Binary encoding in JavaScript

A story about efficient communication between JS and Rust, and how binary encoding might be used with WebWorkers and WebAssembly.

About me

- I’m Simon Korzunov 👋
- Former game developer 🎮
- Love programming languages (especially FP 👽)
- Learned Typescript before JS
- @dropbox I work on JS perf on desktop 🏎

Yawn!

The origin of this talk

https://github.com/cztomsik/graffiti

Started as a mixture of Node.js and FireFox rendering engine WebRender written in Rust. Ambition: replace Electron

Challenge: How to efficiently communicate between Rust and JS

Tried just passing strings containing JSON data

```
sendToRust(JSON.stringify({ data: "hello from js" }));
```

- 30% of the time decoding JSON strings in Rust 😟
- JS uses utf-16 but Rust uses utf-8

Decided to pass binary data instead

```
sendToRust(new Uint8Array([0, 1, 2, 3]));
```

- But how to encode data?

Encoding into Uint8Array Example #1

```
const hotel = { stars: 3, breakfast: true };

const encoded = new Uint8Array([
  3, // stars
  1  // breakfast
]);

// also we can decode it back
const decoded = {
  stars: encoded[0],
  breakfast: encoded[1] === 1
};
```

Encoding into Uint8Array Example #2

```ts
const user = { name: "Simon", age: 33 };

const nameBytes = new TextEncoder(“utf-8”)
  .encode(“Simon");
// [ 83, 105, 109, 111, 110 ]

const encodedUser = [
  5, 0, 0, 0, 0, 0, 0, 0, // length as u64
  83, 105, 109, 111, 110, // name bytes
  33,                     // age as u8
];
```

I build a library for that: ts-rust-bridge.

Also this is exactly how “bincode” works

bincode

*https://github.com/servo/bincode A compact encoder / decoder pair that uses a binary zero-fluff encoding scheme (*for rust)*

```
// JS: send using ts-rust-bridge library for encoding
sendToRust(encodeUser({ name: "Simon", age: 33 }));
```

```
// Rust: deserialize using bincode & serde crates
#[derive(Serialize, Deserialize)]
struct User {
  name: String,
  age: u8,
}

let decoded: User = bincode::deserialize(&data)
  .unwrap();
```

How Is it relevant to me? 🤔

Games

DataVis

Mobile

Let’s assume that you do need performance

Web Workers

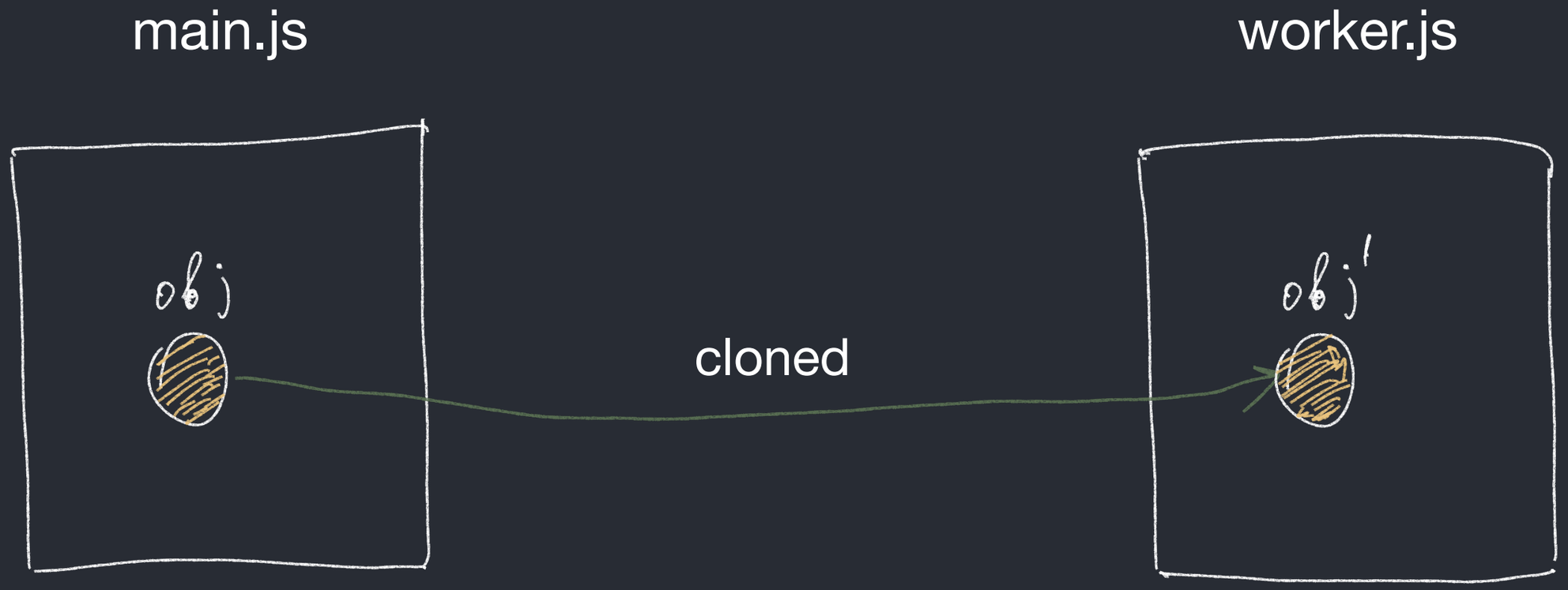
```ts
// copied from lib.dom.d.ts

postMessage(message: any, transfer: Transferable[]): void;

type Transferable = ArrayBuffer | MessagePort | ImageBitmap
```

