Fontes principais

1. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução
   `http://www.ime.usp.br/~song/papers/jai01.pdf`
2. N. A. Lynch: Distributed Algorithms, Morgan Kaufmann Publishers, Inc., 96

   ▷ Algoritmos distribuídos

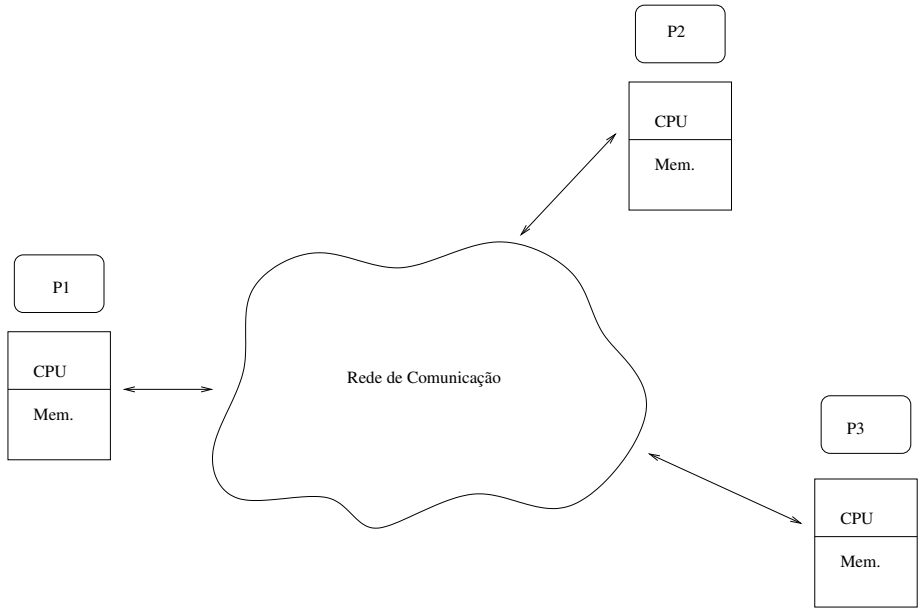
Sistemas distribuídos

Sistemas distribuídos

Coleção de entidades independentes (nós) que cooperam entre si, através de comunicação, para resolver um problema que não pode ser resolvido individualmente por cada nó.

Sistemas distribuídos

Um sistema distribuído existe sempre que houver comunicação entre nós autônomos (agentes inteligentes) e geograficamente distribuídos



Características de Sistemas distribuídos

- Ausência de um relógio global;
- Ausência de memória compartilhada;
- Separação geográfica e incerteza em tempos de comunicação e processamento;
- Possibilidade de falhas independentes de nós e da rede; e
- Dificuldade (impossibilidade) de conhecer o estado global do sistema.

Algoritmos distribuídos

- Algoritmos distribuídos são programas que executam em nós de uma rede (com atrasos e confiabilidade não conhecidos).

- É composto por processos que conjuntamente e de forma coordenada, realizam uma tarefa e precisam manter a consistência.

Premissas desejáveis

O que esperamos de um algoritmo?

- Estar correto
- Ter baixa complexidade

Em algoritmos distribuídos tudo depende do ambiente de execução

- Grau de sincronismo
- Garantia de entrega de mensagens
- Possibilidade de falhas (nós e redes)

Comunicação entre processos (interprocessos)

P_i é um processo que:

- Encapsula um estado (variáveis com valores)
- Reage a eventos externos
- Interage através de envio de mensagens

Sincronização de sistemas distribuídos

Sincronização de sistemas distribuídos

Tempo lógico

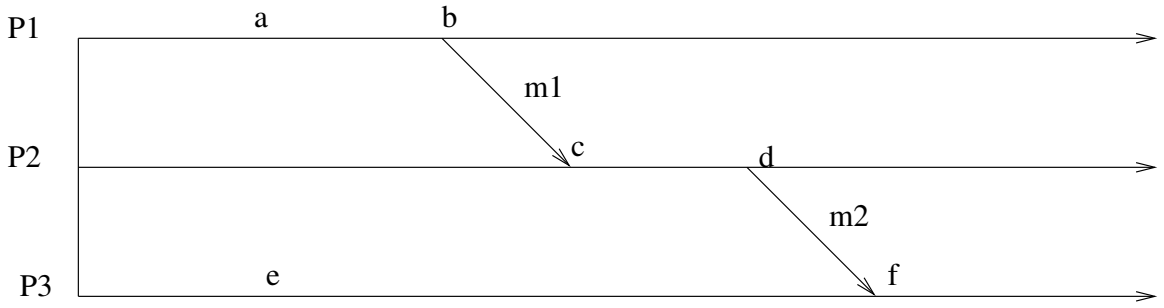
- Em um processo, os eventos são ordenados pelo tempo do relógio local
- Não é possível sincronizar os relógios físicos perfeitamente, logo não tem como utilizá-los para determinar a ordem de ocorrência de 2 eventos em um sistema distribuído.

