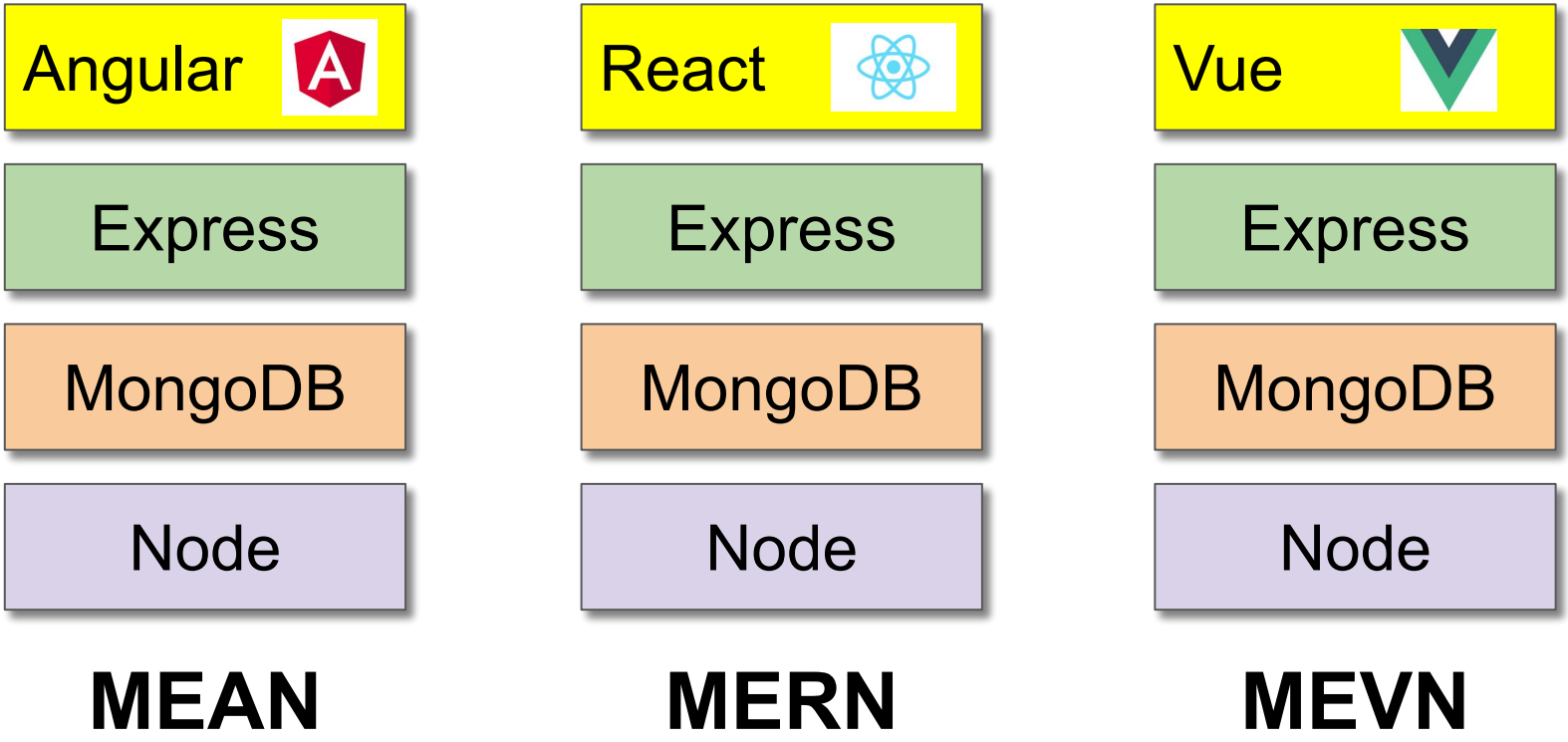




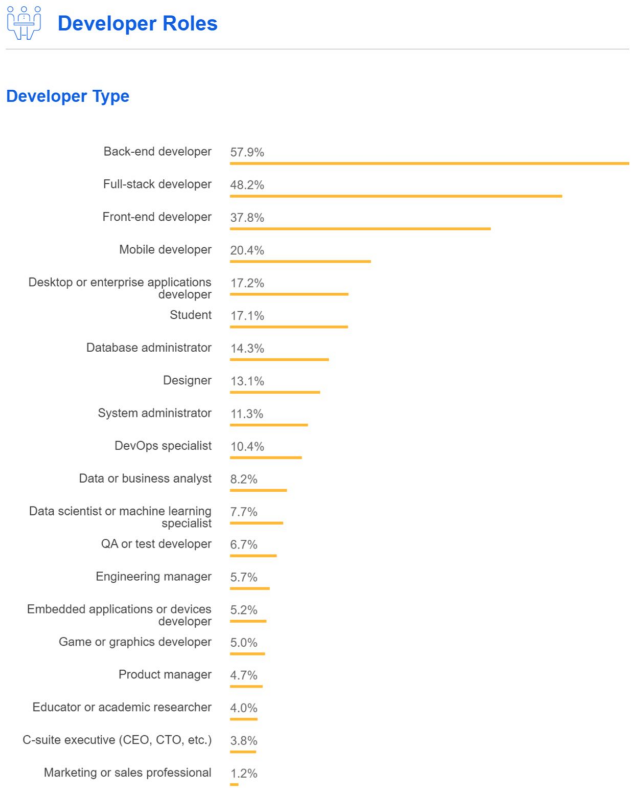
# MEVN-Stack

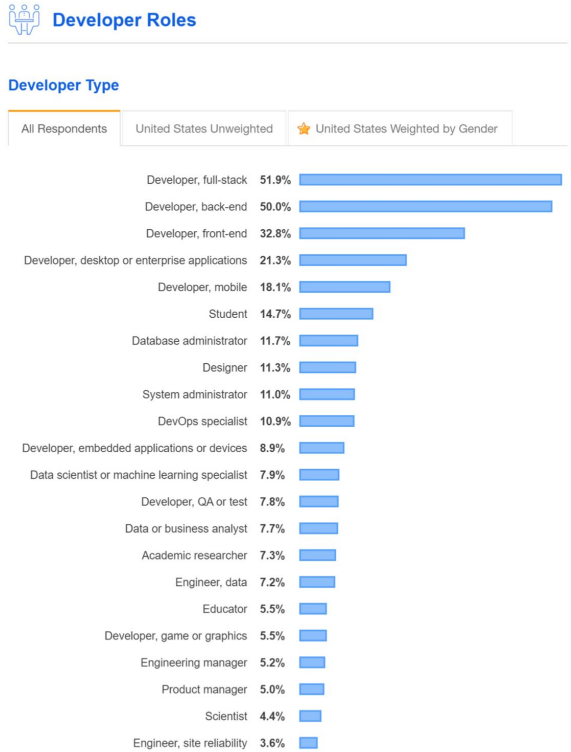
## Full Stack Web Development mit MongoDB, Express.js, Vue.js und Node.js

# Full Stack JavaScript Web Development



# Full-Stack Web Developer Nr.1 seit 2019




Stack Overflow Survey 2018 (links) und 2019 (rechts)

# Populäre Web Stacks:

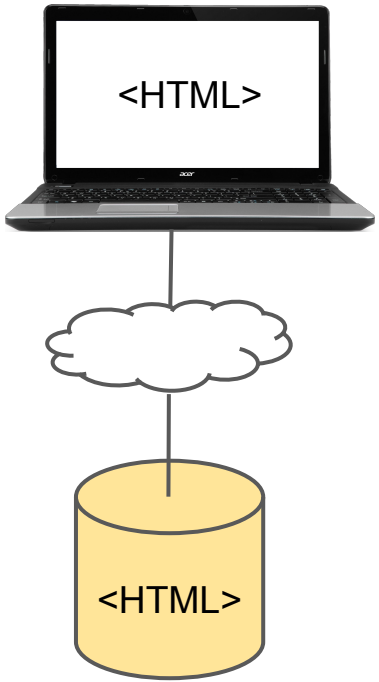
- LAMP - Linux, Apache, MySQl und PHP/Python/Perl
- LAPP - Linux, Apache, PostgreSQL und PHP/Python/Perl
- WAMP - Windows, Apache, MySQL und PHP/Python/Perl

---

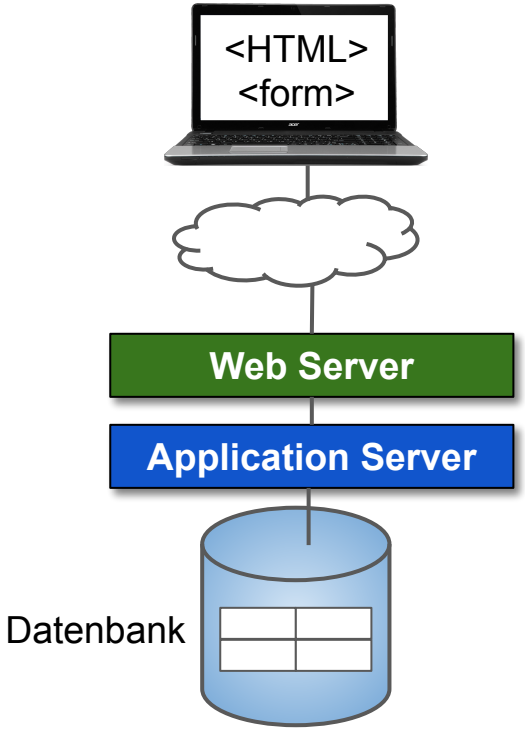
- MERN - Mongo, Express.js, React.js und Node.js
- MEAN - Mongo, Express.js, Angular.js und Node.js
- MEVN - Mongo, Express.js, Vue.js und Node.js

# Node.js für das neue Web-Paradigma am besten geeignet

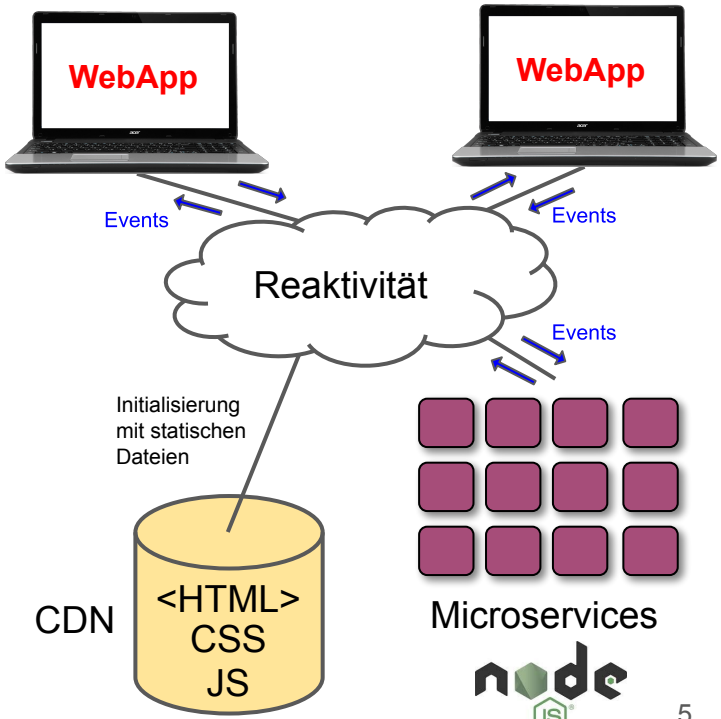
bis 1997
statisches HTML
Paradigma: **Digital Library**




ab 1997
Paradigmen:
**Dokumentverarbeitung**
**Formulare**




heute
Paradigma: **Das Nervensystem der Gesellschaft**




http://gigaom.com/cloud/node-js-and-the-javascript-age/

# Gliederung




1. Node.js
2. npm
3. Express.js
4. MongoDB
5. Ausblick

# Ryan Dahl, 2010: Erste Node.js-Präsentation

Ryan Dahl, 2010
https://en.wikipedia.org/wiki/Node.js

- Server-side JavaScript
- Built on Google´s V8
- Evented, non-blocking I/O. Similar to $\text{EventMachine}_{\text{Ruby}}$ or $\text{Twisted}_{\text{Python}}$.
- CommonJS module system.
- 8000 lines of C / C++, 2000 lines of JavaScript, 14 contributors

# Google V8: Lars Bak

- Däne
- 1991 Programmierer bei Sun
  - HotSpot
    - Java VM
    - Just-in-time Compilation
- 1994 Gründung Animorphic Systems
  - bessere VM
  - Smalltalk, StrongTalk, Self, ...
- 1997 Sun kauft Animorphic
  - Lars Bak leitender Ingenieur bei Sun
  - Optimierung der Java VM
- 2000 zurück in Dänemark
  - Prof. Aarhus Uni
- 2003 Google V8


# Geschichte der Parallelität in CPU & OS

| Jahr | Prinzip | OS-Prinzip | Beispiele |
|---|---|---|---|
| 1990 | 1 CPU = 1 Prozess | **Single Process** | DOS |
| 1995 | 1 CPU + viele Prozesse | **Cooperative Multitasking** | Windows 95, 98 |
| 1996 | 2 CPUs + viele Prozesse | **Preemptive Multitasking** | Windows NT, 2000, Linux |
| 2000 | 2++ CPUs<br>+ viele Prozesse<br>+ viele Threads | **Symmetric Multi Threading (SMT)** | Pentium 4 (Intel): Intel **Hyper-Threading**: New CPU instruction code for parallelization |

# Prozesse vs. Threads vs. Events

| Processes | Threads | Events |
|---|---|---|
| Top-level execution container | runs inside a process | runs inside a thread |
| separate memory space | shared memory space | same memory |
| Communicate via Inter-Process Communication (IPC) | Many communication options, Synchronisierung auf gemeinsamen Speicher notwendig, Mutex, Semaphoren, Critical Region, Rendezvous, Monitore, Warteschlangen, ... | **keine Synchronisierung erforderlich** |

# Design Goals

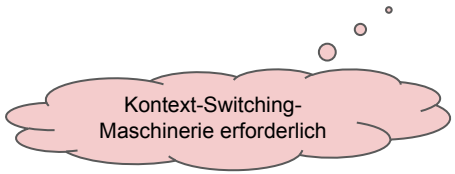
Ryan Dahl, 2010
https://en.wikipedia.org/wiki/Node.js

- **Low-Level Concurrency**
- **Stream everything**
- **purely evented**
  - perfect for JavaScript-Programmers
- **non-blocking**
- highly concurrent

# Vergleich

## Synchrone Programmierung
## Blocking I/O

```
result = db.query("select * from T");
// wait for response
// use result
```




blockiert den ganzen Prozess

## Asynchrone Programmierung
## Non-Blocking I/O

```
var result =
 db.query("select * from T",
    function (data) {
        // use result via Callback
} );
```

erlaubt sofortige Rückkehr zur Event Loop

| | |
|---|---|
| lineare Programmierung | Programmierung von **Callbacks** |
| **einfacher** zu verstehen | schwieriger zu verstehen |
| weniger Kontrolle über Parallelität | mehr Kontrolle über Parallelität |

# Blocking vs. Non-Blocking

```
const fs = require('fs');
const data = fs.readFileSync('/file.md'); // blocks here until file is read
console.log(data);
// moreWork(); will run after console.log
```

- **ohne Callback**
- mit Blockierung

And here is a similar, but not equivalent asynchronous example:

```
const fs = require('fs');
fs.readFile('/file.md', (err, data) => {
  if (err) throw err;
  console.log(data);
});
// moreWork(); will run before console.log
```

- **mit Callback**
- ohne Blockierung

In the first example above, `console.log` will be called before `moreWork()`. In the second example `fs.readFile()` is **non-blocking** so JavaScript execution can continue and `moreWork()` will be called first. The ability to run `moreWork()` without waiting for the file read to complete is a key design choice that allows for higher throughput.

