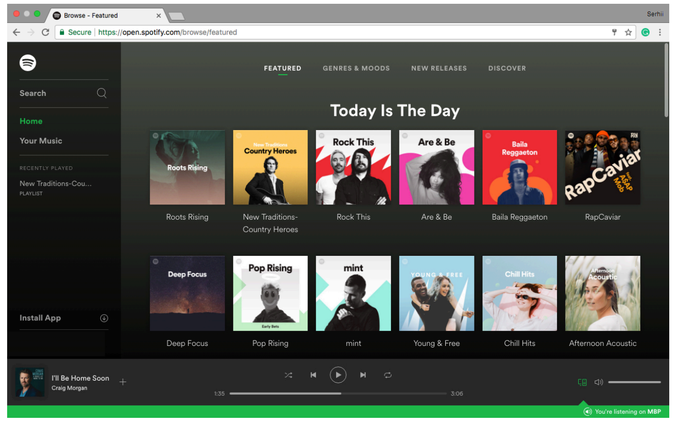
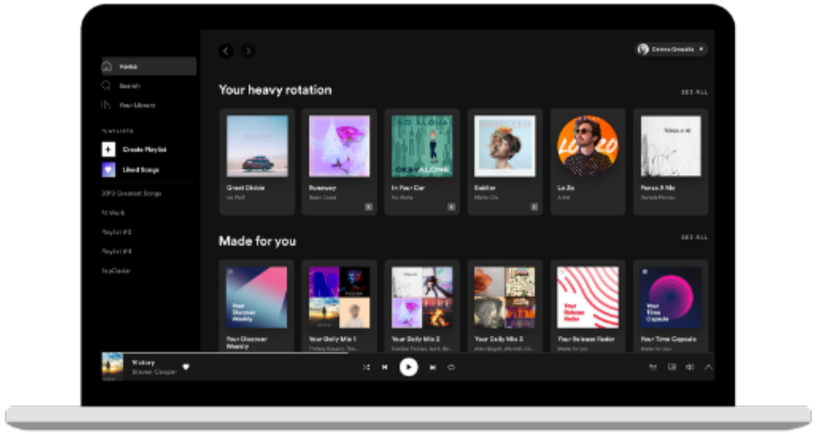
Write once, run everywhere:

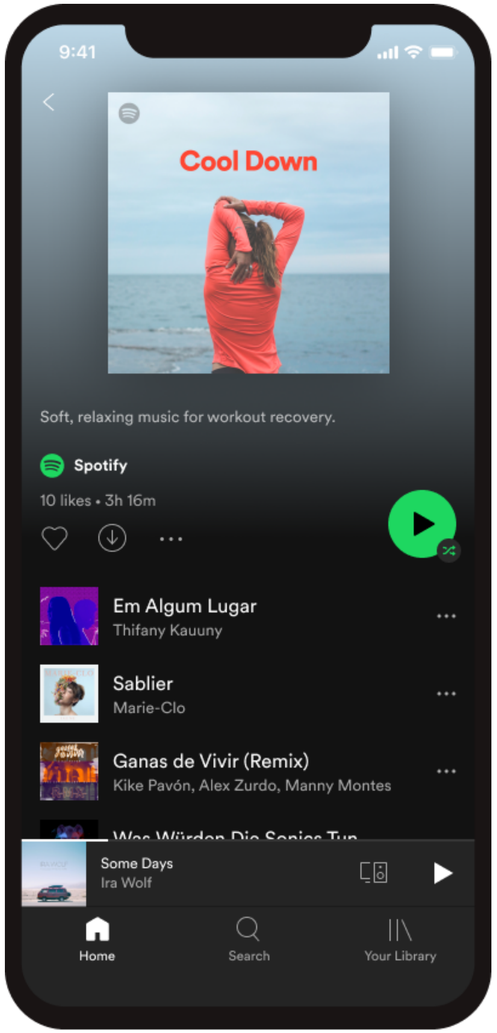
Building apps with Dioxus

Jonathan Kelley - May 10, 2023 @ RustNL

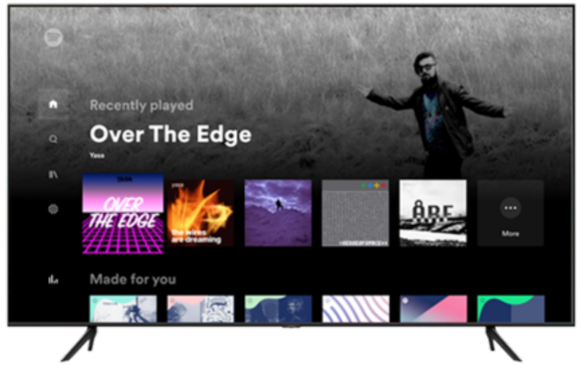
Let’s fix cross-platform app development



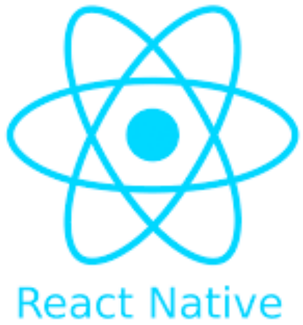


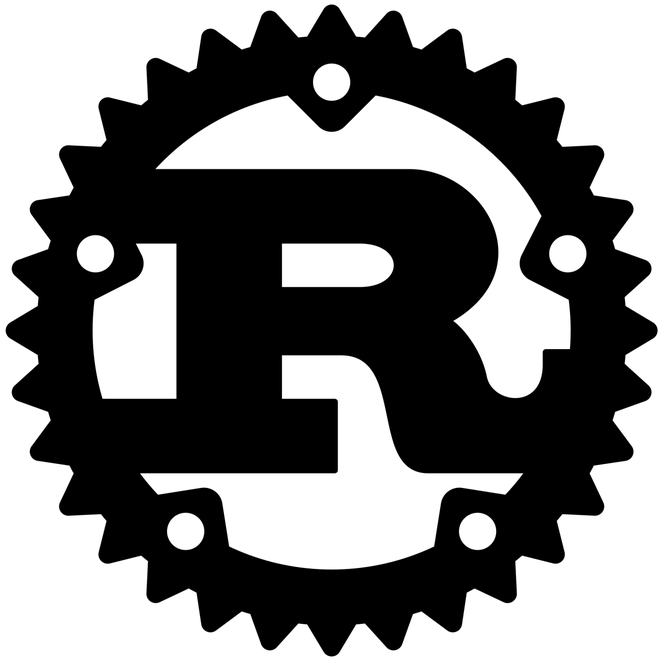


Users Expect Your App Everywhere



What are our options?






