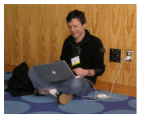
graphical user interfaces in Qt

Eric Lecolinet - Télécom ParisTech

www.telecom-parist.fr/~elc

Oct. 2023

IGR and IGD



Publications
Videos
Software
CV
HDR

DIVA HCI Group
Digiscope
Fab Lab

LTCI Lab.
Télécom Paris
Infres Department

AFIHM
SIGCHI Paris
ACM SIGCHI

Contact
first.last@telecom-paris.fr

URL
http://www.telecom-paris.fr/~elc

Address
19 place Marguerite Perey
F-91120 Palaiseau

Eric Lecolinet

Professor
DIVA HCI Group - LTCI Lab.
INFRES Department - Télécom Paris - Institut Polytechnique de Paris (IP Paris)

Courses

- **Master Interaction, Graphics & Design (IGD)** of Institut Polytechnique de Paris
  - Presentation and Application
  - Courses and Schedule
- **Filière 3D et Systèmes Interactifs (IGR)**
- UE IGR201 : Développement d'applications interactives 2D, 3D, Mobile et Web
- UE IGR203 : Interaction Homme-Machine
- UE INF224 : Paradigmes de Programmation
- UE INF720 : Langage C et Programmation
- **Course material** : Interaction Homme-Machine - Toolkit Qt - Programmation, C++, Swing - Langage C

Research

RECENT ACTIVITIES

- Head of the **DIVA HCI Group** (Design, Interaction, Visualisation & Applications)
- President of **AFIHM** (Association Francophone d'Interaction Homme-Machine) and former member of the **CPPMS** (IHM steering committee)
- Member of the board of the **LTCI** Laboratory (Laboratoire de Traitement et Communication de l'Information)
- Responsible of the **Digiscope** EquipeX for Télécom-Paristech and scientific responsible of the **Télécom FabLab**
- Reviewer for CHI, TOCHI, UIST, Interact, PMC, IHCI, IHM, ErgoIHM...

PUBLICATIONS & VIDEOS

- **Publications** - Google Scholar
- Videos - Talk at FITG'12

POST DOCS & PhD STUDENTS

- **Zhuoming Zhang** (PhD, co-sup. with F. Detienne)
- **Marc Teyssier** (PhD, co-sup. with G. Bailly and C. Pelachaud)

http://www.telecom-paristech.fr/~elc/

perso.telecom-paristech.fr/~elc/igr/

Filière IGR
3D et Systèmes Interactifs

IGR • igr201 • igr202 • igr203 • inf224 • elc

Responsables : Tamy Boubekeur et Eric Lecolinet - Télécom ParisTech, Université Paris-Saclay
URL : http://www.telecom-paristech.fr/~elc/igr/

Descriptif

Cette filière vise à donner aux élèves une formation complète dans les domaines de l'interaction homme-machine et de l'informatique graphique 3D. Elle prépare les futurs ingénieurs à la conception de systèmes interactifs avancés en leur donnant les bases informatiques et mathématiques nécessaires à la modélisation numérique de ces systèmes.

Parmi les débouchés naturels de cette filière, on peut citer : la conception assistée par ordinateur (CAO), les jeux vidéo, les effets spéciaux, les applications mobiles, la simulation, le design d'interaction, la réalité virtuelle et la visualisation.

Cette filière prépare en outre aux métiers scientifiques liés à la recherche en IHM ou en informatique graphique 3D en proposant la possibilité de suivre ensuite un Master 2 spécialisé dans l'un de ces deux domaines (Master Interaction/HCI ou MVA de l'Université Paris-Saclay ou IMA de l'UPMC).

Voir aussi les **transparents de présentation de la filière**

Seconde année

- **Période 1 :** UE IGR201 : Développement d'applications interactives 2D, 3D, Mobile et Web
- **Période 2 :** UE IGR202 : Informatique Graphique 3D et Réalité Virtuelle
- **Période 3 :** UE IGR203 : Interaction Homme-Machine
- **Période 4 :** UE IGR204 : Visualisation (en anglais) et UE IGR205 : Séminaire de projet

Voir aussi la page Synapses de la Filière IGR et de ses UEs.

IGD - Interaction, Graphics & Design Track
Master of Computer Science - IP Paris

Index • All Courses • Interaction • Design • Mixed Reality • Graphics • Multimodal/Robotics • Programming • Projects • Schedule

Presentation

The **IGD Master** track is part of the Computer Science Master of Institut Polytechnique de Paris (IP Paris)

Please see the IGD IP Paris page for a presentation and for application.

Welcome Session

The **Welcome Session** will take place on Monday September 7, 14:00-16:00 at Télécom Paris (room 0A214).

Courses & Schedules

- Full list of courses
- Overall schedule
- The list of choosen courses must first be validated by the student's tutor (see *Policies & Procedures* below).
- Course registration should be done in the institute offering the desired course (e.g. Télécom Paris for TP-xxx courses).

Institutional partners

The IGD Master offers a comprehensive range of courses which are held by the following institutes:

- TP = Télécom Paris
- X = Ecole Polytechnique
- TSP = Télécom SudParis
- ENSTA = Ecole Nationale Supérieure des Techniques Avancées
- HCI = Master HCI - Université Paris-Saclay

All these institutes are located on *Plateau de Saclay*. Maps: IP Paris - Paris-Saclay

Policies & Procedures

IGR201b

- **Android**
- **Qt GUI toolkit**
- **Web interface (basics)**

Cross-platform GUI Toolkits

- Java AWT / Java **Swing** / **JavaFX**
- **Qt**
- GTK, GTK#
- wxWidgets
- FLTK ("light" toolkit)
- Unity (Games)
- Chromium Embedded Framework (Web to app)
- etc. (https://en.wikipedia.org/wiki/List_of_platform-independent_GUI_libraries)

Qt ("cute")

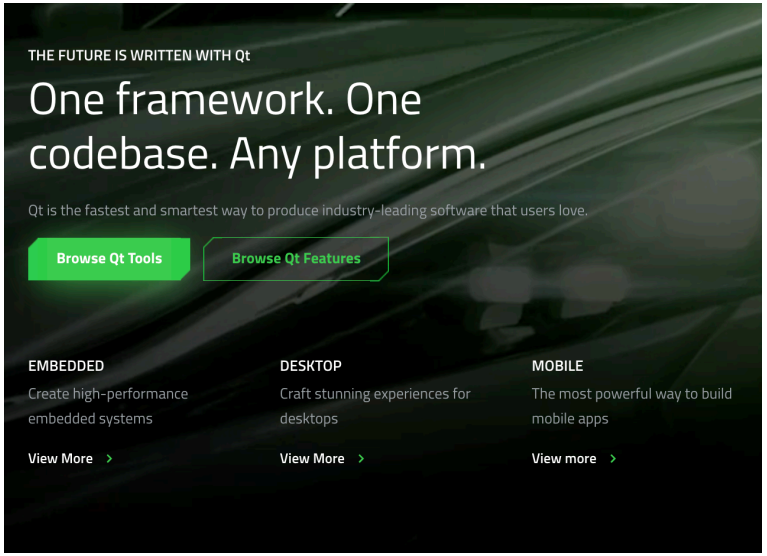
Powerful cross-platform **GUI toolkit**

Supported on various **platforms**

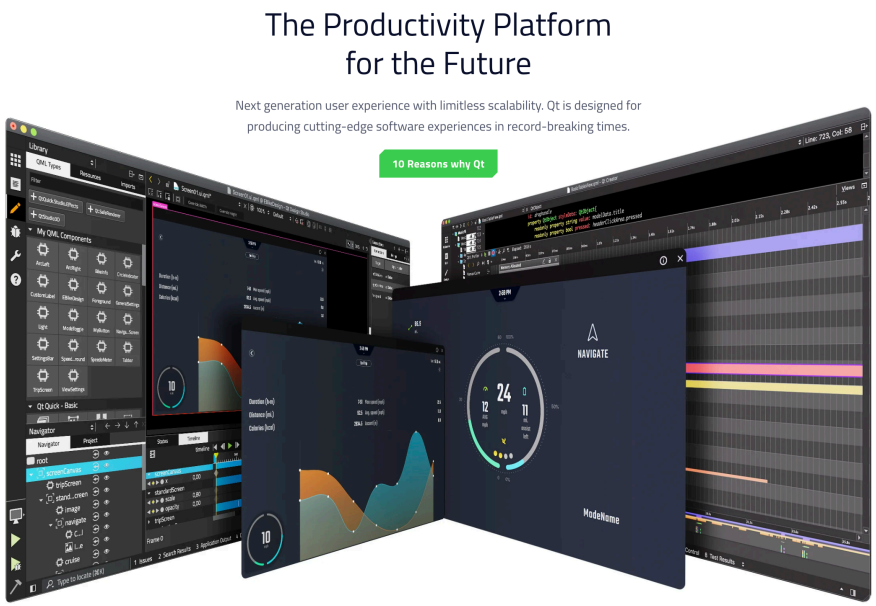
- Desktop / mobile / embedded platforms

C++, bindings in **Python** and other languages

Rich environment with many **tools**



https://www.qt.io/



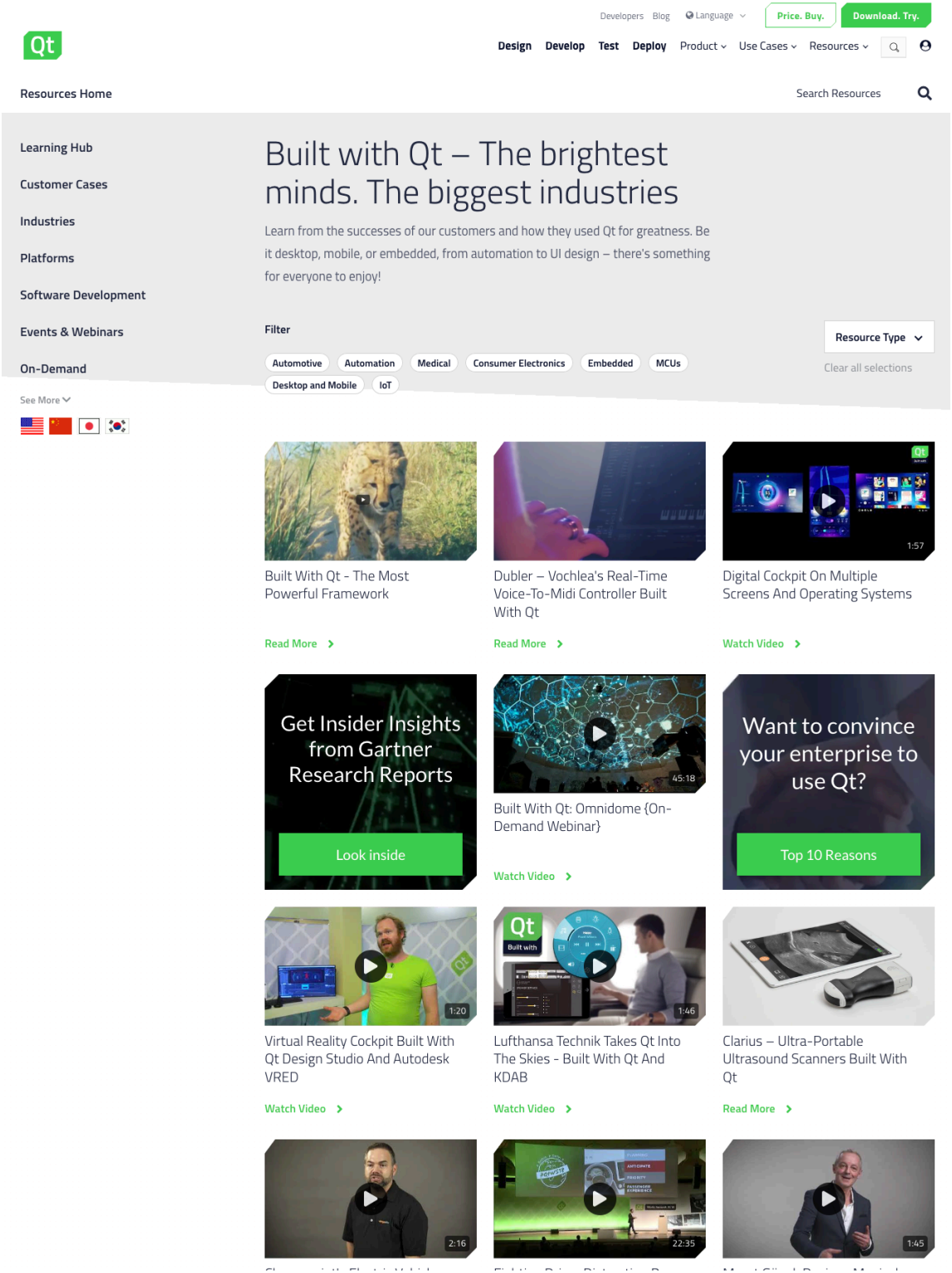
Qt

Licences

- **LGPL** (free):
  - open-source code

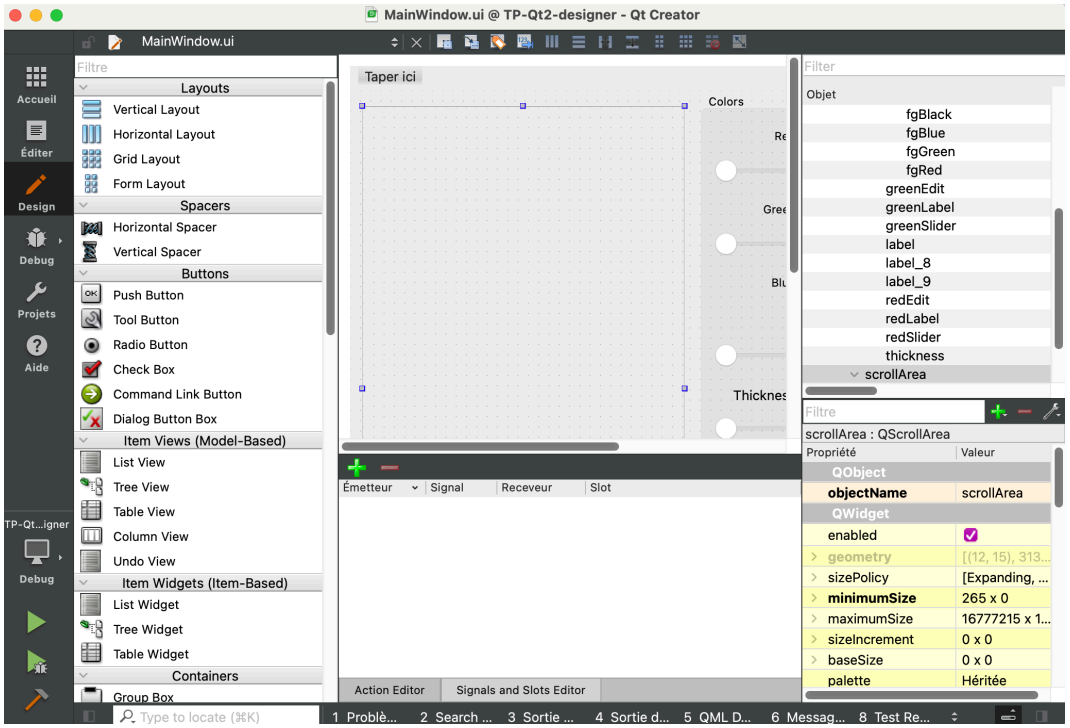
- **Commercial licences** for companies

Many **open-source** and **commercial products** use **Qt**



Tools

- **Design Studio / QtCreator** IDE



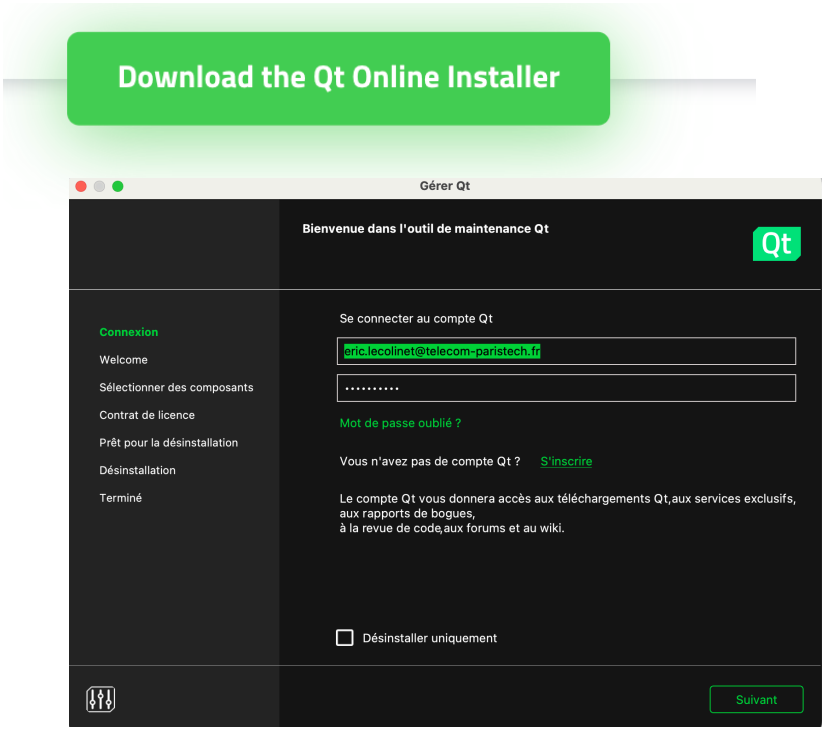
- QtDesigner (interactive builder)
- QtLinguist (internationalization)
- QtAssistant (documentation)
- qmake
- **QtQuick / QML** declarative language (similar to CSS & JSON)

Libs

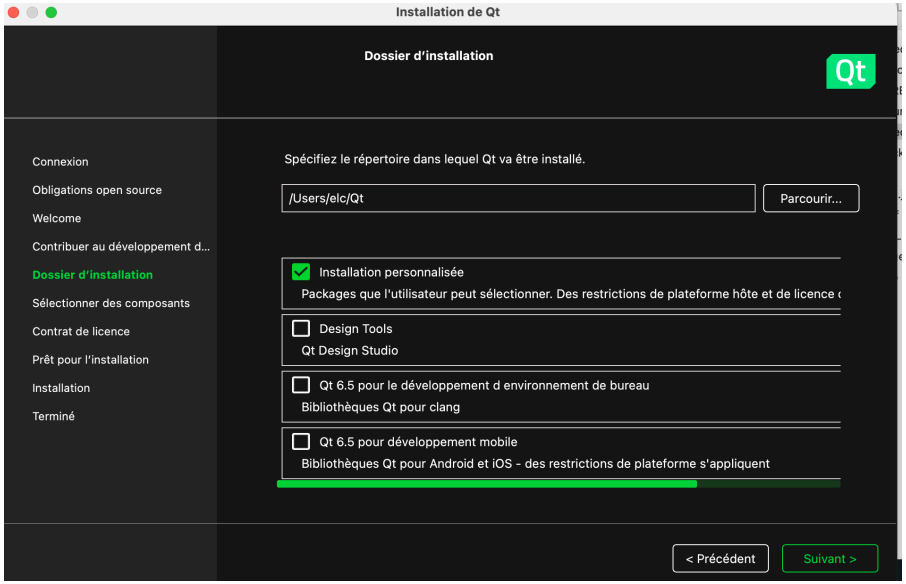
- Qt WebEngine WidgetsThe Qt WebEngine Widgets module provides a web browser engine as well as C++ classe with web content.
- Qt WebKit Widgets The Qt WebKit Widgets module provides a web browser engine as well as C++ classes to ren content.
- Qt3DCore Qt3D Core module contains functionality to support near-realtime simulation systems
- Qt3DInput Qt3D Input module provides classes for handling user input in applications using Qt3D
- Qt3DRenderer Qt3D Renderer module contains functionality to support 2D and 3D rendering using Qt3D
- QtBluetooth Enables basic Bluetooth operations like scanning for devices and connecting them
- QtConcurrent Qt Concurrent module contains functionality to support concurrent execution of program code
- QtCore Provides core non-GUI functionality
- QtDBus Qt D-Bus module is a Unix-only library that you can use to perform Inter-Process Communication using tl
- QtDesigner Provides classes to create your own custom widget plugins for Qt Designer and classes to access Qt
- QtGui Qt GUI module provides the basic enablers for graphical applications written with Qt
- QtHelp Provides classes for integrating online documentation in applications
- QtLocation Provides C++ interfaces to retrieve location and navigational information
- QtMultimedia Qt Multimedia module provides audio, video, radio and camera functionality
- QtNetwork Provides classes to make network programming easier and portable
- QtNfc An API for accessing NFC Forum Tags
- QtOpenGL Qt OpenGL module offers classes that make it easy to use OpenGL in Qt applications
- QtPositioning Positioning module provides positioning information via QML and C++ interfaces
- QtPrintSupport Qt PrintSupport module provides classes to make printing easier and portable
- QtQml C++ API provided by the Qt QML module
- QtQuick Qt Quick module provides classes for embedding Qt Quick in Qt/C++ applications
- QtQuickWidgets C++ API provided by the Qt Quick Widgets module
- QtScript Qt Script module provides classes for making Qt applications scriptable
- QtScriptTools Provides additional components for applications that use Qt Script
- QtSensors Provides classes for reading sensor data
- QtSerialPort List of C++ classes that enable access to a serial port
- QtSql Provides a driver layer, SQL API layer, and a user interface layer for SQL databases
- QtSvg Qt SVG module provides functionality for handling SVG images
- QtTest Provides classes for unit testing Qt applications and libraries
- QtUiTools Provides classes to handle forms created with Qt Designer
- QtWebChannel List of C++ classes that provide the Qt WebChannel functionality
- QtWebSockets List of C++ classes that enable WebSocket-based communication
- QtWidgets Qt Widgets module extends Qt GUI with C++ widget functionality
- QtXml Qt XML module provides C++ implementations of the SAX and DOM standards for XML
- QtXmlPatterns Qt XML Patterns module provides support for XPath, XQuery, XSLT and XML Schema validat

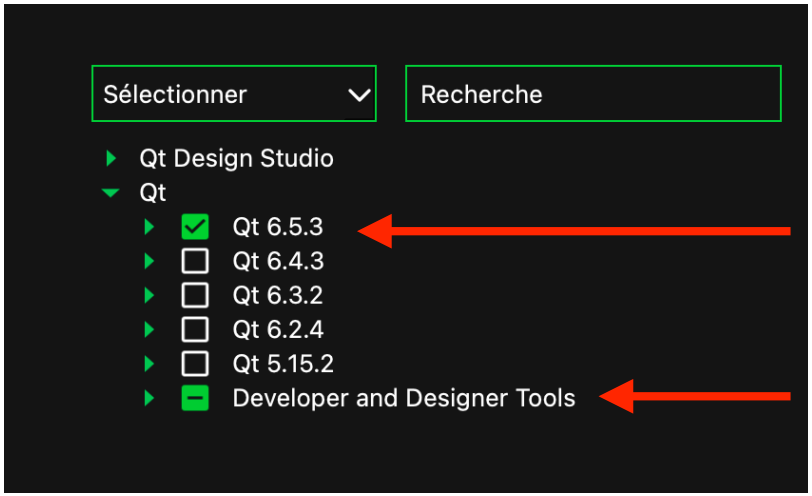
Links & Install

- **Labs**: http://www.telecom-paristech.fr/~elc/qt
- **Qt Web site**: http://www.qt.io/
- **Documentation:** http://doc.qt.io/qt-6/
- **Installation** (Open Source Version):
  - In **https://www.qt.io/download-open-source**
    - *click on "Download the Qt Online Installer"*
    - *download Installer for your OS*
  - Open Qt **MaintenanceTool** app

