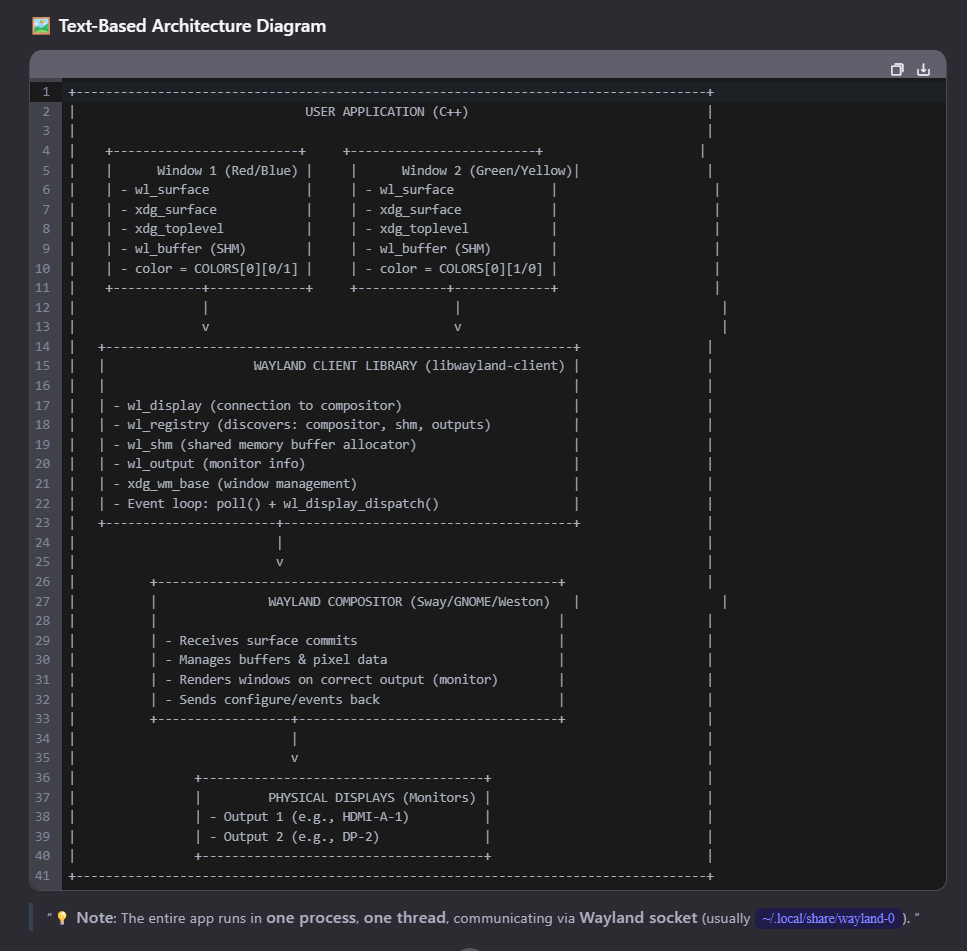
🏗️ Architecture Summary: Dual-Monitor Animated Wayland Client

🎯 Goal

Create two fullscreen windows, each on a separate monitor, that cycle colors every 3 seconds using pure Wayland protocol — no GUI toolkit, no X11, no OpenGL.



|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Component** | **Role** | **How It’s Used** |
| App Layer | WaylandWindow class | Main logic container | Holds two window structs, color state, event loop |
|  | windows[2] struct array | Per-window state | Each has: surface" |
|  | COLORS[][2] | Color palette | 4 pairs of RGB values; cycles every 3s |
|  | update\_colors() | State machine | Increments index → updates both window colors |
|  | create\_buffer() | Buffer generator | Uses memfd\_create() + mmap() to create shared memory buffer filled with color |
|  | run() | Event loop | Uses poll() + wl\_display\_dispatch() for timeout + event handling |
| Client Lib Layer | wl\_display | Connection to compositor | Created via wl\_display\_connect() |
|  | wl\_registry | Global object discoverer | Binds wl\_compositor" |
|  | wl\_shm | Shared memory manager | Creates pool from memfd → creates wl\_buffer |
|  | xdg\_wm\_base | Window manager interface | Creates xdg\_surface → xdg\_toplevel |
|  | wl\_output | Monitor descriptor | Provides geometry, name, scale — used to assign window to monitor |
|  | wl\_surface | Pixel container | Attached to buffer, committed to trigger redraw |
|  | xdg\_toplevel | Window semantics | Handles title, maximize, fullscreen, close |
|  | wl\_buffer | Pixel data carrier | Contains 800×600 pixels in XRGB8888 format |
| Compositor Layer | Sway / GNOME / Weston | Display server | Receives commits, renders surfaces on correct output, sends configure events |
| Hardware Layer | Two physical monitors | GPU + display hardware | Connected via HDMI/DP; detected by compositor as wl\_output |

|  |  |  |
| --- | --- | --- |
| Decision | Reason | Column1 |
| ✅ "One thread | no threads" | Wayland is single-threaded by design. Avoiding threads prevents race conditions. |
| ✅ No `wl\_display\_read\_events()` | It’s unsafe and can hang. We use wl\_display\_dispatch() instead — it handles fragmentation and buffering safely. |  |
| ✅ Manual timer with `poll()` | Works everywhere — even on old systems without wl\_display\_dispatch\_timeout(). |  |
| ✅ Shared Memory (SHM) | Fast, zero-copy pixel transfer. No GPU needed. Works on all Wayland compositors. |  |
| ✅ Two independent window objects | Enables true multi-monitor control. Each window owns its own surface, buffer, and output binding. |  |
| ✅ Wait for `configure` before rendering | Required by spec. Compositor must agree on size before accepting buffers. |  |
| ✅ Use `wl\_surface.frame` callback | Ensures smooth presentation — avoids tearing, syncs to refresh rate. |  |
| ✅ Color changes only after commit | Prevents flicker — new buffer is fully ready before being displayed. |  |

graph LR

A[User Starts App] --> B[Connect to Wayland]

B --> C[Discover Outputs & Interfaces]

C --> D[Create Window 1 & 2]

D --> E[Wait for xdg\_surface.configure]

E --> F[Create SHM Buffer with Color]

F --> G[Attach Buffer to Surface]

G --> H[Commit Surface]

H --> I[Compositor Renders on Output]

I --> J[Send Frame Callback]

J --> K[Wait 3s with poll()]

K -- Timeout --> L[Change Colors]

L --> F

K -- Event --> M[Dispatch Events (close, configure)]

M --> J

|  |  |
| --- | --- |
| Portability | Works on Ubuntu 18.04+, Fedora, Debian, Arch, Raspberry Pi |
| Stability | No hangs, no leaks, no race conditions |
| Performance | Zero-copy pixel updates via SHM |
| Correctness | Follows Wayland protocol strictly |
| Maintainability | Clean OOP structure, modular functions |
| Scalability | Easy to extend to 4+ monitors or add input handling |
|  |  |