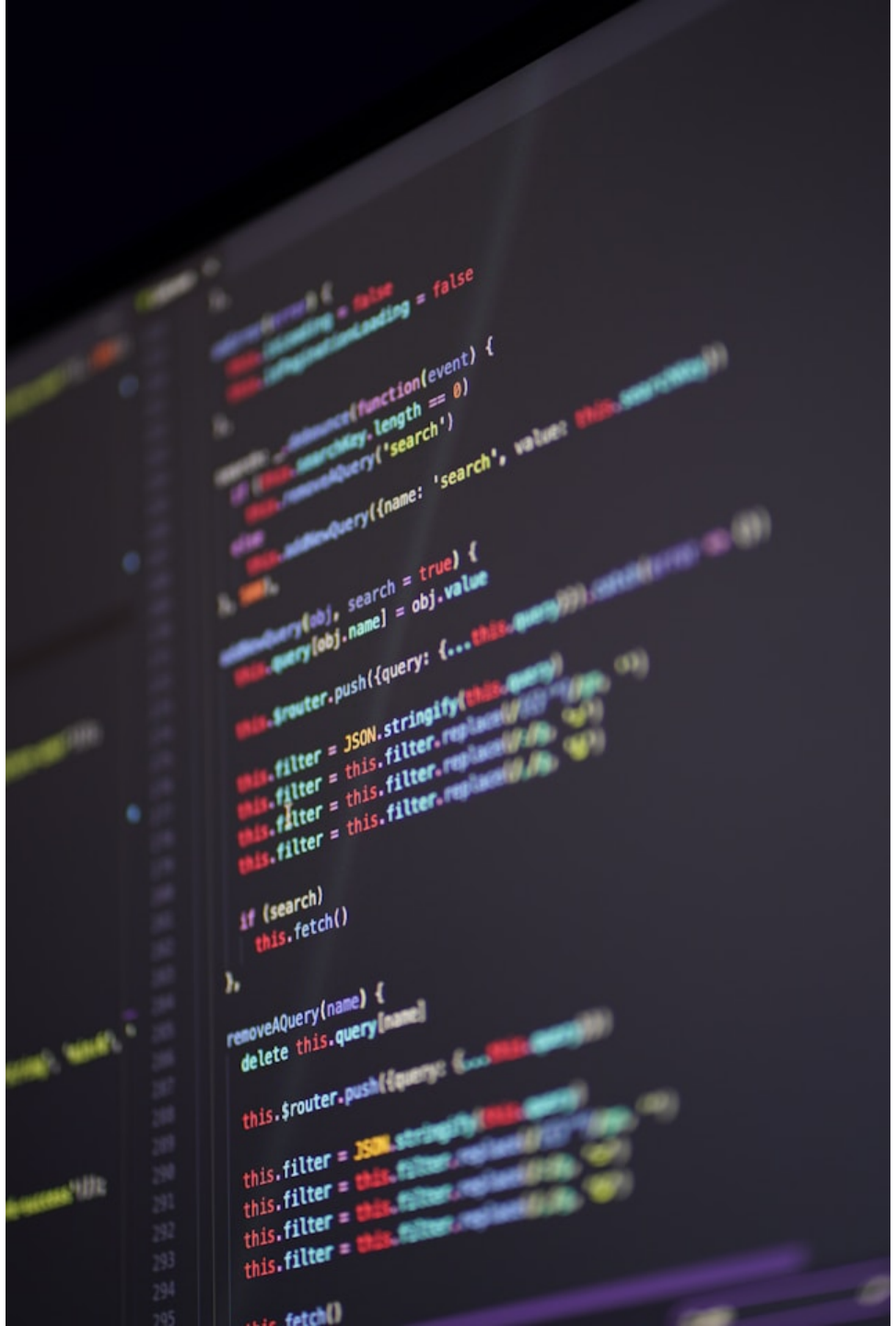


5954025 - Sistemas Distribuídos

Aula 12b - Transações (Parte 2)

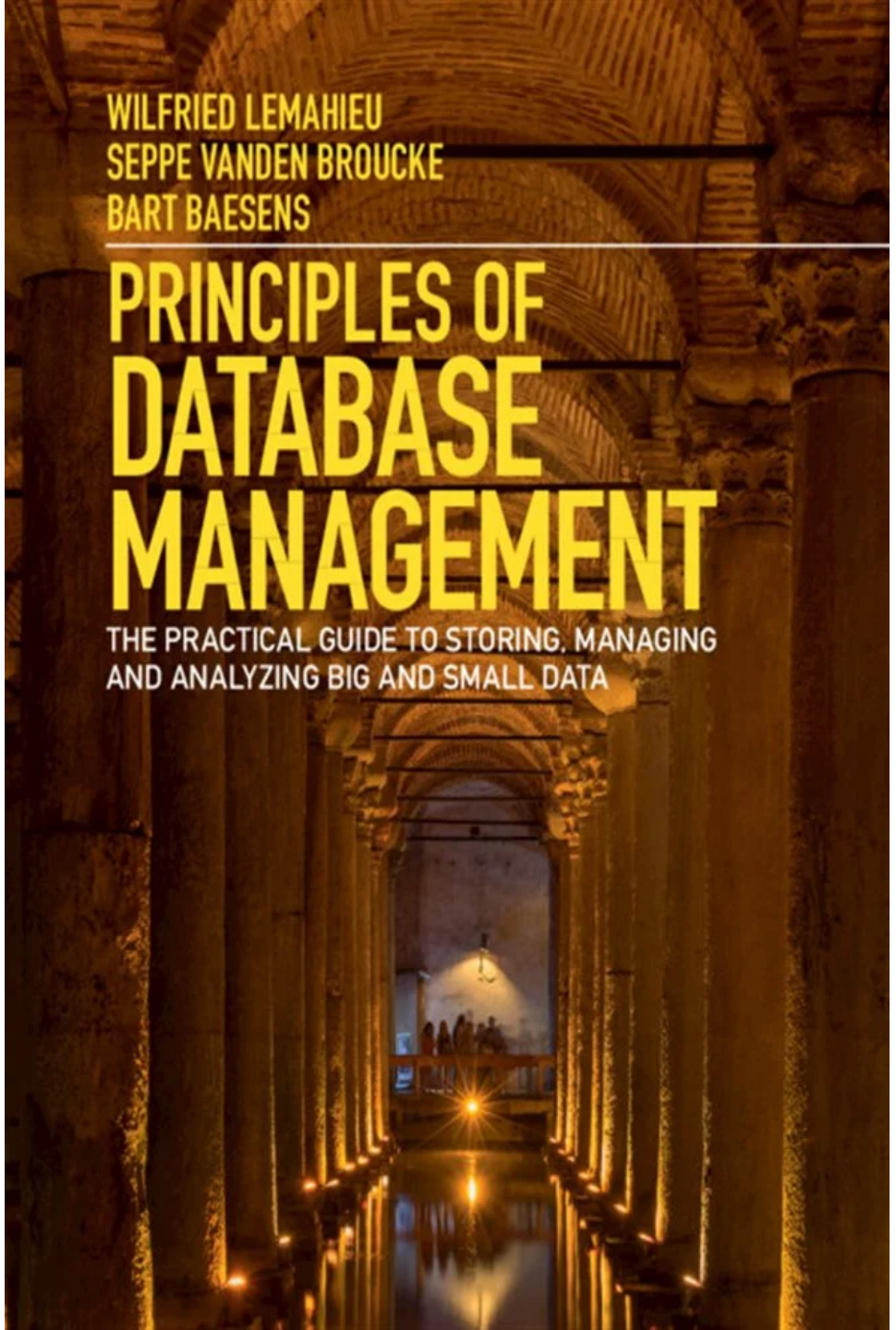
Prof. Dr. Denis M. L. Martins
DCM | FFCLRP | USP

