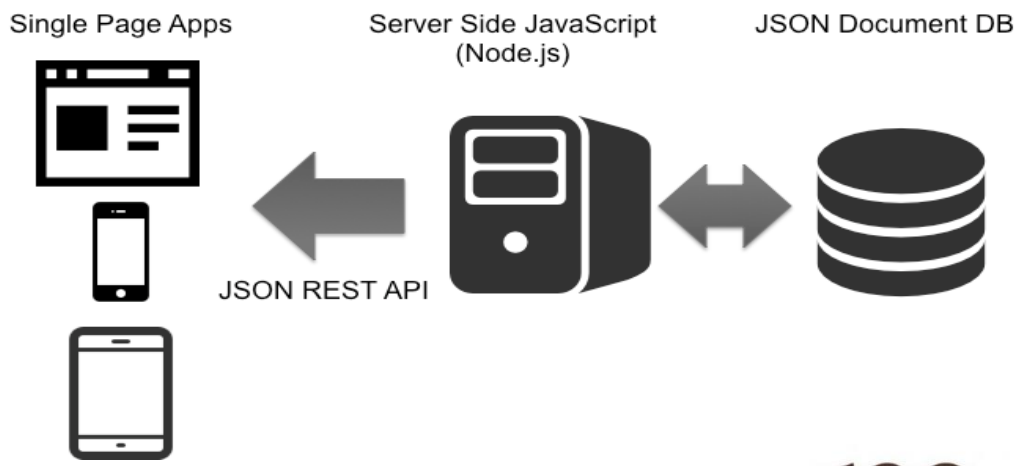

node
JS
TM

Acknowledgement

Some of the slides based on Node.js slides from http://www.slideshare.net/

What is Node.js



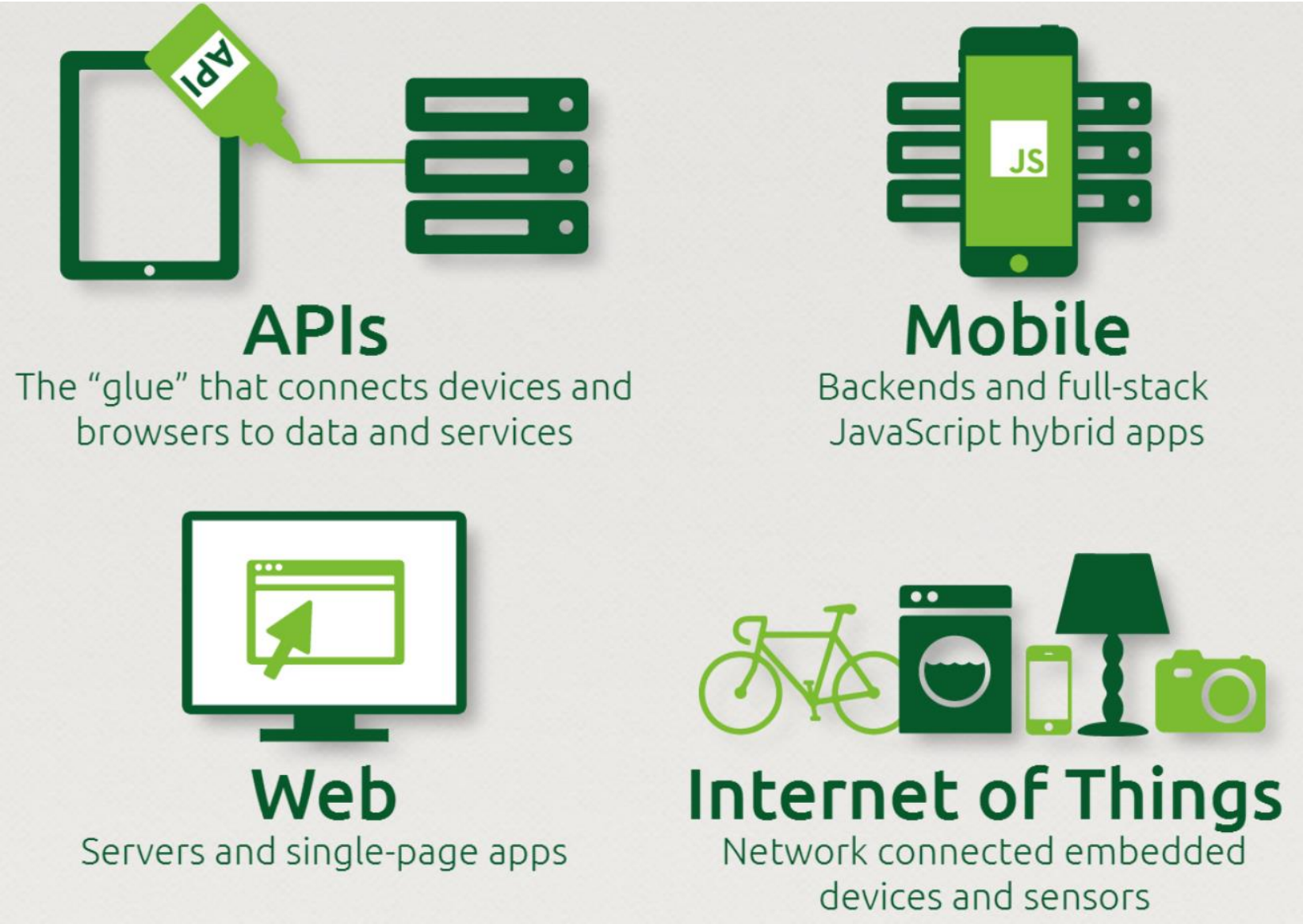
What is Node.js

- Cross platform environment for hosting JavaScript
- Offers end-to-end JavaScript environment
- Ideal for network & I/O based applications
- Non-blocking, event driven
- Open source
- Managed by
  - Node.js Foundation & Joyent
- https://nodejs.org

Why Node.js?

- It is light!
- It is fast!
- It is easy!
- Big community!
- It is fast to code and easy learn!
- It is JavaScript!
- It is scalable!
- It is cross-platform!

Node.js Key User Cases



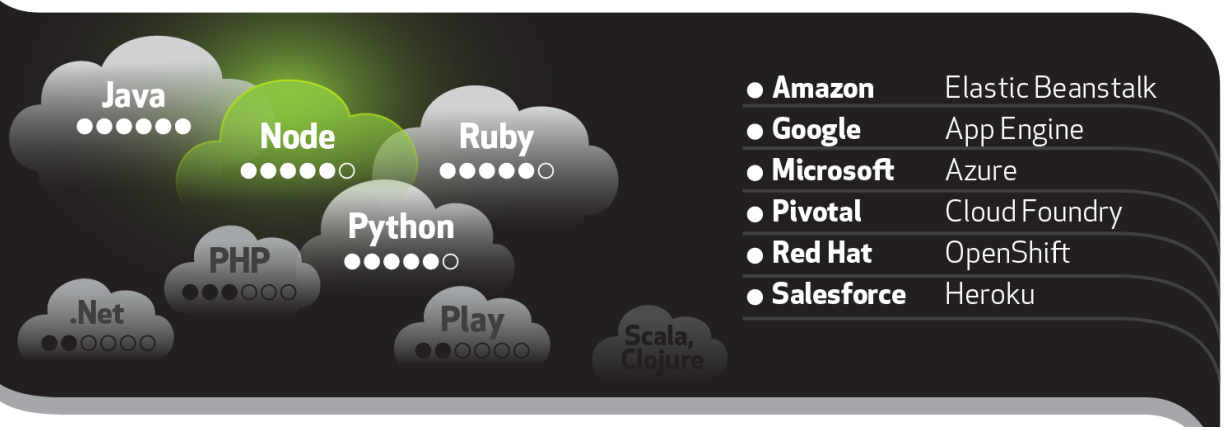
Source: https://strongloop.com/wp-content/uploads/2015/02/final-node-infographic-3-18-15v1.pdf

Who is using Node.js

Node.js in the Cloud

NODE IS TOP 4 IN THE CLOUD

Node.js is one of the top four languages, supported by 5 of the 6 major platform-as-a-service providers.



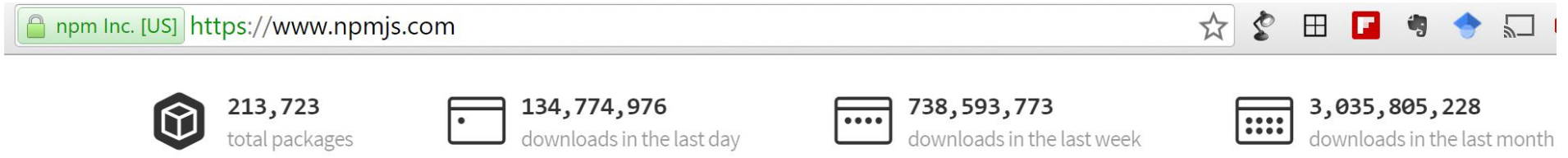
Amazon	Elastic Beanstalk
Google	App Engine
Microsoft	Azure
Pivotal	Cloud Foundry
Red Hat	OpenShift
Salesforce	Heroku

And It's Here To Stay

For source material and more information, visit **www.strongloop.com/infographic**

Why should you care!

https://www.npmjs.com/

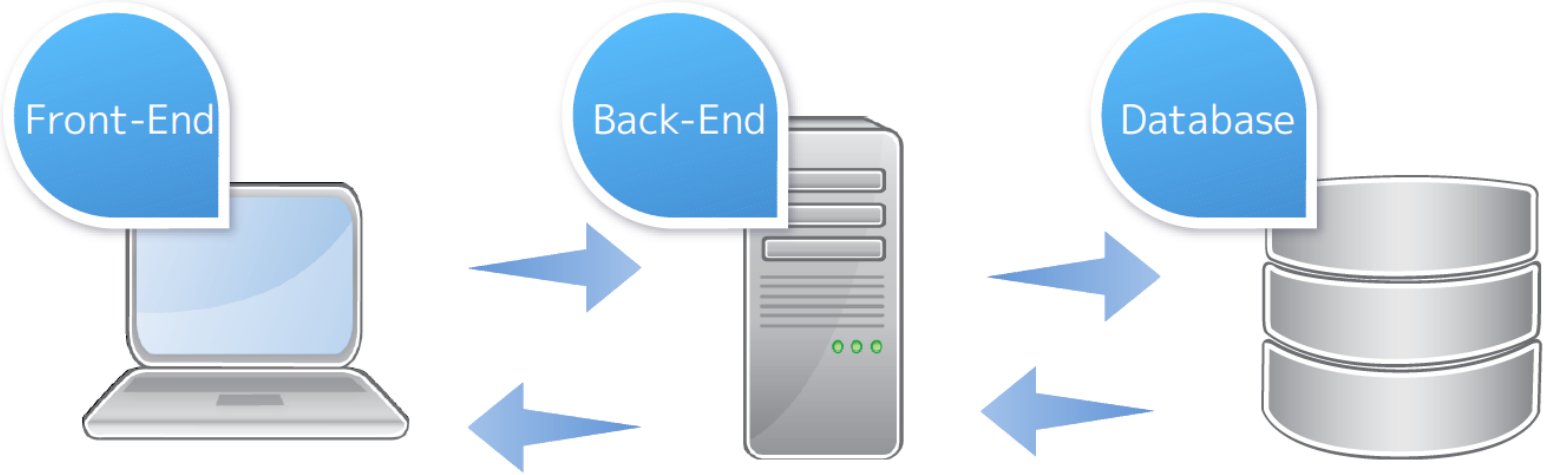


PayPal™

- 2x faster development with fewer developers
- 33% fewer lines of code
- 40% fewer files
- 2x improvement requests/sec
- 35% decrease in avg response time

"We are seeing big scale gains, performance boosts and big developer productivity."

