Computação Distribuída

Cap IX – Transações

Licenciatura em Engenharia Informática

Universidade Lusófona

Prof. Paulo Guedes (paulo.guedes@ulusofona.pt)

Transacções atómicas – ilustração do problema

- Um cliente é possuidor de duas contas A e B e pretende fazer uma transferência de 100.000 Eur da conta A para a conta B. O procedimento para fazer esta transferência pode ser o seguinte:

```
transferencia (contaA, contaB, Valor)
{
    Levantar (contaA, Valor);
    Depositar(contaB, Valor);
}
```

Que acontece se ocorrer uma falha neste momento, depois de levantar mas antes de depositar ?

Transacções Atómicas

```
transferencia (contaA, contaB, Valor)
{
    begin_transaction;
    Levantar(contaA, Valor);
    Depositar(contaB, Valor);
    commit_transaction;
}
```

Uma transacção atómica é um conjunto de operações cujo efeito é o de se executarem **todas**, ou **nenhuma**

Transacção Atómica

- A transacção pode ser interrompida por:
  - Erro de sistema
  - Iniciativa do programa, por verificar a existência de um erro lógico
- Em ambos os casos, o efeito é abortar a transacção

```
transferência (contaA, contaB, Valor)
{
   begin_transaction;
              SaldoA = LerSaldo (contaA);
              if (Valor > SaldoA)
                     abort_transaction;
              else
              {
                     Levantar(contaA, Valor);
                     Depositar(contaB, Valor);
                     commit_transaction;
              }
}
```

Propriedades das transacções - ACID

- Atomicidade
  - Para um observador externo, uma transacção ou se executa na totalidade ou não se executa.
  - Sempre que existam faltas, é possível recuperar o sistema para o estado inicial da transacção.
  - O desfazer das operações (*rollback*) implica manter o estado parcial da transacção numa forma que facilite a sua confirmação ou anulação;
  - Implementação
    - As operações quando se executam modificam um estado volátil (transaction log)
    - Apenas quando é decidido que a transacção se efectivou, os resultados são tornados persistentes (escrita de registo “commit” no transaction log)
    - Para suportar esta funcionalidade é necessário que o sistema seja capaz de repor a situação inicial no caso da transacção abortar.
- Consistência
  - Cada transacção deve, a partir de um estado inicial válido e caso se execute completamente, atingir um novo estado válido
  - Os invariantes associados à estrutura de dados devem permanecer válidos no início e no fim da transacção;

