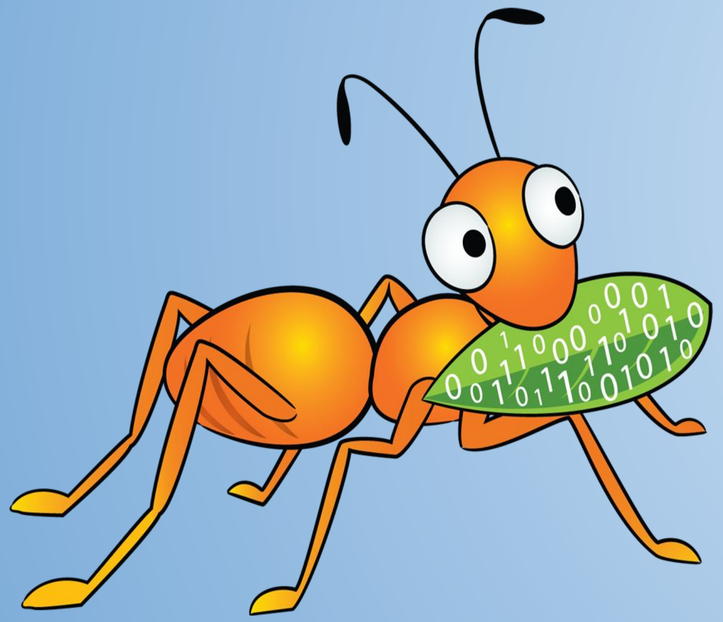
# A *thin arbiter* for glusterfs replication



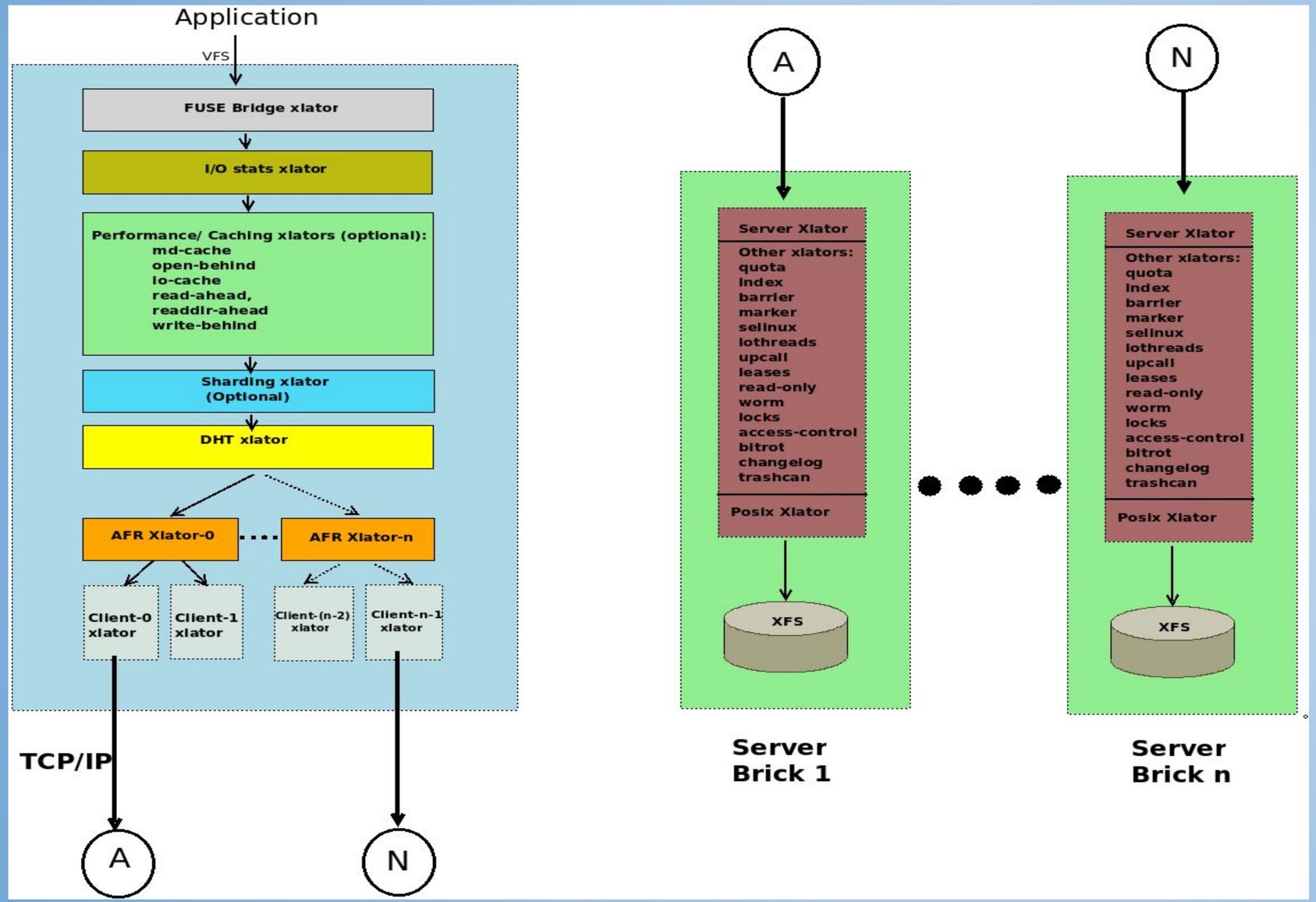
**Ravishankar N.** (@itisravi)
Sr.Software Engineer,

February 2nd, FOSDEM 2020

# Agenda

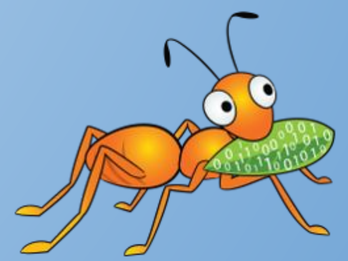
- The 1-slide intro to glusterfs architecture.
- Synchronous replication and the AFR translator.
- Quorum logic and split-brain prevention.
- Thin Arbiter based replication.

