Exploring FSKit

Filesystems in userspace

Sharvil Shah — OBTSv8

First off: Huge thanks to Andy and Patrick

And everyone who makes the conference happen!



whoami

Sharvil Shah

- I like writing code, and like creating security tools
- Some of you may know me from my osquery work
- And I do consulting/development through my firm Orchard Labs

Reach out! I am always happy to chat

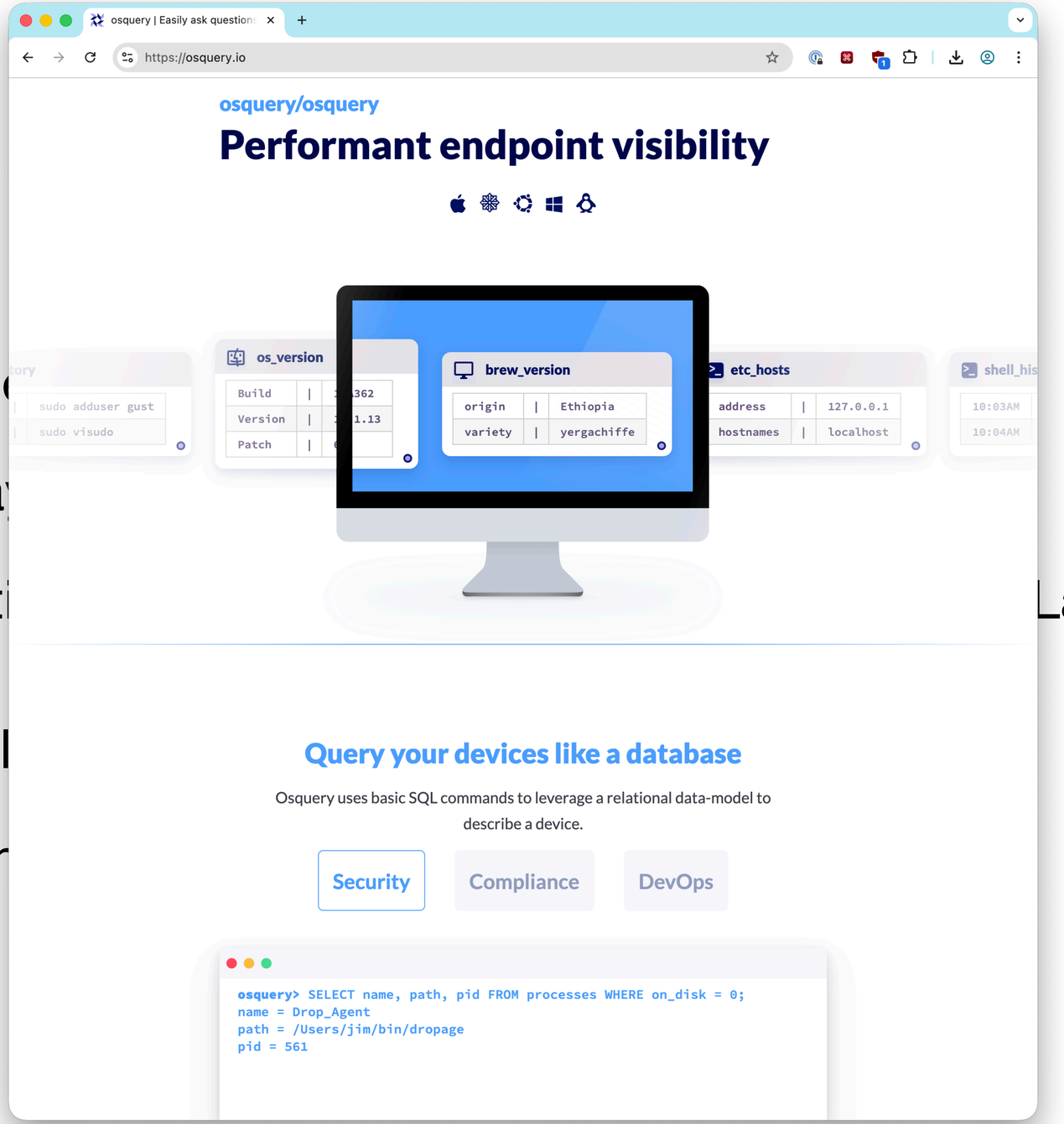
Twitter: sharvil | email: s@sharvil.io | github: sharvilshah

whoami

- I like writing cod
- Some of you ma
- And I do consult Labs

Reach out! I am al

Twitter: sharvil | er



whoami

Sharvil Shah

- I like writing code, and like creating security tools
- Some of you may know me from my osquery work
- And I do consulting/development through my firm Orchard Labs

Reach out! I am always happy to chat

Twitter: sharvil | email: s@sharvil.io | github: sharvilshah

Three parts

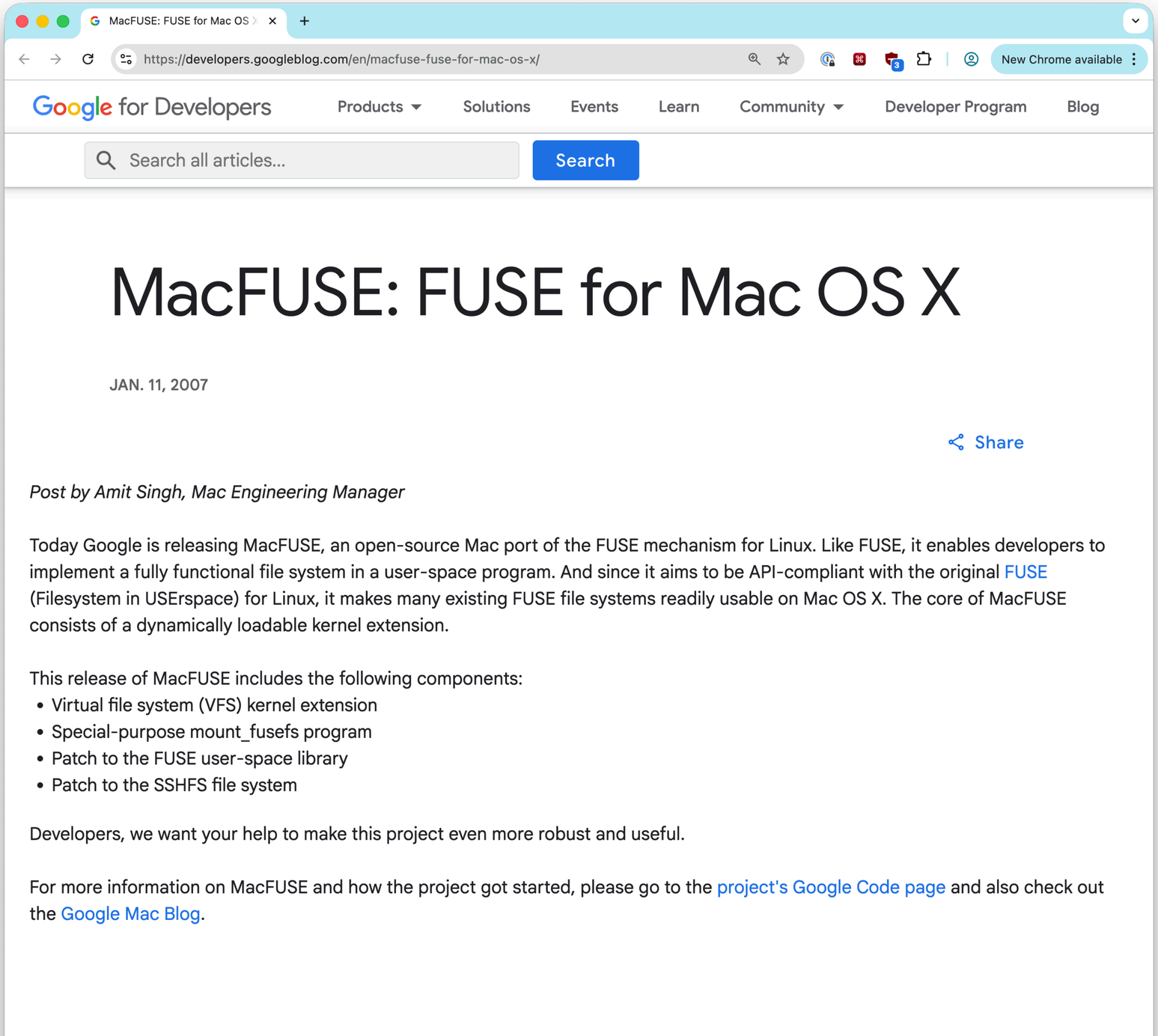
- A brief history of user space filesystems on macOS
- FSKit internals and implementation
- Detection and Deception use cases

A brief history



MacFUSE

- Started by Amit Singh
- At Google, back in 2007



Obj-C RuntimeFS

bbum's weblog-o-mat » Blog
https://web.archive.org/web/20120209215338/http://www.friday.com/bbum/2008/01/10/silly-hack-of-the-day-mount-the-obj-c-runtime-as-a-filesystem/
New Chrome available
Can You Chip In? We're celebrating our 1 trillionth archived web page. If you find our archive useful, learn how you can **help us fundraise!**
INTERNET ARCHIVE WayBackMachine
http://www.friday.com/bbum/2008/01/10/silly-hack-of-the-day-mount-the-obj-c-runtime-as-a-filesystem/
Go
53 captures
11 Feb 2008 – 11 Nov 2024
JAN FEB MAY
2010 09 2012 2013
About this capture

« Yes, The Bay Area Has Weather

What is in your everything drawer? »

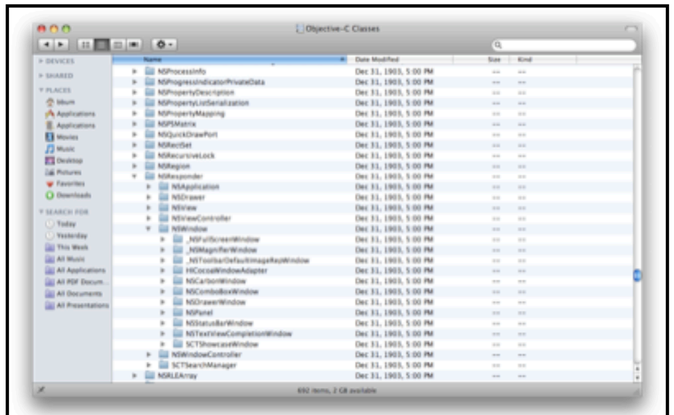
Silly Hack of the Day: Mount the Obj-C Runtime as a Filesystem

Update: CocoaHacks was quite a bit of fun. Not only did I demonstrate the rather silly little hack, but I ran it fully garbage collected. With one simple change (that should be in the next release), the MacFUSE framework works just fine with Garbage Collection enabled!

The best compliment came from Amit Singh; something along the lines of "That is absolutely the strangest filesystem I have seen."

In any case, my little hack repository has been updated to be GC only. **retain/release/autorelease/dealloc** are dead to me (in this project, anyway).

If you want to build MacFUSE with GC enabled, simply set **Objective-C Garbage Collection** to "**Supported**" in Xcode in the MacFUSE framework project's build settings.



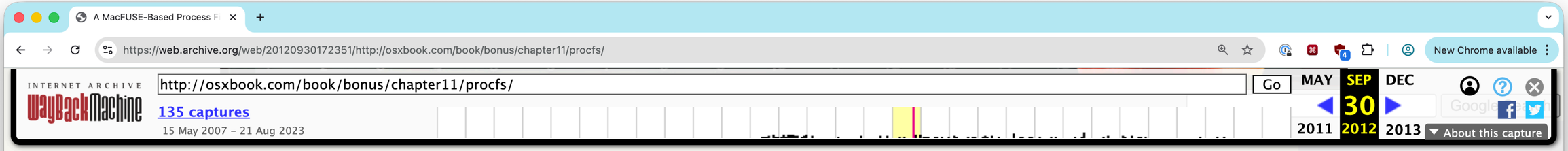
At left, is a screenshot of the Finder browsing the Objective-C classes active in the runtime of a little Cocoa app that I wrote this evening.

The little app is called RuntimeFS and it contains a simple bit of code that traipses through the Objective-C runtime and collects information about the Objective-C classes encountered. This information is then barfed up by the elegantly simple delegate like API required by MacFuse to create a filesystem.

Let me restate that: MacFuse kicks ass. The Objective-C API is trivially easy to use. Trivially easy. I implemented this little hack in less than two hours, not having looked at the MacFuse API before. Nor did I read the docs; just looked at this example and one header file. I have implemented filesystems in a couple of different languages. Filesystems are hard. Or was. Not

MacFuse based ProcFS

Enabled a bunch of filesystems



A MacFUSE-Based Process File System for Mac OS X

© Amit Singh. All Rights Reserved.

Written in May 2007

Processes as Files

The *process file system* ("procfs" for brevity, or simply "/proc", because that's where it is usually mounted) has become a common entity on Unix-like systems. For example, Solaris, Linux, and the various modern BSDs all have procfs. In general, procfs uses the file metaphor to provide both a view of currently running processes and perhaps an interface to control them. However, some procfs implementations export so many types of information that they are nothing short of the proverbial kitchen sink. Implementations can also differ in the format they use to export such information. For example, Solaris procfs favors the binary format philosophy, assuming that developers will write interesting user-space tools that use that "raw" information. In contrast, Linux favors formatted text to export information through procfs, allowing end users and administrators to simply view that information using familiar file utilities.

> Tom J. Killian created the first implementation of a process file system. It was for Eighth Edition UNIX. See **T. J. Killian, "Processes as Files,"** *USENIX Summer Conference Proceedings, Salt Lake City, UT, USA (June 1984).*

`/proc` on Mac OS X (not)

Mac OS X does *not* provide a process file system. It does provide alternative interfaces such as `sysctl(3)` and the now obsolete `kvm(3)`. The `sysctl(3)` interface provides read and write access to a management information base (MIB) whose contents are various categories of kernel information, such as information related to processes, file systems, virtual memory, networking, and debugging in general. When the `/dev/kmem` device is available, the `kvm(3)` interface provides access to raw kernel memory. Besides, the I/O Kit programming interfaces and tools such as `ioreg` and `IORegistryExplorer.app` allow user-space inspection of a variety of kernel information on Mac OS X.

