Mobile Threads: a paradigm for complex distributed applications

Ing. Raffaele Quitadamo

Agent and Pervasive Computing Research Group

quitadamo.raffaele@unimore.it

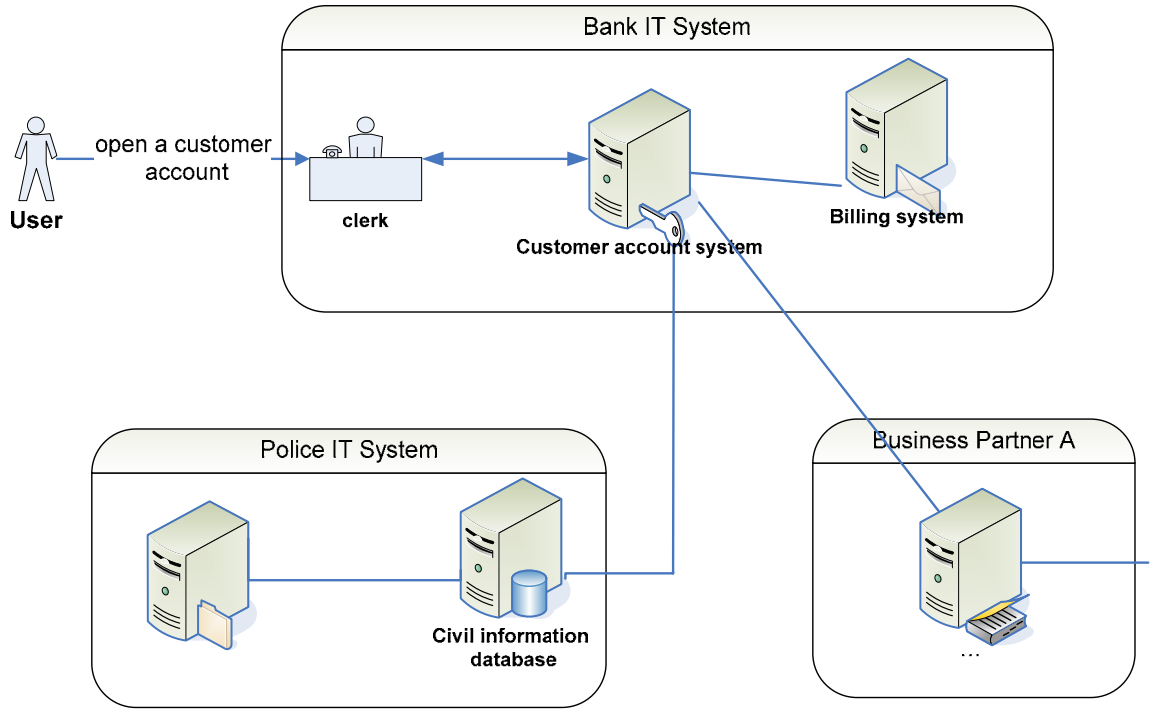
www.agentgroup.ing.unimo.it/Quitadamo

What am I talking about?

- What choices do you have to design a distributed application?
- Do Web Services fit the needs of complex distributed systems?
- What is mobile code?
- How was it conceived?
- Mobile Threads/Processes as “relocatable computations”
- My PhD research: implementing Mobile Threads on top of the Java Virtual Machine

