Implementation and Optimization of Thread-Local Variables for a Race-Free Java Dialect

Yi Zhang, Clark Verbrugge

McGill University

Structure

- Overview & Motivation
- Design
- Implementation
- Experiments
- Conclusion & Future Work

Overview

- implement and optimize thread local access
- a new semantic for Java
- race-free version of Java

Motivation

data-race free property

- Complexity in racy program
  - hard to validate the optimization
  - many optimizations are prohibited

Motivation

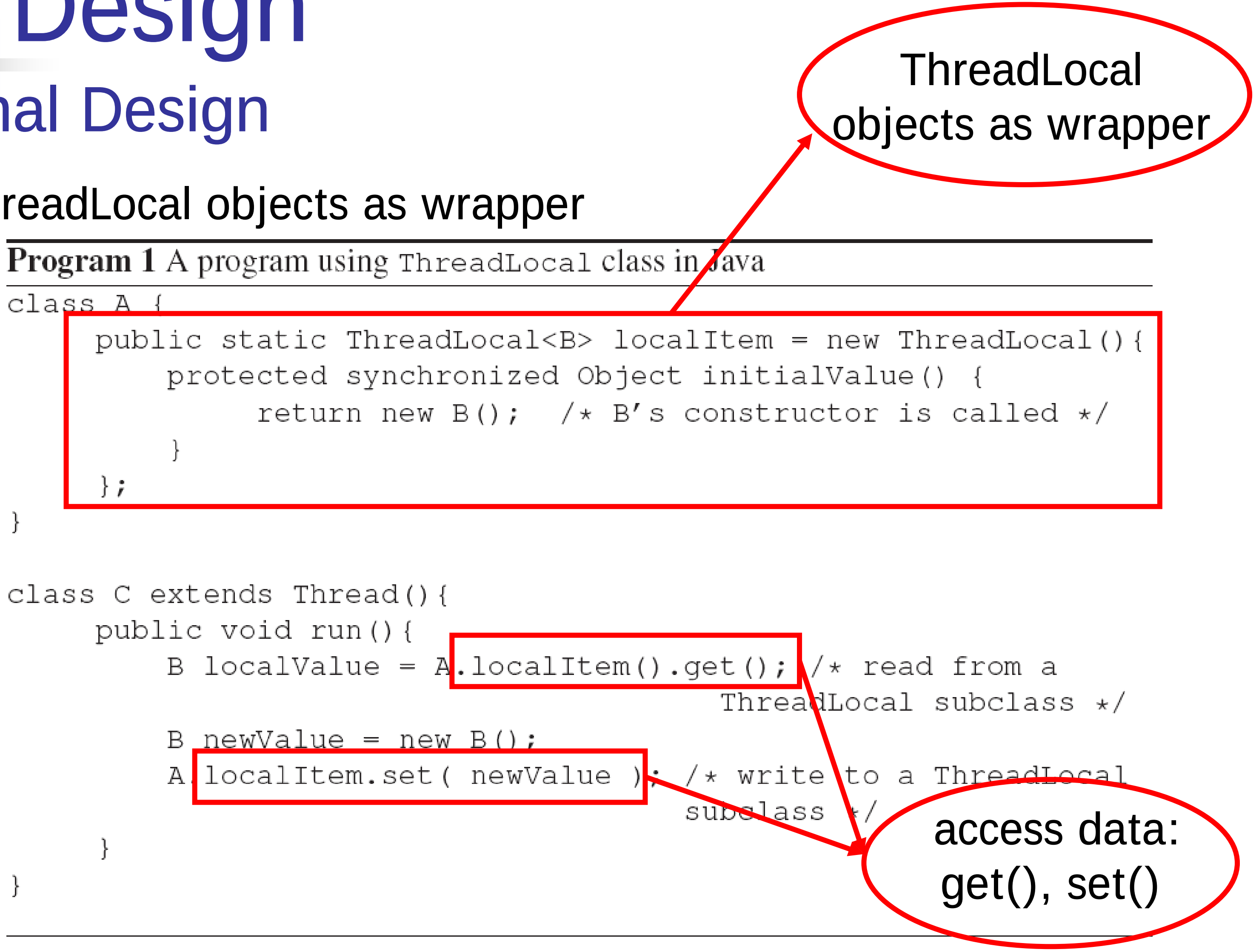
thread-local access

- data are thread local by default and use shared directives for shared data
- ThreadLocal class in Java API

Design

Original Design

- ThreadLocal objects as wrapper


ThreadLocal objects as wrapper
access data: get(), set()

Program 1 A program using `ThreadLocal` class in Java

```
class A {
    public static ThreadLocal<B> localItem = new ThreadLocal(){
        protected synchronized Object initialValue()  {
            return new B();   /* B's constructor is called */
        }
    };
}

class C extends Thread(){
    public void run(){
        B localValue = A.localItem().get(); /* read from a
                                       ThreadLocal subclass */
        B newValue = new B();
        A.localItem.set( newValue ); /* write to a ThreadLocal
                                    subclass */
    }
}
```

Design

Original Design

- each thread holds a ThreadLocalMap
  - First, get map from thread
  - Second, <ThreadLocal as key, value>

Design

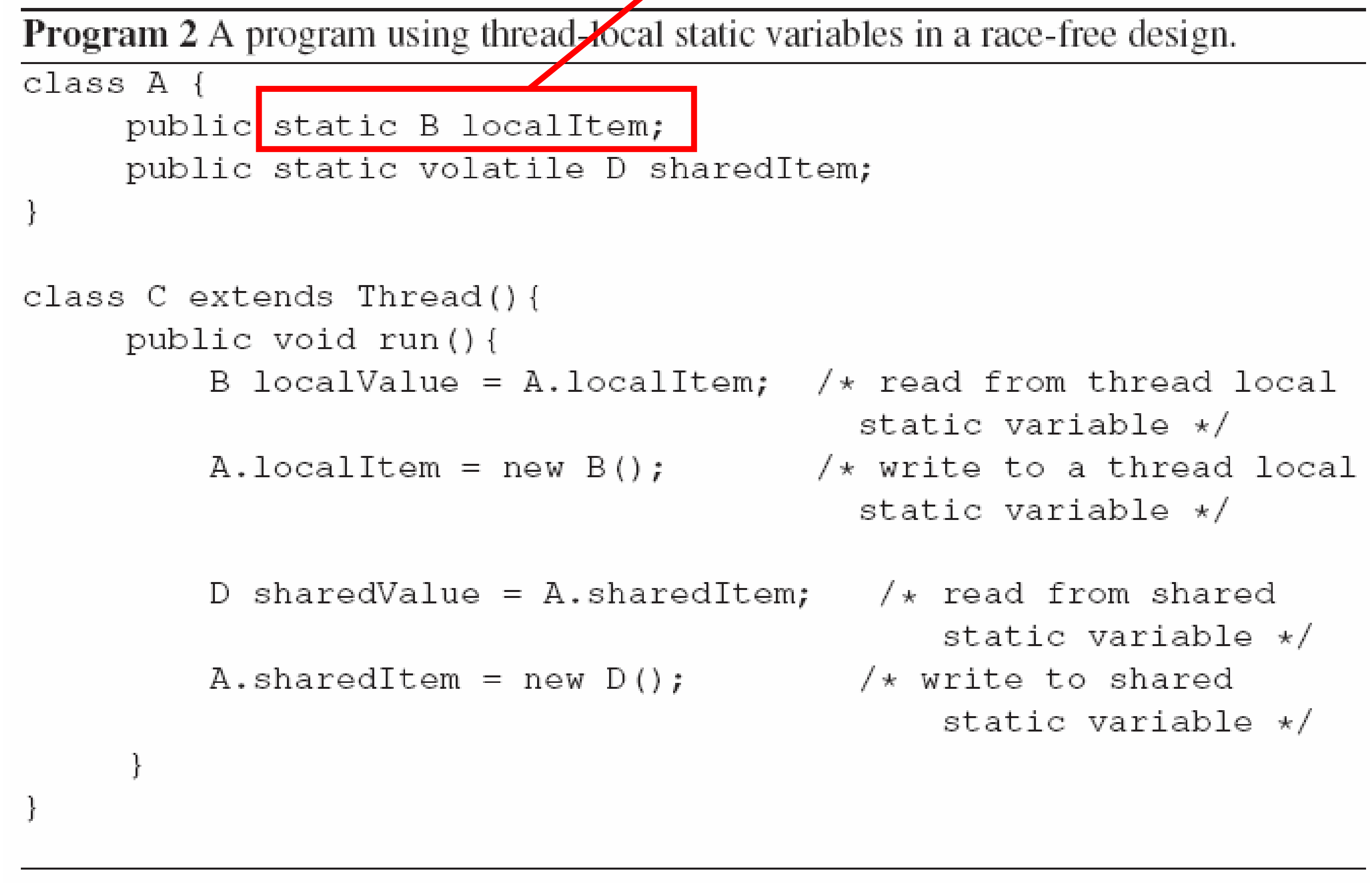
Our Design

- thread-local the default option
- use “volatile” to specify the shared data

Design

Our Design

thread-local the default option



Program 2 A program using thread-local static variables in a race-free design.

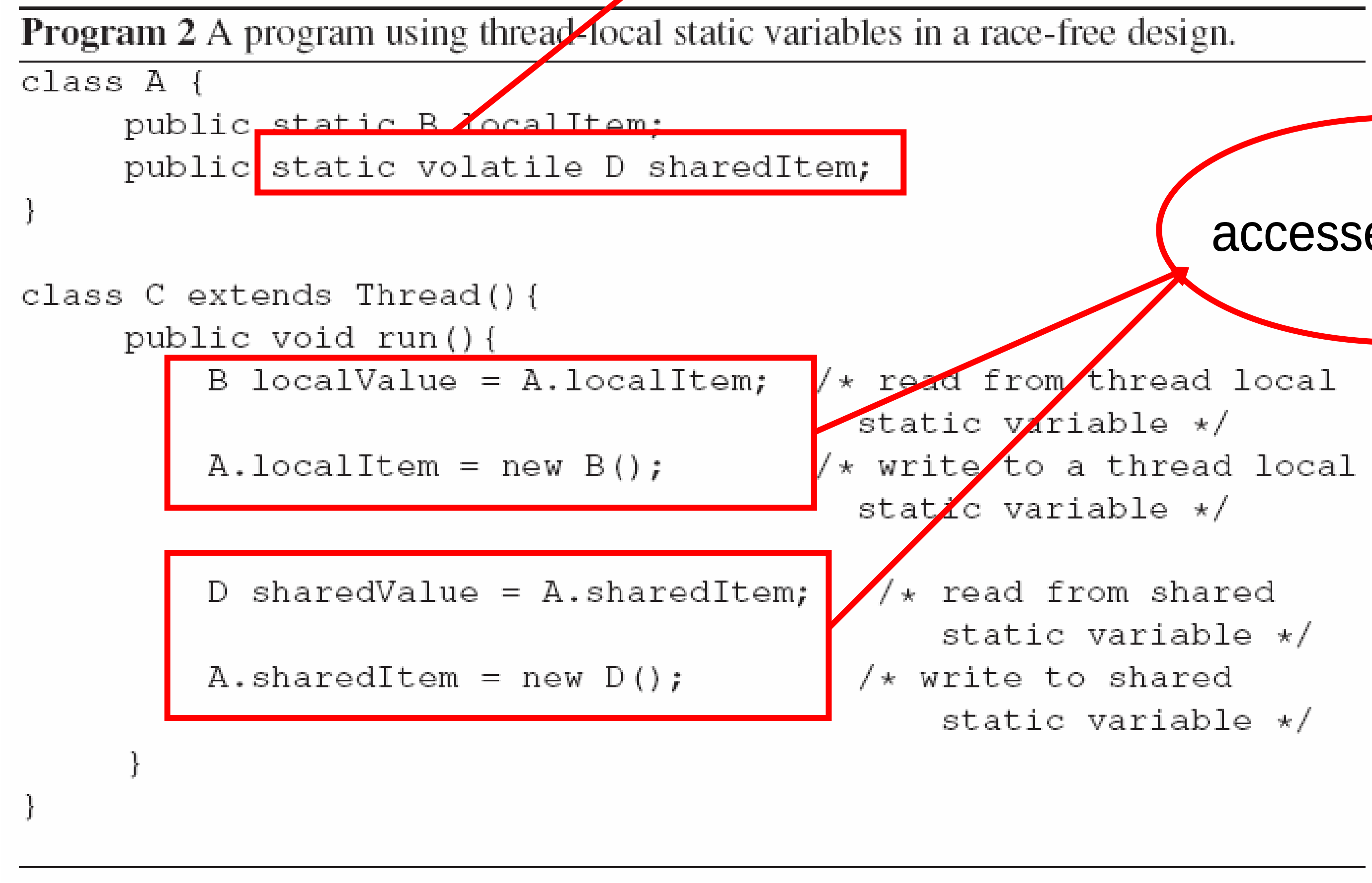
```
class A {
     public static B localItem;
     public static volatile D sharedItem;
}

class C extends Thread(){
     public void run(){
          B localValue = A.localItem;  /* read from thread local
                                          static variable */
          A.localItem = new B();       /* write to a thread local
                                          static variable */

          D sharedValue = A.sharedItem;   /* read from shared
                                             static variable */
          A.sharedItem = new D();        /* write to shared
                                             static variable */
     }
}
```

