The Design, Implementation and Evaluation of a Pluggable Type Checker for Thread-Locality in Java

By: Amanj Sherwany

2011

http://www.amanj.me

Background

- Loci is a static checker for thread-locality for Java-like languages.
- Programmers express thread-locality through annotations in the source code.
- Preservation of thread-locality is checked statically.
- Proposed by Wrigstad et al. in 2009

UPPSALA UNIVERSITET

Informationsteknologi

Why Thread-Locality?

- Simplifying concurrent and parallel programming.
  - Accesses to thread-local data are sequential and easy to reason about.
- There will never be data races or dead locks on thread-local data.

Side-Effects of Thread-Locality

- In real-time systems, thread-locality avoids lock inflation which is important to calculate worst-case run-times/paths.
- Thread-local data can be collected without pausing other threads.
- No need to synchronise local data with main memory.

Thread-Locality in Java

- Java does not have support for programming with thread-local data.
- The little support it provides with **ThreadLocal** API is not enough, because:
  - Allows defining fields for which each accessing thread has its own copy.
  - But, nothing prevents the contents of the field to be shared across threads.

Pluggable Type Checkers

- First proposed by Bracha.
- Allow static checking of different program properties at different stages.
- In Bracha's terms:
  - Have no effect on the run-time semantics of the programming language.
  - Do not mandate type annotations in the syntax.

Pluggable Type Checkers, *Cont'd*

- Since version 5, Java has basic support for pluggable type checkers.
- Java 8 will have:
  - A more expressive annotation system.
  - A framework for designing custom type checkers, called “the Checker framework”.

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics

Memory-partitioning in Loci (Logical)

