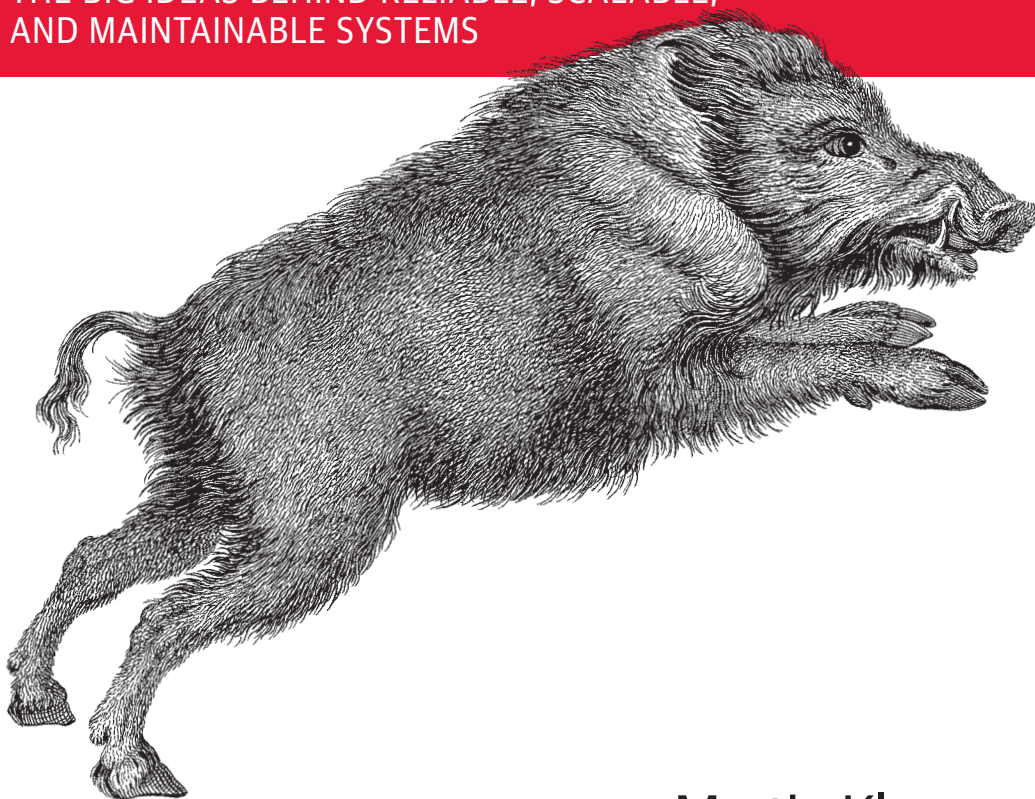


O'REILLY®

# Designing Data-Intensive Applications

---

THE BIG IDEAS BEHIND RELIABLE, SCALABLE,  
AND MAINTAINABLE SYSTEMS



Martin Kleppmann

# Designing Data-Intensive Applications

Data is at the center of many challenges in system design today. Difficult issues need to be figured out, such as scalability, consistency, reliability, efficiency, and maintainability. In addition, we have an overwhelming variety of tools, including relational databases, NoSQL datastores, stream or batch processors, and message brokers. What are the right choices for your application? How do you make sense of all these buzzwords?

In this practical and comprehensive guide, author Martin Kleppmann helps you navigate this diverse landscape by examining the pros and cons of various technologies for processing and storing data. Software keeps changing, but the fundamental principles remain the same. With this book, software engineers and architects will learn how to apply those ideas in practice, and how to make full use of data in modern applications.

- Peer under the hood of the systems you already use, and learn how to use and operate them more effectively
- Make informed decisions by identifying the strengths and weaknesses of different tools
- Navigate the trade-offs around consistency, scalability, fault tolerance, and complexity
- Understand the distributed systems research upon which modern databases are built
- Peek behind the scenes of major online services, and learn from their architectures

**Martin Kleppmann** is a researcher in distributed systems at the University of Cambridge, UK. Previously he was a software engineer and entrepreneur at internet companies including LinkedIn and Rapportive, where he worked on large-scale data infrastructure. Martin is a regular conference speaker, blogger, and open source contributor.

“This book is awesome. It bridges the huge gap between distributed systems theory and practical engineering. I wish it had existed a decade ago, so I could have read it then and saved myself all the mistakes along the way.”

