Algoritmos de consenso distribuído

Edward Ribeiro
(@edward_ribeiro)

$ whoami

- Servidor Público
  - Serpro, STM, TSE, CL-DF, Senado (atualmente)
- Professor Universitário
- Pesquisador na área de sistemas distribuídos (UnB)
- Contribuidor (eventual) de software livre
  - Cassandra, Solr, Kafka, ZooKeeper, VoltDB, etc
- Software Engineer - DataStax (2014/2015)
  - DSE Search

O que são algoritmos de consenso distribuído?

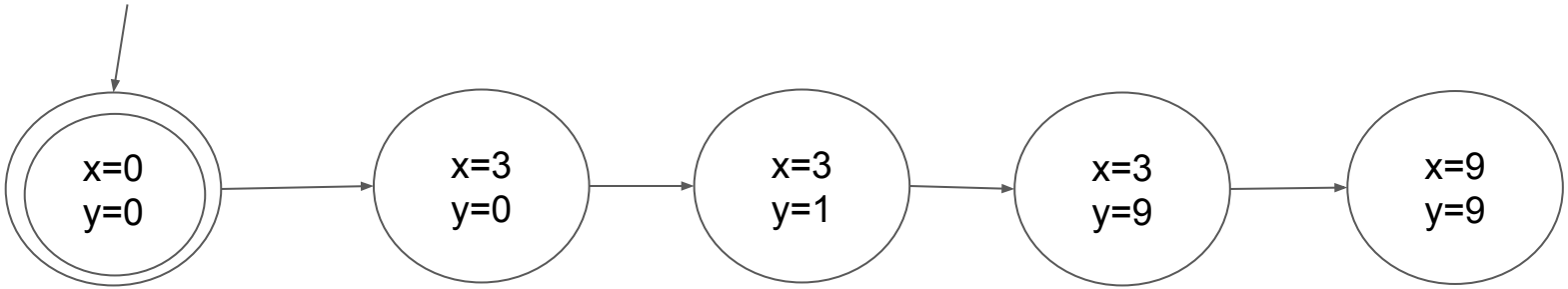
- ***Algoritmos que permitem a um grupo de processos chegar a um acordo em relação a um valor;***
- Permite **coordenar** processos distribuídos;
- Surgem no contexto de máquinas de estado replicadas;
- Tais algoritmos possuem um papel crucial na construção de sistemas distribuídos em larga escala e confiáveis;

Quando usar?

- ***Quando for necessário que processos compartilhem uma visão consistente, atômica, e ordenada de uma série de operações/eventos;***
  - Alta disponibilidade (Tolerância a falhas);
  - Desempenho (Distribuir carga entre milhares de clientes);

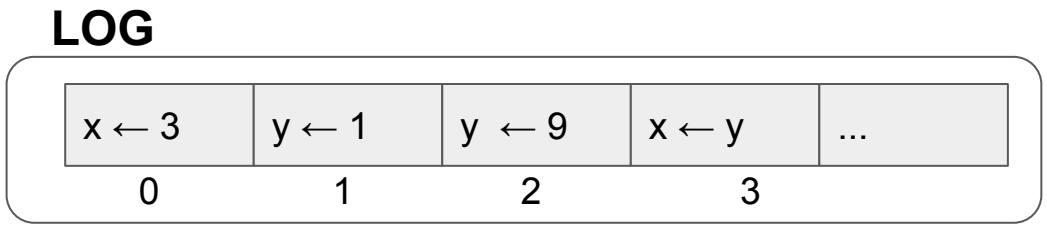
Máquinas de estado replicado

- Descrevem as mudanças de estado **incrementais** através de operações de escrita;
  1. Os clientes enviam comandos para os nós concorrentemente;
  2. Os nós chegam a um acordo acerca do próximo estado;
  3. O estado é replicado para outros nós (alta disponibilidade)



Máquinas de estado replicado

- Geralmente implementada como log replicado entre nós;
  - Cada entrada armazena um comando;
  - Replica-se para tolerância a falhas;



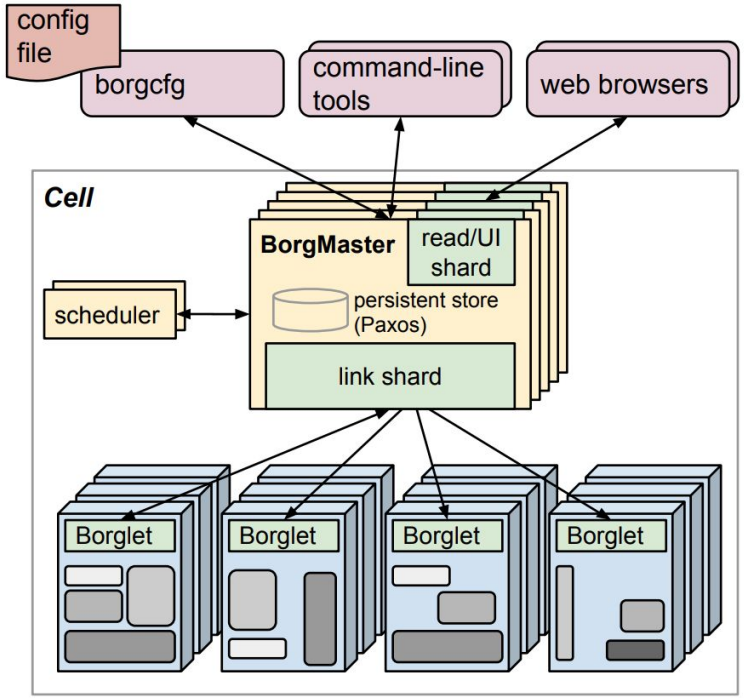
- O trabalho do algoritmo de consenso é manter a consistência do log replicado;

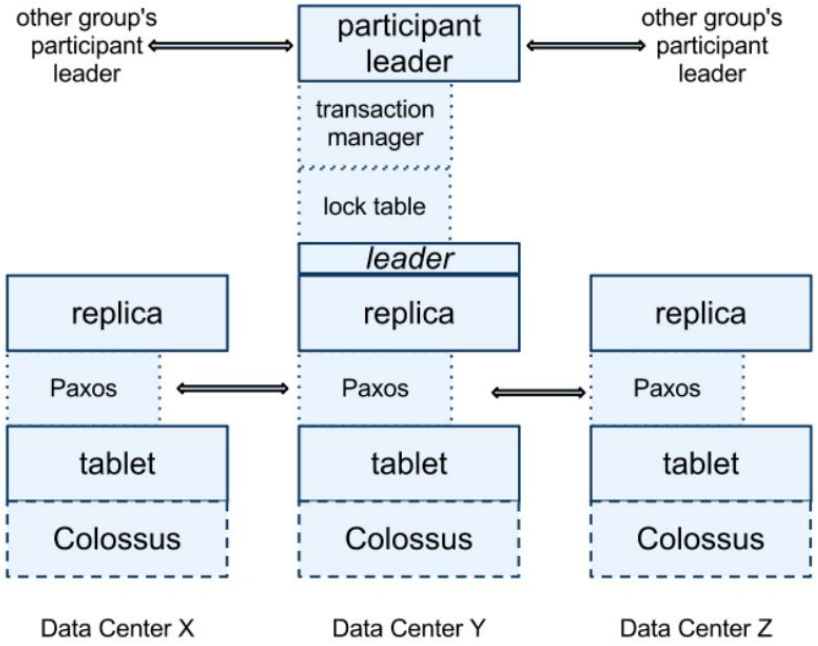
Replicações

- Replicação Ativa
  - Todos os nós implementam uma máquina de estados finitos
    - Acordam entre si (quórum) sobre a ordem das operações e as efetivam localmente;
- Replicação Passiva
  - Um nó é designado **líder**, que *recebe todos os comandos de escrita*;
    - O nó líder efetiva operações quando obtiver quórum com a maioria das réplicas;
    - O nó líder é responsável por replicar os estados para as outras réplicas;

Paxos

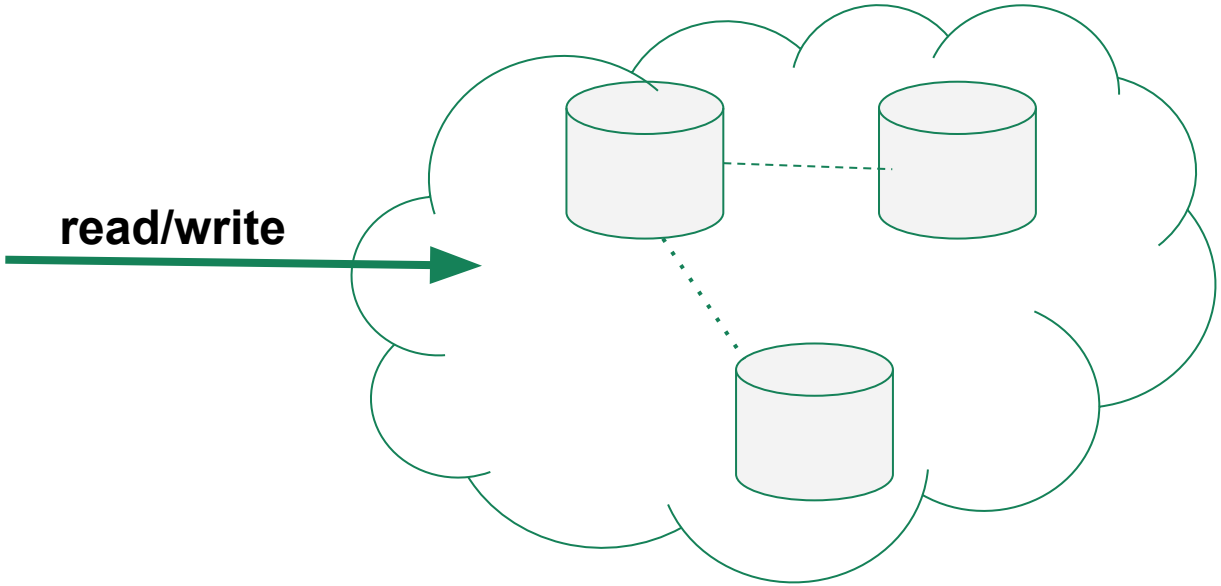
- O mais proeminente algoritmo de consenso distribuído;
- Protocolo criado por Leslie Lamport em 2001;
- ***Notoriamente difícil de entender e implementar;***
- Família de algoritmos: Paxos, Multi-paxos, egalitarian-paxos, etc.
- As implementações diferem muito do algoritmo original:
    - Requer mudanças complexas para suportar sistemas em produção;
    - Pressupostos e lacunas no algoritmo original;


config file
borgcfg
command-line tools
web browsers
Cell
BorgMaster
read/UI shard
persistent store (Paxos)
scheduler
link shard
Borglet
Borglet
Borglet
Borglet


other group's participant leader
participant leader
other group's participant leader
transaction manager
lock table
leader
replica
replica
replica
Paxos
Paxos
Paxos
tablet
tablet
tablet
Colossus
Colossus
Colossus
Data Center X
Data Center Y
Data Center Z

ZooKeeper

- Um ***serviço*** de ***coordenação*** de processos distribuídos open source escrito em Java (parte da suíte Hadoop);



Pra que é utilizado?

