Bohemia Automation

Cross-platform desktop apps in Rust and Qt

Serhij Symonenko / Bohemia Automation

Bohemia Automation

Desktop vs Web in 2023

- No HTTP: industrial devices, databases, clients for 3rd party web services etc.
- Sometimes desktop apps are just faster than web ones, require less RAM
- Sometimes apps require more access to the local machine than WASM can provide
- No limitations - use any crates, use multiple threads, use async runtimes (tokio)
- Create apps with no HTML and CSS coding
- Users must download and install your app – take in account rules of good taste in your sector

Bohemia Automation

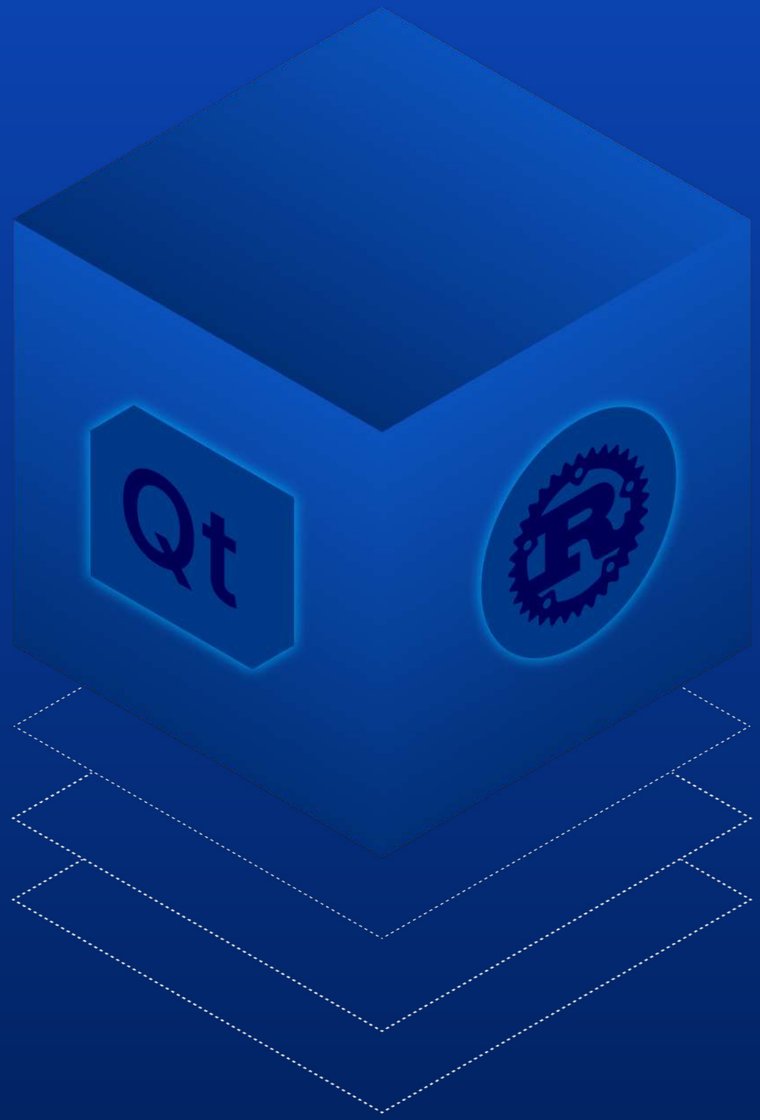
Why Qt?

