Asynchronous

Processing

in Web Development

Abraham N. Aldaco-Gastélum, Ph.D.

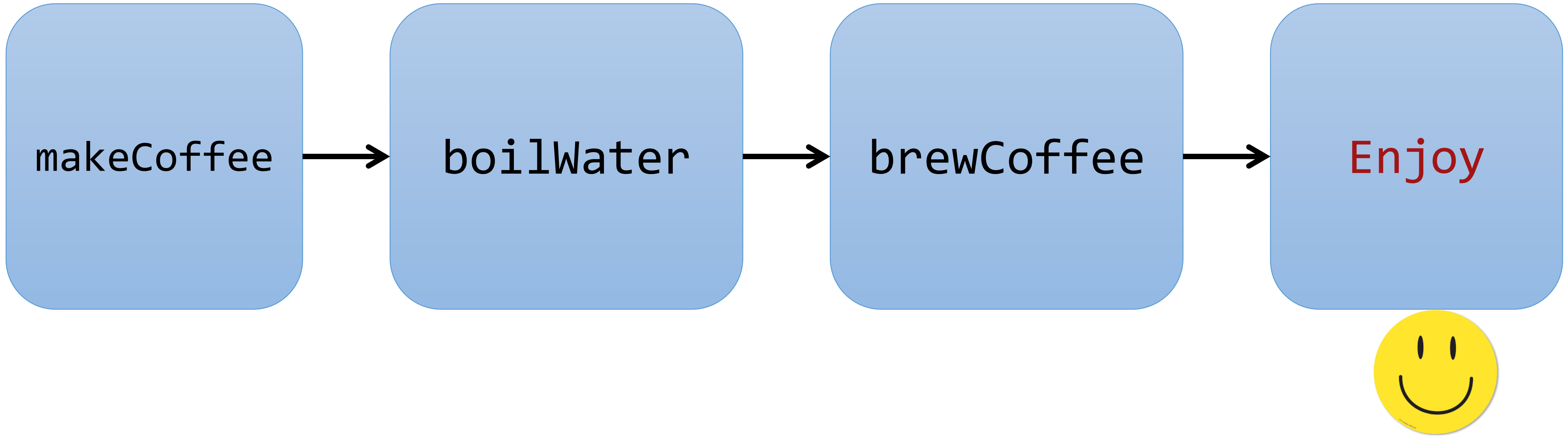
Synchronous Processing

Synchronous Processing

Before showing Asynchronous Processing,

We want to review a simpler and more typical situation in programming processing which is Synchronous processing.

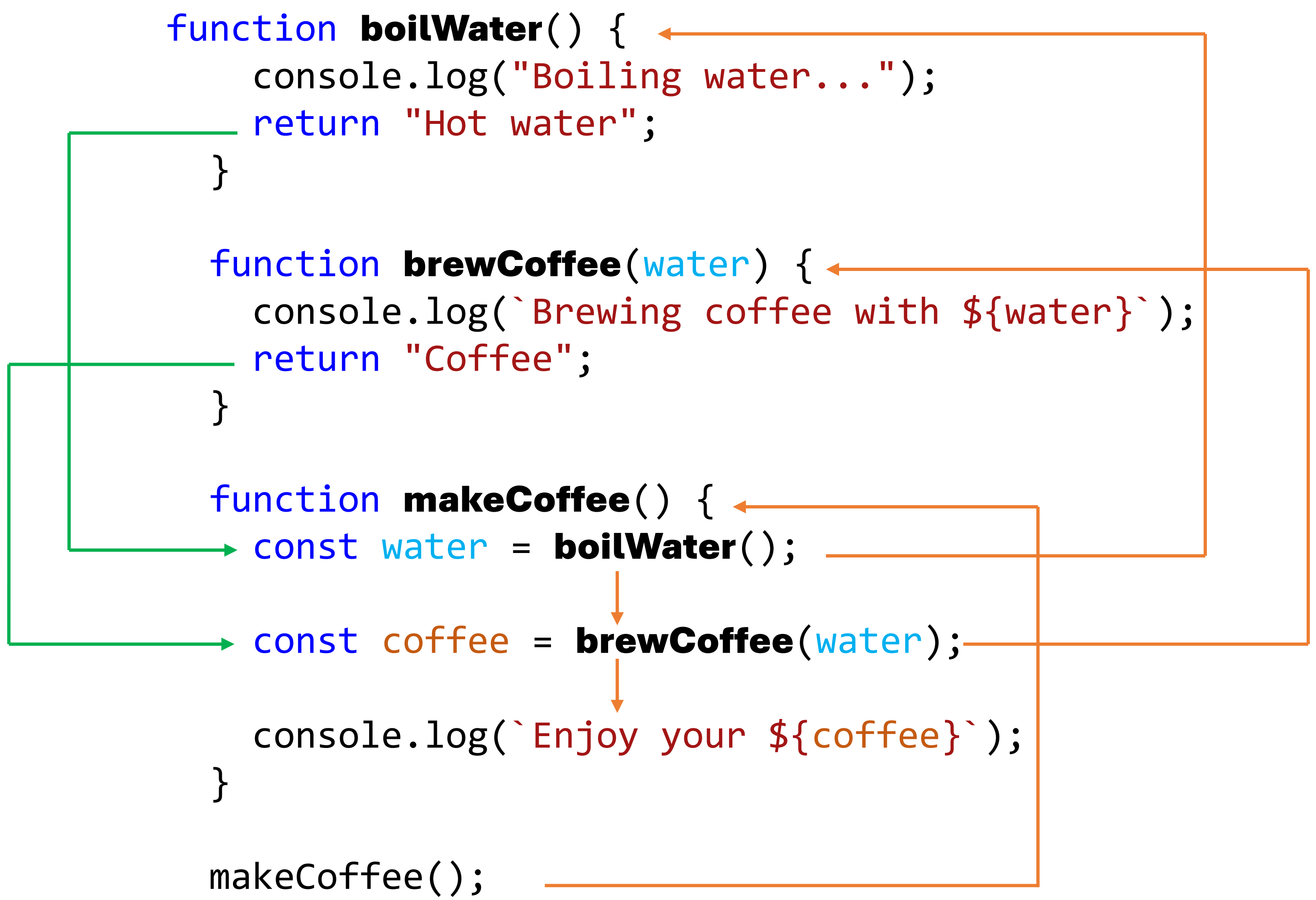
Let’s think in the next process :



In JavaScript, we can prepare coffee in the next program:

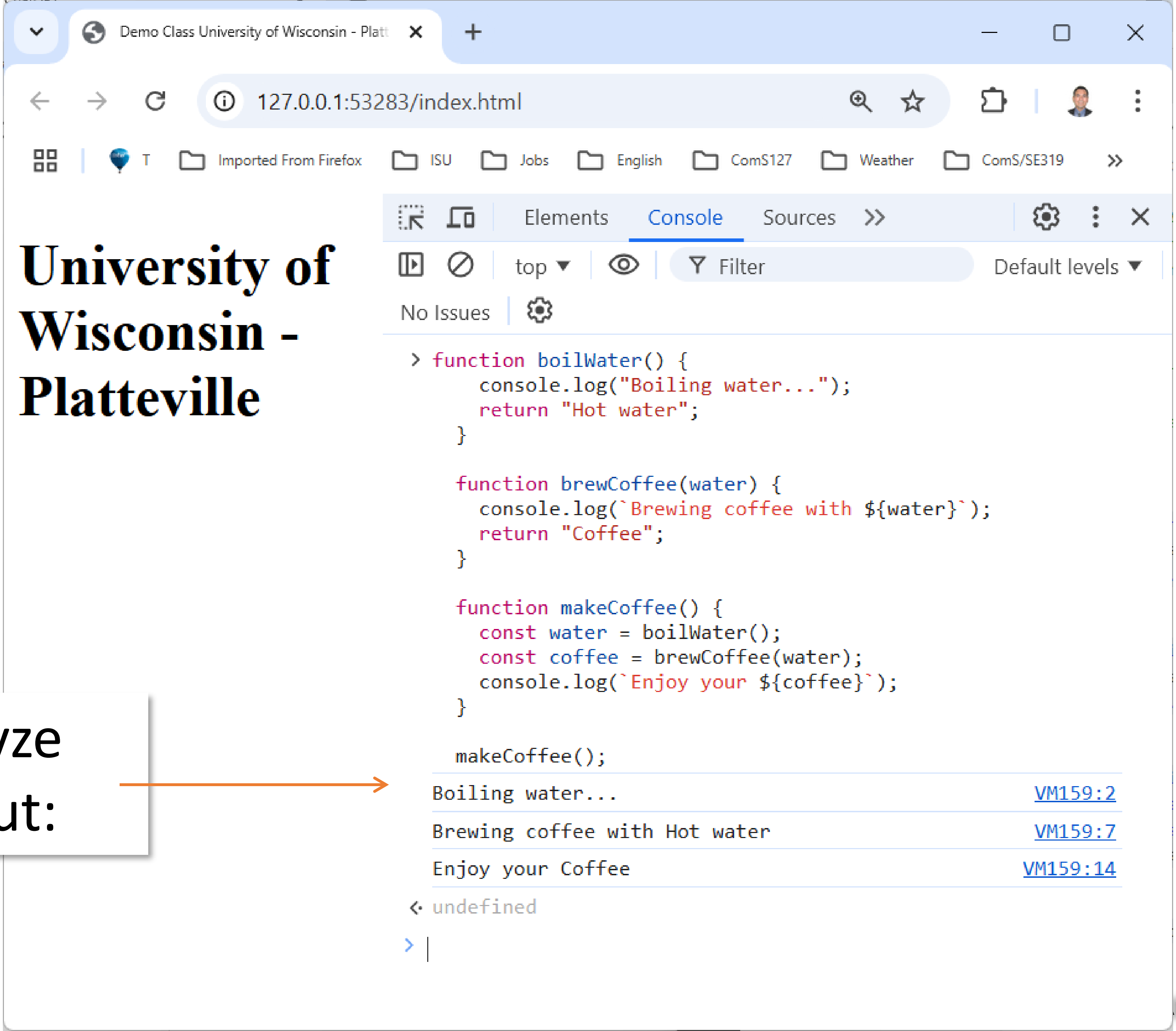
```
function boilWater() {
    console.log("Boiling water...");
    return "Hot water";
  }

  function brewCoffee(water) {
    console.log(`Brewing coffee with ${water}`);
    return "Coffee";
  }

  function makeCoffee() {
    const water = boilWater();

    const coffee = brewCoffee(water);

    console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
```

