

Introduction à PyQt

Sylvain Malacria

<http://www.malacria.com/>

<mailto:sylvain.malacria@inria.fr>



Diapositives inspirées de Gilles Bailly

Documentation

<http://pyqt.sourceforge.net/Docs/PyQt5/>

Qu'est ce que Qt ?

Librairie graphique écrite en C++ par la société TrollTech

- ▶ Mécanisme pour interagir :
 - avec l'utilisateur (bouton, liste déroulante, ..)
 - avec le système (OpenGL, Xml, SQL, sockets, plugin...)

Multi-Plateforme

- ▶ Windows | Mac | Linux
- ▶ Android | iOS | WinRT | BlackBerry* | SailFish
- ▶ Embedded Linux | Embedded Android | Windows Embedded

Gratuit (GPL), mais dispose aussi d'une licence commerciale

Approche : Ecrire une fois, compiler n'importe où

Historique

1990	Haavard & Eirik ont l'idée de créer une librairie graphique
1995	Qt 0.9 Première distribution publique pour X11/Linux
1996	Qt 1.0 (licences commerciales et open source)
1999	Qt 2.0 en Open Source (Licence QPL)
2001	Qt 3.0 support Mac et <i>Qt designer</i>
2005	Qt 4.0 (licence GPL 2.0 pour toutes plateformes)
2008	Nokia rachète Trolltech (société mère de Qt) et ses 250 employés
2009	Distribution Qt 4.5 avec <i>QtCreator</i>
2008	Qt 4.5
2010	Qt 4.6 animation; QGraphicsScene; machine à état; gestes
2011	Qt est racheté par Digia (objectif <i>Android, iOS et Windows 8</i>)
2012	Qt 5.0 Qt Quick (création d'interfaces dynamiques)
2015	20ème anniversaire de la première distribution publique
2016	Support Qt Mobile (iOS, Android)

Pourquoi Qt?

Performance (C++)

Relativement Simple

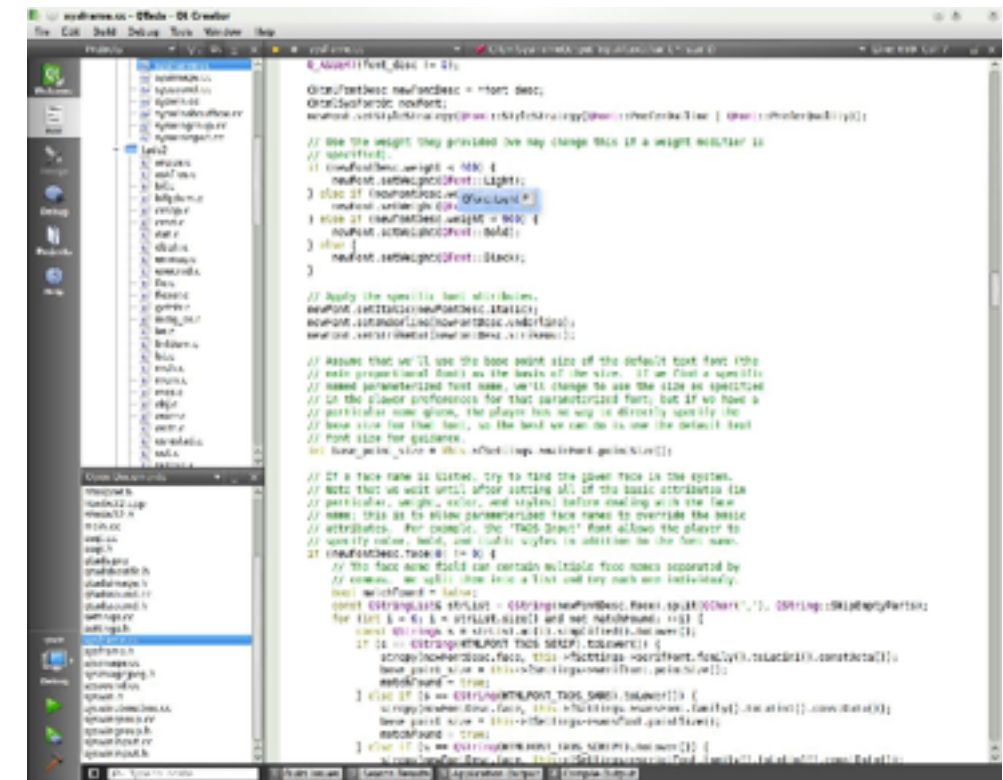
Gratuit (GPL) et code source

Nombreux outils

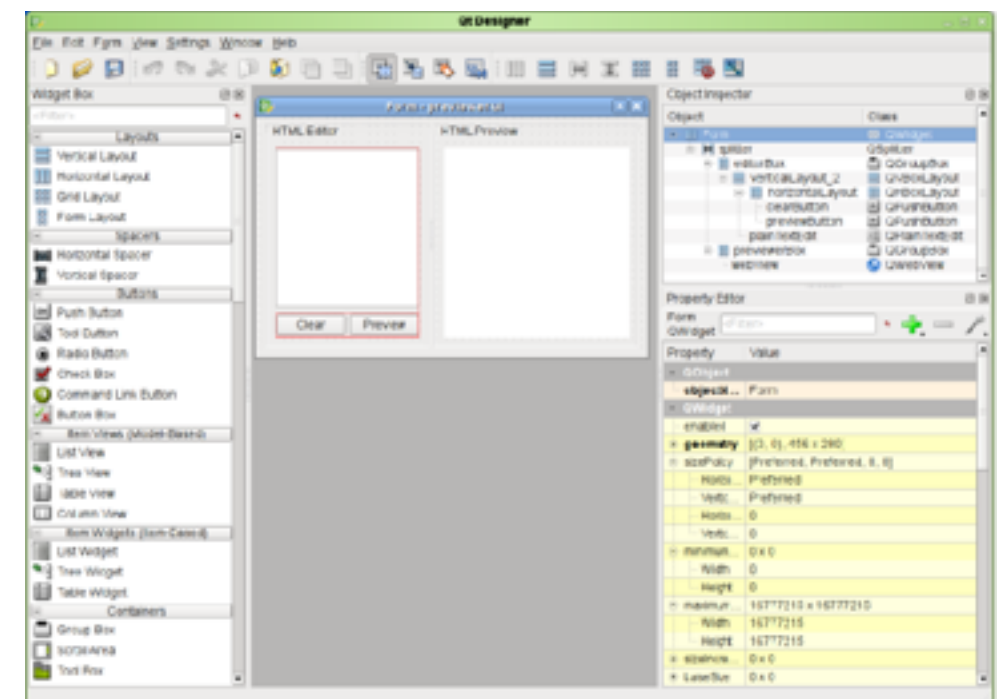
- ▶ Générateur d'interface : Qt Designer
- ▶ Internationalisation : Qt Linguist
- ▶ Documentation : Qt Assistant
- ▶ Exemples : Qt Examples
- ▶ Programmation : Qt Creator (eclipse)

Multi-Plateformes

- ▶ Look and feel simulé



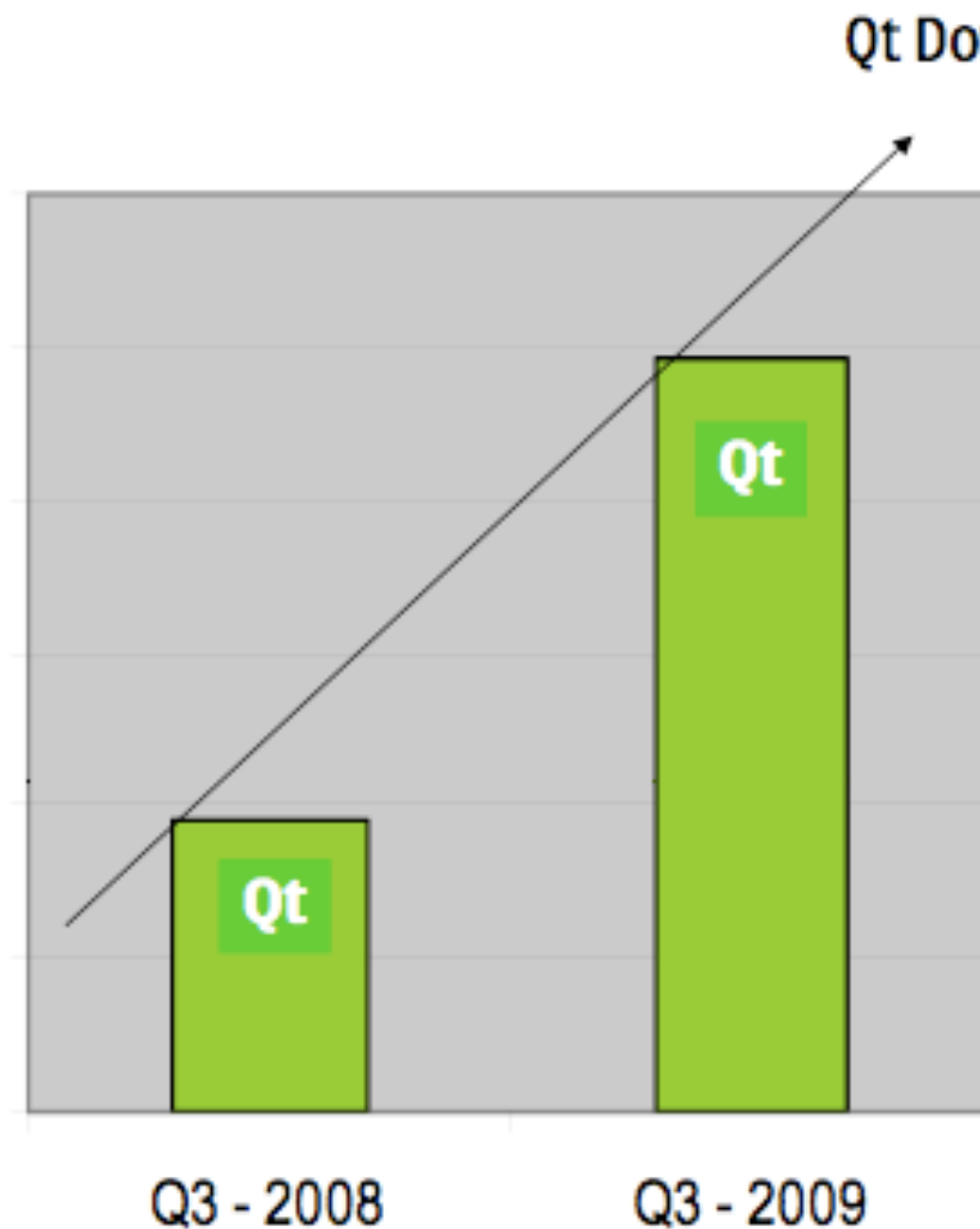
Qt creator



Designer

Aujourd'hui

Augmentation de 250% des téléchargements de la GPL



Qt 5 : 10 000 téléchargement par jour



Aujourd'hui

Utilisateurs de Qt :

- ▶ ESA, Nokia, Nasa, Adobe, Motorola, Google, ...

Bindings (java, **python**, c#)



Qt in Automotive
Infotainment



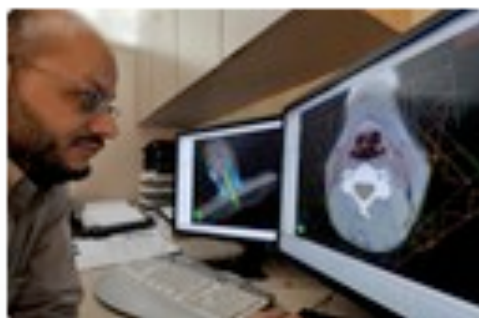
Qt in Aerospace



Qt in Home Media



Qt in IP Communication



Qt in Medical



Qt in Oil & Gas

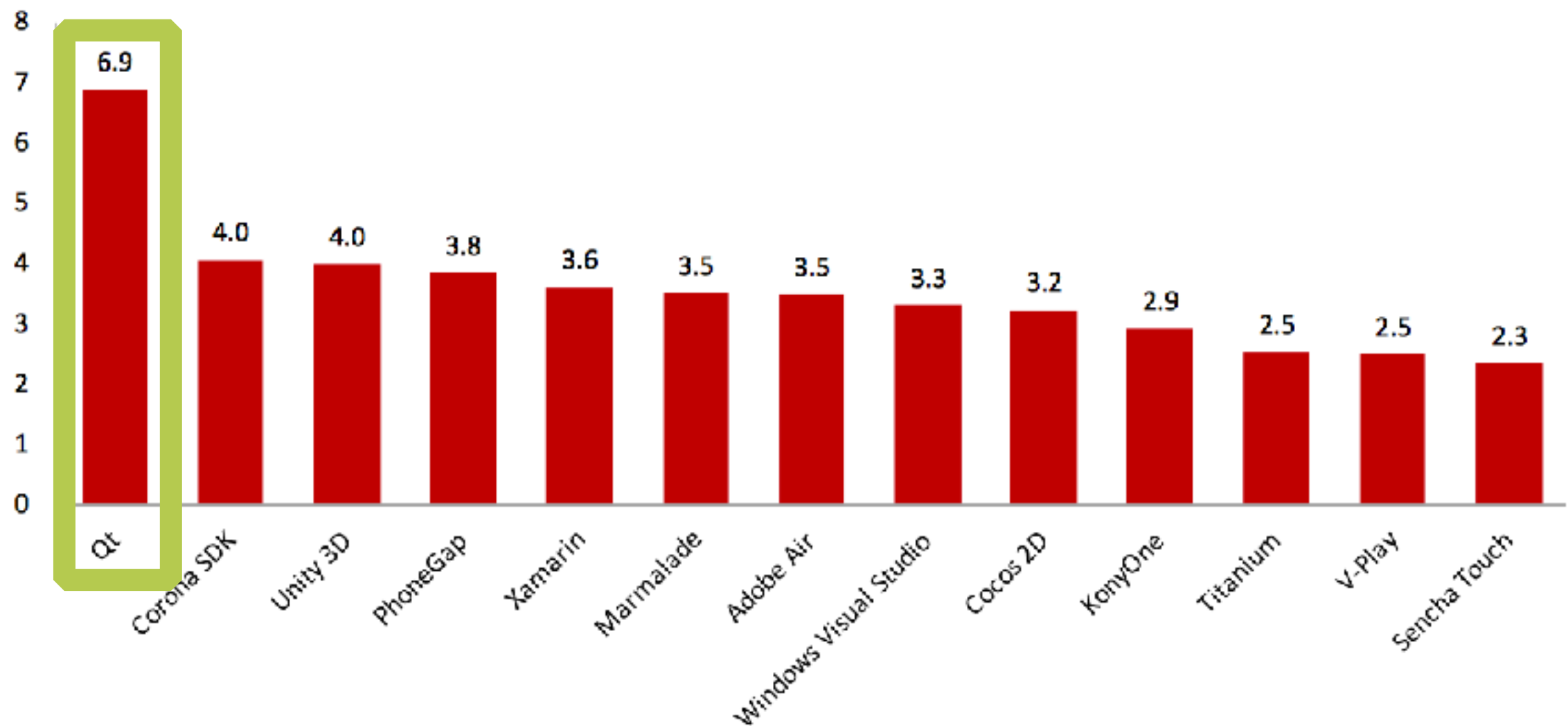


Qt in Visual Effects

Qt is the leader of cross-platform app development

Qt is the leader in true cross-platform app development. Users of Qt publish their apps on almost 7 different platforms, whereas all the other users release their apps on 4 or less platforms.

