How to write a Node.js package in Rust?

Rust on Web @ BeCode

https://functional.cafe/@yvan

⚠️ The NPM modules we will talk about can't run in the browser!

(this is not WASM magic)

Why?!

Why?!

- You want **to mix languages in your codebase**, e.g. using Rust for safety or performance critical code and JS for everything else (because, e.g., you have a team of 10 developers that know JS but only 2 know Rust) ;

Why?!

- You want **to mix languages in your codebase**, e.g. using Rust for safety or performance critical code and JS for everything else (because, e.g., you have a team of 10 developers that know JS but only 2 know Rust) ;
- You want **to rely on a legacy library that can’t compile to WASM** and you don’t want to rewrite it from scratch (because, e.g., its code have been proven or audited) ;

NEON

Electrify your Node with the power of Rust!

```
// JavaScript
function hello() {
  let result = fibonacci(10000);
  console.log(result);
  return result;
}
```

```
// Neon
fn hello(mut cx: FunctionContext) -> JsResult<JsNumber> {
  let result = fibonacci(10000);
  println!("{}", result);
  Ok(cx.number(result))
}
```

GET STARTED

Simple tooling.

No build scripts. No finicky system dependencies. Just Node and Rust.

Guaranteed safety.

If a Neon module compiles, it is guaranteed by the Rust compiler to be memory-safe.

Easy parallelism.

Safely run multiple threads—without data races.

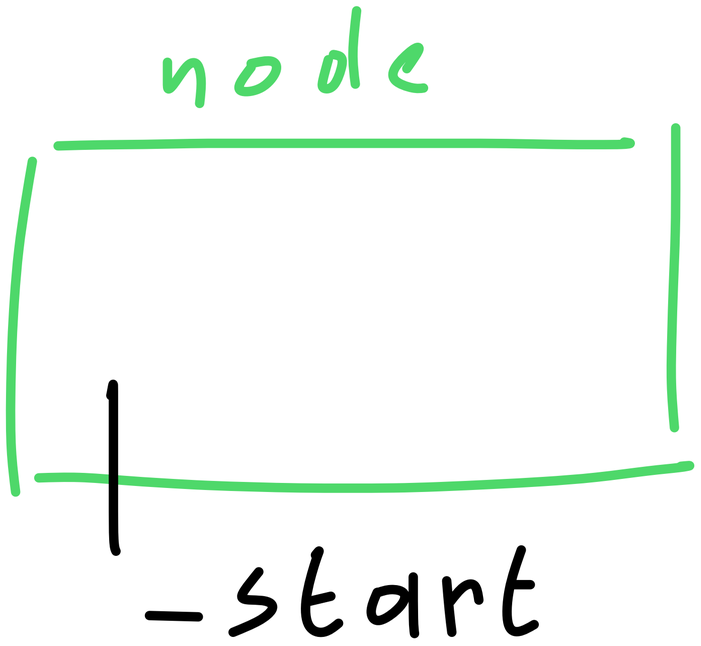
This is a talk about ... FFI!

(Foreign Function Interface)

