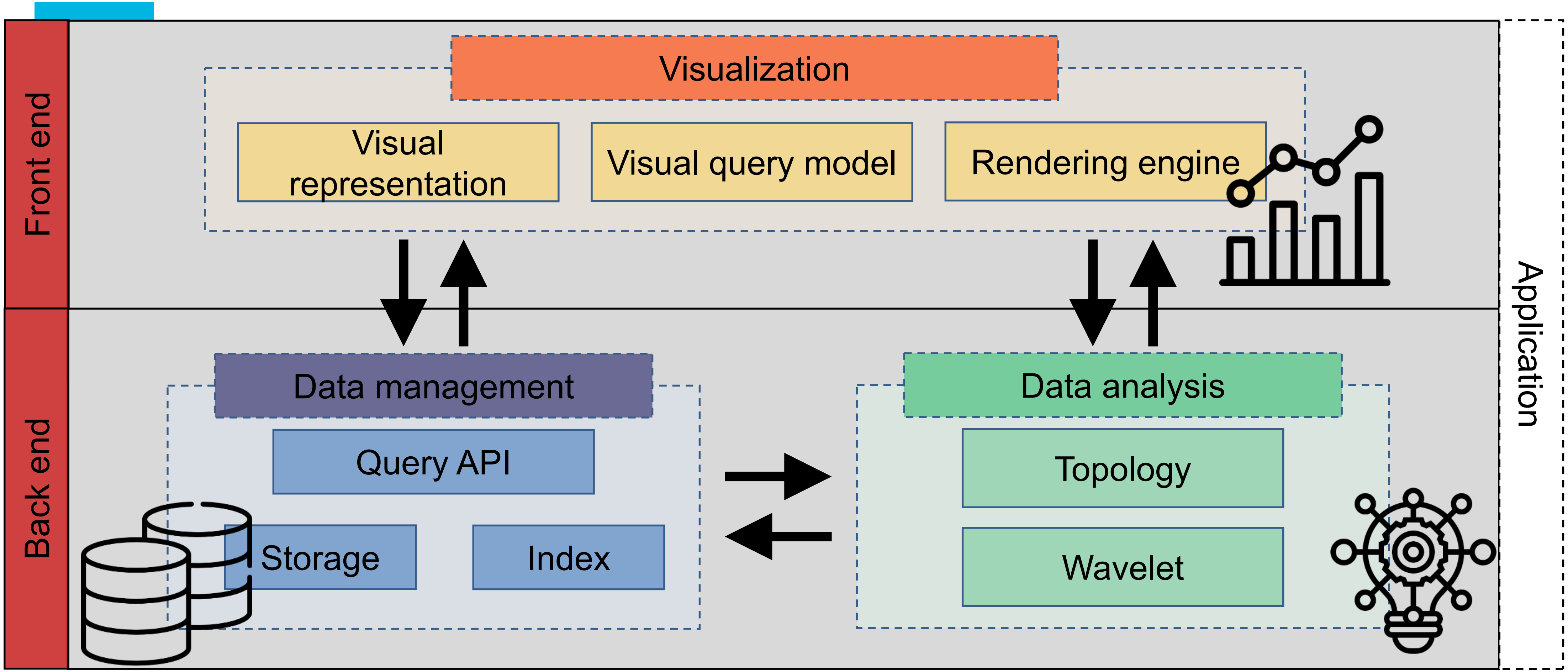
Building blocks

CS524: Big Data Visualization & Analytics

Fabio Miranda

https://fmiranda.me

Big data visualization system



Big data visualization system

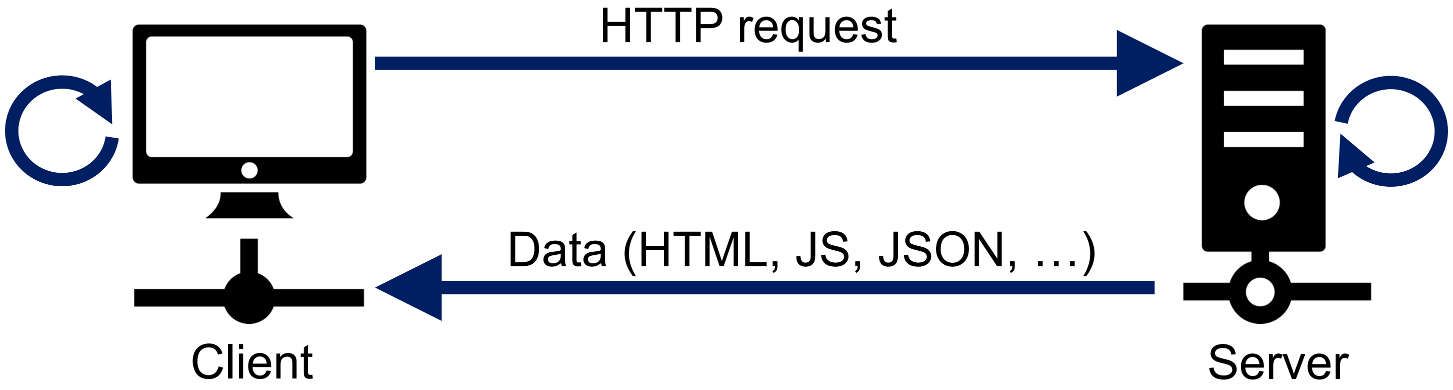
- Why separate front-end and back-end development?
  - Separation of concerns between presentation layer (front end) and data layer (back end).
  - Easily mapped to a client-server model.
    - Client: front end
    - Server: back end
  - Easy deployment.

Building blocks:

- Front-end:
  - Web technologies
  - Web environment
  - JavaScript
  - D3
  - TypeScript
  - Angular
- Front-end and back-end communication:
  - Flask (Python)
  - Mongoose (C / C++)
- Back-end building blocks:
  - Boost
  - Qt
  - CUDA

Client and server

Client-side:
- Rendering
- Interaction
- Light-weight aggregation
- Filtering



HTTP request

Data (HTML, JS, JSON, …)

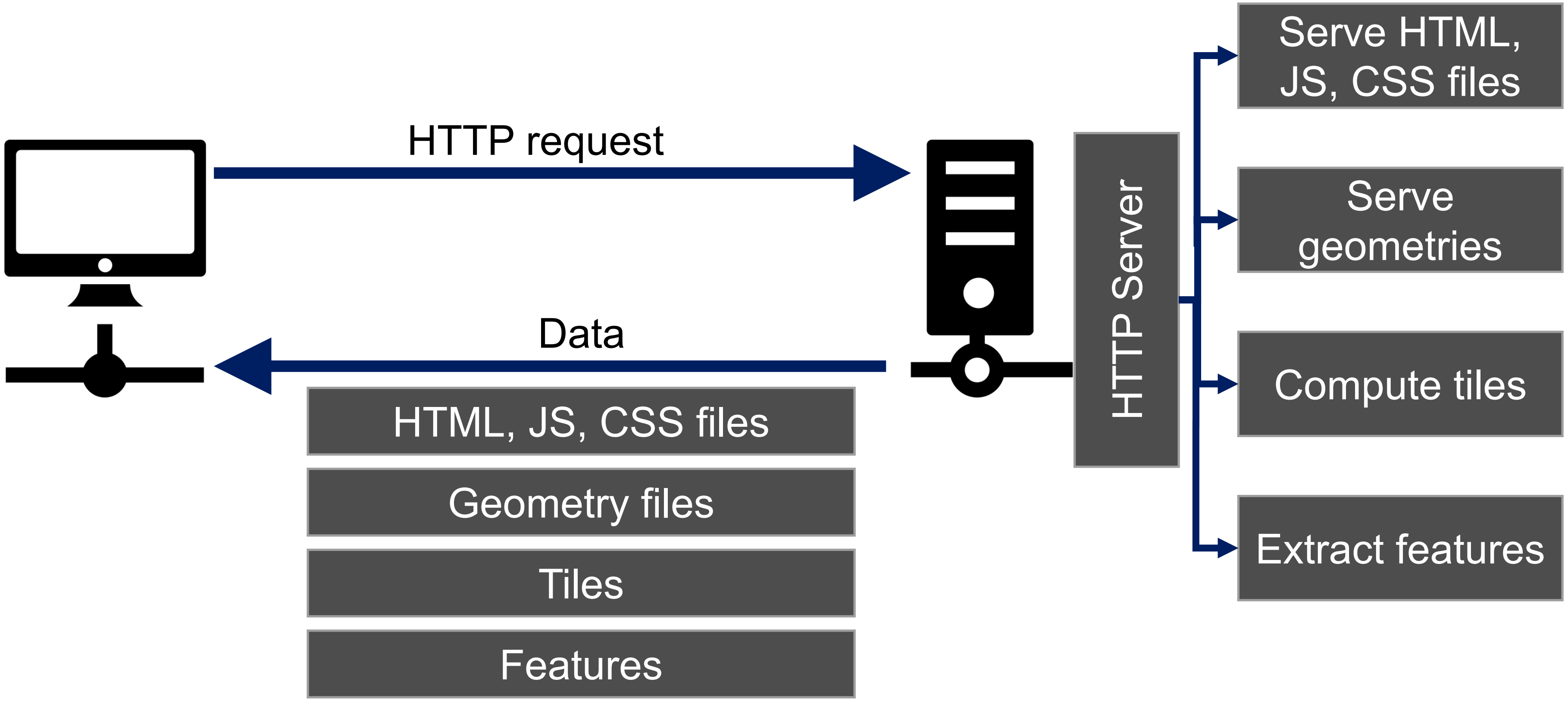
Client

Server

Server-side:
- Database query
- Feature extraction
- Data mining

Client and server

Application



HTTP request

Data

HTML, JS, CSS files

Geometry files

Tiles

Features

HTTP Server

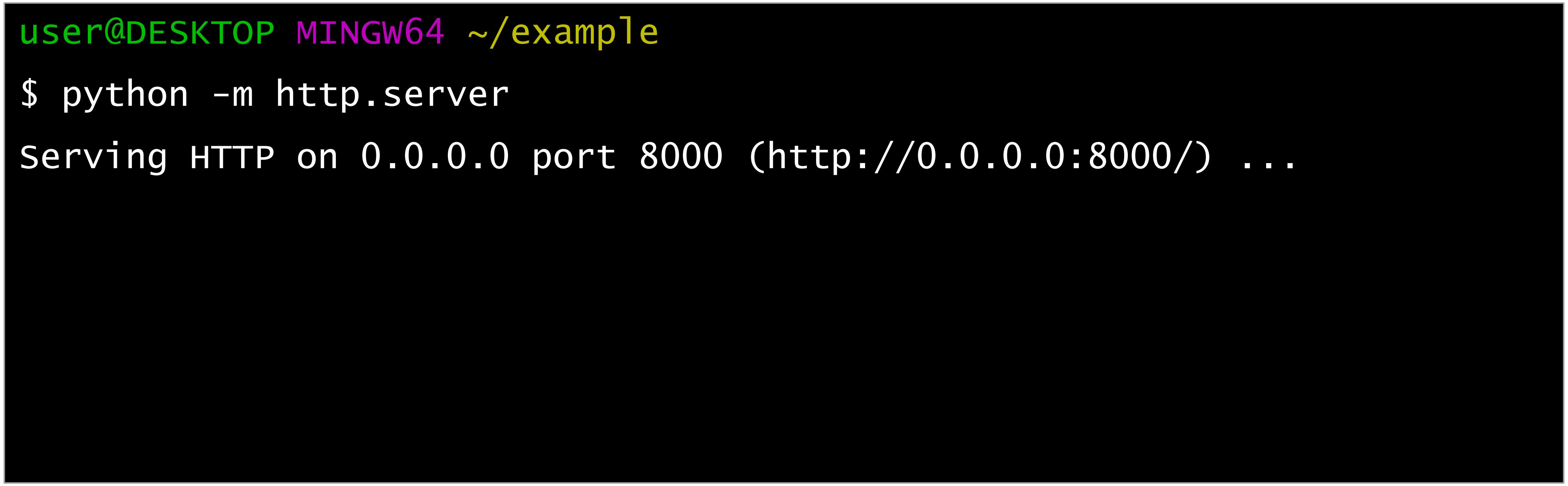
Serve HTML, JS, CSS files

Serve geometries

Compute tiles

Extract features

Simple server



```
user@DESKTOP MINGW64 ~/example
$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Detailed steps: https://mzl.la/3bSLff0

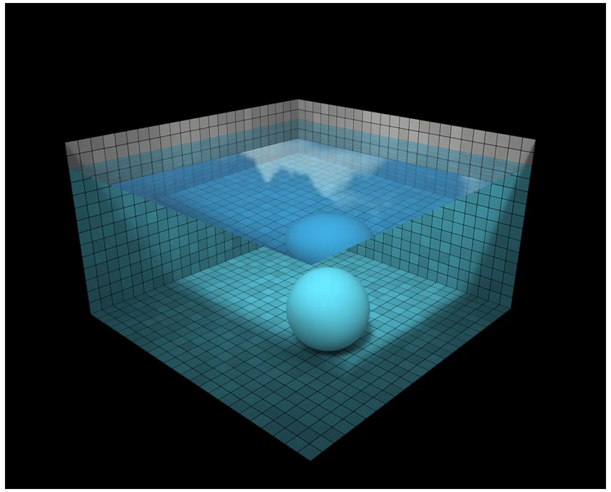
JavaScript and D3

JavaScript:
a client-side programming language

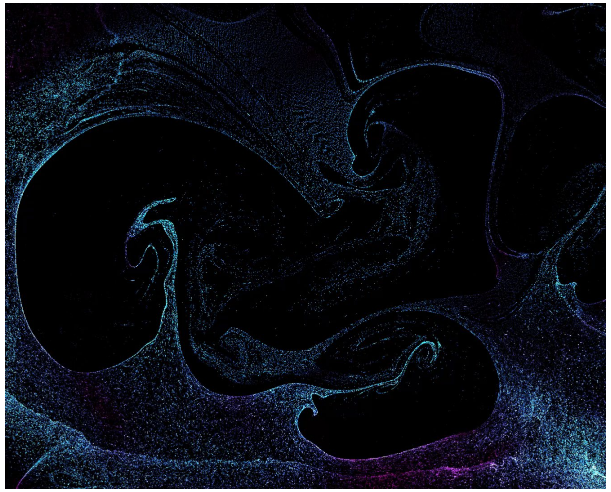
- Interpreted object-oriented language.
- Loosely typed language.
  - Does not require a variable type to be specified.
- Add, delete, and modify nodes from the document tree.
- Integration with other frameworks and toolkits:
  - Qt
  - Swift

JavaScript: a client-side programming language

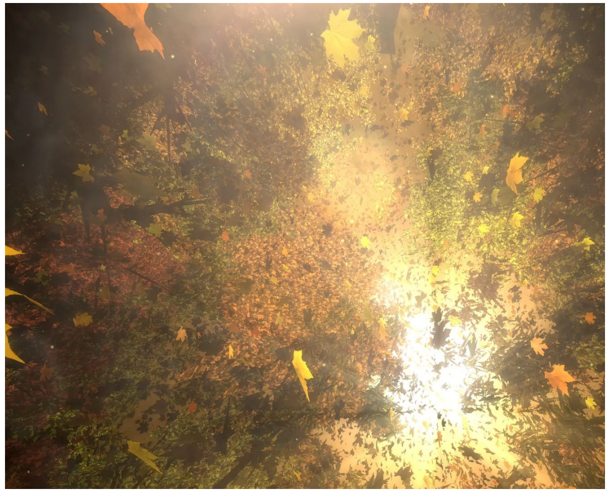
- Is JavaScript slow?
  - JavaScript engines in browsers are getting much faster.
  - Not an issue for graphics, since we transfer the data to the GPU with WebGL.



http://madebyevan.com/webgl-water/



https://haxiomic.github.io/projects/webgl-fluid-and-particles/



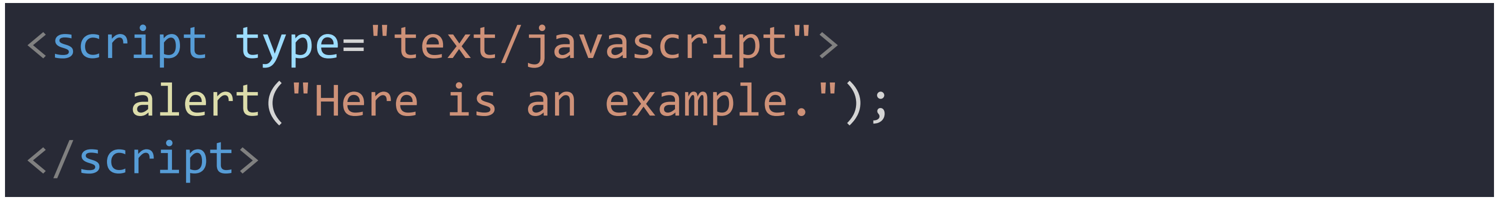
http://oos.moxiecode.com/js_webgl/autumn/

JavaScript basics

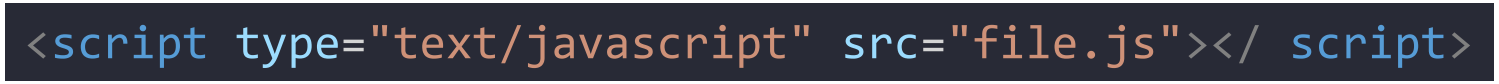
- Two scopes:
  - Local
  - Global
- Variable created inside a function with ‘var’ keyword: local to function.
  - Created and destroyed every time function is called.
  - BUT: variables declared without ‘var’ keyword are always global.
- Variable created outside a function: global

JavaScript basics

- Inserting JavaScript code in a web page:
  - Inside an HTML tag script.
  - In an external file.
  - As an HTML attribute value.



```
<script type="text/javascript">
    alert("Here is an example.");
</script>
```

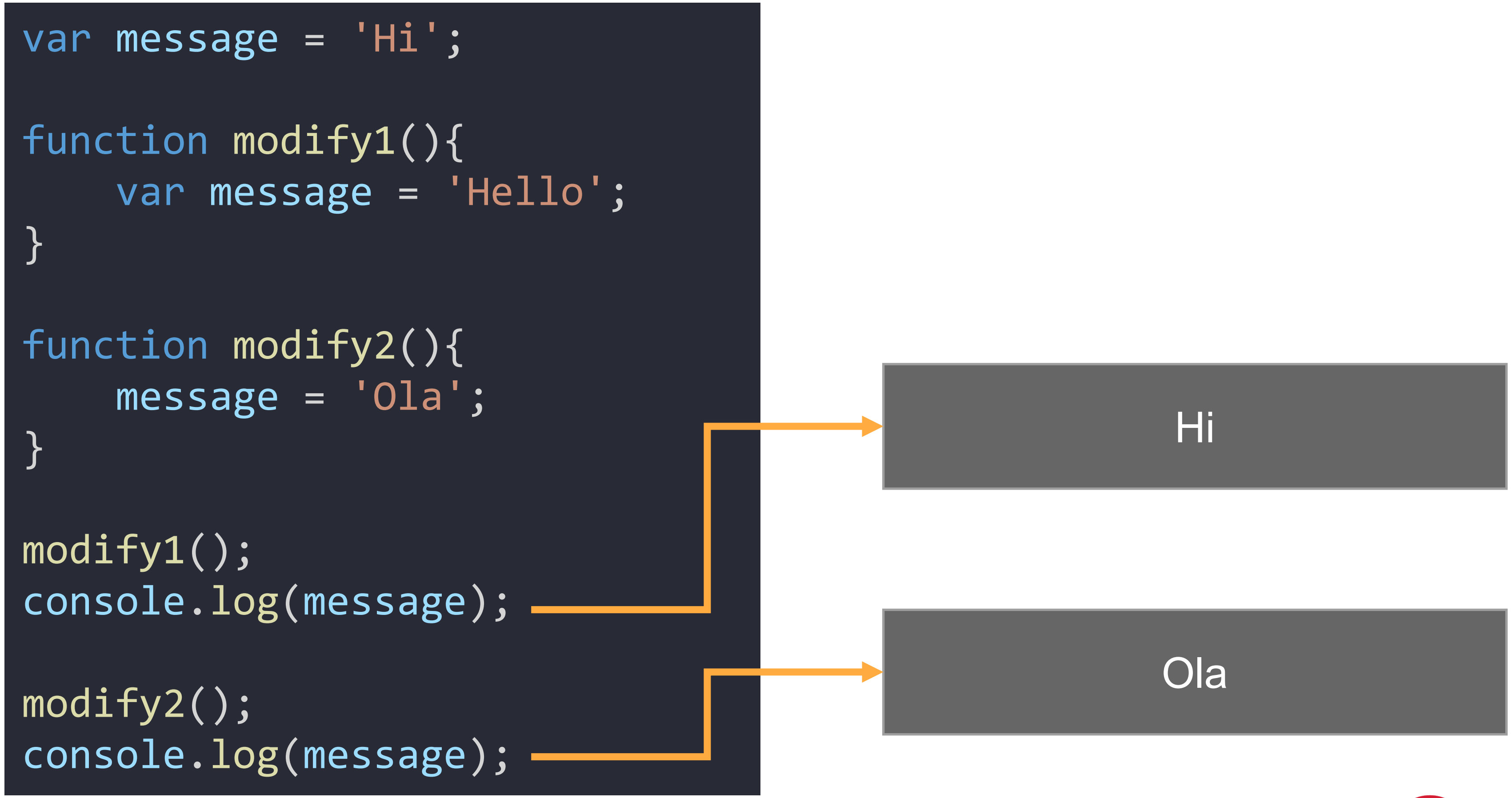


```
<script type="text/javascript" src="file.js"></ script>
```

Statements, comments, and variables

- Statements: separated by new line or semicolon.
- Comment:
    - Single line: // here is a comment
    - Multi line: /* here is a comment */
- Loops and iteration:
    - for, for…in, for…of, do…while, while.
- Variables:
    - Assignment operator (=) to assign values.

Variable scope



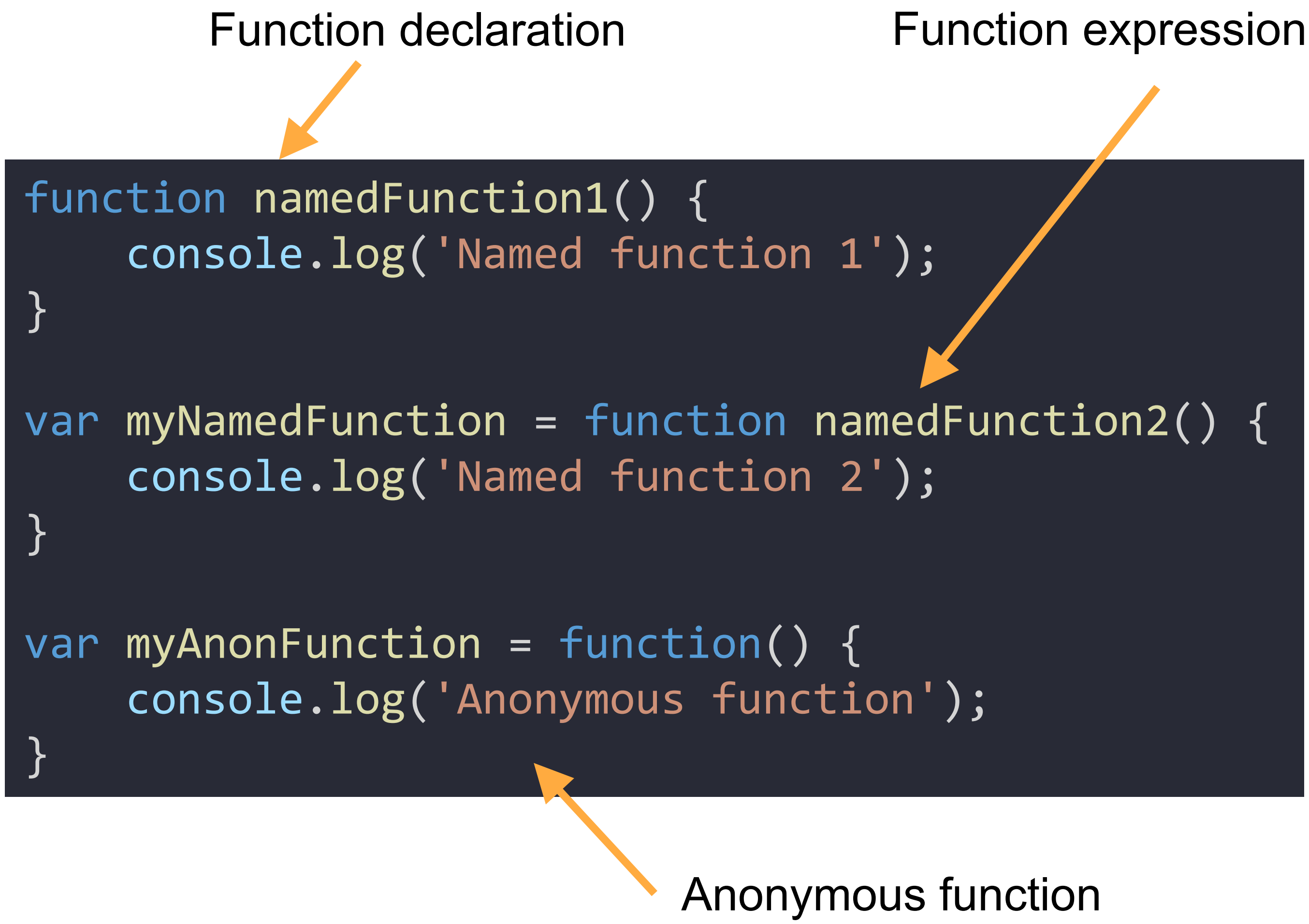
```
var message = 'Hi';

function modify1(){
    var message = 'Hello';
}

function modify2(){
    message = 'Ola';
}

modify1();
console.log(message);

modify2();
console.log(message);
```

```
Hi
```

```
Ola
```

Functions

- Different ways to define functions:
    - Named
    - Anonymous
- Function expressions cannot be used before they appear in the code.



Function declaration

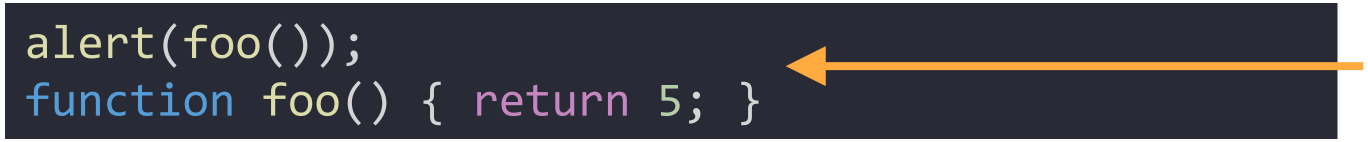
Function expression

