

Web Workers Workshop

NG-BE

Sam Verschueren, Kwinten Pisman & Dominic Elm

Sam Verschueren

 @SamVerschueren

 SamVerschueren

 StackBlitz



Kwinten Pisman

 @KwintenP

 KwintenP

 StackBlitz



Dominic Elm

 @elmd_

 d3lm

 StackBlitz





StackBlitz

Introduction

Process vs. Thread

Process

Process

- Execution of a program

Process

- Execution of a program
- Can create child processes

Process

- Execution of a program
- Can create child processes
- Isolated
 - Has it's own stack, memory and data

Process

- Execution of a program
- Can create child processes
- Isolated
 - Has its own stack, memory and data
- Communication through IPC

Thread

Thread

- Execution unit, part of a process

Thread

- Execution unit, part of a process
- Process can have multiple threads (executing at the same time)

Thread

- Execution unit, part of a process
- Process can have multiple threads (executing at the same time)
- Information is shared
 - Global variables
 - File descriptors
 - Heap memory

Thread

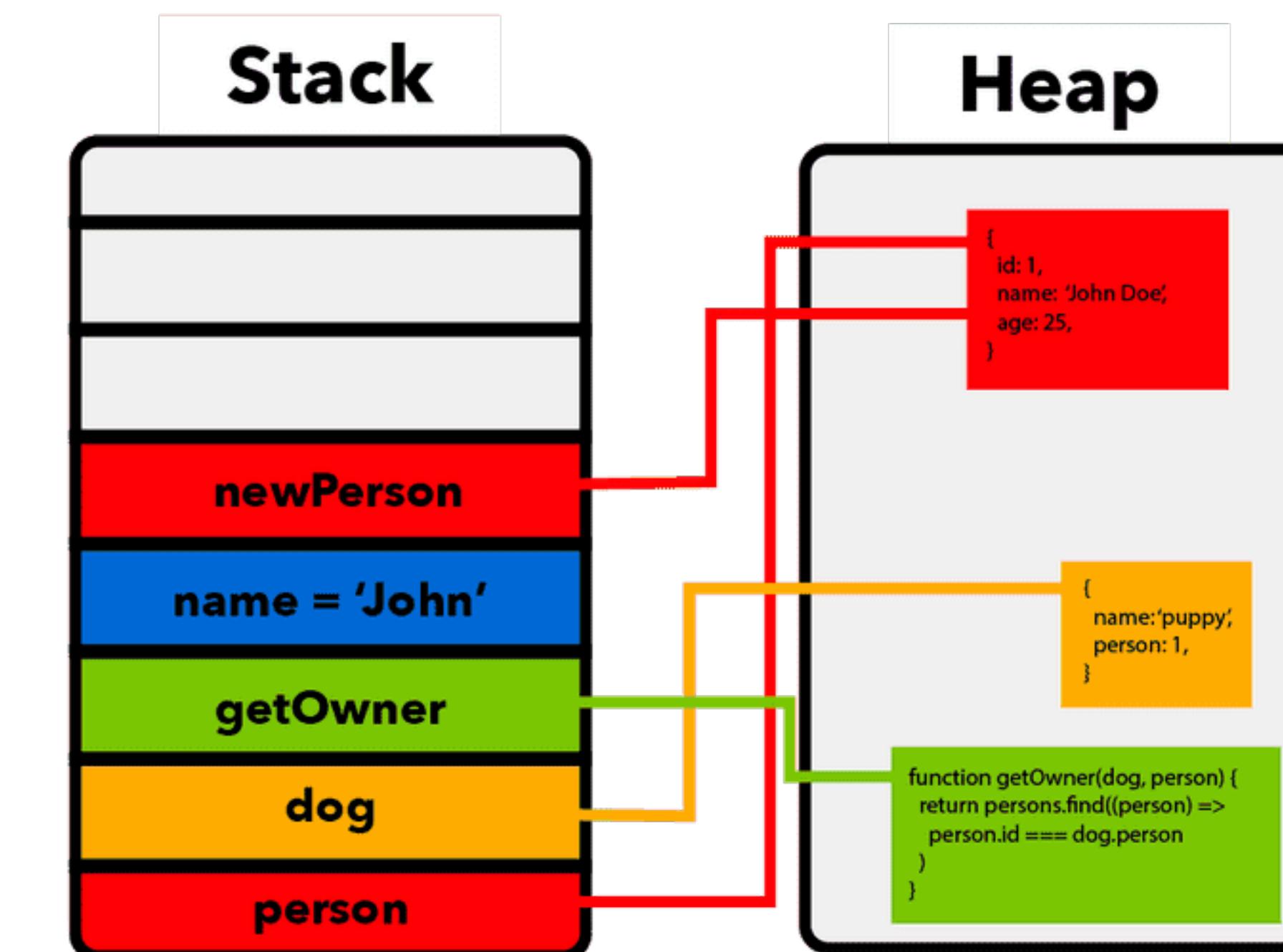
- Execution unit, part of a process
- Process can have multiple threads (executing at the same time)
- Information is shared
 - Global variables
 - File descriptors
 - Heap memory
- Communication between threads is faster

Process vs. Thread

	Process	Thread
Lightweight	No	Yes
Creation time	Slower	Faster
Termination Time	Slower	Faster
Intercommunication time	Slower	Faster
Context switching	Slower	Faster
Memory	Isolated	Shared
Data sharing	IPC	Memory

Stack and Heap

```
const person = {  
    id: 1,  
    name: 'John',  
    age: 25,  
}  
  
const dog = {  
    name: 'puppy',  
    personId: 1,  
}  
  
function getOwner(dog, persons) {  
    return persons.find((person) =>  
        person.id === dog.person  
    )  
}  
  
const name = 'John';  
  
const newPerson = person;
```

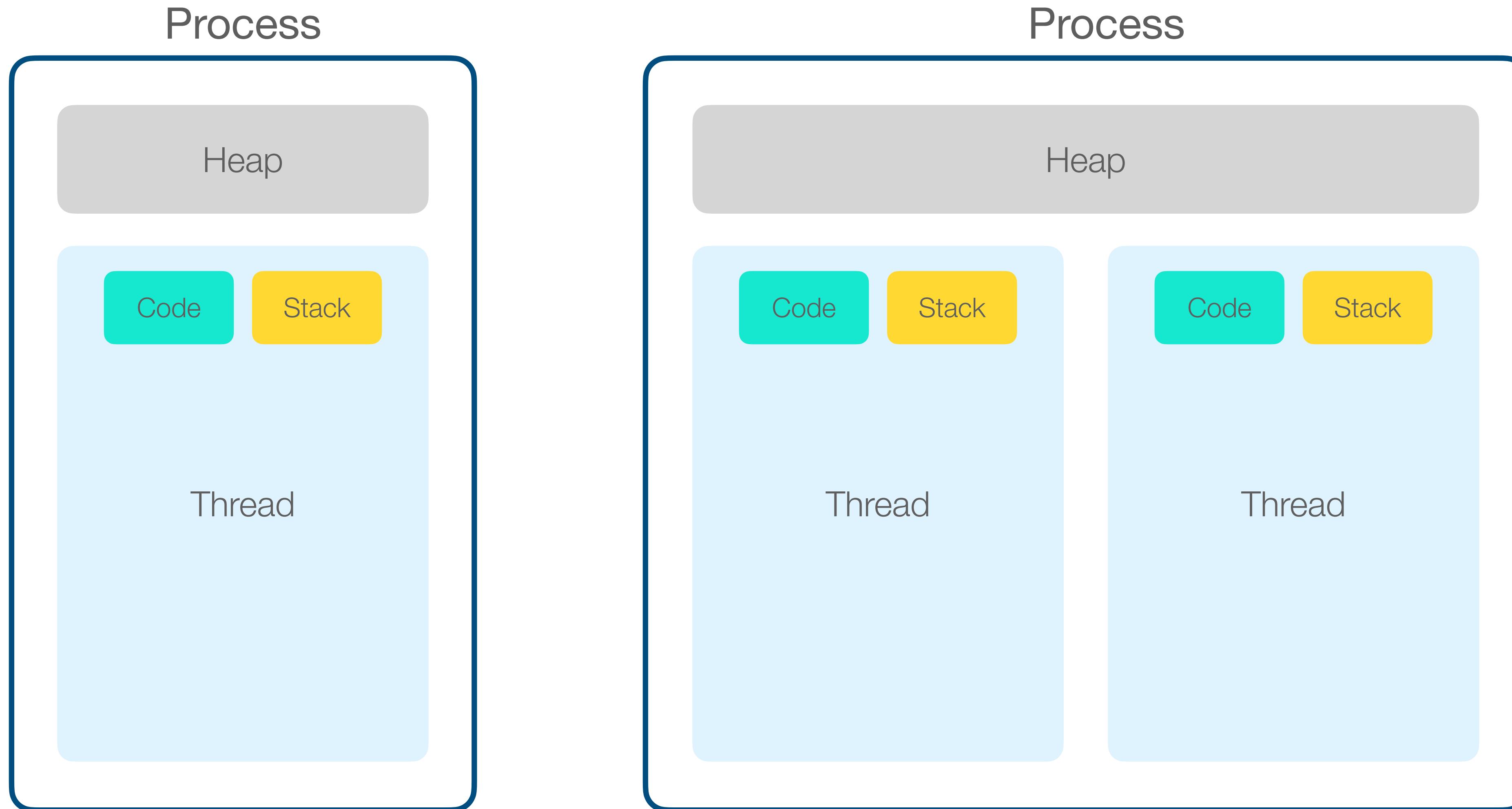


Stack and Heap

Stack	Heap
Primitive values and references	Objects and functions
Size known at compile time	Size known at run time
Fixed amount of memory	No limit per object



Process vs. Thread



Web Workers

What is a Web Worker?

What is a Web Worker?

- Runs a script in an **isolated background thread**
 - Does not interfere with the UI/main thread

What is a Web Worker?

- Runs a script in an **isolated background thread**
 - Does not interfere with the UI/main thread
- Don't share global execution context

What is a Web Worker?

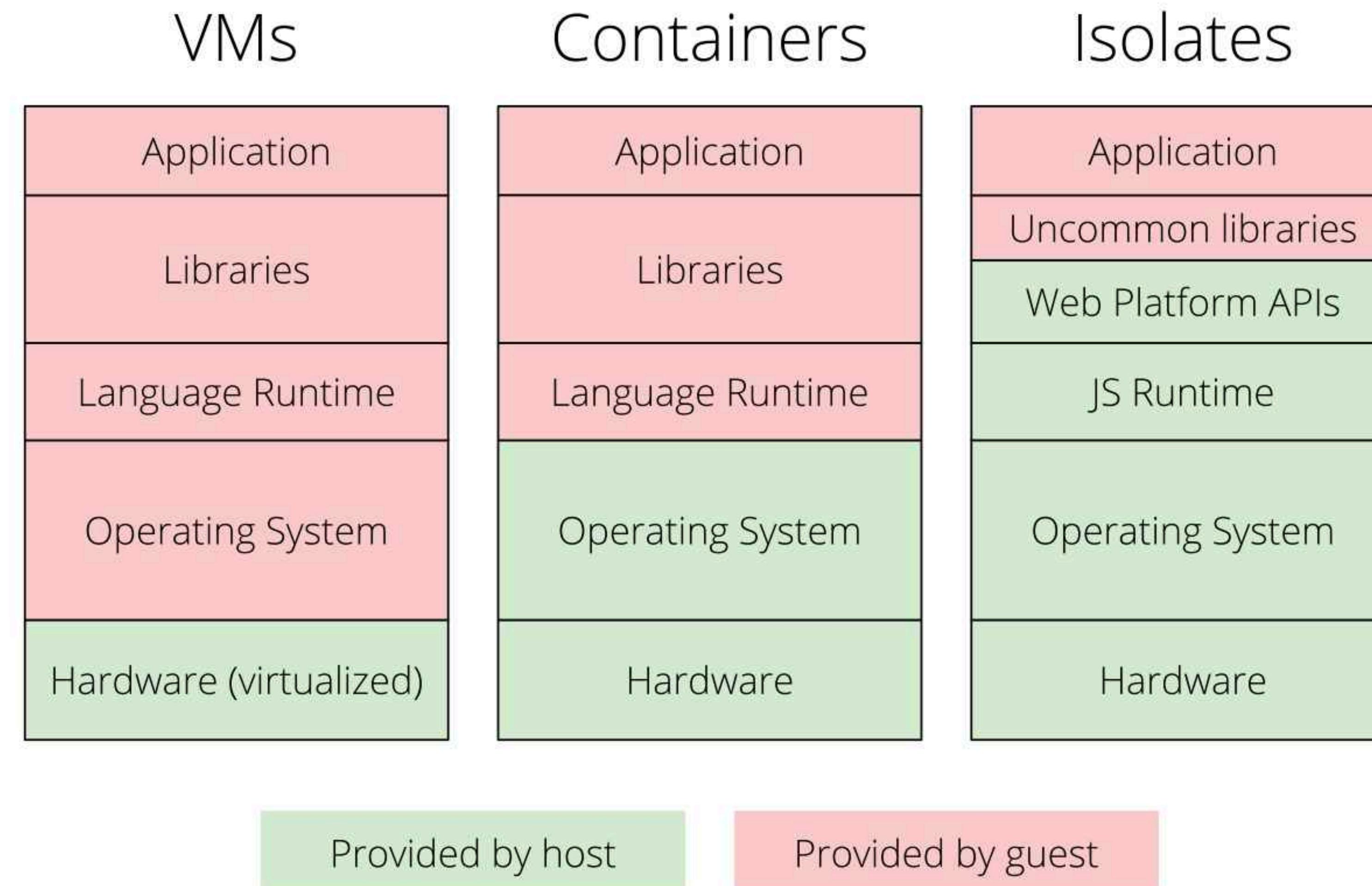
- Runs a script in an **isolated background thread**
 - Does not interfere with the UI/main thread
- Don't share global execution context
- Communication via sending/receiving messages or shared memory

What is a Web Worker?

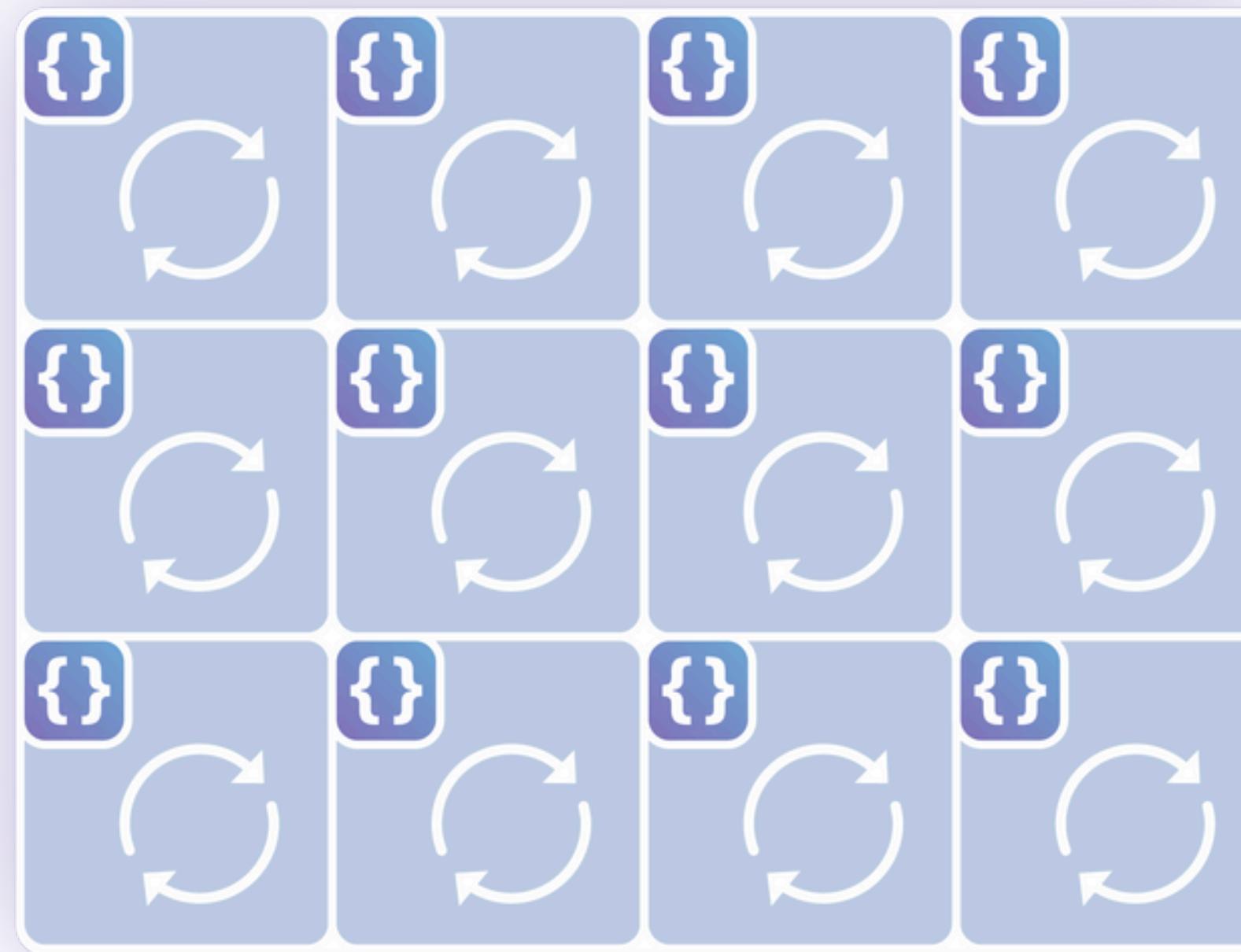
- Runs a script in an **isolated background thread**
 - Does not interfere with the UI/main thread
- Don't share global execution context
- Communication via sending/receiving messages or shared memory
- Implemented via V8 Isolates

Isolates

Isolates



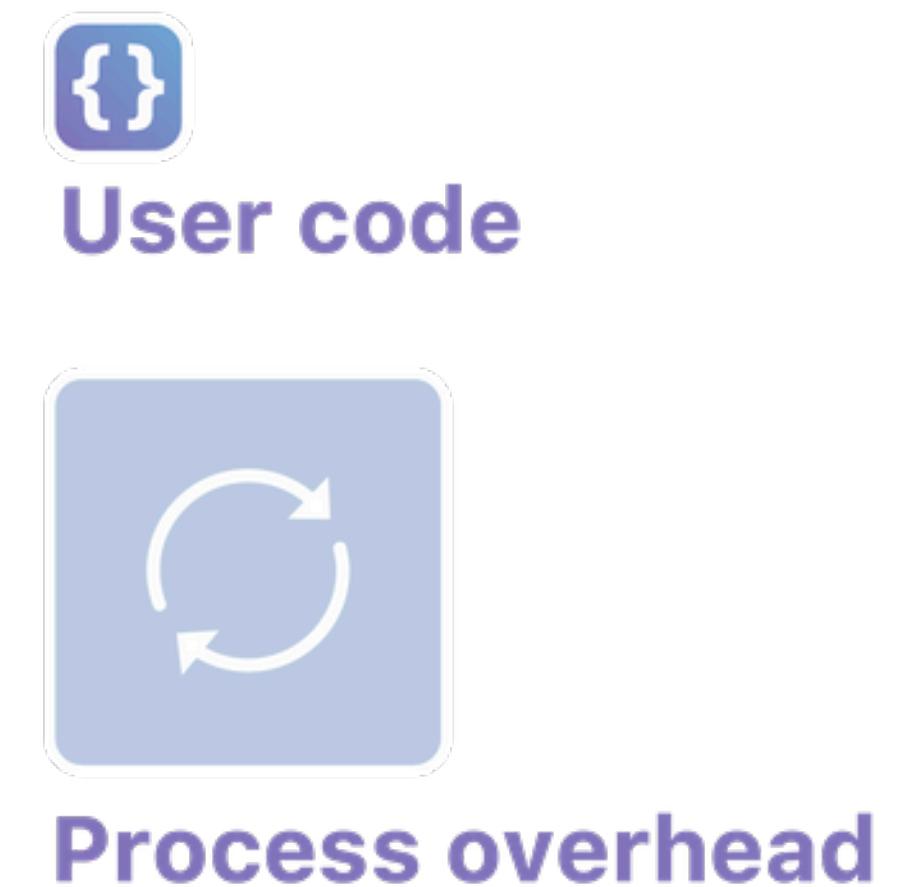
V8 isolates



Virtual machine



Isolate model



V8 isolates

V8 isolates

- Independent copy of the V8 runtime
 - Including heap manager, garbage collector, JIT compiler, ...

V8 isolates

- Independent copy of the V8 runtime
 - Including heap manager, garbage collector, JIT compiler, ...
- JS Runtime -> Garbage Collector, JIT compiler

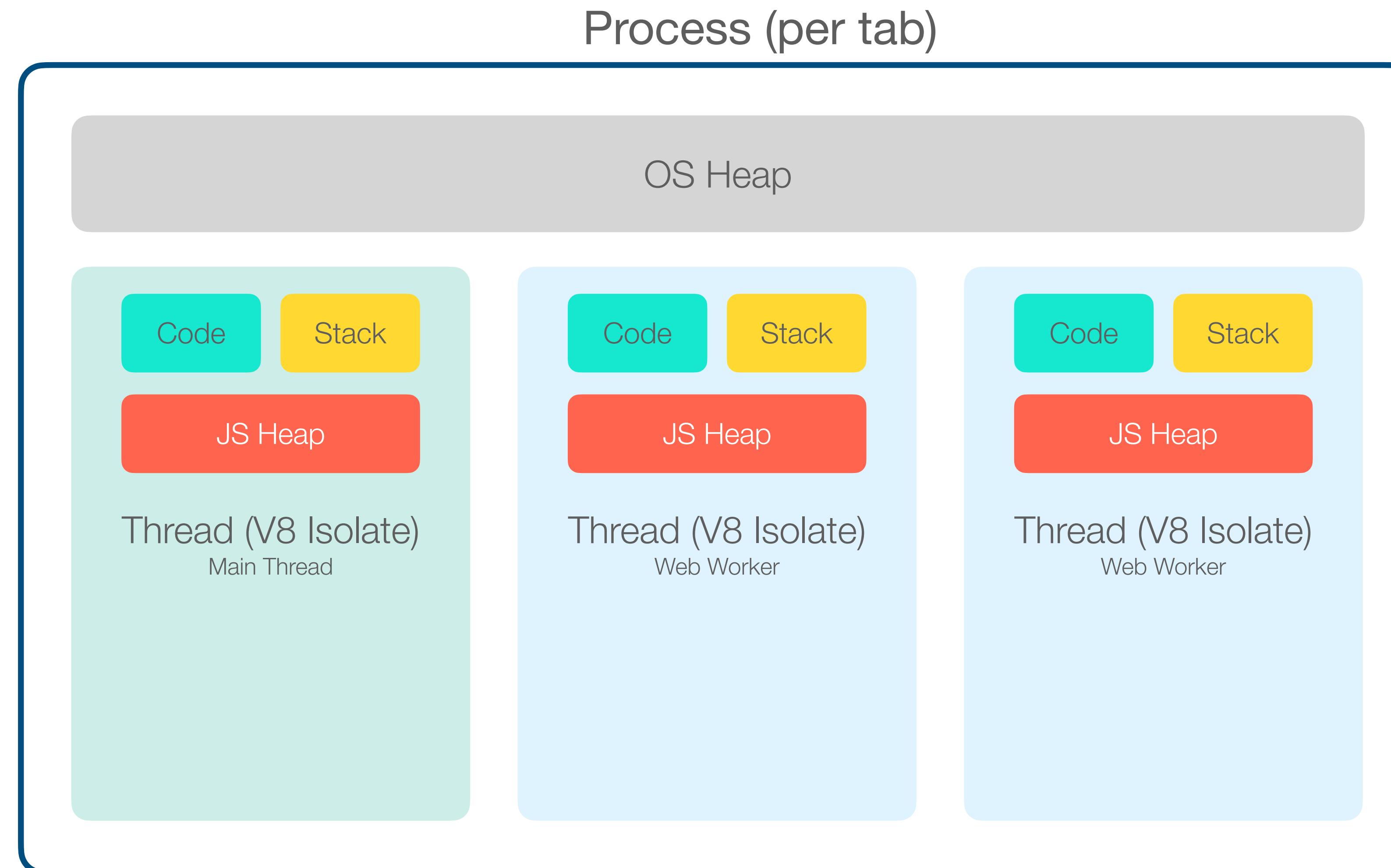
V8 isolates

- Independent copy of the V8 runtime
 - Including heap manager, garbage collector, JIT compiler, ...
- JS Runtime -> Garbage Collector, JIT compiler
- High level APIs (same HTTP implementation shared)
 - Containers (API is the sys call API which is very low-level)

V8 isolates

- Overhead of JS runtime is paid once (due to snapshots)
- Each isolate's memory is completely isolated

Process vs. Thread



Web Workers

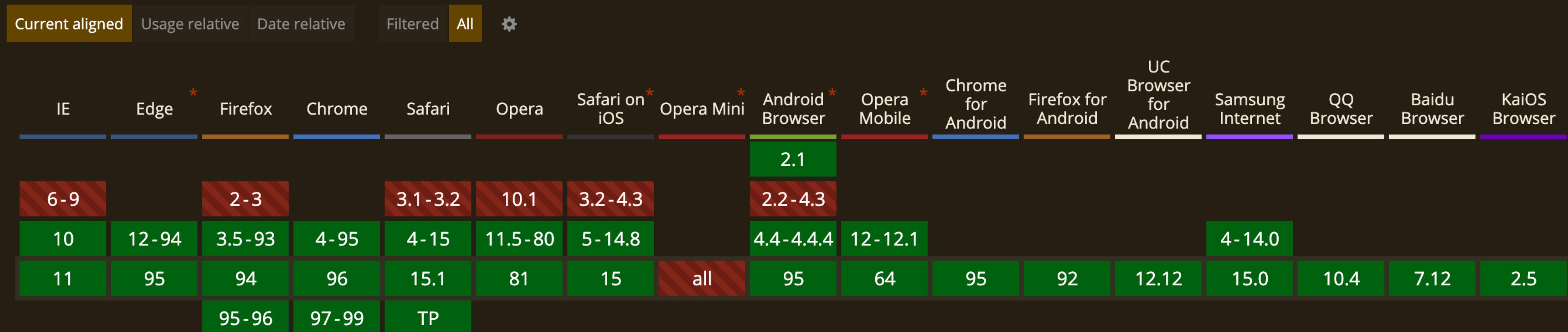
Support

Web Workers - LS

Global

98.52%

Method of running scripts in the background, isolated from the web page



Creating a Web Worker

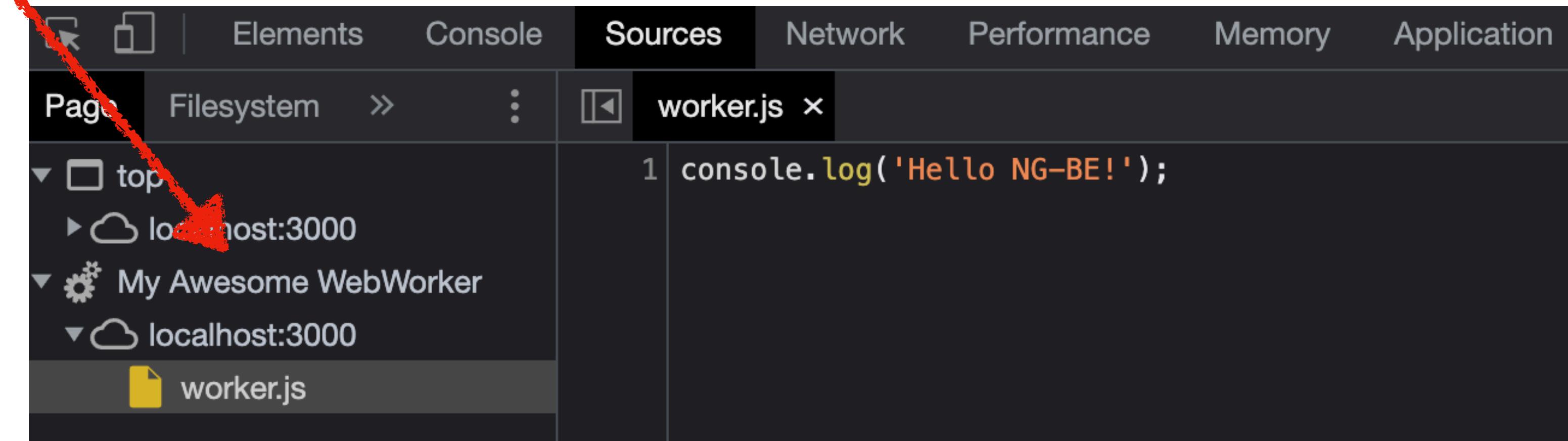
```
new Worker(scriptURL: string | URL, options?: WorkerOptions): Worker;
```

Creating a Web Worker

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

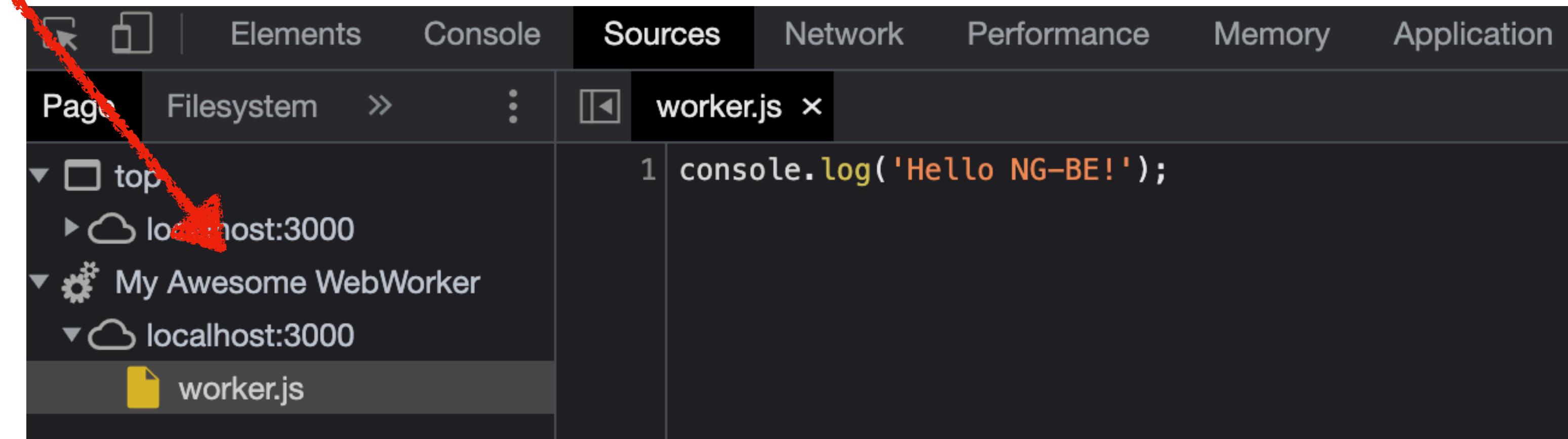
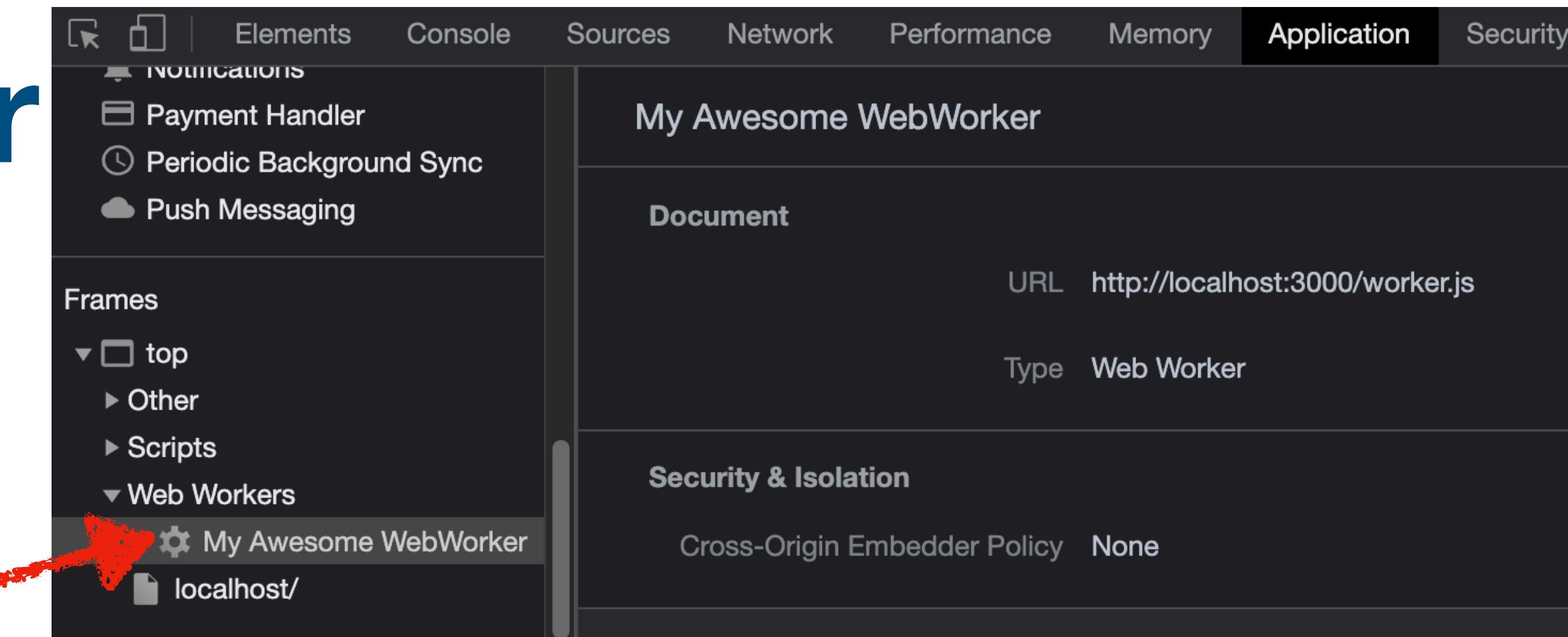
Creating a Web Worker

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```



Creating a Web Worker

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```



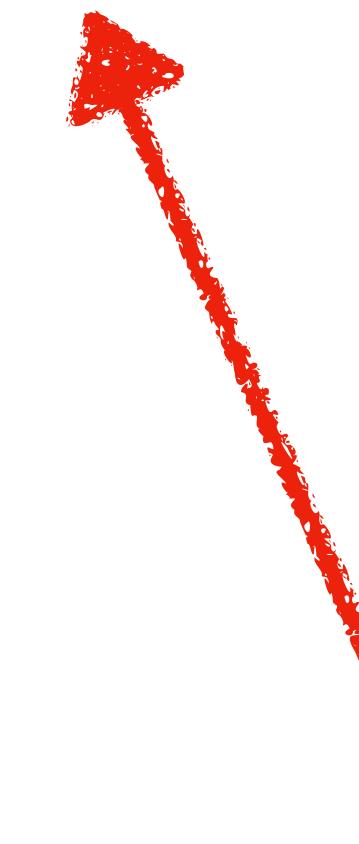
Communication

Sending data

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Sending data

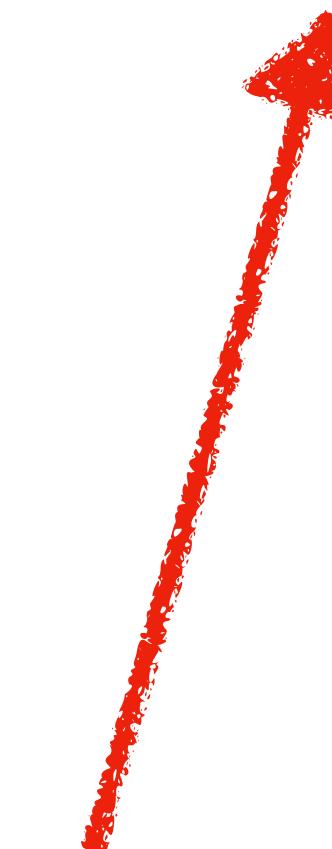
```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```



Any value or JavaScript object handled
by structured clone algorithm

Sending data

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```



We'll come back to this...

Structured Clone

Structured Clone

Structured Clone

- Deep clone of provided value

Structured Clone

- Deep clone of provided value
- Supports circular references

Structured Clone

- Deep clone of provided value
- Supports circular references
- Used by *postMessage* and *IndexedDB*

Structured Clone

- Deep clone of provided value
- Supports circular references
- Used by `postMessage` and `IndexedDB`

Object type	Notes
All primitive types	However, not symbols.
Boolean objects	
String objects	
Date	
RegExp	<code>lastIndex</code> is not preserved.
Blob	
File	
FileList	
ArrayBuffer	
ArrayBufferView	Including other typed arrays .
ImageBitmap	
ImageData	
Array	
Object	Only plain objects (e.g. from object literals)
Map	
Set	

Structured Clone Gotcha's

Structured Clone Gotcha's

- Cloning Function **throws** DataCloneError

Structured Clone Gotcha's

- Cloning Function **throws** DataCloneError
- Cloning DOM node **throws** DataCloneError

Structured Clone Gotcha's

- Cloning Function **throws** DataCloneError
- Cloning DOM node **throws** DataCloneError
- Cloning native Error types works in Chrome, Firefox is working on it

Structured Clone Gotcha's

- Cloning Function **throws** DataCloneError
- Cloning DOM node **throws** DataCloneError
- Cloning native Error types works in Chrome, Firefox is working on it
- Object properties
 - lastIndex of RegExp **not preserved**
 - Prototype chain **not walked or duplicated**
 - Property descriptors, setters, getters, and other metadata-like features **are not duplicated**. Readonly property becomes read/write in the duplicate.

Structured Clone

```
structuredClone(value: any, { transfer?: Transferable[] }): any;
```

Structured Clone

```
structuredClone(value: any, { transfer?: Transferable[] }): any;
```

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Structured Clone

```
structuredClone(value: any, { transfer?: Transferable[] }): any;  
  
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Structured Clone

```
structuredClone(value: any, { transfer?: Transferable[] }): any;
```

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;

const clone = structuredClone(original);
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;

const clone = structuredClone(original);

assert(clone !== original);
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;

const clone = structuredClone(original);

assert(clone !== original);
assert(clone.name === 'Sam Verschueren');
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;

const clone = structuredClone(original);

assert(clone !== original);
assert(clone.name === 'Sam Verschueren');
assert(clone.itself === clone);
```

Structured Clone

```
const original = { name: 'Sam Verschueren' };
original.itself = original;

const clone = structuredClone(original);

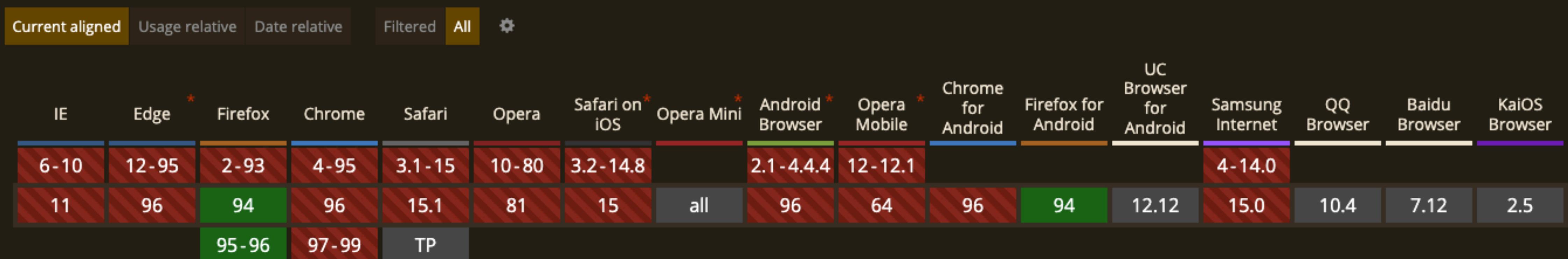
assert(clone !== original);
assert(clone.name === 'Sam Verschueren');
assert(clone.itself === clone);
```

Support

structuredClone API

Global

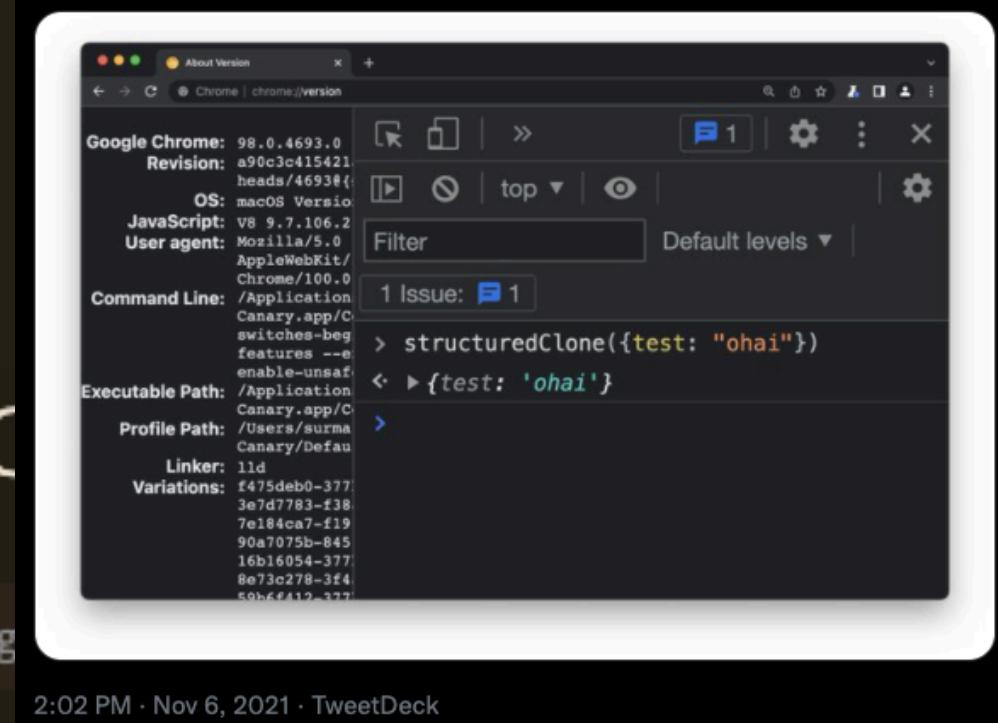
0.3%



Supp

Chrome Canary just got `structuredClone()`! That means ***all*** browsers (yes, even Safari) and Deno now support `structuredClone()` in their nightlies (some even shipped it to stable already!)

Massive shoutout to [@AndreuBotella](#) (not a Chromie!) for implementing structuredClone() <3



structuredC

Global

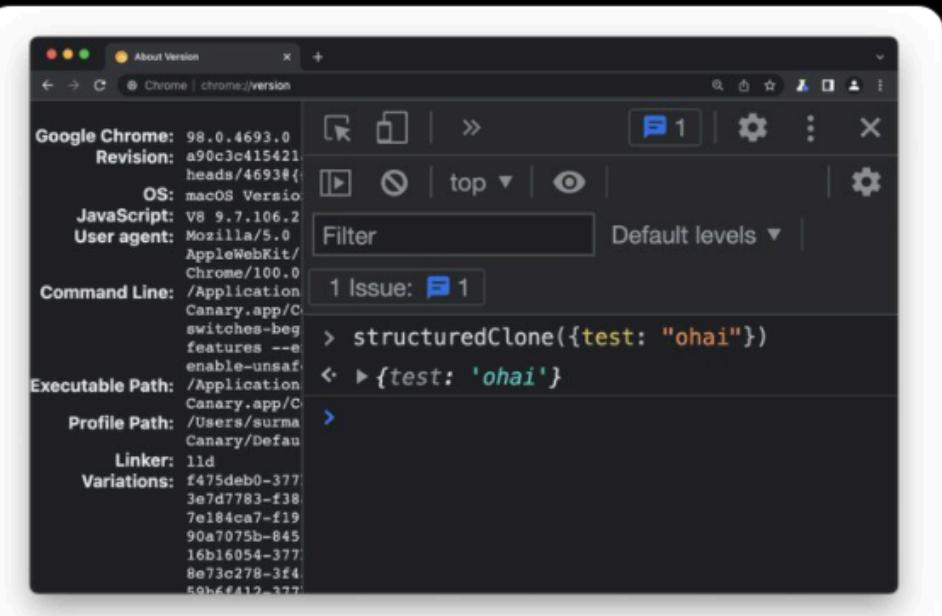
0.3%

Supp

 Surma
@DasSurma

Chrome Canary just got structuredClone()! That means *all* browsers (yes, even Safari) and Deno now support structuredClone() in their nightlies (some even shipped it to stable already!)

Massive shoutout to [@AndreuBotella](#) (not a Chromie!) for implementing structuredClone() <3



structuredC

Global

0.3%

Current aligned

Usage

2:02 PM · Nov 6, 2021 · TweetDeck

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS	Opera Mini *	Android Browser	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baidu Browser	KaiOS Browser
6-10	12-95	2-93	4-95	3.1-15	10-80	3.2-14.8		2.1-4.4.4	12-12.1			4-14.0				
11	96	94	96	15.1	81	15	all	96	64	96	94	12.12	15.0	10.1	7.12	2.5
		95-96	97-99	TP												



Oh I forgot, Node 17 shipped structuredClone() as well!!

