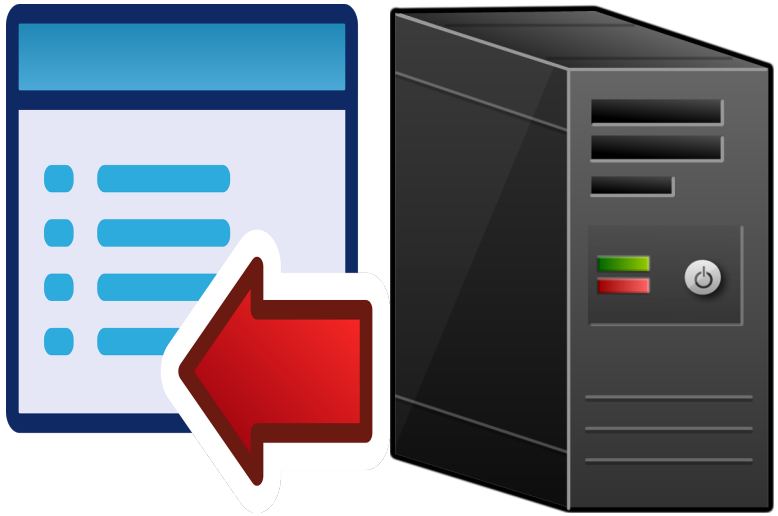
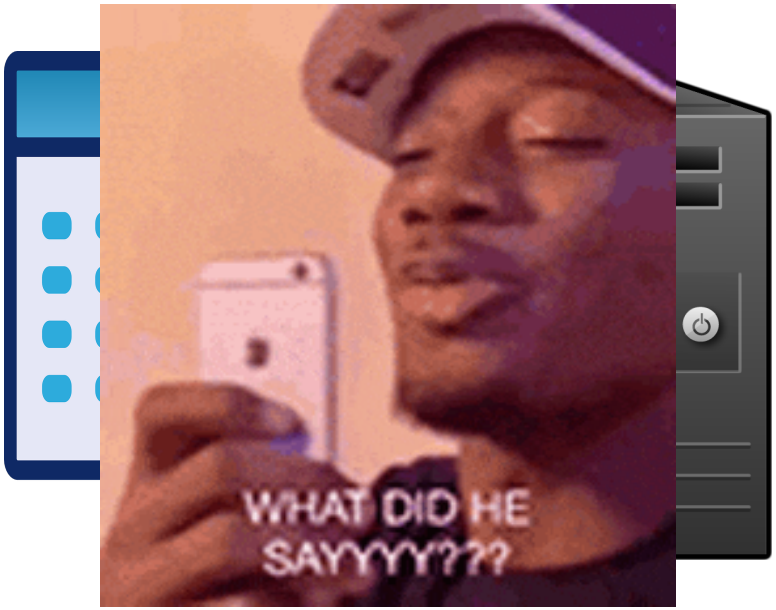
# WEB WORKERS AND HOW YOU CAN USE THEM

# CONVENTIONAL RELATIONSHIP

# CONVENTIONAL RELATIONSHIP



# CONVENTIONAL RELATIONSHIP



# ~~USELESS~~ CPU INTENSIVE WORK

```
1 function hardWork() {
2   let dummyVar = 0;
3   for (let i = 0; i < SOME_BIG_NUMBER; i += 1) {
4     dummyVar += 1;
5   }
6 }
```

# DEMO: CPU INTENSIVE WORK

# WEB WORKERS AND HOW YOU CAN USE THEM

# ROADMAP

- Background on Javascript
- How not to use Promises
- Web Workers

# WHY HAVE I NOT HEARD ABOUT WEB WORKERS BEFORE THIS?

# WHY JAVASCRIPT WORKS SO WELL (MOST OF THE TIME)

# WHY JAVASCRIPT WORKS SO WELL (MOST OF THE TIME)

- Non-blocking event loop - does not block on IO

# WHY JAVASCRIPT WORKS SO WELL (MOST OF THE TIME)

- Non-blocking event loop - does not block on IO
- CPU intensive tasks are not common

# PROMISES

- Async, Await syntax
  *“eventual completion (or failure) of an asynchronous operation and its resulting value”*

