Wayland Full-Screen Shell

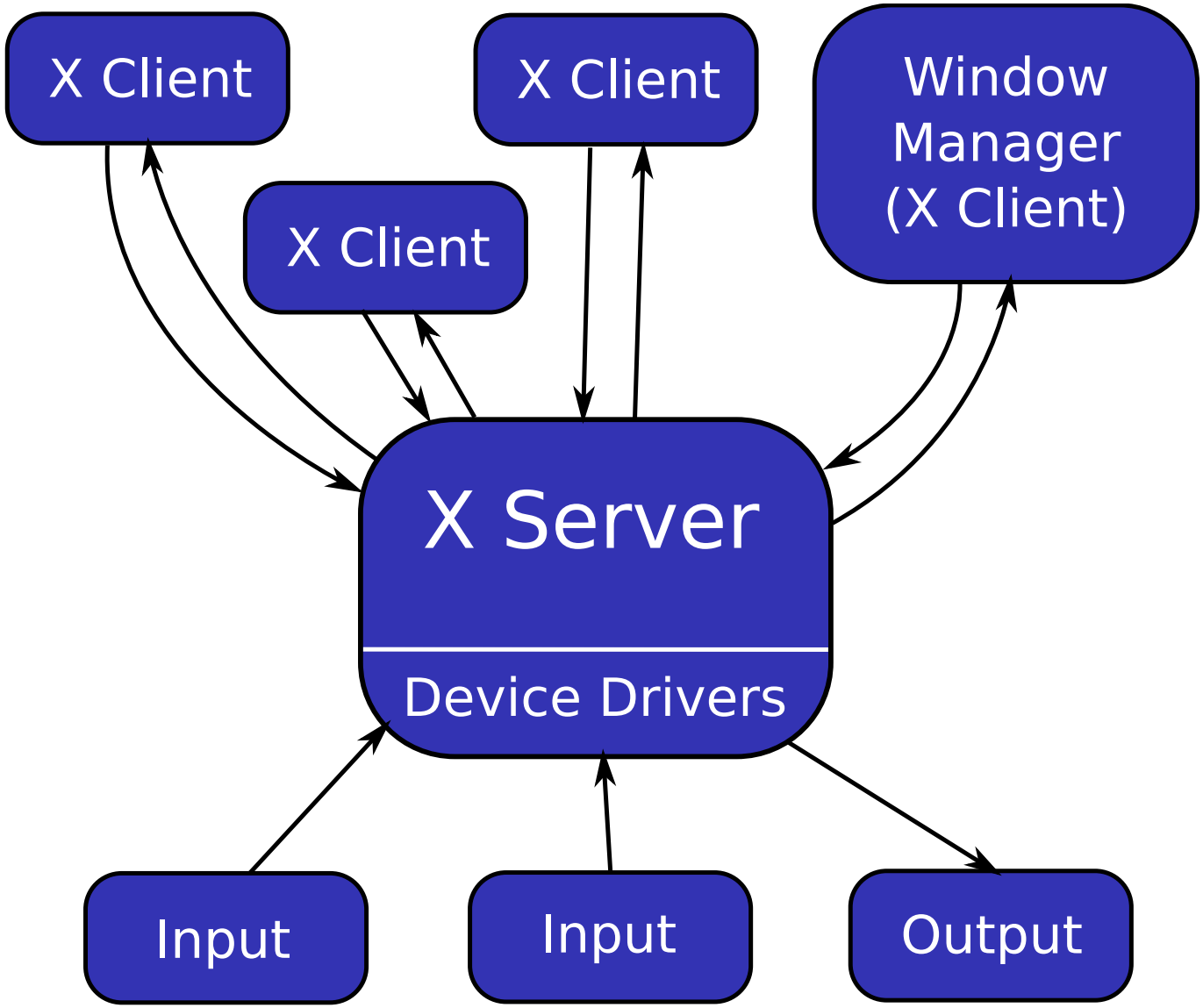
Jason Ekstrand

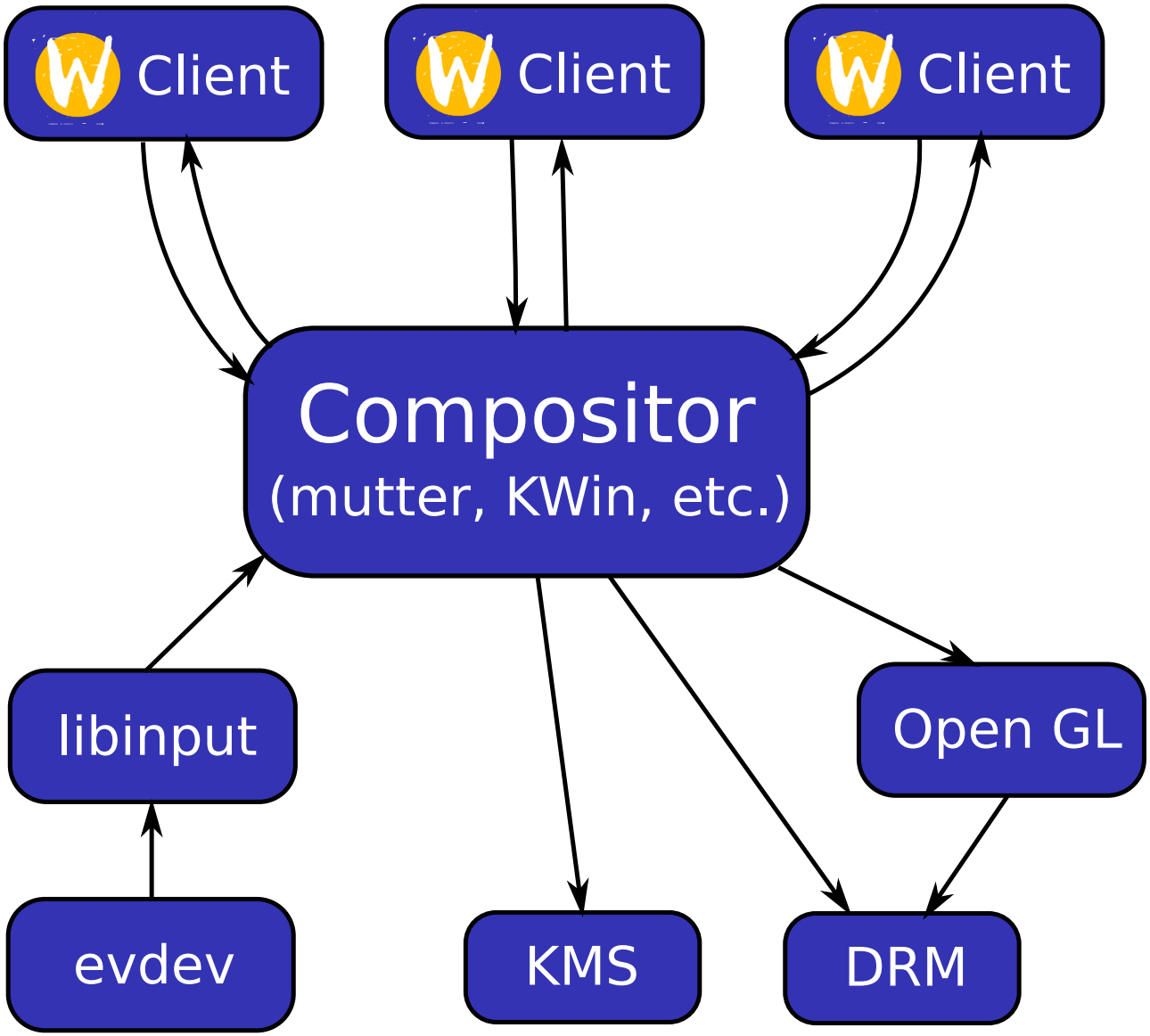
Intel Corporation
Open-Source 3D Driver Team

October 6, 2014

About Me

- Ph.D. student in mathematics at Iowa State University
- Involved in Wayland since early 2013
- Working for Intel on the i965 driver since June


X Client
X Client
X Client
Window Manager (X Client)
X Server
Device Drivers
Input
Input
Output


Client
Client
Client
Compositor
(mutter, KWin, etc.)
libinput
Open GL
evdev
KMS
DRM

What about code re-use?

