Transações

Bases de Dados

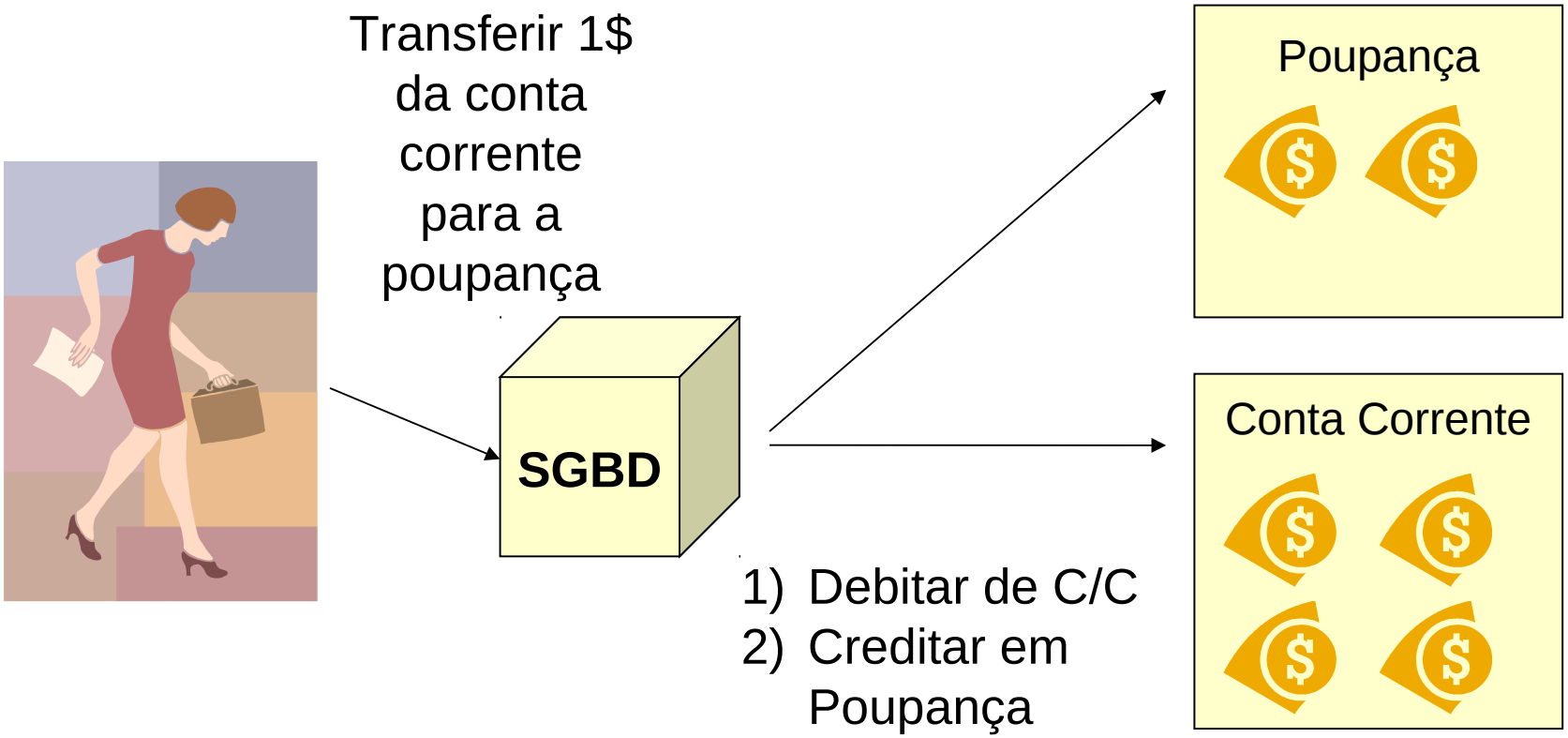
Moacir P. Ponti Jr.