research2guidance 26: Average number of platforms which users publish their apps developed with a CP Tool on

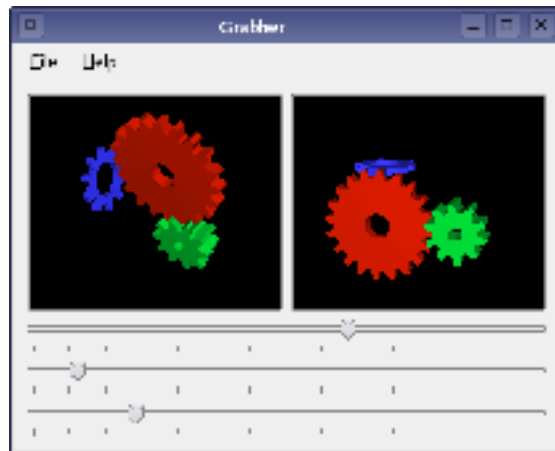


research2guidance 39: Top 10 Cost performance-ratio

Rank	Tool	Poor value or costly	Average	Okay or good value	# Ratings
1	Qt	-2%	0%	98%	104
2	Titanium	-6%	2%	92%	51
3	Unity	-4%	5%	91%	103
4	Corona SDK	-7%	2%	91%	97
5	Windows Visual Studio	0%	16%	84%	64
6	Cocos 2D	0%	17%	83%	54
7	Adobe Air	-4%	13%	83%	82
8	Xamarin	-7%	13%	80%	99
9	PhoneGap	-3%	17%	80%	88
10	KonyOne	-11%	11%	78%	55
Benchmark (Average all tools)		-5%	14%	81%	

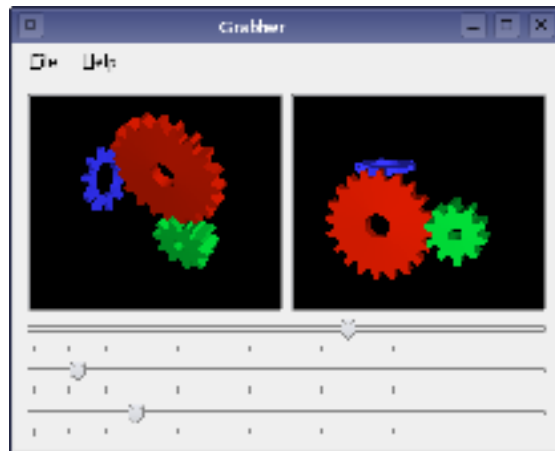
Research2guidance, CPT Benchmarking 2014

Aujourd'hui

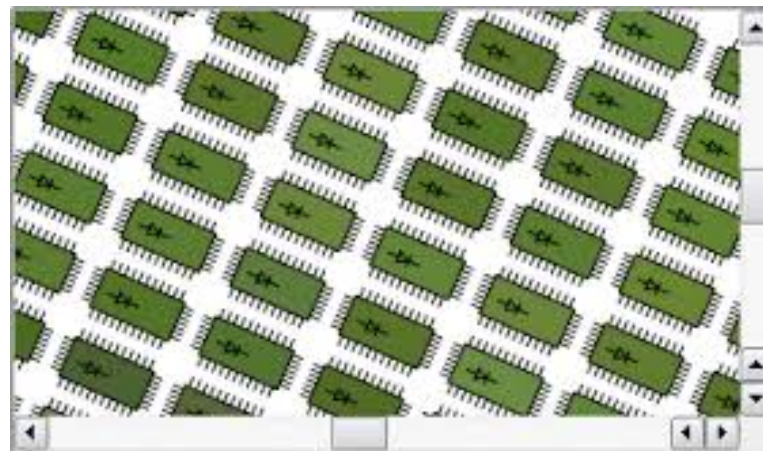


Qt Widgets

Aujourd'hui



Qt Widgets

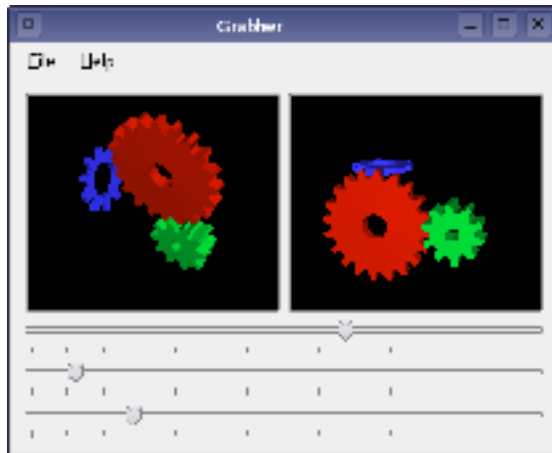


Qt Graphics view



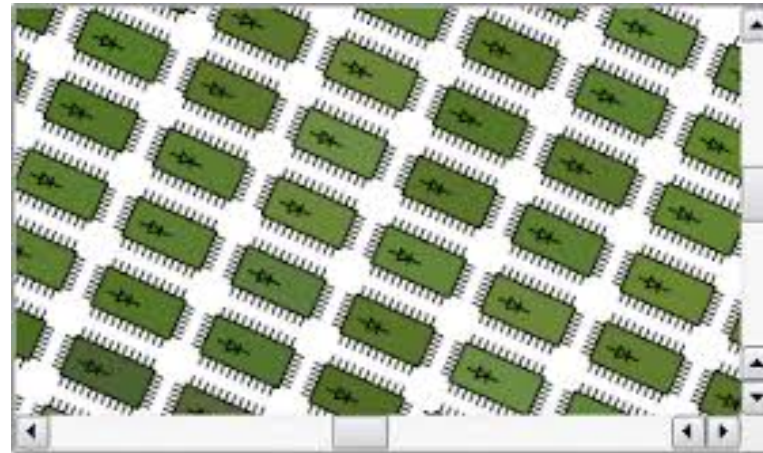
Qt quick /QML

Aujourd'hui



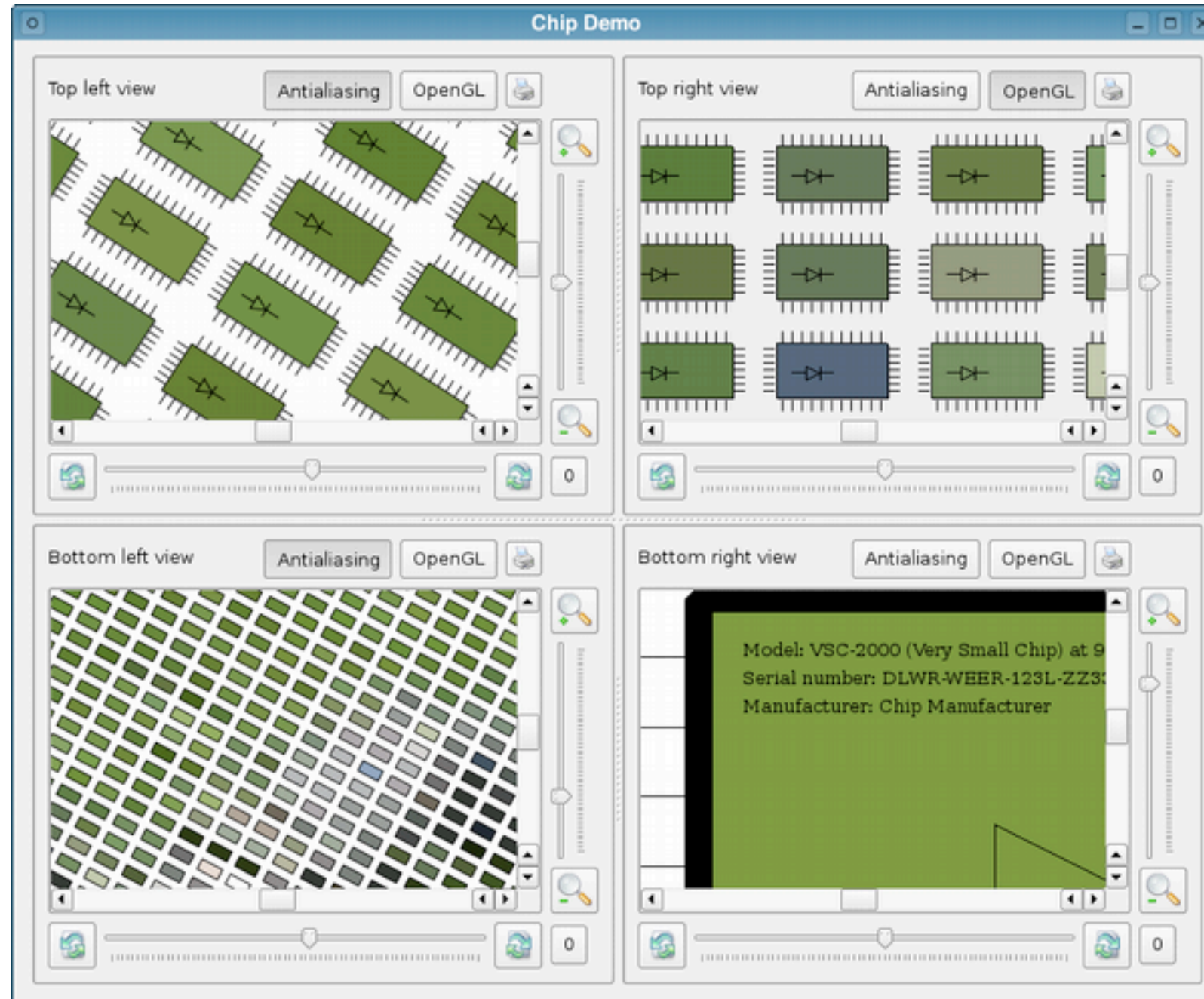
Qt Widgets

V.S.



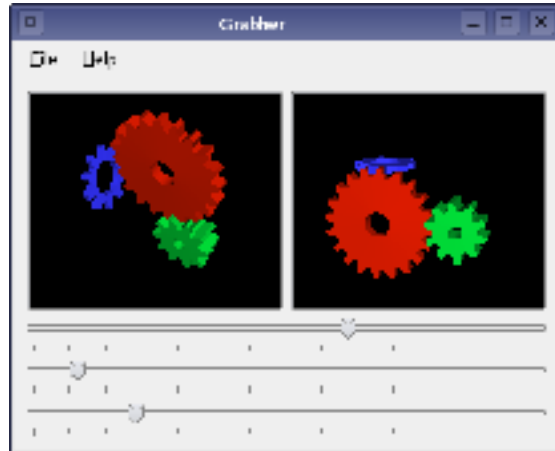
Qt Graphics view

- ▶ Widgets ne peuvent pas être *transformés*
- ▶ Widgets utilisent des coordonnées en *pixels* ; GraphicsItems en unités logiques (int vs doubles)
- ▶ Widgets peuvent être utilisés dans des layouts
- ▶ 4000000 widgets rament, mais 4000000 items fonctionnent très bien



<http://doc.qt.io/qt-5/qtwidgets-graphicsview-chip-example.html>

Aujourd'hui



Qt Widgets

V.S.



Qt quick/QML

```
import QtQuick 2.0

Rectangle {
    id: canvas
    width: 200
    height: 200
    color: "blue"

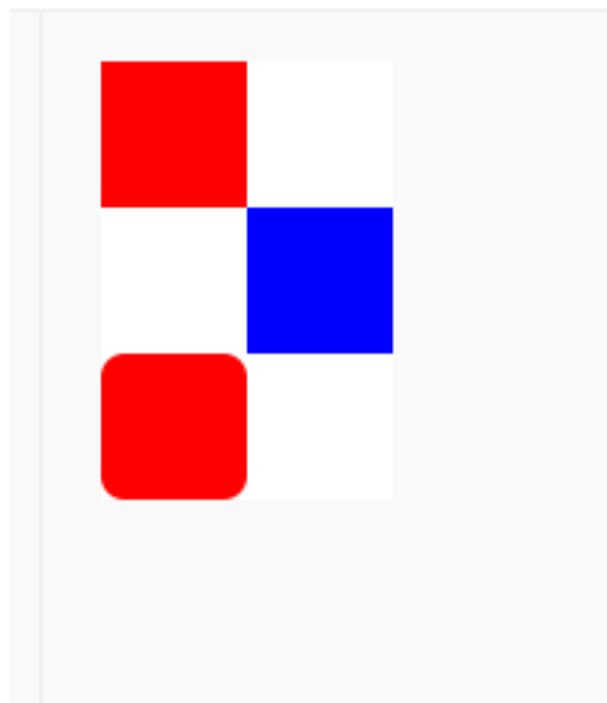
    Image {
        id: logo
        source: "pics/logo.png"
        anchors.centerIn: parent
        x: canvas.height / 5
    }
}
```

Language déclaratif

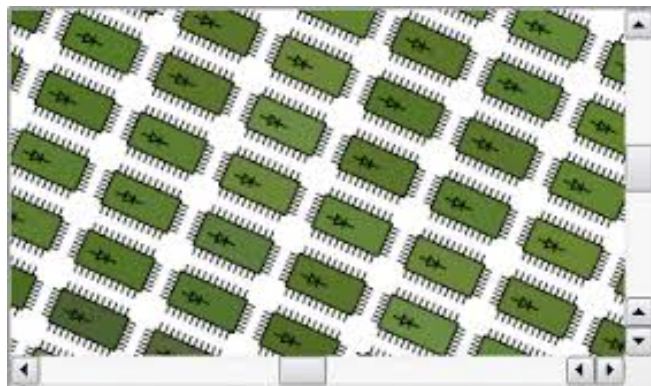
- ▶ QML est basé sur JSON (Language); QtQuick (library)
- ▶ QWidgets sont plus matures, flexibles and ont plus de fonctionnalités
- ▶ Qt Quick se concentre sur les animation and transition
- ▶ Qt Quick est (pour l'instant) plutôt pour dispositifs mobiles
- ▶ Qt Quick va (peut-être) remplacer QWidgets un jour
- ▶ Qt Quick est (peut-être) mieux pour les designers (non-informaticiens)

```
// application.qml
import QtQuick 2.3

Column {
    Button { width: 50; height: 50 }
    Button { x: 50; width: 100; height: 50; color: "blue" }
    Button { width: 50; height: 50; radius: 8 }
}
```



Aujourd'hui



V.S.



```
import QtQuick 2.0

Rectangle {
    id: canvas
    width: 200
    height: 200
    color: "blue"

    Image {
        id: logo
        source: "pics/logo.png"
        anchors.centerIn: parent
        x: canvas.height / 5
    }
}
```

QGraphicsView

Qt quick/QML

Language déclaratif

- ▶ Qt Quick Graphics engine only works with OpenGL
- ▶ Drawing complex shapes easier with QGraphicsView
- ▶ Qt Quick: QML & Javascript
- ▶ QML is more compatible with Widget

PyQt

Bindings Python v2 et v3 pour Qt

Développé par Riverbank Computing Limited

Existe pour les mêmes plateformes que Qt

Binding le plus populaire avec *PySide*

PyQt5 pour **Qt5**, PyQt4 pour Qt4

Licences différentes de Qt (GNU GPL 3 et license commerciale)

PyQt peut générer du code python depuis Qt Designer

Possibilité d'ajouter des widgets écrits en PyQt à Qt Designer



PyQt n'est pas QUE pour la programmation de GUI !!!

