WASMCON

BETTER TOGETHER

Components for Every Language

Kyle Brown, SingleStore

#WasmCon

WASMCON

BETTER TOGETHER

Every language can be made into a Component

WASMCON

BETTER TOGETHER

Components in different languages work together

Overview

- Components
- Componentizing
  - Components by Hand
  - WIT-Bindgen Method (Rust, C, C++)
  - Runtime Wrapping (JavaScript, Python)
  - Future Opportunities
- Composing
  - How do Components Compose?
  - Demo: Tower of Wasm

WASMCON

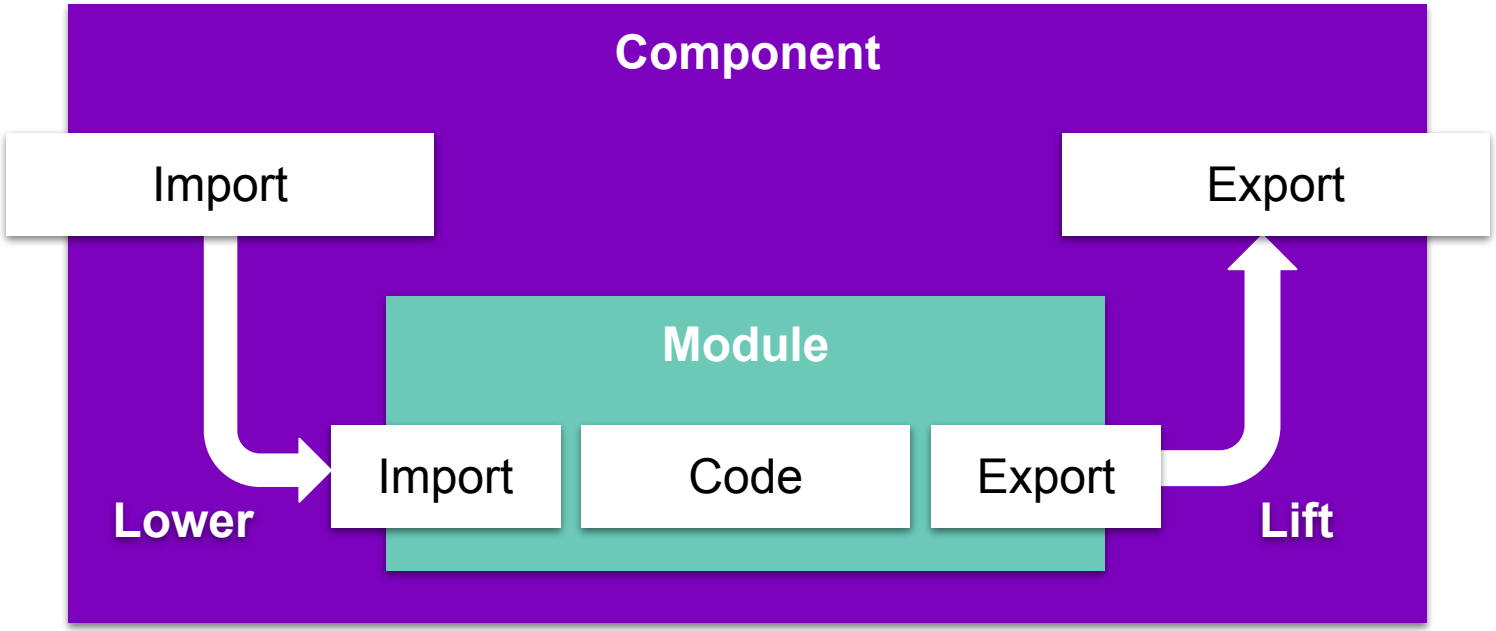
BETTER TOGETHER

Components

What is the Component Model?

	Defines	Abstraction Level
Component Model	Components	Lists, Records, Strings, …
Core WebAssembly	Modules	Numbers, Memory, …

Anatomy of a Component



Inside a Component - Import Function "foo"

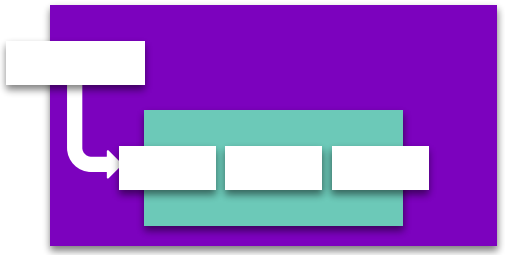
```
(component
   (import "foo" $c-import (func ...))
   ...
```

Inside a Component - Lower Import

```
(component
   (import "foo" $c-import (func ...))
   (canon lower $c-import ...opts...
      (core func $m-import))
   ...
```

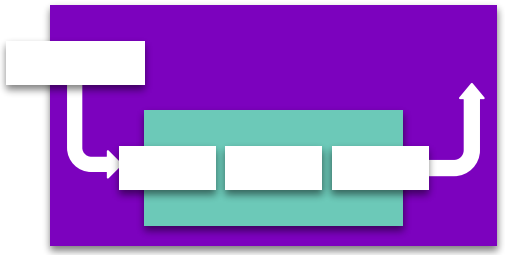
Inside a Component - Define & Instantiate Module

```
(component
   (import "foo" $c-import (func ...))
   (canon lower $c-import ...opts...
      (core func $m-import))
   (core module $m
      ...full module bytes...)
   (core instance $m (instantiate $M
         (with "foo" (core func $m-import))))
   ...
```



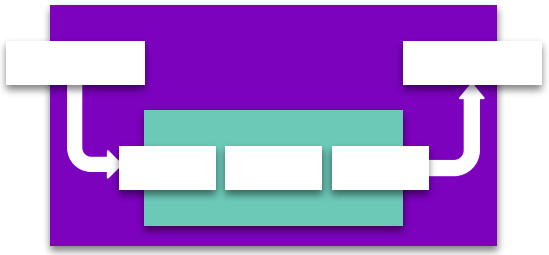
Inside a Component - Lift Export

```
(component
   (import "foo" $c-import (func …))
   (canon lower $c-import …opts…
      (core func $m-import))
   (core module $m
      …full module bytes…)
   (core instance $m (instantiate $M
         (with "foo" (core func $m-import))))
   (canon lift
      (core func $M "bar") …opts…
      (func $c-export …))
   …
```



