Extracted from:

Programming Erlang, Second Edition

Software for a Concurrent World

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Programming Erlang

Software for a Concurrent World

Second Edition

Joe Armstrong

Edited by Susannah Davidson Pfalzer

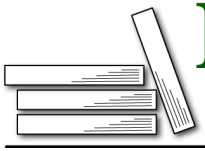
Programming Erlang, Second Edition

Software for a Concurrent World

Joe Armstrong

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Susannah Davidson Pfalzer (editor)
Potomac Indexing, LLC (indexer)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-53-6
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2013

Introduction

New hardware is increasingly parallel, so new programming languages must support concurrency or they will die.

> "The way the processor industry is going is to add more and more cores, but nobody knows how to program those things. I mean, two, yeah; four, not really; eight, forget it." —*Steve Jobs, Apple* [1]

Well, Steve was wrong; we do know how to program multicores. We program them in Erlang, and many of our programs just go faster as we add more cores.

Erlang was designed from the bottom up to program concurrent, distributed, fault-tolerant, scalable, soft, real-time systems. Soft real-time systems are systems such as telephone exchanges, banking systems, and so on, where rapid response times are important but it's not a disaster if the odd timing deadline is missed. Erlang systems have been deployed on a massive scale and control significant parts of the world's mobile communication networks.

If your problem is concurrent, if you are building a multiuser system, or if you are building a system that evolves with time, then using Erlang might save you a lot of work, since Erlang was explicitly designed for building such systems.

> "It's the mutable state, stupid." —*Brian Goetz, Java Concurrency in Practice*

Erlang belongs to the family of *functional programming languages*. Functional programming forbids code with side effects. Side effects and concurrency don't mix. In Erlang it's OK to mutate state within an individual process but not for one process to tinker with the state of another process. Erlang has no mutexes, no synchronized methods, and none of the paraphernalia of shared memory programming.

1. http://bits.blogs.nytimes.com/2008/06/10/apple-in-parallel-turning-the-pc-world-upside-down/

Processes interact by one method, and one method only, by exchanging messages. Processes share no data with other processes. This is the reason why we can easily distribute Erlang programs over multicores or networks.

When we write an Erlang program, we do not implement it as a single process that does everything; we implement it as large numbers of small processes that do simple things and communicate with each other.

What's This Book About?

It's about concurrency. It's about distribution. It's about fault tolerance. It's about functional programming. It's about programming a distributed concurrent system without locks and mutexes but using only pure message passing. It's about automatically speeding up your programs on multicore CPUs. It's about writing distributed applications that allow people to interact with each other. It's about design patterns for writing fault-tolerant and distributed systems. It's about modeling concurrency and mapping those models onto computer programs, a process I call *concurrency-oriented programming*.

Who Is This Book For?

The target audience for this book ranges from the experienced Erlang programmer who wants to learn more about Erlang internals and the philosophy behind Erlang to the absolute beginner. The text has been reviewed by programmers at all levels, from expert to beginner. One of the major differences between the second and first editions has been the addition of a large amount of explanatory material especially targeted at the beginner. Advanced Erlang programmers can skip over the introductory material.

A second goal has been to demystify functional, concurrent, and distributed programming and present it in a way that is appropriate to an audience that has no prior knowledge of concurrency or functional programming. Writing functional programs and parallel programs has long been regarded as a "black art"; this book is part of an ongoing attempt to change this.

While this book assumes no specific knowledge of either functional or concurrent programming, it is addressed to somebody who already is familiar with one or two programming languages.

When you come to a new programming language, it's often difficult to think of "problems that are suitable for solution in the new language." The exercises give you a clue. These are the kind of problems that are suitably solved in Erlang.

New in This Edition

First, the text has been brought up-to-date to reflect all the changes made to Erlang since the first edition of the book was published. We now cover all official language changes and describe Erlang version R17.

The second edition has been refocused to address the needs of beginners, with more explanatory text than in the first edition. Material intended for advanced users, or that might change rapidly, has been moved to online repositories.

The programming exercises proved so popular in the first edition that exercises now appear at the end of each chapter. The exercises vary in complexity, so there's something for both beginner users and advanced users.

In several completely new chapters, you'll learn about the Erlang type system and the Dialyzer, maps (which are new to Erlang, as of R17), websockets, programming idioms, and integrating third-party code. A new appendix describes how to build a minimal stand-alone Erlang system.

The final chapter, "Sherlock's Last Case," is a new chapter that gives you an exercise in processing and extracting meaning from a large volume of text. This is an open-ended chapter, and I hope that the exercises at the end of this chapter will stimulate future work.

Road Map

You can't run until you can walk. Erlang programs are made up from lots of small sequential programs running at the same time. Before we can write concurrent code, we need to be able to write sequential code. This means we won't get into the details of writing concurrent programs until Chapter 11, *Real-World Concurrency*, on page ?.

- Part I has a short introduction to the central ideas of concurrent programming and a whirlwind tour of Erlang.
- Part II covers sequential Erlang programming in detail and also talks about types and methods for building Erlang programs.
- Part III is the core of the book where we learn about how to write concurrent and distributed Erlang programs.
- Part IV covers the major Erlang libraries, techniques for tracing and debugging, and techniques for structuring Erlang code.
- Part V covers applications. You'll learn how to integrate external software with the core Erlang libraries and how to turn your own code into open

source contributions. We'll talk about programming idioms and how to program multicore CPUs. And finally, Sherlock Holmes will analyze our thoughts.