From HTML Living Standard:

*`transfer` can be passed as a list of objects that are to be **transferred** rather than **cloned**.*

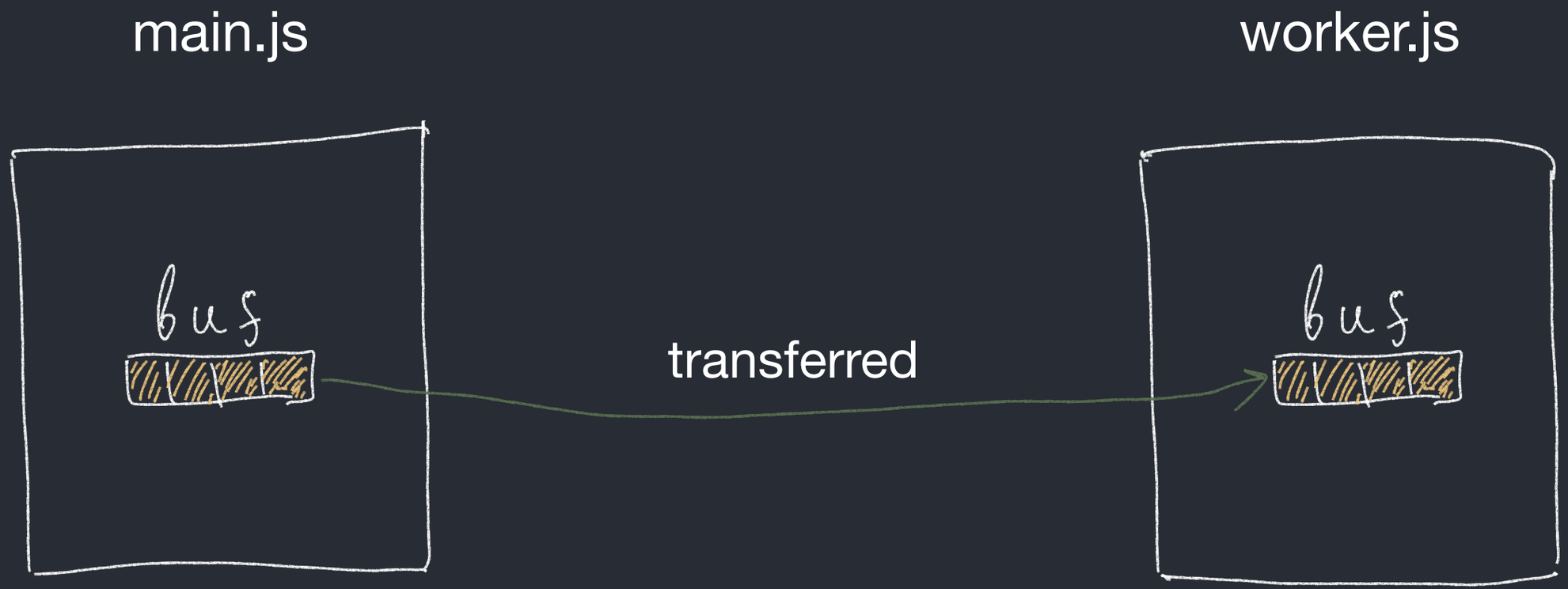
Worker threads in Node.js have similar api 🤔

Web Workers




```
postMessage(obj);
```

Web Workers




```
postMessage(buf,[buf]);
```

WebAssembly

```
const memory = new WebAssembly.Memory({…});

WebAssembly.instantiateStreaming(fetch("some.wasm"), {
  js: { mem: memory }
}).then(…);
```

