CS241 Principles and Practice of Problem Solving

Lecture 10: GUI Programming with Qt I

Yuye Ling, Ph.D.

Shanghai Jiao Tong University
John Hopcroft Center for Computer Science

October 17, 2019

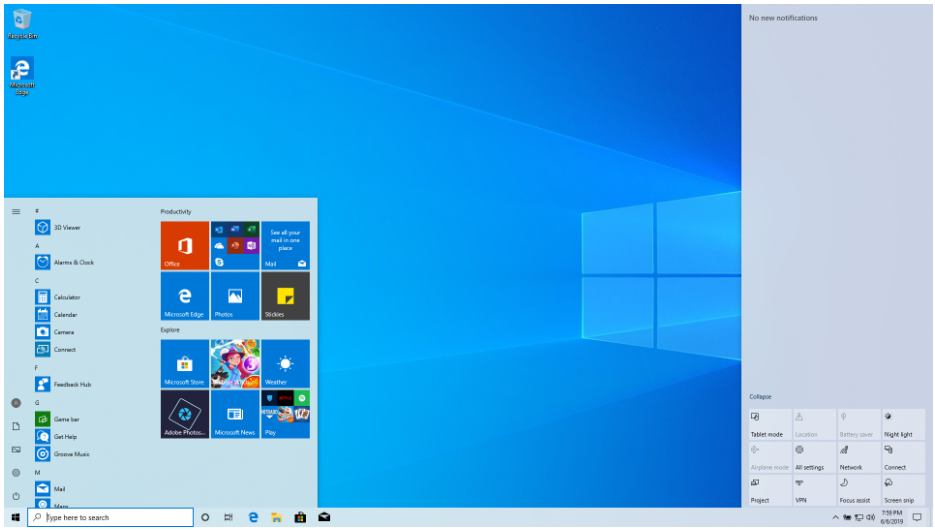
Copyright Notice

A large portion of the contents in the following pages come from Digia(Legacy)'s training slides.
Freely accessible copies could be found here.

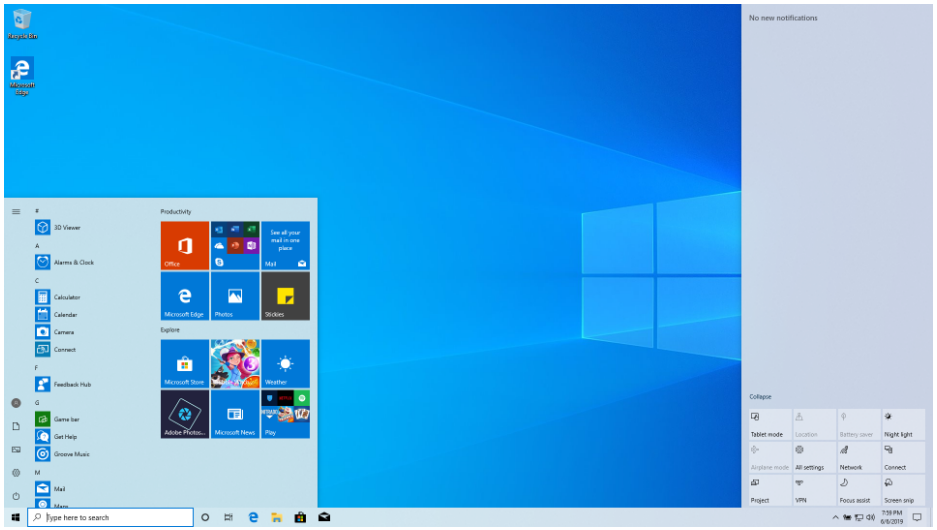
What is a graphical user interface?

What is a graphical user interface?

What is a graphical user interface?



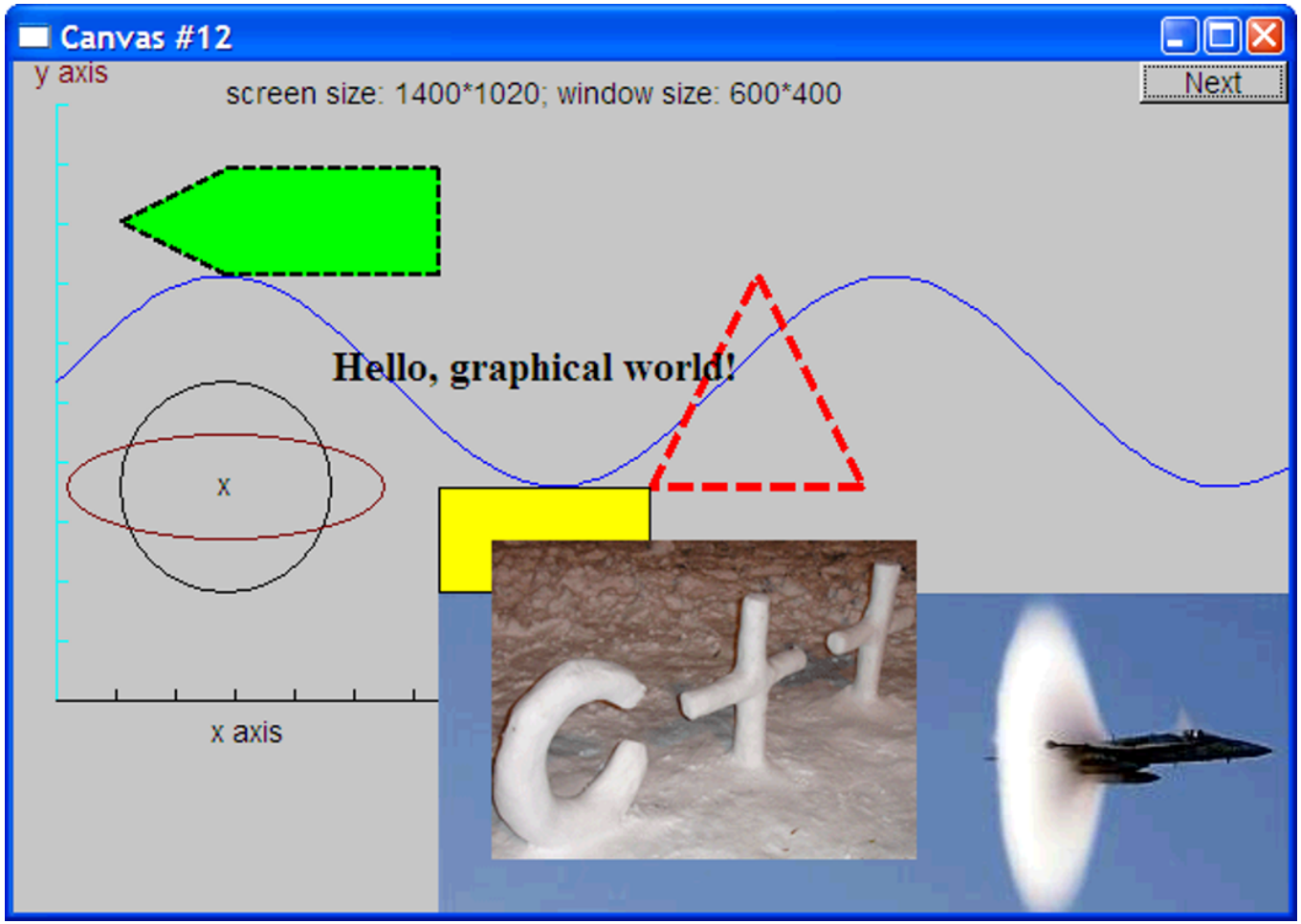
What is a graphical user interface?



Definition: a computer program designed to allow a computer user to interact easily with the computer typically by making choices from menus or groups of icons.

What (I think) is NOT a GUI

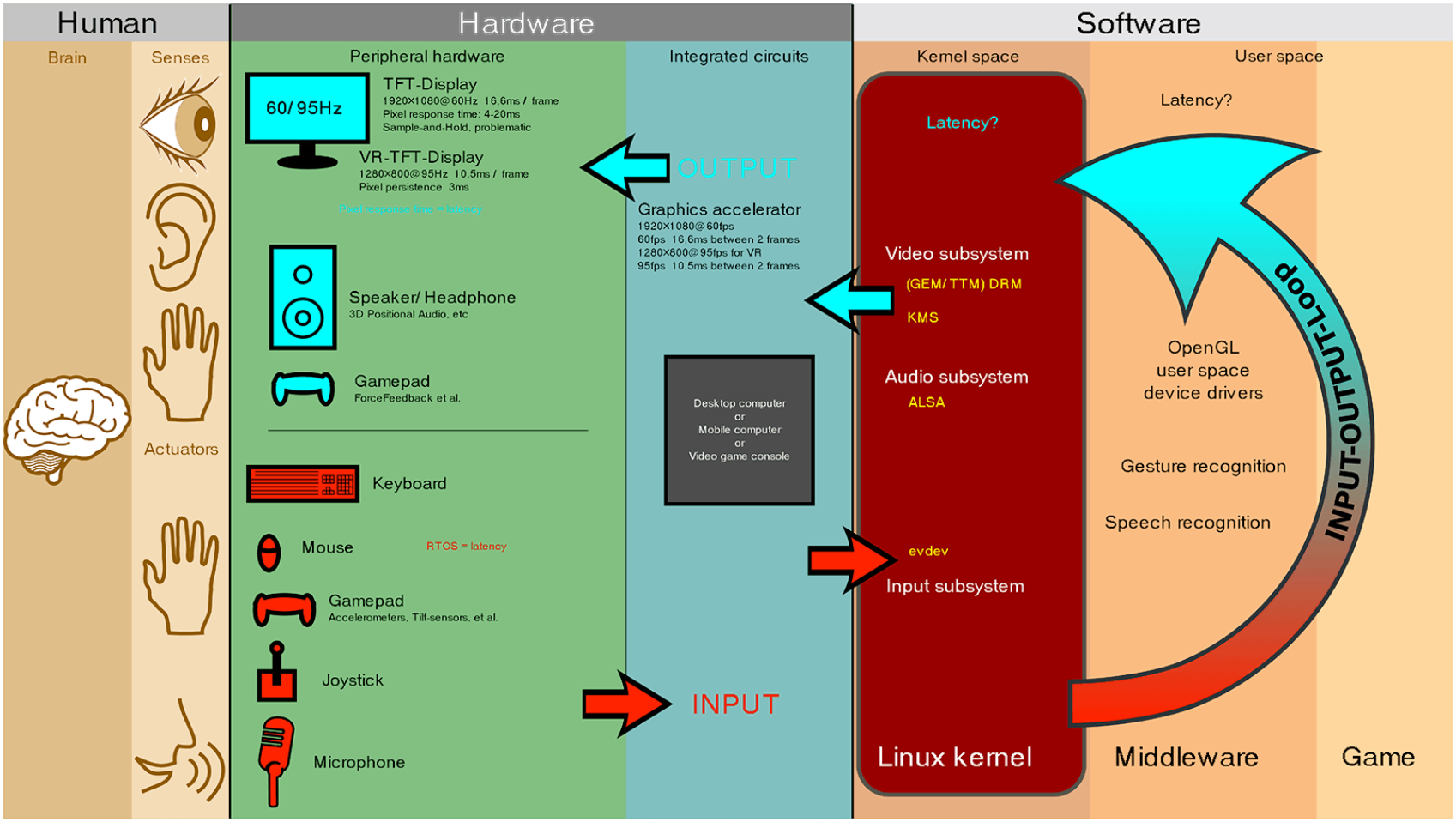
GUI is more than drawing a picture on the canvas.