Sincronização de sistemas distribuídos

Solução para o problema:

- Utilizar um esquema baseado em **causalidade** (*happened before* = "aconteceu antes")
- $a \rightarrow b$, se e somente se, a ocorre antes de b
- Invés de sincronizar os relógios físicos ordenamos os eventos

Sincronização de sistemas distribuídos



$a \rightarrow b$ (em P_1)

$c \rightarrow d$ (em P_2)

$b \rightarrow c$ por causa de m_1

$d \rightarrow f$ por causa de m_2

$a//e$ eventos concorrentes

Sincronização de sistemas distribuídos

Duas situações:

(1) Se os eventos e_1 e e_2 acontecem no mesmo processo, então $e_1 \rightarrow e_2$ (ordem observada entre os processos);

(2) Quando o processo envia uma mensagem m para outro processo, o $envia()$ acontece antes do $recebe()$ ($envia(m) \rightarrow recebe(m)$.

Note que a relação $\rightarrow$ é transitiva.

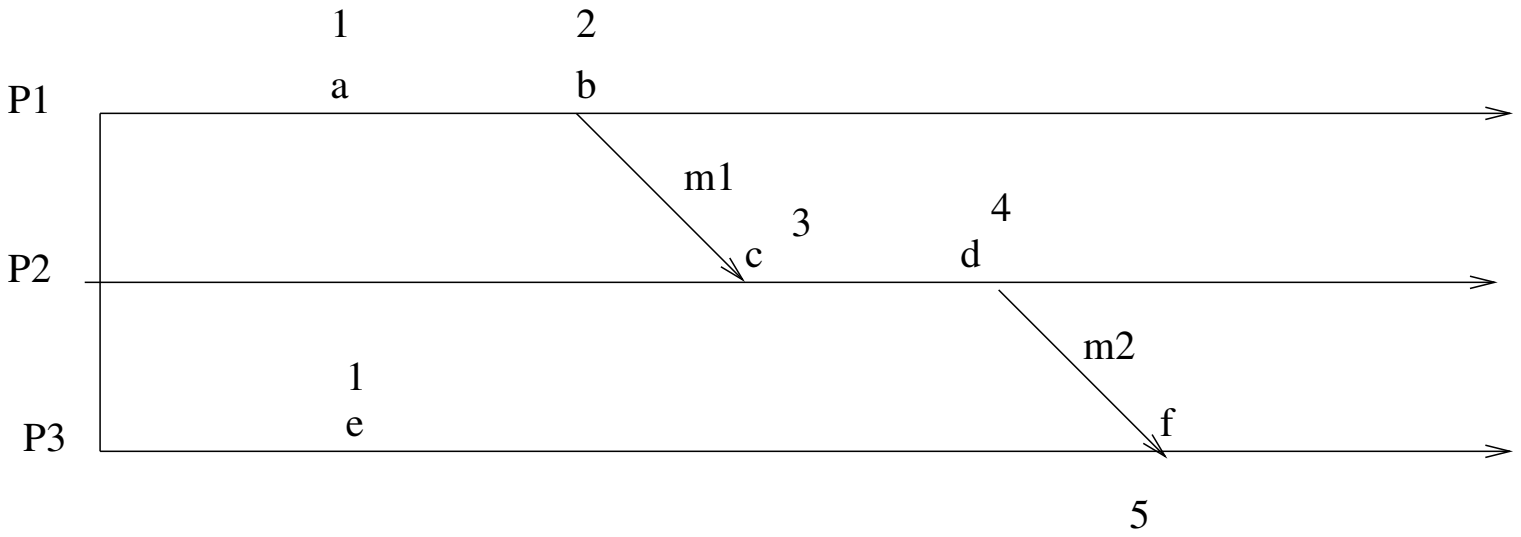
O relógio lógico de Lamport (1978)

- Mecanismo numérico para capturar e representar a relação $\rightarrow$ (acontece antes);
- Cada processo P_i tem um relógio lógico (contador) designado por L_i;
- Não está ligado a um relógio físico.

O relógio lógico de Lamport (1978)

Timestamp (marcas de tempo) são colocados nos eventos de acordo com as regras.

- Evento local: L_i é incrementado antes de colocá-lo como timestamp em um evento
- $envia$: Quando P_i envia m, ele copia em m o valor do seu L_i
- $recebe$: Quando P_i recebe uma mensagem m com timestamp t, ele efetua o cálculo $L_j = max\{L_j, t\} + 1$



Exemplo

- L_1, L_2 e L_3 são inicializados com 0 (zero)
- No envio de m_1, P_1 manda junto $L_1 = 2$
- P_2 recebe m_1 e calcula $L_2 = max\{0, 2\} + 1$

Exclusão mútua

Exclusão mútua

Problema: recursos não podem ser usados simultaneamente por vários processos

Solução: garantir exclusividade de acesso (exclusão mútua)

Exclusão mútua

Regiões críticas

- Concorrência
- Consistência

Semáforos: Sistemas monoprocessados ou com multiprocessados com memória compartilhada.

Exclusão mútua

- Sistemas distribuídos?