Design

Our Design

use “volatile” to specify the shared data

accesses of data



Program 2 A program using thread-local static variables in a race-free design.

```
class A {
     public static B localItem;
     public static volatile D sharedItem;
}

class C extends Thread(){
     public void run(){
         B localValue = A.localItem;  /* read from thread local
                                        static variable */
         A.localItem = new B();       /* write to a thread local
                                        static variable */

         D sharedValue = A.sharedItem;  /* read from shared
                                            static variable */
         A.sharedItem = new D();       /* write to shared
                                            static variable */
     }
}
```

Design

	Original	New
semantic	thread-local is not inherent need support from ThreadLocal class	thread-local is inherent with in semantics
data access	map searching	• static: table look-up based approach • non-static: normal access without overhead
initial value	• fixed initial value • manually and statically	• inherent initial value from parents • automatically and at run-time

Implementation

Thread Local Accesses

- at the start of thread, make local copy all reachable reference objects if that field is not volatile

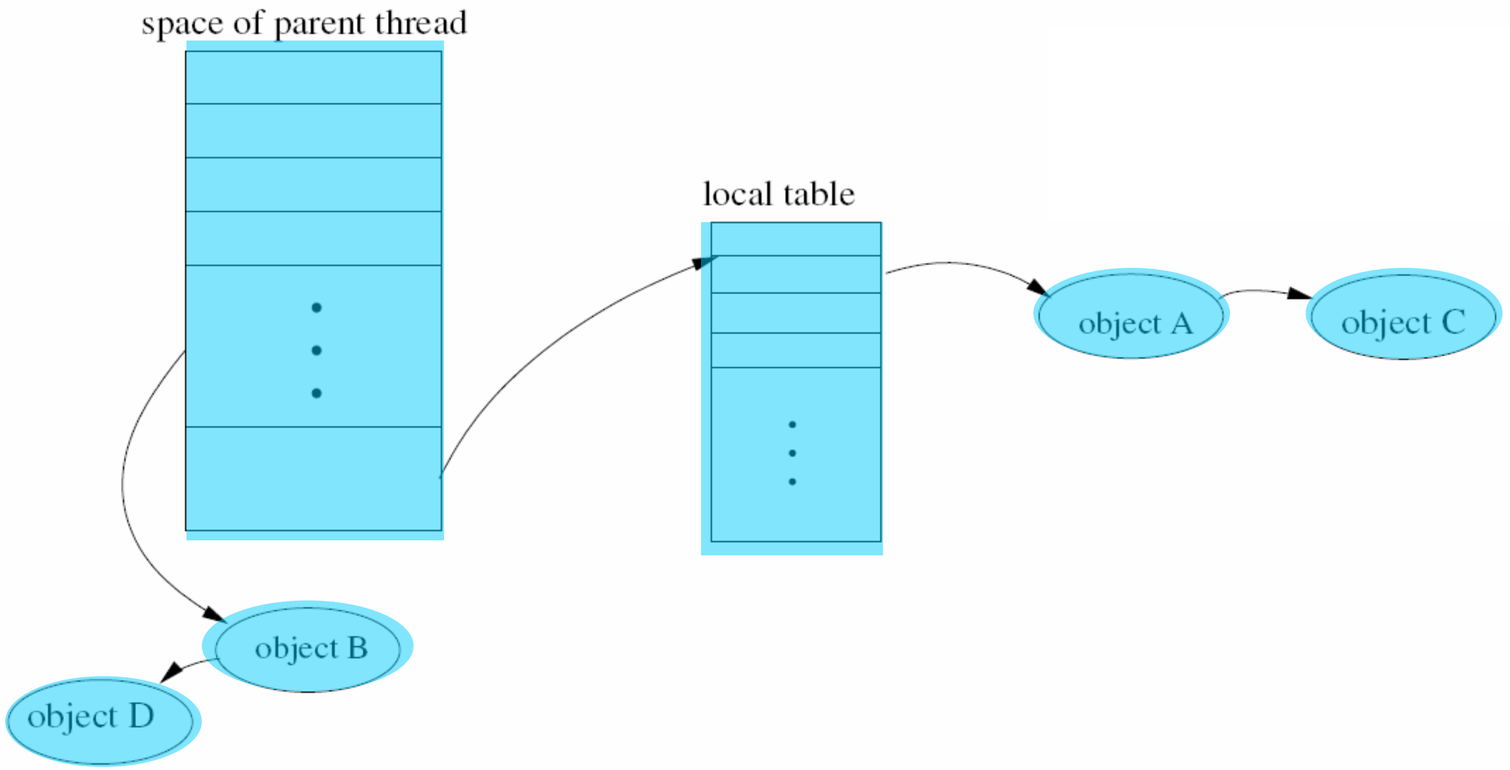
Class.staticField

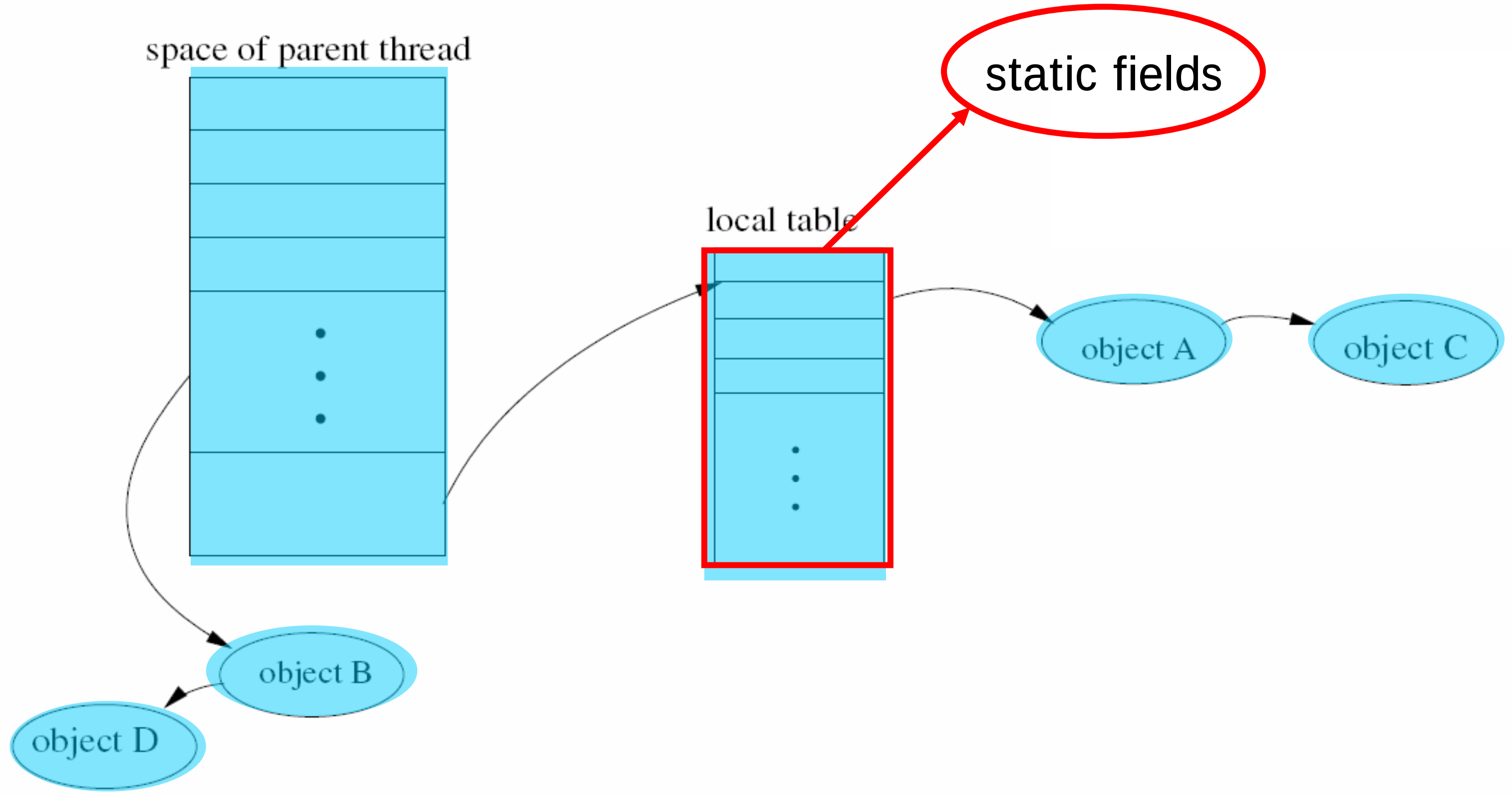
shared with all threads

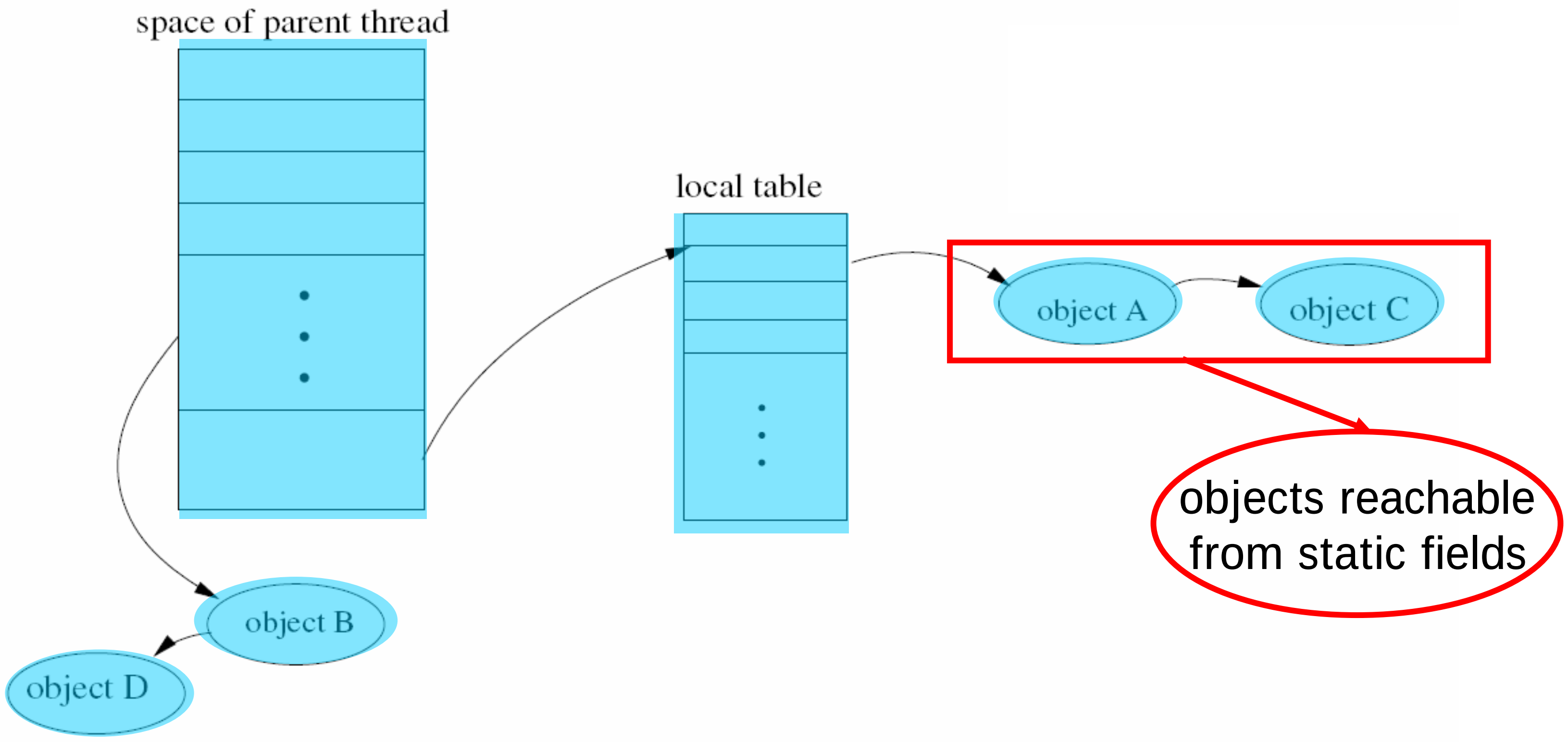
this.field

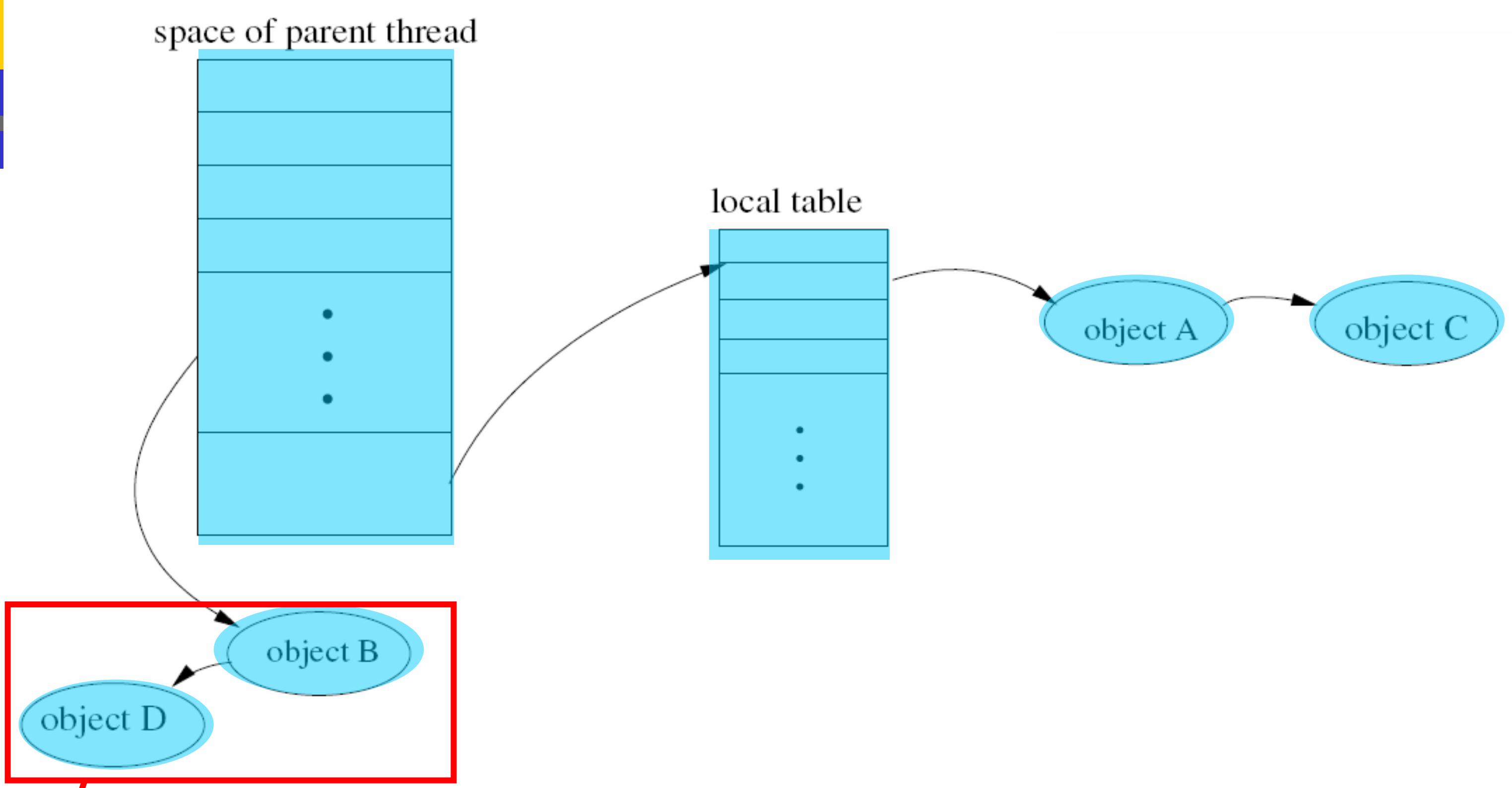
shared with parent threads

- we do this through deep-copying

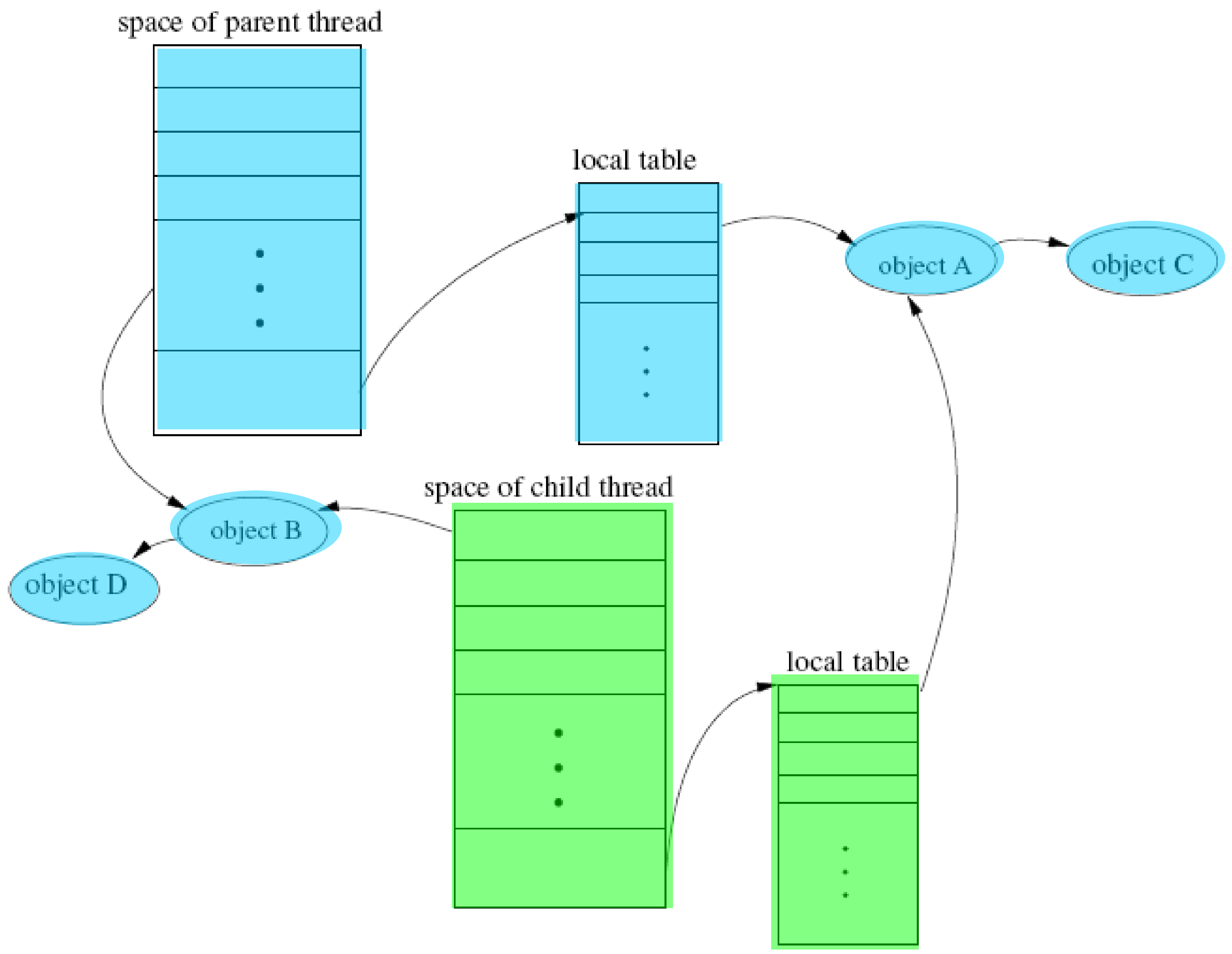

space of parent thread
local table
object A
object C
object B
object D

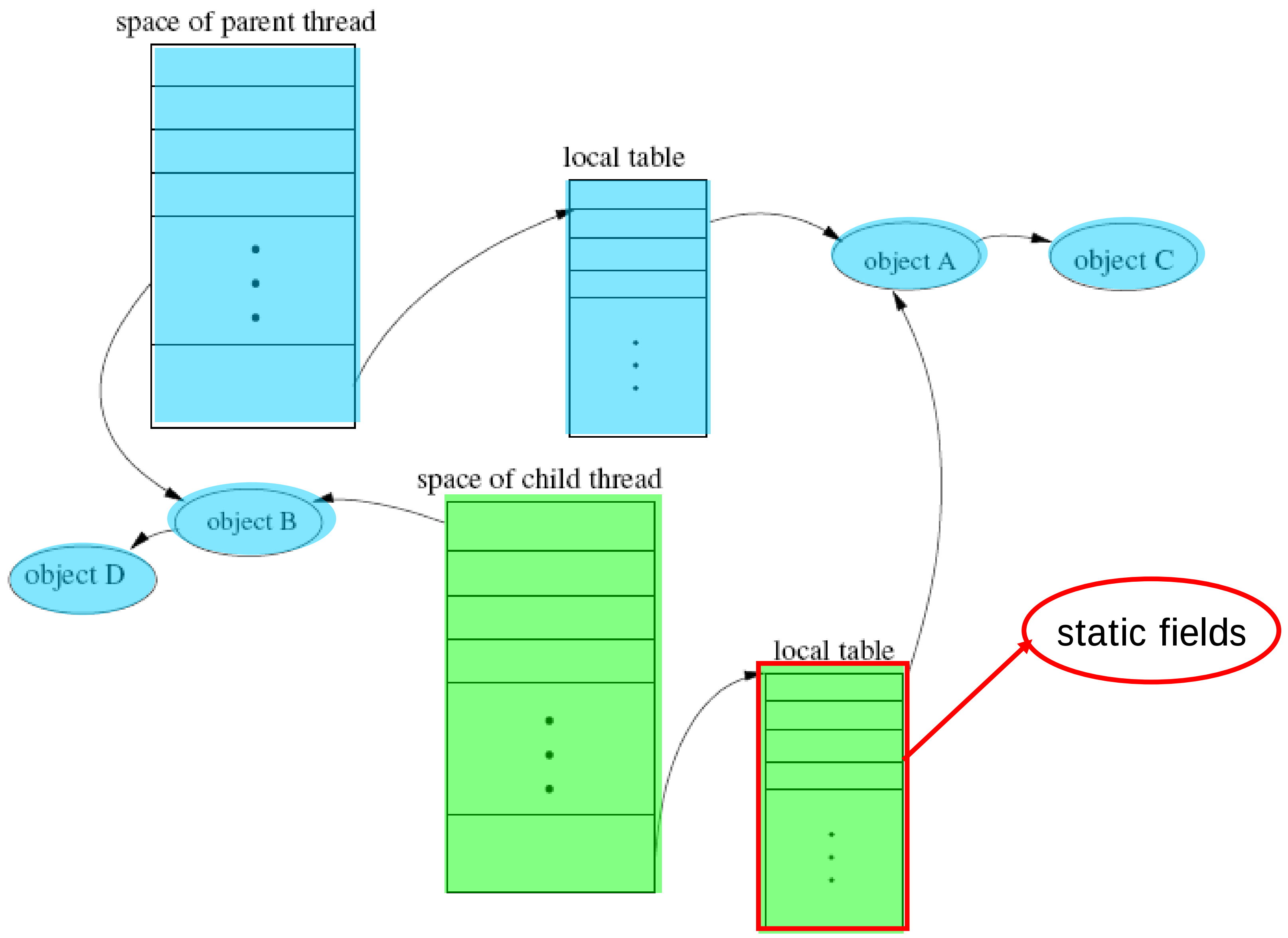

space of parent thread
local table
static fields
object A
object C
object B
object D

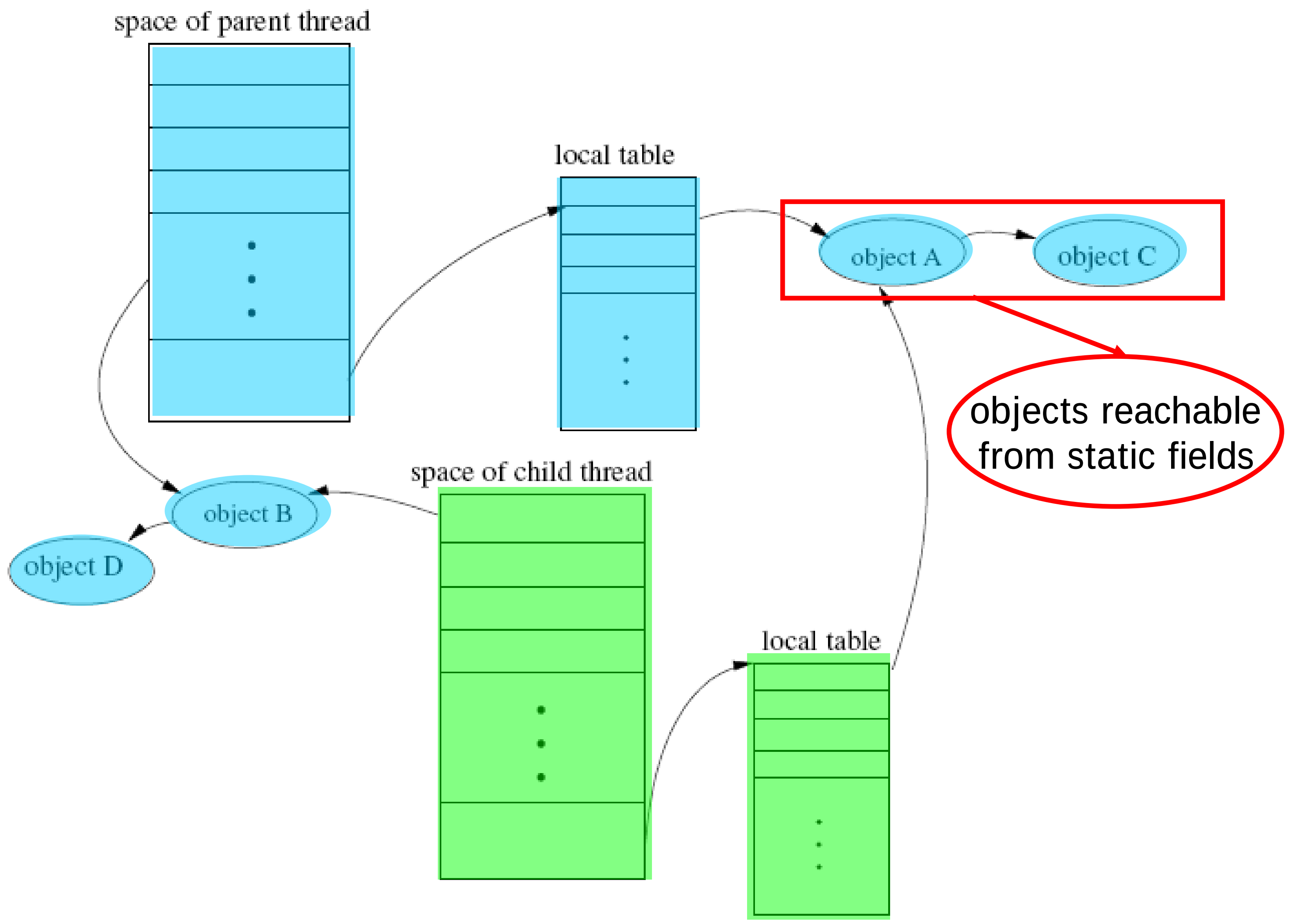

space of parent thread
local table
object A
object C
object B
object D
objects reachable from static fields

