Faculty of Science,
Technology
and Medicine

Web Programming

Volker Müller
University of Luxembourg

npm (www.npmjs.com)

npm = package manager for JS, written in JS

Default package manager for Node.JS

Manages also dependencies (similar to composer)

Searchable central repository with JS modules publicly accessible at npmjs.com

Global mod. installation: npm install -g <modname>

Local package installation done with configuration file package.json (created with npm init)

News (www.heise.de – 3.11.2020)

"Somebody" succeeded to publish new package twilio-npm in central NPM repository

→ Package opens Reverse Shell on the local system, gives access to the system to "somebody"

→ Malware!! → "Brandjacking" attack

Example shows that usage of central code repositories can lead to security issues

Remark: same happened in the past with Docker

Build System - Gulp

Gulp = popular build system developed in JS

Used for automation of time-consuming and repetitive tasks like code minification, concatenation, translation TS → JS, unit testing, linting, …

Based on node.js

Tasks defined in configuration "gulpfile.js"

TypeScript

TypeScript is a strongly typed superset of JS (ES6 and newer), which compiles to plain JS

Developed by Microsoft (www.typescriptlang.org)

TypeScript cannot be run directly in browser, but has to be translated to JS to run

Installation: npm install -g typescript

Translation with tsc integrated in build process (mvn, gradle, gulp)

Example TypeScript (Extension .ts)

```
interface Person { firstName: string; lastName:
string;}

function greeter(p: Person)

  { return "Hello " + p.firstName; }


let x = "Volker";

let y = greeter(x);   // produces error
```

Advantages

Supports static typing, strong typing

Supports classes and interfaces

Provides error-checking at compile time

Provides many (already existing) definition files for external JS libraries

→ Speeds up development time, since errors (especially wrong types) can be found earlier

Node.js

Open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side

Uses event-driven, non-blocking I/O model that makes it lightweight and efficient

Often used in combination with nginx as proxy (or other web server) and a NoSQL database

Website: https://nodejs.org/

First Simple Example (index.js)