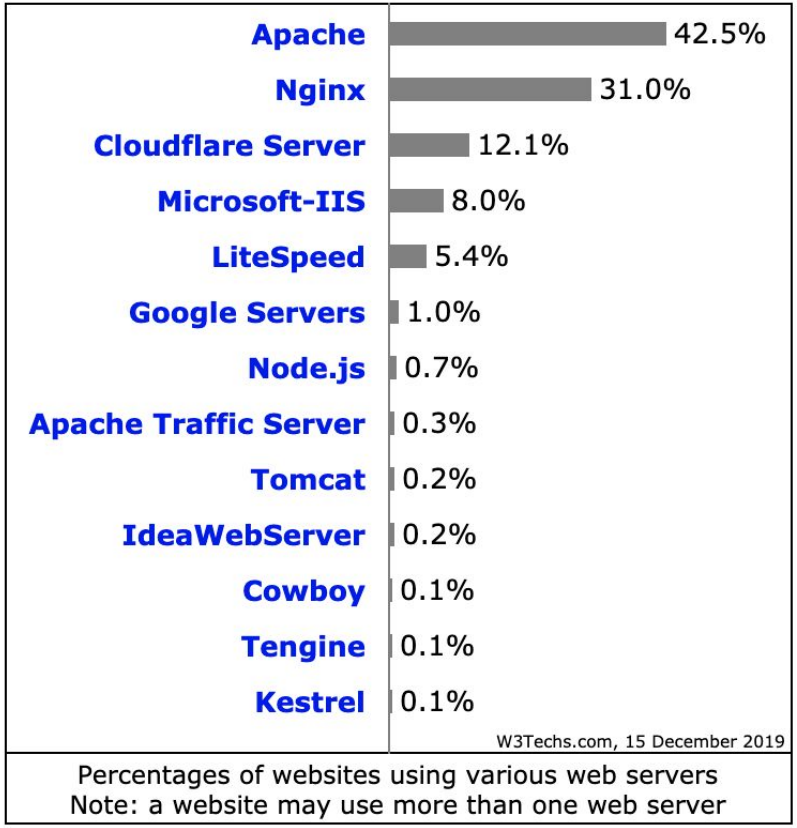
https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

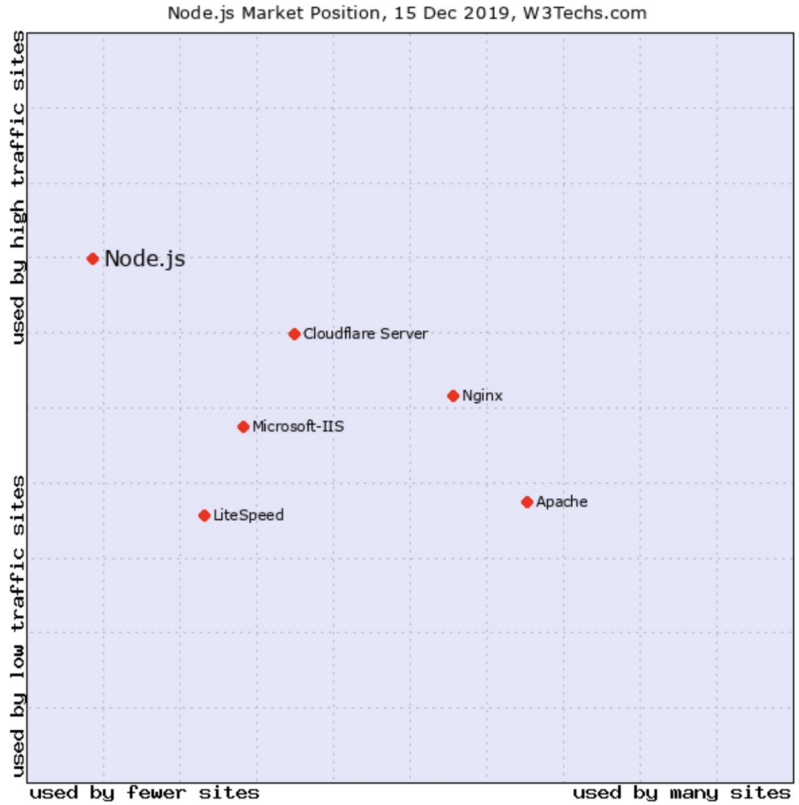
# Server-Architekturen




| LAMP | Java EE | Node.js |
|---|---|---|
| multi-process | multi-threaded | single-threaded |
| 1 process per request | 1 thread per request | 1 event per request |
| PHP interpretiert | compiliert | JS interpretiert |
| blocking I/O<br>process switching | blocking I/O<br>thread switching | non-blocking I/O<br>event-driven |
| beliebtester WWW-Server | beliebt in Unternehmen | 20% schneller als Java EE |
| Verschiedene Experten für Frontend und Backend | Verschiedene Experten für Frontend und Backend | JS-Programmierer für beide Seiten, Client & Server |

# Comparison of the usage 2019




https://w3techs.com/technologies/overview/web_server/all

# Server-side JavaScript

"The Node.js is 20% faster than the Java EE solution for the problem at hand. That amazed me. An **interpreted language** as fast as a **compiled language** on a VM in which **years of optimization** have gone into. Not bad at all!"

https://dzone.com/articles/performance-comparison-between

# Wie schafft Node.js 10.000 Requests / sec ?

Singlethreaded Server:

```
user do an action
       |
       v
 application start processing action
   └──> loop ...
          └──> busy processing
 end loop
   └──> send result to user
```

```
user do an action
       |
       v
 application start processing action
   └──> make database request
          └──> do nothing until request completes
 request complete
   └──> send result to user
```

99% CPU time waiting for the database to return

Multithreaded network app:

```
request ──> spawn thread
              └──> wait for database request
                     └──> answer request
request ──> spawn thread
              └──> wait for database request
                     └──> answer request
request ──> spawn thread
              └──> wait for database request
                     └──> answer request
```

Singlethreaded event loop:

```
request ──> make database request
request ──> make database request
request ──> make database request
database request complete ──> send response
database request complete ──> send response
database request complete ──> send response
```

https://stackoverflow.com/questions/34855352/how-in-general-does-node-js-handle-10-000-concurrent-requests

# Nodejs.org


Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

New security releases to be made available Dec 17, 2019

Download for macOS (x64)

12.13.1 LTS
Recommended For Most Users

13.3.0 Current
Latest Features

Other Downloads | Changelog | API Docs

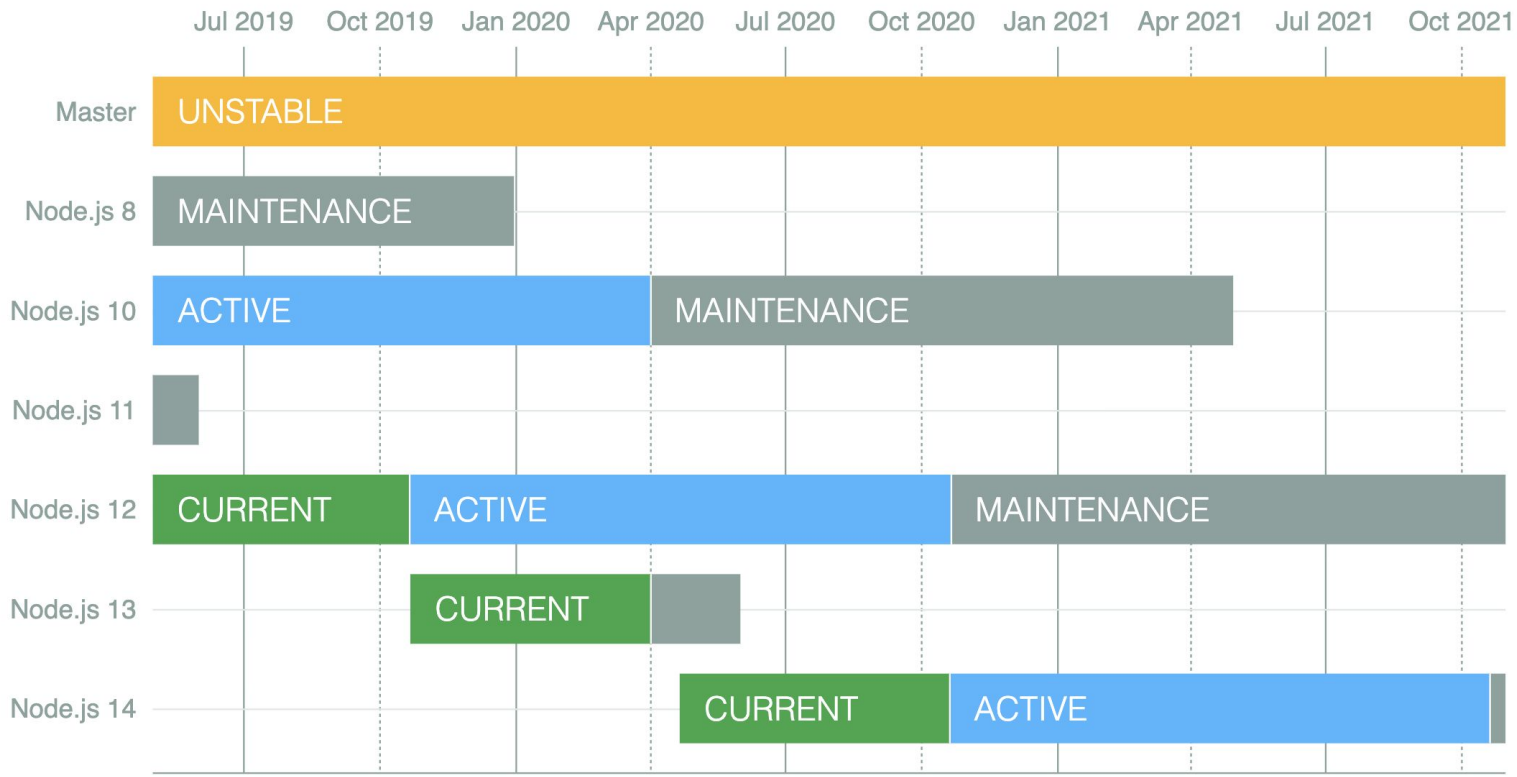
Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

https://nodejs.org/

# Gerade Release-Nummern = Long Term Support (LTS)




https://github.com/nodejs/LTS#lts-schedule1

# Node Version Manager (nvm)

## Listing versions

If you want to see what versions are installed:

```
nvm ls
```

If you want to see what versions are available to install:

```
nvm ls-remote
```

## Usage

To download, compile, an

```
nvm install node
```

And then in any new shell

```
nvm use node
```

https://github.com/creationix/nvm

# Hello World in Node.js

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

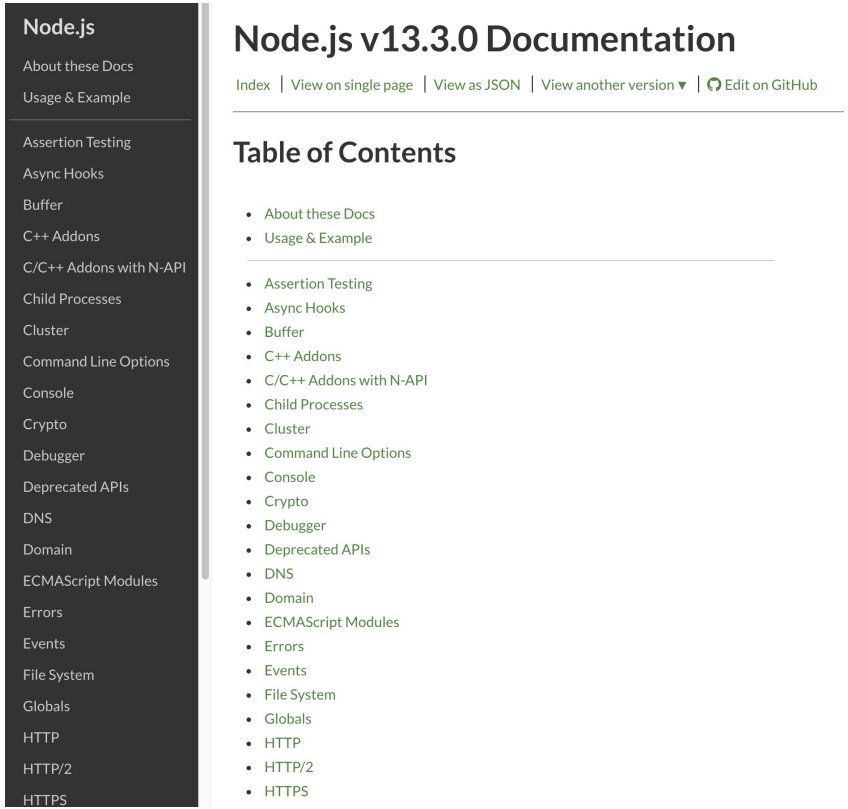
wie im Browser:

- nur 1 Thread für User
- ⇒ no concurrency

Parallelität nur in der Node.js-VM, nicht in den Programmen, die man für Node.js schreibt.

Idee: Das Betriebssystem und die Datenbank haben bereits Threads. Das reicht. Der Anwendungsprogrammierer sollte damit nicht belastet werden.

# Node.js Documentation




"File System"-Modul importieren

```
const fs = require('fs');

fs.readFile('text_file.txt', 'utf8', (err, data) => {
 if (err) throw err;
 console.log(data);
});
```

https://nodejs.org/en/docs/

# Unterschiede: Browser ↔ Node.js

| Browser | Node.js |
|---|---|
| Globales Objekt heißt "`window`" | Globales Objekt heißt "`global`" |
| `var x` ⇒ `window.x` | `var x` ⇒ keine Property in `global` |
| Keine Module (bis ES6 `import`) | Module mit `require()` importieren |
| DOM via `document`-Objekt | kein `document`-Objekt, kein DOM (bis auf jsdom) |
| Sicherheit via Sandbox:<br>z.B. Kein Zugriff auf lokales Filesystem<br>Kein Zugriff auf Betriebssystem | ohne Sicherheitsbeschränkungen,<br>z.B. voller Zugriff auf lokales Filesystem und Betriebssystem |
| Man entwickelt für eine Menge von Browsern. | Man wählt sich eine passende Node.js-Version. |

http://stackoverflow.com/questions/23959868/differences-between-node-environment-and-browser-javascript-environment

