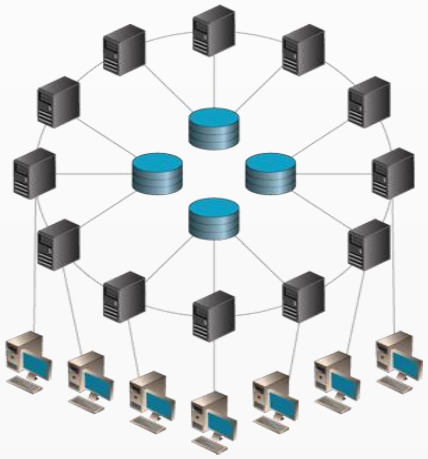
Consenso Distribuído

SMD0050 - SISTEMAS DISTRIBUÍDOS



Slides são baseados nos slides do Couloris e Tanenbaum

Consenso Distribuído

- Processos precisam ter uma visão idêntica (ou ação coordenada)
  - Mesmo valor a ser dado a uma saída (output) do sistema
  - Mesmo valor de uma variável compartilhada
  - Mesmo estado ou ação do sistema (abortar, desligar)
  - Definição de uma ordem total entre eventos (ou mensagens)
- Esses processos estão distribuídos e podem falhar
- Equivalência com outros problemas
  - Acordo Bizantino
  - Multicast Atômico

Modelo de Sistema

- Modelo de Sistema:
  - conjunto de processos Pi (i= 1,2,..,N)
  - a comunicação por mensagem é confiável
  - só os processos podem apresentar falhas
- Tipos:
  - falha de parada (crash)
  - falta arbitrária (bizantina)
- até f processos podem falhar simultaneamente do remetente

- É possível identificar o remetente de qq mensagem
  - Mensagens recebem uma assinatura digital para garantir a autenticidade do remetente
- Impede-se que processos faltosos possam falsificar identidade

