Sincronização em Sistemas Distribuídos

problemas clássicos

- ordenação de mensagens
- exclusão mútua distribuída
- eleição de líder
- ...
- transações

PUC RIO

ordenação de acontecimentos

- relógio físico
  - dificuldades
- relógio lógico
  - não é um relógio de fato...

relógio lógico

- cada processo é modelado como sequência de eventos *e1*, *e2*, …
  - na realidade os eventos que importam são os de entrada/saída
- um evento *e1* precede causalmente outro *e2* se:
  - *e1* aconteceu antes de *e2* em um mesmo proc *P*
  - *e1* é o evento de envio de uma msg *M* e *e2* o evento de seu recebimento
  - existe *e3* tal que *e1* precede *e3* e *e3* precede *e2*
- um relógio lógico associa a cada evento um valor tq
  - e1 precede e2 $\Rightarrow$ rl(e1) < rl(e2)

PUC RIO

relógio lógico

- cada processo P inicialmente faz $RL_P = 0$
- a cada evento *e*, P associa a *e* o valor corrente de RL_P e incrementa RL_P ao enviar uma mensagem *M*, P acrescenta a *M* um timestamp *ts(M)* que é o valor corrente de RL_P
- ao receber uma mensagem *M* com timestamp ts, P faz

  $RL_P = \max (RL_P , ts) + 1$

PUC RIO

exemplo de uso - relógio lógico

- algoritmo de exclusão mútua distribuída
  - implementação centralizada
  - implementação distribuída
    - arbitragem por relógio lógico
    - uso de token

PUC RIO

exclusão mútua com uso de RL

- entrada na região crítica:
  - envia mensagem (reqEM, meuId, meuRL) e entra em espera por oks com RLesp = meuRL
  - espera (ok, ID) de todos os demais processos
- ao receber (reqEM, IdO, RLO)
  - não está na região crítica ou em espera por oks:
    - envia (ok, meuID) para IdO
  - está em espera por oks:
    - se RLO < RLesp, envia (ok, meuID) para IdP
    - caso contrário, coloca req em lista
  - está na região crítica:
    - coloca req em lista
- ao sair da Região Crítica:
  - envia (ok, ID) para cada req na lista

ler

- artigo de Andrews (“Paradigms for interaction…”):
  - seção 6

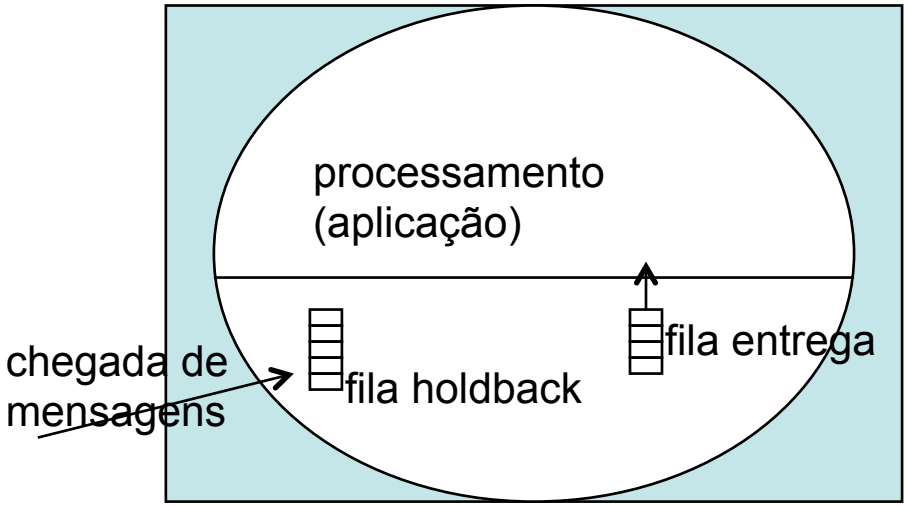
PUC RIO

ordenação em broadcast ou multicast

- ordem total
- ordem causal
  - ligados a acoplamento forte entre processos!
  - ordenação: exemplo de problema de coordenação distribuída

