Evolution of FUSE and OverlayFS

Miklos Szeredi
miklos@szeredi.hu

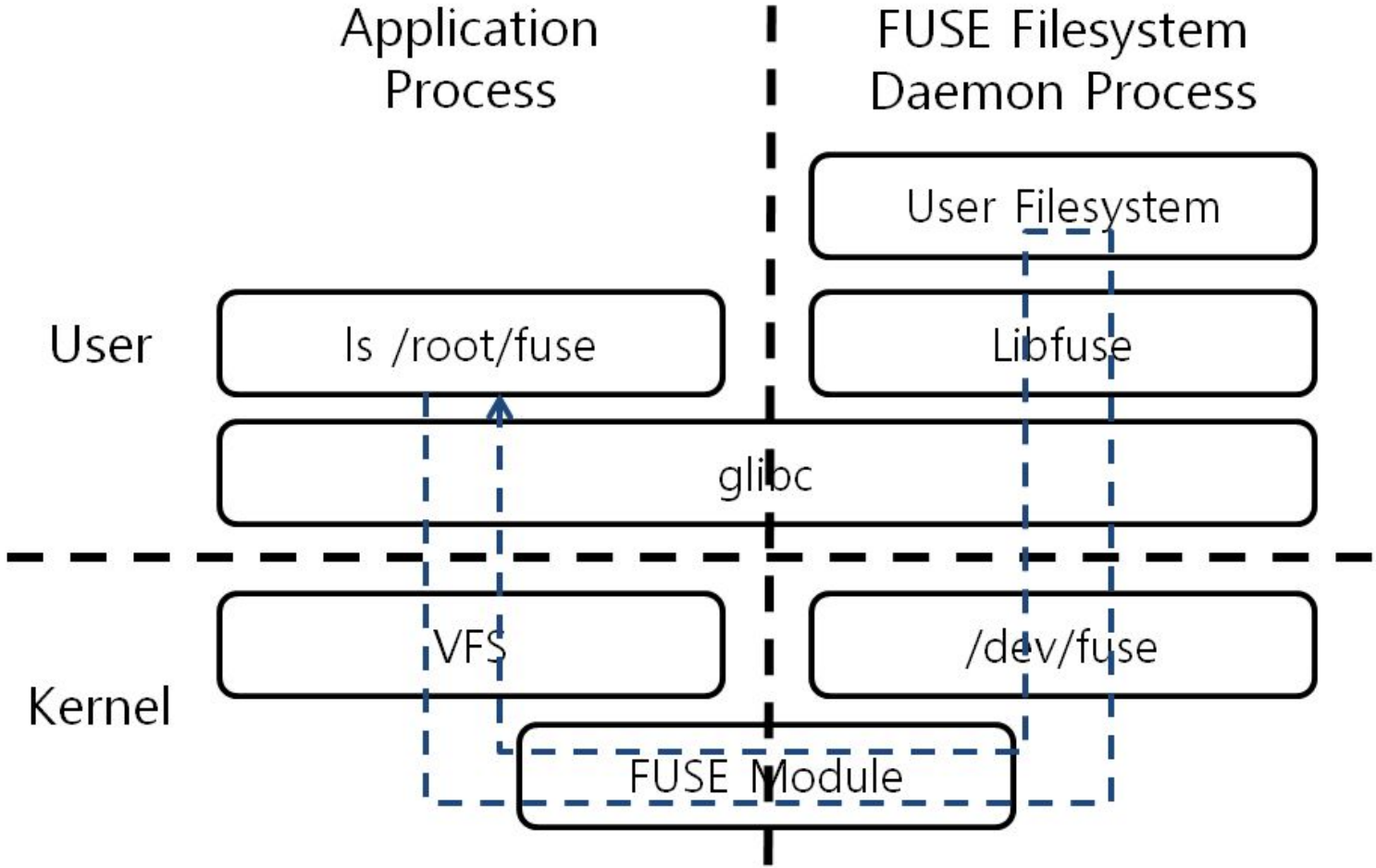
Filesystem in Userspace

Design principles

Filesystems should **run unprivileged**, must not be able to compromise the kernel

Interface should be as **simple** as possible

Mounting unprivileged should be supported (still only through suid helper)


Application Process
FUSE Filesystem Daemon Process
User Filesystem
User
ls /root/fuse
Libfuse
glibc
VFS
/dev/fuse
Kernel
FUSE Module

Path based API

Easy to understand for userspace developers

stat(“*/mnt/dir/foo*”, stbuf) -> getattr(“*/dir/foo*”, stbuf)

mkdir(“*/mnt/dir/bar*”, 0700) -> mkdir(“*/foo/bar*”, 0700)

Low level API

Closer to the kernel internal filesystem API

stat("*/mnt/dir/foo*", stbuf) ->

lookup(req, 1, "*dir*") -> fuse_reply_entry(req, {*DIR_INO*, ...}, ...)

lookup(req, *DIR_INO*, "*foo*") -> fuse_reply_entry(req, {*FOO_INO*, …}, …)

getattr(req, *FOO_INO*, fi) -> fuse_reply_attr(req, stbuf, ...)