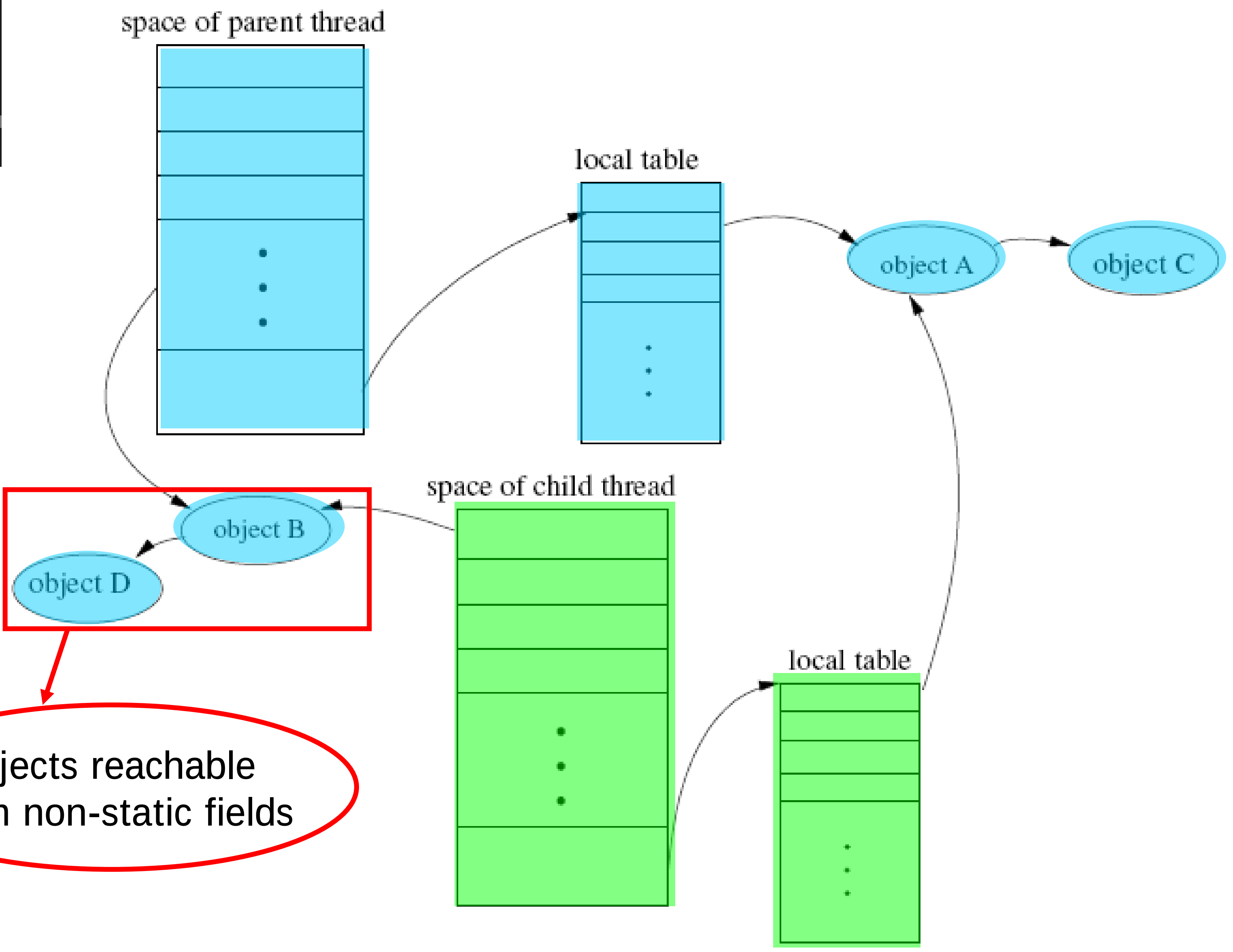


objects reachable from parent thread

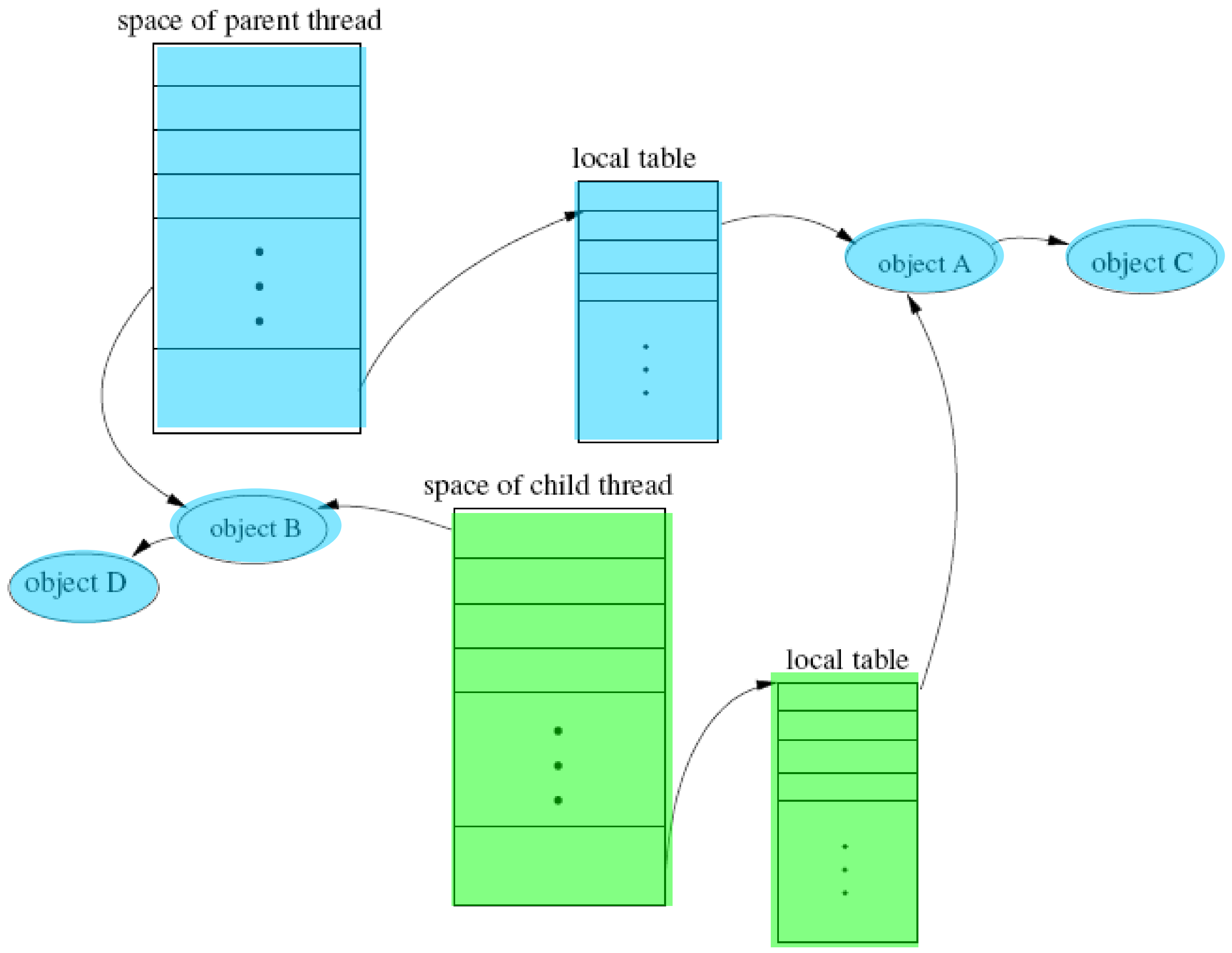

space of parent thread
local table
object A
object C
object B
object D
space of child thread
local table

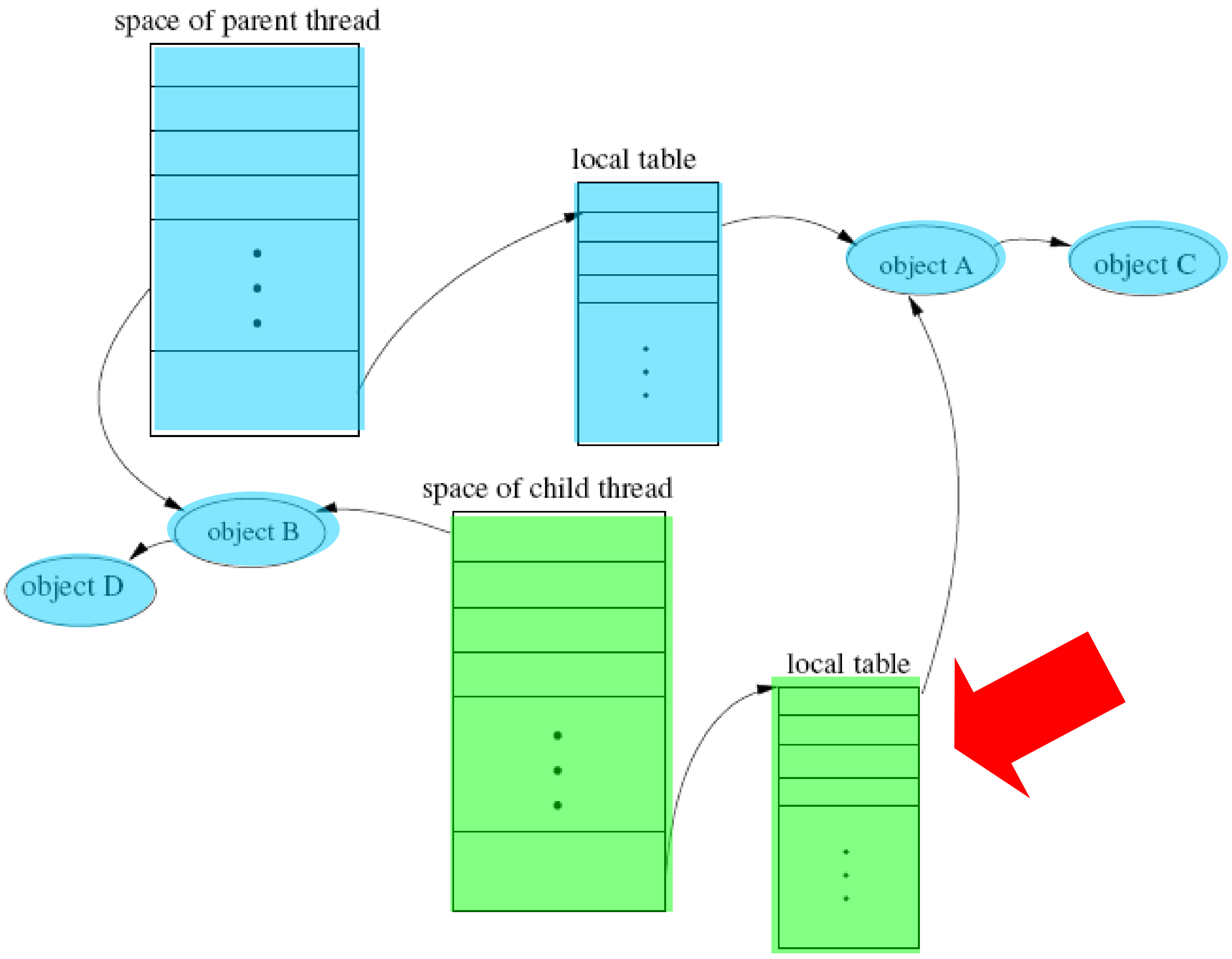

space of parent thread
local table
object A
object C
object B
object D
space of child thread
local table
static fields

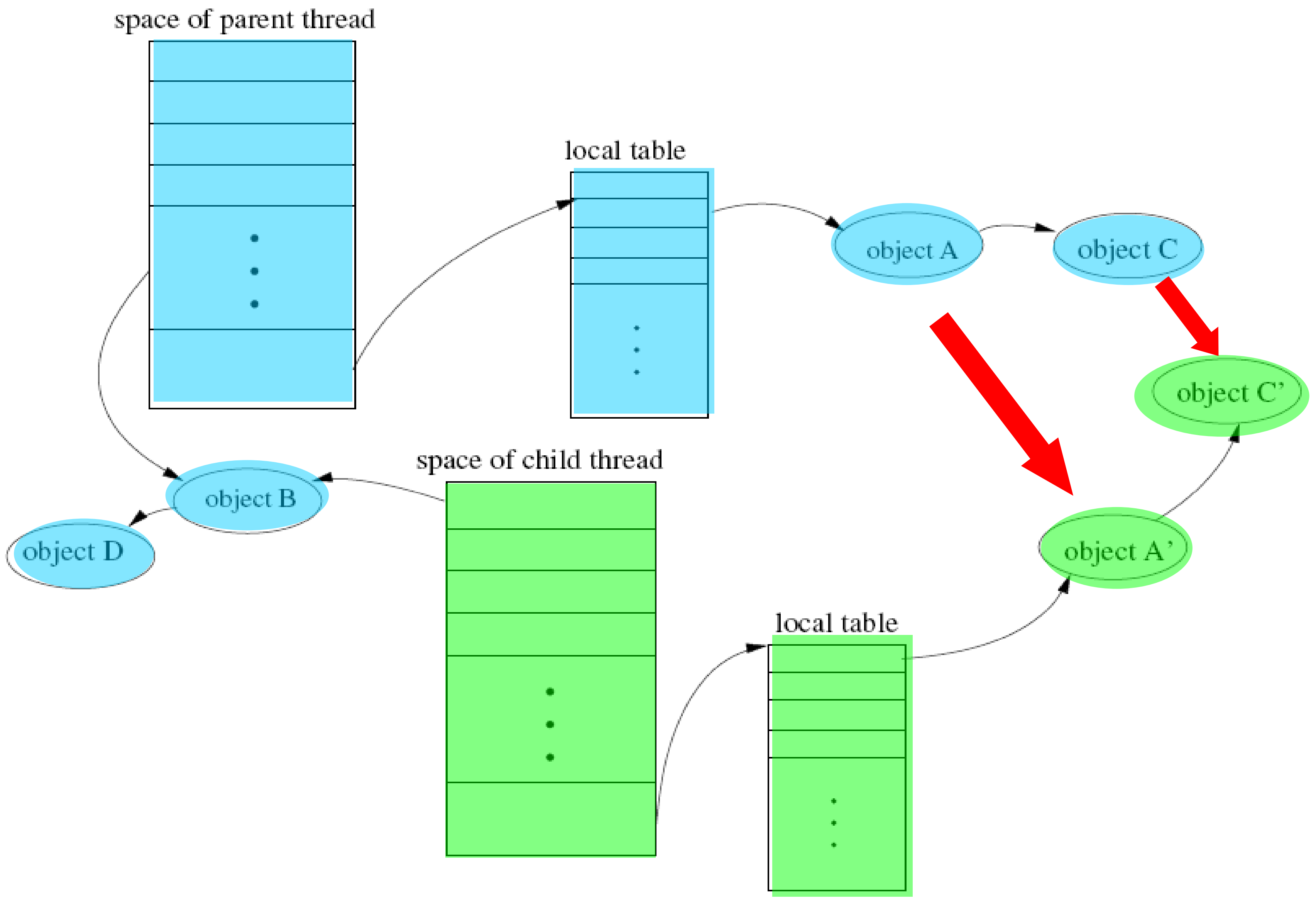

space of parent thread
local table
object A
object C
objects reachable from static fields
space of child thread
object B
object D
local table