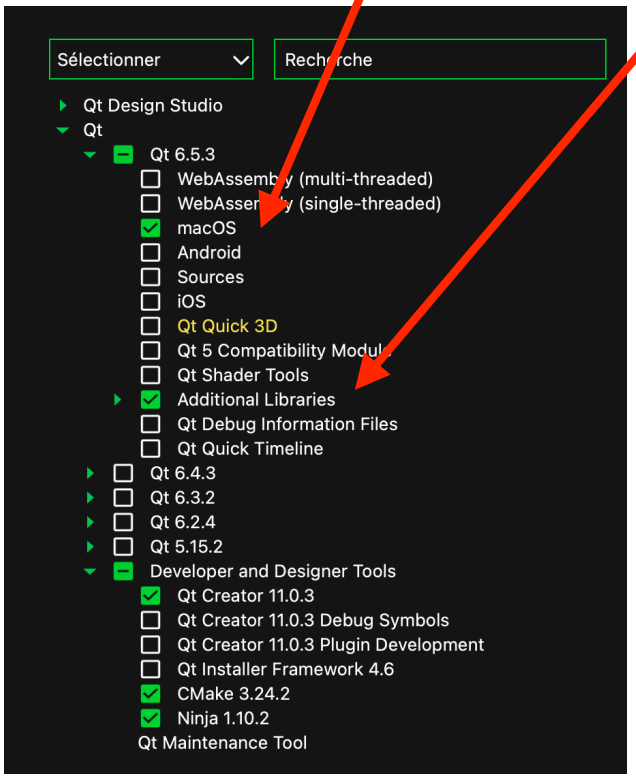


Install (MaintenanceTool)

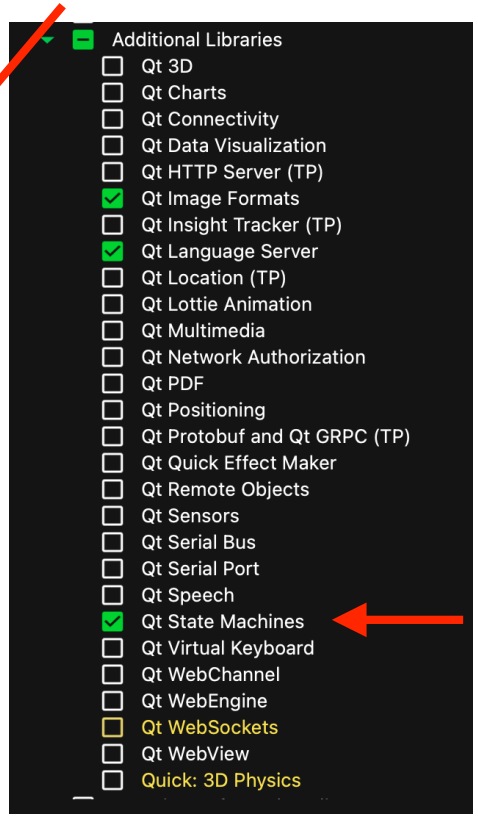




your OS



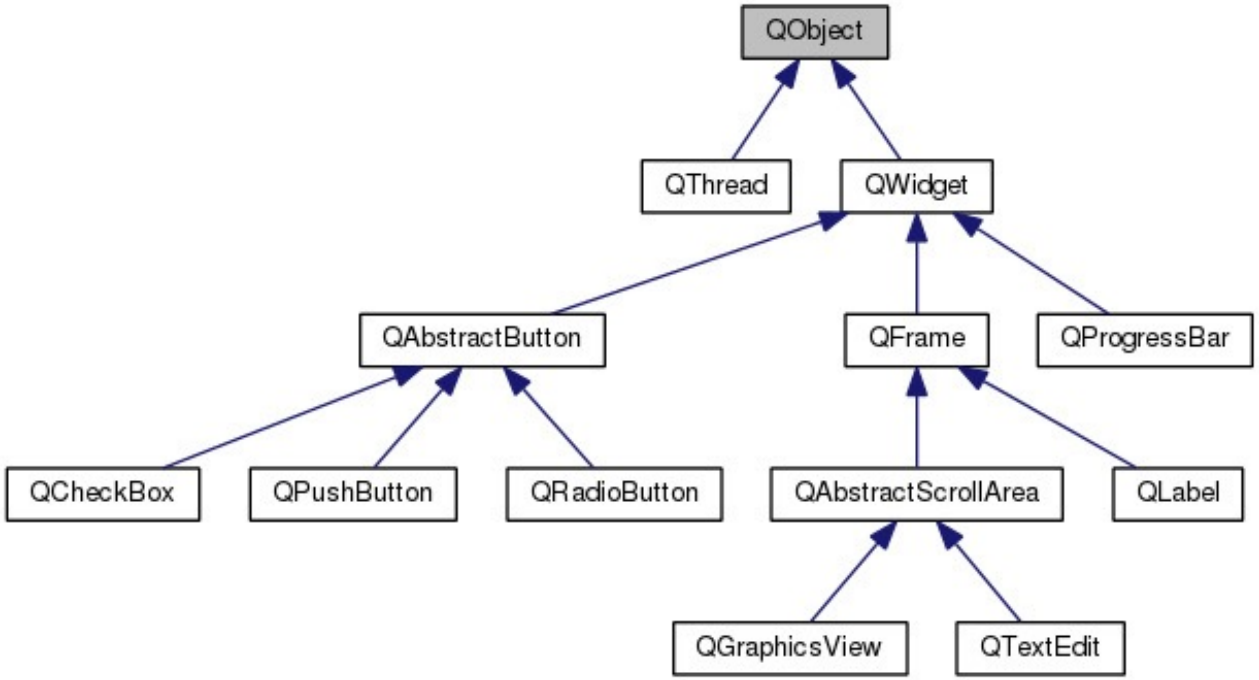
additional libraries



Main widgets

QObject: root class

QWidget: base class of all **graphical objects**



Interactors

QLabel

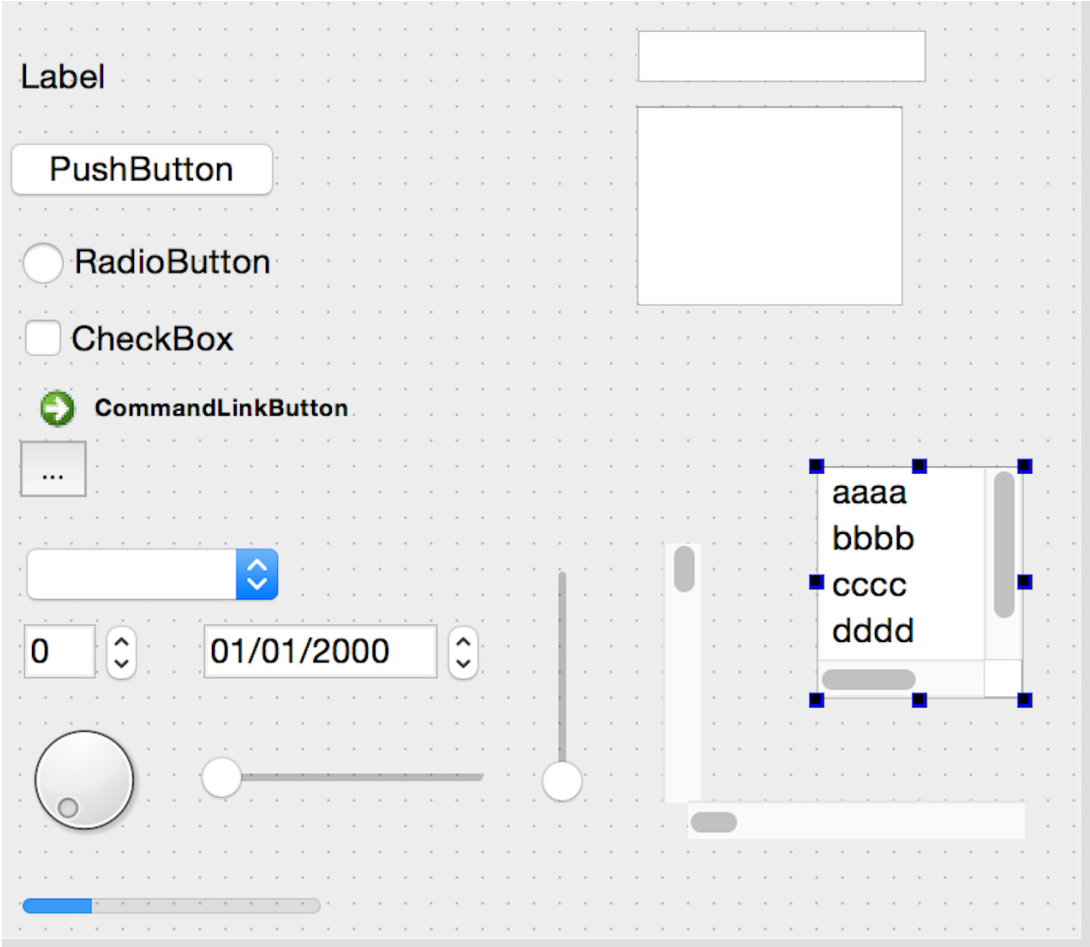
QPushButton

QSlider

QLineEdit

QTextEdit

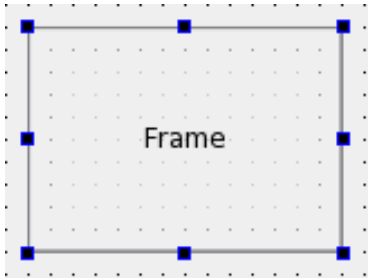
etc.



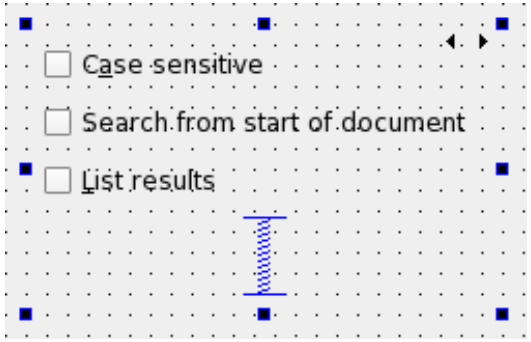
Containers

A **QWidget** can contain **other widgets**

There as also **special purpose containers**

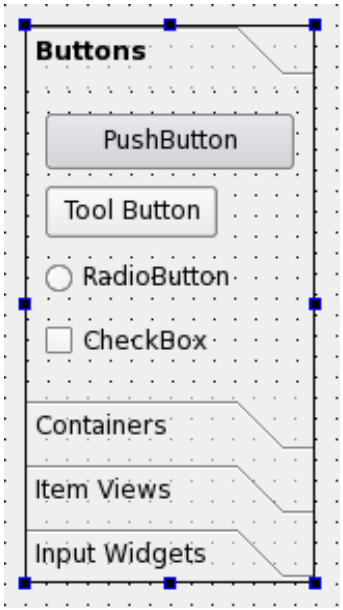


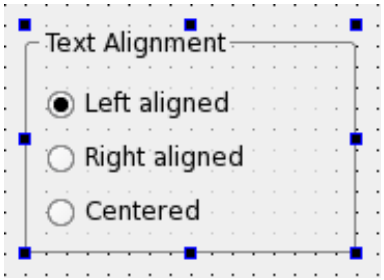
QFrame



QStackedWidget

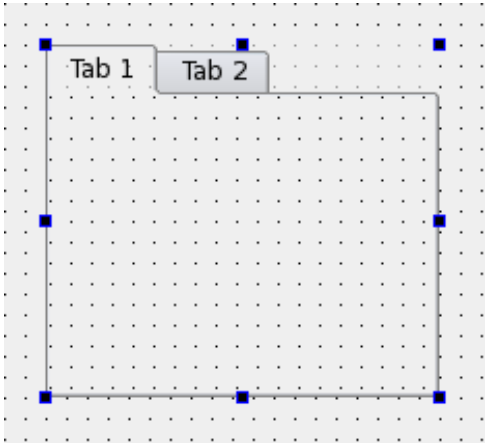
QToolbox



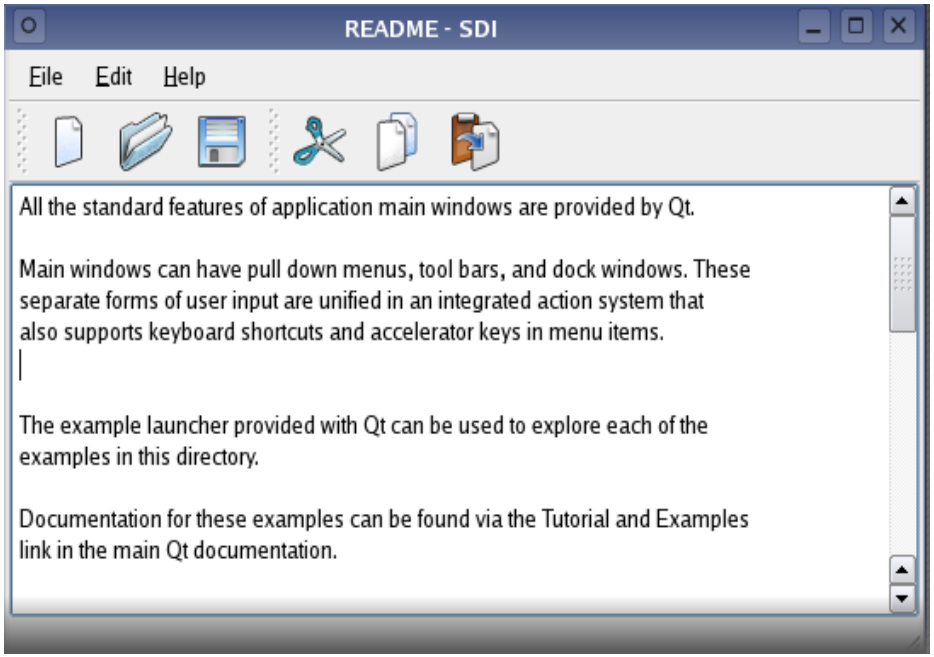


QGroupBox

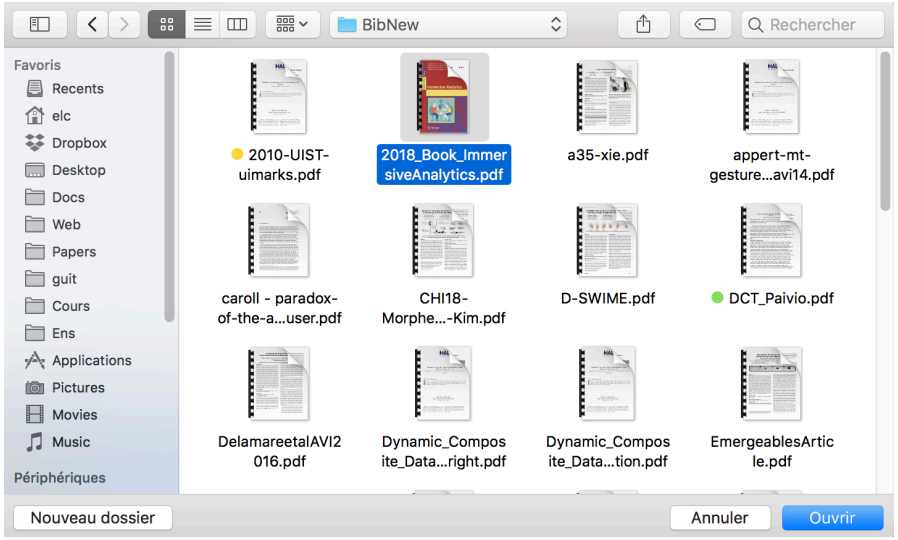
QTabWidget



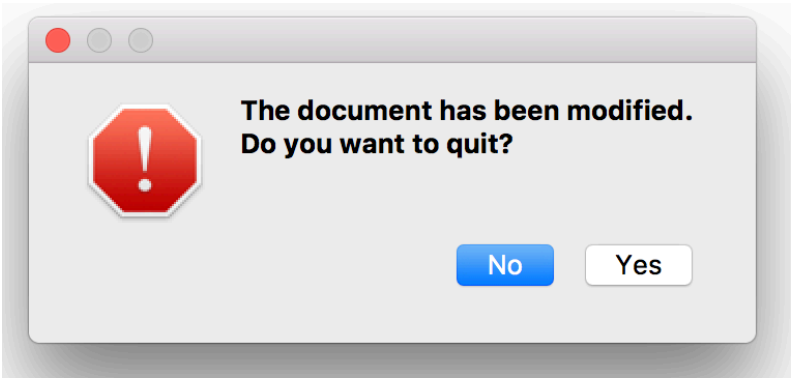
Main window and dialog boxes



QMainWidget



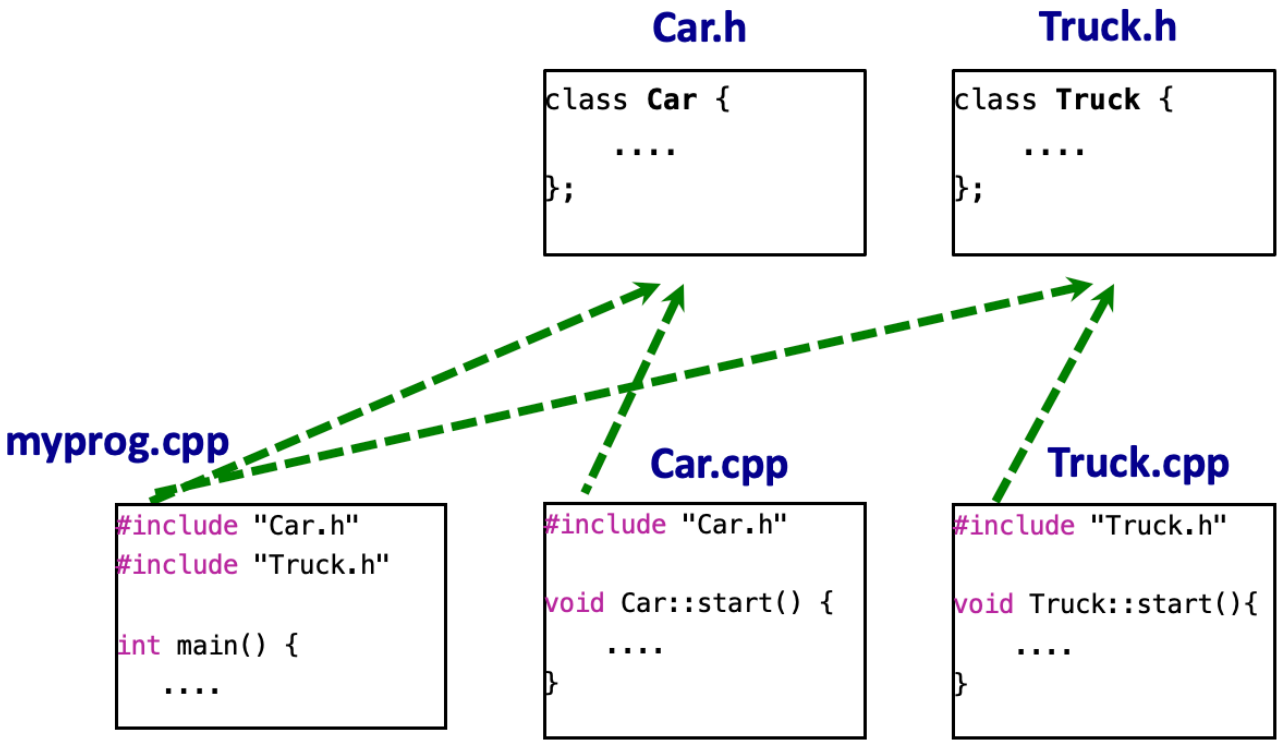
QFileDialog



QMessageBox

C++ vs Java

usually: **header** + **implementation** files



C++ vs Java

header file circle.h

```
class Circle {
private:
    int x{}, y{};
    unsigned int radius = 0;

public:
    Circle(int x, int y, unsigned int radius);

    void setRadius(unsigned int);
    unsigned int getRadius() const;
    unsigned int getArea() const;
    ....
};      // !!!
```

instance variables

{} initializes variables (**unitialized** by default)

constructor(s)

instance methods

C++ vs Java

```
#include "Circle.h"

Circle::Circle(int _x, int _y, unsigned int _r) {
  x = _x;
  y = _y;
  radius = _r;
}

void Circle::setRadius(unsigned int r) {
  radius = r;
}

unsigned int Circle::getRadius() const {
  return radius;
}

unsigned int Circle::getArea() const {
  return 3.1416 * radius * radius;
}
```

implementation file circle.cpp

inserts the content of Circle.h

:: specifies the class

C++ vs Java

MyClass.java

```
class MyClass {
    public void whatever(int i) {
        Zombat z = new Zombat(i);
        z.foo();
    }
}
```

MyClass.h

```
class MyClass {
    public:
        void whatever(int i);
} ;
```

MyClass.cpp

```
void MyClass::whatever(int i) {
    Zombat * z = new Zombat();
    z->foo();
}
```

don't forget **the** *

-> **instead of** **.** for accessing members

C++ vs Java

MyClass2.cpp

```
void MyClass::whatever(int i) {
    Zombat z = new Zombat();
    z.foo();
}
```

Correct?

MyClass3.cpp

```
void MyClass::whatever(int i) {
    Zombat z = Zombat();
    z.foo();
}
```

Correct?

Note: C++ vs Java

MyClass2.cpp

```
void MyClass::whatever(int i) {
    Zombat z = new Zombat();
    z.foo();
};
```

WRONG!

MyClass3.cpp

```
void MyClass::whatever(int i) {
    Zombat z;  // = Zombat();  useless
    z.foo();
};
```

OK but z created in the stack!

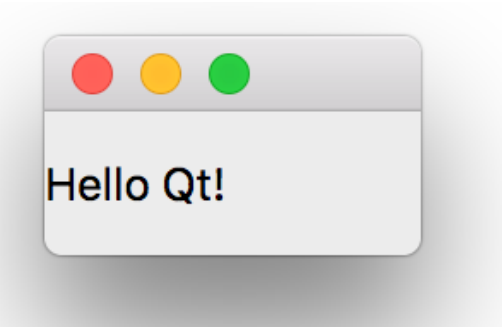
MyClass.cpp

```
void MyClass::whatever(int i) {
    Zombat * z = new Zombat();
    z->foo();
}
```

OK

Hello Word!

```
#include <QApplication>    // headers
#include <QLabel>

int main(int argc, char **argv) {
    QApplication * app = new QApplication(argc, argv);
    QLabel * hello = new QLabel("Hello Qt!");
    hello->show( );
    return app->exec( );
}
```

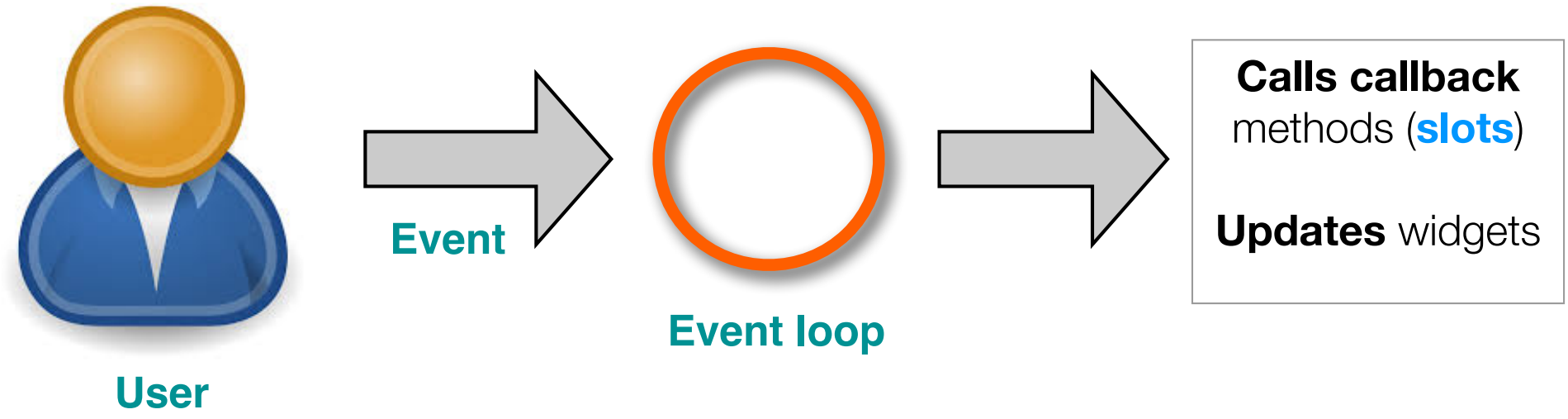


shows the widget

hello becomes the **top-level** widget because it has **no parent**

starts the event loop

Event loop



The event loop

- **Takes control** : **exec**() **blocks** the program
- Indefinitely:
  - **waits** for the next event
  - **processes** the event

```
while (true) {
    event = getNextEvent()
    processEvent(event);
}
```

basic event loop

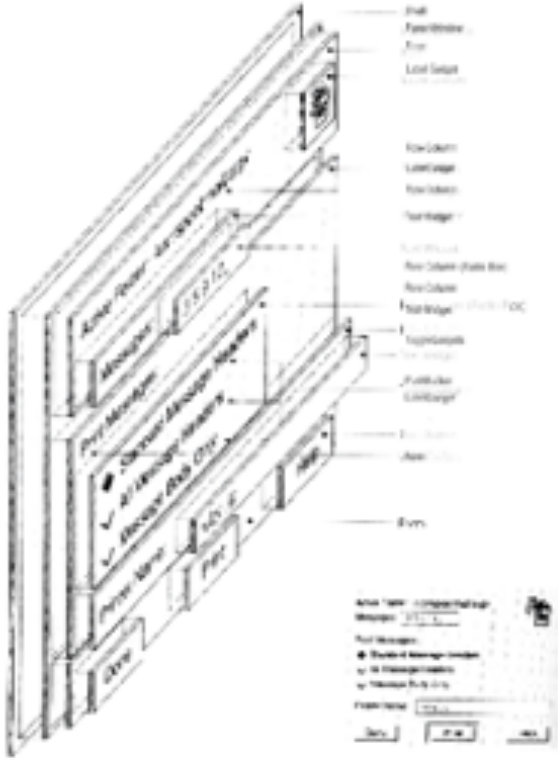
Instance graph

A QWidget can contain QWidgets that contain QWidgets...