Raw kernel ABI

Read & write binary protocol on */dev/fuse*

```
struct fuse_in_header {
    uint32_t    len;
    uint32_t    opcode;
    uint64_t    unique;
    uint64_t    nodeid;
...
```

Features

Zero Copy

ssize_t **splice**(*int* **fd_in**, *loff_t* ***off_in**, *int* **fd_out**, *loff_t* ***off_out**, *size_t* **len**, *unsigned int* **flags**);

Works on pipe buffers

Data doesn’t need to cross between kernel/userspace

Writeback cache

FUSE cannot provide guarantees about writeback

Need to be careful about how much “dirty” data to allow

Need to be careful not to deadlock while waiting for memory to be freed

libfuse3

Maintenance taken over by **Nikolaus Rath**

Libfuse2 accumulated >10years of backward compatibility cruft

Libfuse3 breaks source and binary compatibility with libfuse2

But only **minor changes** to filesystems needed

Future plans

Namespaces

Pid namespace support done

User namespace support - work stalled

Scalability

Minimise cost of switching tasks

Minimise cost of switching between kernel and userspace

Minimise cost of copying data

Caching

Try caching everything that can be cached

- Directory contents
- Symlink contents
- Extended attributes

Overlay Filesystem

Design Principles

No underlying filesystem format changes

No special tools required (e.g. to backup)

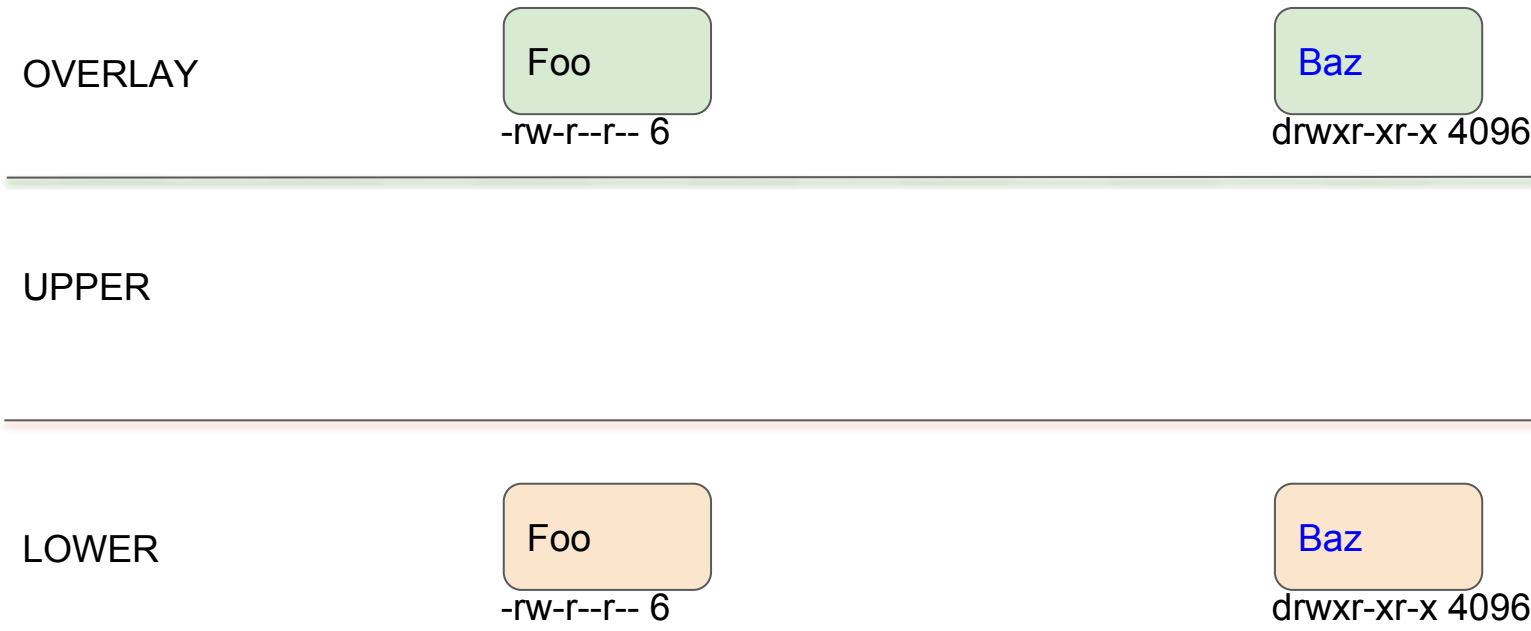
Implementation should be as simple as possible