  Resposta: semáforos distribuídos

  - Centralizado
  - Descentralizado (Não veremos) - pouco eficiente
  - Distribuído
  - Anel

Algoritmo centralizado

Similar a um sistema monoprocessado

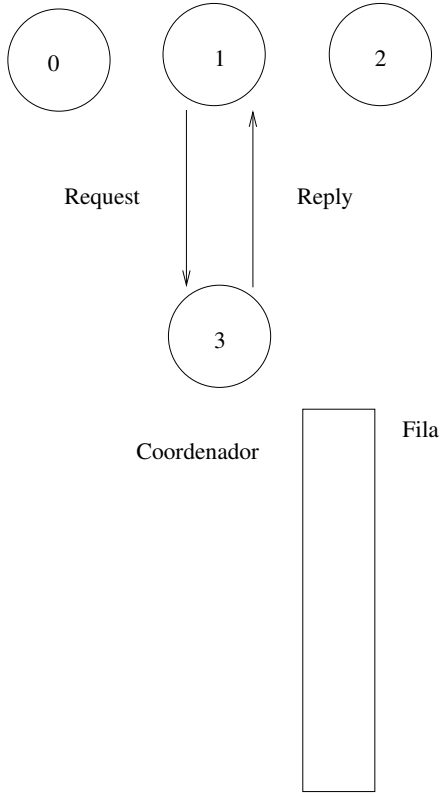
Um dos processos é eleito coordenador

O coordenador garante a exclusão mútua

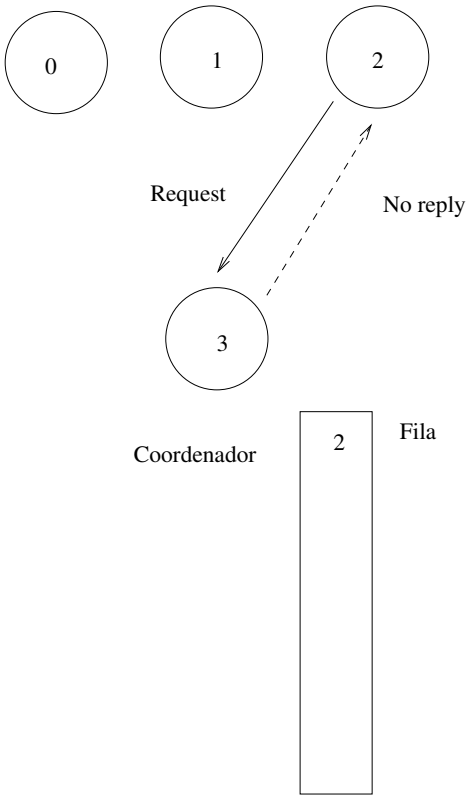
Três tipos de mensagem

- Request (requisição)
- Reply (conceder)
- Release (Liberação)

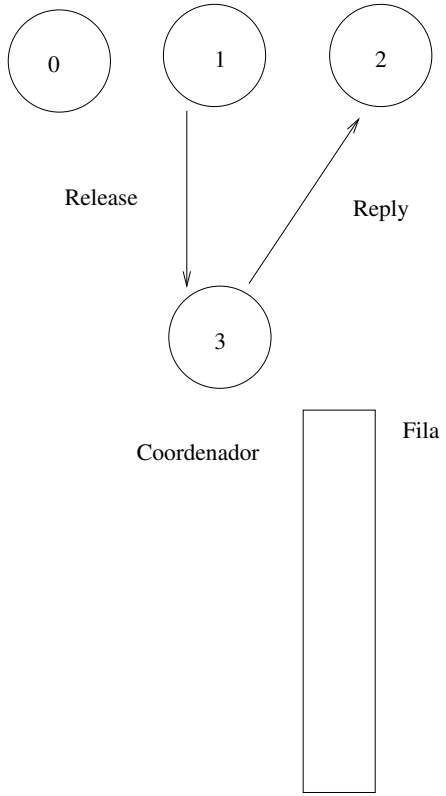
Algoritmo centralizado



1 solicita permissão para o coordenador e entra na RC



2 solicita permissão para entrar na RC, mas o coordenador "nega"



1 libera a RC e o coordenador e o coordenador concede permissão para 2

Algoritmo distribuído

Não existe coordenador e decisões são tomadas em grupo

Proposta de Ricarte Agrawala é uma melhoria de idéias de Lamport

Tal solução exige ordenação global dos eventos do sistema (*timestamp*)

Procedimento

Se um processo i deseja entrar em uma Região Crítica x (RCx) ele gera um *Timestamp* (TS) e envia $Request(i, RCx, TS)$.

O processo i saberá que RCx está livre se receber *Reply* de todos os outros processos.

