RedirFS
Pimp your filesystem

Frantisek Hrbata
frantisek.hrbata@redirfs.org
www.redirfs.org

October 30, 2007

LinuxAlt 2007

Content

- Virtual Filesystem Switch
- Stackable Filesystems
- Linux Security Module
- Dazuko
- Redirecting FileSystem
- Filters

Virtual Filesystem Switch

- one of first implementation in SunOS 2.0 by Sun Microsystems in 1985
- handles system calls related to filesystems
- provides common interface to several kinds of filesystems
- abstraction layer between user space processes and filesystem drivers

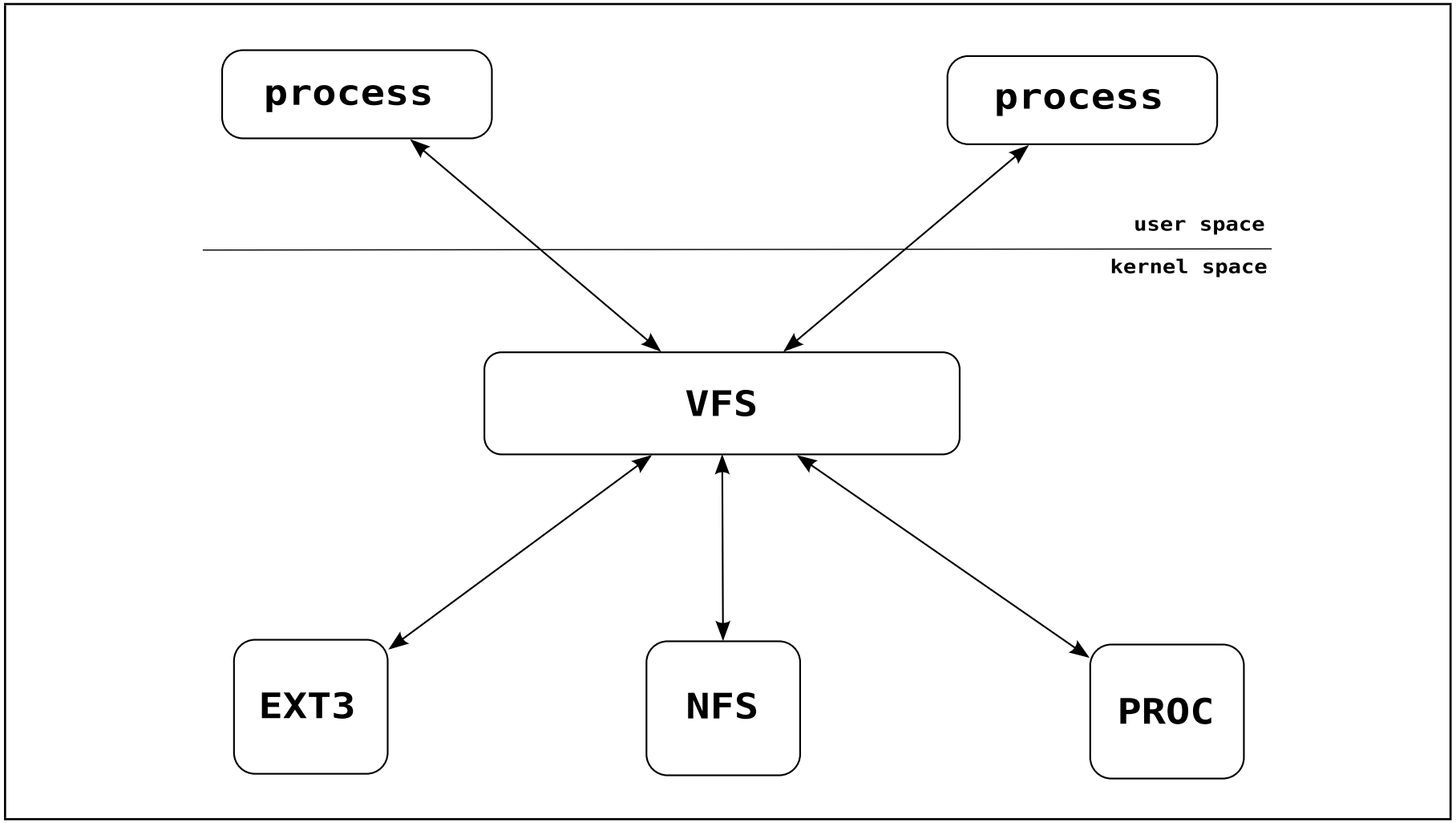
Virtual Filesystem Switch

- common filesystem model able to handle different filesystem types
- filesystem drivers have to translate their physical organization to the VFS' common model
- caches to eliminate calls to filesystem drivers

Filesystems Classes

- Disk-based filesystems – ext*, reiserfs, ufs, iso9669, hpfs
- Network filesystems – nfs, cifs, afs
- Special filesystems – sysfs, proc

