

NoSQL Datenbanken

Vorlesung – Hochschule Mannheim

Klassifizierungen

Inhaltsverzeichnis

- ▶ Key-Value Stores
- ▶ Spaltenorientierte Datenbanken
- ▶ Dokumentenbasierte Datenbanken
- ▶ Graphendatenbanken




















Key-Value Datenbanken



Project Voldemort
A distributed database.

Weitere

- ▶ Big Table (Google)
- ▶ Hbase (Apache)
- ▶ CouchDB (Apache 2 Lizenz)
- ▶ Cassandra (facebook, jetzt Apache 2.0 Lizenz)

Rang			DBMS	Datenbankmodell	Punkte		
Nov 2015	Okt 2015	Nov 2014			Nov 2015	Okt 2015	Nov 2014
1.	1.	1.	Redis 	Key-Value Store	102,41	+3,61	+20,06
2.	2.	2.	Memcached	Key-Value Store	32,39	+0,82	-0,20
3.	3.	 4.	Amazon DynamoDB	Multi-Model 	21,75	+0,42	+9,43
4.	4.	 3.	Riak 	Key-Value Store	15,06	-0,35	+2,03
5.	5.	5.	Ehcache	Key-Value Store	7,88	-0,36	+0,42
6.	6.	6.	Hazelcast	Key-Value Store	7,06	+0,03	+1,91
7.	7.	 10.	OrientDB	Multi-Model 	5,50	+0,57	+3,48
8.	8.	 9.	Oracle Coherence	Key-Value Store	3,99	-0,16	+1,26
9.	9.	 7.	Berkeley DB	Key-Value Store	3,79	-0,23	+0,67
10.	 11.	 13.	Aerospike 	Key-Value Store	3,35	+0,32	+2,34
11.	 10.	 8.	Amazon SimpleDB	Key-Value Store	3,22	+0,00	+0,25
12.	12.	 11.	Oracle NoSQL	Key-Value Store	2,67	+0,08	+1,18
13.	13.	 12.	Infinispan	Key-Value Store	2,42	-0,03	+1,26
14.	14.	14.	LevelDB	Key-Value Store	1,88	+0,11	+1,01
15.	15.	 22.	ArangoDB 	Multi-Model 	1,61	+0,13	+1,30

Quelle: <http://db-engines.com/de/ranking/key-value+store>

Geschichte

- ▶ Älteste Datenmodell (werden seit den 70ern eingesetzt)
- ▶ Durch Web 2.0 und Cloud-Zeitalter einen große Aufschwung
- ▶ Anhänger: Facebook, Amazon und Google
- ▶ Durch ihre Einfachheit sehr gut skalierbar

Definition

“Key-Value Stores sind die wohl einfachste Form von Datenbankmanagementsystemen. Sie können lediglich Paare von Schlüsseln und Werten abspeichern, sowie die Werte anhand des Schlüssels wieder zurückliefern.”-
<http://db-engines.com/>

Datenmodell

- ▶ Einfaches Schlüssel- und Werteschema
- ▶ Schlüssel können in Namensräume sowie in Datenbanken aufgeteilt sein.
- ▶ Values werden anhand ihrer Schlüssel angerufen
- ▶ Operationen:
 - `put(key, value)`
 - `get(key)`
 - `remove(key)`

Anwendung

- ▶ Performanz:
 - Hohe Lese./Schreibe-Rate durch einfache Indizierung
- ▶ Session-Daten, Warenkörbe
- ▶ Embedded Systems
- ▶ In-Process Datenbanken

Implementierungen

- ▶ Welche Datenstrukturen werden unterstützt?
- ▶ Wie werden die Daten gespeichert? (RAM, Festplatte)
- ▶ Skalierung über Sharding, Master-Slave oder Master-Master
- ▶ Strenge Konsistenz oder eventually?

Vorteile

- ▶ Skalierbarkeit
- ▶ Effiziente und schnelle Datenverarbeitung
- ▶ Schemafrei
- ▶ Kein blockieren bei Schreibe und Lesezugriffe

Nachteile

- ▶ Zugriff nur über einen Schlüssel
- ▶ Eingeschränkte Abfragemöglichkeiten

Redis



flickr[®]

digg

Redis

- ▶ **RE**mote **DI**ctionary **S**erver
- ▶ In-Memory Datenbank
- ▶ Erschienen 2009 von Salvatore Sanfilippo
- ▶ Aktuelle Version 3.0.5 (15.Oktober 2015)
- ▶ Plattformunabhängig
- ▶ In ANSI C geschrieben
- ▶ Weit verbreiteste Key-Value Store

Data Types

Kein Plain Key-Value Store

Data structures server

► Datentypen:

- Lists (linked lists)
- Sets
- Sorted sets
- Hashes (maps)

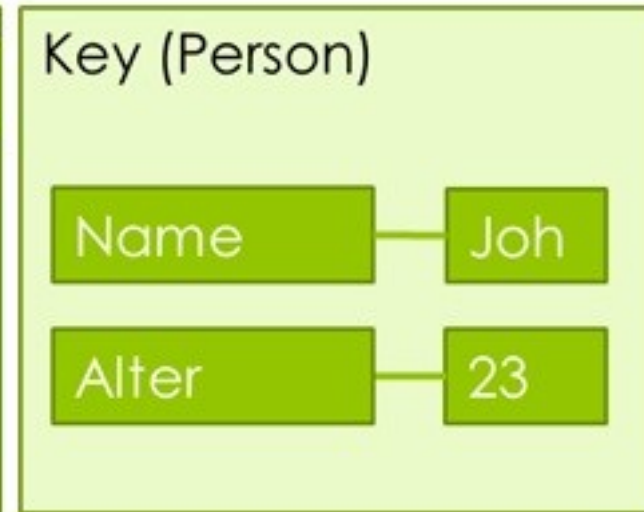
Set



Liste



Hash



In-Memory Datenbank

- ▶ Behält Daten im Hauptspeicher
- ▶ Kurze Zugriffszeiten
- ▶ Verzicht auf Persistenz
 - Zusätzliche Module implementieren Persistenz

Implementierung

- ▶ Unterstützte Treiber: C, C++, Dart, Erlang, Haskell, JavaScript (Node.js) Perl, PHP, Python
- ▶ Einsatz als Cache
- ▶ Performanz vor Features bzgl. Speicherbedarf
- ▶ API/ Zugriffskonzept: RESP: Redis Serialization Protocol
- ▶ Sharding als Partitionierungsmechanismus
- ▶ Eventual Consistency
- ▶ Schemafrei
- ▶ Master-Slave Replikation

Installation

```
$ wget http://download.redis.io/releases/redis-3.0.5.tar.gz  
$ tar xzf redis-3.0.5.tar.gz  
$ cd redis-3.0.5  
$ make
```

Start Server

```
$ src/redis-server
```

Start built-in client:

```
$ src/redis-cli
```

API – Store Data

SET server:name "fido"

GET server:name => "fido"

SET <namespace>:name "fido"

GET <namespace>:name => "fido"

SET katze "martin"

GET katze => "martin"

Wird zweimal derselbe Key hinzugefügt, wird alter Value überschrieben.