Tópicos

1. Conceito de Transação
2. Estado da Transação
    1. Máquina de Estados
3. Execuções Concorrentes
4. Transações em SQL

Transações

- Um conjunto de várias operações no banco de dados é uma única unidade do ponto de vista do usuário



Transações

- No entanto, como vimos, em muitos casos se tratam de diversas operações
- E se uma delas der errado?
- O que é preferível no caso de falha?
  - **Que apenas uma delas ocorra.**
  - **Que nenhuma delas ocorra.**

Transações

- As operações que formam uma única unidade lógica de trabalho, são chamadas de *Transações*.
- Um sistema de banco de dados precisa garantir a execução apropriada das transações a despeito de falhas
  - ou a transação é executada por completo ou nenhuma parte dela é executada

1 - Conceito de Transação

- **Transação** é uma unidade de execução de programa que acessa e/ou atualiza dados
- A transação em geral é o resultado da execução de um programa de usuário, escrito em uma linguagem de manipulação de dados ou outra
  - **SQL**
  - **Java**
  - **C**
  - **...**

1 - Conceito de Transação

- Para assegurar **integridade dos dados**, é preciso que o sistema tenha as seguintes propriedades:
  - **Atomicidade**: ou todas as operações da transação se realizam com sucesso, ou nenhuma será.
  - **Consistência**: a execução de uma transação isolada (sem a execução de outra concorrente), deve preservar a consistência do banco de dados
  - **Isolamento**: cada transação é executada de forma independente, não tomando conhecimento de outras concorrentes no sistema
  - **Durabilidade**: depois da transação completar-se, as mudanças feitas persistem.

1 - Conceito de Transação

- Para exemplificar, considere que a base de dados reside em disco, mas que, para ser processado precisa ser copiado, temporariamente, na memória principal.
- O acesso ao banco é dado pelas operações

  read(X) – transfere o item de dados X do banco para um buffer local.

  write(X) – transfere o item de dados X do buffer local para o banco de dados.

1 - Transação: Consistência

- Seja T_i uma transação que transfere 50 reais de uma conta A para uma conta B, a transação pode ser definida como:

T_i: **read**(A);
A = A – 50;
write(A);
read(B);
B = B + 50;
write(B);

Consistência:

- Neste caso, a consistência é obtida se ao final da transação a soma A+B permanece inalterada.
- Tarefa de resposabilidade do programador da aplicação, mas pode ser facilitada por ferramentas do banco de dados.

1 - Transação: Atomicidade

- Seja T_i uma transação que transfere 50 reais de uma conta A para uma conta B, a transação pode ser definida como:

T_i: **read**(A);
A = A – 50;
write(A);
read(B);
B = B + 50;
write(B);

Atomicidade:

- Suponha que durante a execução de T_i, houve uma falha logo antes da execução de **write**(B).
- O *estado do sistema será* inconsistente. Todas as operações devem ser desfeitas.
- Isto é feito mantendo um backup em disco (controle maioria SGBDs).

1 - Transação: Atomicidade

- Seja T_i uma transação que transfere 50 reais de uma conta A para uma conta B, a transação pode ser definida como:

T_i: **read**(A);
A = A – 50;
write(A);
read(B);
B = B + 50;
write(B);

Durabilidade:

- Se a transação foi completada com sucesso e o usuário que a disparou for notificado da transferência do dinheiro, não houve falha no sistema.
- Assim, estas alterações devem persistir no banco de dados (serem gravadas no disco), mesmo que haja falha posteriormente.

1 - Transação: Atomicidade

- Seja T_i uma transação que transfere 50 reais de uma conta A para uma conta B, a transação pode ser definida como:

T_i: **read**(A);
A = A – 50;
write(A);
read(B);
B = B + 50;
write(B);

Isolamento:

