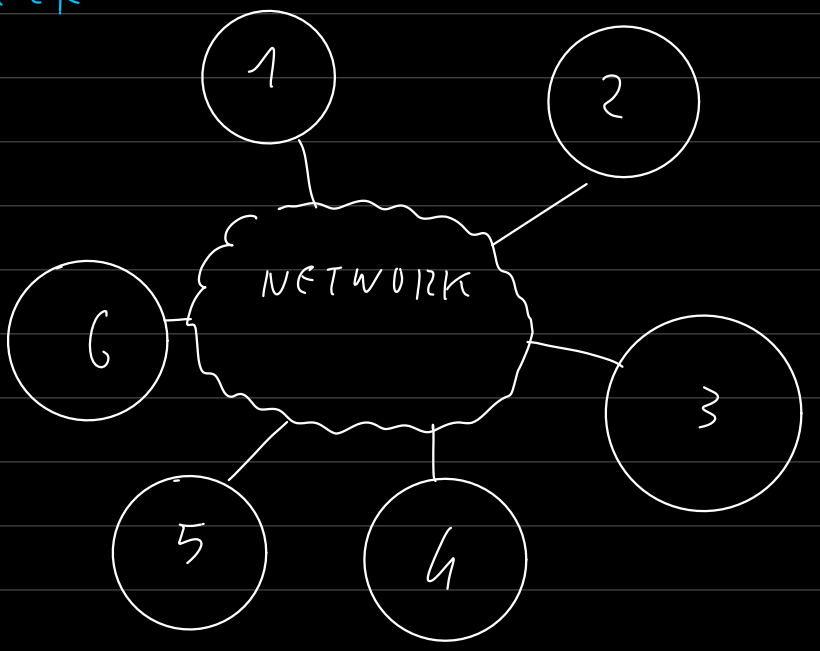
RAFT - ARCHIVIATION

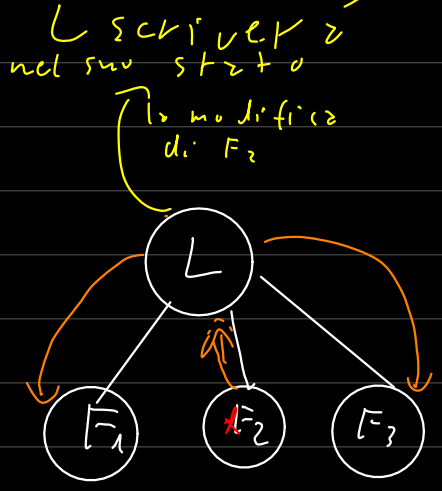
OBBIETTIVI:

1- CREAZIONE SISTEMA DI ARCHIVIAZIONE DISTRIBUITO SENZA CENTRALIZZAZIONE (CLUSTER)
2- UPLOAD FILES (write)
3- DOWNLOAD FILES (read)
4- DELETE FILES (delete)

---

1) CLUSTER



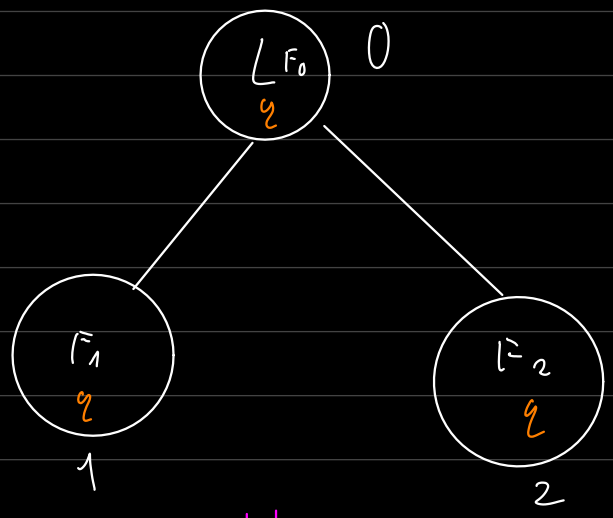


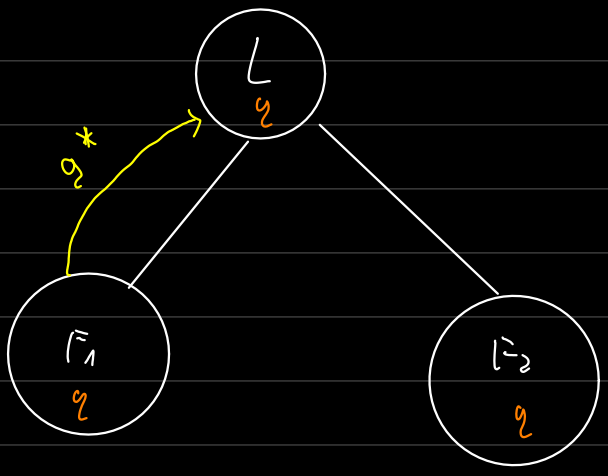
INVARIANTI:

