GlusterFS Translators Conceptual Overview

Jeff Darcy

August 28, 2012

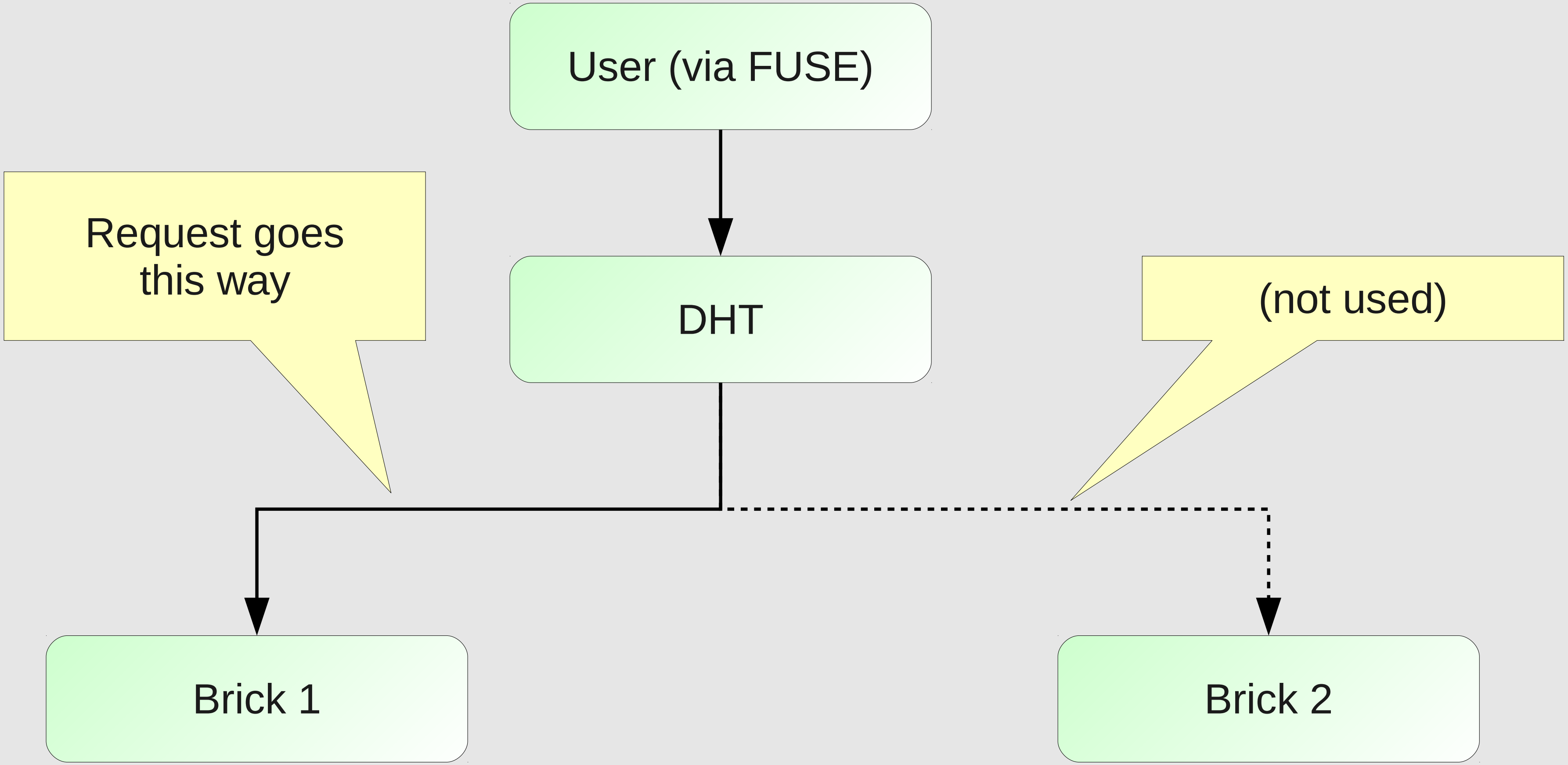
We Should Have Called It DIYFS

- GlusterFS is a not a file system
- It's a way to build new file systems
- We happen to have built a fairly nice one
  - distribution, replication, NFS/Swift/Hadoop, . . .
  - come see that presentation tomorrow
- Don't like it? Build your own!

