C++ Coroutines and Qt

Akademy 2021

Daniel Vrátil

me@dvratil.cz

@dvratil | danvratil

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Blocking – Root of All Evil

```
QString RemotePlayer::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = QDBusReply<QString>{player.call("Title")};
    return title;
}
```

Let’s Fix It!

```
QCoro::Task<QString> PlayerController::getTitle() {
    QDBusInterface player{
        QStringLiteral(“org.mpris.MediaPlayer2.spotify”),
        QStringLiteral(“/org/mpris/MediaPlayer2”),
        QStringLiteral(“org.mpris.MediaPlayer2.Player”),
        QDBusConnection::sessionBus()
    };
    const QString title = co_await QDBusReply<QString>{player.call(“Title”)};
    co_return title;
}
```

Let's Fix It!

```
QCoro::Task<QString> PlayerController::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = co_await QDBusReply<QString>{player.call("Title")};
    co_return title;
}
```

Let's Fix It!

```
QCoro::Task<QString> PlayerController::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = co_await QDBusReply<QString>{player.call("Title")};
    co_return title;
}
```

Let's Fix It!

```
QCoro::Task<QString> PlayerController::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = co_await QDBusReply<QString>{player.call("Title")};
    co_return title;
}
```

Let's Fix It!

```
QCoro::Task<> PlayerWidget::update() {
    const auto title = co_await remotePlayer.getTitle();
    ui.titleLabel->setText(title);
    ...
}
```

Let's Fix It!

```
QCoro::Task<> PlayerWidget::update() {
    const auto title = co_await remotePlayer.getTitle();
    ui.titleLabel->setText(title);
    ...
}
```

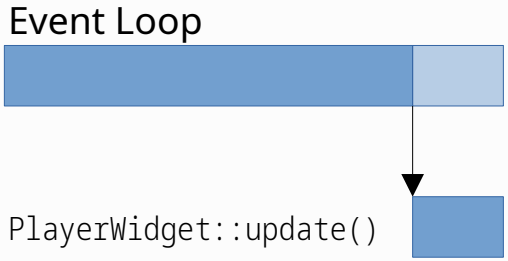
Let's Fix It!

```
QCoro::Task<> PlayerWidget::update() {
    const auto title = co_await remotePlayer.getTitle();
    ui.titleLabel->setText(title);
    ...
}
```

