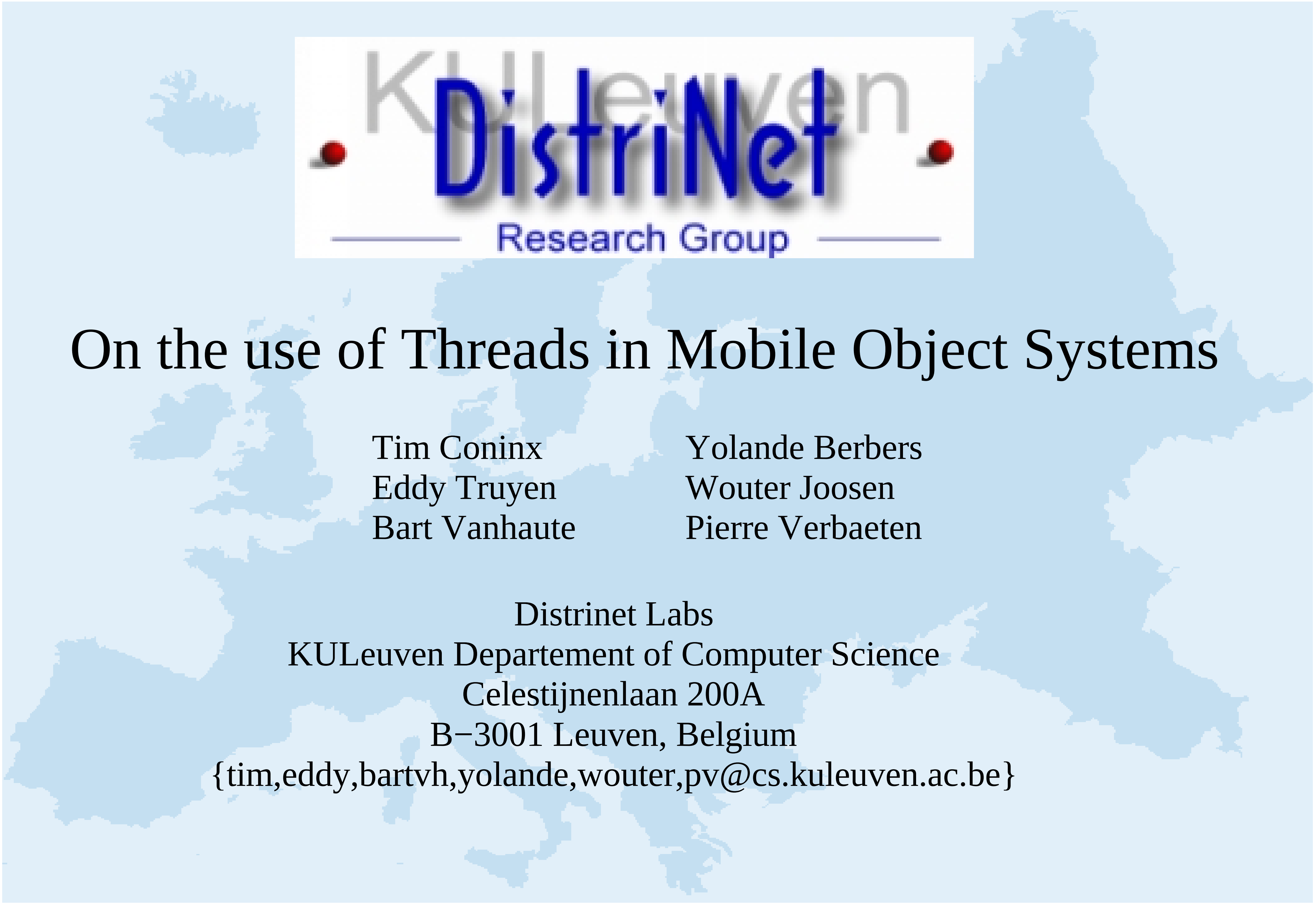


On the use of Threads in Mobile Object Systems

Tim Coninx
Eddy Truyen
Bart Vanhaute
Yolande Berbers
Wouter Joosen
Pierre Verbaeten

