UNIFESSPA
UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ

Universidade Federal do Sul e Sudeste do Pará

Sistemas Distribuídos

Prof.: Warley Junior

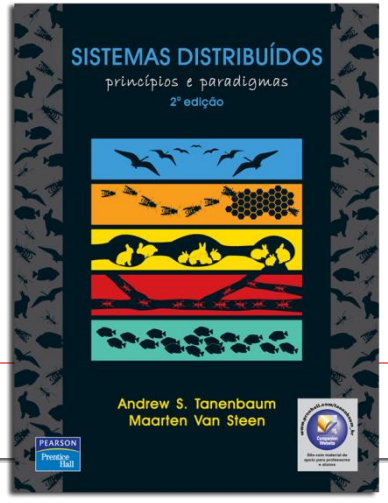
wmvj@unifesspa.edu.br

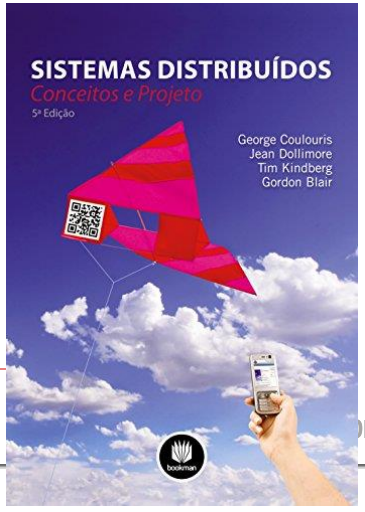
Agenda

- AULA 13:
- Coordenação e Acordo
  - Algoritmo do Valentão
  - Algoritmo do Anel
  - Exclusão Mútua
    - Algoritmo centralizado
    - Algoritmo descentralizado
    - Algoritmo distribuído
    - Algoritmo *Token Ring*