At the end of each chapter, you'll find a selection of programming exercises. These are to test your knowledge of the chapter and to challenge you. The problems vary from easy to difficult. The most difficult problems would be suitable research projects. Even if you don't try to solve all the problems, just thinking about the problems and how you would solve them will enhance your understanding of the text.

The Code in This Book

Most of the code snippets come from full-length, running examples that you can download.[2] To help you find your way, if a code listing in this book can be found in the download, there'll be a bar above the snippet (just like the one here):

shop1.erl

```
-module(shop1).
-export([total/1]).

total([{What, N}|T]) -> shop:cost(What) * N + total(T);
total([])            -> 0.
```

This bar contains the path to the code within the download. If you're reading the ebook version of this book and your ebook reader supports hyperlinks, you can click the bar, and the code should appear in a browser window.

Help! It Doesn't Work

Learning new stuff is difficult. You will get stuck. When you get stuck, rule 1 is to not silently give up. Rule 2 is to get help. Rule 3 is to ask Sherlock.

Rule 1 is important. There are people who have tried Erlang, gotten stuck and given up, and not told anybody. If we don't know about a problem, we can't fix it. End of story.

The best way to get help is to first try Google; if Google can't help, send mail to the Erlang mailing list.[3] You can also try #erlounge or #erlang at irc.freenode.net for a faster response.

Sometimes the answer to your question might be in an old posting to the Erlang mailing list but you just can't find it. In Chapter 27, *Sherlock's Last*

2. http://www.pragprog.com/titles/jaerlang2/source_code
3. erlang-questions@erlang.org

Case, on page ?, there's a program you can run locally that can perform sophisticated searches on all the old postings to the Erlang mailing list.

So, without further ado, I'll thank the good folks who helped me write this book (and the first edition), and you can skip on to Chapter 1, where we'll take a lightning tour of Erlang.

Acknowledgments

First Edition

Many people helped in the preparation of this book, and I'd like to thank them all here.

First, Dave Thomas, my editor: Dave taught me to write and subjected me to a barrage of never-ending questions. Why this? Why that? When I started the book, Dave said my writing style was like "standing on a rock preaching." He said, "I want you to talk to people, not preach." The book is better for it. Thanks, Dave.

Next, I had a little committee of language experts at my back. They helped me decide what to leave out. They also helped me clarify some of the bits that are difficult to explain. Thanks here (in no particular order) to Björn Gustavsson, Robert Virding, Kostis Sagonas, Kenneth Lundin, Richard Carlsson, and Ulf Wiger.

Thanks also to Claes Vikström who provided valuable advice on Mnesia, to Rickard Green who gave information on SMP Erlang, and to Hans Nilsson for the stemming algorithm used in the text-indexing program.

Sean Hinde and Ulf Wiger helped me understand how to use various OTP internals, and Serge Aleynikov explained active sockets to me so that I could understand.

Helen Taylor (my wife) proofread several chapters and provided hundreds of cups of tea at appropriate moments. What's more, she put up with my rather obsessive behavior for seven months. Thanks also to Thomas and Claire; and thanks to Bach and Handel, my cats Zorro and Daisy, and my Sat Nav Doris, who helped me stay sane, purred when stroked, and got me to the right addresses.

Finally, to all the readers of the beta book who filled in errata requests: I have cursed you and praised you. When the first beta went out, I was unprepared for the entire book to be read in two days and for you to shred every page with your comments. But the process has resulted in a much better book

than I had imagined. When (as happened several times) dozens of people said, "I don't understand this page," then I was forced to think again and rewrite the material concerned. Thanks for your help, everybody.

Second Edition

First, my new editor, Susannah Pfalzer, helped a lot in suggesting new ways to reorganize and refocus the book. It was great working with you; you've taught me a lot.

Kenneth Lundin and the guys in the OTP group worked hard to deliver the new language features described in the second edition.

Many readers of the first edition provided feedback on things they didn't understand, so I hope these are now rectified.

The design of maps is inspired by the work of Richard A. O'Keefe (who called them frames). Richard has championed the cause of frames on the Erlang mailing list for many years. Thanks, Richard, for all your comments and suggestions.

Kostis Sagonas provided lots of helpful feedback on the treatment of the type system.

I'd also like to thank Loïc Hoguin for his permission to use some examples from the cowboy web server from Nine Nines and the guys from Basho who wrote the code for BitLocker. I'd also like to thank Dave Smith for his work with rebar.

A number of people helped me by reviewing various drafts of the second edition. I'd like to thank all of them; they made this a better book. So, thanks to Erik Abefelt, Paul Butcher, Mark Chu-Carroll, Ian Dees, Henning Diedrich, Jeremy Frens, Loïc Hoguin, Andy Hunt, Kurt Landrus, Kenneth Lundin, Evan Miller, Patrik Nyblom, Tim Ottinger, Kim Shrier, and Bruce Tate for your help.

Helen Taylor (Twitter @mrsjoeerl) made countless cups of tea and cheered me up when I thought the book would never get finished.

Gustav Mahler, Sergei Rachmaninoff, Richard Wagner, and George Frideric Handel composed music (and Bob Dylan and few other guys…) that I played in the background while writing much of this book.