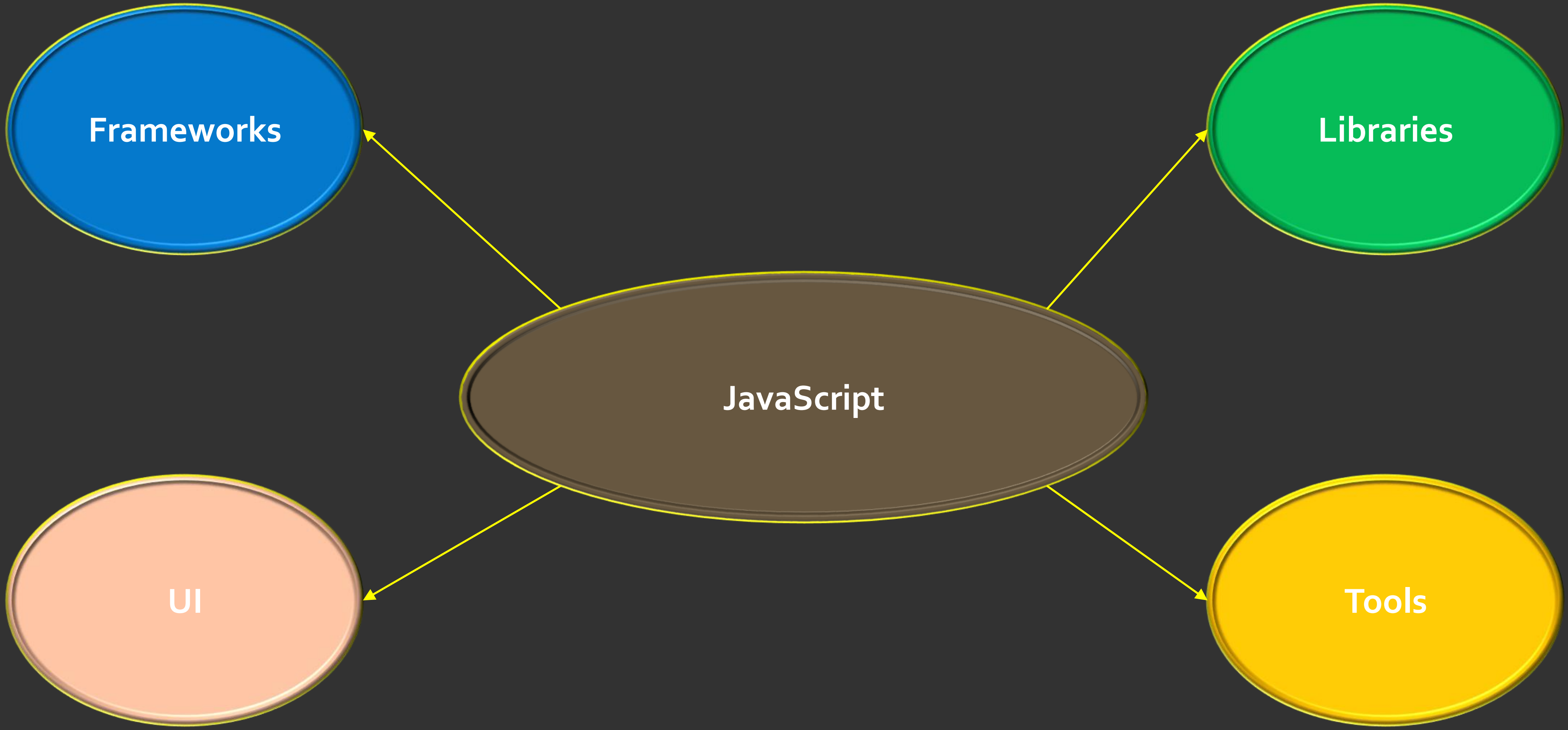
CONSUMER ORIENTED APPLICATIONS

- Apps designed for
  - All devices
  - All Browsers
  - Partner App Integration
  - Consumer App Integration
  - Client-Side Processing Capabilities
  - Security

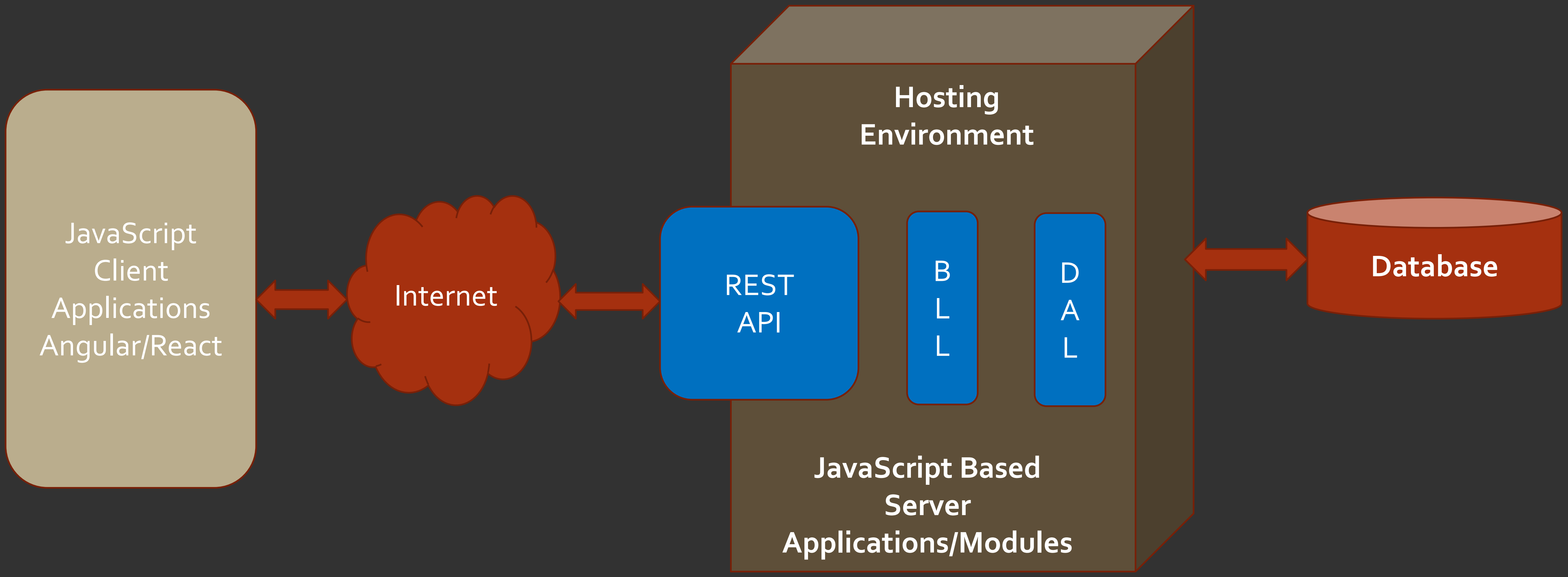
WORLD OF JAVASCRIPT



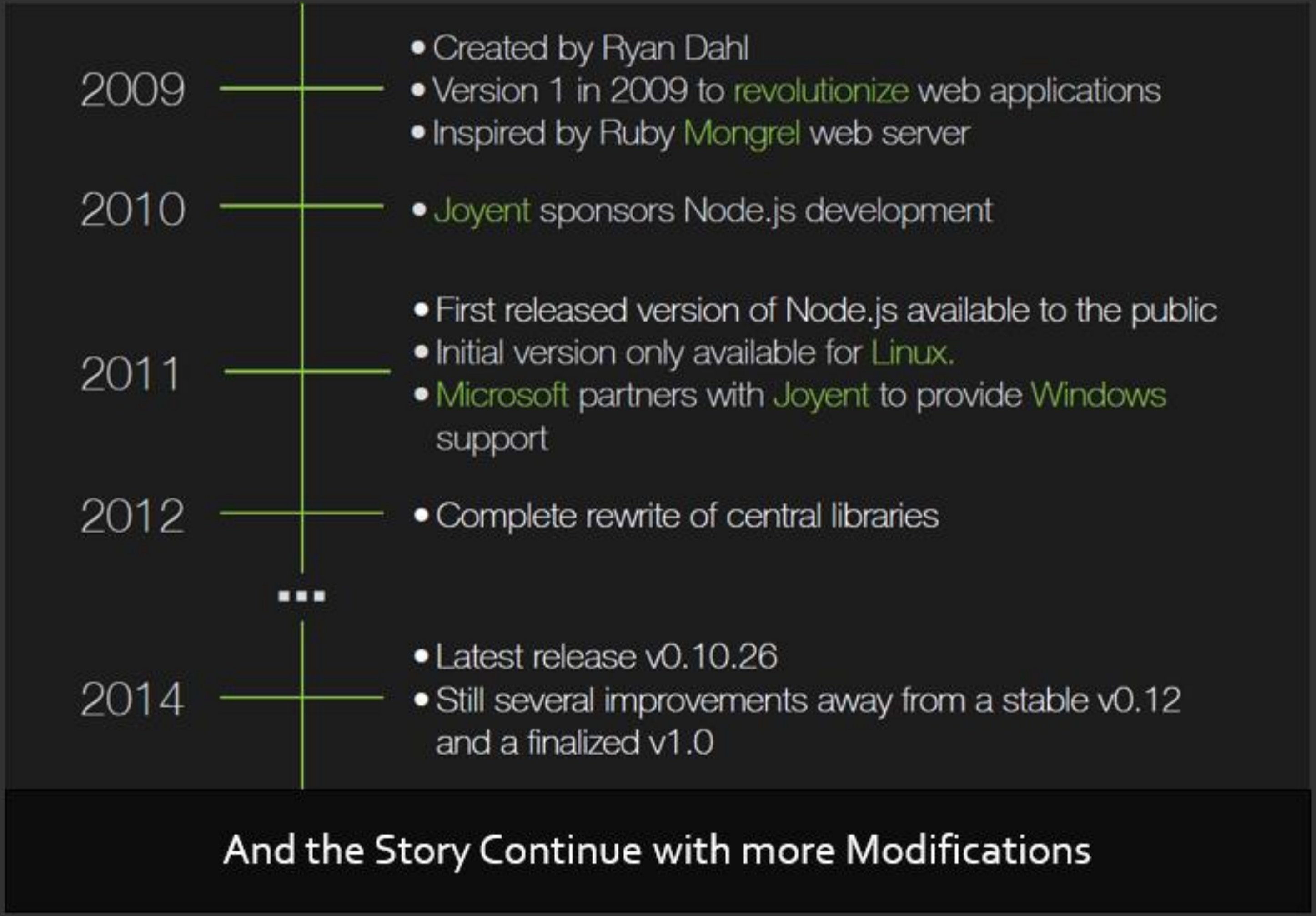
POWER OF JAVASCRIPT

- Improved Client-Side Capabilities
- Availability in the form of
  - Libraries
  - Frameworks
  - Components
- More Expectations
  - Isomorphic Applications.
  - Modern Web applications.
- A World of Isomorphic Applications
  - Using JavaScript on a web server as well as the browser reduces the impedance mismatch between the two programming environments which can communicate data structures via JSON that work the same on both sides of the equation. Duplicate form validation code can be shared between server and client, etc.

ISOMORPHIC APPLICATIONS


JavaScript Client Applications Angular/React
Internet
Hosting Environment
REST API
B L L
D A L
JavaScript Based Server Applications/Modules
Database

NODE.JS



2009
- Created by Ryan Dahl
- Version 1 in 2009 to revolutionize web applications
- Inspired by Ruby Mongrel web server

2010
- Joyent sponsors Node.js development

2011
- First released version of Node.js available to the public
- Initial version only available for Linux.
- Microsoft partners with Joyent to provide Windows support

2012
- Complete rewrite of central libraries

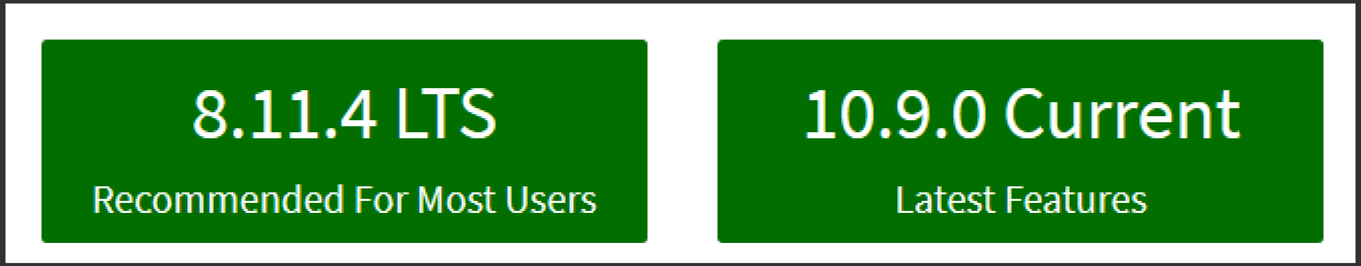
...

2014
- Latest release v0.10.26
- Still several improvements away from a stable v0.12 and a finalized v1.0

And the Story Continue with more Modifications

NODE.JS

- Open Source
- Extensively dependent on the JavaScript
- Uses Asynchronous Capabilities of JavaScript
  - Promises
- Uses Event Based Queuing defined by V8 Engine
  - Engine written for Chrome or Chromium Platform.

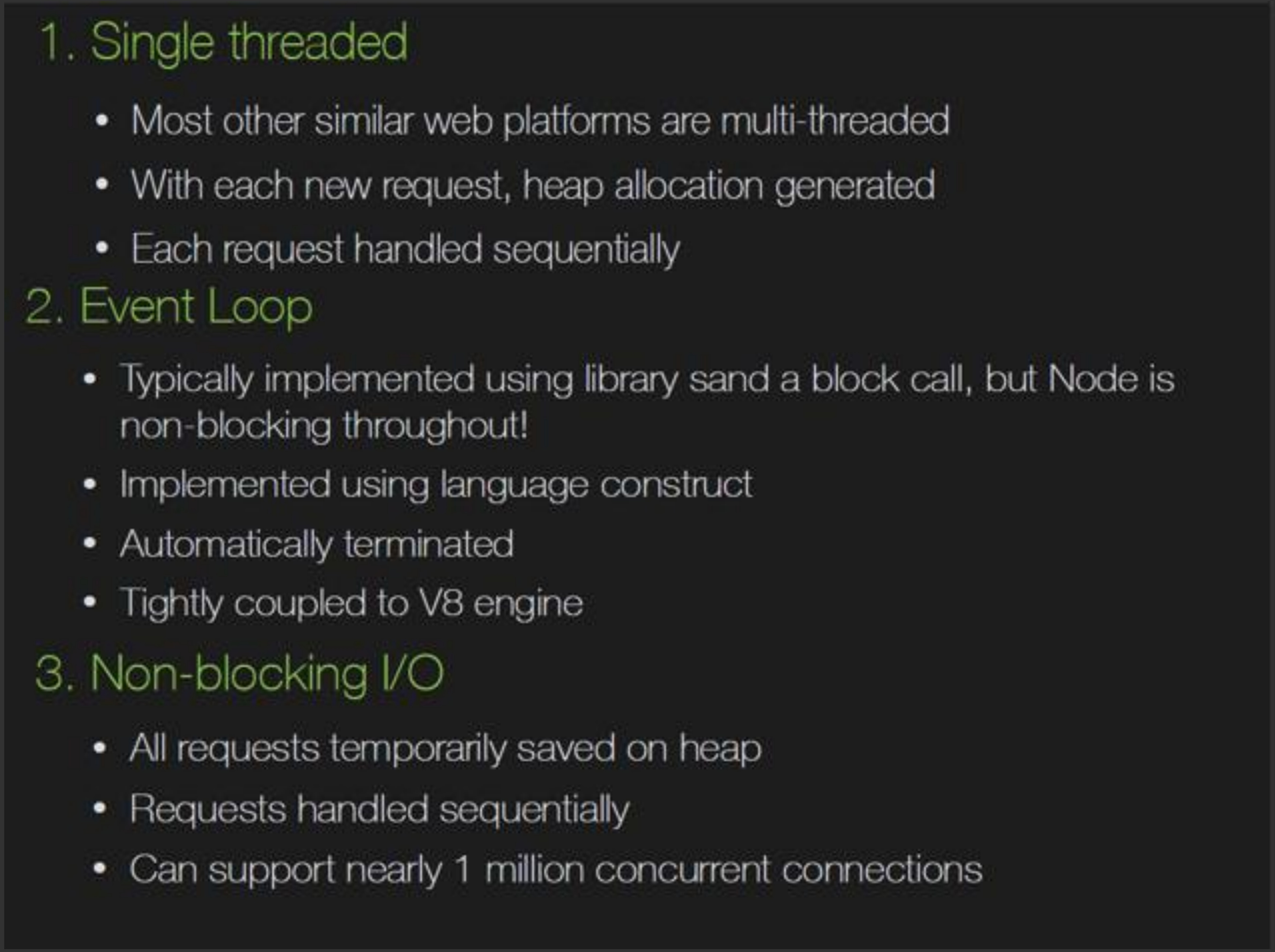


Long Term Support

WHAT IS NODE.JS ?

- Evented I/O for JavaScript.
- Server Side JavaScript.
- Runs on Google's V8 JavaScript Engine.
- Server-Side Solution for JS.
- Runs over the command line.
- Designed for high concurrency
  - Without threads or new processes
  - Perfect for data-intensive real time applications running across distributed devices.

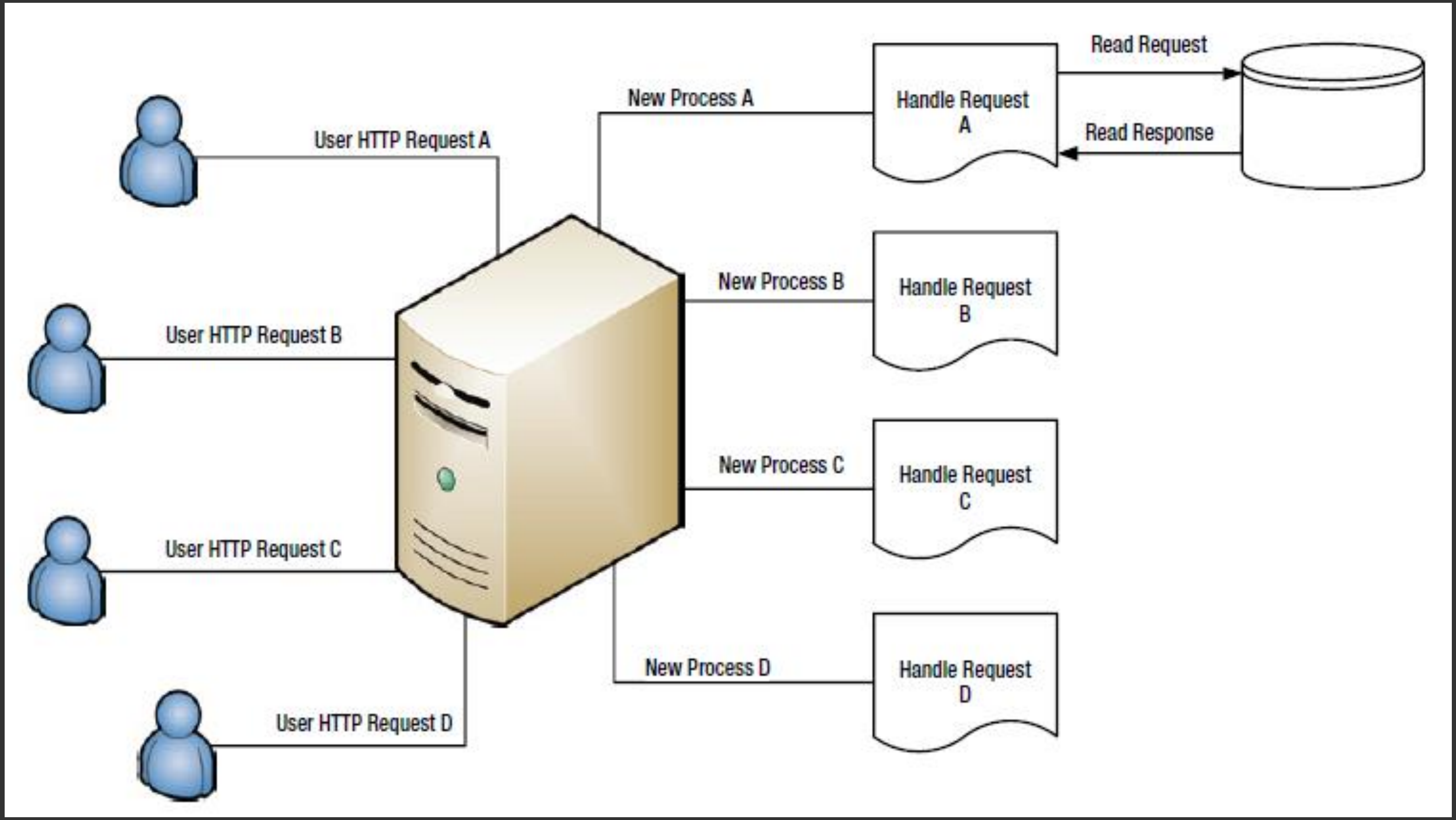
NODE JS



1. Single threaded
   - Most other similar web platforms are multi-threaded
   - With each new request, heap allocation generated
   - Each request handled sequentially
2. Event Loop
   - Typically implemented using library sand a block call, but Node is non-blocking throughout!
   - Implemented using language construct
   - Automatically terminated
   - Tightly coupled to V8 engine
3. Non-blocking I/O
   - All requests temporarily saved on heap
   - Requests handled sequentially
   - Can support nearly 1 million concurrent connections

