

Técnicas de Controle de Concorrência

- Pessimistas

- supõem que sempre ocorre interferência entre transações e garantem a serializabilidade enquanto a transação está ativa
- técnicas
 - bloqueio (*locking*)
 - *timestamp*

- Otimistas

- supõem que quase nunca ocorre interferência entre transações e verificam a serializabilidade somente ao final de uma transação
- técnica
 - validação

Técnicas Baseadas em Bloqueio

- Técnicas mais utilizadas pelos SGBDs
- Princípio de funcionamento
 - controle de operações *read(X)* e *write(X)* e postergação (através de bloqueio) da execução de algumas dessas operações de modo a evitar conflito
- Todo dado possui um **status de bloqueio**
 - **liberado** (*Unlocked - U*)
 - com **bloqueio compartilhado** (*Shared lock - S*)
 - com **bloqueio exclusivo** (*eXclusive lock - X*)

Modos de Bloqueio

- **Bloqueio Compartilhado (S)**
 - solicitado por uma transação que deseja realizar leitura de um dado D
 - várias transações podem manter esse bloqueio sobre D
- **Bloqueio Exclusivo (X)**
 - solicitado por uma transação que deseja realizar atualização ou leitura+atualização de um dado D
 - uma única transação pode manter esse bloqueio sobre D
- **Matriz de Compatibilidade de Bloqueios**

	S	X
S	verdadeiro	falso
X	falso	falso

- **Informações de bloqueio são mantidas no DD**
<ID-dado, status-bloqueio, ID-transação>

Operações de Bloqueio na História

- O *Scheduler* gerencia bloqueios através da invocação automática de operações de bloqueio conforme a operação que a transação deseja realizar em um dado
- Operações
 - $ls(D)$: solicitação de bloqueio compartilhado sobre D
 - $lx(D)$: solicitação de bloqueio exclusivo sobre D
 - $u(D)$: libera o bloqueio atual sobre D

