GlusterFS as a Development Platform

(Tips For Writing a GlusterFS Translator)

Kaleb KEITHLEY
Red Hat
8 June, 2012

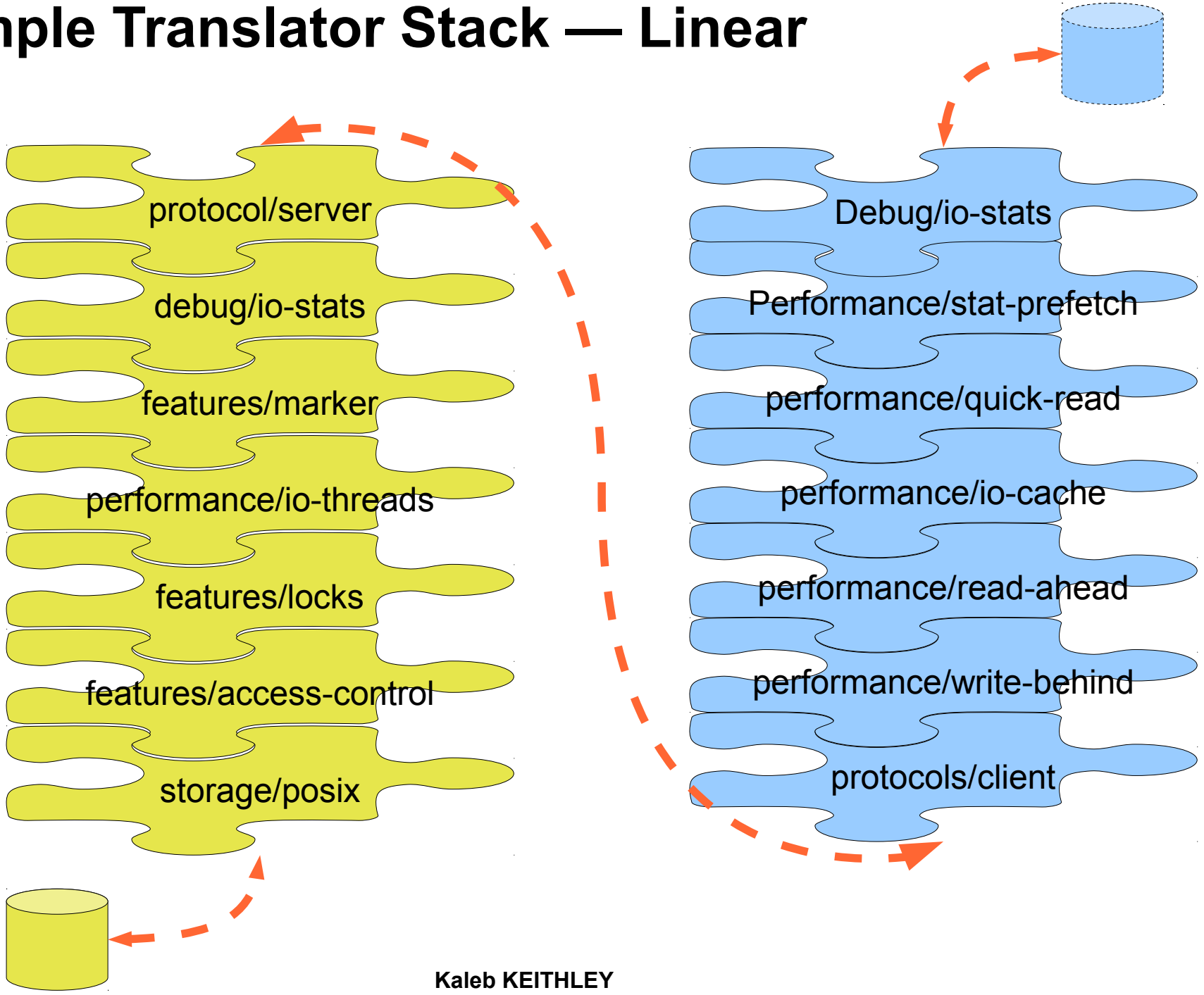
Extending GlusterFS with Translators

- What is a translator?
- An example: the HekaFS uidmap translator
- Translator basics
- Translator file-ops (fops) methods
- Inside fop methods — the nuts and bolts
- Building in-tree or out-of-tree
- Resources

