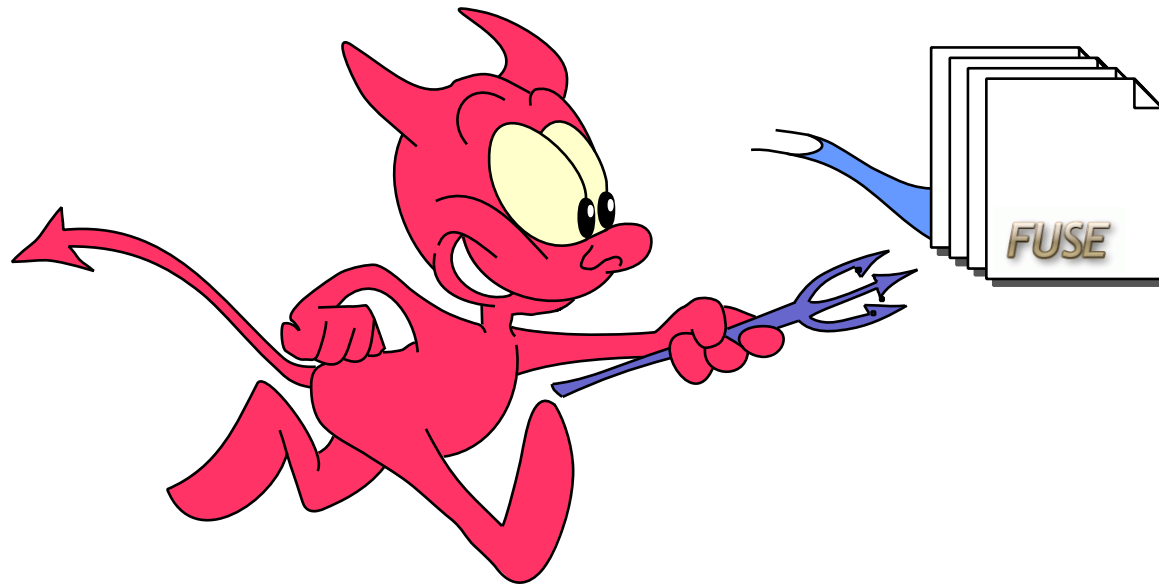


FUSE and beyond: bridging filesystems



Emmanuel Dreyfus, september 2014

Goals

- Distributed file system beyond NFS
 - High availability, without SPOF
 - Storage clustering
 - Elasticity
 - POSIX semantics, with locking
 - Performances
 - Secure communications (crypto)



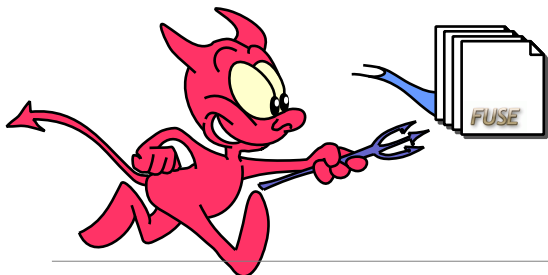
File systems

- Used to be implemented in kernel
 - Not easily portable
 - Long development cycles
- User space file systems
 - FS are Unix processes
 - Kernel is FS client
 - Microkernel-like approach



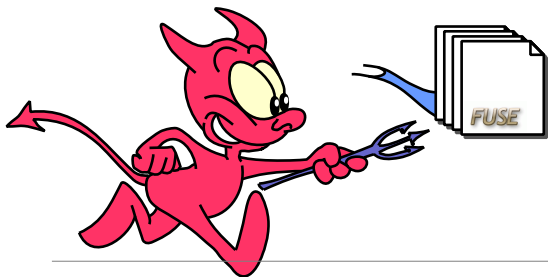
Many solutions

- Lustre Hadoop GlusterFS XtremFS Ceph...
- Requirements to mount a FS
 - Kernel module (OS-dependent)
 - FUSE
 - Path of least resistance
 - Many FUSE FS available
- Goal : GlusterFS through FUSE

