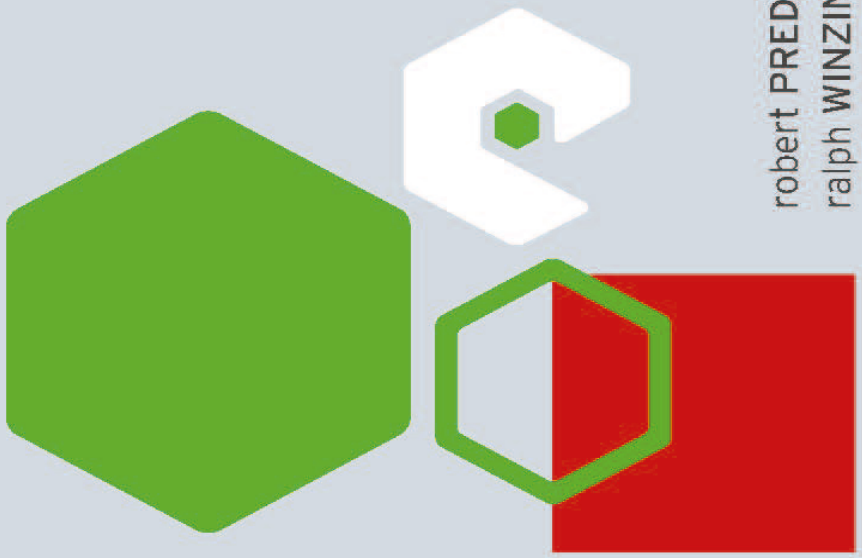
robert PREDIGER
ralph WINZINGER



# NODE.js

Professionell
hochperformante Software
entwickeln

HANSER

HANSER

**Robert Prediger**
**Ralph Winzinger**

# Node.js

Professionell hochperformante
Software entwickeln

**Die Autoren:**

*Robert Prediger,* Wang
*Ralph Winzinger,* Nürnberg



Bibliografische Information der Deutschen Nationalbibliothek: Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über **http://dnb.d-nb.de** abrufbar.


**www.hanser-fachbuch.de**

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Petra Kienle, Fürstenfeldbruck
Herstellung: Irene Weilhart

Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk

ISBN 978-3-446-43722-7
ISBN ePub 978-3-446-44721-9

Verwendete Schriften: SourceSansPro und SourceCodePro (Lizenz)
CSS-Version: 1.0

# Inhalt

## 3.7 Monitoring

### 3.7.1 Kommerzielle Monitoring-Services

## 3.8 Alternativen zu Node.js

### 3.8.1 Vert.x – die polyglotte JVM-Alternative

# Vorwort

Als Erstes möchten wir uns an dieser Stelle für Ihr Interesse an unserem Buch bedanken und die Gelegenheit nutzen, ein paar Worte über unser Buch zu verlieren. Das heißt, wir wollen uns, die Autoren, kurz vorstellen und unsere Motivation erklären, dieses Buch zu schreiben. Außerdem gibt es natürlich das eine oder andere über den Aufbau und das Format zu sagen, über die enthaltenen Beispiele und über JavaScript.

## Über die Autoren …

Es war von Anfang an klar, dass dieses Buch ein Gemeinschaftsprojekt sein soll. JavaScript – und damit auch Node.js – leben mit dem Ruf, keine professionelle Entwicklungsumgebung darzustellen. Insbesondere stößt man in der „Enterprise“-Welt auf diese Meinung. Um das objektiv zu hinterfragen, braucht es aber Beitragende aus beiden Lagern. So sind wir – Robert und Ralph – letztendlich aufeinandergetroffen, der eine seit erstaunlich langer Zeit professionell im Node.js-Umfeld tätig, der andere seit vielen Jahren als Software-Architekt in der Java-Enterprise-Welt unterwegs. Einer, der eine gewisse

Erwartungshaltung mitbringt, der andere, der diese Erwartungshaltung mit den richtigen Bibliotheken, Tools und Methoden adressieren muss. Für uns war es eine fruchtbare Zusammenarbeit und bedeutete obendrein noch großen Spaß. Wir hoffen, beides in den kommenden Kapiteln weitergeben zu können.

