Writing an adaptive Stack in JavaScript or Rust

Using a language neutral API via WASI 0.2

Christof Petig

11 Jun 2024

15th AUTOSAR Open Conference

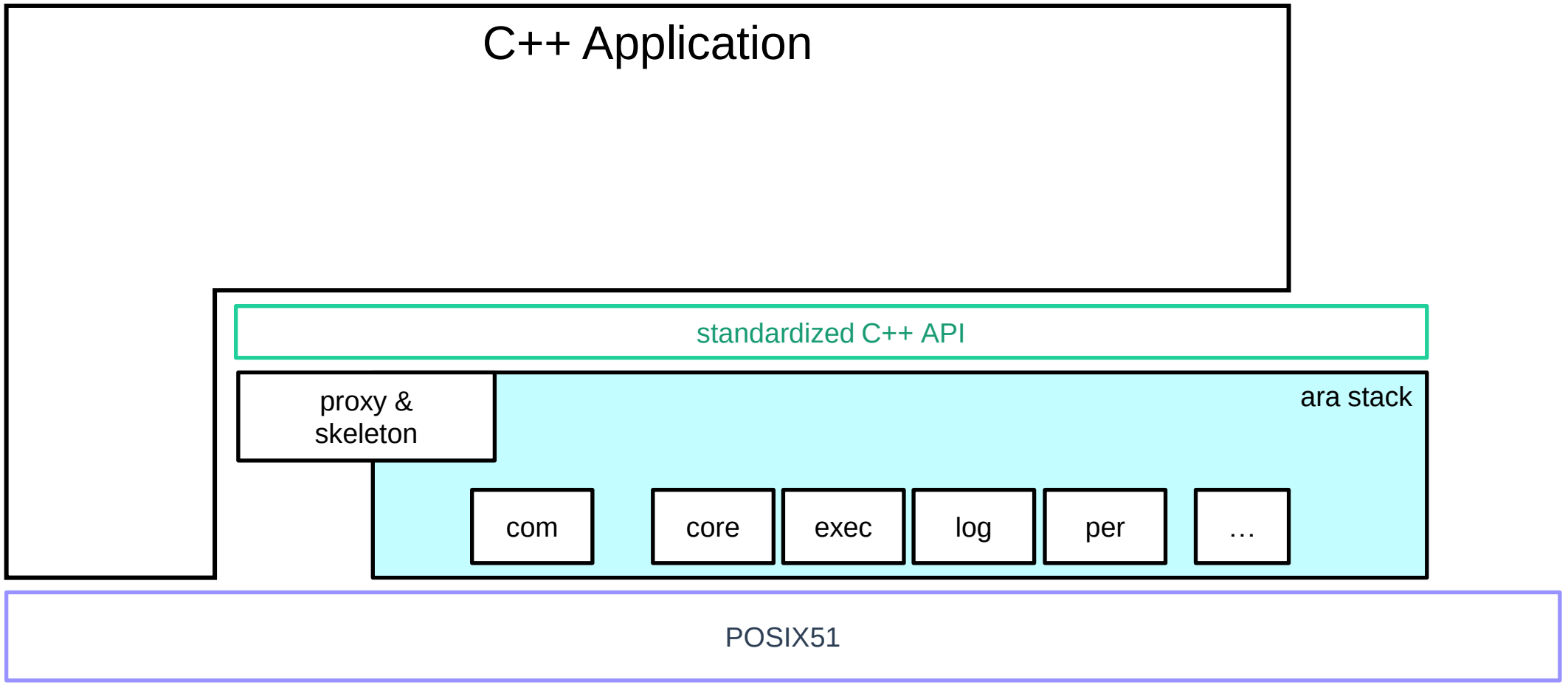
Hilton Tokyo Odaiba

Overview of this talk

- Motivation: Rust Applications for Adaptive Platform
- Solution: Language Neutral Binary Interface
- Practical: Running an AP Application in a Browser
- Practical: With a Rust Stack and deploy for Embedded
- Outlook: Future Optimizations