Underlying layers are just storage

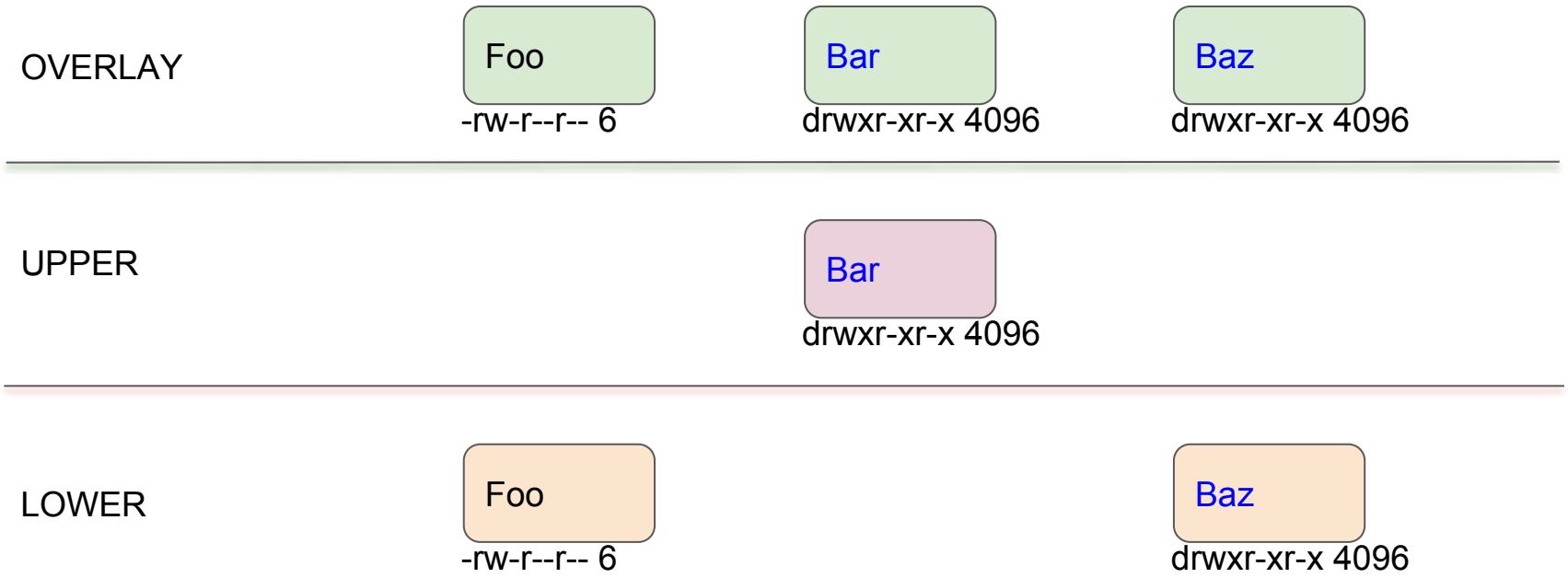
Page cache should be shared

Can recycle upper layer to a lower layer

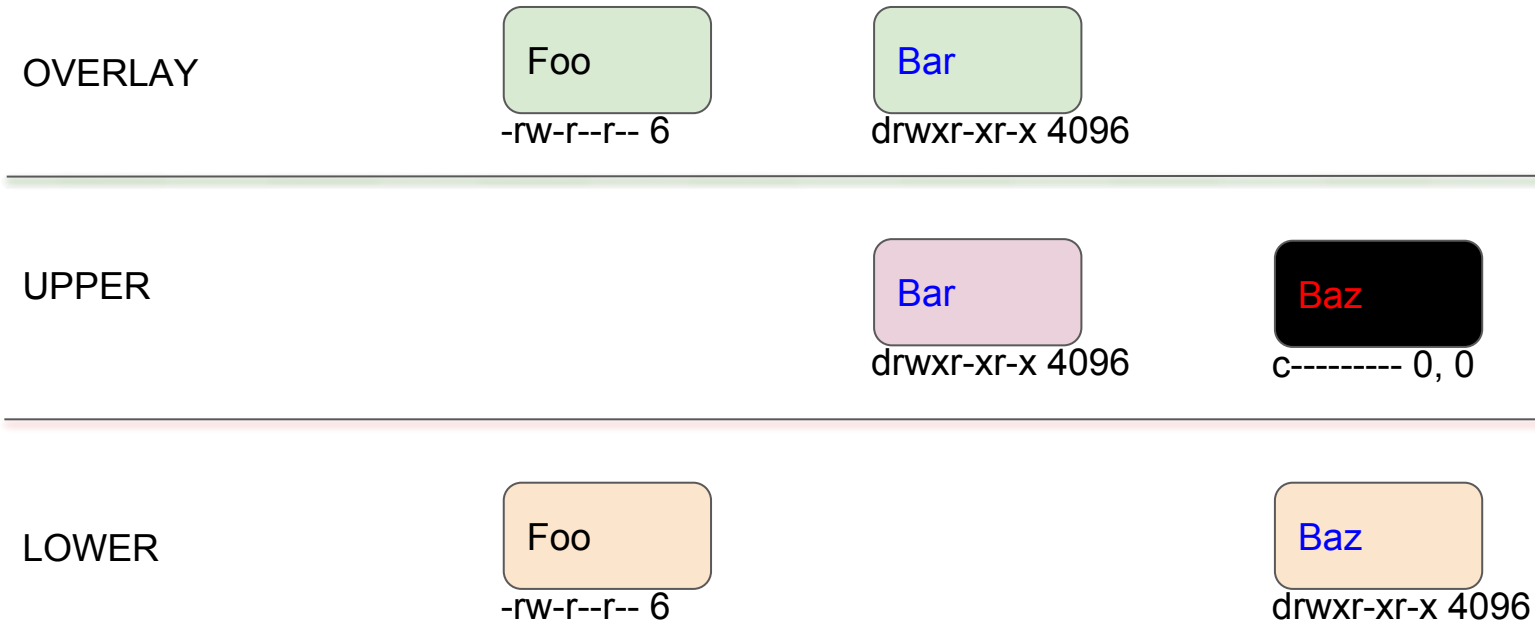
Basic operation



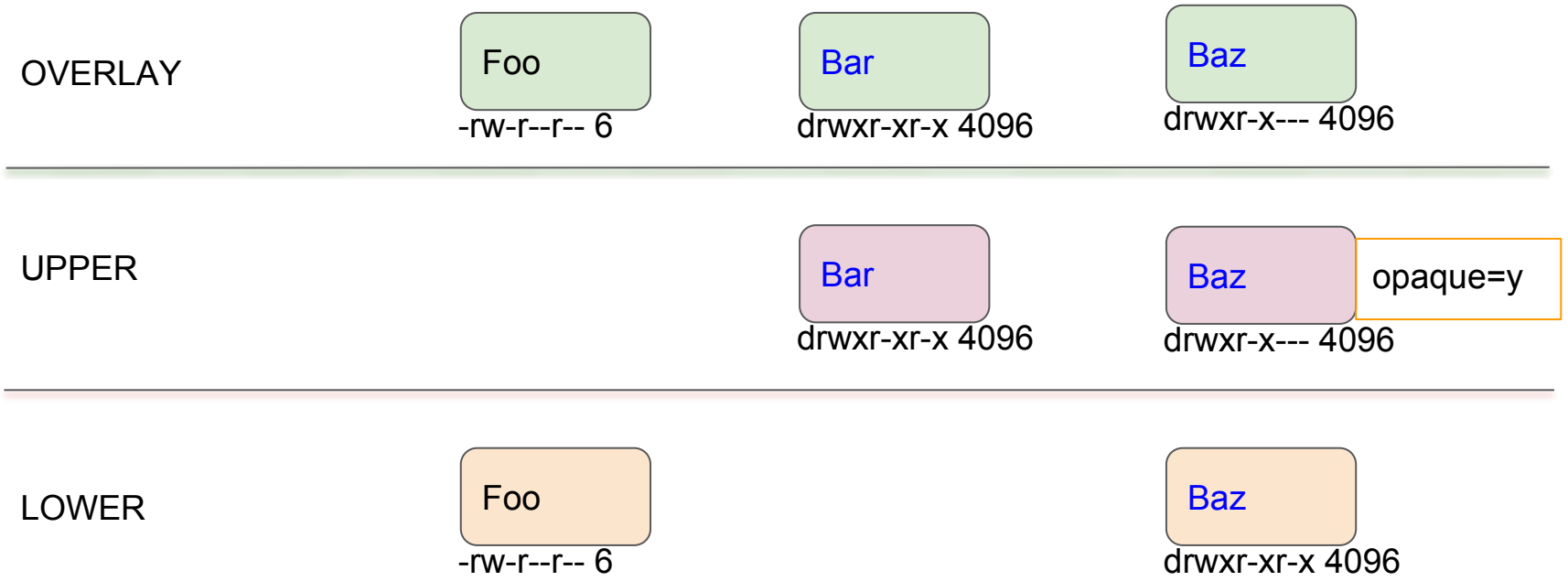
mkdir Bar


OVERLAY
Foo
-rw-r--r-- 6
Bar
