The MirAL Story

Alan Griffiths

alan@octopull.co.uk

An Experience Report

- A story
- To compare experiences
- To inspire and learn

Technical Debt

- A story about technical debt
- About the debt
- About “repaying” the debt
- A happy ending

Technical Debt

"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise."

— Ward Cunningham, 1992

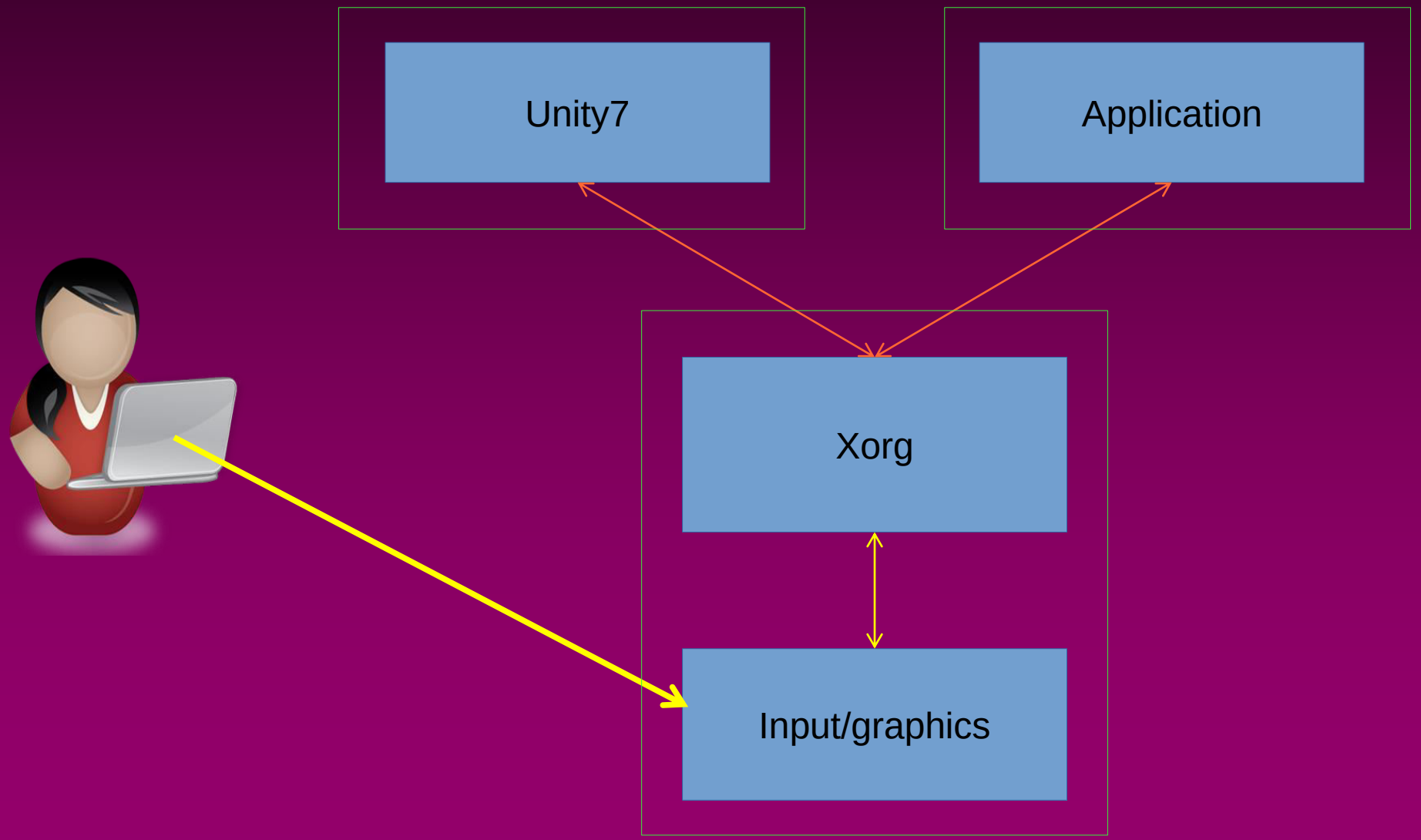
Technical Debt

- A metaphor
- Doesn’t communicate the issue
- Debt is normal for a business
- Bank loan
- Mortgage
- Technical Debt
- Payday loan

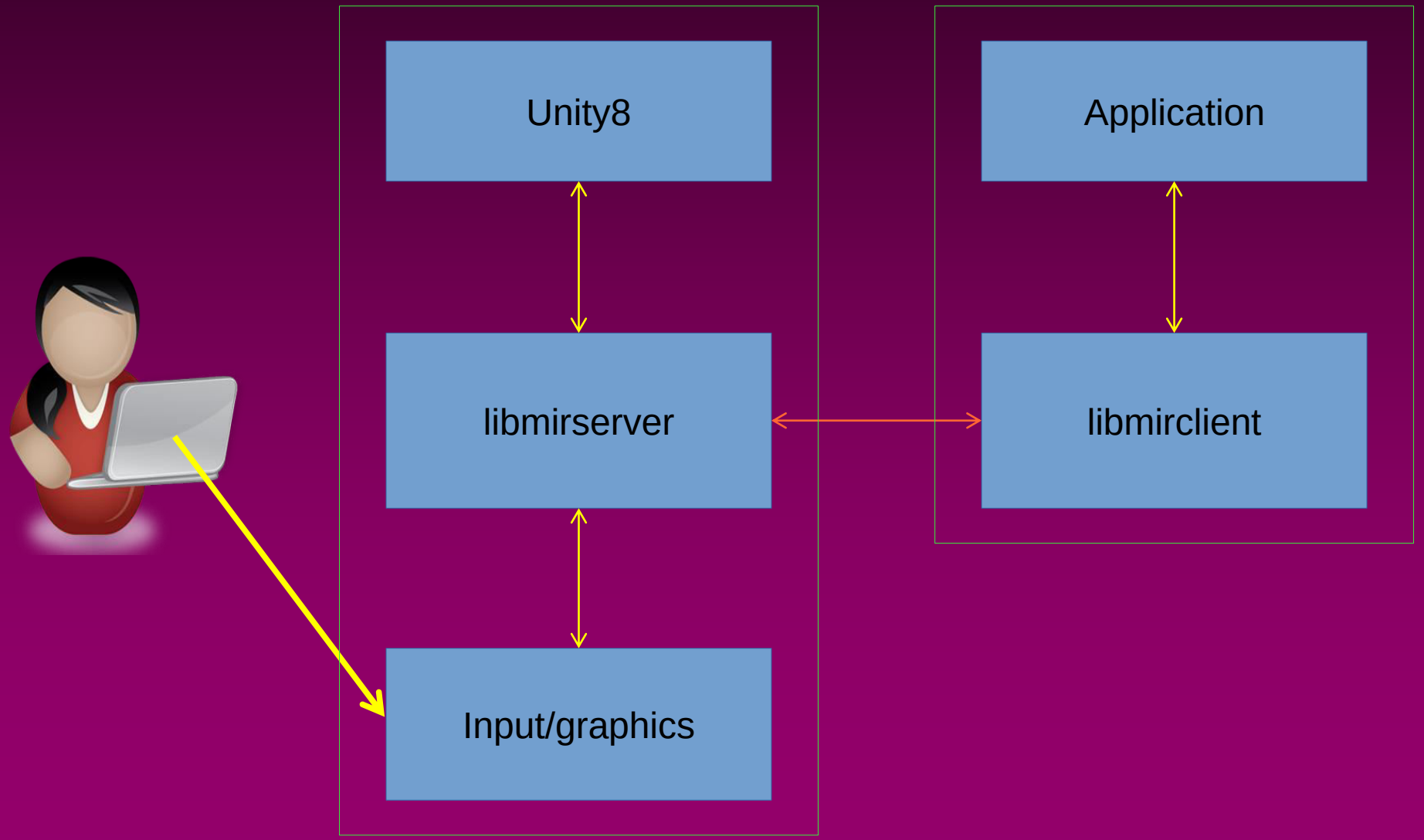
Setting the scene: what is Mir?

