Symbolic Computation in Java: an Appraisement

Laurent Bernardin
ETH Zurich
www.inf.ethz.ch/personal/bernardi

Joint work with: Bruce Char (Drexel University)
Erich Kaltofen (North Carolina State U)

Java advantages

- GUI
- Distributed computing (Schreiner 99, Masdis)
- Visual programming (MathBeans)
- Black-box functions (Sandbox, Melissa)
- Serialization standard (PDG Openmath)
- Standard libraries
- Platform independence

2nd Edition
Ignores Java 1.1

JAVA IN A NUTHOUSE

A Desktop Quick Irreverence to Microsoft Java

O'REILLY

Kevin McCurley

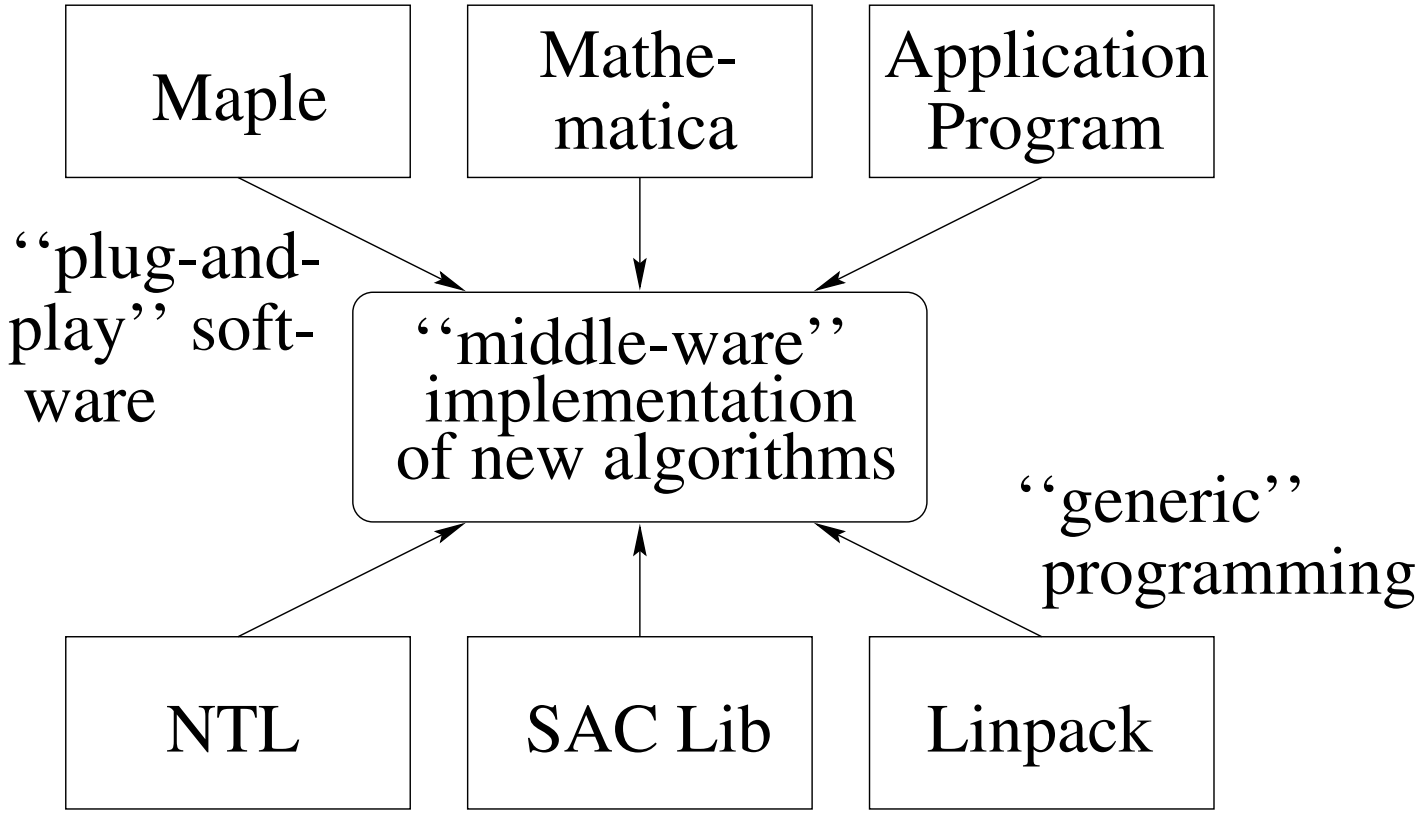
Is Java fit for large-scale computing?

