Q

What are, exactly, arrays in Javascript?

Arrays

```
a=[];
a = new Array(); //discouraged
a[0]=3; a[1]=“hello”; a[10]=new Rectangle(2,2);
a.length() => 11;

a[“name”]=“Jaric”; ⇔  a.name="Jaric";
```

Arrays are

SPARSE, INHOMOGENEOUS , ASSOCIATIVE

See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Arrays

Array give you functions to (e.g.):

- add/remove an element at the end
- add/remove an element at the front
- add/remove an element by index
- remove a number of elements starting from an index
- find the index of an element
- make a copy of an array

`See` https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Arrays

See also a set of examples of common tasks performed on array (such as e.g. find the maximum, sort):

- https://www.w3schools.com/js/js_array_sort.asp
- https://www.w3schools.com/js/js_array_iteration.asp

Q

How does the + operator work on objects?

+ Operator with objects

We already know that the first rule with the + operator is to convert objects to primitive values.
The rule to execute the conversion is:

Check if the object:

- is not a Date AND
- has a valueOf() method AND
- its valueOf() method returns a primitive value.

If yes, use the valueOf() method.

Else, use the toString() method.

Note: the array [1,"a",2] would be converted to "**1,a,2**".
An empty object {} is converted to **"[object Object]**"

Quick intro to Node.js

Q

Can JavaScript be used server-side?

Server-Side JavaScript

A substitute for CGI.

Server-dependent technology to process the Web page ***before*** passing it to the client.

An approach which started long ago (Netscape SSJS)

Then mostly forgotten, later revived by Rhino (a bridge between JS and Java) and more recently by **Node.js**

Node.js

an open-source, cross-platform JavaScript engine:
not a framework or a library, but a run-time environment based on Chrome's V8 JavaScript engine for executing JavaScript code server-side.

- **event-driven architecture**
- **asynchronous I/O**

Optimizes throughput and scalability
- in Web applications with many input/output operations,
- for real-time Web applications

https://www.w3schools.com/nodejs/default.asp

Node.js

1. The official Node.js website has installation instructions for Node.js:
   https://nodejs.org

1. Download and install.

2. Create a file called "node hello.js"

3. execute "node hello.js"

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

http.createServer(
   function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.end('Hello World!');
   }
).listen(8080);
```

Node.js: built-in modules

Module	Description
cluster	To split a single Node process into multiple processes
crypto	To handle OpenSSL cryptographic functions
events	To handle events
fs	To handle the file system
http	To make Node.js act as an HTTP server
https	To make Node.js act as an HTTPS server.
os	Provides information about the operation system
path	To handle file paths
querystring	To handle URL query strings
timers	To execute a function after a given number of milliseconds
url	To parse URL strings
util	To access utility functions
zlib	To compress or decompress files

see https://www.w3schools.com/nodejs/ref_modules.asp for a full list

Node.js: modules

Create your own modules – save the following in "myModule.js

```
exports.myDateTime = function () {
  return Date();
};
```

Use your own modules

```
var dt = require('./myModule');
```

Obtain modules from the cloud

```
npm install upper-case
```

Use your own modules

```
var dt = require('upper-case');
```

Express.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

see https://expressjs.com/en/starter/hello-world.html

cookie-session