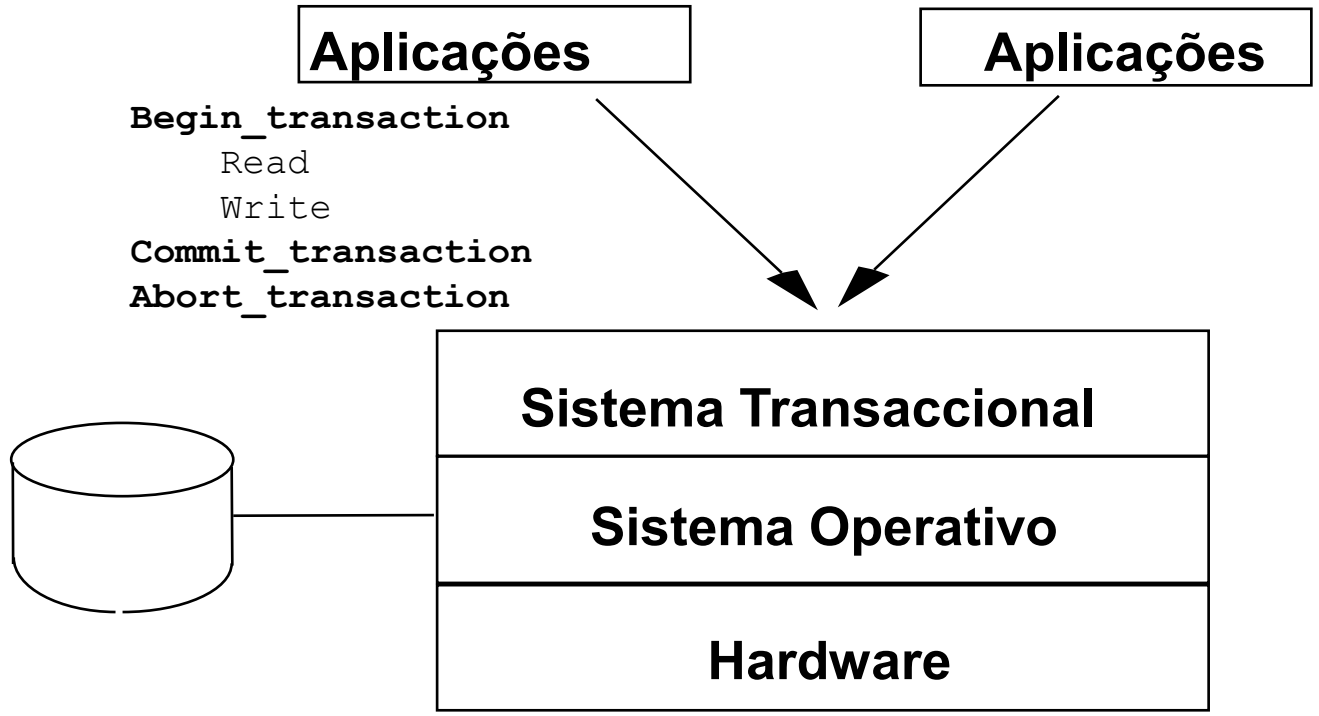
Propriedades das transacções - ACID

- Seriabilidade (*Isolation*)
  - Se diversas transacções se executarem em paralelo sobre os mesmos objectos, tudo se passa como se as transacções se executassem em série numa determinada ordem.
  - Os seus efeitos são sequenciais
  - Implementação
    - Cada transacção tem de usar mecanismos de sincronização que garantam que a ordem de execução é correcta.
    - O mecanismo habitual é designado por sincronização em duas fases (two-phase-locking).

- Persistência (*Durability*)
  - Os resultados de uma transacção que confirmou permanecem depois de esta acabar e são supostos sobreviver ao conjunto de faltas expectáveis dos mecanismos de armazenamento.
  - Implementação:
    - Resultados têm que ser escritos para disco ou outra memódia durável

ACID - Atomic, Consistent, Isolated, Durable

Sistema Transaccional



Sistema Transaccional

- `Begin_transaction`
  - A operação consiste na atribuição de um identificador à transacção, preparação das estruturas de dados de suporte e registo no diário do início da transacção.
- `Read`
  - Lê dados da base de dados
  - Previamente, tem que obter um “read lock” sobre os dados
- `Write`
  - Escreve dados na base de dados
  - Previamente, tem que obter um “write lock” sobre os dados
- `Commit_transaction`
  - O resultado das operações `Write` é tornado persistente
  - Libertam-se os locks sobre os dados
- `Abort_transaction`
  - “Apagam-se” as modificações à base de dados
  - Libertam-se os locks sobre os dados

Sincronização

- O sistema transaccional tem que implementar sincronização entre as operações

Operações	Read	Write
Read	Compatível	Incompatível
Write	Incompatível	Incompatível

- Existem diversas formas de implementar a sincronização
  - História de operações: sequência das operações `Read` e `Write`
  - A sincronização tem que garantir que as histórias são serializadas
  - Uma história é serializada se para qualquer duas transacções Ti e Tj todas as operações de Ti aparecerem antes de Tj ou vice-versa

Implementação dos Mecanismos de Sincronização

- Modelo pessimista
  - Pressupõe que os conflitos são frequentes e obriga à prévia sincronização de todos os acessos.

```
Read (t, obj1) {
   lock_read (t, obj);
   read_data (obj);
}

Write(t, obj2) {
   lock_write (t, obj);
   write_data (obj);
}

Commit_transaction(t) {
   write_data (t, "Commit");
   unlock_all (t);
}
```

- Modelo optimista
  - Considera que os conflitos são raros e que, portanto, as transacções se podem executar mais rapidamente sem sincronização
  - Se no final detectar conflitos, aborta uma das transacções

Sincronização com Locks

- Sincronização em duas fases estrita (*strict two phase locking*)
  - Na primeira fase a transacção começa por adquirir sucessivamente todos os trincos que lhe permitem aceder aos objectos.
  - Na segunda fase liberta-os.
- Interblocagem (deadlock)
  - Este tipo de sincronização pode conduzir a deadlock obrigando a mecanismos para a resolver:
    - Prevenção (ex: ordenação das operações)
    - Detecção (ex: timeouts)

Perguntas a que devo ser capaz de responder

- Qual é o problema que as transações procuram resolver ?
- Quais são as falhas que podem ser resolvidas com transações ?
- O que significa cada uma das propriedades ACID das transações ?
- Porque é que as transações têm que ter um mecanismo de sincronização associado ?
- Qual é o papel do diário (“Log”) nas transações ?

