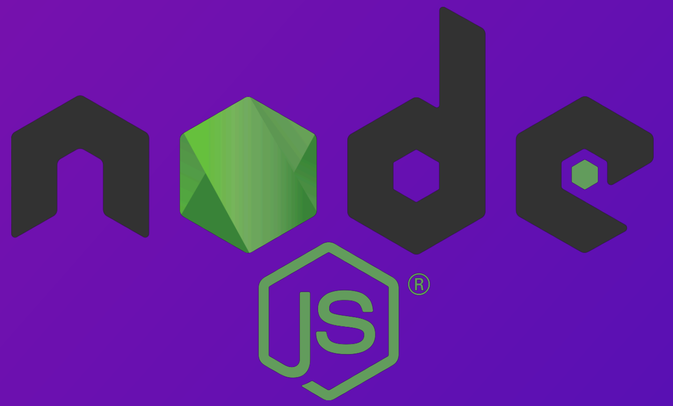
Visualize Future

How to avoid Memory Leak In Javascript Programming

nguyentvk@uiza.io

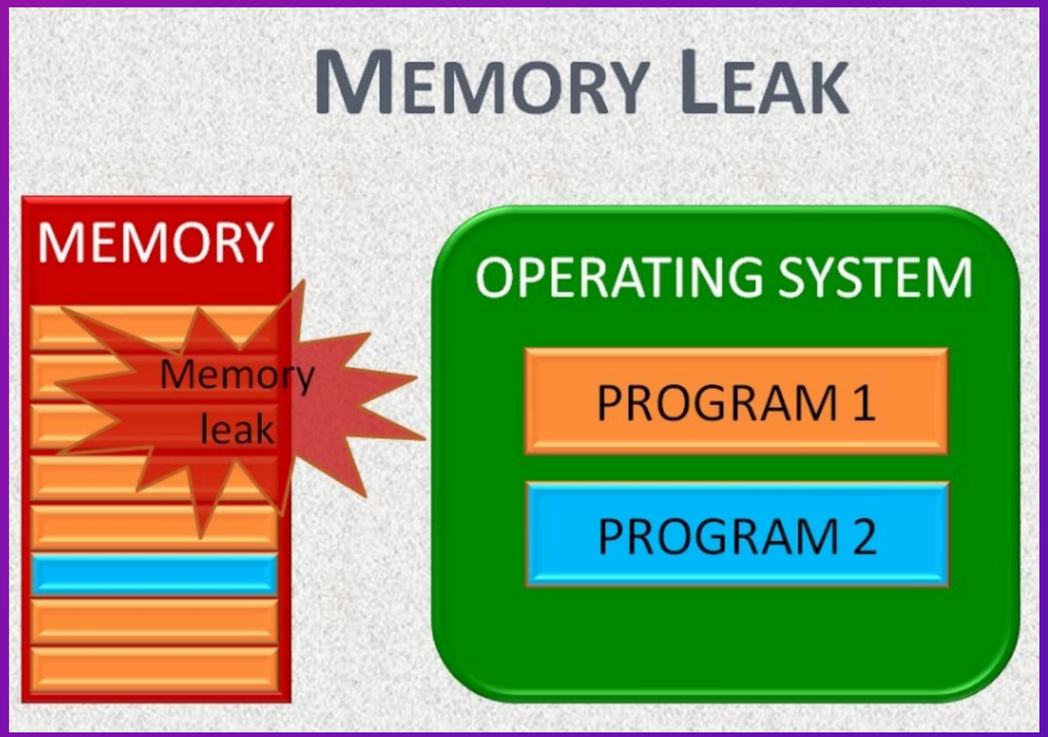
11-12-2018

- Introduction
- What is memory leak?
- Memory management in Javascript
- 4 types of memory leaks
- How to detect memory leaks
- Reference