## Robert Prediger

Ich bin Robert, der Node.js-Freak. Mein größtes Projekt vor meiner Node.js-Zeit war die Entwicklung eines Accounting-Systems für internationale Hotelketten in Progress. Mit der Erstellung von Webapplikationen beschäftige ich mich seit gut 15 Jahren.

Vor knapp vier Jahren bin ich auf Node.js gestoßen und mich hat die Umgebung von Anfang an, vor allem aufgrund ihrer Stabilität und Geschwindigkeit, überzeugt und fasziniert. Mittlerweile sind diverse Projekte mit Node.js im Rahmen meiner web4biz Consulting ans Laufen gebracht worden.

Seit kurzem bin ich als Co-Founder an der whogloo Inc. beteiligt, deren Ziel es ist, eine Plattform zur Erstellung von Enterprise-Business-Applikationen zu erstellen – auf Basis von Node.js natürlich.

So habe ich in meinem täglichen Umfeld permanent mit Node.js zu tun. Das ist anstrengend, denn die Technik ist nach wie vor neu, teilweise bereits den Kinderschuhen entwachsen, dennoch sehr lernintensiv, wenn man permanent auf dem Laufenden bleiben will. Und das muss man auch, denn die Entwicklung in diesem Umfeld ist rasend schnell. Aber bereut habe ich es bisher nicht, mich komplett auf Node.js einzulassen. Es macht immer wieder Spaß und es fasziniert stets aufs Neue, mit welch teilweise

einfachen Mitteln man Programme entwickeln kann, die auch einem hohen Standard genügen.

Und ich bin gespannt, wo uns die Reise mit Node.js noch hinführt.

## Ralph Winzinger

Ich bin Ralph, der Java-Architekt. Ich arbeite für Senacor Technologies, ein Beratungsunternehmen mit vielen großen Kunden aus der Finanzwelt, bei denen ich Architekturen geplant, Entwicklungsprozesse eingeführt und Software erstellt habe, gerne auch mal auf der Frontend-Seite, aber hauptsächlich im Backend. EJBs, Webservices, Spring, JPA … das ist in der Regel meine Welt. Und zugegeben, ein großer Freund von JavaScript war ich lange Zeit nicht. Ich glaube, mich noch ungefähr an meine erste Reaktion auf Node.js erinnern zu können: „Serverside JavaScript? Wer braucht denn bitte eine Alert-Box auf dem Server?!?“ Wie gesagt, das war die allererste, spontane Reaktion.

Ich bin aber nicht mit der Java-Enterprise-Welt verheiratet. Es macht mir Spaß, neue Technologien zu ergründen und mir so manche Nacht mit Sourcecode um die Ohren zu schlagen. So ist es nicht verwunderlich, dass mich der Charakter von Node.js bald fasziniert hat.

„High-Scalability“ ist ein Thema, das auch im Enterprise-Umfeld immer wichtiger wird. Eine riesige Zahl von mobilen Endgeräten, die in jeder Branche zu einem extrem wichtigen Kundenkanal geworden sind, und die Entwicklung im Feld von „Internet of Things“ verbieten es geradezu, sich weiter exklusiv auf traditionelle Java Application Container und die damit verbundenen Technologien zu verlassen.

Ob sich ausgerechnet Node.js in meinen Projekten und bei meinen Kunden jemals etablieren wird, ist derzeit noch schwer zu beurteilen. Aber den Konzepten von Node.js wird sich unsere Branche sicherlich nicht verschließen können. Und ich hätte nun meine Meinung zum professionellen Einsatz von JavaScript.