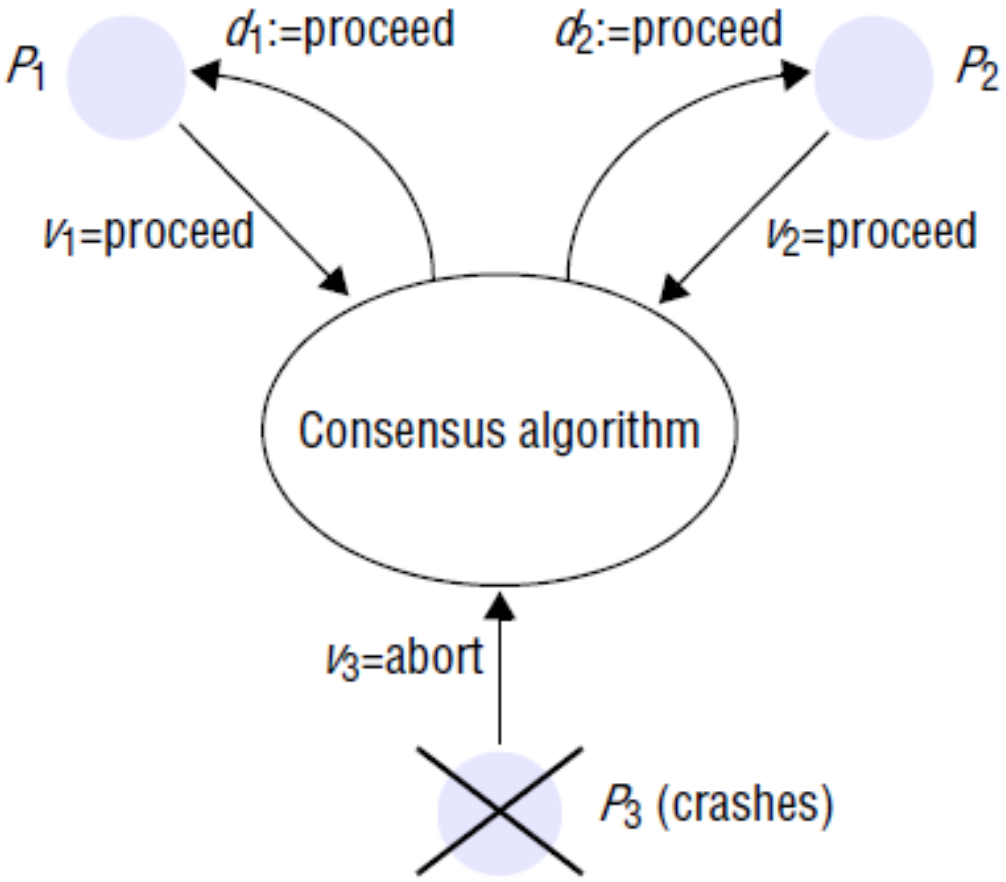
Problema do Consenso

- Existem N processos, dos quais **f** processos podem falhar
  - cada processo Pi propõe um único valor *vi* $\in D$
  - todos os processos interagem entre si para a troca de valores
  - em algum momento, os processos entram no estado "decided", em que definem p valor da variável de decisão *di* , que depois disso não é mais modificada
- O valor de **di** é sempre determinado em função dos valores vi fornecidos pelos N-f processos corretos.

Principais Requisitos

- Terminação
  - Em algum momento, cada processo correto atinge o estado "decided" e atribui um valor à variável de decisão di
- Acordo
  - todos os processos corretos atribuem o mesmo valor para a variável de decisão
- Integridade
  - se todos os processos corretos propuseram o mesmo valor vi =v, então qq processo correto no estado"decided" terá decidido di =v

Exemplo


P_1
d_1:=proceed
d_2:=proceed
P_2
v_1=proceed
v_2=proceed
Consensus algorithm
v_3=abort
P_3 (crashes)

Relembrando

Sistemas Síncronos

VS

Sistemas Assíncronos

O Modelo Síncrono

- Cada "passo de execução" em um processo demora entre [min, max] unidades de tempo
  - A transmissão de qualquer mensagem demora um limite máximo de tempo
  - O relógio local a cada processo tem uma defasagem limitada em relação ao relógio real
- tarefas (processos) têm tempos de processamento fixos e bem conhecidos
  - controle de processos, processador paralelo

O Modelo Síncrono

A execução ocorre em rodadas de "processamento & comunicação" perfetamente sincronizadas

➔ O não-recebimento de uma mensagem em determinado período também traz informação!

Em sistemas reais, é difícil determinar estes limites de tempo. Mas este modelo é adequado quando a rede é dedicada e o atraso é determinístico

Tarefa

Proponha uma solução de consenso distribuído:

1. em um Modelo Síncrono
2. Suponha que não há falhas!

Exemplo de Consenso para Modelo Síncrono

```
Pseudo-código para nó i:
Init => {
        escolhe valor v;
        broadcast (v:i) para todos os vizinhos
        localset ← {(v:i)}
}
Enquanto (!terminou){ // Nova rodada
        recebe conjunto c de mensagens dos demais;
        diff  ← c - localset; // diferença entre conjuntos
        se diff ≠ ∅ {
                localset ← localset ∪ diff;
        } senão se (já recebeu de todos nós) {
                terminou ← true;
        }
}
d ← acha_mais_votado(localset);
imprime o valor de consenso d;
```

Propriedade:

Assumindo-se comunicação confiável e rede com N processos, após N-1 rodadas, um processo deverá ter recebido o valor de todos os nós ativos (não falhos).

Cada nó escolhe um valor $v \in [0, max]$ e difunde este valor para todos os demais nós na rede. Quando um nó tiver recebido o valor de todos os demais nós, escolhe o valor mais votado (ou no caso de empate, um valor default) como o valor de consenso d.

Análise

- Terminação
  - Como não há falhas todos os nós receberam as mensagens dos nós participantes do sistema
  - Processos esperam até receber os valores de todos os outros
- Acordo
  - Processos usaram o valor mais votado do conjunto (que é o mesmo para todos)
- Integridade
  - As mensagens com os valores são entregues a todos os processos

Desafio dos dois generais

Consenso em Sistemas Assíncronos

Se processos podem ter falhas tipo crash (omissão), então a terminação não estará garantida, a menos que se detecte a falha.