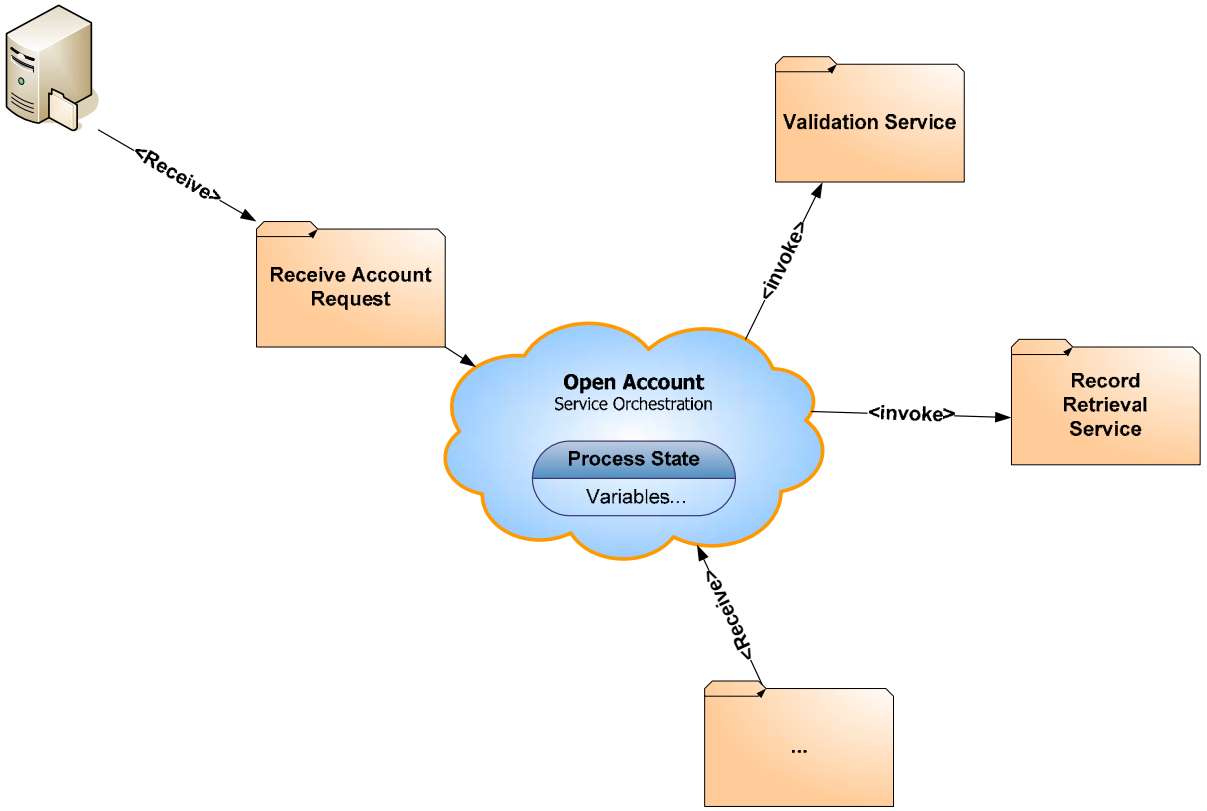
Distributed Applications

- Let's suppose you are working for a bank customer and they told you to implement an application to open a bank account



- How many different institutions/information systems are involved?
- My bank, the police system, and possibly other partners banks

Web Services: a well-known choice



- The different systems may provide a set of services that other clients may exploit to build their business processes.
- The process is orchestrated by a single fixed host, which usually keeps the process state (i.e. variables)

Three different possibilities

more challenging

- Distributed client/server computations: CORBA, DCOM, Web Services…
- REV (Remote EValuation): a component A has the know-how (i.e. code) to perform the service but it lacks the resources. Consequently, A sends the code to a computational component B located at a remote site. B executes the code using the resources available and delivers the results back to A.
- COD (Code On Demand): component *A* is already able to access the resources it needs located at *SA*. However, no information about how to manipulate such resources is available. Thus, *A* interacts with a component *B* by requesting the code. E.g. Java Applets.
- Mobile Agents/Threads ?

Fuggetta, G. P. Picco, G. Vigna, “Understanding Code Mobility”,
IEEE Transactions on Software Engineering, Vol 24, 1998

Threads as movable units of computation

- In your past courses you have learned about "Processes" (Sistemi Operativi), "Java Threads" (Principi di Sistemi Operativi)

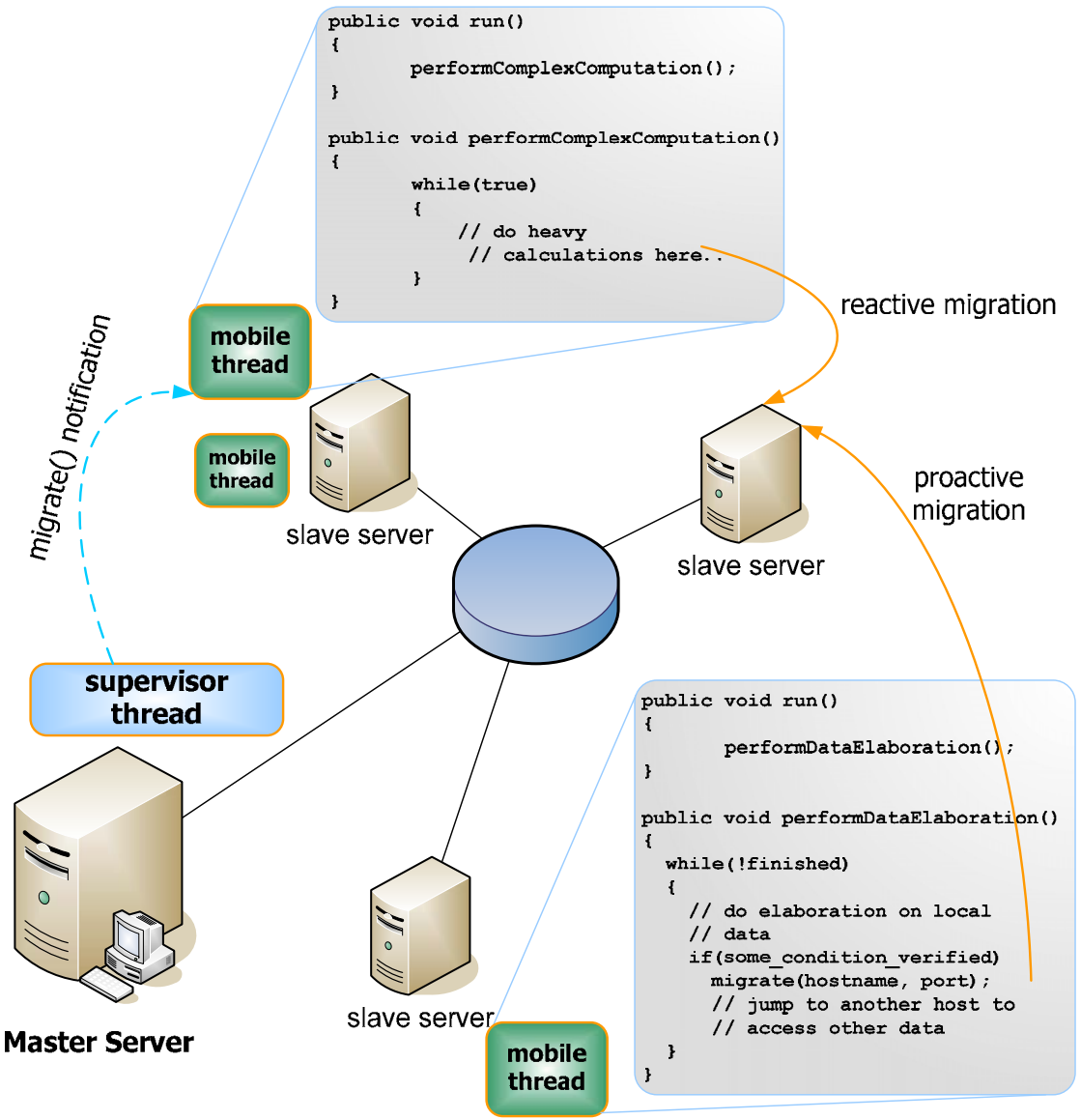
- The idea of Mobile Threads (strong mobility):