We push the common functionality into external libraries:

- KMS for modesetting
- DRM for buffer graphics buffer management
- OpenGL [ES] for GPU-accelerated compositing
- pixman for CPU-accelerated compositing
- libinput for handling different input devices

X is using these too!

What about code re-use?

We push the common functionality into external libraries:

- KMS for modesetting
- DRM for buffer graphics buffer management
- OpenGL [ES] for GPU-accelerated compositing
- pixman for CPU-accelerated compositing
- libinput for handling different input devices

X is using these too!

What about code re-use?

We push the common functionality into external libraries:

- KMS for modesetting
- DRM for buffer graphics buffer management
- OpenGL [ES] for GPU-accelerated compositing
- pixman for CPU-accelerated compositing
- libinput for handling different input devices

X is using these too!

Why not just use X?

X is a fine compositor if you want to let it handle input and compositing.

Modern desktop environments (GNOME, KDE, E) want to handle input and compositing themselves.

Why not just use X?

X is a fine compositor if you want to let it handle input and compositing.

Modern desktop environments (GNOME, KDE, E) want to handle input and compositing themselves.

Why not just use X?

X is a fine compositor if you want to let it handle input and compositing.

Modern desktop environments (GNOME, KDE, E) want to handle input and compositing themselves.

X uses a common global namespace for everything

- Any client can resize/reposition any window
- Any client can get the contents of any window
- Any client can give any other client input

Window managers are just clients that manage other clients windows using the above mechanisms

X uses a common global namespace for everything

- Any client can resize/reposition any window
- Any client can get the contents of any window
- Any client can give any other client input

Window managers are just clients that manage other clients windows using the above mechanisms

X uses a common global namespace for everything

- Any client can resize/reposition any window
- Any client can get the contents of any window
- Any client can give any other client input

Window managers are just clients that manage other clients windows using the above mechanisms

In Wayland...

- The compositor is at the center and each client has its own namespace
- Clients get input directly from the compositor
- Clients aren't, in general, aware of other clients' existence
- A client's surface contents is kept between the compositor and the client

This has all sorts of security and other benefits

In Wayland...

- The compositor is at the center and each client has its own namespace
- Clients get input directly from the compositor
- Clients aren't, in general, aware of other clients' existence
- A client's surface contents is kept between the compositor and the client

This has all sorts of security and other benefits

In Wayland...

- The compositor is at the center and each client has its own namespace
- Clients get input directly from the compositor
- Clients aren't, in general, aware of other clients' existence
- A client's surface contents is kept between the compositor and the client

This has all sorts of security and other benefits

The problem

- X provided a common *userspace* input/output layer
- ”All you have to do” is implement a DDX and everything just runs on it (more-or-less)
- DRM, KMS, and OpenGL are focused on getting images to hardware
- Not all output devices are hardware

The problem

- X provided a common *userspace* input/output layer
- ”All you have to do” is implement a DDX and everything just runs on it (more-or-less)
- DRM, KMS, and OpenGL are focused on getting images to hardware
- Not all output devices are hardware

The problem

- X provided a common *userspace* input/output layer
- ”All you have to do” is implement a DDX and everything just runs on it (more-or-less)
- DRM, KMS, and OpenGL are focused on getting images to hardware
- Not all output devices are hardware

