Aula 8

Transações atômicas

1.1 O quê?
1.2 Porquê?
1.3 Como?
- 1.3.1 Log based recovery
- 1.3.2 Transações atômicas concorrentes
- 1.3.3 Two phase locking protocol

O quê são transações atômicas (1)

Do *Houais* - Átomo:

gr. átomos,os,on 'que não pode ser cortado, indivisível'

Transação atômica:

“Uma coleção de operações que executa uma operação lógica de forma indivisível”

O quê são transações atômicas (2)

Imagine que você está fazendo uma compra:

- Você entrega seu cartão para o caixa
  ... e digita sua senha
- O caixa se conecta com seu banco
  ... o banco retira o valor da compra de sua conta
  ... mas, antes da loja receber seu dinheiro
  ... a energia cai!

  Para onde foi seu dinheiro???

O quê são transações atômicas (3)

Dois cheques são depositados par você em duas agências:

- R$200,00 na 1a e R$500,00 na 2a

- A 1a obtém seu saldo: R$300,00
  ... mas o canal é lento, e ela demora a seguir
- A 2a obtém seu saldo: R$300,00
  O canal é rápido, e ela ajusta seu saldo: R$800,00
- Agora, a primeira ajusta o saldo: R$500,00!

Para onde foi o seu dinheiro?

O quê são transações atômicas (4)

Para se garantir a consistência dos dados é necessário garantir-se que as transações sejam executadas da maneira correta – **atômica - indivisível**:

- Ou a função lógica é executada completamente (a venda e o depósito, nos exemplos anteriores), ou não é executada. Não pode ser executada “pelas metades”.

Porquê é difícil implementar (1)

Uma maneira de se garantir que a transação seja executada de maneira correta é usar a **exclusão mútua**

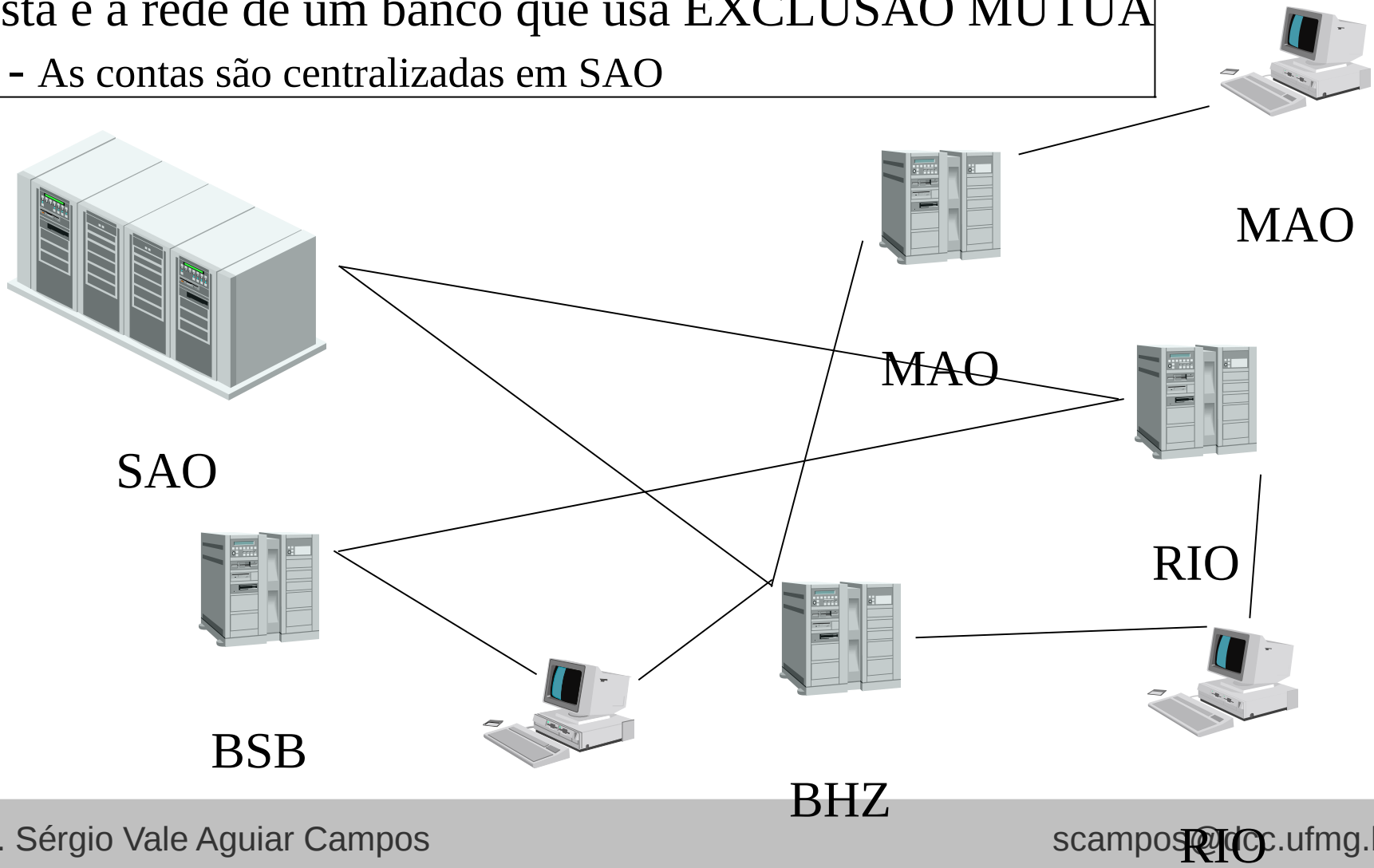
Mas...isto não é uma boa idéia!