- Two big grown-up players: Qt and GTK
- Alternatives – Flutter, Slint etc
  are either too young or less focused on desktop apps
- Qt usually looks neater and more Enterprise-ready
  (but take into account the license)
- Qt perfectly works with 4K screens
- Qt correctly works with resizable windows
- Qt has got commercial support, including lots
  of 3rd party consulting companies
- Qt has got Qt Designer and QML
- Qt 5 or 6? Check KDE: https://iskdeusingqt6.org

Bohemia Automation

Qt in Rust

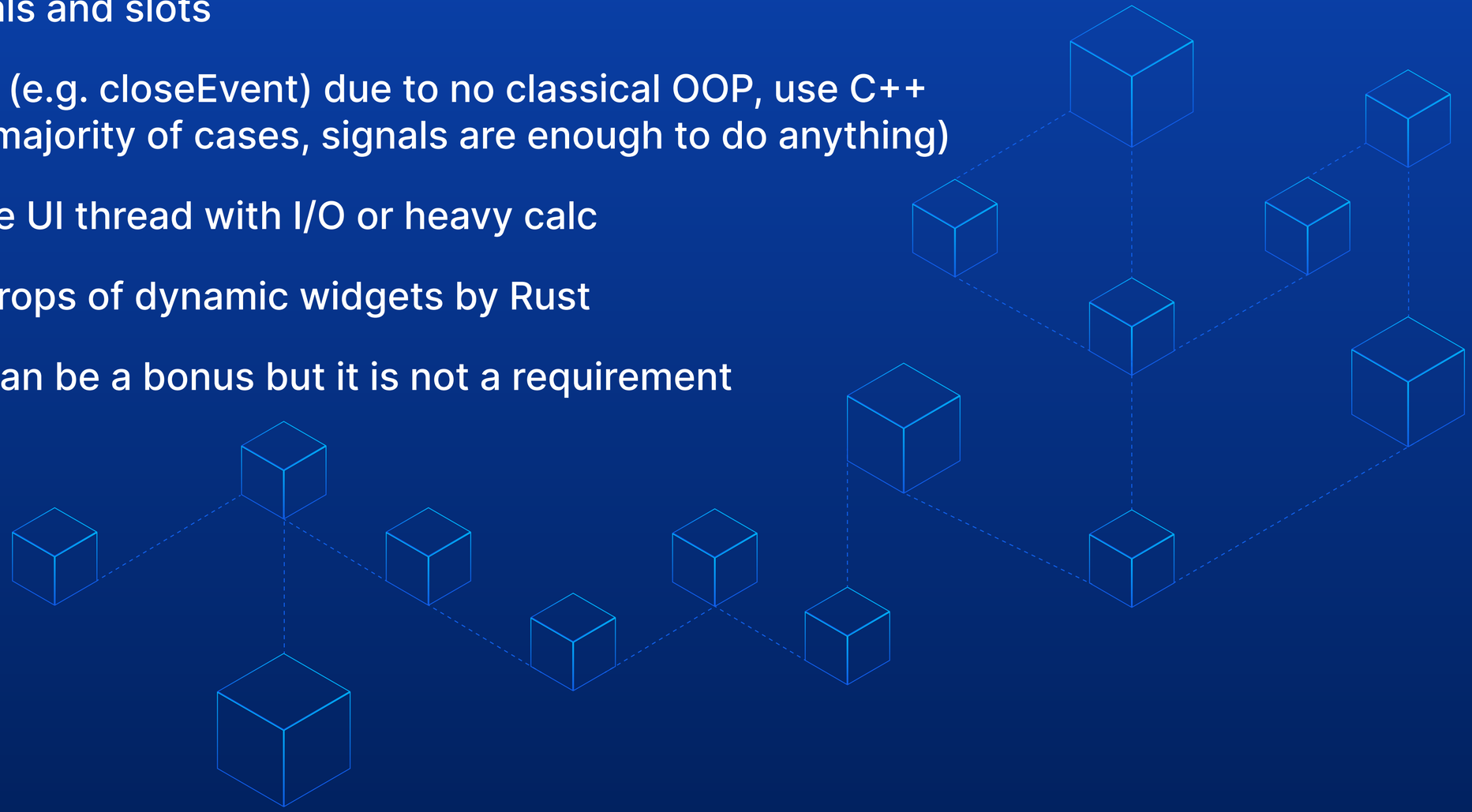
- Ritual
  https://rust-qt.github.io
- qmetaobject
  https://github.com/woboq/qmetaobject-rs
- cxx-qt
  https://www.kdab.com/cxx-qt/