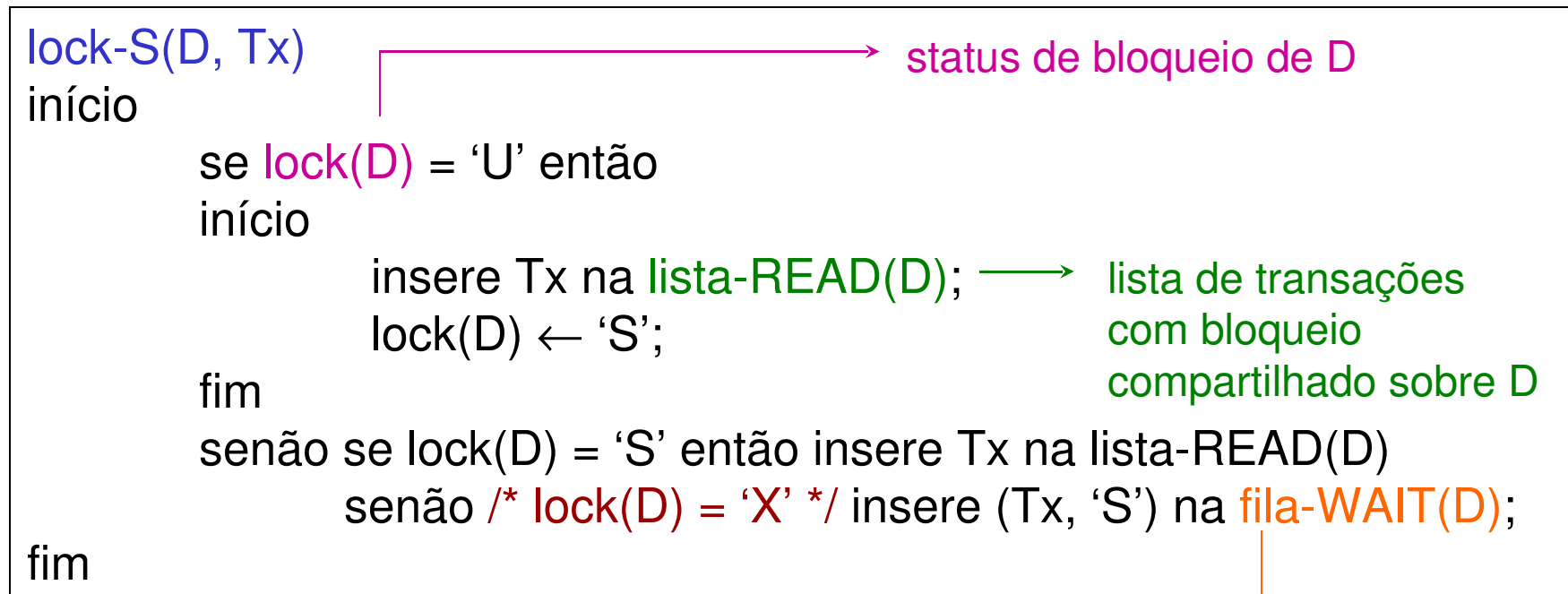
Exemplo de História com Bloqueios

T1	T2
lock-S(Y)	
read(Y)	
unlock(Y)	
	lock-S(X)
	lock-X(Y)
	read(X)
	read(Y)
	unlock(X)
	write(Y)
	unlock(Y)
	commit()
lock-X(X)	
read(X)	
write(X)	
unlock(X)	
commit()	

H = ls1(Y) r1(Y) u1(Y) ls2(X) lx2(Y)
r2(X) r2(Y) u2(X) w2(Y) u2(Y) c2
lx1(X) r1(X) w1(X) u1(X) c1

Implementação das Operações

- Solicitação de bloqueio compartilhado



Obs.: supor que os métodos de inclusão/exclusão de elementos nas EDs automaticamente alocam/desalocam a ED caso ela não exista/se torne vazia

fila de transações aguardando a liberação de um bloqueio conflitante sobre D

Exercício 2

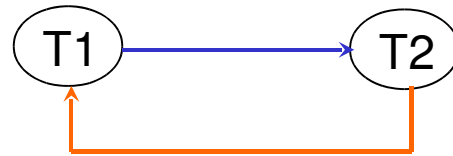
1. Propor algoritmos de alto nível para as operações:
 - a) $\text{lock-X}(D, Tx)$
 - b) $\text{unlock}(D, Tx)$ (considere que essa operação também pode retirar transações da fila-WAIT e solicitar novos bloqueios)
2. O algoritmo $\text{lock-S}(D, Tx)$ apresentado anteriormente pode gerar *starvation* (espera indefinida de Tx , se Tx solicitou $\text{lock-X}(D, Tx)$ e lista- $\text{READ}(D)$ nunca fica vazia!). Modifique os algoritmos das operações de bloqueio (aqueles que forem necessários) de modo a evitar *starvation*

Uso de Bloqueios “S” e “X”

- Não garantem escalonamentos serializáveis
- Exemplo

$H_{N-SR} = ls1(Y) \text{ r1}(Y) \text{ u1}(Y) \text{ ls2}(X) \text{ r2}(X) \text{ u2}(X) \text{ lx2}(Y) \text{ r2}(Y)$

$lx1(X) \text{ r1}(X) \text{ w1}(X) \text{ w2}(Y) \text{ u2}(Y) \text{ c2} \text{ u1}(X) \text{ c1}$

