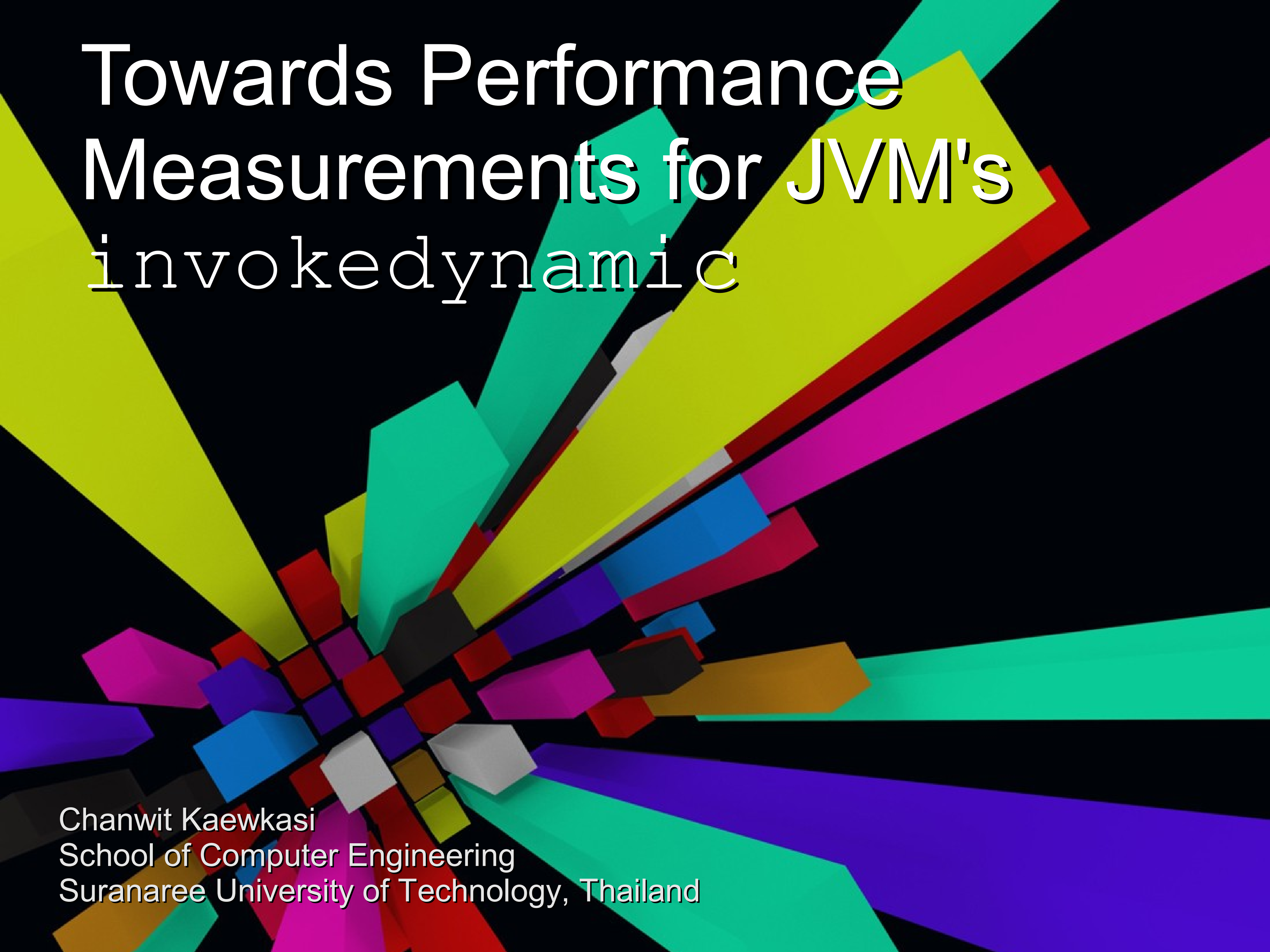
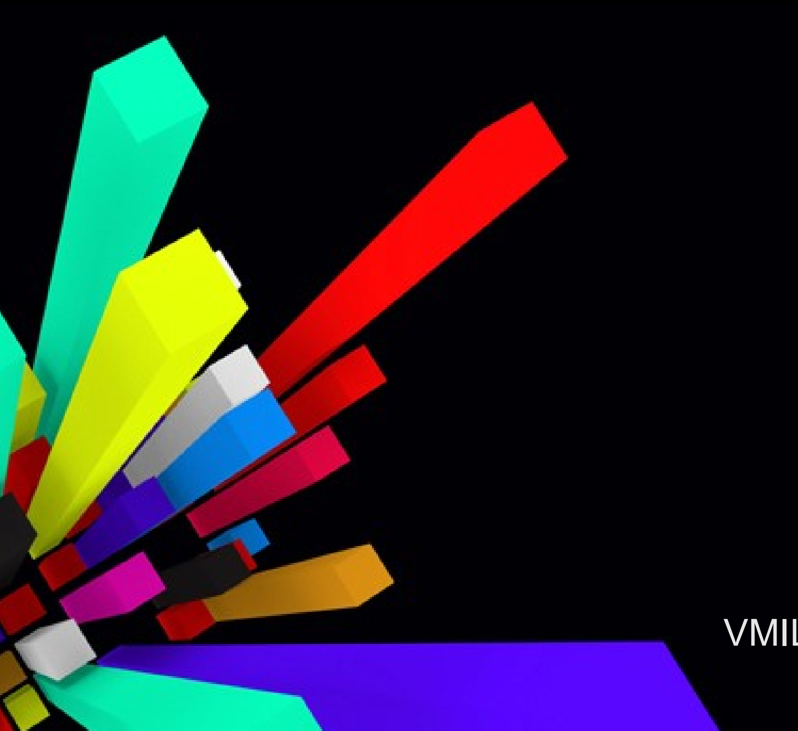
Towards Performance Measurements for JVM's `invokedynamic`



