Real World WASI Components

Colin Murphy

Content Authenticity Initiative

Adobe

What is CAI?

Content Credentials

Coalition for Content Provenance and Authenticity (C2PA)

Creator Assertions Working Group (CAWG)

Open-source libraries and tools

Adobe services

Adobe

Why use components?

Adobe

Services and SDKS

Public SDKS - Rust, Web, nodeJS, Python, C, C++

Adobe Internal SDKS

Adobe Services

Cameras and devices

Adobe

How do you make components?

Adobe

1 Dependencies

Ensure you libraries will compile to wasm32-wasi

wasm32-wasip1 + component adapter

or

wasm32-wasip2

Adobe

2 Define interfaces

Define a wit file based on your API

import vs use vs export

use streams!

wasi:filesystem/types.{descriptor} -> Seekable streams

Adobe

2 Define interfaces

```
world cai {
    export c2pa;
    export types;
    import c2pa-signer;
}

world signer {
    import types;
    export c2pa-signer;
}

world c2pa-tool {
    import c2pa;
}

world c2pa-webserver {
    import c2pa;
    export wasi:http/incoming-handler@0.2.4;
}
```

Adobe

3 Implement API component

wit-bindgen

additional_derives

wstd

Adobe

3 Implement API component

```
pub struct Manifest;

impl Guest for Manifest {
    fn format_from_path(path: String) -> Option<String> {
        c2pa::format_from_path(Path::new(&path))
    }

    type Builder = ComponentBuilder;
    type Reader = ComponentReader;
}

pub struct ComponentBuilder {
    builder: RefCell<C2paBuilder>,
}
```

Adobe

3 Implement API component

```
impl GuestBuilder for ComponentBuilder {
    fn new(json: Option<String>) -> Self {
        Self {
            builder: match json {
                Some(json) => C2paBuilder::from_json(&json).unwrap().into(),
                None => C2paBuilder::new().into(),
            },
        }
    }

    fn add_resource(&self, uri: String, stream: Input) -> Result<(), Error> {
        let seekable_stream = seekable_input_stream(stream).unwrap();
        self.builder
            .borrow_mut()
            .add_resource(&uri, seekable_stream)?;
        Ok(())
    }
```

Adobe

3 Implement API component

```
mod bindings {
    use crate::Manifest;

    wit_bindgen::generate!({
        world: "cai",
        with: {
            "wasi:clocks/wall-clock@0.2.4": generate,
            "wasi:io/streams@0.2.4": ::wasi::io::streams,
            "wasi:io/poll@0.2.4": ::wasi::io::poll,
            "wasi:io/error@0.2.4": ::wasi::io::error,
            "wasi:filesystem/types@0.2.4": ::wasi::filesystem::types,
        },
        path: "../wit",


        additional_derives: [serde::Serialize, serde::Deserialize],
        additional_derives_ignore: ["output", "input"],
    });

    export!(Manifest);
}
```

First class functions - API

```rust
pub fn sign<R, W>(
    &mut self,
    signer: &dyn Signer,
    format: &str,
    source: &mut R,
    dest: &mut W,
) -> Result<Vec<u8>>
where
    R: Read + Seek + Send,
    W: Write + Read + Seek + Send,
{
```

Adobe

First class functions - API

```rust
#[wasm_bindgen(js_name = signFileSystemHandle)]
pub async fn sign_file_system_handle(
    mut self,
    signer_ptr: u32,
    format: &str,
    source_handle: &FileSystemFileHandle,
    dest_handle: &FileSystemFileHandle,
) -> Result<JsValue, JsSysError> {
    let signer = JavaScriptAsyncSigner::from_ptr(signer_ptr)?;
```

Adobe

First class functions - WIT

```
world cai {
    export c2pa;
    export types;

    import c2pa-signer;
}

world signer {
    export c2pa-signer;
}
interface c2pa-signer {
    use types.{error};
    sign: func(data: list<u8>) -> result<list<u8>, error>;
}

interface c2pa {
    use wasi:io/streams@0.2.2.{input-stream, output-stream};
    use types.{assertion-type, error, input, manifest-definition, output, signer-config, signing-algorithm};

    resource builder {
        constructor(json: option<string>);
        sign: func(config: signer-config, format: string, source: input, dest: output) -> result<list<u8>, error>;
```

First class functions - Implementation

```rust
impl Signer for C2paSignerBinding {
    fn sign(&self, data: &[u8]) -> Result<Vec<u8>, c2pa::Error> {
        bindings::adobe::cai::c2pa_signer::sign(data)
            .map_err(|e| c2pa::Error::OtherError(Box::new(e)))
    }
```