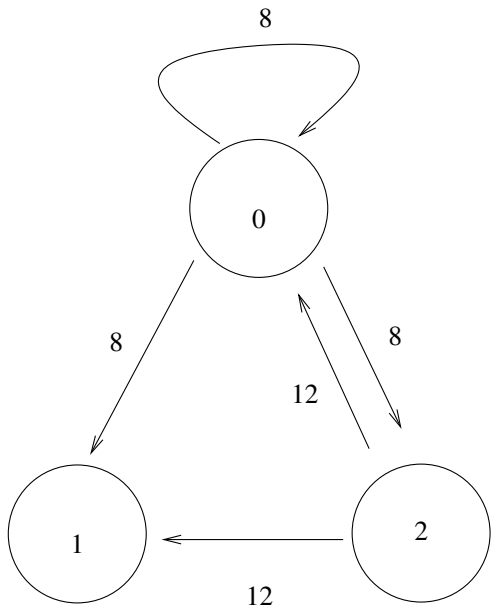
Algoritmo distribuído

Quando um processo recebe *Request*

- Se não estiver acessando a RCx e não quiser acessá-la devolve OK (Reply)
- Se ele estiver acessando a RCx não responde e coloca o Request em uma fila de espera
- Se ele também deseja acessar a RCx, mantém uma fila de espera e envia o Reply (OK) para o Request com menor TS

Algoritmo distribuído

Exemplo:

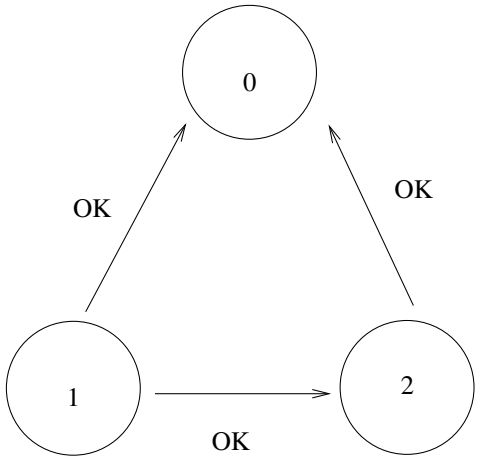


Processos 0 e 2 querem acessar um recurso ao mesmo "tempo"

8 e 12 são timestamps dos requests.

Algoritmo distribuído

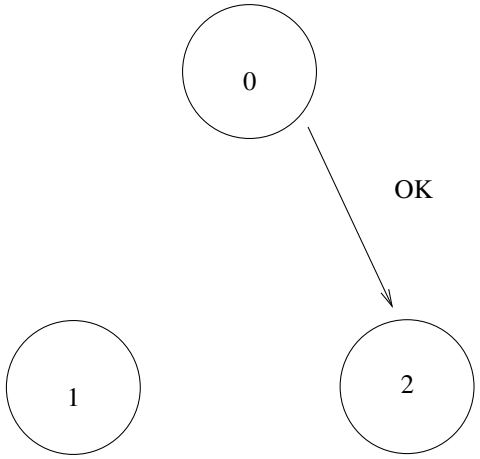
Acessa o recurso (Região crítica)



Processos 1 não tem interesse no recurso e envia OK para 0 e 2

0 e 2 percebem o conflito, mas vence quem tem o menor timestamp.

Algoritmo distribuído



Quando o processo 0 liberar o recurso ele envia OK para o processo 2 que estava na fila de espera.

Algoritmo em anel

Conhecido como Token Ring

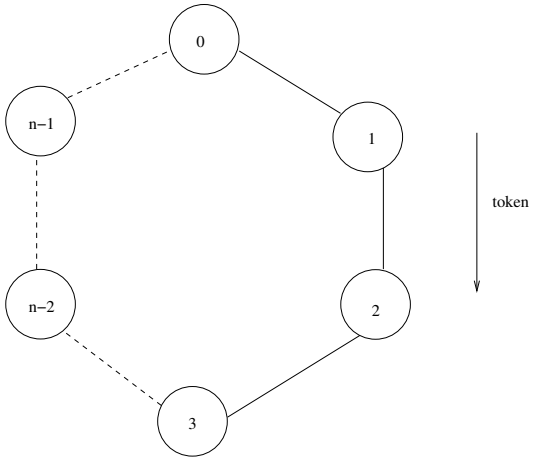
Processos pertence a um anel lógico

Quem possui o token (ficha) é o que tem direito a acessar a região crítica.

Algoritmo em anel

Algoritmo	Problema
Centralizado	Queda do coordenador
Distribuído	Queda de qualquer processo
Anel	Perda do token ou queda de um processo

Algoritmo em anel



Quando o anel é inicializado o processo 0 recebe o token

Algoritmo de eleição

Algoritmo de eleição

Necessidade de um coordenador em ambiente distribuído

A solução geral consiste em eleger o processo com maior ID

Todos os processos precisam concordar com a eleição

Algoritmo do valentão

Algoritmo do valentão

Desenvolvido por Garcia-Molina (1982)

Quando um processo percebe que o coordenador não está respondendo requisições, ele inicia uma nova eleição.

Convocações de eleição

- P envia mensagem de eleição para todos os processos com IDs maiores.

Algoritmo do valentão

Eleições

- ▷ Se ninguém responde, P vence a eleição e torna-se coordenador.
- ▷ Se algum processo com id maior responde, ele desiste.