RustNL

Azul crate repo

Azul GUI is a free, functional, reactive GUI framework for rapid development of desktop applications written in Rust and C, using the Mozilla WebRender rendering engine.

downloads 7.3k

WebRender

cacao crate repo docs

Rust bindings for AppKit (macOS/Airyx/GNUStep, beta) and UIKit (iOS/tvOS, alpha).

downloads 6.6k

MacOS | iOS

conrod-core crate repo

An easy-to-use, 100% Rust, extensible 2D GUI library.

downloads 102k

Immediate mode API

core-foundation crate repo docs

Bindings to Core Foundation for macOS

downloads 28M

Bindings | MacOS

CXX-Qt crate repo docs

CXX-Qt is a library that automatically generates code to transfer data between Rust and C++ through common interfaces such as QObjects that can be exposed directly into QML. It uses the cxx crate for safe interaction between Rust and C++.

downloads 999

Bindings | Qt | QML

Dioxus crate repo docs

Elegant React-like library for building user interfaces for desktop, web, mobile, SSR, liveview, and more.

downloads 48k

Dominator crate repo docs

Zero-cost ultra-high-performance declarative DOM library using FRP signals

downloads 71k

druid crate repo

Druid is an experimental Rust-native UI toolkit. Its main goal is to offer a polished user experience. There are many factors to this goal, including performance, a rich palette of interactions (hence a widget library to support them), and playing well with the native platform.

downloads 78k

piet

egui crate repo docs

Highly portable immediate mode GUI library in pure Rust

downloads 800k

Immediate mode API

fltk crate repo docs

The FLTK crate is a crossplatform lightweight gui library which can be linked to statically to produce small, self-contained and fast binaries

downloads 119k

Bindings | FLTK

flutter_rust_bridge crate repo docs

High-level memory-safe binding generator for Flutter/Dart <-> Rust

downloads 338k

Bindings

fui-core crate repo docs

MVVM-oriented (properties, observable collections, bindings), ui! macro, multiplatform, renderer-agnostic, with styles.

downloads 1.7k

winit | proc-macro

gladis crate repo docs

Deprecated. Use gtk-rs's CompositeTemplate derive macro instead.

downloads 12k

GTK

GTK crate repo docs

Rust bindings and wrappers for GLib, GDK 3, GTK+ 3 and Cairo.

downloads 1.6M

Bindings | GTK

Iced crate repo docs

A renderer-agnostic GUI library for Rust focused on simplicity and type-safety. Inspired by Elm.

downloads 217k

imgui crate repo docs

High-level Rust bindings to dear imgui

downloads 473k

Bindings | Immediate mode API

iui crate repo docs

Simple, small, easy to distribute GUI bindings.

downloads 16k

Bindings

KAS crate repo docs

The toolKit Abstraction System leverages generics and macros to compose widgets with encapsulated user data, provides a powerful event-handling model and scales to at least 100'000 widgets per window.

downloads 4.7k

winit | WebGPU | proc-macro

lvgl crate repo docs

LVGL bindings for Rust. A powerful and easy-to-use embedded GUI with many widgets, advanced visual effects (opacity, antialiasing, animations) and low memory requirements (16K RAM, 64K Flash).

downloads 2.5k

Bindings | Embedded

Makepad crate repo

Makepad is a new VR, web and native-rendering UI framework.

downloads 277

WebGPU

native-windows-gui crate repo docs

A rust library to develop native GUI applications on the desktop for Microsoft Windows. Native-windows-gui wraps the native win32 window controls in a rustic API

downloads 60k

Bindings | proc-macro | WinApi

OrbTk crate repo docs

The Orbital Widget Toolkit is a multi platform (G)UI toolkit for building scalable user interfaces with the programming language Rust. It's based on the Entity Component System Pattern and provides a functional-reactive API.

downloads 149k

qmetaobject-rs docs

A framework empowering everyone to create Qt/QML applications with Rust. It does so by building QMetaObjects at compile time, registering QML types (optionally via exposing QQmlExtensionPlugins) and providing idiomatic wrappers.

crates.io not found

proc-macro | Bindings | Qt | QML

qt_widgets crate repo docs

Ritual Qt bindings

downloads 19k

Bindings | Qt

relm crate repo docs

Asynchronous, GTK+-based, GUI library, inspired by Elm, written in Rust

downloads 65k

Bindings | GTK

Relm4 crate repo docs

An idiomatic GUI library inspired by Elm and based on gtk4-rs

downloads 24k

GTK | Bindings | MacOS | CSS

rust-qt-binding-generator crate repo docs

Generate code to build Qt applications with Rust

downloads 5.4k

Bindings | Qt | QML

sciter-rs crate repo docs

Rust bindings for Sciter - Embeddable HTML/CSS/script engine (cross-platform desktop GUI toolkit). Also capable with DirectX / OpenGL.

downloads 32k

HTML | CSS

Slint repo docs

Slint is a toolkit to efficiently develop fluid graphical user interfaces for any display: embedded devices and desktop applications. It comes with a fast OpenGL renderer, a designer-friendly markup language and is written in Rust.

downloads 22k

proc-macro | winit | Embedded | MacOS

tauri repo docs

Tauri is a framework for building tiny, blazing fast binaries for all major desktop platforms.

downloads 671k

HTML | MacOS | GTK

vgtk crate repo docs

A declarative UI framework for GTK

downloads 4k

GTK

WebRender crate repo docs

A GPU accelerated 2D renderer for web content

downloads 10k

WebRender

WinSafe crate repo docs

Win32 GUI and related APIs in safe, idiomatic Rust.