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

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

server.listen(8080, '127.0.0.1', () => {
  console.log('Server running');  });
```

Often: “express.js” module used since much easier

Node.js Modules

Many pre-build modules exist for Node.js for direct use:

- HTTP, HTTPS, HTTP/2
- Crypto, DNS, File system, OS, I18N, URL, String decoder, …

See a complete list including APIs at nodejs.org/en/docs

Node.js as Event-Driven Tool

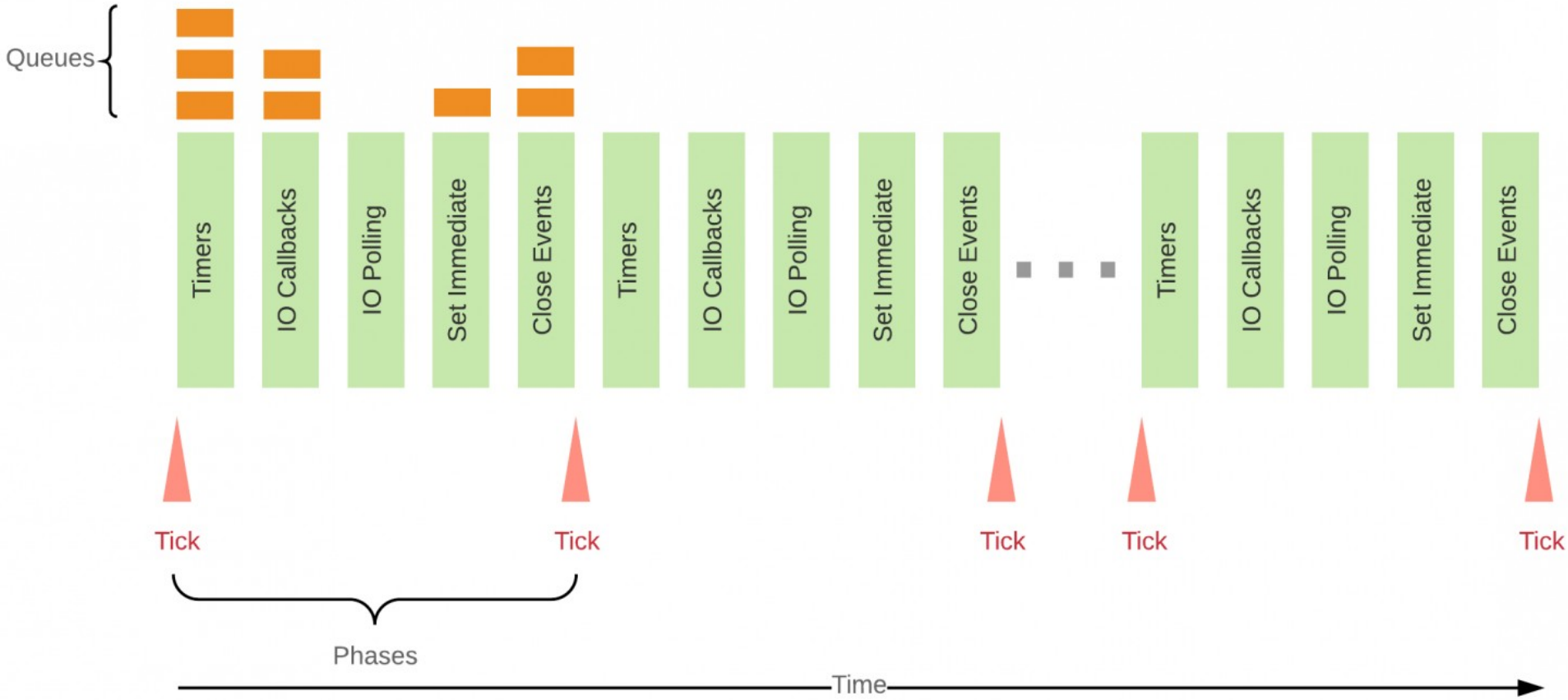
Event-driven = everything happens as reaction to an event

Events are treated in an "event loop", the main thread of Node.js

All tasks (must) have callbacks, executed after asynchronous part in event loop

Asynchronous work is whenever possible offloaded to non-blocking threads of OS

Event-Loop Visualized



Source: dynatrace.com

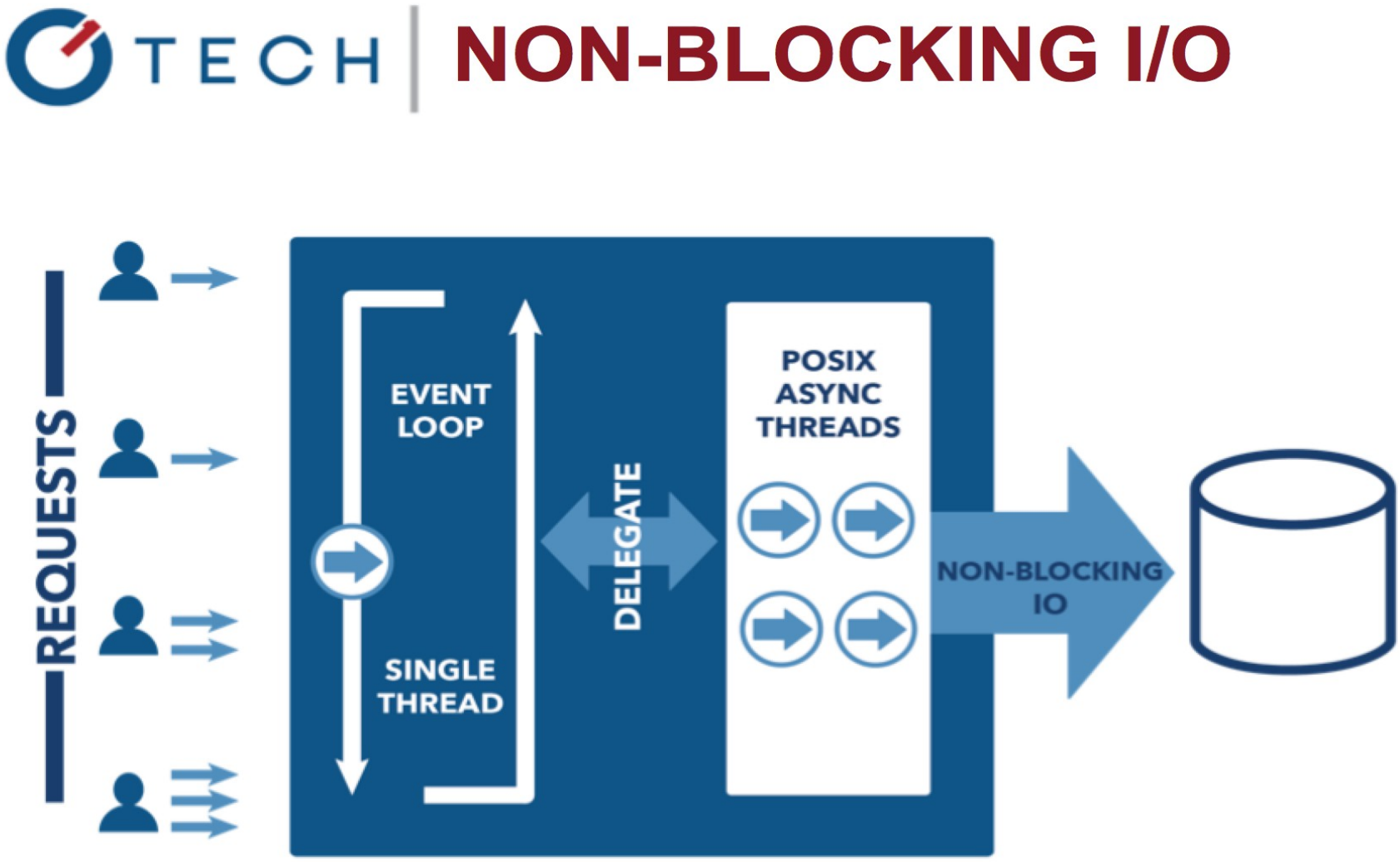
Node.js vs Traditional Web Servers

Traditional: each request spawns a new thread in userspace, taking up system RAM (→ number of threads limited)

Blocking IO syscalls used, since there are no callback functions to come back to

Node.js: only 4 (userspace) threads, delegates IO to non-blocking syscalls, which place callback into event-loop when finished

Non-Blocking IO



Source: think360studio.com

Advantages of Node.js

JS is well known programming language

Fullstack solution: JS on server and client

Supports many modern technologies like Web Sockets (to push data) or JSON (often used in NoSQL DB, Ajax) out-of-the box