# Glusterfs Architecture




Some keywords:

- Servers
- Bricks
- Peers
- Trusted Storage Pool
- Clients
- Volinfo
- Volume graph
- Translators
- FOP (File operation)
- gfid
- xattrs

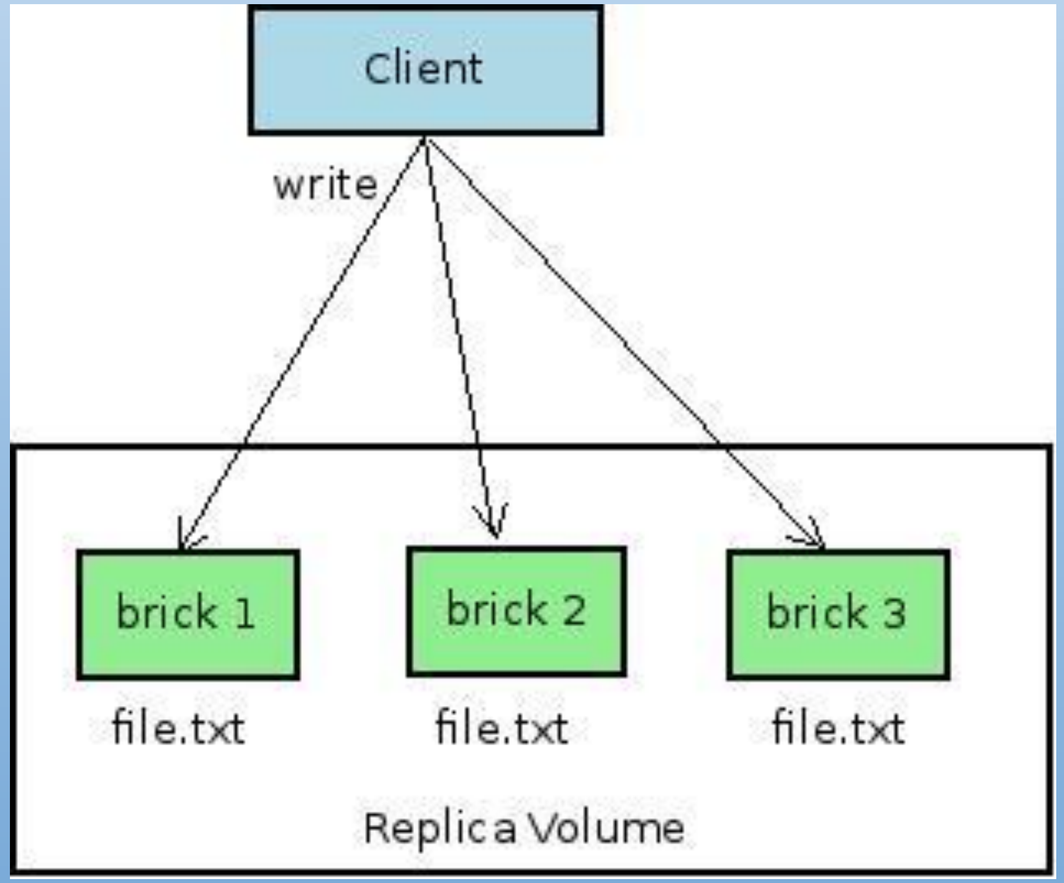


# Synchronos replication in gluster

Automatic File Replication (AFR)

- client driven.
- strong consistency model.
- writes follow a 5-step transaction (with optimizations).
- reads served from one of the replicas.
- slowest brick dictates write performance.
- auto self-healing of partial/missed writes.
- CLI to monitor heals and resolve conflicts.

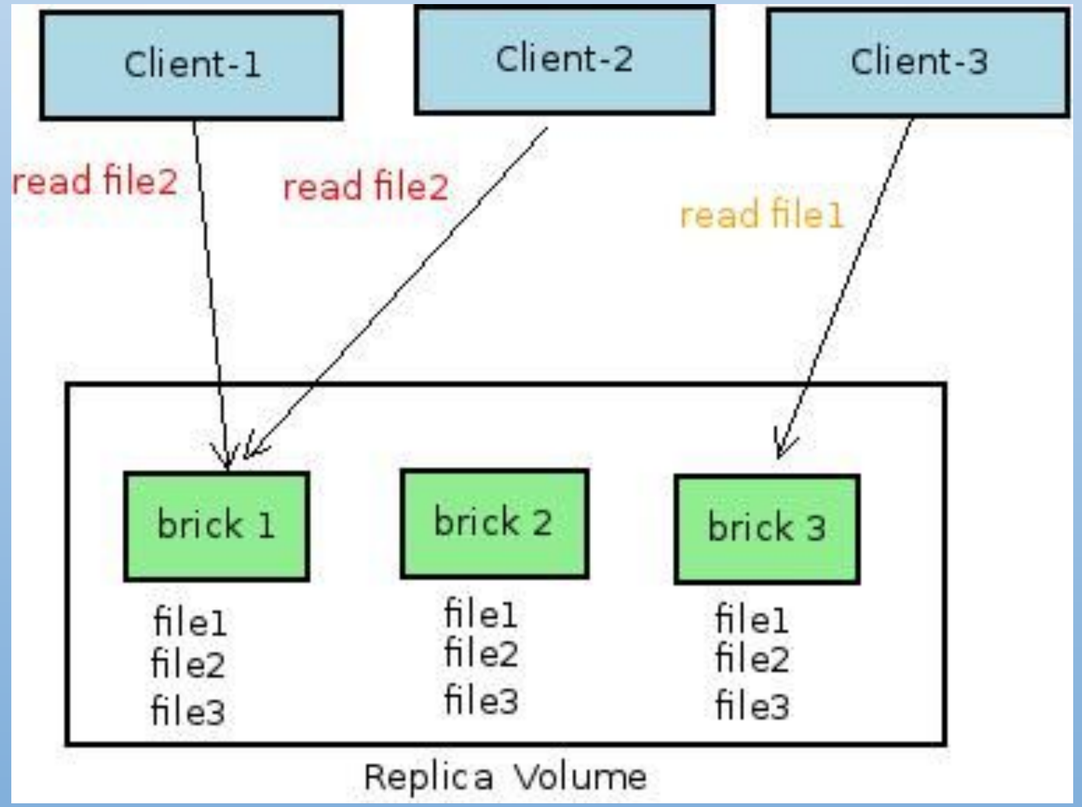
# Automatic File Replication - Writes




5 Phase Transaction Model

1. Lock
2. Pre-op (set dirty xattr on files)
3. Actual FOP (write, setfattr etc)
4. Post-op (clear dirty, set pending xattr for failures)
5. Unlock