downloads 4.9k

Bindings | WinApi

Play Into Rust's Strengths

1. Rust can cross compile basically everywhere.

2. Rust gives us the power to manually manage memory and performance.

3. Rust emphasizes the developer experience.

Dioxus

User interfaces that run anywhere. 🦀

Written in Rust, inspired by React

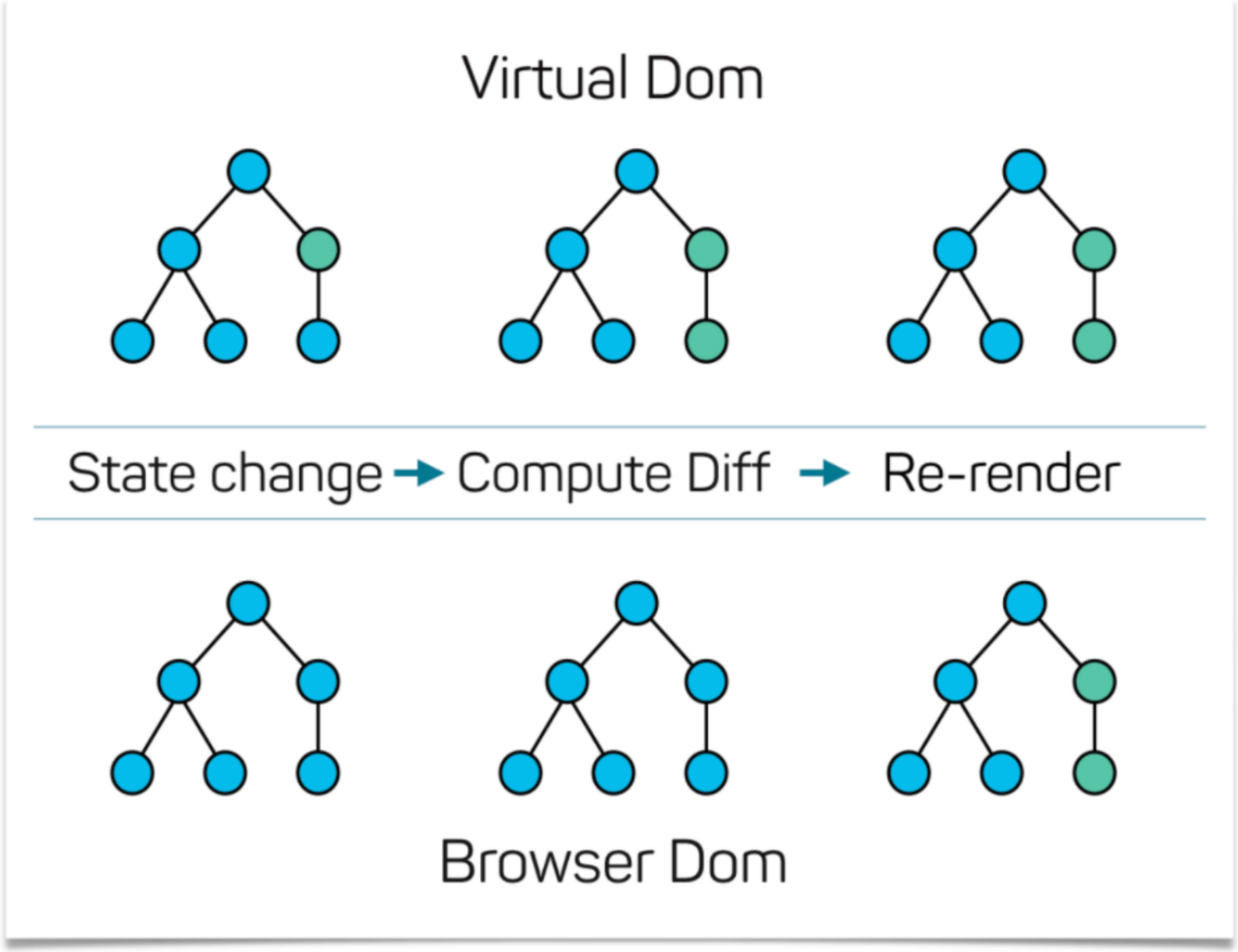
Dioxus

User interfaces that run anywhere. 🦀

Written in Rust, inspired by React

- Declarative and Component-based
- React inspired
- Uses a VirtualDom
- Targets web, desktop, mobile, TUI, liveview, and more
- Exceptional performance
- Integrated hotreloading and dev tools

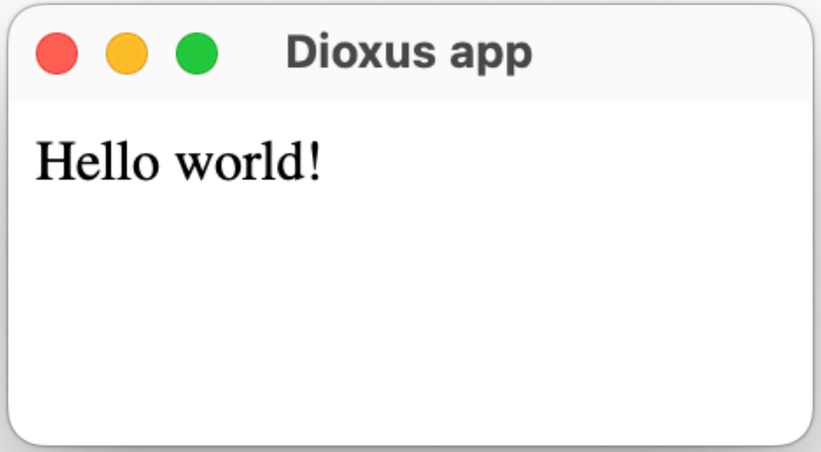
Dioxus uses a VirtualDOM



Demos!

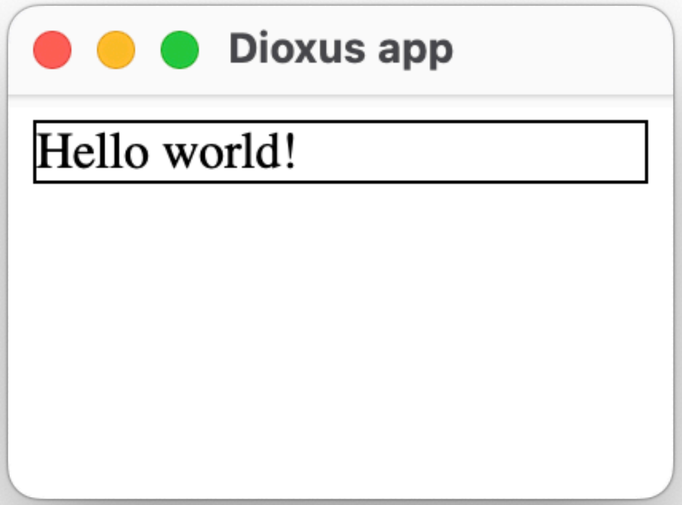
The simplest *hello world*

```rust
use dioxus::prelude::*;

▶ Run | Debug
fn main() {
    dioxus_desktop::launch(app);
}

fn app(cx: Scope) -> Element {
    cx.render(rsx! {
        div { "Hello world!" }
    })
}
```