- Part of the Linux “graphics stack”
- Supports
  - client
    - “applications” (or “apps”)
  - servers
    - shells
    - desktop environments
    - system compositors

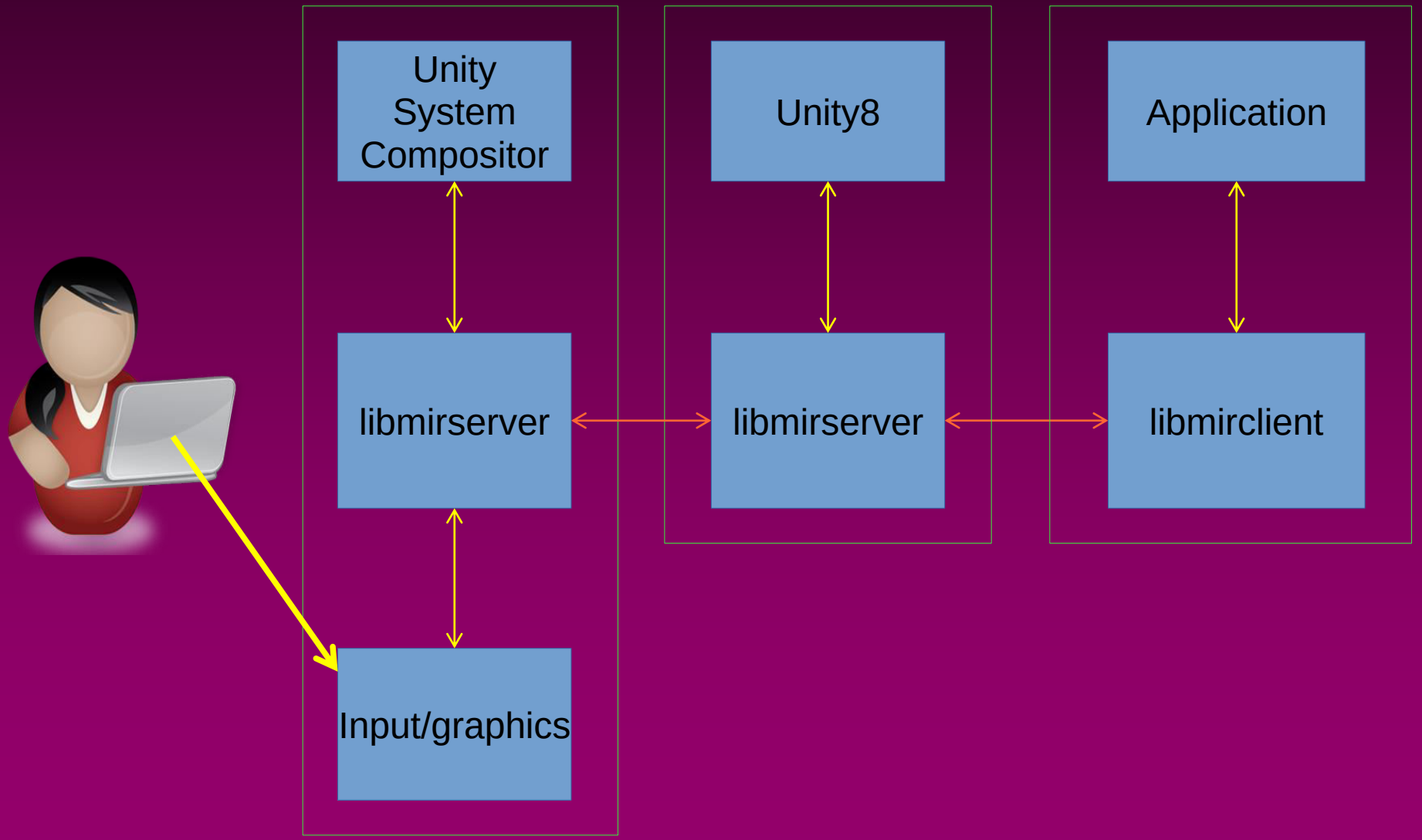
X11 client/server



Mir client/server



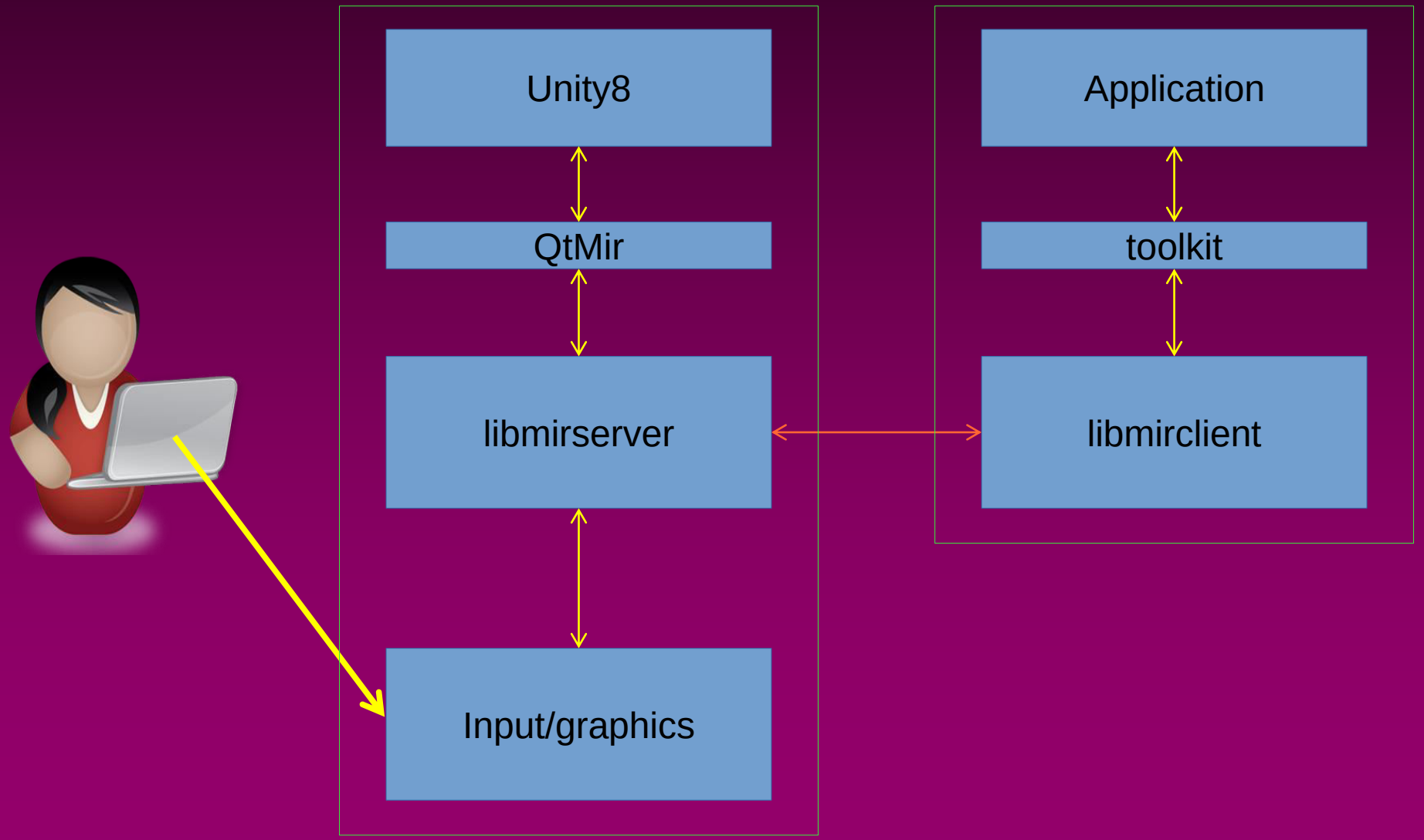
Mir client/server



Setting the scene: Mir servers

- Servers can be:
- Unity8 – on phone, tablet and desktop
- unity-system-compositor – ditto
- ???