—Jay Kreps

Creator of Apache Kafka  
and CEO of Confluent

“This book should be required reading for software engineers. *Designing Data-Intensive Applications* is a rare resource that connects theory and practice to help developers make smart decisions as they design and implement data infrastructure and systems.”

—Kevin Scott

Chief Technology Officer at Microsoft

DATA | HADOOP

US \$44.99

CAN \$59.99

ISBN: 978-1-449-37332-0



Twitter: @oreillymedia  
facebook.com/oreilly

---

# Designing Data-Intensive Applications

*The Big Ideas Behind Reliable, Scalable,  
and Maintainable Systems*

*Martin Kleppmann*

Beijing • Boston • Farnham • Sebastopol • Tokyo

**O'REILLY®**

## Designing Data-Intensive Applications

by Martin Kleppmann

Copyright © 2017 Martin Kleppmann. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://oreilly.com/safari>). For more information, contact our corporate/institutional sales department: 800-998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

**Editors:** Ann Spencer and Marie Beaugureau

**Production Editor:** Kristen Brown

**Copyeditor:** Rachel Head

**Proofreader:** Amanda Kersey

**Indexer:** Ellen Troutman-Zaig

**Interior Designer:** David Futato

**Cover Designer:** Karen Montgomery

**Illustrator:** Rebecca Demarest

March 2017:            First Edition

### Revision History for the First Edition

2017-03-01:    First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781449373320> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Designing Data-Intensive Applications*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-449-37332-0

[LSI]

*Technology is a powerful force in our society. Data, software, and communication can be used for bad: to entrench unfair power structures, to undermine human rights, and to protect vested interests. But they can also be used for good: to make underrepresented people's voices heard, to create opportunities for everyone, and to avert disasters. This book is dedicated to everyone working toward the good.*



*Computing is pop culture. [...] Pop culture holds a disdain for history. Pop culture is all about identity and feeling like you're participating. It has nothing to do with cooperation, the past or the future—it's living in the present. I think the same is true of most people who write code for money. They have no idea where [their culture came from].*

—Alan Kay, in interview with *Dr Dobb's Journal* (2012)





---

# Table of Contents

Preface.....	xiii
--------------	------

---

## Part I. Foundations of Data Systems

<b>1. Reliable, Scalable, and Maintainable Applications.....</b>	<b>3</b>
Thinking About Data Systems	4
Reliability	6
Hardware Faults	7
Software Errors	8
Human Errors	9
How Important Is Reliability?	10
Scalability	10
Describing Load	11
Describing Performance	13
Approaches for Coping with Load	17
Maintainability	18
Operability: Making Life Easy for Operations	19
Simplicity: Managing Complexity	20
Evolvability: Making Change Easy	21
Summary	22
<b>2. Data Models and Query Languages.....</b>	<b>27</b>
Relational Model Versus Document Model	28
The Birth of NoSQL	29
The Object-Relational Mismatch	29
Many-to-One and Many-to-Many Relationships	33
Are Document Databases Repeating History?	36

Relational Versus Document Databases Today	38
Query Languages for Data	42
Declarative Queries on the Web	44
MapReduce Querying	46
Graph-Like Data Models	49
Property Graphs	50
The Cypher Query Language	52
Graph Queries in SQL	53
Triple-Stores and SPARQL	55
The Foundation: Datalog	60
Summary	63
<b>3. Storage and Retrieval. ....</b>	<b>69</b>
Data Structures That Power Your Database	70
Hash Indexes	72
SSTables and LSM-Trees	76
B-Trees	79
Comparing B-Trees and LSM-Trees	83
Other Indexing Structures	85
Transaction Processing or Analytics?	90
Data Warehousing	91
Stars and Snowflakes: Schemas for Analytics	93
Column-Oriented Storage	95
Column Compression	97
Sort Order in Column Storage	99
Writing to Column-Oriented Storage	101
Aggregation: Data Cubes and Materialized Views	101
Summary	103
<b>4. Encoding and Evolution. ....</b>	<b>111</b>
Formats for Encoding Data	112
Language-Specific Formats	113
JSON, XML, and Binary Variants	114
Thrift and Protocol Buffers	117
Avro	122
The Merits of Schemas	127
Modes of Dataflow	128
Dataflow Through Databases	129
Dataflow Through Services: REST and RPC	131
Message-Passing Dataflow	136
Summary	139