- Coordenação de processos distribuídos
  - Eleição de líder;
  - Armazenamento de configuração;
  - Descoberta de recursos;
  - Notificação de eventos;
  - Filas de trabalho;
  - etc

ZooKeeper & SolrCloud

- Todos os nós do SolrCloud conectam-se ao ZooKeeper (“cérebro” do cluster);
- Armazena configuração dinâmica do cluster (shards, replicas, live_nodes, configsets, etc);
- Ao mudar a configuração (ex: máquina sai do ar), notifica os servidores através de watches.
- Notificação de falhas em nós;

ZooKeeper & Kafka

- Membership do cluster Kafka
- Configuração de tópicos (lista de tópicos, partições, réplicas, líder da partição, etc);
- ACL (Access Control Lists);
- Notificação de falhas em nós;
- etc

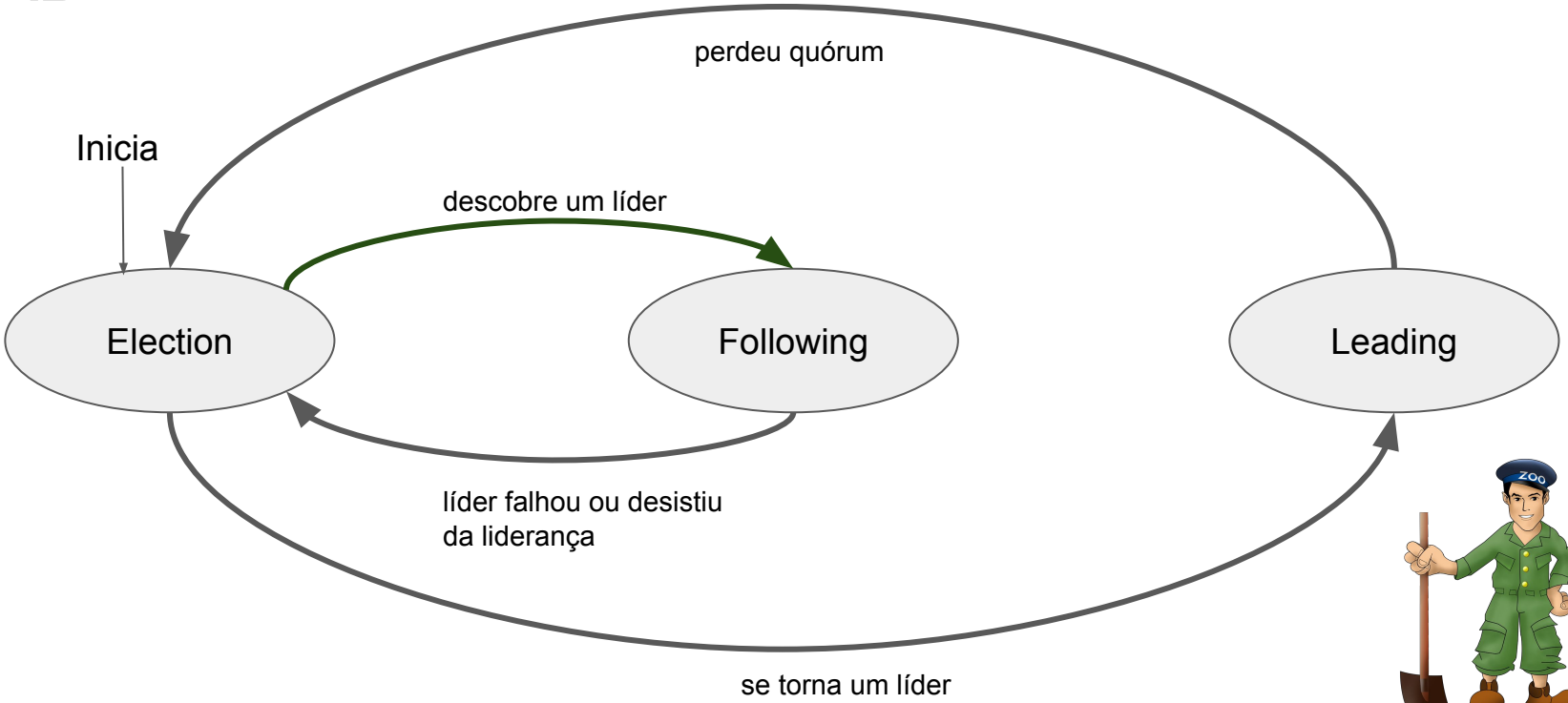
HBase, BooKeeper & ZK

- Mantém os metadados do HBase e BK;
- Que servidores estão ativos;
- Notifica sobre falhas em nós;
- etc

ZAB - ZooKeeper Atomic Broadcast Protocol

- Implementação (em Java) fortemente entrelaçada ao projeto;
- Replicação passiva;
  - Elege-se um líder com a história mais recente (a partir de um quórum de nós)
- Três papéis:
  - Leader
  - Follower
  - Observer
- Um follower ou observer somente se conectará a um leader por vez;
- Um leader também é um follower;

ZAB



Observers não participam de eleição

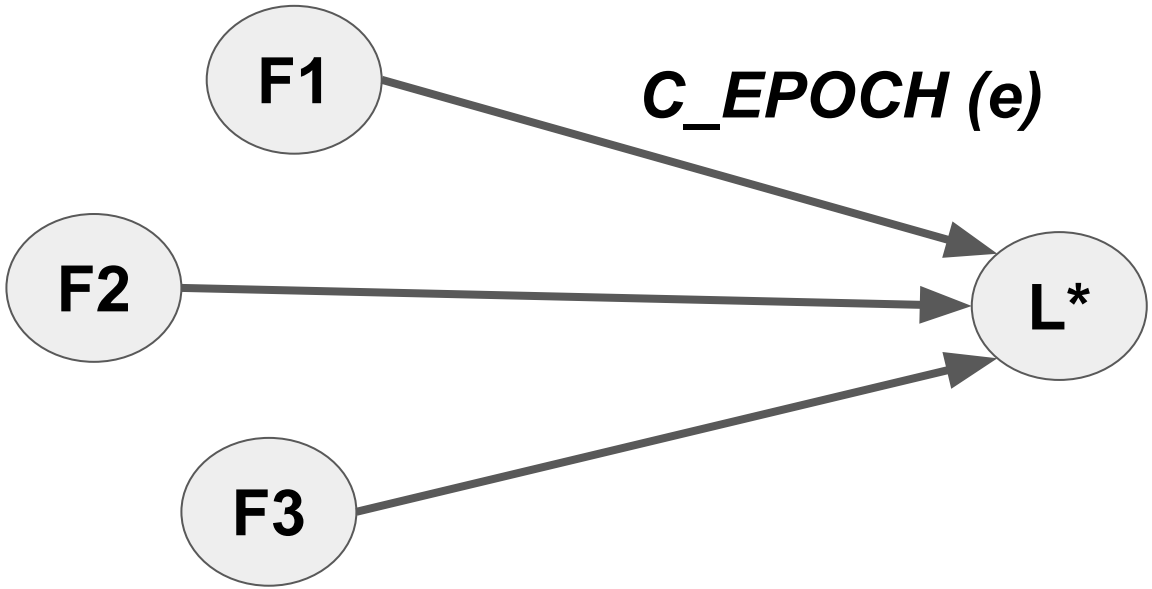
ZAB

- Algoritmo possui três fases:
  - *Fase 1 - Descoberta*
  - *Fase 2 - Sincronização*
  - *Fase 3 - Broadcast*
- Transações são identificadas por zxid, que são números monotonicamente crescentes;