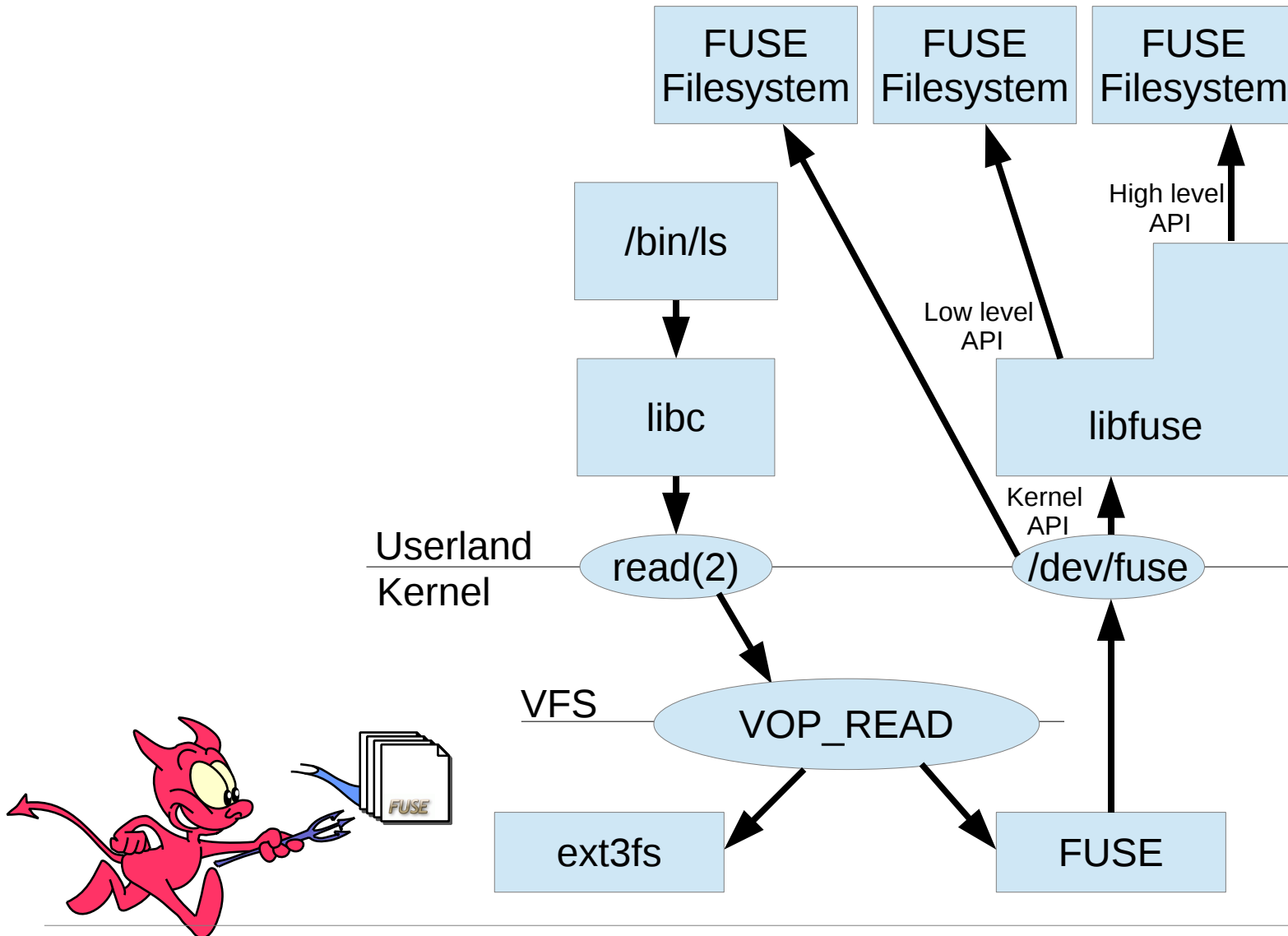


Implementations : FUSE (1)

- Used to be a Linux project
- *de facto* standard : FreeBSD, MacOS X
- Kernel is FUSE client
- Message-passing interface through /dev/fuse
- User level library : libfuse
- Three (!) API for FUSE file systems



Implementations : FUSE (2)



About VFS

- Virtual File System, introduced by Sun in 1985
- Multiplexer to multiple FS (UFS and NFS)
- In-kernel VFS interface for all FS
- Code below VFS interface is FS-dependent
- Objects : mounts, vnode
- Methods: lookup, open, read, write...

