Next Generation File Replication In GlusterFS

Jeff, Avra, Kotresh, Karthik, Rafi KC

About me

- Rafi KC, Software Engineer at Red Hat
  - Rdma, snapshot, tiering, replication

Agenda

- Overview Of GlusterFS
- Existing Replication Model
- Proposed Solution
- JBR-Client
- Leader and Leader Election
- Journaling and Log Replication
- Reconciliation
- Log Compaction
- Q&A

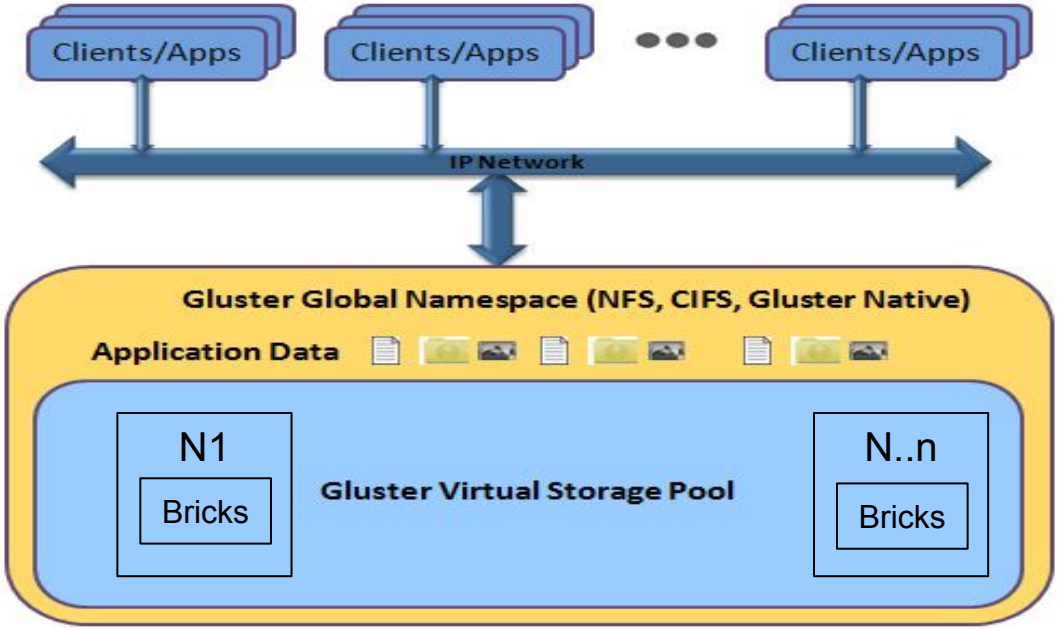
What is GlusterFS

Distributed File System

Software Define NAS

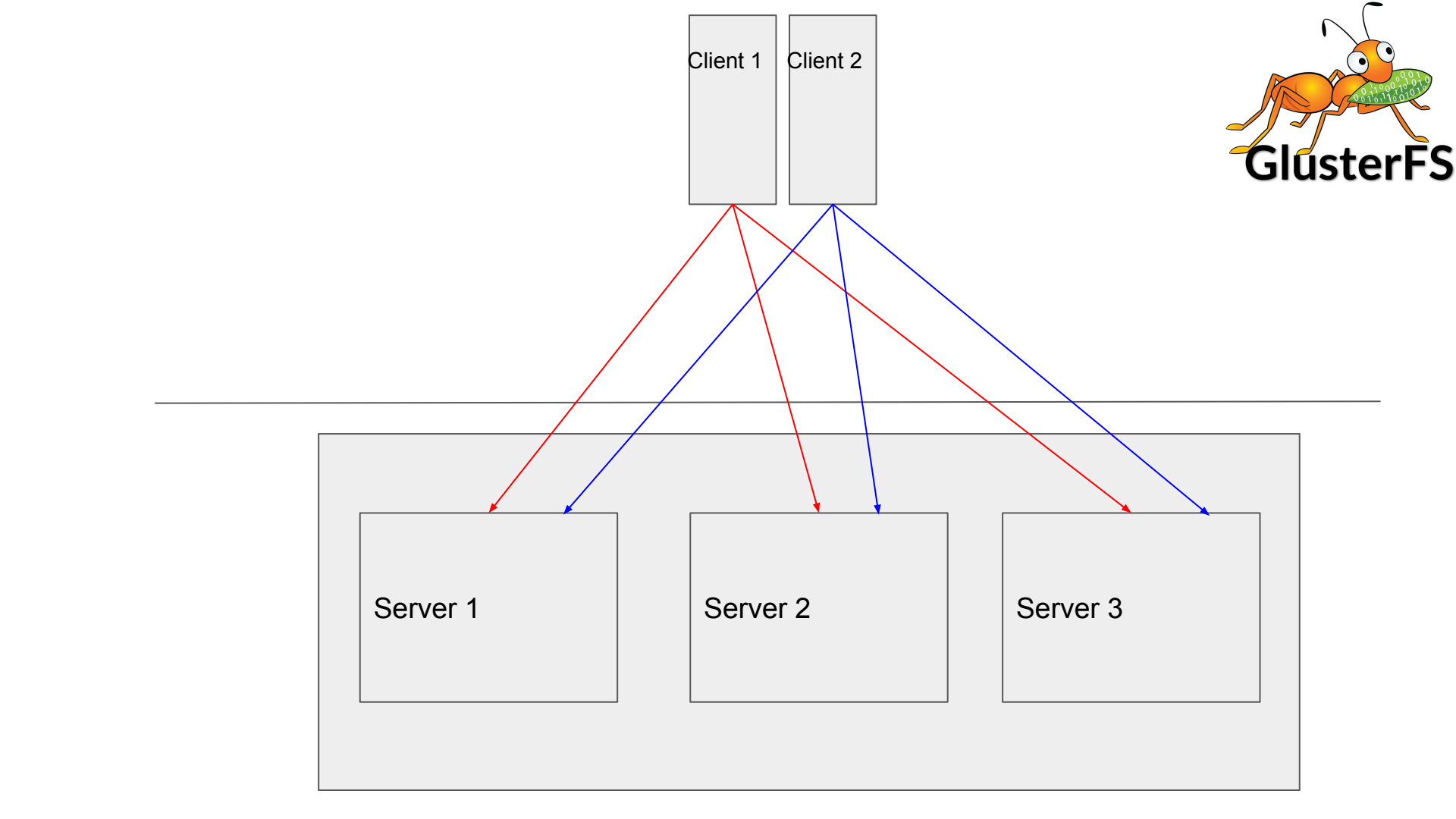
TCP/IP or RDMA

Native Client, SMB, NFS



Existing Replication

- Client side replication
- Asymmetric replication
- It uses client bandwidth
- Synchronous
- Full file heal
- Uses client bandwidth