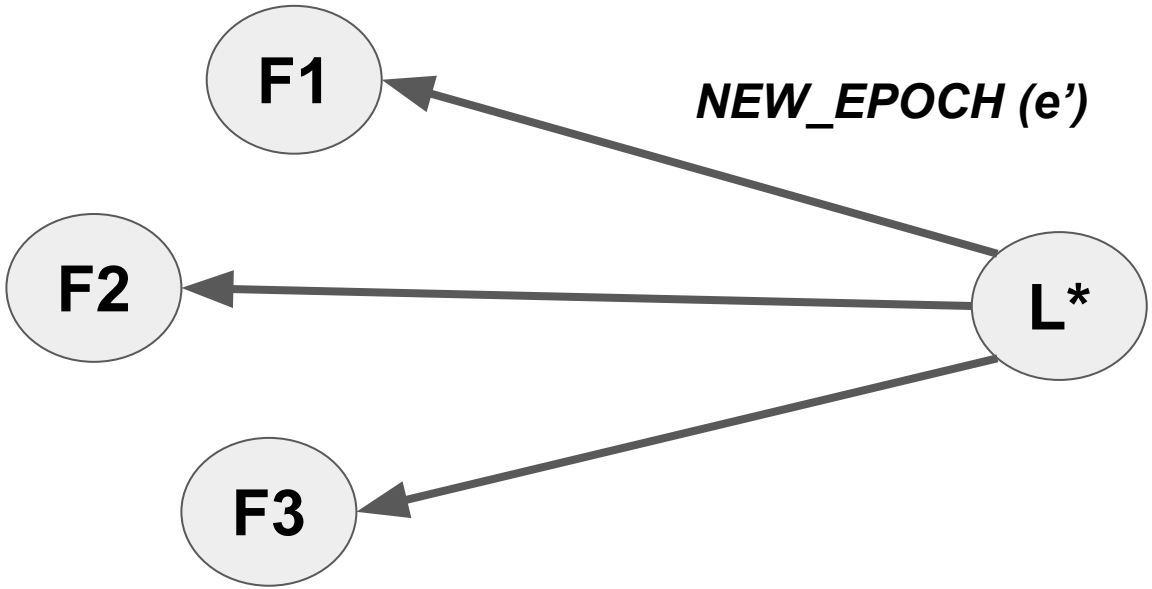
ZXID

EPOCH	COUNTER

ZAB, Fase 1 - Descoberta

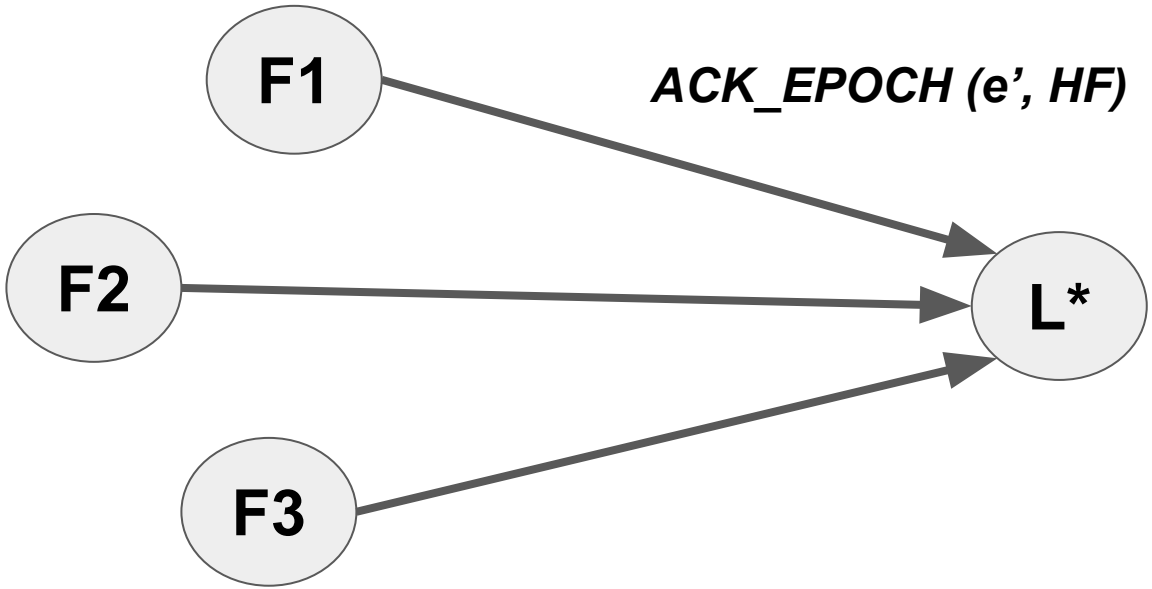


ZAB, Fase 1 - Descoberta

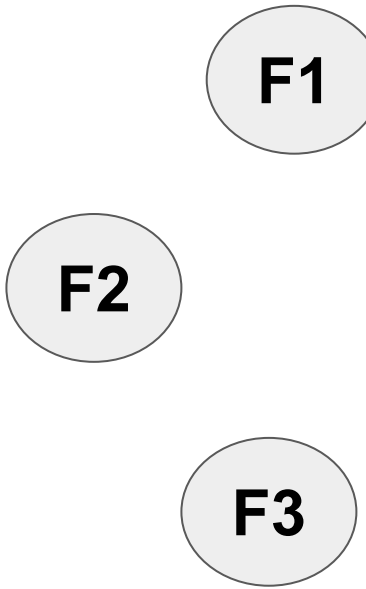


e' maior que qualquer outro epoch recebido dos followers

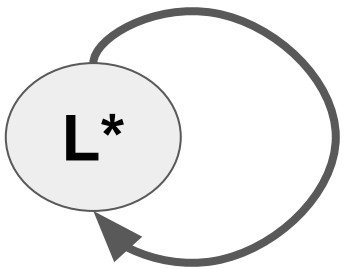
ZAB, Fase 1 - Descoberta



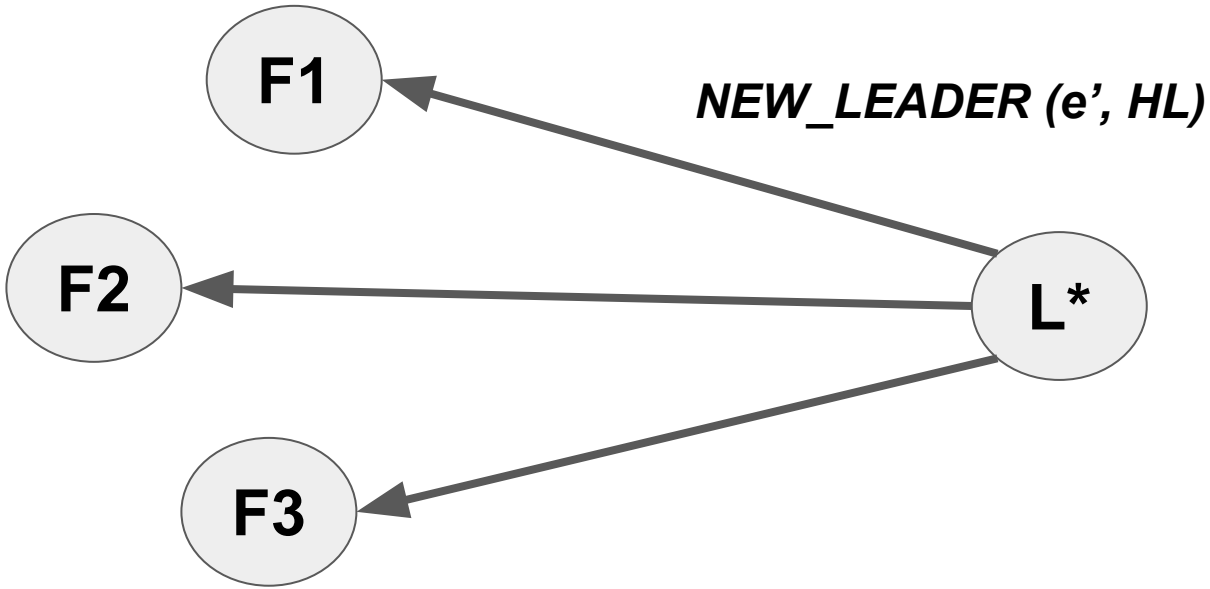
ZAB, Fase 1 - Descoberta



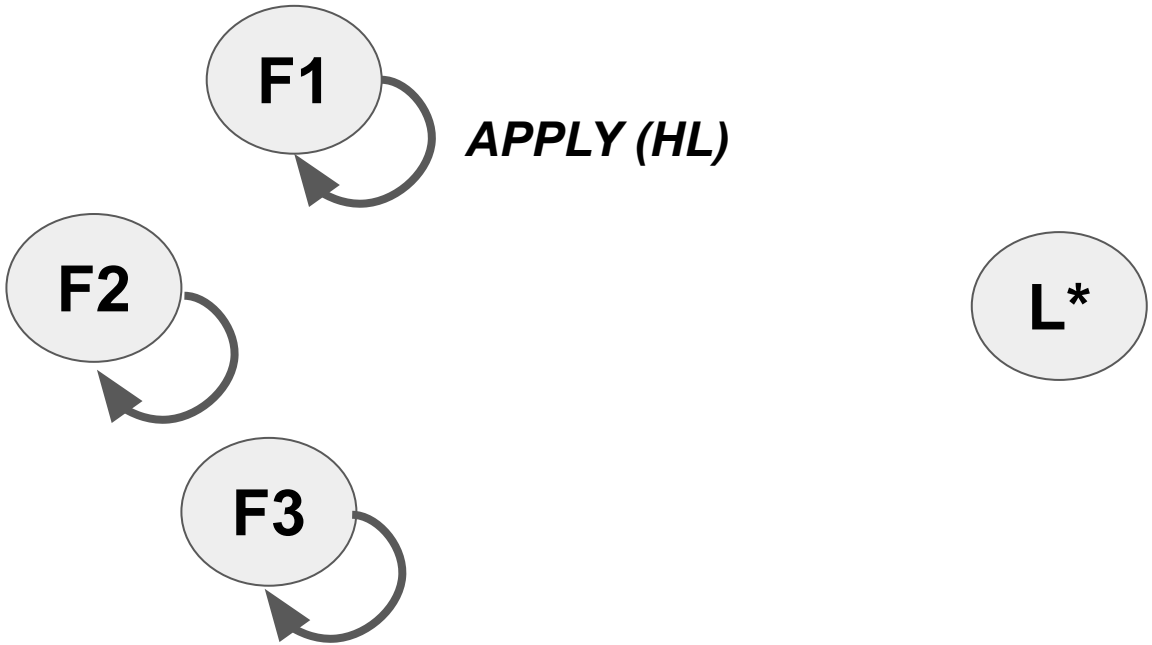
Escolhe a história H mais recente dentre os followers