```
surma@SurmBook ~/src/scratch/test/lol
$ node
Welcome to Node.js v17.0.1.
Type ".help" for more information.
> x = {lol: "ohai"}
{ lol: 'ohai' }
> y = structuredClone(x)
{ lol: 'ohai' }
> x = y
false
>
```

2:11 PM · Nov 6, 2021 · TweetDeck

**Back to
communication**

Sending data

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

Sending data

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});  
  
worker.postMessage({ name: 'Sam' });
```

Sending data

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});  
  
worker.postMessage({ name: 'Sam' });
```

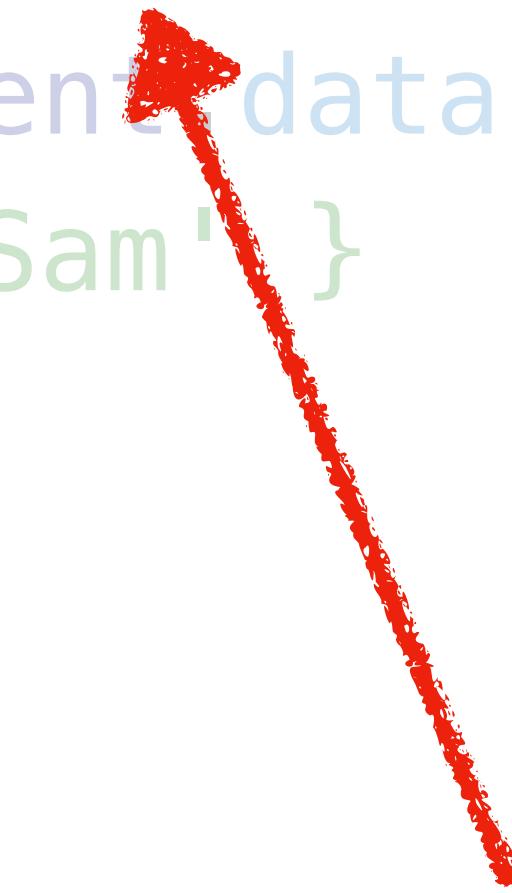
Receiving data

Receiving data

```
self.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

Receiving data

```
self.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

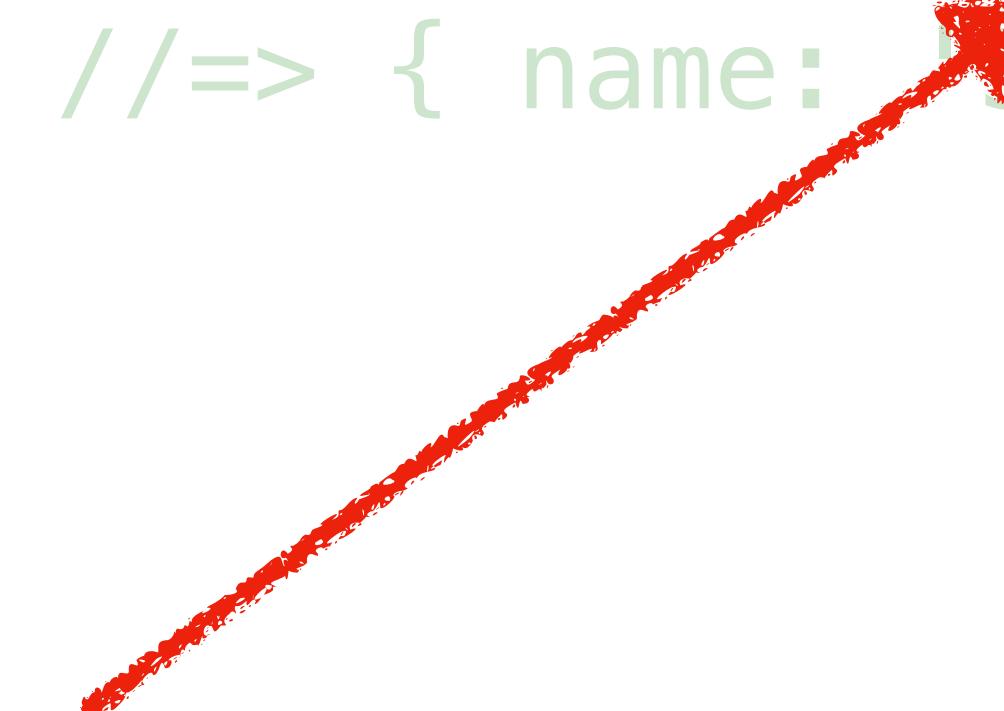


MessageEvent

Receiving data

```
self.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};

worker.postMessage({ name: 'Sam' });
```



Receiving data

```
self.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

Receiving data - Alternative

```
self.addEventListener('message', (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
});
```

Communication

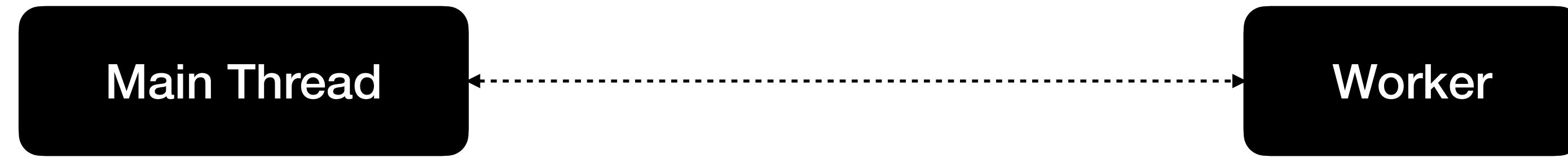
Main Thread

Communication

Main Thread

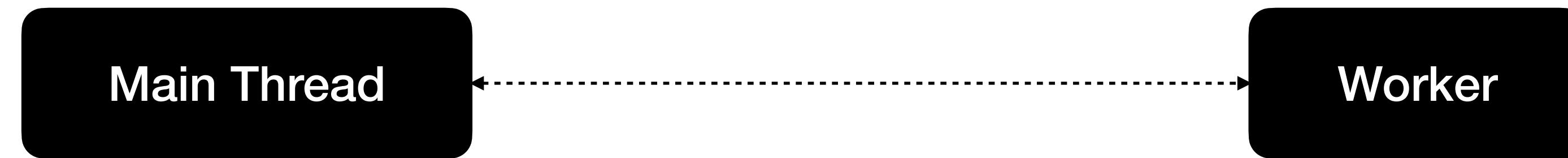
```
new Worker('./worker.js');
```

Communication



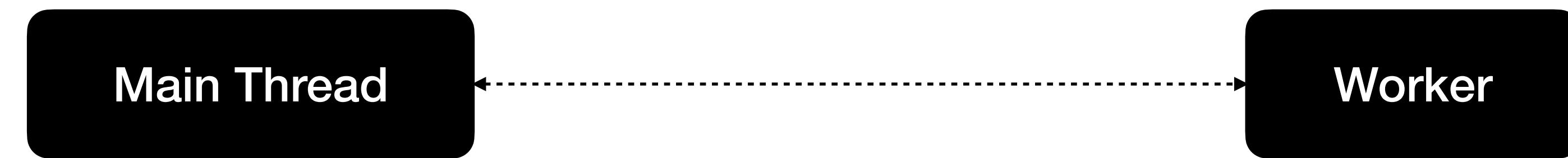
```
new Worker('./worker.js');
```

Communication



```
self.onmessage = (event) => {  
  console.log(event.data);  
};
```

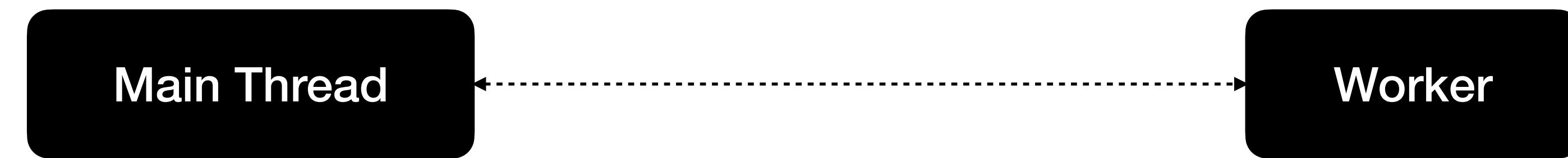
Communication



```
worker.postMessage({ name: 'Sam' });
```

```
self.onmessage = (event) => {  
  console.log(event.data);  
};
```

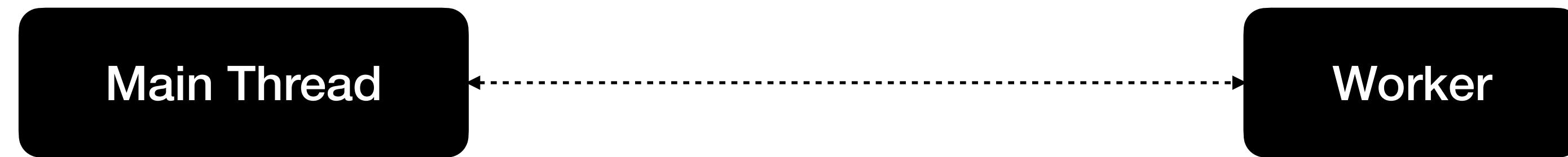
Communication



```
worker.postMessage({ name: 'Sam' });
```

```
self.onmessage = (event) => {  
  console.log(event.data);  
};
```

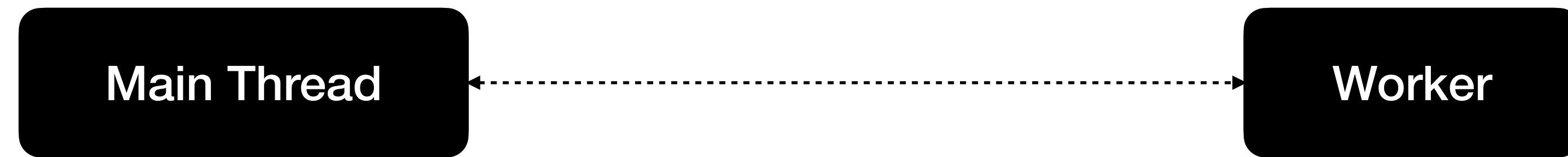
Communication



```
structuredClone({ name: 'Sam' });
```

```
worker.postMessage();  
self.onmessage = (event) => {  
  console.log(event.data);  
};
```

Communication



```
structuredClone({ name: 'Sam' });
```

```
worker.postMessage();
```

```
self.onmessage = (event) => {  
  console.log(event.data);  
};
```

Communication

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

Communication

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});  
  
worker.postMessage({ name: 'Sam' });
```

Communication

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

```
worker.postMessage({ name: 'Sam' });
```

```
self.onmessage = ({data}) => {  
  self.postMessage(`Hello, ${data.name}!`);  
};
```

Communication

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

```
worker.onmessage = ({data}) => {  
  console.log(data);  
  //=> Hello, Sam!  
};
```

```
worker.postMessage({ name: 'Sam' });
```

```
self.onmessage = ({data}) => {  
  self.postMessage(`Hello, ${data.name}!`);  
};
```

Communication

```
const worker = new Worker('./worker.js', {  
  name: 'My Awesome WebWorker'  
});
```

```
worker.onmessage = ({data}) => {  
  console.log(data);  
  //=> Hello, Sam!  
};
```

```
worker.postMessage({ name: 'Sam' });
```

```
self.onmessage = ({data}) => {  
  self.postMessage(`Hello, ${data.name}!`);  
};
```

Limitations

Limitations

- Can't access `window`
 - And thus no DOM access

Limitations

- Can't access `window`
 - And thus no DOM access
- Can't access `LocalStorage`
 - You **can** access `IndexedDB`

Limitations

- Can't access `window`
 - And thus no DOM access
- Can't access `LocalStorage`
 - You **can** access `IndexedDB`
- Can't use `Canvas`
 - `OffscreenCanvas` behind feature flag

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 1-fibonacci > Exercise 1

Terminator

Terminating a Web Worker

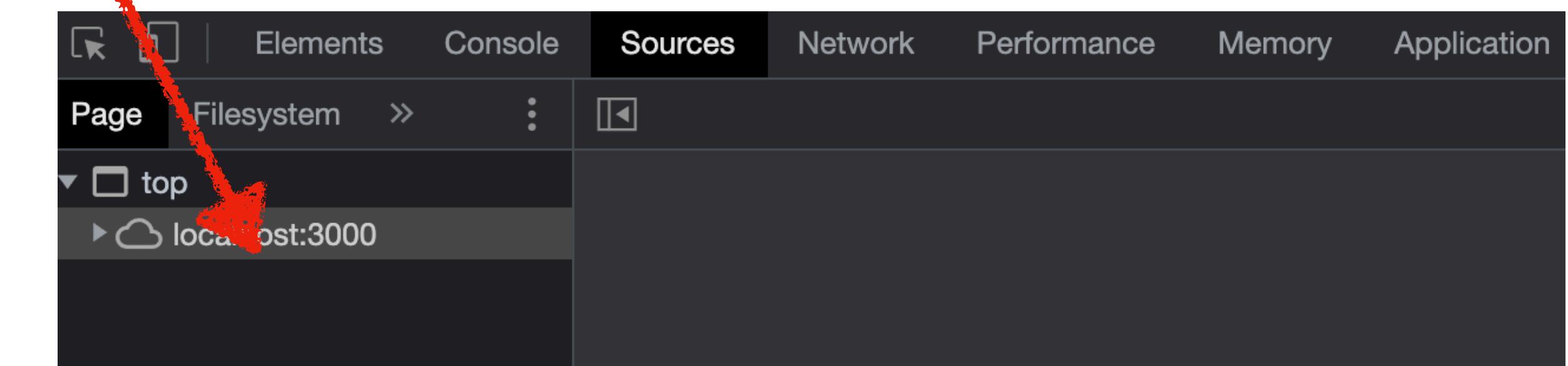


Terminating a Web Worker

```
worker.terminate(): void;
```

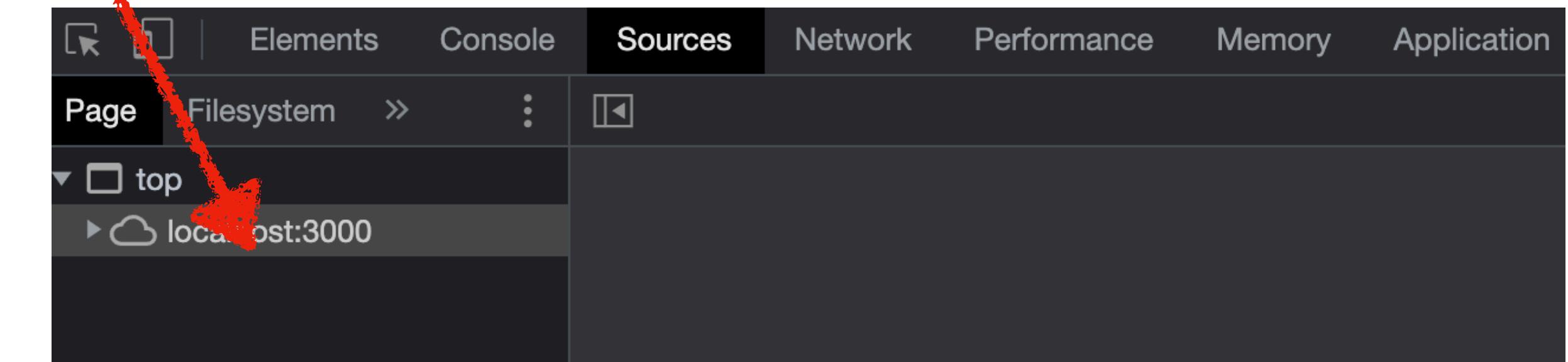
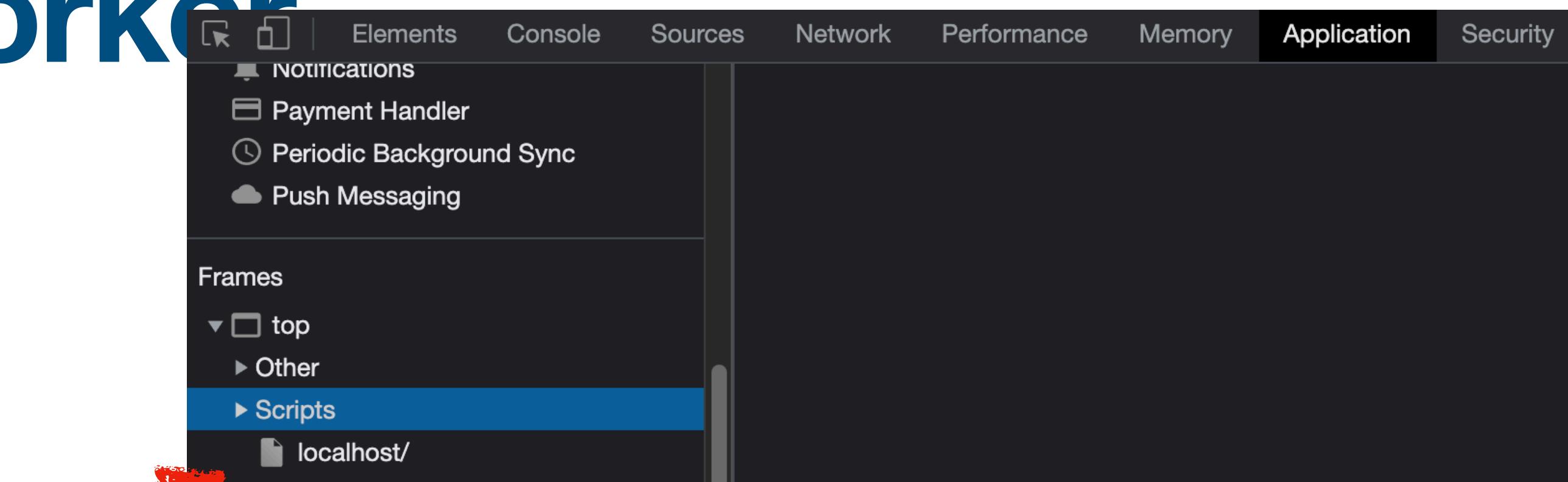
Terminating a Web Worker

```
worker.terminate(): void;
```



Terminating a Web Worker

```
worker.terminate(): void;
```



Stopping a Web Worker

- Immediately terminates the Worker
- Worker is not able to finish operations
- Same worker cannot be re-used

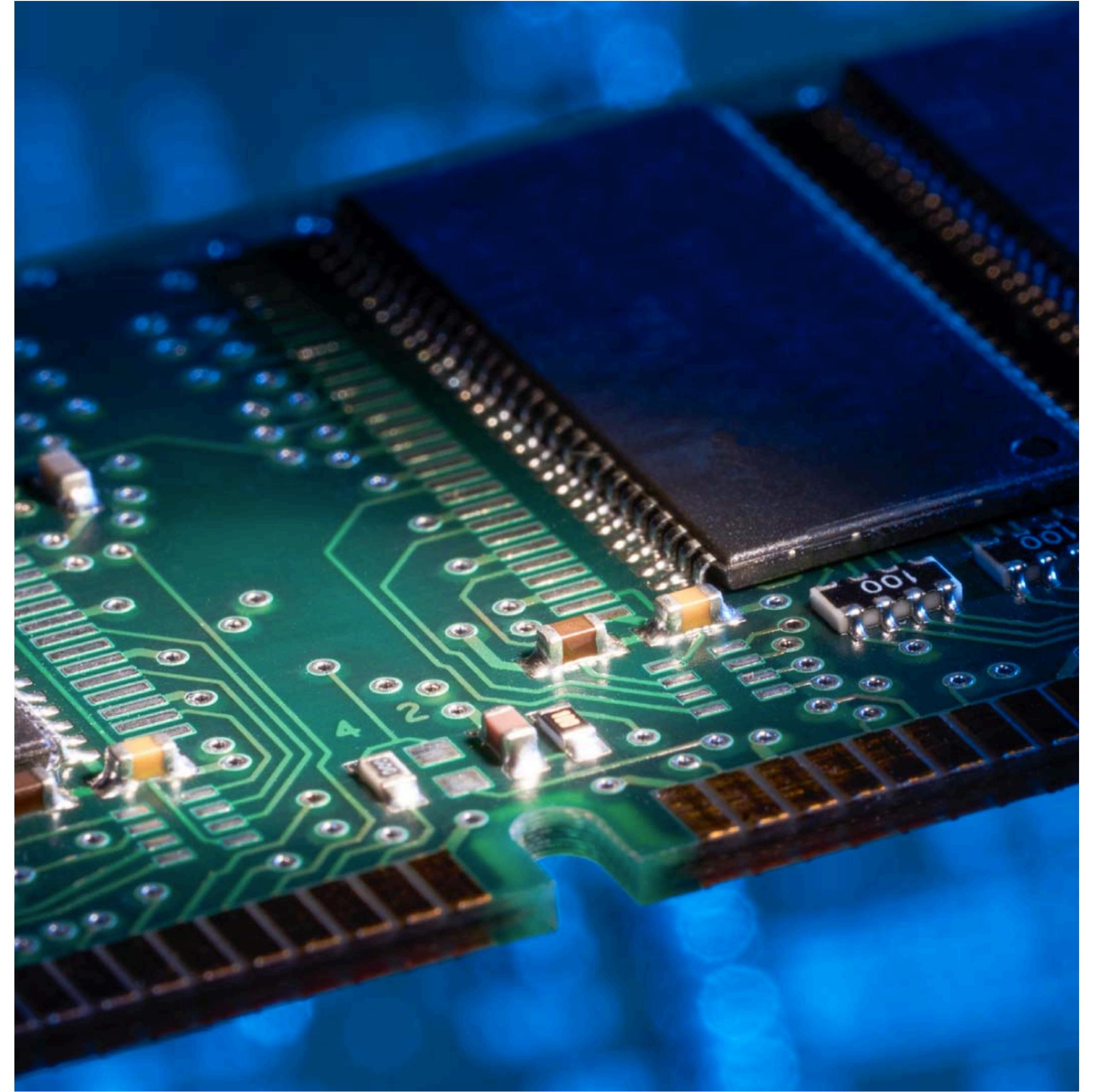
Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 1-fibonacci > Exercise 2

Implement with multiple workers so we can cancel previous calculations

Shared Memory



Shared Memory

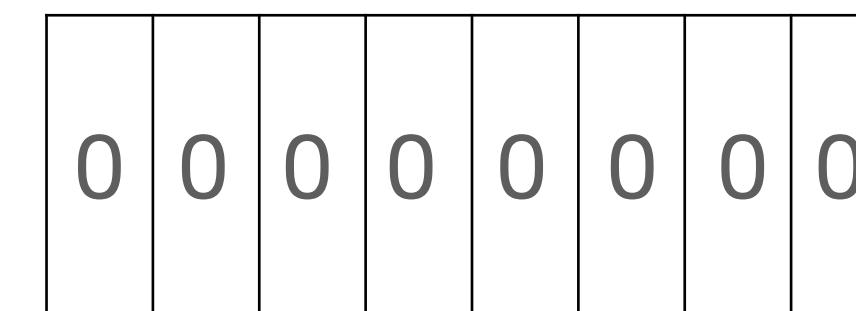
Main Thread

Worker 1

Worker 2

Shared Memory

Main Thread

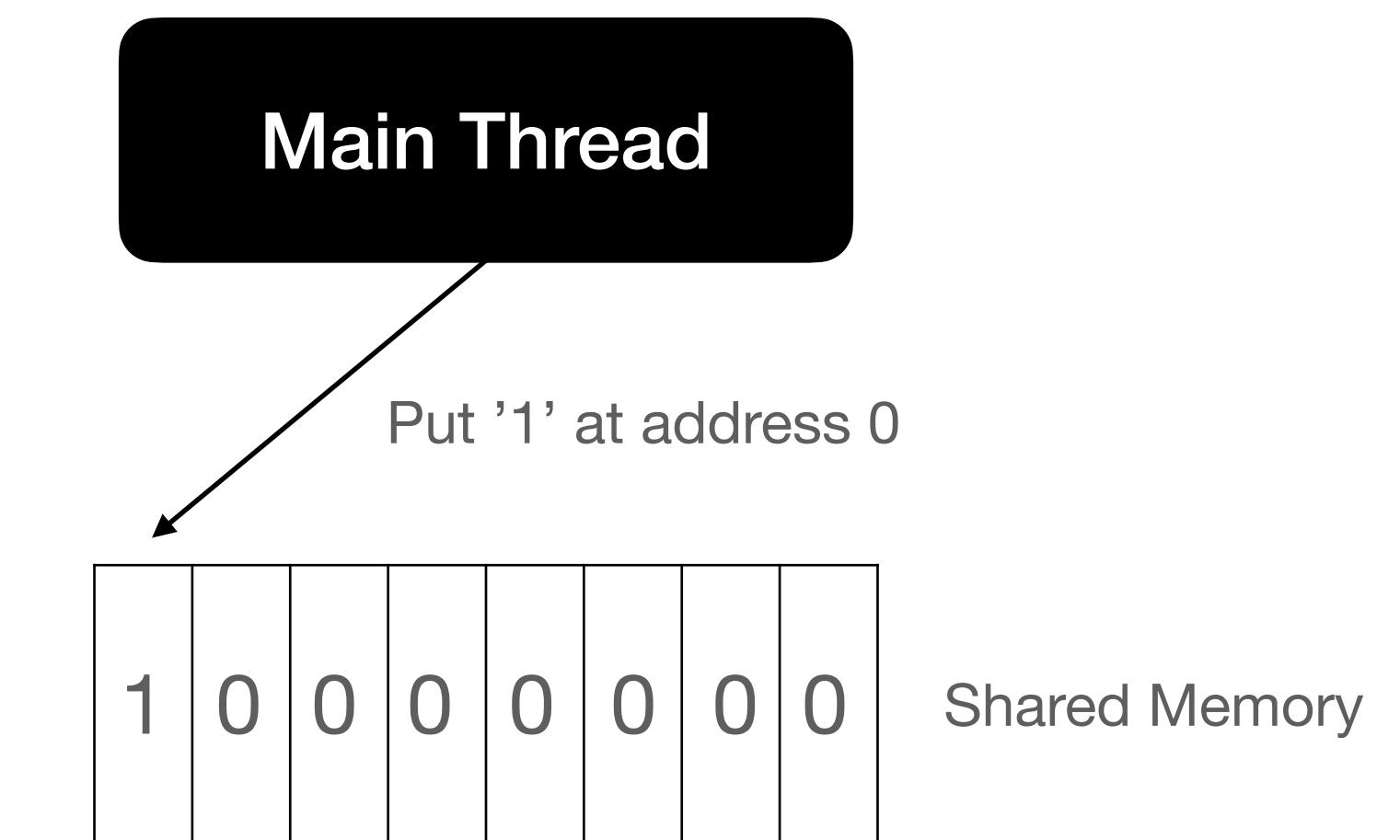


Shared Memory

Worker 1

Worker 2

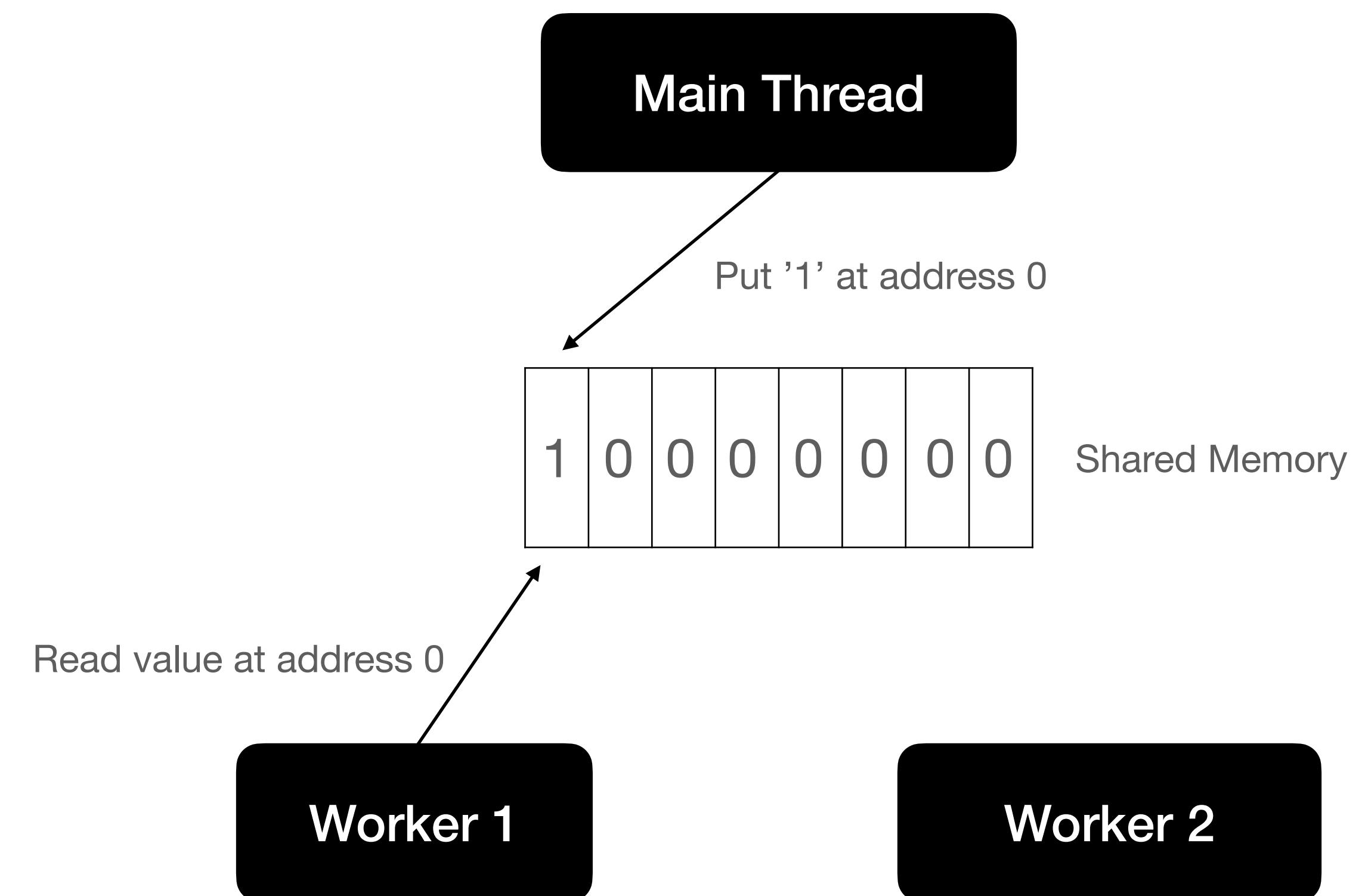
Shared Memory



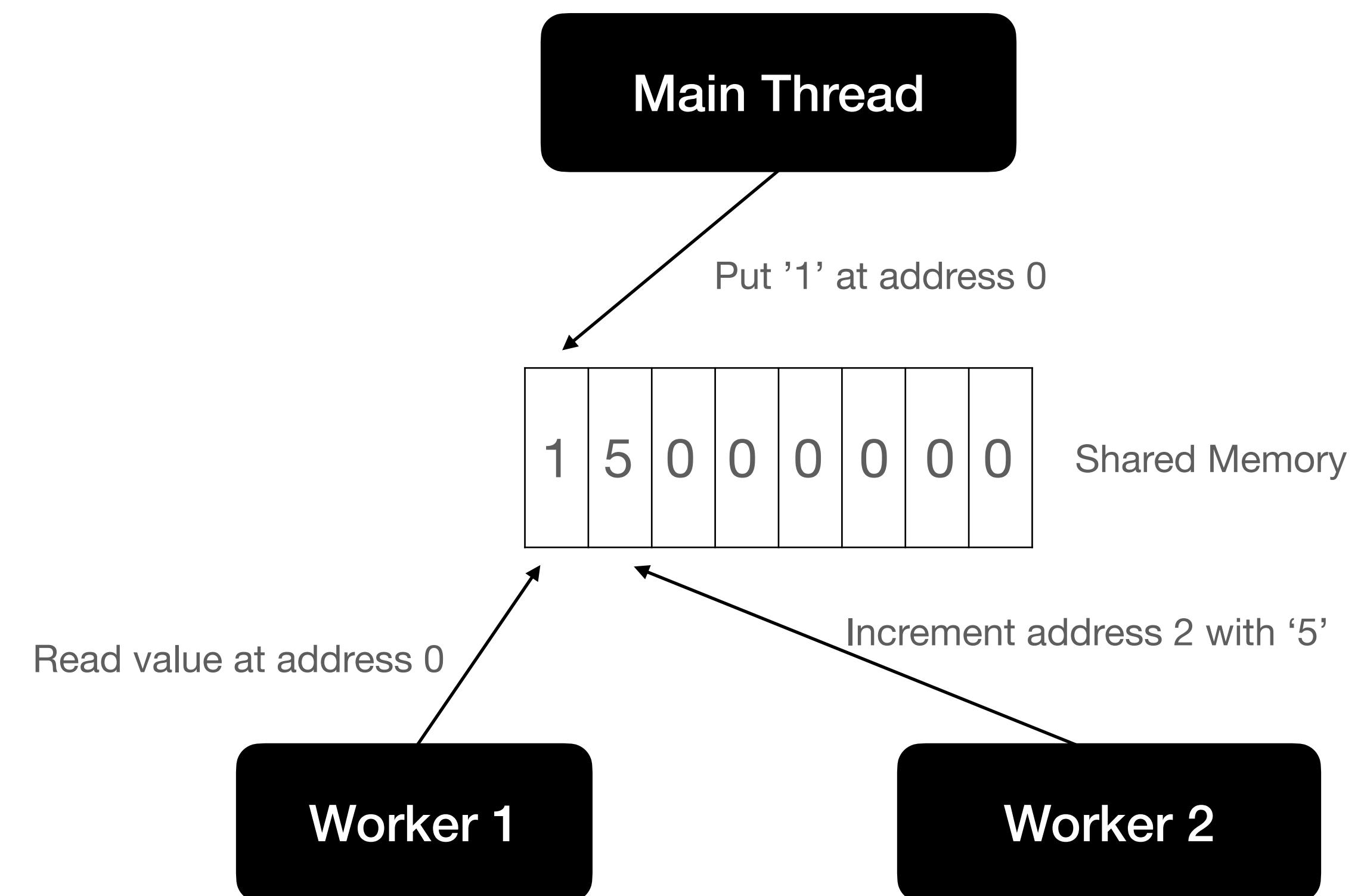
Worker 1

Worker 2

Shared Memory



Shared Memory



Hello SharedArrayBuffer



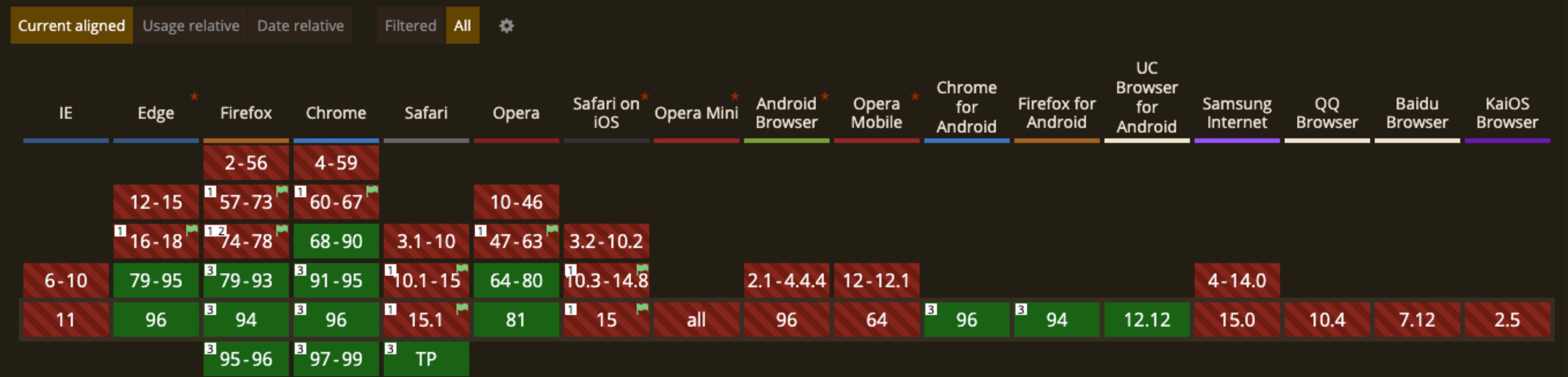
Support

Shared Array Buffer  - OTHER

Global

73.89%

Type of ArrayBuffer that can be shared across Workers.



Allocating memory

```
new SharedArrayBuffer(byteLength: number): SharedArrayBuffer;
```

SharedArrayBuffer

SharedArrayBuffer

- Fixed-length raw binary data buffer

SharedArrayBuffer

- Fixed-length raw binary data buffer
- Other data structures needed to update data
 - TypedArray or DataView

Typed Arrays

Type	Value Range	Size in bytes	Description	Web IDL type	Equivalent C type
Int8Array	-128 to 127	1	8-bit two's complement signed integer	byte	int8_t
Uint8Array	0 to 255	1	8-bit unsigned integer	octet	uint8_t
Uint8ClampedArray	0 to 255	1	8-bit unsigned integer (clamped)	octet	uint8_t
Int16Array	-32768 to 32767	2	16-bit two's complement signed integer	short	int16_t
Uint16Array	0 to 65535	2	16-bit unsigned integer	unsigned short	uint16_t
Int32Array	-2147483648 to 2147483647	4	32-bit two's complement signed integer	long	int32_t
Uint32Array	0 to 4294967295	4	32-bit unsigned integer	unsigned long	uint32_t
Float32Array	-3.4E38 to 3.4E38 and 1.2E-38 is the min positive number	4	32-bit IEEE floating point number (7 significant digits e.g., 1.234567)	unrestricted float	float
Float64Array	-1.8E308 to 1.8E308 and 5E-324 is the min positive number	8	64-bit IEEE floating point number (16 significant digits e.g., 1.23456789012345)	unrestricted double	double
BigInt64Array	-2^63 to 2^63 - 1	8	64-bit two's complement signed integer	bigint	int64_t (signed long long)
BigUint64Array	0 to 2^64 - 1	8	64-bit unsigned integer	bigint	uint64_t (unsigned long long)

Typed Arrays

ArrayBuffer (16 bytes)

Shared Memory

```
const sab = new SharedArrayBuffer(1024);
```

Shared Memory

```
const sab = new SharedArrayBuffer(1024);  
  
const typedArray = new Uint32Array(sab);
```

Shared Memory

```
const sab = new SharedArrayBuffer(1024);
```

```
const typedArray = new Uint32Array(sab);
```



Uint32Array	0 to 4294967295	4	32-bit unsigned integer	unsigned long	uint32_t
-----------------------------	--------------------	---	-------------------------	---------------	----------

Shared Memory

```
const sab = new SharedArrayBuffer(1024);
```

```
const typedArray = new Uint32Array(sab);
```

The diagram illustrates the creation of a SharedArrayBuffer and its conversion into a Uint32Array. It consists of two main parts: a code snippet at the top and a memory map below it.

The code snippet shows:

```
const sab = new SharedArrayBuffer(1024);  
const typedArray = new Uint32Array(sab);
```

The memory map below shows the structure of the SharedArrayBuffer:

Uint32Array	0 to 4294967295	4	32-bit unsigned integer	unsigned long	uint32_t
-----------------------------	--------------------	---	-------------------------	---------------	----------

Red arrows point from the code to specific parts of the memory map:

- A red arrow points from the variable `sab` in the first code line to the `SharedArrayBuffer` constructor in the memory map.
- A red arrow points from the variable `typedArray` in the second code line to the `Uint32Array` constructor in the memory map.
- A red arrow points from the value `1024` in the first code line to the `length` field in the memory map.
- A red arrow points from the value `4` in the second code line to the `byteLength` field in the memory map.

byteLength equals 1024

length equals 256 (4 bytes per element)

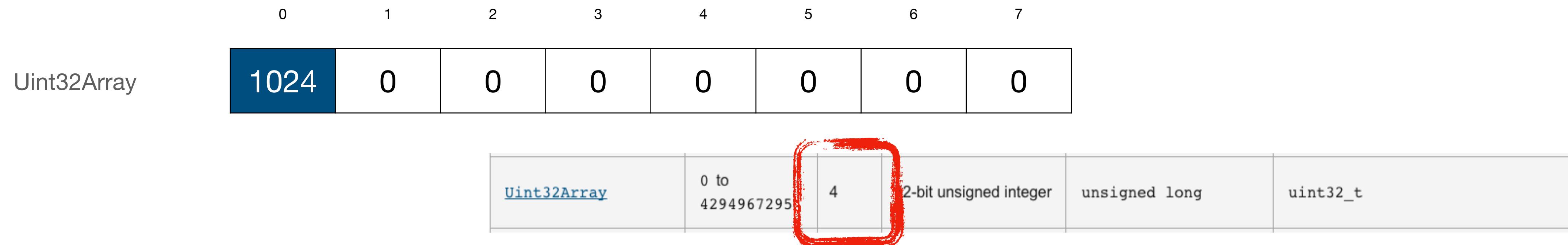
Shared Memory