= **instance graph** (more precisely it's a tree)

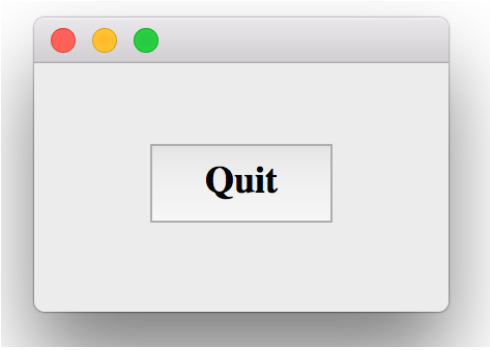
parenthood means:

- **superimposition**:
  children **on top** of parents
- **clipping**:
  children parts outside parent area **not visible**



Instance graph

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main(int argc, char **argv) {
    auto * app = new QApplication(argc, argv);

    auto * box = new QWidget( );
    box->resize(200, 120);

    auto * btn = new QPushButton("Quit", box);      // box is the parent of btn
    btn->resize(100, 50);
    btn->move(50, 35);
    btn->setFont(QFont("Times", 18, QFont::Bold));

    box->show( );
    return app->exec( );
}
```



box

quitBtn

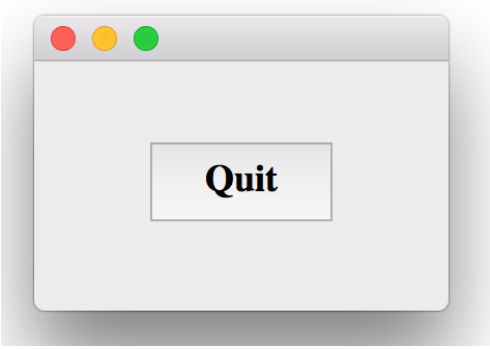
The **parent** is given as an argument of the **constructor**

Except for **top-level** widgets (no parent)

Note the **auto** keyword

Signals and slots

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main( int argc, char **argv ) {
    auto * app = new QApplication(argc, argv);

    auto * box = new QWidget();
    box->resize(200, 120);

    auto * btn = new QPushButton("Quit", box);
    // etc....

    QObject::connect( btn, SIGNAL(clicked( )), app, SLOT(quit( )) );

    box->show( );
    return app->exec( );
}
```



connects a signal to a slot

Signals and slots

A signal is emitted by an object

– e.g. when **clicking** a QPushButton

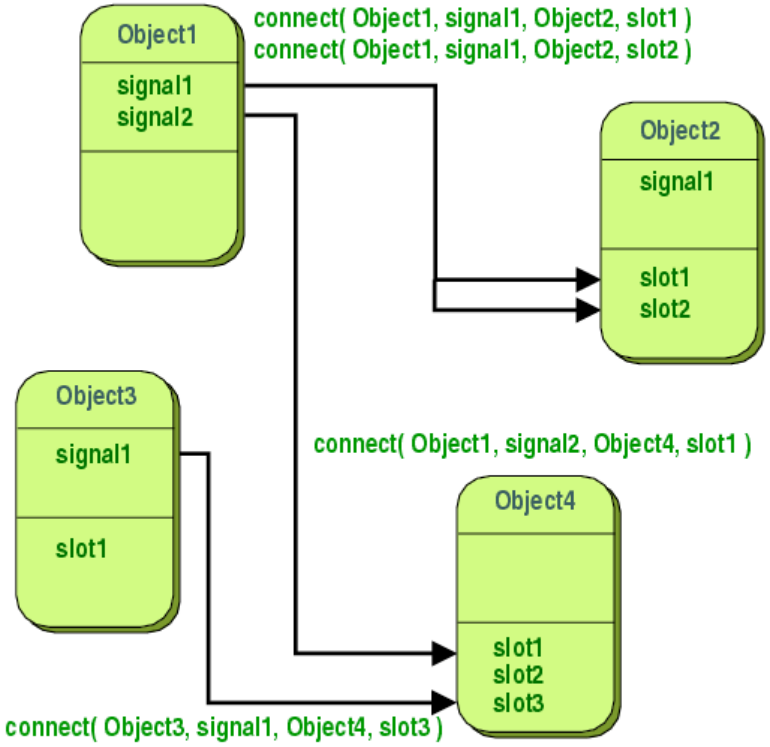
A slot is a method

– that can be **connected** to a signal

When a signal is emitted the slots are executed

– A signal can be connected to **several** slots

– **Several** signals can be connected to a slot

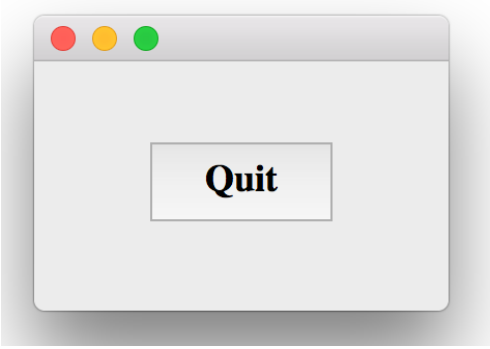


Key aspect of Qt

- differs from **callback functions** or **Java listeners**

Signals and slots

```
#include <QApplication>
#include <QPushButton>
#include <QWidget>
#include <QFont>

int main(int argc, char **argv) {
    auto * app = new QApplication(argc, argv);

    auto * box = new QWidget();
    box->resize(200, 120);

    auto * btn = new QPushButton("Quit", box);
    // etc....

    QObject::connect( btn, SIGNAL(clicked( )), app, SLOT(quit( )) );

    box->show( );
    return app->exec( );
}
```



standard signal
of QPushButton

slot (method)
of QApplication

Declaring a slot

In the header file (.h)

- **MyClass** must **derive** from **QObject, QWidget** or an inherited class
- **keywords Q_OBJECT** and **slots**
  - needed by the **moc pre-compiler**
- **slots** must be **implemented**
  - usually, in the .cpp file

```
class MyClass : public QWidget {
    Q_OBJECT
public:
    MyClass( );
public slots:
    void mySlot( int value );
};
```

Arguments

Signals and slots can have arguments

- which must **correspond** (same order, same types)

```
connect( btn, SIGNAL(clicked( )), app, SLOT(quit( )) );
```

- **stateChanged()** conveys an ***int*** that is received by **mySlot()**:

```
connect( checkBox, SIGNAL(stateChanged(int)), myobj, SLOT(mySlot(int)) );
```

- A slot can have **less** arguments than a signal:

```
connect( checkBox, SIGNAL(stateChanged(int)), app, SLOT(quit( )) );
```

New syntax

Traditional Qt syntax

```
connect( btn, SIGNAL(clicked( )), app, SLOT(quit( )) );
```

New syntax

```
connect( btn, & QPushButton::clicked, app, & QApplication::quit);
```

Relies on standard **C++ method pointers**

Errors checked at **compilation time**

Lambdas

```
void MyClass::foo() {
    int val = 10;
    QObject::connect( btn, & QPushButton::clicked,  [=] { this->setBalance(val); } );
}
```

[=] { *this*->setBalance(*val*); } is a lambda

The **lambda captures** *this* and **local variables**

Notes:

[=] **copies** variables

[&] pass variables by **reference**

allows **changing** their value, **!WATCH: crashes** if they don't exist anymore)

other options are available

Misc.

Disconnects the slot

```
disconnect(from, SIGNAL(balanceChanged(int)), to, SLOT(setBalance(int)) );
```

Propagates the signal

```
connect(from, SIGNAL(balanceChanged(int)), to, SIGNAL(changed(int)) );
```

Compilation

Meta Object Compiler (MOC)

- Signals and slots require **precompilation** by the MOC
- The **MOC** generates **additional** source code
- Do not forget Q_OBJECT (the MOC relies on it)

```
class MyClass : public QWidget {
    Q_OBJECT
public:
    MyClass( );

public slots:
    void mySlot( int value );
};
```

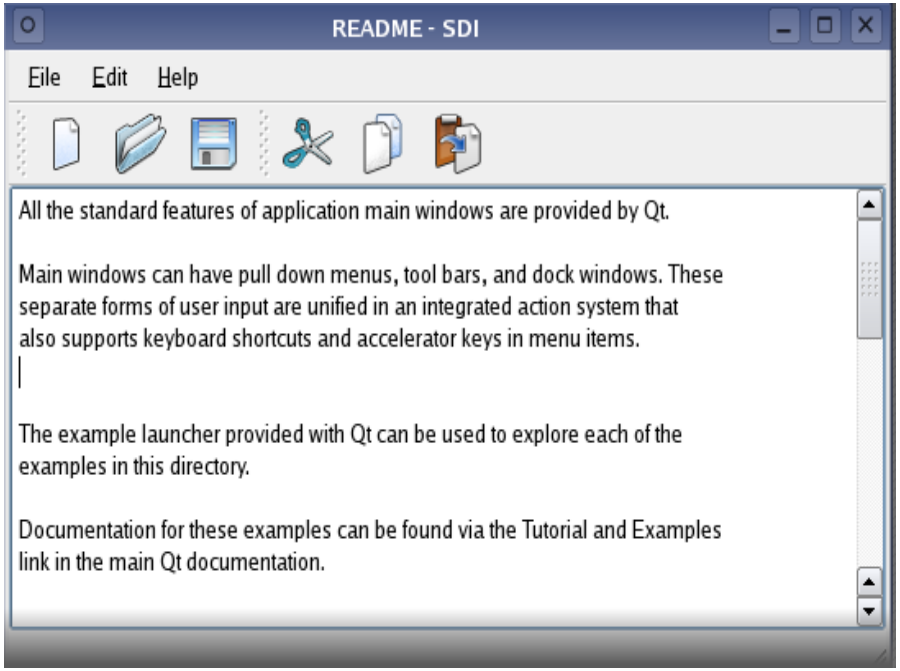
qmake command

- in the directory containing the **source files:**
  - **qmake** -project // creates **xxx.pro**
  - **qmake** // creates the **Makefile** (or equivalent)
  - **make** // creates **.moc** and binary files

Main Window
(QMainWindow)

Predefined areas for:

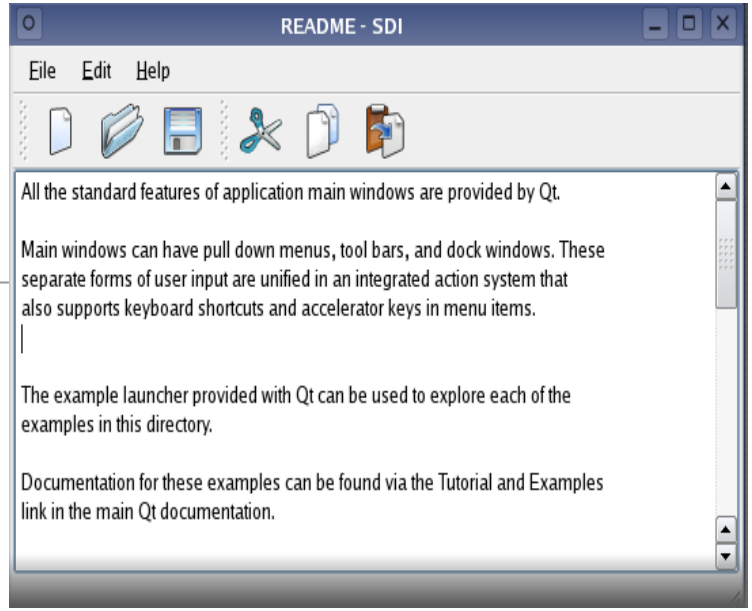
- a **menu bar**
- a **tool bar**
- a **status bar**
- a **central area** (working area)



Principle

- Create a **subclass** a **QMainWindow**
- Which **constructor** creates the widgets

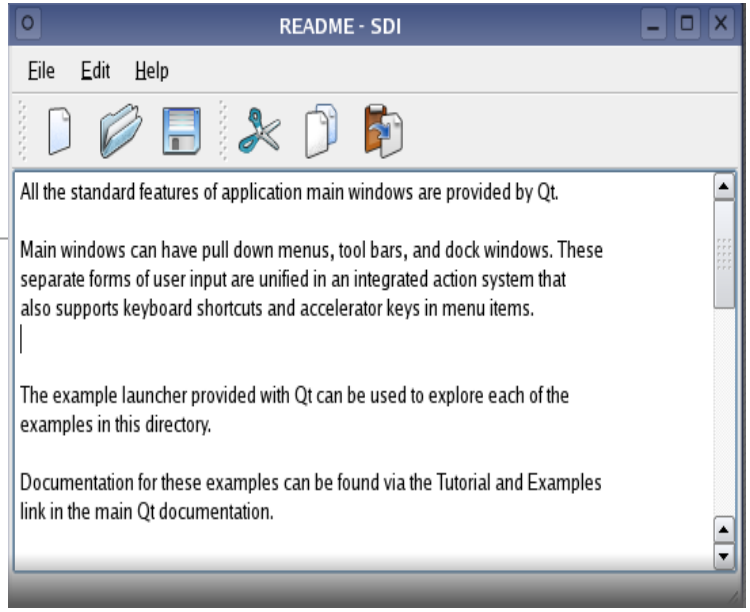
Main Window



In the constructor of the class deriving from QMainWindow

```
QMenuBar * mbar = this->menuBar( );                    // adds and returns a menu bar
QMenu * fmenu = mbar->addMenu( tr("&File") );          // adds a pulldown menu

                                                       // new.png is in a resource file
QAction * action = new QAction( QIcon(":/new.png"), tr("&New..."), this);

fmenu->addAction(action);                              // adds the action to the menu
connect(action, SIGNAL(triggered( )), this, SLOT(open( )));   // slot connection

action->setShortcut( tr("Ctrl+N") );      // hot key
action->setToolTip( tr("New file") );     // tool tip
action->setStatusTip( tr("New file") );   // info on status bar
```

Main Window



Actions

- can both be added to **menus** and **toolbars**

```
QToolBar * toolbar = this->addToolBar( tr("File") );
toolbar->addAction(action);
```

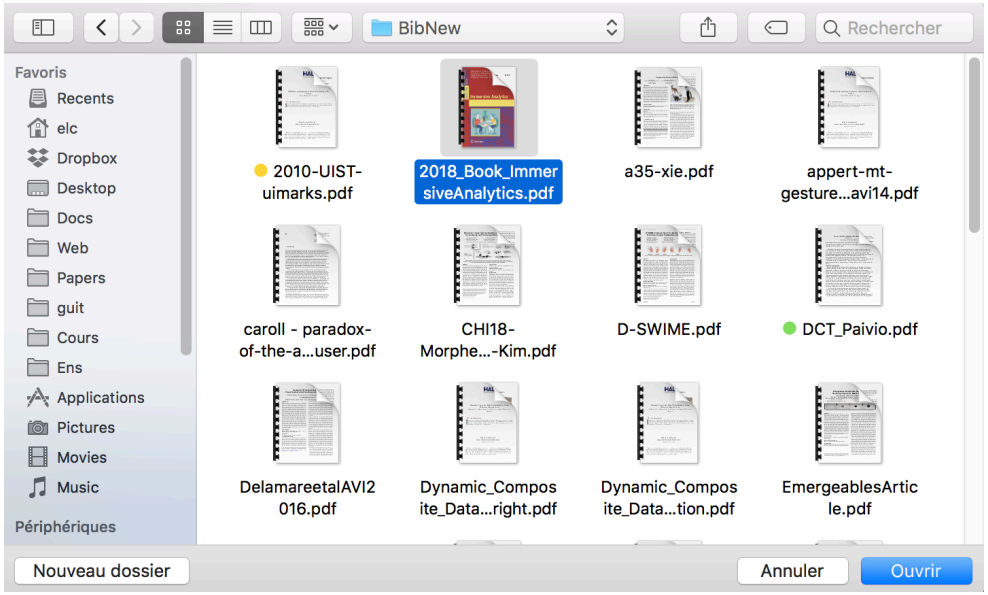
Central area

```
QTextEdit * text = new QTextEdit(this);
this->setCentralWidget(text);
```

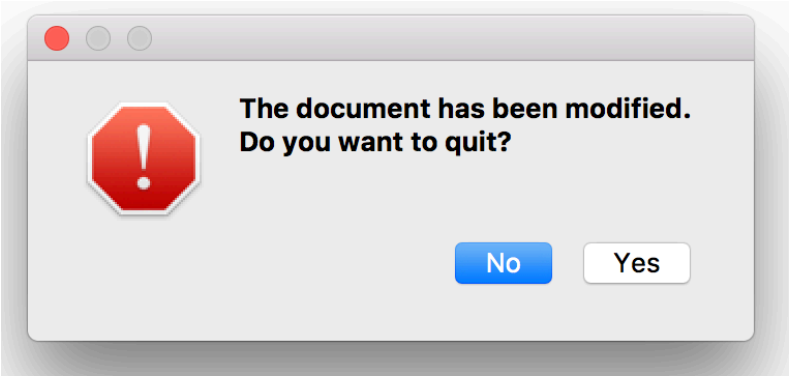
Internationalization

- **tr( )** **translates** the text

Dialog boxes

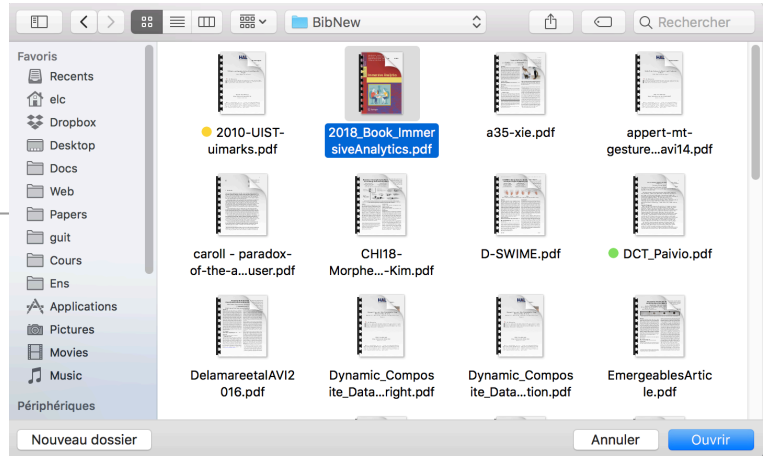


QFileDialog



QMessageBox

File dialog




```
QString filename =
    QFileDialog::getOpenFileName(this,
                                 tr("Open Image"),                    // title
                                 "/home/jana",                        // directory
                                 tr("Image Files (*.png *.jpg *.bmp)")  // filter
                                 );
```

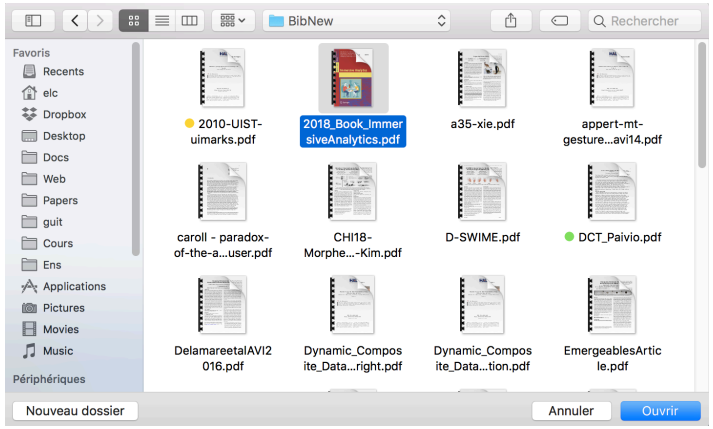
A **modal** **dialog box** **blocks** **the interaction** => forces the user to answer !

- Advantage: **easy** to program!
- Drawback: not always **convenient** for the user

File dialog

Retrieving several files

```
QFileDialog * dialog = new QFileDialog(parent);
dialog->setFilter("Text files (*.txt)");
QStringList fileNames;

if (dialog->exec( ) == QDialog::Accepted) {
    fileNames = dialog->selectedFiles();
    QString name1 = fileNames[0];
    ...
}
```



dialog.exec() and **getOpenFileName()** starts a **secondary event loop**

Internationalized text

Problem

- so many languages, so many symbols!
- solution: **UNICODE**

QString: Unicode strings

- made of **QChar**s (16 bits each)
- **one symbol** = **one QChar** (or **two QChar** for rare languages, Emojis, etc.)

QString conversions:

- **qPrintable**(string) : converts to the **local encoding**
- **toAscii**( ) : **ASCII** 8 bits
- **toUtf8**( ) : **UTF-8** Unicode multibyte (1 char = 1 à 4 octets)
- **toLocal8Bit**( ) : local encoding 8 bits
- etc.

Internationalized text

QFile:
for opening a file

```
QFile file(filename);
if (file.open(QIODevice::ReadOnly | QIODevice::Text))  { .... }
if (file.open(QIODevice::WriteOnly))  { ... }
```

QTextStream:
for reading or writing text

```
QFile file("output.txt");
if (file.open(QIODevice::WriteOnly)) {
    QTextStream out (&file);
    out << "Result: " << 10 << " " << 3.14;     // writes on out
}
```

- **To read a word :**

  stream >> *arg1* >> *arg2*; // stops reading before a space

- **To read a line :**

  stream.readLine(*size*); // returns the line, **no size limit** if size = 0

- **To read the whole text file :**

  stream.readAll( ); // only for small files!

Debugging

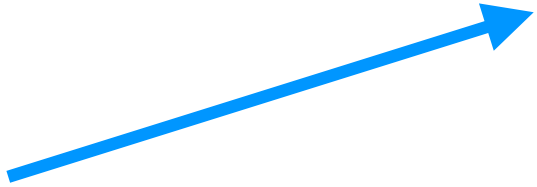
qDebug: for displaying debug messages

```
#include <QDebug>
 ......
 int val;
 QString str;
 qDebug( ) << "value: "  <<  str << " / " << val;

 // or, using standard C++ IO
 std::cout << "value: "  <<  qPrintable(str) << " / " << val << std::endl;
```

don't forget endl !