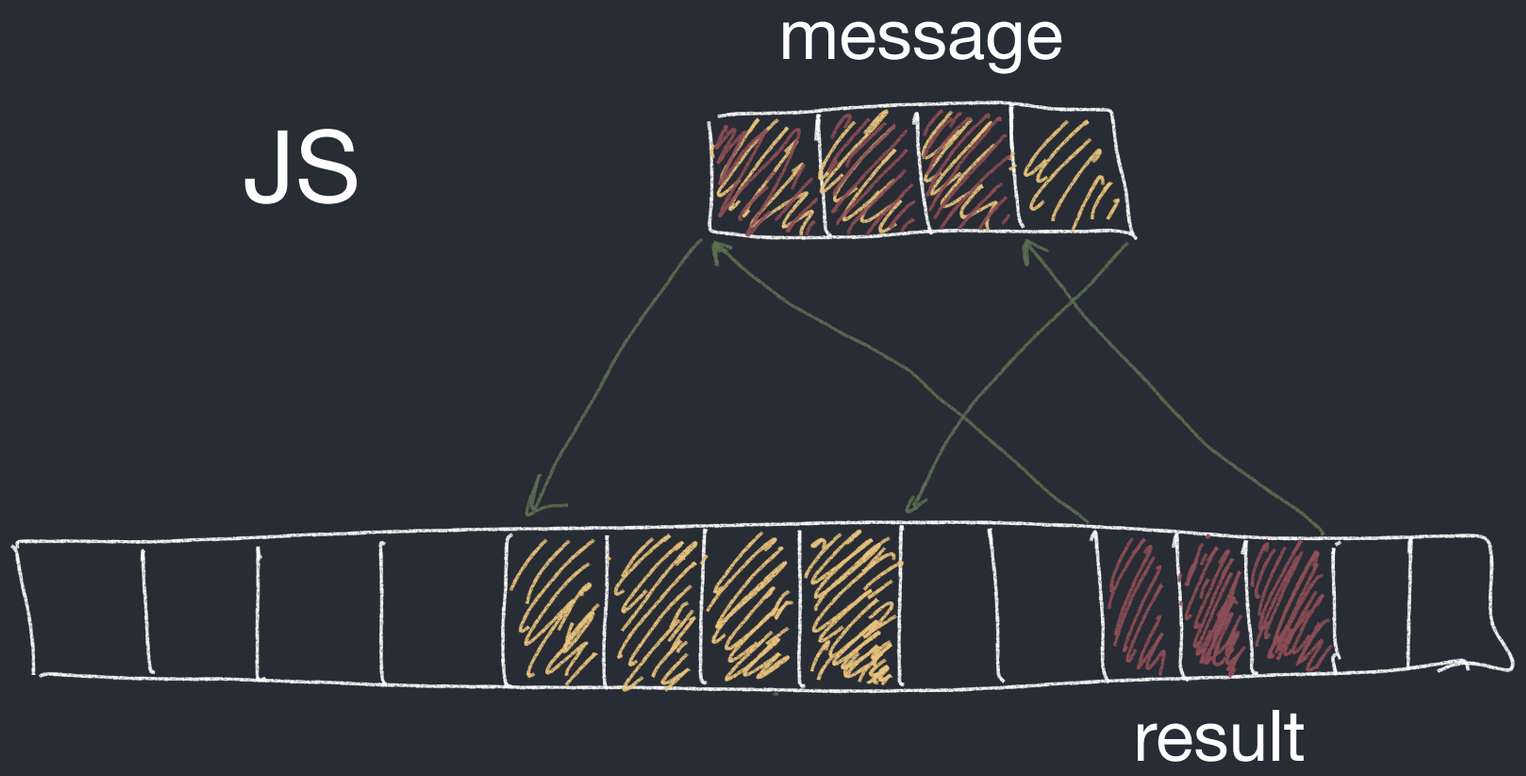
```
// we can use this talk to WASM!!! 🤯
const memoryView = new Uint8Array(memory.buffer);
```

From mdn:

A memory created by JavaScript or in WebAssembly code will be accessible and mutable from both JavaScript and WebAssembly.

WebAssembly

JS



WASM

Demo project

Send and receive 100 messages to a worker that will echo them back.

```
const payload = {
  str: "some random string",
  f64: 544501.0378817371,
  tuple: [true, -39590],
  struct: {
    bool: true,
    i32vec: [-859965, 345717, -37999, -902347, -737603]
  }
  // plus a couple more
};

const message = new Array(1000).fill(payload);
```

Demo project

```
// Structural Cloning
postMessage(message);


// JSON.stringify
postMessage(JSON.stringify(message));
```

```
// ts-rust-bridge
const msg: Uint8Array = encodeIntoBinary(message);
postMessage(msg.buffer, [msg.buffer]);


// WASM + ts-rust-bridge (in a worker)
const memView: Uint8Array = wasm.allocate(msg.length);
memView.set(msg);
const result: Uint8Array = wasm.handle_message();
```

Results

Structural cloning	90ms
JSON.stringify	107ms
ts-rust-bridge	56ms
WASM + ts-rust-bridge	41ms

* *measured in FireFox on mac mini. Each message was ~300Kb.*
* *time = sum(20% of fastest round trips)*

Conclusion

- I had a lot of fun
- You might benefit from binary encoding when dealing with Web Workers.
- WASM understands only numbers and binary data (at least today)
- Avoid encoding strings into utf-8. It is slow
- Look up SharedArrayBuffer 👽

Thanks!

Github: twop

Repo: https://github.com/twop/ts-binary-types-workers-demo

Demo: https://bin-demo.twop.now.sh/

Twitter: @twopSK

Demo time! 😱