Gluster Technical Overview

Kaleb KEITHLEY
Red Hat
11 April, 2015

Concepts:

- Translator — a shared library that implements storage semantics
  - e.g. distribution, replication, performance, etc.
- Translator Stack — a directed graph of translators
- Brick — a server and a directory
- Volume — a collection of bricks

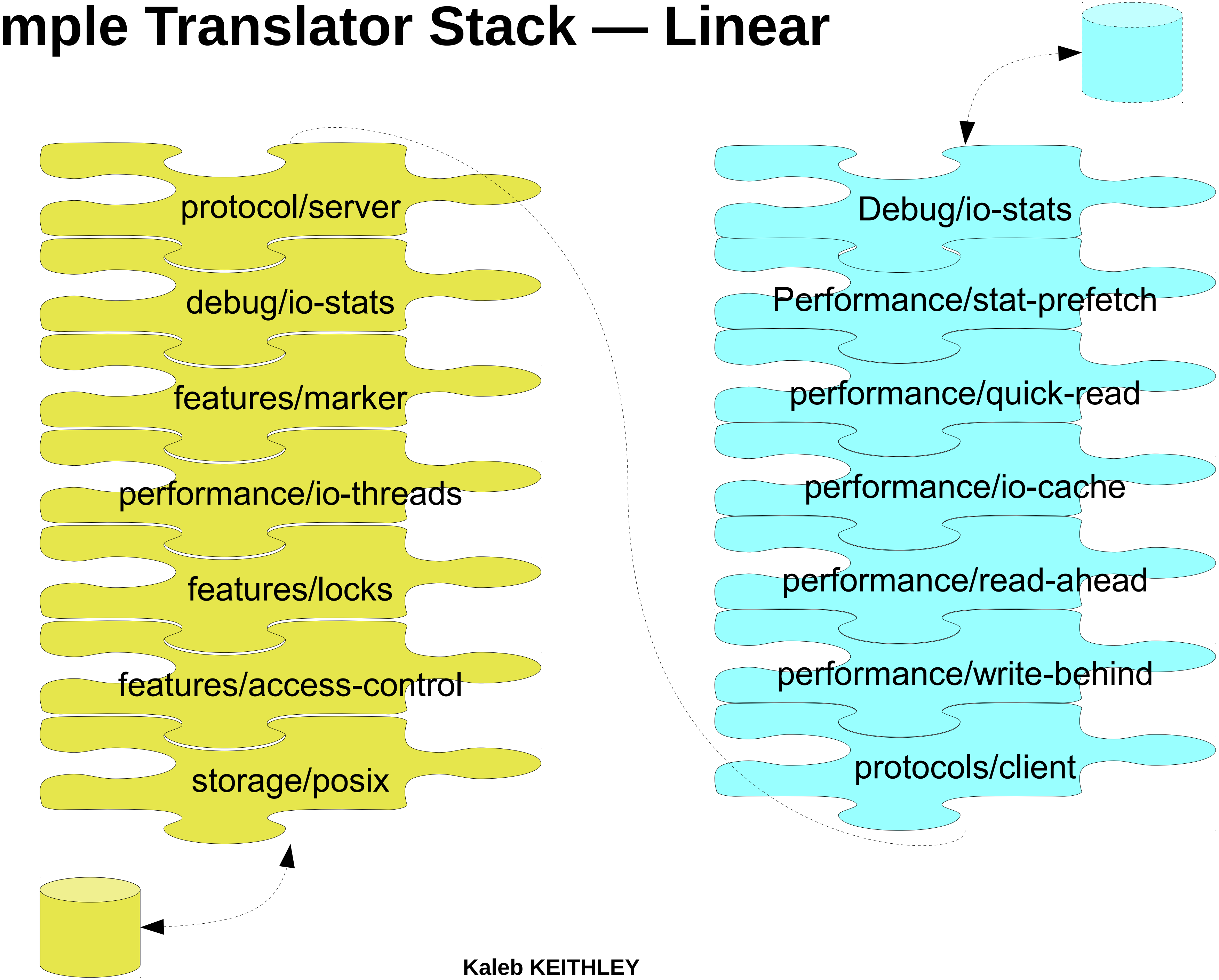
Gluster Processes

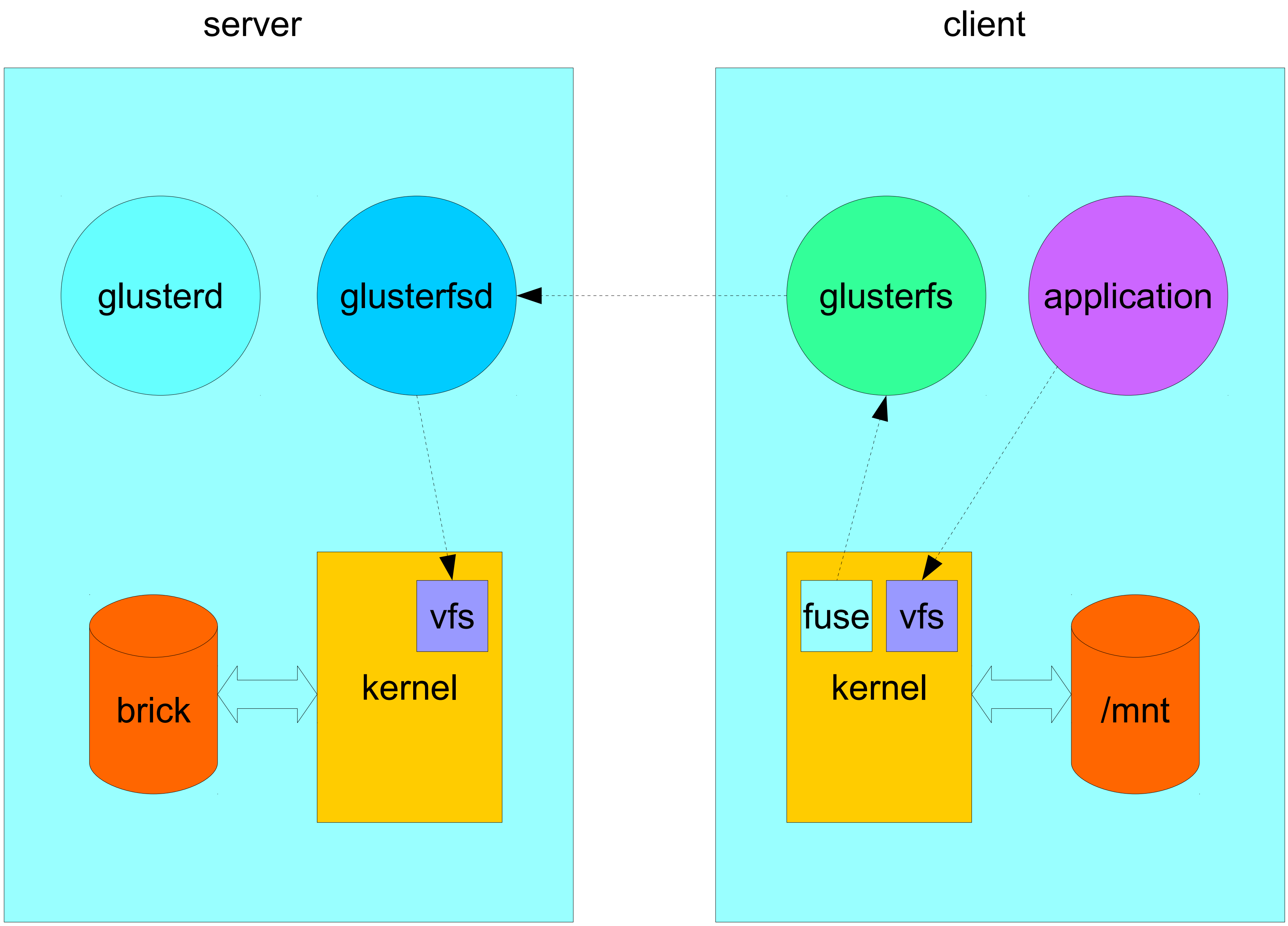
- Server: glusterfsd — the heart of it all
- Clients:
  - FUSE, glusterfs
  - NFS server, glusterfs (gluster client, NFS server)
- Management: glusterd
- Healing: glustershd
- And more——