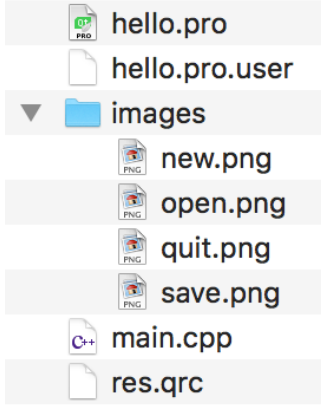
Icons & resource files

```
QAction * action = new QAction( QIcon(":/images/new.png"), tr("&New..."), this);
```



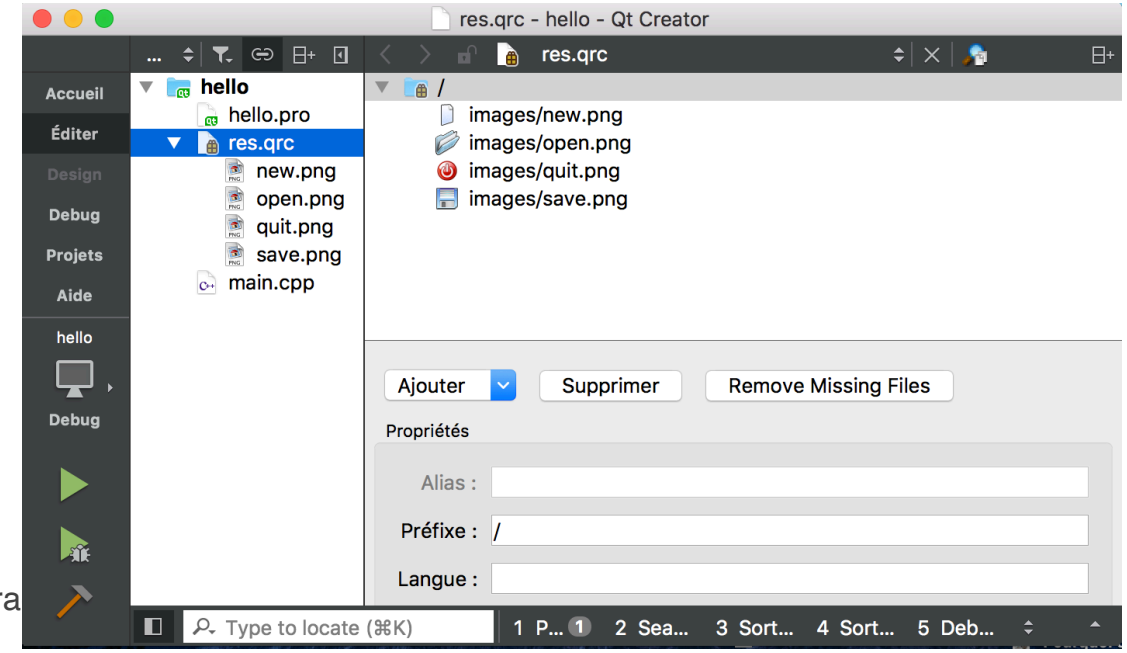
: means that the path is **relative to the application**

here **new.png** is in sub-directory **images**



Caution!

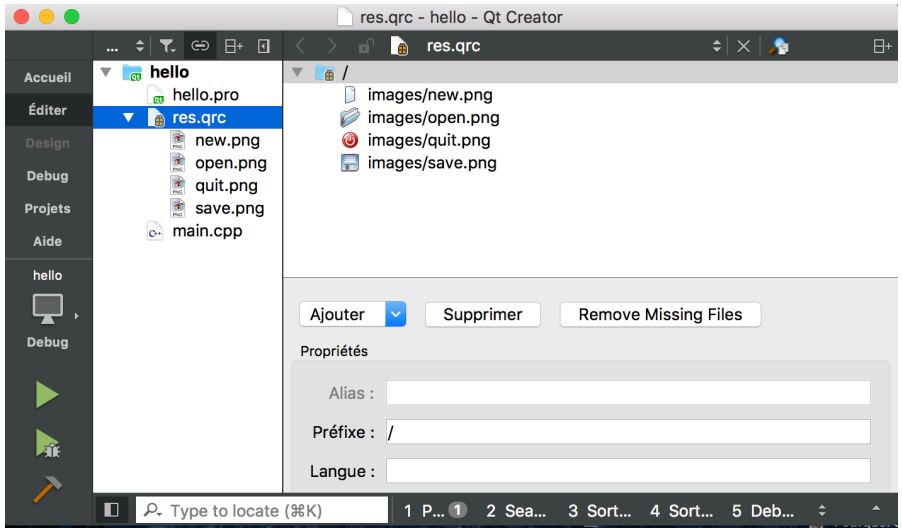
The **Prefix** must be **empty or /**

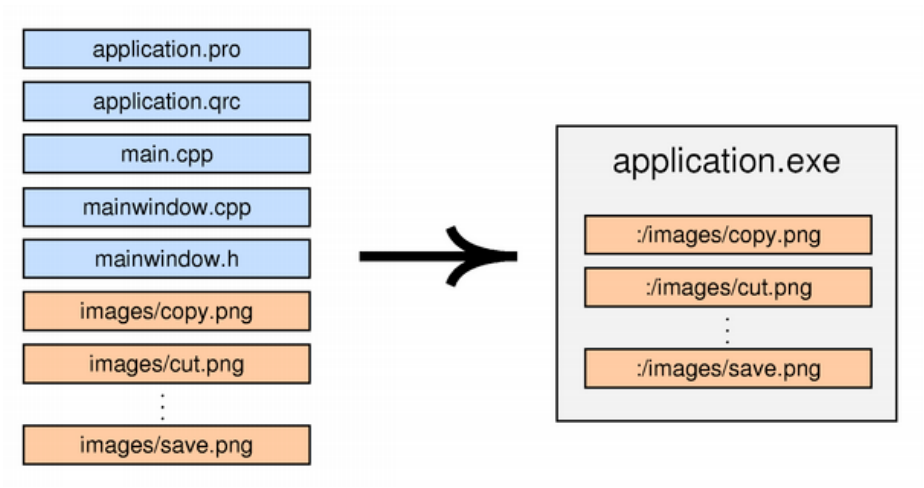


Ressource files

.qrc file

- created by hand or by **QtCreator**
- images are **compiled** and **included** in the executable
  - *(depends on config)*






```
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
    <file>images/copy.png</file>
    <file>images/cut.png</file>
    <file>images/new.png</file>
    <file>images/open.png</file>
    <file>images/paste.png</file>
    <file>images/save.png</file>
</qresource>
</RCC>
```

Ressource files & localization

- tr( ) **translates** the text
- **QLinguist** tool

```
QMenuBar * mbar = this->menuBar( );                    // adds and returns a menu bar
QMenu * fmenu = mbar->addMenu( tr("&File") );          // adds a pulldown menu

                                                       // new.png is in a resource file
QAction * action = new QAction( QIcon(":/new.png"), tr("&New..."), this);

action->setShortcut( tr("Ctrl+N") );      // hot key
action->setToolTip( tr("New file") );     // tool tip
action->setStatusTip( tr("New file") );   // info on status bar
```

More on signals / slots

Custom signals / slots

In the header file (.h)

- **QObject** or **QWidget** subclass
- **Q_OBJECT**, **slots**, **signals** keywords
- **slots** must be **implemented**
- **signals** are **not** implemented (done by the **moc**)

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int balance = 0;
public:
    BankAccount( ) { balance = 0; }
    int getBalance( ) const { return balance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

Implementing a slot

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int balance = 0;
public:
    BankAccount( ) { balance = 0; }
    int getBalance( ) const { return balance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

In the implementation file (.cpp)

```
void BankAccount::setBalance(int newBalance)
{
    if (balance != newBalance) {
        balance = newBalance;
        emit balanceChanged(balance);
    }
}
```

emit

- emits the **balanceChanged**() **signal**
- the signal "holds" a **value** (the balance)

One way connection

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int balance = 0;
public:
    BankAccount( ) { balance = 0; }
    int getBalance( ) const { return balance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

```
auto * x = new BankAccount();
auto * y = new BankAccount();

connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));

x->setBalance(10);
```

```
void BankAccount::setBalance(int newBalance)
{
    if (balance != newBalance) {
        balance = newBalance;
        emit balanceChanged(balance);
    }
}
```

One way connection

```
class BankAccount : public QObject {
    Q_OBJECT
private:
    int balance = 0;
public:
    BankAccount( ) { balance = 0; }
    int getBalance( ) const { return balance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
};
```

```
auto * x = new BankAccount();
auto * y = new BankAccount();

connect(x, SIGNAL(balanceChanged(int)), y, SLOT(setBalance(int)));

x->setBalance(10);
```

```
void BankAccount::setBalance(int newBalance)
{
    if (balance != newBalance) {
        balance = newBalance;
        emit balanceChanged(balance);
    }
}
```

- **x**->balance is set to 10
- ➜ **balanceChanged**(**x**->balance) is **emitted**
- ➜ **y**->**setBalance**(**x**->balance) is called
- ➜ **y**->balance is set to 10

Two-way connection

```
connect(x, SIGNAL(balanceChanged(int)), y,  SLOT(setBalance(int)));
connect(y, SIGNAL(balanceChanged(int)), x,  SLOT(setBalance(int)));
x->setBalance(10);
```

```
void BankAccount::setBalance(int newBalance)
{
    if (balance != newBalance) {
        balance = newBalance;
        emit balanceChanged(balance);
    }
}
```

OK?

Two-way connection

```
connect(x, SIGNAL(balanceChanged(int)), y,  SLOT(setBalance(int)));
connect(y, SIGNAL(balanceChanged(int)), x,  SLOT(setBalance(int)));
x->setBalance(10);
```

```
void BankAccount::setBalance(int newBalance)
{
    if (balance != newBalance) {
        balance = newBalance;
        emit balanceChanged(balance);
    }
}
```

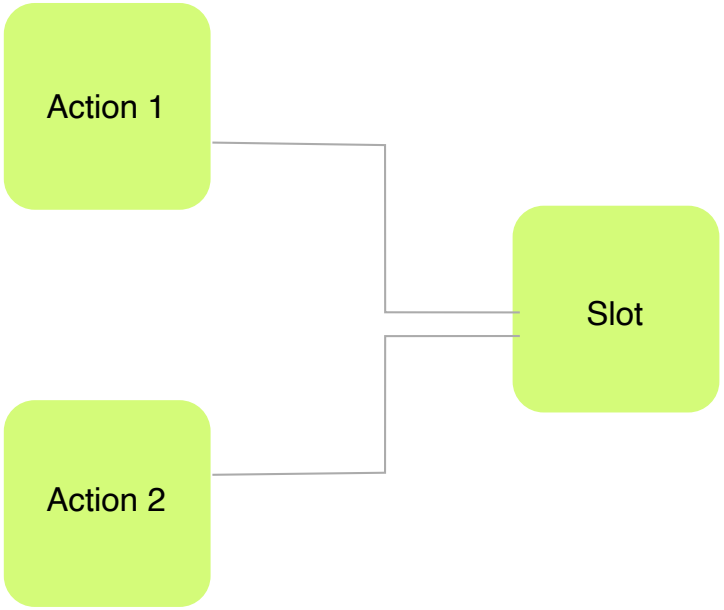
OK : because **emit** is not called if the value is unchanged

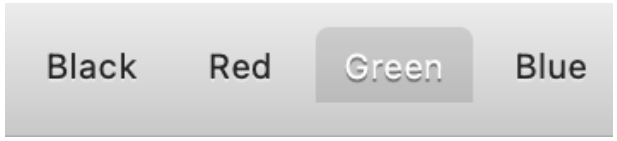
infinite loop otherwise!

Shared slots

How can we know the sender when several objects call the same slot?

- **signal / slot** model
  => the slot **does not know** the emitter

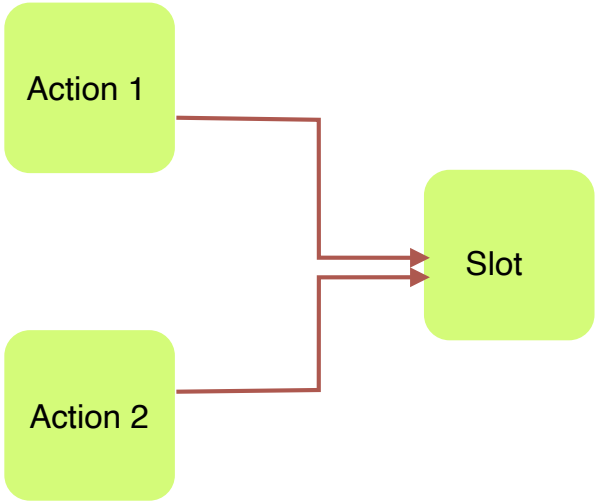




*example: call same **setColor**() slot*

Solution 1: Lambdas

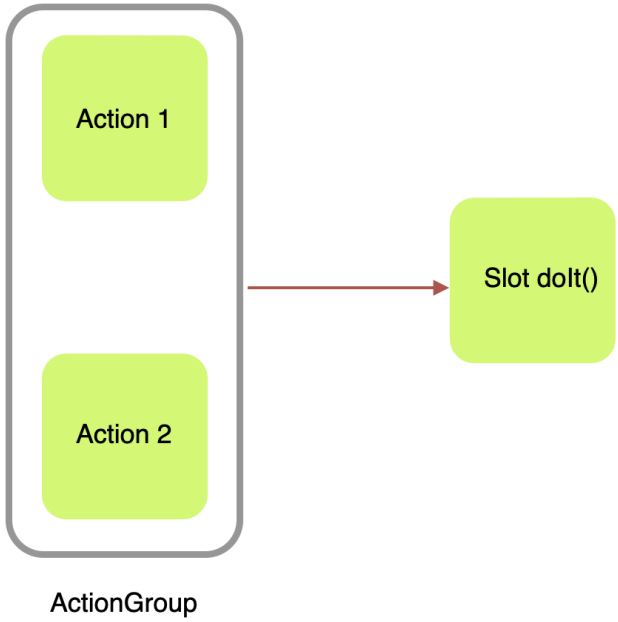
```
// instance variables of MyClass
QAction * action1{}, * action2{};

void MyClass::createGUI( ) {
    action1 = new QAction(tr("Action 1"), this);
    connect( action1, &QAction::triggered, [=] { doIt(action1); } );

    action2 = new QAction(tr("Action 2"), this);
    connect( action2, &QAction::triggered, [=] { doIt(action2);} );
    ....
}

void MyClass::doIt(QAction * sender) {
    if (sender == action1) ....;
    else  if (sender == action2) .... ;
    ....
}
```



action1 and action2 are **instance** variables

Solution 2: QActionGroup

```
// instance variable of MyClass
QAction * action1{}, * action2{};


void MyClass::createGUI( ) {
    auto * group = new QActionGroup(this);
    // only one connect !
    connect(group, SIGNAL(triggered(QAction *)),
            this, SLOT(doIt(QAction *)));

    action1 = group->addAction(tr("Action 1"));
    action2 = group->addAction(tr("Action 2"));
    ...
}

void MyClass::doIt(QAction * sender) {
    if (sender == action1) ....;
    else  if (sender == action2) .... ;
    ....
}
```



Exclusive selection by default

can be changed, see: **setExclusive**()

QButtonGroup works the same with buttons

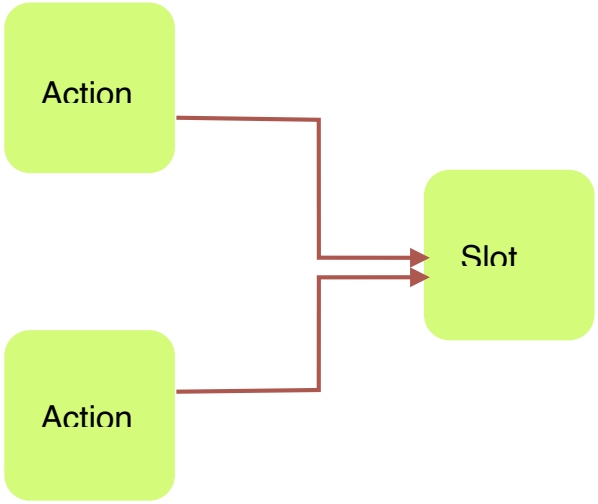
with signals:

buttonClicked(QAbstractButton *)

buttonClicked(int id)

Solution 3: QObject::sender()

```
// instance variables of MyClass
QAction * action1{}, * action2{};

void MyClass::createGUI( ) {
    action1 = new QAction(tr("Action 1"), this);
    connect( action1, SIGNAL(triggered()), this, SLOT(doIt()) );

    action2 = new QAction(tr("Action 2"), this);
    connect( action2, SIGNAL(triggered()), this, SLOT(doIt()) );
    .....
}

void MyClass::doIt( ) {
    auto * sender = QObject::sender( );          // similar to getSource() in Java/Swing
    if (sender == action1) ....;
    else  if (sender == action2) .... ;
    ....
}
```



Also: QSignalMapper

QSignalMapper Redirects signals

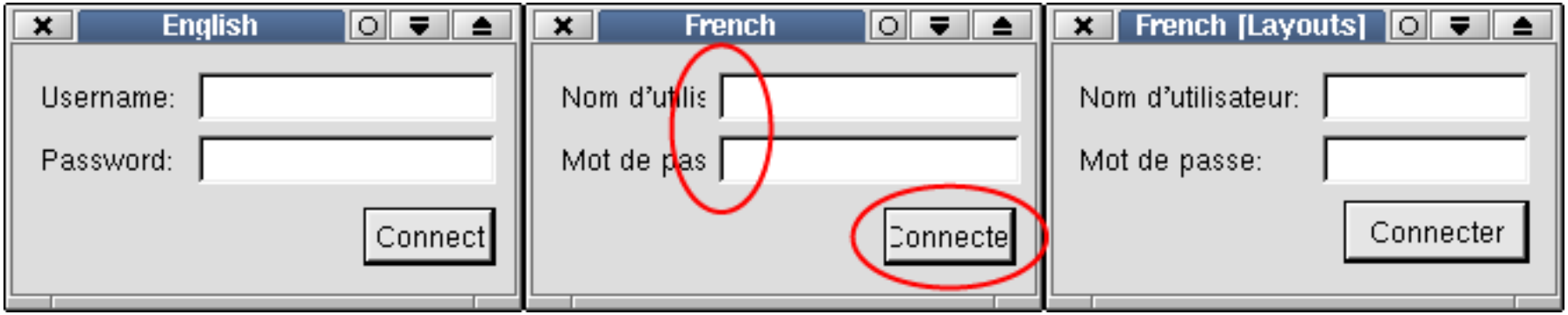
for: int, QString&, QWidget* et QObject*

```
void MyClass::createGUI( ) {
    QSignalMapper* mapper = new QSignalMapper (this) ;
    connect(mapper, SIGNAL(mapped(int)), this, SLOT(doIt(int))) ;

    QPushButton * btn1 = new QPushButton("Action 1"), this);
    connect (btn1, SIGNAL(clicked( )), mapper, SLOT(map( ))) ;
    mapper->setMapping (btn1, 1) ;

    QPushButton * btn2 = new QPushButton("Action 1"), this);
    connect (btn2, SIGNAL(clicked( )), mapper, SLOT(map( ))) ;
    mapper->setMapping (btn2, 2) ;
     ...
}

void MyClass::doIt(int value) {
    ....
}
```

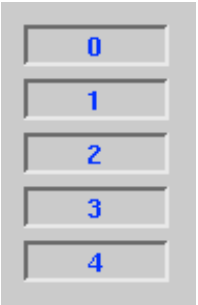
Layout

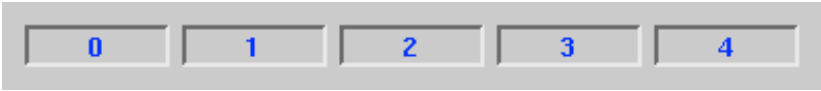
Goal

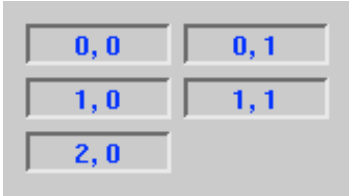
- **Internationalization** : widget sizes **adapt** to **text and font sizes**
- Allow the user to **resize widgets interactively**
- Layout computed by toolkit => **easier to program**



Layout

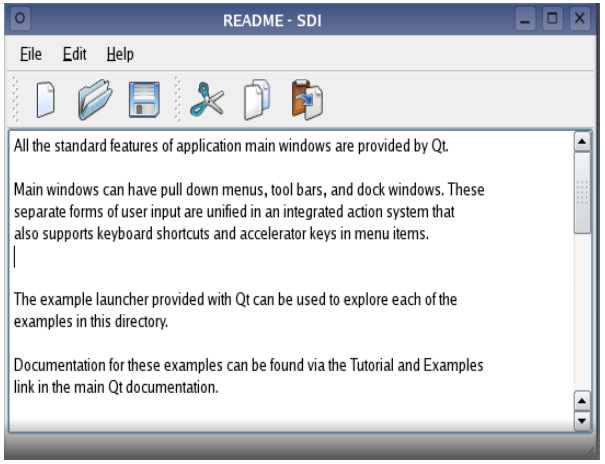






QHBoxLayout, QVBoxLayout, QGridLayout

QFormLayout



QMainWindow

Layout

```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget(new QPushButton("OK"));
v_layout->addWidget(new QPushButton("Cancel"));
v_layout->addStretch( );
v_layout->addWidget(new QPushButton("Help"));
```

- Layouts can be **nested**

- **addStretch**() and **addSpacing**()
  add flexible and not flexible **space**

Layout

```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch( );
v_layout->addWidget( new QPushButton( "Help" ) );
```


```
QListBox * country_list = new QListBox( this );
country_list->insertItem("Canada");
...etc...

QHBoxLayout * h_layout = new QHBoxLayout( );
h_layout->addWidget(country_list);
h_layout->addLayout(v_layout);
```

nested layouts

Layout

```
QVBoxLayout * v_layout = new QVBoxLayout( );
v_layout->addWidget( new QPushButton( "OK" ) );
v_layout->addWidget( new QPushButton( "Cancel" ) );
v_layout->addStretch( );
v_layout->addWidget( new QPushButton( "Help" ) );

QListBox * country_list = new QListBox( this );
countryList->insertItem( "Canada" );
...etc...

QHBoxLayout * h_layout = new QHBoxLayout( );
h_layout->addWidget( country_list );
h_layout->addLayout( v_layout );

QVBoxLayout * top_layout = new QVBoxLayout( );
top_layout->addWidget(new QLabel("Select a country", this));
top_layout->addLayout(h_layout);

window->setLayout(top_layout);
window->show( );
```

Layout constraints

For each QWidget

- **sizeHint** = **preferred size** of the widget
- **sizePolicy** : **how** the widget is **resized**

- **Fixed**: = sizeHint
- **Minimum**: >= sizeHint
- **Maximum**: <= sizeHint
- **Preferred**: ~ sizeHint
  can be shrunk, can be expanded
- **Expanding**: get as much space as possible,
  can be shrunk
- **MinimumExpanding**: same, but can't be shrunk
- **Ignored**: same, ignores sizeHint

canvas : Canvas

Propriété	Valeur
Hauteur	373
▼ **sizePolicy**	[Expanding,...
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

settingBox : QToolBox

Propriété	Valeur
▼ **sizePolicy**	[Fixed, Pref...
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Layout constraints

For each QWidget

- **sizeHint** = **preferred size** of the widget
- **sizePolicy** : **how** the widget is **resized**

canvas : Canvas

Propriété	Valeur
Hauteur	373
▼ **sizePolicy**	[Expanding,...
Politique hori...	Expanding
Politique verti...	Expanding
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	300 x 0
Largeur	300
Hauteur	0
▼ maximumSize	16777215 x ..