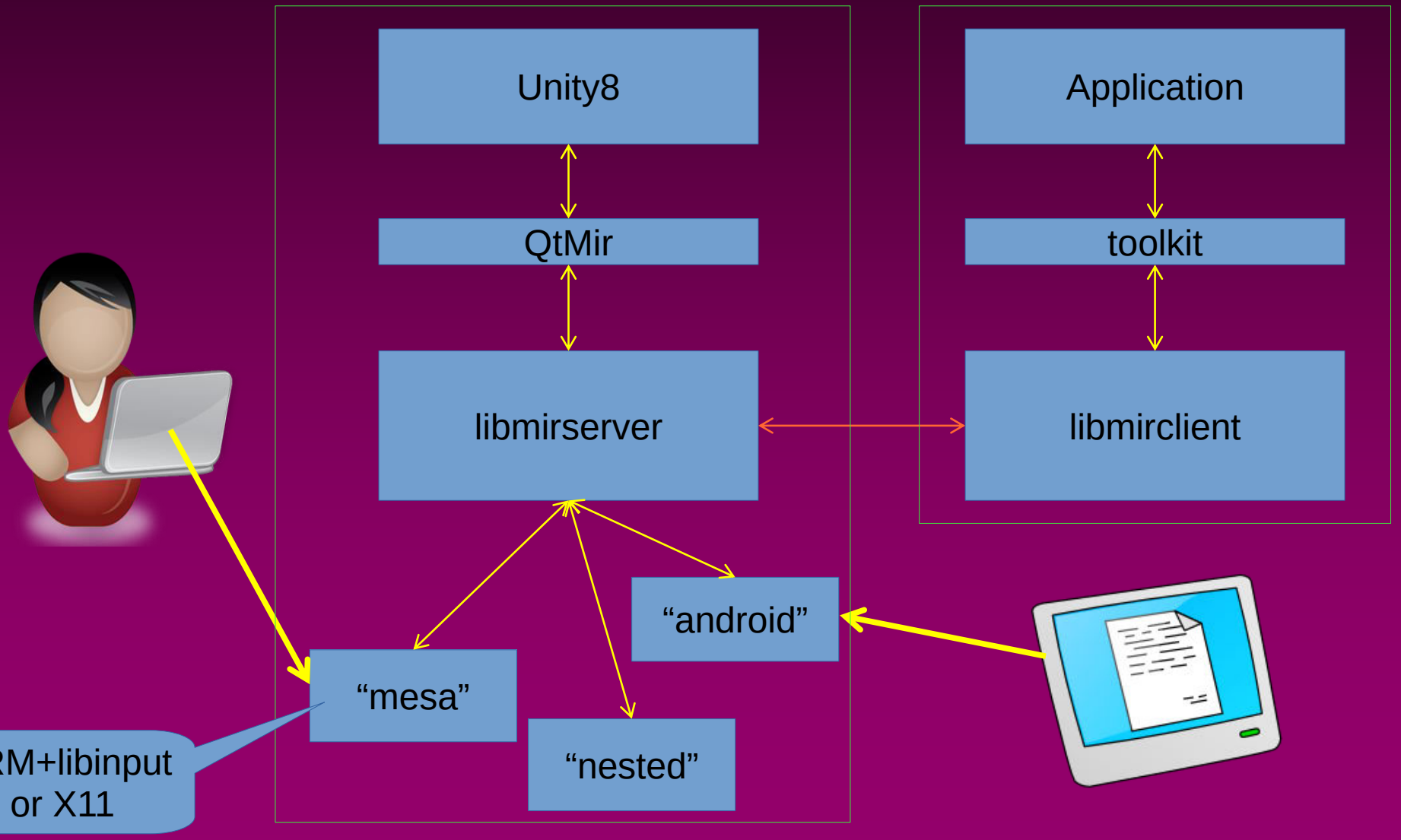
Mir client/server



Setting the scene: Mir platforms

- The graphics platform can be:
- the “Mesa” stack
- Desktop
- Internet of Things
- the “Android” stack
- Phone or tablet
- ???

Mir client/server



Mir servers

- Unity8
  - on tablet/phone
  - on desktop
- unity-system-compositor

- Mir demos
  - mir_demo_server
  - mir_proving_server

Working S/W vs Technical Debt

- We ship working software
- But we incurred some “technical debt”

APIs and ABIs

- API “Application Programming Interface”
- Used by other programmers to write code
- An “API breaking change” means...
- Rewriting the programs

- ABI “Application Binary Interface”
- Used by other programs when they runs
- An “ABI breaking change” means...
- Rebuilding the programs

Mir is a set of libraries

- There are headers (APIs) and libraries (ABIs)
- For client development
- For server development

Mir APIs and ABIs

- For clients:
  - APIs
    - backward compatible
  - ABIs
    - backward compatible
- For servers:
  - APIs
    - sometimes break
  - ABIs
    - usually break

A slide from ACCU 2013

```
class ServerConfiguration
{
public:
    virtual std::shared_ptr<frontend::Communicator>         the_communicator()
= 0;
    virtual std::shared_ptr<shell::SessionStore>            the_session_store()
= 0;
    virtual std::shared_ptr<graphics::Display>
            the_display() = 0;
    virtual std::shared_ptr<compositor::Drawer>             the_drawer() = 0;
    virtual std::shared_ptr<input::InputManager>
            the_input_manager() = 0;

protected:
    ServerConfiguration() = default;
    virtual ~ServerConfiguration() = default;

    ServerConfiguration(ServerConfiguration const&) = delete;
    ServerConfiguration& operator=(ServerConfiguration const&) = delete;
};
```

An ABI problem

```
class DefaultServerConfiguration : public virtual ServerConfiguration
{
public:
…
 virtual std::shared_ptr<Shell> the_shell();
…
 virtual std::shared_ptr<DisplayLayout> the_shell_display_layout();
```

- A flexible way to the configure system
- Almost every change caused an ABI break
- The vtable layout matters

The Mir server API and ABI

- libmirserver-dev rapidly evolving API
- Not designed for ABI stability
- Every release had ABI breaking changes
- When ABI breaks
- Downstream projects need rebuilding
- When API breaks
- Downstream projects need reworking

The cost: “interest payments”

- Releasing Mir means
- Updating downstreams
- QtMir & Unity8
- Unity System Compositor
- A silo containing downstreams
- Automated tests of downstreams
- Manual tests of full stack
- And triaging errors from multiple projects

Bad, but not bad enough?

- Releasing Mir is expensive
- Can’t just release Mir code when ready
- Downstream projects need updates
- Reviews and testing
- Test failures are not isolated to Mir
- It takes man-days effort and weeks elapsed

For Mir server projects

- Each and Every Mir release
- Need to be rebuilt and retested
- And probably updated
- Only possible if “owned” by Canonical

Mir clients

- Client applications can be:
- “Native” – using the Mir client API directly
- GTK3 – GDK has a “Mir backend”
- Qt – Qt has a “Mir backend”
- SDL – SDL has a “Mir backend”
- Kodi – Kodi has a “Mir backend”
- X11 – using Xmir

Bad, but not bad enough?

- The server projects are Canonical's
- So we can change them "easily"
- There are "only a few"

- Client toolkits are not Canonical's
- So we can't change them "easily"
- There are a lot of client toolkits
- We don't break the client ABI or API

Debt reduction

- The developers have tried to reduce the cost
- By “unpublishing” unused APIs
- By replacing unstable ABIs in libmirserver
- Write a new API & deprecate old
- Update downstream
- Delete (or “unpublish”) the old API

A more stable ABI

```
class DefaultServerConfiguration : public virtual ServerConfiguration
{
public:
…
 virtual std::shared_ptr<Shell> the_shell();
…
 virtual std::shared_ptr<DisplayLayout> the_shell_display_layout();
```

```
class Server
{
public:
…
  void wrap_shell(Wrapper<Shell> const& wrapper);
…
  void wrap_display_configuration_policy(
        Wrapper<DisplayConfigurationPolicy> const& wrapper);
```

Other Priorities

- We've tried to reduce the cost but...
- We deliver new functionality
- We improve performance
- We support new hardware
- We fix bugs
- The result:
- As fast as we improved things
- other issues come along

“Elevating a system from chaos to order takes energy and conscious effort.”

— Thomas Voss, 2016

“Friday Labs”

- Canonical allows ½ day for approved “side projects”
- So I made the case for tackling the Mir ABI
- Management were sceptical

“Friday Labs”

- Canonical allows ½ day for approved “side projects”
- So I made the case for tackling the Mir ABI
- Management were willing to let me try
- I started a “Mir Abstraction Layer” project

MirAL: Mir Abstraction Layer

- A separate project to “abstract” the Mir API
- Design the API with ABI stability in mind
- Narrow focus on Window Management

Window Management

- What is a “shell”?
- a.k.a. “Desktop Environment”
- KDE, Gnome, Awesome, Cinnamon, LXDE, …
- Controls
- Where application windows appear
- What window states (“fullscreen”, “restored”, …) mean
- Switching applications
- Other “chrome” (launchers, status, notifications)

MirAL: `init()`

- I copied “example servers” from Mir
- And set up a new project to work in
- And immediately found packaging bugs in Mir
- Headers referenced, but not published
- Dependencies missed in pkg-config
- A “test framework” that wouldn’t link