Elements are just HTML + CSS

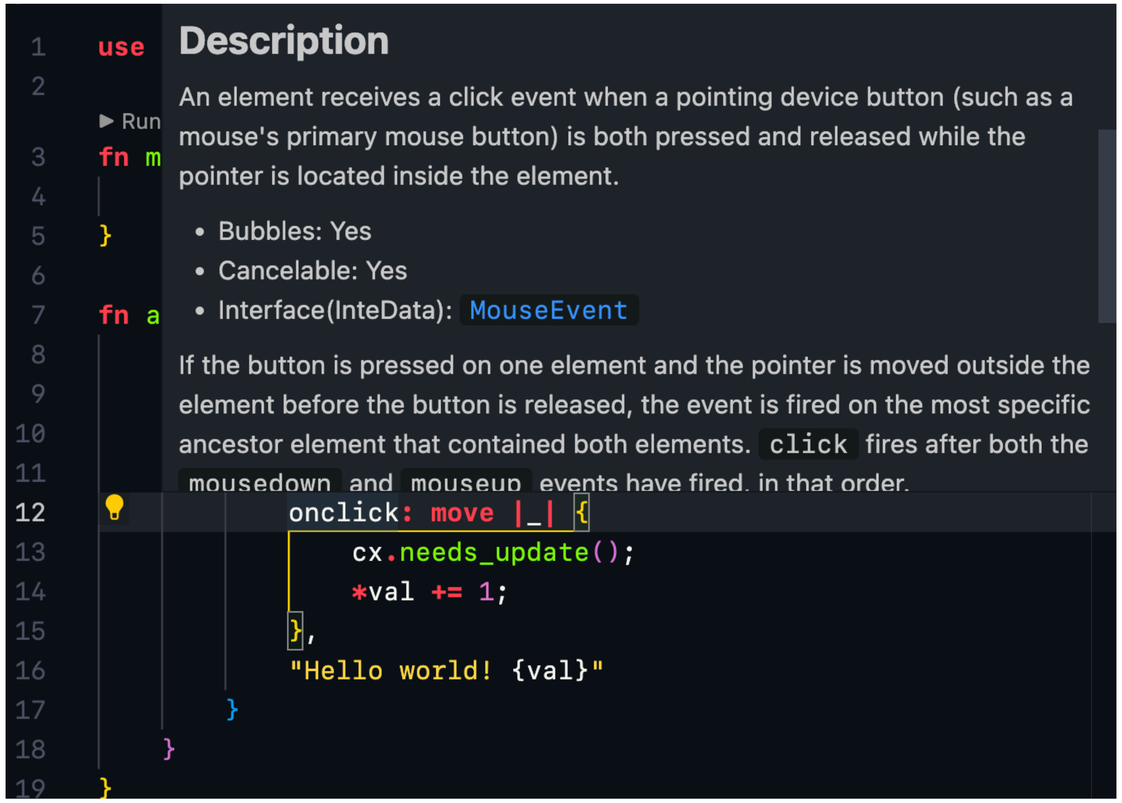
```rust
use dioxus::prelude::*;

▶ Run | Debug
fn main() {
    dioxus_desktop::launch(app);
}

fn app(cx: Scope) -> Element {
    cx.render(rsx! {
        div {
            border: "1px solid black",
            "Hello world!"
        }
    })
}
```

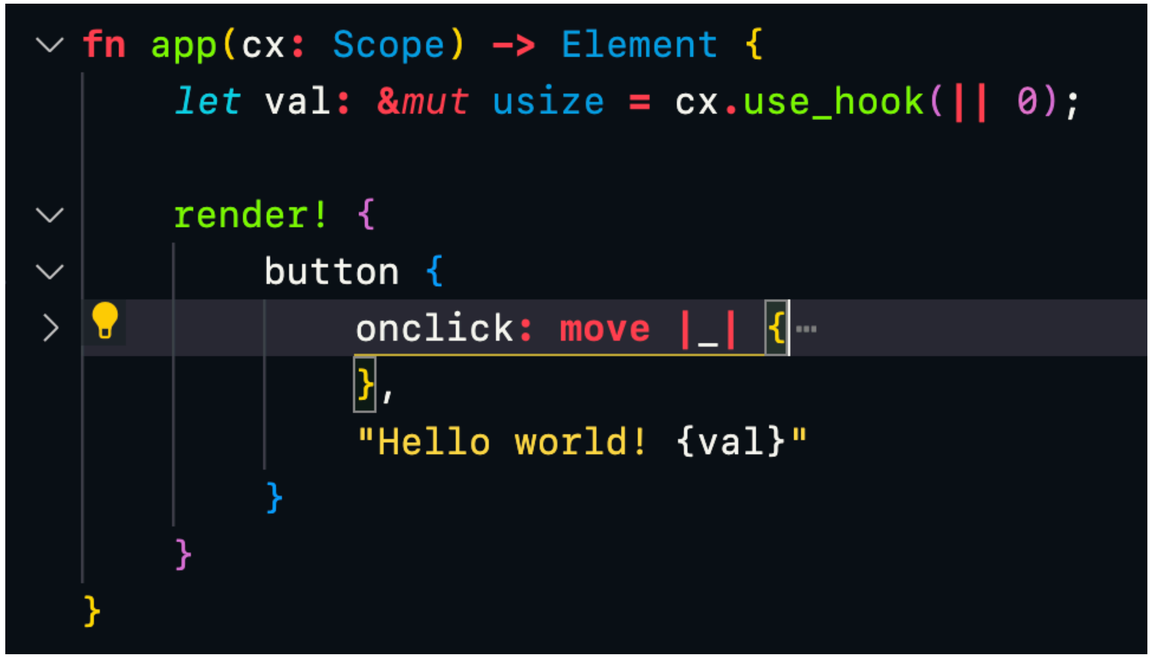


The power of RSX

```rust
fn app(cx: Scope) -> Element {
    cx.render(rsx! {
        h1 { "Hello world!" }
        img { src: "logo.png" }
        ul {
            (0..3).map(|i| rsx! {
                li { "Item {i}" }
            })
        }
        if true {
            rsx! { "Hello world!" }
        }
    })
}
```