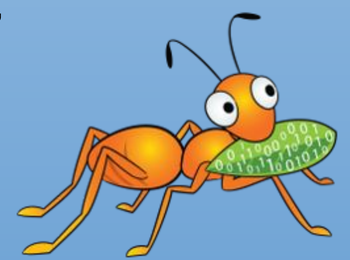
# Automatic File Replication - Reads




-Reads are served from one of the (good) bricks.

- which brick? configurable via policies.

```
typedef enum {
    AFR_READ_POLICY_FIRST_UP,
    AFR_READ_POLICY_GFID_HASH,
    AFR_READ_POLICY_GFID_PID_HASH,
    AFR_READ_POLICY_LESS_LOAD,

    AFR_READ_POLICY_LEAST_LATENCY,
    AFR_READ_POLICY_LOAD_LATENCY_HYBRID,

} afr_read_hash_mode_t;
```

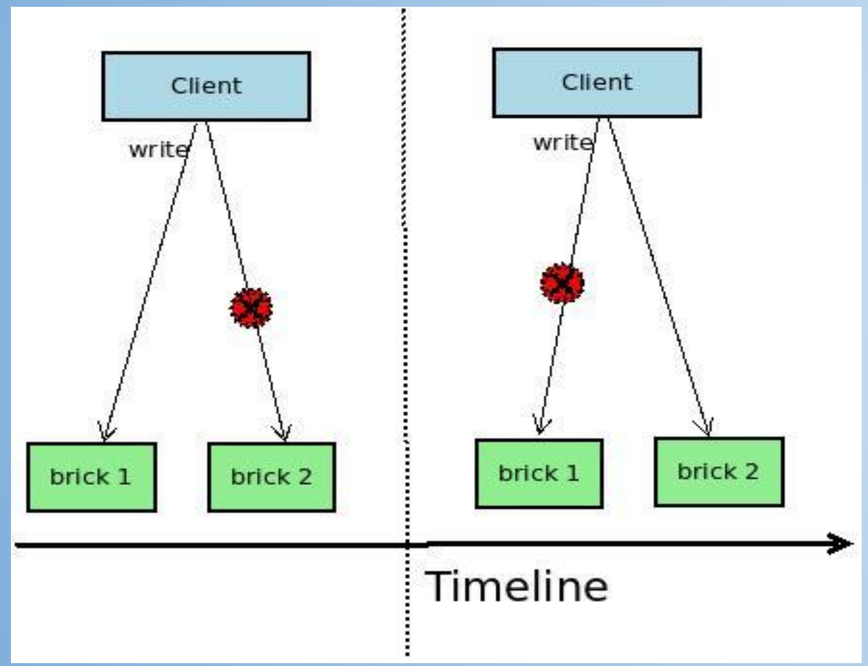


# Automatic File Replication - Self-heal

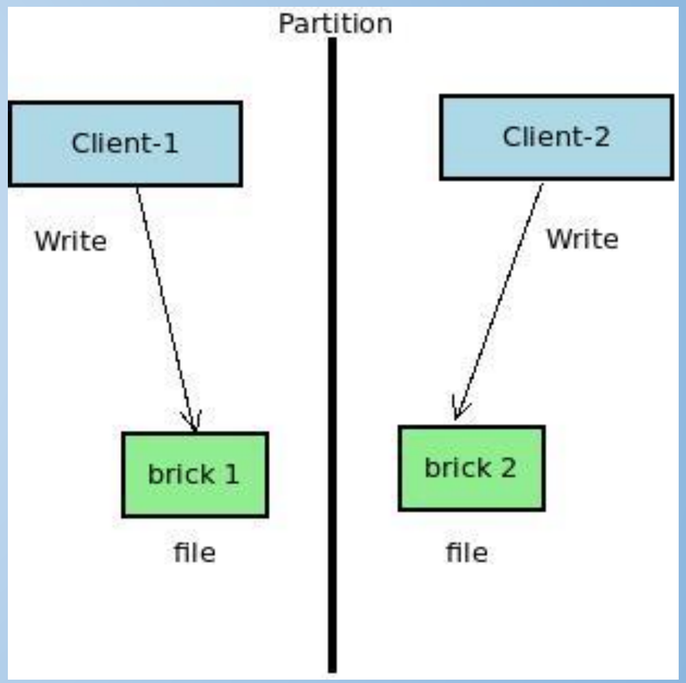
- The self-heal daemon (shd) runs on every node.
- Heals data/metadata/ entries of all volumes on that node.
- GFIDs of files that need heal are stored inside .glusterfs/indices folder of the bricks.
- Shd crawls this folder every 10 minutes (configurable) and heals the files.
- Healing takes place under locks for mutual exclusion from client I/O.

# Automatic File Replication - Replica 2

Replica 2 config – prone to split-brains: in time and space.