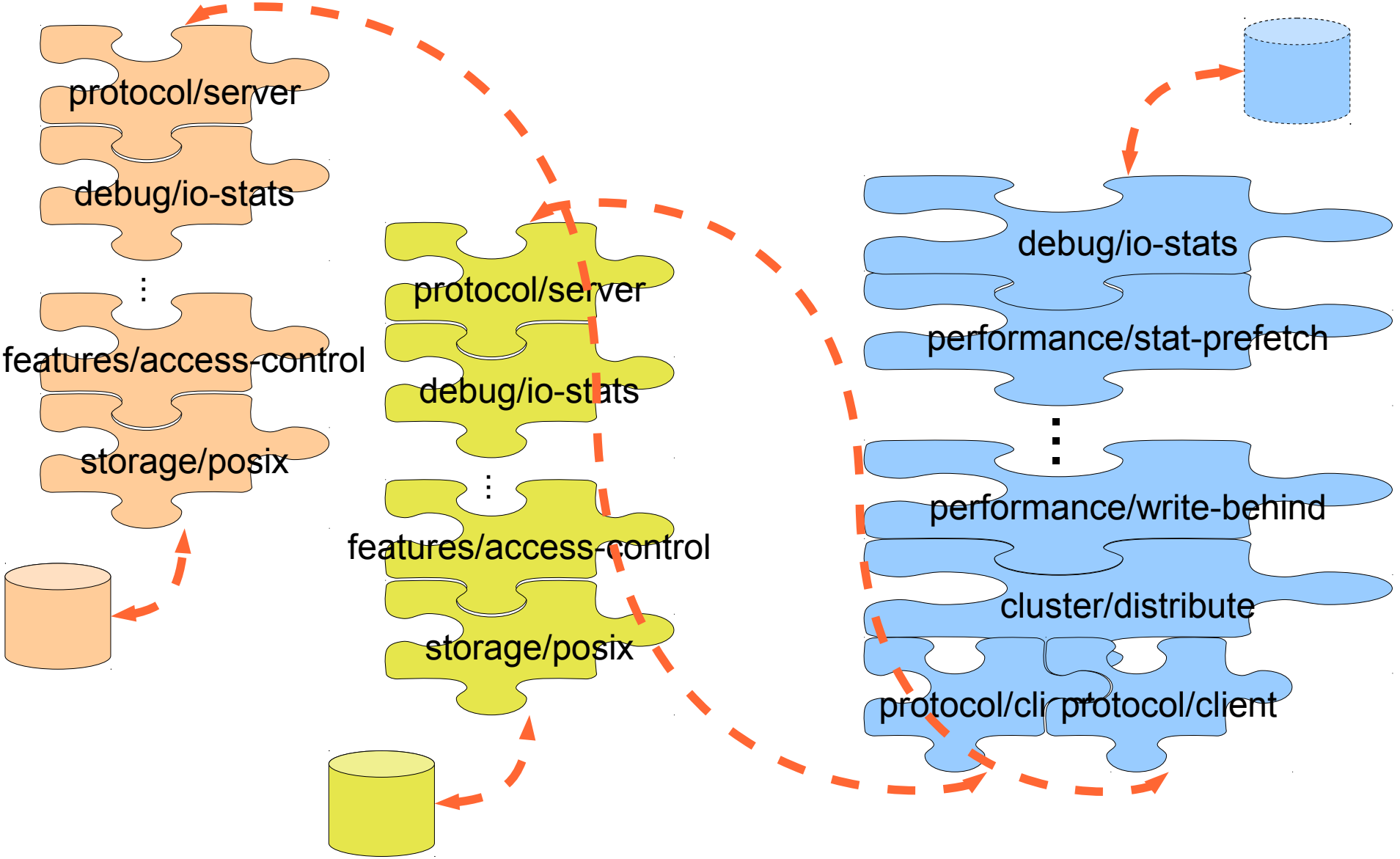
What is a GlusterFS translator

- Pluggable software component
- Provision Storage == Create a directed graph of translators
  - Linear, e.g. trivial one brick volume, NFS volume
  - Trees, e.g. Distribution, replication, stripe
- Translators are on both the server and on the client
  - Translators may be moved from client to server and vice versa
  - Every translator implements the same API and ABI