Adobe

First class functions - Implementation

```rust
fn sign(
    &self,
    config: SignerConfig,
    format: String,
    source: Input,
    dest: Output,
) → Result<Vec<u8>, Error> {
    let mut input_stream = seekable_input_stream(source)?;
    let (mut output_stream, original_output_stream) = seekable_output_stream(dest)?;
    let signer = C2paSignerBinding::new(config);
    let manifest = self.builder.borrow_mut().sign(
        &signer,
        &format,
        &mut input_stream,
        &mut output_stream,
    )?;
```

Adobe

4 Implement dependent components

JCO for Web and Node

Rust for CLI

Adobe

4 jco for Node.js

Generate types
jco types wit -n c2pa-webserver -o node-c2pa/types

Build Component
jco componentize dist/http-c2pa.js --wit ../wit/ --world-name c2pa-webserver --out c2pa-node.wasm

WAC together (x2)
wac plug c2pa-node.wasm --plug c2pa_component.wasm -o c2pa-node-fused.wasm

Serve
jco serve c2pa-node-fused.wasm

Adobe

4 jco for web

WAC together
wac plug ../target/wasm32-wasip2/release/c2pa_component.wasm --plug ../target/wasm32-wasip2/release/signer.wasm

Transpile
jco transpile c2pa_component.wasm -o ./types

Serve
http-server

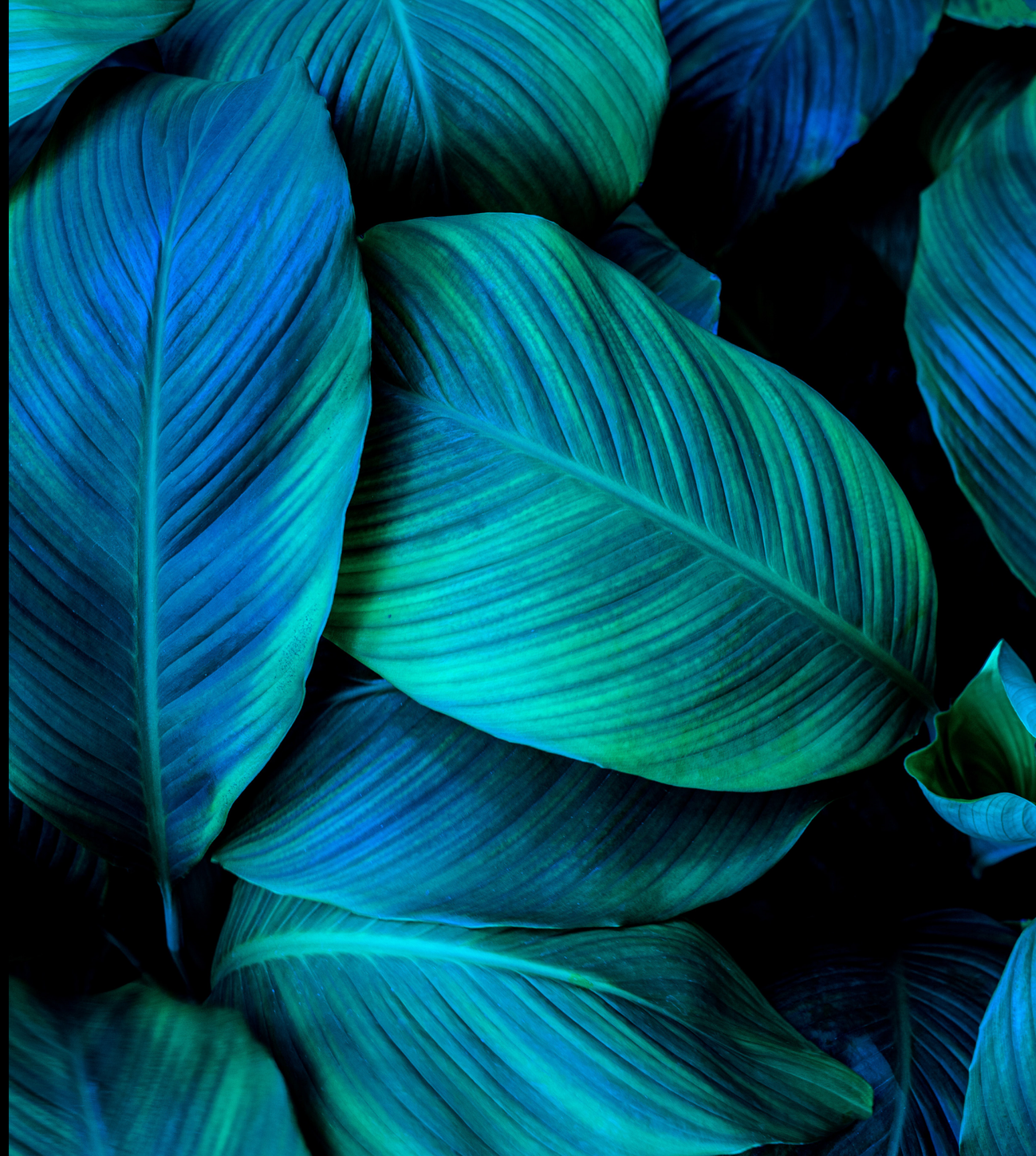
Adobe

4 jco for web

Access to script at 'adobe:cai/c2pa-signer' from origin 'http://localhost:8080' has been blocked by CORS policy: Cross origin requests are only supported for protocol schemes: chrome, chrome-extension, chrome-untrusted, data, http, https, isolated-app.

Adobe

Demo



Adobe

Where do we go from here?