1- Tutti i nodi convergono allo stesso stato (a un certo punto avranno gli stessi dati)

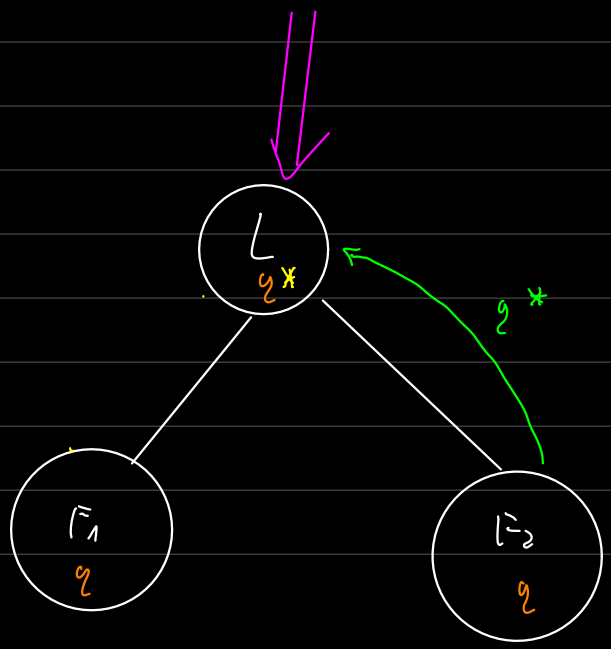
⟹ RAFT

CRITICITA'