# Browser vs. Node.js

## Object.keys(window) vs. Object.keys( global )

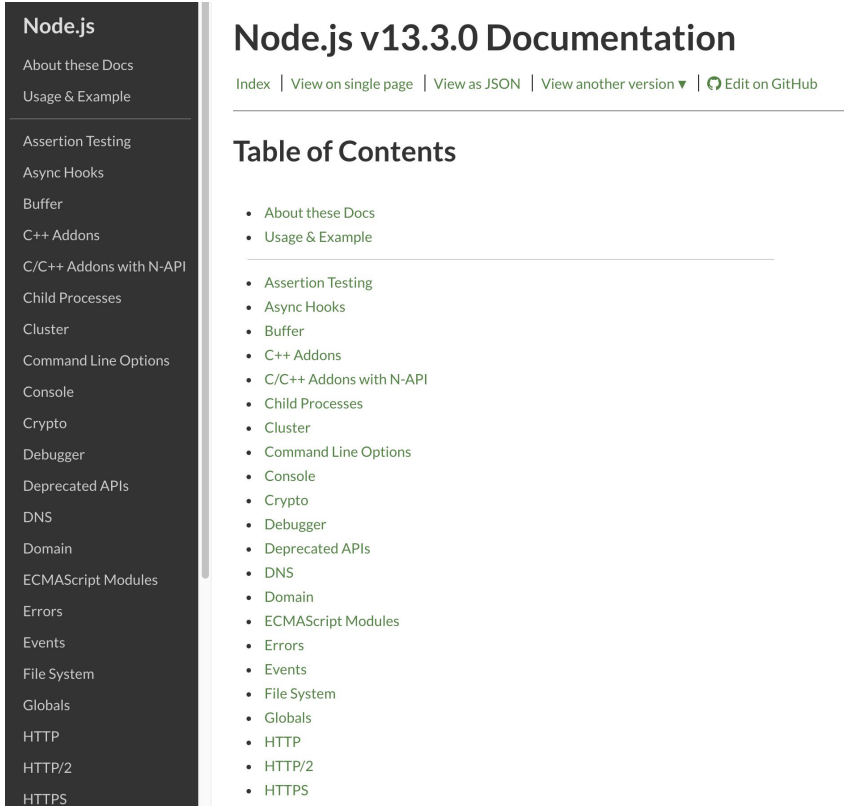
["InstallTrigger", "alert", "applicationCache", "atob", "blur", "btoa", "caches", "cancelAnimationFrame",
"captureEvents", "clearInterval", "clearTimeout", "close", "closed", "confirm", "content",
"createImageBitmap", "crypto", "customElements", "devicePixelRatio", "document", "dump", "external",
"fetch", "find", "focus", "frameElement", "frames", "fullScreen", "getComputedStyle",
"getDefaultComputedStyle", "getSelection", "history", "indexedDB", "innerHeight", "innerWidth",
"isSecureContext", "length", "localStorage", "location", "locationbar", "matchMedia", "menubar", "moveBy",
"moveTo", "mozInnerScreenX", "mozInnerScreenY", "mozPaintCount", "name", "navigator", "onabort",
"onabsolutedeviceorientation", "onafterprint", "onanimationend", "onanimationiteration", "onanimationstart",
"onauxclick", "onbeforeprint", "onbeforeunload", "onblur", "oncanplay", "oncanplaythrough", "onchange",
"onclick", "onclose", "oncontextmenu", "ondblclick", "ondevicelight", "ondevicemotion",
"ondeviceorientation", "ondeviceproximity", "ondrag", "ondragend", "ondragenter", "ondragexit",
"ondragleave", "ondragover", "ondragstart", "ondrop", "ondurationchange", "onemptied", "onended",
"onerror", "onfocus", "onhashchange", "oninput", "oninvalid", "onkeydown", "onkeypress", "onkeyup",
"onlanguagechange", "onload", "onloadeddata", "onloadedmetadata", "onloadend", "onloadstart",
"onmessage", "onmousedown", "onmouseenter", "onmouseleave", "onmousemove", "onmouseout",
"onmouseover", "onmouseup", "onmozfullscreenchange", "onmozfullscreenerror", "onoffline", "ononline",
"onpagehide", "onpageshow", "onpause", "onplay", "onplaying", "onpopstate", "onprogress",
"onratechange", "onreset", "onresize", "onscroll", "onseeked", "onseeking", "onselect", "onselectstart",
"onshow", "onstalled", "onstorage", "onsubmit", "onsuspend", "ontimeupdate", "ontoggle",
"ontransitioncancel", "ontransitionend", "ontransitionrun", "ontransitionstart", "onunload",
"onuserproximity", "onvolumechange", "onwaiting", "onwebkitanimationend", "onwebkitanimationiteration",
"onwebkitanimationstart", "onwebkittransitionend", "onwheel", "open", "opener", "outerHeight",
"outerWidth", "pageXOffset", "pageYOffset", "parent", "performance", "personalbar", "postMessage", "print",
"prompt", "releaseEvents", "requestAnimationFrame", "resizeBy", "resizeTo", "screen", "screenX",
"screenY", "scroll", "scrollBy", "scrollByLines", "scrollByPages", "scrollMaxX", "scrollMaxY", "scrollTo",
"scrollX", "scrollY", "scrollbars", "self", "sessionStorage", "setInterval", "setResizable", "setTimeout",
"showModalDialog", "sidebar", "sizeToContent", "speechSynthesis", "status", "statusbar", "stop", "toolbar",
"top", "updateCommands", "window"]



[
'global',
'process',
'Buffer',
'clearImmediate',
'clearInterval',
'clearTimeout',
'setImmediate',
'setInterval',
'setTimeout',
'console',
'module',
'require' ]

# Node Core Modules

- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Command Line Options
- Console
- Crypto
- Debugger
- Deprecated APIs
- DNS
- Domain
- Errors
- Events
- File System
- Globals
- HTTP
- HTTPS
- Modules
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- Readline
- REPL
- Stream
- String Decoder
- Timers
- TLS/SSL
- Tracing
- TTY
- UDP/Datagram
- URL
- Utilities
- V8
- VM
- ZLIB




https://nodejs.org/api/

# Cluster

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`Master ${process.pid} is running`);

  // Fork workers.
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }

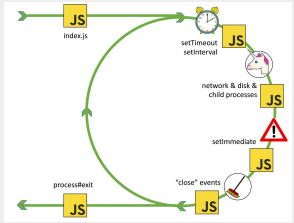
  cluster.on('exit', (worker, code, signal) => {
    console.log(`worker ${worker.process.pid} died`);
  });
}
```

https://nodejs.org/dist/latest-v11.x/docs/api/cluster.html

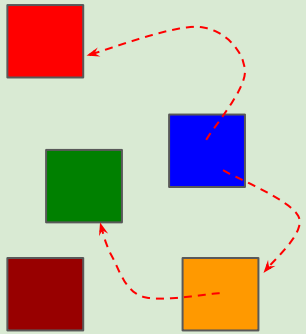
# JavaScript Execution is single-process single-threaded

## V8-Runtime

### Event-Loop

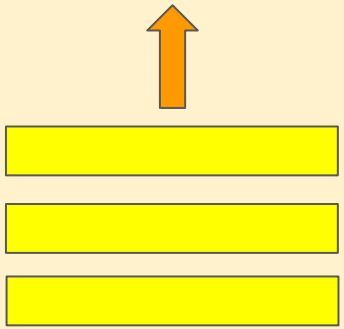



### Heap



Memory Allocation
Garbage Collection

### Stack



Execution Context

single-threaded:
only one stack
Do one thing at a time.
No concurrency.

Google V8

- seit 2006
- High performance JavaScript Engine
- in C++ geschrieben
- inkrementeller GC
- Just-in-time compilation
- wie im Chrome-Browser
- in Node.js
- Open Source
- Basis-Klassen JavaScript

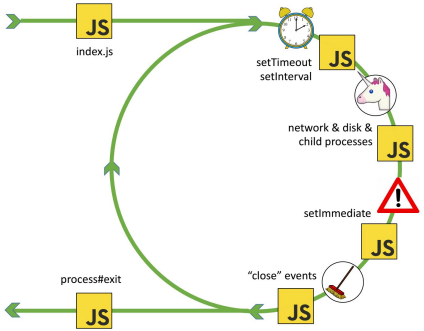
# Fast alle Beschreibungen der Event-Loop sind falsch


Bert Belder, IBM

https://youtu.be/PNa9OMajw9w?t=59s

# Die Node.js - Lösung des Komplexitätsproblems paralleler Prozesse

- Programmierung paralleler Prozesse oder Threads ist schwer und sollte Experten vorbehalten sein.
- Der Anwendungsprogrammierer sollte strikt seriell programmieren.
- Aber: Die Anwendungsprogramme sollten keine Ressourcen blockieren oder die Zeit mit Warten zu verbringen.
- Node.js - Lösung: **Callbacks**, die als Closures in Warteschlangen gelegt werden.
- JavaScript wird **immer seriell** abgearbeitet. **Keine Callbacks laufen parallel.** Jedoch können immer wieder neue Callbacks an das Ende von Warteschlangen gesetzt werden, die in der nächsten Runde der Event Loop abgearbeitet werden.




Event-Loop in Node.js

Alle Nebenläufigkeits-probleme werden in der Node.js Runtime gelöst

| Node.js Laufzeitumgebung | Anwendung in JavaScript |
|---|---|
| parallel | seriell |
| multi-threaded | single-threaded |
| Warteschlangen | Callbacks, Promise, async |

Hier gibt es keine Nebenläufigkeit



https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

# Wie schreibt man asynchrone Funktionen in ES8?

1. async **function**
2. Promise
3. setTimeout
4. setInterval
5. setImmediate
6. process.nextTick()
7. durch Aufruf einer asynchronen Funktion aus dem Node.js Kernel

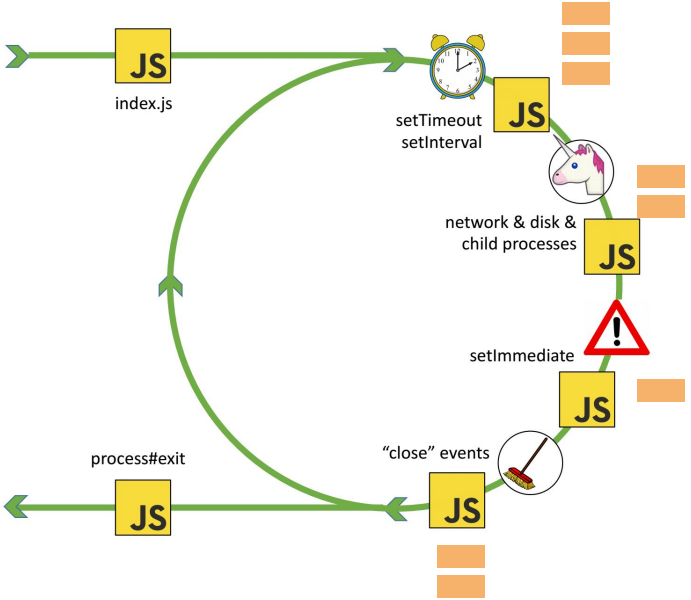
```
const fs = require('fs');

function readFilePromisified( filename ) {
 return new Promise(
   function ( resolve, reject ) {
     fs.readFile( filename, { encoding: 'utf8' },
       ( error, data ) => {
         if (error) {
           reject( error );
         } else {
           resolve( data );
         }
       });
   });
}
```