LAMP Stack



Javascript

Linux

Apache

Perl , PHP, Python

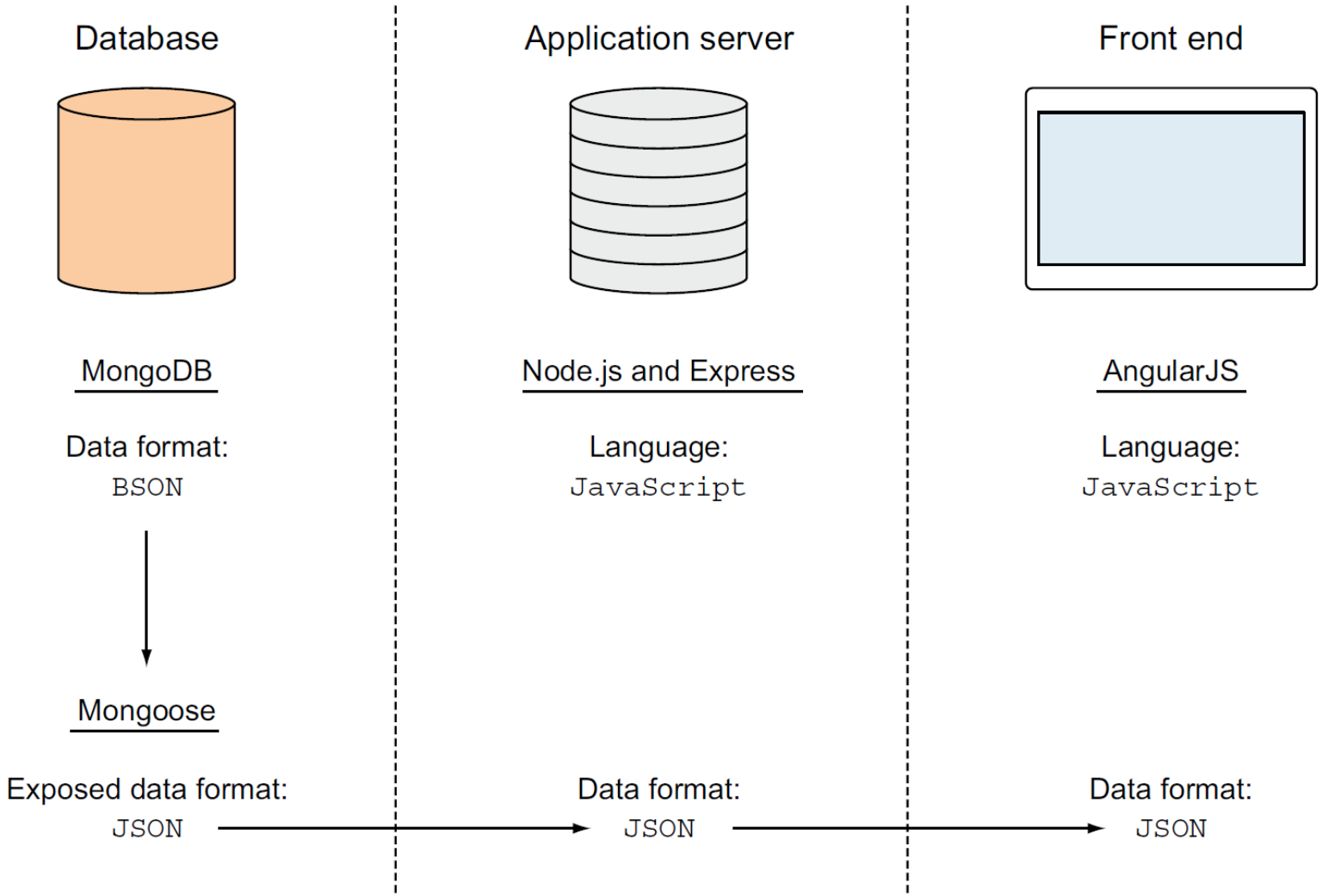
MySQL

Data Transformation problem

Three Languages problem

Scale-out problem

JavaScript is the common language throughout the mean stack, and JSON is the common data format

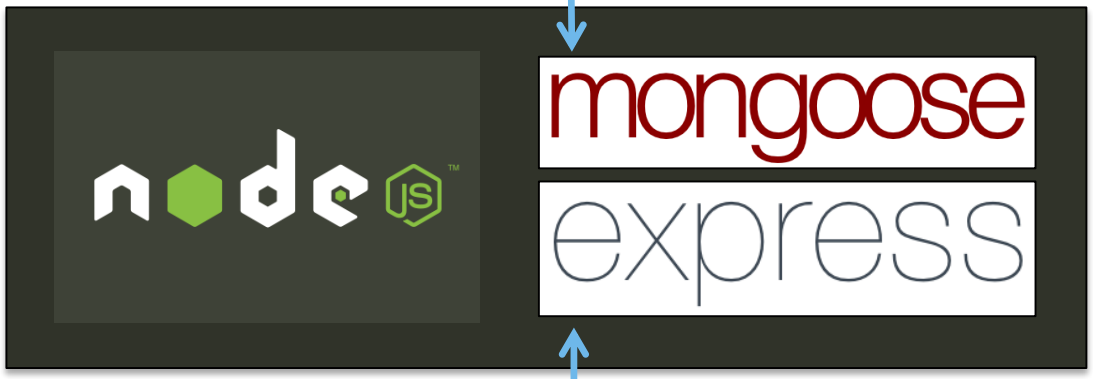


Node.js **plays such a pivotal role in the MEAN (Mongo, Express, Angular, and Node) stack** (alternative to LAMP)

MEAN Stack

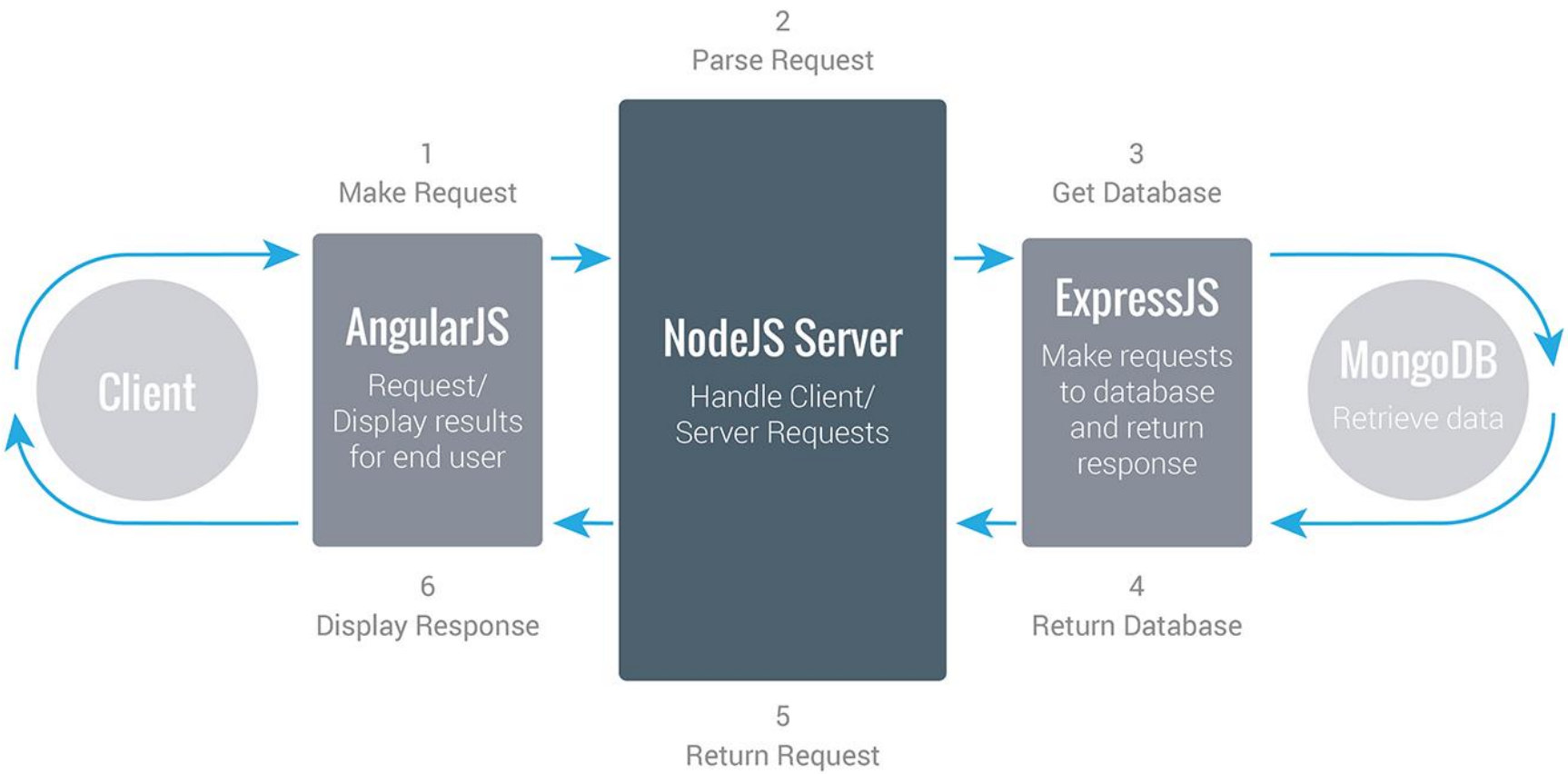
Database

Server



Client

Mean ReCAP


2
Parse Request
1
Make Request
3
Get Database
Client
AngularJS
Request/
Display results
for end user
NodeJS Server
Handle Client/
Server Requests
ExpressJS
Make requests
to database
and return
response
MongoDB
Retrieve data
6
Display Response
4
Return Database
5
Return Request

Node.JS Advantages

Architecture

- Single Thread
- App == Server
- Middleware

Deployment

- XCopy
- Run Everywhere

Community

- 200,000+ Packages
- 130M+ Download in day.

