# Programação Concorrente e Paralela

## Memória Transacional

Noemi Rodriguez

2016

# Motivação

- dificuldades de se trabalhar com memória compartilhada e locks
  - deadlocks e serialização
- sincronização *wait-free*: soluções específicas para cada estrutura de dados
  - listas, filas, tabelas de hash, ...

PUC RIO

DEPARTAMENTO DE INFORMÁTICA PUC-RIO

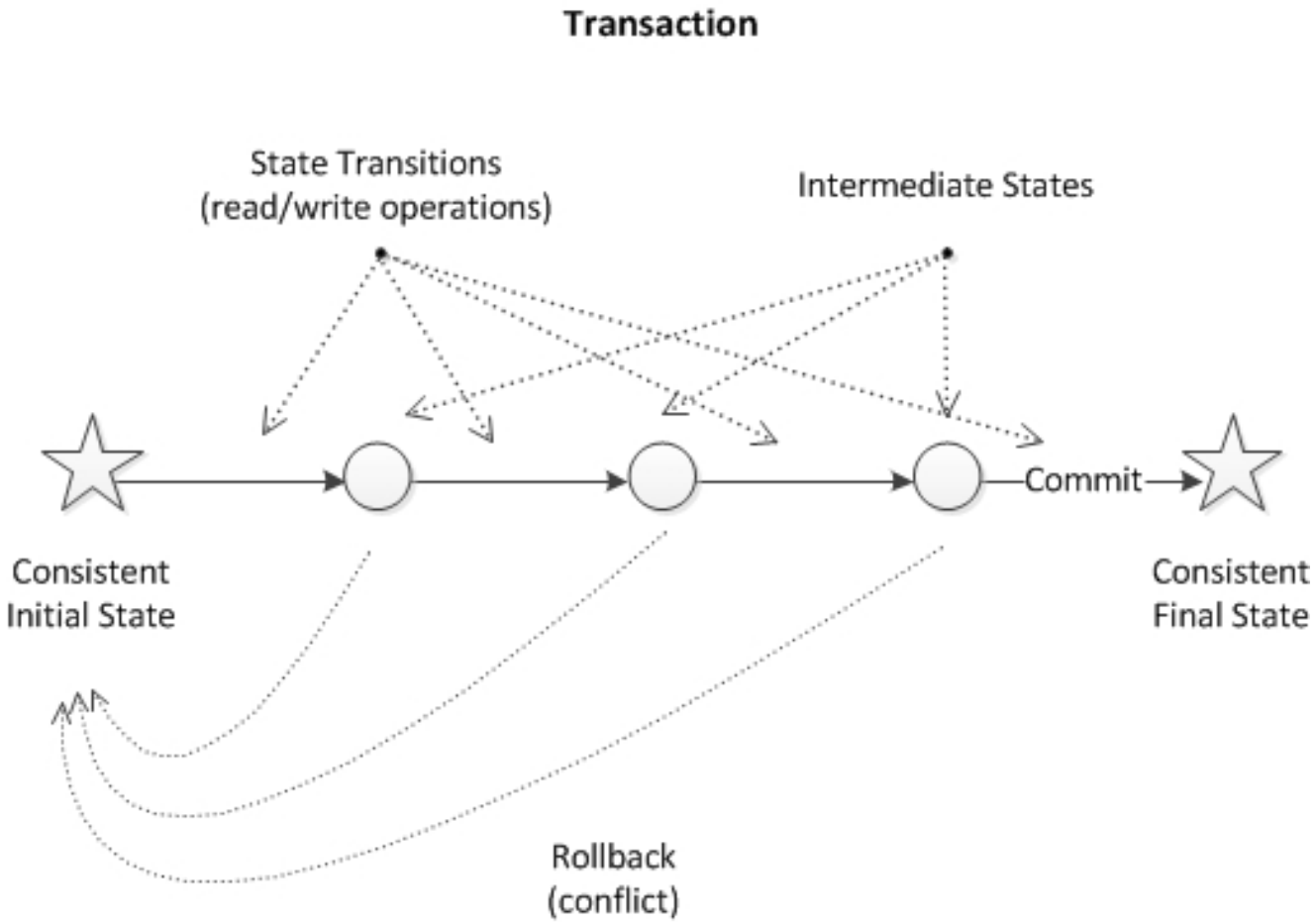
# Memória Transacional

- uso de construções similares às de transações de bancos de dados
- extensão de Java (descrita por Herlihy&Shavit):

```
void transfer (account from, int amount) {
  atomic {
    if (from.balance() >= amount) {
      from.withdraw(amount);
      this.deposit(amount);
    }
}
```

- propriedades ACID
  - atomicidade, consistência, isolamento, durabilidade

# Transação – Conceito



# Implementação

- propostas de implementação por hardware e software

# Implementação

- um lock "guarda-chuva" pedido em todos os trechos atômicos seria uma implementação conceitualmente correta?
  - e como compreensão semântica?