WHY USE NODE.JS

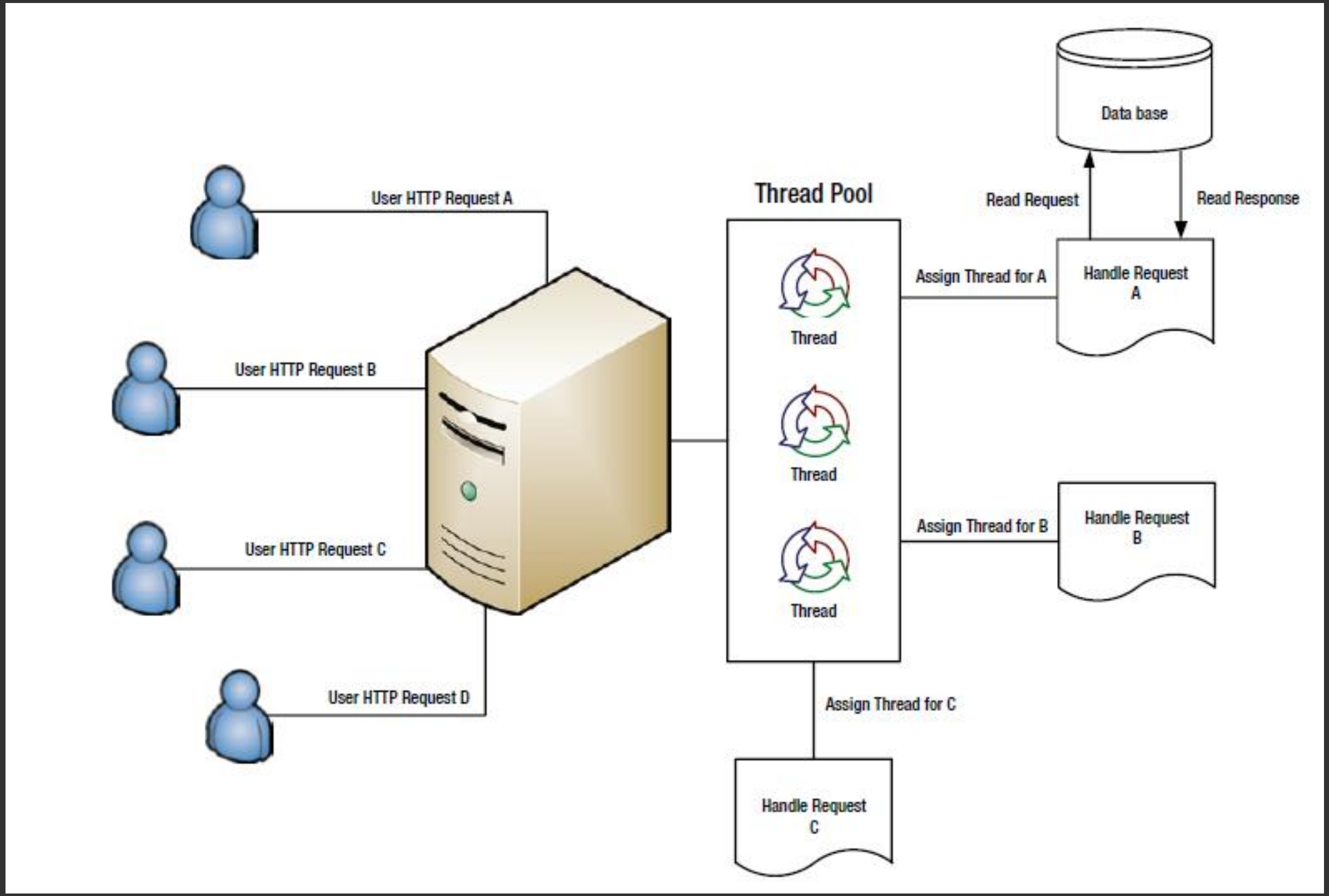
- Traditional Web Servers Using a Process Per Request



References: Beginning Node.js by Apress

WHY USE NODE.JS

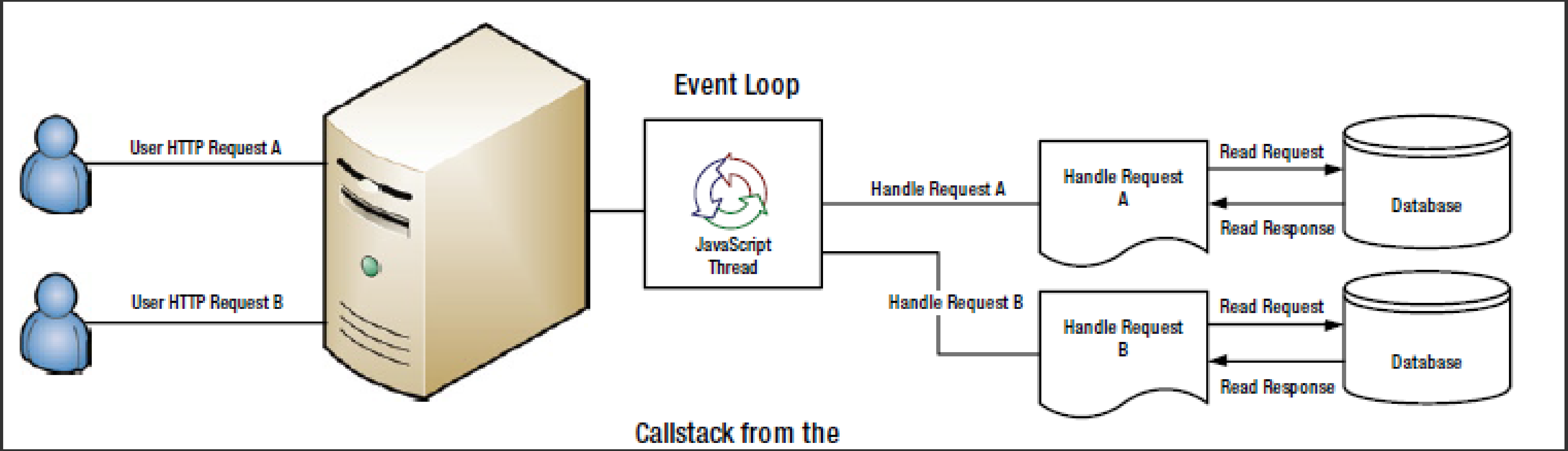
- Traditional Web Servers Using a Thread Pool



References: Beginning Node.js by Apress

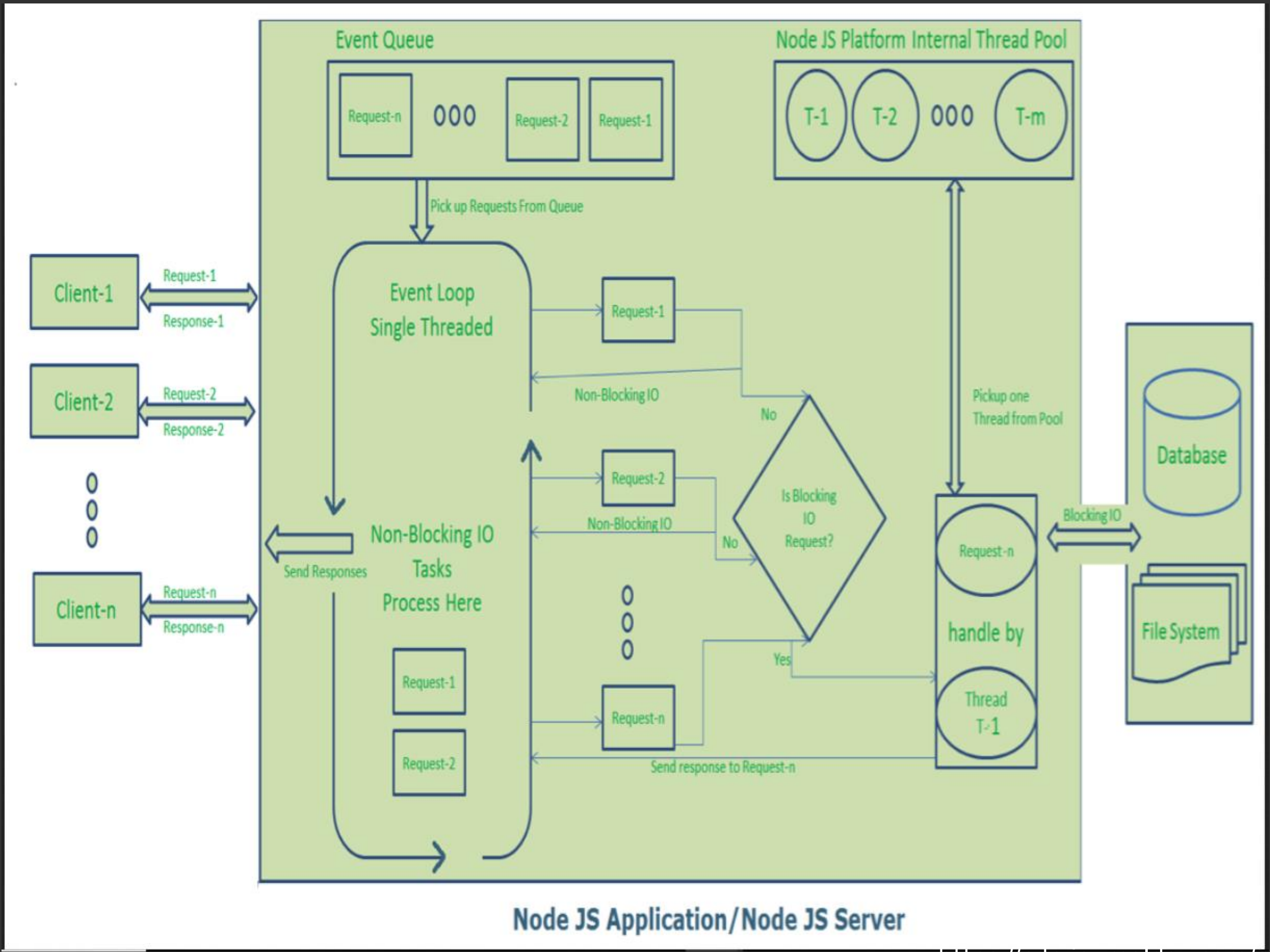
WHY TO USE NODE.JS

- Node.js event loop



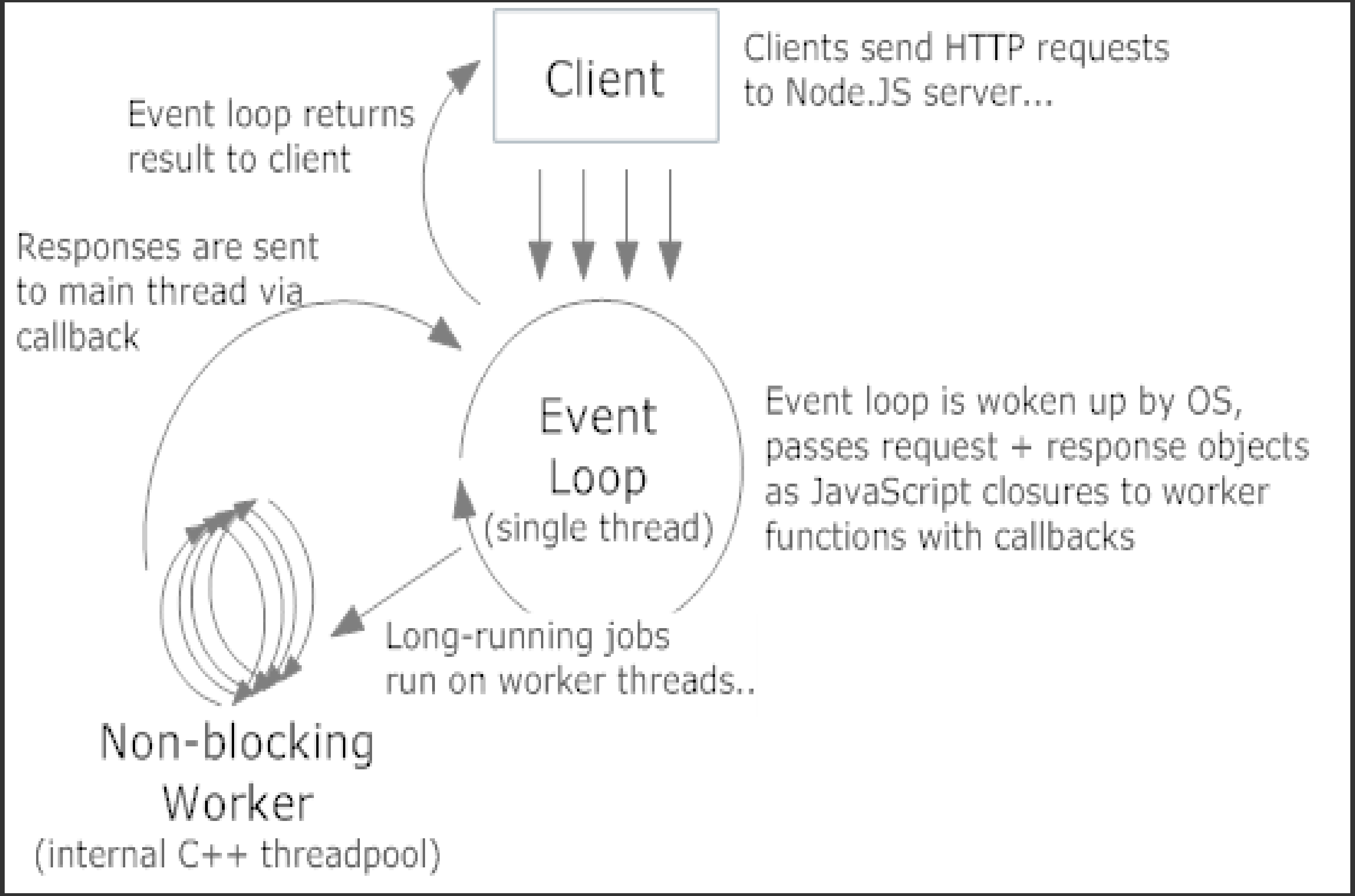
References: Beginning Node.js by Apress

NODE .JS SINGLE THREADED EVENT MODEL



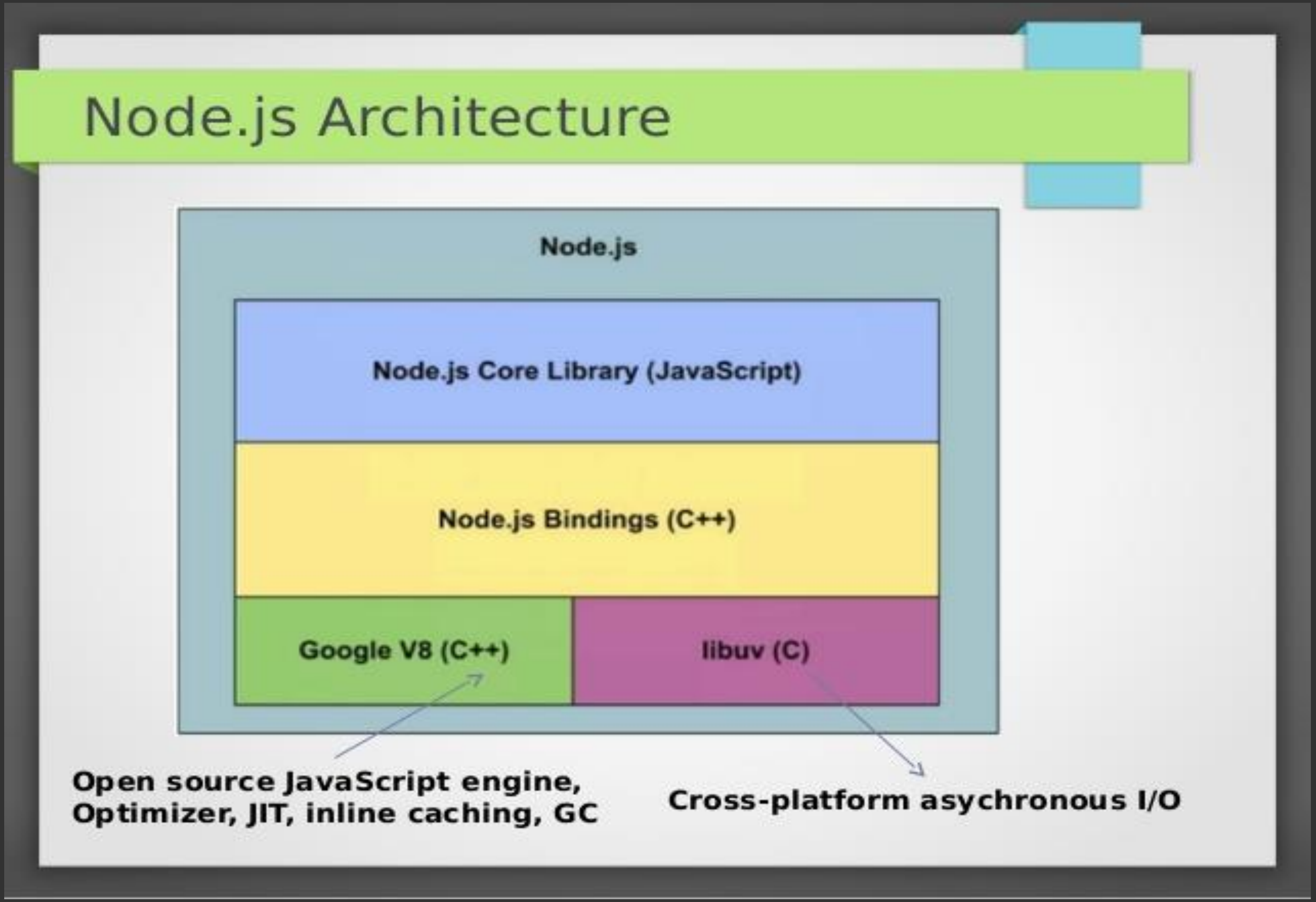
https://cdn.journaldev.com/wp-content/uploads/2015/04/NodeJS-Single-Thread-Event-Model.png

NODE.JS EVENT LOOP


Client
Clients send HTTP requests to Node.JS server...
Event loop returns result to client
Responses are sent to main thread via callback
Event Loop (single thread)
Event loop is woken up by OS, passes request + response objects as JavaScript closures to worker functions with callbacks
Long-running jobs run on worker threads..
Non-blocking Worker (internal C++ threadpool)

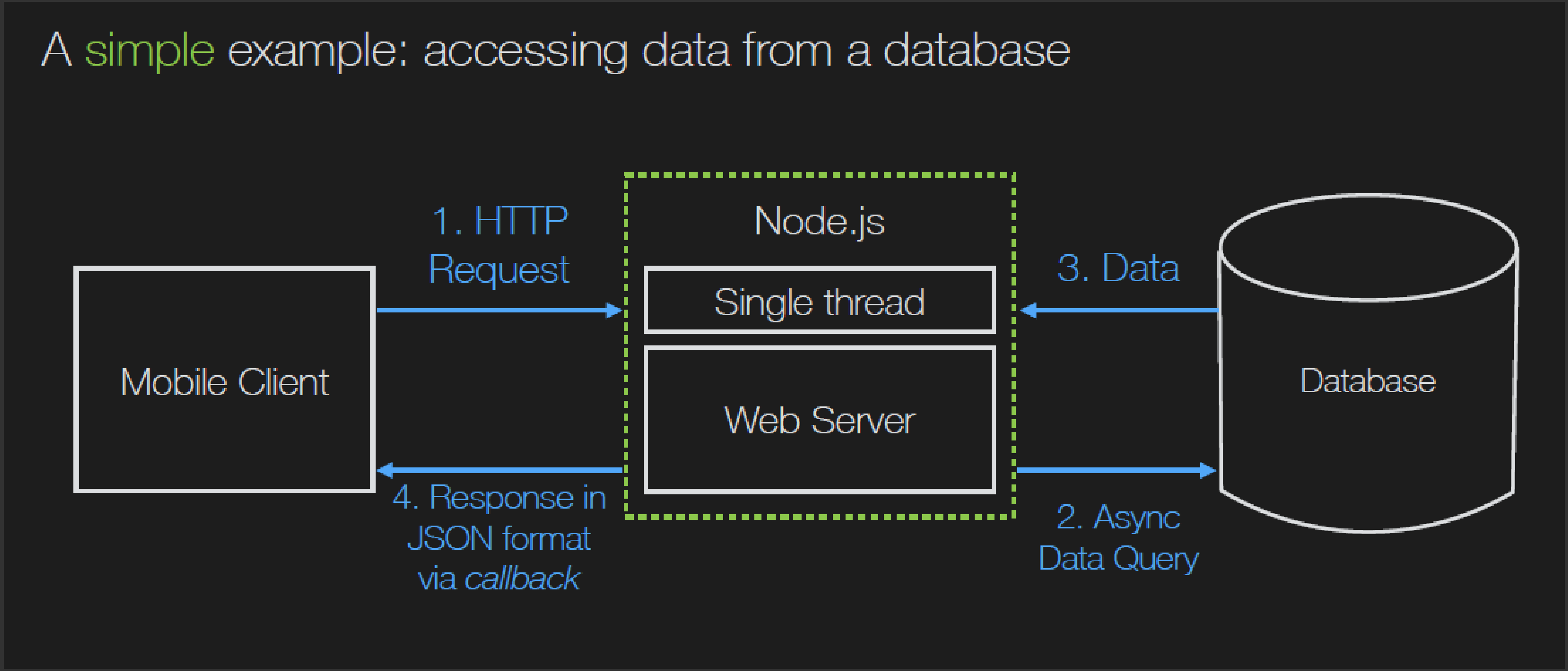
Event loop is a programming construct that waits for and dispatches events in a program once their asynchronous operation completes.