ordenação de mensagens

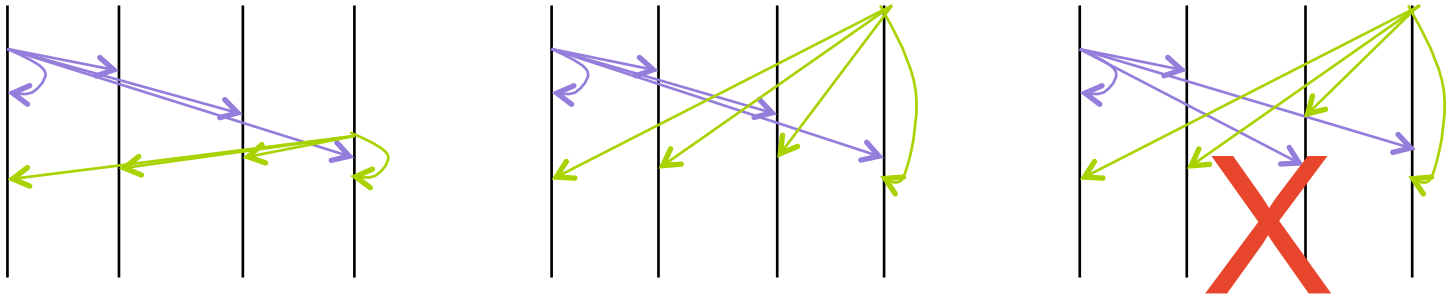
- implementação
  - uso de uma fila de *holdback* em camada de comunicação



PUC RIO

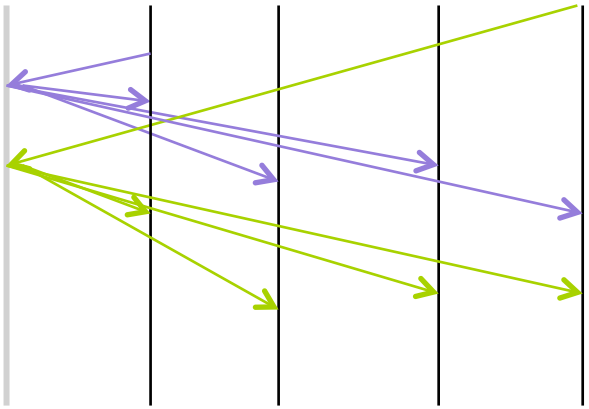
ordenação total

- todos os pontos devem processar as mensagens em uma mesma ordem
  - independentemente da real ordem de envio



uma implementação de ordem total

- utilizando um sequenciador para multicast
  - problema se reduz a ordenação ponto a ponto
  - gargalo e ponto único de falha



PUC RIO

outra implementação de ordem total

- processo de origem envia (msg, id_único) a grupo de processos
- cada processo destinatário responde com uma proposta de número de ordem
  - e deixa (msg, id_único, num_ordem_proposto) em fila de *holdback,* ordenada por num_ordem
- processo emissor coleta respostas, escolhe o maior número de ordem e envia (msg, id_único, num_ordem_final) a destinatários
- processo destinatário troca id_proposto por id_final

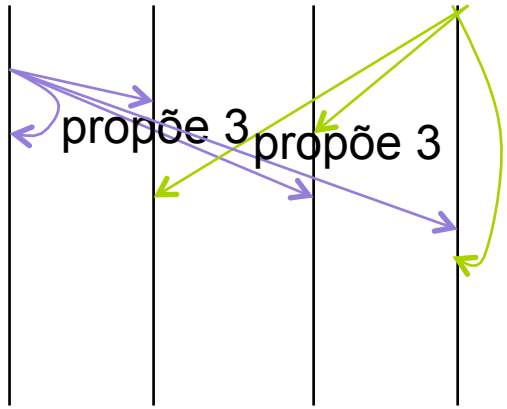
ordem total (cont)