```
const sab = new SharedArrayBuffer(1024);  
  
const typedArray = new Uint32Array(sab);  
  
typedArray[0] = 1024;
```

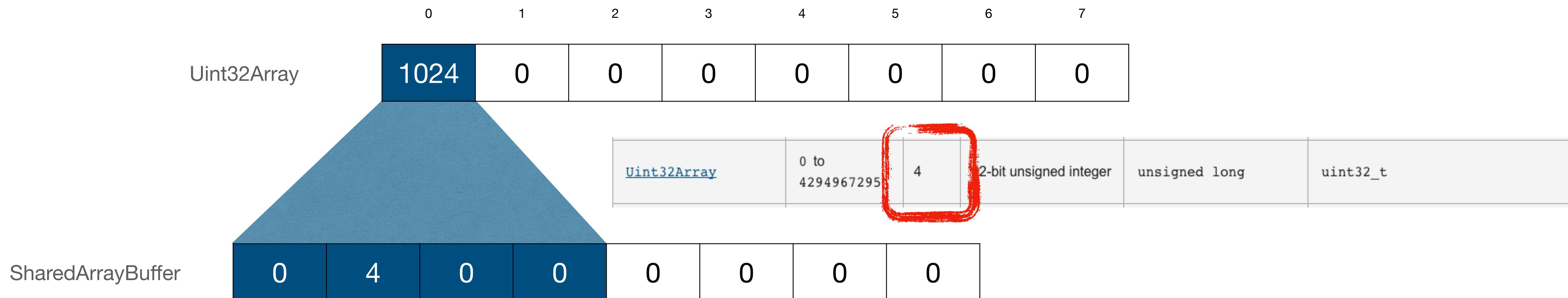
Shared Memory

	0	1	2	3	4	5	6	7
Uint32Array	1024	0	0	0	0	0	0	0

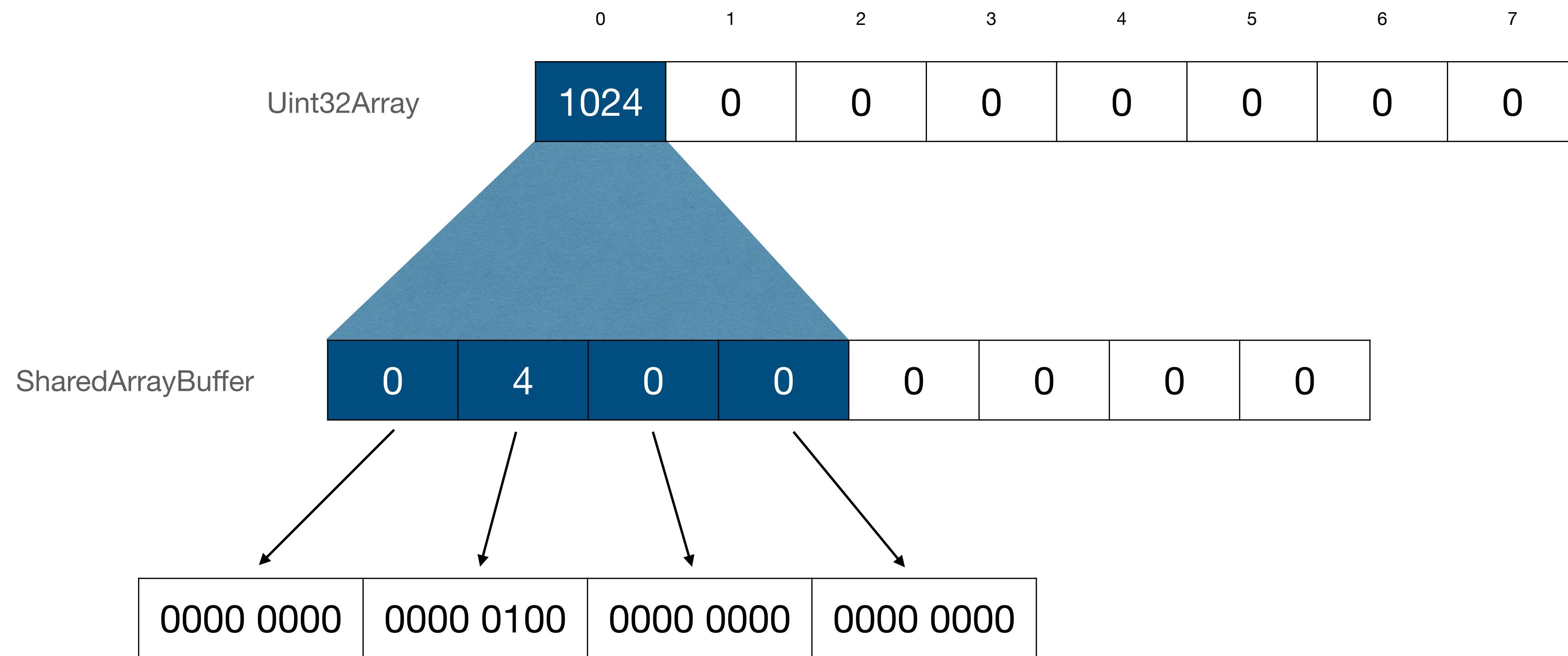
Shared Memory



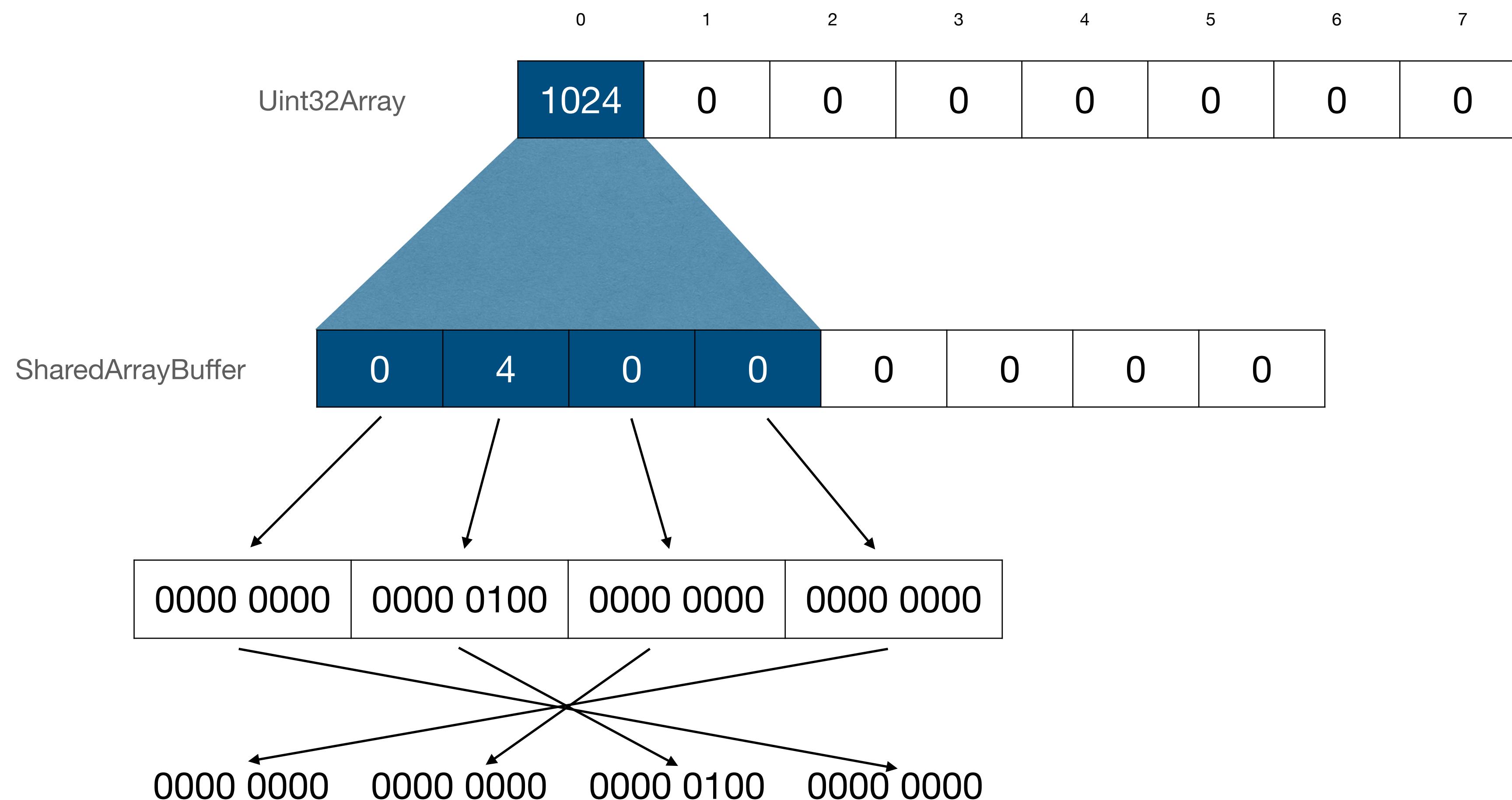
Shared Memory



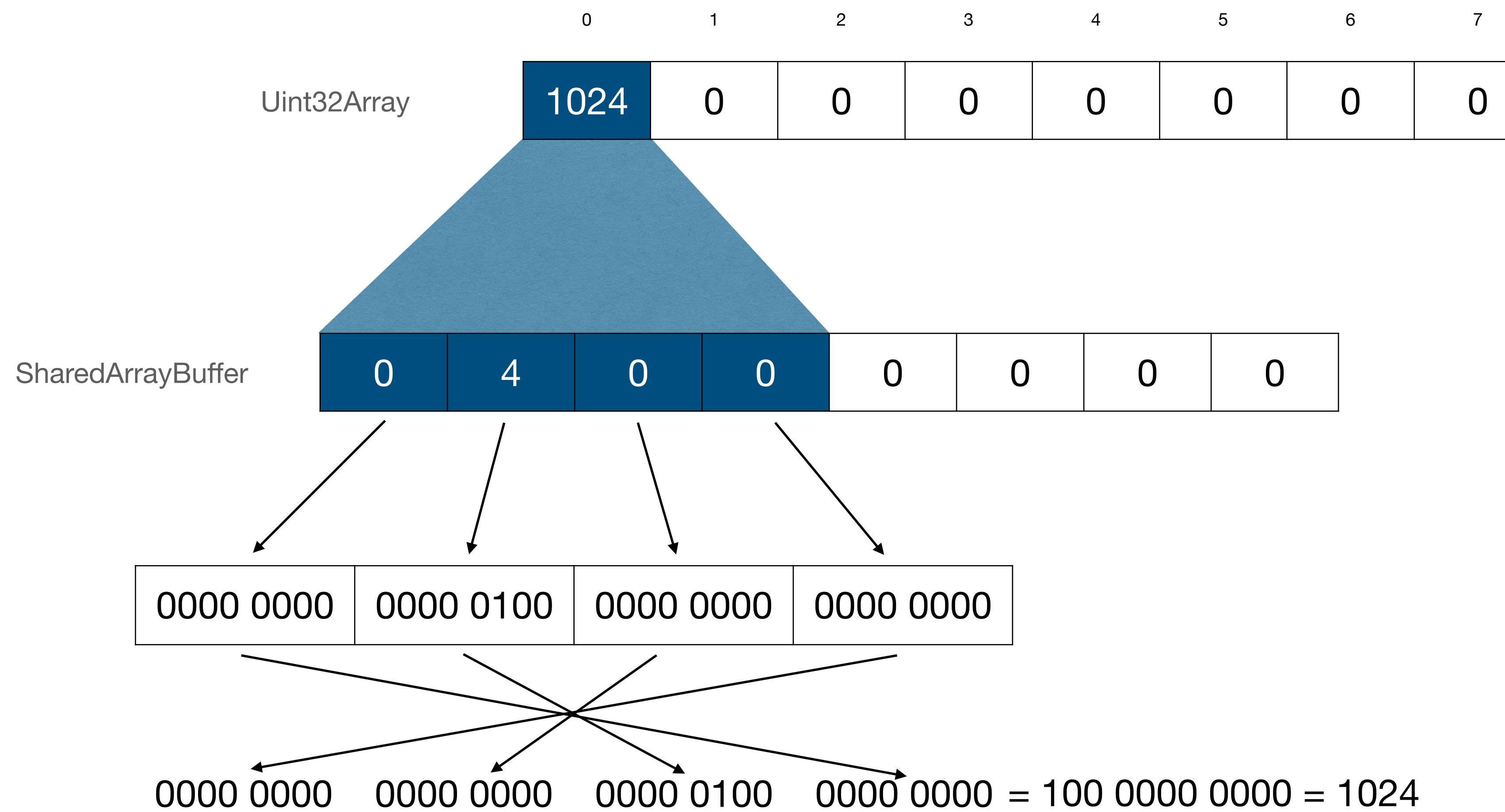
Shared Memory



Shared Memory



Shared Memory



Shared Memory

Shared Memory

```
const sab = new SharedArrayBuffer(1);
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);
```

```
self.onmessage = ({data}) => {
};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.postMessage({ memory: sab });
```

```
self.onmessage = ({data}) => {
};

};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.postMessage({ memory: sab });
```

```
self.onmessage = ({data}) => {
  const typedArray = new Uint8Array(data.memory);

};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.postMessage({ memory: sab });
```

```
self.onmessage = ({data}) => {
  const typedArray = new Uint8Array(data.memory);

  let count = 0;

  while (typedArray[0] !== 1) {
    count++;
  }

  self.postMessage(count);
};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.onmessage = ({data}) => {
  console.log(`Total count: ${data}`);
};

worker.postMessage({ memory: sab });
```

```
self.onmessage = ({data}) => {
  const typedArray = new Uint8Array(data.memory);

  let count = 0;

  while (typedArray[0] !== 1) {
    count++;
  }

  self.postMessage(count);
};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.onmessage = ({data}) => {
  console.log(`Total count: ${data}`);
};

worker.postMessage({ memory: sab });

setTimeout(() => {
  typedArray[0] = 1;
}, 100);
```

```
self.onmessage = ({data}) => {
  const typedArray = new Uint8Array(data.memory);

  let count = 0;

  while (typedArray[0] !== 1) {
    count++;
  }

  self.postMessage(count);
};
```

Shared Memory

```
const sab = new SharedArrayBuffer(1);
const typedArray = new Uint8Array(sab);

worker.onmessage = ({data}) => {
  console.log(`Total count: ${data}`);
};

worker.postMessage({ memory: sab });

setTimeout(() => {
  typedArray[0] = 1;
}, 100);
```

```
self.onmessage = ({data}) => {
  const typedArray = new Uint8Array(data.memory);

  let count = 0;

  while (typedArray[0] !== 1) {
    count++;
  }

  self.postMessage(count);
};
```



MELTDOWN



SPECTRE

Meltdown & Spectre

- Discovered in January 2018
- Both allow an attacker to access or find out secrets
- SharedArrayBuffer was disabled as it allows timing attacks
- More info on meltdownattack.com

Please Sir...



May I have my SharedArrayBuffer back?

Enter... Cross-Origin Isolation

Cross-Origin Isolation

- Requires 2 headers on the top-level document
 - COOP - Cross-Origin-Opener-Policy
 - COEP - Cross-Origin-Embedder-Policy
- Can be checked with `window.crossOriginIsolated`
- It's a PITA

Cross-Origin Isolation

Cross-Origin-Opener-Policy: same-origin

Cross-Origin-Embedder-Policy: require-corp

Cross-Origin Isolation

The screenshot shows the Chrome DevTools Application tab open. On the left, there's a sidebar with sections for Back-forward Cache, Background Services (including Background Fetch, Background Sync, Notifications, Payment Handler, Periodic Background Sync, and Push Messaging), and Frames. Under Frames, the 'top' frame is selected, indicated by a blue bar. In the main panel, under the 'Security & Isolation' section, it shows 'Secure Context' as 'Yes', 'Cross-Origin Isolated' as 'Yes', 'Cross-Origin Embedder Policy' as 'RequireCorp', and 'Cross-Origin Opener Policy' as 'SameOriginPlusCoep'. Below this, the 'API availability' section notes that availability depends on the document being cross-origin isolated, with a link to 'Learn more'. It also lists 'SharedArrayBuffers' as 'available, transferable'.

Setting	Value
Secure Context	Yes
Cross-Origin Isolated	Yes
Cross-Origin Embedder Policy	RequireCorp
Cross-Origin Opener Policy	SameOriginPlusCoep
SharedArrayBuffers	available, transferable

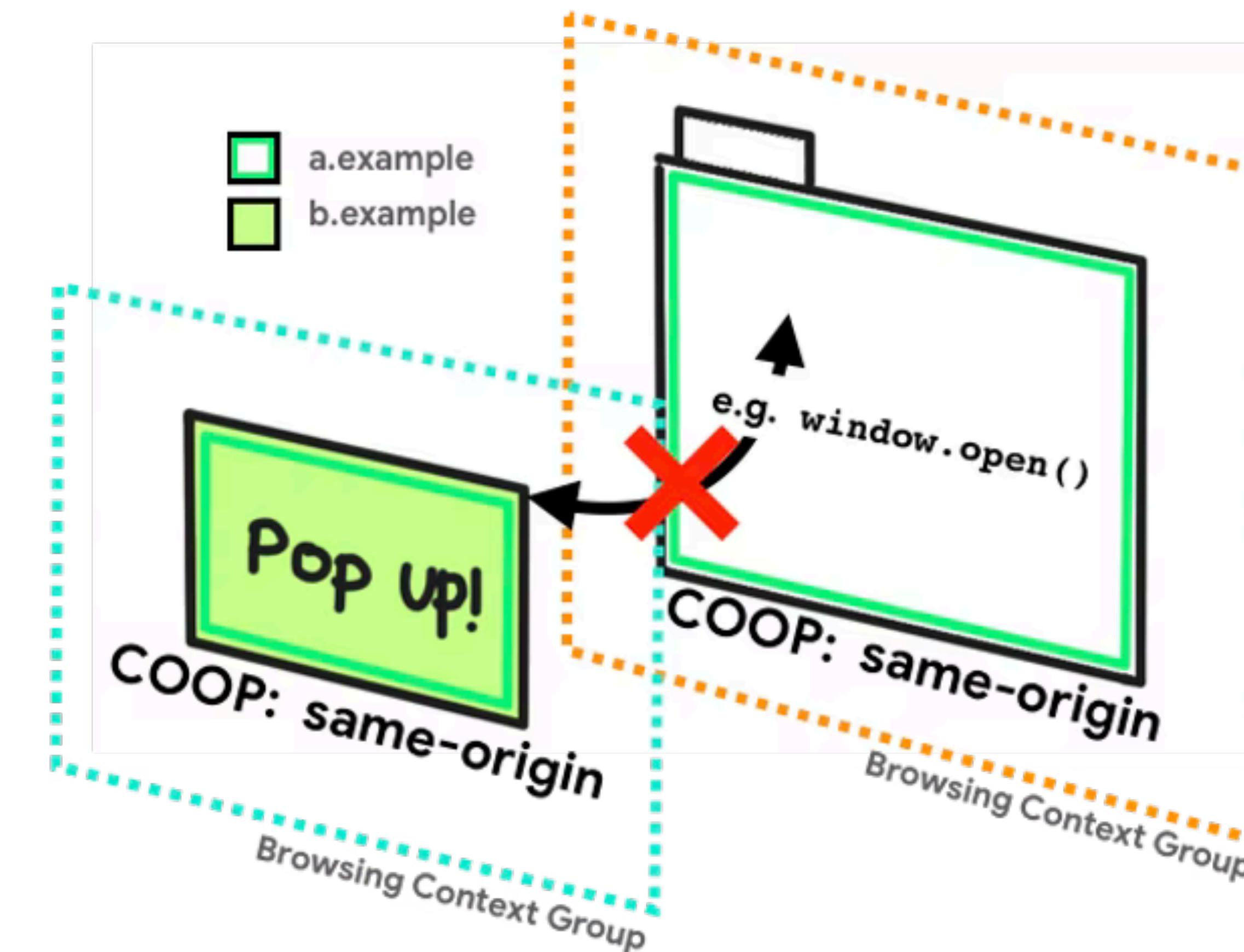
Cross-Origin Isolation

The screenshot shows the Chrome DevTools Application tab interface. On the left, there's a sidebar with sections for Back-forward Cache, Background Services (including Background Fetch, Background Sync, Notifications, Payment Handler, Periodic Background Sync, and Push Messaging), and Frames (with top selected). The main area is titled "Security & Isolation" and contains the following information:

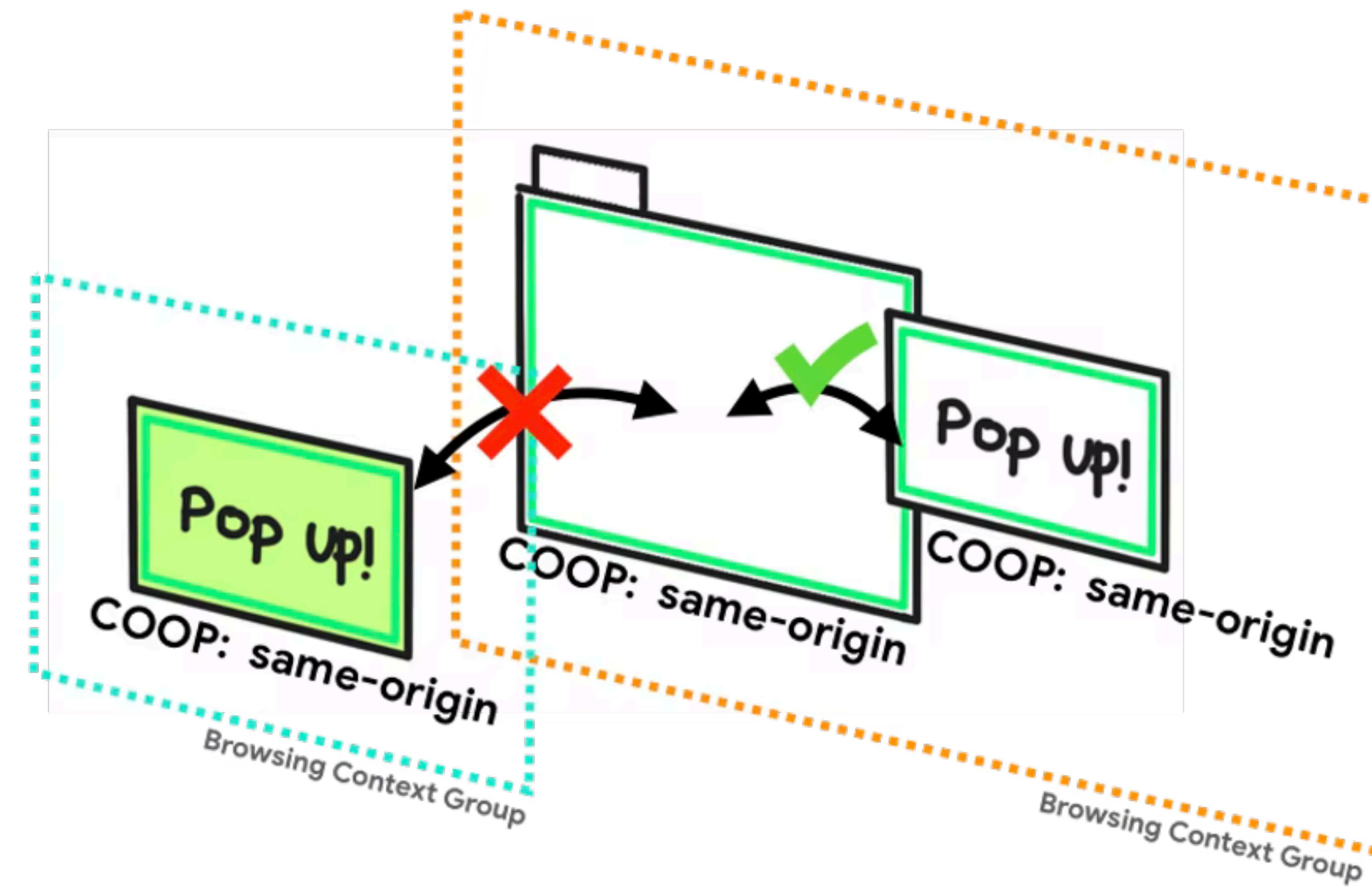
- Secure Context: Yes
- Cross-Origin Isolated: Yes (highlighted with a red box)
- Cross-Origin Embedder Policy: RequireCorp
- Cross-Origin Opener Policy: SameOriginPlusCoep

Below this, under "API availability", it says: "Availability of certain APIs depends on the document being cross-origin isolated." A link to "Learn more" is provided. Under "SharedArrayBuffers", it says: "available, transferable".

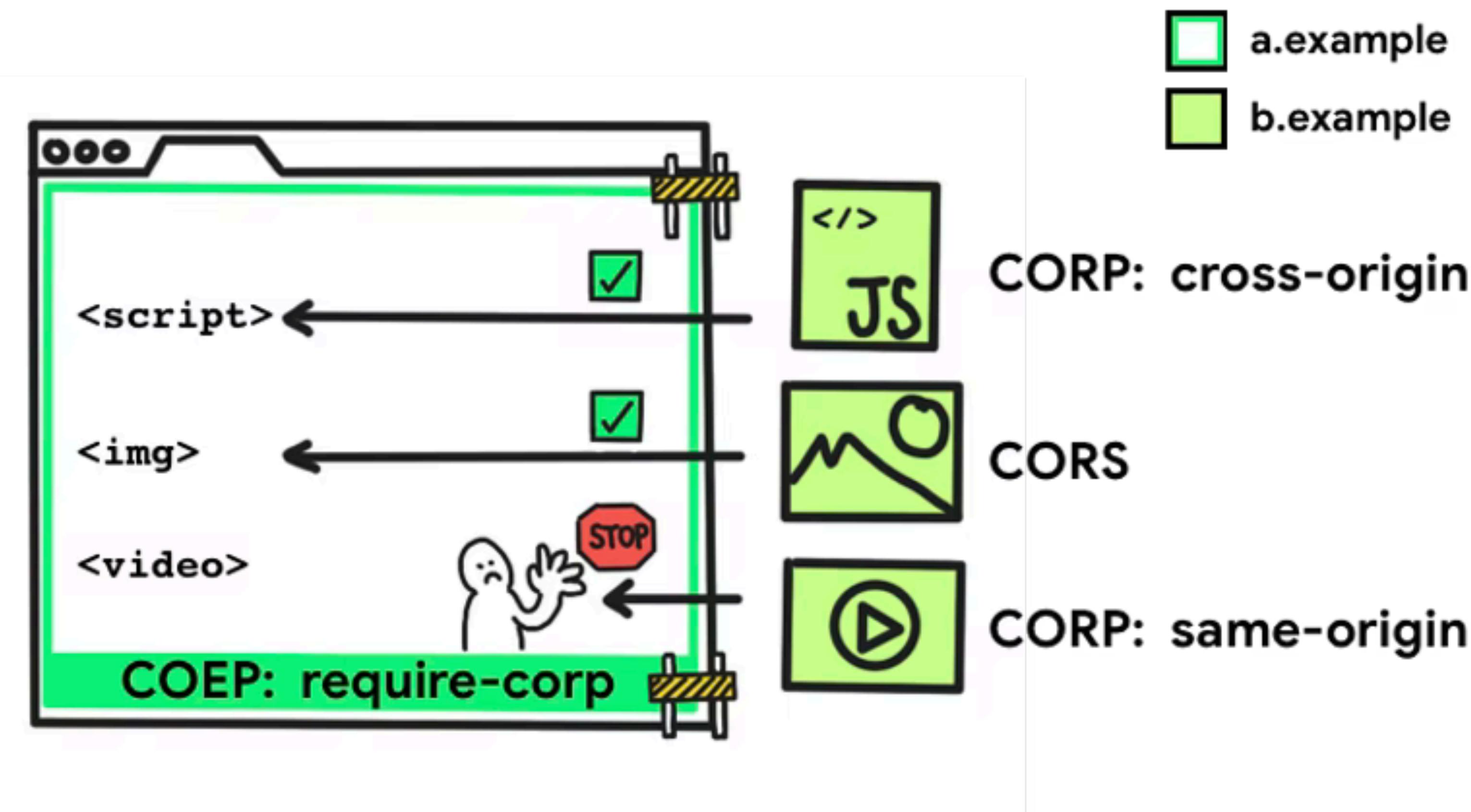
Cross-Origin-Opener-Policy



Cross-Origin-Opener-Policy



Cross-Origin-Embedder-Policy



What's the PITA?

The PITA

- Difficult to deploy at scale
- Requires all subresources to explicitly opt-in
- Dependency problem for sites that have user generated content
 - Google Earth
 - Social Media
 - Forums
 - StackBlitz 😞

Credentialless

~~WINTER~~ IS COMING



Credentialless

Cross-Origin-Opener-Policy: same-origin

Cross-Origin-Embedder-Policy: credentialless

Credentialless

- Does not require CORP header for every subresource
- Resources are requested without credentials
 - Cookies
 - Client certificates
 - ...

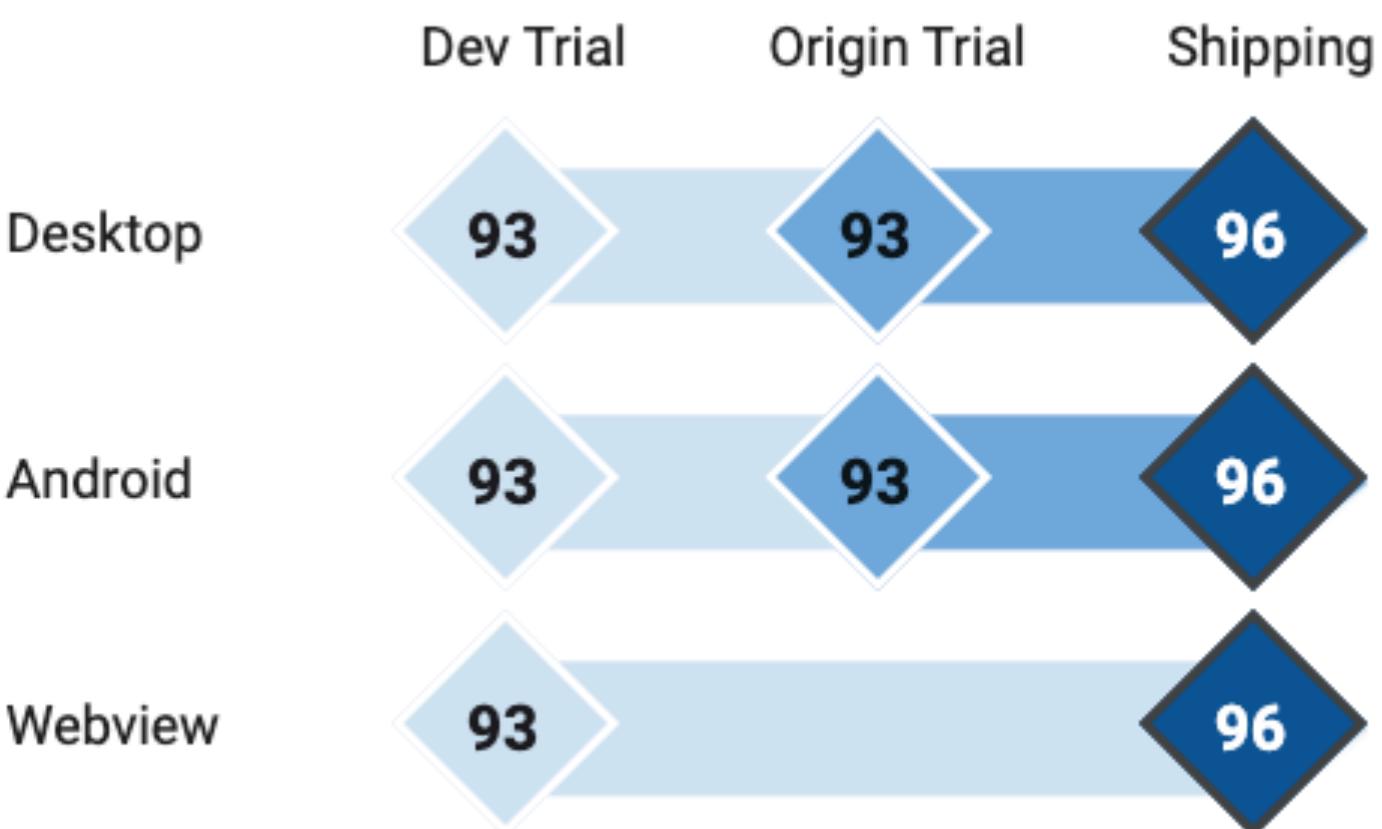


Status in Chromium

Blink components: [Blink>SecurityFeature](#)

Implementation status: Enabled by default ([tracking bug](#))

Estimated milestones:



Consensus & Standardization

After a feature ships in Chrome, the values listed here are not guaranteed to be up to date.

Firefox:

[Worth prototyping](#)

Safari:

[No signal](#)

Web Developers:

Positive

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 1-fibonacci > Exercise 3

Implement by using 1 worker which is aborted by using shared memory

Importing other files

Import

- Separate logic in multiple files
- Re-use utility functions that are also used by the main thread
- Synchronous
- require for Web Workers

Import

```
function fibonacci(position) {  
    if (position < 2) {  
        return position;  
    }  
  
    return fibonacci(position - 1) + fibonacci(position - 2);  
}
```

Import Scripts

```
self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.js

Import Scripts

```
self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.js

Import Scripts

```
importScripts('./fibonacci.js');

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.js

Downsides?

Downsides

- Pollutes the worker global scope
- Not sure what we imported
- Private utility functions leaked to the outside

Solution?

Import Scripts - ESM

```
function fibonacci(position) {  
    if (position < 2) {  
        return position;  
    }  
  
    return fibonacci(position - 1) + fibonacci(position - 2);  
}
```

Import Scripts - ESM

```
export function fibonacci(position) {  
    if (position < 2) {  
        return position;  
    }  
  
    return fibonacci(position - 1) + fibonacci(position - 2);  
}
```

Import Scripts - ESM

```
export function fibonacci(position) {  
    if (position < 2) {  
        return position;  
    }  
  
    return fibonacci(position - 1) + fibonacci(position - 2);  
}
```

fibonacci.mjs

Import Scripts - ESM

```
importScripts('./fibonacci.js');

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.mjs

Import Scripts - ESM

```
importScripts('./fibonacci.js');

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.mjs

Import Scripts - ESM

```
import { fibonacci } from './fibonacci.mjs';

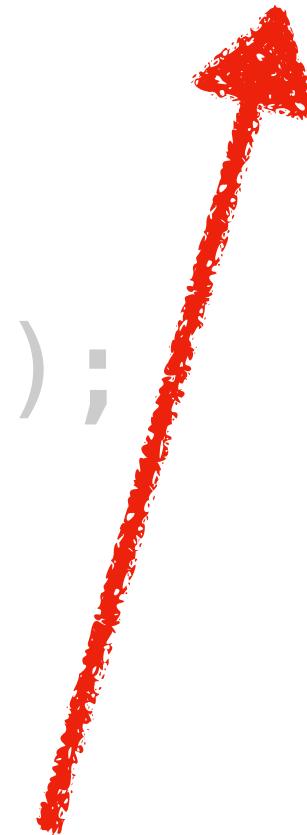
self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.mjs

Import Scripts - ESM

```
import { fibonacci } from './fibonacci.mjs';

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

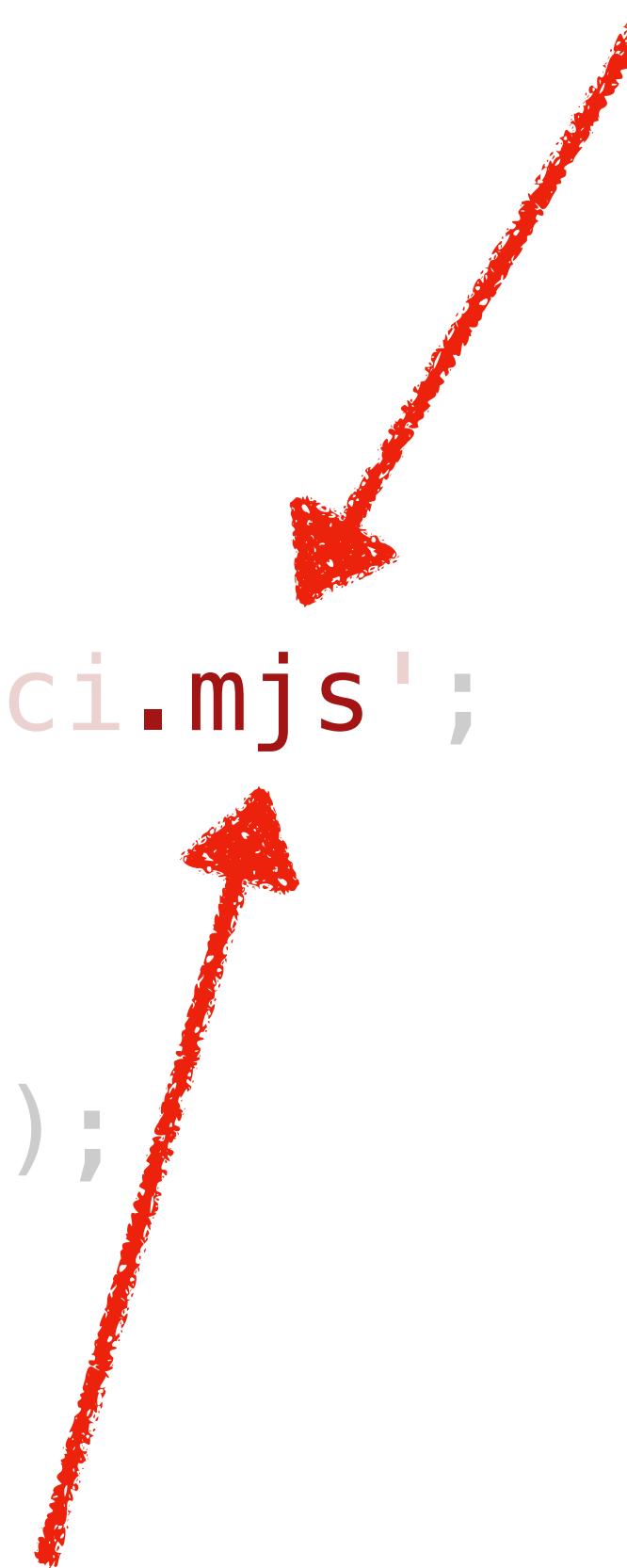


.js works as well

Import Scripts - ESM

Extensions are required in ESM

```
import { fibonacci } from './fibonacci.mjs';  
  
self.onmessage = ({data}) => {  
  self.postMessage(fibonacci(data));  
};
```



.js works as well

Creating a Web Worker - ESM

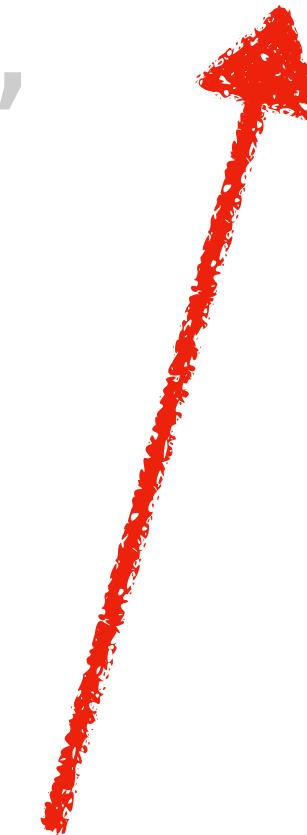
```
const worker = new Worker('./worker.mjs', {  
  name: 'My Awesome WebWorker',  
  type: 'module'  
});
```

Creating a Web Worker - ESM

```
const worker = new Worker('./worker.mjs', {  
  name: 'My Awesome WebWorker',  
  type: 'module'  
});
```

Creating a Web Worker - ESM

```
const worker = new Worker('./worker.mjs', {  
  name: 'My Awesome WebWorker',  
  type: 'module'  
});
```



.js works as well

Import Scripts - ESM

```
import { fibonacci } from 'https://unpkg.com/.../esm/index.mjs';

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

Import Scripts - ESM

```
import { fibonacci } from 'https://unpkg.com/.../esm/index.mjs';

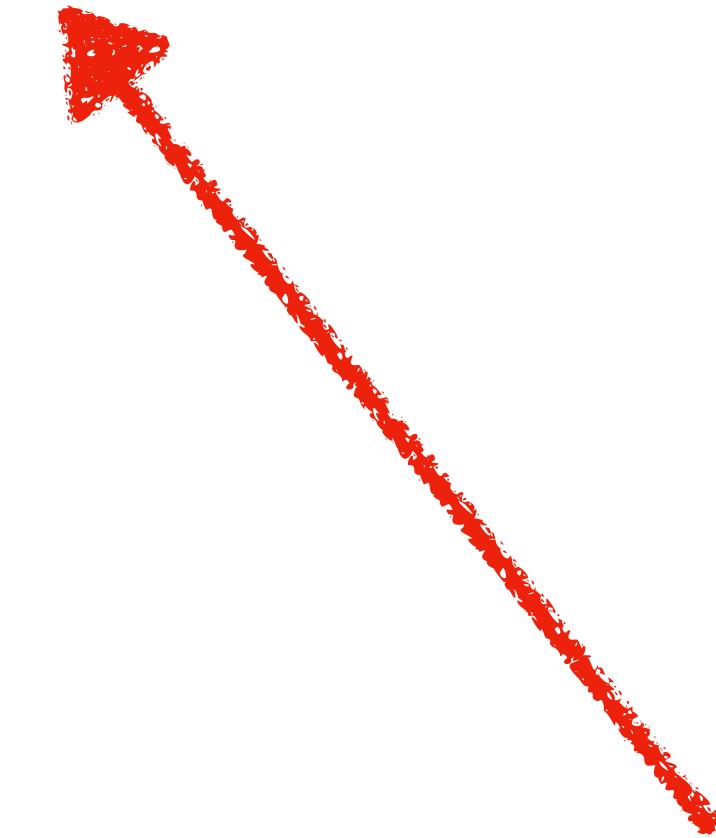
self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```

worker.mjs

Import Scripts - ESM

```
import { fibonacci } from 'https://unpkg.com/.../esm/index.mjs';

self.onmessage = ({data}) => {
  self.postMessage(fibonacci(data));
};
```



Requests are synchronous

Can we make Web Workers even
more enjoyable?