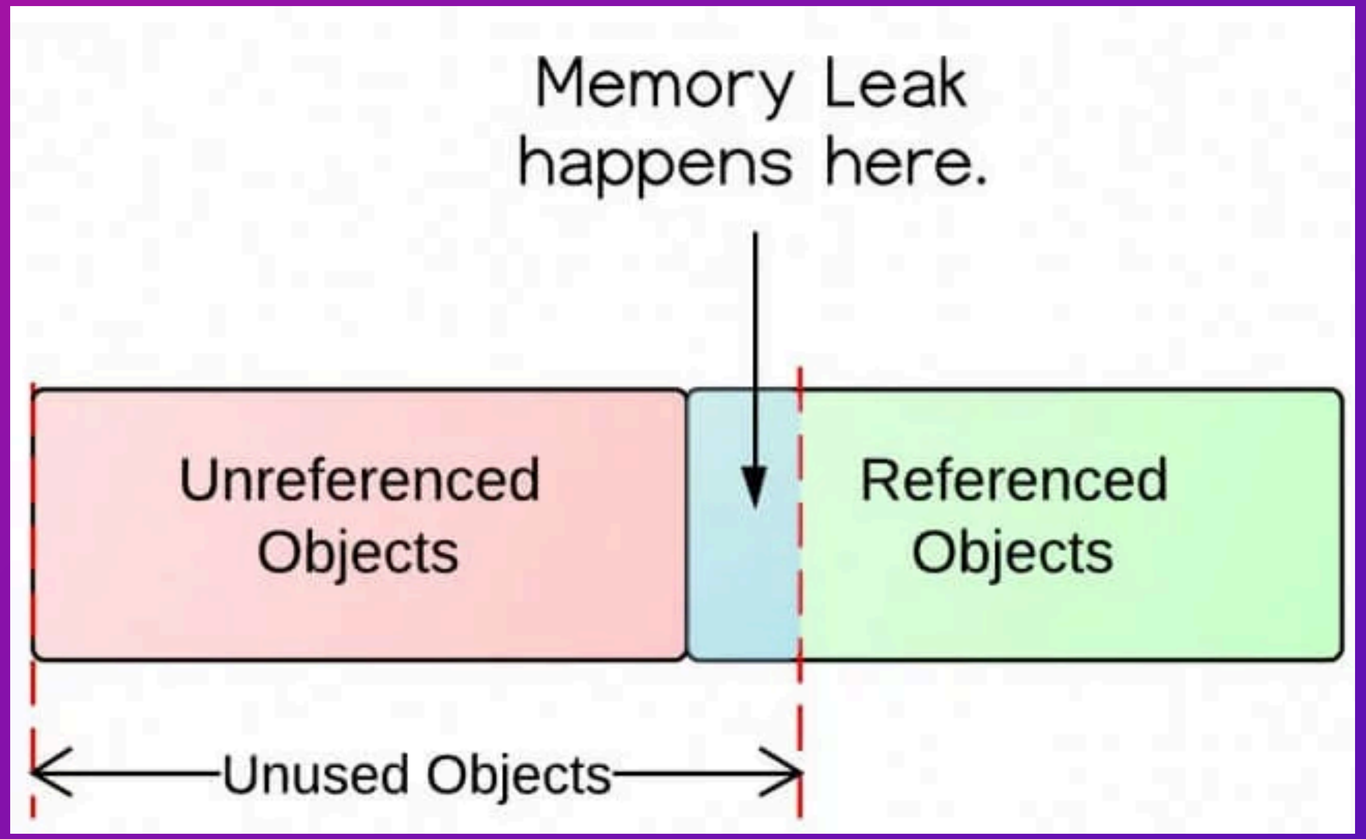
Introduction

- Memory leak is very common in any application: NodeJs, Java, Android, Php, …
- Leaks are the cause of whole class of problems: slowdowns, crashes, high latency and even problems with other applications
- Memory leaks are a problem every developer has to face eventually



What is memory leak ?