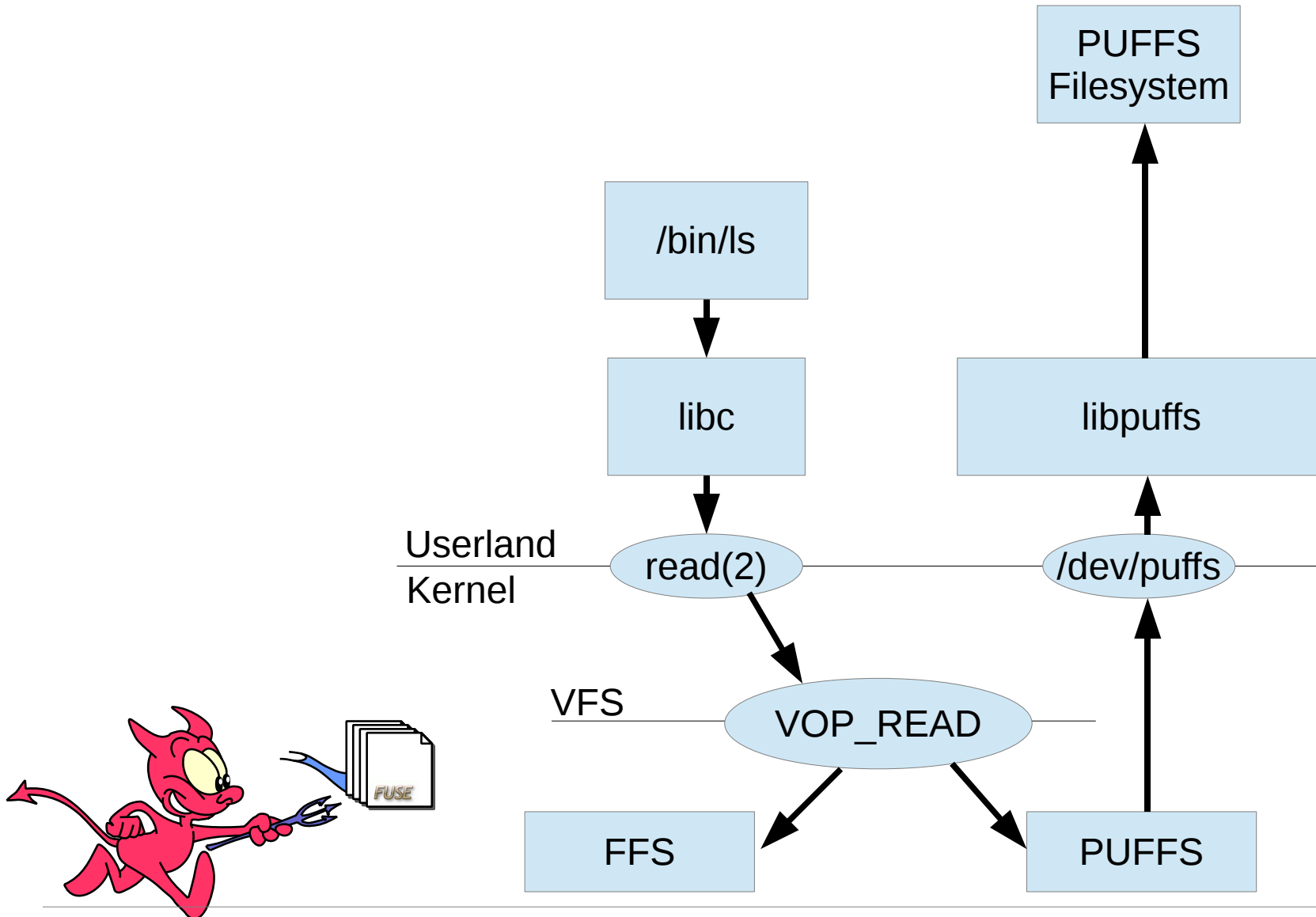


Implementations : PUFFS (1)