Client 1
Client 2
GlusterFS
Server 1
Server 2
Server 3

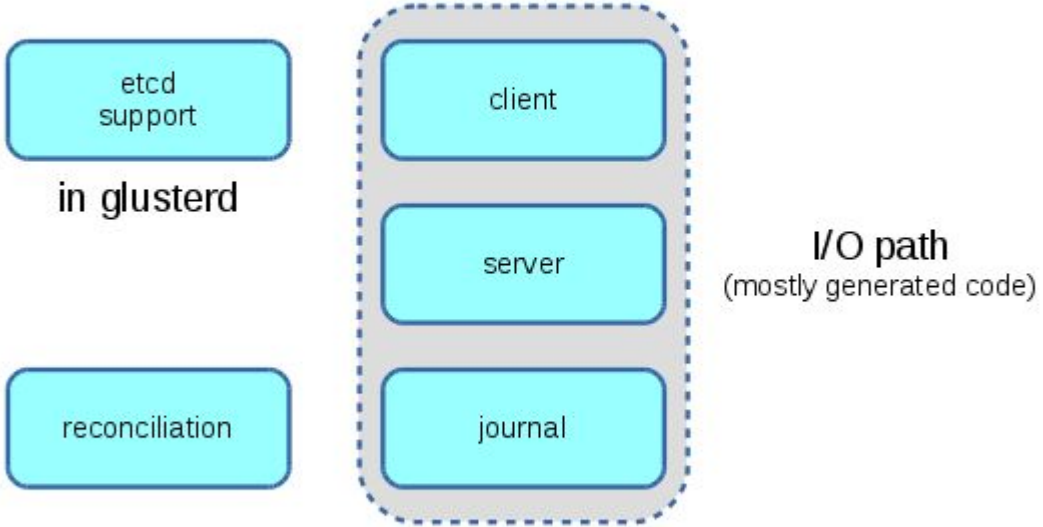
Proposed Solution

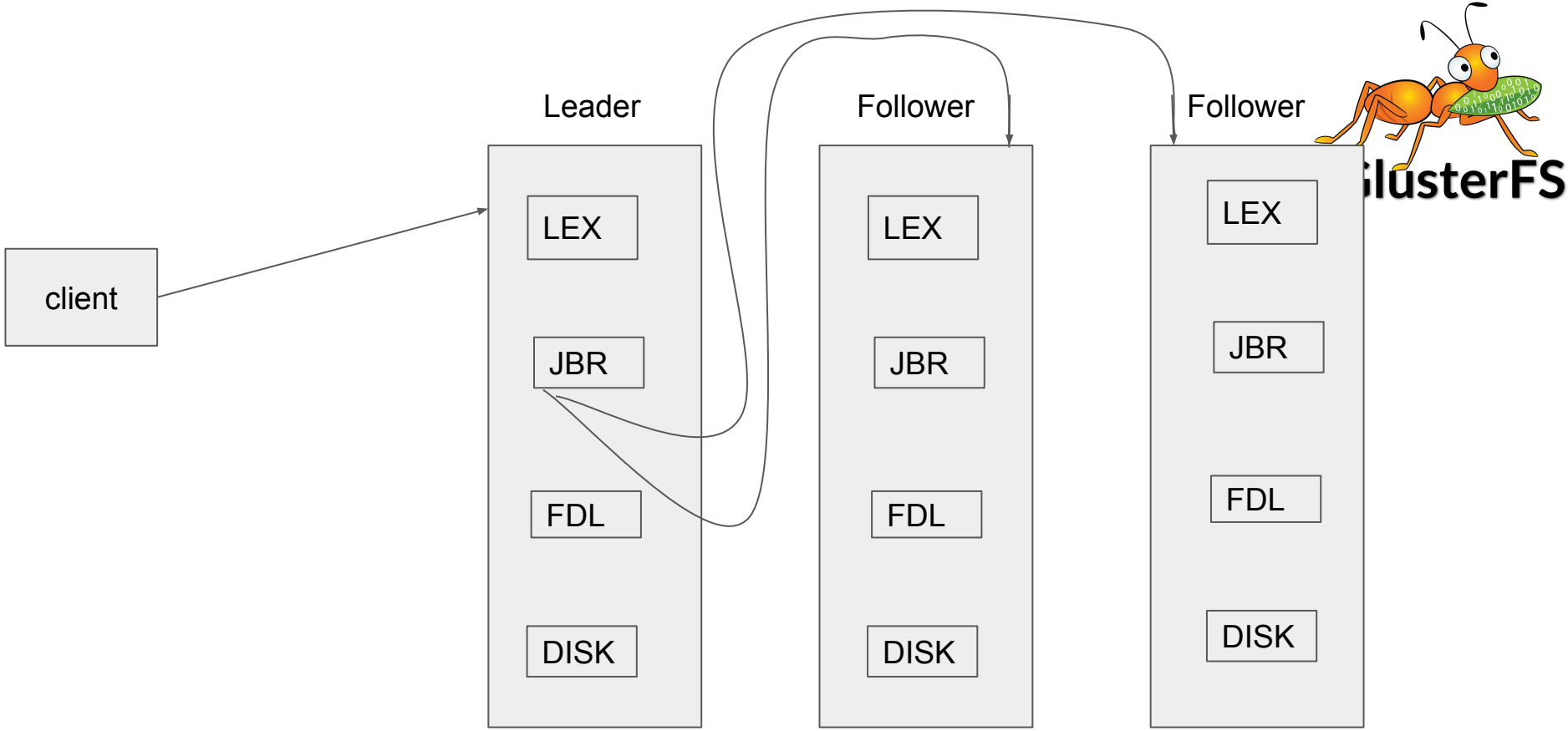
- Server to server
- Faster I/O path for most deployments/workloads
- Temporarily elected leader
  - Simplifies coordination (no locking between clients/shd)