Ideal for the data-intensive, real-time applications (due to non-blocking IO) – all applications with tasks that can be mostly delegated to the OS

Advantages of Node.js (cont)

Highly extensible: many JS modules available, easy to use with npm

Module caching integrated for better performance

Node.js applications can be easily scaled – either with a cluster (together with a proxy like nginx) or by hardware upgrade – see the provided Docker app

Disadvantages of Node.js

Node.js does not automatically work well for all web applications

Don’t use Node.js for CPU-intensive operations – slow compared to other web servers if tasks cannot be delegated to OS

Node.js must use callback functions for almost all operations, since an uncaught exception could stall the main event loop

"Practical comparison"

Be careful with such comparisons!

Task	NGINX - PHP 7.0.6	Node.js 6.11.2
1 mio string concats	54.6 ms	137.8 ms
1 mio int additions	51.8 ms	7 ms
1 mio filling array	52.4 ms	87.3 ms
100 mysql query	9.5 ms	18.4
1000 queries in 100 threads	32 ms per request	200 ms per request

Task	NGINX – PHP 7.0.22	Node.js 8.6.0
String concatenation	5310 req / sec	9703 req / sec
Int additions	6695 req / sec	12713 req / sec
Filling array	5992 req / sec	10602 req / sec

Sources: thinkmobiles.com/blog/php-vs-nodejs , grigorov.website/blog/performance-comparison-php-vs-node-js

Express.js

De facto standard web application framework for Node.js (expressjs.com)

Provides small and robust tooling for Node.js, including auto-generation, routing, redirection, caching, content negotiations, view support for template engines

