The Enlightenment of Wayland

The story of Enlightenment, EFL, Tizen & Wayland

Carsten Haitzler

c.haitzler@samsung.com
raster@rasterman.com

What

What is... ?

- Tizen
  - A Linux distribtion for Consumer Electronics
    - Mobile
      - Samsung Z1, Z3
    - Wearables
      - Samsung Gear 2, Gear 2 Neo, Gear S, Gear S2
    - TV
      - Samsung Smart TVs 2016 and beyond (also part of 2015)
    - Fridges
      - Samsung Smart Fridge
    - ... and more
  - Open Source - http://source.tizen.org

What is... ?

- Enlightenment
  - A window manager, compositor and desktop shell for X11
    - Now ... also for Wayland
  - Window manager and compositor for Tizen
    - On both X11 and now Wayland

- EFL
  - Enlightenment Foundation Libraries
  - The libraries built to make Enlightenment and other applications
    - LGPLv2 + BSD Licensing
  - Libraries behind Tizen native development and core apps and tools

https://www.enlightenment.org

What is... ?

- Wayland
  - Replaces X11
  - A new display system protocol
  - A new set of client and server libraries to build display servers with
  - A set of conventions clients and servers agree to
  - Primarily focused on Linux
  - Built around the assumption of open drivers
    - Using DRM/KMS etc.
  - Focus on "every frame is perfect"
  - Focus on security and application isolation
  - Merges Display Server, Window Manager and Compositor into one

http://wayland.freedesktop.org

Why Wayland?

- It's cool
- Everyone else is doing it

Why Wayland?

- It's cool
- Everyone else is doing it

But really ...

Why Wayland?

- It's cool
- Everyone else is doing it

But really ...

- Wayland is ...
  - Free of legacy design issues X11 has to maintain
  - Smaller codebase than X11
  - Easier to get a “perfect UI” in than X11
  - Easier to support hardware display features than X11
  - More secure than X11

Why Wayland?

- It's cool
- Everyone else is doing it

  But really ...

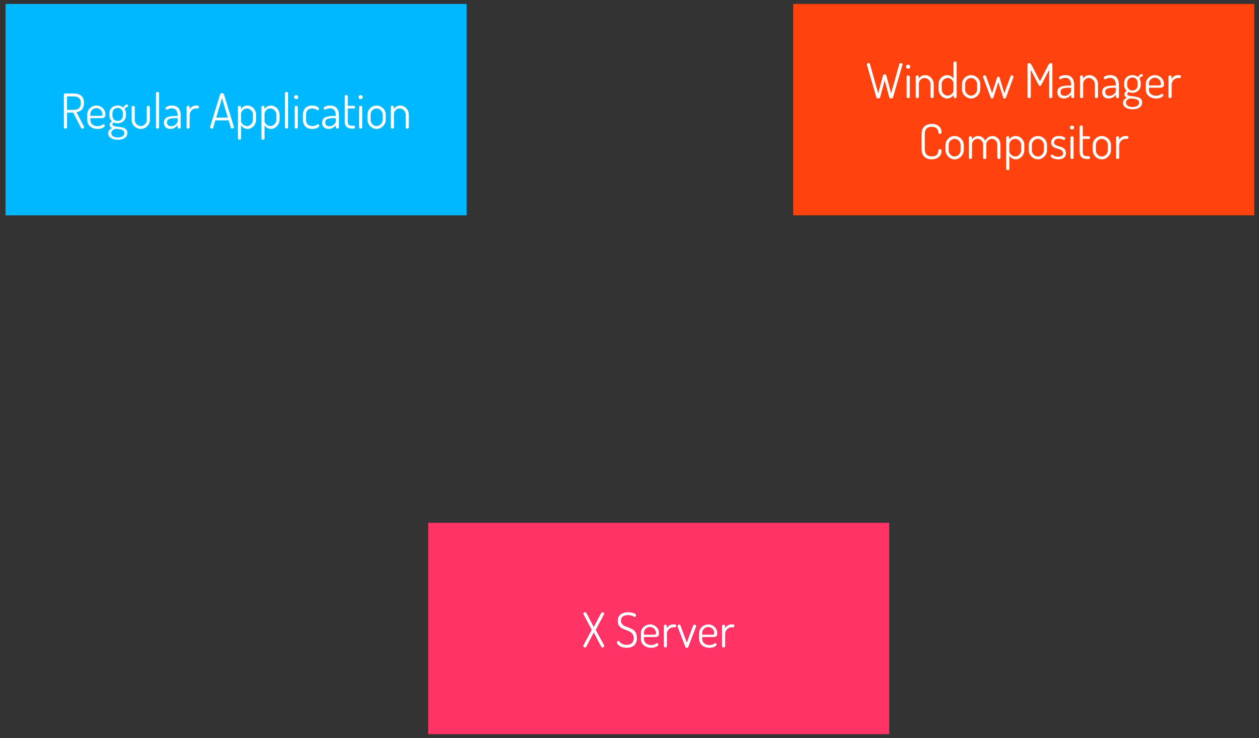
- Wayland is ...
  - Free of legacy design issues X11 has to maintain
  - Smaller codebase than X11
  - Easier to get a “perfect UI” in than X11
  - Easier to support hardware display features than X11
  - More secure than X11