What IS a GUI

What IS a GUI

What IS a GUI


Human
Brain
Senses
Actuators
Hardware
Peripheral hardware
60/95Hz
TFT-Display
1920X1080@60Hz 16.6ms / frame
Pixel response time: 4-20ms
Sample-and-Hold, problematic
VR-TFT-Display
1280X800@95Hz 10.5ms / frame
Pixel persistence 3ms
Pixel response time = latency
Speaker/ Headphone
3D Positional Audio, etc
Gamepad
ForceFeedback et al.
Keyboard
Mouse
RTOS = latency
Gamepad
Accelerometers, Tilt-sensors, et al.
Joystick
Microphone
Integrated circuits
OUTPUT
Graphics accelerator
1920X1080@60fps
60fps 16.6ms between 2 frames
1280X800@95fps for VR
95fps 10.5ms between 2 frames
Desktop computer
or
Mobile computer
or
Video game console
INPUT
Software
Kernel space
Latency?
Video subsystem
(GEM/ TTM) DRM
KMS
Audio subsystem
ALSA
evdev
Input subsystem
Linux kernel
User space
Latency?
OpenGL
user space
device drivers
Gesture recognition
Speech recognition
Middleware
Game
INPUT-OUTPUT-LOOP

Class schedule

- I: Introduction & Object communication
- II: Widgets & Painting
- III: Application & Dialog
- IV: Multi-threading in Qt
- V: Brief introduction on OpenGL

Some general issues before we go

Why study GUIs?

- Learn about *event-driven programming techniques*

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize
  - Part of learning is "letting go."

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize
  - Part of learning is "letting go."
  - First, learn the fundamental concepts and general ideas.

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize
  - Part of learning is "letting go."
  - First, learn the fundamental concepts and general ideas.
  - Then, look things up as you need them

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize
  - Part of learning is "letting go."
  - First, learn the fundamental concepts and general ideas.
  - Then, look things up as you need them
- A chance to see how it is designed and learn from it:

Why study GUIs?

- Learn about *event-driven programming techniques*
- Practice learning and using a large, complex application programming interface (API)
  - Caution: There is way more here than you can memorize
  - Part of learning is "letting go."
  - First, learn the fundamental concepts and general ideas.
  - Then, look things up as you need them
- A chance to see how it is designed and learn from it:
  - E.g. Design patterns
  - Model-view separation, callbacks, inheritance v.s. delegation

How can we implement a GUI?

How can we implement a GUI?

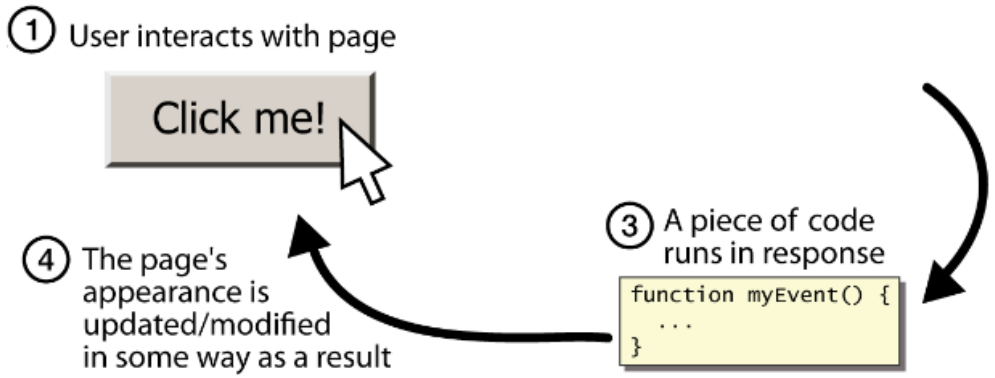
General question

How do we get from "the user clicks a button" to your business logic?

How can we implement a GUI?

General question

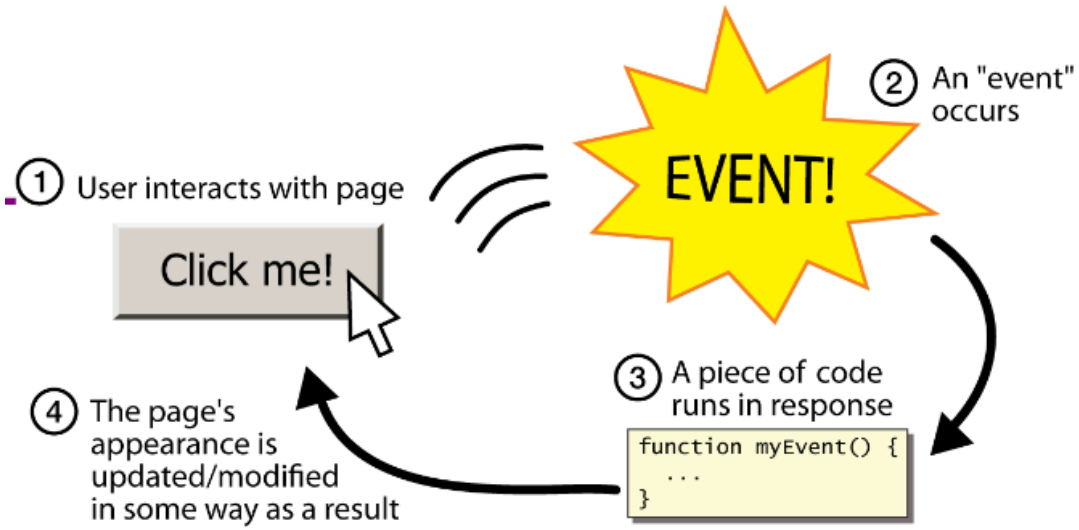
How do we get from "the user clicks a button" to your business logic?



How can we implement a GUI?

General question

How do we get from "the user clicks a button" to your business logic?



Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing
  - Question: is command-line interface an even-driven interface?

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing
  - Question: is command-line interface an even-driven interface?
- The program loads, then waits for user input events.

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing
  - Question: is command-line interface an even-driven interface?
- The program loads, then waits for user input events.
- As each event occurs, the program runs particular code to respond.

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing
  - Question: is command-line interface an even-driven interface?
- The program loads, then waits for user input events.
- As each event occurs, the program runs particular code to respond.
- The overall flow of what code is executed is determined by the series of events that occur

Event-driven programming

A style of coding where a program's overall flow of execution is dictated by events.

- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing
  - Question: is command-line interface an even-driven interface?
- The program loads, then waits for user input events.
- As each event occurs, the program runs particular code to respond.
- The overall flow of what code is executed is determined by the series of events that occur
- Quick summary: It is essentially a communication system

Graphical events

Event

- An object that represents a user's interaction with a GUI component
- Can be "handled" to create interactive component

Graphical events

Event

- An object that represents a user's interaction with a GUI component
- Can be "handled" to create interactive component
- Question: does this recall you anything we have learned before?

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.
- The listener will be notified when the event occurs (e.g. button click).

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.
- The listener will be notified when the event occurs (e.g. button click).
- Used in Java(Script)

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.
- The listener will be notified when the event occurs (e.g. button click).
- Used in Java(Script)

The main body of the program is

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.
- The listener will be notified when the event occurs (e.g. button click).
- Used in Java(Script)

The main body of the program is an event loop.

Possible solution I: Observer pattern (Listener)

Listener

An object that waits for events and responds to them.

- To handle an event, attach a *listener* to a component.
- The listener will be notified when the event occurs (e.g. button click).
- Used in Java(Script)

The main body of the program is an event loop.
Abstractly:

```
do {
    e = getNextEvent();
    process event e;
} while (e != quit)
```

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered
- Formally: A callback is a pointer to a function that is called when an event occurs, any function can be assigned to a callback.

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered
- Formally: A callback is a pointer to a function that is called when an event occurs, any function can be assigned to a callback.

Example

If a user clicks a `Close` button, we want the window's `close()` function to be called.

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered
- Formally: A callback is a pointer to a function that is called when an event occurs, any function can be assigned to a callback.

Example

If a user clicks a `Close` button, we want the window's `close()` function to be called.

- Again, this sounds familiar, huh?

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered
- Formally: A callback is a pointer to a function that is called when an event occurs, any function can be assigned to a callback.

Example

If a user clicks a `Close` button, we want the window's `close()` function to be called.

- Again, this sounds familiar, huh?
  - From certain perspective, *exception handling* could be viewed as an unusual event handling

Possible solution II: Callbacks

- Conceptually: perform an action when an event is triggered
- Formally: A callback is a pointer to a function that is called when an event occurs, any function can be assigned to a callback.