Transacções num Contexto Distribuído

- Uma transacção distribuída pode ser visualizada como um processo que tem um determinado estado e que pode propagar-se a múltiplas máquinas. Pode também lançar subprocessos para optimizar a sua execução.

- Operações
  - Manipular dados guardados localmente
  - Manipular dados guardados remotamente
  - Iniciar uma operação executada remotamente

Transacções distribuídas – ilustração do problema

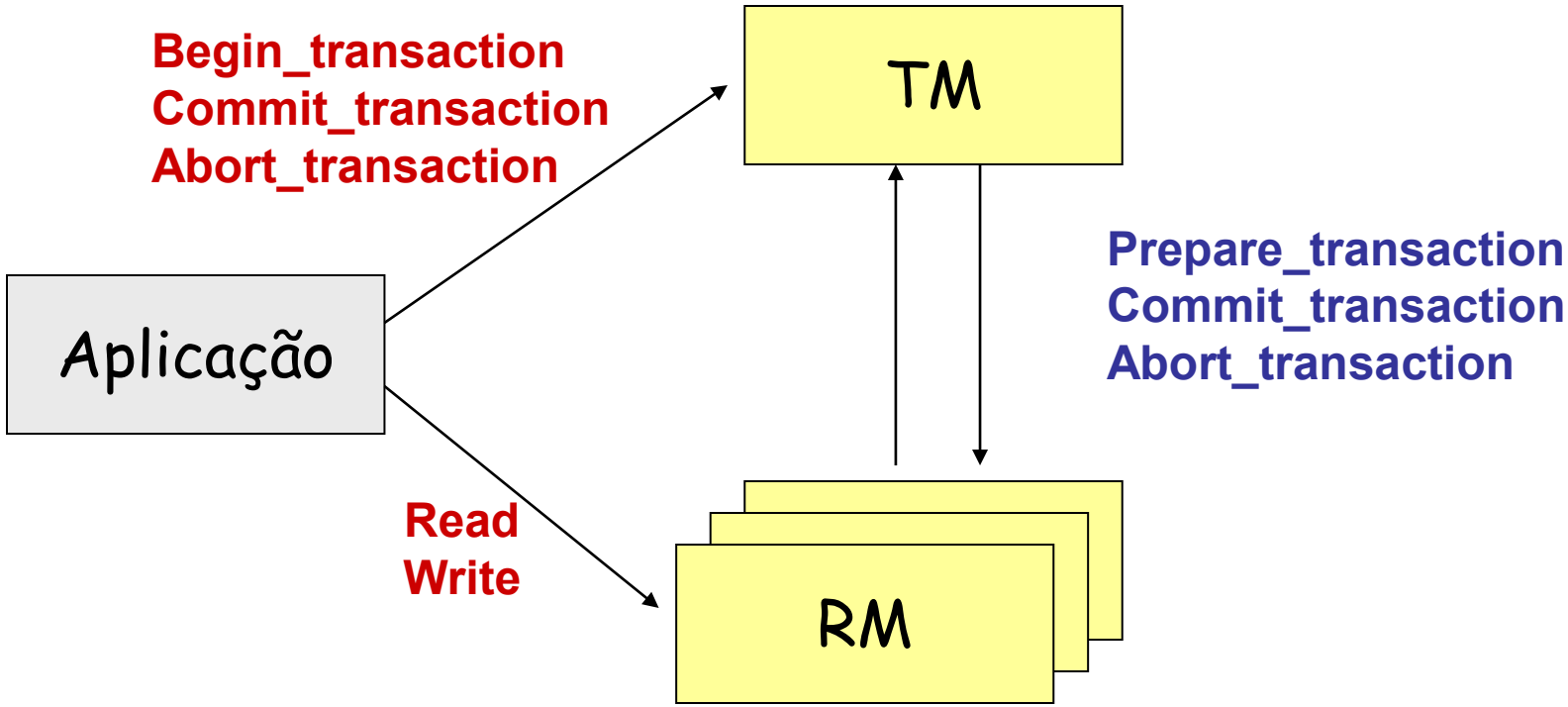
```
transferencia (contaA, contaB, Valor)
{
   Levantar (contaA, Valor);
   Depositar(contaB, Valor);
}
```