Role



VFS Objects

- VFS object = data + table of operations
- operations filled by filesystem

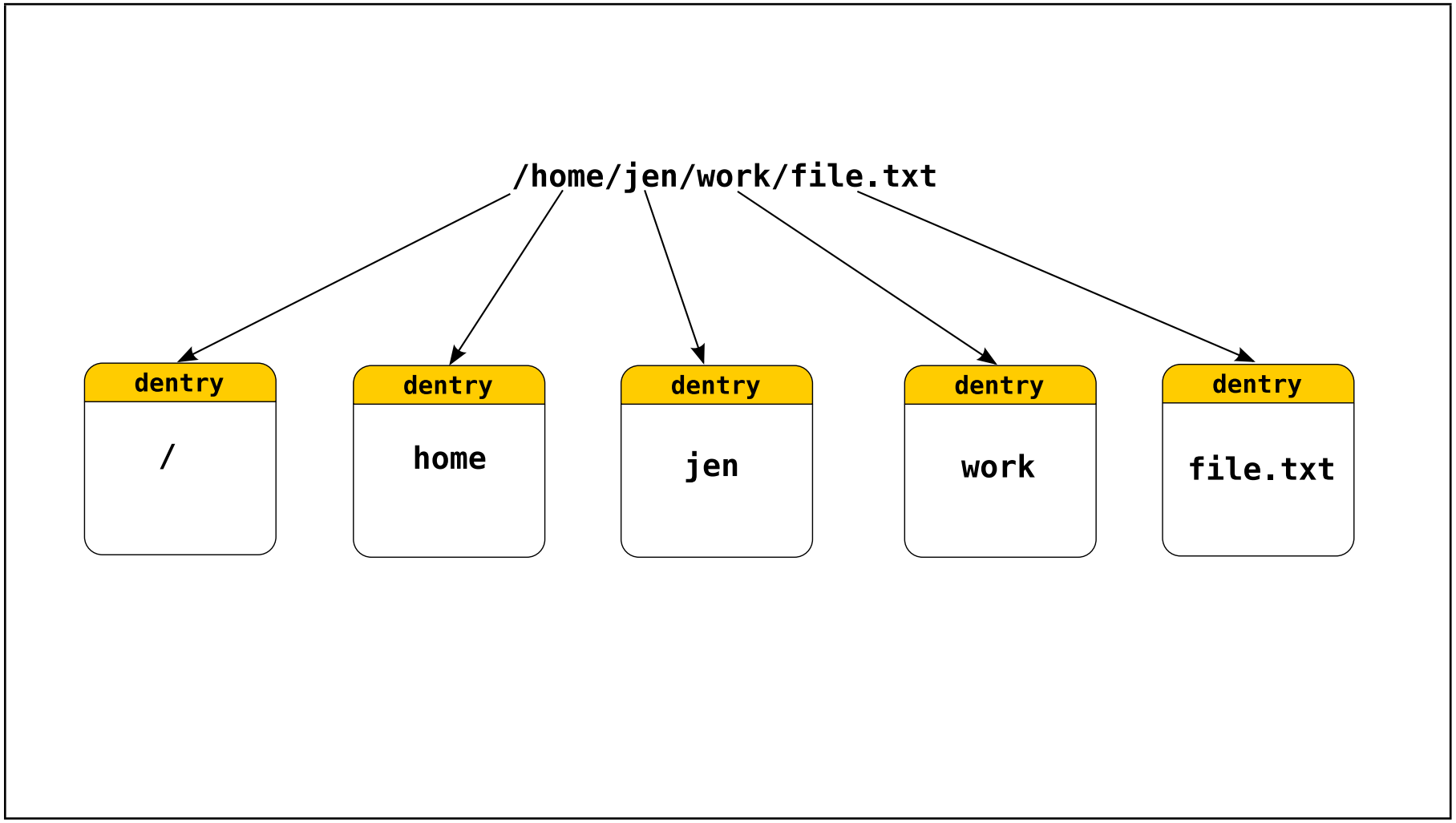
Dentry Object

- dentry – directory entry
- dentry object has no corresponding image on disk
- used for pathname lookup
- pathname lookup common for all filesystems
- created for each path component
- dentry has pointer to inode object

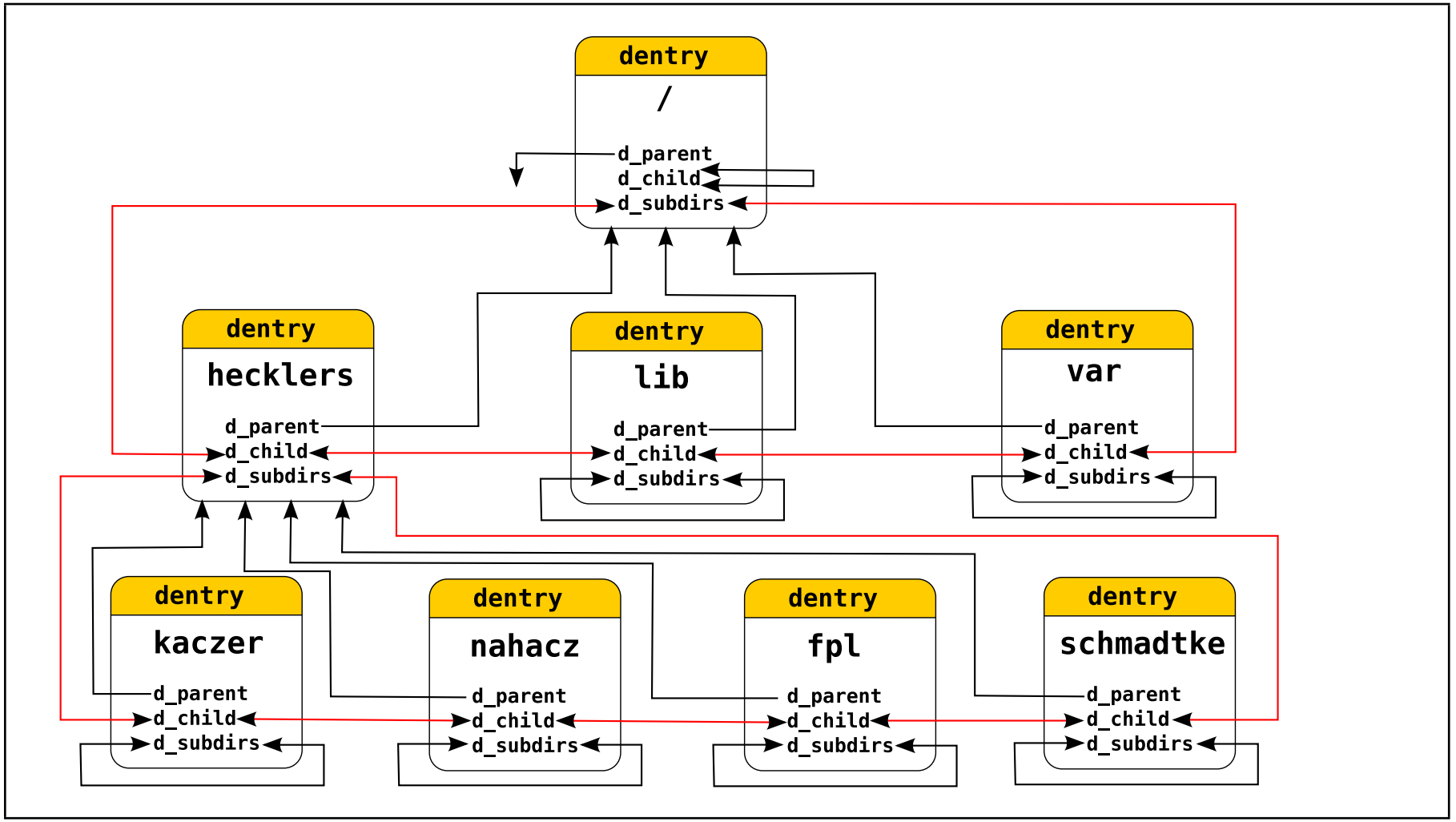
Dentry Object

- dentry without inode – negative dentry (pathname lookup speed–up)
- dentry objects are connected into the dentry tree
- dentry cache + dentry hash table
- dentry cache protected by `dcache_lock`
- dentry_operations

Dentries Created for Filename Path



Dentry Tree


dentry
/
d_parent
d_child
d_subdirs
dentry
hecklers
d_parent
d_child
d_subdirs
dentry
lib
d_parent
d_child
d_subdirs
dentry
var
d_parent
d_child
d_subdirs
dentry
kaczer
d_parent
d_child
d_subdirs
dentry
nahacz
d_parent
d_child
d_subdirs
dentry
fpl
d_parent
d_child
d_subdirs
dentry
schmadtke
d_parent
d_child
d_subdirs

Inode Object

- VFS inode vs. disk inode
- duplicates some of the data in the disk inode
- inode cache + inode hash table
- inode does not contain name
- list of dentry objects
- inode_operations
- file_operations