drwxr-xr-x 4096
Baz
drwxr-xr-x 4096
UPPER
Bar
drwxr-xr-x 4096
LOWER
Foo
-rw-r--r-- 6
Baz
drwxr-xr-x 4096

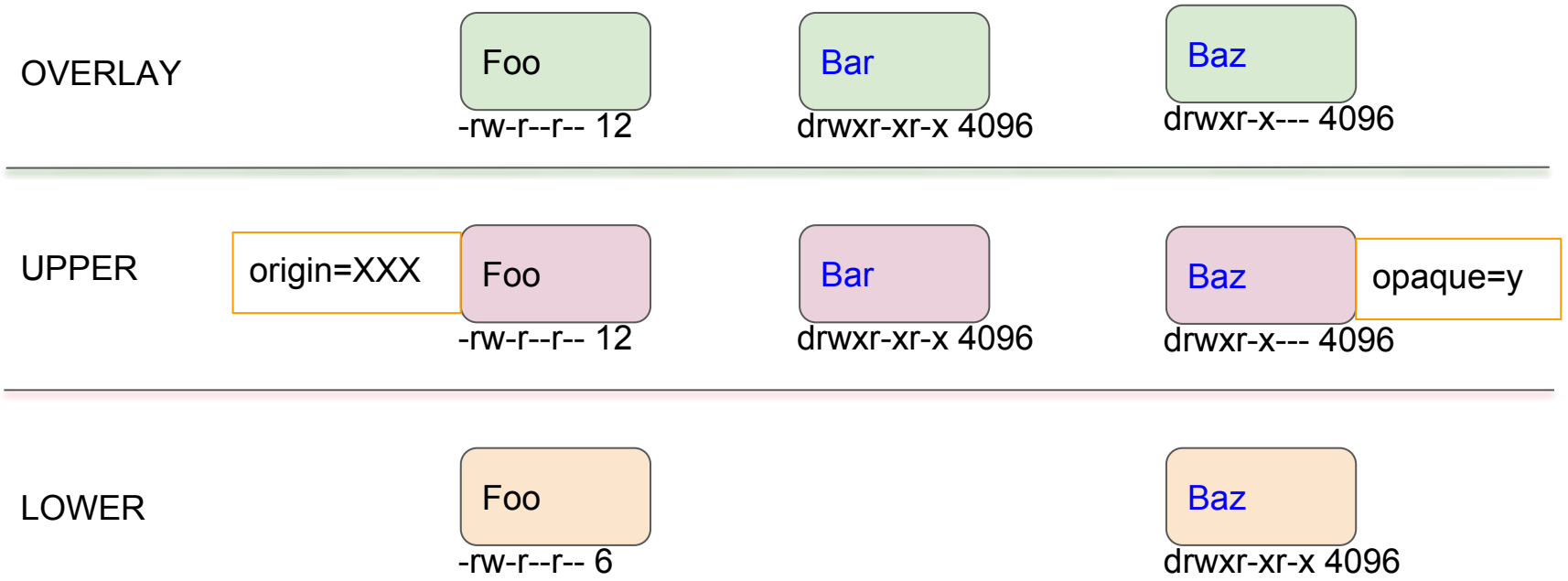
rmdir Baz


OVERLAY
Foo
-rw-r--r-- 6
Bar
drwxr-xr-x 4096
UPPER
Bar
drwxr-xr-x 4096
Baz
c--------- 0, 0
LOWER
Foo
-rw-r--r-- 6
Baz
drwxr-xr-x 4096