Inside a Component - Export Function "bar"

```
(component
   (import "foo" $c-import (func ...))
   (canon lower $c-import ...opts...
      (core func $m-import))
   (core module $m
      ...full module bytes...)
   (core instance $m (instantiate $M
         (with "foo" (core func $m-import))))
   (canon lift
      (core func $M "bar") ...opts...
      (func $c-export ...))
   (export "bar" (func $c-export))
)
```




WASMCON
BETTER TOGETHER
Componentizing

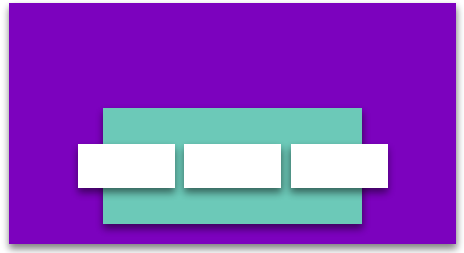
WASMCON

BETTER TOGETHER

Let's write a Component by hand

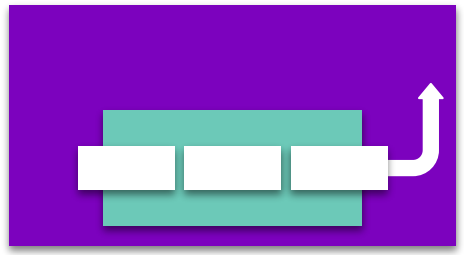
WebAssembly Text Format (WAT)

```
(component
   (core module $m
      (memory $memory (export "mem")
         (data "\08\00\00\00" "\15\00\00\00" "Hello from WAT!")
      )
      (func $m-greet (result i32)
         i32.const 0
      )
      (export "greet" (func $m-greet))
   )
   (core instance $M (instantiate $m))
   (alias core export $M "mem" (core memory $c-mem))
   …
```



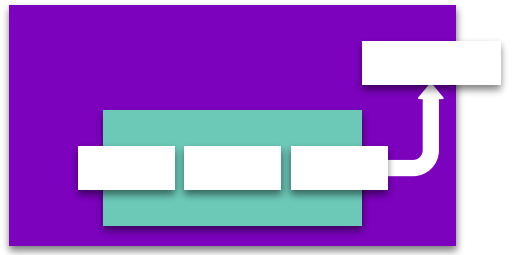
WebAssembly Text Format (WAT)

```
(component
   (core module $m
      (memory $memory (export "mem")
         (data "\08\00\00\00" "\15\00\00\00" "Hello from WAT!")
      )
      (func $m-greet (result i32)
         i32.const 0
      )
      (export "greet" (func $m-greet))
   )
   (core instance $M (instantiate $m))
   (alias core export $M "mem" (core memory $c-mem))
   (func $c-greet (result string)
      (canon lift (core func $M "greet")
         string-encoding=utf8
         (memory $c-mem)
      )
   )
   ...
```



WebAssembly Text Format (WAT)

```
(component
   (core module $m
      (memory $memory (export "mem")
         (data "\08\00\00\00" "\15\00\00\00" "Hello from WAT!")
      )
      (func $m-greet (result i32)
         i32.const 0
      )
      (export "greet" (func $m-greet))
   )
   (core instance $M (instantiate $m))
   (alias core export $M "mem" (core memory $c-mem))
   (func $c-greet (result string)
      (canon lift (core func $M "greet")
         string-encoding=utf8
         (memory $c-mem)
      )
   )
   (instance (export (interface "wasmcon2023:greet/interface"))
      (export "greet" (func $c-greet))
   )
)
```



WASMCON

BETTER TOGETHER

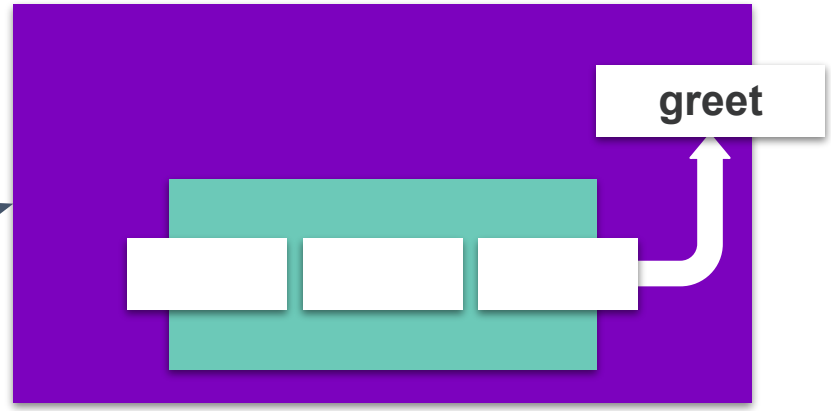
The Wit-Bindgen Method

Wasm Interface Text Format (WIT)

```
package wasmcon2023:greet

interface %interface {
    greet: func() -> string
}
```