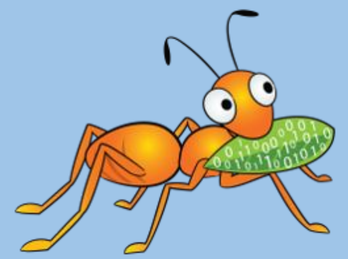

Split-brain in time



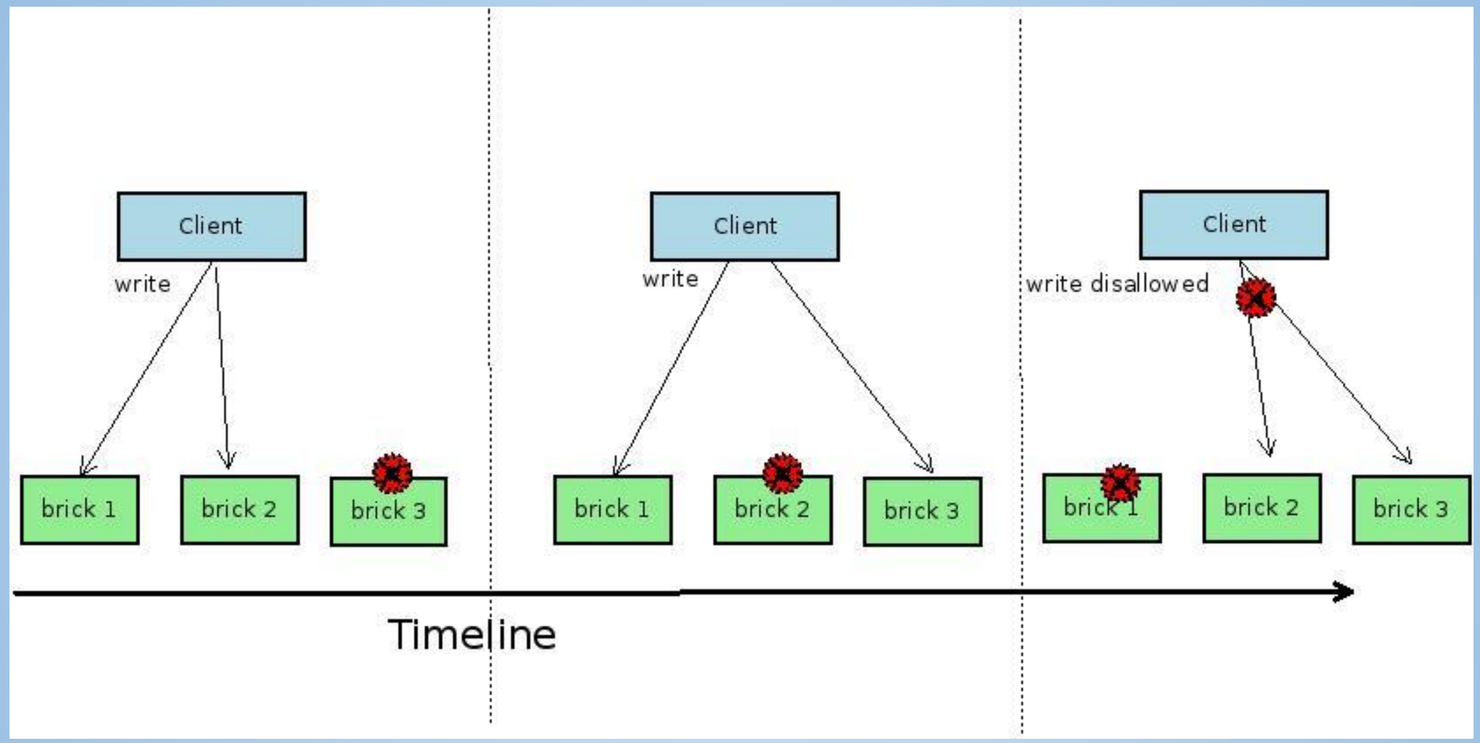

Split-brain in space

# Automatic File Replication - Replica 3

- To prevent split-brains, we need odd no. of replicas.
- We can then establish quorum (majority voting).
- In a (2n+1) replica, clients can continue to work with at most ‘n’ replicas going down.
- So for replica 3, at most 1 brick can be down.
  - However, if the only good copy is down, then I/O will fail even if 2 bricks are up.

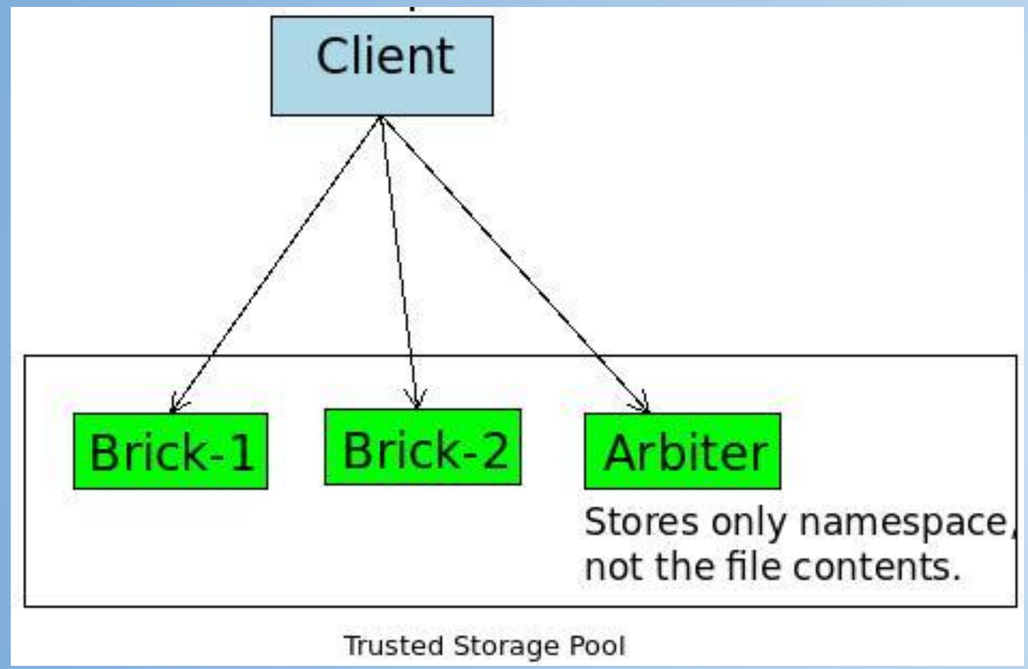


# Automatic File Replication - Replica 3




- Since we have 3 copies of afr xattrs, we can avoid split-brains.
- There must be at least one brick that is not blamed by the others.