```
readFilePromisified("file1.txt")
 .then(text => {
   console.log(text);
 })
 .catch(error => {
   console.log(error);
 });
```

# Funktionen, Events und Queues in der Event-Loop

| Funktion | Events & Queues |
|---|---|
| setTimeout()<br>setInterval() | Timer Event Queue |
| fs.readFile() etc.<br>alle asynchronen C++<br>-Funktionen aus dem<br>Node.js-Kernel | Queue for Network, Disk & OS Events |
| setImmediate() | Immediate Queue |
| Promise.resolve() | Promise Microtask Queue |
| process.nextTick() | nextTick Microtask Queue |



# Asynchrone C++ -Funktionen aus dem Node.js-Kernel

## Kernel

tcp / udp sockets, servers

unix domain sockets, servers

pipes

tty input

dns.resolveXXXX

## Thread Pool

files

fs.*

dns.lookup

pipes (exceptional)

Userland
single-threaded

Runtime
multi-threaded

## Signal Handler

(posix only)

child processes

signals

## Wait Thread

(windows only)

child processes

console input

tcp servers (exceptional)

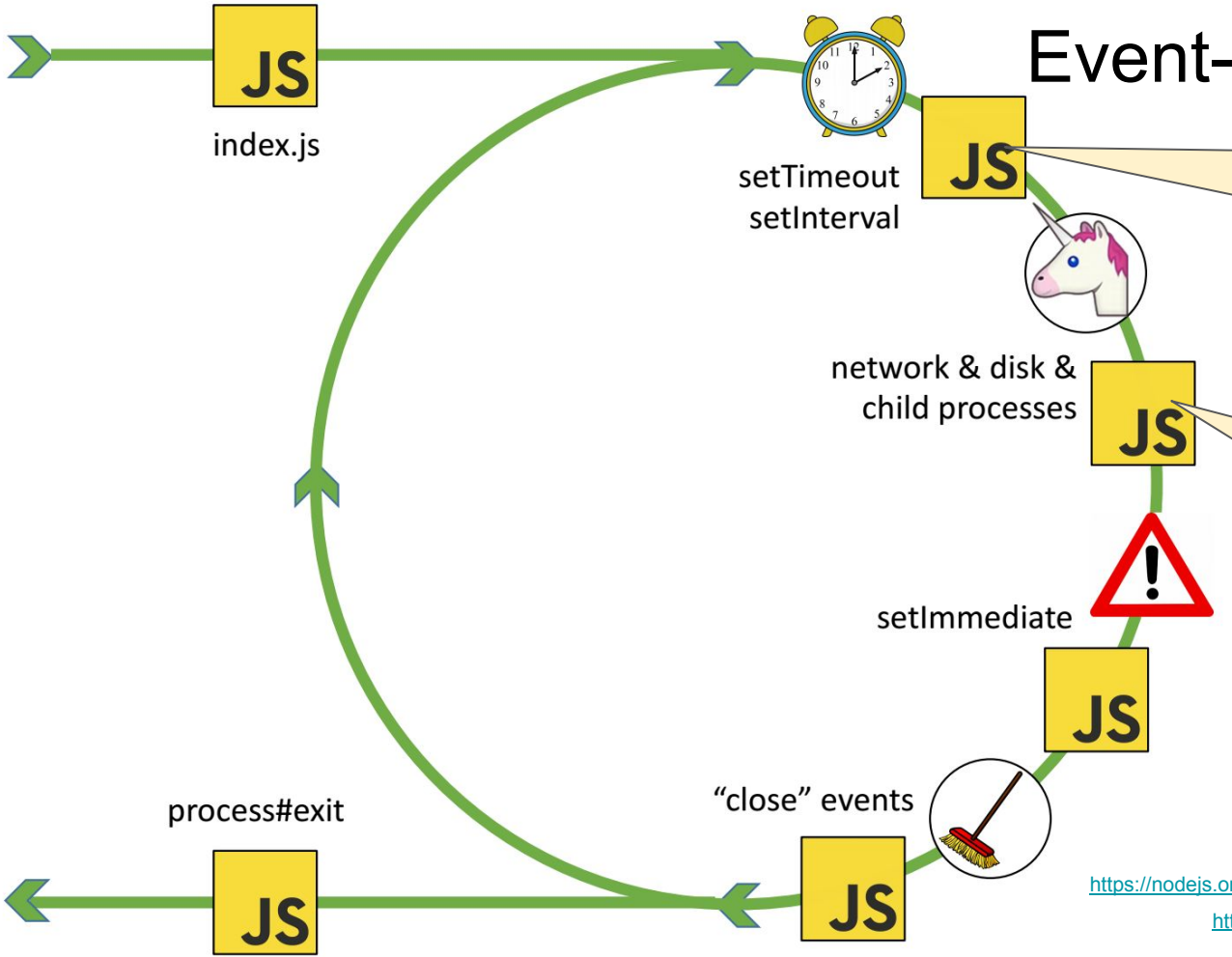
**Node.js 0.1**

- 20% JavaScript
- 80% C / C++

https://www.youtube.com/watch?v=PNa9OMajw9w

https://drive.google.com/file/d/0B1ENiZwmJ_J2a09DUmZROV9oSGc/view

# Event-Loop in Node.js




Arbeite hier alle Callbacks in der Warteschlange ab, die mittels setTimeout oder setInterval irgendwann erzeugt wurden.

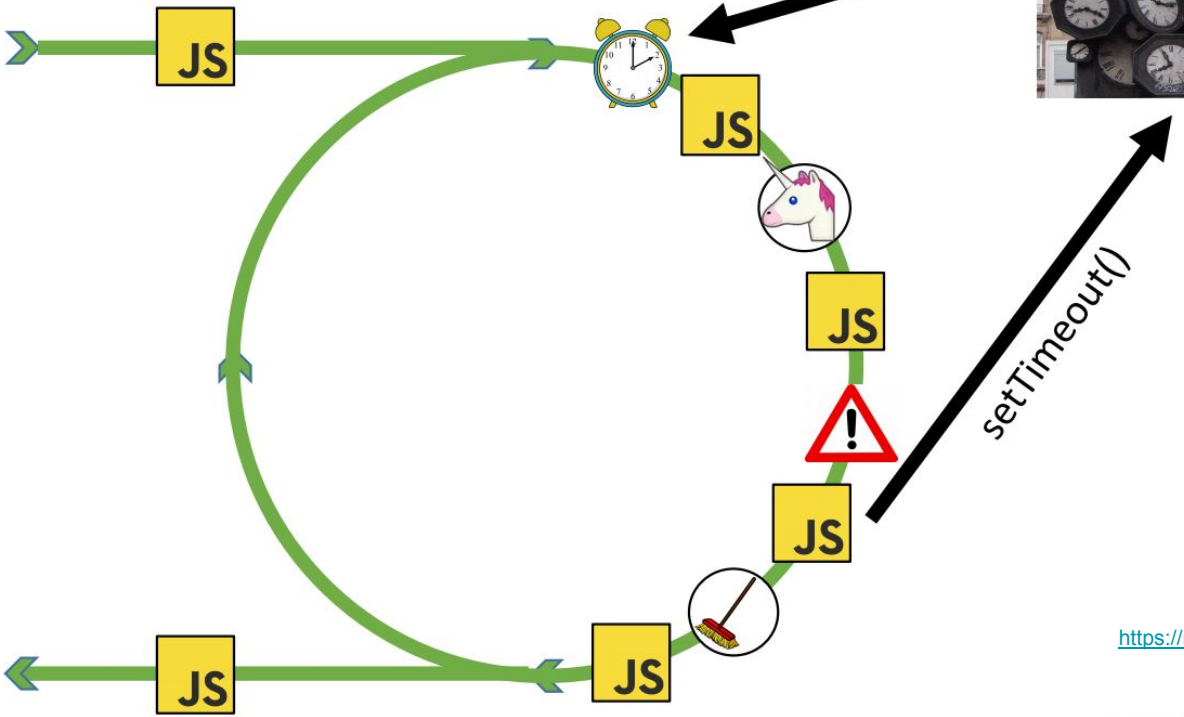
Arbeite hier alle Callbacks in der Warteschlange ab, die OS-seitig von network, disk oder OS-Kinderprozessen irgendwann erzeugt wurden.

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

https://www.youtube.com/watch?v=PNa9OMajw9w

https://drive.google.com/file/d/0B1ENiZwmJ_J2a09DUmZROV9oSGc/view

# Beispiel setTimeout()




*"Warteschlange der Callbacks"*

**2.** Warteschlange der Closures, die mit **setTimeout()** erzeugt wurden, wird in der nächsten Runde abgearbeitet.

| cb1 | cb2 | cb3 | cb4 |
|---|---|---|---|

**1.** Mit **setTimeout(cb4)** wird **cb4** ans Ende der Warteschlange gesetzt. Erst im nächsten Durchlauf der Event Loop wird **cb4** abgearbeitet.

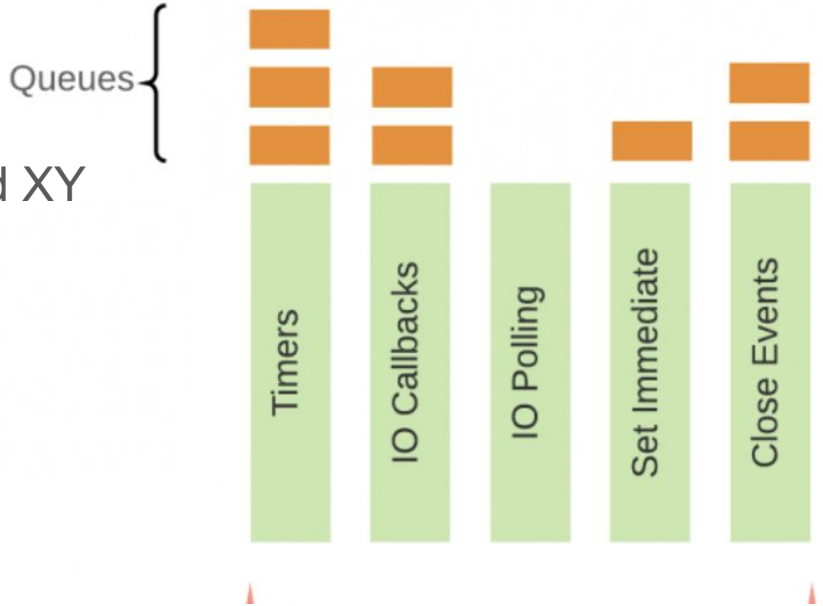
https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

https://www.youtube.com/watch?v=PNa9OMajw9w

https://drive.google.com/file/d/0B1ENiZwmJ_J2a09DUmZROV9oSGc/view

Aus der Node.js Dokumentation:

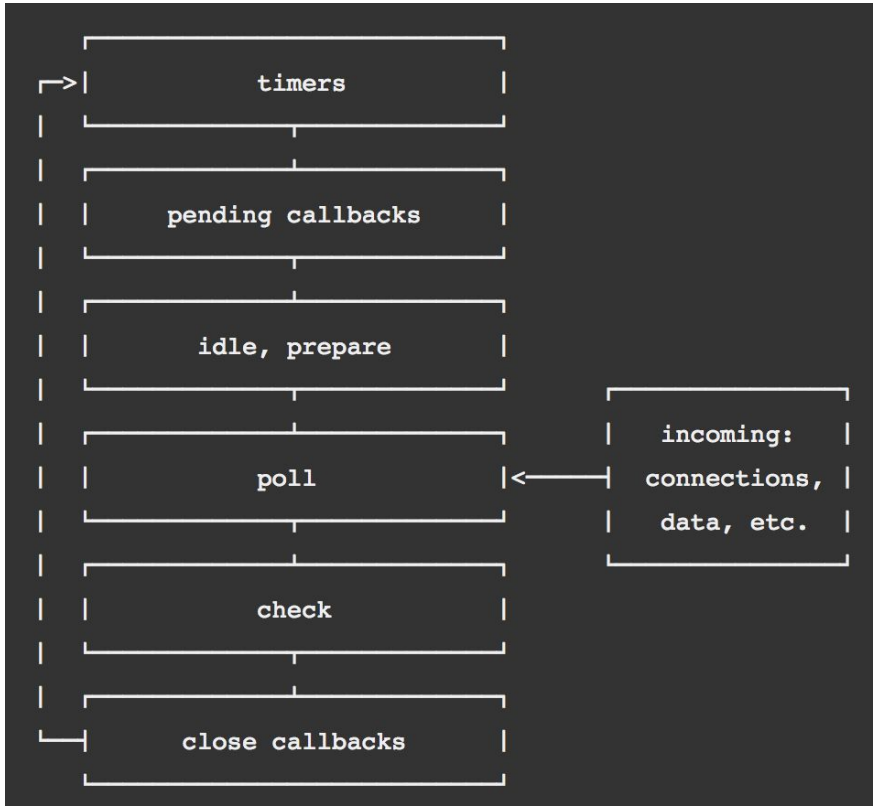
# The Node.js Event Loop

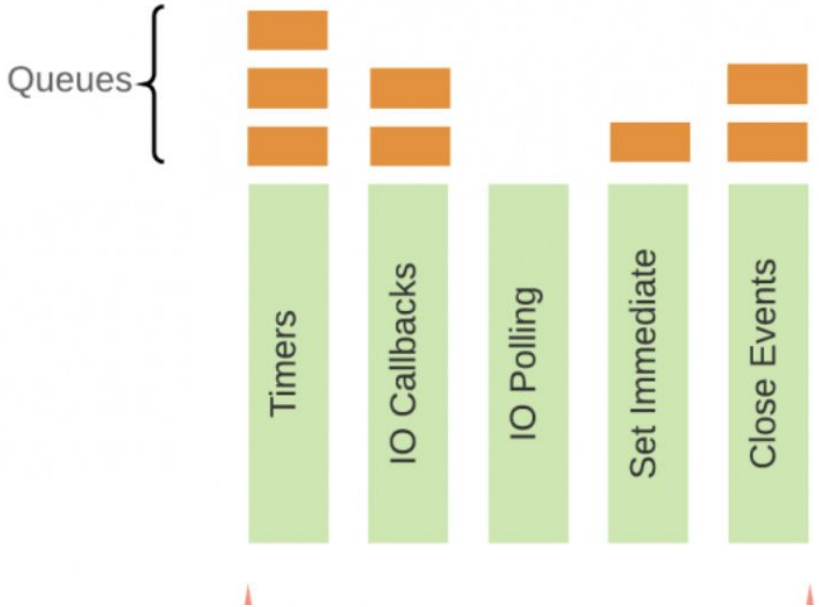



- Node.js = 1 Process 1 Thread + Evented XY
  - Evented Timers
    - setTimeout() or setInterval()
  - Evented Callbacks
    - Userland Callbacks
  - Evented Polling
    - collect callbacks for the next run
  - Evented Immediates
    - callbacks registered by setImmediate()
  - Evented Close Actions
    - Here all on('close') event callbacks are processed.

https://www.dynatrace.com/news/blog/all-you-need-to-know-to-really-understand-the-node-js-event-loop-and-its-metrics/

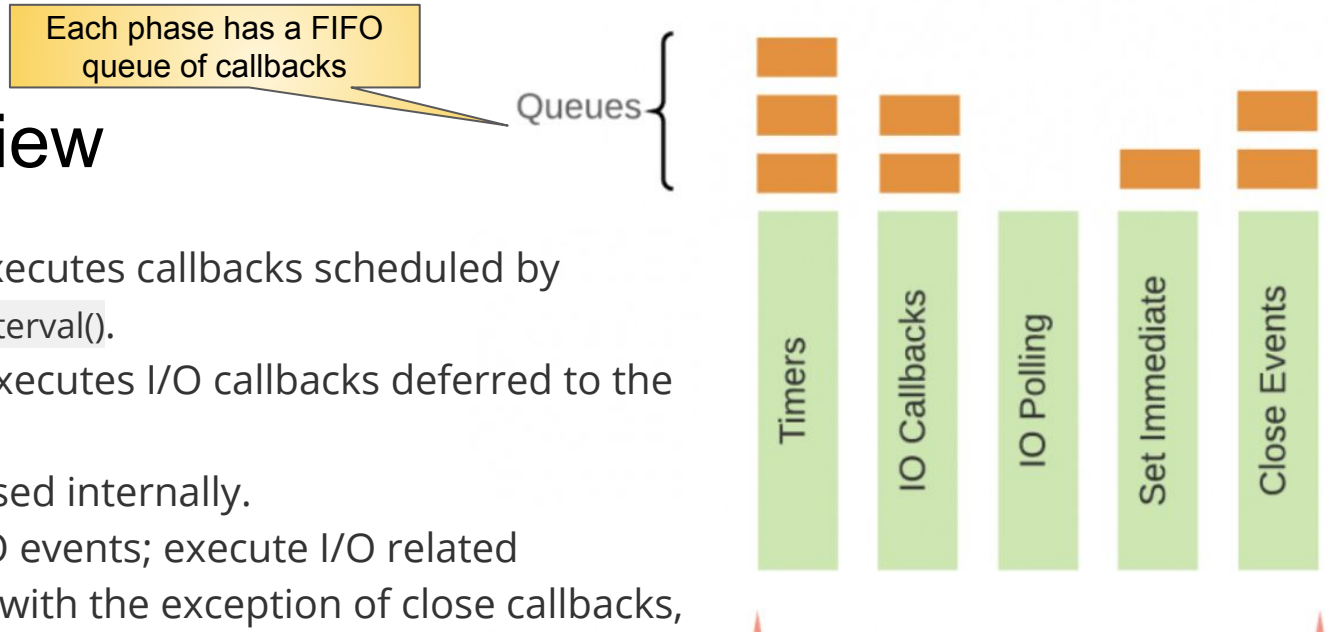
# The Node.js Event Loop




https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

# Event-Loop Phases Overview

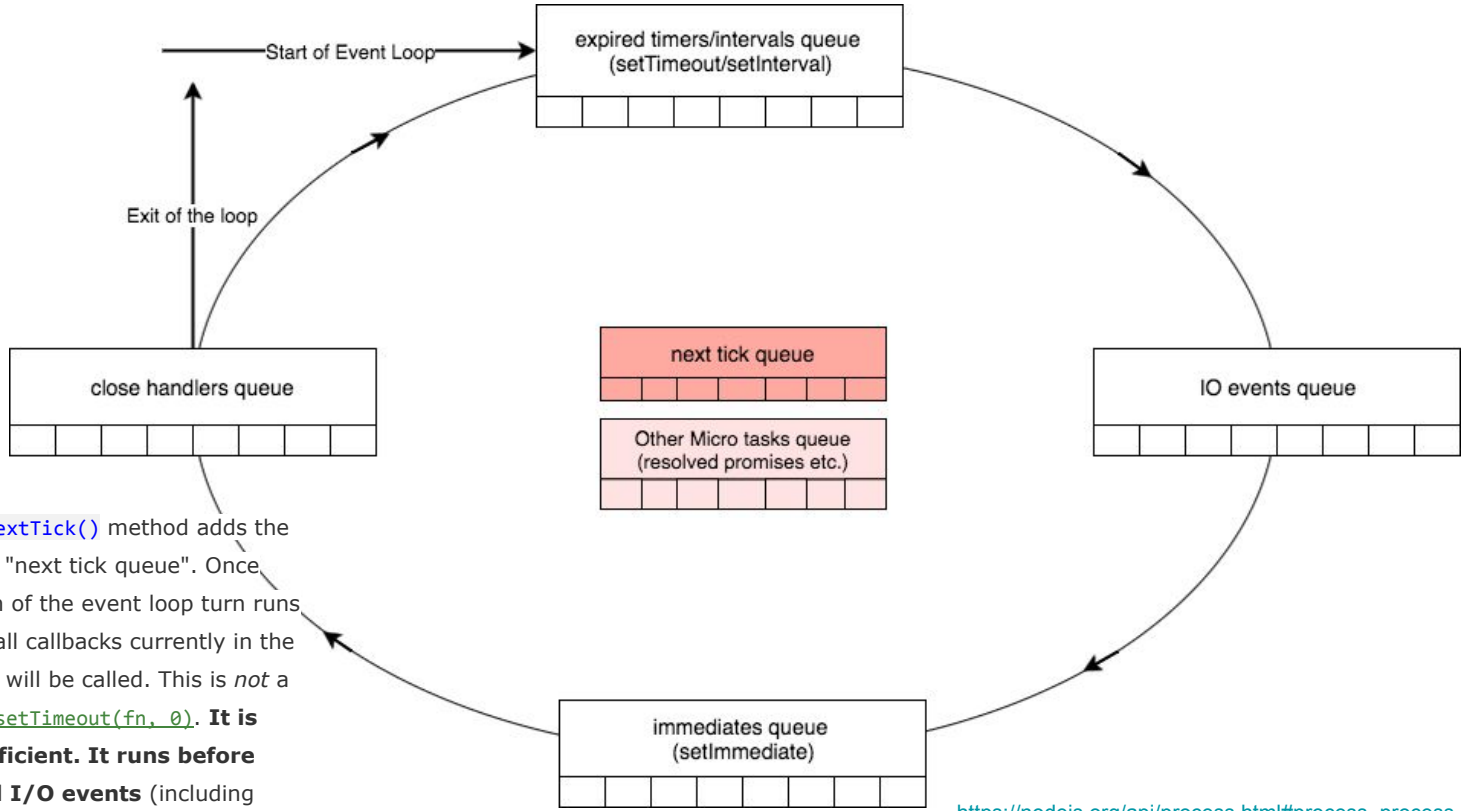



- timers: this phase executes callbacks scheduled by `setTimeout()` and `setInterval()`.
- pending callbacks: executes I/O callbacks deferred to the next loop iteration.
- idle, prepare: only used internally.
- poll: retrieve new I/O events; execute I/O related callbacks (almost all with the exception of close callbacks, the ones scheduled by timers, and `setImmediate()`); node will block here when appropriate.
- check: `setImmediate()` callbacks are invoked here.
- close callbacks: e.g. `socket.on('close', ...)`.

Between each run of the event loop, Node.js checks if it is waiting for any asynchronous I/O or timers and shuts down cleanly if there are not any.

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

# Für ganz Eilige: process.nextTick() ⇒ next tick queue




"The `process.nextTick()` method adds the `callback` to the "next tick queue". Once the current turn of the event loop turn runs to completion, all callbacks currently in the next tick queue will be called. This is *not* a simple alias to `setTimeout(fn, 0)`. **It is much more efficient. It runs before any additional I/O events** (including timers) fire in subsequent ticks of the event loop."

https://nodejs.org/api/process.html#process_process_nexttick_callback_args

https://jsblog.insideattack.net/promises-next-ticks-and-immediates-nodejs-event-loop-part-3-9226cbe7a6aa

# Beispiel mit Promise und nextTick