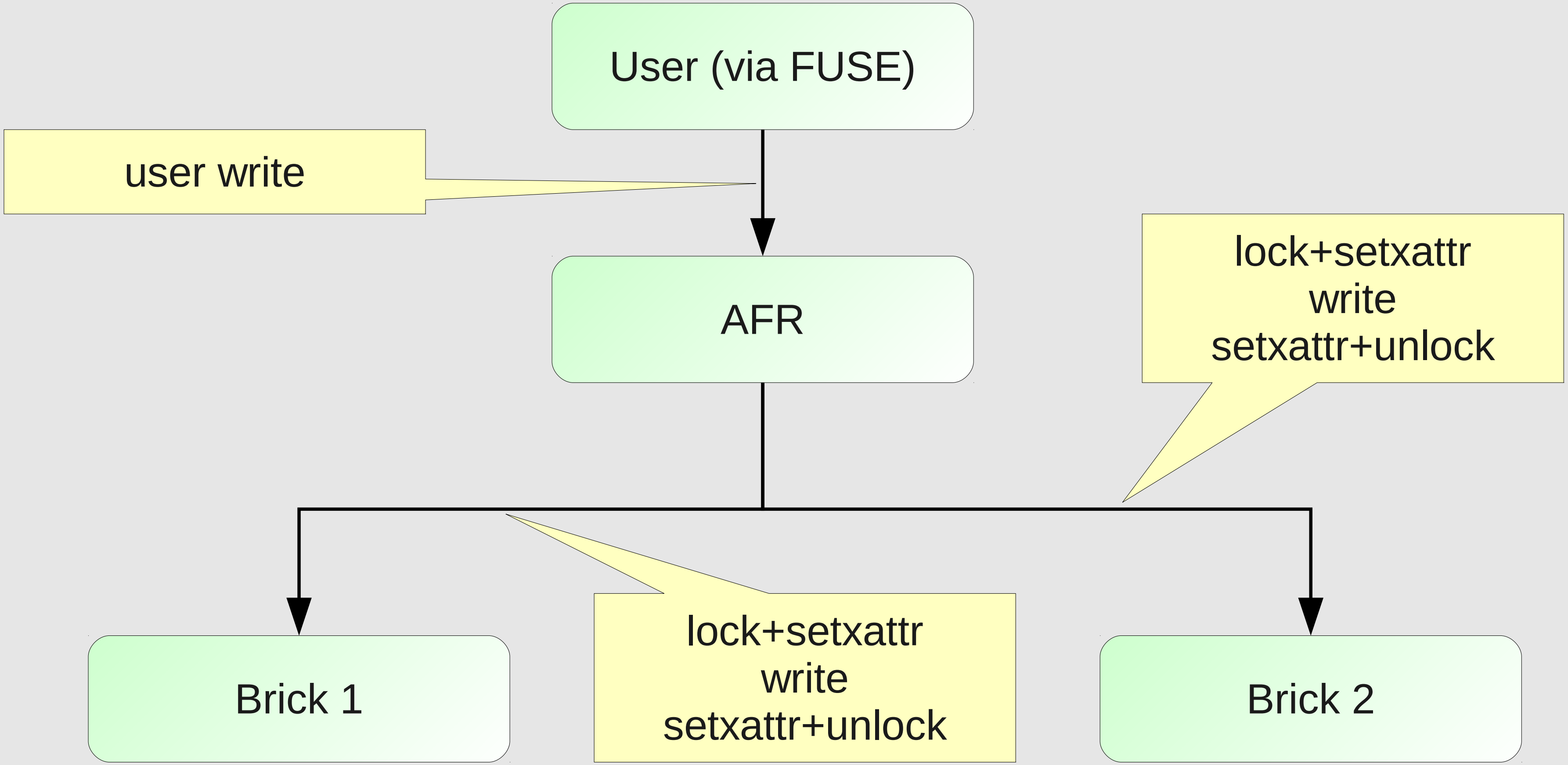
Translating “translator”