Bibliografia Atualizada

Boa parte do material desta aula foi inspirado no livro:

Principles of database management: the practical guide to storing, managing and Analyzing big and small Data.

Autores: Wilfried Lemahieu, Seppe Vanden Broucke e Bart Baesens.



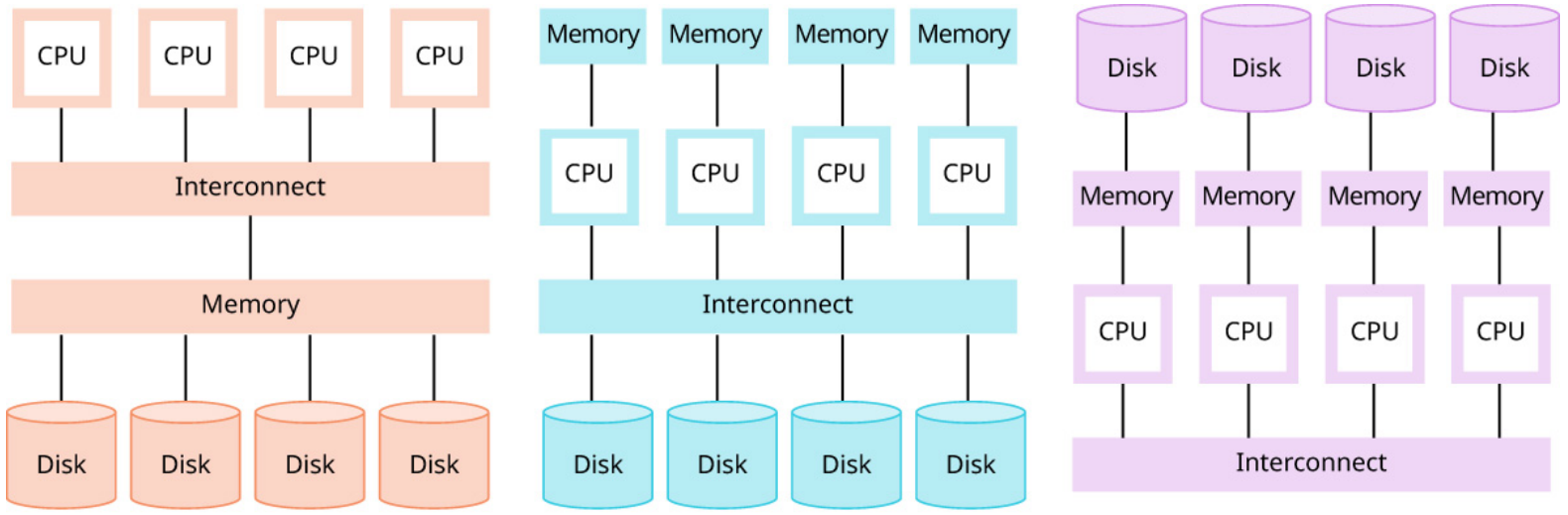
Prof. Dr. Denis M. L. Martins | martins.denis@usp.br

Objetivos de Aprendizagem

- Descrever o conceito de **fragmentação de dados** e diferenciar fragmentação horizontal e fragmentação vertical.
- Compreender que, em consultas distribuídas, o otimizador precisa considerar não apenas custo de processamento, mas também **custo de comunicação entre nós**.
- Explicar o conceito de **escalonamento de transações** e sua importância para controlar a execução concorrente de múltiplas transações.
- Compreender o conceito de **escalonamento serializável**, explicando por que uma execução intercalada pode ser correta quando é equivalente a alguma execução serial.
- Descrever o funcionamento do **Two-Phase Locking Protocol (2PL)** como mecanismo de controle de concorrência baseado em bloqueios.

Introdução aos Bancos de Dados Distribuídos

- Bancos de dados distribuídos distribuem o armazenamento de dados e consultas em múltiplas fontes de dados ou localizações.
- **Objetivo**: aumentar eficiência enquanto fornece uma visão integrada e transparente da distribuição de dados.