Let's get real — creating a volume

```
Srv1% gluster peer probe srv1

...

srv1% gluster volume create my_vol
srv1:/bricks/my_vol

srv1% gluster volume start my_vol


…

client1% mount -t glusterfs srv1:my_vol
/mnt
```

Simple Translator Stack — Linear

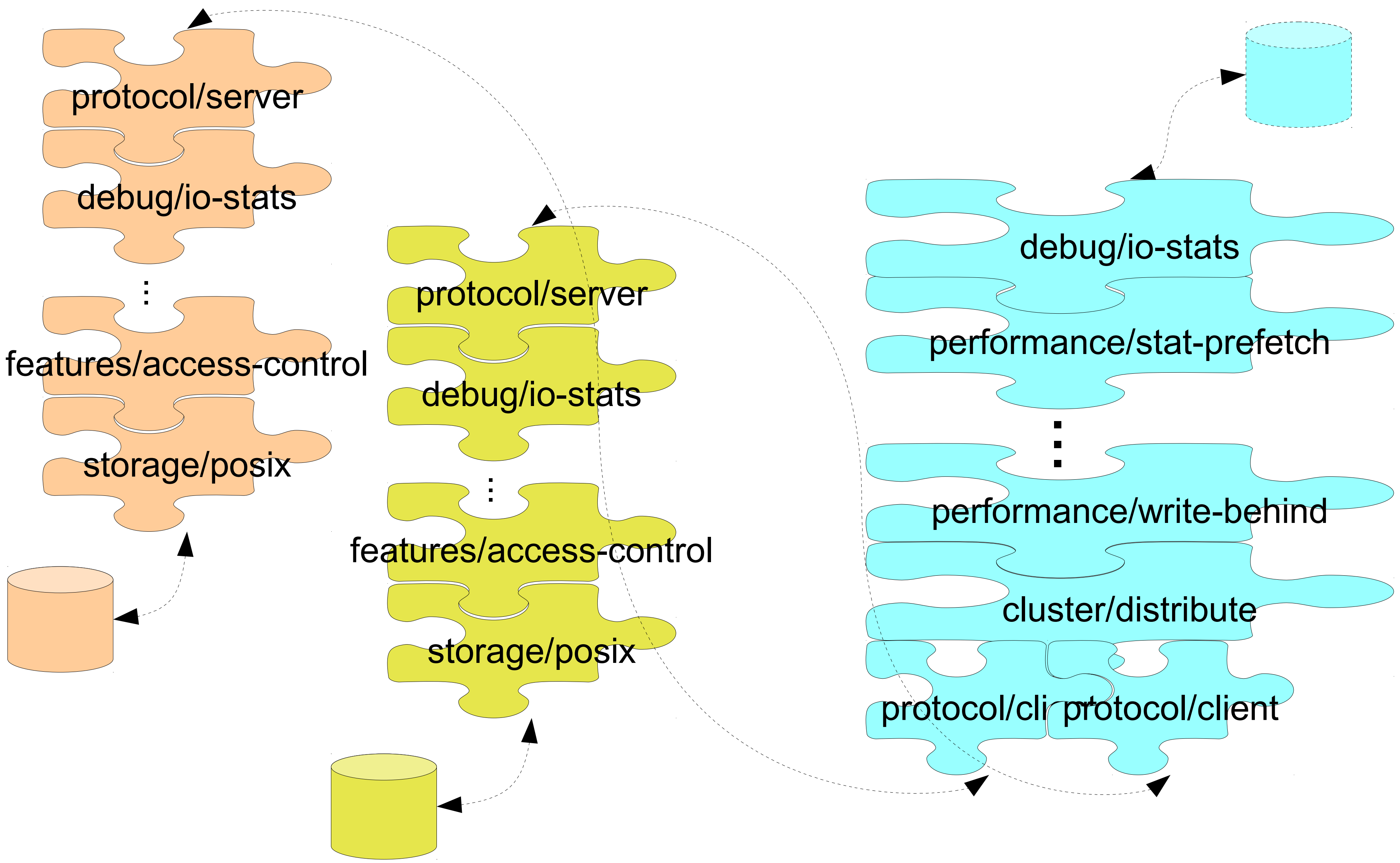


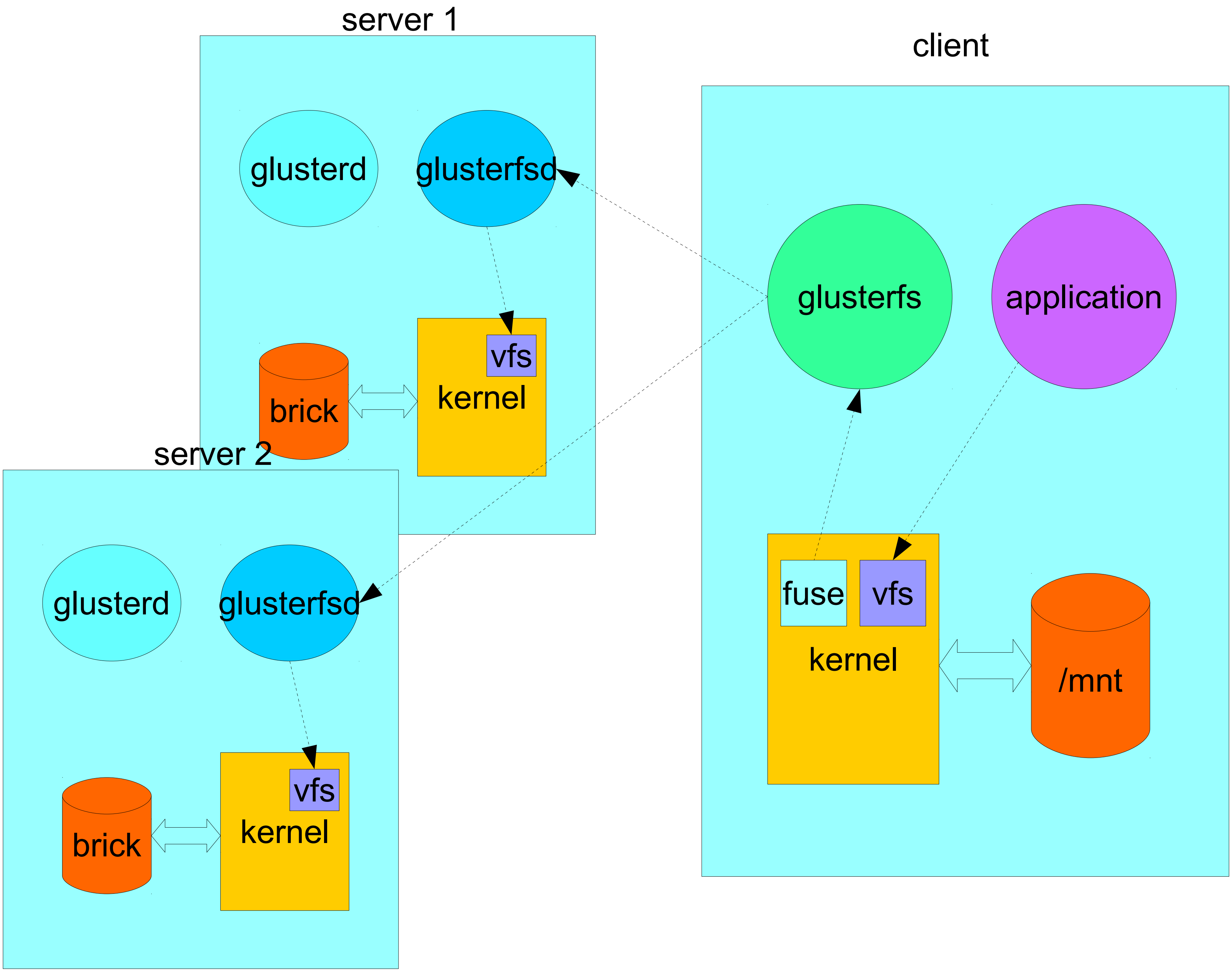

server
client
glusterd
glusterfsd
glusterfs
application
vfs
kernel
brick
fuse
vfs
kernel
/mnt