```
app.get('/', function(req, res, next) {…… ; next( ) })
```

req, res = HTTP request/response, next = callback

Routing Variants Supported

Express.js supports routing with regular expressions or parameters:

```
app.get('/about', …
app.get('/ab?cd', …
app.get('/ab*cd', ….
app.get(/a/, ….
app.get('/users/:userId/books/:bookId', …
```

Example: Using MySQL

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

var con = mysql.createConnection({ host: "...", user: "...",
password: "...", database: "…" });

con.connect(  (err) => {

  if (err) throw err;

  con.query("SELECT * FROM customers",

          (err, result, fields) =>

              { if (err) throw err; console.log(result); …. });

});
```

Ex.: Web Sockets (Server side)

```
const WebSocket = require('ws')

const wss = new WebSocket.Server({ port: 8080 })

wss.on('connection', ws => {

    ws.on('message', message => {

     console.log(`Received message => ${message}`)

    })

ws.send('Message from server!')  })
```

Web Services

Software system designed to support interoperable machine-to-machine interaction over a network

Often working over HTTP (port 80) or HTTPS (443)

Clients and servers exchange messages in XML format (SOAP) or some other text-based format (JSON)

Now: RESTful WS with Node.js

Web Services: Where Can Request Data Be Encoded?

SOAP WS transports request in message body, encoded in XML

HTTP messages have headers and a body

HTTP also defines request methods → could be used as synonym for name of “remote procedure”

Method could be encoded as part of URL

(Small) parameters could also be encoded in URL

“Request Format” for RESTful WS

RESTful WS: method and parameters encoded in access URL or HTTP headers, format of response defined by server

Only five methods available: create (PUT), read (GET), update (POST), delete (DELETE), HEAD

Example: http://localhost/user/vmueller with GET

Read information for user with name “vmueller”

“Response Format”

Response included in body of HTTP response

Format of response can be determined arbitrarily by server, but very often fixed as JSON

Common practice:

- Client defines requested format as MIME-type in HTTP header “Accept”
- Server encodes response in that format (if supported)

Example: RESTful WS with Node.js

RESTful WS can be easily implemented with Express.js

Simple Docker-based example provided on Moodle which reads data from a file (instead of a DB), also does no response type negotiation, only returns JSON data

Example: Node.js + Redis

Node.js also often used with NoSQL DB

On Moodle: Docker application with

- Redis as NoSQL DB
- 3 instances of Node.js servers
- NGINX as proxy → forwards requests to Node.js instances with Round Robin

Conclusion on Node.js

Node.js is a hot topic with many demands on the market

Very efficient in certain scenarios, but not always the best choice

Often a combination of a traditional web server (nginx) acting as proxy and Node.js used, where some specific requests are proxied to Node.js

Very often used in combination with JS frameworks

Excursus: PHP Swoole (www.swoole.co.uk)

PHP Swoole is asynchronous programming framework for PHP

Allows developers to write asynchronous code for PHP

Event-driven, asynchronous, non-blocking IO → easy scalable

Implemented as PHP extension, but requires support from web server software

→ Bring some ideas of Node.js to PHP

PHP Swoole Example

```
<?php
$http = new swoole_http_server("127.0.0.1", 9501);
$http->on("start", function ($server)
{   echo "Http server started \n";   });
$http->on("request", function ($request, $response) {
    $response->header("Content-Type", "text/plain");
    $response->end("Hello World\n");
});
$http->start(); ?>
```

Next Week

Javascript Frameworks

Single-Page Applications