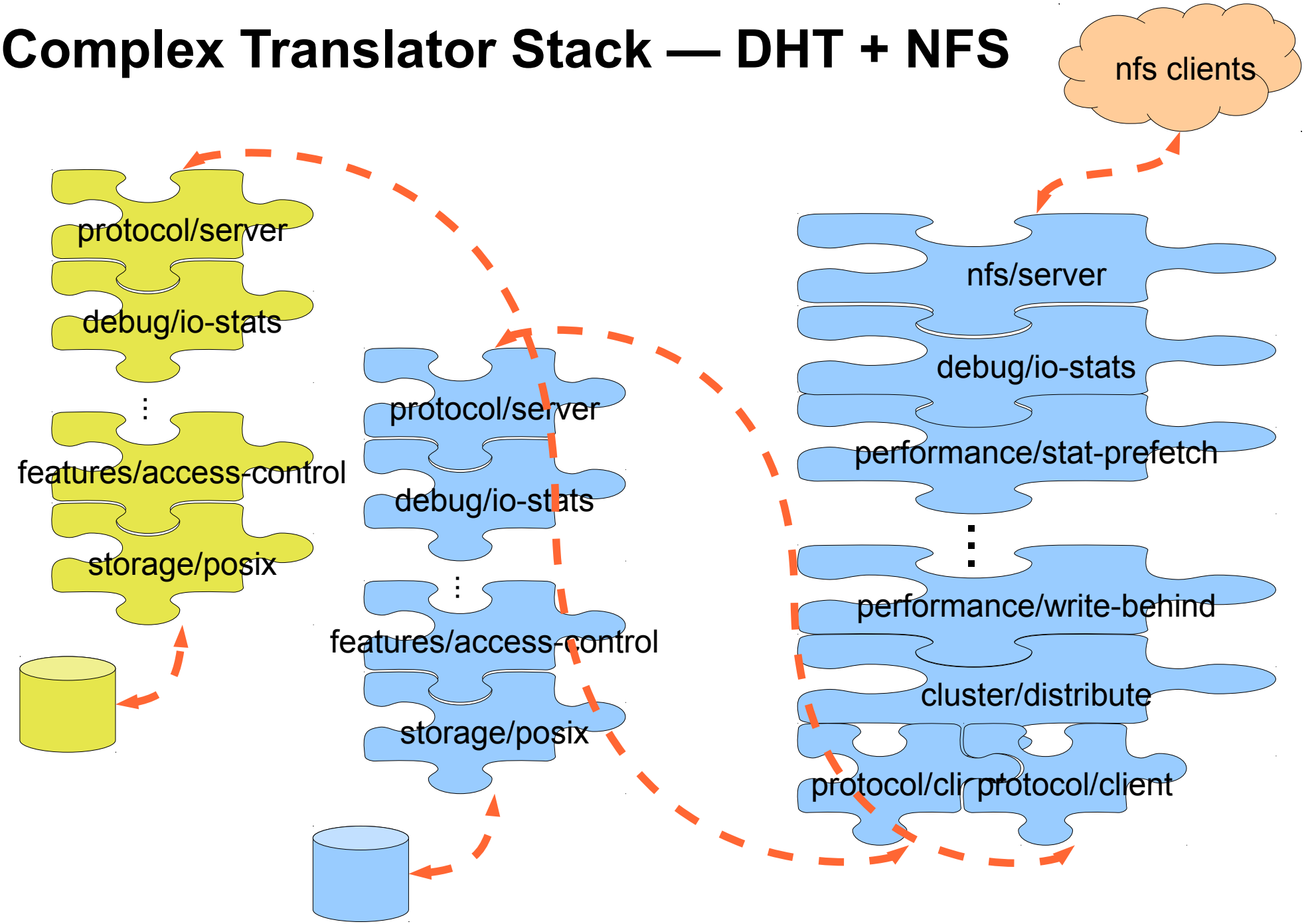
Simple Translator Stack — Linear



Complex Translator Stack — Distribute (DHT)