Adobe

Wishlist

Java

Component test frameworks

Rust crates: tokio, axum, reqwest

No more nightly for stdlib on wasip2

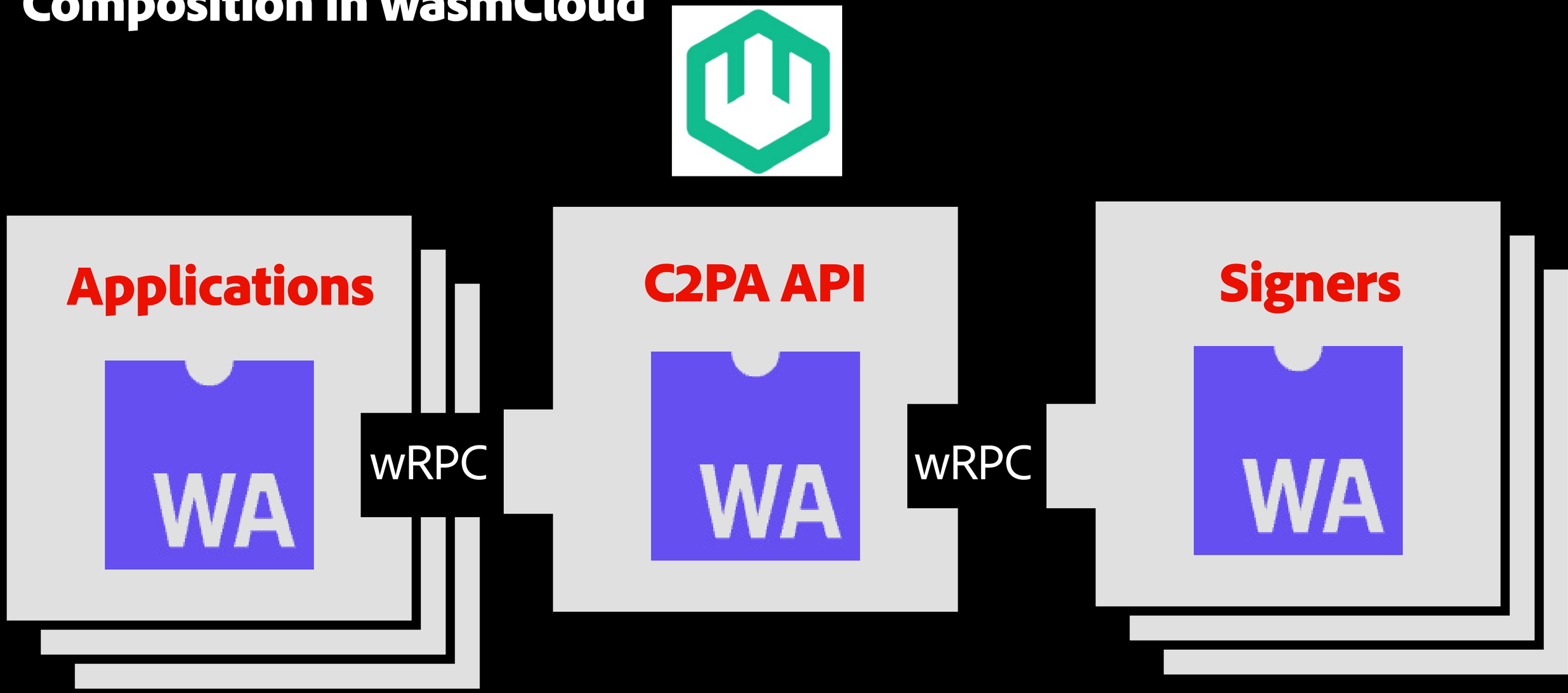
1st class WIT functions