Tools

METEOR

PhoneGap

Gulp

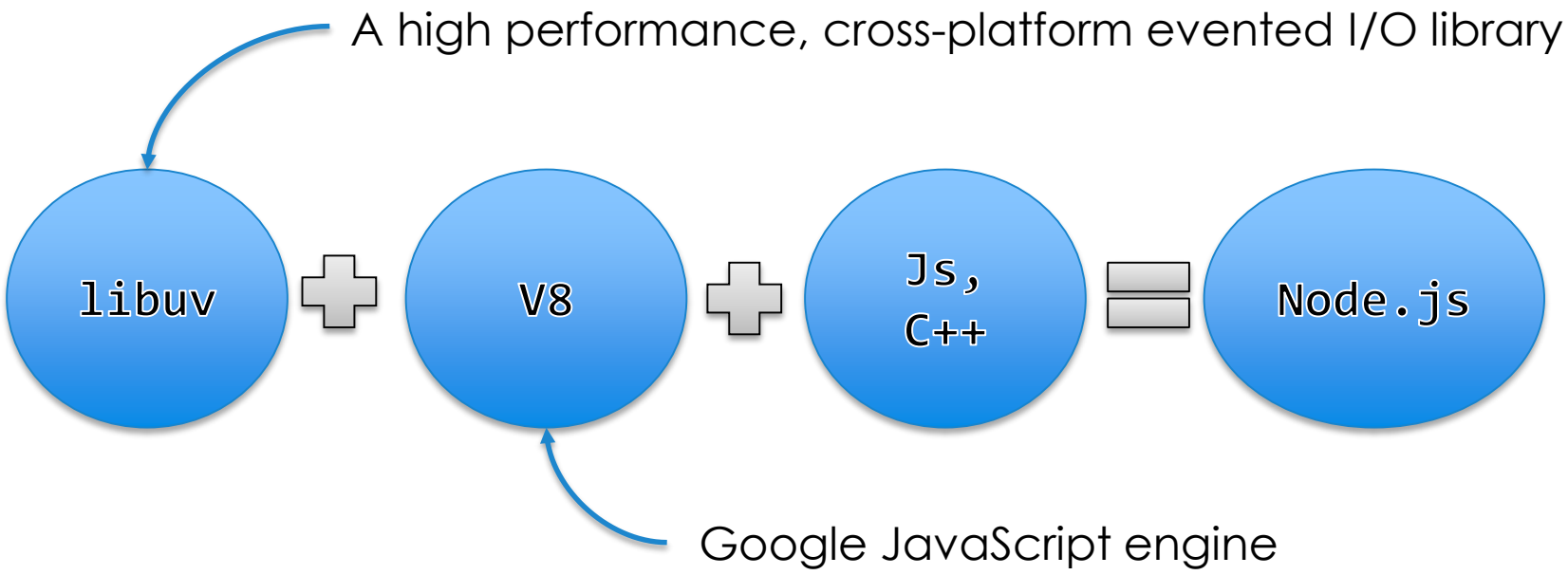
YEOMAN

GRUNT

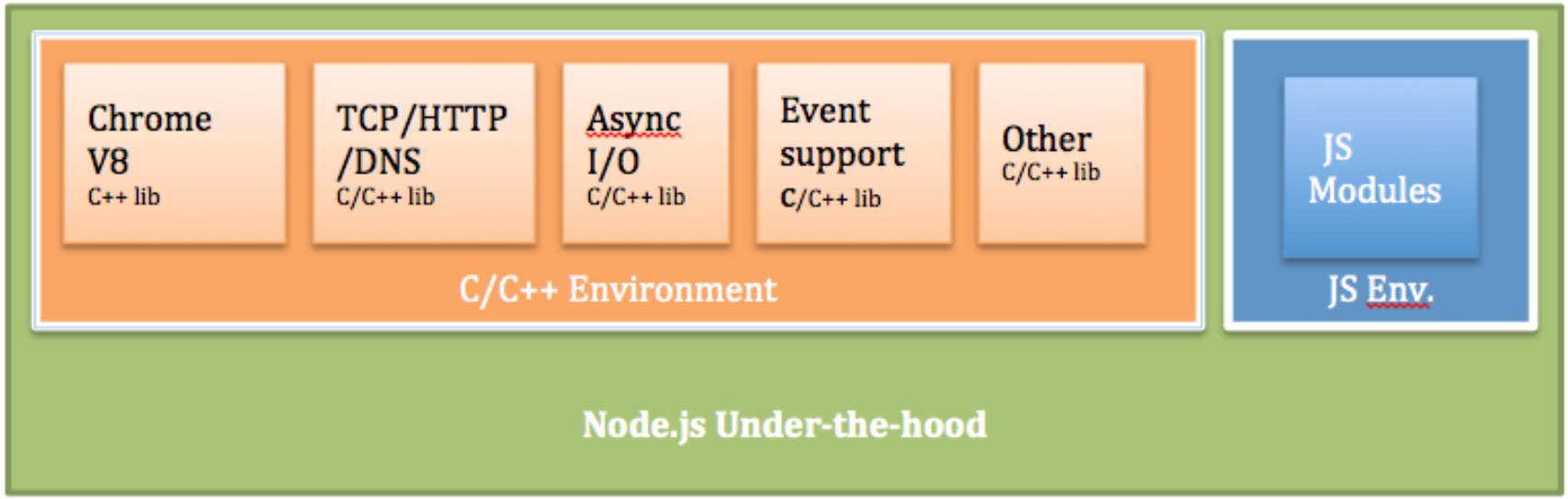
BOWER

Node.js Architecture

Node.js Building Blocks

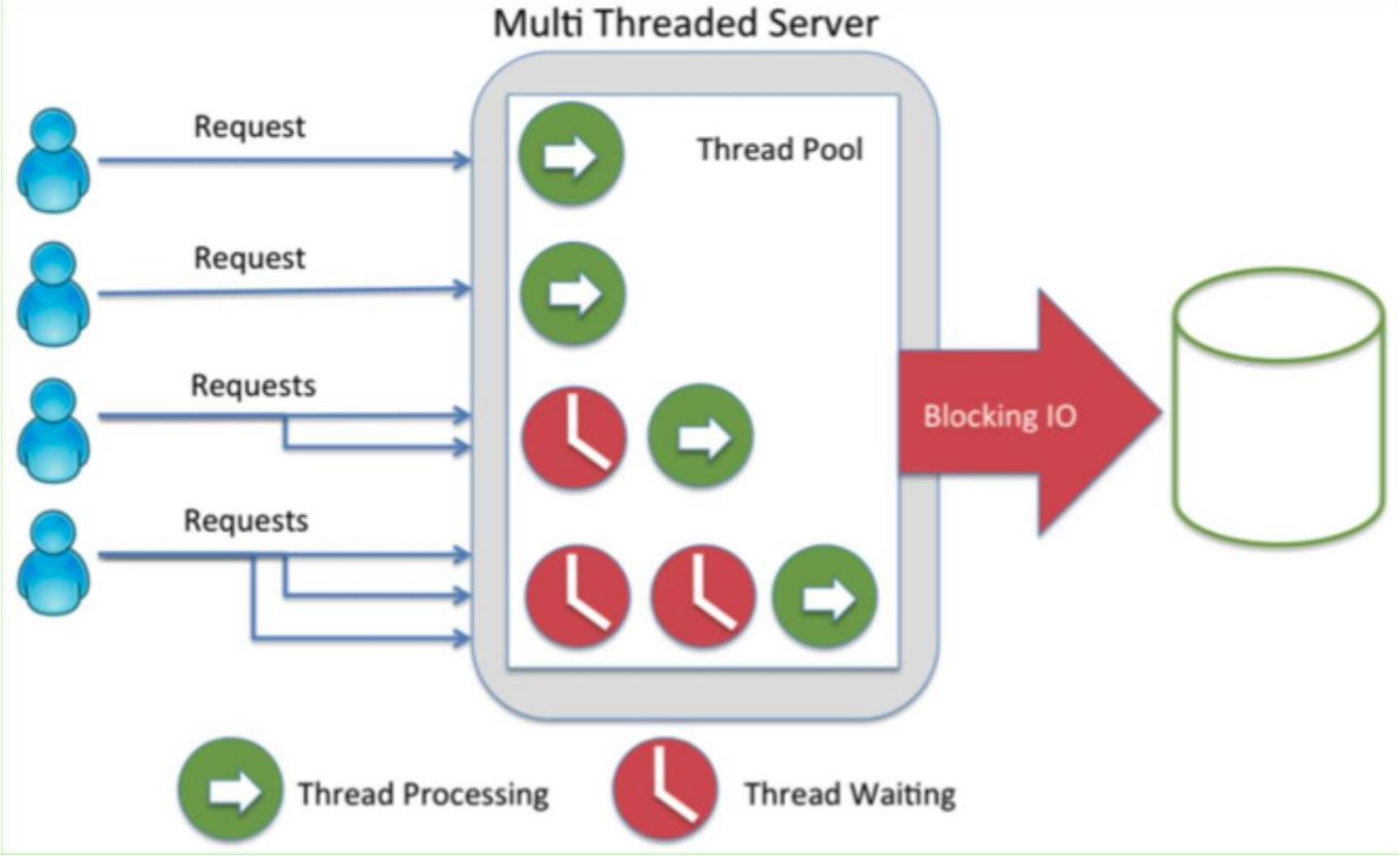


Node.js Under The Hood



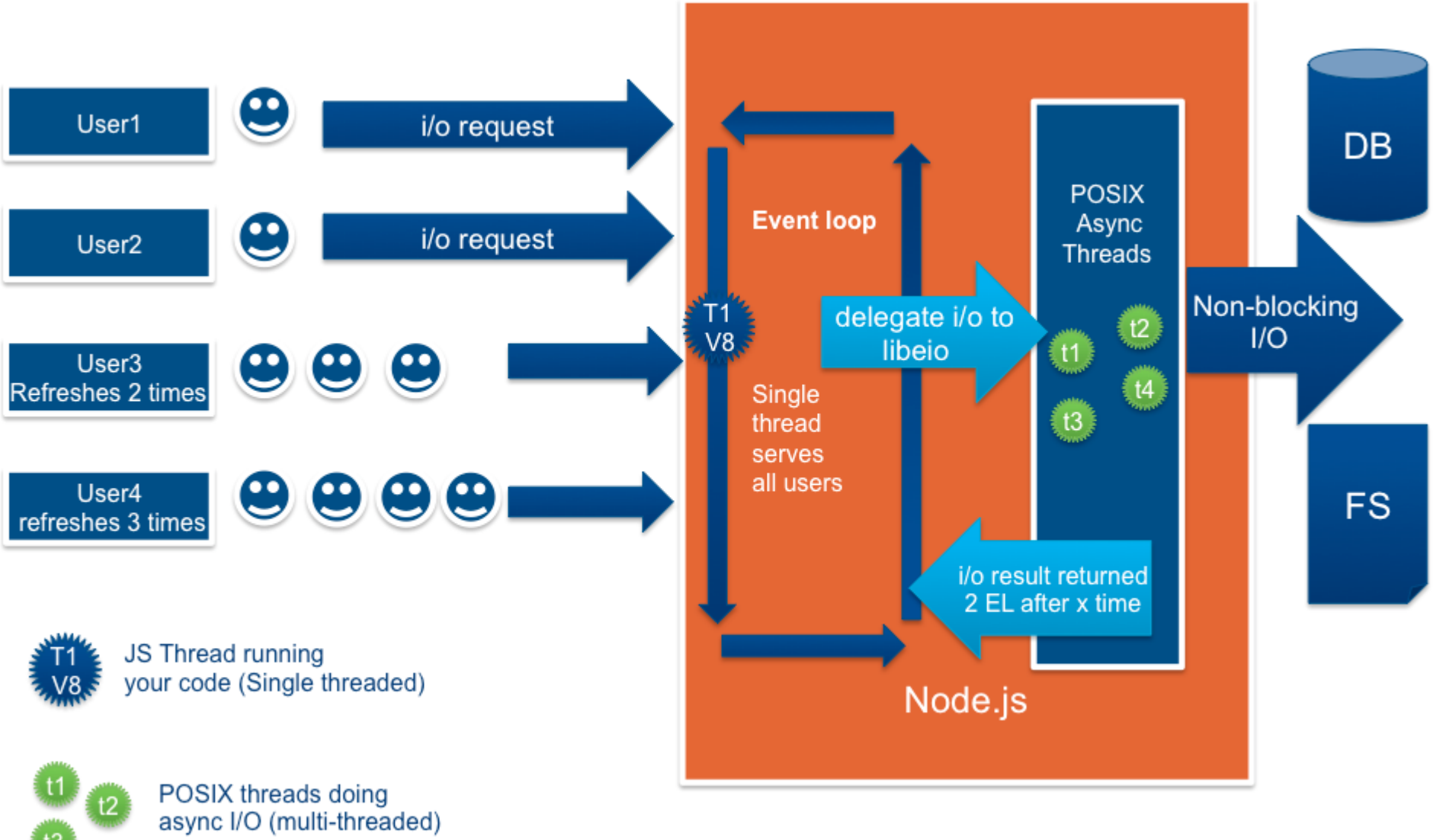
(source: Future-proofing Your Apps: Cloud Foundry and Node.js)

Multi-threaded HTTP Server Using Blocking I/O



(source: strongloop.com)

Node.js uses Event-driven, Non-Blocking I/O



(source: Future-proofing Your Apps: Cloud Foundry and Node.js)

Node.js Basics

- Modules
- Asynchronous
- Non-blocking I/O
- Event-driven

Modules System

Modules

- An elegant way of encapsulating and reusing code
- Node has a simple module loading system.
  - Files and modules are in one-to-one correspondence.

foo.js

```
var circle = require('./circle.js');
console.log('The area of radius 4: '+ circle.area(4));
```

circle.js

```
var PI = Math.PI;
exports.area = function (r) {return PI * r * r;};
exports.circumference = function (r) {return 2 * PI * r;};
```

The variable **PI** is **private** to circle.js

File Modules

- A module prefixed:
  - '/' is an absolute path to the file.
  - './' is relative to the file calling require().
  - Without a leading '/' or './' to indicate a file, the module is either a "**core module**" or is loaded from a **node_modules** folder.