  - Gives leader complete control over ordering and parallelism
  - Within one replica set, not whole volume/cluster

Proposed Solution-cont

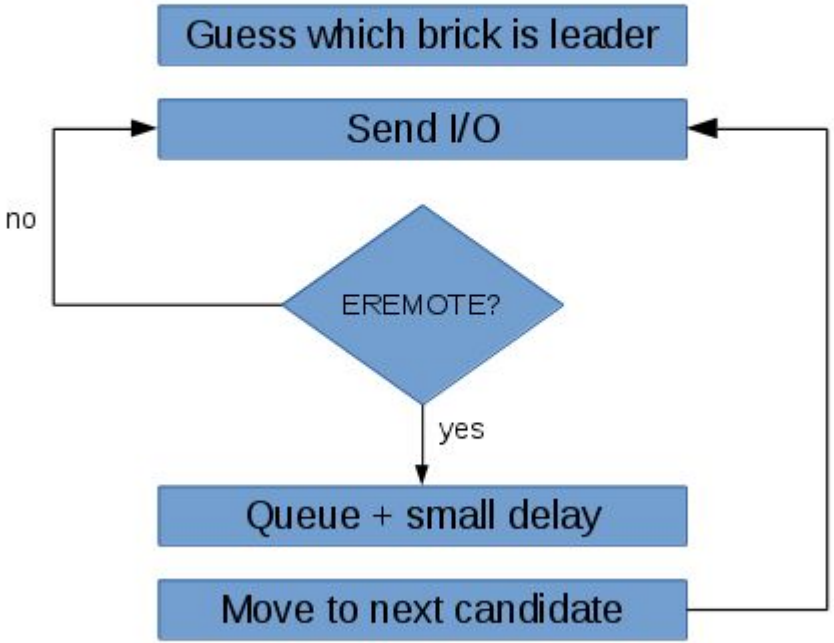
- Log based
  - Allows precise repair
    - No content comparison for multi-GB files
- Flexible consistency
- JBR client and JBR servers