The sequence of execution is the next:



```
function boilWater() {
    console.log("Boiling water...");
    return "Hot water";
  }

  function brewCoffee(water) {
    console.log(`Brewing coffee with ${water}`);
    return "Coffee";
  }

  function makeCoffee() {
    const water = boilWater();

    const coffee = brewCoffee(water);

    console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
```

Try the code in a browser:



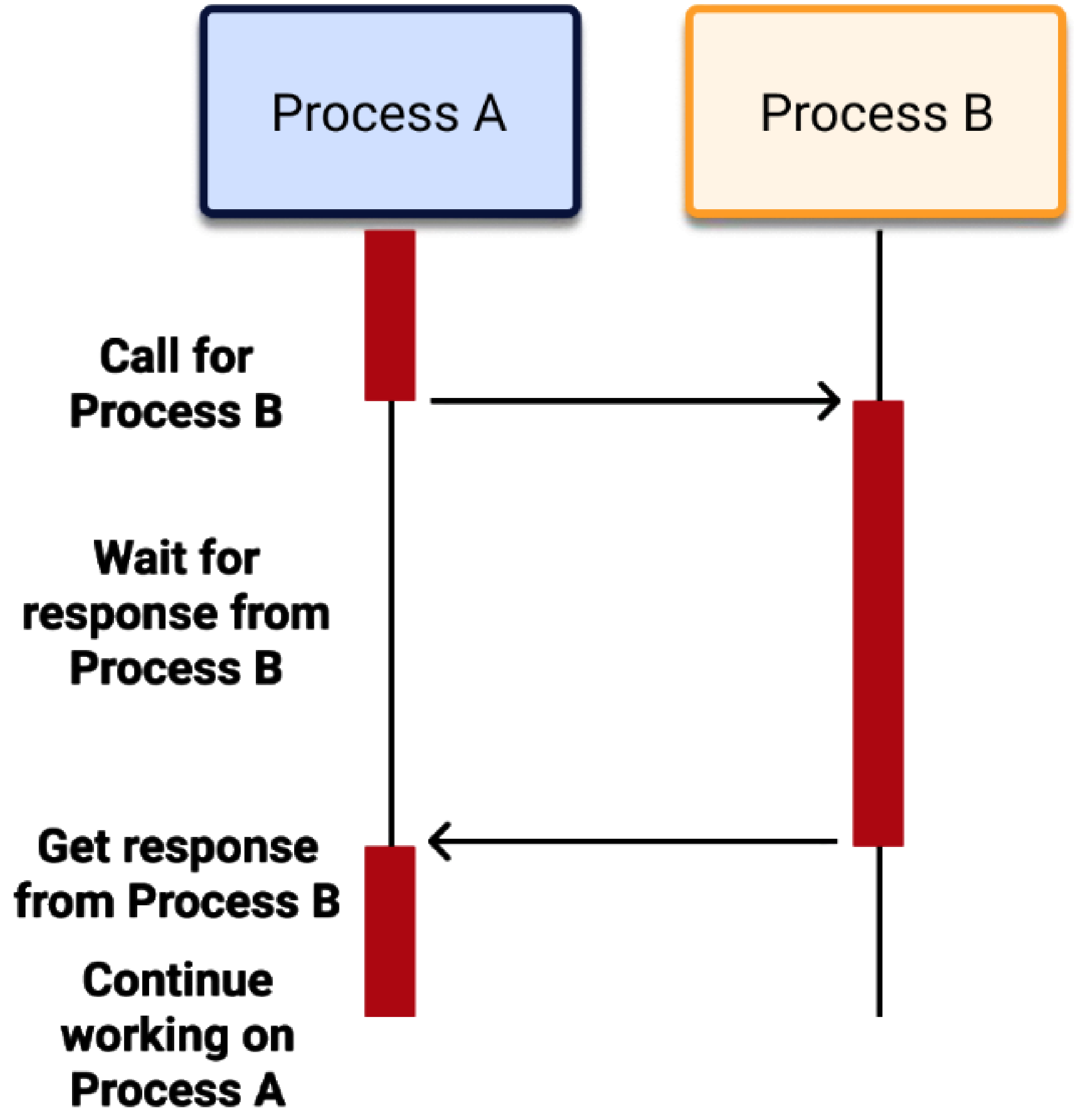
University of Wisconsin - Platteville

```
> function boilWater() {
      console.log("Boiling water...");
      return "Hot water";
    }

    function brewCoffee(water) {
      console.log(`Brewing coffee with ${water}`);
      return "Coffee";
    }

    function makeCoffee() {
      const water = boilWater();
      const coffee = brewCoffee(water);
      console.log(`Enjoy your ${coffee}`);
    }

    makeCoffee();
  Boiling water...                                    VM159:2
  Brewing coffee with Hot water                       VM159:7
  Enjoy your Coffee                                   VM159:14
<· undefined
>
```

Analyze output:

```
liveserver
python -m http.server 8888
```

Synchronous Processing


Process A
Process B
Call for Process B
Wait for response from Process B
Get response from Process B
Continue working on Process A

Abraham N. Aldaco-Gastélum, Ph.D.

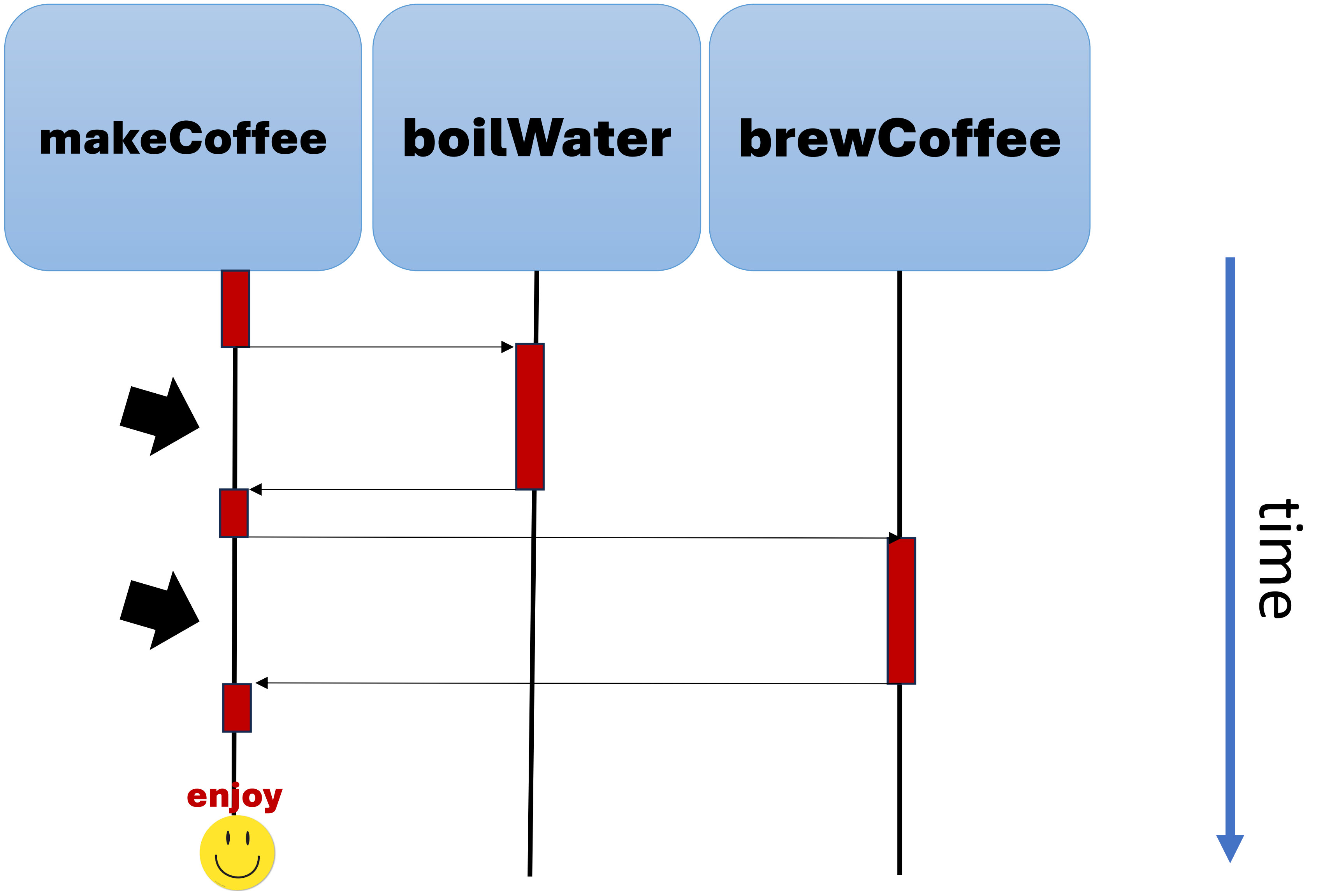
In our program there are several parts of code waiting their turn…

```
function boilWater() {
    console.log("Boiling water...");
    return "Hot water";
  }

  function brewCoffee(water) {
    console.log(`Brewing coffee with ${water}`);
    return "Coffee";
  }

  function makeCoffee() {
    const water = boilWater();

    const coffee = brewCoffee(water);

    console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
```

1 **brewCoffee(water)** waits to boilWater() to complete.