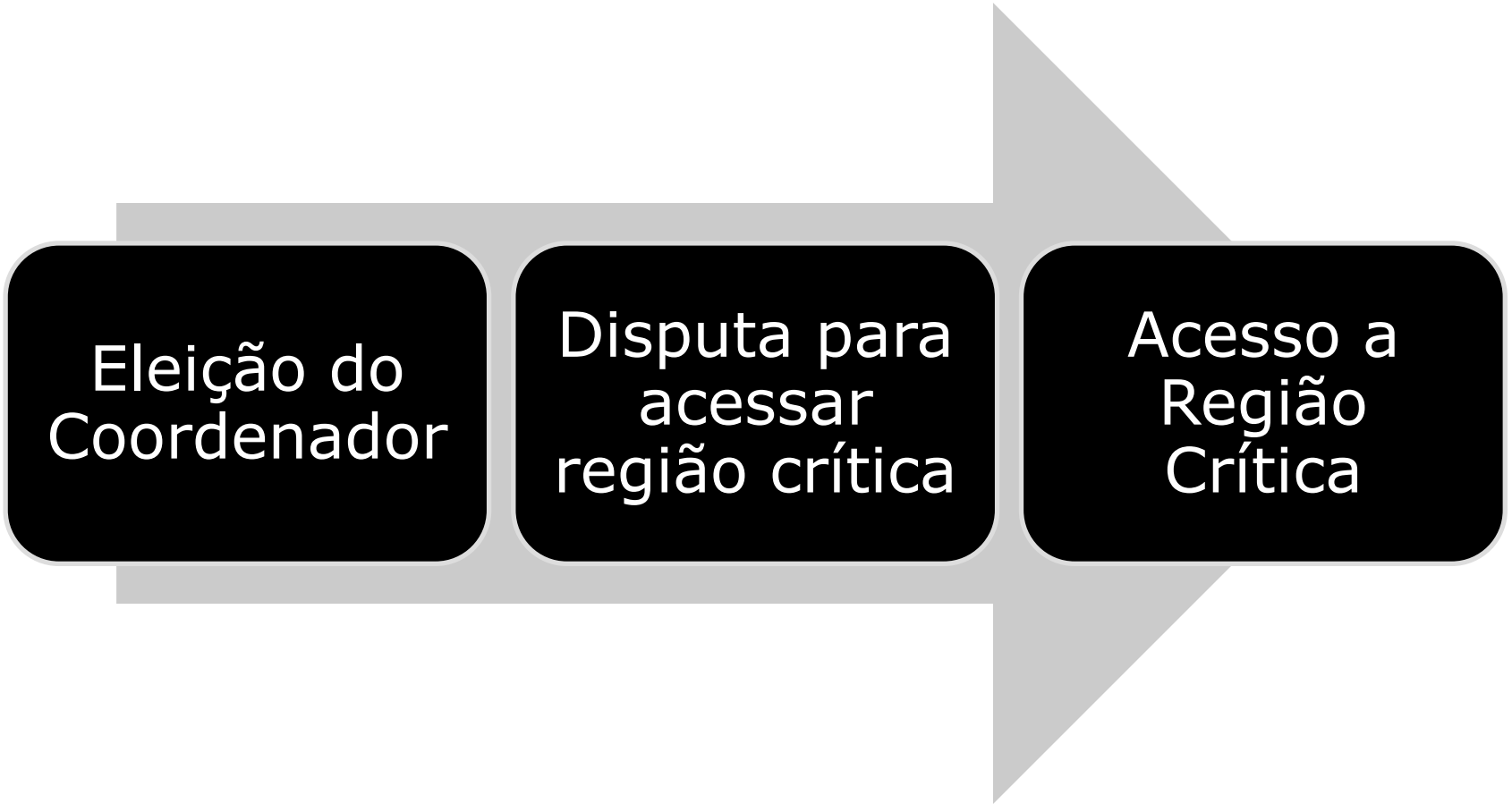
Leitura Prévia

- COULOURIS, George. **Sistemas distribuídos: conceitos e projetos**. 4ª ed. Porto Alegre: Bookman, 2013.
  - Capítulo 15.
- TANENBAUM, Andrew S. **Sistemas distribuídos: princípios e paradigmas**. 2ª ed. São Paulo: Pearson Prentice Hall, 2007.
  - Capítulo 06.





Passo a passo para Eleição e Coordenação



Eleição e Coordenação

- Muitos algoritmos requerem um processo para atuar como coordenador.
- Exemplo: Coordenador no algoritmo centralizado de exclusão mútua.
- Em geral, não importa qual processo irá atuar como coordenador, mas é preciso elegê-lo.
  - Como eleger um coordenador?

Eleição e Coordenação

- Premissas:
  - Cada processo possui um número único (por exemplo, seu endereço de rede) separado.
  - Todo processo conhece o número do processo de qualquer outro processo.
  - Os processos não sabem quais processos estão “vivos” e quais estão “mortos”.