  - Less mature and tested than X11

Connect/Display

Wayland vs. X11

Display (or modification) of windows



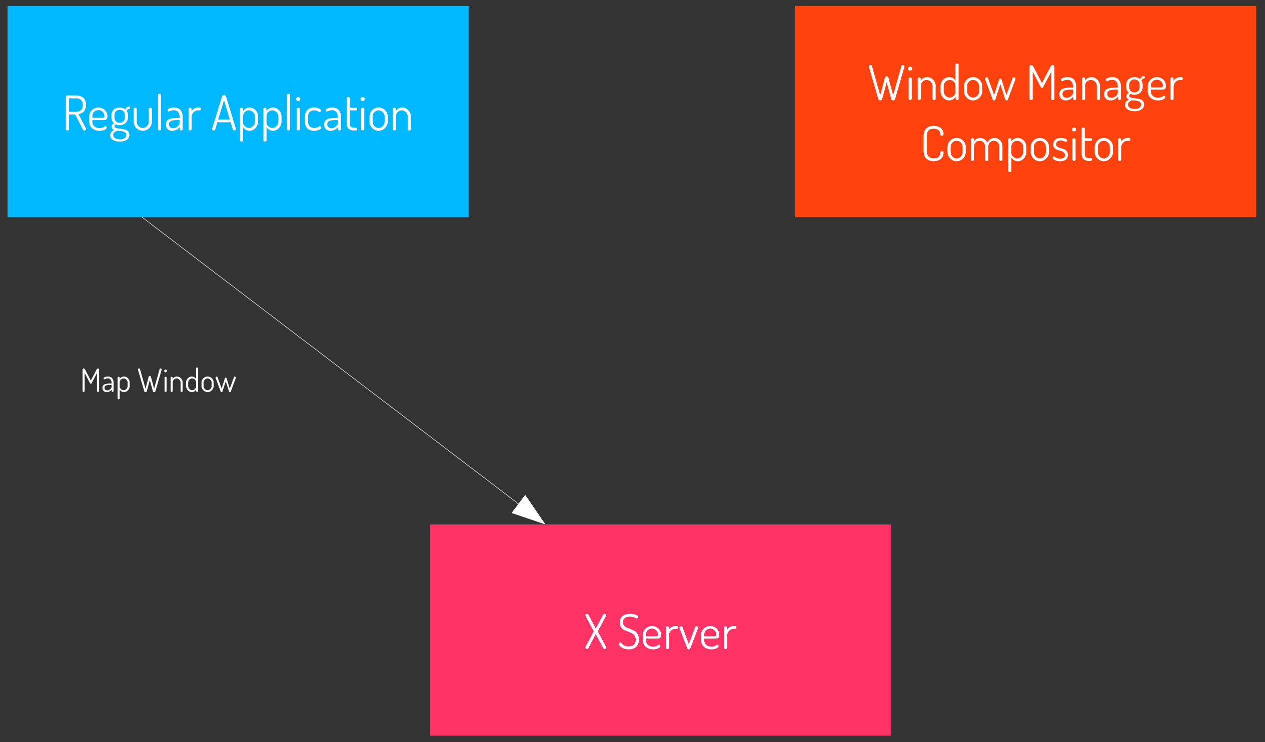
Regular Application

Window Manager
Compositor

X Server

Wayland vs. X11

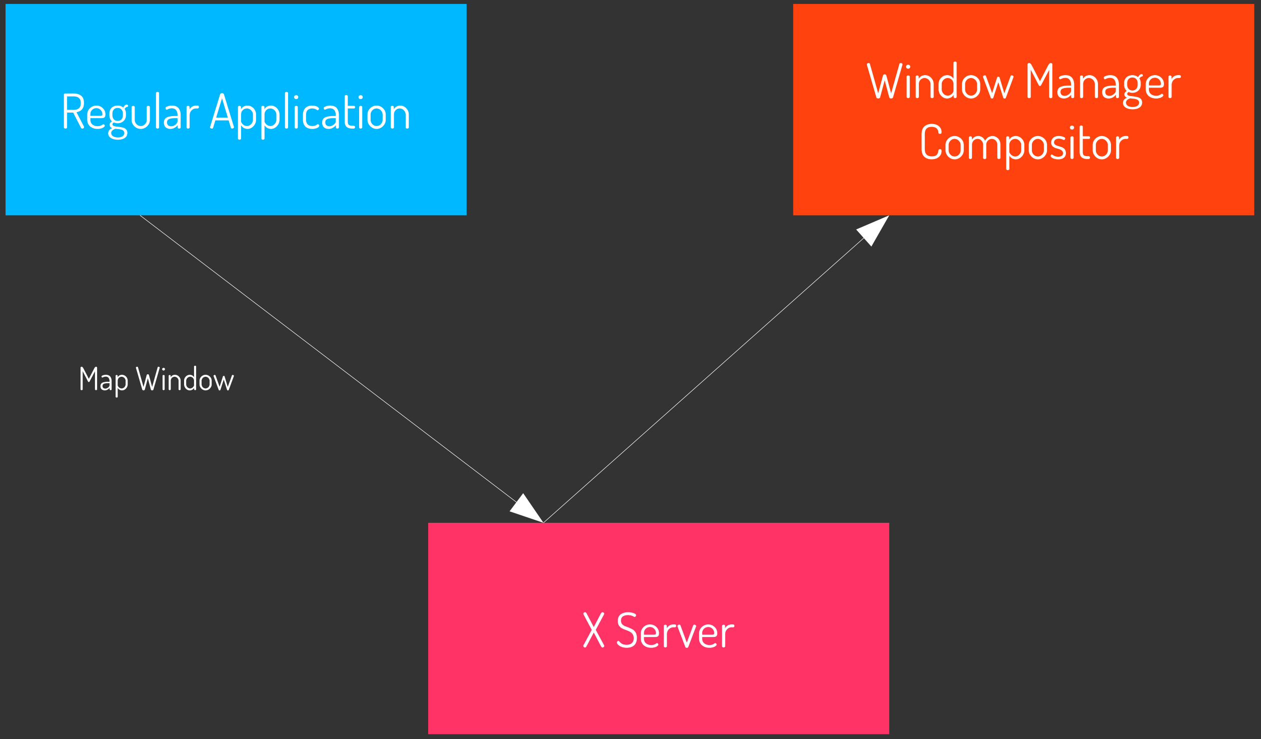
Display (or modification) of windows



BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

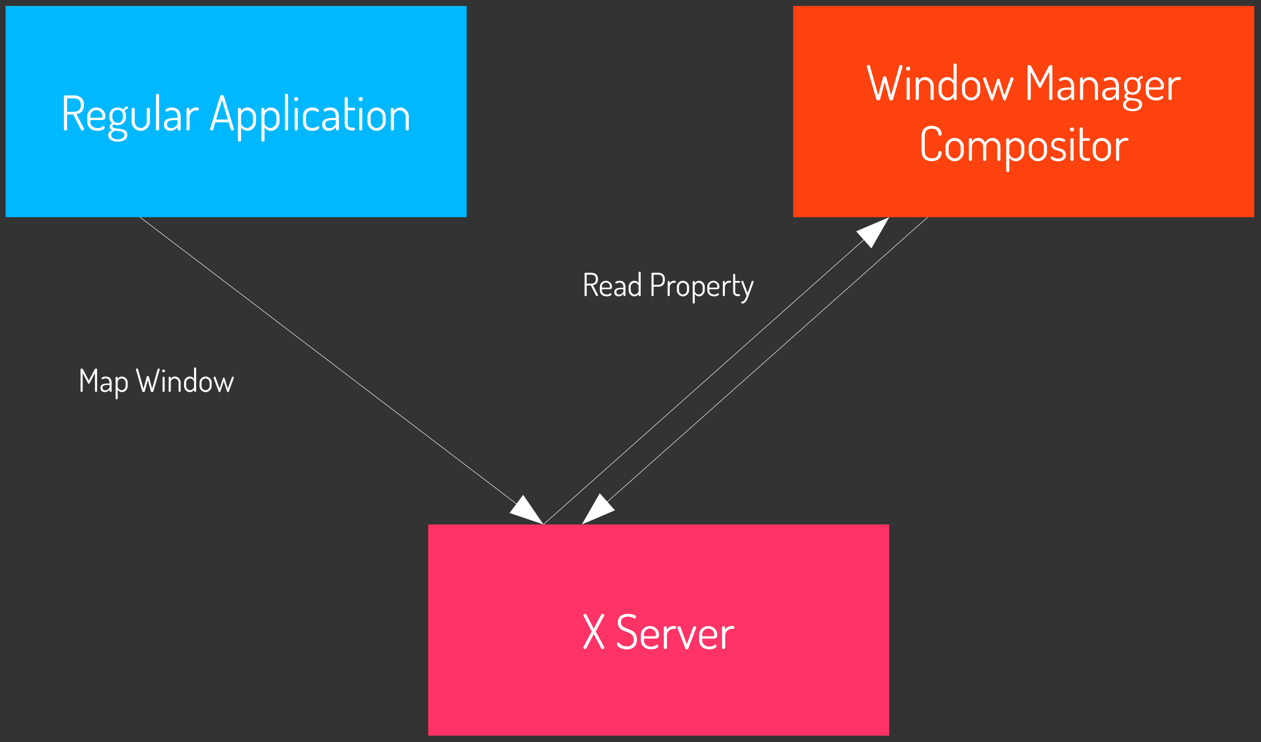
Display (or modification) of windows



BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

Display (or modification) of windows

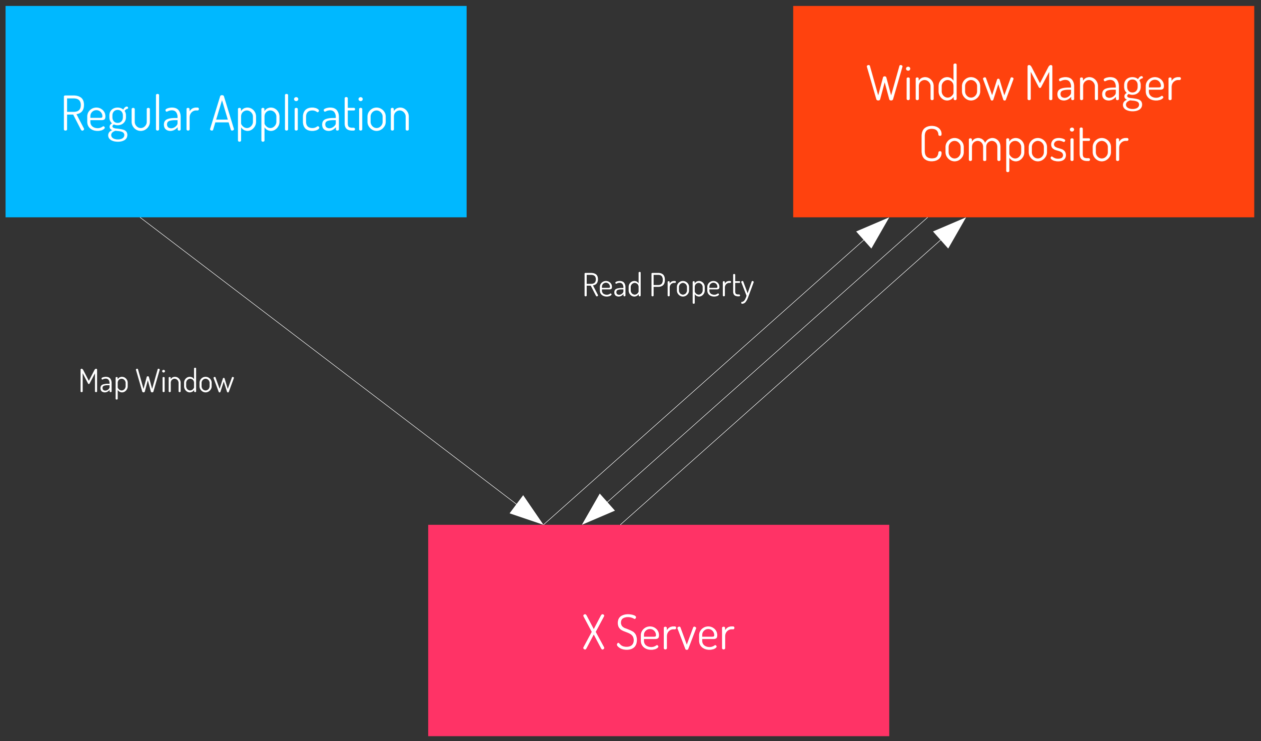


WINDOW ∴ BUFFER
(PIXMAP) SIZES
CONTROLLED HERE

BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

Display (or modification) of windows

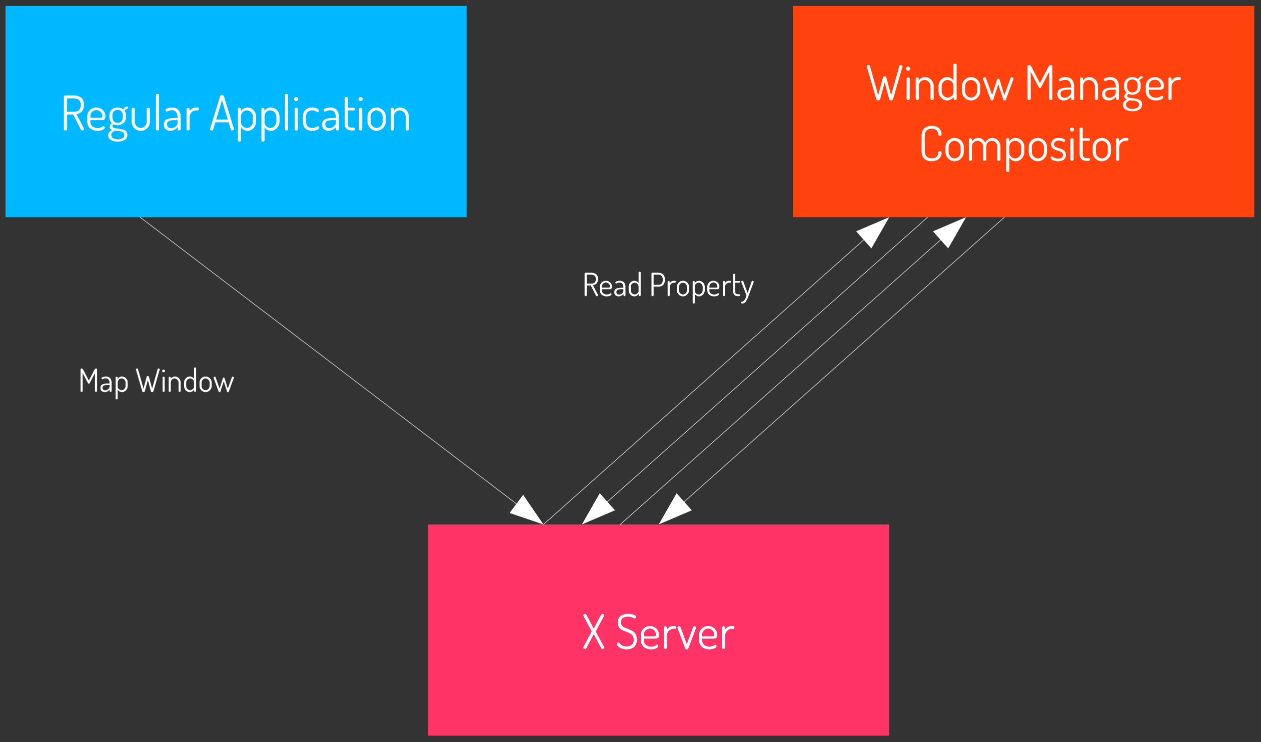


WINDOW ∴ BUFFER
(PIXMAP) SIZES
CONTROLLED HERE

BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

Display (or modification) of windows

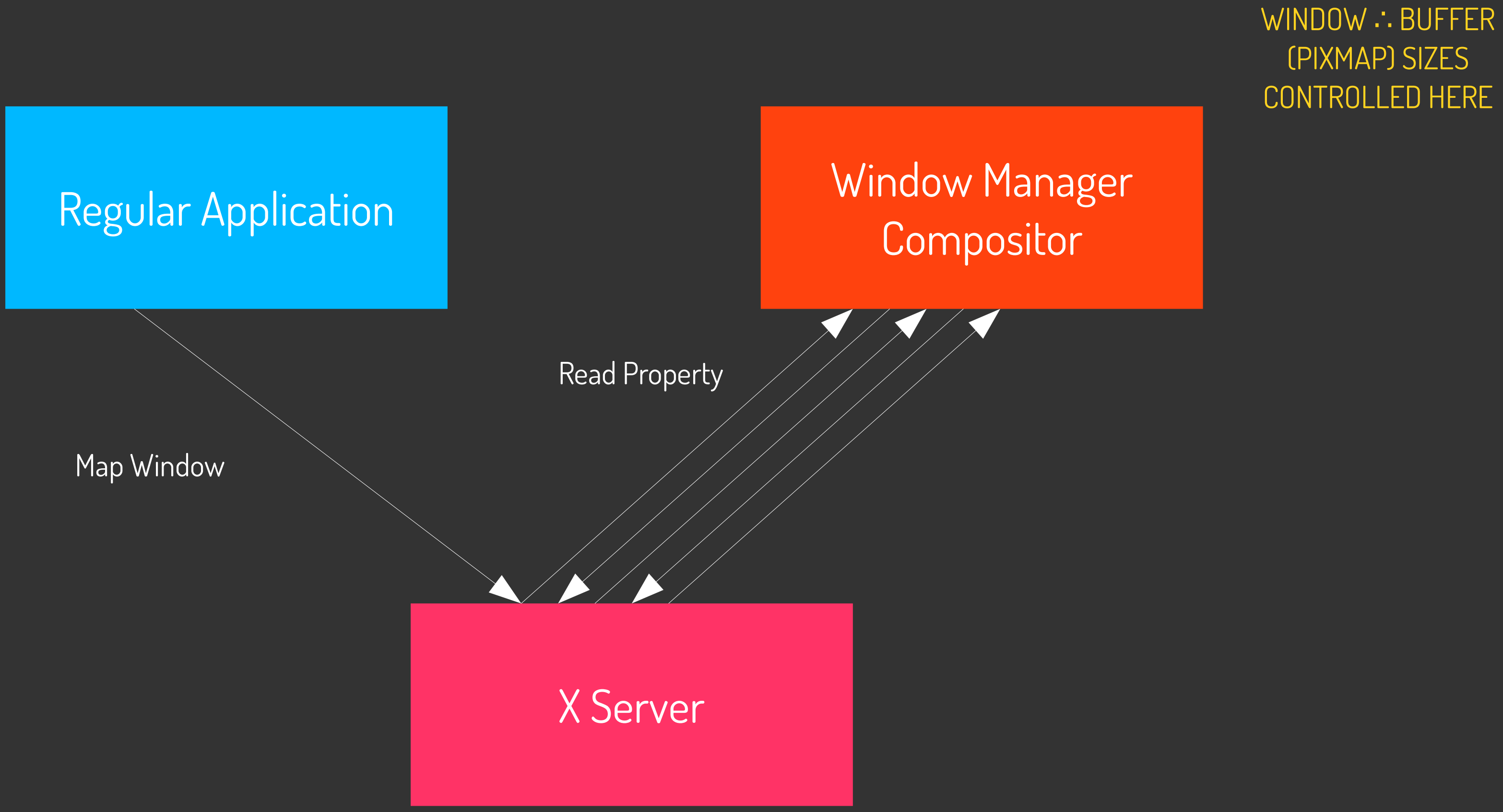


WINDOW ∴ BUFFER
(PIXMAP) SIZES
CONTROLLED HERE

BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

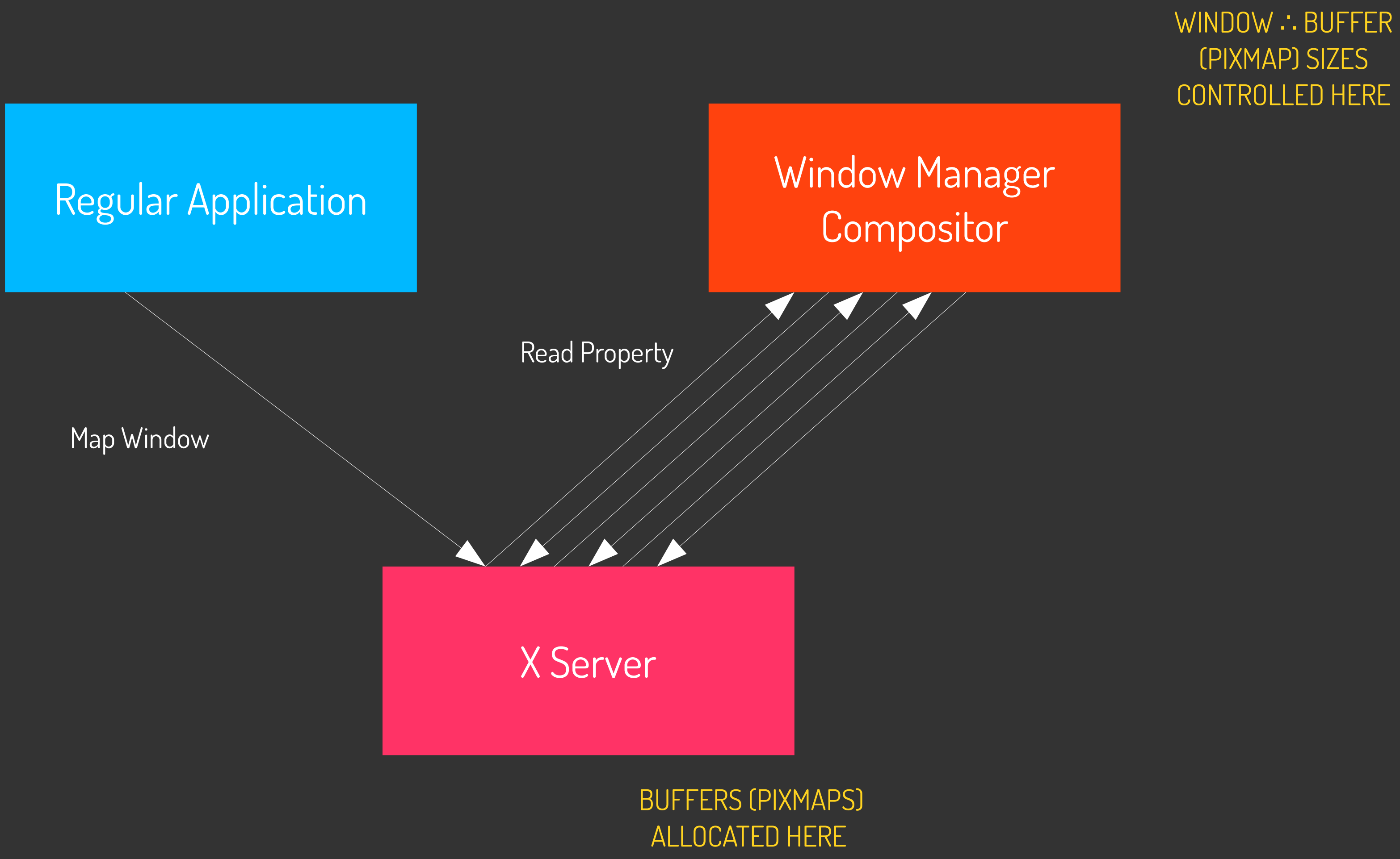
Display (or modification) of windows



BUFFERS (PIXMAPS)
ALLOCATED HERE

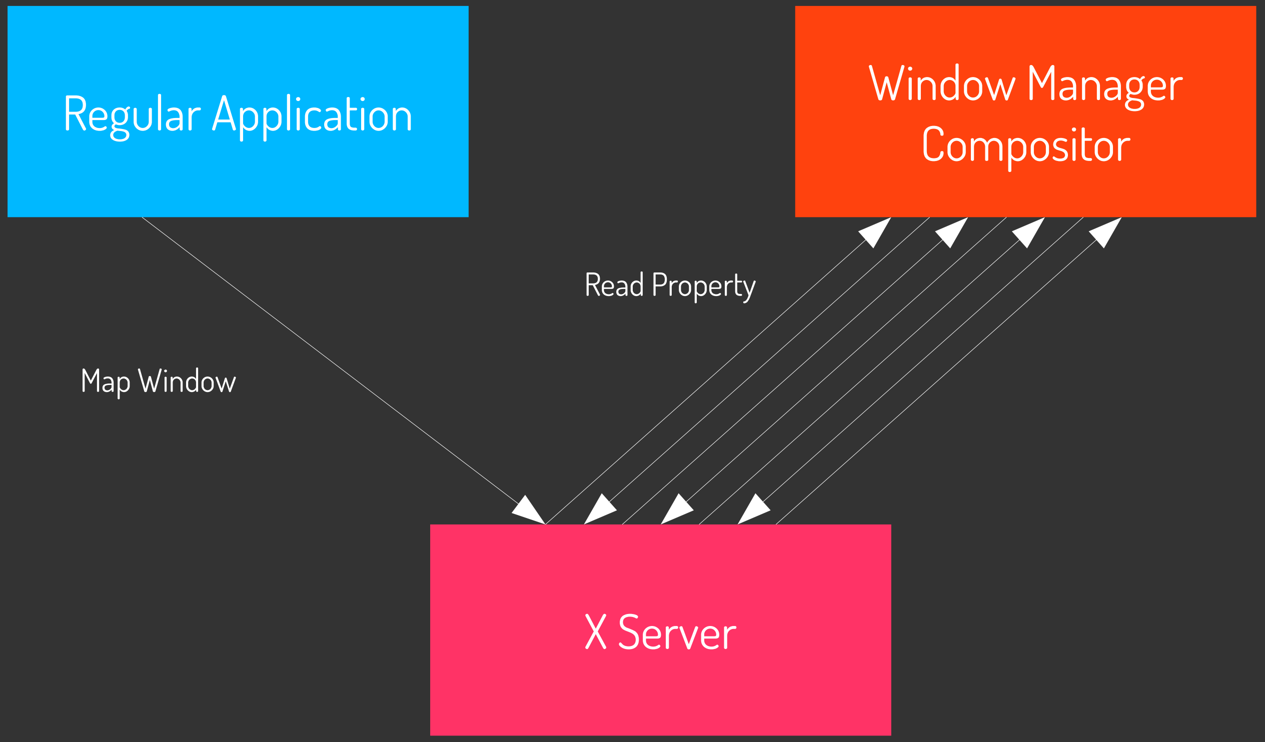
Wayland vs. X11

Display (or modification) of windows



Wayland vs. X11

Display (or modification) of windows

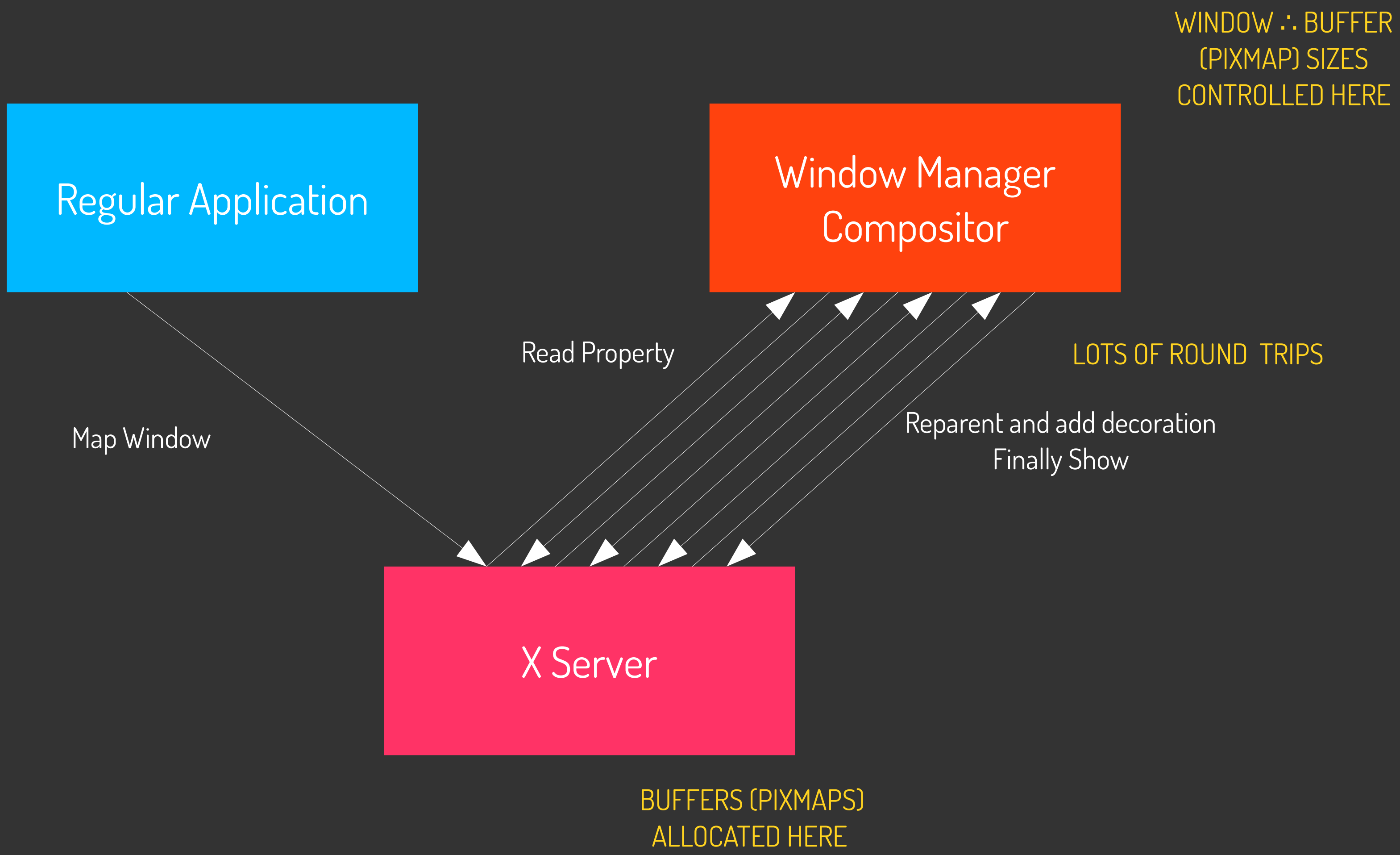


WINDOW ∴ BUFFER
(PIXMAP) SIZES
CONTROLLED HERE

BUFFERS (PIXMAPS)
ALLOCATED HERE

Wayland vs. X11

Display (or modification) of windows



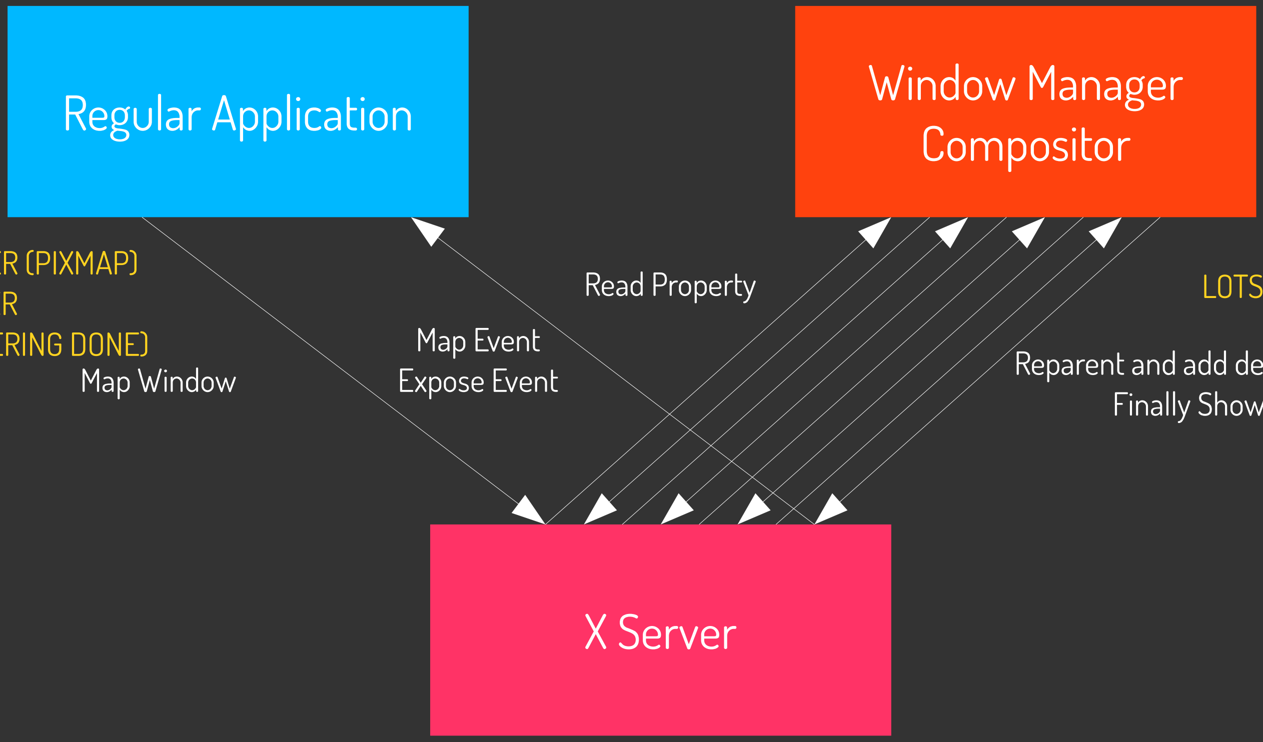
WINDOW ∴ BUFFER (PIXMAP) SIZES CONTROLLED HERE

LOTS OF ROUND TRIPS

BUFFERS (PIXMAPS) ALLOCATED HERE

Wayland vs. X11

Display (or modification) of windows



WINDOW ∴ BUFFER
(PIXMAP) SIZES
CONTROLLED HERE

Regular Application

Window Manager
Compositor

KNOW ABOUT BUFFER (PIXMAP)
SIZES LATER
(OFTEN AFTER RENDERING DONE)

Map Window

Map Event
Expose Event

Read Property

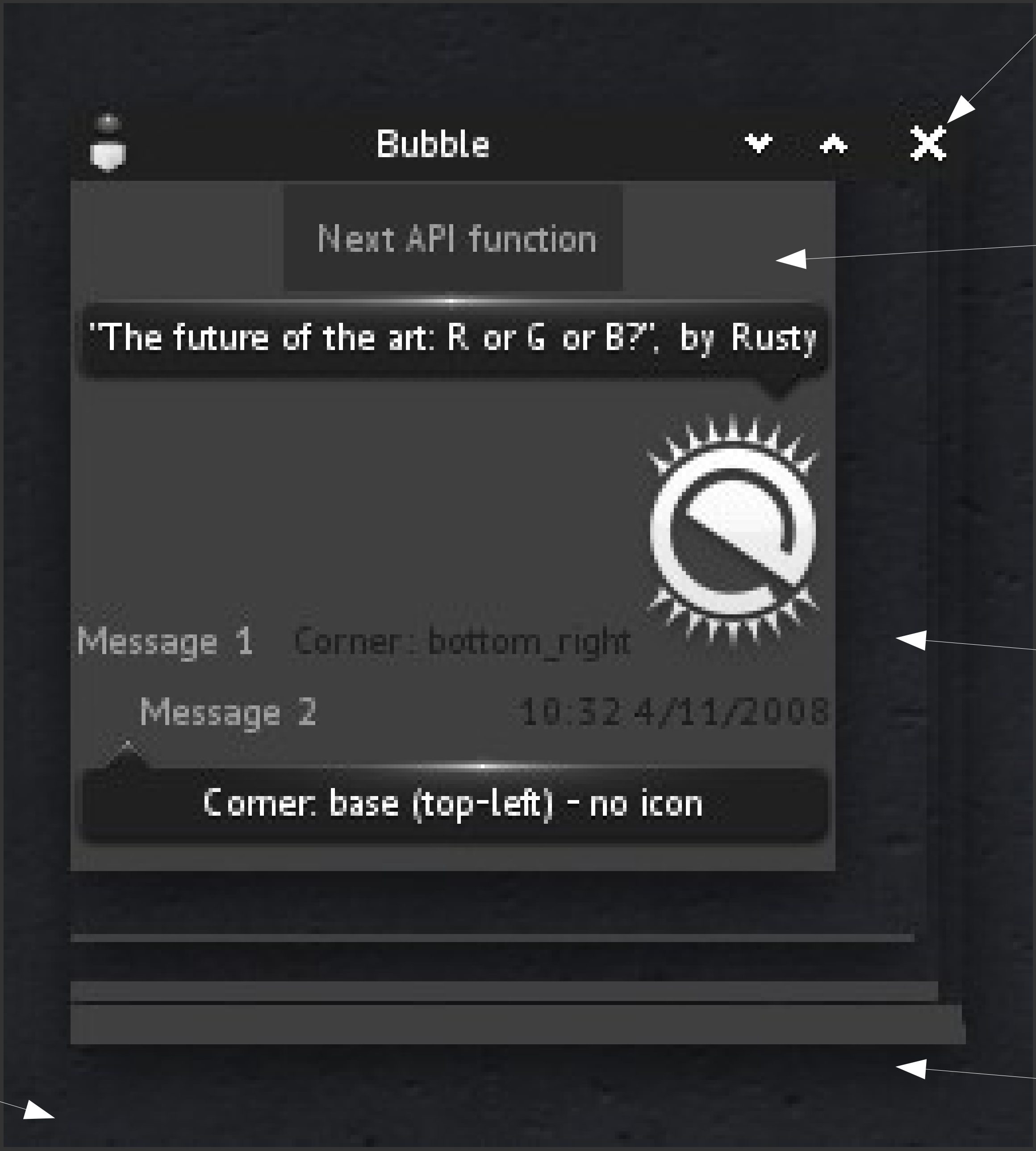
LOTS OF ROUND TRIPS

Reparent and add decoration
Finally Show

X Server

BUFFERS (PIXMAPS)
ALLOCATED HERE

Often results in this...