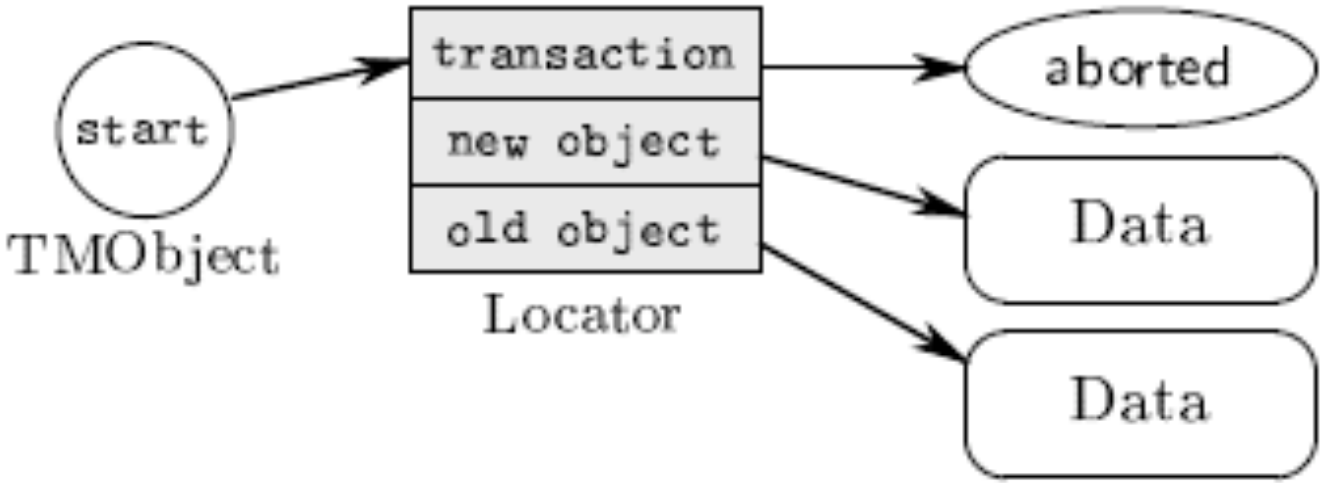
# Implementação

- uso de logging como em BDs
- uso de marcações e objetos intermediários

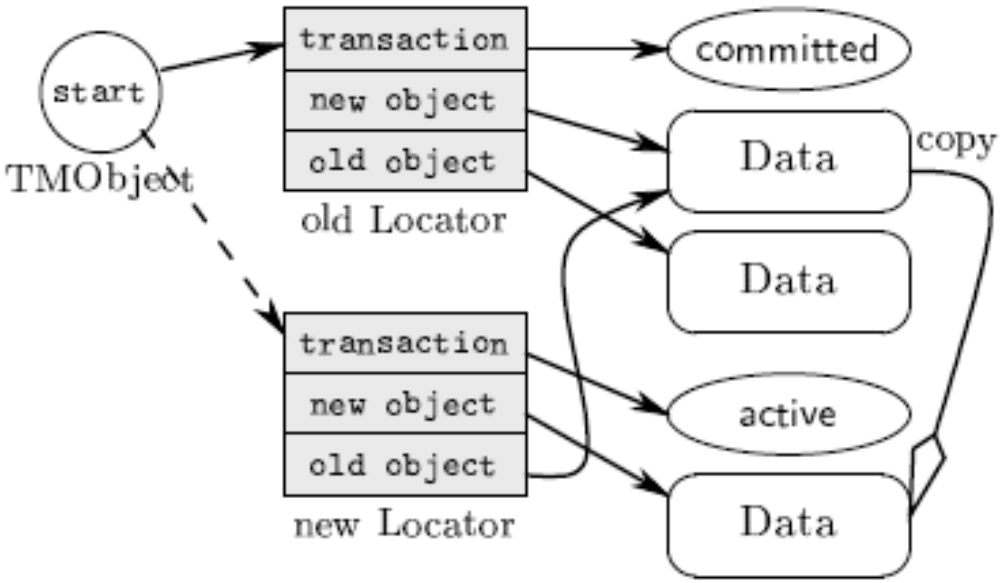
- políticas otimistas e pessimistas

# Objetos Atômicos – Indireção

- descritor de objeto atômico:



# Objetos Atômicos – Indireção




- instruções do tipo *compareAndSet* usadas para substituição final

# Objetos Atômicos – Exemplo

```
public boolean insert(int v)
                        throws TMException {
  List newList = new List(v);
  TMObject newNode = new TMObject(newList);
  TMThread thread =
           (TMThread)Thread.currentThread();
  while (true) {
    thread.beginTransaction();
    boolean result = true;
    try {
      List prevList =
        (List)this.first.open(WRITE);
      List currList =
        (List)prevList.next.open(WRITE);
      while (currList.value < v) {
        prevList = currList;
        currList =
          (List)currList.next.open(WRITE);
      }
      if (currList.value == v) {
        result = false;
      } else {
        result = true;
        newList.next = prevList.next;
        prevList.next = newNode;
      }
    } catch (Denied d){}
    if (thread.commitTransaction())
      return result;
  }
}
```

# Memória transacional – Implementações

- Haskell
- ...
- C++11

- especificação de transações (*Draft Specification of Transactional Memory Constructs for C++*)
- construções têm comportamento bem definido *apenas para programas sem condições de corrida*.
- novas palavras chave `__transaction_atomic`, `__transaction_relaxed` e `__transaction_cancel`
  - também `__transaction_safe` e `__transaction_unsafe` (atributos de funções)

# C++11 – atomic transactions

## `__transaction_atomic`

- isolamento rígido
- atomicidade: ou tem efeito por completo ou não tem efeito algum
- diversas restrições sobre código protegido

# C++11 – relaxed transactions

**`__transaction_relaxed`**

- executa sem observar alterações realizadas por outras transações durante sua execução
- podem executar ações *irrevogáveis*

# Exemplo em C++11

```
void sched_inc_lpcount( void ) {
  pthread_mutex_lock( &mutex_lp_count );
  lpcount++;
  pthread_mutex_unlock( &mutex_lp_count );
}
```

**Figure 1.** Increasing the active Lua process count with locks.

```
void sched_inc_lpcount( void ) {
  __transaction_relaxed {
    lpcount++;
  }
}
```

**Figure 2.** Increasing the active Lua process count with a transaction.

# Exemplo em C++11

```
void sched_wait( void ) {
  pthread_mutex_lock( &mutex_lp_count );
  if( lpcount != 0 ) {
    pthread_cond_wait( &cond_no_active_lp,
                       &mutex_lp_count );
  }
  pthread_mutex_unlock( &mutex_lp_count );
}
```

**Figure 3.** Waiting until there are no more active Lua processes with locks.

```
void sched_wait( void ) {
  while ( __transaction_relaxed( lpcount )
          != 0 );
}
```

**Figure 4.** Waiting until there are no more active Lua processes with a transaction.

- 2 operações estão em conflito se uma delas modifica uma posição de memória e a outra acessa ou modifica a mesma posição
- a execução de um programa contém uma *condição de corrida* se contém operações conflitantes em threads distintos, pelo menos uma das quais não é uma operação atômica, e nenhuma *acontece antes* da outra

**data-race free**

Um programa é dito *livre de condições de corrida* se nenhuma de suas execuções contém uma condição de corrida

# Memória Transacional – condições de espera – proposta

- retry: aborta transação e volta a tentar quando *algum dos valores lidos no trecho tiver sido alterado*

```
public void enq (T x) {
  atomic {
    if (count == items.length)
      retry;
    items [tail] = x;
    if (++tail == items.length)
      tail = 0;
    ++count;
  }
}
```

# Memória Transacional – Dificuldades (1)

- implementações por hardware: não adotadas
- implementações por software:
  - custo alto
  - retry: problemas semelhantes aos dos monitores com testes implícitos:
    - quando tentar de novo?
  - dificuldades com entrada e saída

# Memória Transacional – Dificuldades (1)

- API é apropriada?
  - o mundo é transacional...?
  - capacidade de rollback não necessariamente ligada a concorrência
  - ações irreversíveis como comunicação com outra thread ou E/S
  - rollback em caso de falhas: como detectar motivos da falha?
- biblioteca .NET

> Back in January Joe Duffy, Microsoft's best known researcher on parallel and concurrent programming, cited four reasons why he becomes disillusioned with STM in his Brief Retrospective on Transactional Memory.
> (http://www.infoq.com/news/2010/05/STM-Dropped)

# Referências


- M. Herlihy, N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- Maurice Herlihy, Victor Luchango, Mark Moir, and III William N. Scherer. Software transactional memory for dynamic-sized data structures. PODC 2003: 92-101, Jul 2003.
- Ali-Reza Adl-Tabatabai and others (eds). Draft Specification of Transactional Language Constructs for C++. 02/2012.
- Hans-J. Boehm. Transactional memory should be an implementation technique, not a programming interface. In Proceedings of the First USENIX conference on Hot topics in parallelism (HotPar'09). 2009.
- A. Skyrme and N. Rodriguez. From locks to transactional memory: Lessons learned from porting a real-world application. In the 8th ACM SIGPLAN Workshop on Transactional Computing, 2013.