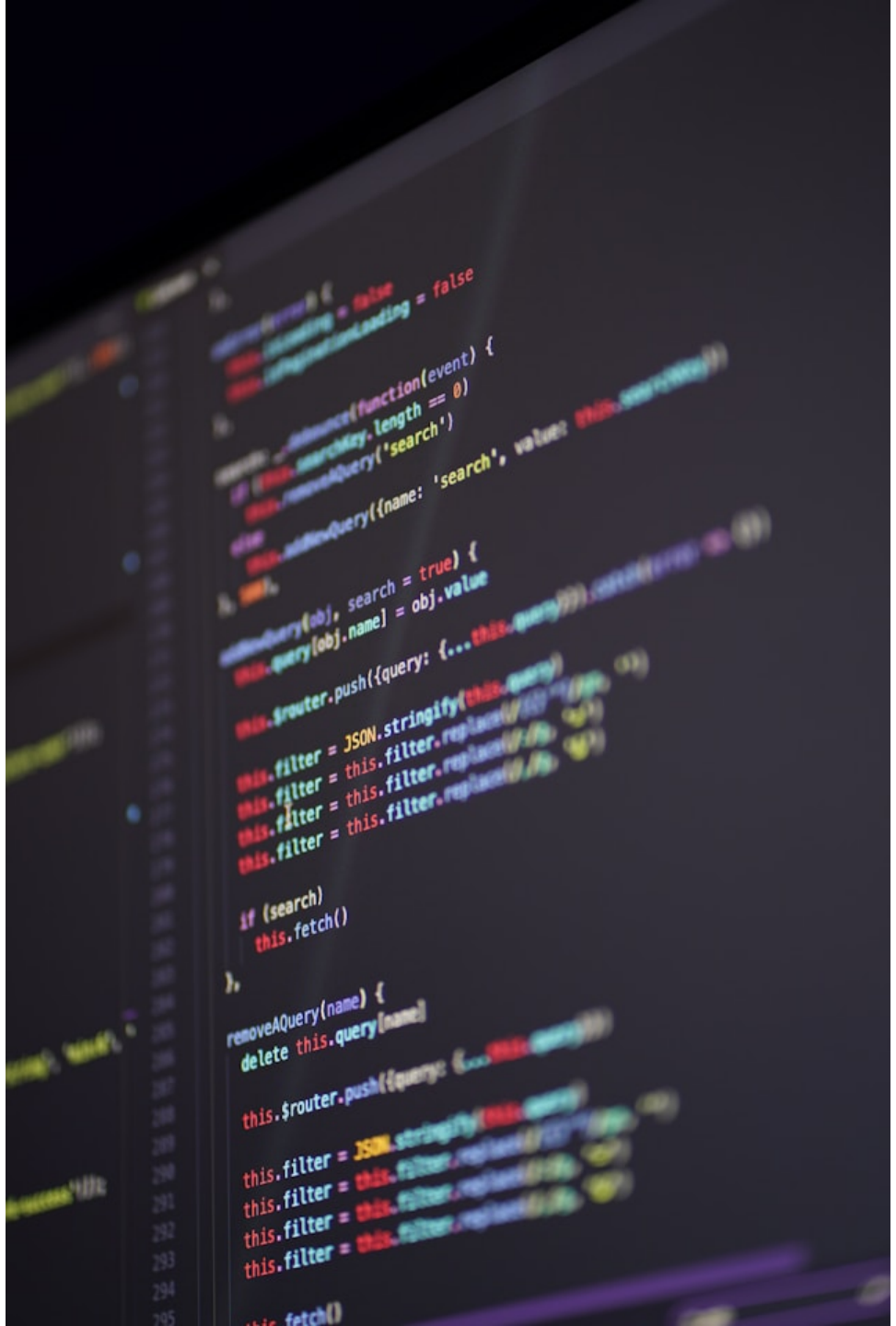


5954025 - Sistemas Distribuídos

Aula 12c - Transações (Parte 3)

Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Complementando a aula passada: 2PL distribuído

Cada nó possui seu próprio gerente de travamento (lock manager).

1. Solicita travas nos nós participantes
2. Executa operações locais
3. Confirma ou aborta globalmente
4. Libera travas

- Desafio: Pode gerar deadlocks globais
  - Nó 1: T1 espera T2
  - Nó 2: T2 espera T1
  - Cada nó vê apenas parte do problema.
  - Solução: Construir um grafo global de espera.