Quando um processo recebe mensagem de eleição

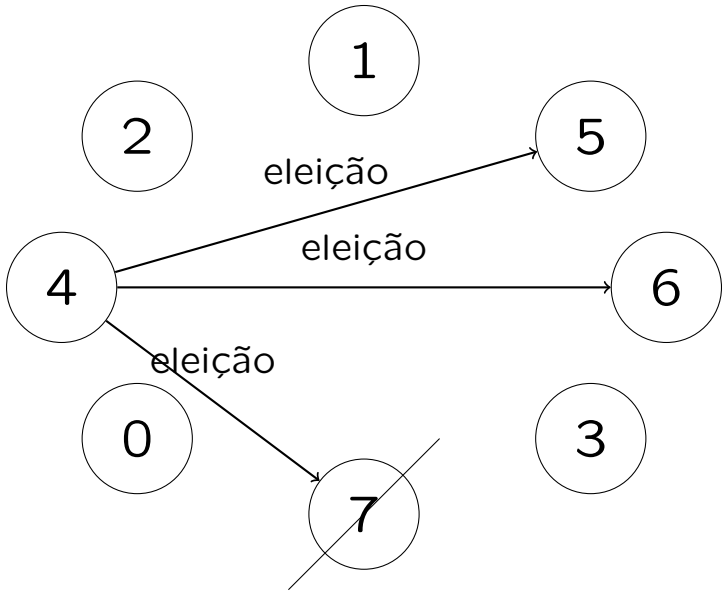
- ▷ Se o ID recebido é menor que o dele, envia OK para o remetente para indicar que está vivo e assume a coordenação.

Algoritmo do valentão

Todos os processos desistem, menos o que tiver maior ID.

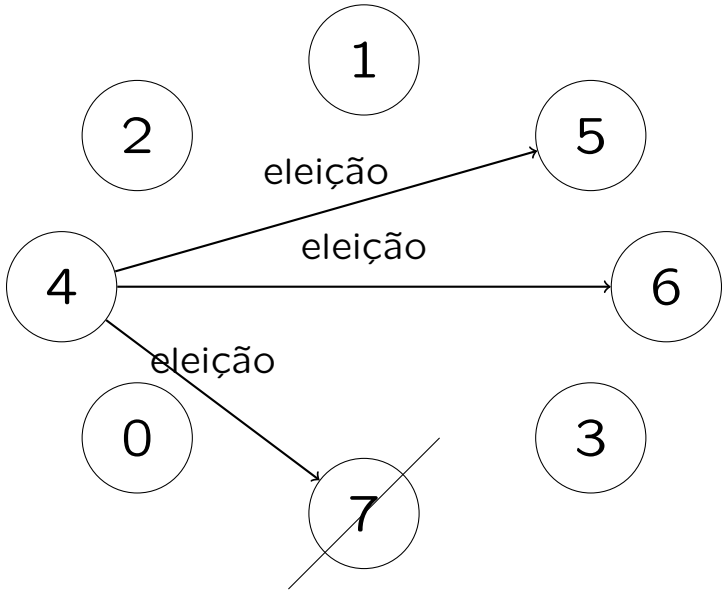
Se o processo que estava indisponível voltar é iniciada uma nova eleição.

Algoritmo do valentão



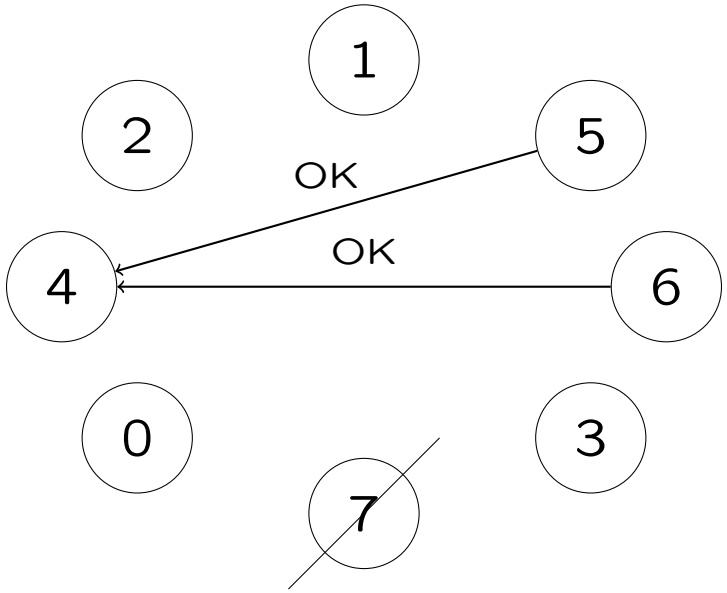
Processo 7 caiu.

Algoritmo do valentão



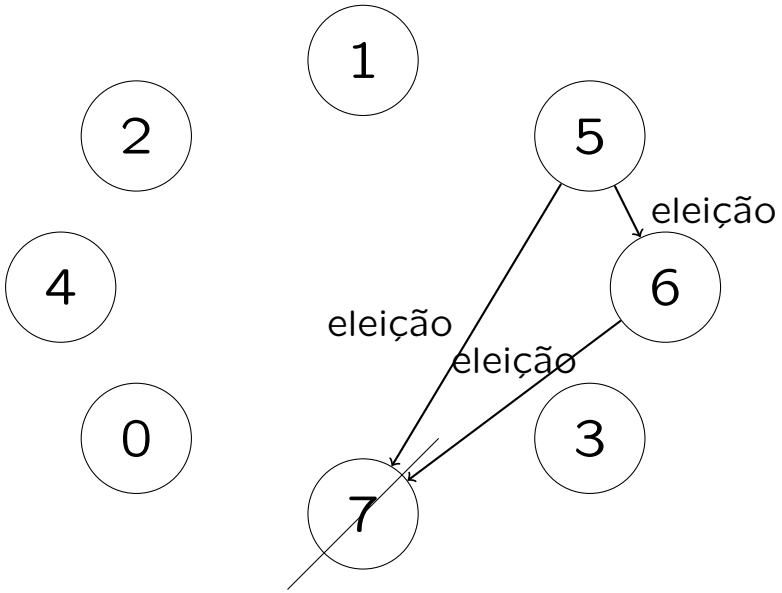
Processo 4 percebeu e convocou nova eleição

