



AUTOSAR™

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



BOSCH Continental



STELLANTIS

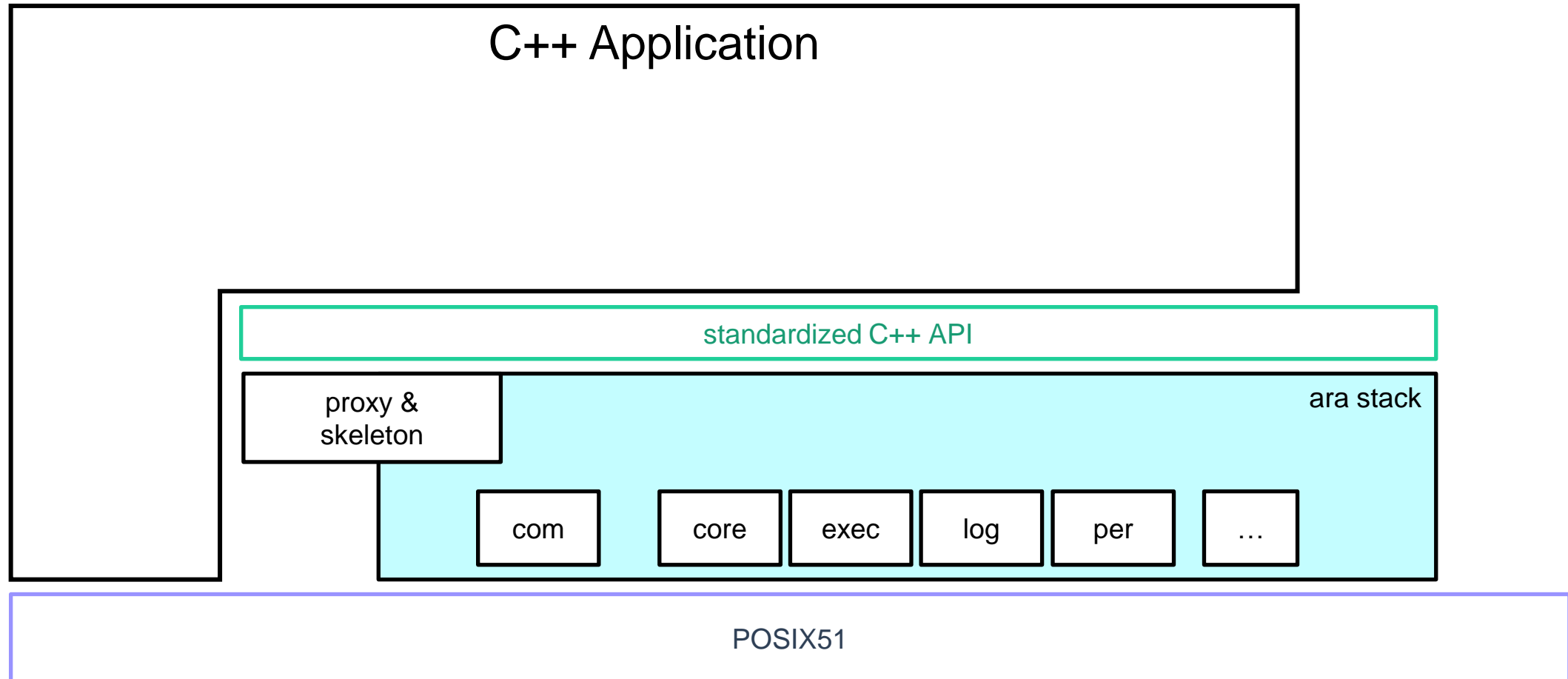
TOYOTA VOLKSWAGEN GROUP

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

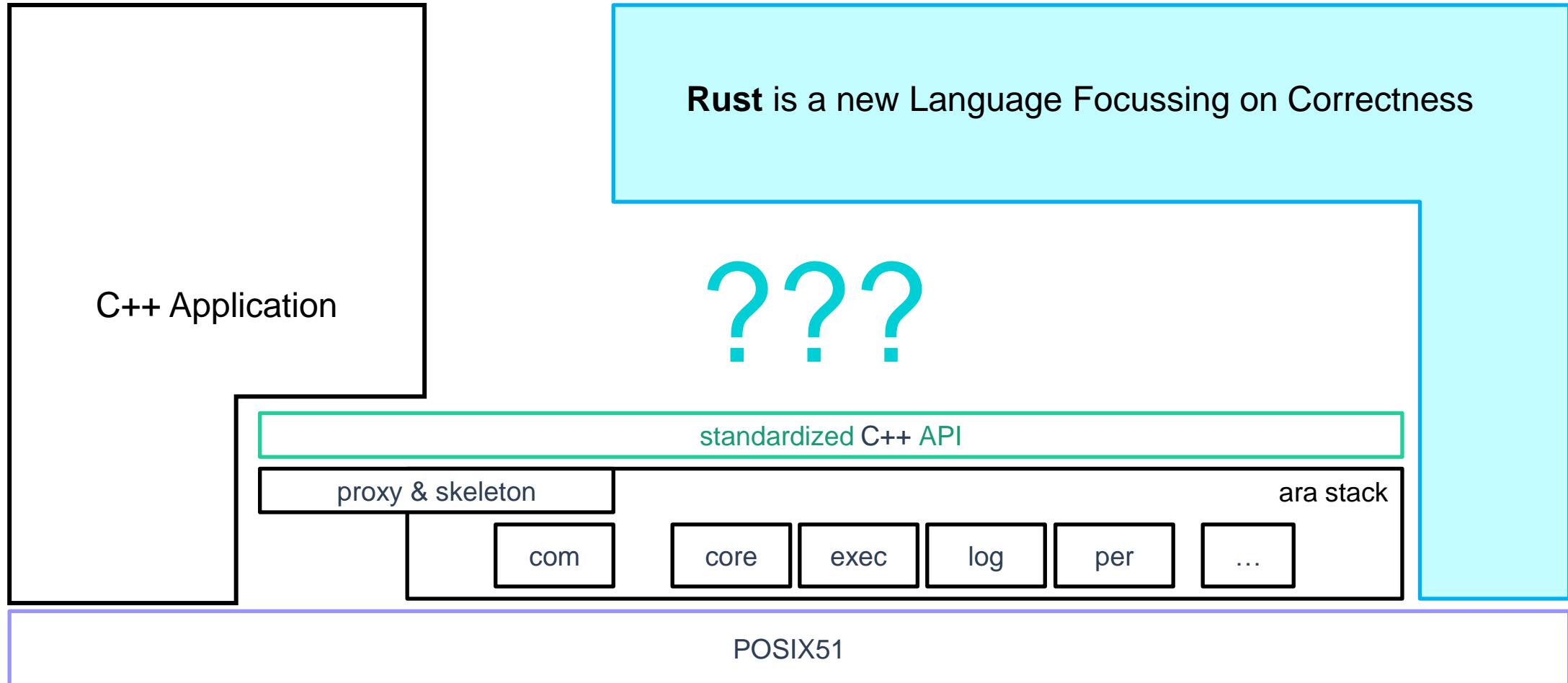