Components


etcd
support
in glusterd
reconciliation
client
server
journal
I/O path
(mostly generated code)


Leader
Follower
Follower
client
LEX
JBR
FDL
DISK
LEX
JBR
FDL
DISK
LEX
JBR
FDL
DISK
GlusterFS

JBR-Client


Guess which brick is leader
Send I/O
no
EREMOTE?
yes
Queue + small delay
Move to next candidate

Leader Election

- LEX relies heavily on a common store in between nodes participating in the leadership election.
- We use etcd compare and swap with ttl (time to live)
- LEX is so modular, can be used independently
- Every set of participating nodes will have a unique key
- Nodes participate the leader election based on certain conditions, ie eligibility check

Leader Election

- Once a leader is elected, it asks for followers to reconcile
- Leader has to renew its leadership in a periodic interval
- After quorum number of nodes reconciled, leader will start replicating the fops from the client.
- Any failure in leader will result in a leadership change.

JBR Server

- Will be loaded in all replication servers
- Leader module will send to all followers
- Take decision based on the response from the followers
- Queue the conflicting fops
- Send rollback request if it failed to replicate on quorum number of followers
- It also stamps the fops to order it when flushing to disk

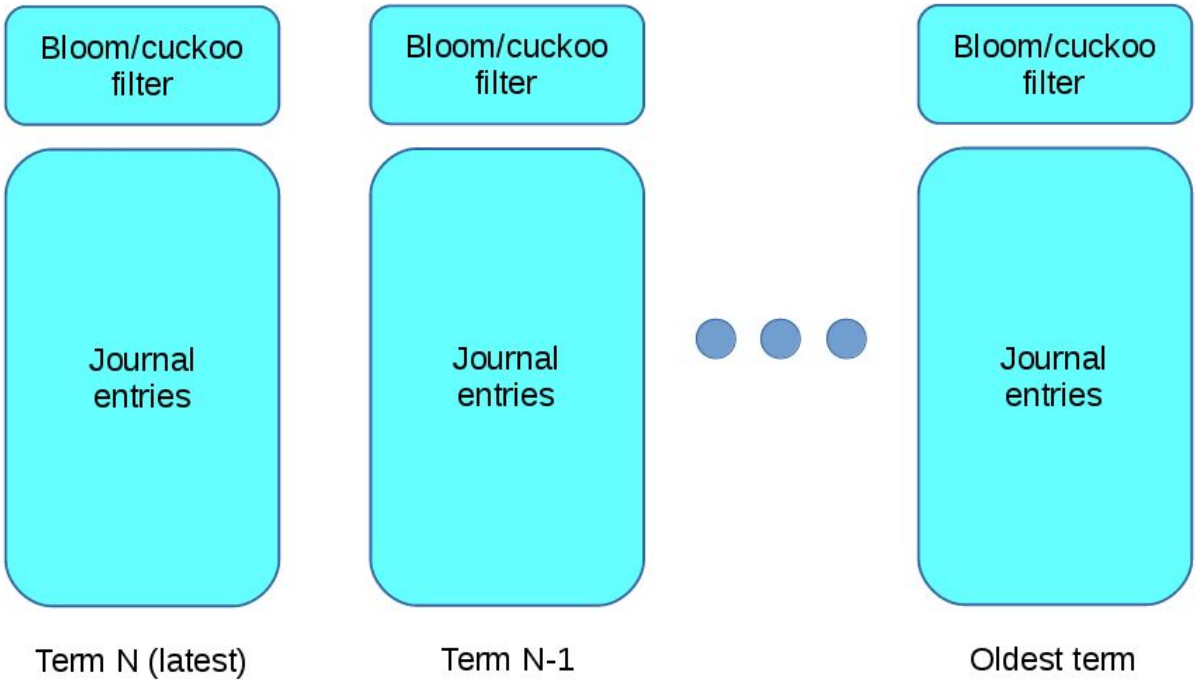
Journals -Terms

- Logs are divided into terms
  - leadership change always implies new term
  - Terms changes may also occur voluntarily (to keep terms short)
    - But no change in leader
- Order of terms is always known
- Information about terms is stored in etcd
- Terms and log index together used as eligibility for leader election
- Journal for each term (on each replica) is stored separately from other terms
  - separate files make space management easier
  - simple/efficient access patterns (later slide)