### … ihre Helfer …

Zum Buch haben nur wir beide beigetragen? Nein. Eigentlich sollten vier Namen auf dem Cover stehen. Zwei geschätzte Kollegen und sehr gute Freunde haben diese Erkundungstour mit uns gestartet. Teils aus privaten, teils aus beruflichen Gründen mussten aber beide unser Buchprojekt verlassen, bevor wir es zum Abschluss bringen konnten. Trotzdem haben sie wichtige Impulse gegeben und auch wichtige Inhalte beigetragen. Wir sprechen von Victor Volle und Charles-Tom Kalleppally, beide in der Java-Welt verwurzelt, sehr versiert im Umgang mit Architektur, Design und Code und ebenfalls immer bereit, ausgetretene Pfade zu verlassen und sich auch mal neuen Technologien zuzuwenden.

An dieser Stelle großen Dank an Victor und Charly!

## … und ihre Motivation

Unsere Motivation haben wir in den einleitenden Sätzen ja schon skizziert: Der Node.js-Mensch wollte eine Lanze für seine Technologie brechen und der Java-Mensch „mal was Verrücktes tun“ – mit dem Ziel, die Entwicklung mit Node.js und JavaScript ein wenig ins rechte Licht zu rücken.

Wir haben die Maßstäbe aus der Java-Enterprise-Entwicklung Node.js und JavaScript angewandt, sowohl technisch als auch methodisch. Toolunterstützung für eine effiziente Arbeitsweise, Qualitätssicherung, um höchsten Ansprüchen zu genügen, State-of-the-Art Deployment und Monitoring … all das haben wir uns angesehen und die aus unserer Sicht und zum aktuellen Zeitpunkt besten oder vielversprechendsten Ansätze aus dem Node.js-Umfeld zusammengetragen.

Wir sind der Meinung, dass es für Node.js oder verwandte Technologien auch im Enterprise-Umfeld Bedarf gibt und dass sich Node hier durchaus verwenden lässt. Nichtsdestotrotz bleibt es zu einem gewissen Teil vorerst noch Kopfsache, ob man Node.js und JavaScript das nötige Vertrauen schenkt, geschäftskritische Teile der Systemlandschaft zu realisieren.

## Das Zielpublikum

Wir richten uns mit diesem Buch ganz klar an Entwickler und entwickelnde Architekten. Es ist ein sehr technisches Buch und wir gehen an vielen Stellen davon aus, dass entsprechendes Wissen vorhanden ist, um den einen oder anderen Sachverhalt zu verstehen.

Es werden vor allem diejenigen Freude am Buch haben, die nicht einfach nur möglichst viel Code in möglichst kurzer Zeit produzieren wollen. Das Design des Codes, die Paradigmen der Laufzeitumgebung, aber auch der Betrieb von Enterprise-Software sind uns wichtig und sollten es auch unseren Lesern sein.

Und nicht zuletzt wünschen wir uns ein wenig Spieltrieb. JavaScript ist eine sehr flexible Sprache, Node.js ein sehr schnell wachsendes Ökosystem, in dem es immer wieder Neues zu entdecken gibt. Man kann es sich hier noch viel weniger leisten, den Blick für die aktuelle Entwicklung und aktuelle Trends zu verlieren, als das im eher trägen Enterprise-Umfeld der Fall ist.

Die Arbeit mit JavaScript, mit seinem dynamischen Typsystem und den offenen Sourcecodes führt auch ganz natürlich dazu, dass man sich immer wieder in den Tiefen von fremdem Code verliert, um herauszufinden, wie eine Bibliothek funktioniert, welche versteckten Optionen eine Funktion eventuell noch bietet.

Das alles muss man mögen. Wir mögen es und wir hoffen, unsere Leser ebenfalls.

# Das Buch

Es ist nicht nur wichtig, wer ein Buch verfasst hat und weshalb ein Buch geschrieben wurde. Nachdem wir einen bestimmten internen Aufbau verfolgt haben und auch über die verwendete Formatierung die Verständlichkeit erhöhen wollten, möchten wir unsere Richtlinien an dieser Stelle kurz skizzieren.

## Aufbau

Wir haben unser Buch in drei Teile gegliedert.