Na imagem: Diferentes arquiteturas de distribuídas. Fonte: Introduction to Computer Science by OpenStax

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

Implicações Arquiteturais

Bancos de Dados em Paralelo:

- Foco na distribuição de dados para otimizar desempenho
- Palalelismo: **intra-query** *versus* **inter-query**

Bancos de Dados Federados:

- Nós em arquitetura **shared-nothing**
- Cada nó roda uma instância independente de SGBD com fragmentação horizontal (veja a seguir).

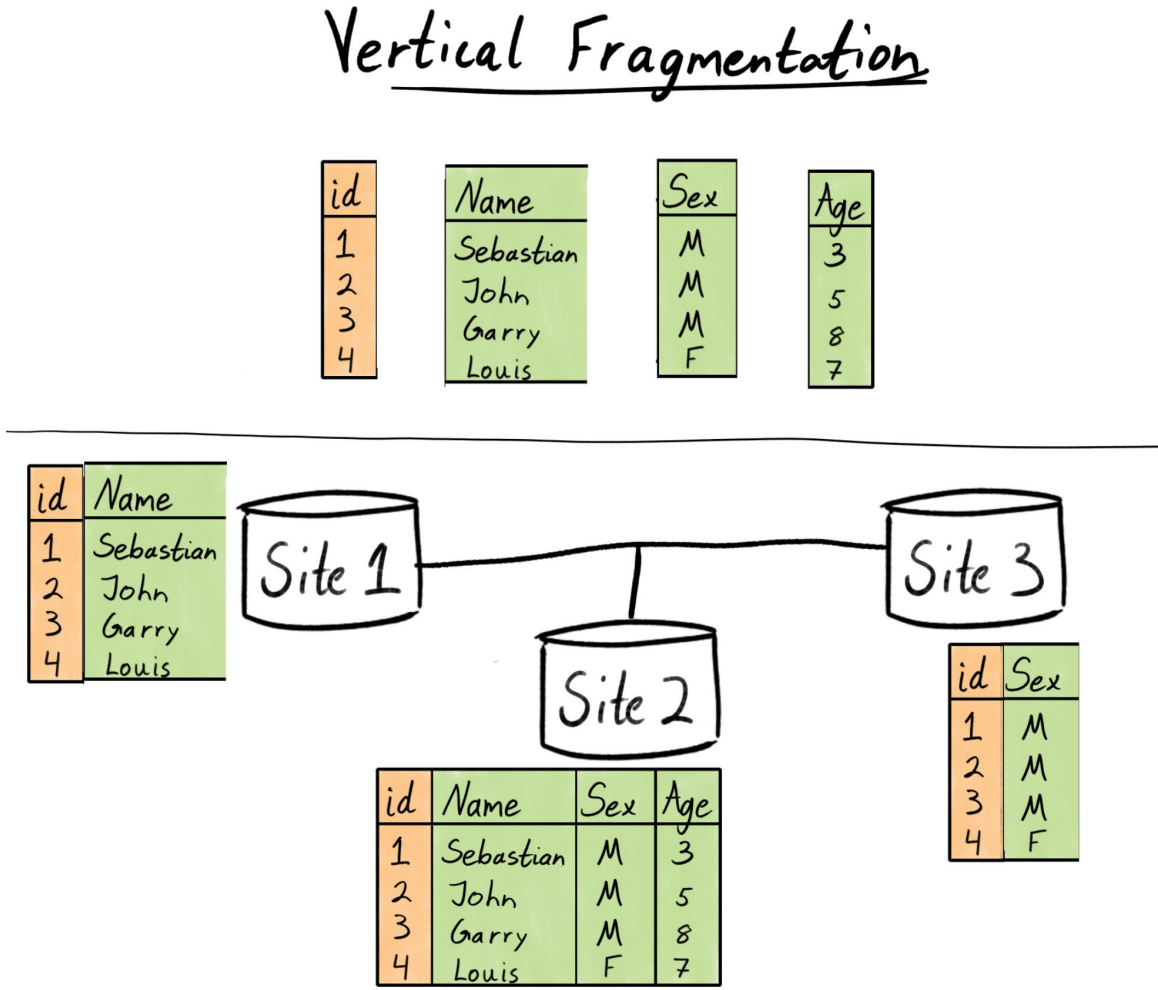
Fragmentação: Particionamento de dados em subconjuntos (fragmentos) com base em metas de desempenho, autonomia local e disponibilidade.

Fragmentação Vertical

O fragmento consiste em um subconjunto das colunas da tabela.

- A visão global é reconstruída com uma operação $JOIN$.
- Útil se apenas alguns atributos de uma tupla são relevantes para um nó específico.