Bohemia Automation

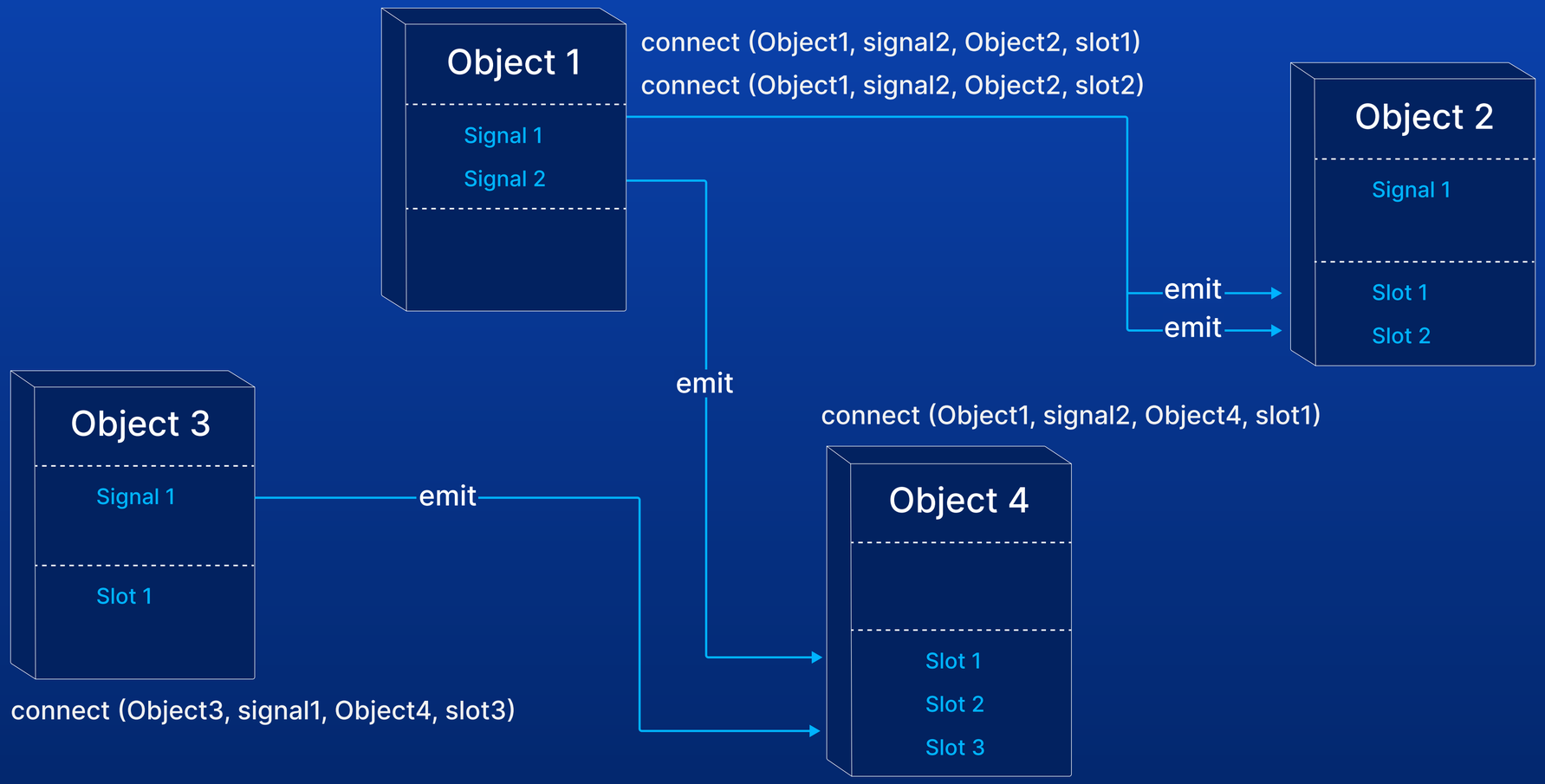
Qt basics for rustaceans

- Learn about signals and slots
- No events in Rust (e.g. closeEvent) due to no classical OOP, use C++ wrappers (in the majority of cases, signals are enough to do anything)
- Avoid blocking the UI thread with I/O or heavy calc
- Beware of auto-drops of dynamic widgets by Rust
- C++ knowledge can be a bonus but it is not a requirement