Main methods

- **sizeHint**()
- **setMinimumSize**(), **setMaximumSize**()
- **setSizePolicy**(QSizePolicy)
- **setSizePolicy**(QSizePolicy::Policy horizontal, QSizePolicy::Policy vertical)
- **setHorizontalPolicy**()
- **setVerticalPolicy**()

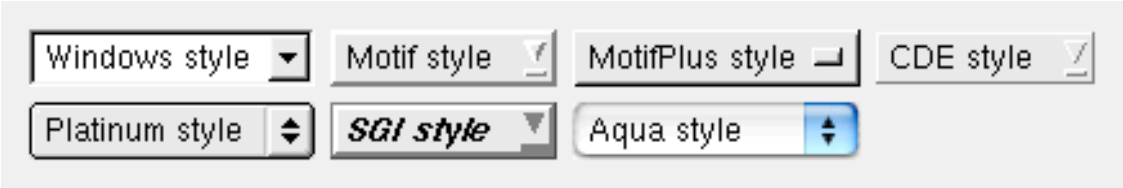
settingBox : QToolBox

Propriété	Valeur
▼ **sizePolicy**	[Fixed, Pref...
Politique hori...	Fixed
Politique verti...	Preferred
Étirement hori...	0
Étirement vert...	0
▼ **minimumSize**	200 x 0
Largeur	200
Hauteur	0
▼ maximumSize	16777215 x ...
Largeur	16777215

Styles & Style Sheets

QStyle: emulates the native "look and feel"

- The **look and feel** is emulated and can be changed (as with Java Swing)



QStyleSheet: for customizing the "look"

- **CSS**-like files

```
myTextEdit->setStyleSheet("color: blue;"
                          "background-color: yellow;"
                           "selection-color: yellow;"
                           "selection-background-color: blue;");
```

Events & 2D Graphics

Repainting a widget

A widget is repainted when needed

- When the **application** is **started** or **restored** (deiconified)
- When a **window** is **shown** or **moved** (depending on the OS, etc.)
- When **requesting** it **explicitely** by calling **update**( )

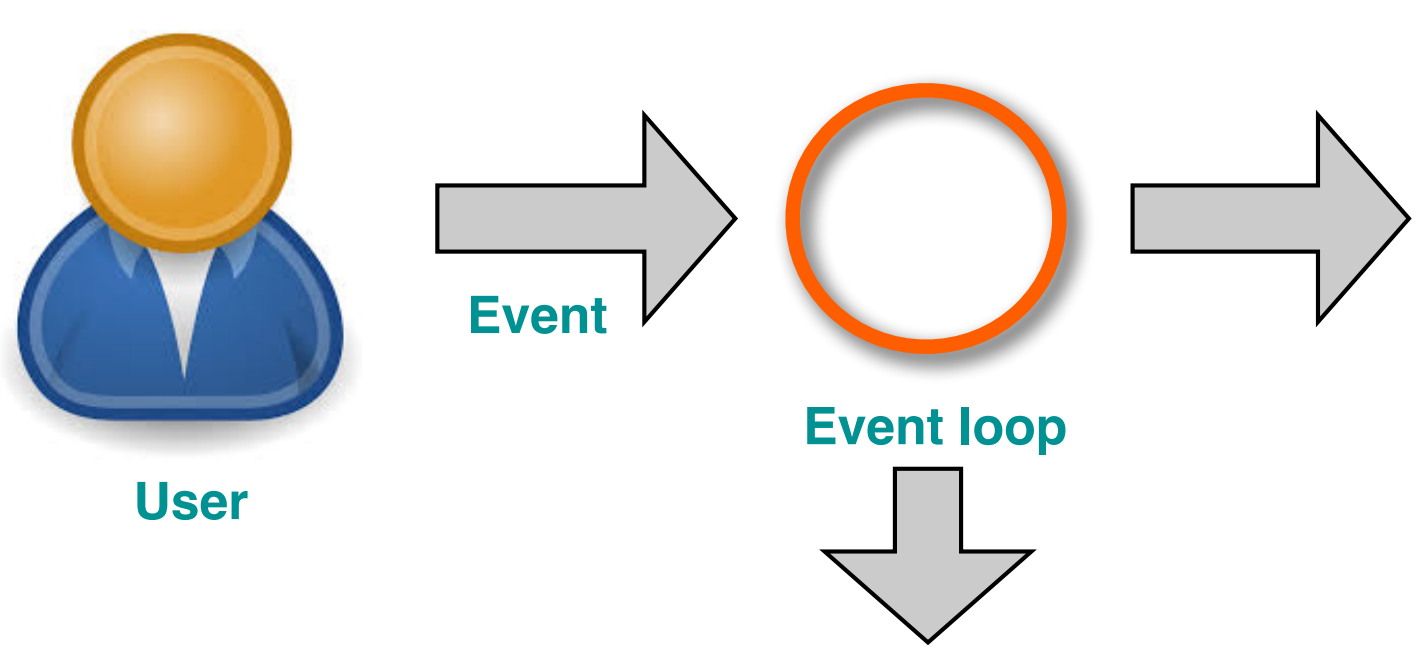
damaged / repaint model

- Differs from **3D applications** (usually, the scene is **constantly repainted**)

Damaged / repaint model

update() tells the system that an area has been damaged

paintEvent() is then automatically called after processing slots




```
void mySlot(...)
    ......
    update();
    ......
    update(x, y, w, h);
    ......
    update(region1);
    update(region2);
    ......
}
```

```
void paintEvent(QPaintEvent* e) {
    ......
}
```

Damaged / repaint model

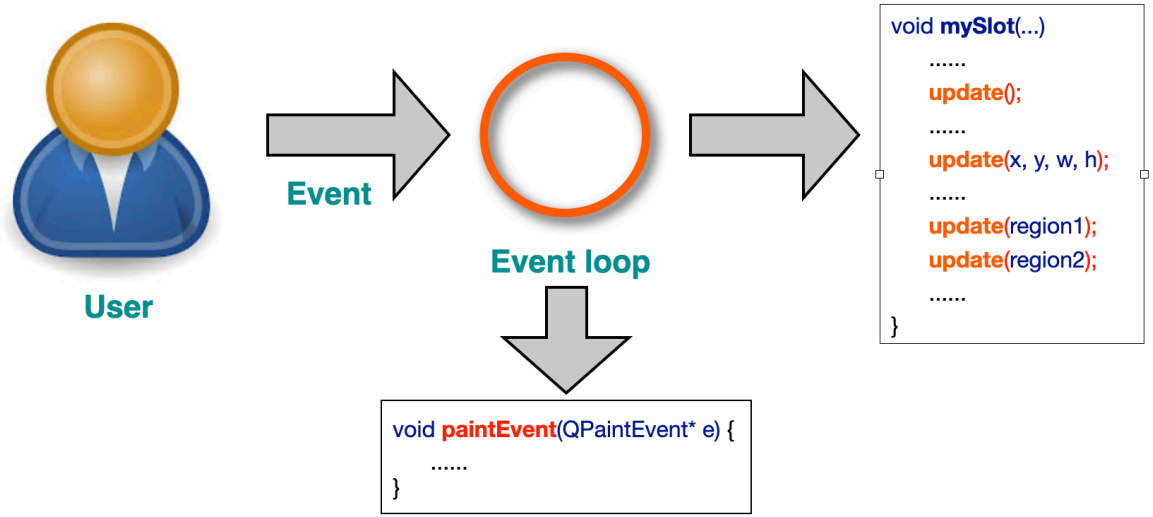
Stores requests in a **waiting list**

update() tells the system that an area has been damaged

paintEvent() is then automatically called

Caution!

- **Don't call paintEvent()**
- **Draw only** in **paintEvent()**



Notes:

repaint() redraws **immediately**

- for special cases (animations, etc.)

Java works the same way **but**:

- **update() Qt** == **repaint() Java**
- **paintEvent() Qt** == **paint() Java**

Painting

Principle: create a subclass of QWidget that redefines paintEvent()

Canvas.h

```
#include <QWidget>

class Canvas : public QWidget {
public:
    Canvas(QWidget* parent) : QWidget(parent) {}

protected:
    virtual void paintEvent(QPaintEvent*);
};
```

Canvas.cpp

```
#include <QPainter>
#include "Canvas.h"

void Canvas::paintEvent(QPaintEvent* e) {
    // standard behavior (draws the background)
    QWidget::paintEvent(e);

    // creates a painter for this widget
    QPainter painter(this);
    painter.drawLine(50, 10, 100, 20);
}
```

Painting

Use QPainter only in paintEvent()
(because of double buffering)

QPainter is allocated on the stack!

=> **auto-deleted** when the function returns

=> **avoid** using **new!**

slower

memory leak if you dont delete it !

Canvas.cpp

```
#include <QPainter>
#include "Canvas.h"

void Canvas::paintEvent(QPaintEvent* e) {
    // standard behavior (draws the background)
    QWidget::paintEvent(e);

    // creates a painter for this widget
    QPainter painter(this);
    painter.drawLine(50, 10, 100, 20);
}
```

Detecting events

Principle: create a subclass of QWidget that redefines these functions

=> **no signals / slots** in this case!

```
void mousePressEvent(QMouseEvent*);

void mouseReleaseEvent(QMouseEvent*);

void mouseDoubleClickEvent(QMouseEvent*);

void mouseMoveEvent(QMouseEvent*);

void setMouseTracking(bool)

bool hasMouseTracking()
```

Detects **drag events**
if **setMouseTracking(true)** was called

drag event = **move event**
with mouse button **pressed**

Detecting events

Canvas.h

```
#include <QWidget>
#include <QMouseEvent>

class Canvas : public QWidget {
public:
    Canvas(QWidget* p) : QWidget(p) {}

protected:
   void mousePressEvent(QMouseEvent*);
};
```

Canvas.cpp

```
#include "Canvas.h"

void Canvas::mousePressEvent(QMouseEvent* e) {
    if (e->button() == Qt::LeftButton) {
            ......
            .....
      update();          // dont forget!
    }
}
```

Detecting events

QMouseEvent methods

- **button**() **button** that triggered the event
  ex: Qt::LeftButton
- **buttons**() State of the **other** buttons
  (ORed mask). ex: Qt::LeftButton | Qt::MidButton
- **modifiers**() Keyboard **modifiers**
  ex: Qt::ControlModifier | Qt:: ShiftModifier

- **position**( ) relative to the **widget**
- **scenePosition**( ) relative to **window**
- **globalPosition**( ) relative to **screen**

Canvas.cpp

```
#include "Canvas.h"

void Canvas::mousePressEvent(QMouseEvent* e) {
    if (e->button() == Qt::LeftButton) {
        ......
        .....
      update();        // dont forget!
    }
}
```

Cursor position

- **QCursor::pos**( )

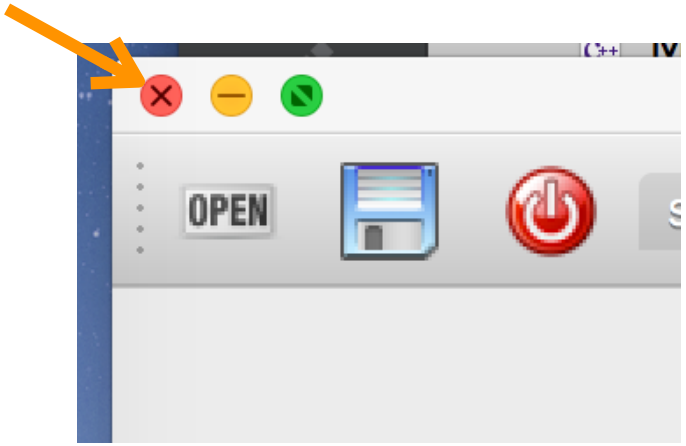
Coordinates

- **QWidget::mapToGlobal**( )
- **QWidget::mapFromGlobal**( )

Ignoring events

To ignore events

- **QEvent::ignore()** => The receptor **ignores** the event



Example: quitting an application

- When clicking on the window's **close button :**
    - The **QEvent::closeEvent()** method is called
- **=> Redefine closeEvent() so that it calls ignore()**

```
void MainWindow::closeEvent(QCloseEvent *e) {
    e->ignore();
    myMethod();
}
```

Painting


Stroke


Fill


Composite


Clipping Path
Clip


Shearing Transform


Aa
Bb

QPen

Attributes

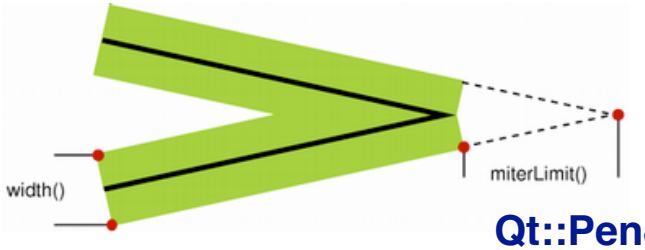
- style
- width
- brush
- capStyle
- joinStyle

default value is 0
= "**cosmetic pen**"

```
// in paintEvent()

QPen pen;        // default pen
pen.setStyle(Qt::DashDotLine);
pen.setWidth(3);
pen.setBrush(Qt::green);
pen.setCapStyle(Qt::RoundCap);
pen.setJoinStyle(Qt::RoundJoin);

QPainter painter(this);
painter.setPen(pen);
// etc....
```

Qt::PenStyle

Qt::PenCapStyle



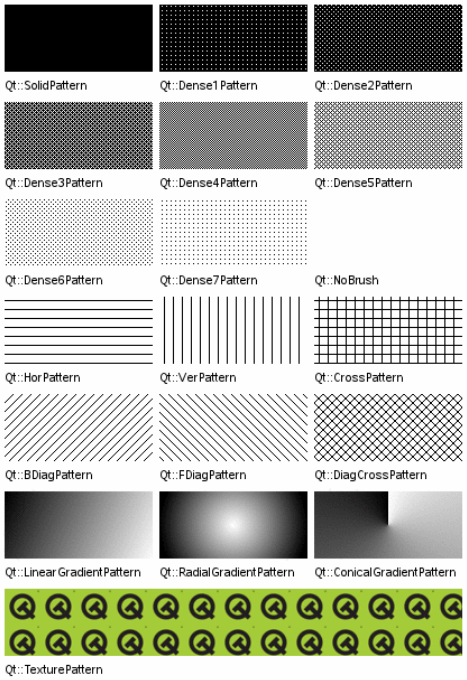
Qt::PenJoinStyle

QBrush

Attributes

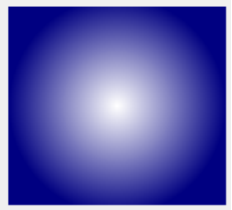
- style
- color
- gradient
- texture

Qt::GlobalColor (predefined colors)



Qt::BrushStyle

QLinearGradient



QRadialGradient

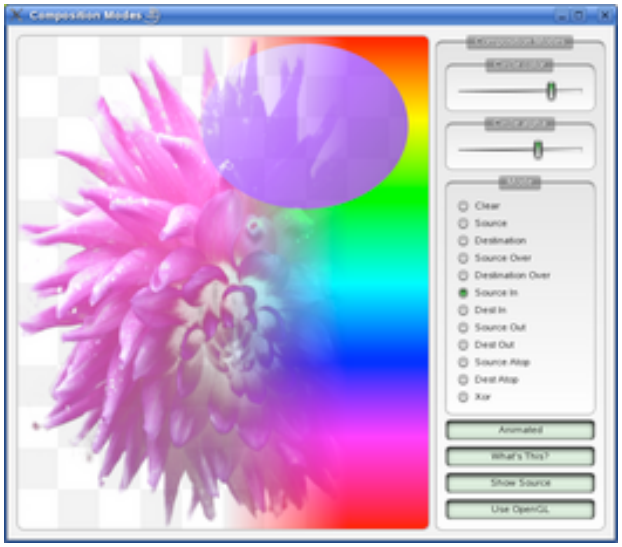
QConicalGradient

```
QLinearGradient
gradient( QPointF(0, 0), QPointF(100, 100) );

gradient.setColorAt(0, Qt::white);
gradient.setColorAt(1, Qt::blue);

QBrush brush(gradient);
```

Compositing

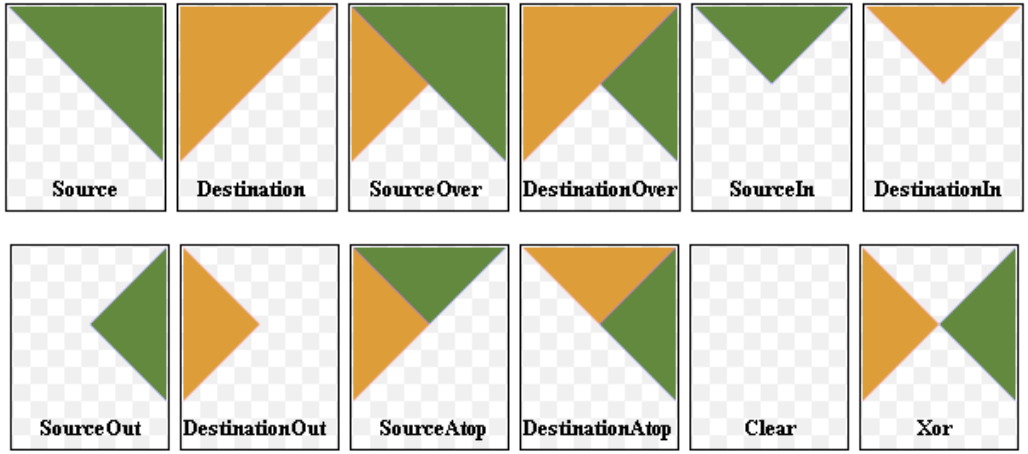
Porter Duff operators

- Specify: **F(source, destination)**
  - **QPainter::setCompositionMode()**
- Default: **alpha blending** (**SourceOver** mode)
  - dest <= a_{src} * source + (1-a_{src}) * a_{dest} * dest



Limitations

- hardware, implementation
- what your are drawing on



Clipping

QPainter functions

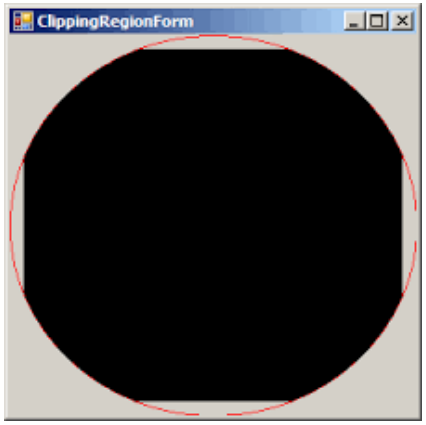
- QPainter::setClipping(), setClipRect(), setClipRegion(), setClipPath()

Operations on QRegion

- united(), intersected(), subtracted(), xored()

Example: intersection between a rectangle and an ellipse

```
QRegion r1(QRect(100, 100, 200, 80), QRegion::Ellipse);   // r1: elliptic region
QRegion r2(QRect(100, 120, 90, 30));                      // r2: rectangular region
QRegion r3 = r1.intersected(r2);                          // r3: intersection

QPainter painter(this);
painter.setClipRegion(r3);
...etc...
```

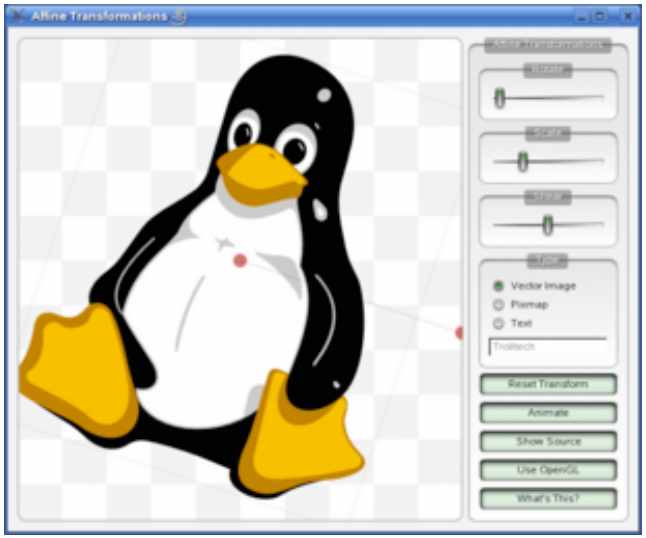


Geometrical transformations

Operations

- QPainter::setTransform()
- translate(), rotate(), scale(), shear()

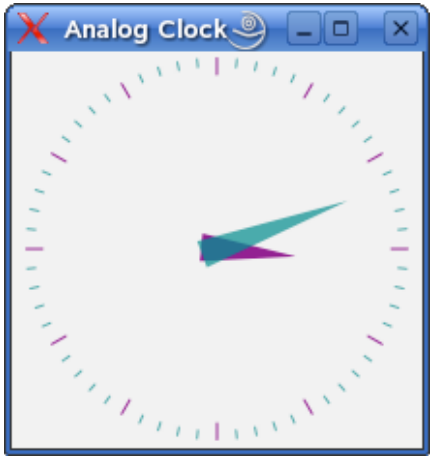
See Qt demos



Context state

- save(), restore()

Example: Clock hand




```
QPainter painter( this );
painter.translate( width( ) / 2, height( ) / 2 );
painter.save( );                    // saves current state
painter.rotate( 30.0 * ((time.hour() + time.minute() / 60.0)) );
painter.drawConvexPolygon( hourHand, 3 );
painter.restore( );                 // restore previous state
```

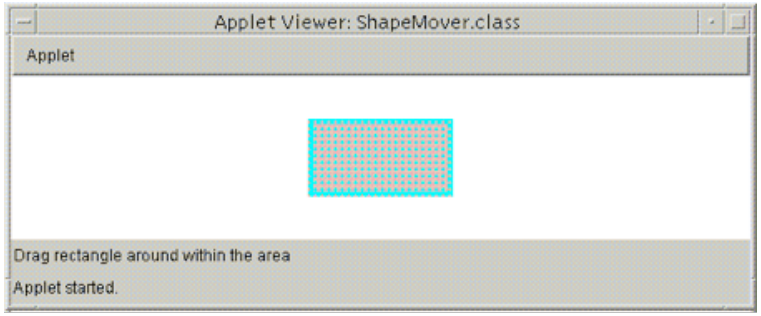
Picking / Interaction

Methods of QRect, QRectF, QPainterPath, etc.

- intersects(), contains() : returns true or false
- intersected() : returns the **intersection**

Notes

- A "**tolerance**" is often neeeded
  - eg: interset a **small rectangle** rather than a **point**
- Trick to intersect the **border** of a **closed shape**:
  - test if the shape **contains** the point
  - test if a **smaller** shape **contains** the point




```
QRect rect;

void mousePressEvent(QMouseEvent* e) {
    if (rect.contains( e->pos() )) {
        rect.moveCenter( e->pos() );
        update( );
}

void paintEvent(QPaintEvent* e ) {
    QPainter painter(this);
    .....
    painter.drawRect(rect);
}
```

Example

Clicking on the rectangle centers the rectangle

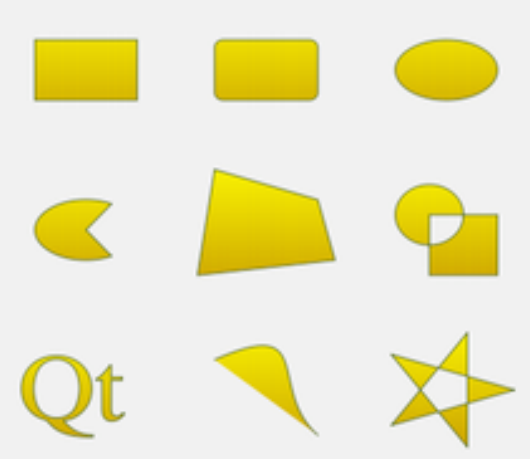
Paths

QPainterPath

- Combines **graphical primitives** and **geometrical operations**
- Provides **picking** for complex shapes
- Similar to Web: **SVG Paths** (Scalable Vector Graphics)
- **QPainter::drawPath**()

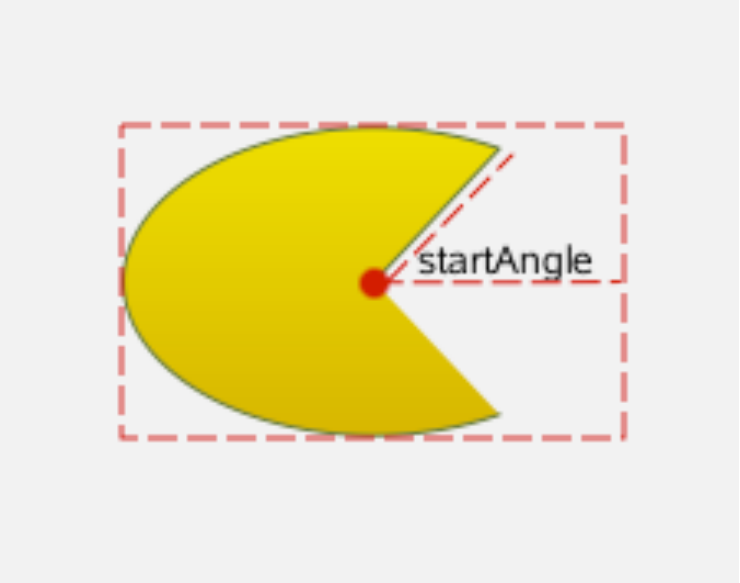
Operations

- **Move the pen**: moveTo(), arcMoveTo()
- **Draw a line**: lineTo(), arcTo()
- **Draw a curve**: quadTo(), cubicTo()
- **Add a primitive**: addRect(), addEllipse(), addPolygon(), addPath() ...
- **Text:** addText()
- **Geometrical operations:** translate(), etc.

