



# Algoritmos de Eleição

Slides são baseados nos slides do Couloris e Tanenbaum

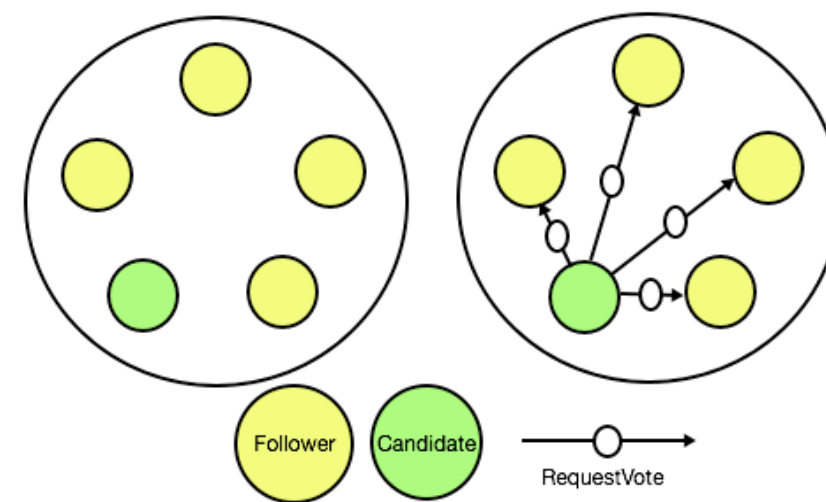
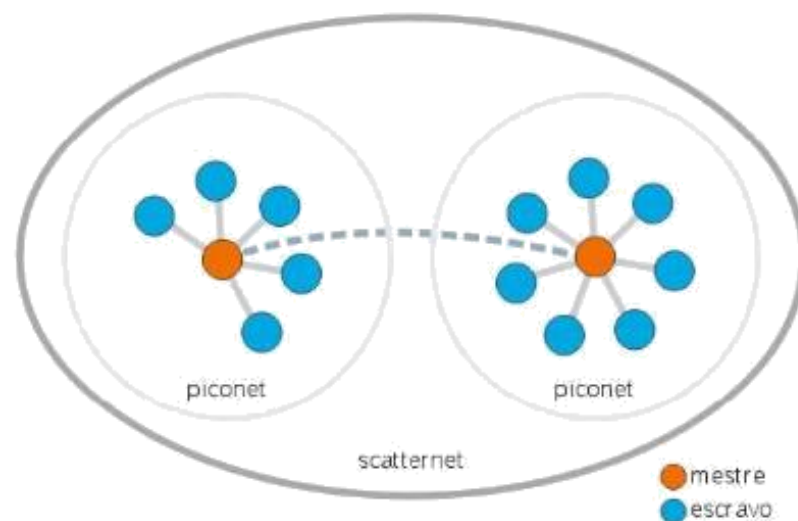
**Professores:**

**Fernando Antonio Mota Trinta**

**Windson Viana de Carvalho**

## Algoritmos de Eleição

- Usados quando há necessidade de um ou mais nós devem agir como coordenador
- Coordenar a replicação, difusão, agregação
  - Formação de Scartternet em Bluetooth
- Iniciar/coordenar um processo de resolução de consenso
  - Paxos, RAFT



## Exemplos de Algoritmos de Eleição

- De propósito geral
  - Algoritmo do valentão
  - Algoritmo de anel
- Soluções para ambientes específicos
  - Algoritmos para Ad Hoc (Bluetooth Topology Construction)
  - Algoritmos para sistemas de grande escala (Blockchain)



## Requisitos Importantes

- **Safety** - Todos os participantes precisam chegar a um consenso ou serem informados de quem foi o escolhido
- **Liveness** – Em algum momento, deve-se definir o coordenador
- **Stability** – Devem ser evitadas reeleições desnecessárias
- Nem sempre é possível saber o número total de processos
  - inundação, topologia em anel



## Requisitos Importantes

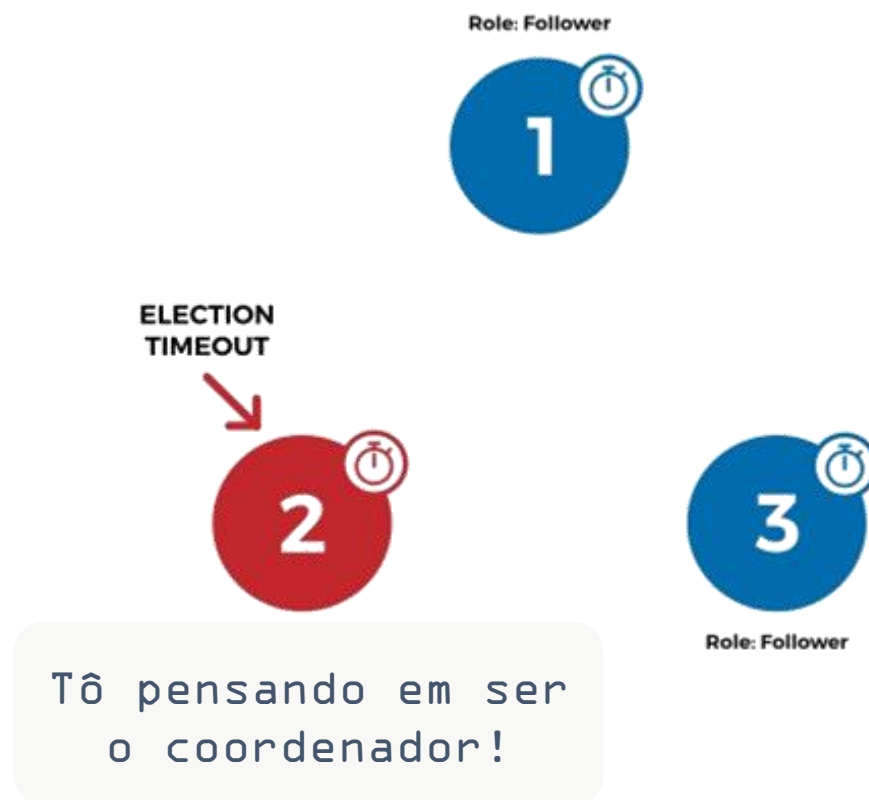
- A eleição é iniciada como reação a uma **detecção de falha** do antigo coordenador
- Pode haver falha durante a escolha do novo coordenador
- Múltiplas eleições podem ocorrer em paralelo mas devem chegar a um mesmo resultado