Der erste Teil soll dafür sorgen, dass sich unsere Leser langsam an die Welt von Node.js gewöhnen. Die Geschichte, die Idee, das

Tooling stehen hier im Vordergrund. Die Entwicklung an sich ist noch kein Thema, allenfalls am Rande.

Im zweiten und weitaus größten Teil bewegen wir uns durch die Schichten und Problemfelder einer Enterprise-Anwendung. Angefangen beim Dateisystem über Webanwendungen und Service-Schnittstellen bis hin zu einer ausgefeilten Testunterstützung werden die typischen Fragestellungen anhand von vielen vorgestellten Modulen und Codebeispielen beantwortet.

Der letzte Teil kümmert sich im Wesentlichen um die Zeit nach der Entwicklung. Wie nehme ich eine solche Anwendung in Betrieb und wie halte ich sie in Betrieb? Deployment, Monitoring oder auch Performance sind die Themen in diesem Teil. Und nicht zuletzt ein Blick auf eine Alternative – ähnliche Konzepte und Performance, aber nicht (nur) JavaScript.

## Formate

Bezüglich der verwendeten Formate haben wir uns zurückgehalten. Natürlich sind alle coderelevanten Passagen als solche zu erkennen. Schreiben wir beispielsweise im Text über Code, so ist das wie hier `console.log(„this is code“)` entsprechend gekennzeichnet. Längere Codepassagen sind natürlich nicht im Fließtext untergebracht, sondern erhalten ihren eigenen Abschnitt:

```
var fs = require(„fs“);
var i = 5;
console.log(„this is still code“)
```

Auf dieselbe Art sind auch Kommandozeileninteraktionen formatiert. Auch diese können im Fließtext (`npm install express`) oder in einem größeren Block zu finden sein.

```
$ node app.js
INFO: this is some console output
```

Und dann sind da noch Modulangaben. Beziehen wir uns im Text auf Module wie *Express* oder *restify*, so sind diese immer kursiv dargestellt. Manchmal gibt es in diesem Zusammenhang auch eine gewisse Grauzone. So kann es sein, dass man von dem Modul *npm* spricht oder aber von dem Kommando `npm`. Damit ist dasselbe Wort dann vielleicht auch im selben Abschnitt verschieden formatiert.

## Beispiele

Die allermeisten unserer Beispiele sind tatsächlich funktionierender Code, der seine Korrektheit zunächst in einem Test oder wenigstens durch eine Ausführung unter Beweis stellen musste. Erst dann durfte er aus der IDE über copy & paste ins Buch springen. Natürlich können auch dabei immer wieder mal Fehler passieren, die bitten wir zu entschuldigen.


### DIE CODEBEISPIELE

Die Codebeispiele in diesem Buch müssen natürlich nicht abgetippt werden; sie stehen Ihnen online auf einem GitHub Repository zur Verfügung:

**https://github.com/WinzingerPrediger/node.js**

# Die Welt von JavaScript

Eines liegt uns noch sehr am Herzen. Wie erwähnt, ist die Welt von JavaScript und Node.js sehr kurzlebig. Ein Modul, das heute noch *das* Modul für eine bestimmte Problemstellung ist, wird morgen vielleicht schon von einem neuen Modul abgelöst, welches riesigen Zuspruch in der Community erhält und schnell zum De-facto-Standard wird.

Natürlich stellen wir im Buch ganz konkrete Module vor und laufen damit potenziell Gefahr, dass diese einige Monate nach dem Erscheinungstermin des Buchs schon nicht mehr existieren oder niemanden mehr interessieren. Wir haben aber auch immer versucht, die aus unserer Sicht relevante Funktionalität herauszustellen – welches Problem wird von dem Modul eigentlich gelöst? Mit diesen Hintergrundinformationen sollte es einfacher sein, die Eignung neuer Module zu hinterfragen oder gezielt nach Alternativen zu suchen.

## „Lerne ich hier JavaScript?“

Diese Frage lässt sich einfach beantworten: nein. Vielleicht kann man sich an der einen oder anderen Stelle einen kleinen Kniff abschauen, aber wir geben hier noch nicht mal eine gezielte Einführung in JavaScript.