- Se a contaA e a contaB estiverem no mesmo banco e na mesma aplicação, a transacção pode ser implementada na aplicação, ou na sua base de dados
- Mas se a contaA e a contaB estiverem em bancos diferentes, como é que se lida com um erro na operação Depositar, uma vez que o levantamento já foi feito ?

Transacções distribuídas

- Uma transacção distribuída lida com operações que envolvam vários componentes que possam falhar independentemente
- A transacção distribuída obriga a duas coisas:
  - Coordenação entre componentes diferentes e independentes
  - Eventual alteração do comportamento desses componentes, para lidarem com as transacções distribuídas
- Arquitectura de referência considera dois tipos de componentes
  - Monitor Transaccional - TM (Transaction Manager)
    - Coordena os vários participantes na transacção
    - Executa os protocolos de iniciação e terminação das transacções
  - Gestores de Recursos - RM (Resource managers)
    - Bases de dados “convencionais”
    - Armazenam os dados e suportam as operações de leitura e escrita
    - Recebem indicações do TM para iniciarem e terminarem as transacções

Transacções distribuídas: modelo de referência

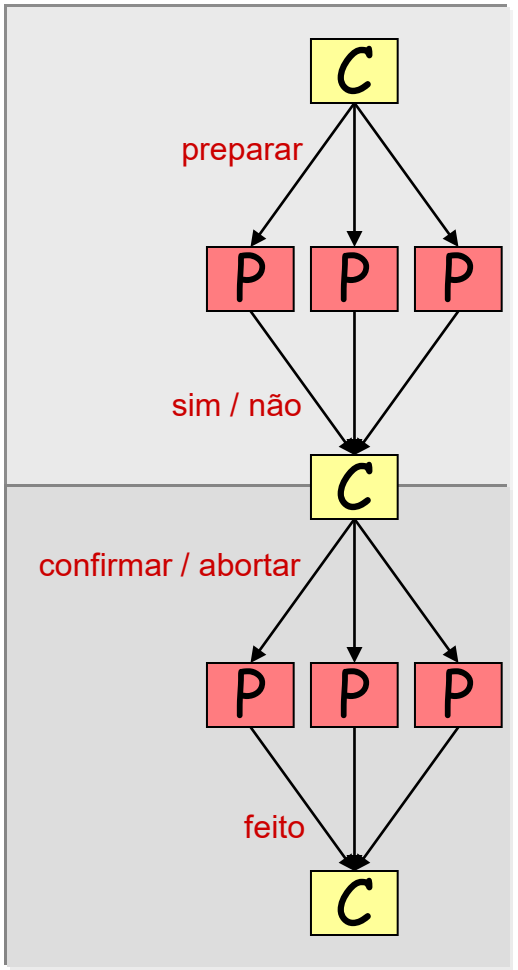


Perguntas a que devo ser capaz de responder

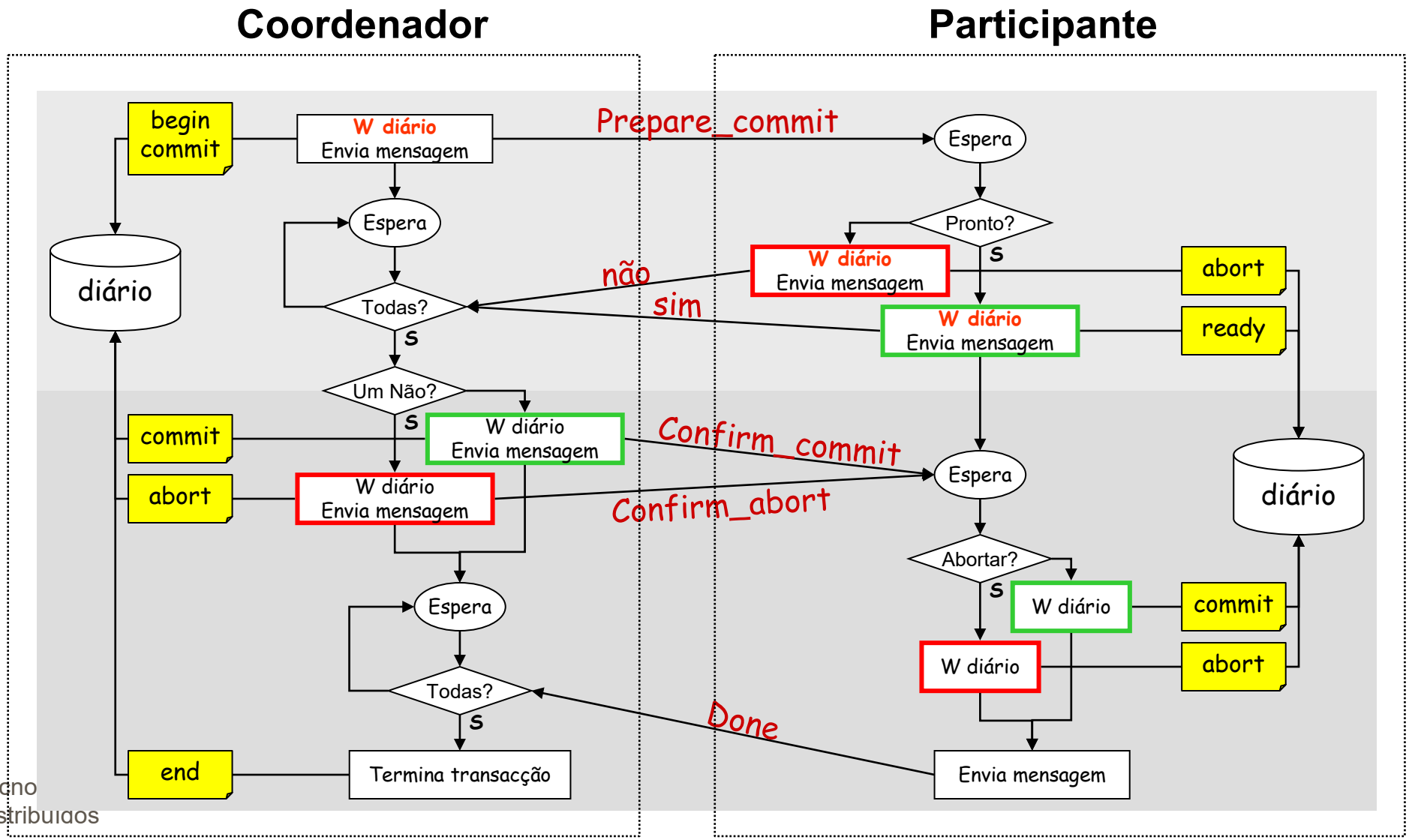
- Quais são os problemas adicionais que se colocam em transações distribuídas ?
- Qual é a função de cada um componentes de um sistema transacional distribuído: Aplicação, Transaction Monitor, Resource Monitor ?

Transacções distribuídas: Two-Phase Commit (2PC)

- Protocolo de coordenação entre vários participantes numa transacção distribuída
  - Coordenador: TM
  - Participantes: vários RM
  - Modelo de falhas: falha por paragem de um ou mais participantes
  - Protocolo de consenso distribuído
- Fase 1
  - O Coordenador prepara todos os participantes para confirmarem a transacção
- Fase 2
  - Todos confirmam (ou não) a transacção
- Regra de confirmação global
  - O Coordenador aborta a transacção se um dos Participantes votar contra
  - O Coordenador confirma a transacção se, e só se, todos os Participantes votarem a favor