API – Delete Key

SET connections 10

INCR connections => 11

INCR connections => 12

DEL connections

INCR connections => 1

API – Expire

```
SET resource:lock "Redis Demo 1"
```

```
EXPIRE resource:lock 120
```

```
TTL resource:lock => 119 //Wielange der Value nochgültig ist
```

```
SET resource:lock "Redis Demo 2"
```

```
TTL resource:lock => -1
```


API – Listen

//Erstellt Liste cats und pushed Value an das Ende der Liste

R PUSH cats "Morle"

R PUSH cats "Christoph"

//Erstell Liste cats und pushed Value an den Anfang der Lise

R PUSH cats "Tiger"

//gibt die gewünschte Elemente deer Liste cats zurück

L RANGE friends 0 -1 => 1) "Tiger", 2) "Morle", 3) "Christoph"

L RANGE friends 0 1 => 1) "Tiger", 2) "Alice"

L RANGE friends 1 2 => 1) "Morle", 2) "Christoph"

API – Listen

//Gibt aktuelle Länge der Liste zurück

LLEN cats => 3

//entfernt erstes / letztes Element und gibt dieses zurück

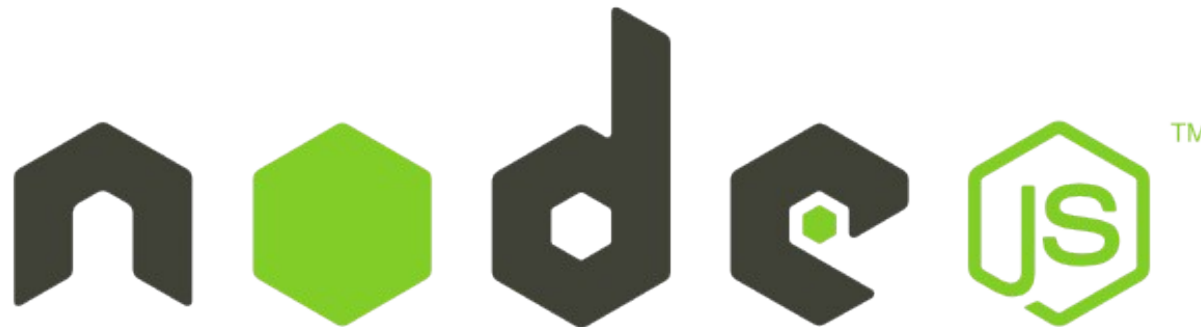
LPOP/ RPOP friends => "Tiger" / "Christoph"

DO it Yourself

1. Installiert Redis
2. Startet Server und Client
3. Legt folgende Key Value Paare an:
 1. Katze - “Morle”
 2. Professor - “Smits”
 3. Name - “Martina”
4. Legt eine Liste cats an und speichert dort 11 Katzen
5. Löscht Elemente heraus und überprüft regelmäßig die Länge

WebServer

- ▶ Um mit der Datenbank reden zu können benötigt es einen WebServer



The background features abstract, overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the frame, creating a modern, layered effect.

Why should I node?

Kleiner Ausflug in die Geschichte ...

Anforderungen an gute Web-Server

- ▶ Viele gleichzeitig eintreffende Requests **schnell** und **zufriedenstellend** verarbeiten.
- ▶ Viel: Das C10k Problem
- ▶ Schnell: „Echtzeit“
- ▶ Zufriedenstellend: Nach Verarbeitung ein Ergebnis

Echtzeit

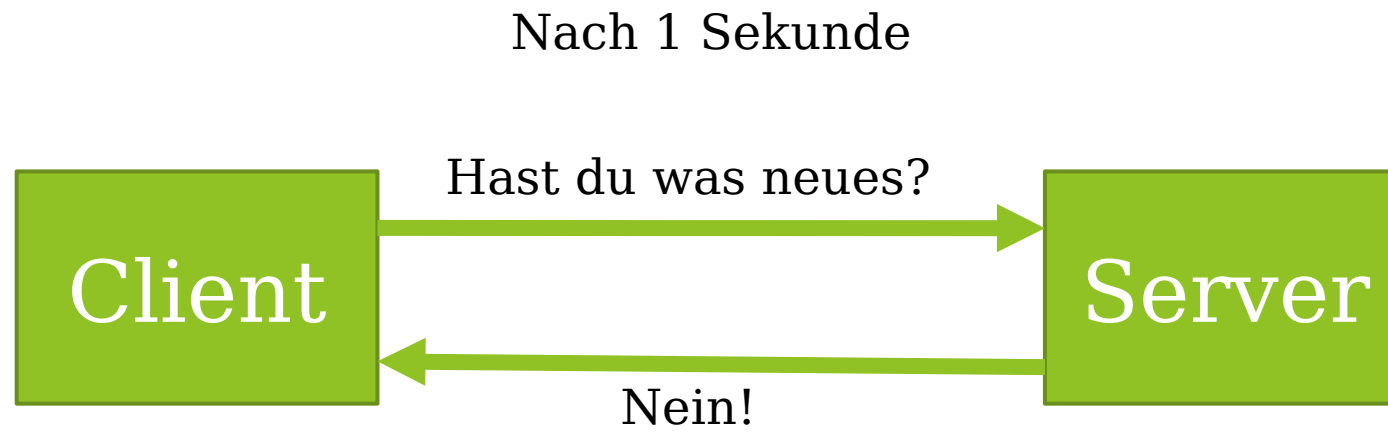
► Das sagt Wiki:

„Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen“

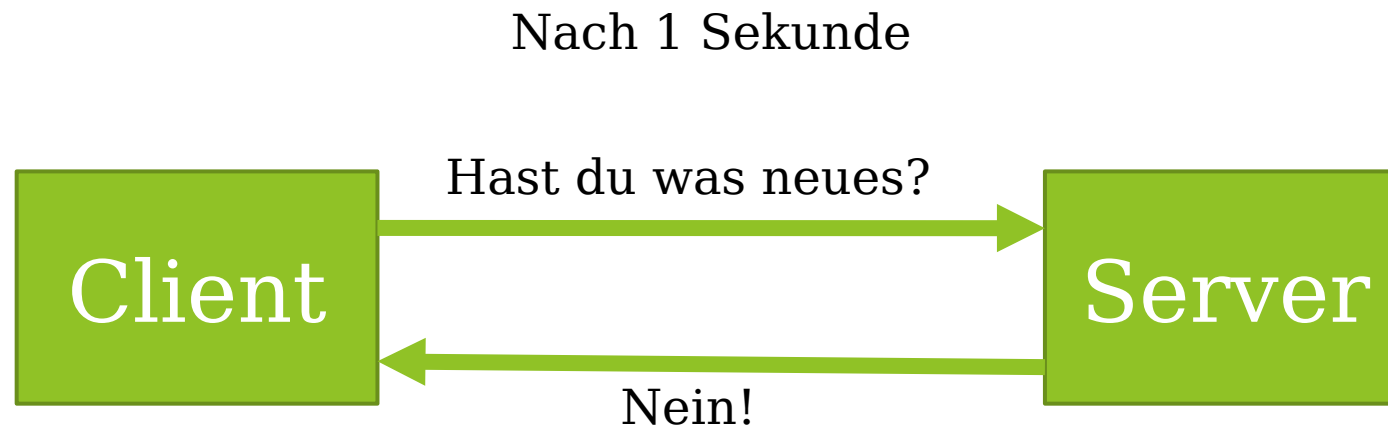
- Web-Seiten und Applikationen sollen sich wie Desktop-Anwendungen anfühlen
- Keine Verzögerung bei Client-Server Traffic

Umsetzung?

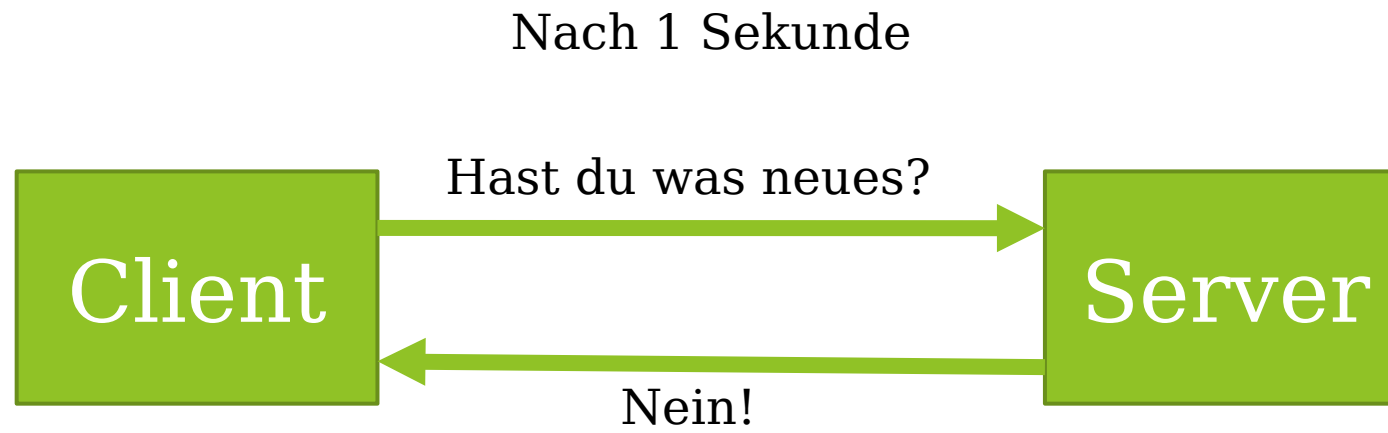
Polling(2000)



Polling(2000)



Polling(2000)



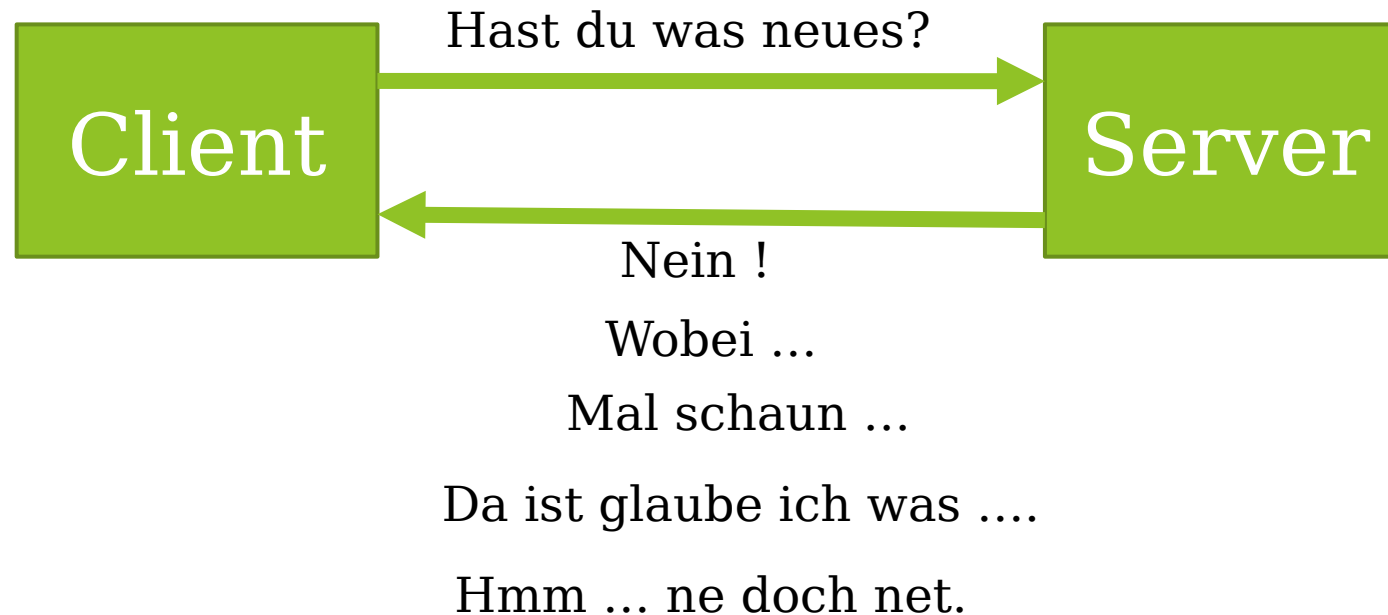
Polling(2000)

Client

Server



Comet / Long-Polling (2006)

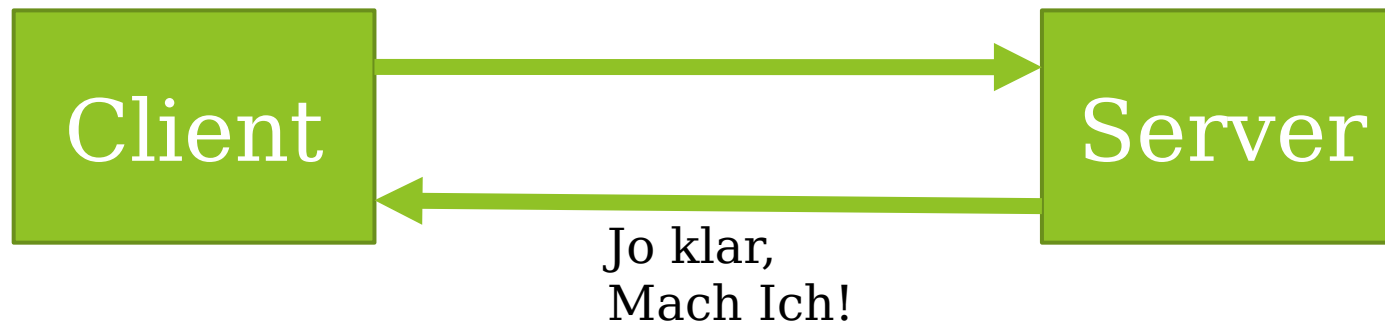


Comet / Long-Polling (2006)