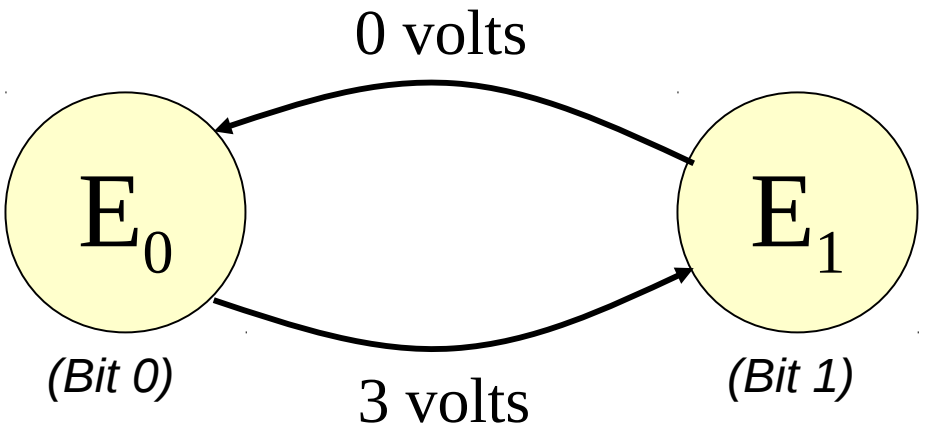
- Se outra transação fizer uma leitura das contas A e B logo após a transação T_i executar write(A), ela fará uma leitura irreal.
- Pode-se executar as transações em série (uma depois da outra)
- Mas há ganho de performance em executar de modo concorrente, o que requer um controle.

2 – Estados da Transação

- Na ausência de falhas, uma transação se completa com sucesso.
- Mas nem sempre isto é possível e a transação deve ser abortada quando há falhas.
  - O sistema deve garantir que uma transação com falhas não tenha nenhum efeito sobre o estado do banco de dados
- Quando há falhas, a transação deve ser desfeita (*rolled back*)
- Quando é completada com sucesso, a transação deve ser efetivada (*committed*)

