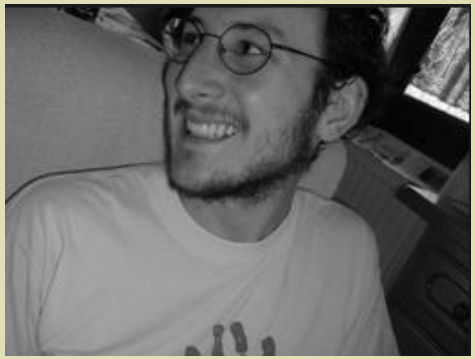
Node.js

An Introduction

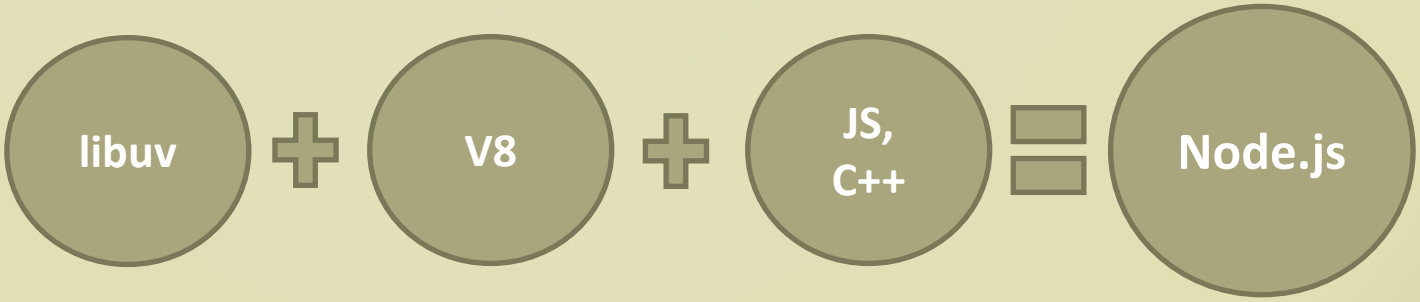
Outline

- Node.js background, advantages and limits
- Node event loop
- Single vs. Multi Threaded approach
- Synchronous vs. Asynchronous Code
- Callback Function and Events
- Modules and NPM
- Exciting Tasks

Background



- Introduced by founder **Ryan Dahl** in 2009
- Node.js is a server side web framework written in java script
- Aimed for real-time webites with push capability



What is the Advantage?

- Node.js is ideal for real-time web applications employing **push technology** over websockets
- Node allows easily building **fast, scalable network applications**
- Node.js uses an **event-driven, non-blocking I/O model** that makes it lightweight and efficient
- Node.js is perfect for **data-intensive real-time applications** that run across distributed devices.

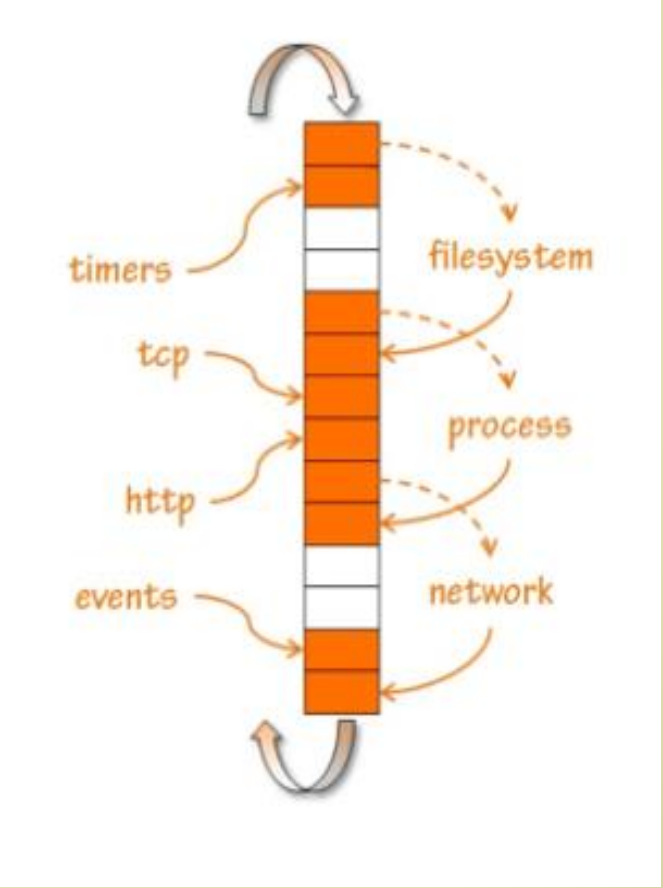
Where does Node.js not shine?