Bohemia Automation

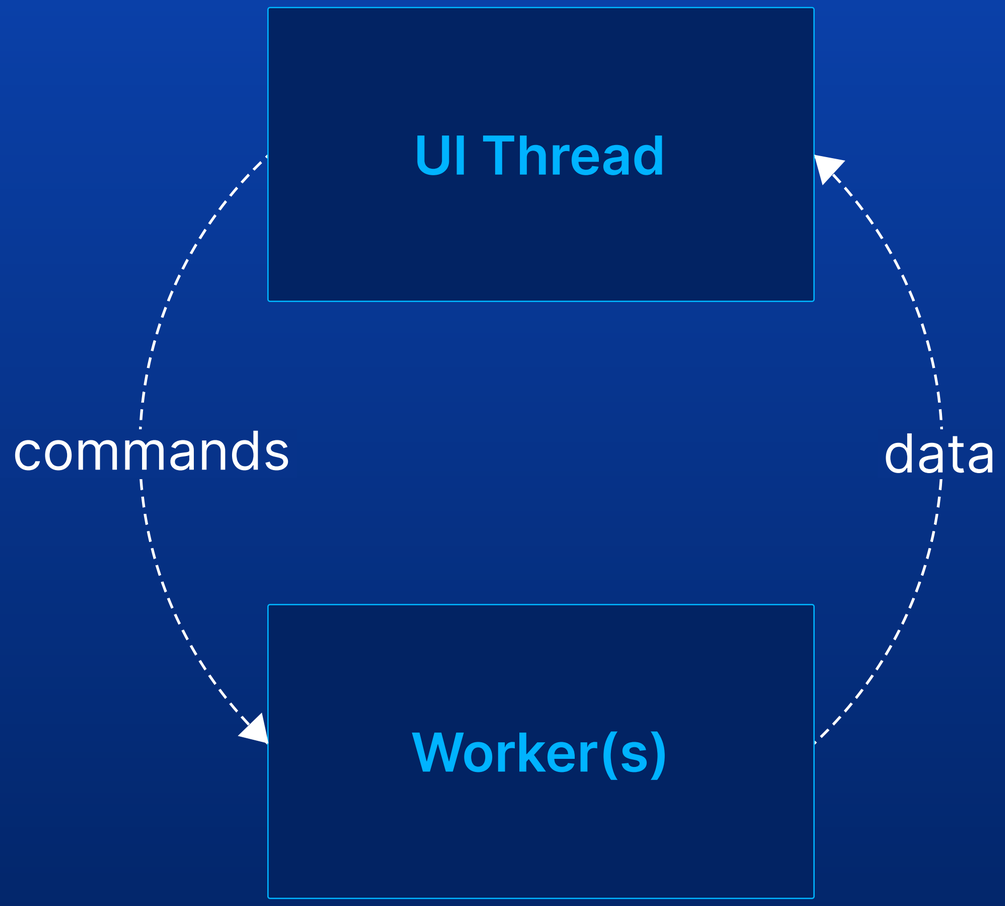
Signals and Slots ecosystem



Note: Signal objects are not completely thread-safe but signal.emit() function is an exception