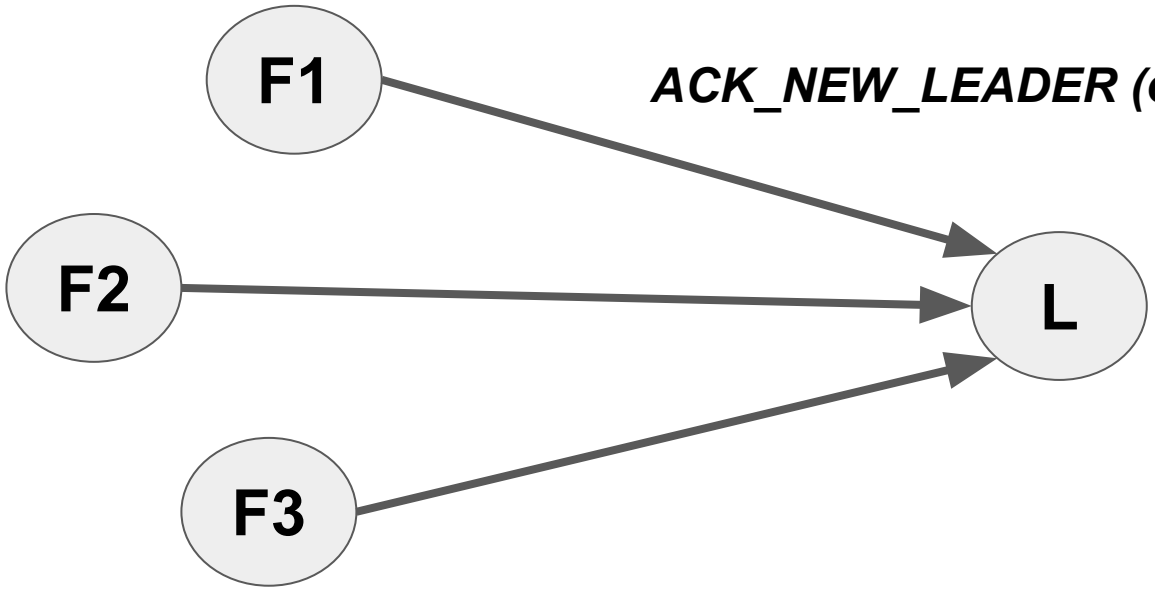
ZAB, Fase 2 - Sincronização



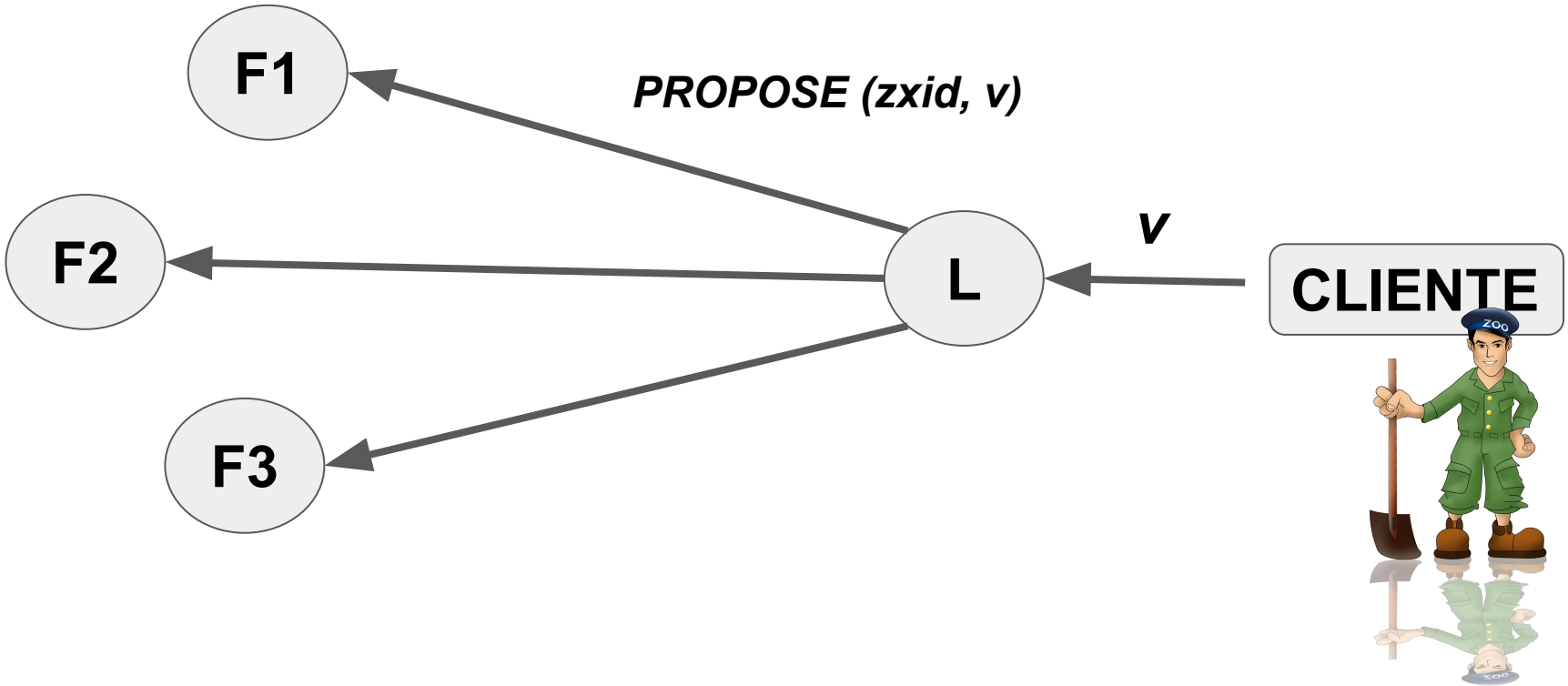
ZAB, Fase 2 - Sincronização



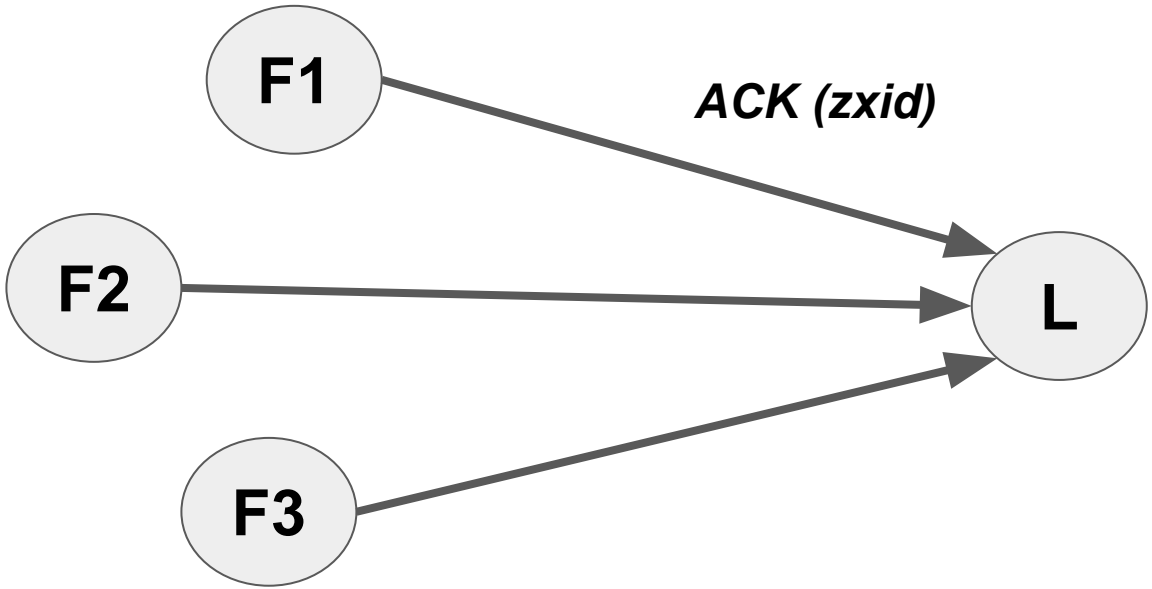
ZAB, Fase 2 - Sincronização



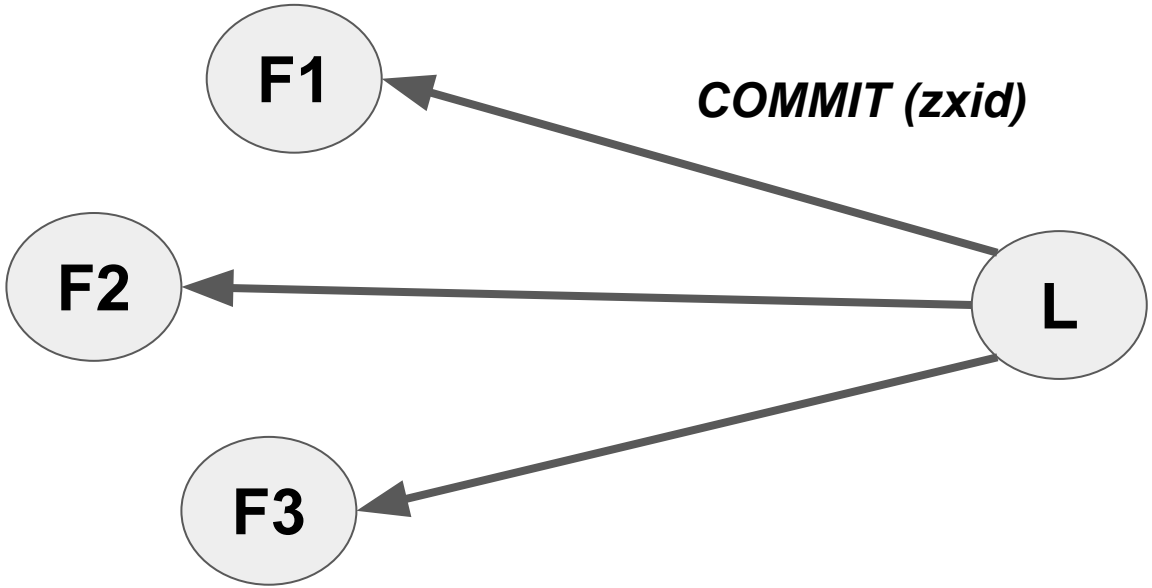
ZAB, Fase 3 - Broadcast



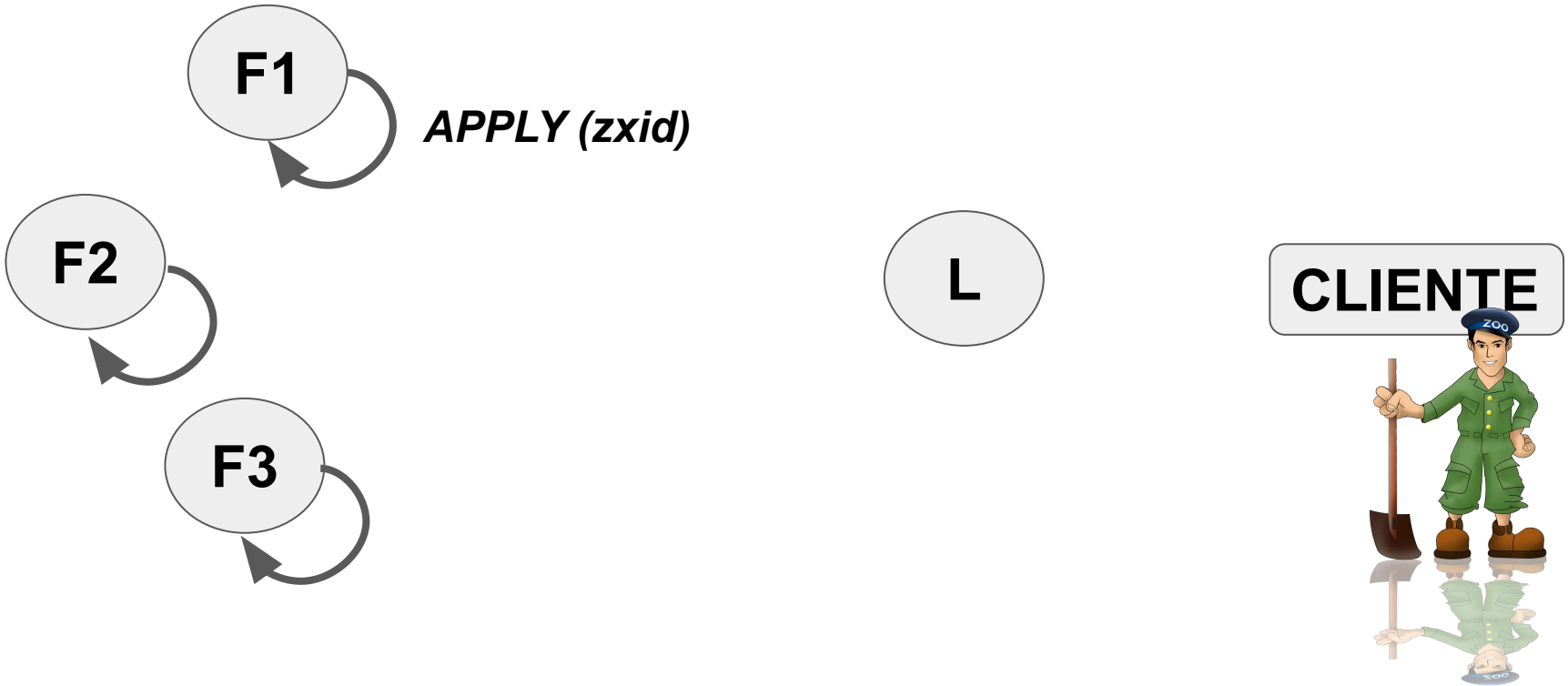
ZAB, Fase 3 - Broadcast



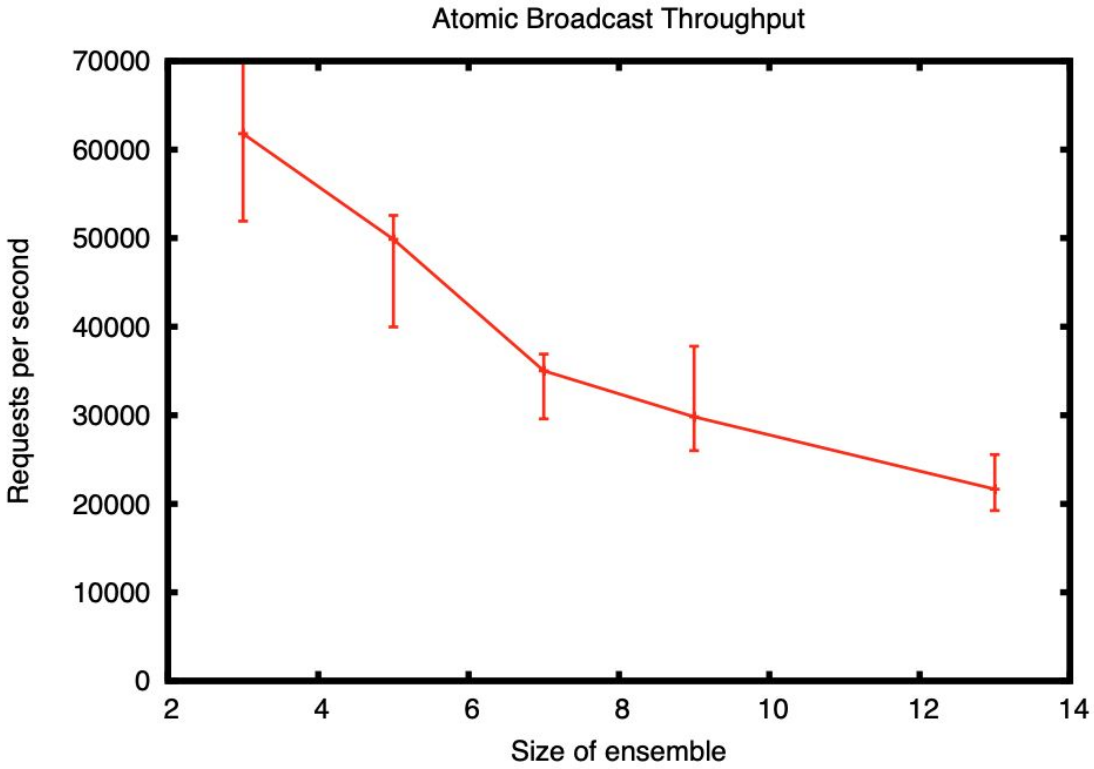
ZAB, Fase 3 - Broadcast



ZAB, Fase 3 - Broadcast



Limitações


Atomic Broadcast Throughput
Requests per second
70000
60000
50000
40000
30000
20000
10000
0
2
4
6
8
10
12
14
Size of ensemble

Limitações

- 3 e 5 são quantidades típicas de servidores ZK/Raft;
  - Cinco servidores suportam 2 falhas simultâneas (quórum = N/2 + 1)
- A escalabilidade horizontal é limitada
  - Quanto mais nós, mais mensagens trocadas para atingir consenso
- Ainda sujeito a split brains e indisponibilidade (e.g. GC do ZooKeeper);
- Monitore e faça ajuste fino, se necessário, do Garbage Collector (ZooKeeper);

ZooKeeper

1. ZooKeeper **não é um banco de dados!**
   a. **Mas possui um BD interno sincronizado entre os nós do cluster.**

2. ZooKeeper **não é um sistema de arquivos!**

   a. Mas o modelo de dados é como um sistema de arquivos hierárquico (tipo UNIX), **residente em memória RAM.**
   b. Não utilize ZK para armazenar dados, e sim meta-dados

Melhores práticas

- ZK e etcd utilizam RAM para armazenar as estruturas de dados, monitore esse uso;
- Configurar o máximo de conexões simultâneas (throttling);
- Separar logs de transação de snapshots em discos diferentes;
- **Use nós dedicados para ZK/etcd (evite co-alocar com outros containers, serviços, principamente que usam I/O e rede intensamente);**

Melhores práticas

- Ajuste os parâmetros de heartbeat timeout, election timeout, session timeout, de acordo com o deployment (on premise, cloud);