Es gibt sehr viel Literatur und sehr viele Quellen im Internet, die das leisten können. Wir haben uns das nicht zum Ziel gesetzt.

Viel Spaß beim Lesen, Lernen und Spielen,

*Robert und Ralph*

# 1 Hello, Node.js

Was ist Node.js eigentlich und wie arbeitet man nun damit? Das ist die zentrale Frage, die im ersten Teil des Buchs beantwortet werden soll. Hier geht es nicht darum, welche Bibliotheken bei der Erstellung großer Softwareprojekte besonders hilfreich sind oder wie man dafür sorgt, dass eine Node.js-Anwendung stabil läuft, auch wenn sie eine Unmenge von Anfragen abarbeiten muss. Nein, dieser Teil des Buchs liegt zeitlich vor den ersten Zeilen Anwendungscode, die hoffentlich bald entstehen.

Am Anfang dürfen natürlich ein paar einführende Worte zu Node.js nicht fehlen. Woher kommt dieses Node eigentlich und was ist daran besonders? Die technischen Tiefen werden erst im letzten Teil des Buchs erreicht, aber eine Idee von der Funktionsweise sollte man trotzdem von Anfang an haben.

Und neben der Idee bedarf es natürlich auch noch des notwendigen Handwerkszeugs. Node.js braucht eine Laufzeitumgebung – für welche Systeme existiert die und wo kann man sie bekommen? Mit welchen Tools kann Quellcode editiert werden? Wo findet man Bibliotheken und wie kann man diese benutzen? Wenn auch diese Fragen alle geklärt sind, hat man keine Ausreden mehr, sich die Finger am Code schmutzig zu machen. Und auch wenn man kein Enterprise-System realisieren

möchte – Spaß kann man mit Node.js allemal haben. Und das ist schon mal ein ganz wichtiger Punkt!

## 1.1 Einführung in Node.js

Node.js hat ein unglaubliches Wachstum hingelegt. Von der ersten Ankündigung durch Ryan Dahl 2009[1] bis heute (Ende 2014) sind nur fünf Jahre vergangen. Das Node.js Repository auf GitHub ist aktuell auf Platz zwei der „most starred repositories“[2] (und unter den ersten 15 der „most forked repositories“[3]).

Auf der anderen Seite gibt es auch einen geradezu aggressiven „Backlash“[4]. Natürlich ist das „trolling“ – sinnvolle Argumente sind in solchen Fällen nur selten auszumachen. Auch Beiträge wie „Node.JS Is Stupid And If You Use It So Are You“[5] fallen ganz klar in diese Kategorie.

Warum ist das so? Node.js ist immer noch vergleichsweise neu, hat sich aber schon an vielen Stellen einen festen Platz in der Anwendungslandschaft ergattert. Nichtsdestotrotz gibt es aber immer noch eine Art missionarischen Effekt. Überzeugte Anwender möchten Node.js weiter aus seiner Nische holen und versuchen, andere von seinen Vorteilen zu überzeugen. Dann gibt es natürlich auch den gegenläufigen Effekt: Jemand probiert etwas aus, von dem er gerade Lobpreisungen über sich hat ergehen lassen. Und dann passt es nicht zu seinem Problem, seinen Erwartungen oder einfach seiner aktuellen Denkweise. Er hat Zeit investiert und ist enttäuscht. Also muss die Technologie nutzlos, unsinnig oder überflüssig sein. Unter Umständen reicht

für viele Entwickler alleine schon die Tatsache, dass Node.js auf JavaScript basiert, um es als indiskutabel einzustufen.