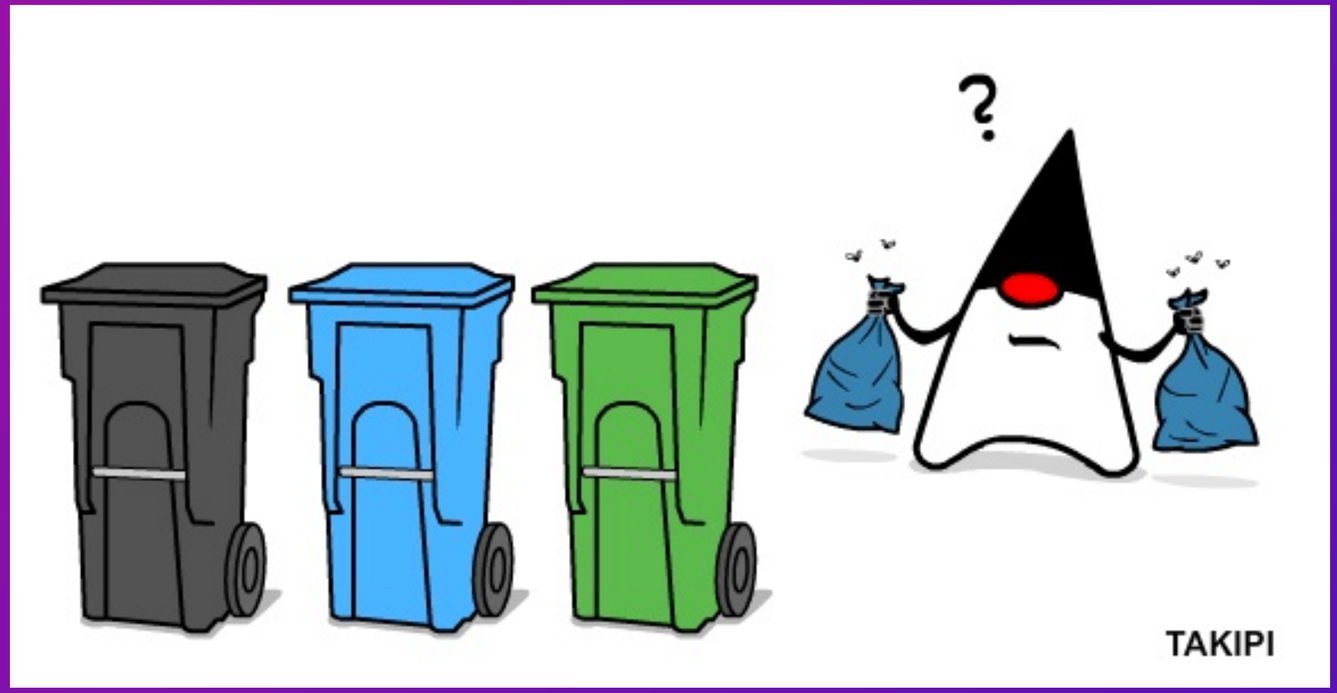
- Memory leaks can be defined as memory that is not required by an application anymore that for some reason is not returned to the operating system or the pool of free memory