Bohemia Automation

Organizing data flow



Bohemia Automation

Why do people still use Ritual

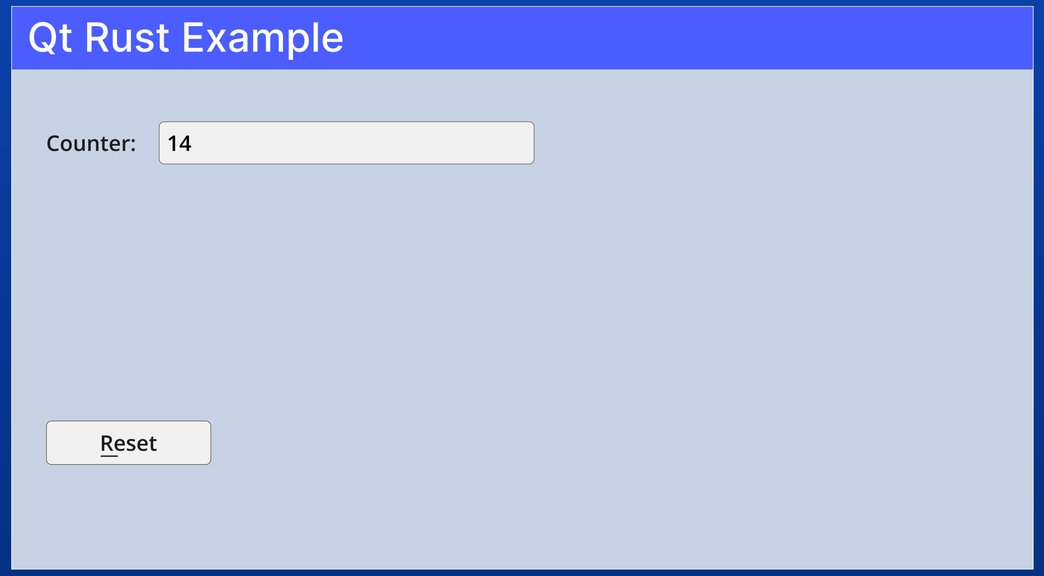
- The oldest and the most popular Rust-Qt project with no serious issues
- The most complete bindings for Qt5
- Supports Qt Designer-generated forms
- Supports dynamic widget creation
- Minimal Rust overhead
  - Qt has got a good documentation and issue discussions. Majority of that can be used with Ritual almost as-is
  - In case of project death and zero alternatives,
    UI logic can be ported to pure C++ with low costs

When Ritual should be avoided

- You prefer to keep away from unsafe blocks in your code
- You use QML only
- You do not care about Qt knowledgebase and prefer a Rust way
- You need Qt6 support

Bohemia Automation

Program example (Ritual)



https://github.com/divi255/qtx

- A background worker increases the counter value every second
- When the value is increased, a data event is being sent to the UI thread
- UI reset button can send reset commands to the background worker to reset the counter
- UI and the worker thread talk via standard mpsc channels
- The worker thread uses Qt Signal with no arguments to notify UI thread about new events in the data channel

Bohemia Automation

The main function (Ritual)

Initializing channels

```rust
fn main() {
    QApplication::init(|_| {
        let (command_tx, command_rx) = std::sync::mpsc::sync_channel::<Command>(64);
        let (data_tx, data_rx) = std::sync::mpsc::sync_channel::<Data>(64);
        let data_signal = UnsafeSend::new(unsafe { SignalNoArgs::new() });
        // construct UI
        let ui = Ui::new(command_tx.clone(), data_rx);
        // connect data signal with UI handle_data slot method
        unsafe {
            data_signal.connect(&ui.slot_handle_data());
            // display the UI (non-blocking)
            ui.window.widget.show();
        }
        // spawn the background worker
        std::thread::spawn(move || {
            worker(command_rx, data_tx, data_signal);
        });
        // execute the Qt application until user exits (blocking)
        let result: i32 = unsafe { QApplication::exec() };
        // optionally terminate the background worker
        command_tx.send(Command::Quit).unwrap();
        result
    })
}
```

https://github.com/divi255/qtx

Bohemia Automation

The main function (Ritual)