Bubble
Next API function
"The future of the art: R or G or B?", by Rusty
Message 1
Corner : bottom_right
Message 2
10:32 4/11/2008
Corner: base (top-left) - no icon

Decoration

Client application content

Undrawn parts of the application buffer

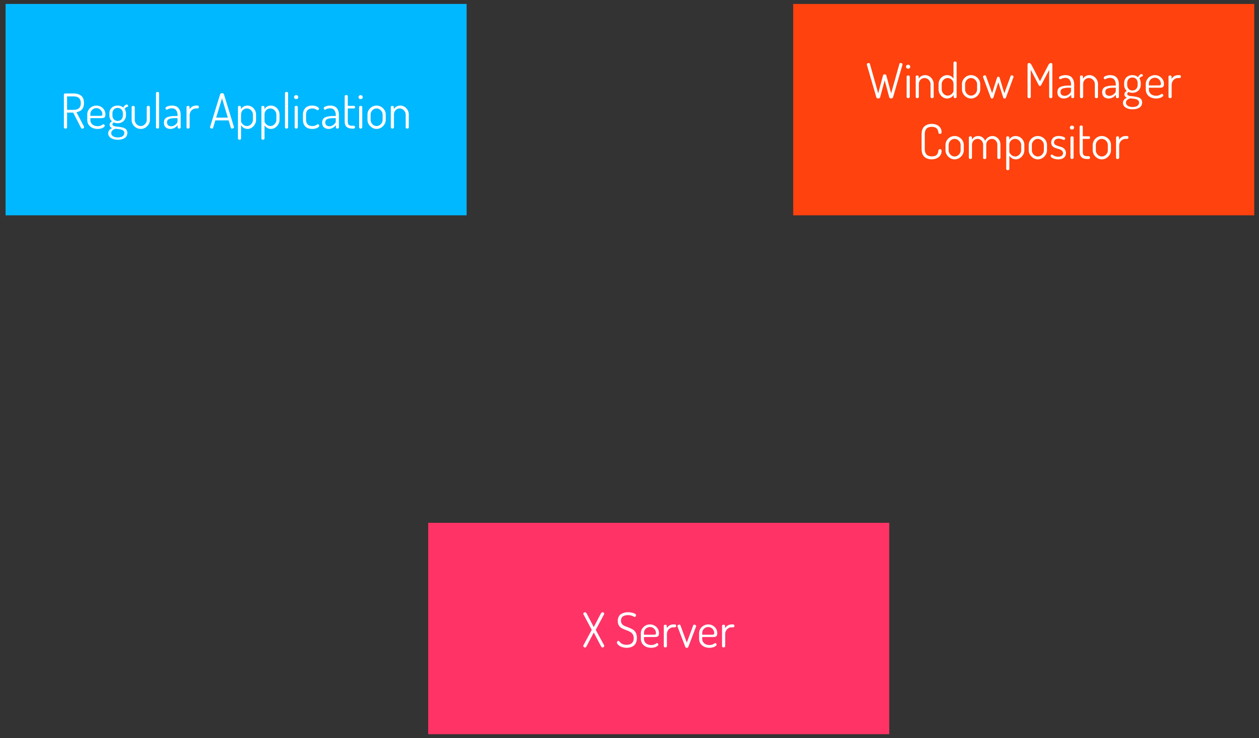
(resized after rendering began)

Background handled by WM/Compositor

Shadows drawn by WM/Compositor

Wayland vs. X11

Rendering updates



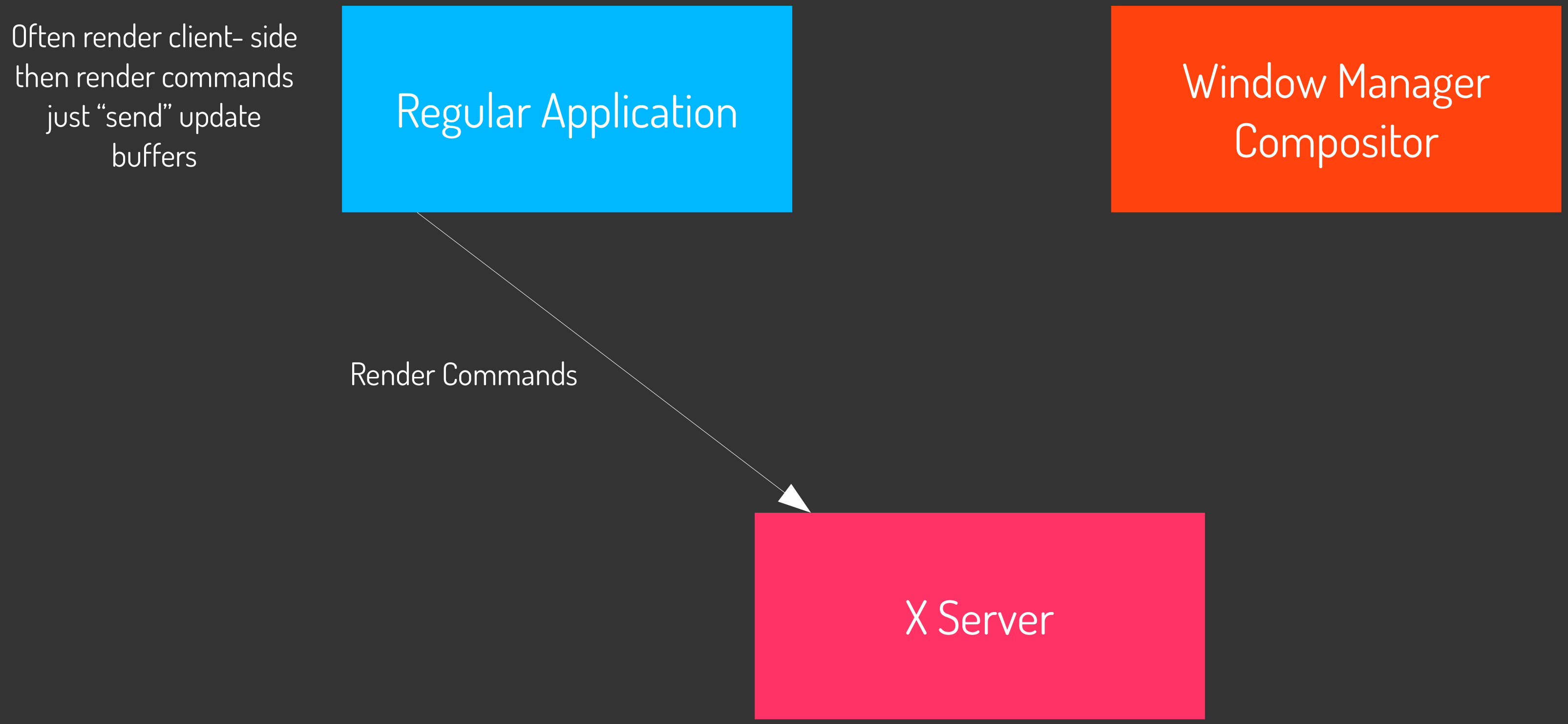
Regular Application

Window Manager
Compositor

X Server

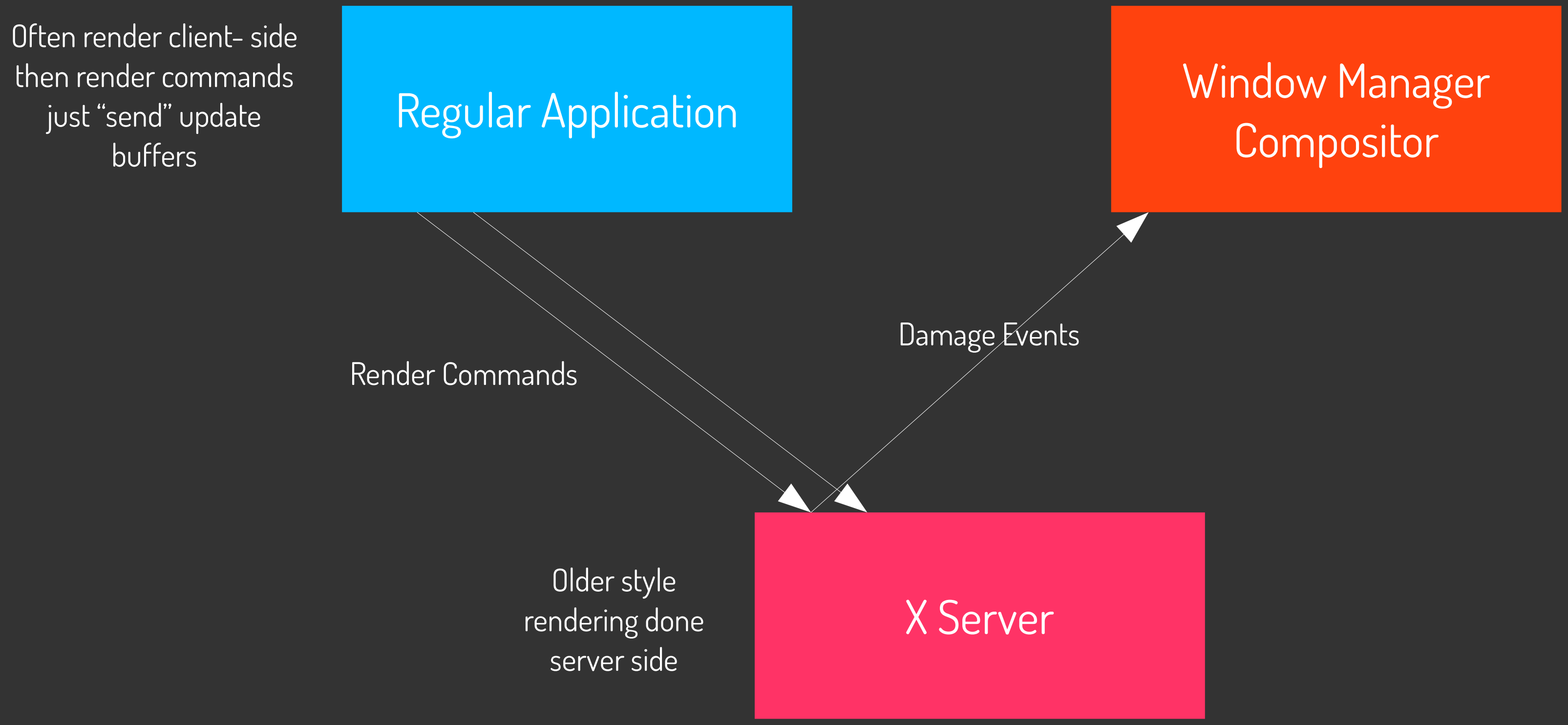
Wayland vs. X11

Rendering updates


Often render client- side then render commands just “send” update buffers
Regular Application
Window Manager Compositor
Render Commands
X Server

Wayland vs. X11

Rendering updates



Often render client- side then render commands just “send” update buffers

Regular Application

Window Manager
Compositor

Render Commands

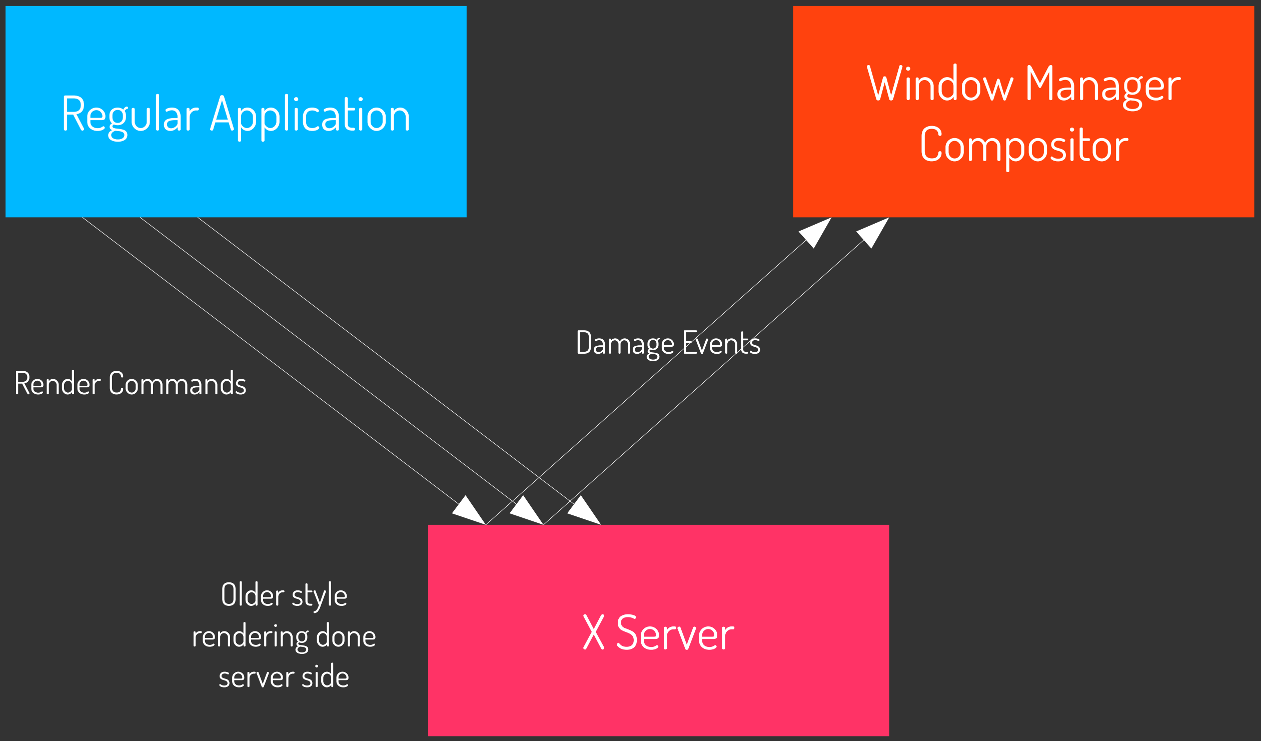
Damage Events

Older style rendering done server side

X Server

Wayland vs. X11

Rendering updates



Often render client- side then render commands just “send” update buffers

Regular Application

Window Manager Compositor

Render Commands

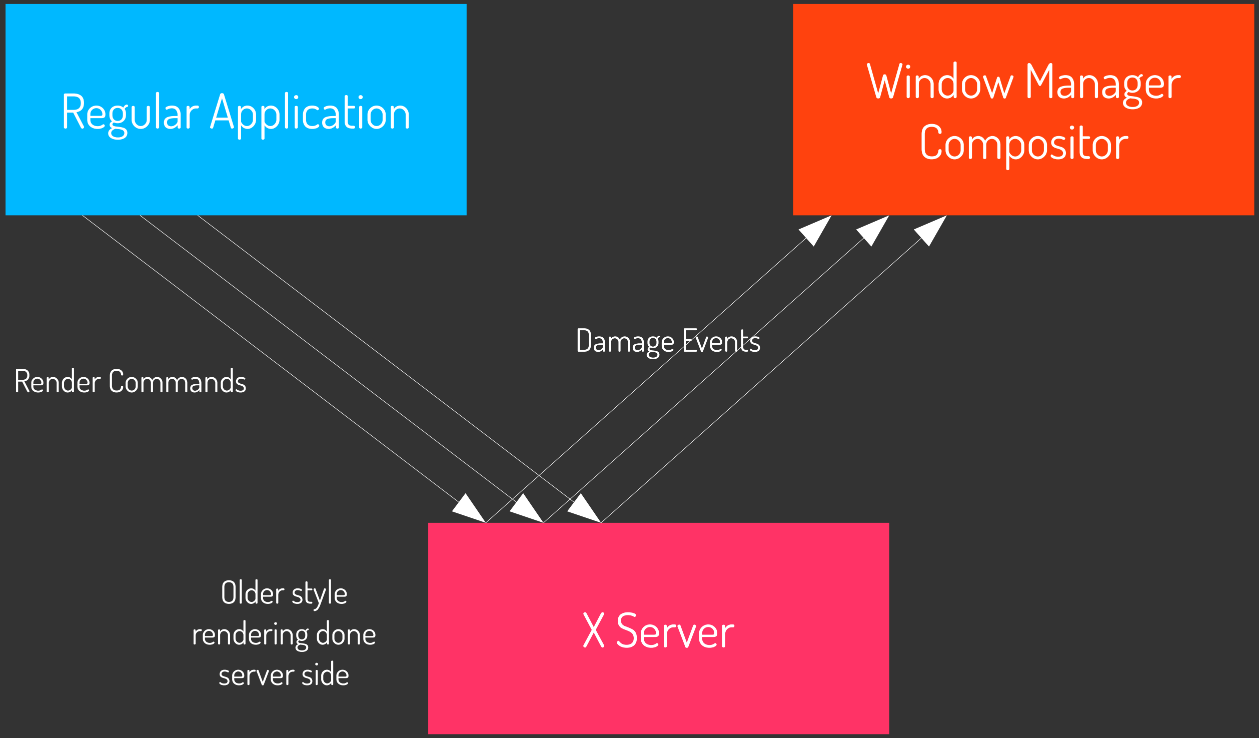
Damage Events

Older style rendering done server side

X Server

Wayland vs. X11

Rendering updates



Often render client- side then render commands just “send” update buffers

Regular Application

Window Manager Compositor

Render Commands

Damage Events

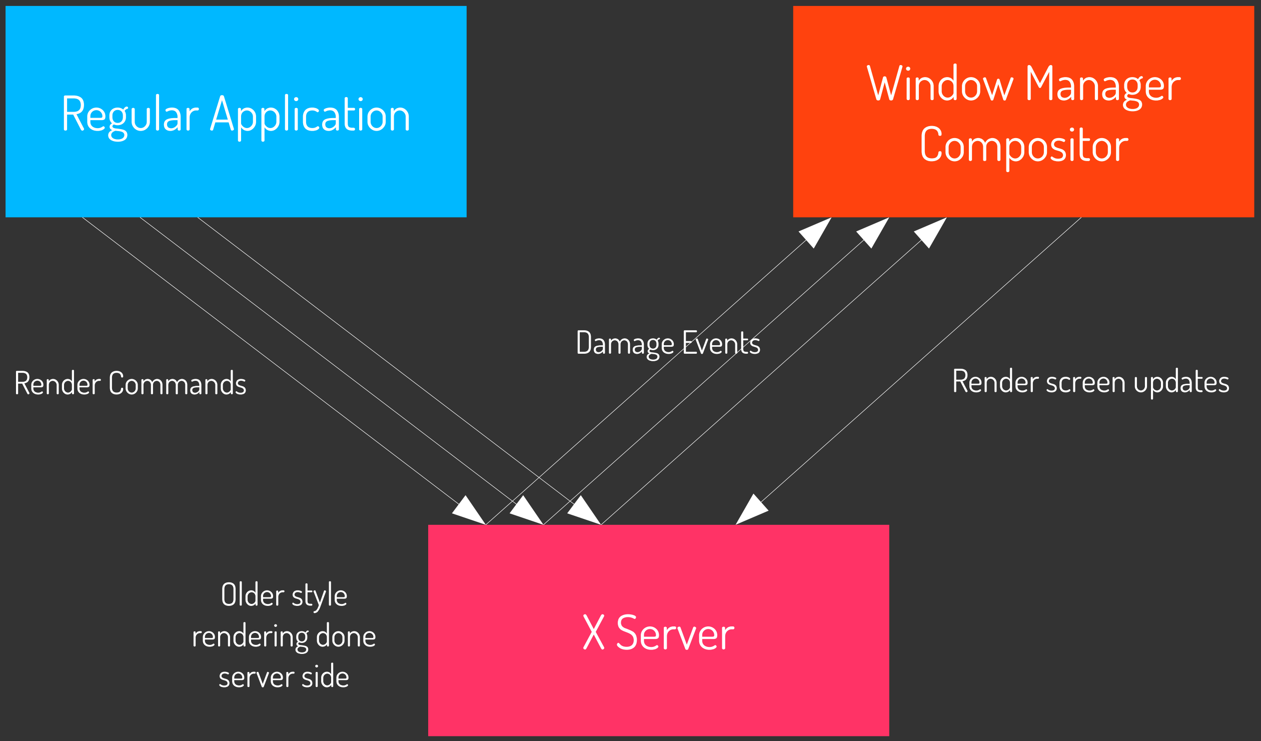
X Server

Older style rendering done server side

Wayland vs. X11

Rendering updates

Often render client- side then render commands just “send” update buffers

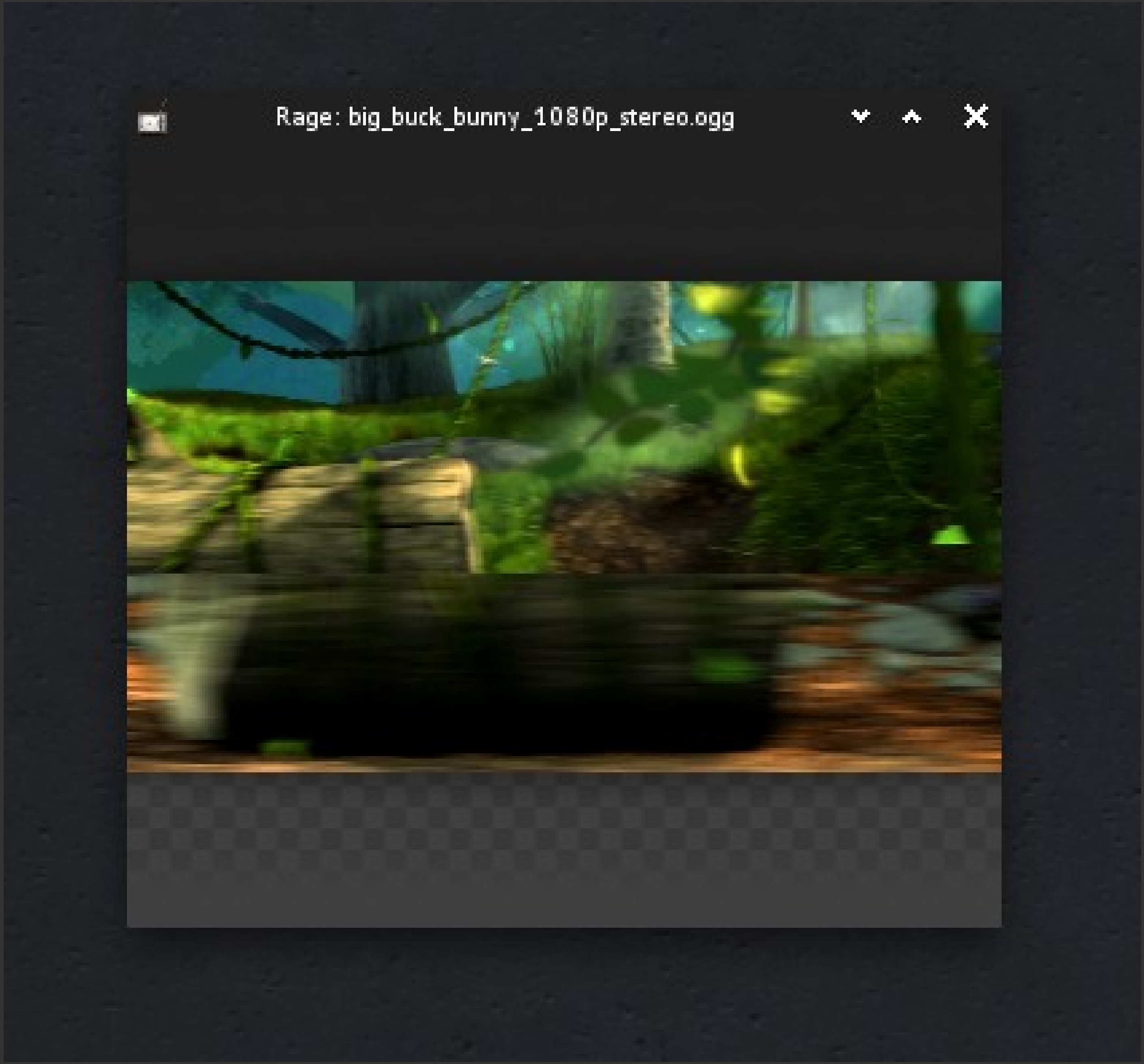


Older style rendering done server side

Problems as a result

- Sometimes compositor renders partial content
  - Responds to first damage event, and misses others
    - Other damages are fixed up next frame

