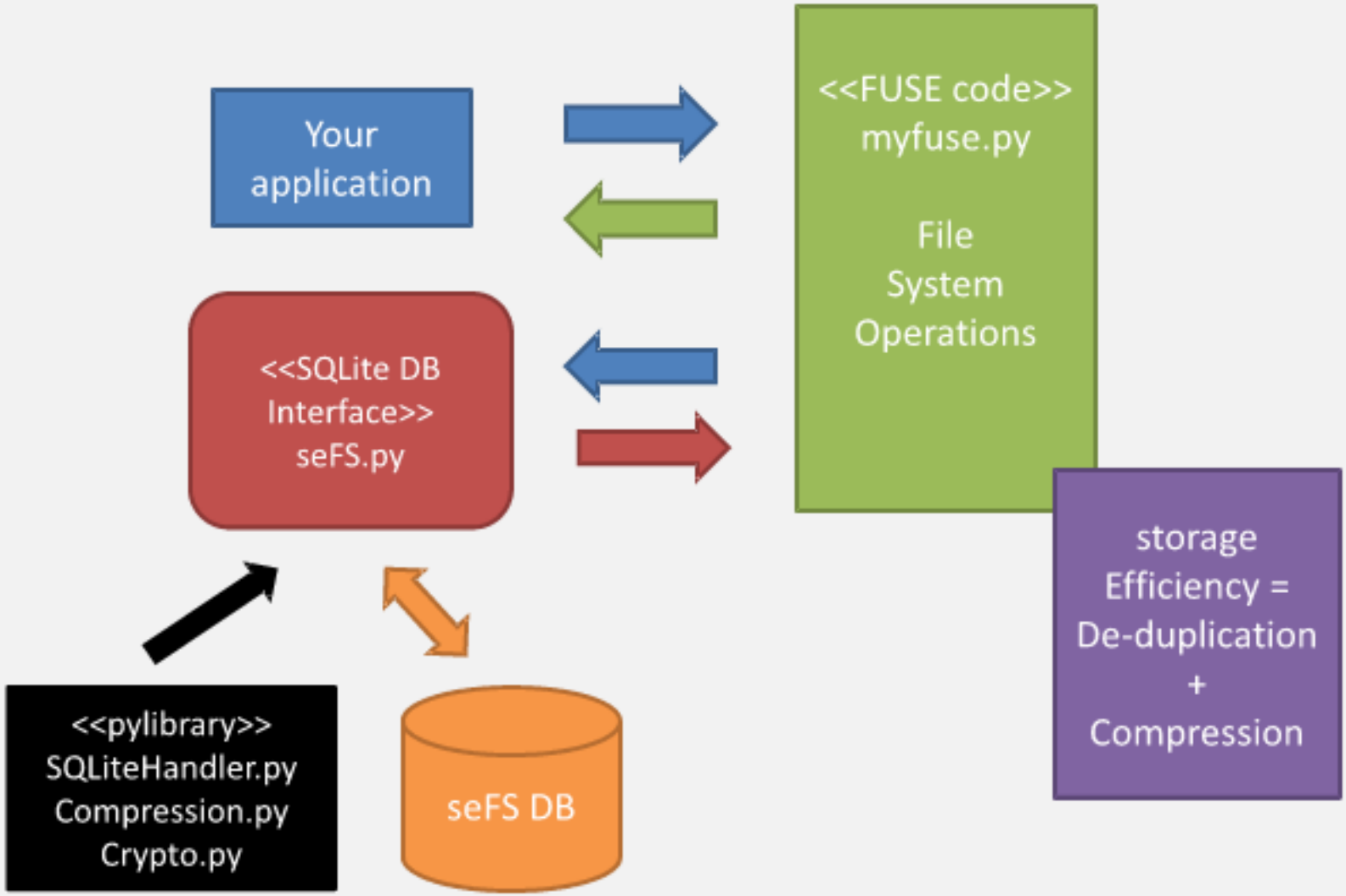
- File management
  - open(path)
  - create(path, mode)
  - read(path, length, offset)
  - write(path, data, offset)
- Directory and file system management
  - unlink(path)
  - readdir(path)
- Metadata operations
  - getattr(path)
  - chmod(path, mode)
  - chown(path, uid, gid)

seFS – storage efficient FS

- A prototype, experimental file system with:
  - Online data de-duplication (SHA1)
  - Compression (text based)
- SQLite
- Ubuntu 11.04, Python-Fuse Bindings
- Provides following services:

open()	write()	chmod()
create()	readdir()	chown()
read()	unlink()	

seFS Architecture



seFS: Database

```
CREATE TABLE metadata(
    "id" INTEGER,
    "abspath" TEXT,
    "length" INTEGER,
    "mtime" TEXT,
    "ctime" TEXT,
    "atime" TEXT,
    "inode" INTEGER);
```


```
CREATE TABLE data(
    "id" INTEGER
    PRIMARY KEY
    AUTOINCREMENT,
    "sha" TEXT,
    "data" BLOB,
    "length" INTEGER,
    "compressed" BLOB);
```