Journal

- Manages memory + one or more files per term
- can be in memory until fsync/O_SYNC
- can be on separate (faster) device than main store
- Preallocate (in background) + direct/async I/O
- very efficient and flash-friendly

Journal


Bloom/cuckoo filter
Bloom/cuckoo filter
Bloom/cuckoo filter
Journal entries
Journal entries
Journal entries
Term N (latest)
Term N-1
Oldest term

Journal

- All fops are journal only mode except create
- Create is a write-through journal (log in journal + perform the fop in main store)
- Fops need to serve from journal
- Fops are first performed in the main store
- Based on the journal entries response will be altered

Journal

- Uses bloom filters
- Entries point to journal data
- Used to service reads (for consistency when writes are pending)
- One per term

Roll back

- Always roll forward
- If something fail, then invalidate the fop
- Invalidation has to be logged in majority of nodes

Reconciliation

- Separate process spawned
- Get information about terms from etcd
- Get information within terms from nodes
- Step through entries in order
- check for overlaps, discard any part that's no longer relevant
- figure out which replicas are in which state
- propagate from more current to less current
- mark entry as completed

Reconciliation

- When a term change happens we start reconciliation
- In most cases we will have only one term to reconcile
- In most cases reconciliation happens from leader

Reconciliation

- When all entries in a term are complete, term itself might be complete
- Exception: operations still completing locally (no fsync)
- "complete locally" and "journal replicated remotely" are separate
- there might never be a time when all replicas are up
- limit reduced-copy-count windows regardless

Log compaction

- We delete the terms once every node replicated the entries
- What if a node was down for days..
- Since it full data logging, the logs size would be huge
- We fall back to indexing mode

Future

- Future: fully log-structured (no "main store")`

Resources

- IRC
  - #gluster-dev
  - #gluster
- Mailing list
  - gluster-devel@gluster.org
  - gluster-users@gluster.org
- Design Doc
  - https://docs.google.com/document/d/1m7pLHKnzqUjcb3RQo8wxaRzENyxq1h1r385jnwUGc2A/edit?usp=sharing

Questions and/or Suggestions

A Journal Entry's Life Cycle

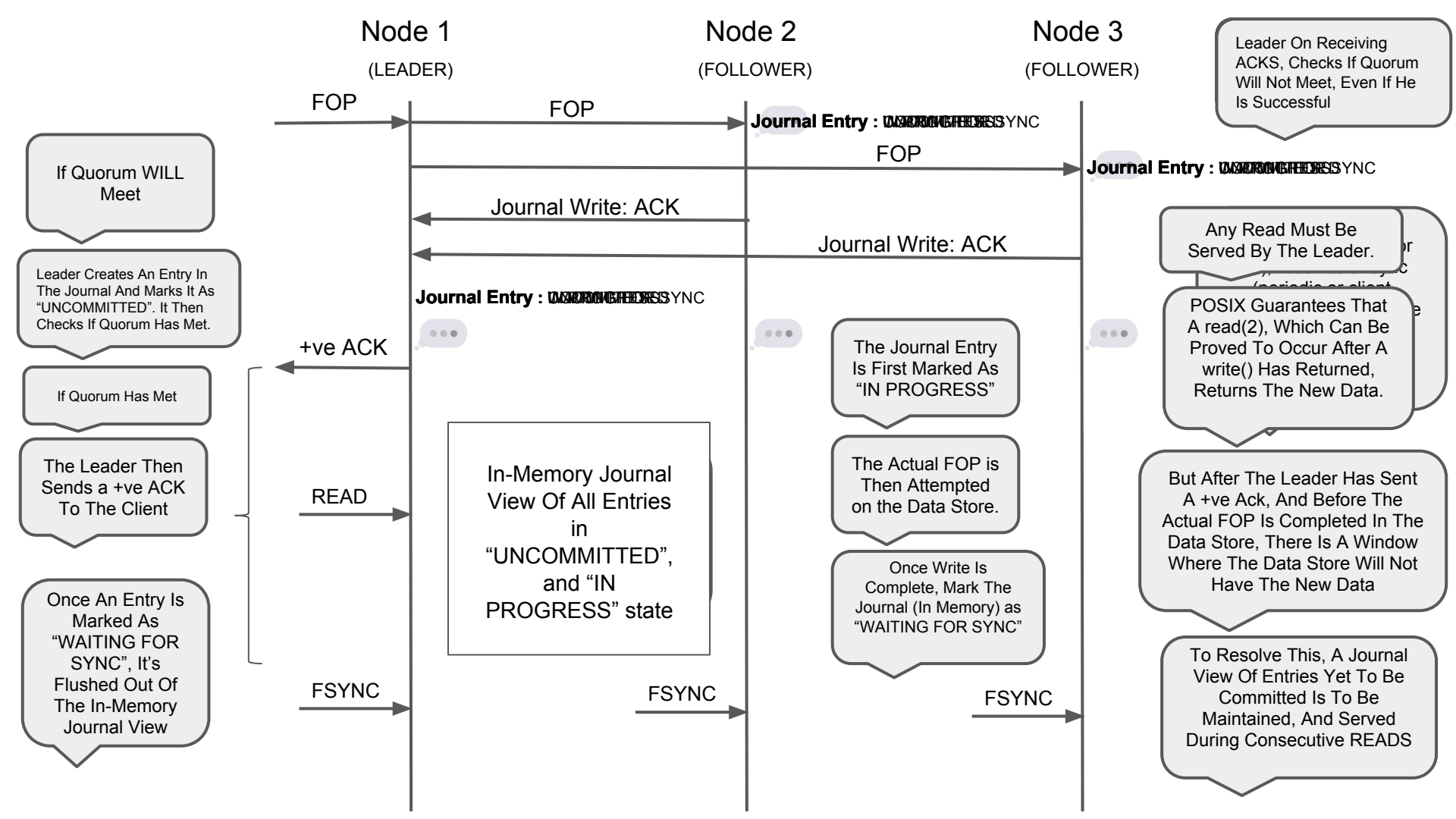
1. Uncommitted : This is the first state every Journal Entry is going to be in, when it's first introduced in the "state machine". This also means that this particular Journal Entry has not yet been acted upon and the actual fop is still pending.