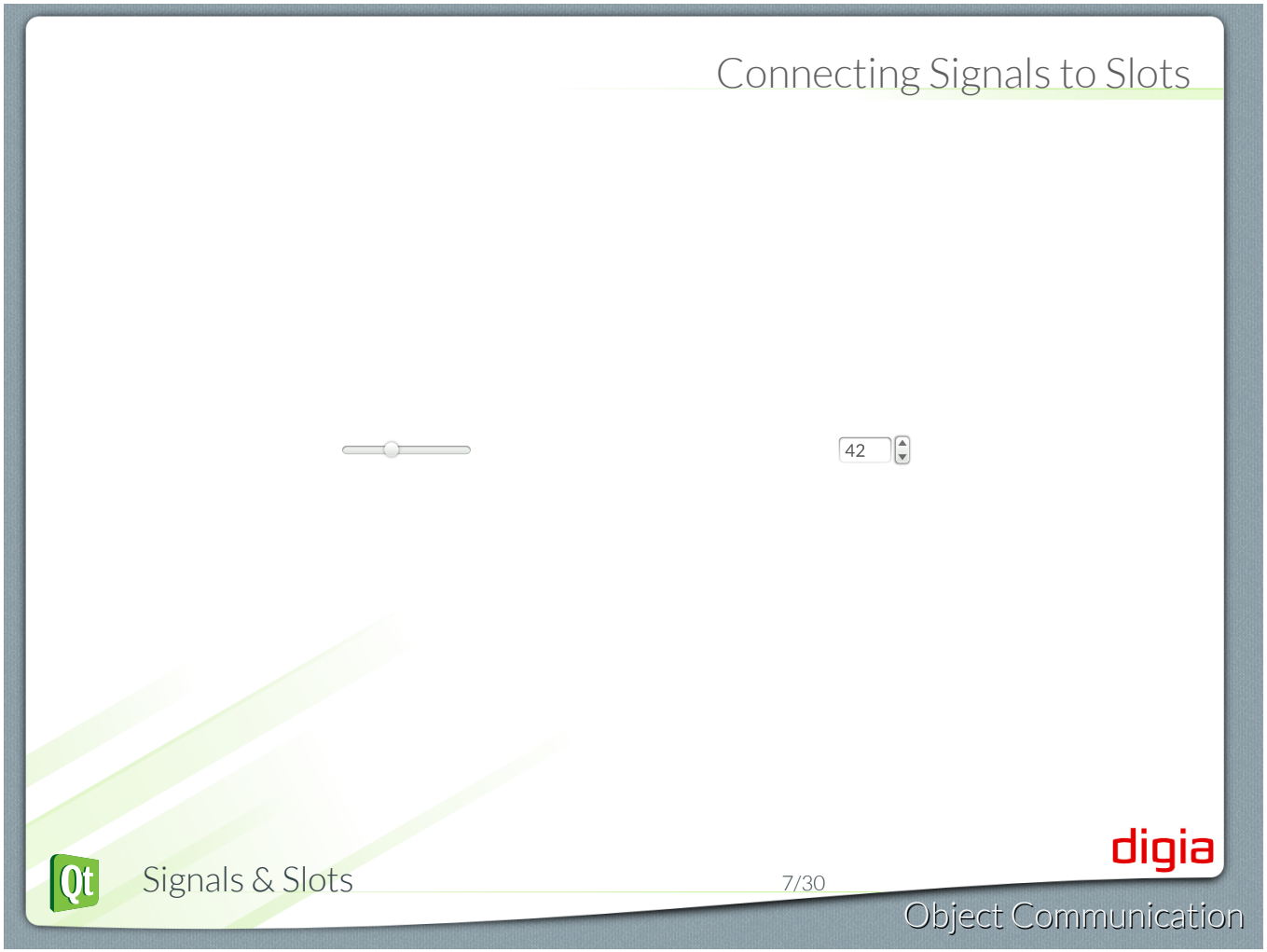
Example

If a user clicks a `Close` button, we want the window's `close()` function to be called.

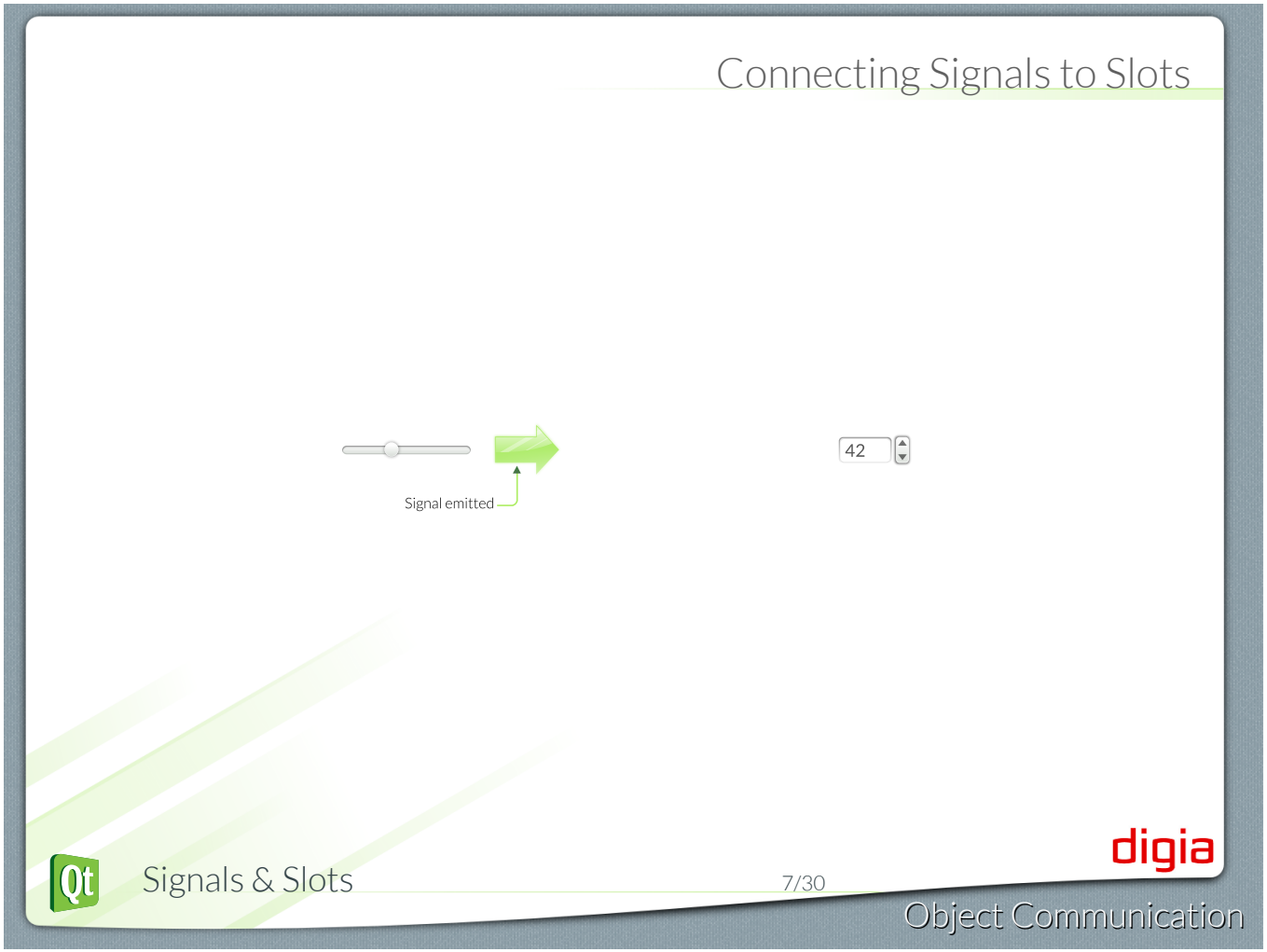
- Again, this sounds familiar, huh?
  - From certain perspective, *exception handling* could be viewed as an unusual event handling
  - Why unusual?