- A translator converts requests from users into requests for storage
  - one to one, one to many, one to zero (e.g. caching)
- A translator can modify requests on the way through
  - convert one request type into another
  - modify paths, flags, even data (e.g. encryption)
- ...intercept or block them (e.g. access control)
- ...or spawn new requests (e.g. pre-fetch)

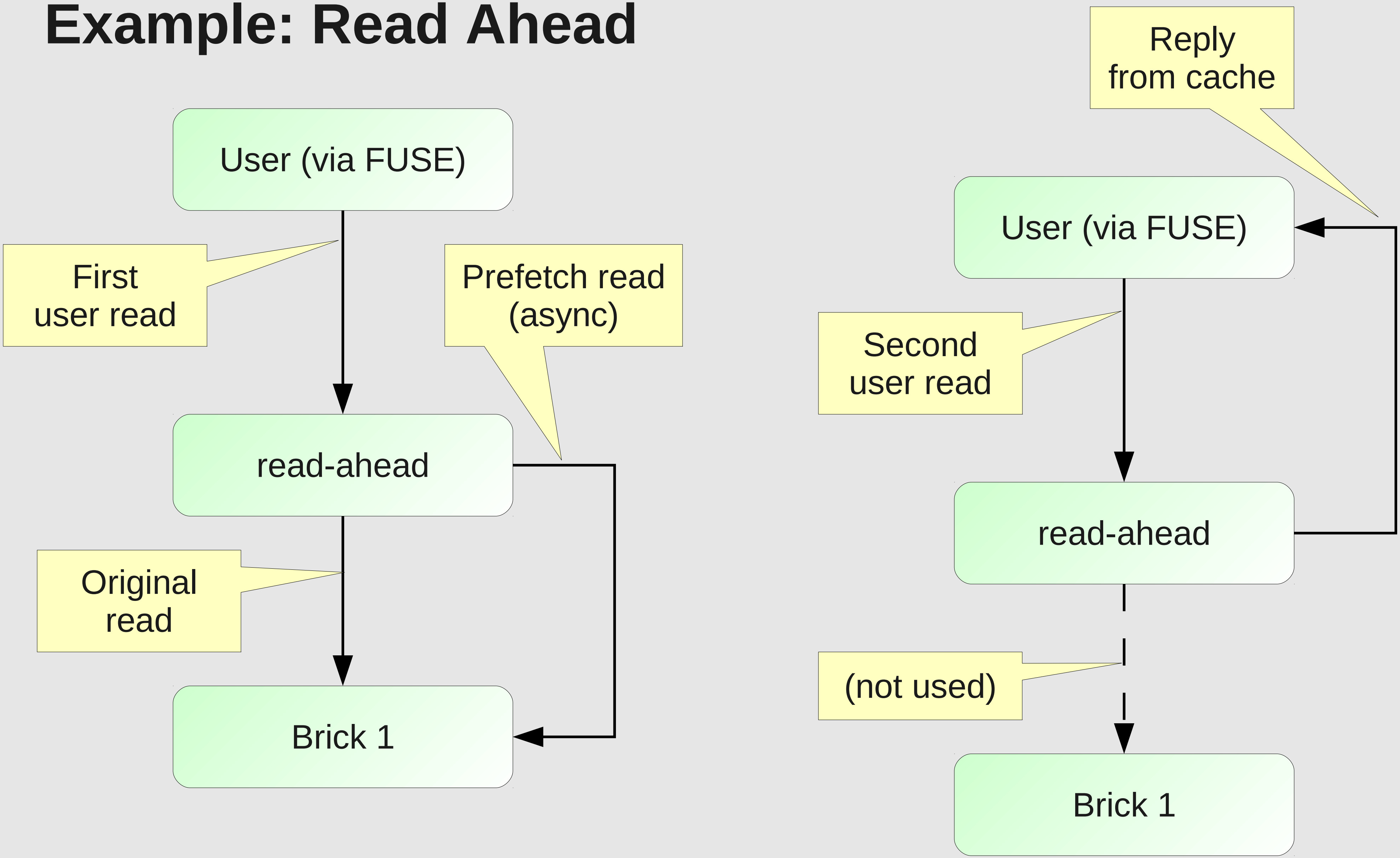
Example: Request Routing in DHT



Example: Request Fan Out in AFR



Example: Read Ahead



Why Build Your Own?

