Sistemas Distribuídos
Aula 20

Aula passada

- Sistema transacional
- ACID
- Exemplos
- *2-Phase Locking*
- *SS2PL*

Aula de hoje

- Estado distribuído
- *2-Phase Commit*
- Deadlocks
- *3-Phase Commit*

Duas Fases da Transação

- **Ideia:** dividir transação em fases, facilita execução
- **Fase 1: Preparação**
  - adquirir locks de leitura e determinar tudo que é necessário para alterar estado global
  - gerar lista com mudanças no estado global
  - adquirir locks de escrita
- **Fase 2: Commit ou Abort**
  - atualizar estado global, se tudo correu bem
  - abortar a transação sem modificar estado global, se algum imprevisto ocorreu
  - liberar todos os locks

Exemplo

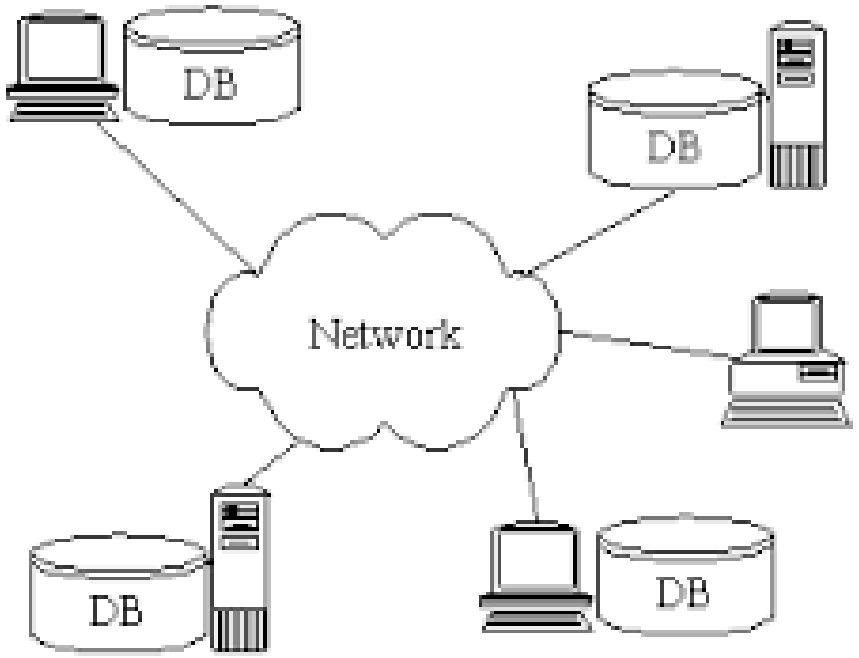
- SS2PL implementado dividindo transação em duas fases
- Melhor desempenho, facilita gerenciamento

```
transferencia(c1, c2, v) {
   L={c1,c2}  // locks
   U={}       // updates
   acquire(L)
   s1 = get_saldo(c1)
   s2 = get_saldo(c2)
   se (s1 - v >= 0)
      s1 = s1 – v
      s2 = s2 + v
      U ={“put_saldo(c1,s1)”,
          “put_saldo(c2,s2)”}
      commit(U,L)
      retorna 0
   abort(L)
   retorna -1
}
```

- **acquire(L)**: obtém todos os locks em L em alguma ordenação global
- **commit(U,L)**: realiza todas as operações em U e libera todos os locks em L
- **abort(L):** libera todos os locks em L

Estado Global Distribuído

- *Two-Phase Locking*: mecanismo de controle de concorrência
  - assume que estado global está “centralizado”
- Considere estado global distribuído



- Estado do sistema é a *união* dos estados locais
- Transações leitura/escrita em diferentes locais
- Muitos sistemas trasacionais são assim
- 2PL não serve!

Transações Distribuídas

- **Ideia**: oferecer ACID ao sistema transacional distribuído
- **Problema**: transação muda estado em diferentes locais, de forma condicionada
- Atomicidade: tudo ou nada, I: isolamento, ...

- Como garantir ACID?

Protocolo para coordenar transação!