Problema do Controle de Concorrência *Pessimista*

Se considerarmos que os conflitos entre transações são *raros* e que a maioria delas possui *curta duração*, então forçar as transações a adquirirem travas (como no 2PL) adiciona uma sobrecarga desnecessária.

Objetivos de Aprendizagem

- Explicar o funcionamento do **Controle de Concorrência Otimista**, descrevendo suas fases principais: trabalho, validação e atualização.
- Compreender em quais cenários o OCC é mais adequado.
- Diferenciar **validação para trás** e **validação para frente**, identificando quais transações são comparadas em cada abordagem e como conflitos de leitura e escrita são detectados.
- Explicar o papel do **Two-Phase Commit (2PC)** em transações distribuídas, relacionando suas fases de votação e decisão com a garantia de atomicidade.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Controle de Concorrência Otimista (OCC)

O Optimistic Concurrency Control (OCC) usa *timestamps* para validar transações. Funciona melhor quando há poucos conflitos, por exemplo:

- transações majoritariamente de leitura;
- transações que acessam subconjuntos diferentes de dados;
- bases grandes sem grande concentração de acesso nos mesmos registros.
- Funcionamento:
  - A transação prossegue como se não houvesse nenhuma possibilidade de conflito com outras transações, até que a transação termine (pedido de **closeTransaction**).
  - Quando surge um conflito, alguma transação é normalmente cancelada e precisará ser reiniciada.
- **Fases de uma transação**: Fase de trabalho, Fase de validação e Fase de atualização.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Fase de Trabalho

No OCC, cada transação possui um espaço de trabalho privado. Assim, nenhuma outra transação consegue ler modificações ainda não confirmadas.

- Permite que a transação seja cancelada sem nenhum efeito sobre os objetos reais.
- Operações de leitura são realizadas imediatamente sobre a versão tentativa ou efetivada.
- Operações de escrita registram os novos valores de objetos como valores de tentativa, os quais são invisíveis para outras transações.
- São mantidos dois registros dos objetos acessados dentro de uma transação:
  - Conjunto de **leitura**: contém os objetos lidos pela transação.
  - Conjunto de **escrita**: contém os objetos modificados pela transação.

Fase de Validação

Quando pedido de **closeTransaction** é recebido, a **transação é validada** para estabelecer se as operações sobre os objetos entram em conflito ou não com as operações de outras transações sobre os mesmos objetos.

- Se a validação for bem sucedida, então a transação poderá ser efetivada.
- Se a validação falhar, então esta ou outra(s) transação(-ões) em conflito deverá(-ão) ser cancelada(s).

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Fase de Atualização

Se uma transação é **validada**, então todas as alterações registradas em suas versões de tentativa tornam-se permanentes.

- Transações de somente leitura podem ser efetivadas **imediatamente**.
- Transações de escrita estão prontas para serem efetivadas quando as versões de tentativa dos objetos **tiverem sido registradas** no meio de armazenamento permanente.

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Validação de Transações

Cada transação recebe um número ao entrar na fase de validação.

- Número é um valor inteiro atribuído em sequência ascendente
- Transação T_i precede transação T_j se $i < j$
- Teste de validação na transação T_v é baseado no conflito entre operações em pares de transação T_i e T_v

Regras de conflito de operação

Regra	T_v	T_i	Descrição
R1	Escrita	Leitura	T_i não deve ler objetos escritos por T_v
R2	Leitura	Escrita	T_v não deve ler objetos escritos por T_i
R3	Escrita	Escrita	T_v não deve escrever em objetos modificados por T_i e T_i não deve escrever em objetos modificados por T_v

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Validação de Transações (cont.)

Como as fases de **validação** e **atualização** geralmente têm curta duração, uma simplificação pode ser obtida com uma nova regra:

- Apenas **uma** transação pode estar na fase de **validação** e **atualização** em um dado momento.
- Uso de seção crítica para restringir o acesso.
  - Neste caso, quando duas transações não podem se sobrepor na fase de atualização a Regra 3 é satisfeita.
- Muito restritiva se toda a validação e atualização estiverem na seção crítica.

> A validação de uma transação deve garantir que as regras 1 e 2 sejam obedecidas, testando as sobreposições entre os objetos de pares de transações T_v e T_i.

Estratégias de Validação

Na fase de validação, o sistema verifica conflitos de leitura-escrita e escrita-escrita para garantir que os conflitos sigam uma única direção temporal.