- GlusterFS represents a particular set of design choices
  - e.g. data safety is first priority
  - . . . then consistency is second . . .
  - . . . finally performance
- Those choices aren't right for everyone
- Tuning only gets you so far
- We can never cover all of the use cases
  - this is where HekaFS came from

Tradeoff Example: Slow Replication

- Principle: data safety before performance
- We do extra operations to make sure data survives a crash
- That means more network round trips
- Optimizations work well for buffered sequential writes
  - not so much for small/random/synchronous writes
- Lesson: AFR (today) might not be right for some workloads (e.g. virtual-machine images)
  - . . . so I wrote bypass, hsrepl

Tradeoff Example: Slow Directory Listings

- Principle: consistency before performance
- We assume other clients might have added, changed, or deleted files
- We do a new lookup/stat/getxattr each time
- This especially hurts us e.g. with PHP scripts, git service
- Lesson: tune cache/prefetch translators, use autoloaders/APC
  - . . . or try xattr-prefetch, negative-lookup

Benefit of DIY

- Let's see how negative-lookup helps “PHP” workload
  - 1000 files spread across 10 directories
  - power-law distribution: 80% of hits to 10% of files
- Measure time to find each file

```
[root@gfs-i8c-04 phpsim]# ./worker.py
average latency = 2.478ms
```

GlusterFS 3.3

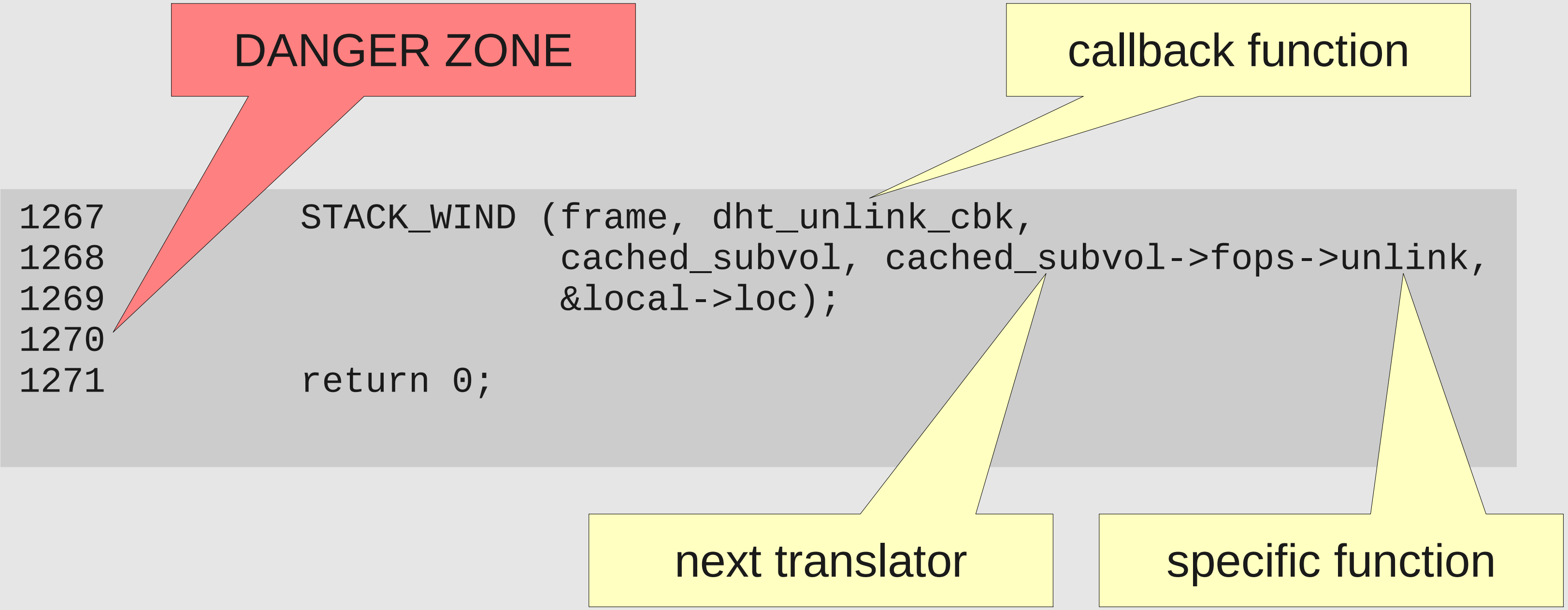
```
[root@gfs-i8c-04 phpsim]# ./worker.py
average latency = 0.690ms
```