Tearing



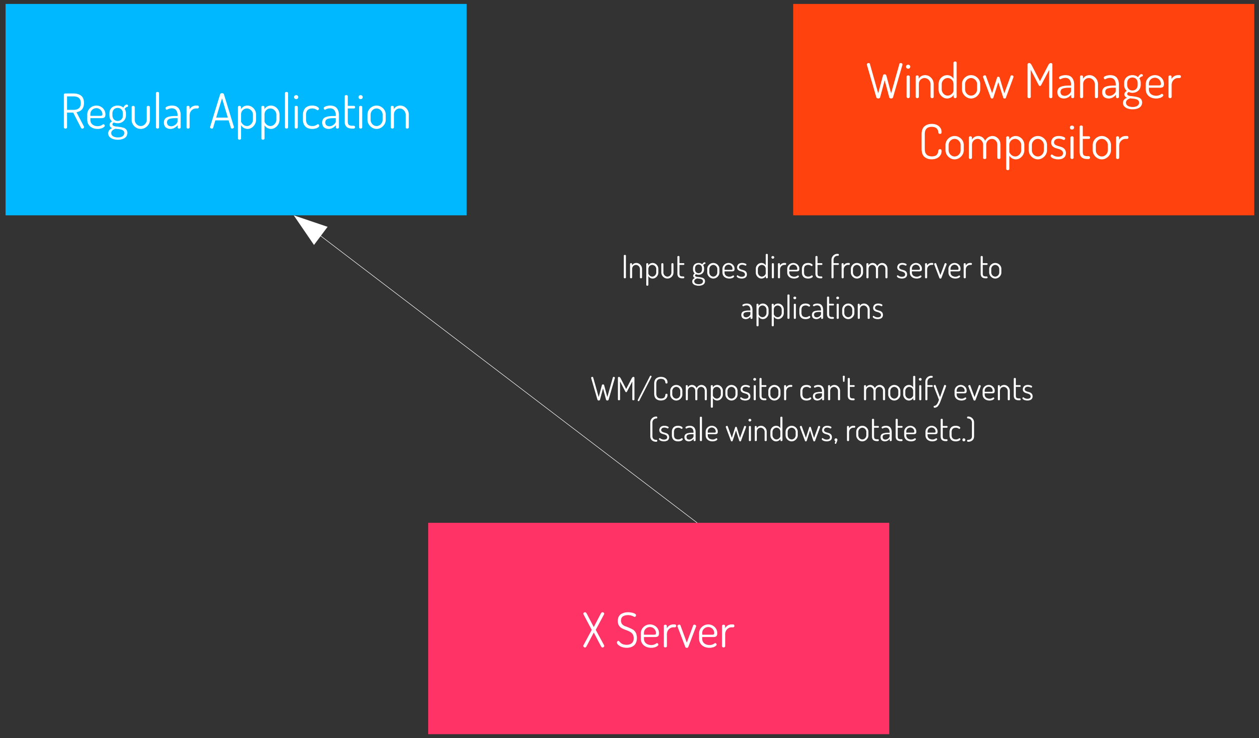
Rage: big_buck_bunny_1080p_stereo.ogg

Problems as a result

- If rendering client-side, most pixels end up being copied to the target
  - Huge amounts of memory bandwidth needed
    - ~500MB/sec for 1080p @ 60HZ needing copying
      - 2GB/sec for UHD ...
    - Even worse if you don't use OpenGL or MIT-SHM extension
    - This can easily drop framerates by 20-50%
- Requires display server to have complete drawing subsystem
  - A legacy decision for X11 before shared libraries existed
    - Allows sharing rendering code via the XServer process
  - Must remain pixel-perfect to retain compatibility

Wayland vs. X11

Input events



Regular Application

Window Manager Compositor

Input goes direct from server to applications

WM/Compositor can't modify events (scale windows, rotate etc.)

X Server

Problems here...

- WM/Compositor can't rotate, zoom or transform content
  - Input event co-ordinates can only match “original” window geometry
- WM can set what window has focus
  - Clients can too
    - Leads to possible fighting between clients and WM
- Clients can listen to all input
  - Huge security issue – e.g. any app can be a keylogger
- Clients can steal input locking everyone out
  - This can affect even screensavers and screenlocks by preventing screenlocks
    - The infamous “leave a menu open to prevent a screen locking” bug

Why does Tizen REALLY want Wayland

- Security and client isolation
  - Tizen needs to sandbox apps properly
  - Apps may be downloaded and not audited or able to be trusted
    - May be closed source
    - Could contain backdoors or trojans
- If 3rd party apps can't be trusted, they need to be isolated & secure
  - Cannot get access to data unless approved by the user
    - e.g. Contacts, Photos, Microphone, Camera etc. etc.
  - Cannot manipulate other apps
  - Cannot listen into input except their own

Why does Tizen REALLY want Wayland

- Far better zero-copy rendering support
  - Tizen targets embedded devices which often have very little processing power
    - Need to limit copies

Why does Tizen REALLY want Wayland

- Ensure you don't see partial updates
  - Tizen is meant to have “commercial quality display”
    - Partial updates and tearing are not acceptable
  - Major competitors have tear-free display
    - Can't compete without at least matching

Why does Tizen REALLY want Wayland

- Massively reduce round-trips
  - Performance matters much more on low-end embedded devices
  - Users expect almost instant responsiveness
    - Wayland can improve startup time of applications on target devices by several 100ms vs X11
      - Tests have shown ~400ms improvements
  - Memory usage reduced
    - Apps can save between ~1 to ~11MB
    - Compositor saves ~ 48MB
  - All of this while keeping the same (approximately) functionality, look and feel.

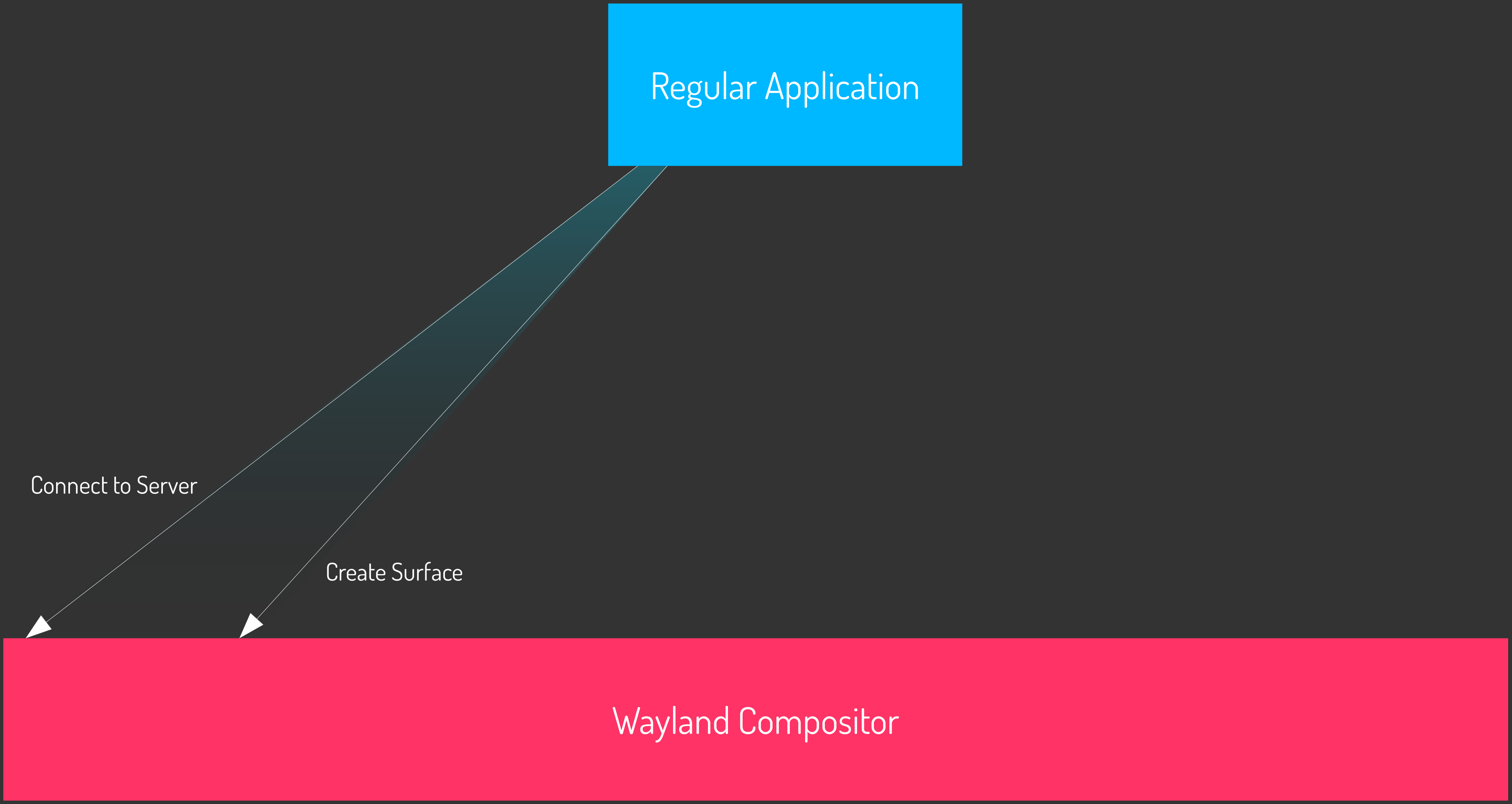
Why does Tizen REALLY want Wayland

- Far better designed support for hardware layers
  - Embedded hardware often supports several RGBA and YUV overlays
    - This allows zero-copy buffer assignment not just for fullscreen apps but for multiple windows
    - Regular mid-range hardware often supports 5 layers or more
  - Wayland can make better use of this via Surfaces and Sub-Surfaces
  - Allows compositor to effectively “turn off” and…
    - Wake up to deliver input events to client apps
    - Wake up on new buffer display
      - Assign application output buffer handles/pointers to the correct display output layer

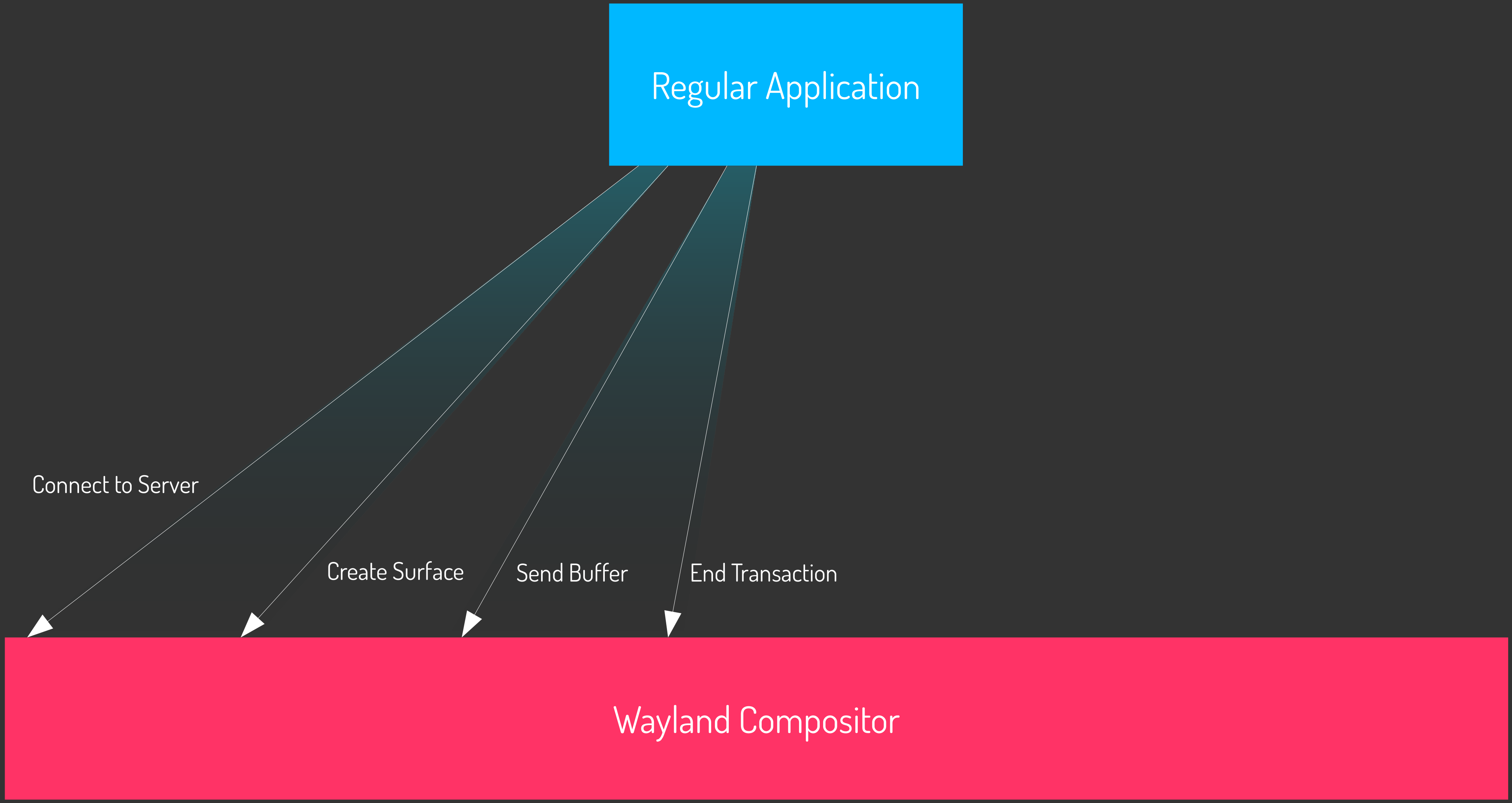
Why does Tizen REALLY want Wayland

- Rotation
  - We need good, clean rotation support for Tizen and Wayland delivers
    - Phones, Tablets and Wearables need to rotate
    - Even TVs need rotation (to become vertical banner displays)
    - We currently do it in X11 with lots of tricks and client-side support
      - Wayland can clean this up.
  - Opens up possibilities of things like shared “touch tables”
    - Multiple people around a single table
    - Different pieces of content (windows) at differing rotations per person or content

