# Положительные отзывы к книге «Семь моделей конкуренции за семь недель»

Уже не одно десятилетие профессиональные программисты реализуют параллельное выполнение кода с применением потоков и блокировок. Но эта модель – лишь одна из многих, что ярко демонстрирует книга «Семь моделей конкуренции за семь недель». Если вы хотите добиться успеха в мире, где основные языки программирования постепенно обретают поддержку акторов, CSP[1], параллелизма данных, функционального программирования и заимствуют унифицированную модель последовательностей из языка Clojure, обязательно прочитайте эту книгу.

➤ **Стюарт Хэллоуэй (Stuart Halloway)**
Соосновательь, Cognitect

С появлением микропроцессоров, обладающих большим числом ядер, понимание конкуренции стало еще более важным, чем прежде. Вы познакомитесь с преимуществами функционального программирования с точки зрения конкуренции, узнаете, как применять акторы для разработки распределенного программного обеспечения, и исследуете приемы параллельной обработки огромных объемов информации на нескольких процессорах. Эта книга поможет вам приобрести новые навыки в разработке программ, благодаря чему вы будете готовы решать сложные задачи в ближайшие несколько лет.

➤ **Хосе Валим (José Valim)**
Соосновательь Plataformatec

В книге «Семь моделей конкуренции за семь недель», которая является поучительным обзором различных приемов параллельного/конкурентного выполнения, автору удалось найти баланс между описанием скучной теории и живыми экспериментами.

➤ **Фредерик Чанг (Frederick Cheung)**
Технический директор, Dressipi

Мир постоянно изменяется, и каждый действующий программист должен изучать приемы конкурентного программирования. Теперь, когда меня будут спрашивать: «Как сделать это?», – я смогу предложить

[1] Communicating sequential processes – взаимодействующие последовательные процессы. – *Прим. перев.*

прочитать эту книгу. Я многое узнал из нее и с радостью могу порекомендовать ее.

➤ **Эндрю Хейли (Andrew Haley)**
Ведущий программист на Java, Red Hat

По мере того, как закон Адмала[2] начинает преобладать над законом Мура[3], происходит переход от объектно-ориентированного к конкурентно-ориентированному программированию. Вследствие этого данная книга появилась как нельзя кстати. Пол проделал фантастическую работу, описав наиболее важные модели конкуренции и дав читателю обширный набор инструментов, из которого он сможет выбирать наиболее подходящие для его потребностей. Эту книгу должен прочитать каждый, кто разрабатывает программное обеспечение в нашу многоядерную эпоху.

➤ **Франческо Цезарини (Francesco Cesarini)**
Основатель и технический директор Erlang Solutions

В своей книге Пол дает превосходное введение в тернистую тему конкуренции и параллелизма, ясно и доходчиво описывая различные подходы.

➤ **Шон Эллис (Sean Ellis)**
Архитектор GPU, ARM

Просто о сложном. Я бы с удовольствием прослушал университетский курс по этой теме, взяв в качестве руководства «Семь моделей конкуренции за семь недель».

➤ **Карлос Сесса (Carlos Sessa)**
Разработчик для Android, Groupon

Пол Батчер (Paul Butcher) взял проблему, которая бросает в дрожь многих разработчиков, и наглядно продемонстрировал целую серию парадигм программирования, которые с успехом можно использовать для реализации конкуренции в программном обеспечении.

➤ **Пайди Крид (Páidí Creed)**
Программист, SwiftKey

Имея опыт сотрудничества с Полом, могу рекомендовать его как опытного эксперта в области архитектуры языков программирования. Эта книга является ясным описанием темы, часто недооцененной, но жизненно важной для разработки программного обеспечения.

➤ **Бен Медлок (Ben Medlock)**
Сооснователь и технический директор, SwiftKey

---

[2] https://ru.wikipedia.org/wiki/Закон_Адмала – *Прим. перев.*

[3] https://ru.wikipedia.org/wiki/Закон_Мура – *Прим. перев.*

# ОГЛАВЛЕНИЕ

# ПРЕДИСЛОВИЕ

Эта книга рассказывает историю.

Кому-то может показаться странным, что книга начинается с этих слов, но для меня они наполнены смыслом. Видите ли, при создании книги «Семь моделей конкуренции за семь недель» мы отклонили десятки предложений от разных авторов, считавших, что можно собрать в кучу семь не связанных между собой статей и назвать это книгой. Мы думаем иначе.