# Automatic File Replication - Arbiter

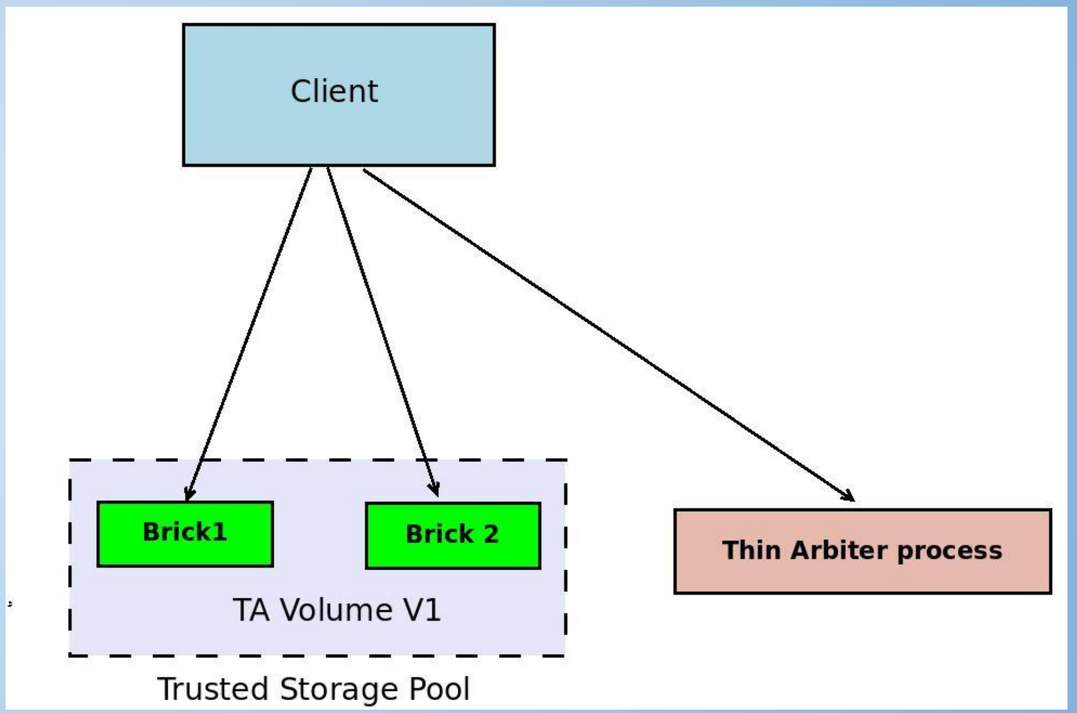



- Unlike replica 3, arbiter brick stores only file names. i.e. 0 byte files.
- But since each file also stores the afr xattrs, quorum logic for preventing split-brains will work.
- Availability is less compared to replica-3.

# Replication with Thin Arbiter (TA)

- TA volume = replica 2 volume + lightweight TA process.

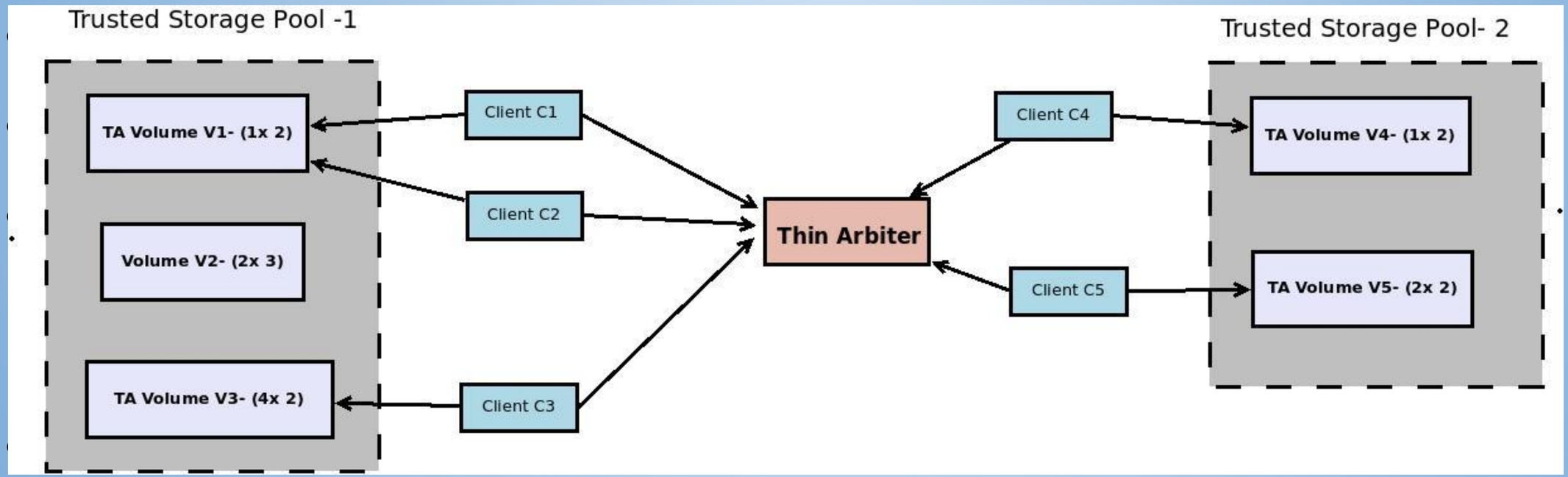
```
[root@ravi1 ~]# gluster volume info

Volume Name: testvol
Type: Replicate
Volume ID: 2bbbb050-7a70-443f-8890-a589bbc5ed2f
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: 10.70.41.242:/bricks/brick1
Brick2: 10.70.41.247:/bricks/brick2
Thin-arbiter-path: 10.70.43.183:/bricks/brick_ta
Options Reconfigured:
transport.address-family: inet
storage.fips-mode-rchecksum: on
nfs.disable: on
performance.client-io-threads: off
```



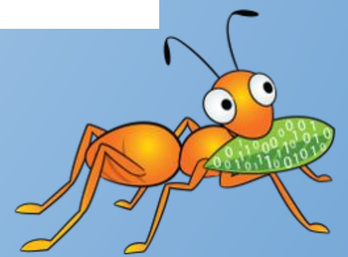

- The TA process resides on a separate node **outside** the gluster storage pool.
- The node is not a peer, i.e. it does **not** run glusterd (mgmt daemon).