Comlink

Comlink

<https://github.com/GoogleChromeLabs/comlink>

Comlink

- 1.1kB
- No more `postMessage` and `onmessage`
- RPC implementation for `postMessage`

Comlink - Worker - Object

```
const obj = {  
  counter: 0,  
  inc() {  
    this.counter++;  
  }  
};
```

worker.mjs

Comlink - Worker - Object

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const obj = {
    counter: 0,
    inc() {
        this.counter++;
    }
};
```

worker.mjs

Comlink - Worker - Object

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const obj = {
    counter: 0,
    inc() {
        this.counter++;
    }
};

Comlink.expose(obj);
```

worker.mjs

Comlink - Worker - Object

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const obj = {
  counter: 0,
  inc() {
    this.counter++;
  }
};

Comlink.expose(obj);
```

Comlink - Worker - Functions

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

Comlink - Worker - Functions

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

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

Comlink - Worker - Functions

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

function calculateSum(a, b) {
  return a + b;
}

Comlink.expose(calculateSum);
```

worker.mjs

Comlink - Worker - Functions

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

function calculateSum(a, b) {
  return a + b;
}

Comlink.expose(calculateSum);
```

Comlink - Main Thread

Comlink - Main Thread

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});
```

Comlink - Main Thread

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});
```



Required because of ESM syntax (import statement)

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

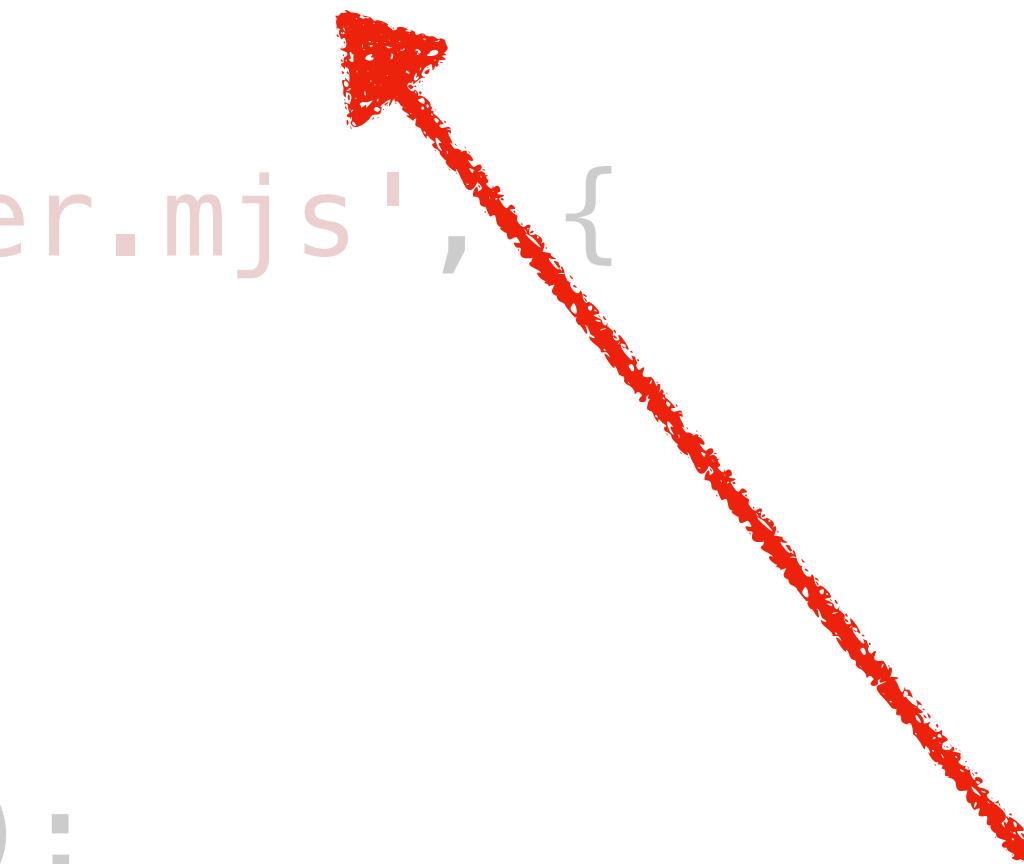
const obj = Comlink.wrap(worker);
```

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);
```



ESM syntax. <script type="module"> required!

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);

console.log(`Counter: ${await obj.counter}`);
```

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);

console.log(`Counter: ${await obj.counter}`);
```



Only ESM supports top-level await

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);

console.log(`Counter: ${await obj.counter}`);

await obj.inc();
```

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);

console.log(`Counter: ${await obj.counter}`);

await obj.inc();

console.log(`Counter: ${await obj.counter}`);
```

Comlink - Main Thread

```
import * as Comlink from 'https://unpkg.com/comlink/dist/esm/comlink.mjs';

const worker = new Worker('worker.mjs', {
  type: 'module'
});

const obj = Comlink.wrap(worker);

console.log(`Counter: ${await obj.counter}`);

await obj.inc();

console.log(`Counter: ${await obj.counter}`);
```

What about callbacks?

Comlink - Callbacks

```
async function remoteFunction(cb) {  
    await cb('A string from a worker');  
}  
  
Comlink.expose(remoteFunction);
```

Comlink - Callbacks

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});
```

```
const remoteFunction = Comlink.wrap(worker);
```

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});  
  
const remoteFunction = Comlink.wrap(worker);  
  
function callback(value) {  
  console.log(`Result: ${value}`);  
}
```

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});  
  
const remoteFunction = Comlink.wrap(worker);  
  
function callback(value) {  
  console.log(`Result: ${value}`);  
}  
  
await remoteFunction(callback);
```

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});  
  
const remoteFunction = Comlink.wrap(worker);  
  
function callback(value) {  
  console.log(`Result: ${value}`);  
}  
  
await remoteFunction(callback);
```

Functions cannot be structurally cloned

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});  
  
const remoteFunction = Comlink.wrap(worker);  
  
function callback(value) {  
  console.log(`Result: ${value}`);  
}  
  
await remoteFunction(Comlink.proxy(callback));
```

Comlink - Callbacks

```
const worker = new Worker('worker.mjs', {  
  type: 'module'  
});  
  
const remoteFunction = Comlink.wrap(worker);  
  
function callback(value) {  
  console.log(`Result: ${value}`);  
}  
  
await remoteFunction(Comlink.proxy(callback));
```

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 1-fibonacci

Refactor the previous abort solution by using Comlink

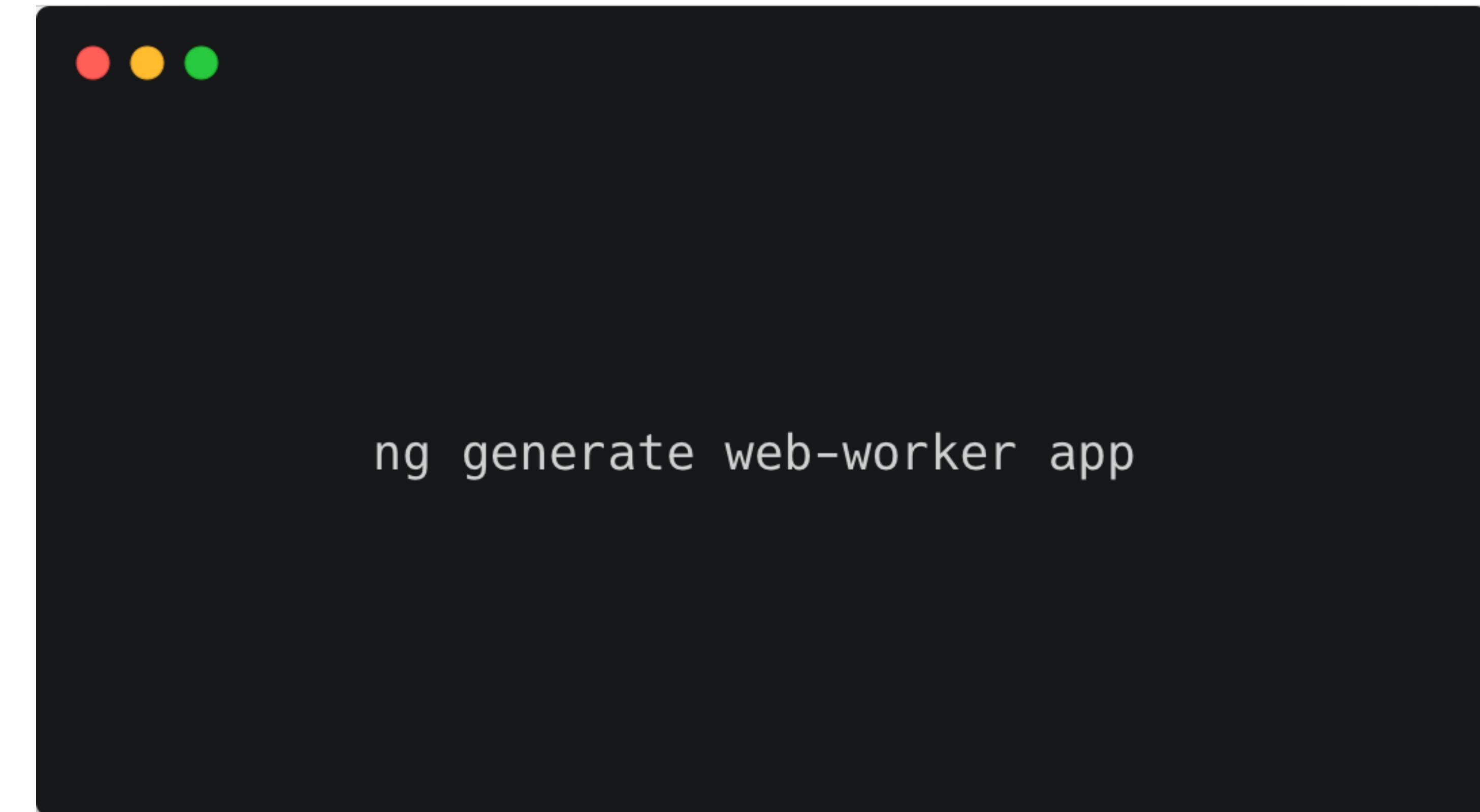
Angular Support

Web Worker



Out-of-the-box support



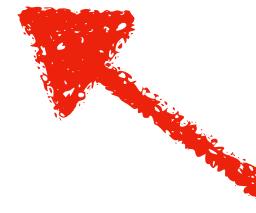


app.worker.ts

```
/// <reference lib="webworker" />

addEventListener('message', ({ data }) => {
  const response = `worker response to ${data}`;
  postMessage(response);
});
```

app.worker.ts



This is a .ts file!!!

```
/// <reference lib="webworker" />

addEventListener('message', ({ data }) => {
  const response = `worker response to ${data}`;
  postMessage(response);
});
```

Updated app.component.ts

```
if (typeof Worker !== 'undefined') {
  // Create a new
  const worker = new Worker(new URL('./app.worker', import.meta.url));
  worker.onmessage = ({ data }) => {
    console.log(`page got message: ${data}`);
  };
  worker.postMessage('hello');
} else {
  // Web Workers are not supported in this environment.
  // You should add a fallback so that your program still executes correctly.
}
```

Let's clean it up a bit

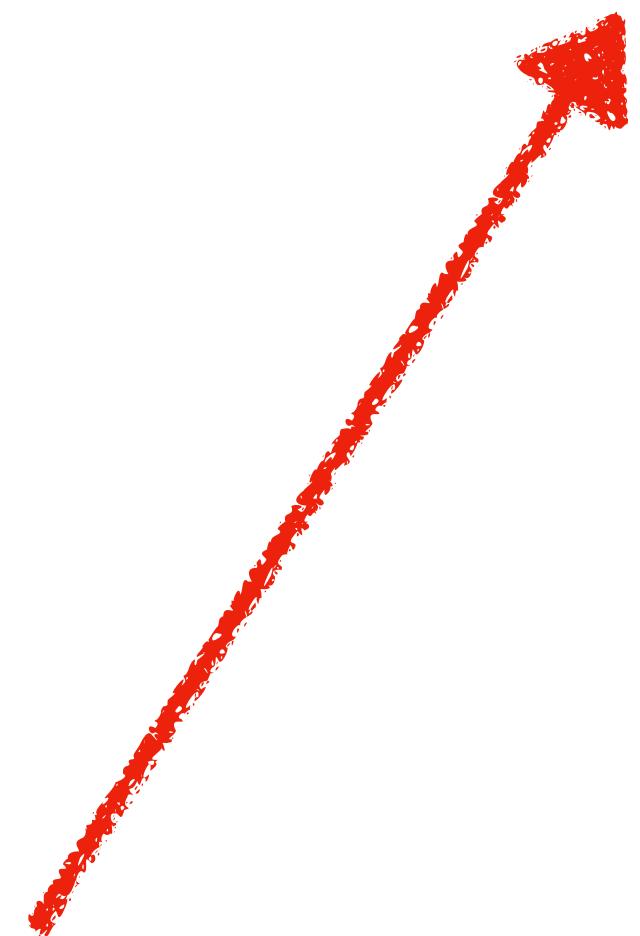
```
const worker = new Worker(new URL('./app.worker', import.meta.url));  
  
worker.onmessage = ({ data }) => {  
  console.log(`page got message: ${data}`);  
};  
  
worker.postMessage('hello');
```

Let's clean it up a bit

```
const worker = new Worker(new URL('./app.worker', import.meta.url));  
  
worker.onmessage = ({ data }) => {  
  console.log(`page got message: ${data}`);  
};  
  
worker.postMessage('hello');
```

Let's clean it up a bit

```
const worker = new Worker(new URL('./app.worker', import.meta.url));  
  
worker.onmessage = ({ data }) => {  
  console.log(`page got message: ${data}`);  
};  
  
worker.postMessage('hello');
```



Base URL of the document
e.g. <https://app.ng-be.org/>

Let's clean it up a bit

<https://app.ng-be.org/310.40fbf0d07e872117.js>

```
const worker = new Worker(new URL('./app.worker', import.meta.url));  
  
worker.onmessage = ({ data }) => {  
  console.log(`page got message: ${data}`);  
};  
  
worker.postMessage('hello');
```

Let's clean it up a bit

Runs inside NgZone

```
const worker = new Worker(new URL('./app.worker', import.meta.url));  
  ↓  
worker.onmessage = ({ data }) => {  
  console.log(`page got message: ${data}`);  
};  
  
worker.postMessage('hello');
```

Importing libraries

```
/// <reference lib="webworker" />
import * as Comlink from 'comlink';

function fibonacci(position: number) {
    // ...
}

Comlink.expose(fibonacci);
```

Importing libraries

```
/// <reference lib="webworker" />
import * as Comlink from 'comlink';

function fibonacci(position: number) {
    // ...
}

Comlink.expose(fibonacci);
```

Worker existence

```
typeof Worker !== 'undefined'
```

Worker existence

```
typeof Worker !== 'undefined'
```



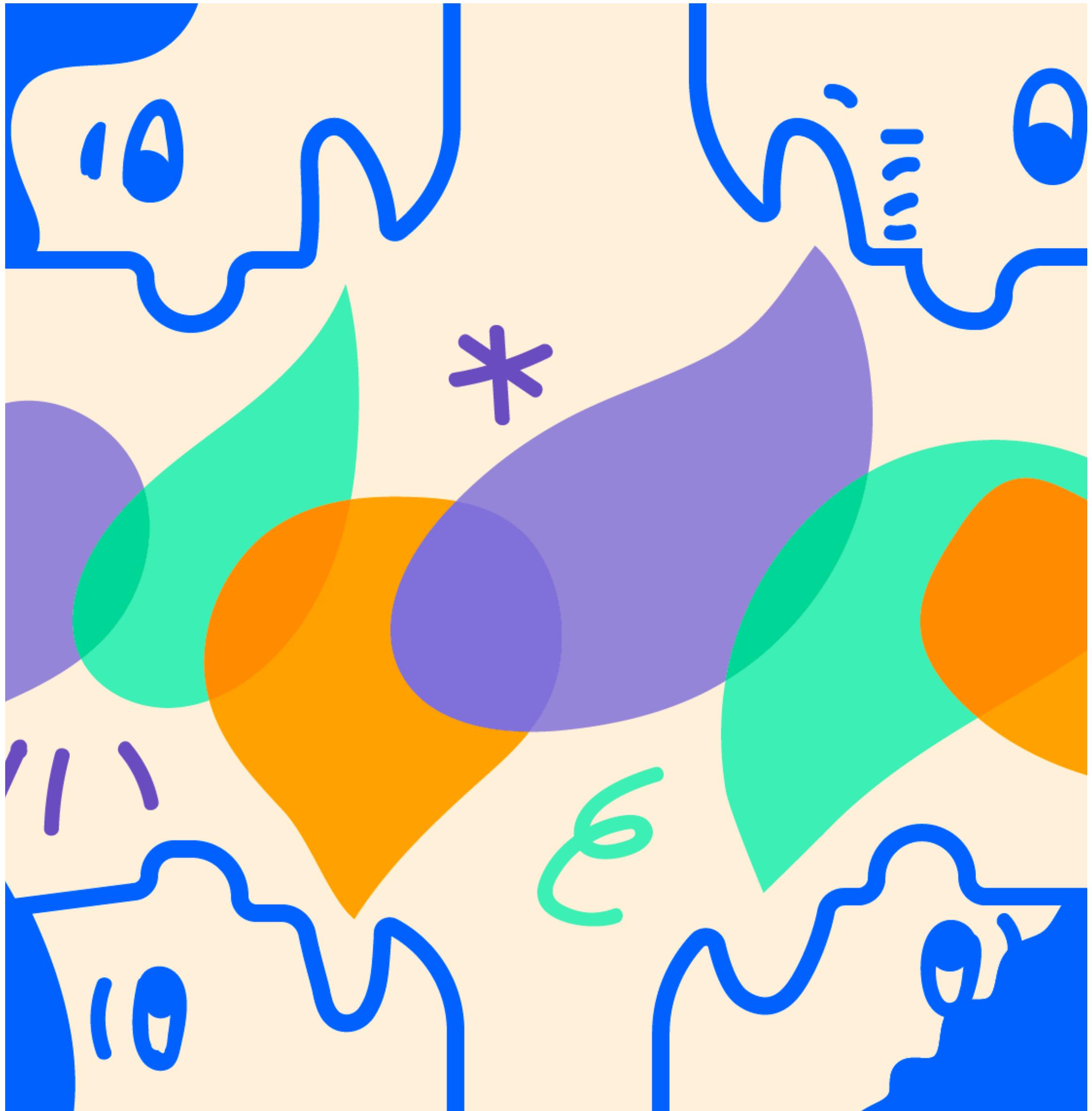
Some environments might not have this available
For instance, @angular/platform-server used in SSR

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 2-angular-starter

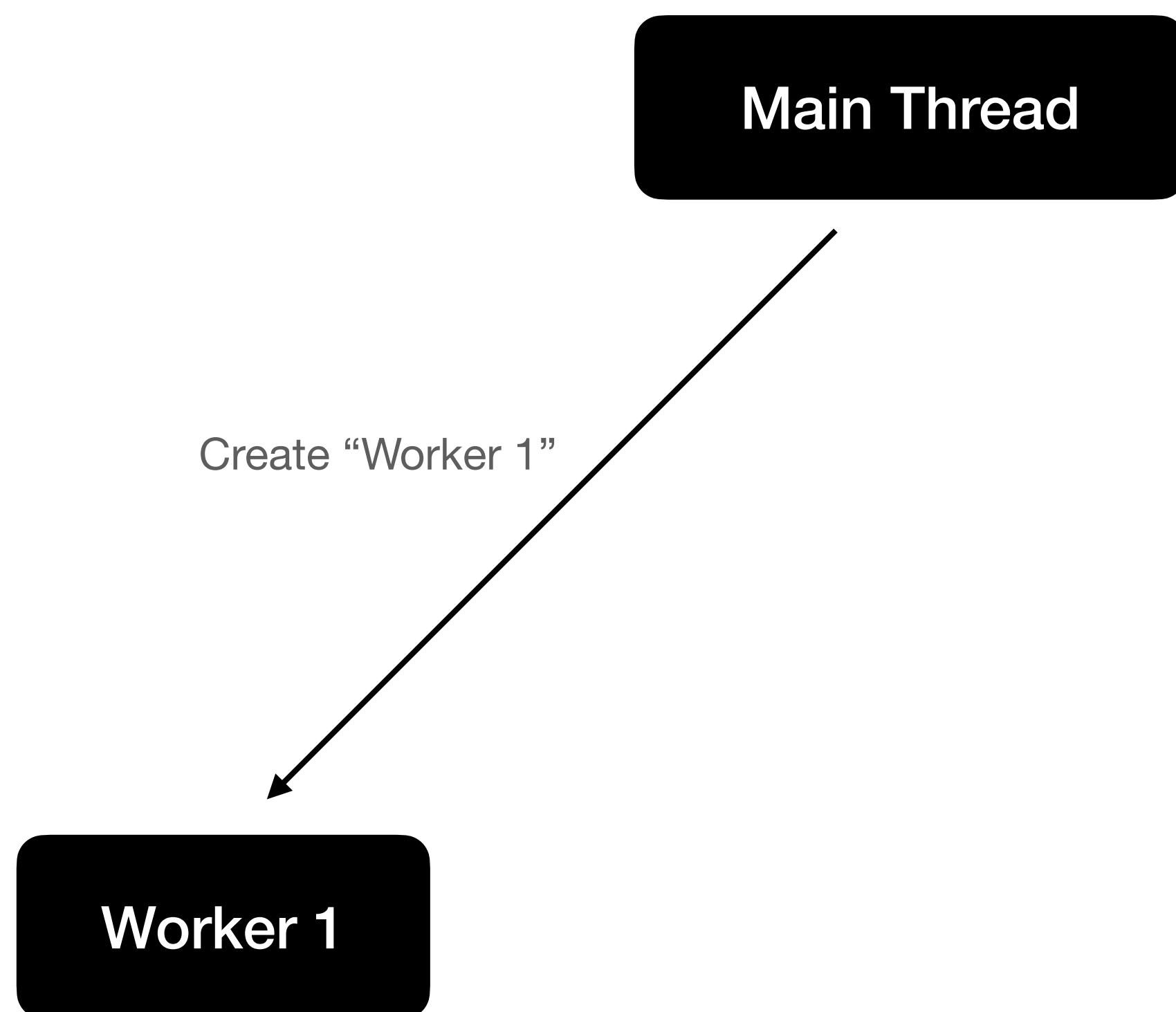
Web Worker intercommunication



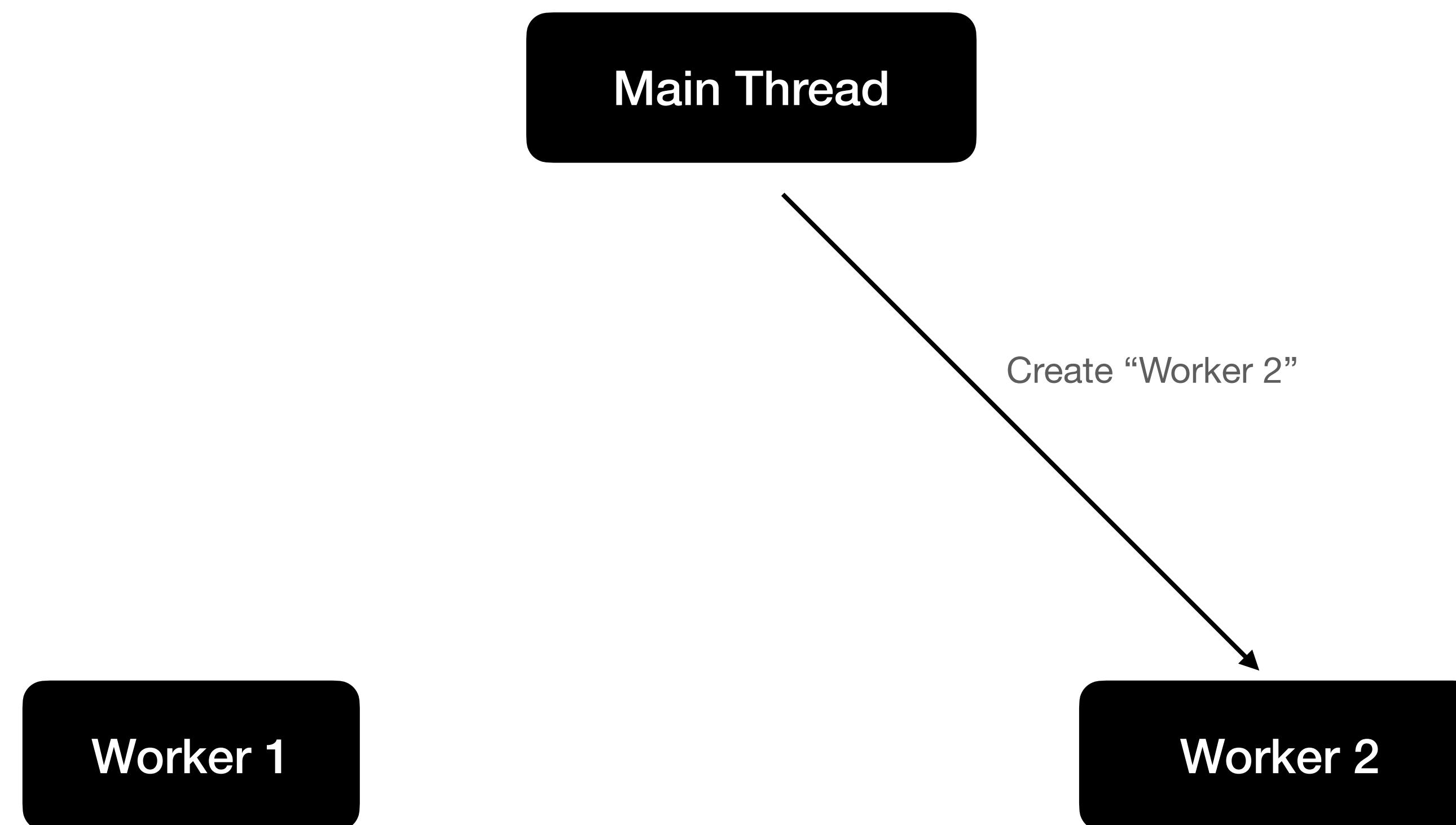
Intercommunication

Main Thread

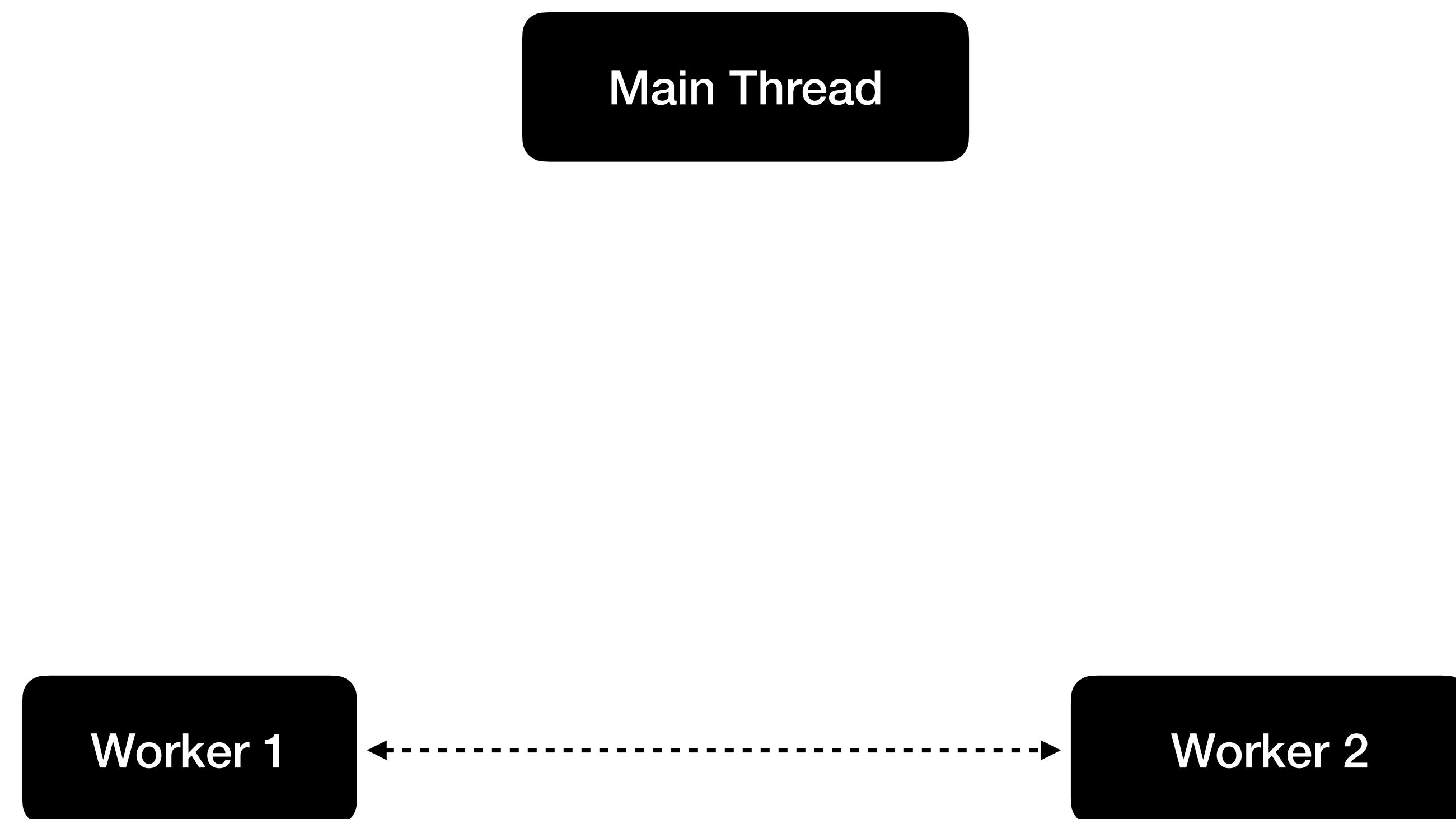
Intercommunication



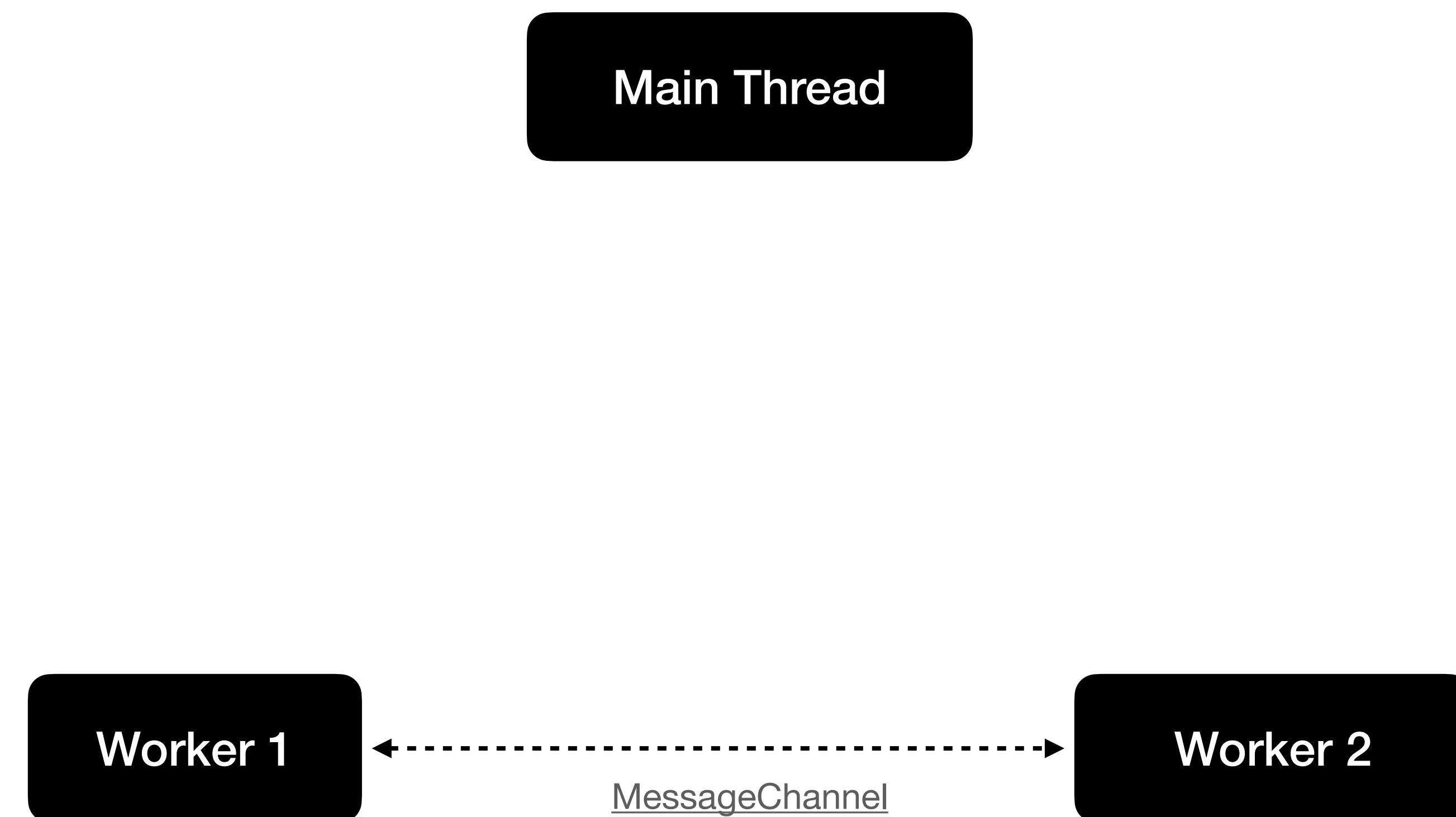
Intercommunication



Intercommunication



Intercommunication



MessageChannel

```
new MessageChannel();
```

MessageChannel

```
interface MessageChannel {  
    /**  
     * Returns the first MessagePort object.  
     */  
    readonly port1: MessagePort;  
  
    /**  
     * Returns the second MessagePort object.  
     */  
    readonly port2: MessagePort;  
}
```

MessageChannel

```
interface MessageChannel {  
    /**  
     * Returns the first MessagePort object.  
     */  
    readonly port1: MessagePort;  
  
    /**  
     * Returns the second MessagePort object.  
     */  
    readonly port2: MessagePort;  
}
```

MessageChannel

```
new MessageChannel();
```

MessageChannel

```
const {port1, port2} = new MessageChannel();
```



MessageChannel

Main Thread

Worker 1

Worker 2

MessageChannel

Main Thread

```
const {port1, port2} = new MessageChannel();
```

Worker 1

Worker 2

MessageChannel

Main Thread

```
const { , port2} = new MessageChannel();
```

port1

Worker 1

Worker 2

MessageChannel

Main Thread

```
const { , } = new MessageChannel();
```

port1

Worker 1

port2

Worker 2

MessageChannel

Main Thread

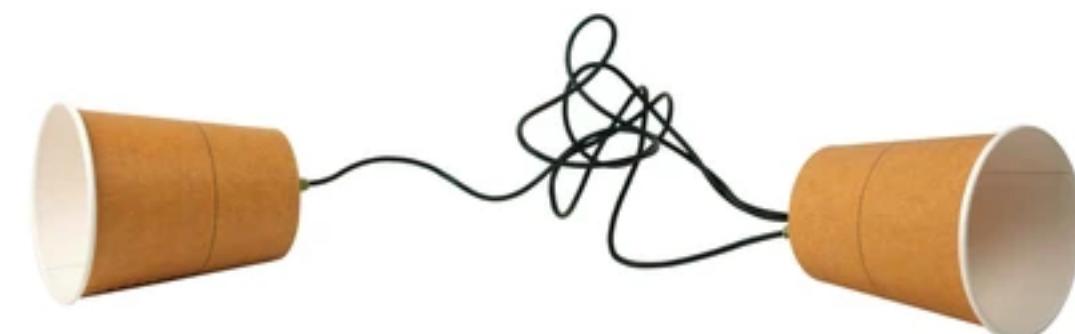
```
const { , } = new MessageChannel();
```

port1

Worker 1

port2

Worker 2



How?

MessageChannel

```
const {port1, port2} = new MessageChannel();
```

```
worker1.postMessage(port1);
```

```
worker2.postMessage(port2);
```



MessageChannel

Uncaught DOMException: Failed to execute 'postMessage' on 'Worker': A MessagePort could not be cloned because it was not transferred.

MessageChannel

Uncaught DOMException: Failed to execute 'postMessage' on 'Worker':
A MessagePort could not be cloned because it was not transferred.

Transferables!

Transferables

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Transferables

```
worker.postMessage(message: any, transfer?: Transferable[]): void;
```

Transferables

An optional array of Transferable objects to transfer ownership of.

Transferables

An optional array of Transferable objects to transfer ownership of.

If the ownership of an object is transferred, it becomes unusable in the context it was sent from and becomes available only to the worker it was sent to.

Transferables

An optional **array of Transferable objects** to transfer ownership of.
If the **ownership** of an object **is transferred**, it becomes **unusable** in the context
it was sent from and becomes available only to the worker it was sent to.

Transferables

- ArrayBuffer
- MessagePort
- ImageBitmap
- No SAB
- No typed array (Int8Array, ...)
- No null

MessageChannel

```
const {port1, port2} = new MessageChannel();
```

```
worker1.postMessage(port1);
```

```
worker2.postMessage(port2);
```

MessageChannel

```
const {port1, port2} = new MessageChannel();
```

```
worker1.postMessage(port1, [port1]);
```

```
worker2.postMessage(port2, [port2]);
```

MessageChannel

MessageChannel

```
self.onmessage = (event) => {  
    const port = event.data  
  
};
```

MessageChannel

```
self.onmessage = (event) => {
  const port = event.data

  port.onmessage = (message) => {
    console.log(message.data)
  };
};
```

MessageChannel

```
self.onmessage = (event) => {
  const port = event.data

  port.onmessage = (message) => {
    console.log(message.data)
  };

  port.postMessage('Hello, NG-BE!');
};
```

MessageChannel

```
self.onmessage = (event) => {
  const port = event.data

  port.onmessage = (message) => {
    console.log(message.data)
  };

  port.postMessage('Hello, NG-BE!');
};
```

MessageChannel

```
port.addEventListener('message', (event) => {
  console.log(event.data);
});
```

MessageChannel

```
port.addEventListener('message', (event) => {
  console.log(event.data);
});

port.start();
```

MessageChannel

```
port.addEventListener('message', (event) => {  
  console.log(event.data);  
});
```

```
port.start();
```



Implied when using `port.onmessage`

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 2-message-channel

Do we have more?

SharedWorker

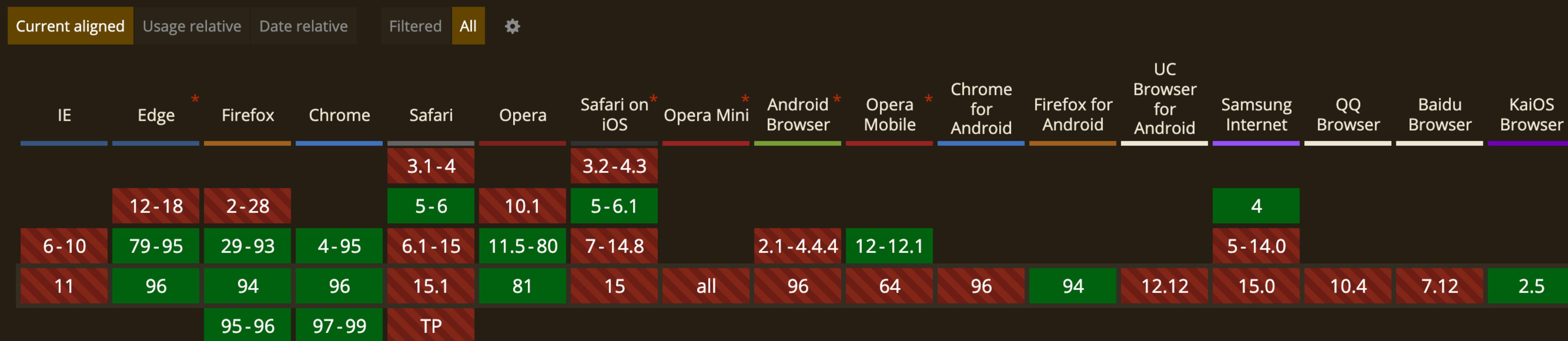
Support

Shared Web Workers - LS

Global

37.25%

Method of allowing multiple scripts to communicate with a single web worker.



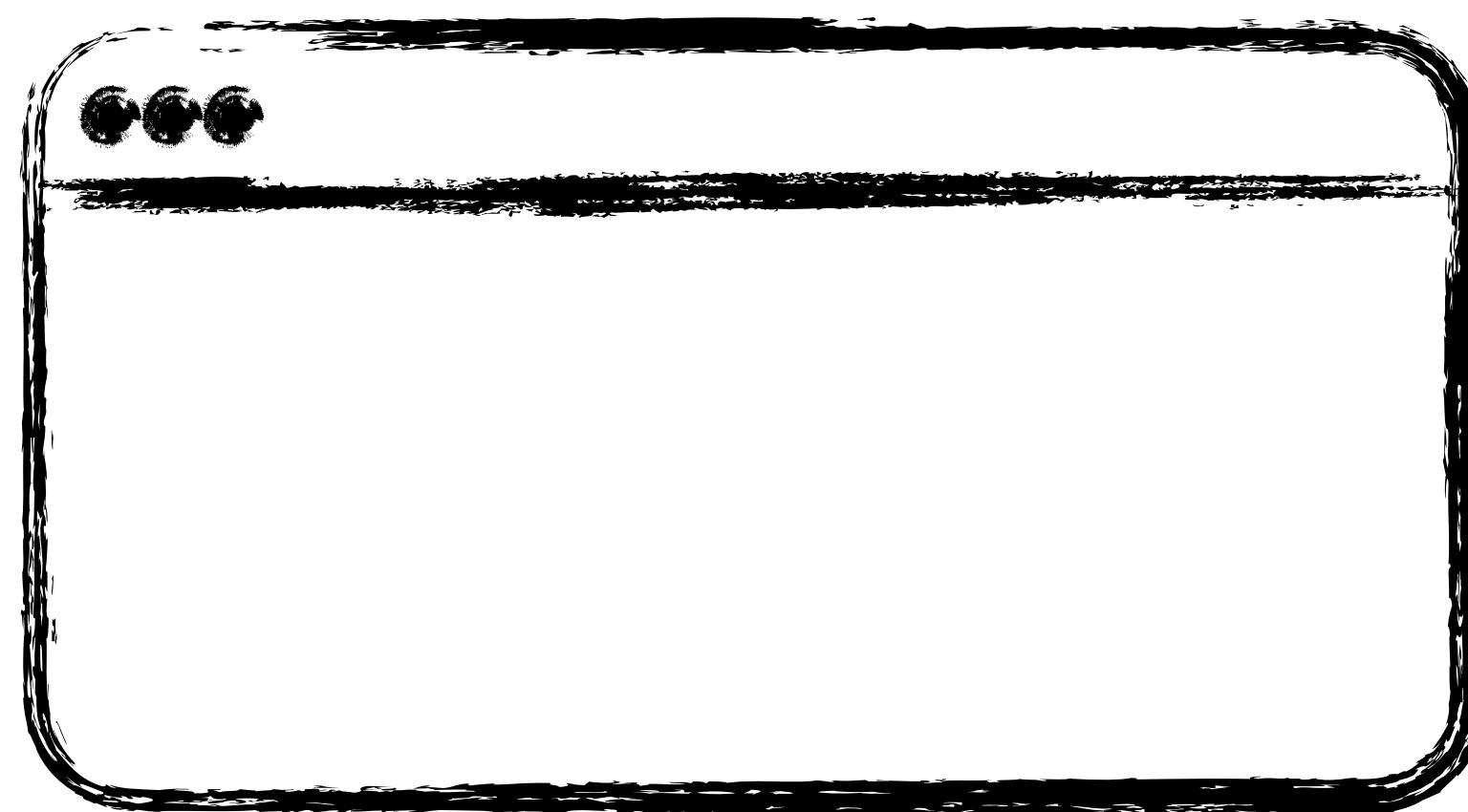
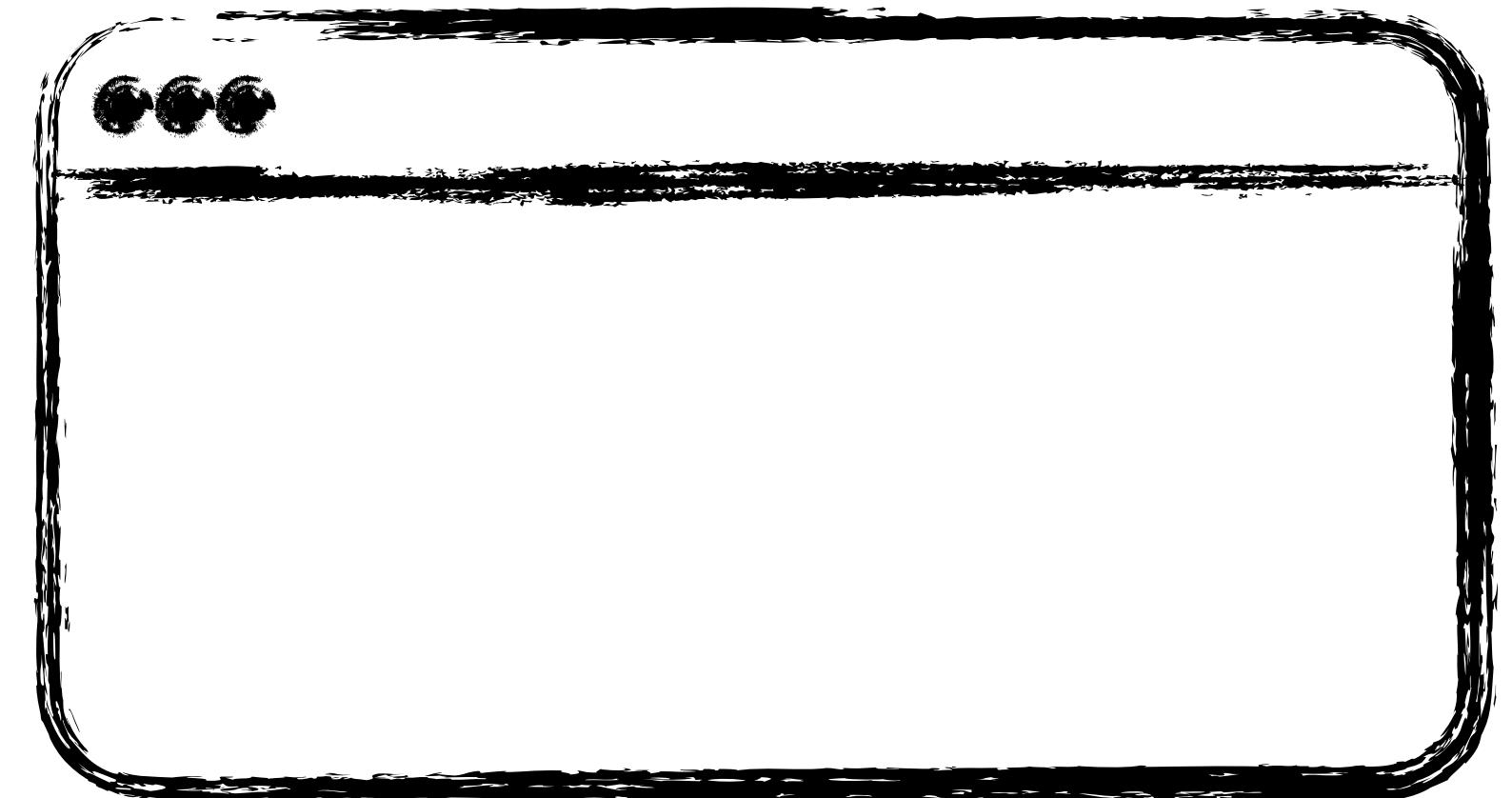
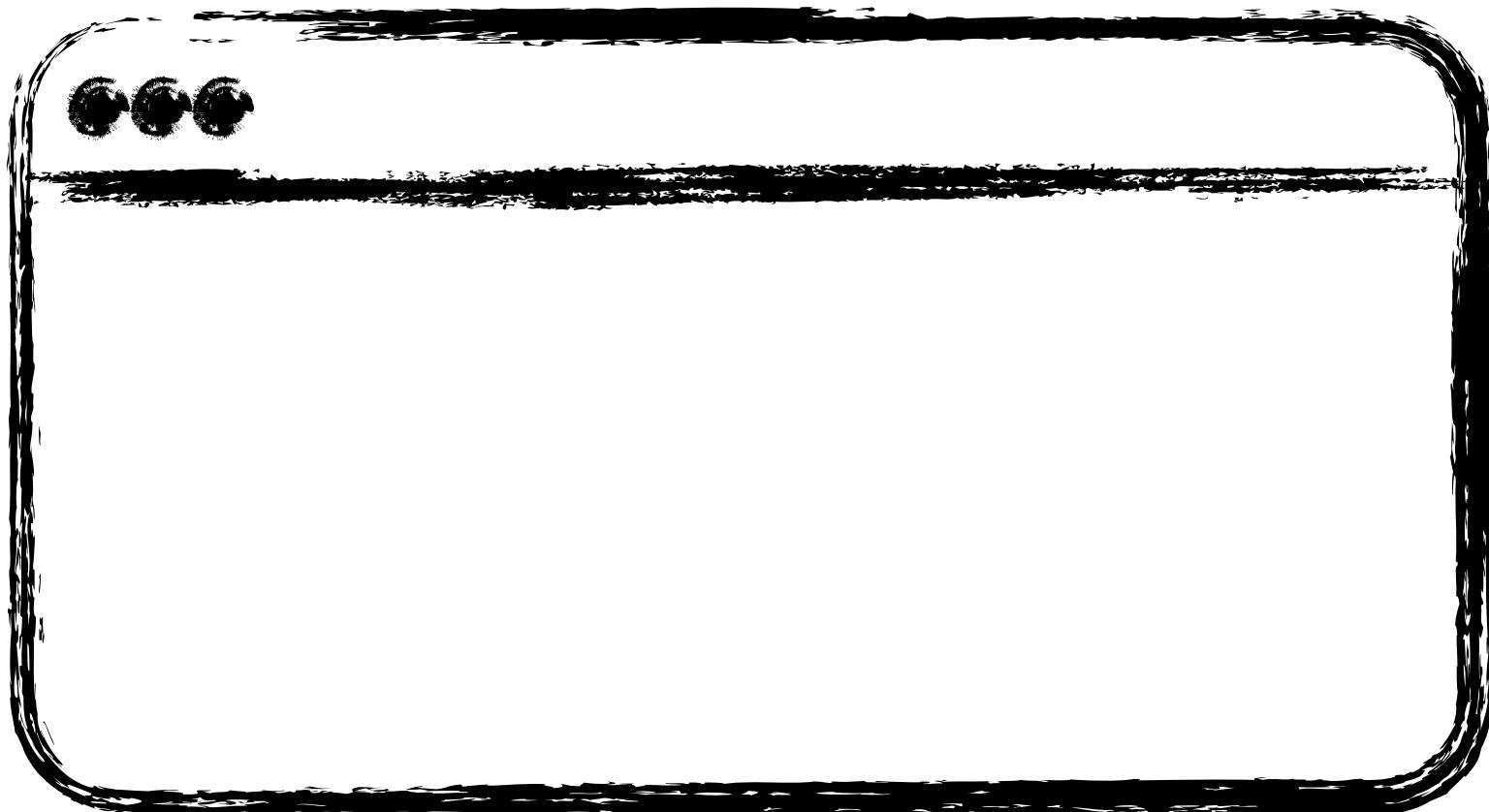
Shared Workers

The `SharedWorker` interface represents a specific kind of worker that can be accessed from several browsing contexts, such as several windows, iframes or even workers. They implement an interface different than dedicated workers and have a different global scope, `SharedWorkerGlobalScope`.