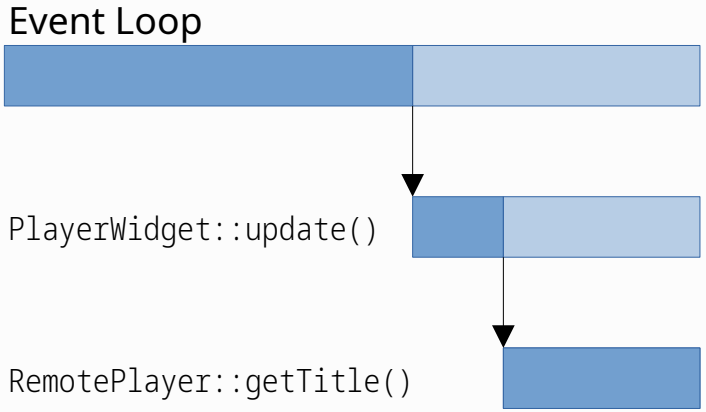
Execution Flow with Coroutine

Event Loop

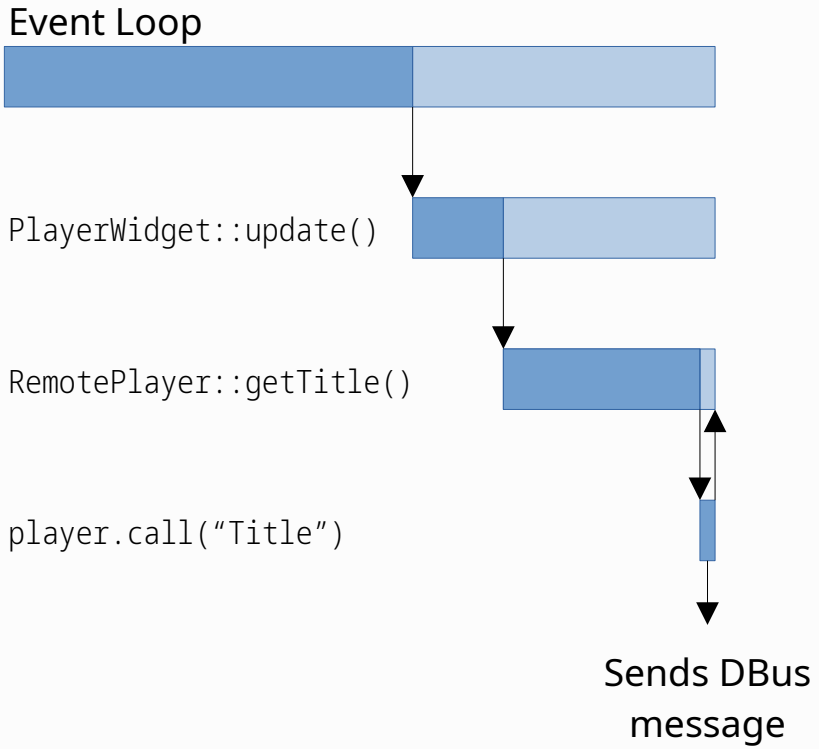
Execution Flow with Coroutine



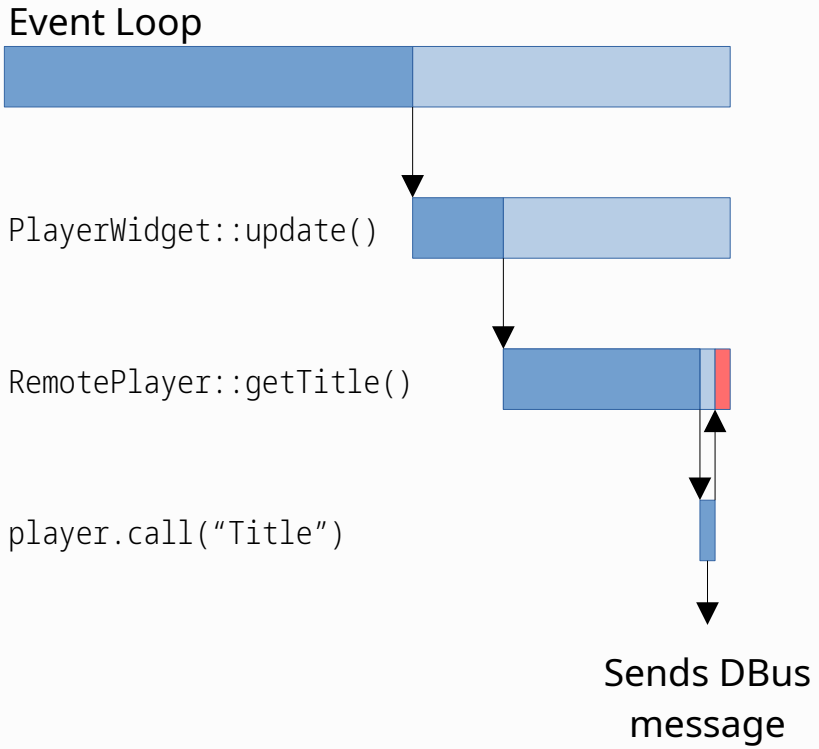
Execution Flow with Coroutine