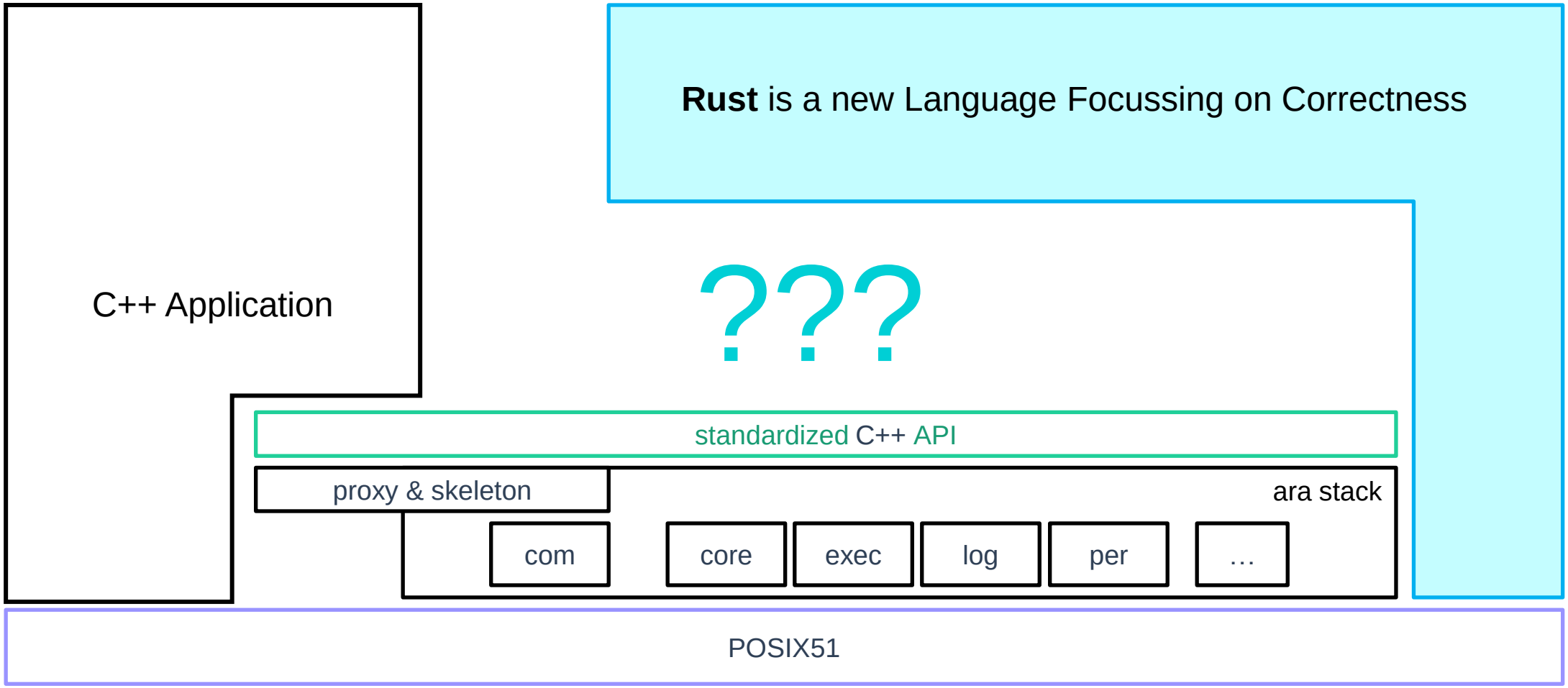
AUTOSAR Adaptive Platform Stack

The classical situation for C++ Applications



AUTOSAR Adaptive Platform Stack

But how to write a Rust Application?



Combining C++ with other languages

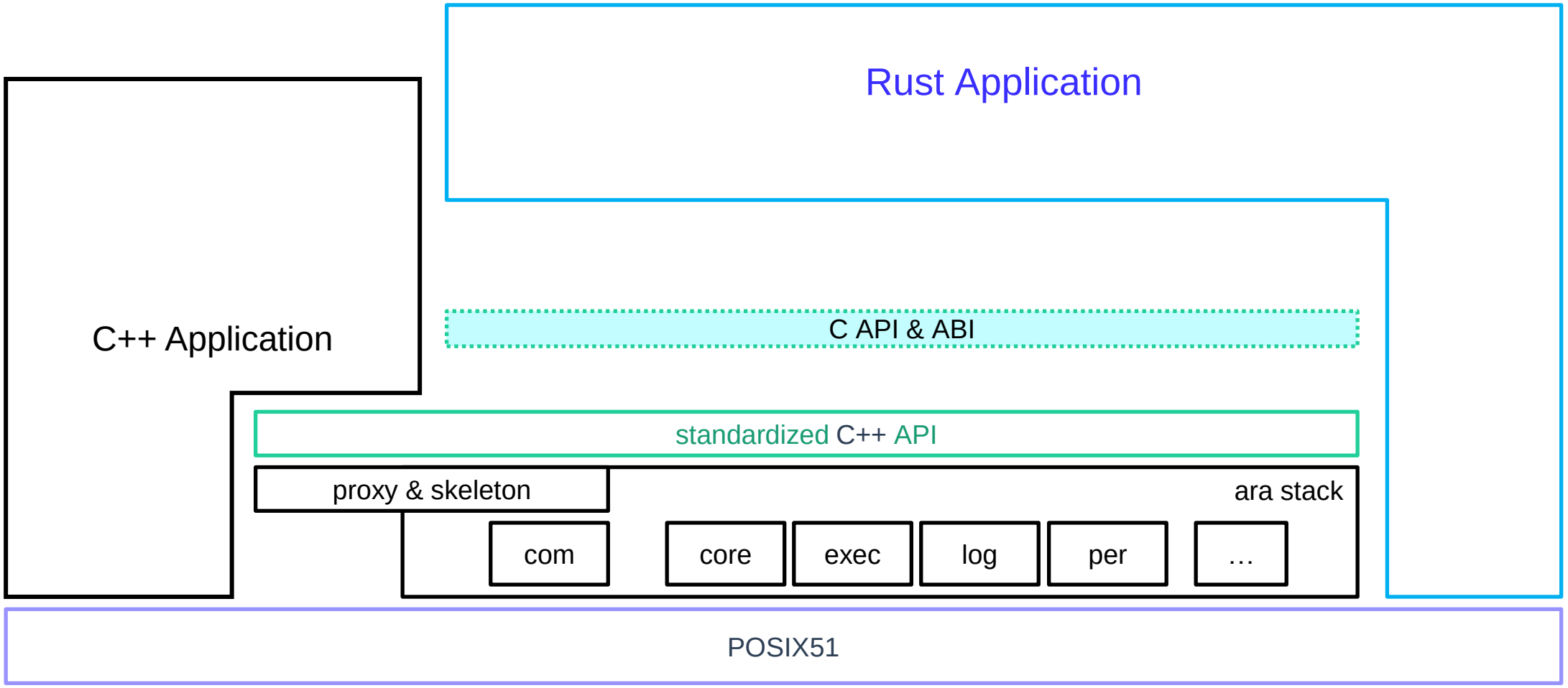
- No Standardized Binary Interface for neither C++ nor Rust
- There is a Standardized Binary Interface (ABI) for C