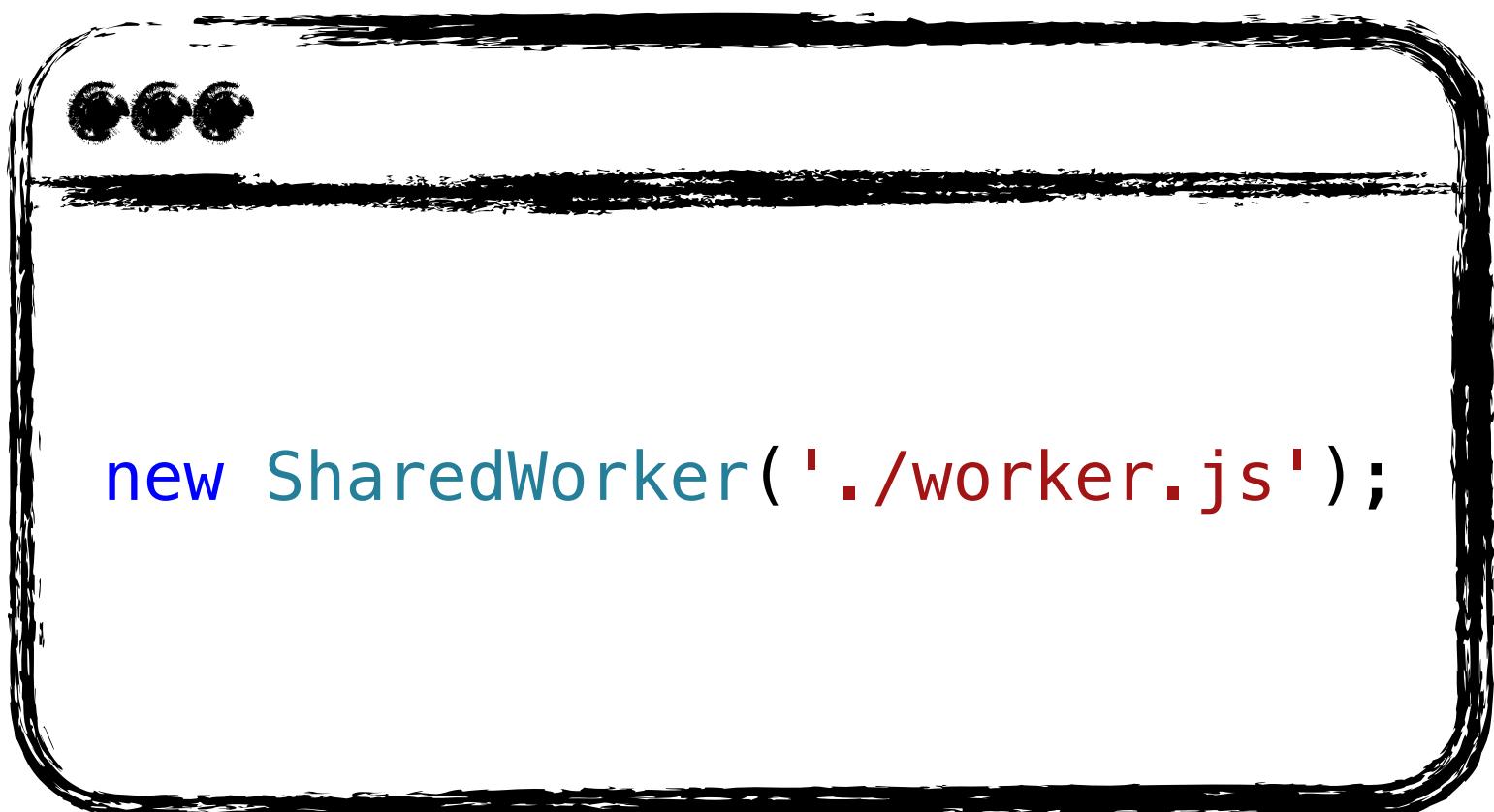
Shared Workers

The `SharedWorker` interface represents a specific kind of worker that can be accessed from several **browsing contexts**, such as several **windows**, **iframes** or even **workers**. They implement an interface different than dedicated workers and have a different global scope, `SharedWorkerGlobalScope`.

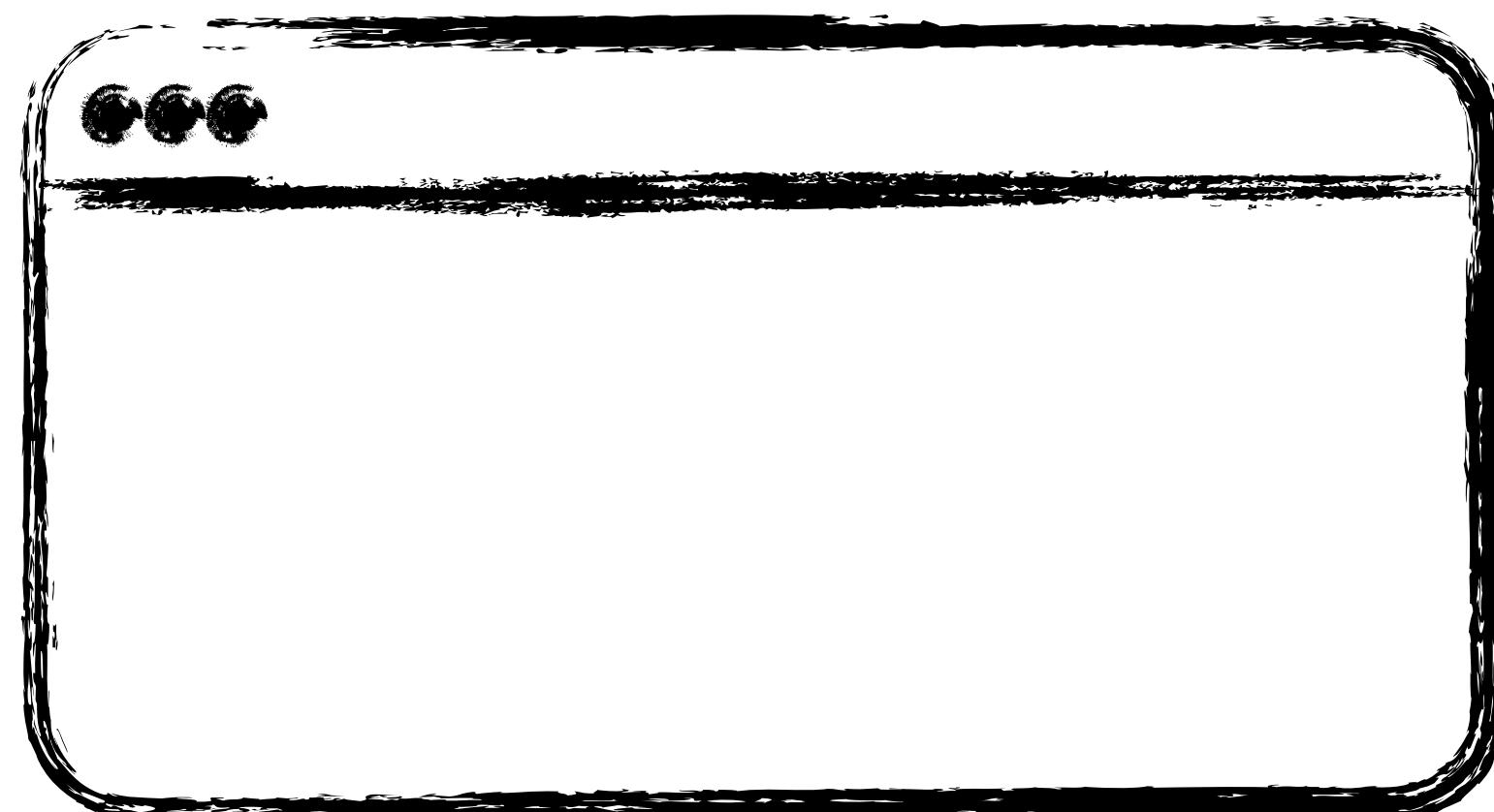
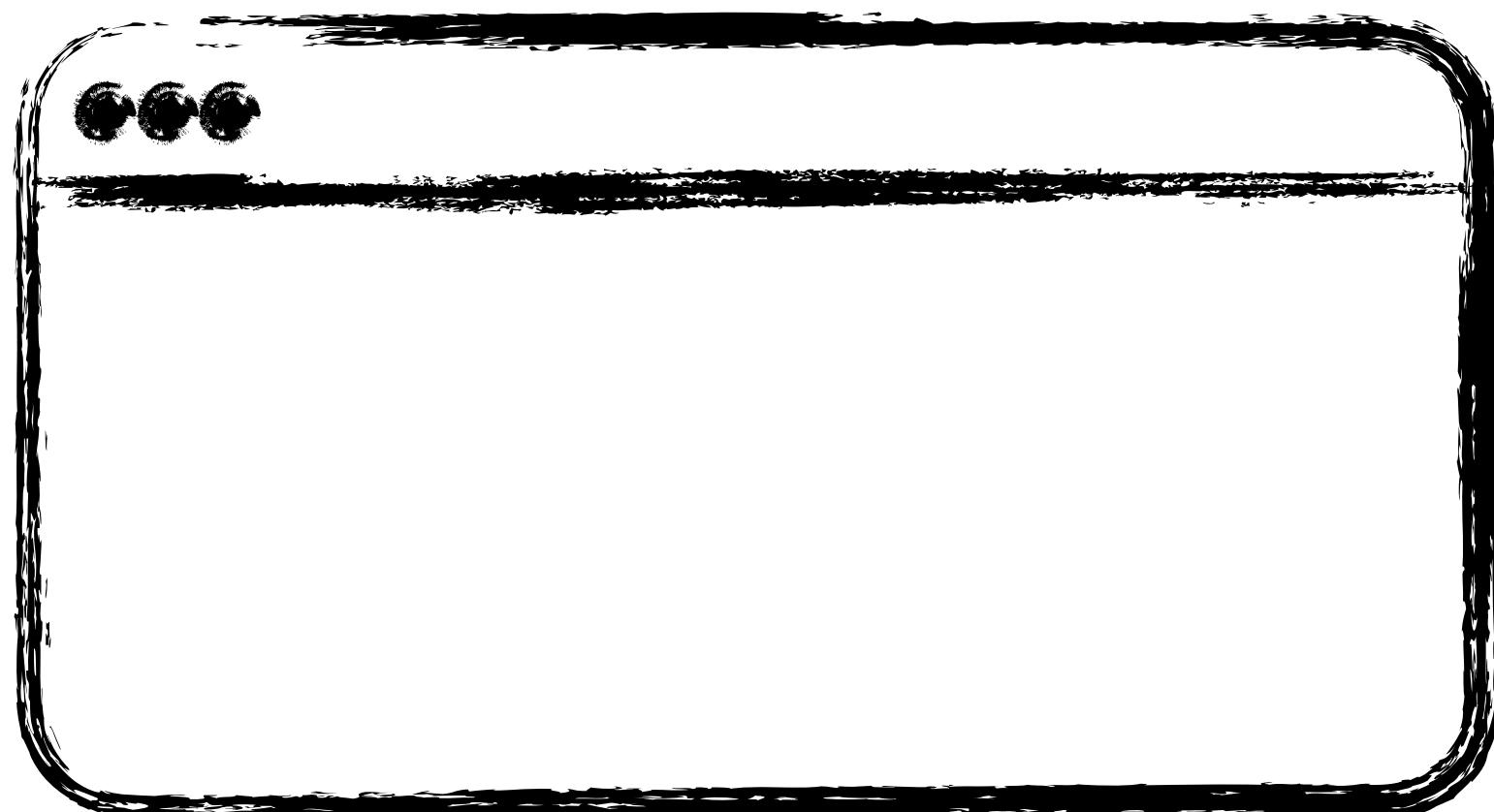
Shared Worker



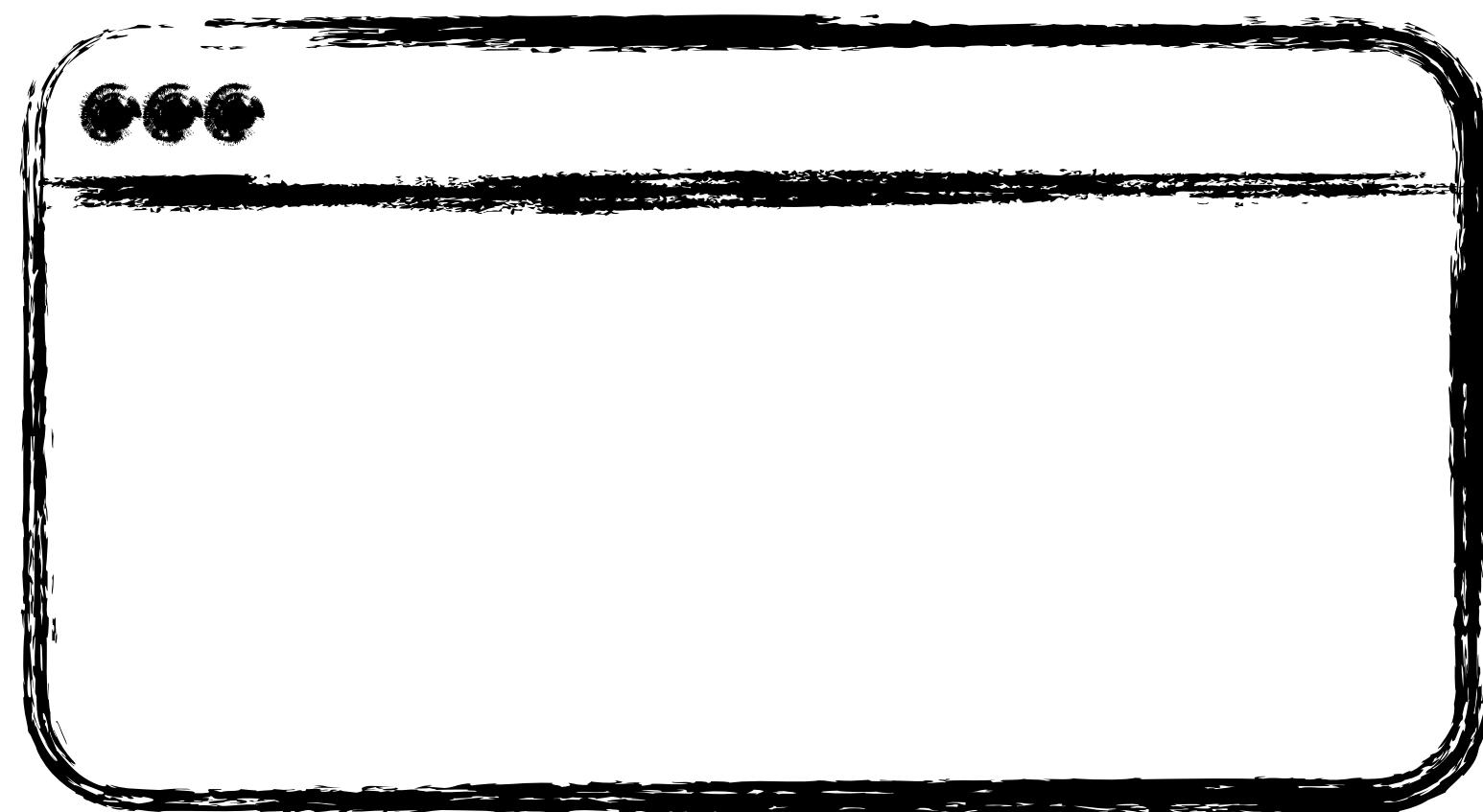
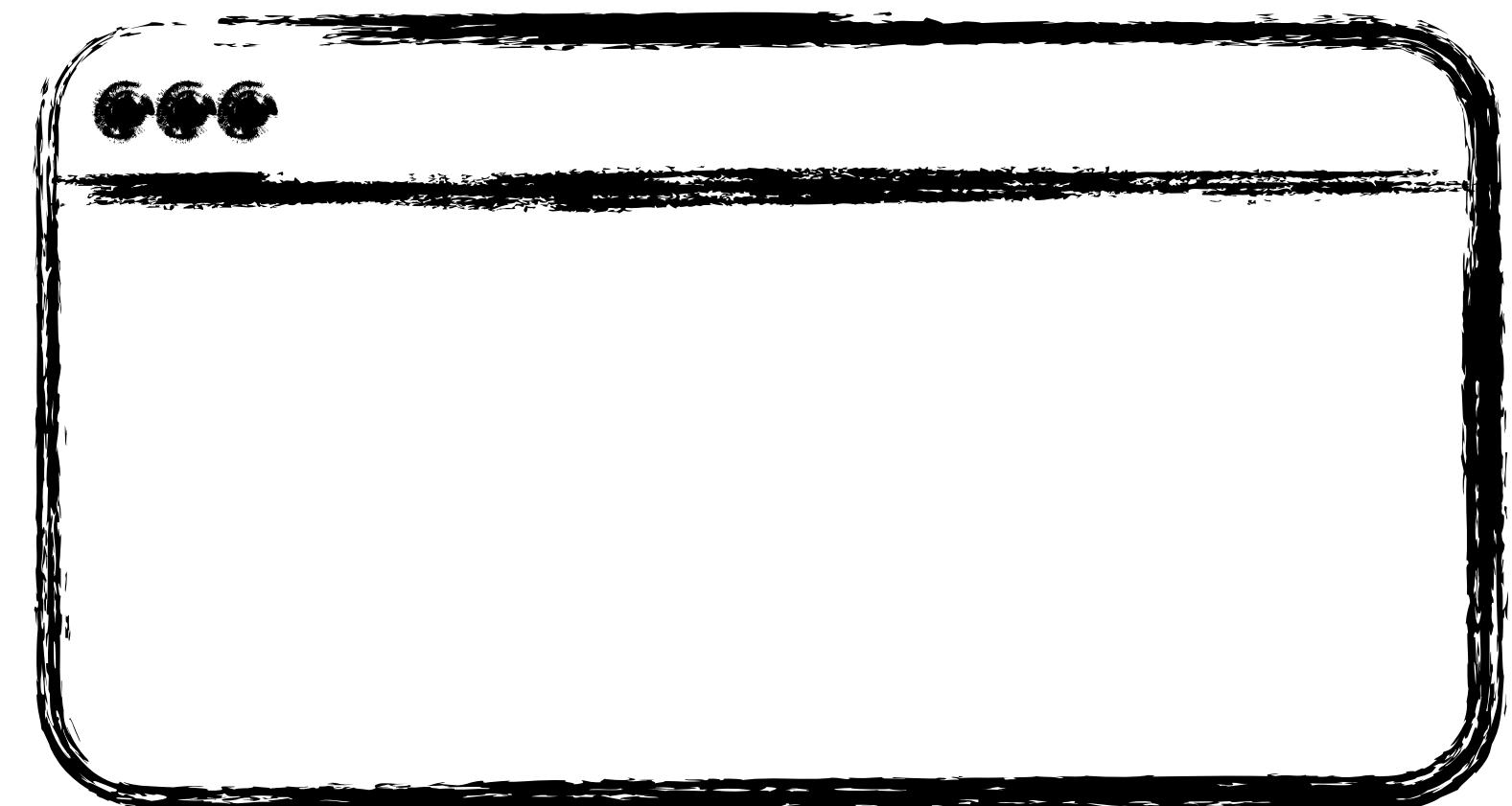
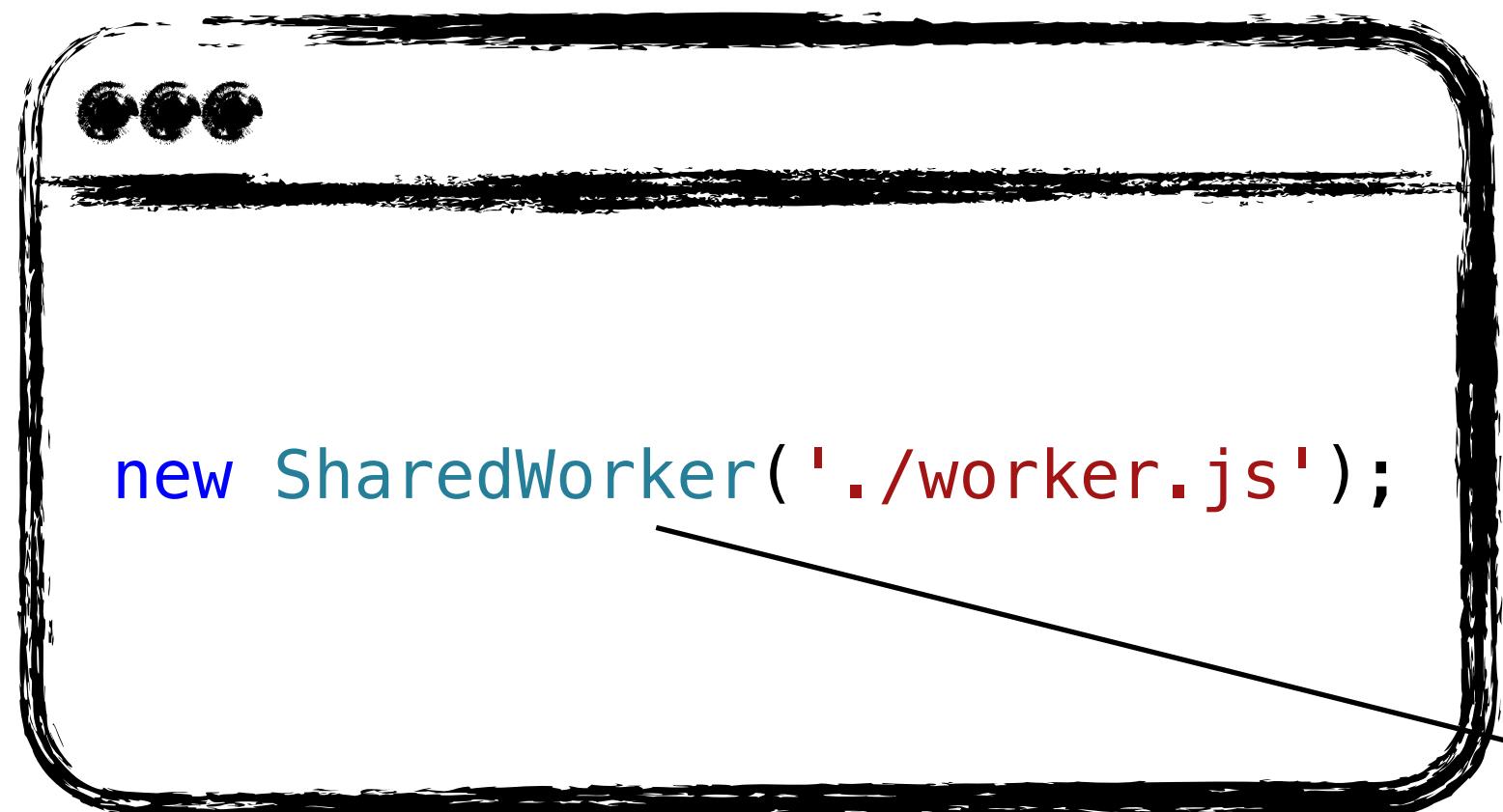
Shared Worker



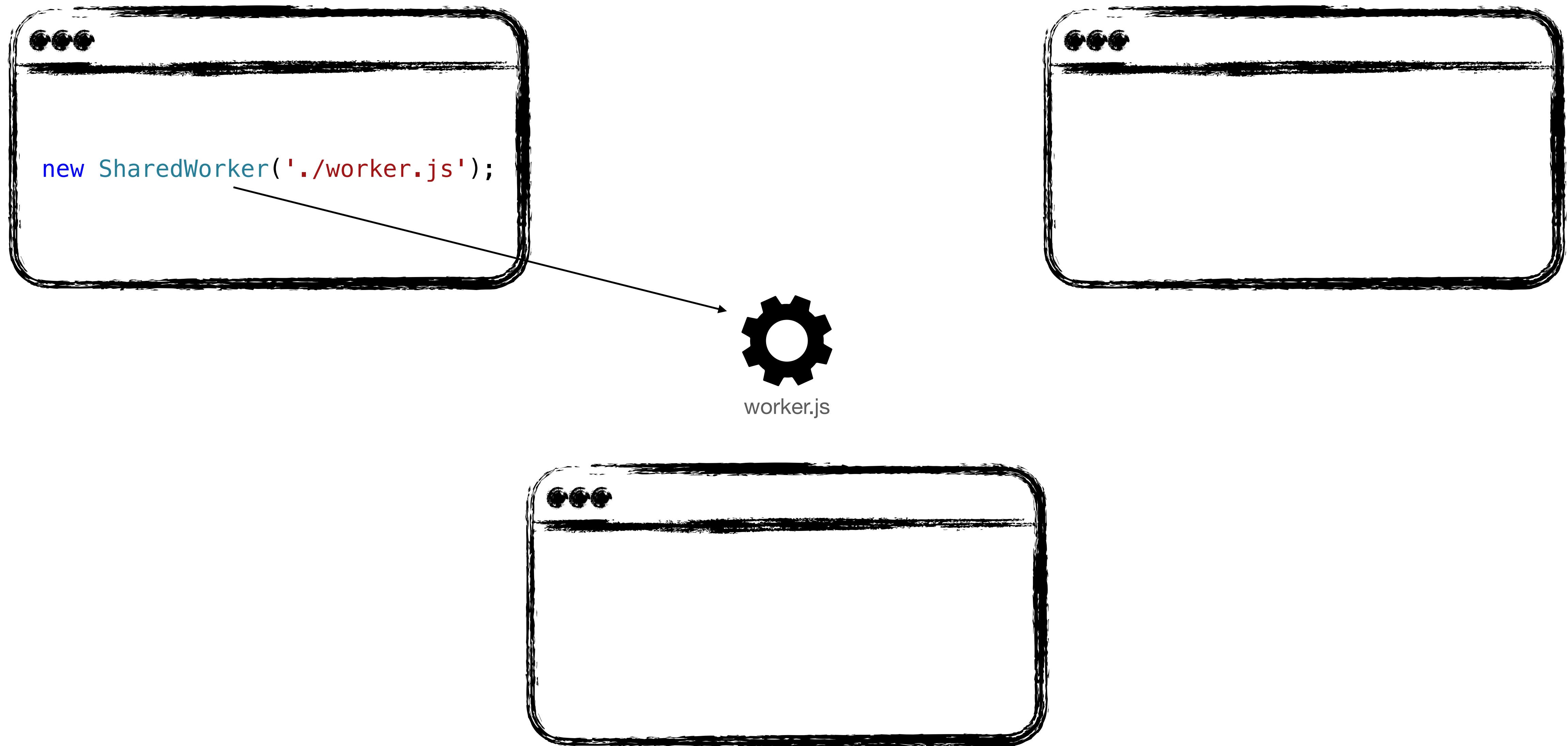
```
new SharedWorker('./worker.js');
```



