INF5750

More JavaScript

Outline

- I. More JS concepts
- II. New ES2015 features
- III. Node.js web service

I. More JS concepts

- Scope and closure
- *this* binding
- Classes and prototypes
- Event loop and async programming

Scope and closure

- Scope: set of rules storing and retrieving variables
- Scope is defined by functions*
- JS scope is lexical - defined by source code
- Scopes can be nested - global it the outermost scope

Nested scope

```
var a = "global scope";
scope1();

function scope1() {
    var b = "scope of scope1()";

    function scope2() {
        var c = "scope of scope2()";

    }

    console.log(c); //ReferenceError
}
```

Closure

- A function can have *closure* over a scope
- Variables of that scope will be available even if the function is executed outside this scope
- Modules generally rely on closure to ensure private variables can be used outside the module itslelf

https://jsbin.com/vecayu/edit?js,console

https://jsbin.com/hejabup/41/edit?js,console

this binding

- *this* is a special keyword defined in the scope of every function

- Runtime binding that depends on how the function was called - the *call-site*

- Does *not* refer to:

  - The function itself

  - The function's lexical scope

- Implicit mechanism for passing object references

this binding rules

- Default binding: call with plain global object
- Implicit binding: call from "owning" object
- Explicit binding: call using .call() .apply() or use .bind()
- *new* binding: call using new returns new object which is *this*

https://jsbin.com/xijaze/90/edit?html,js,output

- Order: *new* - explicit - implicit - default

Prototypes and classes

- JS is object oriented, but does not have (real) classes
- Class-based languages are based on instantiation of classes, i.e. making new *copies* of a class
- JS is based on making new objects with *links* to prototypes
- Class-based object oriented design patterns can largely be replicated in JS

Prototypes

- All objects have a hidden [[Prototype]] property linked to an object that becomes the prototype

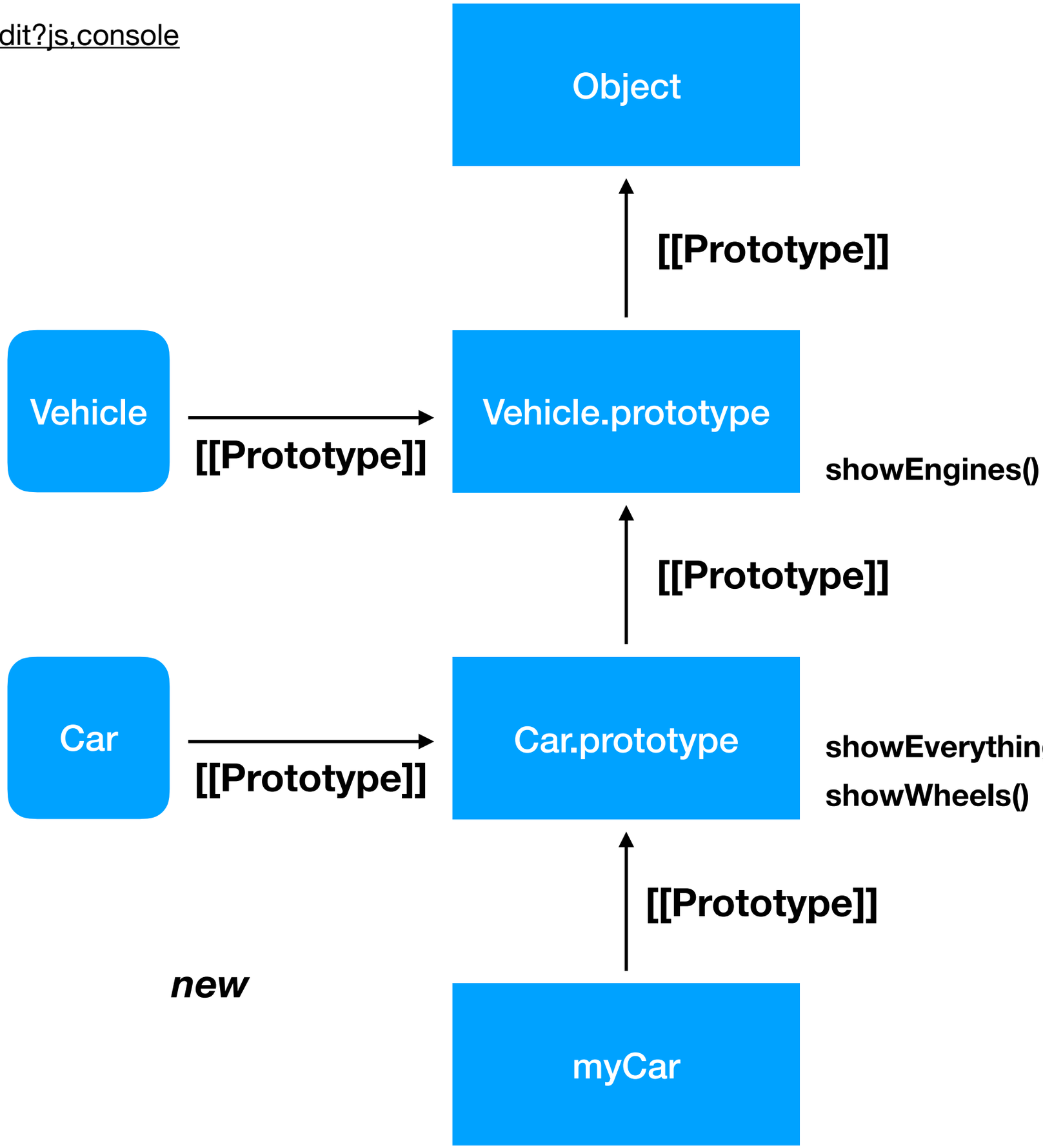
- Can be a chain of linked objects, ending with an Object prototype