Component support in browsers

Adobe

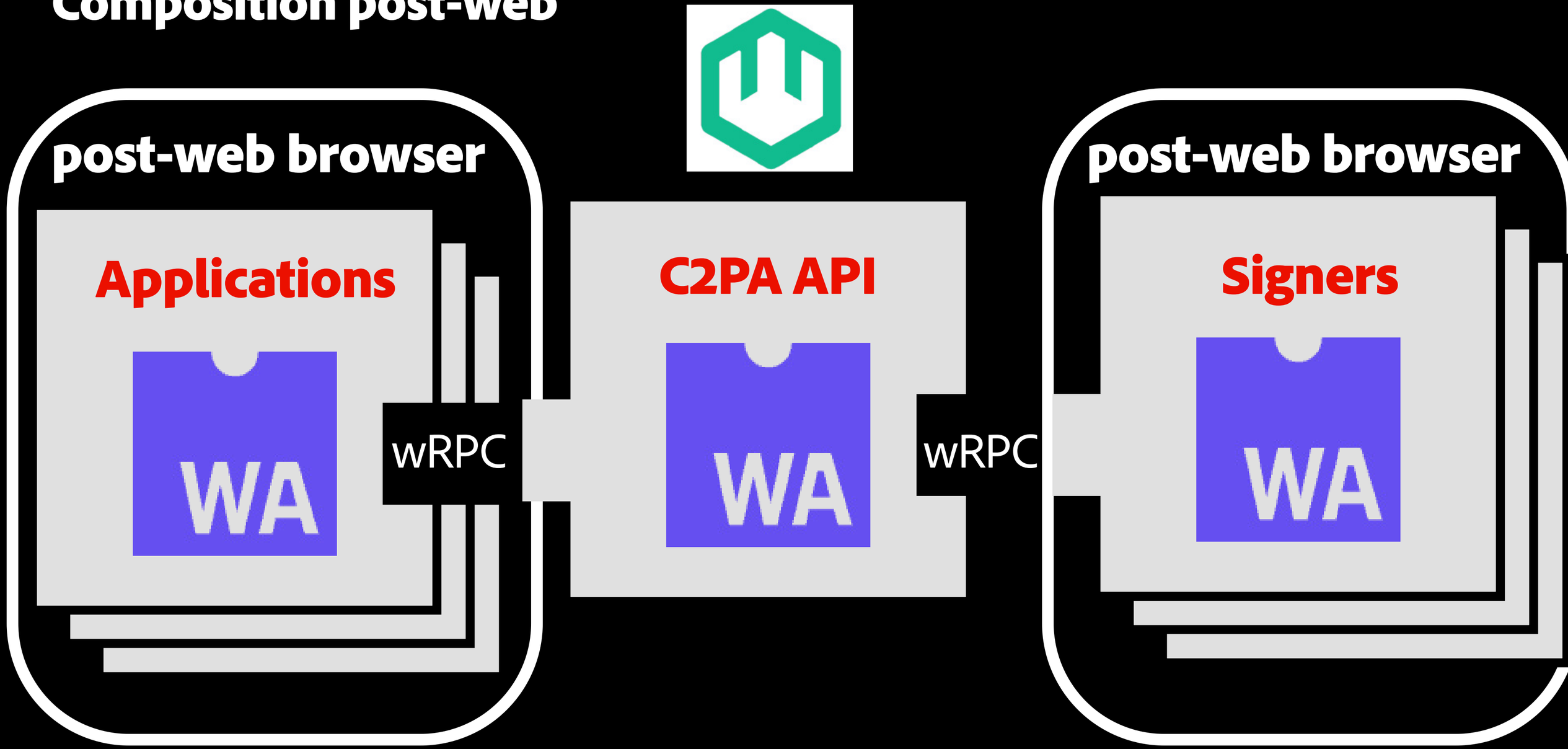
Composition

Composition in wasmCloud



Adobe

Composition post-web



Thanks!

https://github.com/cdmurph32/c2pa-component

Adobe


Adobe