Algoritmo do valentão



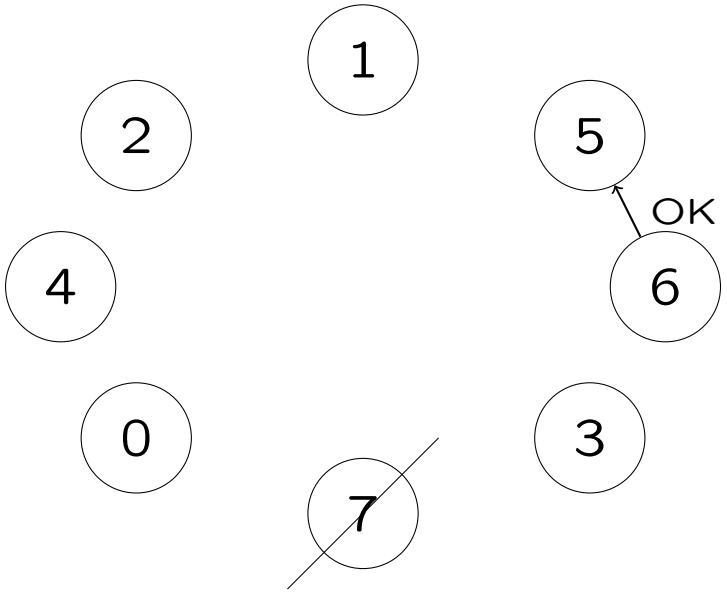
Processos 5 e 6 respondem OK para o processo 4 que desiste.

Algoritmo do valentão



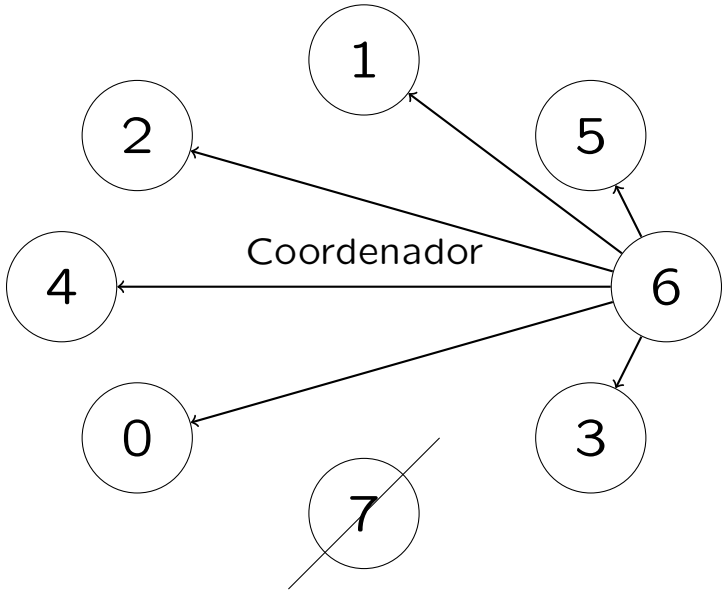
Processos 5 e 6 continuam a eleição.

Algoritmo do valentão



Processos 6 envia OK para o processo 5 que desiste.

Algoritmo do valentão



Processos 6 avisa aos demais processos que ele é o novo líder.

Algoritmo em anel

Algoritmo em anel

Não utiliza token

Processos fisicamente ou logicamente ordenados - cada processo reconhece seu sucessor

Quando um processo percebe que o coordenador caiu ele inicia uma eleição.

Algoritmo em anel

O processo envia mensagem de eleição para o sucessor, com seu ID.

Se o sucessor está indisponível, envia para o próximo, e assim sucessivamente.

A cada passo, o processo que recebe a mensagem adiciona seu ID e repassa para o sucessor.