```
function namedFunction1() {
    console.log('Named function 1');
}

var myNamedFunction = function namedFunction2() {
    console.log('Named function 2');
}

var myAnonFunction = function() {
    console.log('Anonymous function');
}
```

Anonymous function

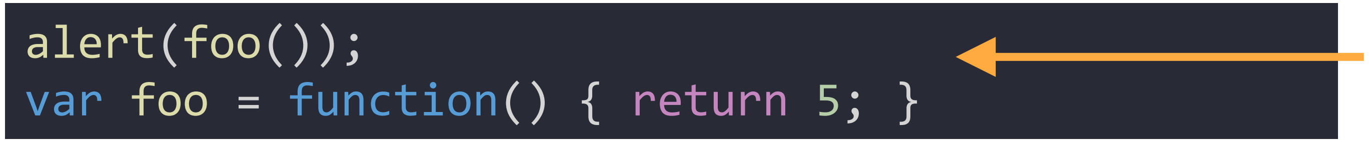
Functions

- Function declarations load before any code is executed, while function expressions load only when the interpreter reaches that line.
- Function expressions: closures, arguments to other functions



```
alert(foo());
function foo() { return 5; }
```

Function declaration: error in this case, as foo wasn't loaded yet.



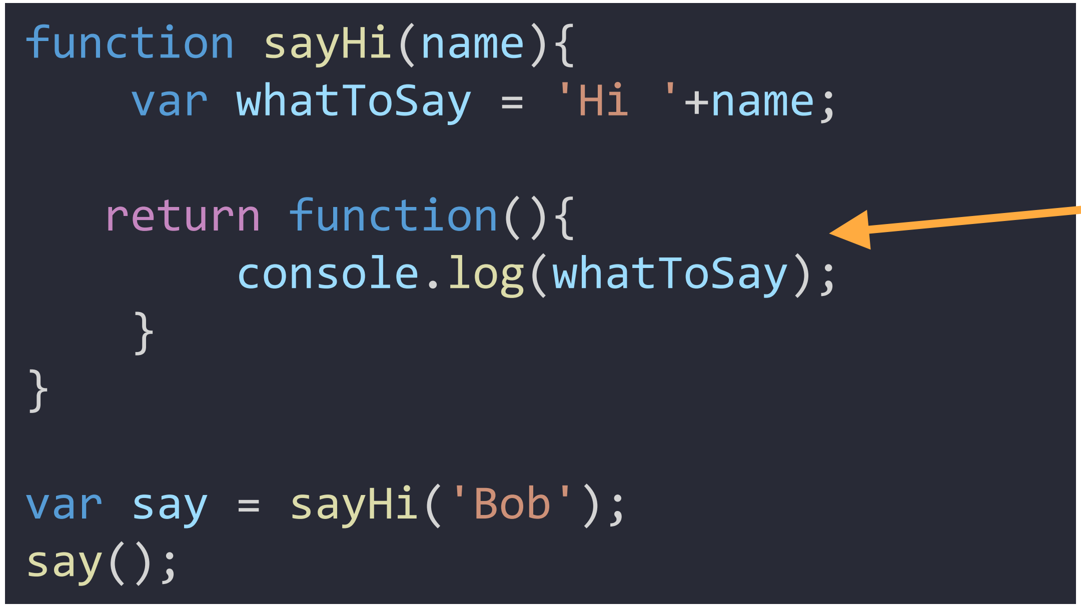
```
alert(foo());
var foo = function() { return 5; }
```

Function expression: alerts 5. Declarations are loaded before any code can run.

Functions

- Functions are first-class objects:
  - Supports passing functions to other functions.
  - Returning them as values from other functions.
  - Assigning them to variables or data structures.
- Closure:
  - Function that maintains the local variables of a function, after the function has returned.

Closure example



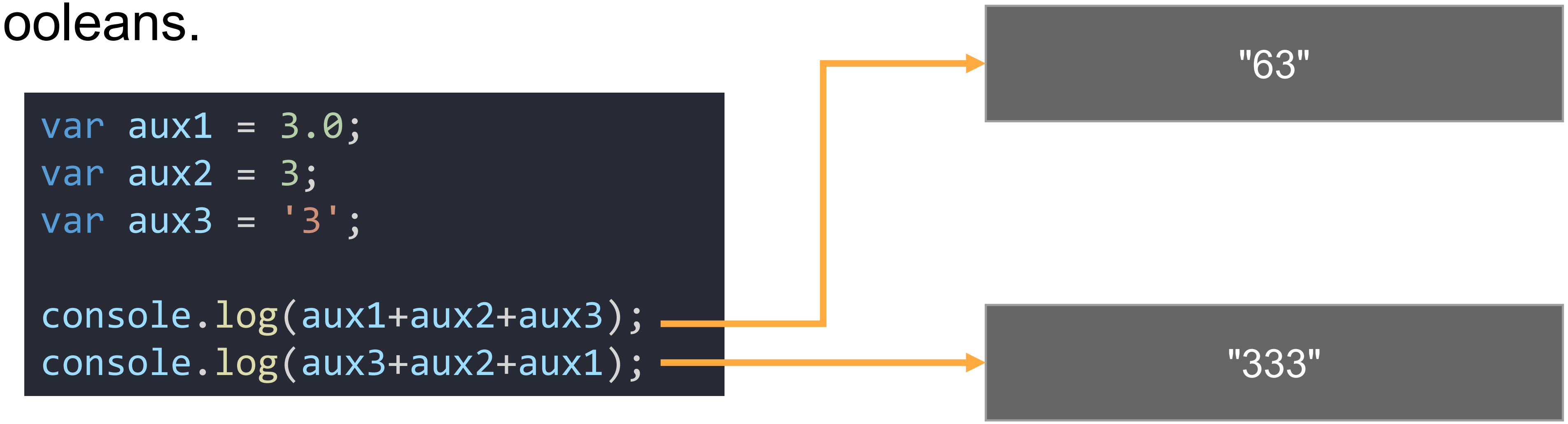
```
function sayHi(name){
    var whatToSay = 'Hi '+name;

   return function(){
        console.log(whatToSay);
    }
}

var say = sayHi('Bob');
say();
```

A closure: a function inside a function

No matter where it is executed, closure function will always remember variables from sayHi.

Data types: numbers and strings

- Numbers: a primitive data type (32-bit float).
- String: sequence of characters.
- Booleans.



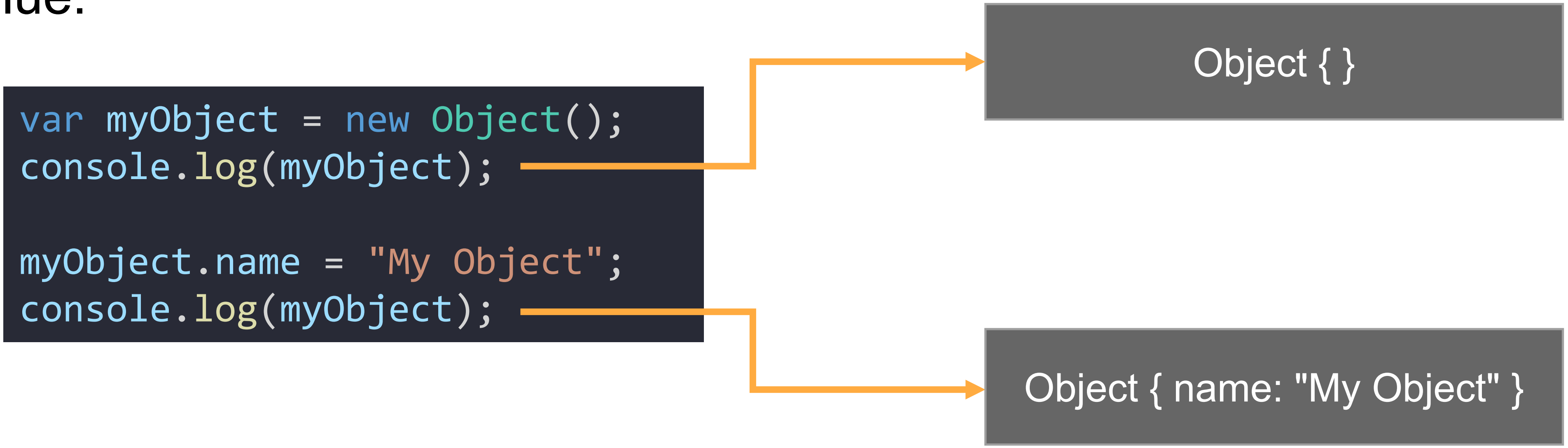
```
var aux1 = 3.0;
var aux2 = 3;
var aux3 = '3';

console.log(aux1+aux2+aux3);
console.log(aux3+aux2+aux1);
```

"63"

"333"

Objects

- In JavaScript, objects are a collection of properties with a name and a value.



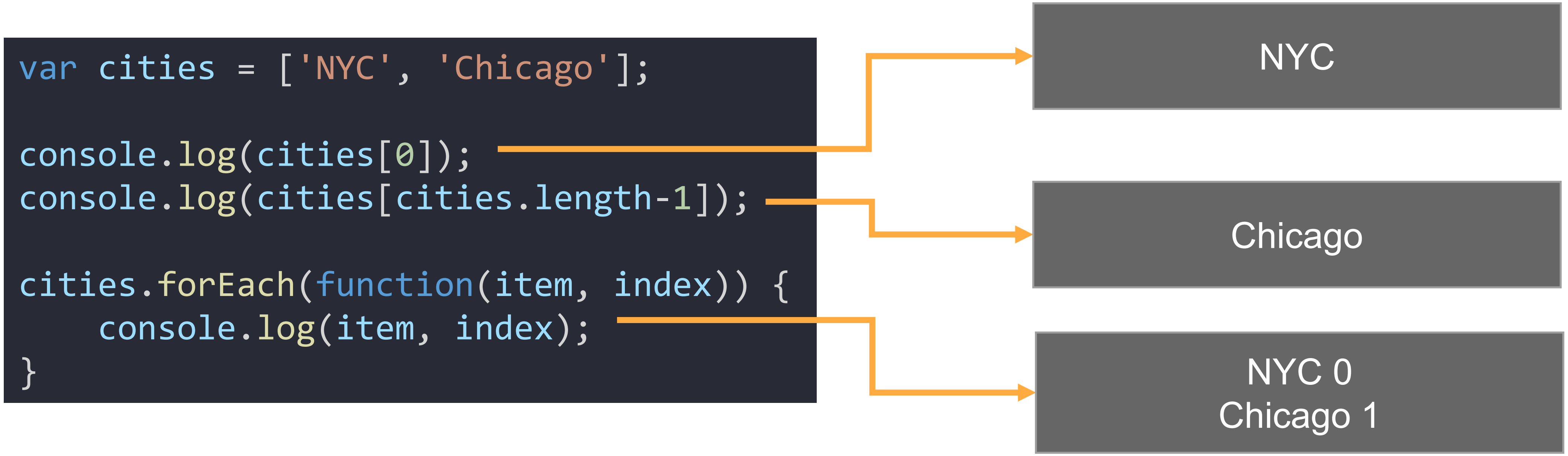
```
var myObject = new Object();
console.log(myObject);

myObject.name = "My Object";
console.log(myObject);
```

Object { }

Object { name: "My Object" }

Arrays

- List-like objects.



```
var cities = ['NYC', 'Chicago'];

console.log(cities[0]);
console.log(cities[cities.length-1]);

cities.forEach(function(item, index)) {
    console.log(item, index);
}
```

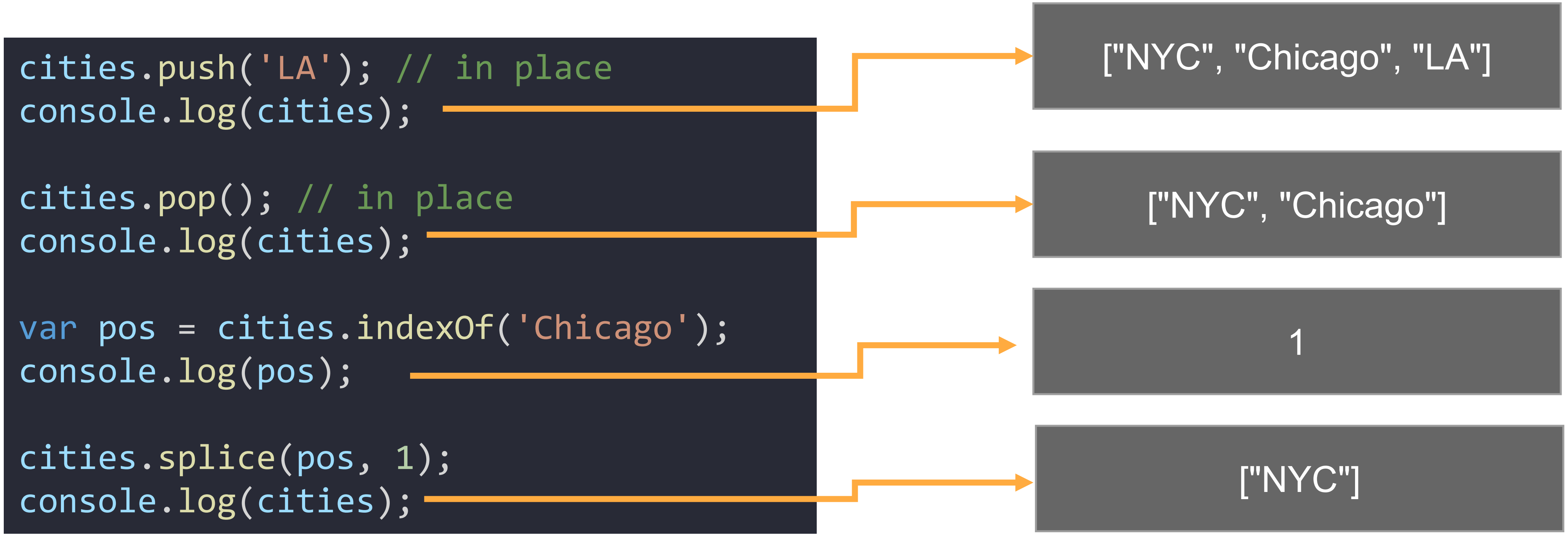
```
NYC
```

```
Chicago
```

```
NYC 0
Chicago 1
```

Arrays

- List-like objects.



```
cities.push('LA'); // in place
console.log(cities);

cities.pop(); // in place
console.log(cities);

var pos = cities.indexOf('Chicago');
console.log(pos);

cities.splice(pos, 1);
console.log(cities);
```

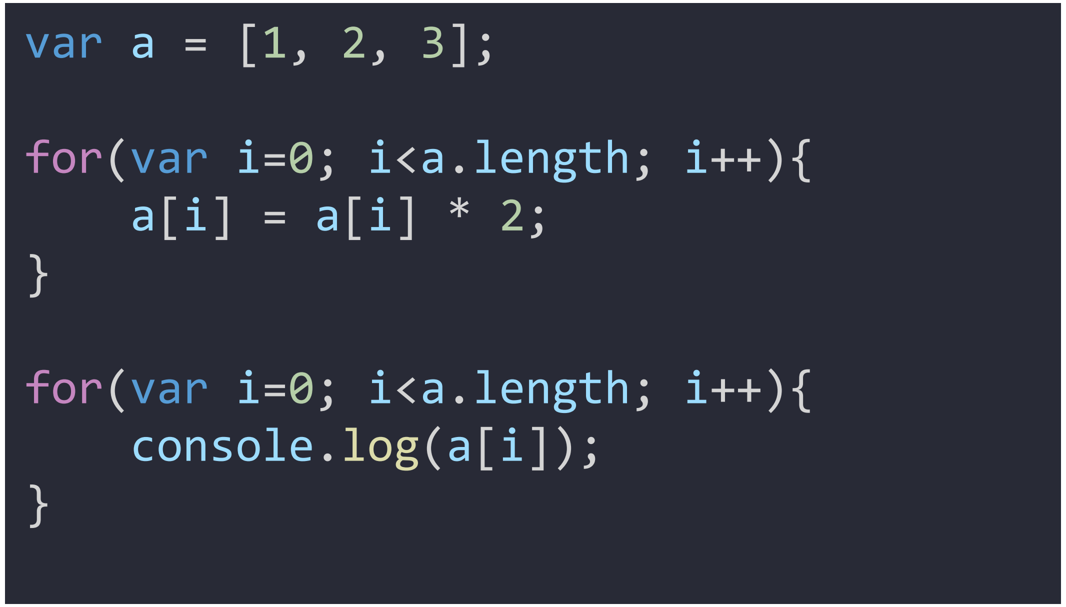
```
["NYC", "Chicago", "LA"]
```

```
["NYC", "Chicago"]
```

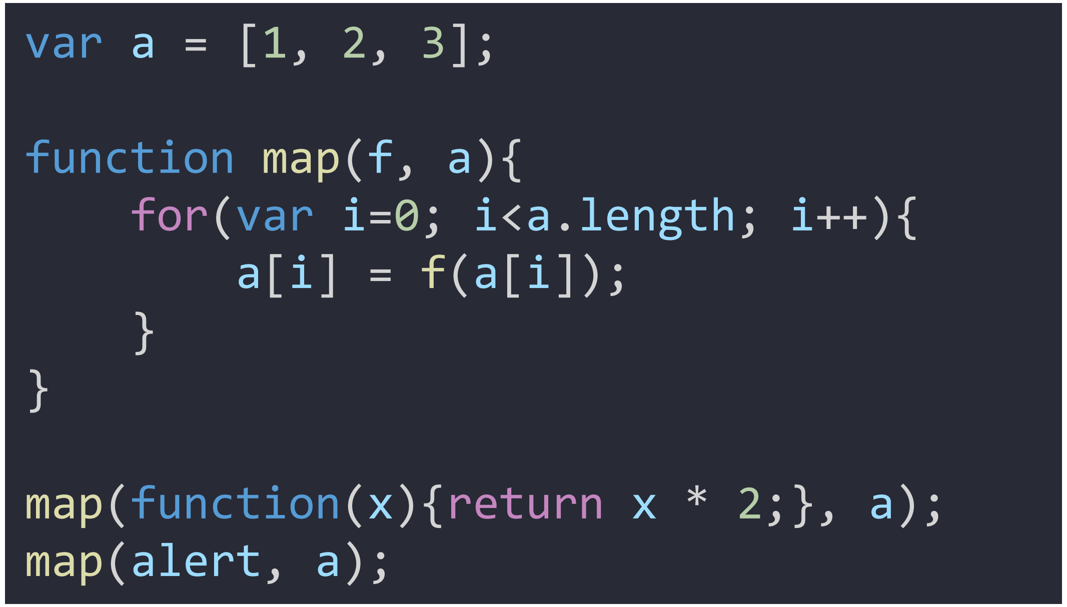
```
1
```

```
["NYC"]
```

Example: map

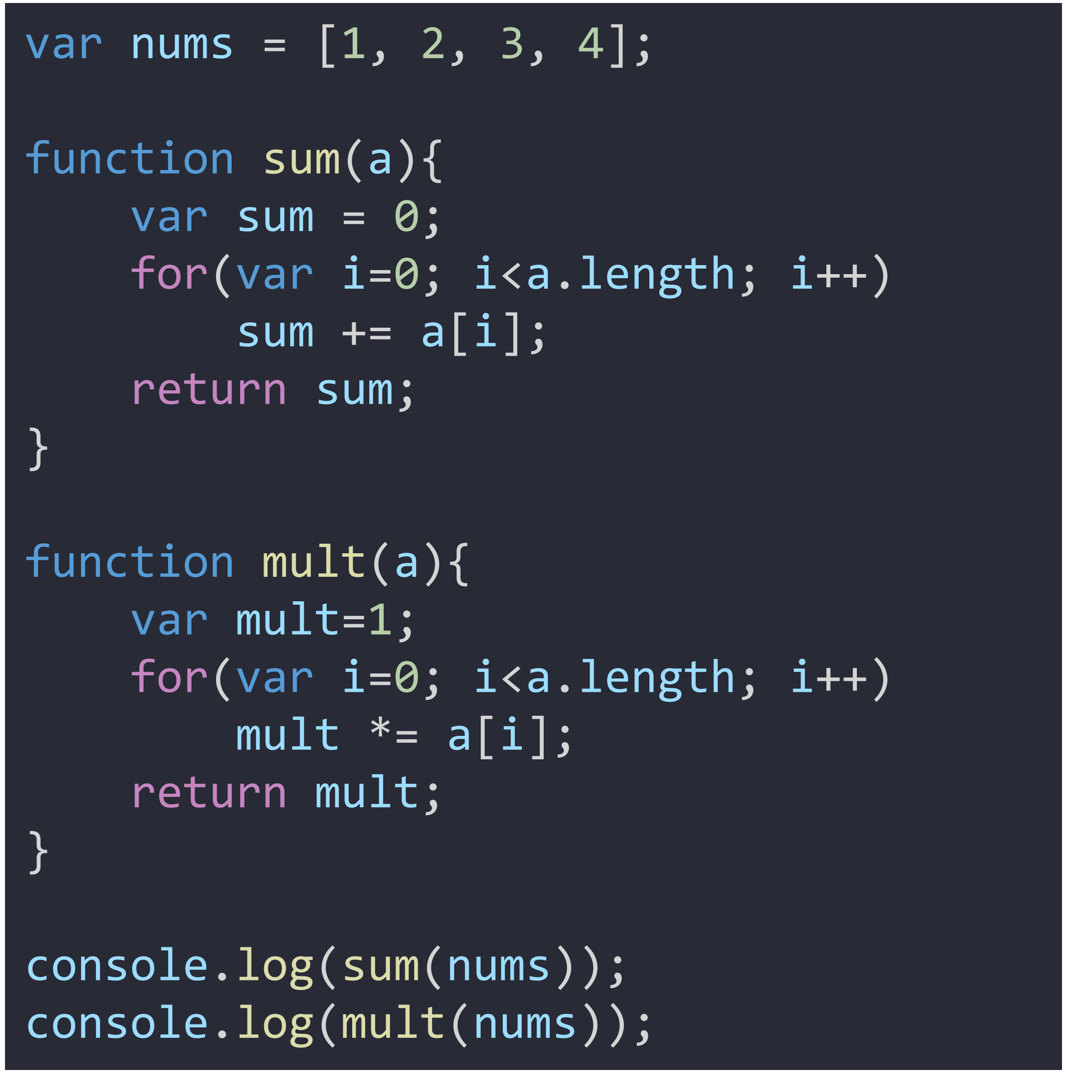


```
var a = [1, 2, 3];

for(var i=0; i<a.length; i++){
    a[i] = a[i] * 2;
}

for(var i=0; i<a.length; i++){
    console.log(a[i]);
}
```

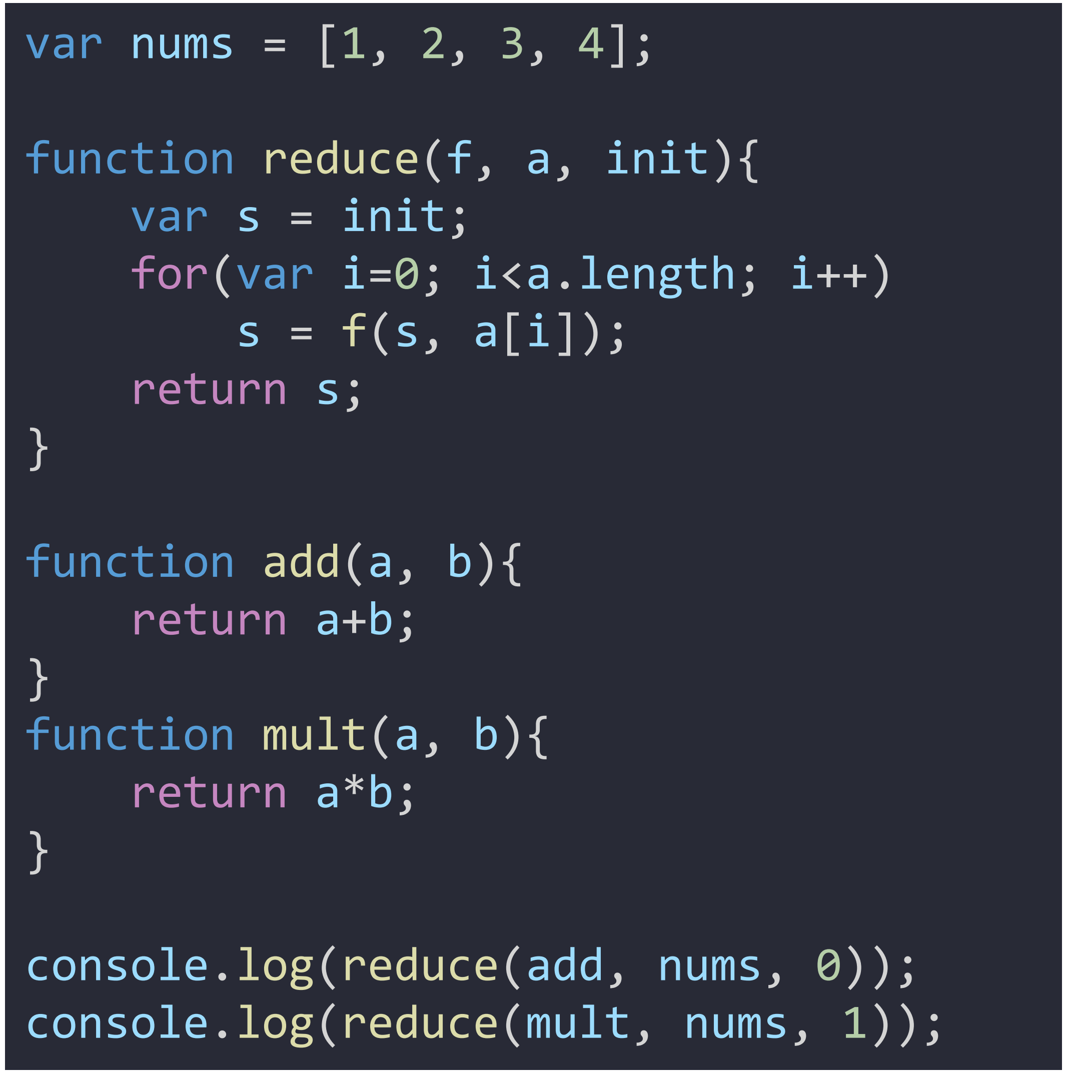