Memory management in Nodejs

- JavaScript is one of the so called garbage collected languages

- Garbage collected languages help developers manage memory

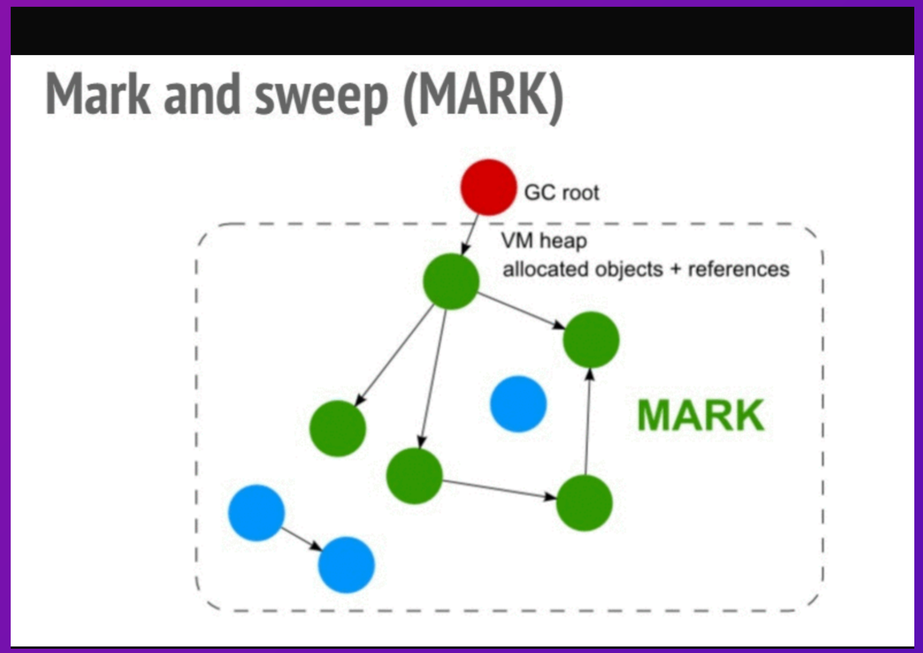


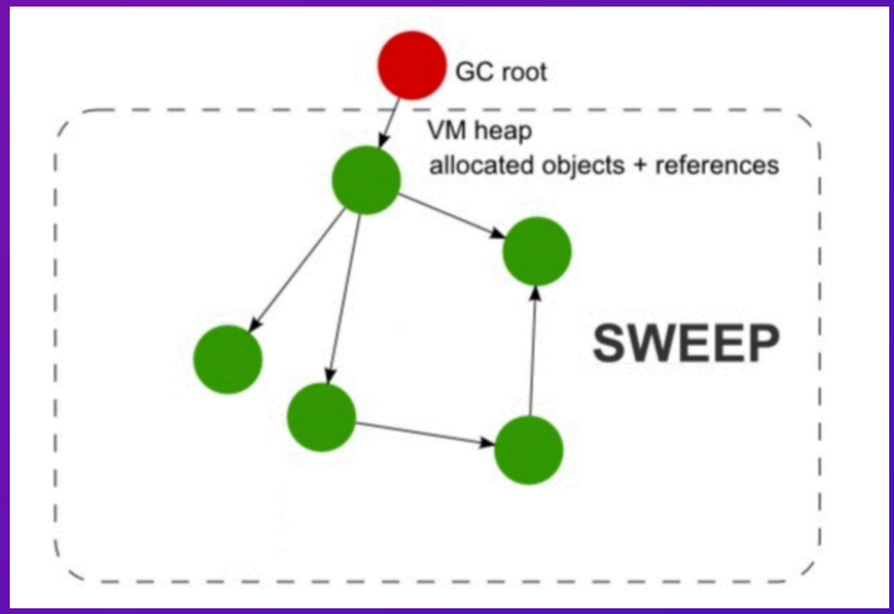
Memory management in Nodejs

- The main cause for leaks in garbage collected languages are unwanted references

- Most garbage collectors use an algorithm known as mark-and-sweep. The algorithm consists of the following steps:

    1. The garbage collector builds a list of "roots". The "root" window object is always present. so it can be not garbage)
    2. All roots are inspected and marked as active. All children are inspected recursively as well.
    3. All pieces of memory not marked as active can now be considered garbage. The collector can now free that memory and return it to the OS





4 types of memory leaks

Accidental global variables

```
> function foo(arg){ bar = arg; }
<· undefined
> foo("hello")
<· undefined
> window.bar
<· "hello"
```

```
> function foo(arg){ this.bar = arg; }
<· undefined
> foo("hello world")
<· undefined
> window.bar
<· "hello world"
```

Global variable can not be garbage

Solution

- use “use strict”
- Define variable with scope: var, let
- Set variable to null or reset after use

4 types of memory leaks

Forgotten timers or callbacks

```
var someResource = getData();
setInterval(function() {
    var node = document.getElementById('Node');
    if(node) {
        // Do stuff with node and someResource.
        node.innerHTML = JSON.stringify(someResource));
    }
}, 1000);
```

```
export class TestComponent {

  ngOnInit() {
    this.form = new FormGroup({...});
    this.valueChanges  = this.form.valueChanges.subscribe(console.log);
    this.statusChanges = this.form.statusChanges.subscribe(console.log);
  }

  ngOnDestroy() {
    this.valueChanges.unsubscribe();
    this.statusChanges.unsubscribe();
  }

}
```

- clear Interval when something done or unuseful

- Angular 2+ unsubscribe inside the ngOnDestroy lifecycle hook

4 types of memory leaks

Out of DOM references

Button has been removed by removeButton function, but it also has been reference by elements variable

So it can’t be release by Gargabe collection

```
var elements = {
  button: document.getElementById('button'),
};
function doStuff() {
  button.click();
  // Much more logic
}
function removeButton() {
  // The button is a direct child of body.
  document.body.removeChild(document.getElementById('button'));
  // At this point, we still have a reference to #button in the global
  // elements dictionary. In other words, the button element is still in
  // memory and cannot be collected by the GC.
}
```

4 types of memory leaks

Closures

- The variable unused holds a closure that has a reference to originalThing (theThing from the previous call to replaceThing)

- Unused is never used, its reference to originalThing forces it to stay active (prevents its collection)

```
var theThing = null;
var replaceThing = function () {
  var originalThing = theThing;
  var unused = function () {
    if (originalThing)
      console.log("hi");
  };
  theThing = {
    longStr: new Array(1000000).join('*'),
    someMethod: function () {
      console.log(someMessage);
    }
  };
};
setInterval(replaceThing, 1000);
```

How to detect memory leaks

There’re many tools and libraries used to detect memory leaks in NodeJS

- require('memwatch-next')
The memwatch-next module is great for detecting memory leaks
https://www.npmjs.com/package/memwatch-next