Node.js is a platform that fills a particular need, however there are examples where Node.js is the wrong choice

- You definitively do not want to use Node.js for **CPU-intensive operations**
- Using libraries with **blocking methods** does not work for Node.js

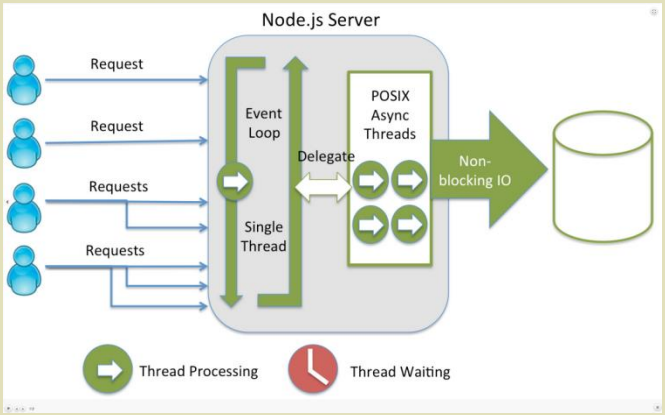
Node's Event Loop

- Event loop is one of the key concept of node for the non-blocking approach
- Node's event loop is constantly listening on events on the server side (e.g. File system, data base, ...)