```
var a = [1, 2, 3];

function map(f, a){
    for(var i=0; i<a.length; i++){
        a[i] = f(a[i]);
    }
}

map(function(x){return x * 2;}, a);
map(alert, a);
```

Example: reduce



```
var nums = [1, 2, 3, 4];

function sum(a){
    var sum = 0;
    for(var i=0; i<a.length; i++)
        sum += a[i];
    return sum;
}

function mult(a){
    var mult=1;
    for(var i=0; i<a.length; i++)
        mult *= a[i];
    return mult;
}

console.log(sum(nums));
console.log(mult(nums));
```

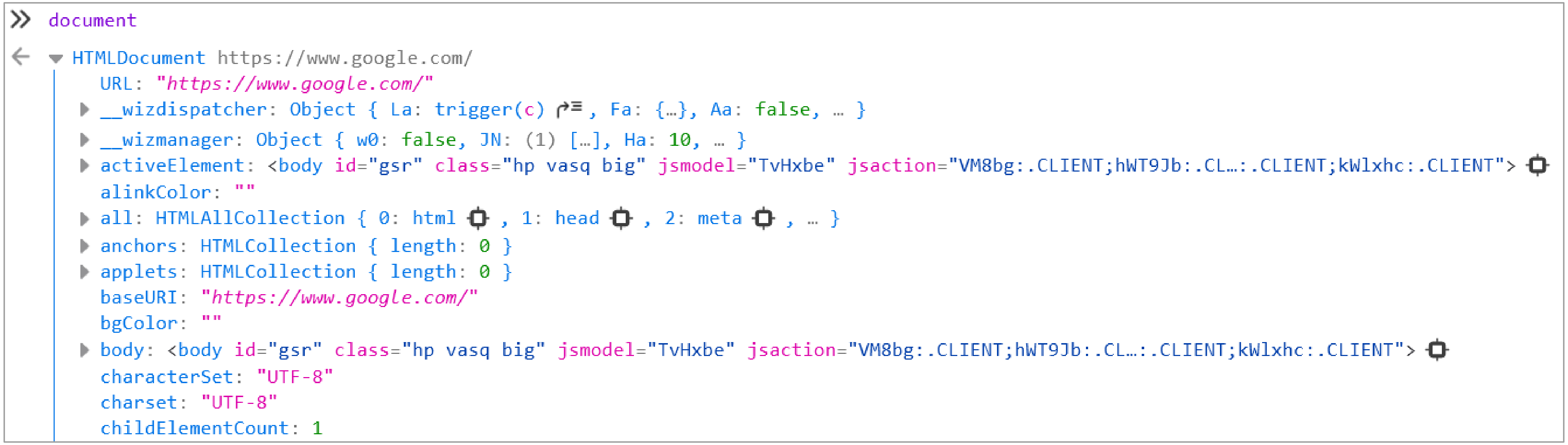


```
var nums = [1, 2, 3, 4];

function reduce(f, a, init){
    var s = init;
    for(var i=0; i<a.length; i++)
        s = f(s, a[i]);
    return s;
}

function add(a, b){
    return a+b;
}
function mult(a, b){
    return a*b;
}

console.log(reduce(add, nums, 0));
console.log(reduce(mult, nums, 1));
```

Manipulating documents

- So far: HTML, CSS, JavaScript.
- But how can we use JavaScript to modify nodes from DOM?
- Answer: **document object**.
- When an HTML document is loaded by a browser, it becomes a **document object**, containing the root node of the HTML document.

Document object

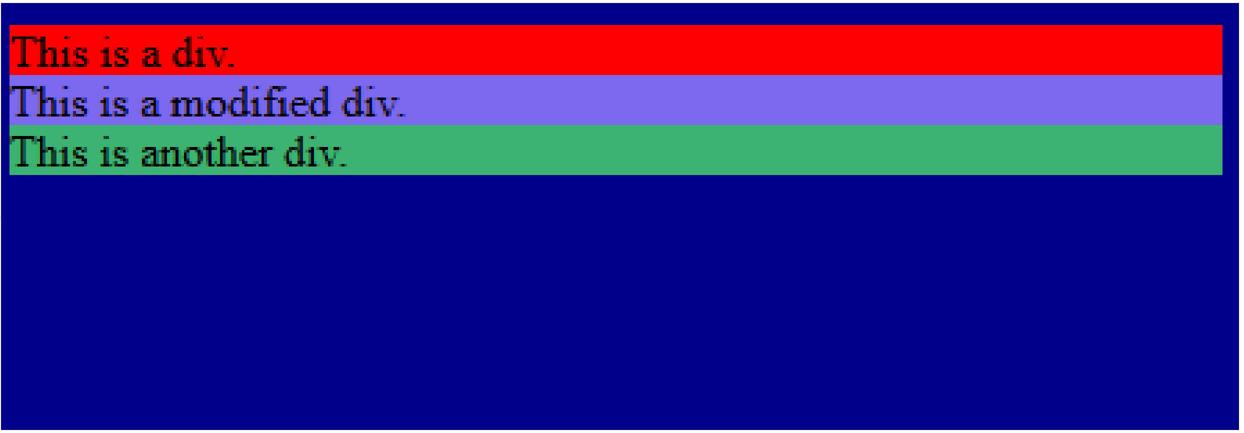


```
>> document
← ▾ HTMLDocument https://www.google.com/
      URL: "https://www.google.com/"
    ▸ __wizdispatcher: Object { La: trigger(c) , Fa: {…}, Aa: false, … }
    ▸ __wizmanager: Object { w0: false, JN: (1) […], Ha: 10, … }
    ▸ activeElement: <body id="gsr" class="hp vasq big" jsmodel="TvHxbe" jsaction="VM8bg:.CLIENT;hWT9Jb:.CL…:.CLIENT;kWlxhc:.CLIENT">
      alinkColor: ""
    ▸ all: HTMLAllCollection { 0: html , 1: head , 2: meta , … }
    ▸ anchors: HTMLCollection { length: 0 }
    ▸ applets: HTMLCollection { length: 0 }
      baseURI: "https://www.google.com/"
      bgColor: ""
    ▸ body: <body id="gsr" class="hp vasq big" jsmodel="TvHxbe" jsaction="VM8bg:.CLIENT;hWT9Jb:.CL…:.CLIENT;kWlxhc:.CLIENT">
      characterSet: "UTF-8"
      charset: "UTF-8"
      childElementCount: 1
```

DOM elements using selectors

```
var allDivs = document.querySelector('div');
var myDiv = document.querySelector('#mydiv');
var mySecondDiv = document.querySelector('#myseconddiv');
var myClass = document.querySelector('.myclass');
```

```
mySecondDiv.textContent = 'This is a modified div.';
```



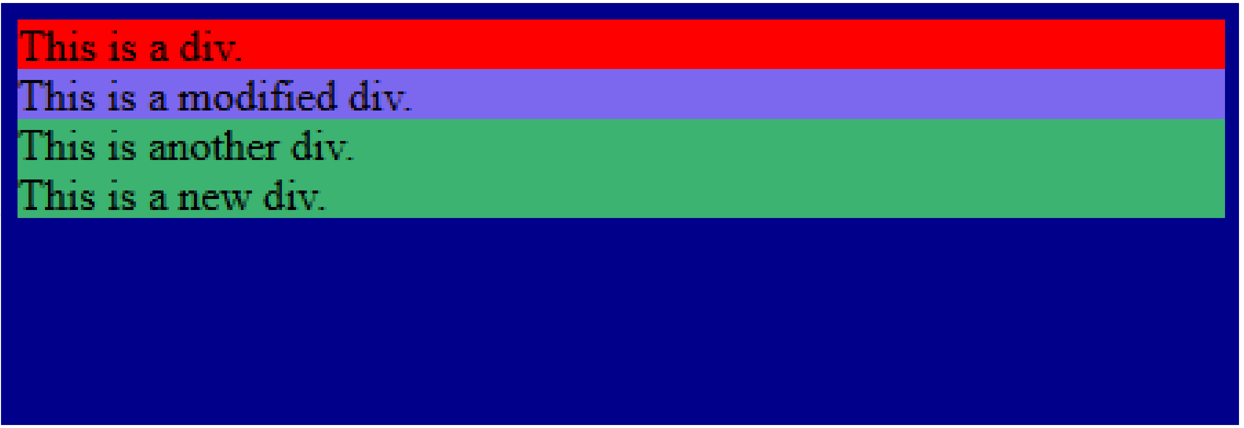
This is a div.

This is a modified div.

This is another div.

DOM elements using selectors

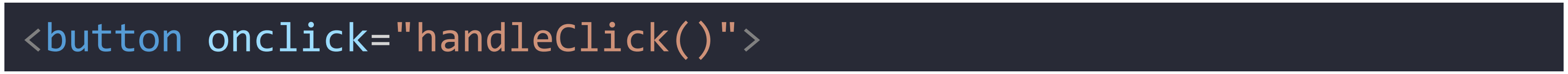
```
var newDiv = document.createElement('div');
newDiv.textContent = 'This is a new div.';
newDiv.className = 'myclass';
document.querySelector('body').appendChild(newDiv);
```



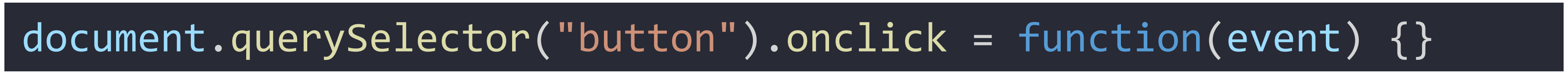
This is a div.
This is a modified div.
This is another div.
This is a new div.

Event handlers

- Events are actions like being clicked, pressed keys, getting focus, etc.
- Different ways to specify handlers for a particular event:

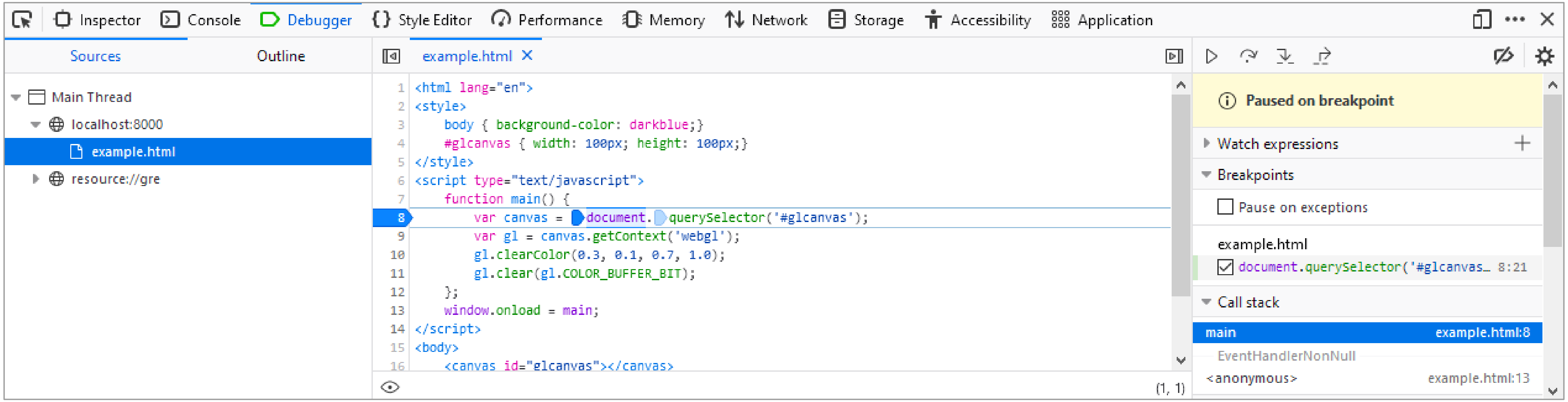


```html
<button onclick="handleClick()">
```



```javascript
document.querySelector("button").onclick = function(event) {}
```

Debugging JavaScript



```
<html lang="en">
<style>
    body { background-color: darkblue;}
    #glcanvas { width: 100px; height: 100px;}
</style>
<script type="text/javascript">
    function main() {
        var canvas = document.querySelector('#glcanvas');
        var gl = canvas.getContext('webgl');
        gl.clearColor(0.3, 0.1, 0.7, 1.0);
        gl.clear(gl.COLOR_BUFFER_BIT);
    };
    window.onload = main;
</script>
<body>
    <canvas id="glcanvas"></canvas>
```

Paused on breakpoint

Watch expressions

Breakpoints

□ Pause on exceptions

example.html

☑ document.querySelector('#glcanvas… 8:21

Call stack

main	example.html:8
EventHandlerNonNull	
<anonymous>	example.html:13

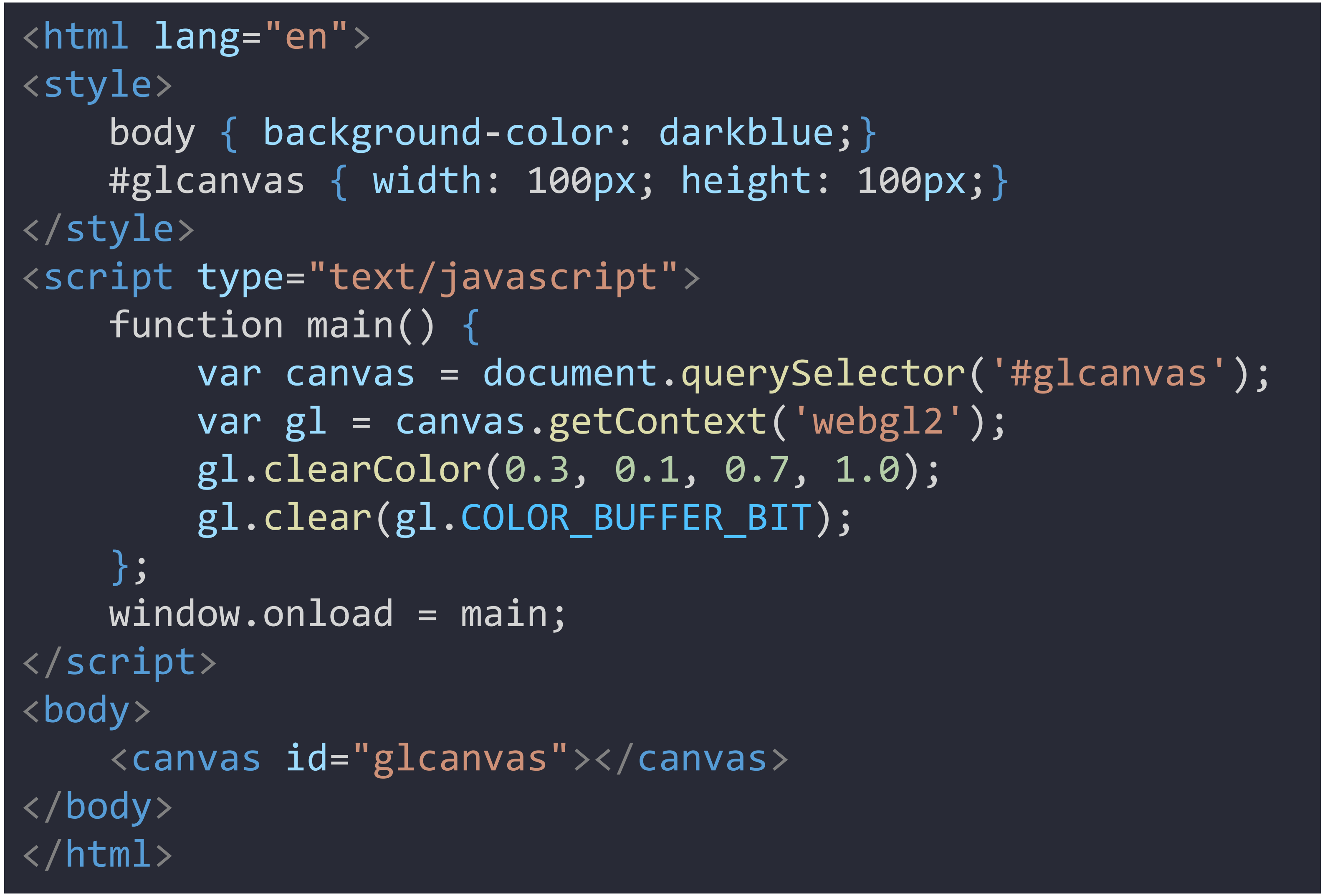
Finally drawing something

- Several ways to draw graphics on the web:
  - SVG
    - XML-based format for vector images.
    - Simple option for small data.
    - Easy event and CSS integration.
  - Canvas
    - HTML element.
    - No object-level interaction.
  - WebGL
    - Complex 3D geometries.
    - Uses rendering pipeline.
    - Hardware acceleration.

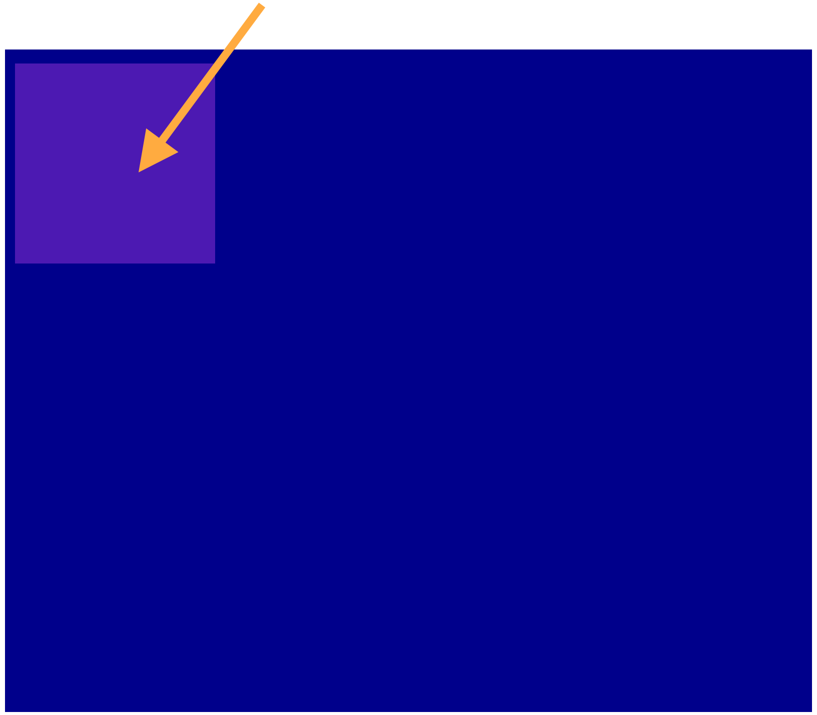
WebGL: a bird’s-eye view

- API for rendering graphics within a web browser without plug-ins.
- Hardware accelerated.
- Shader based (no fixed-function API).
  - Fixed function pipeline: set of calls for matrix transformation, lighting.
  - Programmable pipeline: shaders for vertex and fragment processing.
- WebGL 2.0 based on OpenGL ES 3.0.

WebGL: a bird’s-eye view



```
<html lang="en">
<style>
    body { background-color: darkblue;}
    #glcanvas { width: 100px; height: 100px;}
</style>
<script type="text/javascript">
    function main() {
        var canvas = document.querySelector('#glcanvas');
        var gl = canvas.getContext('webgl2');
        gl.clearColor(0.3, 0.1, 0.7, 1.0);
        gl.clear(gl.COLOR_BUFFER_BIT);
    };
    window.onload = main;
</script>
<body>
    <canvas id="glcanvas"></canvas>
</body>
</html>
```



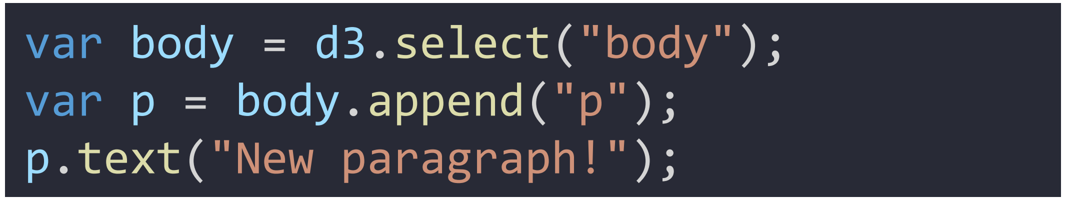
WebGL canvas

D3.js: a bird’s-eye view

- Library for manipulating documents based on data.
- Facilitates DOM manipulation to visualize data.
- D3 is not:
  - a visualization library (but you can visualize data using it).
  - a map library (but you can visualize maps using it).
  - restricted to DOM manipulation (there are several others auxiliary functions).

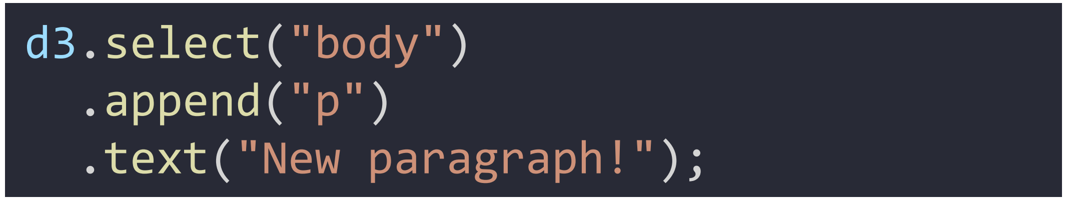
D3.js: first steps

Creating a paragraph



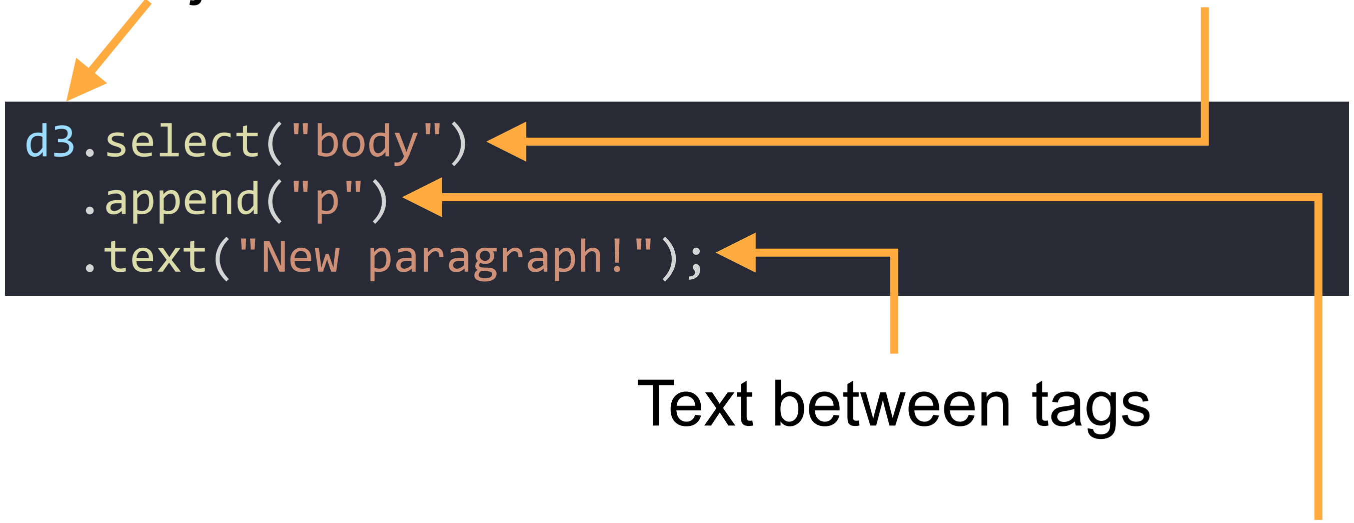
```
var body = d3.select("body");
var p = body.append("p");
p.text("New paragraph!");
```

Creating a paragraph with D3.js



```
d3.select("body")
  .append("p")
  .text("New paragraph!");
```

D3.js: selecting elements



D3 object

Selection: d3.select() and d3.select() accept a CSS selector and return elements

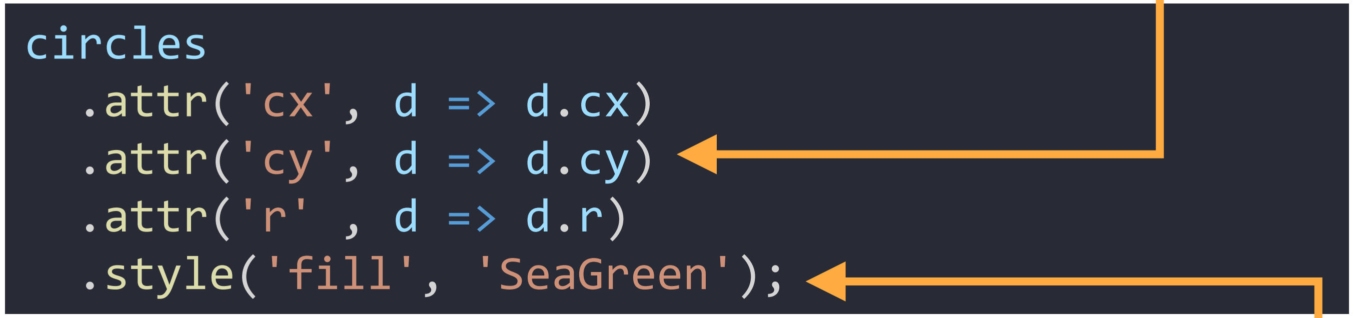
```
d3.select("body")
  .append("p")
  .text("New paragraph!");
```

Text between tags

Append: insert elements in the DOM at the current selection

D3.js: setting attributes

Set attributes, accept anonymous functions

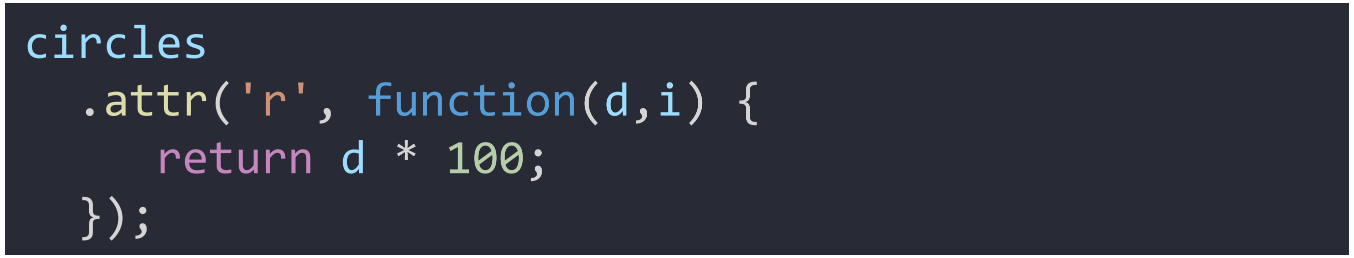


