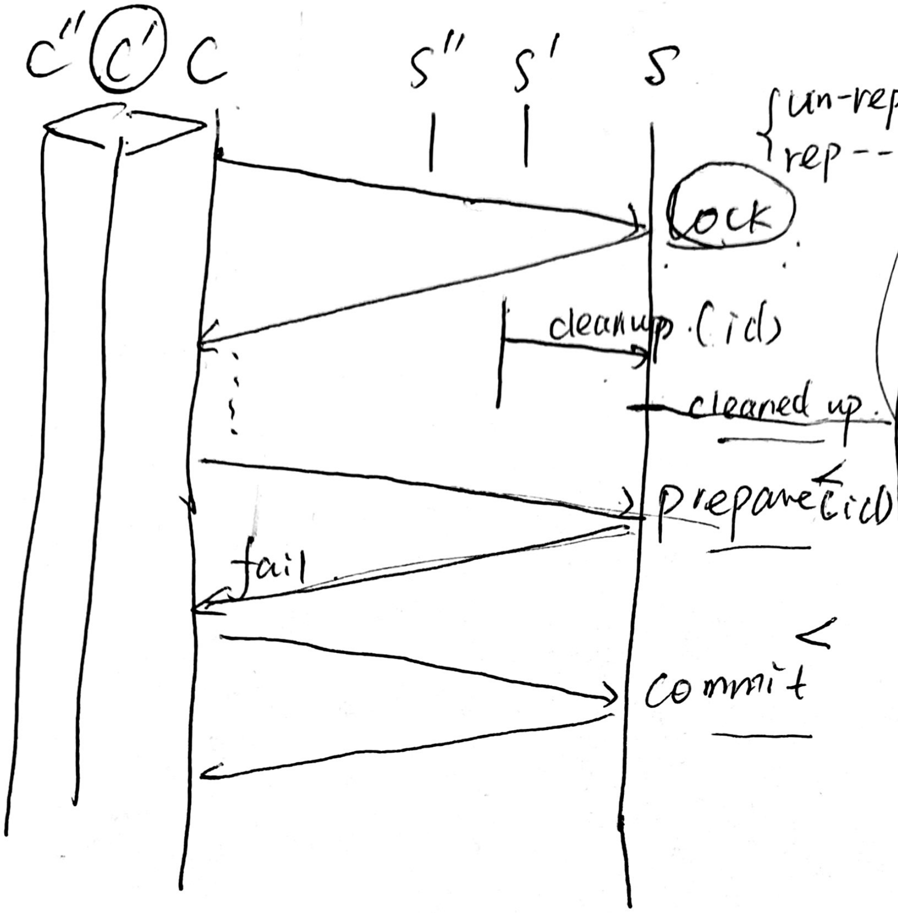


{un-rep --
{rep --

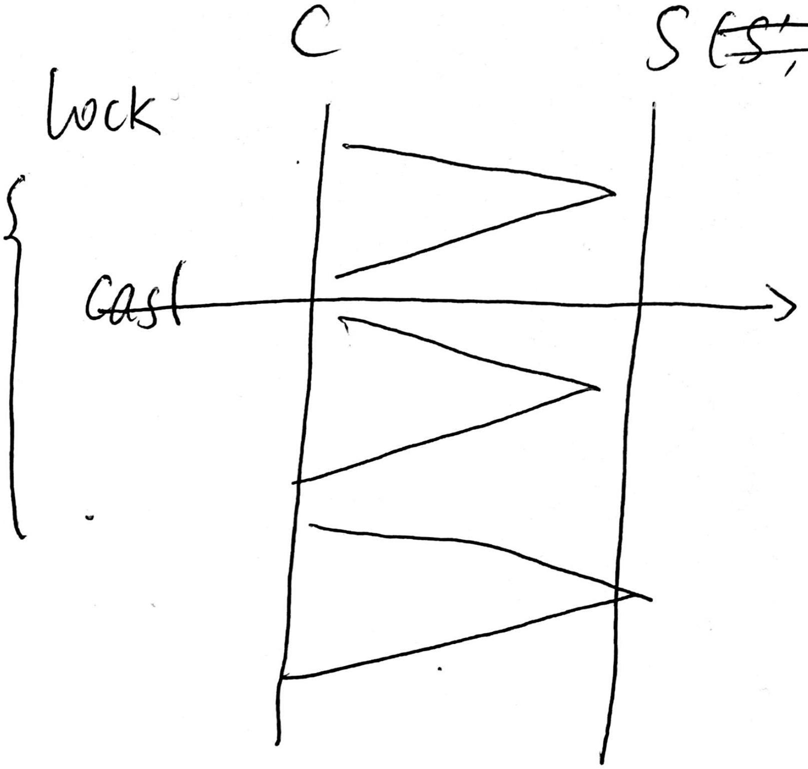
```
status_flag = ----
if (status_fag == unlocked)
    ------
elif (-----)

else
    wait . . .
```