- se msgs no início da fila de holdback estão com id_final, passa-as para a fila de entrega

43521		10	proposto
72170		13	final
63098		15	
39188		16	
42901		19	

- problema possível: diferentes processos proporem o mesmo número de ordem para msgs diferentes

ordem total (cont)

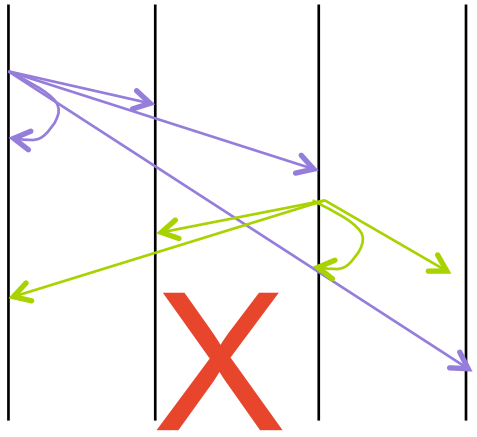


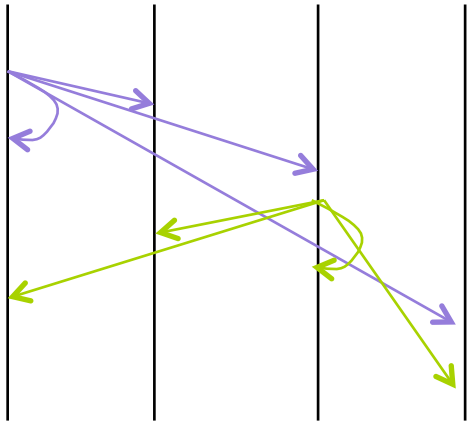
- números propostos devem ser únicos…
  - por exemplo, max(conhecido) + 1 + I/N

PUC RIO

ordenação causal

- aplicações só devem ser processar uma mensagem depois de ter visto mensagens "precedentes causalmente"
  - independentemente da real relação de causa e efeito
  - não implica ordem total





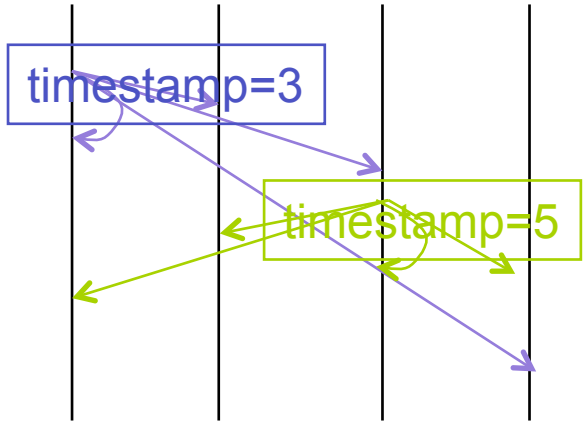
PUC RIO

ordenação causal

- ligada ao conceito de tempo lógico
  - Lamport
  - diante da dificuldade de usar relógios físicos para arbitrar precedência, usa-se o conceito de "pode ter precedido causalmente"
- útil em uma série de algoritmos de coordenação distribuída

relógio lógico e multicast causal

- pode ser usado para ordenar possíveis precedências causais



- como saber se existe alguma msg anterior?
  - expiração de timer
  - ou...? rede sem tempo máximo de entrega