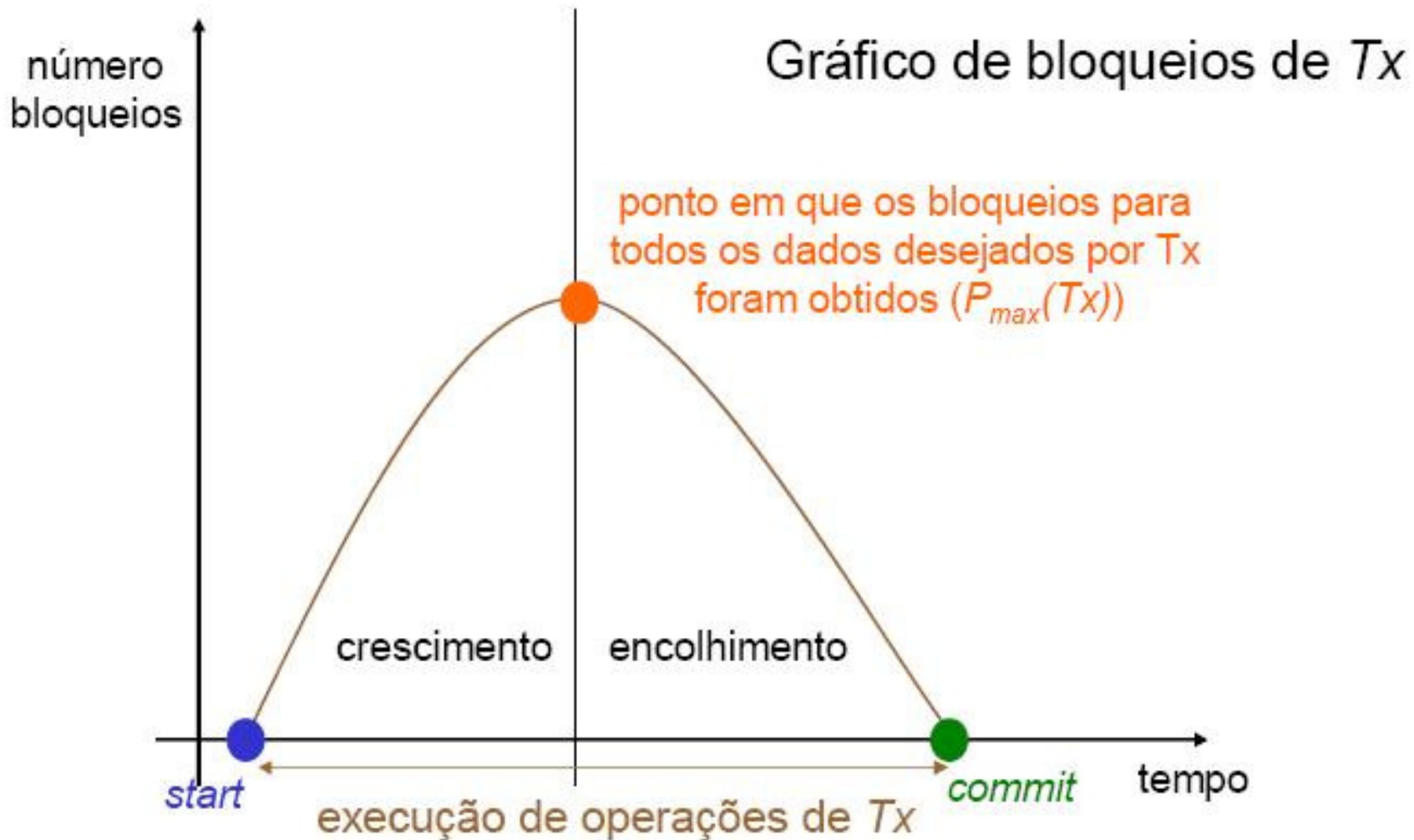


- Necessita-se de uma técnica mais rigorosa de bloqueio para garantir a serializabilidade
 - técnica mais utilizada
 - bloqueio de duas fases (*two-phase locking* – 2PL)

Bloqueio de 2 Fases – 2PL

- Premissa
 - *“para toda transação Tx, todas as operações de bloqueio de dados feitas por Tx precedem a primeira operação de desbloqueio feita por Tx”*
- Protocolo de duas fases
 1. Fase de expansão ou crescimento
 - Tx pode obter bloqueios, mas não pode liberar nenhum bloqueio
 1. Fase de retrocesso ou encolhimento
 - Tx pode liberar bloqueios, mas não pode obter nenhum bloqueio

Scheduler 2 PL – Funcionamento

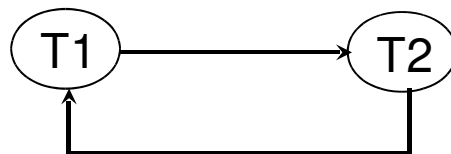


Scheduler 2PL - Exemplo

- T1: r(Y) w(Y) w(Z)
- T2: r(X) r(Y) w(Y) r(Z) w(Z)

- Contra-Exemplo

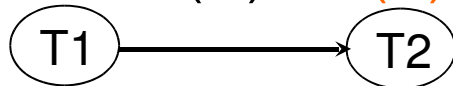
$H_{N-2PL} =$ lx1(Y) r1(Y) ls2(X) r2(X) u2(X) w1(Y) u1(Y) lx2(Y)
 r2(Y) w2(Y) u2(Y) lx2(Z) r2(Z) w2(Z) c2 lx1(Z) w1(Z)
 u1(Z) c1



não é 2PL!
 não garantiu SR!