Initializing UI

```rust
fn main() {
    QApplication::init(|_| {
        let (command_tx, command_rx) = std::sync::mpsc::sync_channel::<Command>(64);
        let (data_tx, data_rx) = std::sync::mpsc::sync_channel::<Data>(64);
        let data_signal = UnsafeSend::new(unsafe { SignalNoArgs::new() });
        // construct UI
        let ui = Ui::new(command_tx.clone(), data_rx);
        // connect data signal with UI handle_data slot method
        unsafe {
            data_signal.connect(&ui.slot_handle_data());
            // display the UI (non-blocking)
            ui.window.widget.show();
        }
        // spawn the background worker
        std::thread::spawn(move || {
            worker(command_rx, data_tx, data_signal);
        });
        // execute the Qt application until user exits (blocking)
        let result: i32 = unsafe { QApplication::exec() };
        // optionally terminate the background worker
        command_tx.send(Command::Quit).unwrap();
        result
    })
}
```

https://github.com/divi255/qtx

Bohemia Automation

The main function (Ritual)

Spawning Rust workers

```rust
fn main() {
    QApplication::init(|_| {
        let (command_tx, command_rx) = std::sync::mpsc::sync_channel::<Command>(64);
        let (data_tx, data_rx) = std::sync::mpsc::sync_channel::<Data>(64);
        let data_signal = UnsafeSend::new(unsafe { SignalNoArgs::new() });
        // construct UI
        let ui = Ui::new(command_tx.clone(), data_rx);
        // connect data signal with UI handle_data slot method
        unsafe {
            data_signal.connect(&ui.slot_handle_data());
            // display the UI (non-blocking)
            ui.window.widget.show();
        }
        // spawn the background worker
        std::thread::spawn(move || {
            worker(command_rx, data_tx, data_signal);
        });
        // execute the Qt application until user exits (blocking)
        let result: i32 = unsafe { QApplication::exec() };
        // optionally terminate the background worker
        command_tx.send(Command::Quit).unwrap();
        result
    })
}
```

https://github.com/divi255/qtx

Bohemia Automation

The main function (Ritual)

Running the application

```rust
fn main() {
    QApplication::init(|_| {
        let (command_tx, command_rx) = std::sync::mpsc::sync_channel::<Command>(64);
        let (data_tx, data_rx) = std::sync::mpsc::sync_channel::<Data>(64);
        let data_signal = UnsafeSend::new(unsafe { SignalNoArgs::new() });
        // construct UI
        let ui = Ui::new(command_tx.clone(), data_rx);
        // connect data signal with UI handle_data slot method
        unsafe {
            data_signal.connect(&ui.slot_handle_data());
            // display the UI (non-blocking)
            ui.window.widget.show();
        }
        // spawn the background worker
        std::thread::spawn(move || {
            worker(command_rx, data_tx, data_signal);
        });
        // execute the Qt application until user exits (blocking)
        let result: i32 = unsafe { QApplication::exec() };
        // optionally terminate the background worker
        command_tx.send(Command::Quit).unwrap();
        result
    })
}
```

https://github.com/divi255/qtx

Bohemia Automation

The main function (Ritual)

Shutting down

```
1  fn main() {
2      QApplication::init(|_| {
3          let (command_tx, command_rx) = std::sync::mpsc::sync_channel::<Command>(64);
4          let (data_tx, data_rx) = std::sync::mpsc::sync_channel::<Data>(64);
5          let data_signal = UnsafeSend::new(unsafe { SignalNoArgs::new() });
6          // construct UI
7          let ui = Ui::new(command_tx.clone(), data_rx);
8          // connect data signal with UI handle_data slot method
9          unsafe {
10             data_signal.connect(&ui.slot_handle_data());
11             // display the UI (non-blocking)
12             ui.window.widget.show();
13         }
14         // spawn the background worker
15         std::thread::spawn(move || {
16             worker(command_rx, data_tx, data_signal);
17         });
18         // execute the Qt application until user exits (blocking)
19         let result: i32 = unsafe { QApplication::exec() };
20         // optionally terminate the background worker
21         command_tx.send(Command::Quit).unwrap();
22         result
23     })
24 }
```