- Funciona para as duas agências.
- Mas não ajuda em caso de queda de energia.
- E pode travar muito mais recursos que o necessário!

Porquê é difícil implementar (2.1)

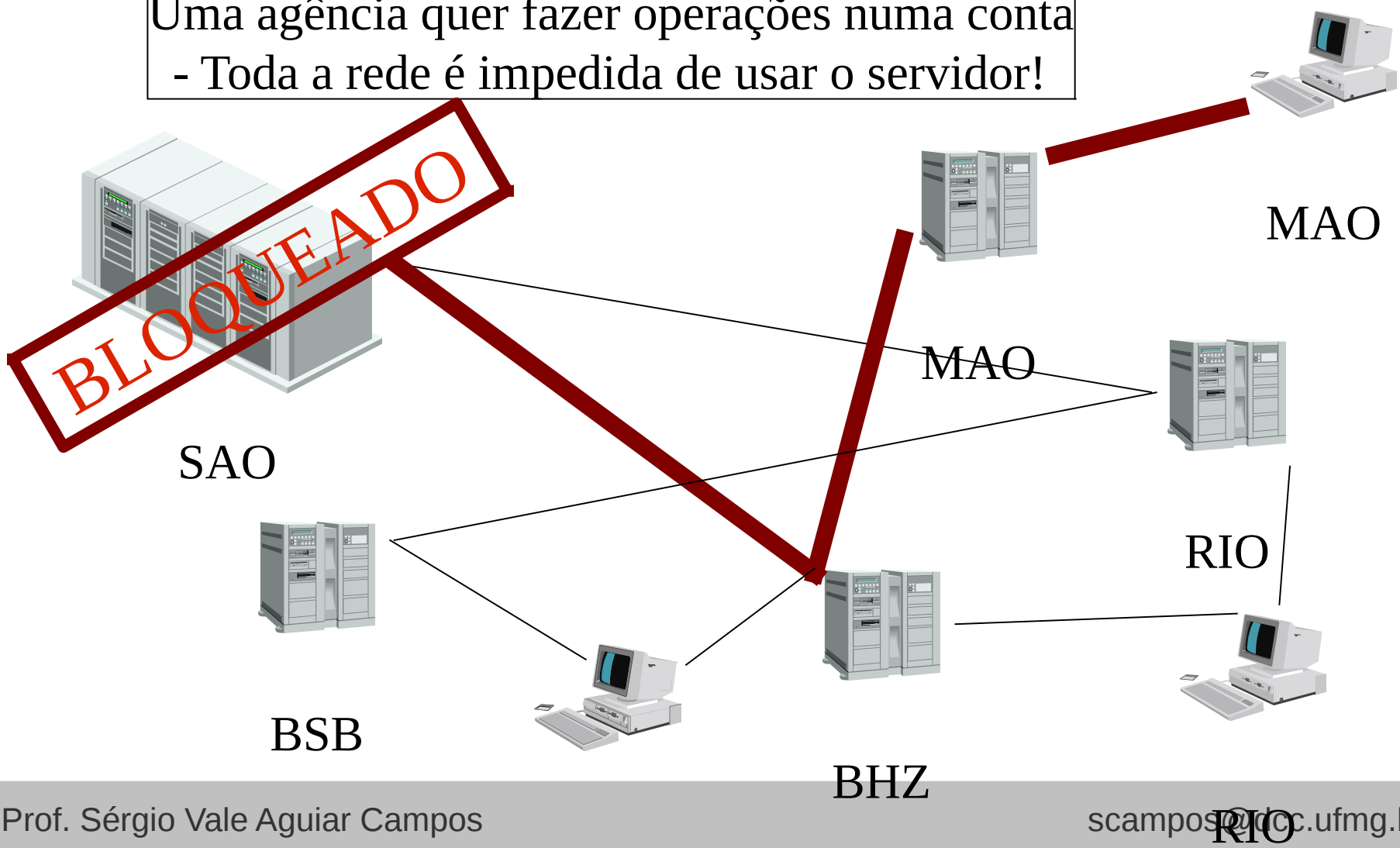
Esta é a rede de um banco que usa EXCLUSÃO MÚTUA