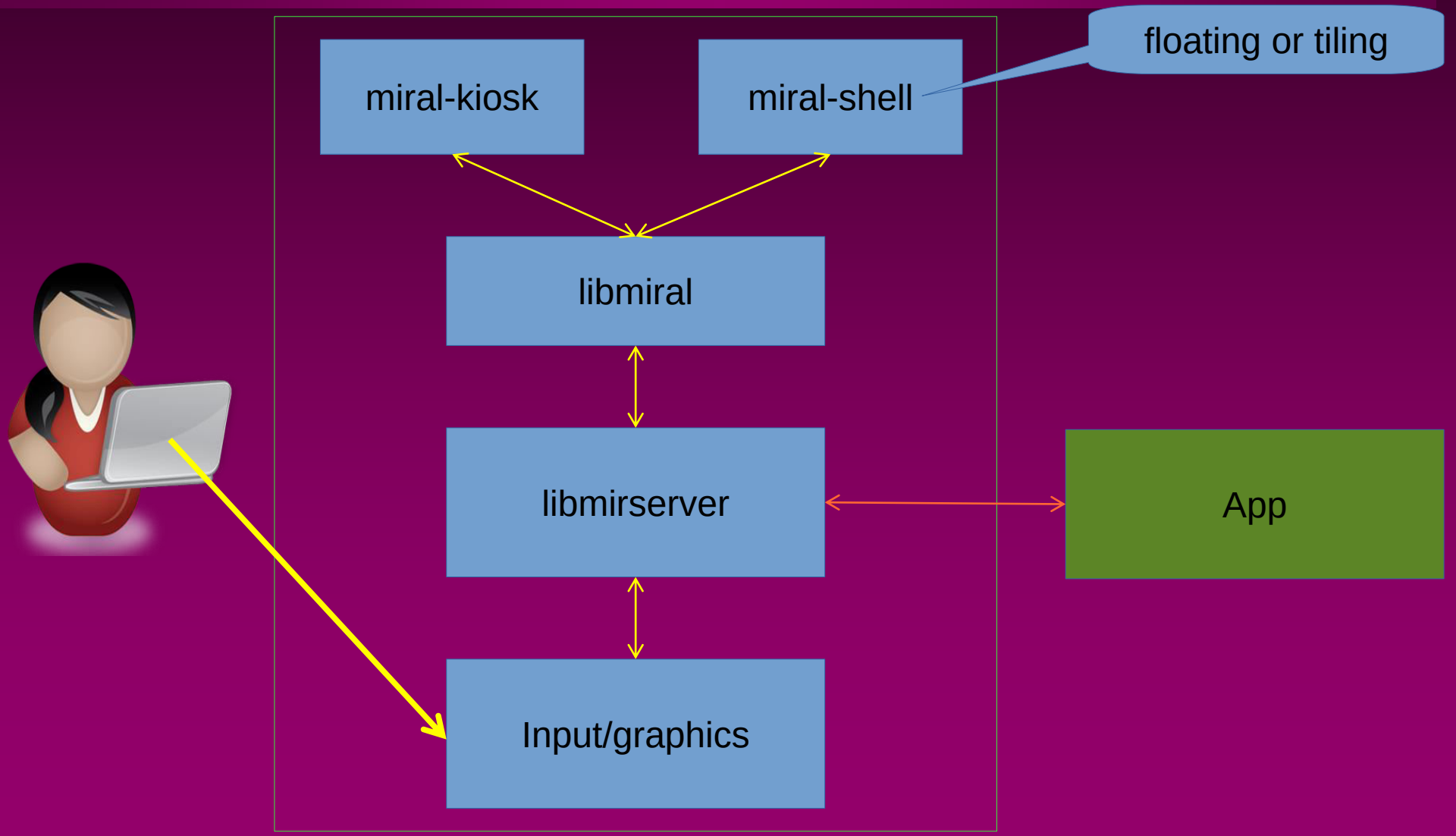
MirAL: `init()`

- I filed bugs against Mir
- ...and fixed them in my “day job”

MirAL: `init()`

- I started refactoring the example code
- Focussed on Window Management
- We had three styles
  - “floating”
  - “tiling”
  - “fullscreen”
- Extracted commonality
- Mined abstractions

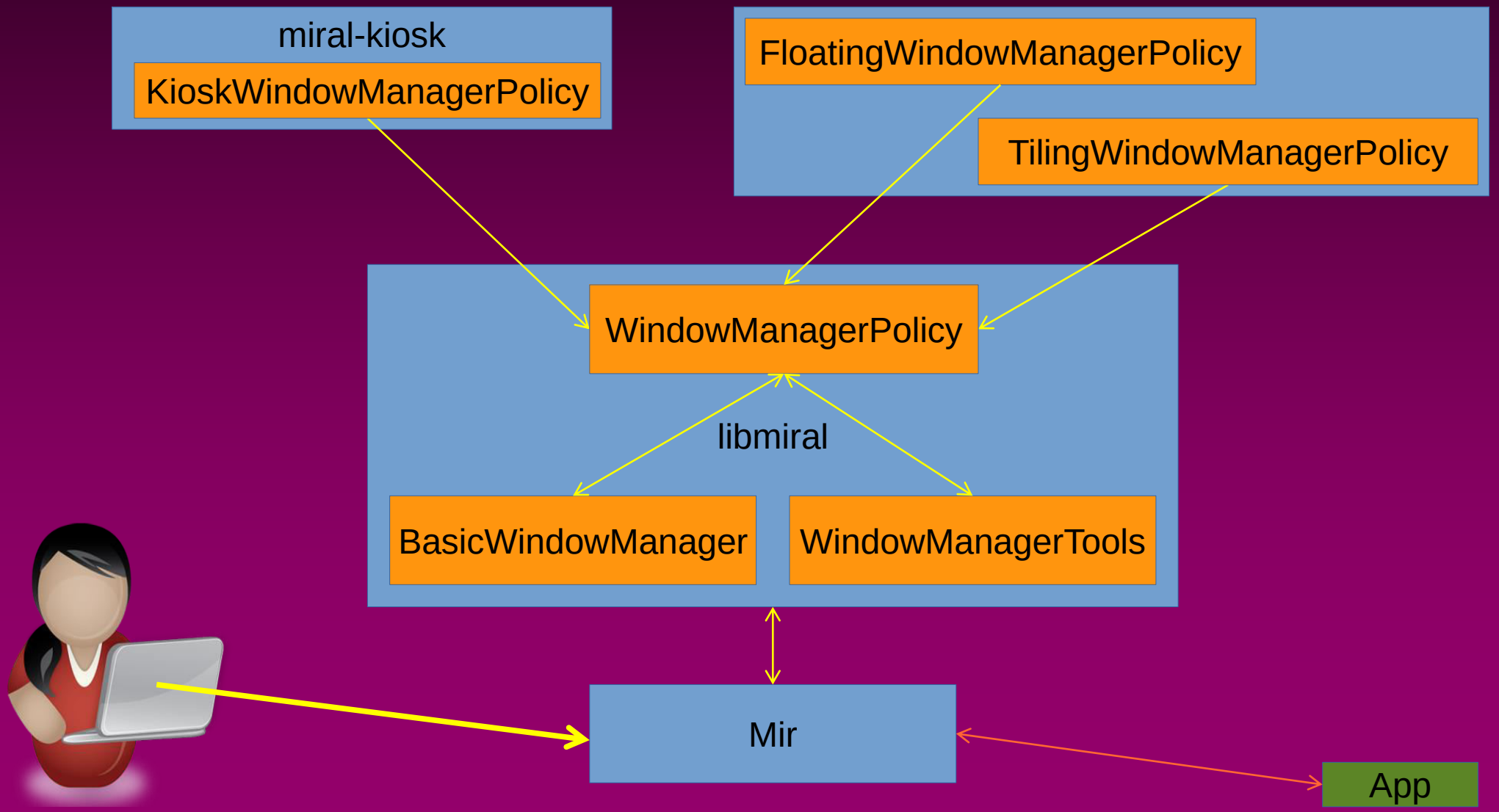
MirAL


floating or tiling
miral-kiosk
miral-shell
libmiral
libmirserver
App
Input/graphics

MirAL: The Main Abstractions

- There were three principle roles
- A “basic window manager”
- Generic behaviours and defaults
- A “window management policy”
- Used by the manager
- The customization point
- And “window management tools”
- Used by the policy

MirAL


miral-kiosk
KioskWindowManagerPolicy
FloatingWindowManagerPolicy
TilingWindowManagerPolicy
WindowManagerPolicy
libmiral
BasicWindowManager
WindowManagerTools
Mir
App