NODE.JS ARCHITECTURE


Node.js Architecture
Node.js
Node.js Core Library (JavaScript)
Node.js Bindings (C++)
Google V8 (C++)
libuv (C)
Open source JavaScript engine, Optimizer, JIT, inline caching, GC
Cross-platform asychronous I/O

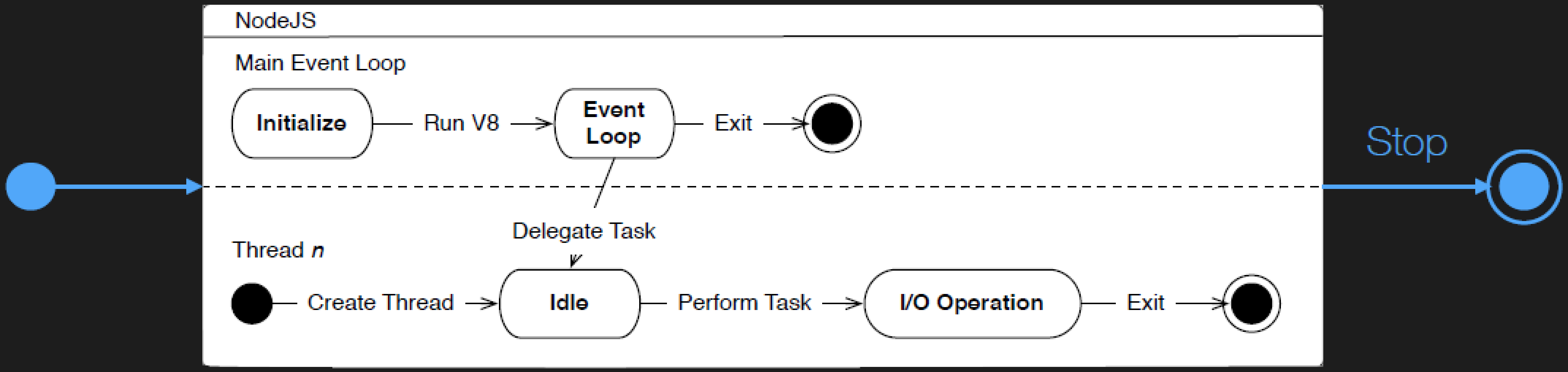
https://www.google.co.in/imgres?imgurl=http%3A%2F%2Fimage.slidesharecdn.com%2F15fuv8js7ulf1nzuakfe-140628073041-phpapp01%2F95%2Fnodejs-enterprise-middleware-25-638.jpg%3Fcb%3D1403940779&imgrefurl=http%3A%2F%2Fwww.slideshare.net%2FBehradZari%2Fnodejs-enterprise-middleware-36409983&docid=X20DF-BPmtIn4M&tbnid=sJSk8mG3P9BLAM%3A&w=638&h=479&bih=758&biw=1536&ved=0ahUKEwjzil23mKXNAhUJLY8KHQcVBS0QMwgkKAYwBg&iact=mrc&uact=8

NODE JS

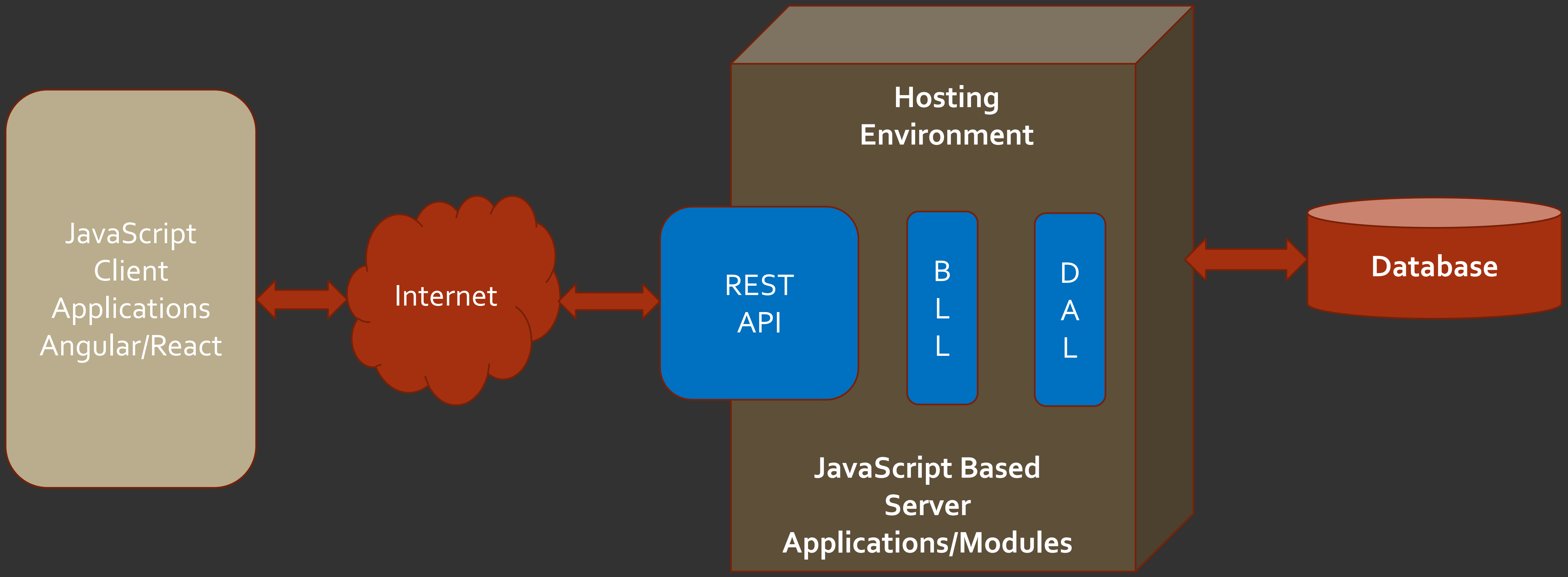

A simple example: accessing data from a database
Mobile Client
1. HTTP Request
Node.js
Single thread
Web Server
3. Data
Database
4. Response in JSON format via *callback*
2. Async Data Query

NODE JS

A generic model of Node.js



ISOMORPHIC APPLICATIONS


JavaScript Client Applications Angular/React
Internet
Hosting Environment
REST API
B L L
D A L
JavaScript Based Server Applications/Modules
Database

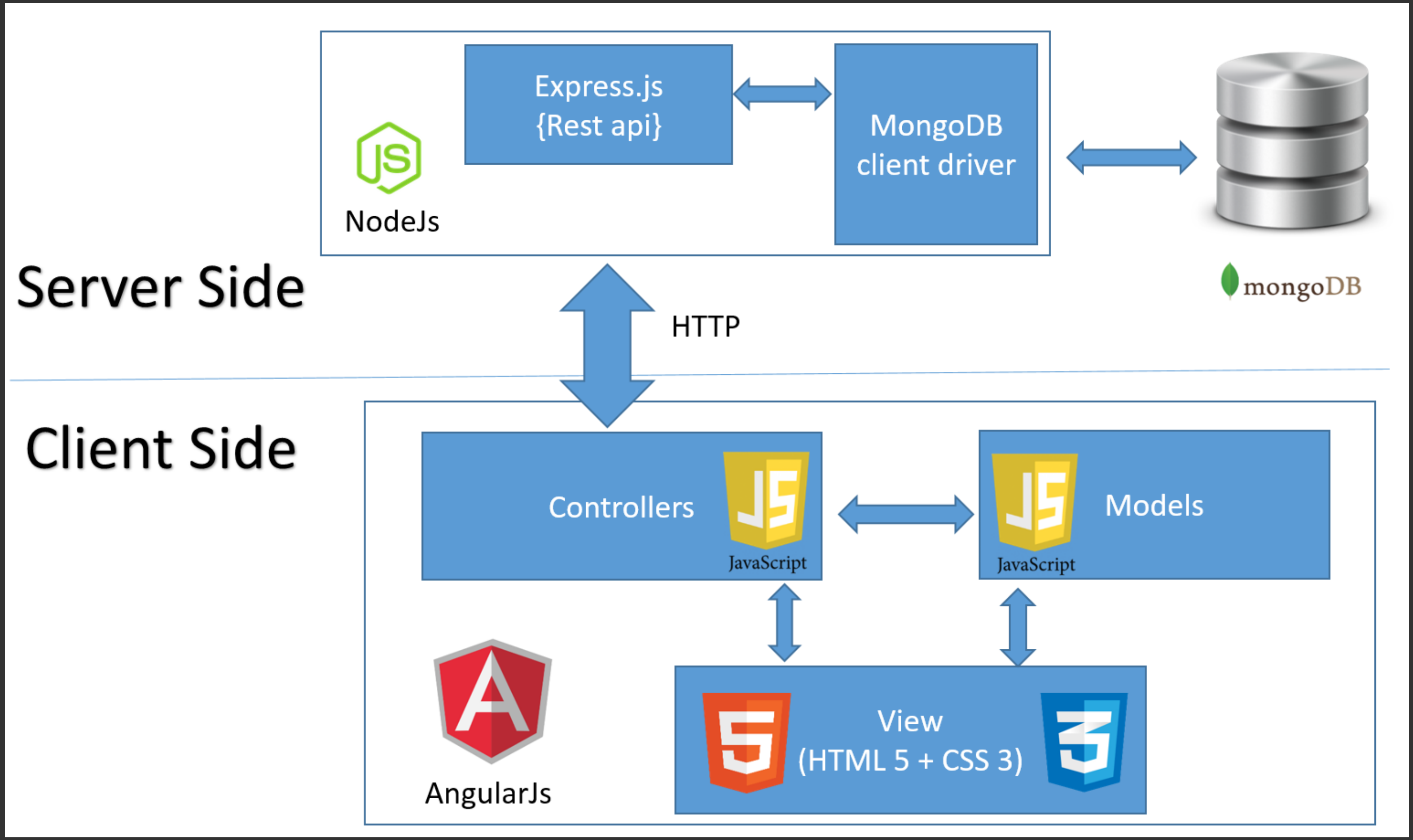
NODE JS

Microsoft PayPal™ in

ebay YAHOO!

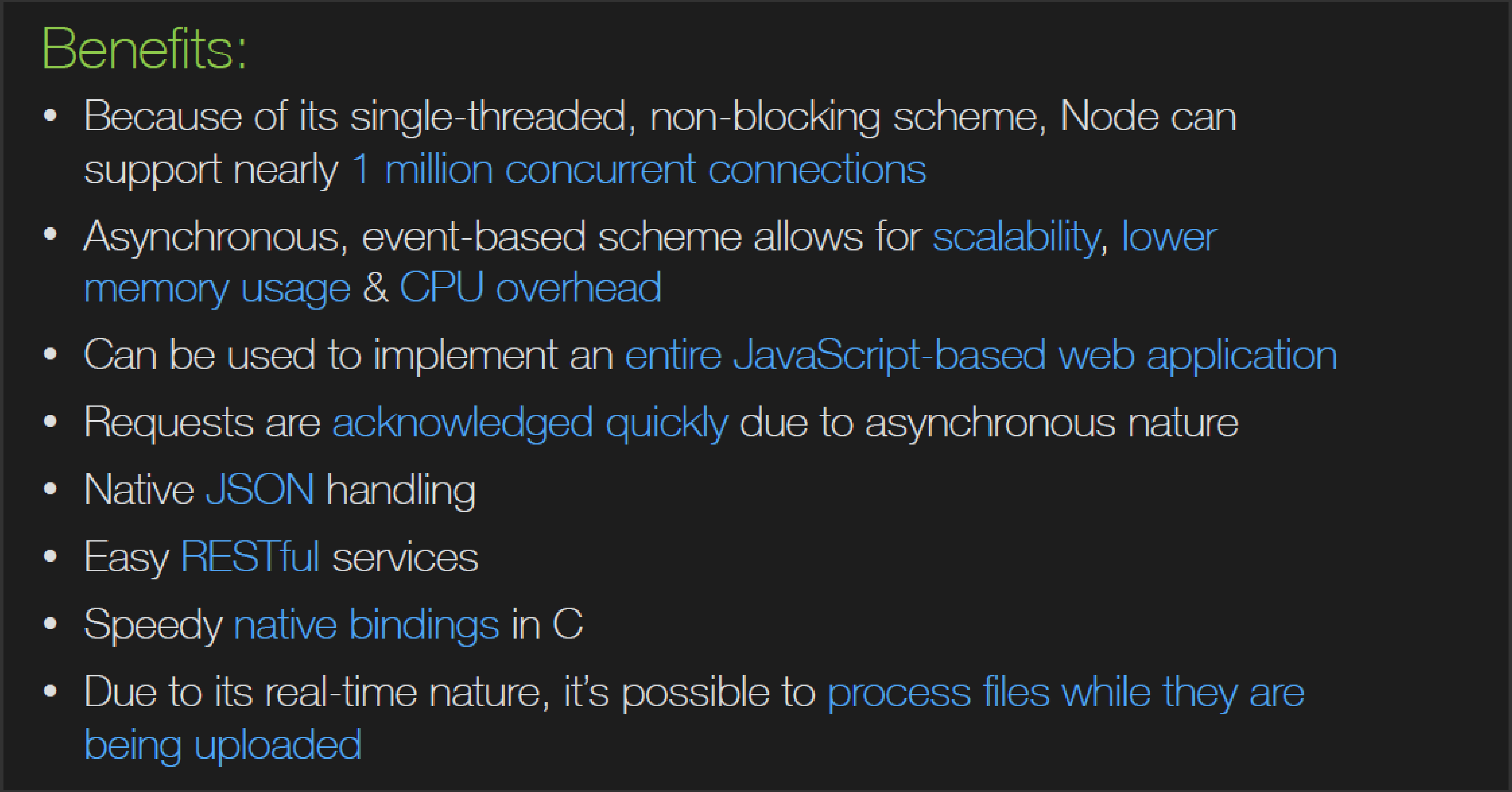
The New York Times

APPLICATION ARCHITECTURE WITH NODE.JS



http://chskr.blogspot.in/2016/01/crud-application-using-angularjs-nodejs.html

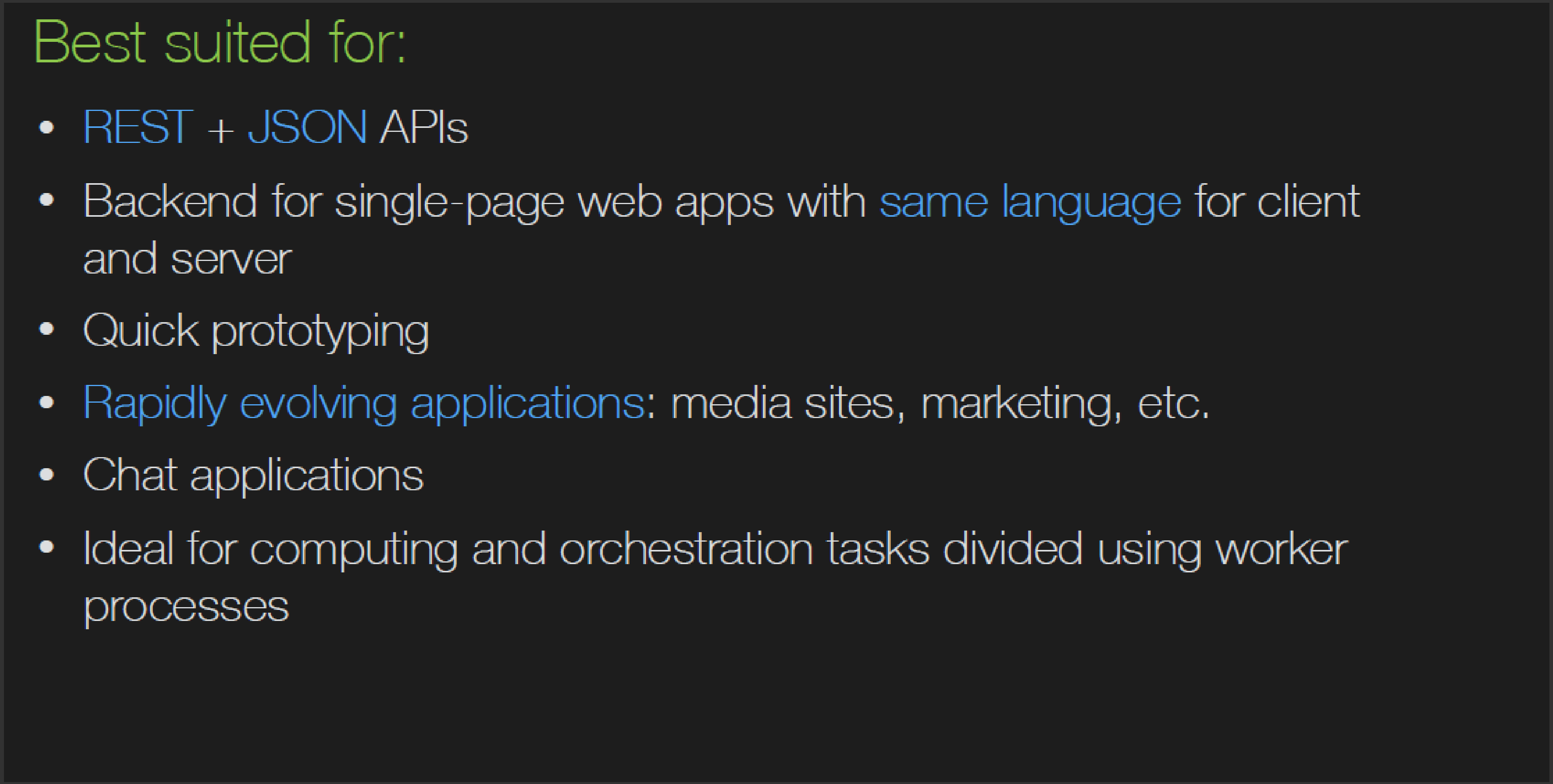
NODE JS



Benefits:

- Because of its single-threaded, non-blocking scheme, Node can support nearly 1 million concurrent connections
- Asynchronous, event-based scheme allows for scalability, lower memory usage & CPU overhead
- Can be used to implement an entire JavaScript-based web application
- Requests are acknowledged quickly due to asynchronous nature
- Native JSON handling
- Easy RESTful services
- Speedy native bindings in C
- Due to its real-time nature, it's possible to process files while they are being uploaded

NODE JS



Best suited for:

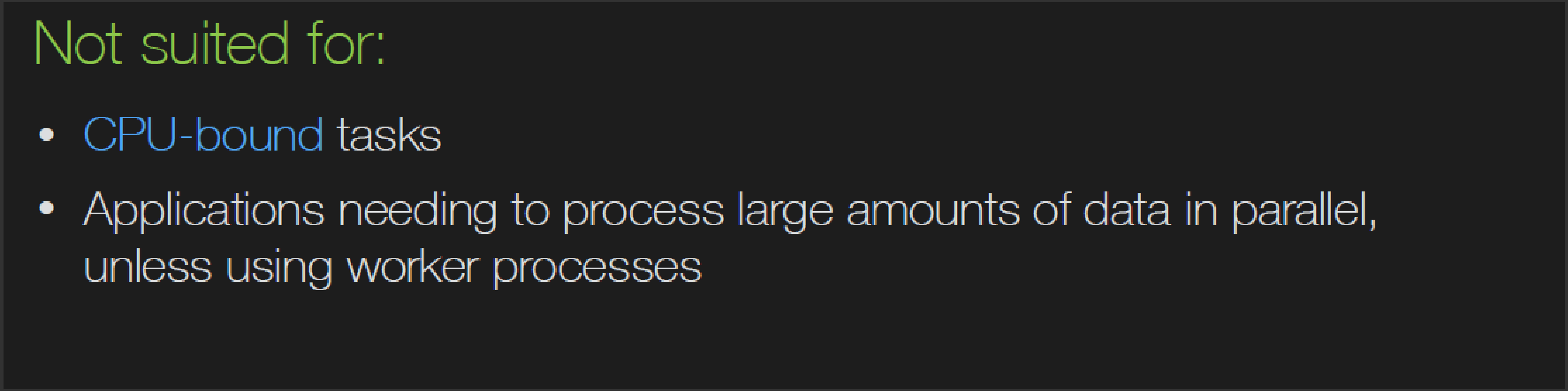
- REST + JSON APIs
- Backend for single-page web apps with same language for client and server
- Quick prototyping
- Rapidly evolving applications: media sites, marketing, etc.
- Chat applications
- Ideal for computing and orchestration tasks divided using worker processes