🟦 Subheap

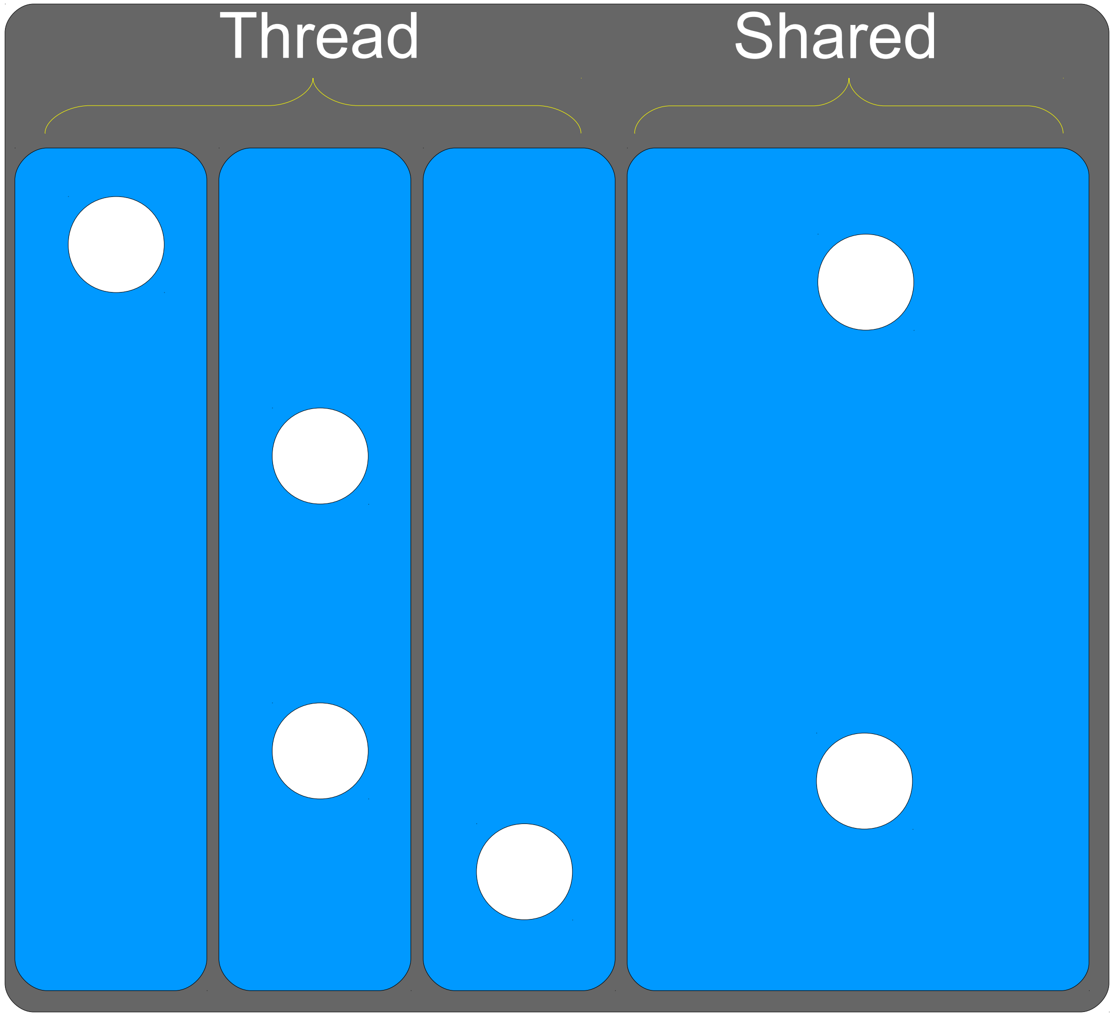
○ Objects

➡ Reference

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



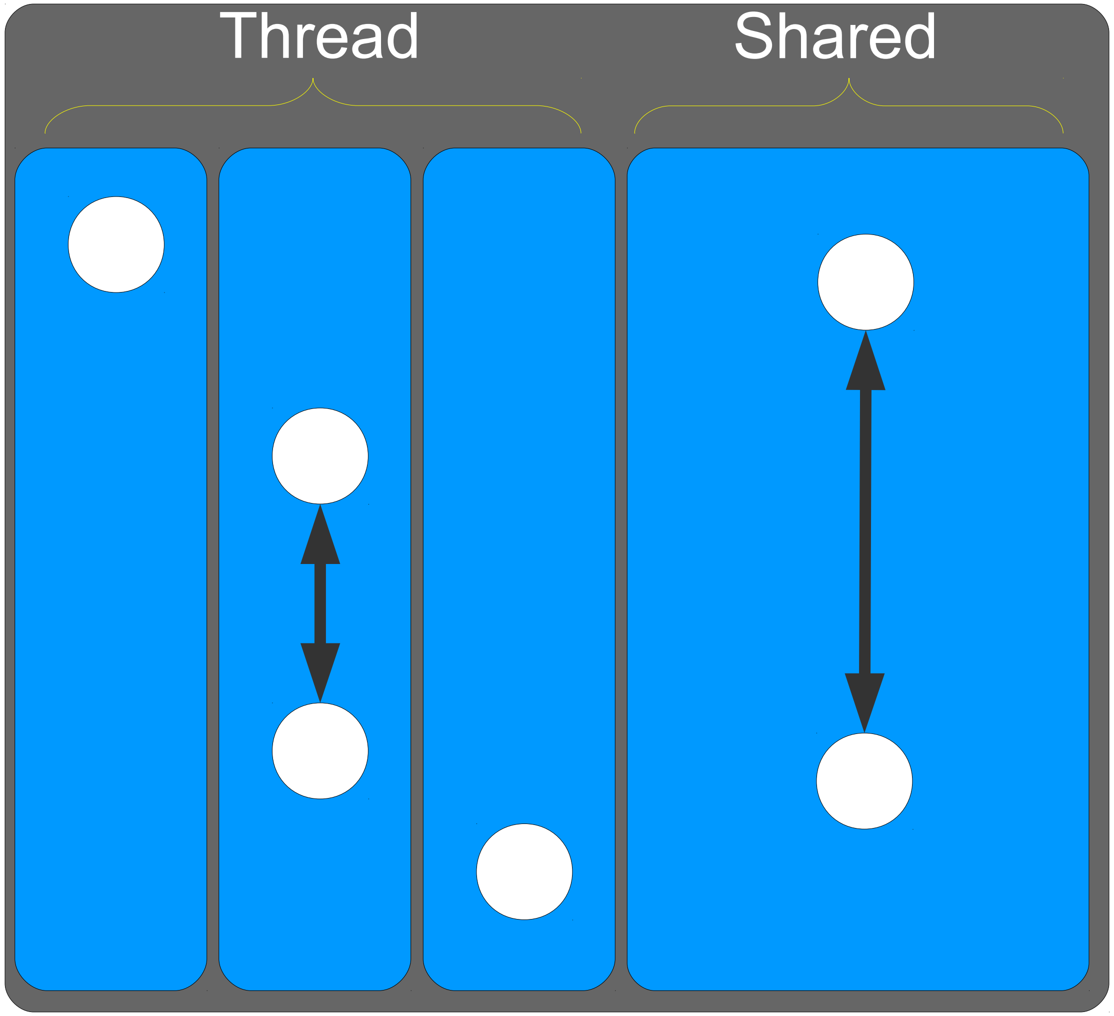
Allowed:

Disallowed:

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



Allowed:

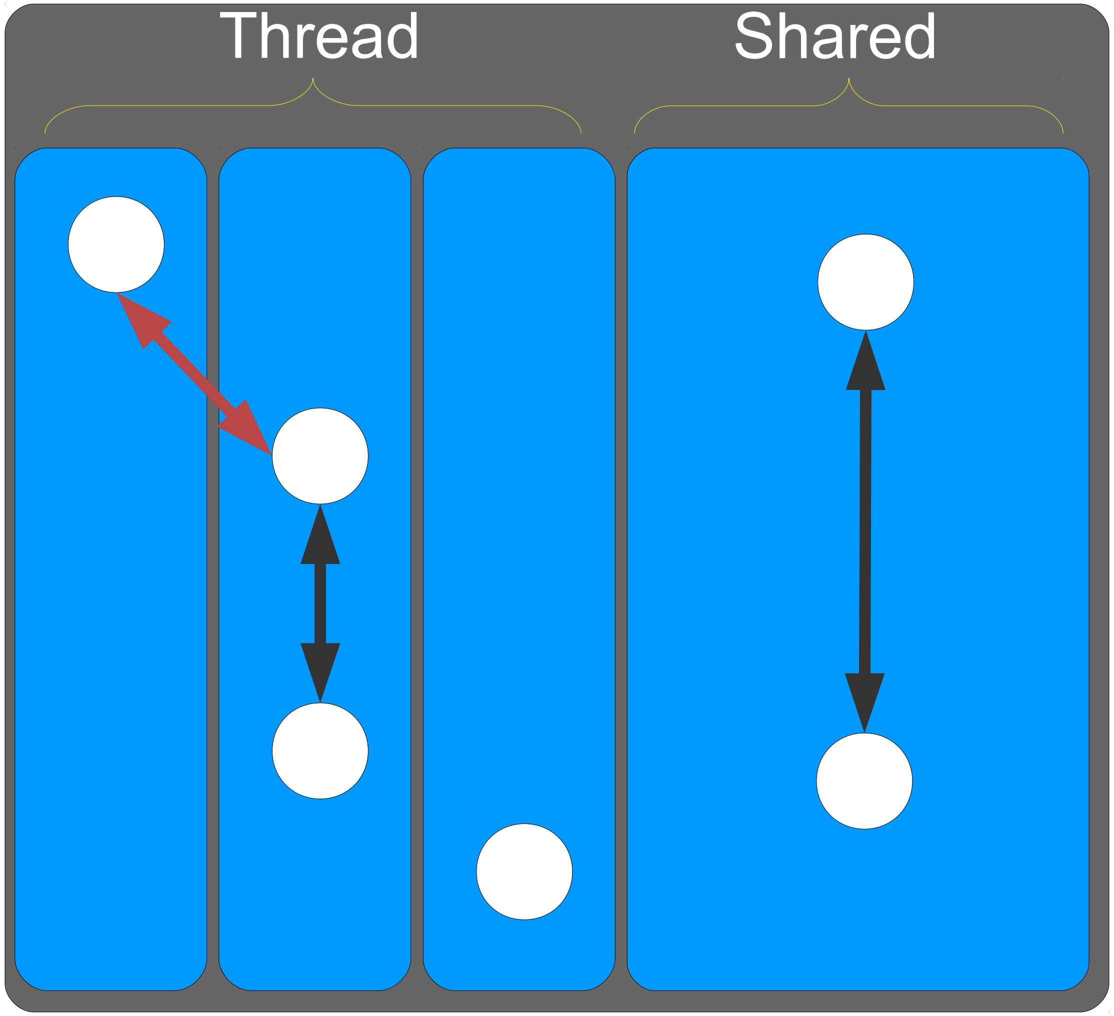
- Intra-thread & Intra-shared

Disallowed:

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



Allowed:

- Intra-thread & Intra-shared

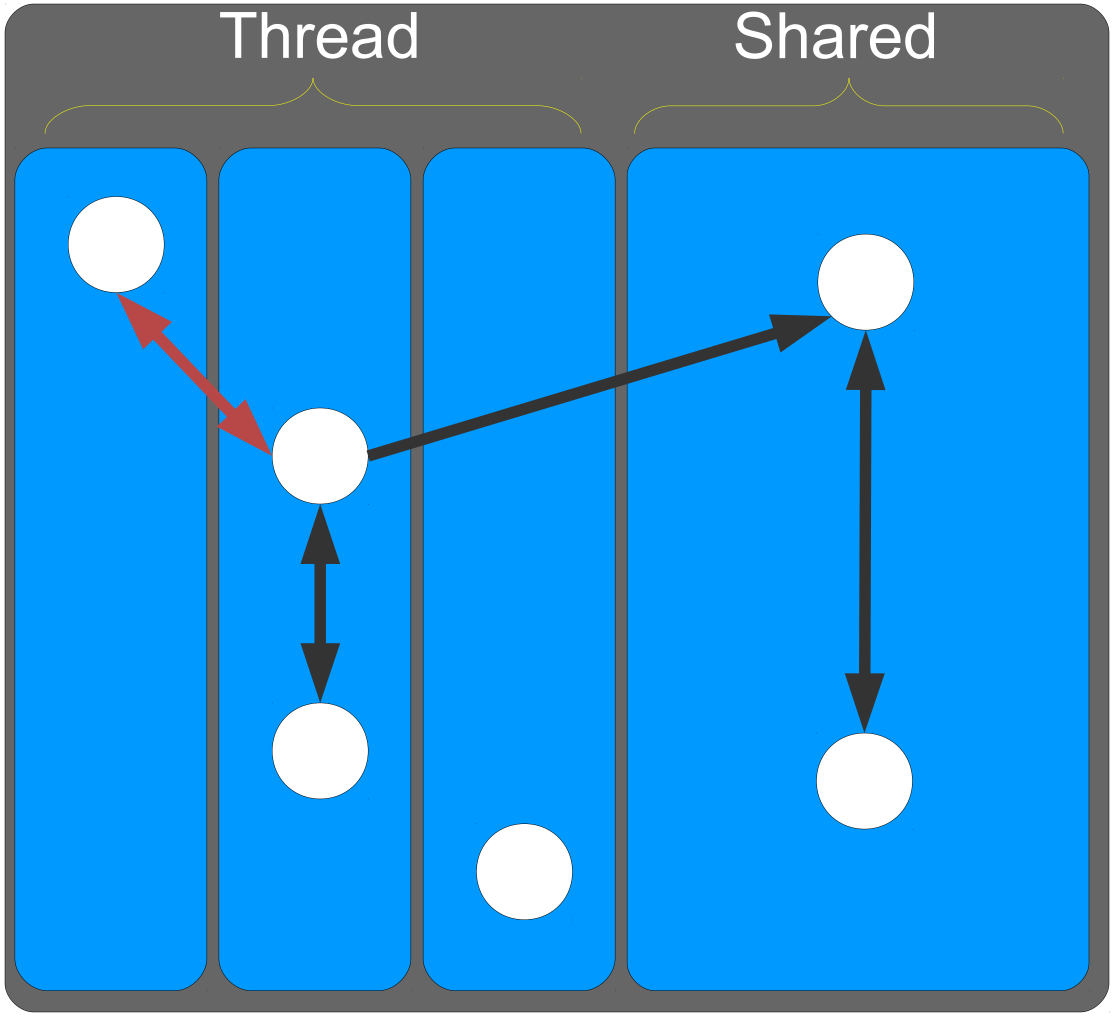
Disallowed:

- Inter-thread

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



Allowed:

- Intra-thread & Intra-shared
- *Thread to Shared*

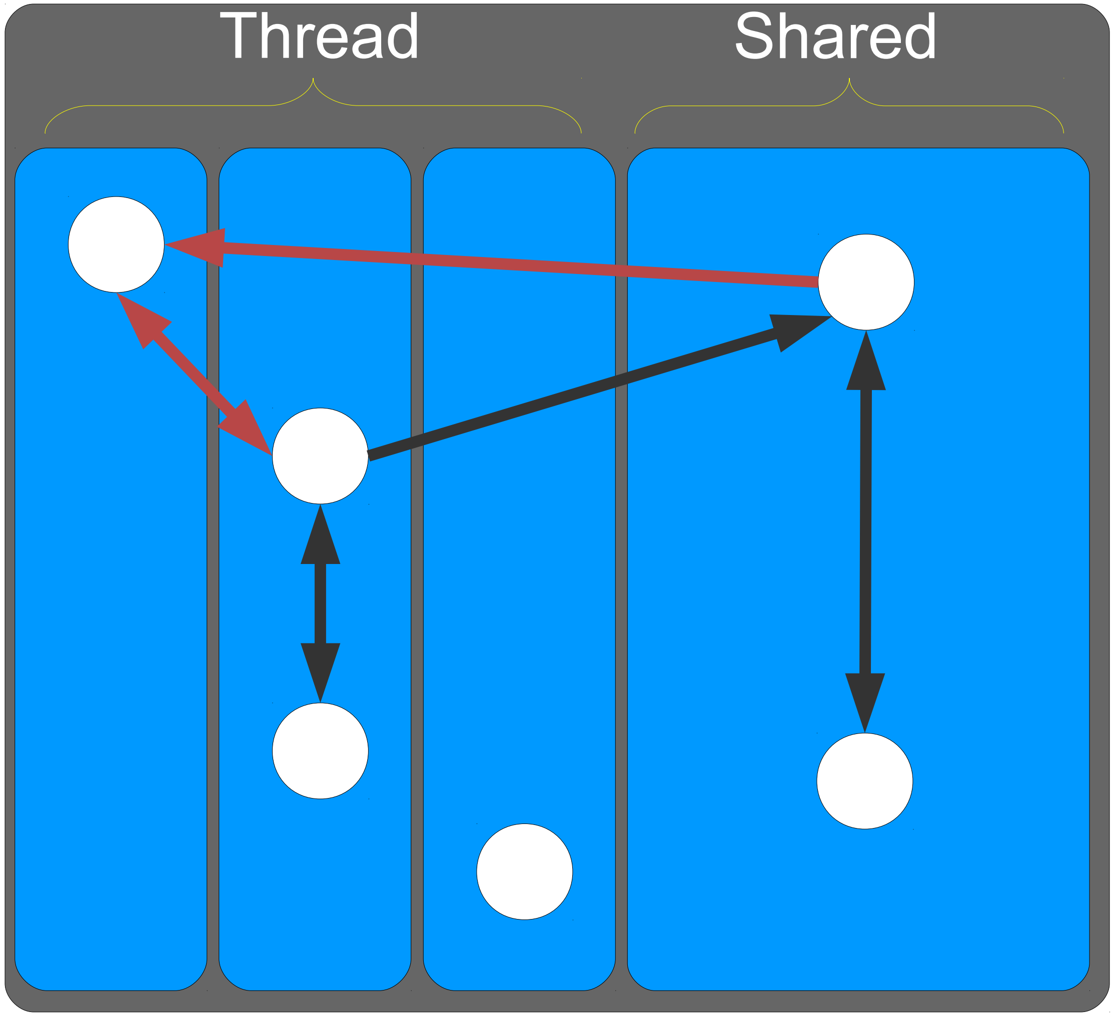
Disallowed:

- Inter-thread

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



Allowed:

- Intra-thread & Intra-shared
- *Thread to Shared*

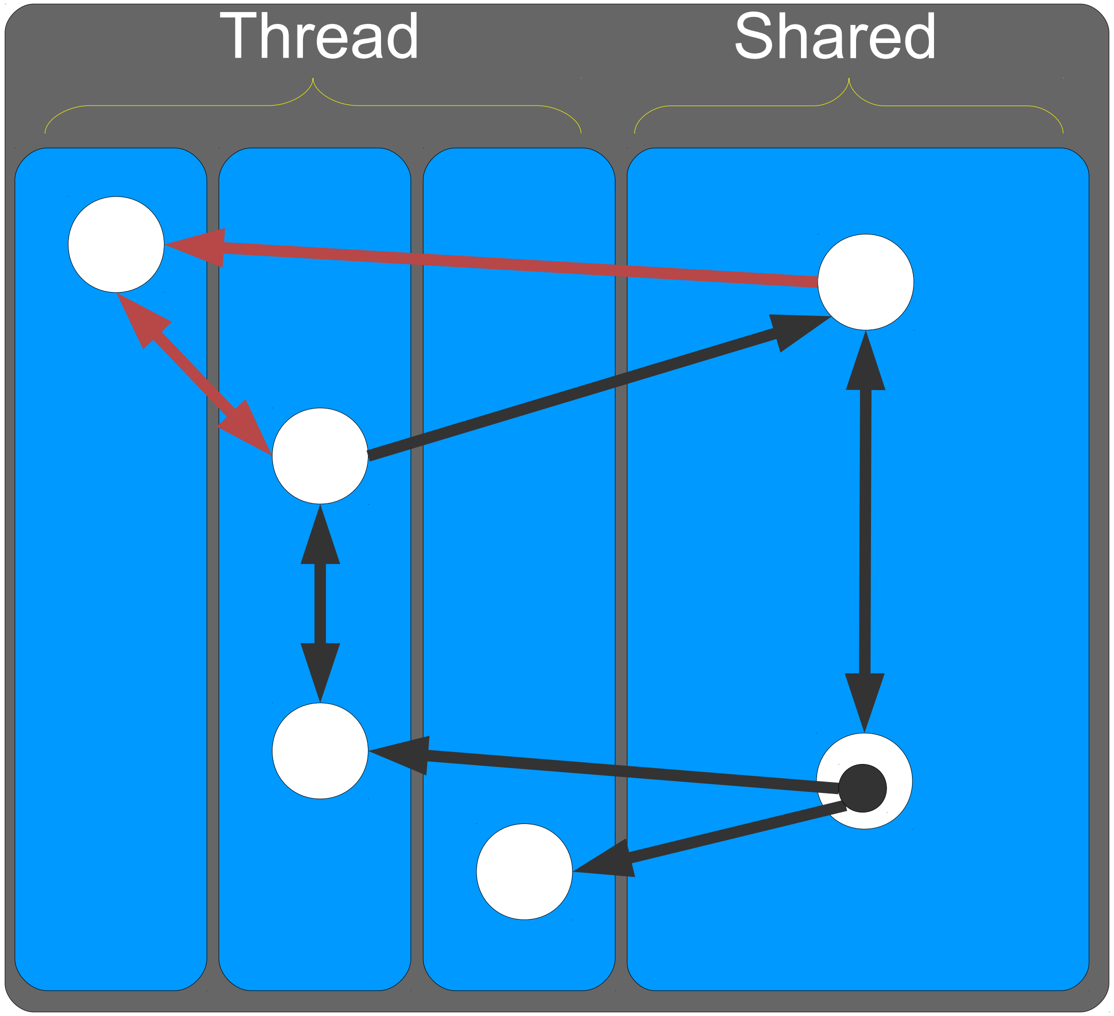
Disallowed:

- Inter-thread
- *Shared to thread*

UPPSALA UNIVERSITET

Informationsteknologi

Loci Semantics



Allowed:
- Intra-thread & Intra-shared
- *Thread to Shared*

Disallowed:
- Inter-thread
- *Shared to thread (unless guarded by thread local fields)*

UPPSALA UNIVERSITET

Informationsteknologi

Loci 1.0

- The old (initial) version of Loci uses the standard Java annotation system.
- Is available as an Eclipse plugin only.
- Does not support generics (due to the limitations in Java annotation system in JDK 6).
- Does not cover all the features in Java.

What the Thesis is About

- **Extending** Loci into Loci 2.0:
  - Support for generics.
  - Support for locality-polymorphic methods.
  - More flexible annotations and support for corner cases.
    - Equality test between objects from different thread-localities.
    - Static utility methods, like sorting (more later).

UPPSALA UNIVERSITET

Informationsteknologi

What the Thesis is About - *Cont'd*

- **Re-implementing** Loci using the Checker framework in Java 8, instead of the standard Java annotation system.
  - Allows more flexible annotations.
  - Fully annotated Java API.
- **Evaluating** our extended Loci system, and comparing the results with the previous version.

Loci Annotations

- **@Local**, which denotes a thread-local value.
- **@Shared**, which denotes a value that can be arbitrarily shared between threads.
- **@ThreadSafe,** which denotes a value that must be treated in such a way that thread-locality is preserved, but the value may not be thread-local in practice.

Data Flow Constraints

$$\begin{tikzcd}
 & \text{@ThreadSafe} \arrow[loop above] & \\
\text{@Local} \arrow[ur] \arrow[loop left] & & \text{@Shared} \arrow[ul] \arrow[loop right] \\
 & \bot \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

UPPSALA UNIVERSITET

Informationsteknologi

Basics of Loci

@Local class A{...} *//A thread-local class*
@Shared class B{...} *//A shared class*

class D **extends** A{...} *//An implicit thread-local class*
class E **extends** B{...} *//An implicit shared class*
@Shared F **extends** A{...} *//Invalid*
@Local G **extends** B{...} *//Invalid*

A a; *//A thread-local data*
B b; *//A shared data*
@Shared A bad1; **@Local** B bad2; *//Invalid*

Basics of Loci, *Cont'd*

class A{...} *//A flexible class*

@Local class B **extends** A{...} *//A thread-local class*
@Shared class D **extends** A{...} *//A shared class*
F **extends** A{...} *//A flexible class*

@Local A a; *//A thread-local data*
@Shared A b; *//A shared data*
@ThreadSafe A c; *//A thread-safe reference*
A d; *//The same thread-locality as the enclosing object (next slide)*

UPPSALA UNIVERSITET

Informationsteknologi

Basics of Loci, *Cont'd*

- The golden rule:

"Unless they are **explicitly** annotated, the thread-locality of instances of **flexible** classes follow the thread-locality of their enclosing objects."

The “Object” Class

```
public class Object {
        public final native @Shared Class getClass();
        public boolean equals(@ThreadSafe Object obj);
        protected native Object clone();
}
```

The “Object” Class

Object is a flexible class

```
public class Object {

        public final native @Shared Class getClass();

        public boolean equals(@ThreadSafe Object obj);

        protected native Object clone();
}
```

The “Object” Class

Object is a flexible class

Inter-thread-locality equality test

```
public class Object {

        public final native @Shared Clas        tClass();

        public boolean equals(@ThreadSafe Object obj);

        protected native Object clone();
}
```

UPPSALA UNIVERSITET

Informationsteknologi

The “Object” Class

Object is a flexible class

The thread-locality of the cloned instance follows the original instance (the golden rule)

Inter-thread-locality equality test

```
pu
                                   Clas        etClass();

       public boolea          als(@ThreadSafe Object obj);

       protected native Object clone();
}
```

UPPSALA UNIVERSITET

Informationsteknologi

The “ThreadLocal” Class

```
@Shared public class
        ThreadLocal<T extends @Local Object>{

        protected T initialValue();
        public T get();
        public void set(T value);
}
```

UPPSALA UNIVERSITET

Informationsteknologi

The “ThreadLocal” Class

ThreadLocal is a shared class

```
@Shared public class
        ThreadLocal<T extends @Local Object>{

        protected T initialValue();
        public T get();
        public void set(T value);
}
```