---

## Part II. Distributed Data

<b>5. Replication.....</b>	<b>151</b>
Leaders and Followers	152
Synchronous Versus Asynchronous Replication	153
Setting Up New Followers	155
Handling Node Outages	156
Implementation of Replication Logs	158
Problems with Replication Lag	161
Reading Your Own Writes	162
Monotonic Reads	164
Consistent Prefix Reads	165
Solutions for Replication Lag	167
Multi-Leader Replication	168
Use Cases for Multi-Leader Replication	168
Handling Write Conflicts	171
Multi-Leader Replication Topologies	175
Leaderless Replication	177
Writing to the Database When a Node Is Down	177
Limitations of Quorum Consistency	181
Sloppy Quorums and Hinted Handoff	183
Detecting Concurrent Writes	184
Summary	192
<b>6. Partitioning.....</b>	<b>199</b>
Partitioning and Replication	200
Partitioning of Key-Value Data	201
Partitioning by Key Range	202
Partitioning by Hash of Key	203
Skewed Workloads and Relieving Hot Spots	205
Partitioning and Secondary Indexes	206
Partitioning Secondary Indexes by Document	206
Partitioning Secondary Indexes by Term	208
Rebalancing Partitions	209
Strategies for Rebalancing	210
Operations: Automatic or Manual Rebalancing	213
Request Routing	214
Parallel Query Execution	216
Summary	216
<b>7. Transactions.....</b>	<b>221</b>
The Slippery Concept of a Transaction	222

The Meaning of ACID	223
Single-Object and Multi-Object Operations	228
Weak Isolation Levels	233
Read Committed	234
Snapshot Isolation and Repeatable Read	237
Preventing Lost Updates	242
Write Skew and Phantoms	246
Serializability	251
Actual Serial Execution	252
Two-Phase Locking (2PL)	257
Serializable Snapshot Isolation (SSI)	261
Summary	266
<b>8. The Trouble with Distributed Systems. ....</b>	<b>273</b>
Faults and Partial Failures	274
Cloud Computing and Supercomputing	275
Unreliable Networks	277
Network Faults in Practice	279
Detecting Faults	280
Timeouts and Unbounded Delays	281
Synchronous Versus Asynchronous Networks	284
Unreliable Clocks	287
Monotonic Versus Time-of-Day Clocks	288
Clock Synchronization and Accuracy	289
Relying on Synchronized Clocks	291
Process Pauses	295
Knowledge, Truth, and Lies	300
The Truth Is Defined by the Majority	300
Byzantine Faults	304
System Model and Reality	306
Summary	310
<b>9. Consistency and Consensus. ....</b>	<b>321</b>
Consistency Guarantees	322
Linearizability	324
What Makes a System Linearizable?	325
Relying on Linearizability	330
Implementing Linearizable Systems	332
The Cost of Linearizability	335
Ordering Guarantees	339
Ordering and Causality	339
Sequence Number Ordering	343

Total Order Broadcast	348
Distributed Transactions and Consensus	352
Atomic Commit and Two-Phase Commit (2PC)	354
Distributed Transactions in Practice	360
Fault-Tolerant Consensus	364
Membership and Coordination Services	370
Summary	373

---

## Part III. Derived Data

<b>10. Batch Processing.....</b>	<b>389</b>
Batch Processing with Unix Tools	391
Simple Log Analysis	391
The Unix Philosophy	394
MapReduce and Distributed Filesystems	397
MapReduce Job Execution	399
Reduce-Side Joins and Grouping	403
Map-Side Joins	408
The Output of Batch Workflows	411
Comparing Hadoop to Distributed Databases	414
Beyond MapReduce	419
Materialization of Intermediate State	419
Graphs and Iterative Processing	424
High-Level APIs and Languages	426
Summary	429
<b>11. Stream Processing.....</b>	<b>439</b>
Transmitting Event Streams	440
Messaging Systems	441
Partitioned Logs	446
Databases and Streams	451
Keeping Systems in Sync	452
Change Data Capture	454
Event Sourcing	457
State, Streams, and Immutability	459
Processing Streams	464
Uses of Stream Processing	465
Reasoning About Time	468
Stream Joins	472
Fault Tolerance	476
Summary	479

