

Curso

Profesional de JavaScript

Richard B. Kaufman López
@sparragus

Introducción

**Qué significa ser
un profesional
de JavaScript**





Junior

Senior



Junior

**El duro camino lleno
de experiencia**

Senior

¿Qué forma a un profesional?

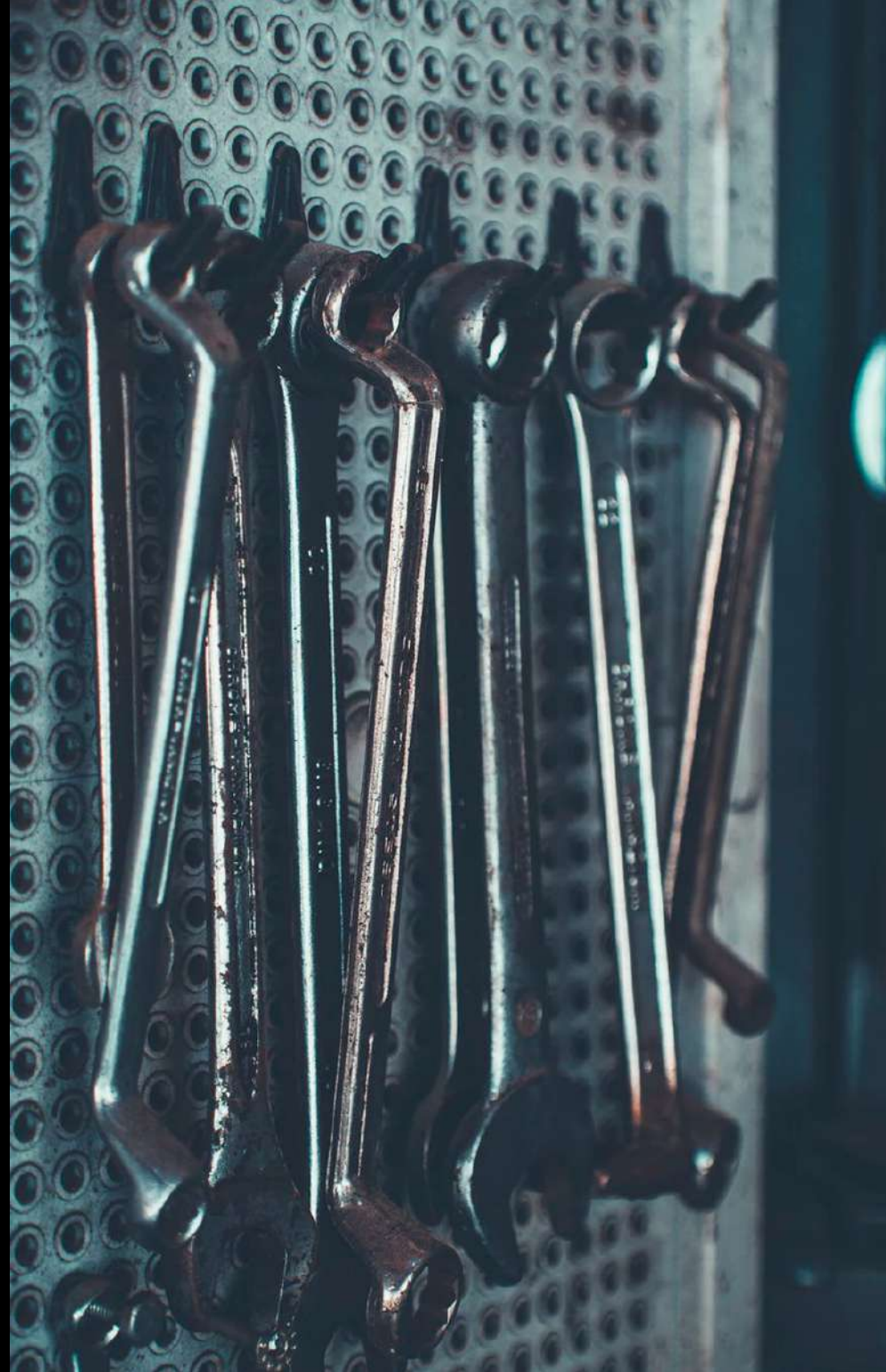
- Conocimiento del lenguaje
- Conocimiento de los entornos de programación
- Mejores prácticas
- Versado en código
- Herramientas
- Ética / Profesionalismo
- Experiencia

El lenguaje: JavaScript

Fundamentos

No-fundamentos

Cómo funciona



No Fundamentos

Promesas (Nivel Pro)

Getters, Setters

Proxies

Generadores

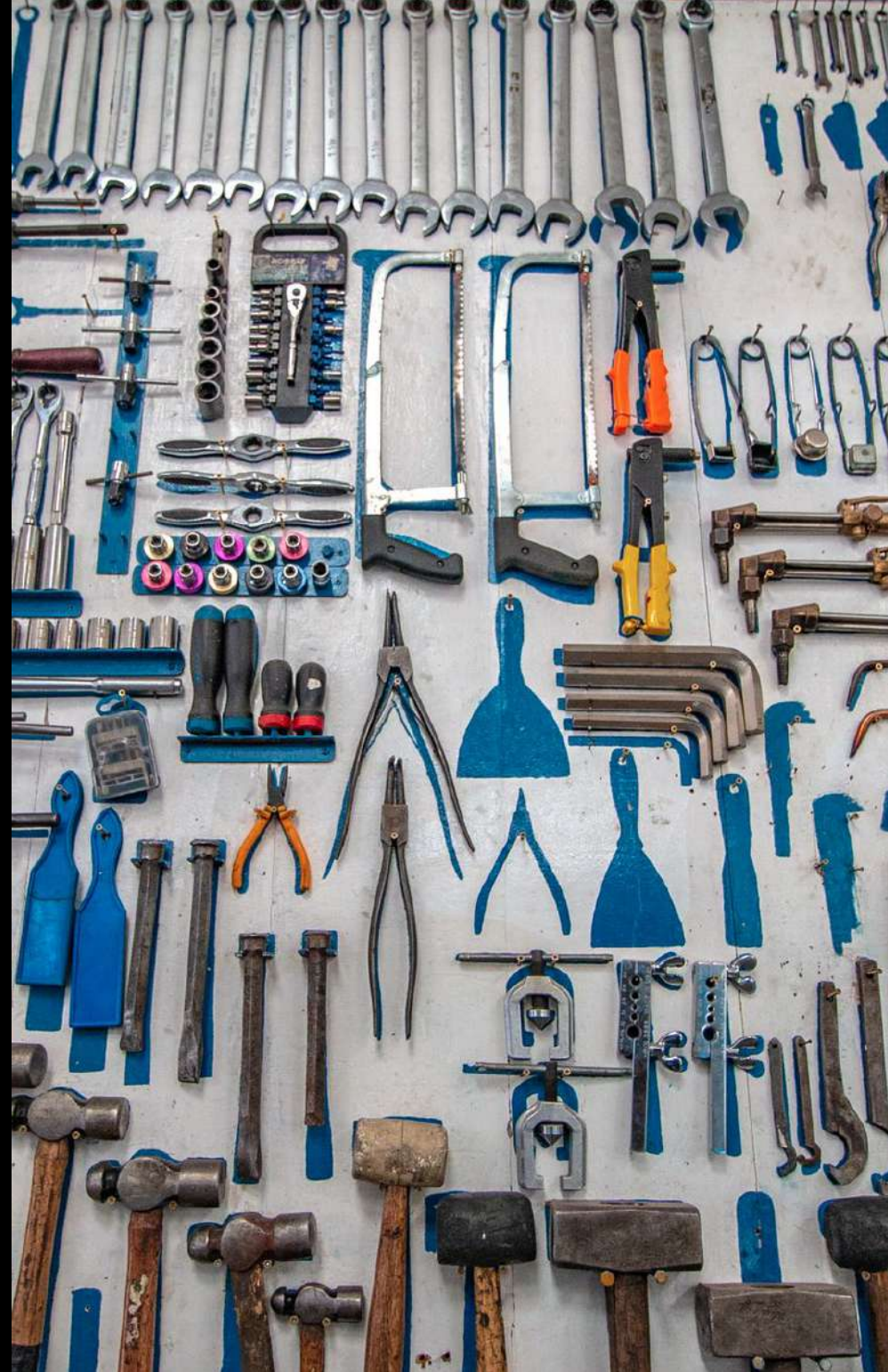


Cómo funciona

JavaScript Engine

Herencia prototipal

Event loop



Entornos de Programación

Browser y el DOM API



Versado en código

Hay que leer código

Mucho

Constantemente



Mejores prácticas

No reinventamos la rueda

Probamos nuestro código



Ética

Ser responsable

Entregar a tiempo

Saber decir que no

No hacer daño



Experiencia

Nada le gana a esto

No se puede enseñar

Está en ti

Perseverancia





Cómo funciona JavaScript

Cómo llega un **script**
al navegador

DOM

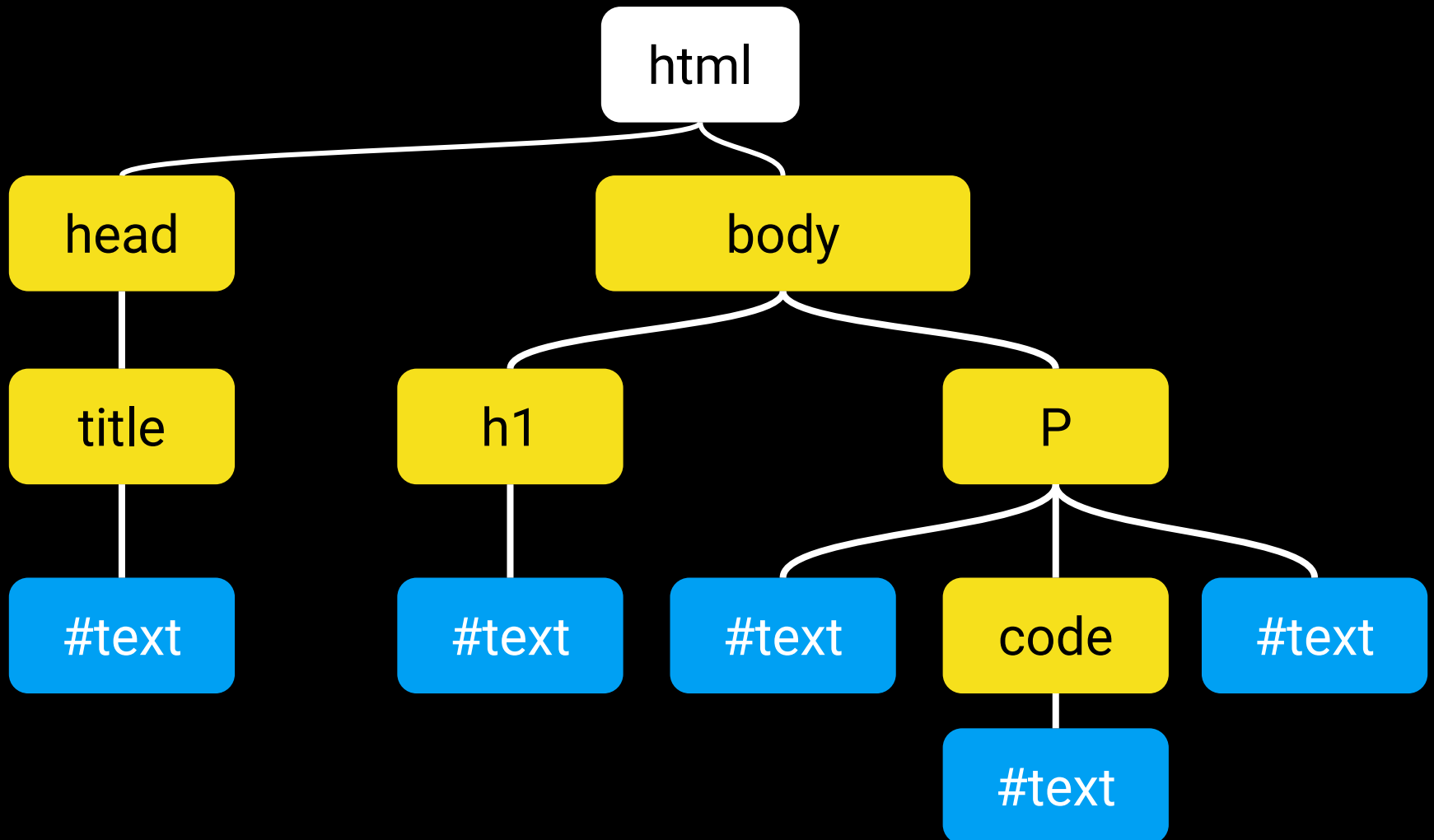
```
<html>
<head><title>DOM</title></head>
<body>
  <h1>DOM</h1>
  <p>
```

Cuando llega el HTML al browser, este lo empieza a parsear: va leyendo etiqueta por etiqueta y va creando el DOM.

Cuando este proceso termina por completo, es cuando obtenemos el evento `DOMContentLoaded`.

```
</p>
</body>
</html>
```

DOM



“

DOMContentLoaded

”

El Navegador

Scripts externos o embebido

```
<html>
<head>
  <!-- Global site tag (gtag.js) - Google Analytics -->
  <script async
src="https://googletagmanager.com/gtag/js?id=U-123456-1"></script>
  <script>
    window.dataLayer = window.dataLayer || [];
    function gtag(){dataLayer.push(arguments);}
    gtag('js', new Date());
    gtag('config', 'GA_MEASUREMENT_ID');
  </script>
</head>
<body><!-- ... --></body>
</html>
```

Scripts embebidos

HTML

Script Execution

```
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());
  gtag('config', 'GA_MEASUREMENT_ID');
</script>
```



```
<body>
```

```
  <script>
```

```
    let formEl = document.querySelector("#loginForm");
```

```
    formEl.addEventListener("submit", function (event) {
```

```
      // ...
```

```
    });
```

```
  </script>
```

```
  <form id="loginForm" action="/login" method="POST">
```

```
    <!-- ... -->
```

```
  </form>
```

```
</body>
```

TypeError



```
<body>
  <script>
    let formEl = document.querySelector("#loginForm");
    formEl.addEventListener("submit", function (event) {
  });
</script>

<form id="loginForm" action="/login" method="POST">
  <input type="text" />
  <input type="password" />
  <input type="submit" value="Login" />
</form>
</body>
```

```
<body>
```

```
  <form id="loginForm" action="/login" method="POST">
```

```
    <!-- ... -->
```

```
  </form>
```

```
  <script>
```

```
    let formEl = document.querySelector("#loginForm");
```

```
    formEl.addEventListener("submit", function (event) {
```

```
      // ...
```

```
    });
```

```
  </script>
```

```
</body>
```

Scripts externos

HTML

Script Fetching

Script Execution

```
<form id="loginForm"><!-- ... --></form>
<script
src="https://unpkg.com/jquery@3.4.1/dist/jquery.js"></script>
<script>
  $("#loginForm").submit(function (event) { /*...*/ });
</script>
```

Scripts externos (async)

HTML

Script Fetching

Script Execution

```
<head>
```

```
  <!-- Global site tag (gtag.js) - Google Analytics -->
```

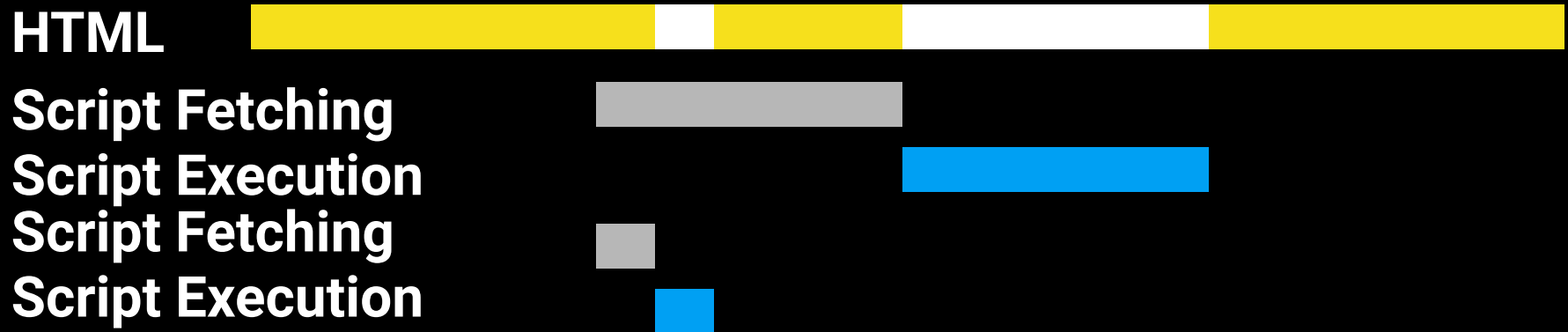
```
  <script async
```

```
  src="https://googletagmanager.com/gtag/js?id=U-123456-1"></script>
```

```
</head>
```

```
<body><!-- ... --></body>
```


Orden de ejecución (async)



```
<body>  
  <script async src="/hugeScript.js"></script>  
  <script async src="/smallScript.js"></script>  
</body>
```

Scripts externos (defer)

HTML

Script Fetching

Script Execution

```
<body>
```

```
  <script defer src="/index.js"></script>
```

```
  <!-- A bunch of html -->
```

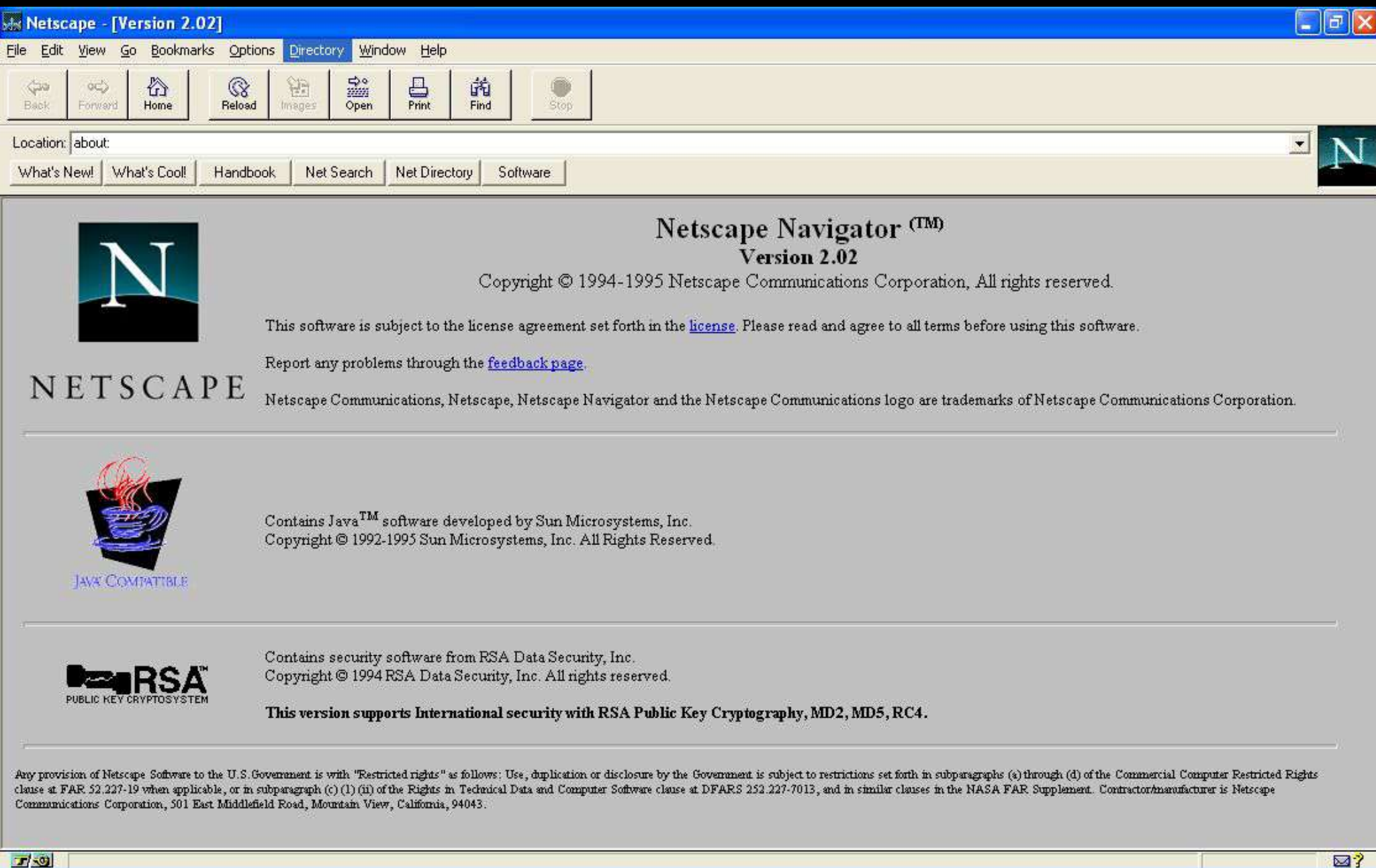
```
</body>
```



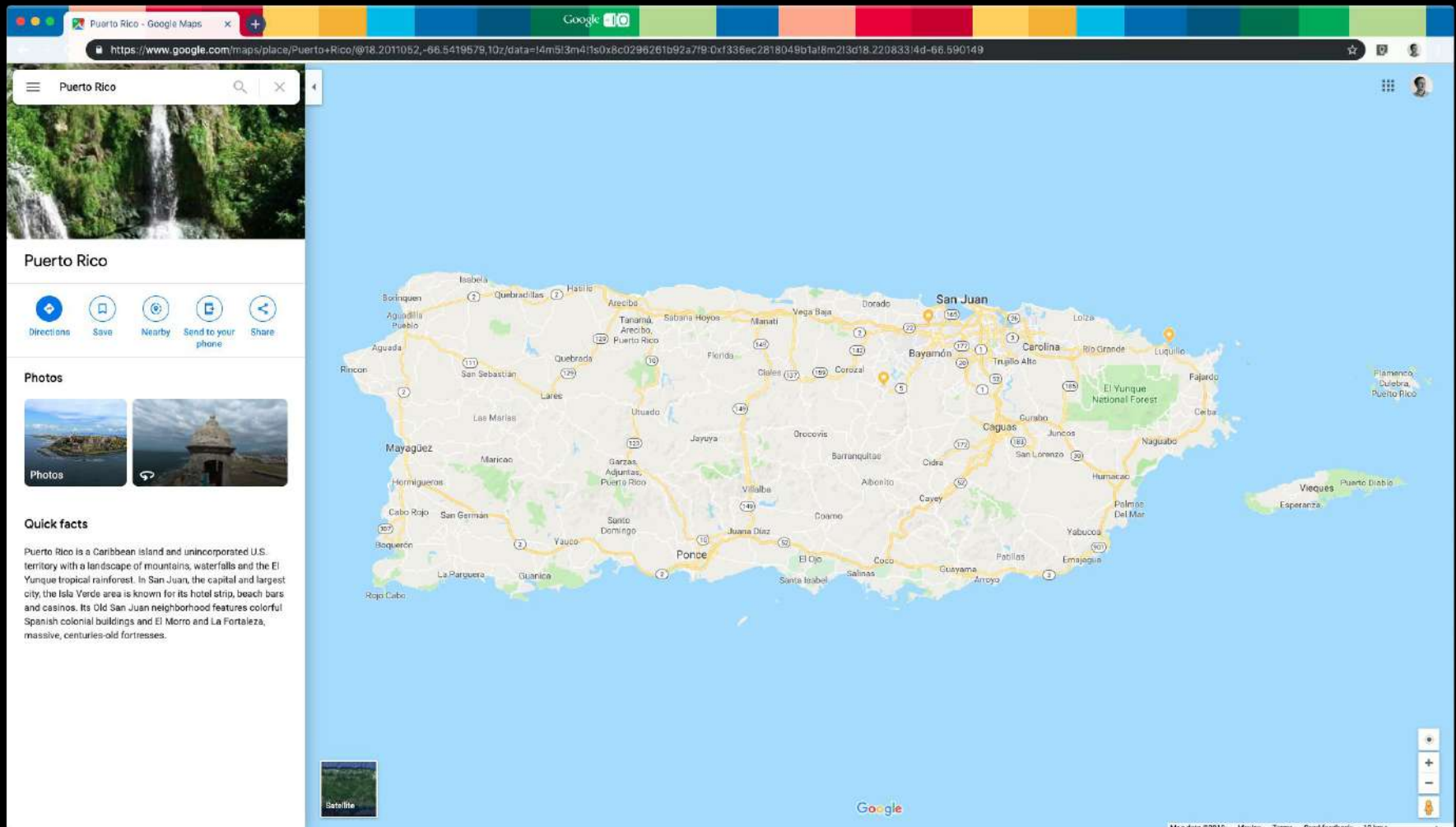
Cómo funciona JavaScript

Parsers y el **Abstract Syntax Tree**

JavaScript es interpretado



La web no es como antes



JavaScript Engines

V8 - Chrome, Node.js

SpiderMonkey - Firefox

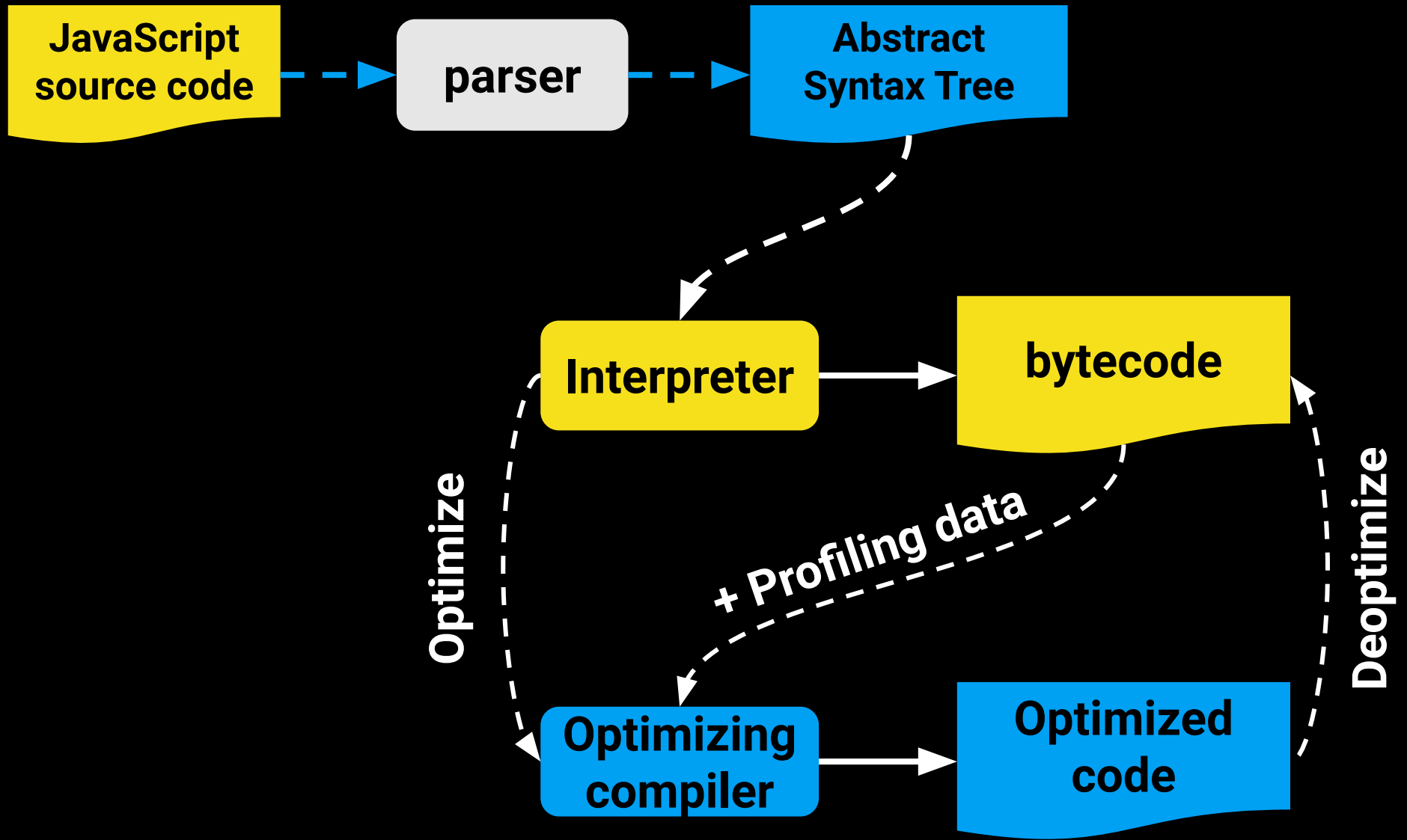
JavaScriptCore - Safari

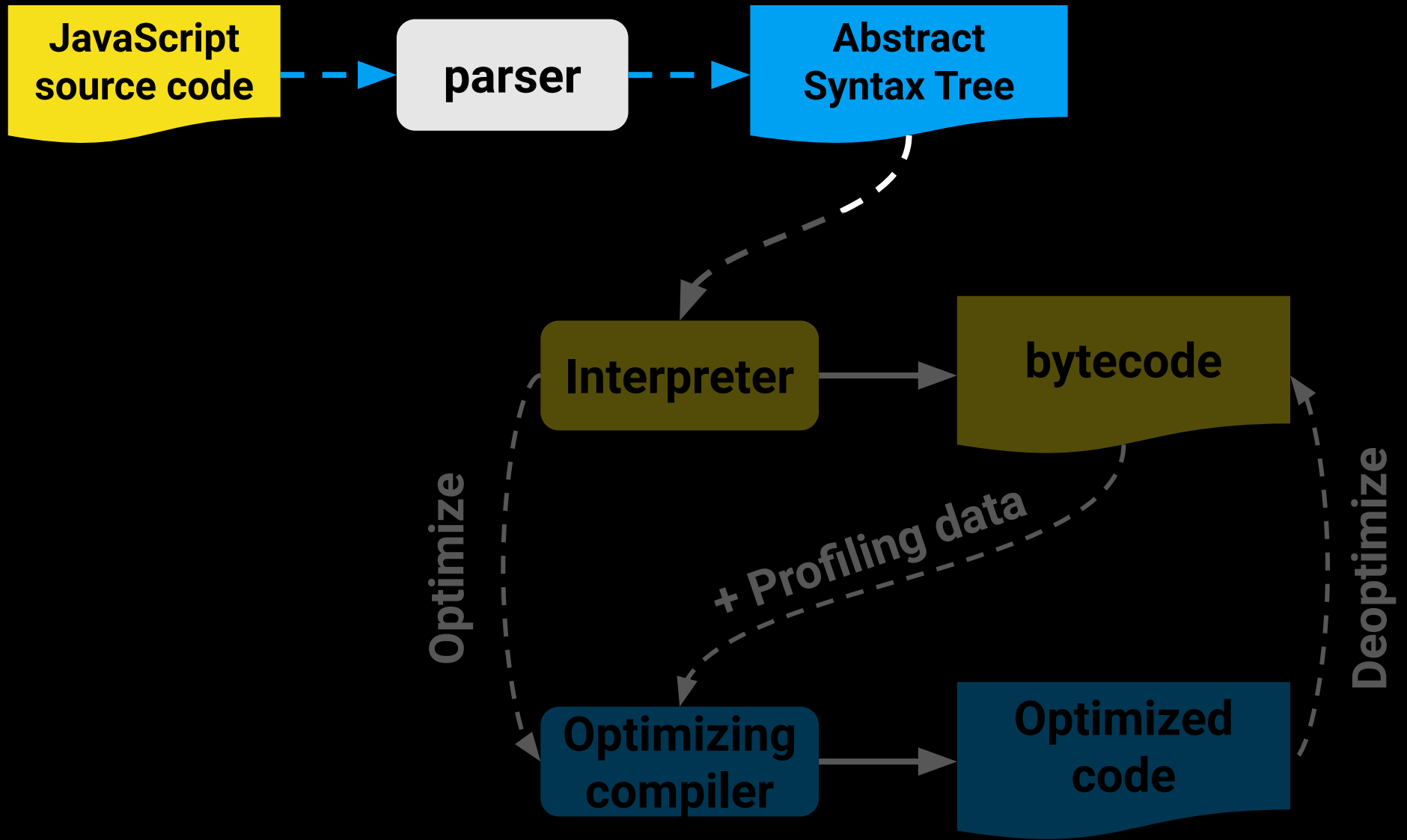
Chakra - Edge



¿Qué hace un JS Engine?

- Recibe código fuente
- Parsea el código y produce un Abstract Syntax Tree (AST)
- Se compila a bytecode y se ejecuta
- Se optimiza a machine code y se reemplaza el código base





¿Qué hace un parser?



¿Qué hace un parser?

Un **SyntaxError** es lanzado cuando el motor de JavaScript se encuentra con partes de código que no forman parte de la sintaxis del lenguaje al momento analizar el código.

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/SyntaxError

Google dice:

- Parsing es **15-20%** del proceso de ejecución.
- La **mayoría** del JavaScript en una página nunca se ejecuta.
- Esto hace que **bundling y code splitting** sea muy importante!

Parser de V8

Eager Parsing:

- Encuentra errores de sintaxis
- Crea el AST
- Construye scopes

Lazy Parsing:

- Doble de rápido que el eager parser
- No crea el AST
- Construye los scopes parcialmente

Demo - Tokens

Abstract Syntax Tree (AST)

Es un grafo (estructura de datos) que representa un programa.