```
circles
  .attr('cx', d => d.cx)
  .attr('cy', d => d.cy)
  .attr('r' , d => d.r)
  .style('fill', 'SeaGreen');
```

Set CSS styles

D3.js: binding data

- Given a selection in D3, one can bind data to it using `.data()`
- This will create a mapping between *each* element in the selection and *each* data element.
  - Default is sequential, i.e, element `i` is mapped to data at index `i`.
- Once bound, one can use data to define attributes:



```
circles
  .attr('r', function(d,i) {
     return d * 100;
  });
```

D3.js: virtual selections

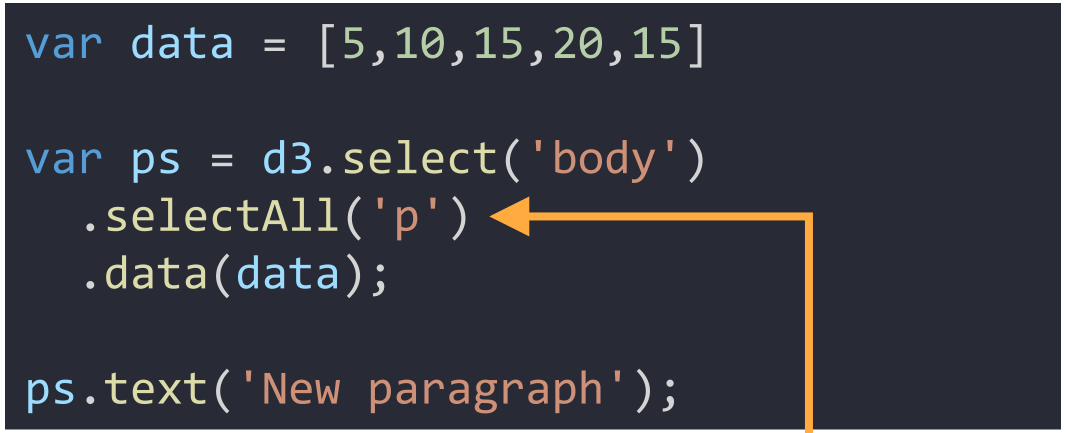
- D3 data operator returns three virtual selections: `enter`, `update`, and `exit`.
- Enter selection: placeholder for missing elements.
- Update selection: update existing elements, bound to data.
- Exit selection: remove remaining elements.

D3.js: virtual selections

```
var data = [5,10,15,20,15]

var ps = d3.select('body')
  .selectAll('p')
  .data(data);
  .enter()
```

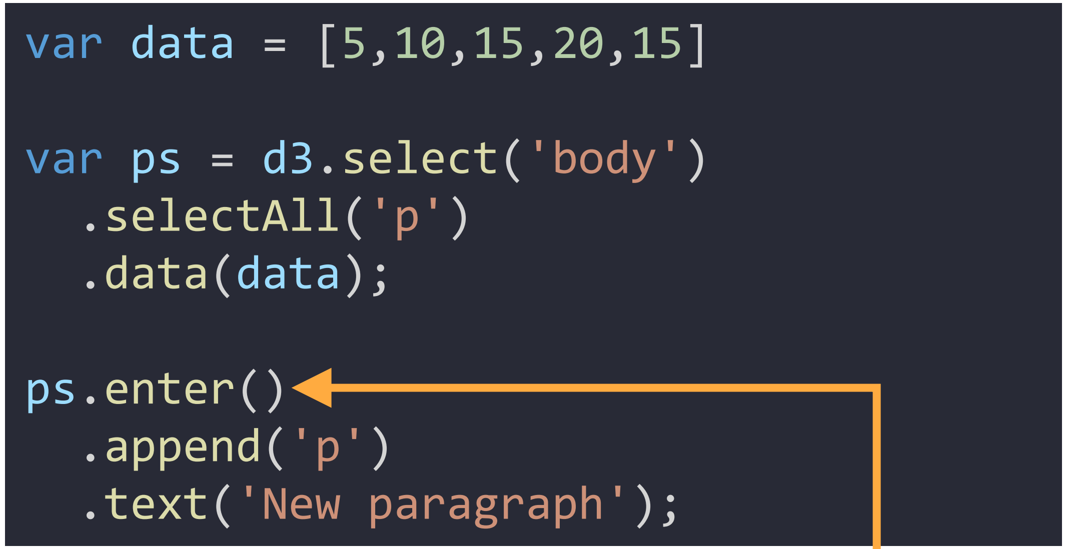
Add placeholder elements for each data element without DOM element correspondent

D3.js: virtual selections



```
var data = [5,10,15,20,15]

var ps = d3.select('body')
  .selectAll('p')
  .data(data);

ps.text('New paragraph');
```

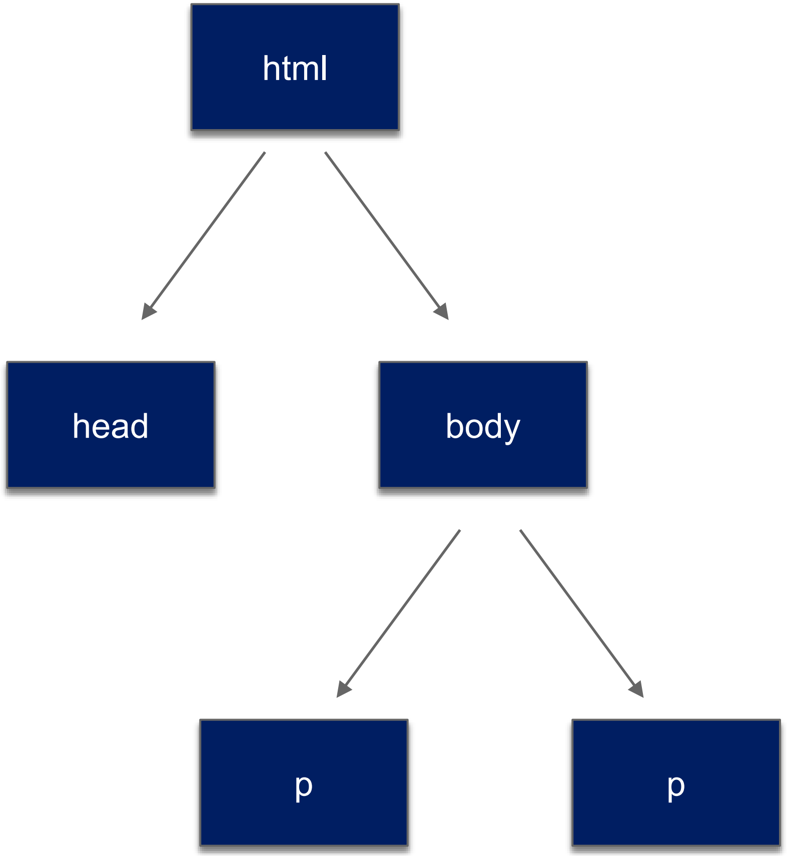
If no previous <p> element inside <body> then this is an **empty** selection



```
var data = [5,10,15,20,15]

var ps = d3.select('body')
  .selectAll('p')
  .data(data);

ps.enter()
  .append('p')
  .text('New paragraph');
```

Creating placeholder elements

D3.js: virtual selections

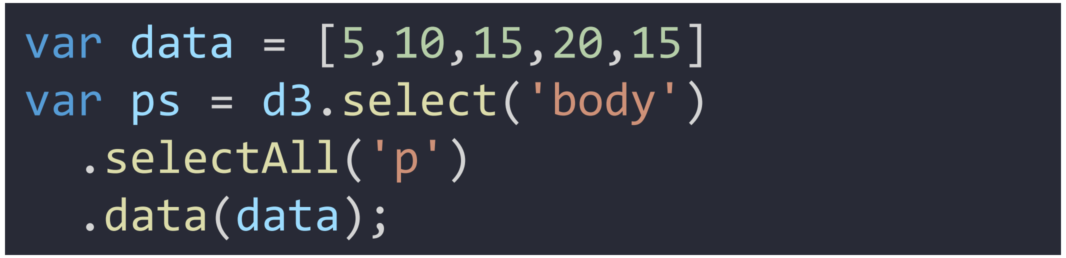

html

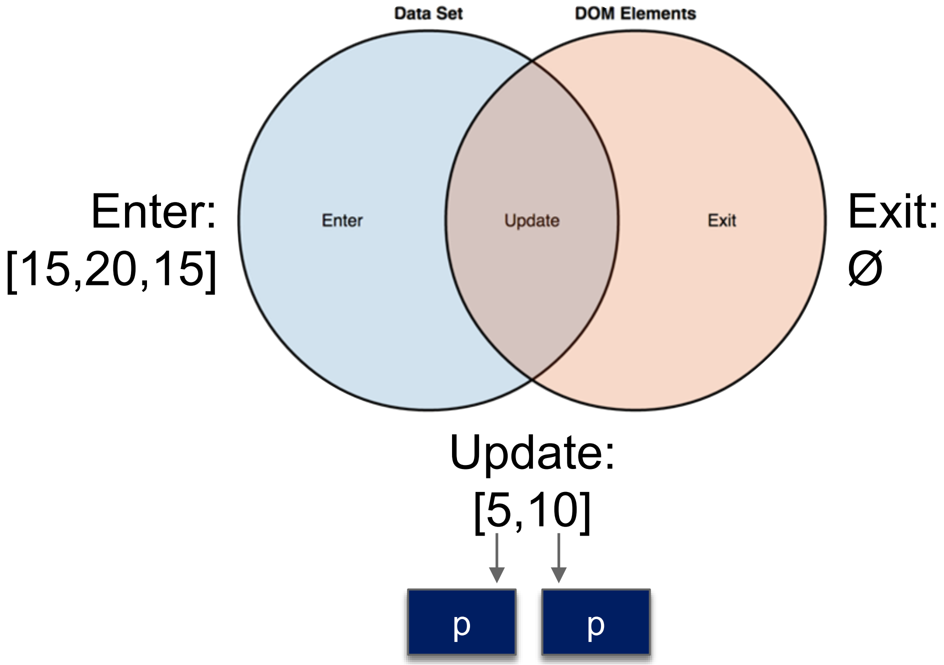
head

body

p

p


```
var data = [5,10,15,20,15]
var ps = d3.select('body')
  .selectAll('p')
  .data(data);
```



Enter:
[15,20,15]

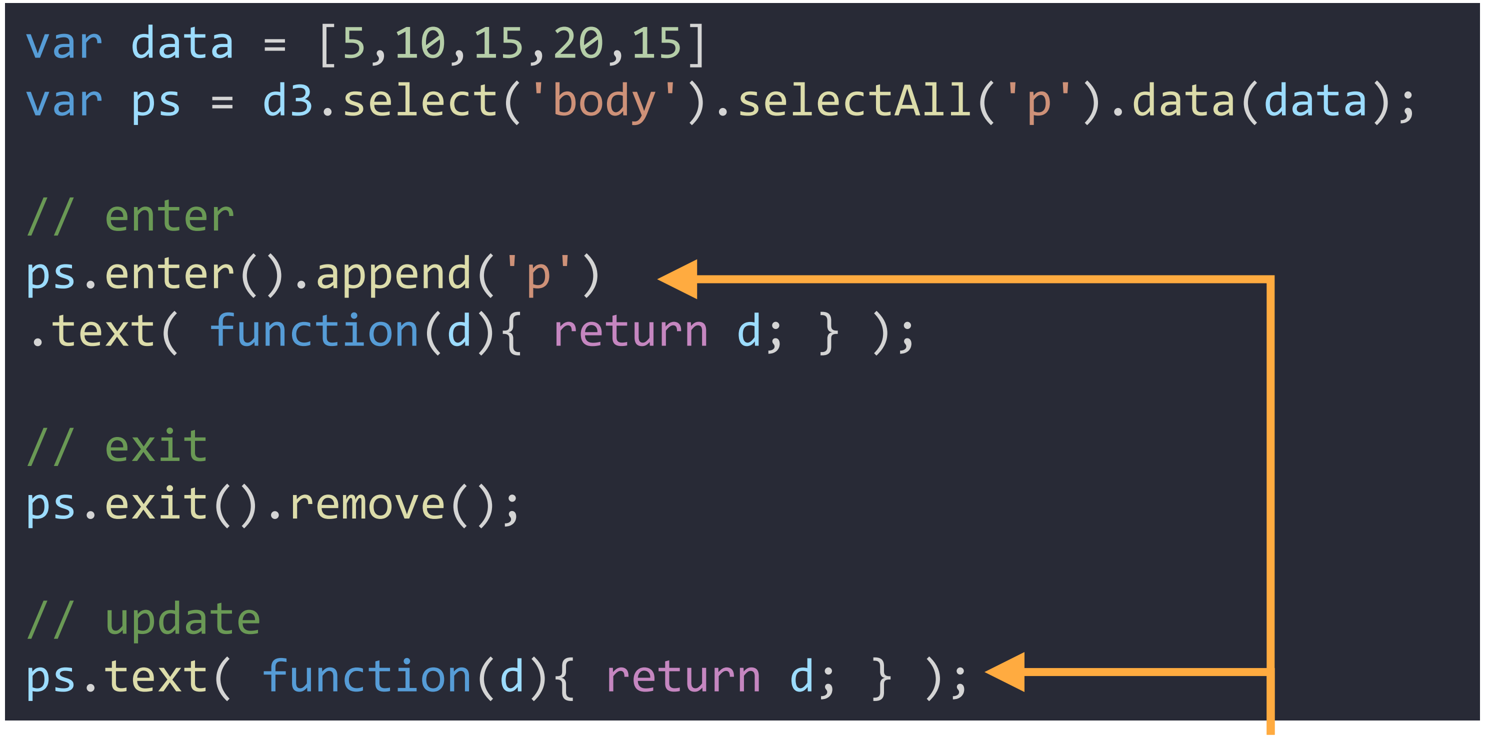
Exit:
Ø

Update:
[5,10]

p

p

D3.js: virtual selections



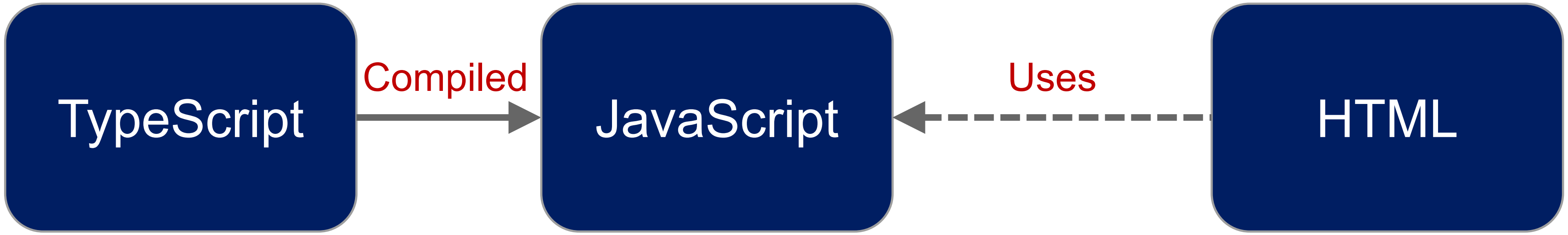
```
var data = [5,10,15,20,15]
var ps = d3.select('body').selectAll('p').data(data);

// enter
ps.enter().append('p')
.text( function(d){ return d; } );

// exit
ps.exit().remove();

// update
ps.text( function(d){ return d; } );
```

Data access

TypeScript and Angular

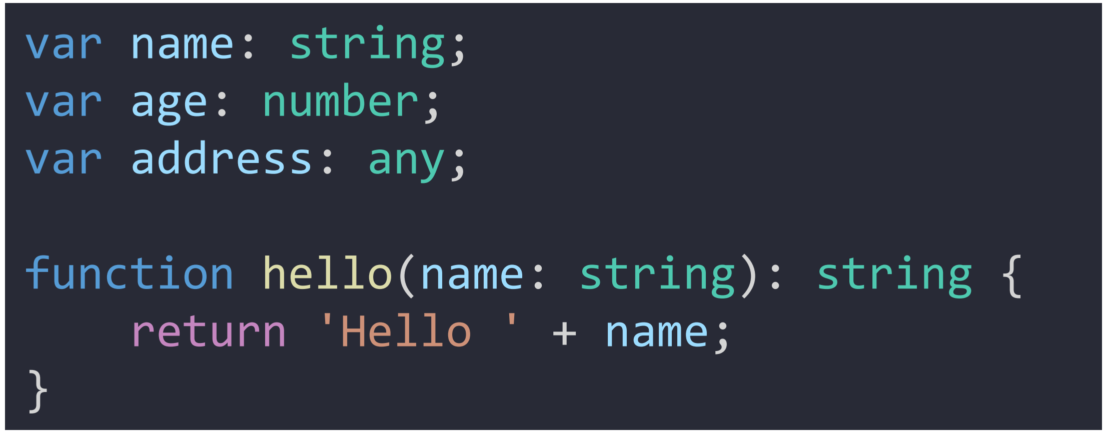
Angular

- Development platform, built on TypeScript (superset of JavaScript).
- Cross-platform component-based framework for building scalable web applications.
- Well-integrated libraries covering a wide variety of features (e.g., client-server communication, DOM, etc.)



TypeScript

- JavaScript-like language: “*JavaScript with Syntax for Types*”
  - Types
  - Classes
  - Imports
- Major change over JavaScript: type checking

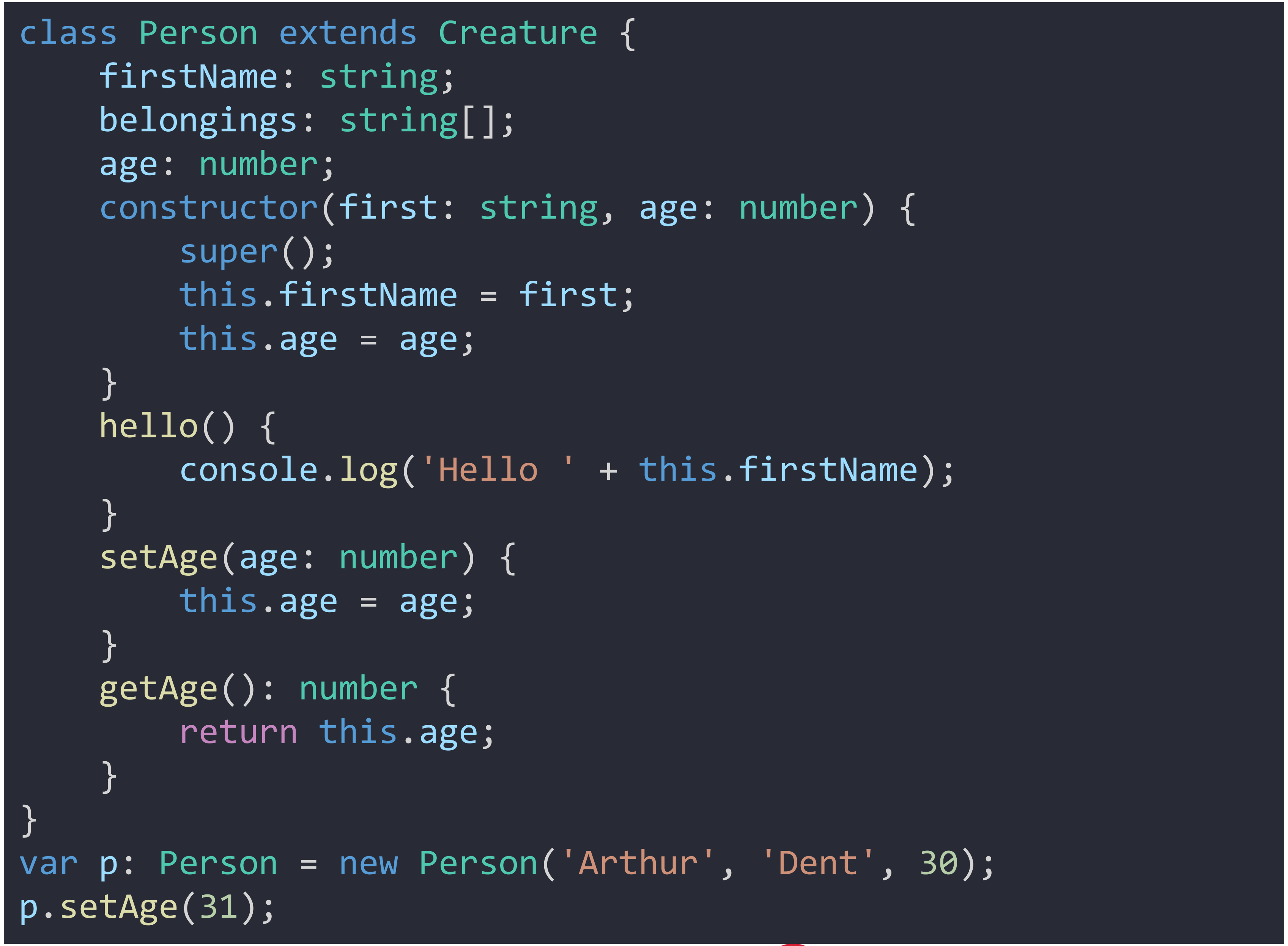


```
var name: string;
var age: number;
var address: any;

function hello(name: string): string {
    return 'Hello ' + name;
}
```

TypeScript

Classes may have properties, methods and constructors.

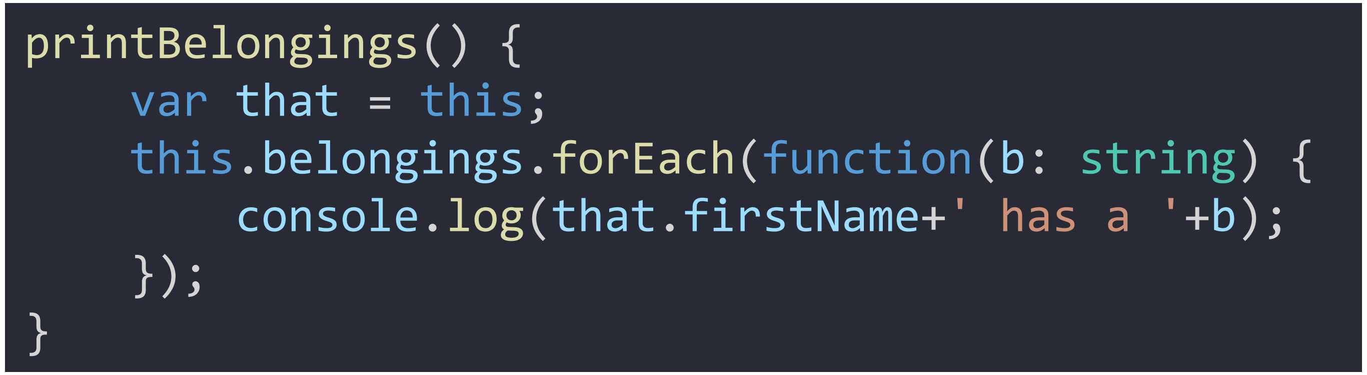
Class inheritance through the `extends` keyword.



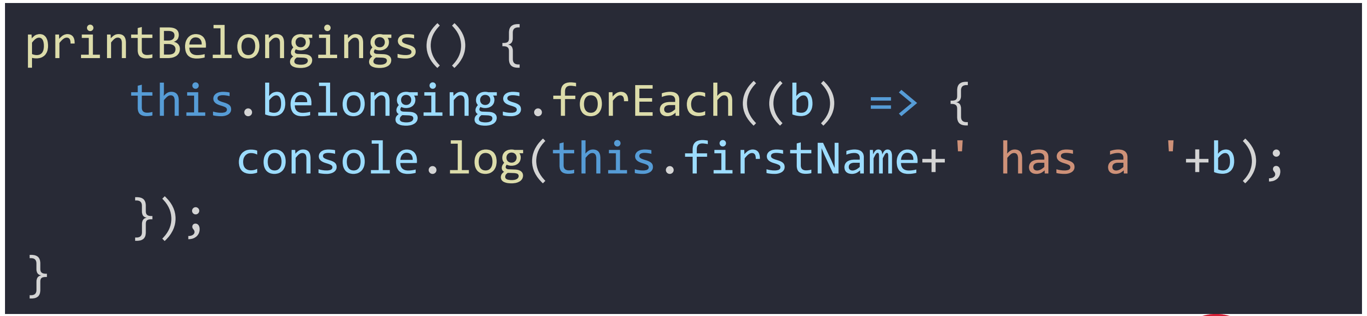
```
class Person extends Creature {
    firstName: string;
    belongings: string[];
    age: number;
    constructor(first: string, age: number) {
        super();
        this.firstName = first;
        this.age = age;
    }
    hello() {
        console.log('Hello ' + this.firstName);
    }
    setAge(age: number) {
        this.age = age;
    }
    getAge(): number {
        return this.age;
    }
}
var p: Person = new Person('Arthur', 'Dent', 30);
p.setAge(31);
```

TypeScript: arrow functions

- Arrow notations are used for anonymous functions.
- Drop the need to use the `function` keyword.
- Arrow functions share the same `this` as the surrounding code.



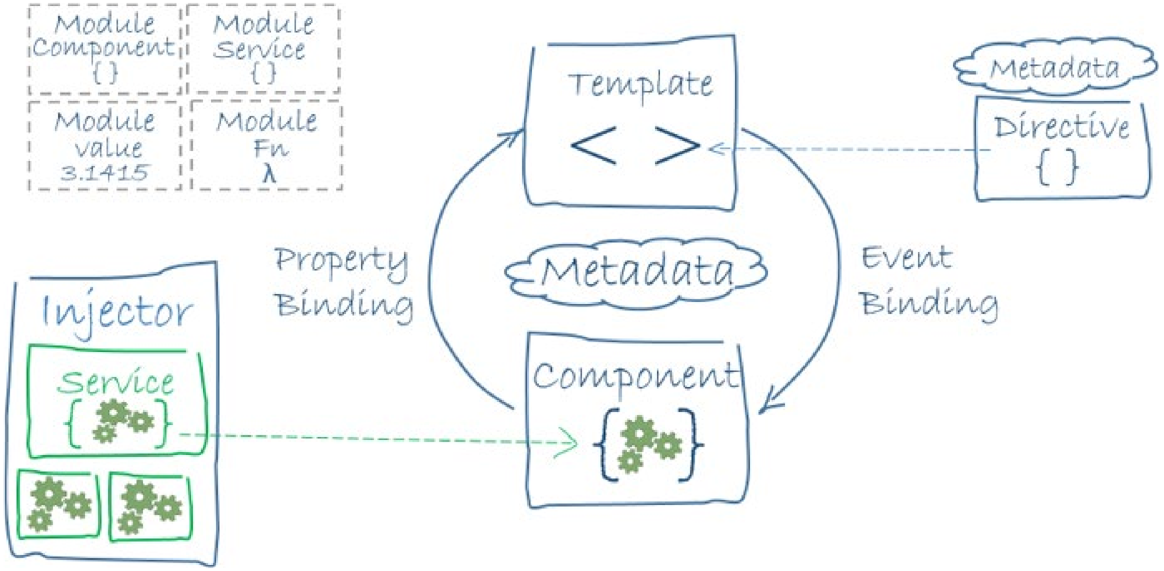
```
printBelongings() {
    var that = this;
    this.belongings.forEach(function(b: string) {
        console.log(that.firstName+' has a '+b);
    });
}
```