- Exemplo

$H_{2PL} =$ ls2(X) r2(X) lx1(Y) r1(Y) lx1(Z) w1(Y) u1(Y) lx2(Y)
 r2(Y) w1(Z) u1(Z) c1 w2(Y) lx2(Z) u2(X) u2(Y) w2(Z)
 u2(Z) c2



$P_{max}(T1)$

$P_{max}(T2)$

é SR!

Scheduler 2PL - Crítica

- Vantagem
 - técnica que sempre garante escalonamentos SR sem a necessidade de se construir um grafo de dependência para teste!
 - se T_x alcança P_{max} , T_x não sofre interferência de outra transação T_y , pois se T_y deseja um dado de T_x em uma operação que poderia gerar conflito com T_x , T_y tem que esperar (teremos aresta $T_x \rightarrow T_y$, mas não teremos aresta $T_y \rightarrow T_x$!)
 - depois que T_x liberar os seus dados, não precisará mais deles, ou seja, T_x não interferirá nas operações feitas futuramente nestes dados por T_y (também evita aresta $T_y \rightarrow T_x$!)

Scheduler 2PL - Crítica

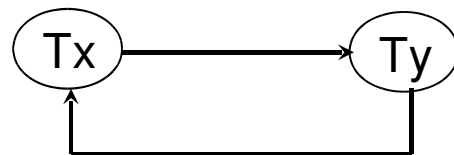
- Desvantagens
 - limita a concorrência
 - um dado pode permanecer bloqueado por T_x muito tempo até que T_x adquira bloqueios em todos os outros dados que deseja
 - 2PL básico (técnica apresentada anteriormente) não garante escalonamentos
 - livres de *deadlock*
 - T_x espera pela liberação de um dado bloqueado por T_y de forma conflitante e vice-versa
 - adequados à recuperação pelo *recovery*

Exercício 3

1. Apresente um início de escalonamento concorrente 2PL básico que recaia em uma situação de *deadlock*
2. Apresente um escalonamento concorrente 2PL básico que não seja recuperável
 - **lembrete**: um escalonamento é recuperável se T_x nunca executa *commit* antes de T_y , caso T_x tenha lido dados atualizados por T_y

Deadlock (Impasse) de Transações

- Ocorrência de *deadlock*
 - T_y está na Fila-WAIT(D1) de um dado D1 bloqueado por T_x
 - T_x está na Fila-WAIT(D2) de um dado D2 bloqueado por T_y
- Pode ser descoberto através de um **grafo de espera de transações**
 - se o grafo é **cíclico** existe *deadlock*!