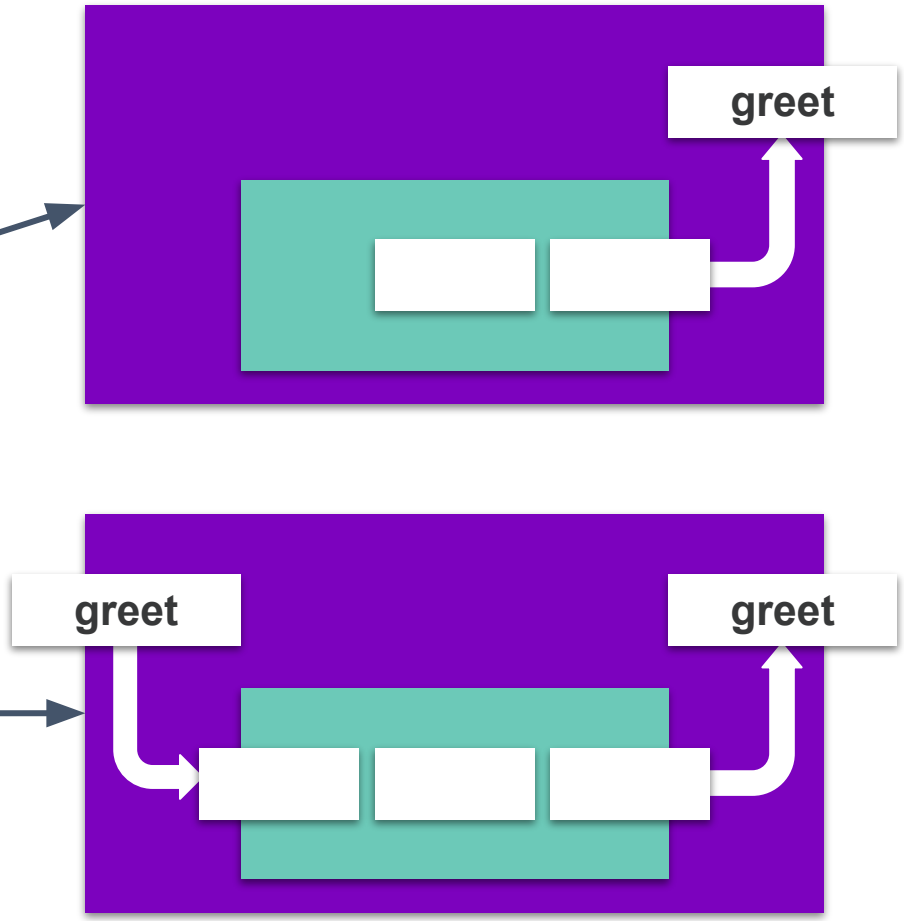
Wasm Interface Text Format (WIT)

```
package wasmcon2023:greet

interface %interface {
    greet: func() -> string
}

world greeter {
    export %interface
}
```

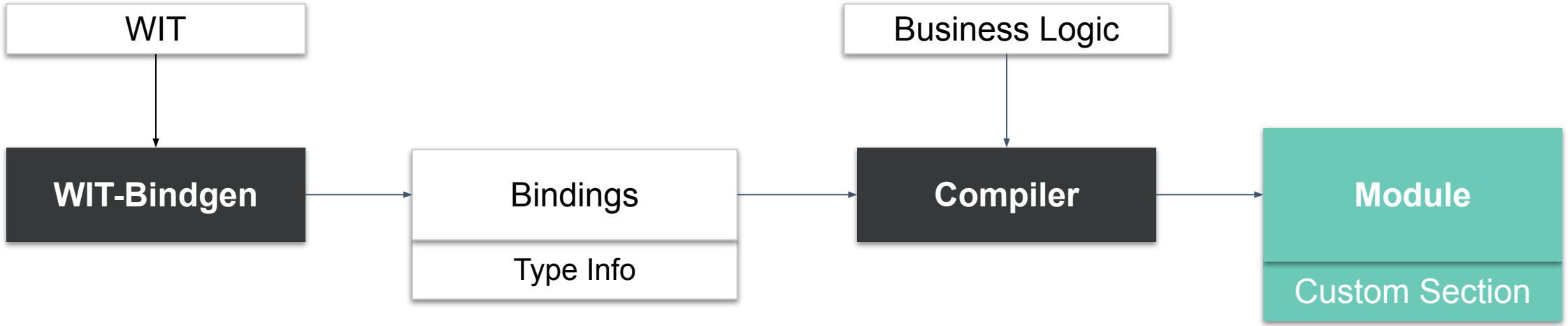


Wasm Interface Text Format (WIT)

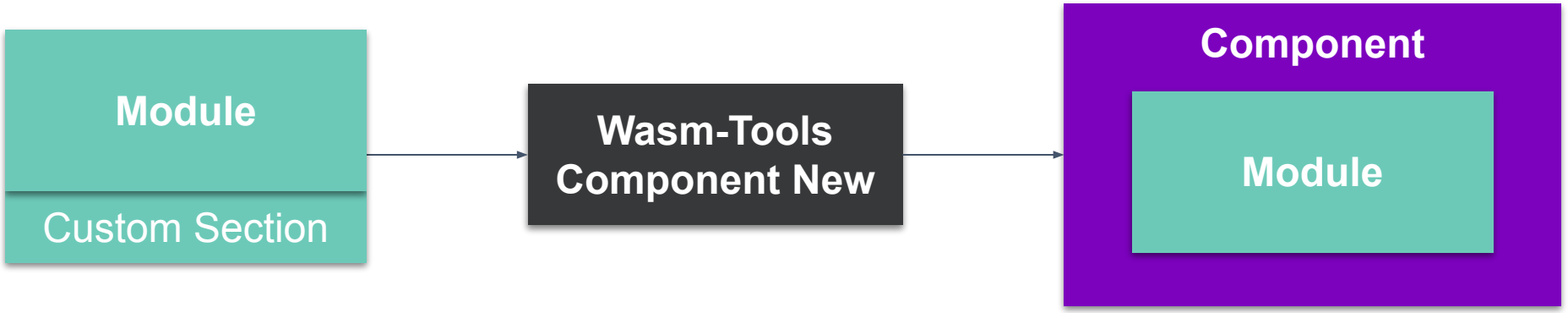
```
package wasmcon2023:greet

interface %interface {
    greet: func() -> string
}

world greeter {
    export %interface
}

world proxy-greeter {
    export %interface
    import %interface
}
```