atomically.

(get, put, cas)

S ~~(S', S'')~~ : Replicated, ~~R~~ Sharded.



lock

```
{
  ~~while(ret~~ do{
    cas ("lock-1", "0", "1")
                        #id
    ↓
  ret=OK / Fail
  } while (ret== Fail)
```

unlock

```
{ok-cas ("lock-1", id, "0")
```

```
lock (wound).

  ~~cas~~("lock-1-metalock" .... )
  id'←get ("lock-1")
   while ----
  if (id > id')
       cas("lock-1", "id'", id )
  else.
       ~~while~~

    cas("lock-1-metalock" ... )
```

Wait Die.

Txid1

id1 > id2.

Tx1 dies.

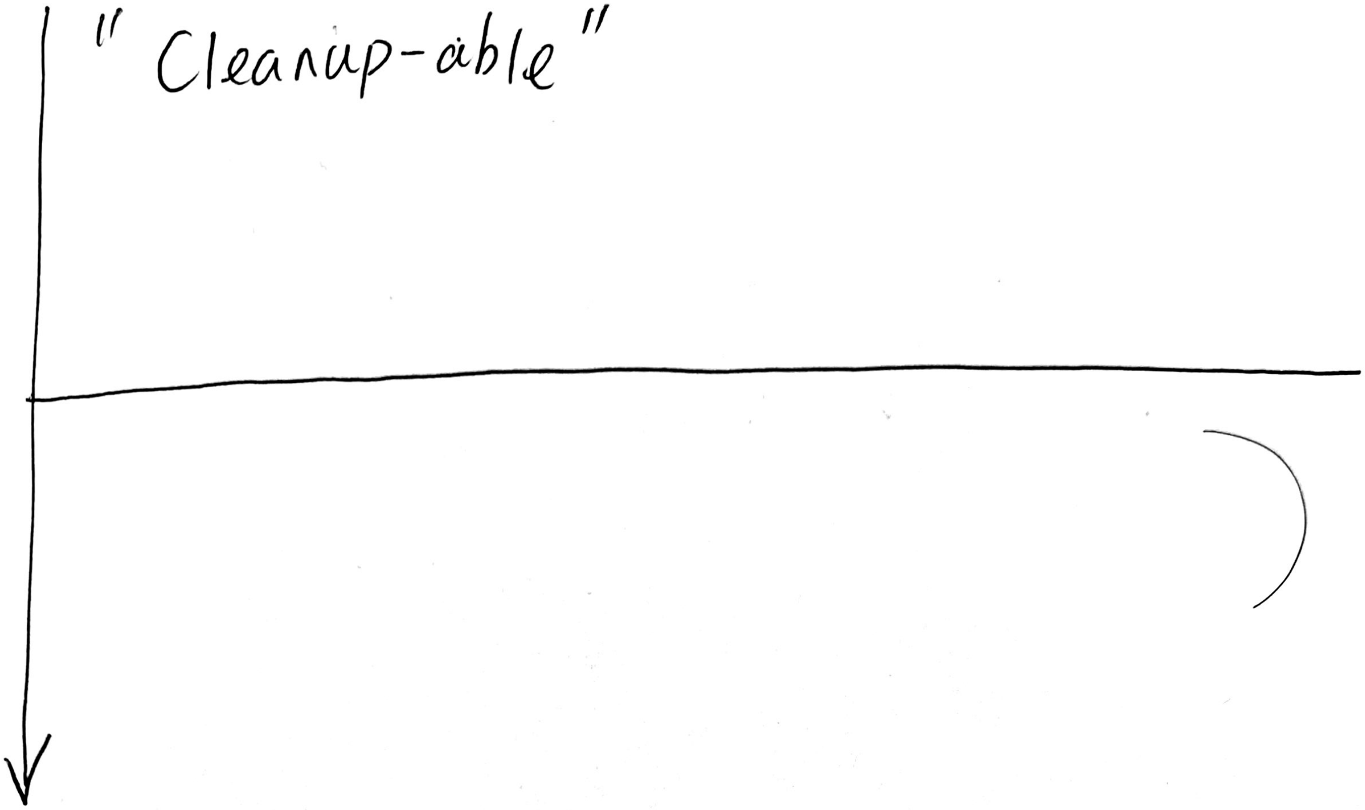
id1 < id2.