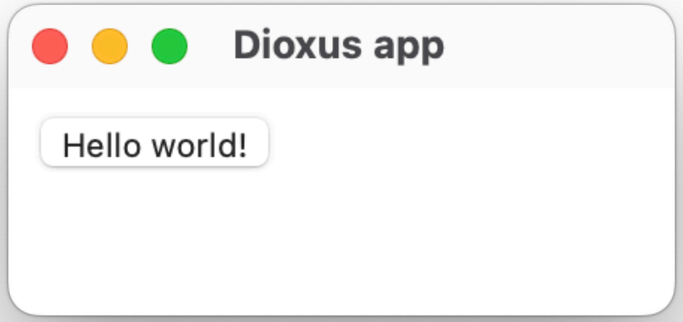
The power of RSX





Components are contained render loops

```rust
fn app(cx: Scope) -> Element {
    println!("Rendering");

    render! {
        button {
            onclick: move |_| println!("Clicked!"),
            "Hello world!"
        }
    }
}
```




```
    Running `target/debug/rustnl-demo`
Rendering
Clicked!
Clicked!
Clicked!
Clicked!
Clicked!
Clicked!
Clicked!
Clicked!
```

Renders need to be scheduled

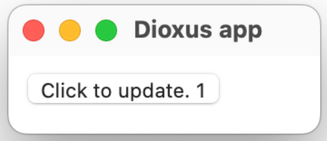
```rust
fn app(cx: Scope) -> Element {
    println!("Rendering");

    render! {
        button {
            onclick: move |_| {
                println!("Clicked!");
                cx.needs_update();
            },
            "Hello world!"
        }
    }
}
```

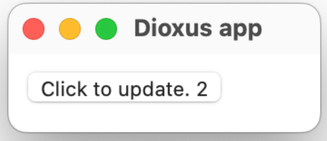
```
    Running `target/debug/rustnl-demo`
Rendering
Clicked!
Rendering
Clicked!
Rendering
Clicked!
Rendering
Clicked!
Rendering
Clicked!
Rendering
```

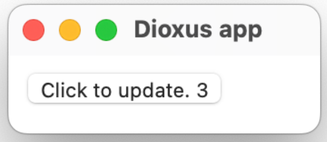
Storing state using hooks

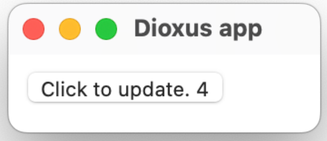
```rust
fn app(cx: Scope) -> Element {
    let count: &mut usize = cx.use_hook(|| 0);
    render! { "{count}" }
}
```

```rust
fn app(cx: Scope) -> Element {
    let count: &mut usize = cx.use_hook(|| 0);

    *count += 1;

    render! {
        button {
            onclick: move |_| cx.needs_update(),
            "Click to update. {count}"
        }
    }
}
```









Primitives compose to fundamental hooks

```rust
fn app(cx: Scope) -> Element {
    let val: &mut usize = cx.use_hook(|| 0);

    render! {
        button {
            onclick: move |_| {
                cx.needs_update();
                *val += 1;
            },
            "Hello world! {val}"
        }
    }
}
```

```rust
pub fn use_state<T: 'static>(
    cx: &ScopeState,
    initial_state_fn: impl FnOnce() -> T,
) -> &UseState<T> {
    let hook = cx.use_hook(move || {
        let current_val = Rc::new(initial_state_fn());
        let update_callback = cx.schedule_update();
        let slot = Rc::new(RefCell::new(current_val.clone()));
        let setter = Rc::new({
            to_owned![update_callback, slot];
            move |new| {
                {
                    let mut slot = slot.borrow_mut();

                    if let Some(val) = Rc::get_mut(&mut slot) {
                        *val = new;
                    } else {
                        *slot = Rc::new(new);
                    }
                }
                update_callback();
            }
        });

        UseState {
            current_val,
            update_callback,
            setter,
            slot,
        }
    });

    hook.current_val = hook.slot.borrow().clone();

    hook
}
```

Child components are just functions

```
7   fn app(cx: Scope) -> Element {
8       render! {
9           div { class: "container",
10              fancy_text {}
11              fancy_text {}
12          }
13      }
14  }
15
16  fn fancy_text(cx: Scope) -> Element {
17      render! {
18          p { class: "fancy-text", "Hello world!" }
19      }
20  }
```

Components can receive properties

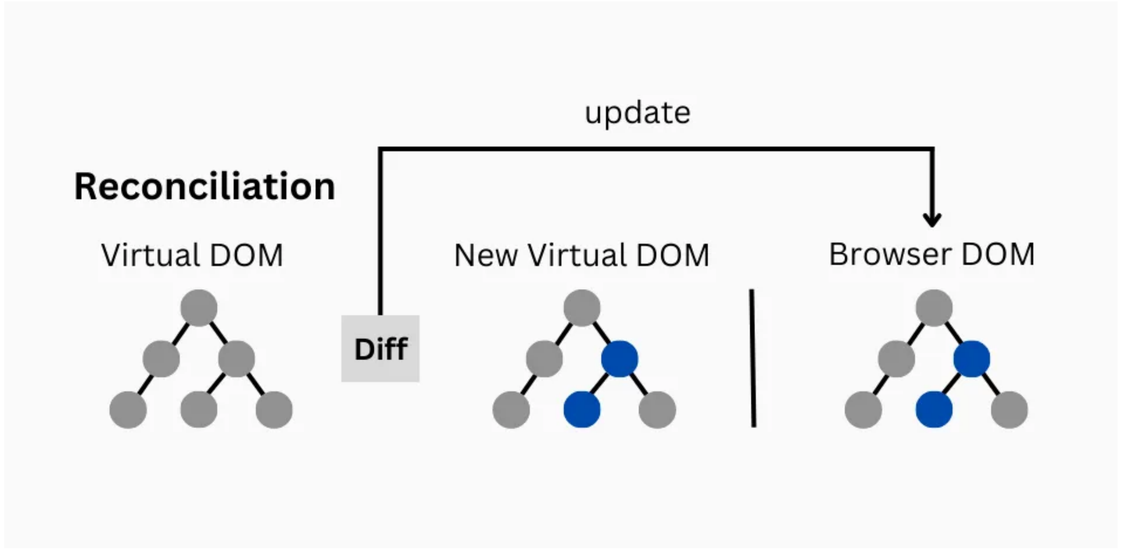
```rust
fn app(cx: Scope) -> Element {
    render! {
        div { class: "container",
            fancy_text { text: "one" }
            fancy_text { text: "two" }
        }
    }
}

#[derive(Props, PartialEq)]
3 implementations
struct FancyTextProps {
    text: &'static str,
}

fn fancy_text(cx: Scope<FancyTextProps>) -> Element {
    render! {
        p { class: "fancy-text", "Hello {cx.props.text}" }
    }
}
```