Há duas abordagens principais:

- **Validação para trás**: compara a transação atual com transações já confirmadas desde o início de sua execução (foco na Regra 2).
- **Validação para frente**: compara a transação que está tentando confirmar com transações mais novas ainda em execução (foco na Regra 1).

> A validação para trás é a abordagem mais comum.

Validação para Trás

Como todas as operações de leitura das transações sobrepostas anteriores foram executadas antes que a validação de T_v começasse, elas não podem ser afetadas pelas escritas da transação corrente (Regra 1 satisfeita).

- A validação da transação T_v verifica se seu conjunto de leitura se sobrepõe a qualquer um dos conjuntos de escrita das transações sobrepostas anteriores T_i (Regra 2)
- Se houver qualquer sobreposição, a validação falhará.

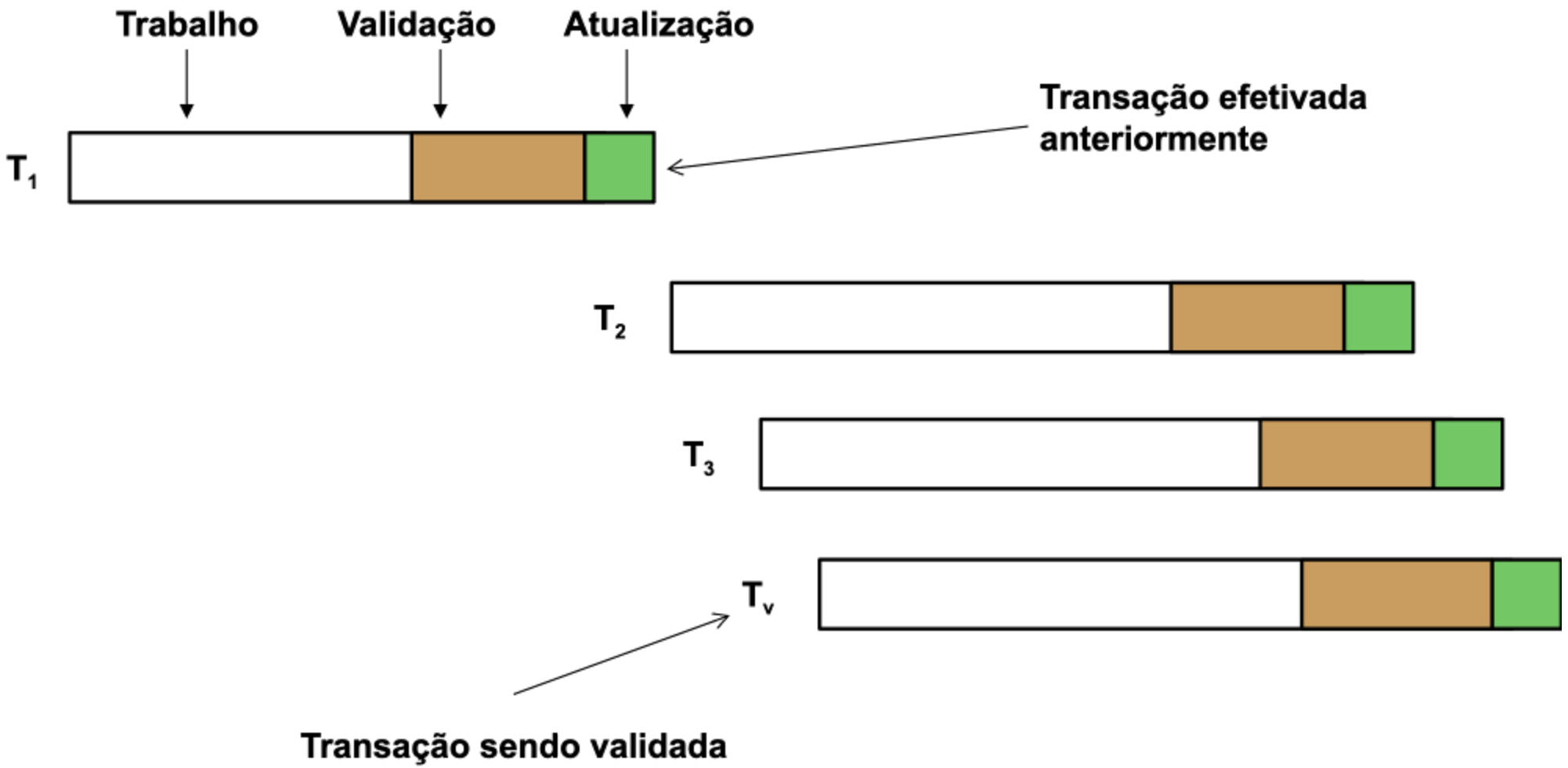
Validação para Trás (cont.)