The problem

- X provided a common *userspace* input/output layer
- ”All you have to do” is implement a DDX and everything just runs on it (more-or-less)
- DRM, KMS, and OpenGL are focused on getting images to hardware
- Not all output devices are hardware

Case Study 1: VNC/RDP Sessions

How this is done on X:

- Implement either a full X server or a DDX for X.org
- Lives entirely in userspace and doesn't require root
- Clients and window managers talk to it like any other X server

Several projects such as FreeRDP or x11vnc have done this

Case Study 1: VNC/RDP Sessions

How this is done on X:

- Implement either a full X server or a DDX for X.org
- Lives entirely in userspace and doesn't require root
- Clients and window managers talk to it like any other X server

Several projects such as FreeRDP or x11vnc have done this

Case Study 1: VNC/RDP Sessions

How this is done on X:

- Implement either a full X server or a DDX for X.org
- Lives entirely in userspace and doesn't require root
- Clients and window managers talk to it like any other X server

Several projects such as FreeRDP or x11vnc have done this

Case Study 1: VNC/RDP Sessions

This poses some problems for Wayland:

- Current input/output abstractions assume hardware
- DRM and KMS require a kernel driver
- External libraries exist, but require native support in every compositor.

Yes, Weston has an RDP backend, but adding backends for every network protocol to every compositor isn't a long-term solution.

Case Study 1: VNC/RDP Sessions

This poses some problems for Wayland:

- Current input/output abstractions assume hardware
- DRM and KMS require a kernel driver
- External libraries exist, but require native support in every compositor.

Yes, Weston has an RDP backend, but adding backends for every network protocol to every compositor isn't a long-term solution.

Case Study 1: VNC/RDP Sessions

This poses some problems for Wayland:

- Current input/output abstractions assume hardware
- DRM and KMS require a kernel driver
- External libraries exist, but require native support in every compositor.

Yes, Weston has an RDP backend, but adding backends for every network protocol to every compositor isn't a long-term solution.

Case Study 2: Screen recording/sharing

How this is done on X:

- Implemented as a regular client
- Any client can grab any other clients contents or the entire front buffer
- Any client can send ”input” to any other client

This raises huge security concerns:

- Clients can grab sensitive information displayed by other clients
- Clients can fake input and control other clients. (What if they are running as root?)

Case Study 2: Screen recording/sharing

How this is done on X:

- Implemented as a regular client
- Any client can grab any other clients contents or the entire front buffer
- Any client can send ”input” to any other client

This raises huge security concerns:

- Clients can grab sensitive information displayed by other clients
- Clients can fake input and control other clients. (What if they are running as root?)

Case Study 2: Screen recording/sharing

How this is done on X:

- Implemented as a regular client
- Any client can grab any other clients contents or the entire front buffer
- Any client can send ”input” to any other client

This raises huge security concerns:

- Clients can grab sensitive information displayed by other clients
- Clients can fake input and control other clients. (What if they are running as root?)

Case Study 2: Screen recording/sharing

How this is done on X:

- Implemented as a regular client
- Any client can grab any other clients contents or the entire front buffer
- Any client can send ”input” to any other client

This raises huge security concerns:

- Clients can grab sensitive information displayed by other clients
- Clients can fake input and control other clients. (What if they are running as root?)

How this is works in Wayland:

- Clients are isolated and no regular client can grab the screen
- Screen capturing is done by the compositor
- The result may be processed by the compositor directly or handed to a special trusted client

Weston has support for some of this:

- Screen recording via a trusted weston-screenshooter client
- Screen recording to h.264 via libva

All this is Weston and format specific

How this is works in Wayland:

- Clients are isolated and no regular client can grab the screen
- Screen capturing is done by the compositor
- The result may be processed by the compositor directly or handed to a special trusted client

Weston has support for some of this:

- Screen recording via a trusted weston-screenshooter client
- Screen recording to h.264 via libva

All this is Weston and format specific

How this is works in Wayland:

- Clients are isolated and no regular client can grab the screen
- Screen capturing is done by the compositor
- The result may be processed by the compositor directly or handed to a special trusted client

Weston has support for some of this:

- Screen recording via a trusted weston-screenshooter client
- Screen recording to h.264 via libva

All this is Weston and format specific

How this is works in Wayland:

- Clients are isolated and no regular client can grab the screen
- Screen capturing is done by the compositor
- The result may be processed by the compositor directly or handed to a special trusted client

Weston has support for some of this:

- Screen recording via a trusted weston-screenshooter client
- Screen recording to h.264 via libva

All this is Weston and format specific

How do we do this in Wayland?

Someone suggested writing trusted protocol:

- Only accessible by trusted clients
- Allow a client to capture the screen and provide input
- Compositors implement one protocol
- Many people can write sharing/recording clients
- Everyone's happy?

This is a lot of protocol!

How do we do this in Wayland?

Someone suggested writing trusted protocol:

- Only accessible by trusted clients
- Allow a client to capture the screen and provide input
- Compositors implement one protocol
- Many people can write sharing/recording clients
- Everyone's happy?

This is a lot of protocol!

How do we do this in Wayland?

Someone suggested writing trusted protocol:

- Only accessible by trusted clients
- Allow a client to capture the screen and provide input
- Compositors implement one protocol
- Many people can write sharing/recording clients
- Everyone's happy?

This is a lot of protocol!

How do we do this in Wayland?