В первой нашей истории («Seven Languages in Seven Weeks: A Pragmatic Guide to Learning Programming Languages» [Tat10]) рассказывалось, что объектно-ориентированные языки были хороши для своего времени, но под давлением всевозрастающей сложности программного обеспечения и необходимости управления конкуренцией в многоядерных архитектурах стали появляться языки функционального программирования, формирующие иные подходы к программированию. Пол Батчер (Paul Butcher) оказался самым полезным из научных редакторов этой книги. После четырехлетнего знакомства я понял, почему.

Пол многое повидал, находясь в первых рядах тех, кто внедрял по-настоящему масштабируемую поддержку конкуренции в действующие приложения. В книге «Seven Languages» он подметил, что решение многих сложных задач все чаще и чаще переносится на уровень языка. Пару лет спустя Пол пришел к нам с предложением написать свою книгу. Он говорил, что языки занимают важное место в общей истории, но они – лишь верхушка айсберга. Он хотел поведать нашим читателям намного более полную историю и простым языком рассказать о наиболее важных инструментах, которые используются в современном программном обеспечении для решения проблем, связанных с параллельным выполнением кода.

Сначала мы скептически отнеслись к его предложению. Такие книги пишутся тяжело – для них требуется больше времени, а процент неудач слишком высок – Пол решил сразиться со слишком большим драконом. Но, став настоящей командой, умы смогли превратить

первоначальный список глав в хорошую историю. По мере того как росла стопка страниц, мы все отчетливее понимали, что Пол не только имел обширные технические знания для раскрытия данной темы, но и азарт, чтобы взяться за нее. Мы также осознали всю особенность этой книги и ее своевременность. По мере углубления в нее вы поймете, о чем я говорю.

Вы будете ежиться вместе с нами, читая главу о потоках выполнения и блокировках – наиболее широко используемых ныне механизмах реализации конкуренции. Вы увидите, в каких ситуациях это решение дает плохие результаты, и затем приметесь за работу. Пол познакомит вас с самыми разными подходами, от лямбда-архитектуры, используемой в некоторых высоконагруженных социальных платформах, до модели на основе акторов, лежащей в основе многих крупнейших в мире, высоконадежных телекоммуникационных систем. Вы увидите языки программирования, которыми пользуются профессионалы, от Java до Clojure и захватывающего языка Elixir, основанного на Erlang. И на каждом шаге этого сложного пути Пол будет поддерживать вас, делясь своими глубокими знаниями посвященного.

Я рад представить вам книгу «Семь моделей конкуренции за семь недель» и надеюсь, что для вас она окажется такой же ценной, как и для меня.

**Брюс Тейт (Bruce A. Tate)**

Технический директор icanmakeitbetter.com
Редактор серии «Семь за семь»
Остин, Техас

# ВСТУПЛЕНИЕ

В 1989-м я приступил к работе над диссертацией, в которой занялся исследованием реализации параллельных и распределенных вычислений на разных языках программирования, – я был убежден, что со временем конкурентное программирование займет господствующее положение. Спустя два десятилетия моя правота была доказана – вокруг ведется масса разговоров о многоядерных архитектурах и о том, как использовать их преимущества.

Но поддержка конкуренции позволяет получить не только высокую производительность многоядерных систем. При правильном использовании конкуренция оказывается ключом к высокой отзывчивости, надежности, эффективности и простоте программного обеспечения.

## О книге

Эта книга имеет ту же структуру, что и остальные книги серии «Семь за семь» издательства The Pragmatic Bookshelf: «Seven Languages in Seven Weeks» [Tat10], «Seven Databases in Seven Weeks» [RW12] и «Seven Web Frameworks in Seven Weeks» [MD14].[1] Для этой книги были отобраны семь подходов, чтобы на их примере продемонстрировать общее положение дел в области поддержки конкуренции. Мы охватим подходы, которые уже прочно заняли господствующее положение, которые вскоре могут занять такое положение, а также те, которые вряд ли получат достаточно широкое распространение, но являются фантастически мощными в своих нишах. Я надеюсь, что по прочтении этой книги вы будете точно знать, какой инструмент выбрать, столкнувшись с той или иной проблемой конкуренции.

Каждая глава спроектирована так, чтобы ее можно было прочитать за выходные, и разбита на три дня. Каждый день завершается упражнениями, выполнение которых поможет закрепить знания, получен-

[1] В издптельстве «ДМК Пресс» выпущено две книги этой же серии:
http://dmkpress.com/catalog/computer/databases/978-5-94074-866-3/
http://dmkpress.com/catalog/computer/programming/978-5-94074-539-6/

ные в этот день, а каждая глава завершается кратким перечислением сильных и слабых сторон подхода, изучавшегося в этой главе.