- Limpe periodicamente os diretórios de snapshots e logs transacionais (se autopurge não estiver habilitado);
- Monitorar latência de I/O em disco, espaço ocupado, taxas de erro, etc;

Anti-padrões

- Usar como o banco de dados (NoSQL) genérico
- Adicionar muitos nós ao serviço (3, 5, 7 são números típicos em produção);
- Usar para serviços de escrita intensos (razão leitura/escrita < 1);
- Usar um único serviço para múltiplos clientes (e.g., mesmo cluster ZK para HBase, Kafka, SolrCloud);

Anti-padrões

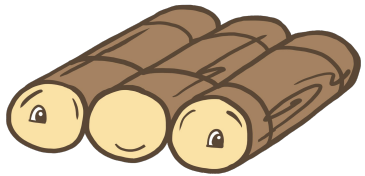
- Não monitorar o uso de disco, rede e CPU dos nós;
- Colocar os nós com processos de CPU e I/O intensos;
- Não acompanhar as issues/tickets do projeto;
- Não acompanhar o tamanho dos snapshots em disco;

Raft

- Algoritmo de consenso para gerenciar um log replicado;
- Equivalente a Paxos
- ***Mais fácil entendimento que Paxos*;***
- ***Projetado em torno de log replicado;***
- ***Logs não podem ter buracos;***
    - Mantém consistência entre logs;
- https://raft.github.io/

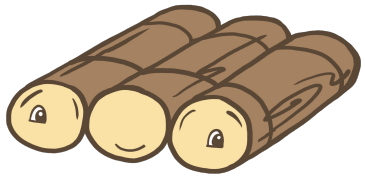
Raft

- Cada servidor deve estar em um de três estados:
  - Líder
    - somente um líder por vez;
    - ***trata todas as requisições de clientes (leitura e escrita)***
  - Follower
    - todos os servidores não líder;
    - são totalmente passivos
      - somente respondem a requests de líderes e candidatos;
  - Candidate
    - usado para eleger um líder;
- Os servidores se comunicam através de RPCs.

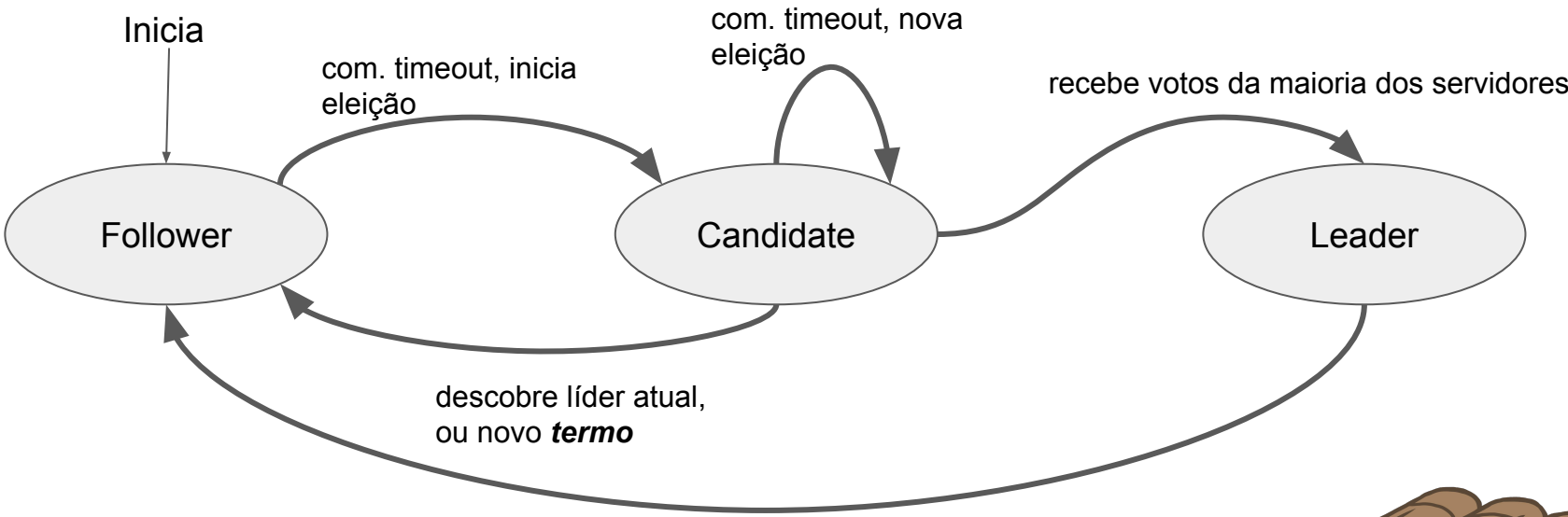


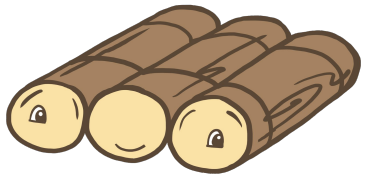
Raft - Propriedades

- Líder forte
  - As entradas no log somente fluem do líder para outros servidores;
- Eleição de líder
  - Utiliza temporizadores randômicos para eleger líderes;
    - resolve conflitos de forma simples e rápida;
- Mudanças de membership
  - Permite ao cluster continuar operando normalmente durante mudanças de configuração;