Signals and slots

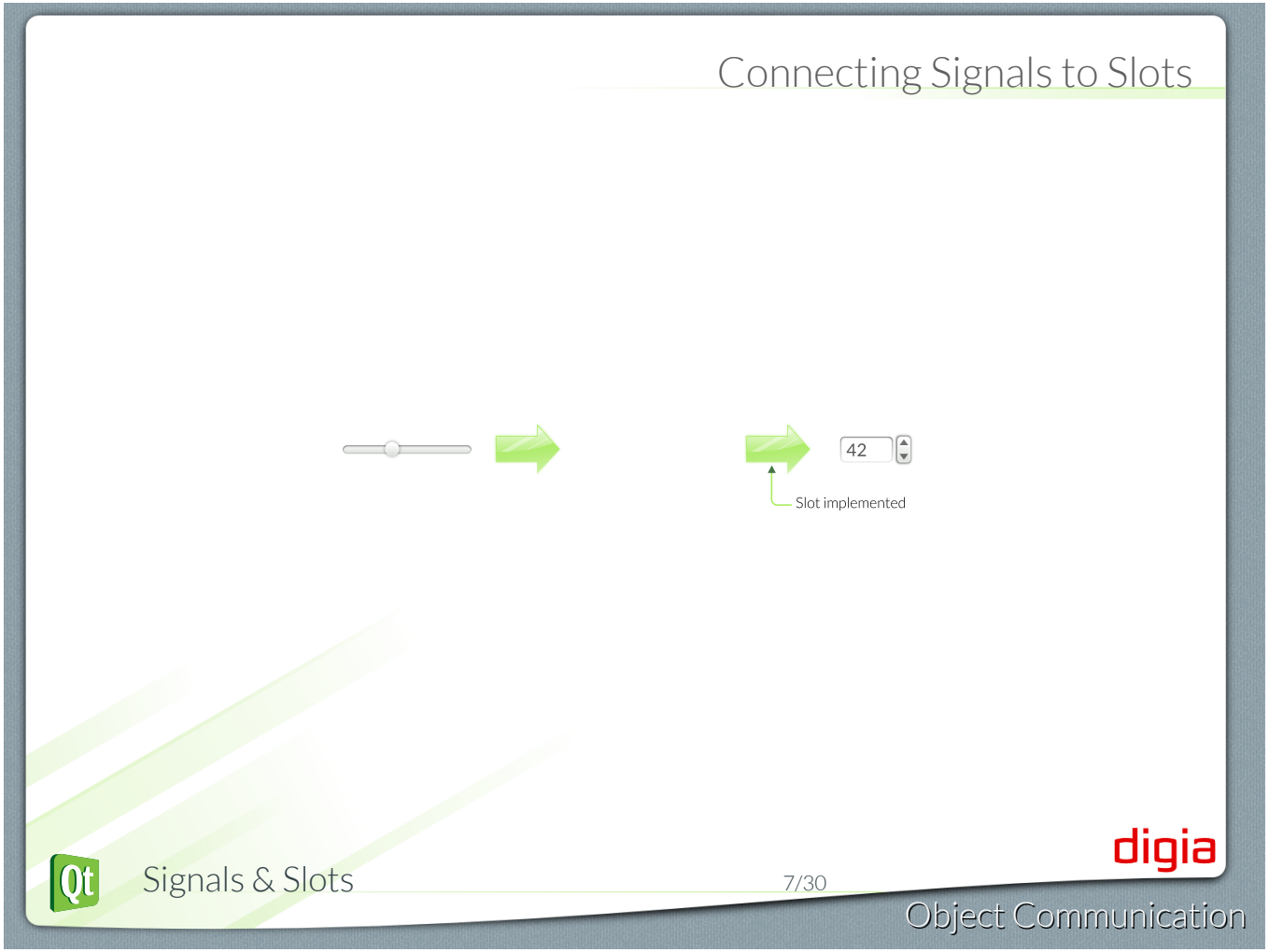
Signals and slots



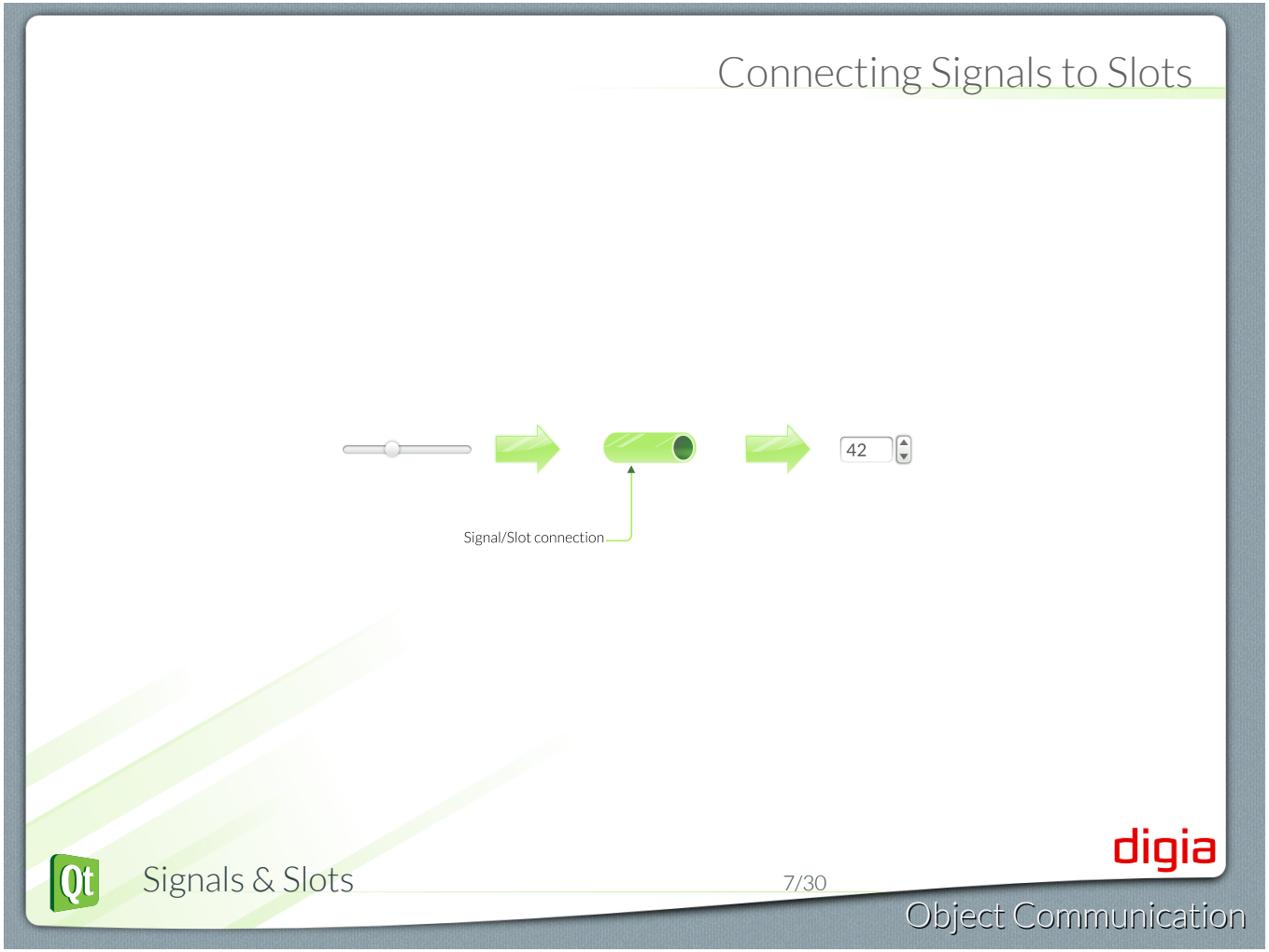
Signals and slots



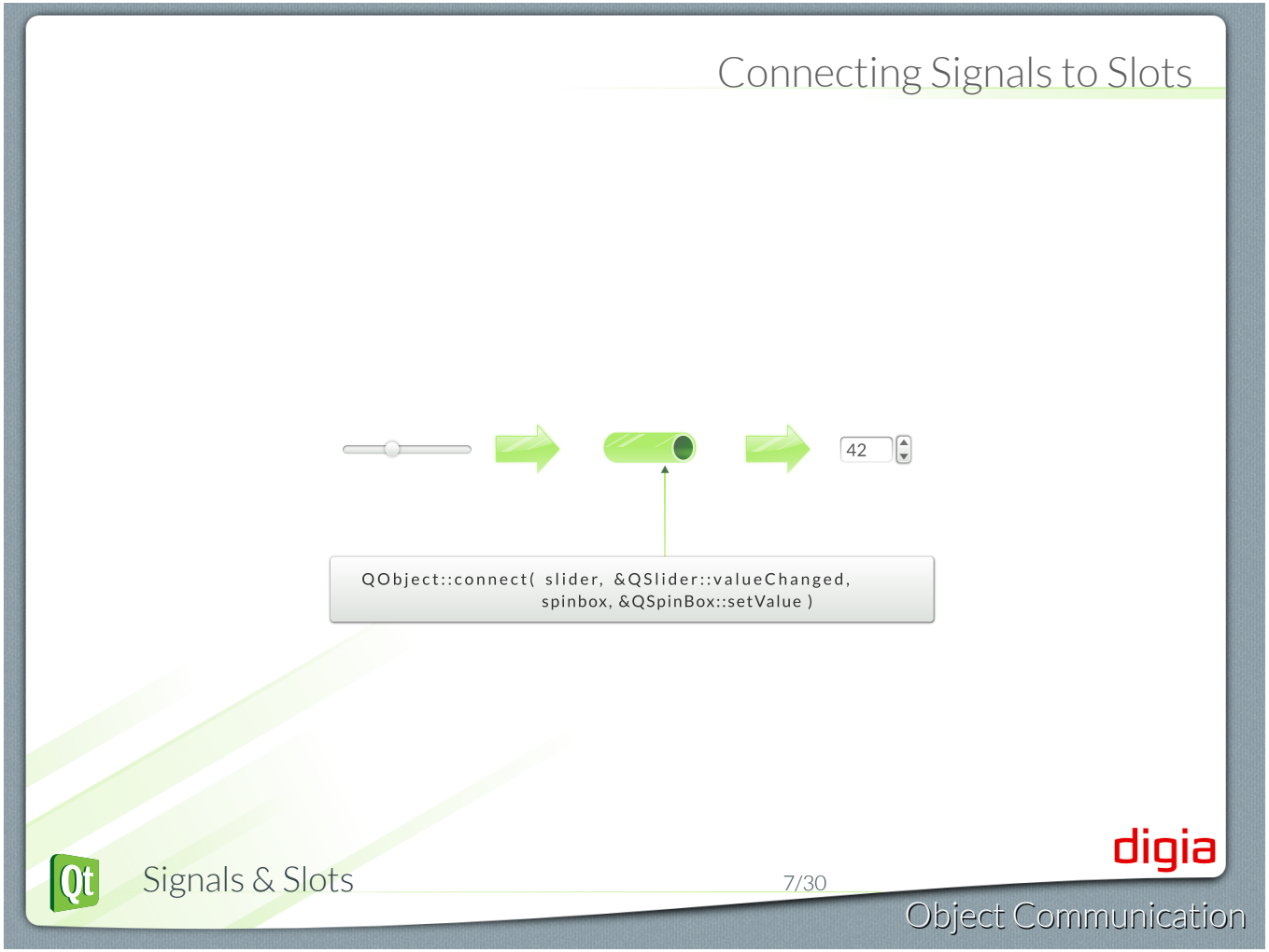
Signals and slots



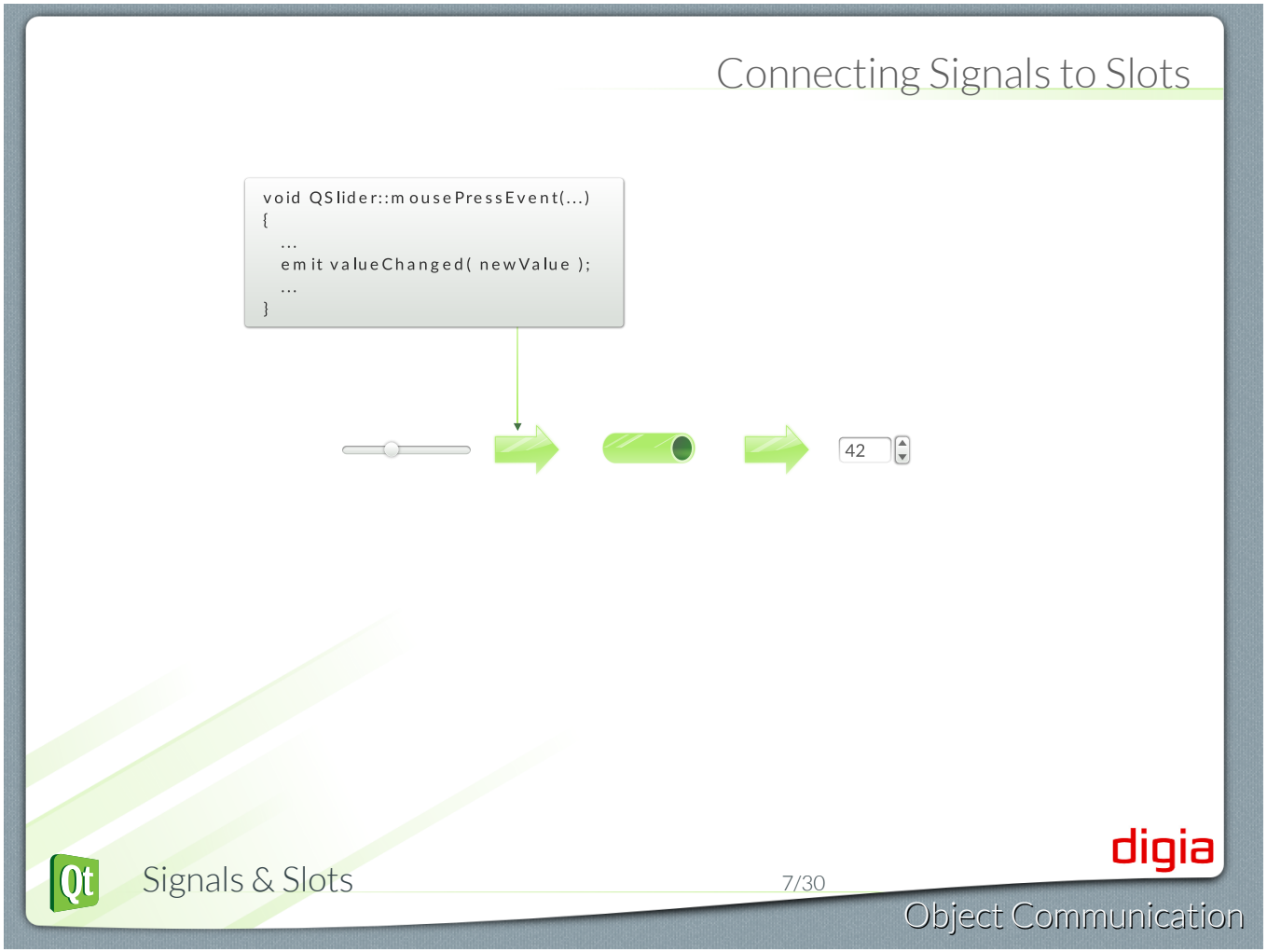