UPPSALA UNIVERSITET

Informationsteknologi

The "ThreadLocal" Class

ThreadLocal is a shared class

Holds thread-local fields

```
@Shared public class
        ThreadLocal<T extends @Local Object>{

        protected T initialValue();
        public T get();
        public void set(T value);
}
```

Standard Java Classes

- We have annotated the standard Java classes.
  - In JDK 6:
    - 541 **@Shared** classes (~15.5%).
    - 2936 **flexible** classes (~84.5%).
- Runnable is annotated **@Shared**
- Throwable is annotated **@Local**

UPPSALA UNIVERSITET

Informationsteknologi

The Loci Tool

- Is a command line tool.
- Implemented as a plugin for the **javac**.
- On top of the Checker framework.
- Works with Java 5 and up!
- Works with ANT, Maven and different IDEs.
- Works on any OS that is supported by Java.

The Loci Tool, *Cont'd*

- Is open source, GPLv3.
- Can be freely downloaded and used.
- Has a production quality.
- Its design allows further enhancements (thanks to the flexibility of the Checker framework).

How to Use Loci?

- Install JSR 308 **javac**.
- Put Loci in your CLASSPATH.
- Annotate your program, and import Loci annotations:

```
import loci.quals.*;
```

- Run Loci, and fix the bugs:

```
javac -processor loci.LociChecker *.java
```

Demo

Demo 1

```
class Example {
        private Foo foo = null; // Should be a thread-local value
        private Foo bar = null; // Possibly shared value

        void frob() {
                bar = foo; // Leak!
        }
}
```

UPPSALA
UNIVERSITET

Informationsteknologi

Demo 1, *Cont'd*

```
class Example {
        ThreadLocal<Foo> foo = new ThreadLocal<Foo>();
        private Foo bar = null; // Possibly shared value

        void frob() {
                bar = foo.get(); // Reading foo requires indirection
        }
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 1, *Cont'd*

```
class Example {
        ThreadLocal<Foo> foo = new ThreadLocal<Foo>();
        private Foo bar = null; // Possibly shared value

        void frob() {
                bar = foo.get(); // Leak!
        }
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 1, *Cont'd*

```
class Example {
        private @Local Foo foo = null;
        private @Shared Foo bar = null;

        void frob() {
                bar = foo; // Not Allowed!
        }
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 2

```
class Example {
        private @Local Foo[] foo = null;
        private @Shared Foo[] bar = null;

        void frob() {
                sort(foo); //Invalid, sort accepts shared arrays
                sort(bar);
        }
        public static Foo[] sort(Foo[] array){...}
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 2, *Cont'd*

```
class Example {
        private @Local Foo[] foo = null;
        private @Shared Foo[] bar = null;
        void frob() {
                sort(foo); //OK, but we lost type information!
                sort(bar);
        }
        public static @ThreadSafe Foo[] sort
                        (@ThreadSafe Foo[] array){...}
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 2, *Cont'd*

```
class Example {
        private @Local Foo[] foo = null;
        private @Shared Foo[] bar = null;
        void frob() {
                foo = sort(foo); //Not OK, thread-safe return type
                sort(bar);
        }
        public static @ThreadSafe Foo[] sort
                        (@ThreadSafe Foo[] array){...}
}
```

UPPSALA UNIVERSITET

Informationsteknologi

Demo 2, *Cont'd*

```
class Example {
        private @Local Foo[] foo = null;
        private @Shared Foo[] bar = null;
        void frob() {
                foo = sort(foo); //OK!
                bar = sort(bar); //OK!
        }