2 **console.log()** waits to brewCofee() to complete

Here, it is very evident that `brewCoffee()` waits for `boilWater()`



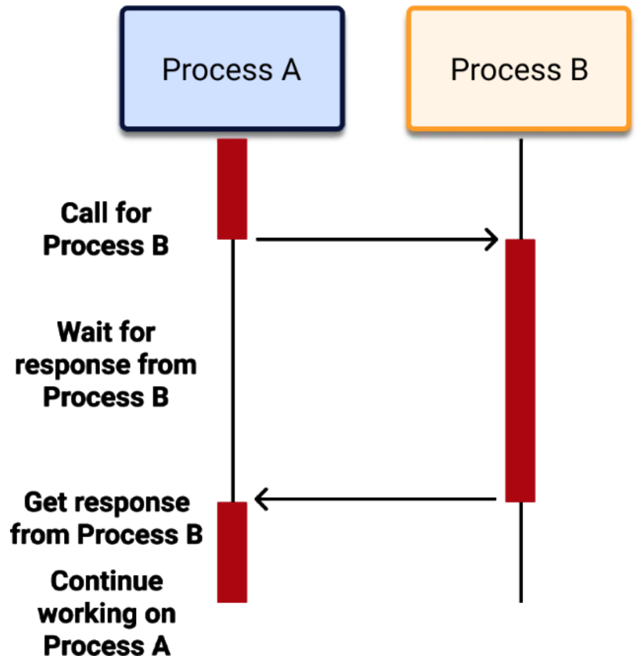
Is synchronous processing good or bad?

It is neither bad nor good,

It is necessary when operations depend on the result of the previous step

In Synchronous processing :

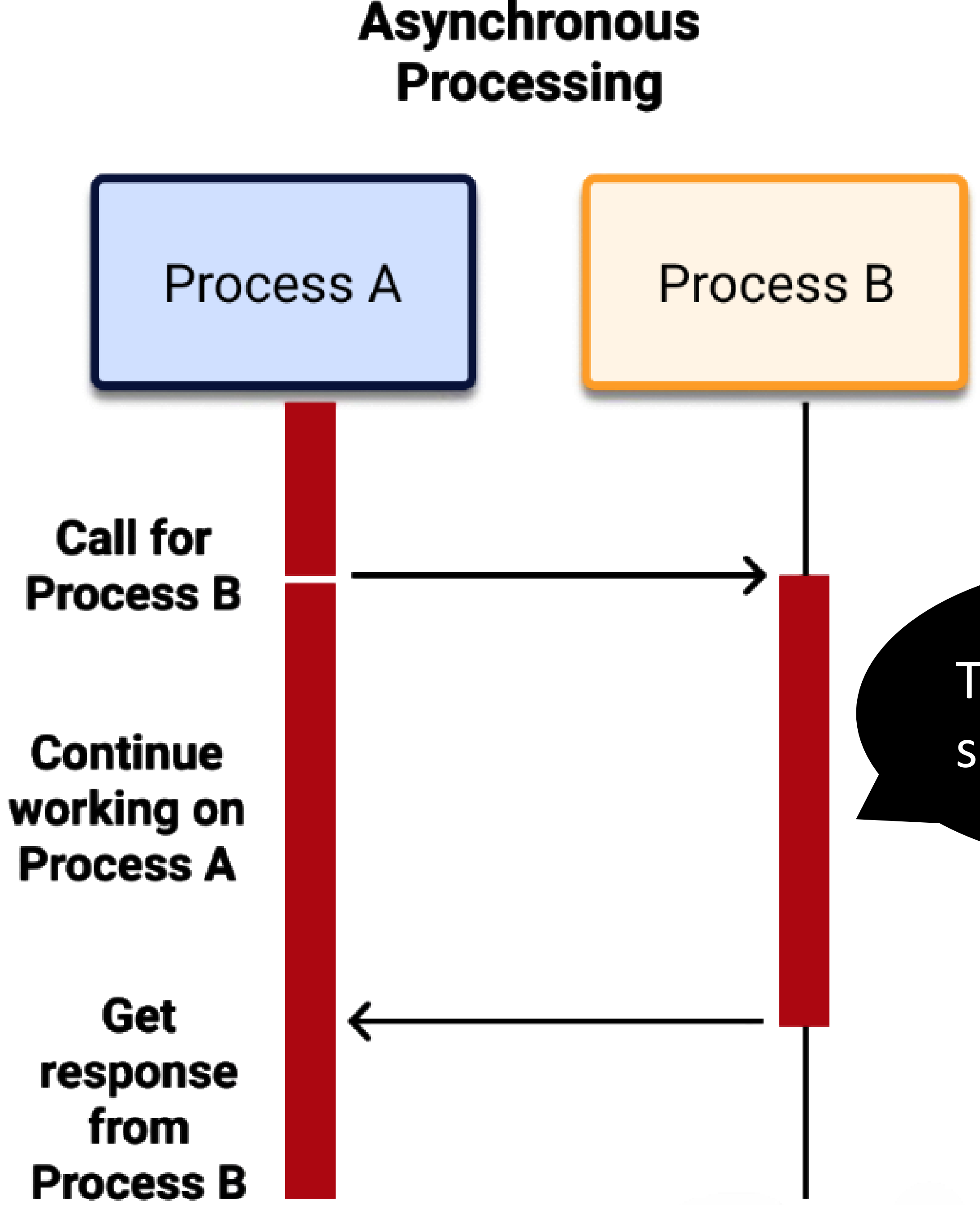
What happens if one of the functions takes too long? How would it affect the program?


Process A
Process B
Call for Process B
Wait for response from Process B
Get response from Process B
Continue working on Process A

Asynchronous Processing

In Asynchronous processing we can continue the execution of other processes

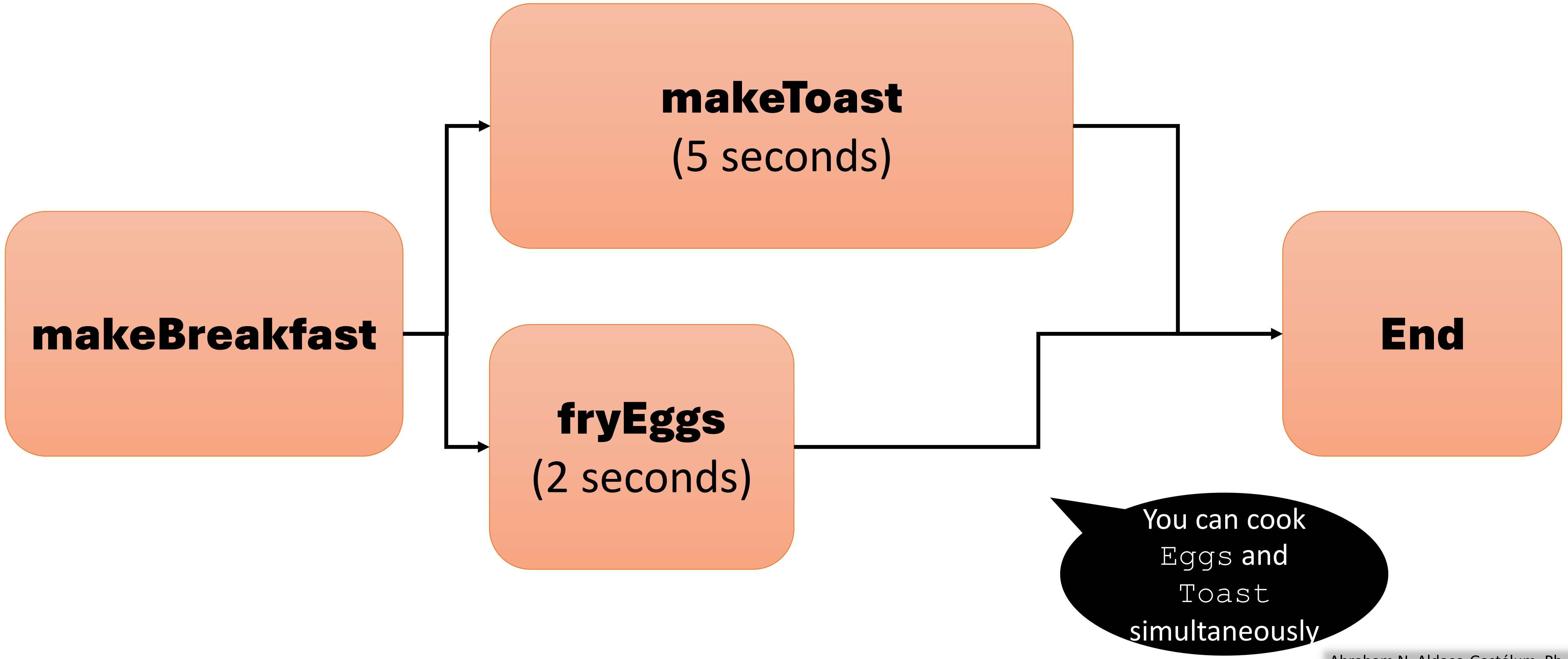
(non blocking processing)



Observe what is happening with `Process A` while `Process B` is running.