Full View | Item View | Script Output

data table

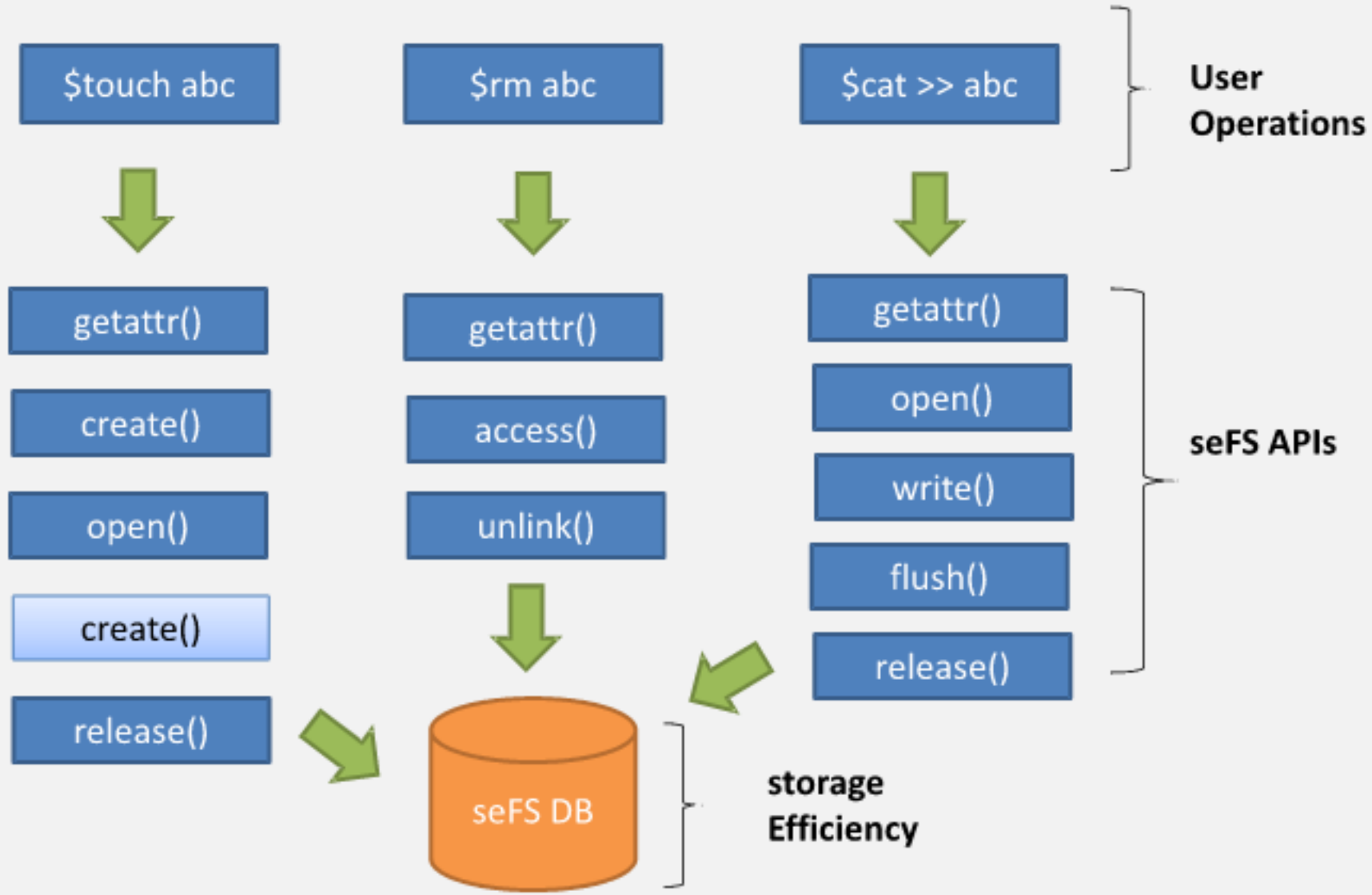
	id	sha	data	length	compressed
1	1	26cdfd1af70617cb768150fe19d24228c881b700	Root of the seFS	16	<function compress at 0xb76bb684>
2	2	22596363b3de40b06f981fb85d82312e8c0ed511	hello world	12	<function compress at 0xb76ba726>
3	3	4e7d9a2d471f438cf98d8239f7eb63909f170b2f	PyCon APAC!!	13	<function compress at 0xb76bc648>