Se usa en:

- JavaScript Engine
- Bundlers: Webpack, Rollup, Parcel
- Transpilers: Babel
- Linters: ESLint, Prettify
- Type Checkers: TypeScript, Flow
- Syntax Highlighters

Demo - AST

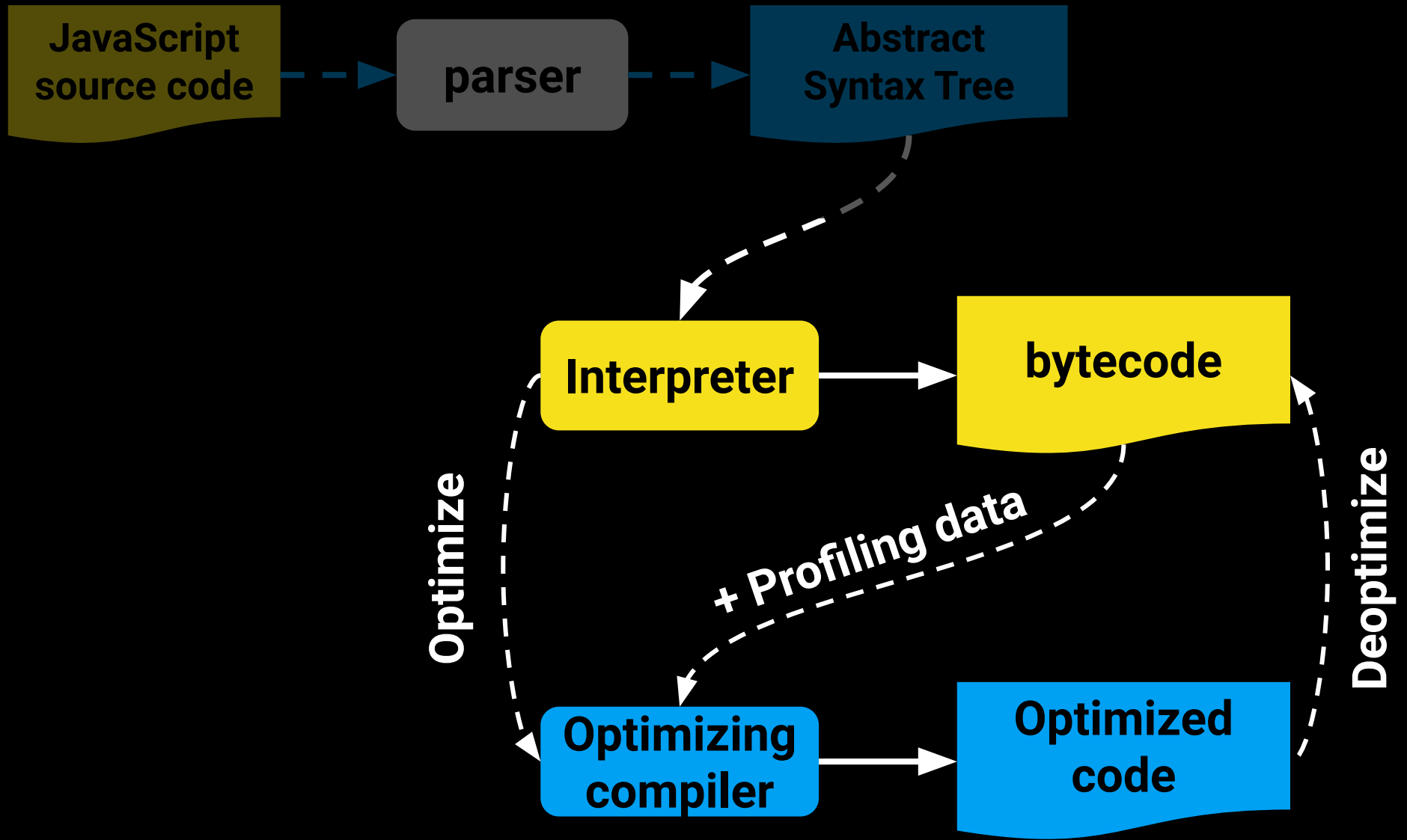
Demo - ESLint Rule

Cómo funciona JavaScript

JavaScript Engine:
Interpreter, Compiler
& Código Optimizado

¿Qué hace un JS Engine?

- Recibe código fuente
- Parsea el código y produce un Abstract Syntax Tree (AST)
- Se compila a bytecode y se ejecuta
- Se optimiza a machine code y se reemplaza el código base



Interpreter vs Optimizing Compiler

Interpreter:

Interpretador
Genera bytecode

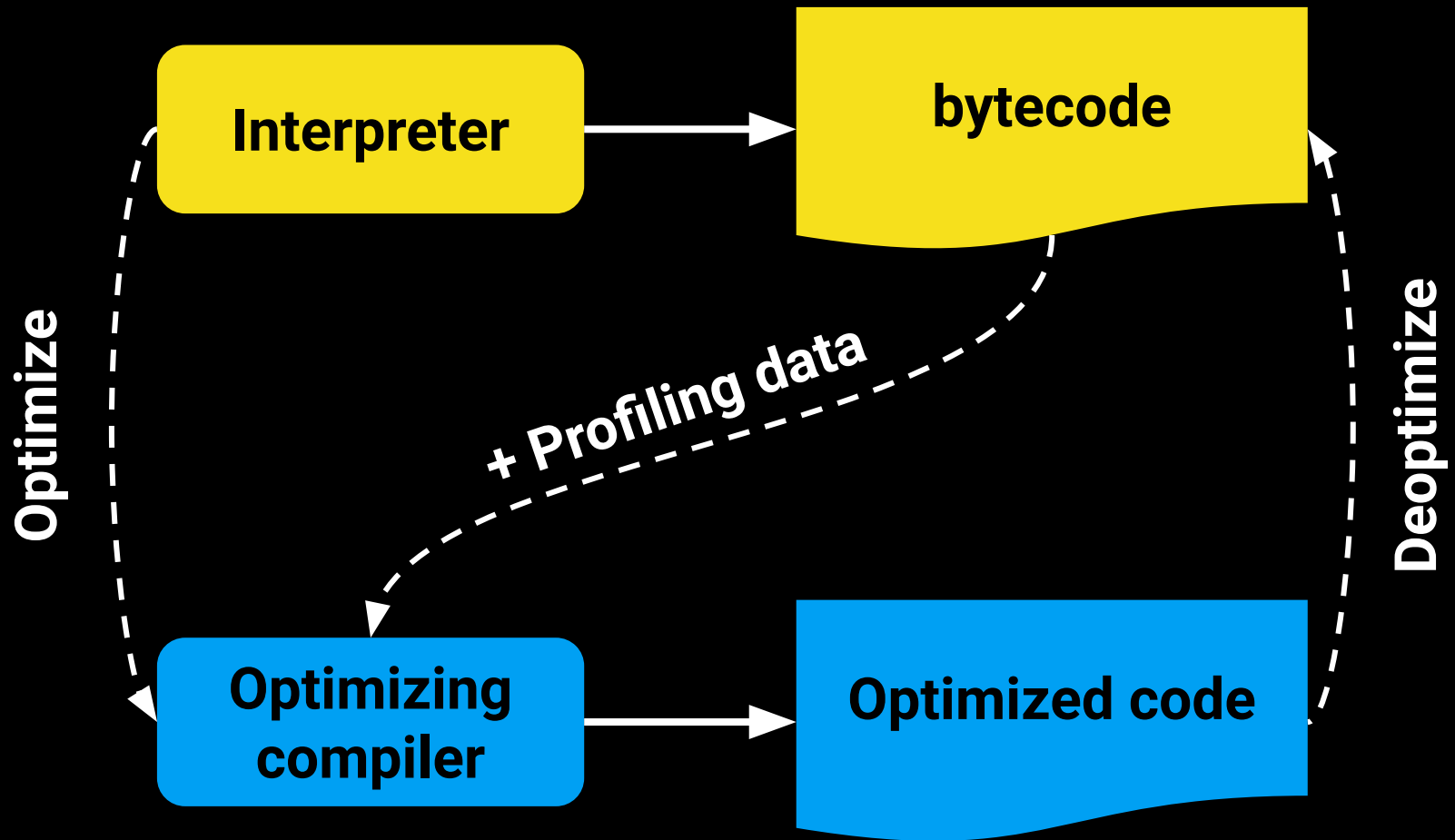
Optimizing Compiler:

Genera machine code
Optimiza el bytecode
usando estadísticas de
ejecución
Just In Time compiler

Bytecode vs Machine Code

- Código parecido a assembly.
 - Portatil.
 - Ejecutado por una virtual machine.
- Binario.
 - Instrucciones específicas a una arquitectura o procesador.

V8



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

```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```

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

```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```

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

```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```



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

```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```

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

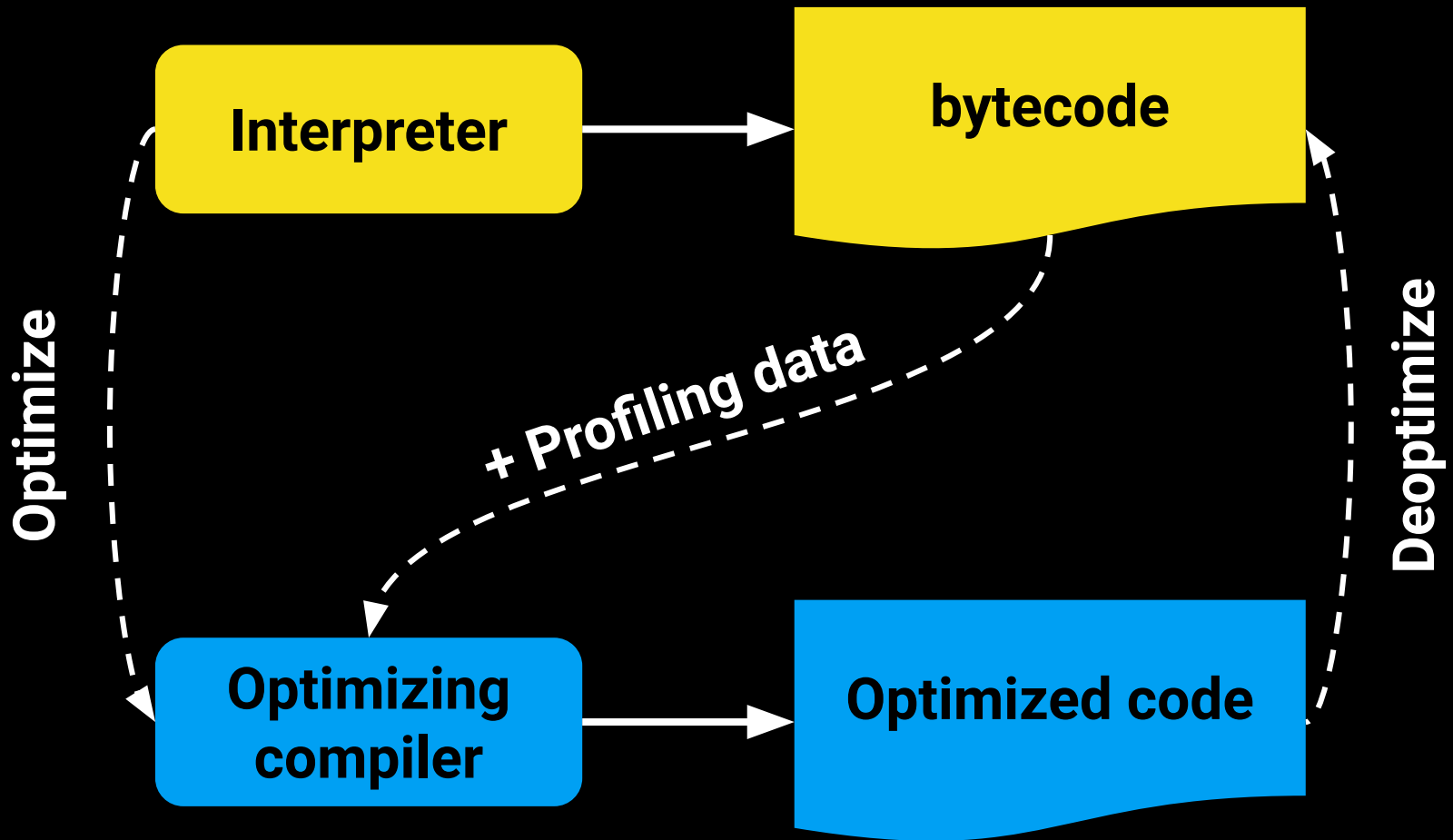
```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```

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



```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}
```

V8

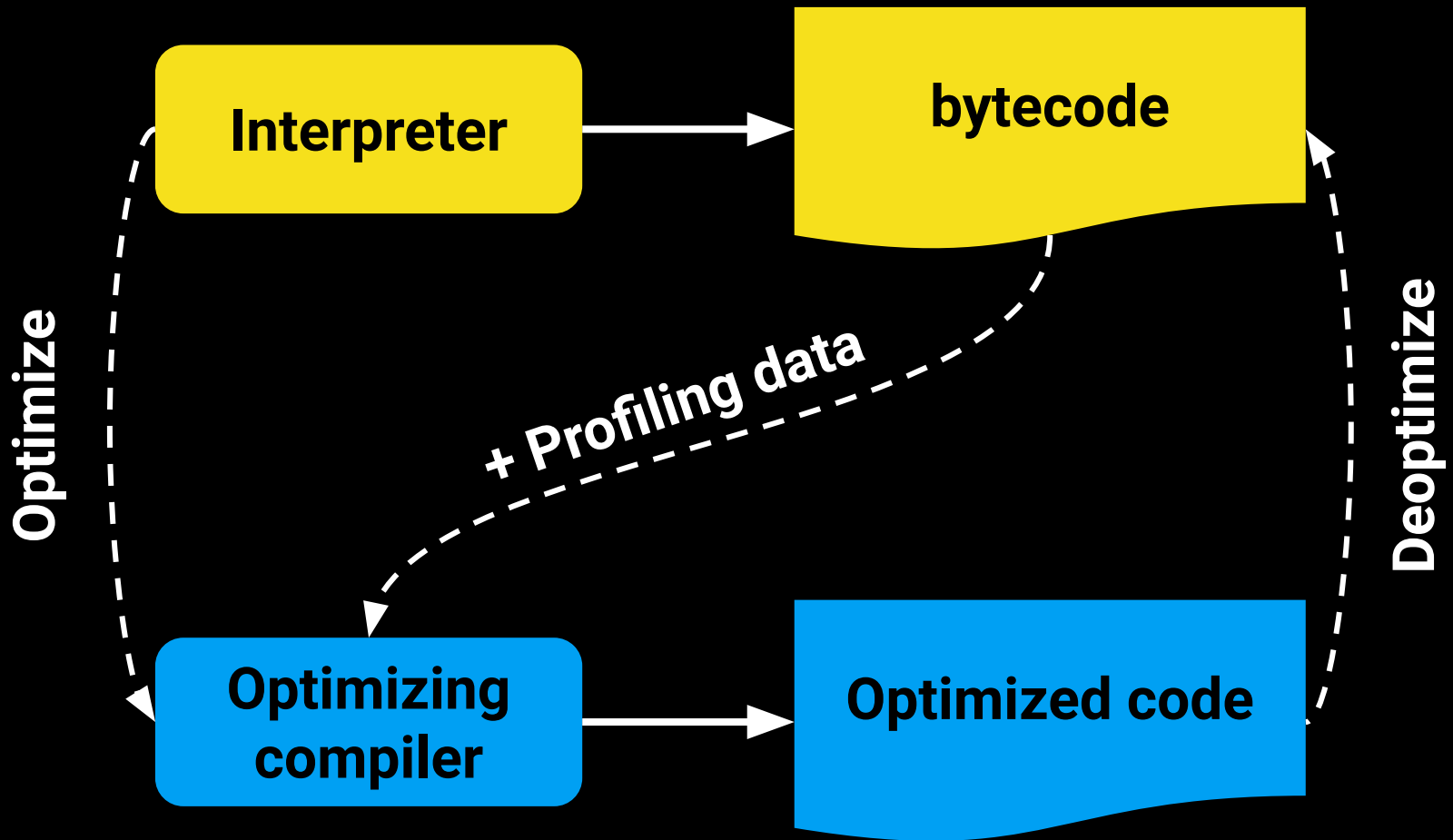


```
function add (a, b) {  
  return a + b;  
}
```

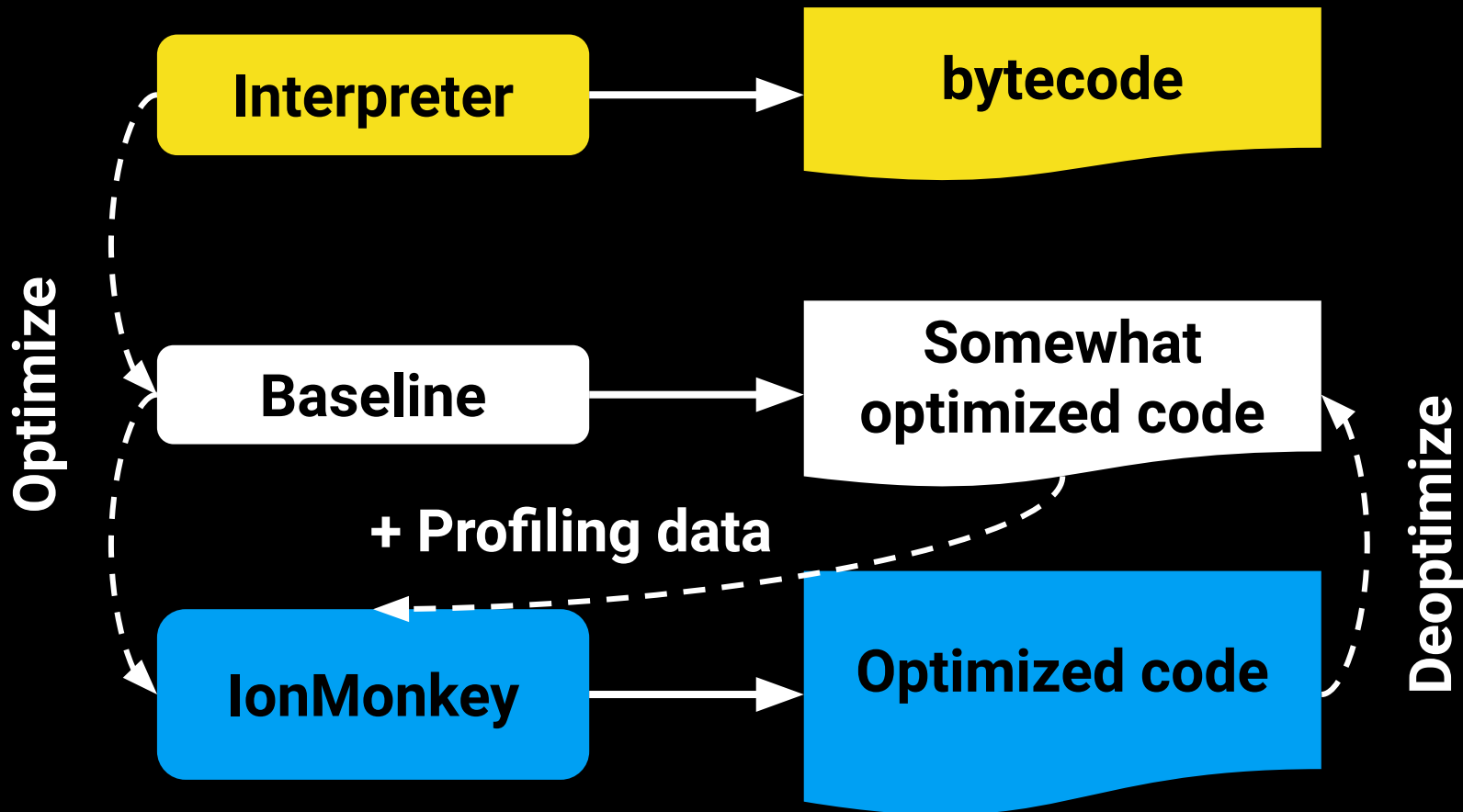