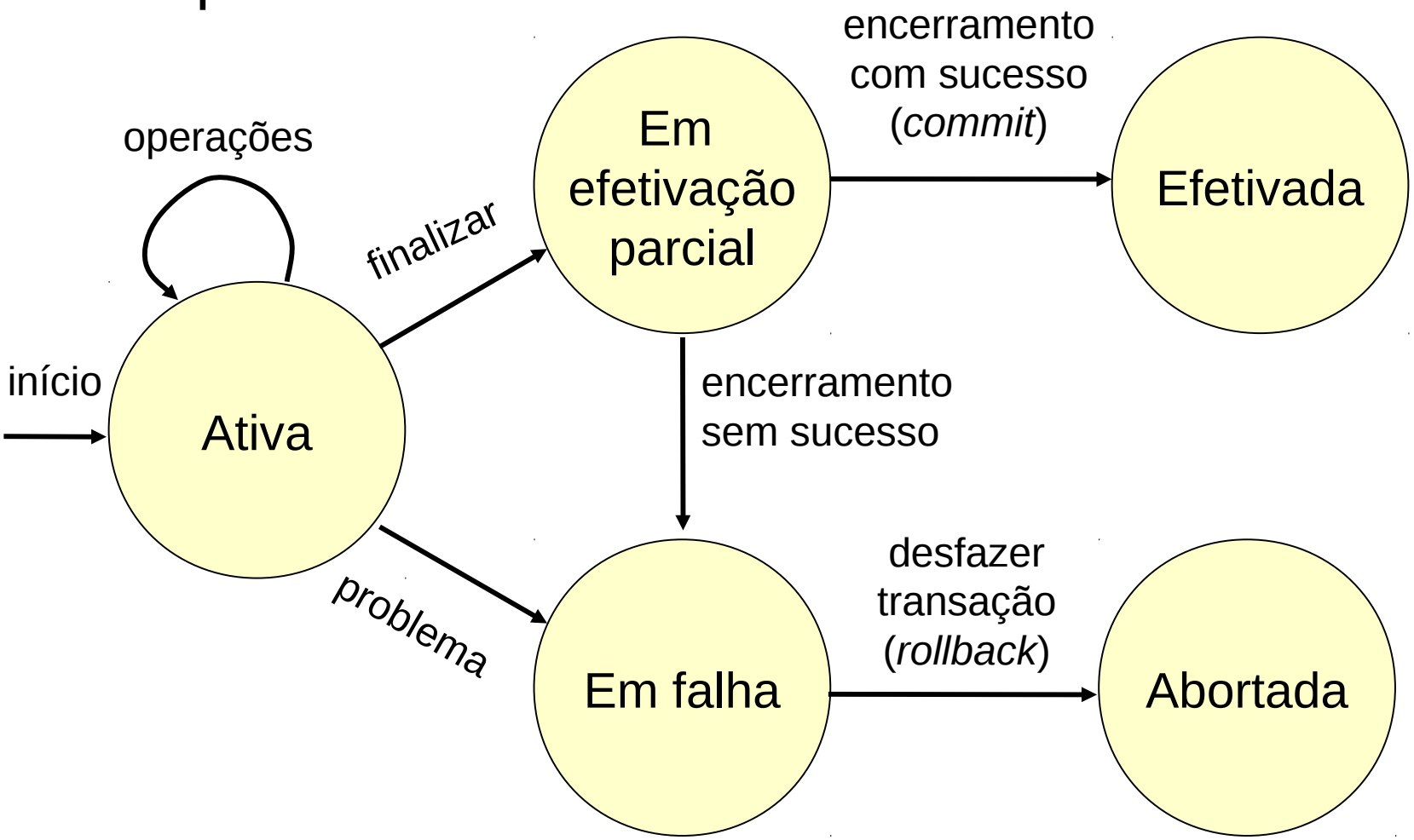
2 – Estados da Transação

- O que é uma máquina de estados?
- É a modelagem de um comportamento
- A máquina permanece em um estado - que fica armazenado - até que alguma ação leve a máquina a outro estado
- O computador é uma máquina de estados