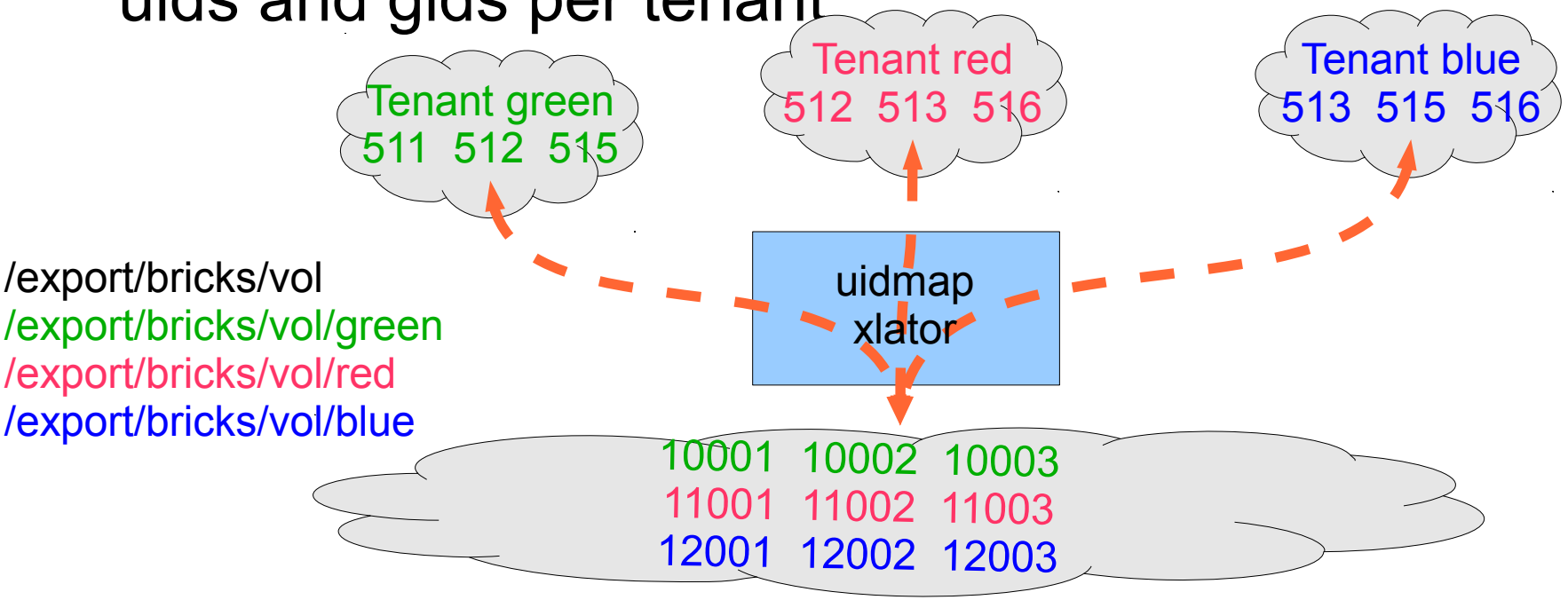
Complex Translator Stack — DHT + NFS



An example: HekaFS uidmap translator

- Consider a service provider with several customers
  - Each customer has thousands of users
  - Collisions in the uid and gid space