**requisitos**

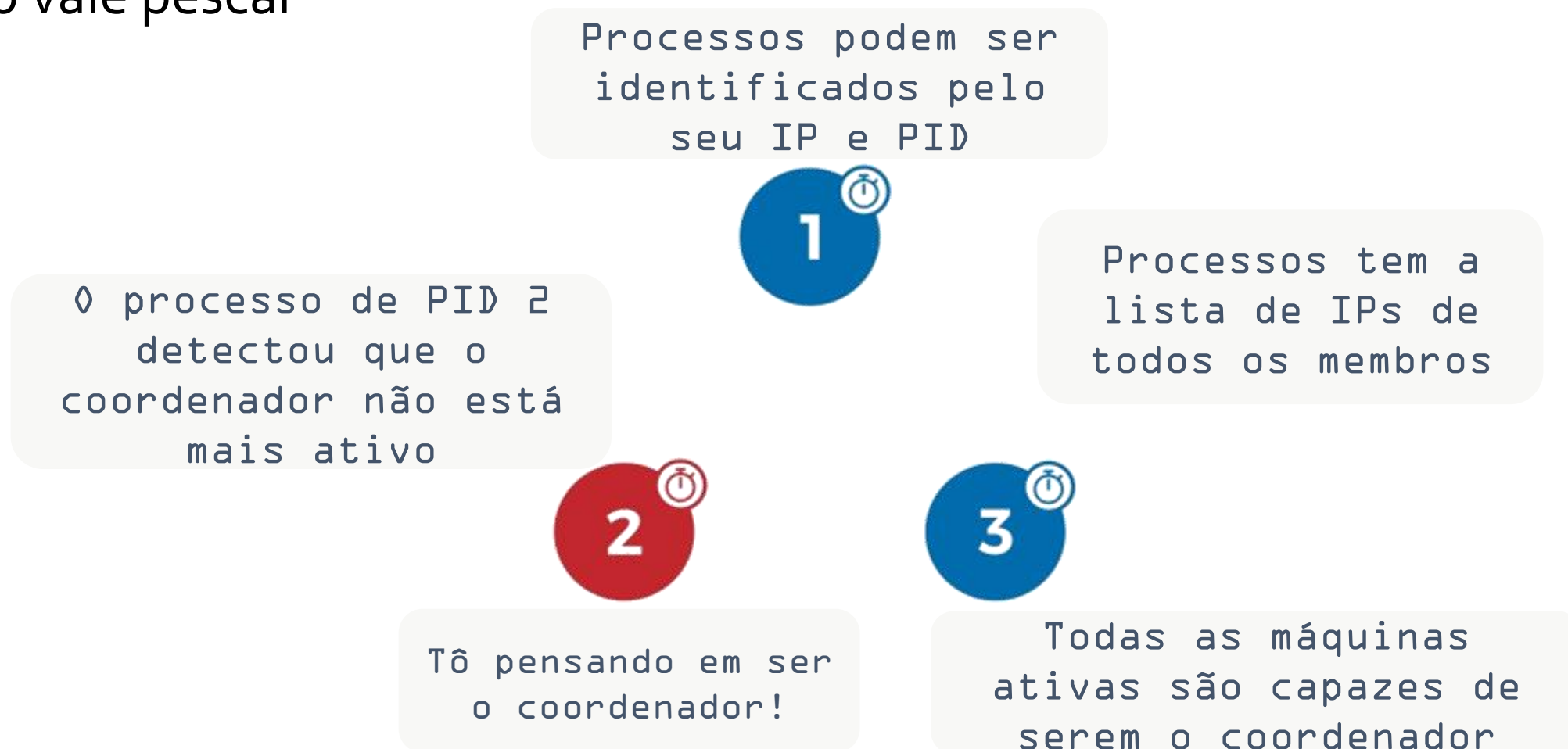
Tarefa 1- Vamos tentar criar o nosso!

- Não vale pescar



Tarefa 1- Vamos tentar criar o nosso!

- Não vale pescar



## Algoritmo do Valentão (Bully)

- Inventado por Garcia-Molina (1982)
- Todos nós possuem um identificador
- Sistema síncrono com falhas tipo fail-stop\*, baseado na difusão de mensagens

É eleito o nó com maior identificador que está ativo

\*fail-stop – processo “cai” e isso é detectável por parceiros.