- As contas são centralizadas em SAO



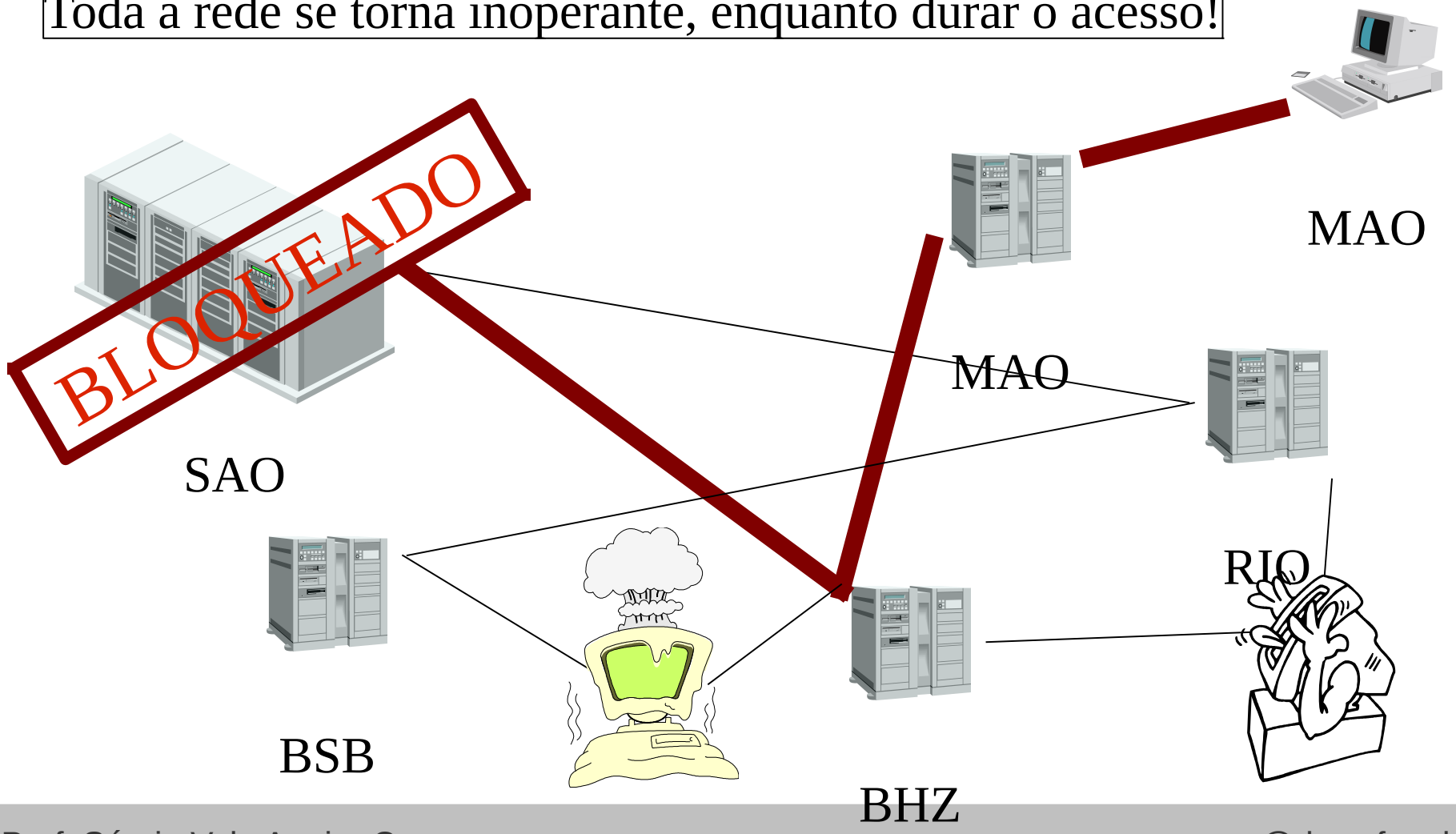
Porquê é difícil implementar (2.2)