- Uidmap xlator maps uids and gids to discrete sets of uids and gids per tenant



Translator basics

- Translators are shared objects (shlibs)
  - Methods
    - int32_t init(xlator_t *this);
    - void fini(xlator_t *this);
  - Data
    - struct xlator_fops fops { ... };
    - struct xlator_cbks cbks { };
    - struct volume_options options [] = { ... };
- Client, Server, Client/Server
- Threads: write MT-SAFE
- Portability: GlusterFS != Linux only
- License: server GPLv3+, client GPLv2 or LGPLv3+

Volume Options

```
struct volume_options options[] = {
        { .key  = {"uidmap-plugin", "plugin"},
          .type = GF_OPTION_TYPE_STR,
        },
        { .key  = {"root-squash"},
          .type = GF_OPTION_TYPE_STR,
          .value = { "yes", "no"}
        },
        { .key  = {"uid-range"},
          .type = GF_OPTION_TYPE_STR,
        },
        { .key  = {"gid-range"},
          .type = GF_OPTION_TYPE_STR,
        },
        { .key  = {NULL} },
};
```

GF_OPTION_TYPE_{ANY,STR,INT,SIZET,PERCENT,BOOL,...}

Translator fops

- Here there be dragons (and they're not dragons of good fortune)
  - Every fop method has a different signature
  - Signatures change from release to release
  - Documentation? :-(

- The nuts and bolts
  - fop methods and fop callbacks
  - STACK_WIND(), STACK_UNWIND(), and friends
  - Calling multiple “children”
  - Dealing with errors
    - Indicating an I/O error
    - Indicating a Method error

Every method has a different signature

- Open fop method and callback

  typedef int32_t (*fop_open_t) (call_frame_t *, xlator_t *, loc_t *, int32_t, fd_t *, dict_t *);

  typedef int32_t (*fop_open_cbk_t) (call_frame_t *, void *, xlator_t *, int32_t, int32_t, fd_t *, dict_t *);

- Rename fop method and callback

  typedef int32_t (*fop_rename_t) (call_frame_t *, xlator_t *, loc_t *, loc_t *, dict_t *);

  typedef int32_t (*fop_rename_cbk_t) (call_frame_t , void *, xlator_t *, int32_t, int32_t, struct iatt *, struct iatt *, struct iatt *, struct iatt *, struct iatt *);

Method signatures change from release to release

- 3.2 rename fop

  typedef int32_t (*fop_open_t) (call_frame_t *, xlator_t *, loc_t *, int32_t, fd_t *, int32_t);

- 3.3 rename fop

  typedef int32_t (*fop_open_t) (call_frame_t *, xlator_t *, loc_t *, int32_t, fd_t *, dict_t *);

Translator Data Types

- call_frame_t —
- xlator_t — translator context
- inode_t — represents a file on disk; ref-counted
- fd_t — represents an open file; ref-counted
- iatt_t — ~= struct stat
- dict_t — ~= Python dict (or C++ std::map)

Utility Functions

- Memory Management
  - GF_MALLOC, GF_CALLOC, GF_FREE
- Logging
  - gf_log, gf_print_trace
- Red-black trees, hashes, etc

fop methods and fop callbacks

```
uidmap_writev (...)
{
        ...
        STACK_WIND (frame, uidmap_writev_cbk,
                           FIRST_CHILD (this),
                           FIRST_CHILD (this)->fops->writev,
                           fd, vector, count, offset, iobref);
        /* DANGER ZONE */
        return 0;
}
```

- Effectively lost control after STACK_WIND
  - Callback might have already happened
  - Or might be running right now
  - Or maybe it's not going to run 'til later

fop methods and fop callback methods, cont.

```
uidmap_writev_cbk (call_frame_t *frame, void *cookie, ...)
{
        ...
        STACK_UNWIND_STRICT (writev, frame
                op_ret, op_errno, prebuf, postbuf);
        return 0;
}
```

- The I/O is complete when the callback is called

STACK_WIND versus STACK_WIND_COOKIE

- Pass extra data to the cbk with STACK_WIND_COOKIE

```
quota_statfs (call_frame_t *frame,
        xlator_t *this, loc_t *loc)
{
        inode_t *root_inode = loc->inode->table->root;
        STACK_WIND_COOKIE (frame, quota_statfs_cbk,
                root_inode, FIRST_CHILD (this),
                FIRST_CHILD (this)->fops->statfs, loc, xdata);
        return 0;
}
```

- There is also frame->local
  - shared by all STACK_WIND callbacks

STACK_WIND, STACK_WIND_COOKIE, cont.