## Algoritmo do Valentão

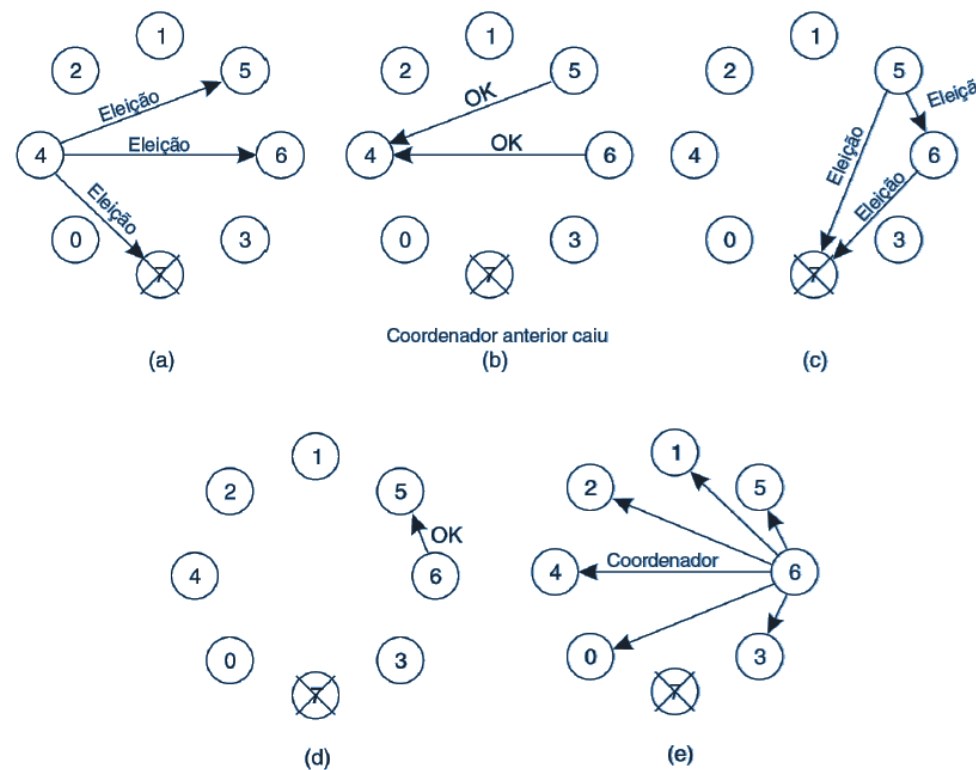
- Sempre que um nó qualquer P nota que o coordenador não responde, P inicia uma eleição:
  1. P envia uma mensagem ELEIÇÃO a todos os processos de números mais altos;
  2. Se nenhum responder (sistema síncrono), P vence a eleição e se torna o coordenador;
  3. Se um dos processos de número mais alto responder, ele toma o poder e o trabalho de P está concluído.

## Algoritmo do Valentão

- Sistema Síncrono - Toda mensagem é entregue em  $T_m$  unidades de tempo após o seu envio
- Todos os processos não falhos respondem a todas as mensagens recebidas em  $T_p$  unidades de tempo
- Definição de um detector de falhas confiável:

Se um processo não responde em  $2T_m + T_p$  unidades de tempo, ele falhou

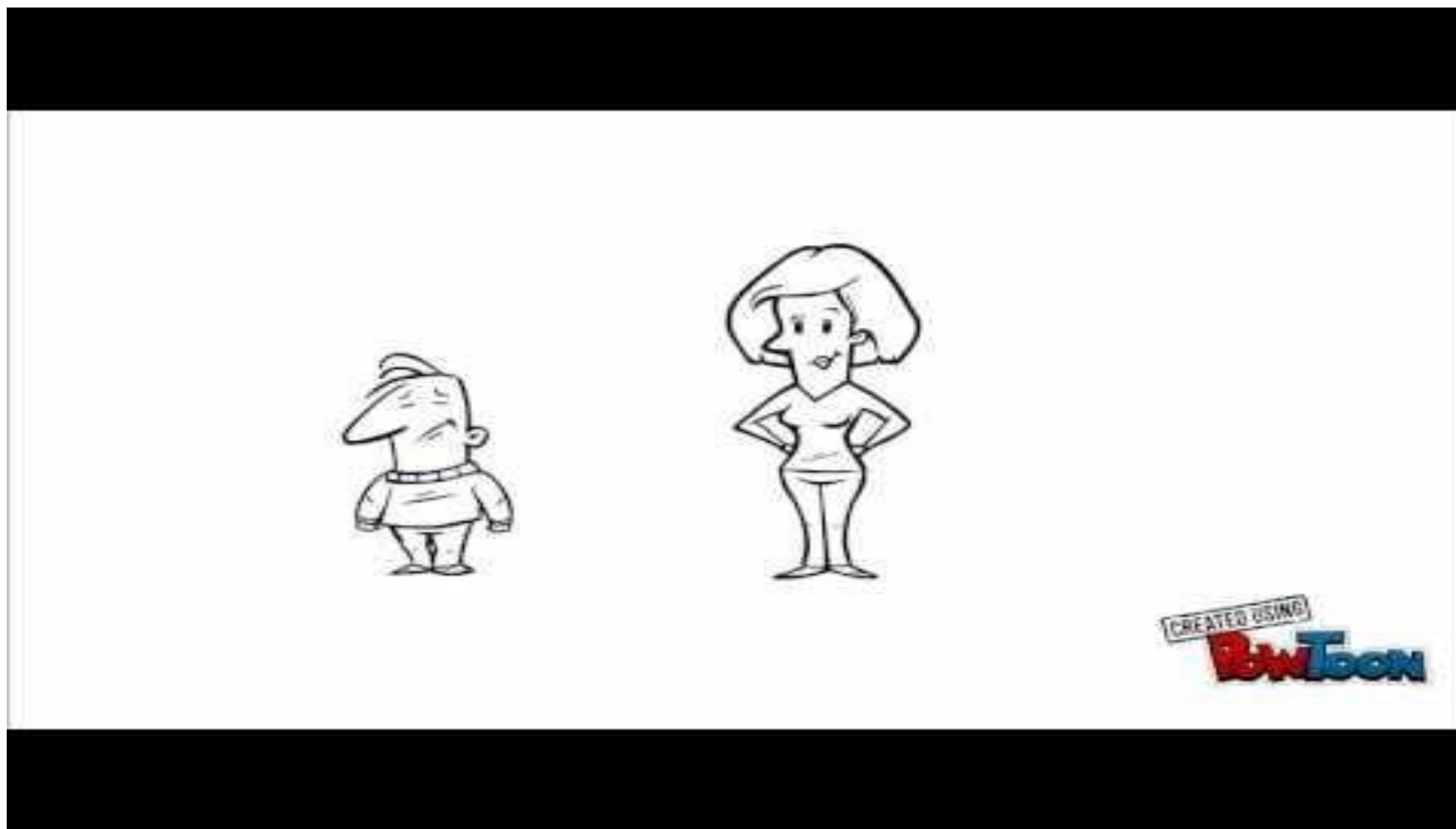
# Algoritmo do Valentão



**Figura 6.19** Algoritmo de eleição do valentão. (a) O processo 4 convoca uma eleição. (b) Os processos 5 e 6 respondem e mandam 4 parar. (c) Agora, cada um, 5 e 6, convoca uma eleição. (d) O processo 6 manda 5 parar. (e) O processo 6 vence e informa a todos.

# Vantagens e Desvantagens?

## Vídeo de Revisão



[https://www.youtube.com/watch?v=K44x\\_VQmUs8](https://www.youtube.com/watch?v=K44x_VQmUs8)

## Algoritmo do Valentão

- Proposto por Le Lann, Chang e Roberts
- Baseado na utilização de anel (físico ou lógico)

**Não usa ficha!**

- Quando qualquer processo nota que o coordenador não está funcionando, monta uma mensagem ELEIÇÃO com seu próprio número (ex: PID+IP) e o envia a seu sucessor ou ao próximo que esteja em funcionamento

## Algoritmo do Anel

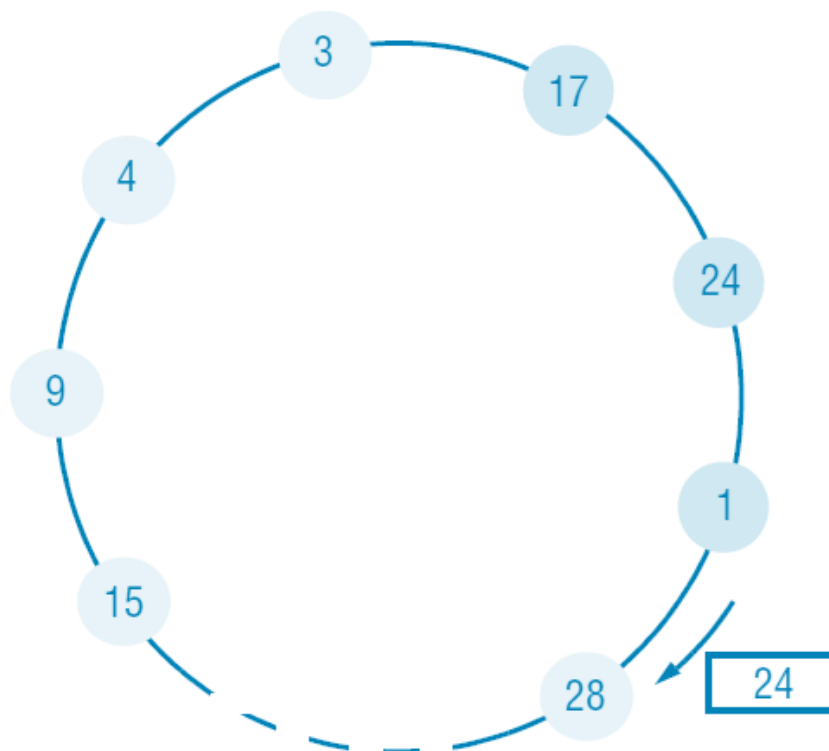
- Se o nó que recebe a mensagem de eleição tem um identificador maior que o informado na mensagem que recebeu, passa uma mensagem de eleição para seu vizinho da direita com seu próprio identificador.
- Caso contrário aceita que o nó que tem o identificador contido na mensagem será o líder e repassa ao seu vizinho da direita.

## Algoritmo do Anel

- A eleição termina quando
- Se o nó recebe uma mensagem com o identificador idêntico ao seu, ele se declara **LÍDER**
- envia essa mensagem pelo Anel
- Este evento só ocorre quando a mensagem contendo o maior identificador circulou por todo o anel tornando todos os seus membros cientes do resultado.



## Algoritmo do Anel



*Note:* The election was started by process 17. The highest process identifier encountered so far is 24. Participant processes are shown in a darker tint.