Let's get real — creating a distribute volume

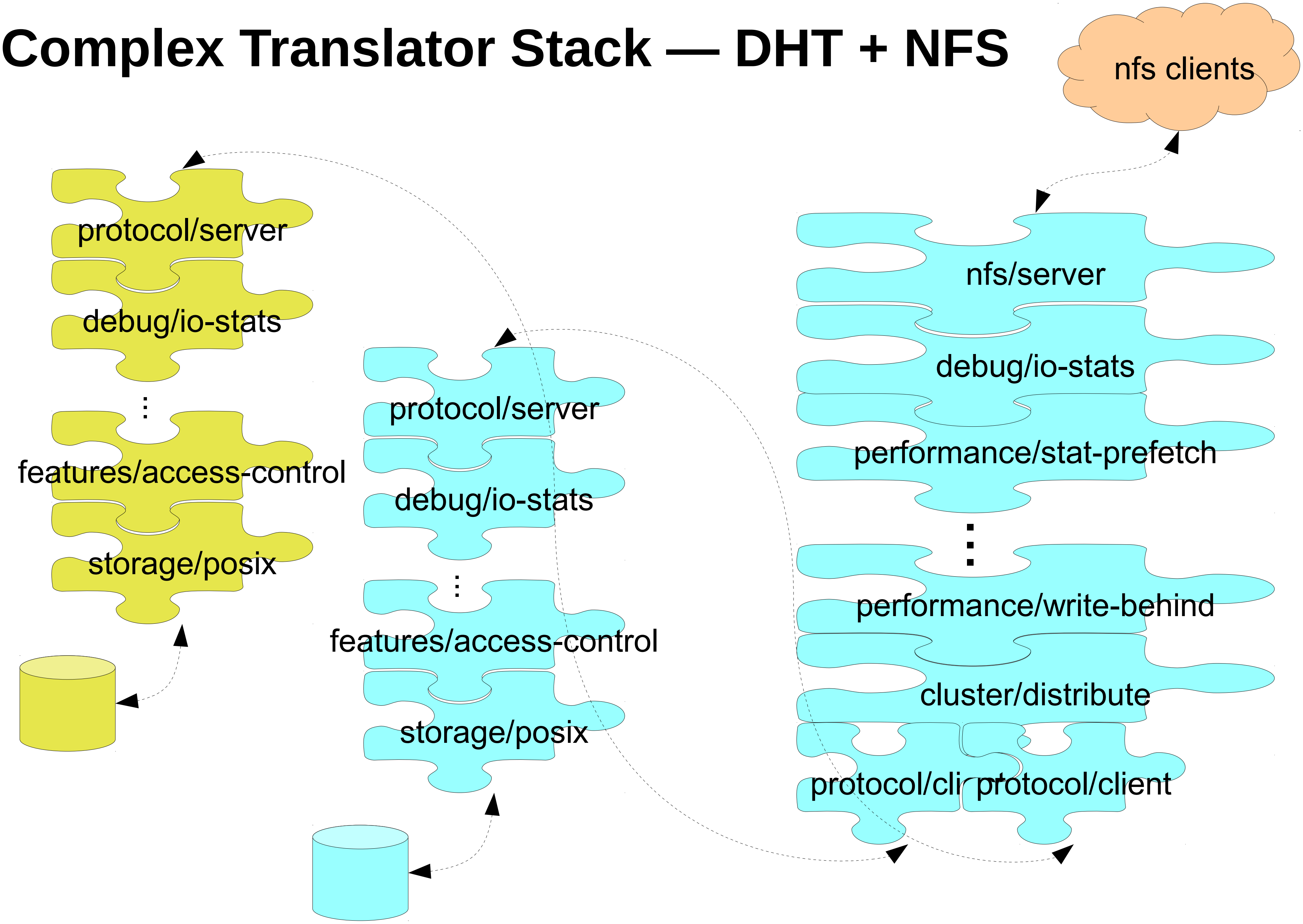
```
srv1% gluster volume create my_dist
srv1:/bricks/my_vol srv2:/bricks/my_vol

srv1% gluster volume start my_dist


...

client1% mount -t glusterfs srv1:my_dist /mnt
```

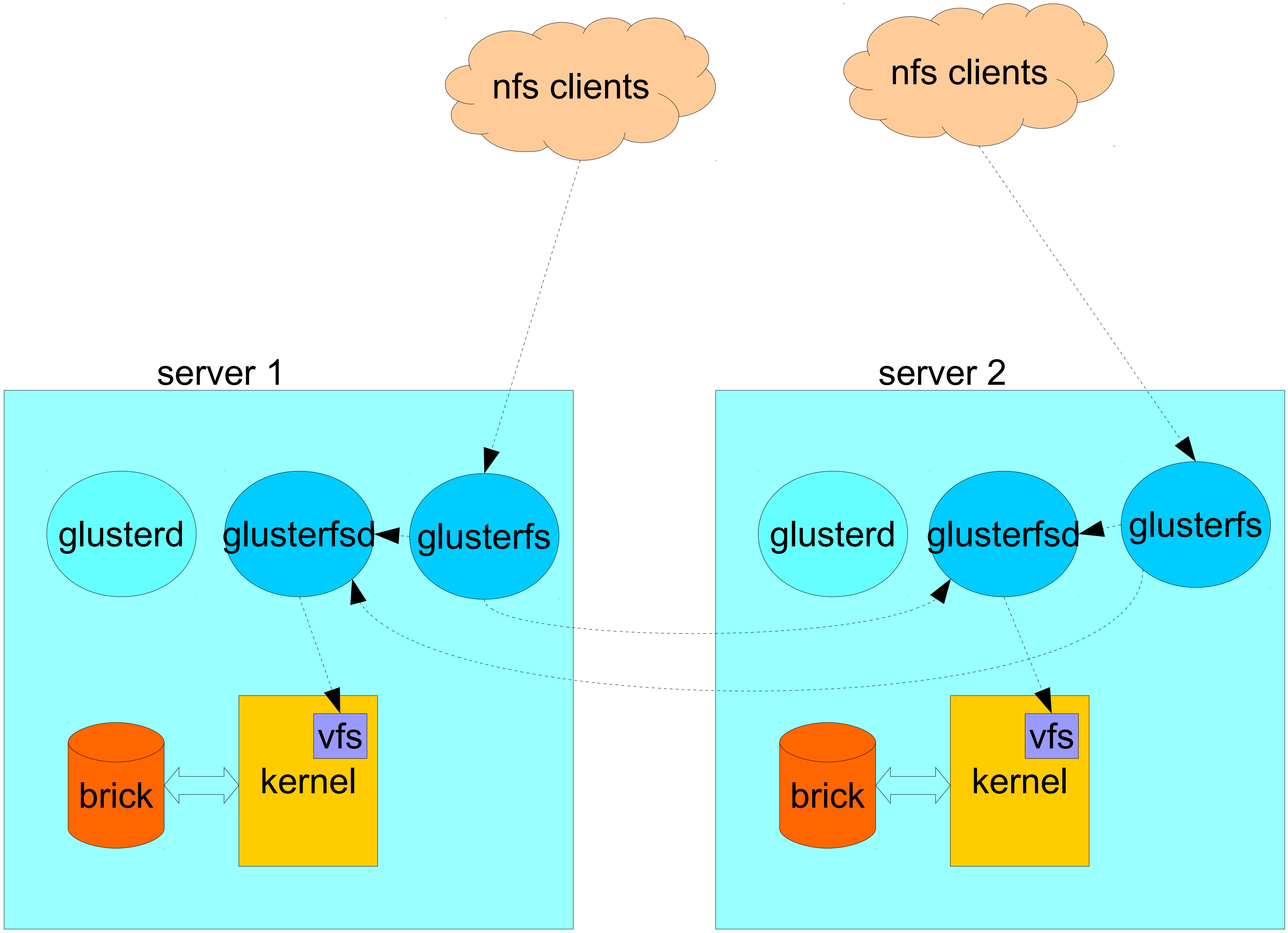
Complex Translator Stack — Distribute (DHT)




server 1
glusterd
glusterfsd
vfs
brick
kernel
client
glusterfs
application
server 2
glusterd
glusterfsd
vfs
brick
kernel
fuse
vfs
kernel
/mnt