Se o sistema é puramente assíncrono, pode ser impossível distinguir entre um crash de um processo e uma mensagem que demora um tempo indeterminado.

Fischer, Lynch & Paterson(*) apresentaram um resultado teórico fundamental:

- é impossível garantir consenso em um sistema puramente assíncrono com falhas tipo crash

FLP Impossibiity Problem

Se processos podem apresentar falhas arbitrárias (bizantinas), então processos falhos podem comunicar valores aleatórios aos demais processos, evitando que os corretos tenham a mesma base de dados {v1,v2,..,vN}, para a tomada de uma decisão uniforme.

Fischer, Lynch, Paterson: Impossibility of Distributed Consensus with one faulty process, Journal of the ACM, v.32 (2), 1985.

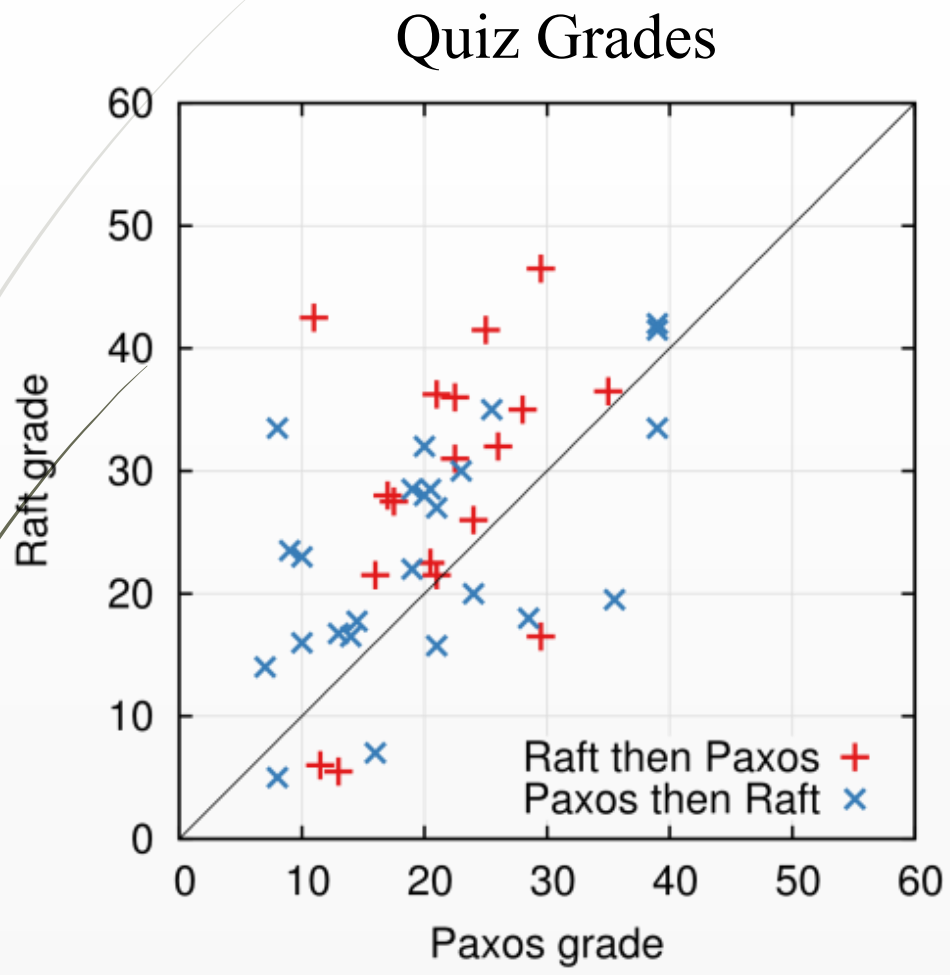
Uma solução por consenso parcial

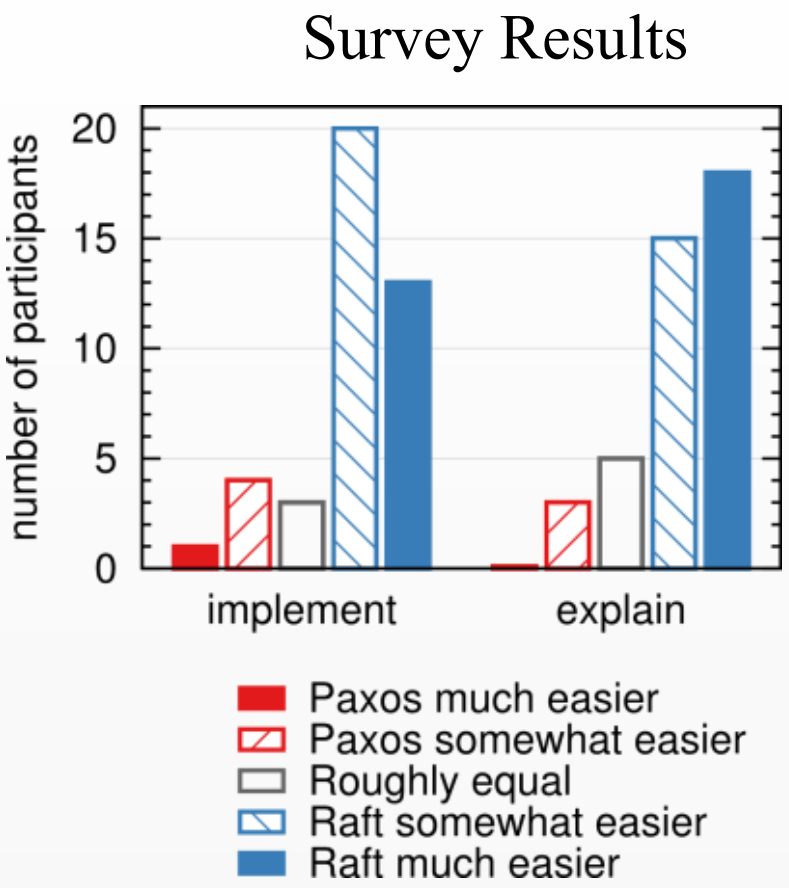
- Algoritmos propostos por Lamport
  - PAXOS (1989)
    - Difícil entendimento e implementação
  - Multi-PAXOS
- RAFT
  - Maior simplicidade, uma forma de distribuir uma máquina de estado em um sistemas distribuído
  - Diego Ongaro PHD (2015)
  - Understandablity first!
- Serviços de consenso
  - Raft é usado em HydraBase (Facebook), Rafter/Basho (NOSQL keyvalue store), ZooKeeper, RAMCloud

RAFT Overview

- É tarefa do algoritmo de consenso manter os logs replicados consistentes.
  - Líder forte (as entradas do log sempre fluem do líder para os membros)
  - mais eficiente do que soluções leader-less e mais simples: operação noranal ou leader election
- Para eleição de líder, usa temporizadores randômicos, para evitar “race condition” na eleição
- Simplicidade
  - decompõe o problema em eleição de líder, replicação do log e segurança