# PROMISES DON'T HELP

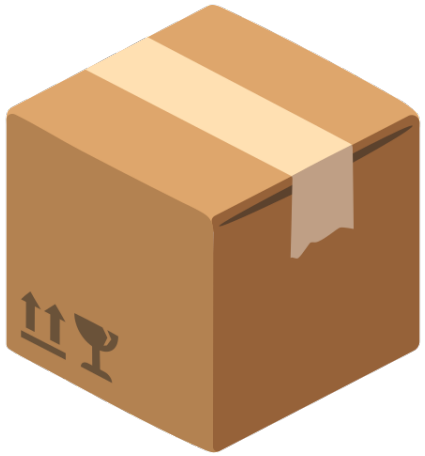
No I/O involved, we need to do the work now or later

# PROMISES WITH IO - ORDER SOMETHING ON AMAZON

# PROMISES WITH IO - ORDER SOMETHING ON AMAZON

# PROMISES WITH IO - ORDER SOMETHING ON AMAZON

# PROMISES WITH IO - ORDER SOMETHING ON AMAZON



## PROMISES WITH NO IO (CPU)

Promising yourself to get something

# PROMISES WITH NO IO (CPU)

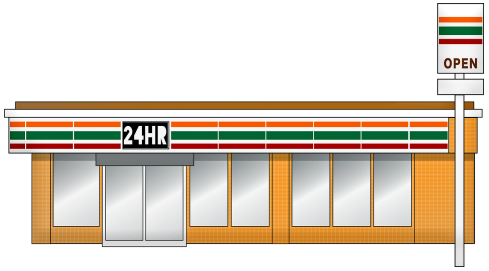
Promising yourself to get something

# PROMISES WITH NO IO (CPU)

Promising yourself to get something

# PROMISES WITH NO IO (CPU)

Promising yourself to get something


OPEN
24HR

# HOW WE MIGHT TRY TO USE PROMISES FOR CPU BOUND TASKS?

```
1  function hardWork() {
2    // The hard work
3  }
4
5  function promiseHardWork() {
6    return new Promise((resolve, reject) => {
7      hardWork()
8      resolve()
9    })
10 }
11
12 function promiseHardWork2() {
13   return new Promise((resolve, reject) => {
14     resolve()
```

# WEB WORKERS

# WEB WORKERS

- Creates new processes(Not formally defined in JS spec)

# WEB WORKERS

- Creates new processes(Not formally defined in JS spec)
- Initialised using a JS file

# WEB WORKERS

- Creates new processes(Not formally defined in JS spec)
- Initialised using a JS file
- Gets work from the main thread

# USING WEB WORKERS (MAIN THREAD)

```
1 const worker = new Worker("worker.js");
2 worker.postMessage(None) // pass worker data if any
3
4 worker.onmessage((e) => { // listen to worker
5   console.log("Worker done with work!")
6 })
```

# USING WEB WORKERS (MAIN THREAD)

```
1 const worker = new Worker("worker.js");
2 worker.postMessage(None) // pass worker data if any
3
4 worker.onmessage((e) => { // listen to worker
5   console.log("Worker done with work!")
6 })
```

# USING WEB WORKERS (MAIN THREAD)

```
1 const worker = new Worker("worker.js");
2 worker.postMessage(None) // pass worker data if any
3
4 worker.onmessage((e) => { // listen to worker
5   console.log("Worker done with work!")
6 })
```

# USING WEB WORKERS (WORKER.JS)

```
1 function hardWork() {
2   // The hard work
3 }
4
5 function onmessage(e) { // Message from parent to start task
6   hardWork()
7   postmessage("Done!") // Notify main thread
8 }
```

# USING WEB WORKERS (WORKER.JS)

```
1 function hardWork() {
2   // The hard work
3 }
4
5 function onmessage(e) { // Message from parent to start task
6   hardWork()
7   postmessage("Done!") // Notify main thread
8 }
```

# USING WEB WORKERS (WORKER.JS)

```
1 function hardWork() {
2   // The hard work
3 }
4
5 function onmessage(e) { // Message from parent to start task
6   hardWork()
7   postmessage("Done!") // Notify main thread
8 }
```