Complex Translator Stack — DHT + NFS

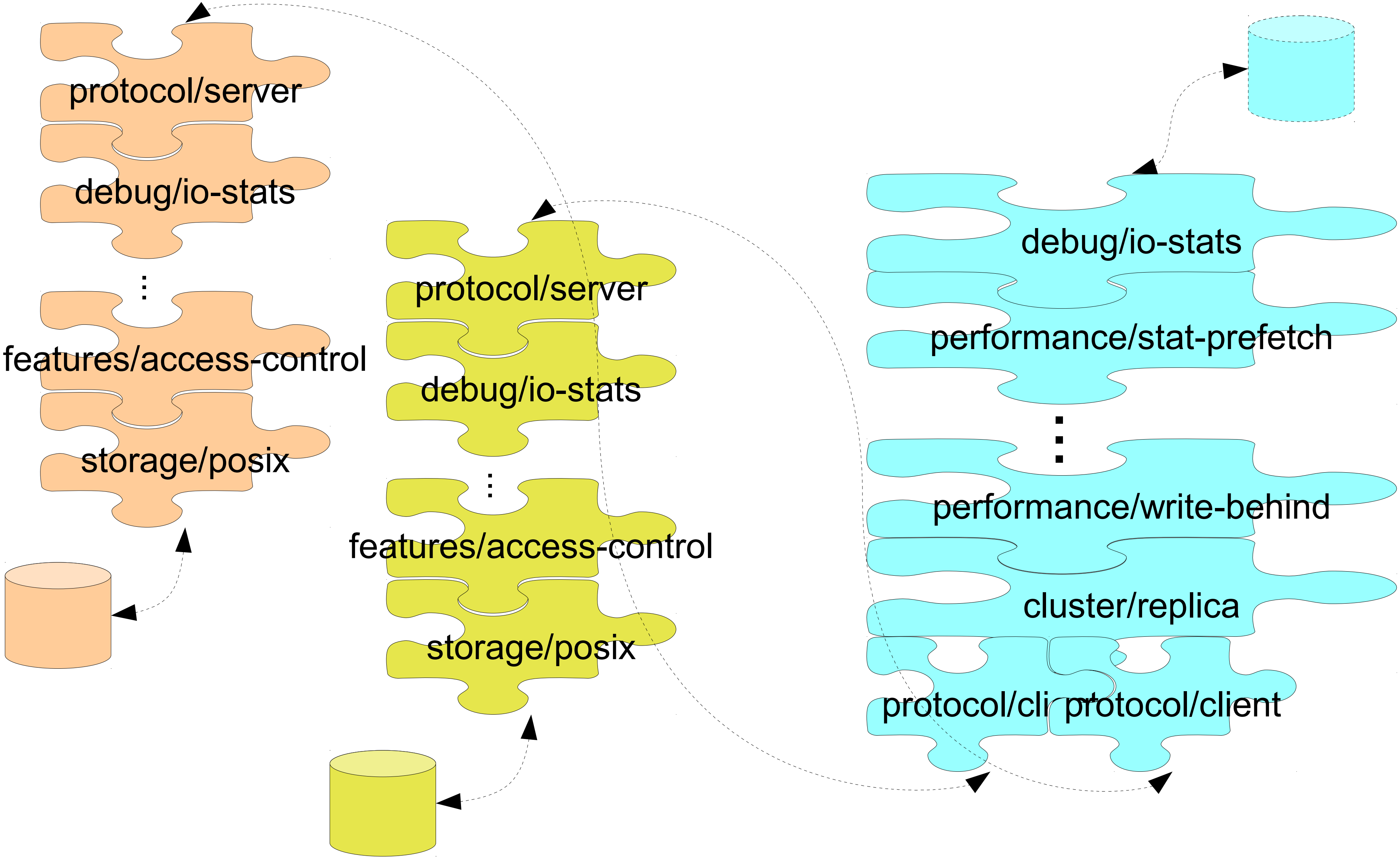


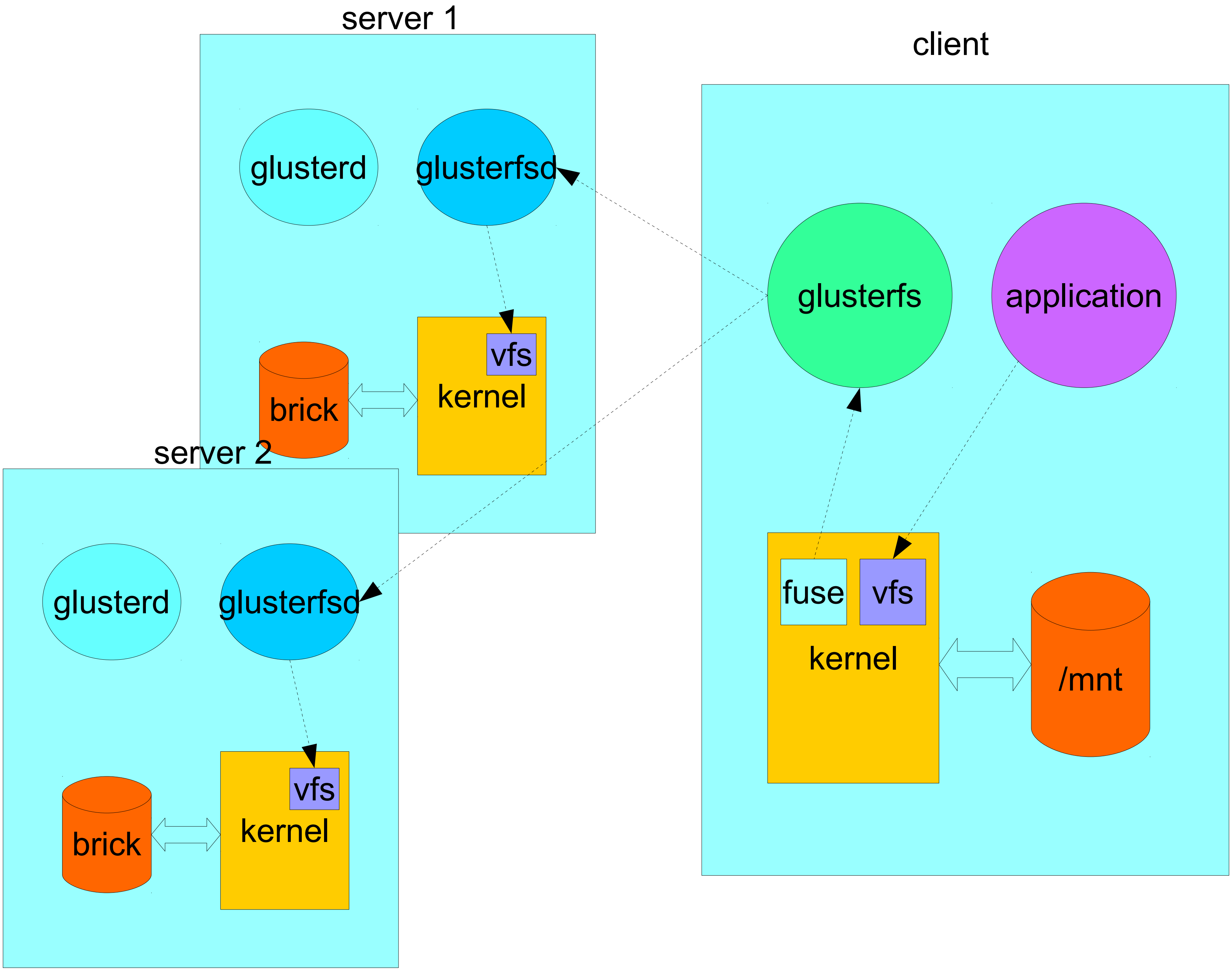

nfs clients
nfs clients
server 1
server 2
glusterd
glusterfsd
glusterfs
glusterd
glusterfsd
glusterfs
vfs
kernel
brick
vfs
kernel
brick

Let's get real — creating a replica volume

```
srv1% gluster volume create replica 2 my_repl
srv1:/bricks/my_vol srv2:/bricks/my_vol

srv1% gluster volume start my_repl


…

client1% mount -t glusterfs srv1:my_repl /mnt
```