Uma agência quer fazer operações numa conta
- Toda a rede é impedida de usar o servidor!



Porquê é difícil implementar (2.3)

Toda a rede se torna inoperante, enquanto durar o acesso!



Porquê é difícil implementar (3)

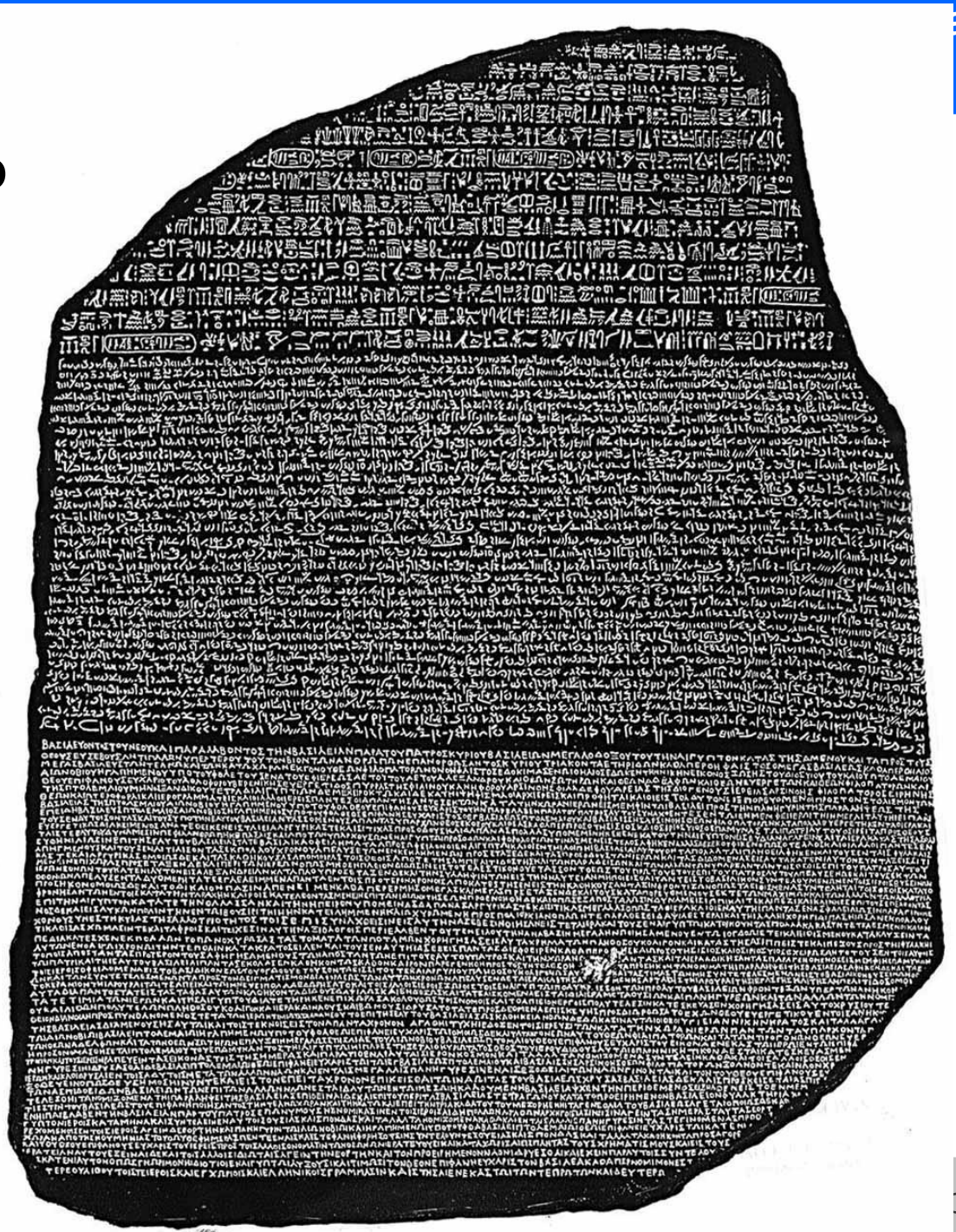
Vários problemas podem afetar o resultado de uma transação atômica:

- Picos de luz
- Queda de conexão
- Falhas de CPU
- Falhas em discos e em memória:
  - Nenhum armazenamento é 100% confiável
- ... e muitas outras falhas

Como implementar – Classificação do armazenamento (1)

- Volátil: informação não sobrevive à queda do sistema
  - Acesso muito rápido
  - Ex: memória principal
- Não volátil: informação normalmente sobrevive à queda do sistema:
  - Acesso significativamente mais lento
  - Ex: disco rígido
- Estável: informação *nunca* se perde
  - Nunca mesmo?
  - Discos replicados, fita, CDs, DVDs, escrito com martelo na pedra...

Armazenamento Estável



Como implementar

Para cada categoria de armazenamento, há diferentes problemas e diferentes soluções.

Nós nos concentraremos nos protocolos de sobrevivência para
falhas no armazenamento volátil.

Como implementar – Exemplo de transação (1)

Exemplo de transação:

Relembrando o que é uma transação...

- **Coleção de operações** que executa uma **função lógica indivisível**

Como implementar – Exemplo de transação (2)

```
vende_peça(peça, quant, valor, cliente)
{
  read(estoque, peça);
  peça->quant -= quant;
  write(estoque, peça);
  read(controle, ultima_transação);
  ultima_transação++;
  write(controle, ultima_transação);
  write(transações, ultima_transação,
        VENDA, peça, quant, valor, data);
  write(contas_a_receber, data, valor,
        cliente);
}
```

Como implementar – *Log based recovery* (1)

Recuperação usando log – *log based recovery*:

Usa-se um arquivo “diário” **em armazenamento estável** para gravar as partes da transação que já foram executadas: arquivo de **log**.

Grava-se neste arquivos **toda e qualquer** operação **antes** de se a executar.

Como implementar – *Log based recovery* (2)

```
vende_peça(peça, quant, valor, cliente)
{
  write(log, "T%d starts", i);
  read(estoque, peça);
  peça->quant -= quant;
  write(log, estoque, peçaold, peça);
  write(estoque, peça);
  read(controle, ult_tr_#);
  ultima_transação++;
  write(log, controle,
        ult_tr_#old, ul_tr_#);
  write(controle, ultima_transação);
  write(log, transações, ...);
  write(transações, ultima_transação,
        VENDA, peça, quant, valor,
   data);
  write(log, contas_a_receber, ...);
  write(contas_a_receber, data, valor,
        cliente);
  write(log, "T%d commits", i);
}
```

Início da transação

Etapas concluídas

Término da transação

Como implementar – *Log based recovery* (3)

Funções de recuperação de falhas simples:

- **undo(i)**
- **redo(i)**

Aonde buscar os valores para **undo** e **redo** usarão para trabalhar?

- No log!

Como implementar – *Log based recovery* (4)

Cuidados:

- **Todo** acesso tem que seguir o protocolo (se alguém esquecer de usar o log, não funciona, porque **undo** e **redo** não funcionarão)
- **undo** e **redo** têm que ser **idempotentes**. Executadas repetidas vezes, o resultado não pode variar (porquê ?)
- A recuperação pode ser **muito** lenta:
  - É preciso varrer todo o log para fazer a recuperação

Como implementar – *Checkpoints*

Varrer todo o log é demorado...

... como acelerar o processo?