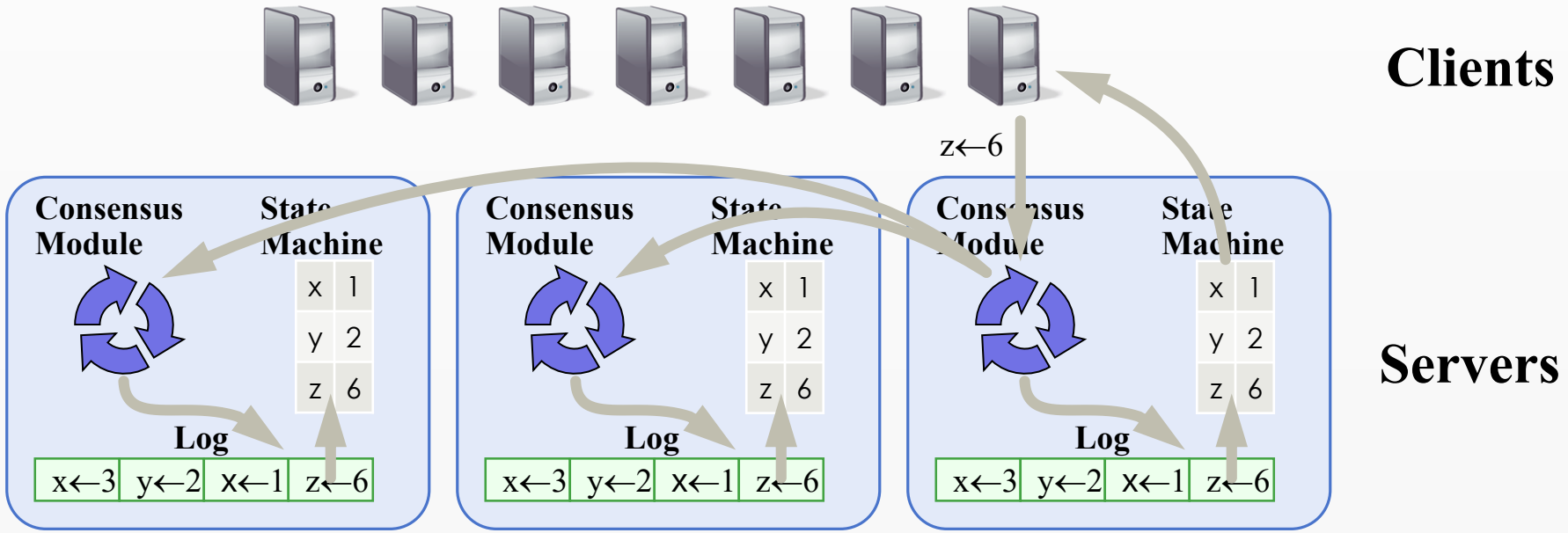
Raft User Study





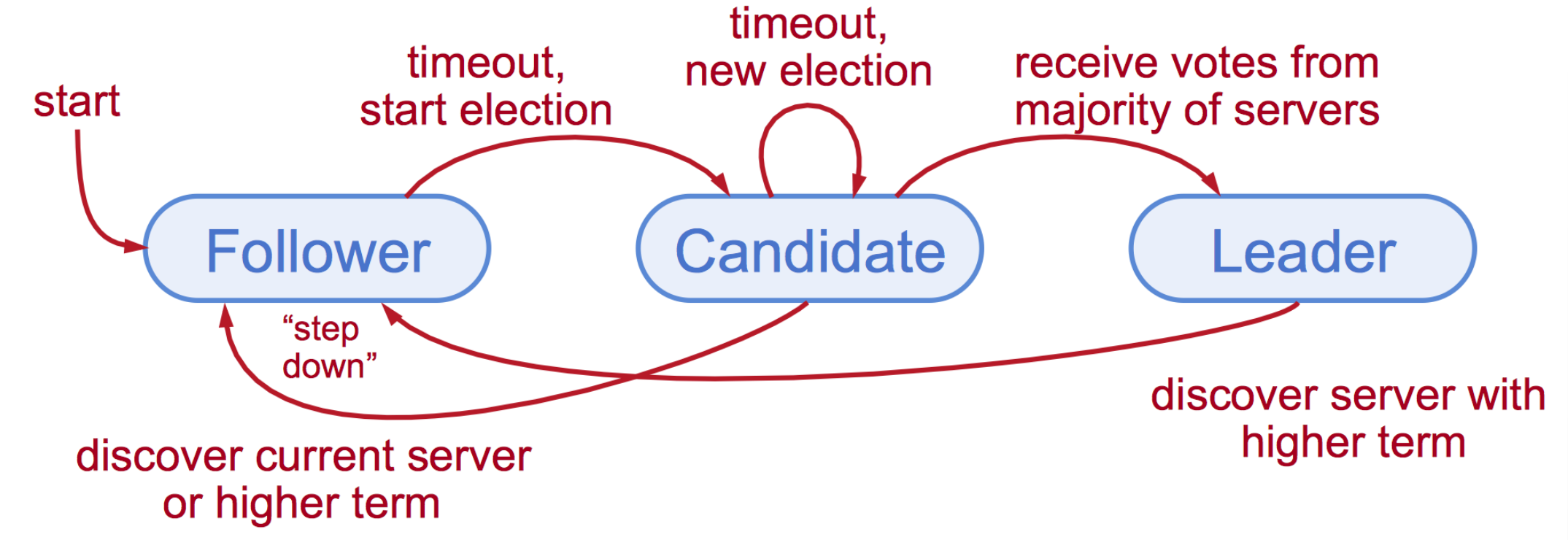
RAFT

- Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. 2014,, Stanford University
  - https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro



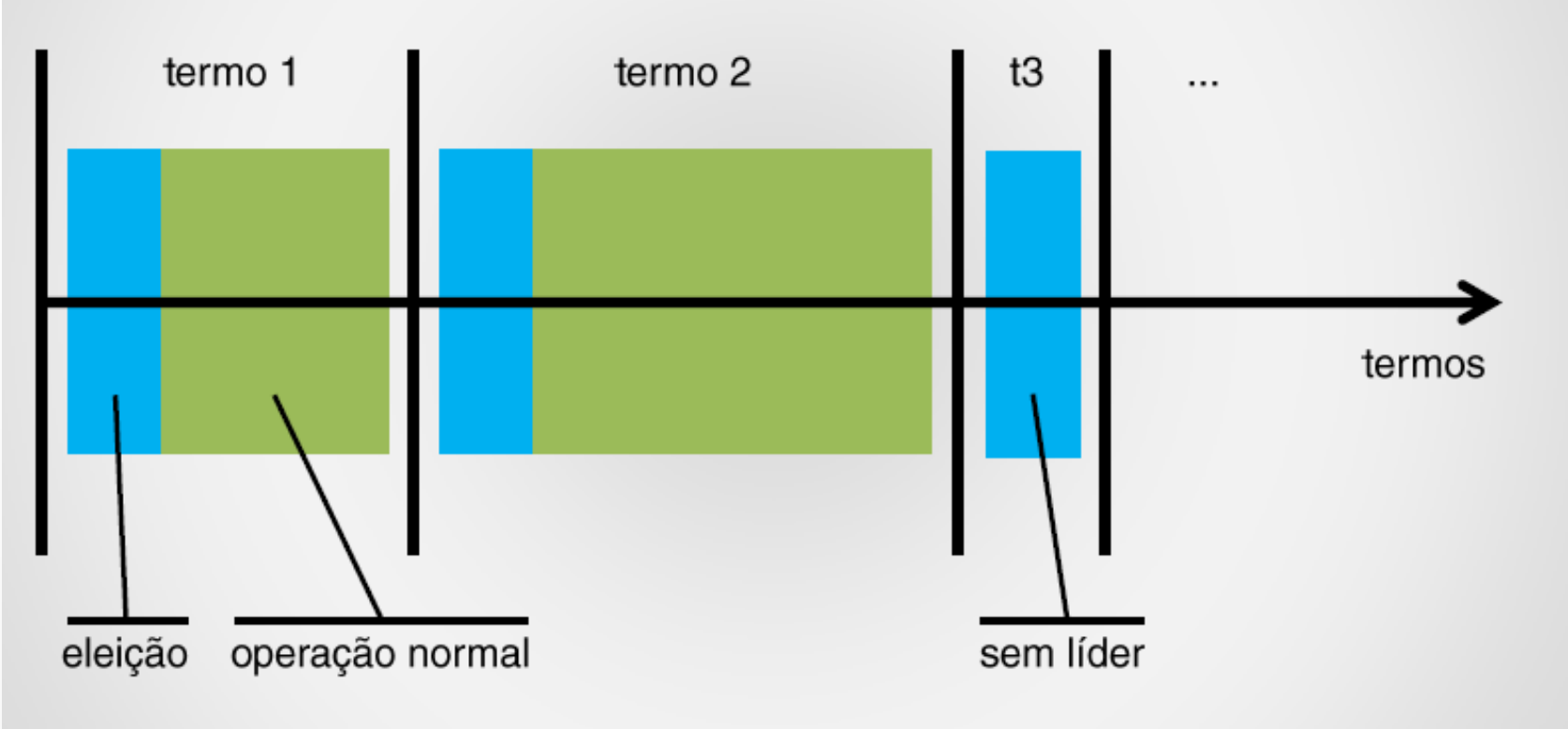
RAFT Overview

1. Leader election
   - Select one of the servers to act as cluster leader
   - Detect crashes, choose new leader
2. Log replication (normal operation)
   - Leader takes commands from clients, appends them to its log
   - Leader replicates its log to other servers (overwriting inconsistencies)
3. Safety
   - Only a server with an up-to-date log can become leader