What Wayland Does



What Wayland Does



Regular Application

Connect to Server

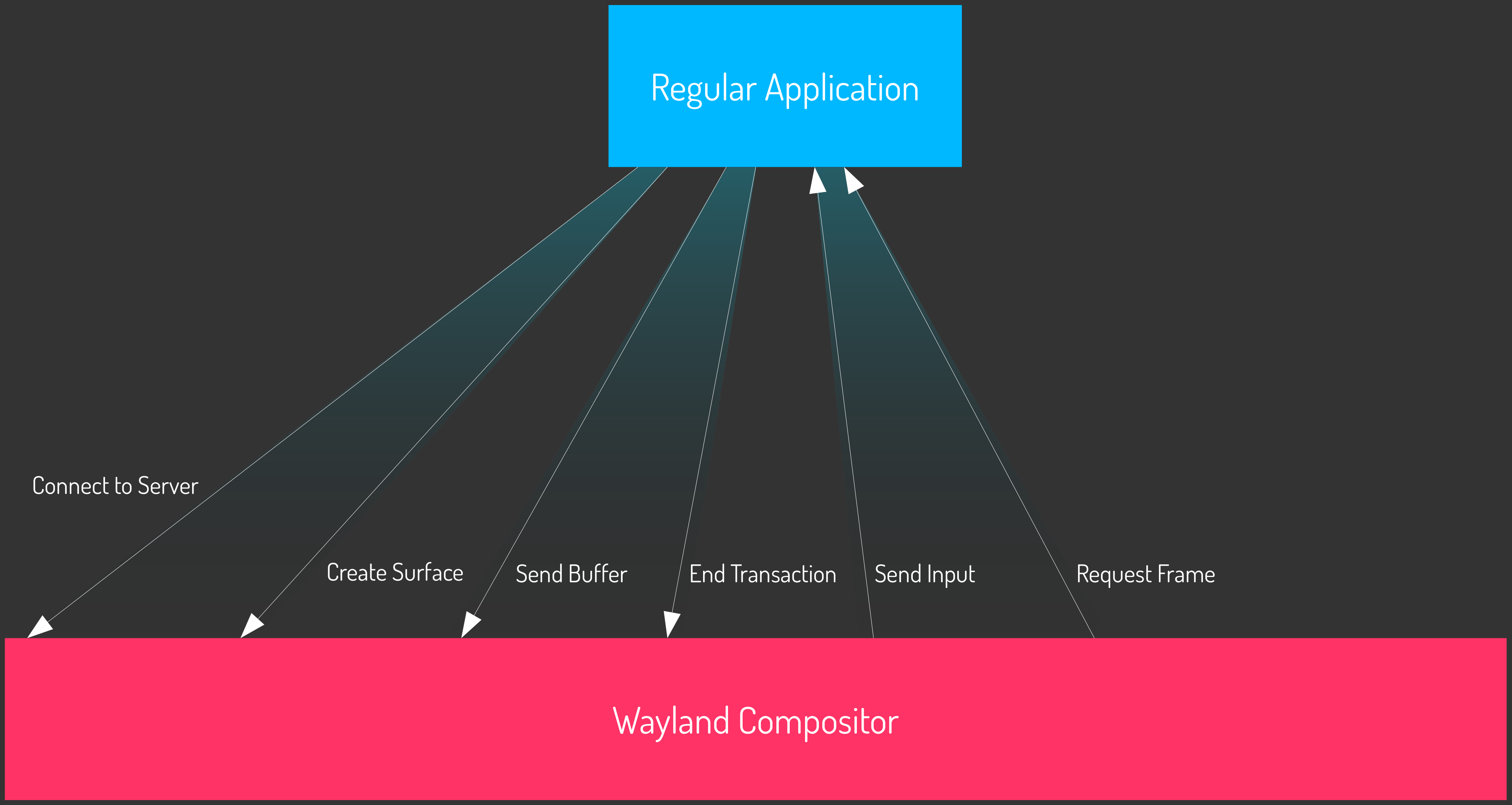
Create Surface

Send Buffer

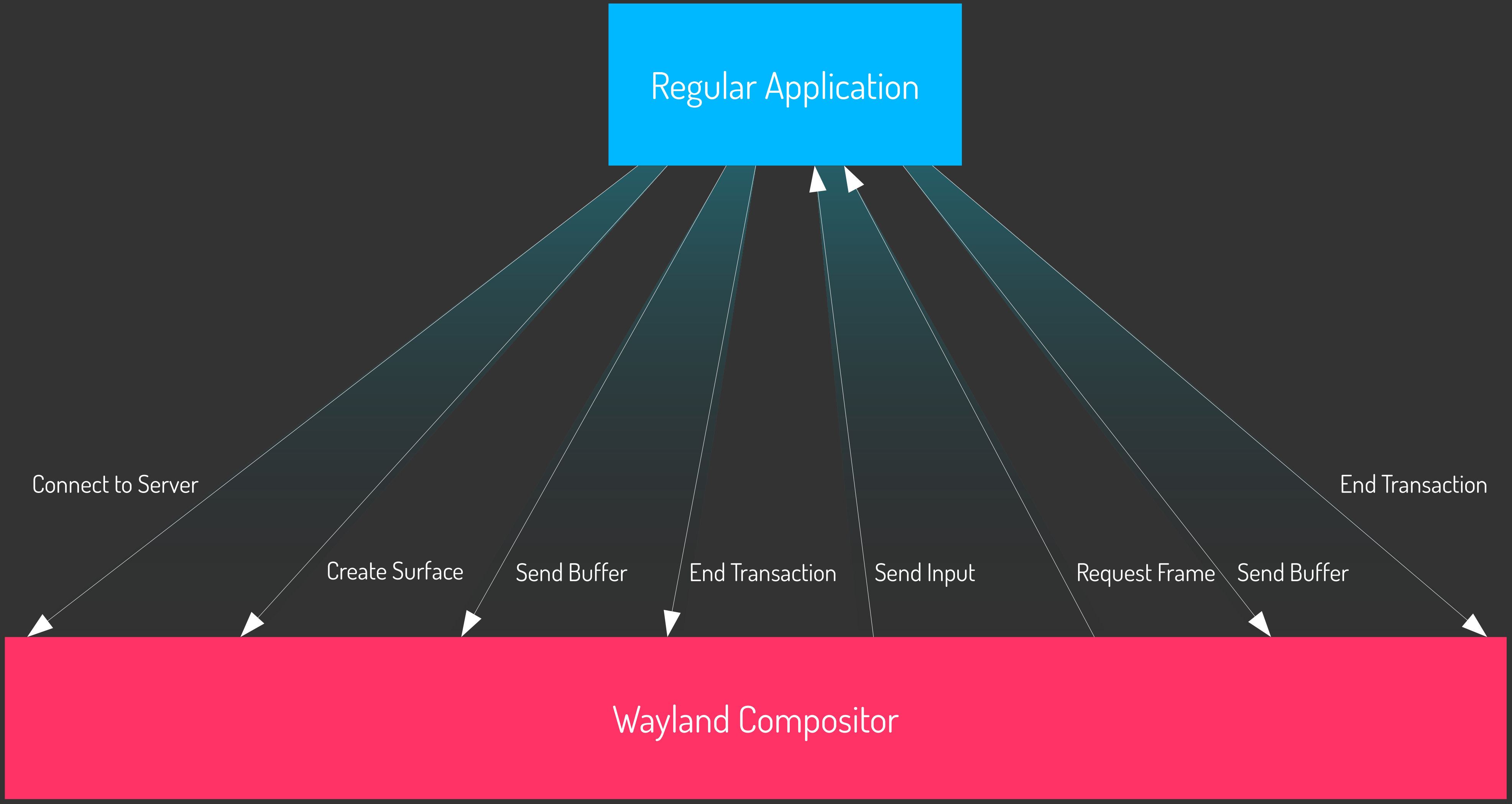
End Transaction

Wayland Compositor

What Wayland Does



What Wayland Does

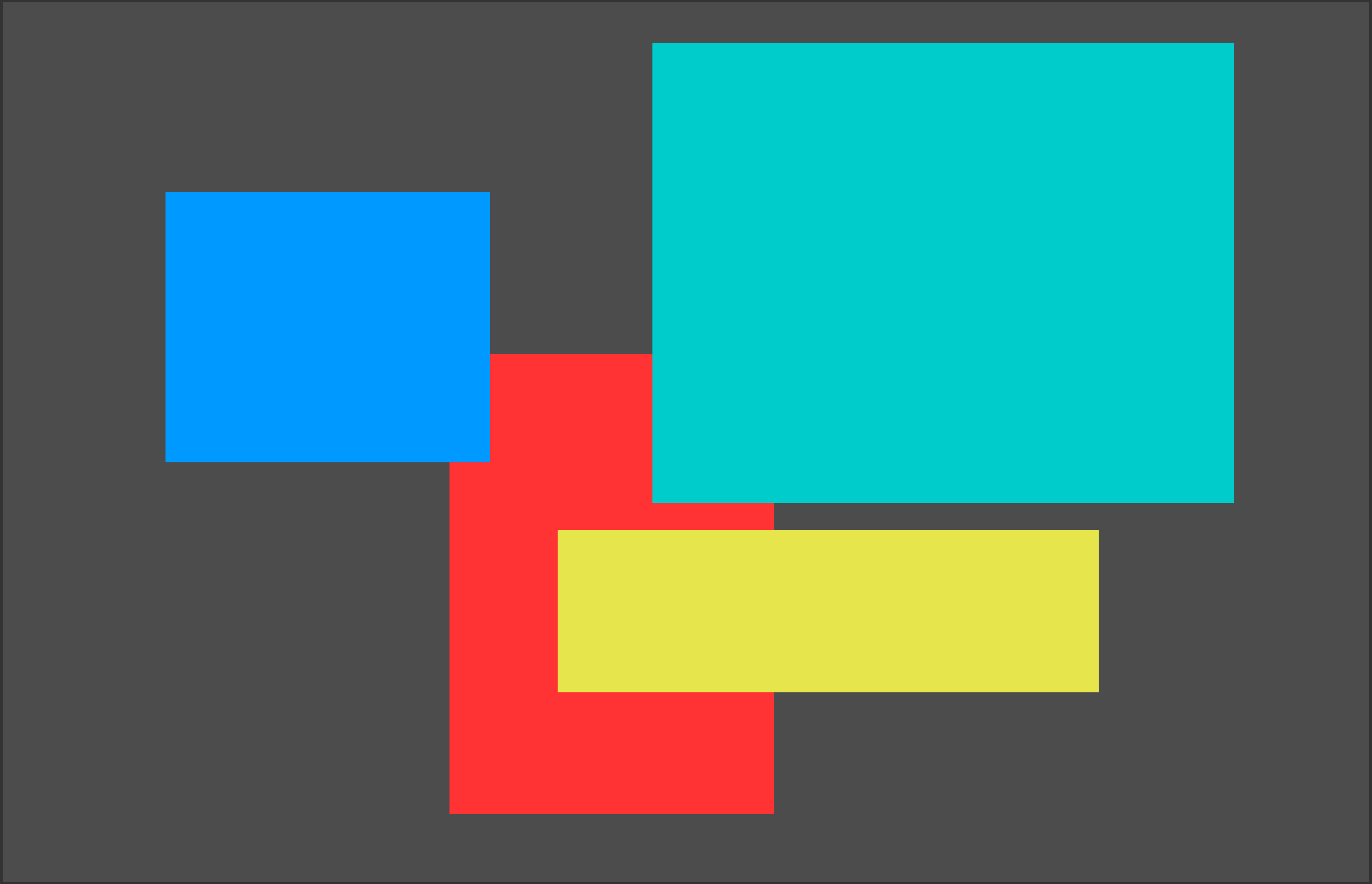


Rendering

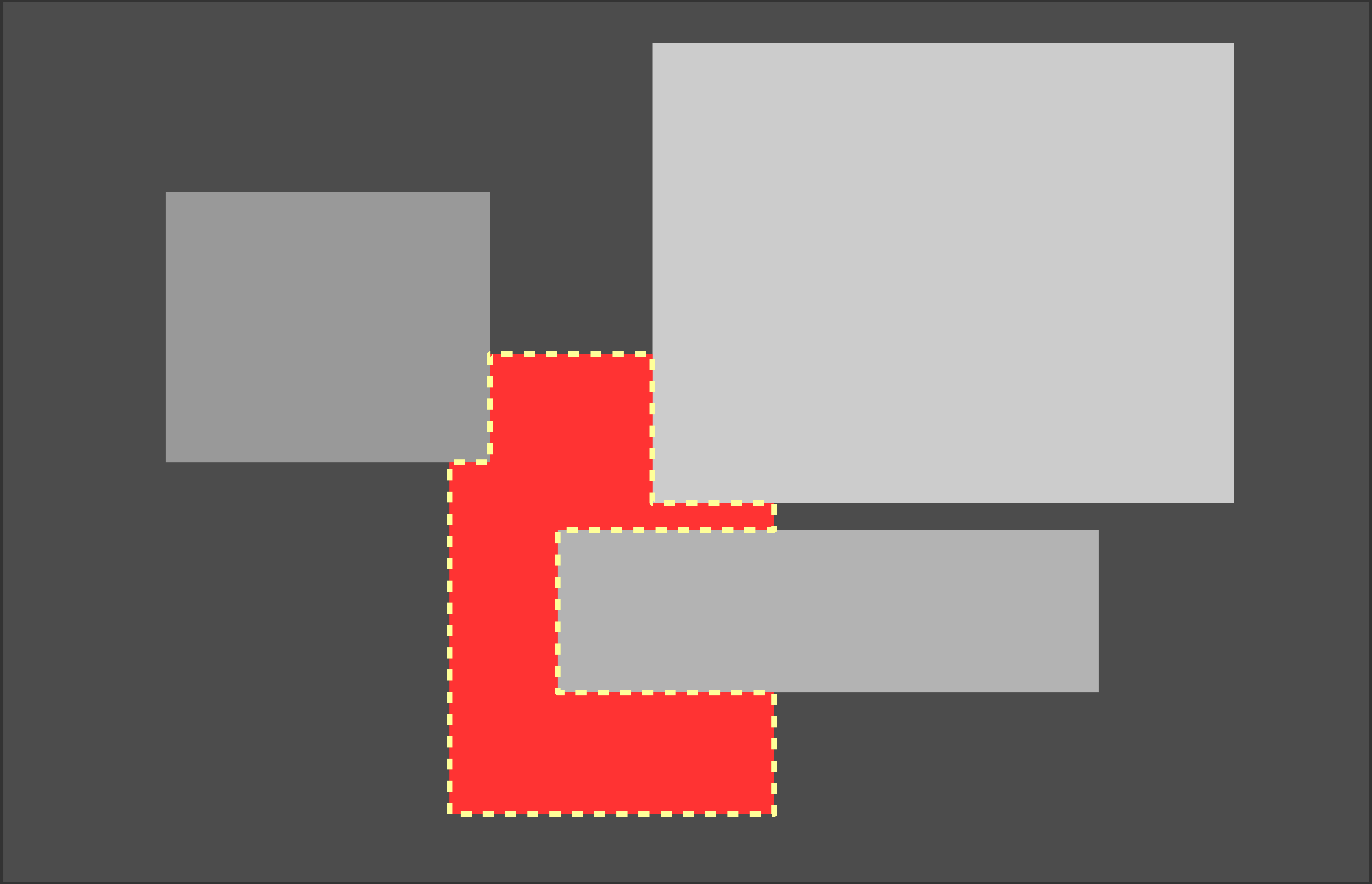
X11 Rendering

- There is only a single framebuffer
  - There is offscreen data like pixmaps – can't be seen (just storage)
- Xserver does the actual rendering to framebuffer or pixmaps
  - Clients cannot directly render to these locations
    - There are exceptions and hacks – another discussion
  - At most clients can:
    - Render to a local memory segment and upload
    - Render with GPU to OpenGL backbuffer then “swap” to a window to display
- Xserver will “clip” rendering only to the correct output regions
  - Invisible parts of windows can avoid beiing drawn entirely
  - It is possible to bypass this – it is very anti-social