id	Name	Sex	Age
1	Sebastian	M	3
2	John	M	5
3	Garry	M	8
4	Louis	F	7



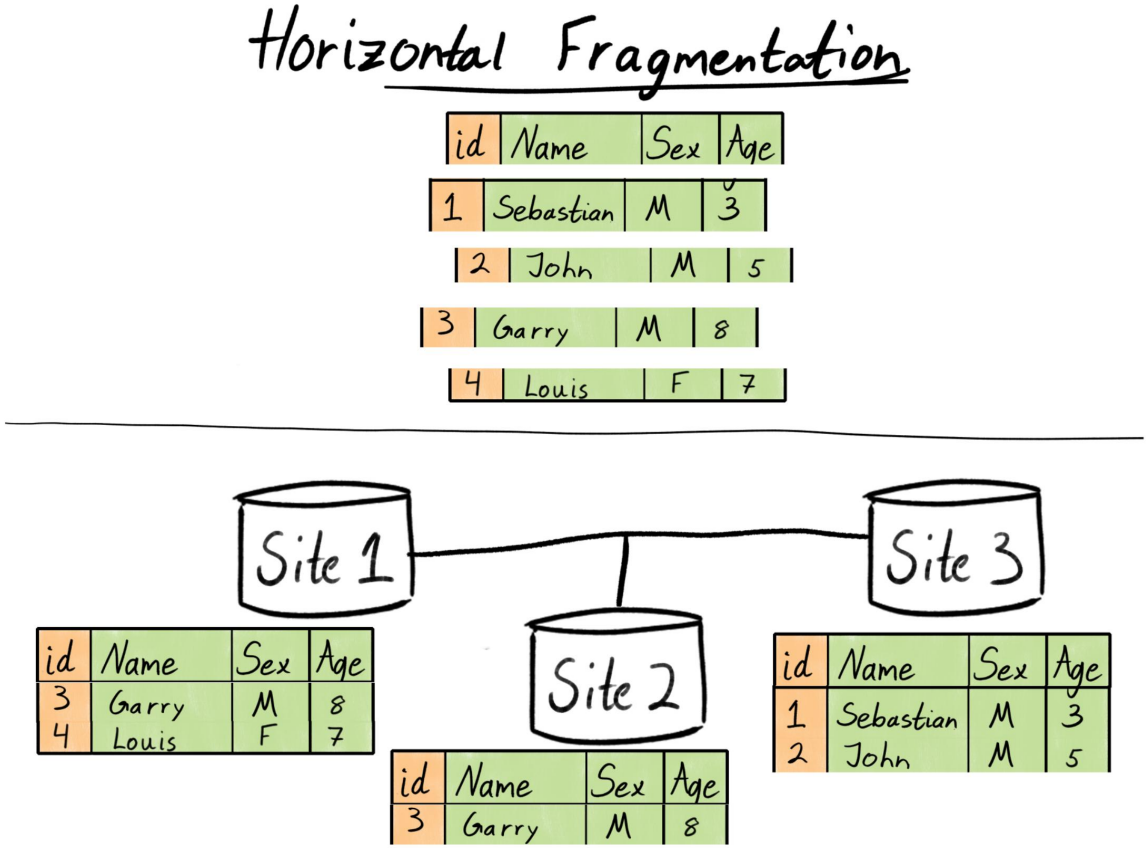
Fonte das imagens: StackOverflow.

Fragmentação Horizontal (Sharding)

O fragmento consiste em linhas que satisfazem um predicado de consulta.

- Comum em bancos de dados NoSQL.
- A visão global é reconstruída com uma operação $UNION$.

id	Name	Sex	Age
1	Sebastian	M	3
2	John	M	5
3	Garry	M	8
4	Louis	F	7



> Fonte das imagens: StackOverflow.

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

Processamento Distribuído de Queries

SUPPLIER (SUPNR, SUPNAME, SUPADDRESS, SUPSTATUS)
PURCHASEORDER (PONR, PODATE, *SUPPLIER*)
PRODUCT (PNR, PNAME, PCOLOR, PWEIGHT, WAREHOUSE, STOCK)
…

Location 1:

SUPPLIER table
SUPNR: 4 bytes
SUPNAME: 30 bytes
Entire row: 84 bytes
Number of rows: 1000

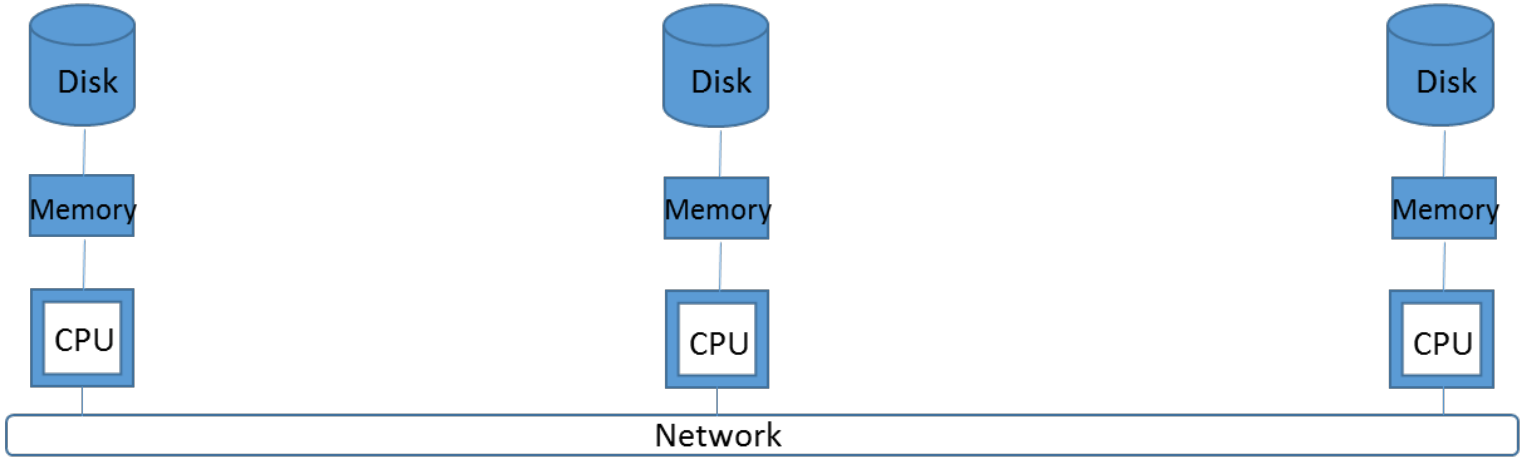
Location 2:

PURCHASEORDER table
PONR: 6 bytes
SUPPLIER: 4 bytes
Entire row: 16 bytes
Number of rows: 3000
On average, there are 200 suppliers with outstanding purchase orders

Location 3:

Query:

```
SELECT PONR, SUPNAME
FROM PURCHASEORDER PO, SUPPLIER S
WHERE PO.SUPPLIER = S.SUPNR
```



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

Processamento Distribuído de Queries (cont.)