Chanwit Kaewkasi
School of Computer Engineering
Suranaree University of Technology, Thailand

Motivation

- Early performance report of `invokedynamic` is > 10 times faster than reflection [1].



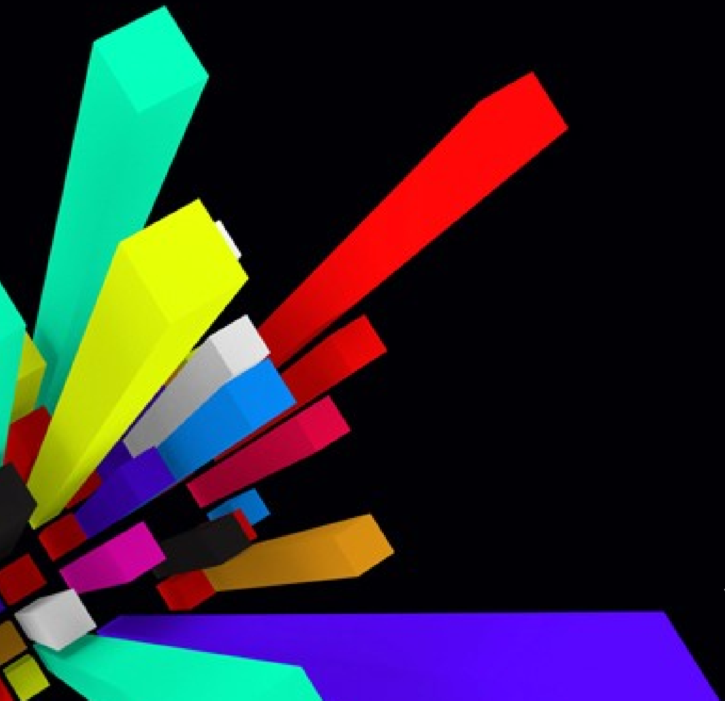
[1] http://www.mail-archive.com/mlvm-dev@openjdk.java.net/msg01816.html

Contributions

- Performance reports on `invokedynamic`
  - Other contributions
    - Rules for binary translation from invoke instructions to `invokedynamic`.
    - An identification of a potential bottleneck in the server VM.
    - A limitation of the bytecode verifier when taking `invokedynamic` into account.

Review `invokedynamic`

- Driven by JSR-292
- A prototype called the Da Vinci Machine
  - a.k.a. Multi-language virtual machine (MLVM)



Review **`invokedynamic`** (2)

- Bytecode **`invokedynamic`**
  - A 5-byte instruction
  - No scope type, use only name-and-type
  - Designed to be a replacement of all other invoke instructions

Review `invokedynamic` (3)

- Bootstrap Method
  - Accept name-and-type information of the current invokedynamic instance.
  - Create a call site object, based on a method handle.