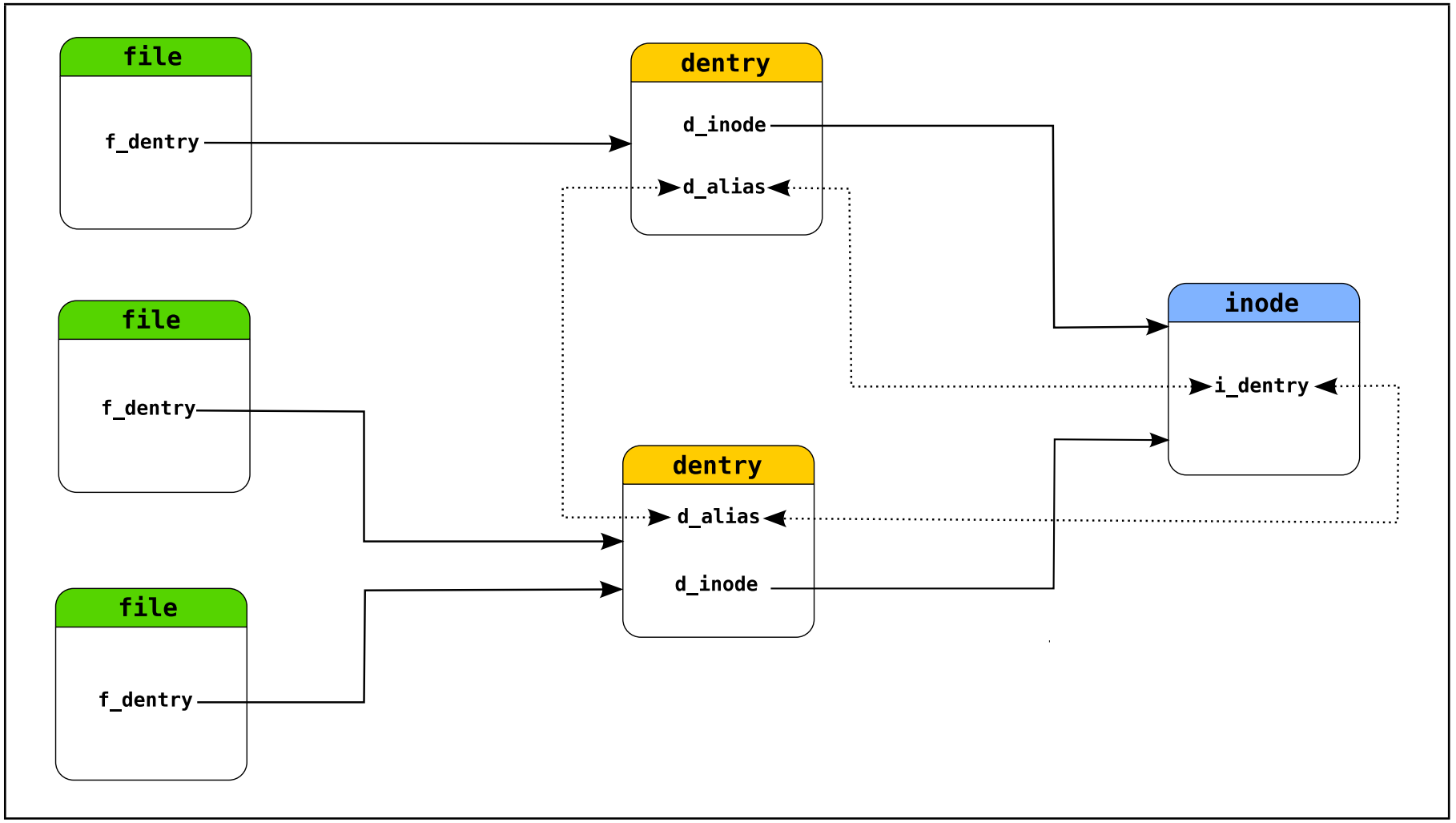
Inode Object

- contains address_space object
- buffer page cache
- radix tree of inode pages (data)
- address_space operations

File Object

- created for each open call
- contains file offset
- contains pointer to dentry object
- file_operations
- operations can be changed in the open operation (special files)

VFS Objects Connections



File_system_type Object

- thanks to this structure kernel knows that driver for filesystem is available
- contains filesystem name – ext2, reiserfs
- contains get_sb operation
- filled by filesystem
- register_filesystem
- list of all file_system_type objects

Superblock Object

- dentry root object for filesystem
- operations how to load and store data from and to filesystem
- read_inode, write_inode
- list of inodes
- list of files

Vfsmount Object

- created for each mount point
- Linux allows to stack multiple mounts on a single mount point
- d_mounted flag in dentry

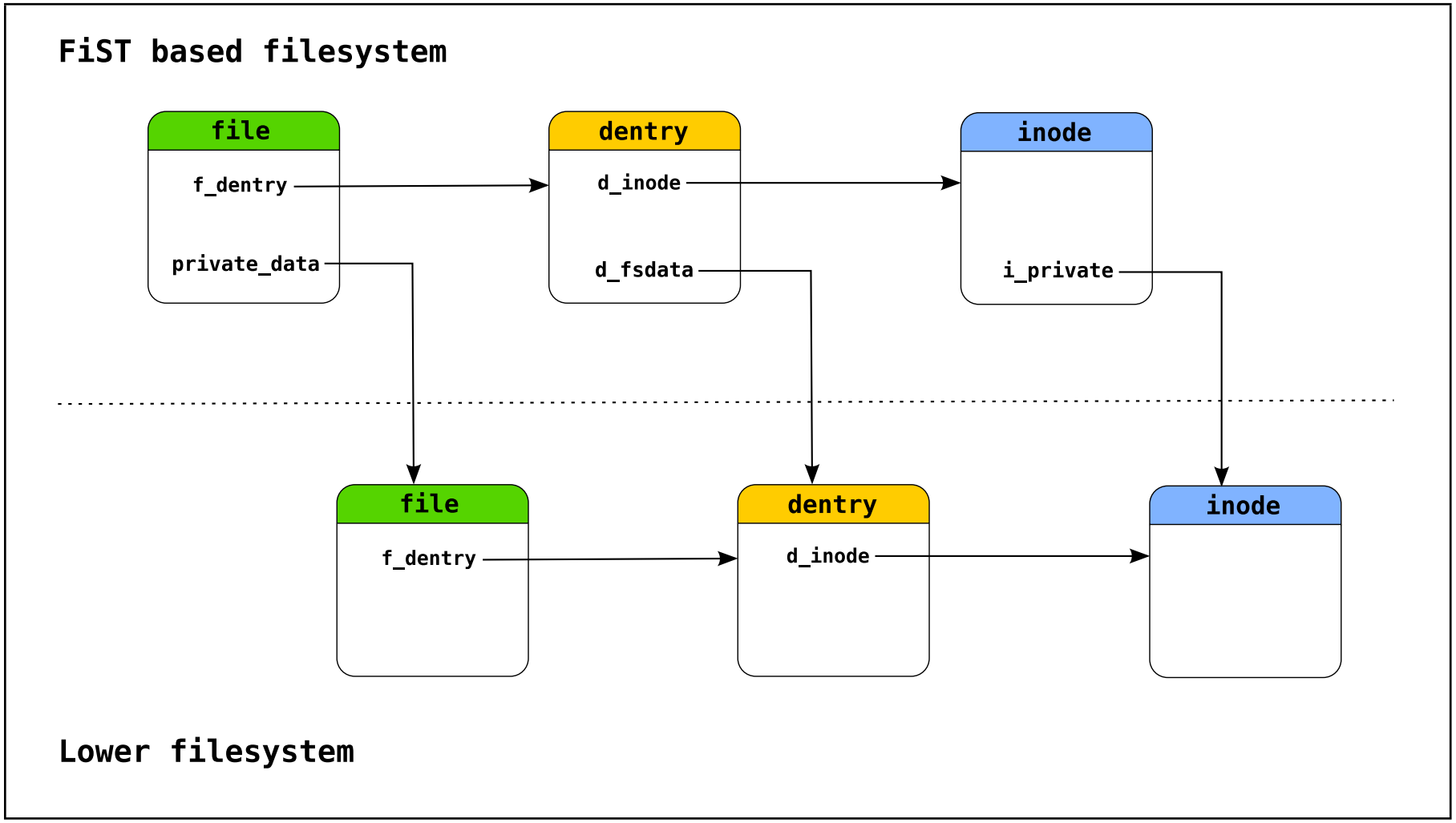
Pathname Lookup

- nameidata structure – vfs_mount and dentry
- finds dentry (inode) for pathname
- d_path reverse way