O sistema grava **checkpoints** assim:

- Grava todos os registros do **log** em memória estável (*flush*)
- Grava todos os registros relativos à transação em memória estável (*flush*)
- Grava um registro **T*i* checkpoint** no log

A recuperação é feita a partir do último checkpoint gravado no log

Como implementar – Transações atômicas concorrentes (1)

Frequentemente, transações atômicas podem ocorrer concorrentemente:

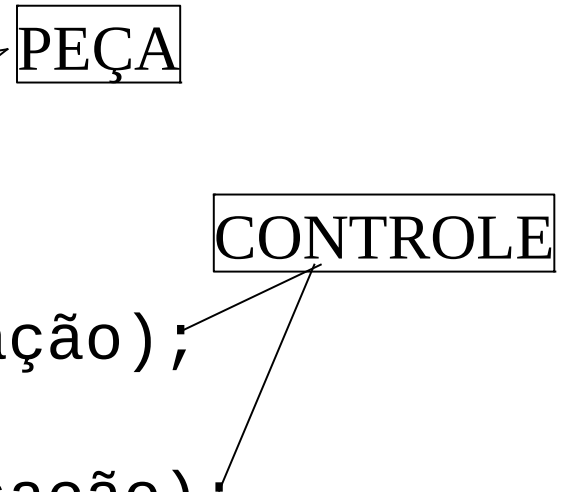
- Duas agências ajustam o saldo de uma conta
- Dois vendedores vendem peças

Então, é preciso garantir a **serializabilidade** (!). O efeito tem que ser:

- Ou A é executado e depois B, ou B é executado e depois A

Como implementar – Transações atômicas concorrentes (2)

```
vende_peça(peça, quant, valor, cliente)
{
  read(estoque, peça);
  peça->quant -= quant;
  write(estoque, peça);
  read(controle, ultima_transação);
  ultima_transação++;
  write(controle, ultima_transação);
  write(transações, ultima_transação,
        VENDA, peça, quant, valor, data);
  write(contas_a_receber, data, valor,
        cliente);
}
```



Como implementar – Transações atômicas concorrentes (3)

T0

```
read(estoque,peça)
peça->quant-= quant
write(estoque,peça)



read(controle,ult_trans)
ult_trans++
write(controle,ult_trans)
```

BOM ESCALONAMENTO

T1

```
read(estoque,peça)
peça->quant-= quant
write(estoque,peça)



read(controle,ult_trans)
ult_trans++
write(controle,ult_trans)
```

Como implementar – Transações atômicas concorrentes (4)

T0

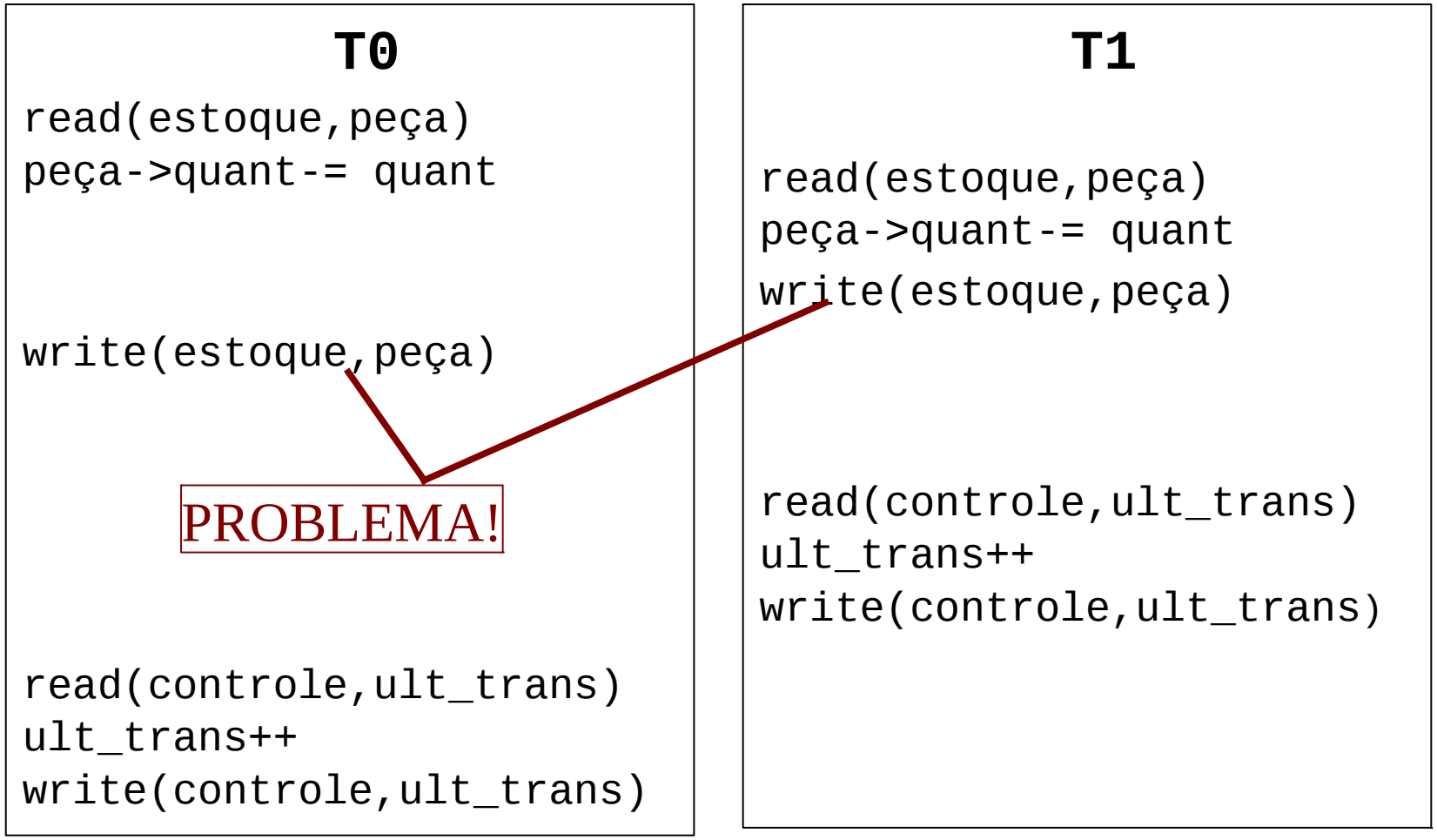
```
read(estoque,peça)
peça->quant-= quant
write(estoque,peça)
read(controle,ult_trans)
ult_trans++
write(controle,ult_trans)
```