They can run simultaneously

Let’s simulate two processes that can be executed without dependency on each other:



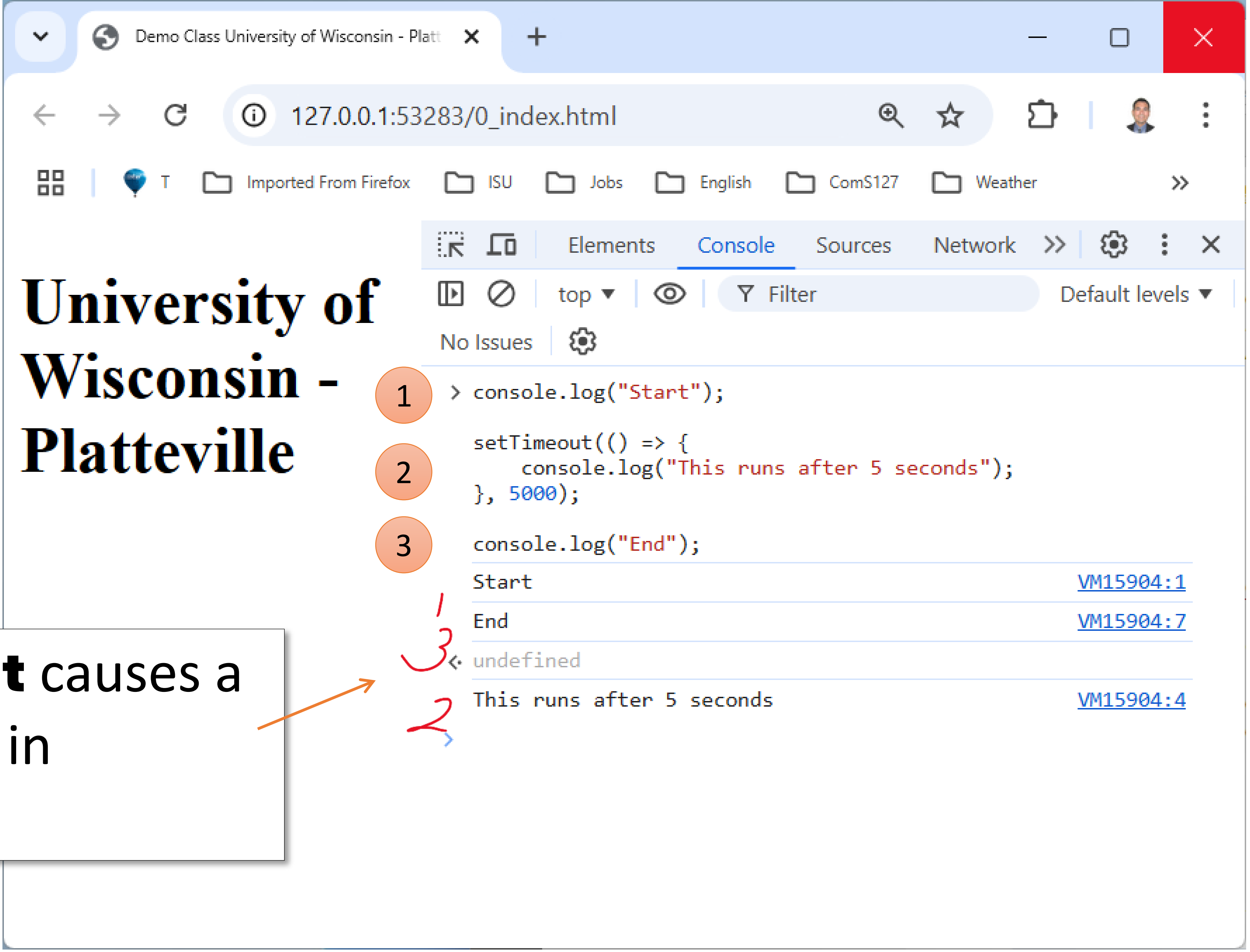
With the help of `setTimeout` we can simulate **delay** in execution :

1
```
console.log("Start");
```

2
```
setTimeout(() => {
    console.log("This runs after 5 seconds");
}, 5000);
```

3
```
console.log("End");
```

Try code in a browser:



```
> console.log("Start");

  setTimeout(() => {
      console.log("This runs after 5 seconds");
  }, 5000);

  console.log("End");
  Start                                  VM15904:1
  End                                    VM15904:7
< undefined
  This runs after 5 seconds              VM15904:4
```

setTimeout causes a 5 secs delay in execution :

```
liveserver
python -m http.server 8888
```

Simulate two processes running simultaneously using `setTimeout()`:

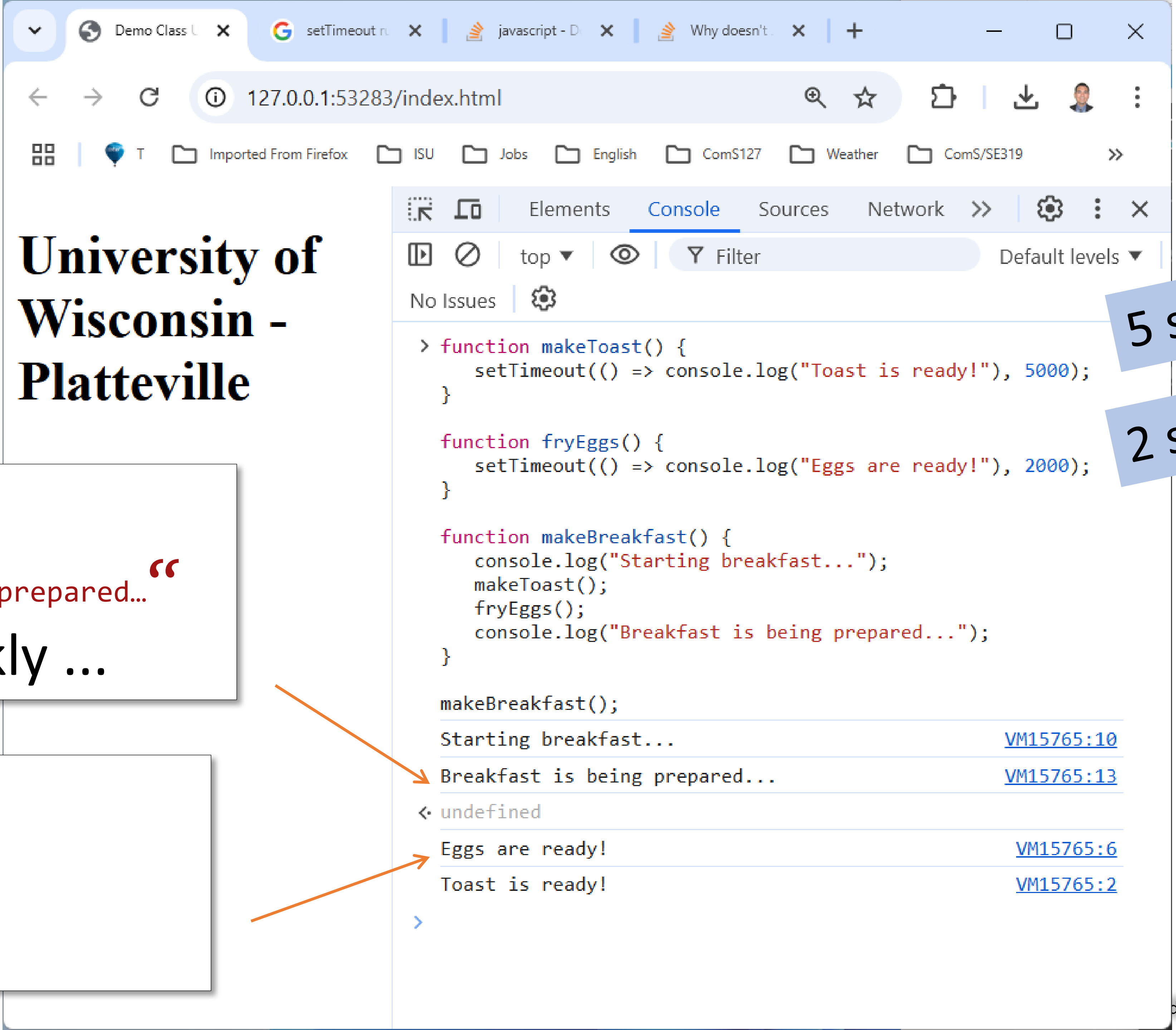
```
function makeToast() {
    setTimeout(() => console.log("Toast is ready!"), 5000);
}
```

5 seconds

```
function fryEggs() {
    setTimeout(() => console.log("Eggs are ready!"), 2000);
}
```

2 seconds

Try code in a browser:



University of Wisconsin - Platteville

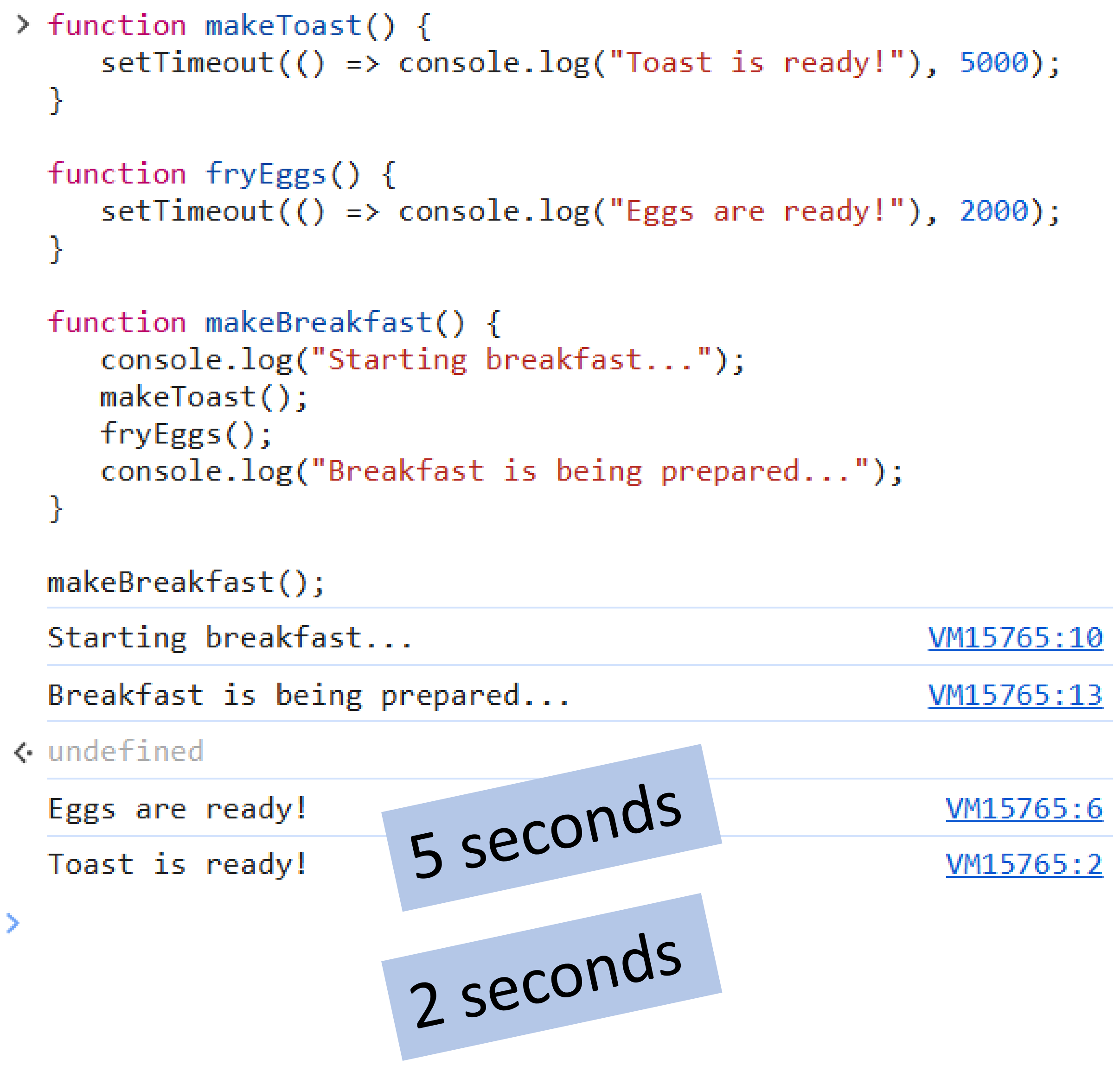
```
> function makeToast() {
     setTimeout(() => console.log("Toast is ready!"), 5000);
  }

  function fryEggs() {
     setTimeout(() => console.log("Eggs are ready!"), 2000);
  }

  function makeBreakfast() {
     console.log("Starting breakfast...");
     makeToast();
     fryEggs();
     console.log("Breakfast is being prepared...");
  }

  makeBreakfast();
  Starting breakfast...                                VM15765:10
  Breakfast is being prepared...                       VM15765:13
<· undefined
  Eggs are ready!                                      VM15765:6
  Toast is ready!                                      VM15765:2
>
```