- Pass extra data to the cbk with STACK_WIND_COOKIE

```
quota_statfs_cbk (call_frame_t *frame, void *cookie, ...)
{
        inode_t *root_inode = cookie;
        ...
}
```


STACK_UNWIND versus STACK_UNWIND_STRICT

- STACK_UNWIND_STRICT uses the correct type

```
/* return from function in a type-safe way */

#define STACK_UNWIND (frame, params ...)

        do {

                ret_fn_t fn = frame->ret;

        ...
```

versus

```
#define STACK_UNWIND_STRICT (op, frame, params ...)

        do {

                fop_##op##_cbk_t fn = (fop_##op##_cbk_t)frame->ret;

        ...
```

- And why wouldn't you want strong typing?

Calling multiple children (fan out)

```
afr_writev_wind (...)
{
        ...
        for (i = 0; i < priv->child_count; i++) {
                if (local->transaction.pre_op[i]) {
                        STACK_WIND_COOKIE (frame, afr_writev_wind_cbk,
                                           (void *) (long) i,
                                           priv->children[i],
                                           priv->children[i]->fops->writev,
                                           local->fd, ...);
                }
        }
        return 0;
}
```


Calling multiple children, cont. (fan in)

```
afr_writev_wind_cbk (...)
{
        LOCK (&frame->lock);
        callcnt = --local->call_count;
        UNLOCK (&frame->lock);
        if (callcnt == 0) /* we're done */
        ...
}
```

- failure by any one child means the whole transaction failed
  - And needs to be handled accordingly

Dealing With Errors: I/O errors

```
uidmap_writev_cbk (call_frame_t *frame, void *cookie,
        xlator_t *this, int32_t op_ret, int32_t op_errno, ...)
{
        ...
        STACK_UNWIND_STRICT (writev, frame, -1, EIO, ...);
        return 0;
}
```

- op_ret: 0 or -1, success or failure
- op_errno: from <errno.h>
  - Use an op_errno that's valid and/or relevant for the fop

Dealing With Errors: method errors

```
uidmap_writev (call_frame_t *frame, xlator_t *this, ...)
{
        ...
        if (horrible_logic_error_must_abort) {
                goto error; /* glusterfs idiom */
        }
        STACK_WIND(frame, uid_writev_cbk, ...);
        return 0;
error:
        STACK_UNWIND_STRICT (writev, frame, -1, EIO, NULL, NULL);
        return 0;
}
```


Building: in-tree or out-of-tree

- In-tree: Gluster.org source hasn't had a -devel package.
  - Good news: starting with 3.3.0 there is now a -devel package if you build the RPM from the glusterfs.spec(.in) file included in the source
  - Bad news: unclear what .deb packagers are doing
- Fedora RPMs have had a -devel package for 3.2.x
  - Use HekaFS sources as a model for building out-of-tree

Resources

- Jeff Darcy's HekaFS.org Translator tutorials
  - http://hekafs.org/index.php/2011/11/translator-101-class-1-setting-the-stage/
  - http://hekafs.org/index.php/2011/11/translator-101-lesson-2-init-fini-and-private-co
  - http://hekafs.org/index.php/2011/11/translator-101-lesson-3-this-time-for-real/
  - http://hekafs.org/index.php/2011/11/translator-101-lesson-4-debugging-a-translato
- GlusterFS documentation
  - http://www.gluster.org/community/documentation/index.php/Main_Page
- GlusterFS Git repos
  - https://github.com/gluster/glusterfs
  - ssh://git.gluster.com/glusterfs.git
- This presentation http://www.fedorapeople.org/kkeithle/LinuxCon-Gluster.odp
- My email mailto:kkeithle@redhat.com

Call to action

- Go forth and write GlusterFS translators!