# Replication with Thin Arbiter (TA)

- One TA process can serve multiple volumes of the *same** storage pool.
- It can also be used across different pools, but the volume names must be *unique**.



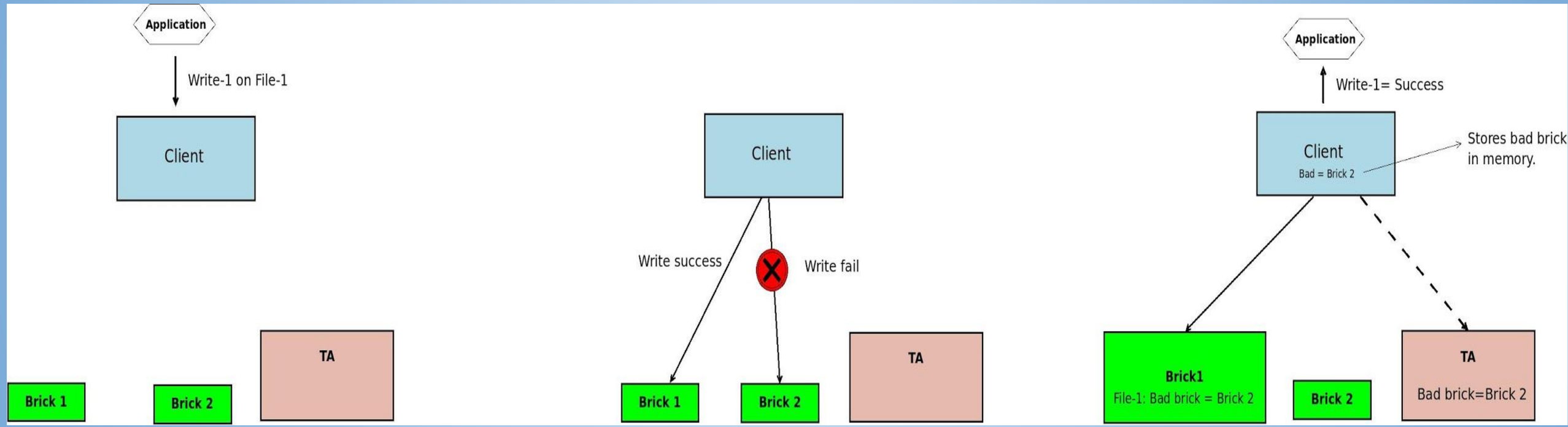

*Support for same TA for multiple storage pools to prevent volname collsion is being worked on.



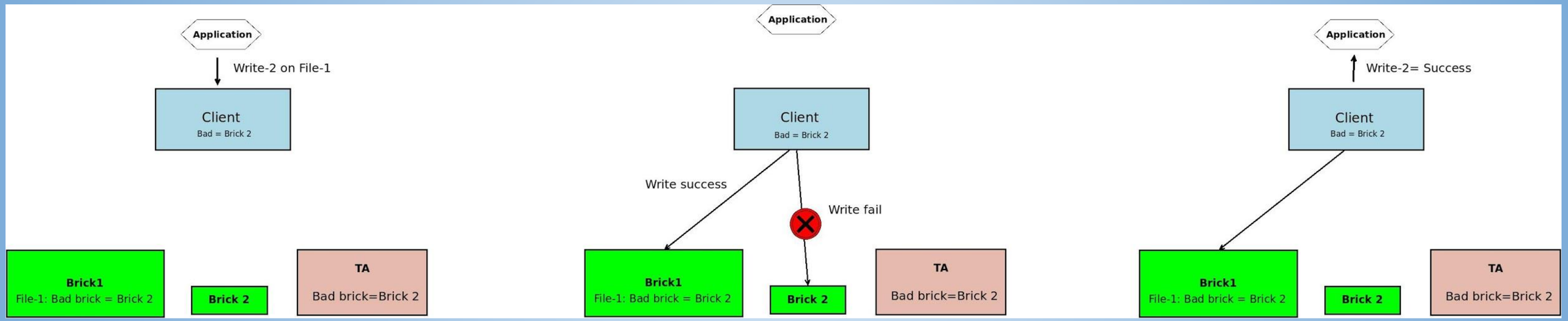
# Thin Arbiter process


- The thin arbiter process similar to a normal brick process but has the thin-arbiter xlator in addition to the other server side xlators.
- It stores zero-byte sized ‘replica ID’ files, one for each replica subvolume.
  - Eg. For a 2x2 TA volume, there will be 2 files: trusted.afr.testvol-ta-2 and trusted.afr.testvol-ta-5
- The ID file has afr xattrs indicating the good or bad (i.e. pending heals) state of the 2 data bricks of that specific replica.
- During the 1st mount of the volume, AFR creates the ID file on TA node.
- The job of the thin-arbiter xlator is to allow only create and xattrop FOPs on the ID file.
- The actual arbitration logic resides on the client side inside AFR.
- In the default setup, it uses port no 24007 to connect with clients.
  - If you decide to start it with a different port no., you need to update the client volfile using *client.ta-brick-port* volume option.