Signals and slots



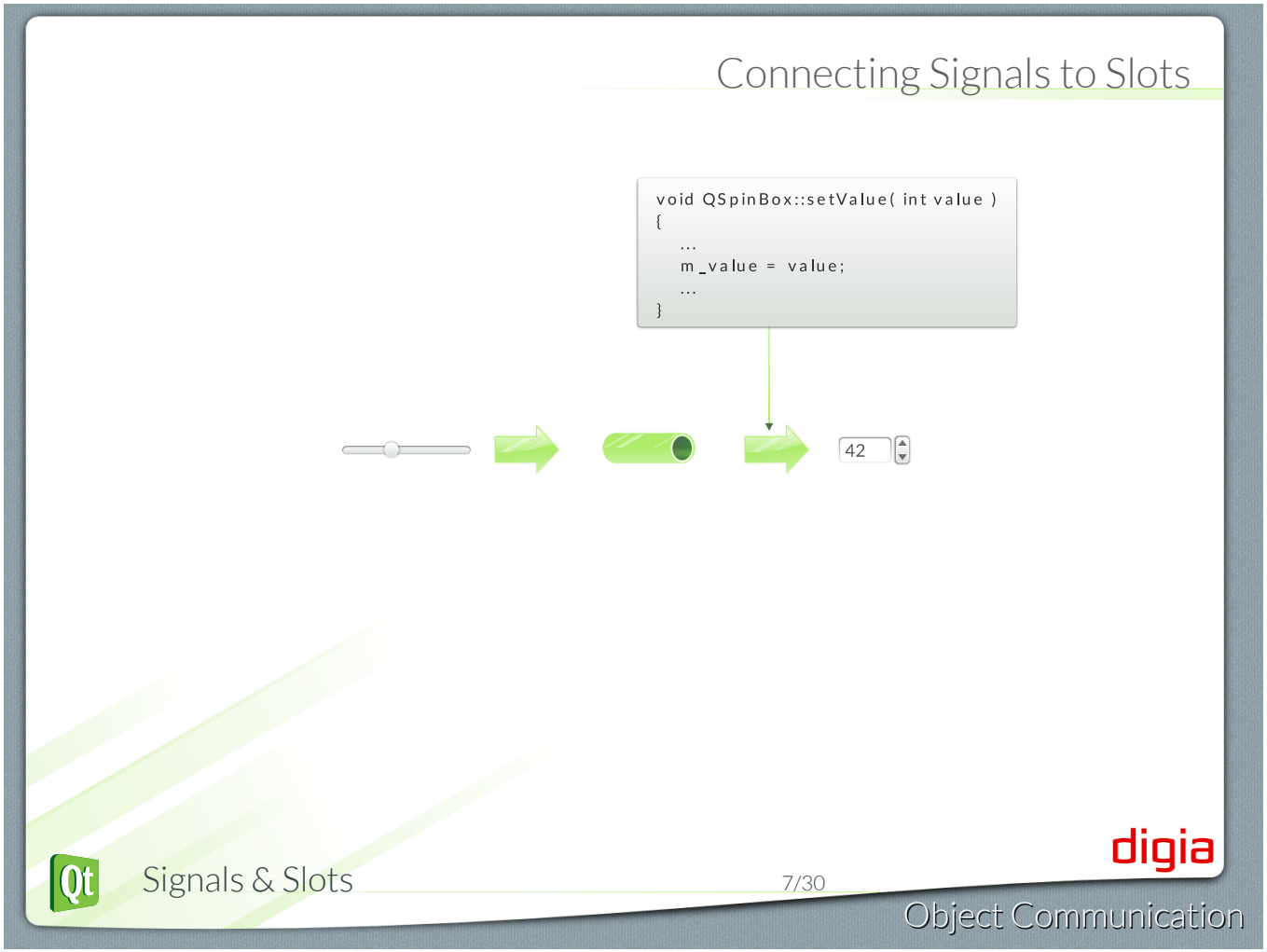
Signals and slots



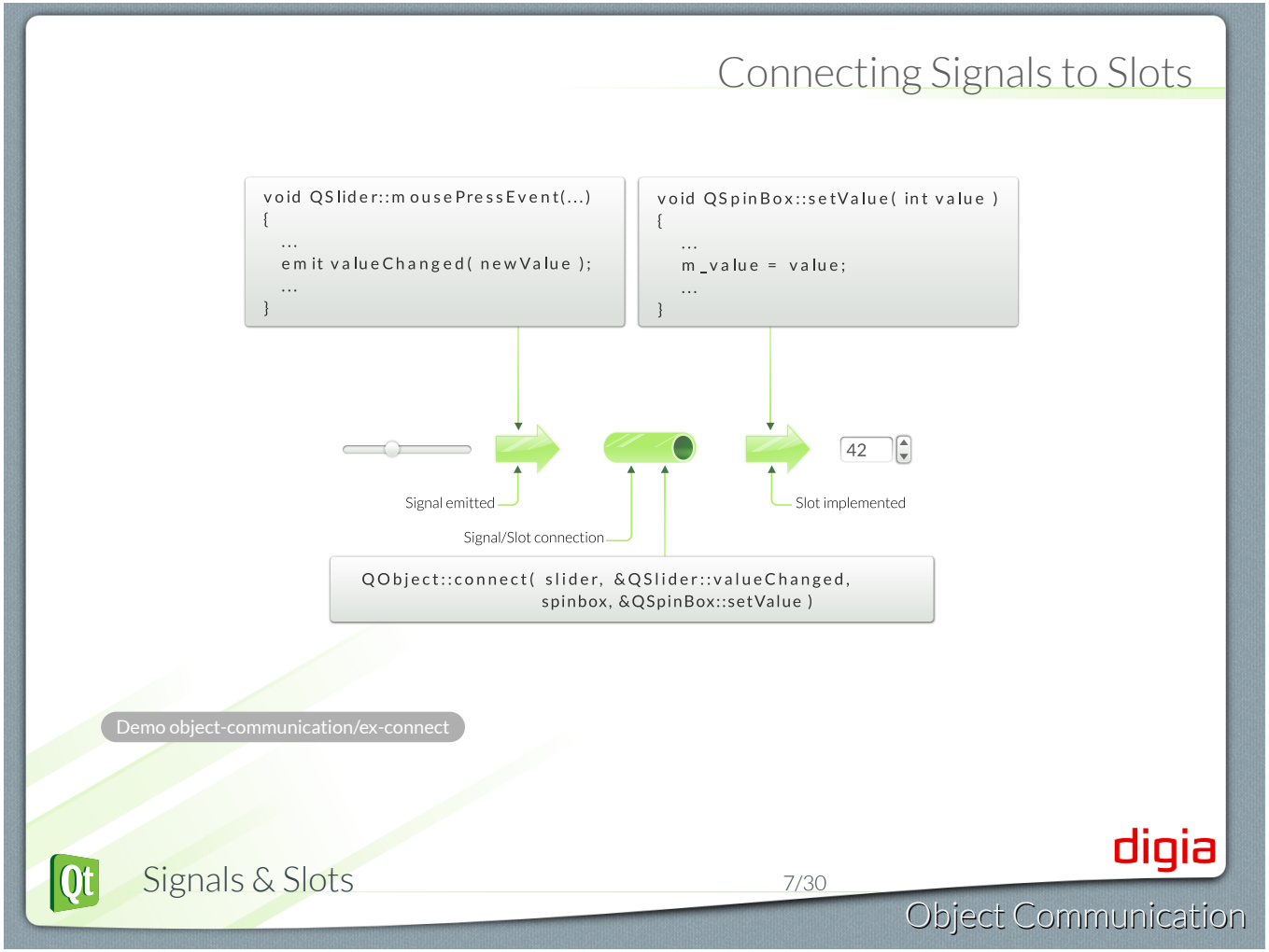
Signals and slots



Signals and slots



Signals and slots



What is a signal?