- The *prototype chain* is traversed when looking for properties on an object

OO vs OLOO

- Object oriented (OO) vs objects linked to other objects (OOLO)
- Disagreement on what is appropriate in general and in JS
- OO based on base/super class that can be extended
- OOLO based on delegating common functionality to a shared object using prototype link

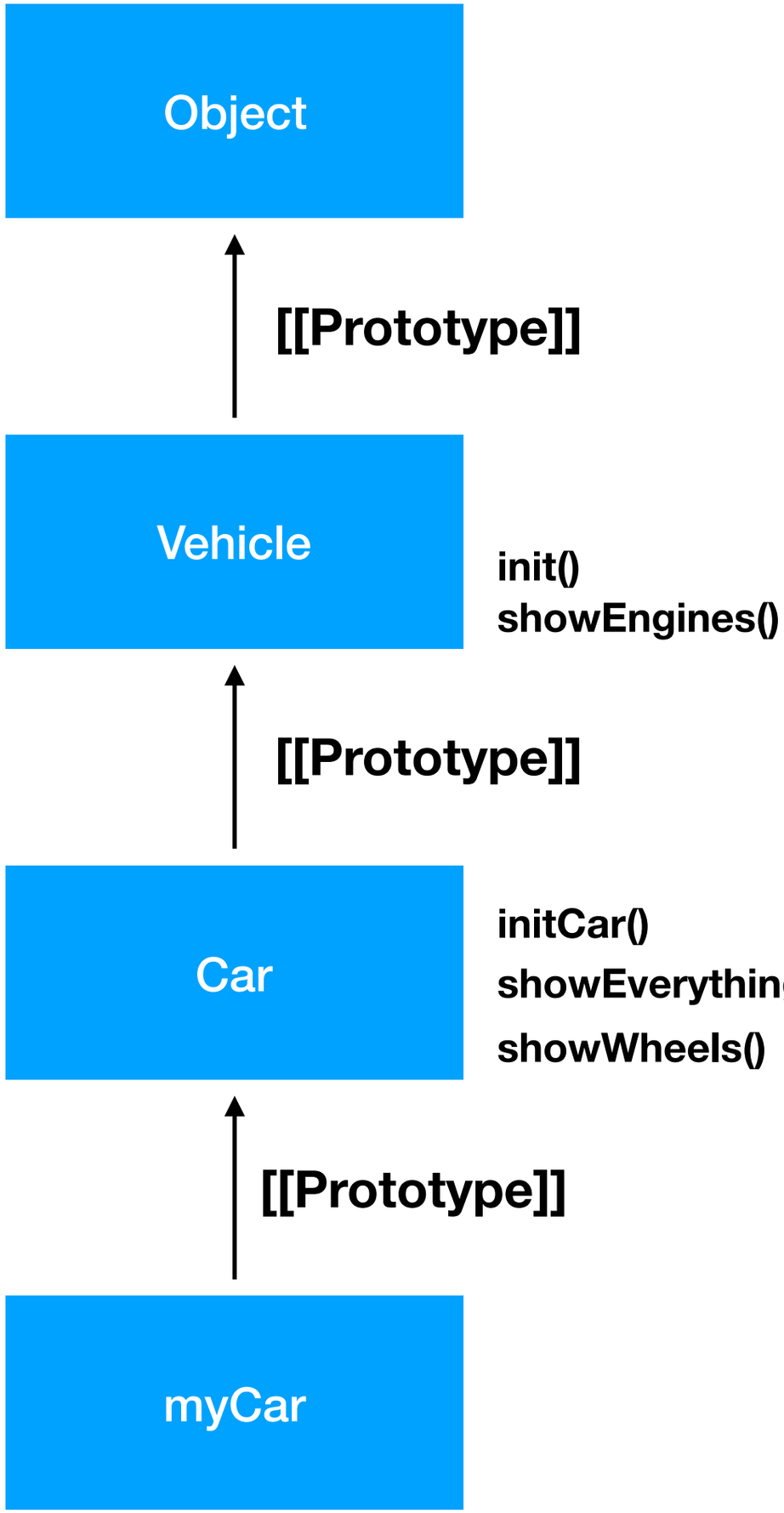
https://jsbin.com/wofuqa/edit?js,console