Objectifs de ce cours

Rappels programmation **Python**

Premiers pas avec **PyQt**

Introduction aux **Signaux** et **Slots** de Qt

Aperçu des principales classes de Qt

Python

Langage de programmation objet

Typage dynamique fort

Placé sous licence libre

Enormément de bibliothèques

Rapide d'utilisation

Nombreuses extensions destinées au calcul numérique

Syntaxe

Syntaxe C	Syntaxe Python
<pre>int factorielle(int n) { if (n < 2) { return 1; } else { return n * factorielle(n - 1); } }</pre>	<pre>def factorielle(n): if n < 2: return 1 else: return n * factorielle(n - 1)</pre>

Syntaxe

Syntaxe C	Syntaxe Python
<pre>int factorielle(int n) { if (n < 2) { return 1; } else { return n * factorielle(n - 1); } }</pre>	<pre>def factorielle(n): if n < 2: return 1 else: return n * factorielle(n - 1)</pre>

Attention aux tabulations !!!

Type dynamique fort

```
int a = 4
```

```
a = 4
```

```
type(a)
```

```
<class 'int'>
```

```
a = 4.1
```

```
type(a)
```

```
<class 'float'>
```

Quelques types de base

Booléen

Numériques

- ▶ int
- ▶ long
- ▶ float
- ▶ complex

Collections

- ▶ list
- ▶ tuple
- ▶ set
- ▶ dict
- ▶ etc.

Quelques types de base

Booléen

Numériques

- ▶ int
- ▶ long
- ▶ float
- ▶ complex

Collections

- ▶ **list**
- ▶ tuple
- ▶ set
- ▶ dict
- ▶ etc.

```
list1 = ['physics', 'chemistry', 1200]  
print(list1[0])
```

```
>>> 'physics'
```

```
list1.append("bla")  
print(list1[3])
```

```
>>> 'bla'
```

```
print(list1[1:3])
```

```
>>> ['chemistry', 1200]
```


Quelques types de base

Booléen

Numériques

- ▶ int
- ▶ long
- ▶ float
- ▶ complex

Collections

- ▶ list
- ▶ **tuple**
- ▶ set
- ▶ dict
- ▶ etc.

```
list1 = ['physics', 'chemistry', 1200]  
print(list1[0])
```

```
>>> 'physics'
```

```
list1.append("bla")  
print(list1[3])
```

```
>>> 'bla'
```

```
print(list1[1:3])
```

```
>>> ['chemistry', 1200]
```

Opérations de base sur collections (List, Tuple)

alphabetT = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k') ➡ Tuple
alphabetL = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k'] ➡ List

alphabetT2 = 'a', 'b', 'c', 'd', ('e', 'é', 'è'), 'f', 'g', 'h', 'i', 'j', 'k'

len(alphabetT) ➡ Nombre d'éléments
>>> 11

len(alphabetT2[4])
>>> 3

alphabetT[-2] ➡ Accès depuis la fin
>>> 'j'

for char in alphabetT: ➡ Boucle for each
 print(char)

for i in range(len(alphabetT)): ➡ Boucle itérative
 print(alphabetT[i])

for i in range(2, len(alphabetT)):
 print(alphabetT[i])

Entrées/Sorties fichiers

```
file = open("text.txt", "r+")  
text = file.read()  
file.write("blabla\n")  
file.close()
```

➡ (r,w,a,r+) read, write, append, read+write

Classes

```
class Voiture(Vehicule):
```

```
    #commentaire
```

```
    nbRoues=4
```

➡ Déclaration d'une classe Voiture qui hérite de véhicule

```
def __init__(self,marque,couleur):
```

➡ Déclaration d'un constructeur

```
    super().__init__()
```

➡ Appel du constructeur de la classe mère

```
    self.couleur=couleur
```

➡ Déclaration d'une variable d'instance

```
    self.marque=marque
```

➡ Déclaration d'une variable d'instance

```
def main():
```

➡ Déclaration conventionnelle d'une méthode main()

```
    audirouge = Voiture('audi','rouge')
```

```
    print(audirouge.couleur)
```

```
if __name__ == "__main__":
```

```
    main(sys.argv)
```

➡ Appel automatique *Main*

Documentation PyQt



<http://pyqt.sourceforge.net/Docs/PyQt5/>

Les principaux modules

QtCore

QtWidgets

QtGUI

QtBluetooth

QtOpenGL

QtScript/QtScriptTools

QtSql

QtSvg

QtWebKit

QtXml/QtXmlPatterns

QtMultimedia

QtSensors

Module QtCore

QObject

Type de base :

- ▶ QChar, QDate, **QString**, QStringList, Qtime,...

File systems :

- ▶ QDir, QFile,...

Container :

- ▶ QList, QMap, Qpair, QSet, QVector,...

Graphique :

- ▶ QLine, QPoint, QRect, QSize ...

Thread :

- ▶ QThread, Qmutex, ...

Autres :

- ▶ QTimer, ...

QString

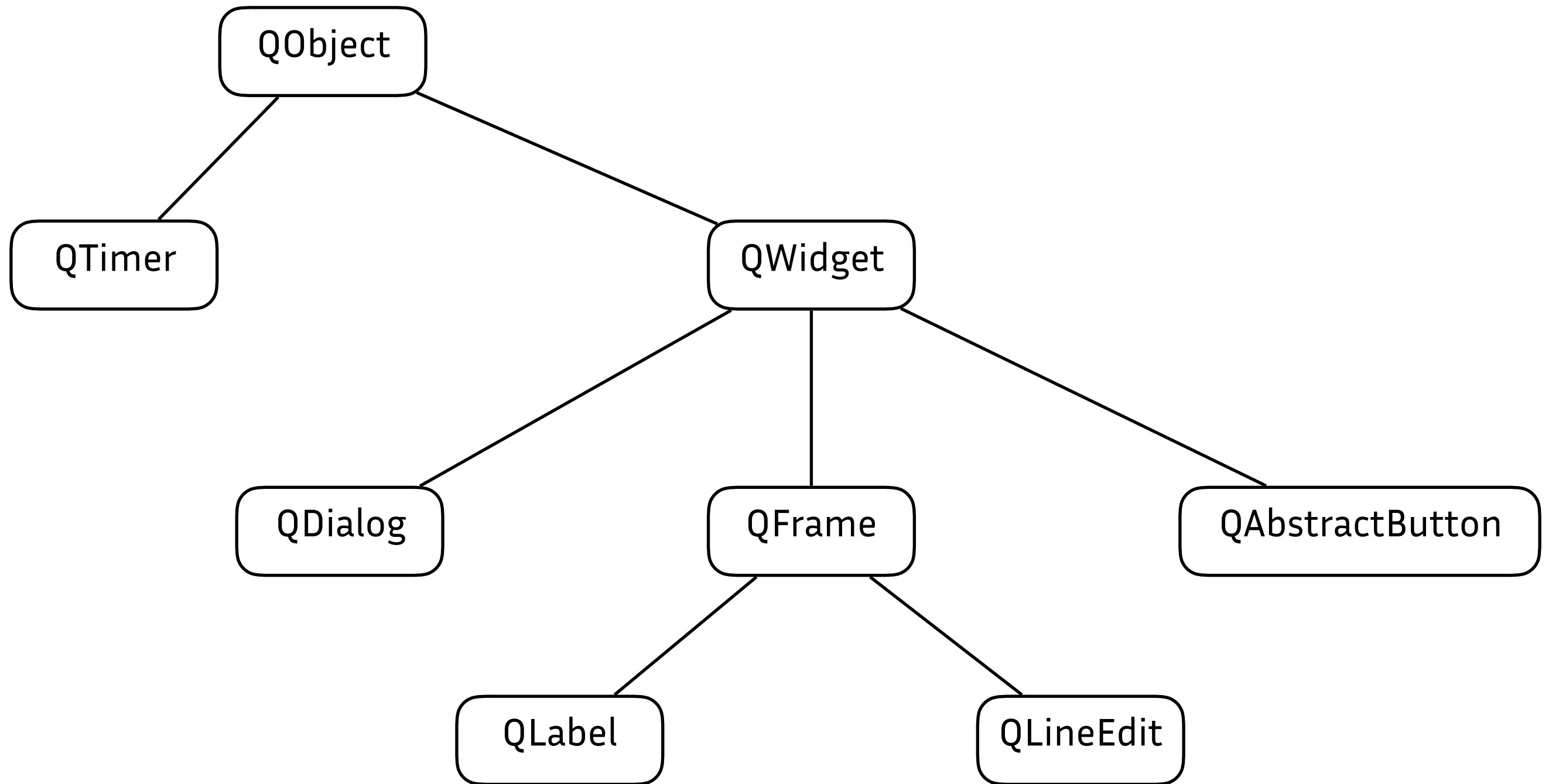
Codage Unicode 16 bits

- ▶ Suite de **QChars**
- ▶ 1 caractère = 1 **QChar** de 16 bits (cas usuel)
- ▶ 1 caractère = 2 **QChars** de 16 bits (pour valeurs > 65535)

Conversions d'une **QString** :

- ▶ **toAscii()** : ASCII 8 bits
- ▶ **toLatin1()** : Latin-1 (ISO 8859-1) 8 bits
- ▶ **toUtf8()** : UTF-8 Unicode multibyte (1 caractère = 1 à 4 octets)
- ▶ **toLocal8Bit()** : codage local 8 bits

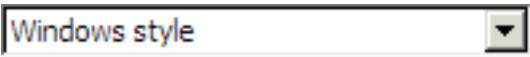
Principaux widgets



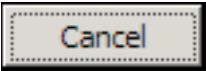
Module QtWidgets

Qwidget

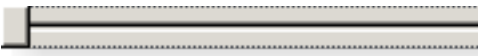
QComboBox



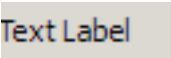
QPushButton



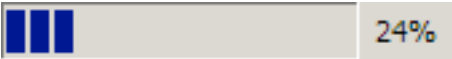
QSlider



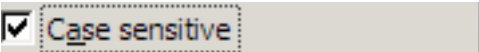
QLabel



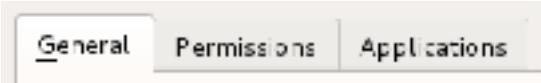
QProgressBar



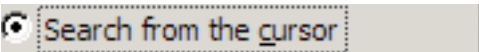
QCheckBox



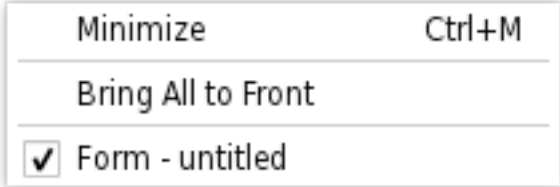
QTabBar



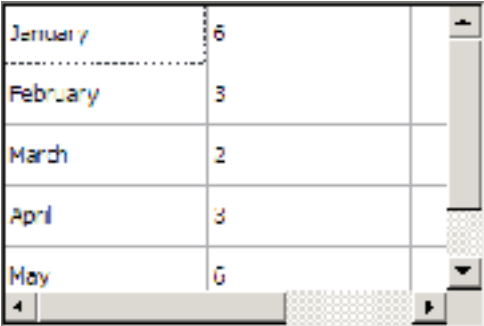
QRadioButton



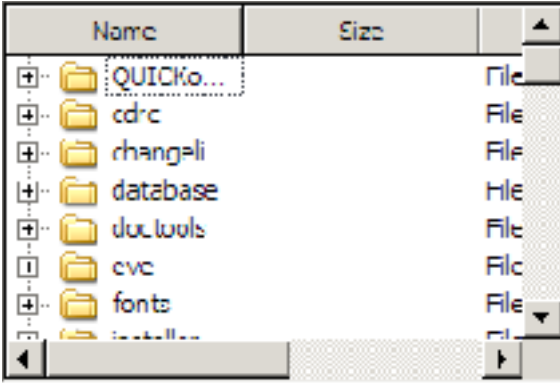
QMenu



QTableView



QTreeView

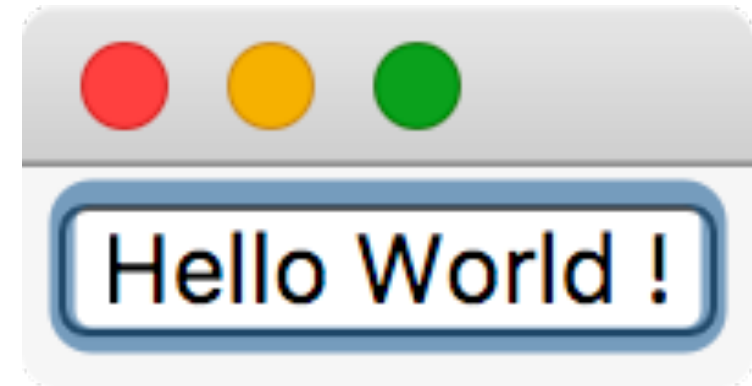


Simple window

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
import sys

def main(args) :
    app = QApplication(args)
    button = QPushButton("Hello World !", None)
    button.resize(100,30)
    button.show()
    app.exec_()

if __name__ == "__main__":
    main(sys.argv)
```



Simple window with button in widget