- User space file systems for NetBSD
- Similar to FUSE, but not compatible
- Started when FUSE was no obvious standard
- Native interface still has merit today
 - Better fit native VFS
 - No roadblocks to adding new features

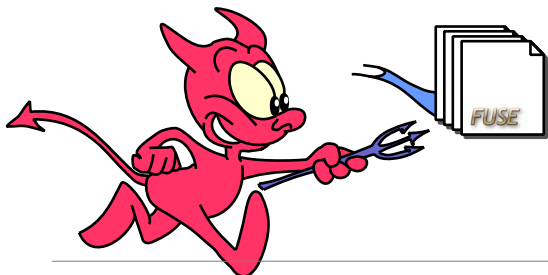


Implementation : PUFFS (2)



Implementation : PUFFS (3)

- PUFFS architecture looks a lot like FUSE...
 - /dev/puffs instead of /dev/fuse
 - libpuffs instead of libfuse
- ...but it is still different
 - Message-passing protocol is different
 - API is different (at least PUFFS has only one !)

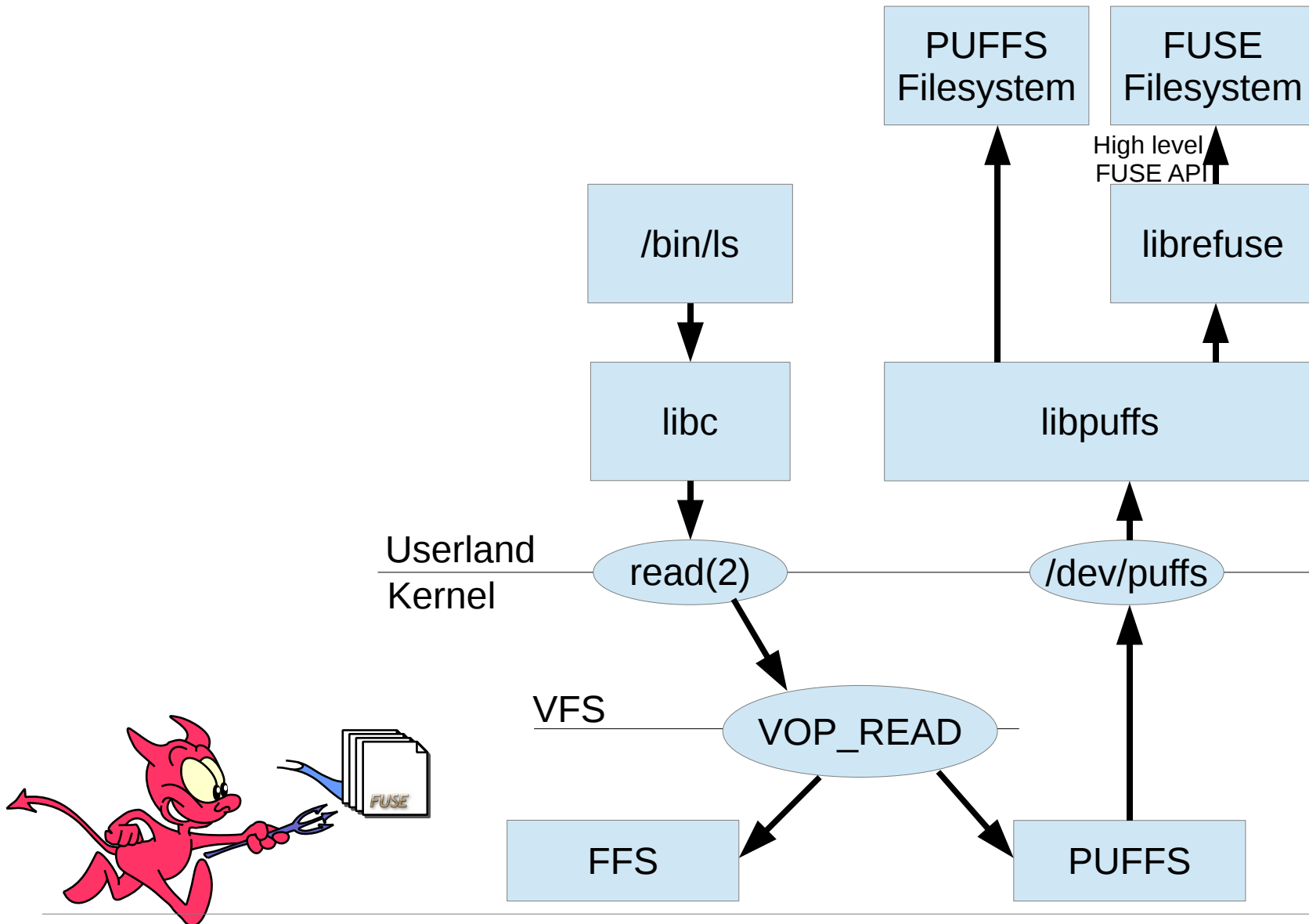


FUSE over PUFFS : REFUSE (1)

- PUFFS has merits (native, not constrained)
- FUSE, is desirable as a *de facto* standard
- FUSE over PUFFS implementation : REFUSE



FUSE over PUFFS : REFUSE (2)



FUSE over PUFFS : REFUSE (3)

- Limitation: we only support FUSE high level API
- No support for FS using low level API
- No support for FS bypassing libfuse
 - GlusterFS directly uses /dev/fuse
- REFUSE is not enough