- Large and complex programs

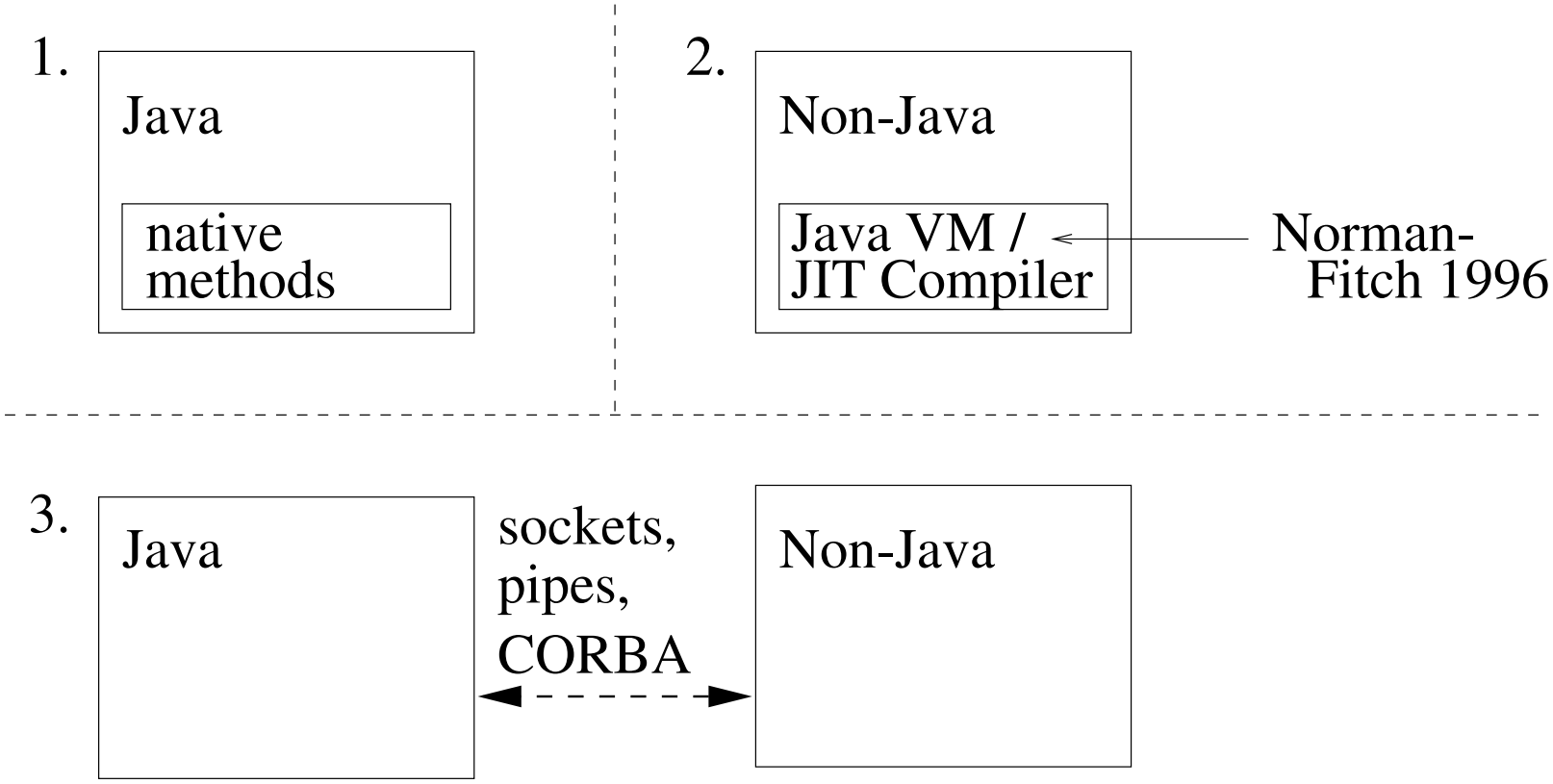
- High performance calculations

Component-wise system design

Plug-and-play components ($\longrightarrow$conglomerates, PSEs)