FiST: Stackable Filesystem

- File System Translator, Erez Zadok
- supports Linux, FreeBSD, Solaris
- code generator fistgen
- ecryptfs – vanilla kernel, based on cryptfs
- unionfs – mm tree
- uses Linux ability to mount one filesystem over other

FiST: Stackable Filesystem



Linux Security Module

- first SELinux by NSA
- set of security hooks to control operations on kernel objects
- security_operations
- only one security module registered directly to LSM – master

Dazuko: Access Control

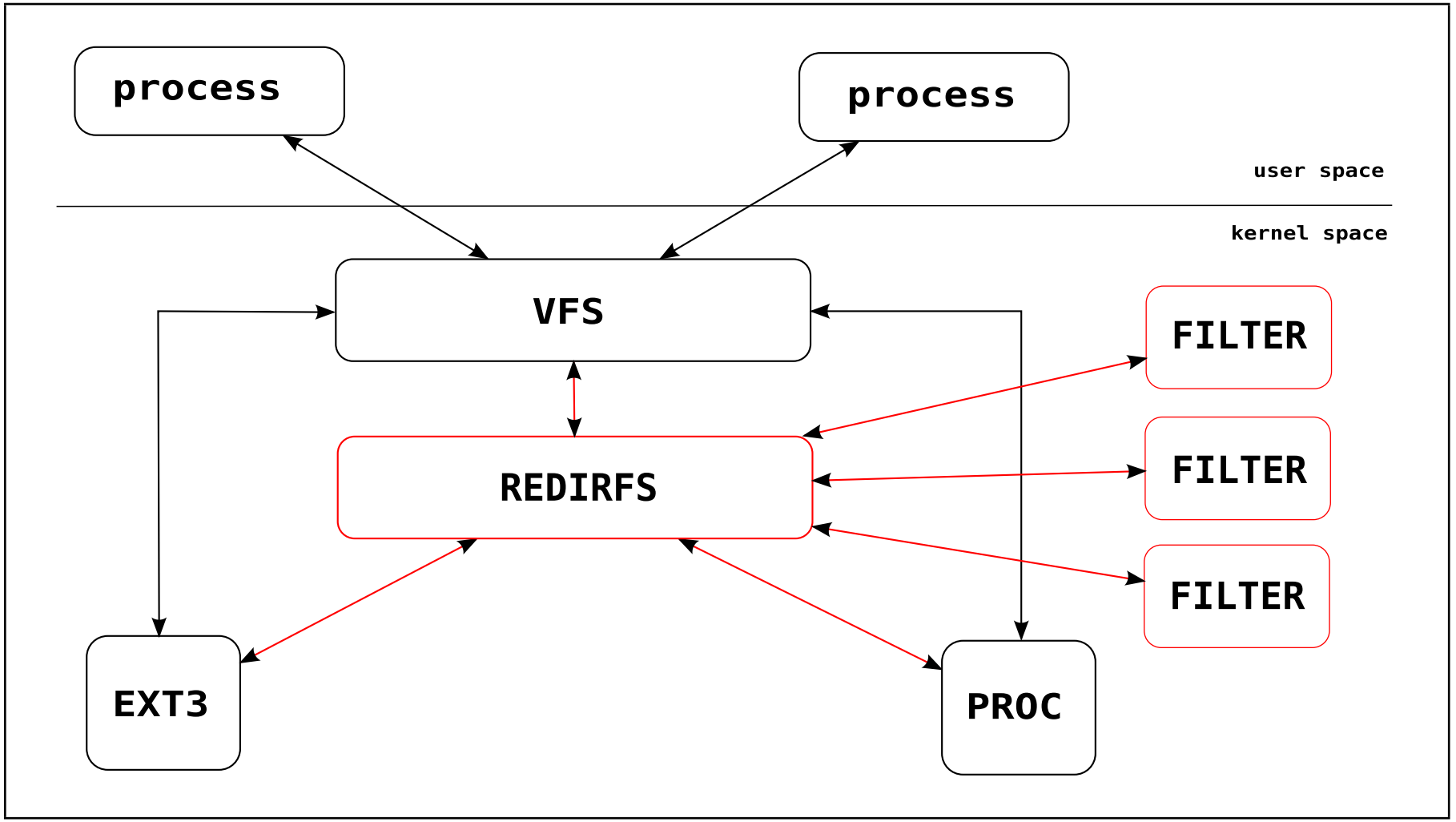
- founded by Avira GmbH (H+BDEV Datatechnik GmbH)
- maintainer – John Ogness
- kernel module + user space library
- mainly used for on-access scanning
- Avira, AVG, NOD32, Avast!, Clam, F-Secure, AntiExploit
- supports Linux and FreeBSD
- system call table hooking, LSM, DazukoFS

Redirecting FileSystem

- allows to redirect file system calls in the VFS layer
- is implemented as an out-of-kernel module for Linux 2.6
- is able to register, unregister and manage one or more filters
- allows filter to set its callback functions on-the-fly
- allows filter to include and exclude its paths on-the-fly
- allows filter to forward data from pre to post callback function
- allows filter to attach its private data to VFS objects

- allows filter to do subcalls for selected file system calls
- calls pre and post callback function of filters in fixed order specified by their priorities (call chain)
- reacts on return value from filter and it is able to interrupt filters call chain
- redirects only operations selected by one or more filters, all other operations go directly to the file system with no time overhead
- modifies only VFS objects which belong to paths selected by one or more filters