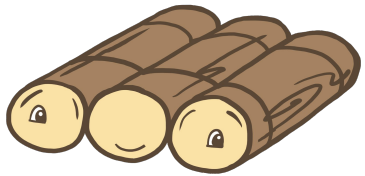
Raft - Máquina de estados





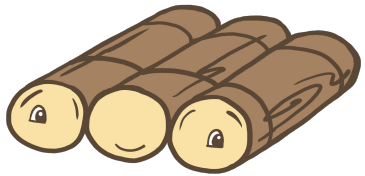
Raft - Termos

- Raft divide o tempo em ***termos*** de tamanho arbitrário;
- Termos são inteiros numerados consecutivamente;
- Cada termo começa com uma eleição;
- O candidato eleito serve como líder do resto do termo;
- Se a eleição não tiver um vencedor (eleição falha), o termo termina sem líder;
  - Nova eleição começa com novo termo;
- Raft garante que existe somente um líder por termo;

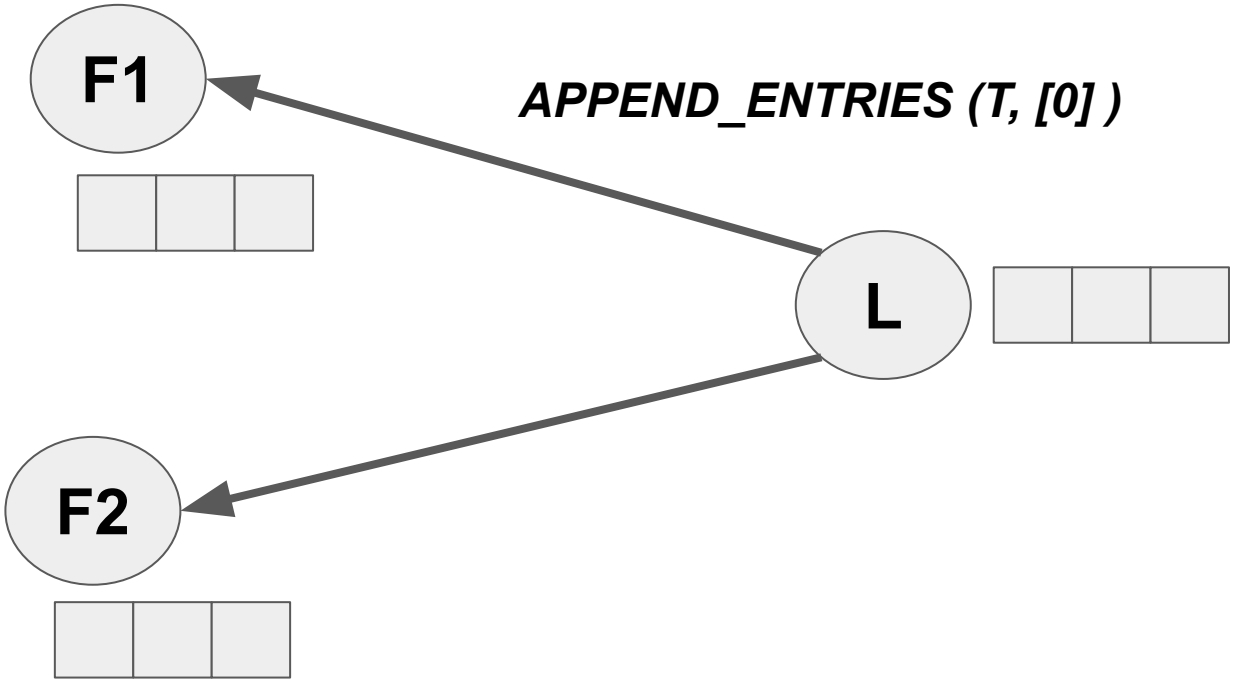


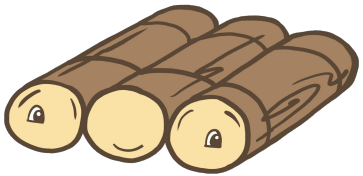
Raft - Protocolo

- 1. Elege um distinguished leader;
- 2. Leader aceita entradas de logs dos clientes, replica entre os servidores;
- 3. Leader informa aos servidores que é seguro aplicar estas entradas a suas respectivas máquinas de estado;