Transacções distribuídas: 2PC - diagramas de interacções



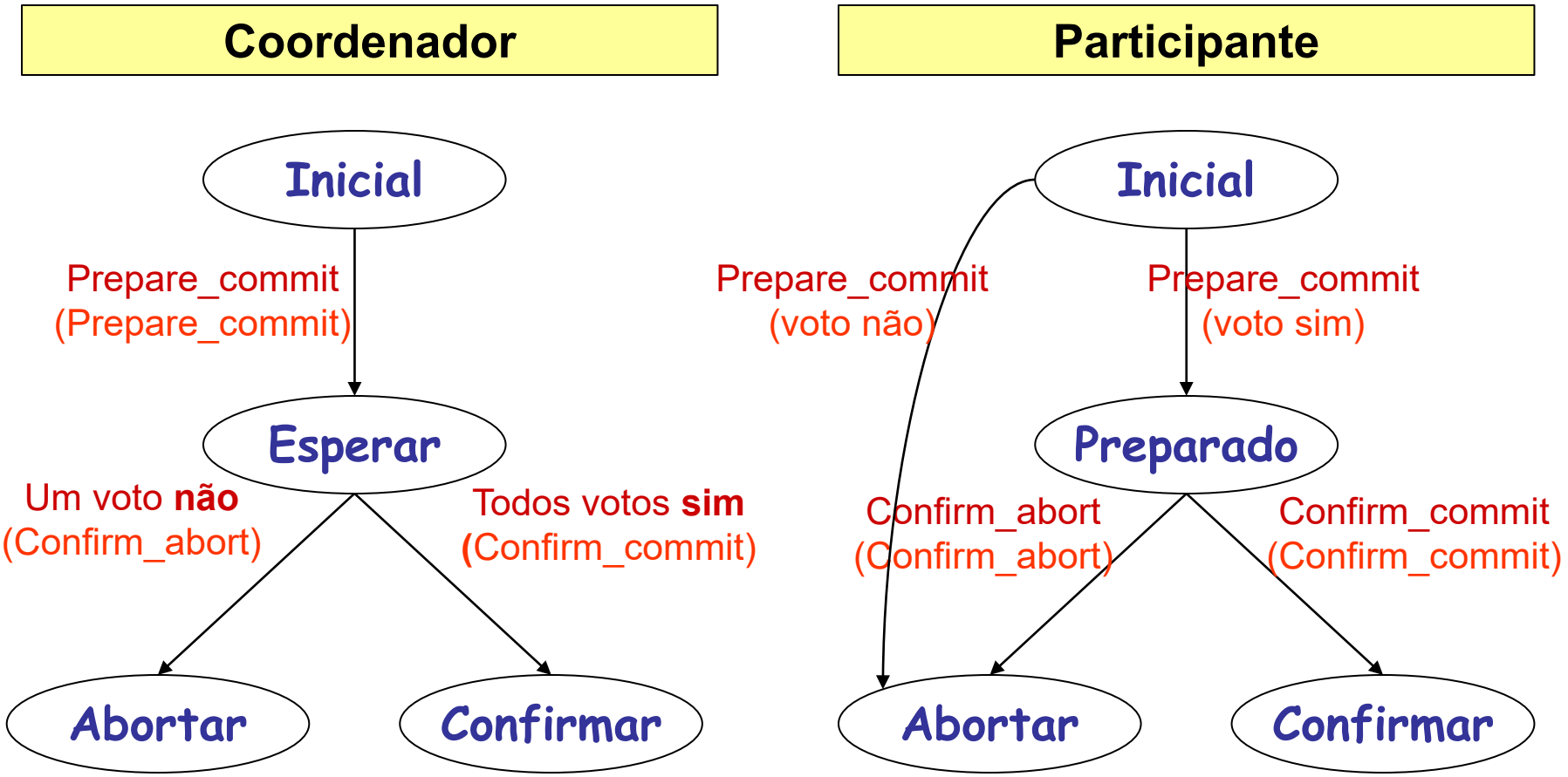
Transacções distribuídas: 2PC (sem faltas) - 1ª Fase

- O Coordenador inicia o protocolo
  - Envia a todos os RM participantes a mensagem `Prepare_commit`
- Os RM participantes reagem ao pedido de commit do TM
  - Registam a indicação de `Prepare_commit`
  - Verificam se podem confirmar a transacção
  - Respondem ao Coordenador com “Sim” ou “Não”
- O TM Coordenador regista as respostas dos RMs participantes
  - Espera pela decisão de todos os RM Participantes
  - Escreve um registo no diário com as respostas dos RM participantes