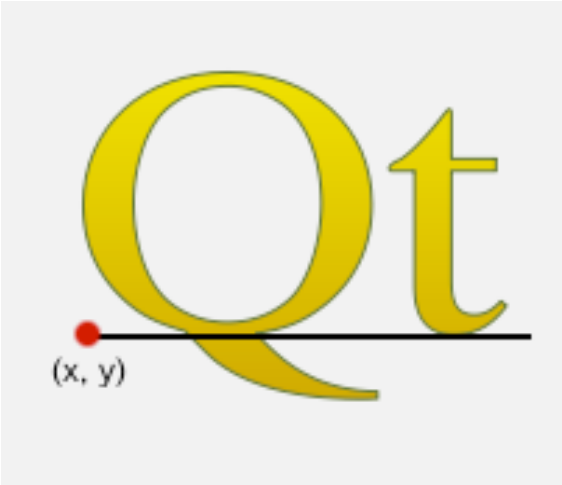


Paths

```
QPointF center, startPoint;
QPainterPath myPath;

myPath.moveTo( center );
myPath.arcTo( boundingRect, startAngle, sweepAngle );

QPainter painter( this );
painter.setBrush( myGradient );
painter.setPen( myPen );
painter.drawPath( myPath );
```




```
QPointF baseline(x, y);
QPainterPath myPath;

myPath.addText( baseline, myFont, "Qt" );

QPainter painter( this );
... etc....
painter.drawPath( myPath );
```



Paths

```
QPainterPath path;
path.addRect(20, 20, 60, 60);
path.moveTo(0, 0);
path.cubicTo(99, 0,  50, 50,  99, 99);
path.cubicTo(0, 99,  50, 50,  0, 0);

QPainter painter(this);
painter.fillRect(0, 0, 100, 100, Qt::white);
painter.setPen(QPen(QColor(79, 106, 25), 1, Qt::SolidLine, Qt::FlatCap, Qt::MiterJoin));
painter.setBrush(QColor(122, 163, 39));
painter.drawPath(path);
```

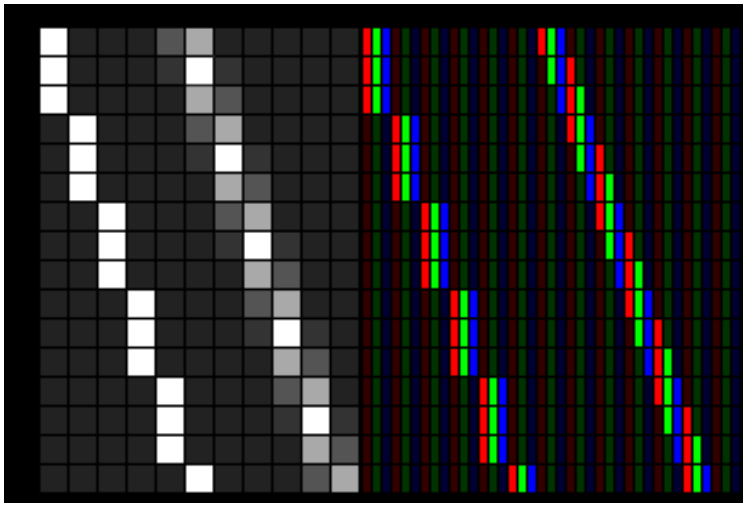
Qt::OddEvenFill (default)

Qt::WindingFill

Antialiasing

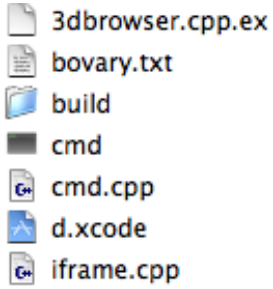
Anti-aliasing

- Because of **discrete geometry**
- Goal: **smooth contours**
- Especially useful for rendering **fonts** (e.g. ClearType**)**

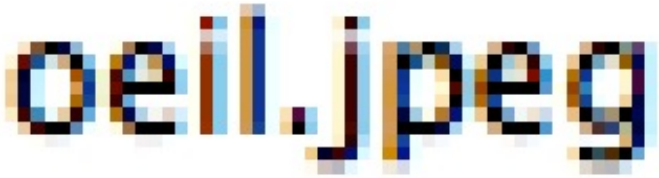


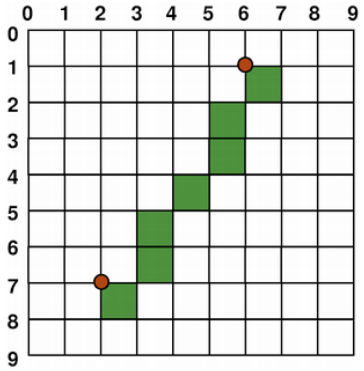
what is actually displayed

what you see

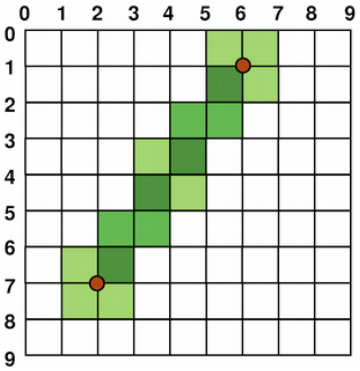


or (subpixel rendering):





aliased



antialiased

QPainter memo

Classes

- QPoint, QLine, QRect...
- QPointF, QLineF
- **QPainterPath**
- QRegion
- etc.

Attributes

- setPen( ) : how lines and outlines are drawn
- setBrush( ) : how space if filled inside line
- setFont( ) : text fonts
- setTransform( ) : geometrical transformations
- setClipRect/Path/Region( ) : clipping
- setCompositionMode( ) : composing (e.g. alpha blending)

Drawing

- drawPoint(), drawPoints(), drawLine(), drawLines()
- drawRect(), drawRects(), fillRect(),
- drawArc(), drawEllipse()
- drawPolygon(), drawPolyline()
- **drawPath**(), **fillPath**() : **combination** of graphical primitives
- drawText()
- drawPixmap(), drawImage(), drawPicture(),
- etc.

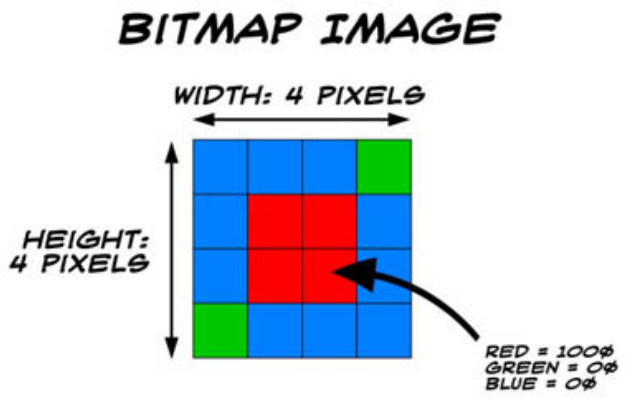
Images

Reading / saving / drawing

- load(), save()
- QPainter::drawImage()

RGB colors

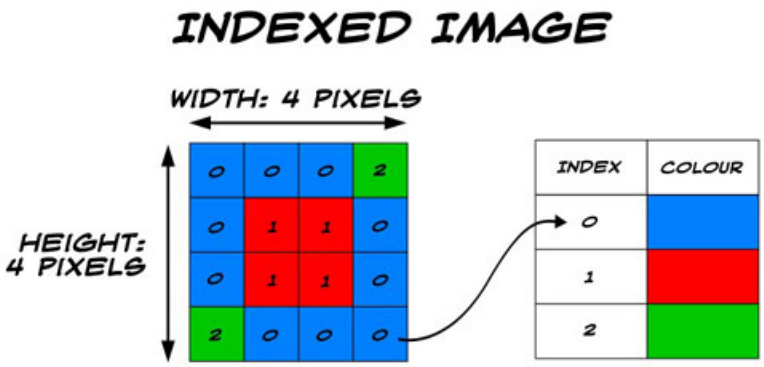
```
QImage image(3, 3, QImage::Format_RGB32);
QRgb value;
value = qRgb(122, 163, 39); // 0xff7aa327
image.setPixel(0, 1, value);
image.setPixel(1, 0, value)
```



Indexed colors

=> false colors

```
QImage image(3, 3, QImage::Format_Indexed8);
QRgb value;
value = qRgb(122, 163, 39); // 0xff7aa327
image.setColor(0, value);
value = qRgb(237, 187, 51); // 0xffedba31
image.setColor(1, value);
image.setPixel(0, 1, 0);
image.setPixel(1, 0, 1);
```



https://jbrd.github.io/2008/02/01/bitmap-and-indexed-images.html

QPicture

Enables recording graphical primitives

Record:

```
QPicture picture;
QPainter painter;
painter.begin( &picture );                // paint in picture
painter.drawEllipse( 10,20, 80,70 );      // draw an ellipse
painter.end( );                           // painting done
picture.save( "drawing.pic" );            // save picture
```

Replay :

```
QPicture picture;
picture.load( "drawing.pic" );            // load picture
QPainter painter;
painter.begin( &myImage );                // paint in myImage
painter.drawPicture( 0, 0, picture );     // draw the picture at (0,0)
painter.end( );                           // painting done
```

Scene graphs

Principle

- **Scene graph** = **graph of objects** describing the graphical primitives and geometrical operations
- Similar concept: **JavaFX**, **SVG**, 3D Graphics (X3D, editors and tools)

Advantages

- graphical objects **automatically repainted**
- automatic **picking**
- **multiple** synced views

Main classes

- **QGraphicsScene**
- **QGraphicsView**
- **QGraphicsItem**

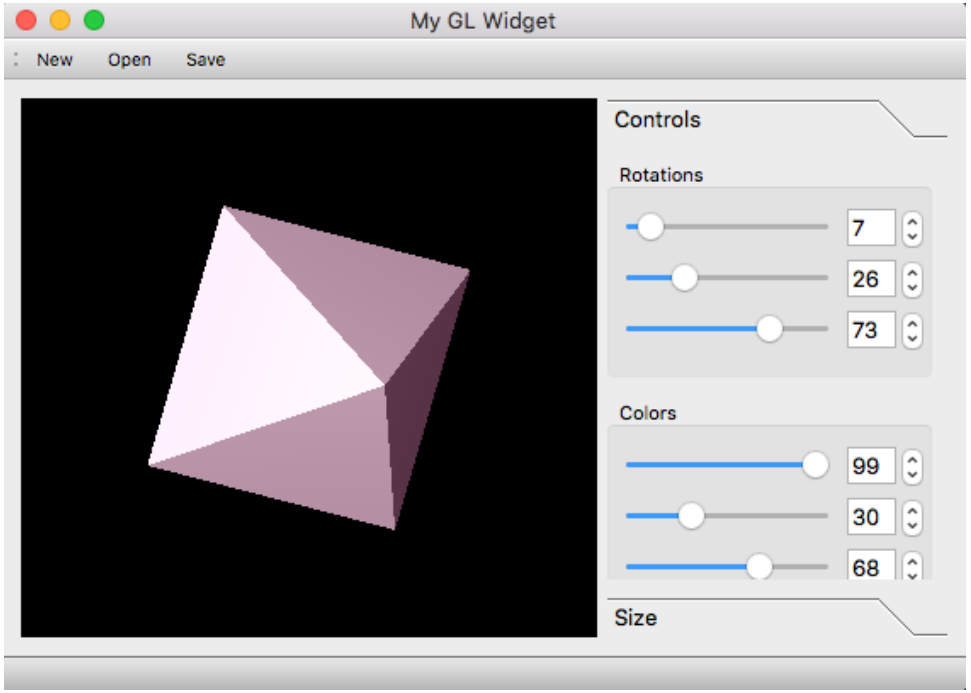
```
QGraphicsScene scene;

auto * rect = scene.addRect( QRectF(0,0,100,100) );
QGraphicsItem *item = scene.itemAt(50, 50);
 .....
QGraphicsView view( &scene );
view.show( );
```

OpenGL

Subclass of **QOpenGLWidget that redefines**:

- virtual void **initializeGL**( )
- virtual void **resizeGL**(int w, int h)
- virtual void **paintGL**( )

```
#include <QOpenGLWidget>
#include <QOpenFunctions>

class Box3D : public QOpenGLWidget,  protected QOpenGLFunctions {
   Q_OBJECT
public:
   Box3D(QWidget * parent);

public slots:
   void setXRot(int angle);
   void setYRot(int angle);
   void setZRot(int angle);

signals:
   void xRotChanged(int angle);
   void yRotChanged(int angle);
   void zRotChanged(int angle);

private:
   void initializeGL( ) override;
   void paintGL( ) override;
   void resizeGL(int w, int h) override;

   int xRot, yRot, zRot;
   GLfloat red, green, blue;
};
```



update( ) asks the widget to be repainted

OpenGL

```
#include "Box3D.h"

Box3D::Box3D(QWidget * parent) : QOpenGLWidget(parent) {
    xRrot = yRot = zRot = 0.;
    red = green = blue = 0.;
}

void Box3D::initializeGL( ) {
   initializeOpenGLFunctions( );     // ne pas oublier !
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    GLfloat lightPosition[4] = {0, 0, 10, 1};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    GLfloat lightColor[4] = {red, green, blue, 1};
    glLightfv(GL_LIGHT0, GL_AMBIENT, lightColor);
}
```

slots

```
void Box3D::setXRot(int rot) {
    if (rot != xRot) {
        xRot = rot;
        emit xRotChanged(rot);
        update( );
    }
}

void Box3D::setYRot(int rot) {
    if (rot != yRot) {
        yRot = rot;
        emit yRotChanged(rot);
        update( );
    }
}

void Box3D::setZRot(int rot) {
    if (rot != zRot) {
        zRot = rot;
        emit zRotChanged(rot);
        update( );
    }
}
```

OpenGL

```
void Box3D::paintGL( ) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    glTranslatef(0.0, 0.0, -10.0);
    glRotatef(xRot, 1.0, 0.0, 0.0);
    glRotatef(yRot, 0.0, 1.0, 0.0);
    glRotatef(zRot, 0.0, 0.0, 1.0);

    glBegin(GL_QUADS);
    glNormal3f(0,0,-1);
    glVertex3f(-1,-1,0);
    glVertex3f(-1,1,0);
    glVertex3f(1,1,0);
    glVertex3f(1,-1,0);
    glEnd();

    glBegin(GL_TRIANGLES);
    glNormal3f(0,-1,0.707);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(-1,-1,0);
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex3f(1,-1,0);
    glColor3f(1.0f, 0.0f, 1.0f);
    glVertex3f(0,0,1.2);
    glEnd();
    ....
```

```
    glBegin(GL_TRIANGLES);
    glNormal3f(1,0, 0.707);
    glVertex3f(1,-1,0);
    glVertex3f(1,1,0);
    glVertex3f(0,0,1.2);
    glEnd();

    glBegin(GL_TRIANGLES);
    glNormal3f(0,1,0.707);
    glVertex3f(1,1,0);
    glVertex3f(-1,1,0);
    glVertex3f(0,0,1.2);
    glEnd();

    glBegin(GL_TRIANGLES);
    glNormal3f(-1,0,0.707);
    glVertex3f(-1,1,0);
    glVertex3f(-1,-1,0);
    glVertex3f(0,0,1.2);
    glEnd();
}

void Box3D::resizeGL(int w, int h) {
    int side = qMin(w, h);
    glViewport((w - side) / 2, (h - side) / 2, side, side);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2, +2, -2, +2, 1.0, 15.0);
    glMatrixMode(GL_MODELVIEW);
}
```

OpenGL : MainWindow

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent)
{
    ...
    Box3D * box3d = new Box3D(parent);
    setCentralWidget(box3d);

    createSlider(box3d, SLOT(setXRot(int)));
    createSlider(box3d, SLOT(setYRot(int)));
    createSlider(box3d, SLOT(setZRot(int)));
    ...
}

void MainWindow::createSlider(Box3D * box3d,
                              const char * slot)
{
    QSlider * slider = new QSlider(QSlider::Horizontal, this);

    connect(slider, SIGNAL(valueChanged(int)), box3d, slot);
}
```

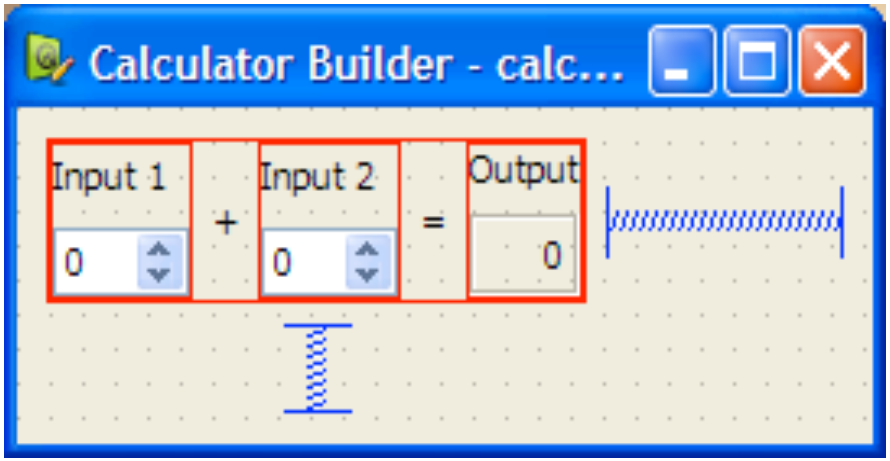
```
int main(int argc, char **argv)  {
    QApplication app(argc, argv);

    QSurfaceFormat format;
    format.setDepthBufferSize(24);
    format.setStencilBufferSize(8);
format.setProfile(QSurfaceFormat::CoreProfile);
    QSurfaceFormat::setDefaultFormat(format);

    MainWindow mainwin;
    mainwin.setWindowTitle("OpenGL Box");
    mainwin.show( );
    return app.exec( );
}
```

Type of a slot

Qt Designer

Files

- **calculator.pro**
- **calculator.ui**
- calculator.h
- calculator.cpp
- main.cpp



calculator.pro

```
TEMPLATE = app
FORMS = calculator.ui
SOURCES = main.cpp
HEADERS = calculator.h
```

calculator.ui

- fichier XML

Qt Designer

calculator.h

```
#include "ui_calculator.h"

class Calculator : public QMainWindow {
    Q_OBJECT
public:
    Calculator(QWidget * parent = 0);
private :
    Ui::Calculator * ui;
};
```

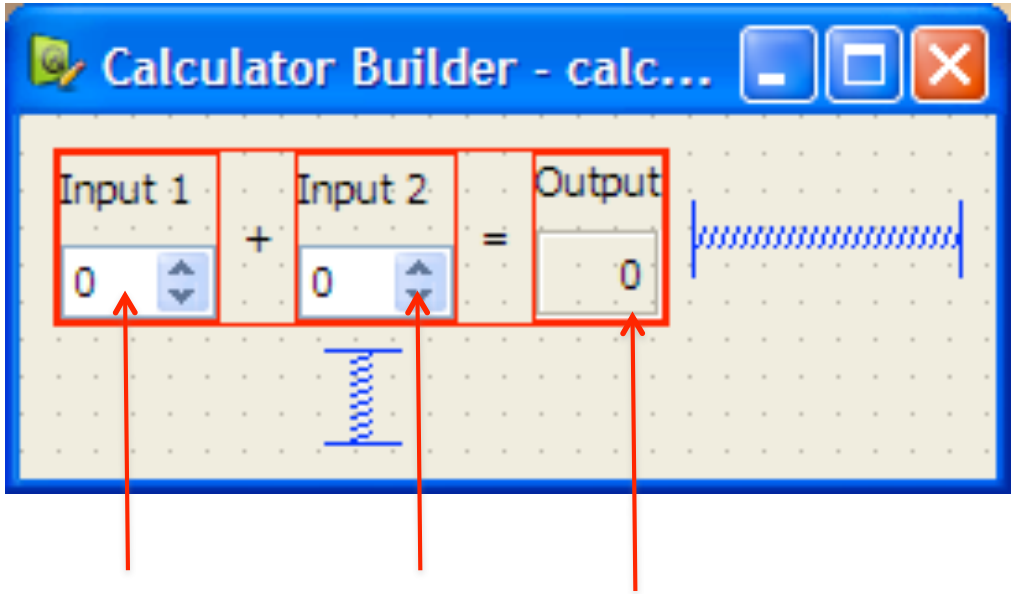
calculator.cpp

```
#include "calculator.h"

Calculator::Calculator(QWidget * parent) :
    QMainWindow(parent),
    ui (new Ui::Calculator)
{
    ui->setupUi(this);
    .......
}
```

ui points to an object that contains the widgets created by Designer

Linking with source code

How can we display the **result of a calculation** in a gadget created using **Designer**?



spinbox1 **spinbox2** **result**

names given to these widgets in **Designer**

```
#include "ui_calculator.h"

class Calculator : public QMainWindow {
    Q_OBJECT
public:
    Calculator(QWidget *parent =0);
private :
    Ui::Calculator * ui;
private slots :
    void on_spinbox1_valueChanged(int value);
    void on_spinbox2_valueChanged(int value);
};
```

Hidden file
ui_calculator.h

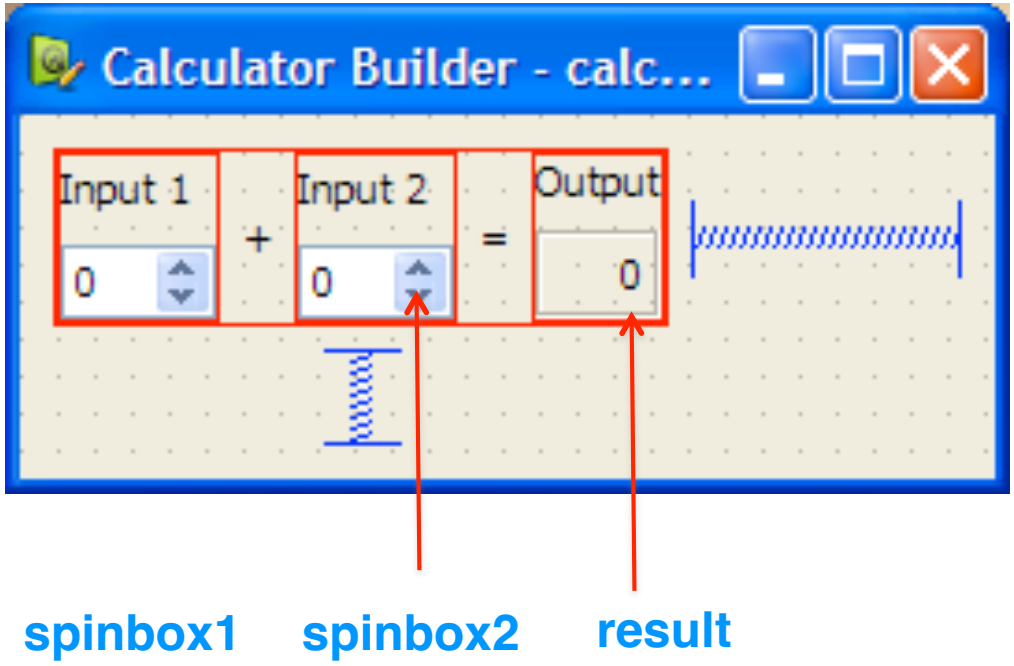
"auto-connect" slots