```
from PyQt5.QtCore import *  
from PyQt5.QtWidgets import *  
import sys
```

```
def main(args) :
```

```
    app = QApplication(args)
```

```
    widget = QWidget(None)
```

```
    widget.resize(400,90)
```

```
    button = QPushButton("Hello World !", widget)
```

```
    button.resize(100,30)
```

```
    widget().show()
```

```
    app.exec_()
```

```
if __name__ == "__main__":
```

```
    main(sys.argv)
```



Signaux et slots

Comment, à partir d'un « clic sur un bouton », je peux exécuter la partie correspondant à la logique de mon application ? (ex: fermer l'application) ??

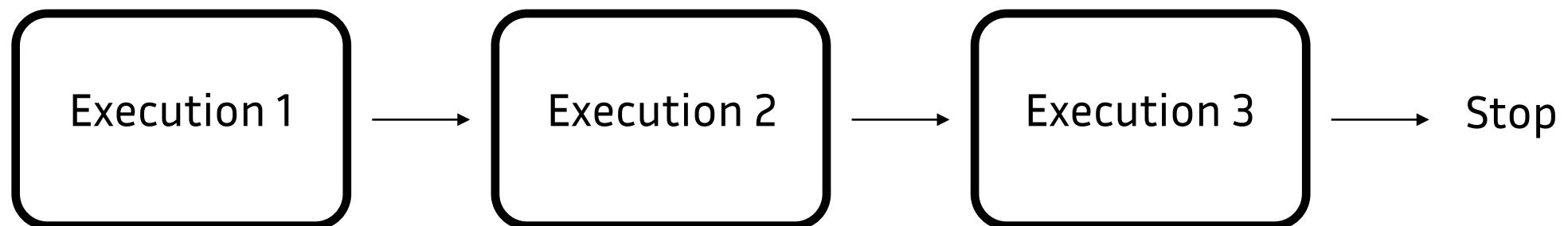
Solutions:

- ▶ MFC (introduit un langage au dessus de C++)
- ▶ Java (utilise des listeners)
- ▶ **Qt (utilise principalement des signaux et slots)**

Application algorithmique

Utilisation de procédures (fonctions) appelées de manière séquentielle

Série d'étapes à réaliser dans un certain ordre



Entrées - sorties utilisateur

Programmation “classique” :

- ▶ Programme principal initialise et appelle des fonctions dans un ordre pre-déterminé
- ▶ Les éventuels événements utilisateurs sont « demandés » (programme en pause)

Programmation “**évènementielle**” :

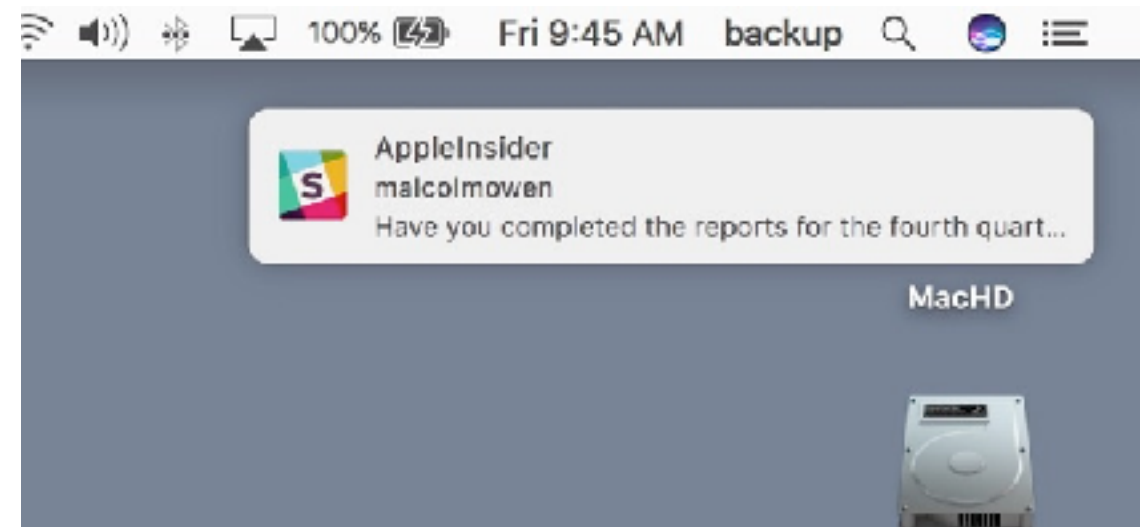
- ▶ Programme principal **initialise** des variables et les fonctions **réagissent aux événements**
- ▶ Le déroulement est contrôlé par la survenue d'événements (dont les actions de l'utilisateur)
- ▶ Boucle principale qui traite les événements (enfouie dans la bibliothèque)

Quels évènements?

Actions utilisateurs

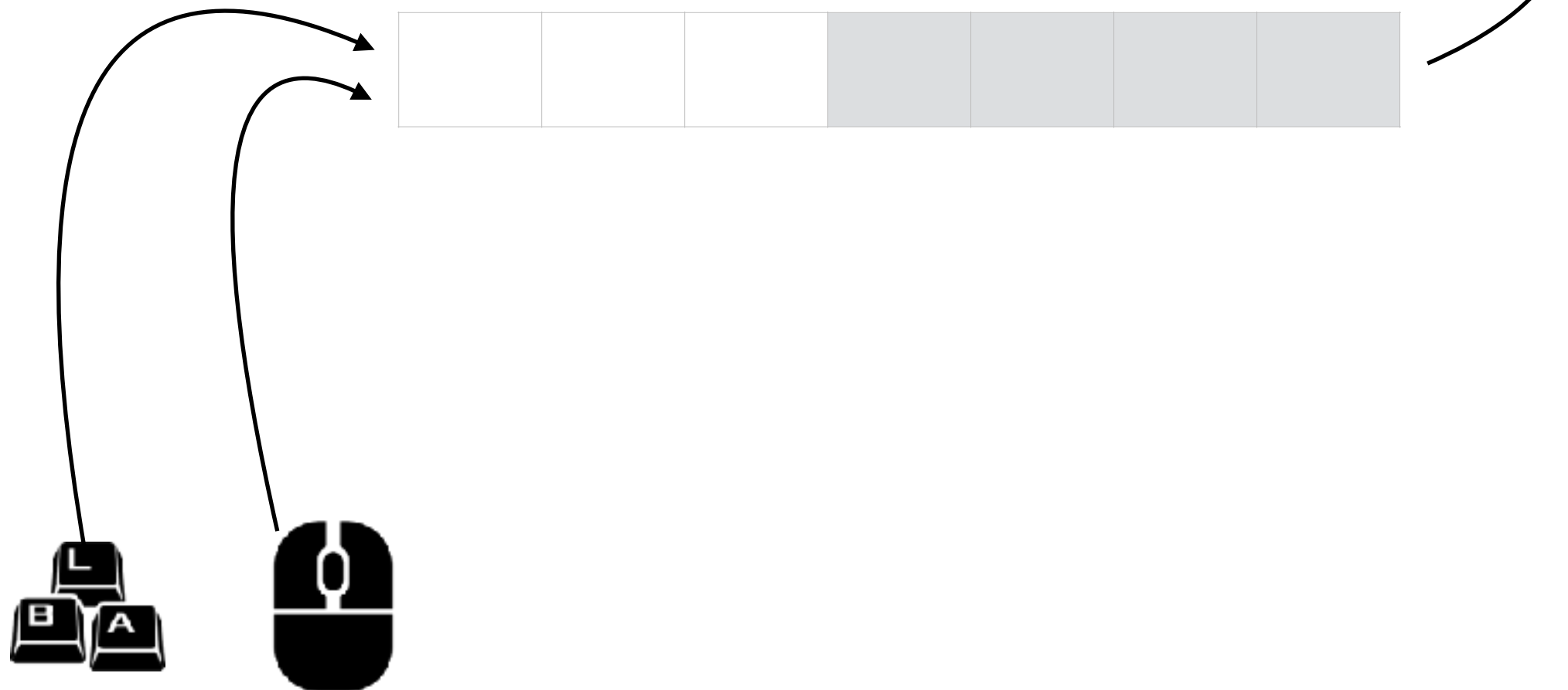
Notifications de processus (applications, OS, MAJ)

Capteurs sensorielles (info ubiquitaire)




```
while (true){  
    if(!queue.isEmpty()){  
        event = queue.nextEvent();  
        source = findSourceForEvent(event);  
        source.processEvent(event);  
    }  
}
```

File d'attente (queue FIFO)



Connecter signaux et slots



Connecter signaux et slots



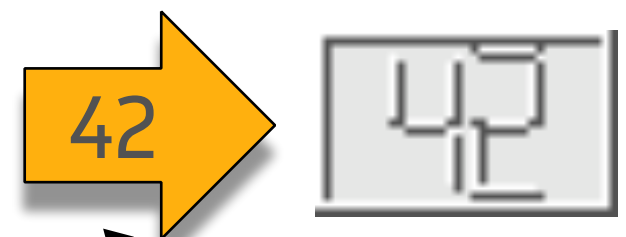
Signal émis



Connecter signaux et slots

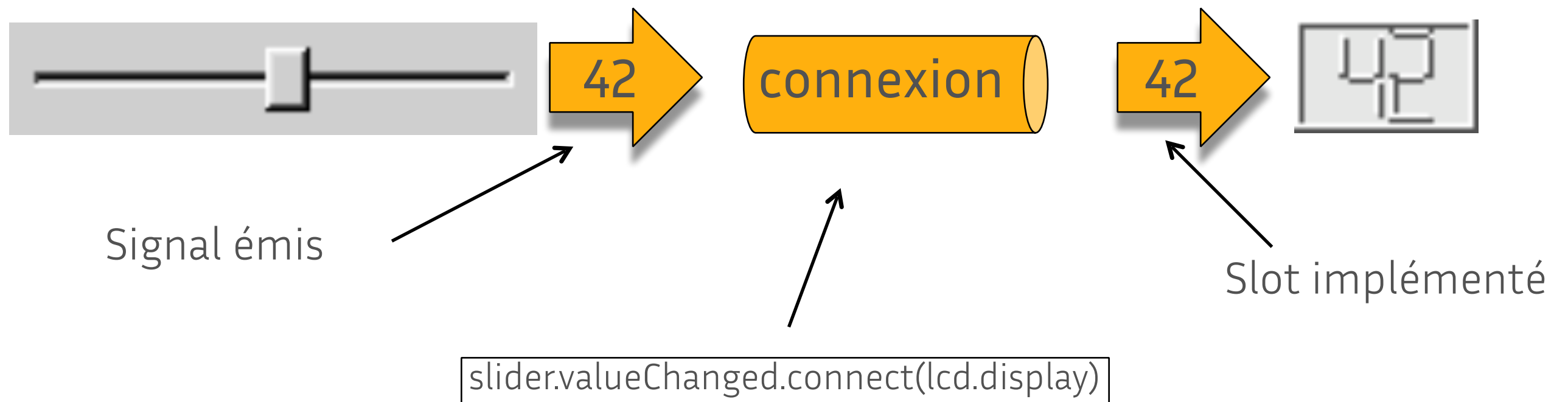


Signal émis



Slot implémenté

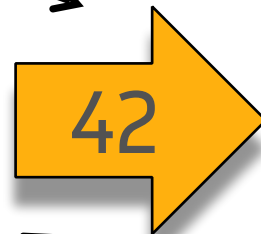
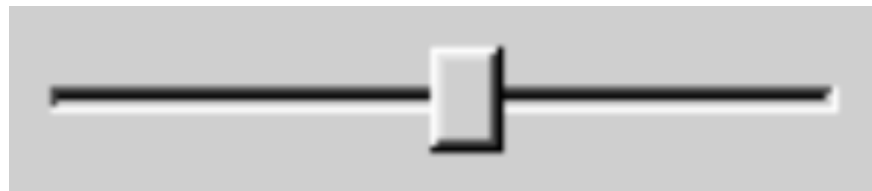
Connecter signaux et slots



Connecter signaux et slots

```
class QSlider(QObject):  
    ...  
    def mousePressEvent(self)  
        self.valueChanged.emit(value)  
    ...
```

slider = QSlider(None)



Signal émis

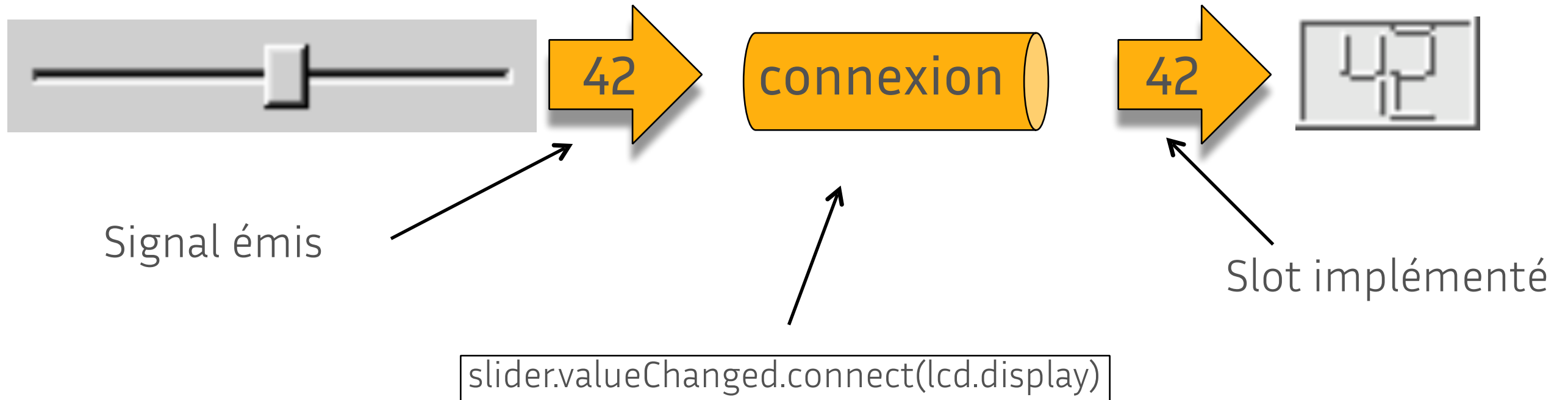
Slot implémenté