Transacções distribuídas: 2PC (sem faltas) - 2ª Fase

- Se uma das respostas é negativa, o TM Coordenador decide abortar a transacção
  - Escreve o registo “Confirm_abort” no seu diário
  - Envia a mensagem “Confirm_abort” aos RM Participantes
- Se todas as respostas são positivas, o TM Coordenador decide confirmar a transacção
  - Escreve o registo de “Confirm_commit” no seu diário
  - Envia a mensagem “Confirm_commit” aos RM participantes
- RM participantes
  - Escrevem o registo “Confirm_abort” ou “Confirm_commit” nos seus diários
  - Confirmam ou abortam localmente a transacção
  - Respondem ao Coordenador que terminaram localmente a transacção
- Coordenador
  - Espera que todos os RM confirmem a terminação local da transacção
  - Quando tal acontecer termina a transacção
    - Escreve o registo de terminação “End_commit” ou “End_abort” no seu diário

Transacções distribuídas: 2PC - diagramas de estados



Transacções distribuídas: Tolerância a faltas no 2PC

- Não recepção de mensagens
  - Detectadas com um temporizador no Coordenador ou nos Participantes
- *Timeout* no Coordenador
  - Estado Esperar
    - Não pode confirmar unilateralmente a transacção
    - Mas pode unilateralmente optar por abortar a transacção
      - Se considerar que o atraso na resposta se deve a uma falha
  - Estados Abortar e Confirmar
    - O Coordenador não pode terminar a transacção
      - Tem que receber a confirmação de todos os Participantes
    - Pode repetir a mensagem global previamente enviada

Transacções distribuídas: Tolerância a faltas no 2PC

- *Timeout* num Participante
  - Estado Inicial
    - Opta unilateralmente por abortar a transacção
  - Estado Preparado
    - Não pode progredir
      - Depende da decisão do Coordenador que já influenciou
      - A transacção fica activa e bloqueada até se saber essa decisão
    - Se os Participantes interactuarem é possível evoluir
      - Obtendo a decisão do coordenador que chegou aos outros Participantes
    - Falha permanente do Coordenador (apenas)
      - Protocolos de eleição de um novo Coordenador

Transacções distribuídas: Tolerância a faltas no 2PC

- Paragem de máquinas
  - O que fazer após a sua recuperação
- Recuperação do Coordenador
  - Estados Inicial e Esperar
    - Repete as mensagens de Preparação para obter novamente a votação dos TMs
  - Estado Confirmar ou Abortar
    - Se ainda não recebeu todas as confirmações repete o envio da mensagem global previamente enviada
- Recuperação de um Participante
  - Estado Inicial
    - Aborta unilateralmente a transacção
  - Estado Preparado
    - Reenvia o seu voto (sim ou não) para o Coordenador

Transacções distribuídas: Problemas do 2PC

- O protocolo é bloqueante
  - Obriga os Participantes a esperar pela recuperação do Coordenador
    - E vice-versa
  - O protocolo bloqueia componentes funcionais por causa de outras com falhas
- Não é possível fazer uma recuperação totalmente independente
- Há alternativas não-bloqueantes
  - Normalmente muito mais complexas (ex. 3PC)

Perguntas a que devo ser capaz de responder

- Quando é que é executado o protocolo 2-Phase Commit (2PC) ? Dê um exemplo com uma aplicação de reservas que pretende reservar um voo e um hotel de forma transacional (ou faz ambas as reservas, ou não faz nenhuma delas)
- Como é o funcionamento do protocolo 2-Phase Commit (2PC) se não existirem falhas ?
- O que é que o Coordenador deve fazer se não conseguir receber resposta de um participante à mensagem “Prepare_commit” ? E se for à mensagem “Confirm_commit” ? E se for à mensagem “Confim_abort” ?
- O que é que um participante deve fazer se não receber do coordenador a mensagem “Prepare_commit” ? E se for a mensagem “Confirm_abort” ? E se for a mensagem “Confirm_commit” ?
- O que é que o Coodenador deve fazer se falhar, e ao recuperar verificar que está no estado “Esperar” (ainda não recebeu todos os votos) ? E se verificar que está no estado “Confirmar” ?
- O que é que um Participante deve fazer se falhar, e ao recuperar verificar que está no estado “Preparado”, pois votou “Commit” mas ainda não recebeu a mensagem de confirmação do Coordenador ?