```
printBelongings() {
    this.belongings.forEach((b) => {
        console.log(this.firstName+' has a '+b);
    });
}
```

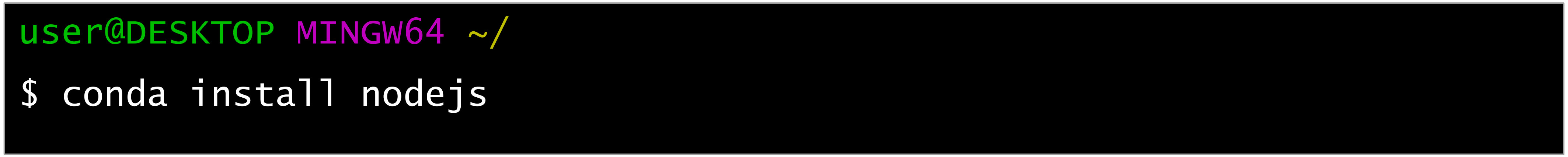
Angular

- An Angular application is a tree of components.
- Top-level component: application itself.
- Components:
  - Composable
  - Reusable
  - Hierarchical
- Angular ≠ AngularJS.

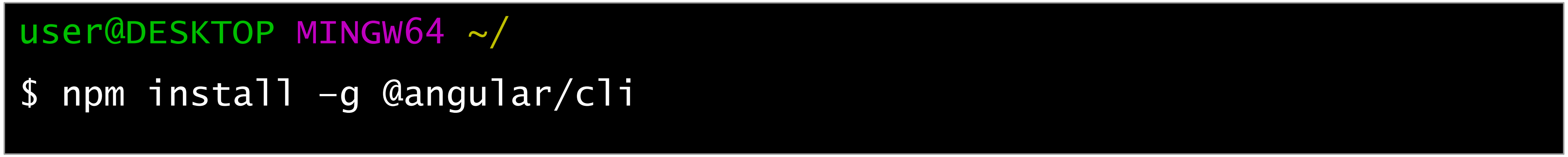


Angular

- Pre-requisites:
  - Node.js: JavaScript runtime environment.
  - npm: package manager for JavaScript.
  - Angular CLI: create projects, generate applications and library code, testing, bundling, and deployment.



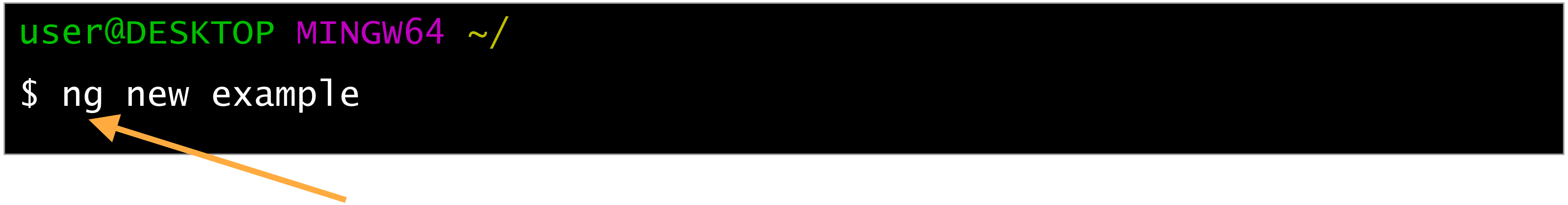
```
user@DESKTOP MINGW64 ~/
$ conda install nodejs
```



```
user@DESKTOP MINGW64 ~/
$ npm install -g @angular/cli
```

Angular: creating an initial application

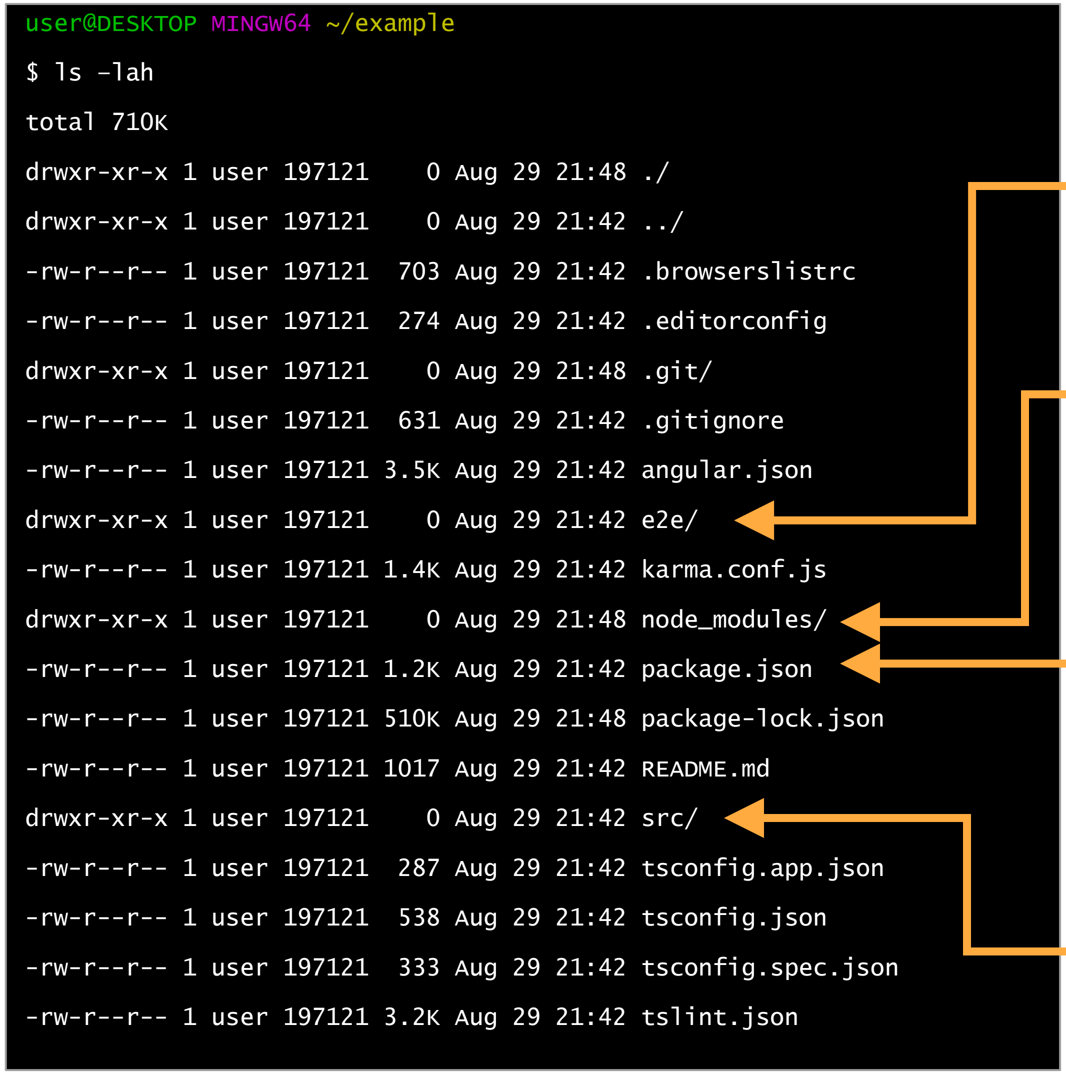
- Create a new application project using angular.
- A project is the set of files that comprise an application or library.



```
user@DESKTOP MINGW64 ~/
$ ng new example
```

ng stands for A**ng**ular!

Angular: creating an initial application



```
user@DESKTOP MINGW64 ~/example
$ ls -lah
total 710K
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 ./
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 ../
-rw-r--r-- 1 user 197121  703 Aug 29 21:42 .browserslistrc
-rw-r--r-- 1 user 197121  274 Aug 29 21:42 .editorconfig
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 .git/
-rw-r--r-- 1 user 197121  631 Aug 29 21:42 .gitignore
-rw-r--r-- 1 user 197121 3.5K Aug 29 21:42 angular.json
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 e2e/
-rw-r--r-- 1 user 197121 1.4K Aug 29 21:42 karma.conf.js
drwxr-xr-x 1 user 197121    0 Aug 29 21:48 node_modules/
-rw-r--r-- 1 user 197121 1.2K Aug 29 21:42 package.json
-rw-r--r-- 1 user 197121 510K Aug 29 21:48 package-lock.json
-rw-r--r-- 1 user 197121 1017 Aug 29 21:42 README.md
drwxr-xr-x 1 user 197121    0 Aug 29 21:42 src/
-rw-r--r-- 1 user 197121  287 Aug 29 21:42 tsconfig.app.json
-rw-r--r-- 1 user 197121  538 Aug 29 21:42 tsconfig.json
-rw-r--r-- 1 user 197121  333 Aug 29 21:42 tsconfig.spec.json
-rw-r--r-- 1 user 197121 3.2K Aug 29 21:42 tslint.json
```

e2e: automated testing

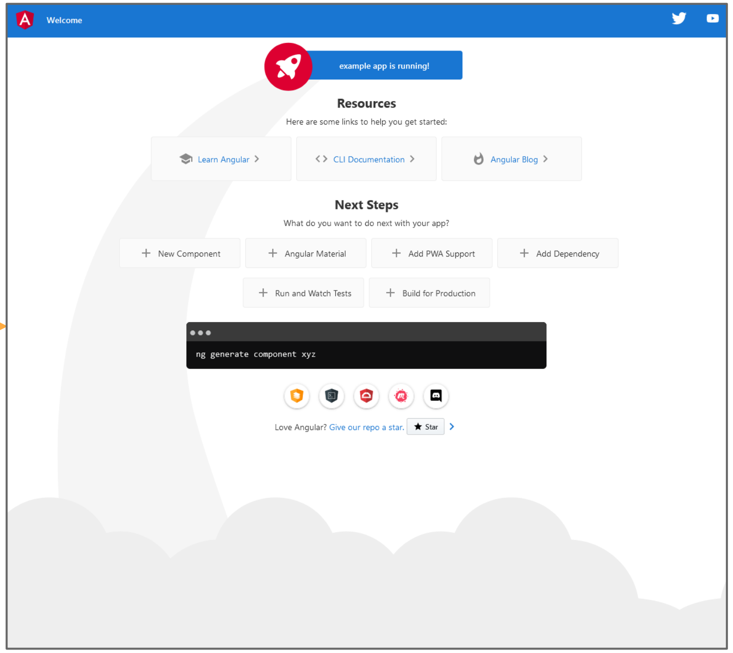
node_modules: development dependencies and dependencies

package.json: information about the libraries added and user in the project, with specified version installed

src: actual project source code, with components, services, etc.

Angular: serving the application

```
user@DESKTOP MINGW64 ~/example
$ ng serve --open
```



example app is running!

Resources

Here are some links to help you get started:

Next Steps

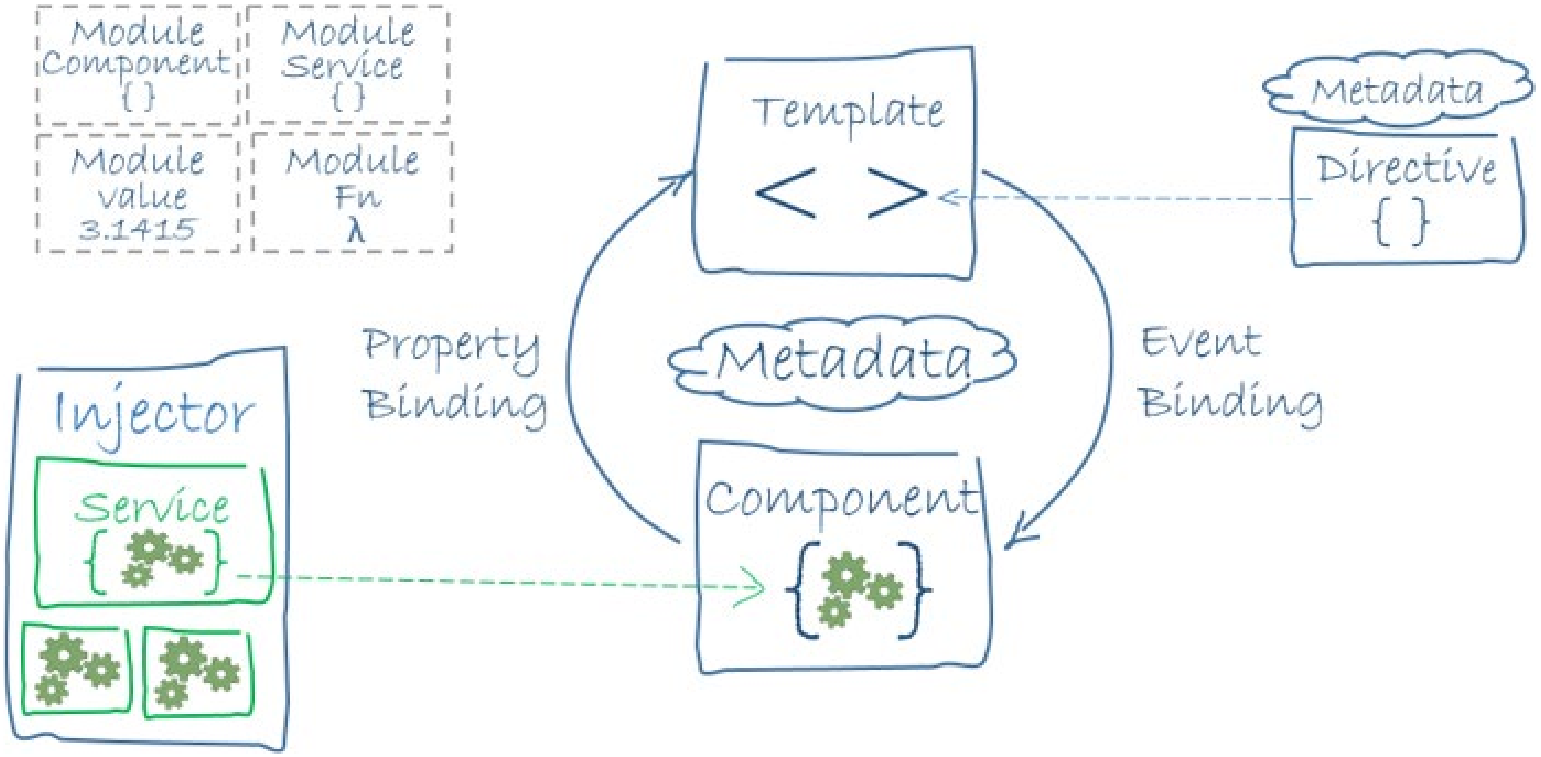
What do you want to do next with your app?

```
ng generate component xyz
```

Love Angular? Give our repo a star.

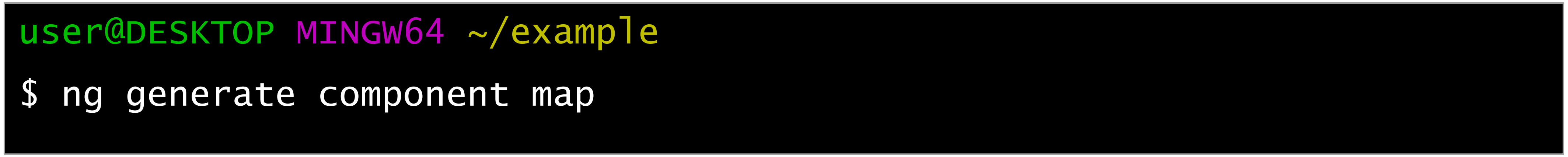
Angular architecture

- Modules
- Components
- Templates
- Metadata
- Data binding
- Directives
- Services

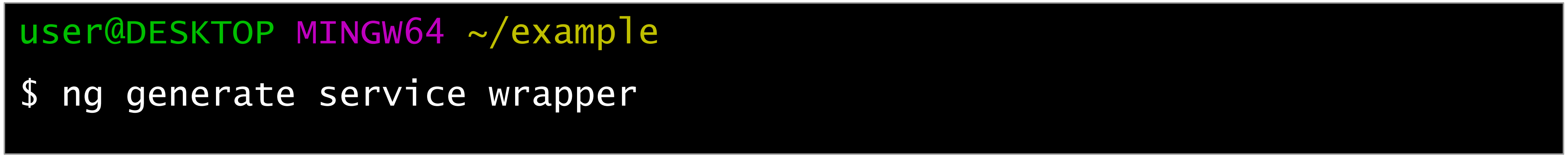


Angular architecture

- Creating components:



```
user@DESKTOP MINGW64 ~/example
$ ng generate component map
```



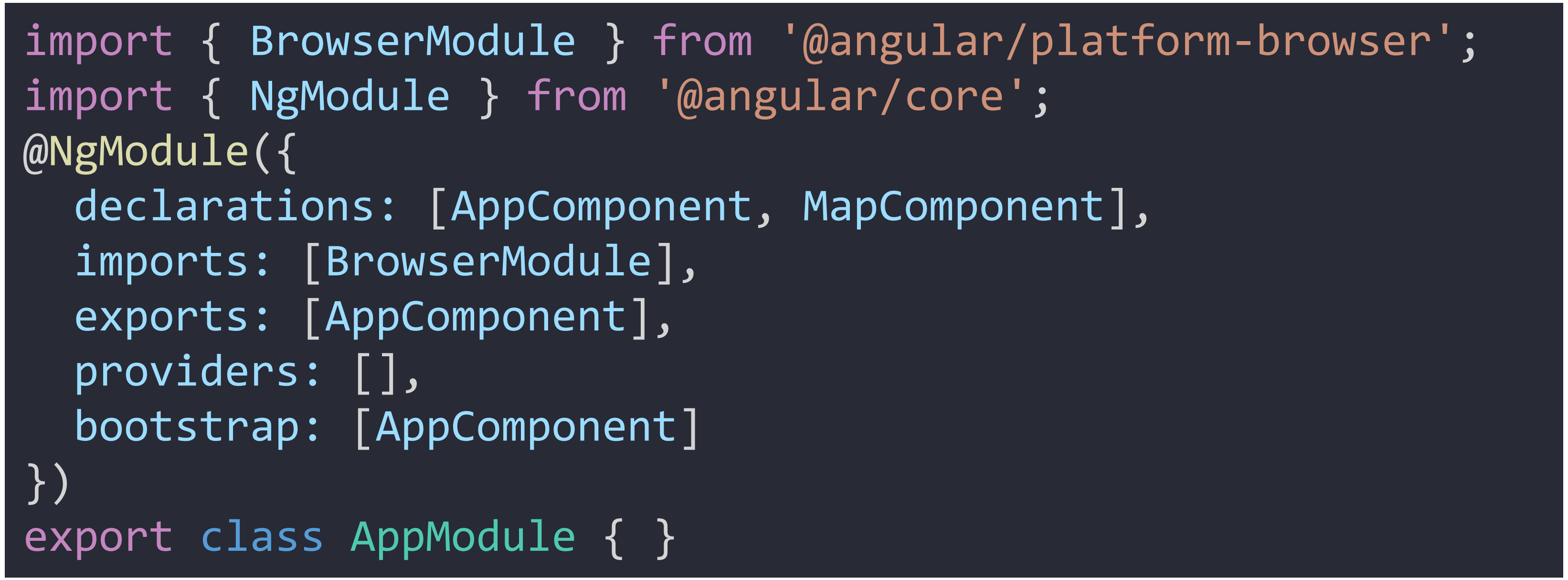
```
user@DESKTOP MINGW64 ~/example
$ ng generate service wrapper
```

Angular architecture: modules

- Angular applications are modular:
    - An application defines a set of modules.
    - Every angular module is a class with `@NgModule` decorator.
- Every angular application has at least one module: root module.
- A module encapsulates a set of components dedicated to an application domain, a workflow, or closely related set of capabilities.
- A module can import functionalities from other modules and allow their own functionalities to be exported.

Angular architecture: modules

- Module properties:
  - Declaration: components, directives, and pipes that belong to the module.
  - Exports: subset of declarations visible and usable by other modules.
  - Imports: external modules.
  - Providers: creators of services.
  - Bootstrap: main application view, the root component.

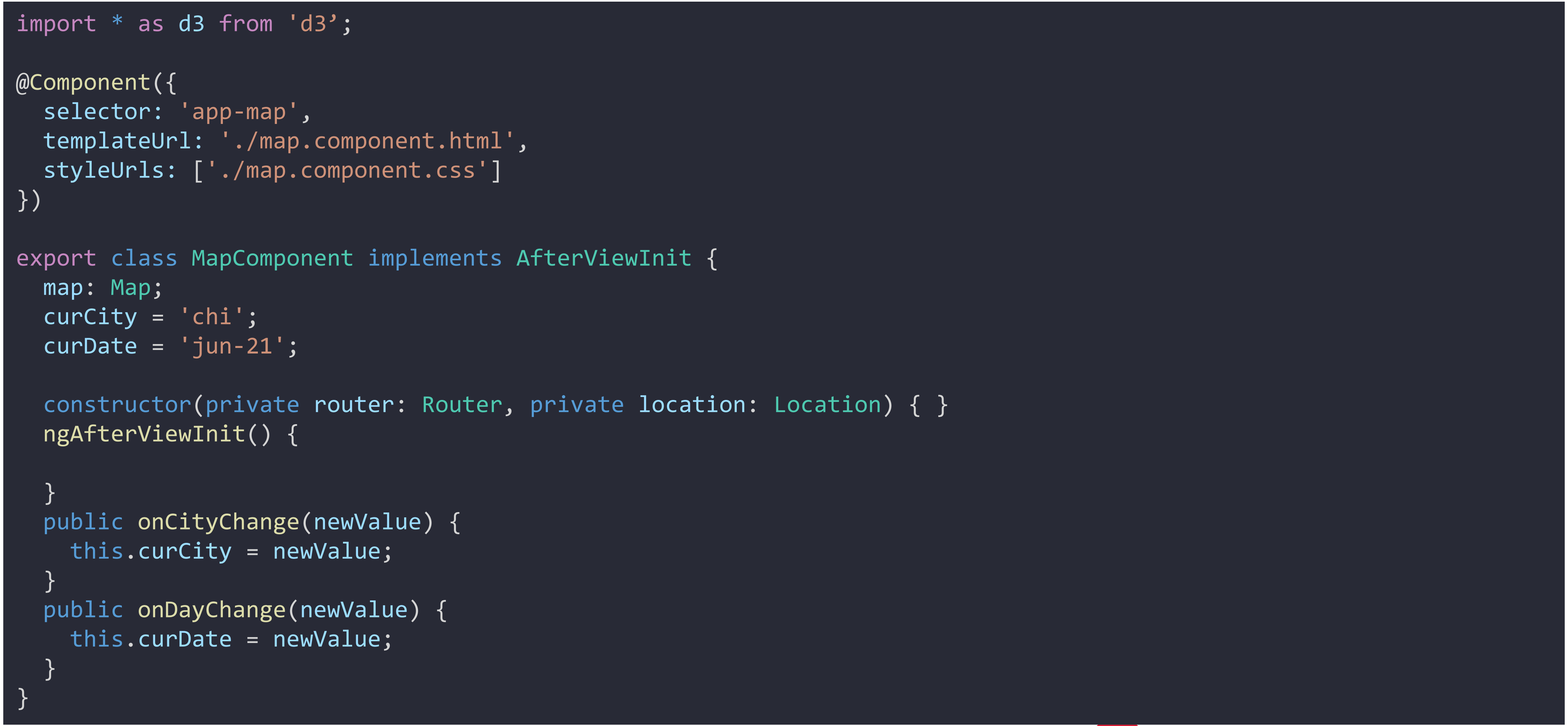


```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
@NgModule({
  declarations: [AppComponent, MapComponent],
  imports: [BrowserModule],
  exports: [AppComponent],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular architecture: components

- Components are the main building blocks of Angular applications.
- Each component consist of:
  - HTML template that declares what renders on the page.
  - TypeScript class that defines behavior.
  - CSS selector that defines how the component is used in a template.
  - CSS styles applied to the template.