Interfaces between Java/non-Java components

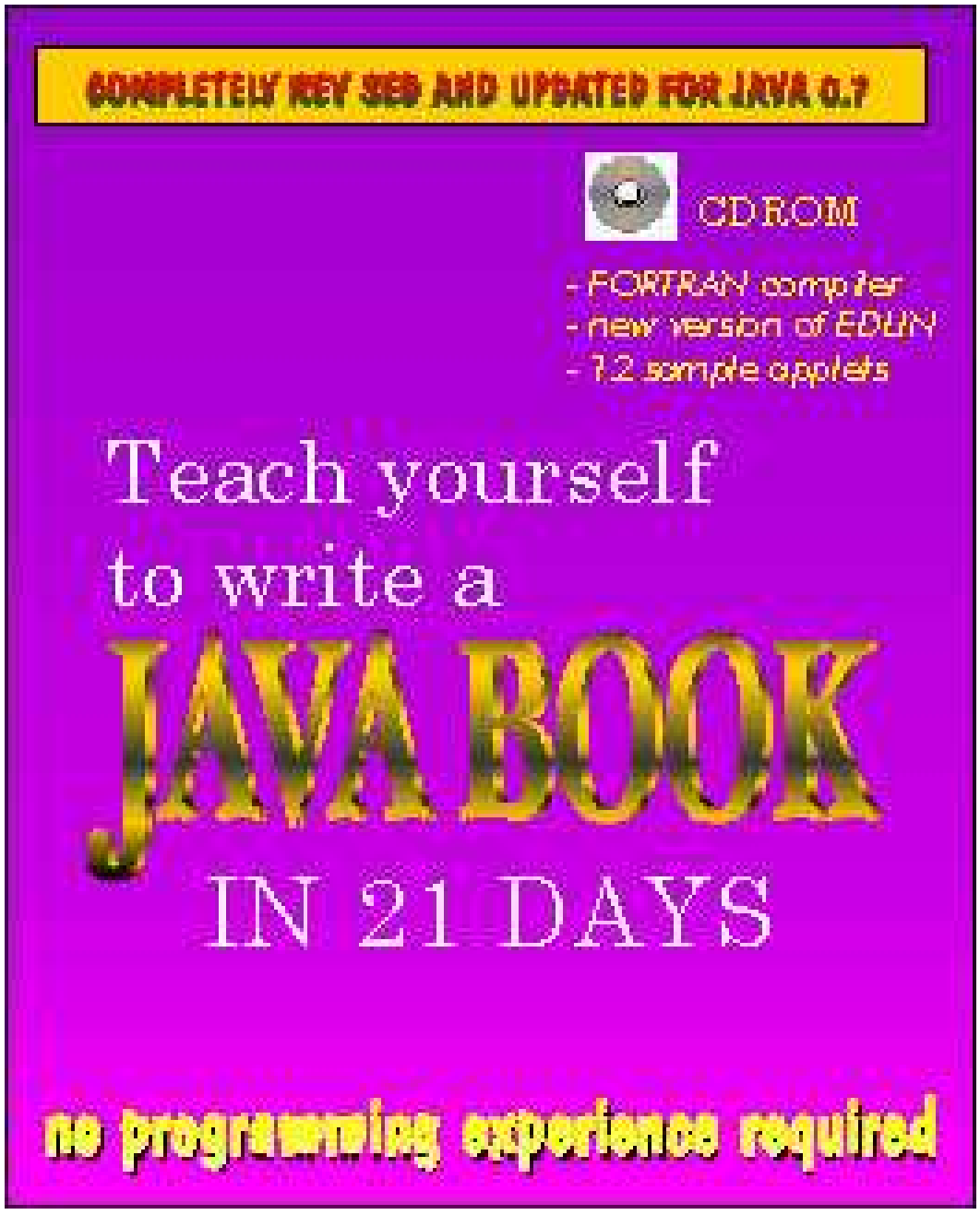


Generic Programming

- static (compile-time) binding: templates (C++, GJ, NextGen)
  performance
- dynamic (run-time) binding: interfaces (Java)
  type-safety
- C++-style templates: GJ/NextGen
  Are templates really needed?
- STL-style allocators as interface "glue" for divergent storage models
- Algorithmic shortcuts into the basic modules
  partial template specialization

```
public interface Ring {
        public interface Element {
                public boolean iszero();
                ...
                }
        public Element fromInteger(int n);
        public Element add(Element a, Element b);
        public Element multiply(Element a,Element b);
        ...  }


public class DensePolynomial implements Ring {
        public class Element implements Ring.Element {
                private Ring.Element[] _coeffs;

                public int degree() {
                        return _coeffs.length-1; }
                ...
                }// end class Ring.Element

        private Ring _R;

        public Element add(Ring.Element a, Element b) { ... }
        }
```

```
Ring R = new Zmod(17);
Ring P = new DensePolynomial(R);
DensePolynomial.Element p = P.add( P.monomial(10),
                                   P.one() );
DensePolynomial.Element q = P.add( P.monomial(3),
                                   P.one() );
p = P.multiply(p,q);
```



(Kevin McCurley)

Performance considerations

- Garbage collection
- Interpreted bytecode / JIT
- Benchmarks

In-Place Polynomial Arithmetic

- Multiply degree n polynomials over $\mathbb{F}_{17}$
- Dense array of hardware integers
- Java: Sun JDK 1.2 (beta 5)
- Maple: modp1 datastructure, R5
- C: Sun Workshop 4.2, flags: -O
- C*: flags: -native -fast -xO4

n	Java	Maple	C	C*	Java vs. C*
10000	7s	9s	9s	3s	2.33
20000	30s	37s	36s	13s	2.31
30000	69s	77s	82s	31s	2.23
40000	124s	137s	146s	56s	2.21
50000	196s	218s	231s	89s	2.20

Generic Polynomial Arithmetic

- Multiply degree n polynomials over a generic ring
- Particular ring is $\mathbb{F}_{17}$ again
- Aldor: 1.1.10b + $\sum^{\text{it}}$
- C++: gcc 2.8.1, templates

n	Java	Aldor	C++	Java vs. C++
1000	2.6s	1.2s	0.9s	2.8
2000	9.1s	4.3s	3.6s	2.6
3000	20s	10s	8s	2.5
4000	36s	19s	14s	2.6
5000	57s	30s	22s	2.6
6000	82s	42s	31s	2.6
7000	111s	56s	42s	2.6
8000	146s	72s	57s	2.6

Conclusions

- Java as component glue and for GUI code
- Performance is improving (Compiler technology)
- Template facilities are being missed