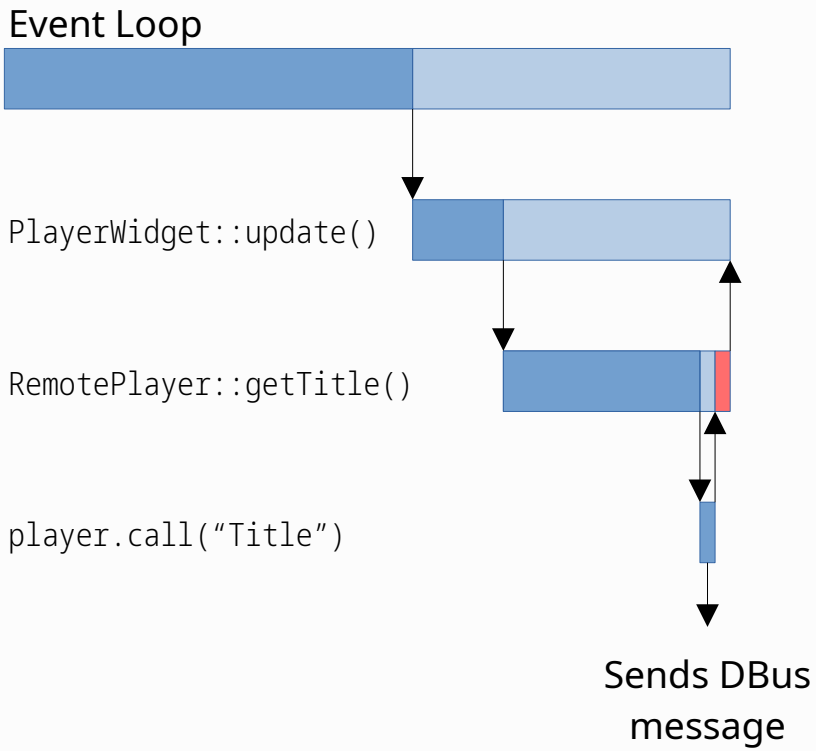
Execution Flow with Coroutine



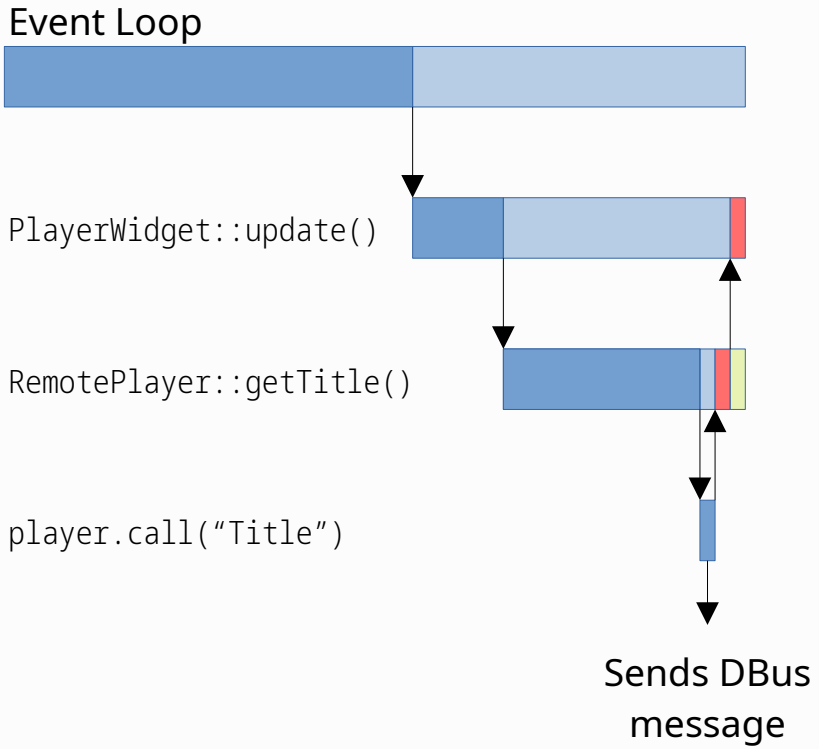
Execution Flow with Coroutine



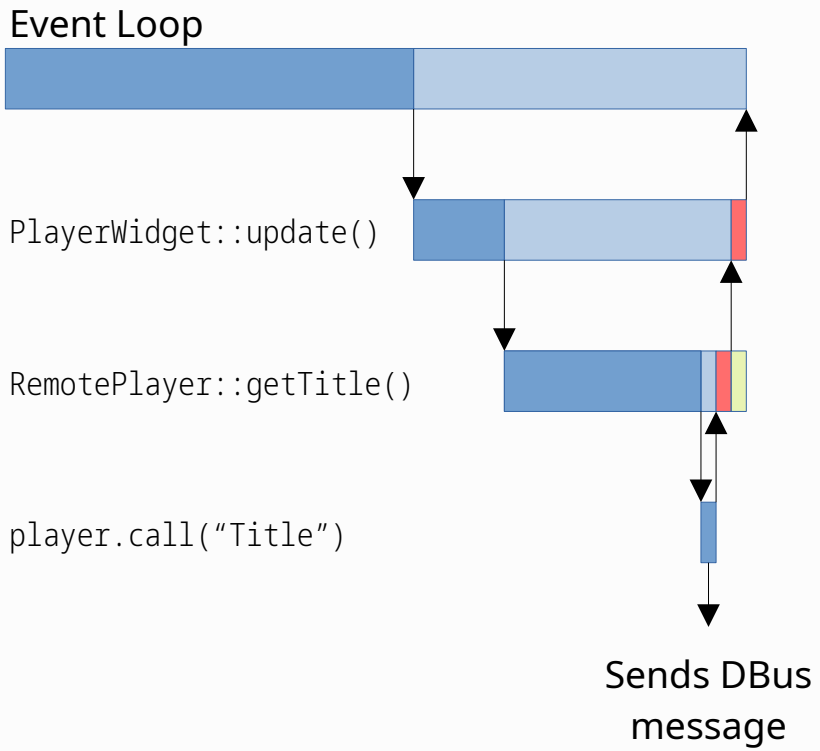
Execution Flow with Coroutine