PUC RIO

relógios vetoriais

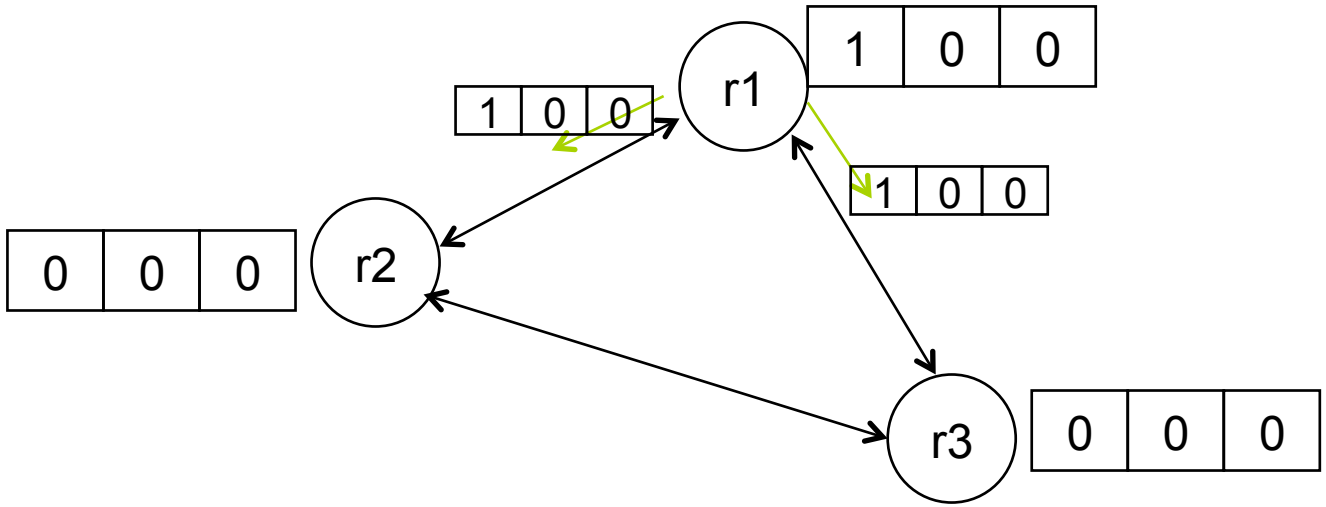
- cada processo mantém um vetor de contadores indicando o último relógio lógico recebido de cada processo
  - número de msgs enviadas por cada processo

3	4	4

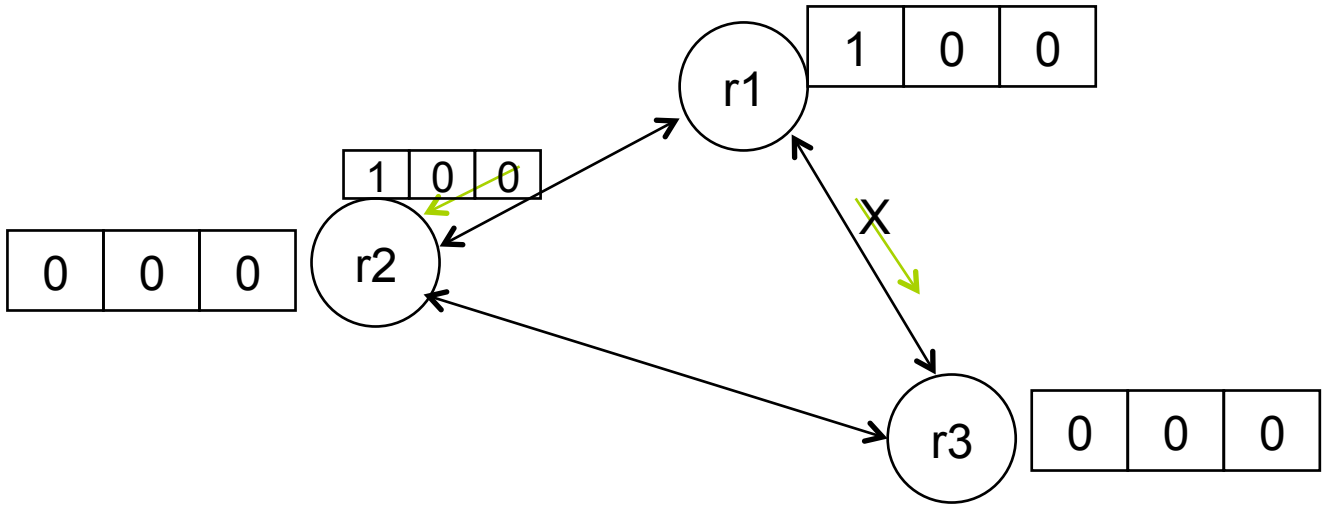
- antes de enviar uma msg, um processo incrementa a posição correspondente a ele mesmo em seu relógio