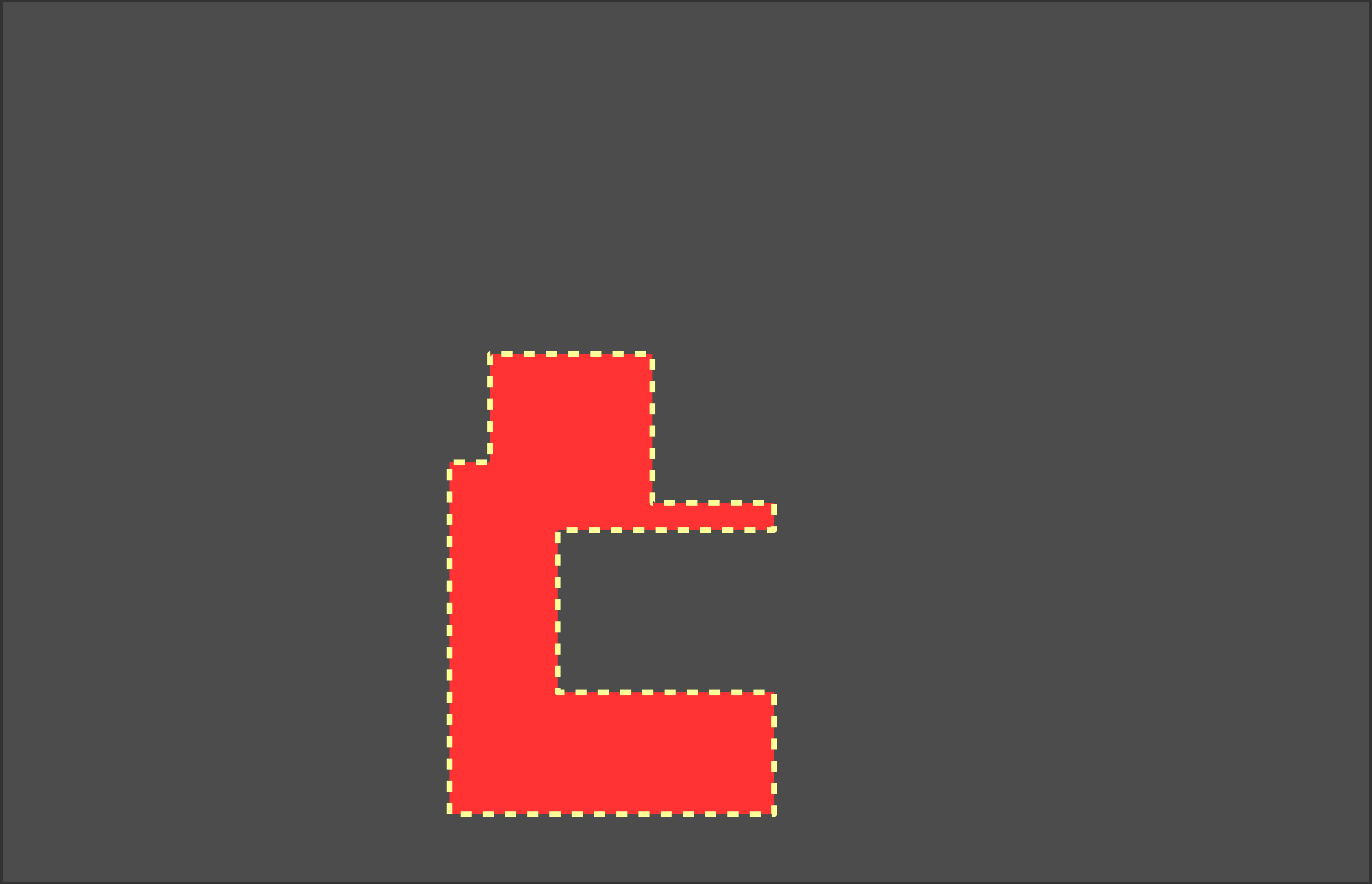
X11 Rendering



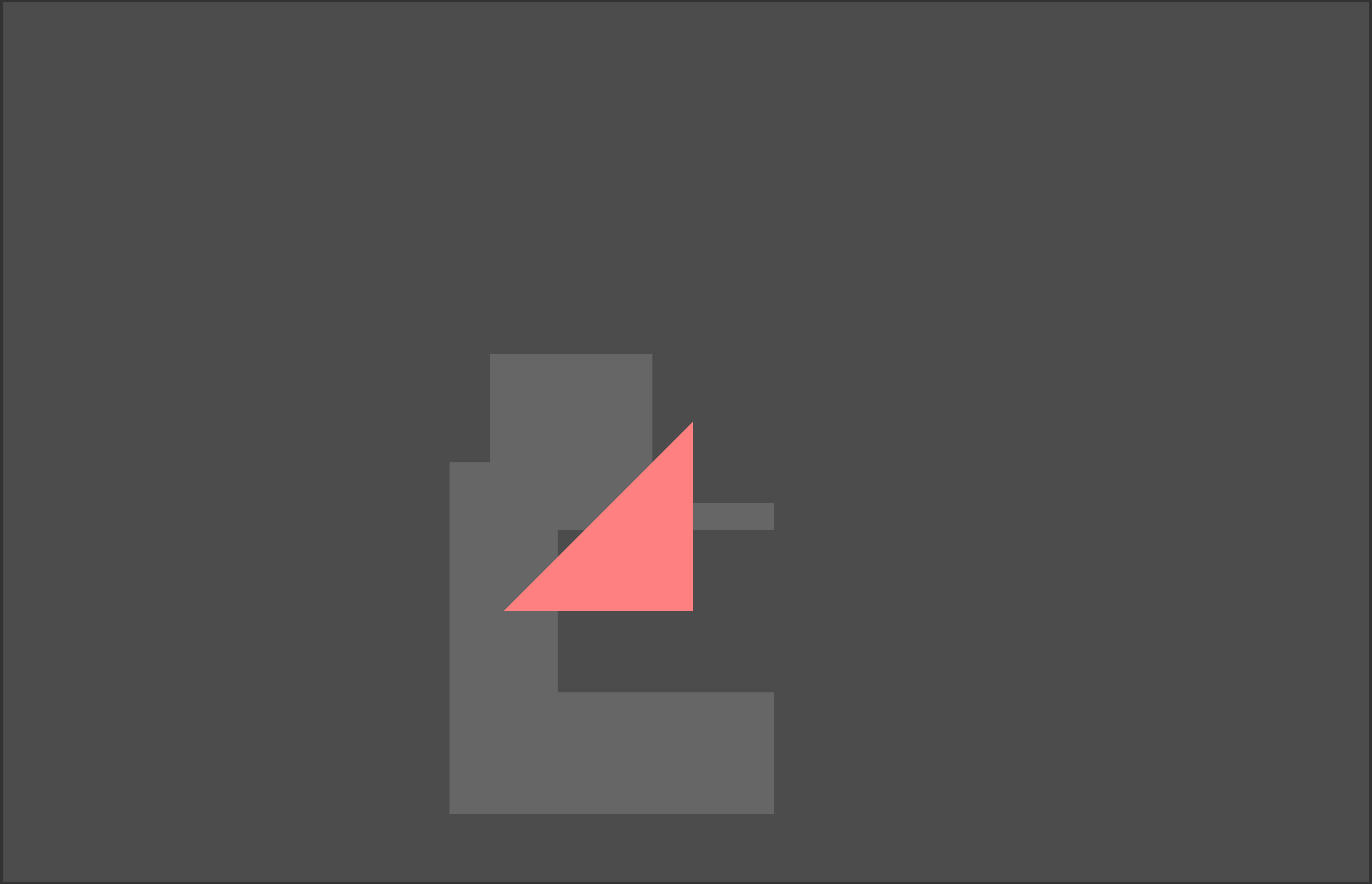
X11 Rendering



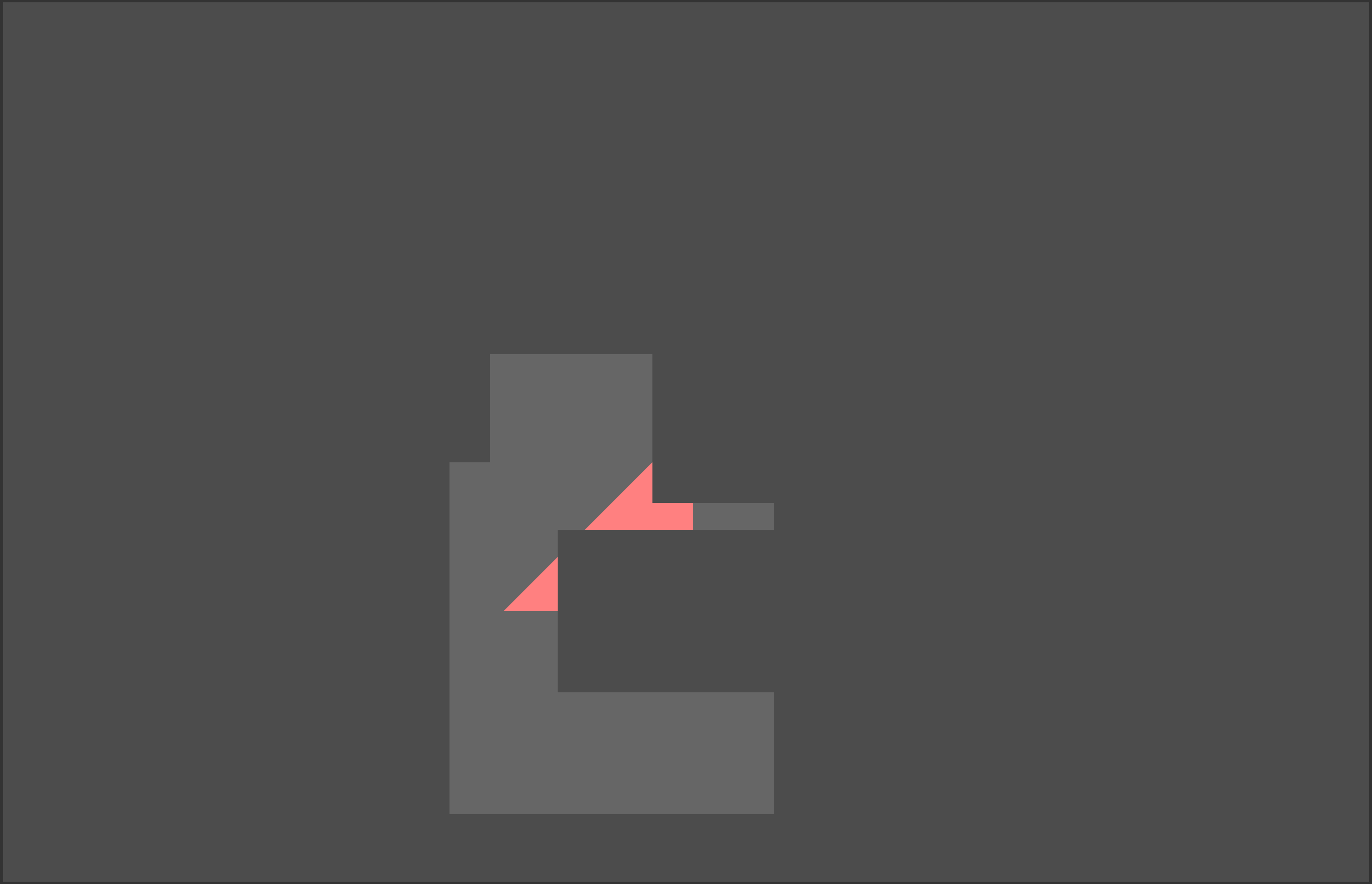
X11 Rendering



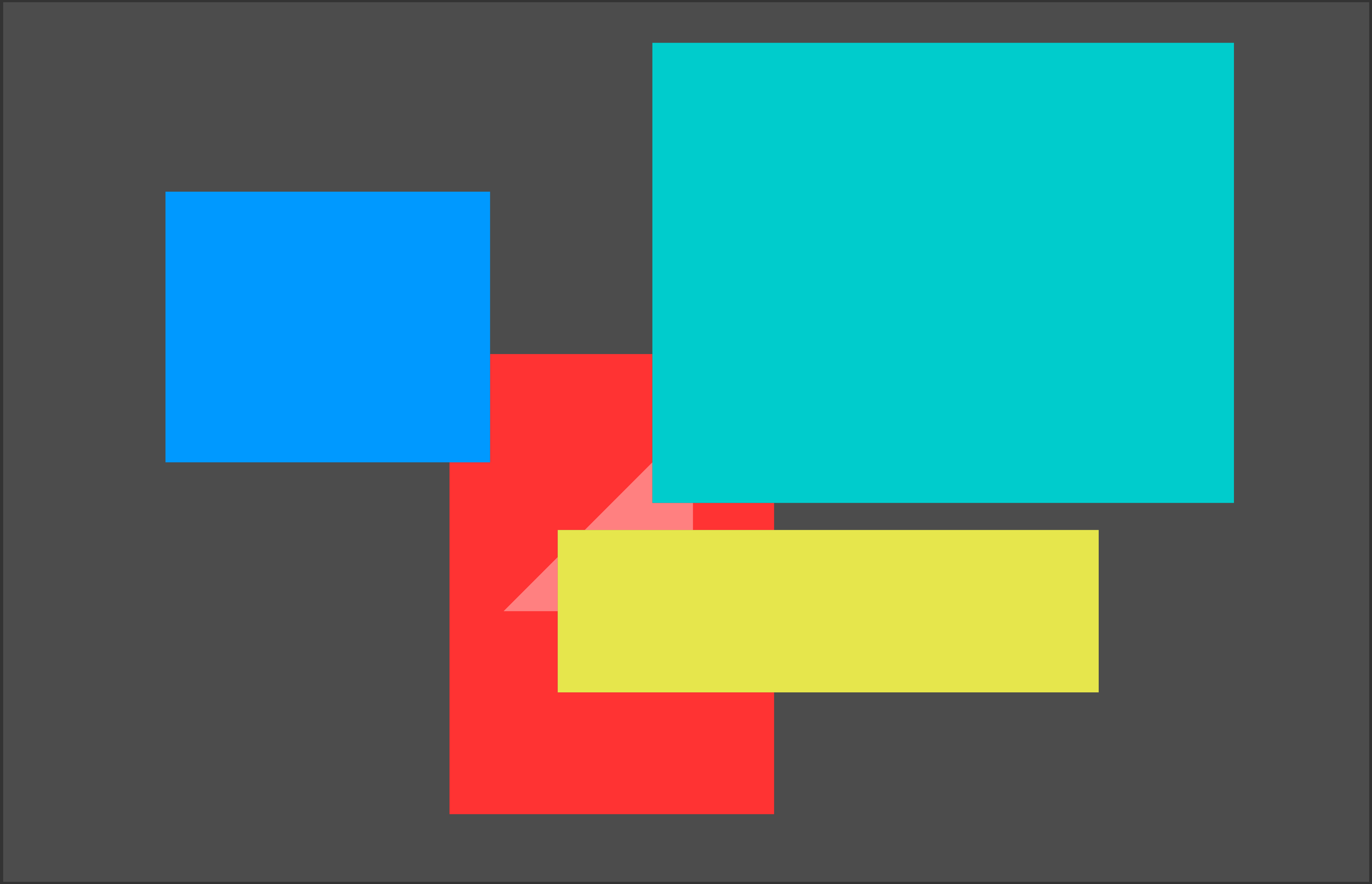
X11 Rendering



X11 Rendering



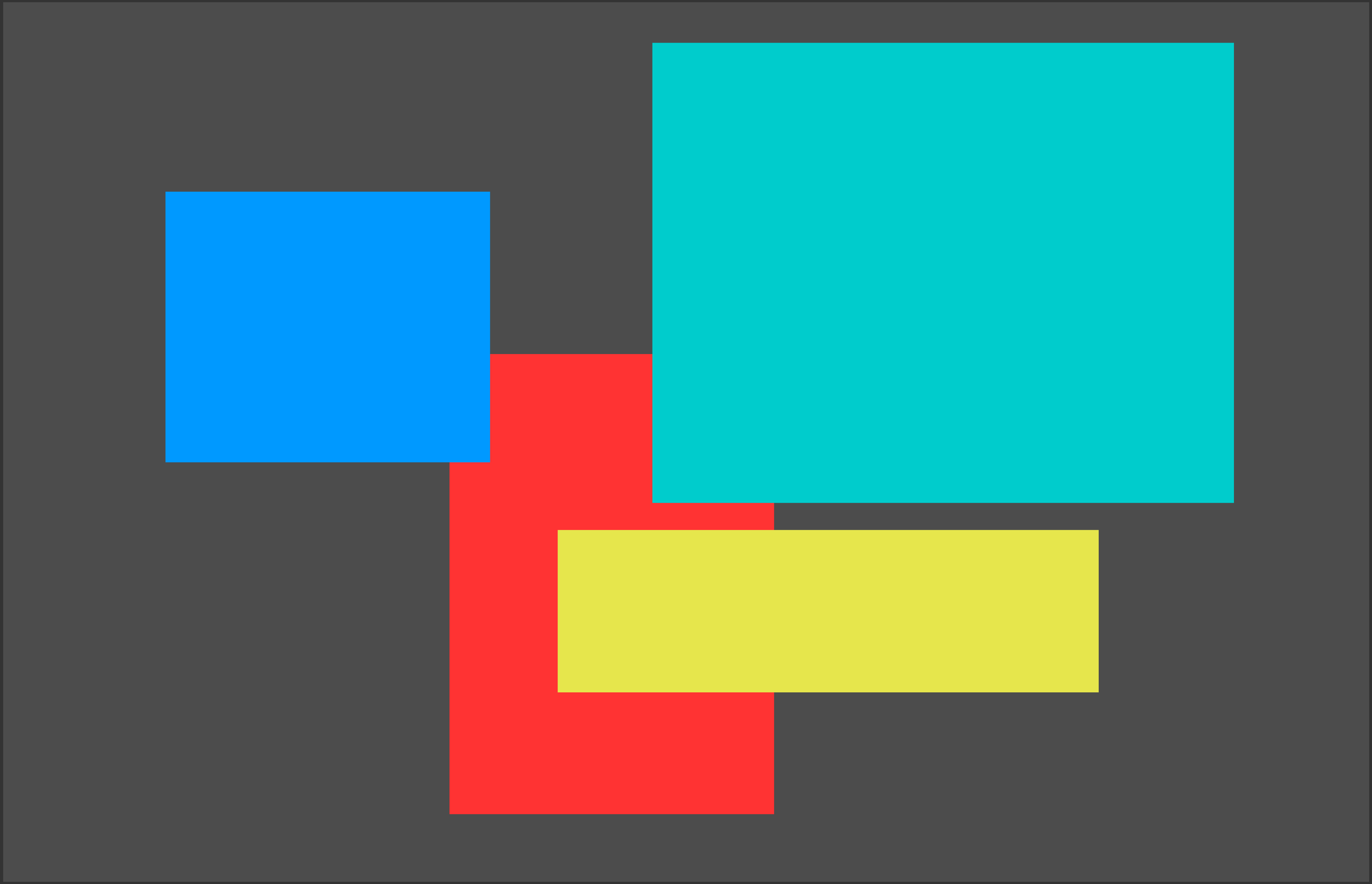
X11 Rendering



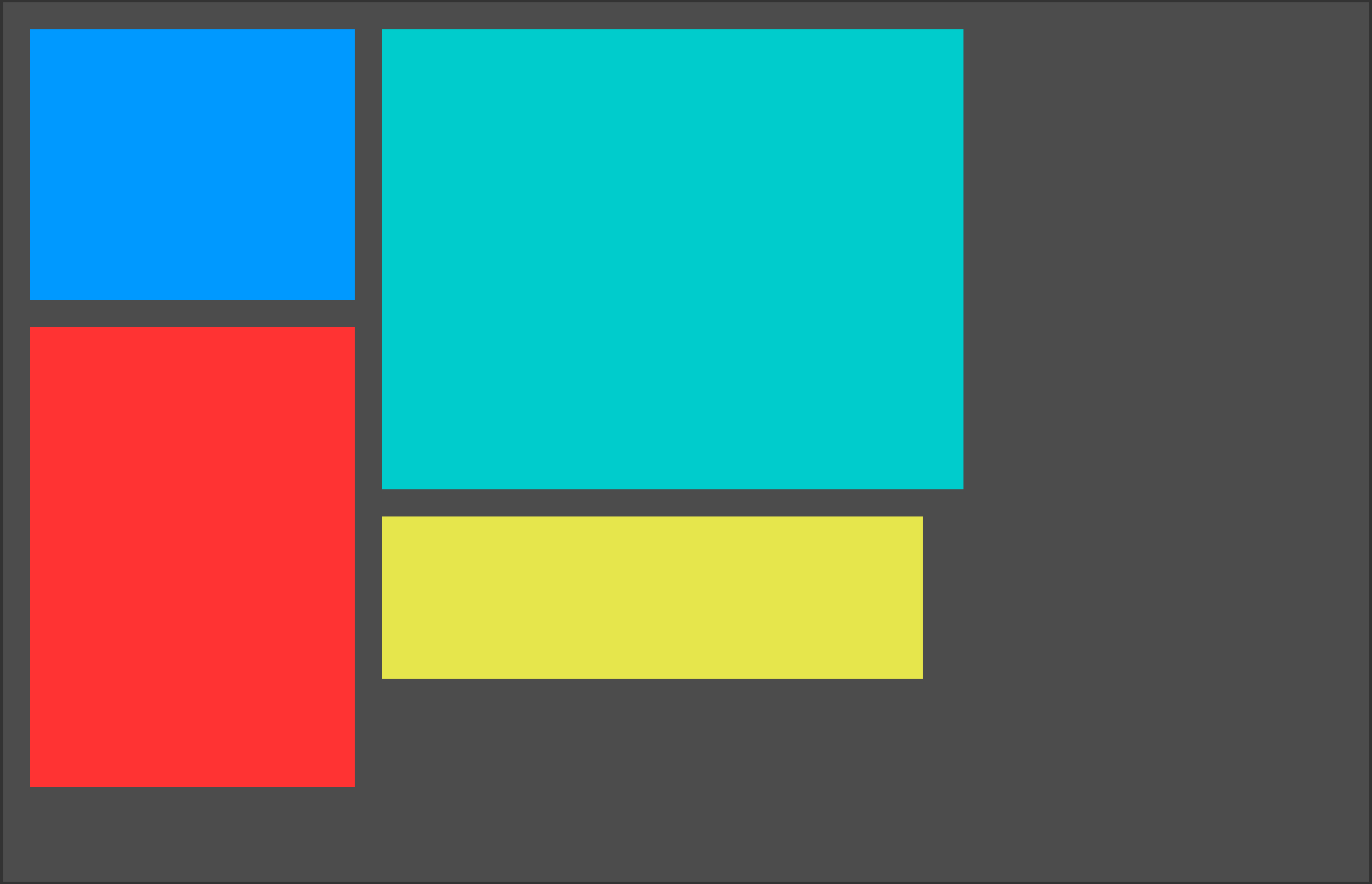
X11 Rendering (Composited)

- Composited X11 forces rendering to a window to redirect
  - Goes to off-screen pixmap that mimics window size
  - Pixmap allocated by Xserver automatically on resize
  - If window is obscured, all rendering still happens

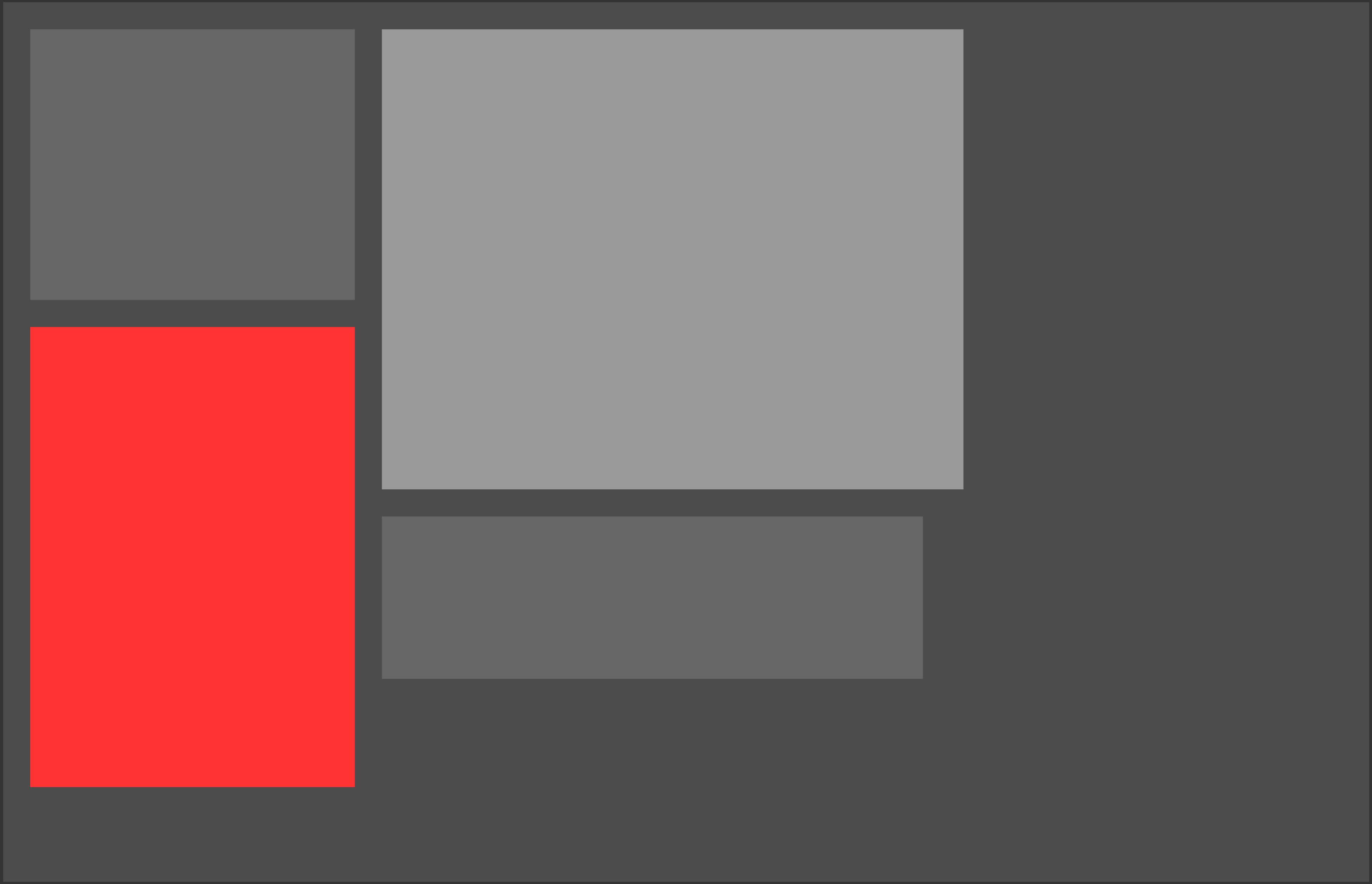
X11 Rendering (Composited)



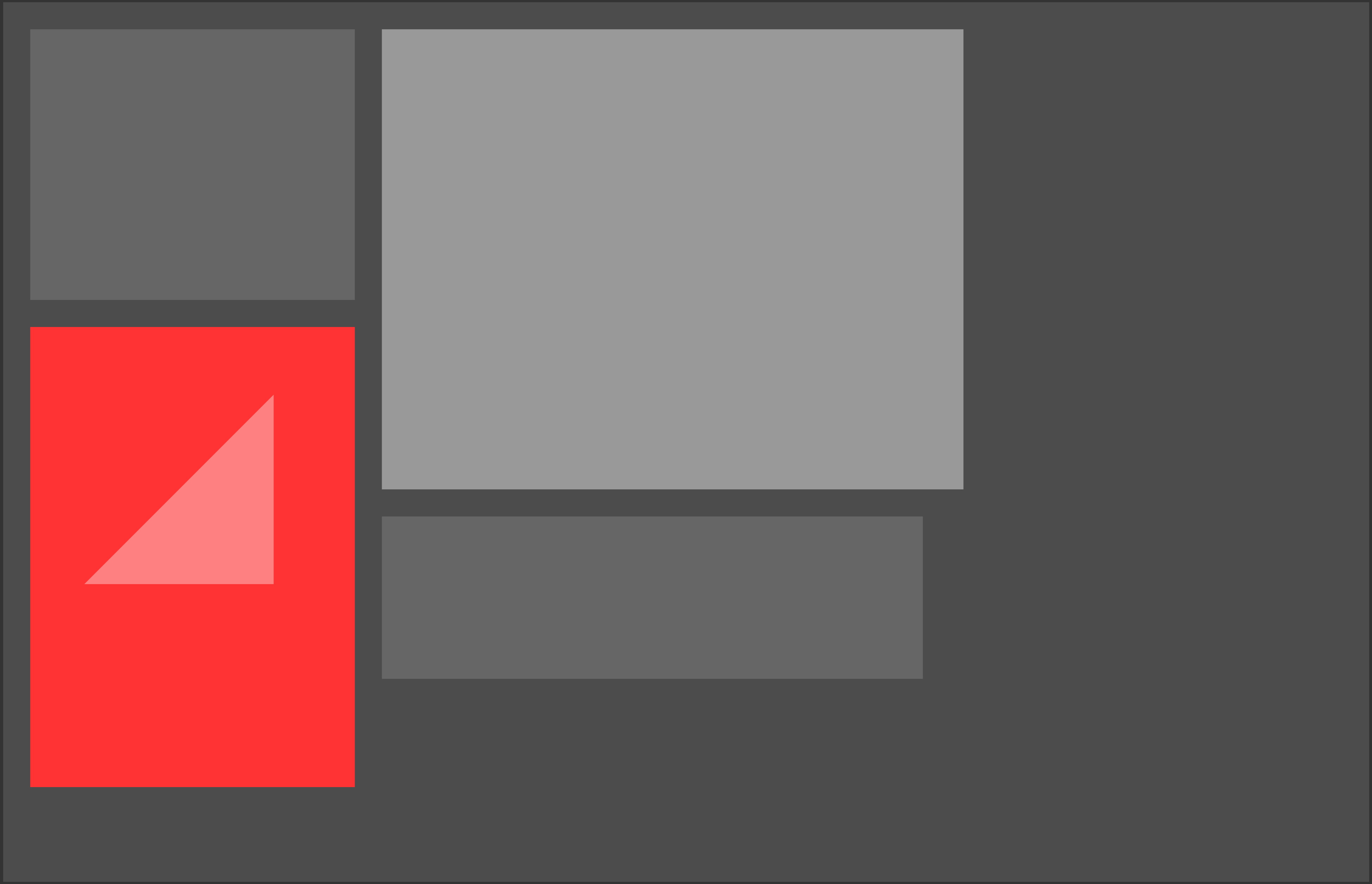
X11 Rendering (Composited)



X11 Rendering (Composited)