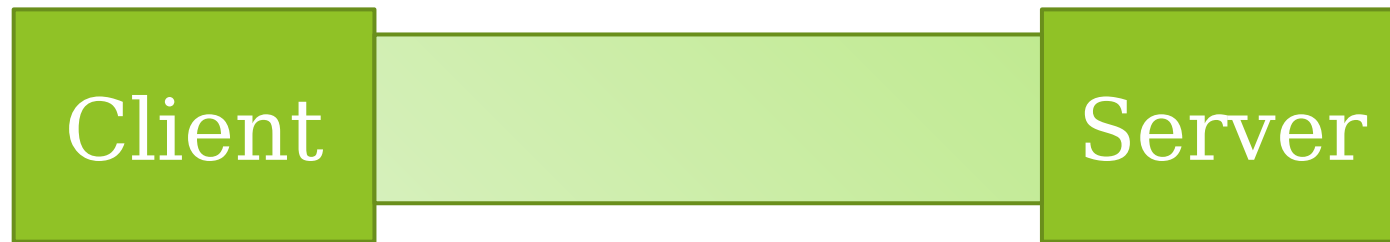


WebSockets(2011)

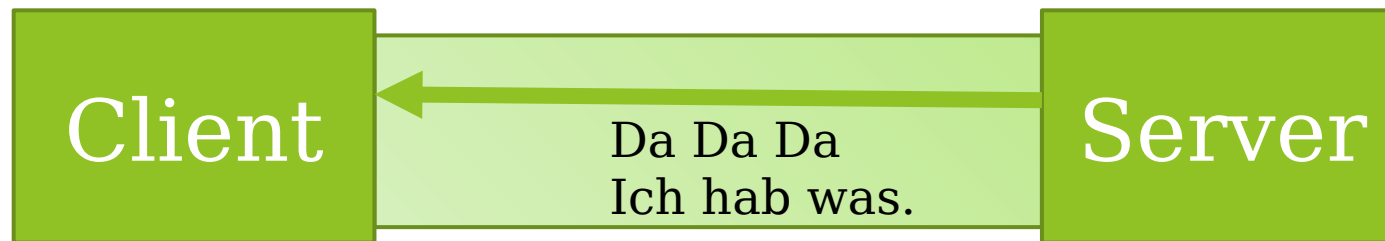
Ey Du gel, wenn du was neues
hast sagste bitte Bescheid.
Können wir dafür auch bitte ,nen anderes Protokoll benutzen?



WebSockets(2011)



WebSockets(2011)



WebSockets(2011)

- ▶ Neues Netzwerkprotokoll in der Anwendungsschicht (über der Transportschicht mit TCP)
- ▶ Erstellt eine bidirektionale Verbindung zwischen Client und Server
- ▶ Handshake:

`GET /chat HTTP/1.1`

`Host: server.example.com`

`Upgrade: websocket`

`Connection: Upgrade`

`HTTP/1.1 101 Switching Protocols`

`Upgrade:`

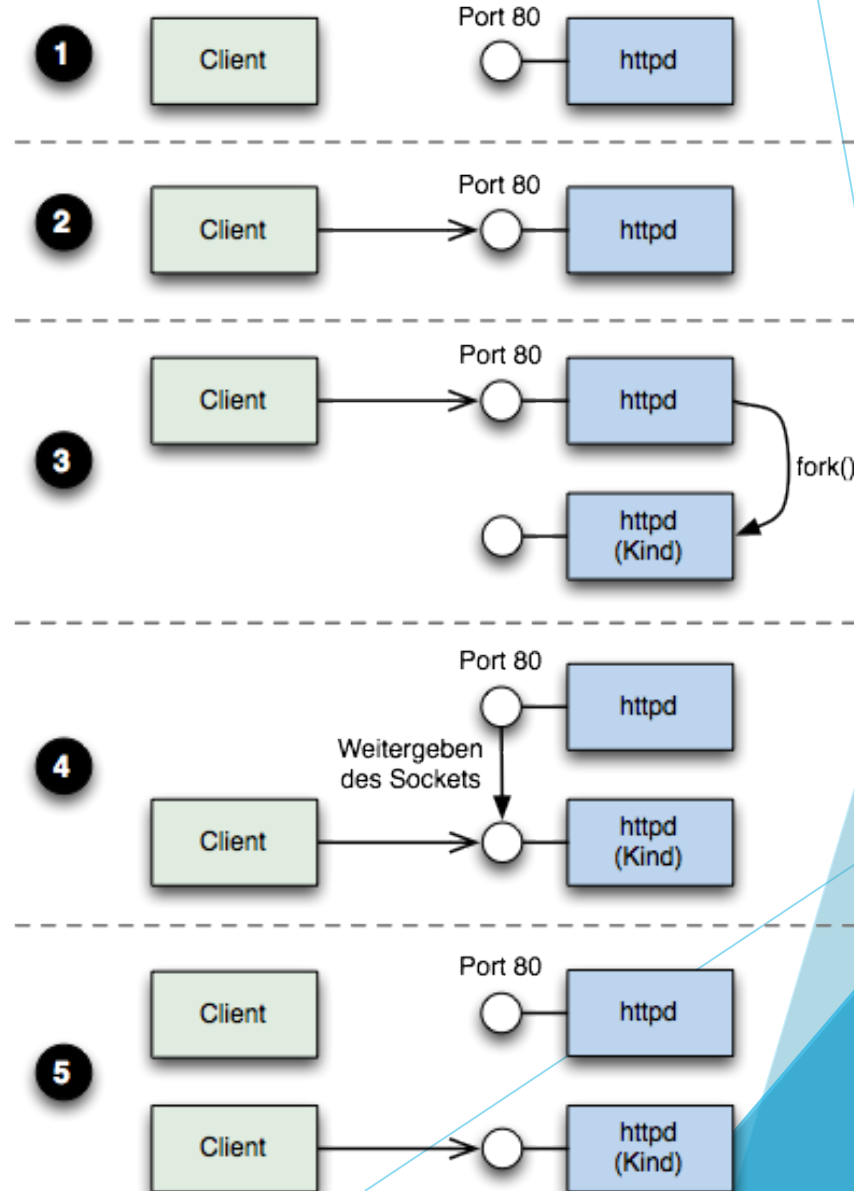
`websocket Connection: Upgrade`

Viele Requests verarbeiten

- ▶ Das C10k Problem
 - ▶ 10.000 Clients gleichzeitig von einem einzigen Web-Server bedienen
- ▶ Apache 1.x Prozessorientiert
- ▶ Apache 2.x Prozess- und threadorientiert
- ▶ Nginx und Node.js Ereignisorientiert