```
slider.valueChanged.connect(lcd.display)
```

Connecter signaux et slots

```
class QLCDNumber(QObject):  
  
    def display(num)  
        m_value = num;  
        ...
```

lcd = QLCDNumber(None)



Une classe avec des signaux et des slots

```
class MyClass(QObject):  
  
    mySignal = pyqtSignal(int)  
  
    void mySlot( self, num ):  
        blabla
```

- Sous class de **QObject**
- Les **signaux** ne sont pas implémentés
- Les **slots** doivent être implémentés

Une classe avec des signaux et des slots

```
class MyClass(QObject):  
  
    mySignal = pyqtSignal(int)  
  
    def __init__(self, parent =None):  
        super(MyClass, self).__init__(parent)  
  
    @pyqtSlot(int)  
    void mySlot( num ): ← parfois nécessaire  
        blabla
```

- Sous class de **QObject**
- Les **signaux** ne sont pas implémentés
- Les **slots** doivent être implémentés

Signaux et slots

Modularité, flexibilité

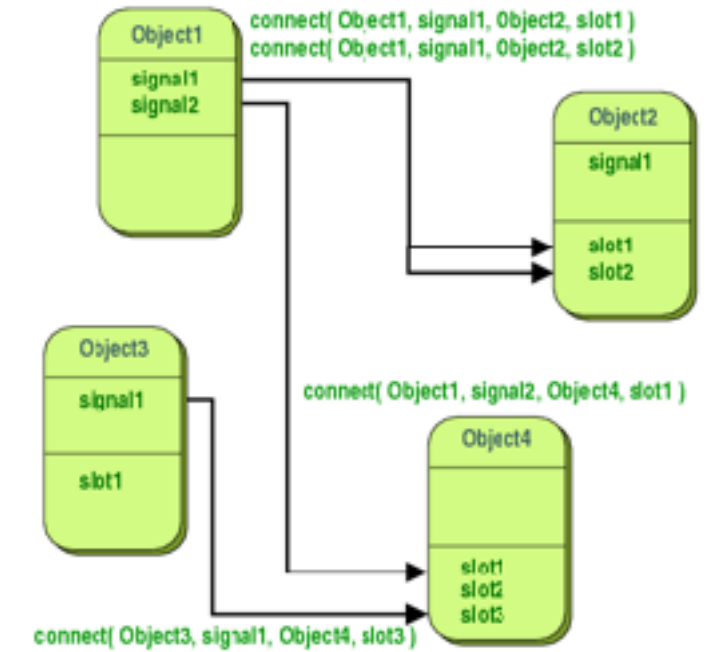
- ▶ Connecter **plusieurs** signaux à **un** slot
- ▶ Connecter **un** signal à **plusieurs** slots

Philosophie

- ▶ L'émetteur n'a pas besoin de connaître le(s) récepteur(s)
- ▶ L'émetteur ne sait pas si le signal a été reçu
- ▶ Le récepteur ne connaît pas non plus l'émetteur
- ▶ Programmation par composant (indépendant, réutilisable)

Sécurité, typage fort

- ▶ Les types des paramètres doivent être les mêmes
- ▶ Un slot peut avoir **moins** de paramètres qu'un signal



Exemple : transfert d'argent entre banques

```
class PunchingBag(QObject):
    punched = pyqtSignal() ← Signal

    def __init__(self):
        # Initialize the PunchingBag as a QObject
        QObject.__init__(self)

    def punch(self): ← Slot
        self.punched.emit()

@pyqtSlot()
def say_punched():
    print('Bag was punched.')

def main(args):
    bag = PunchingBag()
    # Connect the bag's punched signal to the say_punched slot
    bag.punched.connect(say_punched) ← Connexion

    # Punch the bag 10 times
    for i in range(10):
        bag.punch()

if __name__ == "__main__":
    main(sys.argv)
```

Questions

Comment connecter un signal à un slot ?

- ▶ EmetteurObj.<nameSignal>.**connect** (Recepteur.<nameSlot>)

Quel code pour déclarer / implémenter un slot ?

- ▶ rien de particulier (mais on peut rajouter @pyqtSlot())

Est ce qu'un slot peut retourner une valeur ?

- ▶ Oui

Quel code pour déclarer / implémenter un signal ?

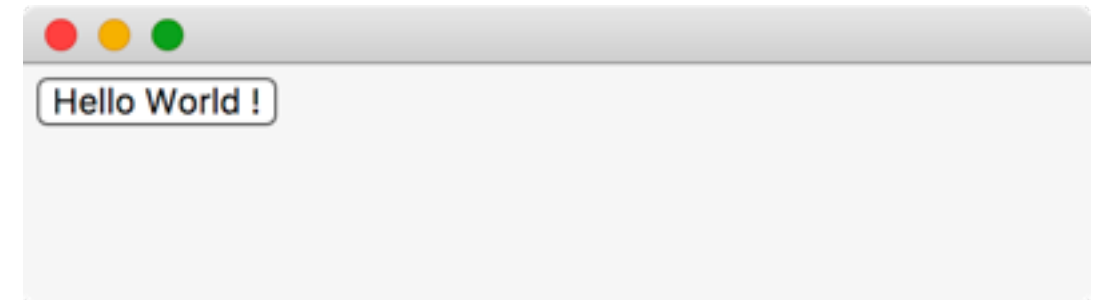
- ▶ mySignal = pyqtSignal()

Simple window with button in widget

```
from PyQt5.QtCore import *  
from PyQt5.QtWidgets import *  
import sys
```

```
def main(args) :  
    app = QApplication(args)  
    widget = QWidget(None)  
    widget.resize(400,90)  
    button = QPushButton("Hello World !", widget)  
    button.resize(100,30)  
    → button.clicked.connect(app.quit)  
    widget().show()  
    app.exec_()
```

```
if __name__ == "__main__":  
    main(sys.argv)
```



Les principaux modules

QtCore

QtWidgets

QtGUI

QtBluetooth

QtOpenGL

QtScript/QtScriptTools

QtSql

QtSvg

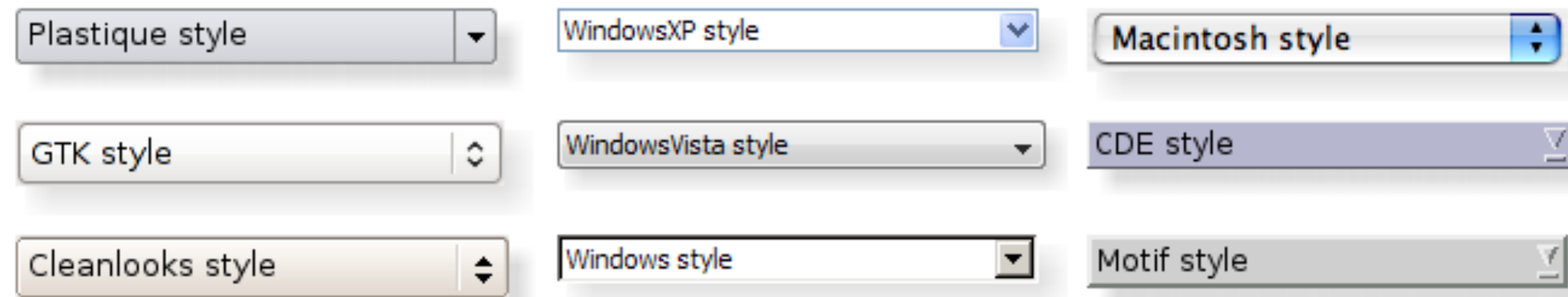
QtWebKit

QtXml/QtXmlPatterns

QtMultimedia

QtSensors

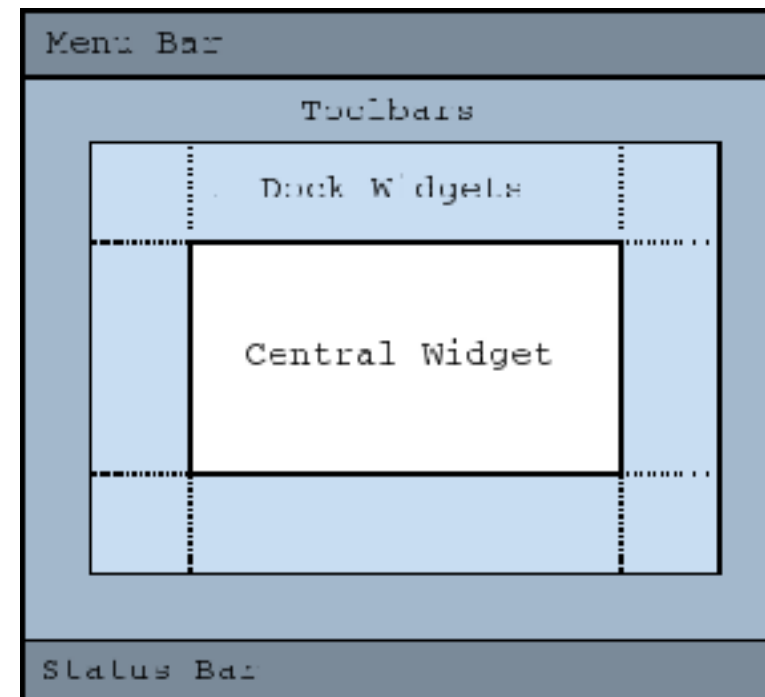
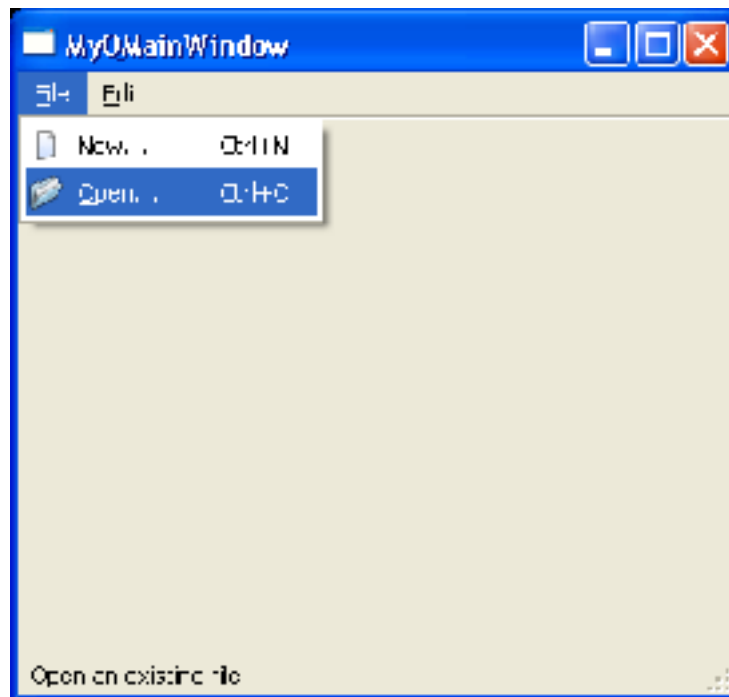
QStyle



Peut être passé en argument à l'exécution du programme

Ex: `python3 test.py -style Windows`

QMainWindow



Méthode 1: créer une instance de QMainWindow

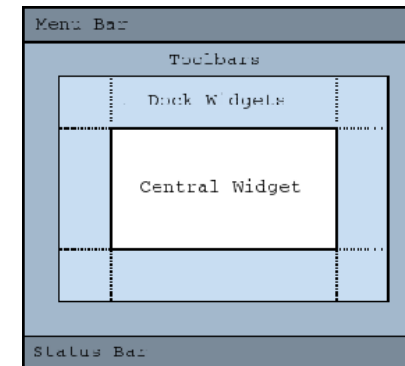
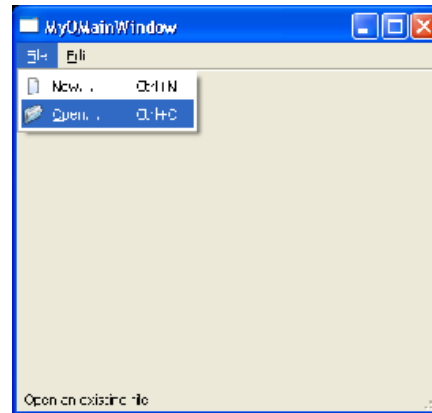
```
win = QMainWindow()  
win.resize(200, 300)
```

Méthode 2: créer une sous-classe de QMainWindow

```
class Win(QMainWindow):  
    def __init__(self):  
        self.resize(200, 300)
```


QMainWindow

Menus



```
bar = self.menuBar()
```



si sous-classe (methode 2)
sinon win.menuBar() (methode 1)

```
fileMenu = bar.addMenu( "File" )
```

```
newAct = QAction(QIcon( "path/images/new.png" ), "New...", None )
```

```
newAct.setShortcut( "Ctrl+N" )
```

```
newAct.setToolTip(tr("New File"))
```

```
newAct.setStatusTip(tr("New file"))
```

```
fileMenu.addAction(newAct)
```

```
newAct.triggered.connect( self.open )
```

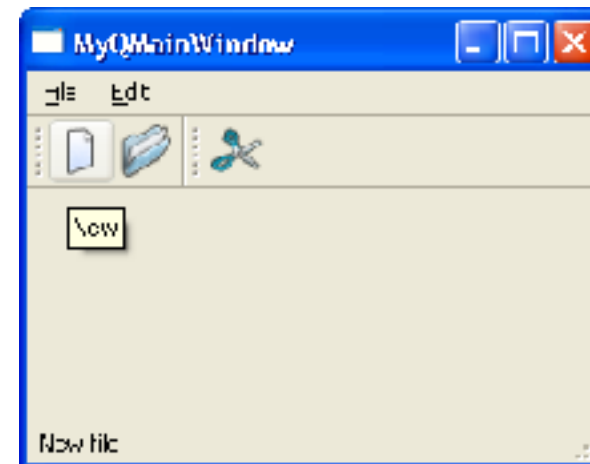
QMainWindow

QMenuBar, QMenu, QAction

QToolBar

- ▶ fileToolBar = QToolBar("File")
- ▶ fileToolBar.addAction(newAct)
- ▶ newAct.setEnabled(false)

← desactive (grise) la commande dans les menus et la toolbar



QToolTip, QWhatsThis

Composant central