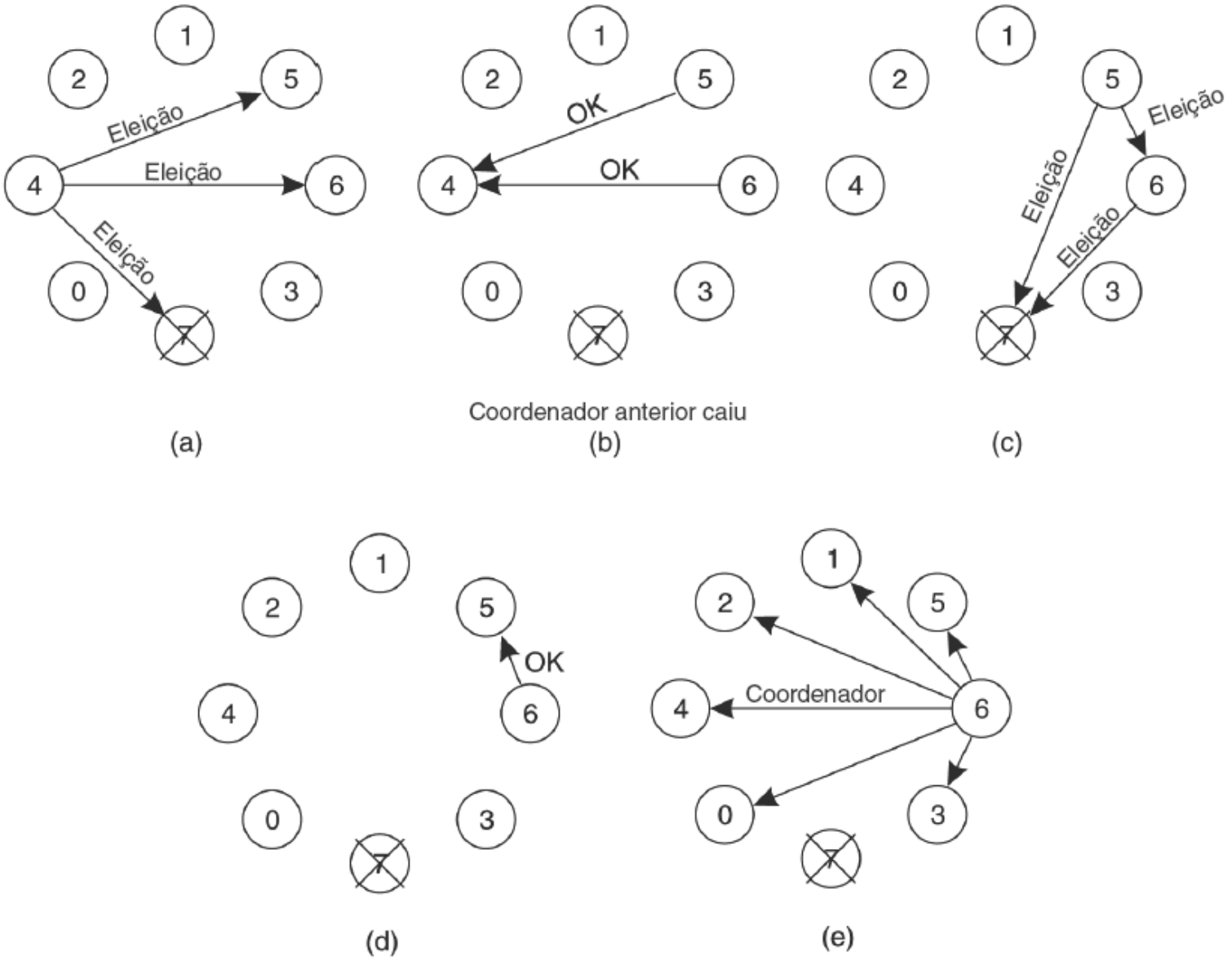
Eleição e Coordenação

- Abordagem:
  - Localize o processo com o número de processo maior e eleja-o como coordenador.
  - Os algoritmos de eleição diferem na forma como fazem a localização.

Algoritmo do valentão (*bully*)

- Inventado por Garcia-Molina (1982);
- Todos os nós possuem um identificador;
- Um nó *P* inicia eleição quando nota que o coordenador não responde:
  1. *P* envia uma mensagem ELEIÇÃO a todos os processos de números mais altos
  2. Se nenhum responder, *P* vence a eleição e se torna o coordenador
  3. Se um dos processos de número mais alto responder, ele toma a eleição e o trabalho de *P* está concluído
- Dessa forma, o nó com maior identificador é eleito!