# USING WEB WORKERS (WORKER.JS)

```
1 function hardWork() {
2   // The hard work
3 }
4
5 function onmessage(e) { // Message from parent to start task
6   hardWork()
7   postmessage("Done!") // Notify main thread
8 }
```

# DEMO: USING WEB WORKERS - UNBLOCKS MAIN THREAD

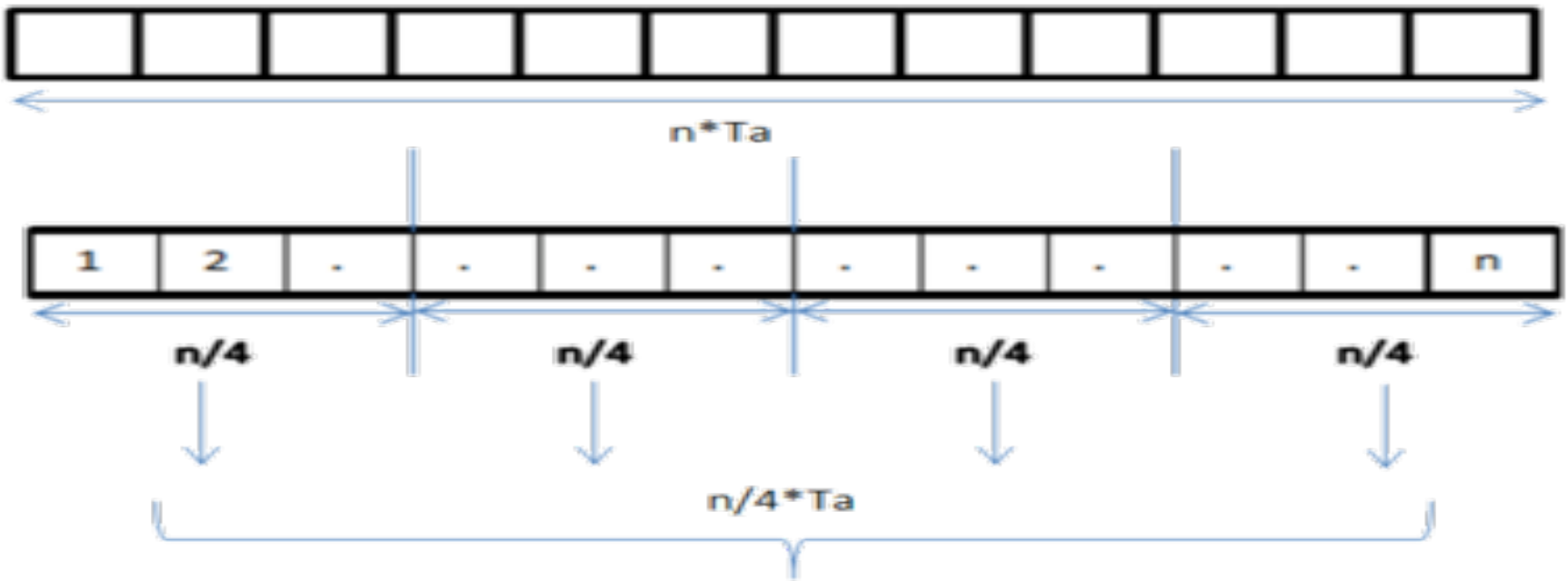
# ADVANTAGES

# BENEFIT 1: MAKES USE OF YOUR COMPUTER HARDWARE!

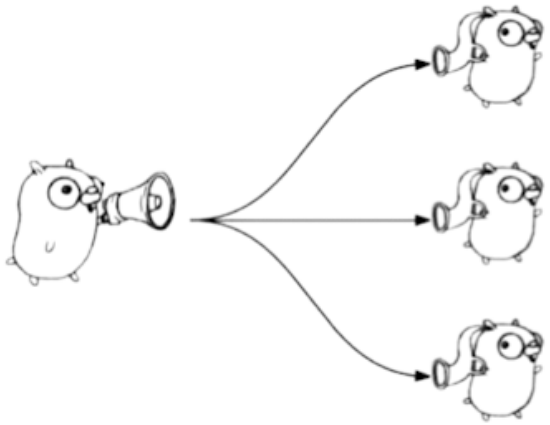
Maximise users hardware and shift computing off the cloud