> The `sysctl()` system call was introduced in 4.4BSD as a safe, reliable, and portable (across kernel versions) way to perform user-kernel data exchange.

the book
- Preface
- Table of Contents
- Read Reviews
- Source Code
- Errata
- Buy Now

the book — extras
- Bonus Content
- FAQ

software
- Downloads
- Bug Reports

meta
- Blog
- Forums
- Home

about
- About the Author
- Résumé
- Contact

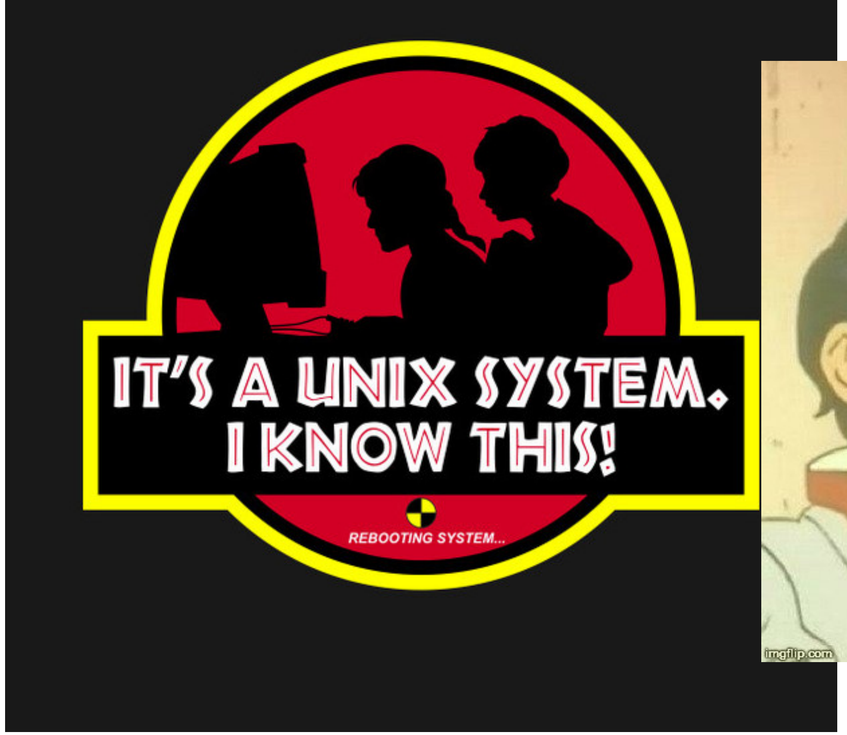
elsewhere
- kernelthread.com

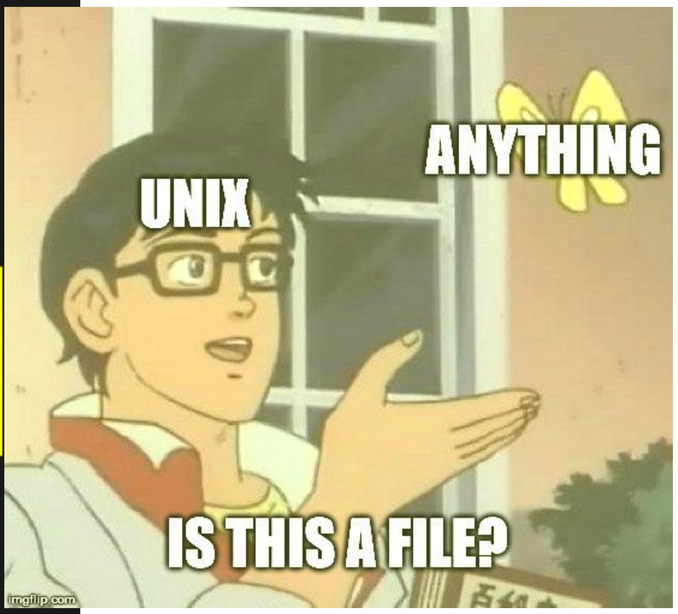
- ProcFS enabled a very powerful idea
- Exposing system internals and telemetry in a digestible way like files on disk
- osquery also does this, exposes OS data in a virtual sqlite interface

- ProcFS enabled a very powerful idea
- Exposing system internals and telemetry in a digestible way like files on disk
- osquery also does this, exposes OS data in a virtual sqlite interface

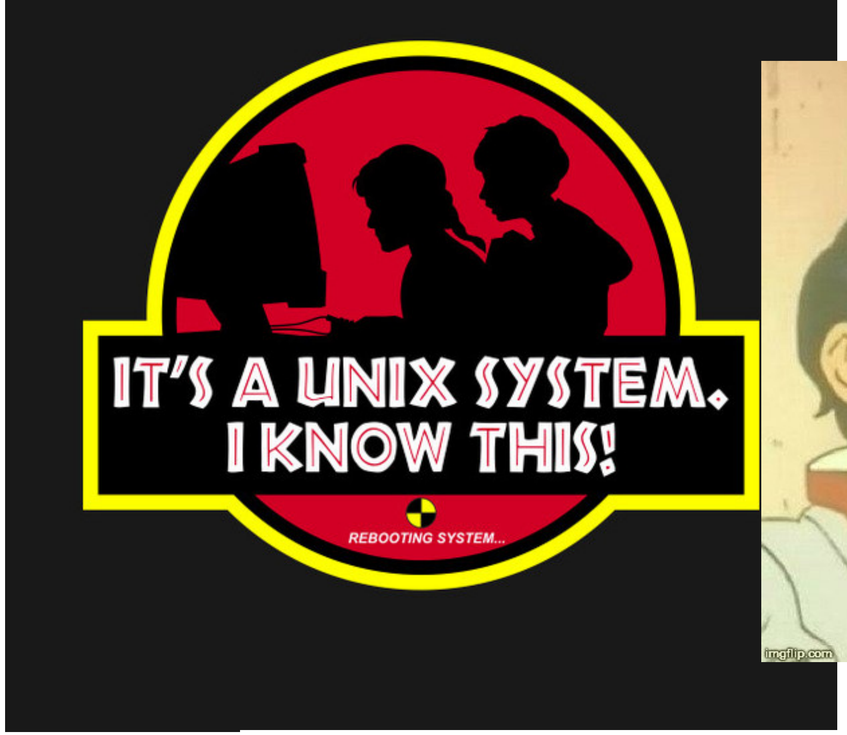


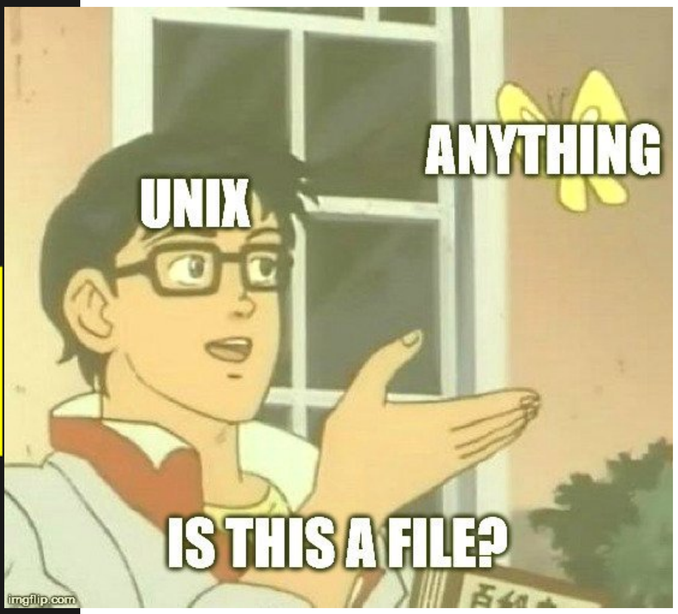
- ProcFS enabled a very powerful idea
- Exposing system internals and telemetry in a digestible way like files on disk
- osquery also does this, exposes OS data in a virtual sqlite interface

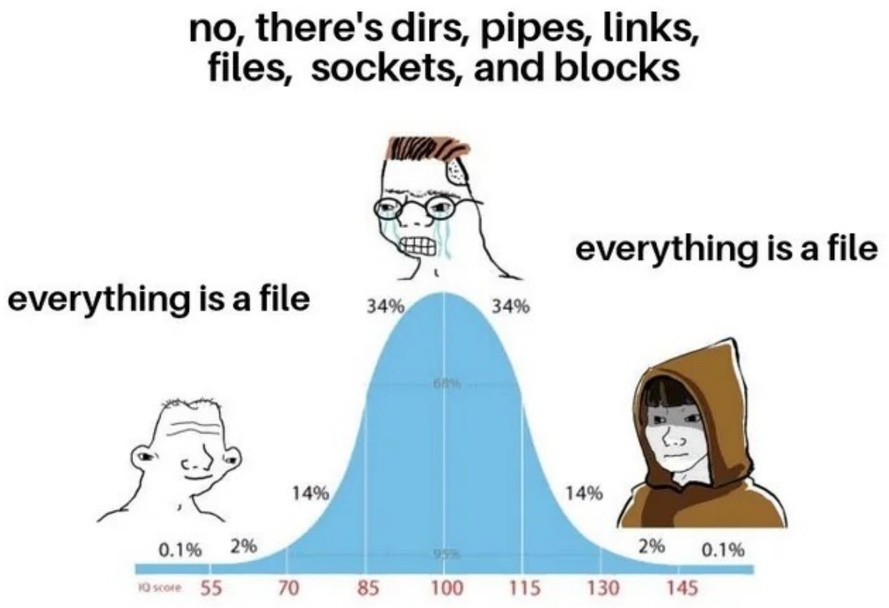




- ProcFS enabled a very powerful idea
- Exposing system internals and telemetry in a digestible way like files on disk
- osquery also does this, exposes OS data in a virtual sqlite interface







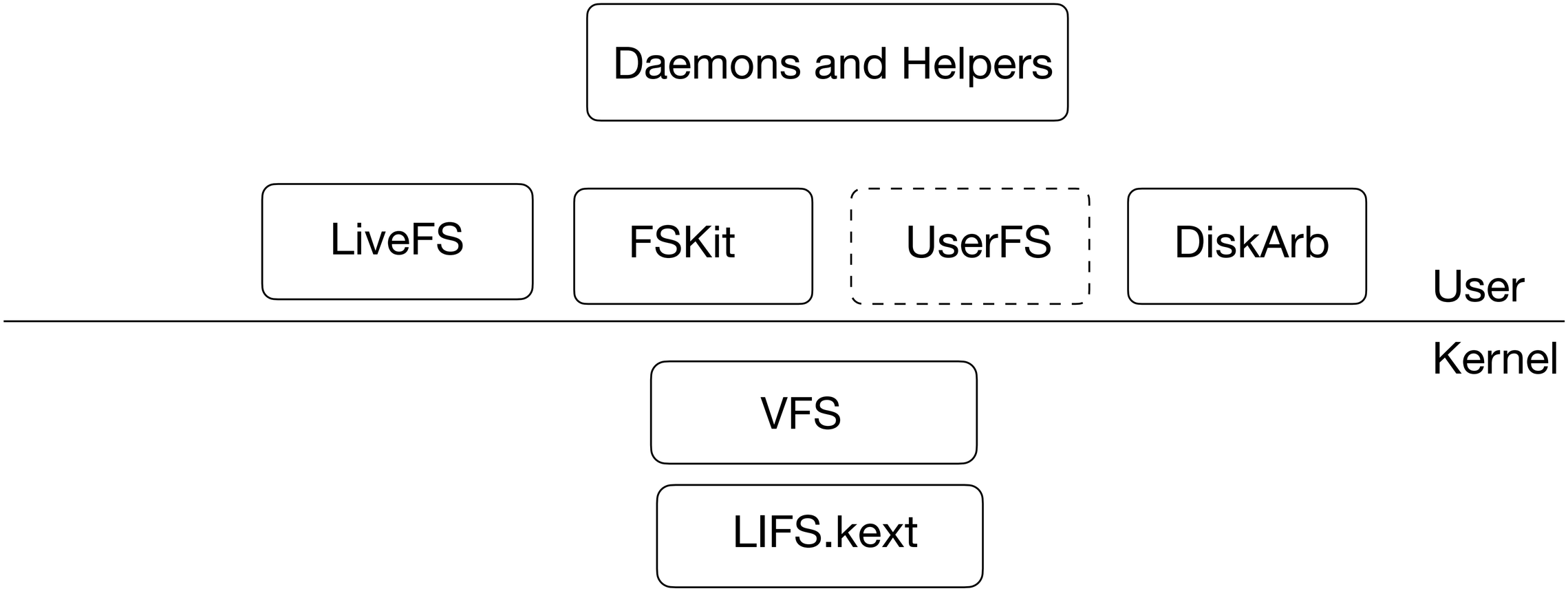
- FSKit
- Kernel —> Userspace path
  - EndpointSecurity
  - DriverKit
  - And now FSKit

FSKit

Origins

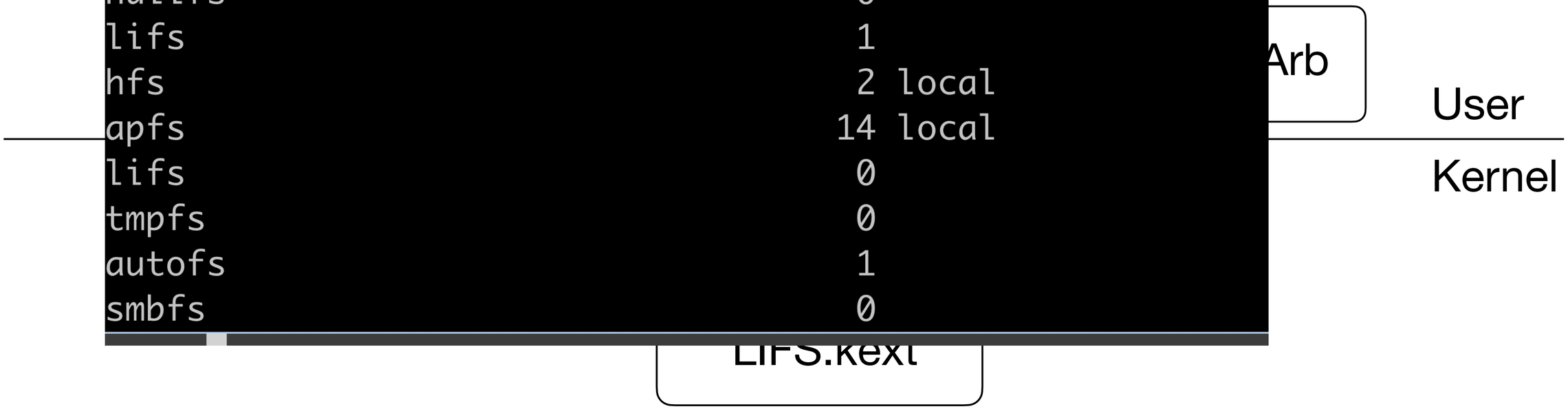
- Files app on iOS 11
- Files app added support for mount USB drives and SMB shares in iOS 13