Full View | Item View | Script Output

metadata table

	id	abspath	length	mtime	ctime	atime	inode
1	1	/	16	1338894911	1338894911	1338894911	1
2	2	/hello	12	1338894935.0	1338894935.0	1338894935.0	2
3	3	/pycon	13	1338894942.0	1338894942.0	1338894942.0	3
4	2	/hello_again	12	1338894980.0	1338894980.0	1338894980.0	4

seFS API flow



seFS: Code

```
#!/usr/bin/python

import fuse
import stat
import time
from seFS import seFS

fuse.fuse_python_api = (0, 2)

class MyFS(fuse.Fuse):
    def __init__(self, *args, **kw):
        fuse.Fuse.__init__(self, *args, *kw)

        # Set some options required by the
        # Python FUSE binding.
          self.flags = 0
          self.multithreaded = 0
          self.fd = 0

          self.sefs = seFS()
          ret = self.sefs.open('/')
          self.sefs.write('/', "Root of the seFS")
          t = int(time.time())
          mytime = (t, t, t)
          ret = self.sefs.utime('/', mytime)
          self.sefs.setinode('/', 1)
```

seFS: Code (1)

```
def getattr(self, path):
    sefs = seFS()
    stat = fuse.stat()
    context  = fuse.FuseGetContext()
    #Root
    if path == '/':
         stat.stat_nlink = 2
         stat.stat_mode = stat.S_IFDIR | 0755
    else:
         stat.stat_mode = stat.S_IFREG | 0777
         stat.stat_nlink = 1
         stat.stat_uid, stat.stat_gid =
                      (context ['uid'], context
['gid'])

         # Search for this path in DB
         ret = sefs.search(path)
         # If file exists in DB, get its times
         if ret is True:
             tup = sefs.getutime(path)
             stat.stat_mtime =
               int(tup[0].strip().split('.')[0])
             stat.stat_ctime =
               int(tup[1].strip().split('.')[0])
             stat.stat_atime =
               int(tup[2].strip().split('.')[0])
             stat.stat_ino =
                         int(sefs.getinode(path))

             # Get the file size from DB
             if sefs.getlength(path) is not None:
                stat.stat_size =
                         int(sefs.getlength(path))
             else:
                 stat.stat_size = 0
             return stat
         else:
            return - errno.ENOENT
```

seFS: Code (2)

```
def create(self, path,flags=None,mode=None):
    sefs = seFS()
    ret = self.open(path, flags)
    if ret == -errno.ENOENT:
    #Create the file in database
        ret = sefs.open(path)
        t = int(time.time())
        mytime = (t, t, t)
        ret = sefs.utime(path, mytime)
        self.fd = len(sefs.ls())
        sefs.setinode(path, self.fd)
        return 0
```

```
def write(self, path, data, offset):
    length = len(data)
    sefs = seFS()
    ret = sefs.write(path, data)
    return length
```

seFS: Learning

- Design your file system and define the objectives first, before development
  - *skip* implementing functionality your file system doesn't intend to support
- Database schema is crucial
- Knowledge on FUSE API is essential
  - FUSE APIs are look-alike to standard POSIX APIs
  - Limited documentation of FUSE API ☹
- Performance?

Conclusion

- Development of FS is very easy with FUSE
- Python aids RAD with Python-Fuse bindings
- seFS: Thought provoking implementation
- Creative applications – your needs and objectives
- When are you developing your own File system?! ☺

Further Read

- Sample Fuse based File systems
  - Sshfs
  - YoutubeFS
  - Dedupfs
  - GlusterFS
- Python-Fuse bindings
  - http://fuse.sourceforge.net/
- Linux internal manual

Contact Us

- Chetan Giridhar
  - http://technobeans.com
  - cjgiridhar@gmail.com
- Vishal Kanaujia
  - http://freethreads.wordpress.com
  - vishalkanaujia@gmail.com

Backup

Agenda

- The motivation
- Intro to *NIX File Systems
- Trade off: code in user and kernel space
- FUSE?
- Hold on – What's VFS?
- Diving into FUSE internals
- Design and develop your own File System with Python-FUSE bindings
- Lessons learnt
- Python-FUSE: Creative applications/ Use-cases

User-space and Kernel space

- Kernel-space
  - Kernel code including device drivers
  - Kernel resources (hardware)
  - Privileged user
- User-space
  - User application runs
  - Libraries dealing with kernel space
  - System resources