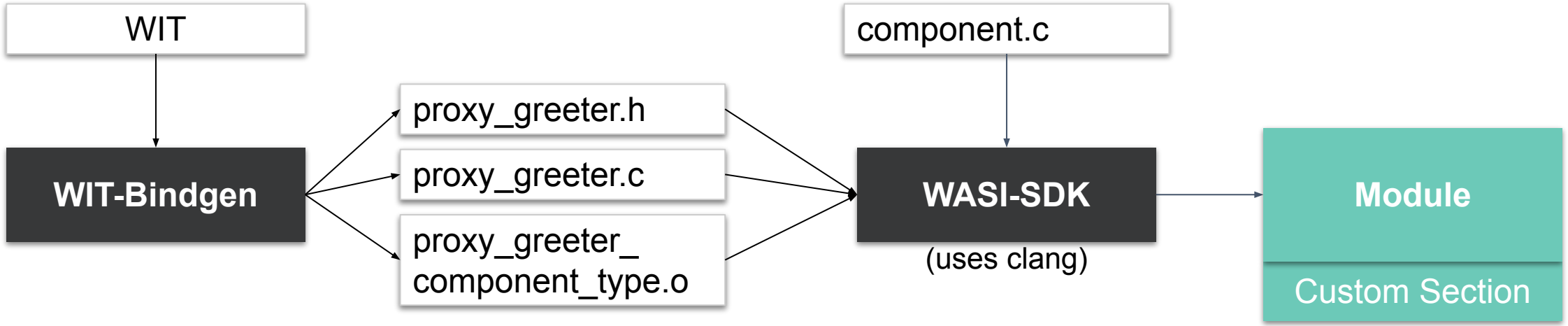
Make a Module



Turn it Into a Component



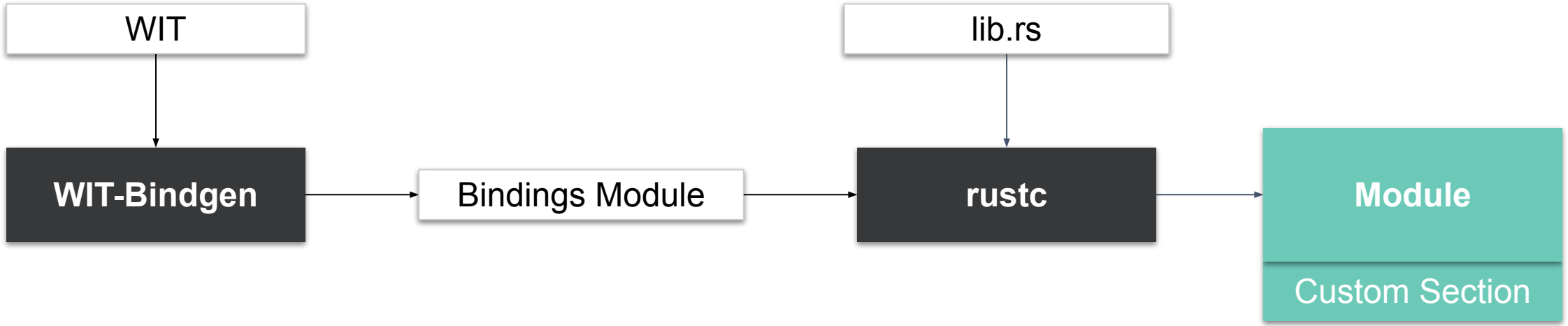
Making a C (or C++) Component



Component.c

```c
#include "proxy_greeter.h"
#include <stdio.h>
#include <stdlib.h>

void exports_wasmcon2023_greet_interface_greet(proxy_greeter_string_t *ret) {
    proxy_greeter_string_t greeting;
    wasmcon2023_greet_interface_greet(&greeting);
    char* suffix = " and C!";
    size_t suffix_len = strlen(suffix);

    ret->len = greeting.len + suffix_len;
    ret->ptr = malloc(ret->len);

    memcpy(ret->ptr, greeting.ptr, greeting.len);
    memcpy(ret->ptr + greeting.len, suffix, suffix_len);

    proxy_greeter_string_free(&greeting);
}
```

Making a Rust Component (inside Cargo Component)



lib.rs

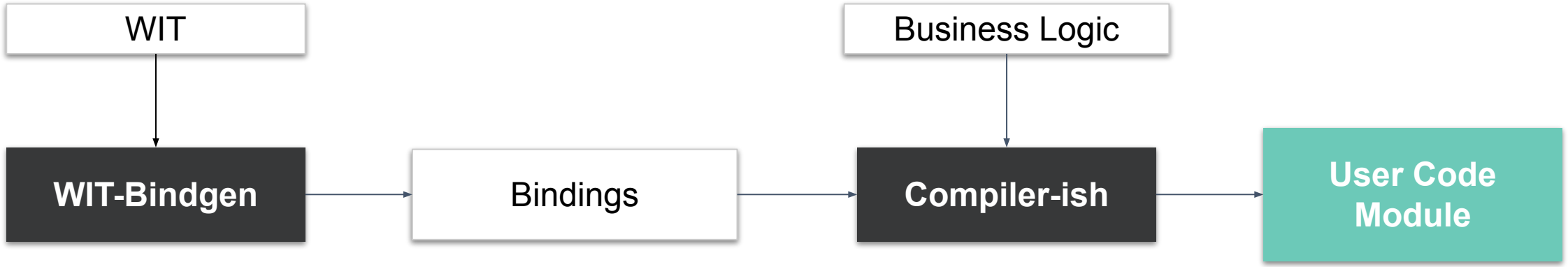
```rust
cargo_component_bindings::generate!();

use bindings::{
    wasmcon2023::greet::interface as import,
    exports::wasmcon2023::greet::interface::Guest
};

struct Component;

impl Guest for Component {
    fn greet() -> String {
        import::greet() + " and Rust 🦀!"
    }
}
```