Anatomy of a module

```
var privateVal = 'I am Private!';

module.exports = {
   answer: 42,

   add: function(x, y) {
            return x + y;
   }
}
```

mymodule.js

Usage

```
var mod = require('./mymodule');

console.log('The answer: '+ mod.answer);

var sum = mod.add(4,5);
console.log('Sum: ' + sum);
```

Asynchronous

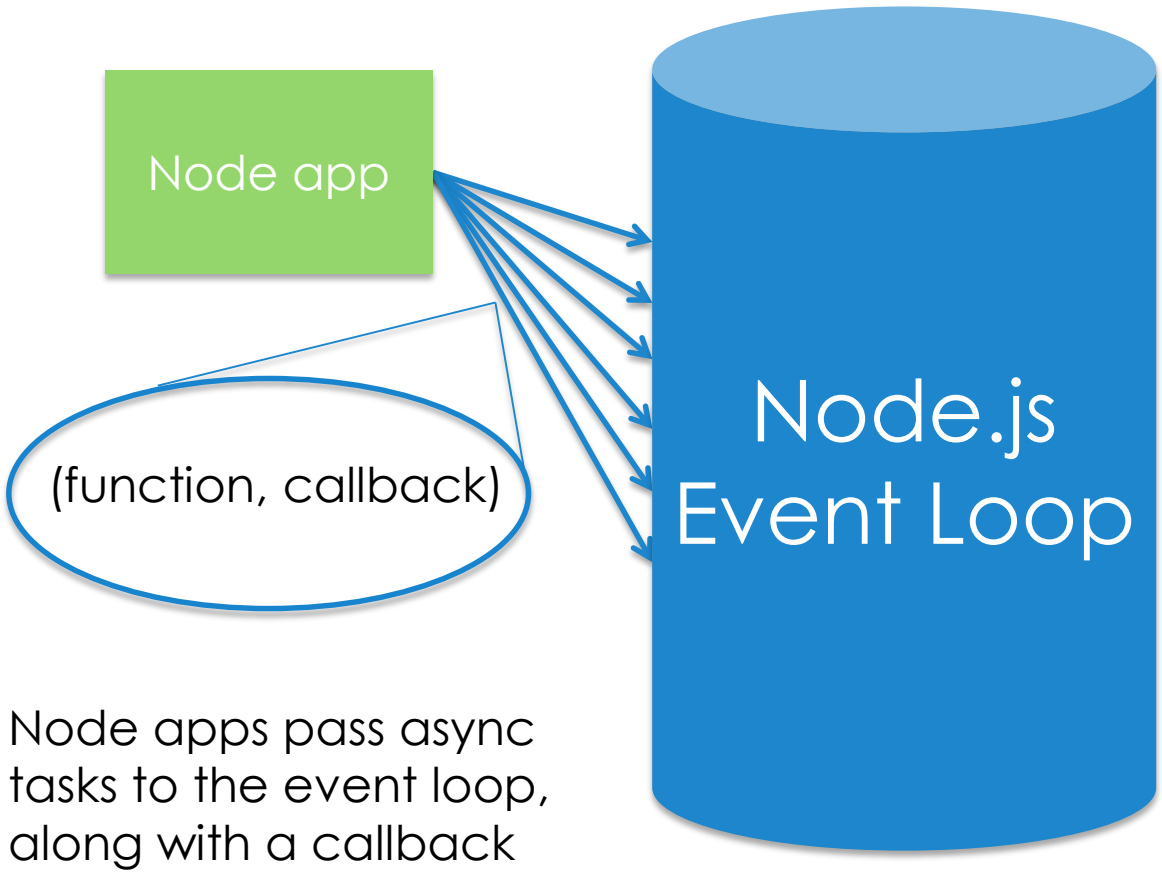
Events

Streams

Asynchronous Programming

- Node is entirely asynchronous
- You have to think a bit differently
- Failure to understand the event loop and I/O model can lead to anti-patterns
- Keep in mind - Your Node.js app is single-threaded

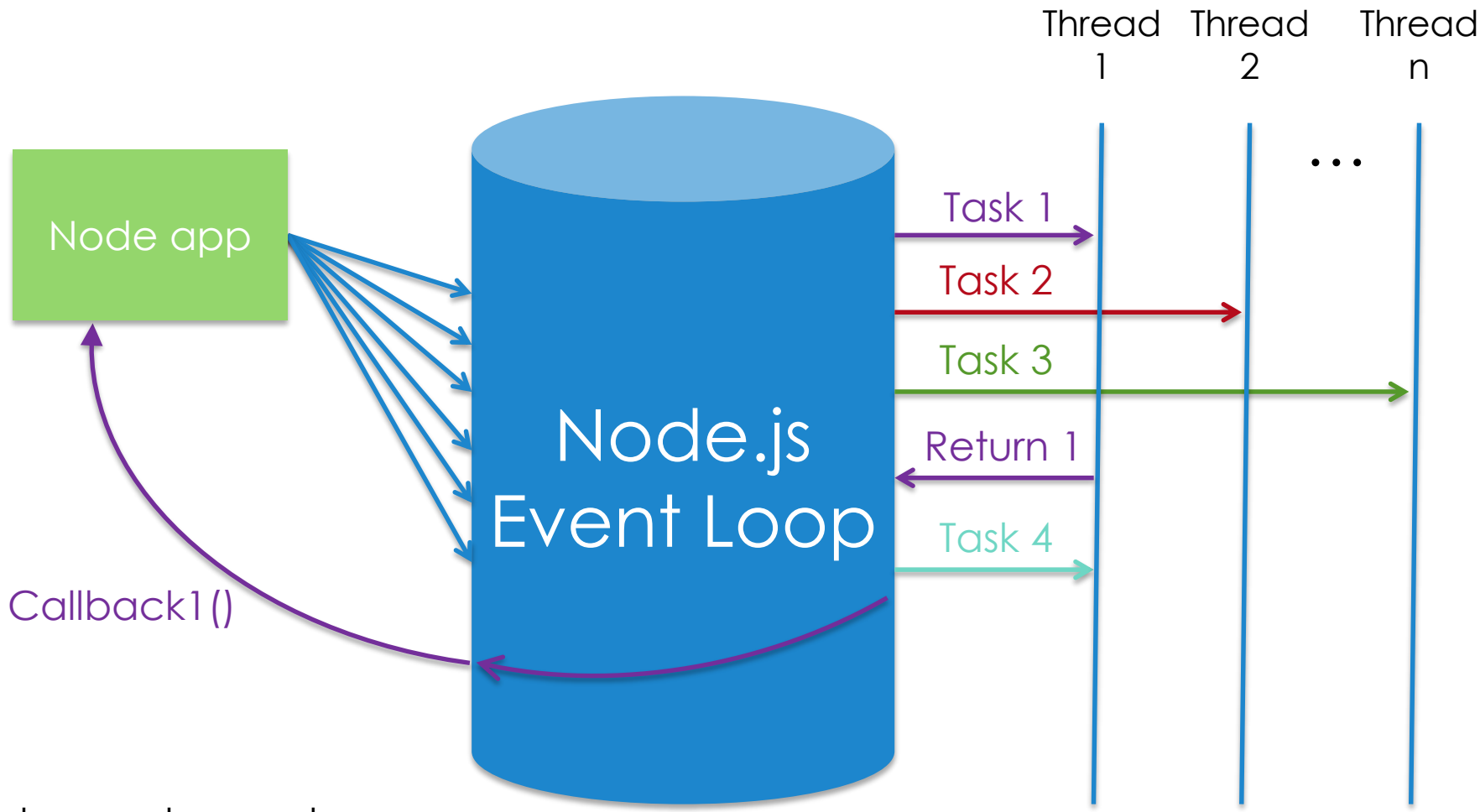
Event Loop



Node apps pass async tasks to the event loop, along with a callback

Event Loop

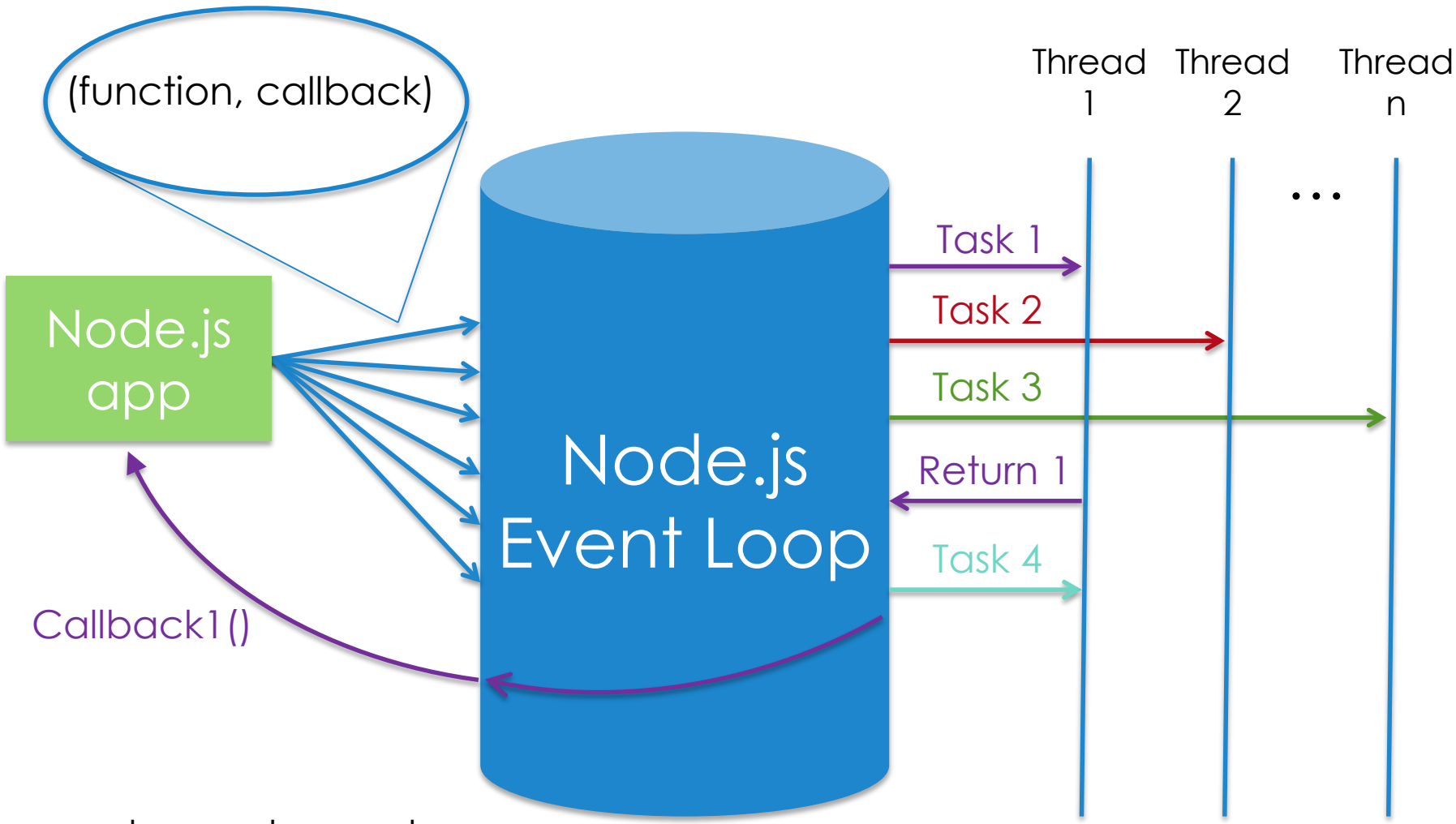
The event loop efficiently manages a thread pool and executes tasks efficiently…



…and executes each callback as tasks complete

1 Node apps pass async tasks to the event loop, along with a callback

2 The event loop efficiently manages a thread pool and executes tasks efficiently…



3 …and executes each callback as tasks complete

Async I/O

The following tasks should be done asynchronously, using the event loop:

- I/O operations
- Heavy computation
- Anything requiring blocking

Anti-pattern: Synchronous Code

Your app only has one thread, so:

```
for (var i = 0; i < 100000; i++){
    // Do anything
}
```

…will bring your app to a grinding halt

Anti-pattern: Synchronous Code

Do not do this in Node.js:

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

for (var i = 0; i < files.length; i++){
    data = fs.readFileSync(files[i]);
  console.log(data);
}
```

…and it will cause severe performance problems

Based on examples from: https://github.com/nodebits/distilled-patterns/

Pattern: Async I/O

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

fs.readFile('f1.txt', 'utf8', function(err,data) {
    if (err) {
        // handle error
    }
    console.log(data);
});
```

Anonymous, inline callback

File System & Streams

The File System

- **__filename** :

  The absolute path of the currently executing file.