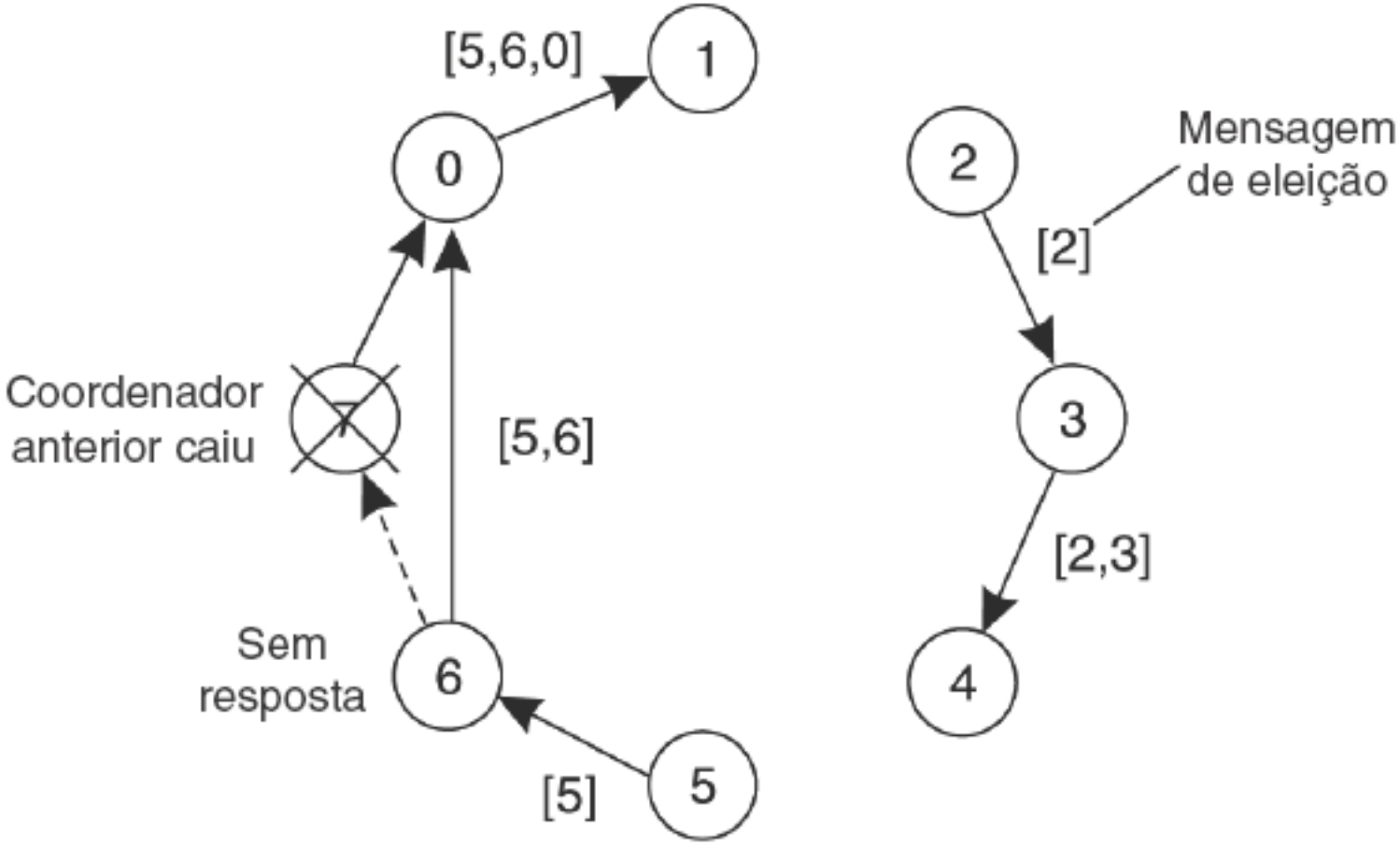
Algoritmo do Valentão



Algoritmo do Anel

- Baseado na utilização de anel (físico ou lógico), mas não usa ficha:
  - Quando qualquer processo nota que o coordenador não está funcionando, monta uma mensagem ELEIÇÃO com seu próprio número e o envia a seu sucessor ou próximo que esteja em funcionamento;
  - A cada etapa, o remetente adiciona seu número de modo a se tornar também um candidato à eleição de coordenador;
  - Ao chegar ao primeiro, este envia a mensagem COORDENADOR contendo o maior identificador da lista.

Algoritmo do Anel



Exclusão Mútua em SD

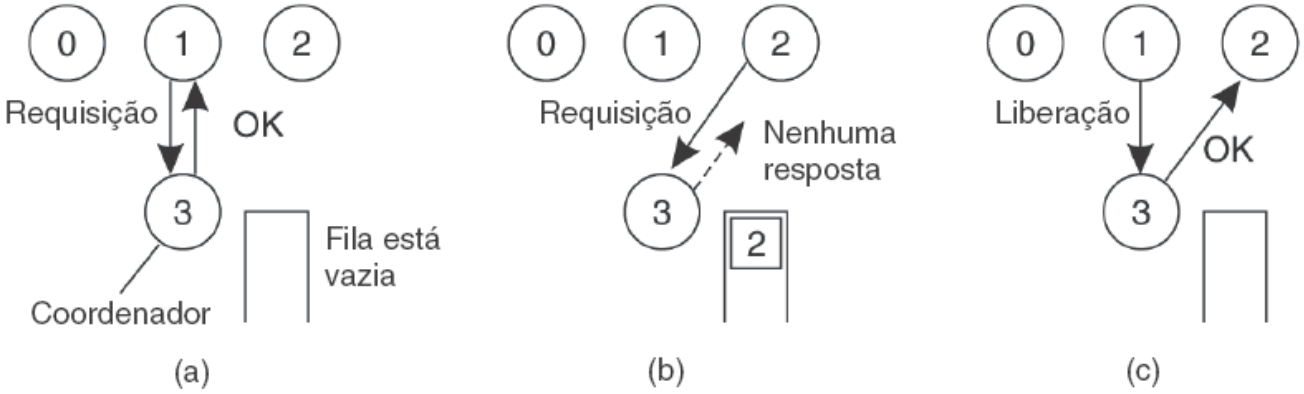
- Como garantir que o acesso concorrente de recursos não gera situações de inconsistência nos dados?
  - Exclusão mútua!
  - Semáforos não são viáveis em Sistemas Distribuídos
    - Memória não é compartilhada
- Duas categorias para soluções em SD:
  - Baseada em permissão
    - Processo que quer recursos deve pedir permissão aos demais
  - Baseada em fichas *(tokens)*
    - Só o processo que está com a ficha tem permissão de acesso