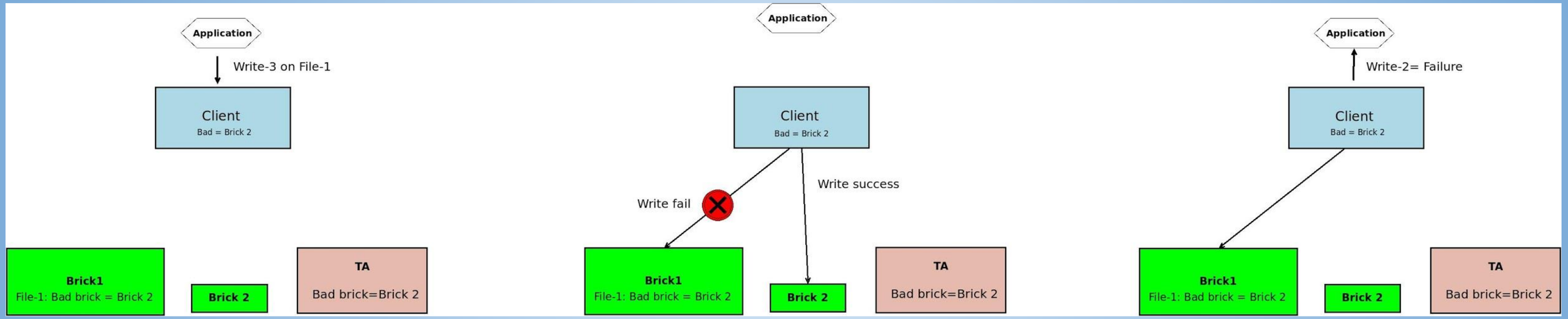
# Thin Arbiter working - writes




# Thin Arbiter working - writes




# Thin Arbiter working - writes



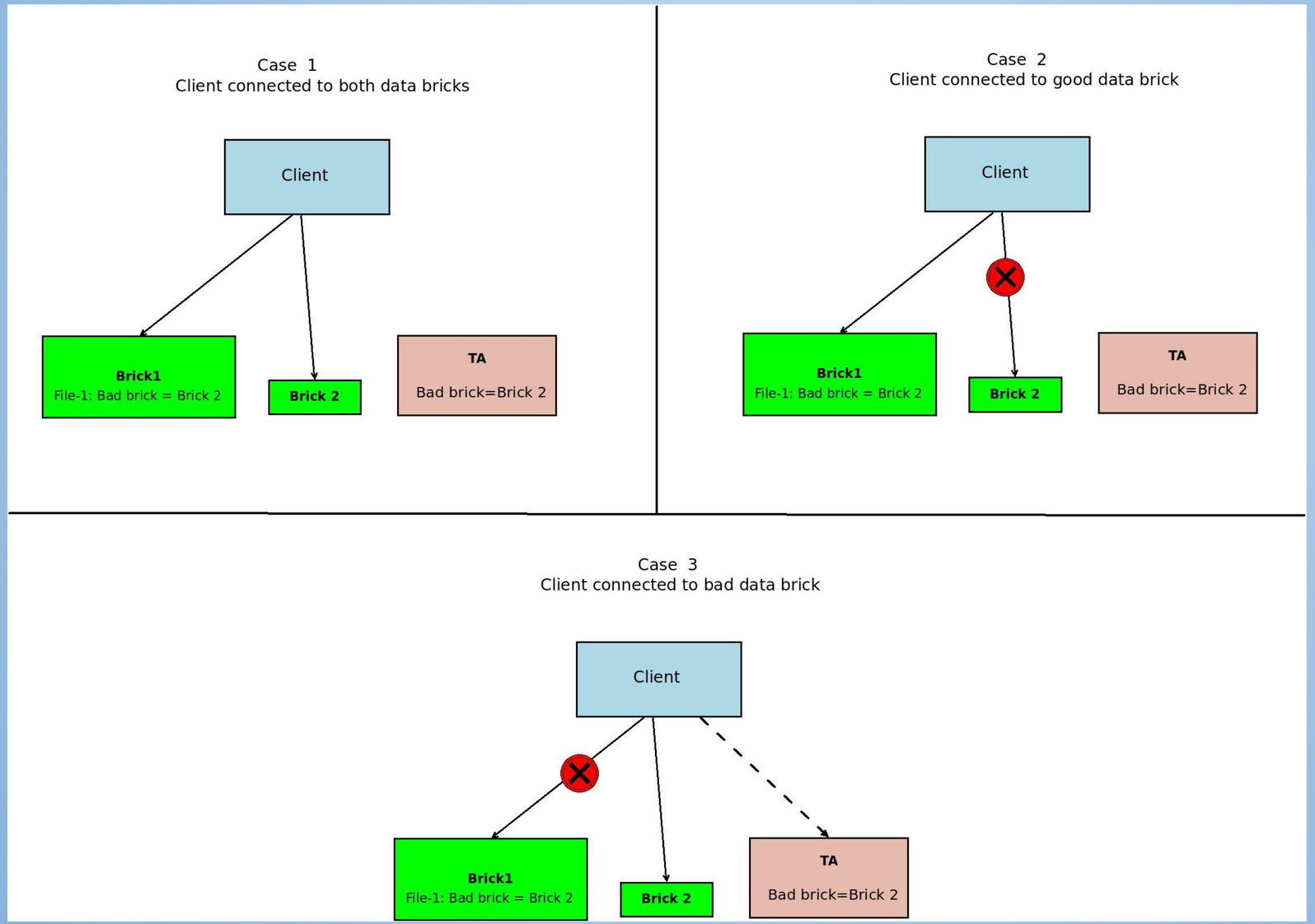

# Thin Arbiter working - writes

Writes:

- If write fails on both data bricks for a file, application receives failure. No marking done on TA node.
- If write fails on brick-2 only, (for say File-1,) mark it as bad on brick-1 and TA.
  - on brick-1, it is captured via afr's pending xattr on File-1.
  - on TA, it is captured via the afr's pending xattr on the ID file.
- The client (AFR) also stores in-memory that brick-2 is bad.
- For subsequent writes on *any file* that fails on brick-2 but succeeds on brick-1, we can return success to the application without asking or setting anything on the thin-arbiter.
- For writes that fail on brick-1 (irrespective of success/failure on brick-2), we return failure to the app.

IOW, If the write succeeds either on both data bricks or at least on the in-memory good copy, it is deemed to be successful.

# Thin Arbiter working - reads



# Thin Arbiter working - reads

Reads:

- If both data bricks are up, serve the read from a good copy (both can be good).
- If one of the data bricks are down:
    - First query the up brick for the file's afr xattrs. If it blames the down brick, serve the read.
    - If it doesn't, query the TA ('cause we can't be sure if the down brick blames the up brick).
    - If TA doesn't blame the up brick, serve the read from it.