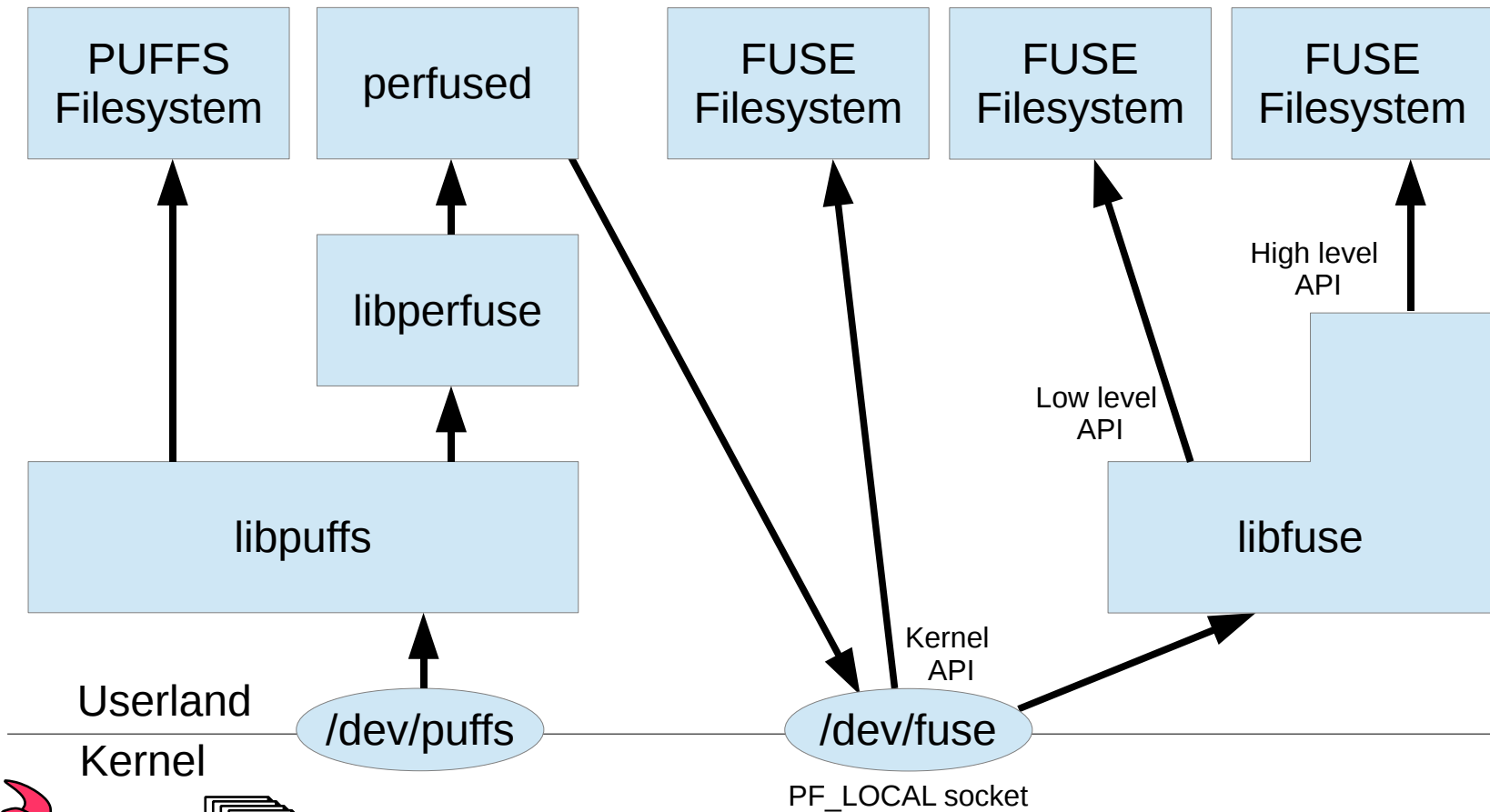


FUSE over PUFFS : PERFUSE (1)

- PERFUSE implements FUSE kernel API
 - But we do not want to work in kernel
 - perfused daemon, /dev/fuse socket
 - perfused translate PUFFS into FUSE
- Original libfuse is supported



FUSE over PUFFS : PERFUSE (2)



FUSE over PUFFS : PERFUSE (3)

- Special handling of /dev/fuse open & mount
 - /dev/fuse is a socket, open() & mount() will fail
 - perfuse_open() and perfuse_mount()
 - <perfuse.h> defines them as open() and mount()
 - Just use #include <perfuse.h> and -lperfuse
 - libfuse was modified upstream for that change
- /dev/fuse replaced by socketpair(2)



About VFS (1)

- VFS operations translated into PUFFS/FUSE
- Root node is obtained at mount time
- LOOKUP is used to find other nodes
 - Give a name, get a node (or an error)
- GETATTR, SETATTR for metadata
- OPEN, READ, WRITE, REaddir, etc...



About VFS (2)

`mount("/gfs") => node1`

`node1.lookup("foo") => ENOENT`

`node1.loookup("bar") => node2`

`node2.getattr() => ino,uid,gid,mode,ctime...`

`node2.open() => 0`

`node2.read() => data`



Node life cycle (1)

- Not obvious operations (NetBSD/PUFFS)
 - RELEASE : close() was called
 - INACTIVE : last reference is drop
 - RECLAIM : free and forget about a node
- Linux/FUSE :
 - RELEASE
 - FORGET



Node life cycle (2)

node2.release => 0

node2.read() => data

node2.inactive()=> 0

node1.inactive() => 0

node2.reclaim() => 0

node1.reclaim() => 0



Bugs and traps

- SOCK_SEQPACKET sockets
- Extended attributes
- Races in GETATTR
- dirname() thread-unsafety
- link() to a symlink
- pagedaemon has a nap
- swapcontext() swaps TLS



SOCK_SEQPACKET sockets

- /dev/fuse vs local socket semantics
 - /dev/fuse supports reliable atomic messages
 - Neither SOCK_STREAM, nor SOCK_DGRAM
 - We implemented SOCK_SEQPACKET



Extended attributes (1)

- Required for GlusterFS server component
- Bring back UFS1 extended attribute
 - Stored in a sparse file, just like quotas were
 - Autostart, backend autocreation
- Support in cp(1) and mv(1)
- Linux API vs FreeBSD API