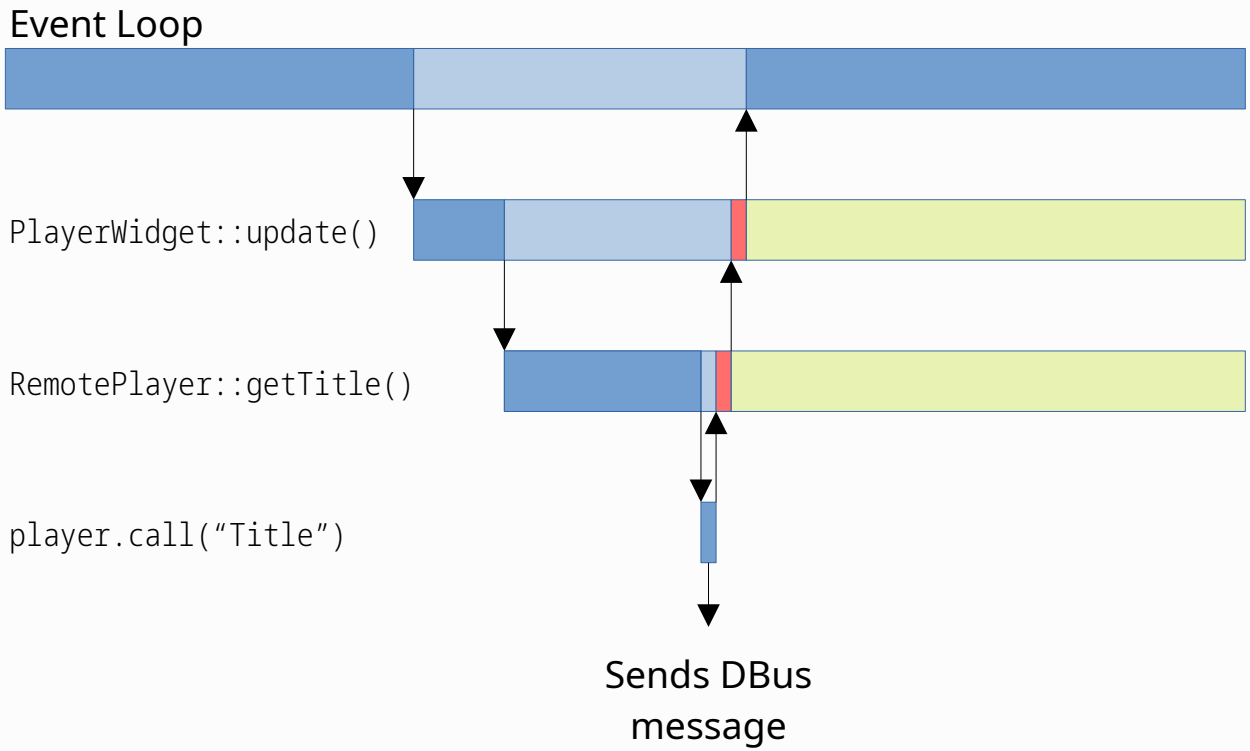
Execution Flow with Coroutine

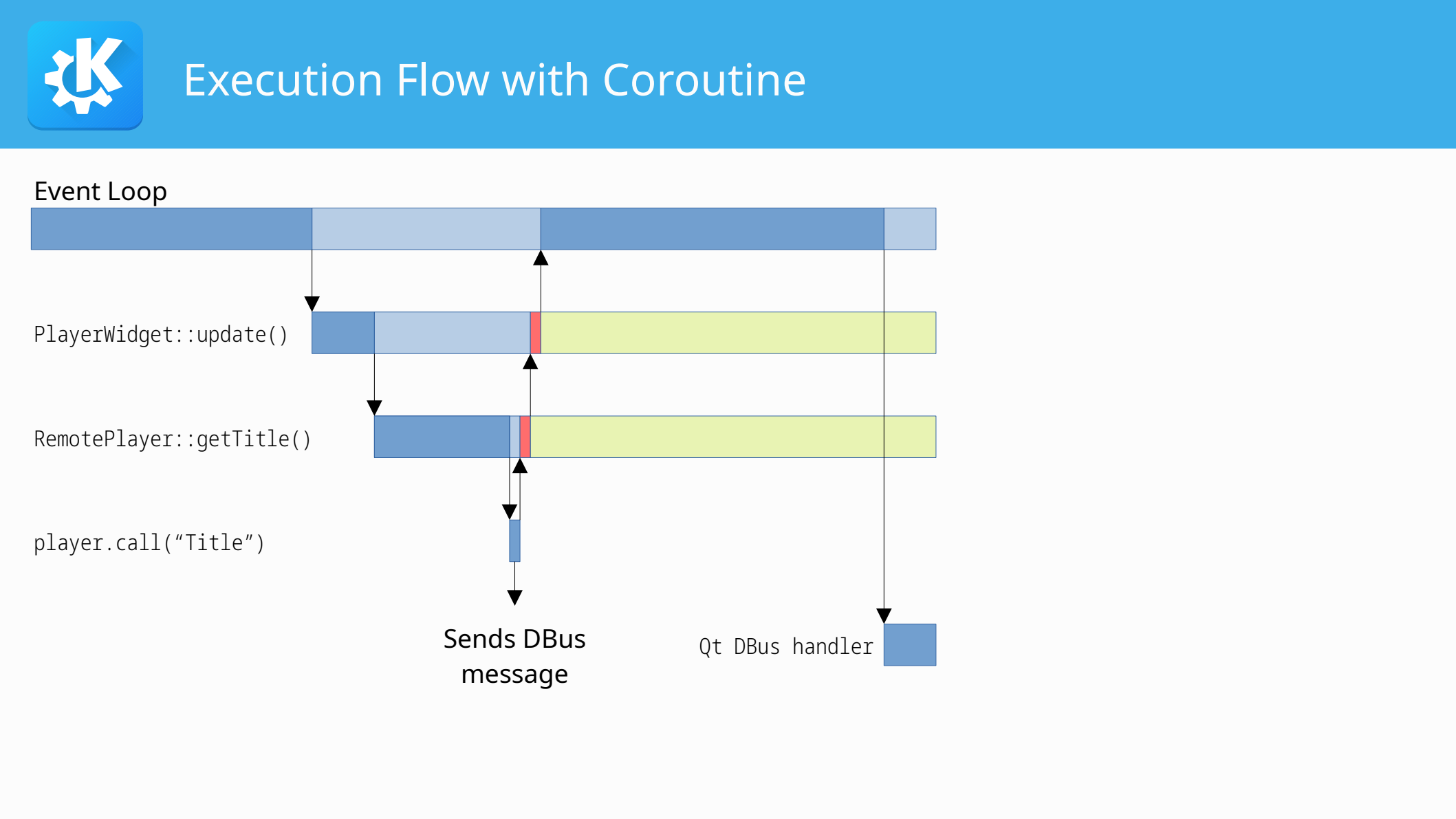


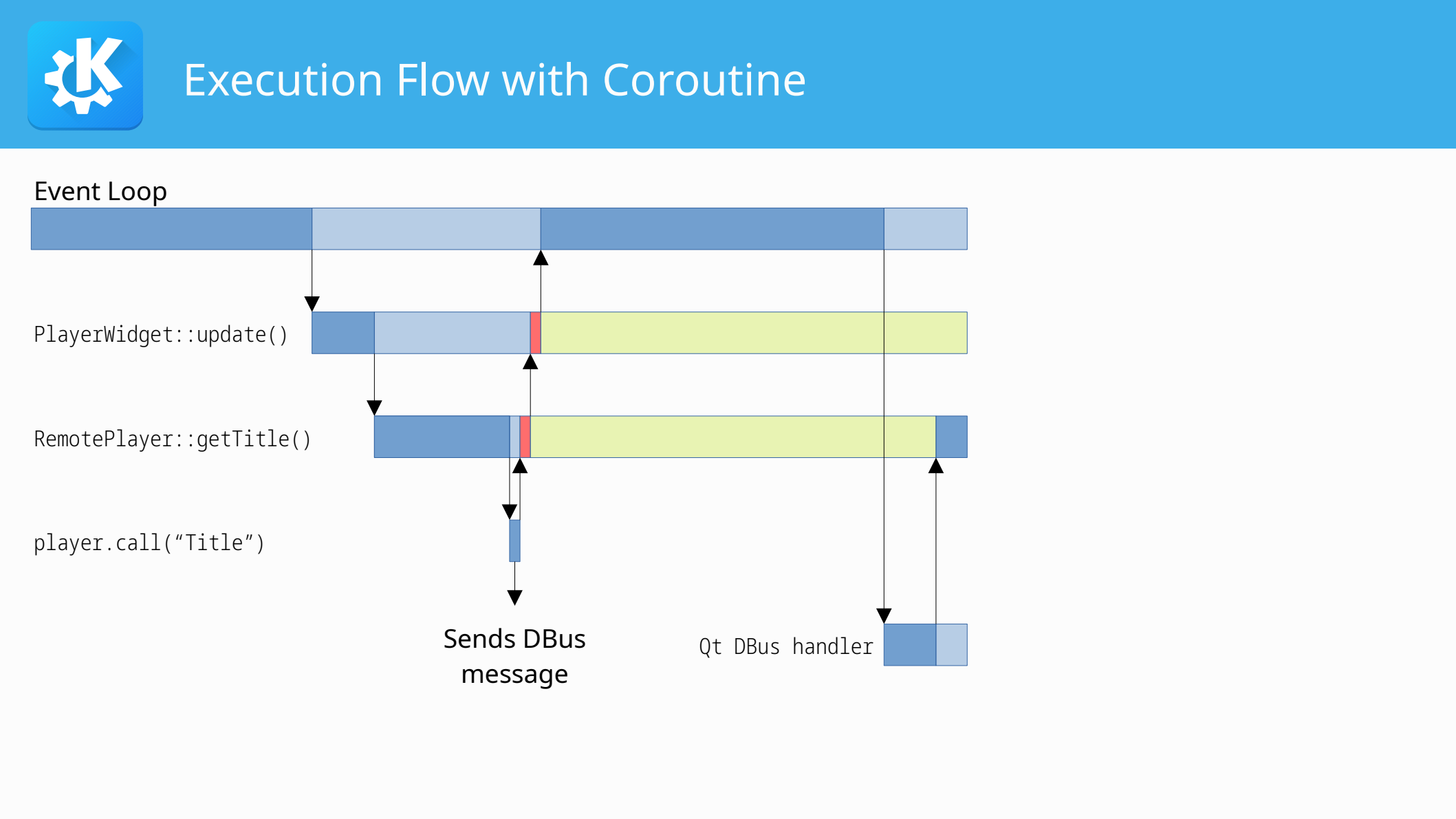
Execution Flow with Coroutine



Execution Flow with Coroutine




Execution Flow with Coroutine