# BENEFIT 2: CAN SPEED UP COMPUTATION

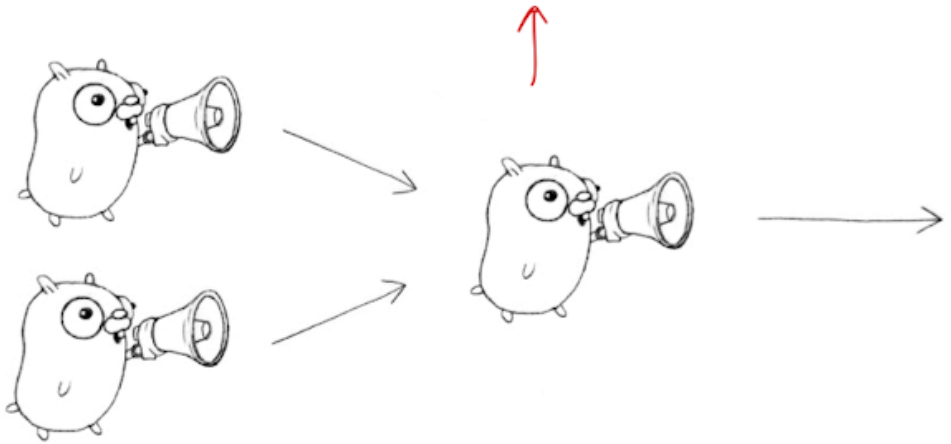
Data parallelism



# Data parallelism patterns



fan-out



fan-in

# Task Parallelism

pipelining

# BENEFIT 3: MEMORY SAFE

No concurrency issues

# WEB WORKER IMPLEMENTATION

Are used via clearly defined API

```
1 postmessage(data)
2
3 onmessage((e) => {
4   // do something
5 })
```

# DISADVANTAGES

## DISADVANTAGE 1: DO NOT SHARE THE SAME MEMORY SPACE

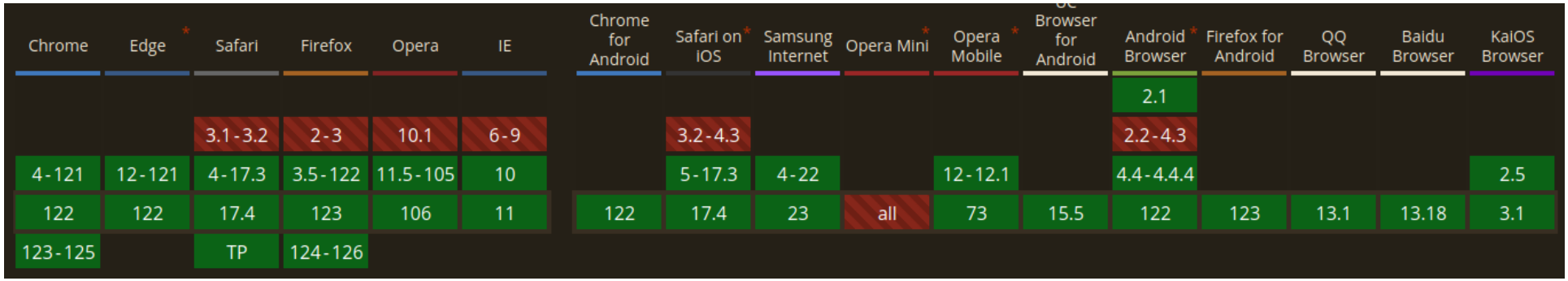
Copy data from main thread to worker vice versa

```
1 postmessage(data)
```