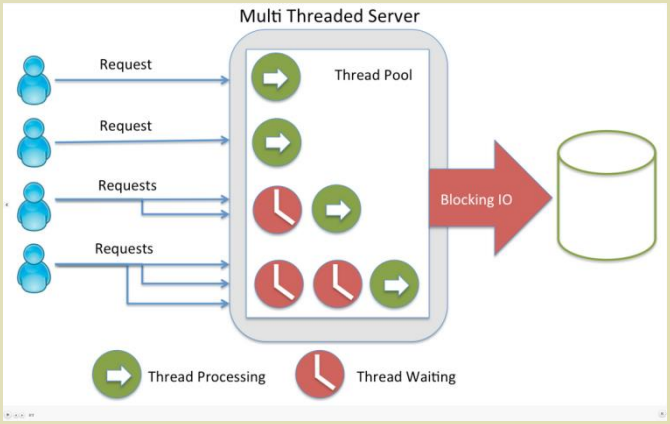


Single- vs. Multi Threaded Approach

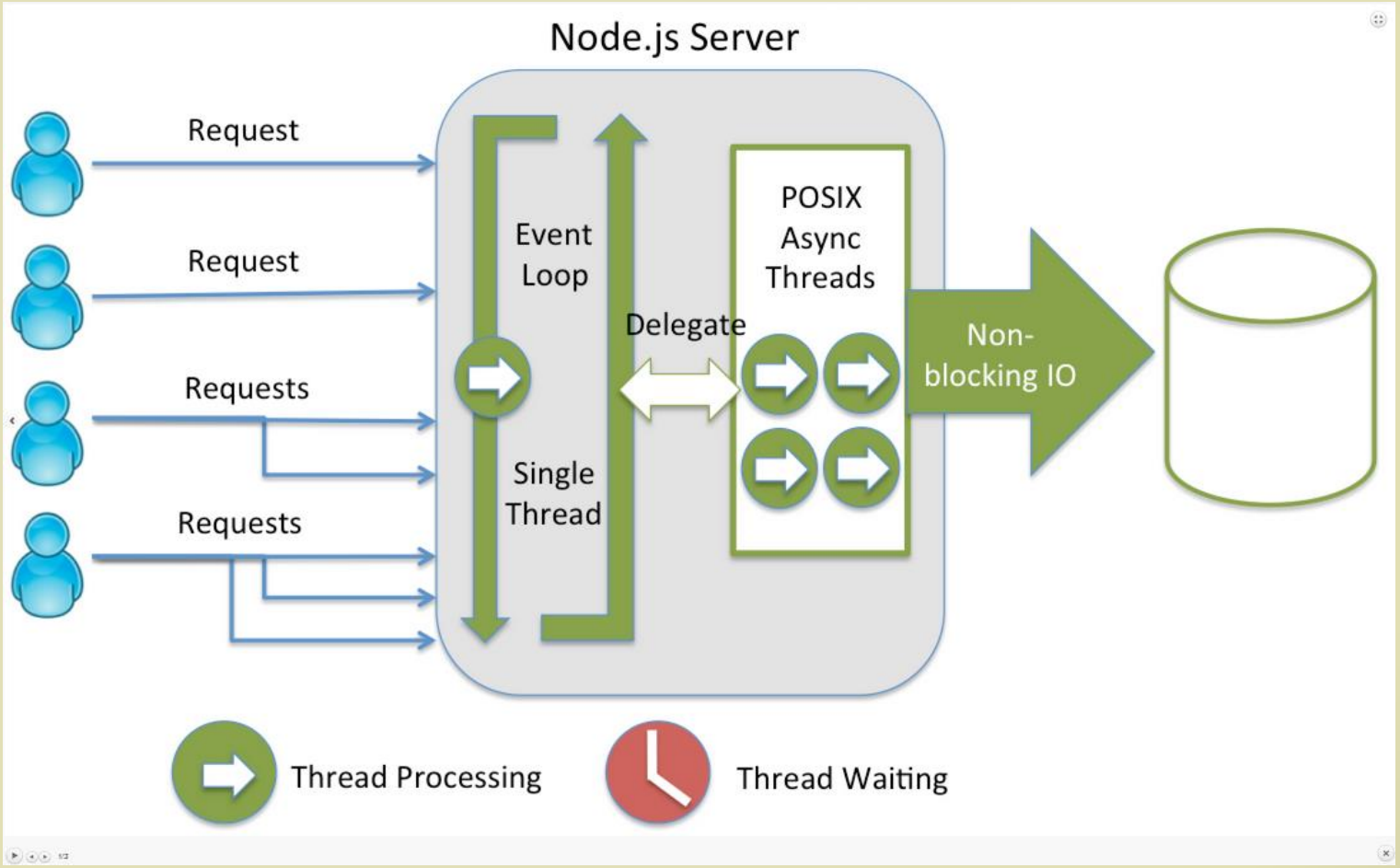
Single Thread Approach



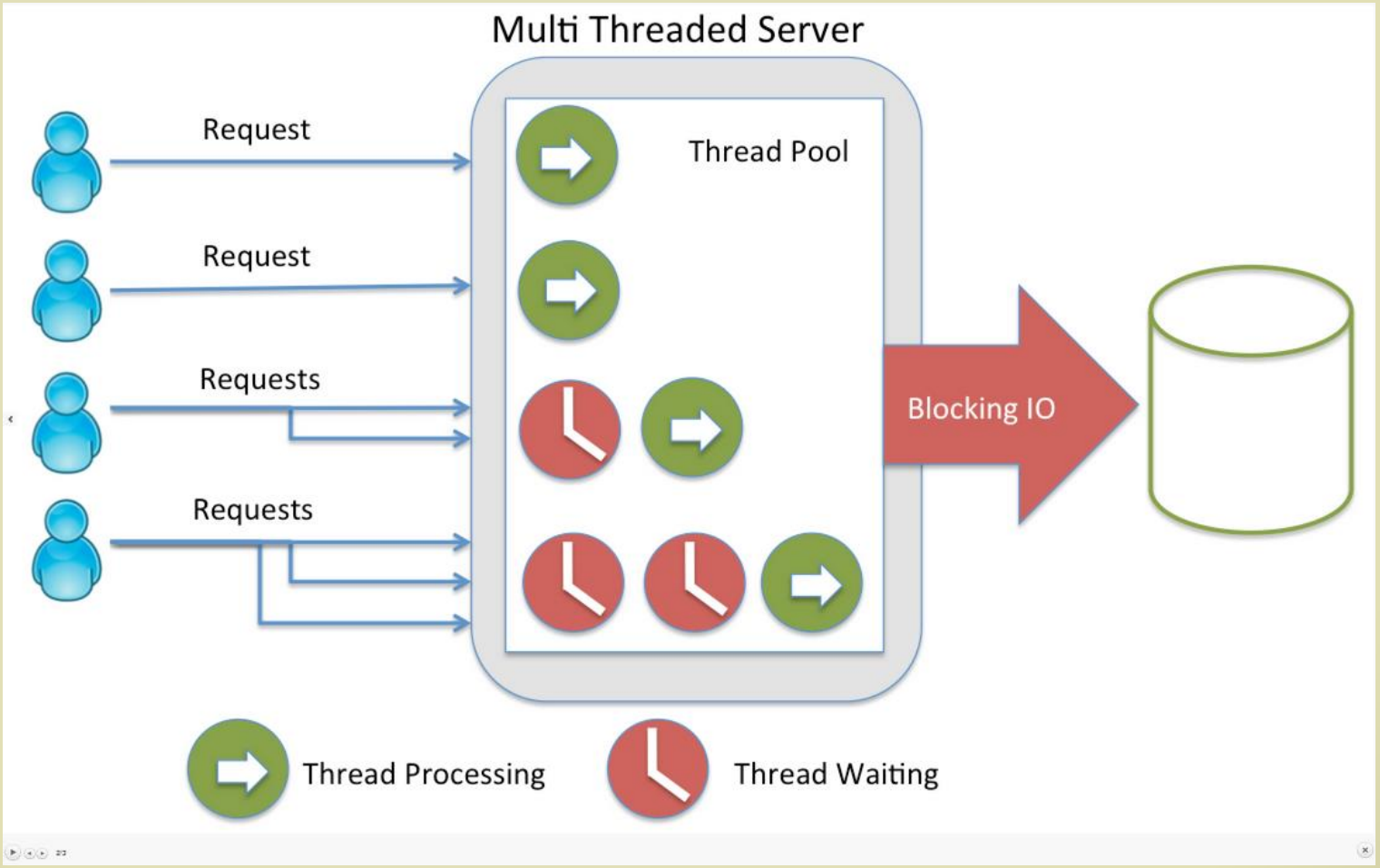
Multi Threading Approach



Single Threaded Approach



Multi Threaded approach



Synchronous vs asynchronous Code

Sync

```
Var conn = getDbConnection(connString);
Var stmt = conn.createStatement();
Var results = stmt.executeQuery(sqlQuery);
For (var i=0; i < results.length; i++){
  console.log(i);}
```

Async

```
getDbConnection(connectionString, function(err, conn){
  conn.createStatement(function(err, stmt){
    var results = stmt.executeQuery(sqlQuery);
    results.on('row', function(result){
      console.log(...)
    });
  });
});
```

Callback Functions

```
// Executing method
for(int i=0; i<10; i++){
  someFunction(i, callbackFunction);
}

Var callbackFunction = function(err, result){
  if(err){console.log("ERROR!");}
  else(console.log("result is: " + result);
};


var someFunction = function(inputParam, callback){
    var delayTime = Math.floor(Math.random*1000);
    if(inputParam == 0){
       setTimeout(function(){callback(new Error("..."))},
                  delayTime);}
    else{
       setTimeout(function(){callback(null, result * result)},
                   delayTime)};}
};
```

Callback vs Events

Callbacks

```
getResults(param,
   function(err, results){
   // chech for erros
   // process results
});
```

Events

```
var results = getResults(param);

results.on('item', function(i){
   // do something
});

results.on('done', function(){
   // no more items
});

results.on('error', function(err){
   // handle error
});
```

- Request / Reply
- No results until all results
- Either error or results

- Publish / Subscribe
- Act on results as they arrive
- Partial results before error

Modules & NPM

There are three different types of modules:

1) Node.js built in modules

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

2) Your Project files

each .js file is its own module and is a great way to modularize your application's code

```
module.exports.someFunction = someFunction;  // function exportet in mod1.js
var mod1 = require(./mod1');
```

3) Third party modules

...to be imported with the Node Package Manager (NPM) registry

visit npmjs.org

Popular NPM Modules

Module	Description
express	a Sinatra-inspired web development framework for Node.js
socket.io	Server-side component of the most common websockets components out there today
mongo	MongoDB wrappers to provide the API for MongoDB object databases in Node.js
forever	Probably the most common utility for ensuring that a given node script runs continuously. Keeps your Node.js process up in production in the face of any unexpected failures