Event Loop
PlayerWidget::update()
RemotePlayer::getTitle()
player.call("Title")
Sends DBus message
Qt DBus handler



Execution Flow with Coroutine

Event Loop

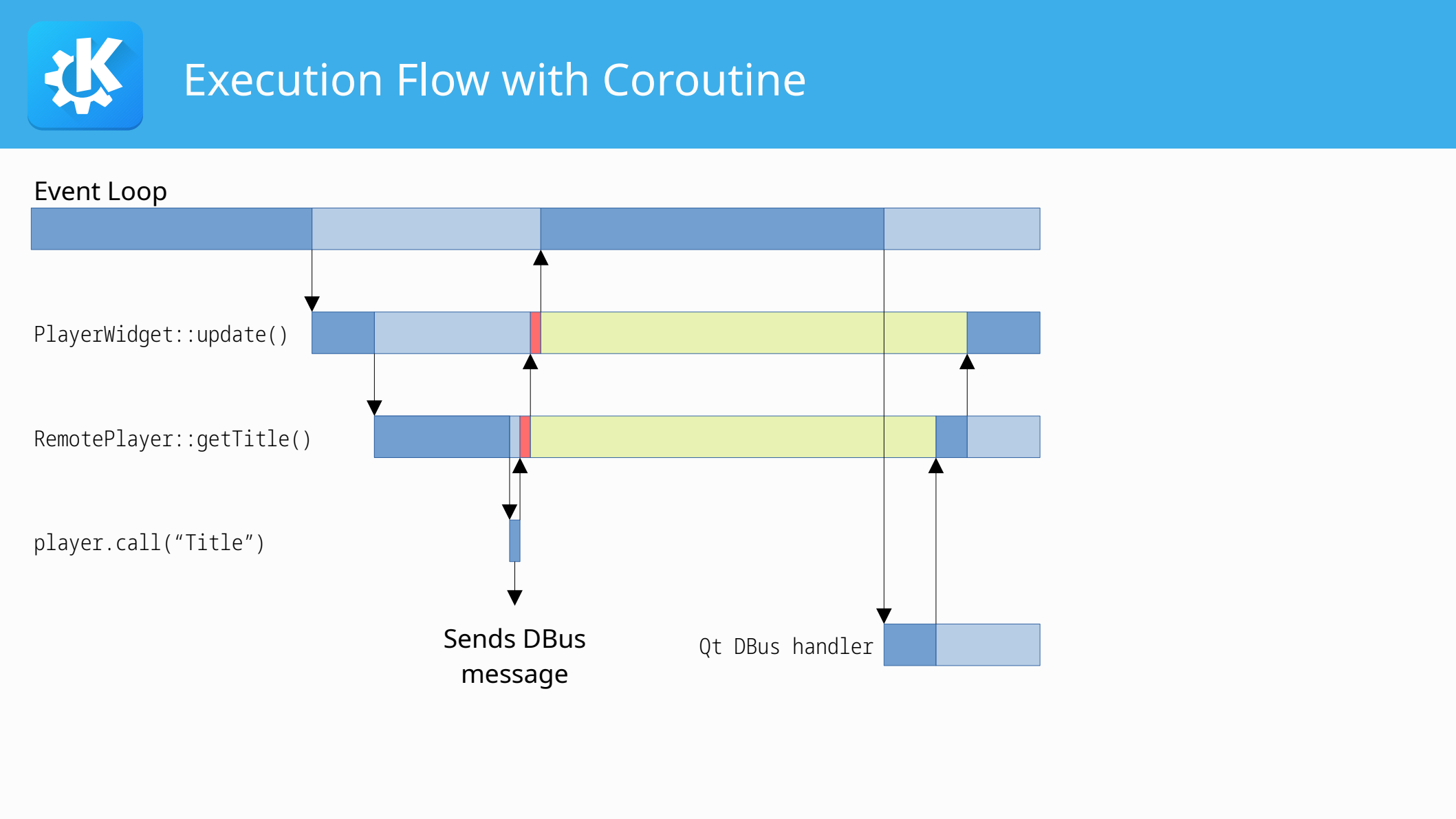
PlayerWidget::update()