"detaching as much as possible a running computation (i.e. a thread) from the underlying execution environment (e.g. CPU, memory layout, file systems, …)"

- Java offers some interesting features (platform independence, dynamic class loading)

- So, we have to know something about the internals of the JVM (Java Virtual Machine)

Why thread mobility?



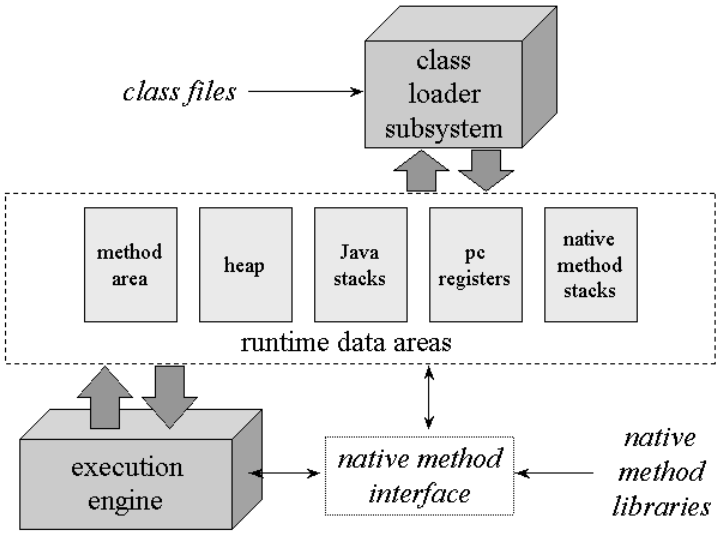
The Java Virtual Machine

JVM = abstract representation of a CPU, executing instructions in an intermediate language (bytecode), standing between Java and the assembly language.

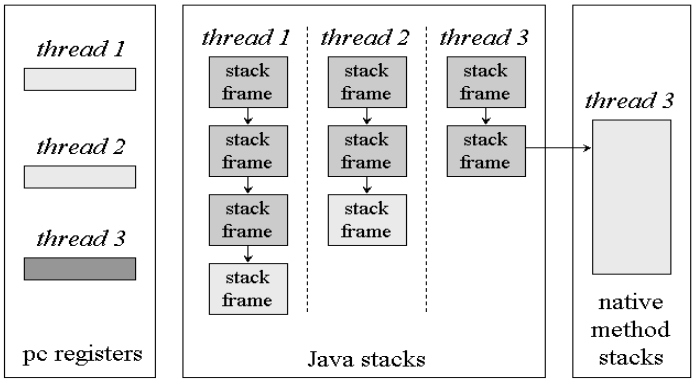
Every application is executed by one or more concurrent Java threads.