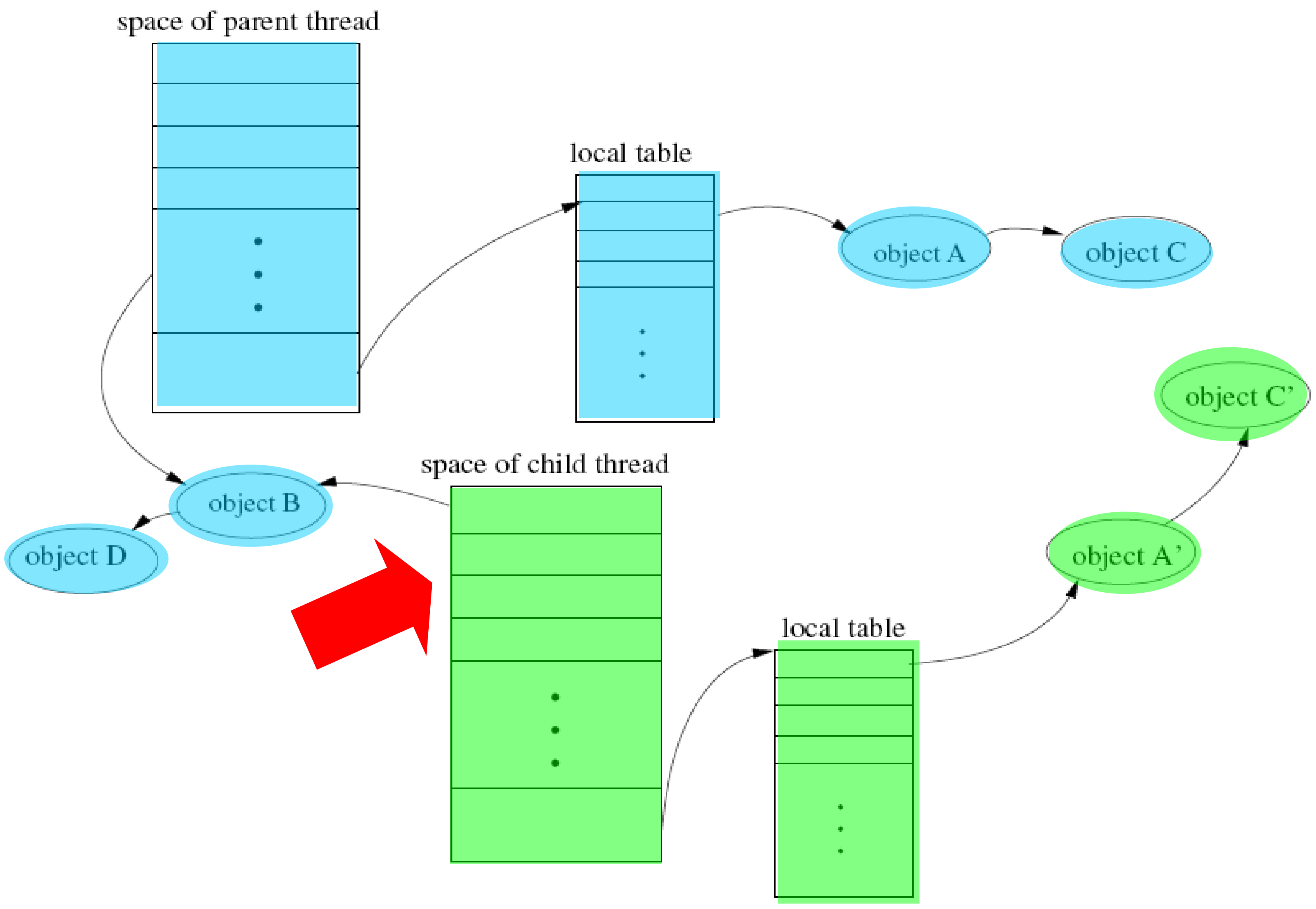


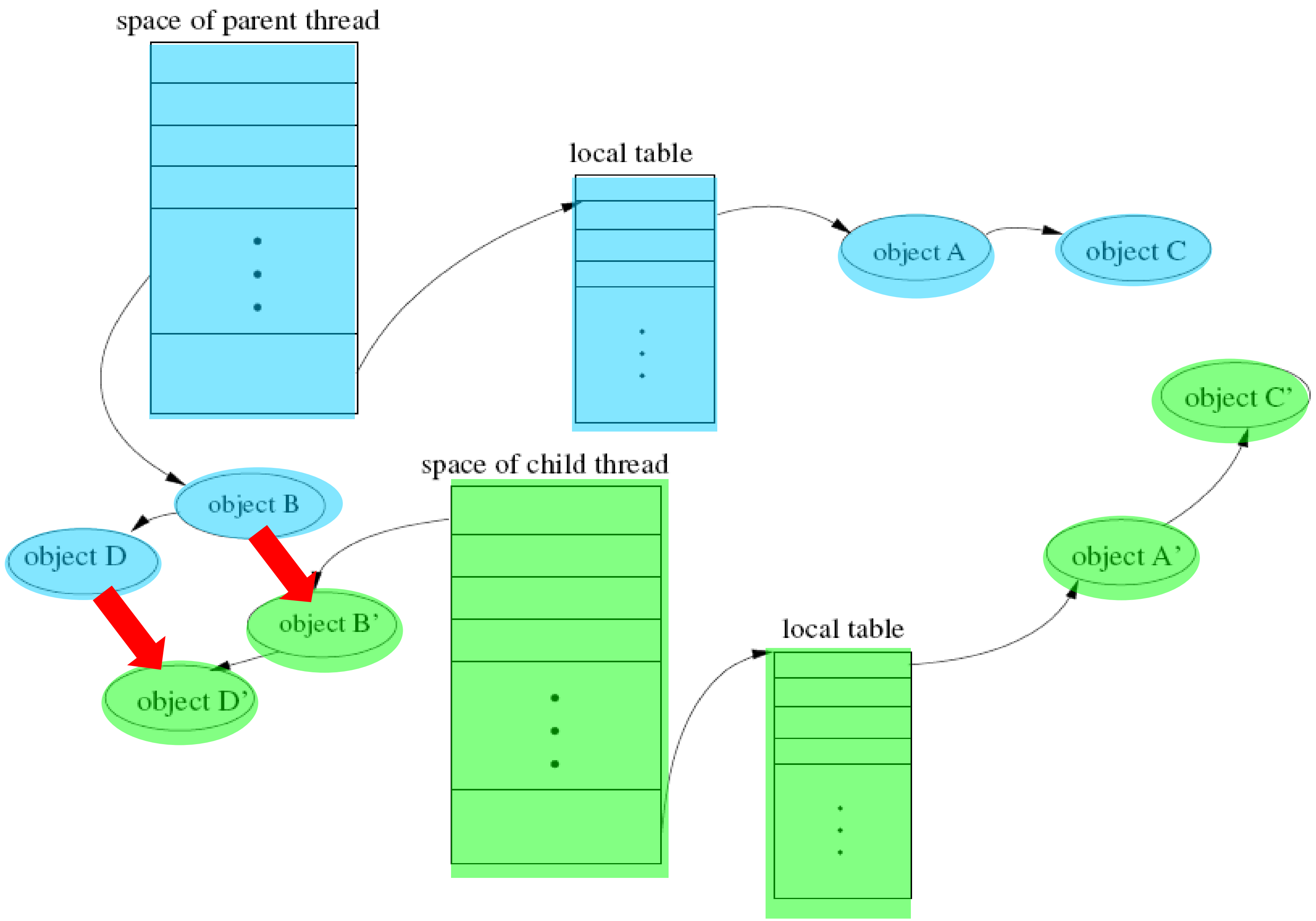
objects reachable from non-static fields

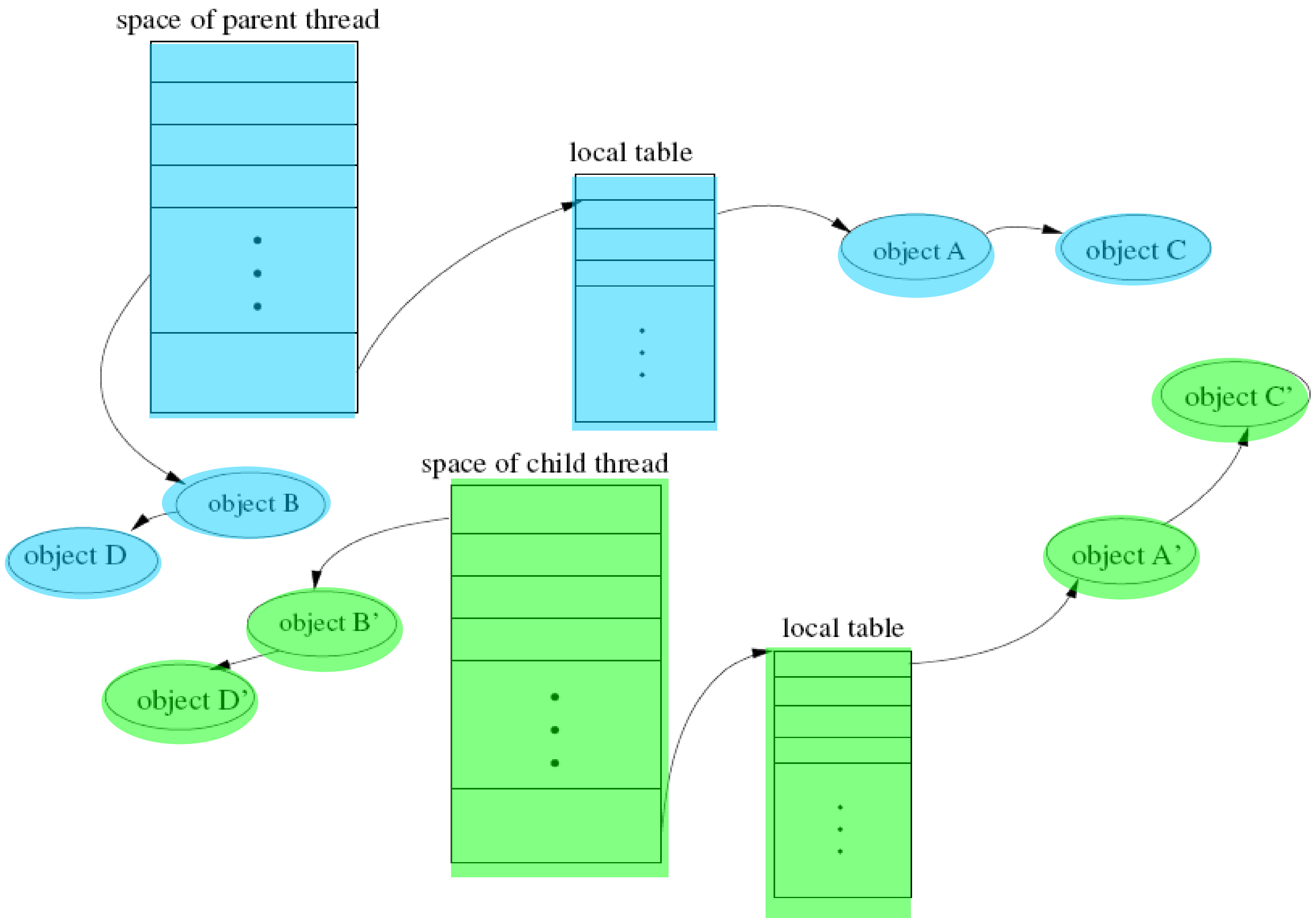

space of parent thread
local table
object A
object C
object B
object D
space of child thread
local table


space of parent thread
local table
object A
object C
object B
object D
space of child thread
local table


space of parent thread
local table
object A
object C
object C'
object B
object D
space of child thread
object A'
local table