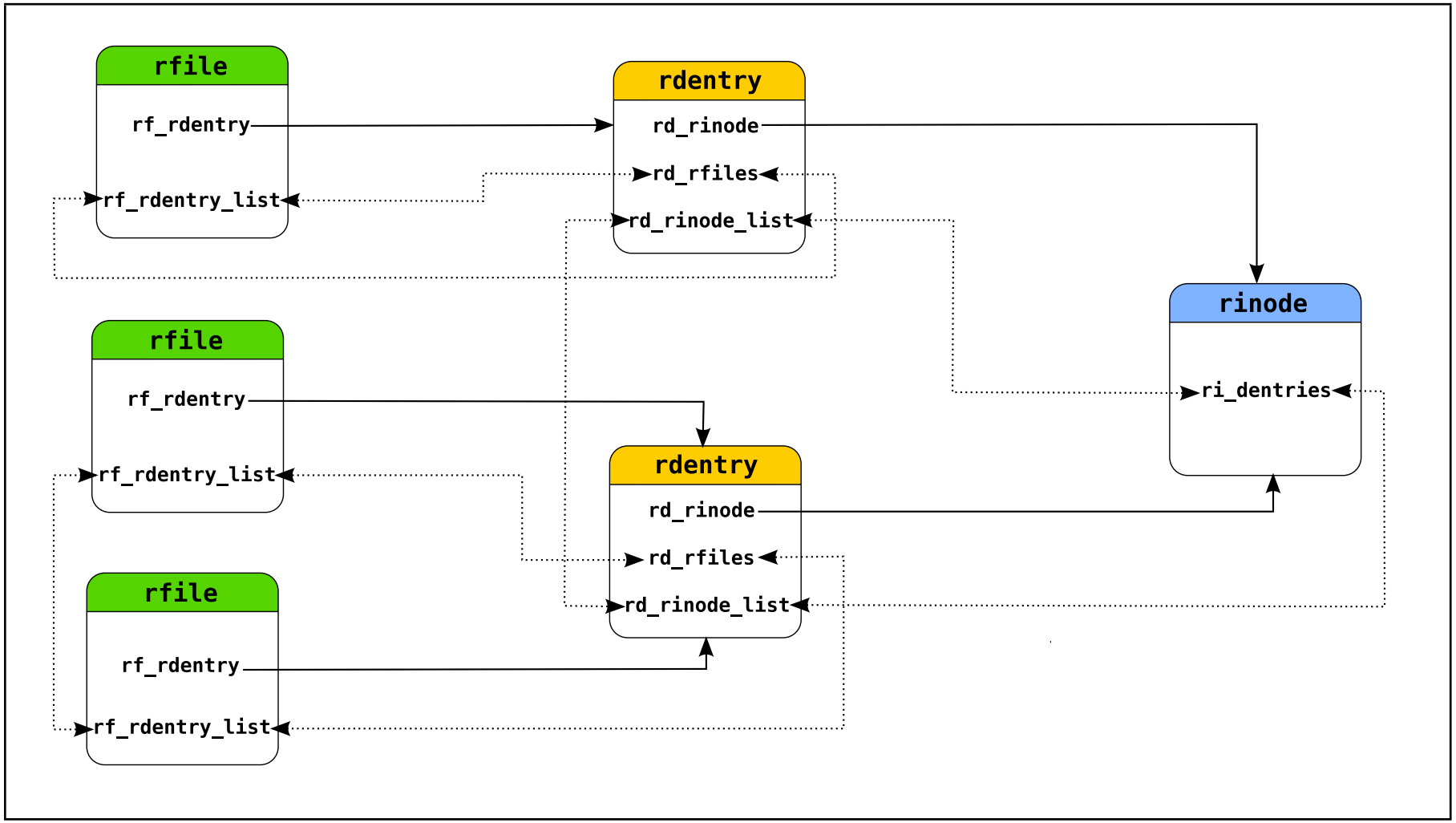
Role


process
process
user space
kernel space
VFS
FILTER
REDIRFS
FILTER
FILTER
EXT3
PROC

Objects

- for each VFS file, dentry, inode object is created corresponding RedirFS rfile, rdentry, rinode object
- RedirFS objects exists along with the corresponding VFS objects
- rfile, rdentry and rinode objects are used as a connection with VFS
- rdentry is created under d_lock, rinode under i_lock
- rfile creation does not need locking

Objects Connections



VFS Object Operation Replacement

- RedirFS is based on replacing of VFS objects operations
- RedirFS is using the fact that pointer assignment in Linux kernel is atomic
- RedirFS creates new operations for each VFS object
- new operations for VFS objects are embedded in RedirFS objects

VFS Object Operation Replacement

- replacement

1. alloc new operations
2. copy old operations
3. set new operations selected by filters callbacks
4. replace VFS object operations

VFS Object Operations Replacement

- operation stays the same as original until one or more filters set callback function for it
- RedirFS needs to keep track of created and deleted VFS objects so it can replace operations for newly created objects and on the other hand restore operations when the VFS objects are deleted.

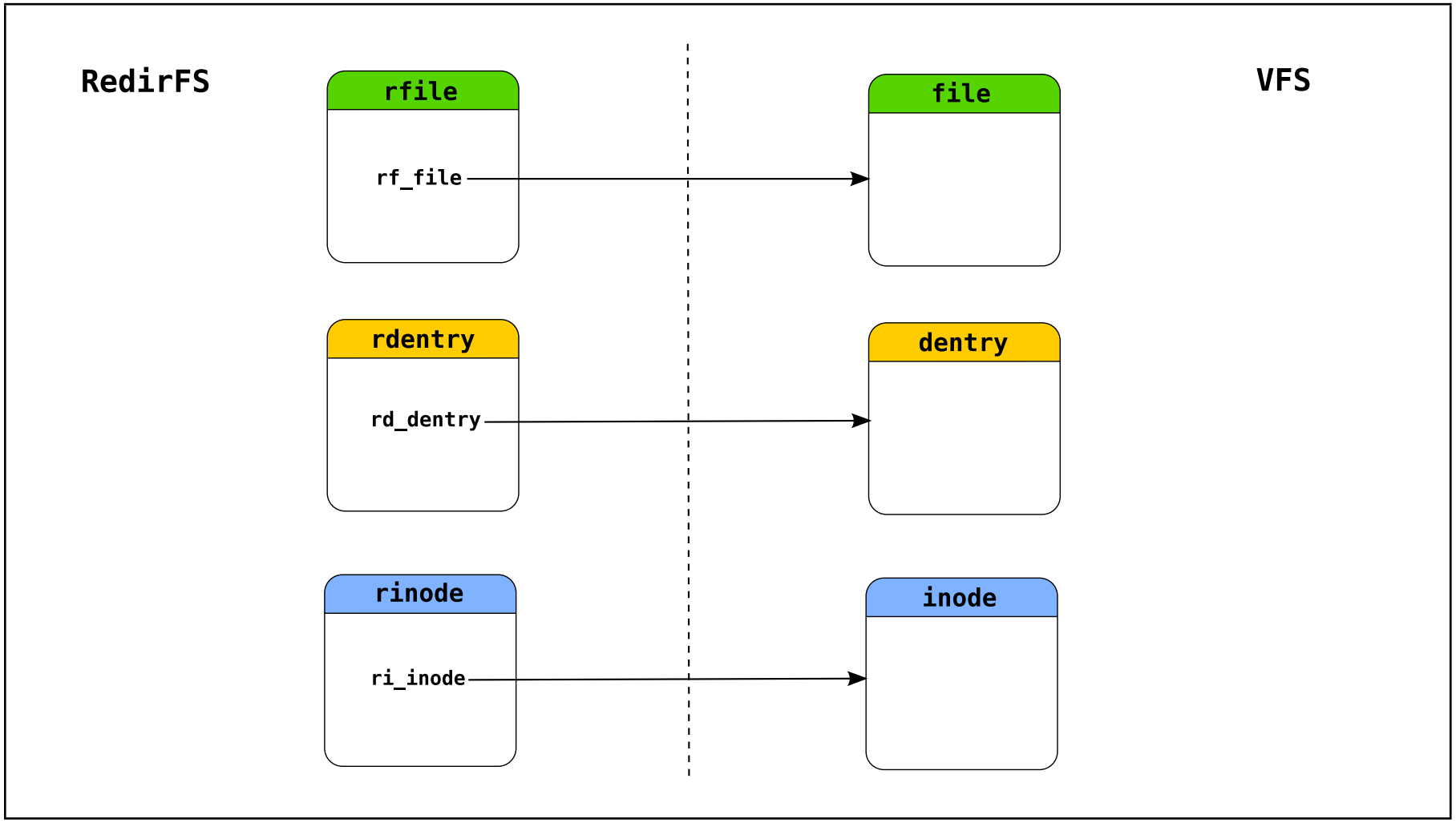
VFS Object Operations Replacement

- file_operations – open, release – all, readdir – dir
- dentry_operations – d_iput, d_release – all
- inode_operations – mkdir, create, link, symlink, mknod – dir, lookup – all