```
textEdit = TextEdit( self );  
self.setCentralWidget( textEdit );
```

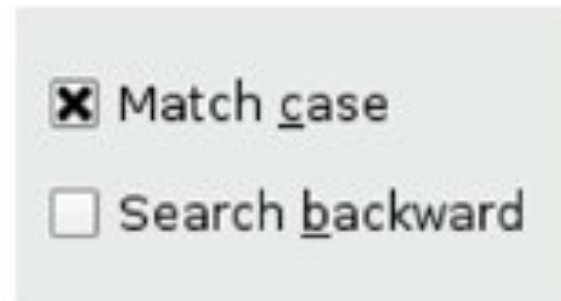
Buttons



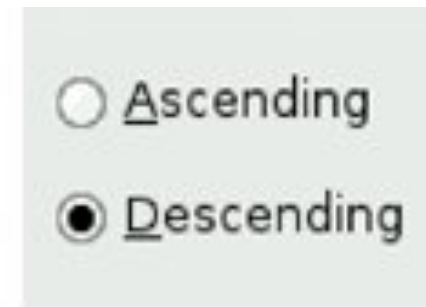
QPushButton



QToolButton



QCheckBox



QRadioButton

Input Widgets



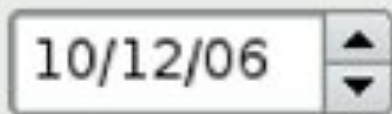
QSpinBox



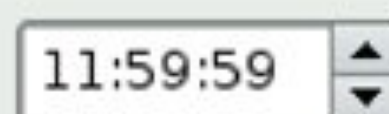
QDoubleSpinBox



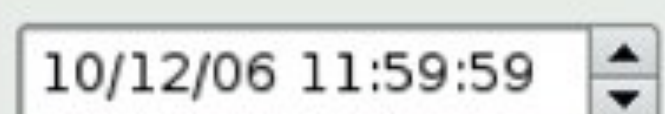
QComboBox



QDateEdit



QTimeEdit



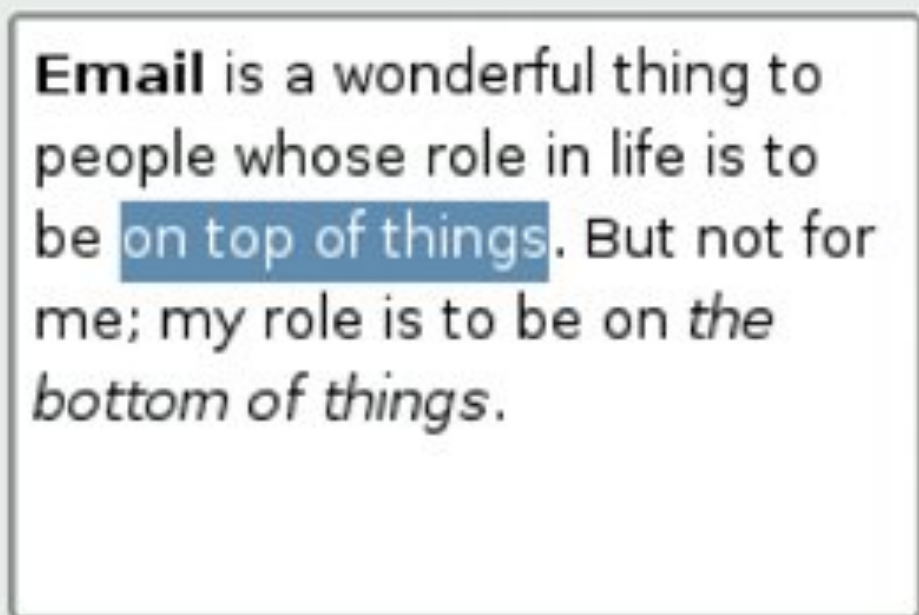
QDateTimeEdit



QScrollBar



QSlider



QTextEdit



QLineEdit

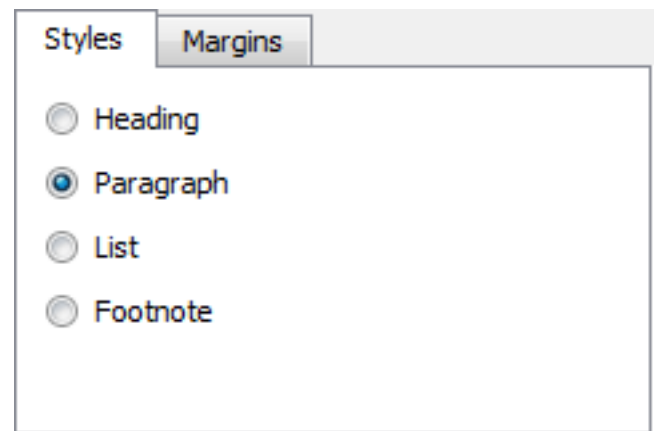


QDial

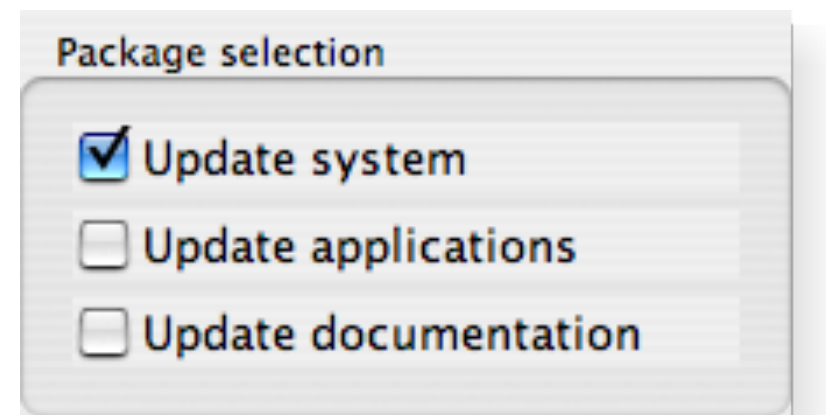
Containers



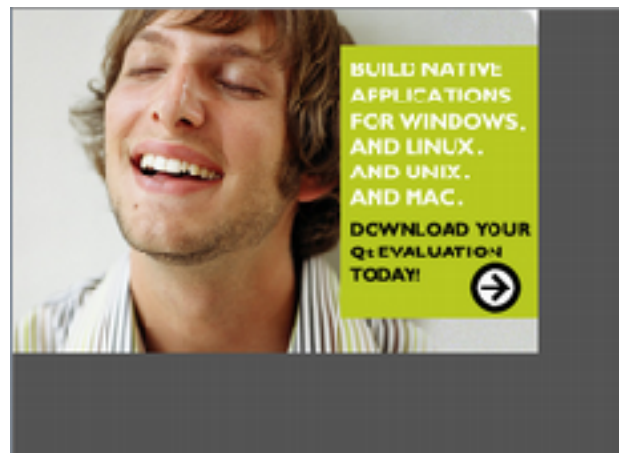
QMidArea



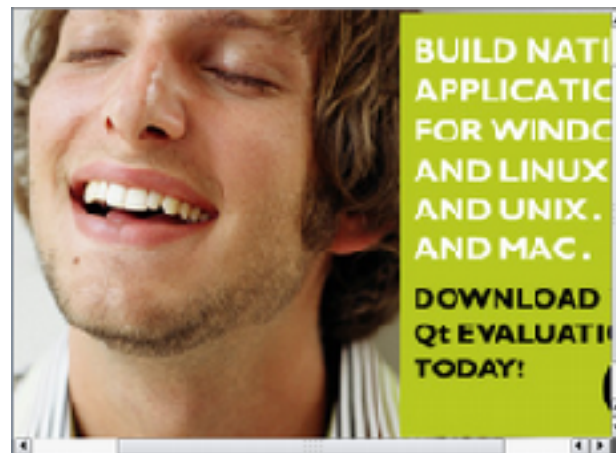
QTabWidget



QGroupBox



QScrollArea



QToolBox

QWidget; QFrame; QDockWidget; QStackedWidget

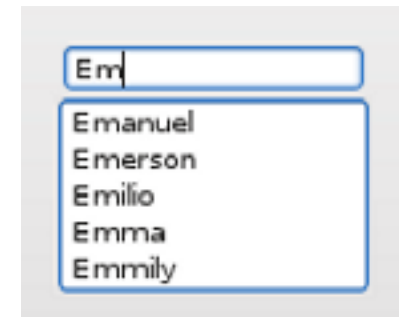
Views



QListView (as list)



QTreeView



QCompleter



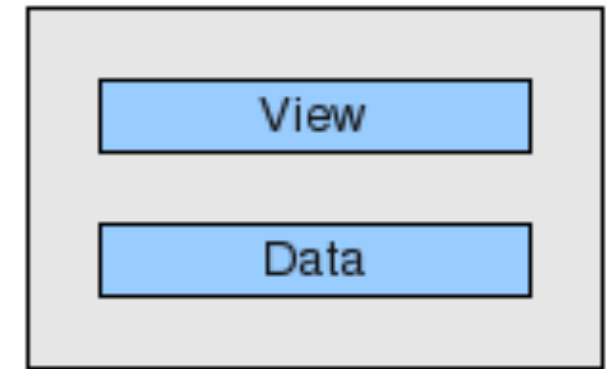
QListView (as icons)

A screenshot of a QTableView widget displaying a table with 4 rows and 3 columns. The columns are labeled A, B, and C. The data is as follows:

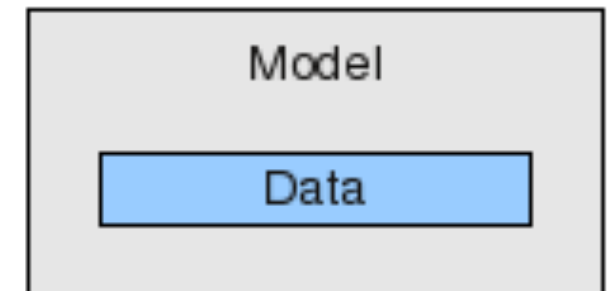
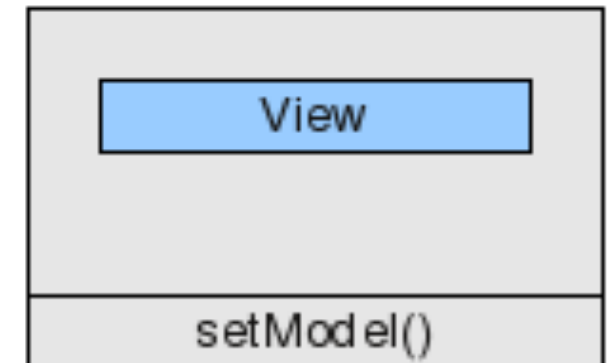
	A	B	C
1	1043.23	250	
2	1037.39	178	
3	970.77		
4	1008.32		

QTableView

standard widgets use data that is part of the widget



View classes operate on external data (the model)



```
def main(args):  
    app = QApplication(args)  
    tableView = QTableView()  
    myModel = MyModel()  
    tableView.setModel( myModel )  
    tableView.show()  
    app.exec()
```



```
class MyModel(QAbstractTableModel):
    def __init__(self):
        QAbstractTableModel.__init__(self)
        self.myData = <dataBase>

    def rowCount( self, parent ):                #Type (parent) ==
    QModelIndex
        return 2

    def columnCount( self, parent ):
        return 2

    def data( self, index, role=Qt.DisplayRole):
        if role == Qt.DisplayRole:
            return self.myData(index.row() + 1, index.column()+1)
```