## Tarefa 2

- Existem  $n$  processos  $P$  que a cada 30s imprimem o nome do coordenador
- Vamos implementar um algoritmo de eleição que eleja qual será o coordenador
- Vamos usar RMI na implementação
- Crie uma Interface Remote chamada  $P$  com o método `startElection()` e `setLeader()`
- Crie uma variável PID em cada processo
- Use `ManagementFactory.getRuntimeMXBean().getName()`

In Java 9 the new [process API](#) can be used:

```
long pid = ProcessHandle.current().pid();
```

## 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  $P_i$  ( $i = 1, 2, \dots, N$ )
    - 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 qualquer 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  $P_i$  propõe um único valor  $v_i \in D$
  - todos os processos interagem entre si para a troca de valores
  - em algum momento, os processos entram no estado “decided”, em que definem o valor da variável de decisão  $d_i$ , que depois disso não é mais modificada
- O valor de  $d_i$  é sempre determinado em função dos valores  $v_i$  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  $d_i$

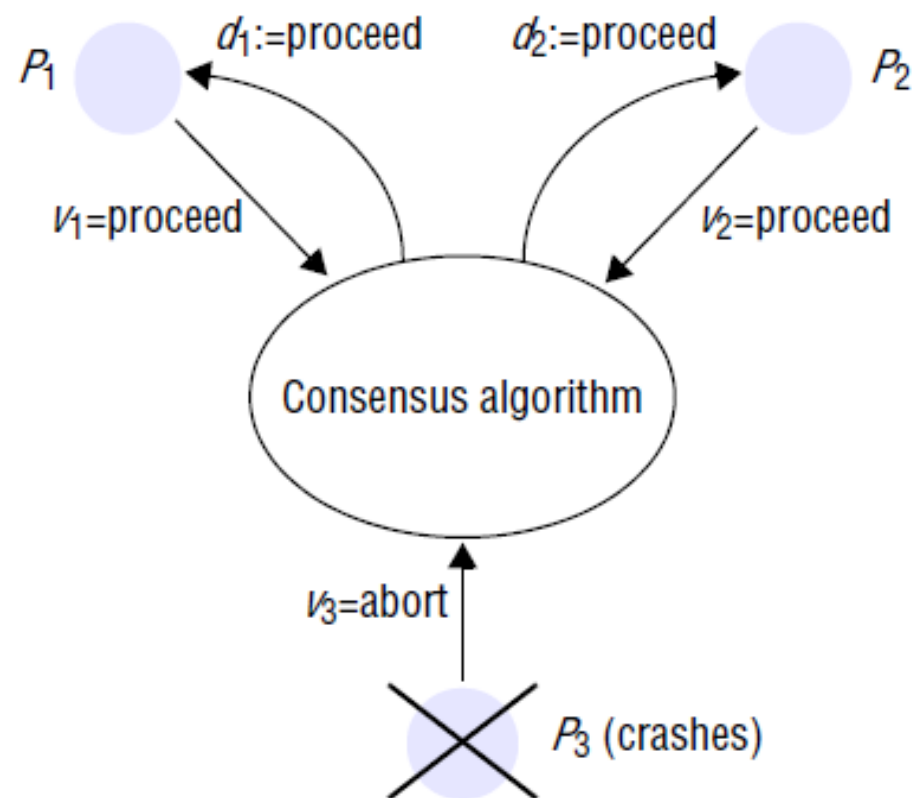
- **Acordo**

- todos os processos corretos atribuem o mesmo valor para a variável de decisão

- **Integridade**

- se todos os processos corretos propuseram o mesmo valor  $v_i = v$ , então qq processo correto no estado “decided” terá decidido  $d_i = v$

## Exemplo



# 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



## □ Modelo Síncrono

- A execução ocorre em rodadas de “processamento & comunicação” perfeitamente 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...

## O Modelo Síncrono

- 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.

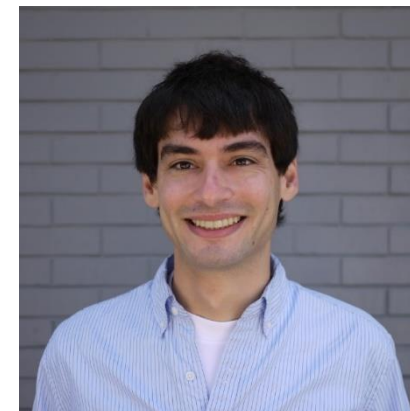
## FLP Impossibility Problem

- 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  $\{v_1, v_2, \dots, v_N\}$ , 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)
  - Understandability first!
- Serviços de consenso
  - Raft é usado em HydraBase (Facebook), Rafter/Basho (NOSQL keyvalue store), ZooKeeper, RAMCloud