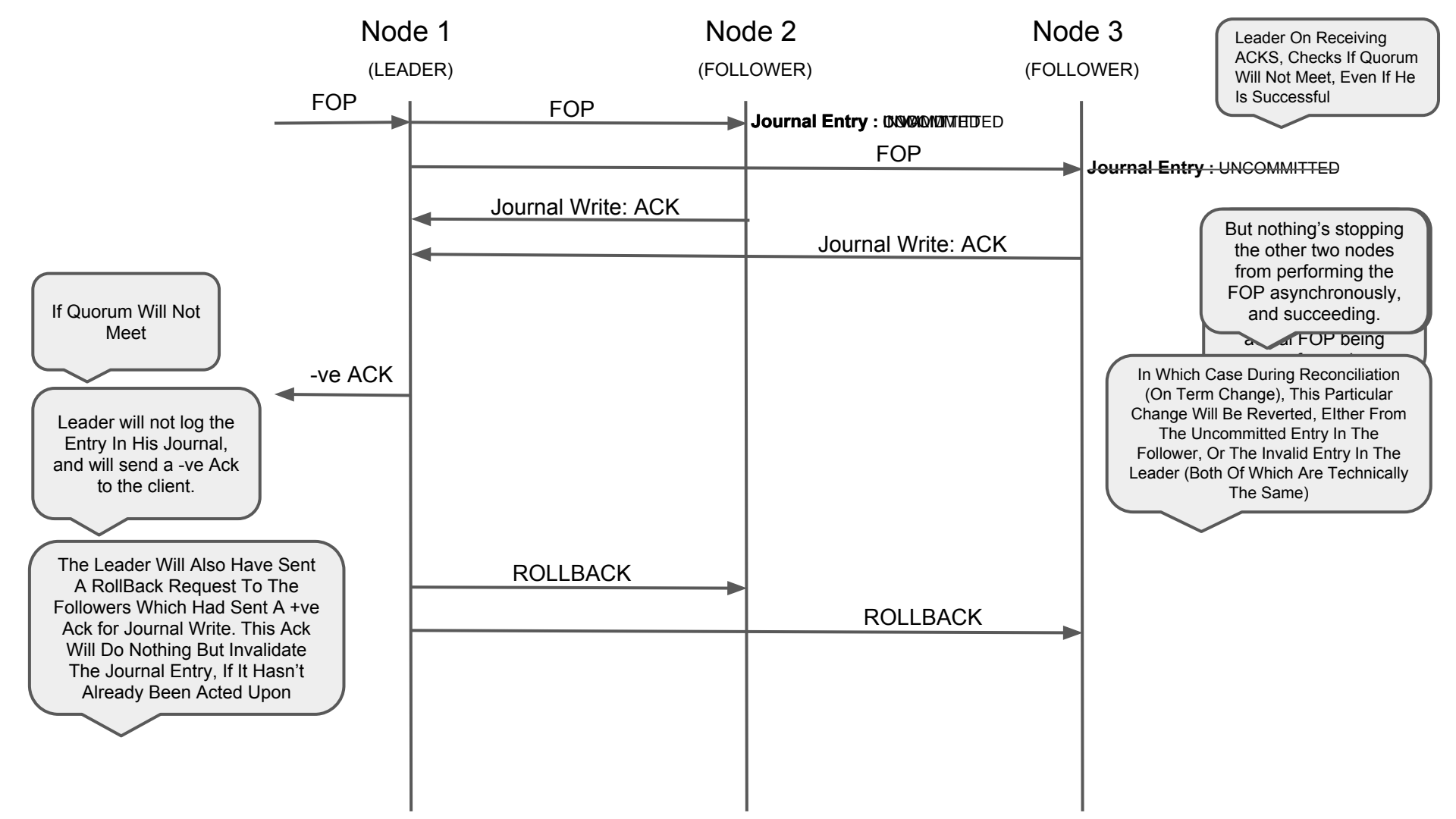
2. In Progress : This is the state that the Journal Entry is moved into, right before the actual fop is performed in the Data Store. This enables us to differentiate between a Journal Entry that has not yet been worked upon, from one that might be in any state of modification as part of the fop.