- Solution: Define a C Interface

…

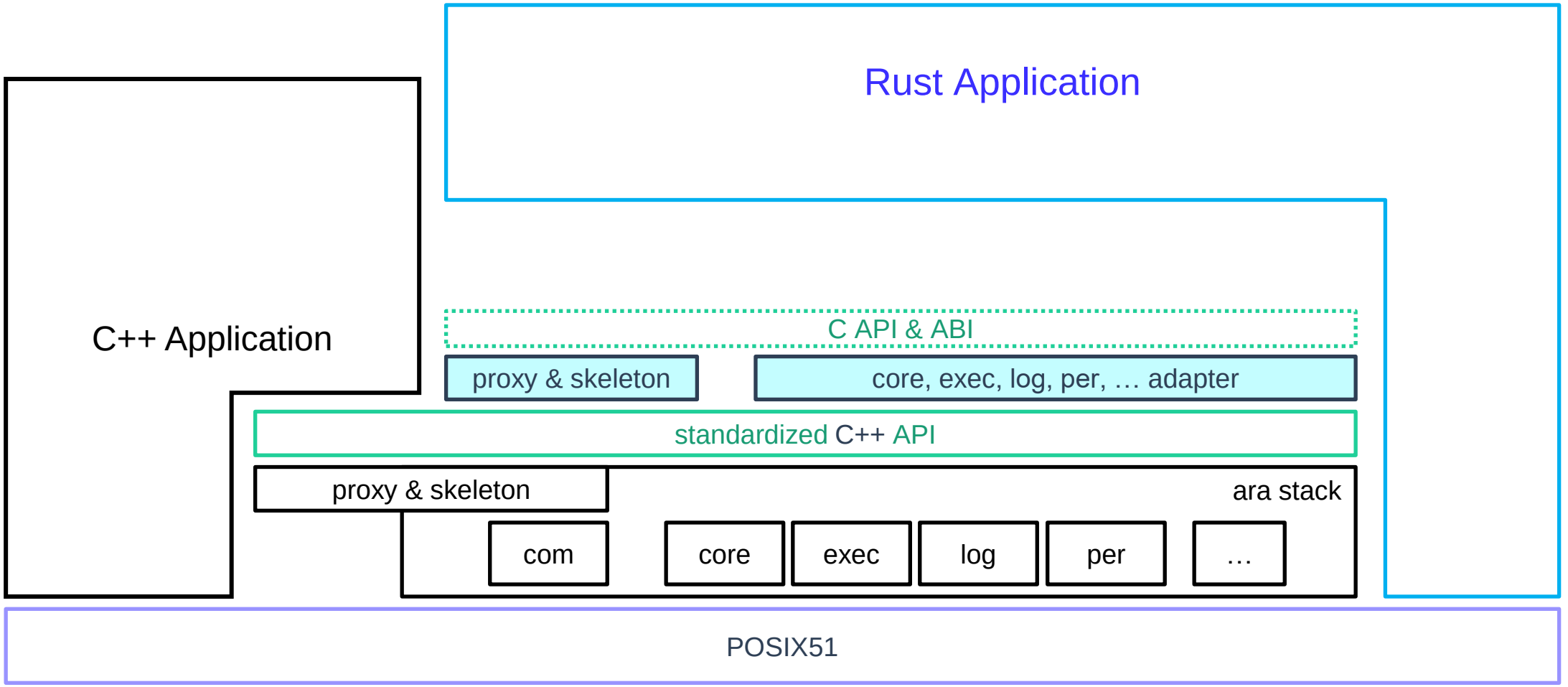
AUTOSAR Adaptive Platform Stack

A C Interface Provides a Defined In-memory Representation



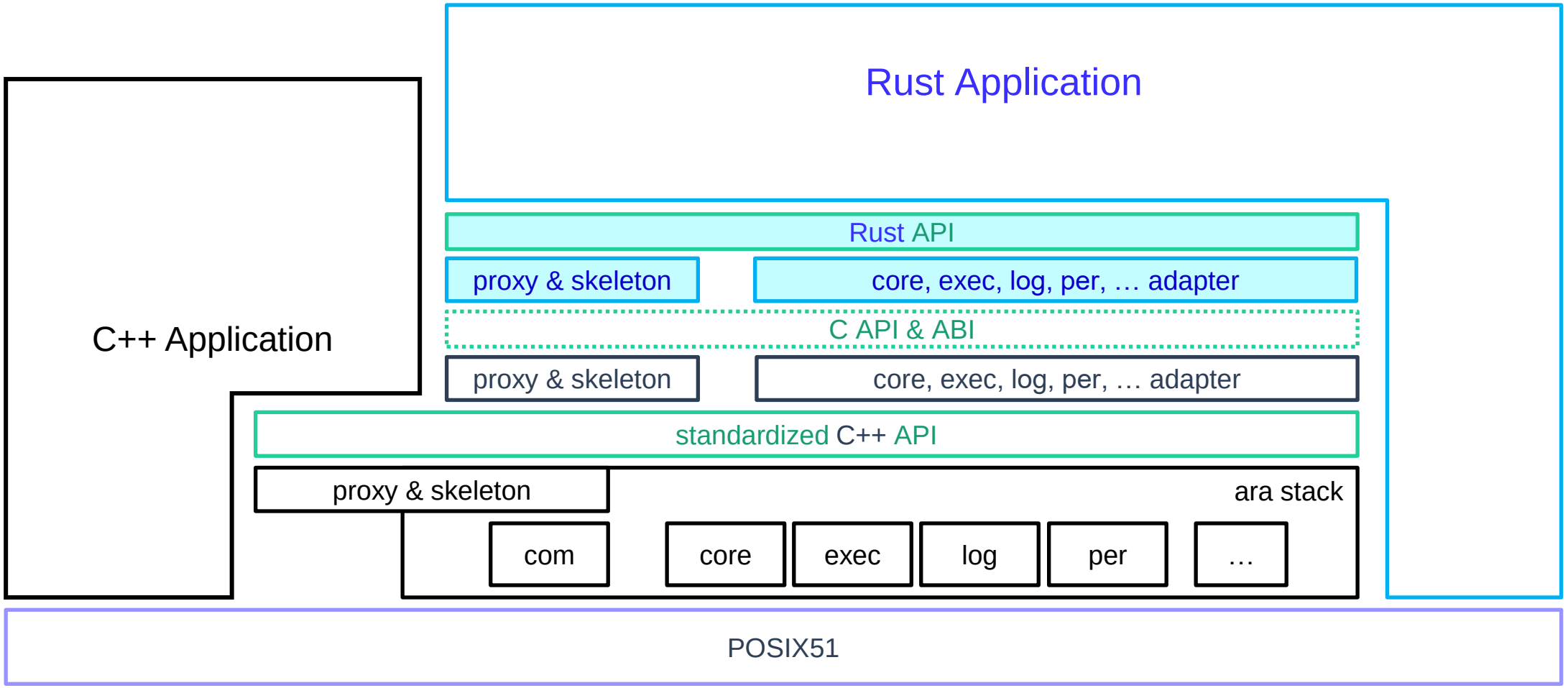
AUTOSAR Adaptive Platform Stack

Impedance matching with C++



AUTOSAR Adaptive Platform Stack

Nicer Rust API. This Solution is quite Complex



Combining C++ with other Languages

More Practical problems of this approach

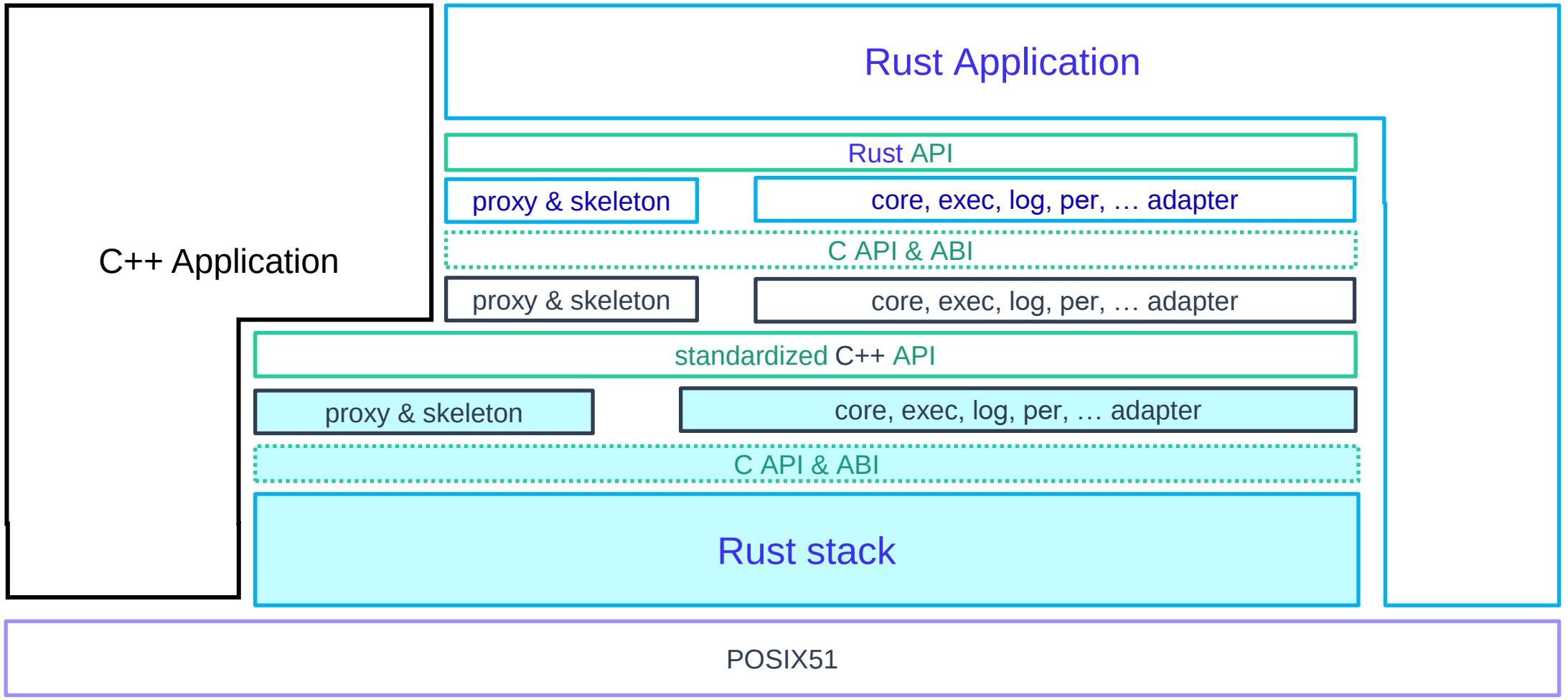
- No Standardized Binary Interface for neither C++ nor Rust
- There is a Standardized Binary Interface (ABI) for C

- Solution: Define a C Interface

- Mapping ara::core::Result<std::vector<T>> to a C type is tedious
- Responsibility to free memory needs manual checking

AUTOSAR Adaptive Platform Stack

And even more Complex for a Rust Stack



Part 2: Solution

- Motivation: Rust Applications for Adaptive Platform
- Solution: Language Neutral Binary Interface
- Practical: Running an AP Application in a browser
- Practical: With a Rust Stack and deploy for Embedded
- Outlook: Future Optimizations

Solution: WebAssembly

A Language Neutral high-level Interface

- WebAssembly was Created to run C, C++ and more in the Browser
- 2018: Standardized by WorldWideWebConsortium (W3C)

- WebAssembly Abstracts the CPU
- Translation to Native Code while Loading

Solution: WebAssembly

A Language Neutral high-level Interface

- WebAssembly was Created to run C, C++ and more in the Browser
- 2018: Standardized by WorldWideWebConsortium (W3C)

- WebAssembly Abstracts the CPU
- Translation to Native Code while Loading

- 2019: WebAssembly Systems Interface (System Calls) was added
- 2024: WASI 0.2; more modular, polyglot components