Apache 1.x Architektur

- Multi-Prozess (pre-fork)
- 1. ein Prozess öffnet Port 80 und wartet auf Anfragen
- 2. er nimmt Anfragen entgegen
- 3. sofort danach wird ein Kindprozess mit `fork()` erzeugt
- 4. die Client-Verbindung wird an den Kindprozess übergeben
- 5. der ursprüngliche Prozess kann wieder Verbindungen entgegennehmen



Apache 2.x Architektur

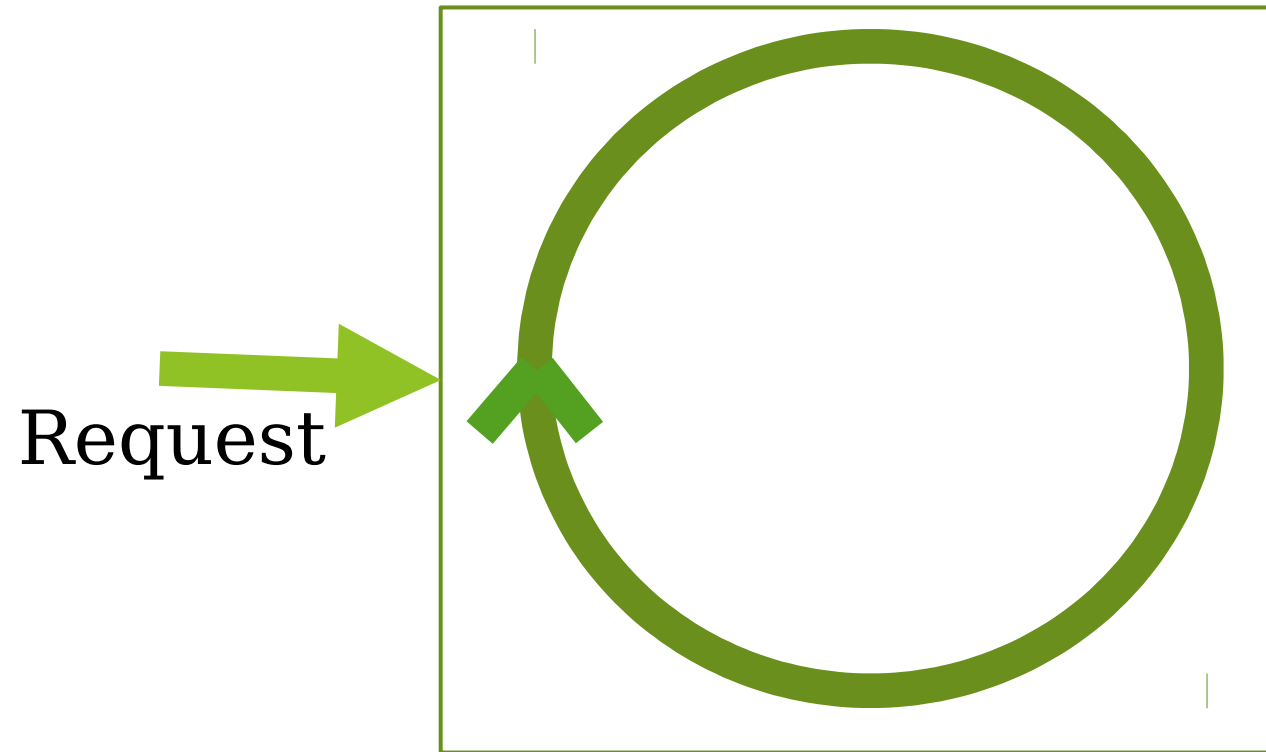
- ▶ Apache 2.x unterstützt neben pre-fork zusätzlich multi-threading (*worker*-Modell) innerhalb eines Servers
 - ▶ innerhalb eines Prozesses können mehrere Threads gestartet werden
 - ▶ jeder Thread kann einen Client bedienen
 - ▶ die Prozesse werden wie bei pre-fork behandelt, können jetzt aber parallel mehrere Anfragen bearbeiten
 - ▶ Modell reduziert drastisch den Speicherverbrauch bei vielen Clients (Für jeden Thread werden 2MByte reserviert -> 20GB)

Node.js: Event-loop

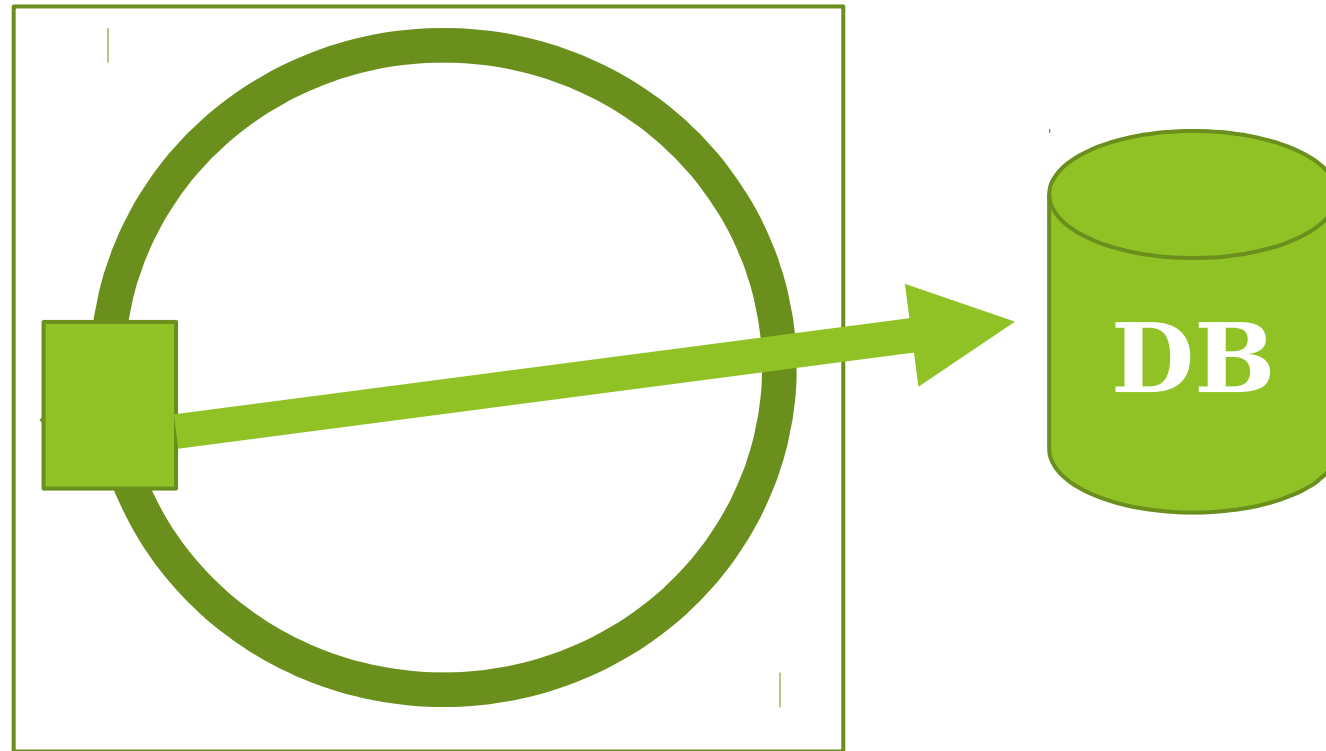
“With an event loop, you ask it to do something and it’ll get back to you when it’s done”