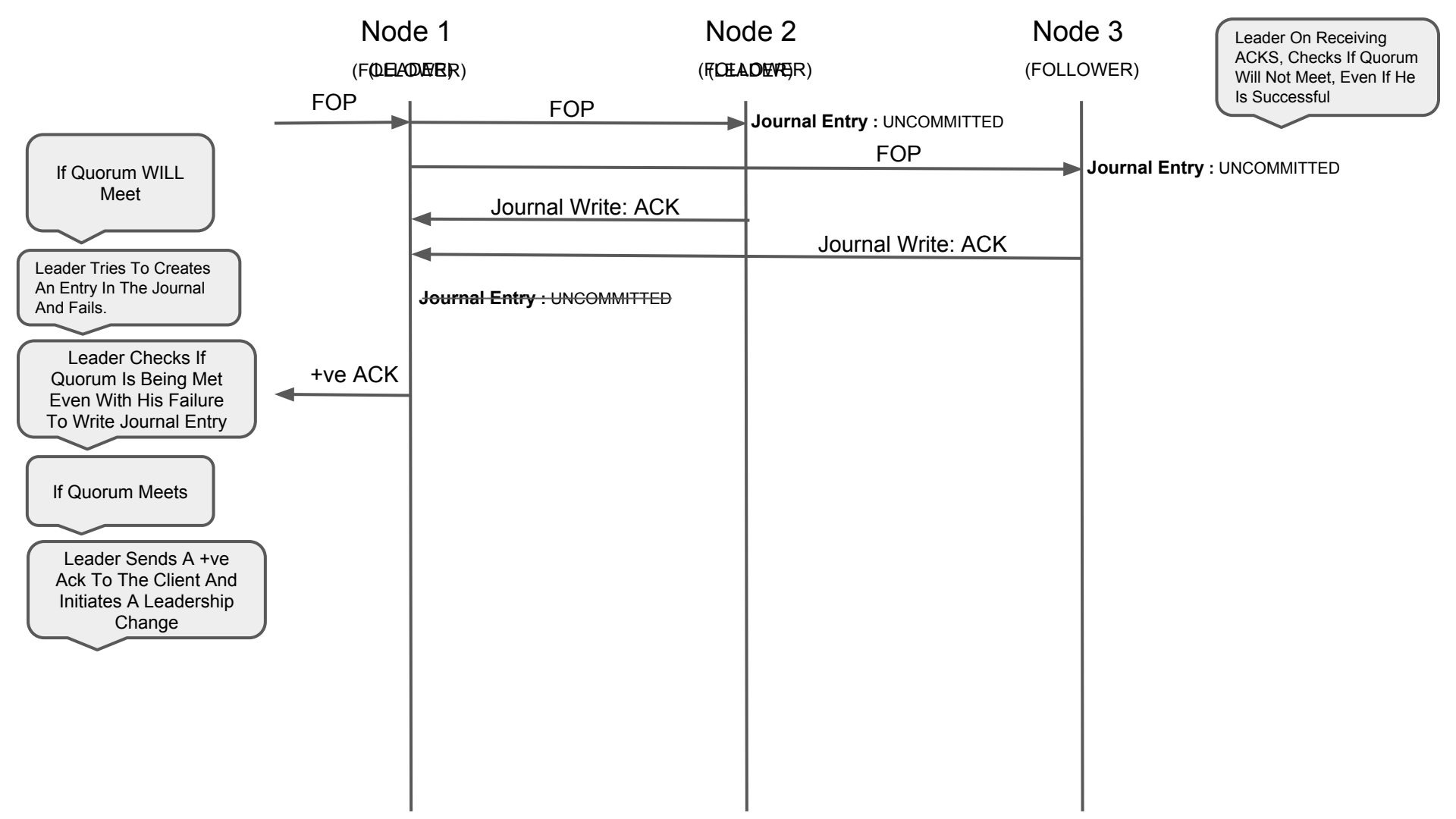
3. Waiting For Sync : This is the state where the Journal Entry will be moved to, once the actual fop is performed, but a fsync is still pending. This means that the data might or might not be in the disk right now, but the fop is successfully complete.