<b>12. The Future of Data Systems.....</b>	<b>489</b>
Data Integration	490
Combining Specialized Tools by Deriving Data	490
Batch and Stream Processing	494
Unbundling Databases	499
Composing Data Storage Technologies	499
Designing Applications Around Dataflow	504
Observing Derived State	509
Aiming for Correctness	515
The End-to-End Argument for Databases	516
Enforcing Constraints	521
Timeliness and Integrity	524
Trust, but Verify	528
Doing the Right Thing	533
Predictive Analytics	533
Privacy and Tracking	536
Summary	543
<b>Glossary.....</b>	<b>553</b>
<b>Index.....</b>	<b>559</b>

---

# Preface

If you have worked in software engineering in recent years, especially in server-side and backend systems, you have probably been bombarded with a plethora of buzzwords relating to storage and processing of data. NoSQL! Big Data! Web-scale! Sharding! Eventual consistency! ACID! CAP theorem! Cloud services! MapReduce! Real-time!

In the last decade we have seen many interesting developments in databases, in distributed systems, and in the ways we build applications on top of them. There are various driving forces for these developments:

- Internet companies such as Google, Yahoo!, Amazon, Facebook, LinkedIn, Microsoft, and Twitter are handling huge volumes of data and traffic, forcing them to create new tools that enable them to efficiently handle such scale.
- Businesses need to be agile, test hypotheses cheaply, and respond quickly to new market insights by keeping development cycles short and data models flexible.
- Free and open source software has become very successful and is now preferred to commercial or bespoke in-house software in many environments.
- CPU clock speeds are barely increasing, but multi-core processors are standard, and networks are getting faster. This means parallelism is only going to increase.
- Even if you work on a small team, you can now build systems that are distributed across many machines and even multiple geographic regions, thanks to infrastructure as a service (IaaS) such as Amazon Web Services.
- Many services are now expected to be highly available; extended downtime due to outages or maintenance is becoming increasingly unacceptable.

*Data-intensive applications* are pushing the boundaries of what is possible by making use of these technological developments. We call an application *data-intensive* if data is its primary challenge—the quantity of data, the complexity of data, or the speed at

which it is changing—as opposed to *compute-intensive*, where CPU cycles are the bottleneck.

The tools and technologies that help data-intensive applications store and process data have been rapidly adapting to these changes. New types of database systems (“NoSQL”) have been getting lots of attention, but message queues, caches, search indexes, frameworks for batch and stream processing, and related technologies are very important too. Many applications use some combination of these.

The buzzwords that fill this space are a sign of enthusiasm for the new possibilities, which is a great thing. However, as software engineers and architects, we also need to have a technically accurate and precise understanding of the various technologies and their trade-offs if we want to build good applications. For that understanding, we have to dig deeper than buzzwords.

Fortunately, behind the rapid changes in technology, there are enduring principles that remain true, no matter which version of a particular tool you are using. If you understand those principles, you’re in a position to see where each tool fits in, how to make good use of it, and how to avoid its pitfalls. That’s where this book comes in.

The goal of this book is to help you navigate the diverse and fast-changing landscape of technologies for processing and storing data. This book is not a tutorial for one particular tool, nor is it a textbook full of dry theory. Instead, we will look at examples of successful data systems: technologies that form the foundation of many popular applications and that have to meet scalability, performance, and reliability requirements in production every day.

We will dig into the internals of those systems, tease apart their key algorithms, discuss their principles and the trade-offs they have to make. On this journey, we will try to find useful ways of *thinking about* data systems—not just *how* they work, but also *why* they work that way, and what questions we need to ask.

After reading this book, you will be in a great position to decide which kind of technology is appropriate for which purpose, and understand how tools can be combined to form the foundation of a good application architecture. You won’t be ready to build your own database storage engine from scratch, but fortunately that is rarely necessary. You will, however, develop a good intuition for what your systems are doing under the hood so that you can reason about their behavior, make good design decisions, and track down any problems that may arise.

## Who Should Read This Book?

If you develop applications that have some kind of server/backend for storing or processing data, and your applications use the internet (e.g., web applications, mobile apps, or internet-connected sensors), then this book is for you.



This book is for software engineers, software architects, and technical managers who love to code. It is especially relevant if you need to make decisions about the architecture of the systems you work on—for example, if you need to choose tools for solving a given problem and figure out how best to apply them. But even if you have no choice over your tools, this book will help you better understand their strengths and weaknesses.