NODE JS

- **Limitations**
  - **Bad concurrency primitives**
    - JavaScript doesn't really have any good primitive for concurrency
    - **Single Threaded**
      - **Is thread fails all requests will be failed**
  - **Developers must implement Exception Handling**
  - **Lack of maturity**
    - **Since most of the modules are written by different groups, its is difficult to verify the quality for the module. It needs to be used as it is.**
  - **JavaScript's semantics and culture**

https://www.quora.com/What-are-the-disadvantages-of-using-Node-js

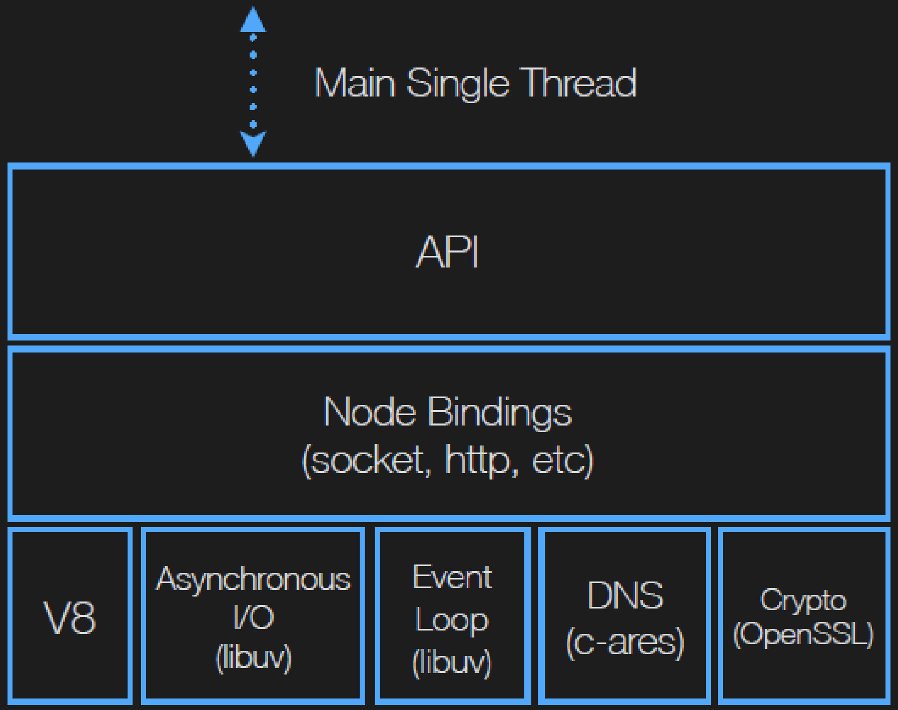
NODE JS



Not suited for:

- CPU-bound tasks
- Applications needing to process large amounts of data in parallel, unless using worker processes

NODE JS



- All requests handled by the Main Single Thread
- API in JavaScript
- Node bindings allow for server operations
- Relies on Google's V8 runtime engine
- Libuv responsible for both asynchronous I/O & event loop

NODE JS

- Heavily influenced by architecture of Unix operating system
- Relies on a small core and layers of libraries and other modules to facilitate I/O operations

- Built-in package manager contributes to the modularity of Node

NODE.JS MODULES

NODE.JS

- Some Installation Commands
- npm install typescript -g

NODE.JS MODULES

- File-Based modules.
  - fs
- Core-module.
  - http
- External module
  - Express
  - Socket
  - Smtp
  - Dns
  - Mongo
  - And many more…

NODE.JS

Creating a Web Server

```
//1.
var http = require('http');
//2.
var server = http.createServer(function (req, resp) {
   //3.
   resp.writeHead(200, { 'Content-Type': 'text/html' });

   //4.
   resp.end("Hello, this is the test web server created");
});
//5.
server.listen(5050);
console.log('Server Started');
```

http Posting Request

```
var server = http.createServer(function (req, resp) {
   if (req.method === "GET") {
      resp.writeHead(200, { 'content-type': 'text/html' });
      resp.end(productPage);
   }
   if (req.method === "POST") {
      var productData = '';
      //6.
      req.on('data', function (prd) {
         productData += prd;
      }).on('end', function () {
         console.log('The received data is ' + productData.toString());
         resp.end('Data received  from you is ' +
productData.toString());
      });
   }
});
```

NODE.JS

- Using ***Fs*** Module
  - Access the File System to process it Synchronously and Asynchronously.
  - Provides Synchronous and Asynchronous functions for Reading/Writing Files.
- Load ***Fs*** Module
- *readFileSync()*
- *readFile()*
- *writeFileSync()*
- *writeFile()*

NODE.JS

File Read

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

//2.
fs.readFile('../Files/MyFile.txt', function (error,
fileData) {
   if (error){
      return console.error('Error Occured while
Reading  ' + error);
   }
   console.log('An Asynchronousn Read' +
fileData.toString());
});
```

File Write

```
//1.
var fs = require('fs');
//2.
fs.writeFile('../Files/testWrite.txt', 'The file is written using
Asynchronous writing',
   function (error) {
   if (error) {
      return console.error(error);
   }
   console.log('Congratulations file is written successfully');
   console.log('Now lets read the data from this file');
});
```

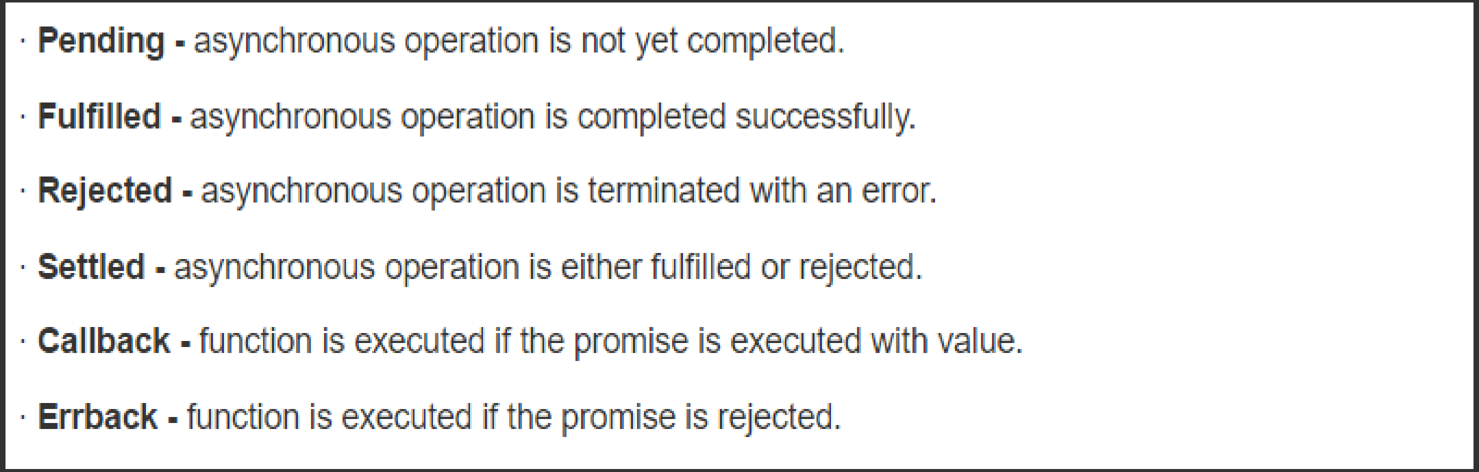
NODE.JS PROMISES

NODE.JS PROMISES

- Writing asynchronous code sometime makes the code unreadable and tedious to maintain.
- Code containing too many callback functions in asynchronous operations, often results in code complexity where it becomes an additional burden for the developer to keep track of inner-functions and outer-functions accurately.
- Typically, performing several asynchronous operations in a sequence or in parallel may be an issue.

NODE.JS PROMISES

- Promise is relatively an easy implementation for asynchronous operation.
- The promise object returned from the function represents an operation which is not completed yet, but it guarantees to the caller of the operation that the operation will be completed in future.

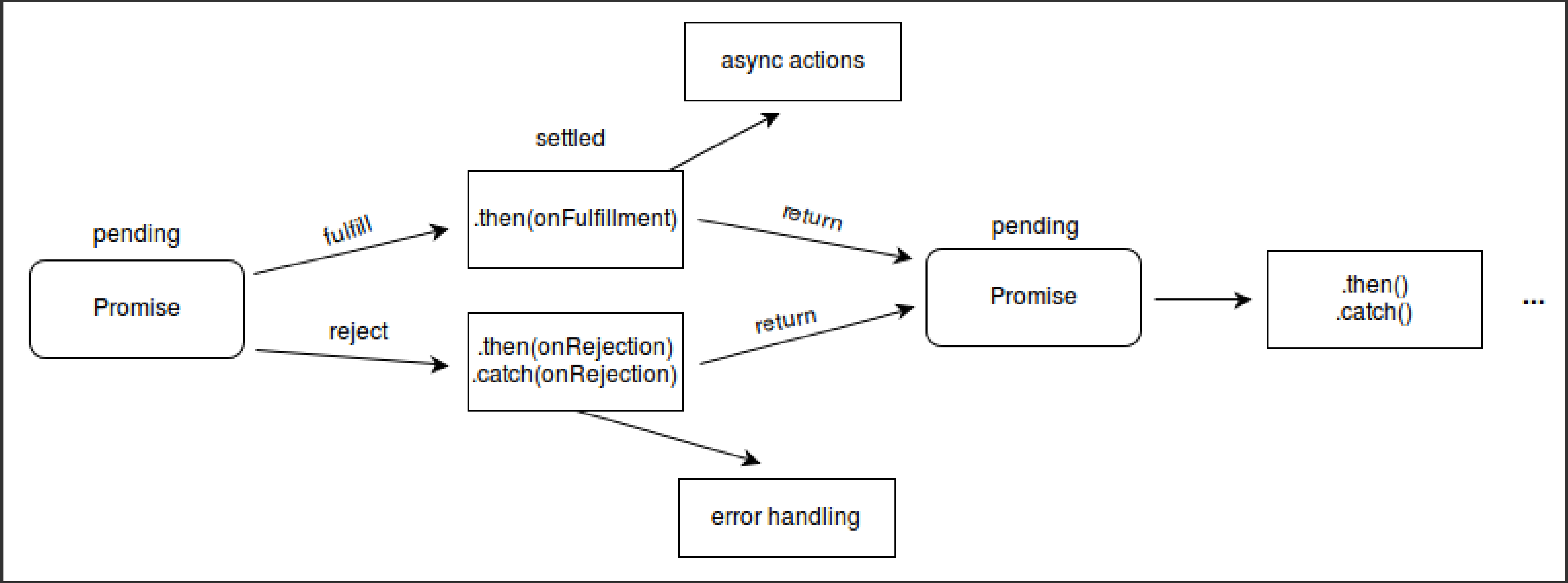


- **Pending -** asynchronous operation is not yet completed.
- **Fulfilled -** asynchronous operation is completed successfully.
- **Rejected -** asynchronous operation is terminated with an error.
- **Settled -** asynchronous operation is either fulfilled or rejected.
- **Callback -** function is executed if the promise is executed with value.
- **Errback -** function is executed if the promise is rejected.