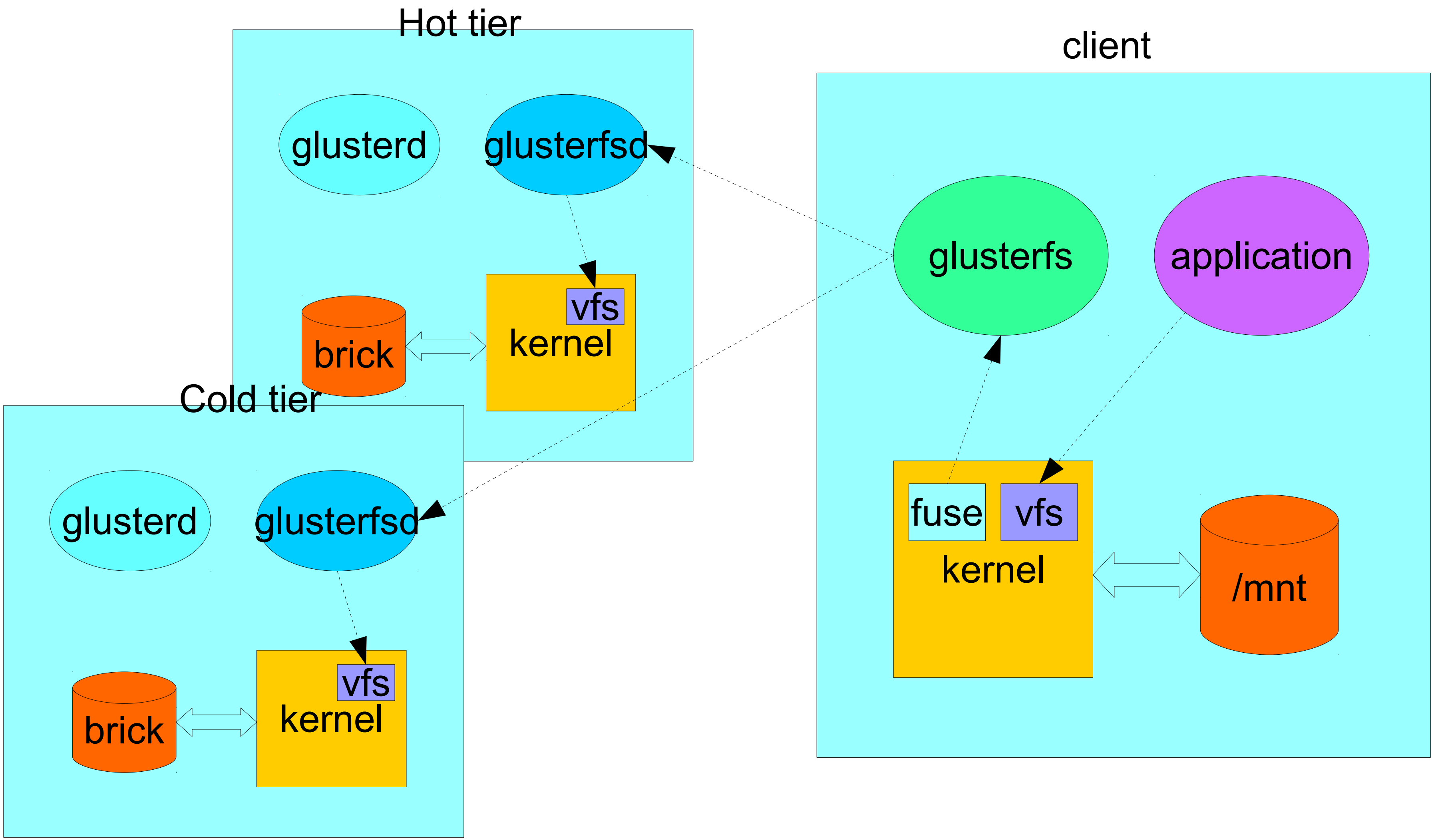
Complex Translator Stack — Replica (AFR)



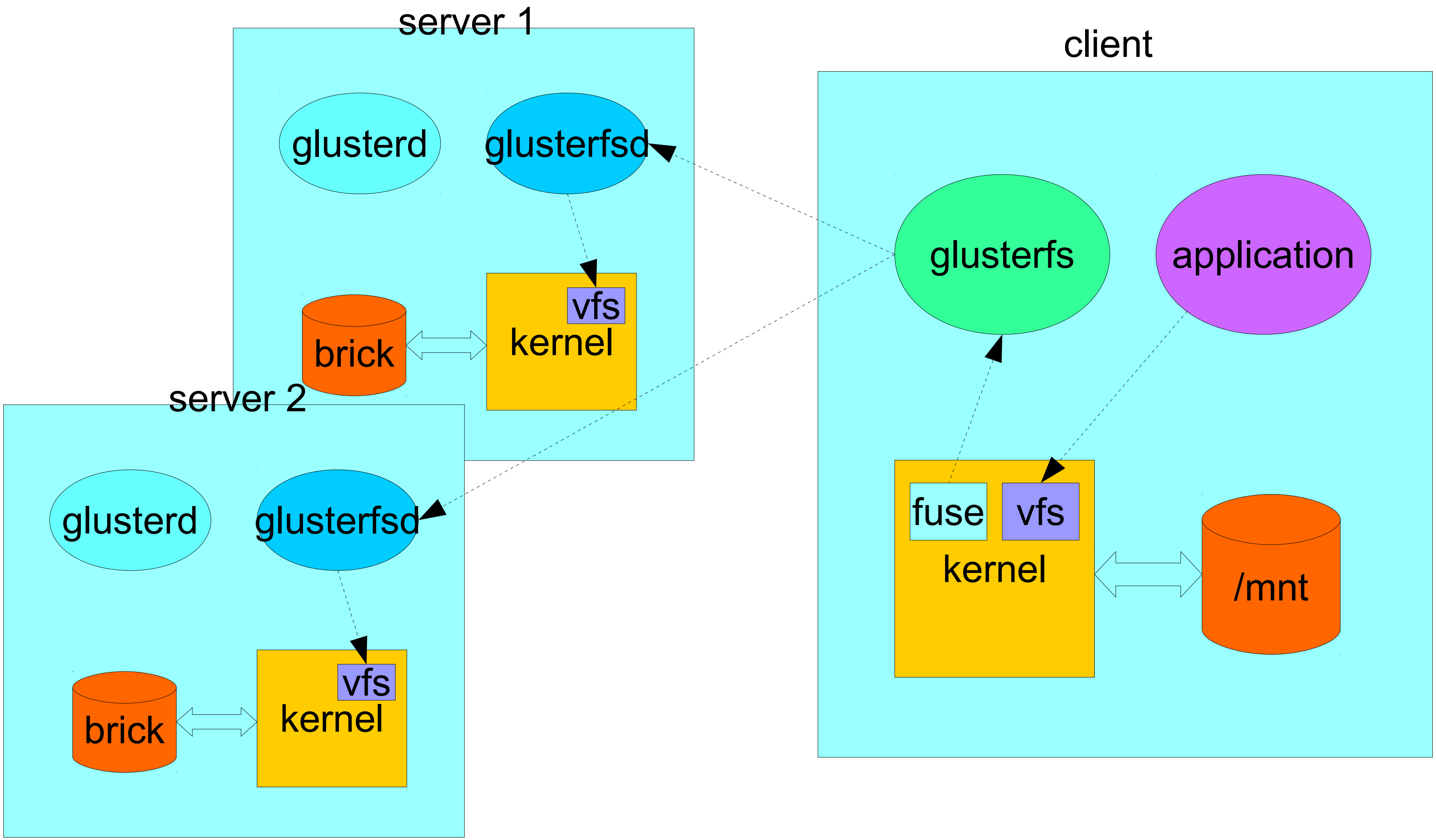

server 1
glusterd
glusterfsd
vfs
brick
kernel
server 2
glusterd
glusterfsd
vfs
brick
kernel
client
glusterfs
application
fuse
vfs
kernel
/mnt