Exclusão Mútua em SD

- A melhor maneira de se conseguir exclusão mútua em um SD é imitar o que é feito em um sistema local.
- Um processo é eleito como Coordenador.
- Quando um processo quer entrar na região crítica, ele envia uma mensagem requisitando ao Coordenador.
- Se nenhum outro processo esta na região crítica, o Coordenador dá o OK.

Algoritmo Centralizado

- Processo 1 pede permissão ao coordenador para entrar em uma região crítica. A permissão é concedida.
- Processo 2 então pede permissão para entrar na mesma região crítica. O coordenador não responde.
- Quando o processo 1 sai da região crítica, ele avisa o coordenador, quando então responde a 2.



Algoritmo Centralizado - Vantagens

- ☺ Garante a exclusão mútua.
- ☺ É justo (os pedidos são concedidos na ordem em que são recebidos).
- ☺ Sem inanição (um processo não espera para sempre).
- ☺ Fácil de implementar.

Algoritmo Centralizado - Desvantagens

- ☹ Coordenador: um único ponto de falha.
- ☹ Traz um gargalo de desempenho.
- ☹ Se os processos normalmente bloqueiam após fazer um pedido, eles não podem distinguir um coordenador morto da "permissão negada".

Algoritmo descentralizado

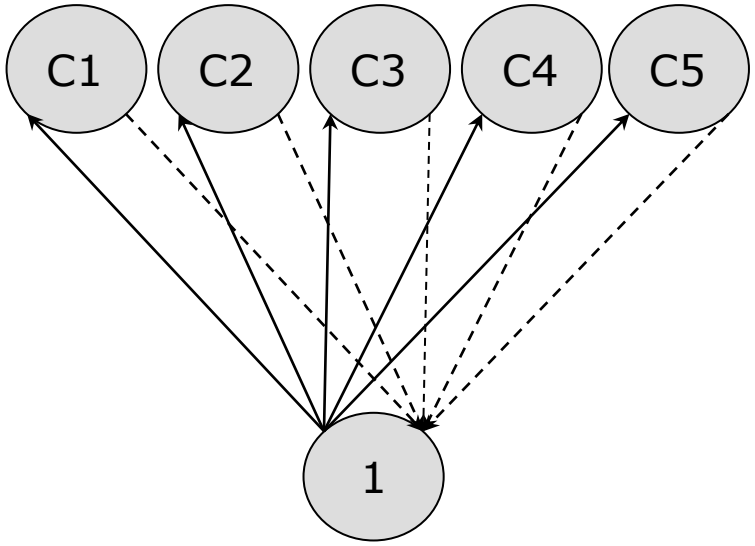
- Para maior disponibilidade, podemos replicar o coordenador
  - coord-1, coord-2, ..., coord-n
- Requisitante precisa receber maioria de votos de coordenadores: **`m > n/2`**
  - *m*: quantidade de votos
  - *n*: quantidade de réplicas
- Quem não tem maioria tenta novamente depois de um tempo aleatório

Algoritmo descentralizado

CENÁRIO 1

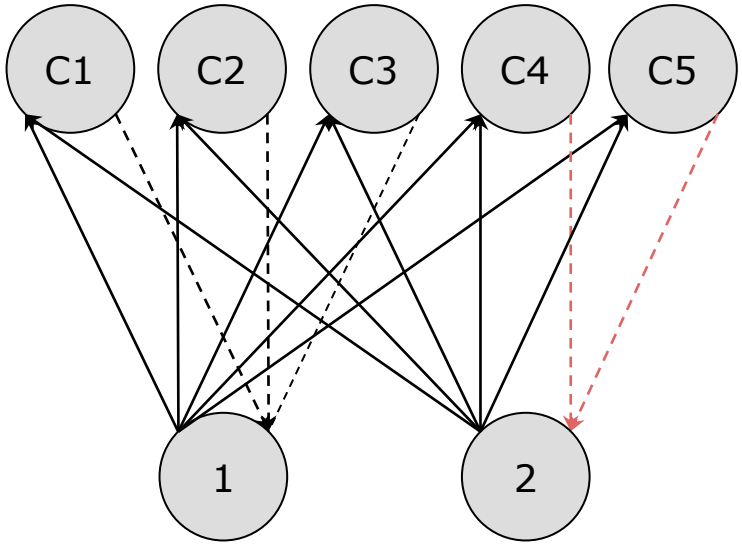
n = 5 réplicas

m_1 > n / 2 ?
m_1 = 5 votos
5 > 2,5
(**permitido!**)



