Wayland and Qt

Wayland is a display server protocol that helps you to create multi-process systems. Multiple client applications (“clients”) render their own contents to off-screen buffers. These buffers are then passed to a compositor process (“the compositor”) using the Wayland protocol. Finally, the compositor composites and positions the content on a physical display.

Why Use Wayland instead of X11 or Custom

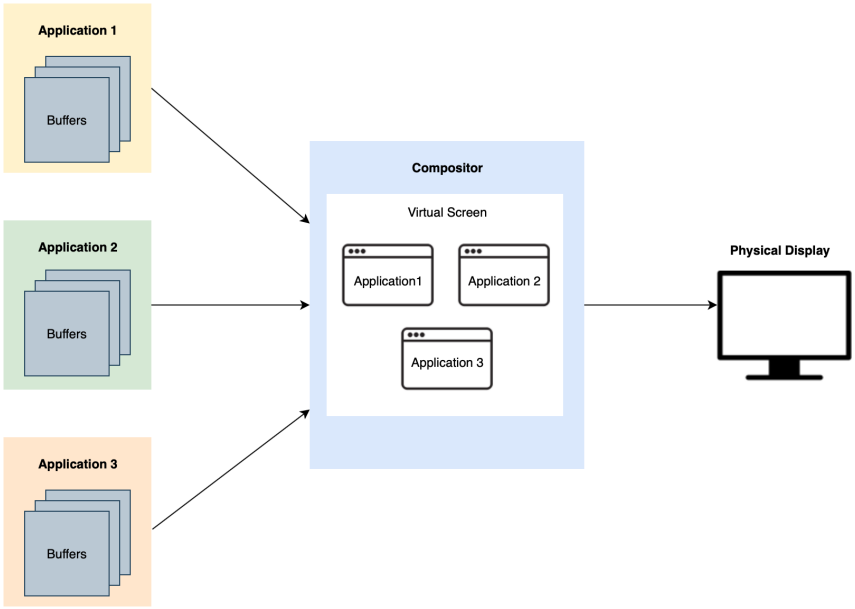
X11, a desktop protocol from the 80s, no longer fits with how modern graphics hardware works today. It is large, complex, and lacks customisability. In fact, it is difficult to run a client fluidly with X11 and reach 60fps without tearing. Wayland, in contrast, is easier to implement, has better performance, and contains all the necessary parts to run efficiently on modern graphics hardware. For embedded, multi-process systems on Linux, Wayland is the standard.

However, if you are working with old hardware or legacy applications, then Wayland may not be a good option. Wayland, due to its focus on security and isolation, lets the compositor decide how and where to render the multi-process clients. While this is beneficial, it is not the case for some old clients. Particularly, if these old clients need to influence how and where they are rendered.

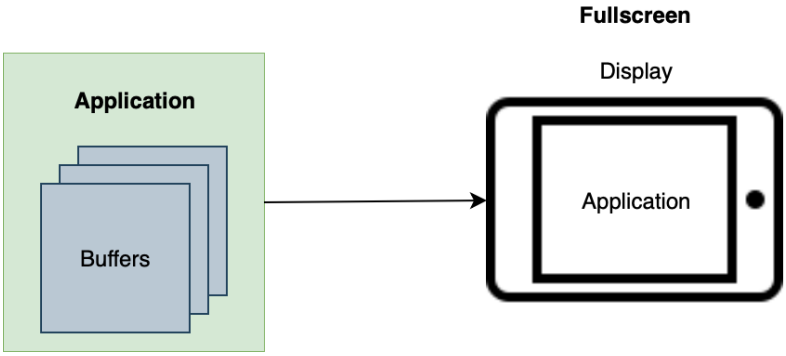
Back when X11 was very popular, developers wrote their own custom solutions to circumvent X11 issues. Older Qt versions had the Qt Windowing System (QWS), which is now discontinued. These days, more custom solutions are being ported over to Wayland.

Why Use Multi-Process

In a single-process system, all clients run in one, single process. In a multi-process system, all clients run in their own, dedicated process. With Qt, at any point in your development process, you can choose to switch between single-process and multi-process.