BOM ESCALONAMENTO

T1

```
read(estoque,peça)
peça->quant-= quant
write(estoque,peça)
read(controle,ult_trans)
ult_trans++
write(controle,ult_trans)
```

Como implementar – Transações atômicas concorrentes (5)



Como implementar – Locking Protocol (1)

Para se garantir a seriali....., este negócio ai, podemos usar o protocolo:
"Cada leitura trava o registro no arquivo”
“O valor pode ser usado e modificado”
“Cada escrita destrava”

```
good_read(arquivo, id) {
  lock(mutex);
  read(arquivo, id, r)
    until (r != “locked”)
  write(arquivo, id, “locked”);
  unlock(mutex);
  return(r);
}
```

Como implementar – Locking Protocol(2)

Pode-se travar o registro como **shared** ou **exclusive**. Lembram-se dos leitores/escritores?

- Se você vai apenas consultar o valor, **shared** é bom o bastante.
- Se você vai ler e alterar, tem que ser **exclusive.**

Todos lêem o arquivo assim.

Como implementar – Two Phase Locking Protocol (1)

O protocolo de leitura/escrita consiste de 2 fases:

- **Growing phase**: a transação pode ler dados (lock), mas não gravar (unlock).
- **Shrinking phase**: a transação pode gravar dados, mas não pode ler.

Resumindo: na hora em que um **write** for executado, nehum **read** pode ser mais executado.

Este protocolo:

- Garante a seriazabilidade.
- Não garante a ausência de *deadlocks.*

Como implementar – Two Phase Locking Protocol (2)

```
vende_peça(peça, quant, valor, cliente)
{
  /* growing phase */
  read(estoque, peça);
  estoque->quant -= quant;
  read(controle, ultima_transação);
  read(transações);
  read(contas_a_receber);

  /* shrinking phase */
  write(estoque, peça);
  ultima_transação++;
  write(controle, ultima_transação);
  write(transações, ultima_transação,
        VENDA, peça, quant, valor, data);
  write(contas_a_receber, data, valor,
        cliente);
}
```

Como implementar – Two Phase Locking Protocol (3)