```
promise.then([onFulfilled],[onRejected])
```

http://www.dotnetcurry.com/nodejs/1242/promises-nodejs-using-q-goodbye-callbacks

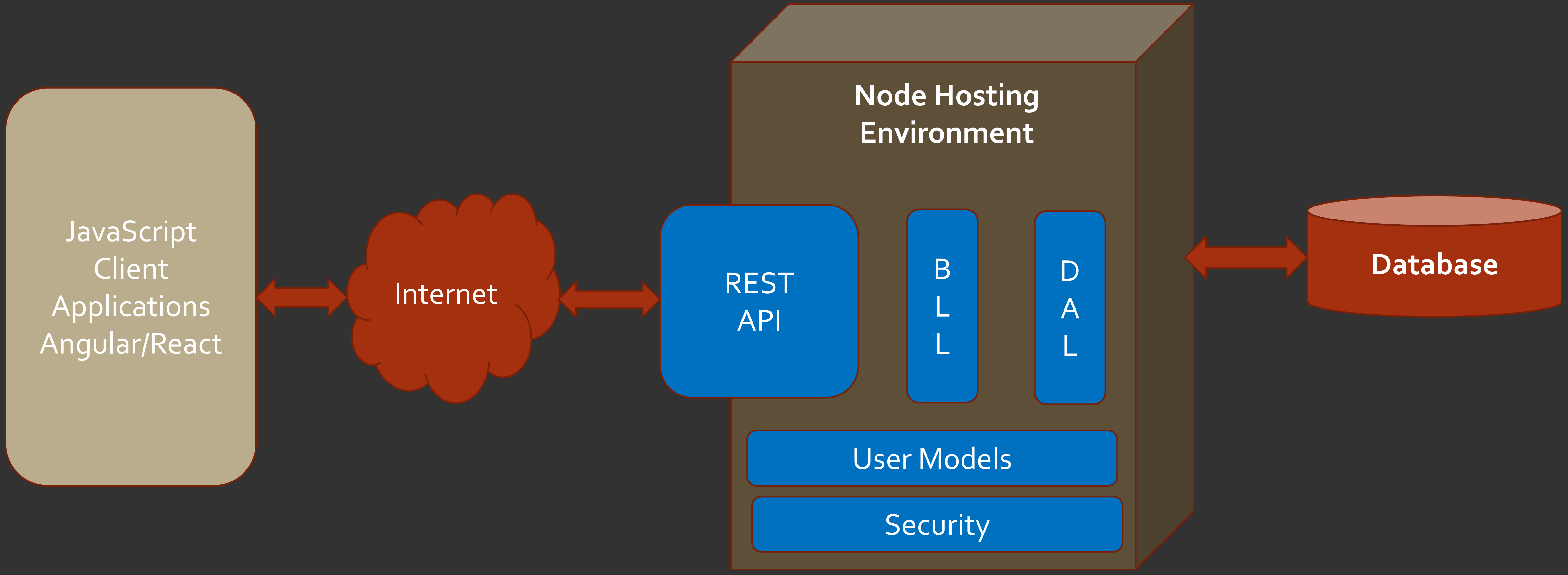
NODE.JS PROMISES

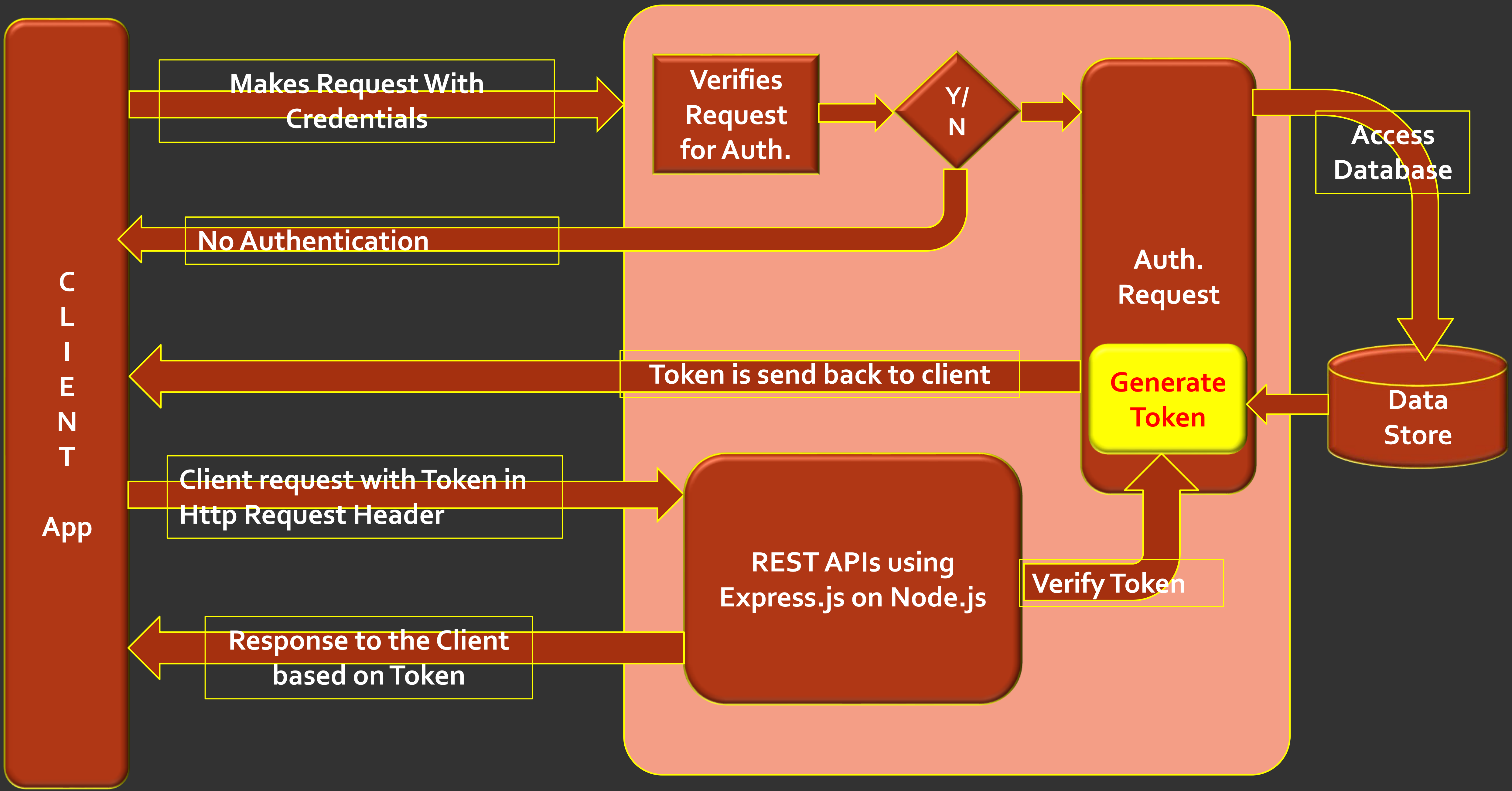


https://www.google.co.in/imgres?imgurl=https://mdn.mozillademos.org/files/8633/promises.png&imgrefurl=https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise&h=297&w=801&tbnid=lU8YHMKZ5uaoLM:&docid=-oTFsZsXOAec5M&ei=_D7kVtzuLMKwuASQrr-gCQ&tbm=isch&ved=0ahUKEwjc3O-QxLvLAhVCGI4KHRDXD5QQMwgsKBAwEA

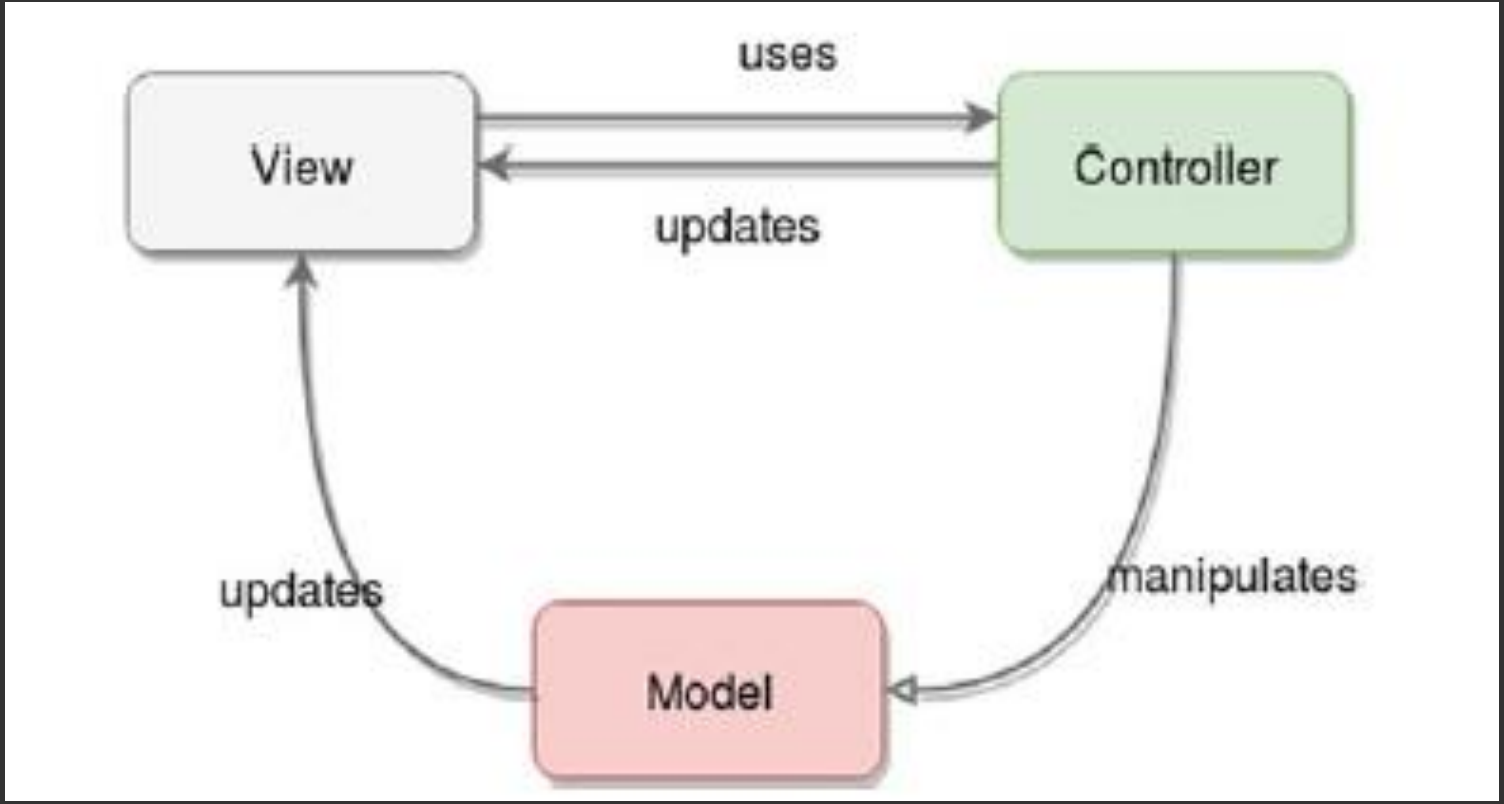
EXPRESS.JS
LINE OF BUSINESS (LOB) APPS

LOB APPLICATIONS




C L I E N T App
Makes Request With Credentials
Verifies Request for Auth.
Y/N
Auth. Request
Access Database
No Authentication
Token is send back to client
Generate Token
Data Store
Client request with Token in Http Request Header
REST APIs using Express.js on Node.js
Verify Token
Response to the Client based on Token

MVC WITH NODE.JS



Client and Server both shares similar concepts

NODE.JS LINE OF BUSINESS APPS

- Node.js
- Express.js
- MongoDB
- Mongoose
- Passport.js
- JWT

EXPRESS.JS

Web Applications
Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

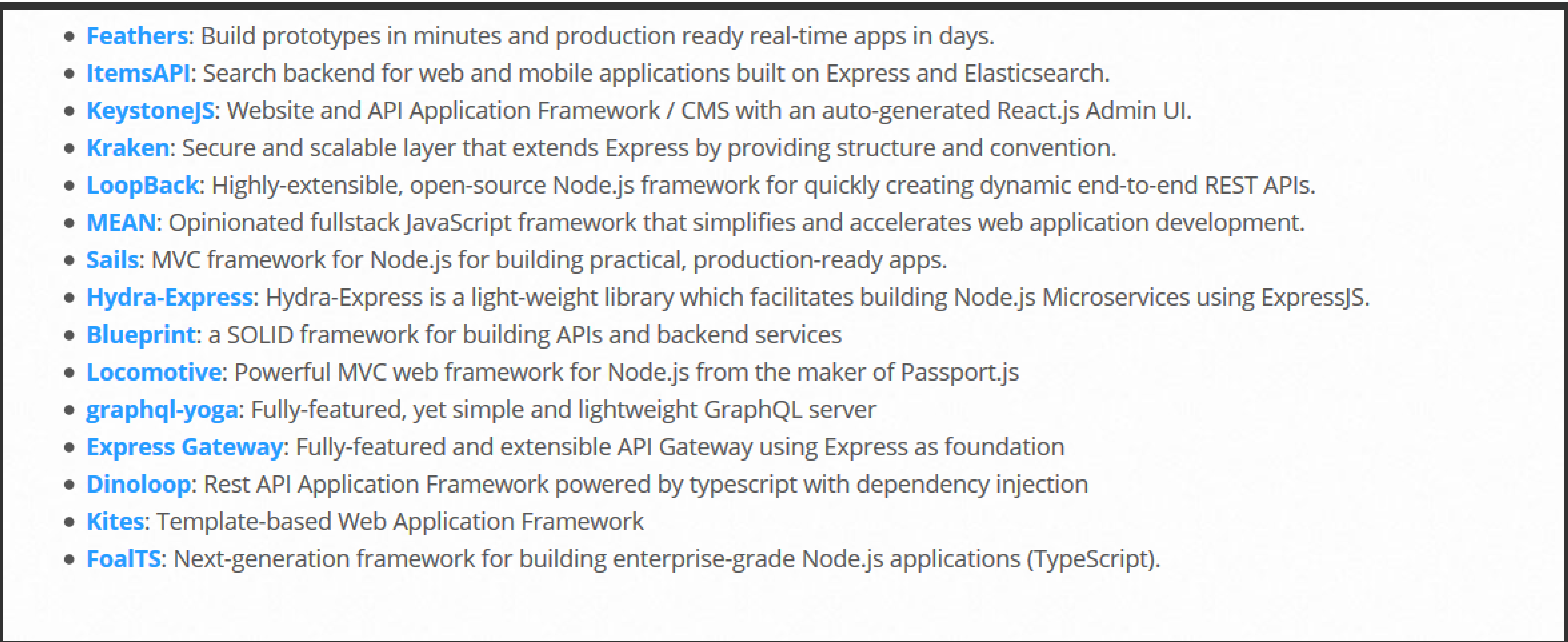
APIs
With a several of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance
Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

EXPRESS.JS

- **Express.js** is a Node**.js** web application server framework, designed for building single-page, multi-page, and hybrid web applications. It is the de facto standard server framework for node**.js**.
- Allows
  - Implementing WEB API.
  - Middleware Logic
  - Routing

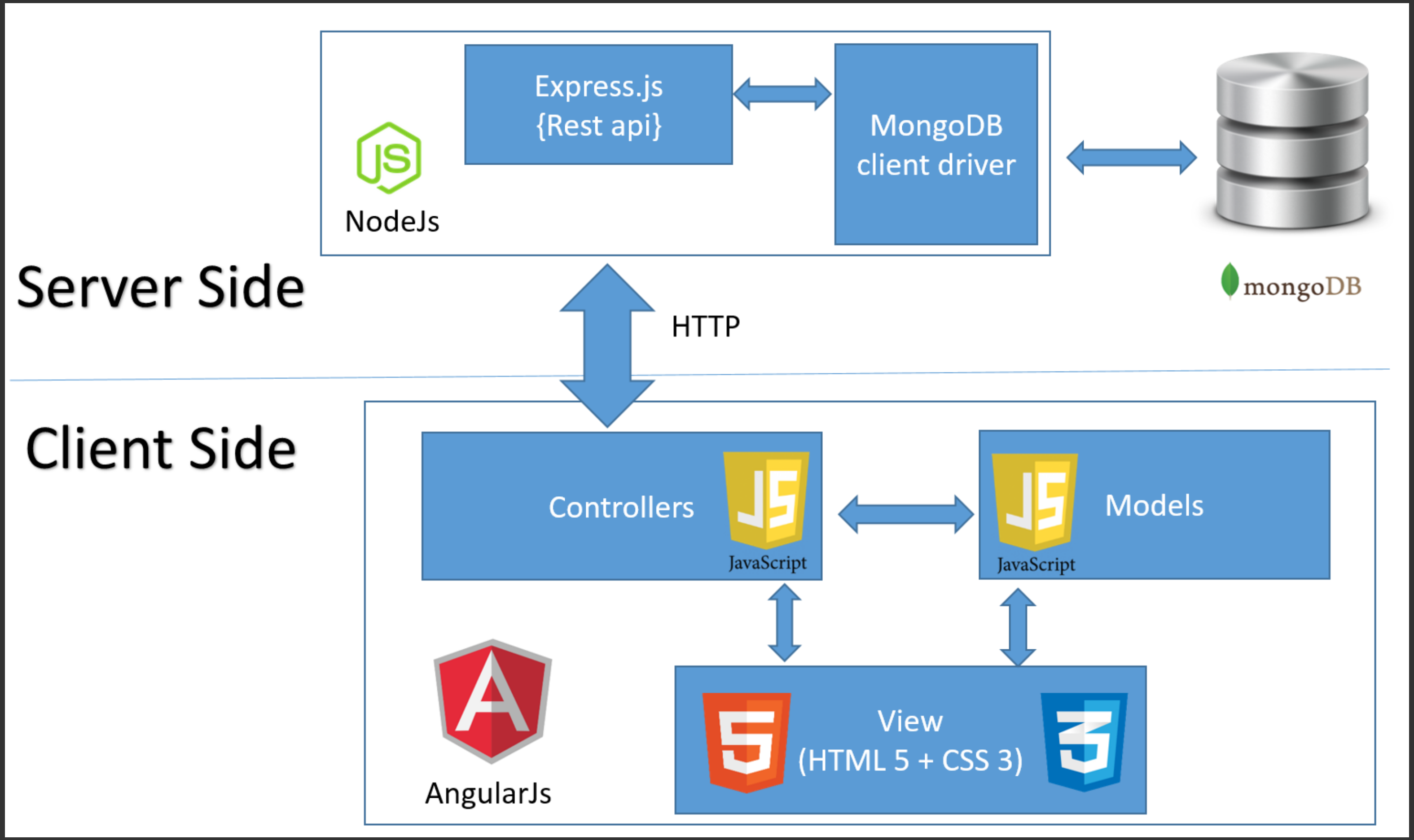
EXPRESS.JS FRAMEWORKS



- **Feathers**: Build prototypes in minutes and production ready real-time apps in days.
- **ItemsAPI**: Search backend for web and mobile applications built on Express and Elasticsearch.
- **KeystoneJS**: Website and API Application Framework / CMS with an auto-generated React.js Admin UI.
- **Kraken**: Secure and scalable layer that extends Express by providing structure and convention.
- **LoopBack**: Highly-extensible, open-source Node.js framework for quickly creating dynamic end-to-end REST APIs.
- **MEAN**: Opinionated fullstack JavaScript framework that simplifies and accelerates web application development.
- **Sails**: MVC framework for Node.js for building practical, production-ready apps.
- **Hydra-Express**: Hydra-Express is a light-weight library which facilitates building Node.js Microservices using ExpressJS.
- **Blueprint**: a SOLID framework for building APIs and backend services
- **Locomotive**: Powerful MVC web framework for Node.js from the maker of Passport.js
- **graphql-yoga**: Fully-featured, yet simple and lightweight GraphQL server
- **Express Gateway**: Fully-featured and extensible API Gateway using Express as foundation
- **Dinoloop**: Rest API Application Framework powered by typescript with dependency injection
- **Kites**: Template-based Web Application Framework
- **FoalTS**: Next-generation framework for building enterprise-grade Node.js applications (TypeScript).

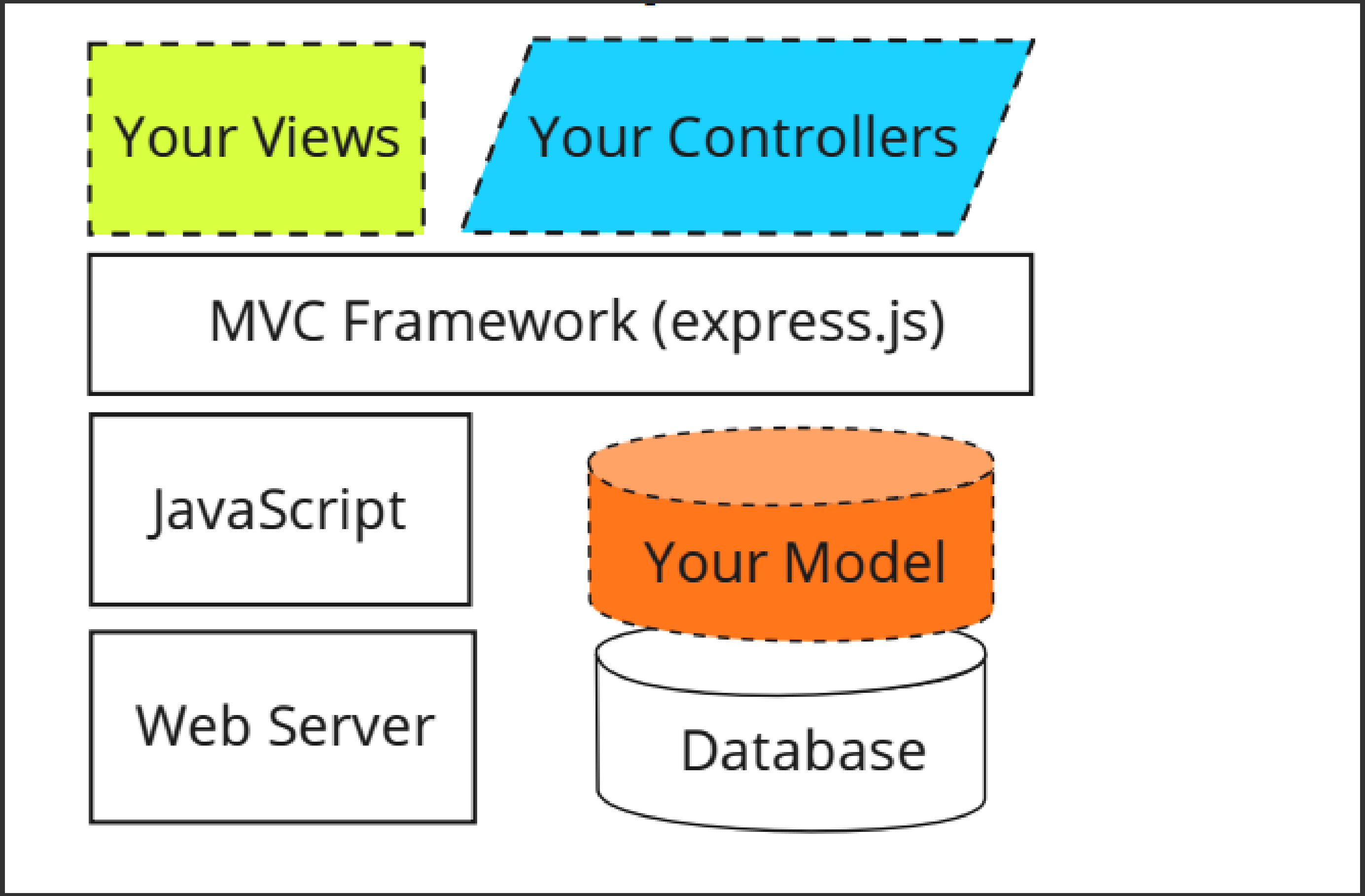
https://expressjs.com/en/resources/frameworks.html

APPLICATION ARCHITECTURE WITH MEAN



http://chskr.blogspot.in/2016/01/crud-application-using-angularjs-nodejs.html

EXPRESS MVC


Your Views
Your Controllers
MVC Framework (express.js)
JavaScript
Your Model
Web Server
Database

USING EXPRESS FOR ROUTING

- Uses to deliver static files from node to client (Browser)
- Uses the Router object
- Combine the route with the **path** module
- Most important in Single/Multi Page applications

USING EXPRESS FOR ROUTING FOR STATIC FILES

- The Implementation

```
var express = require('express');
var path = require('path');
var app = express();
var objExpress = express.Router();
objExpress.get("/", function (request, response) {
    response.sendFile('home.html', { root: path.join(__dirname, './Views')
});
});
```

USING EXPRESS FOR ROUTING FOR DATA

```
objExpress.get('/data', function (req, resp) {

   var Emp = [
      { 'EmpNo': 101, 'EmpName': "A" },
      { 'EmpNo': 102, 'EmpName': "B" },
   ];

   resp.send(Emp);
});
```

USING EXPRESS FOR ROUTING FOR DATA WITH PARAMETERS

```
objExpress.get('/data/:dname', function (req, resp) {
    var id = req.params.dname;
    if (id == 'IT') {
        //Write the Logic


    } else {
        //Write the Logic
    }
    resp.send(Emp);
});
```

REST API USING EXPRESS

- The backbone of the Modern Web and Mobile Applications using Node.
- Uses
  - Router
  - CORS
  - Body-Parser

EXPRESS.JS

- Using Express

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

var app = express();

var cors = require('cors');
app.use(cors());

var bodyParser  =    require("body-parser");

var route = express.Router();

app.use(bodyParser.urlencoded({  extended:false}));

app.use(bodyParser.json());

app.listen(3000,function(){  console.log("Started on PORT 3000");})
```

EXPRESS.JS

```
app.get('/EmployeeList/api/employees', rwOperation.get);

app.post('/EmployeeList/api/employees', rwOperation.add);

app.delete('/EmployeeList/api/employees/:Id',
rwOperation.delete);
```

EXPRESS.JS BASIC AUTH

```
instance.get('/api/employees', function (req, resp) {
  var authValues = req.headers.authorization;
   console.log(authValues);
   var credentials = authValues.split(' ');
  console.log(credentials);
   var credentialValues = credentials[1].split(':');
   if (credentialValues[0] === "mahesh" && credentialValues[1] === "mahesh") {
      resp.send(JSON.stringify(modelData.getAll()));
    }else{
      resp.status(401).send('Invalid Credentials');
    }
   console.log('In Call');
   resp.send(JSON.stringify(modelData.getAll()));

});
```

EXPRESS.JS

- Using Express for Crud
- var mongooseDrv = require('mongoose'); //Get the Mongoose Driver

- mongooseDrv.Promise = global.Promise;
- mongooseDrv.connect('mongodb://localhost/EmployeeDB');
- var db = mongooseDrv.Connection; //The Connection

EXPRESS.JS

- Defining Model

```
var EmployeeInfoSchema = mongooseDrv.Schema({
   EmpNo: String,
   EmpName: String,
   Salary: String,
   DeptName: String,
   Designation: String
});

var EmployeeInfoModel = mongooseDrv.model('Employee', EmployeeInfoSchema,’ Employee‘);
```

EXPRESS.JS

- Model Methods
  - Create
  - Find
  - Remove
  - FindbyIdAndUpdate
  - FindbyIdAndRemove

EXPRESS.JS

```
exports.get = function (req, resp) {
    EmployeeInfoModel.find().exec(function (error, res) {
        console.log(EmployeeInfoModel.log);
        if (error) {
            resp.send(500, { error: error });
        } else {
            resp.send(res);
        }

    });
};
```

EXPRESS.JS

```
exports.add = function (request, response) {
    console.log(request.body);
    var newEmp = { EmpNo: request.body.EmpNo, EmpName: request.body.EmpName, Salary: 
request.body.Salary, DeptName: request.body.DeptName, Designation: 
request.body.Designation };
    console.log(newEmp);

    EmployeeInfoModel.create(newEmp, function (addError, addedEmp) {
        if (addError) {
            response.send(500, { error: addError });
        }
        else {
            response.send({ success: true, emp: addedEmp });
        }
    });
};
```

MONGODB
A POWERFUL NOSQL

MONGODB

- Why NoSQL?
- This new class of technology emerged as answer to the limitations of relational databases in handling Big Data requirements.
- Although NoSQL databases can vary greatly in features and benefits, most offer
  - Greater data model flexibility,
  - Horizontal scalability
  - Superior performance over relational databases.

MONGODB

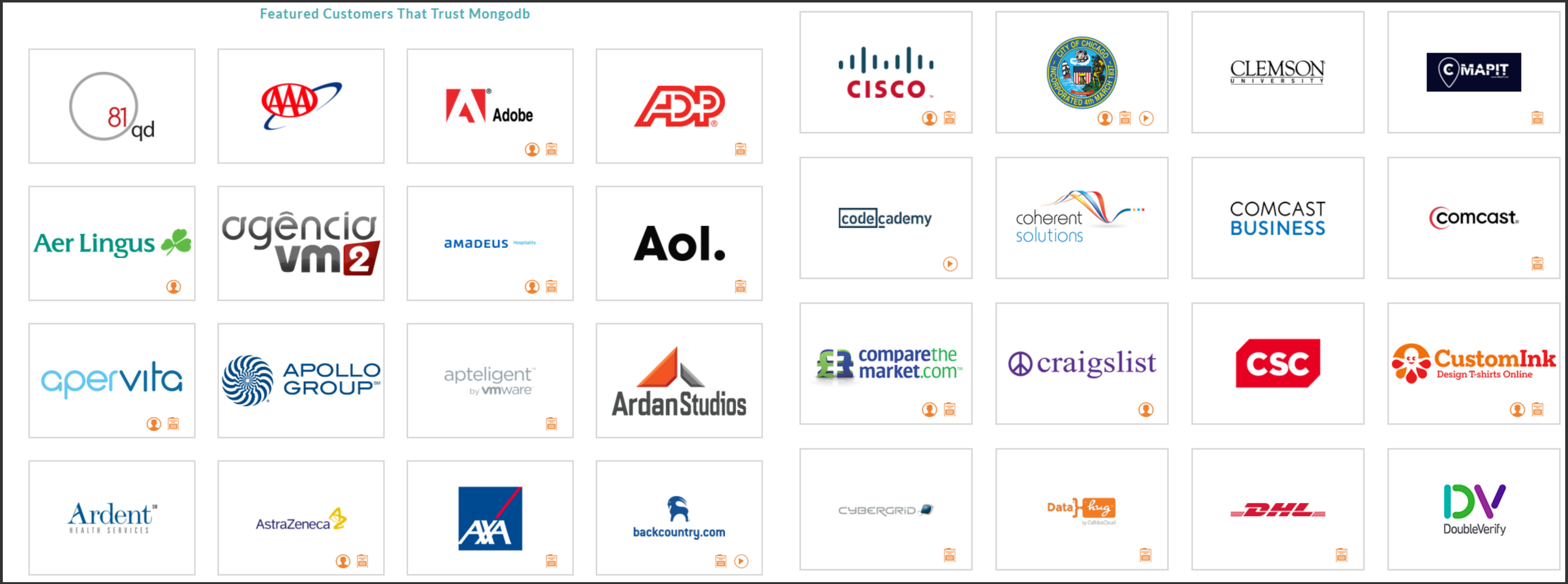
- Why need NoSQL?
- Need to handle large volumes of structured, semi-structured, and unstructured data
- Follow modern development practices such as agile sprints, quick iterations, and frequent code pushes
- Prefer object-oriented programming that is easy to use and flexible
- Want to leverage efficient, scale-out architecture instead of expensive, monolithic architecture

MONGODB

- MongoDB is designed to meet the demands of modern apps with a technology foundation that enables applications through:
- **The document data model** – presenting you **the best way to work with data.**
- **A distributed systems design** – allowing you to **intelligently put data where you want it.**
- A unified experience that gives you the **freedom to run anywhere** – allowing you to future-proof your work and eliminate vendor lock-in.

