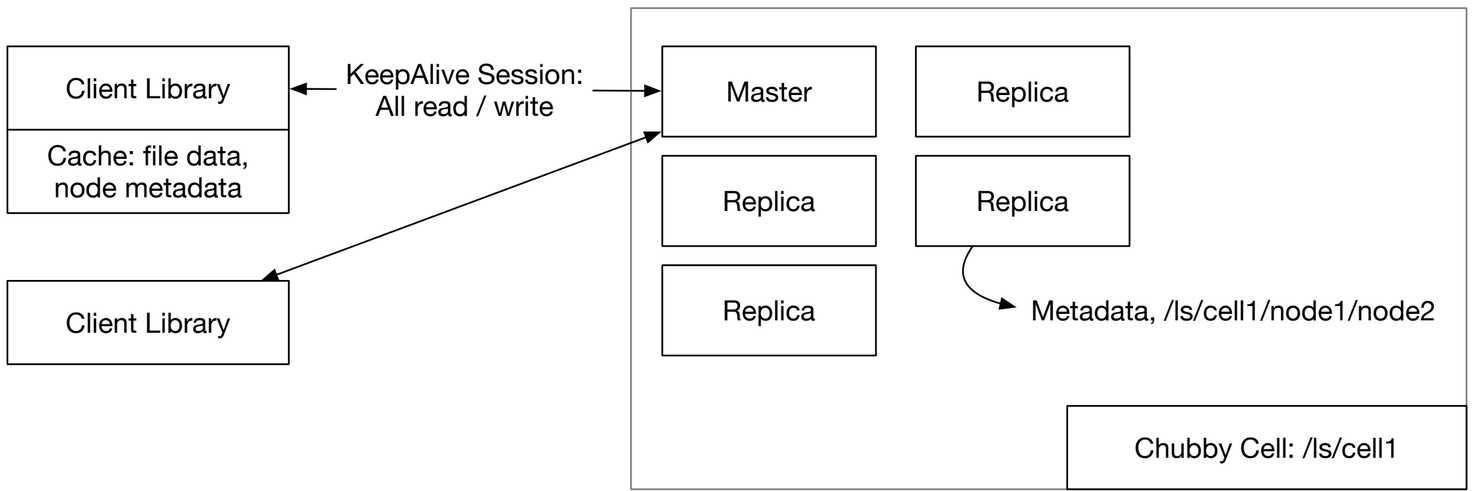
What does it do on a high level: “coordination”
- Allow clients to synchronize their activities (lock) and agree on basic information about their environment (small atomic file share) through building a centralized lock service

Premise:
- Engineering, not research
- Centralized lock service, distributed consensus algorithm could solve the same problem
- Coarse grained use (locks are held longer than fine grained time like sub second level); like electing a master that lives for hours; clients do not interact with chubby all that often

Rationale:
- Why not distributed consensus library
  - Prototypes started out without scaling concerns, code not structured for distributed consensus
  - Client needs something to put / get a small amount of data (consistent client-based caching as opposed to time-based caching)
  - Distributed lock and replicated state machine can achieve the same thing (?), but a lock interface is more familiar to programmers
  - Distributed consensus uses quorum and requires majority to be up to make progress, not suitable when number of replicas is small in the first place
- In what form does the lock manifest: serve small files
  - allow sharing data and config

Architecture:
- One cell contains 1 master, 4 replicas, master is elected among the 5 (using Paxos)
- Client only talks with master (discovered using DNS queries, if a client finds a replica, the replica redirects it to master), reads from master, and when writes, write succeeds after majority has it
- Hierarchical namespace. Each dir / file is a node, node can be ephemeral or permanent. Node access protected by ACL
- Advisory lock. Lock only conflicts with others’ attempts to acquire the same lock. The locked objects can be accessed by anyone holding or not holding the lock. Reader-writer lock, writers are mutually exclusive and exclusive to readers, readers can share
- When you create a handle (file descriptor), you can choose what events to listen to (file modified, child node modified, etc)
- Clients cache file data and node metadata. When node is changed, modification is blocked while master invalidates all the caches (no stale reads)
- Master failover

API:
- handle = open(string path), handle.readAll(), handle.writeAll() (atomic read and writes), handle.lock(), handle.unlock(), handle.updateACL().
- Note how read / write is different from lock / unlock (you need write access to be able to acquire a lock; *you don’t need to hold a lock in order to write something* (?))



Problem with distributed locking:
A holds a lock L, sends a request R, dies; B then acquires L, R then arrives, and operates on something that A no longer holds control of.
(Due to locks being advisory)

Solution:
- Lock cannot be held until timer expires if lock was released due to death
- Sequencer (a lease A sends over with each request essentially)