Outstanding structures:

1. A main thread is initially started by the JVM to run **`main()`**; other threads created on demand (**`new Thread()`**...)
2. One stack per thread, with its context registers (e.g. program counter, stack pointer, machine registers)
3. A native stack to invoke legacy methods (e.g. C/C++ functions, dll), through JNI (Java Native Interface)



The call stack



- The JVM is a stack-based machine
- The stack is made up of **frames or method activations**
- Every **frame** is a snapshot of the current state of the method
- When a method is called, a new frame is push on top of the stack and, when it returns, the frame is popped out.

```
class Example {

int runInstanceMethod(char c,
double d, short s, boolean b){

        // method code here

}

}
```

runClassMethod()

index	type	parameter
0	int	int i
1	long	long l
3	float	float f
4	double	double d
6	reference	Object o
7	int	byte b

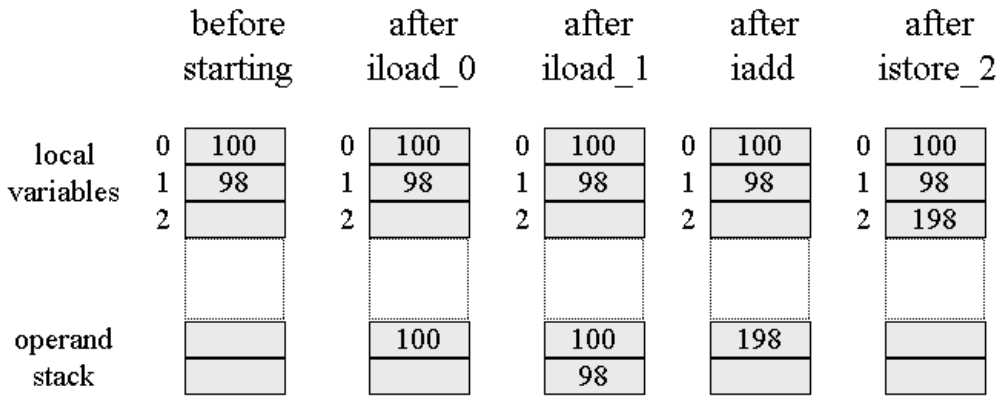
runInstanceMethod()

index	type	parameter
0	reference	hidden this
1	int	char c
2	double	double d
4	int	short s
5	int	boolean b

The call stack (2)