- A signal is defined in the signals section

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

- Always return `void`

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

- Always return `void`
- No need to be implemented

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

- Always return `void`
- No need to be implemented
- Can be connected to multiple slots

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

- Always return `void`
- No need to be implemented
- Can be connected to multiple slots : slots are activated in arbitrary order

What is a signal?

- A signal is defined in the signals section

```
1 signals:
2     void aSignal();
3
```

- Always return `void`
- No need to be implemented
- Can be connected to multiple slots : slots are activated in arbitrary order
- A signal is emitted using the `emit` keyword

```
1     emit aSignal();
2
```

What is a slot?

- Slots are normal C++ functions and can be called normally;

What is a slot?

- Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.

What is a slot?

- Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.
  - It is implemented as an ordinary method
  - It can be called as an ordinary method

What is a slot?

- Slots are normal C++ functions and can be called normally; their only special feature is that signals can be connected to them.
  - It is implemented as an ordinary method
  - It can be called as an ordinary method
- A slot can return values, but not through connections

Connect

Qt 4 style

```
1 connect(slider, SIGNAL(valueChanged(int)),
2         spinbox, SLOT(setValue(int))
3
```

Connect

Qt 4 style

```
1 connect(slider, SIGNAL(valueChanged(int)),
2         spinbox, SLOT(setValue(int))
3
```

Qt 5 style

```
1 connect(slider, &QSlider::valueChanged,
2         spinbox, &QSpinbox::setValue)
3
```

Connect

Qt 4 style

```
1 connect(slider, SIGNAL(valueChanged(int)),
2         spinbox, SLOT(setValue(int))
3
```

String-based syntax

Qt 5 style

```
1 connect(slider, &QSlider::valueChanged,
2         spinbox, &QSpinbox::setValue)
3
```

Connect

Qt 4 style

```
1 connect(slider, SIGNAL(valueChanged(int)),
2         spinbox, SLOT(setValue(int))
3
```

String-based syntax

Qt 5 style

```
1 connect(slider, &QSlider::valueChanged,
2         spinbox, &QSpinbox::setValue)
3
```

Functor-based syntax

Some basic rules about connection

Variations of Signal/Slot Connections

Signal(s)	Connect to	Slot(s)
one	√	many
many	√	one
one	√	another signal

- Signal to Signal connection

```
connect(bt, SIGNAL(clicked()), this, SIGNAL(okSignal()));
```

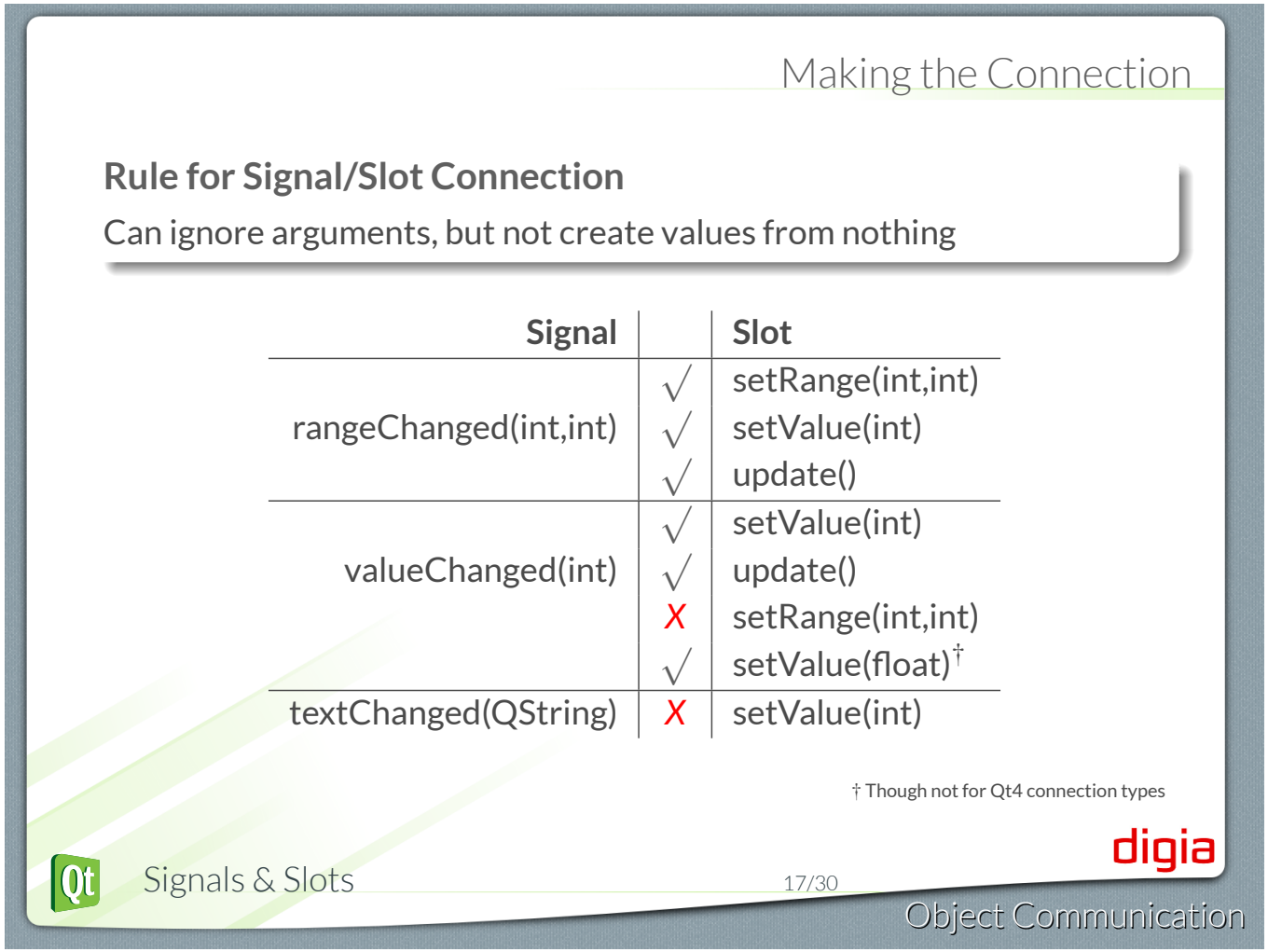
- **Not** allowed to name parameters

connect(m_slider, SIGNAL(valueChanged(int ~~value~~))
this, SLOT(setValue(int ~~newValue~~)))

Signals & Slots 16/30 digia

Object Communication

Some basic rules about connection



Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	update()
valueChanged(int)	√	setValue(int)
	√	update()
	X	setRange(int,int)
	√	setValue(float)†
textChanged(QString)	X	setValue(int)

† Though not for Qt4 connection types

Signals & Slots 17/30 Object Communication

Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.

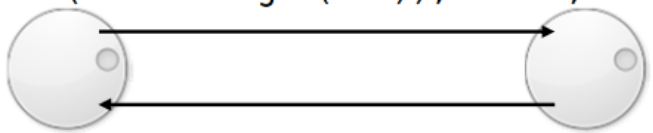
Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.
- We can definitely reversely connect the objects.

Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.
- We can definitely reversely connect the objects.

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```

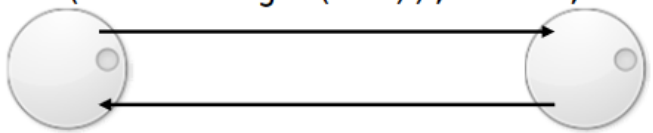


```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.
- We can definitely reversely connect the objects.

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```



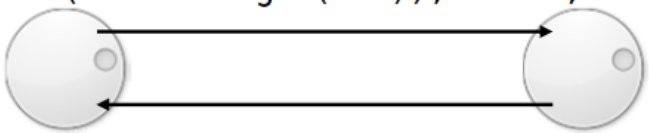
```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

- What will be the potential issue for doing this?
  - Infinite loop!

Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.
- We can definitely reversely connect the objects.

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```



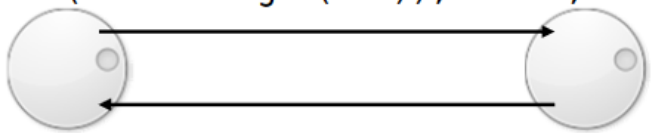
```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

- What will be the potential issue for doing this?
  - Infinite loop! no signal should be emitted unless an actual change takes place.

Synchronizing values

- Sometimes, we want the slide bar syncs with the spinbox as well.
- We can definitely reversely connect the objects.

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```



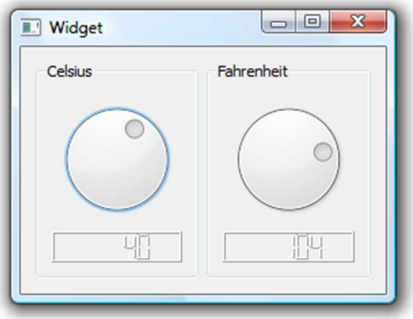
```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

- What will be the potential issue for doing this?
  - Infinite loop! no signal should be emitted unless an actual change takes place.

```
1 void QDial::setValue(int v)
2 {
3     if(v==m_value)
4         return;
5     ...
6
```

Synchronization: example

Temperature Converter



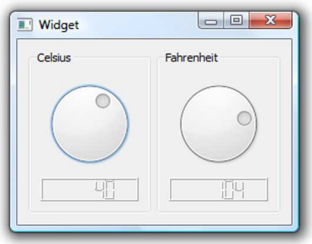
- Uses the `TempConverter` class to convert between Celsius and Fahrenheit
- Emits signals when temperature changes

digia

Synchronization: example

Temperature Converter

- The dialog window contains the following objects
  - A `TempConverter` instance
  - Two `QGroupBox` widgets, each containing
    - A `QDial` widget
    - A `QLCDNumber` widget



digia

Synchronization: example

Temperature Converter

```
class TempConverter : public QObject
{
    Q_OBJECT

public:
    TempConverter(int tempCelsius, QObject *parent = 0);

    int tempCelsius() const;
    int tempFahrenheit() const;

public slots:
    void setTempCelsius(int);
    void setTempFahrenheit(int);

signals:
    void tempCelsiusChanged(int);
    void tempFahrenheitChanged(int);

private:
    int m_tempCelsius;
};
```