negative-lookup
over 3x as fast

How Do Translators Work?

- Shared objects
- Dynamically loaded according to “volfile”
  - dlopen/dlsym
  - set up pointers to parents/children
  - call *init* (constructor)
  - call I/O functions through *fops*
- Conventions for validating/passing options etc.
- “Translator 101” series at hekafs.org

Asynchronous Programming Model



DANGER ZONE

callback function

```
        STACK_WIND (frame, dht_unlink_cbk,
                    cached_subvol, cached_subvol->fops->unlink,
                    &local->loc);

        return 0;
```

next translator

specific function

Danger Zone?

- You lost control when you called STACK_WIND
  - callback might have already happened reentrantly
  - . . . or it might be running on another thread right now
  - . . . or it might not run for a long time
- Data might be modified, freed, still in use
- Be extremely careful doing anything but return after STACK_WIND
  - (please clean up local allocations/references though)

Saving Context

- Pass translator-specific information between original function and call back
- Framework provides *frame->local* for exactly this
  - pointer to whatever structure you want
  - local to call, not translator (that's *this->private*)
  - you allocate from mem_pool, we free when call is done
- Gotcha: *frame* will be shared between STACK_WIND callbacks

Fan Out

local is shared

```
            xlator_list_t    *trav = NULL;
…
            trav = this->children;
...
            local->call_count = priv->child_count;
            while (trav) {
                    STACK_WIND (frame, stripe_lookup_cbk, trav->xlator,
                                trav->xlator->fops->lookup,
                                loc, xattr_req);
                    trav = trav->next;
            }
```

child-iteration idiom

Fan In

lock shared structure

```
LOCK (&frame->lock);
{
        callcnt = --local->call_count;
…
}
UNLOCK (&frame->lock);

if (!callcnt) {
```

how we know
we're done

Deferring Calls

capture arguments
in a structure

```
stub = fop_writev_stub (frame, NULL, fd, vector, count, offset, flags,
                        iobref, xdata);
....
call_resume (stub);
```

from callback
or worker thread

- There's an *fop_xxx_stub* for each operation type
  - . . . and for each callback too
- You can also *call_stub_destroy* instead of resuming

Initiating New Calls

actually creates
a whole stack

sometimes
copy_frame

```
newframe = create_frame(this,&priv->pool);
...
STACK_WIND_COOKIE (newframe, hsrepl_np_cbk, child1,
                   child1, child1->fops->setxattr,
                   &tmploc, dict, 0, NULL);
...
STACK_DESTROY(frame->root);
```

in callback
instead of STACK_UNWIND

Persistent objects

- Inode (*inode_t*) represents a file on disk
- File descriptor (*fd_t*) represents an open file
- Reference counted - *inode_ref*, *fd_unref*
- Translators can add own context
  - e.g. inode_ctx_put (inode, xlator, value)
  - values are 64-bit unsigned (or pointers)
  - adding context causes translator's forget/release to be called when object is deleted

Utility Functions

- Dictionaries
- Memory management with accounting: GF_MALLOC, GF_CALLOC, GF_FREE
- Logging: gf_log, gf_print_trace
- UUIDs, hashes, red/black trees, name resolution
- all sorts of other stuff