You should have some experience building web-based applications or network services, and you should be familiar with relational databases and SQL. Any non-relational databases and other data-related tools you know are a bonus, but not required. A general understanding of common network protocols like TCP and HTTP is helpful. Your choice of programming language or framework makes no difference for this book.

If any of the following are true for you, you'll find this book valuable:

- You want to learn how to make data systems scalable, for example, to support web or mobile apps with millions of users.
- You need to make applications highly available (minimizing downtime) and operationally robust.
- You are looking for ways of making systems easier to maintain in the long run, even as they grow and as requirements and technologies change.
- You have a natural curiosity for the way things work and want to know what goes on inside major websites and online services. This book breaks down the internals of various databases and data processing systems, and it's great fun to explore the bright thinking that went into their design.

Sometimes, when discussing scalable data systems, people make comments along the lines of, “You’re not Google or Amazon. Stop worrying about scale and just use a relational database.” There is truth in that statement: building for scale that you don’t need is wasted effort and may lock you into an inflexible design. In effect, it is a form of premature optimization. However, it’s also important to choose the right tool for the job, and different technologies each have their own strengths and weaknesses. As we shall see, relational databases are important but not the final word on dealing with data.

## Scope of This Book

This book does not attempt to give detailed instructions on how to install or use specific software packages or APIs, since there is already plenty of documentation for those things. Instead we discuss the various principles and trade-offs that are fundamental to data systems, and we explore the different design decisions taken by different products.

In the ebook editions we have included links to the full text of online resources. All links were verified at the time of publication, but unfortunately links tend to break frequently due to the nature of the web. If you come across a broken link, or if you are reading a print copy of this book, you can look up references using a search engine. For academic papers, you can search for the title in Google Scholar to find open-access PDF files. Alternatively, you can find all of the references at <https://github.com/ept/ddia-references>, where we maintain up-to-date links.

We look primarily at the *architecture* of data systems and the ways they are integrated into data-intensive applications. This book doesn't have space to cover deployment, operations, security, management, and other areas—those are complex and important topics, and we wouldn't do them justice by making them superficial side notes in this book. They deserve books of their own.

Many of the technologies described in this book fall within the realm of the *Big Data* buzzword. However, the term “Big Data” is so overused and underdefined that it is not useful in a serious engineering discussion. This book uses less ambiguous terms, such as single-node versus distributed systems, or online/interactive versus offline/batch processing systems.

This book has a bias toward free and open source software (FOSS), because reading, modifying, and executing source code is a great way to understand how something works in detail. Open platforms also reduce the risk of vendor lock-in. However, where appropriate, we also discuss proprietary software (closed-source software, software as a service, or companies' in-house software that is only described in literature but not released publicly).

## Outline of This Book

This book is arranged into three parts:

1. In **Part I**, we discuss the fundamental ideas that underpin the design of data-intensive applications. We start in **Chapter 1** by discussing what we're actually trying to achieve: reliability, scalability, and maintainability; how we need to think about them; and how we can achieve them. In **Chapter 2** we compare several different data models and query languages, and see how they are appropriate to different situations. In **Chapter 3** we talk about storage engines: how databases arrange data on disk so that we can find it again efficiently. **Chapter 4** turns to formats for data encoding (serialization) and evolution of schemas over time.
2. In **Part II**, we move from data stored on one machine to data that is distributed across multiple machines. This is often necessary for scalability, but brings with it a variety of unique challenges. We first discuss replication (**Chapter 5**), partitioning/sharding (**Chapter 6**), and transactions (**Chapter 7**). We then go into


more detail on the problems with distributed systems ([Chapter 8](#)) and what it means to achieve consistency and consensus in a distributed system ([Chapter 9](#)).

3. In [Part III](#), we discuss systems that derive some datasets from other datasets. Derived data often occurs in heterogeneous systems: when there is no one database that can do everything well, applications need to integrate several different databases, caches, indexes, and so on. In [Chapter 10](#) we start with a batch processing approach to derived data, and we build upon it with stream processing in [Chapter 11](#). Finally, in [Chapter 12](#) we put everything together and discuss approaches for building reliable, scalable, and maintainable applications in the future.