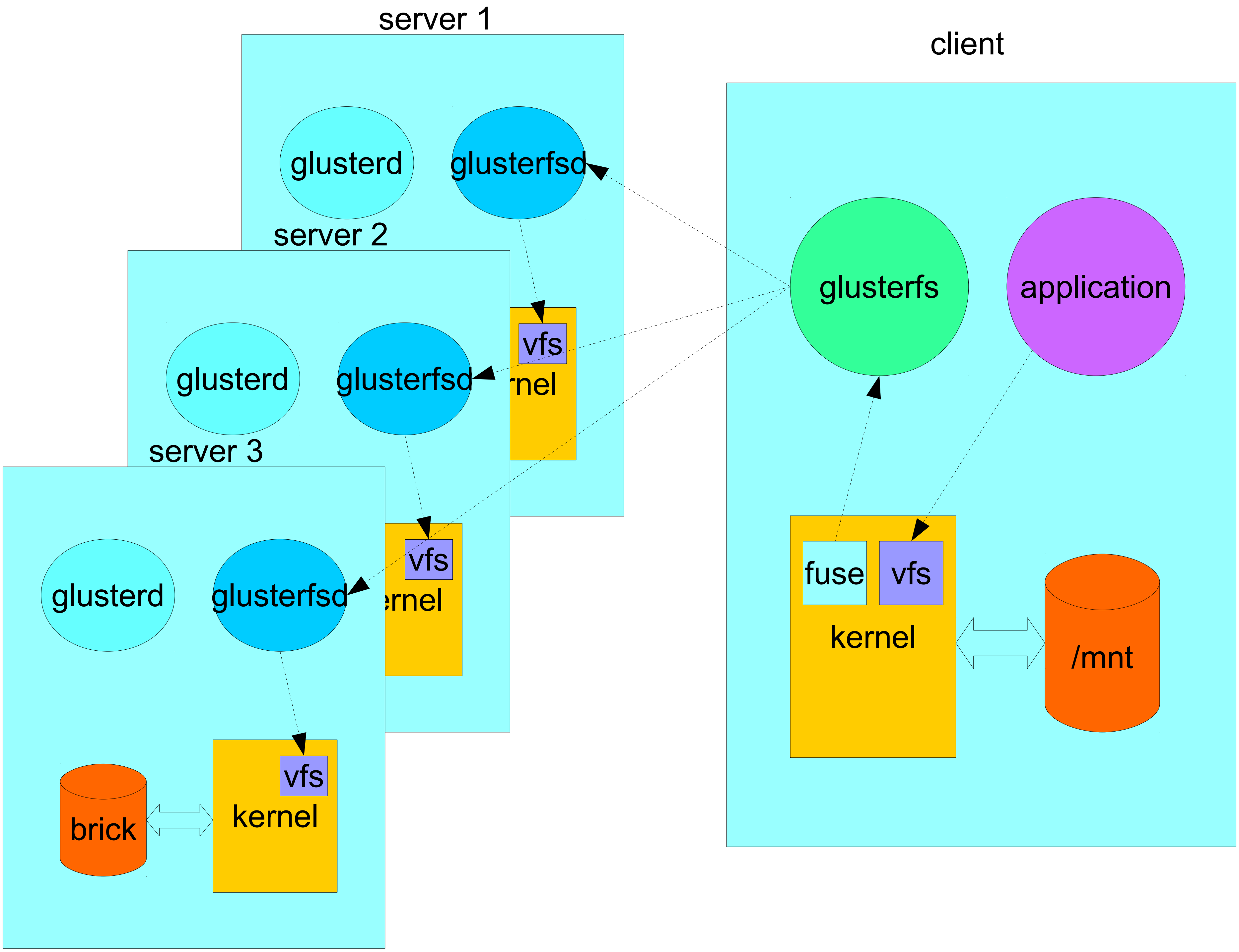
Tiering: a variation on distribution

- Hot and cold data

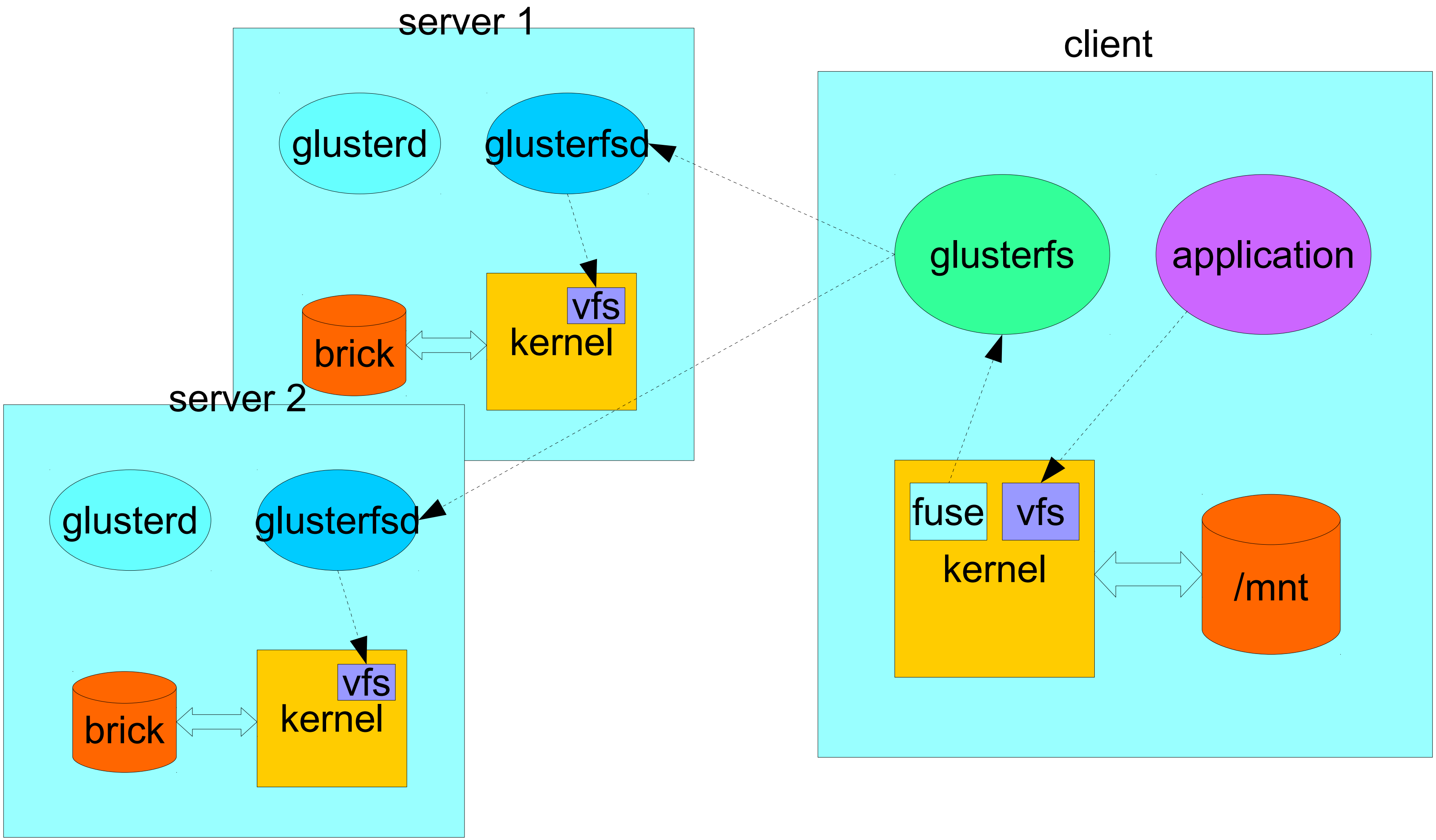


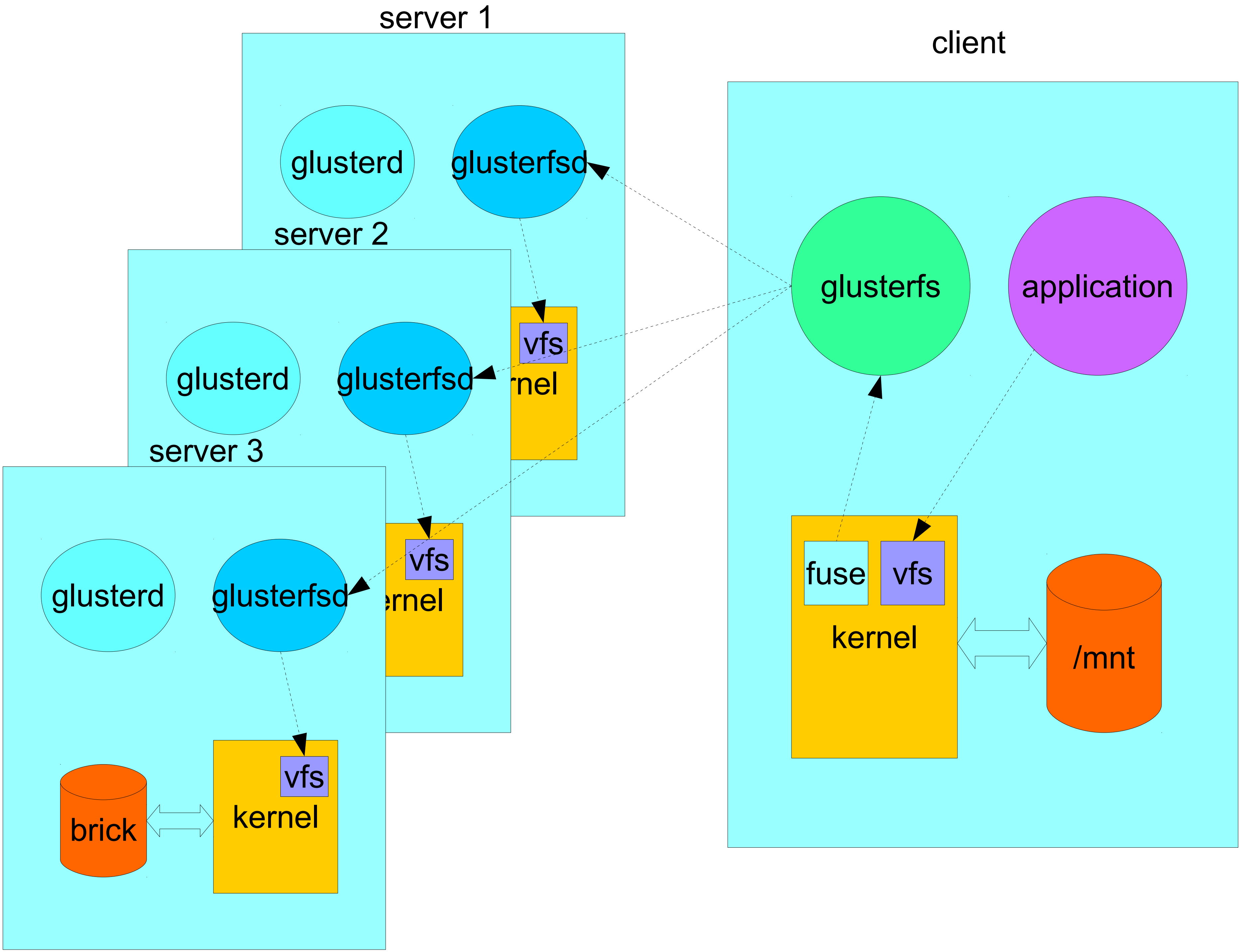
Adding a brick — distributed and rebalancing