4. Committed : When a sync comes, all journals till that point, who were in "Waiting For Sync" state, are moved to "Committed" state. This completes the lifecycle of the Journal Entry.

5. **Invalid :** When a Journal is in Uncommitted state, and has not yet been acted upon, and a rollback request for the same comes, that particular entry is marked as "Invalid", suggesting that this particular Journal Entry will not be acted upon.


Node 1
(LEADER)
Node 2
(FOLLOWER)
Node 3
(FOLLOWER)
FOP
FOP
FOP
Journal Write: ACK
Journal Write: ACK
+ve ACK
READ
FSYNC
FSYNC
FSYNC
If Quorum WILL Meet
Leader Creates An Entry In The Journal And Marks It As "UNCOMMITTED". It Then Checks If Quorum Has Met.
If Quorum Has Met
The Leader Then Sends a +ve ACK To The Client
Once An Entry Is Marked As "WAITING FOR SYNC", It's Flushed Out Of The In-Memory Journal View
In-Memory Journal View Of All Entries in "UNCOMMITTED", and "IN PROGRESS" state
The Journal Entry Is First Marked As "IN PROGRESS"
The Actual FOP is Then Attempted on the Data Store.
Once Write Is Complete, Mark The Journal (In Memory) as "WAITING FOR SYNC"
Leader On Receiving ACKS, Checks If Quorum Will Not Meet, Even If He Is Successful
Any Read Must Be Served By The Leader.
POSIX Guarantees That A read(2), Which Can Be Proved To Occur After A write() Has Returned, Returns The New Data.
But After The Leader Has Sent A +ve Ack, And Before The Actual FOP Is Completed In The Data Store, There Is A Window Where The Data Store Will Not Have The New Data
To Resolve This, A Journal View Of Entries Yet To Be Committed Is To Be Maintained, And Served During Consecutive READS


Node 1
(LEADER)
Node 2
(FOLLOWER)
Node 3
(FOLLOWER)
Leader On Receiving ACKS, Checks If Quorum Will Not Meet, Even If He Is Successful
FOP
FOP
Journal Entry : UNCOMMITTED
FOP
Journal Entry : UNCOMMITTED
Journal Write: ACK
Journal Write: ACK
But nothing's stopping the other two nodes from performing the FOP asynchronously, and succeeding.
If Quorum Will Not Meet
-ve ACK
In Which Case During Reconciliation (On Term Change), This Particular Change Will Be Reverted, EIther From The Uncommitted Entry In The Follower, Or The Invalid Entry In The Leader (Both Of Which Are Technically The Same)
Leader will not log the Entry In His Journal, and will send a -ve Ack to the client.
ROLLBACK
ROLLBACK
The Leader Will Also Have Sent A RollBack Request To The Followers Which Had Sent A +ve Ack for Journal Write. This Ack Will Do Nothing But Invalidate The Journal Entry, If It Hasn't Already Been Acted Upon


Node 1
(FOLLOWER)
Node 2
(FOLLOWER)
Node 3
(FOLLOWER)
Leader On Receiving ACKS, Checks If Quorum Will Not Meet, Even If He Is Successful
FOP
FOP
Journal Entry : UNCOMMITTED
FOP
Journal Entry : UNCOMMITTED
If Quorum WILL Meet
Journal Write: ACK
Journal Write: ACK
Leader Tries To Creates An Entry In The Journal And Fails.
Journal Entry : UNCOMMITTED
Leader Checks If Quorum Is Being Met Even With His Failure To Write Journal Entry
+ve ACK
If Quorum Meets
Leader Sends A +ve Ack To The Client And Initiates A Leadership Change