Components can receive properties

```rust
fn app(cx: Scope) -> Element {
    render! {
        div { class: "container",
            fancy_text { text: "one" }
            fancy_text { text: "two" }
        }
    }
}

#[inline_props]
fn fancy_text(cx: Scope, text: &'static str) -> Element {
    render! {
        p { class: "fancy-text", "Hello {text}" }
    }
}
```

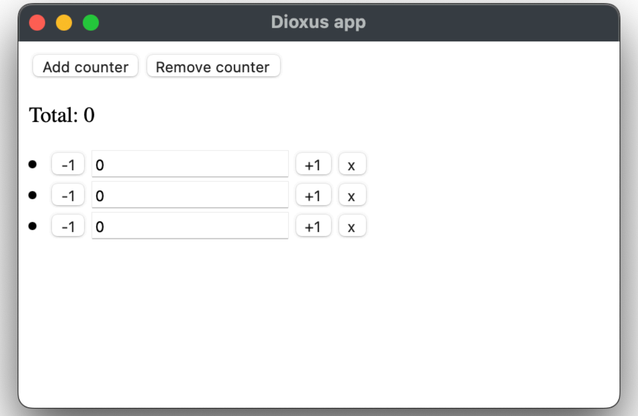
Take-away innovations

Batched Mutations



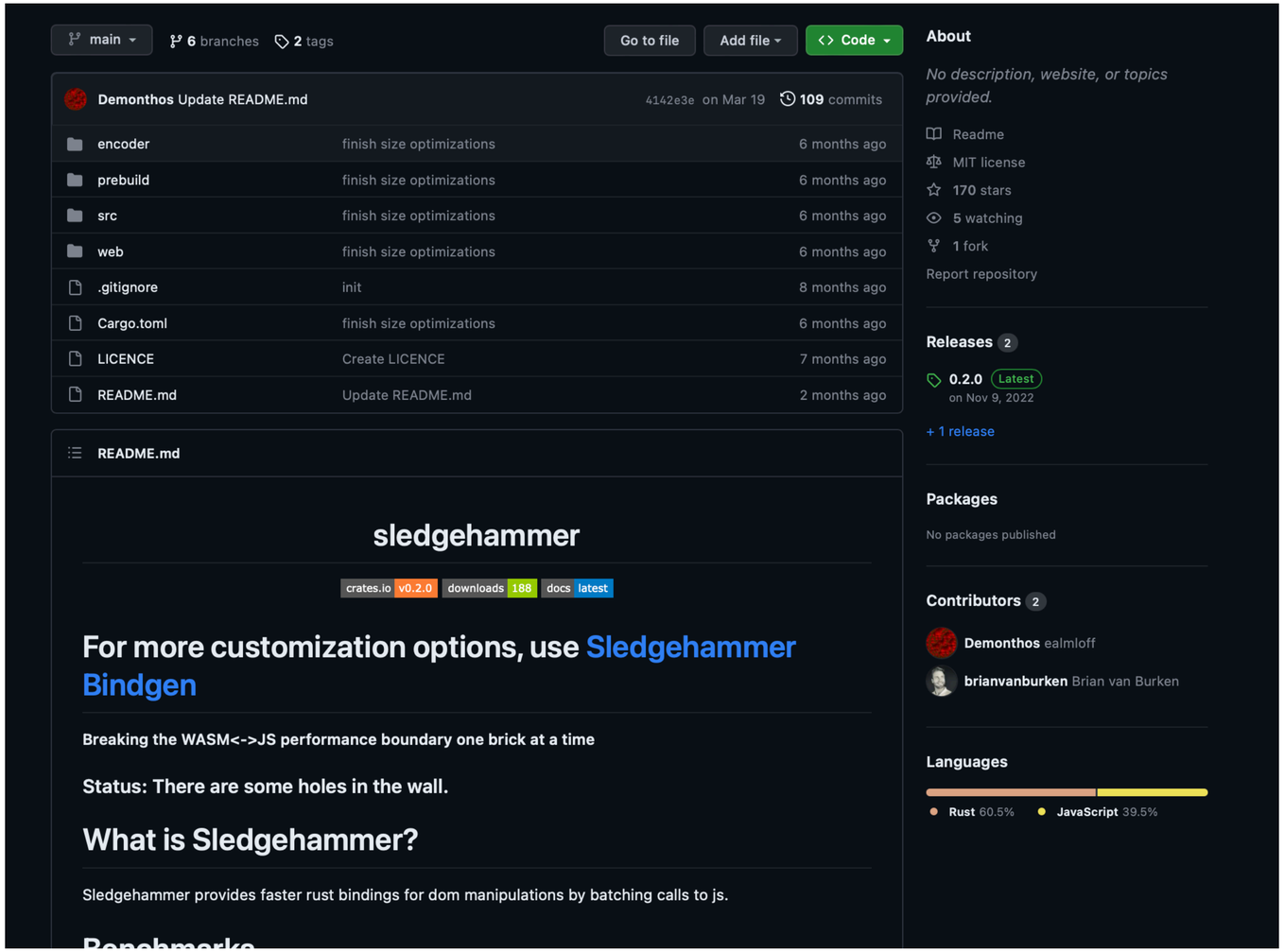

```
[
    // create template
    CreateElement { name: "div" },
    CreateElement { name: "div" },
    CreateStaticText { value: "Hello, world!" },
    CreateElement { name: "div" },
    CreateElement { name: "div" },
    CreateStaticPlaceholder {},
    AppendChildren { m: 1 },
    AppendChildren { m: 1 },
    AppendChildren { m: 2 },
    AppendChildren { m: 1 },
    SaveTemplate { name: "template", m: 1 },
    // The fragment child template
    CreateStaticText { value: "hello" },
    CreateStaticText { value: "world" },
    SaveTemplate { name: "template", m: 2 },
]
```

FFI



- Crossing the FFI Boundary (IPC, Network) is Expensive
- Batching mutations massively increases performance
- Interning values at the FFI boundary is also effective