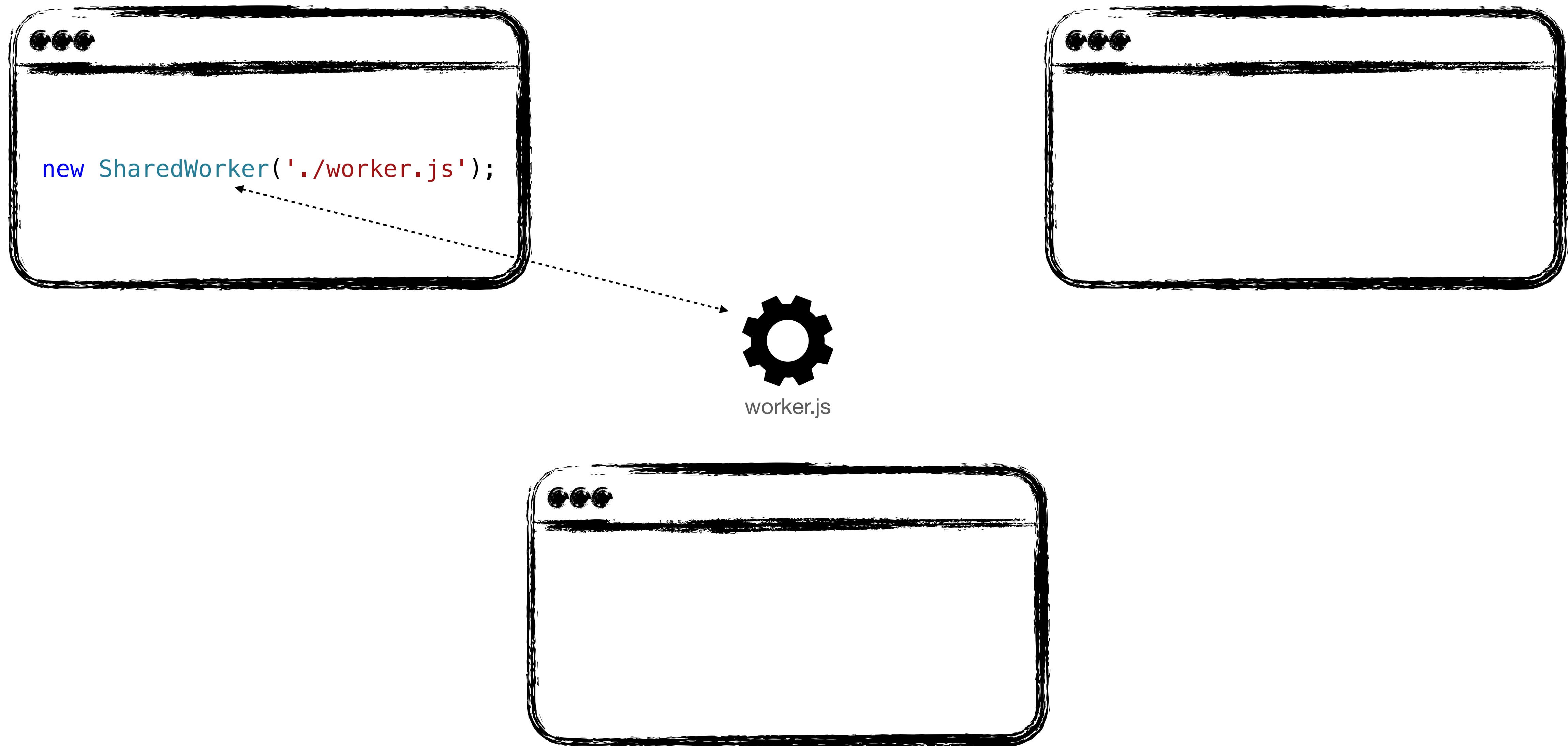
Shared Worker



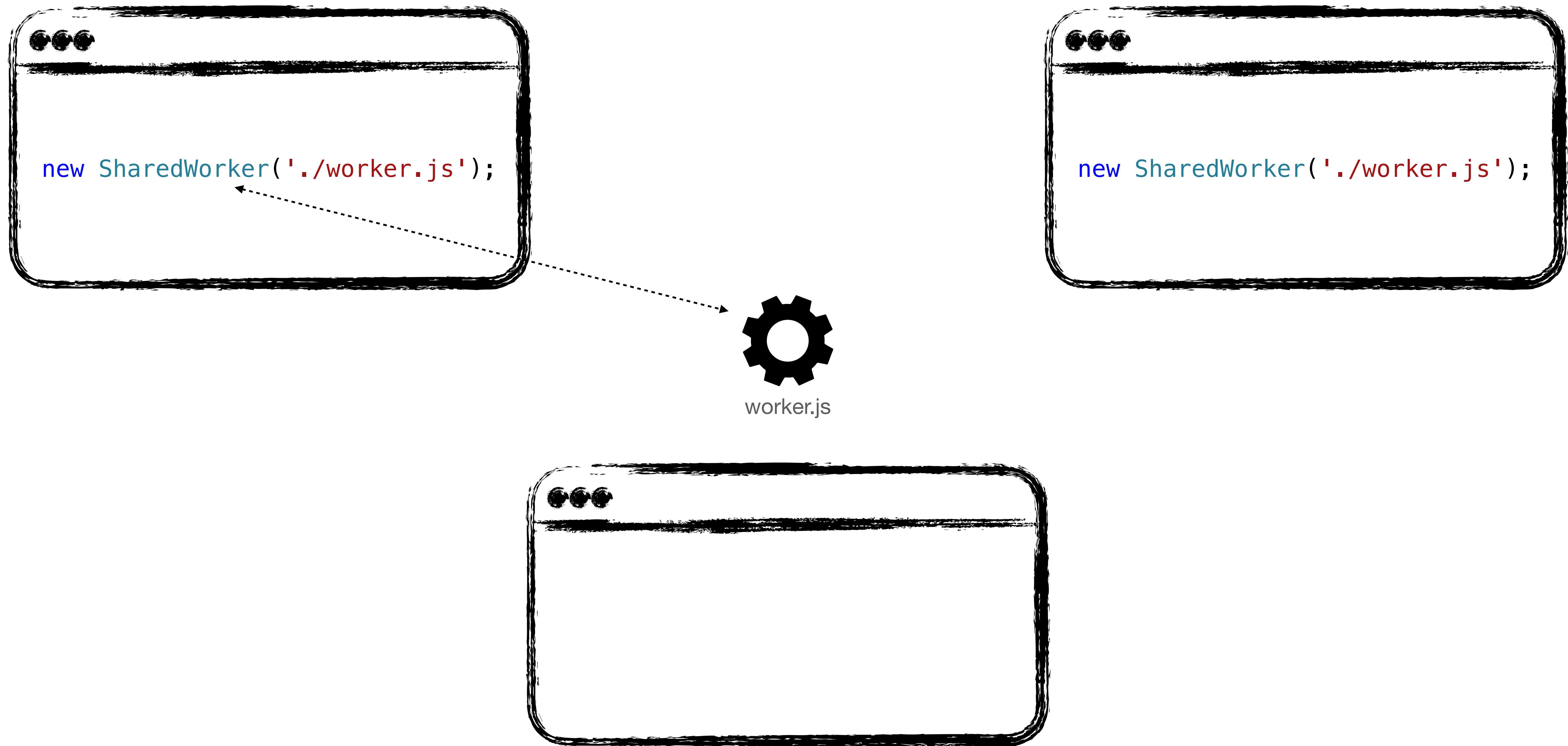
Shared Worker



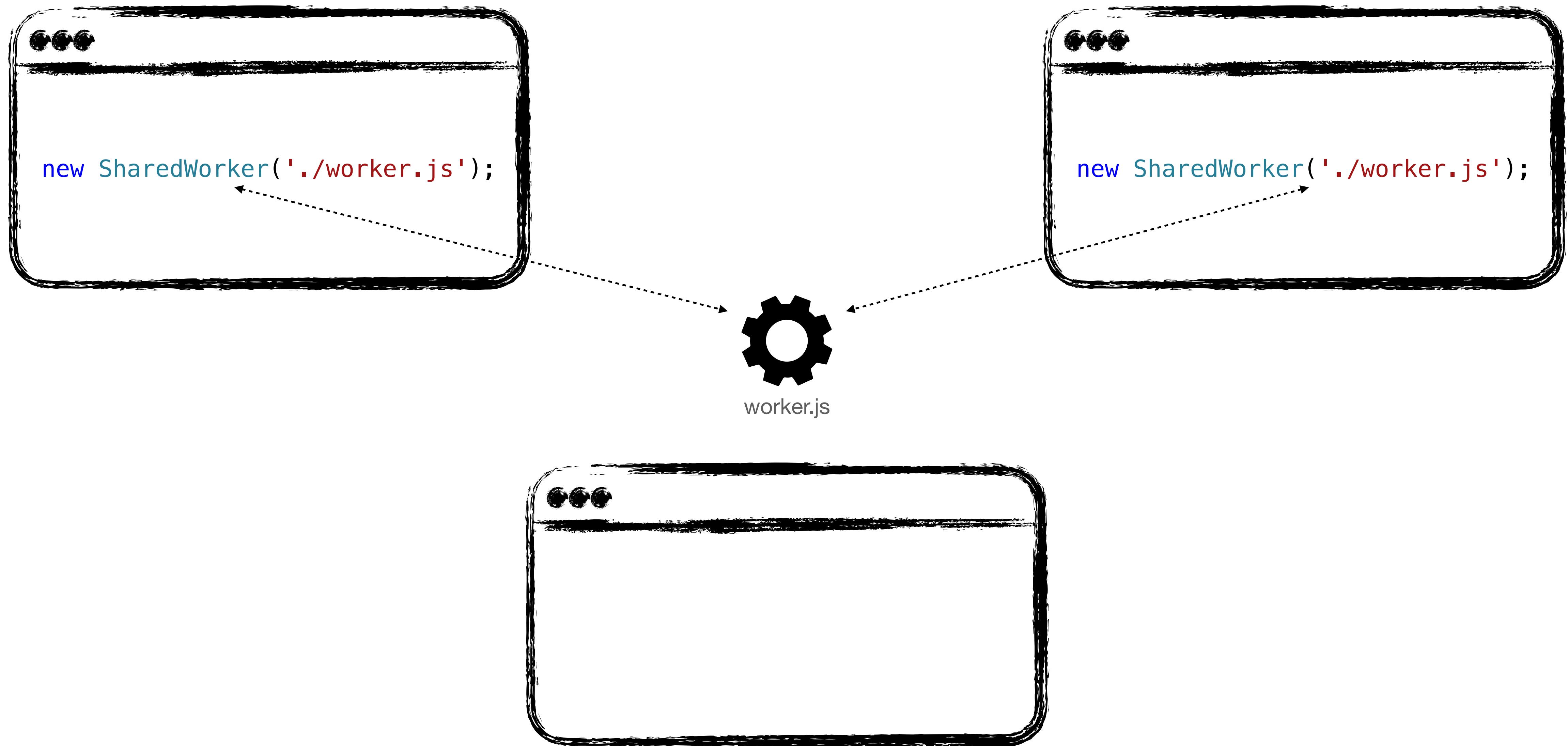
Shared Worker



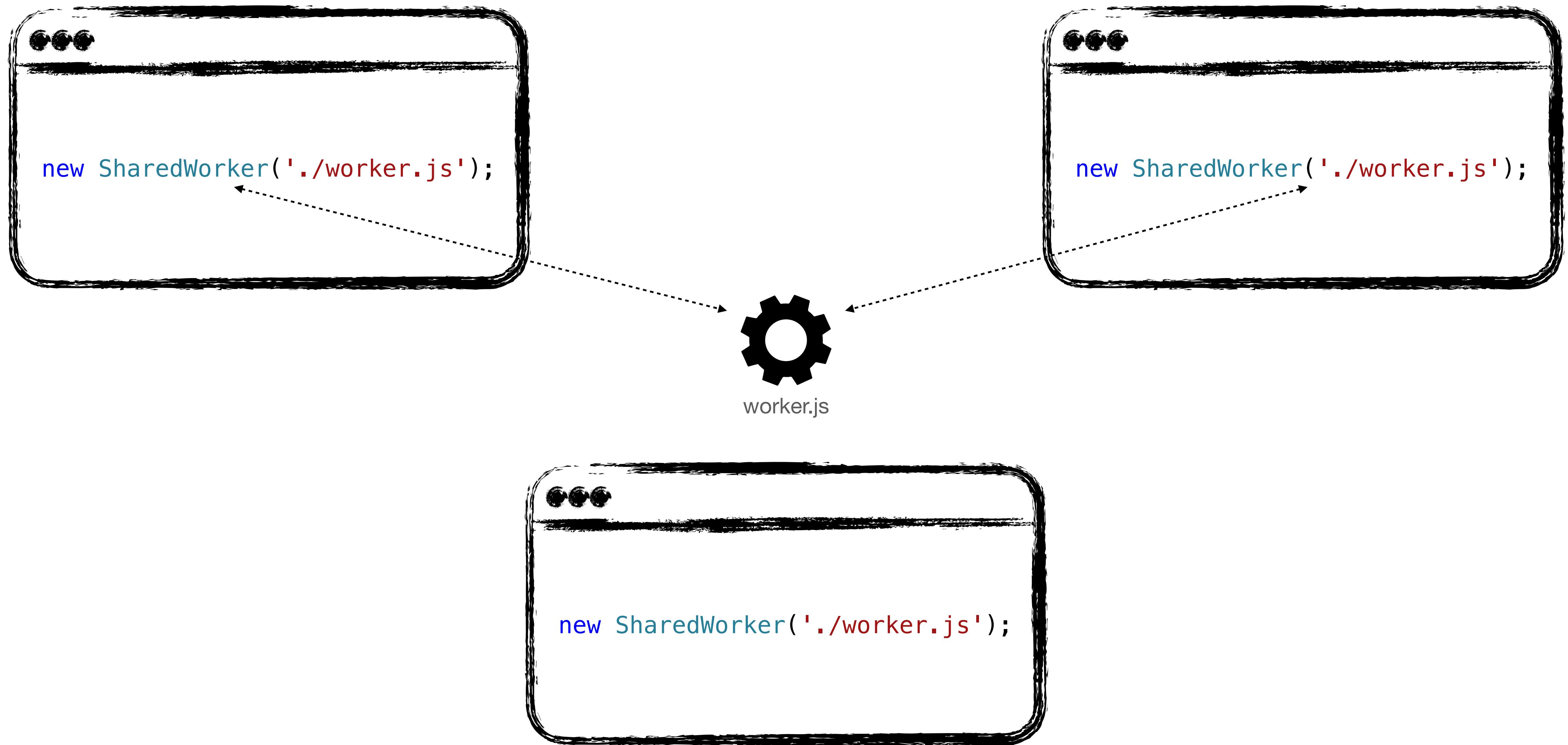
Shared Worker



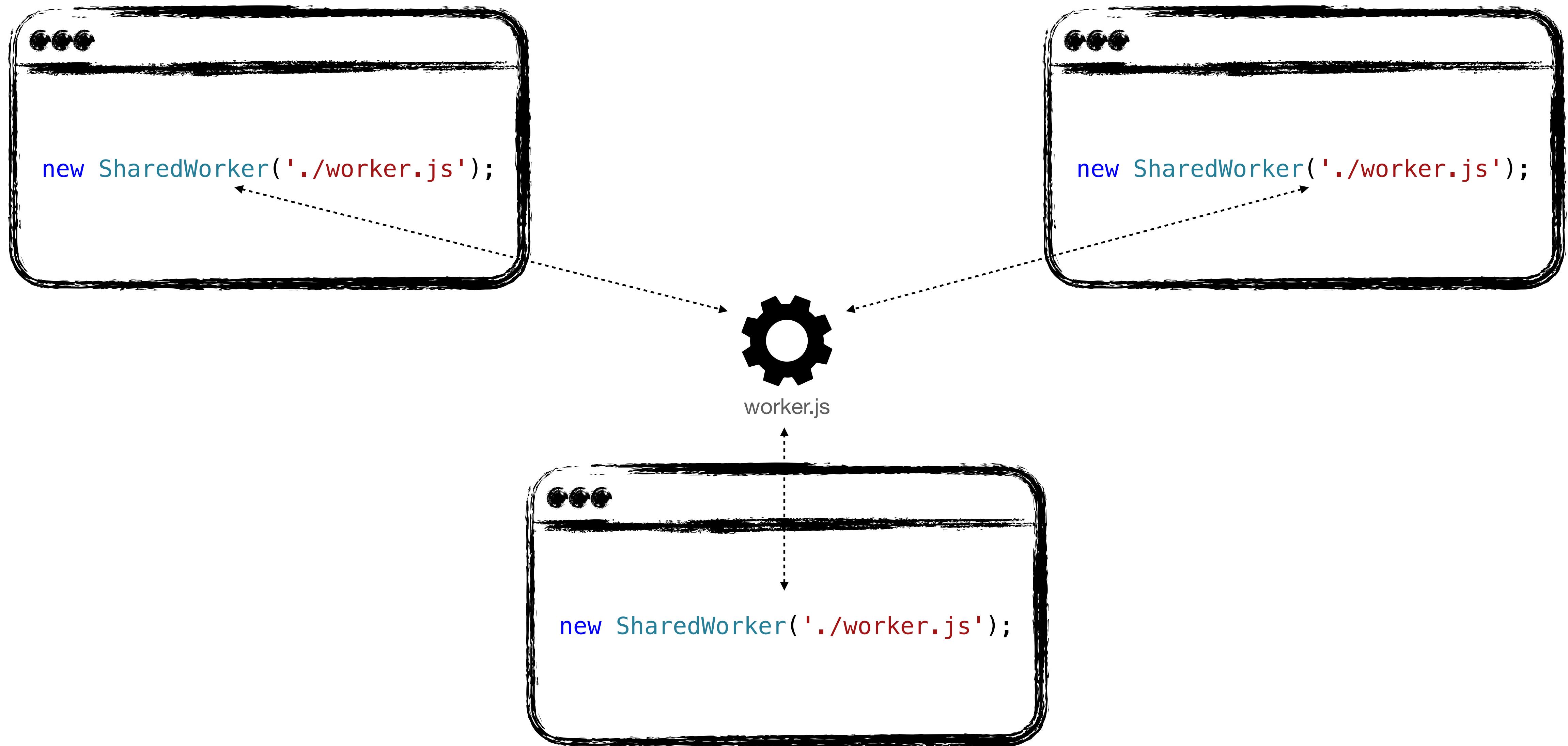
Shared Worker



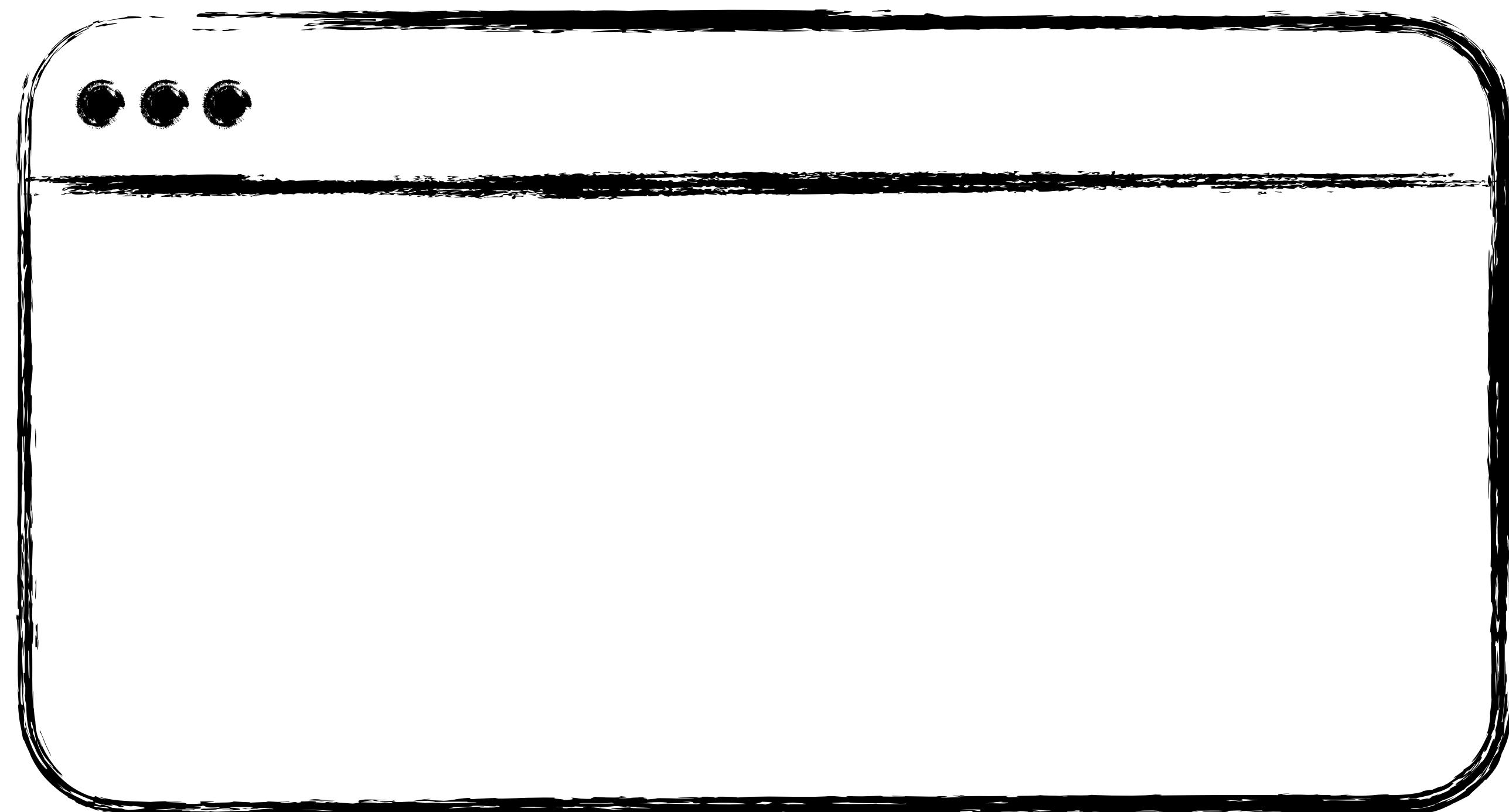
Shared Worker



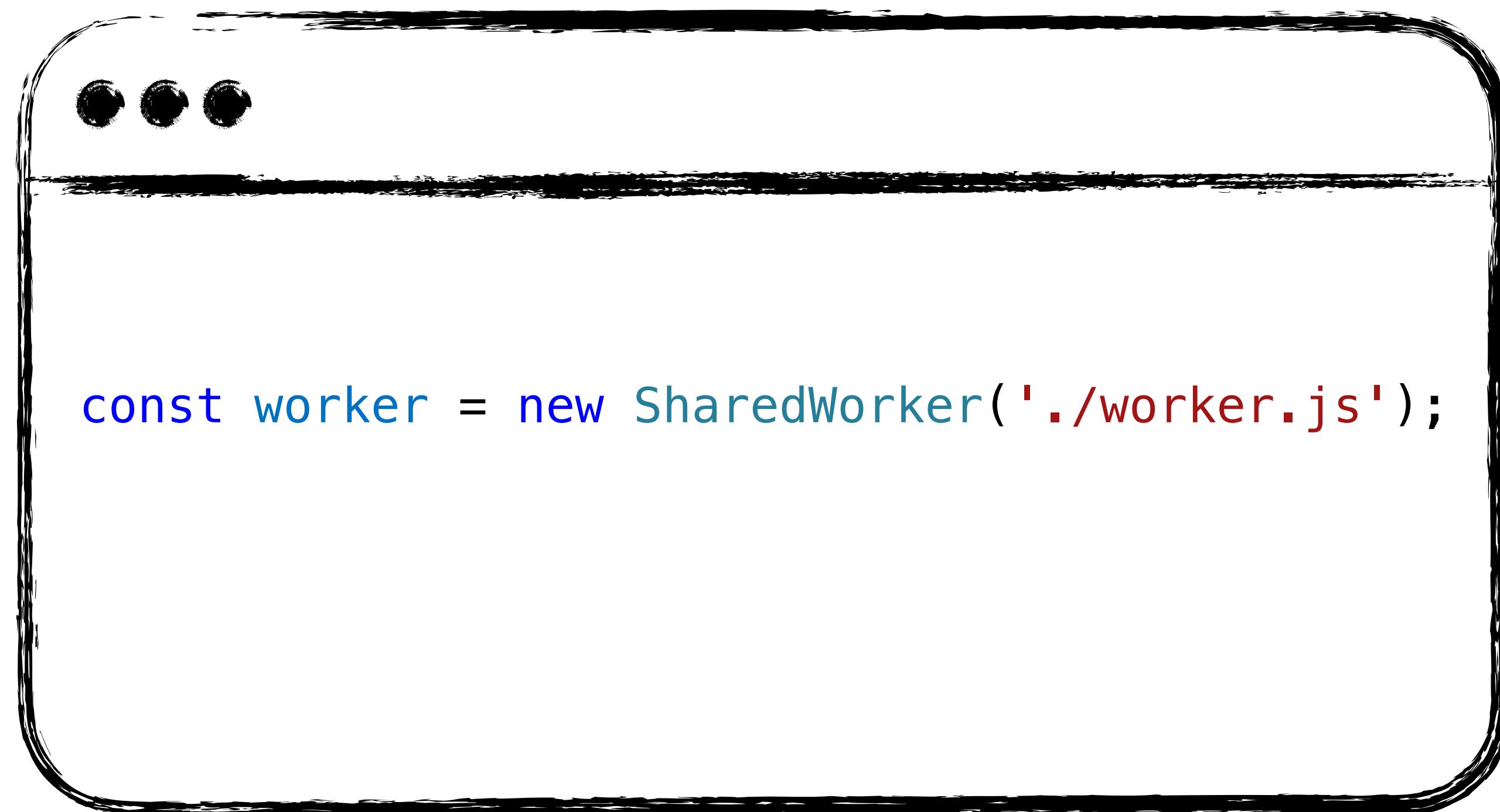
Shared Worker



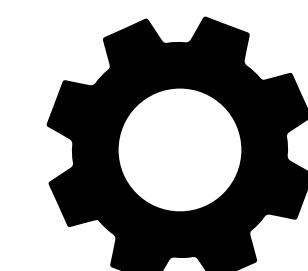
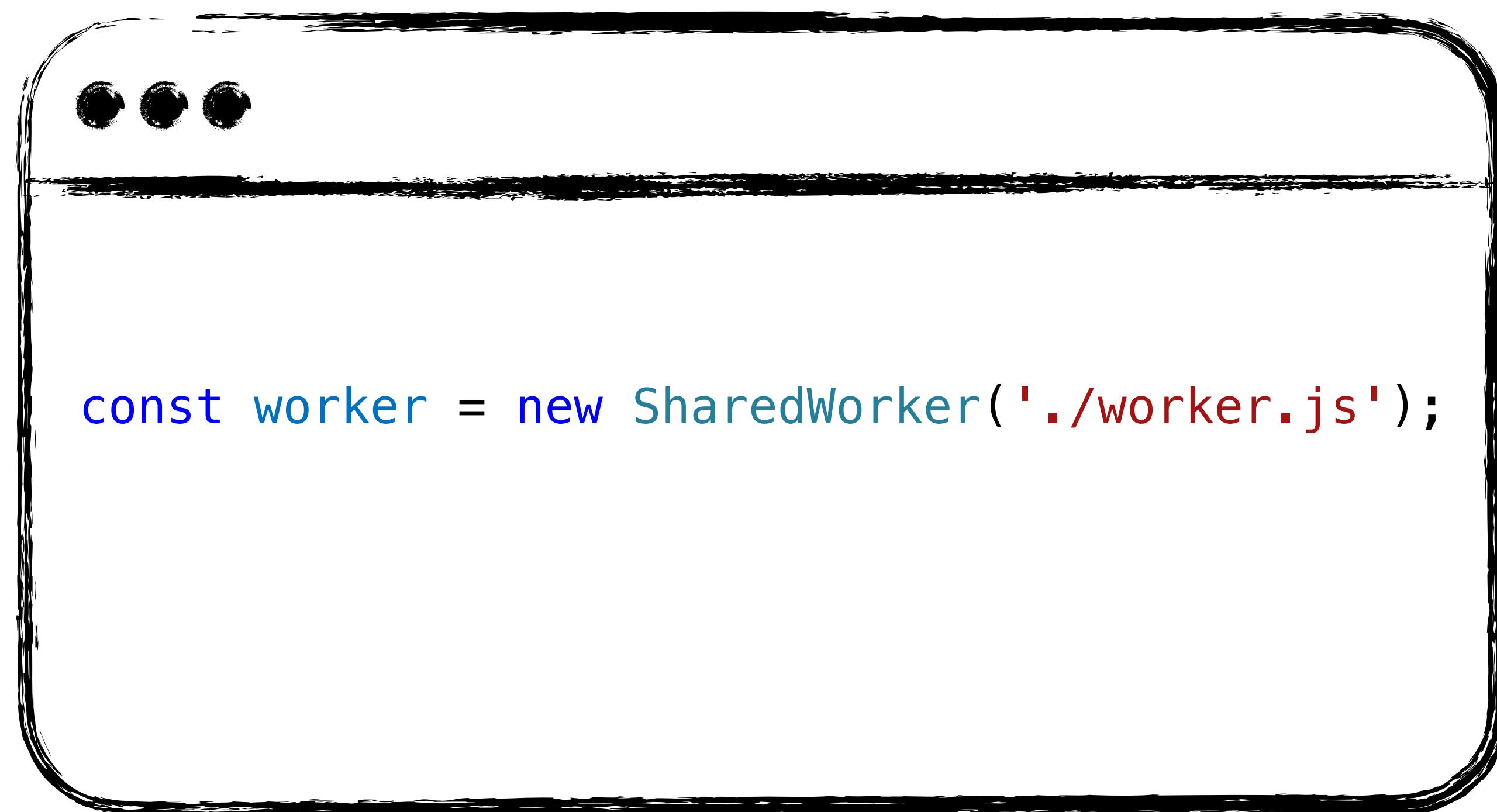
Shared Workers



Shared Workers

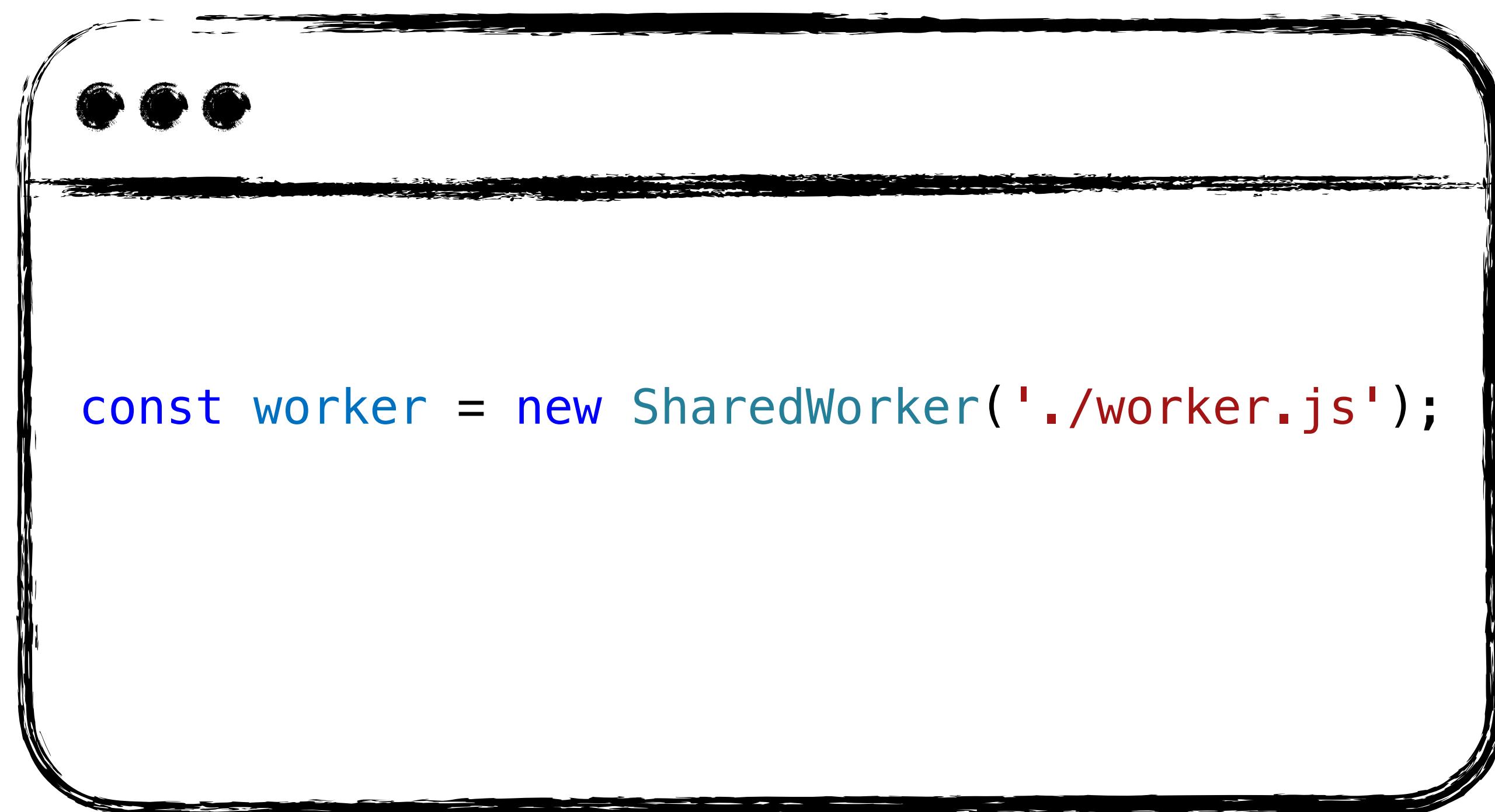


Shared Workers

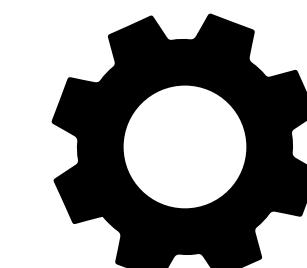


worker.js

Shared Workers

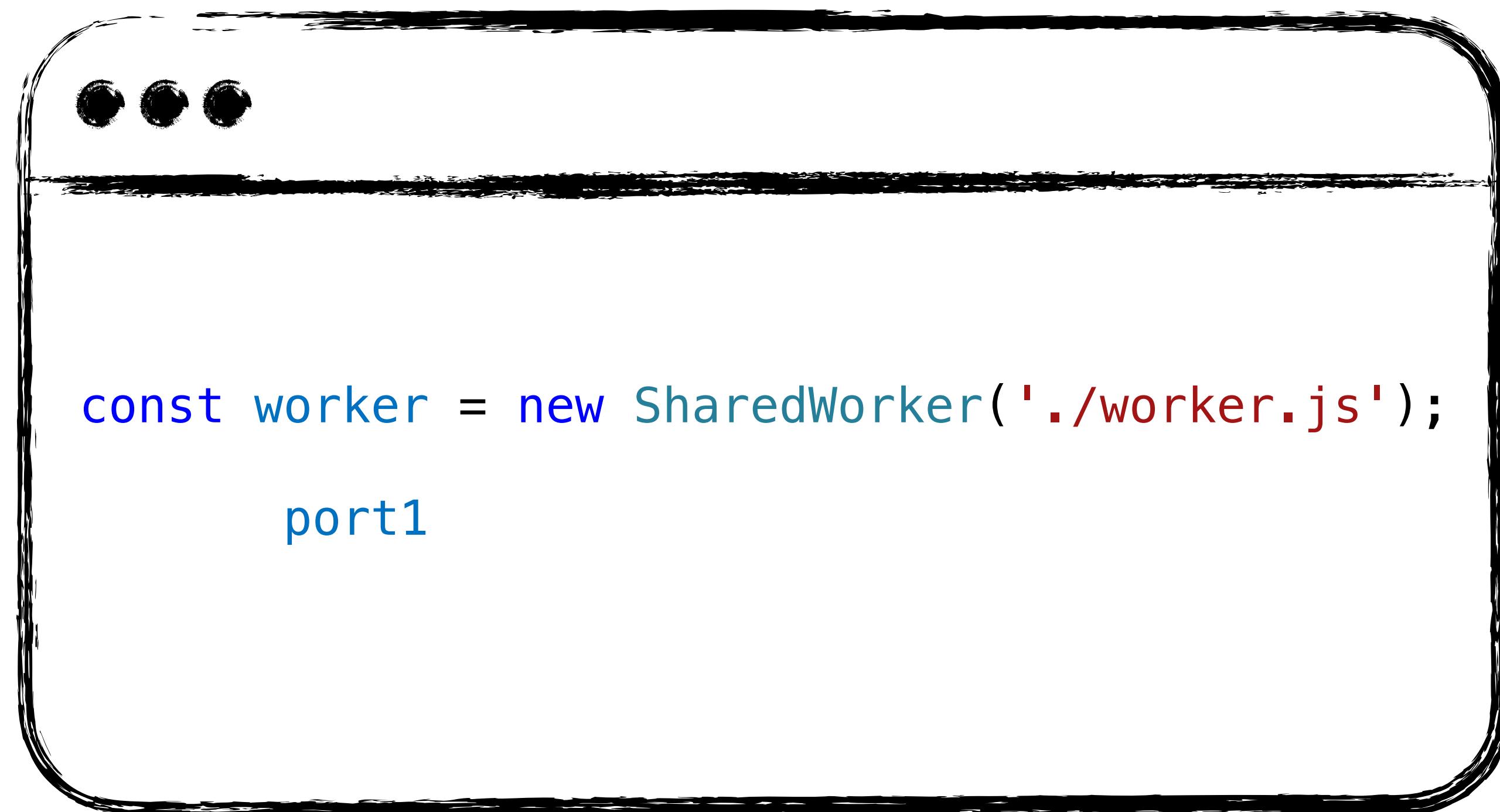


```
const {port1, port2} = new MessageChannel();
```

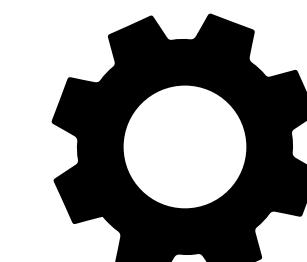


worker.js

Shared Workers

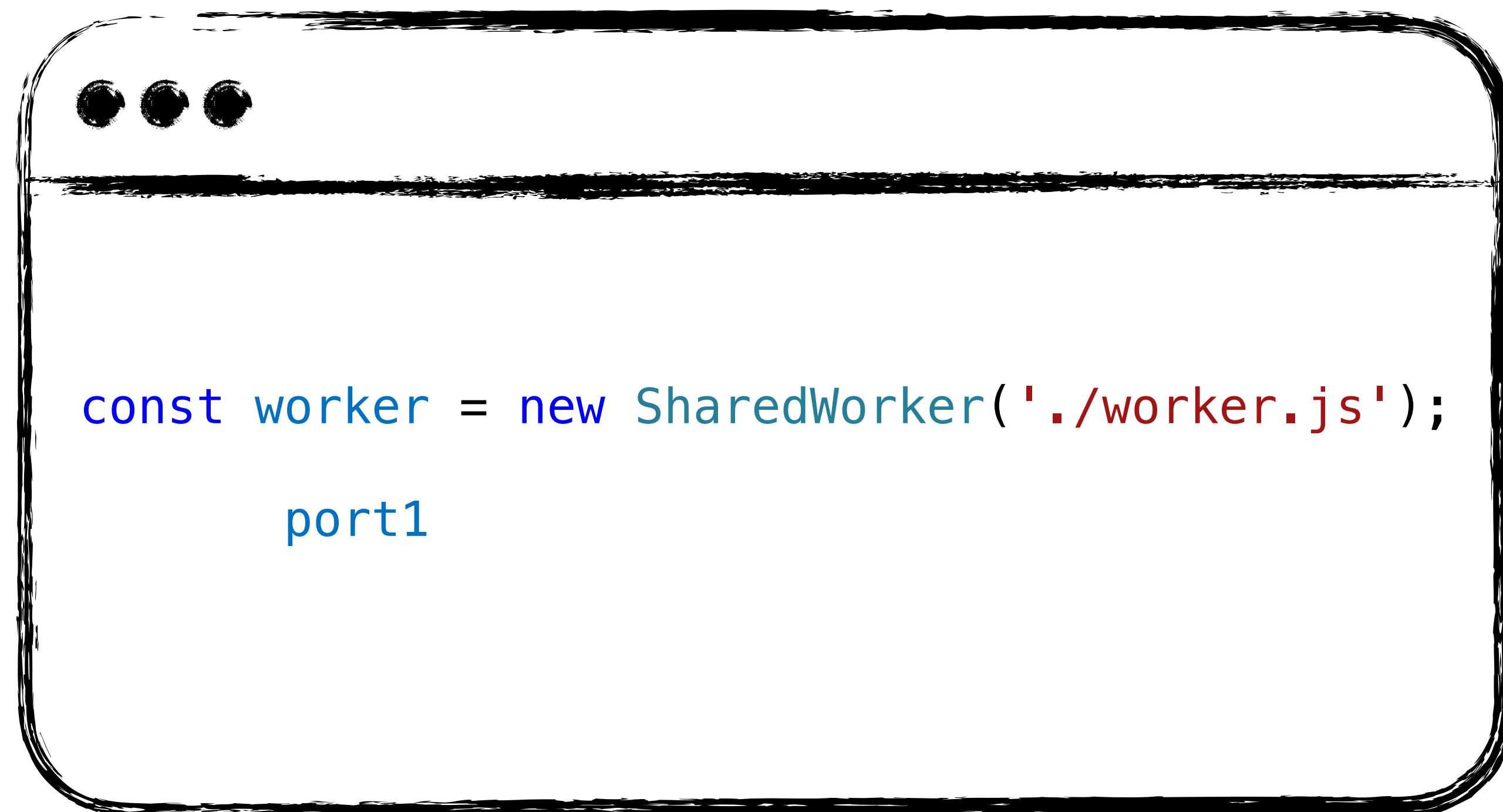


```
const { , port2} = new MessageChannel();
```



worker.js

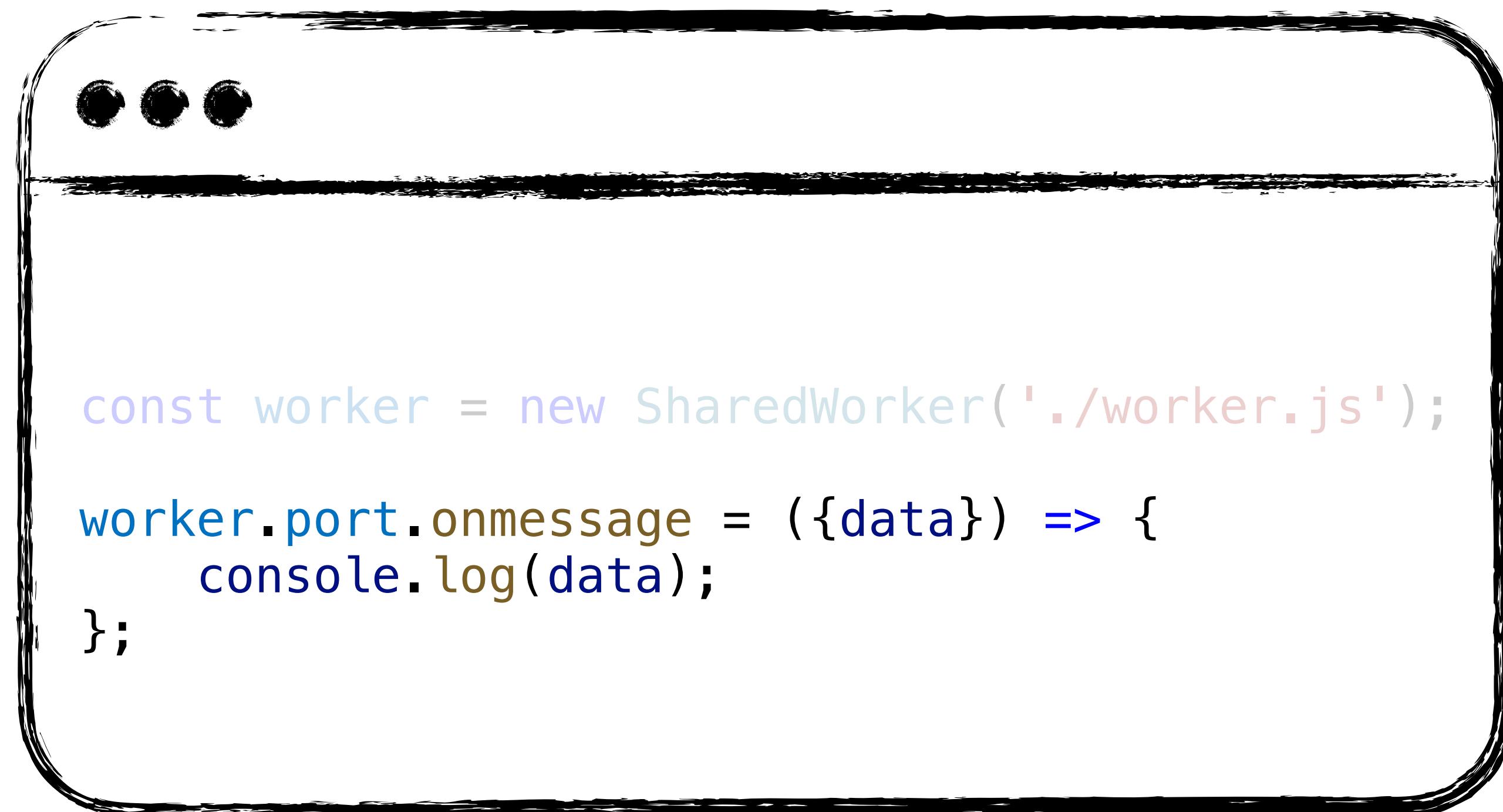
Shared Workers



```
const { , } = new MessageChannel();
```



Shared Workers



port2
worker.js

Let's look at the worker side!

Shared Worker

worker.js

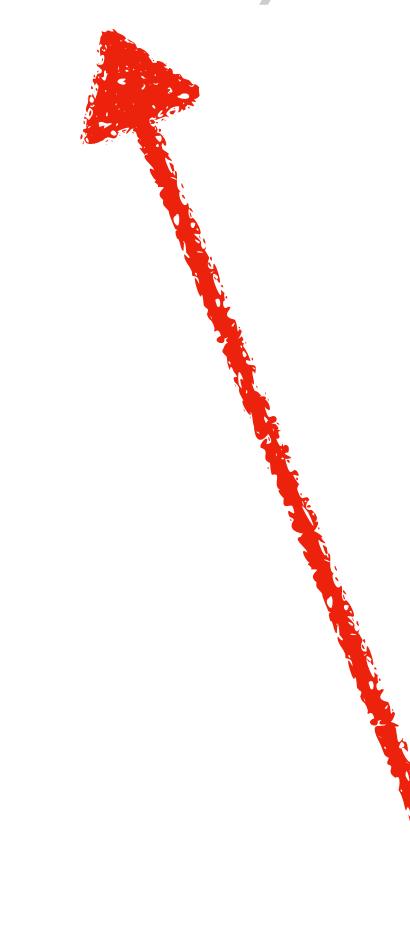
Shared Worker

```
self.onconnect = (event) => {  
};
```

worker.js

Shared Worker

```
self.onconnect = (event) => {  
};  
  
MessageEvent
```



worker.js

Shared Worker

```
self.onconnect = (event) => {  
    const port = event.ports[0];  
  
};
```

worker.js

Shared Worker

```
port2 of the MessageChannel  
self.onconnect = (event) => {  
    const port = event.ports[0];  
};
```

worker.js

Shared Worker

```
self.onconnect = (event) => {
  const port = event.ports[0];

  port.onmessage = ({data}) => {
    console.log(data);
  };
};
```

worker.js

Shared Worker

```
self.onconnect = (event) => {
  const port = event.ports[0];

  port.onmessage = ({data}) => {
    console.log(data);
  };
};
```

worker.js

Shared Worker - Example

```
let connections = 0;

self.onconnect = (event) => {
  const port = event.ports[0];

  connections += 1;

  port.postMessage({connections});
};
```

worker.js

What if... We need a simple proxy?

BroadcastChannel

BroadcastChannel

BroadcastChannel

- Named channel

BroadcastChannel

- Named channel
- Same origin browsing context can subscribe

BroadcastChannel

- Named channel
- Same origin browsing context can subscribe
- Communication between windows, tabs, frames, iframes, workers

BroadcastChannel

- Named channel
- Same origin browsing context can subscribe
- Communication between windows, tabs, frames, iframes, workers
- Additional logic (e.g. filtering, mapping), use SharedWorker

Support

BroadcastChannel - LS

Global

77.69%

`BroadcastChannel` allows scripts from the same origin but other browsing contexts (windows, workers) to send each other messages.

Creating a BroadcastChannel

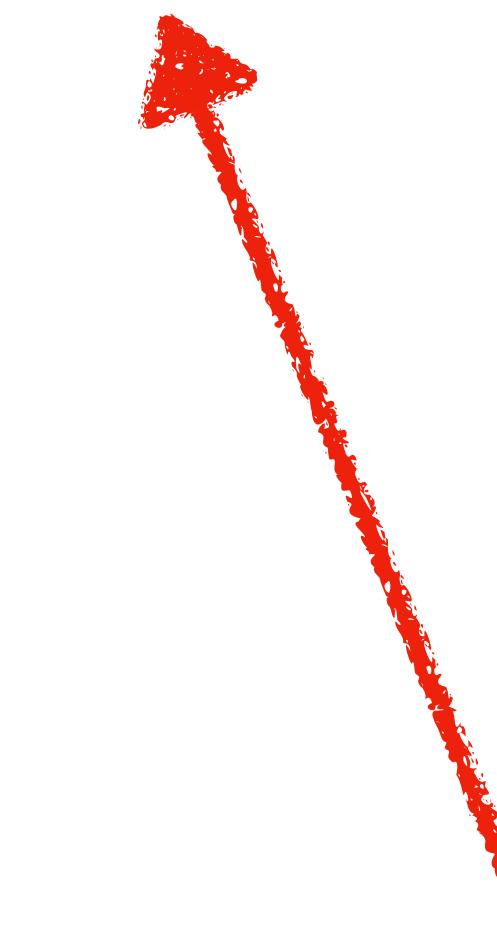
```
new BroadcastChannel(name: string): BroadcastChannel;
```

Communication

```
channel.postMessage(message: any): void;
```

Communication

```
channel.postMessage(message: any): void;
```



No transferable

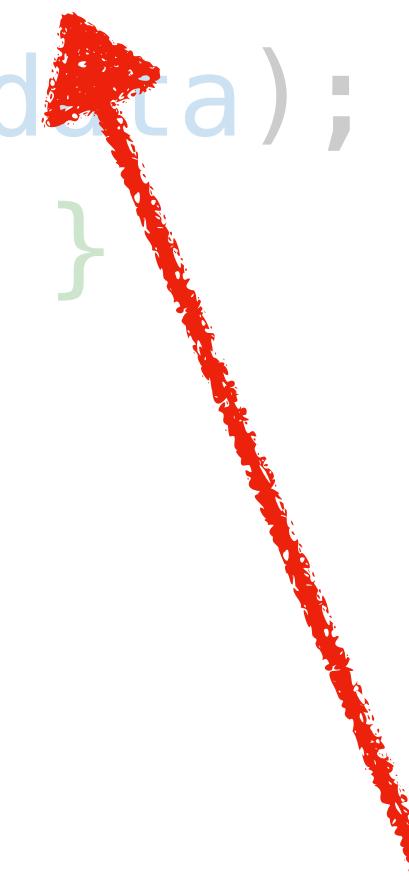
Receiving data

Receiving data

```
channel.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

Receiving data

```
channel.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

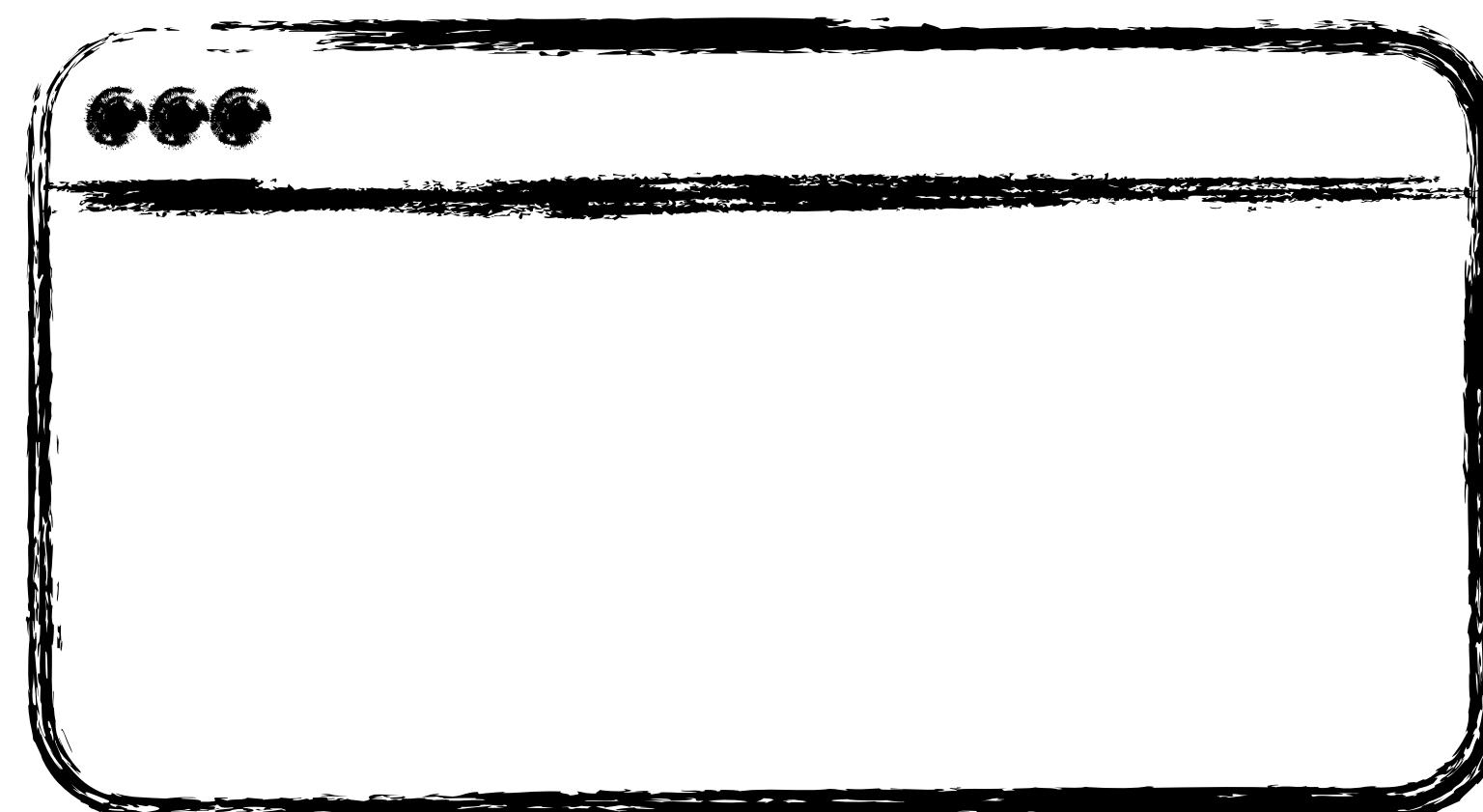
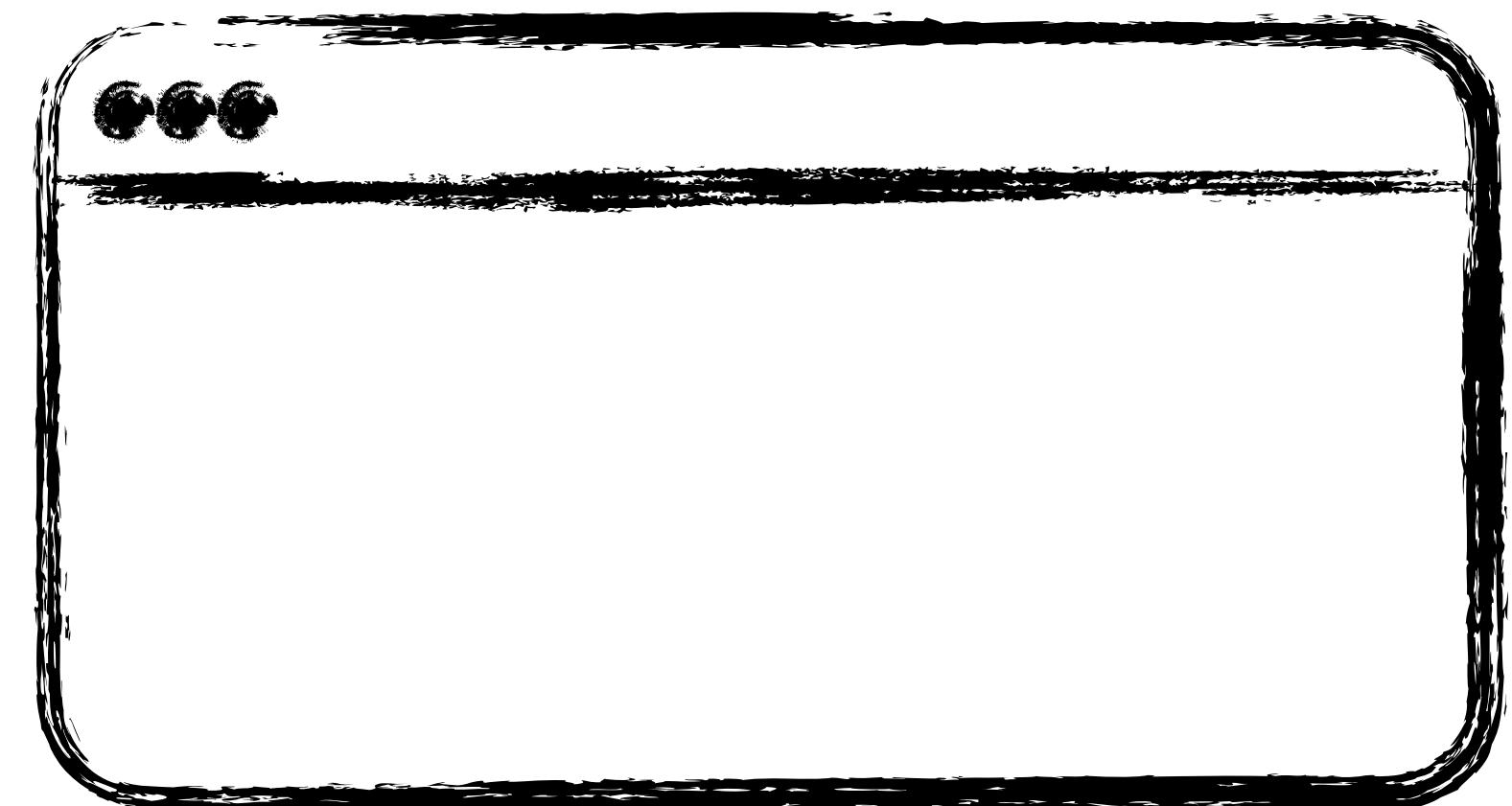
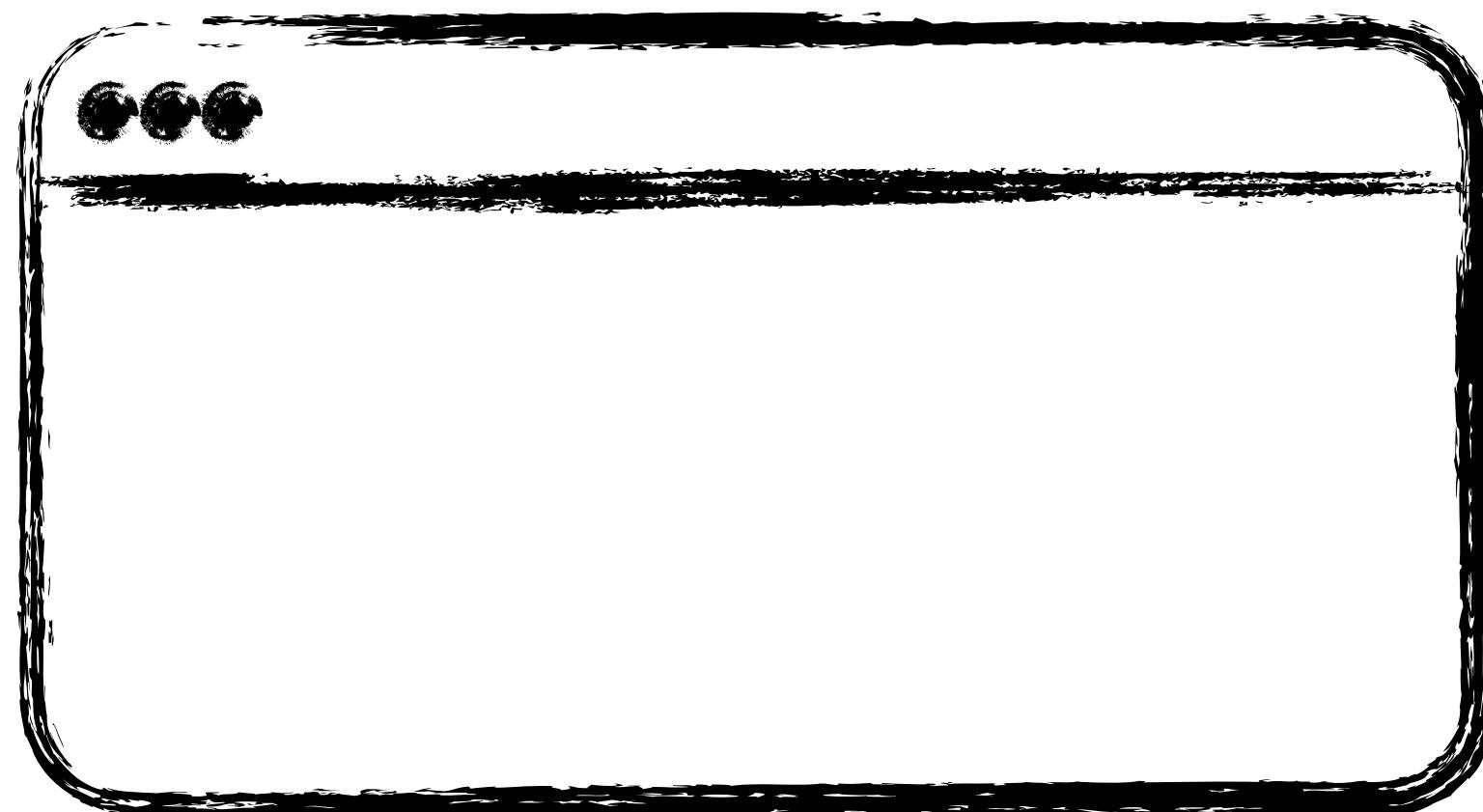