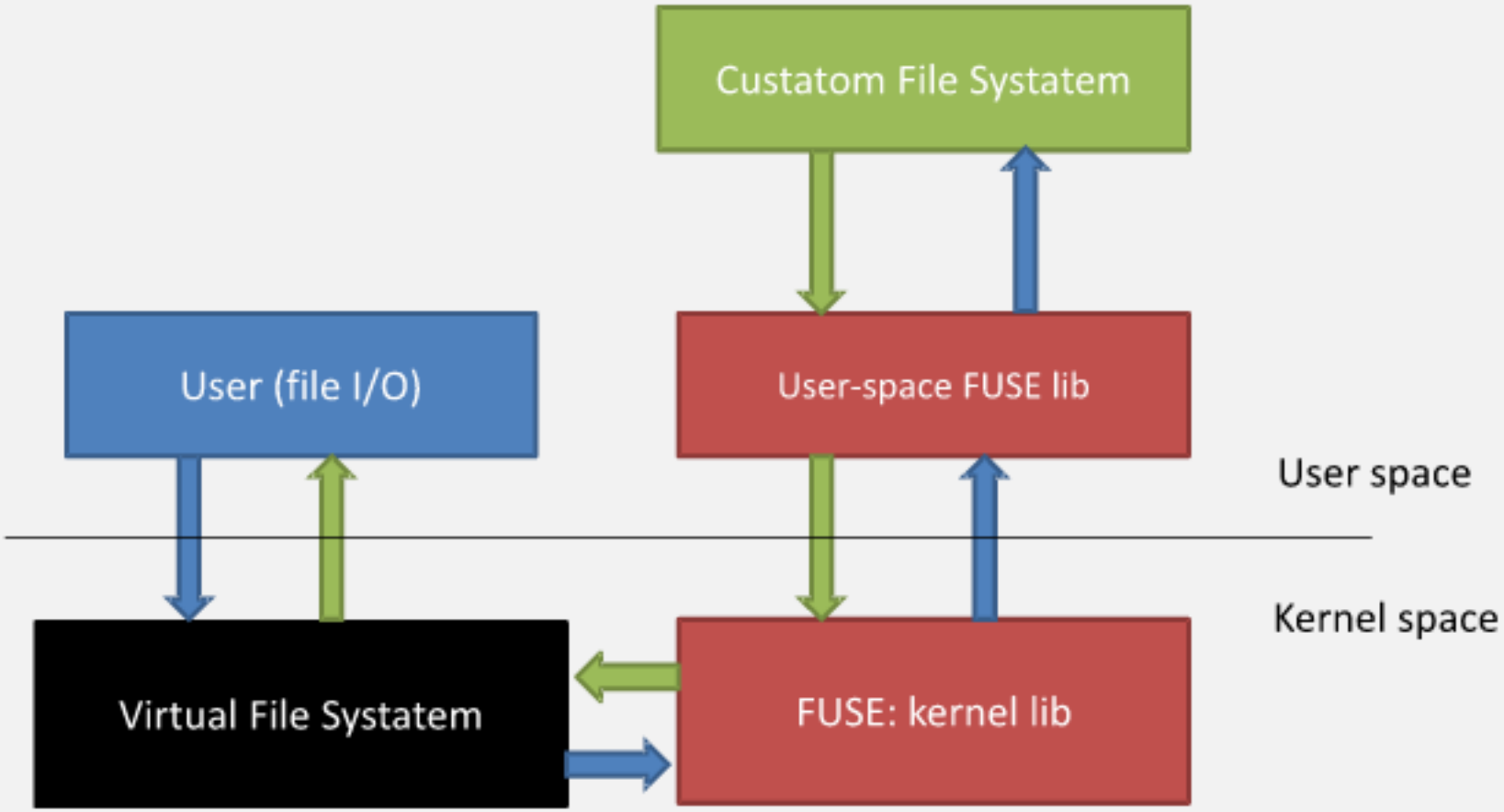
Virtual File-system

- To support multiple FS in *NIX
- VFS is an abstract layer in kernel
- Decouples file system implementation from the interface (POSIX API)
  - Common API serving different file system types
- Handles user calls related to file systems.
  - Implements generic FS actions
  - Directs request to specific code to handle the request
- Associate (and disassociate) devices with instances of the appropriate file system.

FUSE: Internals

- Three major components:
  - Userspace library (libfuse.*)
  - Kernel module (fuse.ko)
  - Mount utility (fusermount)
- Kernel module hooks in to VFS
  - Provides a special device "/dev/fuse"
    - Can be accessed by a user-space process
    - Interface: user-space application and fuse kernel module
    - Read/ writes occur on a file descriptor of /dev/fuse

FUSE Workflow



Facts and figures

- seFS – online storage efficiency
- De-duplication/ compression
  - Managed catalogue information (file meta-data rarely changes)
  - Compression encoded information
- Quick and easy prototyping (Proof of concept)
- Large dataset generation
  - Data generated on demand

Creative applications: FUSE based File systems

- SSHFS: Provides access to a remote file-system through SSH
- WikipediaFS: View and edit Wikipedia articles as if they were real files
- GlusterFS: Clustered Distributed Filesystem having capability to scale up to several petabytes.
- HDFS: FUSE bindings exist for the open source Hadoop distributed file system
- seFS: You know it already ☺