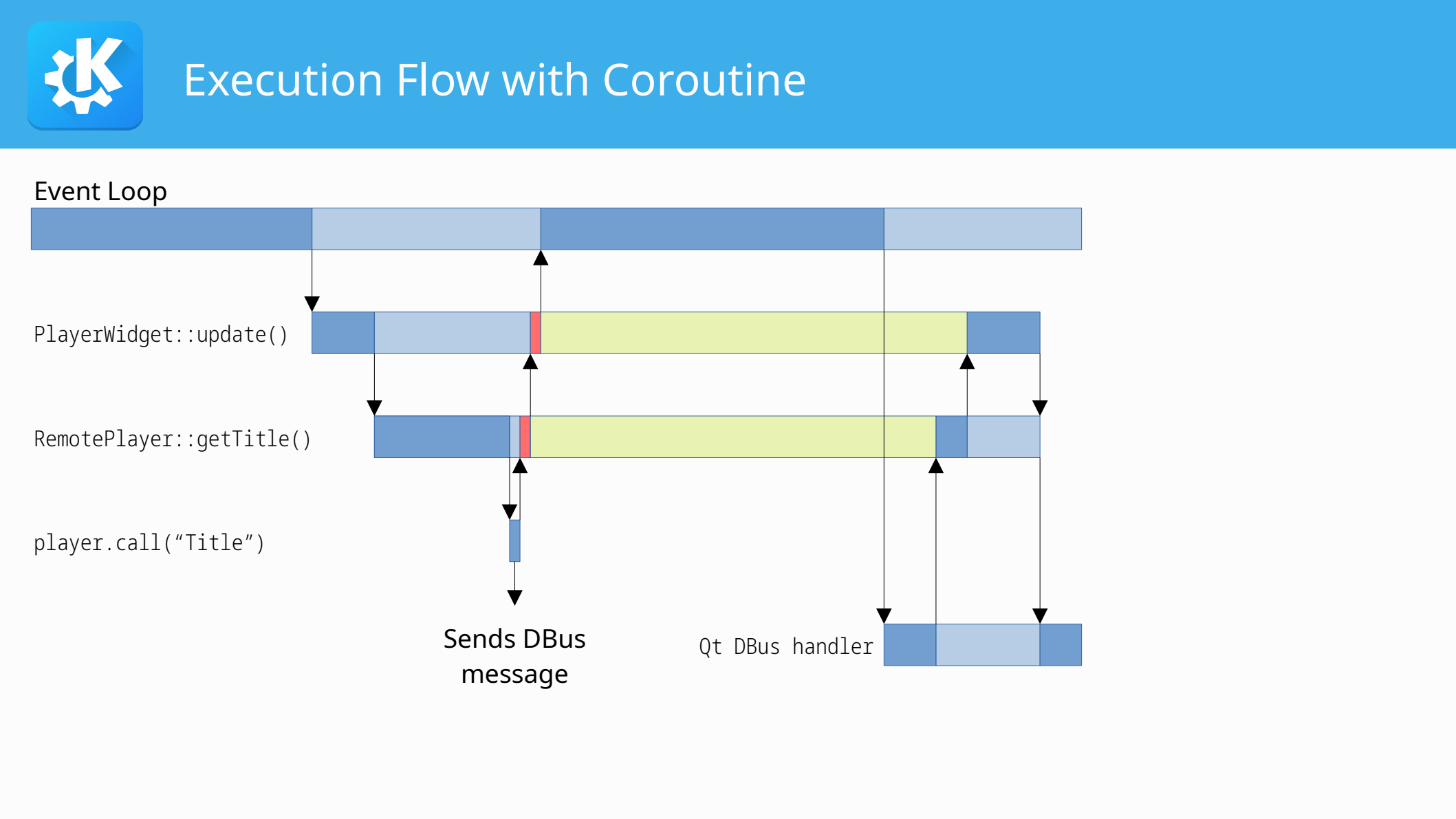
RemotePlayer::getTitle()