Weshalb haben wir nun ein Buch über Node.js geschrieben? Nicht weil wir missionieren möchten und denken, dass Node.js das Beste seit geschnitten Brot ist. Es gibt Situationen, in denen Node.js eine gute Lösung darstellt, aber auch Situationen, in denen man vielleicht zu einer Alternative greifen sollte. In den nächsten Kapiteln versuchen wir nicht nur Vorteile, sondern auch Nachteile zu zeigen – insbesondere auch im Hinblick auf „professionelle Softwareentwicklung“. JavaScript hat nicht den besten Ruf in unserer Industrie und es wird schnell behauptet, dass es nicht als Basis für unternehmenskritische Anwendungen dienen kann. Es ist natürlich schwierig, hier absolute K.O.-Kriterien für oder gegen eine solche Verwendung zu finden – Node.js ist sicherlich erwachsen genug, um nicht sofort auszuscheiden. Wir versuchen aber, viele Aspekte zu betrachten, die in den Bereich der professionellen Softwareentwicklung fallen, und beleuchten, wie diese Aspekte in Node.js und JavaScript behandelt werden. Unserer Meinung nach schneiden Node.js und JavaScript hier nicht immer optimal ab. Allerdings hängt es vom konkreten Szenario ab, ob ein ganz bestimmter Aspekt nun benötigt wird oder nicht. Node.js hat es auf jeden Fall verdient, eine Chance zu bekommen. Es kann vielleicht nicht in allen Bereichen mit etablierten Sprachen und Umgebungen wie dem Java-Ökosystem mithalten, aber es gibt durchaus Situationen, in denen Node.js einfach auf der Überholspur vorbeizieht …

Also, was ist der Nutzen von Node.js? Oder besser: Welches Problem löst Node.js besser als andere Technologien?

Eines der wichtigsten Probleme für Internetanwendungen ist Skalierbarkeit. Wenn die Benutzeranzahl steigt, soll der Ressourcenverbrauch nach Möglichkeit linear steigen. Von den relevanten Ressourcen CPU, I/O und RAM soll Node.js vor allem die Skalierbarkeit bei I/O-intensiven Anwendungen verbessern.

Um dies zu erreichen, setzt Node.js vollständig auf asynchrone I/O-Zugriffe. Wann immer Node.js auf eine Datenbank, einen Webservice oder auf das Dateisystem zugreift, erfolgt der Aufruf asynchron. Was macht das für einen Unterschied?

Im synchronen Fall könnte ein Datenbankaufruf folgendermaßen aussehen:

```
result = database.query("SELECT * FROM user");
result.get…
```

Der aktuelle Thread wartet auf das Ergebnis und wird „geweckt“, wenn das Ergebnis vorliegt (s. **Bild 1.1**). Dieses Verfahren funktioniert gut und ist aus konzeptioneller Sicht leicht zu begreifen. Als Entwickler kann man so programmieren, als ob man keine Nebenläufigkeit hätte. Man muss sich (meist) nicht um Synchronisation kümmern. Und das Wichtigste: Die Kosten für das Wechseln von einem Thread zum nächsten sind vernachlässigbar.

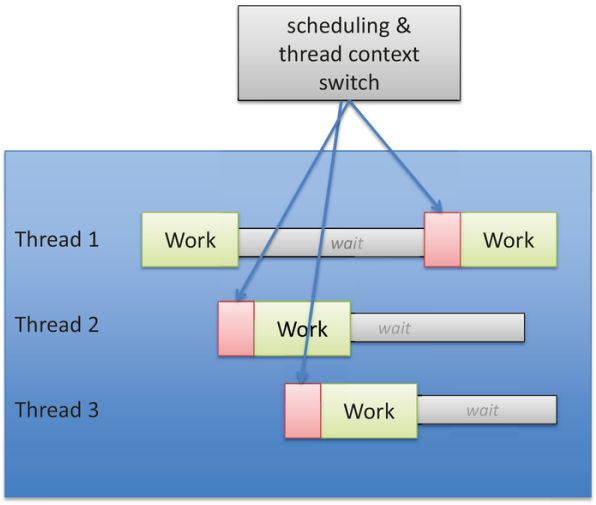



**Bild 1.1** multi-threaded Server

Doch was tun, wenn die Kosten für den Wechsel nicht mehr vernachlässigbar sind? Wenn auf den „Thread Context Switch“ ein signifikanter Anteil an der Gesamtlaufzeit entfällt?