5 seconds

2 seconds

Observe that "breakfast is being prepared…" is shown quickly ...

and "fryEggs()" finishes first.



```
> function makeToast() {
     setTimeout(() => console.log("Toast is ready!"), 5000);
  }

  function fryEggs() {
     setTimeout(() => console.log("Eggs are ready!"), 2000);
  }

  function makeBreakfast() {
     console.log("Starting breakfast...");
     makeToast();
     fryEggs();
     console.log("Breakfast is being prepared...");
  }

  makeBreakfast();
  Starting breakfast...                                 VM15765:10
  Breakfast is being prepared...                        VM15765:13
<· undefined
  Eggs are ready!                                        VM15765:6
  Toast is ready!                                        VM15765:2
>
```

Here, preparing **Eggs** and **Toast** simultaneously is not a problem …

It is delicious !

Abraham N. Aldaco-Gastélum, Ph.D.

Situation in Asynchronous Processing

There are situations where some processes must wait one intermediate result, but we can continue the execution of other processes

(non blocking processing)

Scenario :

If you need to download a large volume of data from a server, which can take a considerable amount of time, it would be inefficient for your program or function to freeze while waiting for the data to be fetched.

Instead, it is common practice to run the **fetching** operation in the background.

Hypothetical situation :

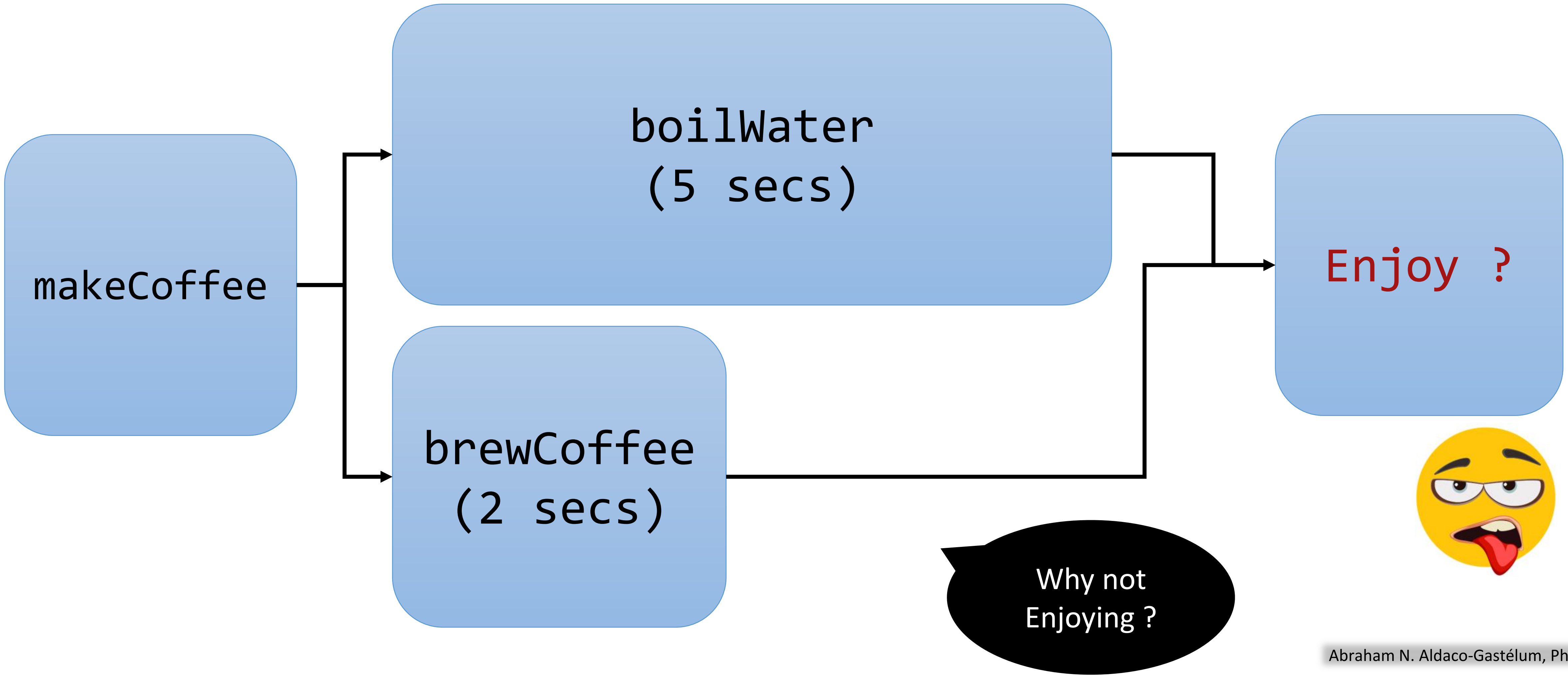
If **processA**() is in charge of loading the data from an external source,

and **processB**() in charge of executing on the data (filter, sort, select, etc),

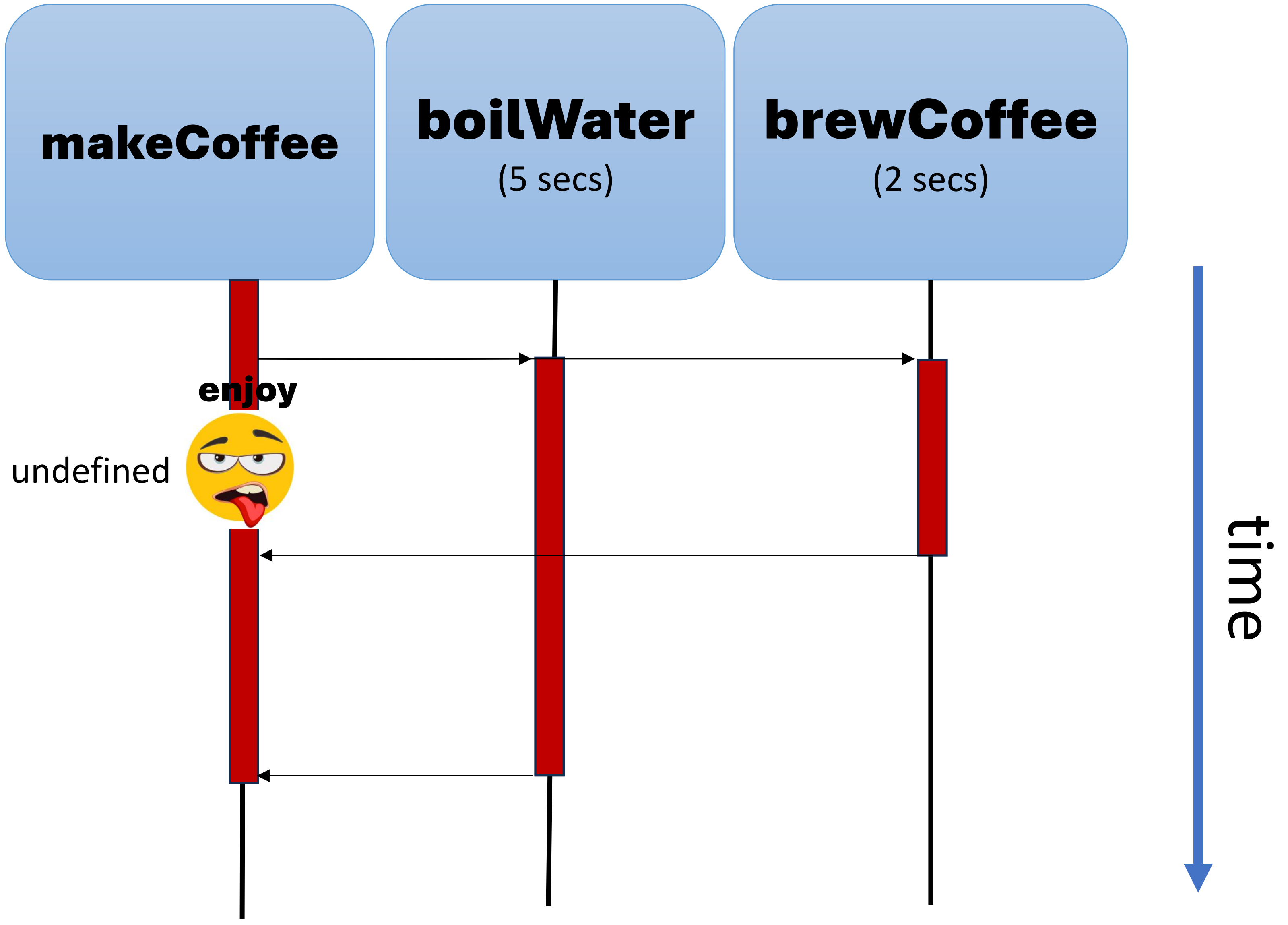
but **processB**() is executed before **processA**() completes,

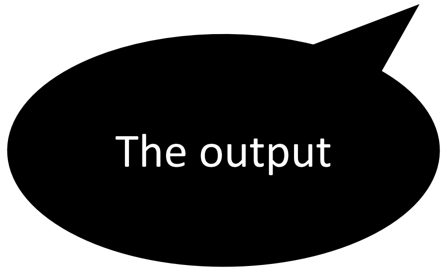
an err could happen.