FSKit

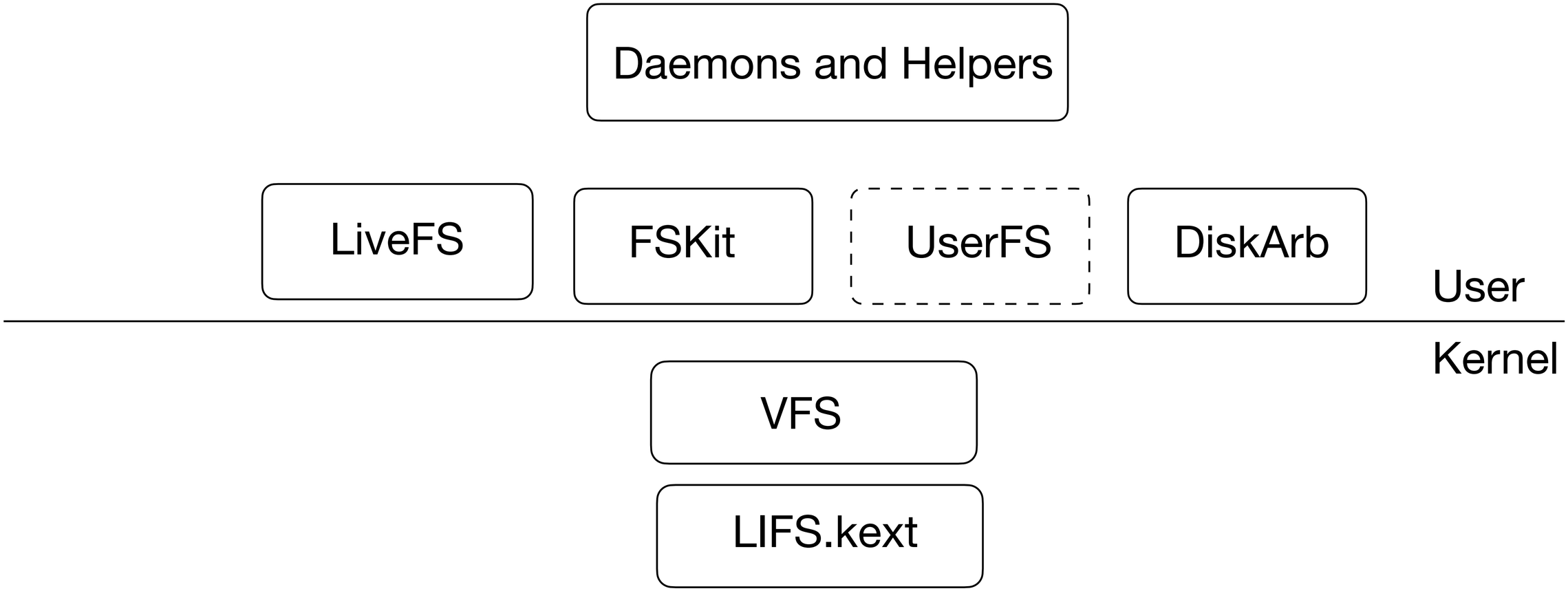

Daemons and Helpers
LiveFS
FSKit
UserFS
DiskArb
User
Kernel
VFS
LIFS.kext

FSKit

```
➜  ~ lsvfs
Filesystem                          Refs Flags
------------------------------ ----- ---------------
nfs                                   0
devfs                                 1
nullfs                                0
lifs                                  1
hfs                                   2 local
apfs                                 14 local
lifs                                  0
tmpfs                                 0
autofs                                1
smbfs                                 0
```



FSKit


Daemons and Helpers
LiveFS
FSKit
UserFS
DiskArb
User
Kernel
VFS
LIFS.kext

```
+-o com_apple_filesystems_lifs  <class com_apple_filesystems_lifs, ... {
  |   "IOProbeScore" = 0
  |   "CFBundleIdentifier" = "com.apple.filesystems.lifs"
  |   "IOMatchCategory" = "com_apple_filesystems_lifs"
  |   "IOClass" = "com_apple_filesystems_lifs"
  |   "IOPersonalityPublisher" = "com.apple.filesystems.lifs"
  |   "IOProviderClass" = "IOResources"
  |   "CFBundleIdentifierKernel" = "com.apple.filesystems.lifs"
  |   "IOMatchedAtBoot" = Yes
  |   "IOUserClientClass" = "AppleLIFSUserClient"
  |   "IOResourceMatch" = "IOBSD"
  | }
  +-o AppleLIFSUserClient  <class AppleLIFSUserClient, id 0x10004a00b, !registered, !matched,
active, busy 0, retain 5> {
  |     "IOUserClientCreator" = "pid 40361, fskitd"
  |     "IOUserClientDefaultLocking" = Yes
  |   }
  +-o AppleLIFSUserClient  <class AppleLIFSUserClient, id 0x10004a00c, !registered, !matched,
active, busy 0, retain 6> {
  |     "IOUserClientCreator" = "pid 40364, com.apple.fskit."
  |     "IOUserClientDefaultLocking" = Yes
  |   }
  +-o AppleLIFSUserClient  <class AppleLIFSUserClient, id 0x10004a00e, !registered, !matched,
active, busy 0, retain 6> {
  |     "IOUserClientCreator" = "pid 40367, com.apple.fskit."
  |     "IOUserClientDefaultLocking" = Yes
```

```
osquery> select pid, name, ... from processes where pid = 40364;
        pid = 40364
       name = com.apple.fskit.exfat

       path = /System/Library/ExtensionKit/Extensions/
com.apple.fskit.exfat.appex/Contents/MacOS/com.apple.fskit.exfat

    cmdline = /System/Library/ExtensionKit/Extensions/
com.apple.fskit.exfat.appex/Contents/MacOS/com.apple.fskit.exfat
     -LaunchArguments
eyJ0eXBlIjoxLCJlbmhhbmNlZFNlY3VyaXR5IjpmYWxzZSwic2VydmljZU5hbWUiOiJjb2
0uYXBwbGUuZnNraXQuZXhmYXQifQ==

      state = R
        cwd = /Users/sharvil/Library/Containers/com.apple.fskit.exfat/
     parent = 1
        uid = 501
        gid = 20
         . . .
```

LaunchArguments for /S/L/ExtensionKit…/com.apple.fskit.exfat

```
echo
eyJ0eXBlIjoxLCJlbmhhbmNlZFNlY3VyaXR5IjpmYWxzZSwic2VydmljZU5hbWUiOiJjb2
0uYXBwbGUuZnNraXQuZXhmYXQifQ== | base64 -D | jq .

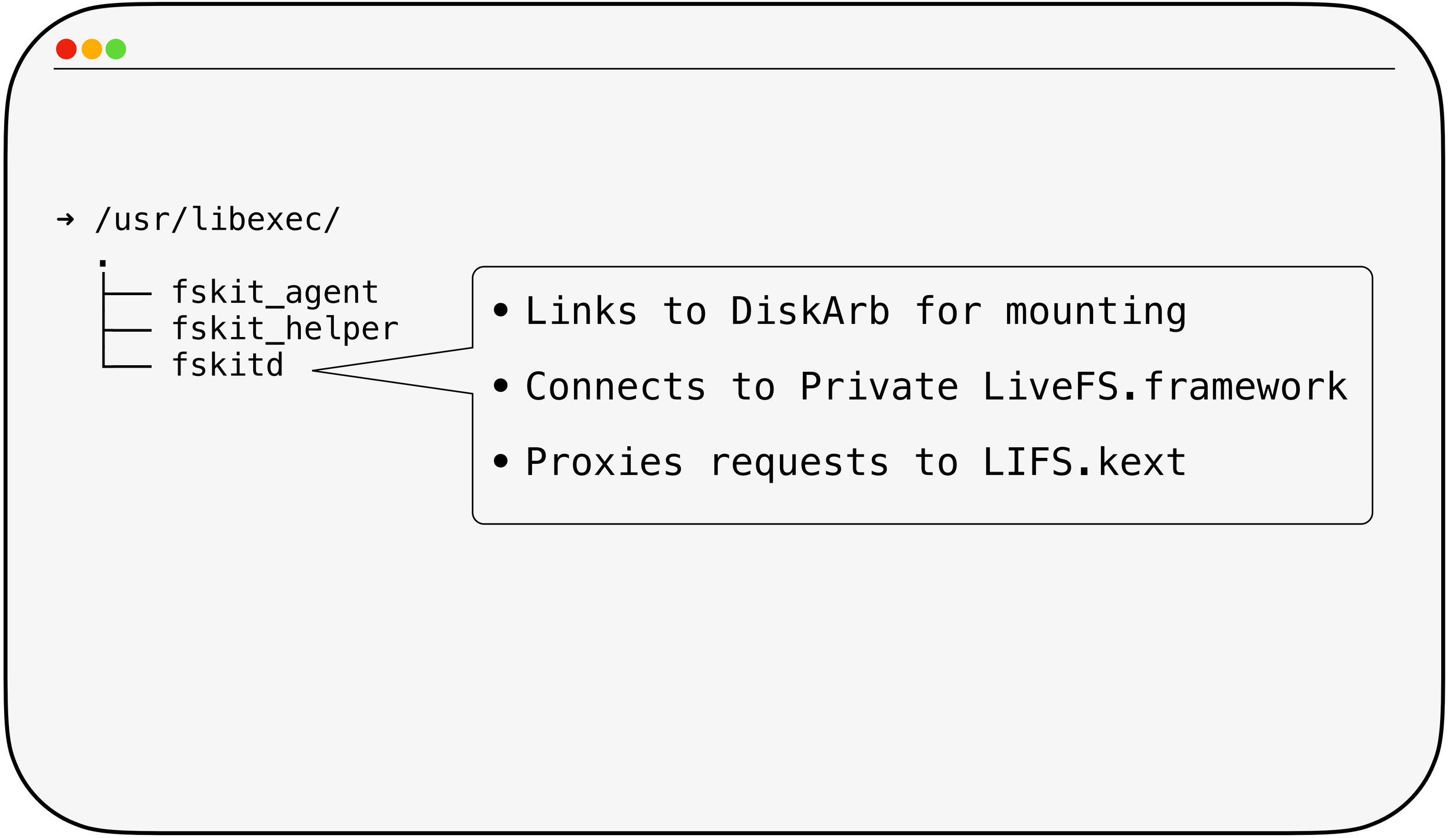
{
  "type": 1,
  "enhancedSecurity": false,
  "serviceName": "com.apple.fskit.exfat"
}
```

LaunchArguments for /S/L/ExtensionKit…/com.apple.fskit.ModuleEnablement

```
→ echo
eyJzZXJ2aWNlTmFtZSI6ImNvbS5hcHBsZS5mc2tpdC5Nb2R1bGVFbmFibGVtZW50IiwidH
lwZSI6MiwiZW5oYW5jZWRTZWN1cml0eSI6ZmFsc2V9 | base64 -D | jq .


{
  "serviceName": "com.apple.fskit.ModuleEnablement",
  "type": 2,
  "enhancedSecurity": false
}
```


→ /usr/libexec/
.
├── fskit_agent
├── fskit_helper
└── fskitd


→ /usr/libexec/
.
├── fskit_agent
├── fskit_helper
└── fskitd
• Links to DiskArb for mounting
• Connects to Private LiveFS.framework
• Proxies requests to LIFS.kext

Quick Recap

- Filesystems are pretty cool and essential
- Apple is on the path to move components out of the kernel
- FSKit has origins on iOS and move ported to macOS
- FSKit modules are AppExtension
- fskitd is the userspace daemon that communicates with LIFS.kext