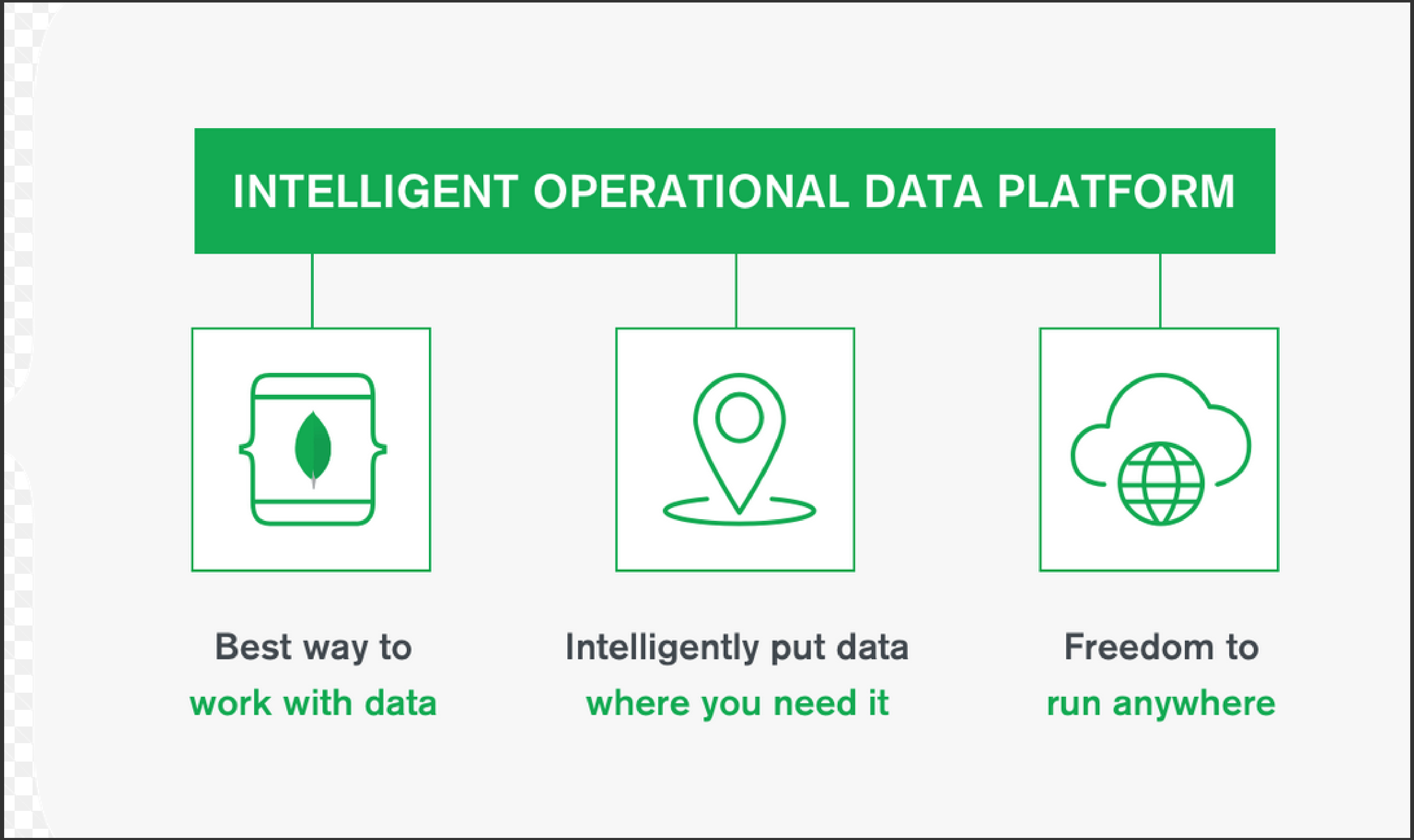
https://www.mongodb.com/mongodb-architecture

MONGODB


Featured Customers That Trust Mongodb

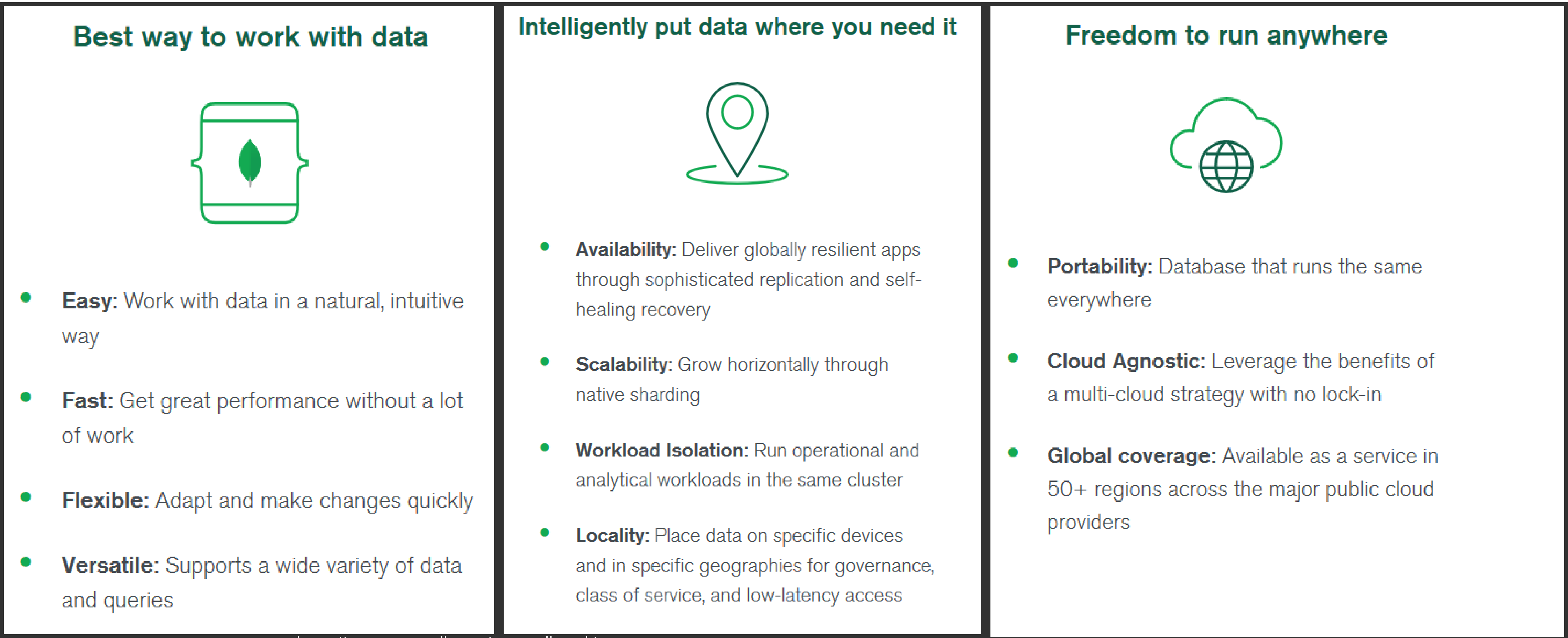
https://www.featuredcustomers.com/vendor/mongodb/customers

MONGODB



https://www.mongodb.com/mongodb-architecture

MONGODB



Best way to work with data

- **Easy:** Work with data in a natural, intuitive way
- **Fast:** Get great performance without a lot of work
- **Flexible:** Adapt and make changes quickly
- **Versatile:** Supports a wide variety of data and queries

Intelligently put data where you need it

- **Availability:** Deliver globally resilient apps through sophisticated replication and self-healing recovery
- **Scalability:** Grow horizontally through native sharding
- **Workload Isolation:** Run operational and analytical workloads in the same cluster
- **Locality:** Place data on specific devices and in specific geographies for governance, class of service, and low-latency access

Freedom to run anywhere

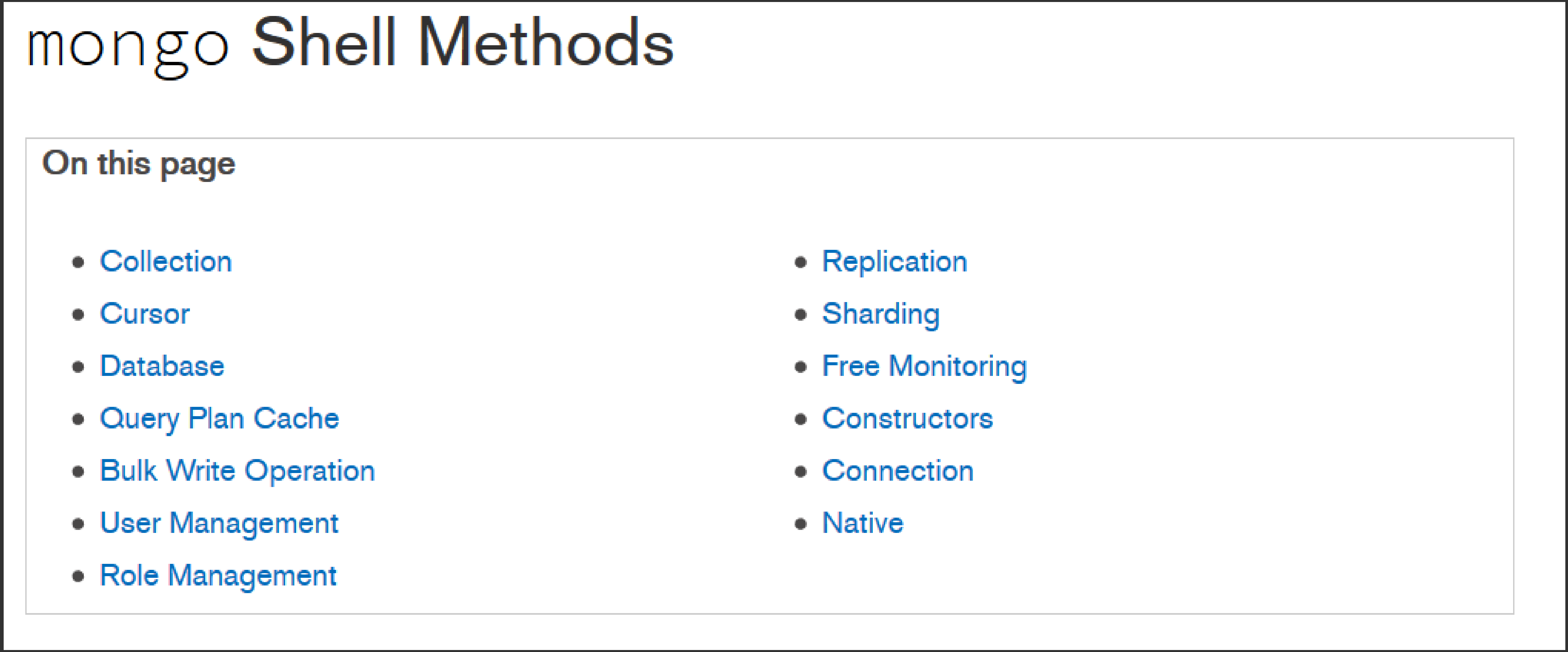
- **Portability:** Database that runs the same everywhere
- **Cloud Agnostic:** Leverage the benefits of a multi-cloud strategy with no lock-in
- **Global coverage:** Available as a service in 50+ regions across the major public cloud providers

https://www.mongodb.com/mongodb-architecture

MONGODB

- Collections
  - Contains JSON Documents
  - ‘_id’
    - The RowId for each document
  - The Variable schema for each document
  - Supports Queries
    - Insert
    - Update
    - Delete
    - Select
  - Object Oriented Queries

MONGODB



mongo Shell Methods

On this page

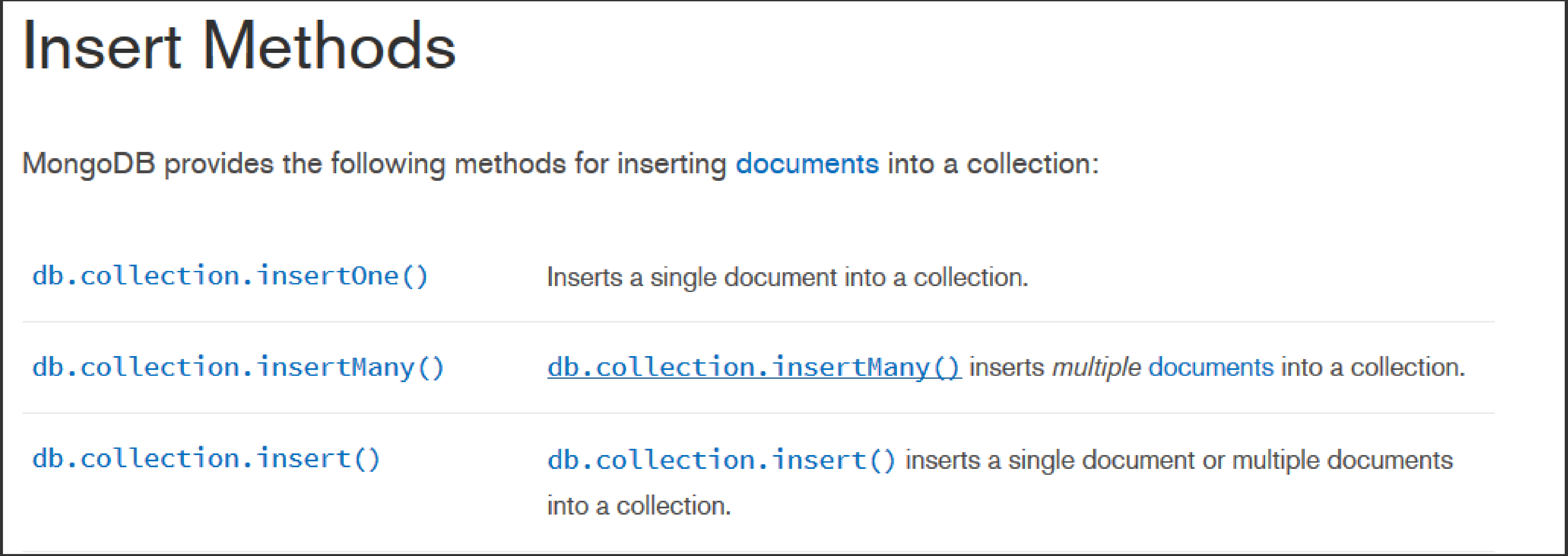
- Collection
- Cursor
- Database
- Query Plan Cache
- Bulk Write Operation
- User Management
- Role Management
- Replication
- Sharding
- Free Monitoring
- Constructors
- Connection
- Native

https://docs.mongodb.com/manual/reference/method/

MONGODB

- https://docs.mongodb.com/manual/reference/method/js-collection/

MONGODB



Insert Methods

MongoDB provides the following methods for inserting documents into a collection:

`db.collection.insertOne()`	Inserts a single document into a collection.
`db.collection.insertMany()`	`db.collection.insertMany()` inserts *multiple* documents into a collection.
`db.collection.insert()`	`db.collection.insert()` inserts a single document or multiple documents into a collection.

MONGODB

```
db.inventory.insertMany( [
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D"
},
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status:
"A" }
]);
```

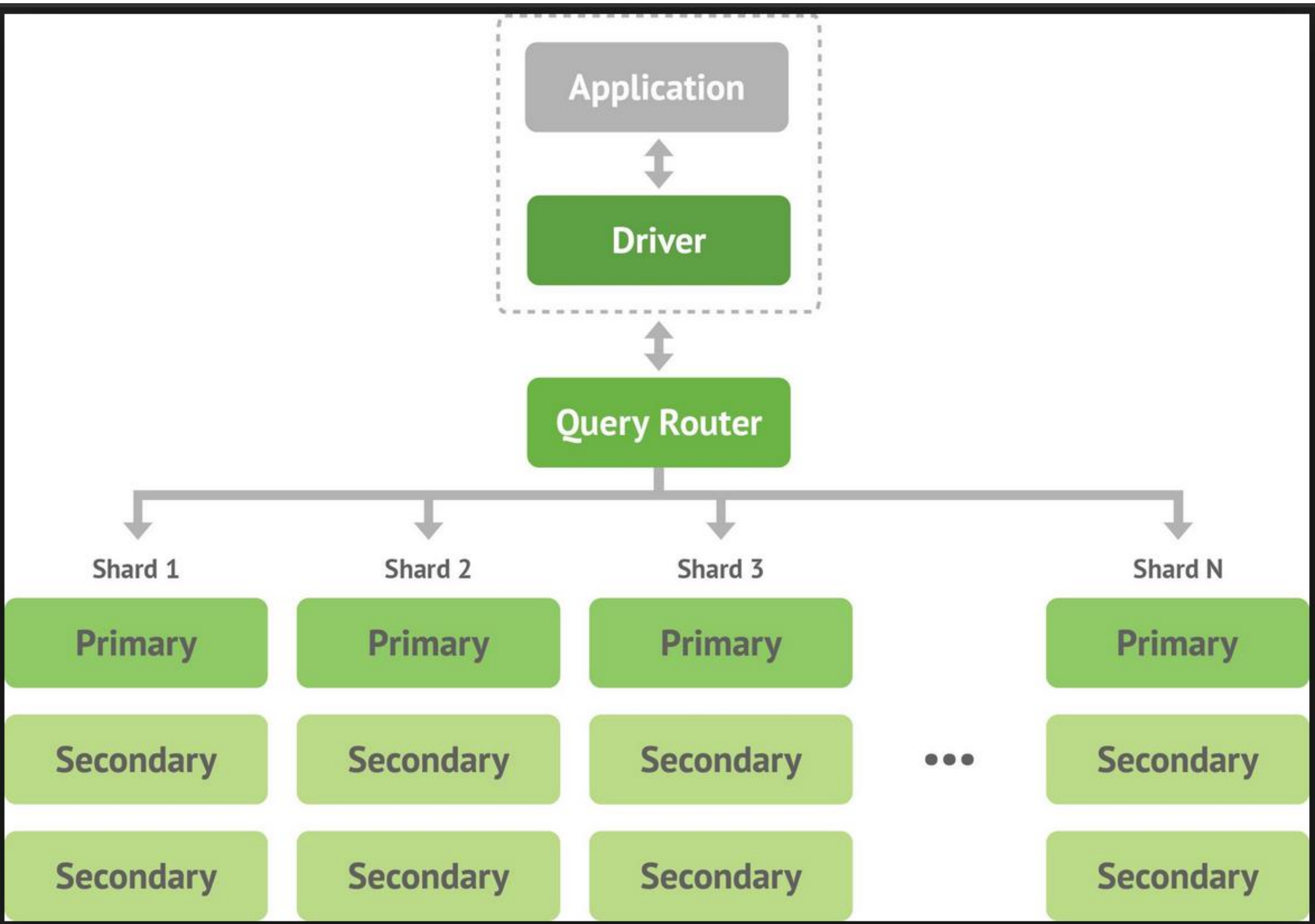
MONGODB

```
db.books.update(
  { _id: 1 },
  {
   $inc: { stock: 5 },
   $set: {
    item: "ABC123",
    "info.publisher": "2222",
    tags: [ "software" ],
    "ratings.1": { by: "xyz", rating: 3 }
   }
  }
)
```

```
try {
  db.restaurant.updateOne(
    { "name" : "Central Perk Cafe" },
    { $set: { "violations" : 3 } }
  );
} catch (e) {
  print(e);
}
```

```
try {
  db.restaurant.updateMany(
    { violations: { $gt: 4 } },
    { $set: { "Review" : true } }
  );
} catch (e) {
  print(e);
}
```

MONGODB



https://d15shllkswkcto.cloudfront.net/wp-content/blogs.dir/1/files/2013/08/MongoDB-Architecture.jpg

NODE + EXPRESS + MONGO

- var mongooseDrv = require('mongoose'); //Get the Mongoose Driver
- mongooseDrv.Promise = global.Promise; // Global Promise to act as a comtainer for all promises in Communication
- mongooseDrv.connect('mongodb://localhost/EmployeeDB');
- var db = mongooseDrv.Connection; //The Connection
- var EmployeeInfoSchema = mongooseDrv.Schema({
-   EmpNo: String,
-   EmpName: String,
-   Salary: String,
-   DeptName: String,
-   Designation: String
- });
- var EmployeeInfoModel = mongooseDrv.model('MyEmpCollection', EmployeeInfoSchema,'MyEmpCollection');