<Adding two integers>

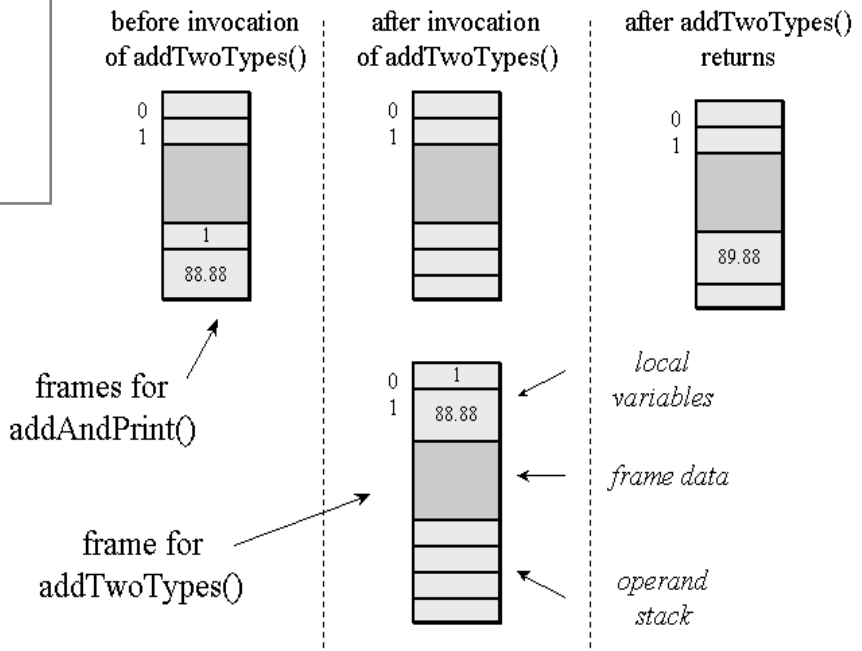
```
iload_0
iload_1
iadd
istore_2
```



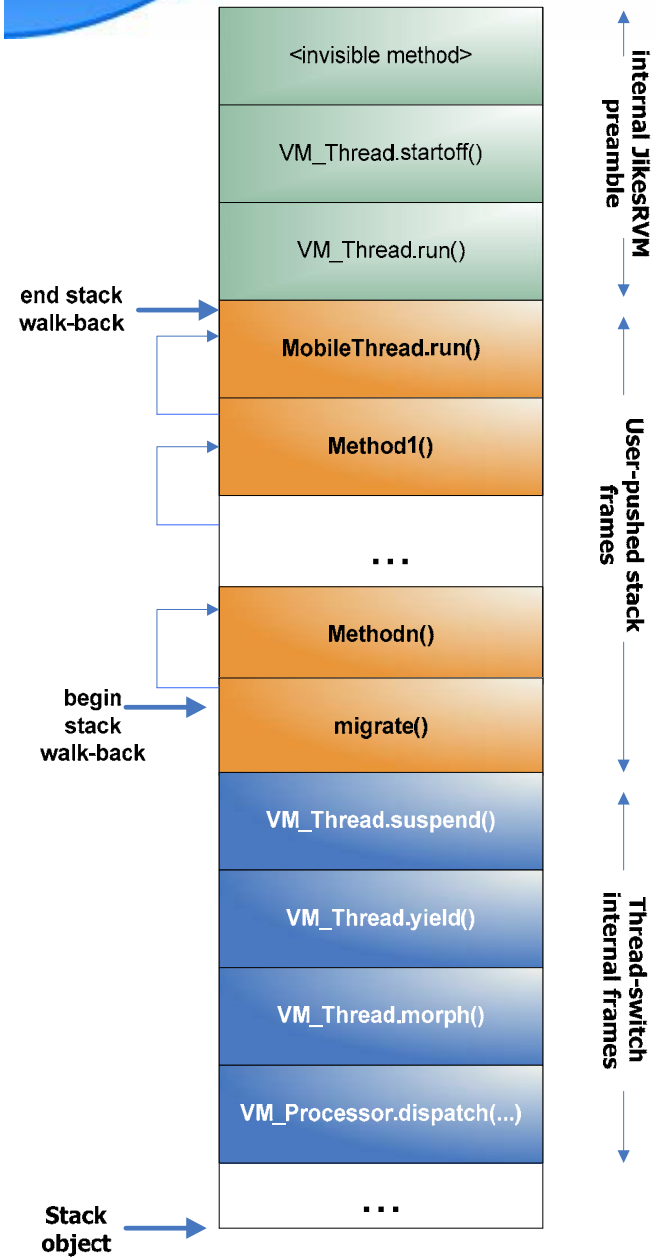
- Operations with variables (e.g. assignments, arithmetics, method calls) use the expression stack for portability reasons
- Optimizing JIT compilers often violate these rule and use registers to reduce memory access latency

Passing parameters to methods

```
class Example3c {
public static void addAndPrint(){
   double result;
   result = addTwoTypes(1, 88.88);
}
static double addTwoTypes(int i, double d)
{
   return i + d;
}
}
```