```
for (let i = 0; i < 1000; i++) {  
  add(i, i);  
}  
add(1, "uno");
```

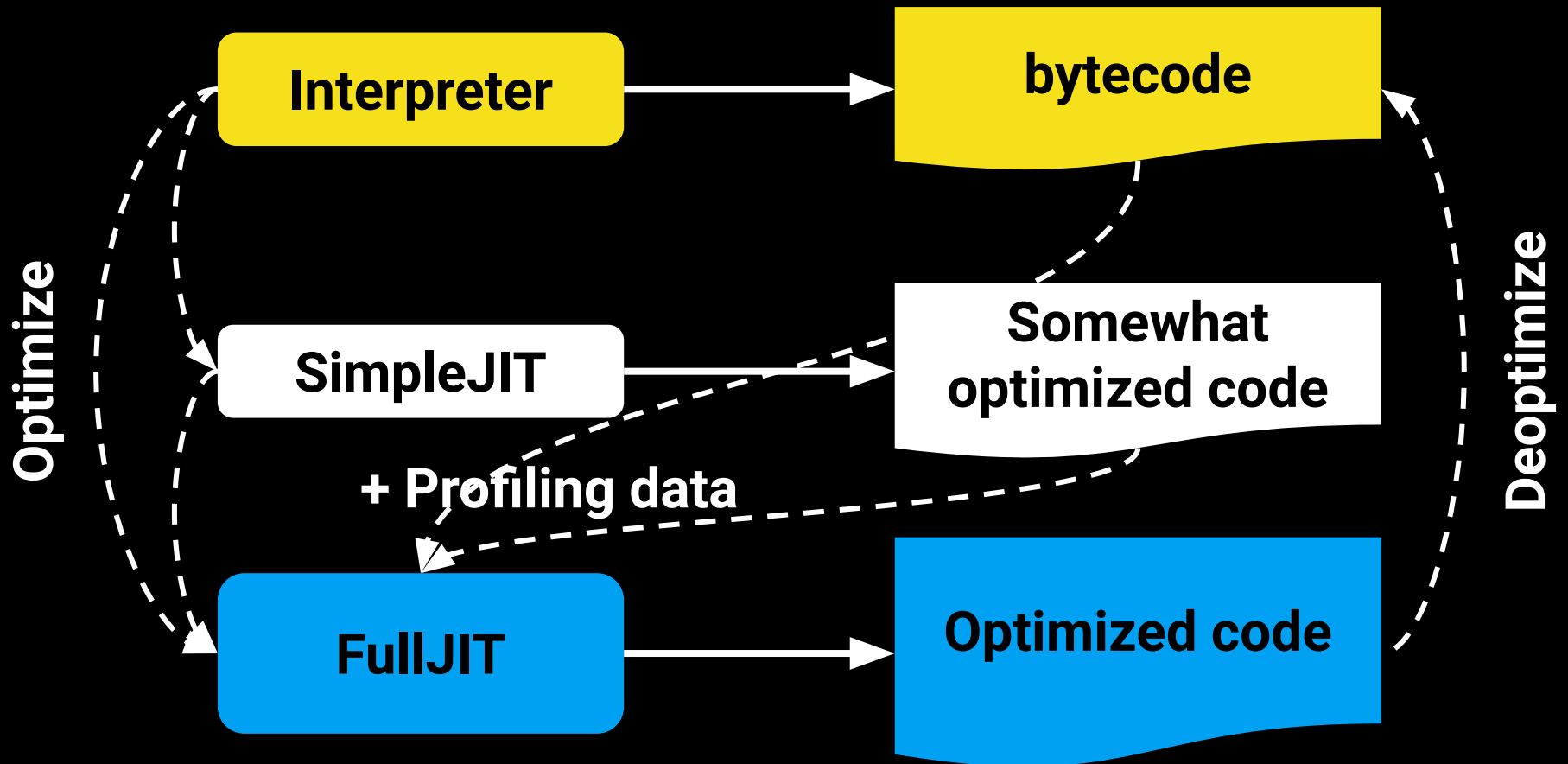
V8



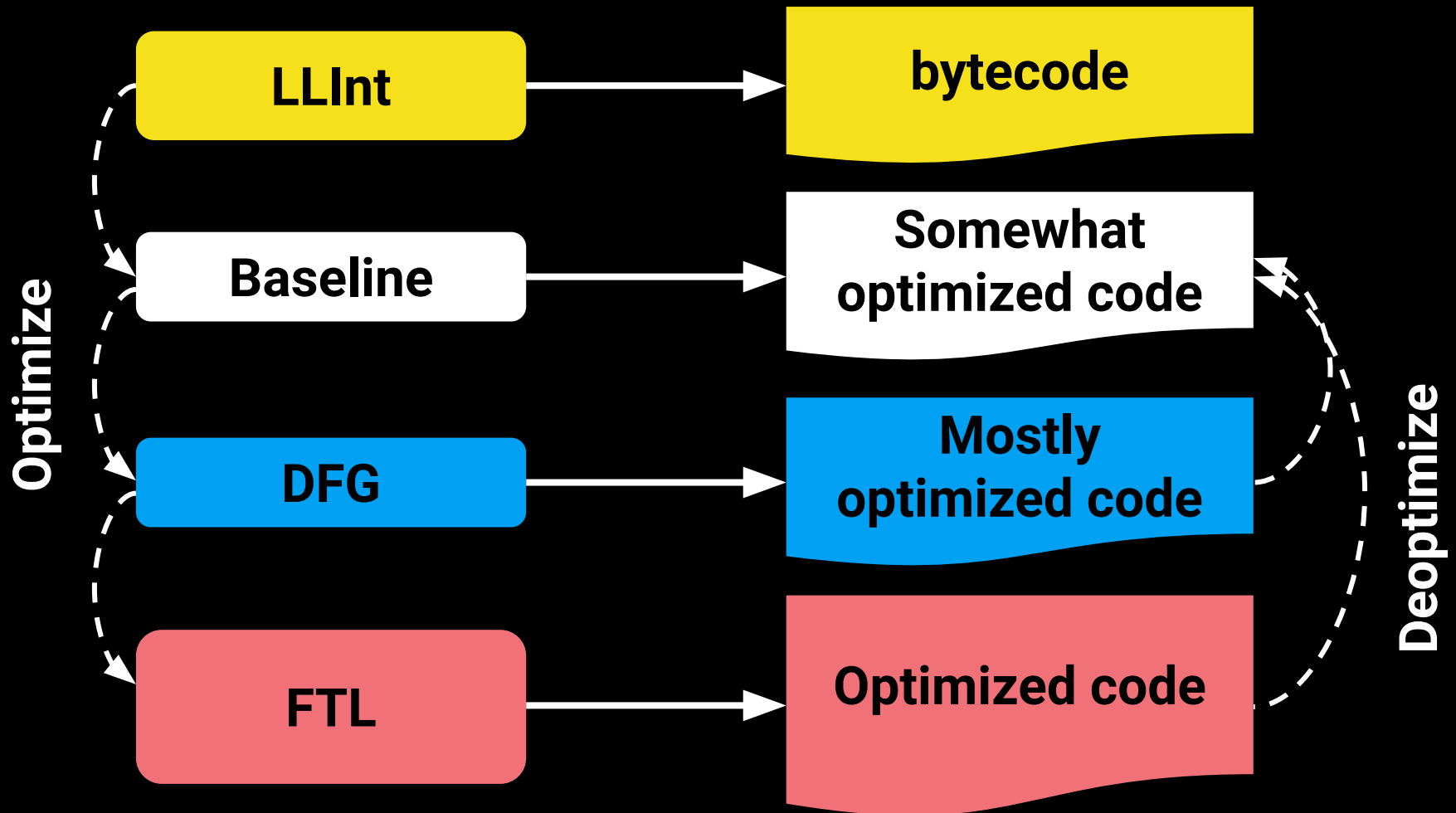
SpiderMonkey



10000000



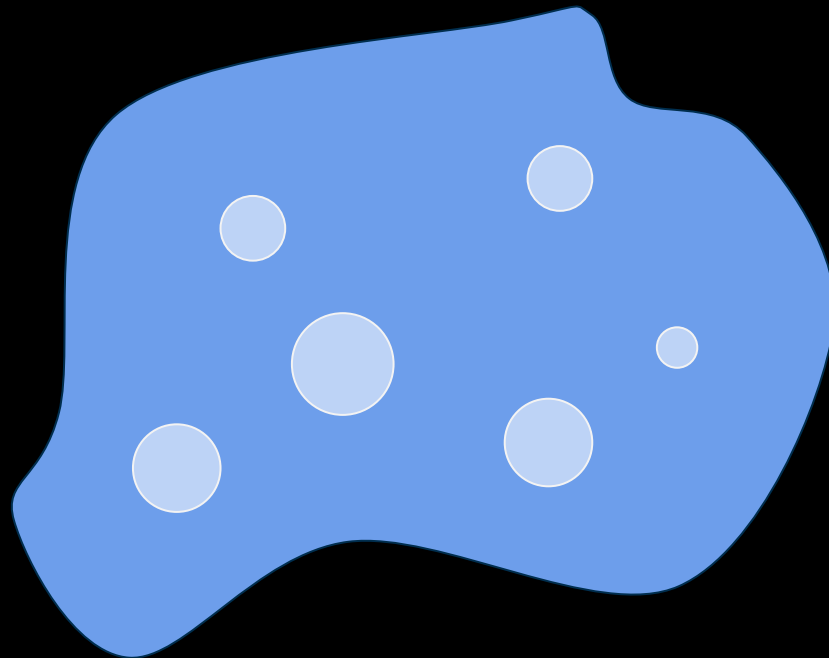
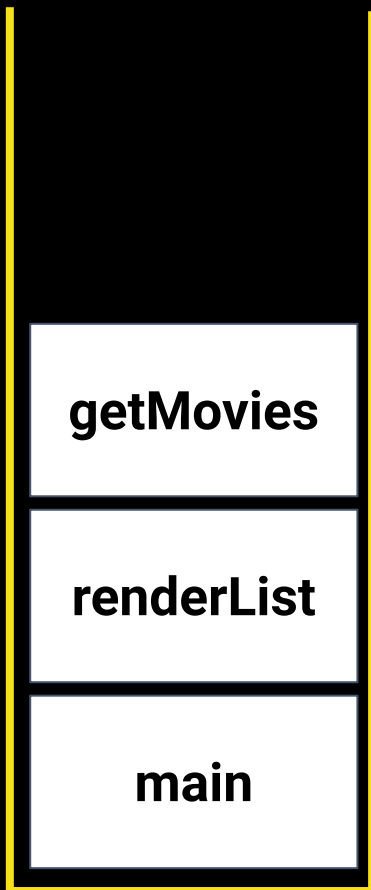
JavaScriptCore



Cómo funciona JavaScript

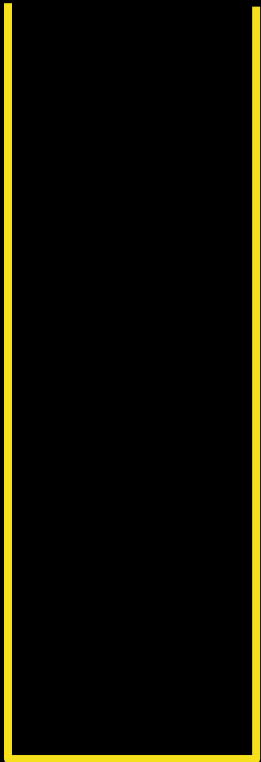
Event loop

Stack y Memory Heap





Stack



Stack

Push

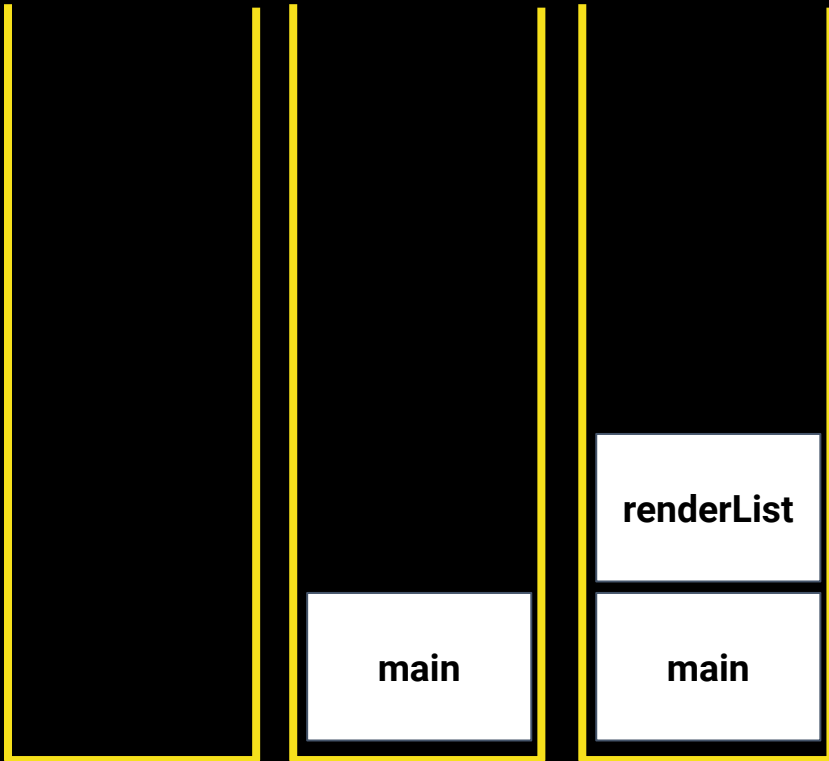


The diagram illustrates a stack data structure. It consists of two vertical yellow lines representing the boundaries of the stack. The right boundary contains a white rectangular box at the bottom, labeled 'main'. The left boundary is empty.

main

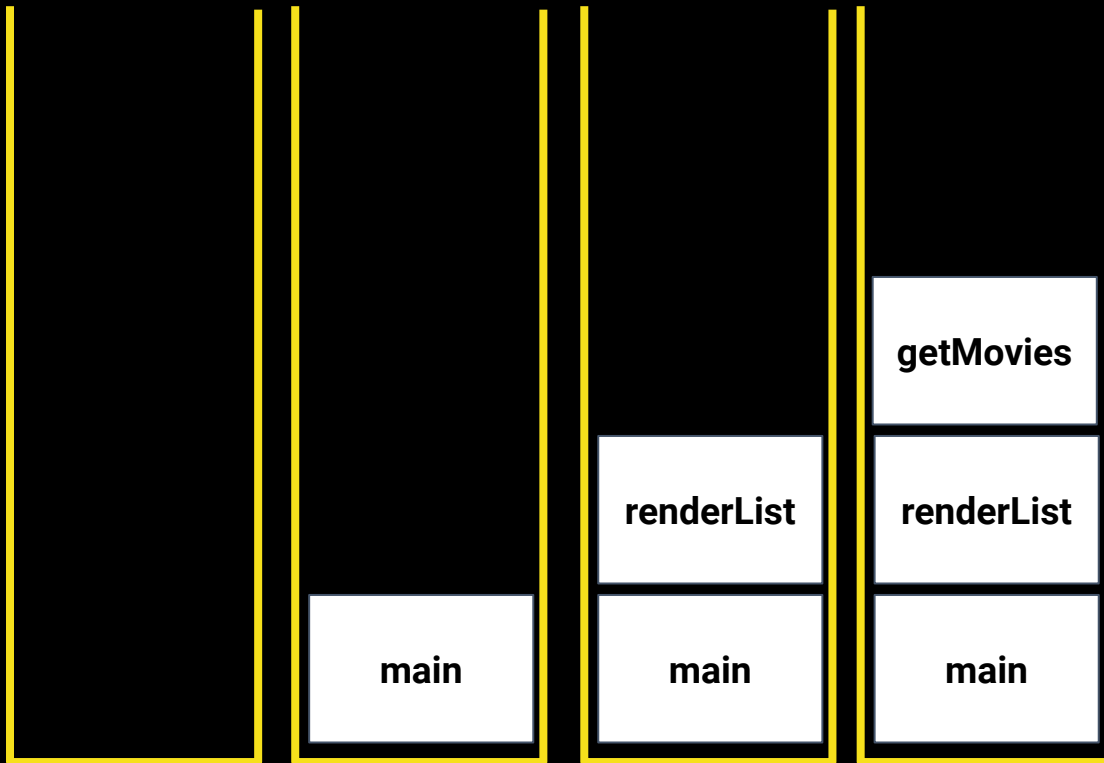
Stack

Push



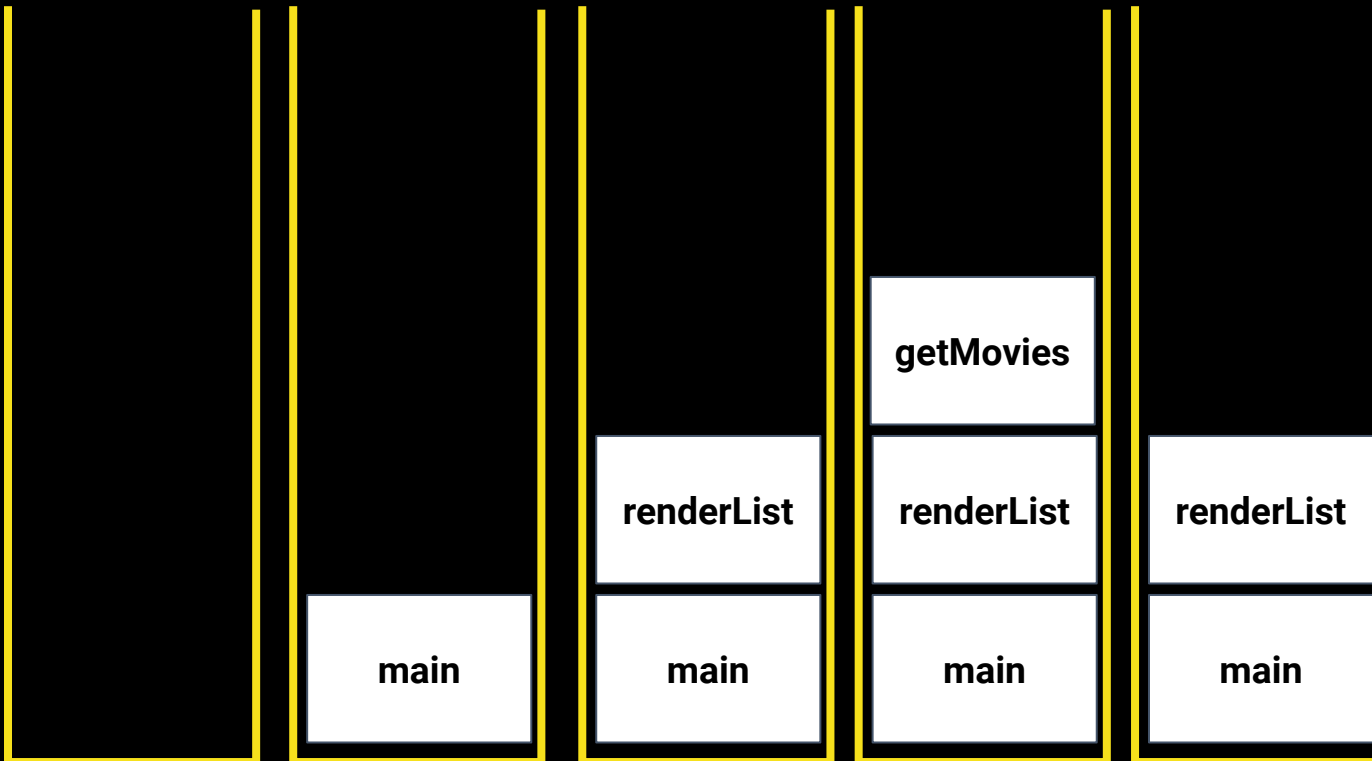
Stack

Push



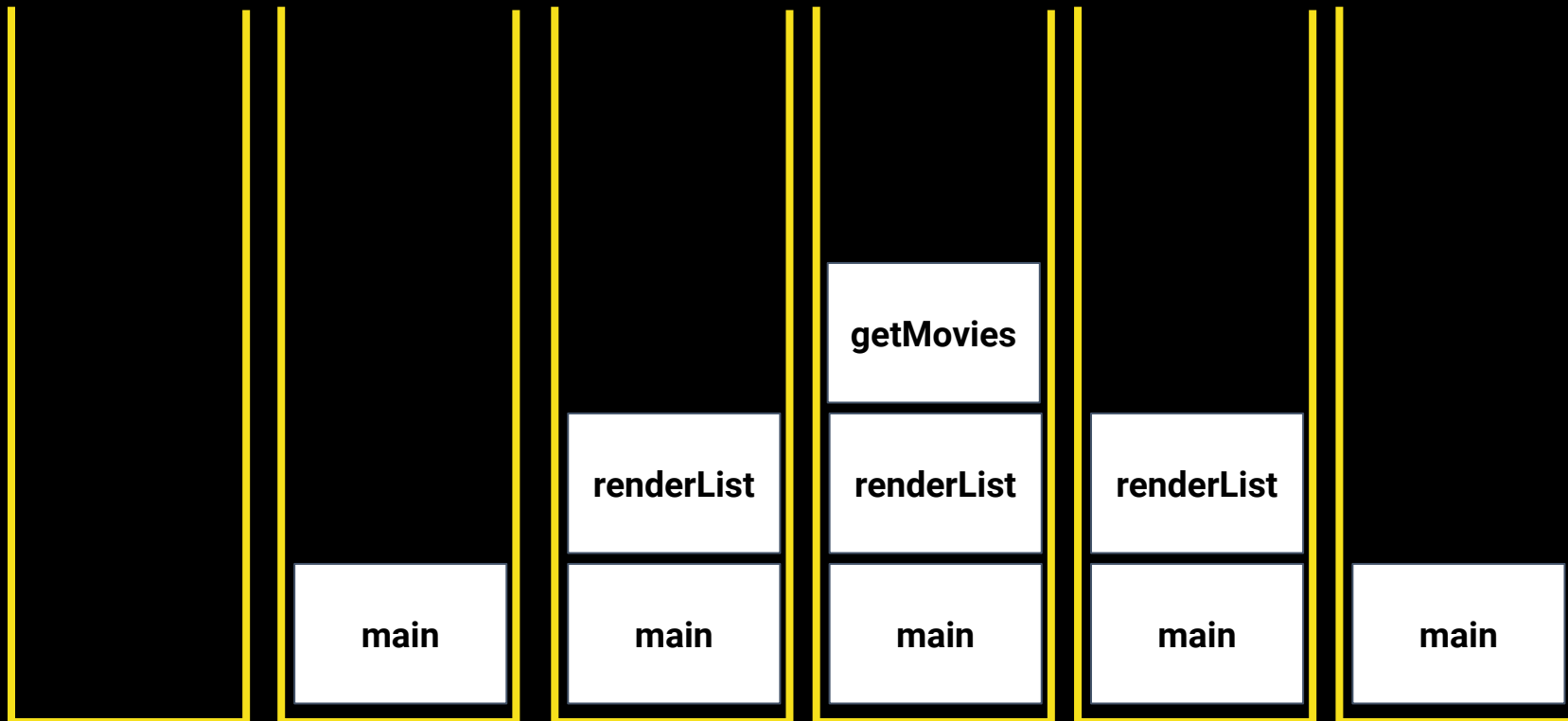
Stack

Pop



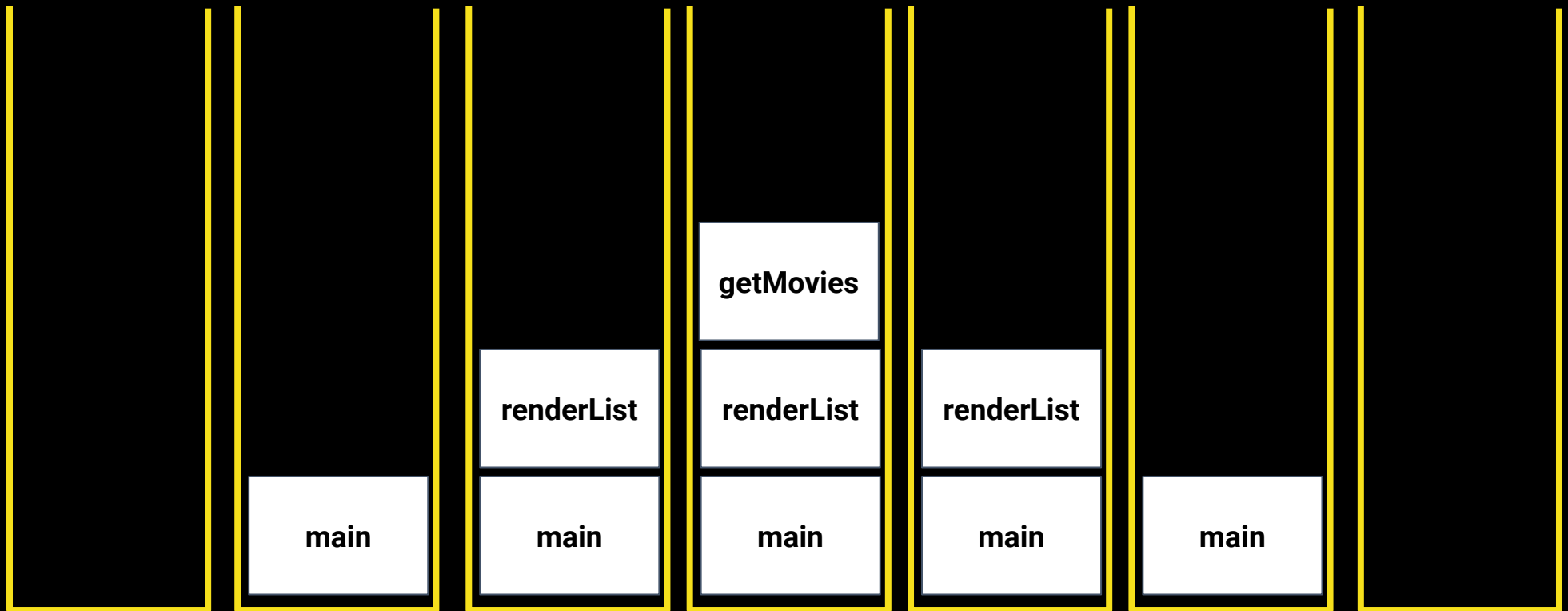
Stack

Pop

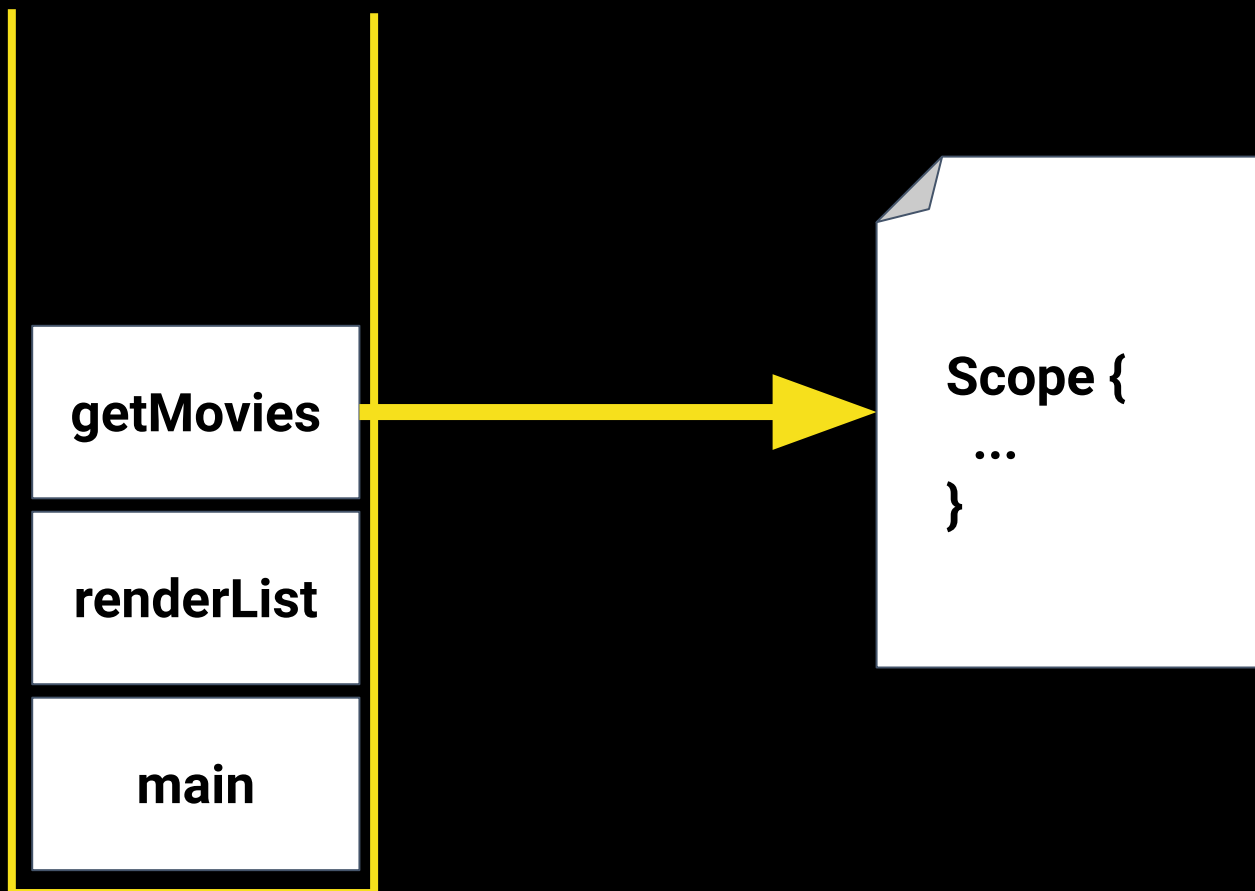


Stack


Pop



Stack



Programa



```
function hello () {  
    console.log("Hello")  
}  
function world() {  
    console.log("World")  
}  
function main() {  
    hello();  
    world();  
}  
main();
```

(anonymous)

Programa

```
function hello () {  
    console.log("Hello")  
}  
→ function world() {  
    console.log("World")  
}  
function main() {  
    hello();  
    world();  
}  
main();
```

(anonymous)

Programa

```
function hello () {  
    console.log("Hello")  
}  
function world() {  
    console.log("World")  
}  
→ function main() {  
    hello();  
    world();  
}  
main();
```

(anonymous)

Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```


(anonymous)

Programa

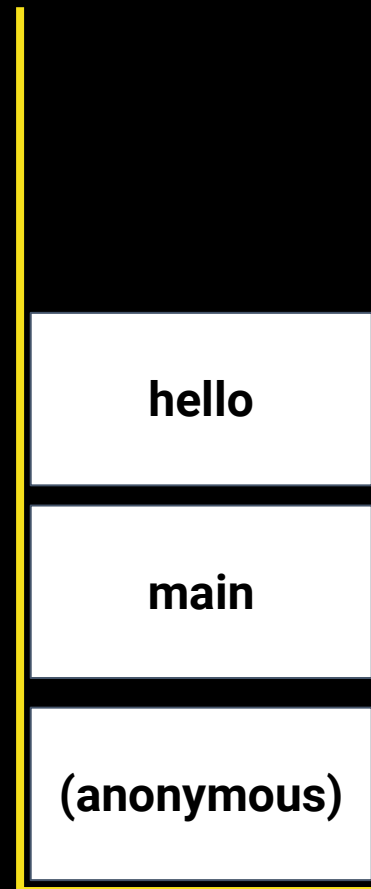
```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa



```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```

console.log

hello

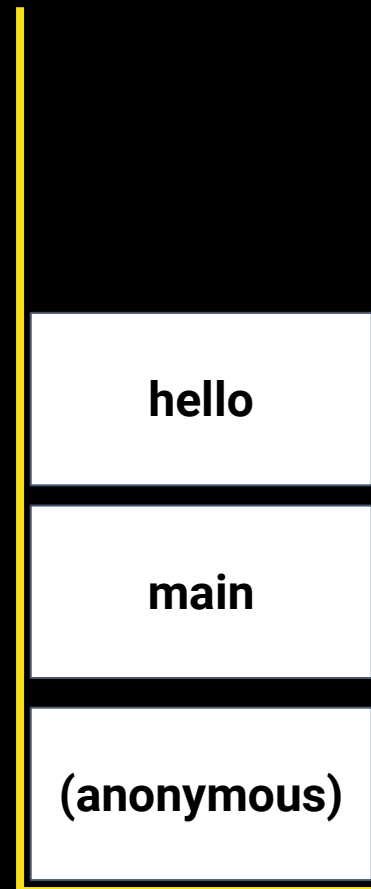
main

(anonymous)

Programa



```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```

console.log

world

main

(anonymous)

Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```



Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```

(anonymous)

Programa

```
function hello () {  
  console.log("Hello")  
}  
function world() {  
  console.log("World")  
}  
function main() {  
  hello();  
  world();  
}  
main();
```

Programa Asíncrono



```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

→ `asyncHelloWorld();`

(anonymous)

Programa Asíncrono



```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

setTimeout

asyncHelloWorld

(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

console.log

asyncHelloWorld

(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```



```
asyncHelloWorld();
```

(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

Programa Asíncrono



```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

console.log

(anonymous)

Programa Asíncrono



```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

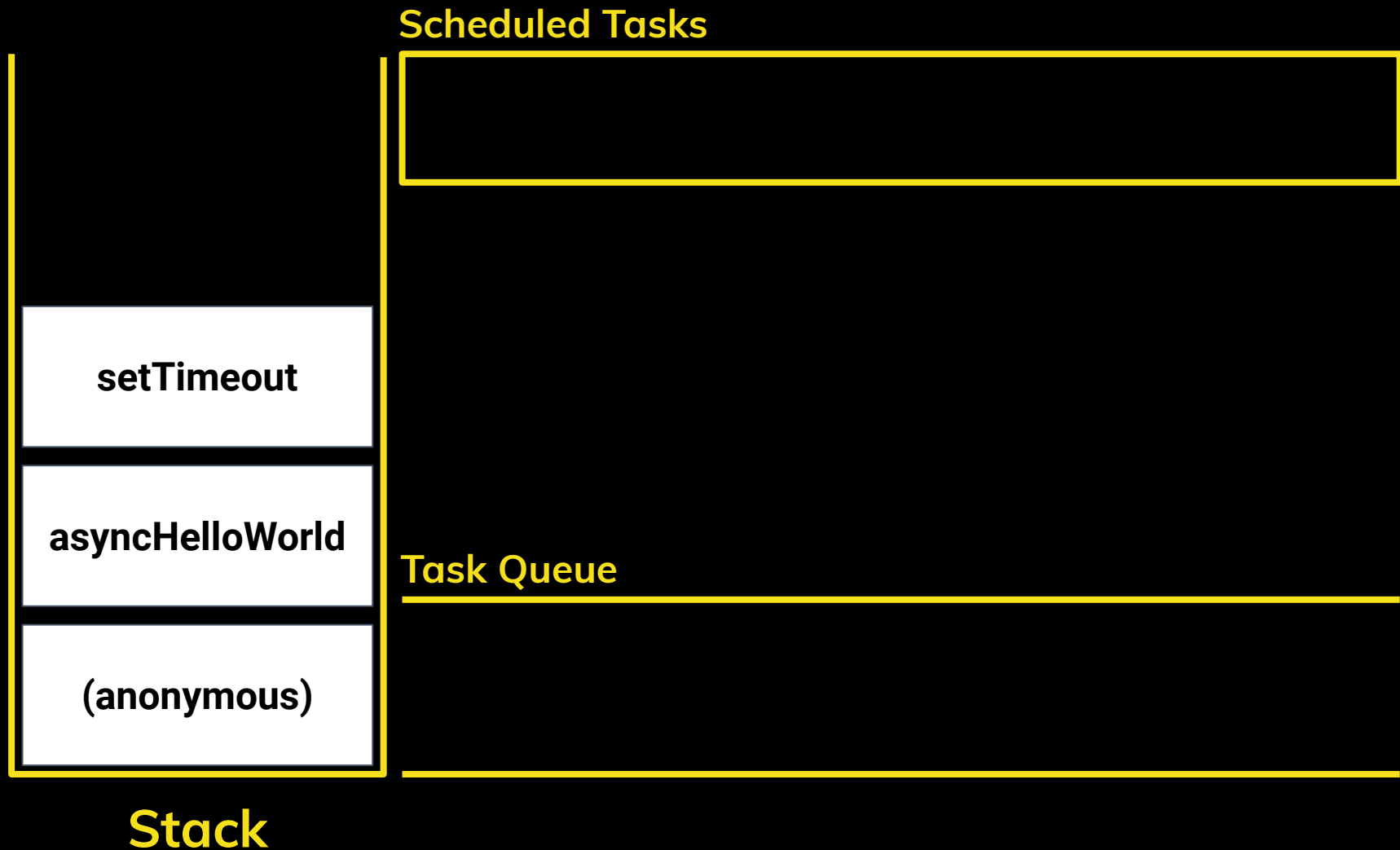
(anonymous)

Programa Asíncrono

```
function asyncHelloWorld () {  
  setTimeout(() => {  
    console.log("hello")  
  }, 1000);  
  console.log("world!");  
}
```

```
asyncHelloWorld();
```

Task Queue



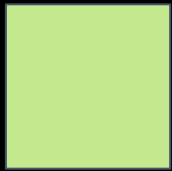


Queue

Queue



Queue



Queue

Queue



Queue

Queue



Queue

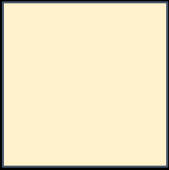
Queue



Queue



Queue



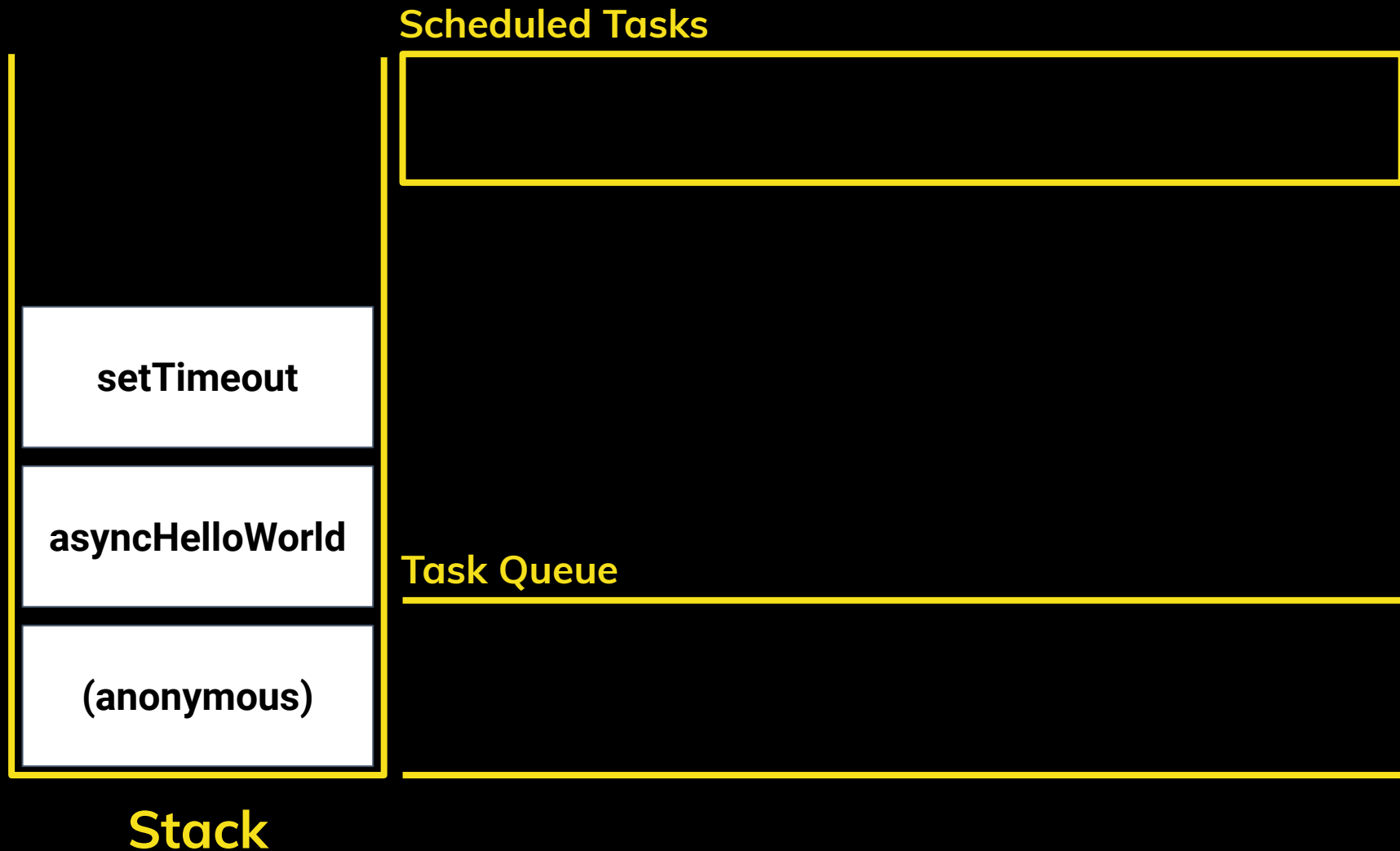
Queue



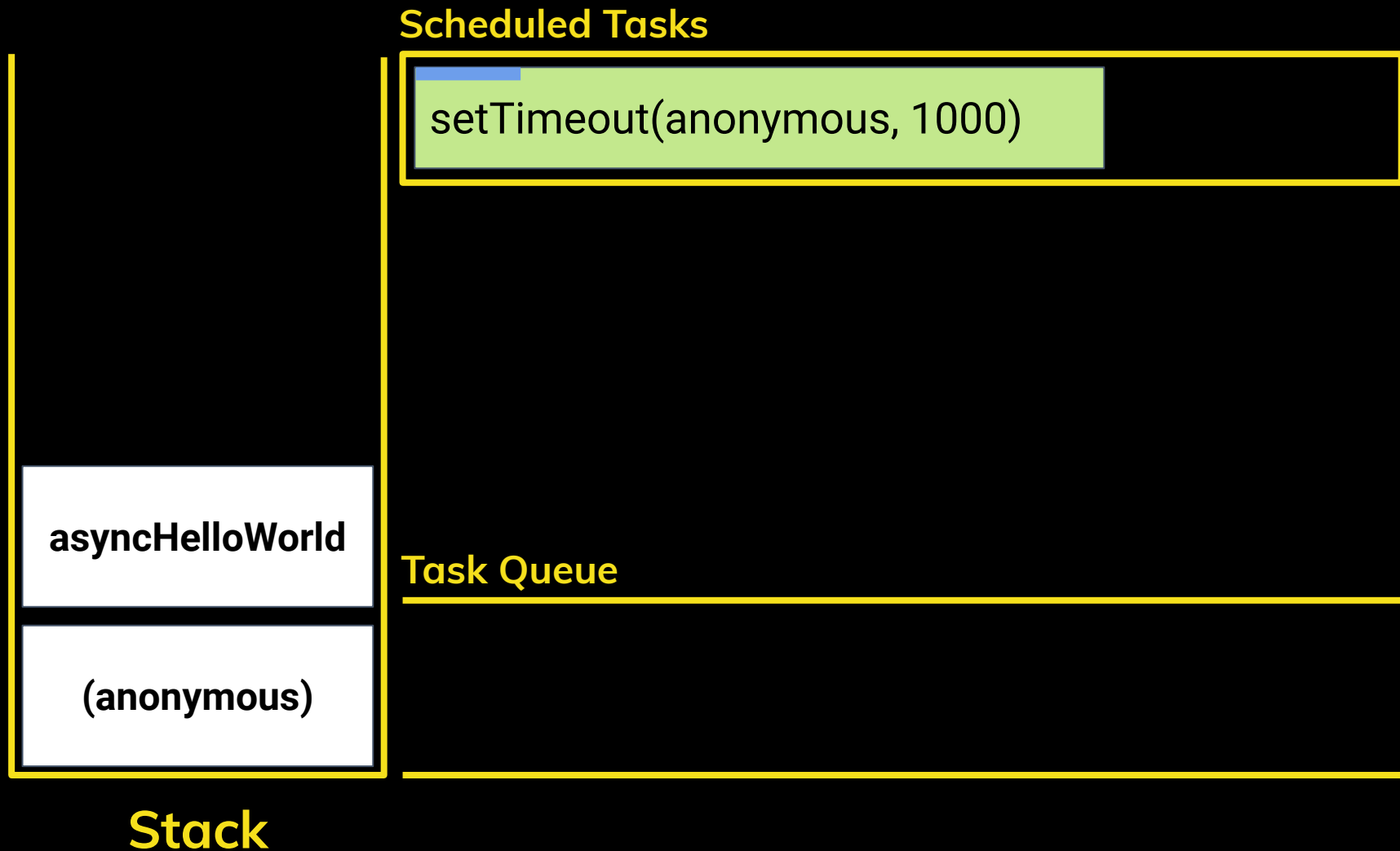
Queue

Queue

Task Queue



Task Queue



Task Queue

Scheduled Tasks

```
setTimeout(anonymous, 1000)
```

Task Queue

Stack

Task Queue

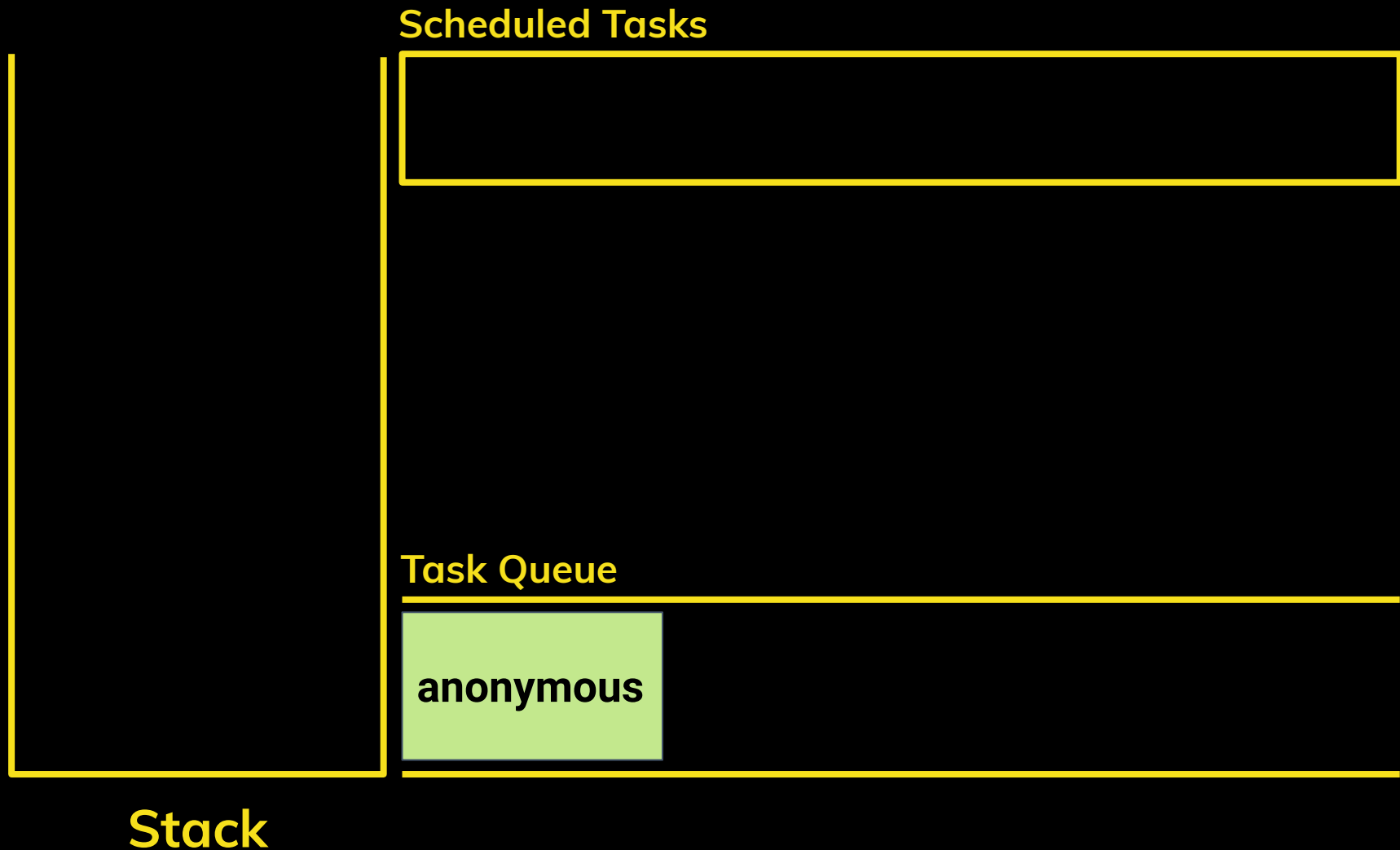
Scheduled Tasks

```
setTimeout(anonymous, 1000)
```