- **Estratégia 1** (Push Data to Query): todas as tabelas copiadas para a localização 3, onde a query será processada.
  - Transferência de dados: 1.000×84 + 3.000×16 bytes = 132.000 bytes
- **Estratégia 2** (Push Query to Data): Tabela **SUPPLIER** copiada para localização 2, onde sofrerá **JOIN** com **PURCHASEORDER**. Resultado será enviado para localização 3.
  - Transferência de Dados: $1.000 \times 84 + 3.000 \times 6 + 30$ bytes = 192.000 bytes

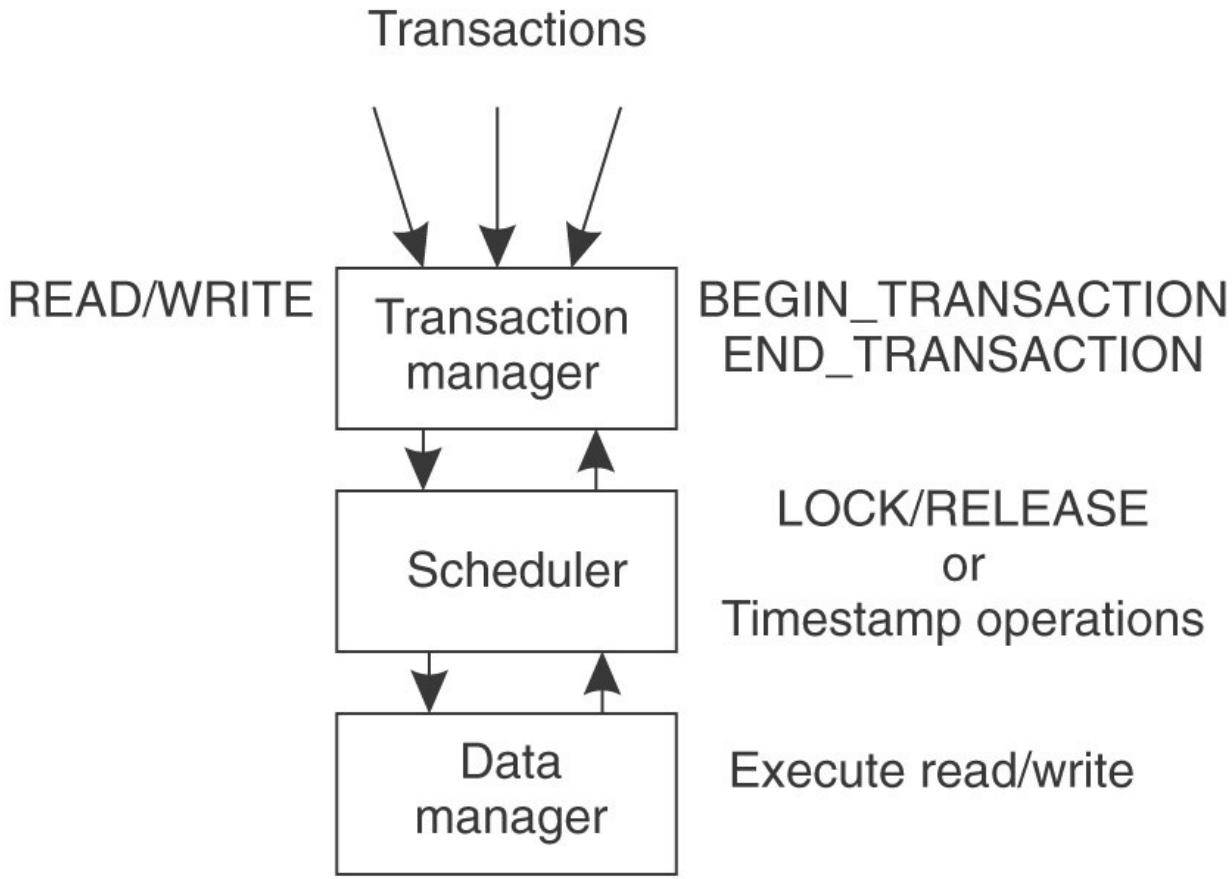
> **Observação**: aspectos de replicação (fragmentos idênticos são alocados a diferentes nós) e transparência (único banco de dados lógico, sendo isolados das complexidades da distribuição) devem ser considerados.