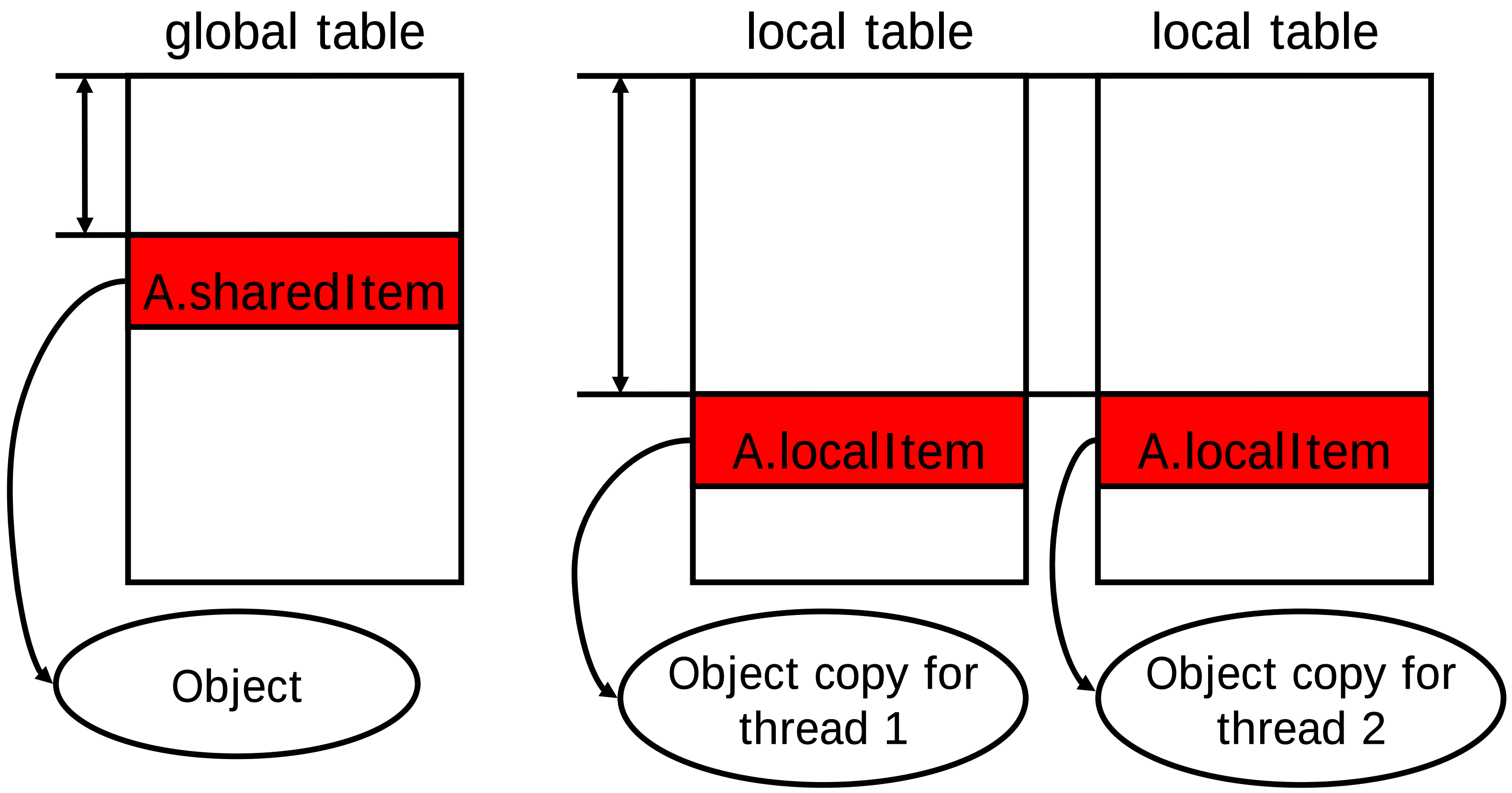

space of parent thread
local table
object A
object C
object C'
object B
space of child thread
object D
object A'
local table


space of parent thread
local table
object A
object C
object B
object D
space of child thread
object C'
object A'
object B'
object D'
local table


space of parent thread
local table
object A
object C
object B
object D
space of child thread
object C’
object A’
object B’
object D’
local table

Implementation

- table look-up based mechanism to speed up



Experiments

Implementation Environment:

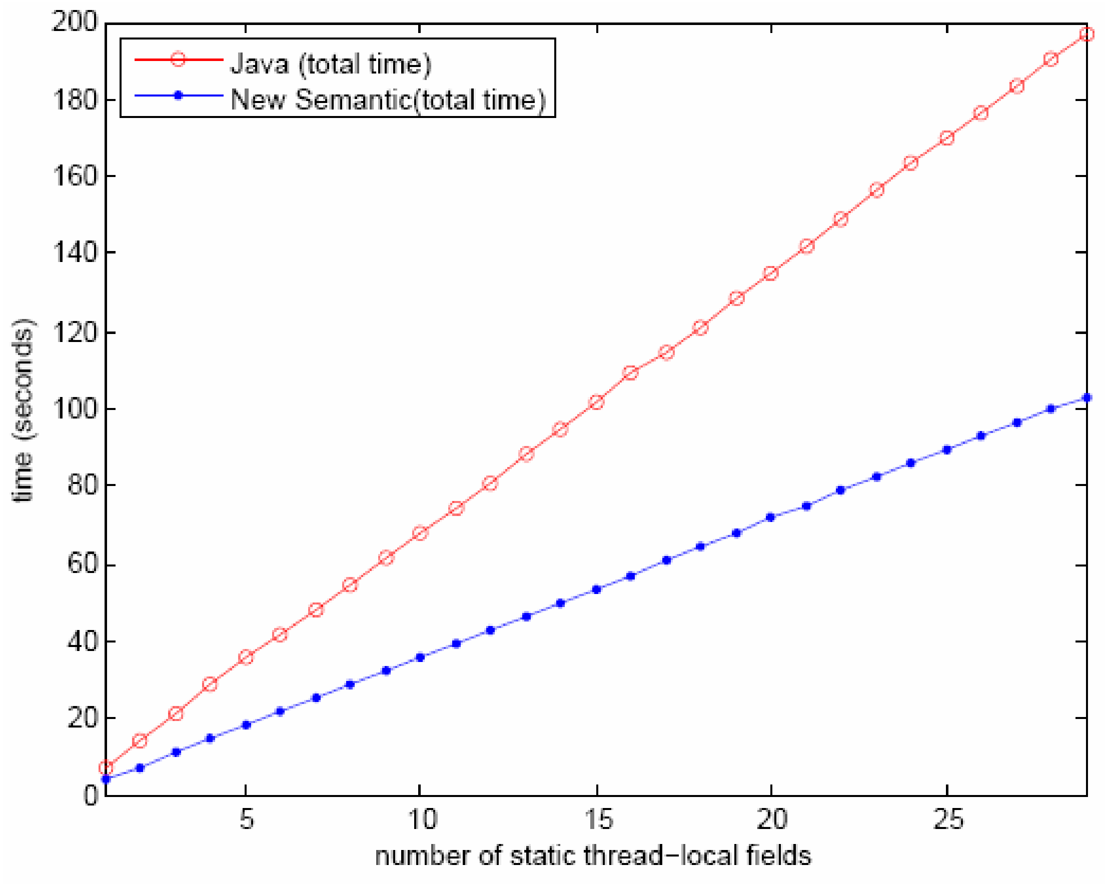
JikesRVM 3.1.1

Micro Benchmarks:

Reads and writes operations on thread-local static field

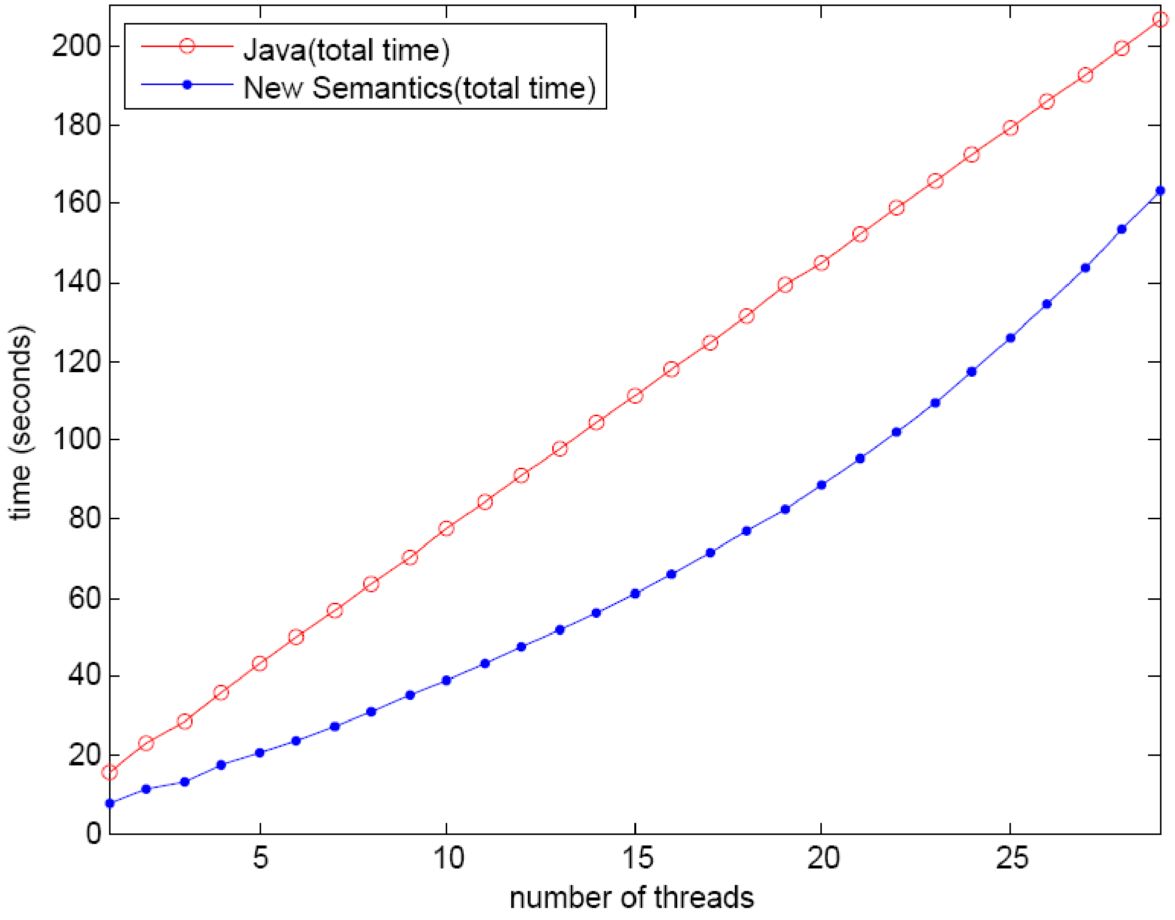
Experiments

Micro Benchmarks



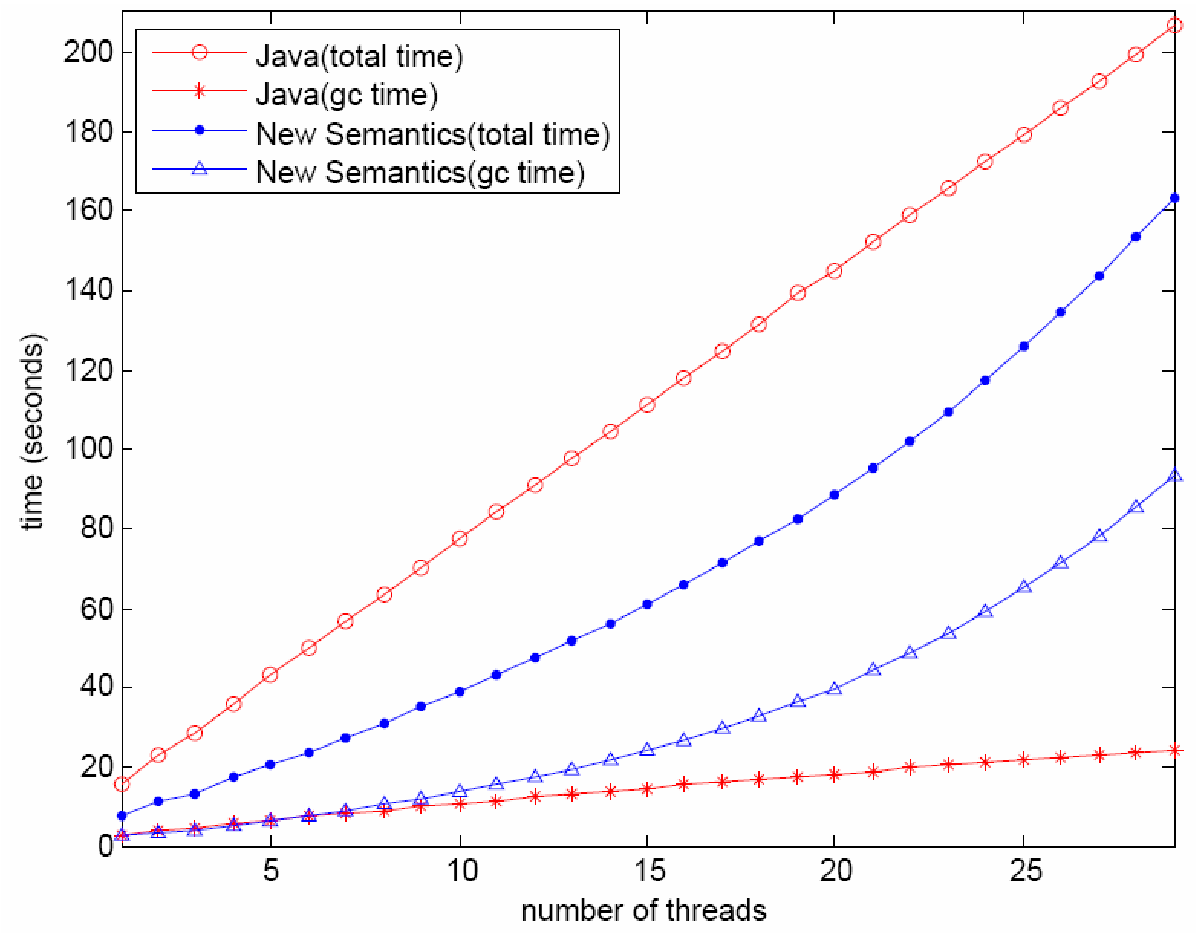
Experiments

Micro Benchmarks



Experiments

Micro Benchmarks

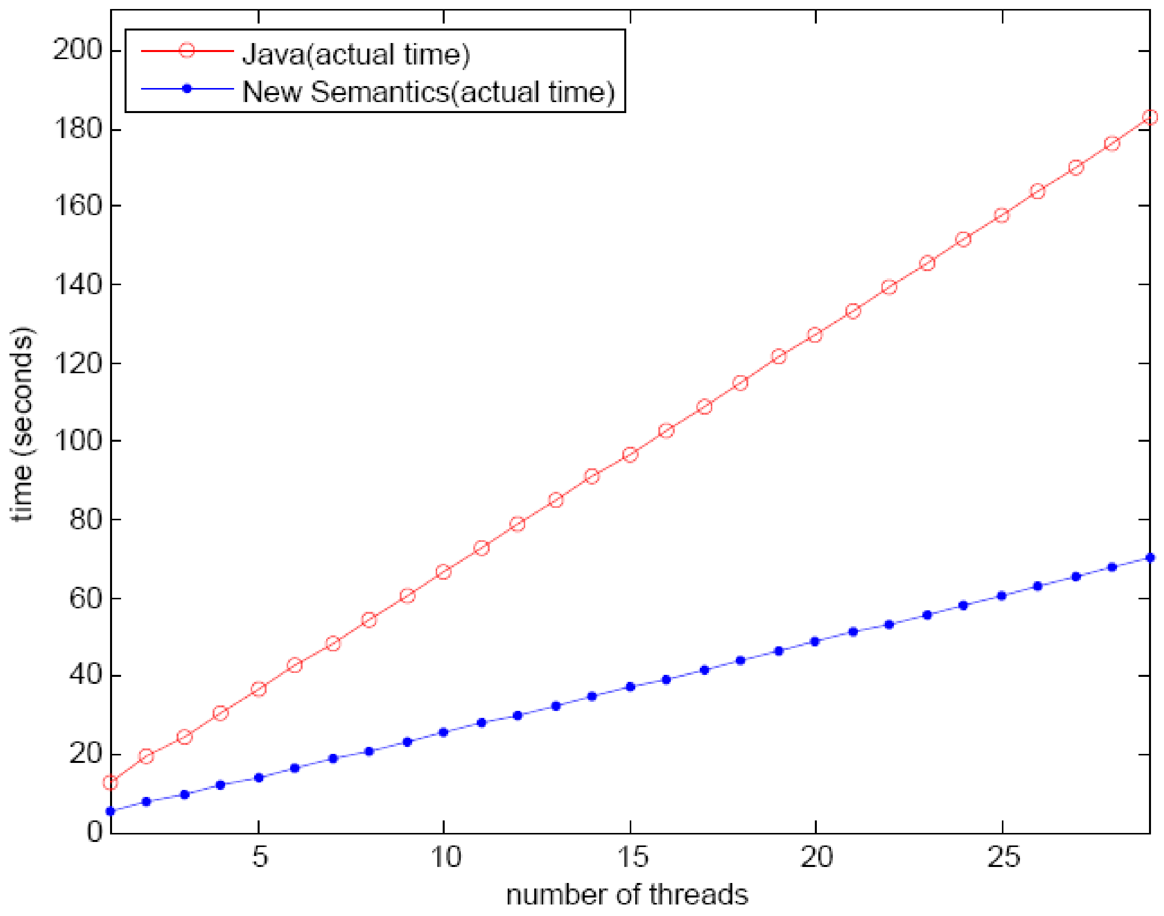


gc time of new design increases faster !!

We need special garbage collector adapted to our dialect

Experiments

Micro Benchmarks



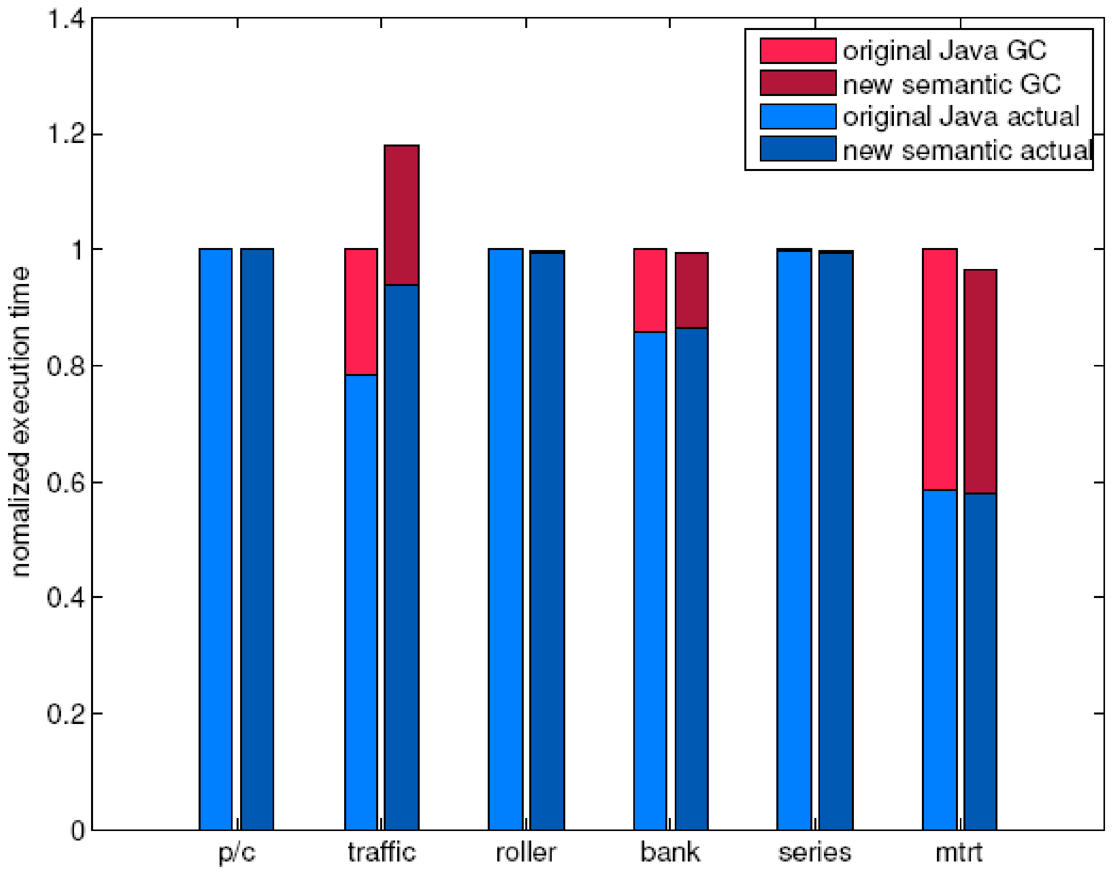
Experiments

Non-trivial Benchmarks:

Sun Java Tutorial:	**Producer/Consumer (P/C)**
Sable Research Group:	**traffic, roller coaster**
Doug Lea:	**bank**
Java Grande Forum Benchmark Suite:	**series**
SPECJVM98:	**mtrt**

Experiments

Non-trivial Benchmarks



- most benchmarks shows comparable performance
- traffic benchmarks runs considerably slower
  - 289 threads->too many

Conclusion & Future Work

Conclusion

- Data-Race-Free
- thread-local data VS shared data
- thread-local and Java Virtual Machine

Conclusion & Future Work

Future Work

- Improve performance of current design
  - reduce the copying overhead
  - reduce gc time
- Full data-race-free language design

Thank You!