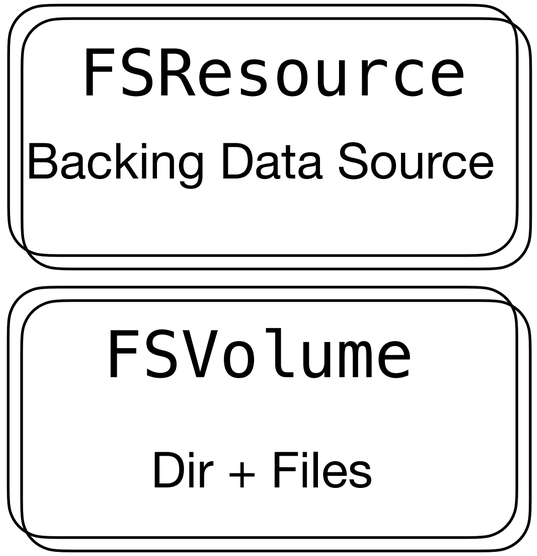
Implementing a filesystem with FSKIT

Three Major Concepts

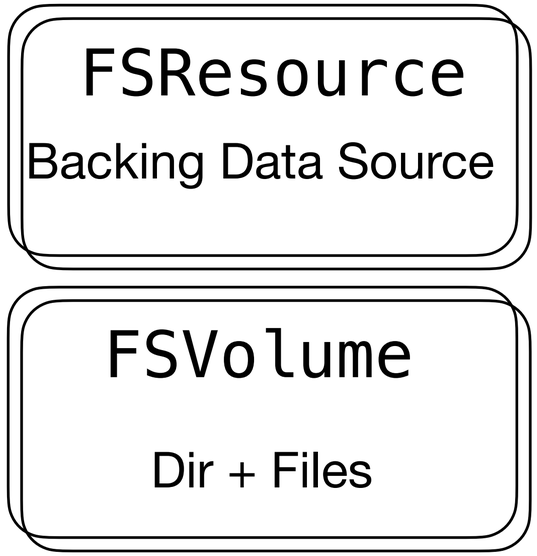
Three Major Concepts



Three Major Concepts



Three Major Concepts

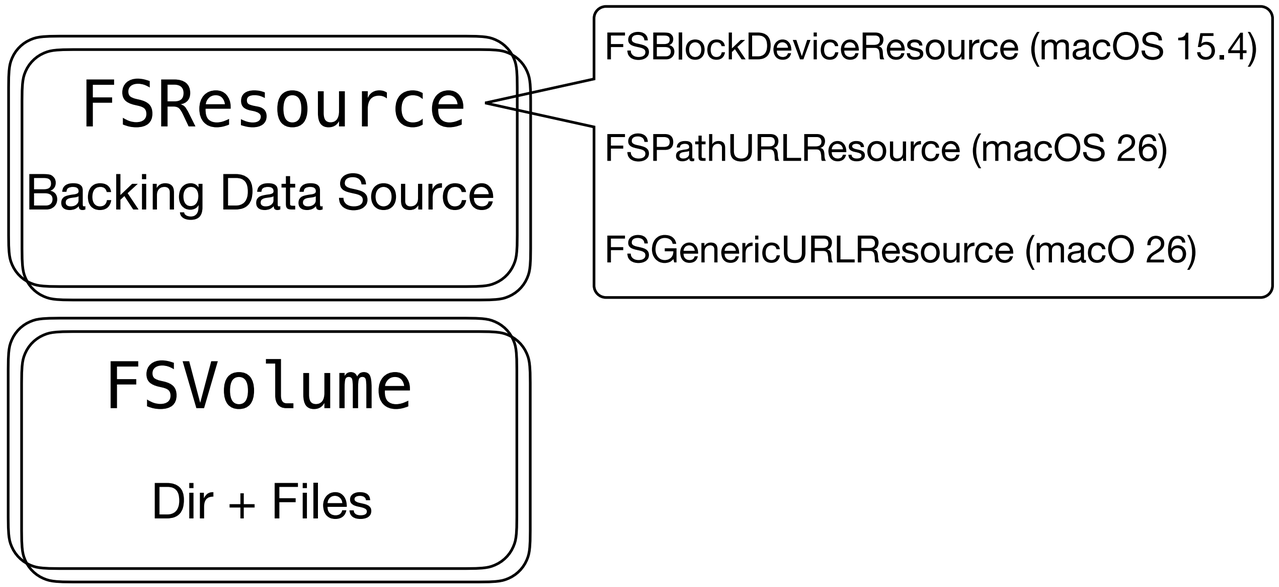


FSBlockDeviceResource (macOS 15.4)

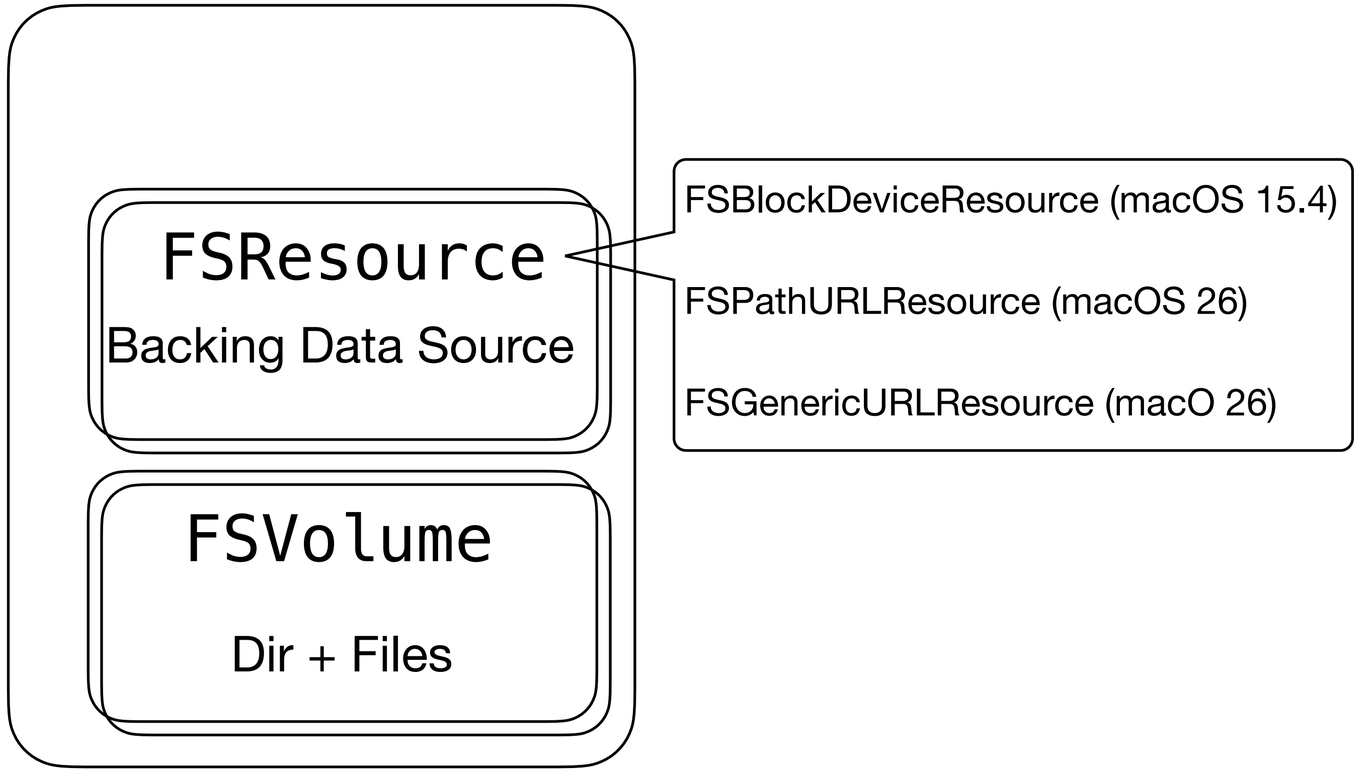
FSPathURLResource (macOS 26)

FSGenericURLResource (macO 26)

