# What This Short Cut Covers

This short cut focuses on a number of coding patterns that are useful when trying to get maximum speed out of performance-critical sections of Ruby code. Anti-patterns, that is, coding idioms, which should be avoided in performance-sensitive code sections are discussed, including details on how to transform such code to make it more efficient. Most patterns were extracted from Stefan Kaes' work on improving performance of the Rails core and his regular Rails performance consulting work. These patterns are largely non-algorithmic, detailing local code transformations to achieve identical results with slightly different and faster code, as even local code changes can sometimes result in orders of magnitude improvements. Some patterns are useful independent of Rails' implementation language, but some of them are specific to Ruby, or more specifically, the current implementation of Ruby. Converts from other languages, especially from statically typed languages such as Java or C++ may find this material useful, as the performance characteristics of certain operations, like performing a function call or accessing object fields/attributes, are quite different from what you expect.

# Introduction

In this short cut, I list a number of coding patterns that are useful when trying to get maximum speed out of performance-critical sections of code. In addition, I list anti-patterns, that is, coding idioms which

should be avoided in performance-sensitive code sections, and show how to transform such code to make it more efficient.

I extracted most of the patterns from my work on improving the performance of the Rails core and my regular Rails performance consulting work, but I can hardly claim that they are original in the sense that no one else has written about similar topics before. The patterns are largely nonalgorithmic, detailing local code transformations to achieve identical results with slightly different (and faster) code. However, even local code changes can sometimes result in orders of magnitude improvements (memorizing function calls is a good example of this phenomenon).

Some patterns are useful independent of Rails' implementation language, but some of them are specific to Ruby, or more specifically, the current implementation of Ruby. Converts from other languages, especially from statically typed languages such as Java and C++, may find this material useful, as the performance characteristics of certain operations, such as performing a function call or accessing object fields/attributes, are quite different from what you expect.

Finally, words of caution are in order: Some of the patterns might make your code look less elegant and/or make it harder to change later in the development process. In many cases, there's a tradeoff between performance gains and code readability and understandability. In such cases, one always needs to verify that optimizing a certain piece of code is really necessary and accompany the decision process with benchmarking.

My observations regarding relative efficiency of different coding patterns are based on experience with Ruby version 1.8.5 (and 1.8.6). In some cases, moving to Ruby 1.9 (the current development branch, including YARV, the new Ruby VM), might make a significant difference.