MessageEvent

Receiving data

```
channel.onmessage = (event) => {
  console.log(event.data);
  //=> { name: 'Sam' }
};
```

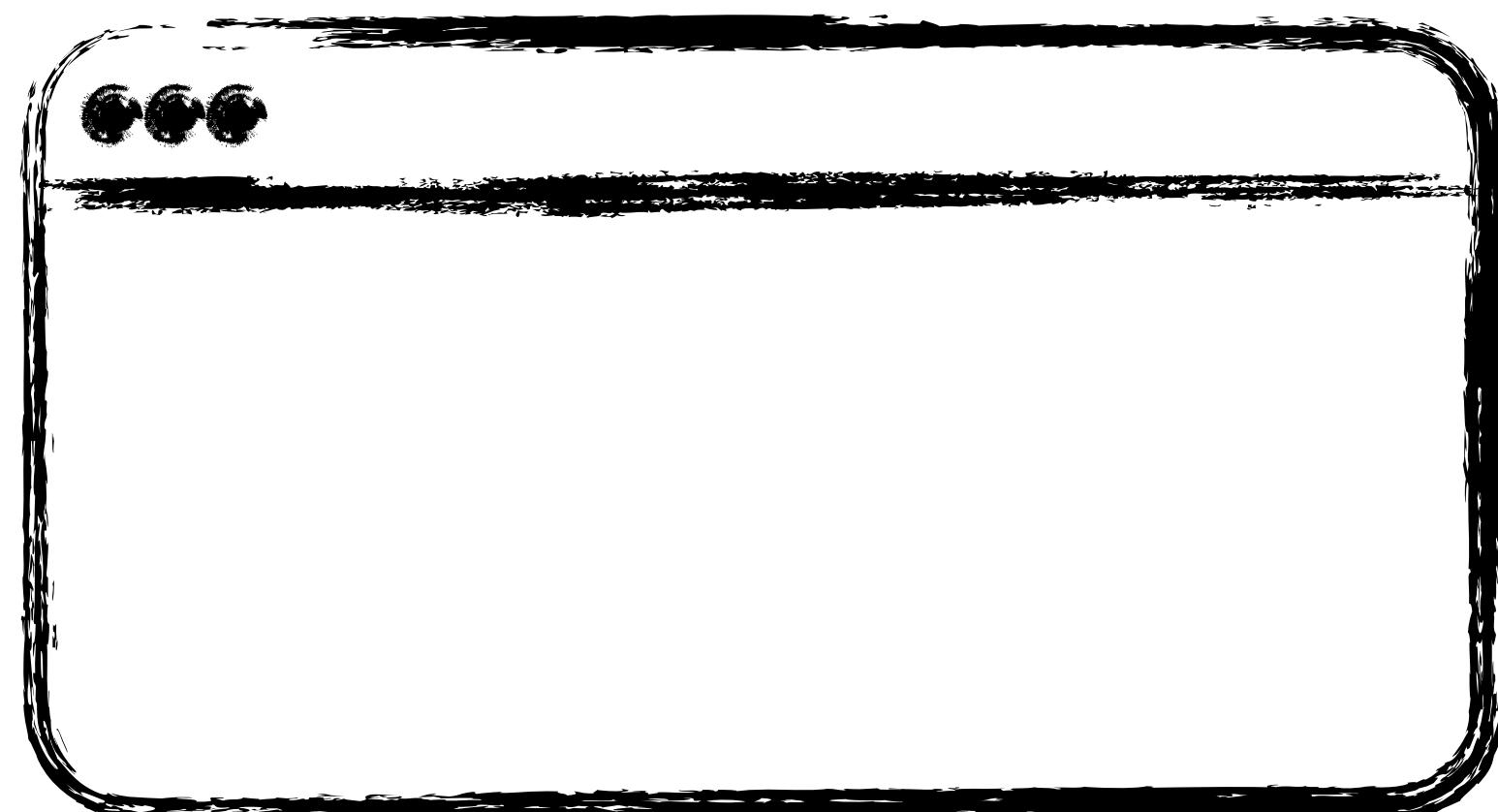
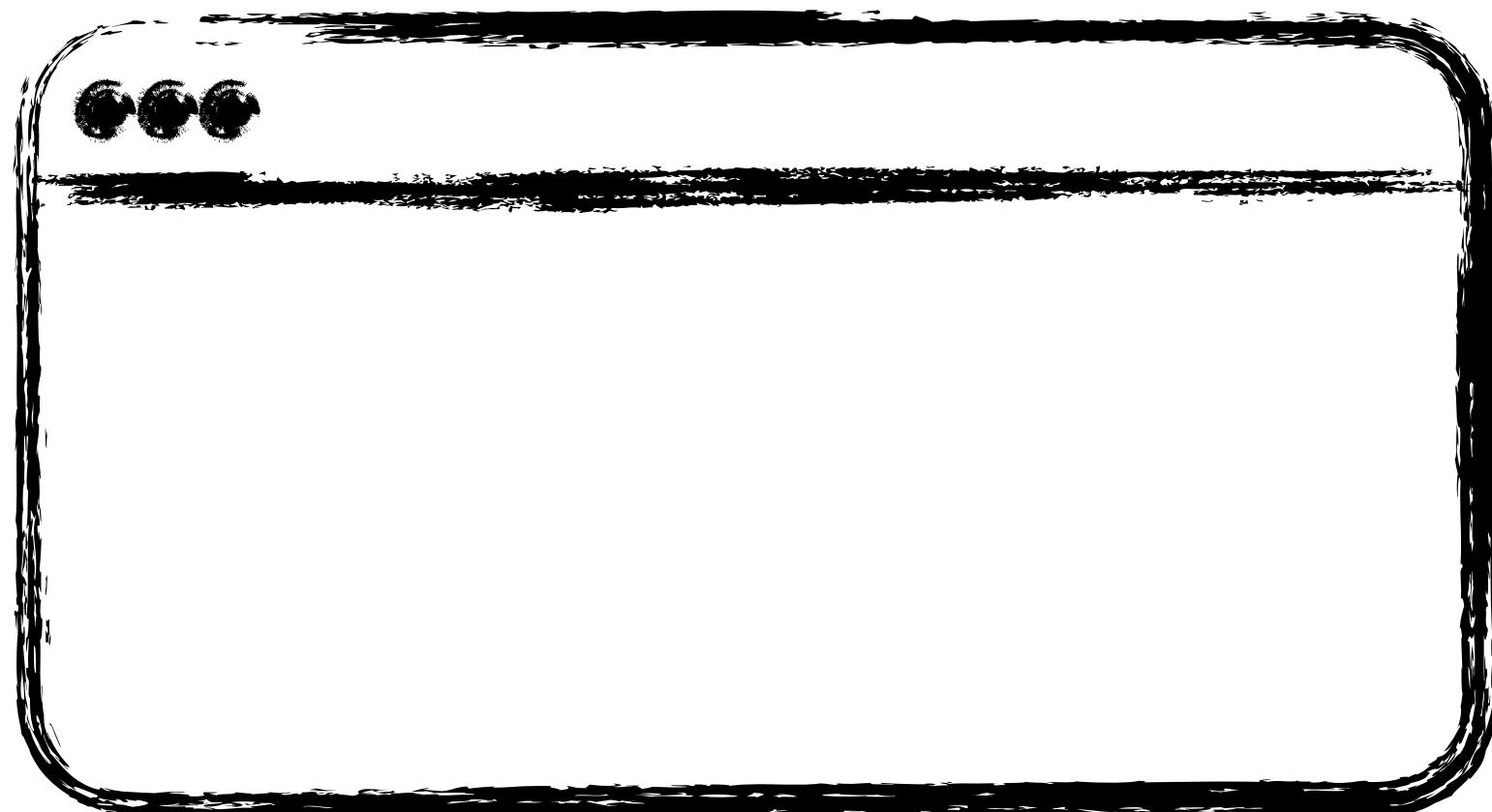
BroadcastChannel



BroadcastChannel



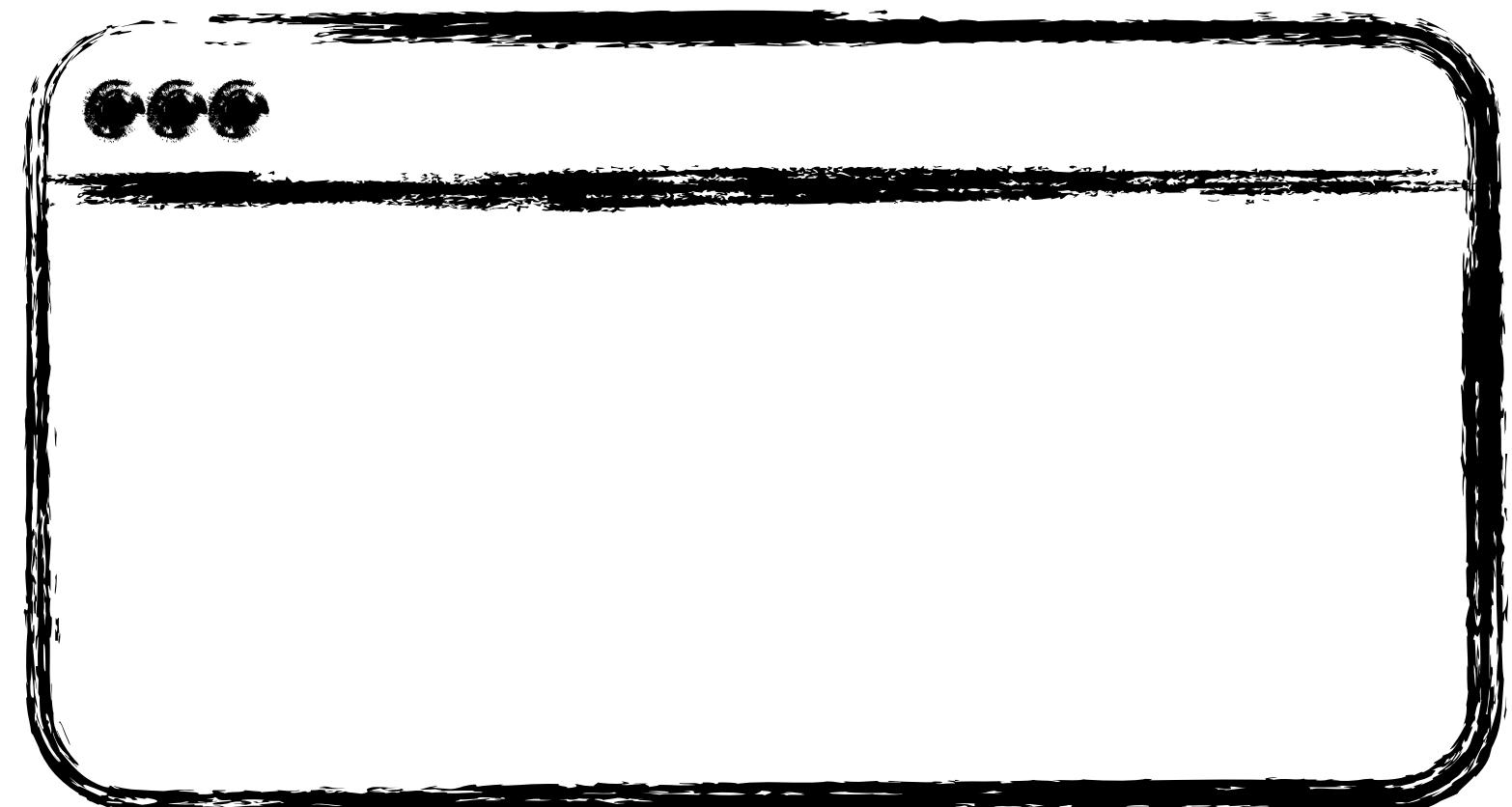
```
new BroadcastChannel('unicorns');
```



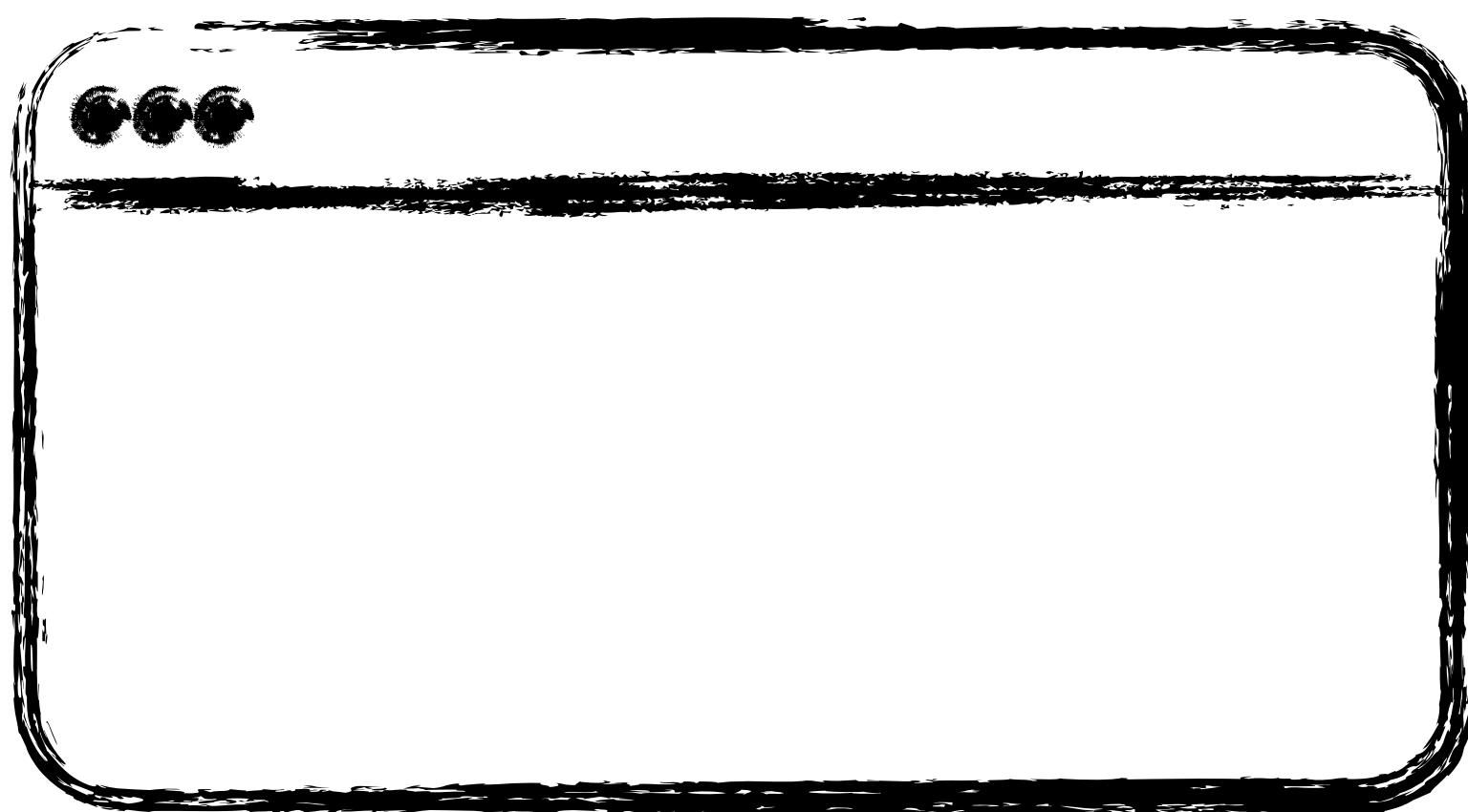
BroadcastChannel



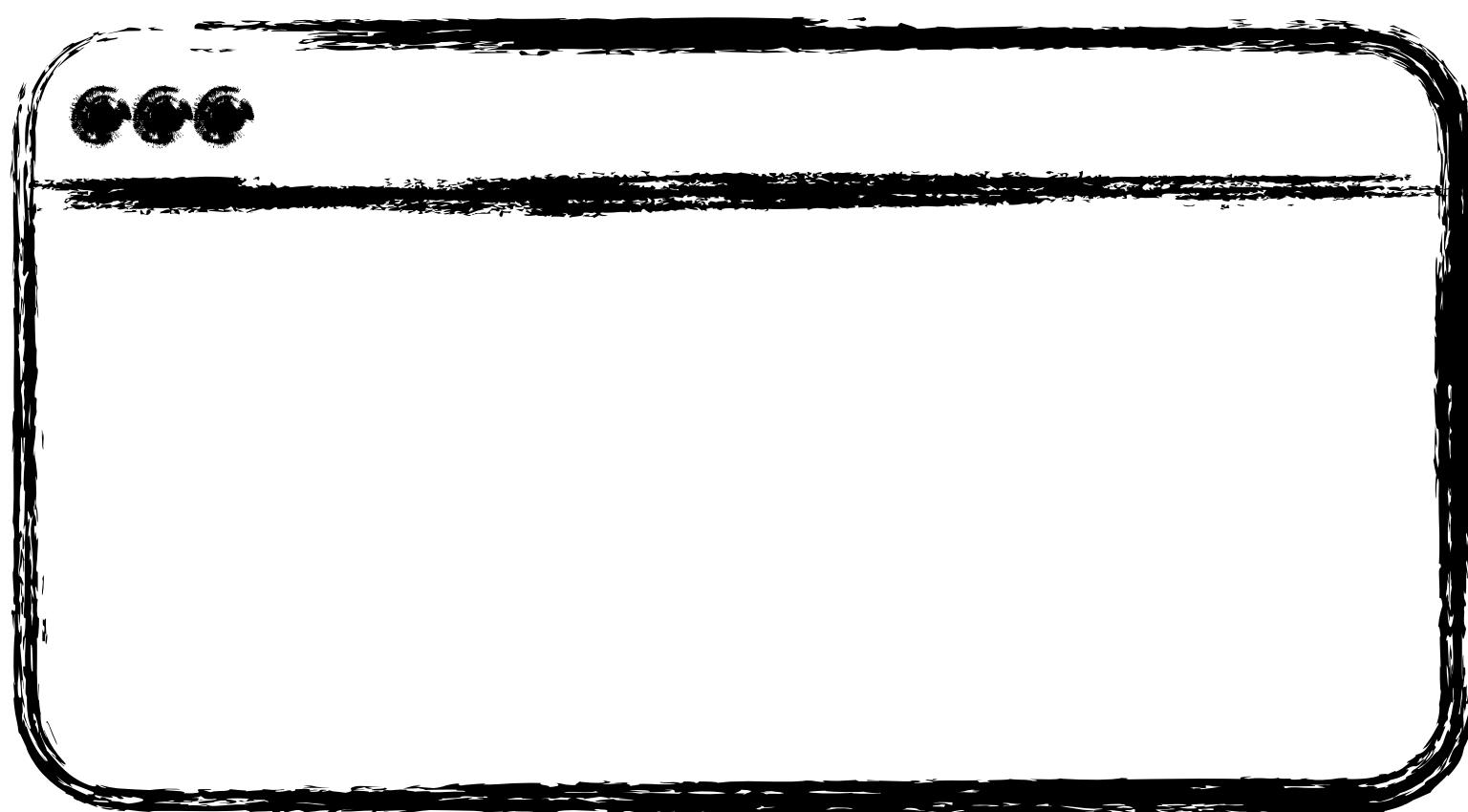
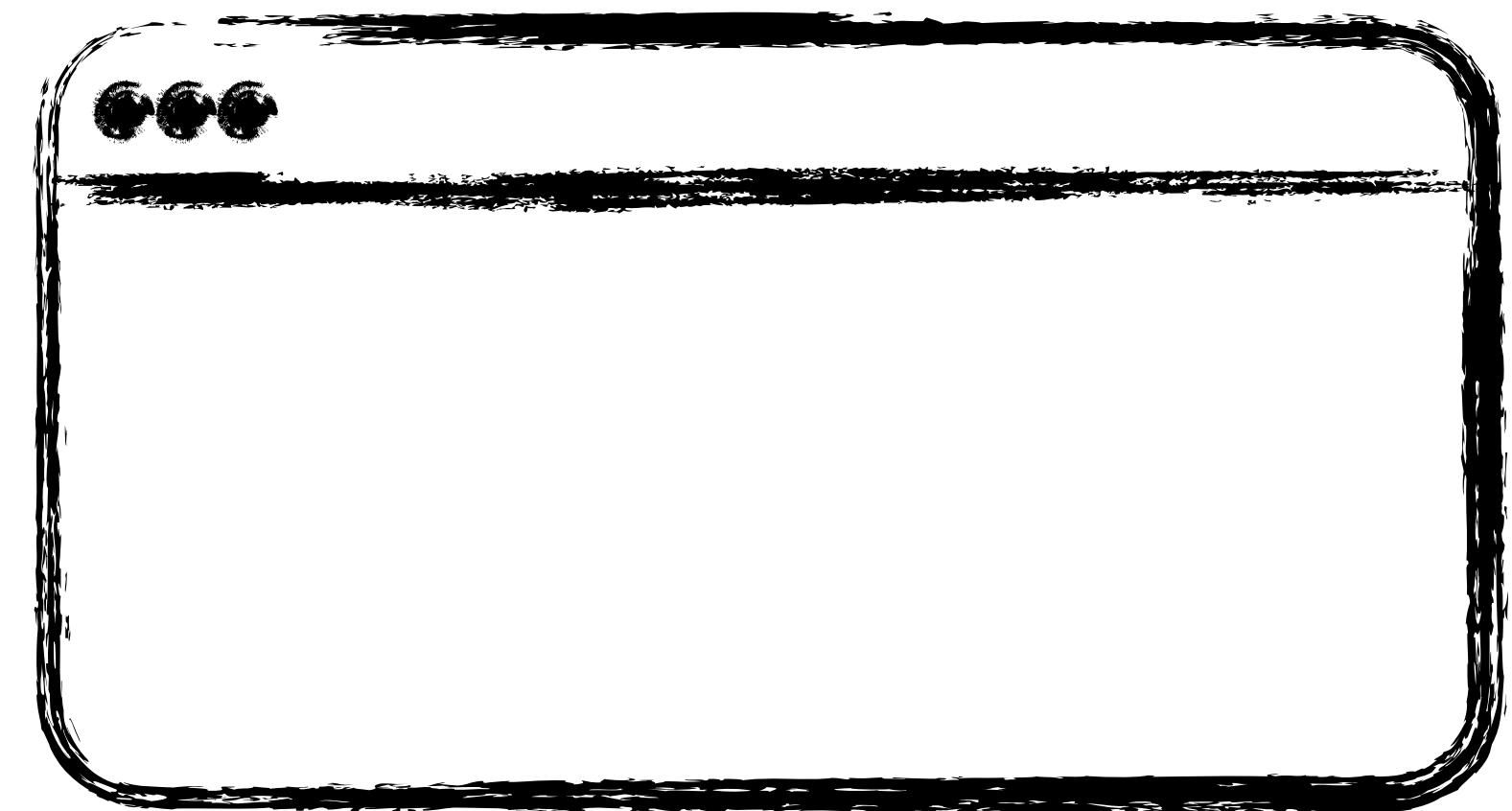
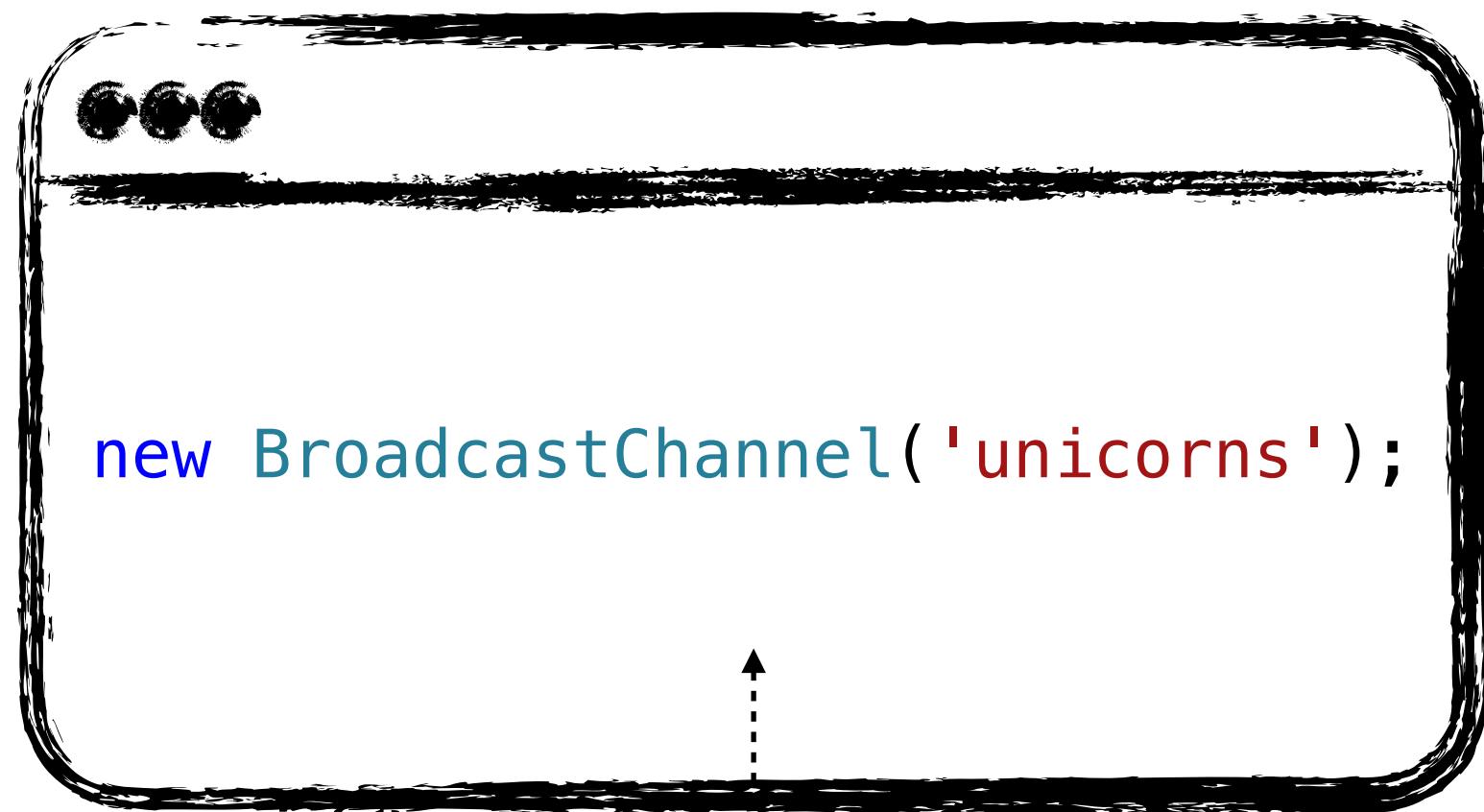
```
new BroadcastChannel('unicorns');
```