```
#include "calculator.h

void Calculator::on_spinbox1_valueChanged(int value) {
    QString res = QString::number(value);
    // or: QString res = QString::number(ui->spinbox1->value());
    ui->result->setText(res);
}
```

result, spinbox1, etc. are:

- the **names** of the widgets in **Designer**
- variables declared in ui_calculator.h

Auto-connection



- Name of the gadget in **Designer:** spinbox1
- Name of the signal: valueChanged
- => the slot on_spinbox1_valueChanged() is auto-connected

qFindChild & dynamic loading

```
class Calculator : public QWidget {
    Q_OBJECT
public:
    Calculator(QWidget *parent = 0);
private:
    QSpinBox * spinbox1{};
    QSpinBox * spinbox2{};
    QLabel * result{};
};
```

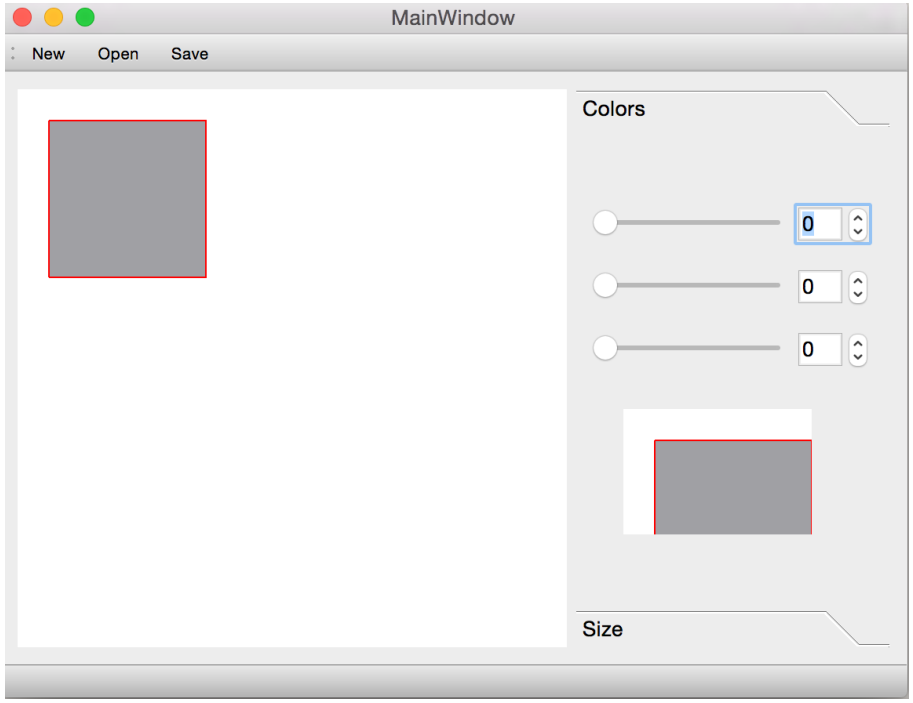
result, spinbox1, etc. can also be retrieved using qFindChild

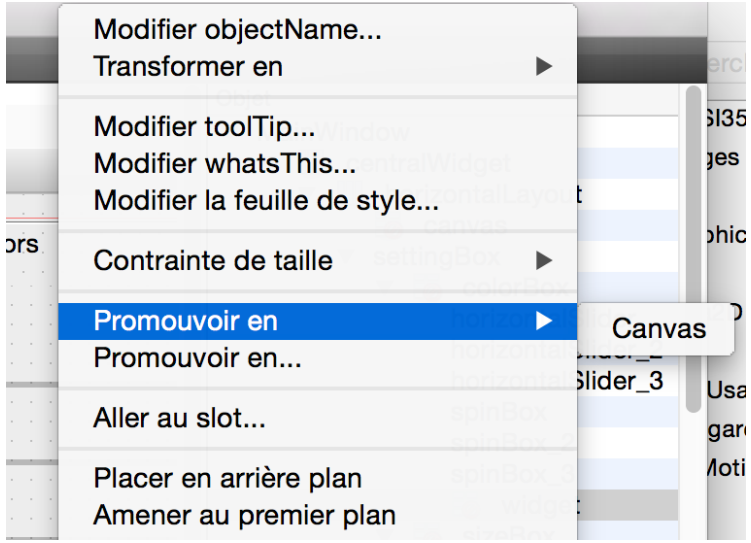
```
#include <QtUiTools>
#include "calculator.h

Calculator::Calculator(QWidget *parent) : QWidget(parent) {
    QUiLoader loader;
    QFile file(":/forms/calculator.ui");
    file.open(QFile::ReadOnly);
    QWidget * widget = loader.load(&file, this);
    file.close();

    spinbox1 = qFindChild<QSpinBox*>(this,  "spinbox1");
    spinbox2 = qFindChild<QSpinBox*>(this, "spinbox2");
    result = qFindChild<QLabel*>(this, "result");
    ....
}
```

« Promotion »

Promote

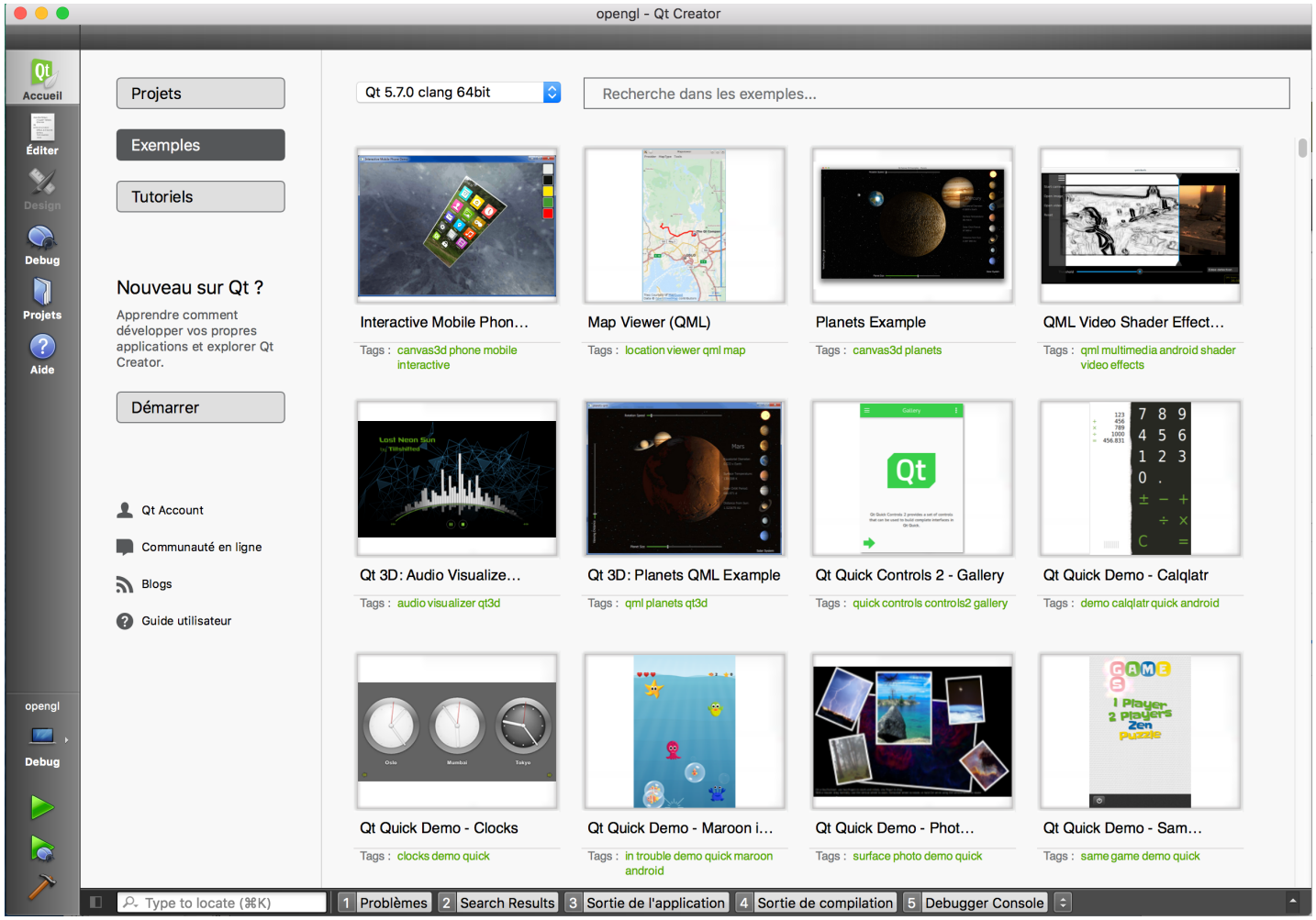
- Allows using **client objects** in **Designer**





Examples

QtCreator > Accueil



Performance issues

Flicker

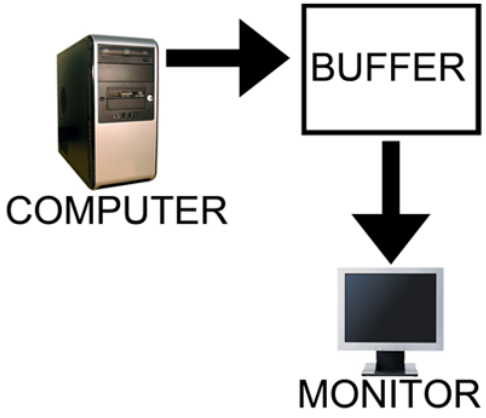
Flickering

- https://www.youtube.com/watch?v=QoZ8L1quWnc

Double buffering

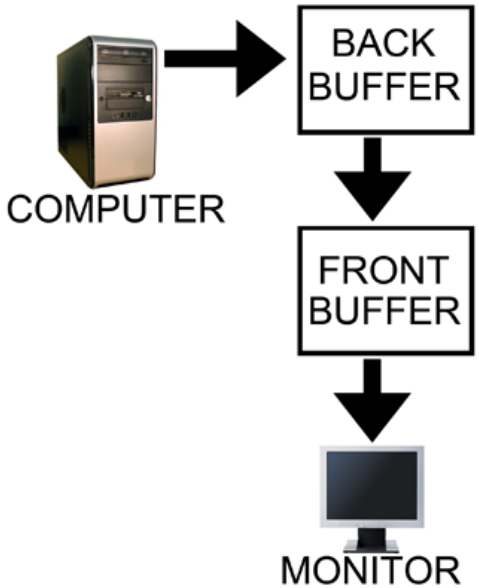
Problem

- Drawing performed directly on the screen
- The eyes perceive changes



Double buffering

- 1) Draw in the **back buffer** (hidden)
- 2) Copy in the **front buffer** (on the screen)

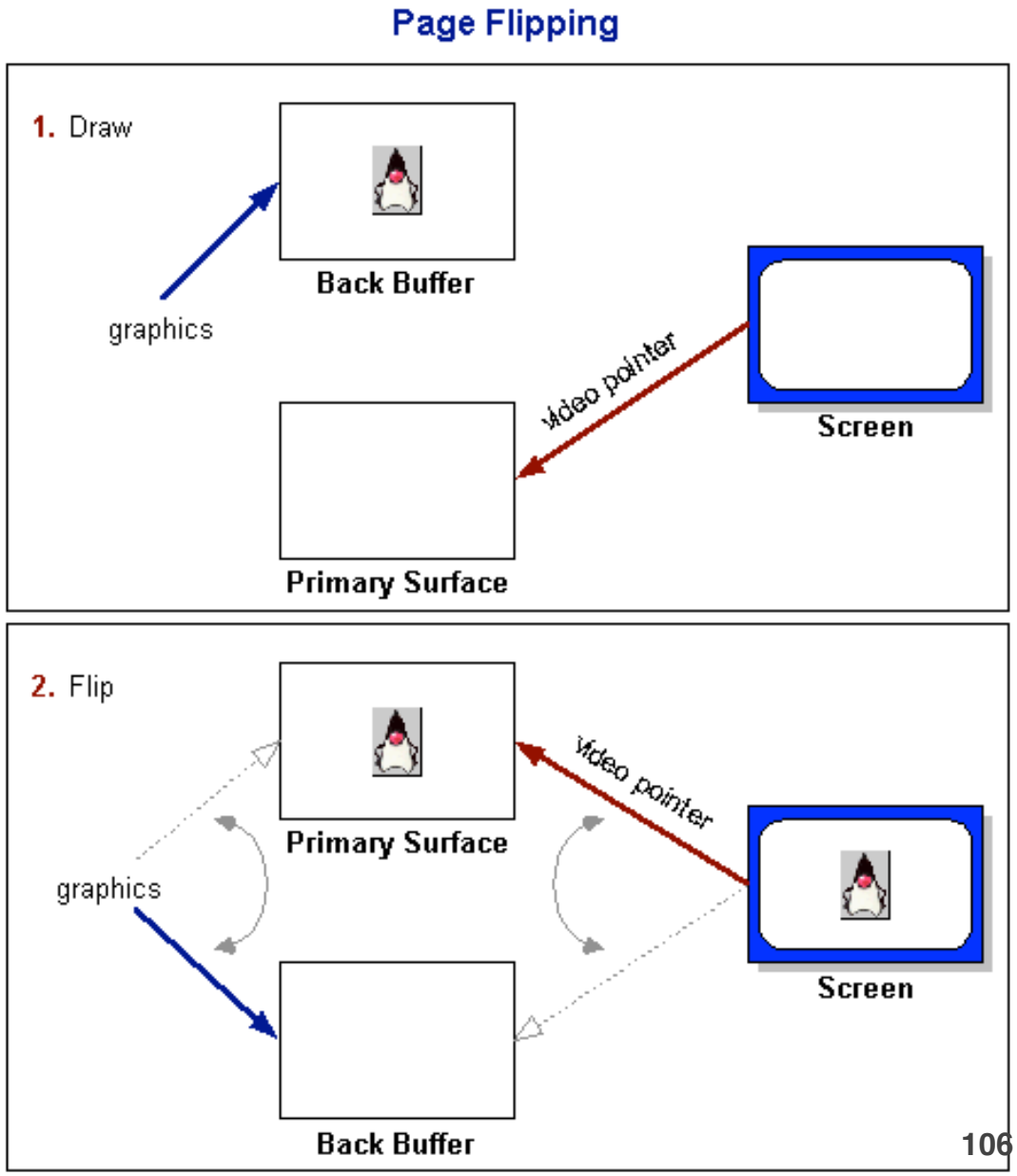


AnandTech

Page flipping

Alternate solution

- **exchanges** buffers instead of copying them

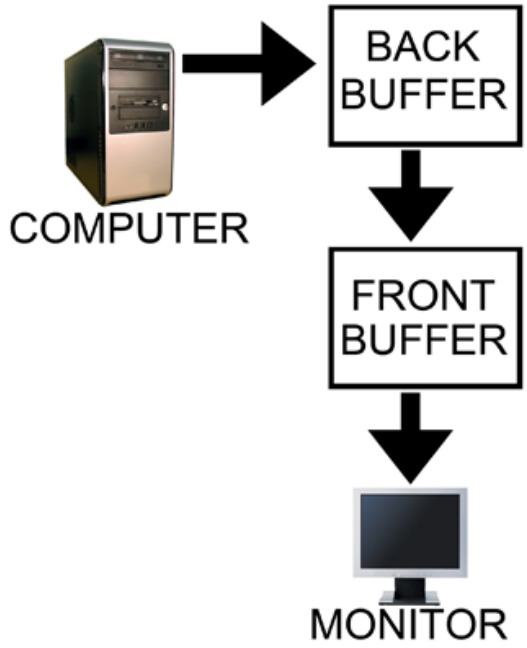


Oracle/JavaSE

Tearing

Tearing

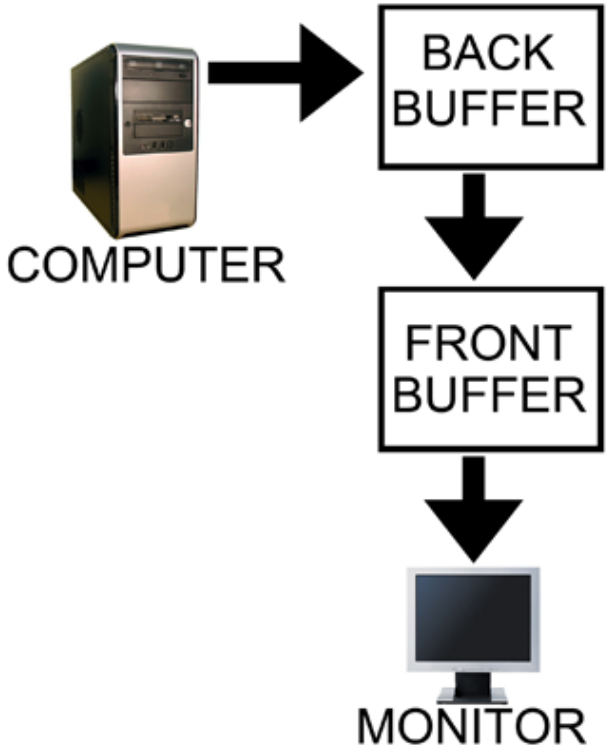
- https://www.youtube.com/watch?v=1nFBtTq0DHo



Tearing

Problem

- The back buffer is copied **too early!**
- A **mixture** of 2 successive frames appear on the screen



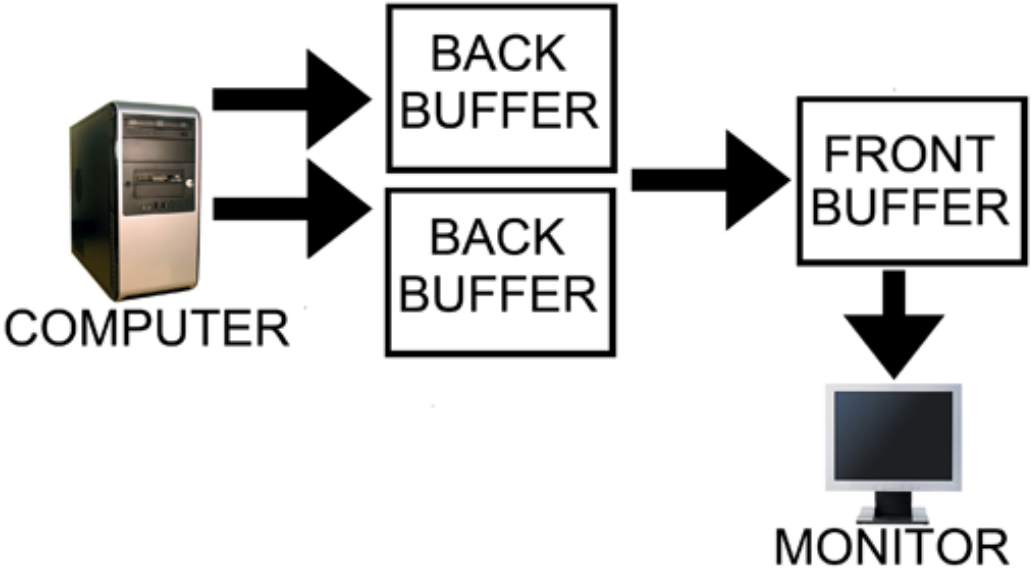
Synchronization needed !

- **Adaptive VSync** (Vertical synchronization) or similar
- Drawback: slower (input lag)

Multiple buffering

Faster

- One of the **2 back buffers** is always ready
- The font buffer can be updated **without waiting**



Latency

Problem

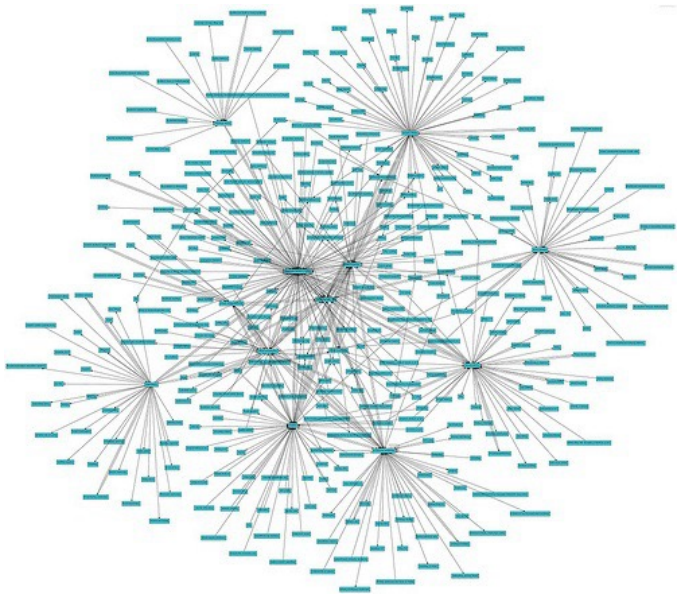
- **Delay** when drawing **interactively** with the mouse

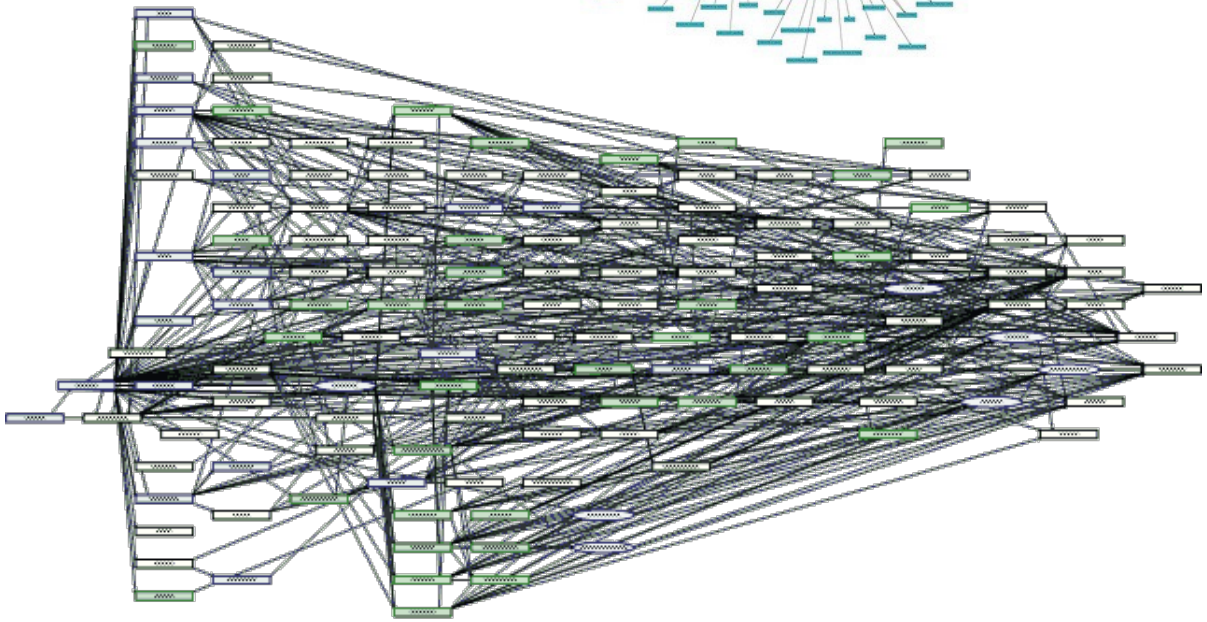
Reason

- The graphics card can't draw **so many** graphical primitives **fast enough**

- Note: double buffering can't help!

Solutions?

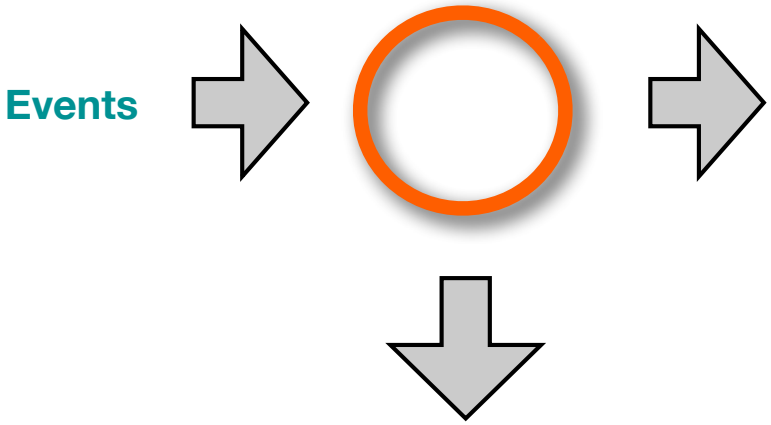




Latency

Solution 1: skip some updating steps

- Drawback: may be **jerky!**