Seja **startTn** o maior número de transação atribuído (para alguma outra transação efetivada) no momento em que a transação T_v começou sua fase de trabalho.
Seja **finishTn** o maior número de transação atribuído no momento em que T_v entrou na fase de validação.

```
boolean valid = true;
for (int Ti = startTn + 1; Ti <= finishTn; Ti ++){
   if (conjunto de leitura de Tv possui interseção com o
       conjunto de escrita de Ti)
        valid = false;
}
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Validação para Trás (cont.)



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Validação para Frente

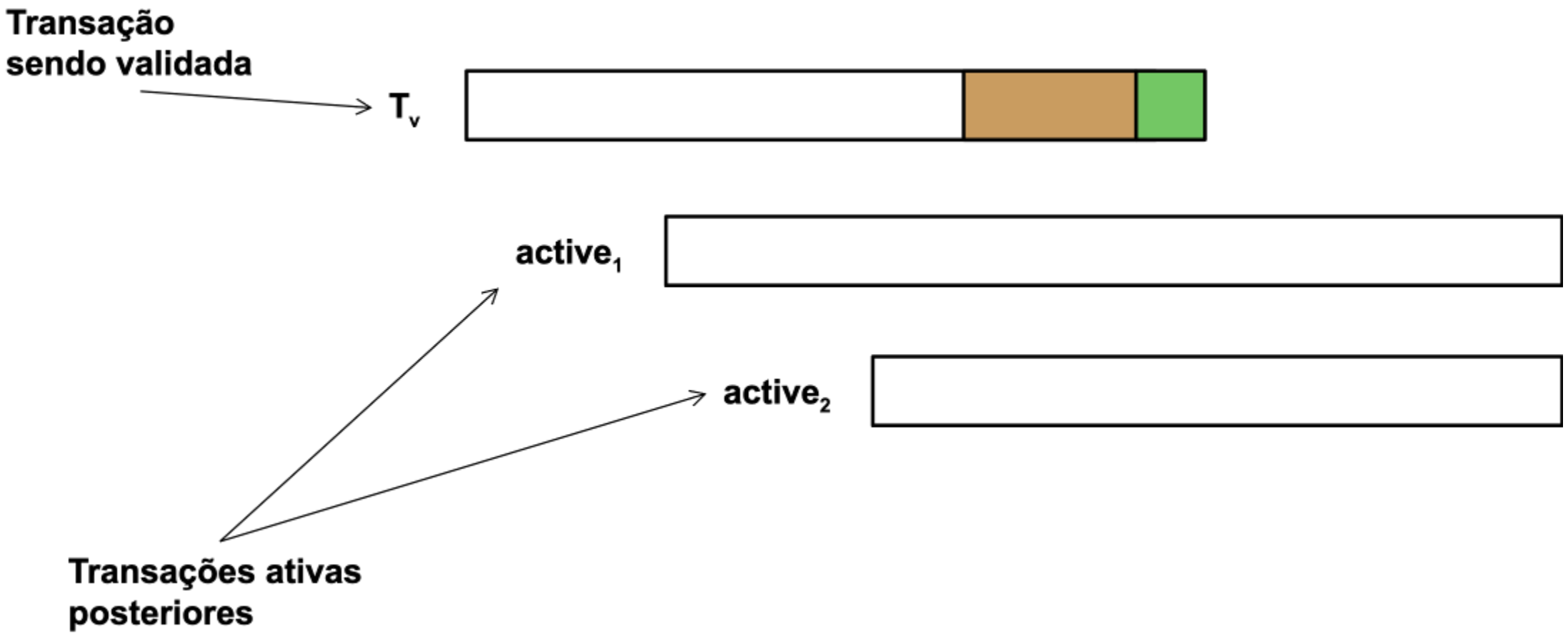
Na validação para frente da transação T_v, o conjunto de escrita de T_v é comparado com os conjuntos de leitura de todas as transações ativas sobrepostas, isto é, ainda na fase de trabalho (Regra 1).

- A Regra 2 é automaticamente satisfeita, pois as transações ativas só escrevem depois que T_v tiver terminado.
- Seja **active1** a **activeN** os identificadores (consecutivos) das transações ativas.