mkdir Baz


OVERLAY
Foo
-rw-r--r-- 6
Bar
drwxr-xr-x 4096
Baz
drwxr-x--- 4096
UPPER
Bar
drwxr-xr-x 4096
Baz
opaque=y
drwxr-x--- 4096
LOWER
Foo
-rw-r--r-- 6
Baz
drwxr-xr-x 4096

echo hello >> Foo



Whiteout files and extended attributes

Whiteout is a character device with 0/0 device number

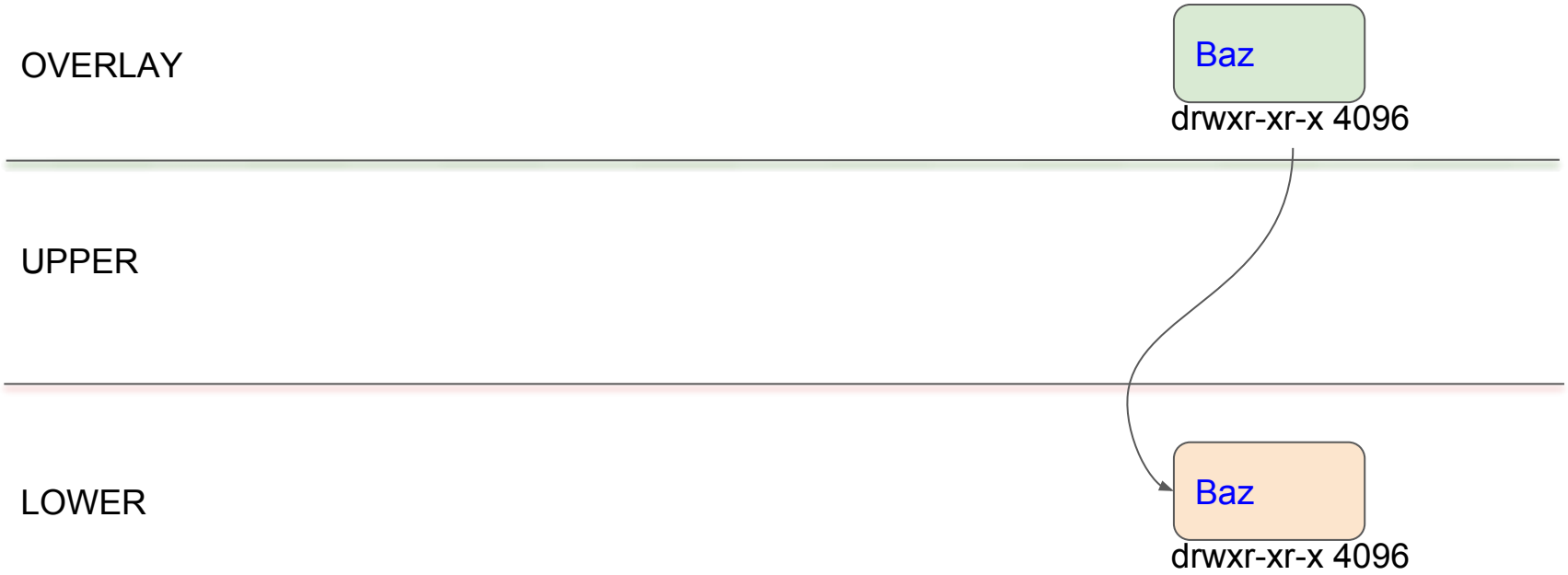
Everything else is represented with extended attributes