```
void mouseMoveEvent(QMouseEvent* e)
    ......
    if (delay < MAX_DELAY) update();
}
```

Don't redisplay systematically
e.g. if delay from previous move event is too long

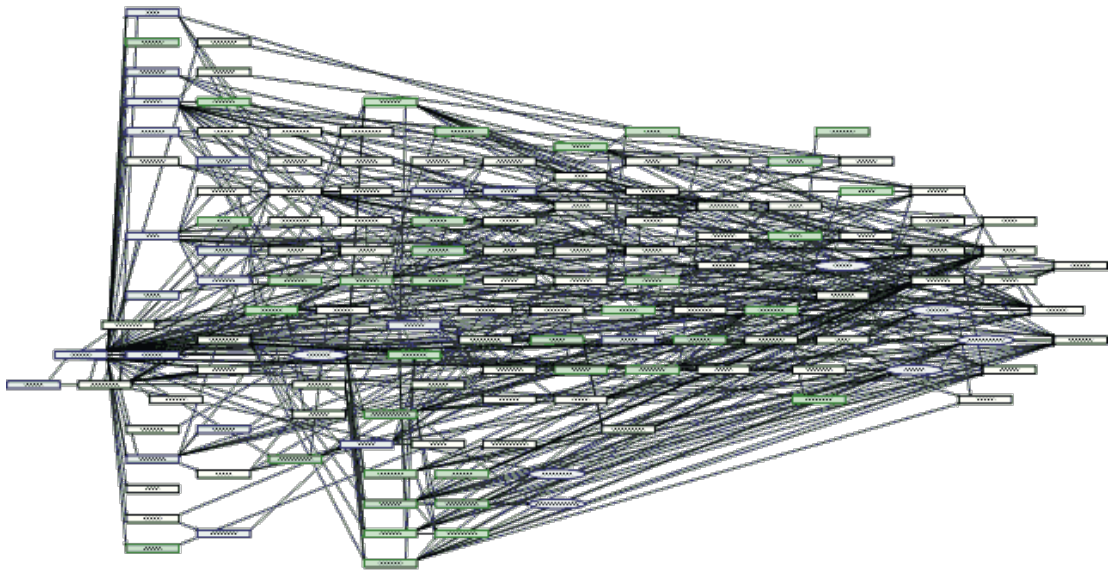
```
void paintEvent(QPaintEvent* e) {
    ......
}
```

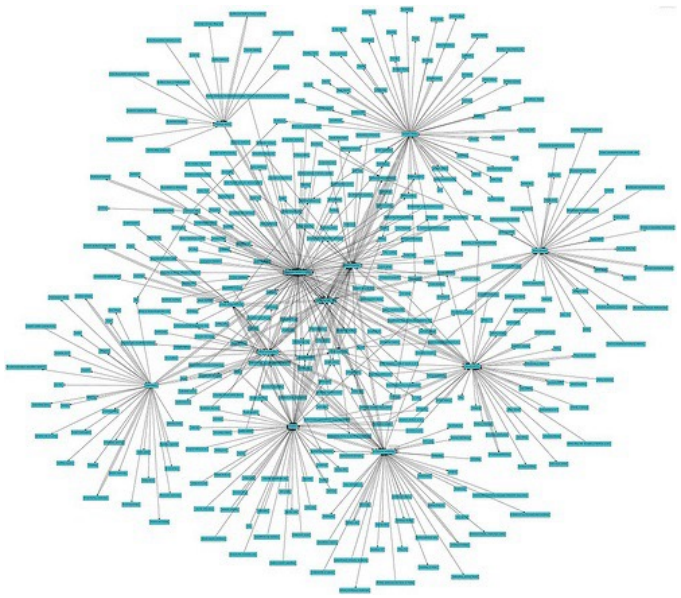
update asynchronously
e.g. use a **QTimer** to update

Latency

Solution 2: don't redraw all elements

- Problem: **double buffering** requires drawing **all elements!**

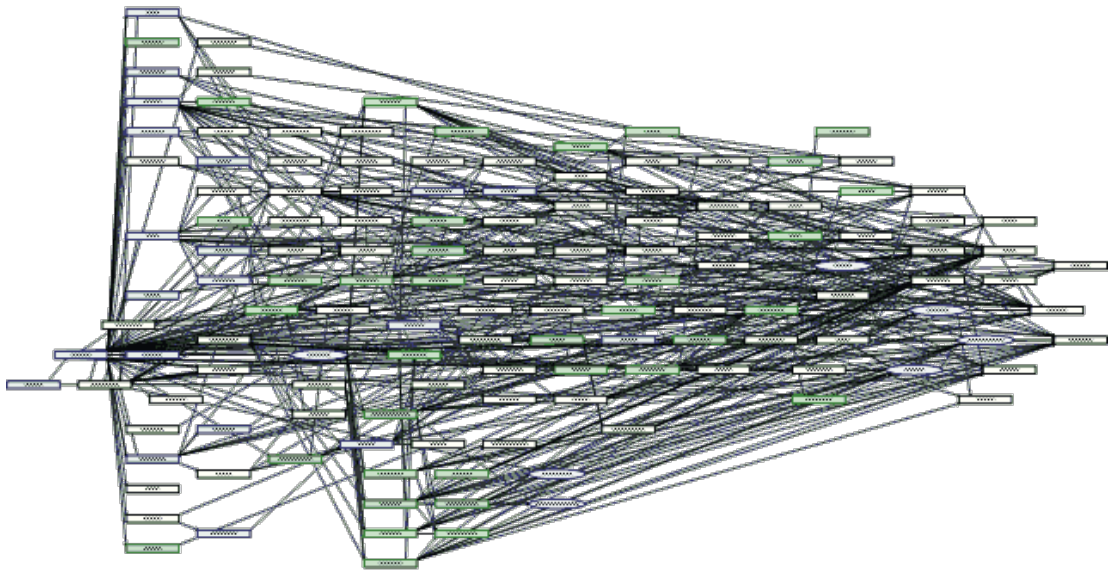


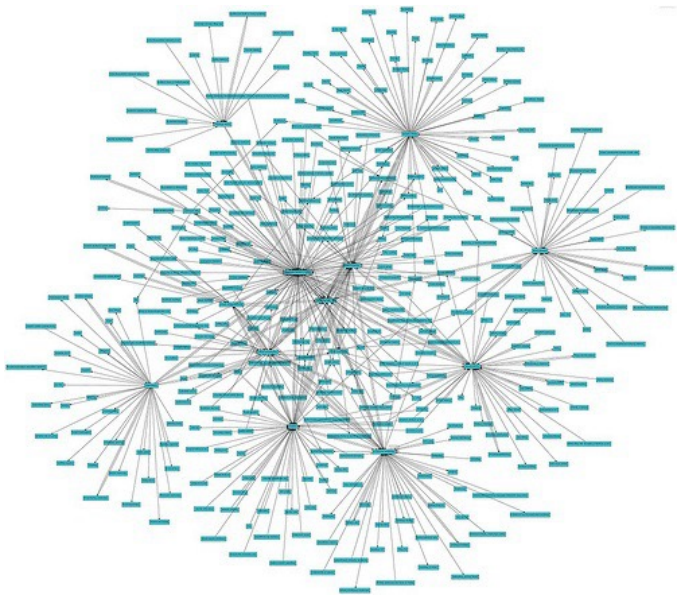


Latency

Solution 2: don't redraw all elements

- Problem: **double buffering** requires drawing **all elements!**
- Solution: **Backing store**
  - Store what **does not change** in an **image**
  - Display the **image** then the part that **has changed**



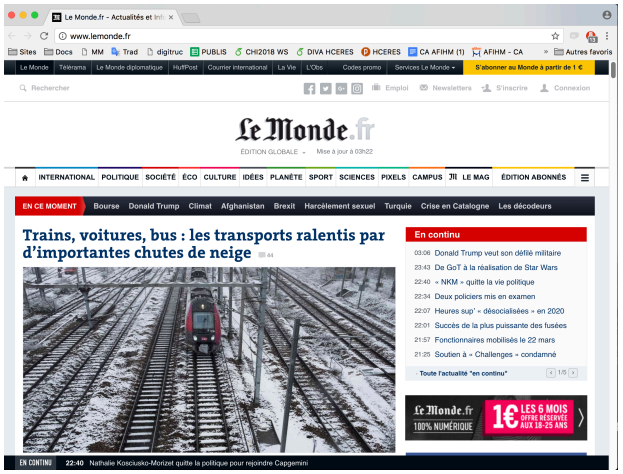


Concurrency

Events are processed sequentially

=> The UI **freezes** if a slot or a callback function :

- Takes **long time** to execute
- **Never returns**

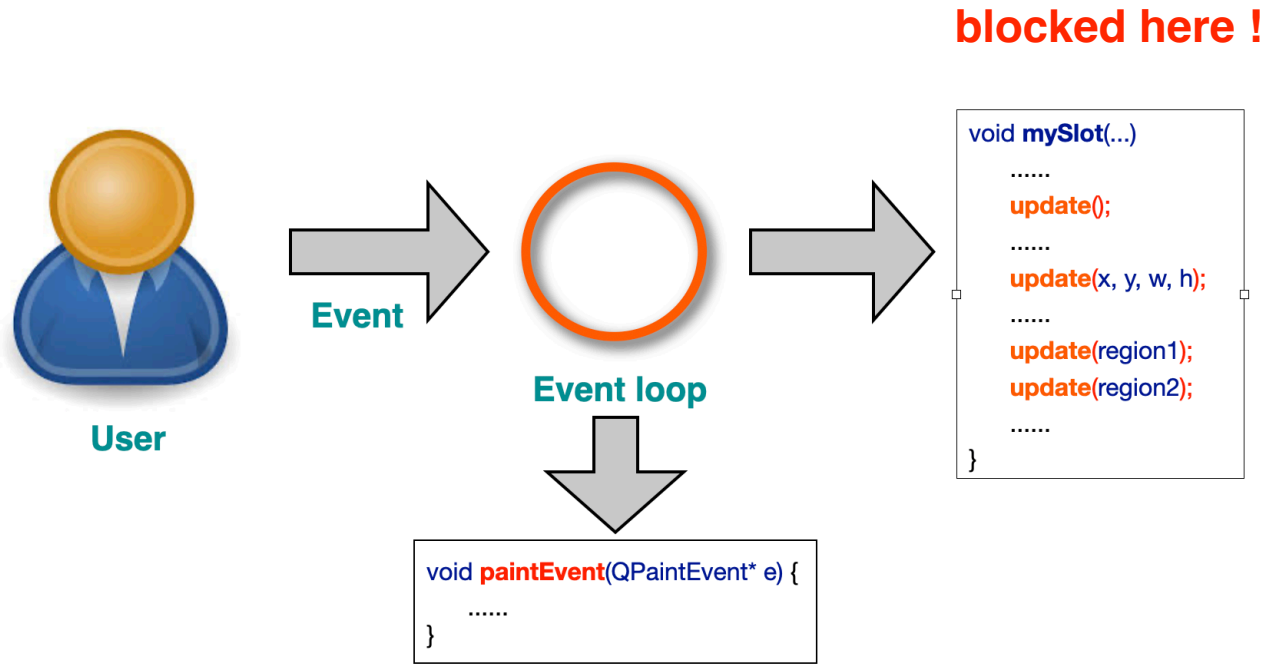


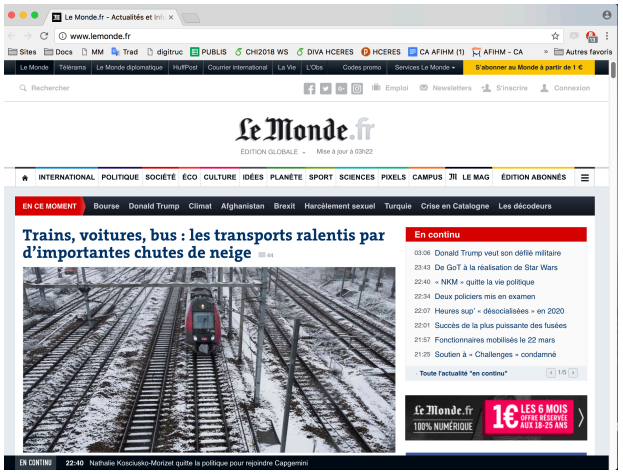
Typical examples

- Heavy **calculations**
- **Network** applications (using sockets, etc.)
  - *ex: Web browser*
- External **sensors**
  - *ex: Kinect*

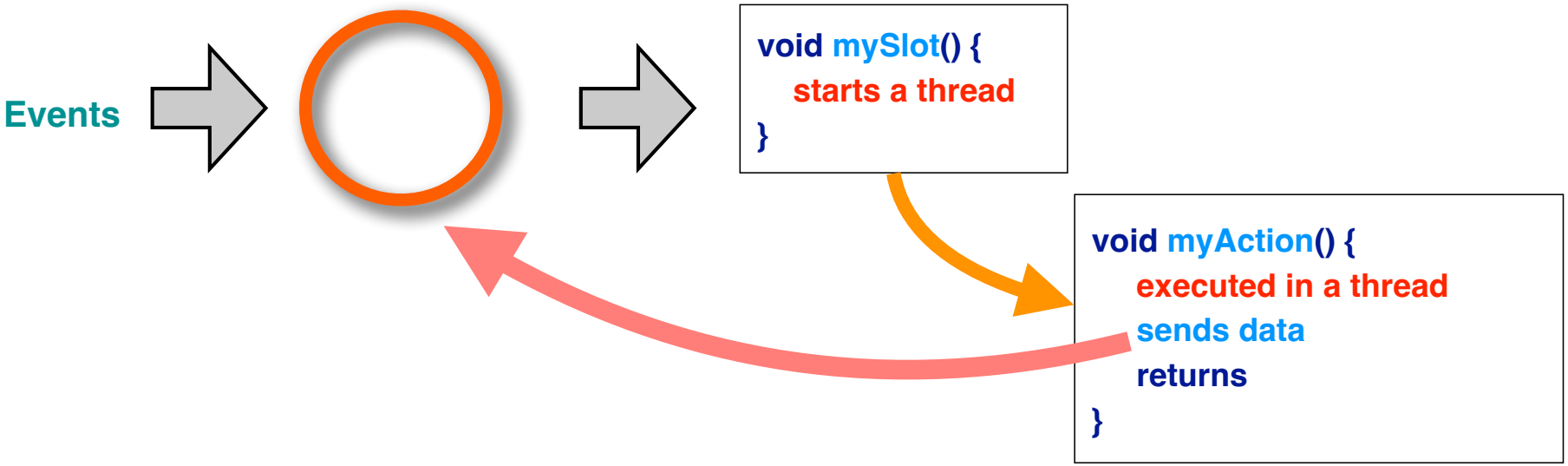
Concurrency

Reason : damaged/repaint model



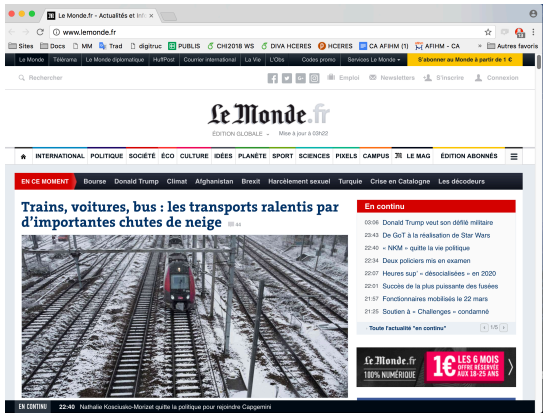


Threads

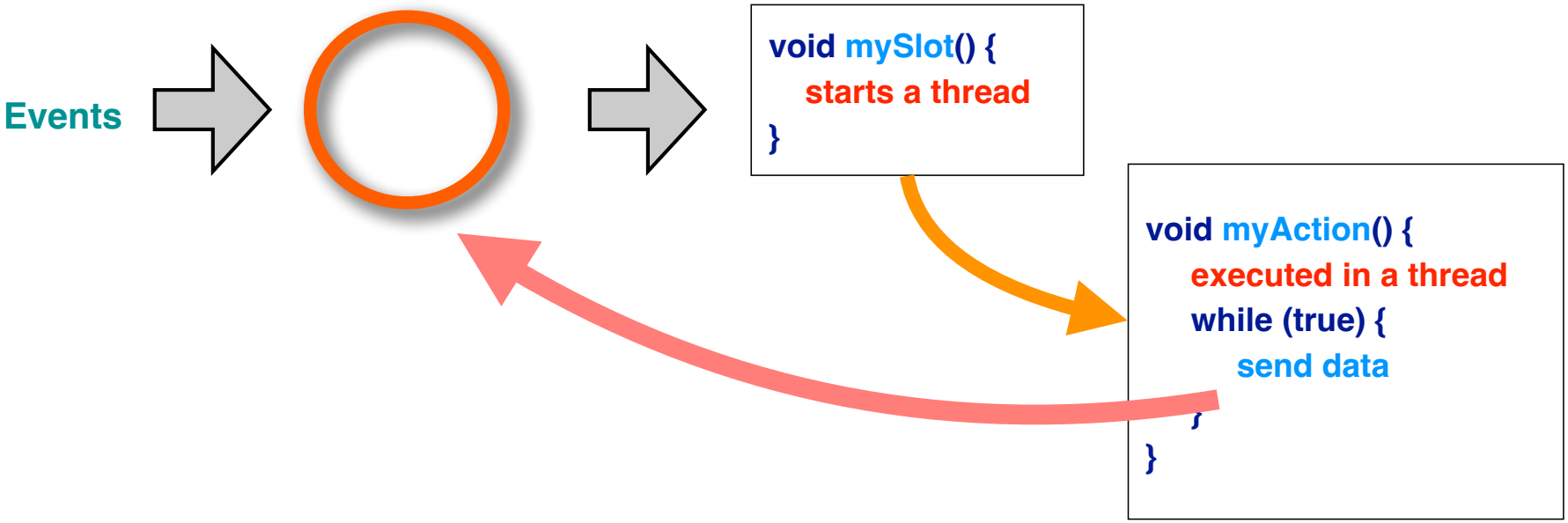


1st case: the slot launches a thread

- **mySlot()** returns **immediately**
- **myAction()** terminates **later**
  - *ex : Web page*



Threads



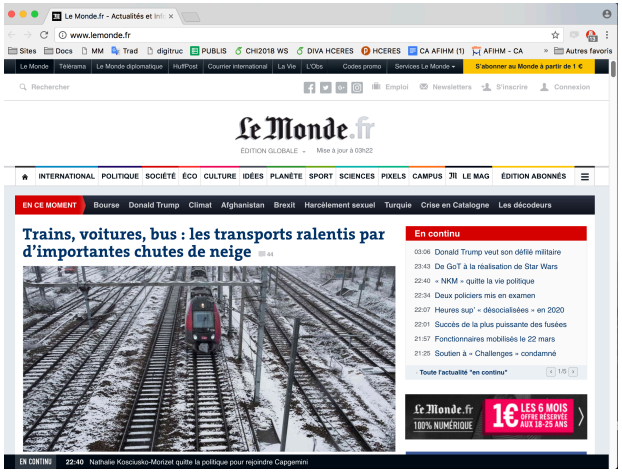
2nd case : the thread is constantly running

- A server, a device, etc. sends data **continuously**
- **myAction()** **runs a loop**
    - *ex : retrieve Kinect data*

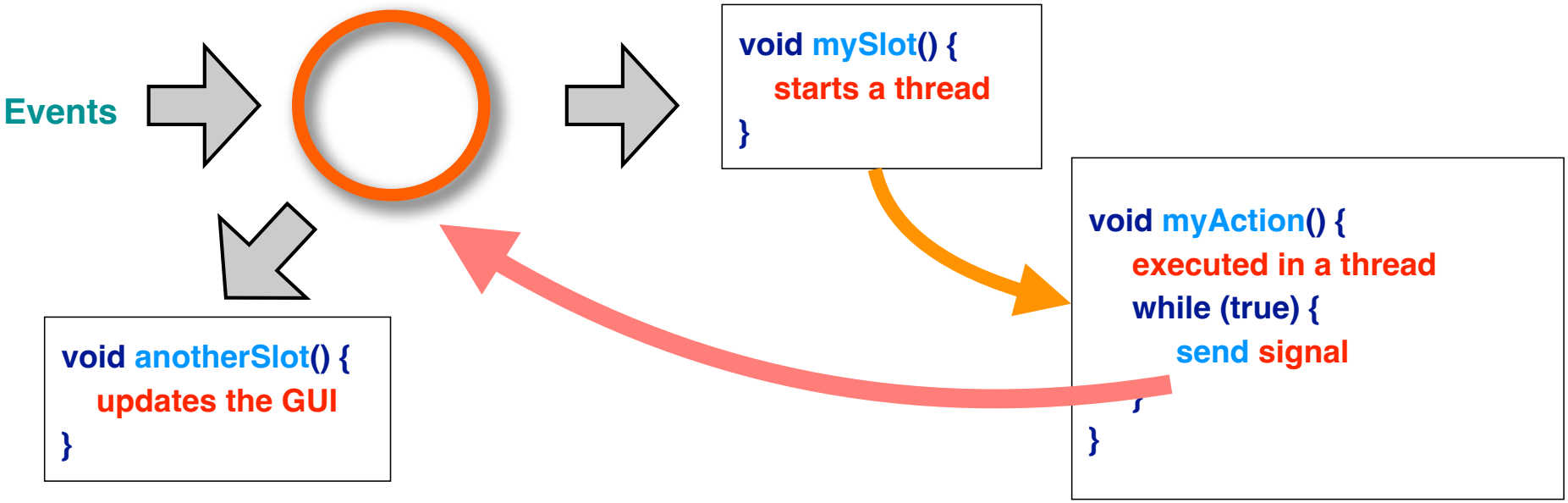
Threads

Problem: threads must not change the GUI !

- because the **event loop** "lives" in the **main thread**
- **common problem** with GUI toolkits



Threads



Solution

- **signals** can be sent **between threads**
- **connect**() provides a **queuing mechanism**
  - *optional argument:* ***Qt::ConnectionType***

```
connect(tracker, SIGNAL(dataReady(QString)),
        this, SLOT(showData(QString))
        Qt::QueuedConnection);
```

QThread

```
class Tracker : public QThread {
    Q_OBJECT
public:
    Tracker(const QString& host, int port);
signals:
    void dataReady(const QString& data);

private:
   void run( ) {        // executed in a new thread
       bool ok = true;
       QString data;

       while (ok) {
           data = .....;            // retrieve data
           emit dataReady(data);
       }
   }
};
```

```
class MyWindow : public QMainWindow {
    Q_OBJECT
    Tracker * tracker{};

public slots:
    void showData(const QString& data);

public:
   MyWindow(QWidget* par) : QMainWindow(par) {

       tracker = new DTracker(hostname, port);

       connect(tracker, SIGNAL(dataReady(QString)),
                   this, SLOT(showData(QString))
                   Qt::QueuedConnection);

       tracker->start();      // starts the thread
    ....
}
```

dataReady() and ShowData() live in different threads !