```
var cookieSession = require('cookie-session')
var express = require('express')
var app = express()
app.use(
   cookieSession(
        { name: 'session',
          keys: [/* secret keys */],
          // Cookie Options
          maxAge: 24 * 60 * 60 * 1000 // 24 hours
        }
   )
)
```

see http://expressjs.com/en/resources/middleware/cookie-session.html

Other Node.js-based frameworks

Meteor.js
Sails.js
Koa.js
Keystone.js
Loopback.js

Node.js stories

Nexflix used JavaScript and NodeJS to transform their website into a single page application.

Traditionally, Netflix has been an enterprise Java shop, but “as we migrated out of the data center to the cloud we moved to a more service-based architecture,” Trott said.

Java still powers the backend of Netflix, but all the stuff that the user sees comes from Node.

Q

What is non-blocking I/O, and why is it relevant ?

I/O

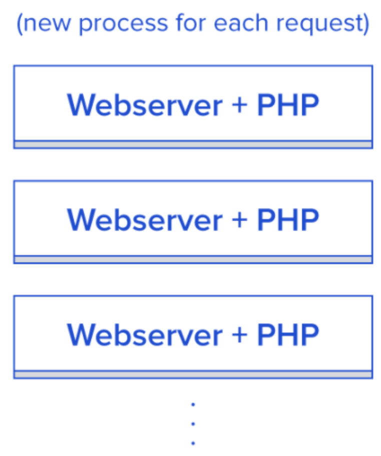
*A **blocking IO** means:*
*a given thread cannot do anything more until the **IO** is fully received (in the case of sockets this wait could be a long time).*

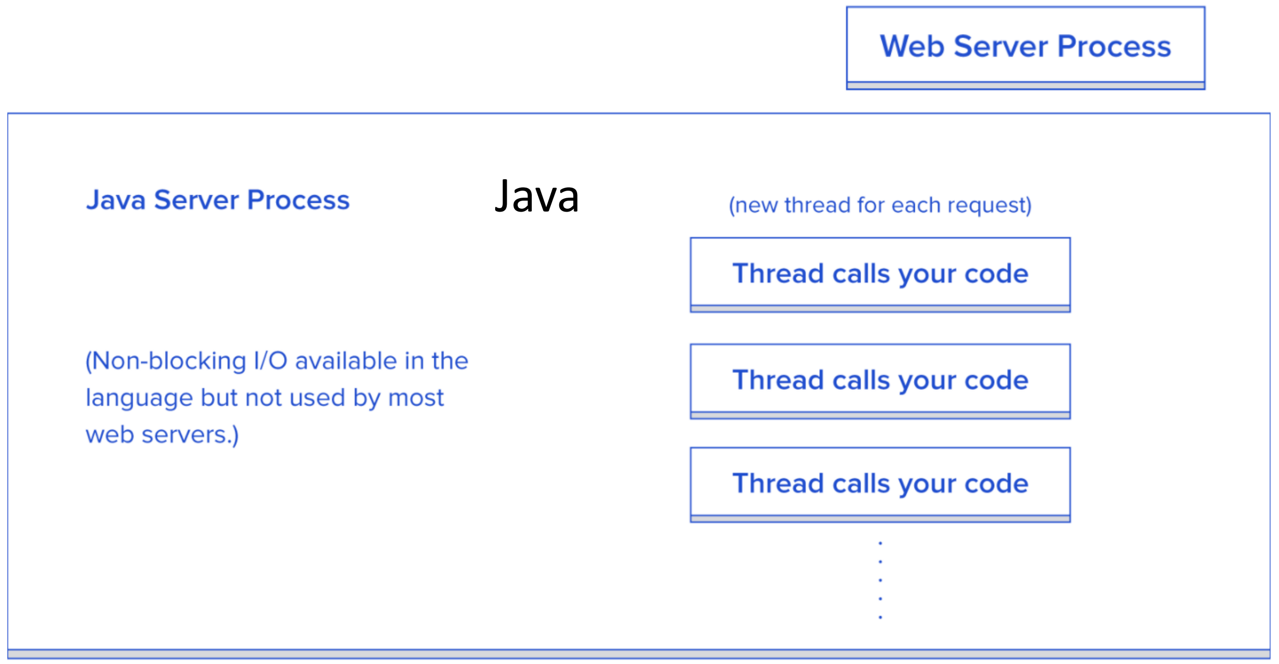
*Non-**blocking IO** means:*
*an **IO** request is queued straight away and the function returns. The actual **IO** is then processed at some later point by the kernel*

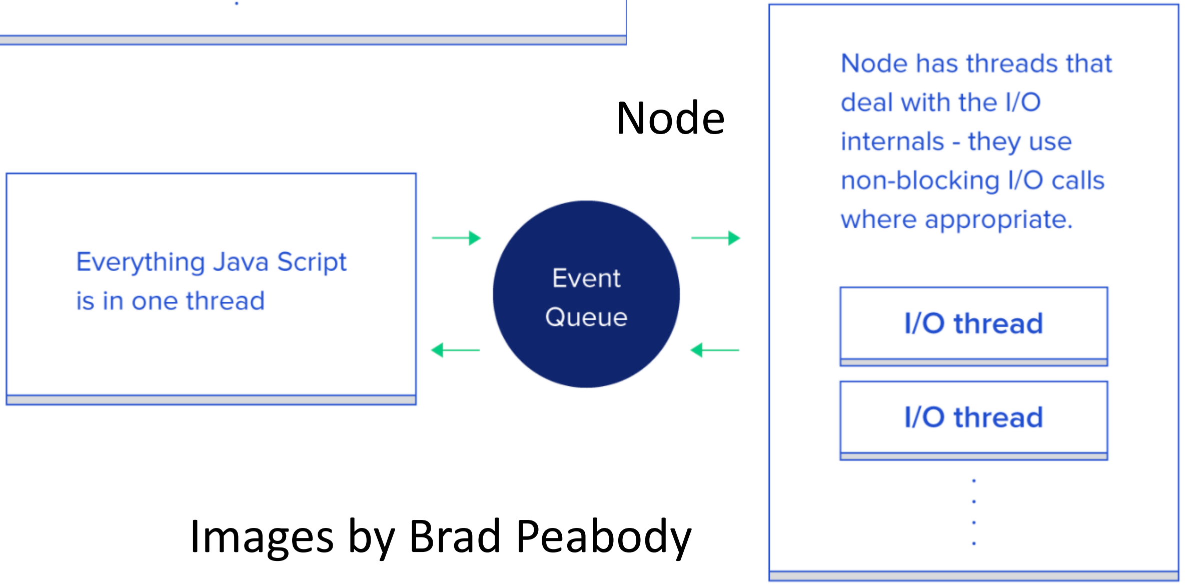
Q

What is the Node.js architecture compared with JavaEE and PHP?

The Node.js model




Web Server Process
Java Server Process
Java
(new thread for each request)
Thread calls your code
Thread calls your code
Thread calls your code
(Non-blocking I/O available in the language but not used by most web servers.)


Node
Everything Java Script is in one thread
Event Queue
Node has threads that deal with the I/O internals - they use non-blocking I/O calls where appropriate.
I/O thread
I/O thread

Images by Brad Peabody

Q

What is best, Node.js or JavaEE?

Node.js vs JavaEE

- *JavaScript has non-blocking I/O*
- *Java I/O is traditionally blocking*