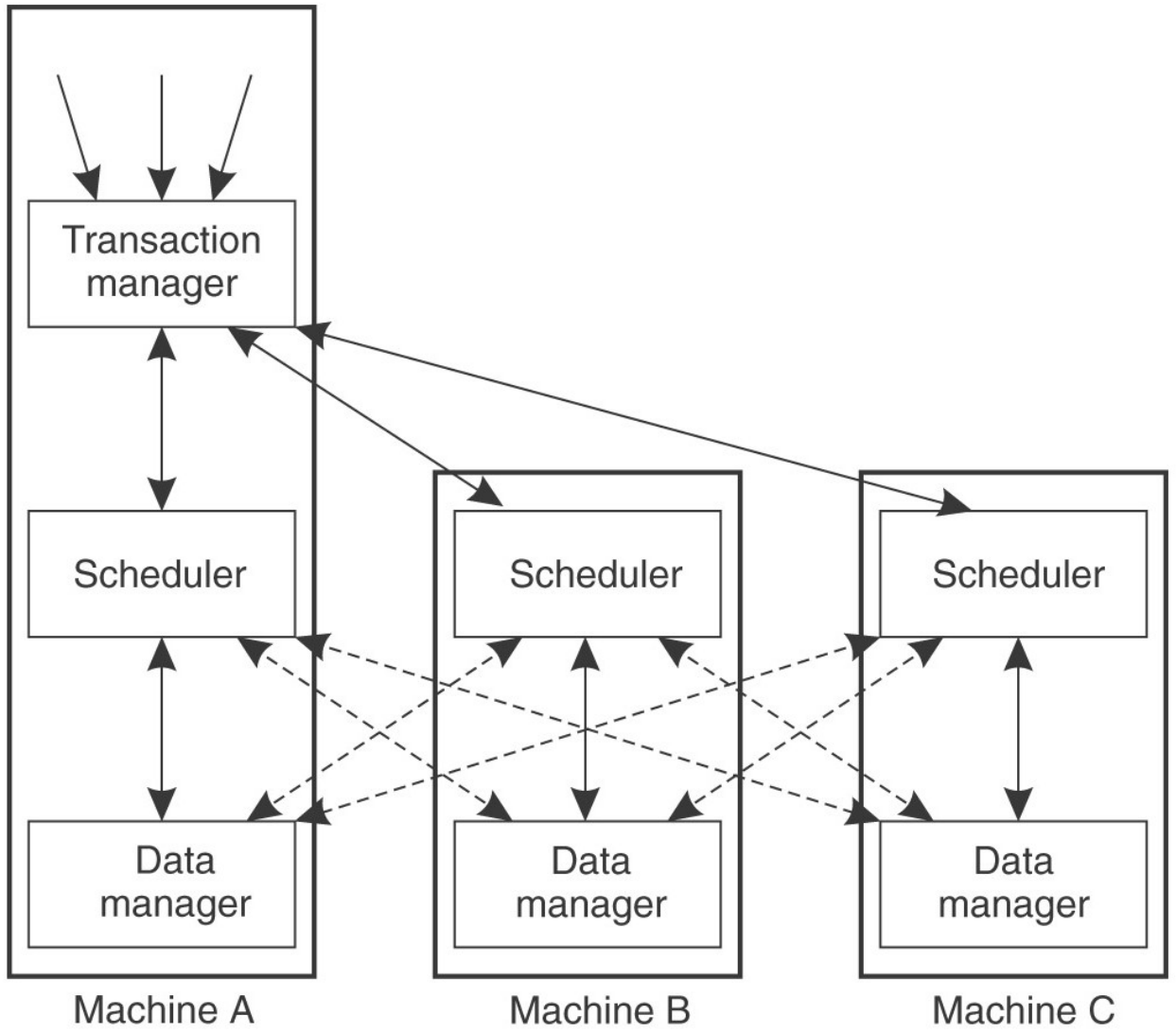
Controle de Concorrência

A execução de mais de uma transação de forma concorrente requer a presença de algum mecanismo para o **controle de concorrência**.

- **Objetivo**: permitir que diversas transações sejam executadas simultaneamente, de modo que o conjunto de dados sendo manipulado permaneça consistente.
- **Gerenciador de Dados**: responsável pela manipulação dos dados propriamente dita.
- **Gerenciador de Escalonamento**: determina quais transações podem solicitar operações para o gerente de dados de modo a preservar o isolamento e a consistência.
  - Principal responsável pelo controle de concorrência.
- **Gerente de Transação**: responsável por garantir a atomicidade das transações.
  - Transforma as primitivas de transação em solicitações de escalonamento para o gerente de escalonamento.

Controle de Concorrência





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

Escalonamentos e Escalonamentos Seriais

Um escalonamento S é um conjunto de n transações e uma ordenação sequencial sobre as instruções dessas transações, tal que:

Para cada transação T que participa de um escalonamento S, e para todas as instruções s_i e s_j pertencentes à mesma transação T, se s_i precede s_j em T, então s_i deve ser escalonada para execução antes de s_j em S.

> O escalonamento preserva a ordem das instruções dentro de cada transação, mas permite uma ordenação arbitrária de instruções entre transações diferentes.

Escalonamentos e Escalonamentos Seriais

- Um escalonamento S é **serial** se todas as instruções s_i da mesma transação T são escalonadas consecutivamente, sem intercalação com instruções de outra transação.
- Escalonamentos seriais impedem a execução paralela de transações.
- Portanto, precisamos de um escalonamento:
  - não serial;
  - correto;
  - equivalente a alguma execução serial.

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

Escalonamentos Serializáveis

- Um escalonamento **serializável** é um escalonamento não serial equivalente a um escalonamento serial.
- Dois escalonamentos S_1 e S_2, com as mesmas transações $T_1, T_2, \ldots, T_n$, são equivalentes se:
  - cada operação de leitura em S_1 lê o mesmo valor produzido pela mesma operação de escrita correspondente em S_2;
  - para cada valor afetado por escrita, a última operação de escrita em S_1 é feita pela mesma transação que realiza a última escrita em S_2.

Serialização: Exemplo

```
openTransaction();
  x = 0;
  x = x + 1;
closeTransaction();
```

```
openTransaction();
  x = 0;
  x = x + 2;
closeTransaction();
```

```
openTransaction();
  x = 0;
  x = x + 3;
closeTransaction();
```

Execução 1:

```
x = 0;  x = x + 1;  x = 0;  x = x + 2;  x = 0;  x = x + 3; // Legal
```

Execução 2:

```
x = 0;   x = 0;  x = x + 1;  x = x + 2;  x = 0;  x = x + 3; // Legal
```

Execução 3:

```
x = 0;  x = 0;  x = x + 1;  x = 0;  x = x + 2;  x = x + 3; // Ilegal
```

> Note que: a Execução 1 é legal e **serializada**. Já a Execução 2 é legal, porém **não serializada**.

Escalonamento Serializável

Um escalonamento **não-serial** que é equivalente a um escalonamento serial.

Equivalência: Dois escalonamentos S_1 e S_2 (com as mesmas transações $T_1, T_2, \ldots, T_n$) são equivalentes se:

- Para cada operação $read_x$ em T_i em S_1: se um valor x, que é lido por esta operação, foi escrito por último por uma operação de $write_x$ de uma transação T_j em S_1, então a mesma operação $read_x$ de T_i em S_2$ deve ler o valor de x como escrito pela operação $write_x$ de T_j em S_2.

- Para cada valor x que é afetado por uma operação de escrita em um dos escalonamentos, a última operação $write_x$ em S_1, executada como parte de uma transação T_i, deve também ser a última operação sobre x em S_2.

> A serializabilidade é uma propriedade usada para verificar se a concorrência preserva a consistência (teste por grafo de precedência).

Escalonadores Otimistas e Pessimistas

O escalonador aplica um protocolo de escalonamento.