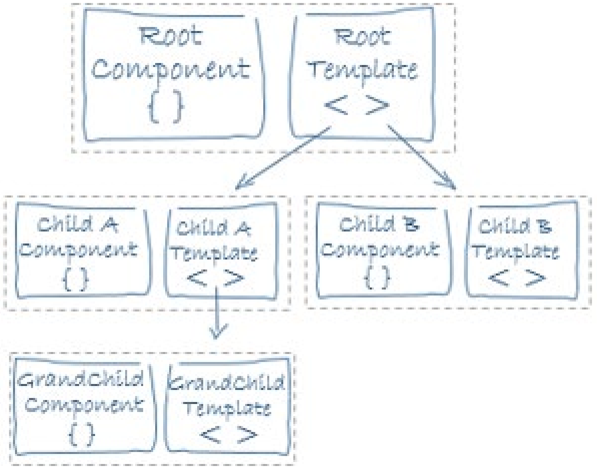
Angular architecture: components



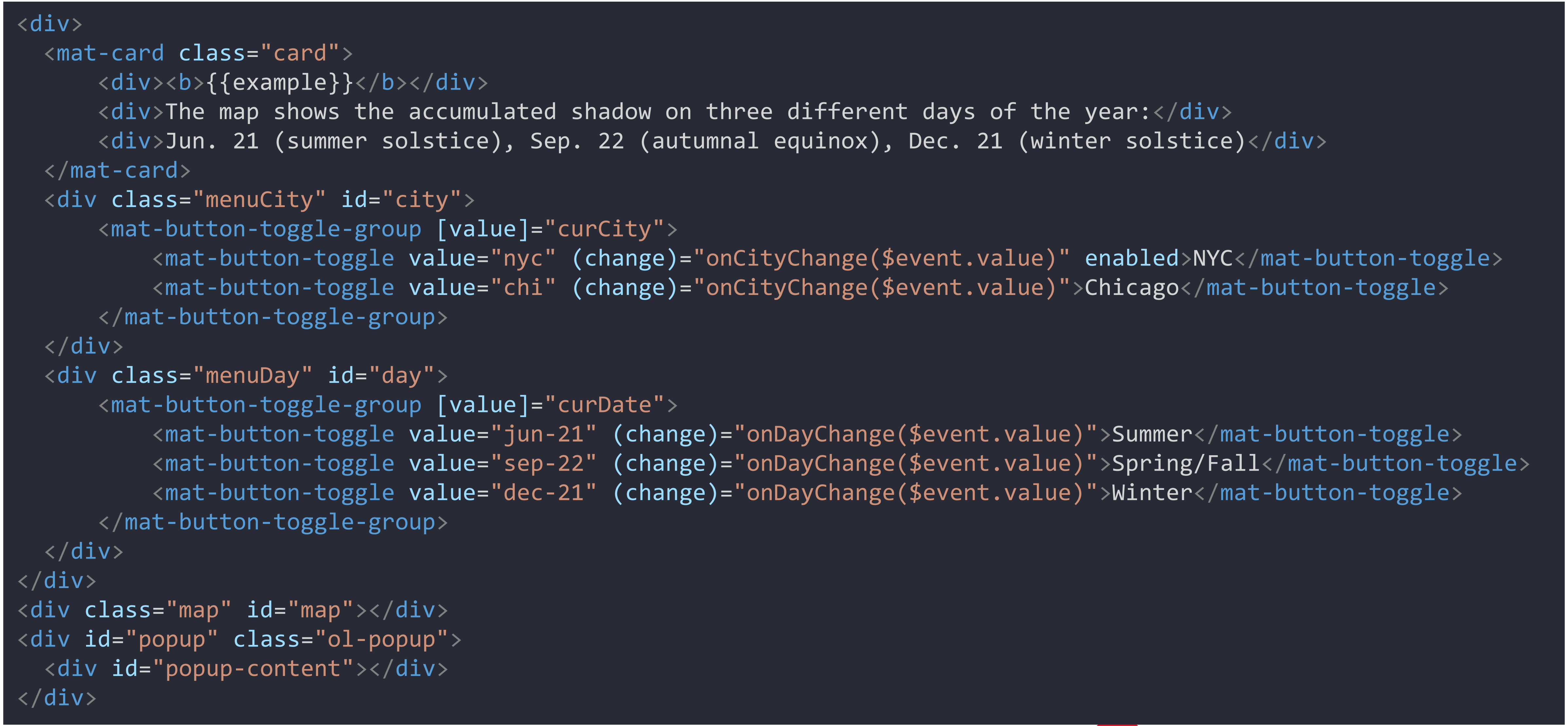
```
import * as d3 from 'd3’;

@Component({
  selector: 'app-map',
  templateUrl: './map.component.html',
  styleUrls: ['./map.component.css']
})

export class MapComponent implements AfterViewInit {
  map: Map;
  curCity = 'chi';
  curDate = 'jun-21';

  constructor(private router: Router, private location: Location) { }
  ngAfterViewInit() {

  }
  public onCityChange(newValue) {
    this.curCity = newValue;
  }
  public onDayChange(newValue) {
    this.curDate = newValue;
  }
}
```

Angular architecture: templates

- A snippet of the HTML code of a component: tells Angular how to render the component.
  - A component's view is defined with its template.
- Uses Angular's template syntax, with custom elements.



Angular architecture: templates

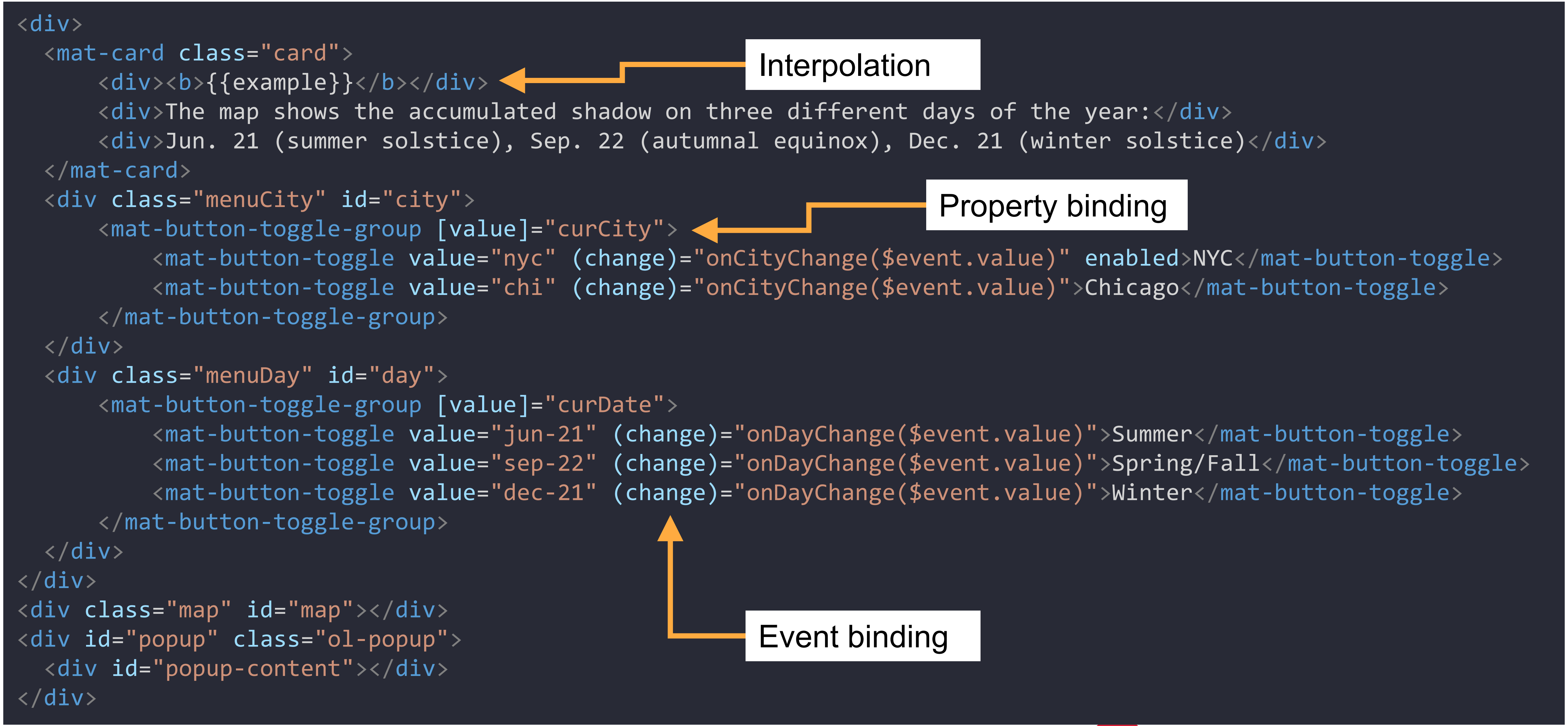


```
<div>
  <mat-card class="card">
      <div><b>{{example}}</b></div>
      <div>The map shows the accumulated shadow on three different days of the year:</div>
      <div>Jun. 21 (summer solstice), Sep. 22 (autumnal equinox), Dec. 21 (winter solstice)</div>
  </mat-card>
  <div class="menuCity" id="city">
      <mat-button-toggle-group [value]="curCity">
          <mat-button-toggle value="nyc" (change)="onCityChange($event.value)" enabled>NYC</mat-button-toggle>
          <mat-button-toggle value="chi" (change)="onCityChange($event.value)">Chicago</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
  <div class="menuDay" id="day">
      <mat-button-toggle-group [value]="curDate">
          <mat-button-toggle value="jun-21" (change)="onDayChange($event.value)">Summer</mat-button-toggle>
          <mat-button-toggle value="sep-22" (change)="onDayChange($event.value)">Spring/Fall</mat-button-toggle>
          <mat-button-toggle value="dec-21" (change)="onDayChange($event.value)">Winter</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
</div>
<div class="map" id="map"></div>
<div id="popup" class="ol-popup">
  <div id="popup-content"></div>
</div>
```

Angular architecture: data binding

- Mechanism for coordinating parts of a template with parts of a component.
- Four main forms:
  - Interpolation: `{{example}}`
    - Incorporate dynamic string values into HTML templates.
  - Property binding: `[example]`
    - Set values for properties of HTML elements.
  - Event binding: `(click)`
    - Listen for and respond to user actions (keystrokes, mouse movements, clicks, touches).
  - Two-way data binding: `[(ngModel)]`
    - Gives components in application a way to share data, using two-way binding to listen for events and update values simultaneously.

Angular architecture: data binding



```
<div>
  <mat-card class="card">
      <div><b>{{example}}</b></div>
      <div>The map shows the accumulated shadow on three different days of the year:</div>
      <div>Jun. 21 (summer solstice), Sep. 22 (autumnal equinox), Dec. 21 (winter solstice)</div>
  </mat-card>
  <div class="menuCity" id="city">
      <mat-button-toggle-group [value]="curCity">
          <mat-button-toggle value="nyc" (change)="onCityChange($event.value)" enabled>NYC</mat-button-toggle>
          <mat-button-toggle value="chi" (change)="onCityChange($event.value)">Chicago</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
  <div class="menuDay" id="day">
      <mat-button-toggle-group [value]="curDate">
          <mat-button-toggle value="jun-21" (change)="onDayChange($event.value)">Summer</mat-button-toggle>
          <mat-button-toggle value="sep-22" (change)="onDayChange($event.value)">Spring/Fall</mat-button-toggle>
          <mat-button-toggle value="dec-21" (change)="onDayChange($event.value)">Winter</mat-button-toggle>
      </mat-button-toggle-group>
  </div>
</div>
<div class="map" id="map"></div>
<div id="popup" class="ol-popup">
  <div id="popup-content"></div>
</div>
```

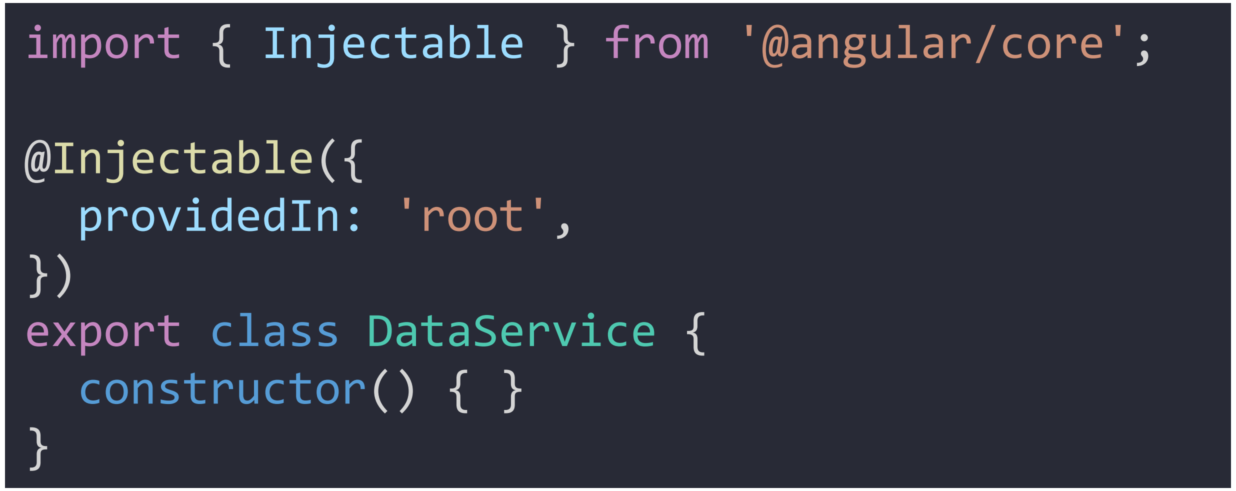
Interpolation

Property binding

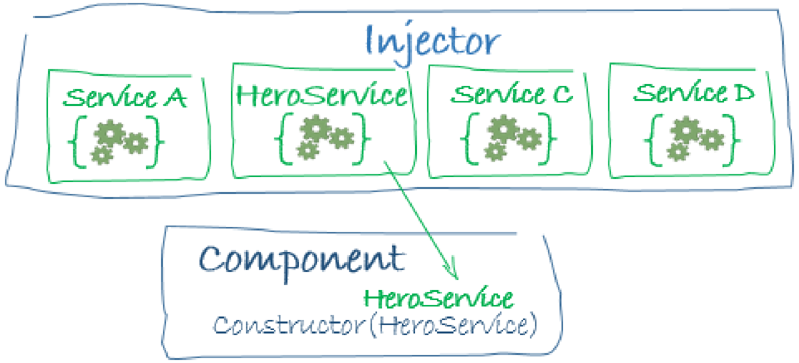
Event binding

Angular architecture: services

- Components shouldn’t fetch or save data directly.
- Components should focus on presenting data, and delegate data access to a service.
- Dependency injection: provide components with services they need.



```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class DataService {
  constructor() { }
}
```



Back-end

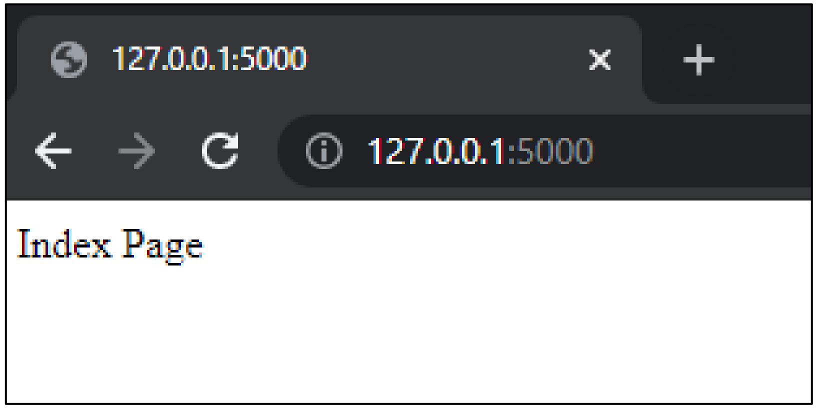
Flask

- Python framework for developing web applications.
- Lightweight applications (when compared to Django).
- Easy integration between front-end and back-end components.

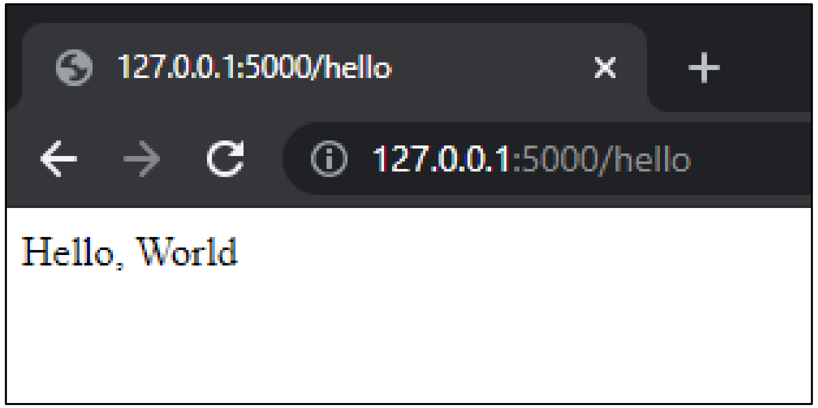
Flask: minimal application

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

```
user@DESKTOP MINGW64 ~/
$ export FLASK_APP=example FLASK_ENV=development
$ flask run
```



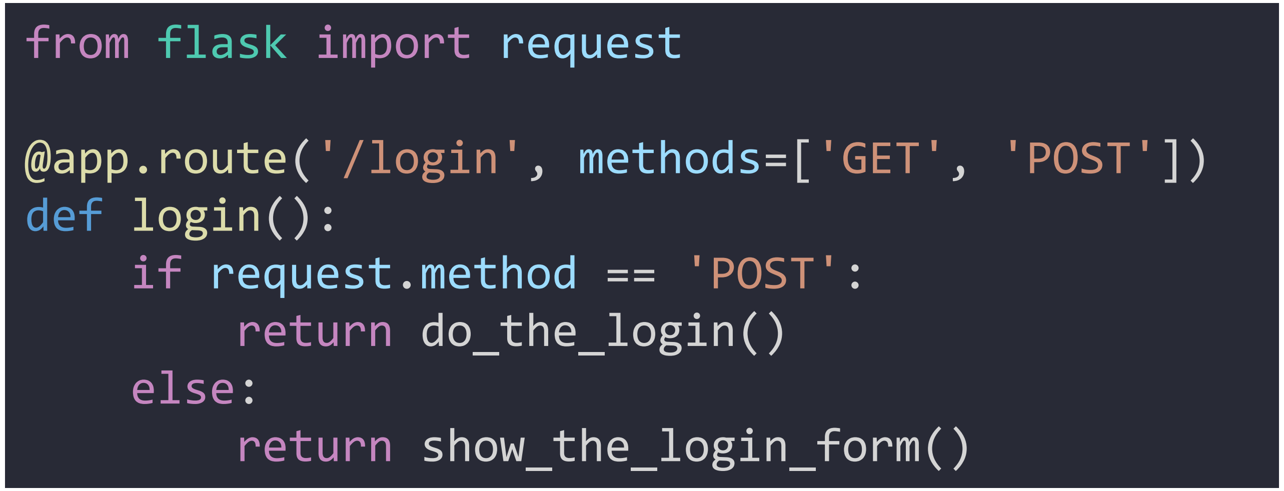
Index Page



Hello, World

Flask: minimal application

- Web applications use different HTTP methods when accessing URLs.
- You can use the `methods` argument to handle different HTTP methods.



```python
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

HTTP request methods

- HTTP is designed to enable communication between clients and servers.
- HTTP works as a request-response protocol between a client and a server.
- HTTP methods:
  - **GET**
  - **POST**
  - PUT
  - HEAD
    DELETE
  - PATCH
  - OPTIONS

HTTP request methods

- GET:
  - Used to request data from a specified resource.
  - One of the most common HTTP methods.

```
/test?name1=value1&name2=value2
```

- POST:
  - Used to send data to a server.
  - Data sent to the server with POST is stored in the **request body** of the HTTP request.

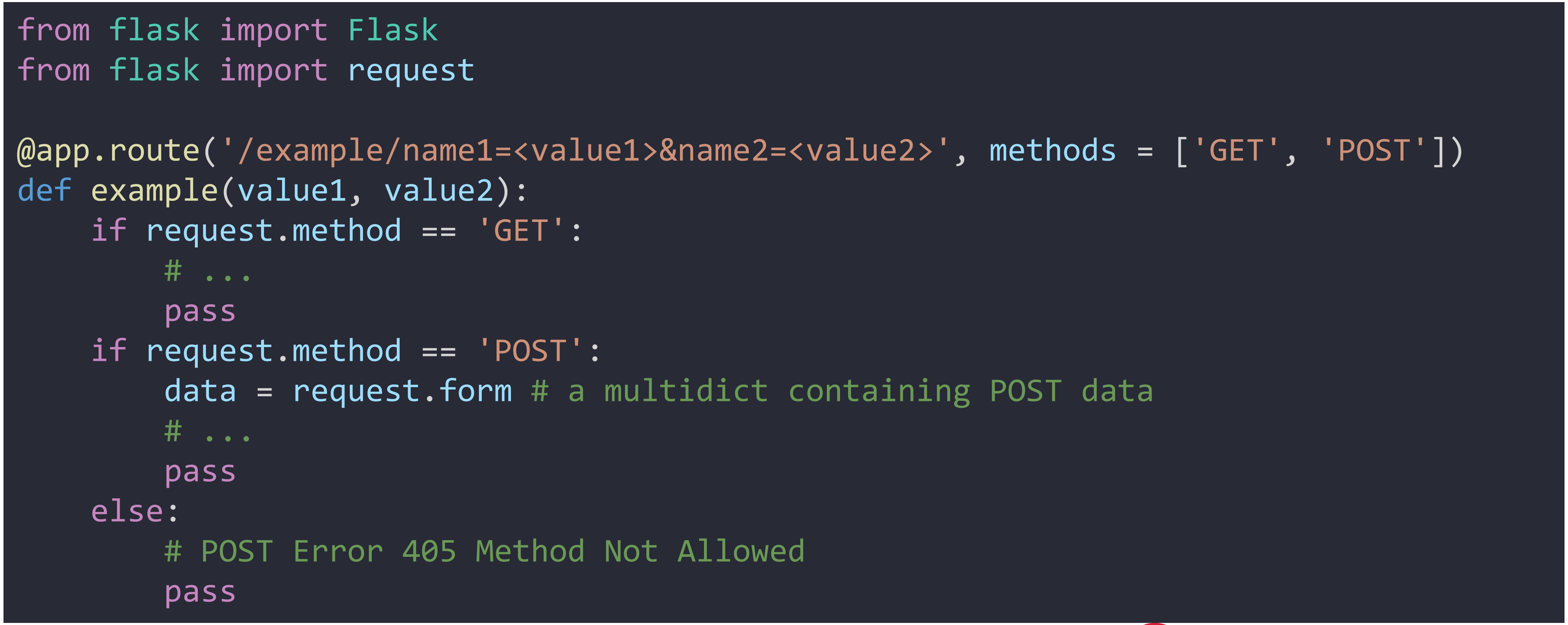
```
POST /test HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

HTTP request methods

	GET	POST
Back button / Reload	Harmless	Data will be re-submitted
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
History	**Parameters remain in browser history**	**Parameters are not saved in browser history**
Restrictions on data length	**Length of a URL is limited: 2048 characters**	**No restrictions**
Restrictions on data type	**Only ASCII characters**	**No restrictions. Binary data is also allowed**
Security	Less secure, data sent is part of the URL	Safer, parameters are not stored in browser history
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

From: https://www.w3schools.com/tags/ref_httpmethods.asp

Flask and HTTP methods



```
from flask import Flask
from flask import request

@app.route('/example/name1=<value1>&name2=<value2>', methods = ['GET', 'POST'])
def example(value1, value2):
    if request.method == 'GET':
        # ...
        pass
    if request.method == 'POST':
        data = request.form # a multidict containing POST data
        # ...
        pass
    else:
        # POST Error 405 Method Not Allowed
        pass
```

Mongoose

- Networking library for C/C++.
- Event-driven non-blocking APIs for TCP, UDP, HTTP, …
- Easy to integrate: mongoose.c and mongoose.h, that is it.

Mongoose: minimal application

- Declare and initialize an event manager:

```
struct mg_mgr mgr;
mg_mgr_init(&mgr);
```

- Create connections with an event handler:

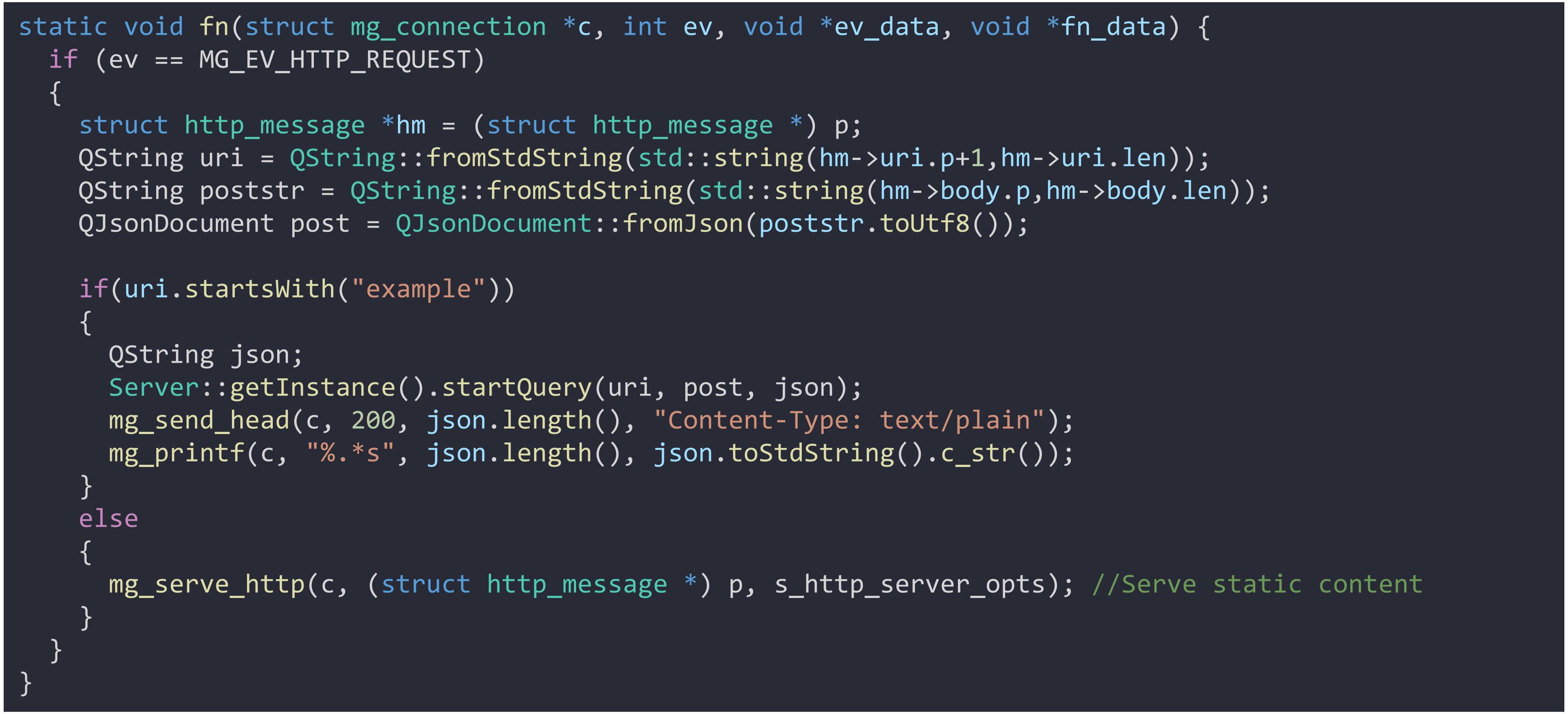
```
struct mg_connection *c = mg_http_listen(&mgr, "0.0.0.0:8000", fn, arg);
```

- Create an event loop:

```
for (;;) {
  mg_mgr_poll(&mgr, 1000);
}
```

Mongoose: minimal application

- Event handler function defines connection’s behavior



```
static void fn(struct mg_connection *c, int ev, void *ev_data, void *fn_data) {
  if (ev == MG_EV_HTTP_REQUEST)
  {
    struct http_message *hm = (struct http_message *) p;
    QString uri = QString::fromStdString(std::string(hm->uri.p+1,hm->uri.len));
    QString poststr = QString::fromStdString(std::string(hm->body.p,hm->body.len));
    QJsonDocument post = QJsonDocument::fromJson(poststr.toUtf8());

    if(uri.startsWith("example"))
    {
      QString json;
      Server::getInstance().startQuery(uri, post, json);
      mg_send_head(c, 200, json.length(), "Content-Type: text/plain");
      mg_printf(c, "%.*s", json.length(), json.toStdString().c_str());
    }
    else
    {
      mg_serve_http(c, (struct http_message *) p, s_http_server_opts); //Serve static content
    }
  }
}
```