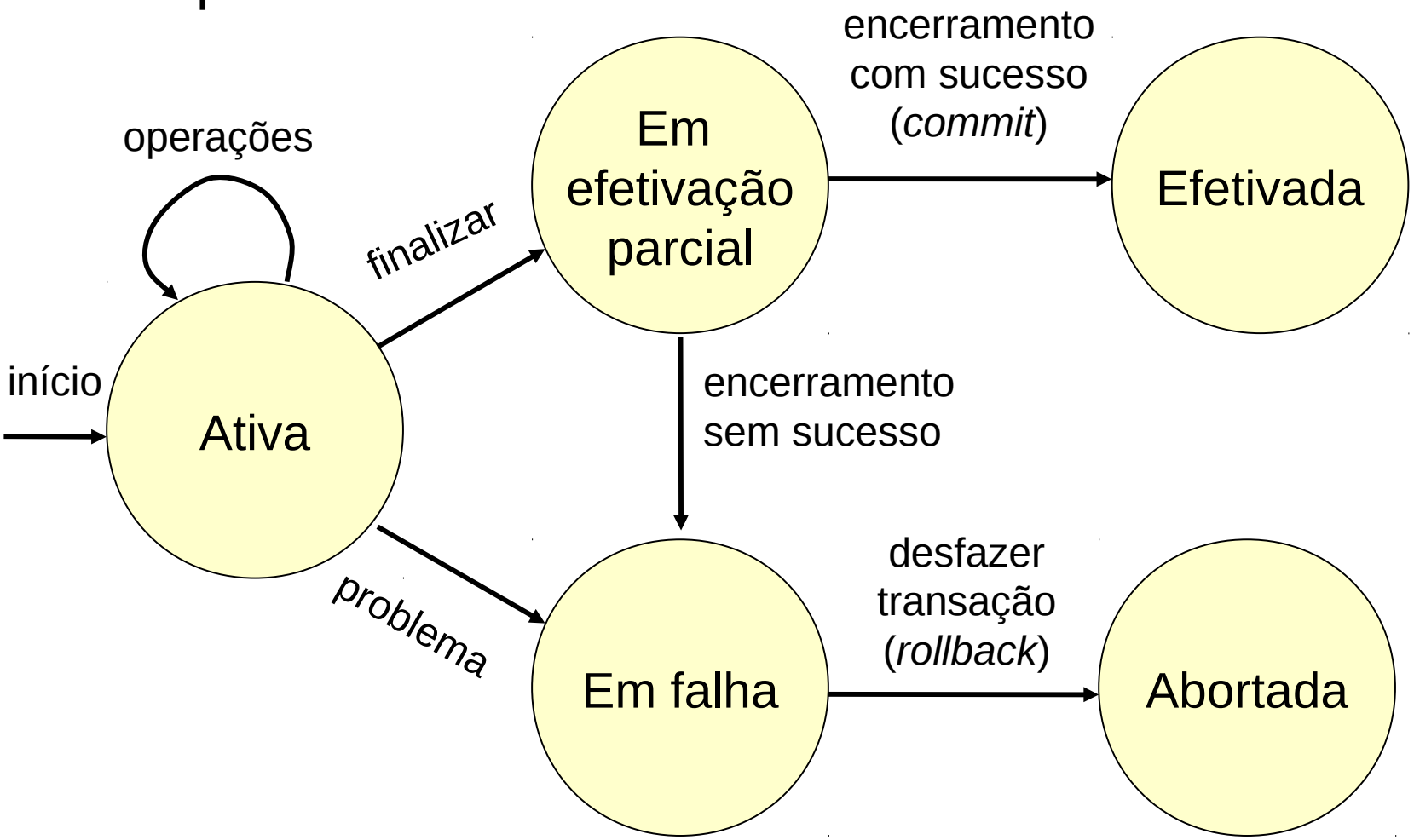
2 – Estados da Transação

Exemplo de Falha



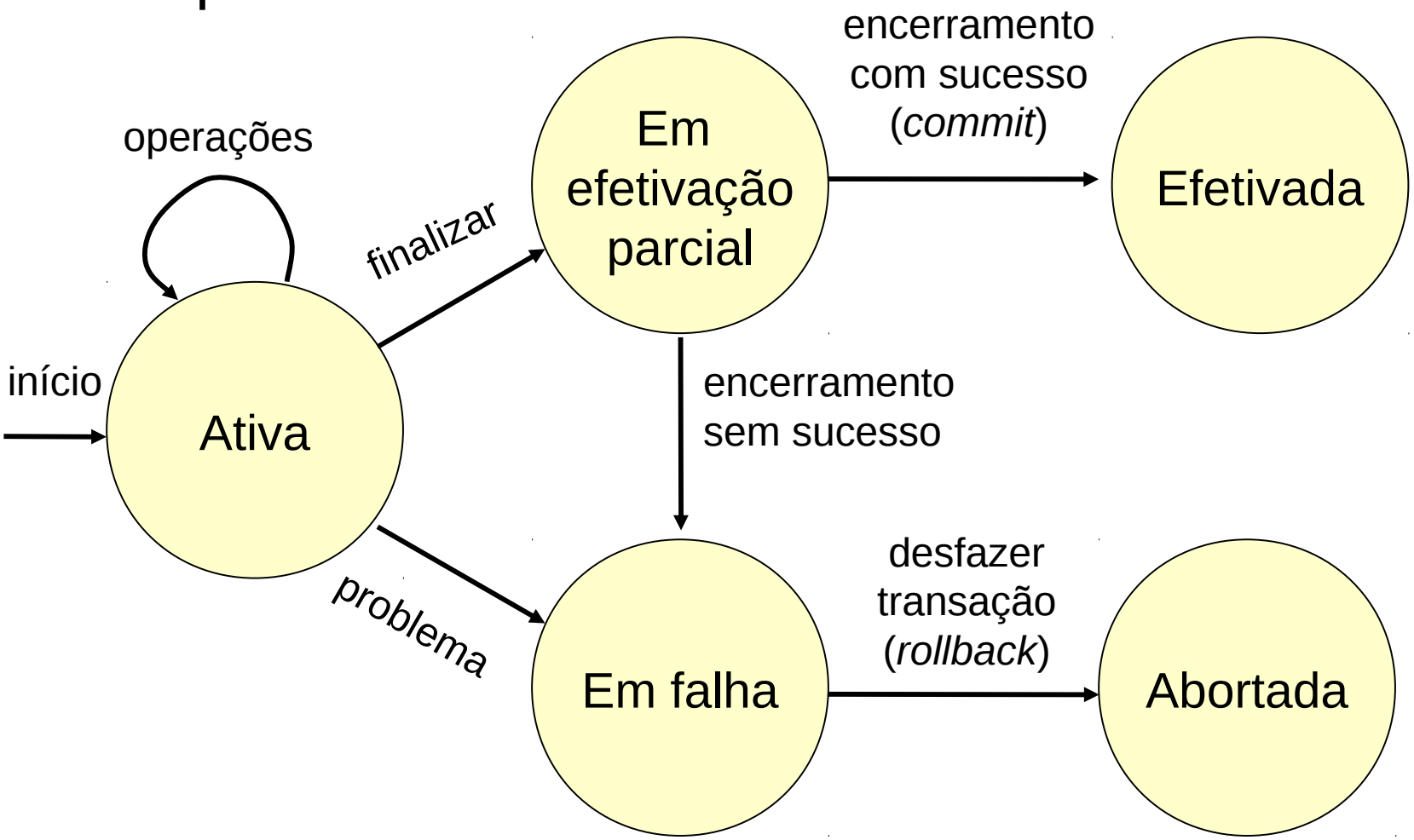
2 – Estados da Transação

Exemplo de Falha 2



2 – Estados da Transação

Exemplo de Sucesso



3 – Execuções Concorrentes

- É muito mais fácil permitir que as transações executem sequencialmente.
- A execução concorrente pode gerar problemas e isto necessita de um maior controle
- Mas há duas boas razões para permitir a concorrência:
  - Uma transação ocorre em diversos passos, que envolvem atividades diferentes (E/S, Processamento, ...), de forma que estas poderiam ser executadas ao mesmo tempo
  - Se as transações estão operando em partes diferentes do BD é melhor deixá-las concorrer de modo a compartilhar os ciclos de CPU e os acessos do disco entre si.

3 – Execuções Concorrentes

- T_1 transfere 50 reais da conta A para a conta B
- T_2 transfere 10% do saldo da conta A para a conta B

```
T1: read(A);
    A = A – 50;
    write(A);
    read(B);
    B = B + 50;
    write(B);
```

```
T2: read(A);
    tmp = A*0,1;
    A = A – tmp;
    write(A);