2-Phase Commit

- 2PC: protocolo para commit atômico distribuído
  - não confundir com 2PL (*nada a ver*)
- Processo que inicia transação age como coordenador
  - outros são participantes (outros = processos que possuem valores lidos/escritos pela transação)
- Duas fases
  - Preparar e votar: processos localmente decidem se podem efetuar a transação
  - Executar (*commit*): processos mudam estado local

Preparar e Responder

- Coordenador envia diferentes partes da transação (subtransação) para diferentes processos
- Cada processo se prepara para executar a subtransação
  - adquire todos os locks necessários (2PL aqui)
- Cada processo responde ao coordenador
  - Sim: tudo certo para execução (mantendo os locks)
  - Não: algo está errado, abortar (liberando os locks)

Executar

- Coordenador recebe respostas de todos processos
- Se todas positivas, envia mensagem *commit*
  - caso contrário, envia mensagem abort
- Ao receber Commit: processo efetua transação, libera locks, responde com OK
- Ao receber Abort: processo aborta transação, libera locks, responde com OK

Exemplo

- Transferência bancária
- Transferência executada por processo em L1, conta de origem em L2, conta de destino em L3
  - L1 age como coordenador, envia subtransação retirada para L2, envia subtransação depósito para L3

Em L2

```
retirada(c, v) {
  acquire(c)
  s = get_saldo(c)
  se (s >= v)
    s = s-v
    U = ”put_saldo(c,s)”
    votar commit
    se (commit=OK)...
  caso contrario
    votar abort
    release(c)
}
```

Em L3

```
deposito(c, v) {
  acquire(c)
  s = get_saldo(c)
  s = s + v
  U = “put_saldo(c, s)”
  votar commit
  se (commit=OK)...
}
```

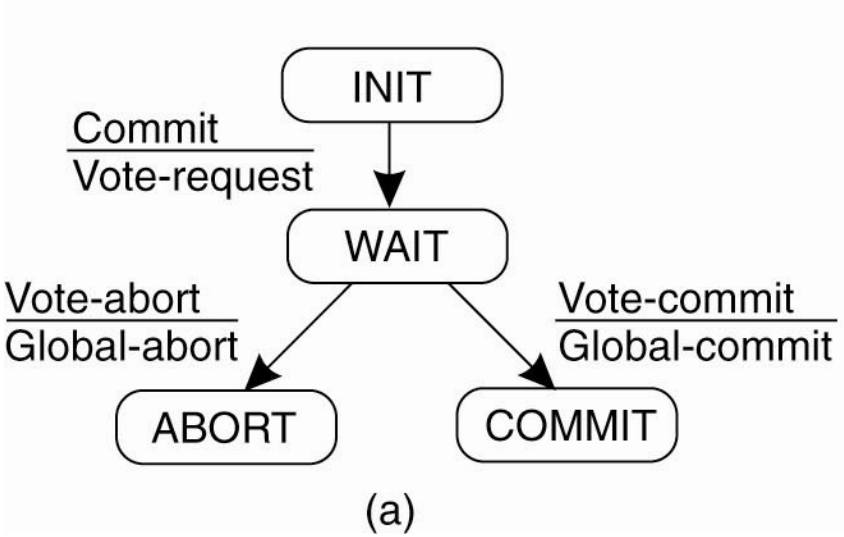
Exemplo

- L2 responde a L1 se pode ou não efetuar a retirada
- L3 diz que pode efetuar depósito, aguarda notificação
- L1 envia commit, caso L2 envie OK
  - Caso contrario, L1 envia abort
- L2 e L3 usam 2PL para transação local, aguardam resposta do coordenador para dar commit
  - liberar locks ao final

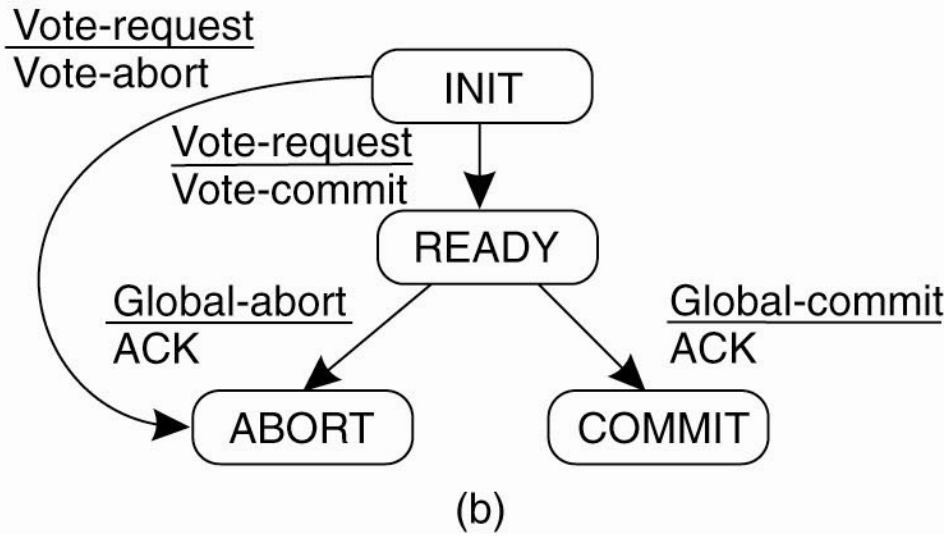
- Como lidar com falhas?