NODE + EXPRESS + MONGO

```
EmployeeInfoModel.find().exec(function
(error, res) {

    console.log(EmployeeInfoModel.log);

    if (error) {

      resp.send(500, { error: error });

    } else {

      resp.send(res);

    }


  });
```

```
EmployeeInfoModel.create(newEmp,
function (addError, addedEmp) {

    if (addError) {

      response.send(500, { error: addError
});

    }

    else {

      response.send({ success: true, emp:
addedEmp });

    }

  });
```

```
EmployeeInfoModel.remove({ _id: id },
function (remError, addedEmp) {

    if (remError) {

      response.send(500, { error: remError
});

    }

    else {

      response.send({ success: 200 });

    }

  });
```

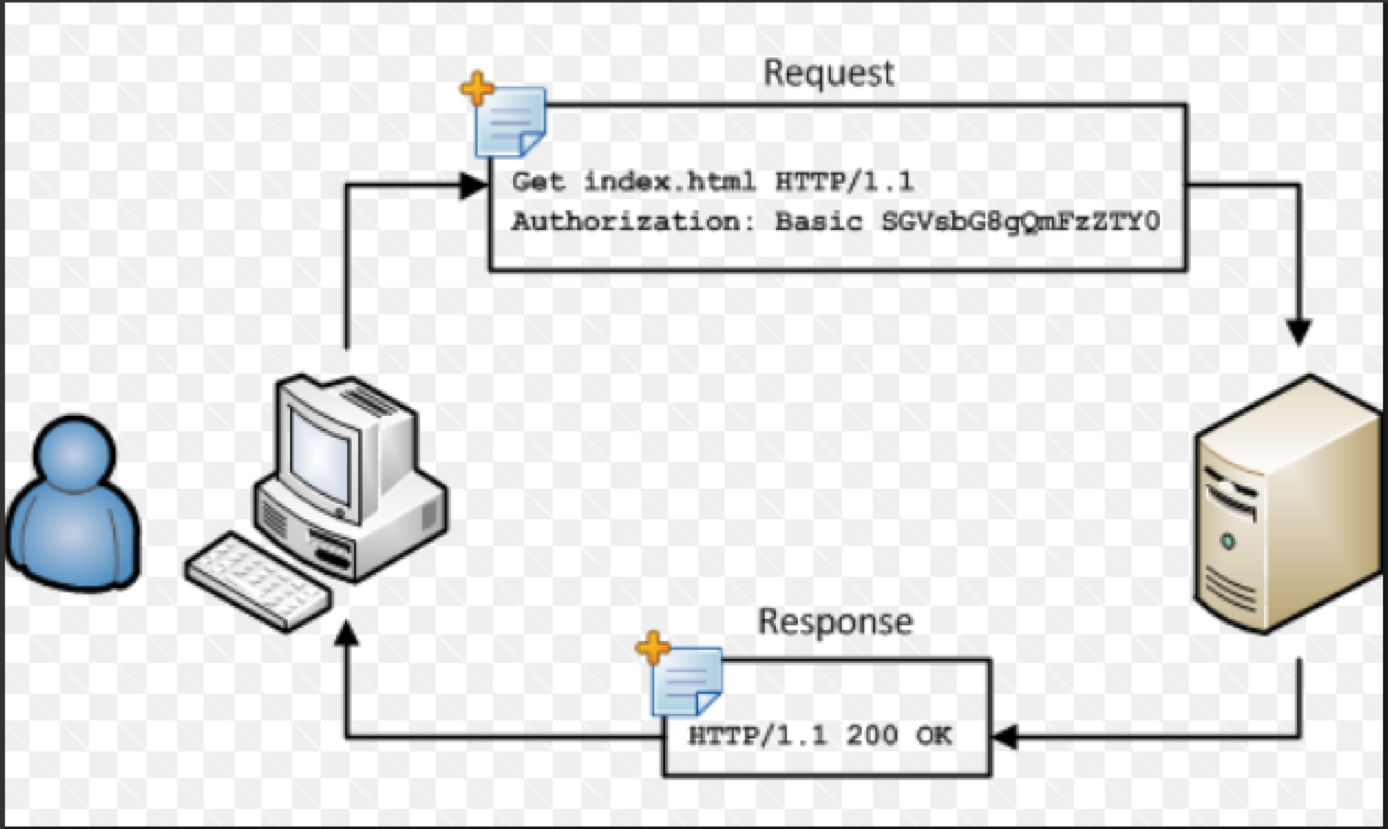
NODE JS
IMPLEMENTING SECURE COMMUNICATION

NODE.JS

- **Secure Communication**
- Making Secure Calls to the Node Server.
  - Use of Basic and Digest.
- Setting Header Value in Basic Authentication
  - request.headers.authorization.replace(/^Basic/, '');
  - Not recommended in the Large Scale Internet Applications
- Digest Authentication
  - Provides Encryption for the credential information

NODE.JS

- Basic Authentication
  - Purely based on UserName and Password
  - Passed in the headers using
    - 'Basic' UserName:Password
- Reading Authentication
  - Request.headers.authorization

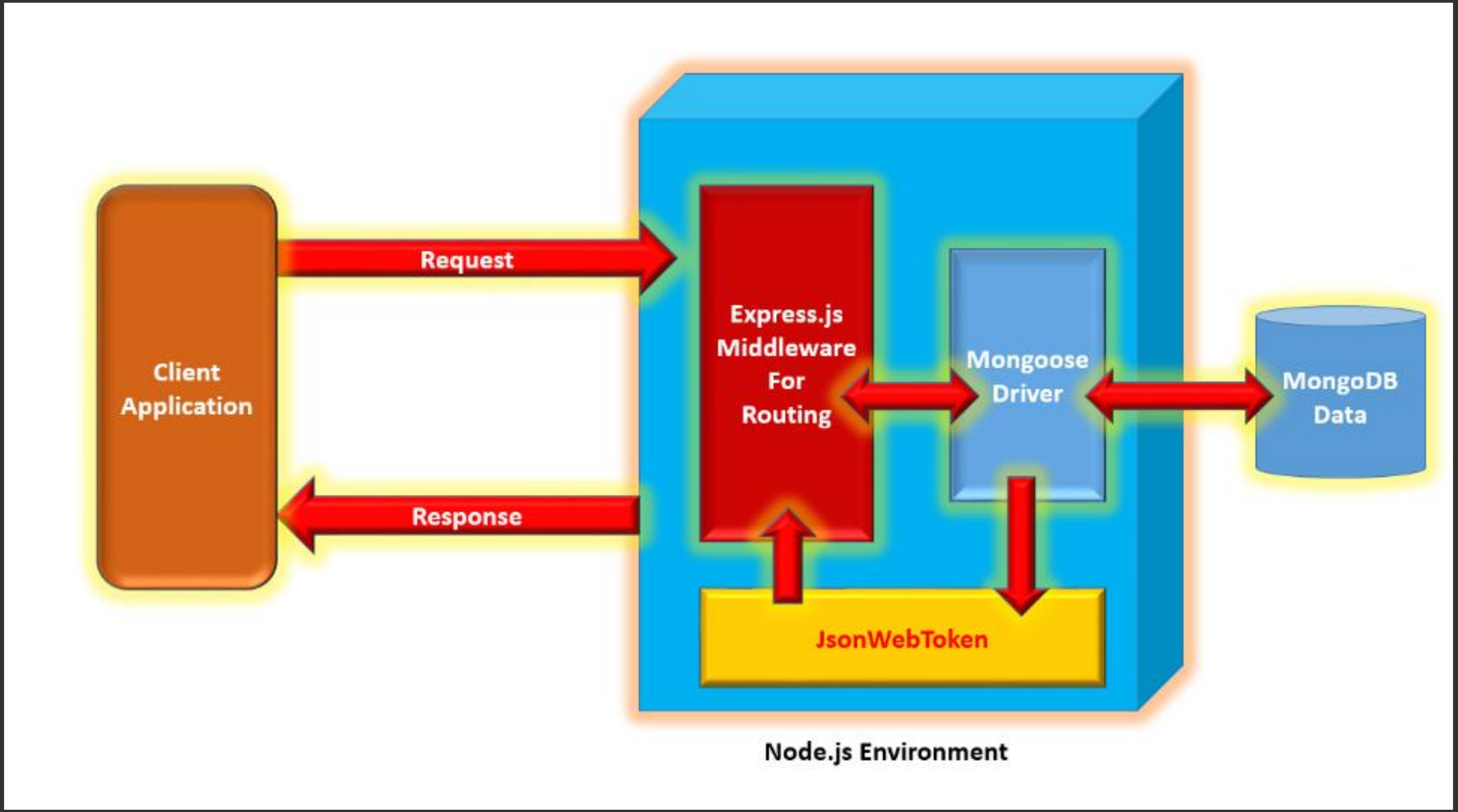


http://planet.jboss.org/post/basic_authentication_with_node_js

NODE.JS

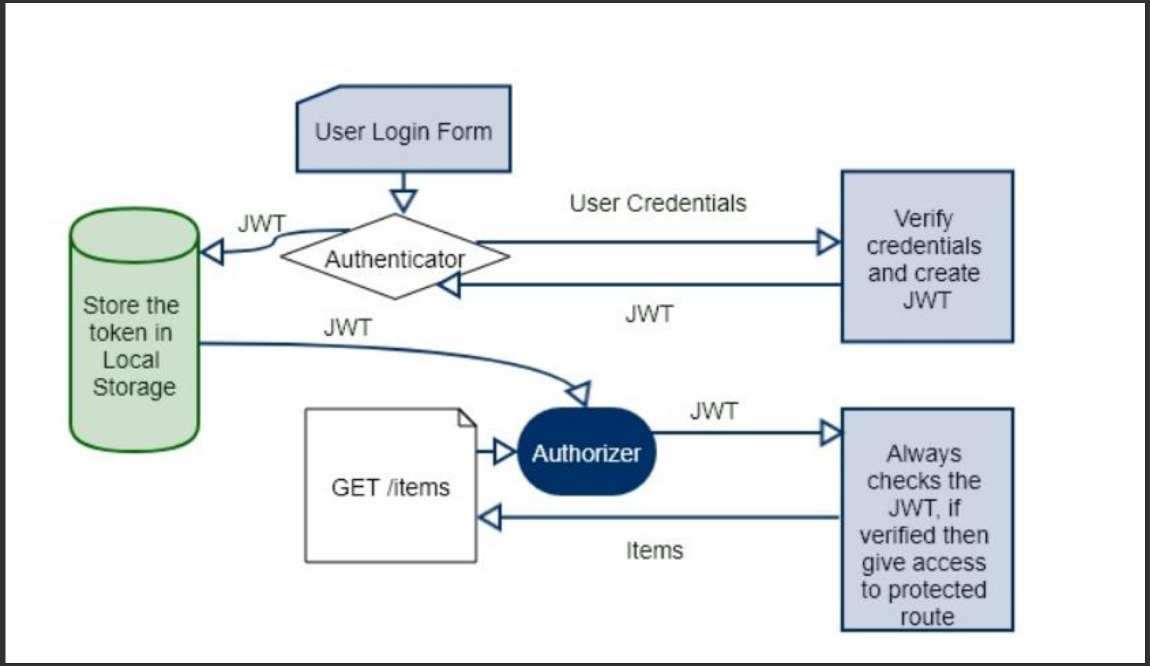
- Token based Authentication
  - JWT Package for token
  - Generate Token for Authenticated User.

TOKEN BASED AUTH



http://www.dotnetcurry.com/nodejs/1302/nodejs-token-based-authentication-security

NODE.JS


User Login Form
User Credentials
JWT
Authenticator
Verify credentials and create JWT
JWT
Store the token in Local Storage
JWT
JWT
Authorizer
GET /items
Items
Always checks the JWT, if verified then give access to protected route

1.First user attempt to login with their credentials.
2.After server verifies the credentials, it sends **JSON Web Token** to the client.
3.A client then saves that token in local storage or any other storage mechanism.
4.Again if a client wants to request a protected route or resource, then it sends **JWT** in a request header.
5.The server verifies that **JWT** and if it is correct then return a 200 response with the information, client needs.
6.If the **JWT** is invalid, then it gives unauthorized access or any other restricted message.

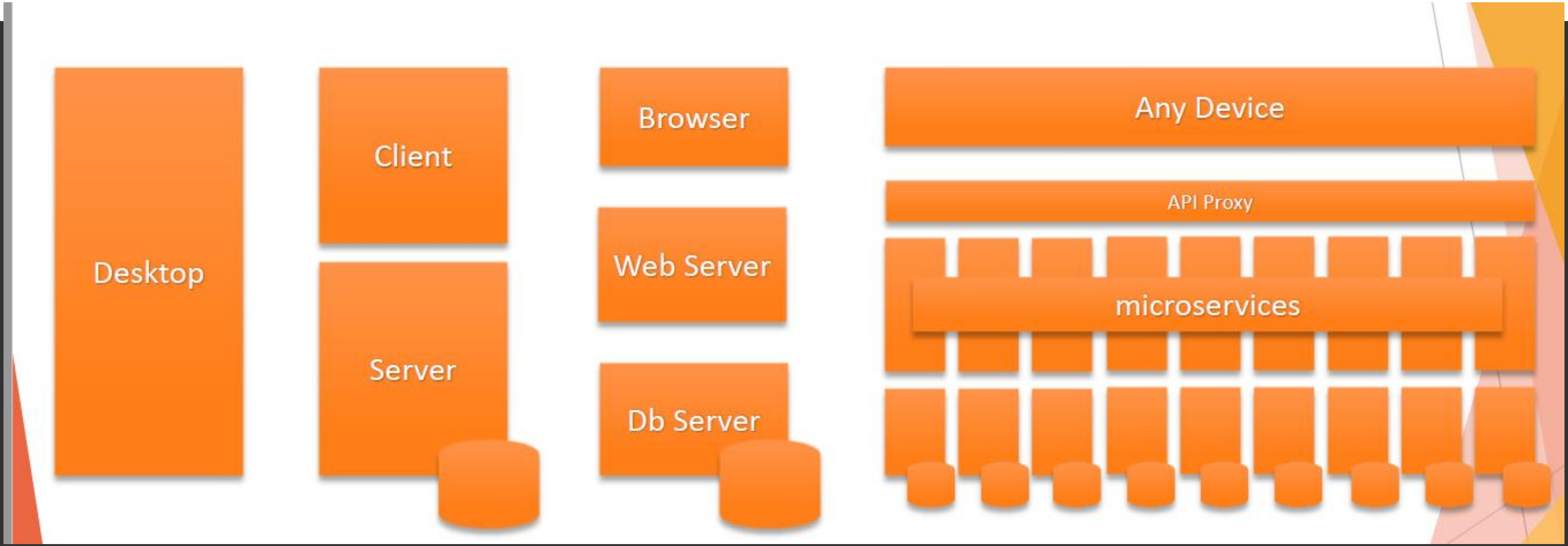
https://appdividend.com/2018/02/07/node-js-jwt-authentication-tutorial-scratch/

NODE.JS

- **What is a Refresh Token?**
  - A **Refresh Token** is a special kind of token that can be used to obtain a renewed access token —that allows accessing a protected resource— at any time. You can request new access tokens until the refresh token is blacklisted. Refresh tokens must be stored securely by an application because they essentially allow a user to remain authenticated forever.
  - Whenever an **Access Token** is required to access a protected resource, a client may use a **Refresh Token** to get a new **Access Token** issued by the *Authentication Server*. Although **Access Tokens** can be renewed at any time using **Refresh Tokens**, they should be renewed when old ones have expired, or when getting access to a new resource for the first time. **Refresh Tokens** never expire. They are usually subject to strict storage requirements to ensure they are not leaked. Nevertheless, they can be blacklisted by the authorization server.

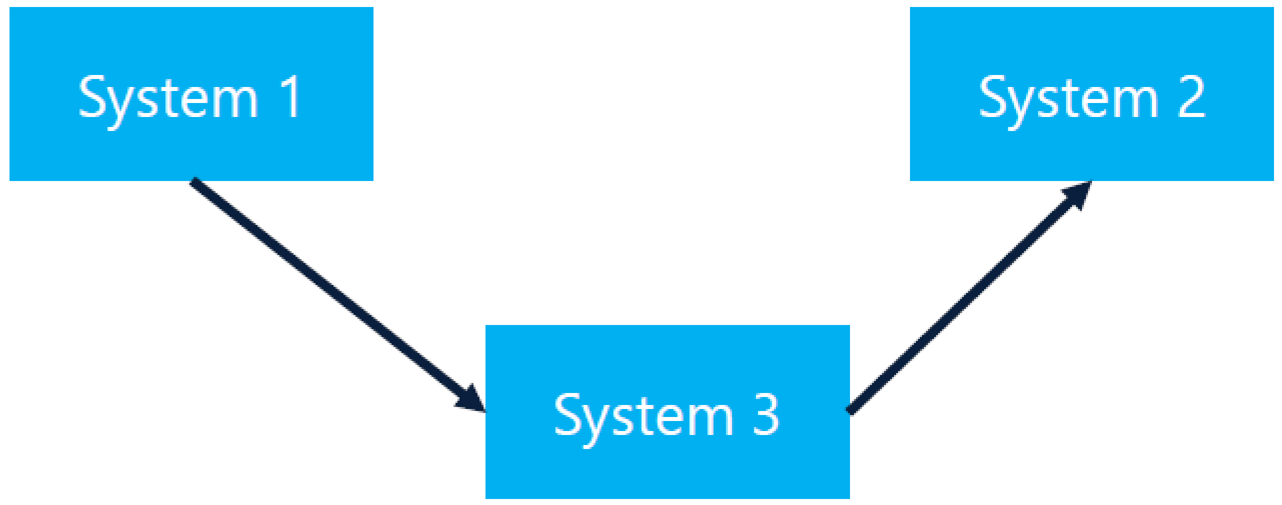
MICROSERVICES

EVOLUTION OF ARCHITECTURE



THE APPROACH OF SOA

"Service orientation is a means for integrating across diverse systems"



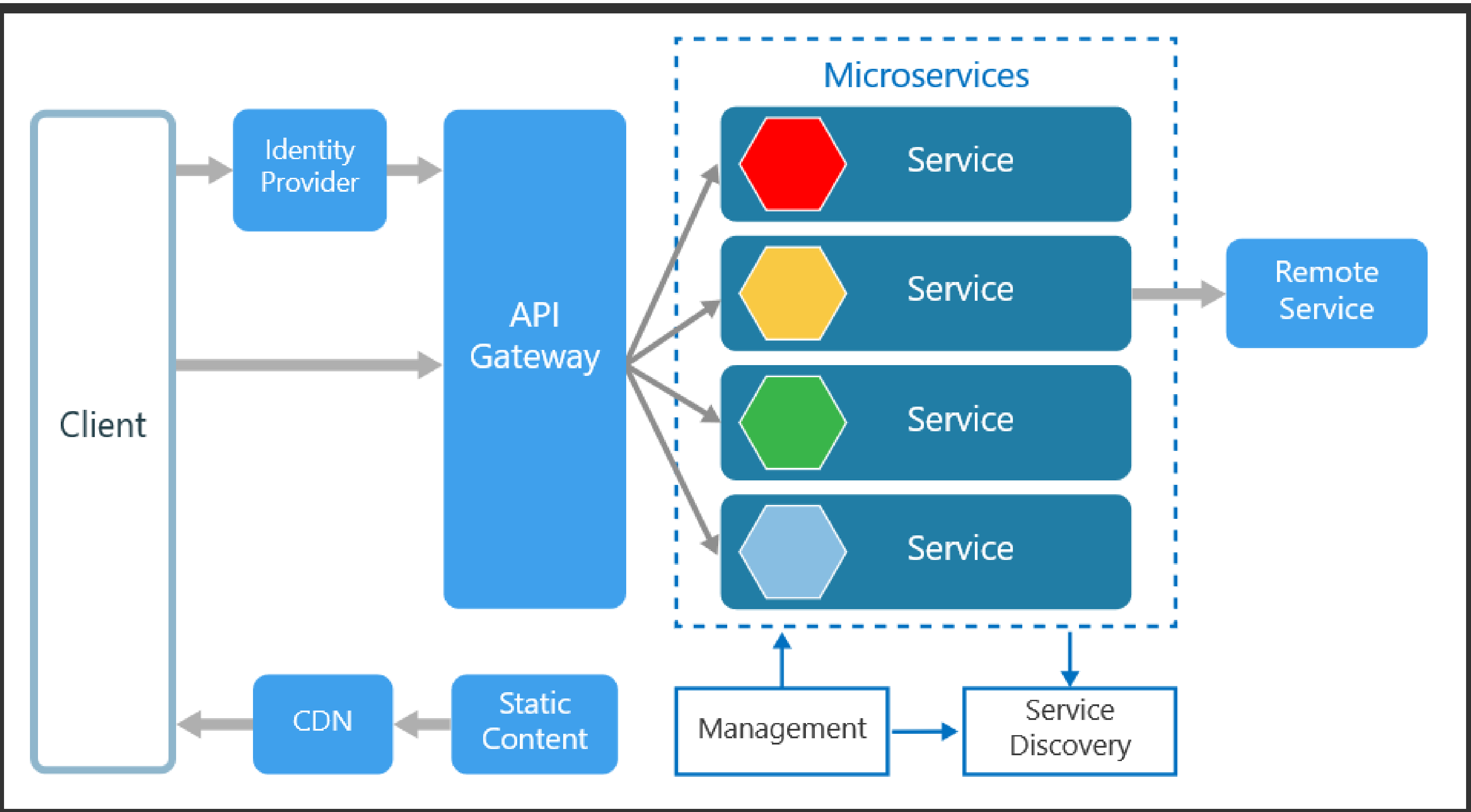
MICROSERVICES

- Microservices - also known as the microservice architecture –
  - It is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.
  - The microservice architecture enables the continuous delivery/deployment of large, complex applications.
  - It also enables an organization to evolve its technology stack.