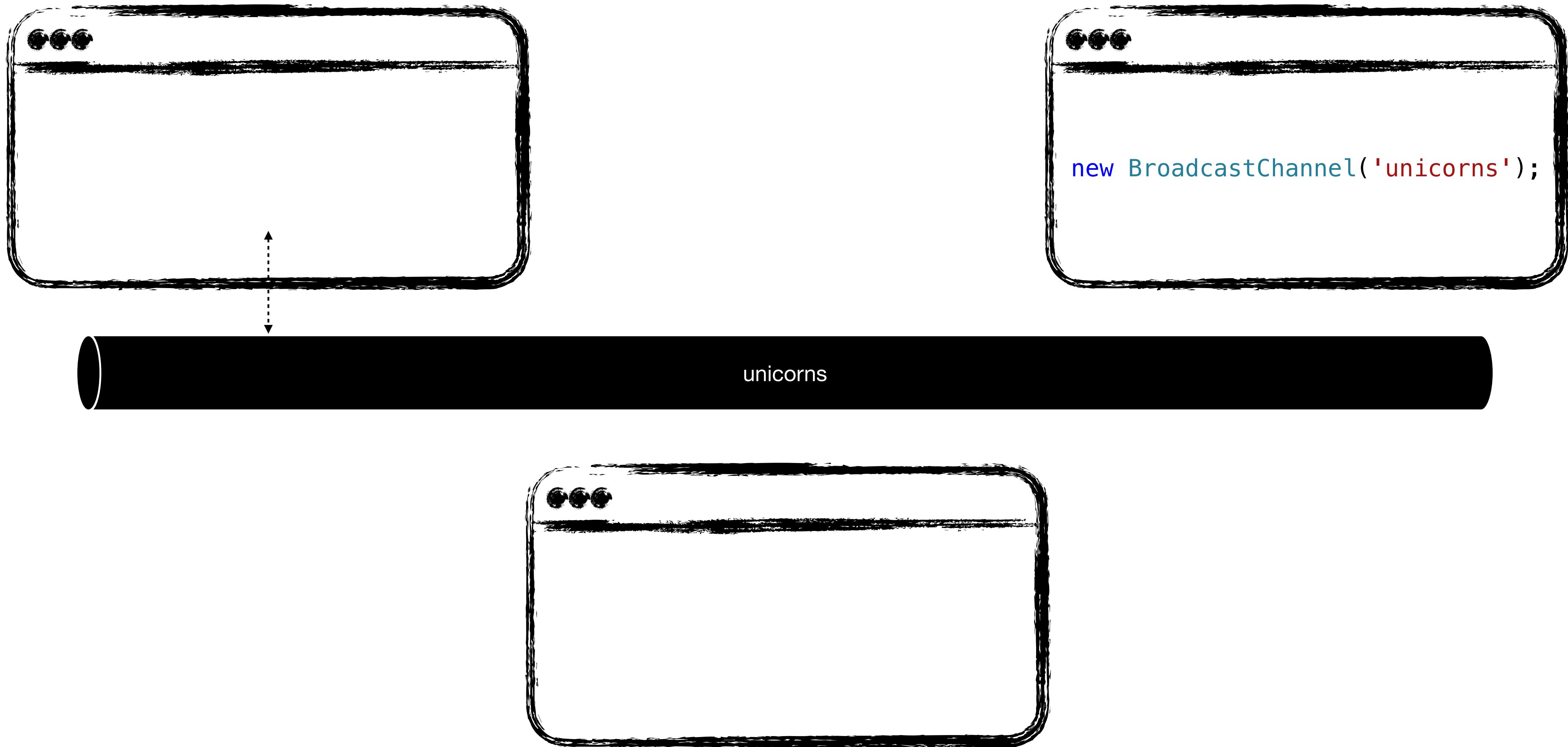
unicorns



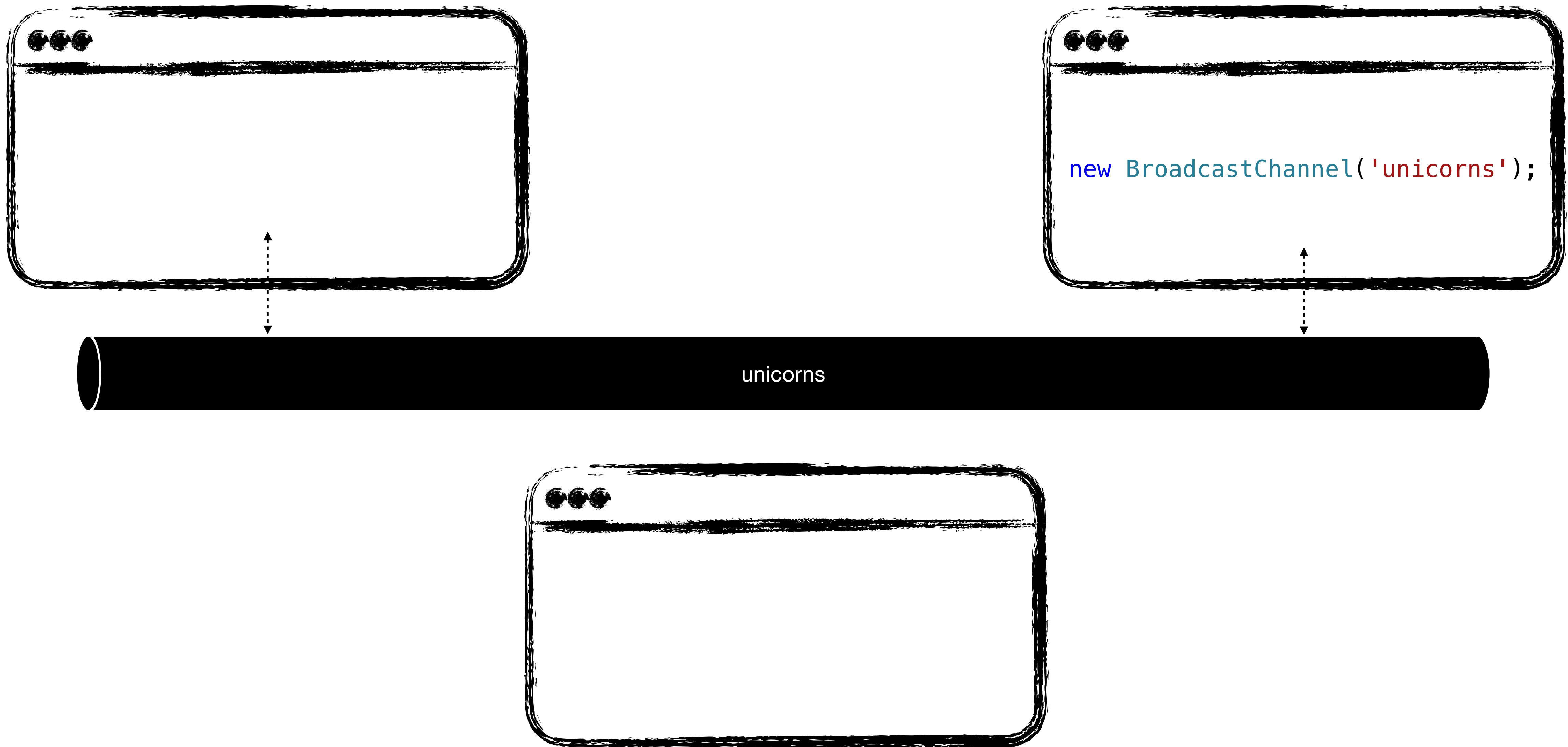
BroadcastChannel



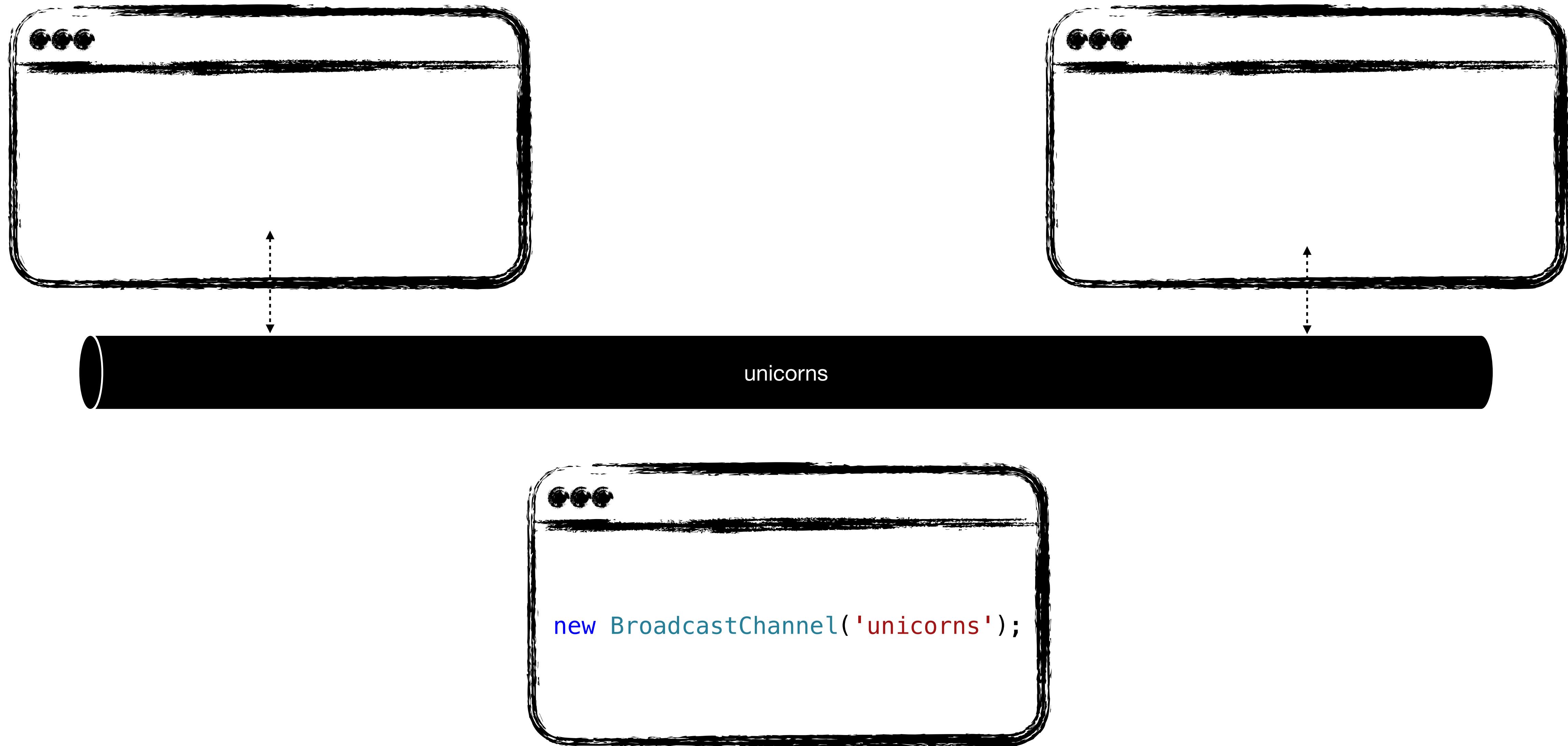
BroadcastChannel



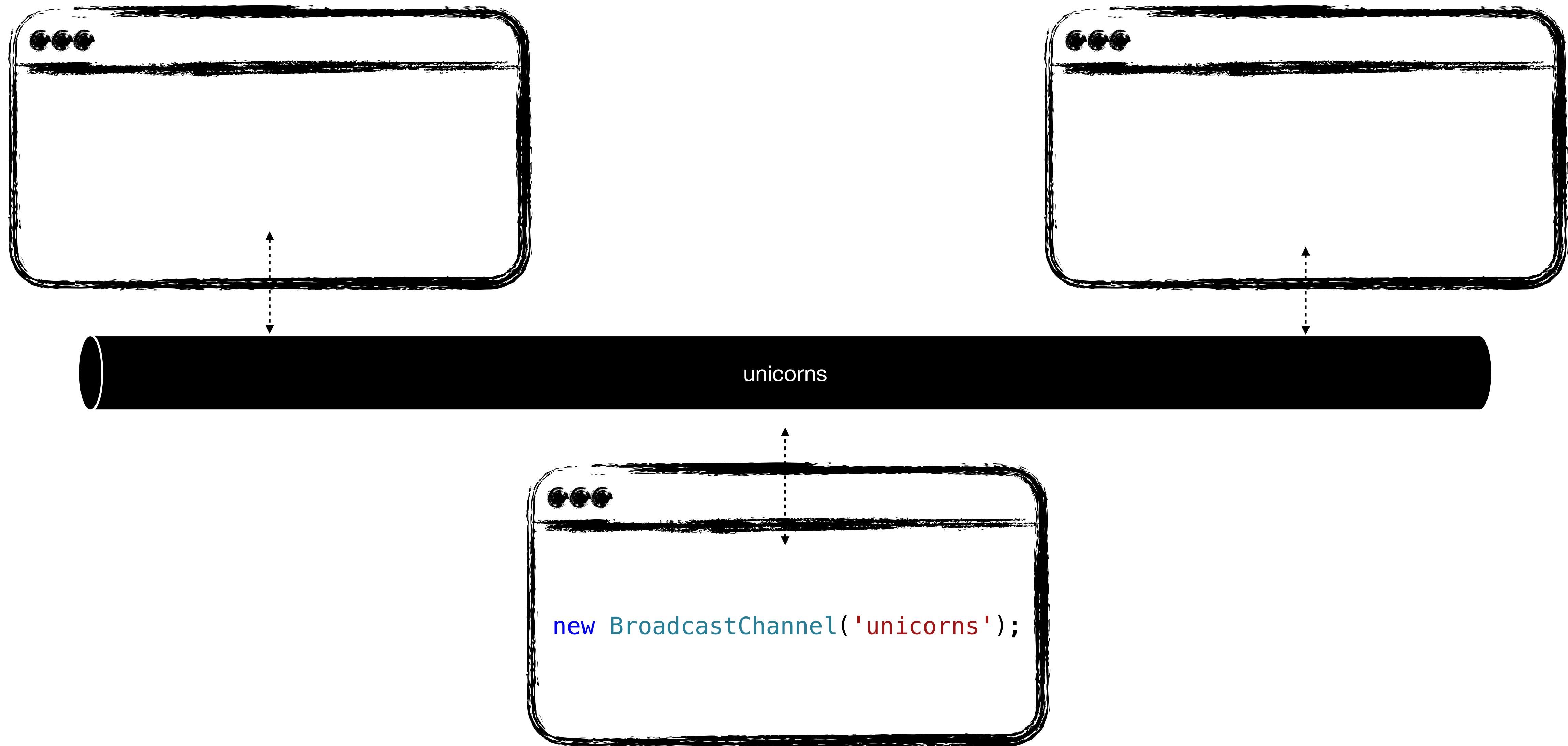
BroadcastChannel



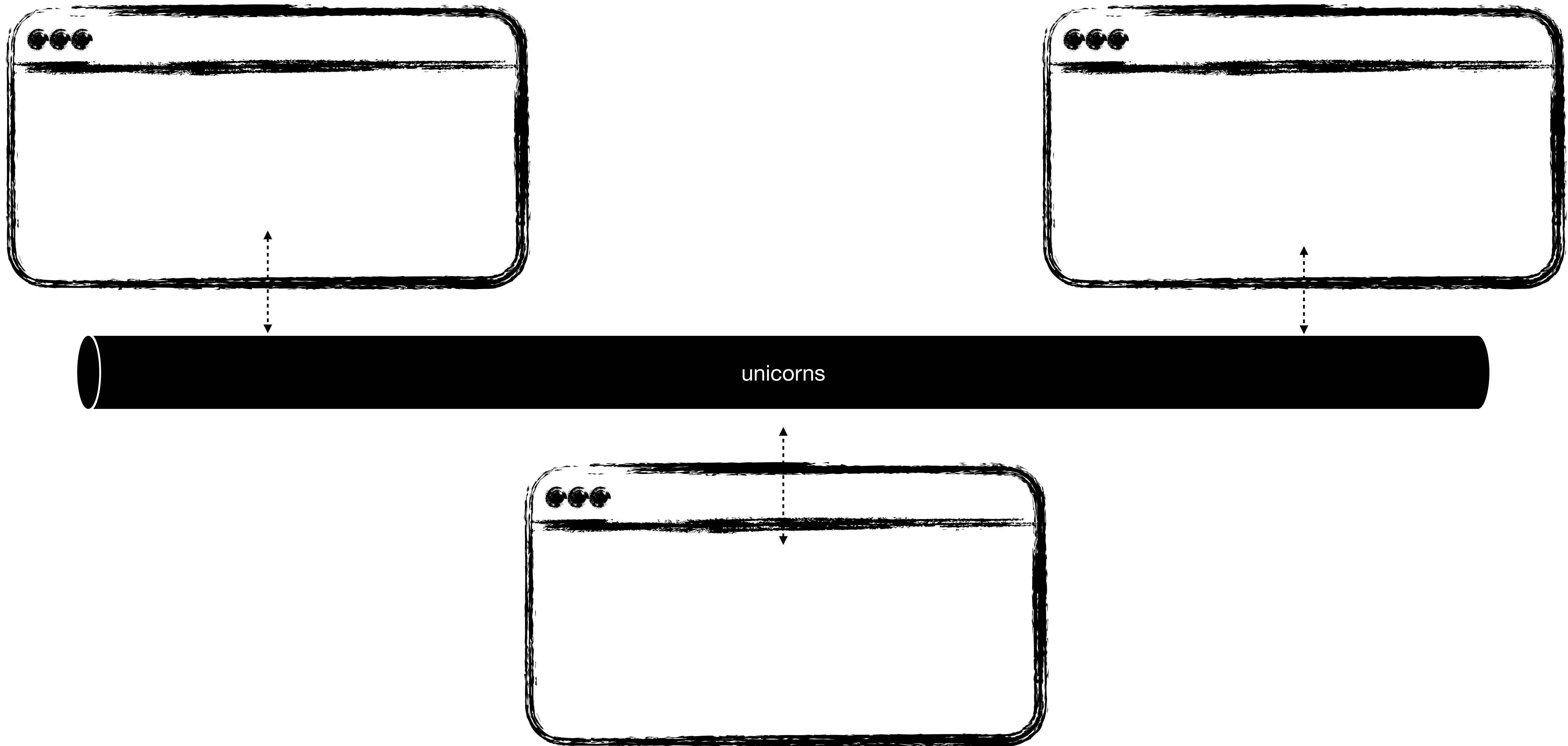
BroadcastChannel



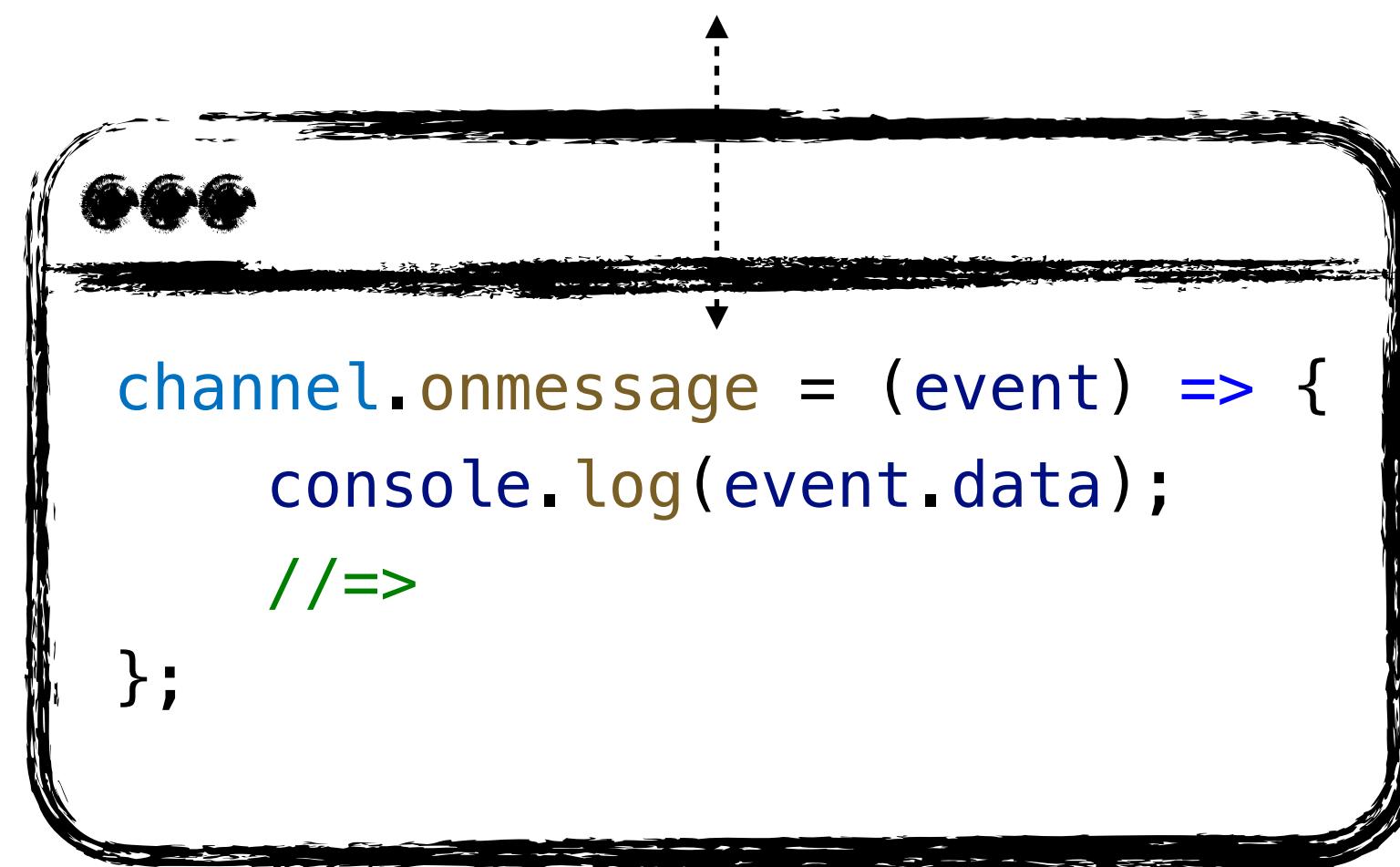
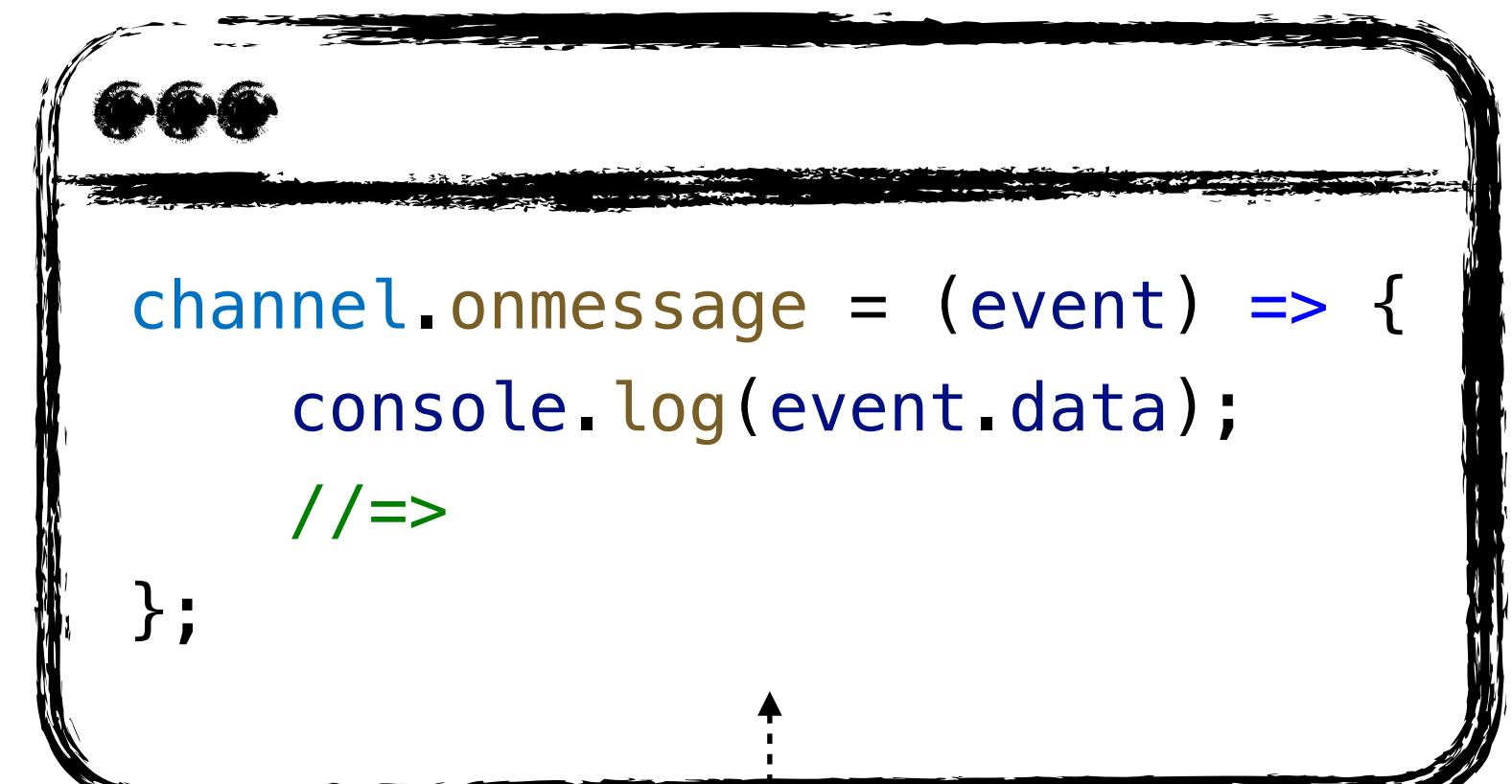
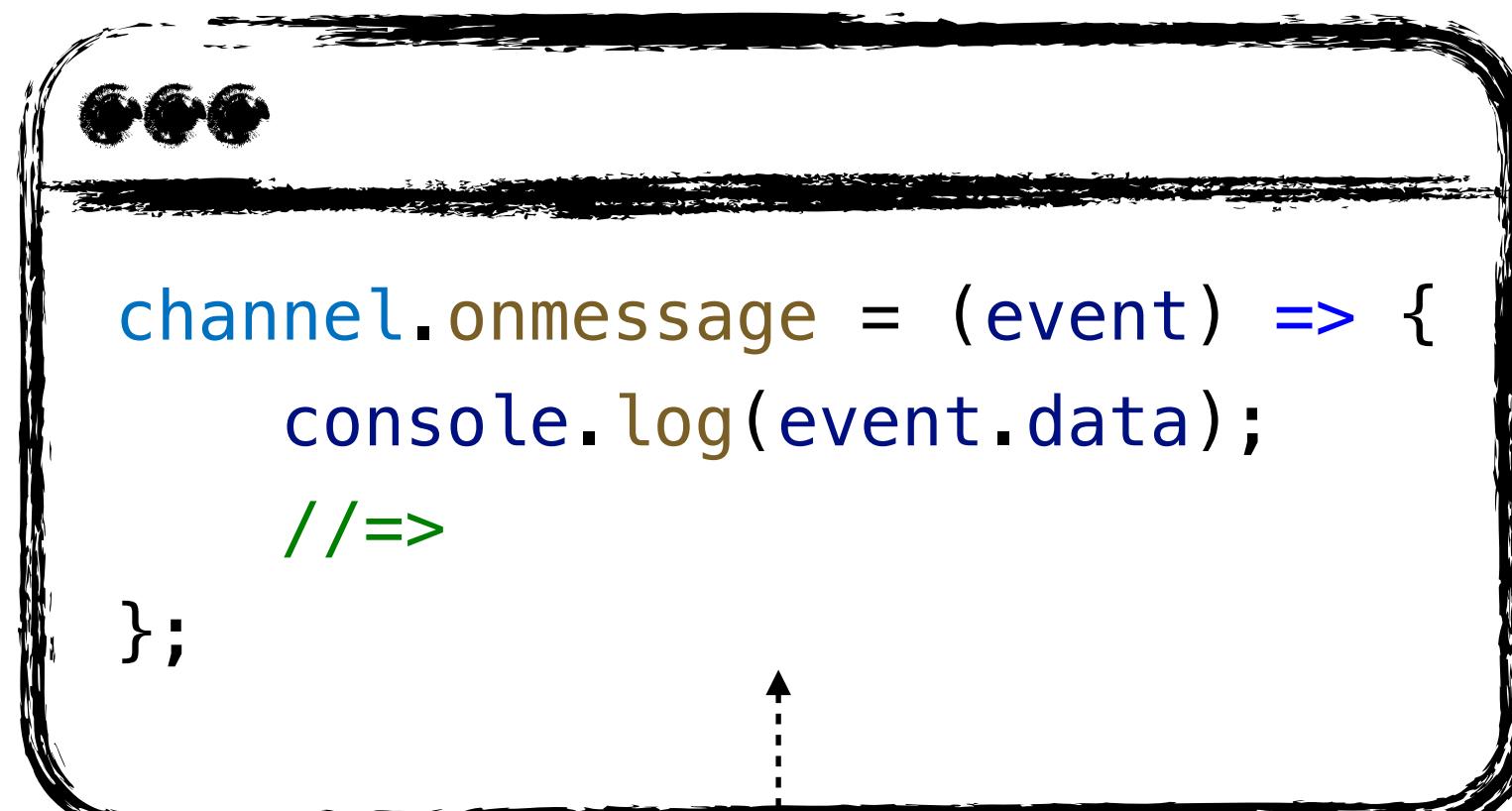
BroadcastChannel



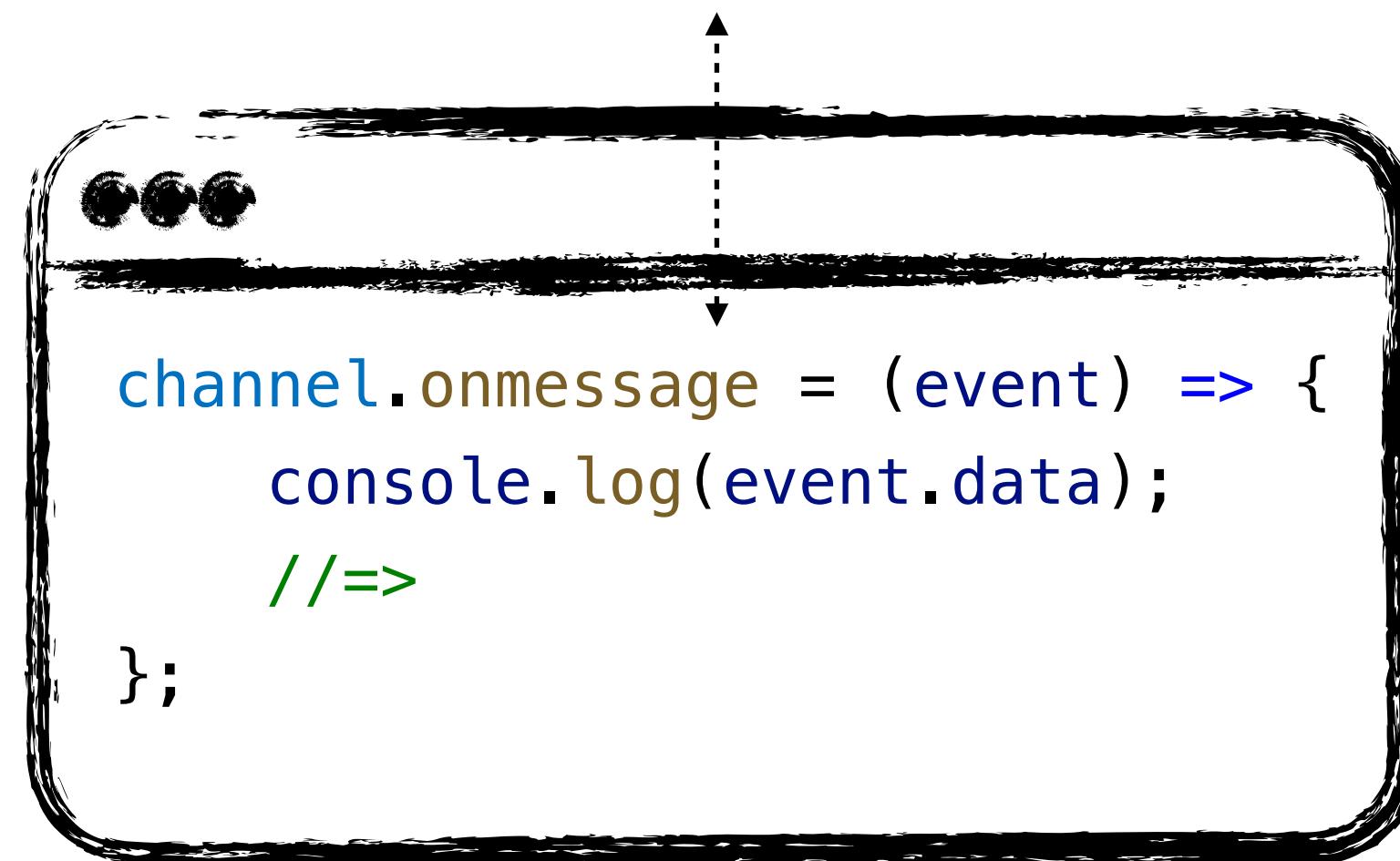
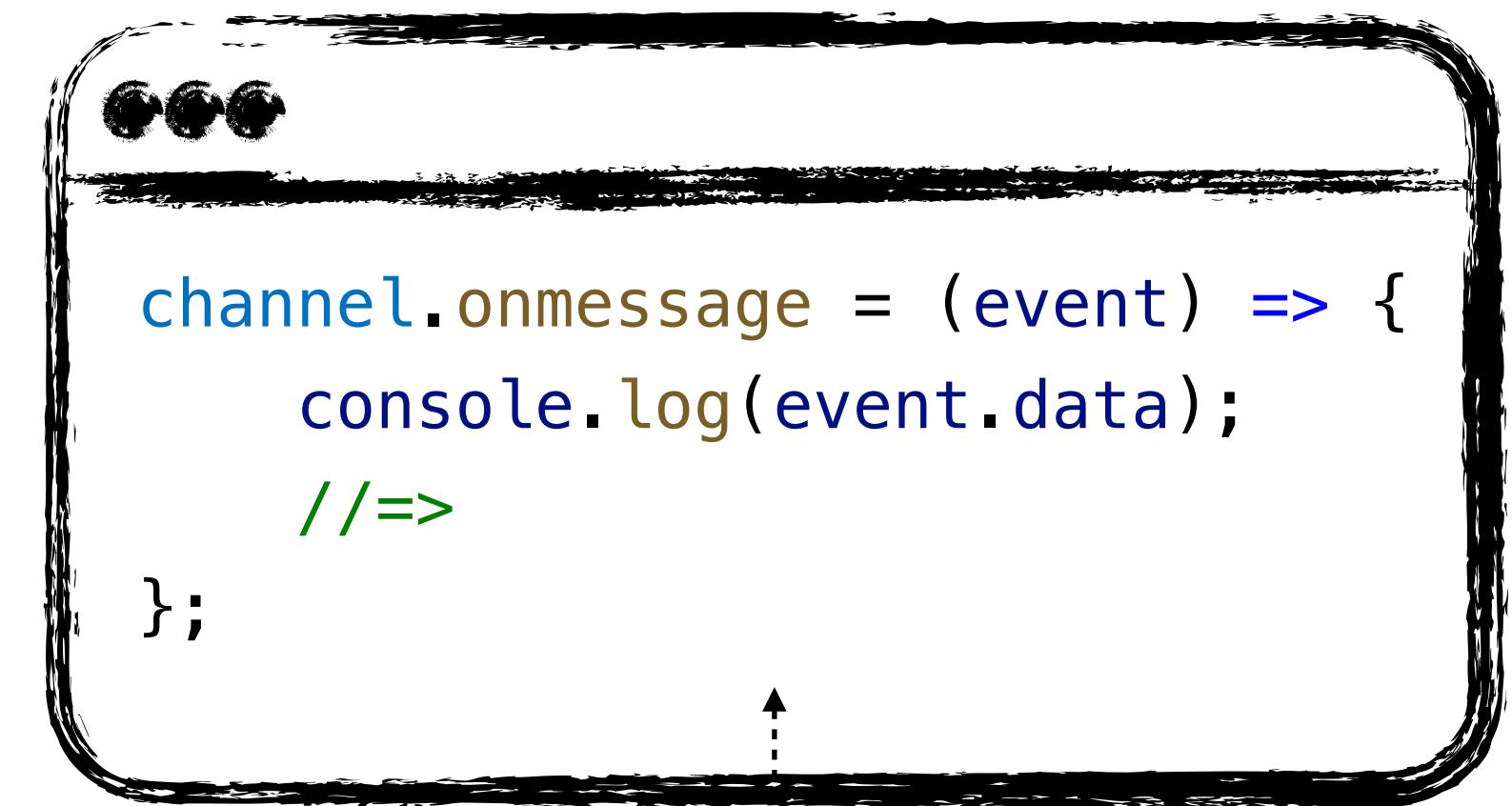
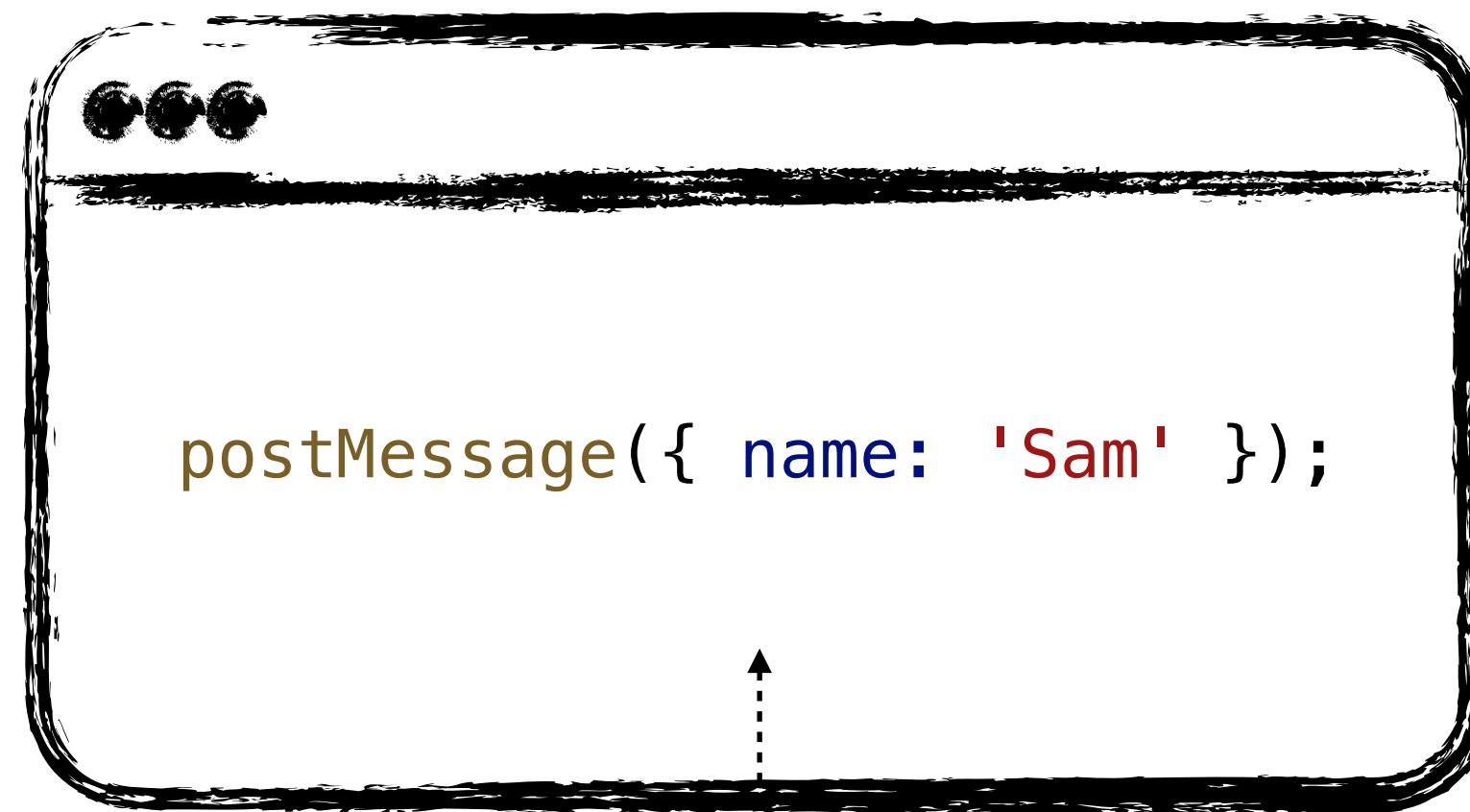
BroadcastChannel



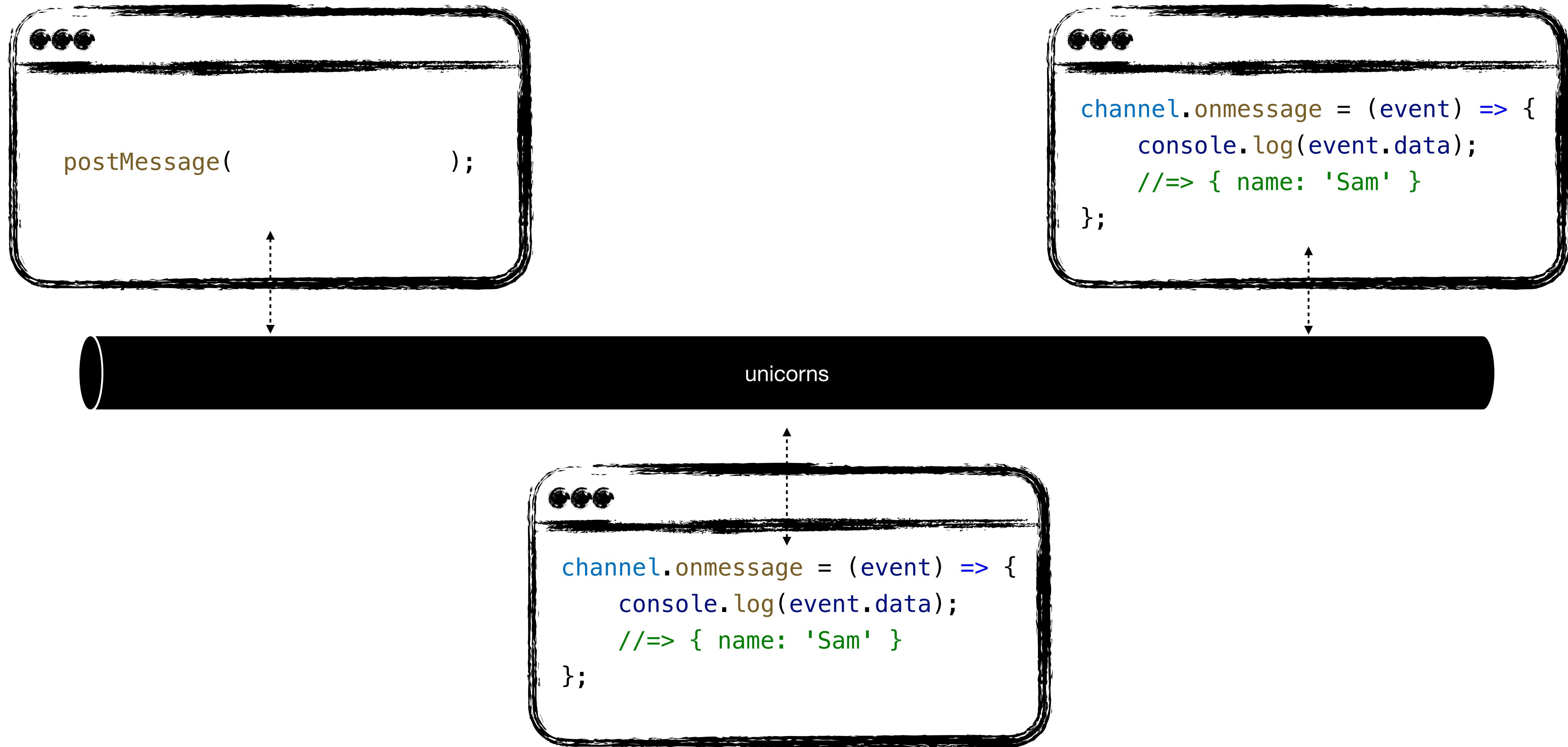
BroadcastChannel



BroadcastChannel



BroadcastChannel

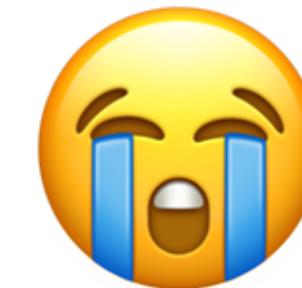


Angular Support

SharedWorker



Not out-of-the-box



<https://github.com/angular/angular-cli/issues/20279>

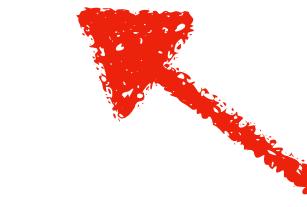
No Angular CLI command 

So what can we do? 🤔

assets/worker.mjs

```
self.onmessage = ({ data }) => {
  const response = `worker response to ${data}`;
  self.postMessage(response);
};
```

assets/worker.mjs



This is not a .ts file!!!

```
self.onmessage = ({ data }) => {
  const response = `worker response to ${data}`;
  self.postMessage(response);
};
```

Downsides

Downsides

- No file hashing

Downsides

- No file hashing
- No easy import statements
 - Besides https urls

Downsides

- No file hashing
- No easy import statements
 - Besides https urls
- No TypeScript

Downsides

- No file hashing
- No easy import statements
 - Besides https urls
- No TypeScript
- `worker.onmessage` does not run inside NgZone!
 - SharedWorker & BroadcastChannel!

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > 3-angular-todo

Exercise 1
Exercise 2

Race conditions

Issues with SAB

- Shared memory
- No control over who writes/reads when
- We could be writing at the same time, corrupting data



Enter Atomics

Atomics

- Exposes static methods
- Assures order in operations
- Synchronisation primitives
- Allows to block a thread 😱 (not the main thread though 😅)

Atomics functions

```
// create a SharedArrayBuffer with a size in bytes
const buffer = new SharedArrayBuffer(16);
const uint8 = new Uint8Array(buffer);
uint8[0] = 7;

// 7 + 2 = 9
console.log(Atomics.add(uint8, 0, 2));
```

Atomics functions

```
// create a SharedArrayBuffer with a size in bytes
const buffer = new SharedArrayBuffer(16);
const uint8 = new Uint8Array(buffer);
uint8[0] = 5;

// 5 + 2 = 7
console.log(Atomics.add(uint8, 0, 2));
// expected output: 5

console.log(Atomics.load(uint8, 0));
// expected output: 7
```

Atomics functions

```
// Main thread
const sab = new SharedArrayBuffer(1024);
const int32 = new Int32Array(sab);
```

```
// Worker
// Wait only works on Int32Arrays
Atomics.wait(int32, 0, 0);
console.log(int32[0]); // 123
```

```
// Main thread
console.log(int32[0]); // 0;
Atomics.store(int32, 0, 123);
Atomics.notify(int32, 0, 1);
```

Storing data into a UintXArray

- Works for numbers
- Doesn't work for anything else, including strings
- We have to transform them first

TextDecoder and TextEncoder

```
const encoder = new TextEncoder();
const msg: Uint8Array = encoder.encode('Hello NgBe');

const decoder = new TextDecoder();

const result = decoder.decode(msg);
// Hello NgBe
```

Practical example

BuilderIO / partytown (Public)

Code Issues 6 Pull requests 2 Actions Security Insights

main 1 branch 16 tags Go to file Add file Code

manucorporat fix: readImplementationMember exception handling (#37)	1 ✓ fb146ee 2 hours ago	297 commits
.github/workflows	feat: implement atomics backend (#33)	5 days ago
scripts	feat: nodejs utils to copy lib files	13 hours ago
src	fix: readImplementationMember exception handling (#37)	2 hours ago
tests	fix: skip non-standard __prefixed props	13 hours ago
.gitattributes	fix: ignore all tests (#20)	2 months ago
.gitignore	feat: nodejs utils to copy lib files	13 hours ago
.prettierignore	chore: add prettier ignore	5 days ago
DEVELOPER.md	update test site url	2 months ago
LICENSE	Initial commit	3 months ago
README.md	docs: update atomics info	5 days ago
package-lock.json	0.0.16	13 hours ago
package.json	0.0.16	13 hours ago

About
Relocate resource intensive third-party scripts off of the main thread and into a web worker. 🎉

partytown.builder.io

javascript performance dom
analytics web-worker 3rd-party
webworker lighthouse 3rdparty
lighthouse-score core-web-vitals

Readme
MIT License

Releases 16
v0.0.16 Latest
13 hours ago
+ 15 releases

Let's build it!

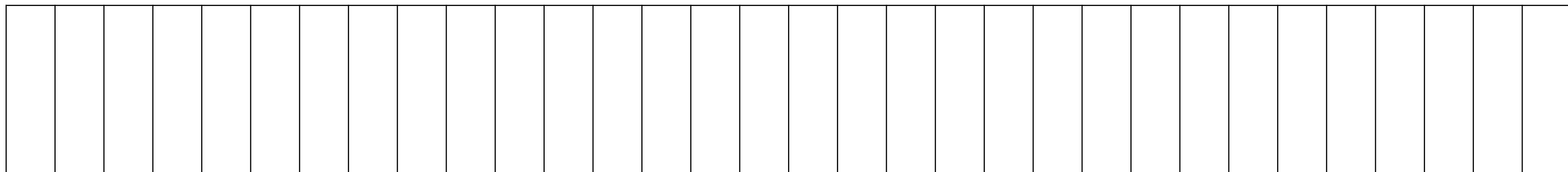
- Dynamically execute the scripts in a web worker
- Intercept calls made to DOM api's
- Block the web worker while waiting for an answer
- Grab the value on the main thread and send it to the worker
- Unblock the worker

Let's build it!

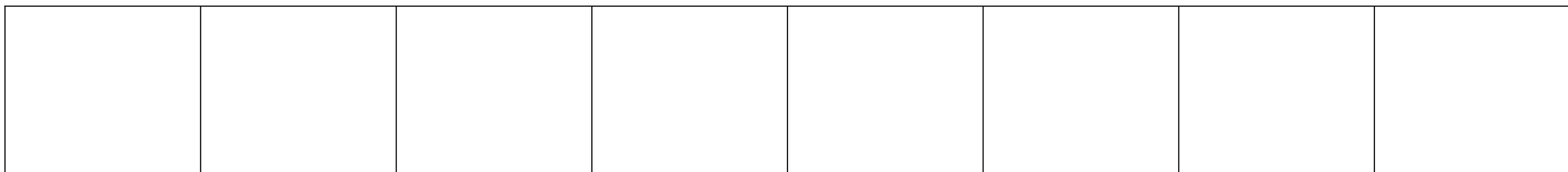
- Dynamically execute the scripts in a web worker → Eval
- Intercept calls made to DOM api's → Proxy
- Block the web worker while waiting for an answer → Atomics.wait
- Grab the value on the main thread and send it to the worker
- Unblock the worker → Atomics.store & Atomics.notify

Data structure

SAB

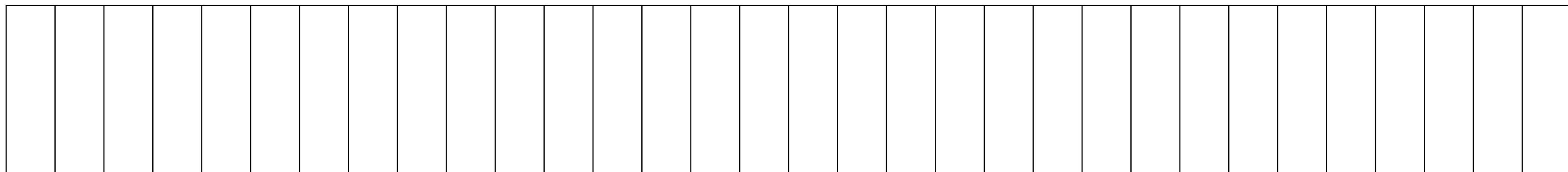


Int32Array

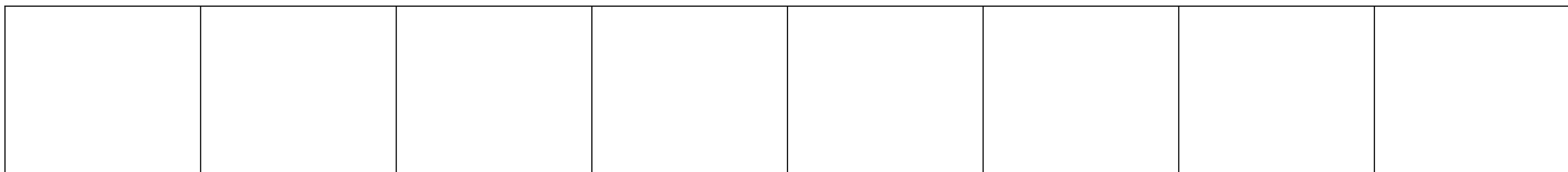


Data structure

SAB

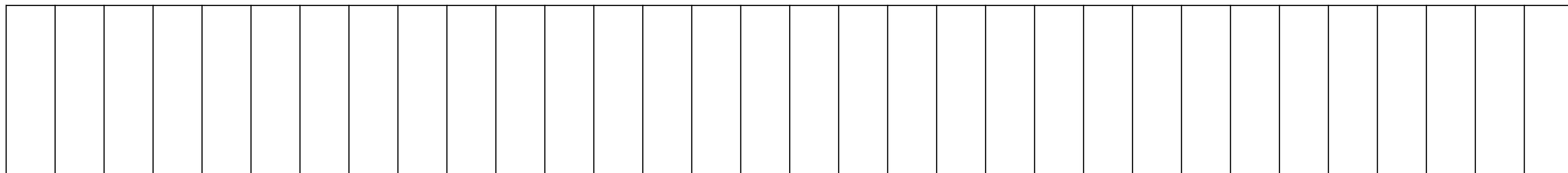


Int32Array

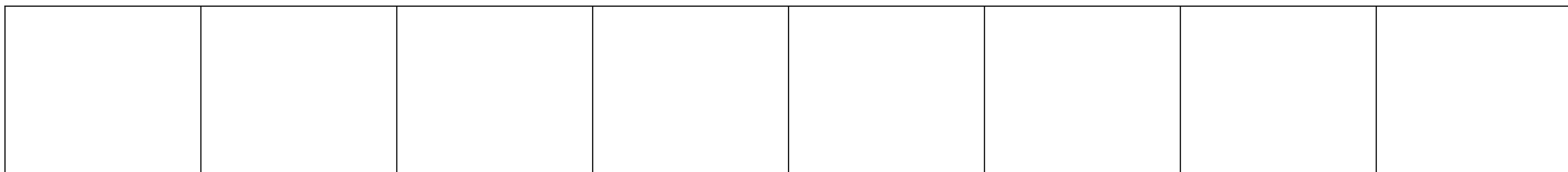


Data structure

SAB



Int32Array

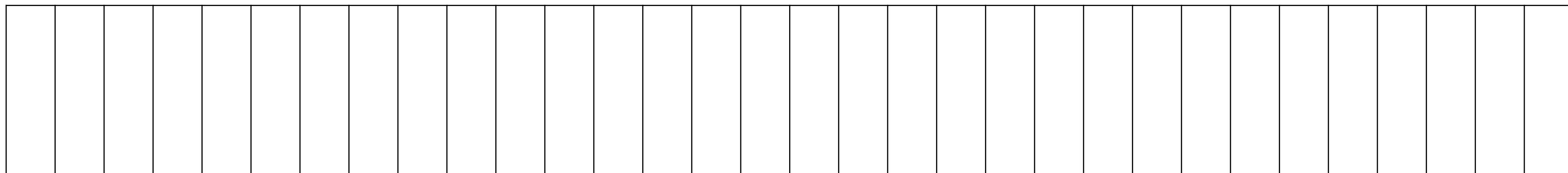


Used to
block and
unblock

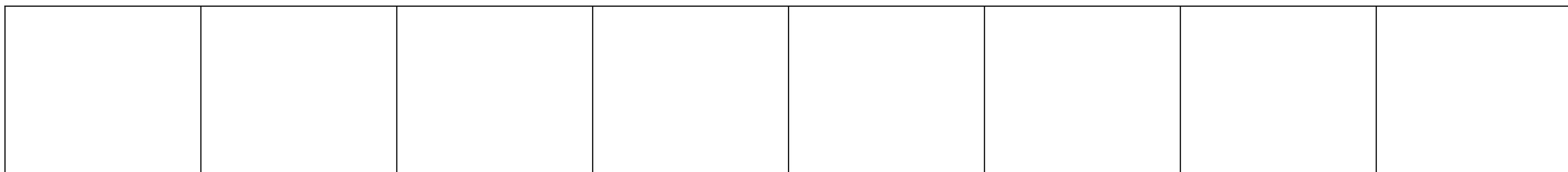


Data structure

SAB



Int32Array

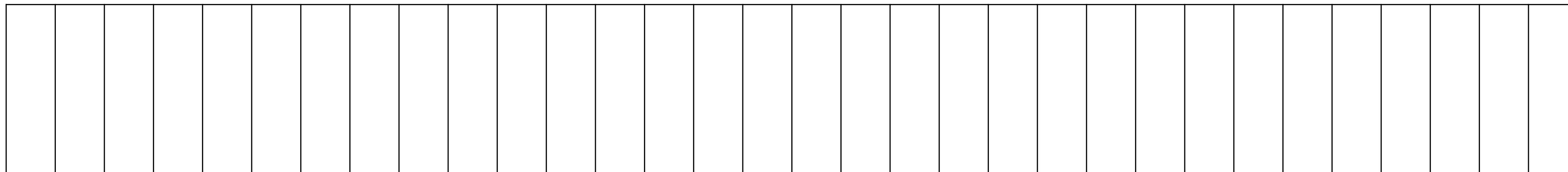


Used to
block and
unblock

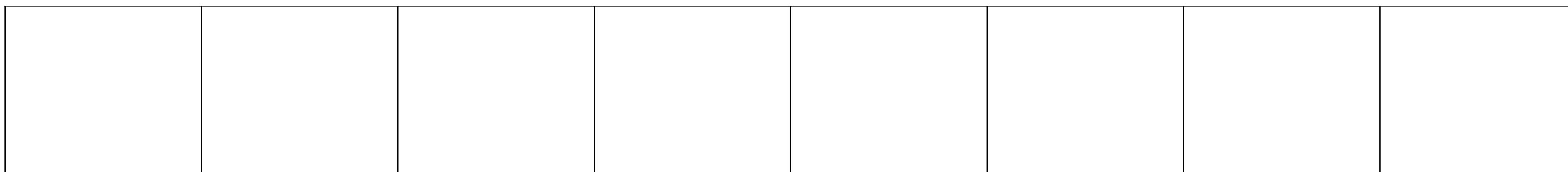
A black curved arrow originates from the bottom of the text "Used to block and unblock" and points downwards towards the center of the SAB diagram.

Data structure

SAB



Int32Array

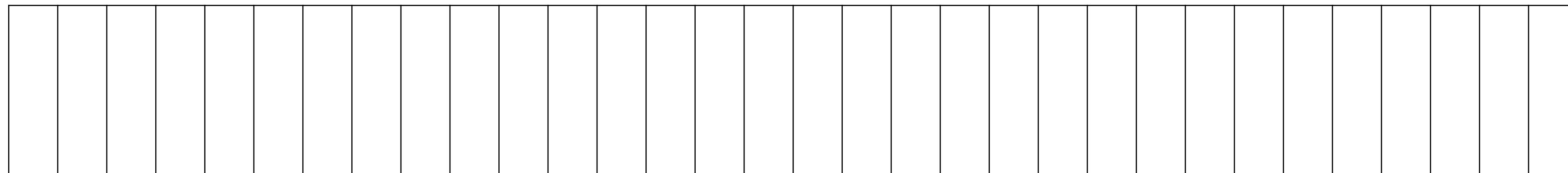


Used to
block and
unblock

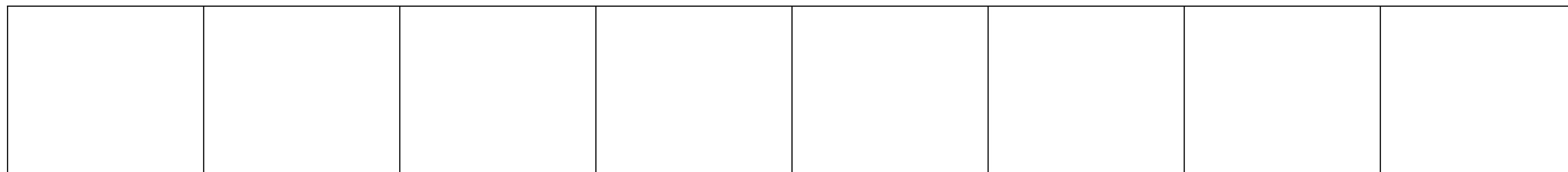
Holds data

Data structure

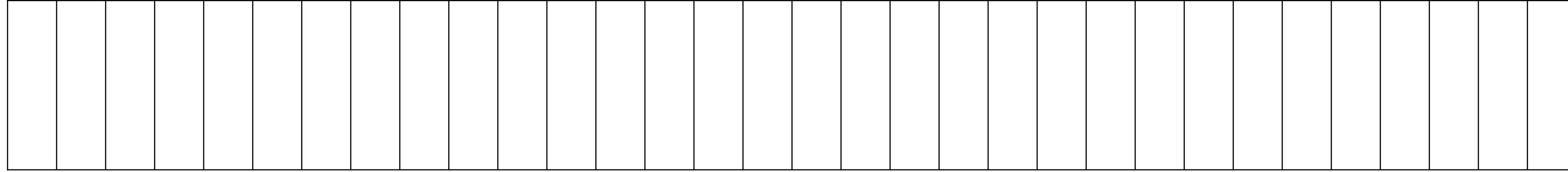
SAB



Int32Array



Uint8Array



Store data

Exercise

<https://github.com/stackblitz/ng-be-workshop>

Exercises > WebWorkers > PartyTown



StackBlitz

Thank you for joining 

NG-BE

Sam Verschueren, Kwinten Pisman & Dominic Elm