AUTOSAR Adaptive Platform Stack

The classical situation for C++ Applications



AUTOSAR Adaptive Platform Stack

But how to write a Rust Application?



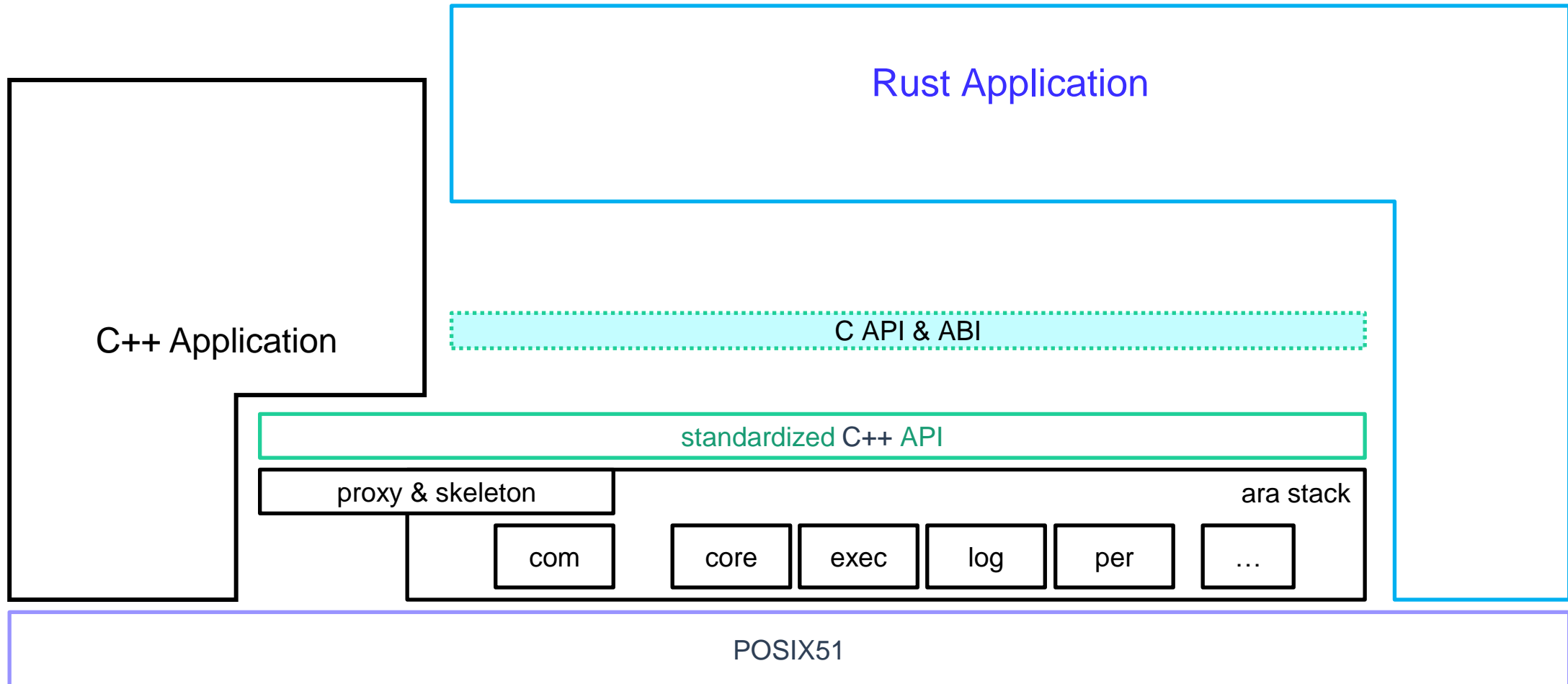
Combining C++ with other languages

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

...