```
boolean valid = true;
for (int Tid = active1; Tid <= activeN; Tid ++){
   if (conjunto de escrita de Tv possui interseção com o
        conjunto de leitura de Tid)
        valid = false;
}
```

> Uma transação recebe um número apenas quando executa um **closeTransaction**.

Validação para Frente (cont.)



Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Validação para Frente (cont.)

- Como os conjuntos de leitura da transação que está sendo validada não são incluídos na verificação, as transações somente de leitura são sempre validadas.
- Como as transações que estão sendo comparadas com a transação que está sendo validada ainda estão ativas, há várias alternativas para tratar este conflito:
  - **adiar** a validação até um momento posterior quando as transações conflitantes tiverem terminado;
    - Não há garantias que o conflito desaparecerá.
  - **cancelar todas** as operações ativas conflitantes e efetivar a transação que está sendo validada;
  - **cancelar a transação** que estiver sendo validada.

Commit de Distribuído

Em uma transação distribuída, várias máquinas participam da mesma operação.

Exemplo:

- Serviço de pagamento
- Serviço de estoque
- Serviço de entrega
- Serviço de pedidos

Pergunta: Como garantir que todos confirmem ou todos cancelem?

Commit Distribuído

Uma transação distribuída só deve ser confirmada se todos os participantes puderem confirmar sua parte local. Caso contrário: A transação global deve ser abortada.

Exemplo: Compra em e-commerce.

1. Criar pedido
2. Confirmar pagamento
3. Reduzir estoque
4. Reservar entrega

Se o pagamento for aprovado, mas o estoque falhar, o sistema fica inconsistente.

Two-Phase Commit Protocol (2PC)

O 2PC divide a decisão em duas fases: **Fase de votação** e **Fase de decisão**.

Objetivo: Obter uma decisão global única: commit ou abort.

Coordenador:

- Inicia o protocolo
- Solicita votos
- Decide o resultado global
- Informa a decisão final

Participantes:

- Executam subtransações locais
- Votam commit ou abort
- Seguem a decisão do coordenador

2PC: Fase 1 (Votação)

O coordenador envia uma mensagem para todos os participantes: **vote request**.

- Cada participante verifica se consegue confirmar sua parte da transação.
- Cada participante responde:
  - **vote commit** se está pronto para confirmar.
  - **vote abort** se não pode confirmar.

2PC: Fase 1 (Votação) - Quando votar **commit**?

Um participante vota **commit** quando:

- Executou sua parte local com sucesso
- Gravou informações necessárias no log
- Está preparado para confirmar posteriormente
- Consegue garantir recuperação após falha

2PC: Fase 1 (Votação) - Quando votar **abort**?

Um participante vota **abort** quando:

- Ocorreu erro local
- Não conseguiu executar sua operação
- Não conseguiu registrar informações no log
- Houve violação de regra de negócio

Exemplo:

```
estoque insuficiente
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

2PC: Fase 2 (Decisão)

O coordenador analisa os votos.

Se todos votarem **commit**:

```
global commit
```

Se algum votar **abort** ou não responder:

```
global abort
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

2PC: Fase 2 (Decisão) - Caso 1: Todos votam commit

```
Participante A → vote commit
Participante B → vote commit
Participante C → vote commit
```

Decisão do **coordenador**:

```
global commit
```

Resultado: Todos confirmam suas alterações locais.

2PC: Fase 2 (Decisão) - Caso 2: Algum participante vota `abort`

```
Participante A → vote commit
Participante B → vote abort
Participante C → vote commit
```

Decisão do **coordenador**:

```
global abort
```

Resultado: Todos desfazem suas alterações locais.

2PC: Fase 2 (Decisão) - Caso 3: Participante não responde

Se um participante não responder dentro do tempo esperado:

```
timeout
```

O coordenador pode decidir:

```
global abort
```

Motivo: Sem unanimidade, não há commit global seguro.

2PC: Exemplo completo

```
Compra #123

Coordenador: Serviço de pedidos

Participantes:
- Pagamento
- Estoque
- Entrega
```


Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Exemplo (2PC): Fase de votação

```
Pedidos → Pagamento: vote request
Pedidos → Estoque: vote request
Pedidos → Entrega: vote request