    read(B);
    B = B + tmp;
    write(B);
```

3 – Execuções Concorrentes

- Suponha que, inicialmente, os valores em A e B são, respectivamente, 1000 e 2000 reais.
  - ∞ Simule duas execuções seqüenciais das transações:
    1. Com T_1 e depois T_2
    2. Com T_2 e depois T_1
  - ∞ Verifique os valores finais e se a consistência foi mantida

T_1:

```
read(A);
A = A – 50;
write(A);
read(B);
B = B + 50;
write(B);
```

T_2:

```
read(A);
tmp = A*0,1;
A = A – tmp;
write(A);
read(B);
B = B + tmp;
write(B);
```

3 – Execuções Concorrentes

T_1	T_2	A = 1000, B = 2000, A+B = 3000
read(A);		
A = A – 50;		
write(A);		A = 950
read(B);		
B = B + 50;		
write(B);		B = 2050
	read(A);	
	tmp = A*0,1;	tmp = 95
	A = A – tmp;	
	write(A);	A = 855
	read(B);	
	B = B + tmp;	
	write(B);	B = 2145

A+B = 3000

3 – Execuções Concorrentes

T_1	T_2	A = 1000, B = 2000, A+B = 3000
	read(A);	
	tmp = A*0,1;	tmp = 100
	A = A – tmp;	
	write(A);	A = 900
	read(B);	
	B = B + tmp;	
	write(B);	B = 2100
read(A);		
A = A – 50;		
write(A);		A = 850
read(B);		A+B = 3000
B = B + 50;		
write(B);		B = 2150