*trusted.overlay.****

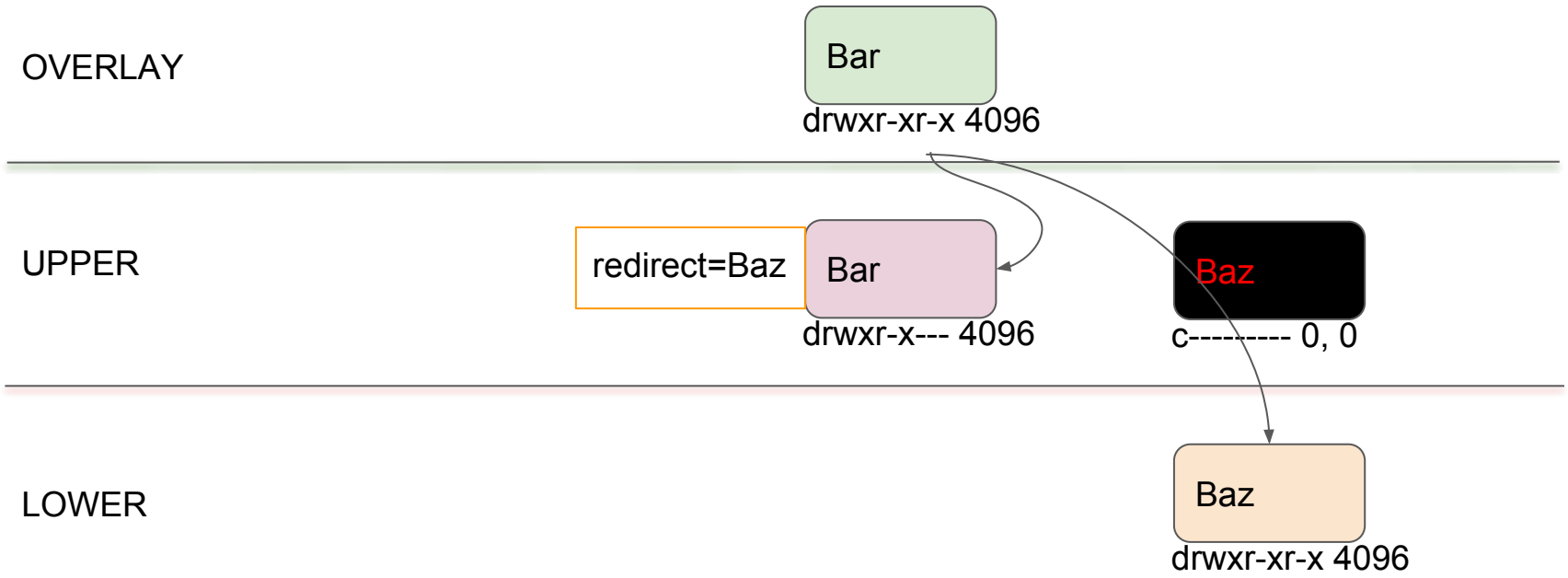
- *opaque*
- *redirect*
- *origin*
- *impure*
- *nlink*
- *upper*

Features

Renaming a directory



mv Baz Bar


OVERLAY
Bar
drwxr-xr-x 4096
UPPER
redirect=Baz
Bar
drwxr-x--- 4096
Baz
c--------- 0, 0
LOWER
Baz
drwxr-xr-x 4096

Renaming a directory

If ***redirect_dir*** feature is **disabled**, then rename will fail:

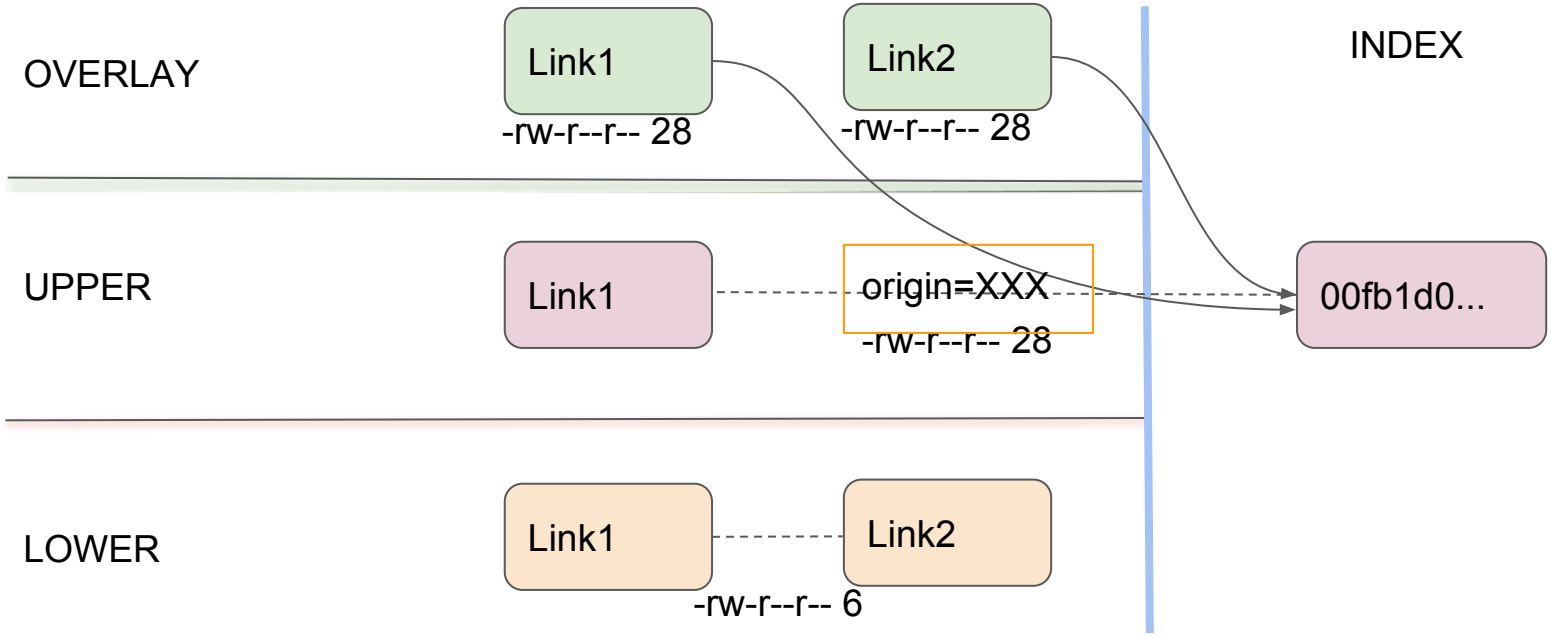
EXDEV -> userspace will often fall back on recursive copy

If ***redirect_dir*** feature is **enabled**, then rename can succeed

Possible security issue:

Can user fake a redirect to some file otherwise not accessible?