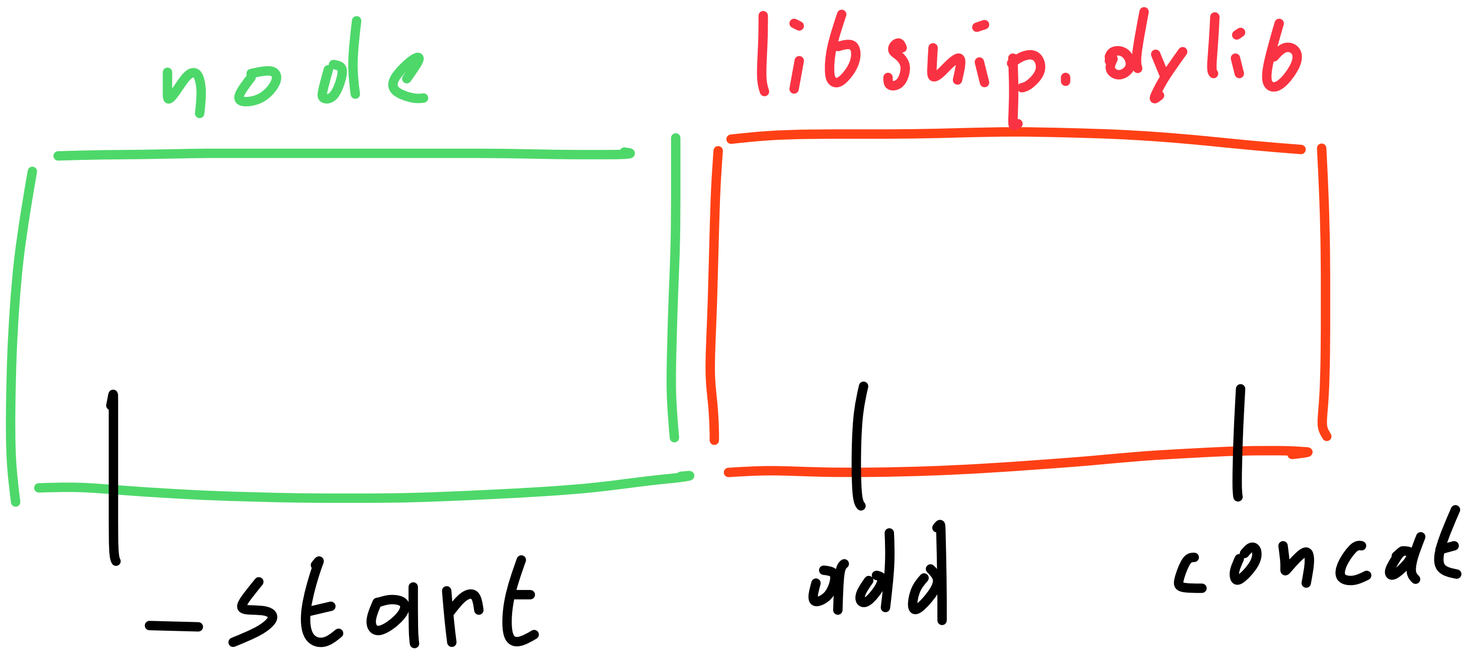
Demo !