- **__dirname** :

  The absolute path to the directory containing the currently executing file.

Process Object

- **`process.cwd()`**
  The Current Working Directory.

- **`process.chdir("/")`**
  Changing the Current Working Directory.

- **`process.execPath`**
  Locating the nodeExecutable.

The fs Module

- Node applications perform file I/O via the **fs** module, a core module whose methods provide wrappers around standard file system operations

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

Stream Events

- **data Events**
  Indicate that a new piece of stream data, referred to as a chunk, is available.
- **end Event**
  Once a stream sends all of its data, it should emit a single end event.
- **close Event**
  indicate that the underlying source of the stream data has been closed.
- **error Events**
  indicate that a problem occurred with the data stream.

File Streams

- createReadStream()

```
var fs = require("fs");
var stream;
stream = fs.createReadStream(__dirname + "/foo.txt");

stream.on("data", function (data) {
    var chunk = data.toString();
    process.stdout.write(chunk);
});

stream.on("end", function() {
    console.log();
});
```

File Streams

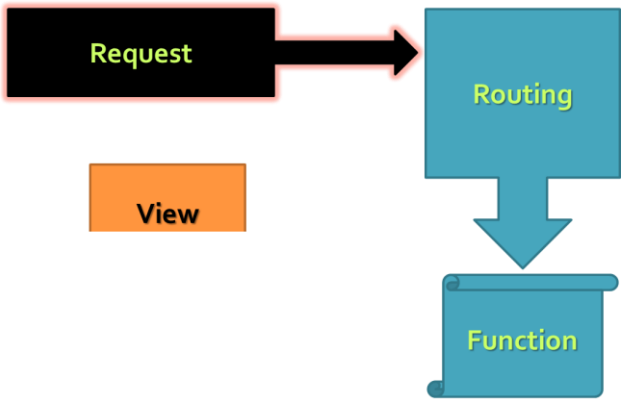
- createWriteStream()

```
var fs = require("fs");
var readStream = fs.createReadStream(__dirname + "/foo.txt");
var writeStream = fs.createWriteStream(__dirname + "/bar.txt");
readStream.pipe(writeStream);
```

Compressing a File Using Gzip Compression

```
var fs     = require("fs");
var zlib   = require("zlib");
var gzip   = zlib.createGzip();
var input  = fs.createReadStream("input.txt");
var output = fs.createWriteStream("input.txt.gz");

input
 .pipe(gzip)
 .pipe(output);
```

Express Web Application Framework for Node.js



A Web Server without Express

```
// Require what we need
var http = require("http");

// Build the server
var app = http.createServer(
    function (request, response) {
        response.writeHead(200, {
                    "Content-Type": "text/plain"
                });
                response.end("Hello world!");
        });

// Start that server
app.listen(1337, "localhost");
console.log("Server running at http://localhost:1337/");
```

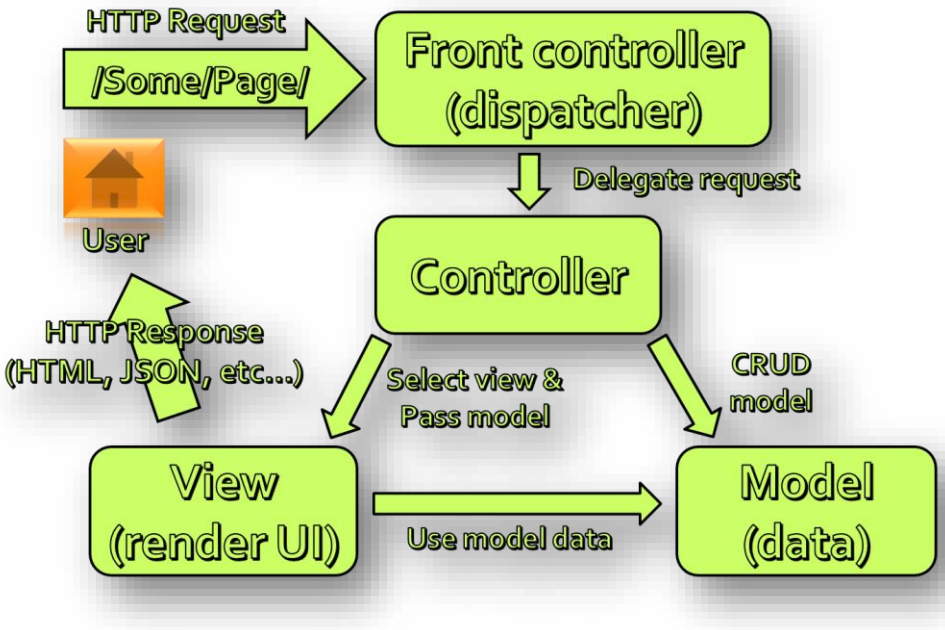
Request Handling

How to Route Without Express

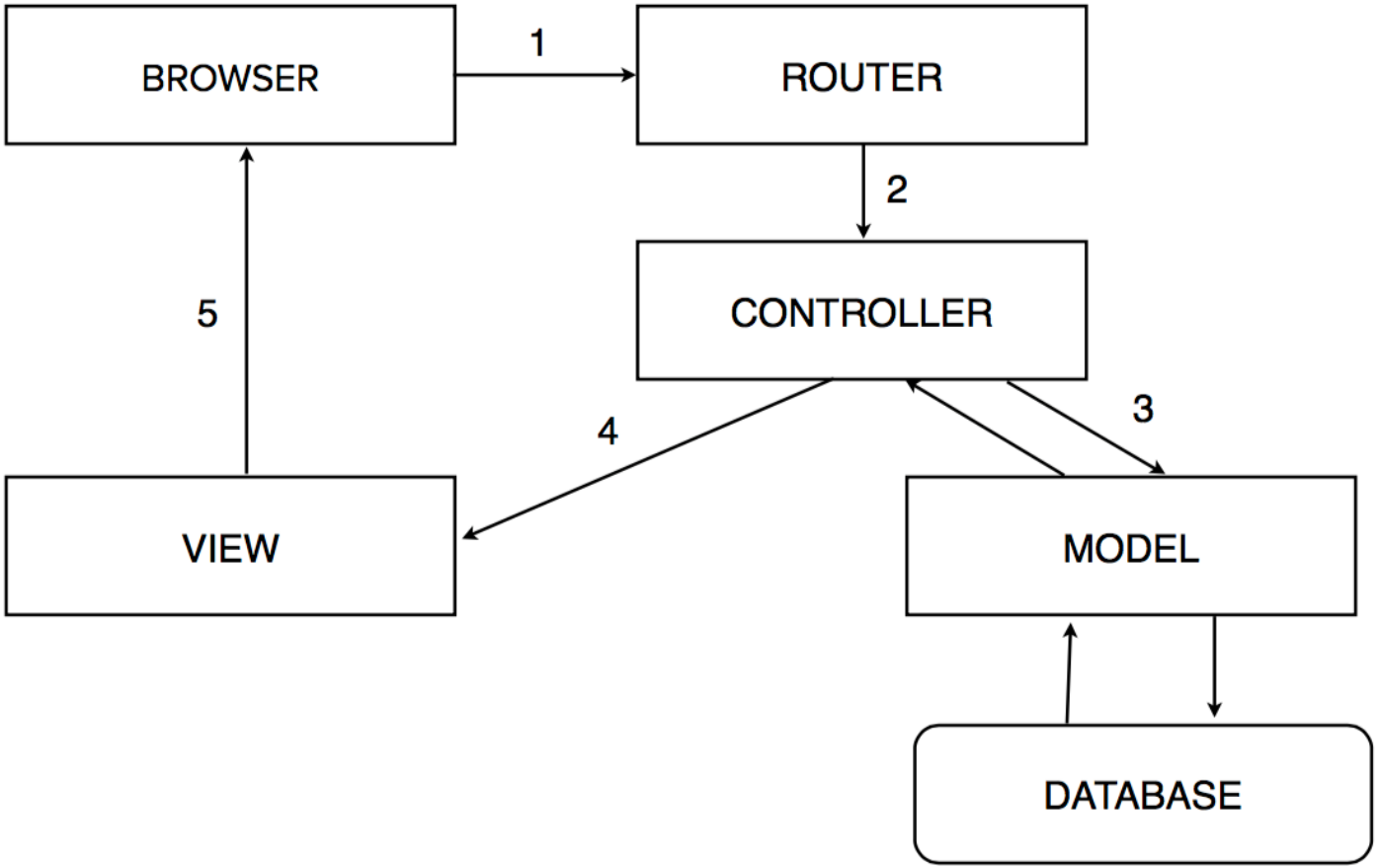
```
var http = require("http");
http.createServer(function (req, res) {

      // Homepage
      if (req.url == "/") {
          res.writeHead(200, { "Content-Type": "text/html" });
          res.end("Welcome to the homepage!");
      }// About page
      else if (req.url == "/about") {
          res.writeHead(200, { "Content-Type": "text/html" });
          res.end("Welcome to the about page!");
      }
      // 404'd!
      else {
          res.writeHead(404, { "Content-Type": "text/plain" });
          res.end("404 error! File not found.");
      }

}).listen(1337, "localhost");
```

Express.js

- Adds functionality to the normal server
  - Request / Response enhancements
  - Routing
  - View Support
  - HTML Helpers
  - Content Negotiation



Model View Controller



First Express App

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

var app = express();

app.get('/', function (request, response) {
    response.send('Welcome to Express!');
});

app.get('/customer/:id', function (req, res) {
    res.send('Customer requested is ' + req.params['id']);
});

app.listen(3000);
```

Express Routing

Express Routing

```
var express = require("express");
var http = require("http");
var app = express();

app.all("*", function (request, response, next) {
    response.writeHead(200, { "Content-Type": "text/plain" });
    next(); });

app.get("/", function (request, response) {
    response.end("Welcome to the homepage!"); });

app.get("/about", function (request, response) {
    response.end("Welcome to the about page!"); });

app.get("*", function (request, response) {
    response.end("404!"); });

http.createServer(app).listen(1337);
```

Express Routing

```
app.get('/users/:id?', function (req, res, next)
{
    var id = req.params.id;
    if (id) {
    // do something
    } else {
        next();
    }
});
```

```
{
    path: '/user/:id?',
    method: 'all' | 'get' | 'post' | 'put' | 'delete',
    callbacks: [ [Function] ],
    keys: [ { name: 'id', optional: true } ],
    regexp: /^\/user(?:\/([^\/]+?))?\/?$/i,
    params: [ id: '12' ]
}
```

Routing and Module

```
// home.js file in Routing folder.
module.exports = function (app) {
    // home page
    app.get('/', function (req, res) {
        res.render('index', { title: 'Home Page.  ' })
    });
    // about page
    app.get('/about', function (req, res) {
        res.render('about', { title: 'About Me.  ' })
    });
}
```

```
var express = require("express");
var http    = require("http");
var app     = express();

// Include a route file
require('./routes/home')(app);

http.createServer(app).listen(1337);
```

Routing Templates

- One to one

```
app.get('/users/:id?', function (req, res, next) { ... }
```

- One to many

```
'/:controller/:action/:id'
```

Configuration

- Conditionally invoke callback when **env** matches app.get('env'), aka **process.env.NODE_ENV**.