Protocolo otimista: Conflitos entre transações simultâneas são considerados excepcionais.

- As operações da transação são escalonadas sem atraso.
- Quando a transação está pronta para confirmar, verifica-se a existência de conflitos.
- Se não houver conflitos, a transação é confirmada; caso contrário, sofre rollback.

Protocolo pessimista: Assume que transações provavelmente irão interferir e causar conflitos.

- A execução das operações é adiada até que o escalonador possa reduzir a chance de conflitos. (Pode reduzir a vazão do sistema)

Escalonadores e Travamento

O objetivo do travamento (**locking**) é garantir que, quando diferentes transações concorrentes tentam acessar o mesmo objeto do banco de dados, o acesso seja concedido apenas de forma que não ocorram conflitos.

- No escalonamento pessimista: travamentos limitam a simultaneidade da execução das transações.
- No escalonamento otimista: travamentos podem ser usados para detectar conflitos durante a execução.

Travamento

- Um travamento é uma variável associada a um objeto do compartilhado.
- O valor dessa variável restringe os tipos de operações que podem ser executadas sobre o objeto naquele momento.
- O **gerenciador de travamentos** é responsável por conceder (**locking**) e liberar (**unlocking**) travamentos, aplicando um protocolo de travamento.
- **Travamento exclusivo** (**x-lock**) ou travamento de escrita: uma única transação adquire o privilégio exclusivo de interagir com aquele objeto. Nenhuma outra transação pode ler ou escrever o objeto.
- **Travamento compartilhado** (**s-lock**) ou travamento de leitura: garante que nenhuma outra transação atualizará o objeto enquanto o travamento estiver ativo.
  - Outras transações também podem manter travamentos compartilhados sobre o mesmo objeto, mas apenas para leitura.

Tipos de Travamento

- Se uma transação deseja atualizar um objeto, é necessário um travamento exclusivo.
- Esse travamento só pode ser adquirido se nenhuma outra transação mantiver qualquer travamento sobre o objeto.

Matriz de compatibilidade

Solicitação / Existente	Sem travamento	Compartilhado	Exclusivo
Compartilhado	Sim	Sim	Não
Exclusivo	Sim	Não	Não

Objetivos dos travamentos

O gerenciador de travamentos implementa um protocolo de travamento:

- conjunto de regras para determinar quais travamentos podem ser concedidos em determinada situação;
- geralmente baseado em uma matriz de compatibilidade.

O gerenciador de travamentos também usa uma tabela de travamentos:

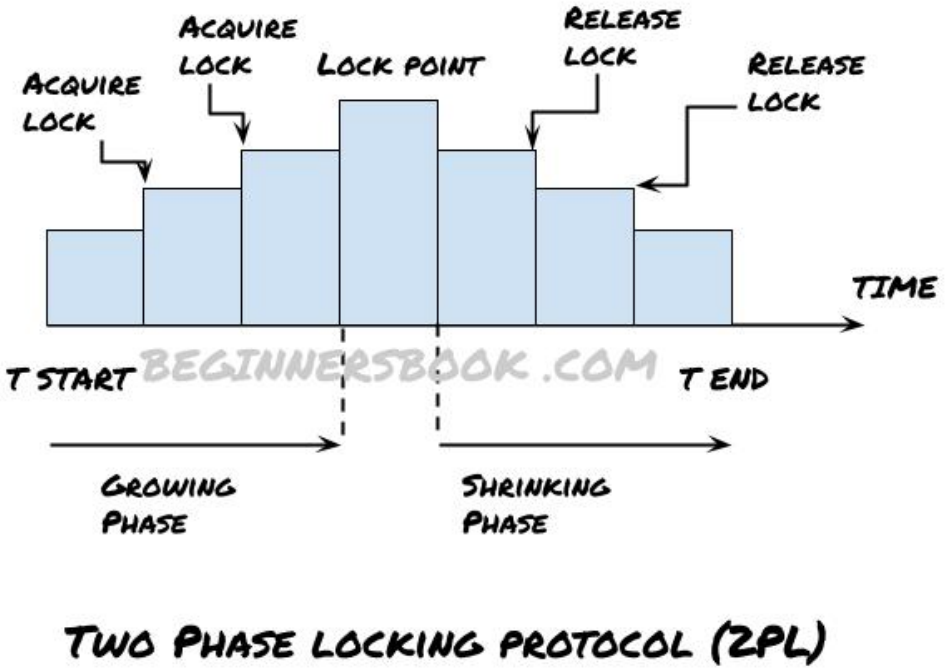
- quais travamentos estão atualmente mantidos por quais transações;
- quais transações aguardam para adquirir certos travamentos;
- entre outras informações.

O gerenciador de travamentos deve garantir justiça no escalonamento, evitando, por exemplo, starvation.

Protocolo de Travamento em Duas Fases

O Two-Phase Locking Protocol (2PL) funciona da seguinte forma:

- Antes de uma transação ler um objeto, deve adquirir um travamento compartilhado sobre ele.
- Antes de atualizar um objeto, deve adquirir um travamento exclusivo sobre ele.
- O gerenciador de travamentos determina se os travamentos solicitados podem ser concedidos, com base na matriz de compatibilidade.
- **Fase de crescimento:** travamentos podem ser adquiridos, mas não liberados;
- **Fase de encolhimento:** travamentos são gradualmente liberados, e nenhum travamento adicional pode ser adquirido.