# Of self-heal and domain locks

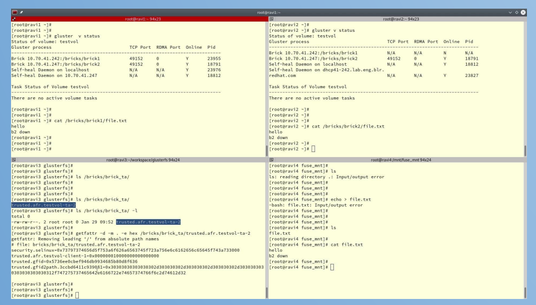
- So clients maintain in-memory which brick is bad. But how does it invalidate this info when self heal heals the bad brick (files)?
  - Using upcall + domain locks.
- Locks translator on the brick has a lock-contention notification feature for inodelk/entrylk.
  - The current lock owner (client) gets a notification whenever another client requests an overlapping blocking lock on the same file.
  - It  also supports locking the same file by the same client if the lock ‘domain’ is different.
- AFR uses these features to invalidate the in-memory info. During the write’s post-op phase on the TA, each client:
  - takes a lock on the ID file in a NOTIFY domain as well as a MODIFY domain.
  - Marks the bad brick on TA (i.e. sets the afr pending xattrs on the ID file)
  - releases only the MODIFY lock.
- So each client has one NOTIFY lock still left on the TA node.

# domain locks contd.

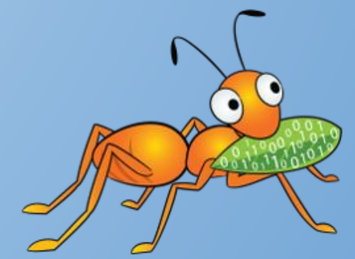
- When shd starts the heal crawl, it attempts a blocking lock on the NOTIFY domain. This triggers an upcall to all clients.
- Clients release their NOTIFY lock held on the ID file on the TA.  If the client still has in-flight writes, it will wait until it is over and then release the NOTIFY lock. It also resets its in-mem info about bad brick.
- shd then inspects the TA file afr xattrs under NOTIFY+MODIFY locks and proceeds with the heal.
- During the heal there are no locks from the shd on the TA.
- If I/O fails during heal, client will again mark the bad brick on the TA and update it’s in-mem info.
- After the heal is over, shd repeats the afr xattr inspection on TA.
- If the  pre and post xattr value are same, there was no additional failures and shd resets the AFR xattrs on TA.
- If xattr values have changed, there were new failures. So the shd attempts the resetting in the next crawl.

# Installation and usage

- On the TA node: Install server rpms and run `setup-thin-arbiter.sh`.
  - creates and starts the TA process.
  - runs as a systemd service - automatically restarted upon crash/ reboot
- Rest of the work flow is normal - peer probe, vol create, vol start, mount and use!
- Create volume syntax:
  - `gluster volume create $volname replica 2 thin-arbiter 1 node1:/brick1 node2:/bricks/brick2 $ta-node:/brick_ta`
  - The data bricks have to be multiples of 2 to create a  dist-rep TA volume. TA node and path needs to be given at the end just once.
- In k8s, kadalu.io is adding support for TA volumes in gluster: https://github.com/kadalu/rfcs/pull/13



<== Demo Video



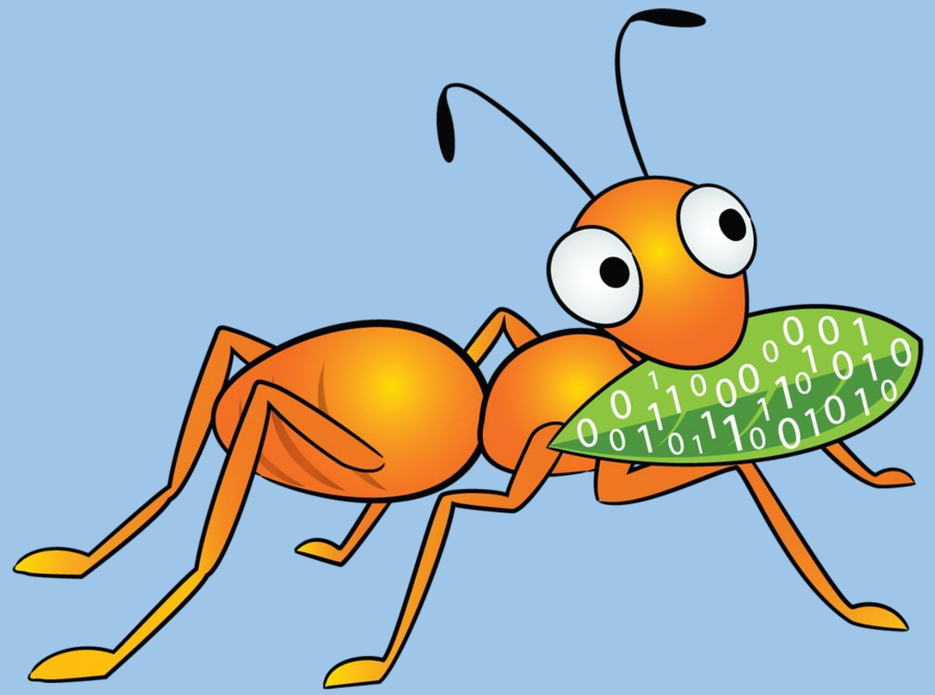
# Things TODO

- Support for add/replace-brick CLI:
  - convert existing replica 2/3/arbiter to TA volume.
  - replace brick for data-bricks and TA node.
- Make reads aware of in-memory information about bad brick.
- Fix reported bugs. 😉

# Reach out to us!

- Mailing lists:
  - gluster-users@gluster.org / gluster-devel@gluster.org
- IRC: #gluster and #gluster-dev on Freenode
- Slack: https://gluster.slack.com
- Links:
  - http://gluster.org/
  - https://docs.gluster.org/en/latest/
  - https://github.com/gluster/

# Questions?



Thank you!