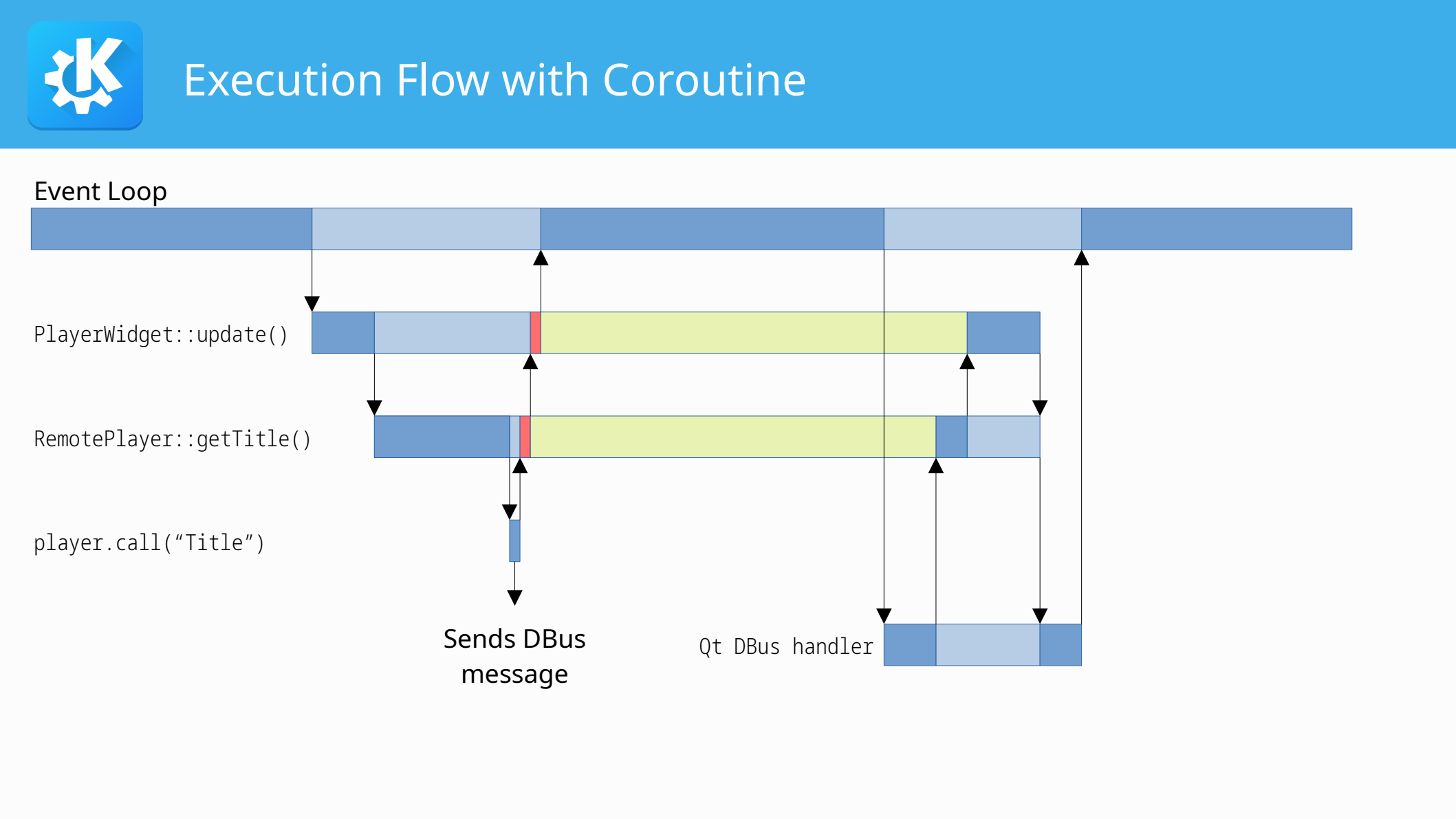
player.call("Title")