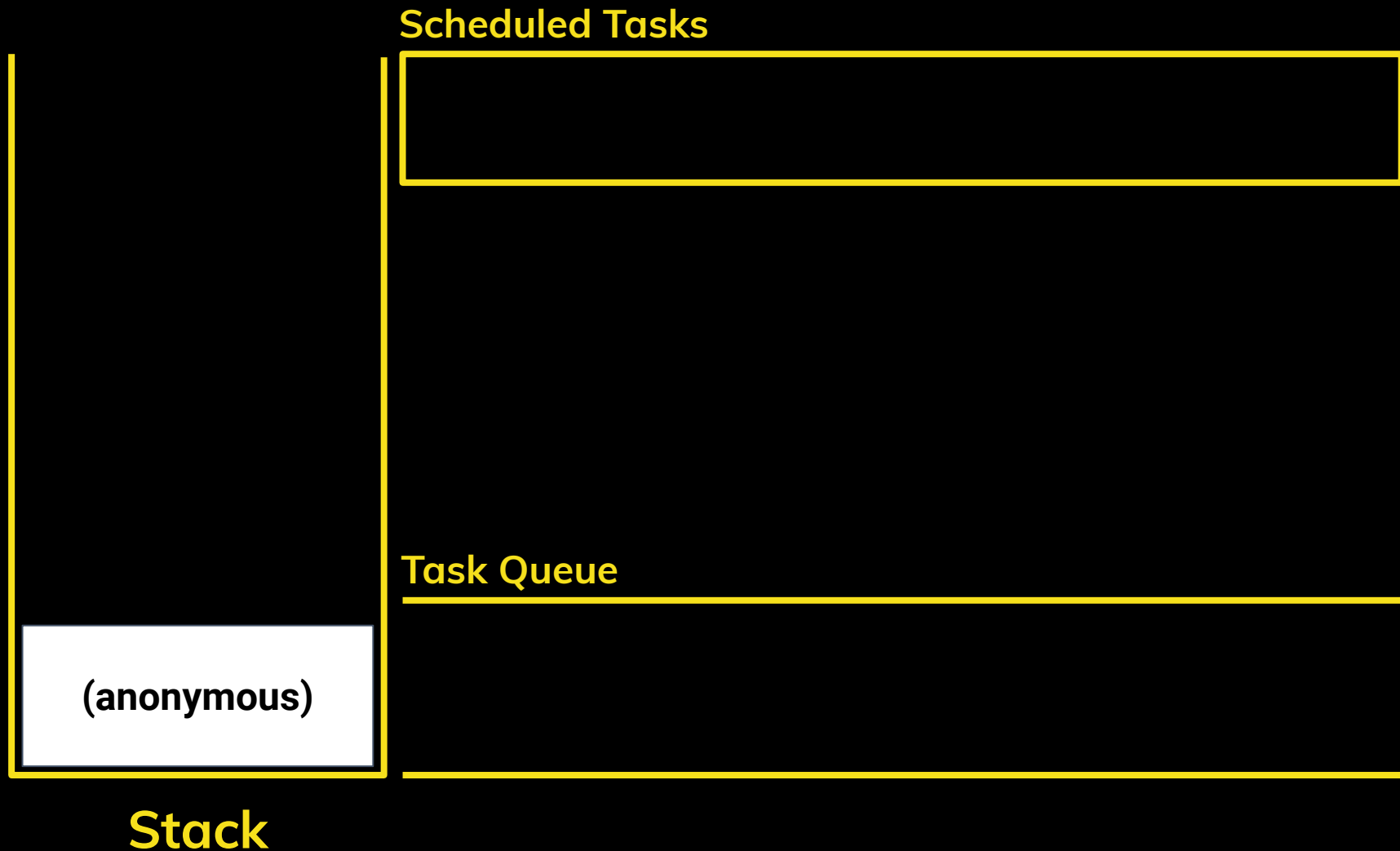
Task Queue

Stack

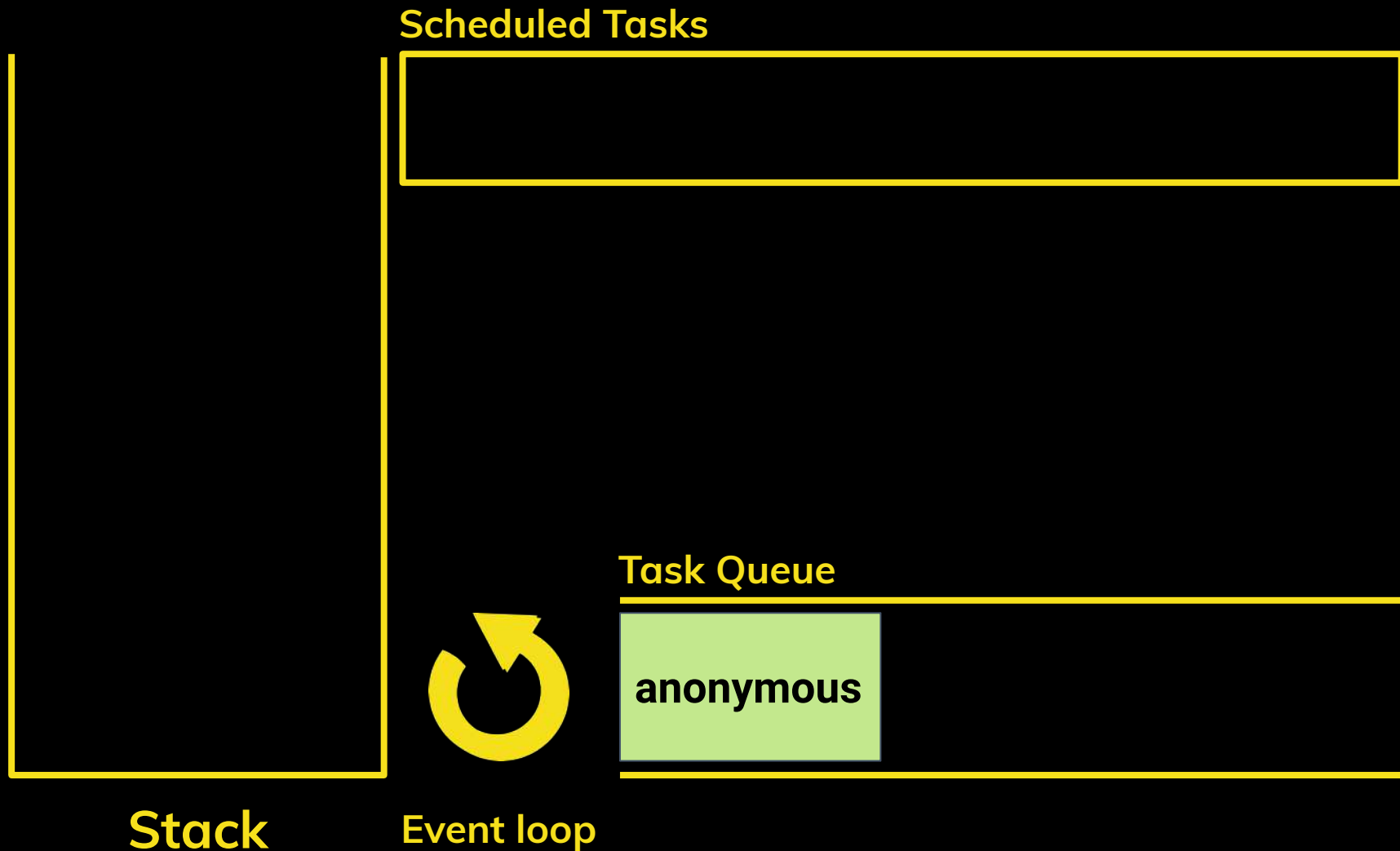
Task Queue



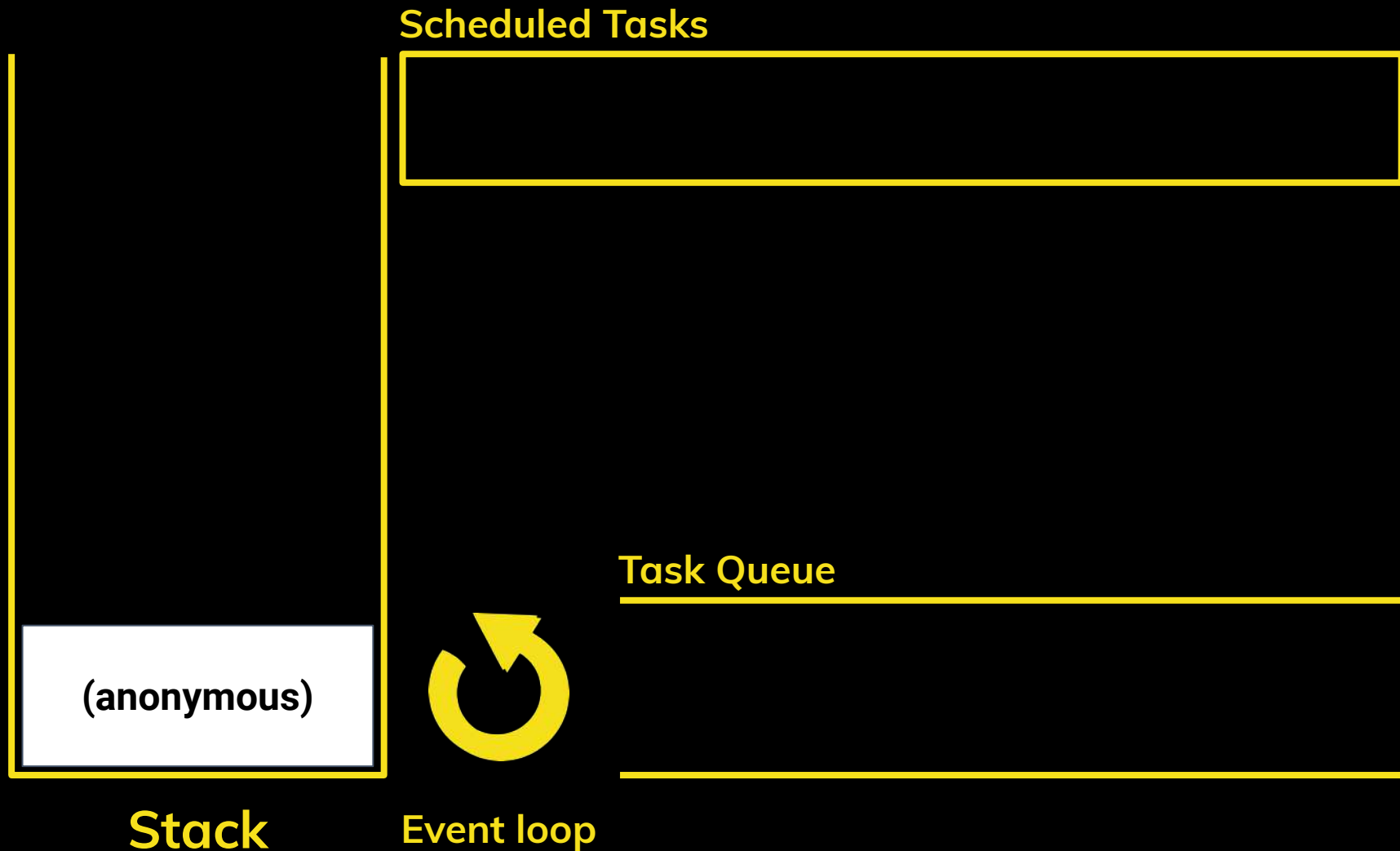
Task Queue



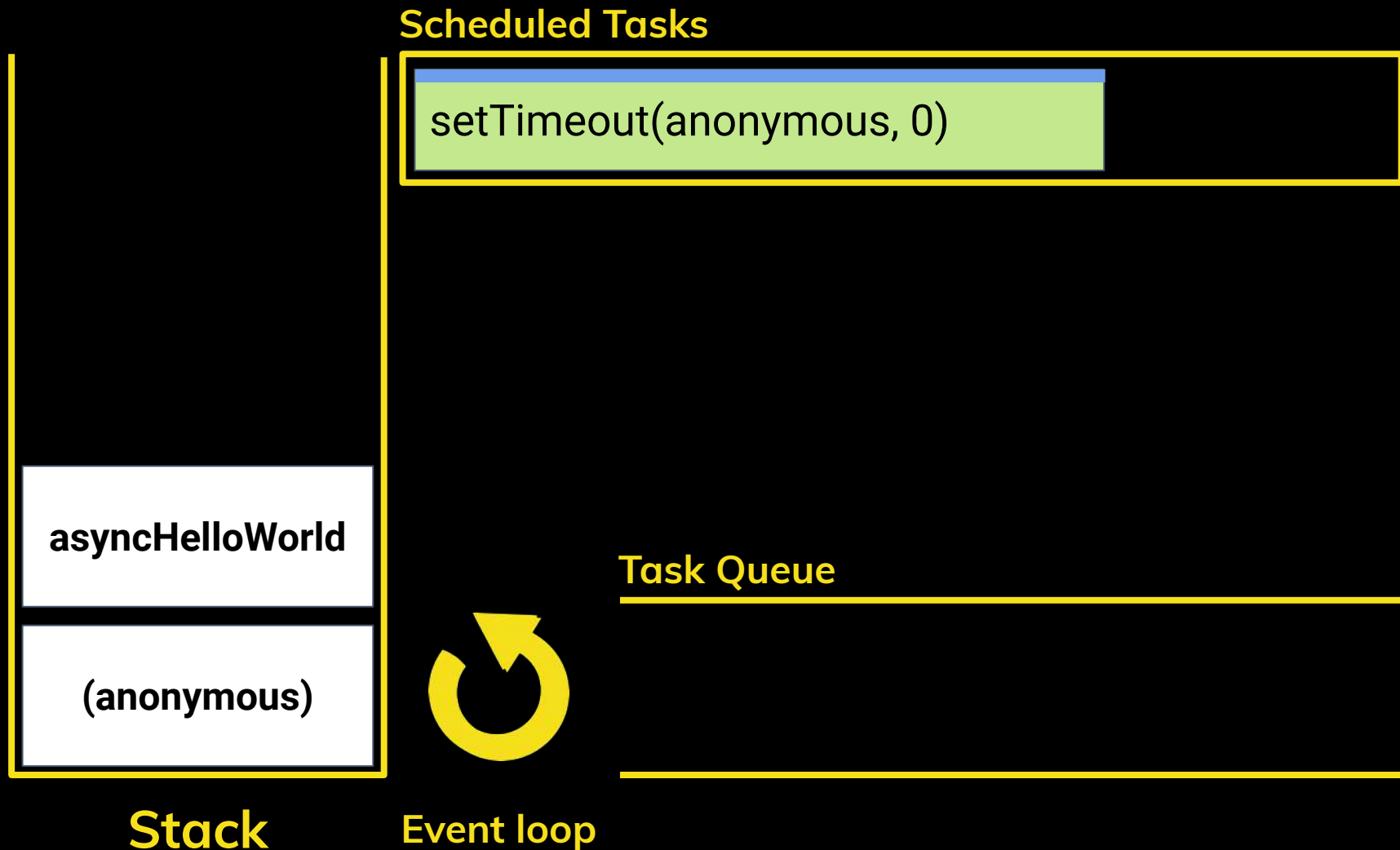
Event loop



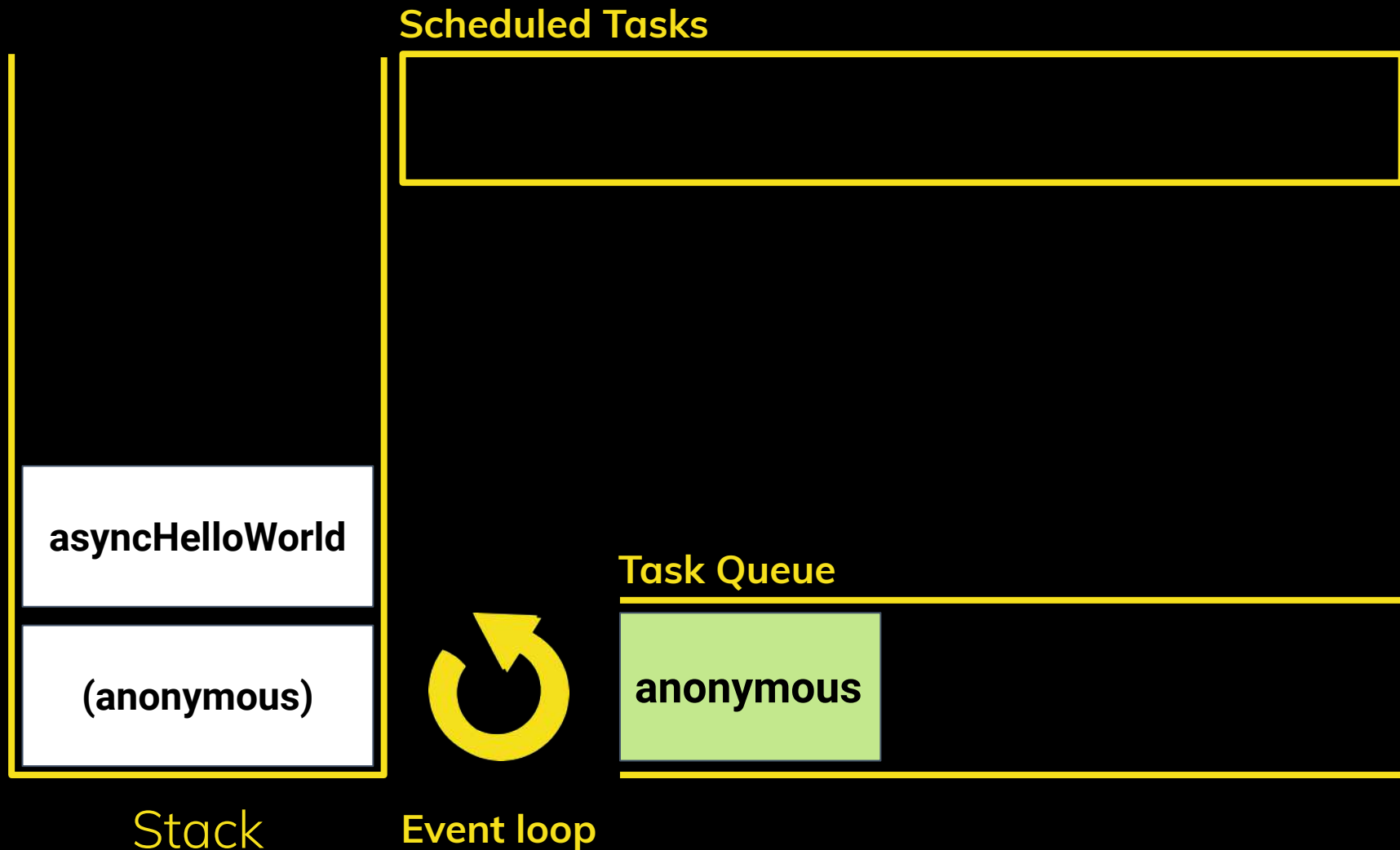
Event loop



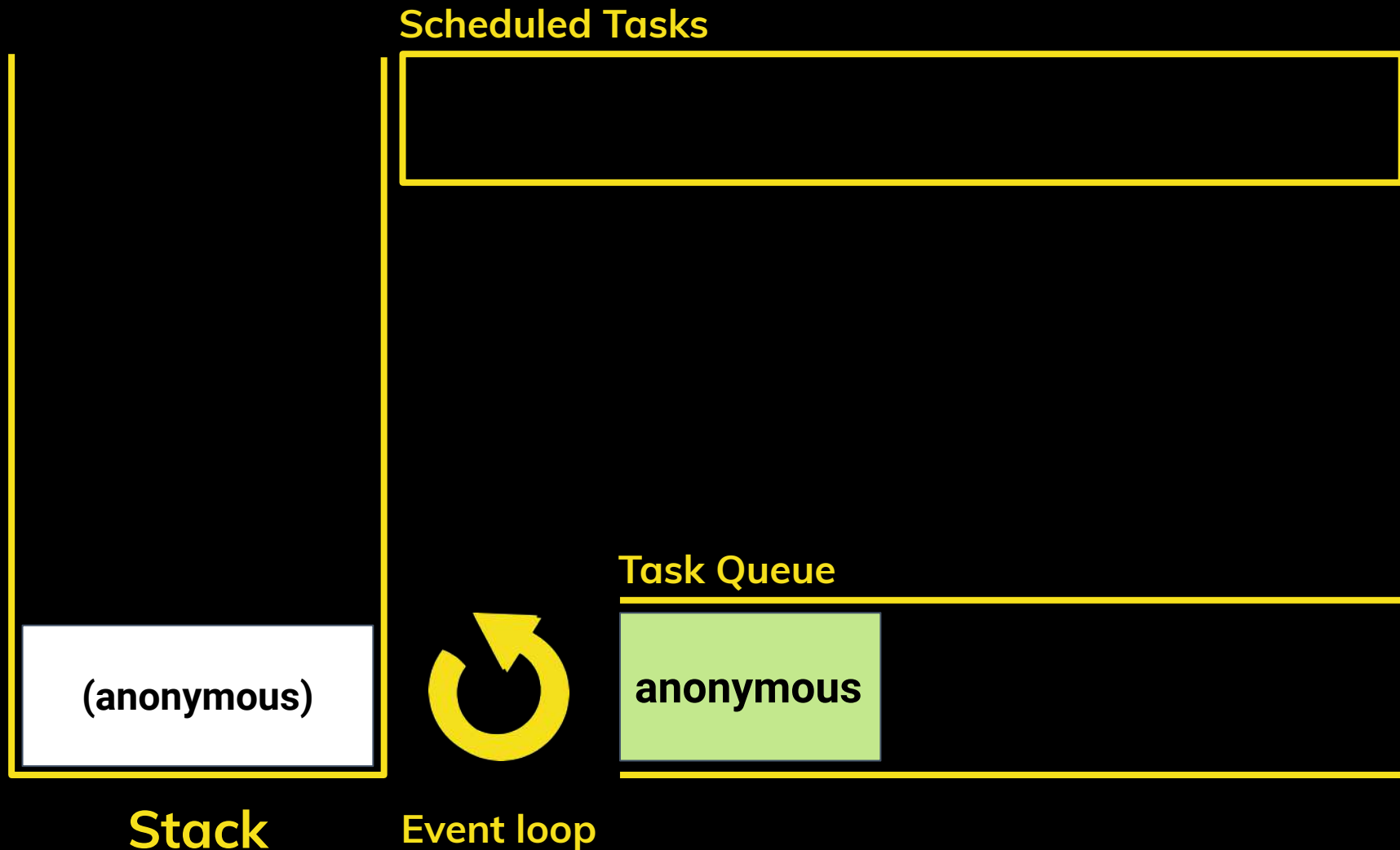
Event loop (ii)



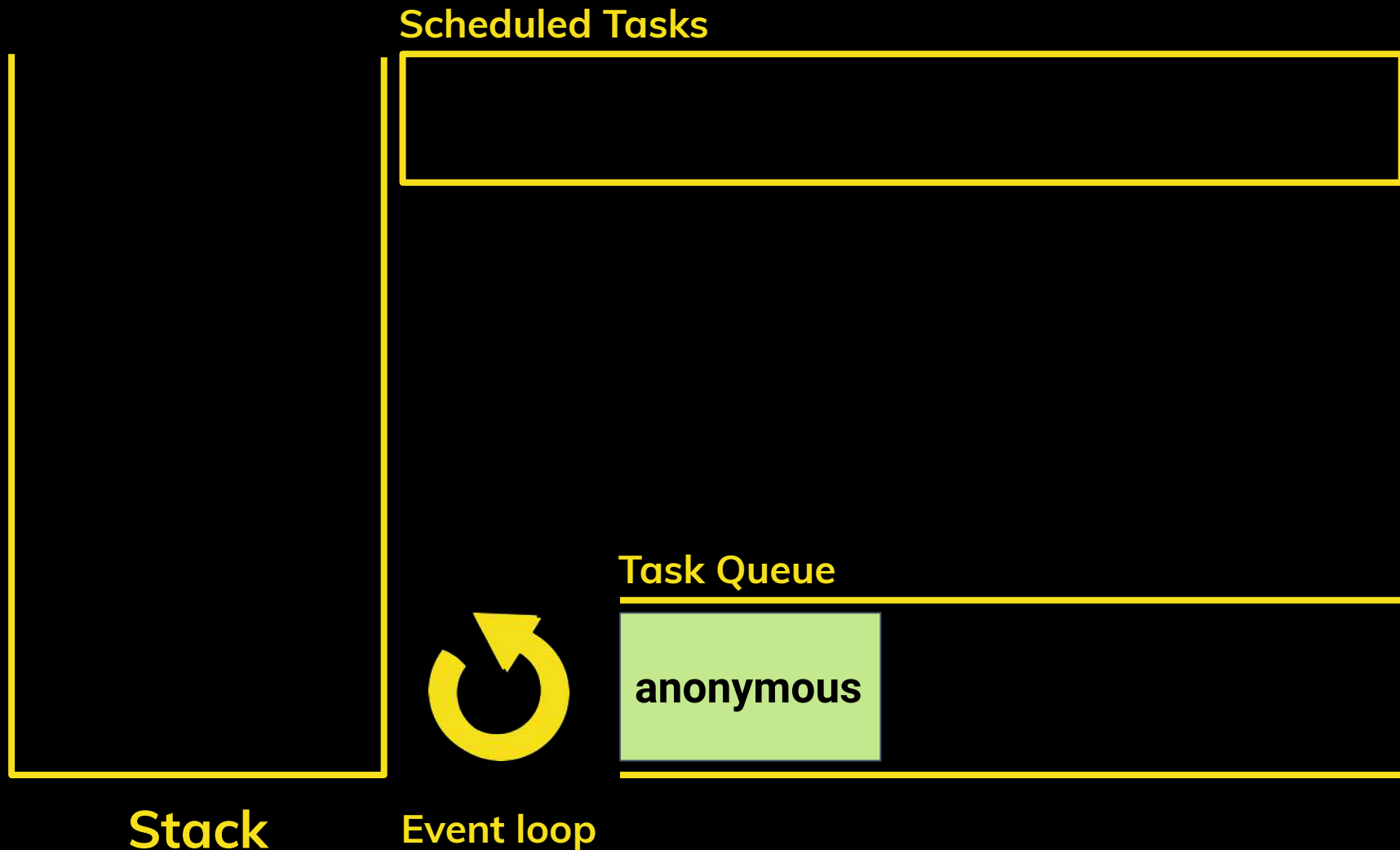
Event loop (ii)



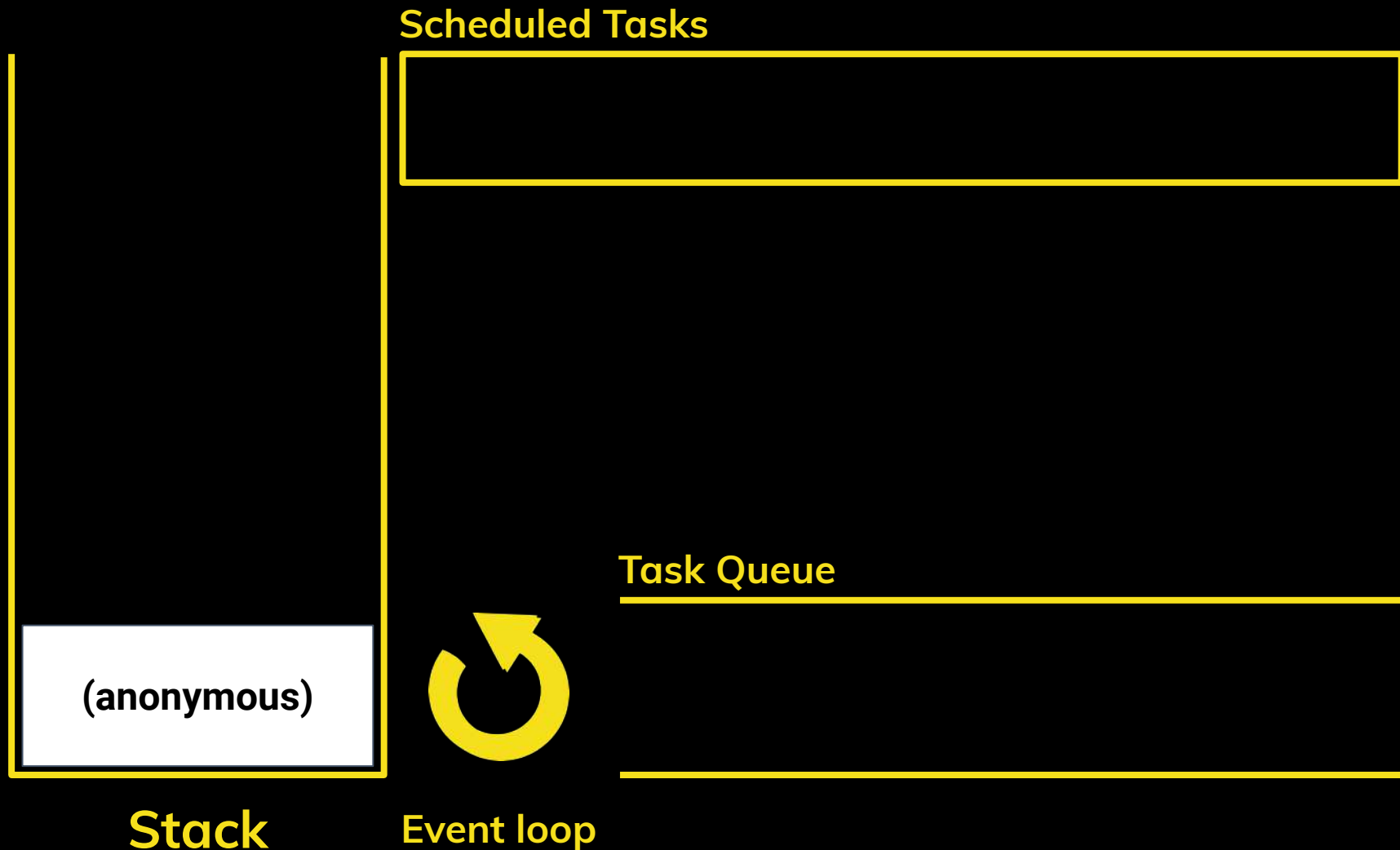
Event loop (ii)



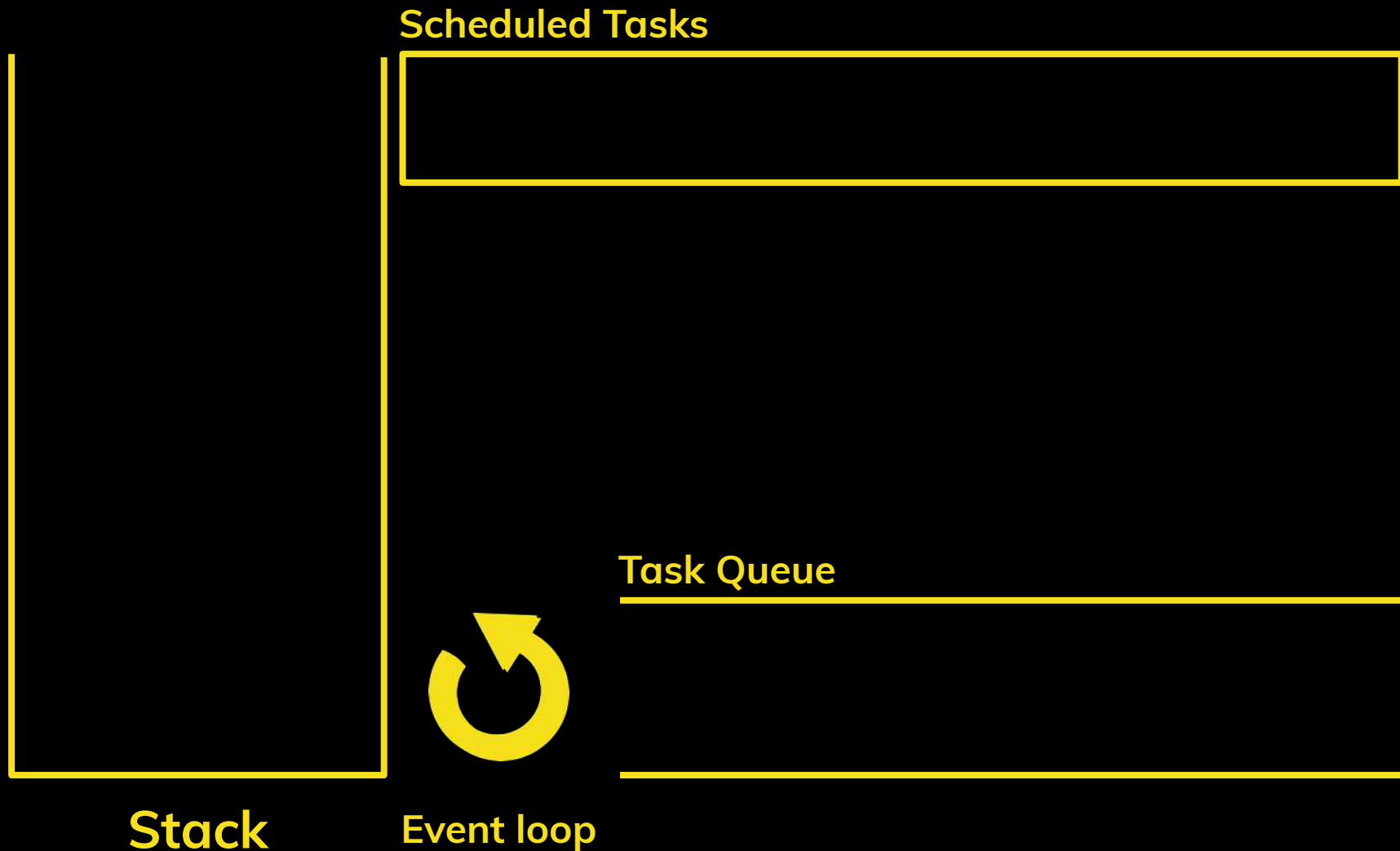
Event loop (ii)