Algoritmo em anel

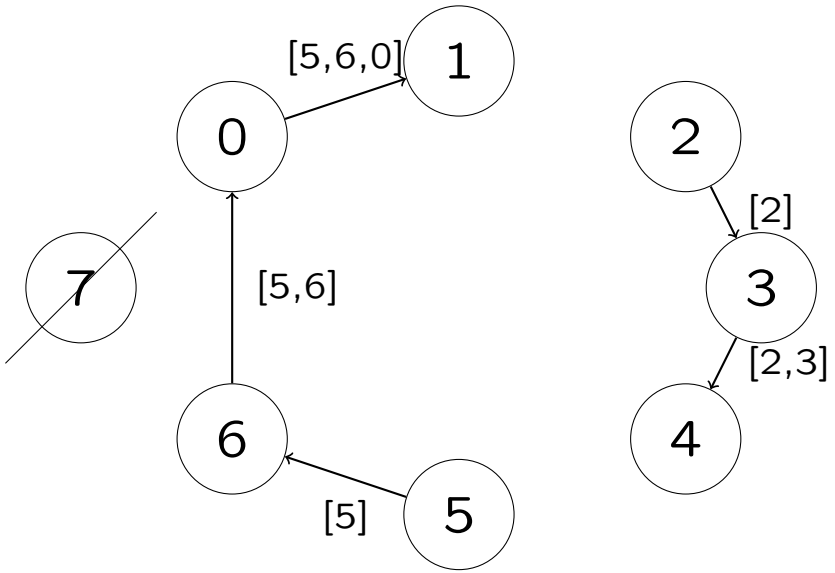
A mensagem deve retornar para quem iniciou a eleição.

Quem enviou reconhece a mensagem por conter o seu ID.

Decide quem deve ser o coordenador pelo maior ID.

Avisa com uma nova rodada quem é o novo coordenador.

Algoritmo em anel - Exemplo



Processo (coordenador) 7 caiu, 2 e 5 percebem e iniciam eleição "simultaneamente", no final o 6 é o novo coordenador.

Tolerância a falhas

Tolerância a falhas

Tolerância a falhas: propriedade que permite que sistemas continuem operando adequadamente após falha.

Resiliência: conceito físico, mas adotado no contexto computacional, pode ser entendido como a capacidade de adaptação ou recuperação de um sistema.

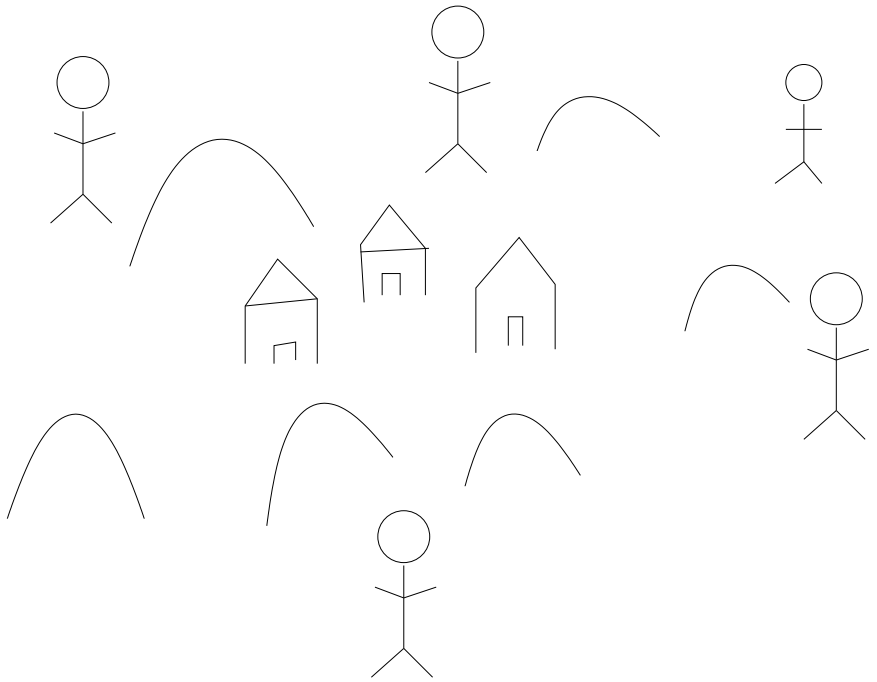
Tolerância a falhas

Falha: Quando o componente de um sistema deixa de se comportar conforme a especificação correta.

Erro: Causa de falha.

Problema dos Generais Bizantinos

Problema dos Generais Bizantinos



Problema dos Generais Bizantinos

Generais binzantinos sitiam uma cidade

- ▷ alguns generais são traidores
- ▷ mas devem chegar no concenso de atacar ou recuar.
- ▷ generais traidores não podem/devem atrapalhar o concenso.

Concordância bizantina: consenso na presença de falhas arbitrárias.

Problema dos Generais Bizantinos

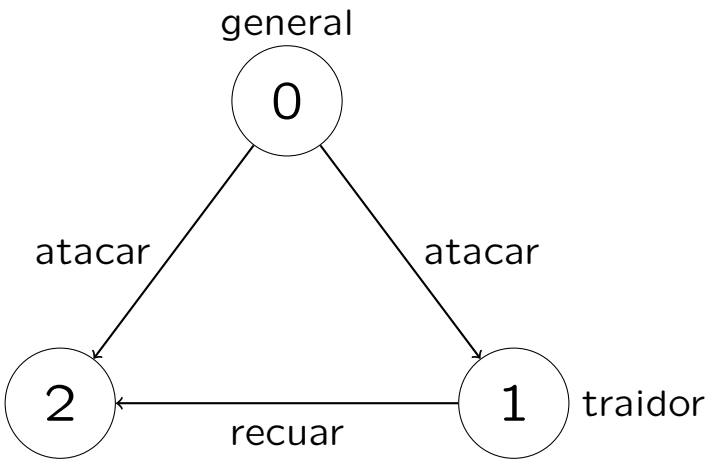
Meta:

Conseguir o consenso entre todos os nós sem defeitos (nós não traidores)

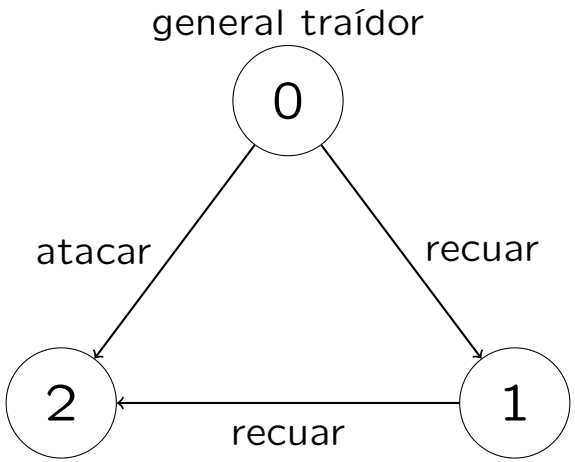
▷ generais não traidores devem tomar a mesma decisão (atacar ou recuar)

▷ generais traidores podem enviar mensagem diferentes para generais diferentes.

Exemplo com 3 Generais Bizantinos



Exemplo com 3 Generais Bizantinos



Problema insolúvel, o nó 2 não consegue distinguir uma situação da outra.

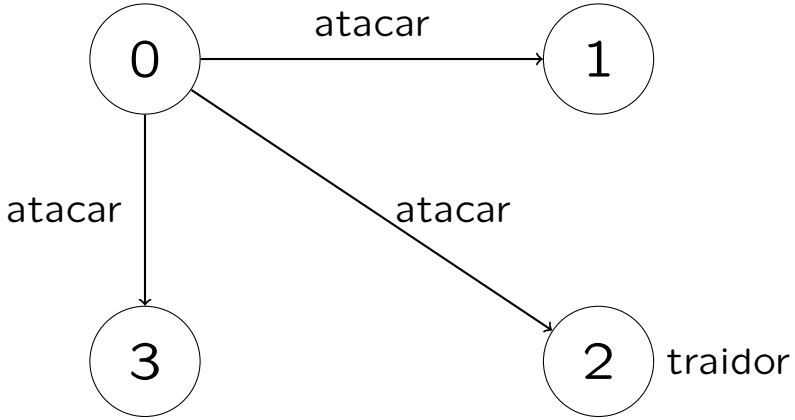
Problema dos Generais Bizantinos

Lamport, Shostak e Pease (1982)

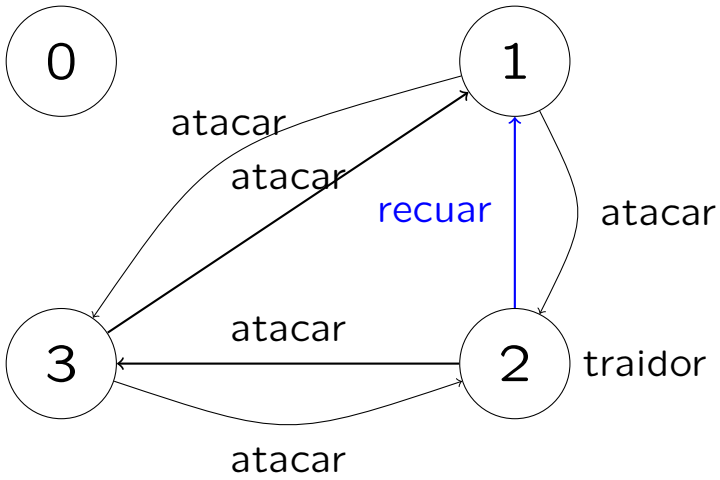
▷ Dados n processos (generais), pode ser tolerado até m generais traidores sendo que $n \geq 3m + 1$.

Exemplo: 4 processos podem tolerar até 1 processo traidor.

Problema com 4 Generais Bizantinos



Problema com 4 Generais Bizantinos



0,1 e 3 decidem atacar (consenso)

Note que estamos considerando um sistema síncrono (timeout)

Problema dos Generais Bizantinos

O que acontece em um cenário com 9 generais, onde 1 general traidor envia 4 mensagens de atacar para 4 generais e 4 mensagens de recuar para os outros 4 generais?

Problema dos Generais Bizantinos

E se fossem 6 mensagens de atacar e 2 de recuar?

Considere também o caso com 5 mensagens de recuar e 3 de atacar.

Problema dos Generais Bizantinos

Diversos algoritmos foram propostos para tratar o problema da concordância bizantina.

Teorema 1. *Se* $2/3 + 1$ *dos generais não são traidores, então existe uma solução (algoritmo) que resulta em uma ação comum que independe das mensagens enviadas pelos traidores.*

Problema dos Generais Bizantinos

Teorema 2. *Se* 1/3 *ou mais dos generais são traidores, então não existe uma solução para o problema.*

Exercício

Implemente um programa com MPI que trate/resolva o problema dos generais bizantinos.

Fim