QObject as parent

Q_OBJECT macro first

parent pointer

Read and write methods

Emitted on changes of the temperature

Internal representation in integer Celsius.

digia

Synchronization: example

Temperature Converter

- The `setTempCelsius` slot:

```
void TempConverter::setTempCelsius(int tempCelsius)
{
    if(m_tempCelsius == tempCelsius)
        return;

    m_tempCelsius = tempCelsius;

    emit tempCelsiusChanged(m_tempCelsius);
    emit tempFahrenheitChanged(tempFahrenheit());
}
```

Test for change to break recursion

Update object state

Emit signal(s) reflecting changes

- The `setTempFahrenheit` slot:

```
void TempConverter::setTempFahrenheit(int tempFahrenheit)
{
    int tempCelsius = (5.0/9.0)*(tempFahrenheit-32);
    setTempCelsius(tempCelsius);
}
```

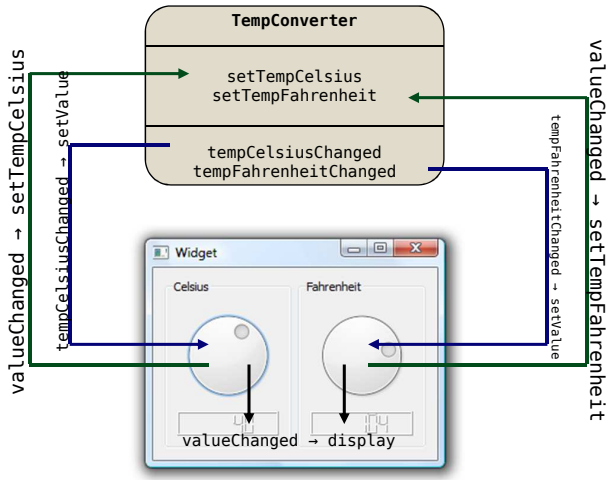
Convert and pass on as Celsius is the internal representation

digia

Synchronization: example

Temperature Converter

- The dials are interconnected through the `TempConverter`
- The LCD displays are driven directly from the dials