Hinzu kommt, dass Threads Hauptspeicher verbrauchen. Ein Java-Applikationsserver hat oft 200 bis 300 Threads. Auf einem 64-Bit-Linux-System wird für einen Thread 1 MB für den Stack reserviert. Somit werden ca. 250 MB verbraucht, die nur für die Threads benötigt werden. In vielen Anwendungen spielt das keine Rolle, da für den Heap allein 8 GB veranschlagt werden. Doch wenn die Anwendung

- sehr viele parallele Zugriffe hat,
- viel I/O (Datenbank, Dateisystem etc.) benötigt,

wird es eng. Was heißt in diesem Zusammenhang „viel“? 100 Zugriffe pro Minute sind (unserer *Meinung* nach) noch nicht viel. Spätestens bei 100 parallelen Zugriffen pro Sekunde wird Node.js definitiv interessant.

## Keine Threads!

Was macht Node.js anders? Node.js verwendet nur einen Thread. Das ist alles. Nun ja, das ist nun etwas plakativ formuliert und technisch auch nicht ganz korrekt, aber in einer ersten Näherung ist es das, was Node.js anders macht als die meisten gewohnten Umgebungen. Wie handhabt Node.js dann Hunderte von parallelen Zugriffen? Es arbeitet einfach einen Auftrag nach dem anderen ab – und wann immer dann auf einen langsamen I/O-Zugriff gewartet werden muss, wird der aktuelle Auftrag einfach beendet und ein neuer Auftrag für die Verarbeitung der Antwort erzeugt.

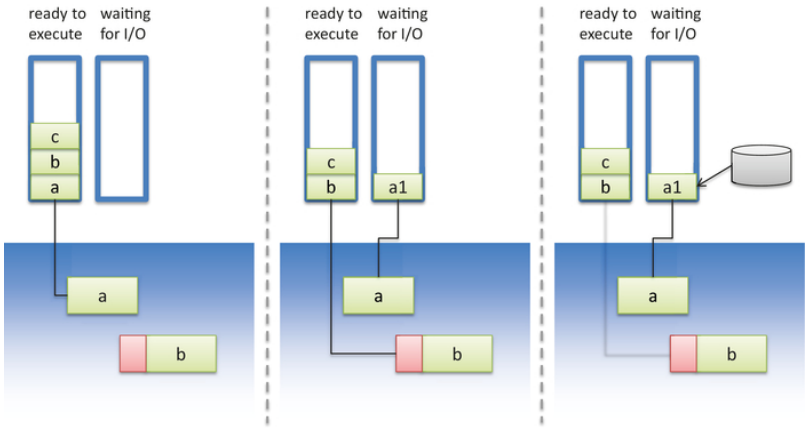



**Bild 1.2** Node.js Event-Loop

In der Abbildung „Node.js Event-Loop“ ist der Ablauf – stark vereinfacht – skizziert. Es gibt eine Warteschlange mit Aufträgen, die bereit zur Verarbeitung sind („ready to execute“). Sobald der Verarbeitungs-Thread frei ist, beginnt er mit der Abarbeitung des Auftrags „a“. Wenn dann jedoch ein I/O-Zugriff erfolgt, wird die Abarbeitung des Auftrags beendet und ein Folgeauftrag „a1“ erzeugt. Dieser Auftrag wird jedoch erst dann in die „ready-to-

execute“-Queue eingestellt, wenn der I/O-Zugriff abgeschlossen ist.

Bei Node.js muss der Entwickler jeden Folgeauftrag als „Callback-Funktion“ selbst definieren – ein Vorgehen, das jedem JavaScript-Entwickler sehr vertraut sein dürfte und zu Code wie dem folgenden führt:

```
database.query(
  "SELECT * FROM user",
  function(result) {
    result…
  }
);
```

Wirklich schwer zu lesen wird es, wenn bei der Verarbeitung der Antwort wiederum ein I/O-Zugriff mit der nächsten Callback-Funktion zu schreiben ist. Statt der geschachtelten anonymen Funktionen kann man natürlich auch mit explizit benannten Funktionen arbeiten:

```
function schritt1(result) {
  ...
  database.query("...", schritt2);
}
function schritt2(result) {
  ...
}
database.query("...", schritt1);
```

Damit wird es jedoch schwerer, den tatsächlichen Ablauf in den Codestrukturen nachzuvollziehen.