Falhas e 2PC

- Usar temporizadores (*timers*) nas esperas
  - permite sair do estado bloqueado
- Usar armazenamento permanente (*log*)
  - permite voltar ao último estado local
- Diagrama de transição de estados do 2PC
  - cada transição: mensagem(evento)/resposta(ação)



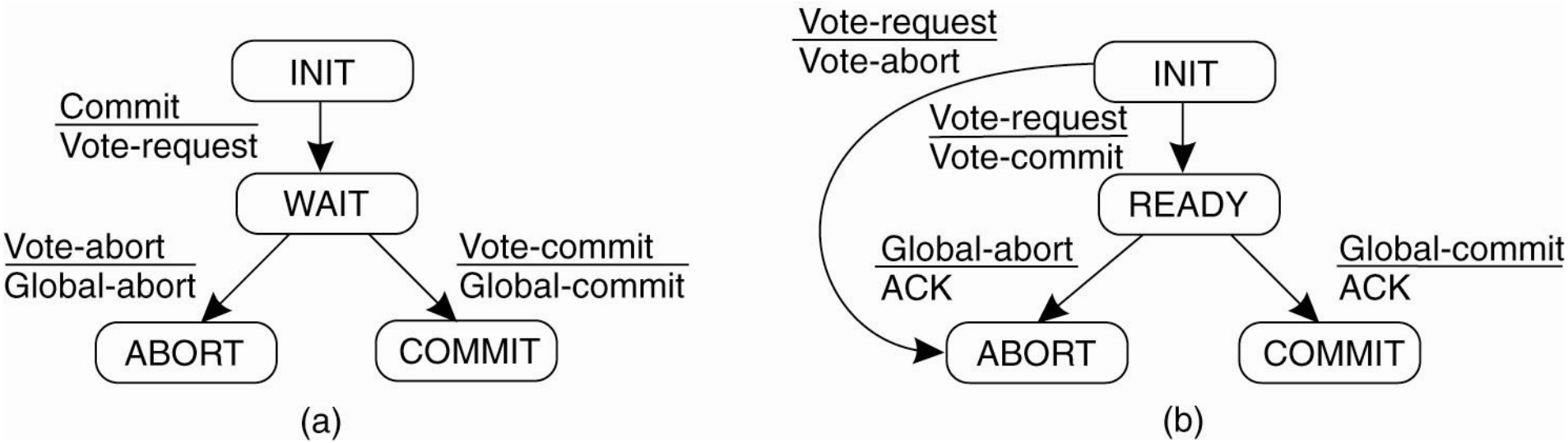
coordenador



participante

Falhas e 2PC

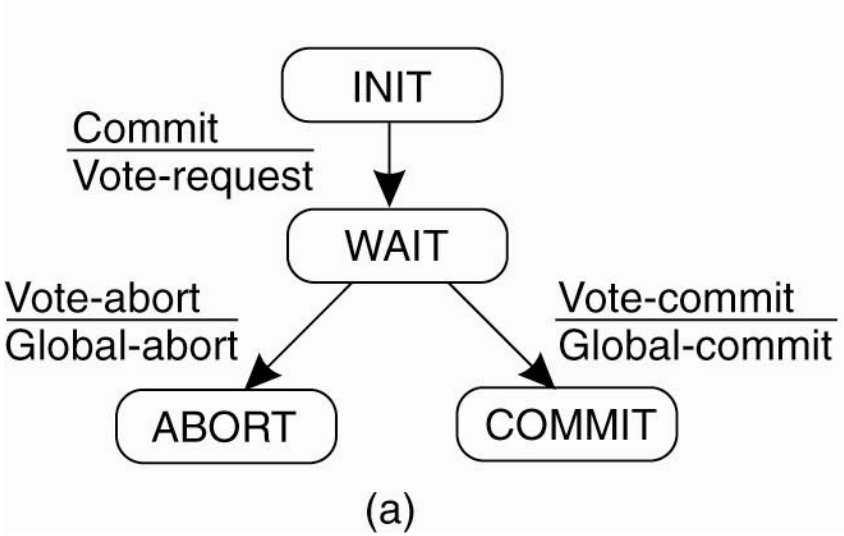
- O que ocorre se processo P falha em INIT?
  - coordenador não recebe resposta, envia abort
- O que ocorre se processo P falha em READY?
  - ao recuperar, P descobre se foi abort ou commit, e faz o mesmo
- O que ocorre se C falha em INIT?
  - nada, processos P não recebem transação



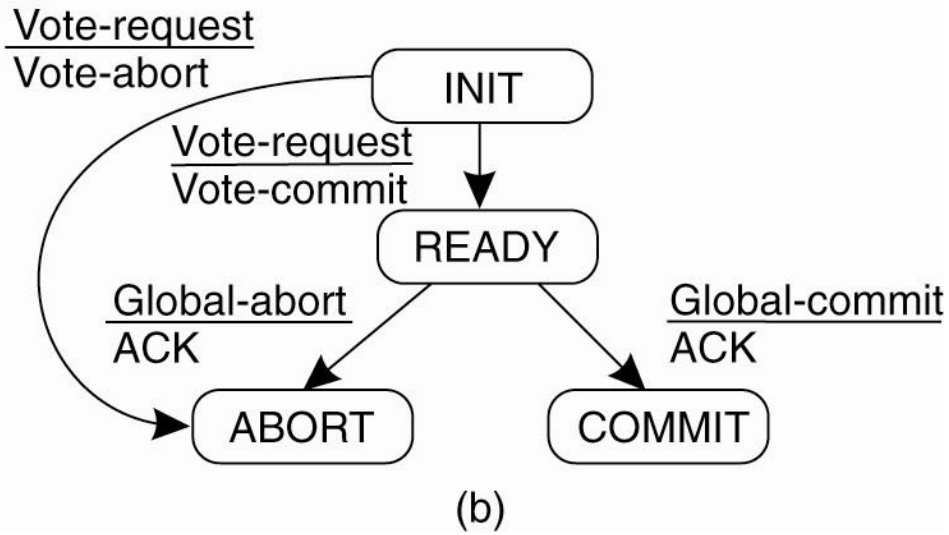
coordenador **participante**

Falhas e 2PC

- O que ocorre se C falha em WAIT?
  - Processo em READY contacta outro processo Q, repete ação de Q se em COMMIT ou ABORT
  - Se Q em READY? Se todos os participantes em READY?
- Não podem tomar uma decisão, dependem de C
  - aguardam até C recuperar
- 2PC é bloqueante em C



coordenador



participante

Outros Problemas

- Muitas coisas podem dar erradas, mesmo sem falhas
- Principal problema?

Deadlocks!

- Execução distribuída das transações pode causar uma dependência cíclica nos locks
  - ex. transf(c1,c2,v1), transf(c2,c1,v2)
- Processos não podem votar (aguardam locks)
  - coordenador(es) aguardam indefinidamente
- Solução: *timeouts*
  - Processos aguardam um tempo, depois abortam (esperando locks, ou esperando mensagens)

Resolvendo *Deadlocks*

- Após abortar, coordenador aguarda um tempo e tenta transação novamente
  - evita deadlocks
- Problema?

Livelocks!