Несмотря на небольшие философские отступления, основное внимание в книге будет уделяться практическим примерам. Я настоятельно рекомендую поработать с этими примерами – нет ничего более убедительного, чем действующий код.

## Чем не является эта книга

Эта книга не является справочным руководством. Я буду использовать языки, с которыми вы, возможно, плохо знакомы, такие как Elixir и Clojure. Но, поскольку главной темой этой книги является конкуренция, а не языки, я не буду подробно описывать те или иные аспекты языков, используемые в примерах. Надеюсь, что все и так будет понятно из контекста, но также надеюсь, что вы самостоятельно сможете заняться исследованием того или иного языка, если это потребуется. Возможно, удобнее будет читать эту книгу с веб-браузером под рукой, чтобы можно было оперативно обратиться к описанию языка в Интернете.

Эта книга не содержит инструкций по установке. Чтобы опробовать примеры кода, вам придется установить и настроить различные инструменты – какие-то подсказки вы найдете в файлах `README`, сопровождающих примеры, но в общем и целом вам придется проявить свою самостоятельность. Я использовал широко распространенные инструменты для всех примеров, поэтому вы не будете испытывать недостатка в руководствах по установке, которые легко можно найти в Интернете.

Наконец, эта книга не является исчерпывающим источником информации – в книге не так много места, чтобы охватить каждую тему во всех подробностях. Какие-то аспекты я буду упоминать лишь мимоходом, а какие-то просто пропущу. По возможности я преднамеренно использовал неидиоматический код, чтобы тем, кто незнаком с используемыми языками программирования, было проще понять, как он действует. Если вы решите заняться более глубокими исследованиями какой-либо из представленных здесь технологий, обращайтесь к специализированным книгам, упоминаемым в тексте.

## Примеры кода

Все примеры кода, демонстрируемые в книге, можно загрузить с сайта книги[2]. Каждый пример включает не только исходный код, но и

[2] http://pragprog.com/book/pb7con/

систему сборки. Для каждого языка я выбрал наиболее популярную систему сборки (Maven – для Java, Leiningen – для Clojure, Mix – для Elixir, sbt – для Scal и GNU Make – для C).

В большинстве случаев эти системы не только создадут из примера выполняемую программу, но и автоматически загрузят дополнительные зависимости. А системы sbt и Leiningen загрузят даже соответствующие версии компиляторов Scala и Clojure, поэтому вам остается только установить соответствующий инструмент сборки, а описание процесса установки вы без труда найдете в Интернете.

Главным исключением является код на языке C, демонстрирующийся в главе 7 «Параллелизм данных», для опробования которого вам придется установить инструмент OpenCL для вашей операционной системы и графической карты (если только вы не пользуетесь Mac OS, среда разработки Xcode в которой уже включает все необходимое).

## Примечание для пользователей IDE

Все системы сборки были проверены из командной строки. Если вы пользуетесь исключительно интегрированной средой разработки (Integrated Development Environment, IDE), вам придется импортировать систему сборки в свою IDE – большинство IDE уже поддерживают систему сборки Maven и имеют расширения для поддержки sbt и Leiningen. Но я не проверял этого, поэтому многим из вас проще будет научиться пользоваться командной строкой.

## Примечание для пользователей Windows

Все примеры были протестированы в OS X и Linux. Они также должны работать и в Windows, но я не проверял их в этой операционной системе.

Исключением является код на языке C, демонстрирующийся в главе 7 «Параллелизм данных», для компиляции которого используется GNU Make и GCC. Должно быть, несложно перенести этот код на Visual C++, но, опять же, я не проверял этого.

## Ресурсы в Сети

Приложения и примеры, демонстрирующиеся в этой книге, можно найти на веб-сайте Pragmatic Programmers.[3] Там же вы найдете форум читателей и форму для отправки сообщений, с помощью которой вы

[3] http://pragprog.com/book/pb7con

сможете сообщить нам о найденных ошибках и опечатках или внести предложения к будущим изданиям книги.

**Пол Бутчер (Paul Butcher)**
Ten Tenths Consulting
paul@tententhsconsulting.com
Кембридж, Великобритания, июнь 2014

# ГЛАВА 1. Введение

Конкурентное программирование не является чем-то новым, но оно лишь недавно превратилось в горячую тему. Языки программирования, такие как Erlang, Haskell, Go, Scala и Clojure, получили известность отчасти благодаря превосходной поддержке конкуренции.