Algoritmo descentralizado

CENÁRIO 2



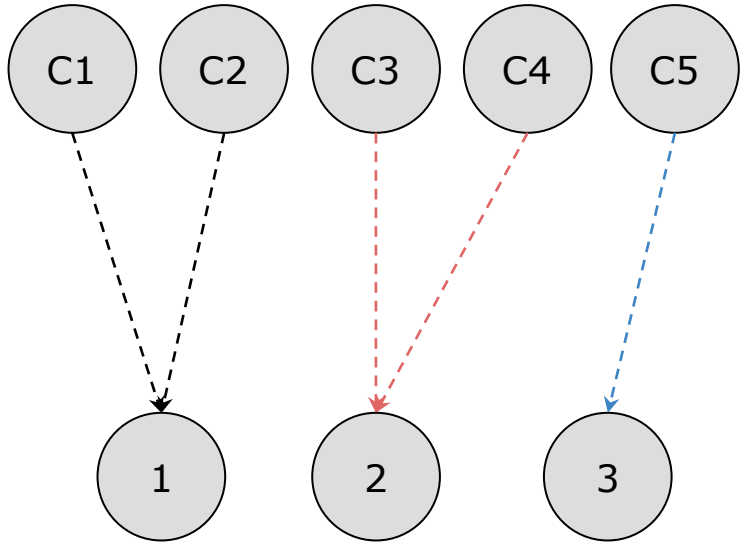
n = 5 réplicas

m_1 > n / 2 ?
m_1 = 3 votos
3 > 2,5
(**permitido!**)

m_2 > n / 2 ?
m_2 = 2 votos
2 < 2,5

Algoritmo descentralizado

CENÁRIO 3



n = 5 réplicas

m_1 > n / 2 ?
m_1 = 2 votos
2 < 2,5

m_2 > n / 2 ?
m_2 = 2 votos
2 < 2,5

m_3 > n / 2 ?
m_3 = 1 voto
1 < 2,5

Ninguém recebe permissão!

Algoritmo descentralizado

- Pró
  - 😊 Torna a solução centralizada menos vulnerável a falhas de único coordenador
- Contras
  - ☹ Não protege contra inanição
    - Se muitos nós querem acessar o mesmo recurso, nenhum nó conseguirá votos suficientes e os recursos ficarão ociosos
  - ☹ Há chance (muito baixa) de dar a 2 nós acesso simultâneo ao mesmo recurso
    - Em casos de falha de coordenadores