Consider our previous synchronous example preparing coffee, This time, execute the processes **boilWater**() and **brewCoffee**() Asynchronously :

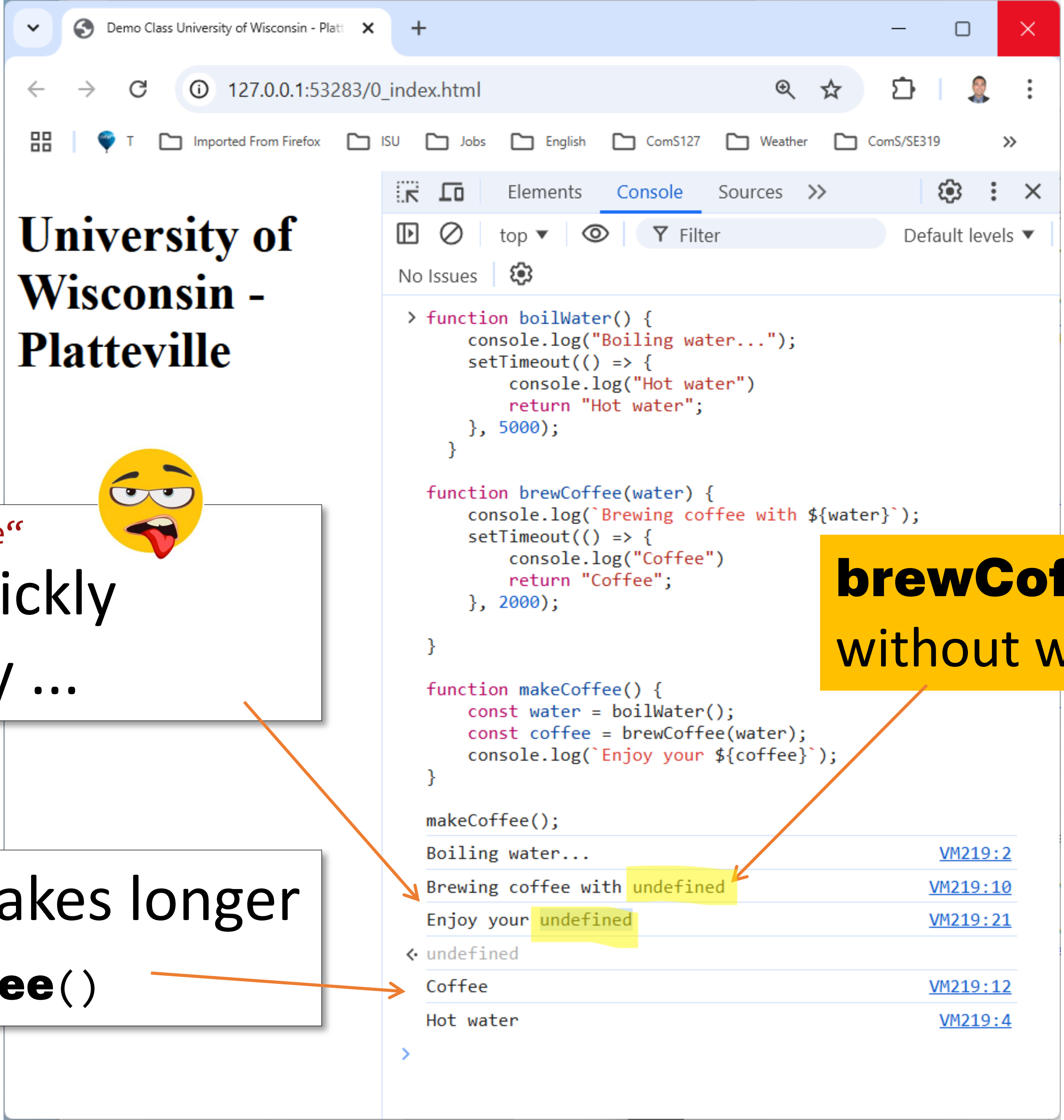


Now, `brewCoffee`() finishes before `boilWater()` and the coffee is not enjoyed.





Try code in a browser:



University of Wisconsin - Platteville

```
> function boilWater() {
      console.log("Boiling water...");
      setTimeout(() => {
          console.log("Hot water")
          return "Hot water";
      }, 5000);
  }

  function brewCoffee(water) {
      console.log(`Brewing coffee with ${water}`);
      setTimeout(() => {
          console.log("Coffee")
          return "Coffee";
      }, 2000);

  }

  function makeCoffee() {
      const water = boilWater();
      const coffee = brewCoffee(water);
      console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
  Boiling water...                       VM219:2
  Brewing coffee with undefined          VM219:10
  Enjoy your undefined                   VM219:21
< undefined
  Coffee                                 VM219:12
  Hot water                              VM219:4
>
```

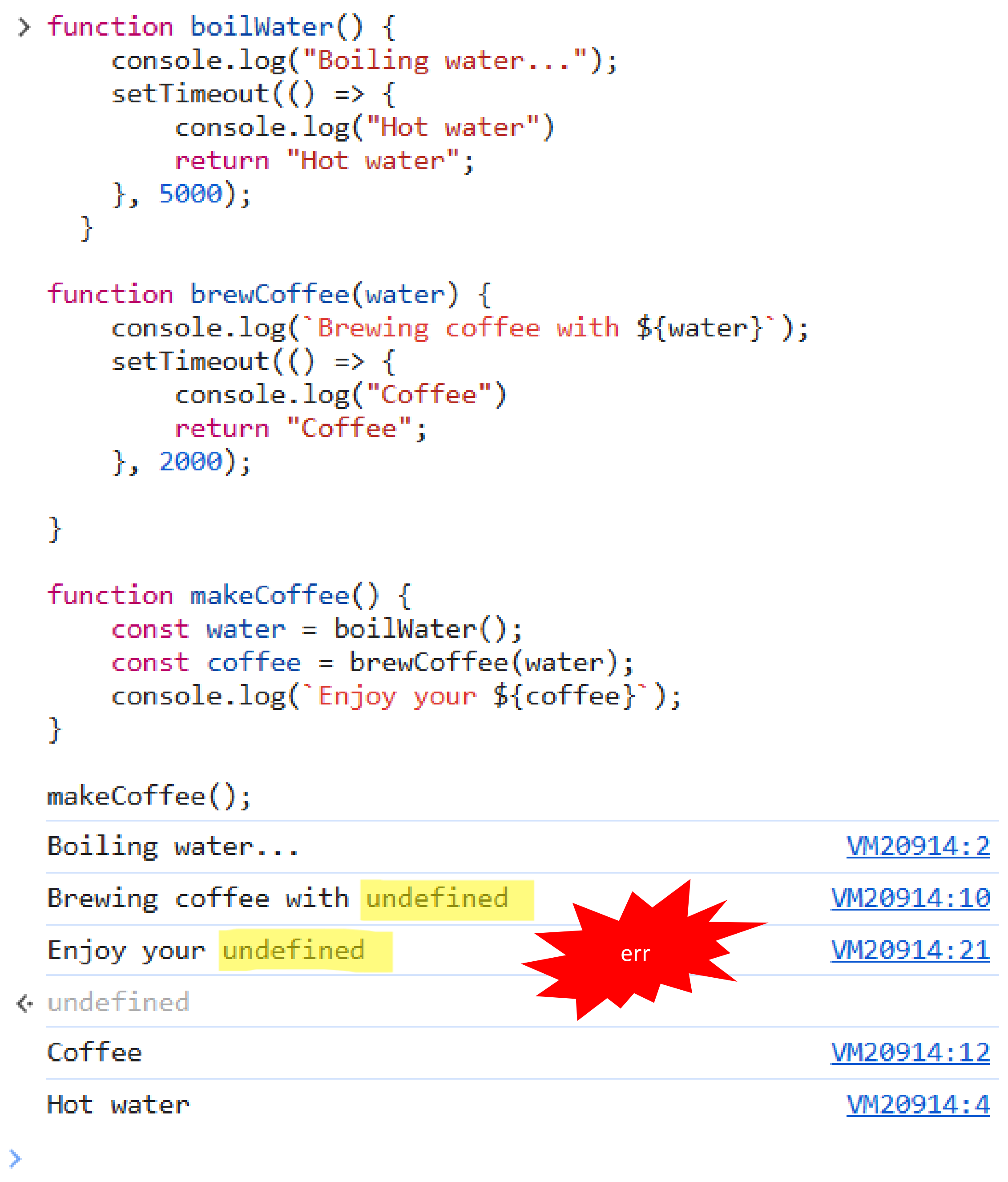
“Enjoy your Coffee“ is shown quickly immediately …

brewCoffee(water) executes without waiting “Hot water”

boilerWater() takes longer than **brewCoffee**()

Not what we want

Abraham N. Aldaco-Gastélum, Ph.D.



```
> function boilWater() {
      console.log("Boiling water...");
      setTimeout(() => {
          console.log("Hot water")
          return "Hot water";
      }, 5000);
    }

  function brewCoffee(water) {
      console.log(`Brewing coffee with ${water}`);
      setTimeout(() => {
          console.log("Coffee")
          return "Coffee";
      }, 2000);

  }

  function makeCoffee() {
      const water = boilWater();
      const coffee = brewCoffee(water);
      console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
  Boiling water...                                  VM20914:2
  Brewing coffee with undefined                     VM20914:10
  Enjoy your undefined                              VM20914:21
< undefined
  Coffee                                            VM20914:12
  Hot water                                         VM20914:4
>
```

