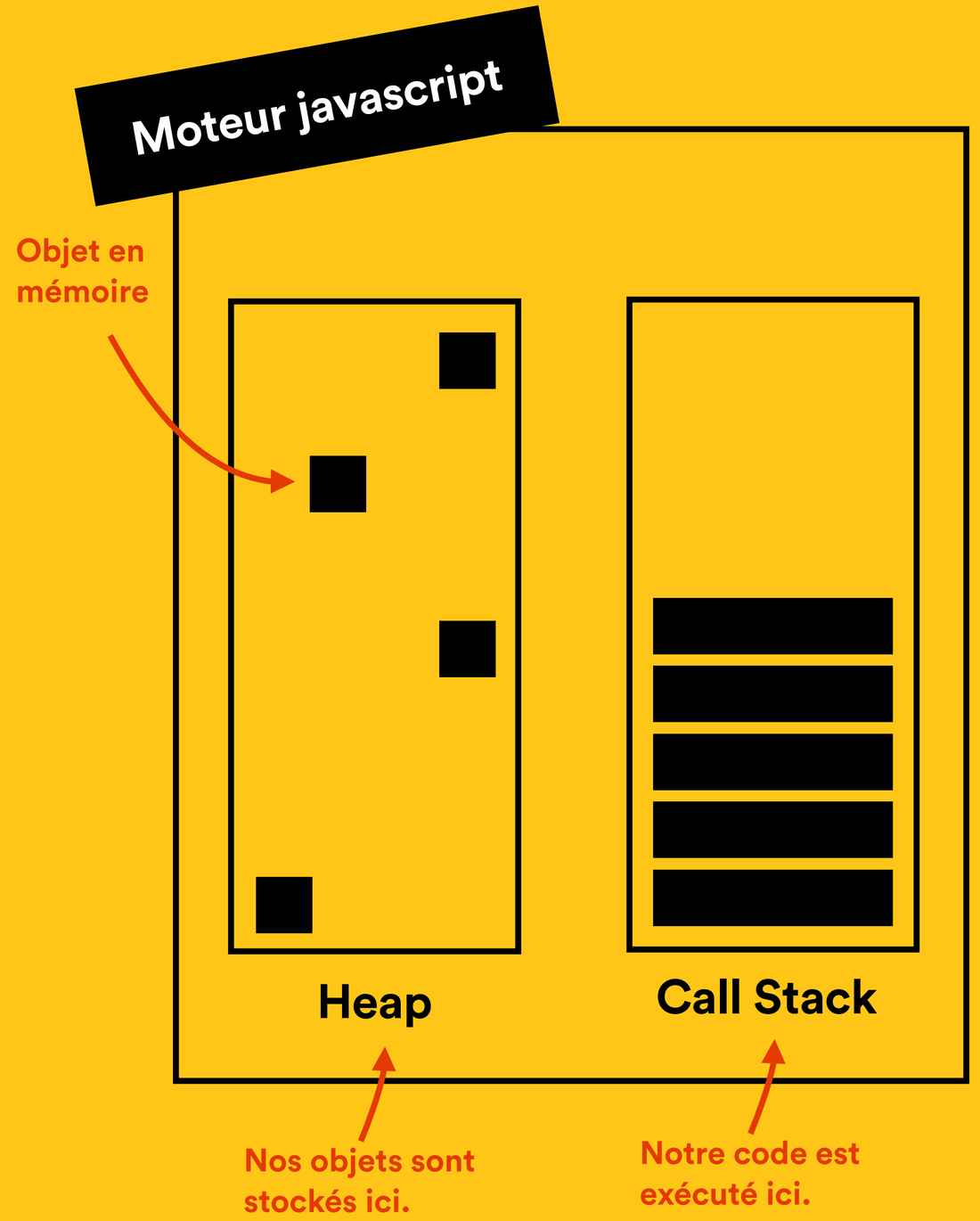
HEIG-VD / INGÉNIERIE DES MÉDIAS / PROGRAMMATION WEB

EVENT LOOP + ASYNC/AWAIT


Moteur javascript
Objet en mémoire
Heap
Call Stack
Nos objets sont stockés ici.
Notre code est exécuté ici.

Call Stack

Web APIs

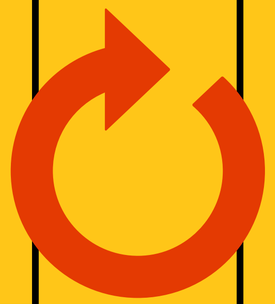
```
const el = document.querySelector("img");

el.src = "sympa.jpg";

el.addEventListener("load", () => {
  el.classList.add("fadeIn");
});

fetch("https://url.ch/api")
    .then((res) => console.log(res));
```

Callback Queue



Event Loop

Call Stack

Web APIs

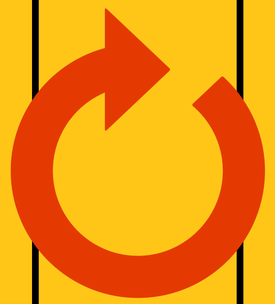
```
const el = document.querySelector("img");

el.src = "sympa.jpg";

el.addEventListener("load", () => {
  el.classList.add("fadeIn");
});

fetch("https://url.ch/api")
    .then((res) => console.log(res));
```

Callback Queue



Event Loop

HE IG VD

async/await est une façon d’écrire du code asynchrone dans un style synchrone. Cette technique rend la consommation de Promesses plus facile à raisonner.

Il s’agit de “sucre syntaxique”. C’est à dire qu’il s’agit juste d’une abstraction plus lisible, construit sur le système des Promesses.

```
const add = async (a, b) ⇒ {
    return a + b
};

console.log(add(2, 2)) // Promise
```

Le mot async devant une fonction indique simplement que cette dernière va toujours retourner une Promise.

```
const add = async (a, b) ⇒ {
    return a + b
};

console.log(add(2, 2)) // Promise

add(2, 2).then(alert) // 4
```

On pourrait croire dans l'exemple ci-contre qu'on retournerait l'addition des paramètres a et b.

Mais non, on retourne une promesse dont la valeur de résolution est a + b.

On peut donc appeler un méthode `.then` sur la fonction pour traiter le contenu de la promesse.

HEIG VD

```
const fn = async () ⇒ {
    let res = await fetch("url");
    console.log(res)
    // Valeur de la promesse
}
```

Le mot clé `await` pause l'exécution de notre fonction jusque'à ce que la promesse associée soit résolue. La valeur associée à notre promesse peut ensuite être stockée dans une variable.

Bien que la fonction async soit en pause, le reste de notre code continue d'être exécuté.

HEIG VD

```
const fn =() ⟹ {
    let res = await fetch("url");
}
// Erreur
```

Attention, vous ne pouvez pas utiliser `await` dans une fonction qui n’est pas async!

```
const fn = async () ⇒ {
  try {
    let res = await fetch("url");
    console.log(res);
  } catch (e) {
    console.error(`💥${e.message}`);
  }
};
```

On utilise `try…catch` pour gérer les erreurs async/await.

HEIG VD

```
const fn = async () ⇒ {
  let results = await Promise.all([
    fetch(url1),
    fetch(url2)
  ]);
};
```

async/await fonctionne très bien avec `Promise.all`

INDEX.HTML

```html
<script src="app.js" type="module"></script>
```

APP.JS

```
let res = await fetch("url");
```

Dans des navigateurs très récents, au plus haut niveau de note code, à condition que notre JavaScript soit traité comme un module (nous verrons ce que sont les modules mi-janvier).

HEIG VD