Tx1 waits.

```
lock (wait-die)

Loop:
  id' = get ("lock-1")
~~loop~~ if ( id > id')
       goto abort ( unlock ---- )
  if(id < id')
       ~~set~~ sleep(1); goto loop.
  if(id' == 0)
     ret: cas ("lock-1", 0, id) if ret== FAIL, goto loop
```

" Cleanup-able"



Client-id+ "state"

state

state += "lock-1"

cas ~~put~~ (client-id+ "state", state)

lock ("lock1")

state += ··· ; put --- → cas

lock (-----)

Client . Exeution. ~~update~~

state=get (~~stient~~-id+ "state")

cas (--- , state, "~~—~~")

state+" "clean-up"

cas (- - - - "emty")

Commit / abort / cleanup

cas (state ····)

~~Commit / abort~~

(~~cleanup~~)

reentrant.

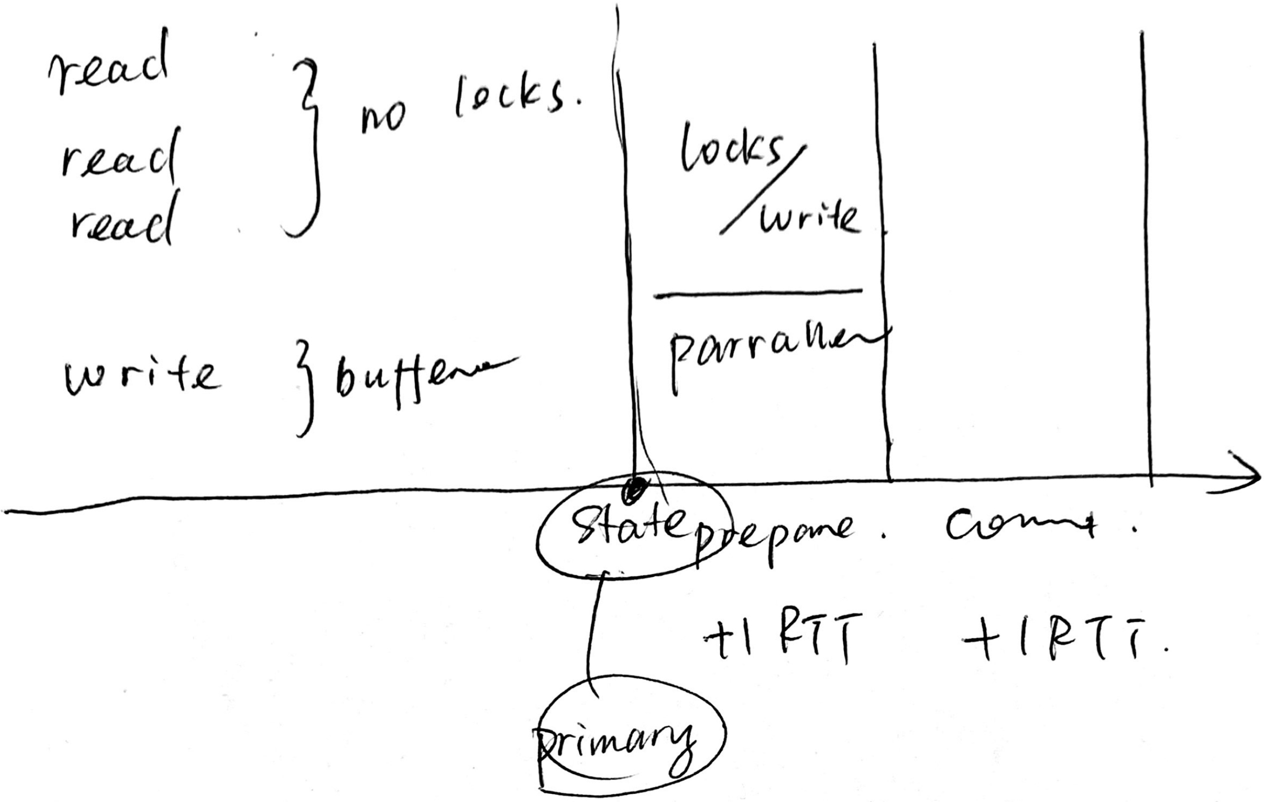
" Cleanup-able."