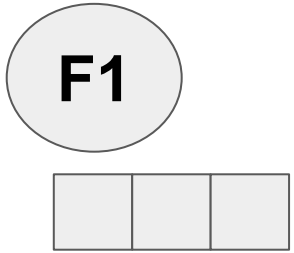


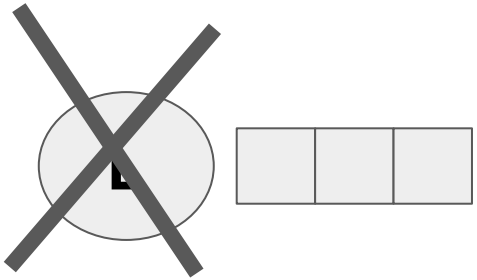
RAFT - Eleição

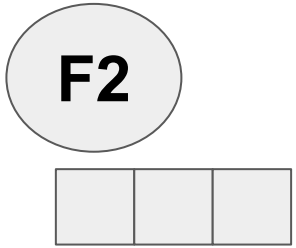




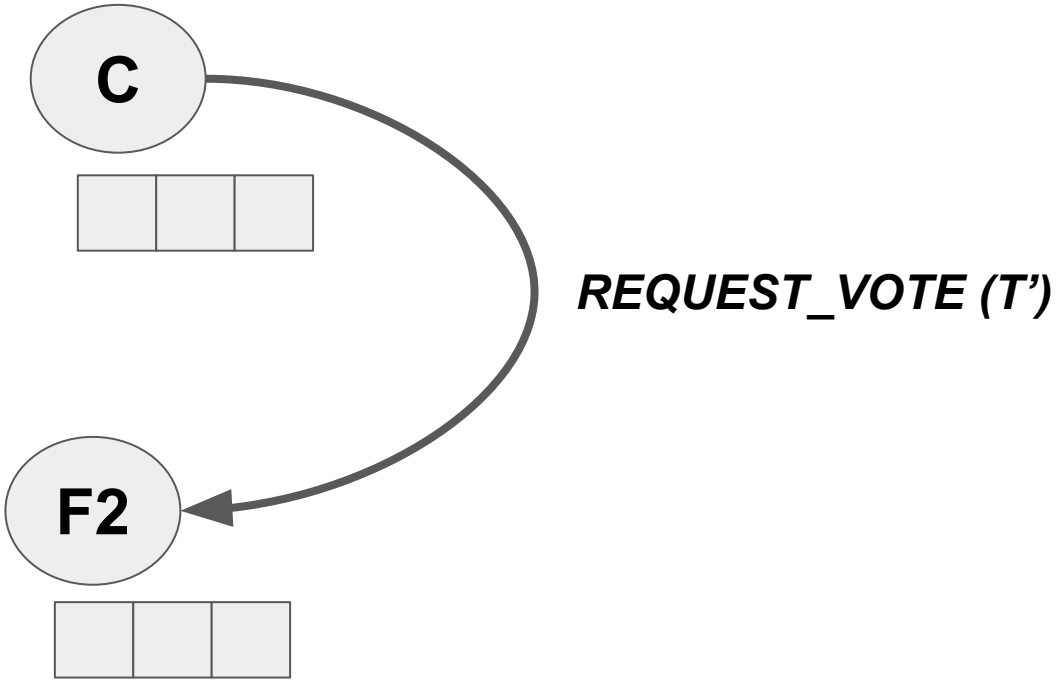
RAFT - Eleição

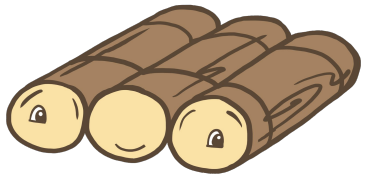




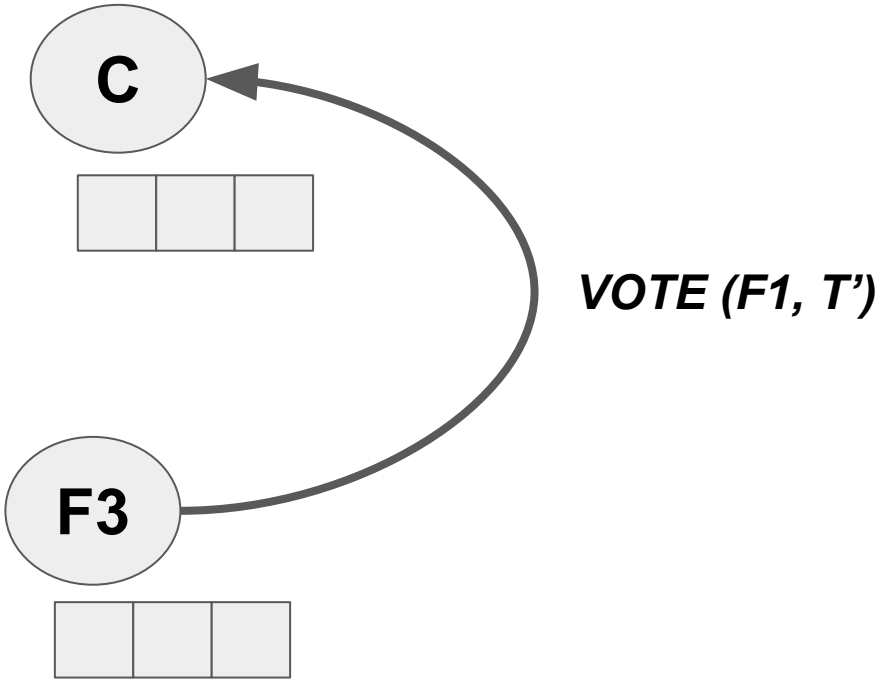


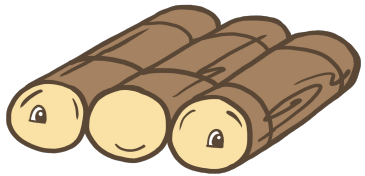
RAFT - Eleição



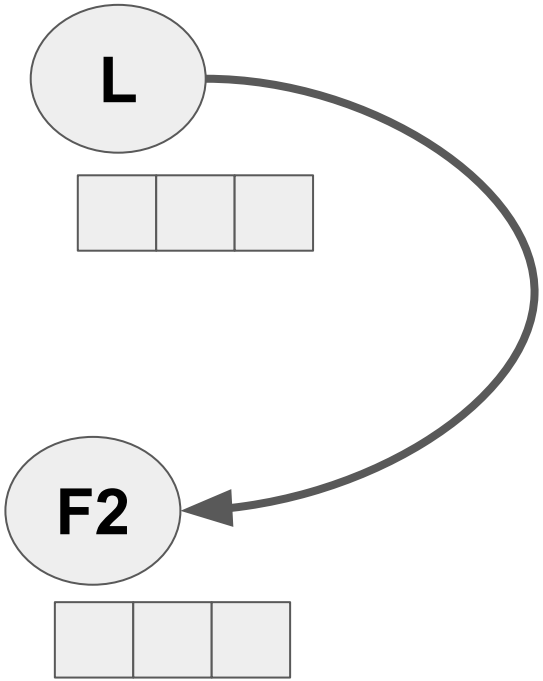


RAFT - Eleição

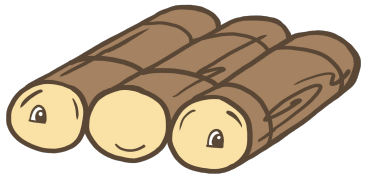




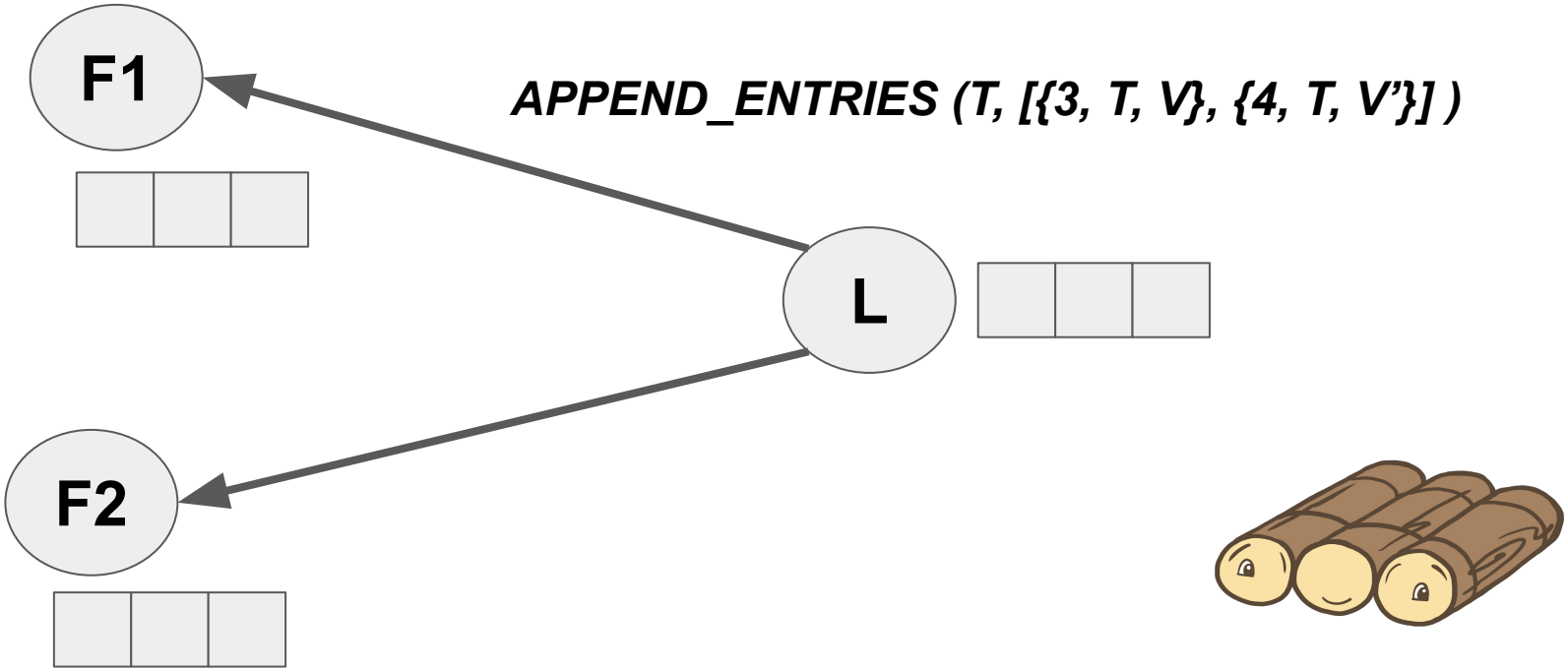
RAFT - Eleição



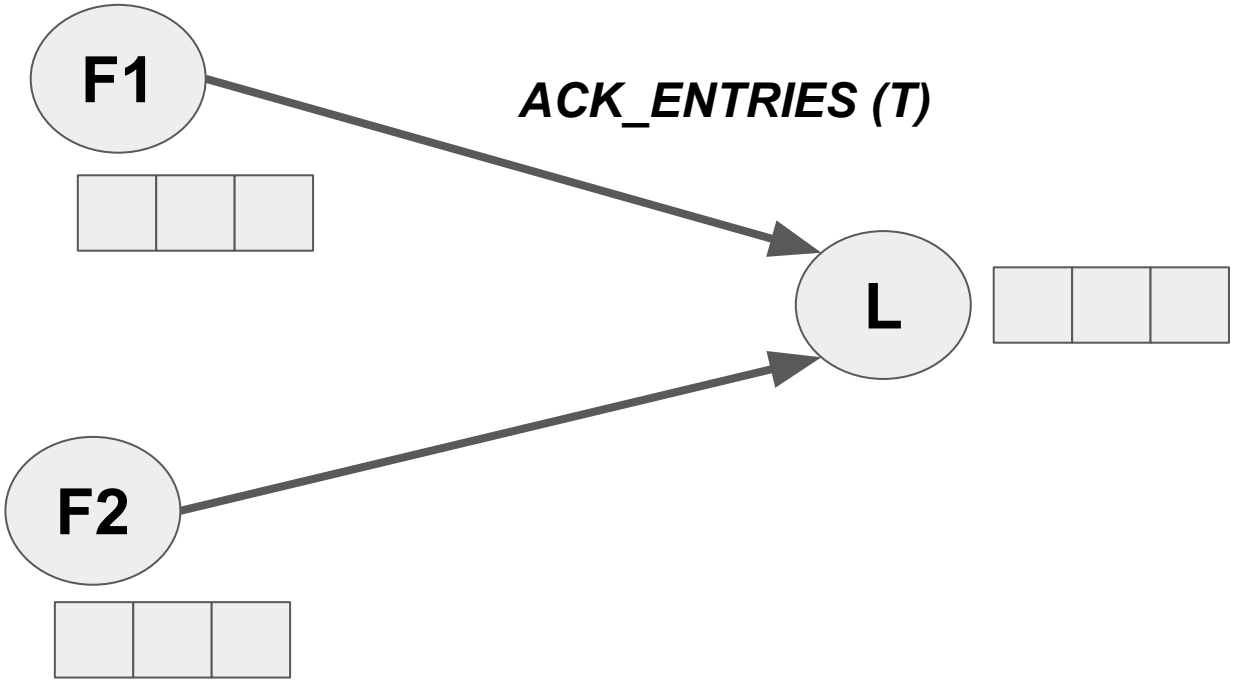
APPEND_ENTRIES (T', [0])

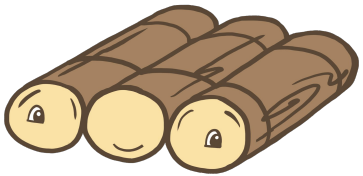


RAFT - Replicação de log

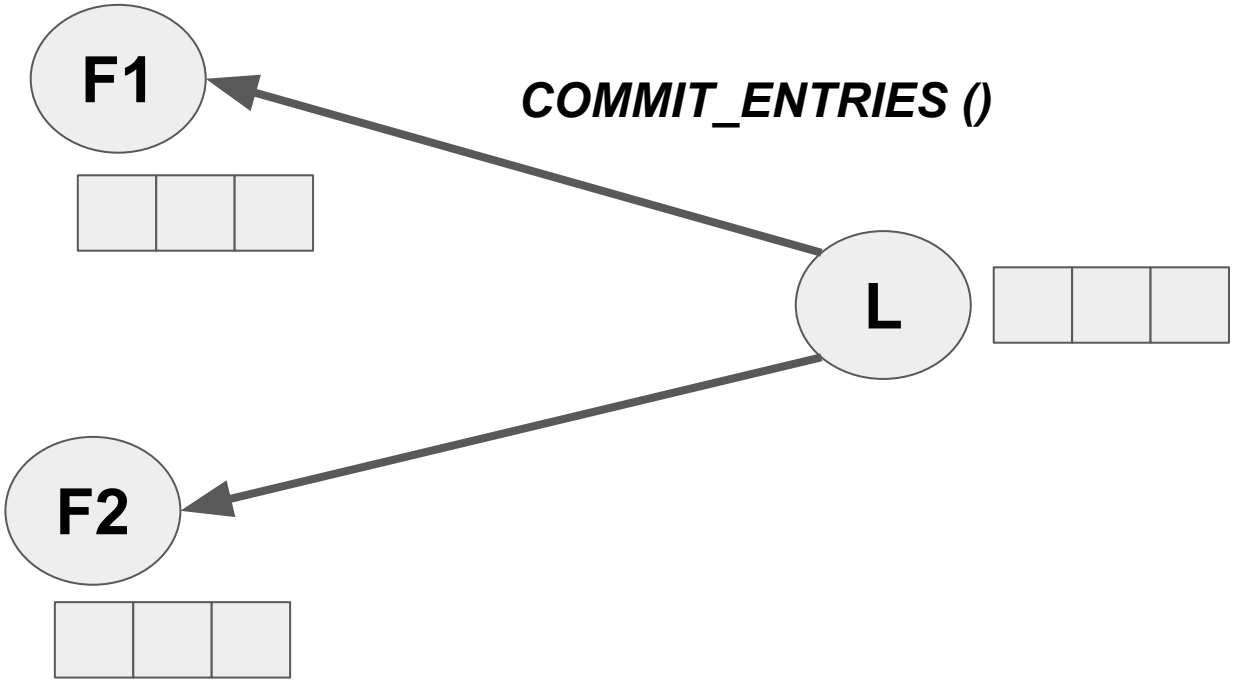


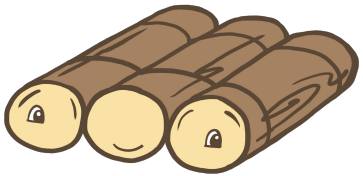
RAFT - Replicação de log



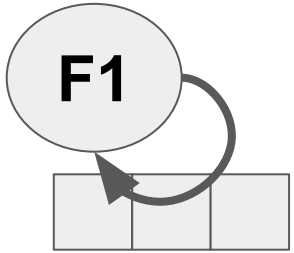


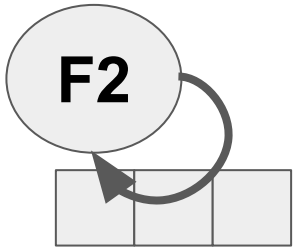
RAFT - Replicação de log



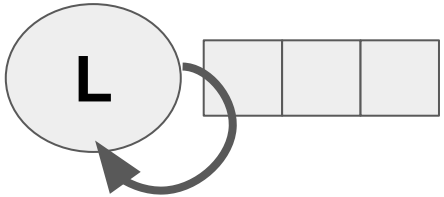


RAFT - Replicação de log



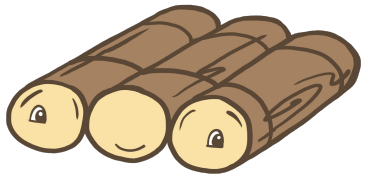


APPLY_ENTRIES (T)