Review **`invokedynamic (4)`**

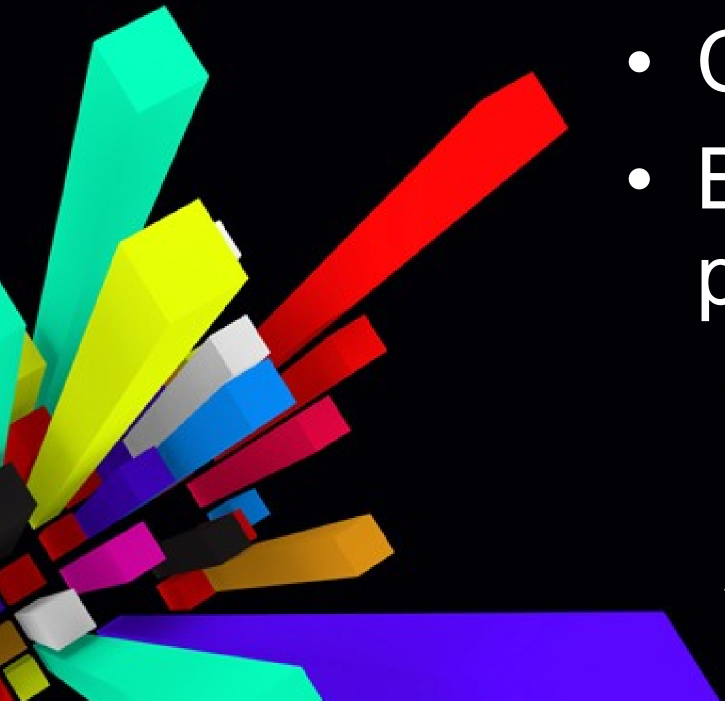
- Method Handles
  - A lightweight structure for invoking JVM methods.
  - Only Direct Method Handles were used in the experiment to reduce run-time overheads in the experiments.

SciMark 2.0

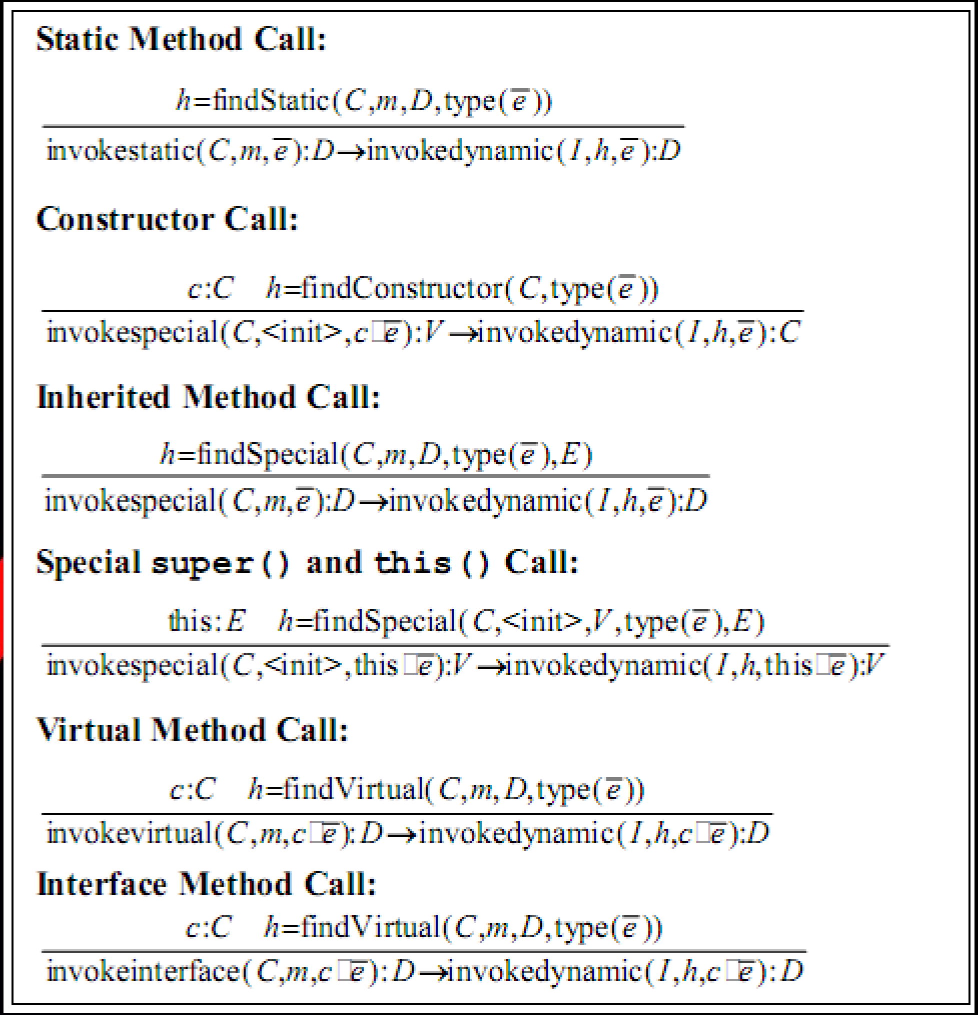
- A set of numerical micro-benchmarks.
  - Simplicity