- Coordenador pode tentar repetidas vezes, sem nunca ter sucesso
  - ex. transação pode de fato ser impossível
- Solução: tentar número máximo de vezes
  - aborta transações, mas evita *livelocks*

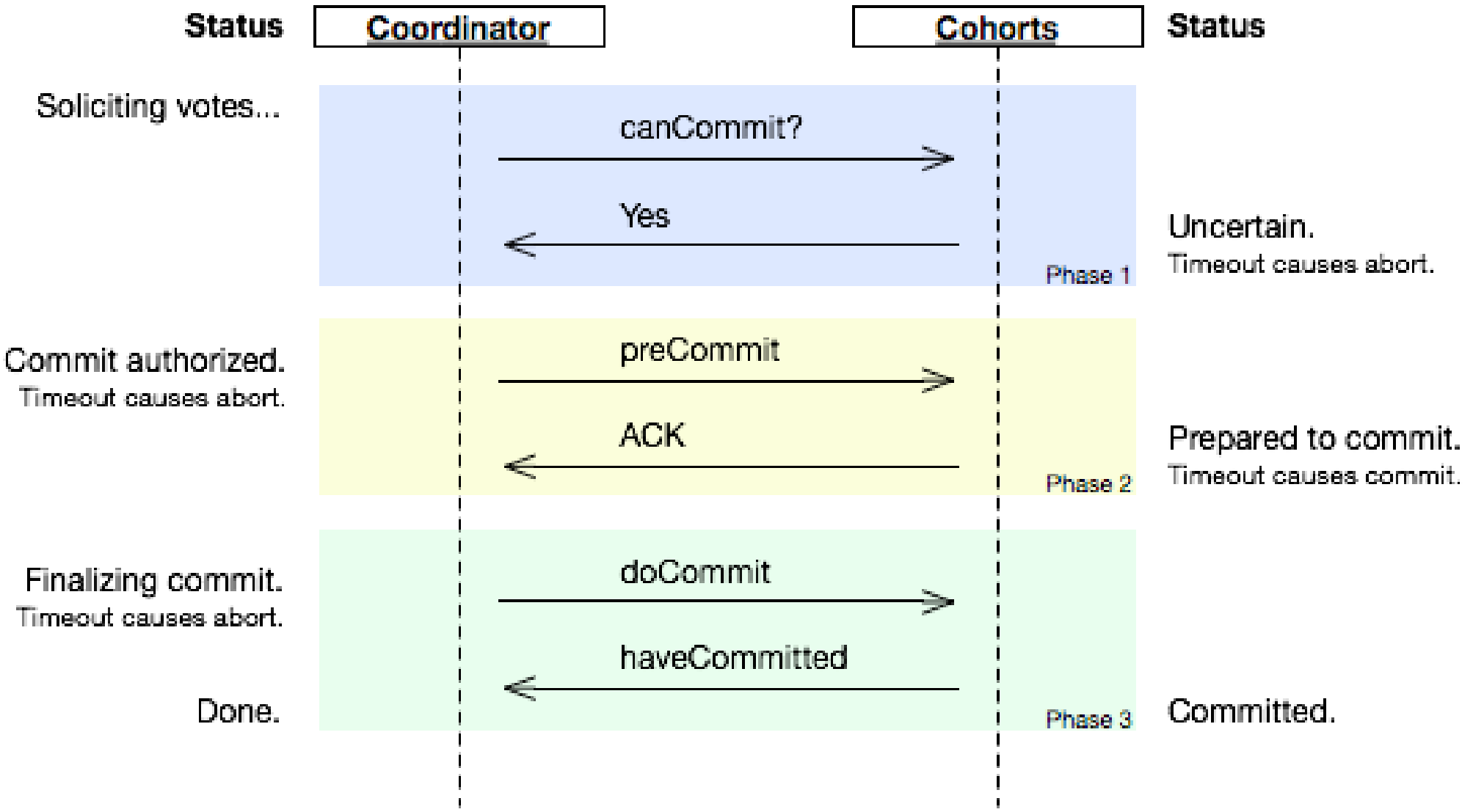
Three-Phase Commit

- Principal desvantagem do 2PC?

Depende do coordenador

- *Three-Phase Commit* (3PC)
  - muito similar ao 2PC
  - fase extra permite commit, sem que coordenador confirme commit final
    - não depende do coordenador
  - Processos podem chegar a decisão de forma distribuída

Three-Phase Commit



Vantagens e Desvantagens

- Não bloqueante no coordenador
  - permite concluir transação (mesmo que sem consenso em alguns casos)
- Timers resolvem naturalmente possíveis deadlocks
- Permite recuperar alguns tipos de falhas que P2C não permite
- Demandam três RTT (*Round-Trip Time*) entre coordenador e participantes
  - e seis mensagens por participante