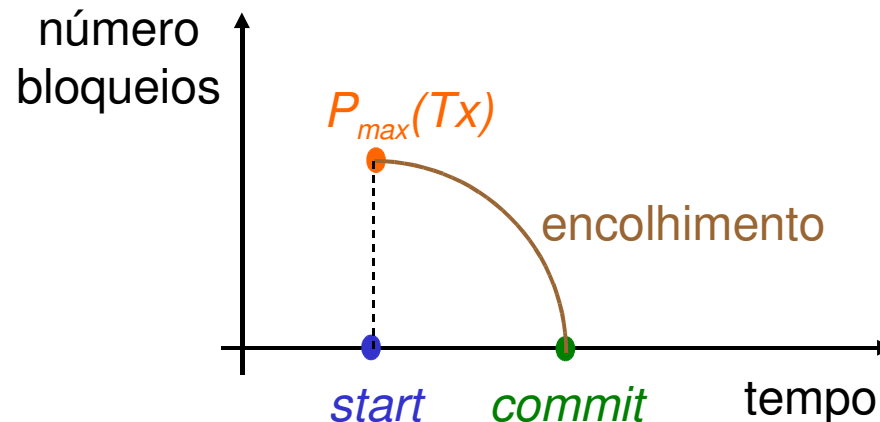


Tratamento de *Deadlock*

- Protocolos de Prevenção
 - abordagens pessimistas
 - *deadlocks* ocorrem com frequência!
 - impõem um *overhead* no processamento de transações
 - controles adicionais para evitar *deadlock*
 - tipos de protocolos pessimistas
 - técnica de bloqueio 2PL conservador
 - técnicas baseadas em *timestamp* (*wait-die* e *wound-wait*)
 - técnica de *espera-cautelosa* (*cautious-waiting*)
 - uso de *timeout*
 - se tempo de espera de $T_x > \text{timeout} \Rightarrow \text{abort}(T_x)$

Scheduler 2PL Conservador

- T_x deve bloquear **todos** os dados que deseja antes de iniciar qualquer operação
 - caso não seja possível bloquear todos os dados, nenhum bloqueio é feito e T_x entra em espera para tentar novamente



- **vantagem**
 - uma vez adquiridos todos os seus bloqueios, T_x não entra em *deadlock* durante a sua execução
- **desvantagem**
 - espera pela aquisição de todos os bloqueios!

Técnicas Baseadas em *Timestamp*

- *Timestamp*

- rótulo de tempo associado à T_x ($TS(T_x)$)

tempo de start de T_x



- Técnicas

- consideram que T_x deseja um dado bloqueado por outra transação T_y

- Técnica 1: esperar-ou-morrer (*wait-die*)

- se $TS(T_x) < TS(T_y)$ então T_x espera

- senão início

- $abort(T_x)$

- $start(T_x)$ com o mesmo TS

- fim

Técnicas Baseadas em *Timestamp*

- Técnicas (cont.)
 - Técnica 2: *ferir-ou-esperar* (*wound-wait*)
 - se $TS(T_x) < TS(T_y)$ então
 - início
 - $abort(T_y)$
 - $start(T_y)$ com o mesmo TS
 - fim
 - senão T_x espera
 - vantagem das técnicas
 - evitam *starvation* (espera indefinida) de uma T_x
 - quanto mais antiga for T_x , maior a sua prioridade
 - desvantagem das técnicas
 - muitos abortos podem ser provocados, sem nunca ocorrer um *deadlock*

Técnica *Cautious-Waiting*

- Princípio de Funcionamento

- se T_x deseja D e D está bloqueado por T_y
então

- se T_y não está em alguma Fila-WAIT

- então T_x espera

- senão início

- $abort(T_x)$

- $start(T_x)$

- fim

- Vantagem

- se T_y já está em espera, T_x é abortada para evitar um possível ciclo de espera

- Desvantagem

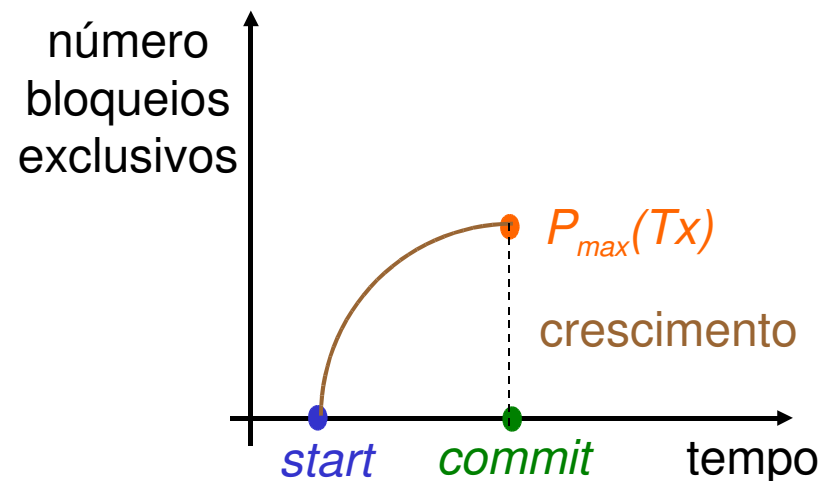
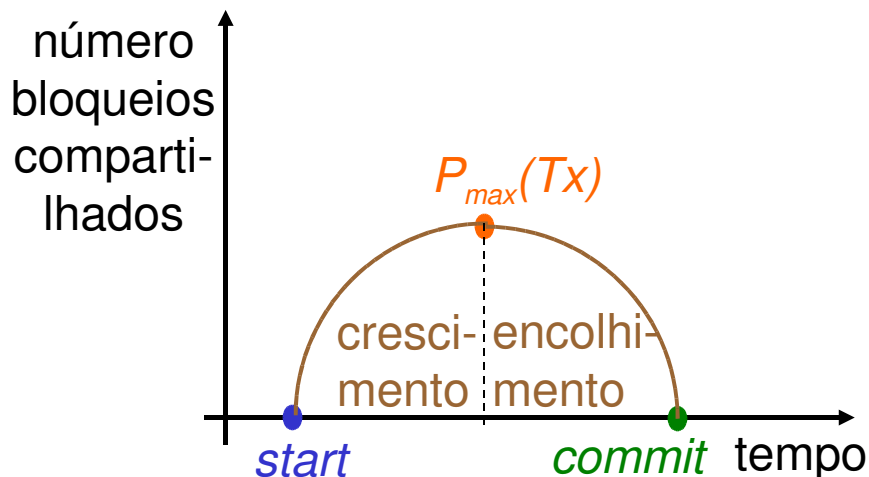
- a mesma das técnicas baseadas em *timestamp*