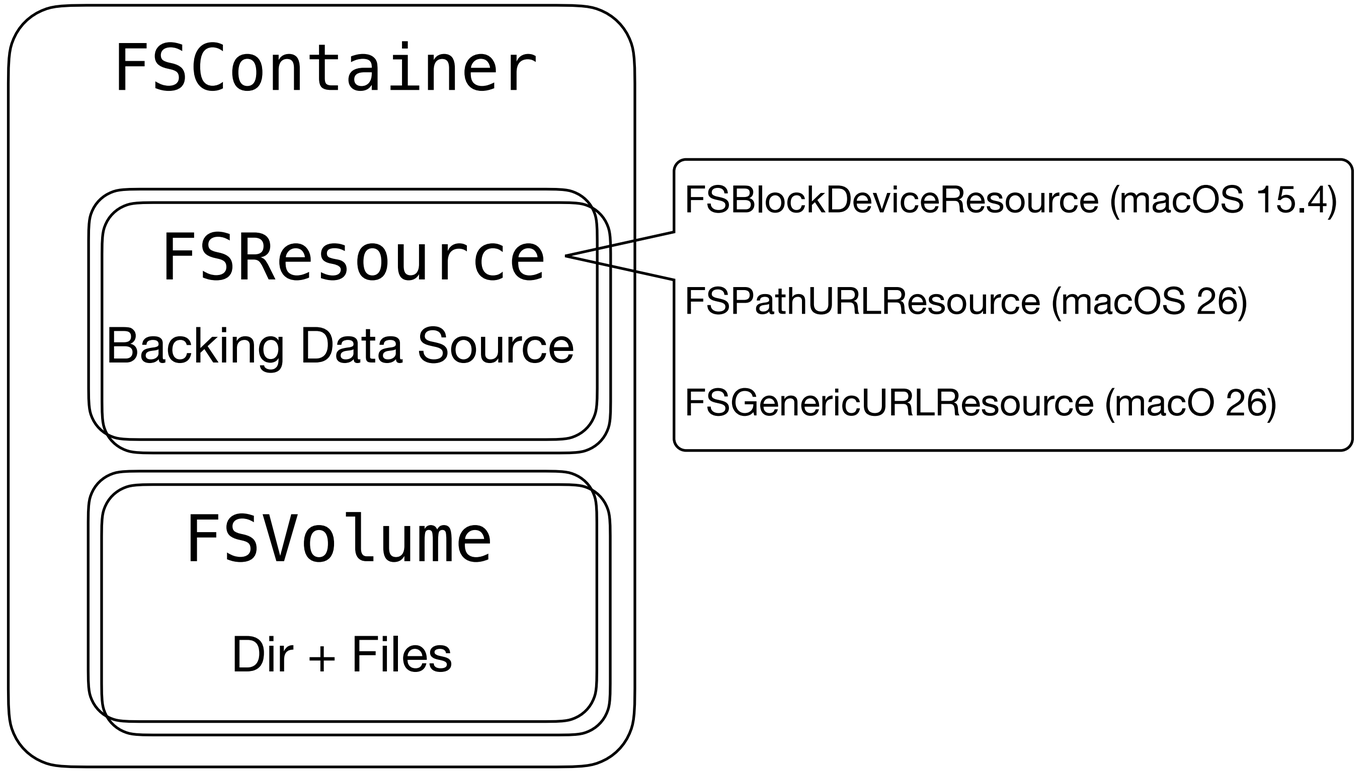
Three Major Concepts



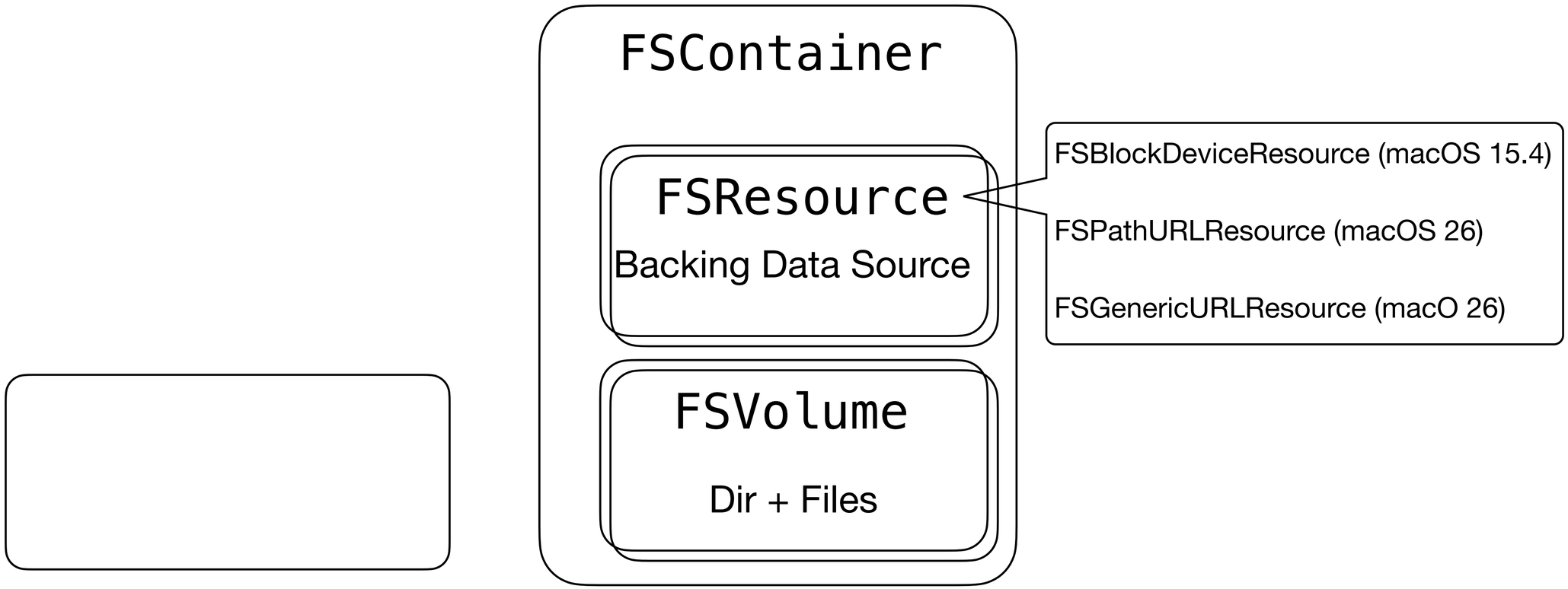
Three Major Concepts



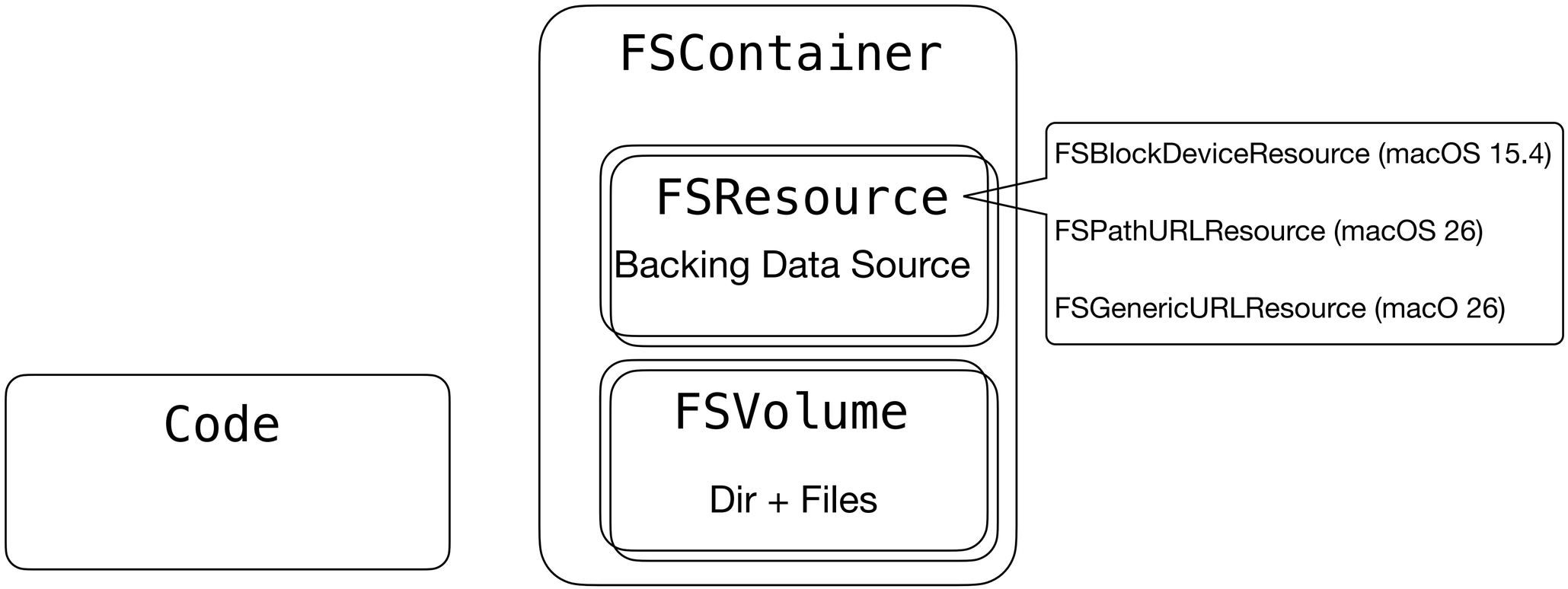
Three Major Concepts



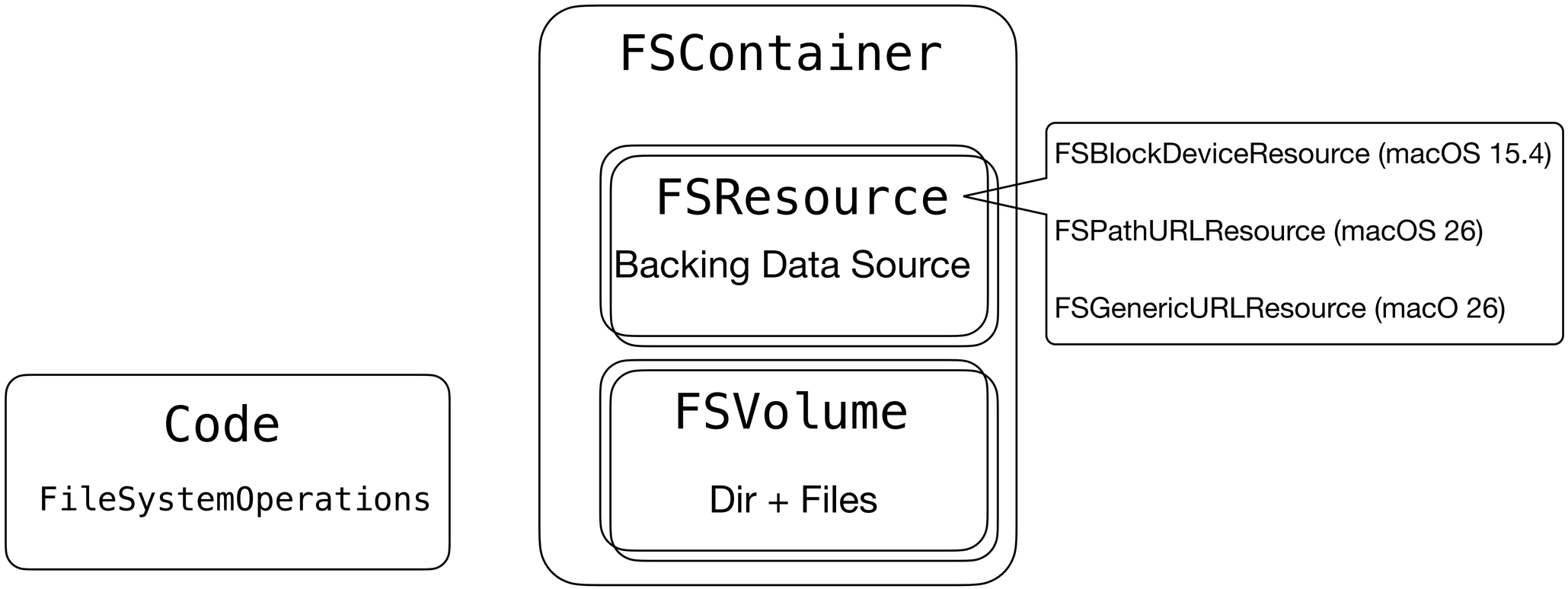
Three Major Concepts



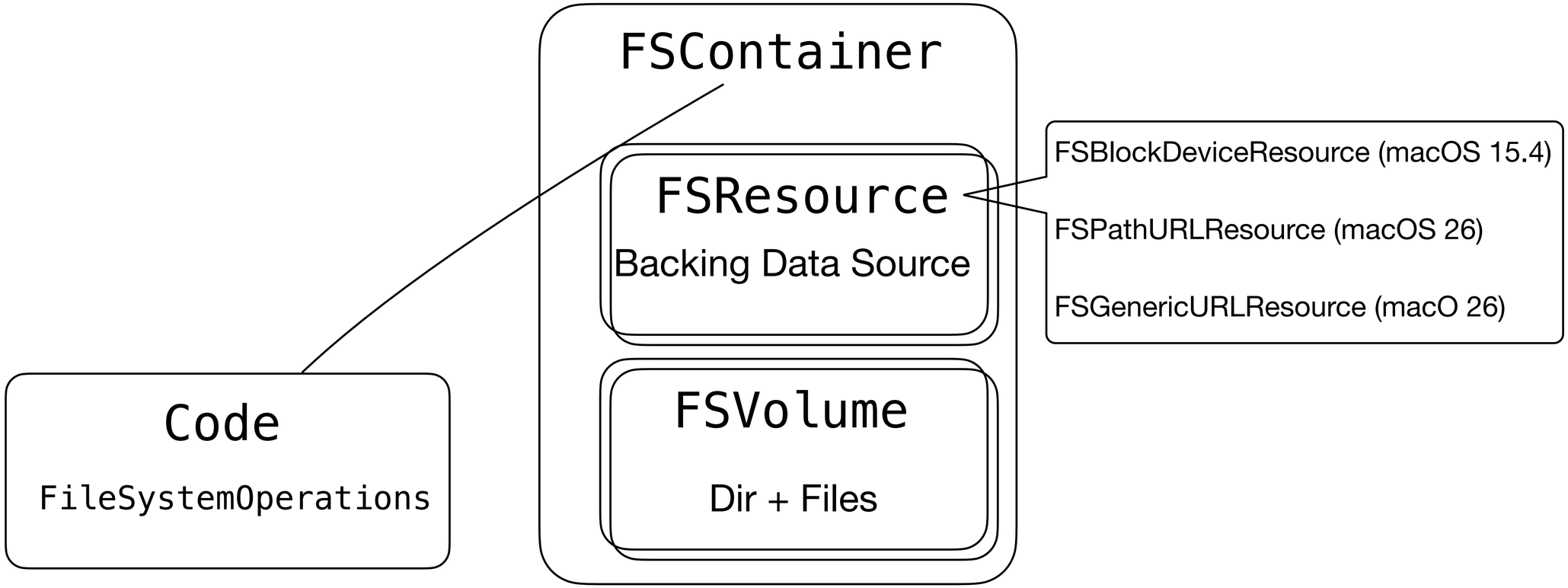
Three Major Concepts



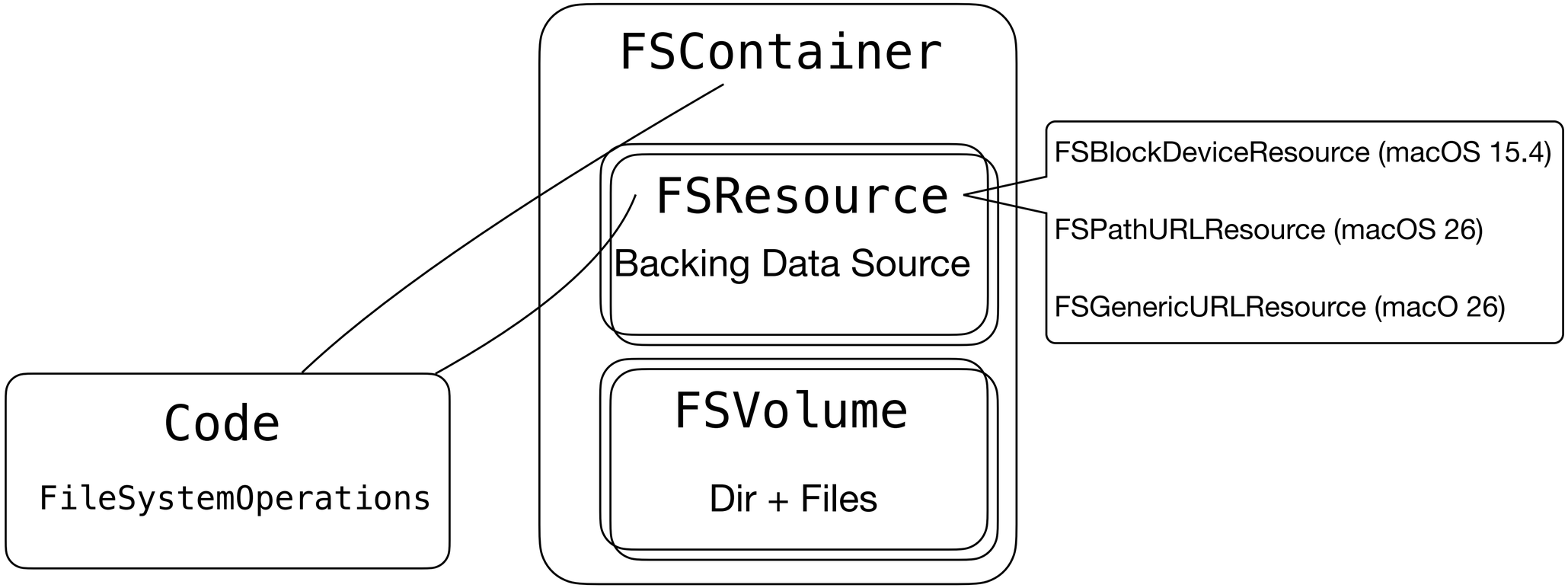
Three Major Concepts



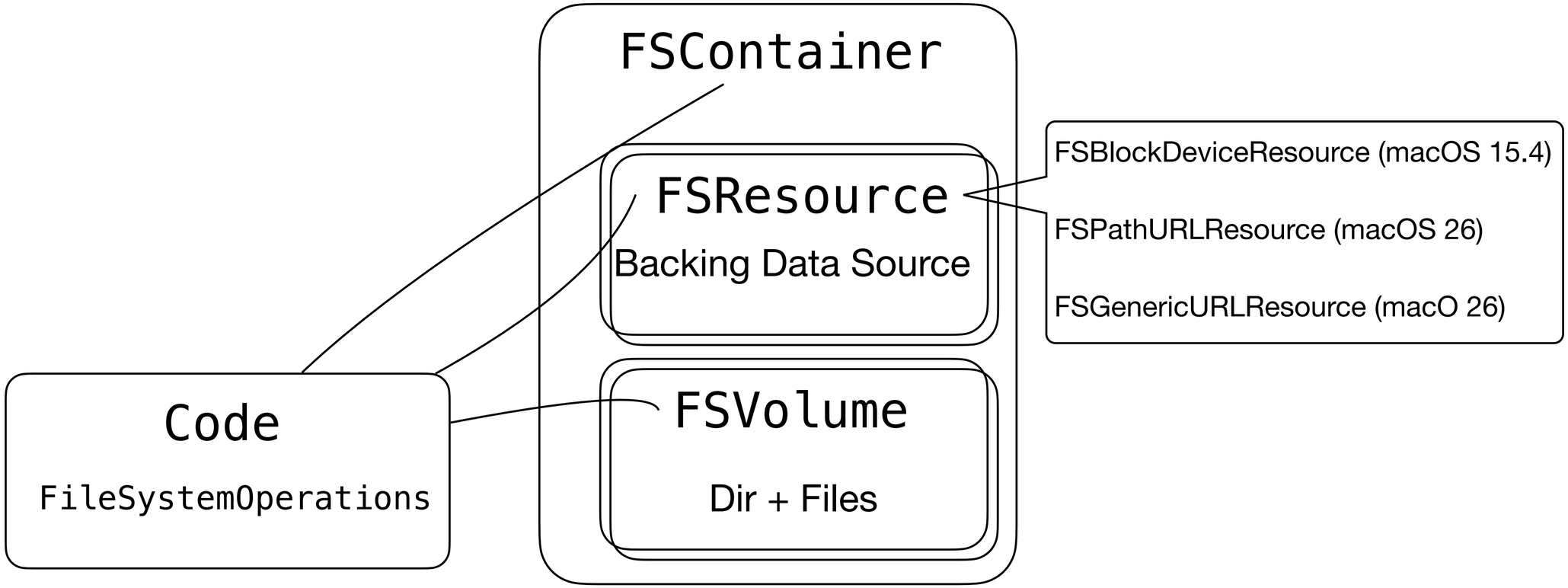
Three Major Concepts



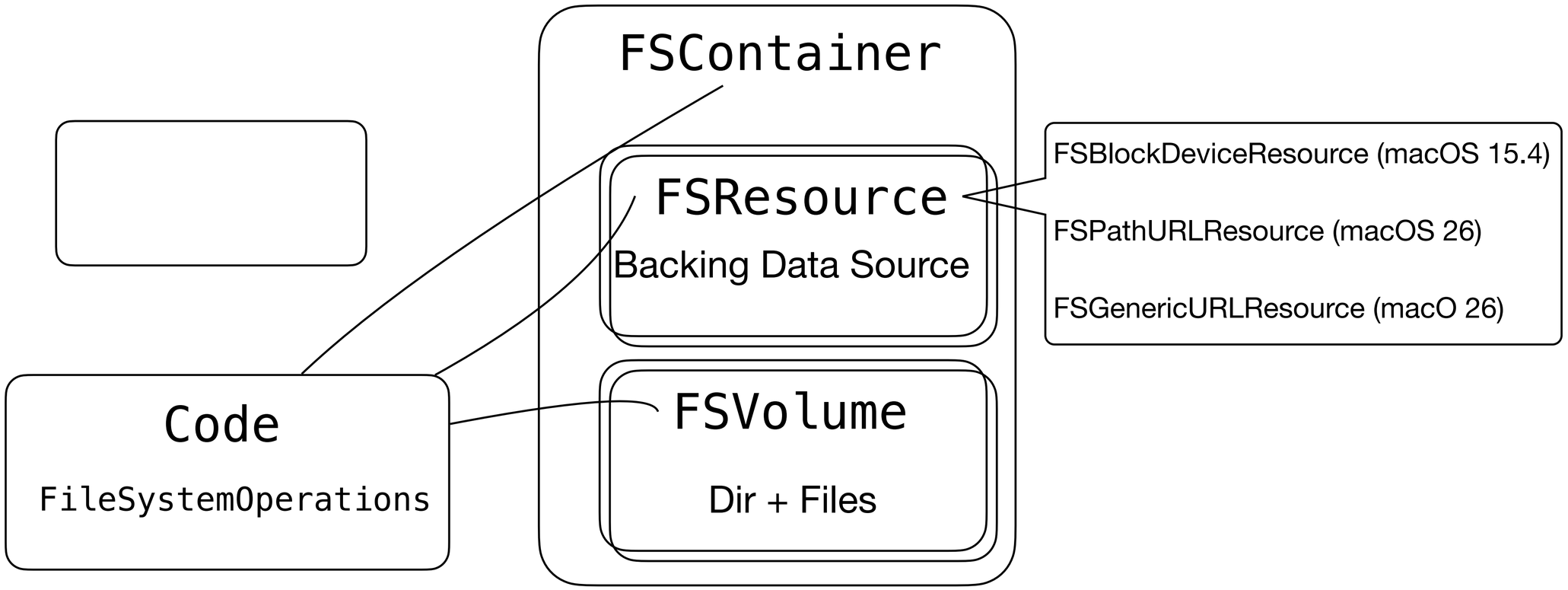
Three Major Concepts



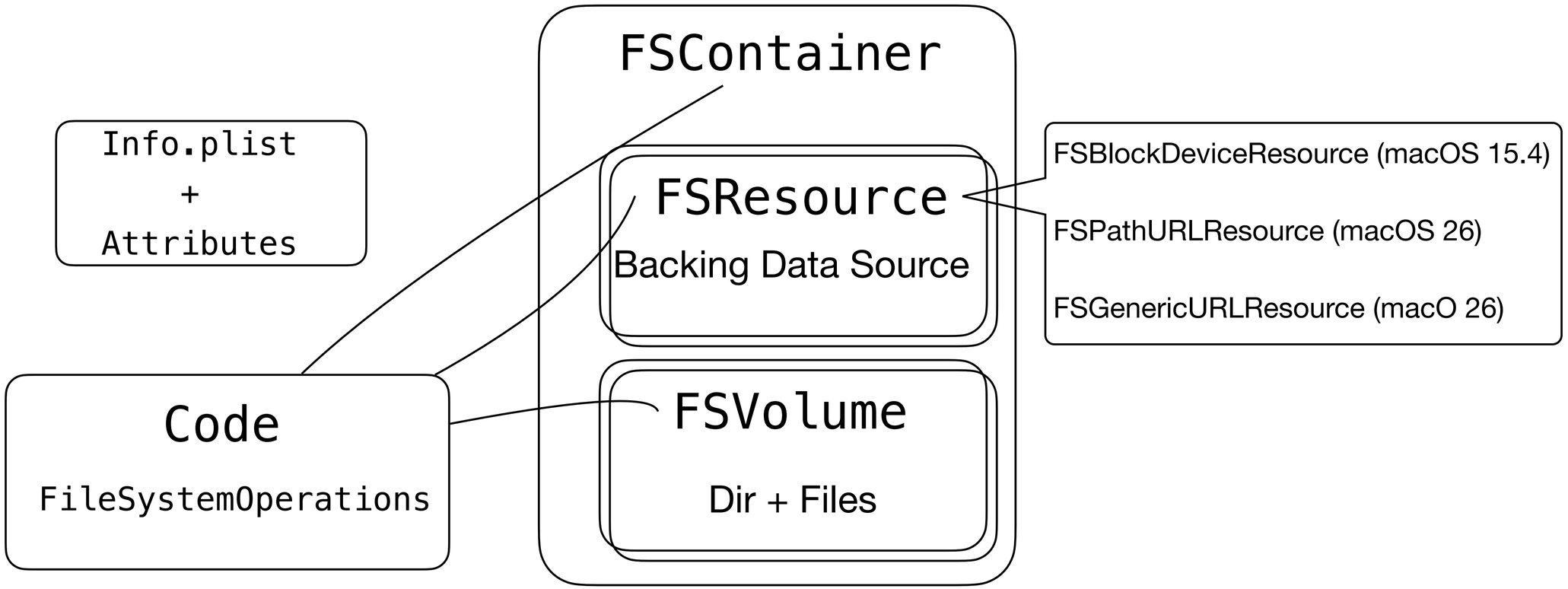