Distrinet Labs
KULeuven Departement of Computer Science
Celestijnenlaan 200A
B−3001 Leuven, Belgium
{tim,eddy,bartvh,yolande,wouter,pv@cs.kuleuven.ac.be}

Overview

- Problem Statement
- Transparent Thread Migration
- Distributed Tasks
- Status
- Future Work

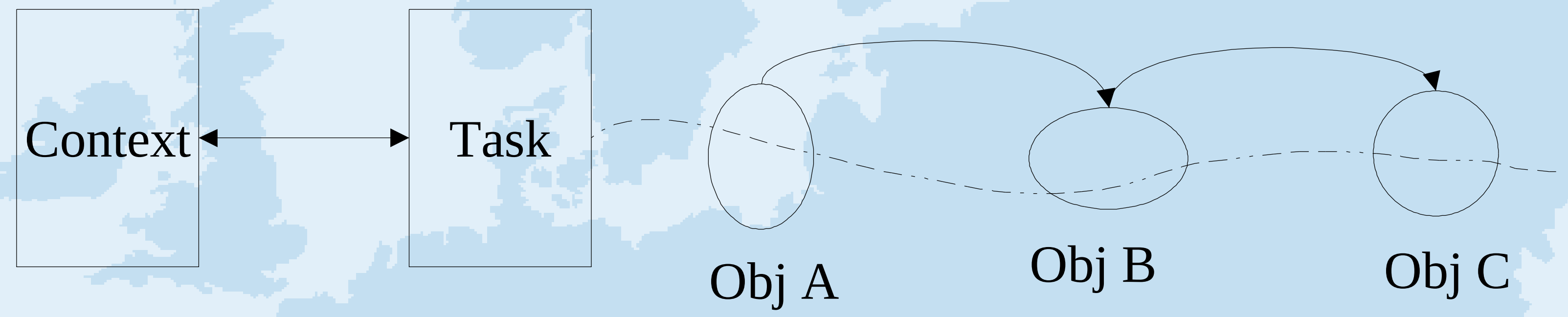
Problem Statement

- How to add migration to distributed applications
  - Transparent for the application programmer
  - Without altering the JVM
  - At Runtime !!

Problem : Dynamic Partitioning

- Object oriented distributed application = large population of fine−grained objects
- Object grouping at runtime to minimize network communication
- Groups of objects can, while executing, be transferred to other locations