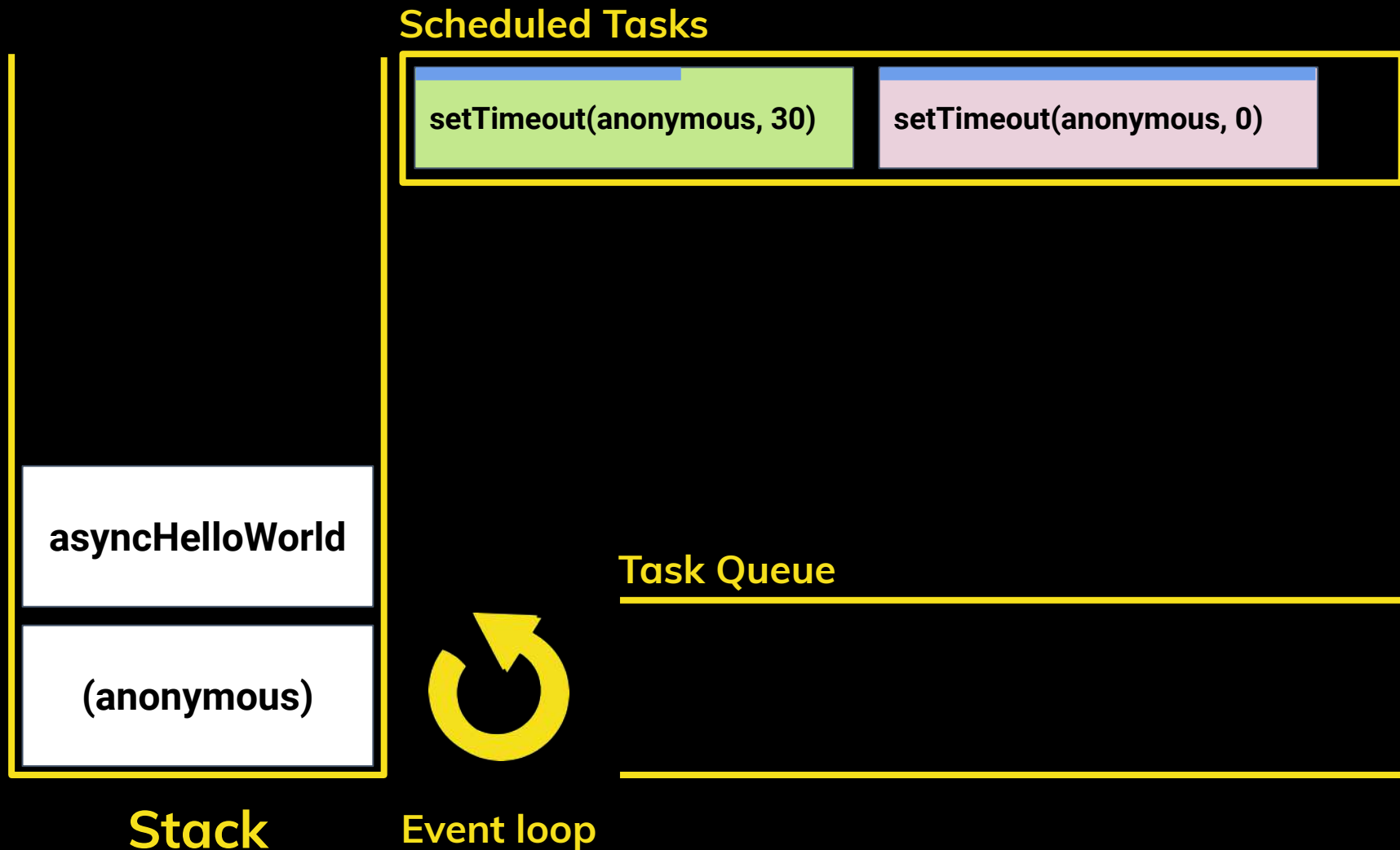
Event loop (ii)



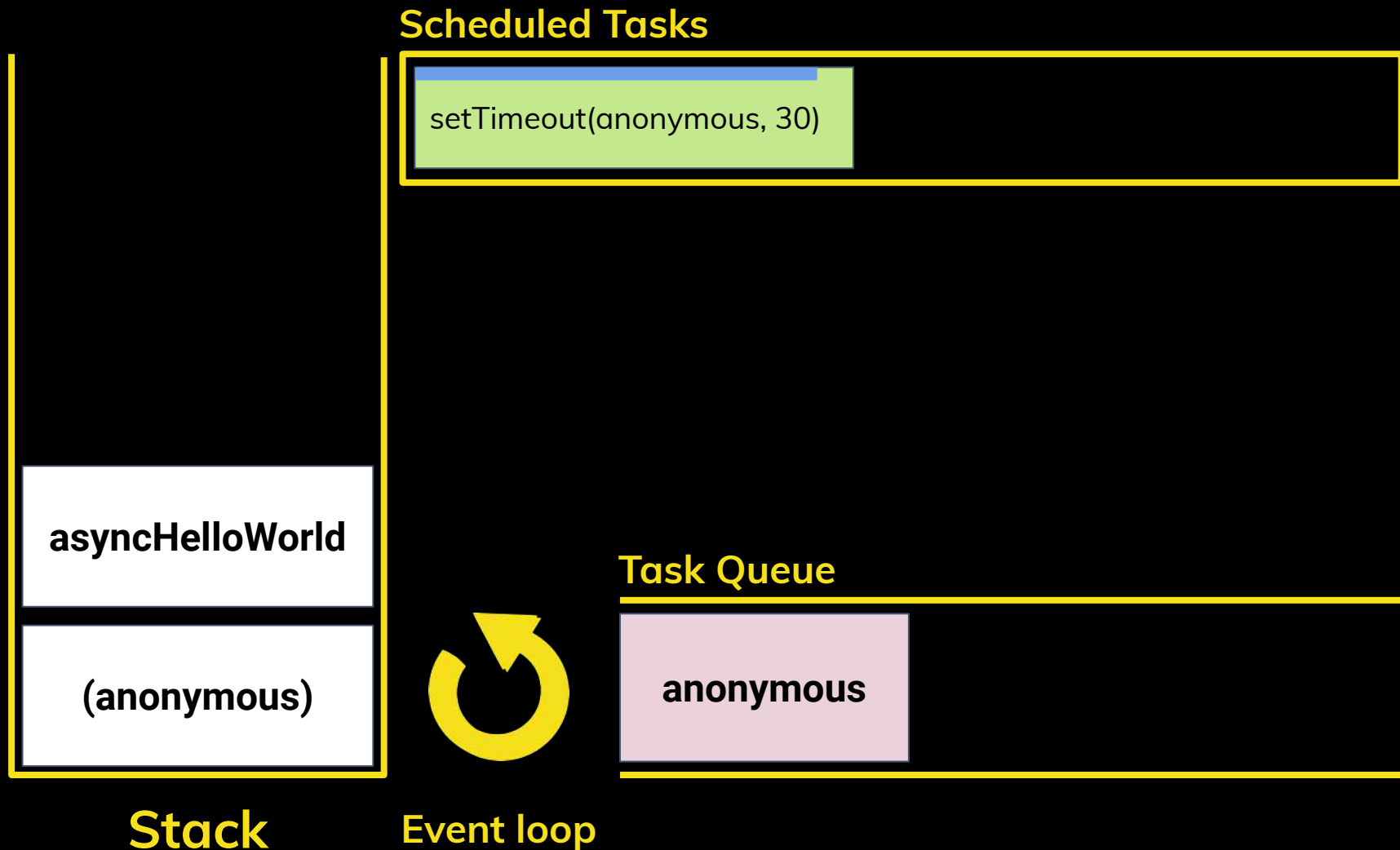
Event loop (ii)



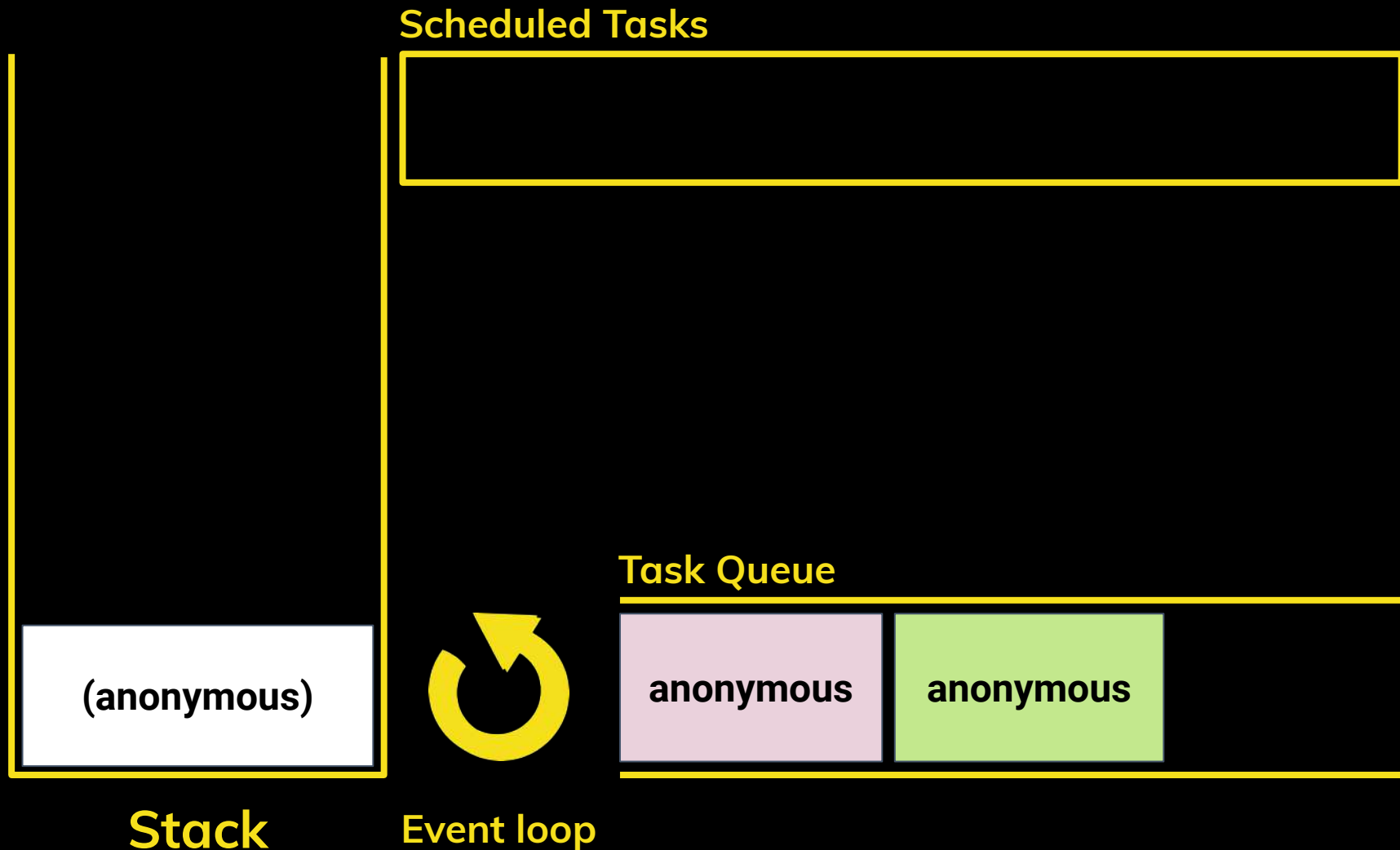
Event loop (iii)



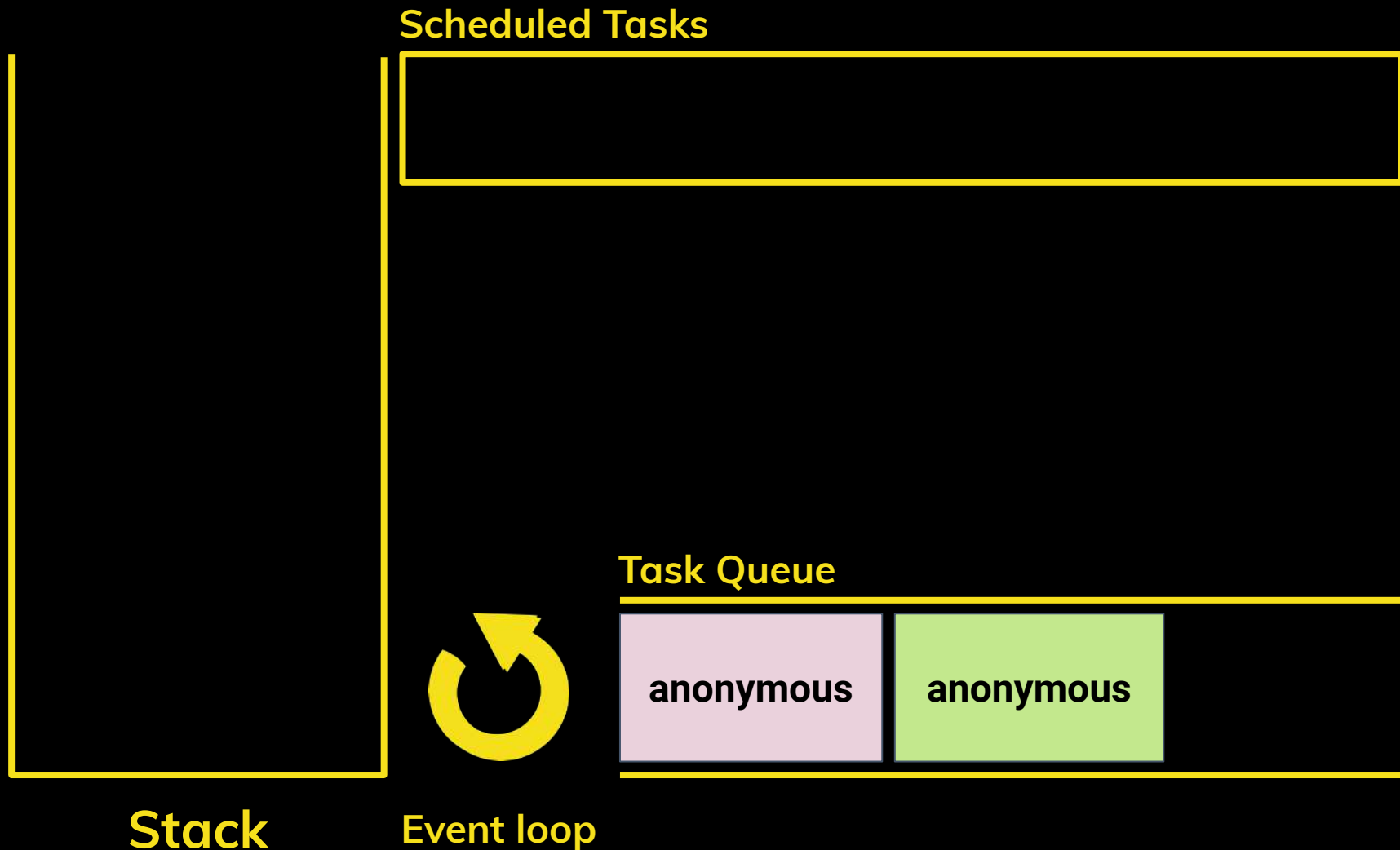
Event loop (iii)



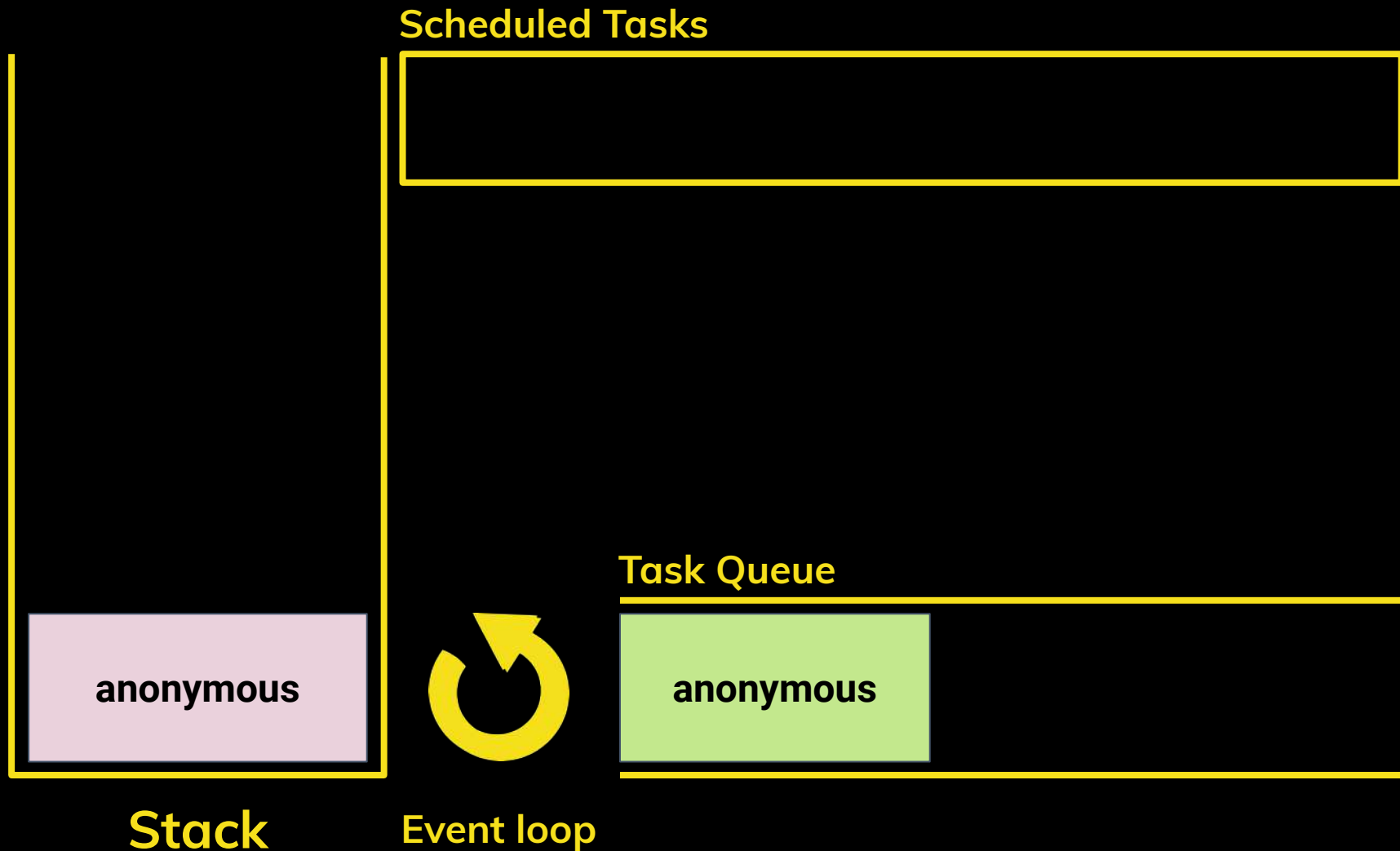
Event loop (iii)



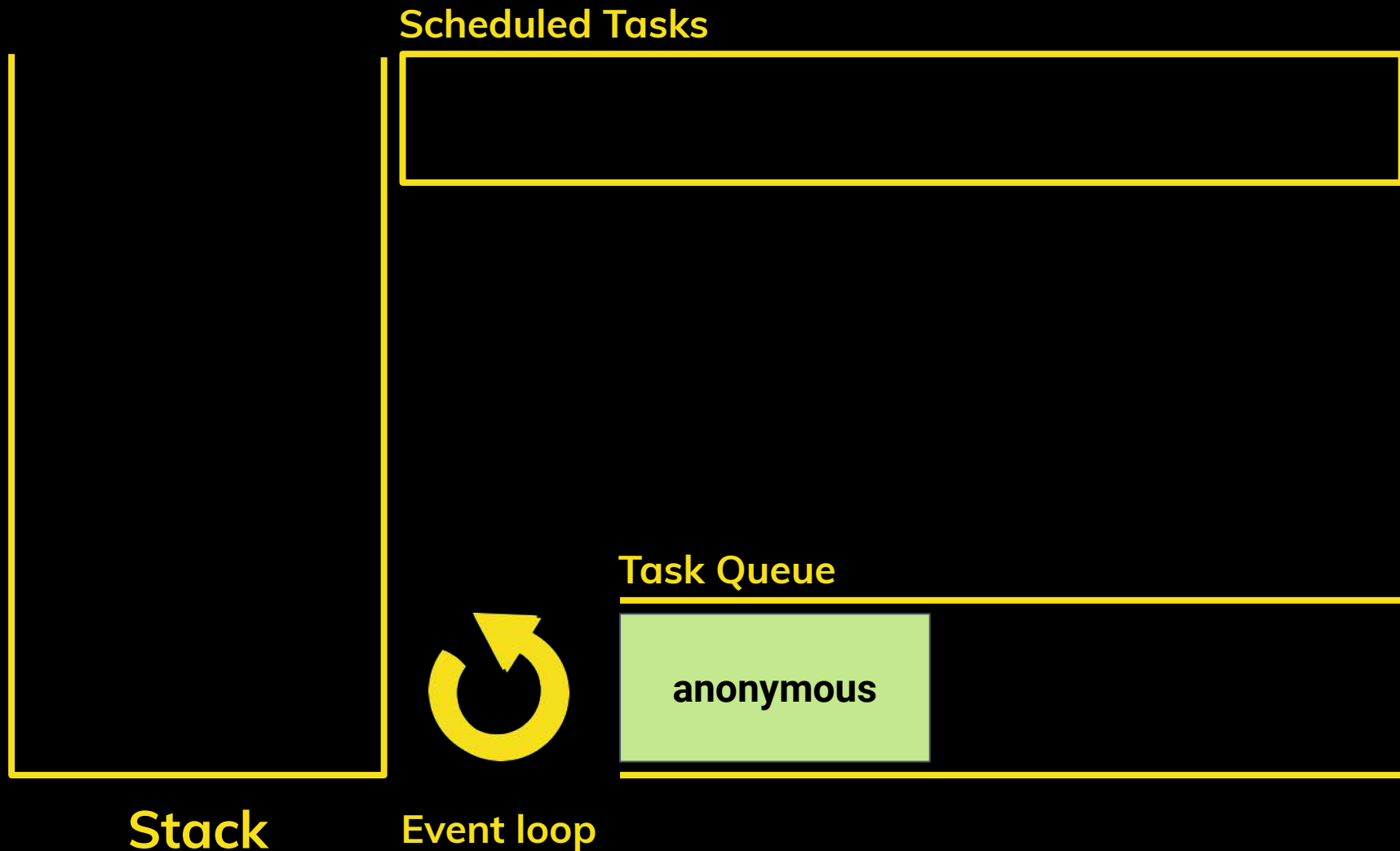
Event loop (iii)



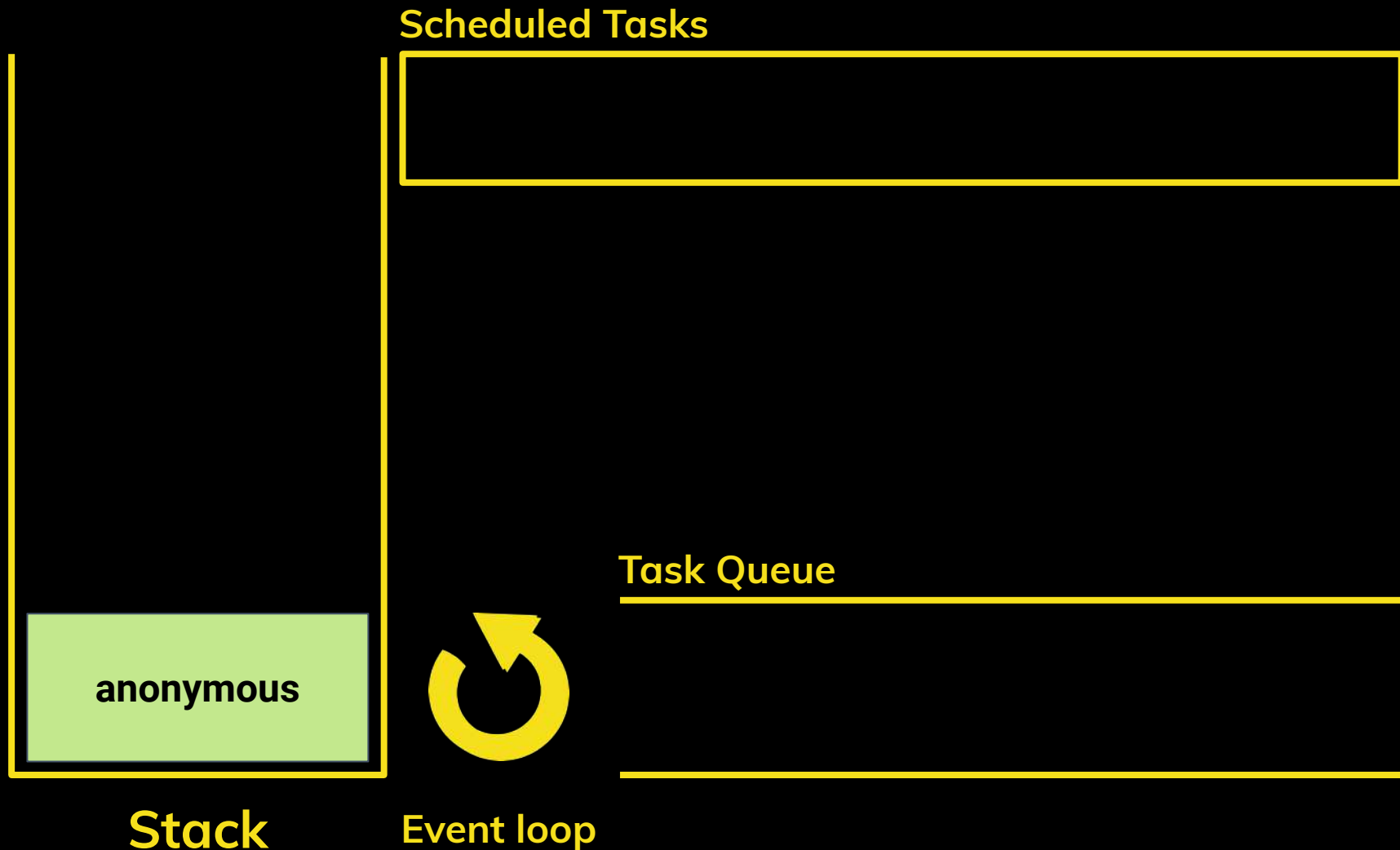
Event loop (iii)



Event loop (iii)



Event loop (iii)



Promesas

```
function moreAsync () {  
  console.log("Start");  
  setTimeout(() => console.log("setTimeout"), 0);  
  Promise.resolve("Promise 1").then(msg => console.log(msg))  
  Promise.resolve("Promise 2").then(msg => console.log(msg))  
  console.log("End")  
}  
  
moreAsync();
```

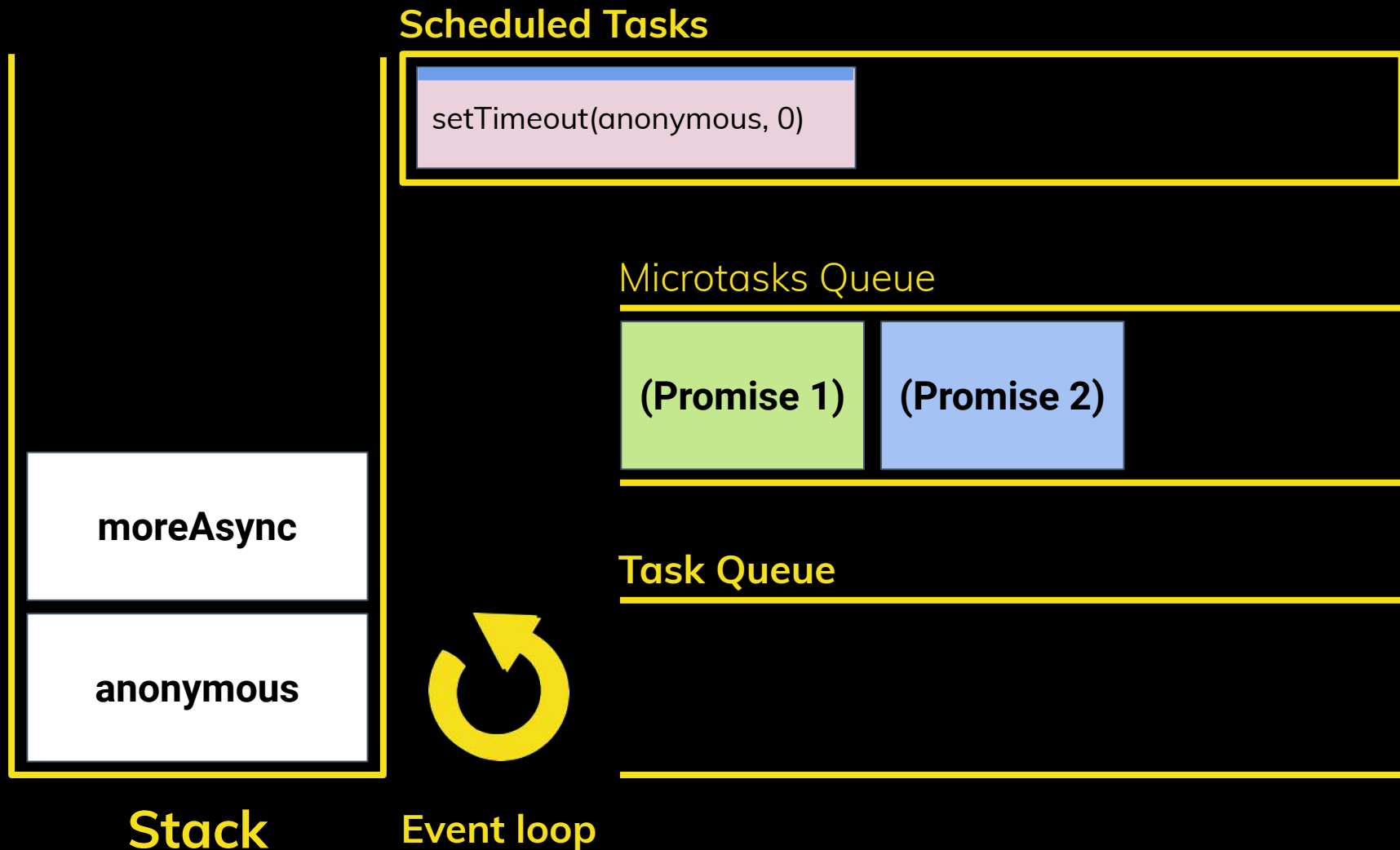
Promesas

```
function moreAsync () {  
  console.log("Start");  
  setTimeout(() => console.log("setTimeout"), 0);  
  Promise.resolve("Promise 1").then(msg => console.log(msg))  
  Promise.resolve("Promise 2").then(msg => console.log(msg))  
  console.log("End")  
}  
  
moreAsync();
```