    Allow manually refactoring to avoid the MLVM's limitation.
  - CPU-bound benchmarks.
  - Benchmarks mainly contain primitive operations:

    To see how good MLVM inlines trivial final methods.



Translation Rules



Static Method Call:

$$\frac{h = \text{findStatic}(C, m, D, \text{type}(\overline{e}))}{\text{invokestatic}(C, m, \overline{e}){:}D \to \text{invokedynamic}(I, h, \overline{e}){:}D}$$

Constructor Call:

$$\frac{c{:}C \quad h = \text{findConstructor}(C, \text{type}(\overline{e}))}{\text{invokespecial}(C, \text{<init>}, c\,\square\,\overline{e}){:}V \to \text{invokedynamic}(I, h, \overline{e}){:}C}$$

Inherited Method Call:

$$\frac{h = \text{findSpecial}(C, m, D, \text{type}(\overline{e}), E)}{\text{invokespecial}(C, m, \overline{e}){:}D \to \text{invokedynamic}(I, h, \overline{e}){:}D}$$

Special `super()` and `this()` Call:

$$\frac{\text{this}{:}E \quad h = \text{findSpecial}(C, \text{<init>}, V, \text{type}(\overline{e}), E)}{\text{invokespecial}(C, \text{<init>}, \text{this}\,\square\,\overline{e}){:}V \to \text{invokedynamic}(I, h, \text{this}\,\square\,\overline{e}){:}V}$$

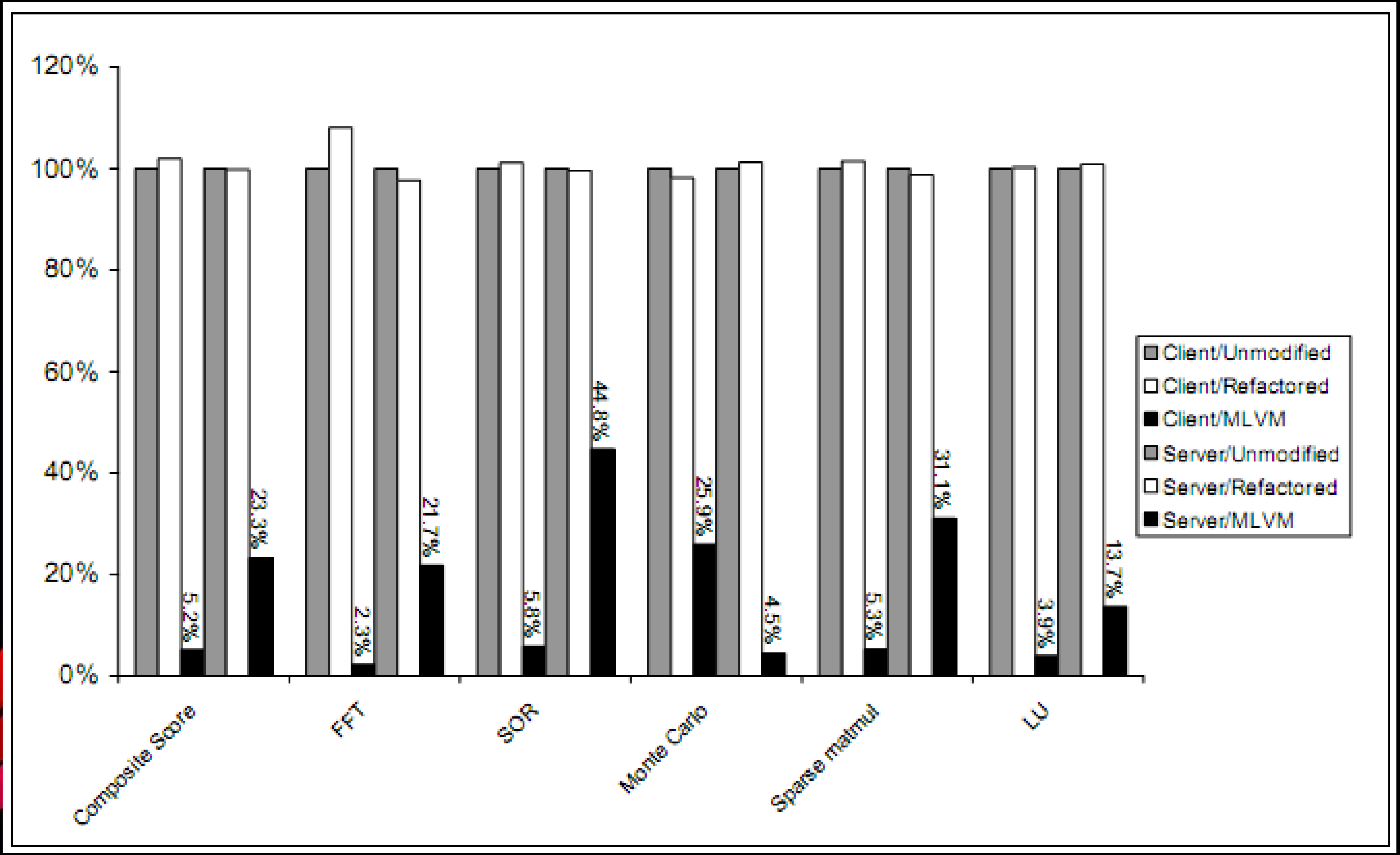
Virtual Method Call:

$$\frac{c{:}C \quad h = \text{findVirtual}(C, m, D, \text{type}(\overline{e}))}{\text{invokevirtual}(C, m, c\,\square\,\overline{e}){:}D \to \text{invokedynamic}(I, h, c\,\square\,\overline{e}){:}D}$$

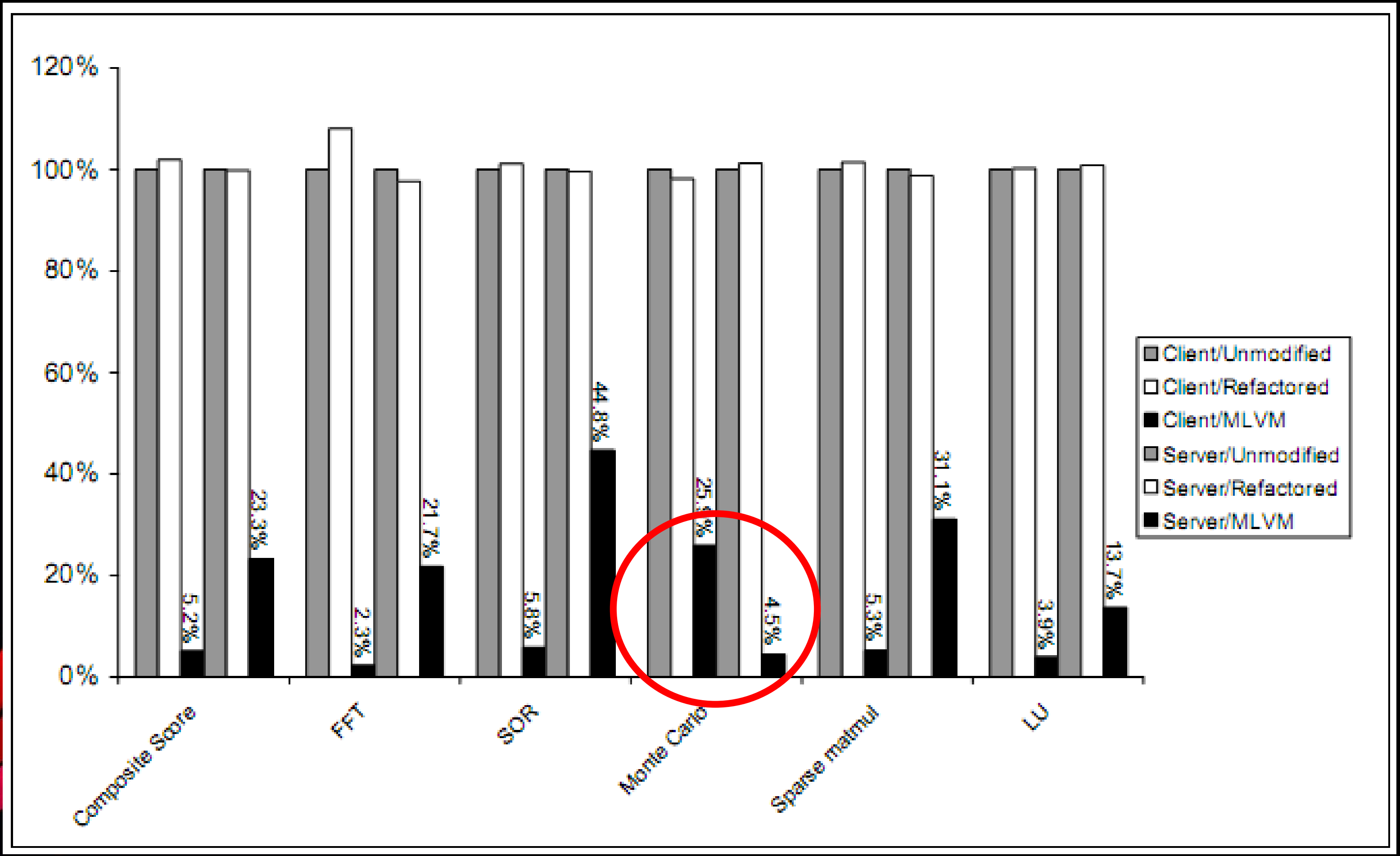
Interface Method Call:

$$\frac{c{:}C \quad h = \text{findVirtual}(C, m, D, \text{type}(\overline{e}))}{\text{invokeinterface}(C, m, c\,\square\,\overline{e}){:}D \to \text{invokedynamic}(I, h, c\,\square\,\overline{e}){:}D}$$

Performance Results



Performance Results



Performance Results

- `invokedynamic` on the server VM is 2-5 times slower than Java native invocations, except the Monte Carlo benchmark.
  - Performance is still not that good on the client VM.

Performance Results

- A bottleneck identified by the Monte Carlo benchmark.
  - Christian Thalinger suspected there may be deopts somewhere [2].

[2] http://www.mail-archive.com/mlvm-dev@openjdk.java.net/msg01923.html

Implementation Notes

- Using `findSpecial` is not allowed to obtain a method handle for `<init>`.
  - `findConstructor` is for `super.m(T)`, not `super()`

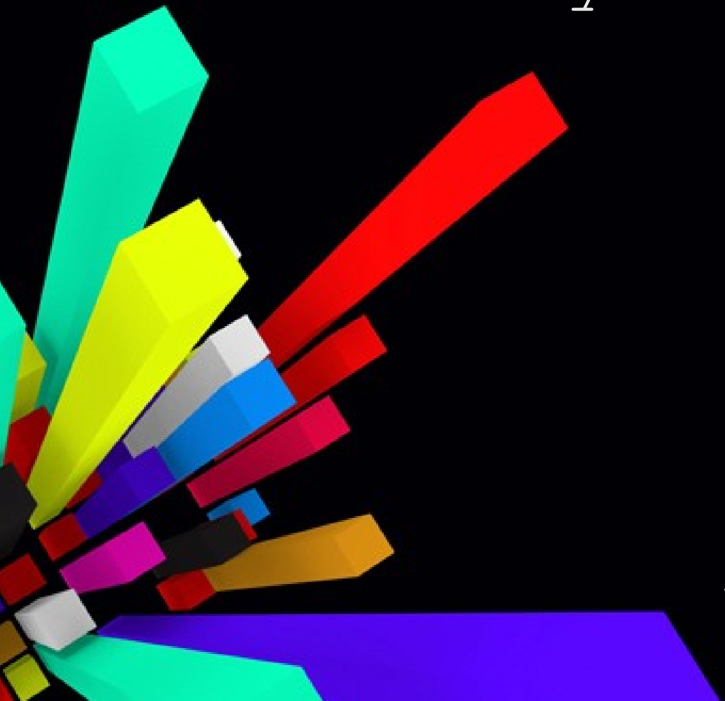
Implementation Notes

- Bytecode verifier rejects the program when `invokedynamic` is in a constructor body
  - John Rose mentioned [3] that JSR 292 EG discussed the similar issue and led to an idea of supporting Categorical Subclasses.

[3] http://permalink.gmane.org/gmane.comp.java.vm.languages/2429

Future Work

- Other numerical benchmark suites
  - Micro-benchmarks are still required.
- Real-world benchmark suites
  - `invokedynamic` DaCaPo is on its way.



Summary

- A new kind of benchmark suites is required to measure performance for this new invocation mode.
- `invokedynamic` is on its way, and available now in JDK 7.
- `invokedynamic` is not that slow, and will be faster.

Thank you very much!