Back-end building blocks

- Boost
- QT
- CUDA

Boost

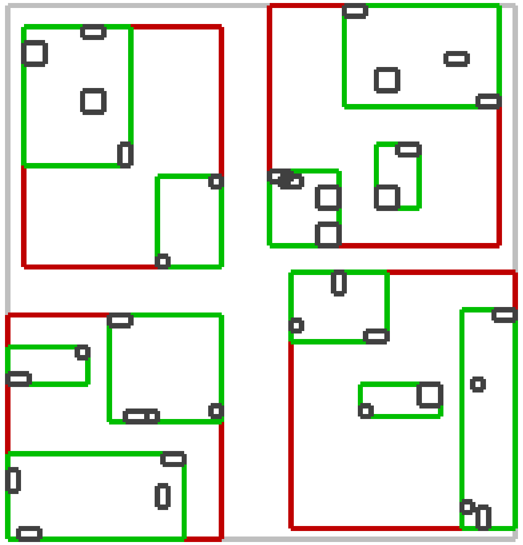
- Libraries for C++ that provide support for linear algebra, multithreading, image processing, etc.
- The most used C++ library (apart of the STL library).
- Supported in most operating systems.
- Integration with other programming languages:
  - Python
  - Java

Boost

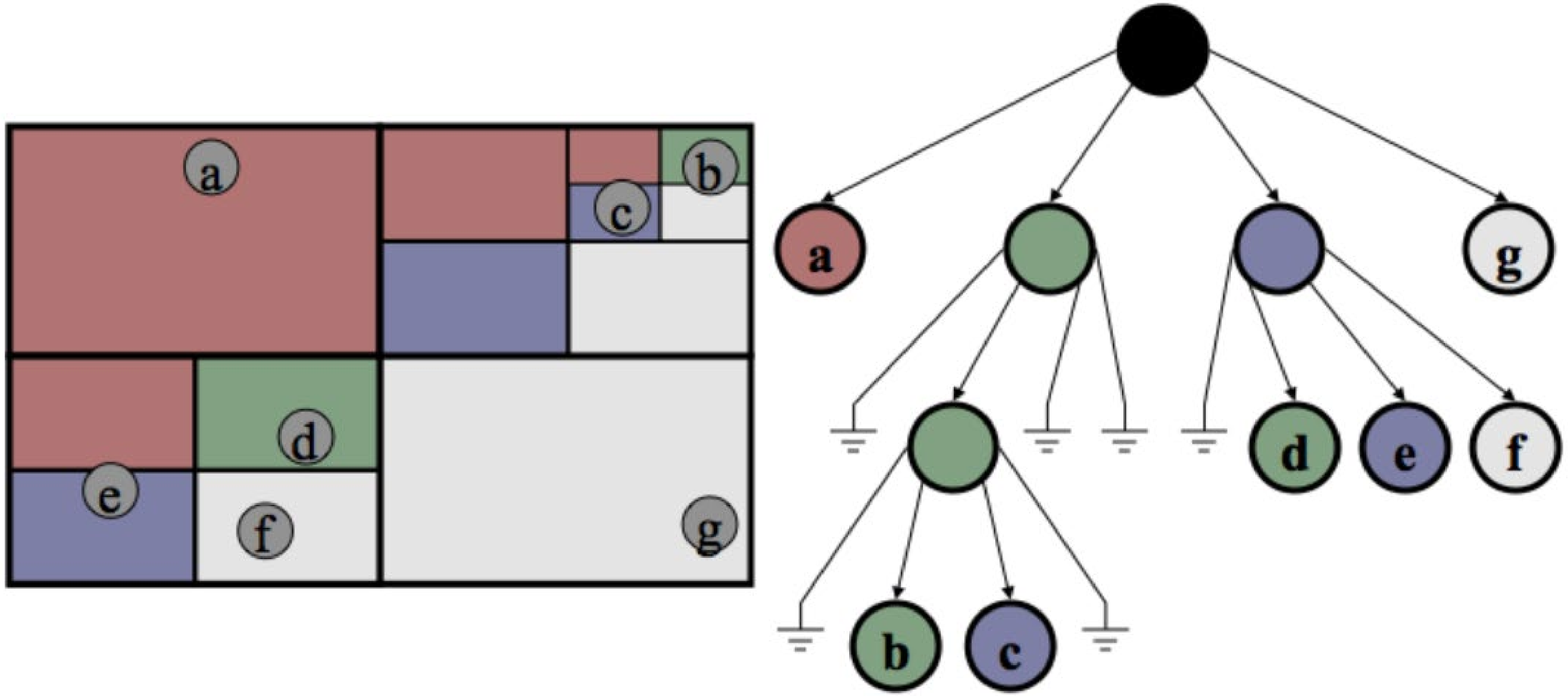
- Example of boost libraries:
  - Algorithms
  - Concurrent programming
  - Containers
  - Data structures
  - Image processing
  - Threads
  - String and text processing
  - Iterators
  - Streams
  - Parsing
  - Memory management
  - …

Boost: spatial indices

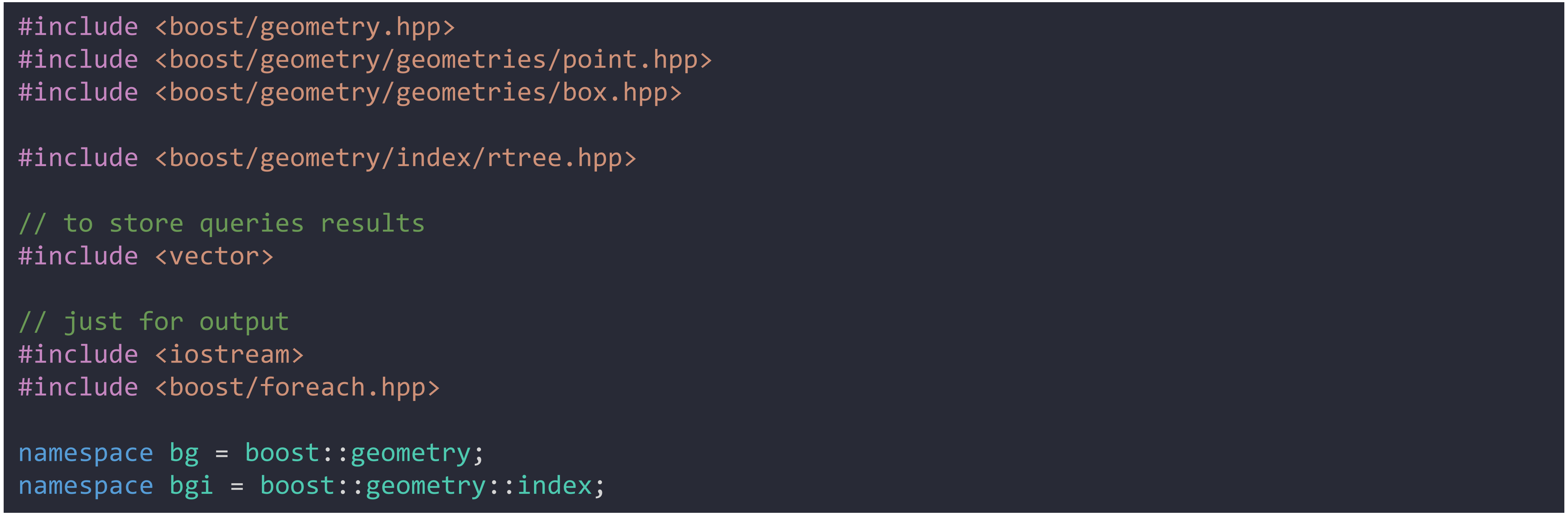
- Boost.Geometry.Index collects data structures for spatial indexing of data.
- Goal: accelerate searching for objects in space.
- R-tree is a self-balanced data structure for spatial access methods.
    - Indexes multi-dimensional information (points, rectangles, polygons).
    - Group nearby objects and represent them with their minimum bounding rectangle.



Quadtree

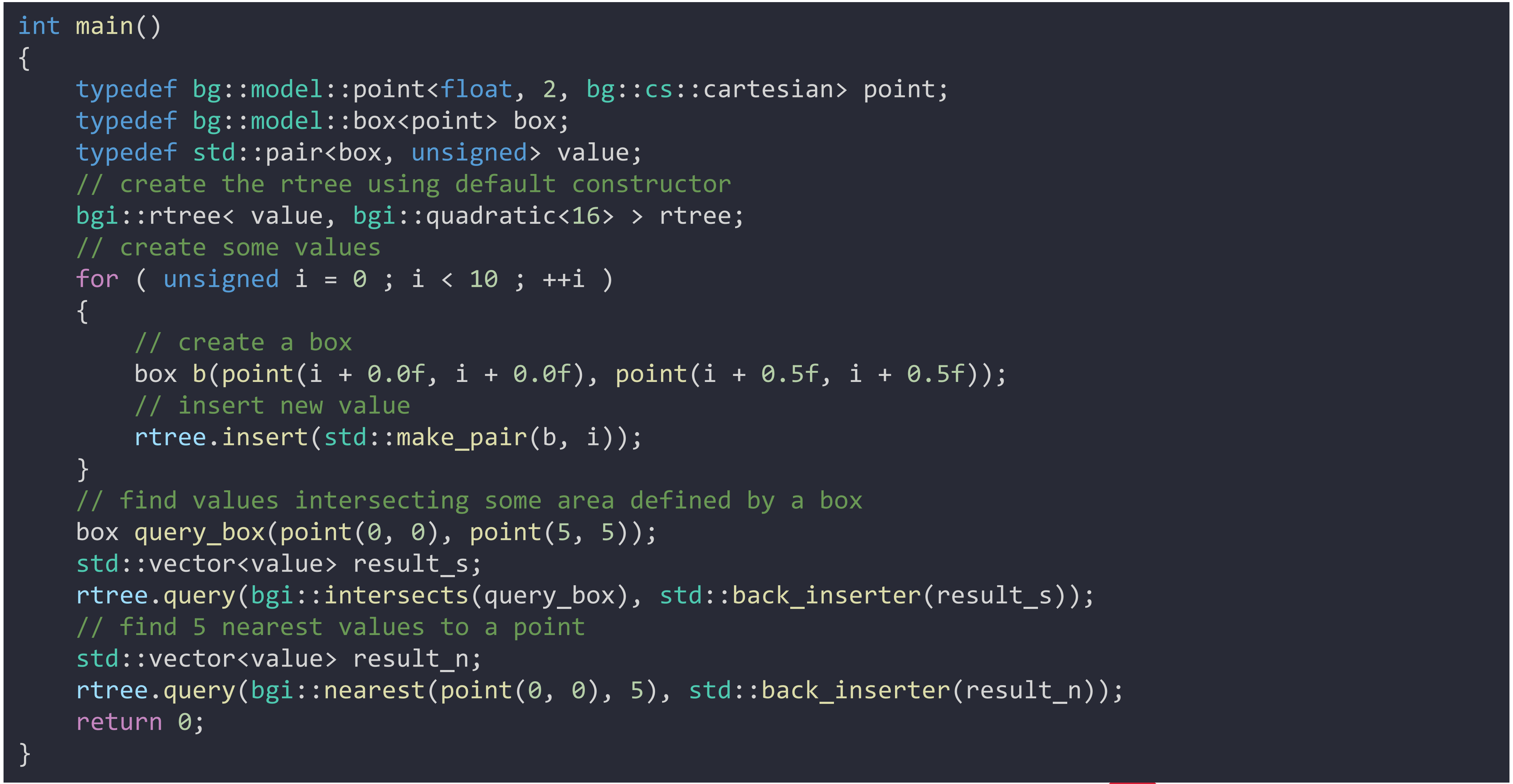


Boost: r-tree example



```cpp
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/point.hpp>
#include <boost/geometry/geometries/box.hpp>

#include <boost/geometry/index/rtree.hpp>

// to store queries results
#include <vector>

// just for output
#include <iostream>
#include <boost/foreach.hpp>

namespace bg = boost::geometry;
namespace bgi = boost::geometry::index;
```

Boost: r-tree example



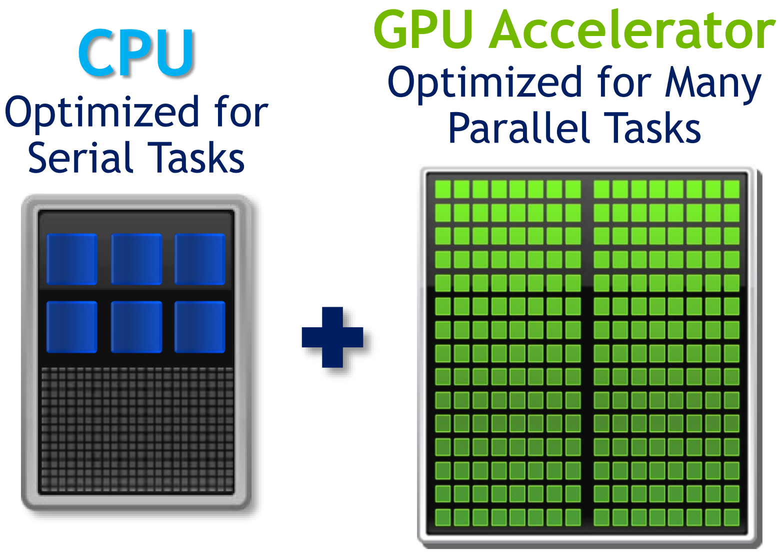
```
int main()
{
    typedef bg::model::point<float, 2, bg::cs::cartesian> point;
    typedef bg::model::box<point> box;
    typedef std::pair<box, unsigned> value;
    // create the rtree using default constructor
    bgi::rtree< value, bgi::quadratic<16> > rtree;
    // create some values
    for ( unsigned i = 0 ; i < 10 ; ++i )
    {
        // create a box
        box b(point(i + 0.0f, i + 0.0f), point(i + 0.5f, i + 0.5f));
        // insert new value
        rtree.insert(std::make_pair(b, i));
    }
    // find values intersecting some area defined by a box
    box query_box(point(0, 0), point(5, 5));
    std::vector<value> result_s;
    rtree.query(bgi::intersects(query_box), std::back_inserter(result_s));
    // find 5 nearest values to a point
    std::vector<value> result_n;
    rtree.query(bgi::nearest(point(0, 0), 5), std::back_inserter(result_n));
    return 0;
}
```

CUDA

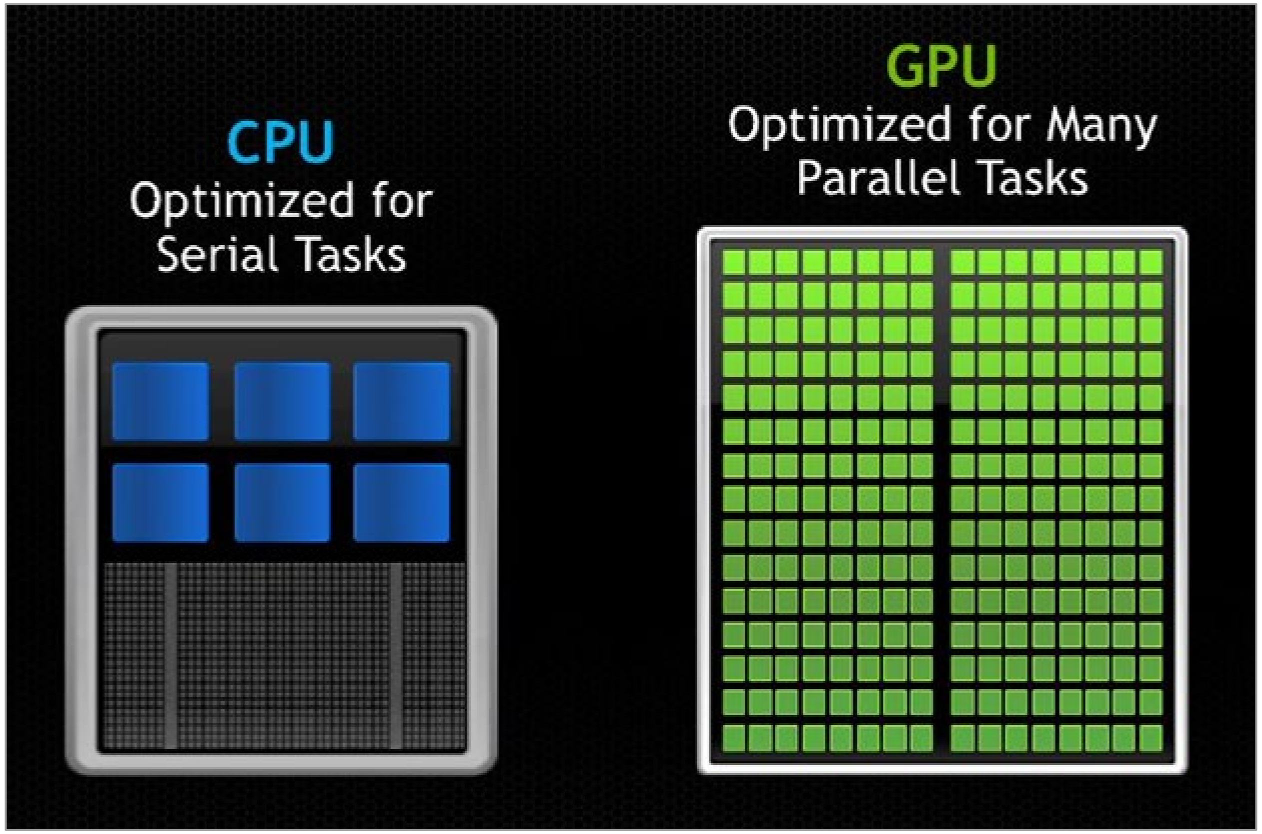
- Parallel computing platform and API that uses the GPU for general purpose computing.
- <u>Software</u> layer that gives direct access to the GPU's parallel computational elements.
- Design to work with other programming languages, such as C, C++, Fortran.

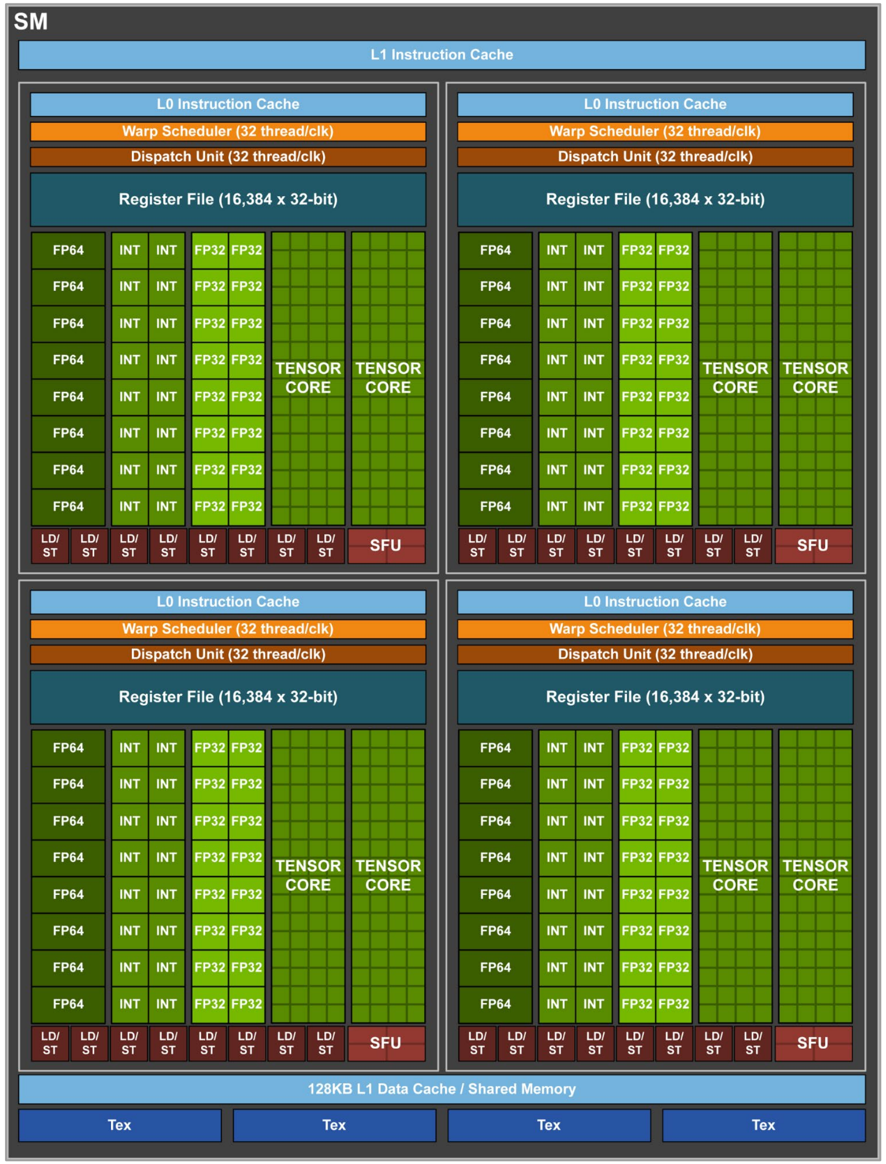
CUDA

- GPUs are designed to perform high-speed parallel calculations for real-time rendering (embarrassingly parallel task).
- 10-100x speed-ups over CPUs when applied in GPGPU.
- Why?
  - CPU contains few powerful cores, GPU contains hundreds of smaller cores.
  - CPU: individual threads execute instructions independently (SISD). GPU: single instruction, multiple threads (SIMT).
  - Shared memory for algorithms with a high degree of locality.

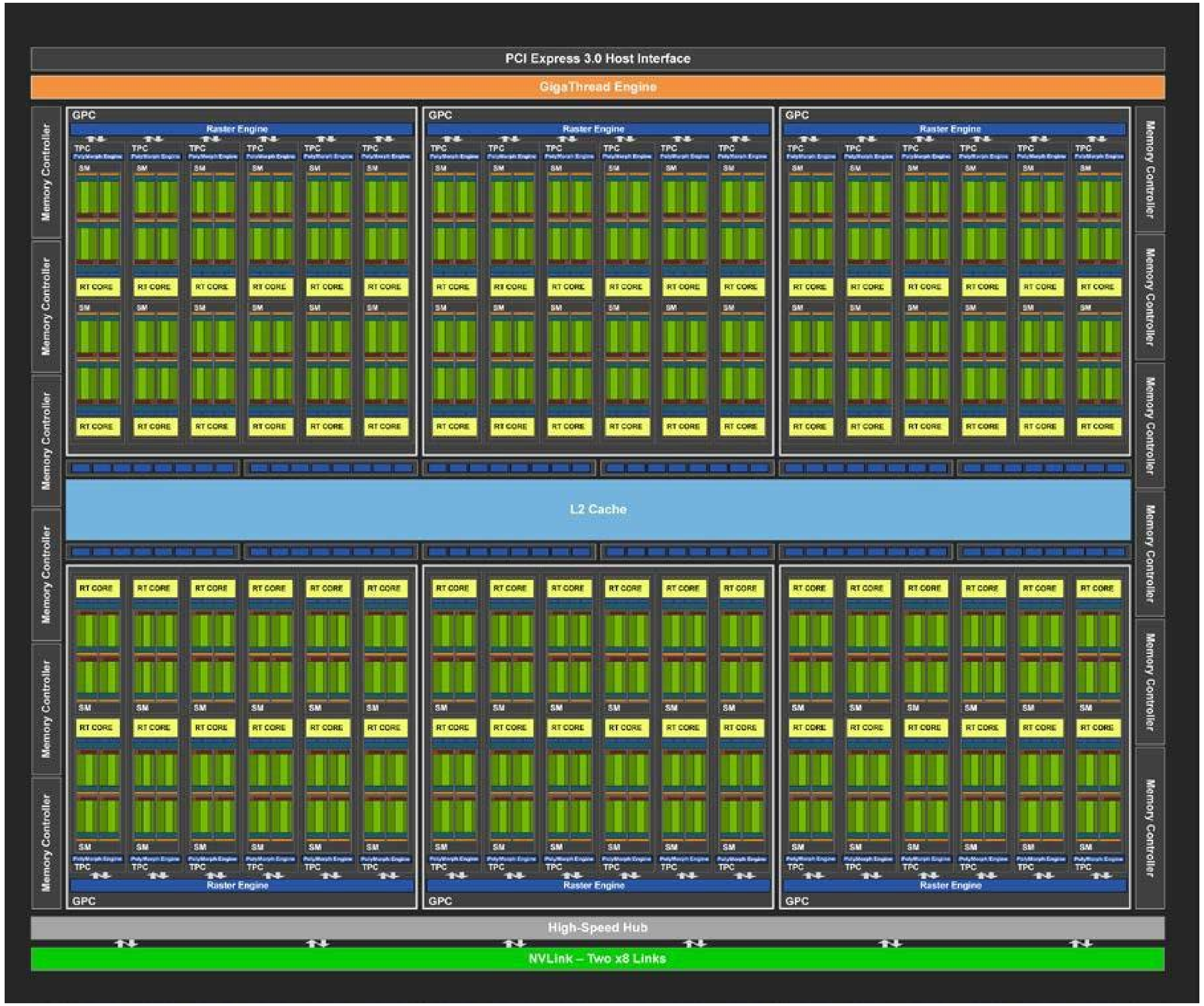


Modern GPUs


CPU
Optimized for Serial Tasks
GPU
Optimized for Many Parallel Tasks

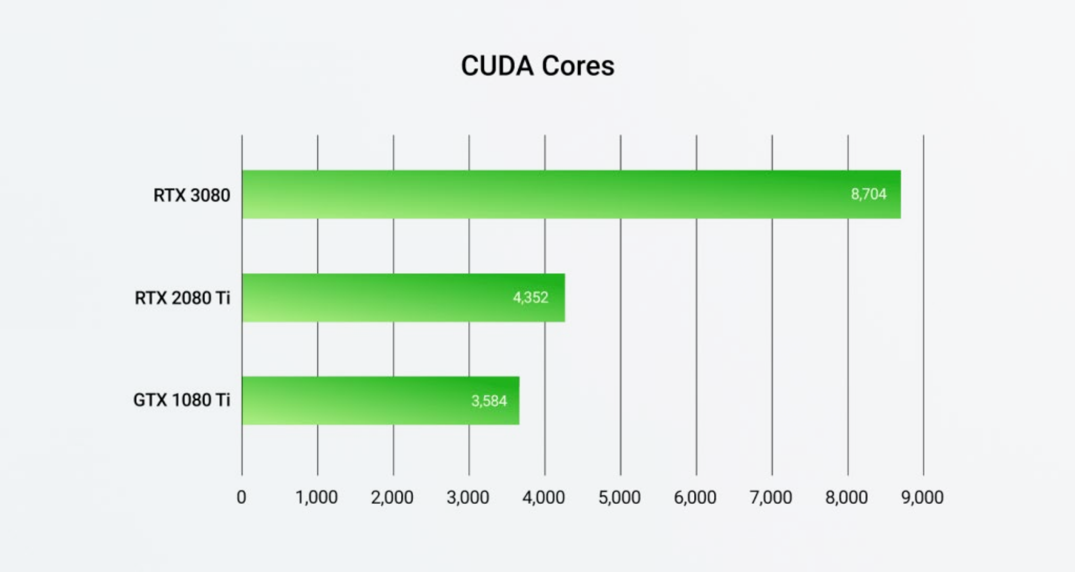

SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

NVIDIA Titan RTX



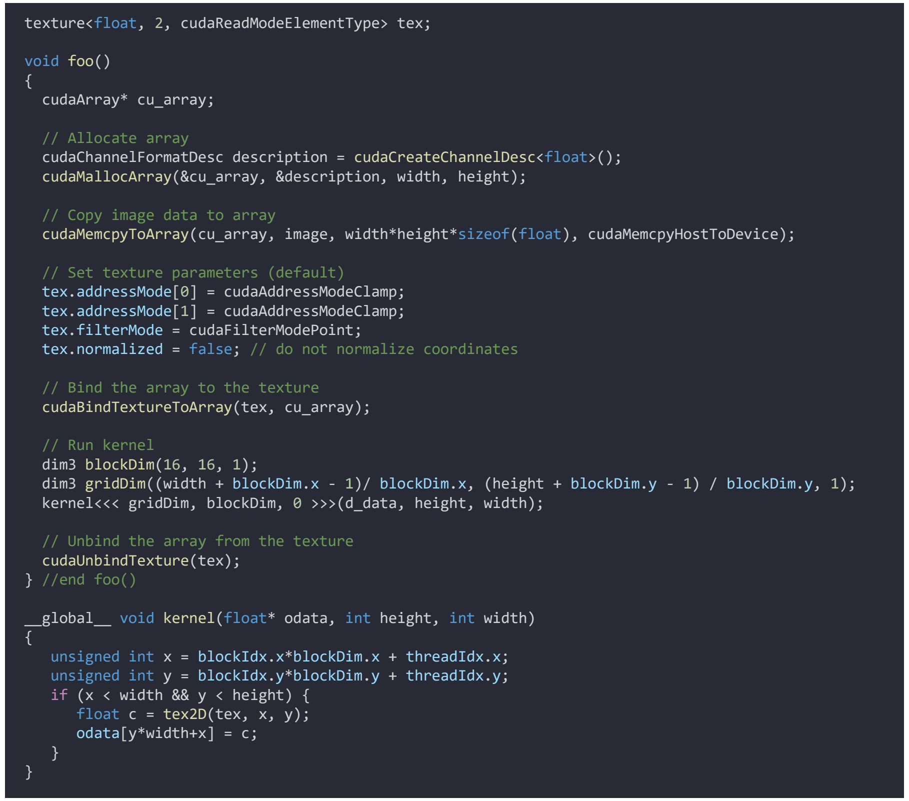
GPU architecture

- Global memory:
  - Similar to CPU’s RAM.
  - Accessible by both CPU and GPU.
  - Limited: < 24 GB
- Streaming multiprocessors (SMs)
  - Perform the actual computations.
  - Each SM has its own control units, registers, caches, **execution pipeline**.
  - 3080 RTX: 68 SMs, each with 128 CUDA cores.