Transparent Thread Migration

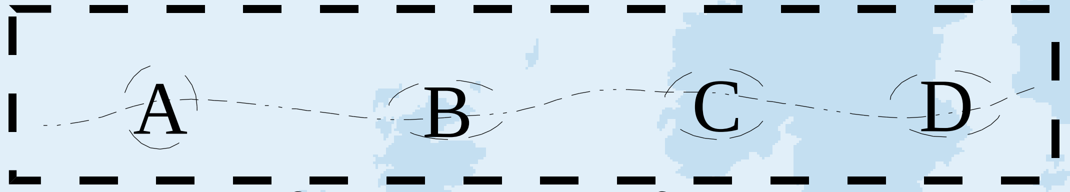


Possible to suspend and reestablish the Java thread

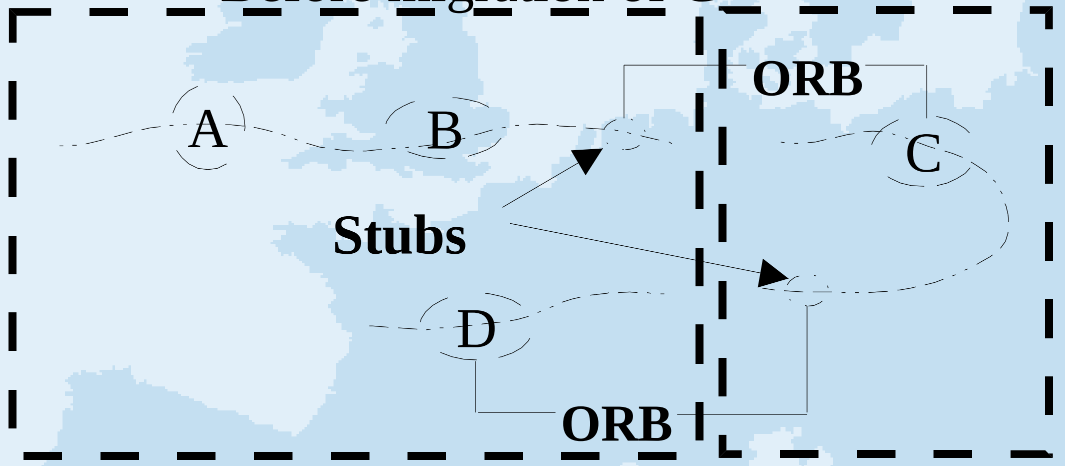
Transparent Thread Migration

- Classes are instrumented by a byte code transformer
  - capturing blocks after every method invocation
  - restoring blocks at the beginning of each method
- Independent of the JVM
- Independent of the original source

Distributed Tasks



Before migration of C

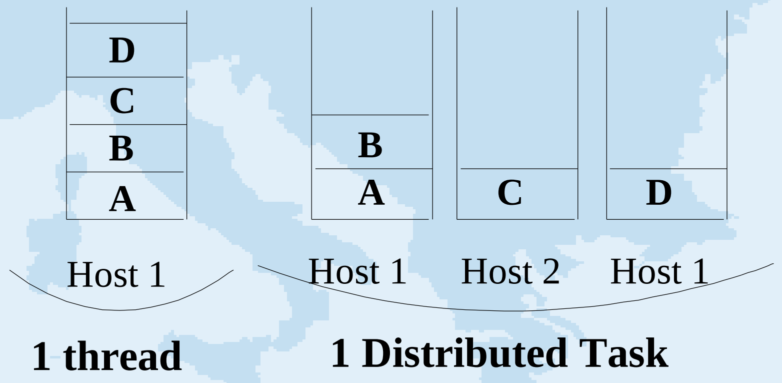


After migration of C

- Problem: Object C wants to migrate
- Thread executing in C is stopped and serialized
- C is migrated and the thread is restarted

Distributed Tasks

- During Reestablishment of the thread
  - split up into three different threads
  - form one logical whole : a Distributed Task
  - object references
    - local reference : restored
    - remote reference : replaced by stub



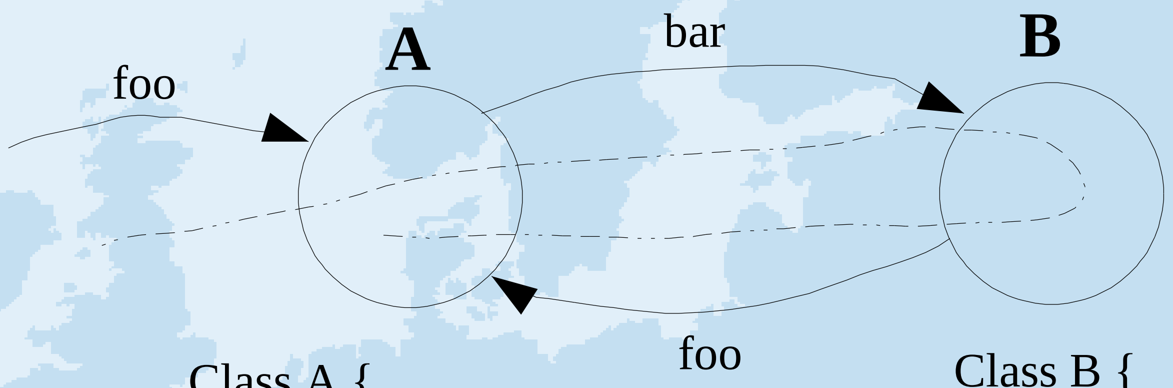
Status of TTM

- Transparent thread migration is more
  - portable than JVM−changing techniques
  - performant than source−changing techniques
- However : mind the bytecode
  - classfile size blowup
  - altering (corrupting) program flow

Future Work

- Framework using distributed tasks
  - TTM is not the only answer
  - Runtime system has to take care of
    - Fault Tolerancy
    - Resource Management
    - Reference Management
    - ...

Problem : Distributed Locks



```
Class A {
        Synchronised void foo() {
                ...
                b.bar(this);
        }
}
```

```
Class B {
        void bar(A a) {
                a.foo();
        }
}
```

Use of a Global Thread Identifier to counter locking problems