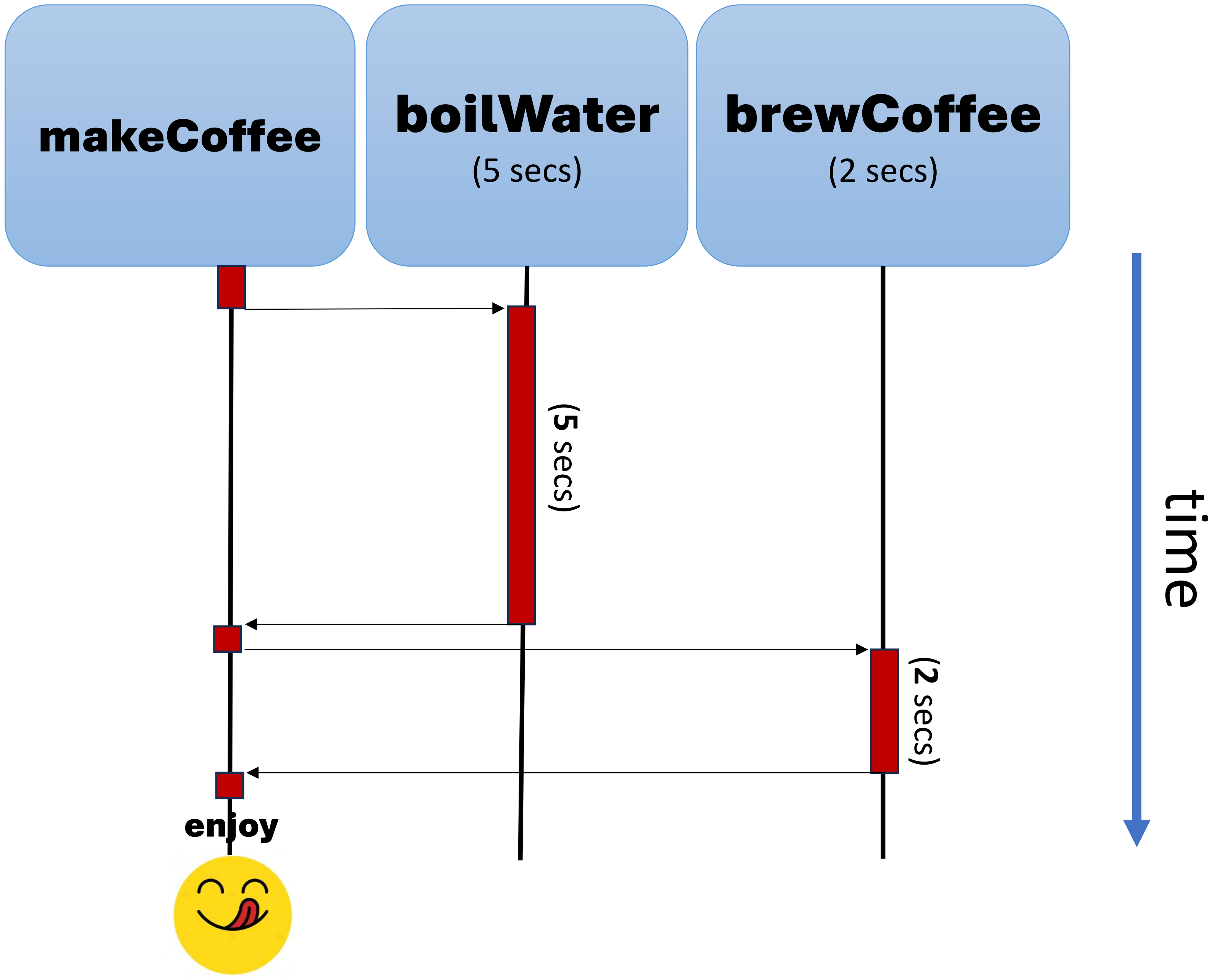
Why is `water` undefined ?

Same with `coffee`?

Solution
`async` / `await`

Let’s make **brewCoffee**() wait for **boilWater**() to finish

Let's make **brewCoffee()** wait for **boilWater()** to finish.



Let’s make **brewCoffee**() wait for **boilWater**() to finish by using `async` / `await` :

```
async function makeCoffee() {
    console.log("Starting coffee...");
    const water = await boilWater();
    const coffee = await brewCoffee(water);
    console.log(`Enjoy your ${coffee}`);
}
```

Some statements are inherently **promises** :

```
fetch("https://fakestoreapi.com/products")
  .then(response => response.json())
  .then(data => console.log(data));
```

With the help of `promise` **we can handle asynchronous processing in regular functions**:

```
async function boilWater() {
    console.log("Boiling water...");
    return new Promise((resolve) =>
      setTimeout(() => {
        console.log("Hot water");
        resolve("Hot water");;
      }, 5000)
    );
}
```

Promise()

setTimeout()

Promise

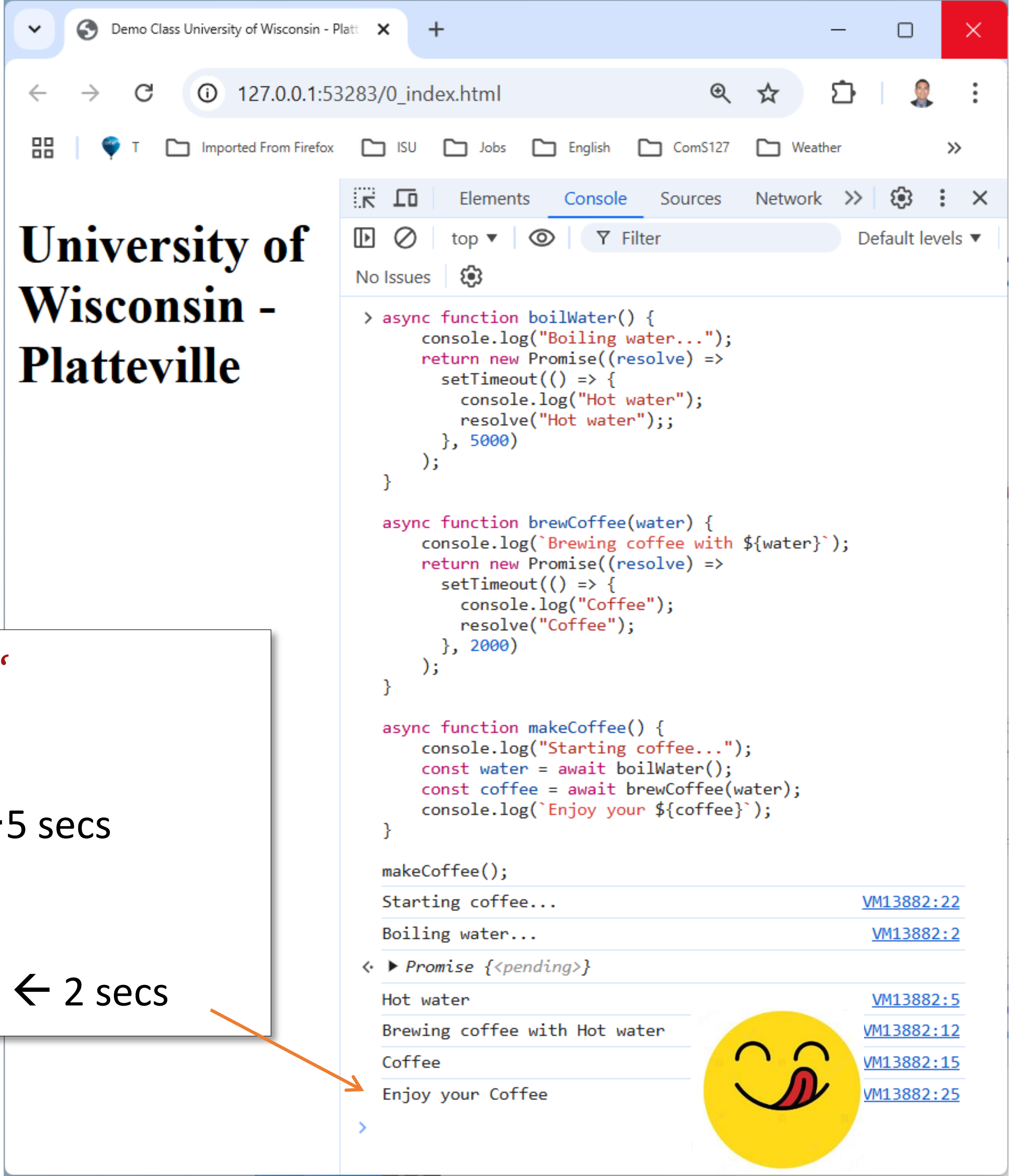
A promise may produce a value some time in the future.

That is, instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

Complete solution using Promise-based setTimeout and async/await:

```
async function boilWater() {
    console.log("Boiling water...");
    return new Promise((resolve) =>
      setTimeout(() => {
        console.log("Hot water");
        resolve("Hot water");;
      }, 5000)
    );
}

async function brewCoffee(water) {
    console.log(`Brewing coffee with ${water}`);
    return new Promise((resolve) =>
      setTimeout(() => {
        console.log("Coffee");
        resolve("Coffee");
      }, 2000)
    );
}

async function makeCoffee() {
    console.log("Starting coffee...");
    const water = await boilWater();
    const coffee = await brewCoffee(water);
    console.log(`Enjoy your ${coffee}`);
}

makeCoffee();
```

Try code in a browser:



University of Wisconsin - Platteville

```
> async function boilWater() {
      console.log("Boiling water...");
      return new Promise((resolve) =>
        setTimeout(() => {
          console.log("Hot water");
          resolve("Hot water");;
        }, 5000)
      );
  }

  async function brewCoffee(water) {
      console.log(`Brewing coffee with ${water}`);
      return new Promise((resolve) =>
        setTimeout(() => {
          console.log("Coffee");
          resolve("Coffee");
        }, 2000)
      );
  }

  async function makeCoffee() {
      console.log("Starting coffee...");
      const water = await boilWater();
      const coffee = await brewCoffee(water);
      console.log(`Enjoy your ${coffee}`);
  }

  makeCoffee();
  Starting coffee...                                  VM13882:22
  Boiling water...                                    VM13882:2
< ▶ Promise {<pending>}
  Hot water                                           VM13882:5
  Brewing coffee with Hot water                       VM13882:12
  Coffee                                              VM13882:15
  Enjoy your Coffee                                   VM13882:25
>
```

"Enjoy your Coffee"
is shown after
boilWater() ←5 secs
and
brewCoffee() ← 2 secs

Abraham N. Aldaco-Gastélum, Ph.D.