Pagamento → Pedidos: vote commit
Estoque → Pedidos: vote commit
Entrega → Pedidos: vote commit
```

Todos estão preparados.

Exemplo (2PC): Fase de decisão

Como todos votaram commit:

```
Pedidos → Pagamento: global commit
Pedidos → Estoque: global commit
Pedidos → Entrega: global commit
```

Resultado: Compra confirmada com sucesso.

Exemplo (2PC) com falha

```
Pagamento → vote commit
Estoque → vote abort
Entrega → vote commit
```

Decisão: global abort

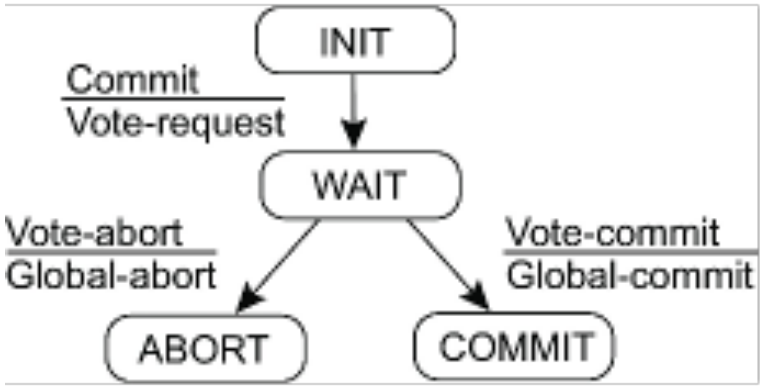
Consequência:

- Pagamento não é confirmado
- Estoque não é alterado permanentemente
- Entrega não é reservada

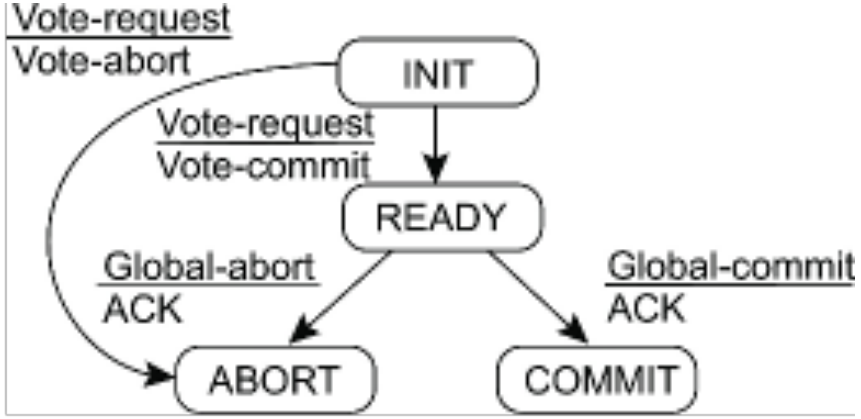
2PC: Estados do coordenador

O coordenador pode passar por estados como:

- **INIT**: protocolo iniciado
- **WAIT**: aguardando votos
- **COMMIT**: decisão global de confirmação
- **ABORT**: decisão global de cancelamento



Coordinator



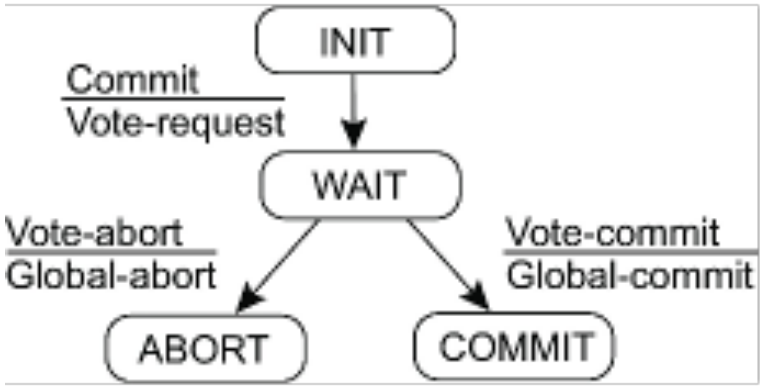
Participant

> Fonte da Imagem: Van Steen e Tanenbaum. Distributed Systems (livro), 4a edição.

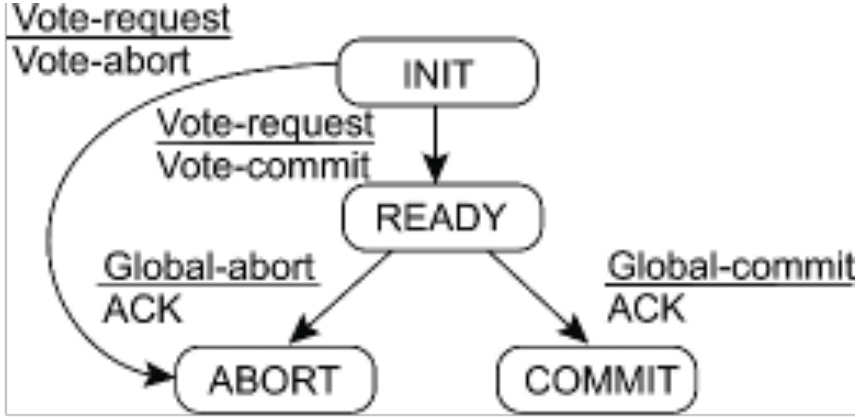
Estados dos participantes

Um participante pode passar por estados como:

- **INIT**: recebeu solicitação
- **READY**: votou commit e aguarda decisão
- **COMMIT**: confirmou localmente
- **ABORT**: cancelou localmente



Coordinator



Participant

> Fonte da Imagem: Van Steen e Tanenbaum. Distributed Systems (livro), 4a edição.

Propriedades Garantidas

O 2PC apoia principalmente:

- **Atomicidade**: Todos confirmam ou todos abortam.
- **Durabilidade**: Decisões registradas em log podem ser recuperadas após falhas.

O 2PC busca garantir:

- Decisão global única
- Consistência entre participantes

Limitação principal

O Two-Phase Commit pode ser **bloqueante**.

Isso ocorre quando:

- Um participante votou commit
- Entrou no estado **READY**
- O coordenador falhou antes de enviar a decisão final

Nesse caso, o participante **não sabe** se deve confirmar ou abortar.

Consequências do bloqueio

Enquanto espera a decisão, o participante pode manter:

- Travas
- Recursos locais
- Dados preparados
- Transações pendentes

Isso pode reduzir desempenho e disponibilidade.