A look at code using MirAL

- The `main()` function
- A “policy” class
- A bit of policy implementation

main() at ACCU 2016

```
class CanonicalWindowManagerPolicy  : public miral::WindowManagementPolicy
```

```
int main(int argc, char const* argv[])
{
  using namespace miral;
  SpinnerSplash spinner;
  return MirRunner{argc, argv}.run_with(
    {
      WindowManagerOptions
        {
          add_window_manager_policy<CanonicalWindowManagerPolicy>("canonical", spinner),
          add_window_manager_policy<TilingWindowManagerPolicy>("tiling"),
        },
      display_configuration_options,
      QuitOnCtrlAltBkSp{},
      StartupInternalClient{"Intro", spinner}
    });
}
```

main() at ACCU 2017

```
int main(int argc, char const* argv[])
{
    using namespace miral;
    std::function<void()> shutdown_hook{[]{}};
    SpinnerSplash spinner;
    InternalClientLauncher launcher;
    ActiveOutputsMonitor outputs_monitor;
    WindowManagerOptions window_managers
        {
            add_window_manager_policy<TitlebarWindowManagerPolicy>("titlebar", spinner, launcher, shutdown_hook),
            add_window_manager_policy<TilingWindowManagerPolicy>("tiling", spinner, launcher, outputs_monitor),
        };
    MirRunner runner{argc, argv};
    runner.add_stop_callback([&] { shutdown_hook(); });
    auto const quit_on_ctrl_alt_bksp = [&](MirEvent const* event)
        {
            if (mir_event_get_type(event) != mir_event_type_input)
                return false;
            MirInputEvent const* input_event = mir_event_get_input_event(event);
            if (mir_input_event_get_type(input_event) != mir_input_event_type_key)
                return false;
            MirKeyboardEvent const* kev = mir_input_event_get_keyboard_event(input_event);
            if (mir_keyboard_event_action(kev) != mir_keyboard_action_down)
                return false;
            MirInputEventModifiers mods = mir_keyboard_event_modifiers(kev);
            if (!(mods & mir_input_event_modifier_alt) || !(mods & mir_input_event_modifier_ctrl))
                return false;
            if (mir_keyboard_event_scan_code(kev) != KEY_BACKSPACE)
                return false;
            runner.stop();
            return true;
        };
    Keymap config_keymap;
    DebugExtension debug_extensions;
    return runner.run_with(
        {
            CommandLineOption{[&](std::string const& ) { },
                              "desktop_file_hint", "Ignored for Unity8 compatibility", "miral-shell.desktop"},
            CursorTheme{"default"},
            window_managers,
            display_configuration_options,
            launcher,
            outputs_monitor,
            config_keymap,
            debug_extensions,
            AppendEventFilter{quit_on_ctrl_alt_bksp},
            StartupInternalClient{"Intro", spinner},
            CommandLineOption{[&](std::string const& typeface) { ::titlebar::font_file(typeface); },
                              "shell-titlebar-font", "font file to use for titlebars", ::titlebar::font_file()}
        });
}
```

main() at ACCU 2017

```
int main(int argc, char const* argv[])
{
    using namespace miral;
    std::function<void()> shutdown_hook{[]{}};
    SpinnerSplash spinner;
    InternalClientLauncher launcher;
    ActiveOutputsMonitor outputs_monitor;
    WindowManagerOptions window_managers
        {
            add_window_manager_policy<TitlebarWindowManagerPolicy>(
                "titlebar", spinner, launcher, shutdown_hook),
            add_window_manager_policy<TilingWindowManagerPolicy>(
                "tiling", spinner, launcher, outputs_monitor),
        };
    MirRunner runner{argc, argv};
    runner.add_stop_callback([&] { shutdown_hook(); });
```

main() at ACCU 2017

```
    auto const quit_on_ctrl_alt_bksp = [&](MirEvent const* event)
        {
            if (mir_event_get_type(event) != mir_event_type_input)
                return false;

            auto const input_event = mir_event_get_input_event(event);
            if (mir_input_event_get_type(input_event) != mir_input_event_type_key)
                return false;

            auto const kev = mir_input_event_get_keyboard_event(input_event);
            if (mir_keyboard_event_action(kev) != mir_keyboard_action_down)
                return false;

            MirInputEventModifiers mods = mir_keyboard_event_modifiers(kev);
            if (!(mods & mir_input_event_modifier_alt) ||
                !(mods & mir_input_event_modifier_ctrl))
                return false;

            if (mir_keyboard_event_scan_code(kev) != KEY_BACKSPACE)
                return false;

            runner.stop();
            return true;
        };
```

main() at ACCU 2017

```
    Keymap config_keymap;
    DebugExtension debug_extensions;
    return runner.run_with(
        {
            CommandLineOption{[&](std::string const& ) { },
                "desktop_file_hint",
                "Ignored for Unity8 compatibility",
                "miral-shell.desktop"},
            CursorTheme{"default"},
            window_managers,
            display_configuration_options,
            launcher,
            outputs_monitor,
            config_keymap,
            debug_extensions,
            AppendEventFilter{quit_on_ctrl_alt_bksp},
            StartupInternalClient{"Intro", spinner},
            CommandLineOption{[&](std::string const& typeface)
                { ::titlebar::font_file(typeface); },
                "shell-titlebar-font",
                "font file to use for titlebars", ::titlebar::font_file()}
        });
}
```

The Kiosk `main()`

```
int main(int argc, char const* argv[])
{
    SwSplash splash;

    CommandLineOption maximise_roots{
        [&](bool maximize_root_windows)
            { KioskWindowManagerPolicy::maximize_root_windows = maximize_root_windows; },
        "kiosk-maximize-root-windows",
        "Force root windows to maximized",
        KioskWindowManagerPolicy::maximize_root_windows};

    CommandLineOption startup_only{
        [&](bool startup_only)
            { KioskAuthorizer::startup_only = startup_only; },
        "kiosk-startup-apps-only",
        "Only allow applications to connect during startup",
        KioskAuthorizer::startup_only};

    return MirRunner{argc, argv}.run_with(
        {
            set_window_management_policy<KioskWindowManagerPolicy>(splash),
            SetApplicationAuthorizer<KioskAuthorizer>{splash},
            Keymap{},
            maximise_roots,
            startup_only,
            StartupInternalClient{"Intro", splash}
        });
}
```

The Kiosk Policy

```
class KioskWindowManagerPolicy : public CanonicalWindowManagerPolicy
{
public:
    KioskWindowManagerPolicy(WindowManagerTools const& tools, SwSplash const&);

    void advise_focus_gained(WindowInfo const& info) override;

    virtual void advise_new_window(WindowInfo const& window_info) override;

    bool handle_keyboard_event(MirKeyboardEvent const* event) override;
    bool handle_touch_event(MirTouchEvent const* event) override;
    bool handle_pointer_event(MirPointerEvent const* event) override;

    static std::atomic<bool> maximize_root_windows;

private:
    SwSplash const splash;
};
```

The Kiosk implementation

```
bool KioskWindowManagerPolicy::handle_touch_event(MirTouchEvent const* event)
{
    auto const count = mir_touch_event_point_count(event);

    long total_x = 0;
    long total_y = 0;

    for (auto i = 0U; i != count; ++i)
    {
        total_x += mir_touch_event_axis_value(event, i, mir_touch_axis_x);
        total_y += mir_touch_event_axis_value(event, i, mir_touch_axis_y);
    }

    Point const cursor{total_x/count, total_y/count};

    tools.select_active_window(tools.window_at(cursor));

    return false;
}
```

Shells based on MirAL

- miral-shell
  - The traditional “floating” example
- miral-shell --window-manager tiling
  - An example of a different WM policy
- miral-kiosk
  - Very basic WM for simple requirements

MirAL: An ABI Stable Design

- Avoiding
  - exposing data layout that might change
  - virtual functions tables that might change
- Using
  - Cheshire Cat (a.k.a. Pimpl) idiom
  - Wrapping Mir types with focussed wrappers

A struct Mir exposes

```
struct SurfaceSpecification
{
    bool is_empty() const;

    optional_value<geometry::Width> width;
    optional_value<geometry::Height> height;
    optional_value<MirPixelFormat> pixel_format;
    optional_value<std::string> name;
…
};
```

A struct Mir exposes

```
struct SurfaceSpecification
{
    bool is_empty() const;

    optional_value<geometry::Width> width;
    optional_value<geometry::Height> height;
    optional_value<MirPixelFormat> pixel_format;
    optional_value<std::string> name;
…
    optional_value<MirShellChrome> shell_chrome;
    optional_value<MirPointerConfinementState> confine_pointer;
    optional_value<std::shared_ptr<graphics::CursorImage>> cursor_image;
    optional_value<StreamCursor> stream_cursor;
};
```

A “struct” MirAL exposes

```
struct WindowInfo
{
    WindowInfo();
    WindowInfo(Window const& window, WindowSpecification const& params);
    ~WindowInfo();
    explicit WindowInfo(WindowInfo const& that);
    WindowInfo& operator=(WindowInfo const& that);

    bool can_be_active() const;
    bool can_morph_to(MirWindowType new_type) const;
...
    auto name() const -> std::string;
    void name(std::string const& name);

    auto type() const -> MirWindowType;
    void type(MirWindowType type);
...
private:
    struct Self;
    std::unique_ptr<Self> self;
};
```

Preserving ABI

- Functions
  - Adding is OK
  - Removing breaks ABI
  - Changing parameter lists breaks ABI
  - Renaming dubious
- Types
  - Adding is OK
  - Removing breaks ABI
  - Changing layout breaks ABI
  - Adding virtual functions dubious
  - Renaming dubious