## References and Further Reading

Most of what we discuss in this book has already been said elsewhere in some form or another—in conference presentations, research papers, blog posts, code, bug trackers, mailing lists, and engineering folklore. This book summarizes the most important ideas from many different sources, and it includes pointers to the original literature throughout the text. The references at the end of each chapter are a great resource if you want to explore an area in more depth, and most of them are freely available online.

## O'Reilly Safari

 **Safari**® *Safari* (formerly Safari Books Online) is a membership-based training and reference platform for enterprise, government, educators, and individuals.

Members have access to thousands of books, training videos, Learning Paths, interactive tutorials, and curated playlists from over 250 publishers, including O'Reilly Media, Harvard Business Review, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Adobe, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, and Course Technology, among others.

For more information, please visit <http://oreilly.com/safari>.

# How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (in the United States or Canada)  
707-829-0515 (international or local)  
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://bit.ly/designing-data-intensive-apps>.

To comment or ask technical questions about this book, send email to [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

## Acknowledgments

This book is an amalgamation and systematization of a large number of other people's ideas and knowledge, combining experience from both academic research and industrial practice. In computing we tend to be attracted to things that are new and shiny, but I think we have a huge amount to learn from things that have been done before. This book has over 800 references to articles, blog posts, talks, documentation, and more, and they have been an invaluable learning resource for me. I am very grateful to the authors of this material for sharing their knowledge.

I have also learned a lot from personal conversations, thanks to a large number of people who have taken the time to discuss ideas or patiently explain things to me. In particular, I would like to thank Joe Adler, Ross Anderson, Peter Bailis, Márton Balassi, Alastair Beresford, Mark Callaghan, Mat Clayton, Patrick Collison, Sean Cribbs, Shirshanka Das, Niklas Ekström, Stephan Ewen, Alan Fekete, Gyula Fóra, Camille Fournier, Andres Freund, John Garbutt, Seth Gilbert, Tom Haggett, Pat Hella, Joe Hellerstein, Jakob Homan, Heidi Howard, John Hugg, Julian Hyde, Conrad Irwin, Evan Jones, Flavio Junqueira, Jessica Kerr, Kyle Kingsbury, Jay Kreps, Carl Lerche, Nicolas Liochon, Steve Loughran, Lee Mallabone, Nathan Marz, Caitie

McCaffrey, Josie McLellan, Christopher Meiklejohn, Ian Meyers, Neha Narkhede, Neha Narula, Cathy O’Neil, Onora O’Neill, Ludovic Orban, Zoran Perkov, Julia Powles, Chris Riccomini, Henry Robinson, David Rosenthal, Jennifer Rullmann, Matthew Sackman, Martin Scholl, Amit Sela, Gwen Shapira, Greg Spurrier, Sam Stokes, Ben Stopford, Tom Stuart, Diana Vasile, Rahul Vohra, Pete Warden, and Brett Wooldridge.

Several more people have been invaluable to the writing of this book by reviewing drafts and providing feedback. For these contributions I am particularly indebted to Raul Agepati, Tyler Akidau, Mattias Andersson, Sasha Baranov, Veena Basavaraj, David Beyer, Jim Brikman, Paul Carey, Raul Castro Fernandez, Joseph Chow, Derek Elkins, Sam Elliott, Alexander Gallego, Mark Grover, Stu Halloway, Heidi Howard, Nicola Kleppmann, Stefan Kruppa, Bjorn Madsen, Sander Mak, Stefan Podkowinski, Phil Potter, Hamid Ramazani, Sam Stokes, and Ben Summers. Of course, I take all responsibility for any remaining errors or unpalatable opinions in this book.

For helping this book become real, and for their patience with my slow writing and unusual requests, I am grateful to my editors Marie Beaugureau, Mike Loukides, Ann Spencer, and all the team at O’Reilly. For helping find the right words, I thank Rachel Head. For giving me the time and freedom to write in spite of other work commitments, I thank Alastair Beresford, Susan Goodhue, Neha Narkhede, and Kevin Scott.

Very special thanks are due to Shabbir Diwan and Edie Freedman, who illustrated with great care the maps that accompany the chapters. It’s wonderful that they took on the unconventional idea of creating maps, and made them so beautiful and compelling.

Finally, my love goes to my family and friends, without whom I would not have been able to get through this writing process that has taken almost four years. You’re the best.

