Reflection benchmark

This paper covers the standard object access vs reflection access in Java. Testing was made in the following order:
- 20 runs with 100 000 calls each - warm up phase,
- actual long test run with 2 000 000 calls tested… repeated 20 times to get the most accurate results.

What did I do with these 20 results? I dropped 2 lowest and 2 highest ones. Then I got an average from 16 remaining measurements (avg from 16 000 000 calls).

Information gathered:
- time in nanoseconds to access a public primitive field,
- time in nanoseconds to access a public reference,
- time in nanoseconds to call a public method.

Testing was made on 64-bit HotSpot VM, Java 1.8.0_66. Running on Mac OS X, i7-4770HQ processor.

Primitive field		Reference		Method	
Normal	Reflection	Normal	Reflection	Normal	Reflection
0,000003 ns	51,53 ns	0,85 ns	53,46 ns	3,24 ns	142,59 ns

Java vs Scala reflection benchmark

I have decided to do one more benchmark. Principles were the same as above.
Below you can see the comparison between the various types of reflection on a public primitive field. Java reflection vs 2 Scala reflection samples:
- "Java-in-Scala" - Java reflection API used in Scala code, the same methods and classes as in Java,
- Scala experimental reflection module - plain Scala "heavy-duty" reflection API with a bunch of useful out-of-the-box methods used to, for example, accessing the private fields, constructors, modifying the final values etc.

Java	„Java-in-Scala”	Scala experimental
51,53 ns	52,11 ns	4809,71 ns