Three Major Concepts



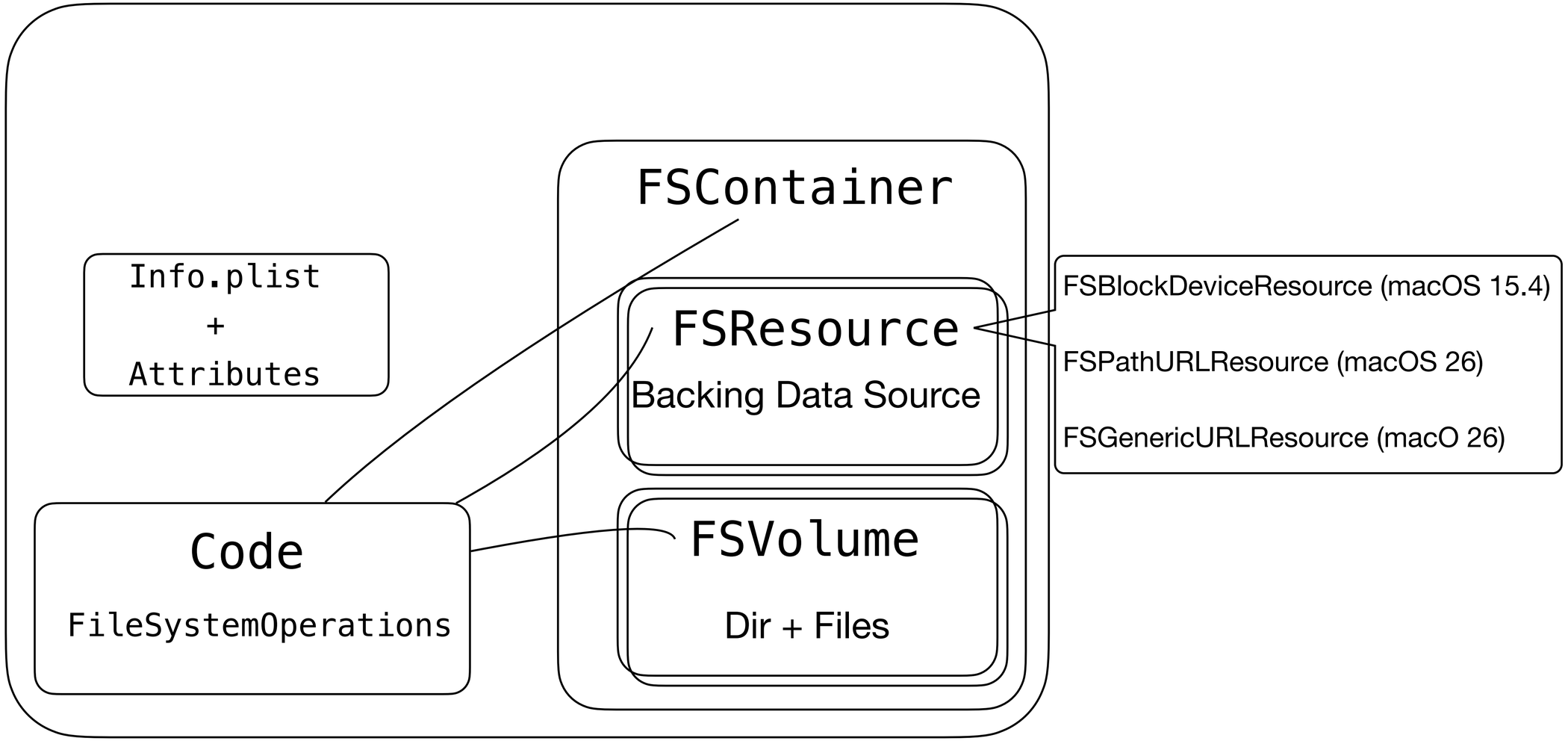
Three Major Concepts



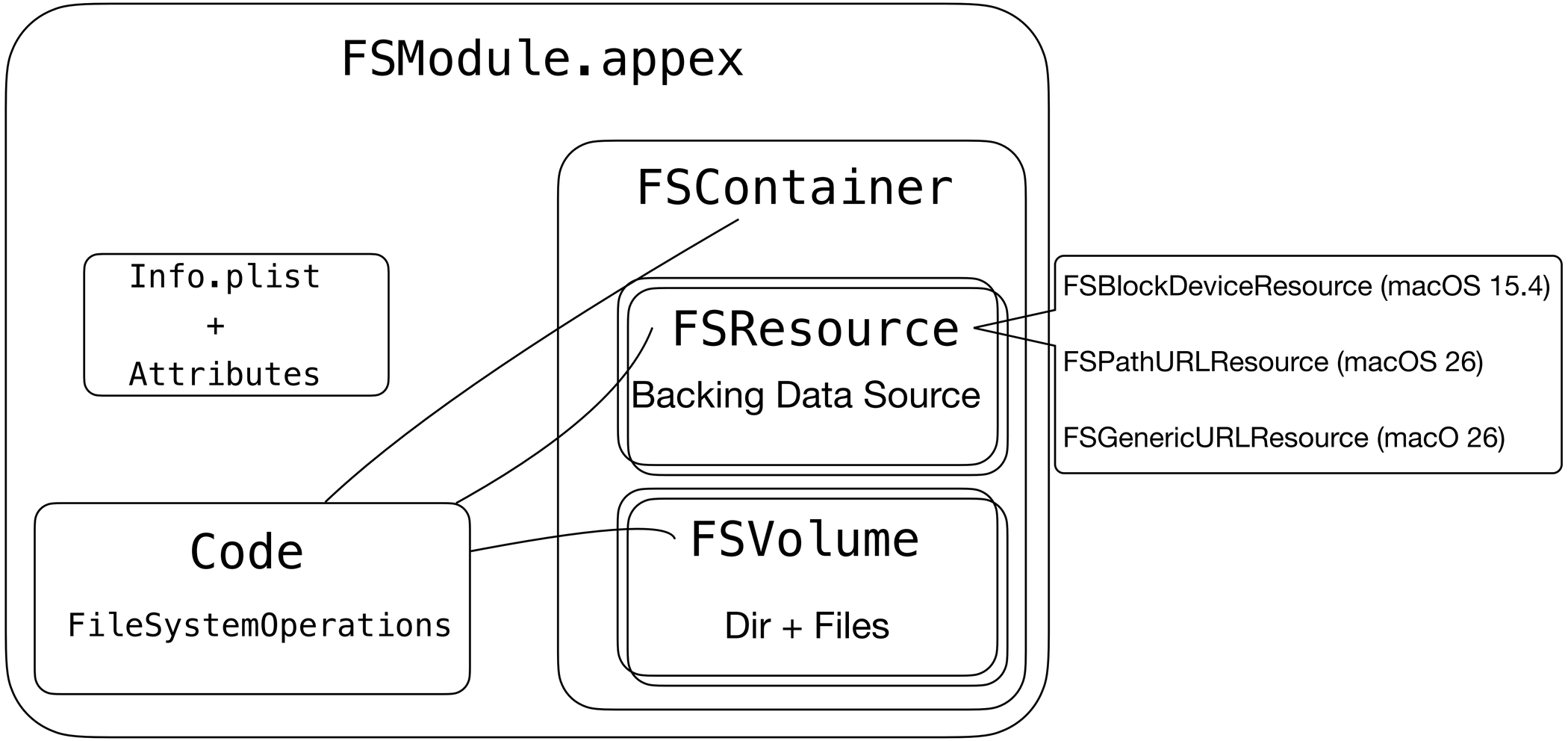
Three Major Concepts



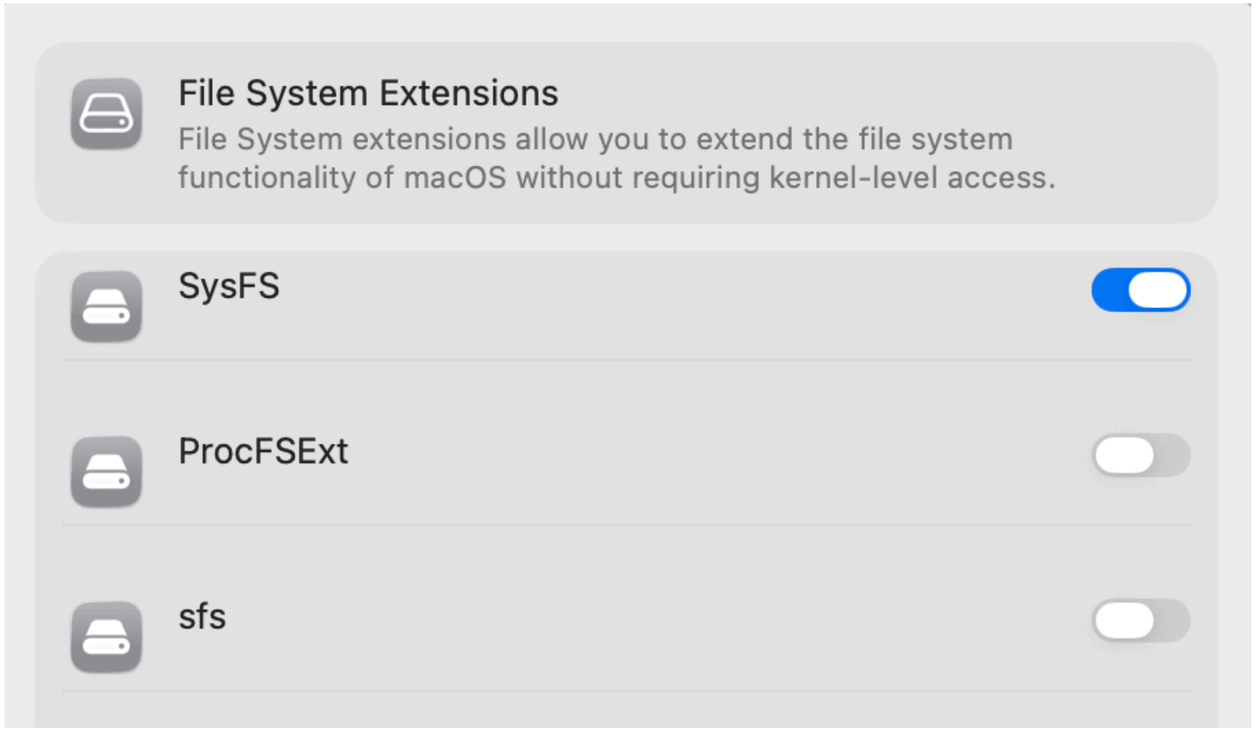
Three Major Concepts



Three Major Concepts



User Interaction Required* to Enable our FSModule





Info.plist

Info.plist

```
26  <key>FSPersonalities</key>
27  <dict>
28    <key>SysFSPersonality</key>
29    <dict>
30    <key>FSName</key>
31      <string>sysfs</string>
32      <key>FSfileObjectsAreCaseSensitive</key>
33      <false/>
34    </dict>
35  </dict>
```