Useful Links

- Official node.js and npmjs webpages
  http://nodejs.org/
  http://nodejs.org/api/
  https://www.npmjs.org/

- Interactive samples
  http://nodeschool.io/

- Ryan Dahl Intro (1h)
  https://www.youtube.com/watch?v=jo_B4LTHi3I
  https://www.youtube.com/watch?v=dO1zf4RXsTg

Get started

1) Download the Node.js Framework from http://node.js.org and install it on your computer

2) Create a HelloWorld folder and a file called HelloWorld.js in it, write the following line into the file and save it

```
Console.Log(«Hello World»);
```

3) Do Shift + Right Mouse Click on the HelloWorld folder and open the command prompt

4) > node HelloWorld.js

Task 1: HelloWorld.js

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

http.createServer(function (request, response) {
        response.writeHead(200, {'Content-Type':    'text/plain'});
response.end('Hello World\n'); }).listen(5000);

console.log('Server running at http://127.0.0.1:5000/');
```

> node example.js
Server running at http://127.0.0.1:5000/

Task 2: Http - Get

Write a program that performas an HTTP GET request on http://ww.google.ch and write the data from the server to the console

1) Use get method of http class
2) Add the necessary method parameter/function
3) Set the corresponding encoding
4) Write the response to the console

Task 3: FileStream

Write a program that starts a web server listening on the port provided the applications argument and return the content of a text file as response to the browser

1) Read data from file and write it to the response
2) Listen on the port provided with the application argument with the ‘process’ object
3) Start the application with
   >node fileStream.js 5000
   and open a web browser on http://localhost

Socket.IO

Server:

```
var io = require('socket.io').listen(80);

io.sockets.on('connection'),
function(socket){
    socket.emit('news', {hello:'world'});

socket.on('other event', function(data){
     console.log(data);
     });
});
```

Browser:

```
<script src="/socket.io/socket.io.js">
</script>

<script>
  var socket = io.connect('http://localhost');

Socket.on('news', function(data){
     console.log(data);
     socket.emit('other event', {my: 'data'});
});
</script>
```

Task 4: Socket.IO

Write a program that continuously pushes the current time stamp to the browser with the real time capable socket.io module

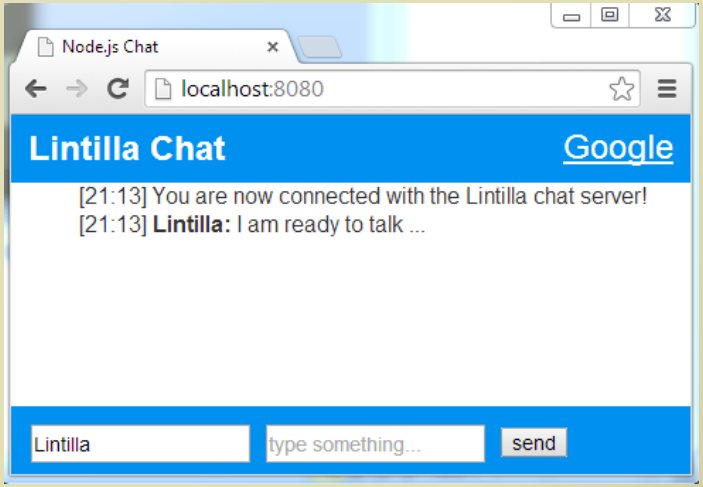
1) Create a function that returns the /index.html file
2) Create a http server and pass the function created in step 1)
3) Implement the socket.io server-side method
4) Implement the socket.io client-side method

Task 5: Chat Room

Write a single chatroom where people come and can exchange messages in one-to-many (all) fashion

On the **server-side** is a simple express.js application which implementes a **GET '/' request handler** which serves the webpage with the index.html file and a **websocket server** that listens for new messages emitted by the websockets clients

On the **client-side** we have an **HTML page** with a couple of handlers set up, one that picks up the send button click event and sends the message to the server and another that listens for new **incoming messages** sent by other users



Homework

1) If you have time, watch a Ryan Dahl video on youtube just to get the Node idea at first hand

2) Wrap up all the tasks in a silent minute. If there are questions I try to help you further

3) For the Chat Room applications think of how one of the following enhancements could be done

   -> Authentification to the chatroom
   -> storing the chat history in order you can continue your conversation after closing and reopening the browser (e.g. With mongodb database)