function

object

https://jsbin.com/sumopos/edit?js,console



function

object

Event loop and async programming

- JS application are single threaded:
    - can only do one thing at the time
    - each "chunk" of code runs from start to finish
- Pieces of code run based on events - user interaction, IO, callbacks
- Runtime environments has an *event queue* with callbacks to the code
- An *event loop* mechanism takes the first event of the queue and runs it to completion, fetches the next event etc

JS Runtime

```
function doNow() {

setTimeout(doLater,
    500);

}


function doLater() {
console.log("later");
}

doNow();
```

Call stack

console.log()

doLater()

API (browser, node.js…)

setTimeout()

Event loop tick!

Event queue

doLater()

Event loop (better) illustrated

https://www.youtube.com/watch?v=8aGhZQkoFbQ&t=22m20s

- Watch the full video later!

Async programming

- Requirement that web pages or web services are not blocked e.g. while waiting for data to be returned over the network

- Rely on callbacks and promises to work asynchronously

- Callbacks from runtime to our applications are added to callback/event queue

- Challenges with callbacks: callback hell and trust issues

https://jsbin.com/niraha/edit?js,console

II. New ES2015 features

- Transpiling and shimming
- New syntax
- Collections
- Classes
- Async programming - promises and generators

- Overview of features: http://es6-features.org

Polyfills and transpilers

- Not all new language features supported in all browsers
- Two solutions/workarounds:
  - New APIs, features on objects etc: polyfills/shims -
  - New syntax: transpiling code into old syntax

    https://babeljs.io/repl/

https://kangax.github.io/compat-table/es6/

Block-scoped variables

- *var* is attached to enclosing function (or global) scope
- *let* and *const* are attached to block-scope: { }
- *const* = constant - cannot be reassigned

https://jsbin.com/sazabul/edit?js,console

Spread and Rest

- … operator - called *spread* or *rest* depending on use
- in front of array (or other *iterable*): spreads individual values
- in front of function parameters: gathers the *rest* of the variables in an array

https://jsbin.com/duwale/edit?js,console

Default parameter values

- Default function parameter values
- Defaults can be explicit value or expression

https://jsbin.com/cihusel/edit?js,console

Destructuring

- Destructured assignment
- Applies to arrays and objects
- Facilitates assignment of values from indexes (arrays) or properties (objects)
- Supports default values, nested properties…

https://jsbin.com/rugacux/edit?js,console

Arrow functions

- Short function syntax: (params) => {expression}
- Single- or multi-line
- Changes *this*-binding - not just a short form

https://jsbin.com/tacili/edit?js,console

Arrow functions

- When is use of arrow functions appropriate?
    - for short, single-state inline function expressions not using *this* binding
    - replacing function currently using workarounds for lexical this binding (self = this, bind(this) etc…)
    - inner functions relying on copies of parameters to bring them into lexical scope
- For everything else use normal function declarations…

Template literals

- String literals using ` as delimiter
- Supports:
  - embedding expressions in strings
  - multiline strings
  - tagged template literals

https://jsbin.com/xavorim/edit?js,console

Collections

- Maps - key-value maps where any object can act as key

https://jsbin.com/gidoqiy/edit?js,console

- Sets - collections of unique values

https://jsbin.com/vijula/edit?js,console

- WeakMap and WeakSet - keys (map) and values (set) are hold weakly - garbage collected when last reference removed

Classes

- ES2016 introduced new class-related keywords: *class, extends, super, static*
- No new functionality, syntactic sugar on existing prototype-based functionality

https://jsbin.com/tomarah/edit?js,console

Promises

- Promises are used for program flow in async code
- Provides a "promise" of a future value - something will be returned in the future
- Not a replacement for callbacks, but an intermediary
- Promises are either resolved or rejected