Solution: WebAssembly Interface Types

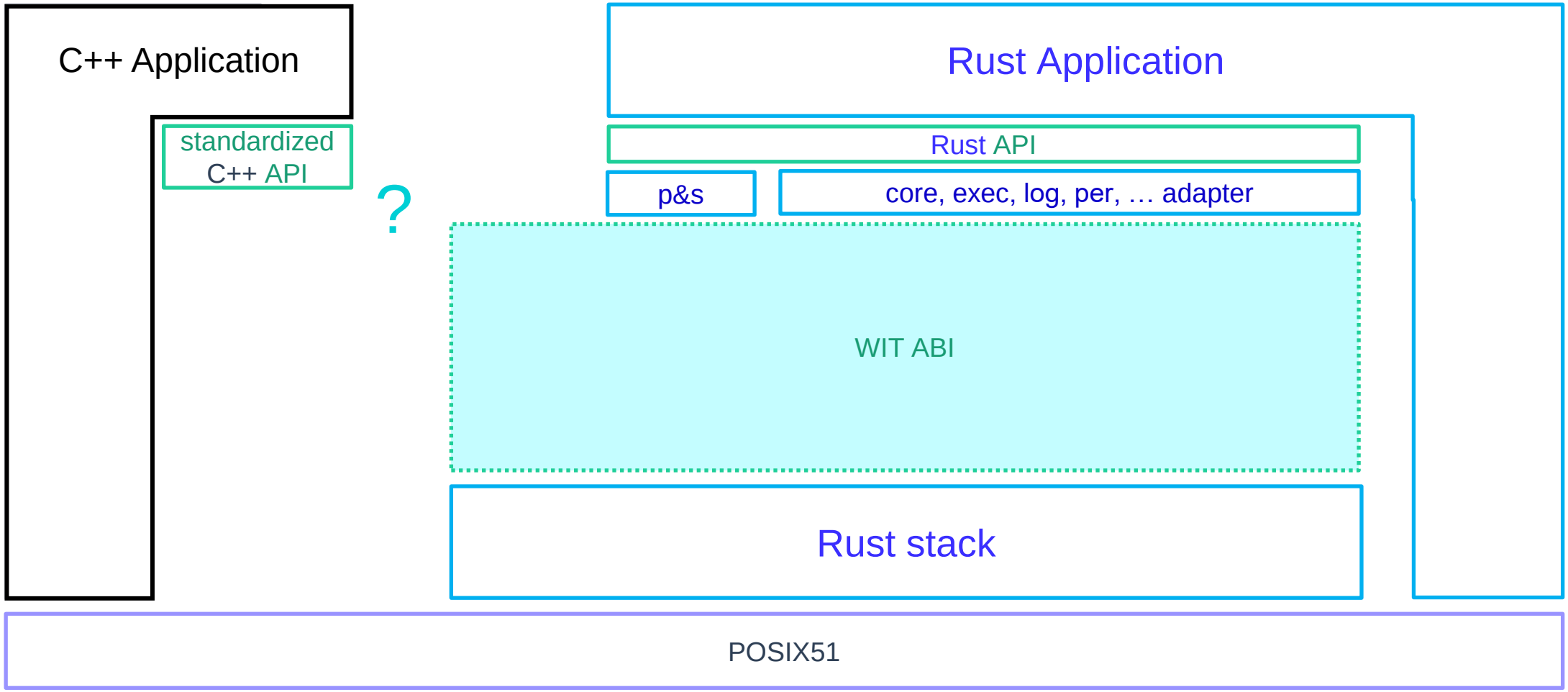
A Language Neutral high-level Interface

- Directly Compatible with a lot of Languages
- Supports Object Methods, Result and Optional
- Will soon support Future and Stream

- Composable Elements
- Direct Function Calls, no JSON Encoding/Decoding
- Shared Nothing
    - Enables full Insulation, Instrumentation and Network Transparency

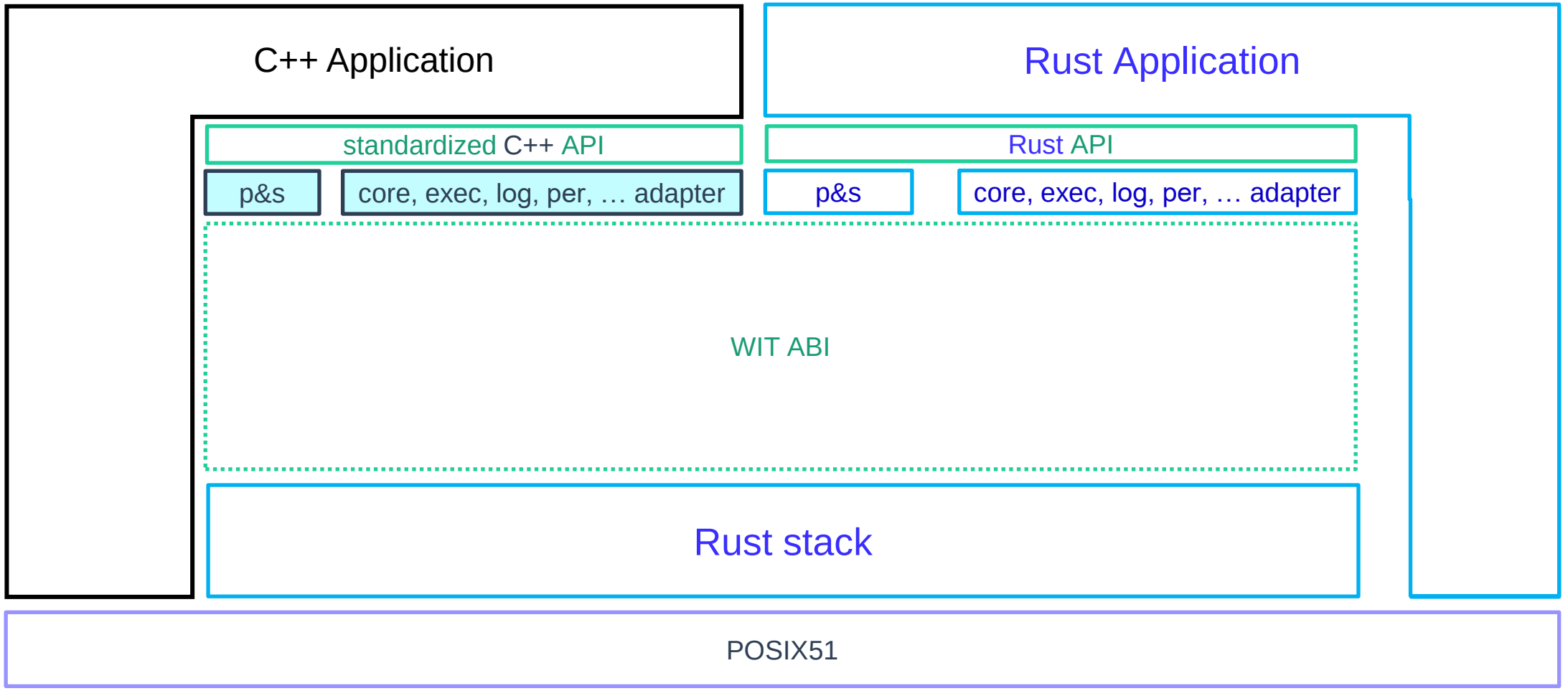
AUTOSAR Adaptive Platform Stack

A better Solution for a Rust Stack, but what about C++?