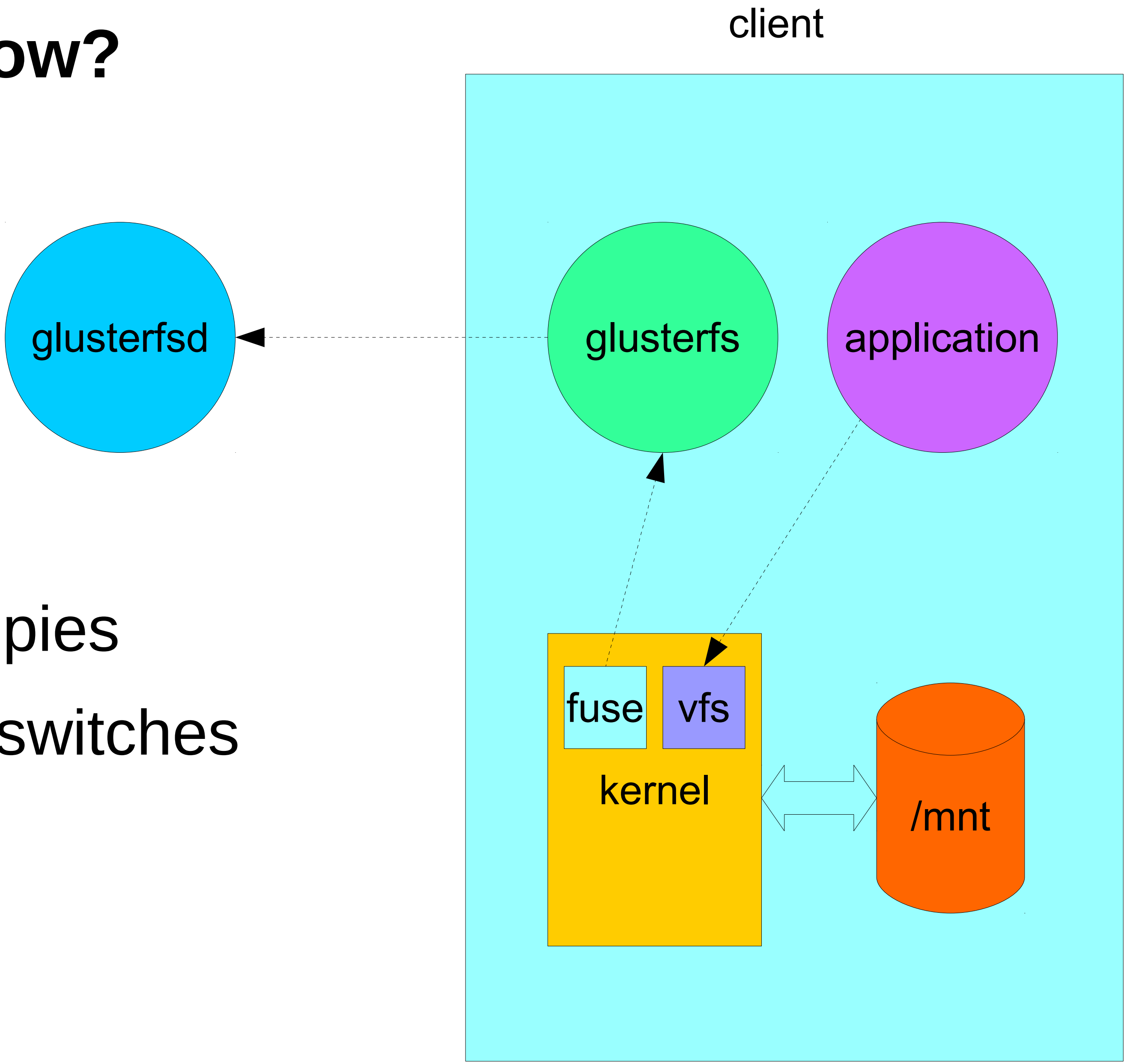

server 1
glusterd
glusterfsd
vfs
rnel
server 2
glusterd
glusterfsd
vfs
ernel
server 3
glusterd
glusterfsd
brick
vfs
kernel
client
glusterfs
application
fuse
vfs
kernel
/mnt