Renaming functions

```
#include <mir/version.h>

#define MIRAL_FAKE_OLD_SYMBOL(old_sym, new_sym)\
    extern "C" __attribute__((alias(#new_sym))) void old_sym();

#define MIRAL_FAKE_NEW_SYMBOL(old_sym, new_sym)\
    extern "C" __attribute__((alias(#old_sym))) void new_sym();

#if (MIR_SERVER_VERSION >= MIR_VERSION_NUMBER(0, 26, 0))
    #define MIRAL_BOTH_VERSIONS(old_sym, new_sym)\
    MIRAL_FAKE_OLD_SYMBOL(old_sym, new_sym)
#else
    #define MIRAL_BOTH_VERSIONS(old_sym, new_sym)\
    MIRAL_FAKE_NEW_SYMBOL(old_sym, new_sym)
#endif
```

Renaming functions

```
#include <mir/version.h>

#define MIRAL_FAKE_OLD_SYMBOL(old_sym, new_sym)\
    extern "C" __attribute__((alias(#new_sym))) void old_sym();

#define MIRAL_FAKE_NEW_SYMBOL(old_sym, new_sym)\
    extern "C" __attribute__((alias(#old_sym))) void new_sym();

#if (MIR_SERVER_VERSION >= MIR_VERSION_NUMBER(0, 26, 0))
    #define MIRAL_BOTH_VERSIONS(old_sym, new_sym)\
    MIRAL_FAKE_OLD_SYMBOL(old_sym, new_sym)
```

```
MIRAL_BOTH_VERSIONS(
    _ZNK5miral10WindowInfo12can_morph_toE14MirSurfaceType,
    _ZNK5miral10WindowInfo12can_morph_toE13MirWindowType)
bool miral::WindowInfo::can_morph_to(MirWindowType new_type) const
...
```

Maintaining layout

```
class Keymap
{
public:
    Keymap();

    /// Specify a keymap.
    explicit Keymap(std::string const& keymap);

    /// Specify a new keymap.
    void set_keymap(std::string const& keymap);

    ~Keymap();
    Keymap(Keymap const& that);
    auto operator=(Keymap const& rhs) -> Keymap&;
    void operator()(mir::Server& server) const;

private:
    struct Self;
    std::shared_ptr<Self> self;
};
```

Adding virtual functions

```
class WindowManagementPolicy
{
public:
    /// before any related calls begin
    virtual void advise_begin();

    /// after any related calls end
    virtual void advise_end();

    virtual auto place_new_window(
        ApplicationInfo const& app_info,
        WindowSpecification const& requested_specification) -> WindowSpecification = 0;
    virtual void handle_window_ready(WindowInfo& window_info) = 0;
    virtual void handle_modify_window(WindowInfo& window_info, WindowSpecification const& modifications) = 0;
    virtual void handle_raise_window(WindowInfo& window_info) = 0;
    virtual bool handle_keyboard_event(MirKeyboardEvent const* event) = 0;
    virtual bool handle_touch_event(MirTouchEvent const* event) = 0;
    virtual bool handle_pointer_event(MirPointerEvent const* event) = 0;
    virtual void advise_new_app(ApplicationInfo& application);
    virtual void advise_delete_app(ApplicationInfo const& application);
    virtual void advise_new_window(WindowInfo const& window_info);
    virtual void advise_focus_lost(WindowInfo const& window_info);
    virtual void advise_focus_gained(WindowInfo const& window_info);
    virtual void advise_state_change(WindowInfo const& window_info, MirWindowState state);
    virtual void advise_move_to(WindowInfo const& window_info, Point top_left);
    virtual void advise_resize(WindowInfo const& window_info, Size const& new_size);
    virtual void advise_delete_window(WindowInfo const& window_info);
    virtual void advise_raise(std::vector<Window> const& windows);
    virtual auto confirm_inherited_move(WindowInfo const& window_info, Displacement movement) -> Rectangle = 0;

    virtual ~WindowManagementPolicy() = default;
    WindowManagementPolicy() = default;
    WindowManagementPolicy(WindowManagementPolicy const&) = delete;
    WindowManagementPolicy& operator=(WindowManagementPolicy const&) = delete;
```

Adding virtual functions

```
class WindowManagementPolicy
{
public:
    /// before any related calls begin
    virtual void advise_begin();

    /// after any related calls end
    virtual void advise_end();

    virtual void advise_adding_to_workspace(
        std::shared_ptr<Workspace> const& workspace,
        std::vector<Window> const& windows);

    virtual void advise_removing_from_workspace(
        std::shared_ptr<Workspace> const& workspace,
        std::vector<Window> const& windows);

    virtual void advise_focus_gained(WindowInfo const& window_info);
    virtual void advise_state_change(WindowInfo const& window_info, MirWindowState state);
    virtual void advise_move_to(WindowInfo const& window_info, Point top_left);
    virtual void advise_resize(WindowInfo const& window_info, Size const& new_size);
    virtual void advise_delete_window(WindowInfo const& window_info);
    virtual void advise_raise(std::vector<Window> const& windows);
    virtual auto confirm_inherited_move(WindowInfo const& window_info, Displacement movement) -> Rectangle = 0;

    virtual ~WindowManagementPolicy() = default;
    WindowManagementPolicy() = default;
    WindowManagementPolicy(WindowManagementPolicy const&) = delete;
    WindowManagementPolicy& operator=(WindowManagementPolicy const&) = delete;
```

Adding virtual functions

```
class WindowManagementPolicy
{
public:
    virtual void advise_adding_to_workspace(
        std::shared_ptr<Workspace> const& workspace,
        std::vector<Window> const& windows);

    virtual void advise_removing_from_workspace(
        std::shared_ptr<Workspace> const& workspace,
        std::vector<Window> const& windows);

    virtual ~WorkspacePolicy() = default;
    WorkspacePolicy() = default;
    WorkspacePolicy(WorkspacePolicy const&) = delete;
    WorkspacePolicy& operator=(WorkspacePolicy const&) = delete;
};
    WindowManagementPolicy() = default;
    WindowManagementPolicy(WindowManagementPolicy const&) = delete;
    WindowManagementPolicy& operator=(WindowManagementPolicy const&) = delete;
```

Adding virtual functions

```
class WindowManagementPolicy
{
public:
    virtual ~WindowManagementPolicy() = default;
    WindowManagementPolicy() = default;
    WindowManagementPolicy(WindowManagementPolicy const&) = delete;
    WindowManagementPolicy& operator=(WindowManagementPolicy const&) = delete;
```

```
class WorkspacePolicy
{
public:
    virtual void advise_adding_to_workspace(
        std::shared_ptr<Workspace> const& workspace,
        std::vector<Window> const& windows);

    virtual void
        std::sha
        std::vec

    virtual ~Wor
    WorkspacePol
    WorkspacePolicy(WorkspacePolicy const&) = delete;
    WorkspacePolicy& operator=(WorkspacePolicy const&) = delete;
};
```

```
class TitlebarWindowManagerPolicy :
    public miral::CanonicalWindowManagerPolicy,
    public miral::WorkspacePolicy
{
...
```

Adding virtual functions

```
auto find_workspace_policy(unique_ptr<WindowManagementPolicy> const& policy)
-> WorkspacePolicy*
{
    WorkspacePolicy* result = dynamic_cast<WorkspacePolicy*>(policy.get());

    if (result)
        return result;

    static WorkspacePolicy null_workspace_policy;

    return &null_workspace_policy;
}


...

BasicWindowManager::BasicWindowManager(
    FocusController* focus_controller,
    shared_ptr<DisplayLayout> const& display_layout,
    shared_ptr<PersistentSurfaceStore> const& persistent_surface_store,
    WindowManagementPolicyBuilder const& build) :
...
    policy(build(WindowManagerTools{this})),
    workspace_policy{find_workspace_policy(policy)},
```

```
WindowManagementPolicy& operator=(WindowManagementPolicy const&) = delete;
```

ABI stability

- Checking
  - Semi-automated update of linker script
  - debian/libmiral.symbols
  - abidiff
- Maintaining
  - Discipline
  - Toolchain tricks

Concept Proven

- MirAL
- Has stable ABI
- Should be usable outside Canonical
- Supports writing a “shell” or “Desktop Enviroment”
- Works on desktop, tablet, phone or IOT
- Comes with worked examples