Asynchronous processing

fetch

So far :

- We have use `setTimeout` to simulate processing delay.
- Then, we add `Promise` to use `async`/`await` and have control over asynchronous processing.

Fetch :

- It is a built-in function that allows you to make **network requests** to retrieve data from a server or API.
- It is considered a **promise-based** function.
- And it executes asynchronously.

If we use fetch incorrectly, we may have error:

```
function fetchData(){
    fetch("./data.json")
    .then(response=>response.json())
    .then(data => console.log(data));

    for (const person of data)
        console.log(person.firstName);
}

fetchData();
```

{JSON}
JavaScript Object Notation

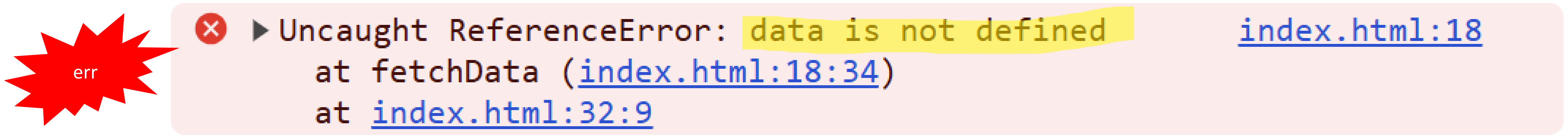
```
[
    {
        "firstName":"Abraham",
        "lastName" : "Aldaco"
    },
    {
        "firstName":"John",
        "lastName" : "Doe"
    },
    {
        "firstName":"Clark",
        "lastName" : "Kent"
    }
]
```

err

```
▶ Uncaught ReferenceError: data is not defined        index.html:18
    at fetchData (index.html:18:34)
    at index.html:32:9
```

```
function fetchData(){
    fetch("./data.json")
    .then(response=>response.json())
    .then(data => console.log(data));

    for (const person of data)
        console.log(person.firstName);
}

fetchData();
```

Why is data not defined ?



err

```
✕ ▸Uncaught ReferenceError: data is not defined        index.html:18
      at fetchData (index.html:18:34)
      at index.html:32:9
```

Abraham N. Aldaco-Gastélum, Ph.D.

Explanation of the error :

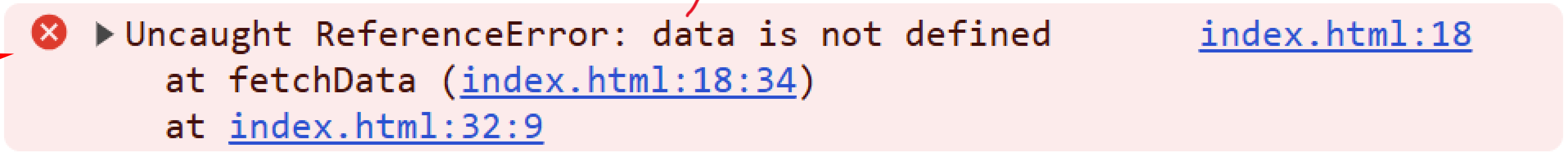
execution

```
function fetchData(){
    fetch("./data.json")
    .then(response=>response.json())
    .then(data => console.log(data));

    for (const person of data)
        console.log(person.firstName);
}

fetchData();
```

`fetch` and `console.log` work fine.

But the `for` statement is executed **without waiting** for the `fetch` to finish.

err

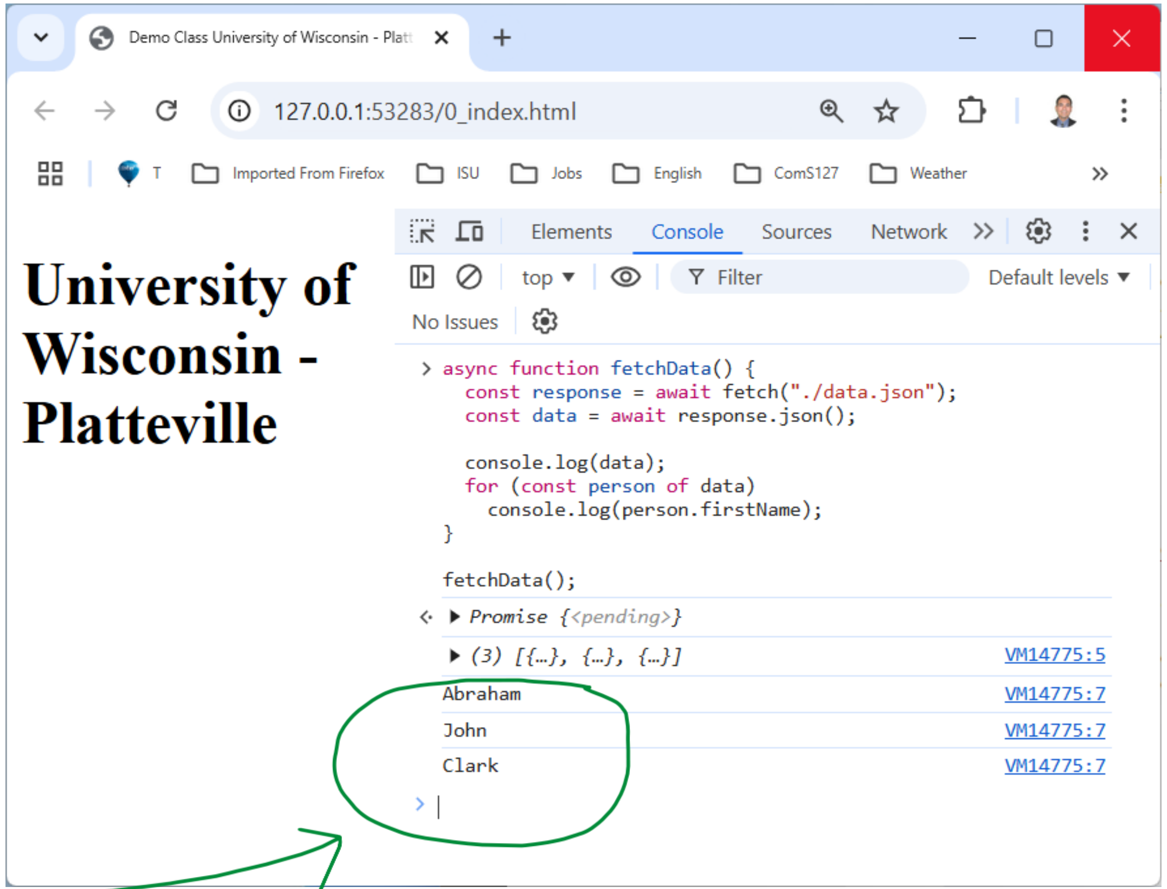


```
▶Uncaught ReferenceError: data is not defined        index.html:18
    at fetchData (index.html:18:34)
    at index.html:32:9
```

Solving the issue adding async/await:

execution

```
async function fetchData(){
    const response = await fetch("./data.json");
    const data = await response.json();
    console.log(data);

    for (const person of data)
        console.log(person.firstName);
}

fetchData();
```



University of Wisconsin - Platteville

```
> async function fetchData() {
    const response = await fetch("./data.json");
    const data = await response.json();

    console.log(data);
    for (const person of data)
      console.log(person.firstName);
  }

  fetchData();
<· ▶ Promise {<pending>}
   ▶ (3) [{…}, {…}, {…}]                    VM14775:5
  Abraham                                   VM14775:7
  John                                      VM14775:7
  Clark                                     VM14775:7
>
```

Abraham N. Aldaco-Gastélum, Ph.D.

Assignment :

Convert to `async`/`await` the next code using `Promise` and `setTimeout`:

Assignment

```
Area of the room: 150 square units
Total flooring cost: $750
```

```
// Function to calculate the area of a rectangle
function calculateArea(length, width) {
    return length * width;
}

// Function to calculate the cost of flooring based on area
function calculateFlooringCost(area, costPerSquareUnit) {
    return area * costPerSquareUnit;
}

function flooringCost() {

    const length = 10;              // Length of the room
    const width = 15;               // Width of the room
    const costPerSquareUnit = 5; // Cost per square unit

    // Step 1: Calculate the area
    const area = calculateArea(length, width);
    console.log(`Area of the room: ${area} square units`);

    // Step 2: Calculate the flooring cost based on the area
    const totalCost = calculateFlooringCost(area, costPerSquareUnit);
    console.log(`Total flooring cost: $${totalCost}`);
}

flooringCost();
```

Thanks !

Questions ?

Backup

Code

https://github.com/aaldacog/uwp

Promise

A Promise is an object representing the eventual completion or failure of an Asynchronous operation.

Reference:
https://www.mitrais.com/news-updates/asynchronous-in-javascript/
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Promise

A promise is an object that may produce a single value some time in the future:

either a resolved value, or a reason that it’s not resolved (e.g., a network error occurred).

if the promise succeed

if the promise fails

Promise

instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

1 Let's to execute a **resolve** promise :

```
new Promise(function(resolve, reject) {
    // the function is executed automatically when the promise is constructed
    // after 1 second signal that the job is done with the result "done“
     setTimeout(() => resolve("done"), 1000);
});
```

The Promise executed successfully.
And the result **value** is ‘done’

Reference:
https://javascript.info/promise-basics

Servers :

```
liveserver

python –m http.server 8888
```