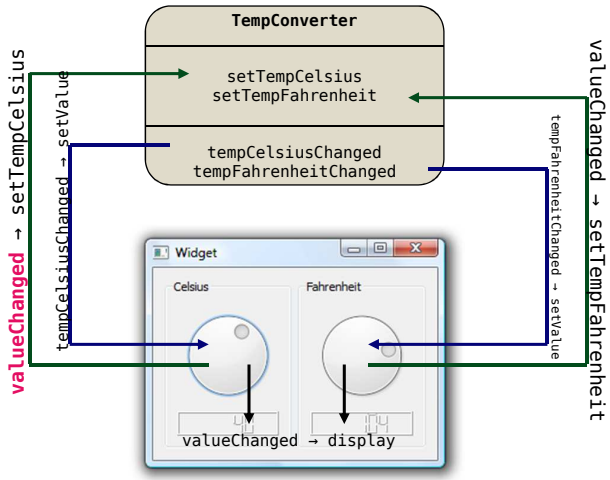

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



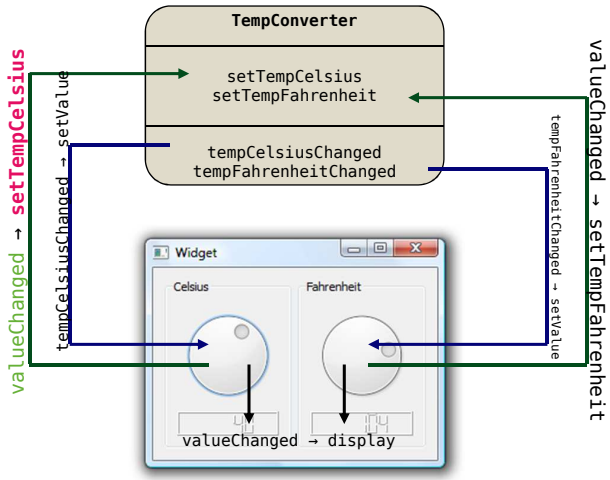

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



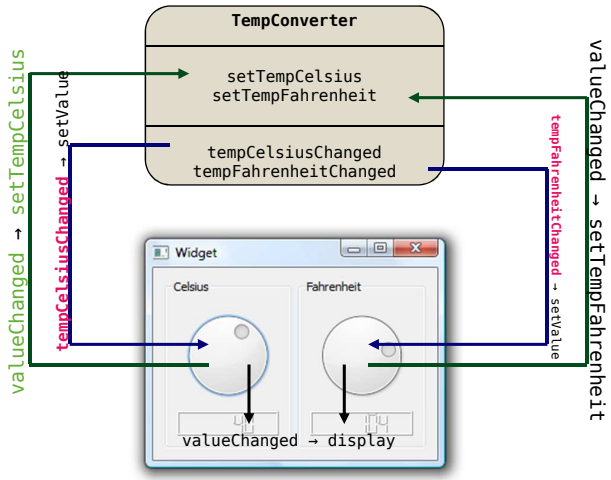

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



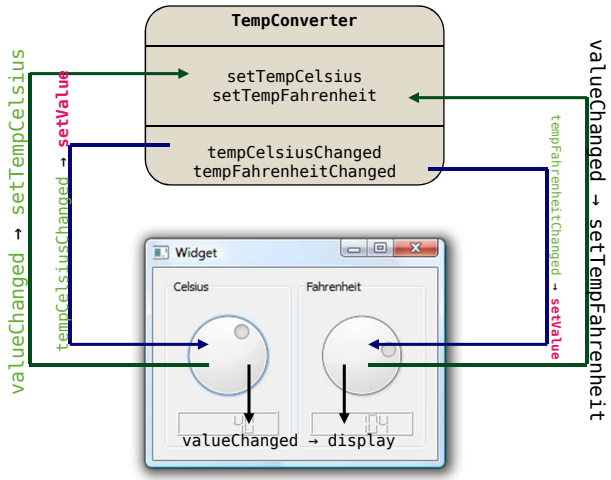

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



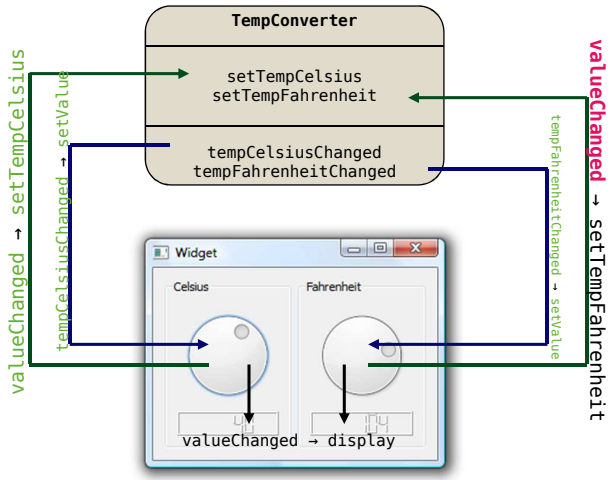

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



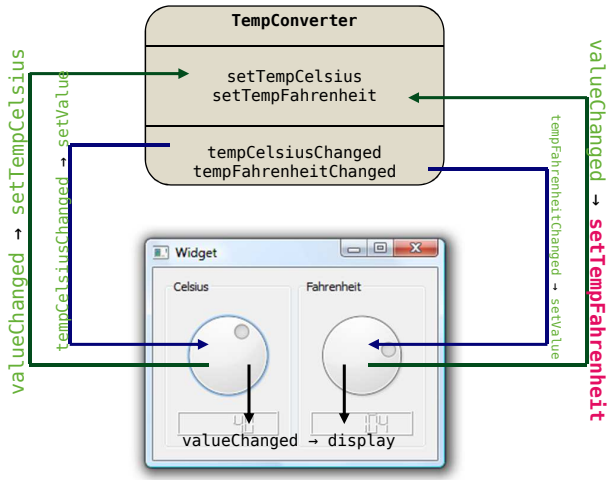

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial



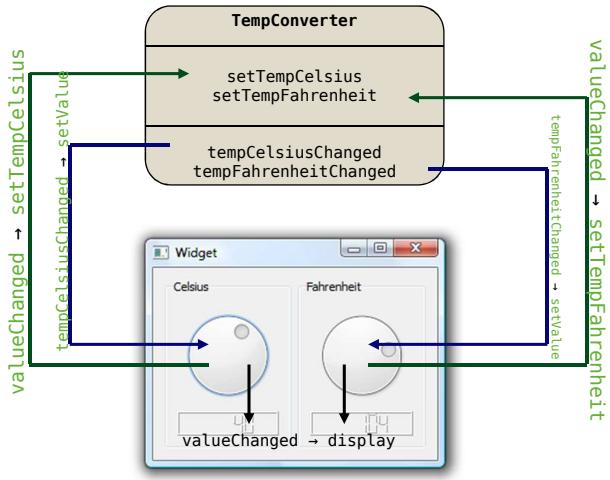

```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Synchronization: example

Temperature Converter

- The user moves the celsiusDial




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

digia

Discussion

- Signals and slots are extremely powerful

Discussion

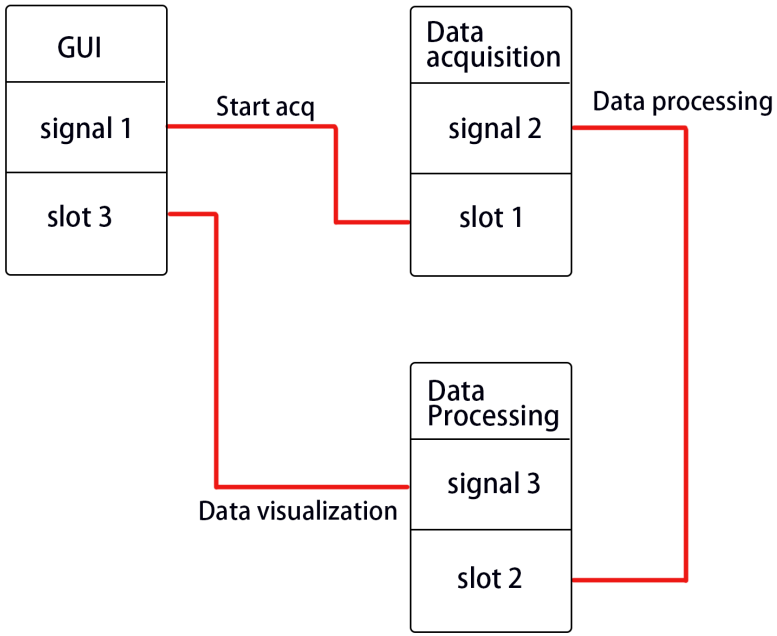
- Signals and slots are extremely powerful
- What else can they achieve?

Discussion

- Signals and slots are extremely powerful
- What else can they achieve?
  - For any interface!

Discussion

- Signals and slots are extremely powerful
- What else can they achieve?
  - For any interface!



What is Qt?

Qt (pronounced "cute") is a cross platform developmentframework written in C++.

What is Qt?

Qt (pronounced "cute") is a cross platform developmentframework written in C++.

- Popular for cross-platform/embedded application development

What is Qt?

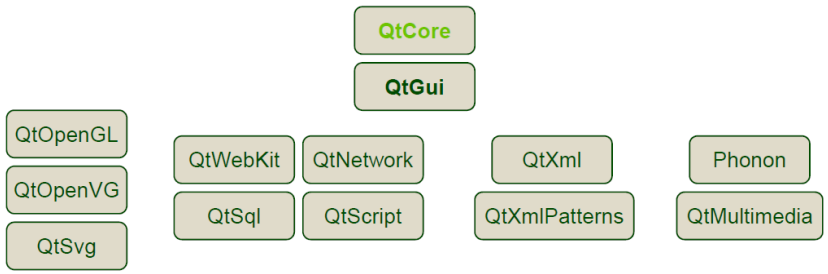
Qt (pronounced "cute") is a cross platform developmentframework written in C++.

- Popular for cross-platform/embedded application development
- Originally for user interfaces now for everything
- Databases, XML, WebKit, multimedia, networking, OpenGL, scripting, non-GUI ...

What is Qt?

Qt (pronounced "cute") is a cross platform developmentframework written in C++.

- Popular for cross-platform/embedded application development
- Originally for user interfaces now for everything
- Databases, XML, WebKit, multimedia, networking, OpenGL, scripting, non-GUI ...
- Qt is made up of modules

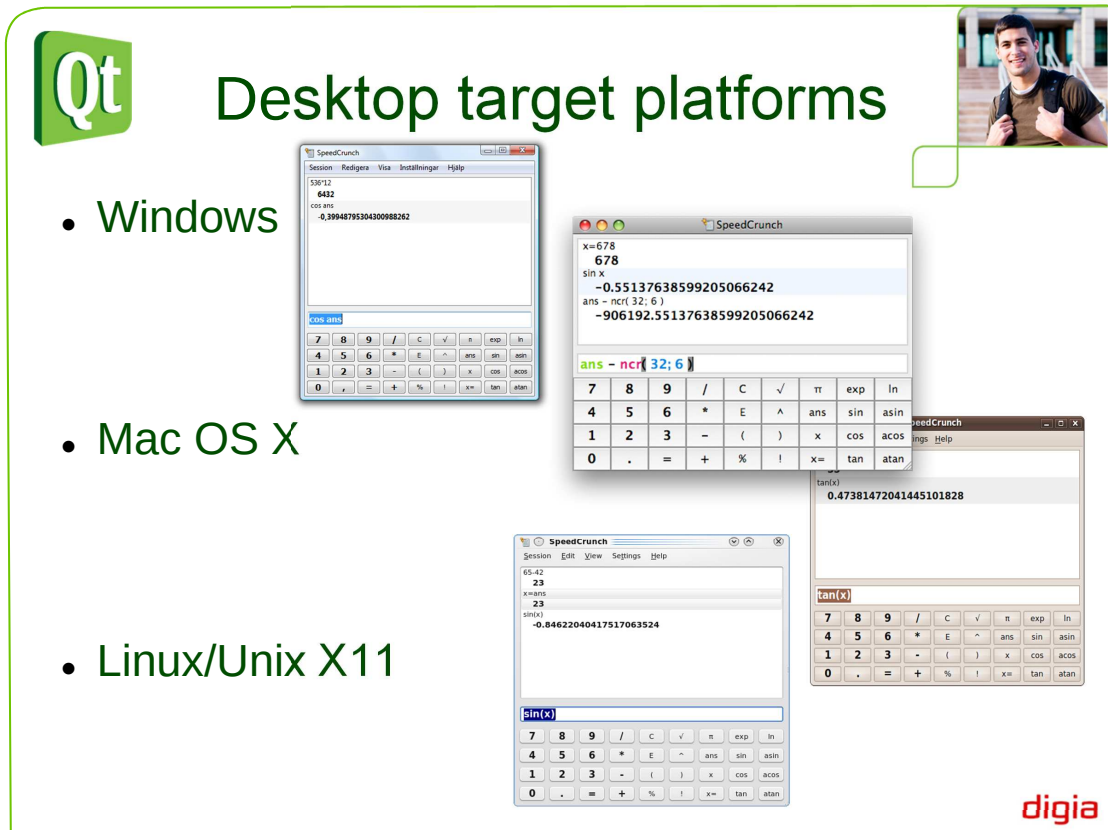


The purpose of Qt

- Cross platform applications built from one source

The purpose of Qt

- Cross platform applications built from one source
- Builds native applications with native look and feel



Qt is used everywhere

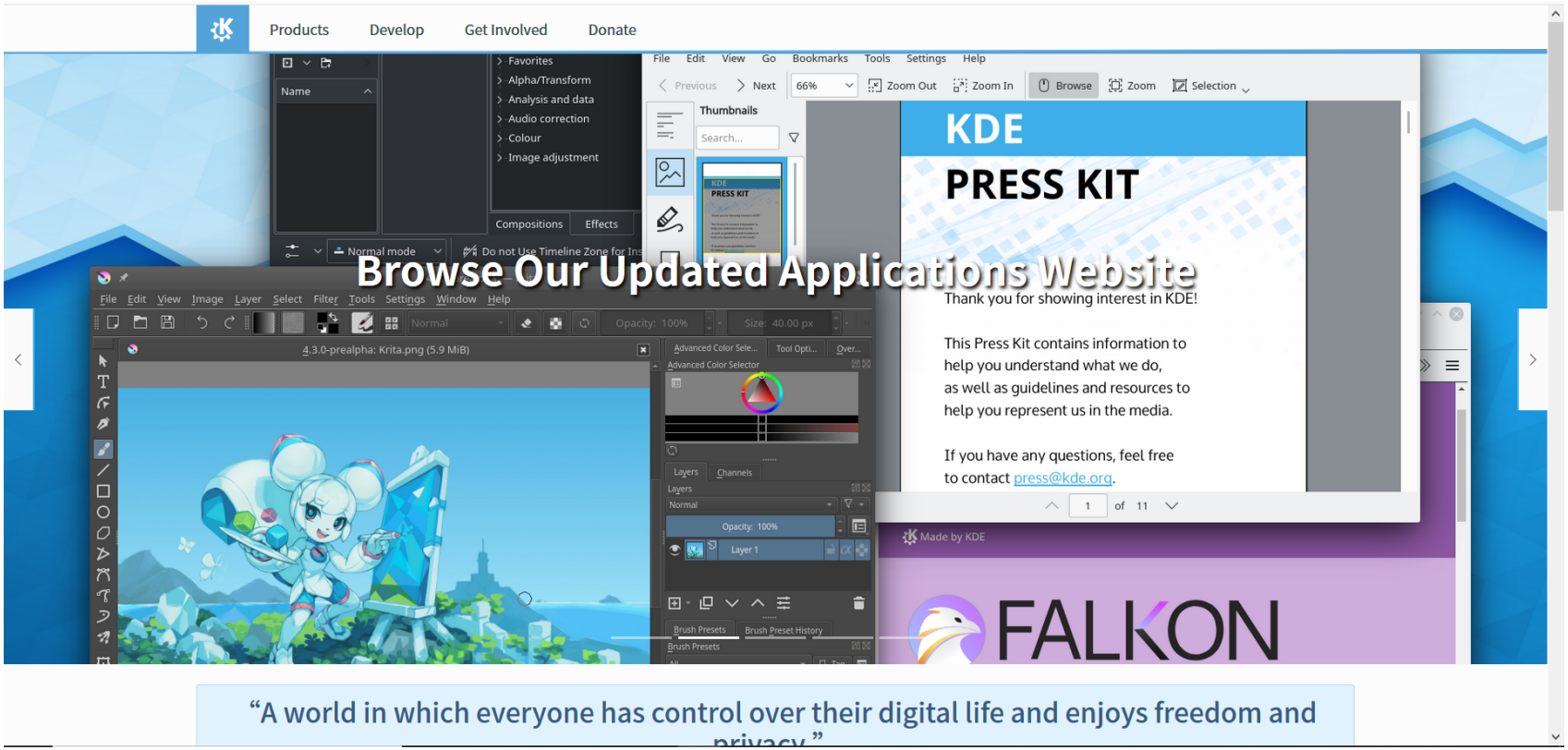
From embedded devices to desktop applications

By companies from many industries

AMD, Google, Canon, Alcatel·Lucent, BT, DAIMLER, EPSON, esa, freescale semiconductor, HITACHI Inspire the Next, GENERAL DYNAMICS C4 Systems, Mentor Graphics, PGS, HONDA The Power of Dreams, MITSUBISHI HEAVY INDUSTRIES, LTD., NORTEL NETWORKS, ORACLE, PHILIPS, Pioneer, skype, Raytheon, Schlumberger, ROHDE&SCHWARZ, SCANIA, SENNHEISER, SHARP, xerox, VOLVO, vodafone, THOMSON REUTERS

KDE

KDE


Products
Develop
Get Involved
Donate
Browse Our Updated Applications Website
KDE
PRESS KIT
Thank you for showing interest in KDE!
This Press Kit contains information to help you understand what we do, as well as guidelines and resources to help you represent us in the media.
If you have any questions, feel free to contact press@kde.org.
Made by KDE
FALKON
"A world in which everyone has control over their digital life and enjoys freedom and privacy"

Conclusion

- In this lecture, we have discussed the underlying mechanism (signals and slots) of Qt

Conclusion

- In this lecture, we have discussed the underlying mechanism (signals and slots) of Qt
- Please follow the walkthrough posted online and install Qt environment on your computer