```
// all environments
app.configure(function() {
    app.set('title', 'My Application');
});

// development only
app.configure('development', function() {
    app.set('db uri', 'localhost/dev');
});

// production only
app.configure('production', function() {
    app.set('db uri', 'n.n.n.n/prod');
});
```

Http Methods

HTTP Methods

- app.get(), app.post(), app.put() & app.delete()

- By default Express does not know what to do with this request body, so we should add the ***bodyParser*** middleware

- ***bodyParser*** will parse ***application/x-www-form-urlencoded*** and ***application/json*** request bodies and place the variables in ***req.body***

```
app.use( express.bodyParser() );
```

Post Sample

```
<form method="post" action="/">
  <input type="hidden" name="_method" value="put" />
  <input type="text" name="user[name]" />
  <input type="text" name="user[email]" />
  <input type="submit" value="Submit" />
</form>
```

```
app.use(express.bodyParser());
```

```
app.post('/', function(){
    console.log(req.body.user);
    res.redirect('back');
});
```

Error Handling

- Express provides the ***app.error()*** method which receives exceptions thrown within a route, or passed to *next(err)*.

```
app.error(function (err, req, res, next) {
    if (err instanceof NotFound) {
        res.render('404.jade');
    } else {
        next(err);
    }
});
```

errorHandler Middleware

- Typically defined very last, below any other app.use() calls.

```
app.use(express.bodyParser());
app.use(app.router);
app.use(function(err, req, res, next){
  // logic
});
```

Views

Views in ExpressJS

- User Interface
- Based on Templates
- Support for multiple View Engines
  - Jade, EJS, JSHtml, . . .
- Default is Jade
  - http://jade-lang.com

```
app.get('/', function (req, res) {
    res.render('index');
});
```

Views in ExpressJS – Example

```
var express = require('express'),
    path = require('path');
var app = express();
app.configure(function () {
    app.set('views', __dirname + '/views');
    app.set('view engine', 'jade');
    app.use(express.static(path.join(__dirname, 'public')));
});
app.get('/', function (req, res) {
    res.render('index');
});
app.listen(3000);
```

```
doctype
html(lang="en")
  head
    title Welcome to this emtpy page
  body
```

Working with Data

- Pass data to the views

```
res.render('index', { title: 'Customer List' });
```

- Read data from the views (bodyParser)

```
let username = req.body.username;
```

- Read and send files

```
var filePath = req.files.picture.path;
// ...
res.download(filePath);
res.sendfile(filePath);
```

View Engines

- **View engine** (template engine) is a framework/library that generates views
- Given a template/view JavaScript generates a valid HTML code
- There are lots of JavaScript view engines, and they can be separated into client and server
  - Client: KendoUI, Handlebars.js, jQuery, AngularJS, etc.
  - Server: Jade, Handlebars, HAML, EJS, Vash, etc.
- **Handlebars.js** is recommended. It is a library for creating client-side or server-side templates

Handlebars - Basic (Server side)

- In app.js

```
var express = require('express'),
    exphbs  = require('express3-handlebars'),
    app = express();

app.engine('hbs', exphbs({defaultLayout:'main', layoutsDir: 'views/',
extname: '.hbs'}));
app.set('view engine', 'hbs');
app.get('/', function (req, res) {
    res.render('home', {layout: 'main'});
});
 app.listen(80);
```

Handlebars - Basic (Server side)

- In views/main.hbs

<html> {{{body}}} </html>

{{{body}}} - placeholder for the main content to be rendered

Handlebars - Helpers

- Help you to do something more in the template
- {{#if}}
  - If the condition is true, display the thing in the middle.
  - {{#if condition}} Display {{dataname}} {{/if}
- {{#each}}
  - {{#each items}} {{name}} {{emotion}} {{/each}}
  - Iterate the the rows in “items”

```
var data = {
  items: [
    {name: "Handlebars", emotion: "love"},
    {name: "Mustache", emotion: "enjoy"},
    {name: "Ember", emotion: "want to learn"}
  ]
};
```

View Rendering

- View filenames take the form "<name>.<engine>", where <engine> is the name of the module that will be required.

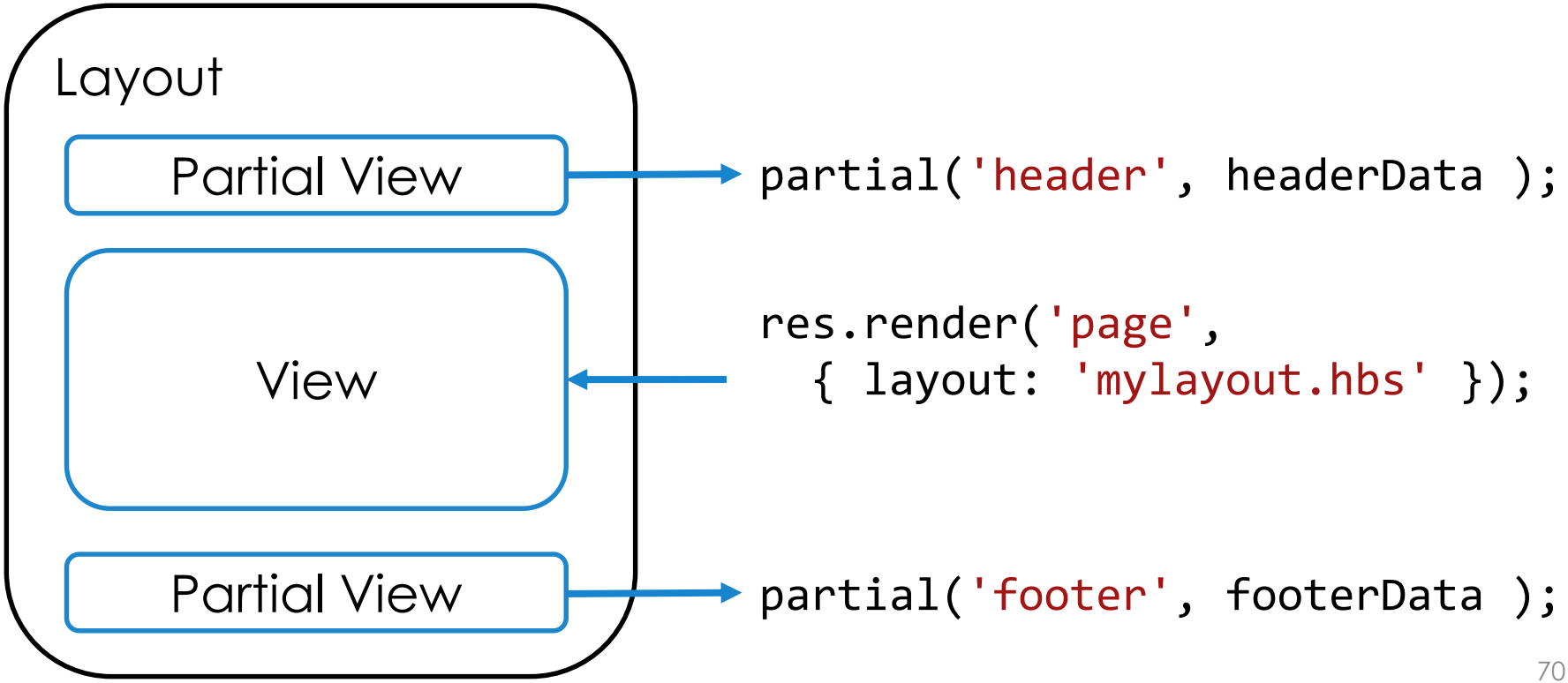
```
app.get('/', function (req, res) {
    res.render( 'index.hbs', { title: 'My Site' } );
});
```

Model

View file

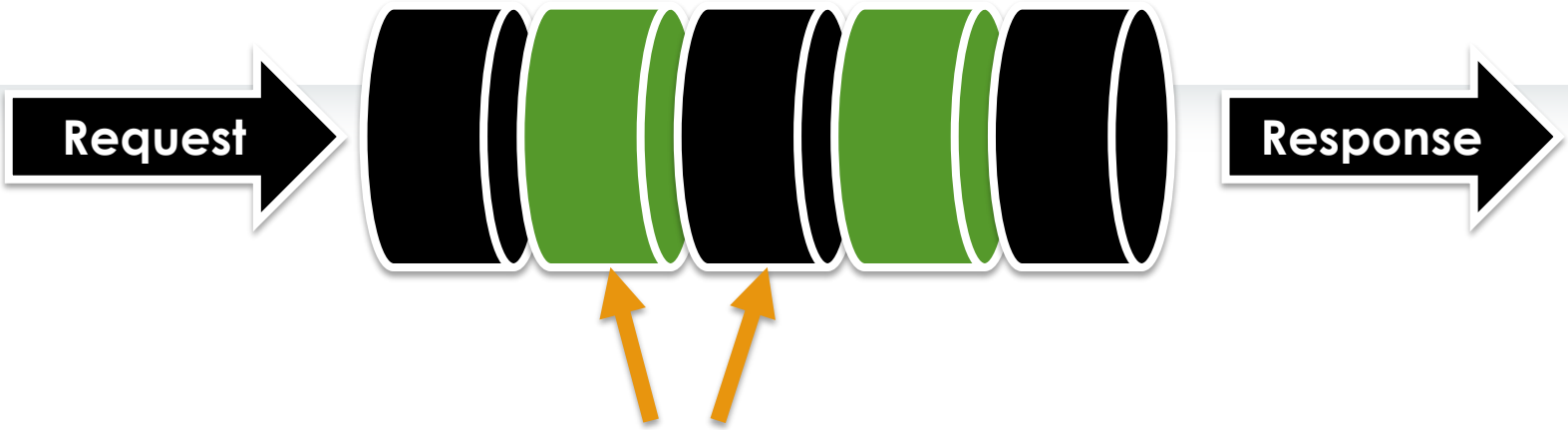
View Layout & Partials

- The Express view system has built-in support for partials which are "mini" views representing a document fragment.



Middleware

= Request Processing Pipeline



Middleware (logging, authentication, etc.)

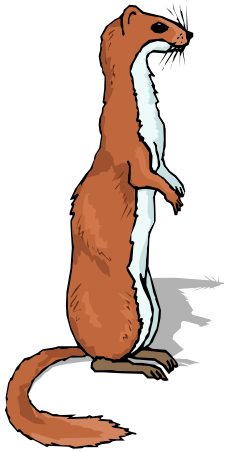
What is middleware?

- Middleware is a request handler.
  - Routing , Controller, Models, Views… Security


```
function myFunMiddleware(request, response, next) {
    // Do stuff with the request and response.
    // When we're all done, call next() to defer
    // to the next middleware.
    next();
}
```

```
// Add some middleware
app.use(connect.logger());
app.use(connect.Security);
app.use(connect.Routing);
app.use(myFunMiddleware);
...
```

Data Management using Mongoose



Document Data Model

Relational

PERSON

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rome

NO RELATION

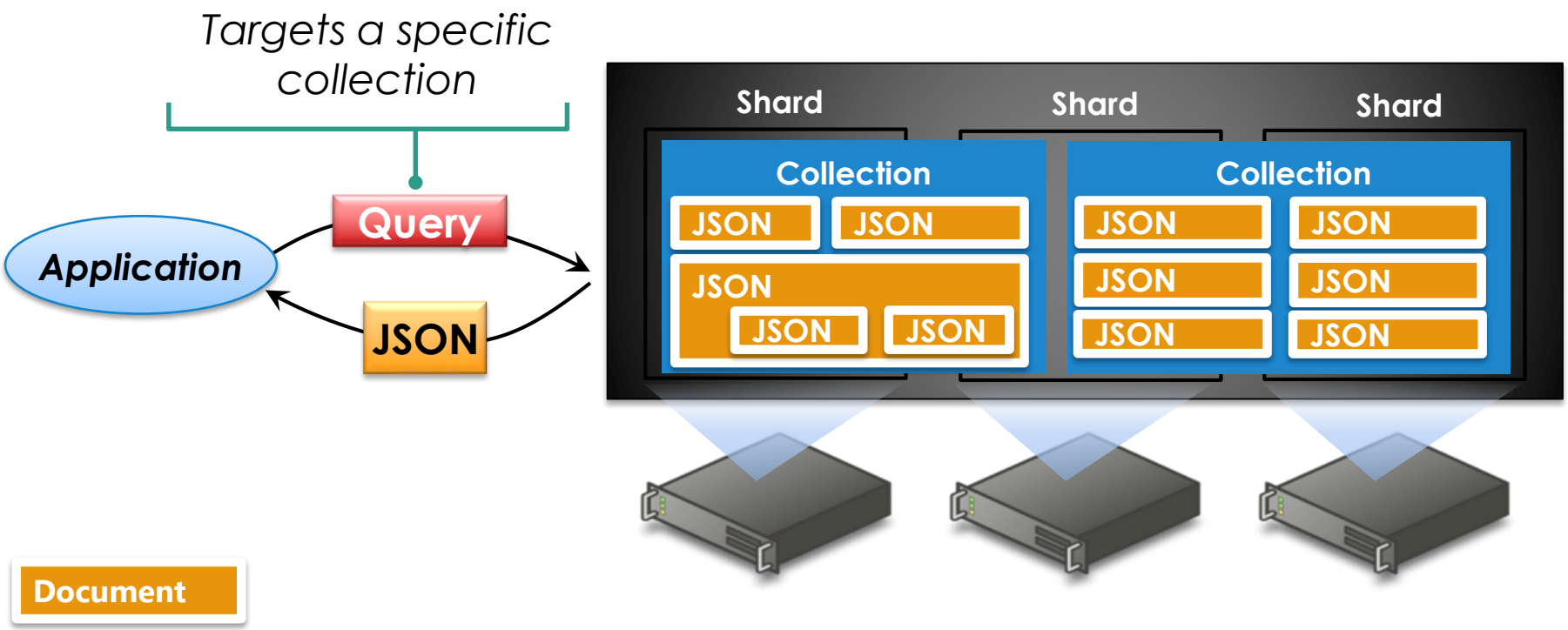
CAR

Car_ID	Model	Year	Value	Pers_ID
101	Bently	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

MongoDB