MEMORY of a PROCESS

MEMORY of a PROCESS

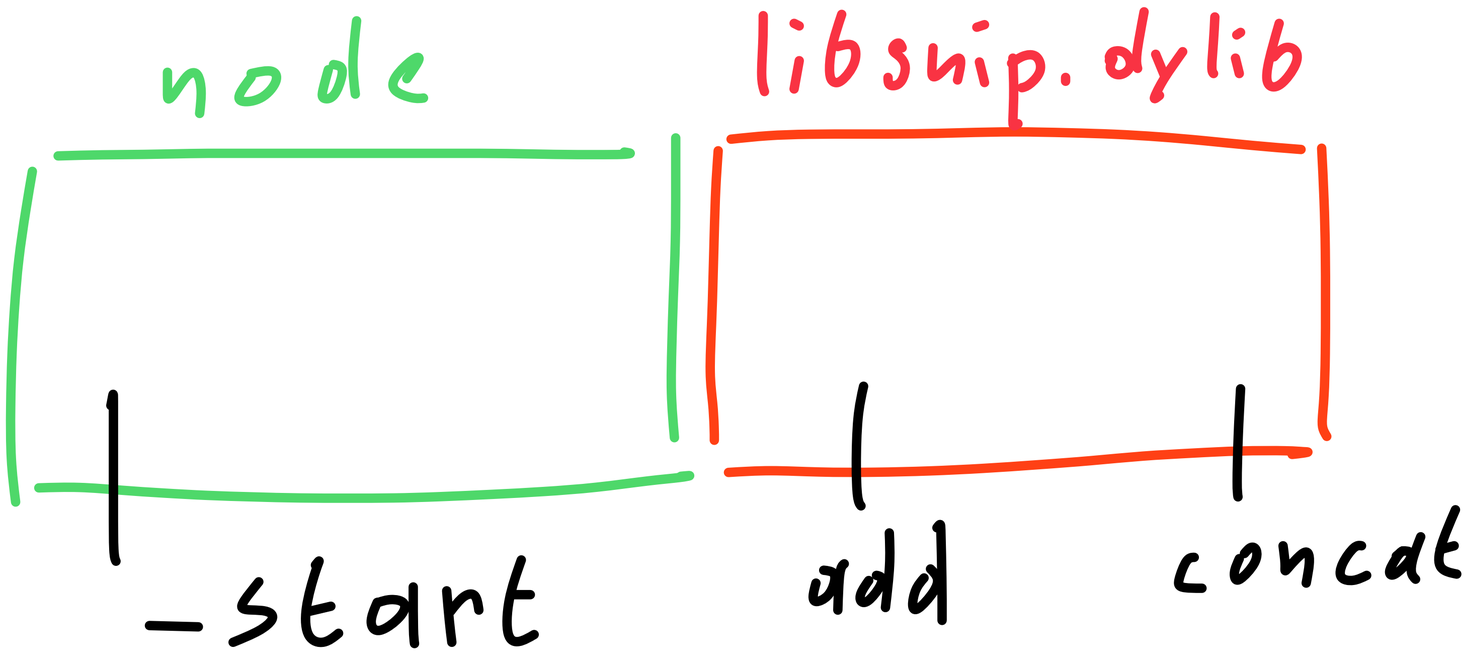


MEMORY of a PROCESS

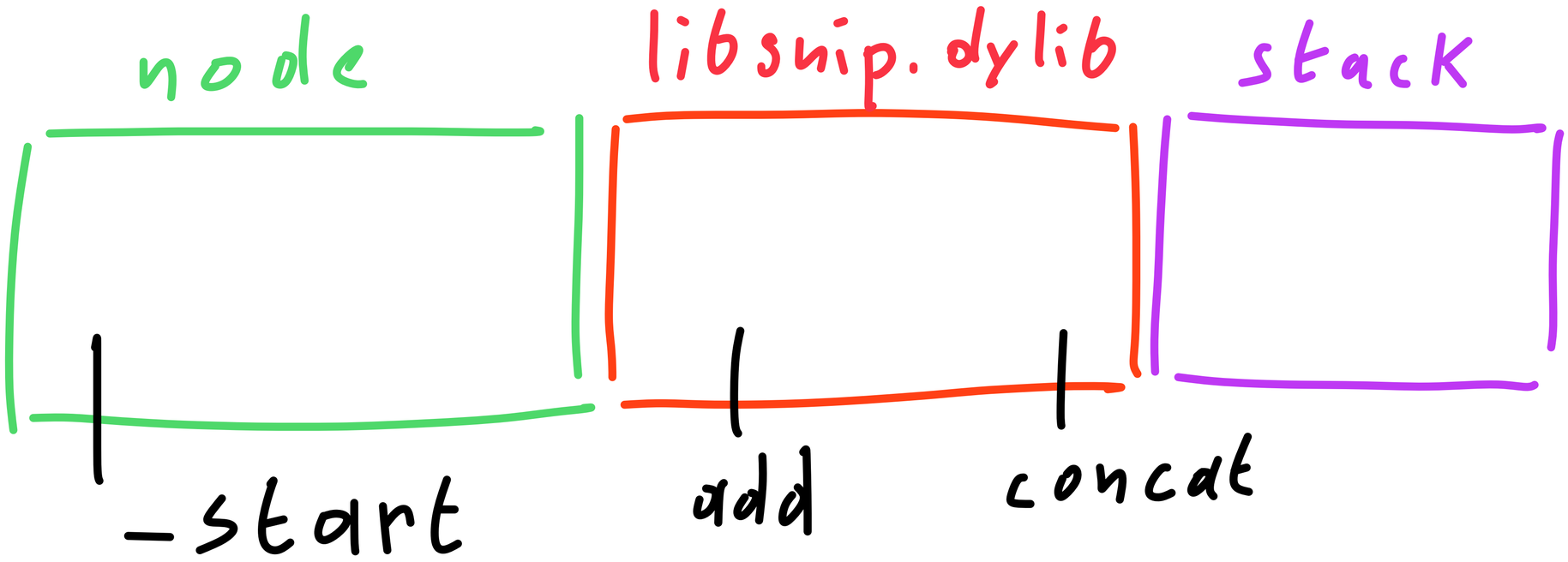


Demo !