~@davidpadbury

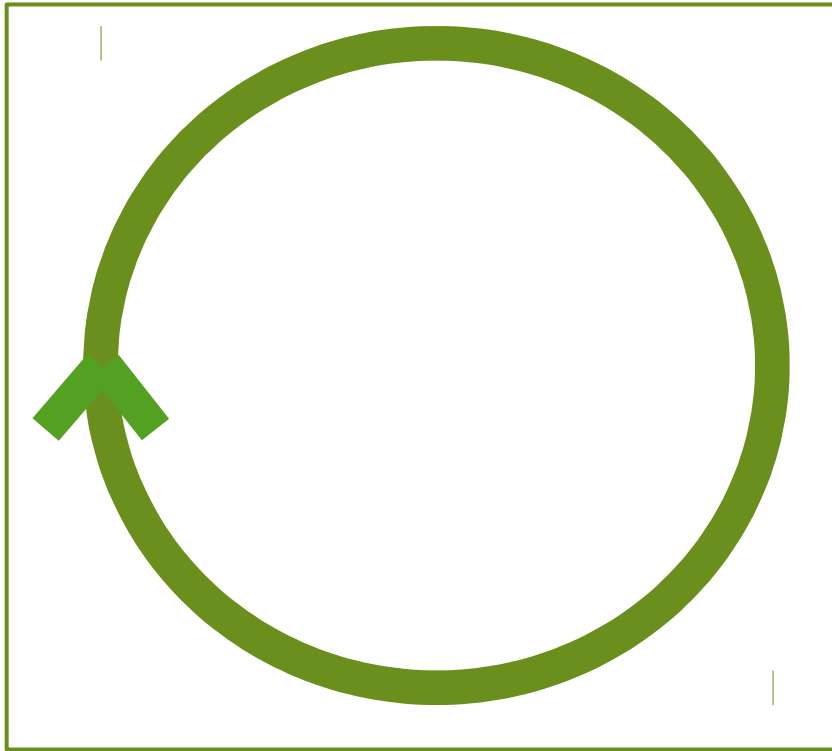
Node.js: Event-loop



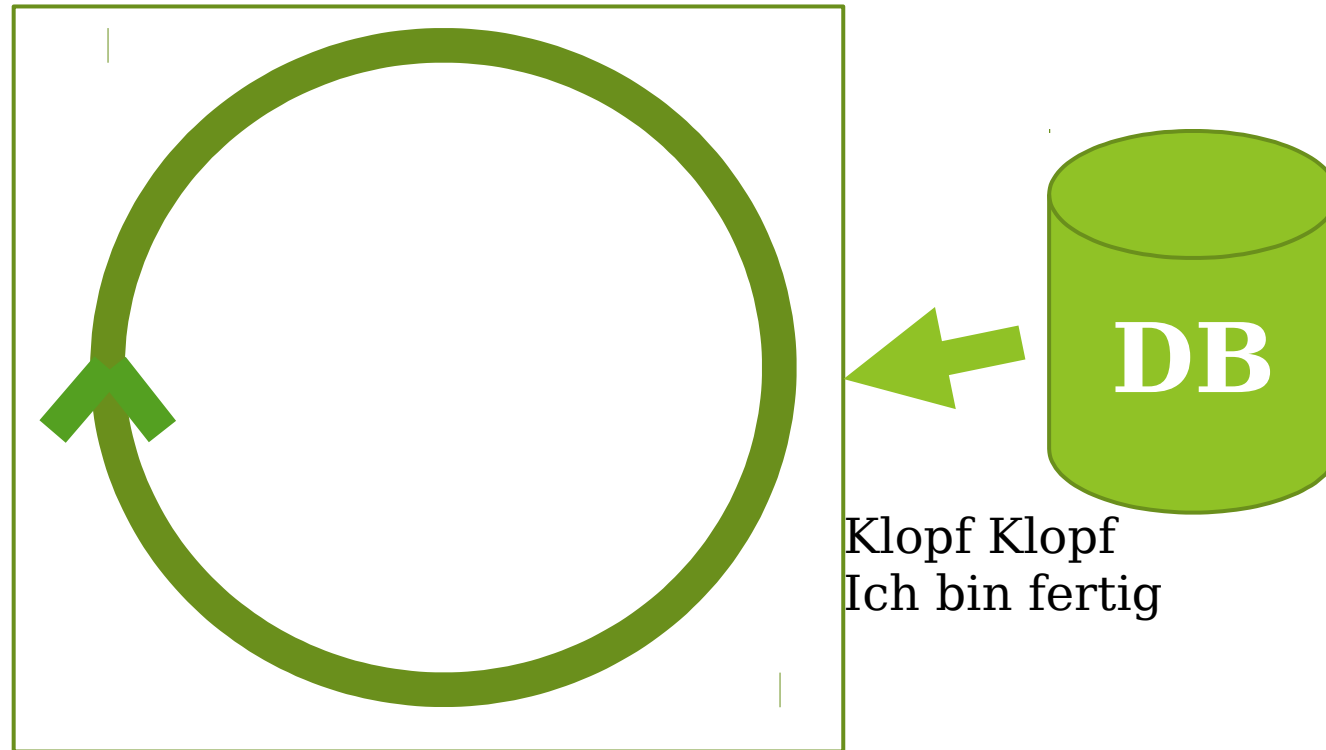
Node.js: Event-loop



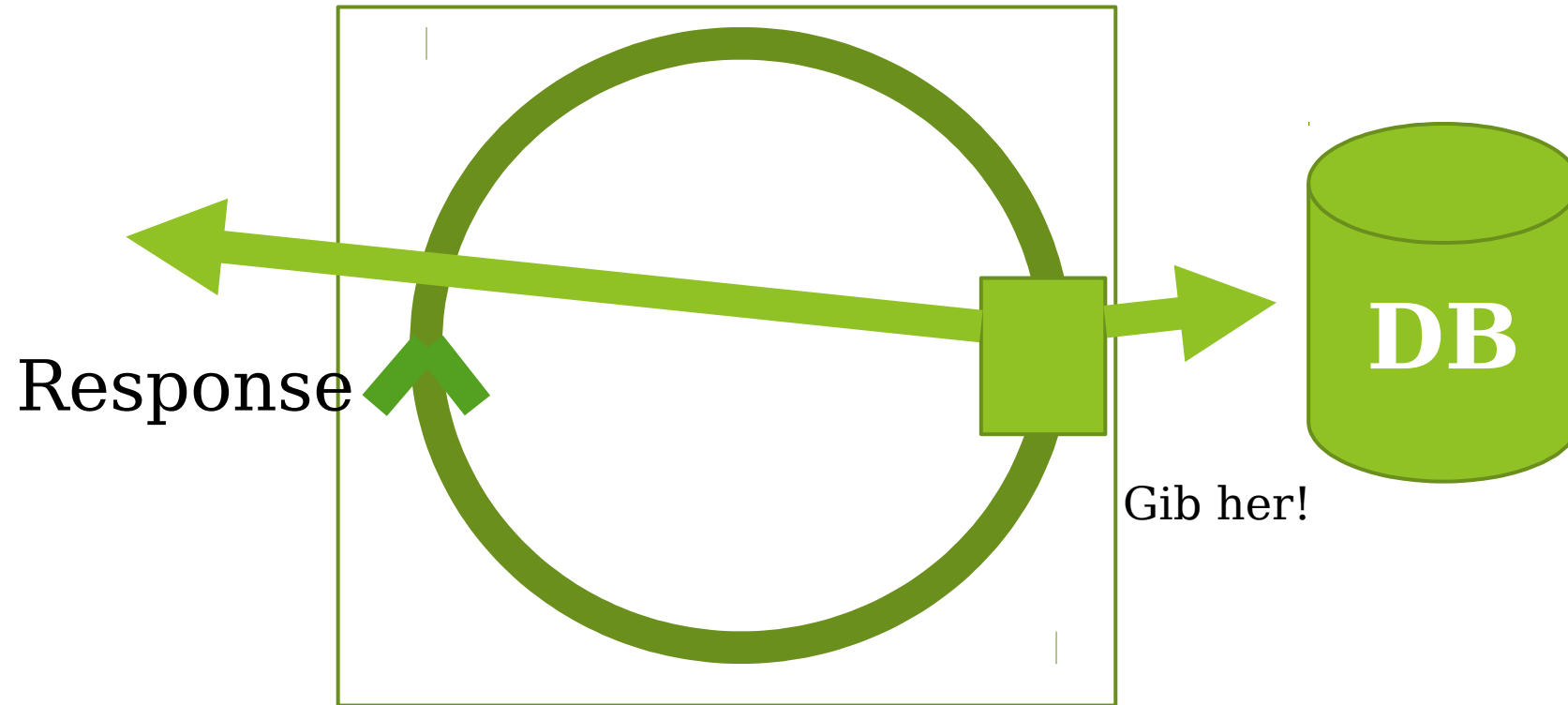
Node.js: Event-loop



Node.js: Event-loop



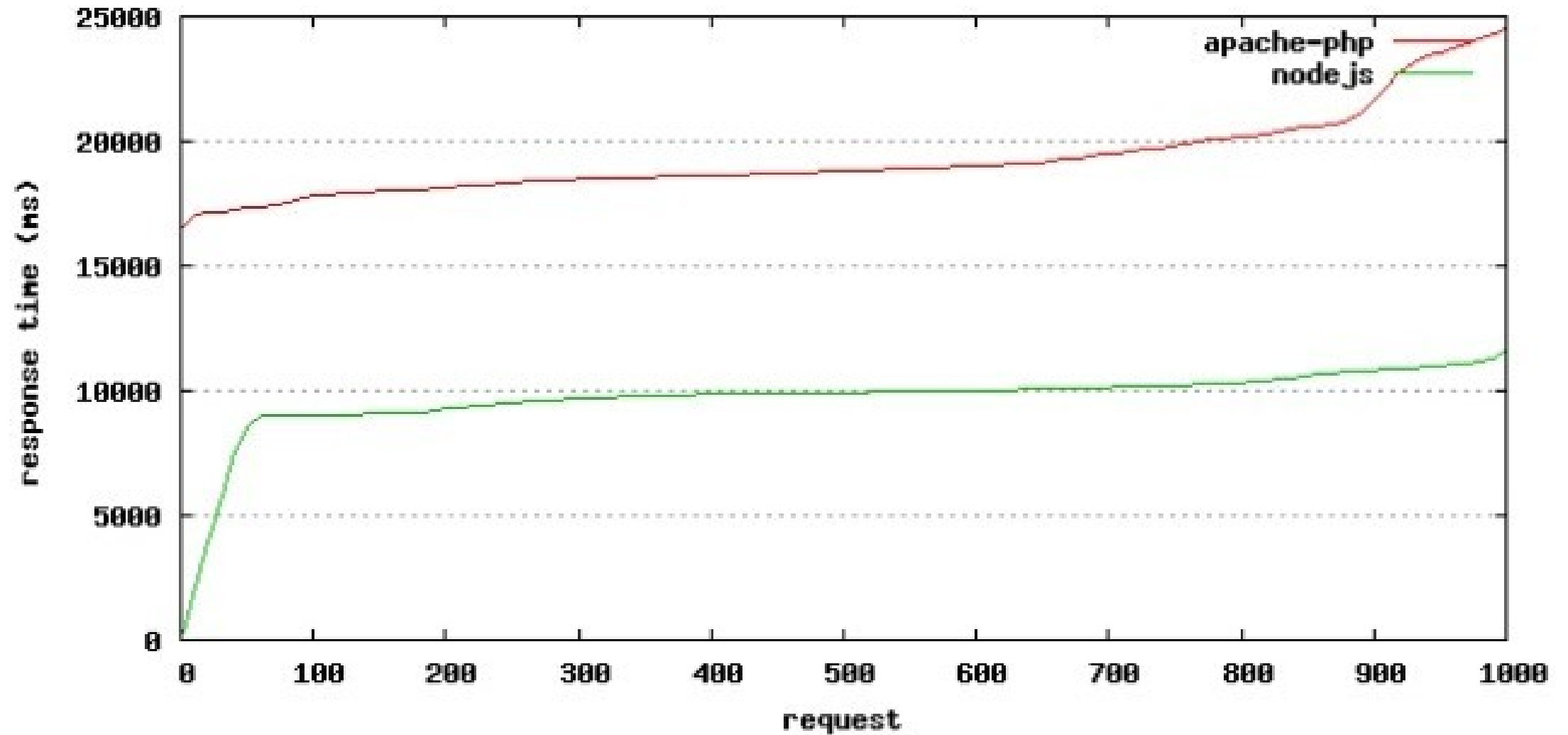
Node.js: Event-loop



One Thread to rule them all

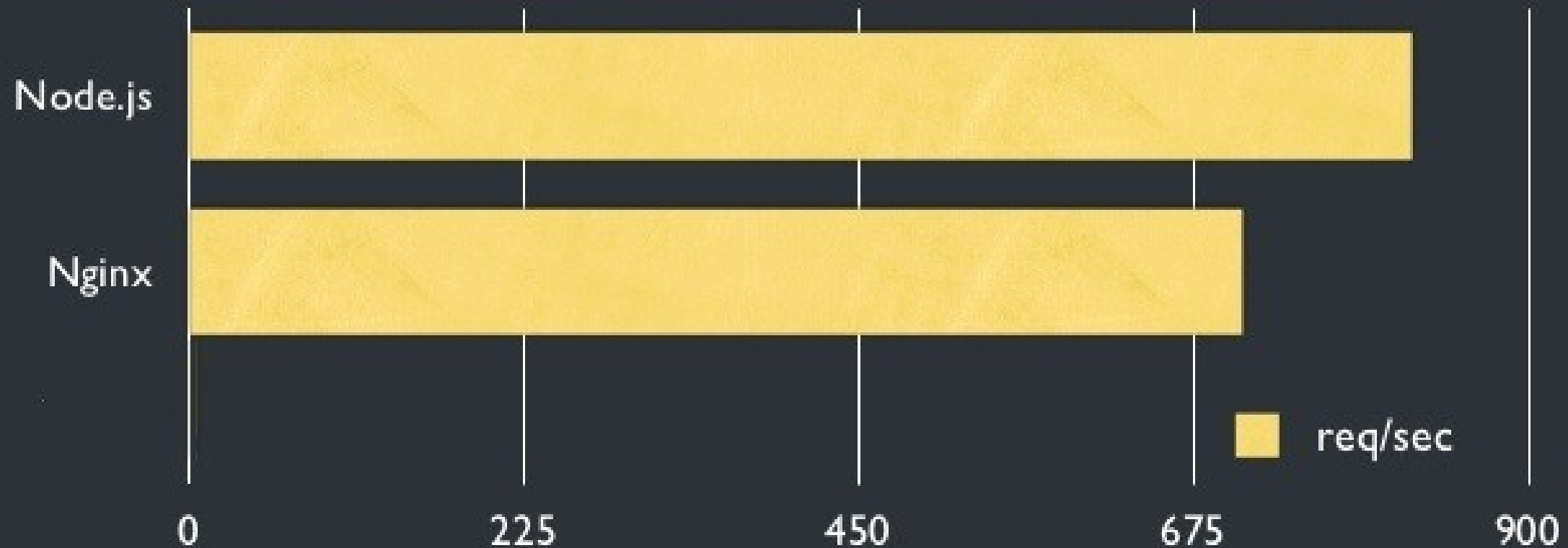
- ▶ leicht zu programmieren
- ▶ keine Deadlocks
- ▶ keine Race-Conditions

ab -n 1000 -c 50 (fetch and process html, read 1000 records from db)



Benchmark

100 concurrent clients
1 megabyte response





- ▶ Entwickelt von Ryan Dahl 2009
- ▶ Mittlerweile Joyent Inc.
- ▶ Google V8 Engine (Chrome)
- ▶ Serverseitiges JavaScript
- ▶ Ausgelegt für viel I/O
- ▶ Asynchron

Traditionelles I/O

```
var db_result = db.query("select * from  
table");  
doSomething(db_result); //wait!  
doSomeOtherVoodoo();    //blocked
```


Non-blocking I/O

```
db.query("select * from table",function(db_result)
{
    doSomething(db_result); //wait
});
doSomeOtherVoodoo();
```

Getting Started

Getting nodejs

- ▶ Aktuelle Version: 5.0.0 Stable
- ▶ nodejs.org/downloads
 - ▶ Windows Installer / Binaries