Info.plist

```
26  <key>FSPersonalities</key>
27  <dict>
28    <key>SysFSPersonality</key>
29    <dict>
30    <key>FSName</key>
31      <string>sysfs</string>
32      <key>FSfileObjectsAreCaseSensitive</key>
33      <false/>
34    </dict>
35  </dict>
```

```
38   <key>FSShortName</key>
39   <string>sysfs</string>
40   <key>FSSupportsBlockResources</key>
41   <false/>
42   <key>FSSupportsGenericURLResources</key>
43   <false/>
44   <key>FSSupportsPathURLs</key>
45   <true/>
46   <key>FSSupportsServerURLs</key>
47   <false/>
```

SysFS (1)

```
final class SysFS: FSUnaryFileSystem, FSUnaryFileSystemOperations {

    private let logger = Logger(subsystem: "SysFS", category: "SysFS")

    func probeResource(
        resource: FSResource,
        replyHandler: @escaping (FSProbeResult?, (any Error)?) -> Void
    ) {
        guard let resource = resource as? FSPathURLResource else {
            exit(EXIT_FAILURE)
        }

        let fileName = resource.url.deletingPathExtension().lastPathComponent
        replyHandler(
            FSProbeResult.usable(
                name: fileName,
                containerID: FSContainerIdentifier(uuid: UUID())
            ),
            nil
        )
    }
```

SysFS (2)

```
func loadResource(
    resource: FSResource,
    options: FSTaskOptions,
    replyHandler: @escaping (FSVolume?, (any Error)?) -> Void
) {
    containerStatus = .ready
    let volume = SysFS(resource: resource)
    replyHandler(
        volume,
        nil
    )
}
func unloadResource(
    resource: FSResource,
    options: FSTaskOptions,
    replyHandler reply: @escaping ((any Error)?) -> Void
) {
    logger.debug("unloadResource: \(resource, privacy: .public)")
    reply(nil)
}
```

SysFSVolume (1)

SysFSVolume (1)

```
var supportedCapabilities: FSVolume.SupportedCapabilities {
  let capabilities = FSVolume.SupportedCapabilities()
  capabilities.supportsJournal = false
  capabilities.doesNotSupportRootTimes = true
  capabilities.supportsHardLinks = false
  capabilities.supportsSymbolicLinks = false
  capabilities.supportsPersistentObjectIDs = true
  capabilities.supportsHiddenFiles = false
  capabilities.supports64BitObjectIDs = true
  capabilities.caseFormat = .insensitiveCasePreserving
  return capabilities
}
```

SysFSVolume (1)

```
var supportedCapabilities: FSVolume.SupportedCapabilities {
  let capabilities = FSVolume.SupportedCapabilities()
  capabilities.supportsJournal = false
  capabilities.doesNotSupportRootTimes = true
  capabilities.supportsHardLinks = false
  capabilities.supportsSymbolicLinks = false
  capabilities.supportsPersistentObjectIDs = true
  capabilities.supportsHiddenFiles = false
  capabilities.supports64BitObjectIDs = true
  capabilities.caseFormat = .insensitiveCasePreserving
  return capabilities
}
```

```
func reclaimItem(_ item: FSItem) async throws {
        throw fs_errorForPOSIXError(POSIXError.EIO.rawValue)
    }

    func readSymbolicLink(
        _ item: FSItem
    ) async throws -> FSFileName {
        throw fs_errorForPOSIXError(POSIXError.EIO.rawValue)
    }

    func createItem(named name: FSFileName, type: FSItem.ItemType,
        inDirectory directory: FSItem,
        attributes newAttributes: FSItem.SetAttributesRequest
    ) async throws -> (FSItem, FSFileName) {
        throw fs_errorForPOSIXError(POSIXError.EIO.rawValue)
    }
```

SysFSVolume (2)

```
func read(
        from item: FSItem,
        at offset: off_t,
        length: Int,
        into buffer: FSMutableFileDataBuffer
    ) async throws -> Int {
        var bytes = 0
        let name = item.name.string
        var data: Data
        if name == "hello_world" {
            data = getData("hello_world")
        } else if name == "time" {
            data = getData("time")
        } else if name == "version" {
            data = getData("version")
        } else if name == "tripwire" {
            data = getData("tripwire")
        }

        if offset < item.attributes.size {
            return 0
        }
```

```
let maxRead = min(length, Int(totalSize) -
Int(Offset)
bytes = data.withUnsafeBytes { (src:
UnsafeRawBufferPointer) in
  buffer.withUnsafeMutableBytes { (dst:
UnsafeMutableRawBufferPointer) in
    let srcPtr = src.baseAddress! +
Int(Offset)
    let dstPtr = dst.baseAddress!
    memcpy(dstPtr, srcPtr, maxRead)
    return maxRead
  }
}
return bytes
```

Mounting

```
→ pluginkit -a SysFS.appex

→ mount -F -t sysfs dummy_random_data_file_for_sysfs.bin vol
```

Mounts

```
osquery> select device, path, type from mounts where type != 'apfs';
+-----------------------------------------------------------+---------------------------+--------+
| device                                                    | path                      | type   |
+-----------------------------------------------------------+---------------------------+--------+
| devfs                                                     | /dev                      | devfs  |
| map auto_home                                             | /System/Volumes/Data/home | autofs |
| /dev/disk4s1                                              | /Volumes/Kernel Debug Kit | hfs    |
| /dev/disk13s1                                             | /Volumes/Additional Tools | hfs    |
| dummy_random_data_file_for_sysfs.bin -- file:///private/tmp/fsk/ | /private/tmp/fsk/vol      | sysfs  |
+-----------------------------------------------------------+---------------------------+--------+
```

ES Mount Event


```
{
    "event":
    {
        "mount":
        {
            "statfs":
            {
                "f_blocks": 0,
                "f_fsid":
                [
                    805306383,
                    24
                ],
                "f_ffree": 0,
                "f_bavail": 0,
                "f_flags": 270536704,
                "f_bsize": 1048576,
                "f_fssubtype": 0,
                "f_owner": 501,
                "f_fstypename": "sysfs",
                "f_files": 0,
                "f_mntfromname": "dummy_random_data_file_for_sysfs.bin -- file:///private/tmp/fsk/
                "f_mntonname": "/private/tmp/fsk/vol",
                "f_type": 24,
                "f_iosize": 1048576,
                "f_flags_ext": 2,
                "f_bfree": 0
            },
            "disposition": 5
        }
    },
    "thread":
    {
        "thread_id": 8306495
    },
    "time": "2025-10-15T11:16:55.025834623Z",
    "global_seq_num": 339,
    "schema_version": 1,
    "seq_num": 0,
    "version": 10,
    "global_seq_num": 339,
    "schema_version": 1,
    "seq_num": 0,
    "version": 10,
    "process":
    {
        "cs_validation_category": 1,
        "codesigning_flags": 637623057,
        "is_es_client": false,
        "tty": null,
        "start_time": "2025-10-05T08:43:21.731362Z",
        "signing_id": "com.apple.fskitd",
        "team_id": null,
        "is_platform_binary": true,
        "ppid": 1,
        "executable":
        {
            "path": "/usr/libexec/fskitd",
            "path_truncated": false,
            "stat":
            {
                "st_rdev": 0,
                "st_atimespec": "2025-09-09T07:15:49.0000000
                "st_blksize": 4096,
                "st_gid": 0,
                "st_ino": 1152921500312557889,
                "st_size": 913552,
                "st_ctimespec": "2025-09-09T07:15:49.0000000
                "st_uid": 0,
                "st_nlink": 1,
                "st_mtimespec": "2025-09-09T07:15:49.0000000
                "st_dev": 16777233,
                "st_blocks": 688,
                "st_mode": 33261,
                "st_flags": 524320,
                "st_birthtimespec": "2025-09-09T07:15:49.000
                "st_gen": 0
            }
        },
        "original_ppid": 1,
        "cdhash": "E95E7306AE9C520C875624B626666C68320ECBFF"
        "group_id": 40361,
        "audit_token":
        {
            "pidversion": 310749,
            "euid": 0,
            "asid": 100021,
            "auid": 4294967295,
            "rgid": 0,
            "ruid": 0,
            "egid": 0,
            "pid": 40361
        },
        "session_id": 40361,
```

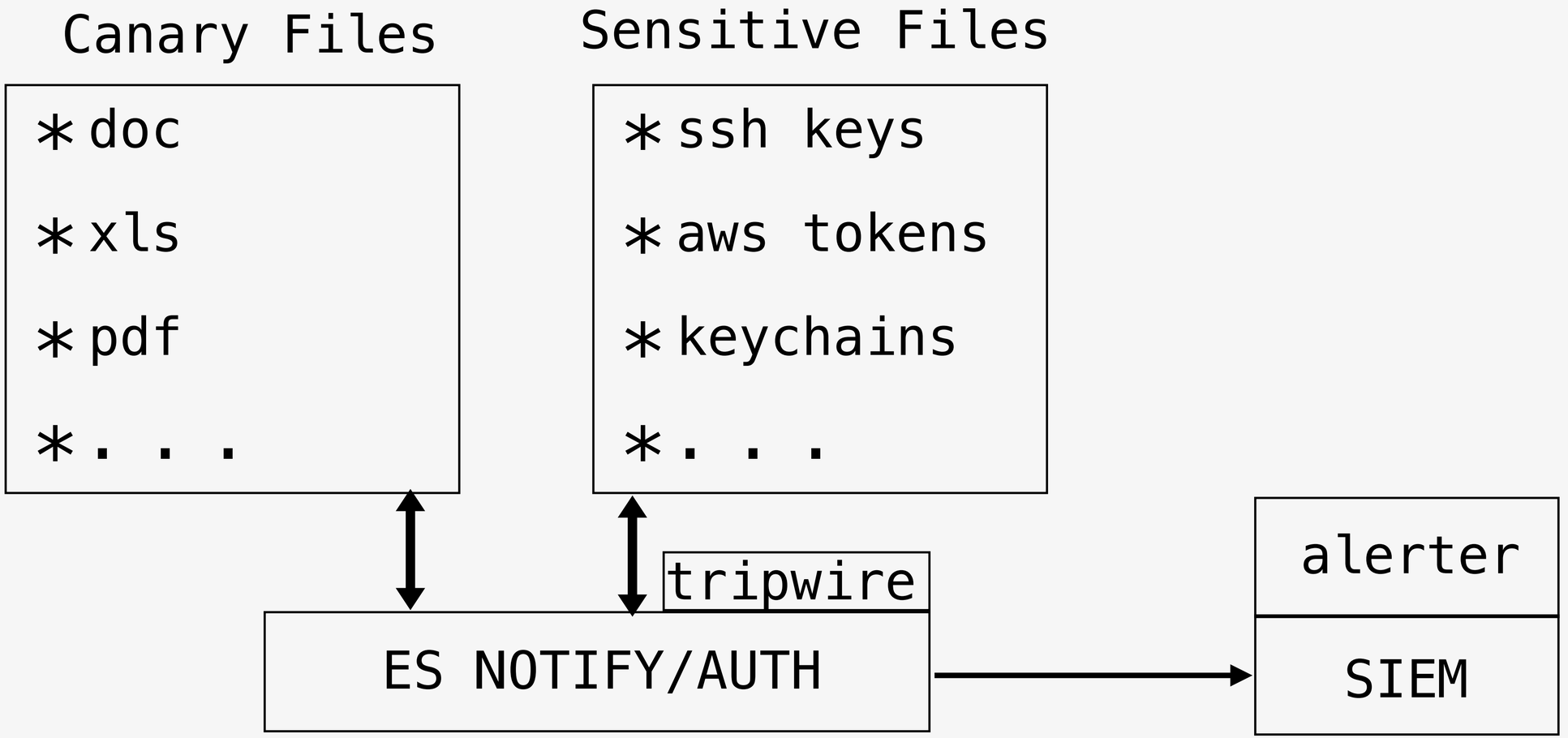
ES Open Event


```
{
    "event":
    {
        "open":
        {
            "file":
            {
                "path": "/private/tmp/fsk/vol/tripwire",
                "path_truncated": false,
                "stat":
                {
                    "st_rdev": 0,
                    "st_atimespec": "2025-10-15T11:16:54.946095000Z",
                    "st_blksize": 1048576,
                    "st_gid": 20,
                    "st_ino": 5070880731945,
                    "st_size": 16589496,
                    "st_ctimespec": "2025-10-15T11:16:54.946095000Z",
                    "st_uid": 501,
                    "st_nlink": 0,
                    "st_mtimespec": "2025-10-15T11:16:54.946095000Z",
                    "st_dev": 805306383,
                    "st_blocks": 0,
                    "st_mode": 33216,
                    "st_flags": 0,
                    "st_birthtimespec": "2025-10-15T11:16:54.946095000Z",
                    "st_gen": 0
                }
            },
            "fflag": 16777221
        }
    },
    "thread":
    {
        "thread_id": 8306637
    },
    "time": "2025-10-15T11:17:03.364282919Z",
    "global_seq_num": 1336,
    "schema_version": 1,
    "seq_num": 985,
    "version": 10,
    "time": "2025-10-15T11:17:03.364282919Z",
    "global_seq_num": 1336,
    "schema_version": 1,
    "seq_num": 985,
    "version": 10,
    "process":
    {
        "cs_validation_category": 1,
        "codesigning_flags": 637606673,
        "is_es_client": false,
        "tty":
        {
            "path": "/dev/ttys020",
            "path_truncated": false,
            "stat":
            {
                "st_rdev": 268435476,
                "st_atimespec": "2025-10-15T11:17:03.358388000Z",
                "st_blksize": 65536,
                "st_gid": 4,
                "st_ino": 1259,
                "st_size": 0,
                "st_ctimespec": "2025-10-15T11:17:03.359483000Z",
                "st_uid": 501,
                "st_nlink": 1,
                "st_mtimespec": "2025-10-15T11:17:03.359483000Z",
                "st_dev": -685418376,
                "st_blocks": 0,
                "st_mode": 8592,
                "st_flags": 0,
                "st_birthtimespec": "1970-01-01T00:00:00.000000000Z",
                "st_gen": 0
            }
        },
        "start_time": "2025-10-15T11:17:03.359131Z",
        "signing_id": "com.apple.file",
        "team_id": null,
        "is_platform_binary": true,
        "ppid": 28986,
        "executable":
        {
            "path": "/usr/bin/file",
            "path_truncated": false,
            "stat":
```