```
1  Promise.resolve().then(() => console.log('promise1 resolved'));
2  Promise.resolve().then(() => console.log('promise2 resolved'));
3  Promise.resolve().then(() => {
4     console.log('promise3 resolved');
5     process.nextTick(() => console.log('next tick inside promise resolve handler'));
6  });
7  Promise.resolve().then(() => console.log('promise4 resolved'));
8  Promise.resolve().then(() => console.log('promise5 resolved'));
9  setImmediate(() => console.log('set immediate1'));
10 setImmediate(() => console.log('set immediate2'));
11 process.nextTick(() => console.log('next tick1'));
12 process.nextTick(() => console.log('next tick2'));
13 process.nextTick(() => console.log('next tick3'));
14 setTimeout(() => console.log('set timeout'), 0);
15 setImmediate(() => console.log('set immediate3'));
16 setImmediate(() => console.log('set immediate4'));
```

next tick1
next tick2
next tick3
promise1 resolved
promise2 resolved
promise3 resolved
promise4 resolved
promise5 resolved
next tick inside promise resolve handler
set timeout
set immediate1
set immediate2
set immediate3
set immediate4

https://jsblog.insiderattack.net/promises-next-ticks-and-immediates-nodejs-event-loop-part-3-9226cbe7a6aa

# Gliederung



1. Node.js
2. npm
3. Express.js
4. MongoDB
5. Ausblick

# npm - Node Package Manager

## What is npm?

Getting started

- 01 - What is npm?
- 02 - How to find & select packages
- 03 - How to set up a new npm account & install npm
- 04 - How to install local

By the numbers

Packages
1.160.831

Downloads · Last Week
14.743.770.891

Downloads · Last Month
62.619.363.682

npm opens up an entire world of JavaScript talent for you and your team. It's the world's largest software registry, with approximately 3 billion downloads per week. The registry contains over 600,000 *packages* (building blocks of code). Open-source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.

Here is a quick introduction to npm:

### Microsoft Windows:

```
C:\ npm install lodash
C:\ dir node_modules

#=> lodash
```

### macOS, Ubuntu, Debian

```
> npm install lodash
> ls node_modules

#=> lodash
```

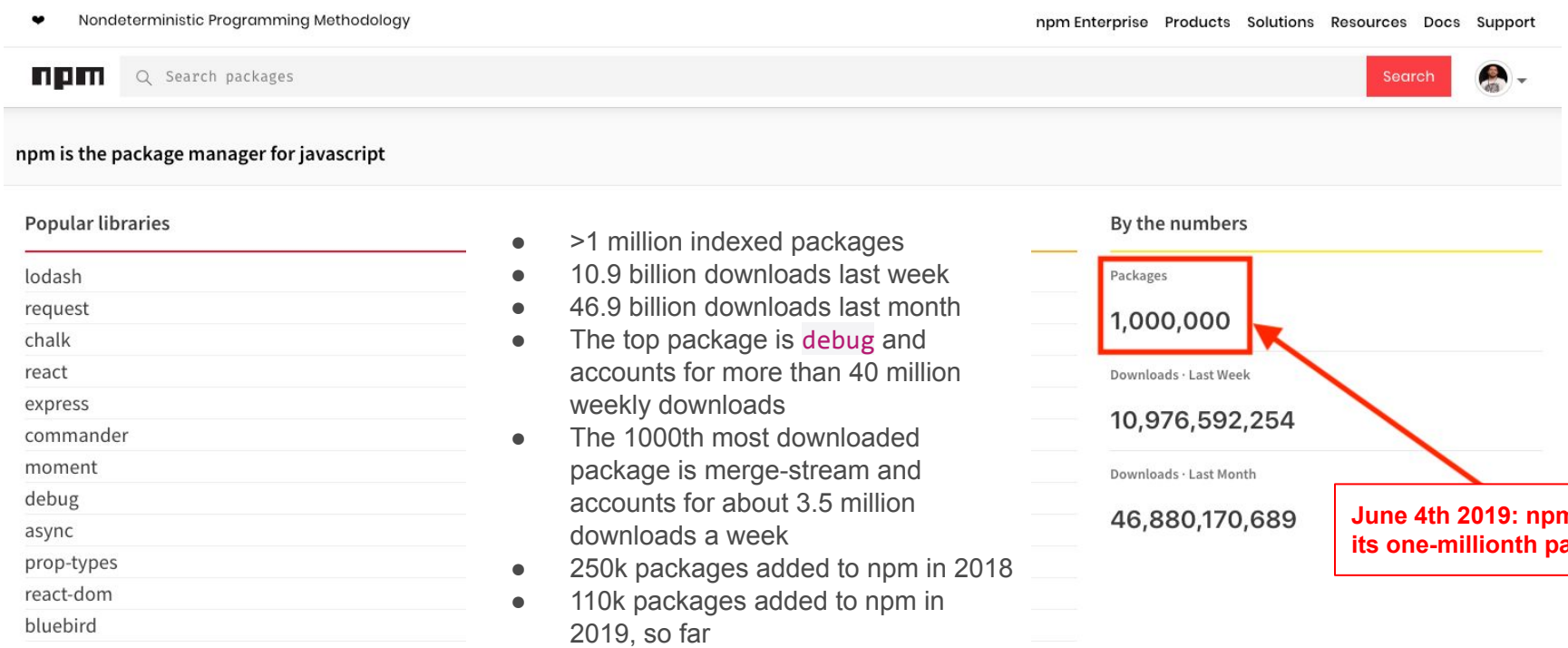
https://docs.npmjs.com/getting-started/what-is-npm

# A brief history of npm

- January **2010**: npm was created by Isaac Z. Schlueter, written in JavaScript.
- March 2016: popular JavaScript package, left-pad, was unpublished from the npm repository due to a naming dispute. After much disruption among the JavaScript world, tightened up its unpublishing policy.
- July 2018: a malicious version of the eslint-scope package was released which exposed npm credentials of users to the attacker.
- November 2018: the malicious package, flatmap-stream, was released to npm which was added as a dependency to the popular event-stream package. The package targeted Copay the bitcoin wallet.
- June 4th **2019**: npm indexes its **one-millionth package**.

https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/

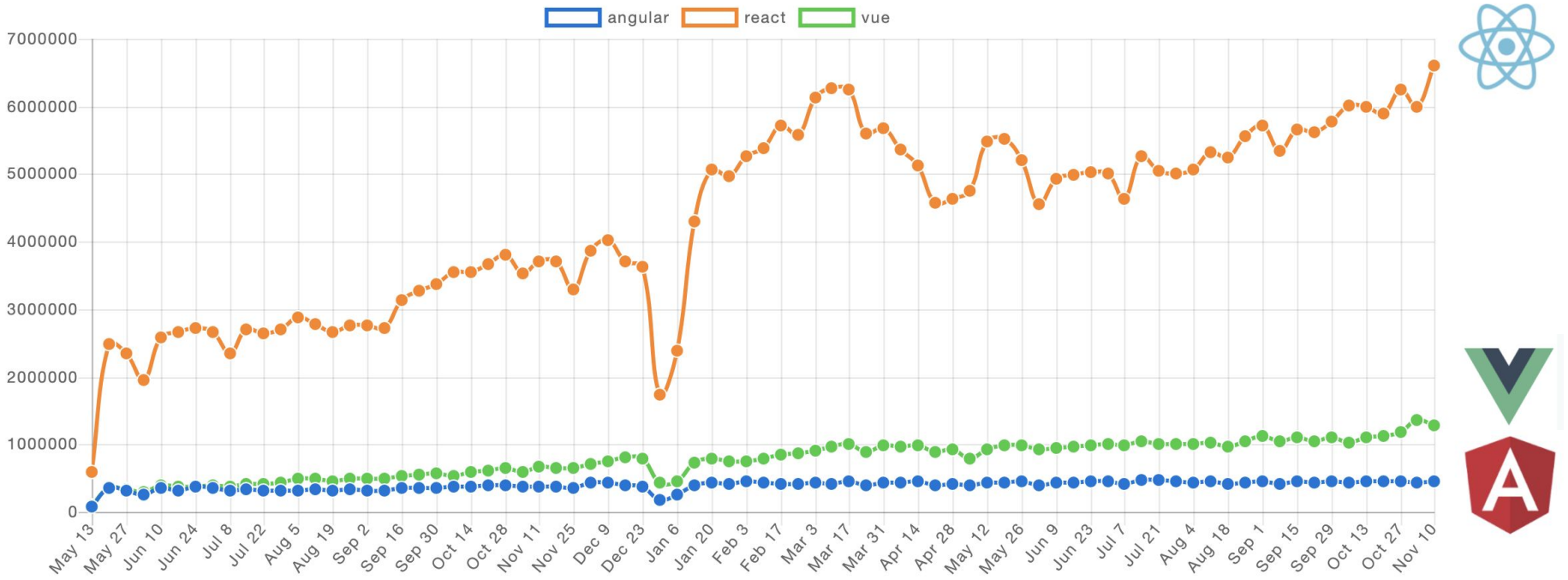
# npm package downloads




- >1 million indexed packages
- 10.9 billion downloads last week
- 46.9 billion downloads last month
- The top package is `debug` and accounts for more than 40 million weekly downloads
- The 1000th most downloaded package is merge-stream and accounts for about 3.5 million downloads a week
- 250k packages added to npm in 2018
- 110k packages added to npm in 2019, so far

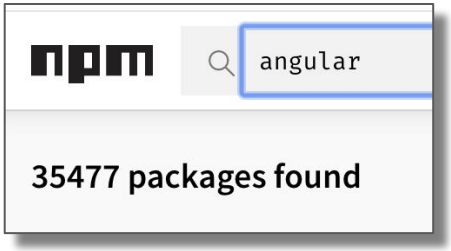
https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/

# npm trends




https://www.npmtrends.com/angular-vs-react-vs-vue

# npm Anzahl der npm-Pakete



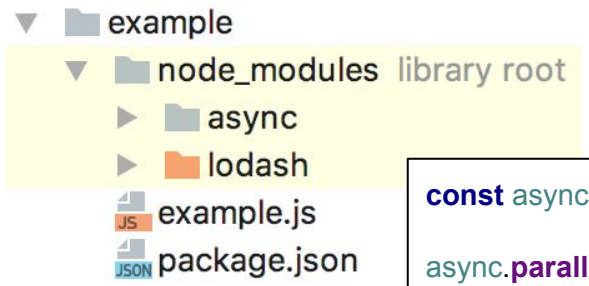

| Framework | Anzahl npm Pakete |
|---|---|
| React | 104.297 |
| Angular | 35.477 |
| Vue | 28.932 |
| jQuery | 8.506 |
| ember | 7.003 |

https://www.npmjs.com/search?q=angular

# Package Install

## Local Install:

- ➔ mkdir example
- ➔ cd example
- ➔ npm init
- ➔ npm install async --save




```
const async = require('async');

async.parallel([
 function() {},
 function() {}
]);
```

## Global Install

npm install <package_name> -g

z.B.

npm install npm@latest -g

alle Command Line Interfaces (cli) mit -g

# Eigenes Package anlegen:

```
mkaul2m(/tmp)$ mkdir example1
mkaul2m(/tmp)$ cd example1/
mkaul2m(/tmp/example1)$ npm init
name: (example1)
version: (1.0.0)
description: example1
entry point: (index.js)
test command: test
git repository:
keywords: test
author: Kaul
license: (ISC) MIT
```

About to write to /private/tmp/example1/**package.json**:

```
{
  "name": "example1",
  "version": "1.0.0",
  "description": "example1",
  "main": "index.js",
  "scripts": {
    "test": "test"
  },
  "keywords": [
    "test"
  ],
  "author": "Kaul",
  "license": "MIT"
}
```

Is this ok? (yes) yes

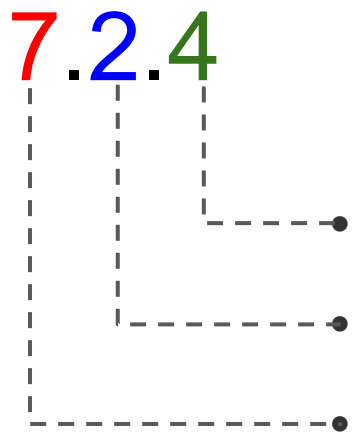
```
$ npm install async --save
example1@1.0.0
/private/tmp/example1
└─┬ async@2.4.0
  └── lodash@4.17.4
```

```
"dependencies": {
   "async": "^2.4.0"
 }
```

Packages werden ohne das **node_modules** - Unterverzeichnis archiviert und gespeichert, da man mit **npm install** dieses Unterverzeichnis wieder schnell herstellen kann.

# SemVer Spec

Semantic versioning




**Bug fixes** and other minor changes:
**Patch release**, increment the last number, e.g. 7.2.5
**New features** which don't break existing features:
**Minor release**, increment the middle number, e.g. 7.3.0
Changes which **break backwards compatibility**:
**Major release**, increment the first number, e.g. 8.0.0

Ranges:
^7.2.4 = Caret Range
~7.2.4 = Tilde Range
https://docs.npmjs.com/misc/semver

https://docs.npmjs.com/getting-started/semantic-versioning

# SemVer Ranges

- ^7.2.4 = Caret Range

  Allows changes that **do not modify the left-most non-zero digit** in the [major, minor, patch] tuple. In other words, this allows patch and minor updates for versions 1.0.0 and above, patch updates for versions 0.X >=0.1.0, and no updates for versions 0.0.X.

- ~7.2.4 = Tilde Range

  Allows patch-level changes if a minor version is specified on the comparator. Allows minor-level changes if not.

  **~1.2.3** := **>=1.2.3 <1.(2+1).0** := **>=1.2.3 <1.3.0**

  **~1.2** := **>=1.2.0 <1.(2+1).0** := **>=1.2.0 <1.3.0** (Same as **1.2.x**)

  **~1** := **>=1.0.0 <(1+1).0.0** := **>=1.0.0 <2.0.0** (Same as **1.x**)

  **~0.2.3** := **>=0.2.3 <0.(2+1).0** := **>=0.2.3 <0.3.0**

  **~0.2** := **>=0.2.0 <0.(2+1).0** := **>=0.2.0 <0.3.0** (Same as **0.2.x**)

  **~0** := **>=0.0.0 <(0+1).0.0** := **>=0.0.0 <1.0.0** (Same as **0.x**)

https://docs.npmjs.com/misc/semver

# Das Modul-Konzept in npm: Common.js http://requirejs.org/docs/commonjs.html

Modul erstellen:

```
// add.js
function add (a, b) {
  return a + b
}

module.exports = add
```

JavaScript hatte kein eigenes Modul-Konzept.
Common.js hat lange vor ES6 diese Lücke gefüllt.

To use the `add` module we have just cre

Modul importieren:

```
// index.js
const add = require('./add')

console.log(add(4, 5))
//9
```

Under the hood, `add.js` is wrapped by Node.js this way:

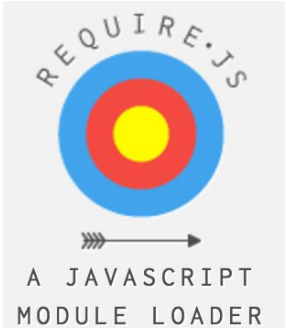
```
(function (exports, require, module, __filename, __dirname) {
  function add (a, b) {
    return a + b
  }

  module.exports = add
})
```

https://blog.risingstack.com/node-js-at-scale-module-system-commonjs-require/

# Module in Node.js

## CommonJS




http://requirejs.org/docs/commonjs.html

- Modul-Konzept fehlte in JavaScript
- Jede Datei ist ein Modul
- Jedes Modul soll seinen eigenen Namensraum haben
  - durch Kapselung jeden Moduls in einer Funktion
- Explizite Exportierung derjenigen Objekte, die außen sichtbar sein sollen
  - module.exports = ...
- Der CommonJS Module Loader wird in Node.js eingesetzt:
  - require()
  - module
- Beispiel:


```
// hello.js

const Hello = "Hello";
module.exports = Hello;
```

```
// world.js

const Hello = require('./hello');
const greeting = Hello + " World!";
console.log( greeting );
```

https://nodejs.org/api/modules.html

# require: Common.JS - Laderegeln in Node.js

- require('**X**') ⇒ If X.js is a file, load X.js as Common.JS module text.
- require('**X.js**') ⇒ If X.js is a file, load X.js as Common.JS module text.
- require('**X.json**') ⇒ If X.json is a file, parse X.json to a JavaScript Object.
  - d.h. JSON.parse( Inhalt von X.json ) **wird automatisch ausgeführt**
- require('**X.node**') ⇒ If X.node is a file, load X.node as binary addon.

Der Node.js - Suchpfad ist konfigurierbar,
typischerweise Unterverzeichnis node_modules

https://nodejs.org/api/modules.html

In order to make modules available to the Node.js REPL, it might be useful to also add the `/usr/lib/node_modules` folder to the `$NODE_PATH` environment variable. Since the module lookups using `node_modules` folders are all relative, and based on the real path of the files making the calls to `require()`, the packages themselves can be anywhere.

# Brücken zwischen 2 Welten: Browser vs. Server

## ES6 import

## CommonJS require

https://www.npmjs.com/package/cjs-to-es6







https://nolanlawson.com/2015/10/19/the-struggles-of-publishing-a-javascript-library/
https://web-ben.de/browserify-vs-webpack-das-drama-der-package-manager/

# Gliederung




1. Node.js
2. npm
3. Express.js
4. MongoDB
5. Ausblick

# npm-Modul Express.js

express express
Fast, unopinionated, minimalist...
4.15.2 published 2 months ago by ...

- express.js ist ein npm-Modul
- Es setzt auf dem http-Modul auf und fügt Routing und viele nützliche Web-Server-Funktionen hinzu
- Routing ist deklarativ

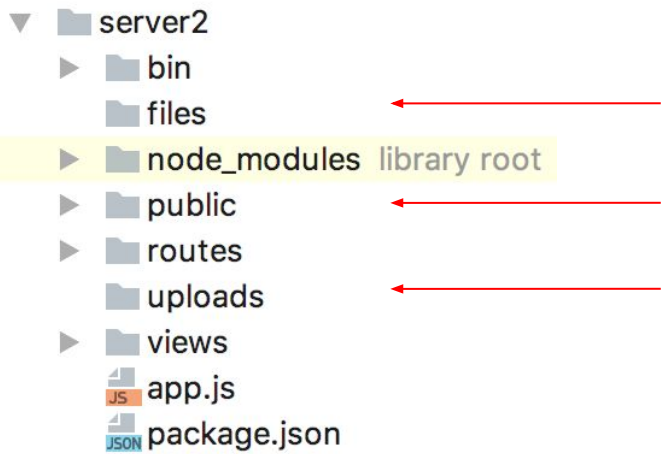

1. Serving static files
2. Dynamische Express-Anwendungen
2.1. Manuelle Installation eines Servers
2.2. Programmierung

npm install express --save

http://expressjs.com/

# 1. Serving static files

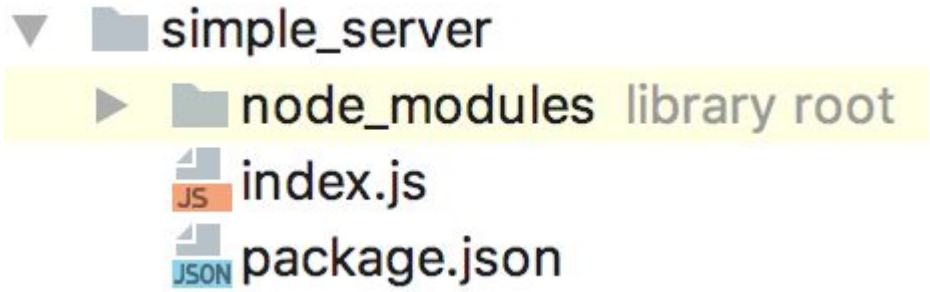
```
const express = require('express');
const app = express();
app.use(express.static(__dirname + '/public'));
app.use(express.static(__dirname + '/files'));
app.use(express.static(__dirname + '/uploads'));
```




static files

# 2. Dynamische Anwendungen mit express.js
# 2.1. Manuelle Installation eines Servers

- mkdir simple_server
- cd simple_server
- npm init
- npm install express --save
- node index
  - starts server




Erste Anwendung von Express.js

```
// index.js
const express = require('express');
const app = express();

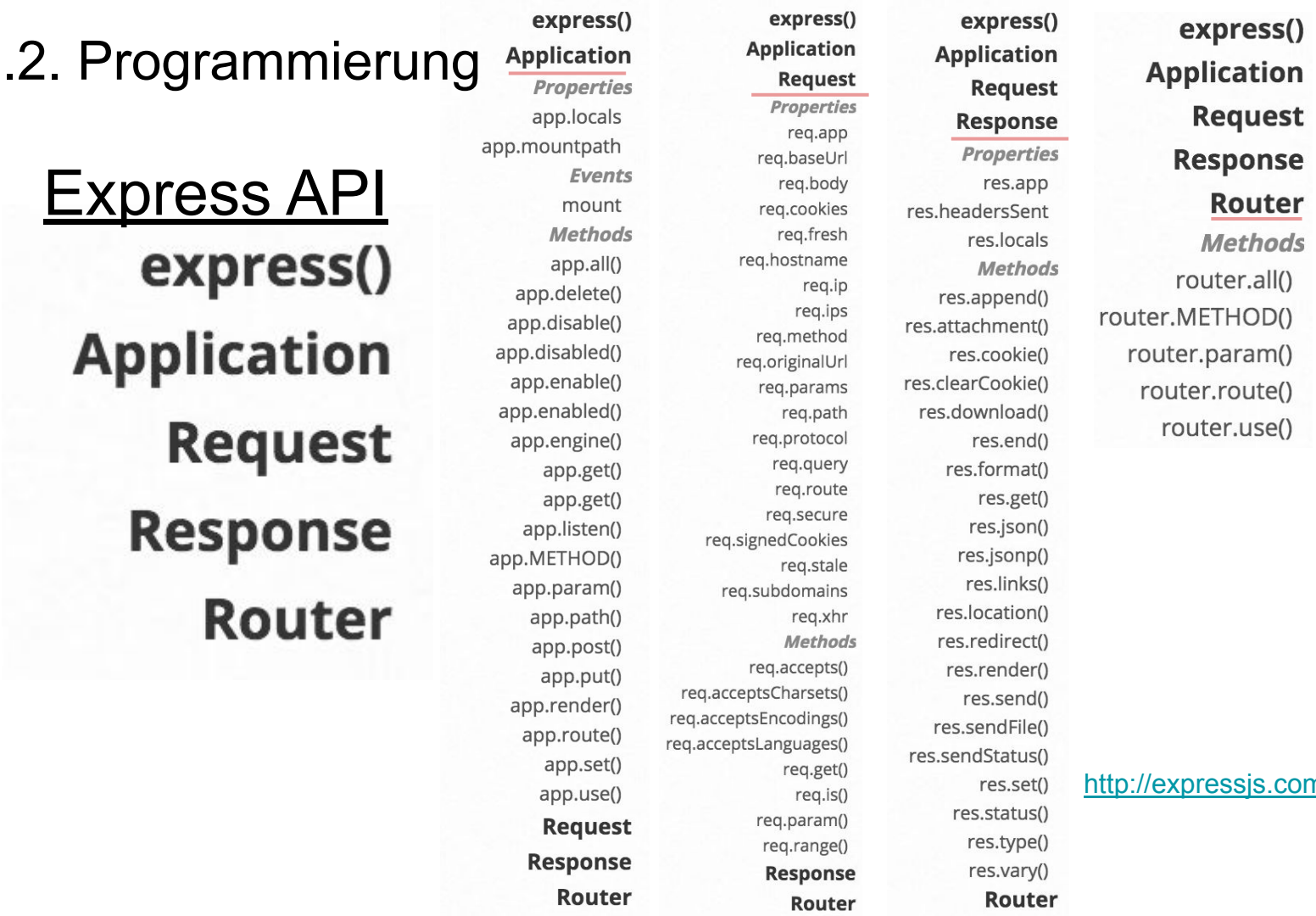
app.get('/', (req, res) => {
 res.writeHead( 200, {"Content-Type": "text/html"} );
 res.end('<h1>Hello World</h1>');
}).listen(3000);

console.log( `Serving at http://localhost:3000` );
```

https://www.npmjs.com/package/express

# 2.2. Programmierung

## Express API




http://expressjs.com/de/4x/api.html

# Express Application = The app - Object

The app object conventionally denotes the Express application. Create it by calling the top-level express() function exported by the Express module:

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
  res.send('hello world');
});

app.listen(3000);
```

The app object has methods for

- Routing HTTP requests; see for example, app.METHOD and app.param.
- Configuring middleware; see app.route.
- Rendering HTML views; see app.render.
- Registering a template engine; see app.engine.

http://expressjs.com/de/4x/api.html#app

# Express Request = req - Parameter

The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on. In this documentation and by convention, the object is always referred to as req (and the HTTP response is res) but its actual name is determined by the parameters to the callback function in which you're working.

For example:

```
app.get('/user/:id', function(req, res) {
  res.send('user ' + req.params.id);
});
```

But you could just as well have:

```
app.get('/user/:id', function(request, response) {
  response.send('user ' + request.params.id);
});
```

http://expressjs.com/de/4x/api.html#req

The req object is an enhanced version of Node's own request object and supports all built-in fields and methods.

# Express Router = Mini-Application unter einer Route

A router object is an isolated instance of middleware and routes. You can think of it as a “mini-application,” capable only of performing middleware and routing functions. Every Express application has a built-in app router.

A router behaves like middleware itself, so you can use it as an argument to app.use() or as the argument to another router’s use() method.

The top-level express object has a Router() method that creates a new router object.

Once you’ve created a router object, you can add middleware and HTTP method routes (such as get, put, post, and so on) to it just like an application. For example:

```
// invoked for any requests passed to this router
router.use(function(req, res, next) {
  // .. some logic here .. like any other middleware
  next();
});

// will handle any request that ends in /events
// depends on where the router is "use()'d"
router.get('/events', function(req, res, next) {
   do_something();  next();
});
```

Ohne **next()** bricht die Bearbeitung ab.
D.h. HTTP response wird nicht gesendet.

You can then use a router for a particular root URL in this way separating your routes into files or even mini-apps.

```
// only requests to /calendar/* will be sent to our "router"
app.use('/calendar', router);
```

http://expressjs.com/de/4x/api.html#router

# Express.js Middleware

- ```
  var express = require('express');
  var app = express();
  var router = express.Router();
  ```
- Einbau von Middleware mit "`use()`"

  ```
  const logger = require('morgan');
  app.use(logger('dev'));
  ```

Initialisierung

- app.use(*express_middleware()*)
  - z.B. **'morgan'** (logger), **'body-parser'** (für das Parsen des Request-Bodys eines Post-Requests)
- router.use(*router_middleware()*)
  - z.B.

```
// simple logger for this router's requests
// all requests to this router will first hit this middleware
router.use(function(req, res, next) {
  console.log('%s %s %s', req.method, req.url, req.path);
  next();
});
```
```

# Express.js Middleware: Beispiel cookie-parser

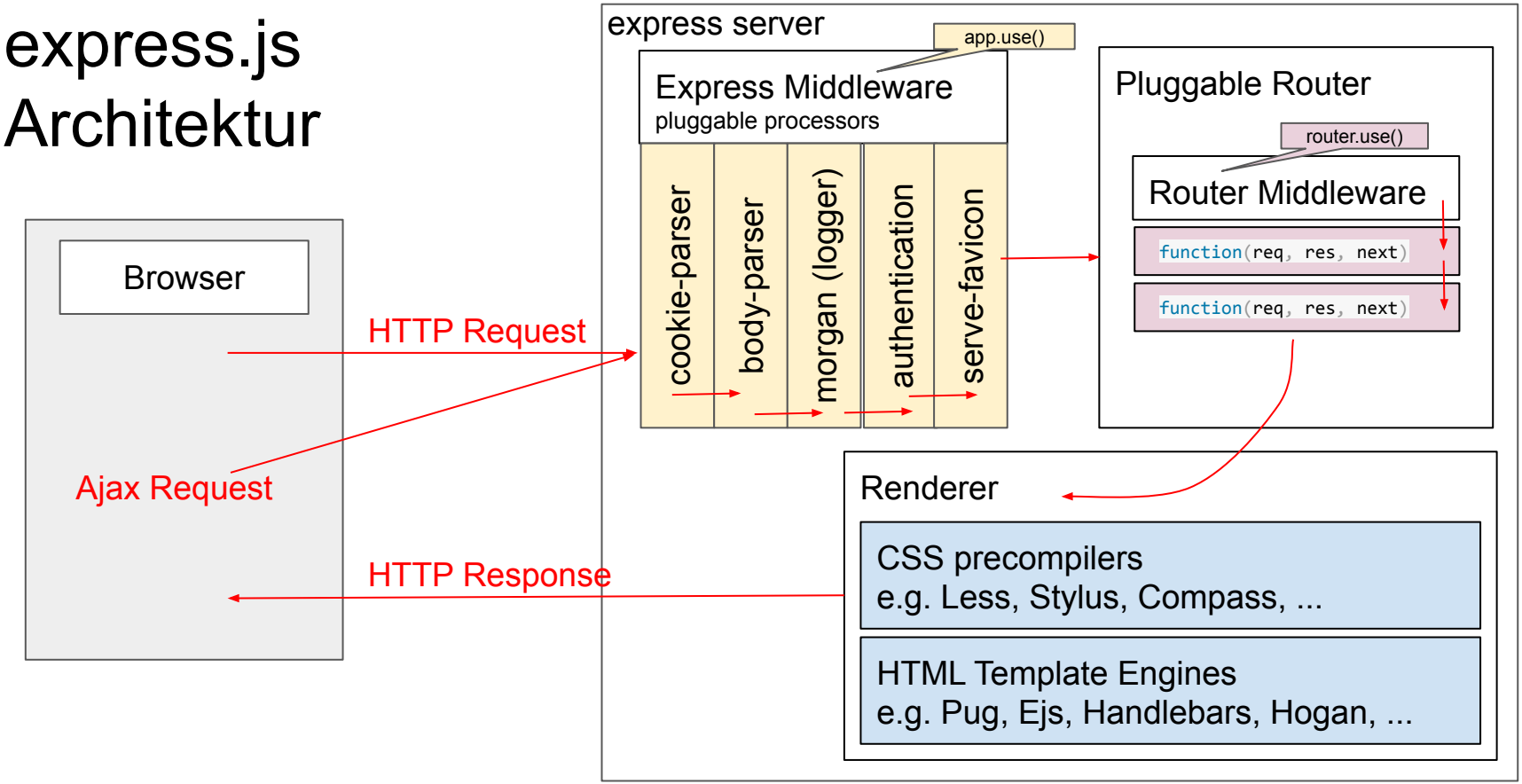
```
$ npm install cookie-parser
```