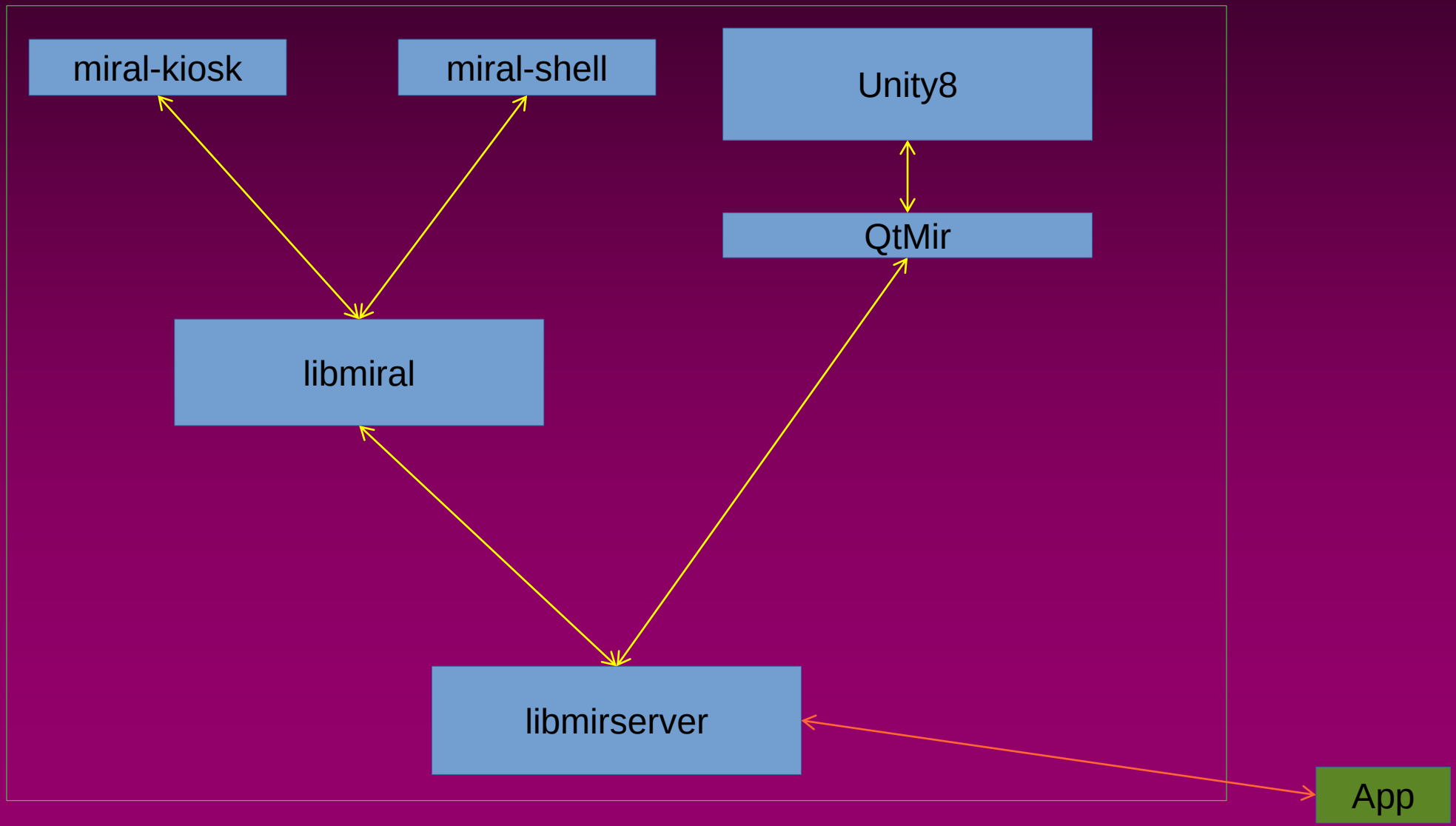
Repurposing MirAL

- “at work” we started thinking about desktop
- Need to consolidate basic window management
- There’s a lot of stuff all shells need
- Unity8 is the “wrong place”
- Should be somewhere useful to all Mir servers
- This was a job for MirAL

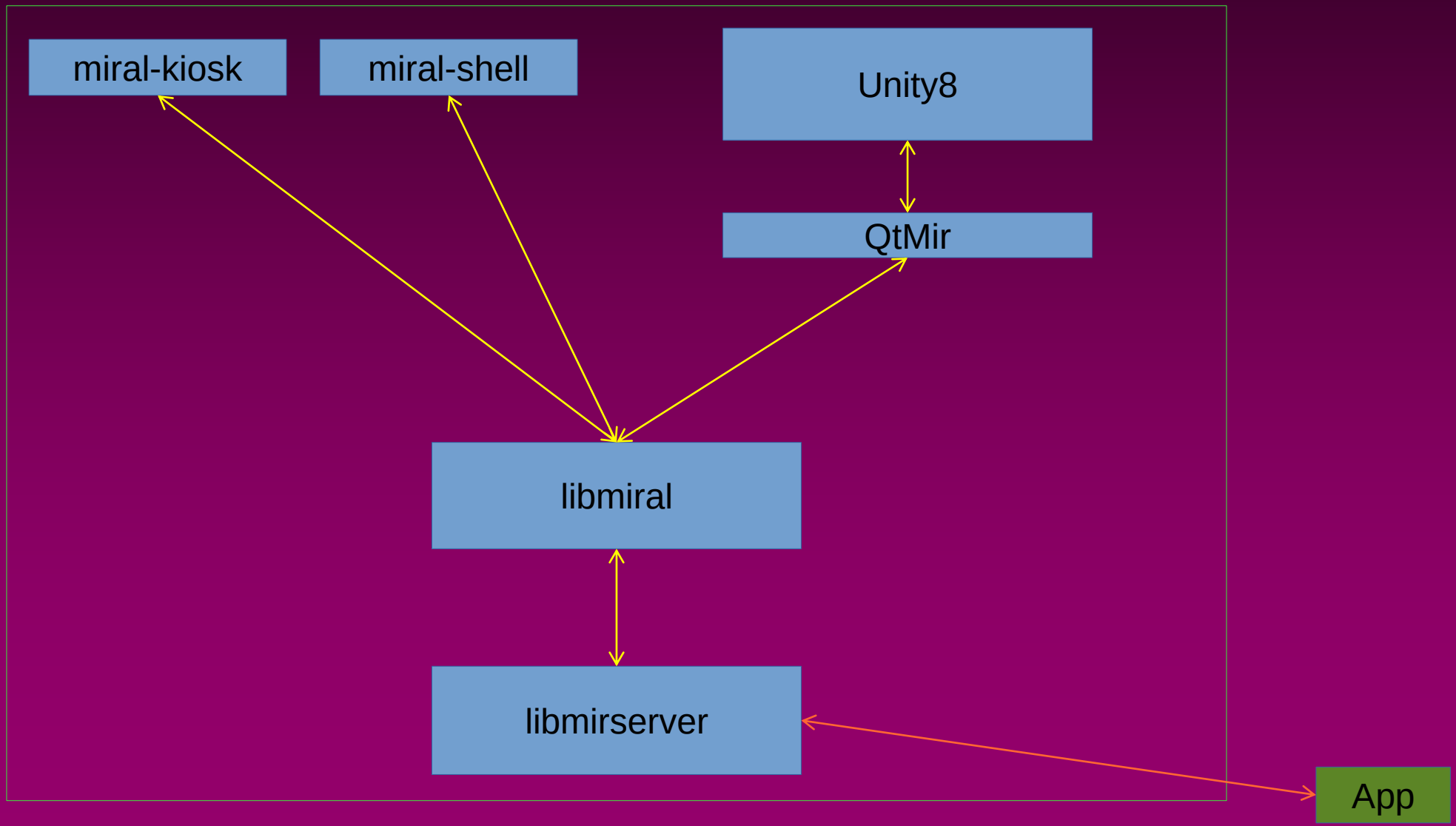
A bit more about QtMir

- QtMir is an adapter between Qt and Mir
- Used by Unity8
- *Might* be usable by other Qt based shells
- We needed to migrate QtMir to use MirAL
- While not stopping QtMir development