AUTOSAR Adaptive Platform Stack

Adding a symmetric solution for C++

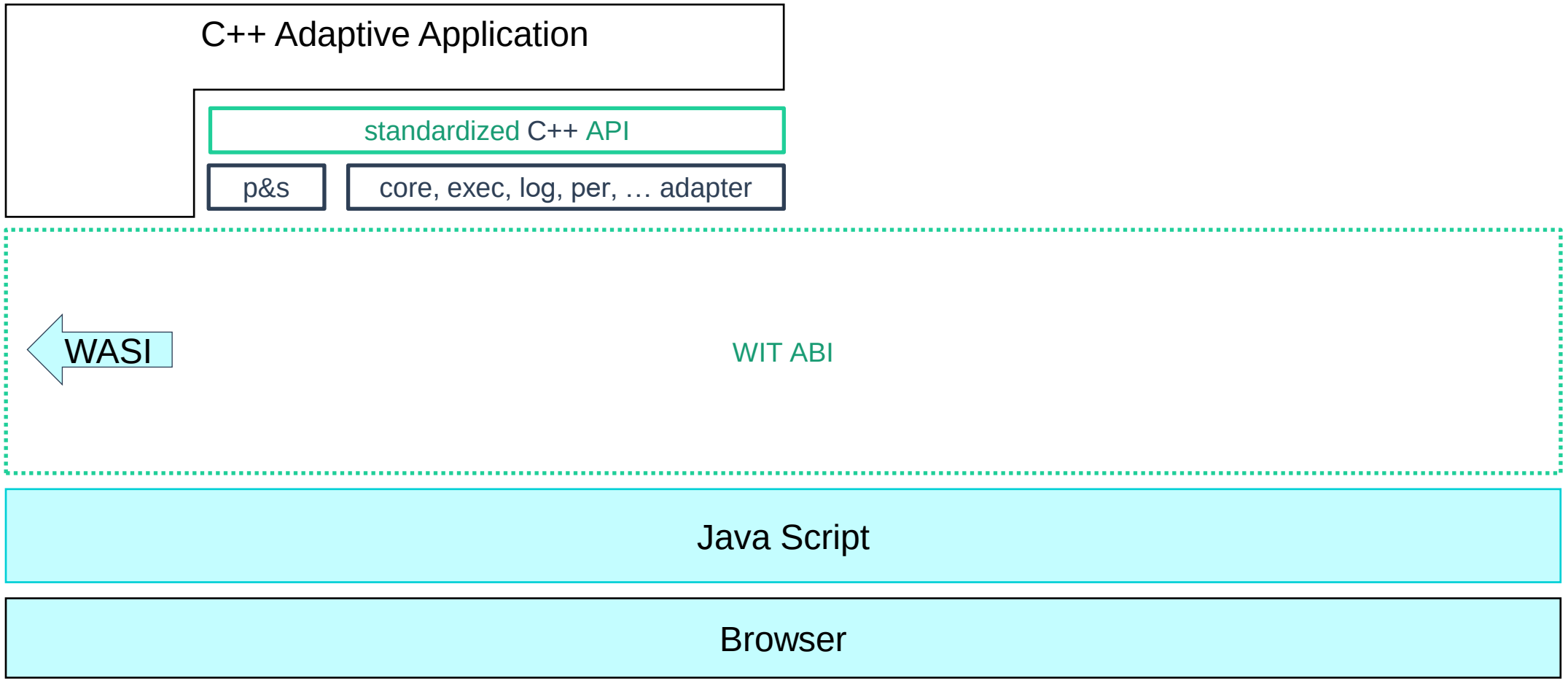


Part 3: Practical

- Motivation: Rust Applications for Adaptive Platform
- Solution: Language neutral binary Interface
- Practical: Running an AP Application in a browser
- Practical: With a Rust Stack and deploy for Embedded
- Outlook: Future Optimizations

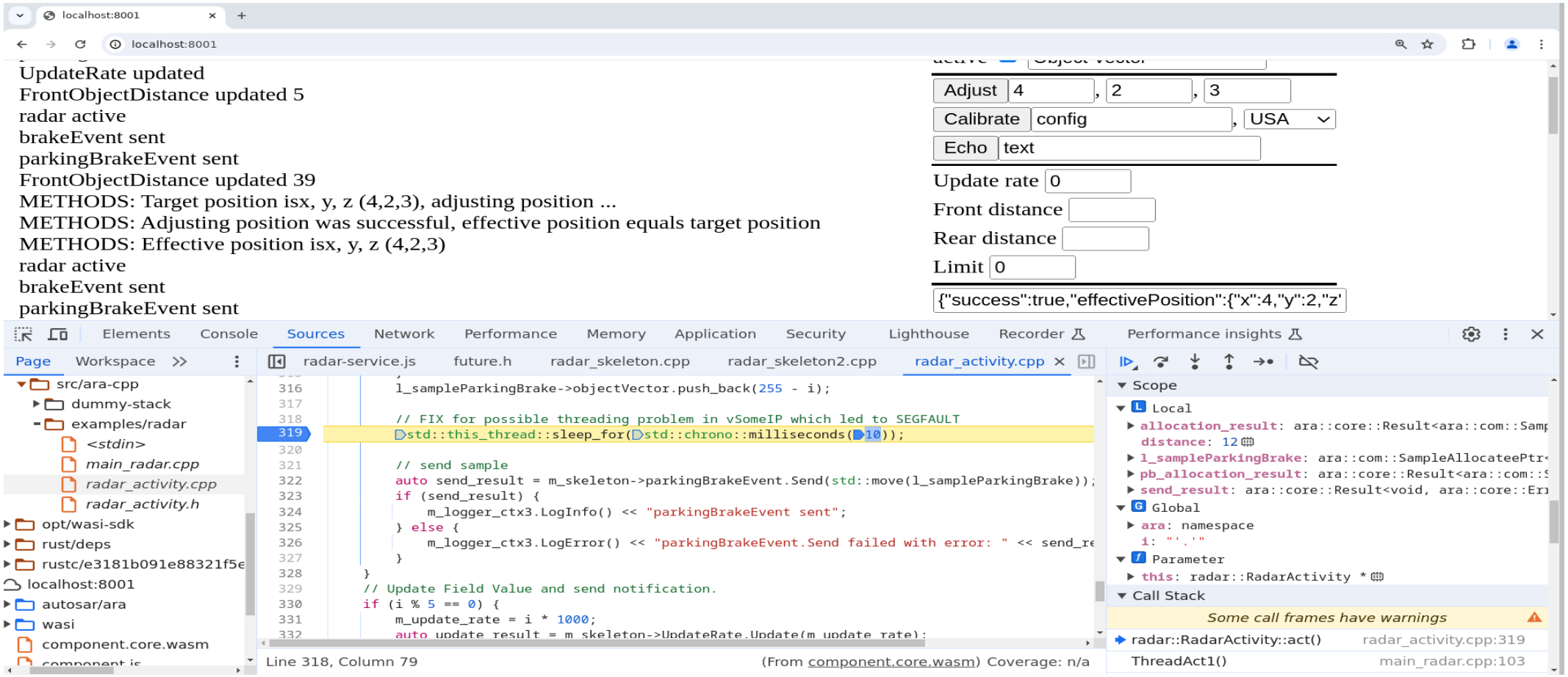
Practical

Compiling the C++ Application to WebAssembly and running it in a browser



JavaScript example

Radar example compiled to WebAssembly



Examplary WIT interface

ara::core::