WASMCON

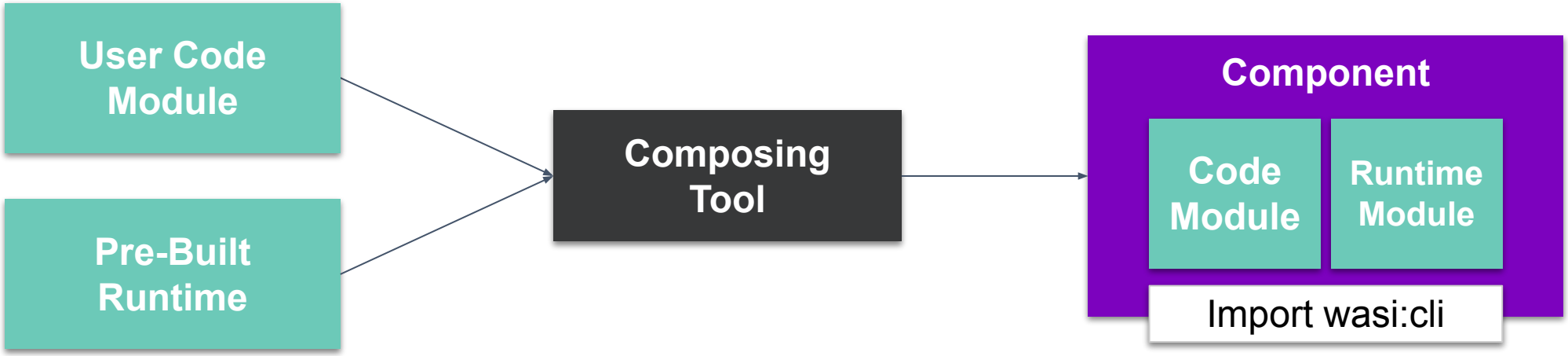
BETTER TOGETHER

Runtime Wrapping

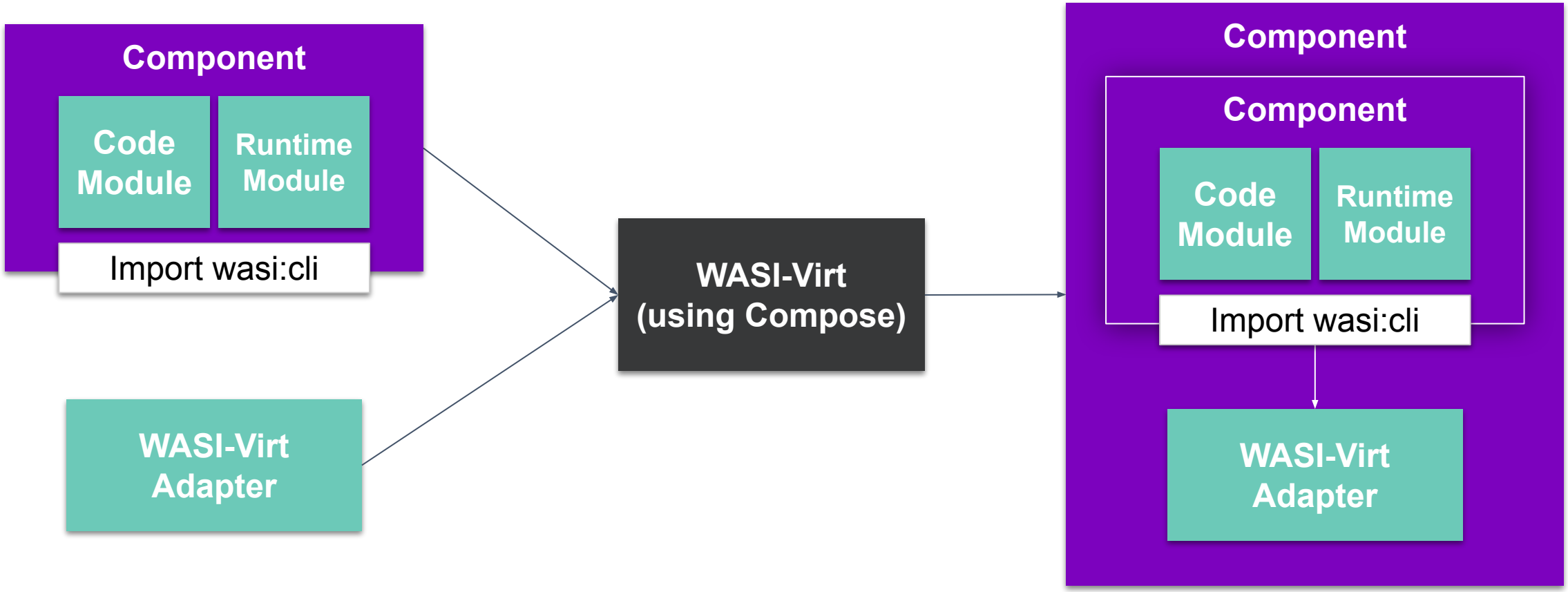
Make a Module



Link it with the Runtime



Virtualize Imports if Needed



Componentize-Py

- Uses pre-built CPython Runtime
- Built using PyO3
- C-Extension Linking
  - Pseudo-Dynamic Linking approach allows C-Extensions compiled to Wasm to be linked into the Component.
  - Important for popular libraries like Numpy, Pandas, SciKit, etc.