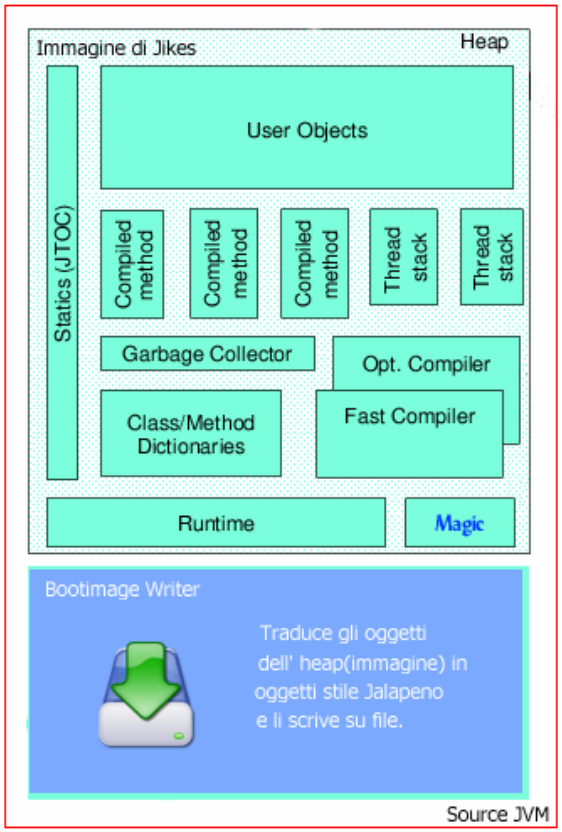
Migrating a whole thread



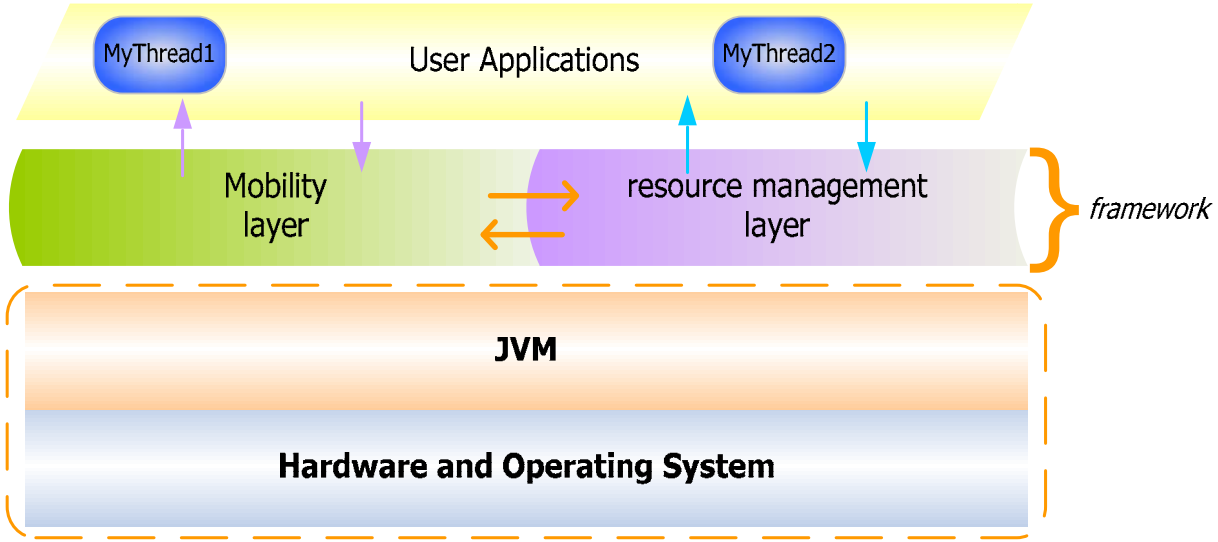
- At time *t*, the current execution of the thread is represented by the set of all frames into the stack.
- We need therefore to
  1. capture each of these frames
  2. move them on another host
  3. reestablish them into a new stack
- The new thread can be resumed and will continue from the next instruction.
- Conventional JVMs (e.g. SUN) do not allow these things, because they carry out strong optimizations and pose security
- We adopted a research JVM from IBM T. J. Watson Research Center

IBM Jikes Research Virtual Machine

- JikesRVM was born in the research laboratories at IBM Watson Research Center (1997)
- Its peculiarity is that it is written almost completely in Java.
- It is now an open-source project, available at http://jikesrvm.sourceforge.net
- It was adopted by many researchers around the world to make their experimentations



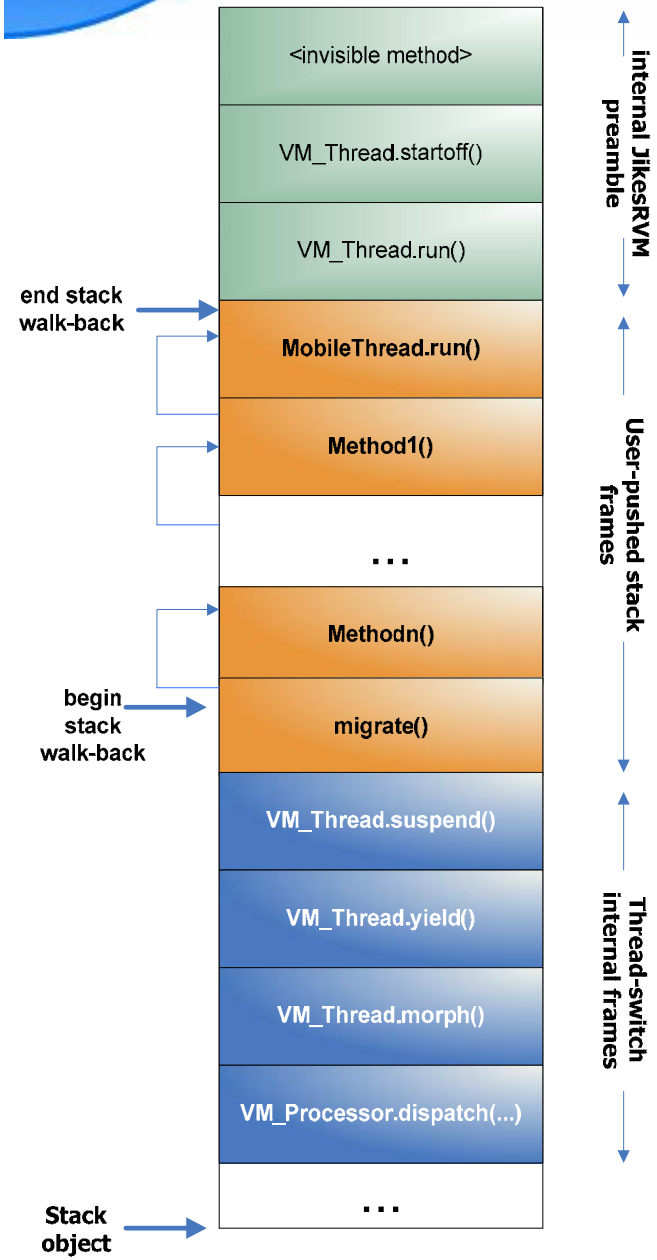
Mobile JikesRVM



- Research (and implementation) work focused on both

1. the mobility layer, contains classes to capture the execution state of a running Java Thread in a bytecode-level portable format

2. the resource management layer, contains a policy registry where programmers can register their resource relocation strategies