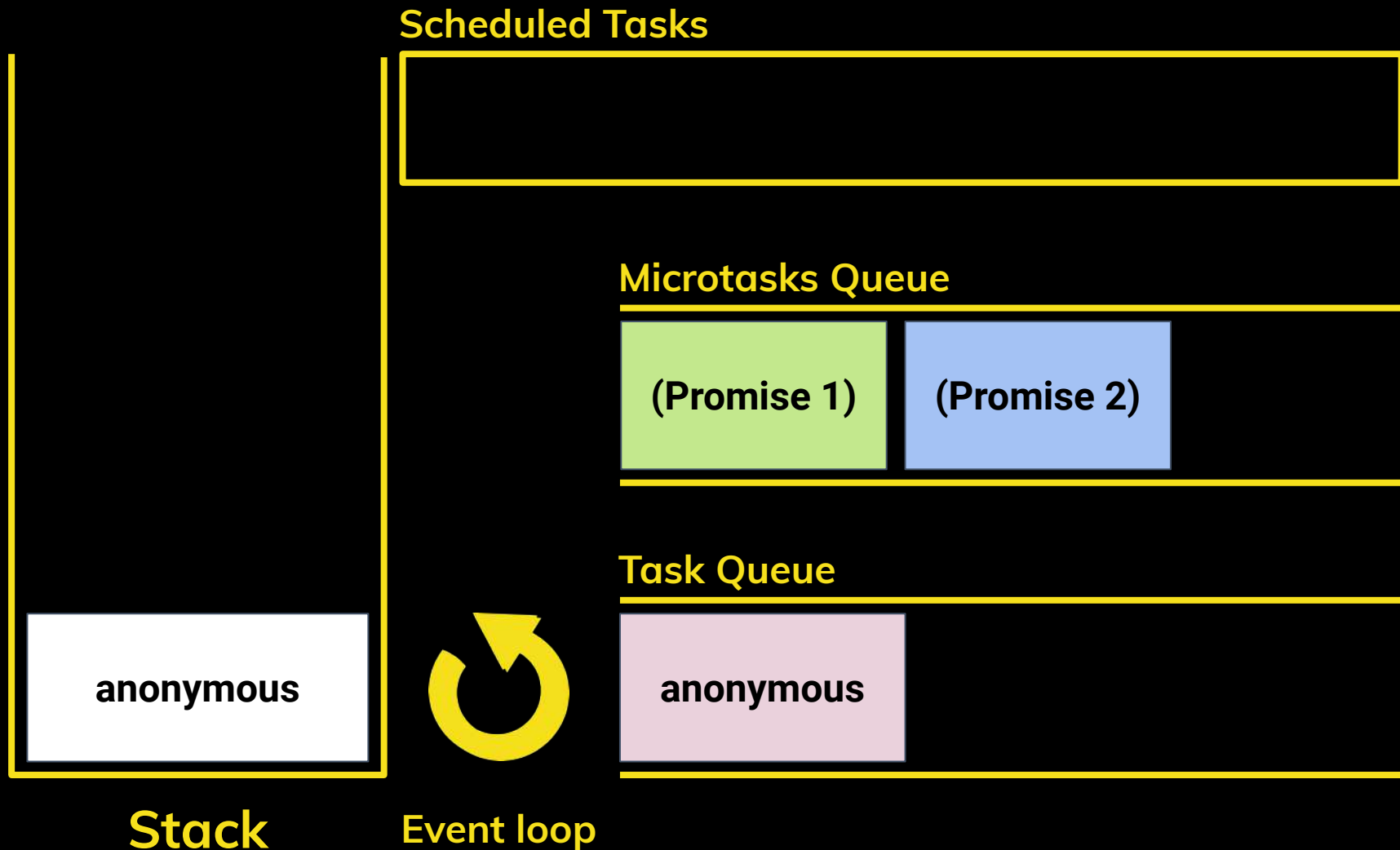
Output:

```
> Start  
> End  
> Promise 1  
> Promise 2  
> setTimeout
```

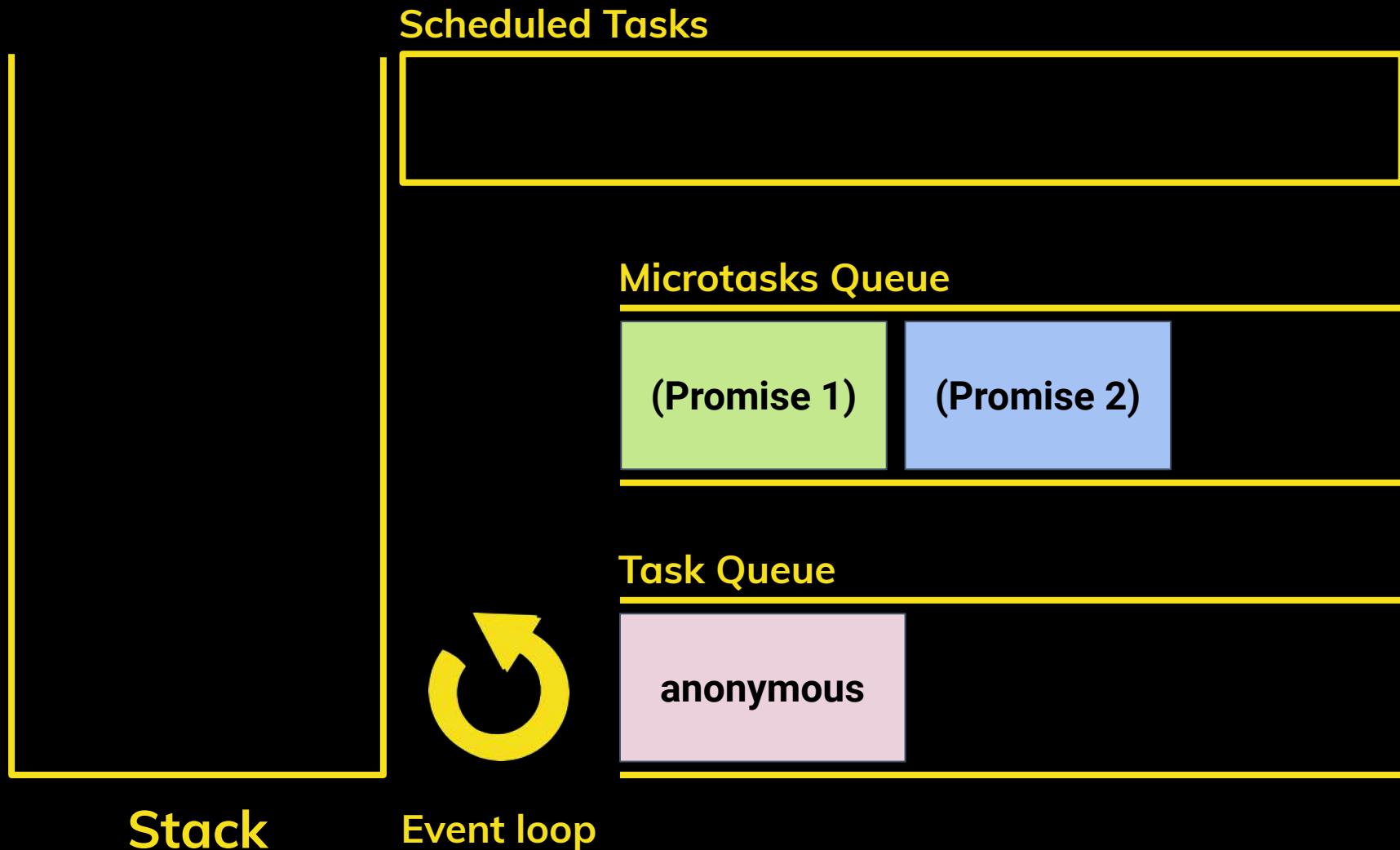
Event loop (iv)



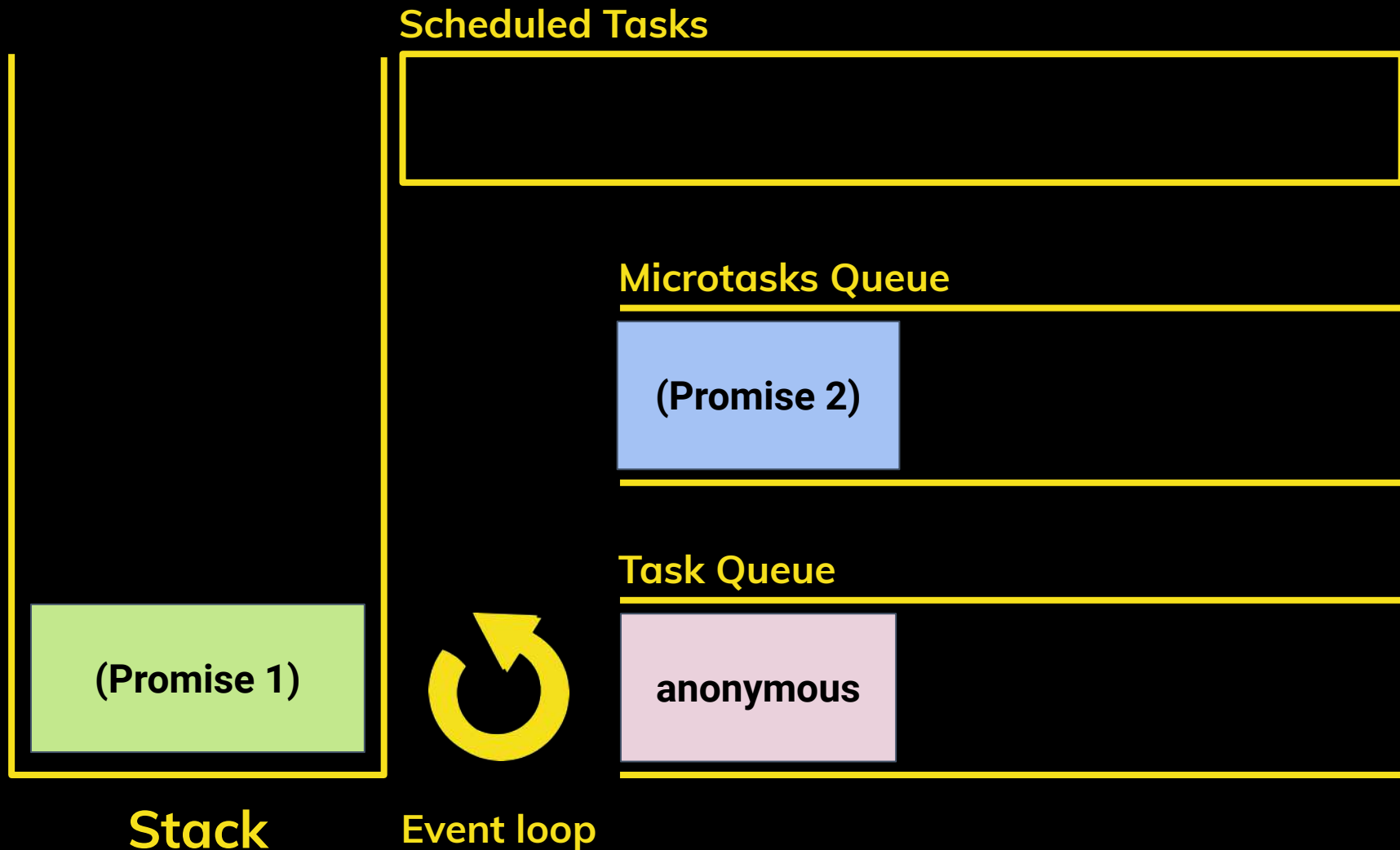
Event loop (iv)



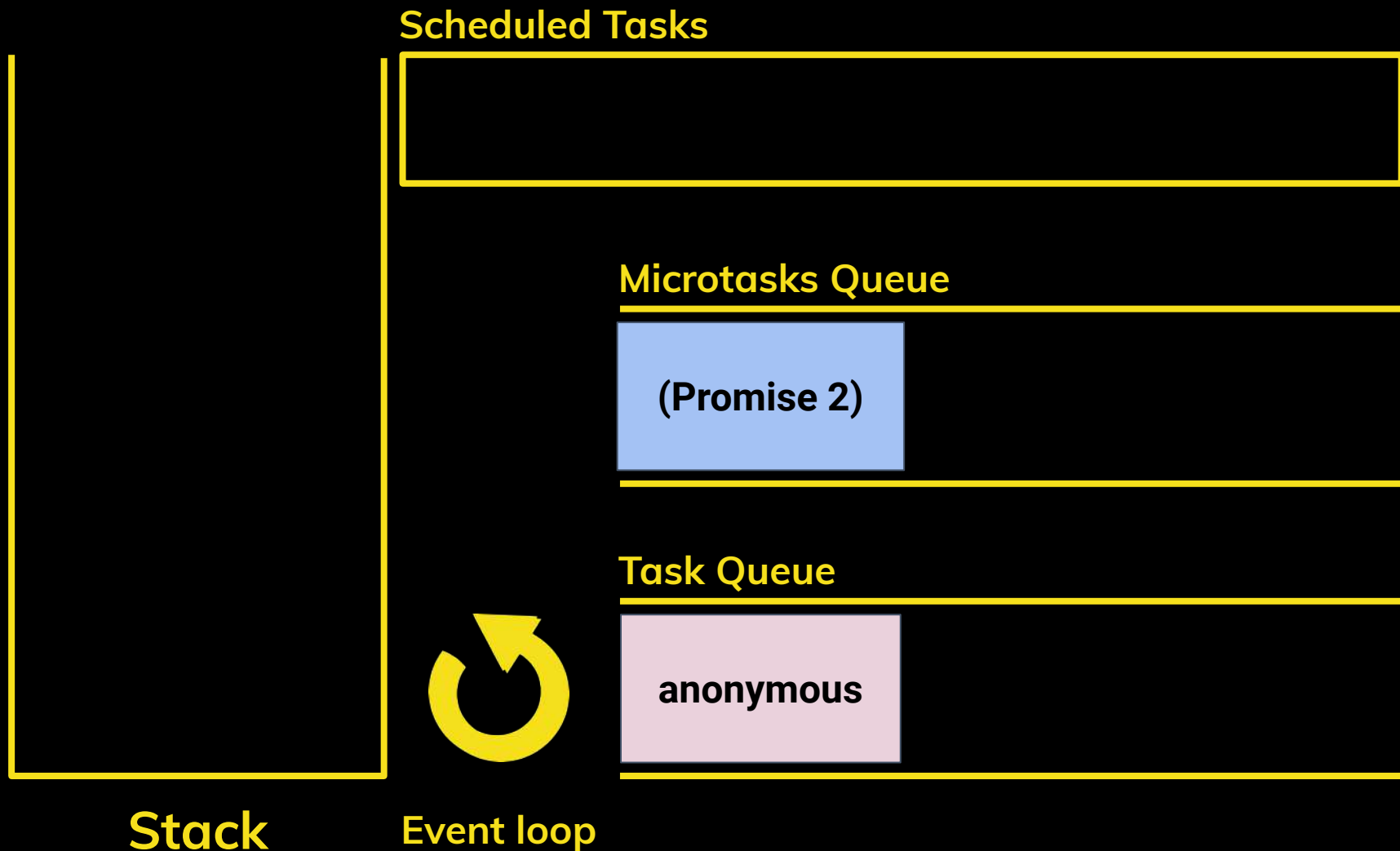
Event loop (iv)



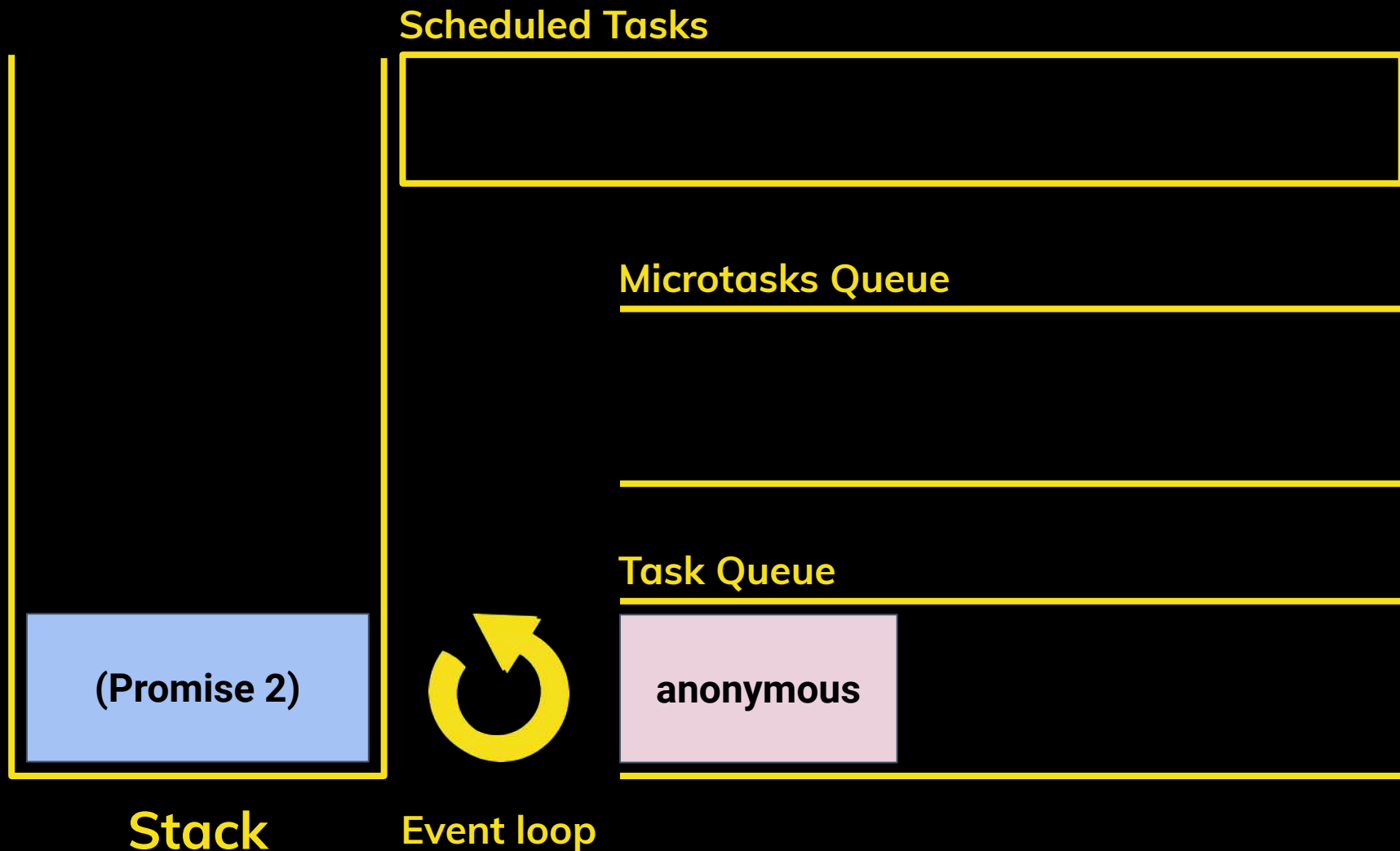
Event loop (iv)



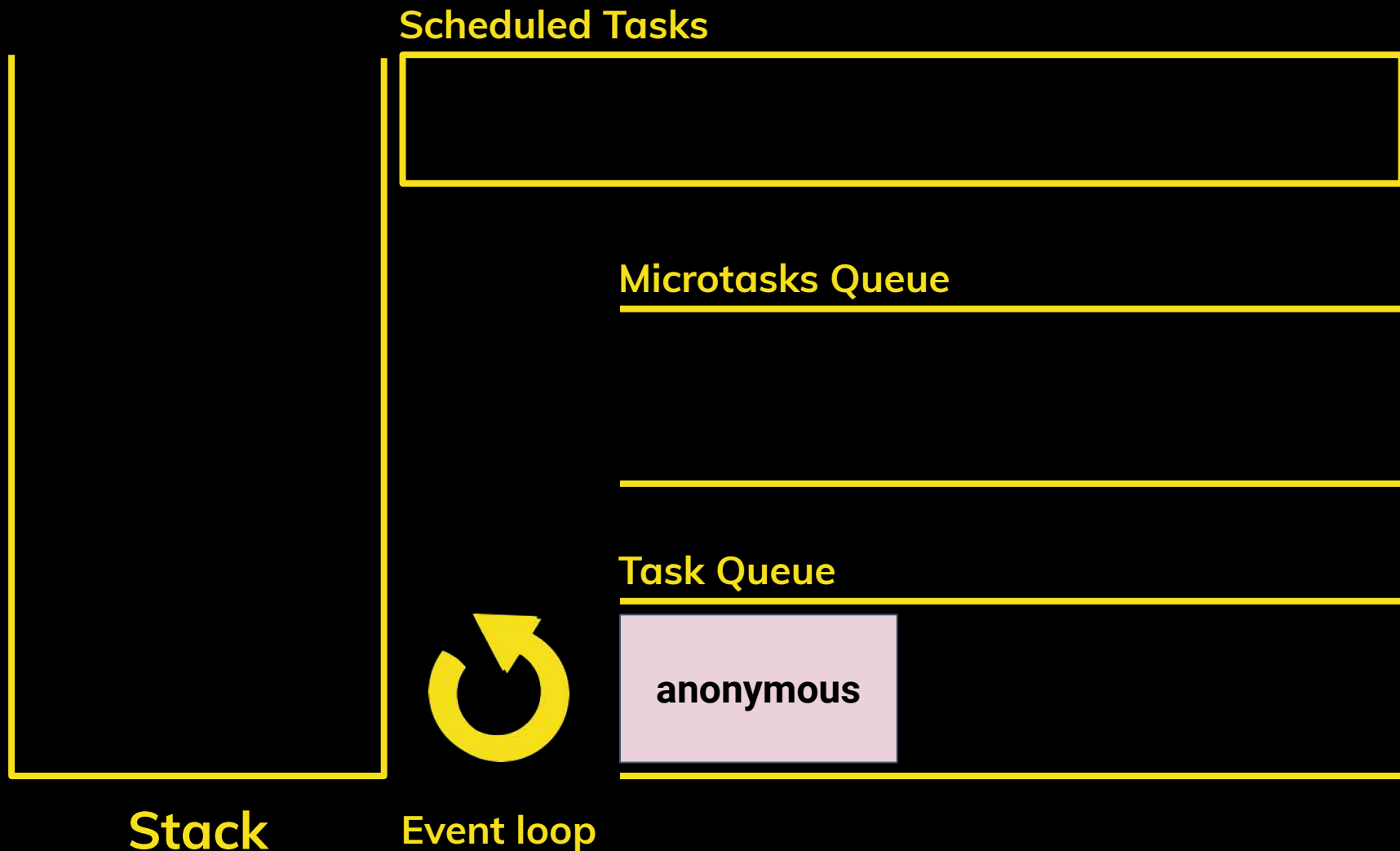
Event loop (iv)



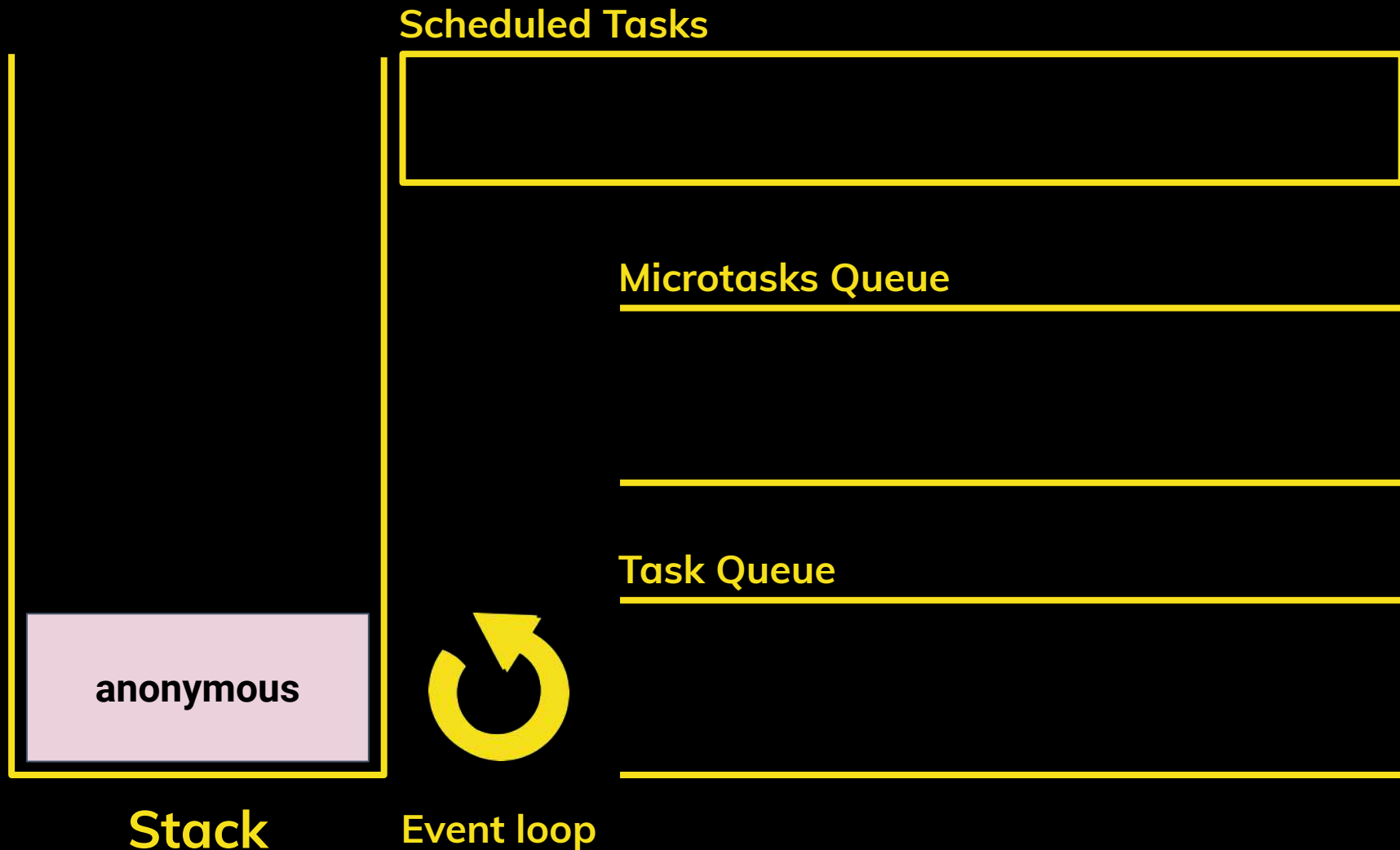
Event loop (iv)



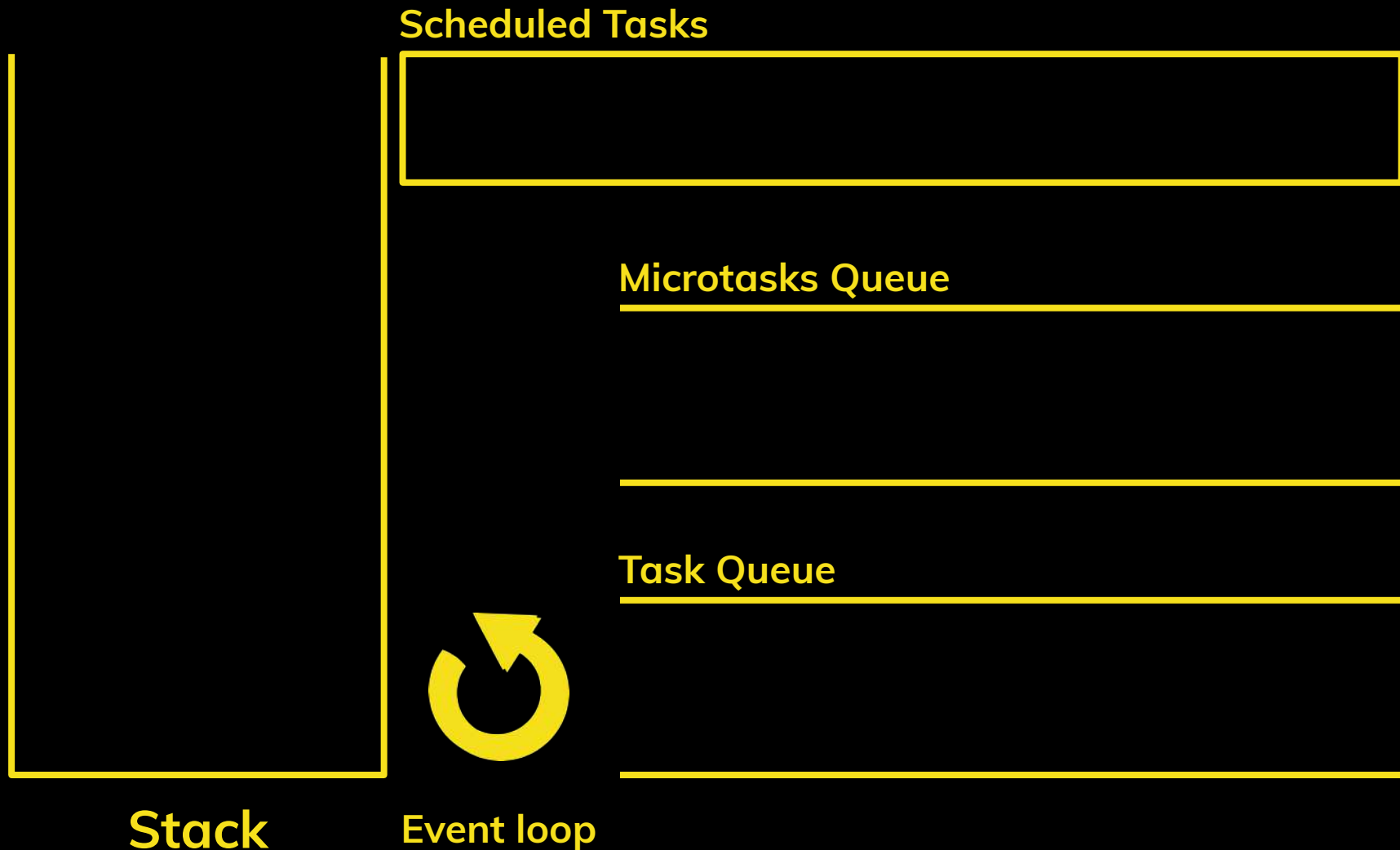
Event loop (iv)



Event loop (iv)



Event loop (iv)



**El event loop es lo que
hace que JavaScript
parezca ser multihilo**

Patrones de Diseño - Introducción

Qué es un
patrón de diseño

**Una solución para
un problema dentro
de un contexto.**





Una solución para un problema dentro de un **contexto**.

- El contexto es la situación donde el patrón aplica.
- Esta situación debe ser recurrente.
- **Ejemplo:** Muchos objetos están interesados en el estado de otro objeto.



Una solución para un **problema** dentro de un contexto.

- Se refiere a la meta que estás tratando de alcanzar dentro del contexto.
- El problema incluye todas las limitaciones que existen dentro de este contexto.
- **Ejemplo:** Estos objetos quieren recibir notificaciones cuando cambie el estado, sin tener que solicitar la información.



Una **solución** para un problema dentro de un contexto.

- Un diseño genérico que alcanza la meta dentro del contexto.
- Ejemplo: Crear una clase donde cualquier objeto se puede suscribir y desuscribir a cambios en el estado.

Una **solución** para un problema dentro de un contexto.

- Un diseño genérico que alcanza la meta dentro del contexto.
- Ejemplo: Crear una clase donde cualquier objeto se puede suscribir y desuscribir a cambios en el estado.

Observer Pattern



Ejemplos



Every Layout

¿Cómo acomodar elementos?



RUDIMENTS

Boxes

Composition

Units

Global and local styling

Modular scale

Axioms

LAYOUTS

every-layout.dev/layouts/stack

► The Stack

The Stack

Nombre

Contexto

TABLE OF CONTENTS

- The problem
- The solution
- Use cases
- The generator
- The component

Problema

Patrón de diseño

Ejemplos

The problem

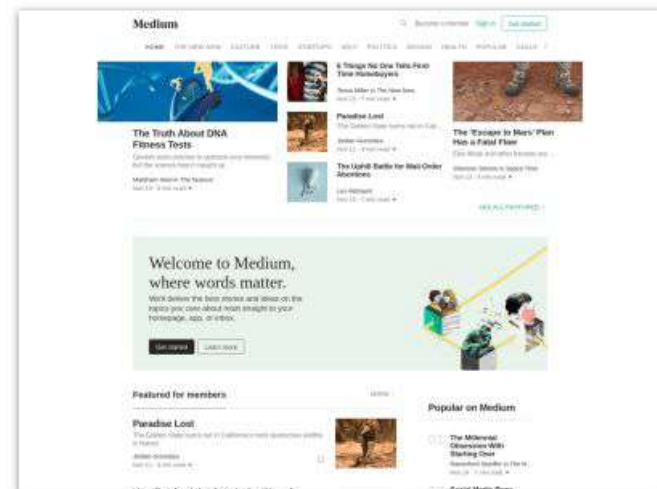
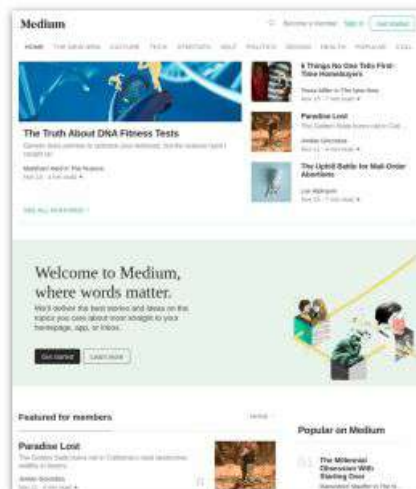
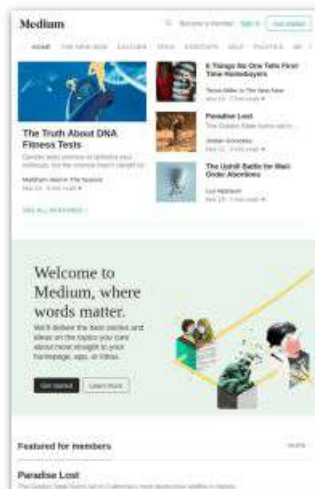
Flow elements require space (sometimes referred to as *white space*) to physically and conceptually separate them from the elements that come before and after them. This is the purpose of the `margin` property.

Media Queries

¿Cómo acomodar la información para diferentes tamaños de pantalla?

Media
Queries

Medium



mediaqueri.es

Patrones de React

React Patterns [on GitHub](#)

Contents

- Function component
- Destructuring props
- JSX spread attributes
- Merge destructured props with other values
- Conditional rendering
- Children types
- Array as children
- Function as children
- Render prop
- Children pass-through
- Proxy component
- Style component
- Event switch
- Layout component
- Container component
- Higher-order component
- State hoisting
- Controlled input

Function component

[Function components](#) are the simplest way to declare reusable components.