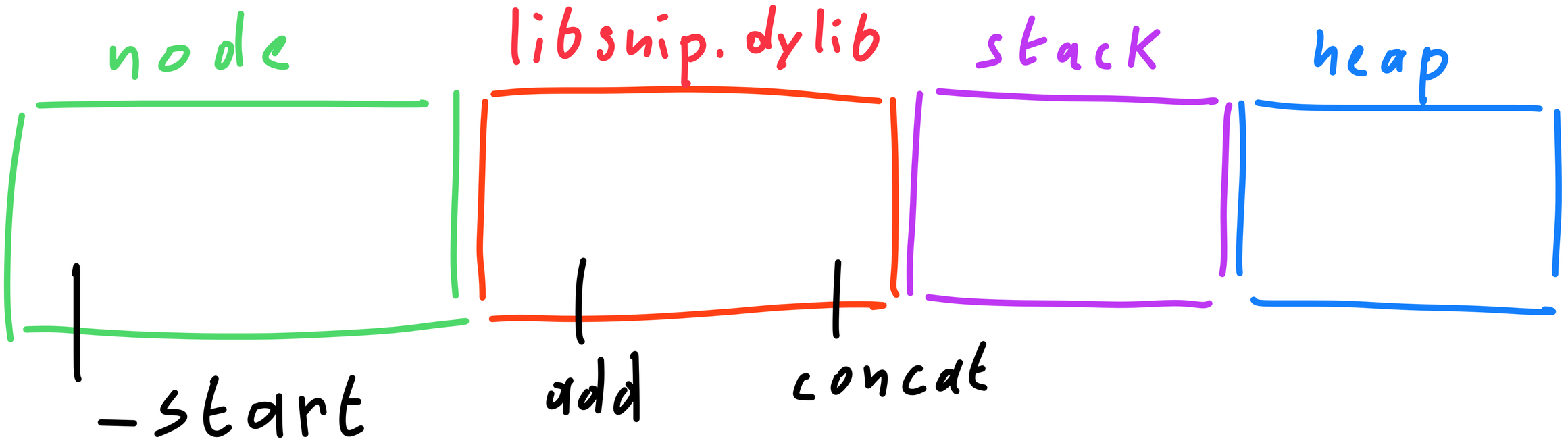
MEMORY of a PROCESS



MEMORY of a PROCESS



MEMORY of a PROCESS

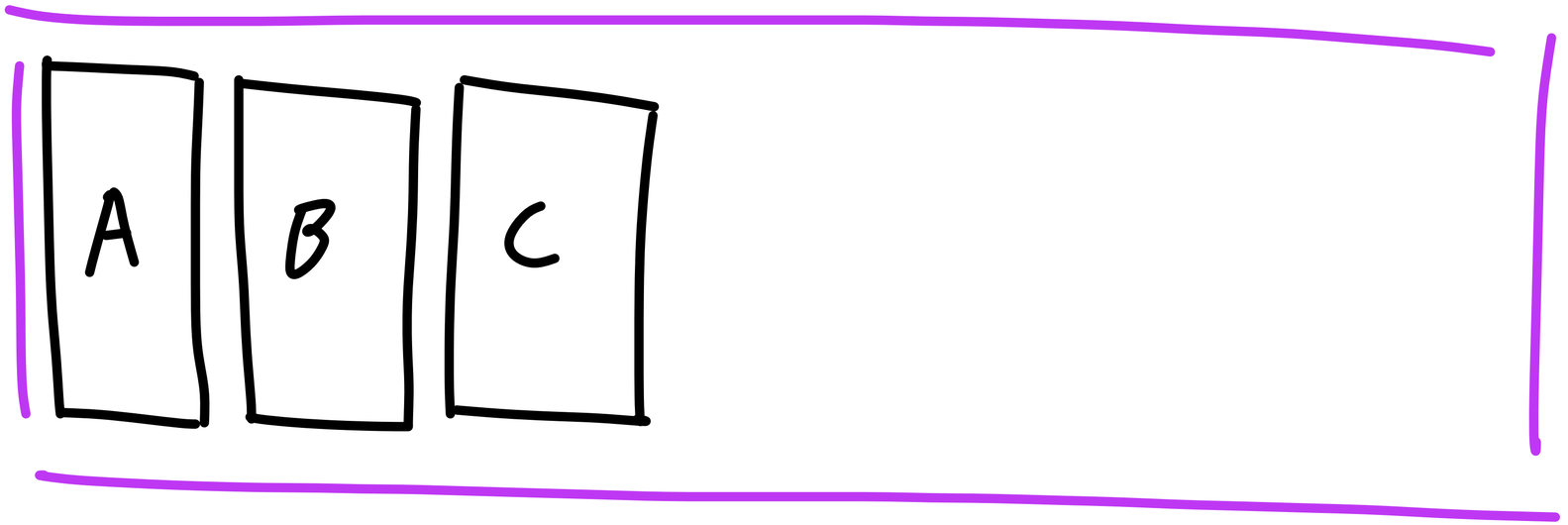


STACK

STACK

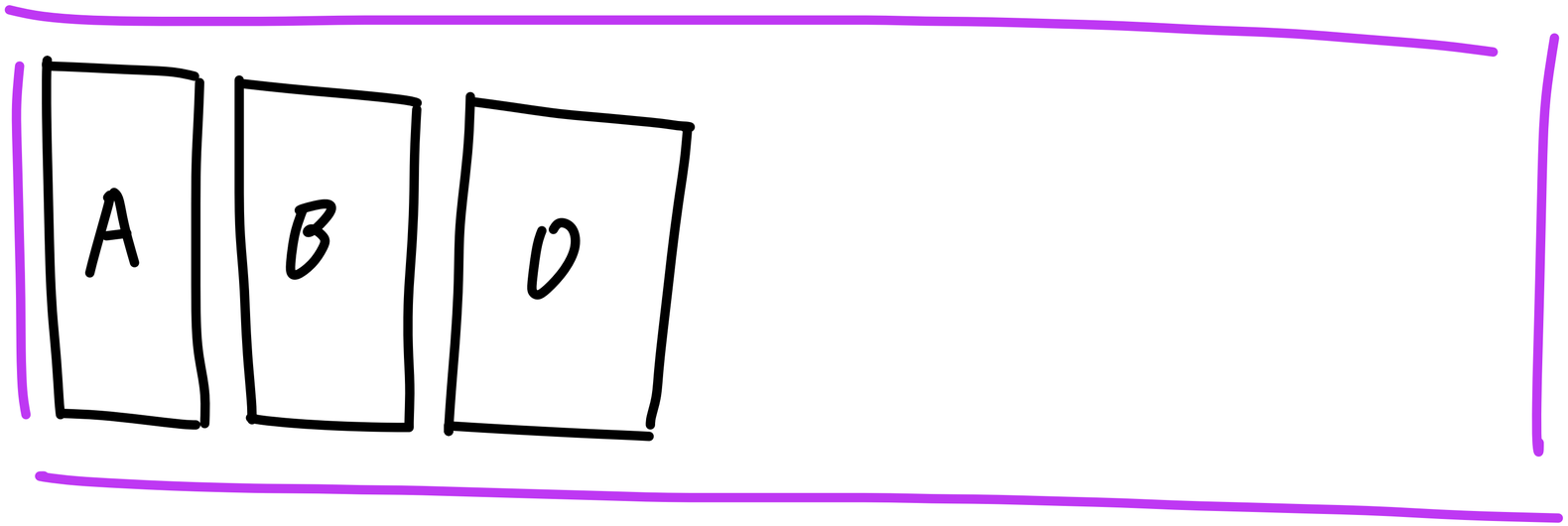
STACK

STACK

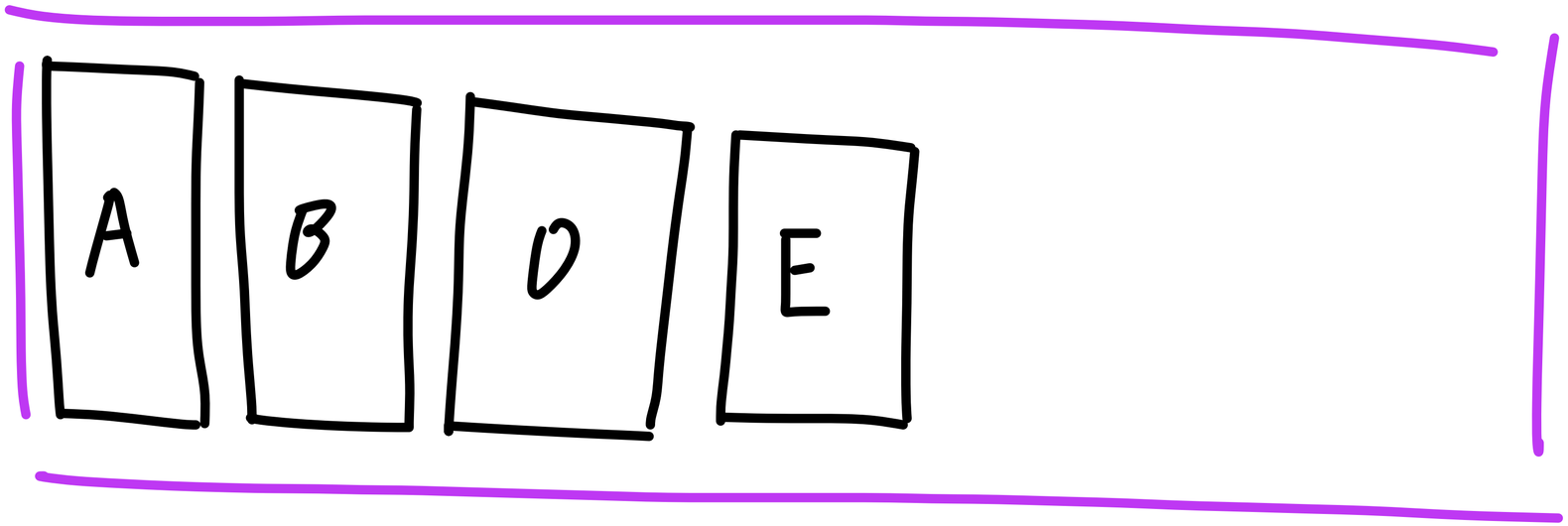


STACK

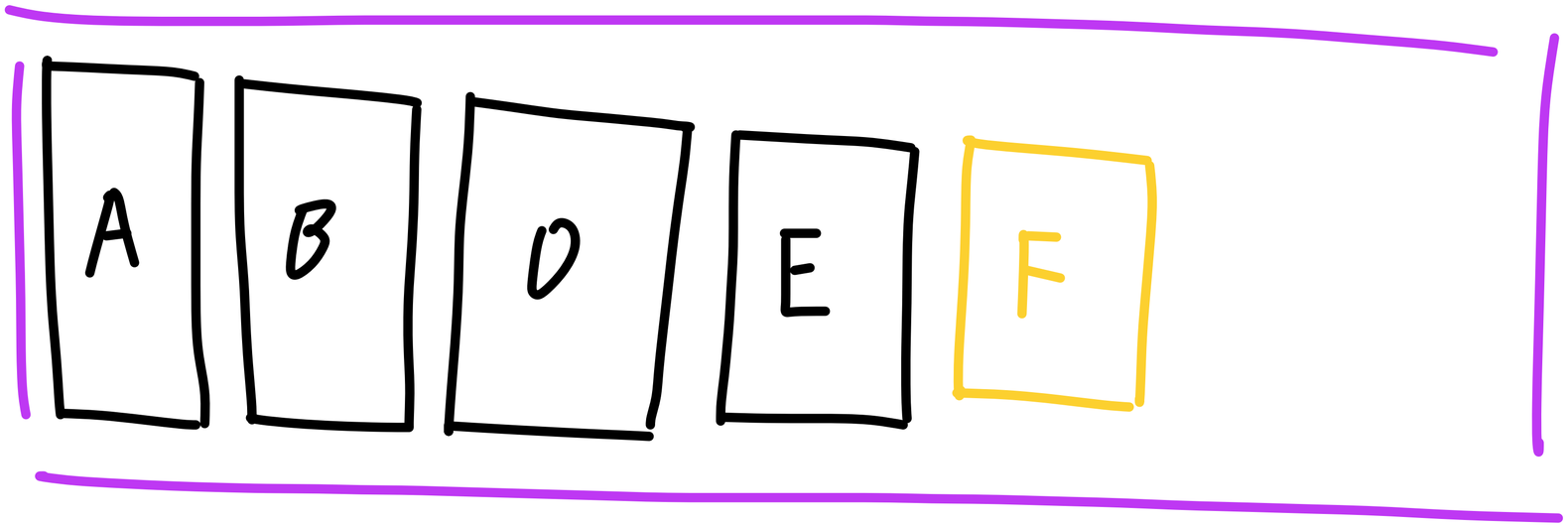
STACK



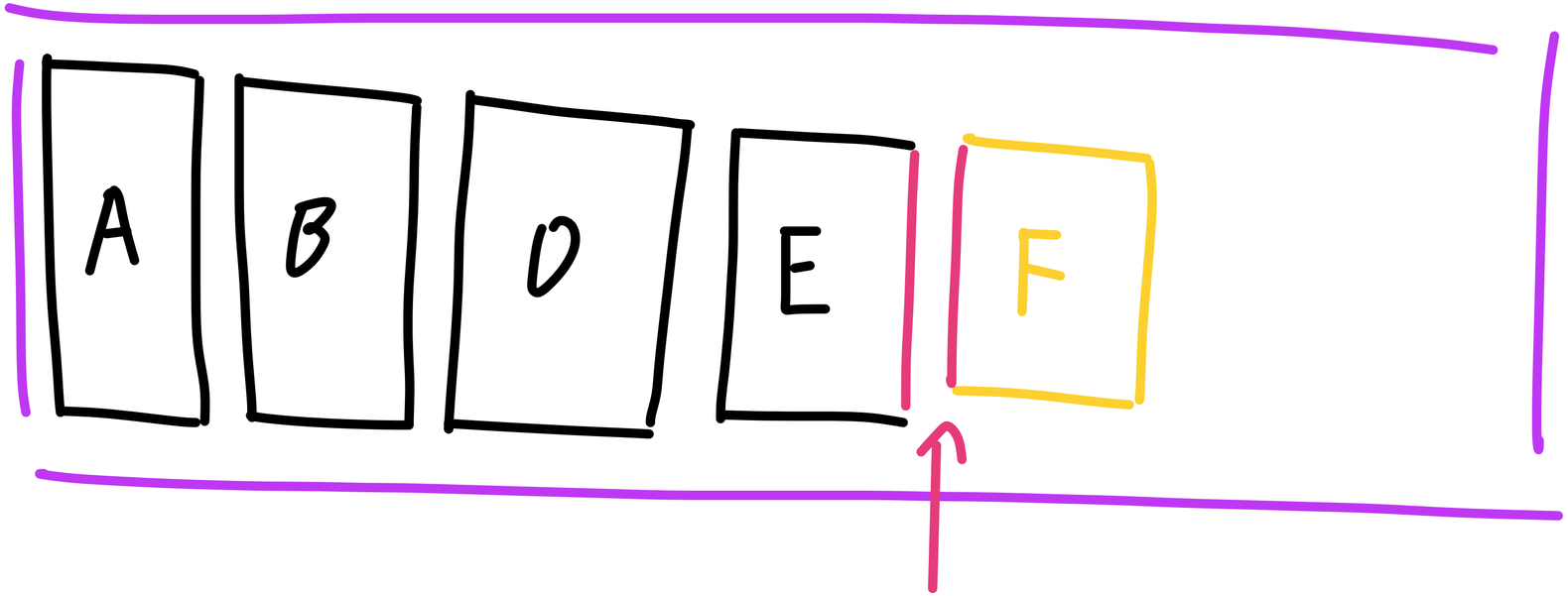
STACK



STACK



STACK



Functions should share the same CALLING CONVENTIONS

But why does it works with `int` then?

By chance, it’s a **Hack**!

But why does it works with `int` then?

By chance, it's a **Hack**!

`ffi-napi` expect **C calling conventions**,
we used **Rust calling conventions** in `libsnip`,
but **Rust** `i32` happen to use the same memory layout than **C** `int` ...

This does not hold for **Rust** `String` that are really different than **C** `char*` ...

Demo !

This bug is called an “use after free”

We read memory that have been freed and display garbage to the user!

Demo !

FFI are so complicated … why bother with that?!

Let’s just make Rust and JS communicate through an HTTP REST API!

FFI are so complicated … why bother with that?!

Let’s just make Rust and JS communicate through an HTTP REST API!

I want FAST interoperability!

You have to avoid I/O (OS roundtrip)

I want FAST interoperability!

You have to avoid I/O (OS roundtrip)

NO HTTP!

I want SAFE interoperability!

Something easy to use that will not make my program CRASH by mistake ...

A good FFI is a FFI you don’t write …

“Bindgen” (bindings code generation) to the rescue!

How do we implement Rust code generation?

How do we implement Rust code generation?

`Macros!`

Demo !

https://github.com/yvan-sraka/snip

Questions?

Thank you!

https://functional.cafe/@yvan