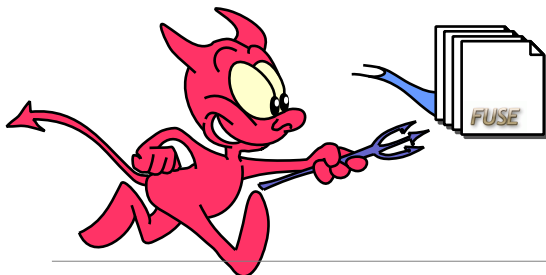
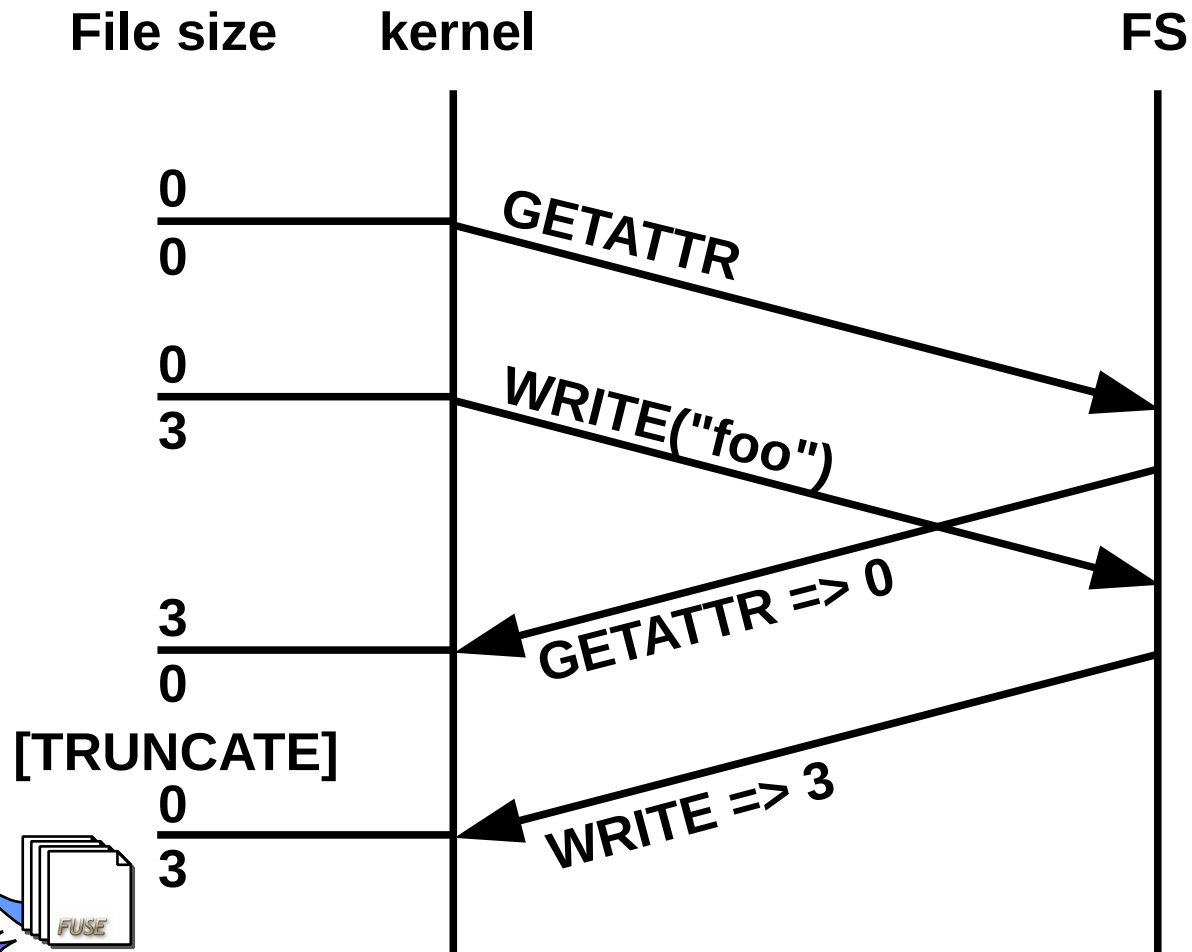