Fonte da Imagem: beginnersbook.com

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

2PL: Exemplo

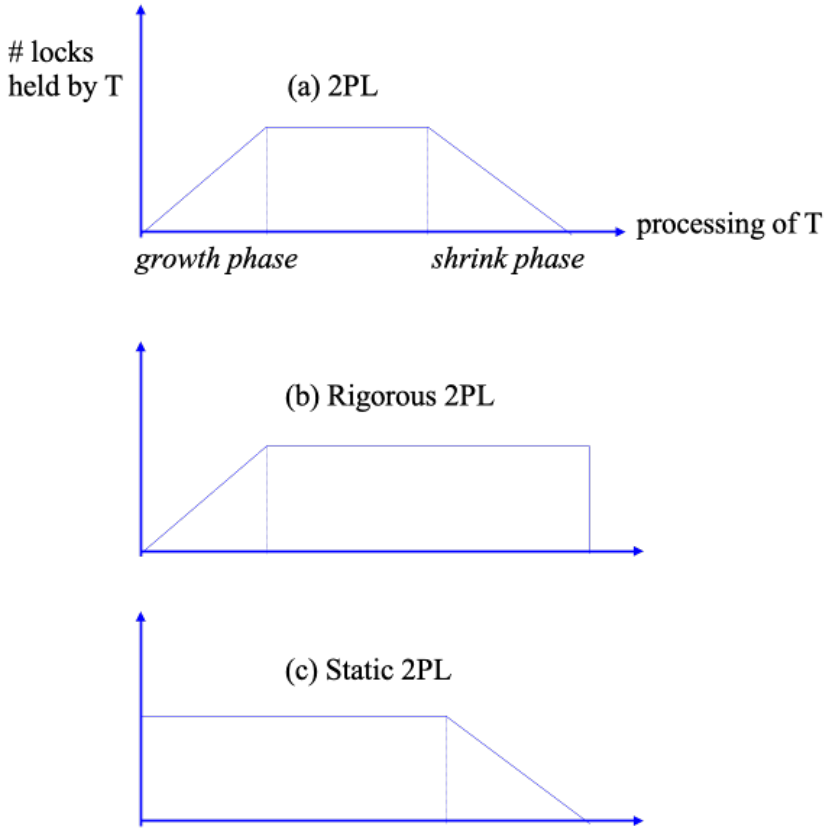
time	T_1	T_2	$amount_x$
t_1		begin transaction	100
t_2	begin transaction	x-lock($amount_x$)	100
t_3	x-lock($amount_x$)	read($amount_x$)	100
t_4	wait	$amount_x = amount_x + 120$	100
t_5	wait	write($amount_x$)	220
t_6	wait	commit	220
t_7	wait	unlock($amount_x$)	220
t_8	read($amount_x$)		220
t_9	$amount_x = amount_x - 50$		220
t_{10}	write($amount_x$)		170
t_{11}	commit		170
t_{12}	unlock($amount_x$)		170

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

2PL: Variantes

- **2PL rigoroso:** a transação mantém todos os travamentos até ser confirmada.
- **2PL estático ou conservador:** a transação adquire todos os travamentos necessários logo no início da transação.

> Depois que uma transação começa a liberar travamentos, ela não pode adquirir novos travamentos.



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

Tratamento de Deadlocks

- A prevenção de deadlocks pode ser obtida com 2PL estático:
  - a transação deve adquirir todos os travamentos no início.
- Detecção e resolução:
  - uso de grafo de espera;
  - nós representam transações ativas;
  - aresta $T_i \rightarrow T_j$ indica que T_i espera um travamento mantido por T_j.
- Um deadlock existe se o grafo de espera contém ciclo.
- A resolução geralmente envolve seleção de uma transação vítima para abortar.

Níveis de Isolamento

- O nível de isolamento oferecido pelo 2PL pode ser muito rigoroso.
- Uma quantidade limitada de interferência pode ser aceitável para aumentar a vazão.
- travamento de longo prazo:
  - mantido por mais tempo, até a confirmação da transação.
- travamento de curto prazo:
  - mantido apenas durante o intervalo necessário para completar a operação associada.
- O uso de travamentos de curto prazo viola uma das regras do 2PL, mas pode melhorar a vazão.

Granularidade de travamentos

O objeto para travamento pode ser:

- tupla;
- coluna;
- tabela;
- bloco de disco;
- entre outros.
- Há um trade-off entre sobrecarga de travamento e vazão das transações:
  - Quanto menor a granularidade, mais precisa é a reserva e maior é o grau de concorrência obtido normalmente requer o uso de um número maior de reservas, além de ser uma solução mais complexa.

Conclusão

Bancos de dados distribuídos permitem armazenar e processar dados em múltiplos nós, favorecendo escalabilidade, disponibilidade e autonomia local.

- A **fragmentação** divide os dados em partes menores, que podem ser distribuídas conforme critérios de desempenho, localização dos usuários e características das consultas.

Em bancos de dados distribuídos, o controle de concorrência é mais complexo porque dados, transações e bloqueios podem estar espalhados por diferentes nós.

- Um escalonamento não serial pode ser eficiente, desde que seja **serializável**, isto é, equivalente a alguma execução serial correta.
- O protocolo **Two-Phase Locking (2PL)** é uma técnica clássica para garantir escalonamentos serializáveis por meio de bloqueios.

Material Adicional: Transactions with Two-Phase Locking (aula do Prof. Andy Pavlo)

Dúvidas e Discussão

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

Questões

Questões para Discussão

1. Explique o que é um banco de dados distribuído e diferencie-o de um banco de dados centralizado. Em sua resposta, discuta pelo menos duas razões pelas quais uma organização poderia optar por distribuir seus dados entre diferentes nós.

2. Discuta como a fragmentação pode melhorar o desempenho de um banco de dados distribuído. Em sua resposta, explique também uma possível desvantagem da fragmentação.

3. Diferencie escalonamento serial e escalonamento não serial. Explique por que um escalonamento não serial pode ser desejável.

4. Explique o conceito de escalonamento serializável. Por que a serializabilidade é importante para o controle de concorrência?

5. Explique o funcionamento do protocolo Two-Phase Locking, destacando suas duas fases principais.

Questão para Reflexão

Uma empresa possui filiais em São Paulo, Recife e Porto Alegre. Cada filial acessa majoritariamente os dados de seus próprios clientes, mas a matriz precisa executar relatórios nacionais. Que estratégia de fragmentação poderia ser adotada? Quais seriam os benefícios e os desafios dessa decisão?