Tratamento de *Deadlock*

- Protocolos de Detecção
 - abordagens otimistas
 - *deadlocks* não ocorrem com freqüência!
 - são tratados quando ocorrem
 - mantém-se um grafo de espera de transações
 - se há *deadlock*, seleciona-se uma transação vítima *Tx* através de um ou mais critérios
 - quanto tempo *Tx* está em processamento
 - quantos itens de dado *Tx* já leu/escreveu
 - quantos itens de dado *Tx* ainda precisa ler/escrever
 - quantas outras transações serão afetadas pelo *abort(Tx)*

Outras Técnicas de Bloqueio 2PL

- *Scheduler 2PL Conservador ou Estático*
 - evita *deadlock*, porém *Tx* pode esperar muito para executar
- *Scheduler 2PL Estrito* (muito usado pelos SGBDs)
 - *Tx* só libera seus bloqueios exclusivos após executar *commit* ou *abort*



Outras Técnicas de Bloqueio 2PL

- *Scheduler* 2PL Estrito

- vantagem: garante escalonamentos estritos
- desvantagem: não está livre de *deadlocks*

- *Scheduler* 2PL (Estrito) Rigoroso

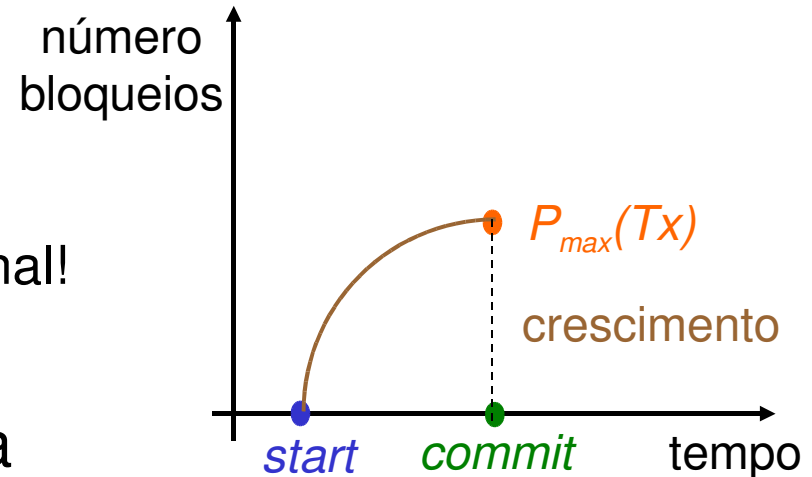
- *Tx* só libera seus bloqueios após executar *commit* ou *abort*

- vantagem

- menos *overhead* para *Tx*
 - *Tx* libera tudo apenas no final!