```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location:
[45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  }
}
```

MongoDB Document Databases

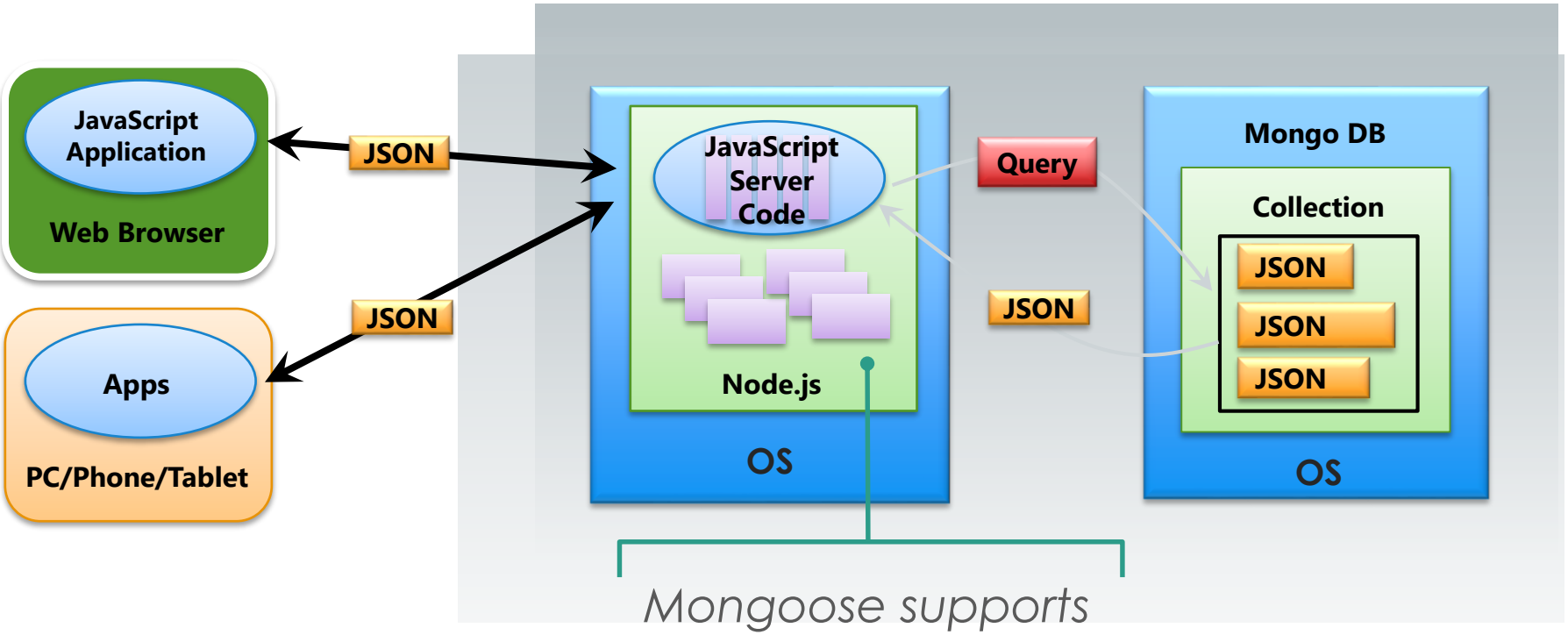


- IDE
http://app.robomongo.org/

- MongoDB in the Cloud
https://mongolab.com/ (500MB free)

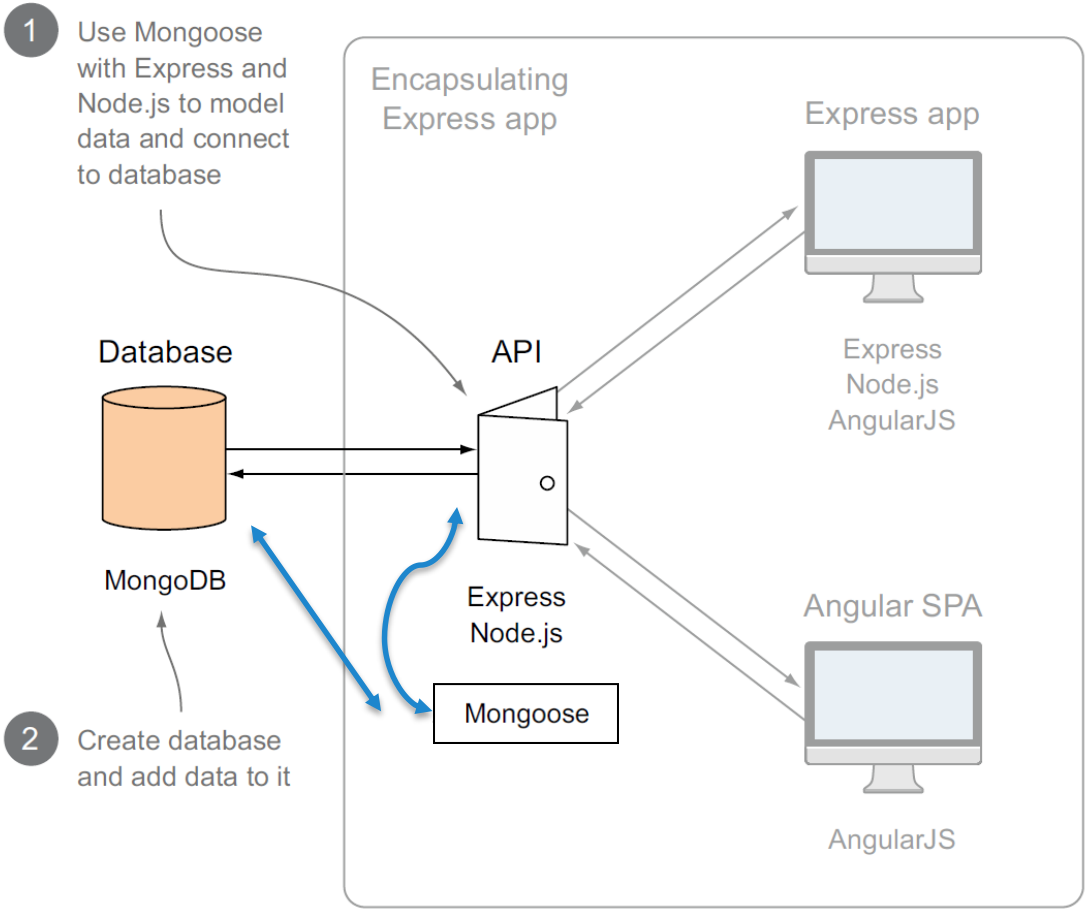
JSON Storage for JavaScript Applications

The complete picture



Mongoose supports connecting then reading/writing to MongoDB

Mongoose inside Express is used to model the data and manage the connection to the database



Node/Express application interacts with MongoDB through Mongoose

Mongoose Overview

- Mongoose is a object-document model module in Node.js for MongoDB
  - Wraps the functionality of the native MongoDB driver
  - Exposes models to read/write documents:
    - Each schema maps to a MongoDB collection
    - Models can be instantiated based on a schema
    - Instances of models represent documents in MongoDB
  - Supports validation on save
  - Extends the native queries

Mongoose (getting started)

- Require Node module

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

- Connect to MongoDB

```
var db = mongoose.connect('mongodb://localhost/opendata');
```

- Define your schema in JSON

```
var ServiceRequestSchema = new Schema({
    service_request_id: {type: Number},
    status: {type: String},
    status_notes: {type: String},
    service_name: {type: String},
    service_code: {type: String},
    description: {type: String},
    agency_responsible: {type: String},
    service_notice: {type: String},
    requested_datetime: {type: Date, default: Date.now},
    updated_datetime: {type: Date, default: Date.now},
    expected_datetime: {type: Date, default: Date.now},
    address: {type: String},
    address_id: {type: Number},
    zipcode: {type: String},
    long: {type: Number},
    lat: {type: Number},
    media_url: {type: String}
});
```

```
> db.service_requests.findOne()
{
        "_id" : ObjectId("53208a81961bffb27dd12d77"),
        "service_request_id" : NumberLong("101002585789"),
        "status" : "open",
        "status_notes" : "In progress - The request has been scheduled.",
        "service_name" : "Road - Pot hole",
        "service_code" : "CSROWR-12",
        "description" : null,
        "agency_responsible" : "311 Toronto",
        "service_notice" : null,
        "requested_datetime" : ISODate("2014-03-09T23:48:14Z"),
        "updated_datetime" : null,
        "expected_datetime" : ISODate("2014-03-14T23:49:00Z"),
        "address" : "HEDDINGTON AVE & ROSELAWN AVE, , former TORONTO",
        "address_id" : 13456980,
        "zipcode" : null,
        "long" : -79.415014617,
        "lat" : 43.707604881,
        "media_url" : null
}
```

- Create the model object

```
var ServiceRequest = db.model('service_requests', ServiceRequestSchema);
```

Mongoose - methods

- Mongoose supports all the CRUD operations:
  - Create –> modelObj.save(callback)
  - Read –> Model.find().exec(callback)
  - Update –> modelObj.update(props, callback)
    –> Model.update(condition, props, cb)
  - Remove –> modelObj.remove(callback)
    –> Model.remove(condition, props, cb)

Installing Mongoose

- Run the following from the CMD