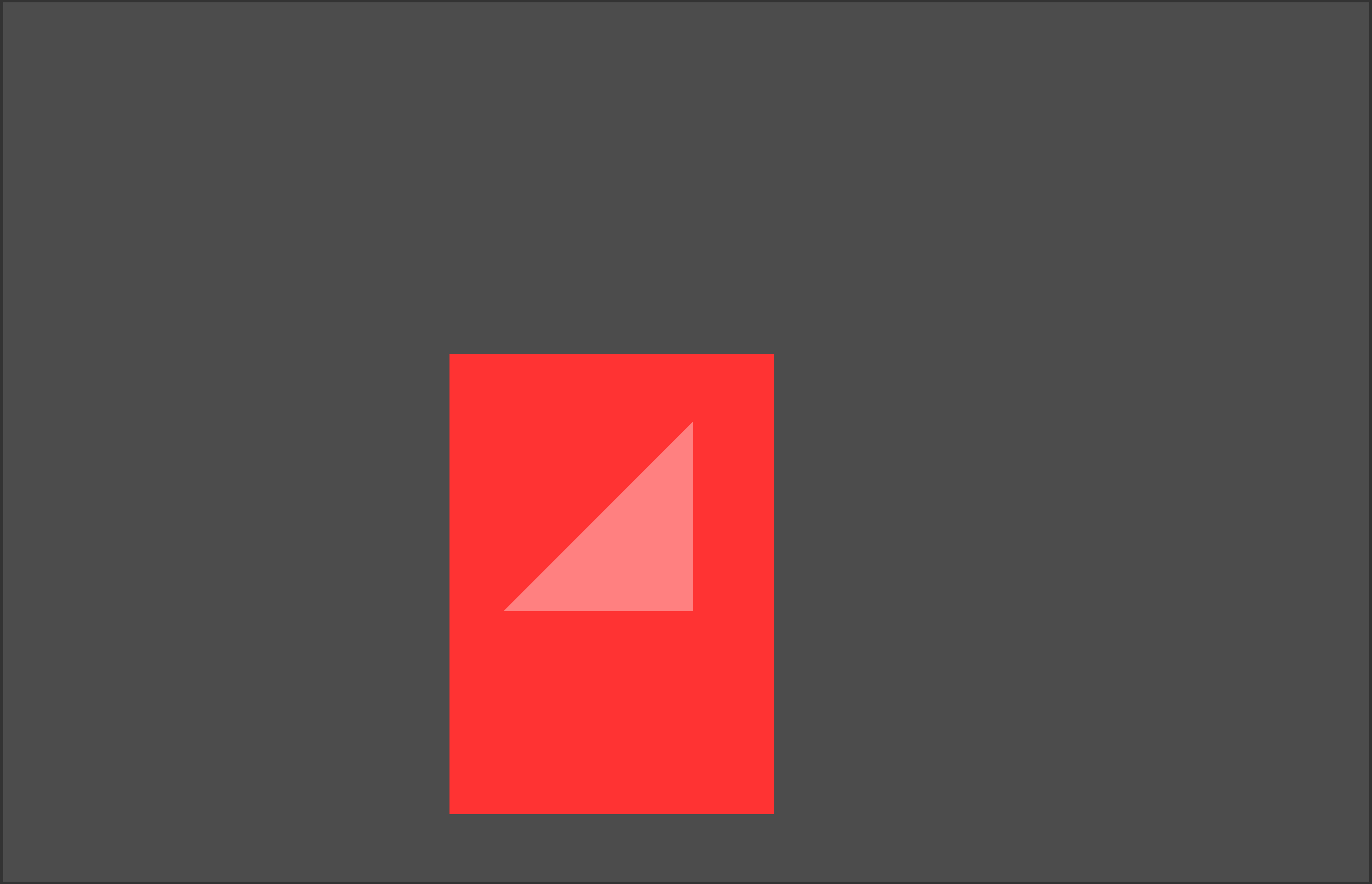


X11 Rendering (Composited)

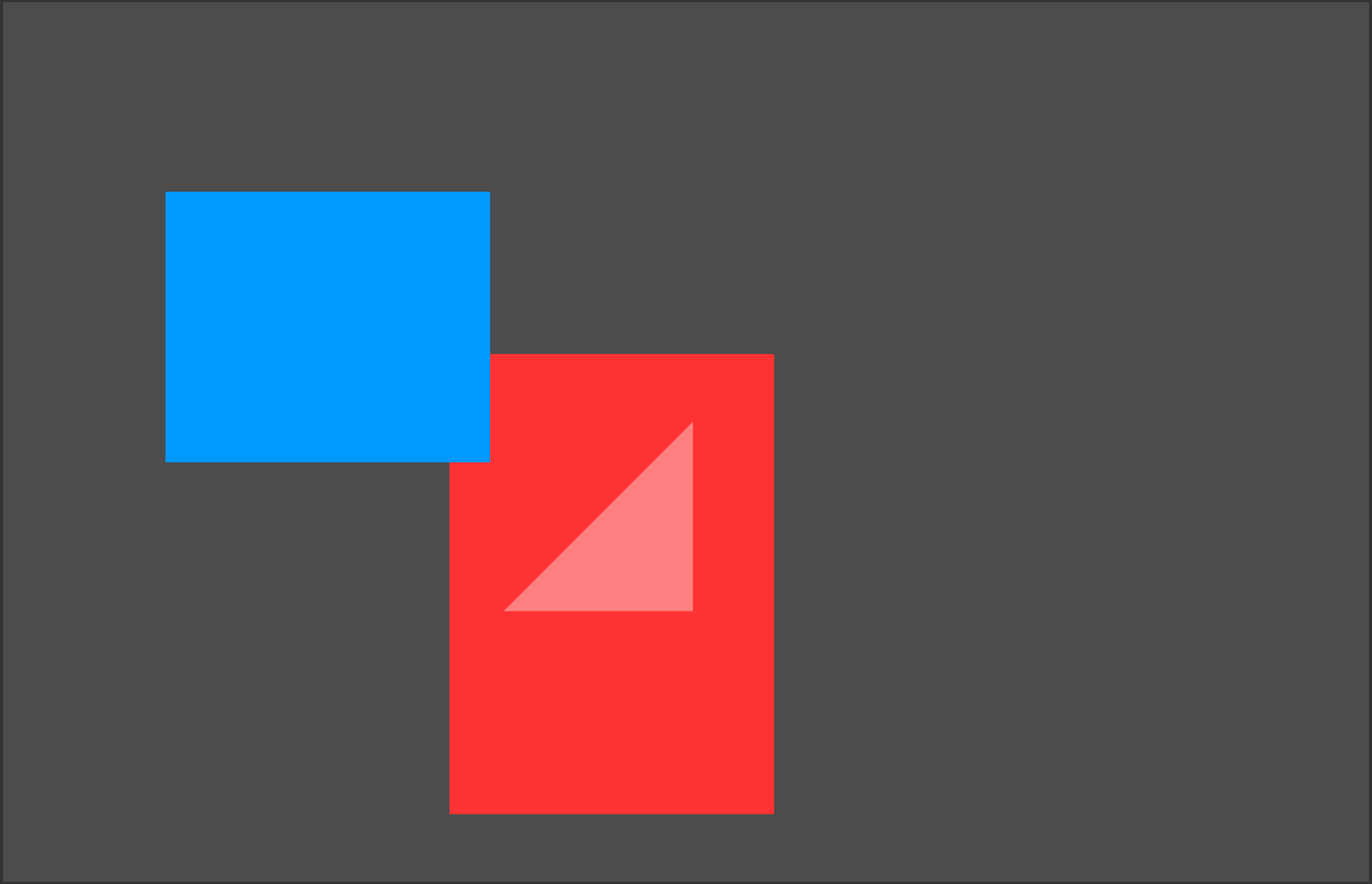


X11 Rendering (Composited)

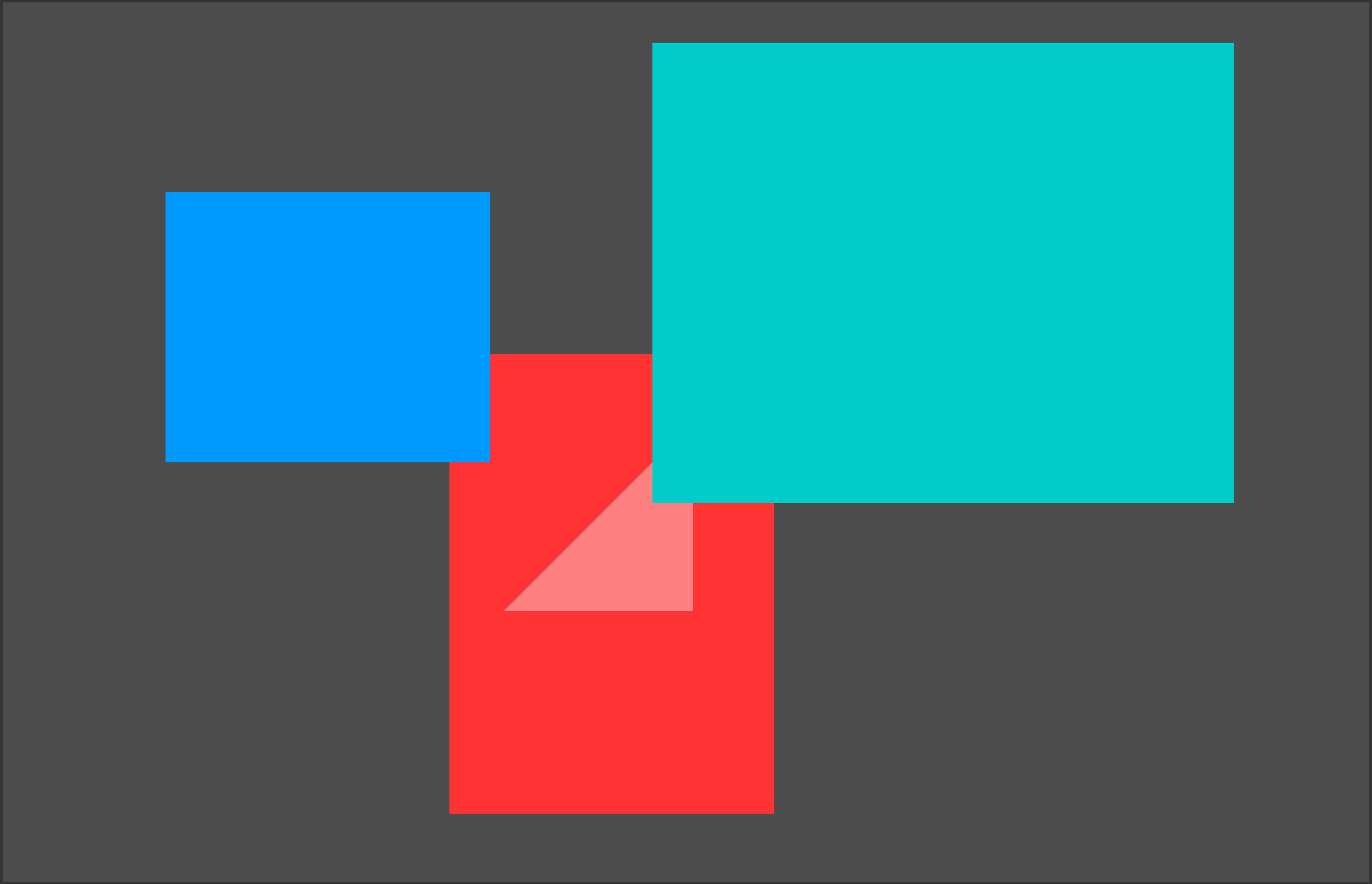
X11 Rendering (Composited)



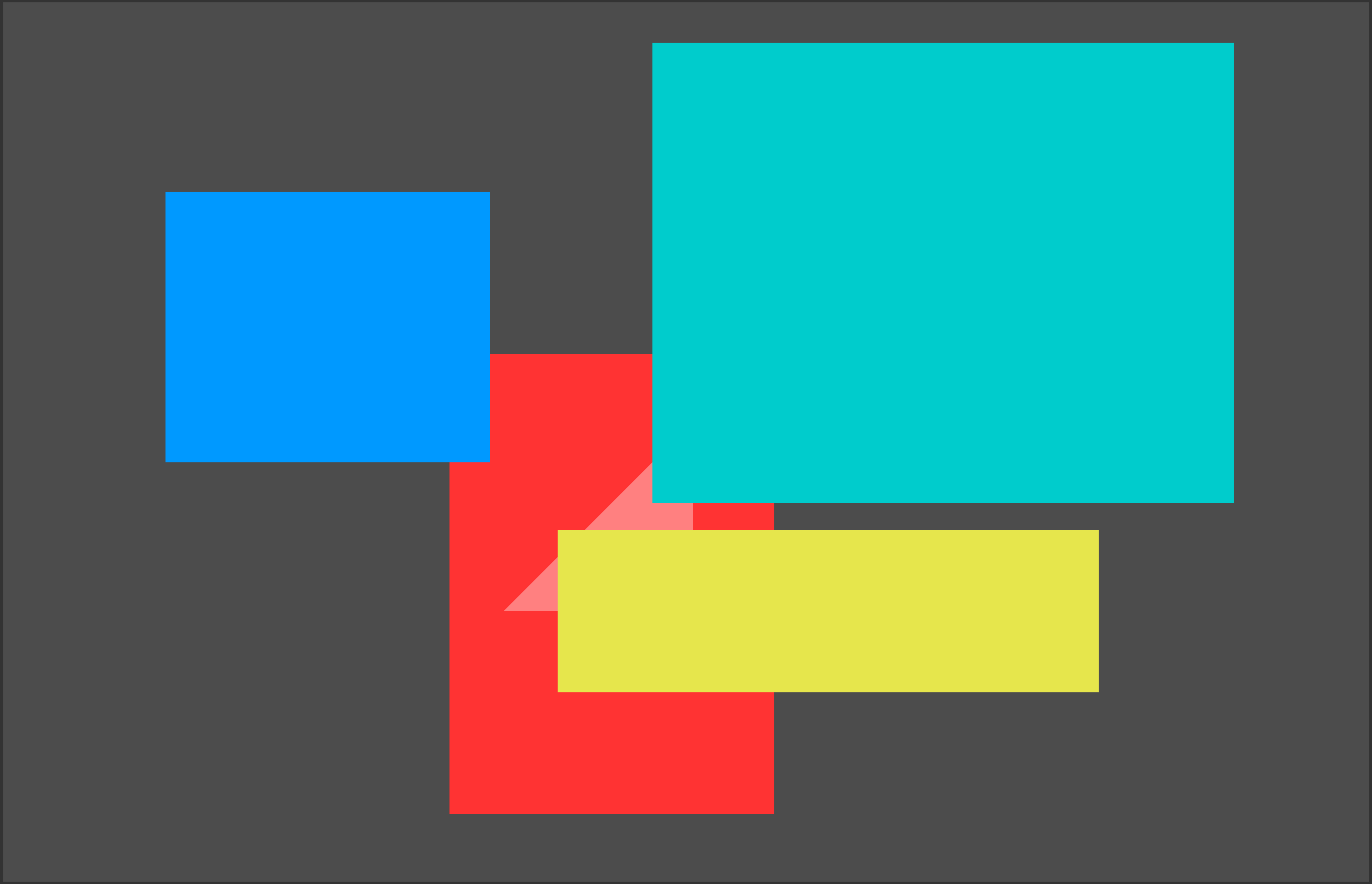
X11 Rendering (Composited)



X11 Rendering (Composited)



X11 Rendering (Composited)



Wayland Rendering

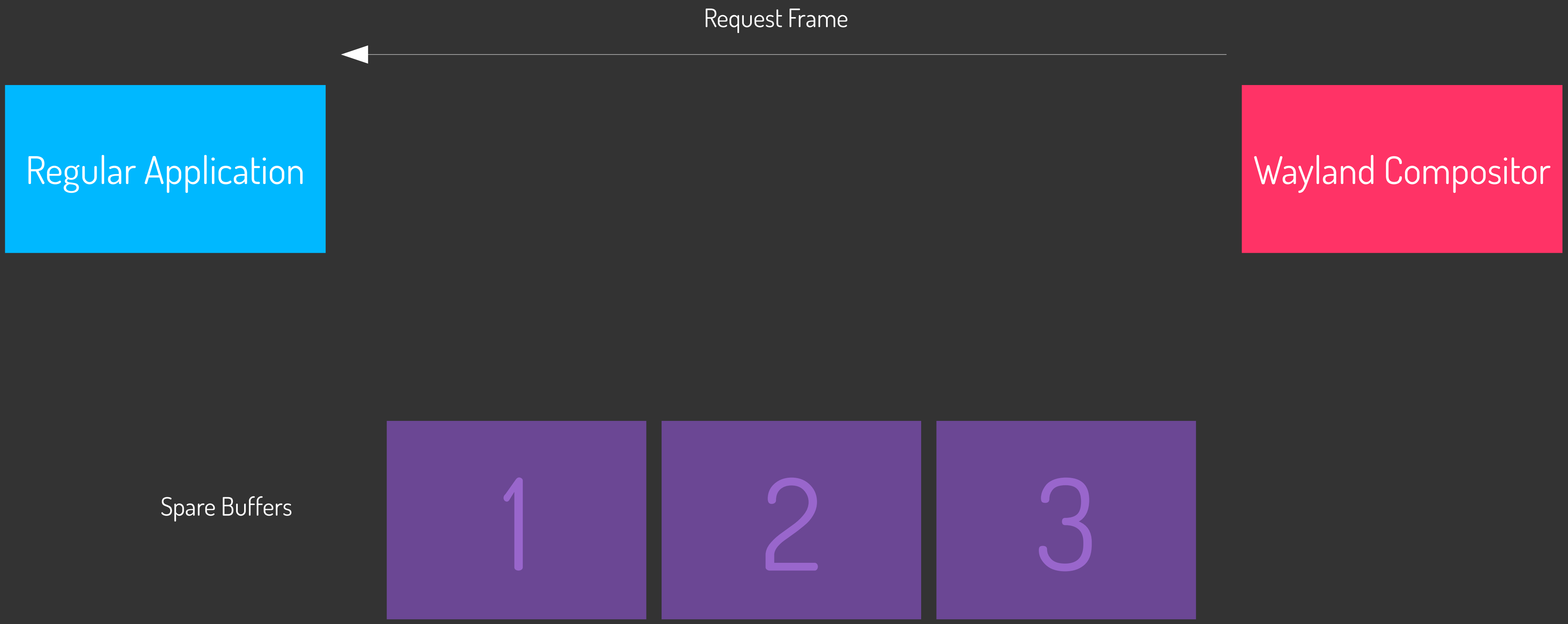
- Closer to X11 Composited Rendering
  - Every Window (Surface) displays a buffer
    - Compositor is in charge of desicding how to display the buffer
  - Clients allocate and fill buffers
    - Can render to buffer any way they like
      - Compositor is not involved in rendering and doesn't know how
    - Send buffer to compositor when done
      - Compositor may need to render to display buffer or assign to hardware scanout