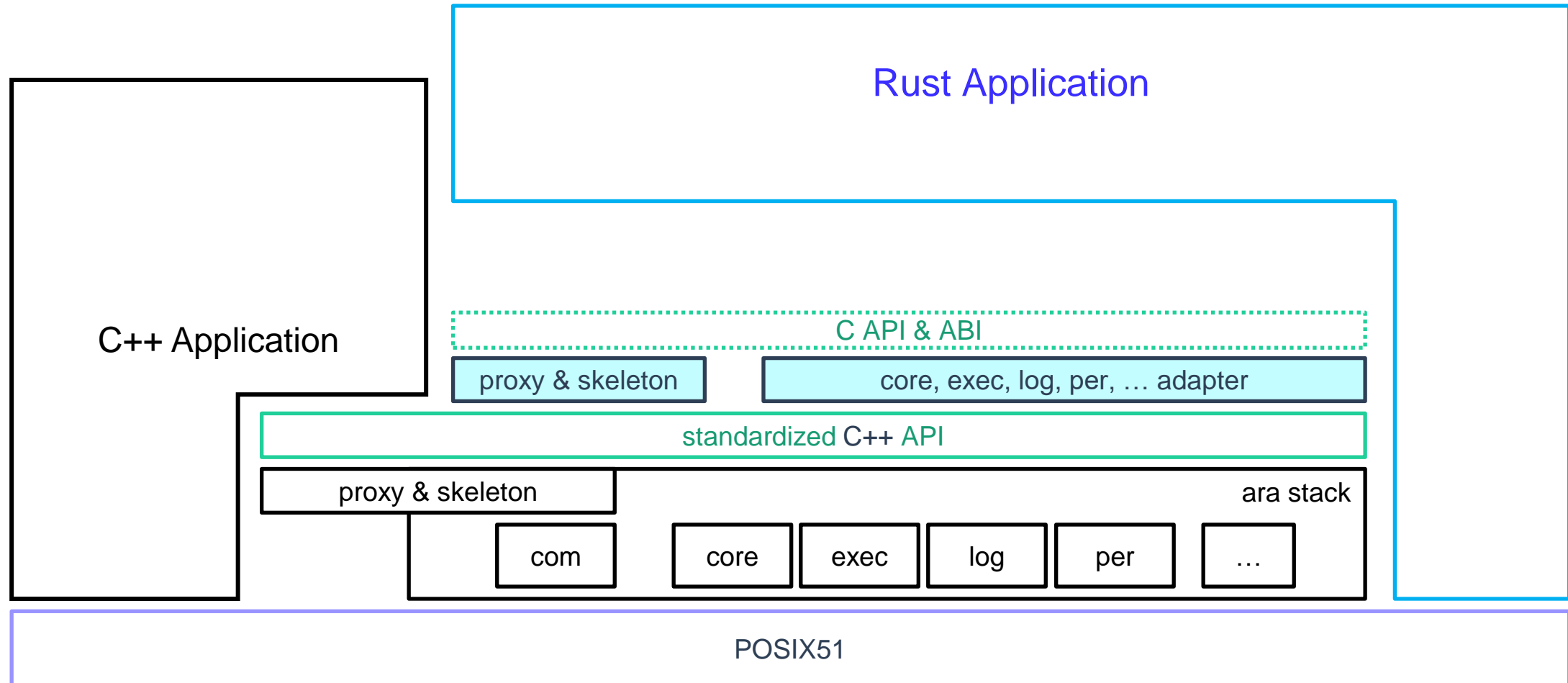
AUTOSAR Adaptive Platform Stack

A C Interface Provides a Defined In-memory Representation



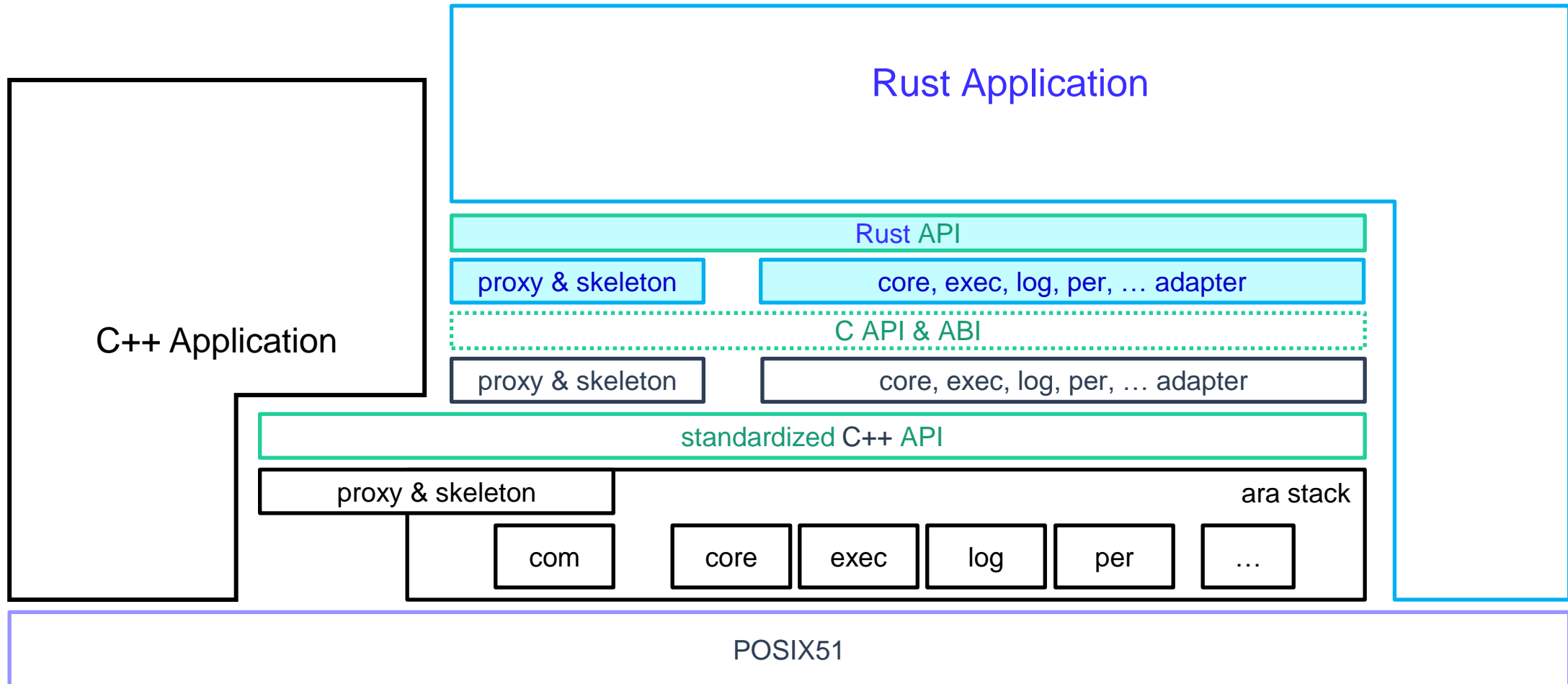
AUTOSAR Adaptive Platform Stack

Impedance matching with C++



AUTOSAR Adaptive Platform Stack

Nicer Rust API. This Solution is quite Complex



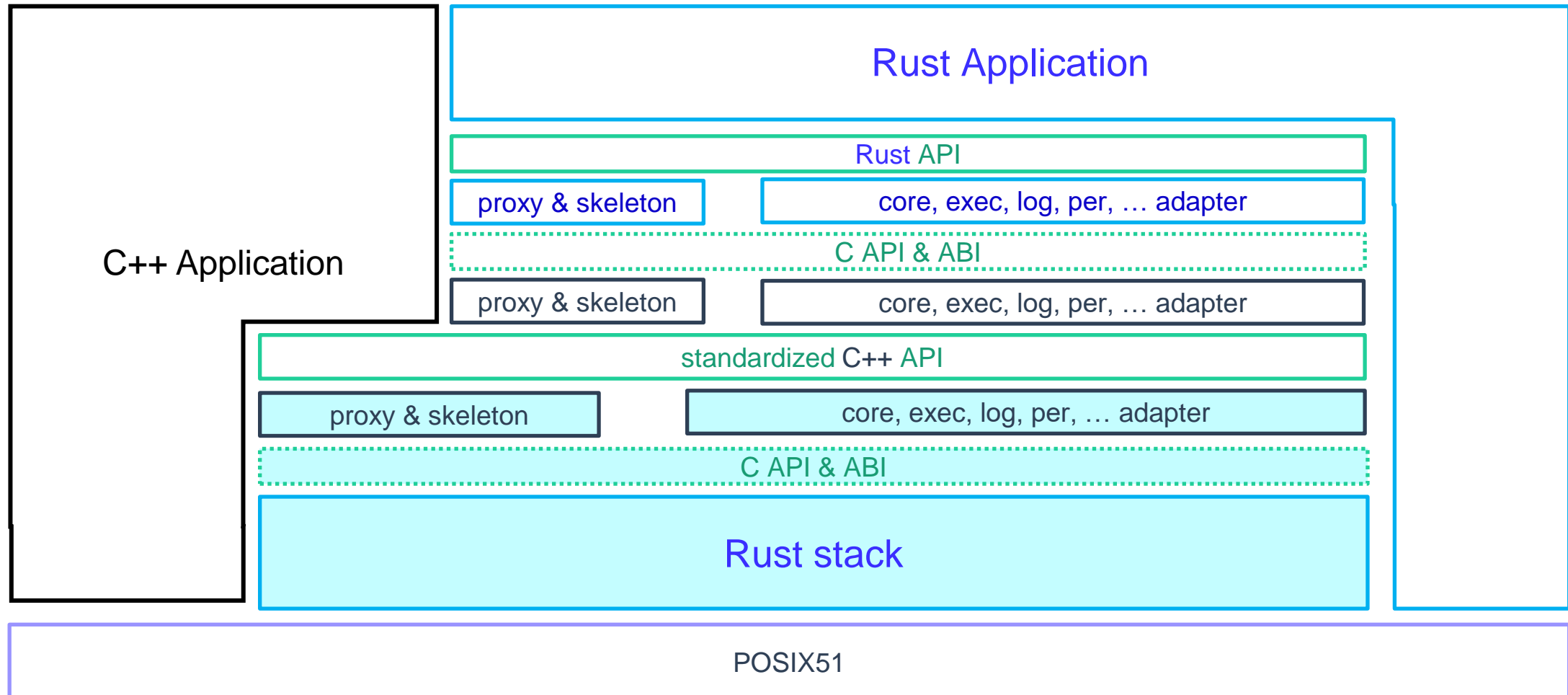
Combining C++ with other Languages

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

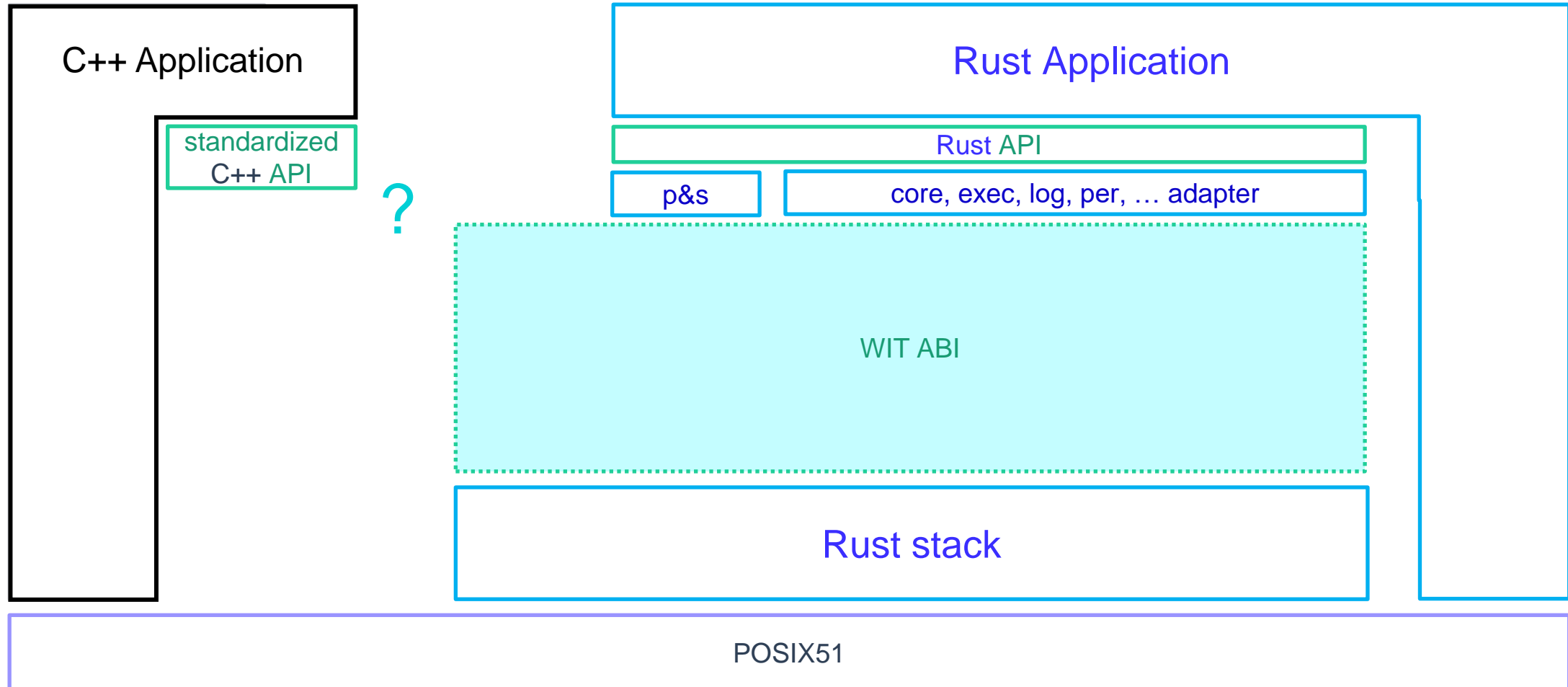
A Language Neutral high-level Interface

- 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

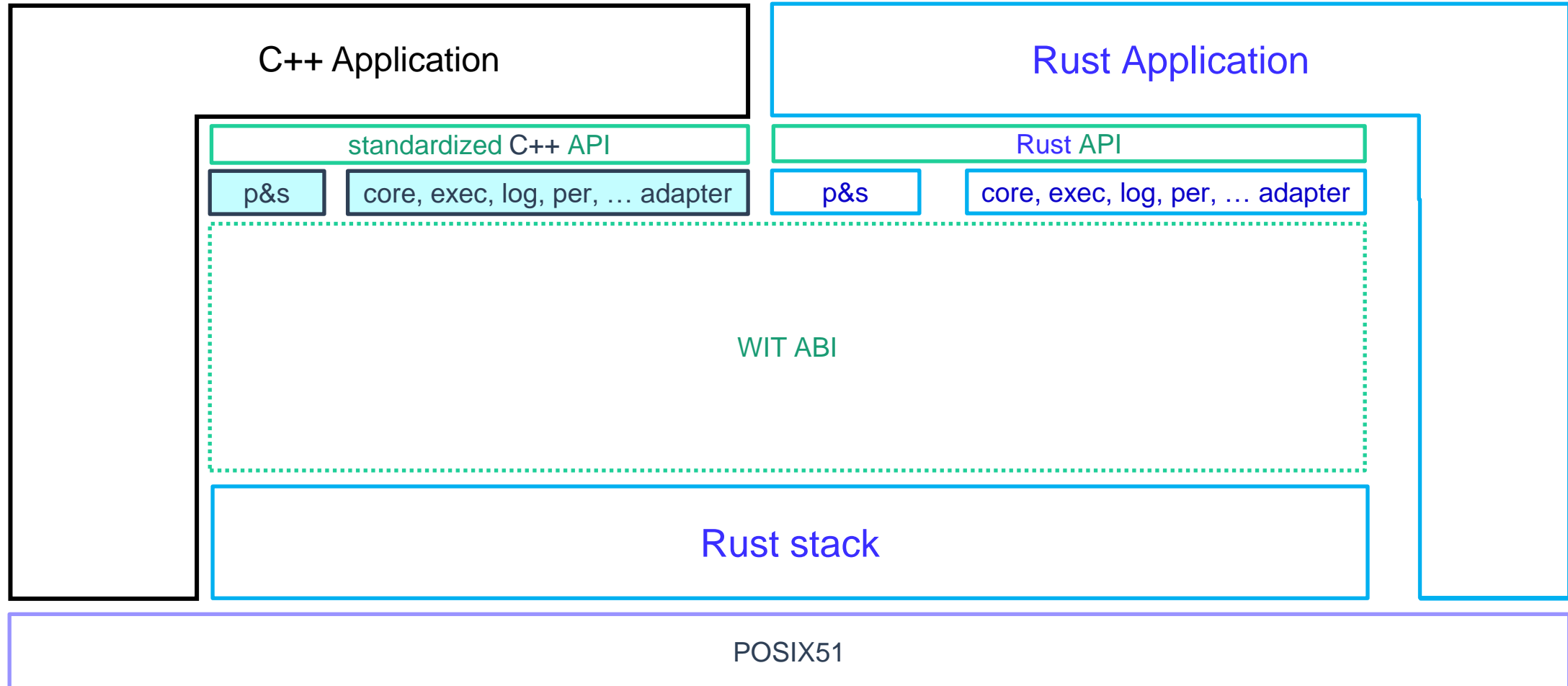
AUTOSAR Adaptive Platform Stack

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