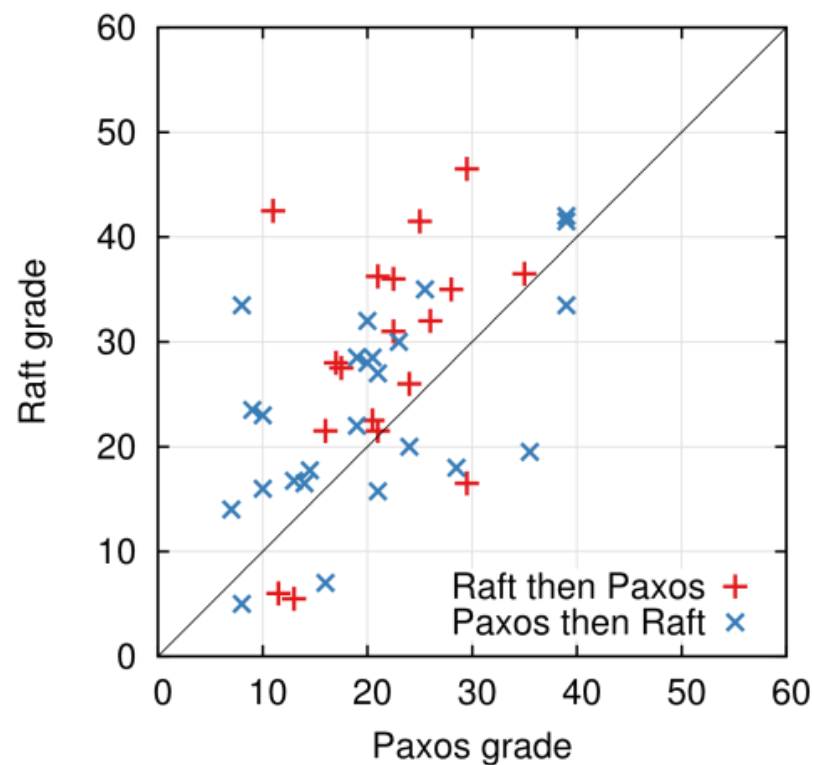


## RAFT Overview

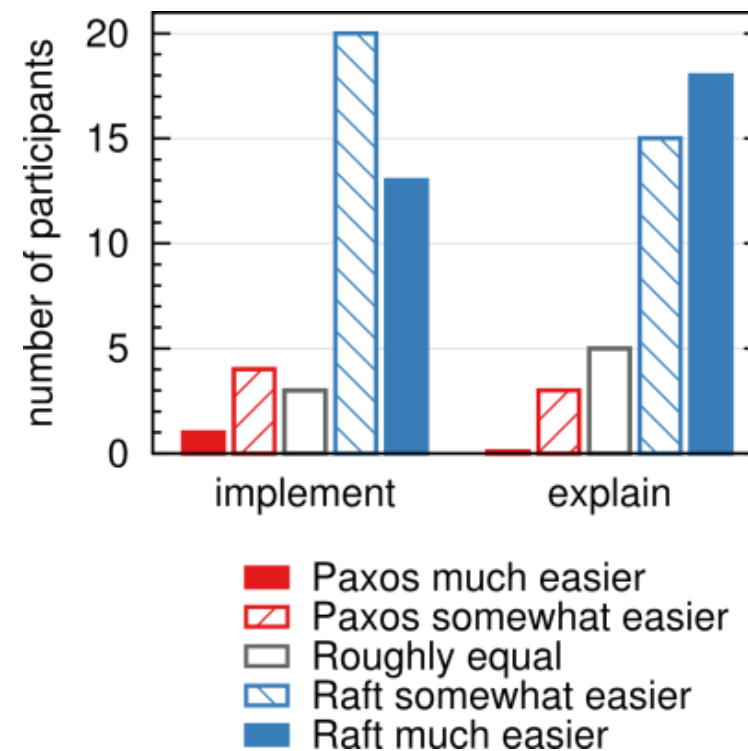
- É tarefa do algoritmo de consenso manter os logs replicados consistentes.
  - Líder forte (as entradas do log sempre fluem do líder para os membros)
  - mais eficiente do que soluções leader-less e mais simples: operação normal ou leader election
- Para eleição de líder, usa temporizadores randômicos,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