Check out Joel Dice's talk on it tomorrow!

greet.py

```python
from proxy_greeter import imports, exports
from proxy_greeter.imports import interface

class Interface(exports.Interface):
    def greet(self) -> str:
        return interface.greet() + " and Python 🐍"
```

Componentize-JS

- Uses pre-built SpiderMonkey Runtime
  - Designed to be instrumented & have arbitrary bindings added
- Allows users to configure…
  - the globals and imports available,
  - the flavor of JS (e.g. whether it's like Deno or Node),
  - prelude scripts that set up the environment.
- Uses snapshotting to improve startup speed
  - Componentize-JS runs your code Ahead-of-Time
  - Execution is completely sandboxed
  - Engine memory is snapshotted and embedded in your component

greet.js

```
import { greet } from 'wasmcon2023:greet/interface';

const greetInterface = {
    greet() {
        return greet() + " and JavaScript!";
    }
};

export { greetInterface as 'interface' }
```

WASMCON

BETTER TOGETHER

The WIT-Bindgen approach & Runtime Wrapping let us Componentize anything

WASMCON

BETTER TOGETHER

Future Opportunities

Garbage Collection Proposal (Wasm-GC)

- GC Languages can be implemented using Linear Memory.
- Challenges with GC
  - Wasm-GC doesn't build-in all language's GC features.
  - GC is usually tightly integrated in runtimes.
  - Wasm-GC support is still WIP outside the browser.
  - Component Model doesn't yet support Wasm-GC at boundary.
- In the future, Wasm-GC *could* be useful to some projects & languages
  - Most plausible for new languages or languages with small runtimes.

Deeper Toolchain Integration

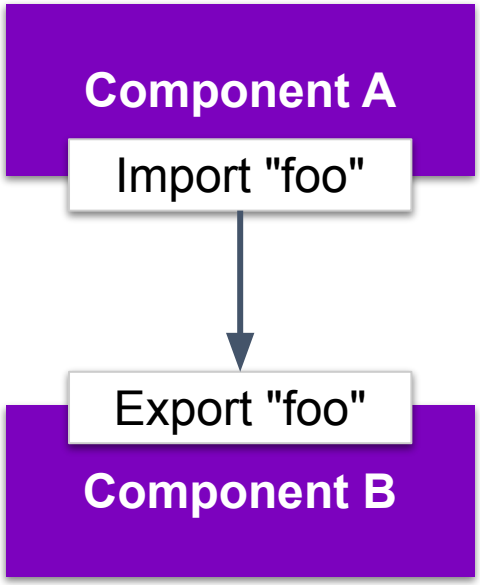
- Compilers can start to directly understand **WIT** & **Components**
  - Map Component types to source code imports directly.
  - Generate lifts & lowers without needing bindings generation.
- Cleaner more integrated compiler UX is possible.
- As async features are added, compilers can integrate them.