Falhas no 2PC

Possíveis falhas:

- Falha do coordenador
- Falha de participante
- Perda de mensagem
- Atraso de rede
- Timeout
  O protocolo precisa de mecanismos de recuperação e tratamento de tempo limite.

Timeout

Se o coordenador não recebe todos os votos: **global abort**.

- Essa é uma decisão segura, pois não houve unanimidade para commit.

Se o participante está antes de votar commit: **pode abortar**.

Se já votou commit e está em **READY**: deve aguardar decisão.

- Esse é o caso bloqueante do 2PC.

2PC: Vantagens e Desvantagens

Vantagens:

- Simples de entender
- Amplamente usado como base conceitual
- Garante atomicidade distribuída
- Adequado para transações curtas
- Funciona bem em ambientes controlados

Desvantagens:

- Pode bloquear participantes
- Depende fortemente do coordenador
- Gera custo de comunicação
- Pode manter travas por mais tempo
- Não é ideal para transações longas ou microsserviços altamente autônomos

Conclusão

- O Two-Phase Commit (2PC) é um protocolo fundamental para garantir atomicidade em transações distribuídas.
- Sua principal ideia é simples: Primeiro todos se preparam; depois todos seguem a mesma decisão.
  - O log é essencial para recuperação
- Apesar disso, o 2PC possui custo de comunicação e pode bloquear participantes, o que exige cuidado em sistemas distribuídos reais.

Material Adicional:

- Two-phase commit (aula do Prof. Martin Kleppmann)
- Timestamp Ordering Concurrency Control (aula do Prof. Andy Pavlo)

Dúvidas e Discussão

Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Questões

Questão 1

Considere uma compra online com três serviços:

- Pagamento
- Estoque
- Entrega

O pagamento vota **commit**, o estoque vota **commit**, mas a entrega não responde.

Perguntas:

1. Qual deve ser a decisão do coordenador?
2. Por que essa decisão é segura?
3. Qual propriedade ACID está sendo protegida?

Questão 2

Em um sistema de saúde, uma transação distribuída atualiza:

- Estoque da Cardiologia
- Estoque da Emergência
- Registro de auditoria

Pergunta: O 2PC seria adequado? Quais riscos existiriam se o coordenador falhasse?