# Tracing the Meta-Level: PyPy's Tracing JIT Compiler

Carl Friedrich Bolz
Heinrich-Heine-Universität Düsseldorf
Institut für Informatik
Softwaretechnik und Programmiersprachen

We attempt to apply the technique of Tracing JIT Compilers [3, 2] in the context of the PyPy project[1][4, 1], i.e. to programs that are interpreters for some dynamic languages, including Python. Tracing JIT compilers can greatly speed up programs that spend most of their time in loops in which they take similar code paths. However, applying an unmodified tracing JIT to a program that is itself a bytecode interpreter results in very limited or no speedup.

In this talk we show how to guide tracing JIT compilers to greatly improve the speed of bytecode interpreters. One crucial point is to unroll the bytecode dispatch loop, based on two hints provided by the implementer of the bytecode interpreter. The technique is already mature enough to be applied to a number of example interpreters, but also to PyPy's full Python interpreter, giving interesting speedups.

## References

[1] C. F. Bolz and A. Rigo. How to *not* write a virtual machine. In *Proceedings of the 3rd Workshop on Dynamic Languages and Applications (DYLA 2007)*, 2007.

[2] A. Gal and M. Franz. Incremental dynamic code generation with trace trees. Technical Report ICS-TR-06-16, Donald Bren School of Information and Computer Science, University of California, Irvine, Nov. 2006.

[3] A. Gal, C. W. Probst, and M. Franz. HotpathVM: an effective JIT compiler for resource-constrained devices. In *Proceedings of the 2nd International Conference on Virtual Execution Environments*, pages 144–153, Ottawa, Ontario, Canada, 2006. ACM.

[4] A. Rigo and S. Pedroni. PyPy's approach to virtual machine construction. In *Companion to the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 944–953, Portland, Oregon, USA, 2006. ACM.

[1]`http://codespeak.net`