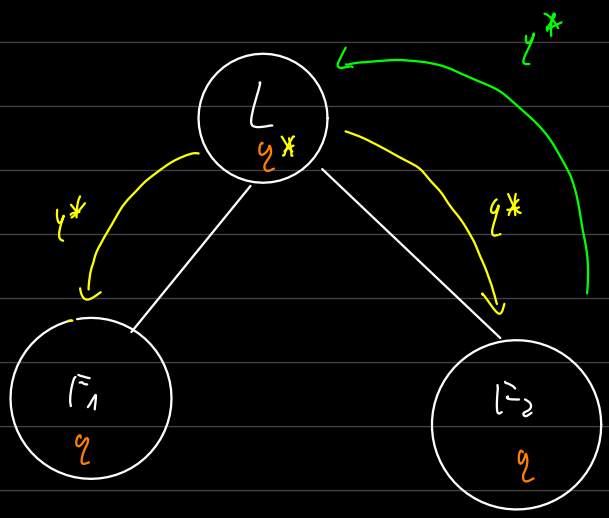


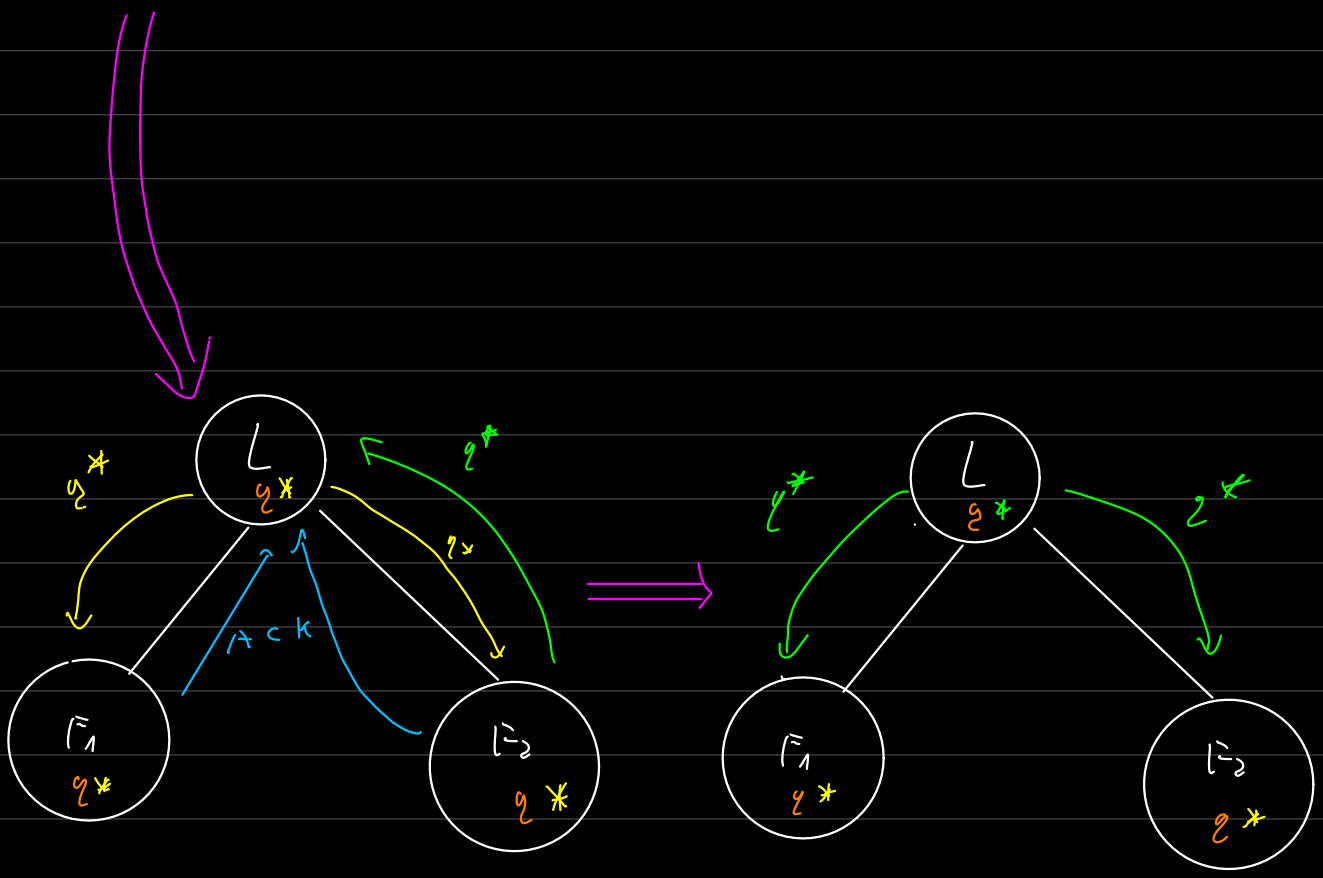


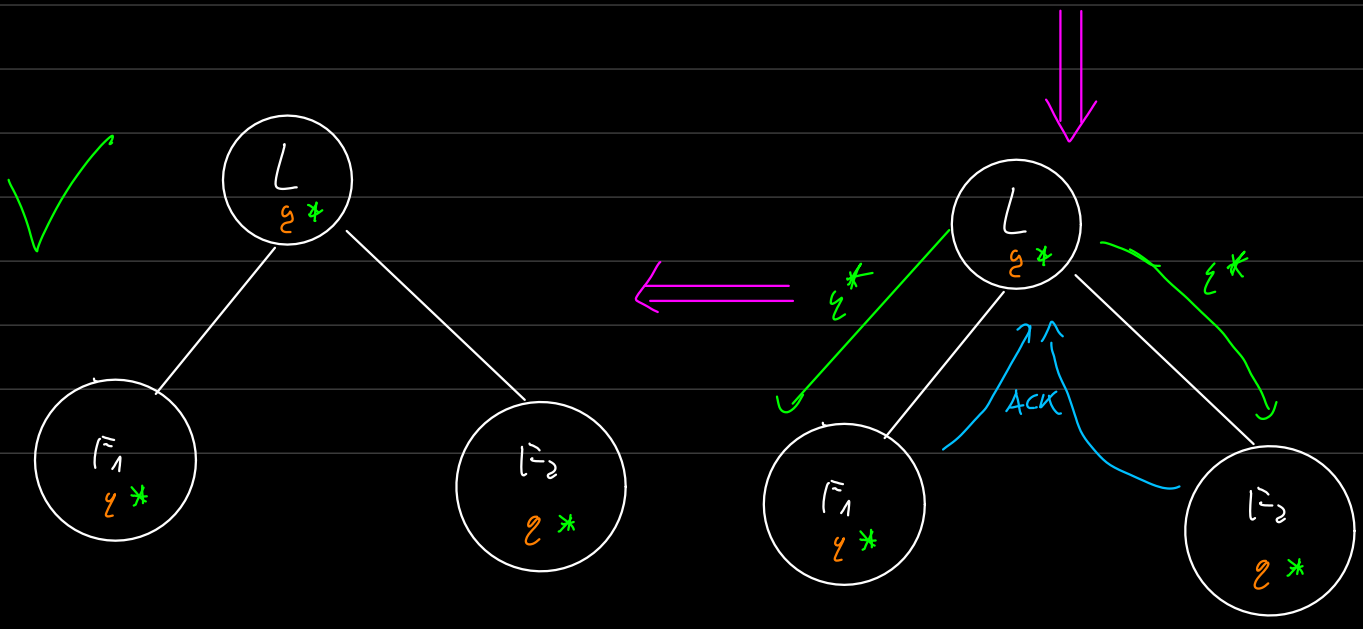
g^* = "hello pluto"