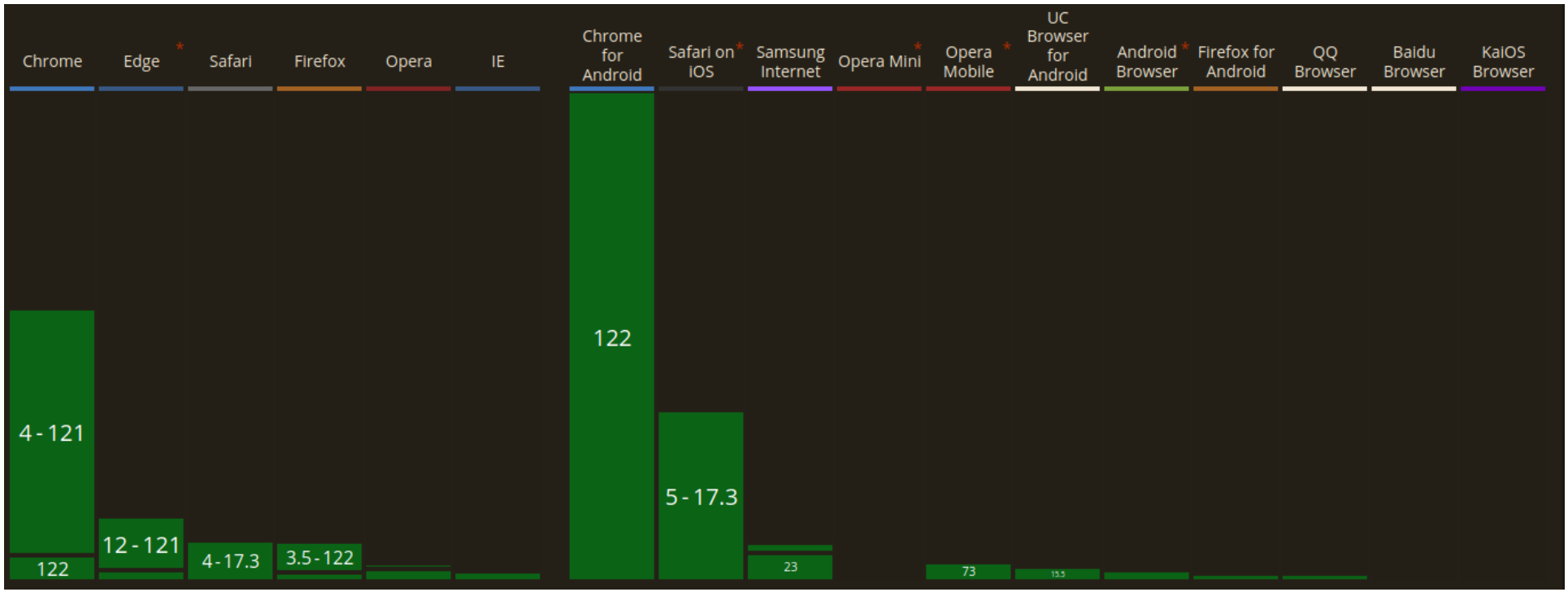
## DISADVANTAGE 2: CANNOT MANIPULATE THE WEB PAGE(DOM)

Only the main thread can interact with the DOM

# DISADVANTAGE 3: NOT ALL BROWSERS SUPPORT IT!



| Chrome | Edge * | Safari | Firefox | Opera | IE | Chrome for Android | Safari on iOS * | Samsung Internet | Opera Mini * | Opera Mobile * | UC Browser for Android | Android Browser * | Firefox for Android | QQ Browser | Baidu Browser | KaiOS Browser |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | 2.1 | | | | |
| | | 3.1-3.2 | 2-3 | 10.1 | 6-9 | | 3.2-4.3 | | | | | 2.2-4.3 | | | | |
| 4-121 | 12-121 | 4-17.3 | 3.5-122 | 11.5-105 | 10 | | 5-17.3 | 4-22 | | 12-12.1 | | 4.4-4.4.4 | | | | 2.5 |
| 122 | 122 | 17.4 | 123 | 106 | 11 | 122 | 17.4 | 23 | all | 73 | 15.5 | 122 | 123 | 13.1 | 13.18 | 3.1 |
| 123-125 | | TP | 124-126 | | | | | | | | | | | | | |


Chrome
Edge *
Safari
Firefox
Opera
IE
Chrome for Android
Safari on iOS *
Samsung Internet
Opera Mini *
Opera Mobile *
UC Browser for Android
Android Browser *
Firefox for Android
QQ Browser
Baidu Browser
KaiOS Browser
4 - 121
122
12 - 121
4 - 17.3
3.5 - 122
122
5 - 17.3
23
73

# WHY TRY WEB WORKERS TODAY?

- Web Workers are memory safe and easy to use
- Make full use of your user's hardware
- Supported by almost devices