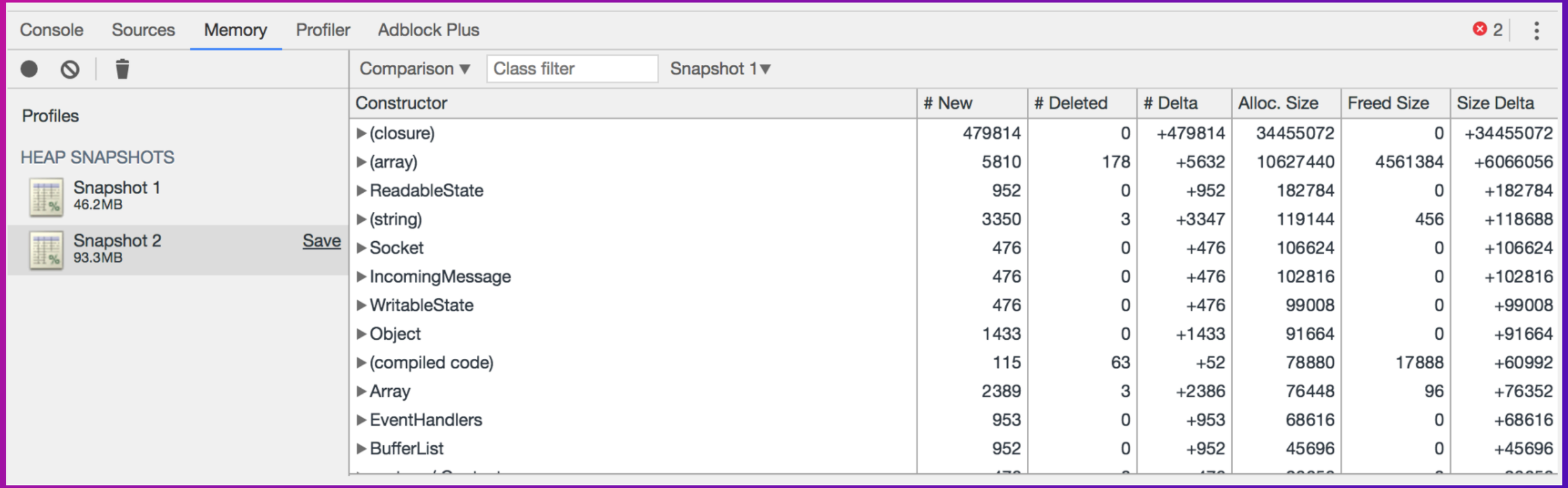

```
memwatch.on('leak', function(info) { ... });
```

A leak event will be emitted when your heap usage has increased for five consecutive garbage collections

```
{ start: Fri, 29 Jun 2012 14:12:13 GMT,
  end: Fri, 29 Jun 2012 14:12:33 GMT,
  growth: 67984,
  reason: 'heap growth over 5 consecutive GCs (20s) - 11.67 mb/hr' }
```

How to detect memory leaks

Record and compare two between snapshot



Constructor	# New	# Deleted	# Delta	Alloc. Size	Freed Size	Size Delta
▶(closure)	479814	0	+479814	34455072	0	+34455072
▶(array)	5810	178	+5632	10627440	4561384	+6066056
▶ReadableState	952	0	+952	182784	0	+182784
▶(string)	3350	3	+3347	119144	456	+118688
▶Socket	476	0	+476	106624	0	+106624
▶IncomingMessage	476	0	+476	102816	0	+102816
▶WritableState	476	0	+476	99008	0	+99008
▶Object	1433	0	+1433	91664	0	+91664
▶(compiled code)	115	63	+52	78880	17888	+60992
▶Array	2389	3	+2386	76448	96	+76352
▶EventHandlers	953	0	+953	68616	0	+68616
▶BufferList	952	0	+952	45696	0	+45696

- benchmarking tool written in node https://www.npmjs.com/package/autocannon
autocannon -c 1 -d 60 http://localhost:1337

- node --expose-gc --inspect testMemoryLeak.js

Demonstrate

Memory leak then crash application

Memory leak with garbage collection

Snapshot memory leak

Reference

How to Self Detect a Memory Leak in Node https://www.nearform.com/blog/how-to-self-detect-a-memory-leak-in-node

4 Types of Memory Leaks in JavaScript and How to Get Rid Of Them
https://auth0.com/blog/four-types-of-leaks-in-your-javascript-code-and-how-to-get-rid-of-them/

Memory Leaks in NodeJS
https://medium.com/tech-tajawal/memory-leaks-in-nodejs-quick-overview-988c23b24dba

https://www.npmjs.com/package/autocannon

https://www.npmjs.com/package/memwatch-next

Thank **You**