Unity8/QtMir and MirAL



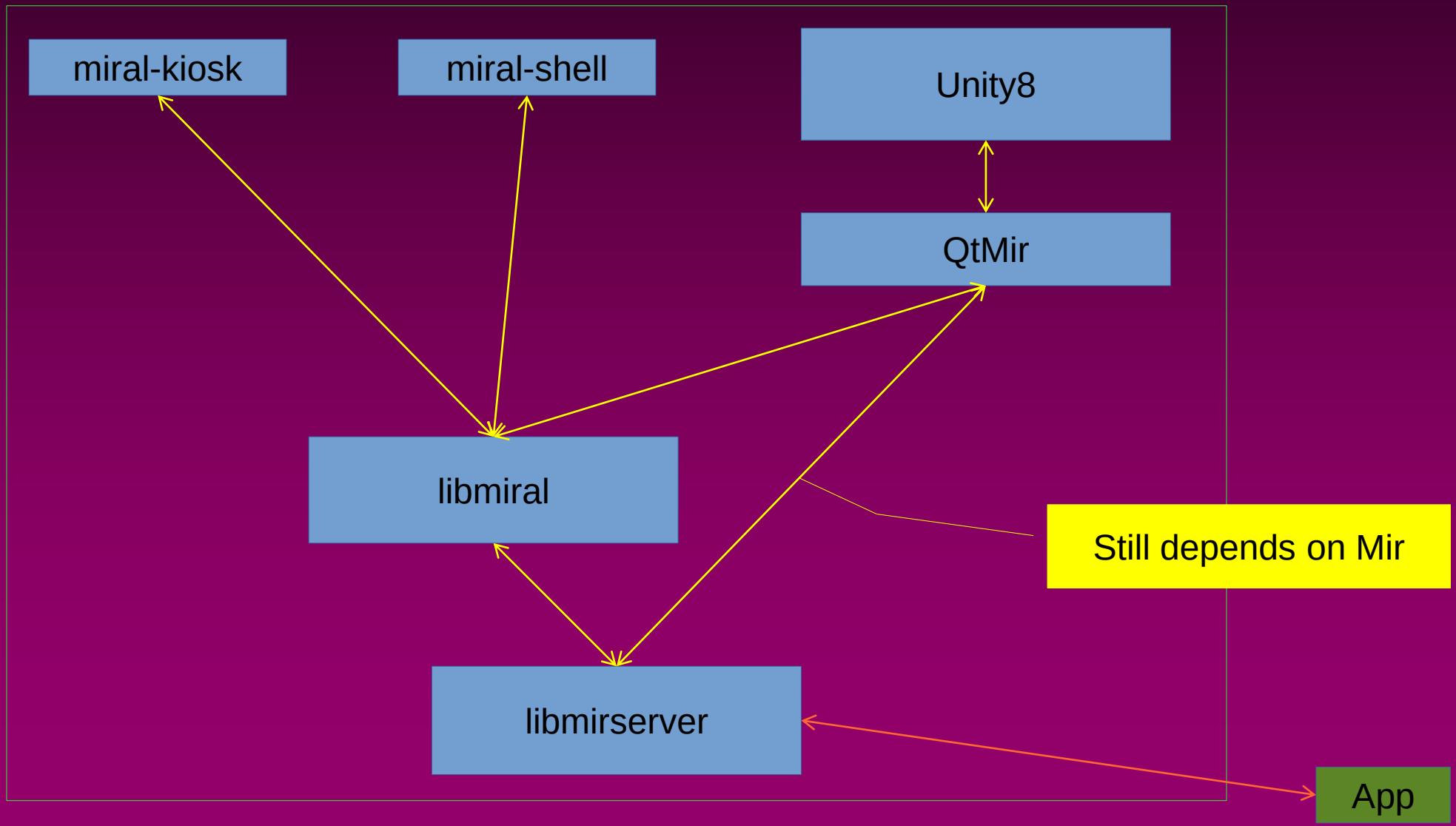
Unity8/QtMir on MirAL (goal)



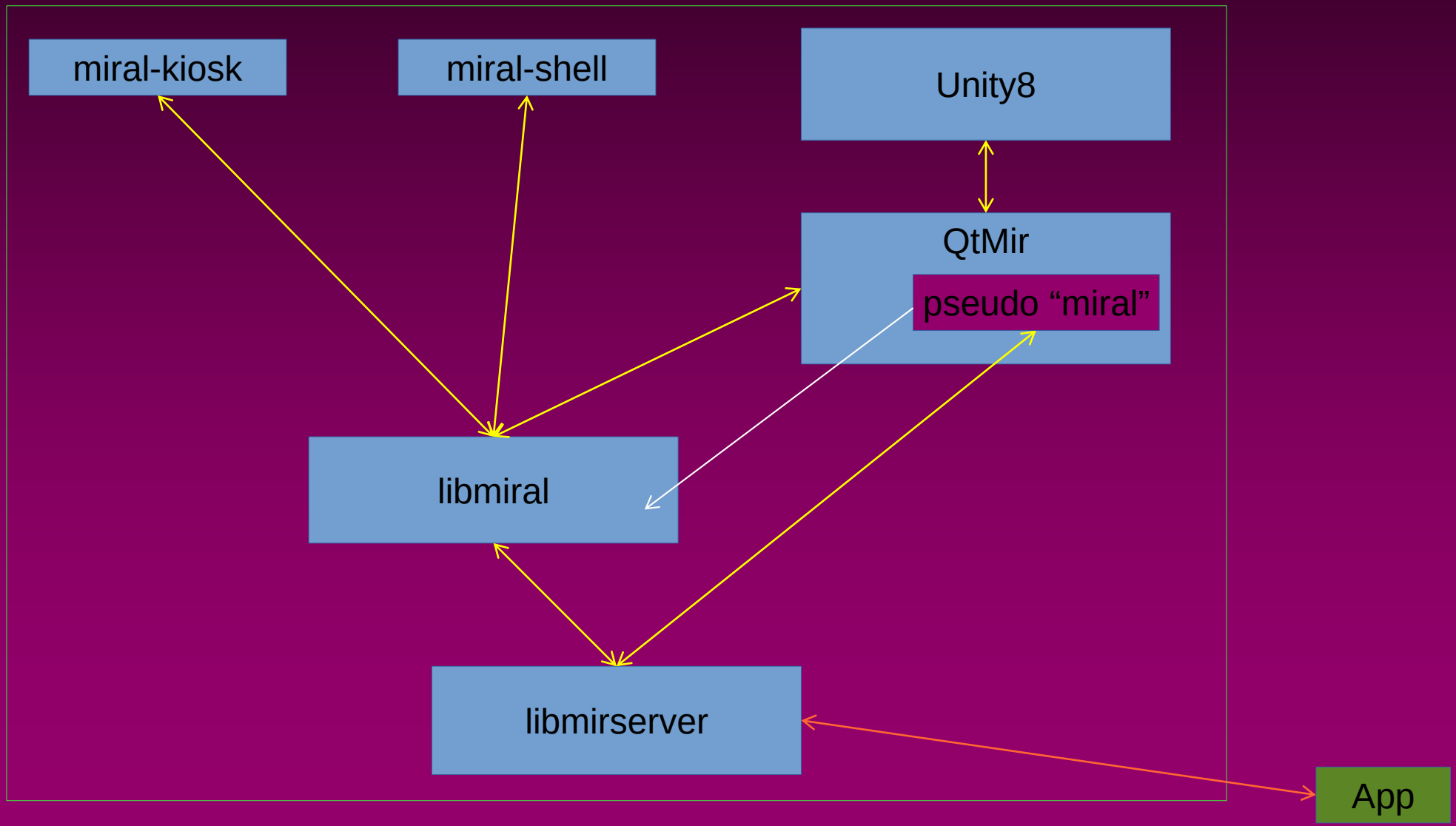
MirAL became my “day job”

- We copied the QtMir source into MirAL
- We set about
  - Removing Mir dependencies and using MirAL
  - Using MirAL window management with Qt
  - Testing and fixing MirAL window management
  - Keeping our QtMir copy in step
- We proved the concept

Unity8/QtMir on MirAL (actual)



Unity8/QtMir on MirAL (work in progress)



Reversal of roles

- Instead of QtMir code in MirAL
- We have “namespace miral” code in QtMir
- Functionality that belongs in libmiral
- But we need experience
- before committing to API & ABI

Shells based on MirAL

- Unity8
  - “convergent” shell for...
    - desktop
    - Phone & tablet
- miral-kiosk
  - Very basic WM for...
    - Dragonboard
    - Raspberry Pi
    - IOT

Shells based on MirAL

- Unity8
  - “convergent” shell for...
  - desktop
  - Phone & tablet
- miral-kiosk
  - Very basic WM for...
  - Dragonboard
  - Raspberry Pi
  - IOT

- miral-shell
  - The canonical example
  - Testing toolkits
- miral-shell --window-manager tiling
  - Demonstrate a different WM policy

Reduced debt

- MirAL releases
- A few hours work
- Within a day elapsed
- No “downstreams” in the silo
- Can deliver features often
- Mir releases only need MirAL in the silo

MirAL and toolkits

- Thinking about desktop also means toolkits
- qtubuntu, gtk-mir, SDL2, …
- These hadn’t been proven against a real Mir server
- MirAL had the necessary support
- This was another job for MirAL

Using MirAL to test window management

```
$ sudo apt install miral-examples
$ miral-app --window-management-trace
$ <toolkit based app>
$ sudo apt install xmir
$ miral-xrun <X11 based app>
```

Using MirAL to develop shells

```
$ sudo apt install libmiral-dev
$ pkg-config --cflags miral
```

- A “shell” or “desktop environment”
- That works on desktop, phone or IOT
- That works with GTK++, Qt and SDL applications
- That works with Xmir

An unintended spin-off

- libmirclientcpp-dev
- A C++ wrapper for the Mir client API
- RAII
- Function chaining

Summary

- We incurred technical debt
- It didn’t solve itself
- But with a bit of imagination we found a way
- Hopefully you found this useful

The MirAL Story

https://launchpad.net/miral

http://voices.canonical.com/tag/miral/

alan@octopull.co.uk