## Raft User Study

## Quiz Grades

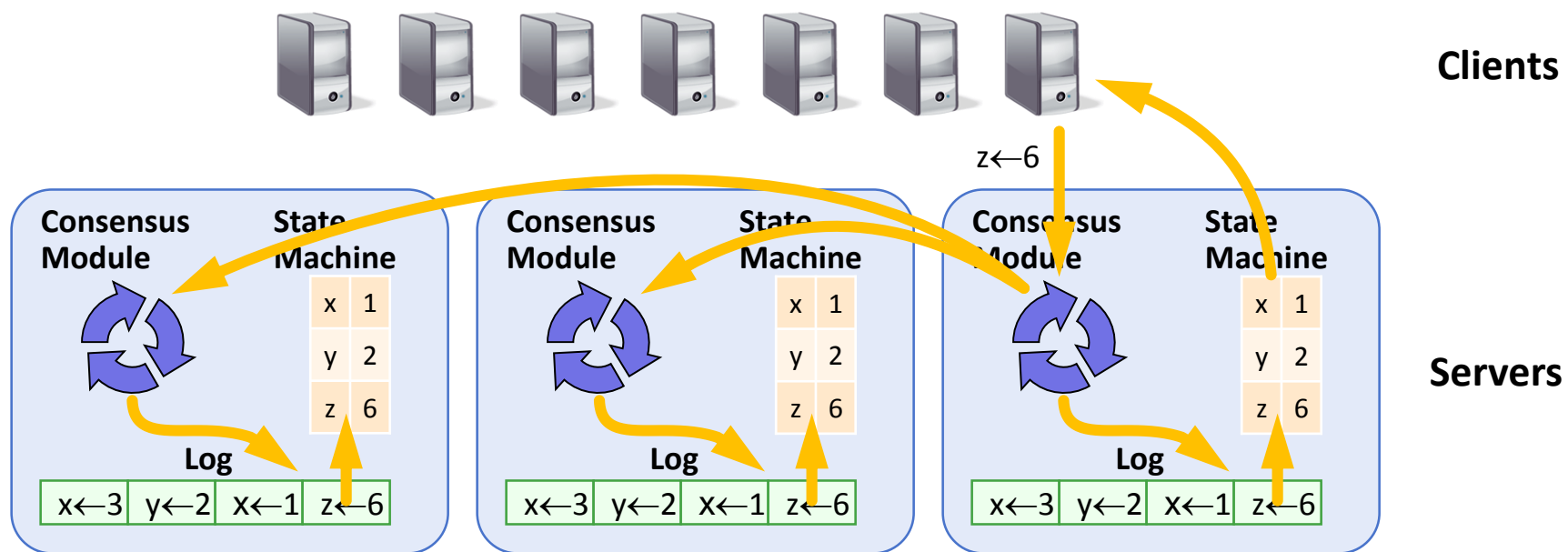


## Survey Results



## 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

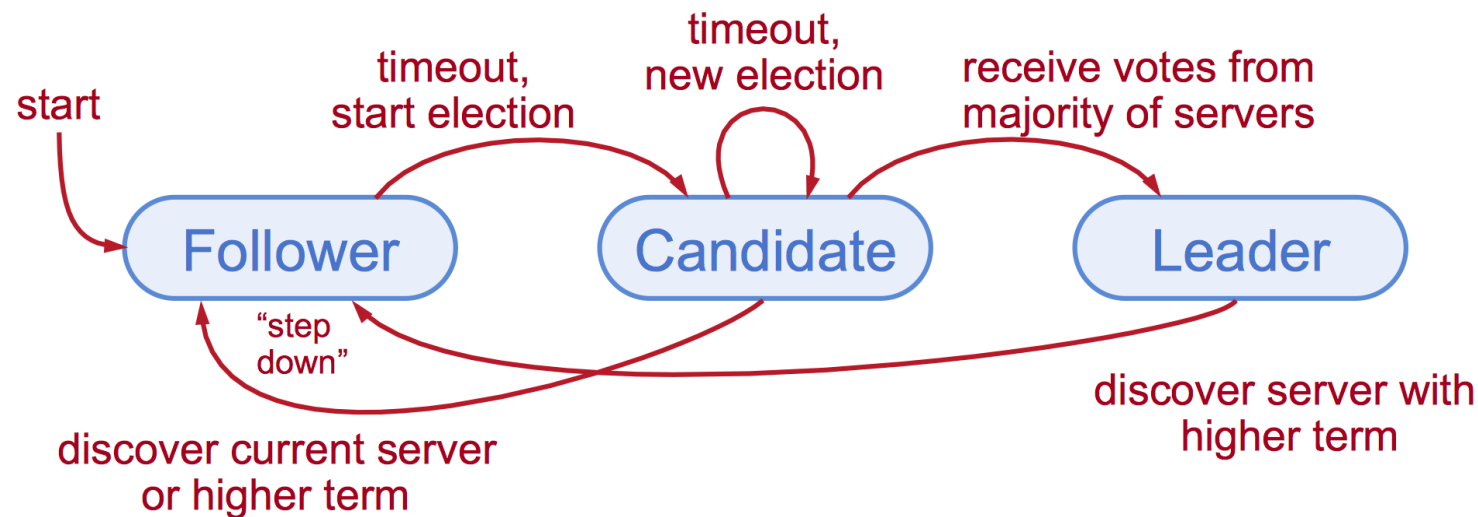
### 2. Log replication (normal operation)

- Leader takes commands from clients, appends them to its log
- Leader replicates its log to other servers (overwriting inconsistencies)

### 3. Safety

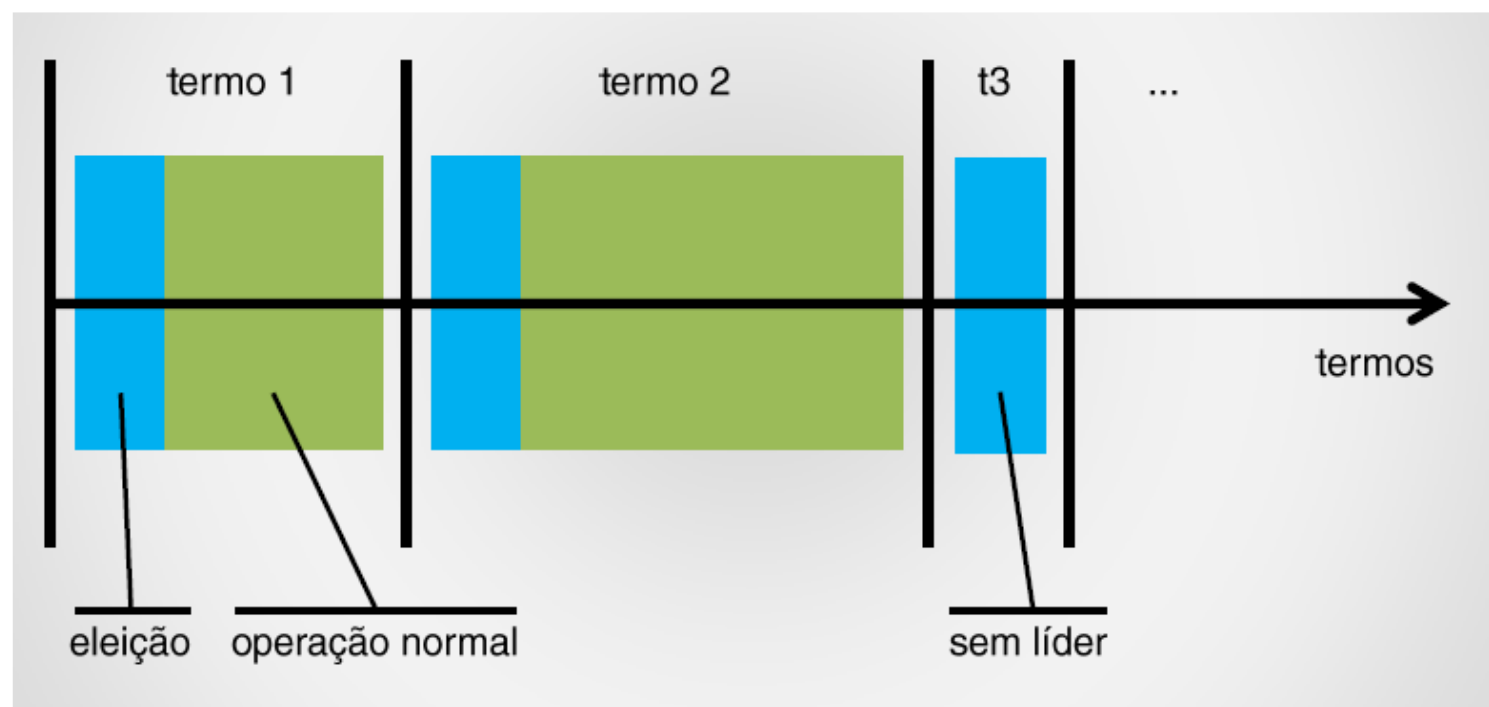
- Only a server with an up-to-date log can become leader