MULTI-PROCESS CLIENT ARCHITECTURE



SINGLE-PROCESS CLIENT ARCHITECTURE

The use of multi-process has the following benefits:

- Stability
- Security
- Performance
- Interoperability

Stability	
Easier to recover when clients hang or crash	If you have a complex UI, then multi-process is useful because if one part of the UI crashes, it doesn't affect the entire system. Similarly, the display won't freeze, even when one client freezes. This way, you can ensure that critical clients that must be present for safety reasons, are always rendered. Note: If your client is mandated by law to render safety-critical information, consider using [Qt Safe Renderer].
Protection against possible memory leaks	In a multi-process system, if one client has a memory leak and consumes lots of memory, that memory is recovered when that client exits. In contrast with single-process, the memory leak remains until the entire system restarts.

Security
In a single-process system, all clients can access each other's memory. For example, there's no isolation for sensitive data transfer; every line of code must be equally trustworthy. This isolation is available, by design, in multi-process systems.

Performance
If you have a CPU with multiple cores, multi-process systems can be more efficient to run. You can run each client on separate cores.

Interoperability
You can interface non-Qt clients in a multi-process system, as long as your clients understand Wayland or X11. For example, if you use gstreamer for video or you have a maps application, you can run these clients alongside your other Qt-based clients.

Possible Trade-Offs with Multi-Process

Use of multi-process systems do bring about the following trade-offs:

Increased video memory consumption
This can be a constrain for Embedded devices. In multi-process, the content of each client is already cached for you. Additionally, every client has its own frame buffer, so you use more video memory.

Increased main memory consumption
On an OS level, there is an overhead with running separate clients as they use more main memory. For example, if you run QML, it requires a separate QML engine. If you run a single client that uses Qt Quick Controls, it's loaded once. If you then split this client into multiple clients, you're loading these Qt Quick Controls multiple times, resulting in a higher startup cost to initialise your clients.

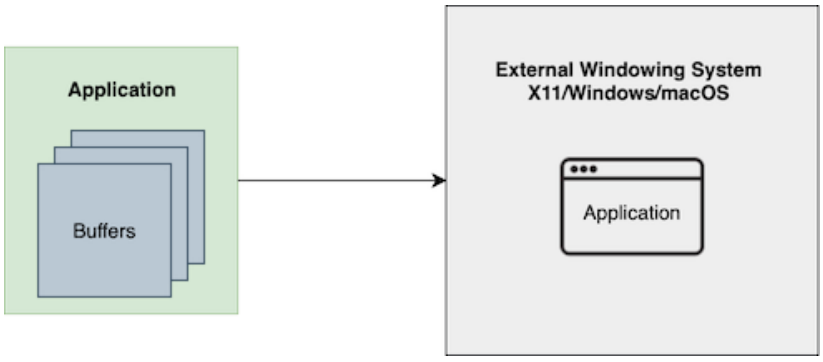
Repeated storage of graphical resources
In a single-process system, if you're using the same textures, background, or icons in many places, those images are only stored once. In contrast, if you use these images in a multi-process system, then you have to store them multiple times. In this case, one solution is to share graphical resource between clients. While Qt has mechanisms to share graphical resources between clients, an equivalent for Qt Wayland is currently being developed.

What Qt Wayland Offers

For Clients

Any Qt program can run as a Wayland client (multi-process) or a standalone client (single-process). This is determined on startup, where you can choose between the different backends. During the development process, you can develop the client on the desktop first, then test it on the target hardware later. You don't need to run your clients on the actual target hardware, all the time.

If you develop on a Linux machine, you can run your compositor built on that machine. You can run your client as a Wayland client, with each client inside the compositor. Then, it resembles your hardware. Without rebuilding the client, you can also run it with "-platform wayland" to run it in inside the compositor. If you use "-platform xcb" (for x11), you can run the client on desktop. Ultimately, you can run your clients before the compositor is ready for use.



SINGLE-PROCESS CLIENT DEVELOPMENT

For Servers

The server, or the compositor, connects to the display and shows the contents of each client on the screen. The compositor handles input and sends input events to the corresponding client. In turn, each client connects to a compositor and sends the content of its windows to the compositor. It is up to the compositor to decide:

- How and where to show the content
- Which content to show
- What to do with the different client graphics buffers

The compositor can also do adventurous things like make a 3D scene, show windows on the walls, on a VR, map them to a sphere, and so on.

Wayland provides a reference compositor, known as Weston, that can also be used in production. Qt clients run on any Wayland compositor, including Weston.