```
$ npm install mongoose --save
```

- **In node**
  - **Load the module**

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

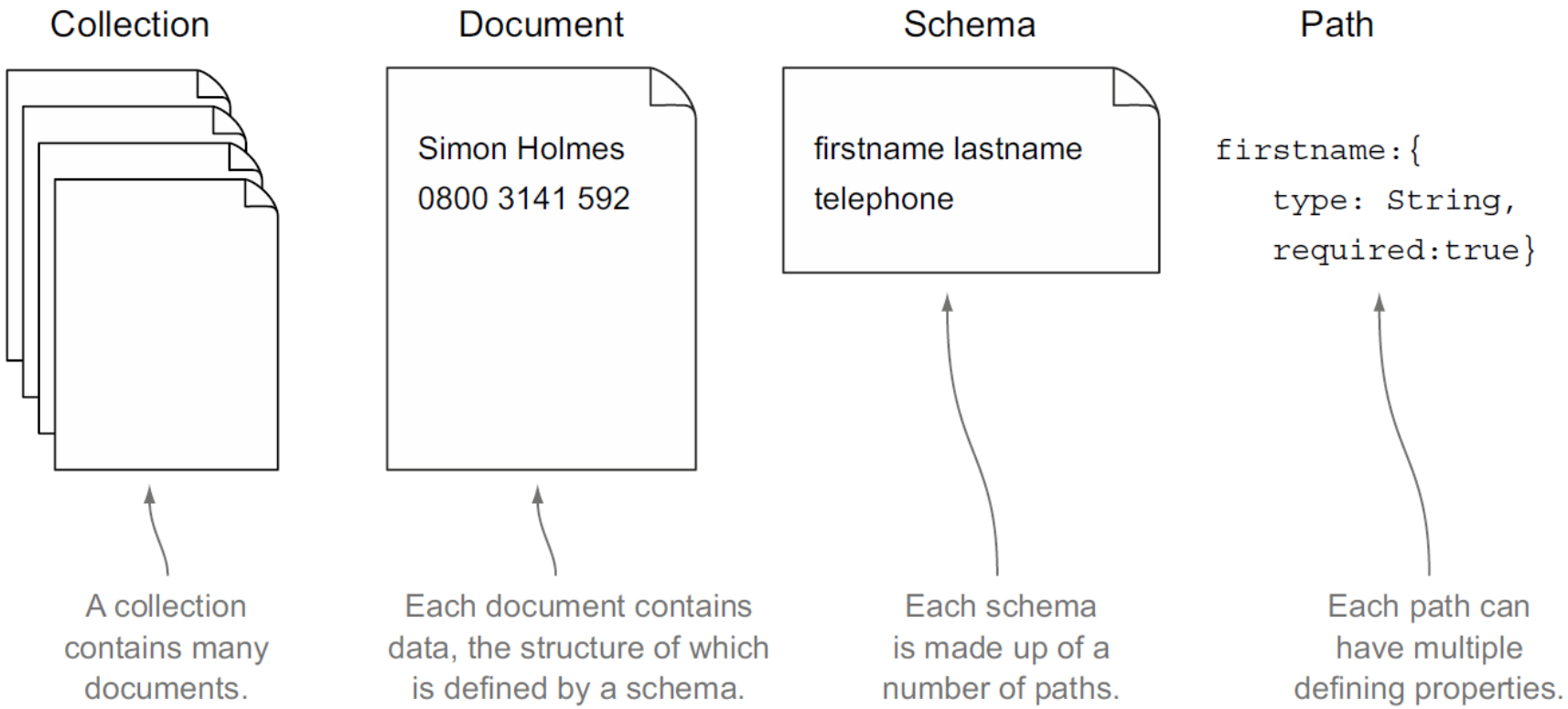
  - **Connect to the database**

```
mongoose.connect(mongoDbPath);
```

  - **Create models and persist data**

```
var Unit = mongoose.model('Unit', { type: String} );
new Unit({type: 'warrior'}).save(callback); //create
Unit.find({type: 'warrior'}).exec(callback); //fetch
```

Relationships among collections, documents, schemas, and paths in MongoDB and Mongoose



Document Instance vs. Schema

```
{
  "firstname" : "Simon",
  "surname" : "Holmes",
  _id : ObjectId("52279effc62ca8b0c1000007")
}
```

Example MongoDB document

```
{
  firstname : String,
  surname : String
}
```

Corresponding Mongoose schema

Mongoose Models

- Mongoose supports models and a model has a schema
- Each of the properties must have a type
  - Types can be Number, String, Boolean, array, object

```
var modelSchema = new mongoose.Schema({
  propString: String,
  propNumber: Number,
  propObject: {},
  propArray: [],
  propBool: Boolean
});

var Model = mongoose.model('Model', modelSchema);
```

Property Validation

- With Mongoose developers can define custom validation on their properties
  - i.e. validate records when trying to save

```
var unitSchema = new mongoose.Schema({…});
unitSchema.path('position.x').validate(function(value){
  return value>=0 && value <= maxX;
});
unitSchema.path('position.y').validate(function(value){
  return value>=0 && value <= maxY;
});
```

Mongoose Queries

- Mongoose supports many queries:
  - For equality/non-equality
  - Selection of some properties
  - Sorting
  - Limit & skip
- All queries are executed over the object returned by Model.find*()
  - Call .exec() at the end to run the query

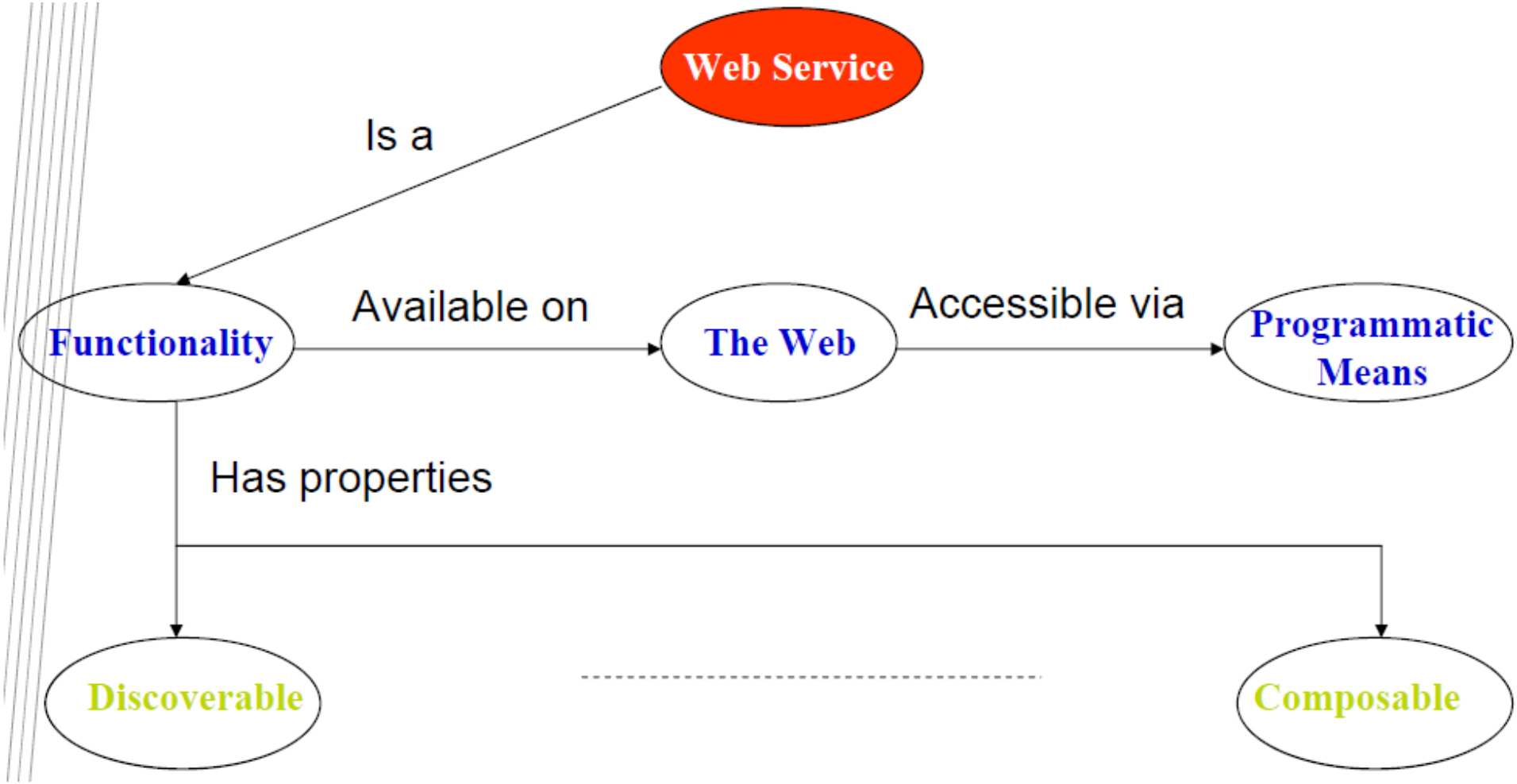
```
.where({conditionOne: true}).or({conditionTwo: true})
```

REST Services

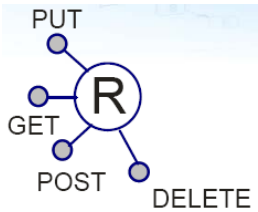
What is a Web Service?

- Web services = component with standard interface to ease integrating applications/ components
- Software component provided through a **network-accessible endpoint**
  - Someone else may own the service and is responsible for its operation
- Services exchange standard ~~XML /~~ JSON messages
- Major design goal = **interoperability between heterogeneous systems**
- Two major types: **REST** Services ~~and **SOAP** Services~~

What is Web Service?



REST Principles



- **Addressable Resources** (nouns): Identified by a URI

(e.g., http://example.com/customers/123)

- **Uniform Interface** (verbs): GET, POST, PUT, and DELETE

  -Use verbs to exchange application state and representation

  -Embracing HTTP as an Application Protocol

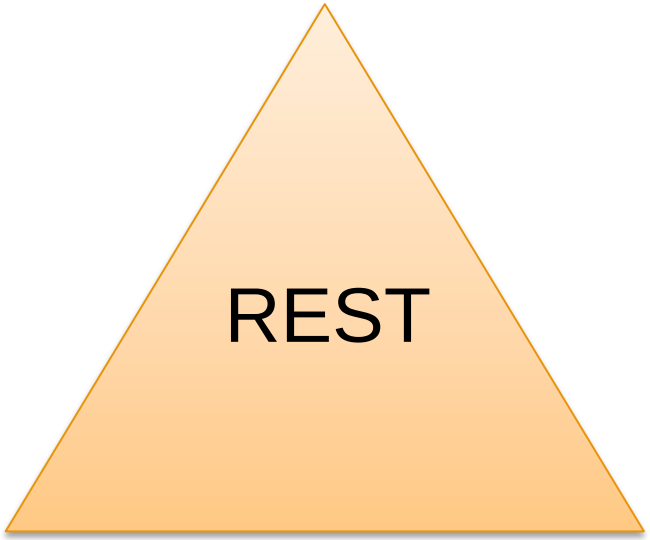
- **Representation-oriented**

  -**Representation of the resource state** transferred between client and server in a variety of data formats: XML, JSON, (X)HTML, RSS..

- **Hyperlinks** define relationships between resources and valid state transitions of the service interaction

REST Services Main Concepts

Nouns (Resources)
e.g., http://example.com/employees/12345



Verbs
e.g., GET, POST

Representations
e.g., XML, JSON

Resources

- The key abstraction in REST is a **resource**
- A resource is a conceptual mapping to a set of entities
  - Any **information that can be named can be a resource**: a document or image, a temporal service (e.g. "today's weather in Doha"), a collection of books and their authors, and so on
- Represented with a global identifier (URI in HTTP)
  - http://www.boeing.com/aircraft/747

Naming Resources

- REST uses URI to identify resources
  - http://localhost/books/
  - http://localhost/books/ISBN-0011
  - http://localhost/books/ISBN-0011/authors
  - http://localhost/classes
  - http://localhost/classes/cmps356
  - http://localhost/classes/cs356/students
- As you traverse the path from more generic to more specific, you are navigating the data

Representations

- Specify the data format used when returning a resource representation to the client

- Two main formats:
  - JavaScript Object Notation (JSON)
  - XML

- It is common to have multiple representations of the same data

Representations

- XML

```
<course>
   <id>cmps356</id>
   <name>Enterprise Application
   Development</name>
</course>
```

- JSON

```
{
  id: ‘cmps356’,
   name: ‘Enterprise Application Development’
}
```

HTTP Verbs

- Represent the actions to be performed on resources
- Retrieve a representation of a resource: **GET**
- Create a new resource:
  - Use **POST** when the server decides the new resource URI
  - Use **PUT** when the client decides the new resource URI. Also **PUT** is also typically used for update
- Delete an existing resource: **DELETE**
- Get metadata about an existing resource: **HEAD**
- See which of the verbs the resource understands: **OPTIONS**

REST Services using Node.js

- See posted Node.js REST Services example
- Test them using Postman Chrome plugin

https://www.getpostman.com/