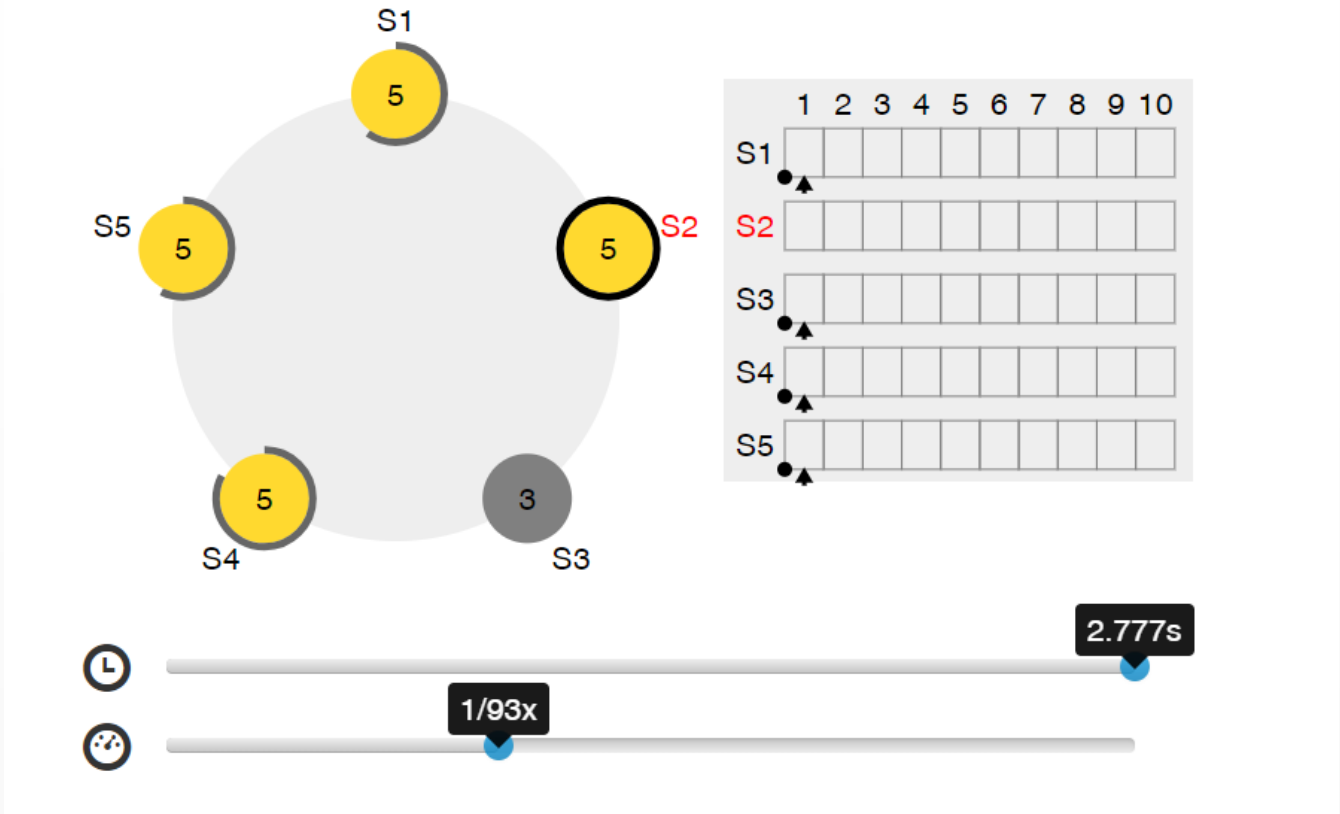


- Candidatos tentam se eleger como líder, e acabam competindo entre si
- Sempre existe no máximo um líder, e todos os demais nós são followers ou candidatos.
- O líder responde a todas as requisições de clientes e os followers são passivos (só respondem requisições do líder}.

Tempo dividido em Termos



Explicação Visual



https://raft.github.io/

Eleições em sistemas de grande escala

- Há situações em que é necessário trabalhar com redes maiores e é necessário eleger maior quantidade de pares, ex.: Superpares em P2P
- Requisitos a serem cumpridos por superpar:
  1. Nós normais devem ter baixa latência de acesso com superpares;
  2. Superpares devem estar uniformemente distribuídos pela rede de sobreposição;
  3. Deve haver uma porção predefinida de superpares em relação ao número total de nós na rede de sobreposição;