Copy up of hard link



Hard links

If ***index*** feature is **disabled**, then

copying up a hard link breaks association with lower object

If ***index*** feature is **enabled**, then

Index file is created

File handle of lower is encoded in index name

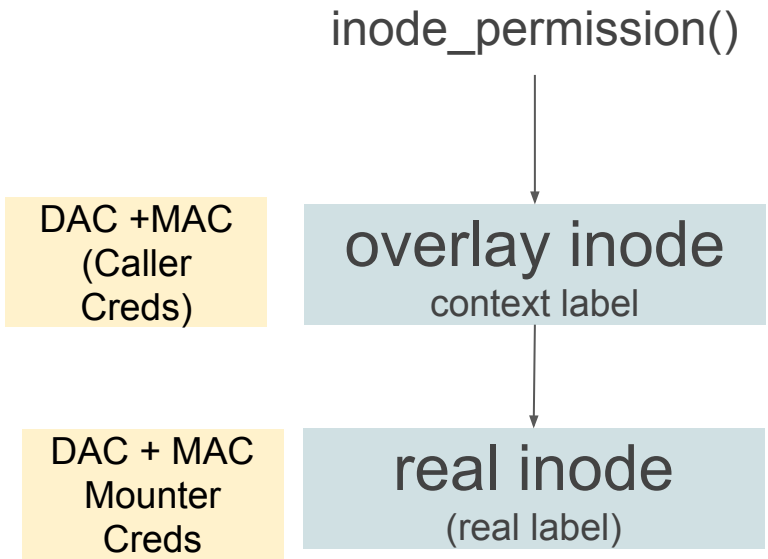
Location: “*workdir/index/*”

trusted.overlay.nlink keeps track of actual nlink

SELinux

Vivek Goyal

Permission checked on two levels:



Current work

NFS export

Need indexing directories and all files

- Directory index gets *trusted.overlay.upper* (cannot hard link directories)
- Index whiteout: don’t find lower file

Enabled with ***nfs_export*** and depends on ***index*** being enabled

Currently in linux-next, bound for 4.16

Metacopy

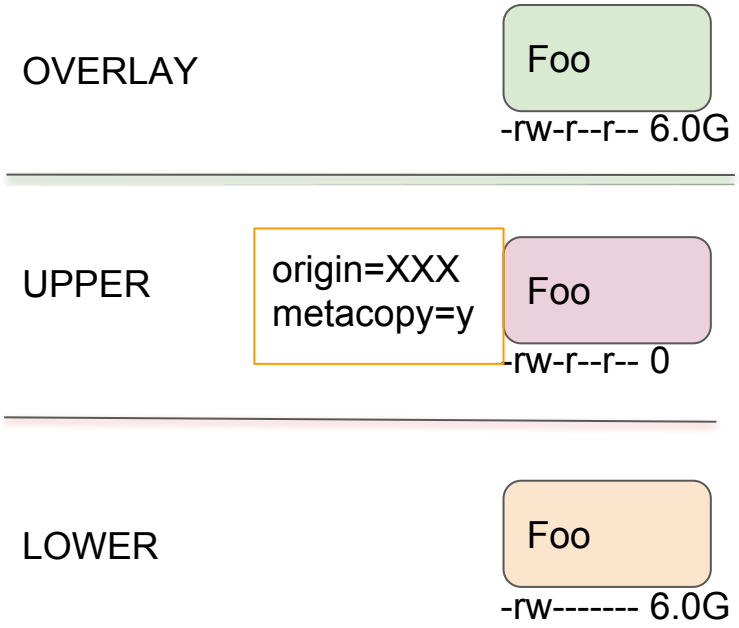
If only metadata is modified, don’t copy up data

E.g. “chmod +r largefile”

Helps if filesystem doesn’t support “clone”

-> 4.17?

Same security issues as “*redirect_dir*”