Adding a brick — replica and healing



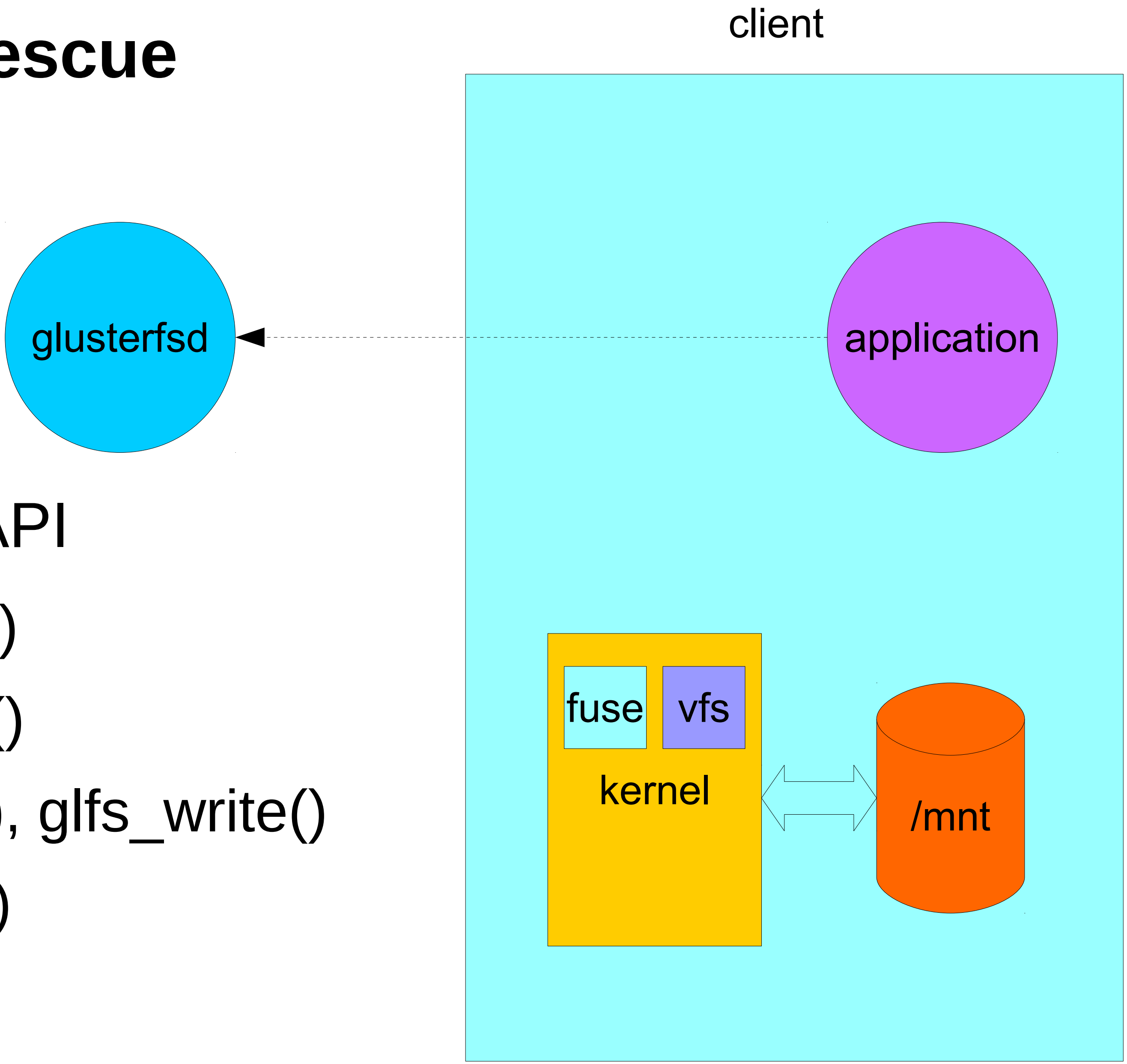

server 1
glusterd
glusterfsd
vfs
rnel
server 2
glusterd
glusterfsd
vfs
ernel
server 3
glusterd
glusterfsd
vfs
kernel
brick
client
glusterfs
application
fuse
vfs
kernel
/mnt

Isn't FUSE slow?



- Count the copies
- And context switches

gfapi to the rescue

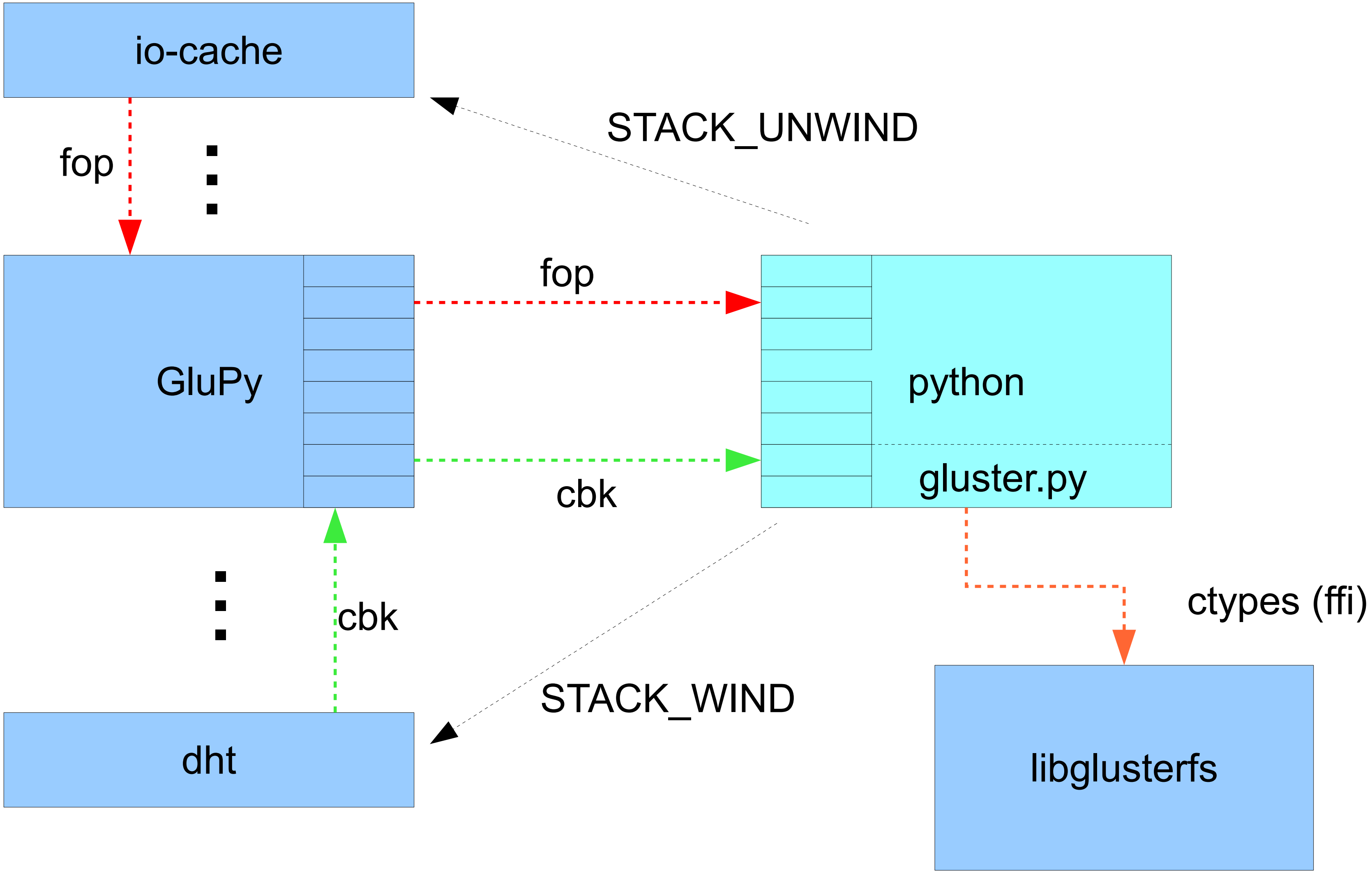


- POSIX-like API
  - glfs_open()
  - glfs_close()
  - glfs_read(), glfs_write()
  - glfs_seek()
  - glfs_stat()
  - etc.

libgfapi — hellogluster.c

```
#include <gluster/api/glfs.h>

int main (int argc, char** argv)

{

    fs = glfs_new (“fsync”);

    ret = glfs_set_volfile_server (fs, “tcp”, “localhost”,
    24007);

    /* or ret = glfs_set_volfile (fs, “/tmp/foo.vol”); */

    ret = glfs_init (fs);

    fd = glfs_creat (fs, filename, O_RDWR, 0644);

    ret = glfs_write (fd, “hello gluster”, 14, 0);

    glfs_close (fd);

    return 0;

}
```

Writing a translator in Python with GluPy



SwiftOnFile — RESTful API, examples with curl

- Create a container:
  ```
  curl -v -X PUT -H 'X-Auth-Token: $authtoken' https://$myhostname:443/v1/AUTH_$myvolname/$mycontainername -k
  ```
- List containers:
  ```
  curl -v -X GET -H 'X-Auth-Token: $authtoken' https://$myhostname:443/v1/AUTH_$myvolname -k
  ```
- Copy a file into a container (upload):
  ```
  curl -v -X PUT -T $filename -H 'X-Auth-Token: $authtoken' -H 'Content-Length: $filelen' https://$myhostname:443/v1/AUTH_$myvolname/$mycontainername/$filename -k
  ```
- Copy a file from a container (download):
  ```
  curl -v -X GET -H 'X-Auth-Token: $authtoken' https://$myhostname:443/v1/AUTH_$myvolname/$mycontainername/$filename -k > $filename
  ```

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: GPLv2 or LGPLv3+

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 *);
```

Data Types in Translators

- 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)
- client_t — represents the connect client

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 lose 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: FOP 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;
}
```

Call to action

- Go forth and write applications for GlusterFS!

£©€é¥¢ßŒ