AUTOSAR Adaptive Platform Stack

Adding a symmetric solution for C++

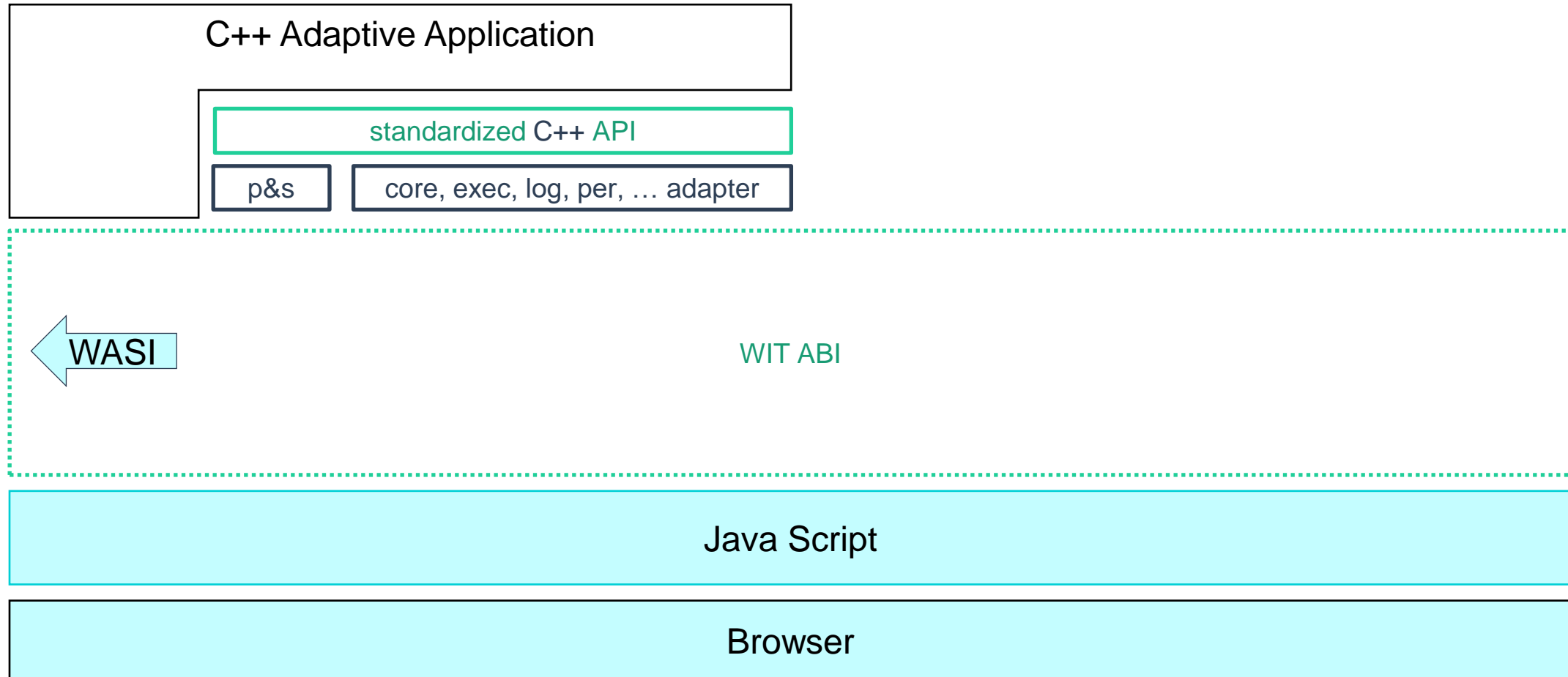


Part 3: Practical

- ▶ 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

UpdateRate updated
FrontObjectDistance updated 5
radar active
brakeEvent sent
parkingBrakeEvent sent
FrontObjectDistance updated 39
METHODS: Target position isx, y, z (4,2,3), adjusting position ...
METHODS: Adjusting position was successful, effective position equals target position
METHODS: Effective position isx, y, z (4,2,3)
radar active
brakeEvent sent
parkingBrakeEvent sent

Adjust 4, 2, 3
Calibrate config, USA
Echo text
Update rate 0
Front distance
Rear distance
Limit 0