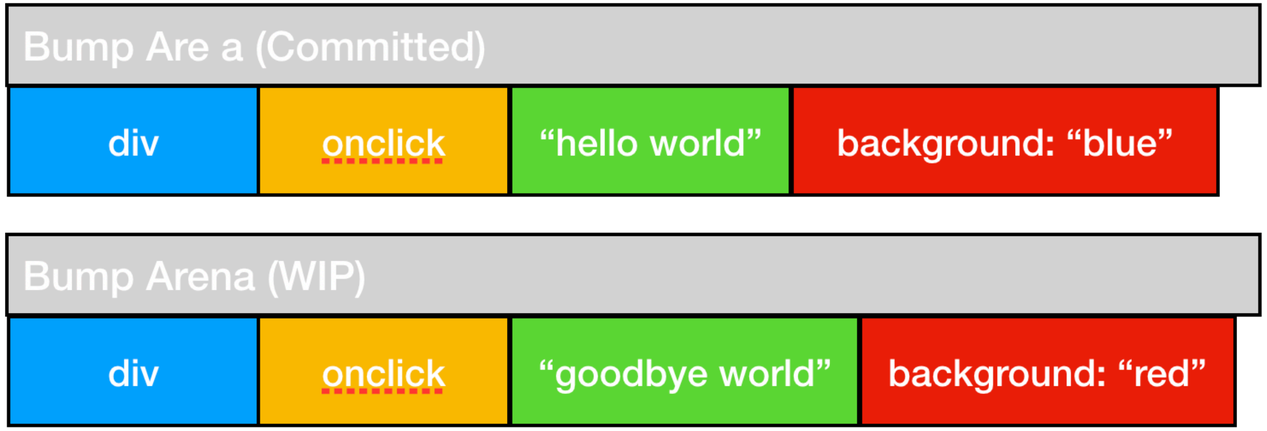
Batched Mutations



Name Duration for...	sledge-hammer-v1.0.0	vanillajs	wasm-bindgen-v0.2.47
Implementation notes	772	772	772
create rows creating 1,000 rows (5 warmup runs).	59.8 ± 0.4 (1.03)	59.4 ± 0.7 (1.02)	62.0 ± 0.6 (1.06)
replace all rows updating all 1,000 rows (5 warmup runs).	64.0 ± 0.7 (1.02)	62.5 ± 1.0 (1.00)	64.0 ± 1.0 (1.02)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	223.8 ± 7.3 (1.06)	226.5 ± 7.2 (1.07)	224.6 ± 7.7 (1.06)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	23.2 ± 0.9 (1.06)	21.9 ± 0.5 (1.00)	23.8 ± 0.5 (1.09)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	40.6 ± 1.1 (1.02)	42.3 ± 1.7 (1.06)	41.3 ± 1.5 (1.03)
remove row removing one row. (5 warmup runs). 4x CPU slowdown.	71.3 ± 1.9 (1.06)	71.9 ± 3.1 (1.07)	76.1 ± 2.2 (1.13)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	656.4 ± 2.0 (1.02)	645.4 ± 1.9 (1.00)	684.6 ± 2.5 (1.06)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	137.8 ± 1.6 (1.01)	141.2 ± 2.3 (1.04)	143.1 ± 3.3 (1.05)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	51.9 ± 2.6 (1.00)	53.8 ± 1.7 (1.04)	55.1 ± 1.6 (1.06)
geometric mean of all factors in the table	1.03	1.03	1.06
compare: Green means significantly faster, red significantly slower	compare	compare	compare

Bump allocator for state

- Creating massive trees of elements puts lots of pressure on the global allocator
- Swapped out the global allocator for a local bump allocator that gets reset between renders
- Bump allocators are extremely fast!




```rust
fn borrows<'a>(cx: Scope<'a>) -> Element<'a> {
    let mut val: &'a UseState<i32> = use_state(cx, || 0);

    render! {
        button { onclick: move |_| val += 1 }
        button { onclick: move |_| val += 1 }
        button { onclick: move |_| val += 1 }
        button { onclick: move |_| val += 1 }
    }
}
```

Template-based architecture

- Macros let us optimize static and dynamic elements.
- We can optimize away the static portions of the rsx.
- Templates can then be cached at the FFI boundary layer.

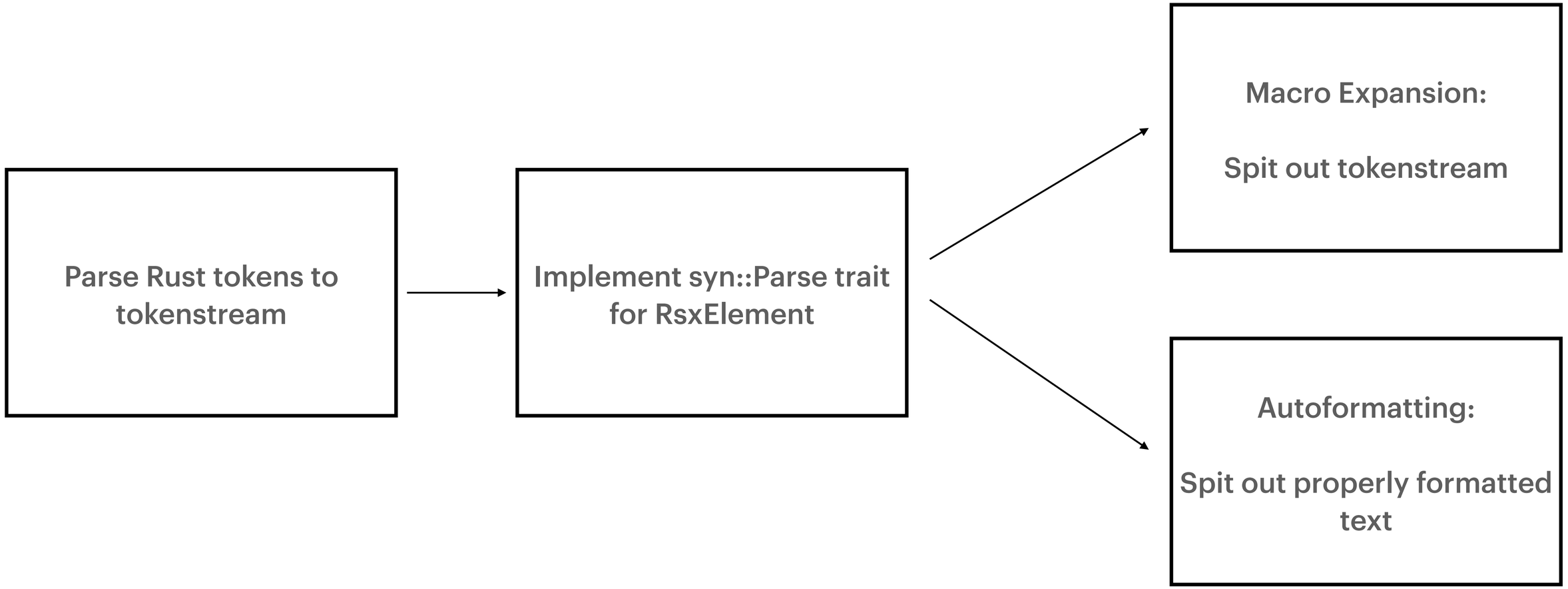
```
rsx! {
    div {
        h1 {"Glorious Counter"}
        p { "Count: {val}" }
        button { onclick: move |_| val += 1, "Increment" }
        button { onclick: move |_| val -= 1, "Decrement" }
    }
}
```

```
static THIS_TEMPLATE: Template = Template { /* */ };

VNode {
    template: THIS_TEMPLATE,
    dynamic_nodes: [
        Text(format_args!("Count: {val}"))
    ]
}
```