Основной причиной всплеска интереса стал так называемый «кризис многоядерных систем». В соответствии с законом Мура число транзисторов на одном кристалле постоянно увеличивается[1], но это приводит не к повышению быстродействия процессора, а к увеличению числа ядер.

Как заметил Герб Саттер (Herb Sutter): «Бесплатного супа больше не будет».[2] Вы больше не сможете получить более высокую скорость выполнения вашего кода, просто подождав, пока появится более быстрое «железо». В наши дни, чтобы добиться более высокой производительности, требуется использовать несколько ядер, а значит, и приемы параллельного программирования.

## Конкуренция или параллелизм?

Эта книга рассказывает о конкуренции, тогда почему мы говорим о параллелизме? Хотя эти два термина часто используются взаимозаменяемо, конкуренция и параллелизм суть разные вещи.

### *Похожие, но разные*

*Конкурентная* программа всегда имеет несколько *потоков выполнения*. Потоки могут выполняться параллельно, но это не обязательно.

---

[1] https://ru.wikipedia.org/wiki/Закон_Мура

[2] http://www.gotw.ca/publications/concurrency-ddj.htm (Перевод на русский язык: http://habrahabr.ru/post/145432/ – *Прим. перев.*)

*Параллельная* программа обычно действует намного быстрее, чем последовательная, решая сразу (параллельно) несколько задач. При этом она необязательно имеет несколько потоков выполнения.

Конкуренцию можно считать аспектом предметной области, например когда программе требуется обрабатывать одновременно (или почти одновременно) возникающие события. Параллелизм, напротив, является аспектом решения, например с целью увеличить скорость работы за счет параллельной обработки разных блоков данных.

Вот как обозначает различия Роб Пайк (Rob Pike),[3]

- Конкуренция – это способ одновременного решения множества разных задач.
- Параллелизм – это способ одновременного выполнения разных частей одной задачи.

Так о чем эта книга, о конкуренции или о параллелизме?

**Джо спрашивает: в чем разница между конкуренцией и параллелизмом?**

Моя жена – учитель. Как и большинство учителей, она может заниматься сразу несколькими делами. В каждый конкретный момент она может делать только что-то одно, но способна делать несколько дел конкурентно. Слушая, как читает кто-то из ее учеников, она может прервать его, чтобы успокоить расшумевшийся класс или ответить на чей-то вопрос. Это – конкурентная работа, а не параллельная (она может делать только что-то одно).

Если ей в помощь дать помощника (один из них будет слушать ученика, читающего текст, а другой – отвечать на вопросы), мы получим вариант работы в паре, который можно назвать конкурентным и параллельным.

Представьте, что класс придумал поздравительные открытки и решает заняться их массовым выпуском. Решить эту проблему можно, поручив каждому ребенку нарисовать пять открыток. Это параллельная организация работы (если отбросить несущественные мелочи), но не конкурентная – выполняется только одна задача.

[3] http://concur.rspace.googlecode.com/hg/talk/concur.html

### За рамками последовательного программирования

Параллелизм и конкуренцию объединяет то, что обе эти парадигмы выходят за рамки модели последовательного программирования, в которой все процессы протекают последовательно, друг за другом. В этой книге мы будем рассматривать обе парадигмы, параллельного и конкурентного программирования (если бы я был педантом, я дал бы этой книге название «Семь моделей конкурентного и/или параллельного программирования за семь недель», но такое название едва ли уместилось бы по ширине обложки).

Конкуренцию и параллелизм часто путают между собой, потому традиционные потоки выполнения и блокировки не имеют прямого отношения к параллелизму. Если вы пожелаете задействовать несколько ядер с помощью потоков и блокировок, вы вынуждены будете создать конкурентную программу и запускать ее на параллельной архитектуре.

Такое ограничение весьма досадно, потому что программы, реализующие конкурентную модель, часто недетерминированы – они дают разные результаты в зависимости от точности синхронизации событий. При решении по-настоящему конкурентной задачи недетерминированность является естественным и вполне ожидаемым следствием. Параллелизм, напротив, необязательно подразумевает недетерминированность – операция удвоения всех чисел в массиве не приведет (по крайней мере, не должна) к недетерминированности только потому, что удвоение половины чисел выполняется одним ядром, а другая половина – другим. Языки с явной поддержкой параллелизма позволяют писать параллельный код, не внося недетерминированности.