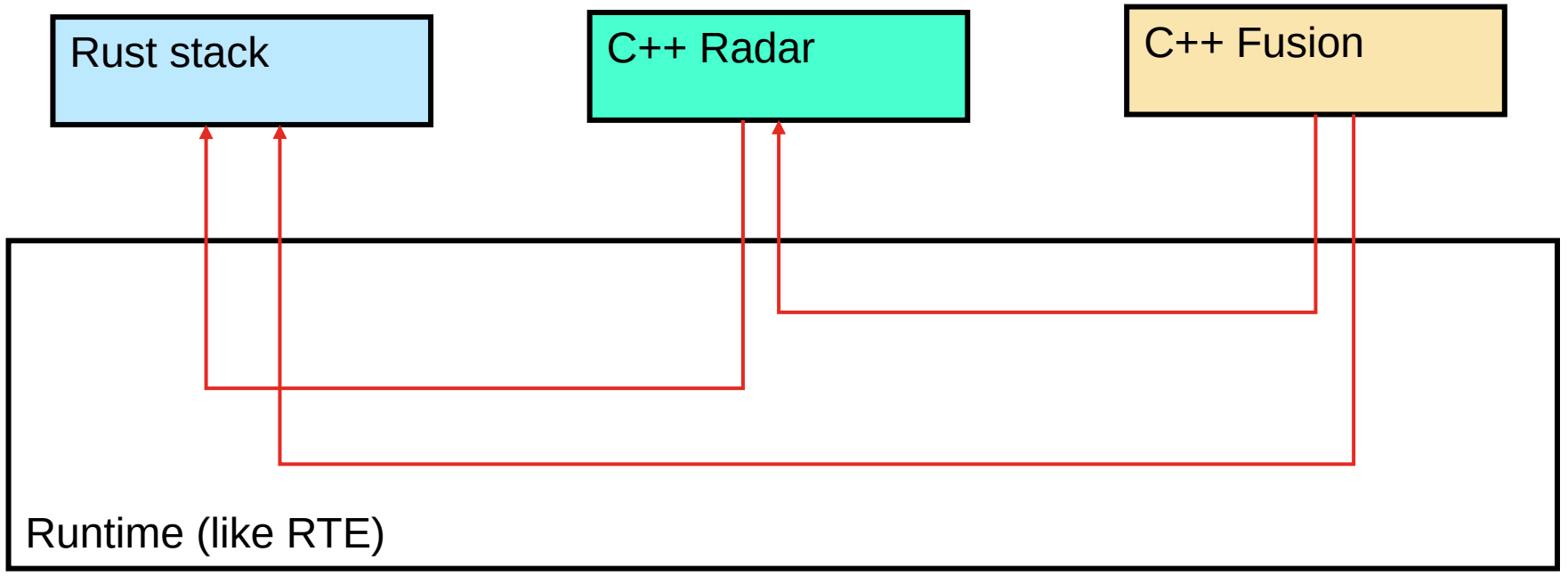
```
interface core {
    use types.{error-code};
    resource instance-specifier {
        to-string: func() -> string;
        clone: func() -> instance-specifier;
        create: static func(spec: string)
                        -> result<instance-specifier, error-code>;
    }

    initialize: func() -> result<_, error-code>;
    deinitialize: func() -> result<_, error-code>;
}
```

Part 3: Practical

- Motivation: Rust Applications for Adaptive Platform
- Solution: Language neutral binary Interface
- Practical: Running an AP Application in a browser
- Practical: With a Rust Stack and deploy for Embedded
- Outlook: Future Optimizations