Templates architecture

Name Duration for...	vanillajs-1	vanillajs	solid-v1.5.4	sledge-hammer-v1.0.0	wasm-bindgen-v0.2.47	dioxus-v0.3.0	leptos-v0.0.3	sycamore-v0.8.0	yew-v0.19.3
Implementation notes	772	772		772	772				
create rows creating 1,000 rows (5 warmup runs).	60.5 ± 0.7 (1.04)	60.1 ± 1.1 (1.04)	58.3 ± 0.6 (1.00)	61.8 ± 1.2 (1.07)	65.6 ± 0.6 (1.13)	62.8 ± 0.3 (1.08)	67.5 ± 0.7 (1.16)	72.5 ± 0.7 (1.25)	89.9 ± 0.4 (1.55)
replace all rows updating all 1,000 rows (5 warmup runs).	63.3 ± 0.7 (1.03)	62.6 ± 0.9 (1.01)	65.7 ± 0.9 (1.07)	64.1 ± 0.5 (1.04)	67.2 ± 1.0 (1.09)	68.2 ± 0.7 (1.10)	73.1 ± 0.5 (1.18)	81.1 ± 1.3 (1.31)	101.4 ± 0.8 (1.64)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	229.2 ± 10.5 (1.08)	233.7 ± 8.5 (1.10)	211.9 ± 5.9 (1.00)	233.7 ± 7.0 (1.10)	240.7 ± 7.1 (1.14)	228.7 ± 10.6 (1.08)	238.5 ± 7.6 (1.13)	256.7 ± 8.4 (1.21)	266.4 ± 6.5 (1.26)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	23.3 ± 0.6 (1.05)	23.9 ± 0.7 (1.07)	23.9 ± 0.7 (1.07)	24.5 ± 0.8 (1.10)	24.9 ± 1.0 (1.12)	30.6 ± 1.4 (1.37)	27.1 ± 1.0 (1.21)	30.1 ± 0.7 (1.35)	40.6 ± 1.5 (1.82)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4x CPU slowdown.	40.7 ± 1.3 (1.01)	41.1 ± 1.2 (1.02)	41.6 ± 1.9 (1.04)	41.1 ± 1.3 (1.02)	41.2 ± 1.3 (1.03)	42.5 ± 1.1 (1.06)	43.2 ± 1.4 (1.08)	42.6 ± 2.9 (1.06)	44.8 ± 1.7 (1.12)
remove row removing one row. (5 warmup runs). 4x CPU slowdown.	73.6 ± 2.3 (1.03)	71.7 ± 1.7 (1.00)	76.0 ± 1.7 (1.06)	72.6 ± 1.9 (1.01)	71.8 ± 1.3 (1.00)	76.9 ± 2.7 (1.07)	78.7 ± 2.8 (1.10)	73.6 ± 2.9 (1.03)	75.5 ± 2.1 (1.05)
create many rows creating 10,000 rows. (5 warmup runs with 1k rows).	681.1 ± 2.2 (1.02)	677.3 ± 3.0 (1.02)	665.0 ± 2.2 (1.00)	692.0 ± 2.7 (1.04)	720.0 ± 1.6 (1.08)	732.8 ± 4.2 (1.10)	770.7 ± 4.5 (1.16)	936.9 ± 6.9 (1.41)	1,326.5 ± 8.3 (1.99)
append rows to large table appending 1,000 to a table of 10,000 rows. 2x CPU slowdown.	145.6 ± 1.8 (1.07)	146.6 ± 2.0 (1.08)	136.2 ± 3.9 (1.00)	149.3 ± 2.0 (1.10)	153.9 ± 4.7 (1.13)	159.1 ± 3.8 (1.17)	165.6 ± 4.4 (1.22)	181.6 ± 2.5 (1.33)	229.1 ± 3.5 (1.68)
clear rows clearing a table with 1,000 rows. 8x CPU slowdown. (5 warmup runs).	55.4 ± 1.2 (1.04)	55.3 ± 2.0 (1.04)	62.3 ± 1.3 (1.17)	55.7 ± 1.4 (1.05)	54.3 ± 2.4 (1.02)	64.3 ± 1.8 (1.21)	71.4 ± 1.6 (1.34)	71.9 ± 1.4 (1.35)	139.5 ± 2.8 (2.63)
geometric mean of all factors in the table	1.04	1.04	1.04	1.06	1.08	1.14	1.17	1.25	1.58
compare: Green means significantly faster, red significantly slower	compare	compare	compare	compare	compare	compare	compare	compare	compare

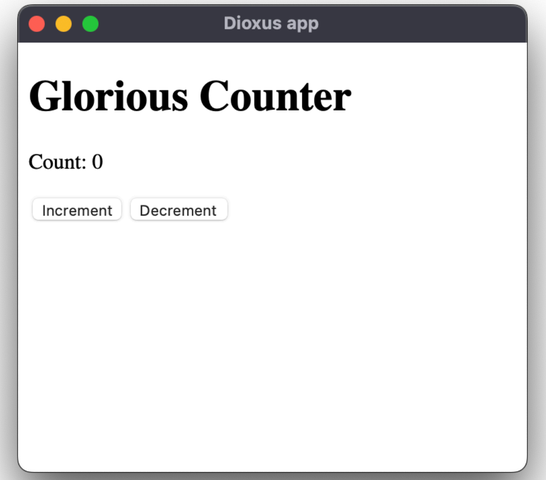
Autoformatting of RSX



Hot-reloading of RSX

```
10  rsx! {
11      div {
12          h1 {"Glorious Counter"}
13          p { "Count: {val}" }
14          button { onclick: move |_| val += 1, "Increment" }
15          button { onclick: move |_| val -= 1, "Decrement" }
16      }
17  }
```

TemplateId: main.rs/10:1



Thanks!

Jonathan Kelley - May 10, 2023 @ RustNL
@jkelleyrtp
dioxuslabs.com