{"success":true,"effectivePosition":{"x":4,"y":2,"z":3}}

```
316     _l_sampleParkingBrake->objectVector.push_back(255 - i);  
317  
318     // FIX for possible threading problem in vSomeIP which led to SEGFault  
319     std::this_thread::sleep_for(std::chrono::milliseconds(10));  
320  
321     // send sample  
322     auto send_result = m_skeleton->parkingBrakeEvent.Send(std::move(_l_sampleParkingBrake));  
323     if (send_result) {  
324         m_logger_ctx3.LogInfo() << "parkingBrakeEvent sent";  
325     } else {  
326         m_logger_ctx3.LogError() << "parkingBrakeEvent.Send failed with error: " << send_result; 327     }  
328 }  
329 // Update Field Value and send notification.  
330 if (i % 5 == 0) {  
331     m_update_rate = i * 1000;  
332     auto update_result = m_skeleton->UpdateRate.Update(m_update_rate);
```

Line 318, Column 79 (From component.core.wasm) Coverage: n/a

Scope
Local
allocation_result: ara::core::Result<ara::com::SampleDistance> distance: 12
l_sampleParkingBrake: ara::com::SampleAllocateePtr<SampleAllocatee>
pb_allocation_result: ara::core::Result<ara::com::SampleParkingBrake>
send_result: ara::core::Result<void, ara::core::Error>
Global
ara: namespace
i: ''
Parameter
this: radar::RadarActivity *
Call Stack
Some call frames have warnings
radar::RadarActivity::act() radar_activity.cpp:319
ThreadAct1() main_radar.cpp:103

Exemplary WIT interface

ara::core::