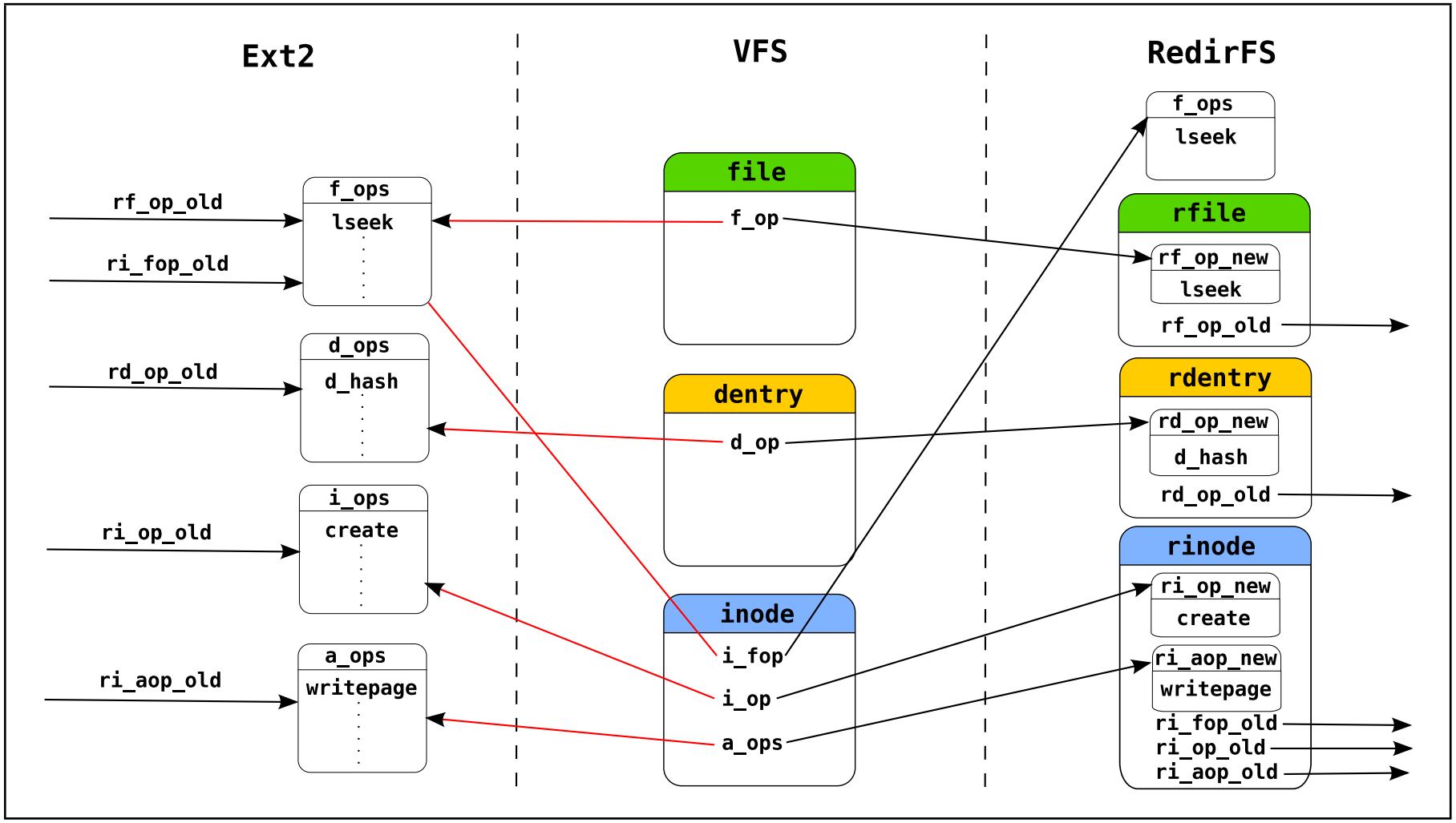
RedirFS and VFS connections

- RedirFS => VFS: RedirFS object has pointer to VFS object
- VFS => RedirFS: pointer of VFS object operations points to operations stored within RedirFS object (container_of macro)
- Problem: VFS layer calls RedirFS operation but right before the RedirFS object is obtained, it is deleted
- Solution: RCU protected pointers to operations and fixed operations (inode – lookup, d_iput – dentry, open – file)

RedirFS Connection with VFS



VFS Connection with RedirFS



Walking Through the Dentry Cache

- uses exported `dcache_lock` and directory i_mutex
- dentry objects are connected into tree
- dentry object has pointer to inode (negative dentry!)
- replacing, restoring and setting VFS objects operations

```
int rfs_walk_dcache(struct dentry *root, int flags,
                    int (*dcb)(struct dentry *dentry, void *dentry_data),
                    void *dcb_data,
                    int (*mcb)(struct dentry *dentry, void *dentry_data),
                    void *mcb_data)
```

Filter

- Filter in the RedirFS is represented by the filter structure. Each filter has a name, a unique priority number, a set of pre and post callback operations and a set of paths

Filters Call Chain

- each RedirFS object contains so called filters call chain
- filters call chain is a list of filters which are interested in one or more operations of the VFS object sorted according to their priorities
- this list tells RedirFS which Filters have to be called before (pre callbacks) and after (post-callbacks) the original operation

Filters Call Chain

- filter can finish or stop the proceeding operation in its pre callback function
- post callback functions are called for each filter for which its pre callback function was called

Subcalls

- subcalls are RedirFS functions for selected VFS operations
- subcall calls only filters which are in the filter call chain past the filter which called the subcall
- subcalls allow filter to call the same VFS operation with its own or modified arguments
- subcalls also allow filter to call different operations

Private Data

- filters are allowed to attach their private data to file, inode, dentry
- synchronization needed
- private data for filters are kept in a list in the RedirFS object corresponding to the VFS object
- filter can attach private data per operation in the pre callback function
- filter has to detach them in the post callback function