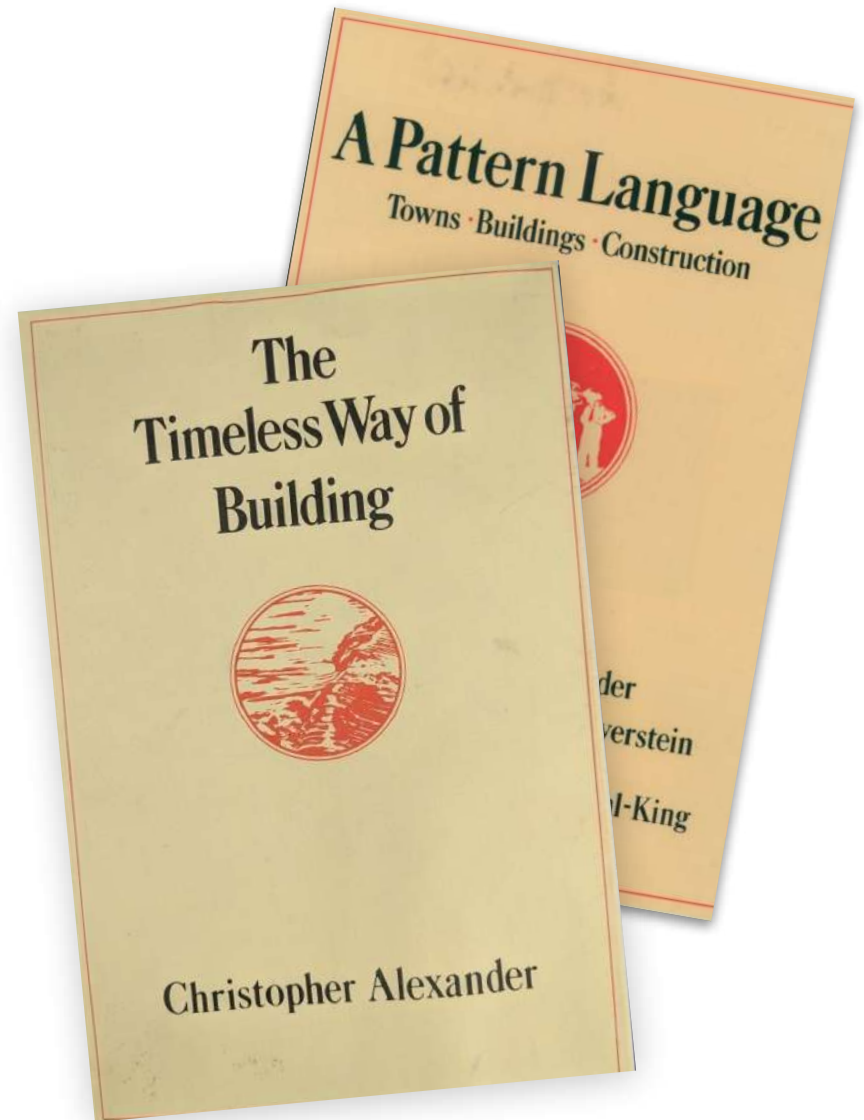
They're just functions.

```
function Greeting() {  
  return <div>Hi there!</div>;  
}
```

Un poco de historia

La idea de los patrones comienza en la arquitectura con **Christopher Alexander**.

Sus libros describen patrones para construir arquitectura dinámica, como casas, pueblos y ciudades.

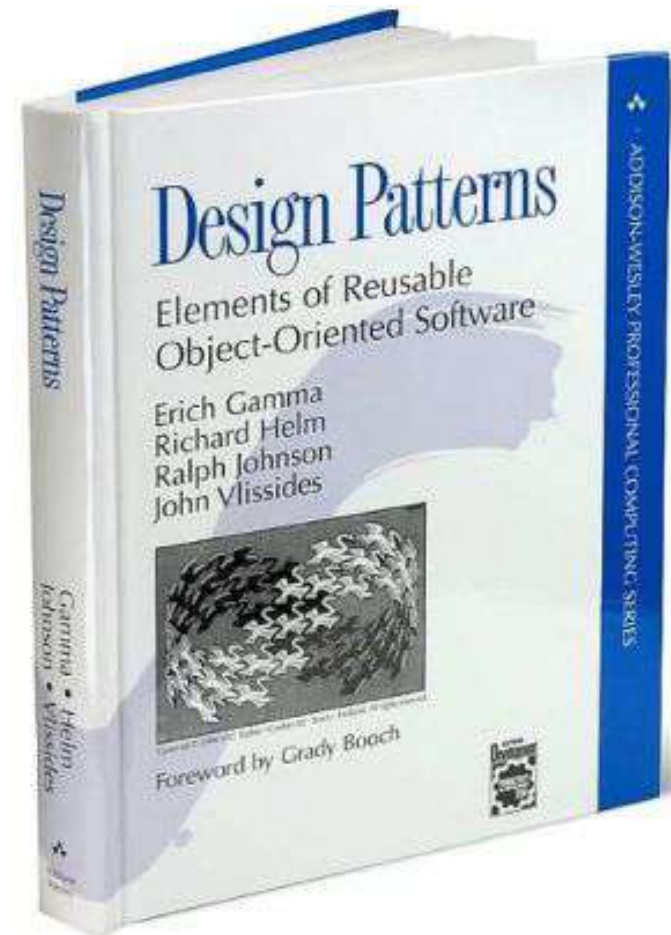


Un poco de historia

El libro que comenzó el campo de patrones de diseño de software.

Publicado en 1995.

Describe los patrones fundamentales.



Beneficios

- Los patrones de diseño son una caja de herramientas de soluciones bien probadas a problemas comunes en diseño de software.
- Te proveen un lenguaje común que permite comunicarse de forma específica y eficiente.

Crítica

- Los patrones de diseño son una forma de complejidad.
- Son soluciones a las limitaciones de un lenguaje de programación. Ejemplo: Java o C# no tienen funciones de alto nivel.
- “Cuando lo que tienes es un martillo, todo te parece un clavo.”

“

Patterns, like all forms of complexity, should be avoided until they are absolutely necessary

”

Jeff Atwood

blog.codinghorror.com/head-first-design-patterns

Conclusión

- Siempre busca pensar en términos de diseño, no de patrones.
- Usa un patrón cuando hay una necesidad natural para usarlos.
- Si existe una alternativa más simple, prefierela.



Patrones de Diseño -
Introducción

Categorías de patrones de diseño

Categorías



Patrones
creacionales



Patrones
estructurales



Patrones de
comportamiento



Patrones creacionales

Proveen diferentes mecanismos
para crear objetos.

Patrones creacionales

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton



Builder

(Constructor)

Es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente.

Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción puede servir para crear representaciones diferentes.



Builder

(Constructor)

Es usado para **permitir la creación de una variedad de objetos complejos desde un objeto fuente.**

Separa la creación de un objeto complejo de su estructura, de tal forma que el mismo proceso de construcción puede servir para crear representaciones diferentes.

```
// Cómo jQuery usa el Builder
```

```
$('<h1 class="super-big">My favorite Pokemons</h1>');
```

```
$('<button class="btn">Llama Ya</button>').click(() => {  
    console.log("Llamando...");  
});
```

```
$("<input />")  
    .attr({ "type": "email", "id": "email"})  
    .appendTo("#login-form");
```



Patrones estructurales

Describen formas de componer
objetos para formar nuevas
estructuras flexibles y eficientes.

Patrones estructurales

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Proxy

// Cómo jQuery usa el Adapter

```
// IE <=11+
```

```
let inputEl = document.createElement("input")
```

```
inputEl.value = "priority-shipping";
```

```
inputEl.type = "radio";
```

```
console.log(inputEl.value) // undefined
```

```
// Con jQuery
```

```
let $input = $("<input />");
```

```
$input.val("priority-shipping");
```

```
$input.attr({ type: "radio" });
```

```
console.log($input.val()) // priority-shipping
```

Patrones de comportamiento

Gestionan algoritmos y responsabilidades entre objetos.

Patrones de comportamiento

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method
- Visitor

```
// Cómo jQuery usa el Composite
```

```
// Un solo elemento
```

```
$("#submit-button").addClass("big-text");
```

```
$("#hero-heading").addClass("big-text");
```

```
// Muchos elementos
```

```
$("button").addClass("big-text");
```

```
$(".title").addClass("big-text");
```

```
$("h1").addClass("big-text");
```

Patrones de Diseño - Singleton

Descripción y Caso de Uso



Singleton

(Creacional)

Es un patrón que te asegura que una clase solo tiene una instancia.

Esta única instancia puede ser consumida por cualquier otro objeto.

Estructura

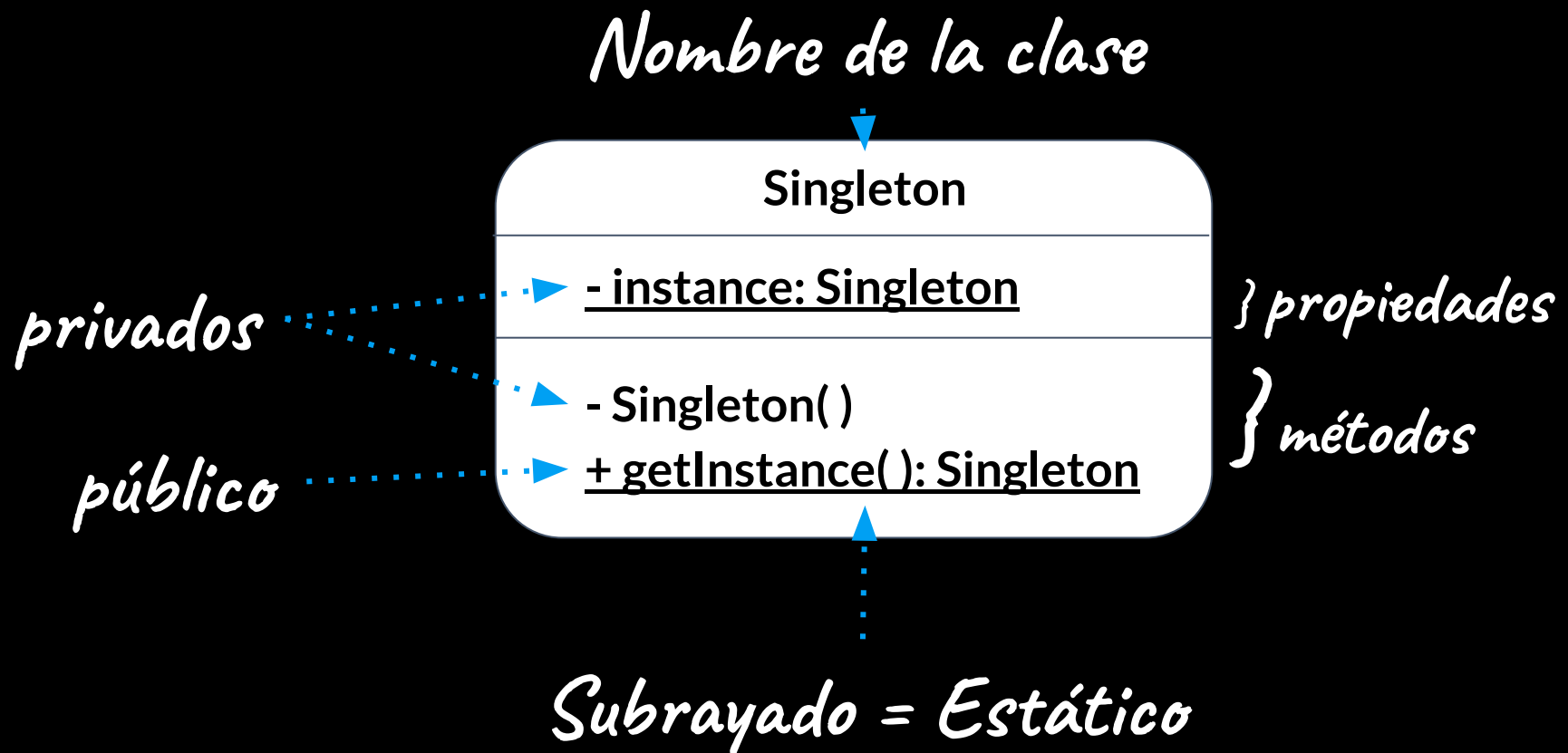
Singleton

- instance: Singleton

- Singleton()

+ getInstance(): Singleton

Estructura



Implementación (TypeScript)

```
class Database {  
    private static instance: Database;  
  
    private constructor() {  
        // init  
    }  
  
    public static getInstance(): Database {  
        if (!Database.instance) {  
            Database.instance = new Database();  
        }  
  
        return Database.instance;  
    }  
  
    public connect(url: string) {  
        // Implementation code  
    }  
}
```

Implementación (Node.js)

```
class Database {  
  constructor() {...}  
  public connect(url) {  
    // Implementation code  
  }  
}  
  
export default new Database();
```



Una analogía

El gobierno de tú país.

Podrán haber varios partidos,
pero un solo gobierno.

Por ejemplo, el “Gobierno de Alpha Centauri”
es un punto de acceso global que identifica
un grupo de personas a cargo.



Mongoose

(Caso de uso)

Mongoose es una librería que te ayuda a interactuar con una base de datos de MongoDB.

Te provee con una interfaz simple para buscar, escribir y validar datos.

En este tipo de librería solo quieres tener una instancia.

```
// Mongoose
```

```
// https://github.com/Automattic/mongoose/blob/master/lib/index.js
```

```
'use strict';
```

```
function Mongoose() {...}
```

```
Mongoose.prototype.connect = function() {...};
```

```
Mongoose.prototype.disconnect = function() {...};
```

```
Mongoose.prototype.Model = function Model () {...};
```

```
Mongoose.prototype.Schema = function Schema () {...};
```

```
module.exports = new Mongoose();
```



Otros casos de uso

Solo quieres tener un FileSystem.

Solo quieres tener un WindowManager

Patrones de Diseño - Observer

Descripción

Imaginate que...

Jaime te marca y te pregunta si tienes fiesta este sábado

> *Umm. No. ¿Por qué tendría una fiesta?*

Margarita te marca y te pregunta si tienes fiesta el miércoles

> *Obvio no. El jueves se trabaja.*

Finalmente sí tienes una fiesta

Deja llamo a Gabriel a avisarle...

> *Dale dale. Yo le caigo.*

Deja llamo a Angela a avisarle...

> *Amor. Pero si vivimos juntos. Obvio no me tenías que decir.*

Deja llamo a Manuel a avisarle...

> *Dude. Yo estoy en España. Cómo crees que voy a poder ir.*

¿Qué pasó?

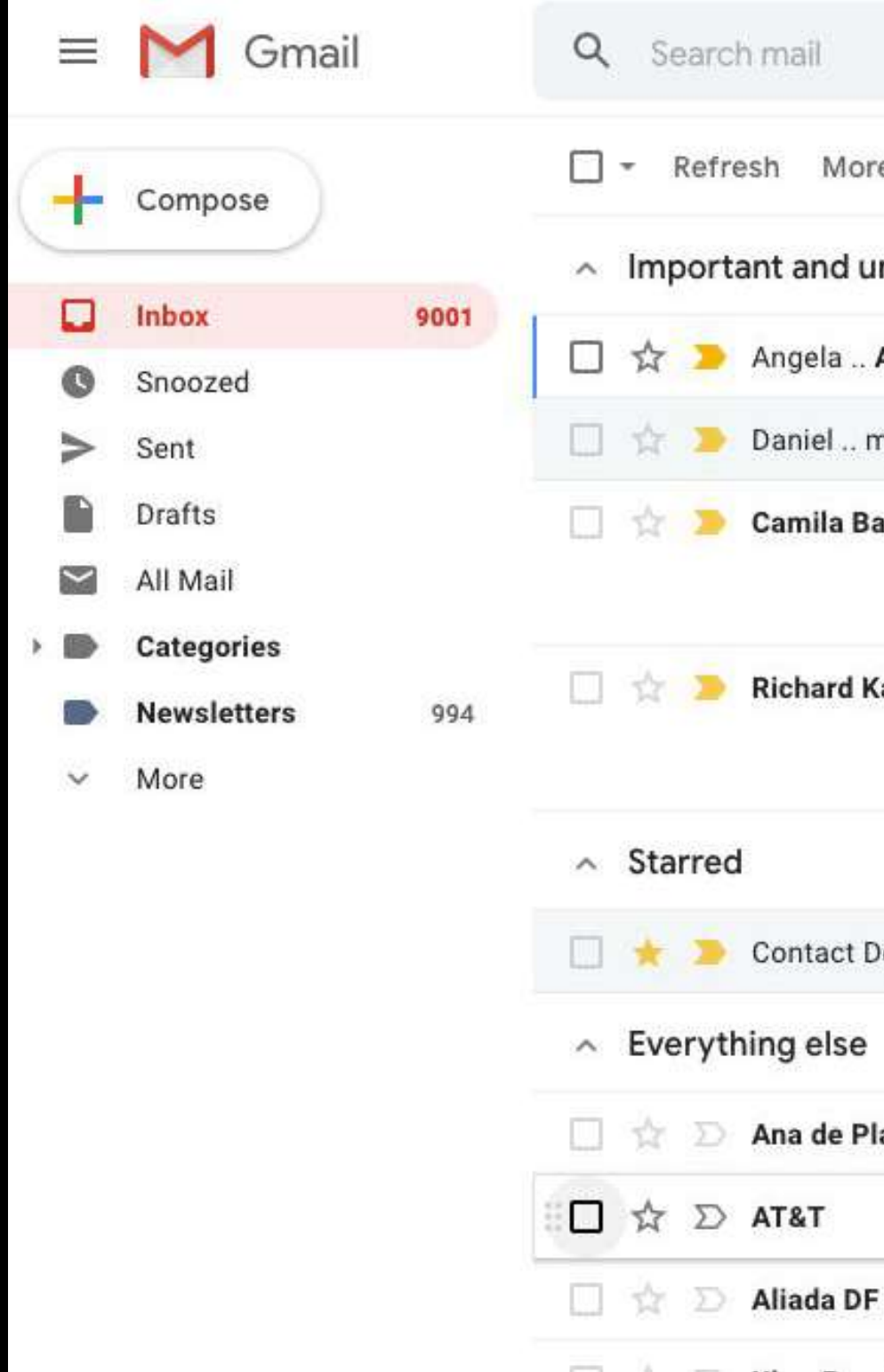
Tus amigos, para asegurarse que no se pierden ni una fiesta, te marcan para preguntarte si hay una pronto. Imaginate que todo el día, todos los días, suene el celular.



¿Qué pasó?

Cuando finalmente sí
tienes una fiesta,
empiezas a marcarle a
TODO el mundo en tu lista
de contactos.

Esto se siente un poco
como spam. Ugh.



Un problema de software

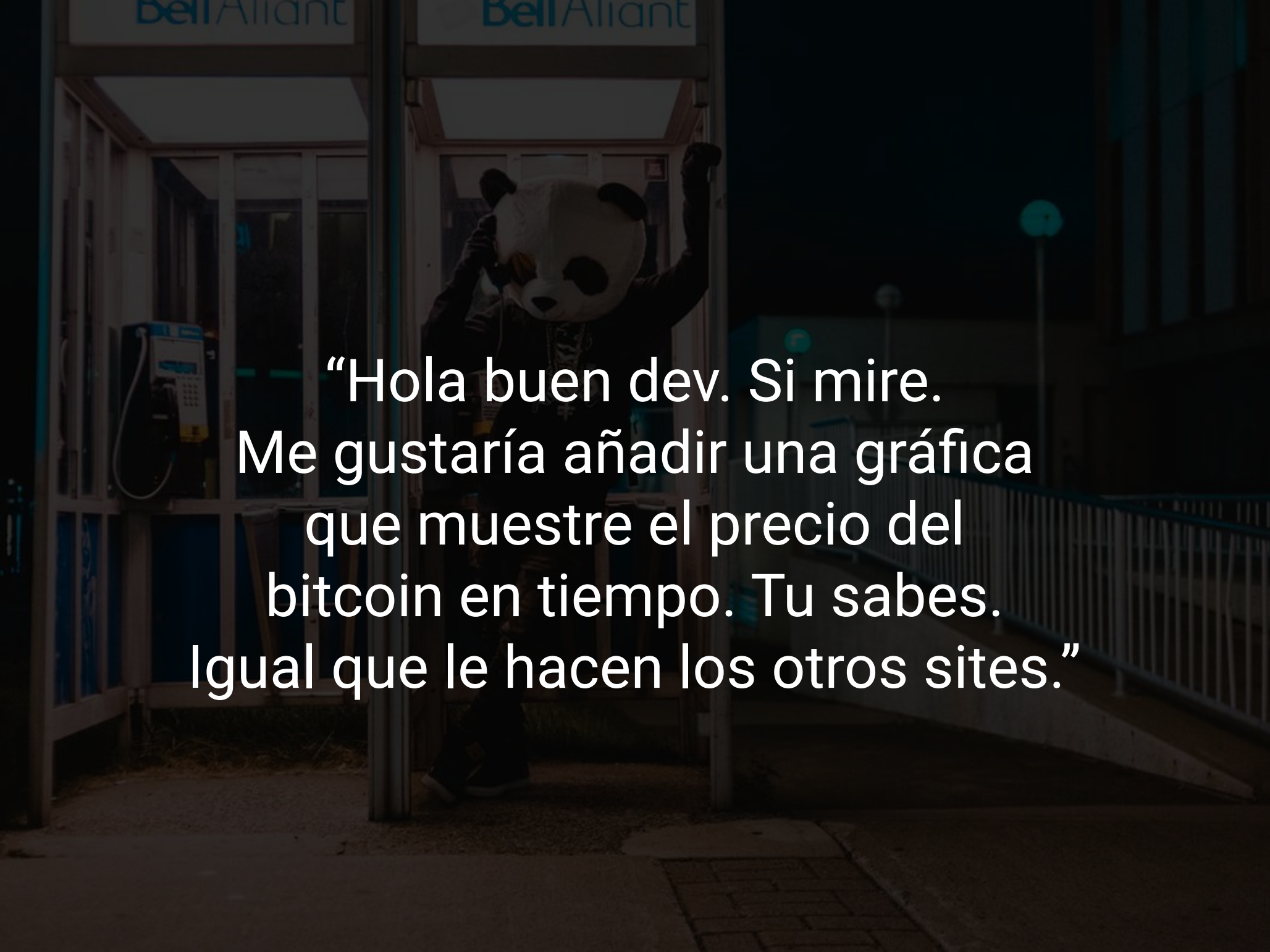
Tienes que hacer un tablero de Bitcoin:

- Tienes que desplegar el último precio de venta.
- Tienes que desplegar el último precio en el <title> de la página.

```
class BitcoinPriceTracker {
  constructor() {
    setInterval(this.notify, 1000);
  }

  private getPrice(): string {
    /* ... */
  }


  notify = async () => {
    let price = await this.getPrice();
    document.title = price;
    document.querySelector("#btc-price").innerText = price;
  };
}
```

A person wearing a large panda mascot costume is standing inside a bus stop at night. The bus stop has a glass roof and metal frame. Above the person, the words "Bell Alliance" are visible on the structure. To the left, there is a public phone. The background shows a dark street with a railing and some distant lights.

“Hola buen dev. Si mire.
Me gustaría añadir una gráfica
que muestre el precio del
bitcoin en tiempo. Tu sabes.
Igual que le hacen los otros sites.”



```
class BitcoinPriceTracker {  
  notify = async () => {  
    let price = await this.getPrice();  
    document.title = price;  
    document.querySelector("#btc-price").innerText = price;  
  };  
}
```



*Hay que conocer la
implementación de cada objeto
que quiere enterarse.*

“

**Programa a una
interfaz, no a una
implementación**

”

Principio de diseño de software

Patrones de Diseño -
Observer

Estructura



Observer

(Comportamiento)

Un objeto tiene alguna información/estado interesante.

Otros objetos están interesados y quieren enterarse de cambios de este estado.

Observer

(Comportamiento)

Subject / Publisher

Un objeto tiene alguna información/estado interesante.

Otros objetos están interesados y quieren enterarse de cambios de este estado.

Observer / Subscriber



Observer

(Comportamiento)

Este patrón sugiere que creamos un objeto (el Subject) con mecanismos de suscripción.

Los objetos interesados (Observer) se pueden suscribir para recibir actualizaciones de un estado.

Observer

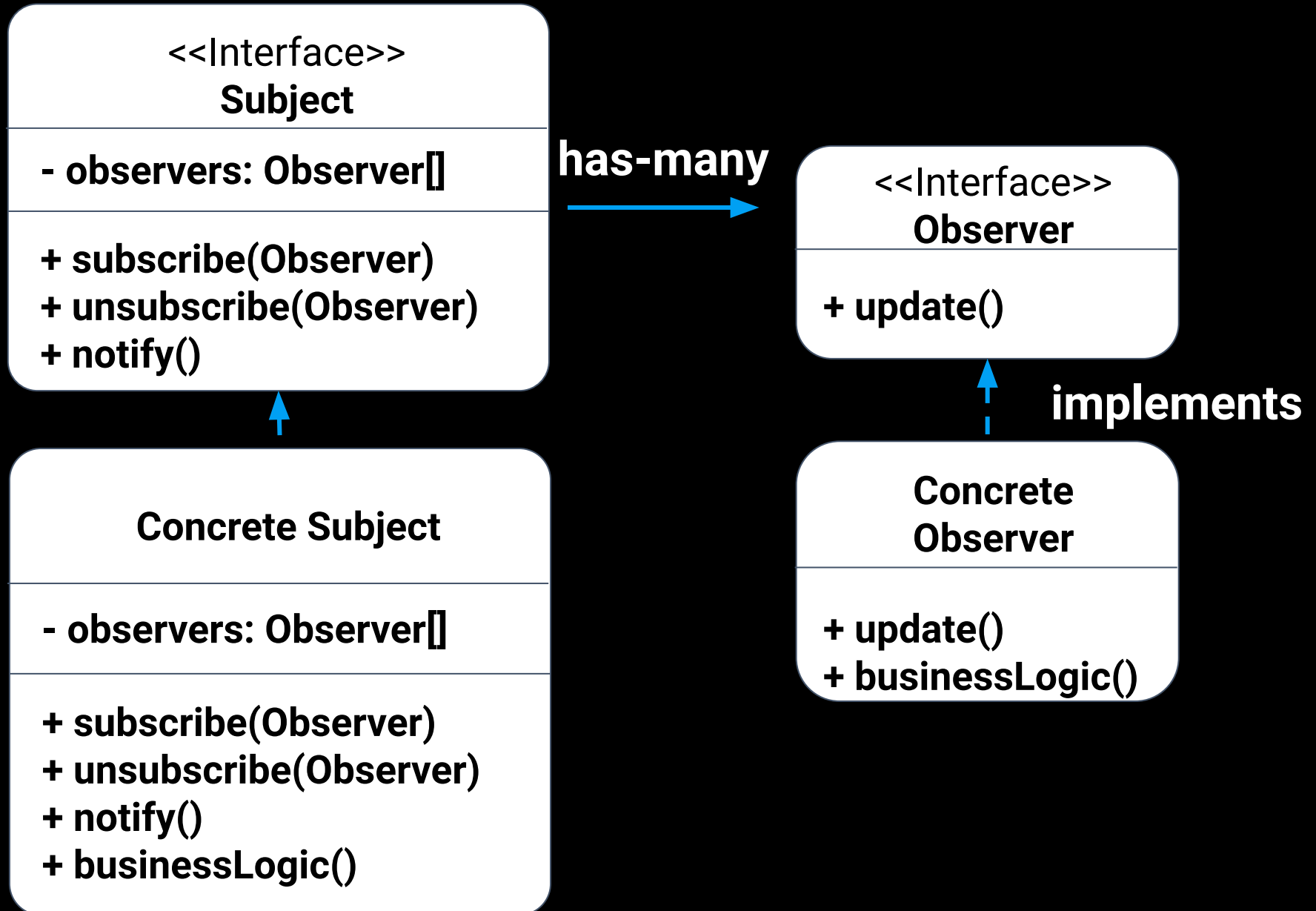
(Comportamiento)

Cuando el estado cambia, el Subject le notifica a todos los Observers.

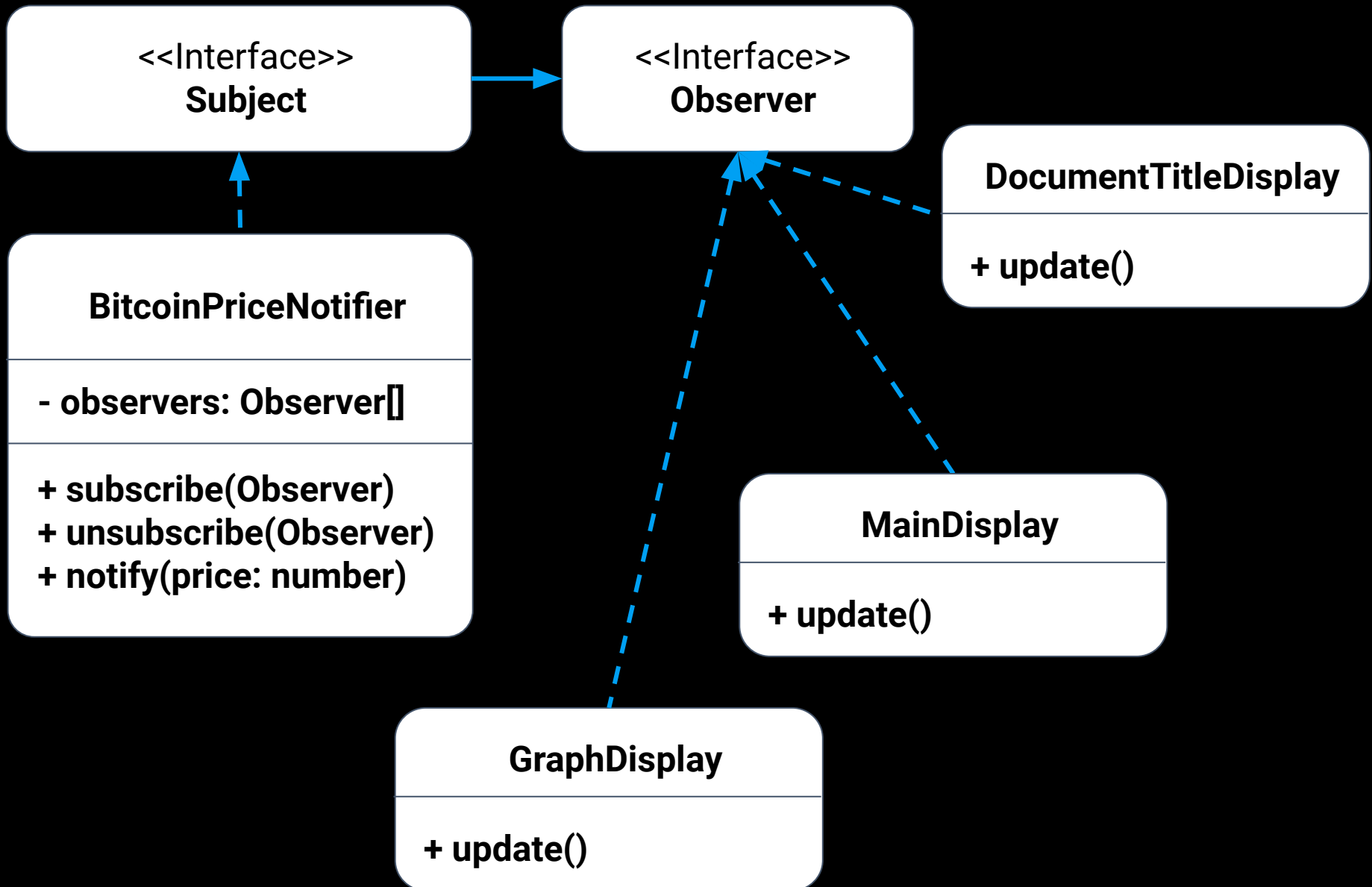
Si un Observer ya no le interesa recibir notificaciones, se puede desuscribir.

Esto suena a un newsletter

Estructura



Estructura



Patrones de Diseño - Observer

Casos de uso

Redux

- Una librería de manejo de estado.
- Inicializas un *store* con un estado y un reducer.
- Despachas acciones que modifican el estado.
- El *store* notifica que el estado cambió.

Redux

- Una librería de manejo de estado.
- Inicializas un *store* con un estado y un reducer.
- Despachas acciones que modifican el estado.
- El *store* **notifica** que el estado cambió.

```
// Redux - API
```

```
import { createStore } from 'redux';
```

```
const store = createStore( ... );
```

```
store.subscribe(function() {...});
```

```
store.dispatch({ type: 'INCREMENT' });
```

```
store.getState();
```

redux/createStore.js at master · reduxjs/redux · GitHub

GitHub, Inc. (US)https://github.com/reduxjs/redux/blob/master/src/createStore.js120%

```
183 function dispatch(action) {
184   if (!isPlainObject(action)) {
185     throw new Error(
186       'Actions must be plain objects. ' +
187       'Use custom middleware for async actions.'
188     )
189   }
190
191   if (typeof action.type === 'undefined') {
192     throw new Error(
193       'Actions may not have an undefined "type" property. ' +
194       'Have you misspelled a constant?'
195     )
196   }
197
198   if (isDispatching) {
199     throw new Error('Reducers may not dispatch actions.')
200   }
201
202   {
203     try {
204       isDispatching = true
205       currentState = currentReducer(currentState, action)
206     } finally {
207       isDispatching = false
208     }
209   }
210
211   {
212     const listeners = (currentListeners = nextListeners)
213     for (let i = 0; i < listeners.length; i++) {
214       const listener = listeners[i]
215       listener()
216     }
217   }
218
219   return action
220 }
```

el estado

los observers

Modifica el estado...