        public static <@X T extends Foo> T[] sort(T[] array){...}
}
```

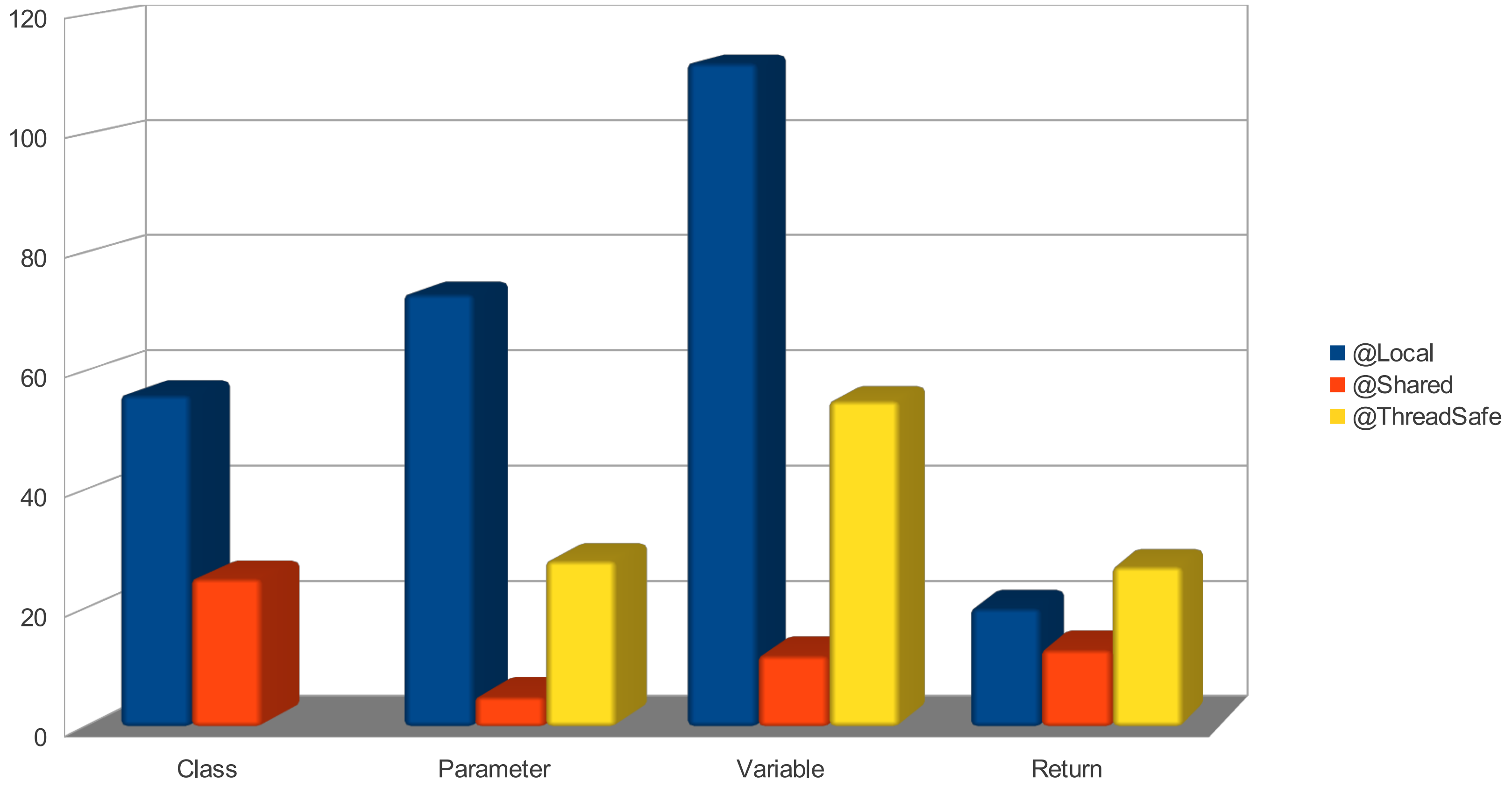
Evaluating Loci

- We annotated over 50000 LOC.
- 262 classes and 13 interfaces
- We chose heavily multi-threaded Java benchmarks.
- Programs from DaCapo and JavaGrande Benchmark suite.
- Less than 15 annotations/KLOC

UPPSALA UNIVERSITET

Informationsteknologi

Evaluating Loci, *Cont'd*



UPPSALA UNIVERSITET

Informationsteknologi

Conclusions

- Loci:
  - Is a useful aid for programmers.
  - Is compatible with existing Java programs.
  - Eliminates thread-locality violations.
  - Requires a low annotation overhead.
  - Is a bit slower than normal **javac** (6 sec vs 45 on my machine) when compiling ~46 KLOC.
  - Has five known bugs!

Future Work

- Supporting object transferring across threads without the need of deep cloning.
- Having cross thread-locality cloning.
- Fixing the bugs that we have.
- Speeding up the tool.

UPPSALA UNIVERSITET

Informationsteknologi

Related Links

- Homepage: http://www.it.uu.se/research/upmarc/loci
- Wiki: http://java.net/projects/loci/pages/Home
- Forum: http://java.net/projects/loci/forums
- Mailing List: http://java.net/projects/loci/lists/
- Repository: http://java.net/projects/loci/sources
- Bugzilla: http://java.net/bugzilla/buglist.cgi?product=loci
- Manual: http://loci.java.net/manual
- Download: http://java.net/projects/loci/downloads

UPPSALA UNIVERSITET

Informationsteknologi

References

- The Java Grande Forum Multi-threaded Benchmarks.
- S. M. Blackburn et 1l. “The DaCapo benchmarks: Java benchmarking development and analysis”, OOPSLA ’06.
- G. Bracha, “Pluggable type systems”, OOPSLA04.

UPPSALA UNIVERSITET

Informationsteknologi

References, *Cont'd*

- T. Wrigstad et al. “Loci: Simple thread-locality for java”, ECOOP 2009.
- W. Dietl et al. “Building and using pluggable type-checkers” , ICSE’11.

UPPSALA UNIVERSITET

Informationsteknologi

Thank You, Questions?