g^* = "hello paperino"


L
q*
q*
q*
q*
F1
q
F2
q


L
q*
q*
q*
q*
Ack
F1
q*
F2
q*
L
q*
q*
q*
F1
q*
F2
q*


L
q*
F1
q*
F2
q*
L
q*
q*
q*
Ack
F1
q*
F2
q*

Definizione dei nodi:

- VIRTUAL MACHINE:

  Mantengo forte isolamento dei nodi, mantengo indipendenza dei nodi, mantengo potenziali grosse differenze tra i nodi, non ho, possibilmente, dipendenze comuni tra i nodi.

Comunicazione:

- Trammite HTTP:
  - formato invio dati: JSON

STORING:

- NO FILESYSTEM DISTRIBUITO
- ogni nodo, internamente, deve avere un filesystem per funzionare
- lo STATO interno di ogni nodo è una directory con all'interno tutti i file dove ogni file avrà un ID che verrà generato a partire dalla hash di:
  virtual directory di storing + nome del file

LEADER ELECTION:

Ogni nodo ha un ID

⟹ evoluzione di Bully algorithm:
ogni nodo ha due timeout:

1- timeout heartbit Leader: unico per tutti
2- timeout invio richiesta elezione: random su ogni nodo.
↳ indica quanto tempo un nodo deve attendere prima di inviare una richiesta di elezione

Poi ogni nodo:

- se non ha indetto un elezione accetta e invia a tutti l'ID della persona che vota. Ogni nodo conta chi ha più voti e quando c'è una maggioranza il nuovo leader è eletto

- Per essere eletto ogni nodo deve essere in uno stato consistente. Per verificare lo stato il candidato al momento della candidatura deve essere accettato da una maggioranza di nodi i quali dovranno verificare che lo stato del candidato sia aggiornato almeno quanto il loro.

- MESSAGGIO DA INVIARE:

TERM : indice numerico
init = 0,
incrementa monotonicamente
si incrementa nel tempo
($TERM_C$ must be $\geq$ $TERM_G$)

ID : Candidate id

LOGICAL CLOCK
||
serve a individuare info obsolete (ex. leader morti).

LAST LONG (LLI) : INDEX — indice dell'ultima entry del candidato

LAST LOG (LLT) TERM — TERM dell'ultima entry del candidato

intero: come work log ss solo DIO

LEADER (già presente):

situazioni di operatività del cluster:

cluster stabile: nessun nodo del cluster vuole modificare lo stato del cluster e ogni nodo ha lo stato conforme

⇒ invia heartbeat (append entry with no data) invia messaggio di aggiornamento dello stato senza dati da modificare.
(lo fa in periodi morti quando quindi non c'è attività nel cluster)

cluster instabile: a) il leader ha modifiche di stato da propagare gli altri nodi sono stabili.

⇒ invoca RPC append entry e alleg a alla chiamata la entry da replicare. il suo stato è per ora volatile. Una volta che la maggioranza dei follower hanno ricevuto la modifica il leader aggiorna il suo stato persistente.

come faccio a sapere se hanno ricevuto?

La RPC appendentry ha un valore di ritorno:

{

TERM_FOLLOWER : CURRENT TERM del follower che il leader per aggiornare se stesso.

BOOL SUCCESS : TRUE/FALSE

}

il TERM value nella RPC APPEND ENTRY è usato sia in input sia in output e viene usato dai nodi per aggiornare il proprio TERM se quello ricevuto è maggiore del proprio

!!! TOCHECK !!!

CASI CRITICI:

1) FOLLOWER CRUSH e non fornisce valore di ritorno di RPC: Leader riprova a mandare la chiamata RPC fino a che la maggioranza dei follower hanno memorizzato la entry.