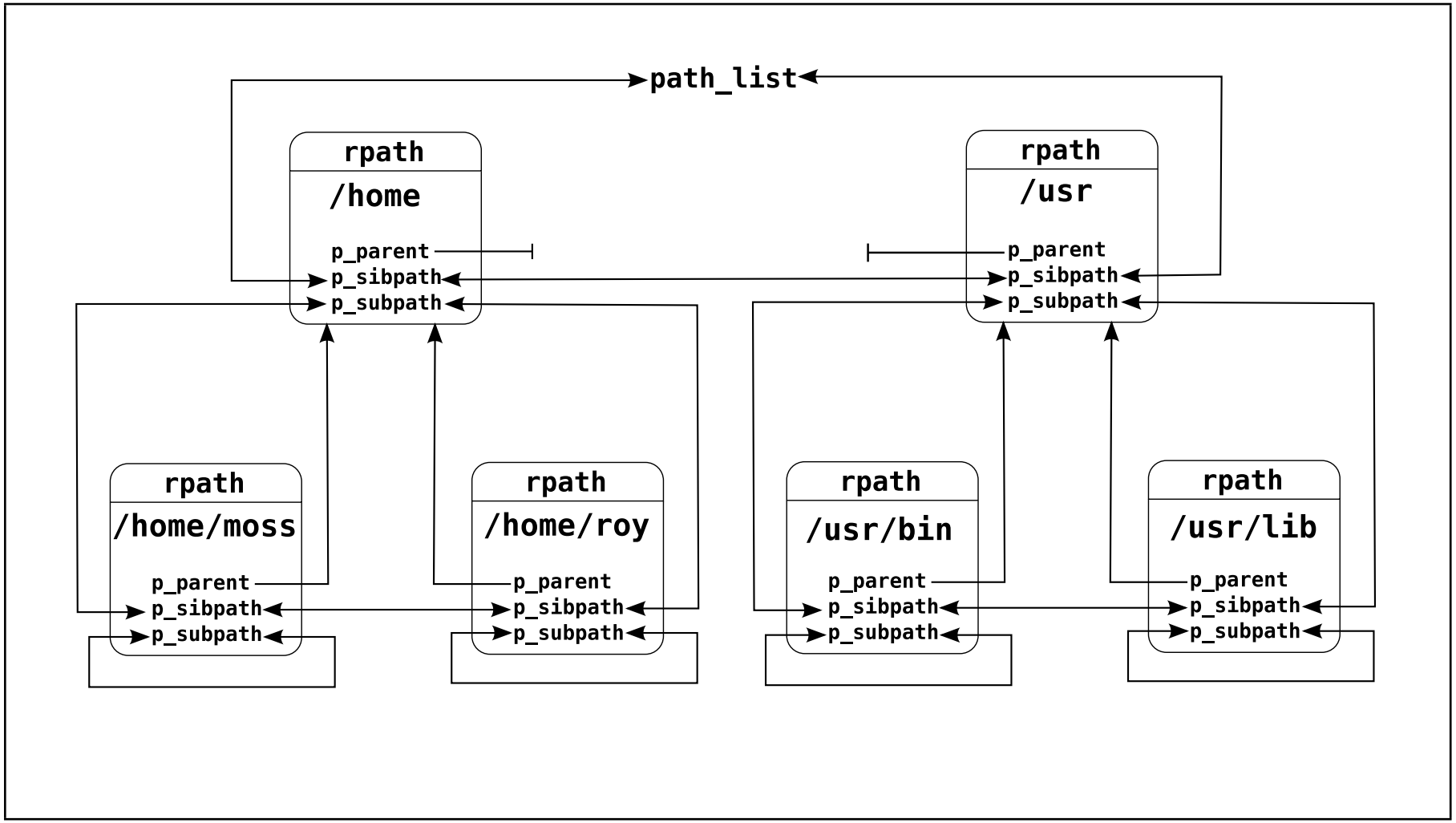
Path Management

- paths selected by filters are in the RedirFS represented by the `rpath` structure
- filters can include or exclude directory subtrees or even single path (file or directory)
- rpath objects are connected in trees and the root of each tree is stored in the global `path_list` list

Path Management

- rpath has lists of filters call chains – `p_inchain`, `p_exchain`, `p_inchain_local`, `p_exchain_local`
- rpath has operations for VFS objects belonging to it
- filter can not set fixed paths

Path Tree


path_list
rpath
/home
p_parent
p_sibpath
p_subpath
rpath
/usr
p_parent
p_sibpath
p_subpath
rpath
/home/moss
p_parent
p_sibpath
p_subpath
rpath
/home/roy
p_parent
p_sibpath
p_subpath
rpath
/usr/bin
p_parent
p_sibpath
p_subpath
rpath
/usr/lib
p_parent
p_sibpath
p_subpath

Walking Through Paths

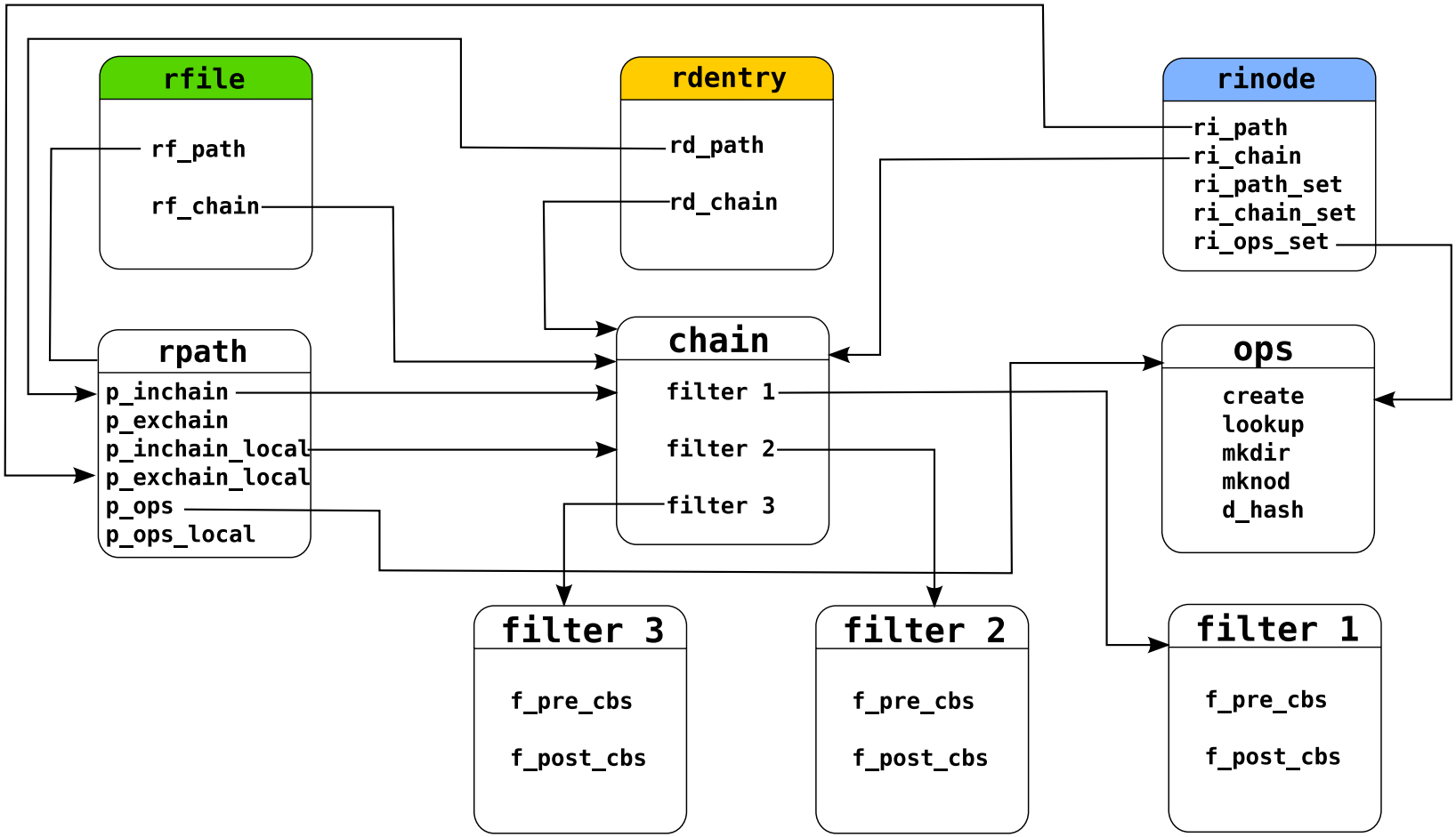
- rpath borders for `rfs_walk_dcache` are stored in rdentry – `rd_root`

```
int rfs_path_walk(struct rpath *path, int walkcb(struct rpath*, void*),
                  void *datacb);
```