## Параллельная архитектура

Большинство из нас привыкло связывать параллелизм с наличием множества ядер, однако в действительности современные компьютеры обеспечивают параллелизм на многих других уровнях. Причина, до недавнего времени объяснявшая увеличение скорости работы ядер с каждым годом, заключается в параллельном использовании всех тех дополнительных транзисторов, предсказанных законом Мура, – как на уровне отдельных битов, так и на уровне инструкций.

## Параллелизм на уровне битов

Почему 32-разрядный компьютер быстрее 8-разрядного? Дело в параллелизме. Чтобы сложить два 32-разрядных числа, на 8-разрядном компьютере потребуется выполнить последовательность 8-разрядных операций. На 32-разрядном компьютере, напротив, сложение можно выполнить в одно действие, параллельно обрабатывая все 4 байта, составляющие каждое 32-разрядное число.

Именно поэтому развитие компьютерной техники, которое мы наблюдали, двигалось от 8- к 16-, 32- и теперь уже 64-разрядным архитектурам. Однако подобный параллелизм имеет свои ограничения, из-за которых мы едва ли увидим в ближайшем будущем 128-разрядные компьютеры.

## Параллелизм на уровне инструкций

Современные микропроцессоры имеют высокую степень параллелизма за счет использования таких технологий, как *конвейерная обработка*, *неупорядоченное и спекулятивное выполнение команд*.

Как программисты мы часто могли просто игнорировать эти особенности, потому что процессоры тщательно маскировали свой параллелизм, создавая иллюзию последовательного выполнения операций.

Однако теперь эта иллюзия постепенно разрушается. Создатели процессоров не в состоянии больше увеличивать скорость работы ядер. А кроме того, так как мы входим в многоядерный мир, мы должны начинать учитывать тот факт, что инструкции выполняются не последовательно. Но подробнее об этом мы поговорим в разделе «Видимость памяти» главы 2.

## Параллелизм данных

Архитектуры с параллельной обработкой данных (иногда их называют SIMD, «single instruction, multiple data» – «с одним потоком команд и многими потоками данных») способны параллельно выполнять одни и те же операции с большим количеством данных. Нельзя сказать, что такие архитектуры подходят для решения всех видов задач, но они могут быть весьма эффективны при правильном применении.

Одним из таких правильных применений является обработка изображений. Например, чтобы увеличить яркость изображения, требуется увеличить яркость каждого пикселя. По этой причине развитие

современных GPU (graphics processing units – графические микропроцессоры) пошло по пути параллельной обработки данных.

## Параллелизм на уровне задач

Наконец мы достигли того, что многие из нас привыкли считать параллелизмом, – наличия множества процессоров. С точки зрения программиста, наиболее важной отличительной чертой многопроцессорных архитектур является модель представления памяти, а именно – является память распределенной или разделяемой.

В модели с *разделяемой памятью* все процессоры могут обращаться к любым участкам памяти, и все взаимодействия между процессорами осуществляются преимущественно через память, как показано на рис. 1.

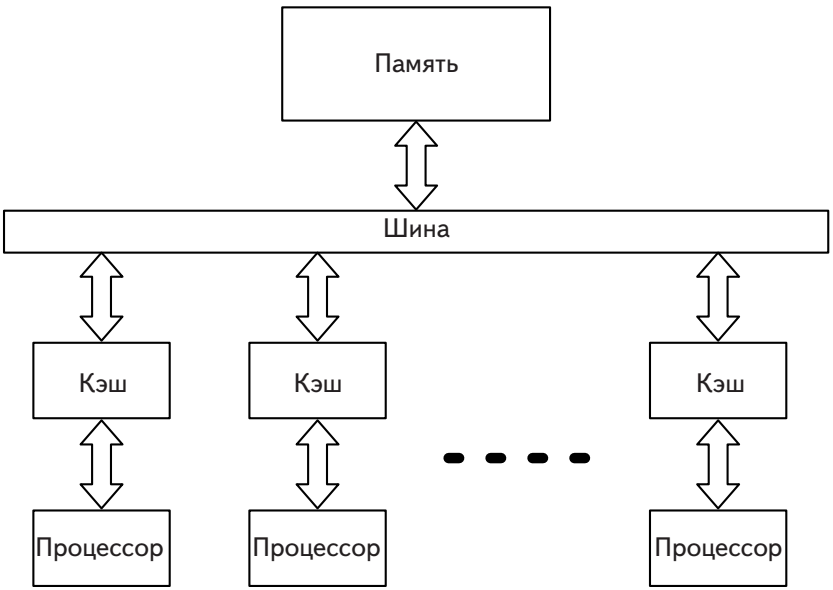


**Рис. 1.** Разделяемая память

На рис. 2 изображена модель системы с распределенной памятью, в которой каждый процессор имеет собственную локальную память, а взаимодействия между ними осуществляются через сеть.