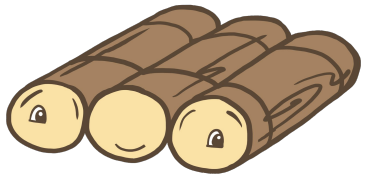
Raft - Replicação de logs

- Se os followers estão fora do ar, ou lentos, o líder continua a tentar replicar as entradas mesmo após retornar OK para o cliente;
- Cada entrada tem o termo para detectar inconsistências entre logs e garantir as propriedades de segurança;
- Cada entrada também possui um índice inteiro que indica sua posição no log;



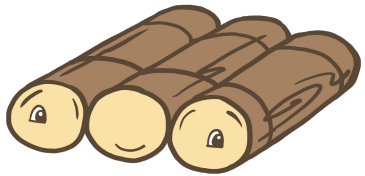
Raft

- O leader obtêm consistência dos logs forçando os followers a replicar seu log;
- O leader nunca deleta entradas de seu próprio log;
- O leader pode forçar os followers a apagarem entradas de seus logs, em caso de inconsistência;
- O leader envia no AppendEntries o índice e termo da entrada previamente efetivada;



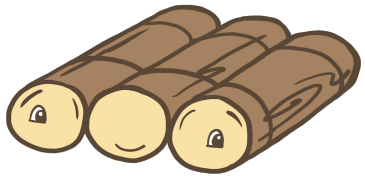
Raft

- Se o follower não tem essa entrada ele retorna erro;
- O leader decrementa a entrada e envia novo AppendEntries.
- Esse processo continua até que o leader e o follower cheguem a um acordo sobre a entrada mais recente que os dois concordam.

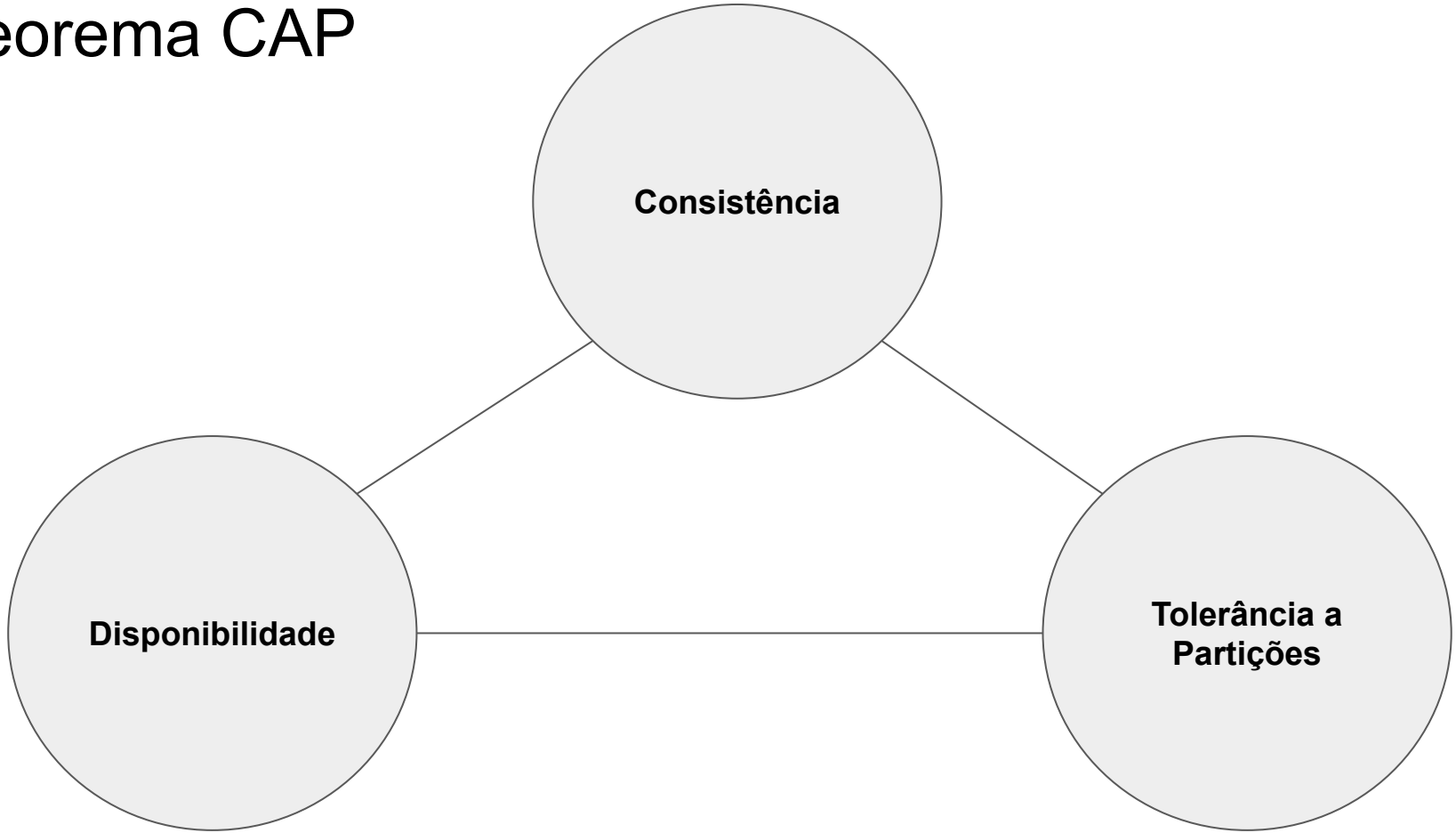


Raft - Implementações

- ***etcd***
  - Implementação do Raft mais utilizada em produção (Kubernetes, Cockroachdb, etc)
- Kudu
- Consul
- Inúmeras implementações open source e proprietárias;

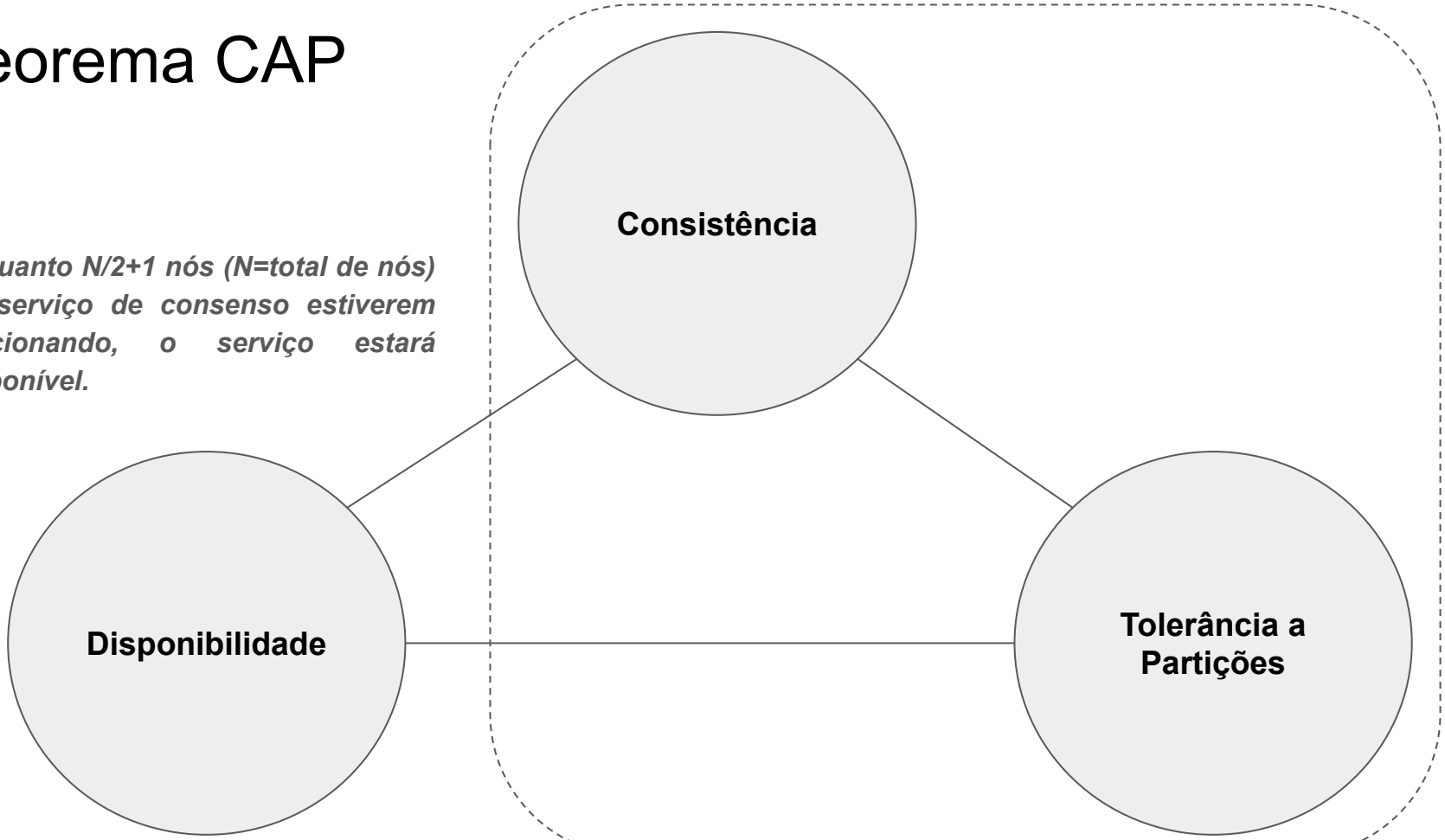


Teorema CAP



Teorema CAP

Enquanto N/2+1 nós (N=total de nós) do serviço de consenso estiverem funcionando, o serviço estará disponível.



Obrigado!