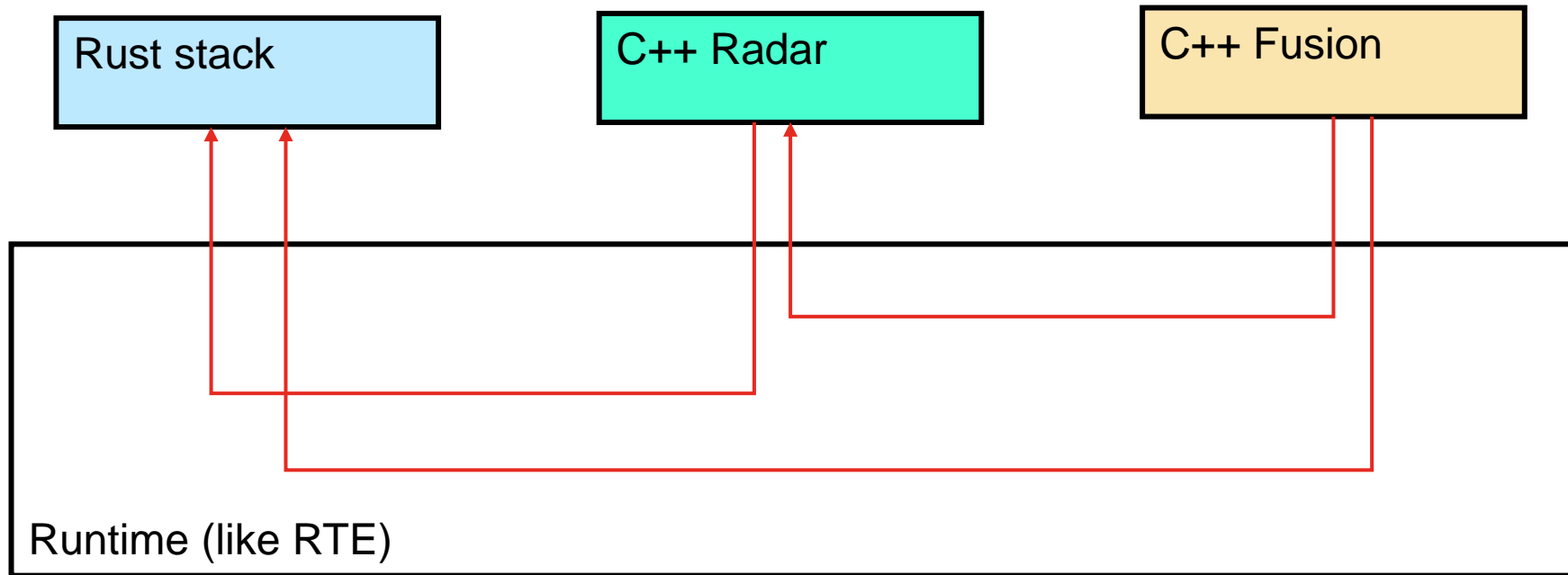
```
interface core {  
    use types.{error-code};  
    resource instance-specifier {  
        to-string: func() -> string;  
        clone: func() -> instance-specifier;  
        create: static func(spec: string)  
            -> result<instance-specifier, error-code>;  
    }  
  
    initialize: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

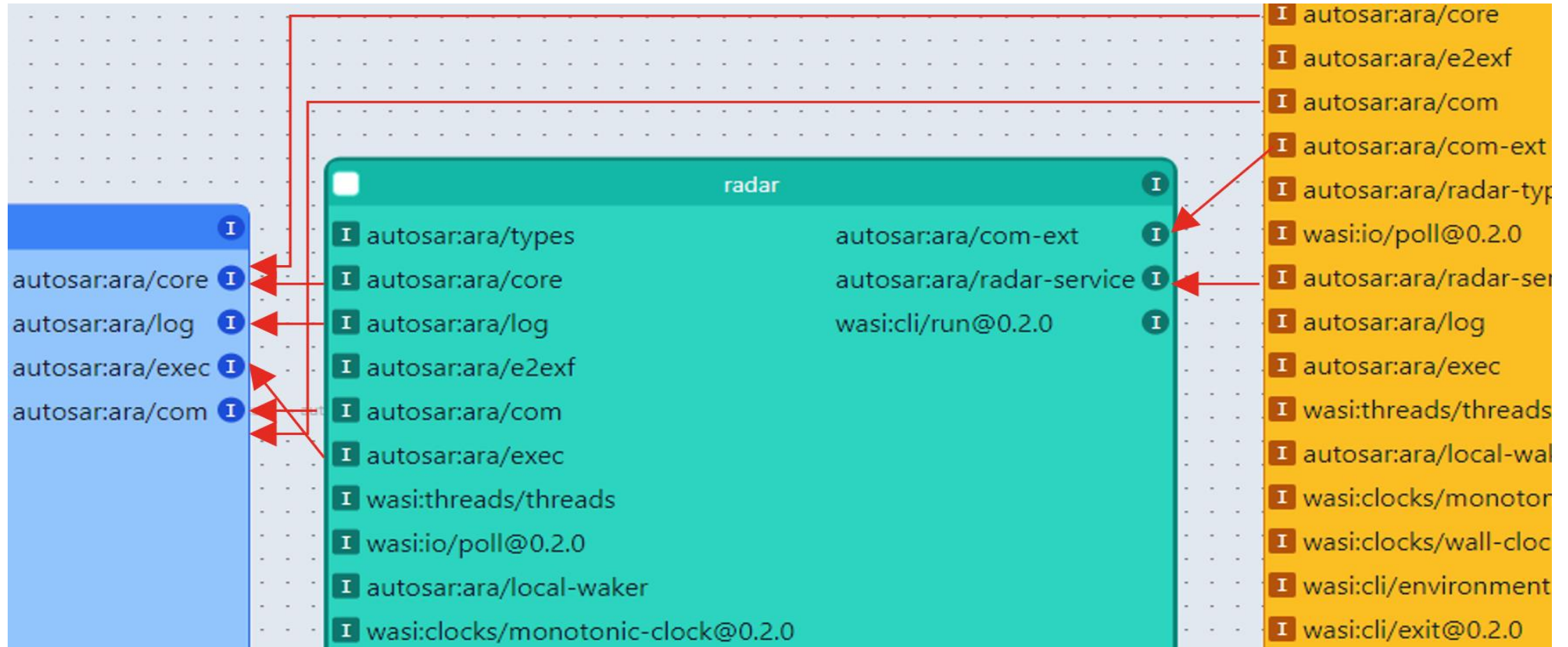
The screenshot displays the wasmbuilder.app graphical composition interface. On the left, a sidebar contains three component categories: 'stack' (blue), 'radar' (teal), and 'fusion' (yellow). The main workspace shows three component windows arranged in a stack:

- stack (blue window):** Lists components such as `autosar:ara/types`, `autosar:ara/e2exf`, `wasi:io/streams@0.2.0-rc-2023-10-18`, `wasi:filesystem/types@0.2.0-rc-2023-10-18`, `wasi:filesystem/preopens@0.2.0-rc-2023-10-18`, `wasi:sockets/tcp@0.2.0-rc-2023-10-18`, `wasi:cli/environment@0.2.0-rc-2023-10-18`, `wasi:cli/exit@0.2.0-rc-2023-10-18`, `wasi:cli/stdin@0.2.0-rc-2023-10-18`, `wasi:cli/stdout@0.2.0-rc-2023-10-18`, `wasi:cli/stderr@0.2.0-rc-2023-10-18`, `wasi:cli/terminal-input@0.2.0-rc-2023-10-18`, `wasi:cli/terminal-output@0.2.0-rc-2023-10-18`, `wasi:cli/terminal-stdin@0.2.0-rc-2023-10-18`, `wasi:cli/terminal-stdout@0.2.0-rc-2023-10-18`, and `wasi:cli/terminal-stderr@0.2.0-rc-2023-10-18`.