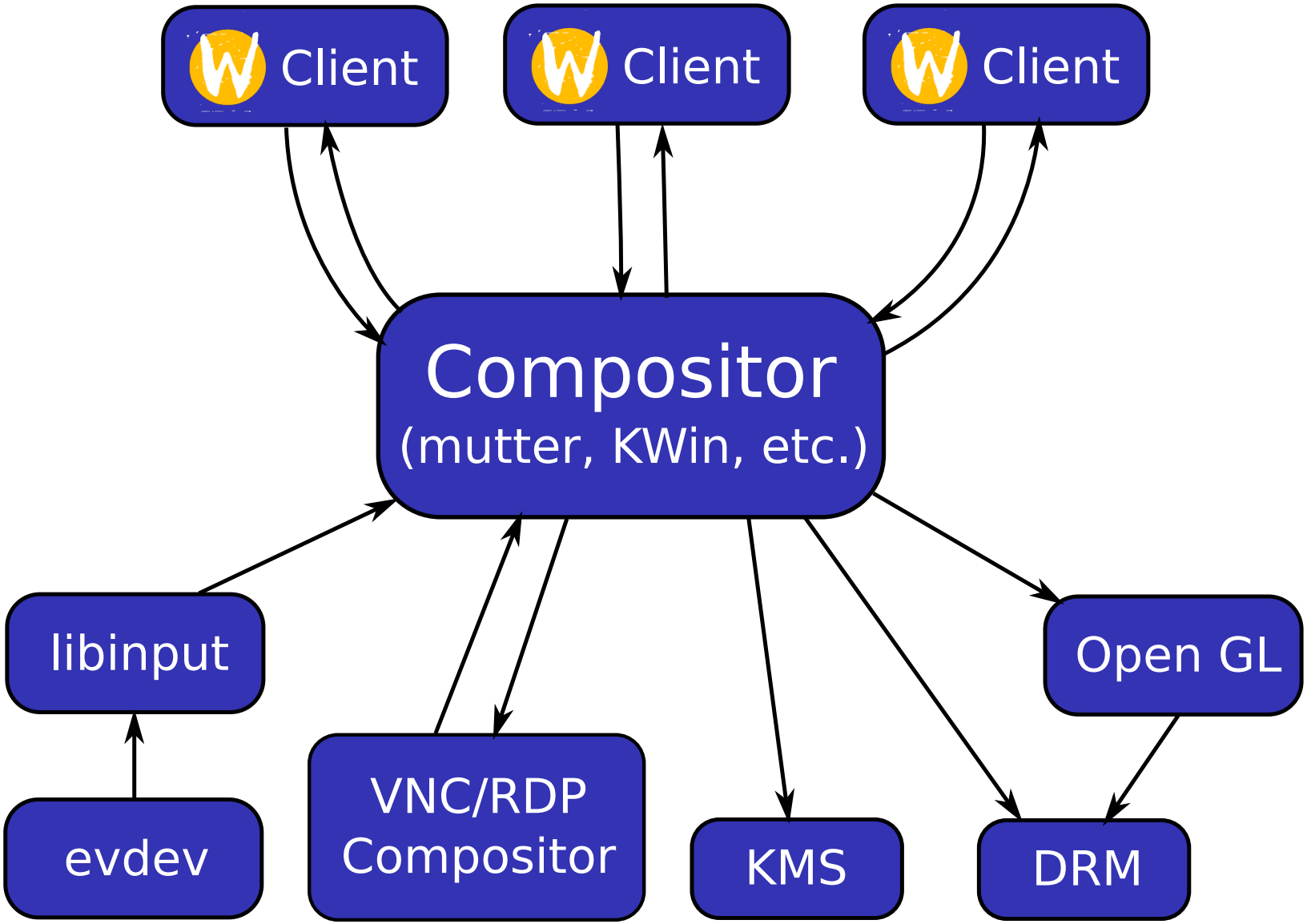
Suggestion: Use the Wayland protocol

- VNC/RDP servers, etc. implement a Wayland compositor
- Compositors launch the subsidiary compositor and connect as a Wayland client
- The compositor is still in control of who gets the screen contents
- The protocol is already written!

How do we do this in Wayland?

Suggestion: Use the Wayland protocol

- VNC/RDP servers, etc. implement a Wayland compositor
- Compositors launch the subsidiary compositor and connect as a Wayland client
- The compositor is still in control of who gets the screen contents
- The protocol is already written!


Client
Client
Client
Compositor
(mutter, KWin, etc.)
libinput
evdev
VNC/RDP
Compositor
KMS
DRM
Open GL

How do we do this in Wayland?

Enter: wl_fullscreen_shell

- present_surface:
  Presents a surface a single surface on an output with a possible scaling mode.
- present_surface_for_mode:
  Provides similar functionality but gives the client control over modesetting

How do we do this in Wayland?

Enter: wl_fullscreen_shell

- present_surface:
  Presents a surface a single surface on an output with a possible scaling mode.
- present_surface_for_mode:
  Provides similar functionality but gives the client control over modesetting

How do we do this in Wayland?

Enter: wl_fullscreen_shell

- present_surface:
  Presents a surface a single surface on an output with a possible scaling mode.
- present_surface_for_mode:
  Provides similar functionality but gives the client control over modesetting

Demo!

Screen recording/sharing is just the beginning!

- Easier to write compositors:
  - Write as a Wayland client and let Weston handle input/output
- Userspace Miracast
- Modesetting when kernel modules aren't an option

Note: This is **not** a replacement for DRM/KMS!

Screen recording/sharing is just the beginning!

- Easier to write compositors:
  - Write as a Wayland client and let Weston handle input/output
- Userspace Miracast
- Modesetting when kernel modules aren't an option

Note: This is **not** a replacement for DRM/KMS!

Screen recording/sharing is just the beginning!

- Easier to write compositors:
  - Write as a Wayland client and let Weston handle input/output
- Userspace Miracast
- Modesetting when kernel modules aren't an option

Note: This is **not** a replacement for DRM/KMS!

Screen recording/sharing is just the beginning!

- Easier to write compositors:
  - Write as a Wayland client and let Weston handle input/output
- Userspace Miracast
- Modesetting when kernel modules aren't an option

Note: This is **not** a replacement for DRM/KMS!

Questions?