```
var express = require('express');
var app = express();
var cookieParser = require('cookie-parser');

// load the cookie-parsing middleware
app.use(cookieParser());
```

http://expressjs.com/de/guide/using-middleware.html#middleware.third-party

# express.js Architektur



# Express.js Middleware

## Typische Express 4.0 Middleware

- morgan: logger
- body-parser: parse the body so you can access parameters in requests in `req.body`. e.g. `req.body.name`.
- cookie-parser: parse the cookies so you can access parameters in cookies `req.cookies`. e.g. `req.cookies.name`.
- serve-favicon: exactly that, serve favicon from route `/favicon.ico`. Should be call on the top before any other routing/middleware takes place to avoids unnecessary parsing.
- cors: a node.js package for providing a Connect/Express middleware that can be used to enable CORS with various options.

The following middlewares are not added by default, but it's nice to know they exist at least:

- compression: compress all request. e.g. `app.use(compression())`
- session: create sessions. e.g. `app.use(session({secret: 'Secr3t'}))`
- method-override: `app.use(methodOverride('_method'))` Override methods to the one specified on the `_method` param. e.g. `GET /resource/1?_method=DELETE` will become `DELETE /resource/1`.
- response-time: `app.use(responseTime())` adds `X-Response-Time` header to responses.
- errorhandler: Aid development, by sending full error stack traces to the client when an error occurs. `app.use(errorhandler())`. It is good practice to surround it with an if statement to check `process.env.NODE_ENV === 'development'`.
- vhost: Allows you to use different stack of middlewares depending on the request `hostname`. e.g. `app.use(vhost('*.user.local', userapp))` and `app.use(vhost('assets-*.example.com', staticapp))` where `userapp` and `staticapp` are different express instances with different middlewares.
- csurf: Adds a **C**ross-**s**ite **r**equest **f**orgery (CSRF) protection by adding a token to responds either via `session` or `cookie-parser` middleware. `app.use(csrf());`
- timeout: halt execution if it takes more that a given time. e.g. `app.use(timeout('5s'));`. However you need to check by yourself under every request with a middleware that checks `if (!req.timedout) next();`.

http://adrianmejia.com/blog/2014/10/01/creating-a-restful-api-tutorial-with-nodejs-and-mongodb/

# Beispiel Logging,
# aber nicht für statische Files unter `'/public'`

```
router.use(express.static(__dirname + '/public'));
router.use(logger());
router.use(function(req, res){
  res.send('Hello');
});
```

Reihenfolge wichtig: Wenn `router.use(logger());` an erster Stelle stünde, würden alle Routen geloggt.

http://expressjs.com/de/4x/api.html#router.use

# Beispiel Router

```
const express = require('express');
const mongodb = require('mongodb');

const router = express.Router();

// Get Posts
router.get('/', async (req, res) => {
 const posts = await loadPostsCollection();
 res.send(await posts.find({}).toArray());
});

// Add Post
router.post('/', async (req, res) => {
 const posts = await loadPostsCollection();
 await posts.insertOne({
   text: req.body.text,
   createdAt: new Date()
 });
 res.status(201).send();
});
```

```
// Delete Post
router.delete('/:id', async (req, res) => {
 const posts = await loadPostsCollection();
 await posts.deleteOne({ _id: new mongodb.ObjectID(req.params.id) });
 res.status(200).send({});
});

async function loadPostsCollection() {
 const client = await mongodb.MongoClient.connect(
   'mongodb://localhost:27017/fullstack'
 );

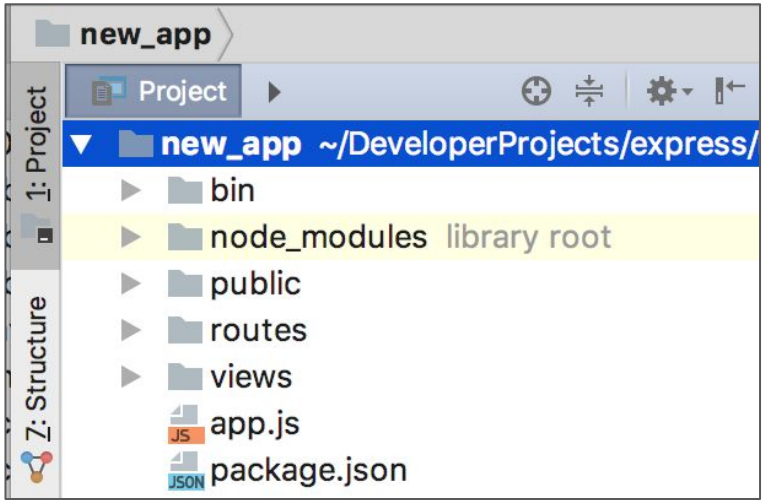
 return client.db('vue_express').collection('posts');
}

module.exports = router;
```

https://www.youtube.com/watch?v=X-JZ-QPApUs

# express.js - Generator-CLI: Generierter Web-Server

- npm install -g express-generator@4 *# erzeugt CLI "express"*
- express --no-view new_app *# erzeugt neues Directory "new_app"*
- cd new_app && npm install *# installiert abhängige Module*
- DEBUG=new-app:* npm start *# startet generierten Server*




https://www.npmjs.com/package/express

# Gliederung



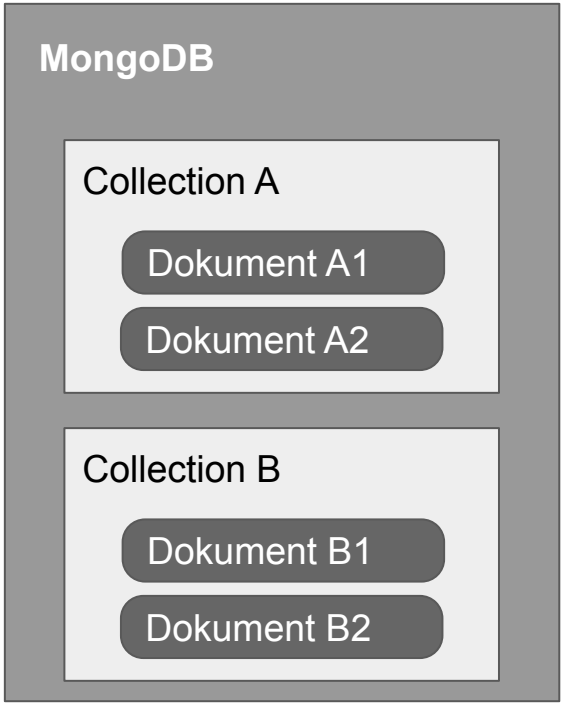
1. Node.js
2. npm
3. Express.js
4. MongoDB
5. Ausblick

# MongoDB

- abgeleitet vom engl. hu**mongo**us, „gigantisch“
- in C++ implementiert seit 2007
- **NoSQL** database
- **dokumentenorientiert**
  - Dokument = JSON-ähnliche Struktur
  - UTF-8-codiert
- **schemalos**
  - zur Laufzeit neue Strukturen erlaubt

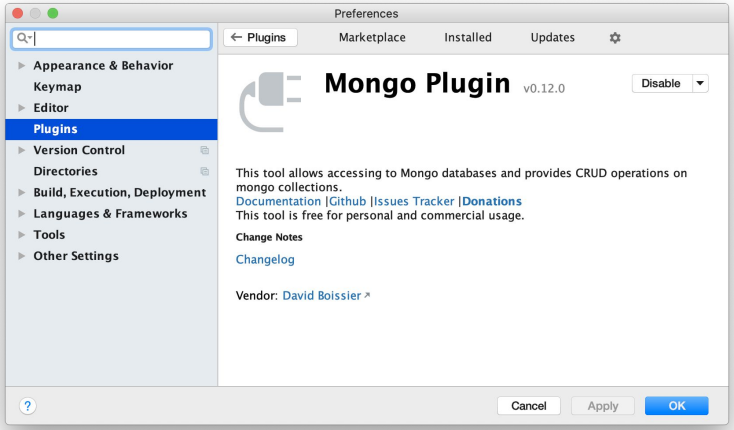
```
$ brew tap mongodb/brew
$ brew install mongodb-community
```

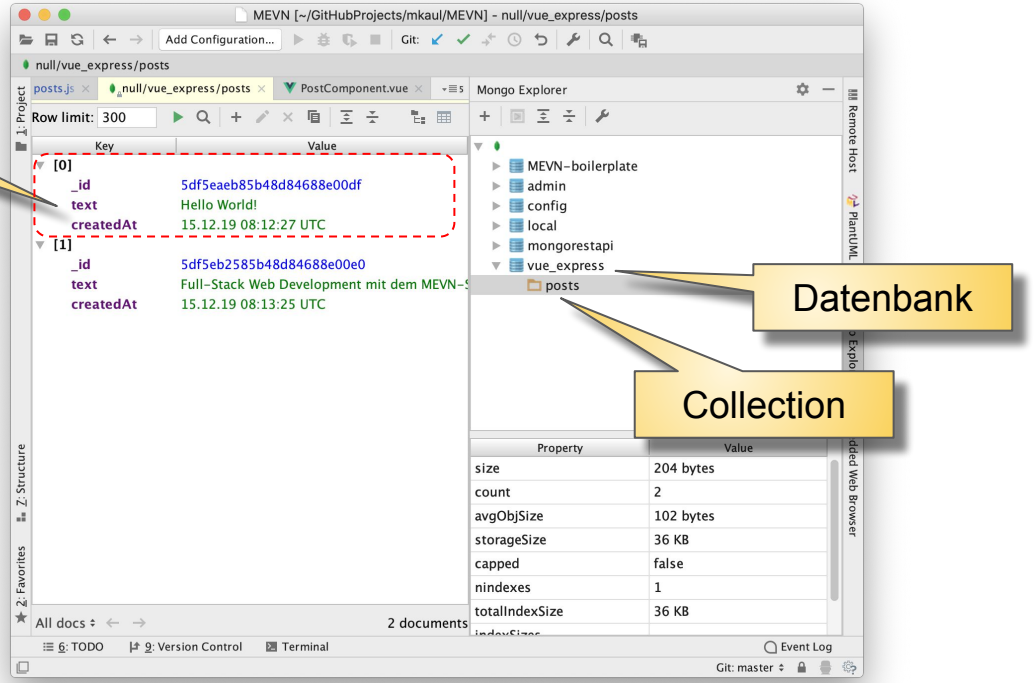
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/




https://docs.mongodb.com/manual/introduction/

# WebStorm Plugin




https://github.com/dboissier/mongo4idea

# WebEng-Lösungen in MongoDB gespeichert

adminMongo is a Web based user interface (GUI) to handle all your MongoDB connections/databases needs.https://adminmongo.markmoffat.com


https://github.com/mrvautin/adminMongo

# Verbindung zur MongoDB herstellen

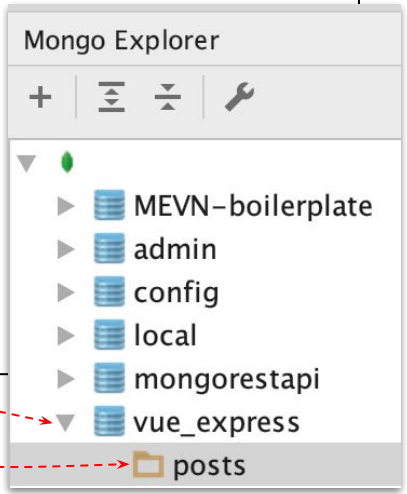
```
const express = require('express');
const mongodb = require('mongodb');

(async function(){
 const client = await mongodb.MongoClient.connect('mongodb://localhost:27017/');
 const router = express.Router();

// Get Posts
 router.get('/', async (req, res) => {
   const posts = await client.db('vue_express').collection('posts');
   res.send(await posts.find({}).toArray());
 });
}());
```

connection string

Query string



# MongoDB Queries

```
const posts = await client.db('vue_express').collection('posts');
const docs = await posts.find({ }).toArray();
```

- all documents in the collection
  - posts.find( { } )
- all documents with a given value abc
  - posts.find( { text: "abc" } )
- all documents with multiple given values A and D
  - posts.find( { text: { $in: [ "A", "D" ] } } )
- all documents with quantity less than 30
  - posts.find( { qty: { $lt: 30 } } )

https://docs.mongodb.com/manual/tutorial/query-documents/

# Schema-Bindung, Typisierung, Validierung mit **mongoose**


```
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/');

const Cat = mongoose.model('Cat', { name: String });

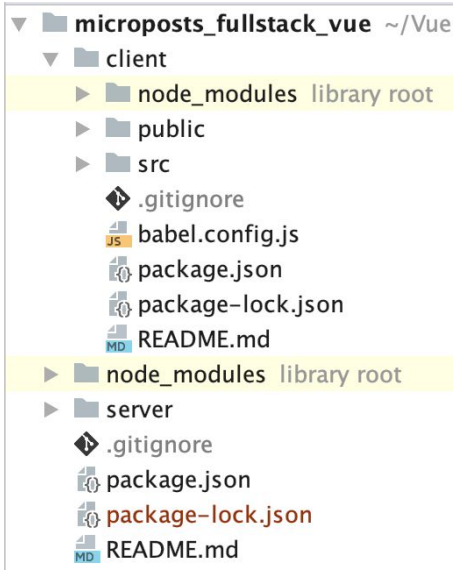
const kitty = new Cat({ name: 'Zildjian' });
kitty.save().then(() => console.log('meow'));
```

connection string

Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

https://mongoosejs.com/

# Vue Client generieren mit @vue/cli




$ vue create client

Vue CLI v4.1.1
✨ Creating project in /Users/mkaul2m/VueProjects/microposts_fullstack_vue/client.
⚙️ Installing CLI plugins. This might take a while...

added 1201 packages from 843 contributors and audited 24234 packages in 15.22s

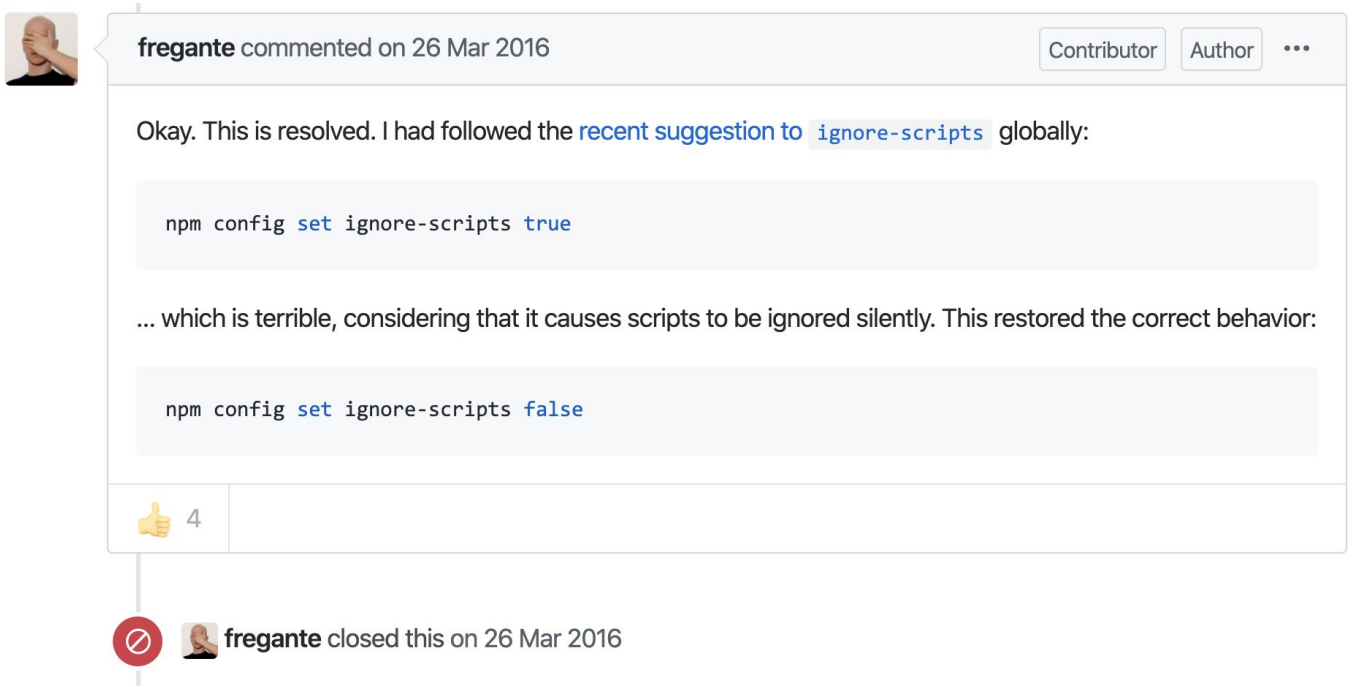
🚀 Invoking generators...
📦 Installing additional dependencies...

added 59 packages from 53 contributors and audited 24521 packages in 5.584s
🎉 Successfully created project client.
👉 Get started with the following commands:

$ cd client
$ npm config set ignore-scripts true
$ npm run serve

https://github.com/npm/npm/issues/12082

# Exkurs:
# Recent Bug: npm run-script doesn't run scripts



**fregante** commented on 26 Mar 2016 — Contributor — Author

Okay. This is resolved. I had followed the recent suggestion to `ignore-scripts` globally:

```
npm config set ignore-scripts true
```

... which is terrible, considering that it causes scripts to be ignored silently. This restored the correct behavior:

```
npm config set ignore-scripts false
```

4

**fregante** closed this on 26 Mar 2016

https://github.com/npm/npm/issues/12082

# Ersetze generierten Vue Client Code durch eigenen

```
<template>
 <div class="container">
   <h1>Latest Posts</h1>
   <div class="create-post">
     <label for="create-post">Insert here</label>
     <input type="text" id="create-post" v-model="text">
     <button v-on:click="createPost">Post!</button>
   </div>
   <hr>
   <div class="posts-container">
     <div class="post" v-for="(post, index) in posts"
          v-bind:item="post" v-bind:key="post._id">
            <p class="text">{{ post.text }}</p>
     </div>
   </div>
 </div>
</template>
```


```
<script>
import PostService from '../PostService'

export default {
 name: 'PostComponent',
 data() {
   return { posts: [], text: '',  }
 },
 async created(){
    this.posts = await PostService.getPosts()
 },
 methods: {
   async createPost() {
     await PostService.insertPost( this.text );
     this.text = '';
     this.posts = await PostService.getPosts()
   }
 }
}
</script>
```

https://www.youtube.com/watch?v=X-JZ-QPApUs

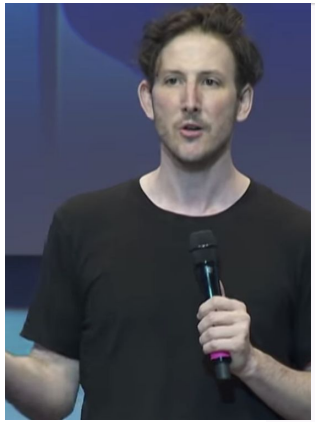
# Gliederung



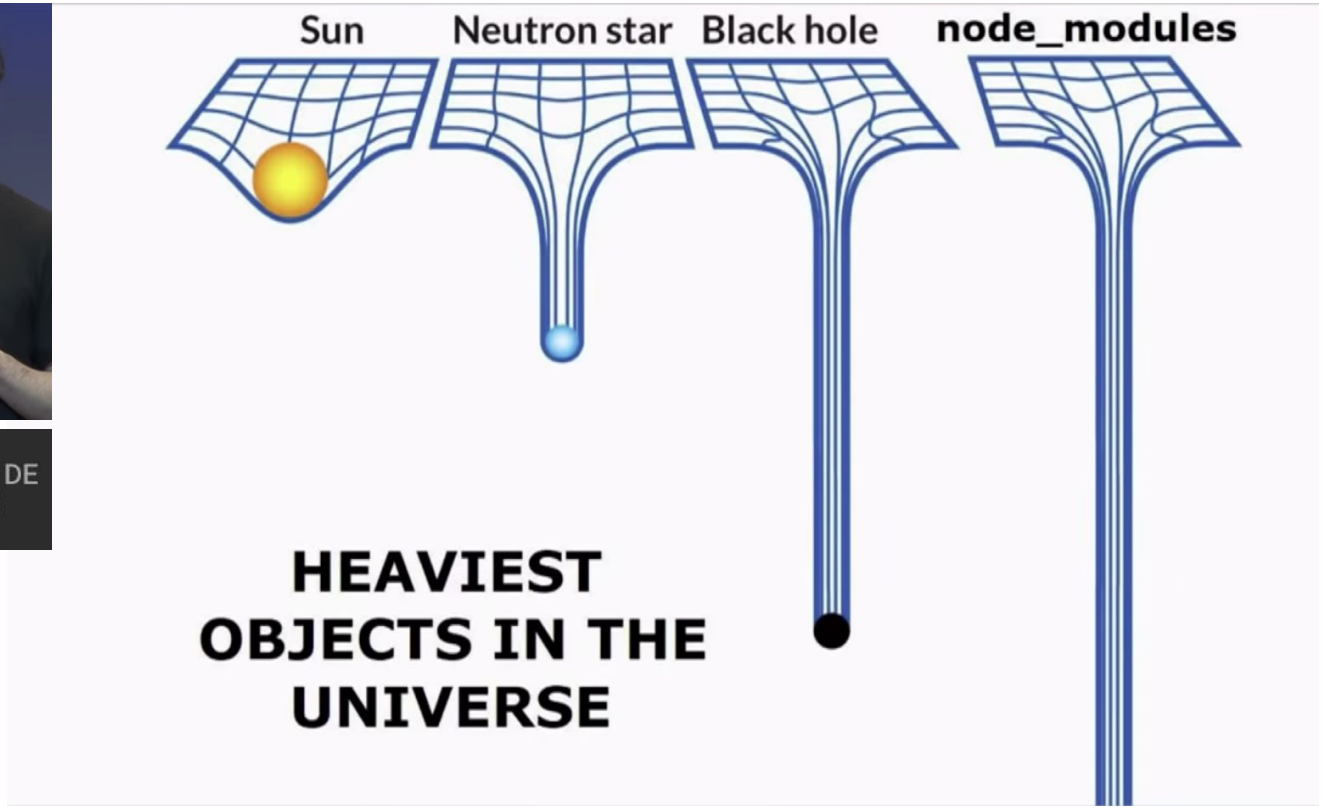
1. Node.js
2. npm
3. Express.js
4. MongoDB
5. Ausblick

# Ausblick

# npm packages are the heaviest objects in universe

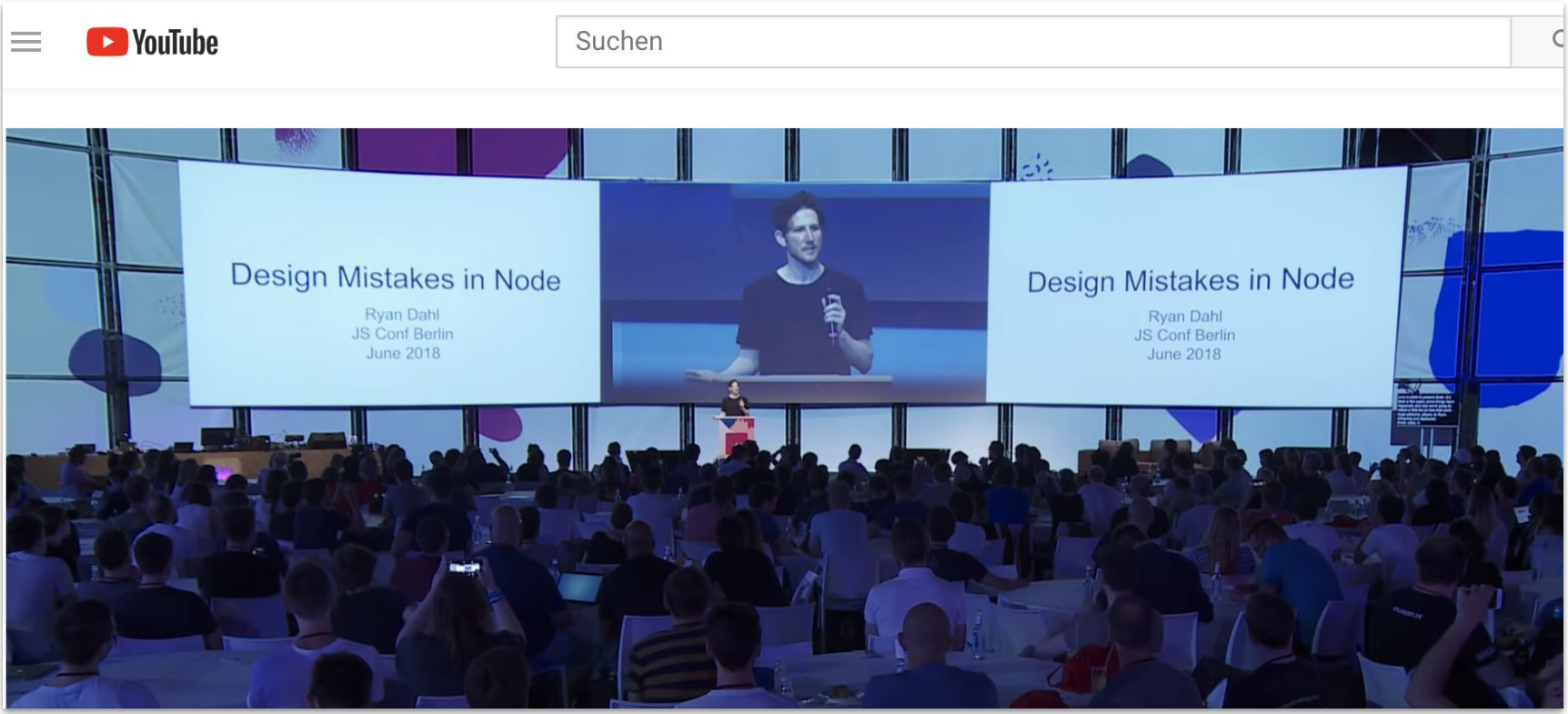


YouTube DE



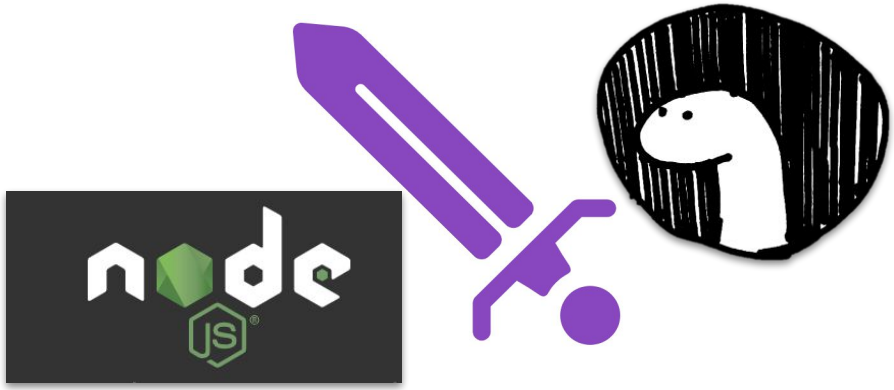

https://youtu.be/M3BM9TB-8yA?t=768

# Zukunft: Deno a secure TypeScript runtime built on V8




https://www.youtube.com/watch?v=M3BM9TB-8yA

# Deno’s top features



- Security
  - only in privileged mode:
    - accessing environment variables
    - accessing the file system
- Bessere Kompatibilität mit der Browser Runtime
  - ⇒ Isomorphic JavaScript
- Module system
  - No package.json, no node_modules.
  - **import { test } from "https://anyserver.com/anyModule.mjs";**
- TypeScript

https://medium.com/lean-mind/deno-node-js-killer-718c8969770b

# Deno

https://github.com/denoland/deno/releases

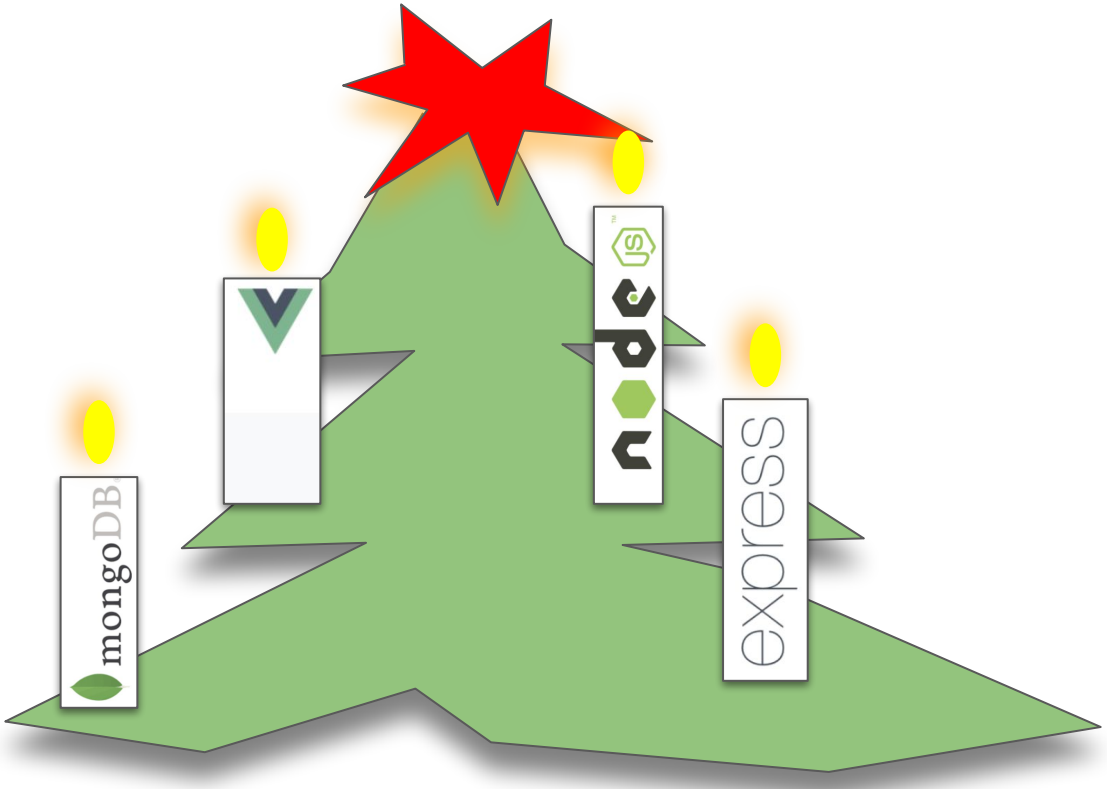
| Linux & Mac | Windows |
| --- | --- |
| | |

## Deno

Deno is a program for executing JavaScript and TypeScript outside of the web browser.

github.com/denoland/deno
Documentation
API Reference
Deno standard modules

### Getting started




Frohe Weihnachten !