- msgs recebidas só podem ter diferença (de 1) no contador do processo origem em relação ao receptor

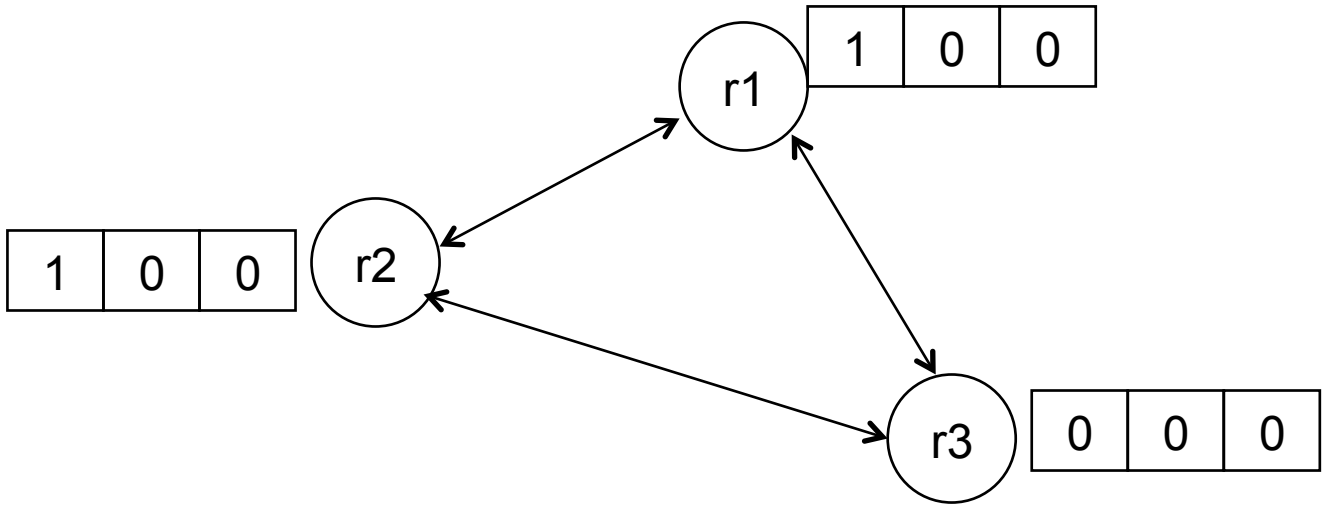
exemplo ordenação causal



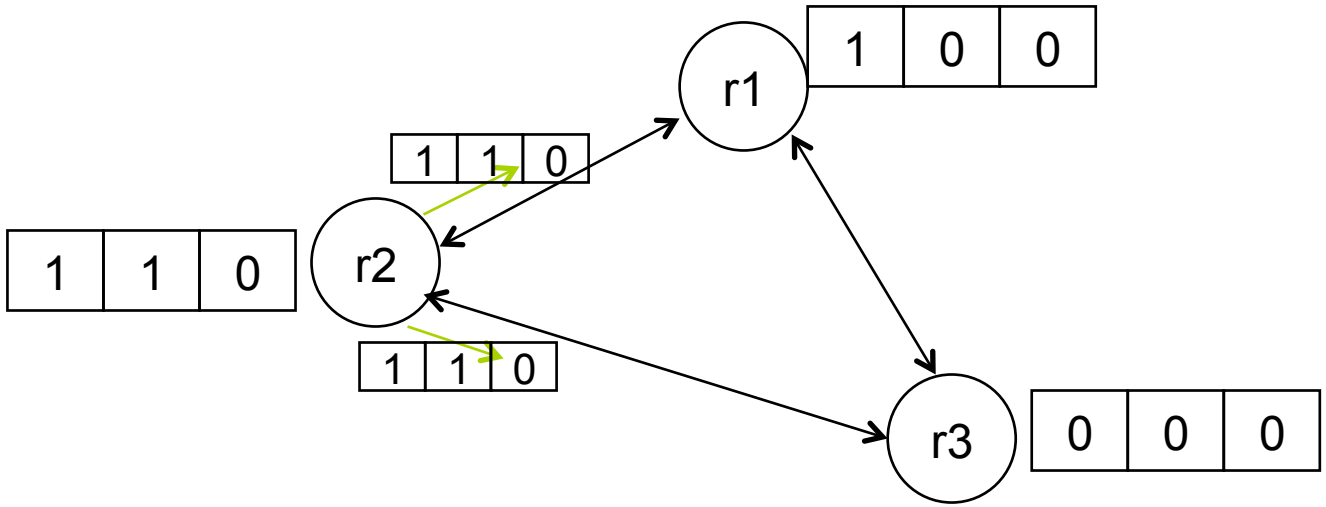
exemplo ordenação causal



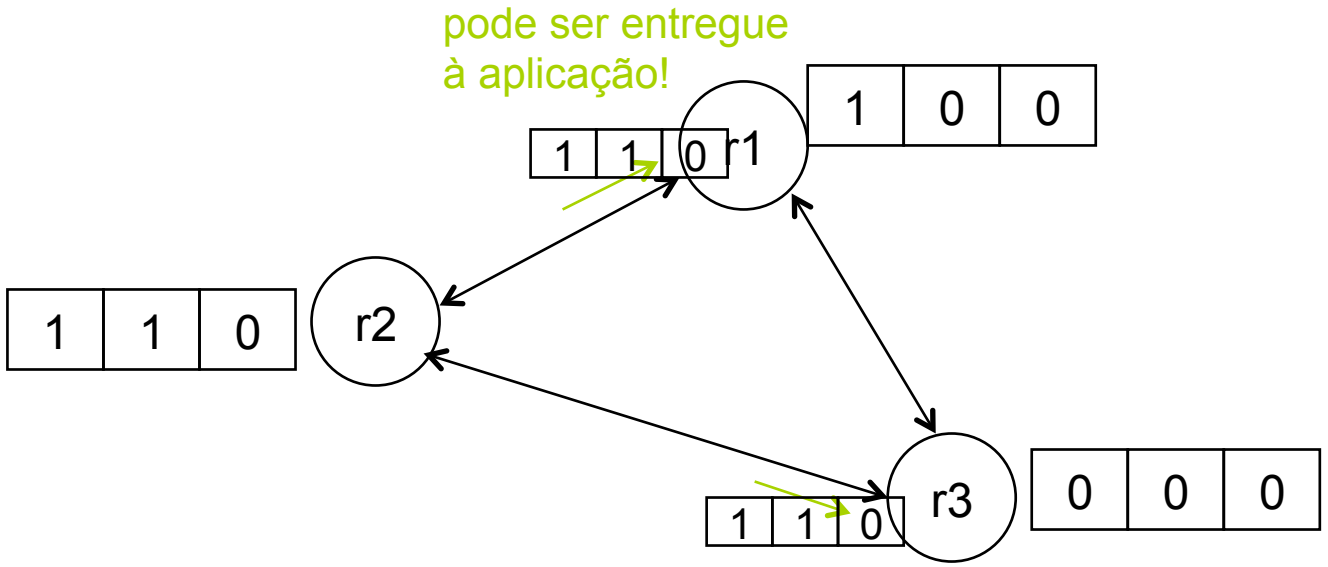
exemplo ordenação causal



exemplo ordenação causal



exemplo ordenação causal



problemas de gerência de grupos

- grupos mtas vezes criados para tolerância a falhas
  - mas e se um dos processos do grupo falhar?
  - como ficam os protocolos de ordenação
- serviços de gerência de grupos
- escalabilidade