```
def main(args):
    app = QApplication(args)
    tableView = QTableView()
    myModel = MyModel()
    tableView.setModel( myModel )
    tableView.show()
    app.exec()
```

Display Widgets

Warning: All unsaved
information will be lost!

QLabel (text)

255

QLCDNumber

36%

QProgressBar



QLabel (image)

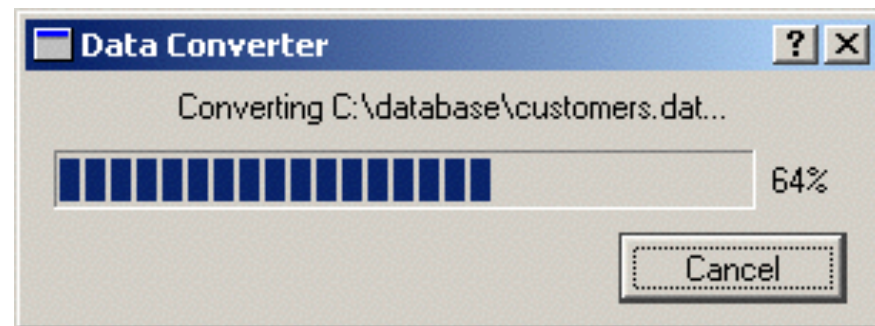
• QUrl & [operator=](#) (cons
• bool [operator==](#) (cons

Static Public Mem

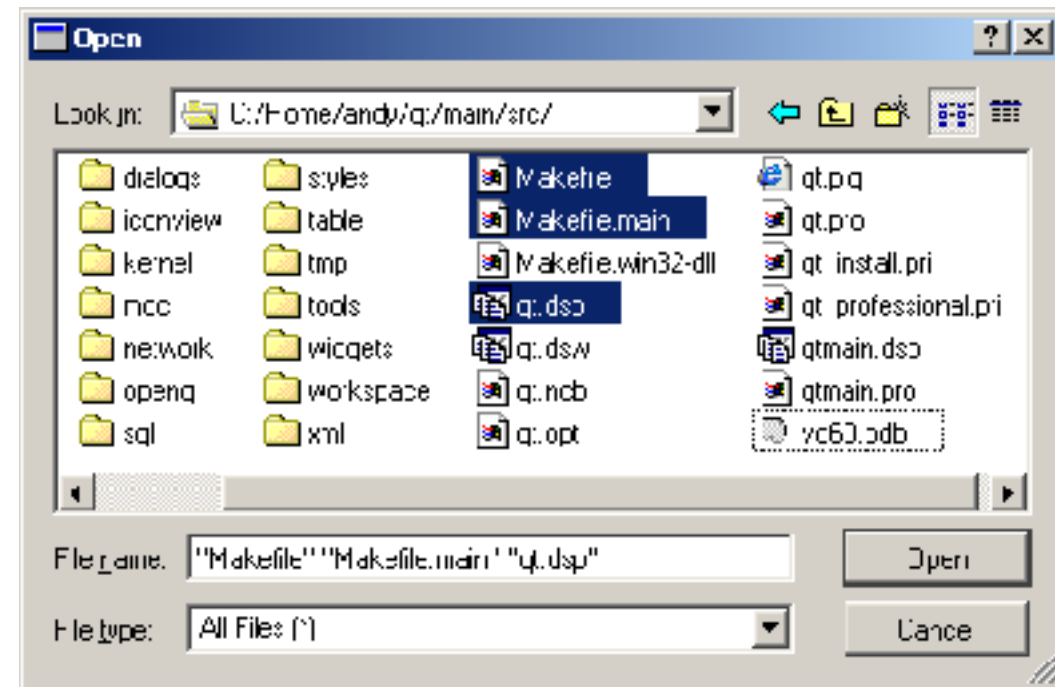
• QUrl [fromEncoded](#) (co
• QUrl [fromLocalFile](#) (co
• QString [fromPercentEn](#)

QTextBrowser

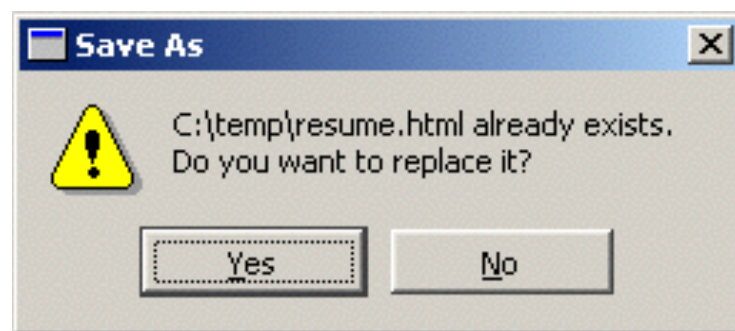
Boites de dialogue



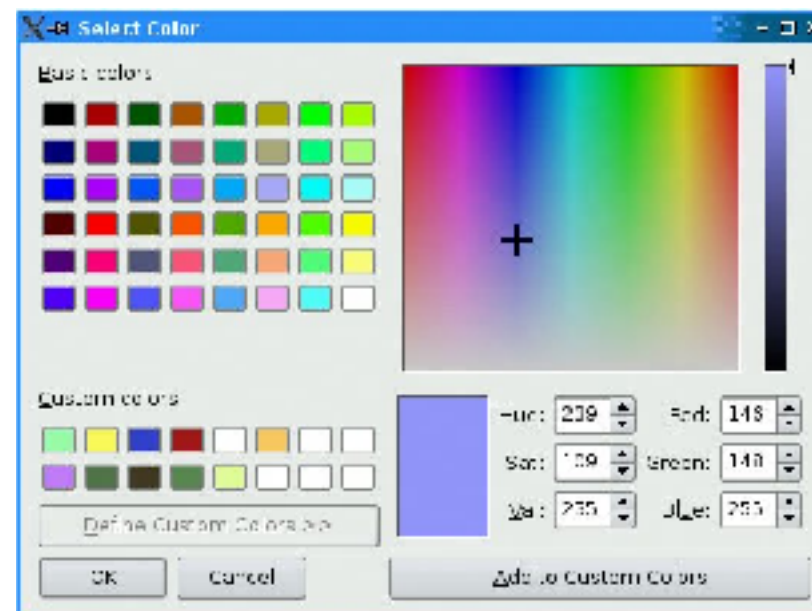
QProgressDialog



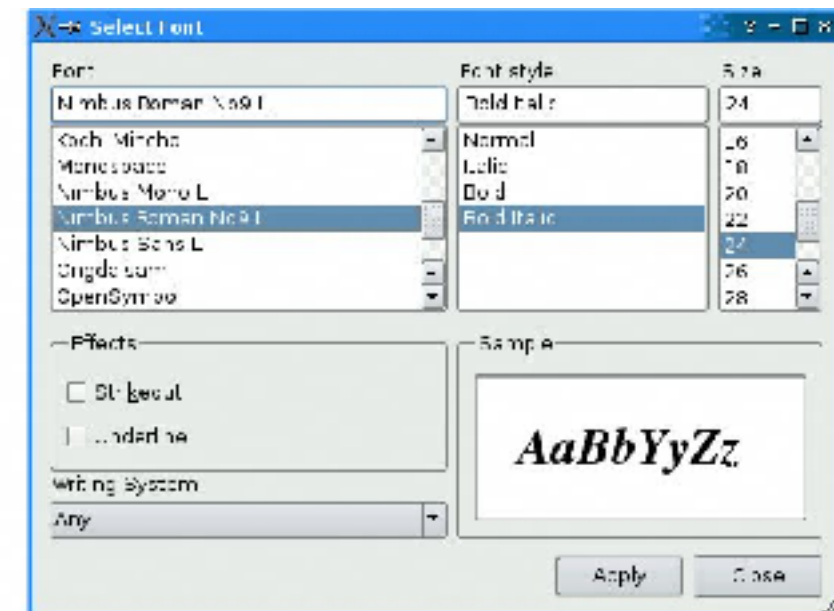
QFileDialog



QMessageBox



QColorDialog



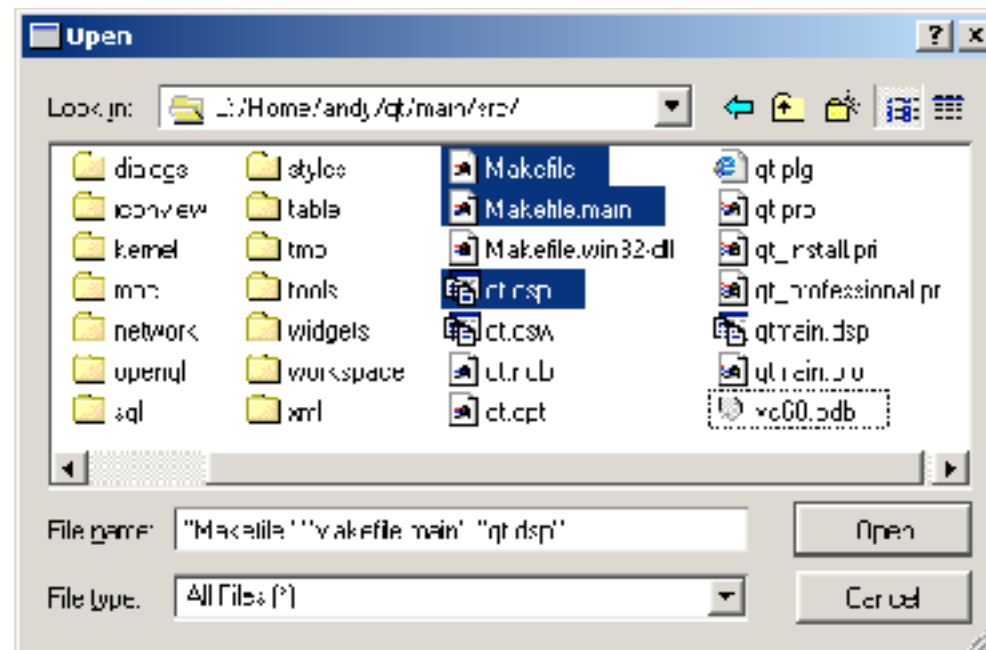
QFontDialog

Boîte de dialogue modale

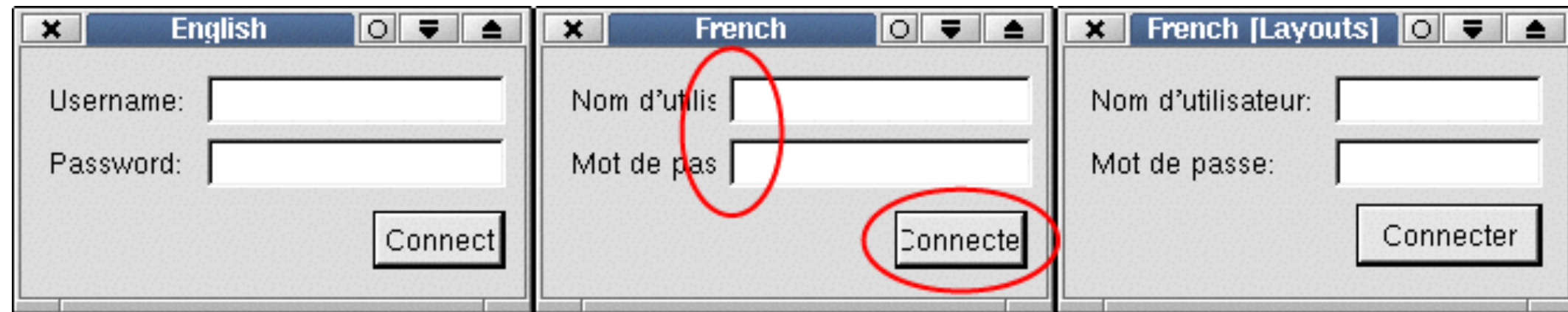
Solution simplifiée

```
fileName = QFileDialog.getOpenFileName( self,           //parent
                                         "Open Image",    // titre
                                         "/home/jana",     // répertoire initial
                                         "*.txt")          // filtre

fileName = QFileDialog.getSaveFileName(...)
```



Layout



Problèmes

- ▶ internationalisation
- ▶ redimensionnement
- ▶ complexité du code

Layout



A Qt-style dialog window titled "Windows" with a blue title bar containing standard window controls (minimize, maximize, close). The main area has a light gray background and contains three form fields arranged vertically, managed by a `QFormLayout`. Each field has a label on the left and an input widget on the right. The first field is labeled "Name:" and contains the text "Gandalf". The second field is labeled "Email address:" and contains the text "gg@troll.no". The third field is labeled "Age:" and contains the text "4000", with a small spin box control to its right.

`QFormLayout`

`QHBoxLayout`



A Qt widget with a light gray background. It contains five buttons arranged horizontally, managed by a `QHBoxLayout`. The buttons are labeled "One", "Two", "Three", "Four", and "Five" from left to right.



A Qt widget with a light gray background. It contains five buttons arranged vertically, managed by a `QVBoxLayout`. The buttons are labeled "One", "Two", "Three", "Four", and "Five" from top to bottom.

`QVBoxLayout`



A Qt widget with a light gray background. It contains five buttons arranged in a grid, managed by a `QGridLayout`. The buttons are labeled "One", "Two", "Three", "Four", and "Five". "One" and "Two" are in the first row, "Three" is in the second row (spanning both columns), and "Four" and "Five" are in the third row.

`QGridLayout`

Layout

Exemple

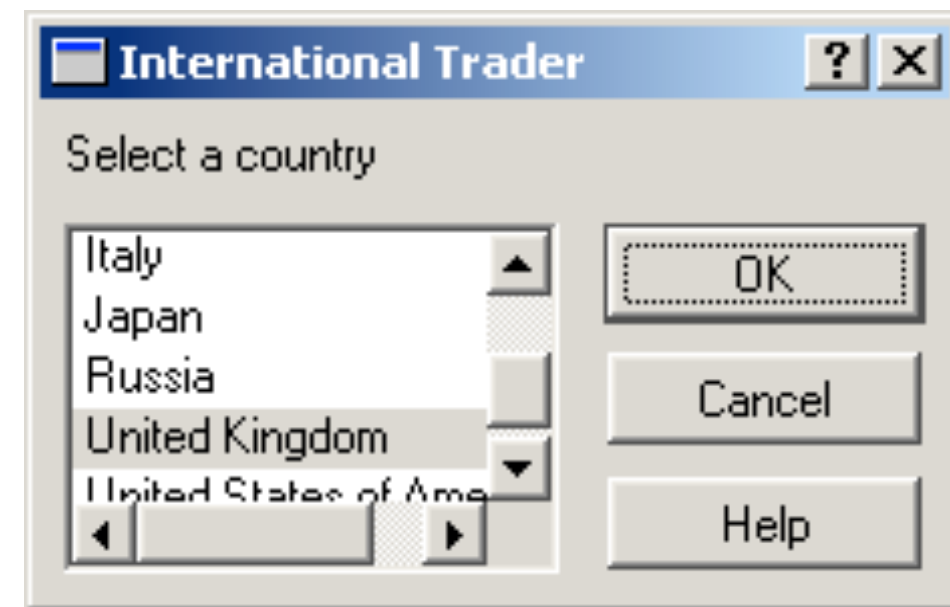
```
v_layout = QVBoxLayout( )
v_layout.addWidget( QPushButton( "OK" ) )
v_layout.addWidget( QPushButton( "Cancel" ) )
v_layout.addStretch( )
v_layout.addWidget( QPushButton( "Help" ) )

country_list = QListBox( );
countryList.insertItem( "Canada" );
...etc...

h_layout = QHBoxLayout( )
h_layout.addWidget( country_list )
h_layout.addLayout( v_layout )

top_layout = QVBoxLayout( )
top_layout.addWidget( QLabel( "Select a country" ) )
top_layout.addLayout( h_layout );

container = QWidget()
container.setLayout( top_layout )
win.setCentralWidget(container)
win.show( )
```



Notes sur layouts :

- peuvent être emboîtés
- pas liés à une hiérarchie de conteneurs comme Java
- cf. le « stretch »

Layout Exemple

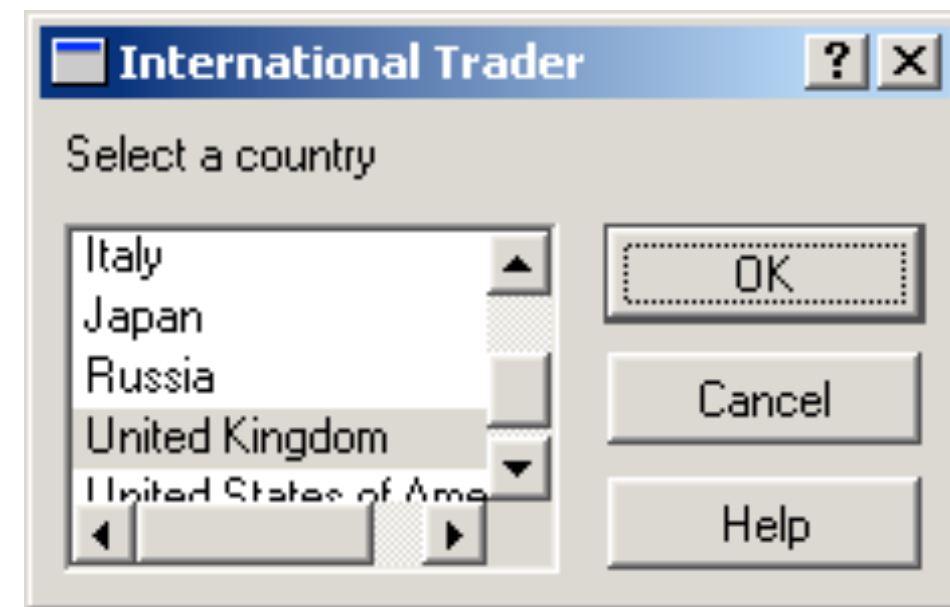
```
v_layout = QVBoxLayout( )
v_layout.addWidget( QPushButton( "OK" ) )
v_layout.addWidget( QPushButton( "Cancel" ) )
v_layout.addStretch( )
v_layout.addWidget( QPushButton( "Help" ) )
```