 - ▶ Mac OS X Installer / Binaries
 - ▶ Linux Binaries (.tar .gz)
 - ▶ Sourcecode
 - ▶ Um nodejs zu kompilieren wird Python 2.6 oder 2.7 benötigt.
- ▶ Das Framework Node.js
- ▶ NPM Paketmanager

Nodejs auf Linux

- ▶ Zwei Möglichkeiten:
 - ▶ Per Hand aus dem SourceCode übersetzen
 - ▶ Vorgefertigtes Paket verwenden

Per Hand

- ▶ Zuvor müssen zwei Abhängigkeiten installiert werden:
 - ▶ Werkzeuge zum Übersetzen basieren auf Python, daher muss Python in der Version 2.6 mindestens vorliegen
 - ▶ Falls Verschlüsselung der Datenströme (SSL/TLS) unterstützt werden soll muss das Paket libssl-dev installiert werden

```
$ sudo apt-get install python
```

```
$ sudo apt-get install libssl-dev
```

- ▶ Dann Node.js mithilfe von wget herunterladen:

```
$ wget http://nodejs.org/dist/v5.0.0/node-v5.0.0.tar.gz
```

- ▶ Entpacken:

```
$ tar xzf node-v5.0.0.tar.gz
```

```
$ cd node-v5.0.0
```

NPM

- ▶ Paketmanager für Node.js
- ▶ Abhängige Module können in einem JSON File definiert werden: package.json
- ▶ Mit `$ npm install` werden diese mit runtergeladen und in `node_modules` gespeichert.

```
$ npm install <packageName>
```

Erste Schritte

- ▶ JavaScript File anlegen.
- ▶ http-Modul (wird Defaultmäßig mitgeliefert importieren durch

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

- ▶ Mit der Methode `createServer(requesthandler)` beschreiben, wie der Server einen Request verarbeitet werden soll
- ▶ Mit der Methode `listen(Port, IP)` den Web-Server auf eine bestimmte Adresse lauschen lassen.

```
$ node server.js
```

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

```
http.createServer(function (req, res){  
    res.writeHead(200, {'Content-Type': 'text/plain'});  
    res.end('Hello World\n');  
}).listen(1337, '127.0.0.1');
```


Los geht's
Erstellt euren eigenen
Web-Server

Redis Treiber für Node

```
npm install redis
```

Server.js

```
var http = require('http'), server,  
    redis_client = require('redis').createClient();  
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/plain'});  
    <connection To database>  
}).listen(1337, "127.0.0.1");
```

Redis Treiber für Node

<RequestBody>

```
client.on("error", function (err) {  
    console.log("Error " + err);  
});
```

```
client.set("string key", "string val", redis.print);
```

https://github.com/NodeRedis/node_redis