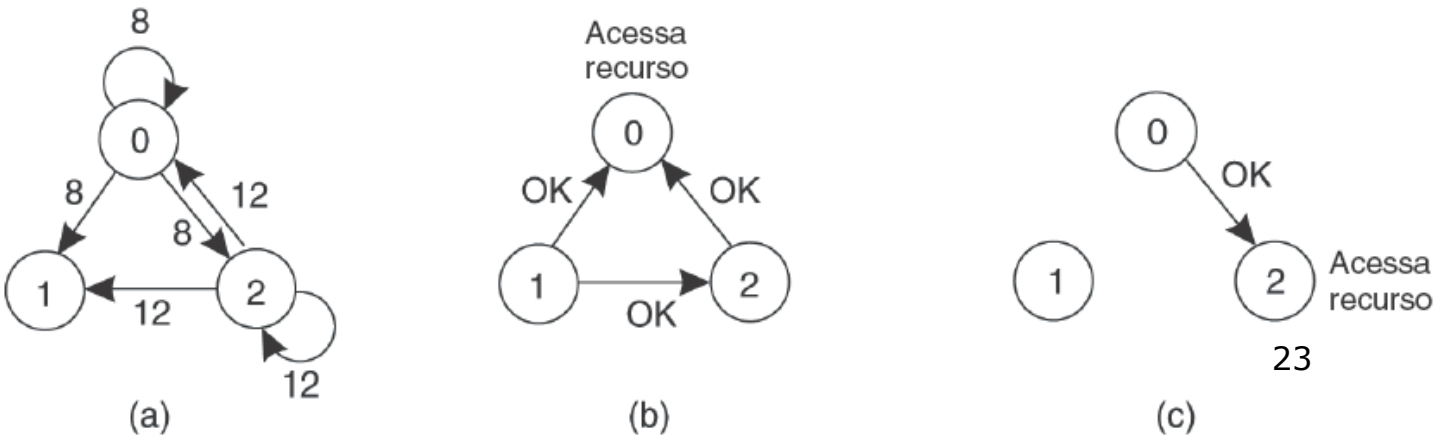
Algoritmo Distribuído

- Quando um processo recebe uma requisição de outro, realiza ação:
  1. Se o receptor não estiver acessando o recurso nem quer acessá-lo, responde "OK"
  2. Se tem acesso ao recurso, não responde e coloca requisição em fila
  3. Se o receptor também quer acessar o recurso, mas ainda não possui a permissão, compara o *timestamp* (hora lógica) da mensagem que chegou com o *timestamp* da mensagem que enviou a todos. O valor mais baixo vence.
- O processo remetente aguarda recebimento da resposta de todos
  - Quando houver permissão de todos, o processo acessa o recurso
- Processo libera o recurso enviando um "OK" a todos os processos

Algoritmo Distribuído

- Dois processos querem entrar na mesma região crítica ao mesmo tempo.
- Processo 0 tem o *timestamp* mais baixo, então ganha.
- Quando o processo 0 finaliza, ele envia um OK, então o processo 2 agora pode entrar na região crítica.

Problema: Falha de qq Processo "crashes"

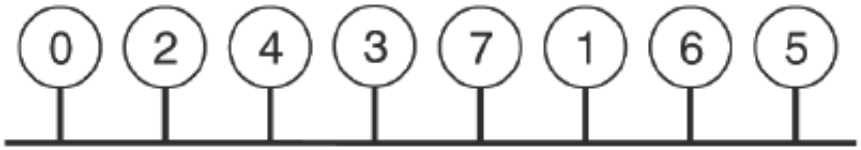


Algoritmo Distribuído

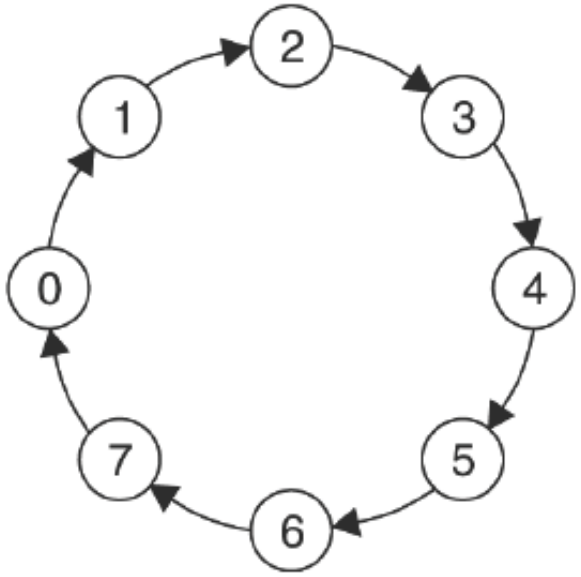
- Vantagens:
  - 😂 Exclusão mútua é garantida, não há deadlock, não há inanição
  - 😂 Quantidade de mensagens necessárias: $2(n-1)$ // n = número de nós
  - 😂 Não há ponto de falha único
- Desvantagens:
  - ☹ Ponto de falha único foi substituído por n pontos de falha
    - se um processo falhar, bloqueia permissões
  - ☹ Deve usar comunicação multicast ou manter uma lista de associação ao grupo em cada processo
    - funciona melhor com poucos processos que se mantêm estáveis

Algoritmo *Token Ring*

- Quando um anel é inicializado, o processo 0 recebe uma ficha.
- A ficha circula ao redor do anel.
- Evita a inanição.



(a)



(b)

Algoritmo Token Ring

- Vantagens
  - Garante a exclusão mútua.
  - Sem inanição.

- Desvantagens
  - Regeneração do token se ele for perdido.
  - Detectar que o token está perdido é difícil – rede pode estar sobrecarregada.