Sysfs Interface

- RedirFS creates same basic attributes for each filter in the sysfs file system. For each filter is created a directory /sys/fs/redirfs/<filter name> with following files (attributes)
- active – activated or deactivated filter
- paths – included and excluded paths, path can be set via this file
- control – flag if filter is accepting setting via the sysfs
- priority – filter's priority

Objects Connections



Filters

- filter is a linux kernel module (LKM) using the RedirFS framework
- filter can add some useful functionality to existing filesystems like transparent compression, transparent encryption, merging contents of several directories into the one, allowing writing to a read-only media and other
- filter has a unique number called priority which defines its order in the filter call chain

Filters

- filter defines a set of pre and post callback functions for selected filesystem calls
- filter defines paths over which its callback functions will be called
- filter can include or exclude paths as a subtrees or as a single paths

Registration & Unregistration

- filter registered to RedirFS is identified by a handle
- registration requires properly filled `rfs_filter_info`

```
rfs_filter filter;

struct rfs_filter_info {
        char *name;
        int priority;
        int active;
        int (*ctl_cb)(struct rfs_ctl *ctl);
};

int rfs_register_filter(rfs_filter *flt, struct rfs_filter_info *filter_info);

int rfs_unregister_filter(rfs_filter filter);
```

Activation & Deactivation

- filter can be active or inactive
- RedirFS calls only active filters
- filter's state can be set during registration
- filter's state can be changed anytime after registration

```
int rfs_activate_filter(rfs_filter filter);
int rfs_deactivate_filter(rfs_filter filter);
```

Operations Settings

- filter can register pre and post callback functions for selected filesystem calls
- each operation is described by the `rfs_op_info` structure

```
struct rfs_op_info {
        enum rfs_op_id op_id;
        enum rfs_retv (*pre_cb)(rfs_context, struct rfs_args *);
        enum rfs_retv (*post_cb)(rfs_context, struct rfs_args *);
};
static struct rfs_op_info op_info[] = {
        {RFS_DIR_IOP_PERMISSION, NULL, dummyflt_permission},
        {RFS_REG_FOP_OPEN, dummyflt_open, NULL},
        {RFS_OP_END, NULL, NULL}
};
int rfs_set_operations(rfs_filter filter, struct rfs_op_info *op_info);
```

Filter's Callback Functions

- all pre and post callback functions have the same interface
- filter can stop filters call chain
- filter can modify or replace original operation arguments

```
enum rfs_retv cb(rfs_context cont, struct rfs_args*rargs);

struct rfs_args {
        union rfs_op_args args;
        union rfs_op_retv retv;
        struct rfs_op_type type;
};
```

Paths Settings

- include or exclude – `RFS_PATH_INCLUDE, RFS_PATH_EXCLUDE`
- subtree or single path – `RFS_PATH_SUBTREE, RFS_PATH_SINGLE`
- filter can not set fixed paths

```
struct rfs_path_info {
        char *path;
        int flags;
}

int rfs_set_path(rfs_filter filter, struct rfs_path_info *path_info);
```

Control Callback Function

- request for new settings entered through the sysfs interface
- filter can decide if it will follow new settings or just ignore it
- only settings common for all filters
- `RFS_CTL_ACTIVATE, RFS_CTL_DEACTIVATE, RFS_CTL_SETPATH`

```
struct rfs_ctl {
        enum rfs_ctl_id id;
        union rfs_ctl_data data;
};
```

Sysfs Interface

- for each filter has a directory `/sys/fs/redirfs/<filter name>`
- sysfs attributes – active, control, priority, paths

```
int rfs_register_attribute(rfs_filter flt, struct rfs_flt_attribute *attr);
int rfs_unregister_attribute(rfs_filter flt, struct rfs_flt_attribute *attr);

struct rfs_flt_attribute {
        struct attribute attr;
        ssize_t (*show)(rfs_filter flt, struct rfs_flt_attribute *attr,
                        char *buf);
        ssize_t (*store)(rfs_filter flt, struct rfs_flt_attribute *attr,
                         const char *buf, size_t size);
        void *data;
};
#define rfs_flt_attr(__name, __mode, __show, __store, __data)
```

Private Data for VFS objects

- filter can attach, detach and get its private data for file, dentry and inode

```
int rfs_attach_data_inode(rfs_filter filter, struct inode *inode,
                          struct rfs_priv_data *data,
                          struct rfs_priv_data **exist);
int rfs_detach_data_inode(rfs_filter filter, struct inode *inode,
                          struct rfs_priv_data **data);
int rfs_get_data_inode(rfs_filter filter, struct inode *inode,
                       struct rfs_priv_data **data);

int rfs_init_data(struct rfs_priv_data *data, rfs_filter filter,
                  void (*cb)(struct rfs_priv_data *));

void rfs_put_data(struct rfs_priv_data *data);
struct rfs_priv_data *rfs_get_data(struct rfs_priv_data *data);
```

Private Data for Operation Context

- filter can attach its private data per operation and pass it from pre to post callback function
- filter has to detach its private data in the post callback function

```
int rfs_init_data_cont(struct rfs_cont_data *data, rfs_filter filter);
int rfs_attach_data_cont(rfs_filter filter, rfs_context *context,
                         struct rfs_cont_data *data);
int rfs_detach_data_cont(rfs_filter filter, rfs_context context,
                         struct rfs_cont_data **data);
```

Subcalls

- RedirFS functions for selected VFS operations
- subcall calls only filters which are in the filter call chain past the filter which called the subcall
- `<type> rfs_<name>_subcall(rfs_filter flt, union rfs_op_args *args);`

```
ssize_t rfs_read_subcall(rfs_filter flt, union rfs_op_args *args);
ssize_t rfs_aio_read_subcall(rfs_filter flt, union rfs_op_args *args);
int rfs_readpage_subcall(rfs_filter flt, union rfs_op_args *args);
ssize_t rfs_write_subcall(rfs_filter flt, union rfs_op_args *args);
```

Thanks for your attention

Questions?

References

- Bovet, P. B., Cesati, M.: Understanding the Linux Kernel. 3rd Edition, U.S.A., O'Reilly 2005
- Love, R.: Linux Kernel Development. 2nd Edition, Indianapolis, Indiana, Novel Press 2005
- Alessandro Rubini, Jonathan Corbet: Linux Device Drivers, 2nd Edition, U.S.A., O'Reilly 2001
- www.filesystems.org, www.dazuko.org
- http://www.nsa.gov/selinux/papers/module/t1.html