## RAFT Overview

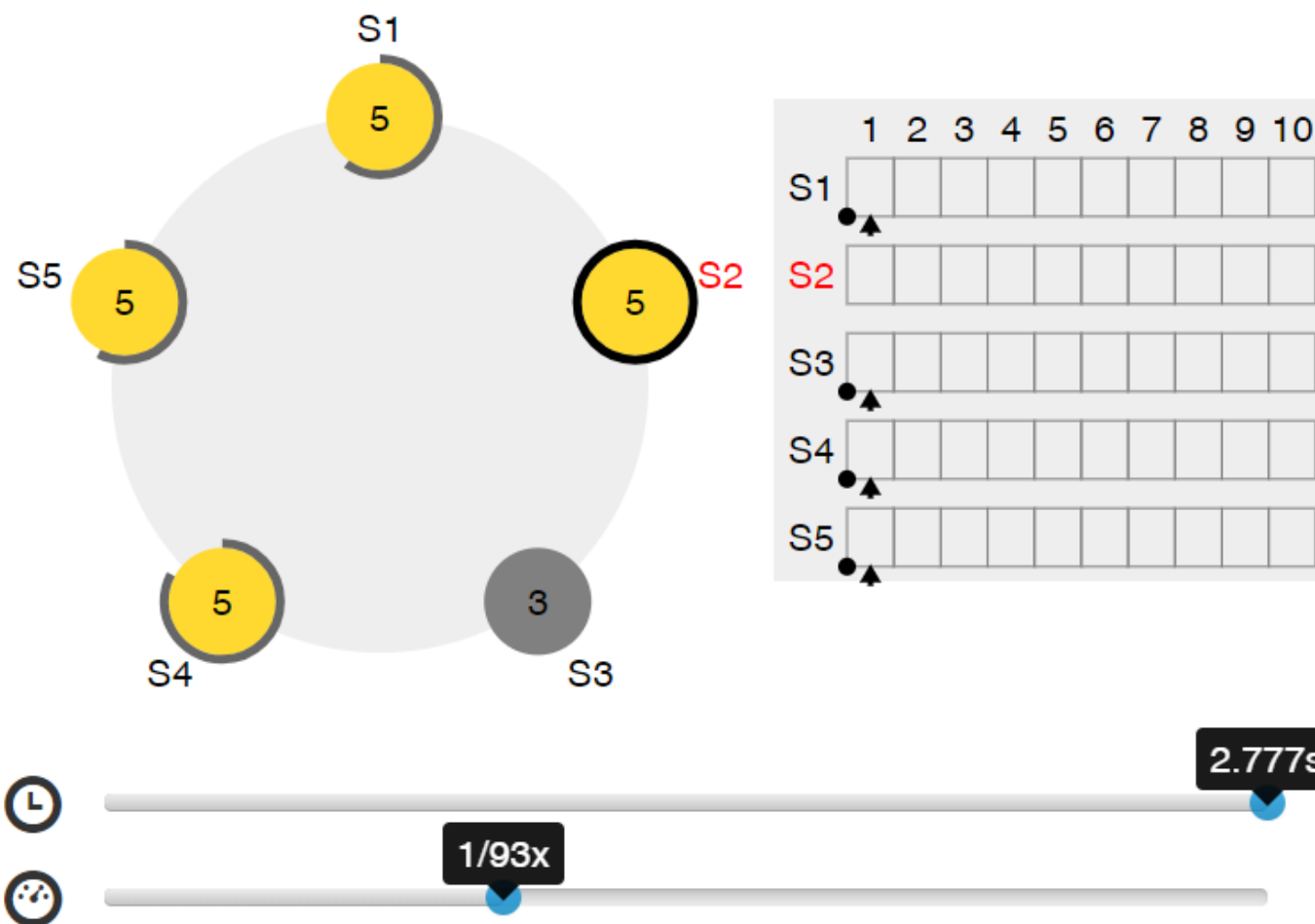


- Candidatos tentam se eleger como líder, e acabam competindo entre si
- Sempre existe no máximo um líder, e todos os demais nós são followers ou candidatos.
- O líder responde a todas as requisições de clientes e os followers são passivos (só respondem requisições do líder).

## Tempo Dividido em Termos



## Explicação Visual

<https://raft.github.io/>



# Dúvidas?