T0

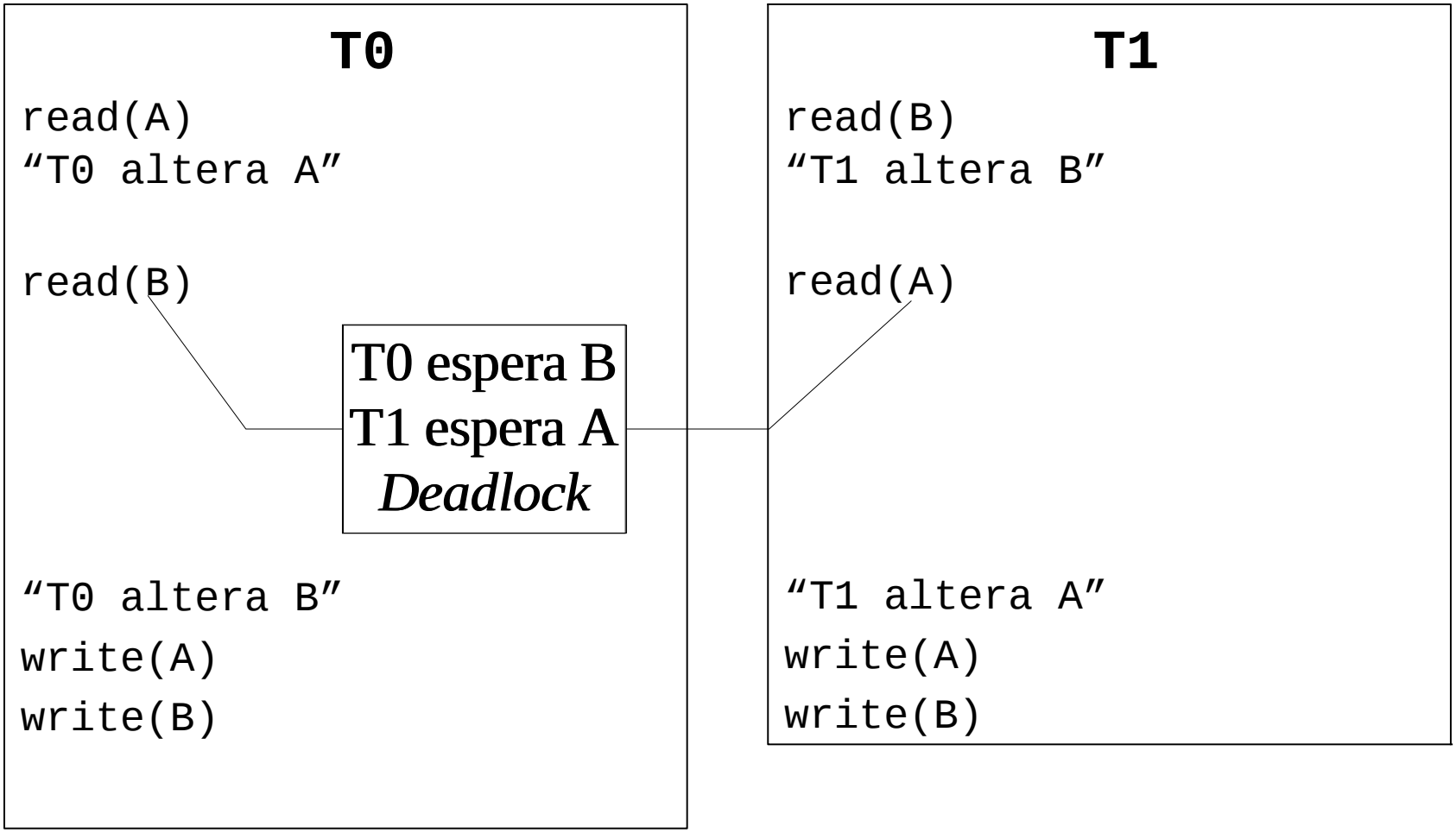
```
read(estoque,peça)
peça->quant-= quant
read(controle,ult_trans)
ult_trans++
write(estoque,peça)
write(controle,ult_trans)
```

T1 pode seguir, porque T0 já liberou *peça*

T1

```
read(estoque,peça)
peça->quant-= quant
read(controle,ult_trans)
ult_trans++
write(estoque,peça)
write(controle,ult_trans)
```

Como implementar – Two Phase Locking Protocol (4)



Resumo – Transações atômicas

Transação atômica:

"Uma coleção de operações que executa uma operação lógica de forma indivisível"

Implementação:

- *Log based recovery*
- *Checkpoints* para acelerer recuperação
- *Two-phase locking protocol*