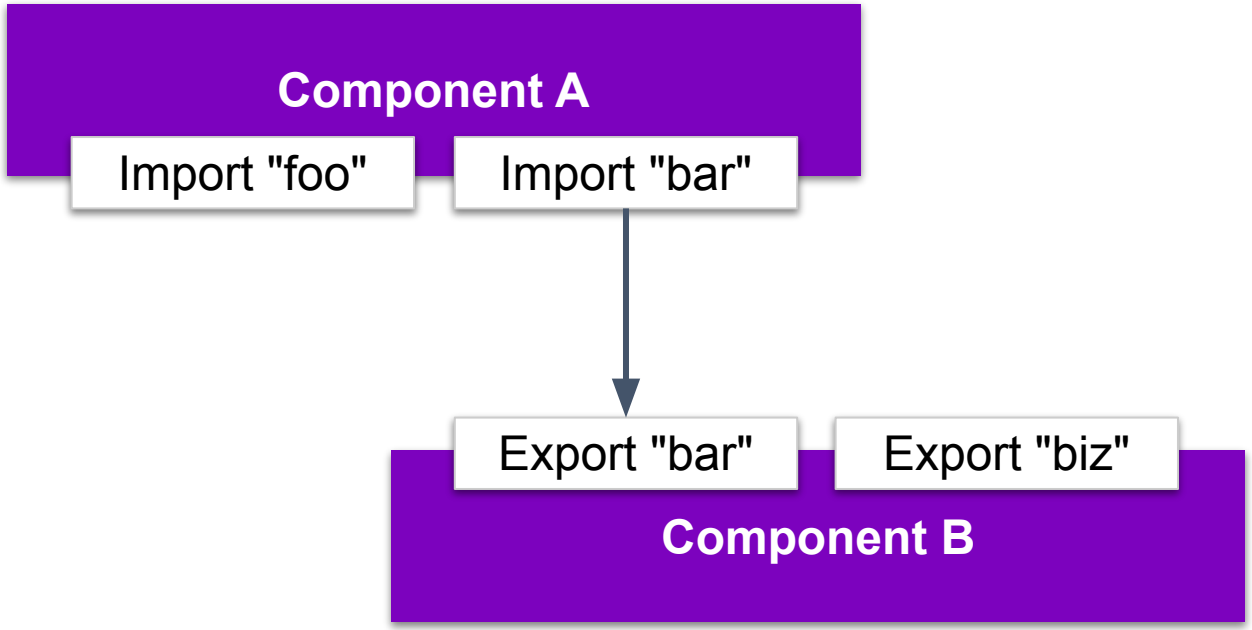
WASMCON

BETTER TOGETHER

Composing

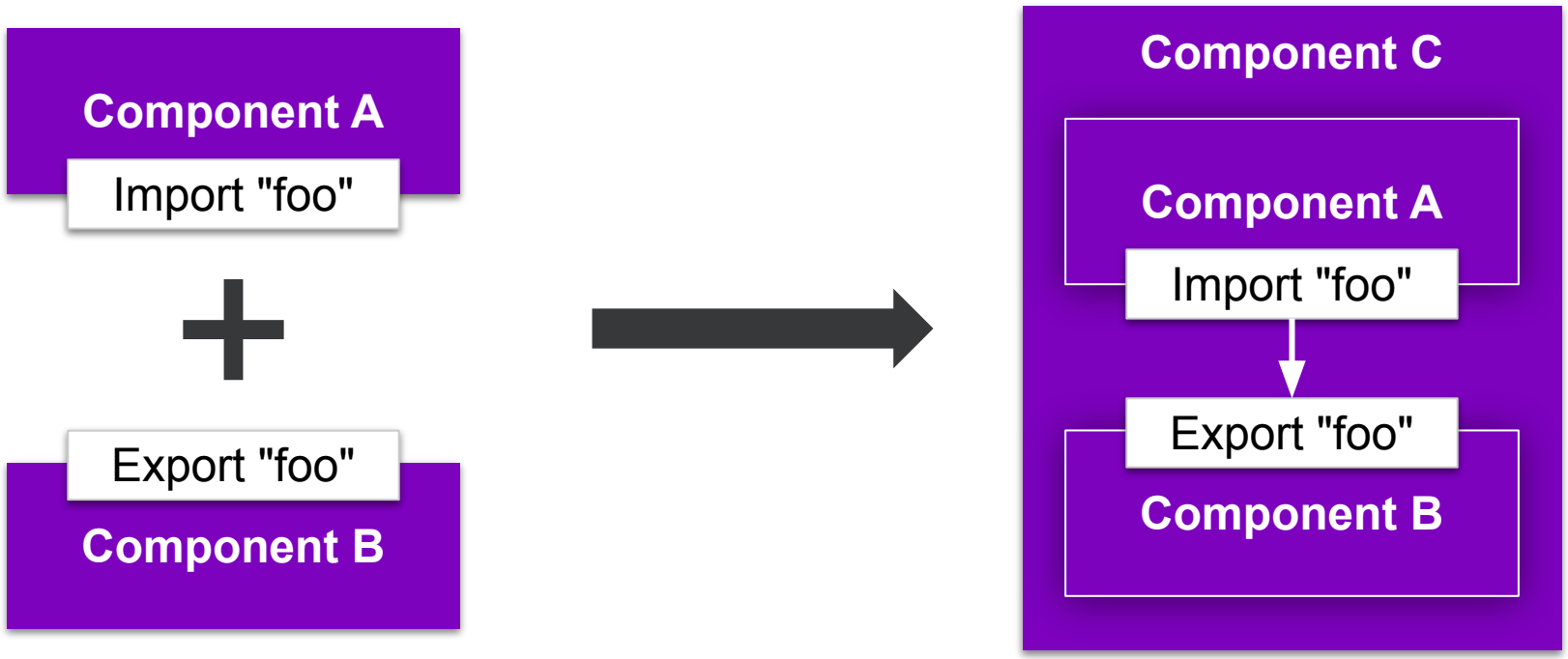
Composition





Static Composition

Creating a new Component containing two or more inner linked Components

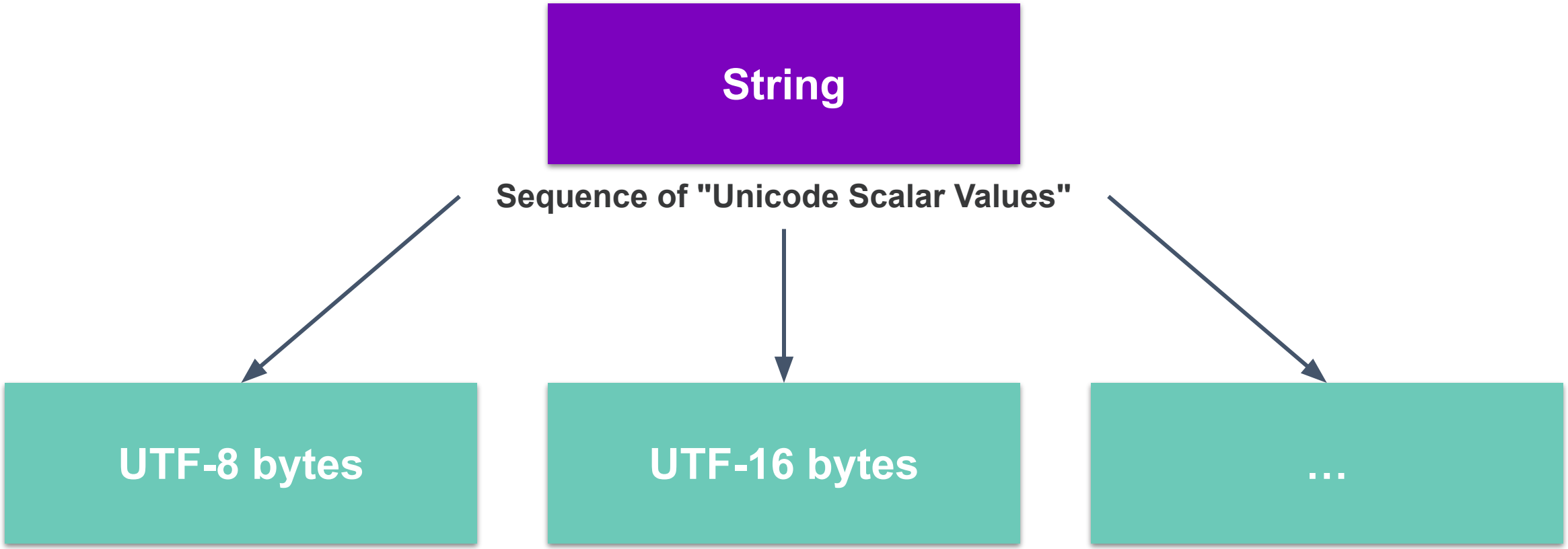


WASMCON

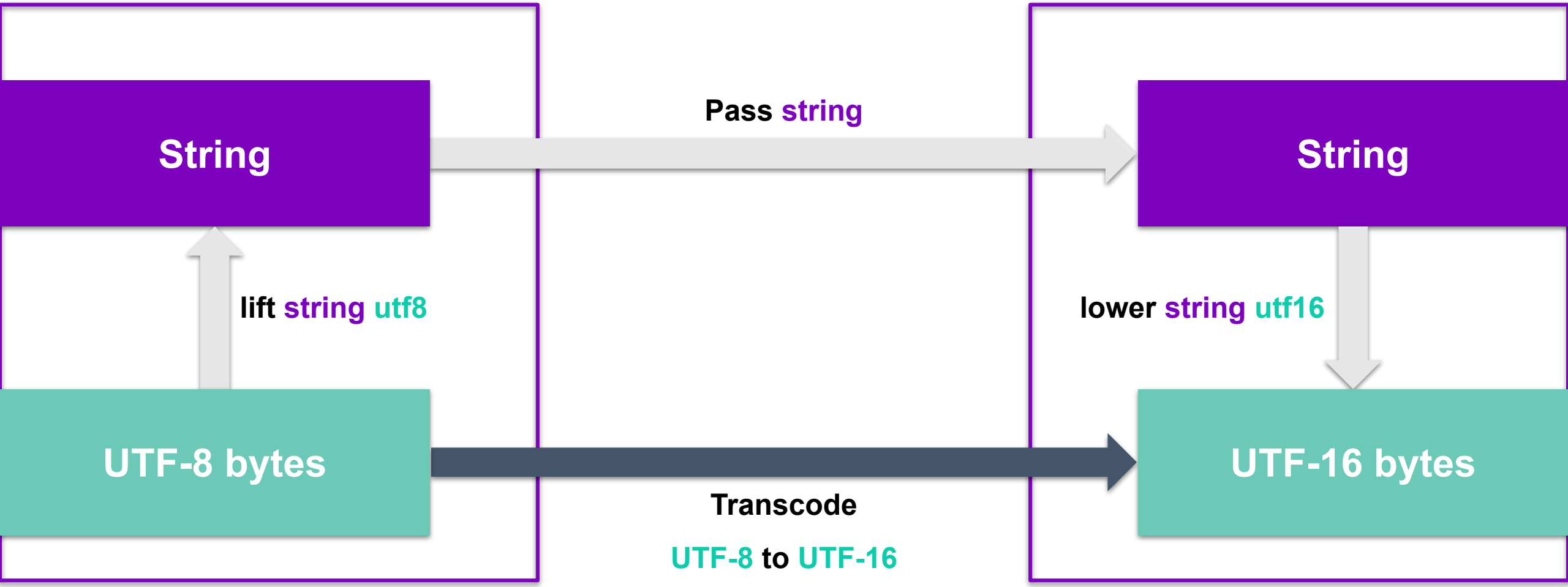
BETTER TOGETHER

Lifting & Lowering Between Components

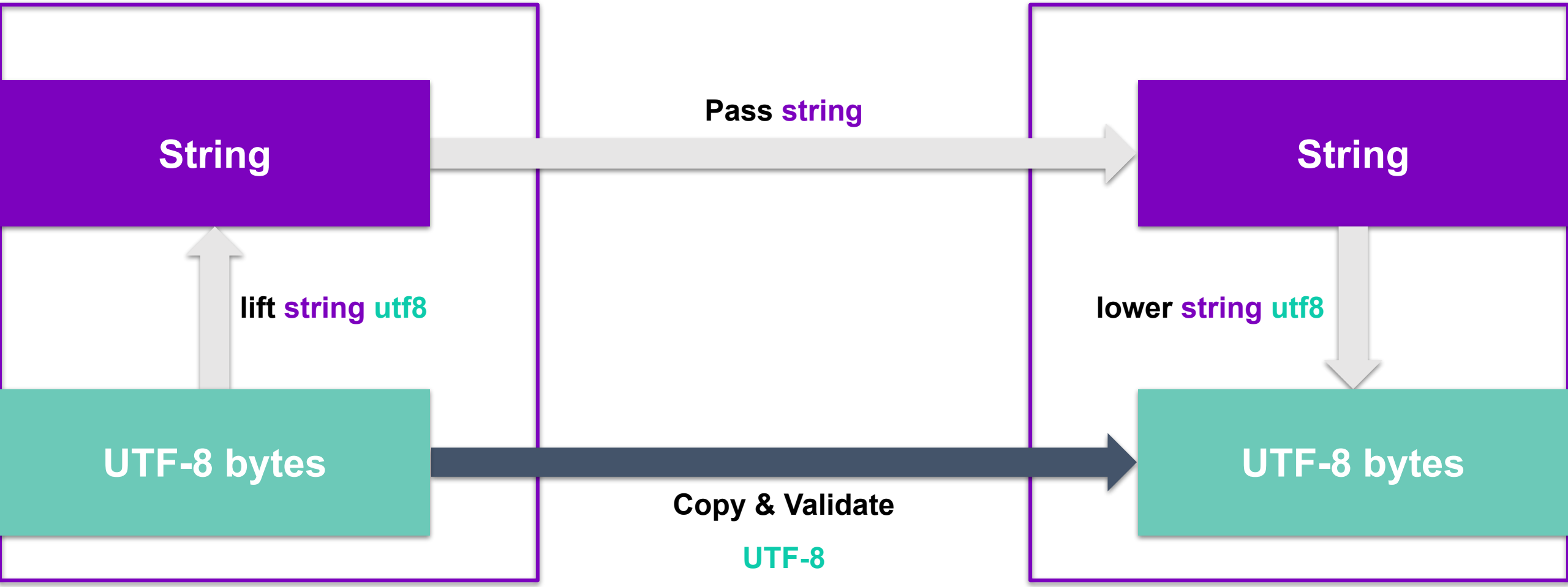
What is a String?



Lifting & Lowering Strings



Lifting & Lowering Strings



WASMCON

BETTER TOGETHER

Demo: Tower of Wasm

Acknowledgements

- **SingleStore** for supporting my work in the Bytecode Alliance
- Everyone who gave input on or reviewed these slides including…
  - Luke Wagner
  - Dan Gohman
  - Guy Bedford
  - Peter Huene
  - Joel Dice
  - Bailey Hayes
  - Daniel Macovei


WASMCON
BETTER TOGETHER