https://github.com/divi255/qtx

Bohemia Automation

UI and Main window

```rust
// main window
#[ui_form("../ui/main.ui")]
struct Main {
    widget: QBox<QWidget>,
    counter: QPtr<QLineEdit>,
    btn_reset: QPtr<QPushButton>,
}

// UI
struct Ui {
    window: Main,
    command_tx: std::sync::mpsc::SyncSender<Command>,
    data_rx: std::sync::mpsc::Receiver<Data>,
}

// reqired to transform Rust functions into slots
impl cpp_core::StaticUpcast<QObject> for Ui {
    unsafe fn static_upcast(ptr: Ptr<Self>) -> Ptr<QObject> {
        ptr.window.widget.as_ptr().static_upcast()
    }
}
```

https://github.com/divi255/qtx

Bohemia Automation

UI Implementation

```rust
impl Ui {
    // Rc required to transform Rust functions into slots
    fn new(
      command_tx: std::sync::mpsc::SyncSender<Command>,
      data_rx: std::sync::mpsc::Receiver<Data>,
    ) -> Rc<Self> {
     let ui = Rc::new(Ui {
       window: Main::load(),
       command_tx,
       data_rx,
     });
     unsafe {
       ui.window
           .btn_reset
           .clicked()
           .connect(&ui.slot_handle_btn_reset());
     }
     ui
    }
    #[slot(SlotNoArgs)]
    fn handle_btn_reset(self: &Rc<Self>) {
      let _ = self.command_tx.send(Command::Reset);
    }
}
```

https://github.com/divi255/qtx

Bohemia Automation

UI Implementation

An alternative slot implementation

```rust
fn new(
    command_tx: std::sync::mpsc::SyncSender<Command>,
    data_rx: std::sync::mpsc::Receiver<Data>,
) -> Rc<Self> {
    let ui = Rc::new(Ui {
        window: Main::load(),
        command_tx: command_tx.clone(),
        data_rx,
    });
    // define a slot manually using a code closure
    unsafe {
        ui.window
            .btn_reset
            .clicked()
            .connect(&SlotNoArgs::new(&ui.window.widget, move || {
                // the main object is under Rc as it needs to be cloned to be moved into a slot
                // closure if required
                let _ = command_tx.send(Command::Reset);
            }));
    }
    ui
}
```

https://github.com/divi255/qtx

Bohemia Automation

Worker and Events

```rust
// commands to the background worker
enum Command {
    Reset,
    Quit,
}
// data from the background worker (alternative: slots, but objects must be Qt-ized)
enum Data {
    Counter(u64),
}
// background worker
fn worker (
    command_rx: std::sync::mpsc::Receiver<Command>,
    data_tx: std::sync::mpsc::SyncSender<Data>,
    data_signal: UnsafeSend<QBox<SignalNoArgs>>,
) {
    let mut counter = 0;
    loop {
      while let Ok(command) = command_rx.try_recv() {
        match command {
          Command::Reset => counter = 0,
          Command::Quit => return,
          }
        }
    if data_tx.try_send(Data::Counter(counter)).is_ok() {
      unsafe {
        data_signal.emit();
      }
    }
    std::thread::sleep(Duration::from_secs(1));
    counter += 1;
  }
}
```

https://github.com/divi255/qtx

Bohemia Automation

Handling data events in UI

```
impl Ui {
    // .................
    // .................
    // add the following method to UI
    #[slot(SlotNoArgs)]
    fn handle_data(self: &Rc<Self>) {
      while let Ok(data) = self.data_rx.try_recv() {
        match data {
          Data::Counter(v) =>{
            unsafe {
                self.window.counter.set_text(&qt_core::qs(v.to_string()));
            };
          }// match other data kinds if exist
        }
      }
    }
}
```

https://github.com/divi255/qtx

Bohemia Automation

How to distribute apps

- cargo bundle (Linux/OSX)
  https://github.com/burtonageo/cargo-bundle
- cargo wix (Windows)
  https://github.com/volks73/cargo-wix

Thank you for watching!

The presentation and the source code:

https://github.com/divi255/qtx

Production app example:

https://info.bma.ai/en/actual/eva4/ecmui/