↓

" harmless "

e.g. persist change due to locks.

{ not much wasted space.
the system can proceed safely.



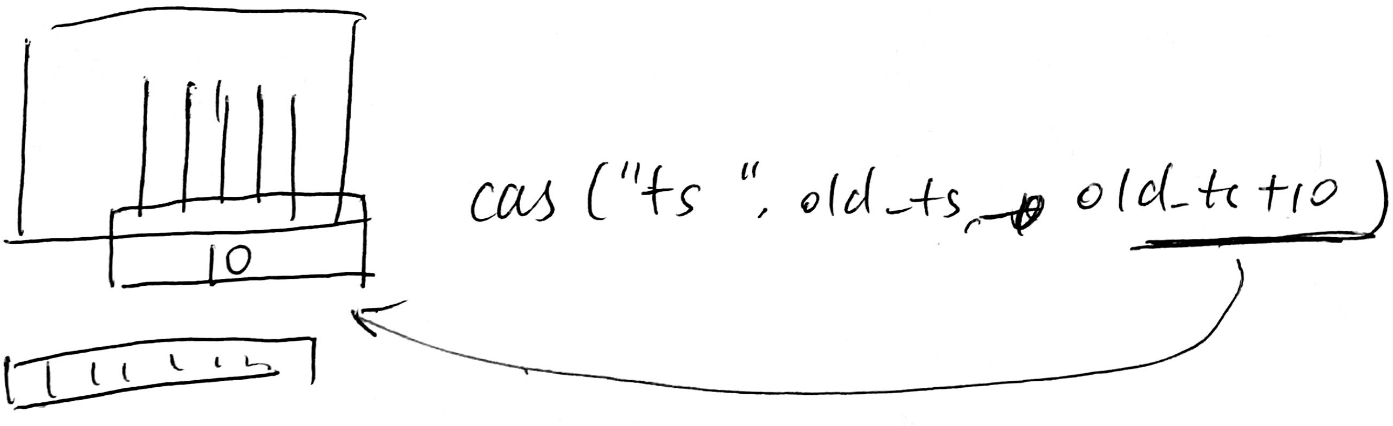
timestamp.

cas ("ts", old_ts, old_ts+1)

↓

Fail.

Problem : Performance



cas ("ts", old_ts, old_ts+10)