Da es vielen Programmierern so geht, gibt es eine große Menge an Node.js-Bibliotheken, die einen besser lesbaren Code ermöglichen. Zwei solche Bibliotheken werden im zweiten Teil im Abschnitt 2.8 gezeigt.

Wenn es keine Threads gibt, stellt sich die Frage: Wie nutzt man Multi-Prozessor und Multi-Core-Maschinen sinnvoll aus? Wenn man Node.js auf einer solchen Maschine laufen lässt, wäre gerade mal ein Core ausgelastet. Deshalb sollte man pro Core eine Node.js-Instanz laufen lassen und noch einen Core für andere Aufgaben freihalten. Das bedeutet aber auch, dass man sich um das Thema Session State kümmern muss.

## Kriterien für den Einsatz von Node.js

Durch die Vermeidung von Threads benötigt Node.js tendenziell weniger Hauptspeicher und durch den Wegfall des Thread-Context-Wechsels auch weniger CPU-Zyklen; was aber – wie gesagt – erst bei einer hohen Anzahl konkurrierender Zugriffe relevant wird.

Machen aber die Kosten für die Server einen relevanten Anteil am Budget aus, sollte man Node.js wirklich in Betracht ziehen. Beispielsweise hat LinkedIn seine Mobile-App von Rails auf Node.js umgestellt und damit die Anzahl der benötigten Server von 30 auf drei reduzieren können[6]. Natürlich macht man bei einem „rewrite“ vieles besser als beim ersten Mal. Man kennt die kritischen Stellen, kann sich auch des angewachsenen Mülls entledigen: Die Reduktion lässt sich sicherlich nicht komplett

Node.js zuschreiben. Dennoch sind wir (und LinkedIn) überzeugt, dass Node.js einen wichtigen Anteil an der Reduktion hat.

Neben den Vorteilen im Betrieb für Anwendungen mit hohem I/O und vielen parallelen Zugriffen gibt es noch einen Vorteil in der Entwicklungsphase: *eine* Sprache für das Frontend und das Backend. „The best tool for the job“ – also für das jeweilige Problem die passende Sprache zu verwenden – kann eine gute Idee sein. Aber dann läuft man Gefahr, eine sehr heterogene Anwendungslandschaft pflegen zu müssen.

Auf der anderen Seite führt die Trennung in Frontend- und Middleware-Team oft zu Barrieren im Austausch, Missverständnissen und doppelter Arbeit. Wenn man also *ein* Team aufbauen kann, das nur eine Programmiersprache benutzt, ist zumindest diese Barriere niedriger. Natürlich sind die benötigten Fähigkeiten für Frontend-Entwicklung anders als die für die Middleware-Entwicklung. Es sind andere Randbedingungen zu beachten. Aber ein Feature vom Frontend bis zu den Backend-Aufrufen von einem Entwickler bauen zu lassen, der – im wörtlichen Sinne – alles „von vorne bis hinten“ versteht und überblickt, reduziert Kommunikationsbarrieren immens. So meint zum Beispiel Ben Galbraith, Vice President for mobile engineering bei Walmart: „We’ve been fascinated for a long time by end-to-end JavaScript.“ [7]

Ab einer bestimmten Teamgröße kommen andere Effekte hinzu. Je größer das Team, desto mehr Zeit wird für Kommunikation benötigt. Ab acht bis zehn Menschen sollte man Teams aufteilen, um diesem Effekt entgegenzuwirken. Aber dennoch muss man sich gut überlegen, ob man die Aufteilung anhand der Applikationsschichten (also Frontend, Middleware, Backend) vornimmt oder entlang von Funktions-Clustern. Stimmen aus