- desvantagem

- limita mais a concorrência



Exercícios 4

1. Apresente exemplos de escalonamentos concorrentes 2PL conservador, 2PL estrito e 2PL rigoroso para as seguintes transações:

T1: r(Y) w(Y) w(Z)

T2: r(X) r(T) w(T)

T3: r(Z) w(Z)

T4: r(X) w(X)

2. Apresente uma situação de *deadlock* em um escalonamento concorrente 2PL estrito

Scheduler Baseado em *Timestamp*

- Técnica na qual toda transação T_x possui uma marca *timestamp* ($TS(T_x)$)
- Princípio de funcionamento (TS-Básico)
 - “no acesso a um item de dado D por operações conflitantes, a ordem desse acesso deve ser equivalente à ordem de TS das transações envolvidas”
 - garante escalonamentos serializáveis através da ordenação de operações conflitantes de acordo com os TS s das transações envolvidas
 - cada item de dado X possui um registro TS ($R-TS(X)$)

$\langle ID-dado, TS-Read, TS-Write \rangle$

TS da transação mais recente
que leu o dado

TS da transação mais recente
que atualizou o dado

Técnica TS-Básico - Exemplo

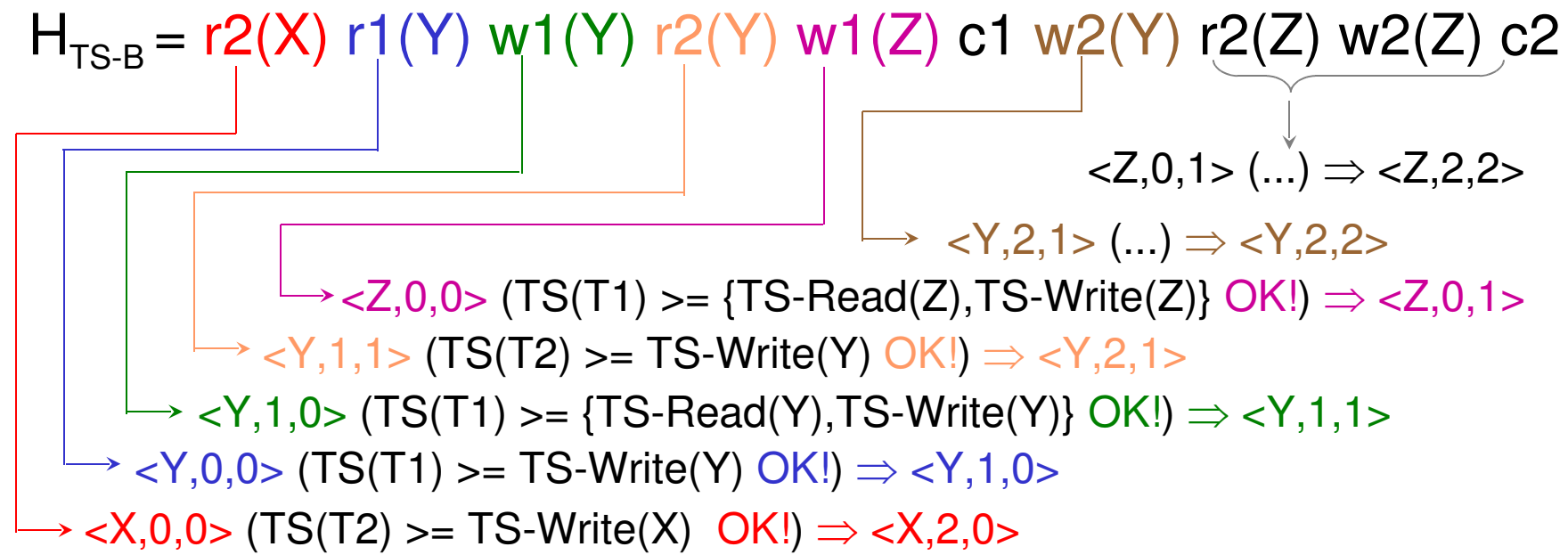
• T1: r(Y) w(Y) w(Z) → TS(T1) = 1

• T2: r(X) r(Y) w(Y) r(Z) w(Z) → TS(T2) = 2

• Registros iniciais de TS de X, Y e Z:

- $\langle X, 0, 0 \rangle$; $\langle Y, 0, 0 \rangle$; $\langle Z, 0, 0 \rangle$