  4. Cada superpar não deve precisar atender mais do que um número fixo de nós normais;

Eleições em sistemas de grande escala
Superpares com *k* de *m* bits como id

- Uma solução é dada quando se usa *m* bits de identificador, separar os *k* bits da extrema esquerda para identificar superpares;
  - Ex.: $\log_2$(N) Superpares, *m*=8,*k*=3.
  - *p* AND 11100000 = Superpar.
- Problema: não garante posicionamento geométrico para organizar os superpares *uniformemente* pela rede

Eleições em sistemas de grande escala
Eleição de pares por fichas repulsoras

- N fichas distribuídas aleatoriamente entre os nós;
- Nenhum nó pode ter mais de uma ficha;
- Fichas possuem uma força de repulsão;
- Um nó que mantiver a ficha por determinado tempo é eleito superpar.

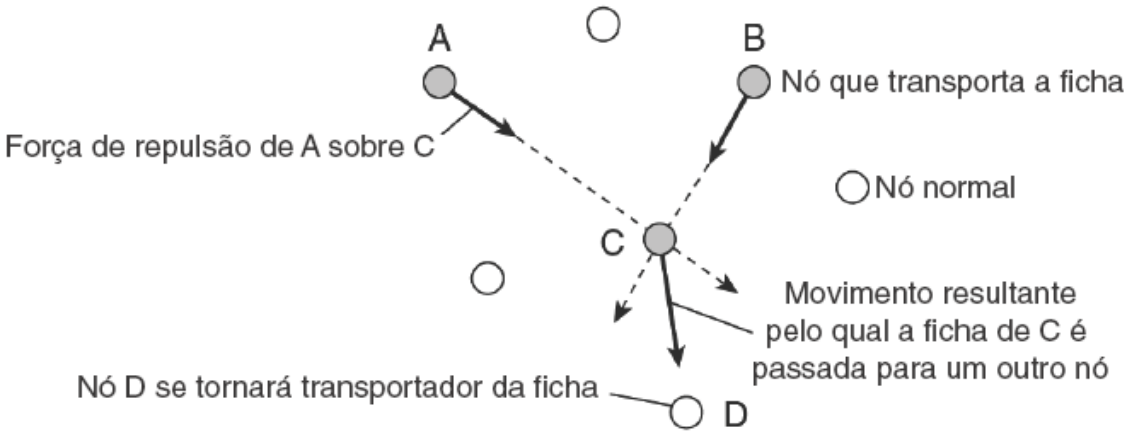


Figura 6.22 Movimentação de fichas em um espaço bidimensional que utiliza forças de repulsão.