Promises in ES2015

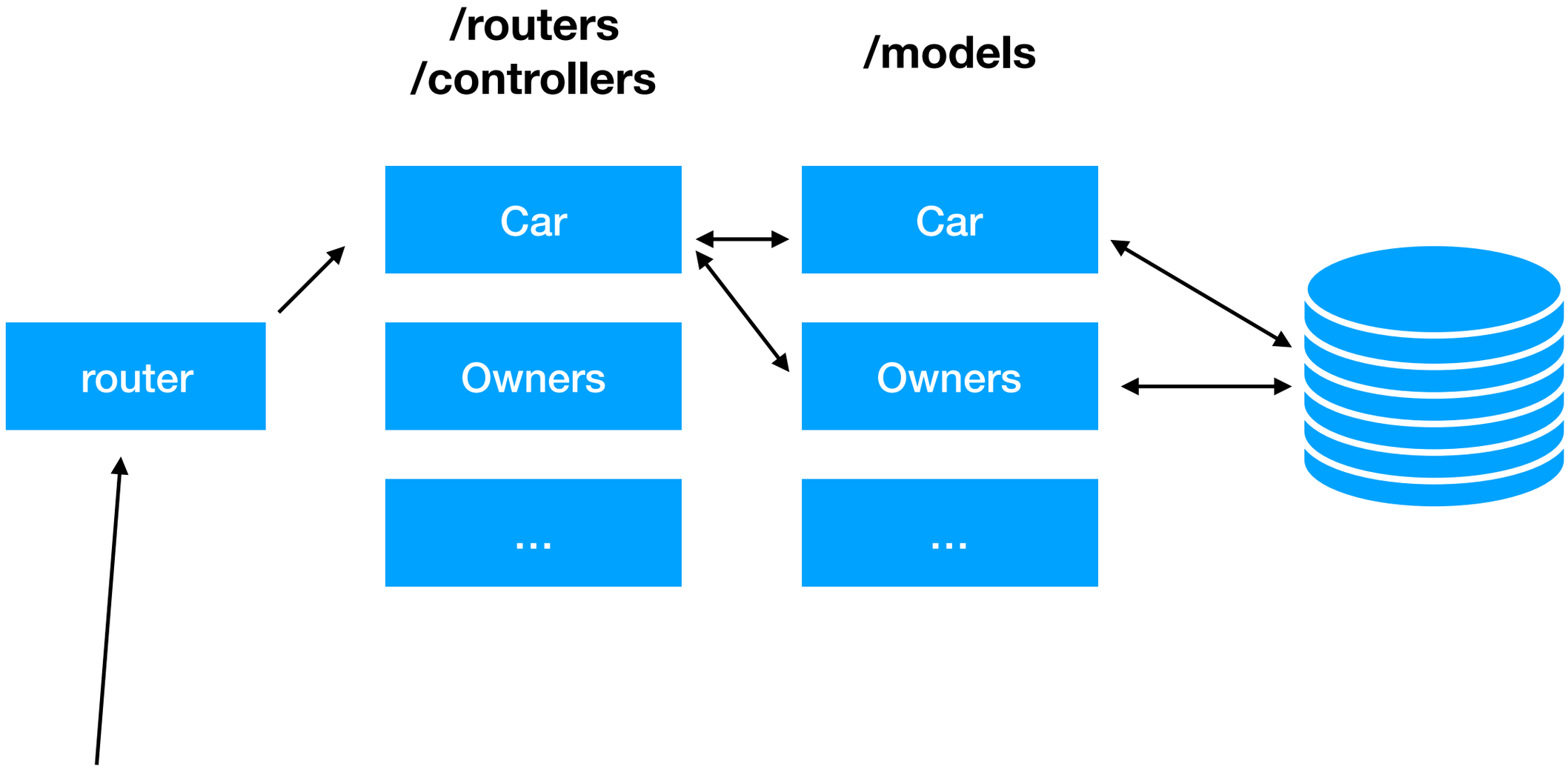
- Constructed as new Promise(*function*)
- The function passed in the constructor call has two functions as parameters used to resolve or reject the promise
- Promises have a .then() function which takes 1-2 functions for handling success and failure of promise

https://jsbin.com/lobekaj/edit?js,console

III. Node.js web service

- Structure of node.js web service

Example structure



GET ../api/cars/1?includeOwner=true

Group teachers

- Group teachers to share git repository with:
    - Group 1: Mustafa - *mustafma* on github.uio.no
    - Group 2: Mustafa - *mustafma* on github.uio.no
    - Group 3: Nikolai - *njsverdr* on github.uio.no
    - Group 4: August - *augusthh* on github.uio.no