• Exemplo de escalonamento serializável por TS



Algoritmo TS-Básico

TS-Básico(Tx, dado, operação)

```
início
  se operação = 'READ' então
    se TS(Tx) < R-TS(dado).TS-Write então
      início      abort(Tx);
                  restart(Tx) com novo TS;
    fim
    senão início executar read(dado);
              se R-TS(dado).TS-Read < TS(Tx) então

/* mantém TS-Read sempre atualizado com a transação mais nova que leu,
   para garantir sempre um controle correto da ordenação */
              R-TS(dado).TS-Read ← TS(Tx);
            fim
  senão início /* operação = 'WRITE' */
            se TS(Tx) < R-TS(dado).TS-Read OU
              TS(Tx) < R-TS(dado).TS-Write então
            início      abort(Tx);
                        restart(Tx) com novo TS;
            fim
            senão início      executar write(dado);
                            R-TS(dado).TS-Write ← TS(Tx);
            fim
  fim
fim
```

Técnica TS-Básico

- Vantagens

- técnica simples para garantia de serializabilidade (não requer bloqueios)
- não há *deadlock* (não há espera)

- Desvantagens

- gera muitos abortos de transações
 - passíveis de ocorrência quando há conflito
- pode gerar abortos em cascata
 - não gera escalonamentos adequados ao *recovery*

- Para minimizar essas desvantagens

- técnica de *timestamp* estrito (TS-Estrito)

Técnica TS-Estrito

- Gera escalonamentos serializáveis e **estritos**
 - passíveis de *recovery* em caso de falha
- Funcionamento
 - baseado no TS-básico com a seguinte diferença
 - “se T_x deseja $read(D)$ ou $write(D)$ e $TS(T_x) > R-TS(D).TS-Write$, então T_x **espera** pelo *commit* ou pelo *abort* da transação T_y cujo $R-TS(D).TS-Write = TS(T_y)$ ”
 - exige *fila-WAIT(D)*
 - não há risco de *deadlock*
 - nunca há ciclo pois somente transações mais novas esperam pelo *commit/abort* de transações mais antigas
 - *overhead* no processamento devido à espera

Técnica TS-Estrito - Exemplo

- T1: r(X) w(X) w(Z) → TS(T1) = 1
- T2: r(X) w(X) w(Y) → TS(T2) = 2
- Exemplo de escalonamento TS-Estrito

$H_{TS-E} = r1(X) w1(X) r2(X) w1(Z) c1 r2(X) w2(X) w2(Y) c2$

T2 espera por T1, pois
 $TS(T2) > R-TS(X).TS-write$
($r2(X)$ não é executado
e T2 é colocada na
Fila-WAIT(X))

T1 já *commitou*!
T2 pode executar
agora $r2(X)$
(tira-se T2 da
fila-WAIT(X))

Exercícios 5

1. Considerando a técnica TS-Básico, verifique se alguma transação abaixo é desfeita e em que ponto
 - a) $H1 = r1(a) \ r2(a) \ r3(a) \ c1 \ c2 \ c3$
 - b) $H2 = r1(a) \ w2(a) \ r1(a) \ c1 \ c2$
 - c) $H3 = r1(a) \ r1(b) \ r2(a) \ r2(b) \ w2(a) \ w2(b) \ c1 \ c2$
 - d) $H4 = r1(a) \ r1(b) \ r2(a) \ w2(a) \ w1(b) \ c1 \ c2$
 - e) $H5 = r2(a) \ w2(a) \ w1(a) \ r2(a) \ c1 \ c2$
 - f) $H6 = r2(a) \ w2(a) \ r1(b) \ r1(c) \ w1(c) \ w2(b) \ c1 \ c2$
2. Apresente o algoritmo *TS-Estrito*(*Tx*, *dado*, *operação*). Há algo a considerar nos algoritmos *Commit*(*Tx*) e *Abort*(*Tx*)?
3. Apresente um exemplo e um contra-exemplo de um escalonamento TS-Estrito para as seguintes transações:
T1: $r(Y) \ w(Y) \ w(Z)$
T2: $r(X) \ r(T) \ w(T)$
T3: $r(Z) \ w(Z)$
T4: $r(X) \ w(X)$