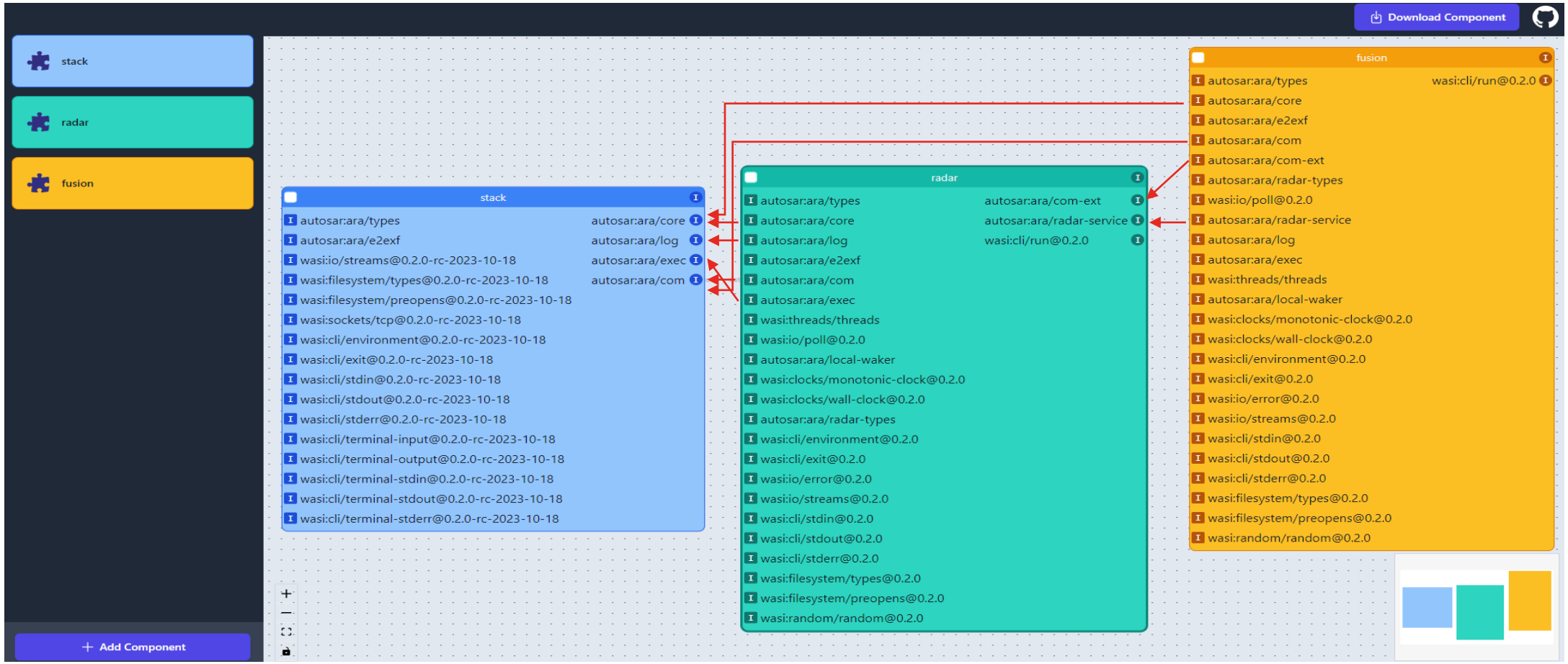
Rust example

Targeting a WebAssembly Runtime



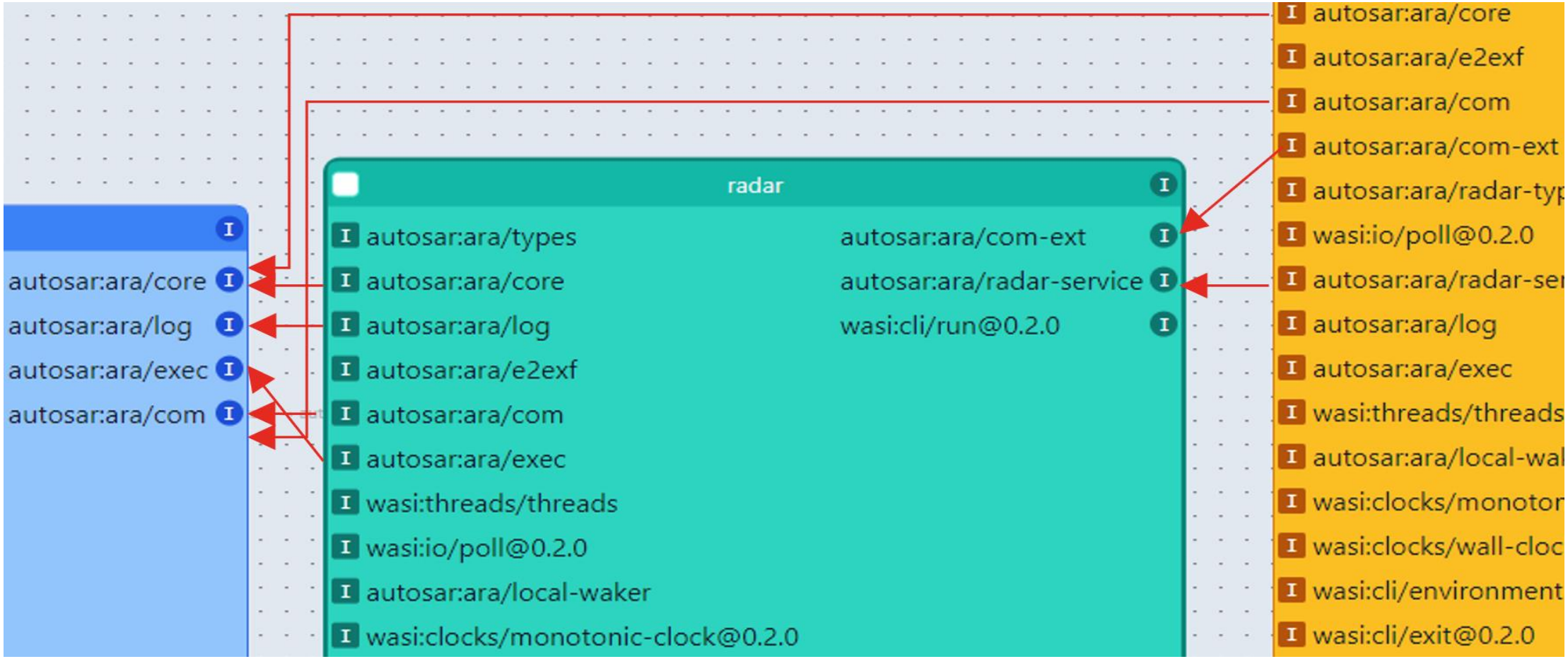
Easy graphical composition

(https://wasmbuilder.app/ isn't complete yet)



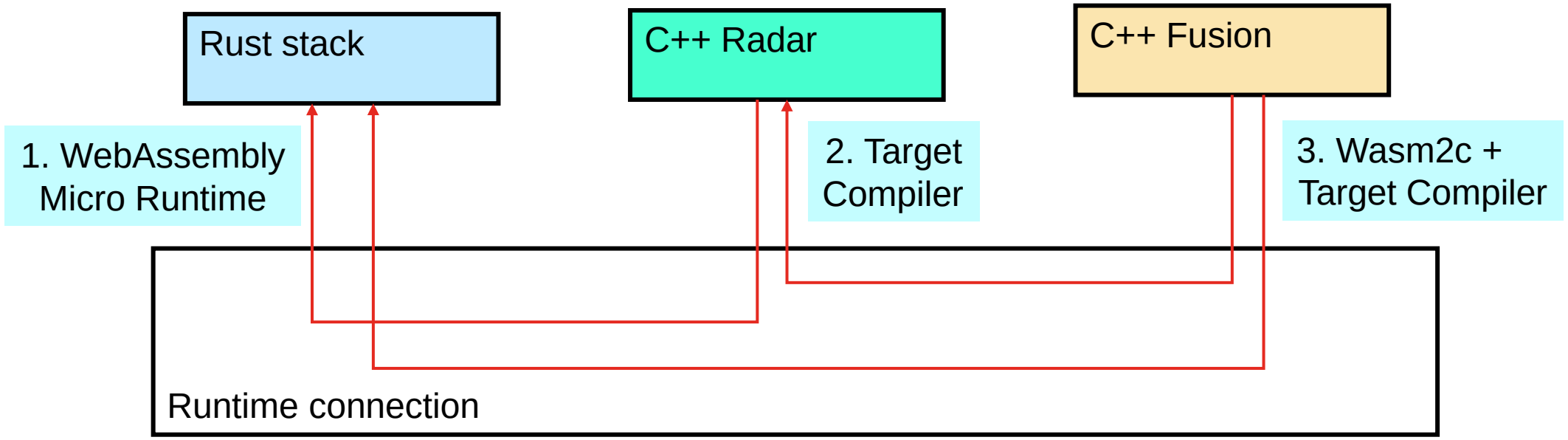
Zooming in

AUTOSAR adaptive (ara) as well as Operating system (wasi) APIs, versioned



Rust example

Three Exchangeable Options for Porting to an Embedded Platform



Part 4: Outlook

- Motivation: Rust Applications for Adaptive Platform
- Solution: Language neutral binary Interface
- Practical: Running an AP Application in a browser
- Practical: With a Rust Stack and deploy for Embedded
- Outlook: The Future of WASI

Current Limits of this technology

Still being worked on

- C++ code generation
- Multiple Threads
- C++ Exceptions
- Asynchronous calls
- SOME/IP
- Qualified execution

Opportunities

- Containers for microcontrollers
  - CPU and OS independent
  - Full Insulation and Deterministic Timing

- Running AUTOSAR Applications on Custom Middleware
- W3C Standardization of Technology
- Running inside VScode

Future options

- Bytecodealliance SIG embedded
  - Work towards efficiency and small size
  - Industrial Interests Represented
- WASI 0.3
  - Futures and Streams
  - Independent choice of asynchronous and blocking for each block


AUTOSAR™
Thank you!