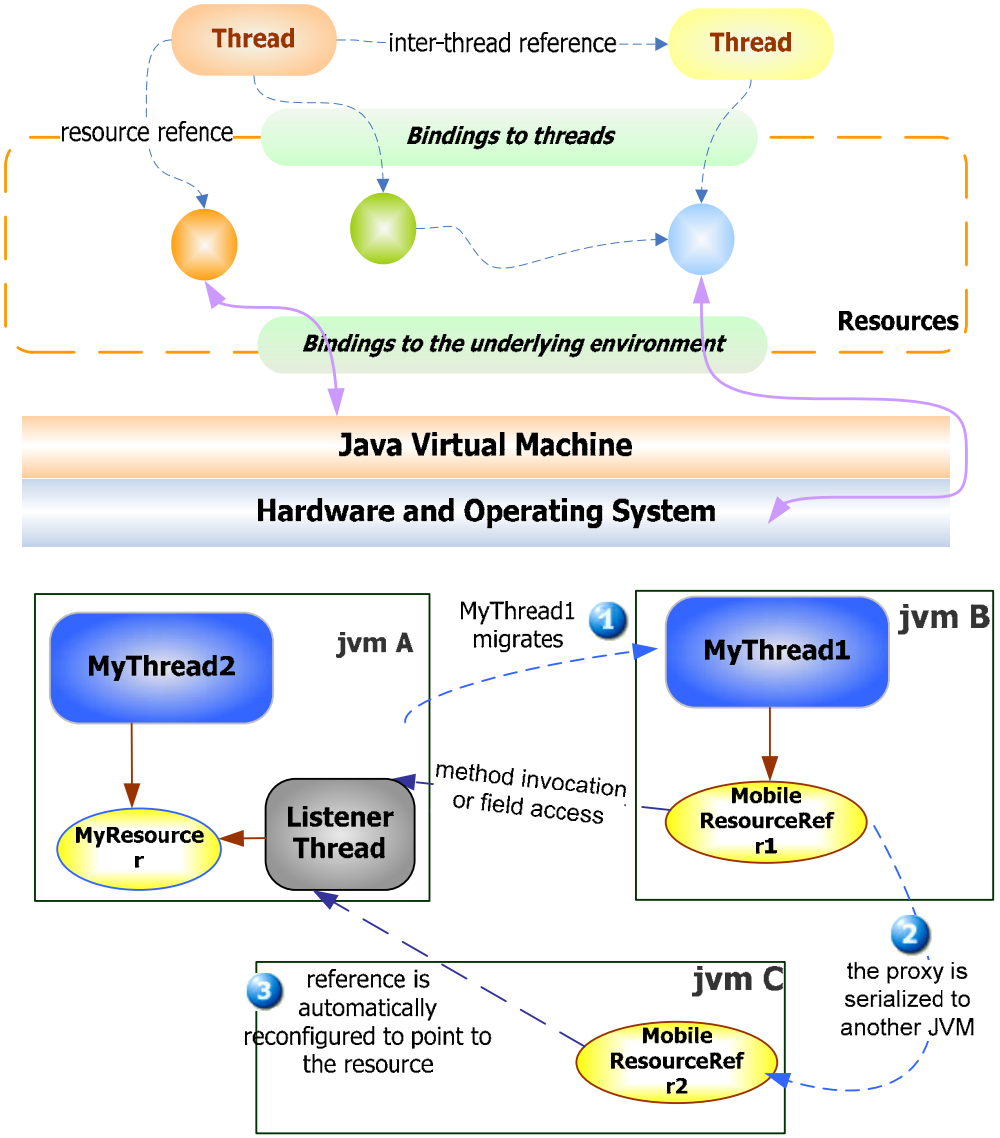
Mobility Layer

- A FrameExtractor component is used to analyze the stack at any moment and extract each method activation
- Every frame is extracted as a MobileFrame, to be platform-independent and thus portable!

New features added:

1. **Reactive migration is fully supported** (thanks to special migration points inserted into the JIT compiled code)
2. **The system's been ported to PowerPC Mac OS** 10.4, so that a Java Thread, born on an IA32 machine, can be migrated and resumed on a PPC32 machine.
3. **The optimizing JIT compiler was patched to allow optimized method frames to be captured** (OSR_Points inserted at call sites to build maps of the frame structure, registers, inlinee, etc...)