Так как взаимодействия через память обычно осуществляются быстрее и проще, чем через сеть, разработка программ для архитектур с разделяемой памятью в общем случае дается гораздо легче. Но с уве-

личением числа процессоров разделяемая память становится узким местом – чтобы максимально использовать преимущества, которые дает наличие большого числа процессоров, необходим переход к использованию *распределенной памяти*. Использование распределенной памяти также является обязательным условием для тех, кто создает высоконадежные системы на основе множества компьютеров с целью избежать отказов в работе из-за выхода из строя аппаратных средств.

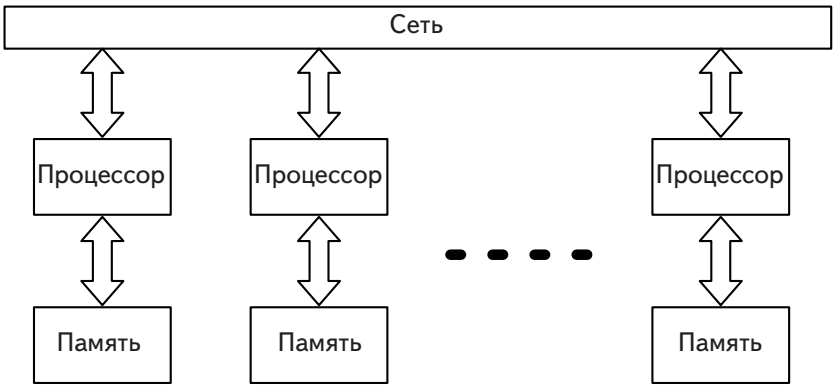


**Рис. 2.** Распределенная память

# Конкуренция: за рамками множества ядер

Конкуренция – это нечто большее, чем простое использование параллелизма: при правильном подходе применение приемов конкурентного программирования позволяет писать более отзывчивые, надежные, эффективные и простые программы.

## *Конкурентные программы для конкурентного мира*

Мир по своей природе является конкурентным, таковыми должны быть и программы, если мы хотим, чтобы они действовали эффективно.

Ваш мобильный телефон может проигрывать музыку, передавать данные по сети, следить за движениями ваших пальцев по экрану, и все это одновременно. Ваша IDE проверяет синтаксис исходного кода

в фоновом режиме, пока вы продолжаете ввод. Бортовой компьютер самолета одновременно опрашивает многочисленные датчики, отображает информацию для пилотов, повинуется командам и управляет рулями.

Конкуренция является ключом к созданию отзывчивых систем. Загружая файлы в фоновом режиме, вы не вынуждаете пользователей в расстроенных чувствах наблюдать песочные часы на экране. Обрабатывая сразу множество подключений, веб-сервер гарантирует, что один медленный запрос не затормозит обслуживание других.

## Распределенные программы для распределенного мира

Иногда географическая разбросанность оказывается ключевой особенностью решаемой задачи. Любое программное обеспечение, распределенное среди множества компьютеров, которые не действуют в жесткой связке, по своей природе является конкурентным.

Кроме того, распределенное программное обеспечение помогает повысить надежность. Вы можете разместить одну половину своих серверов в Европе, другую половину – в США, и тогда отключение электроэнергии в одном месте не приведет к глобальной остановке. Это обстоятельство подводит нас к теме способности противостоять внешним факторам.

## Надежные программы для непредсказуемого мира

Программы содержат ошибки, и иногда программы завершаются аварийно. Но даже если бы мы могли создавать программы, полностью лишенные ошибок, аппаратное обеспечение тоже не вечно и иногда выходит из строя.

Использование приемов конкуренции повышает надежность, или отказоустойчивость программ за счет *независимости* и механизма *обнаружения неисправностей*. Независимость играет важную роль, потому что ошибки в одной задаче не должны приводить к отказу от выполнения других задач. А механизм обнаружения ошибок помогает в случае ошибки в одной задаче (из-за аварийного завершения, «зависания» или выхода из строя аппаратных средств) известить другую задачу о необходимости проведения восстановительных действий.

Последовательные программы никогда не смогут быть столь же надежными, как конкурентные.