- radar (teal window):** Lists components such as `autosar:ara/types`, `autosar:ara/core`, `autosar:ara/log`, `autosar:ara/e2exf`, `autosar:ara/com`, `wasi:threads/threads`, `wasi:io/poll@0.2.0`, `autosar:ara/local-waker`, `wasi:clocks/monotonic-clock@0.2.0`, `wasi:clocks/wall-clock@0.2.0`, `autosar:ara/radar-types`, `wasi:cli/environment@0.2.0`, `wasi:cli/exit@0.2.0`, `wasi:io/error@0.2.0`, `wasi:io/streams@0.2.0`, `wasi:cli/stdin@0.2.0`, `wasi:cli/stdout@0.2.0`, `wasi:cli/stderr@0.2.0`, `wasi:filesystem/types@0.2.0`, `wasi:filesystem/preopens@0.2.0`, and `wasi:random/random@0.2.0`.
- fusion (yellow window):** Lists components such as `autosar:ara/types`, `autosar:ara/core`, `autosar:ara/e2exf`, `autosar:ara/com`, `autosar:ara/com-ext`, `autosar:ara/radar-types`, `wasi:io/poll@0.2.0`, `autosar:ara/radar-service`, `autosar:ara/log`, `autosar:ara/exec`, `wasi:threads/threads`, `autosar:ara/local-waker`, `wasi:clocks/monotonic-clock@0.2.0`, `wasi:clocks/wall-clock@0.2.0`, `wasi:cli/environment@0.2.0`, `wasi:cli/exit@0.2.0`, `wasi:io/error@0.2.0`, `wasi:io/streams@0.2.0`, `wasi:cli/stdin@0.2.0`, `wasi:cli/stdout@0.2.0`, `wasi:cli/stderr@0.2.0`, `wasi:filesystem/types@0.2.0`, `wasi:filesystem/preopens@0.2.0`, and `wasi:random/random@0.2.0`.

Red arrows indicate dependencies between components across the windows. A '+ Add Component' button is visible at the bottom left of the interface.

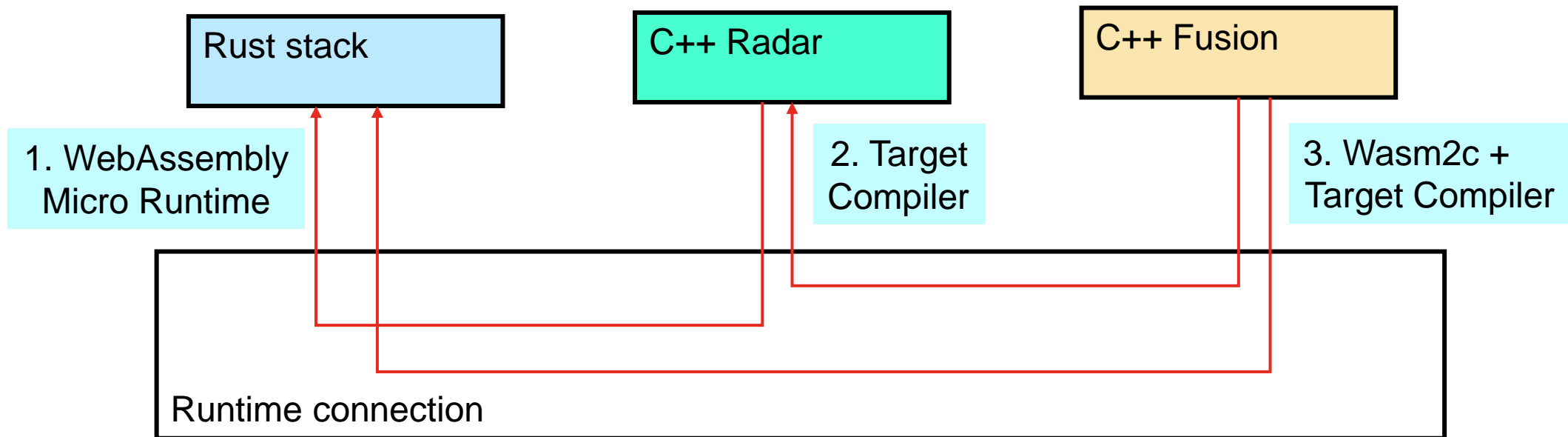
Zooming in

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!



BOSCH Continental



STELLANTIS

TOYOTA

VOLKSWAGEN GROUP