Sends DBus message

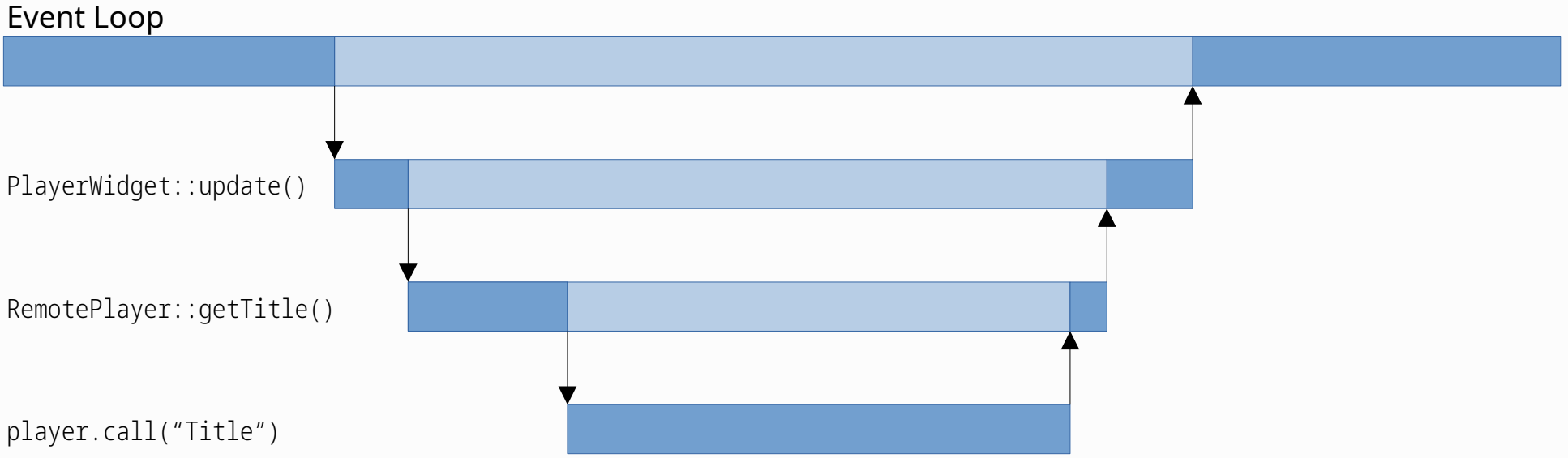
Qt DBus handler


Execution Flow with Coroutine
Event Loop
PlayerWidget::update()
RemotePlayer::getTitle()
player.call("Title")
Sends DBus message
Qt DBus handler


Execution Flow with Coroutine
Event Loop
PlayerWidget::update()
RemotePlayer::getTitle()
player.call("Title")
Sends DBus message
Qt DBus handler


Execution Flow with Coroutine
Event Loop
PlayerWidget::update()
RemotePlayer::getTitle()
player.call("Title")
Sends DBus message
Qt DBus handler

Execution Flow with Regular Functions



Coroutines in General

- Well-established concept in many programming languages
- Coroutine is a function whose execution can be suspended and resumed again at later point in time.
- When a coroutine is suspended, execution returns to its caller.
- When a coroutine is resumed, it’s as if it were called from the function that resumed it.
- Allows writing asynchronous code in a synchronous manner.
- Does not necessarily improve performance.

Coroutines in C++

1) Introduced in C++20
2) Compiler support
   - GCC 10 (`-fcoroutines`), `<coroutine>`
   - Clang 5 (`-fcoroutines-ts -stdlib=libc++`), `<experimental/coroutine>`
   - MSVC 14.28 (VS 2019 16.8) (`/await`), `<coroutine>`
3) Core language extension
4) Limited support in standard library (only bare minimum to support compilers and coroutine library writers)

Coroutines in C++ (cont.)

Three new keywords:

- `co_await` – suspends the coroutine while awating completion
- `co_return` – returns a result from a coroutine (like return)
- `co_yield` – provides intermediate result (useful for generators)

Standard Library

- `coroutine_handle` – handle, allows resuming the coroutine
- `suspend_never` – indicates that the coroutine should never suspend
- `suspend_always` – indicates that the coroutine should always suspend

How To Make a Coroutine

1) Must contain `co_await` or `co_yield` keyword
2) Must have a return type that
   - has a `promise_type` typedef specifying the *Promise* type
   - usually templated – e.g. `Task<T>` - where `T` is the type of the actual coroutine result
   - the STL does not provide any such return type!
   - The return type is an interface for the caller, the *Promise* type is an interface for the coroutine
3) Must use `co_return` instead of `return`

But Wait!

```
QCoro::Task<QString> PlayerController::getTitle() {
    QDBusInterface player{
        QStringLiteral("org.mpris.MediaPlayer2.spotify"),
        QStringLiteral("/org/mpris/MediaPlayer2"),
        QStringLiteral("org.mpris.MediaPlayer2.Player"),
        QDBusConnection::sessionBus()
    };
    const QString title = co_await QDBusReply<QString>{player.call("Title")};
    co_return title;
}
```

Coroutines and Qt

- `co_await` expects an argument that either
  - is *Awaitable* or,
  - there's a compatible `promise_type::await_transform(T)` overload
    - `await_transform` returns a wrapper for `T` that is *Awaitable*

QCoro Library

- A small library that provides
  - a custom *Awaitable* return type for coroutines
  - `await_transform()` implementation for many Qt types
  - thin *Awaitable* wrappers for async operations on various Qt types

QCoro Library

Certain types only have one operation that makes sense to be co_awaited, e.g. `QFuture`, or `QNetworkReply`

```
const int theAnswer = co_await QtConcurrent::run([]() {
    std::this_thread::sleep_for(std::chrono::years{7'500'000});
    return 42;
});
```

```
QNetworkAccessManager nam;
const auto *reply = co_await nam.get("https://kde.org/...");
```

QCoro Library - QProcess

Certain types have multiple operations that can be co_awaited – for those QCoro provides wrapper types using `qCoro()`.

```
QProcess process;
process.start("/usr/bin/gpg2", {"-d"});

co_await qCoro(process).waitForStarted();

process.write(encryptedData);
process.closeWriteChannel();

co_await qCoro(process).waitForFinished();
const QByteArray decryptedData = process.readAll();
```

QCoro Library - QTcpServer/QTcpSocket

```
QTcpServer server;
server.listen(…);
while (!stop) {
    auto *socket = co_await qCoro(server).waitForNewConnection(5s);
    if (socket == nullptr) continue;

    co_await qCoro(socket)->waitForReadyRead();
    const auto data = socket->readAll();
    socket->write(“PONG “ + data);
    socket->close();
    delete socket;
}
```

QCoro Library

- `QDBusPendingCall`, `QDBusPendingReply`
- `QFuture`
- `QIODevice`
- `QNetworkReply`
- `QAbstractSocket`, `QLocalSocket`, `QTcpServer`
- `QProcess`
- `QTimer`
- Qt signal

Questions?

QCoro
github.com/danvratil/qcoro
qcoro.dvratil.cz

Daniel Vrátil
me@dvratil.cz
@dvratil | danvratil