Extended attributes (2)

- Must be added in various utilities
 - Critical for backups : pax(1), dump(8), restore(8)
 - Useful : scp(1), rsync(1)
 - May break standards : tar(1), cpio(1),
- Extended attributes storage improvment
 - Import support for UFS2 from FreeBSD
 - Native filesystem implementation *à la* QUOTA2



Races in GETATTR



- Mutex of size is required

dirname() thread-unsafety

- GNU dirname() vs BSD dirname()
 - Static buffer with const input vs modified input
- ```
printf("=> %s %s\n", dirname("/foo/a"), dirname("/bar/b"));
```
- => /bar /bar
- No consensus for dirname\_r()
  - Add GNU dirname() in GlusterFS contrib/



# Link() to symlink (1)

- link() to symlink : symlink itself or target ?

lrwxrwxrwx 1 manu manu 6 nov. 1 17:06 symlink -> target

\$ ln symlink link

[Linux]lrwxrwxrwx 2 manu manu 6 nov. 1 17:06 link -> target

[xBSD] lrwxrwxrwx 2 manu manu 6 nov. 1 17:06 link -> symlink

- Both behavior are standard-compliant !
- GlusterFS relies on Linux behavior



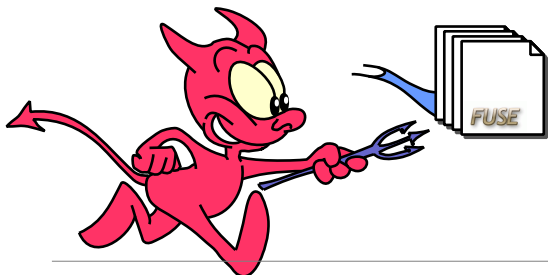
# Link() to symlink (2)

- linkat() has a AT\_NOFOLLOW option
- POSIX extended API set 2 implementation
  - Just a partial linkat(2) in netbsd-6 branch
  - Full set except fexecve(2) in netbsd-7 branch



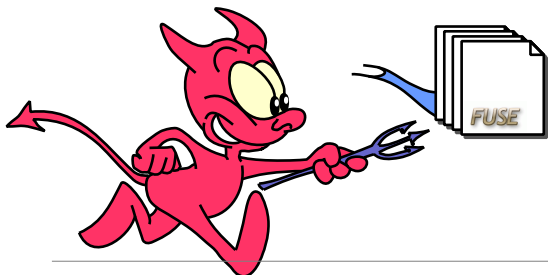
# Pagedaemon has a nap

- pagedaemon frees memory
- It may use PUTPAGE on a PUFFS vnode
- PUFFS message allocation : `kmem_zalloc()`
  - `KM_NOSLEEP` : fail if no memory available
  - `KM_SLEEP` : sleep if no memory available
- Pagedaemon must never sleep !



# swapcontext() swaps TLS (1)

- GlusterFS uses swapcontext() *and* Pthreads
- Should swapcontext() swaps the TLS register?
  - Linux : TLS preserved, swapcontext(3) is thread-safe
  - NetBSD : machine-dependent behavior



# swapcontext() swaps TLS (2)

```
[lwp1] getcontext(&nctx);
```

```
[lwp2] makecontext(&nctx, (void *)*swapfunc, 0);
```

```
[lwp2] swapcontext(&octx, &nctx);
```

```
[lwp2] printf("lwp2 self = %p\n", pthread_self());
```

```
[lwp1] printf("lwp1 self = %p\n", pthread_self());
```

```
lwp2 self = 0xbfa00000
```

```
lwp1 self = 0xbfa00000
```



# swapcontext() swaps TLS (3)

- When should TLS register be preserved?
  - Only if linking with -lpthread
- `_UC_TLSBASE` option
  - Context option to control TLS register fate
  - libpthread overrides setcontext() stub





# TODO

- Extended attributes for dump(8) and friends
- Extended attributes storage *à la* QUOTA2
- FUSE negative caching
- FUSE FS notifications to kernel
- CUSE (char device)