...y notifica a los interesados

```
redux/createStore.js at master · ReduxJS/redux · GitHub
https://github.com/reduxjs/redux/blob/master/src/createStore.js
133%

116 * @param {Function} listener A callback to be invoked on every dispatch.
117 * @returns {Function} A function to remove this change listener.
118 */
119 function subscribe(listener) ← un observer
120   if (typeof listener !== 'function') {
121     throw new Error('Expected the listener to be a function.')
122   }
123
124   if (isDispatching) {
125     throw new Error(
126       'You may not call store.subscribe() while the reducer is executing. ' +
127       'If you would like to be notified after the store has been updated, subscribe from a ' +
128       'component and invoke store.getState() in the callback to access the latest state. ' +
129       'See https://redux.js.org/api-reference/store#subscribe(listener) for more details.'
130     )
131   }
132
133   let isSubscribed = true
134
135   ensureCanMutateNextListeners()
136   nextListeners.push(listener)
137
138   return function unsubscribe() ← unsubscribe
139     if (!isSubscribed) {
140       return
141     }
142
143     if (isDispatching) {
144       throw new Error(
145         'You may not unsubscribe from a store listener while the reducer is executing. ' +
146         'See https://redux.js.org/api-reference/store#subscribe(listener) for more details.'
147       )
148     }
149     ensureCanMutateNextListeners()
150     const index = nextListeners.indexOf(listener)
151     nextListeners.splice(index, 1)
152     isSubscribed = false
153   }
154 }
```

EventEmitter (Node.js)

Node.js

About these Docs

Usage & Example

Assertion Testing

Async Hooks

Buffer

C++ Addons

C/C++ Addons - N-API

Child Processes

Cluster

Command Line Options

Console

Crypto

Debugger

Deprecated APIs

DNS

Domain

ECMAScript Modules

Errors

Events

File System

Class: EventEmitter

Added in: v0.1.26

The `EventEmitter` class is defined and exposed by the `events` module:

```
const EventEmitter = require('events');
```

All `EventEmitters` emit the event `'newListener'` when new listeners are added and `'removeListener'` when existing listeners are removed.

Event: 'newListener'

Added in: v0.1.26

- `eventName` `<string> | <symbol>` The name of the event being listened for
- `listener` `<Function>` The event handler function

The `EventEmitter` instance will emit its own `'newListener'` event *before* a listener is added to its internal array of listeners.

Listeners registered for the `'newListener'` event will be passed the event name and a reference to the listener being added.

The fact that the event is triggered before adding the listener has a subtle but important side effect: any *additional* listeners registered to the same `name` *within* the `'newListener'` callback will be inserted *before* the listener that is in the process of being added.

```
const myEmitter = new MyEmitter();
// Only do this once so we don't loop forever
myEmitter.once('newListener', (event, listener) => {
  if (event === 'event') {
    // Insert a new listener in front
    myEmitter.on('event', () => {
      console.log('B');
    });
  }
});
```

#

#

EventEmitter (Node.js)

emitter.on = subscribe
emitter.off = unsubscribe
emitter.emit = notify

- Class: EventEmitter
 - Event: 'newListener'
 - Event: 'removeListener'
 - EventEmitter.listenerCount(emitter, eventName)
 - EventEmitter.defaultMaxListeners
 - emitter.addListener(eventName, listener)
 - emitter.emit(eventName[, ...args])
 - emitter.eventNames()
 - emitter.getMaxListeners()
 - emitter.listenerCount(eventName)
 - emitter.listeners(eventName)
 - emitter.off(eventName, listener)
 - emitter.on(eventName, listener)
 - emitter.once(eventName, listener)
 - emitter.prependListener(eventName, listener)
 - emitter.prependOnceListener(eventName, listener)
 - emitter.removeAllListeners([eventName])
 - emitter.removeListener(eventName, listener)
 - emitter.setMaxListeners(n)
 - emitter.rawListeners(eventName)



ECMAScript Observable

Hay una propuesta para añadir
Observables al standard library de
ECMAScript.

Aun está en Stage 1.

Patrones de Diseño -
Decorator

Descripción y Casos de Uso

Imagine que...

Entras a apple.com
para comprar una
computadora, pero en
lugar de encontrar “una
Macbook Pro de 13” y
otra de 15”, encuentras...

MacBook Pro


Overview macOS Tech Specs

Add a trade-in
Get a refund of up to \$1400 when you trade in an eligible computer, or recycle it for free.*
[Get started](#)

Choose your new MacBook Pro and select a finish.

13-inch 15-inch

15-inch MacBook Pro




Space Gray


New


Touch Bar and Touch ID
2.6GHz 6-Core Processor with Turbo Boost up to 4.5GHz
256GB Storage


2.6GHz 6-core 9th-generation Intel Core i7 processor
Turbo Boost up to 4.5GHz
Radeon Pro 555X with 4GB of GDDR5 memory
16GB 2400MHz DDR4 memory
256GB SSD storage¹
Retina display with True Tone
Touch Bar and Touch ID
Four Thunderbolt 3 ports

\$2,399.00
[Up to 18 months of special financing >](#)

[Select](#) 

 **Delivery:**
In Stock
Free Shipping
[Get delivery dates](#)

 **Pickup:**
[Check availability](#)




Space Gray


New


Touch Bar and Touch ID
2.3GHz 8-Core Processor with Turbo Boost up to 4.8GHz
512GB Storage

2.3GHz 8-core 9th-generation Intel Core i9 processor
Turbo Boost up to 4.8GHz
Radeon Pro 560X with 4GB of GDDR5 memory
16GB 2400MHz DDR4 memory
512GB SSD storage¹
Retina display with True Tone
Touch Bar and Touch ID
Four Thunderbolt 3 ports

\$2,799.00
[Up to 18 months of special financing >](#)

[Select](#) 

 **Delivery:**
In Stock
Free Shipping
[Get delivery dates](#)

 **Pickup:**
[Check availability](#)



Space Gray

New

Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo



Space Gray

New

Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo
Boost up to 4.8GHz
1TB Storage, 8GB RAM
\$3,199.00

[Up to 18 months of special financing >](#)

Select



Space Gray

New

Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo



Space Gray

New

Touch Bar and
2.3GHz 8-Core
Processor with
Boost up to 4.
512GB Storage
RAM
\$2,999.00

[Up to 18 months of special financing >](#)

Select



Space Gray

New

Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo
Boost up to 4.8GHz
1TB Storage, 32GB RAM
\$3,499.00

[Up to 18 months of special financing >](#)

Select



Space Gray

New

Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo



Space Gray



Space Gray

New

Touch Bar and Touch ID
2.6GHz 6-Core



Space Gray

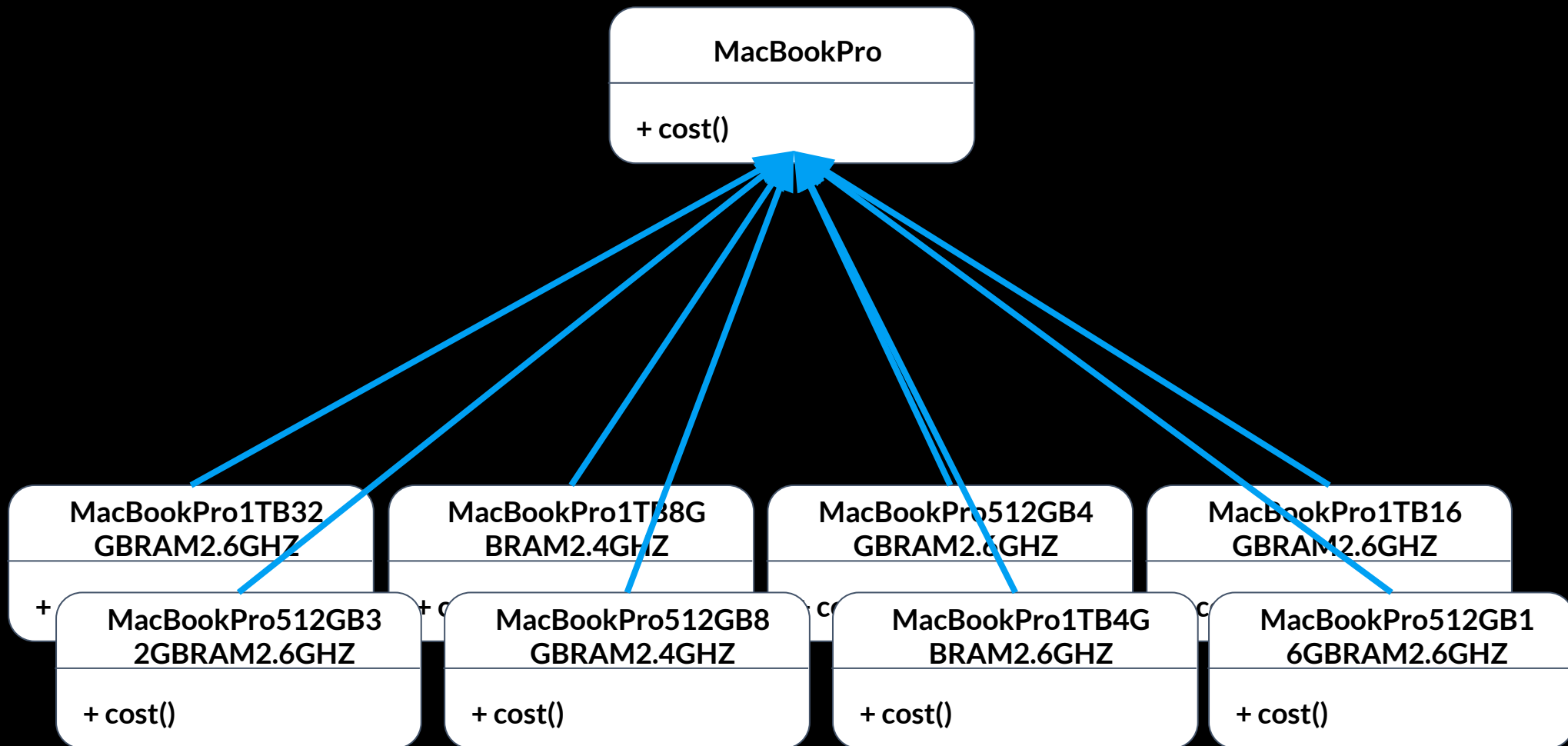
New

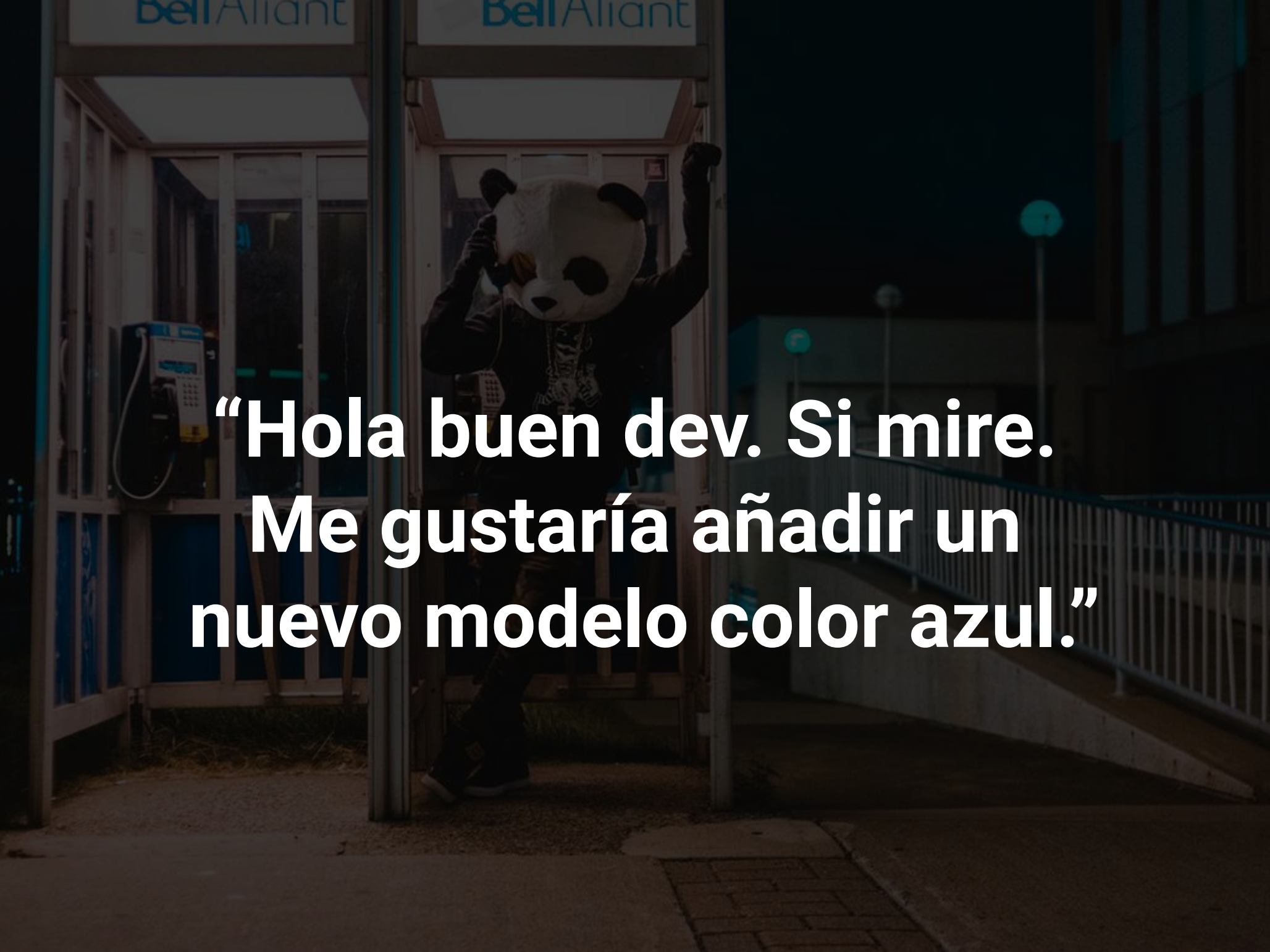
Touch Bar and Touch ID
2.3GHz 8-Core
Processor with Turbo
Boost up to 4.8GHz
512GB Storage, 32GB
RAM
\$3,199.00

[Up to 18 months of special financing >](#)

Select

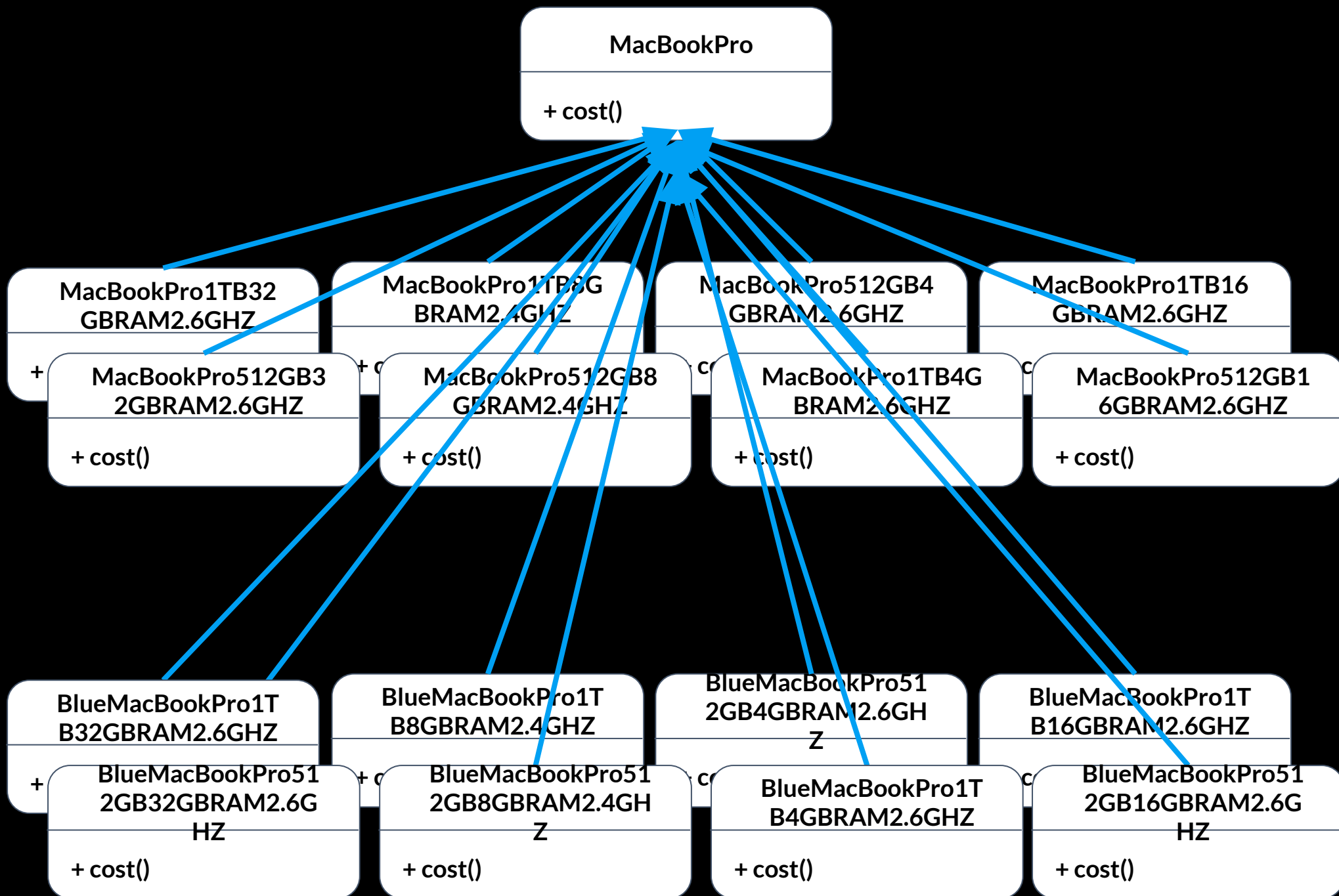


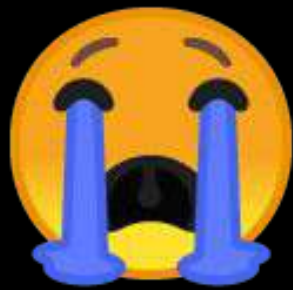


A person wearing a large panda mascot head and a dark jacket is standing inside a public phone booth. The booth has "Bell Alliant" signs above the entrance. The person is holding a phone to their ear. The background is dark, suggesting it is nighttime, with some streetlights and a building visible in the distance.

**“Hola buen dev. Si mire.
Me gustaría añadir un
nuevo modelo color azul.”**

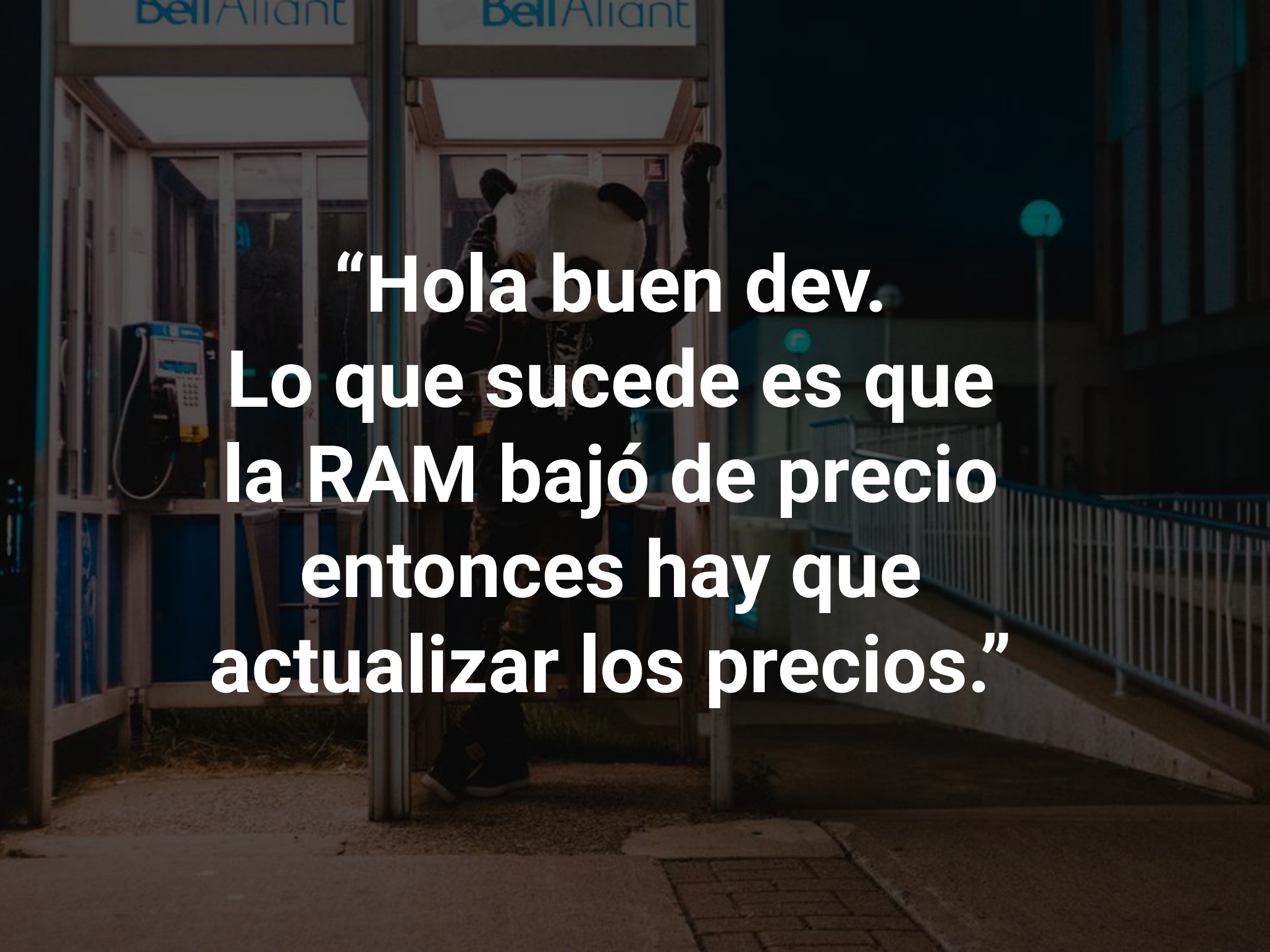






```
class MacBookPro1TB32GBRAM26GHZ {  
    constructor(){ ... }  
  
    cost() {  
        return 3199;  
    }  
}
```

```
class MacBookPro1TB16GBRAM26GHZ {  
    constructor(){ ... }  
  
    cost() {  
        return 2999;  
    }  
}
```

A person wearing a full-body panda costume is standing inside a glass phone booth. The booth has "Bell Alliant" logos on the top. To the left of the booth is a public payphone. The scene is at night, with a city street and a building visible in the background. A ramp with a metal railing is on the right side of the frame.

**“Hola buen dev.
Lo que sucede es que
la RAM bajó de precio
entonces hay que
actualizar los precios.”**

“

**Una entidad de software
(clase, módulo, función, etc.)
debe quedar abierta para su
extensión, pero cerrada
para su modificación**

”

*Principio de abierto/cerrado de diseño de software
(Open Closed Principle)*

Patrones de Diseño -
Decorator

Estructura



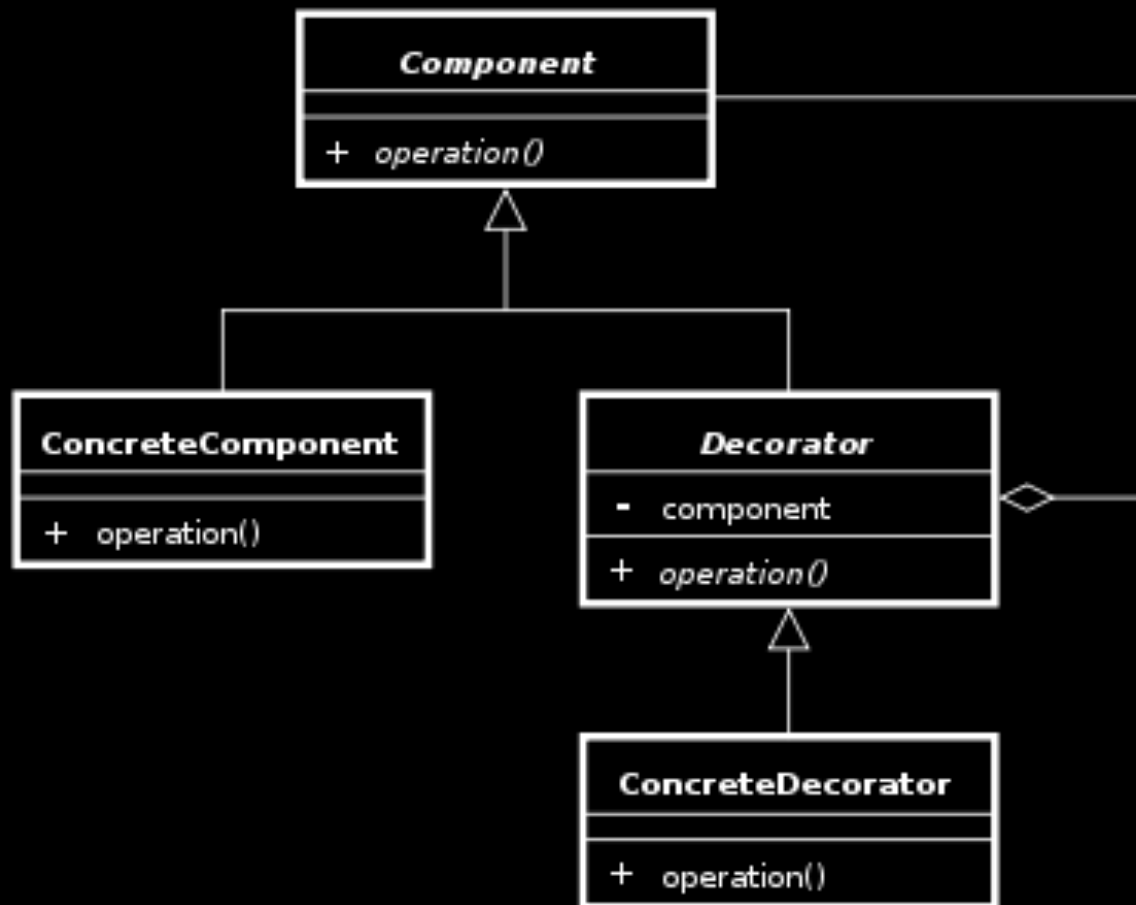
Decorator

(Estructural)

- Añade nuevas responsabilidades a un objeto de forma dinámica.
- Nos permiten extender funcionalidad sin tener que usar subclases.

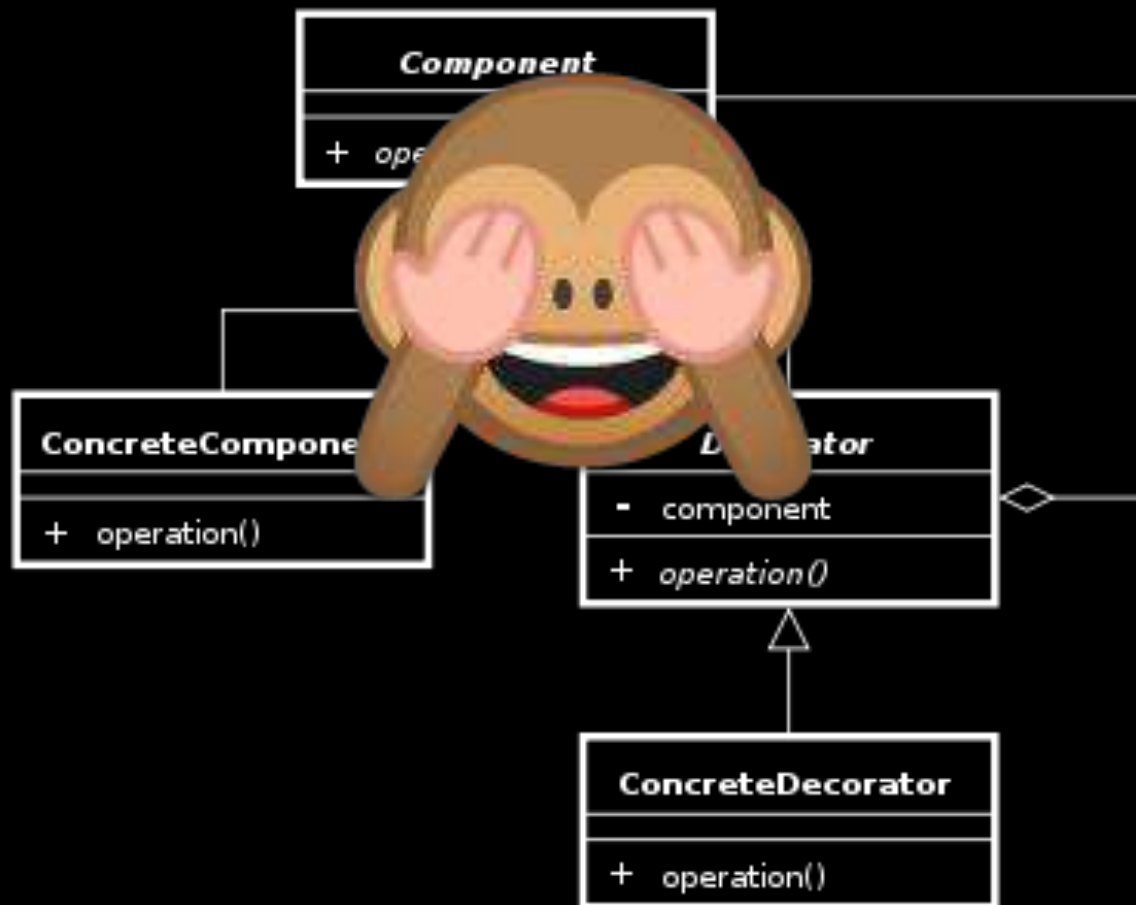
Decorator

(Estructural)



Decorator

(Estructural)



The JavaScript Way

(Monkey patching)

```
class MacbookPro {  
  constructor() { this.memory = 8; }  
  cost() { return 2399; }  
}  
  
function withMemory(amount, computer) {  
  let cost = computer.cost();  
  computer.cost = function () {  
    let memoryCost = Math.max((amount - 8) * 25, 0);  
    return cost + memoryCost;  
  };  
}
```

The JavaScript Way

(Monkey patching)

```
let macbook = new MacbookPro()  
withMemory(32, macbook)  
  
console.log(macbook.cost())  
  
// 2999
```



The JavaScript Way

Closures

Herencia prototipal

Middleware

ECMAScript Decorators (*Stage 2*)

Patrones de Diseño -
Decorator

Casos de uso

Lodash.memoize

```
lodash/lodash.js at 4.17.10 · lod... X
← → ↺ ⓘ GitHub, Inc. (US) | https://github.com/lodash/lodash/blob/4.17.10/lodash.js#L10532 150% ... ☆ ⬇ 📄 🐱 ⋮

10534     * // => ['a', 'b']
10535     *
10536     * // Replace `_.memoize.Cache`.
10537     * _.memoize.Cache = WeakMap;
10538     */
10539     function memoize(func, resolver) {
10540       if (typeof func !== 'function' || (resolver !== null && typeof resolver !== 'function')) {
10541         throw new TypeError(FUNC_ERROR_TEXT);
10542       }
10543       var memoized = function() {
10544         var args = arguments,
10545             key = resolver ? resolver.apply(this, args) : args[0],
10546             cache = memoized.cache;

10547
10548         if (cache.has(key)) {
10549           return cache.get(key);
10550         }
10551         var result = func.apply(this, args);
10552         memoized.cache = cache.set(key, result) || cache;
10553         return result;
10554       };
10555       memoized.cache = new (memoize.Cache || MapCache);
10556       return memoized;
10557     }
10558
10559     // Expose `MapCache`.
10560     memoize.Cache = MapCache;
10561
10562     /**
10563     * Creates a function that negates the result of the predicate `func`. The
10564     * `func` predicate is invoked with the `this` binding and arguments of the
```

Lodash.memoize

```
10534 * // => ['a', 'b']
10535 *
10536 * // Replace `_.memoize.Cache`. función decorada
10537 * _.memoize.Cache = WeakMap;
10538 */
10539 function memoize(func, resolver) {
10540   if (typeof func != 'function' || (resolver != null && typeof resolver != 'function')) {
10541     throw new TypeError(FUNC_ERROR_TEXT);
10542   }
10543   var memoized = function() {
10544     var args = arguments,
10545         key = resolver ? resolver.apply(this, args) : args[0],
10546         cache = memoized.cache;
10547
10548     { Nueva funcionalidad
10549       if (cache.has(key)) {
10550         return cache.get(key);
10551       }
10552       var result = func.apply(this, args);
10553       memoized.cache = cache.set(key, result) || cache;
10554       return result;
10555     }
10556     memoized.cache = new (memoize.Cache || MapCache);
10557     return memoized;
10558   };
10559   // Expose `MapCache`.
10560   memoized.Cache = MapCache;
10561
10562   /**
10563    * Creates a function that negates the result of the predicate `func`. The
10564    * `func` predicate is invoked with the `this` binding and arguments of the
```

```
function measure(fn) {  
  let start = Date.now();  
  fn();  
  console.log(`Time: ${Date.now() - start}ms`);  
}
```

```
function fibonacci(num) {...}
```

```
let fastFibonacci = lodash.memoize(fibonacci);
```

```
measure(() => fastFibonacci(100000)); // "Time: 625ms"
```

```
measure(() => fastFibonacci(100000)); // "Time: 0ms"
```