Future plans

RO/RW consistency

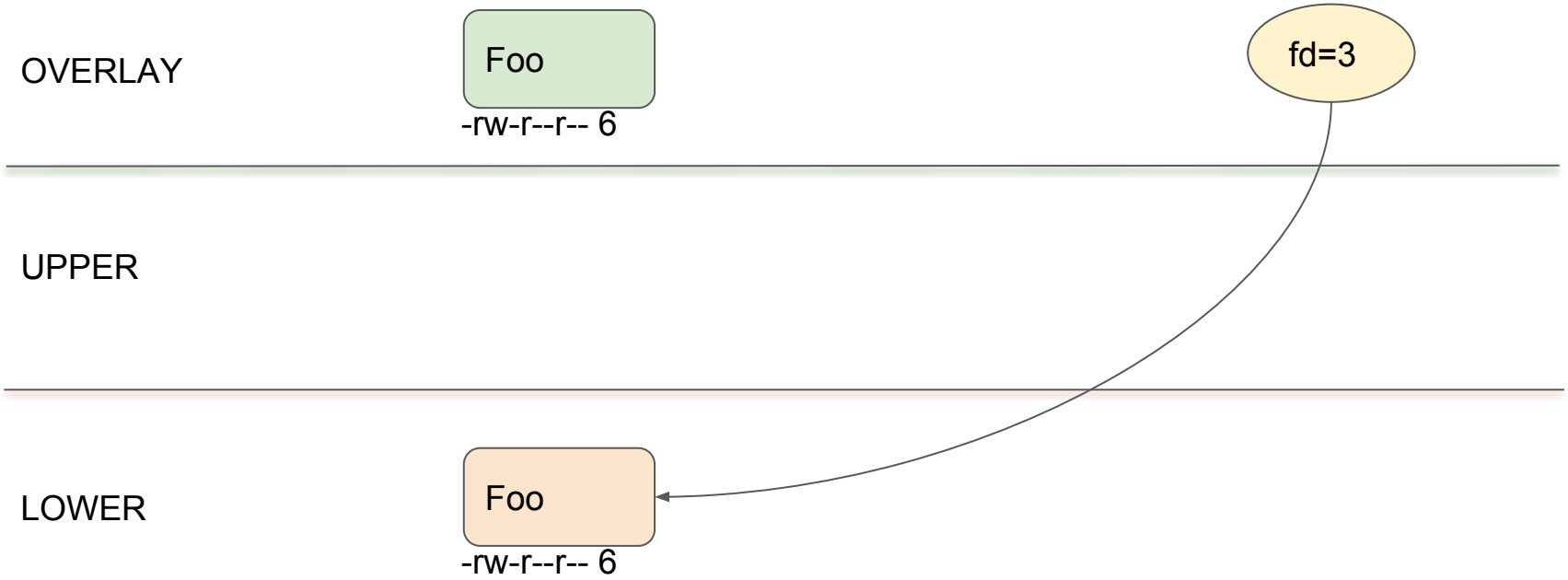
POSIX non-compliant behavior

Implementation detail

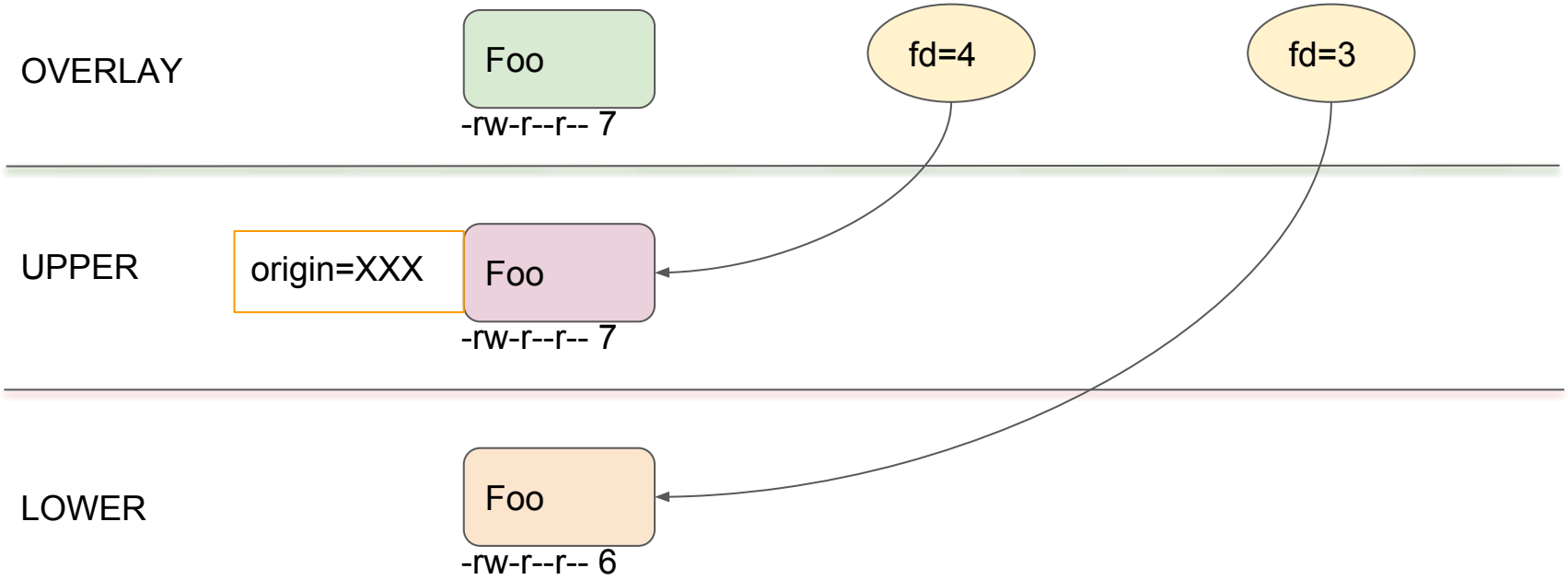
No worries about backward compatibility

But hard to fix without impacting performance, caching

open(“Foo”, O_RDONLY)



open(“Foo”, O_RDWR) + write(4, “goodbye”, 7)



User namespace support

Translation of uid/gid of lower fs:

- “chmod -R” -> this works, but everything is copied up
- Speed up “chmod -R” with metacopy?
- Mapping done inside overlayfs?
- Mapping done by VFS?

Unprivileged mounts

Need to be careful to prevent privilege escalation

redirect, metacopy: following untrusted “links” can lead to problems

Maybe not a problem in user namespace, but need to carefully review all posibilities

Major contributors

Amir Goldstein (CTERA Networks)

Vivek Goyal (Red Hat)

David Howells (Red Hat)

Al Viro (Red Hat)

Thank You!