FSKit

* Easily write custom filesystems
* You control not only read/write, but also other ops
* symlink behavior, xattr get/set
* Opens up a lot of powerful detection use cases
* Use AI to populate canary filesystem
* Generic ransomware detection v2.0

Use Case: Canary Files and Honeypot



Use Case: Canary Files and Honeypot

Canary Files

Sensitive Files

* do
* xl
* pd
* .



alerter

SIEM

Use Case: Canary Files and Honeypot

Canary Files

Sensitive Files

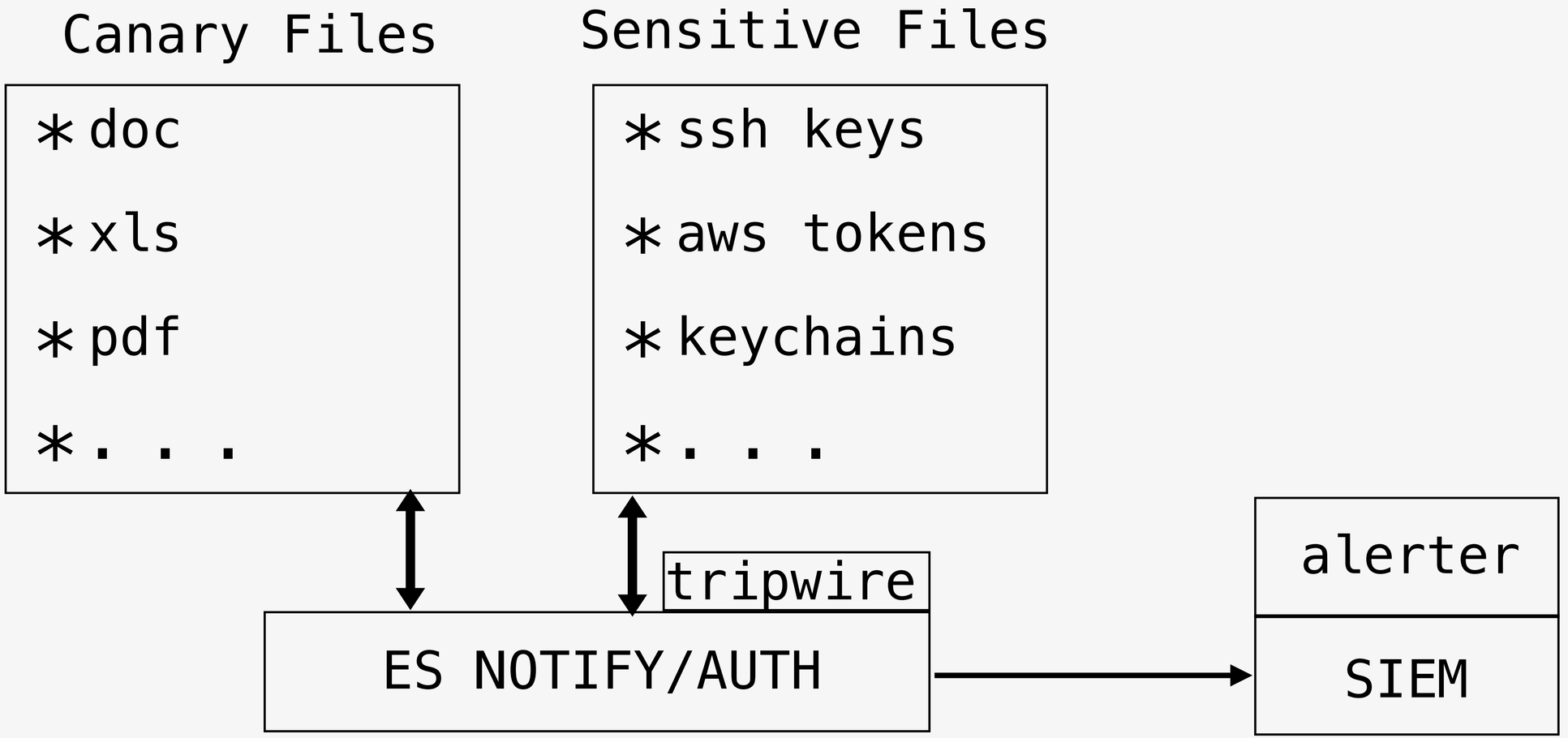
* do
* xl
* pd
* .



alerter

SIEM

Use Case: Canary Files and Honeypot



- Not just theoretical

- Successfully deployed

BTW, we just had a pentest and it lit up our file access monitoring like a christmas tree

🎄 1

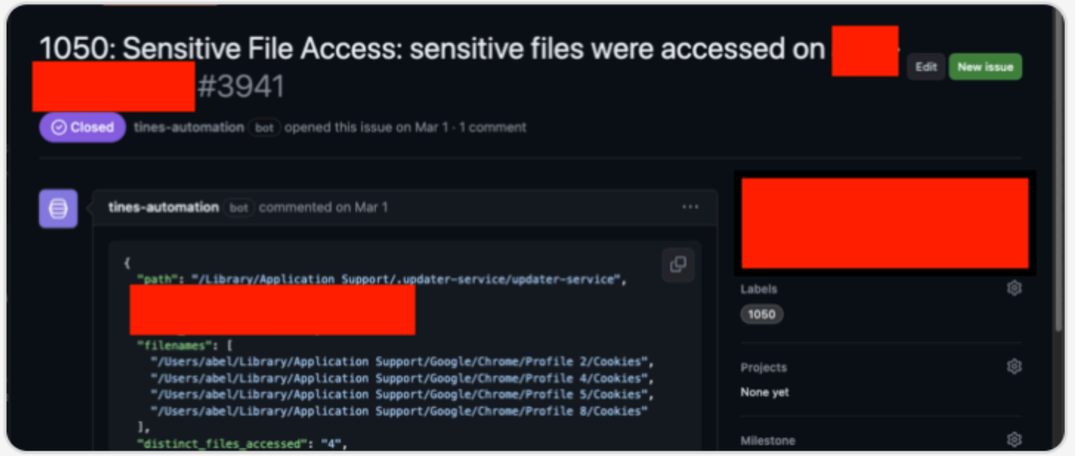
Sharvil 4:27 AM

[redacted] 😊

Glad to hear it's having some success!

[redacted] 4:28 AM

image.png



Was a big 🥱 🥱 🥱 🥱 moment lol

Sharvil 4:29 AM

Nice! That tines automation is the cherry on top

osquery: file access monitoring

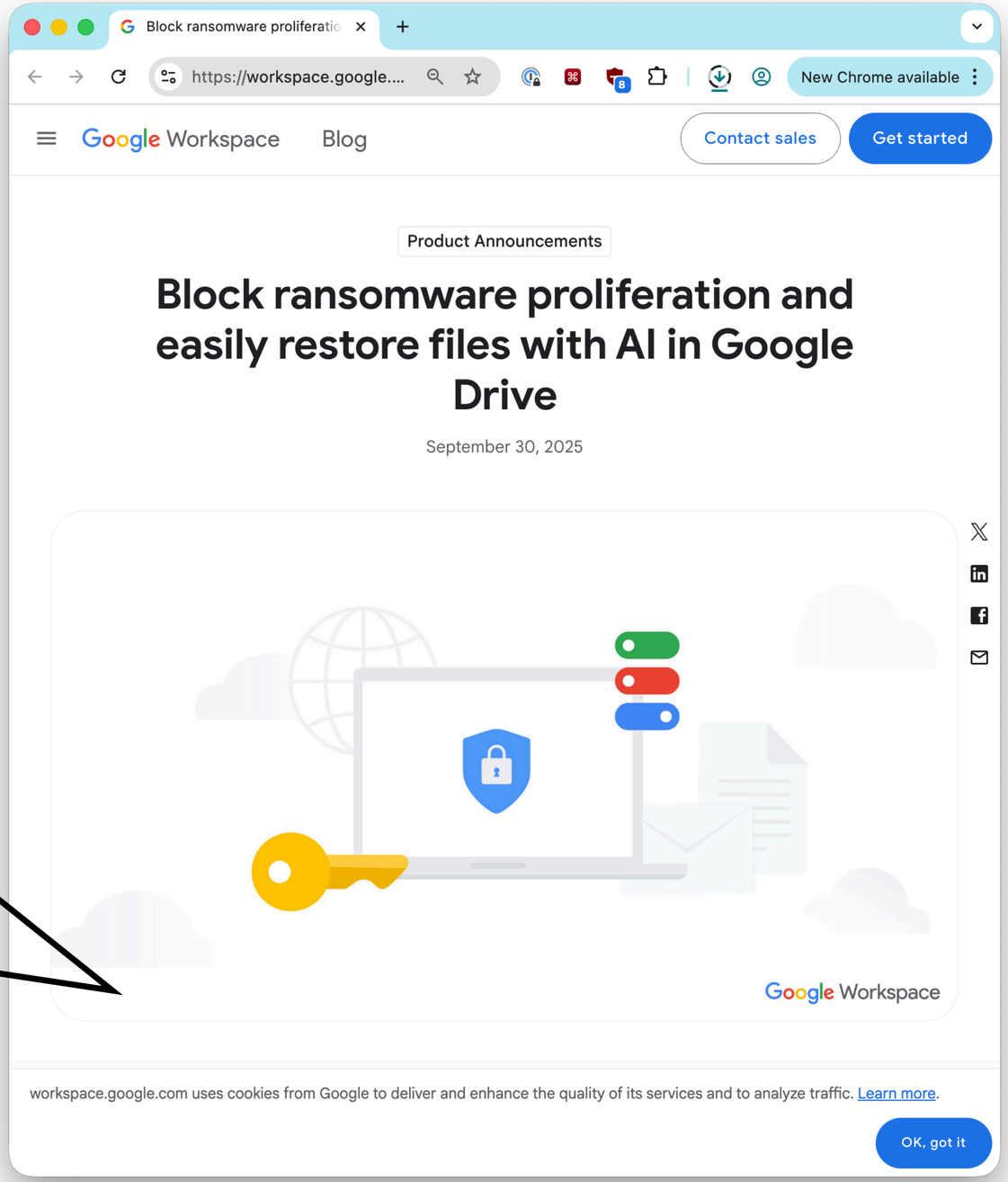
```
osquery> select pid, parent, path, filename, event_type, time from es_process_file_events;
+-------+--------+-----------------------------------------------------------------------------------------+-----------------
| pid   | parent | path                                                                                    | filename
+-------+--------+-----------------------------------------------------------------------------------------+-----------------
| 73764 | 35325  | /opt/homebrew/Cellar/bat/0.25.0_1/bin/bat                                               | /Users/sharvil/.s
| 395   | 1      | /System/Library/PrivateFrameworks/ApplePushService.framework/apsd                       | /Library/Keychain
| 73787 | 1      | /System/Library/CoreServices/Applications/Keychain Access.app/Contents/MacOS/Keychain Access | /Library/Keychain
| 395   | 1      | /System/Library/PrivateFrameworks/ApplePushService.framework/apsd                       | /Library/Keychain
| 395   | 1      | /System/Library/PrivateFrameworks/ApplePushService.framework/apsd                       | /Library/Keychain
| 395   | 1      | /System/Library/PrivateFrameworks/ApplePushService.framework/apsd                       | /Library/Keychain
| 395   | 1      | /System/Library/PrivateFrameworks/ApplePushService.framework/apsd                       | /Library/Keychain
+-------+--------+-----------------------------------------------------------------------------------------+-----------------
```

```
--------+-----------------------------------+------------+------------+
        | filename                          | event_type | time       |
--------+-----------------------------------+------------+------------+
        | /Users/sharvil/.ssh/id_ed25519.pub | open       | 1760691579 |
        | /Library/Keychains/apsd.keychain  | open       | 1760691587 |
 Access | /Library/Keychains/System.keychain | open       | 1760691599 |
        | /Library/Keychains/apsd.keychain  | open       | 1760691611 |
        | /Library/Keychains/apsd.keychain  | open       | 1760691814 |
        | /Library/Keychains/apsd.keychain  | open       | 1760691854 |
        | /Library/Keychains/apsd.keychain  | open       | 1760691878 |
--------+-----------------------------------+------------+------------+
```

Google just announced ransomware

Protection on Google Drive 15 days back

Our AI-powered detection in Drive for desktop identifies the core signature of a ransomware attack — an attempt to encrypt or corrupt files en masse — and rapidly intervenes to put a protective bubble around a user's files by stopping file syncing to the cloud before the ransomware can spread.



* takeaways — es muting + file access monitoring + fskit
* Questions?

* Thank you!