  - *(but NIO introduced non blocking I/O)*
- *Javascript promises*
- *Java CompletableFuture*
  - *(Netty, Undertow web servers)*
- *Java is multithreaded*
- *JavaScript is single threaded*
  - *(but WorkerThreads and ThreadPools have arrived)*

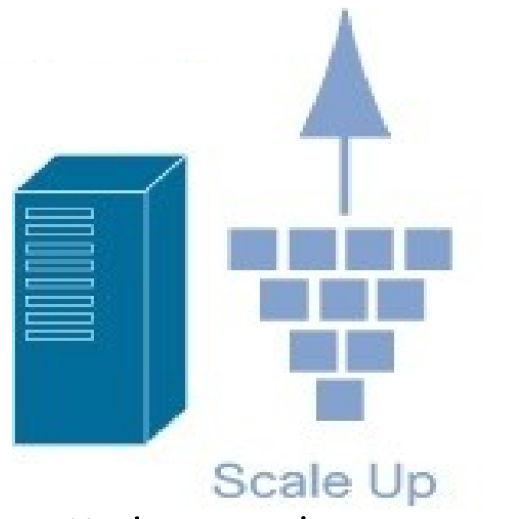
Vertical scaling vs. Horizontal scaling

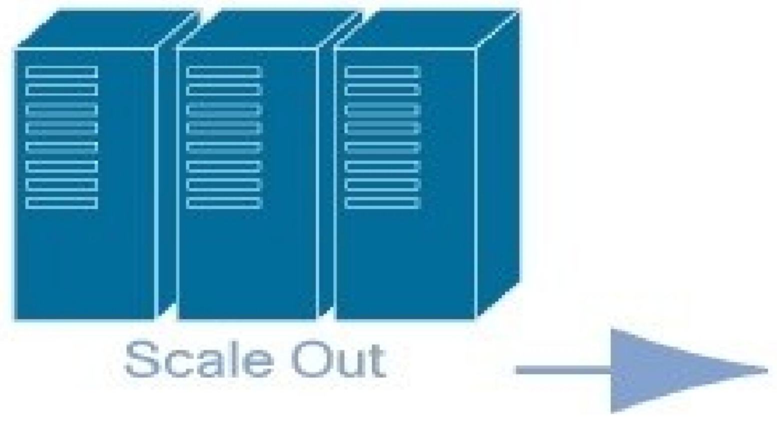
Vertical Scaling
Increases the power of existing system by adding more powerful hardware.

Issues:
- Additional Investment
- Single point of failure (SPOF)

Horizontal Scaling
Adds extra identical boxes to server.

Issues:
- Requires Load balancer for managing connection.
- Distribution of work within the units becomes overhead.
- Additional investment.





from Ratan Kadam et al.

Java vs Node.js

- **Scalability**: Java applications that can be scaled both vertically and horizontally in comparison to horizontal scalability of Node.js.
- **Language advantages:**
  - Node allows using a single language for front-end and back-end (full stack)
  - JS is simple to learn (?) but asynchronous programming may be quite unfamiliar
  - Java has better IDEs
- **Libraries**: They both offer a wide variety of tools and libraries, although Java libraries are more mature and solid. npm has vulnerabilities!
- **Architecture**: Node.js is well suited to microservices, whereas Java EE mainly focuses on the delivery of a monolithic web application.
- **Security:** JavaEE offers a set of built-in security features, Node.js development relies mostly on customized solutions.
- **DB Access**: RDB is underdeveloped for Node (which is better suited to NoSQL DBs, like MongoDB, CouchDB…)

Java vs Node.js: summing up

- Both technologies are an efficient solution that was successfully implemented in a number of projects.
- Applications that depends on a lot of I/O (FinTech, booking systems, media apps, etc.) may be better in Node.js.
- Java is better if you do much computing (IoT, ecommerce platforms, Big Data)