MICROSERVICES

- In a microservices architecture, services are small, independent, and loosely coupled.
- Each service is a separate codebase, which can be managed by a small development team.
- Services can be deployed independently. A team can update an existing service without rebuilding and redeploying the entire application.
- Services are responsible for persisting their own data or external state. This differs from the traditional model, where a separate data layer handles data persistence.
- Services communicate with each other by using well-defined APIs. Internal implementation details of each service are hidden from other services.
- Services don't need to share the same technology stack, libraries, or frameworks.

MICROSERVICES



https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

NODE.JS + DOCKER

DOCKER

- Docker is a computer program that performs operating-system-level virtualization, also known as "containerization".
- It was first released in 2013 and is developed by Docker, Inc. Docker is used to run software packages called "containers".
- Docker provides container software that is ideal for developers and teams looking to get started and experimenting with container-based applications.
- Docker Desktop provides an integrated container-native development experience; it launches as an application from your Mac or Windows toolbar and provides access to the largest library of community and certified Linux and Windows Server content from Docker Hub.

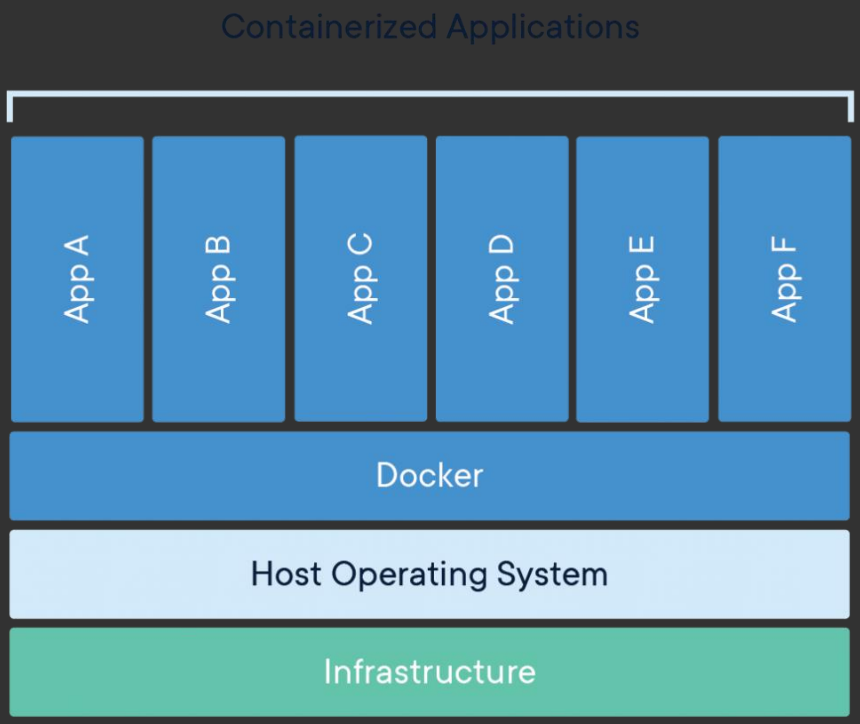
DOCKER, A DEVELOPERS' VIEW

- **Docker for Developers**
  - Building and deploying new applications is faster with containers.
  - Docker containers wrap up software and its dependencies into a standardized unit for software development that includes everything it needs to
    - run: code, runtime, system tools and libraries.
  - This guarantees that your application will always run the same and makes collaboration as simple as sharing a container image.

DOCKER, A DEVELOPERS' VIEW

- **Docker for Developers**
- Docker containers whether Windows or Linux are backed by Docker tools and APIs and help you build better software:
  - Onboard faster and stop wasting hours trying to set up development environments, spin up new instances and make copies of production code to run locally.
  - Enable polyglot development and use any language, stack or tools without worry of application conflicts.
  - Eliminate environment inconsistencies and the "works on my machine" problem by packaging the application, configs and dependencies into an isolated container.
  - Alleviate concern over application security

DOCKER CONTAINER



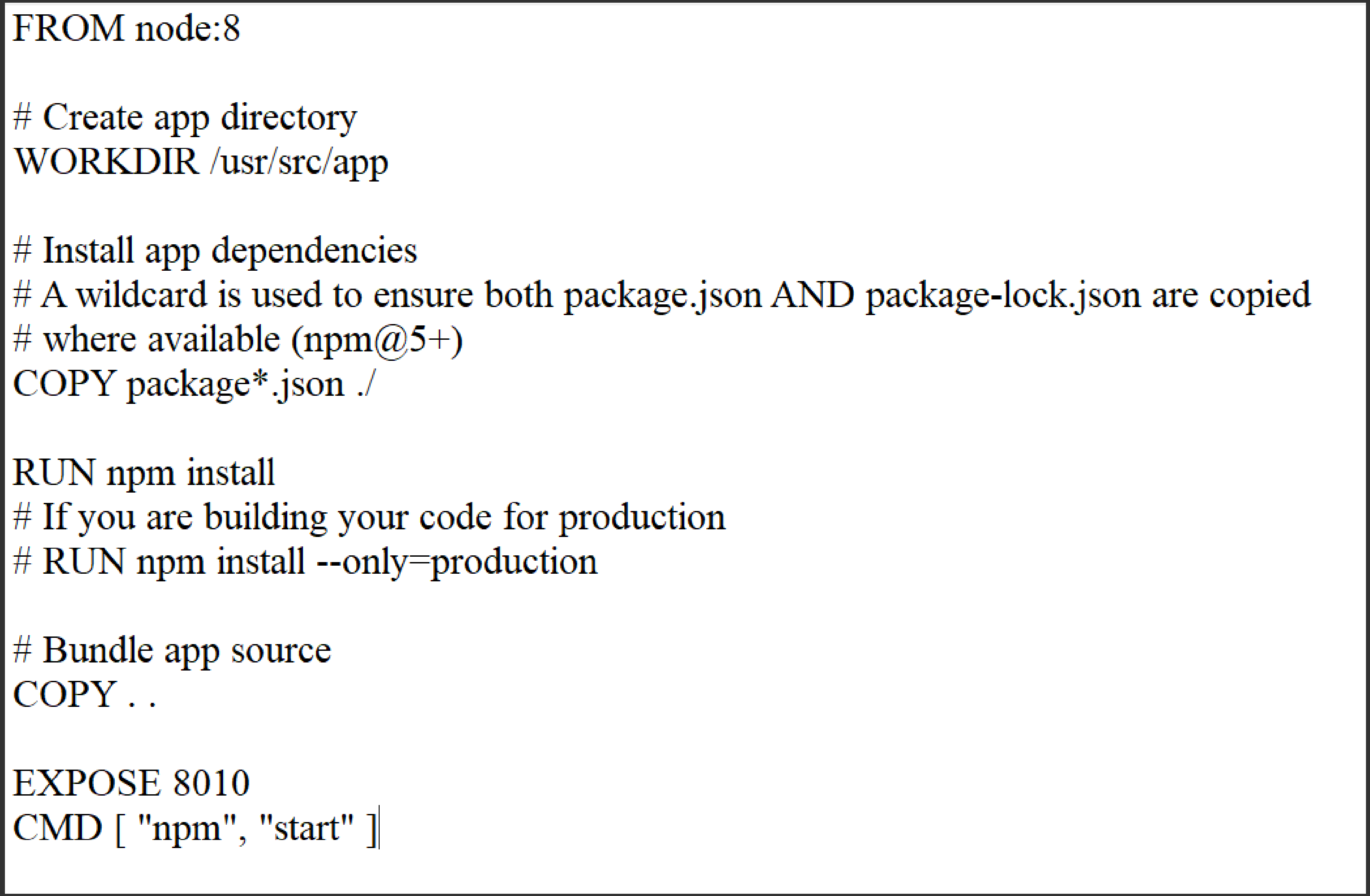
https://www.docker.com/resources/what-container

- A Docker container image is a
  - lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Docker containers that run on Docker Engine:

- **Standard:** Docker created the industry standard for containers, so they could be portable anywhere
- **Lightweight:** Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs
- **Secure:** Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry

DOCKER, PREPARATION NODE.JS



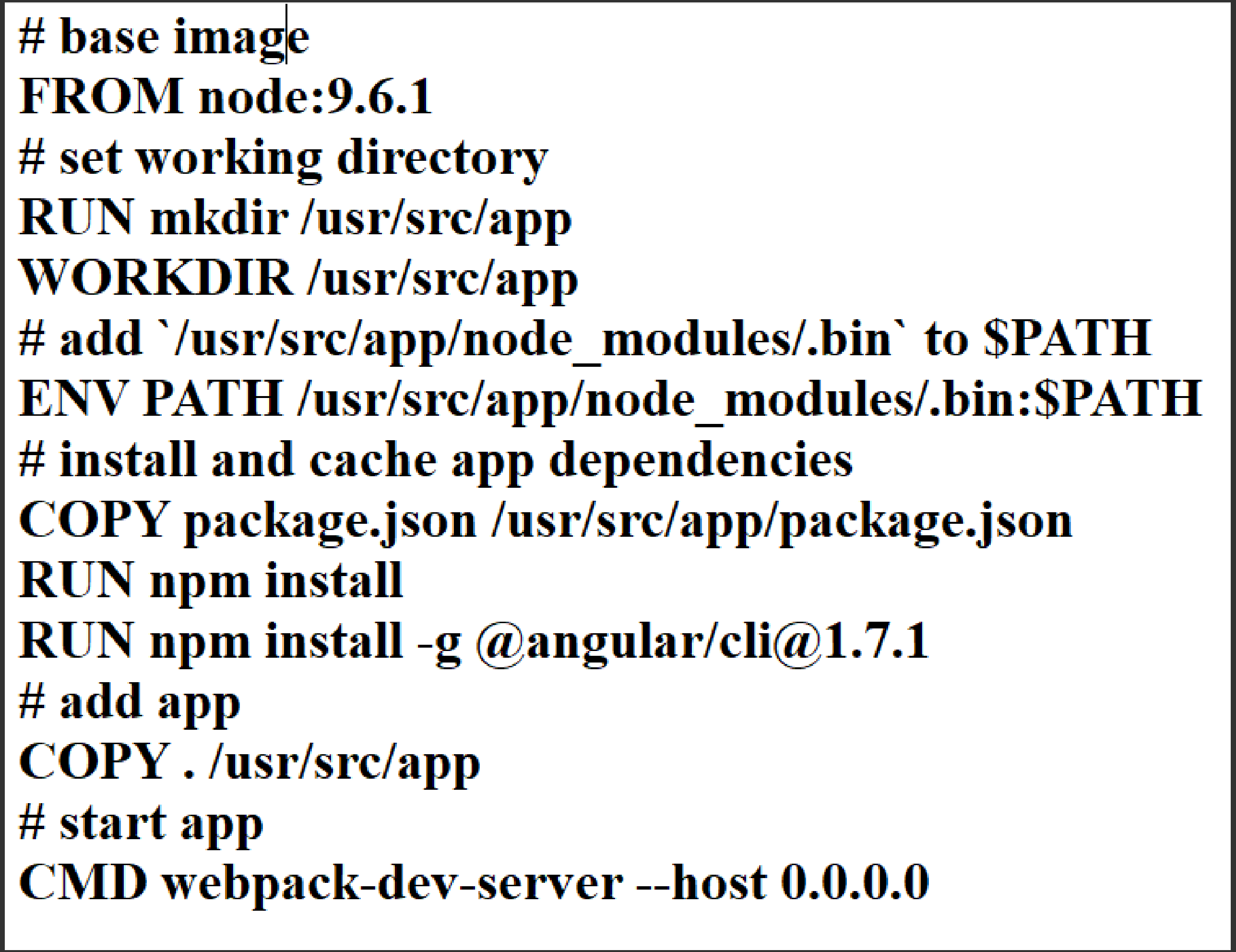
```
FROM node:8

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm install --only=production

# Bundle app source
COPY . .

EXPOSE 8010
CMD [ "npm", "start" ]
```

- Build the Docker image
  - docker build –t <Image_Name> .

- Run the docker container
  - docker run –p 8081:8081 <Image_Name>

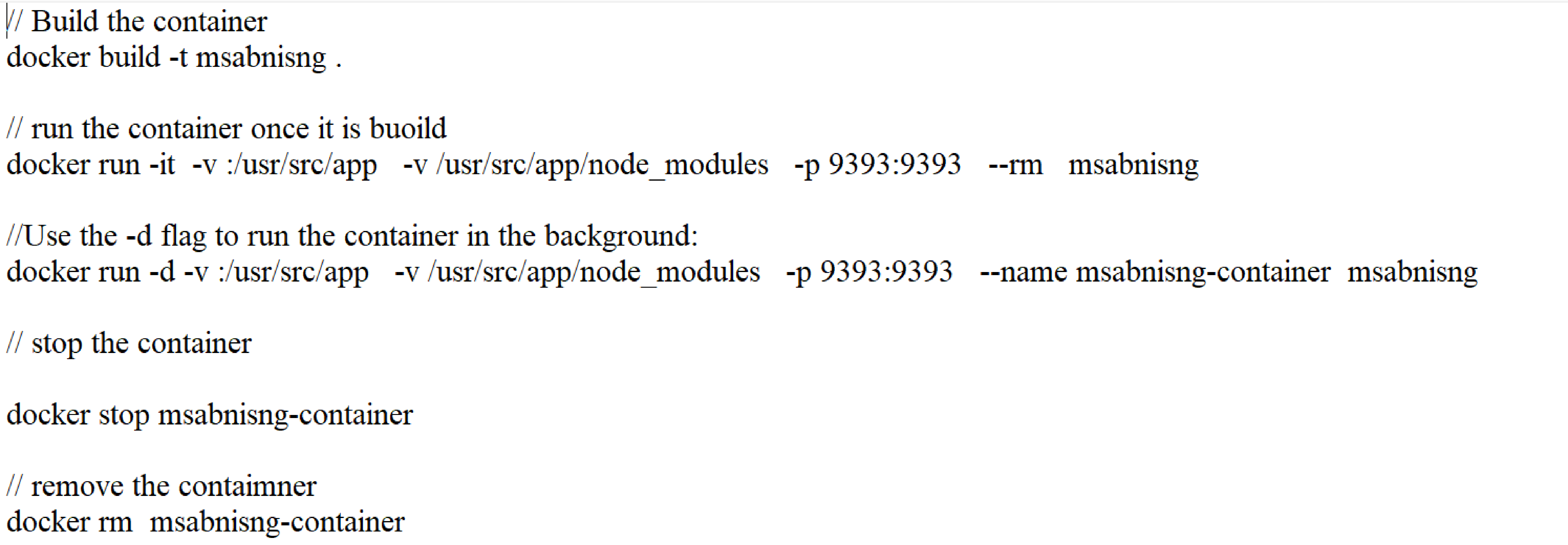
The **.dockerignore file**
node_modules
npm-debug.log

DOCKER, ANGULAR APPLICATION



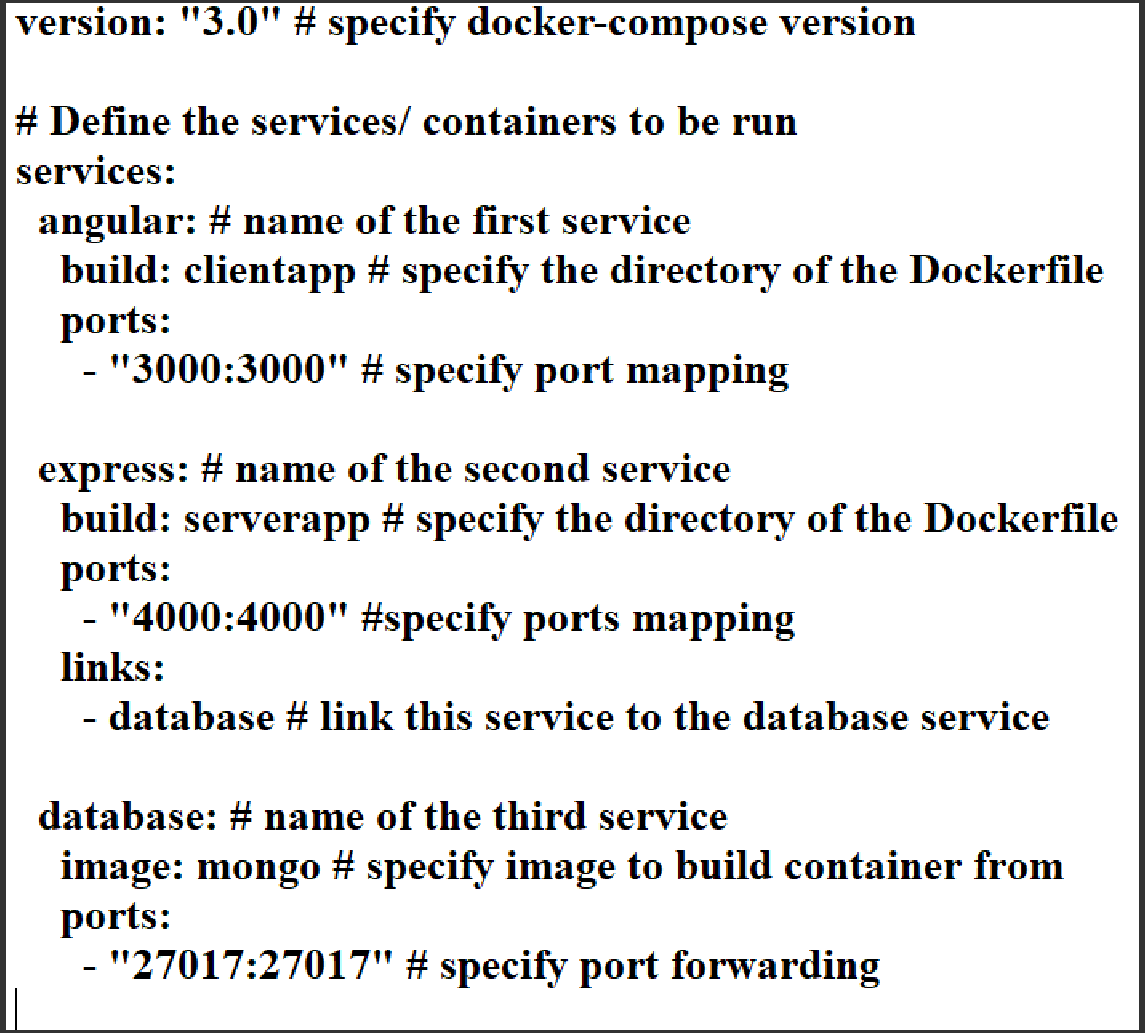
```
# base image
FROM node:9.6.1
# set working directory
RUN mkdir /usr/src/app
WORKDIR /usr/src/app
# add `/usr/src/app/node_modules/.bin` to $PATH
ENV PATH /usr/src/app/node_modules/.bin:$PATH
# install and cache app dependencies
COPY package.json /usr/src/app/package.json
RUN npm install
RUN npm install -g @angular/cli@1.7.1
# add app
COPY . /usr/src/app
# start app
CMD webpack-dev-server --host 0.0.0.0
```

DOCKER COMMANDS ANGULAR



```
// Build the container
docker build -t msabnisng .

// run the container once it is buoild
docker run -it  -v :/usr/src/app   -v /usr/src/app/node_modules   -p 9393:9393   --rm   msabnisng

//Use the -d flag to run the container in the background:
docker run -d -v :/usr/src/app   -v /usr/src/app/node_modules   -p 9393:9393   --name msabnisng-container  msabnisng

// stop the container

docker stop msabnisng-container

// remove the containmner
docker rm  msabnisng-container
```

DOCKER COMPOSING THE MEAN APPLICATION



```
version: "3.0" # specify docker-compose version

# Define the services/ containers to be run
services:
  angular: # name of the first service
    build: clientapp # specify the directory of the Dockerfile
    ports:
      - "3000:3000" # specify port mapping

  express: # name of the second service
    build: serverapp # specify the directory of the Dockerfile
    ports:
      - "4000:4000" #specify ports mapping
    links:
      - database # link this service to the database service

  database: # name of the third service
    image: mongo # specify image to build container from
    ports:
      - "27017:27017" # specify port forwarding
```

This will build multiple 3 Images One each for
Angular App
Express App
MongoDb

Commands
docker-compose build
docker-compose up

Note: Make sure that, the MongoDb Connection string will be

mongodb://database:27017/ProductsAppsDb

Here “database” is the “links” value from docker-compose.yml file