Heterogeneous computing

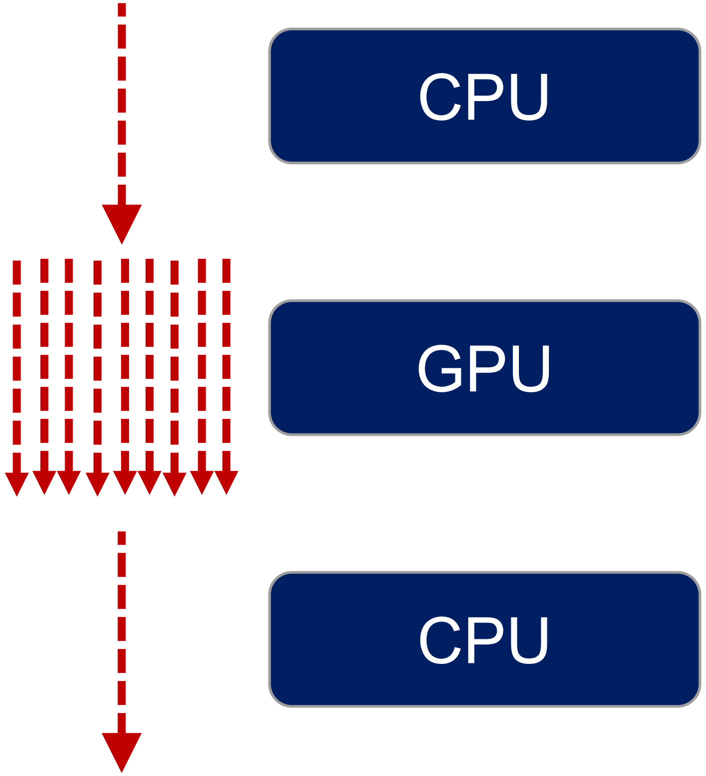
- Host: CPU and its memory.
- Device: GPU and its memory.



```
texture<float, 2, cudaReadModeElementType> tex;

void foo()
{
  cudaArray* cu_array;

  // Allocate array
  cudaChannelFormatDesc description = cudaCreateChannelDesc<float>();
  cudaMallocArray(&cu_array, &description, width, height);

  // Copy image data to array
  cudaMemcpyToArray(cu_array, image, width*height*sizeof(float), cudaMemcpyHostToDevice);

  // Set texture parameters (default)
  tex.addressMode[0] = cudaAddressModeClamp;
  tex.addressMode[1] = cudaAddressModeClamp;
  tex.filterMode = cudaFilterModePoint;
  tex.normalized = false; // do not normalize coordinates

  // Bind the array to the texture
  cudaBindTextureToArray(tex, cu_array);

  // Run kernel
  dim3 blockDim(16, 16, 1);
  dim3 gridDim((width + blockDim.x - 1)/ blockDim.x, (height + blockDim.y - 1) / blockDim.y, 1);
  kernel<<< gridDim, blockDim, 0 >>>(d_data, height, width);

  // Unbind the array from the texture
  cudaUnbindTexture(tex);
} //end foo()

__global__ void kernel(float* odata, int height, int width)
{
   unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
   unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
   if (x < width && y < height) {
      float c = tex2D(tex, x, y);
      odata[y*width+x] = c;
   }
}
```

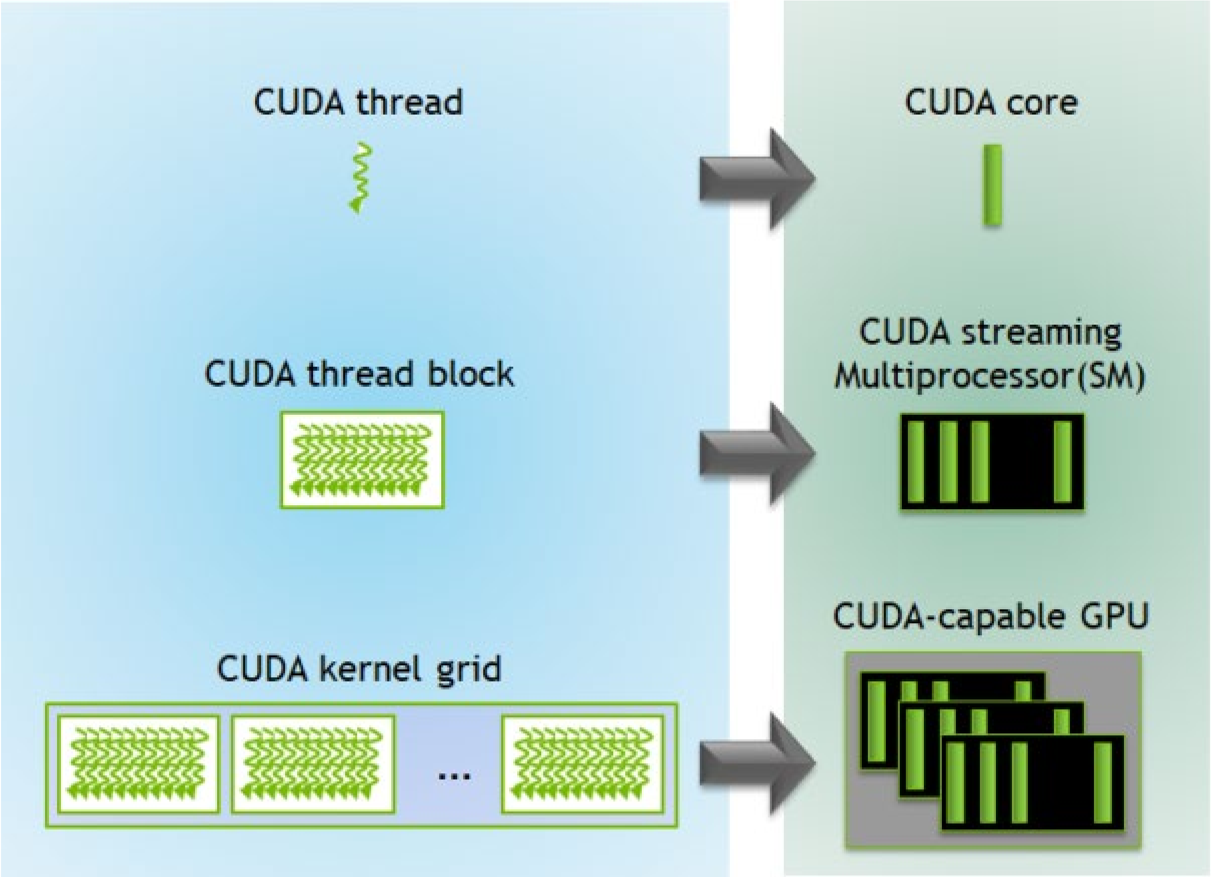
CPU

GPU

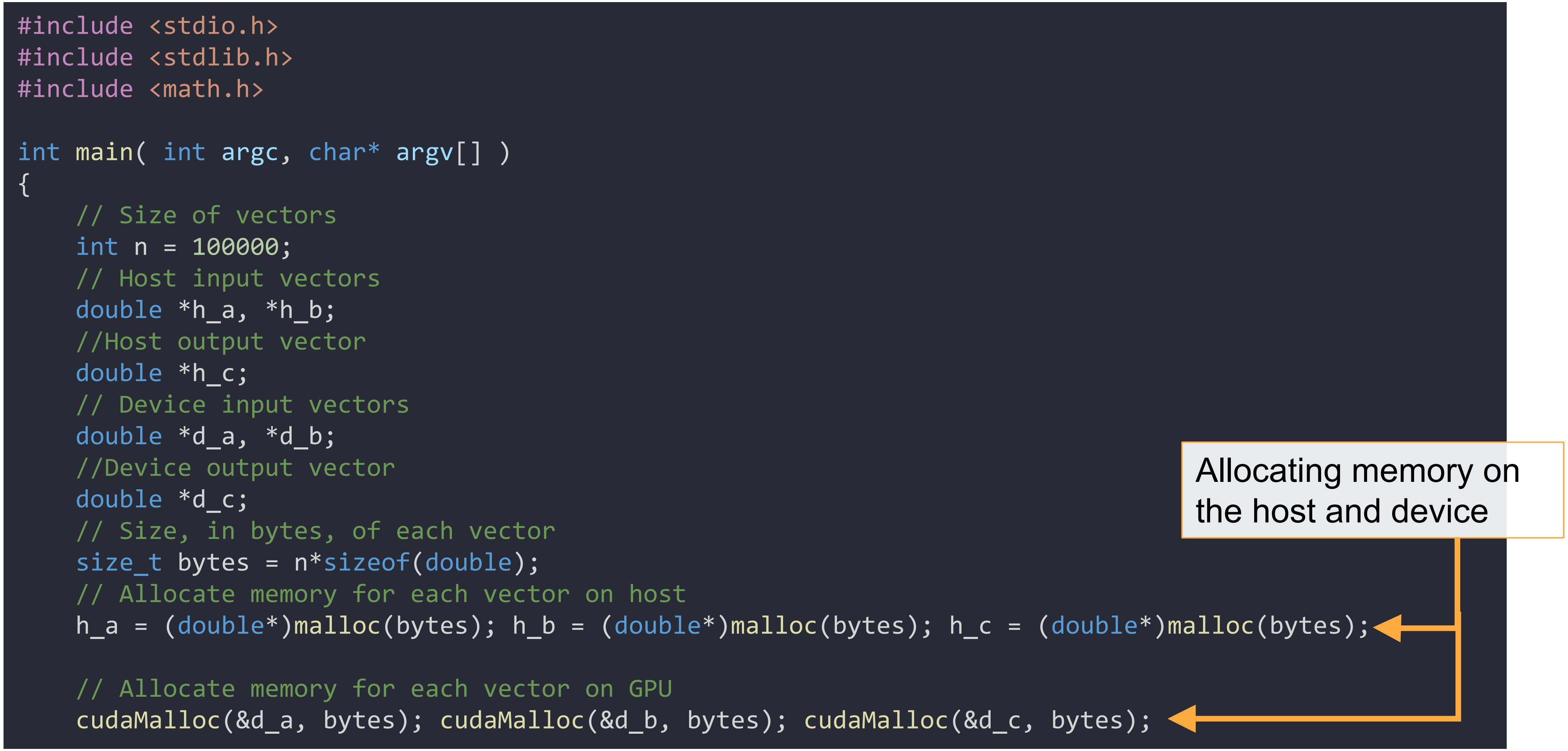


Processing flow

1. Copy input data from CPU to GPU memory.
2. Load GPU program and execute.
   - Group of threads is called a CUDA block, executed by one streaming multiprocessor (SM).
   - Set of blocks is referred to as a grid.
3. Copy results from GPU memory to CPU memory.



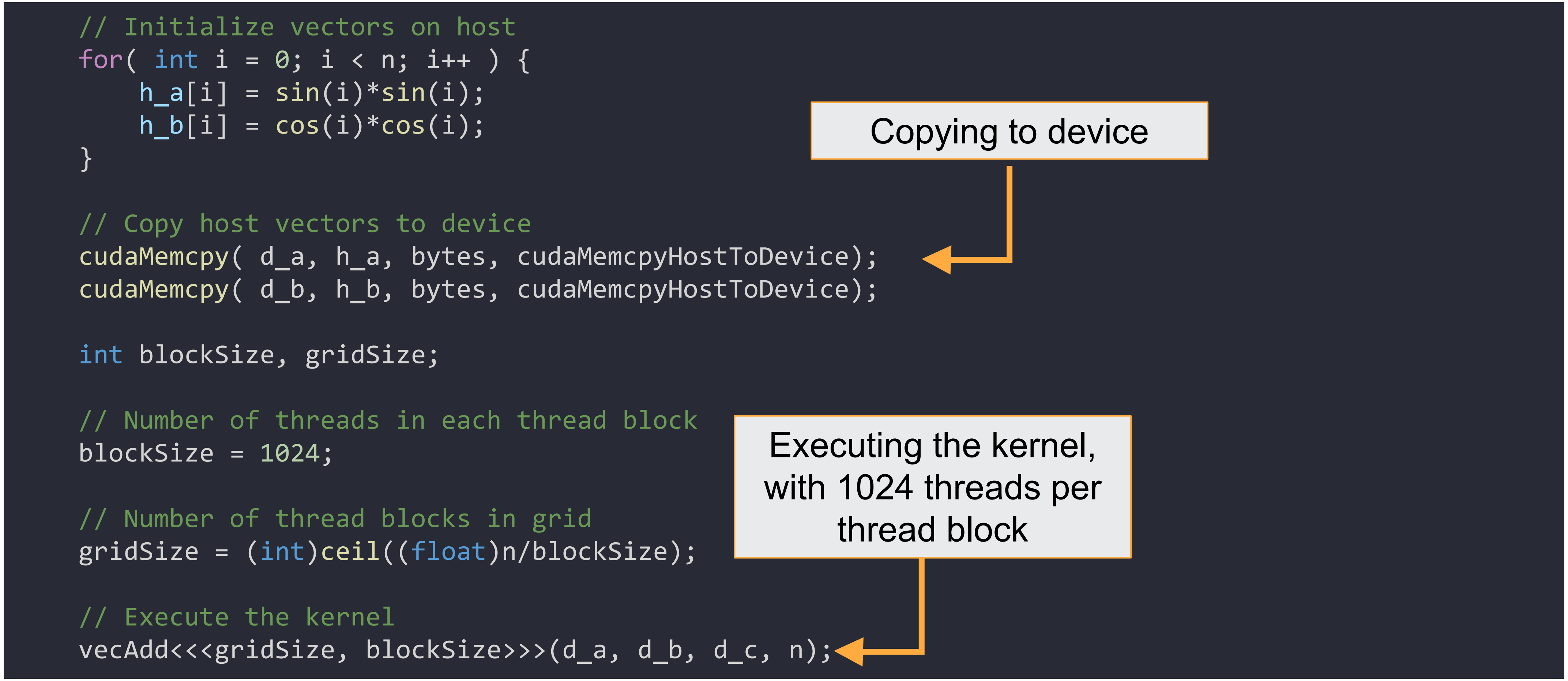
CUDA: vector addition



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main( int argc, char* argv[] )
{
    // Size of vectors
    int n = 100000;
    // Host input vectors
    double *h_a, *h_b;
    //Host output vector
    double *h_c;
    // Device input vectors
    double *d_a, *d_b;
    //Device output vector
    double *d_c;
    // Size, in bytes, of each vector
    size_t bytes = n*sizeof(double);
    // Allocate memory for each vector on host
    h_a = (double*)malloc(bytes); h_b = (double*)malloc(bytes); h_c = (double*)malloc(bytes);

    // Allocate memory for each vector on GPU
    cudaMalloc(&d_a, bytes); cudaMalloc(&d_b, bytes); cudaMalloc(&d_c, bytes);
```

Allocating memory on the host and device

CUDA: vector addition

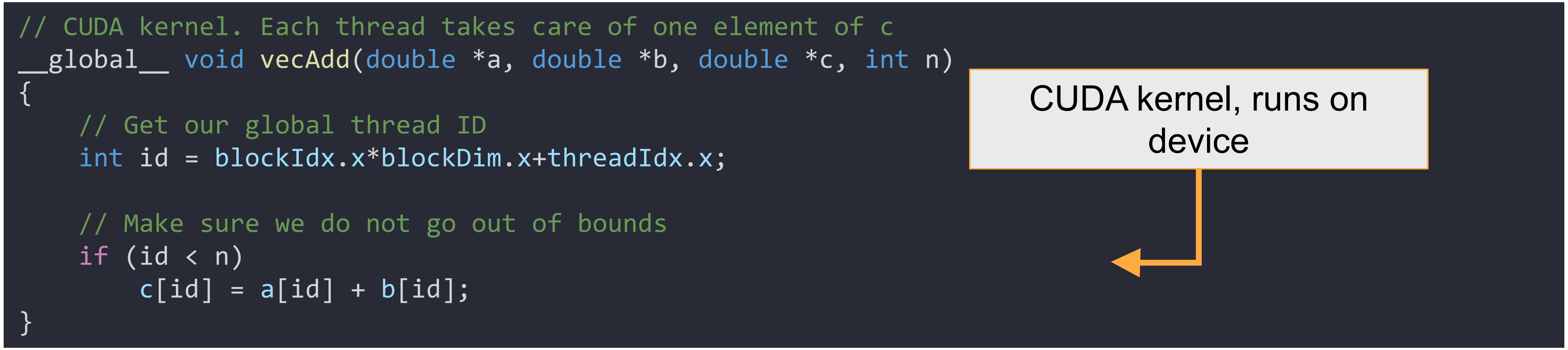


```
// Initialize vectors on host
for( int i = 0; i < n; i++ ) {
    h_a[i] = sin(i)*sin(i);
    h_b[i] = cos(i)*cos(i);
}

// Copy host vectors to device
cudaMemcpy( d_a, h_a, bytes, cudaMemcpyHostToDevice);
cudaMemcpy( d_b, h_b, bytes, cudaMemcpyHostToDevice);

int blockSize, gridSize;

// Number of threads in each thread block
blockSize = 1024;

// Number of thread blocks in grid
gridSize = (int)ceil((float)n/blockSize);

// Execute the kernel
vecAdd<<<gridSize, blockSize>>>(d_a, d_b, d_c, n);
```

Copying to device

Executing the kernel, with 1024 threads per thread block

CUDA: vector addition



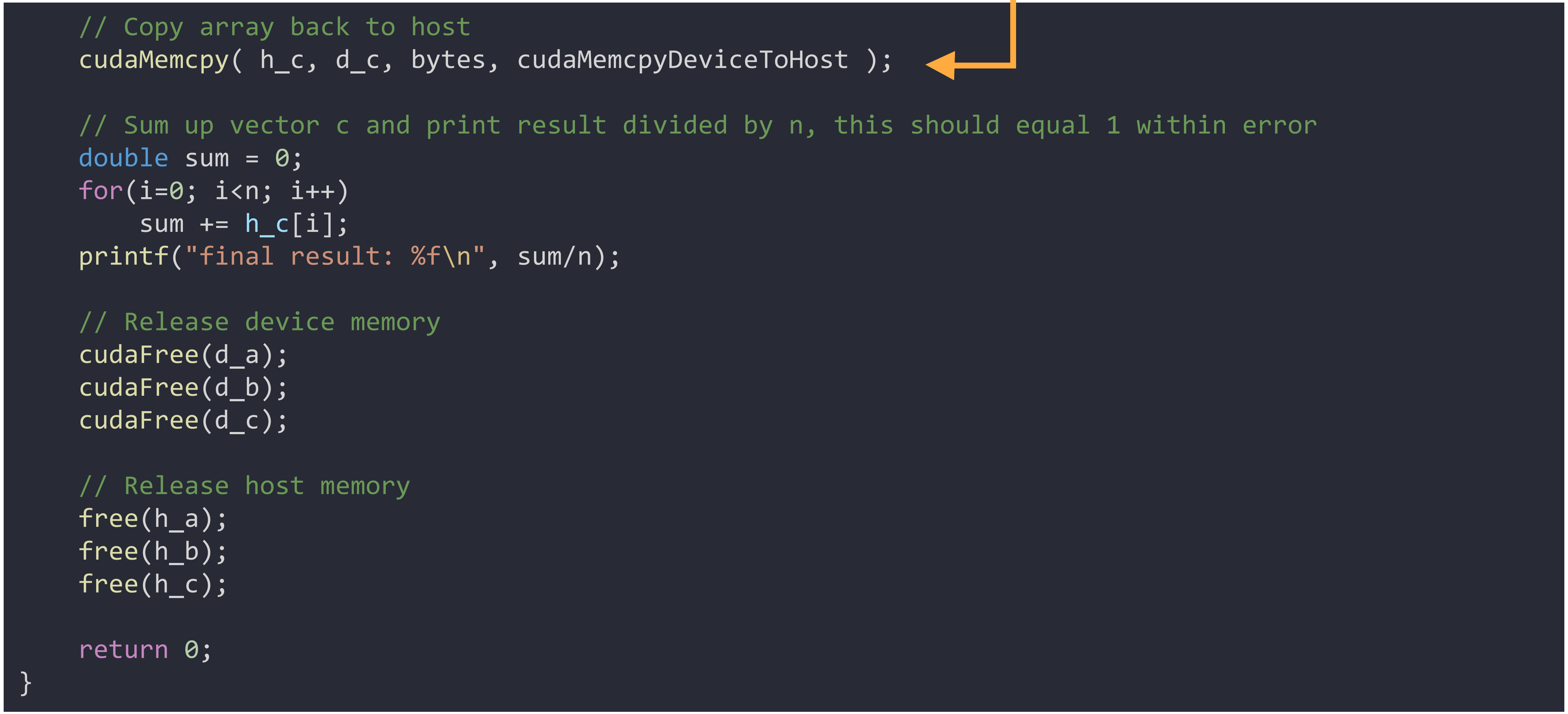
```
// CUDA kernel. Each thread takes care of one element of c
__global__ void vecAdd(double *a, double *b, double *c, int n)
{
    // Get our global thread ID
    int id = blockIdx.x*blockDim.x+threadIdx.x;

    // Make sure we do not go out of bounds
    if (id < n)
        c[id] = a[id] + b[id];
}
```

CUDA kernel, runs on device

From: https://www.olcf.ornl.gov/tutorials/cuda-vector-addition/

CUDA: vector addition

Copying from device to host



```
    // Copy array back to host
    cudaMemcpy( h_c, d_c, bytes, cudaMemcpyDeviceToHost );

    // Sum up vector c and print result divided by n, this should equal 1 within error
    double sum = 0;
    for(i=0; i<n; i++)
        sum += h_c[i];
    printf("final result: %f\n", sum/n);

    // Release device memory
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);

    // Release host memory
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

CUDA libraries

- Math:
  - cuBLAS: basic linear algebra.
  - cuFFT: fast Fourier transforms.
  - cuTENSOR: tensor linear algebra.
  - cuSPARSE: BLAS for sparse matrices.
- Vision, image and video libraries
  - OpenCV: computer vision, machine learning.
  - Gunrock: graph analytics and processing.
- Deep learning:
  - cuDNN: primitives for deep neural networks.
  - Riva: conversation apps.
- Parallel algorithm:
  - Thrust: parallel algorithms and data structures.

Thrust

- C++ template library for CUDA.
- Containers
  - `thrust::host_vector<T>`
  - `thrust::device_vector<T>`
- Algorithms
  - `thrust::sort()`
  - `thrust::reduce()`
  - `thrust::inclusive_scan()`
  - …

Thrust

- Containers to hide `cudaMalloc`, `cudaMemcpy`, `cudaFree`.

```
// allocate host vector with two elements
thrust::host_vector<int> h_vec(2);

// copy host vector to device
thrust::device_vector<int> d_vec = h_vec;

// manipulate device values from the host
d_vec[0] = 13;
d_vec[1] = 27;

std::cout << "sum: " << d_vec[0] + d_vec[1] << std::endl;

// vector memory automatically released w/ free() or cudaFree()
```