Wayland Rendering

Regular Application

Wayland Compositor

Wayland Rendering



Request Frame

Regular Application

Wayland Compositor

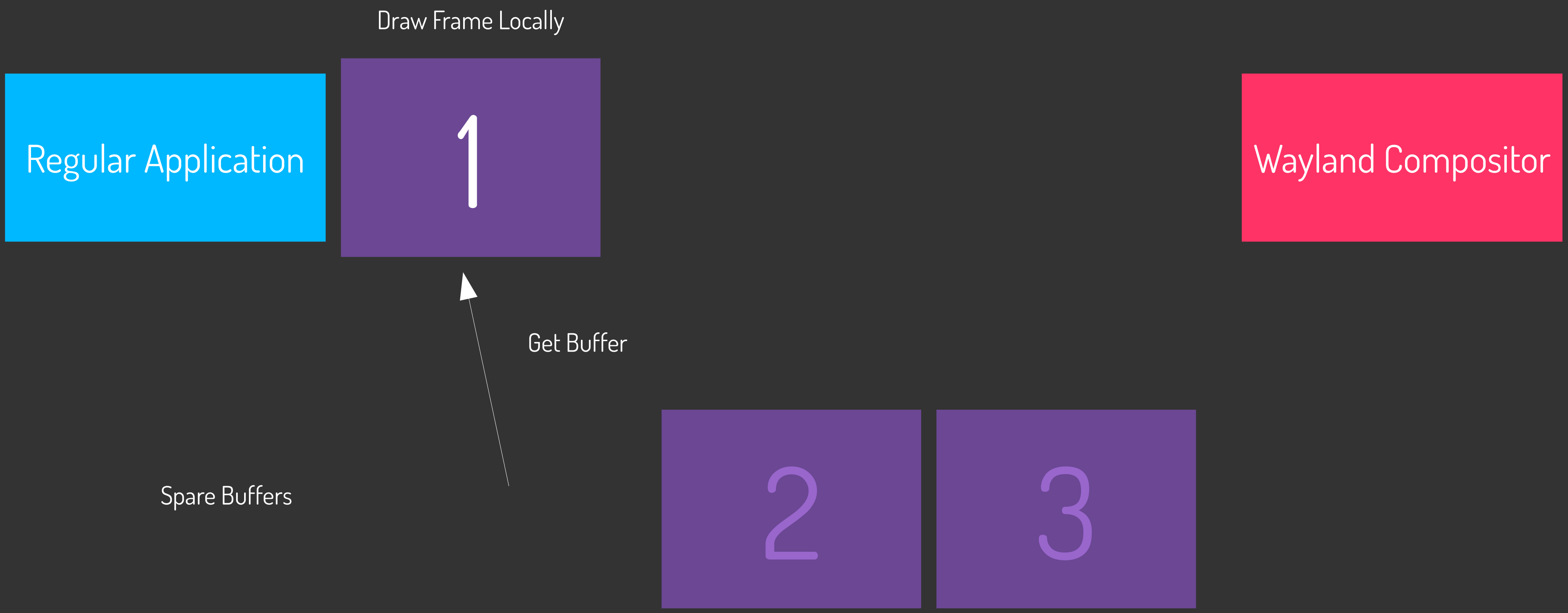
Spare Buffers

1

2

3

Wayland Rendering



Draw Frame Locally

Regular Application

1

Wayland Compositor

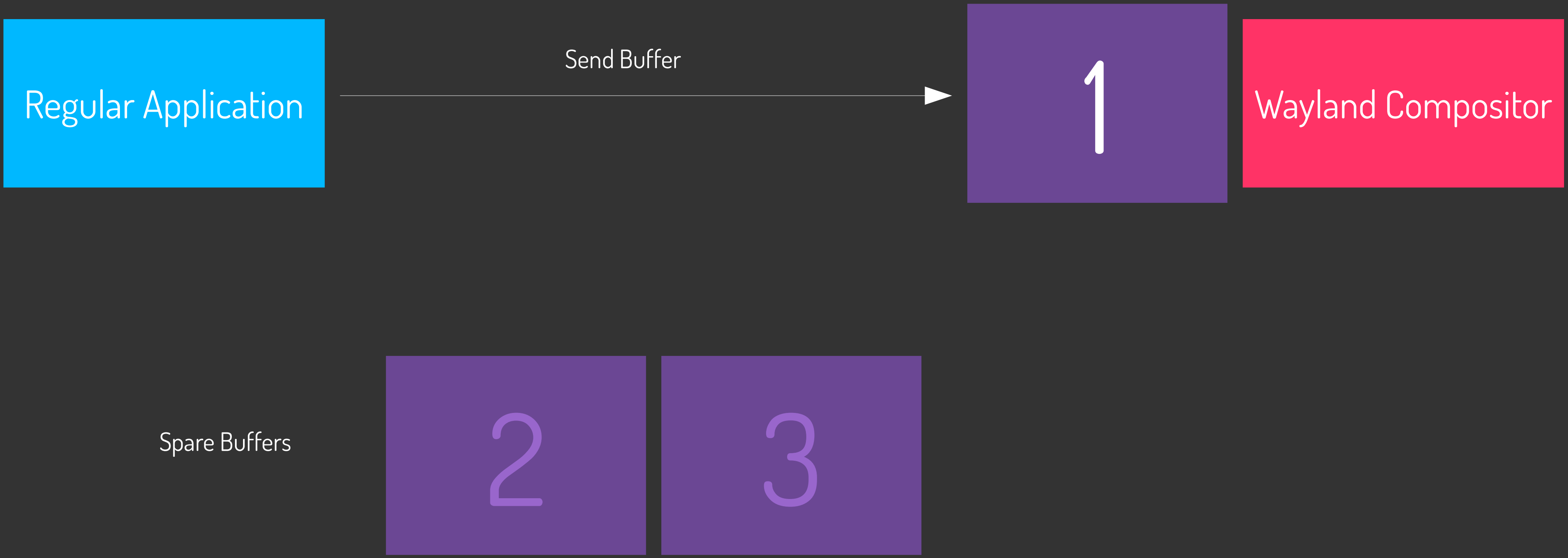
Get Buffer

Spare Buffers

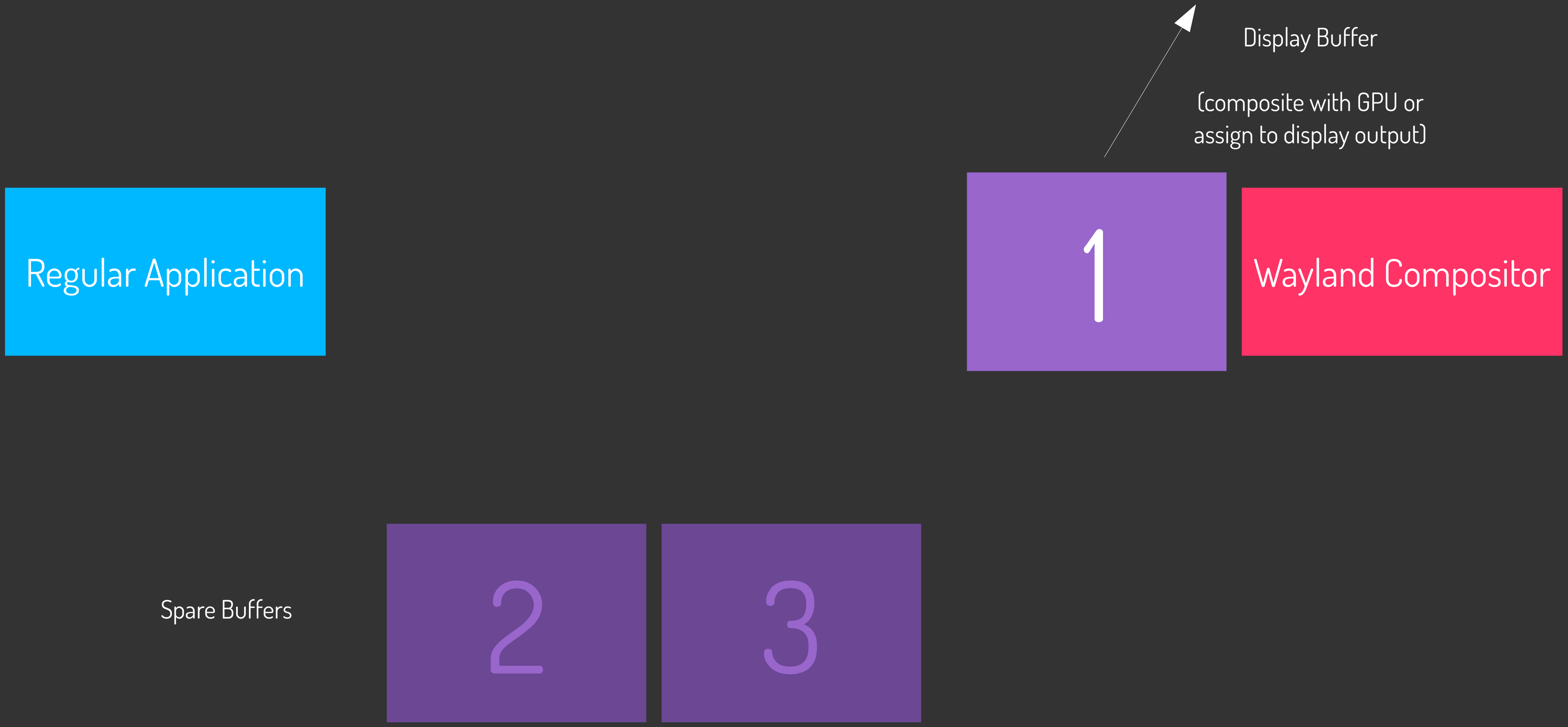
2

3

Wayland Rendering



Wayland Rendering

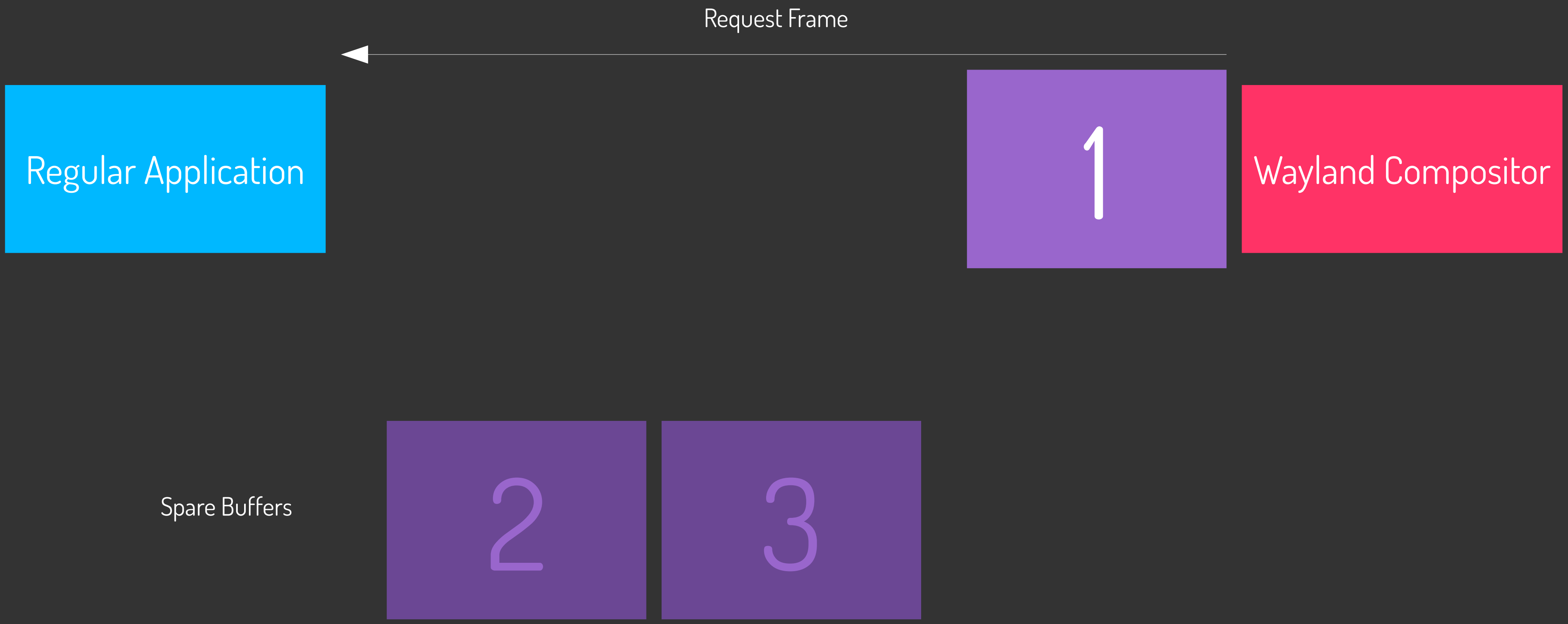


Display Buffer

(composite with GPU or assign to display output)

Spare Buffers

Wayland Rendering



Request Frame

Regular Application

1

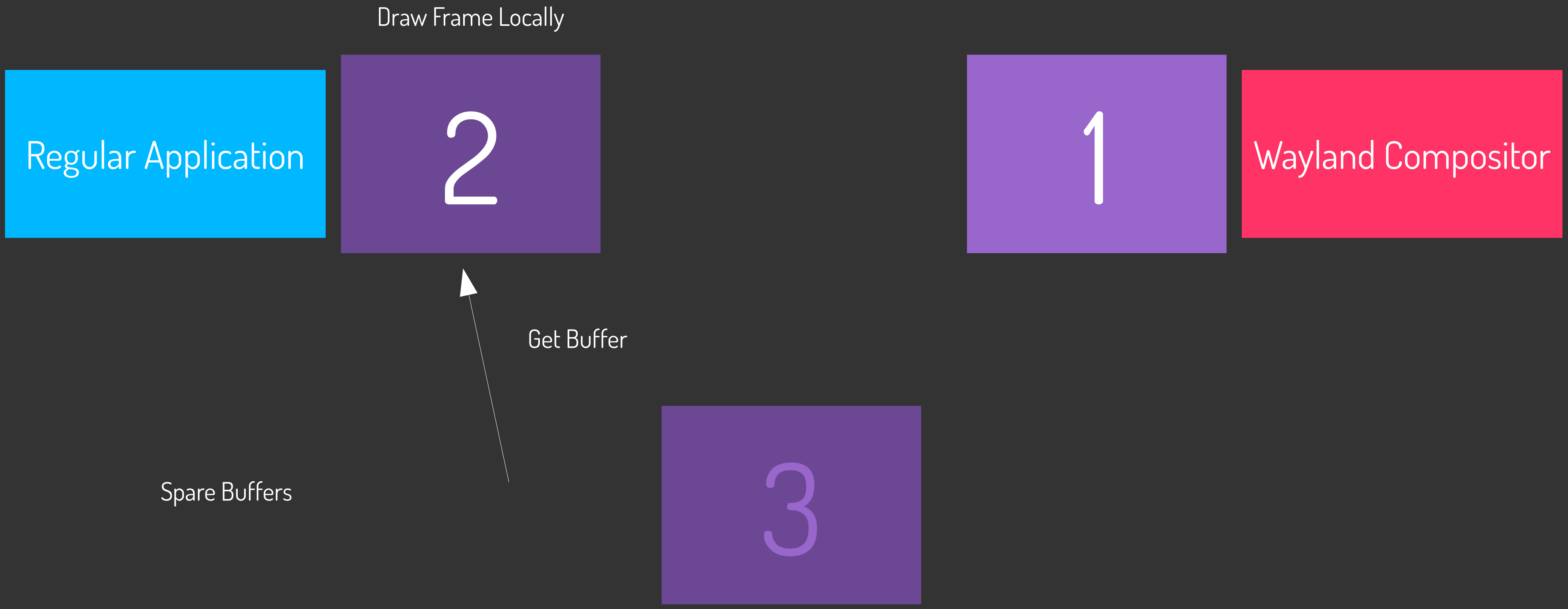
Wayland Compositor

Spare Buffers

2

3

Wayland Rendering



Draw Frame Locally

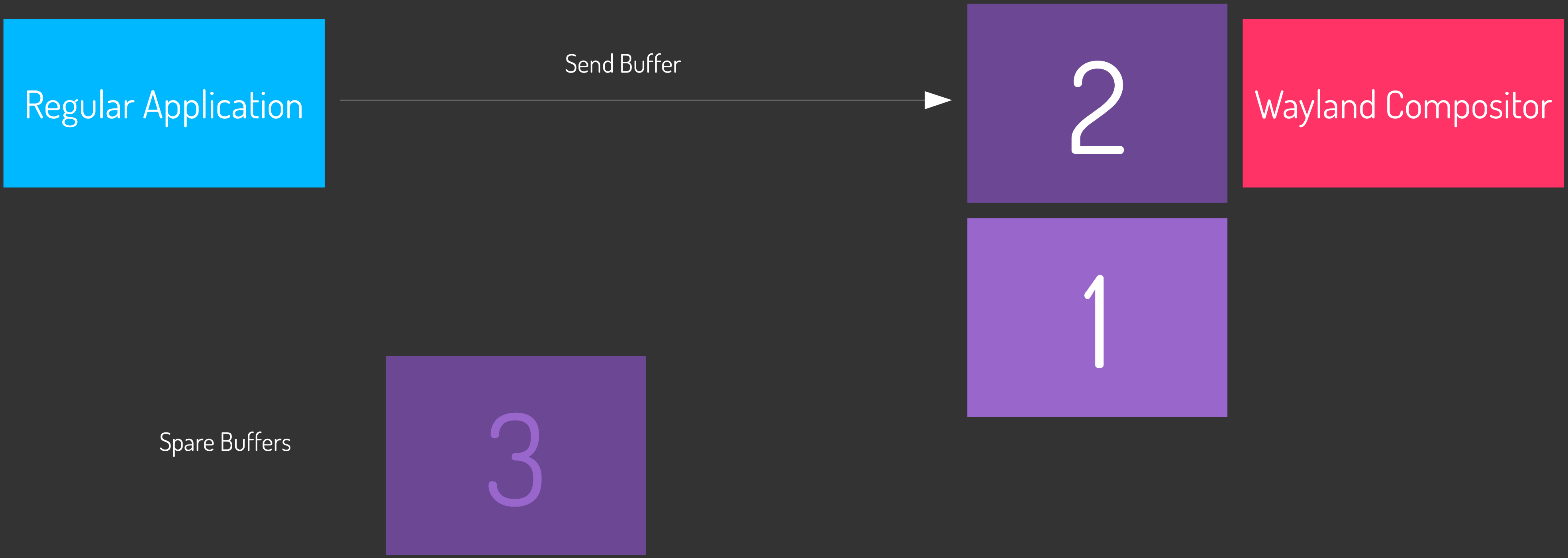
Regular Application

Wayland Compositor

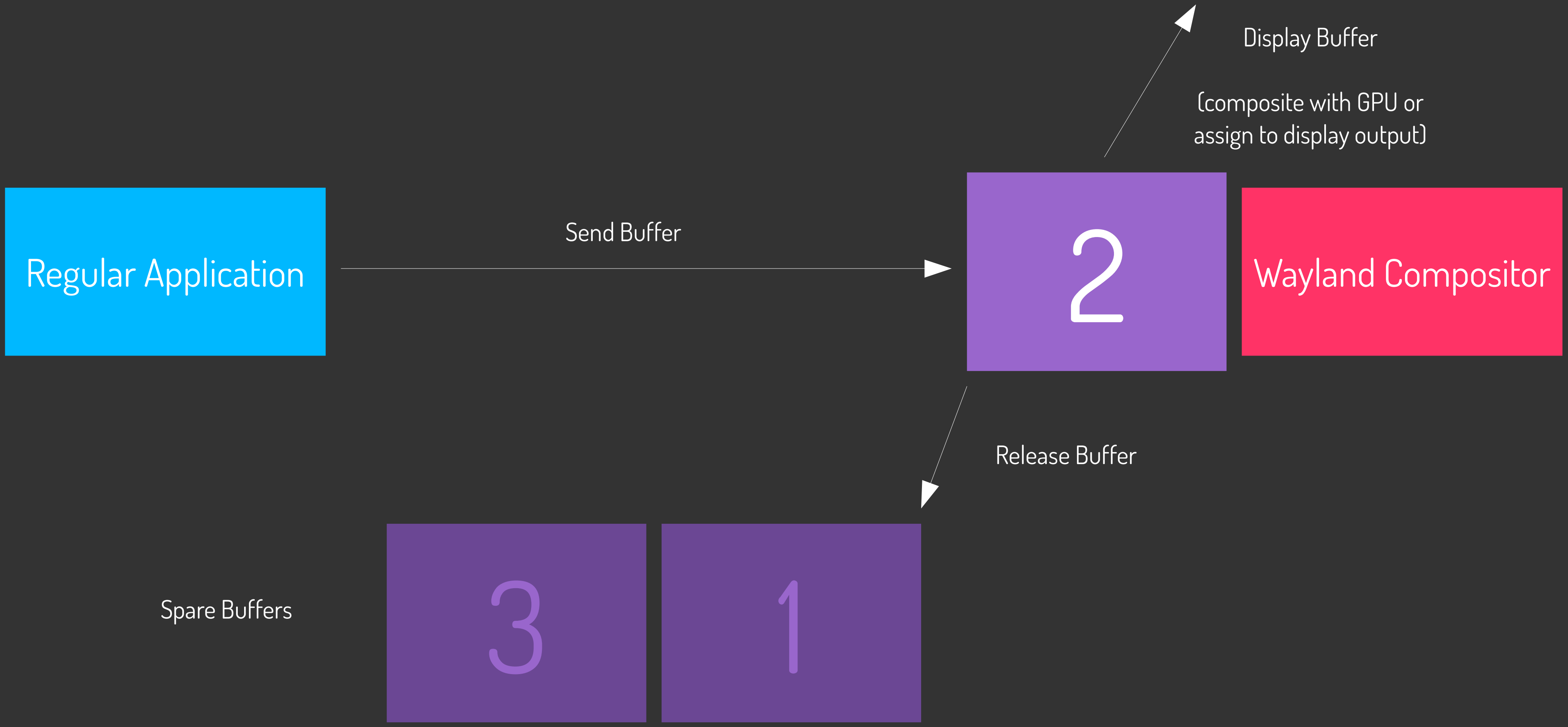
Get Buffer

Spare Buffers

Wayland Rendering



Wayland Rendering



Display Buffer

(composite with GPU or assign to display output)

Regular Application

Send Buffer

2

Wayland Compositor

Release Buffer

Spare Buffers

3

1

What this means

- Display framerate is generally controlled by compositor
  - Can be syncronized to screen refresh
- Sending a buffer is zero-copy
  - Application simply sends protocol with the buffer handle, not data
- Buffers may be Posix Shared Memory
  - `mmap()` the buffers and render directly from them or copy to texture or other destination
- Buffers may be GPU accessible memory
  - Compositor can render them by wrapping texture around buffer or assign buffer to display output hardware if possible
- Result
  - Smooth rendering with no tearing and no unnecessary copies

So…

Wayland is better than X11
Wayland is good for Tizen

Transition

How did the transition to Wayland happen

- Had to transition 2 major things
  - Client side application toolkit
    - Allow applications to display and get input from any Wayland compositor
  - Compositor/Window Manager
    - Enlightenment uses same toolkit as clients
- Client-side toolkit started first
  - Had an existing compositor (Weston) to test against

Client-side

- Ported window layer
  - Windows in X11, Windows, OSX etc. - “surfaces” in Wayland
- Ported rendering
  - First SHM buffer rendering
    - Simpler and relied on no specific driver support
    - All rendering already done for other targets – just need a different target
- Ported input
  - Need to get Mouse and Keyboard input events
    - Are now extending more advanced input devices
- Ported EGL/OpenGL-ES
  - Similar to X11 EGL+GL but with surfaces not X11 windows
  - EGL driver layer library takes care of buffer sending + management

Compositor-side

- Needed to add display engine for:
  - KMS/DRM display (configure display via KMS)
  - Software rendering to fill DRM buffers
    - Map, fill, display
  - EGL+GL for hardware acceleration
- LibInput
  - Use this library to get access to input devices
    - Send input to specific clients

Compositor-side

- Compositor
  - Had to make compositing non-optional
    - X11 allowed compositing as an add-on feature
    - Implmented by extra plug-in module and X11 infrastructure
    - Compositing in core as a non-optional design → the only sane way forward
  - Use new engines
    - Use X11 engines (Software, GL) for X11 compositing
    - Use new DRM and GL DRM modules for software an hardware accelerated display direct to KMS/FB
  - Remove/isolate X11 specific code
    - Window management code for X11 vs Wayland client management
    - X11 code for screen management (Randr)
    - X11 code for backlight controls
    - ... and much more

Results

- Enlightenment now is BOTH:
  - X11 WM+Compositor
  - Wayland Compositor (direct to KMS/FB)
    - *(can even be Wayland compositor in-a-window in X11 like Weston)*
- EFL using apps
  - Can work in X11 AND Wayland
    - And Windows, OSX, basic /dev/fb, ...
- Tizen can move to Wayland
  - Enlightenment is now Tizen's Wayland compositor (Mobile, TV, Wearable ...)
  - Most clients use EFL as the toolkit → so clients work too
  - Still have lots of special use cases to solve for input and display
    - Working on them