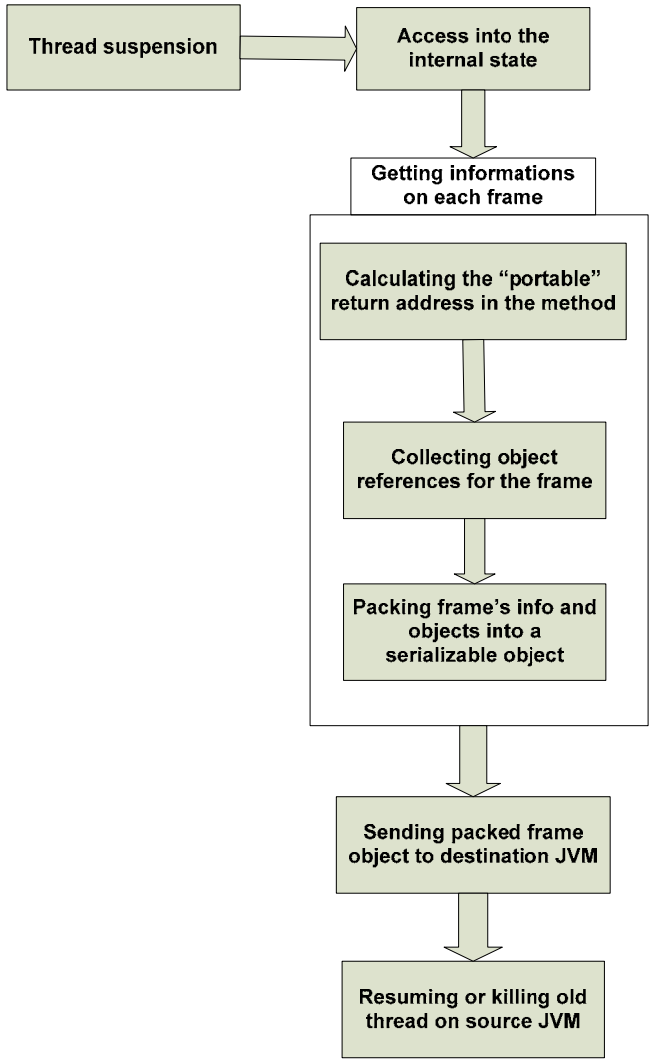
Resource Management



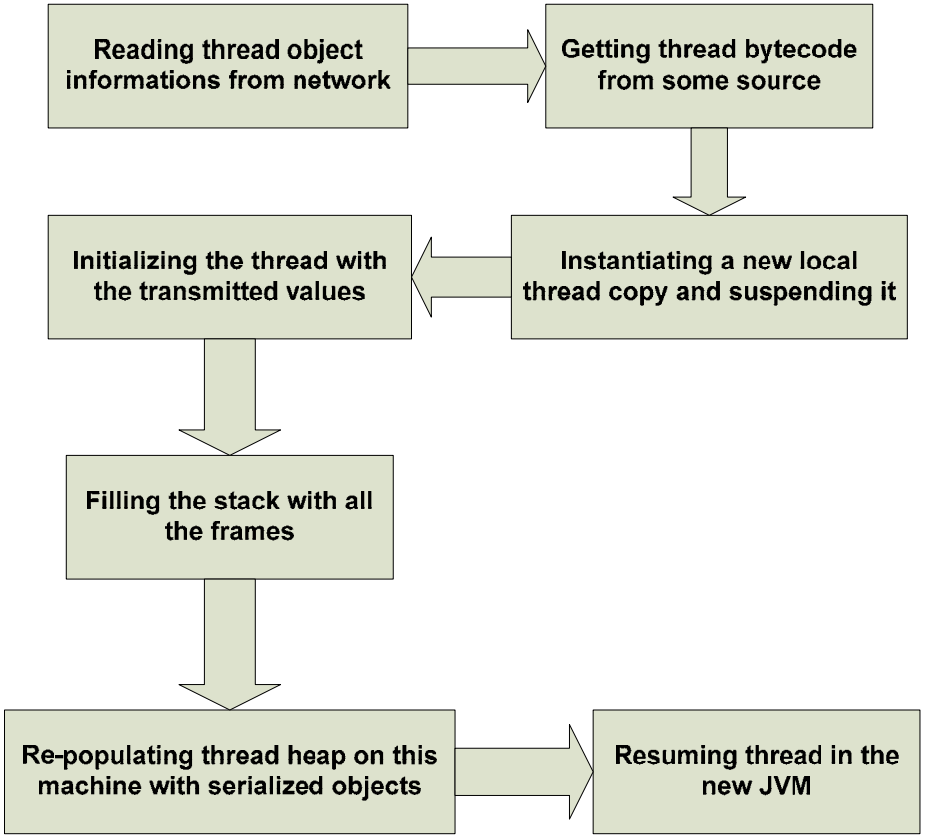
- **Resource relocation policies** to customize the relocation of object upon migration
- All the policies identified in literature are supported: **by copy, by move, by network reference**
- **Network reference policy implemented thanks to the JIT compiler** (getfield, putfield, invokevirtual, invokespecial bytecodes)
- Proxy to references are migrated instead of the real resource
- **Different from RMI**, mainly because of the high transparency

An Overview of the Entire Thread Migration Process

Serialization



Deserialization



Future work

• **Mobile JikesRVM is currently used at Anhui University of Science and Technology (China) and Multimedia University** (Malaysia)

In my third Ph.D. year (after Easter), I will

• **Integrate Mobile JikesRVM into the NOMADS platform for Agile Computing (prof. Niranjan Suri)**, spending a 6 months research period at IHMC (Institute for Human and Machine Cognition), Pensacola, **Florida, US**.

Thanks for your attention!