3 – Execuções Concorrentes

- Se rodássemos as duas transações simultaneamente, no entanto, o sistema operacional iria escalonar o processamento, podendo gerar diferentes sequências
  - Algumas podem gerar estados corretos, outras podem gerar estados inconsistentes

3 – Concorrência: Exemplo 1

- Suponha os valores iniciais de A e B como 1000 e 2000 reais.
  - Simule a execução concorrente ao lado
  - Verifique os valores finais e se a consistência foi mantida

Valores Finais:
A = 855 B = 2145
A+B = 3000

T_1	T_2
read(A);	
A = A – 50;	
write(A);	
	read(A);
	tmp = A*0,1;
	A = A – tmp;
	write(A);
read(B);	
B = B + 50;	
write(B);	
	read(B);
	B = B + tmp;
	write(B);

3 – Concorrência: Exemplo 2

- Suponha os valores iniciais de A e B como 1000 e 2000 reais.
  - Simule a execução concorrente ao lado
  - Verifique os valores finais e se a consistência foi mantida

Valores Finais:
A = 950 B = 2100
A+B = 3050

T_1	T_2
read(A);	
A = A – 50;	
	read(A);
	tmp = A*0,1;
	A = A – tmp;
	write(A);
	read(B);
write(A);	
read(B);	
B = B + 50;	
write(B);	
	B = B + tmp;
	write(B);

3 – Concorrência e Serialização

- As únicas operações significativas, em termos de escala de execução, são as instruções de leitura e escrita.
- Deve-se tomar cuidado com *conflito* quando das operações de escrita e leitura em mesma unidade de dados.
- É possível *travar* uma unidade de dados que se está trabalhando, ou deixá-la aberta para que outras transações possam ler
- Há várias opções possíveis de se implementar, como veremos.

4 – Transações em SQL

- O uso de SQL para controle do nível de isolamento num bloco de transação utiliza:
- START TRANSACTION – define o nível de isolamento do bloco de transação (a ser iniciado)

```
START TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED |
  READ COMMITTED |
  REPEATABLE READ |
  SERIALIZABLE }
```

4 – Transações em SQL

- Níveis de isolamento do SET TRANSACTION:
  - **Serializable** – transação será executada com completo isolamento
  - **Repeatable Read** – transação só poderá ler dados já efetivados e outras transações não poderão escrever em dados lidos por esta transação *(este é o padrão em muitos SGBDs) *alguns dados podem não ser acessíveis*
  - **Read Commited** – transação só poderá ler dados já efetivados, mas outras transações poderão escrever em dados lidos por ela.
  - **Read Uncommited** – transação poderá ler dados que ainda não foram efetivados

4 – Transações em SQL

- COMMIT – solicita a efetivação das transações

```
COMMIT [TRANSACAO];
```

- ROLLBACK – solicita a que as ações sejam desfeitas

```
ROLLBACK [TRANSACAO];
```

4 – Transações em SQL: exemplo 1

```
SET TRANSACTION ISOLATION LEVEL READ COMMITED;

   INSERT INTO Empregados
   VALUES (:Pcodemp, :Pnome, :Pdatanasc, :Psalario);

   UPDATE Empregados
   SET Salario = Salario + 100
   WHERE Funcao = 'Secretariado‘;

// Se a operação não gerou erros
 COMMIT;
// Se a operação gerou erros
 ROLLBACK;
```

4 – Transações em SQL: exemplo 2

- Alguns SGBDs utilizam o formato abaixo

```
BEGIN TRANSACTION TRANSACAO1
ISOLATION LEVEL REPEATABLE READ;

  INSERT INTO Empregados
   VALUES (:Pcodemp, :Pnome, :Pdatanasc, :Psalario);

  UPDATE Empregados SET Salario = Salario + 100
   WHERE Funcao = 'Professor';

// Se a operação não gerou erros
 COMMIT TRANSACTION TRANSACAO1;
// Se a operação gerou erros
 ROLLBACK TRANSACTION TRANSACAO1;
```

Bibliografia Básica

- SILBERSCHATZ, A. et al. **Transações** (capítulo 13). Em: **Sistema de Banco de Dados**. 3.ed. 1999.