```
country_list = QListBox( );
countryList.insertItem( "Canada" );
...etc...
```

```
h_layout = QHBoxLayout( )
h_layout.addWidget( country_list )
h_layout.addLayout( v_layout )
```

```
top_layout = QVBoxLayout( )
top_layout.addWidget( QLabel( "Select a country" ) )
top_layout.addLayout( h_layout );
```

```
container = QWidget()
container.setLayout( top_layout )
win.setCentralWidget(container)
win.show( )
```



Notes sur layouts :

- peuvent être emboîtés
- pas liés à une hiérarchie de conteneurs comme Java
- cf. le « stretch »

Layout Exemple

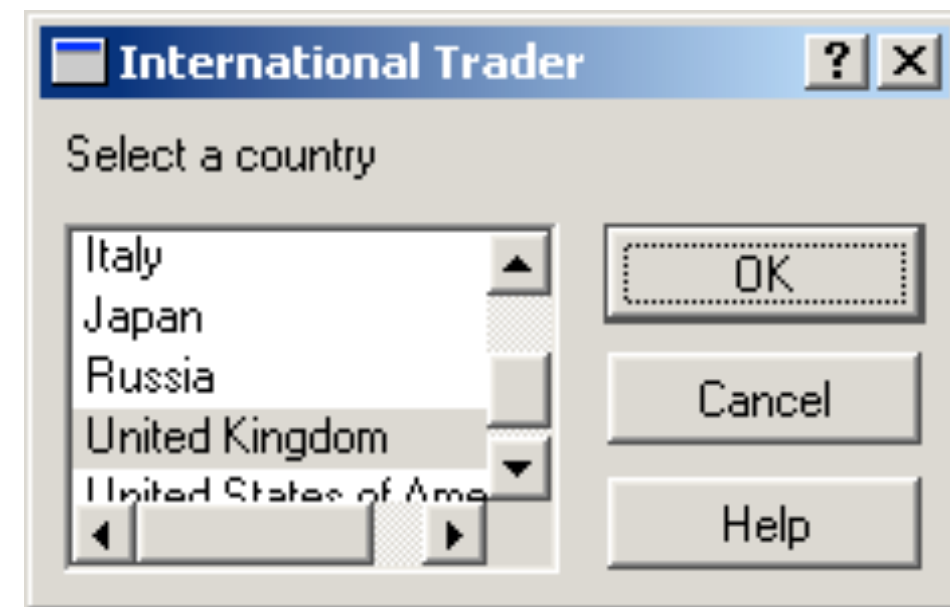
```
v_layout = QVBoxLayout( )  
v_layout.addWidget( QPushButton( "OK" ) )  
v_layout.addWidget( QPushButton( "Cancel" ) )  
v_layout.addStretch( )  
v_layout.addWidget( QPushButton( "Help" ) )
```

```
country_list = QListBox( );  
countryList.insertItem( "Canada" );  
...etc...
```

```
h_layout = QHBoxLayout( )  
h_layout.addWidget( country_list )  
h_layout.addLayout( v_layout )
```

```
top_layout = QVBoxLayout( )  
top_layout.addWidget( QLabel( "Select a country" ) )  
top_layout.addLayout( h_layout );
```

```
container = QWidget()  
container.setLayout( top_layout )  
win.setCentralWidget(container)  
win.show( )
```



Notes sur layouts :

- peuvent être emboîtés
- pas liés à une hiérarchie de conteneurs comme Java
- cf. le « stretch »

Layout Exemple

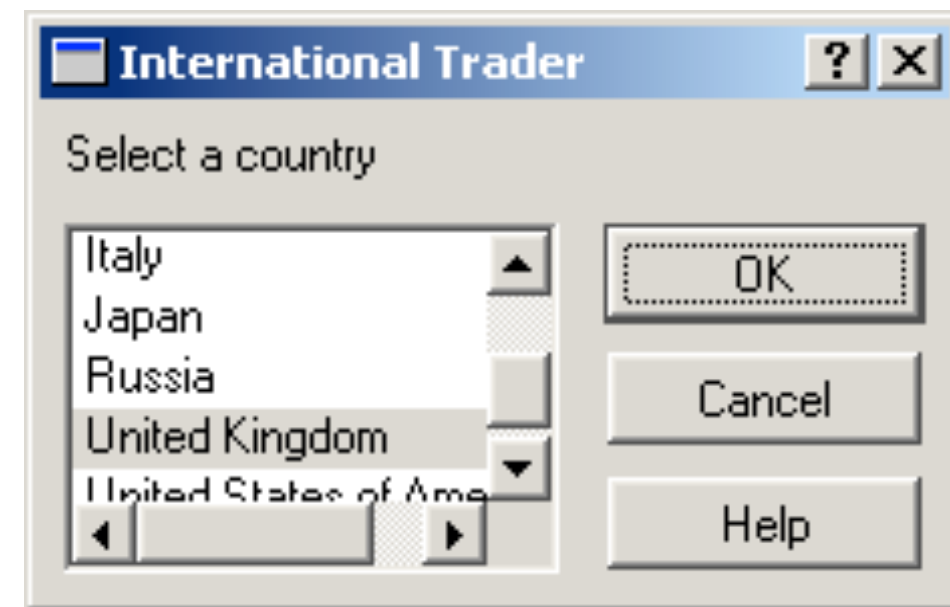
```
v_layout = QVBoxLayout( )  
v_layout.addWidget( QPushButton( "OK" ) )  
v_layout.addWidget( QPushButton( "Cancel" ) )  
v_layout.addStretch( )  
v_layout.addWidget( QPushButton( "Help" ) )
```

```
country_list = QListBox( );  
countryList.insertItem( "Canada" );  
...etc...
```

```
h_layout = QHBoxLayout( )  
h_layout.addWidget( country_list )  
h_layout.addLayout( v_layout )
```

```
top_layout = QVBoxLayout( )  
top_layout.addWidget( QLabel( "Select a country" ) )  
top_layout.addLayout( h_layout );
```

```
container = QWidget()  
container.setLayout( top_layout )  
win.setCentralWidget(container)  
win.show( )
```

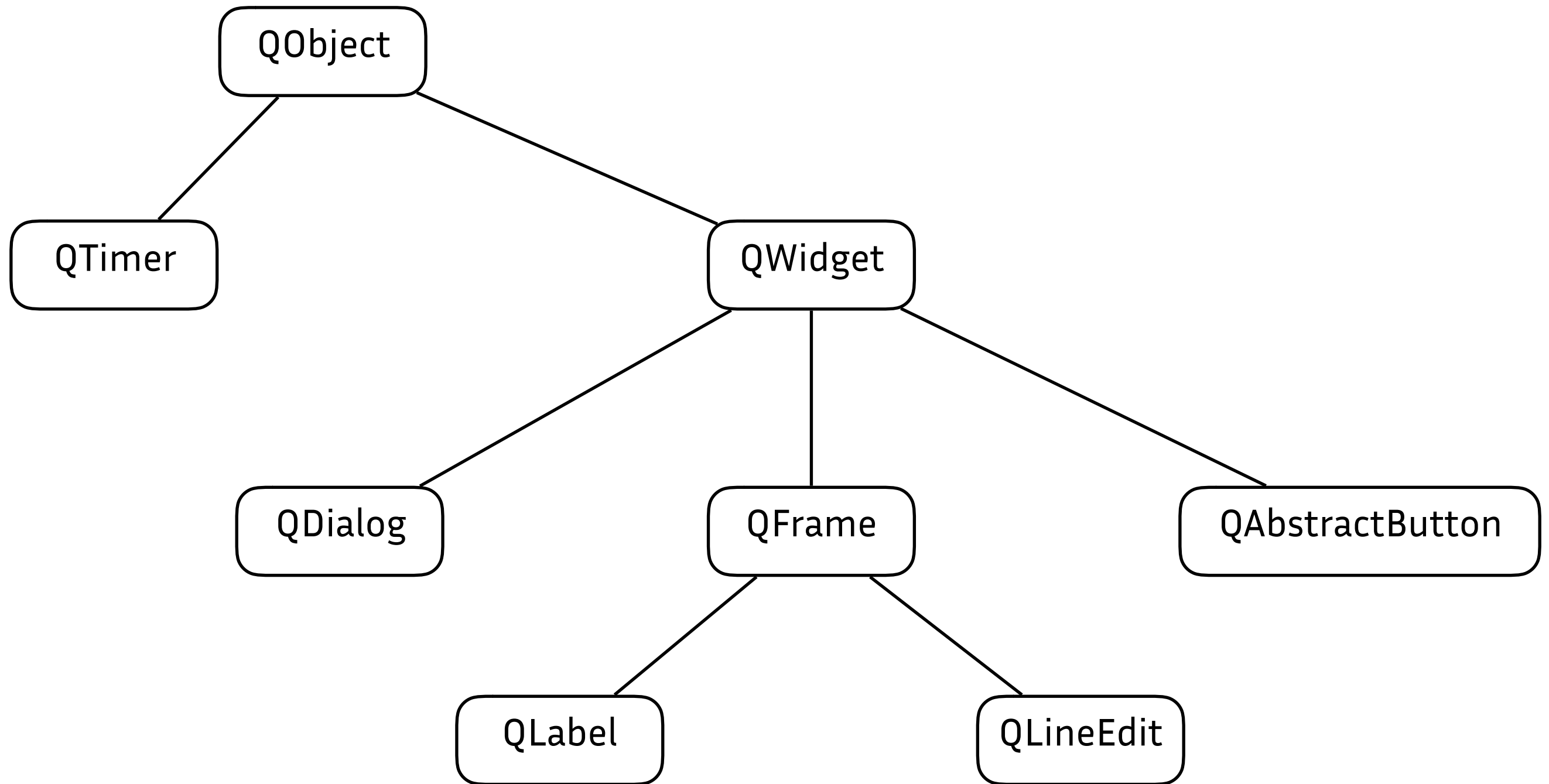


Notes sur layouts :

- peuvent être emboîtés
- pas liés à une hiérarchie de conteneurs comme Java
- cf. le « stretch »

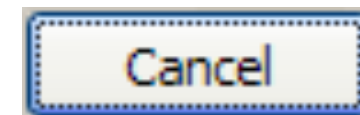
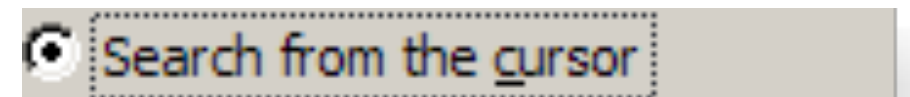
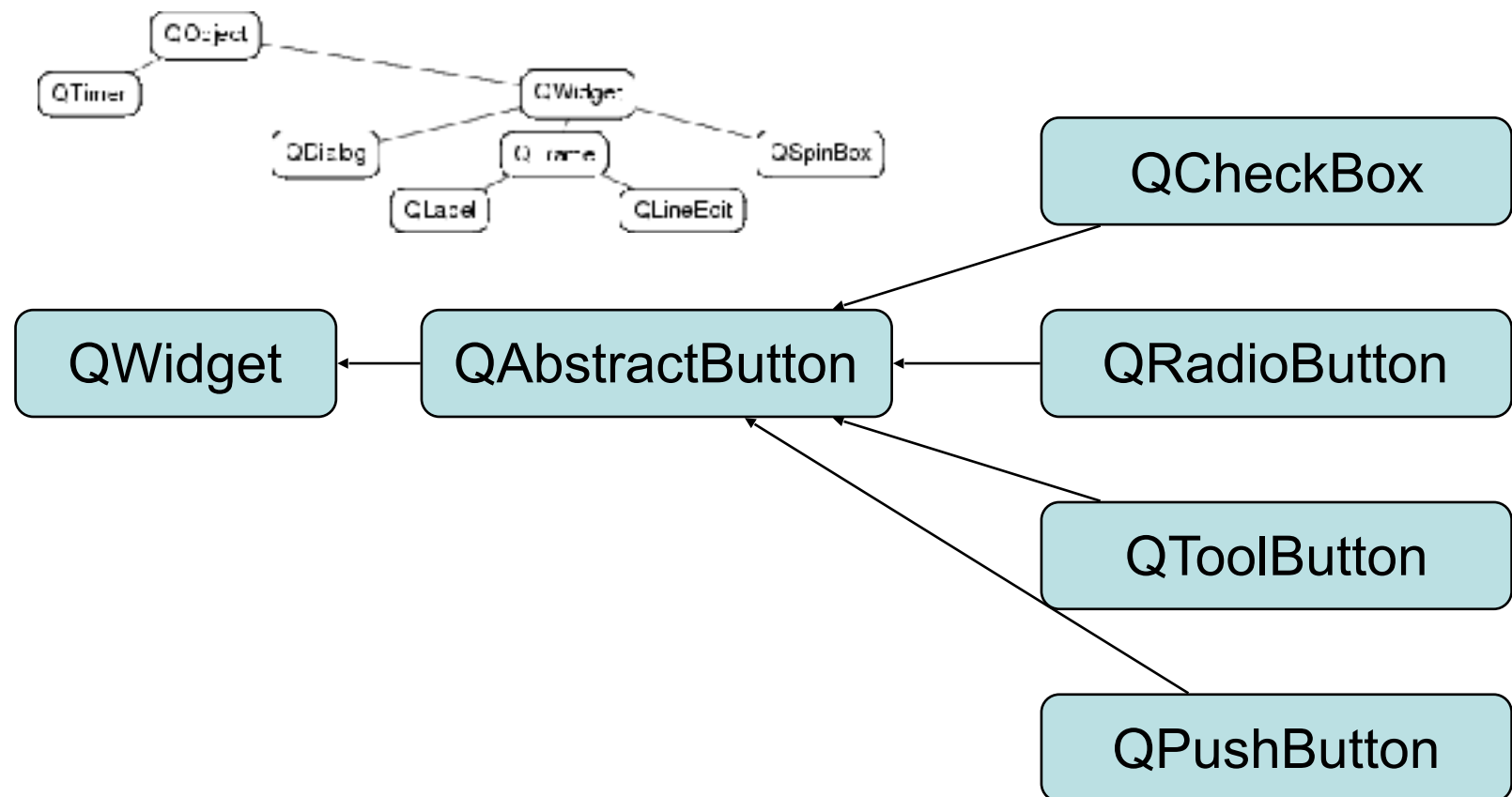
Arbre d'héritage
vs.
arbre d'instanciation

Arbres d'héritage



Principaux widgets

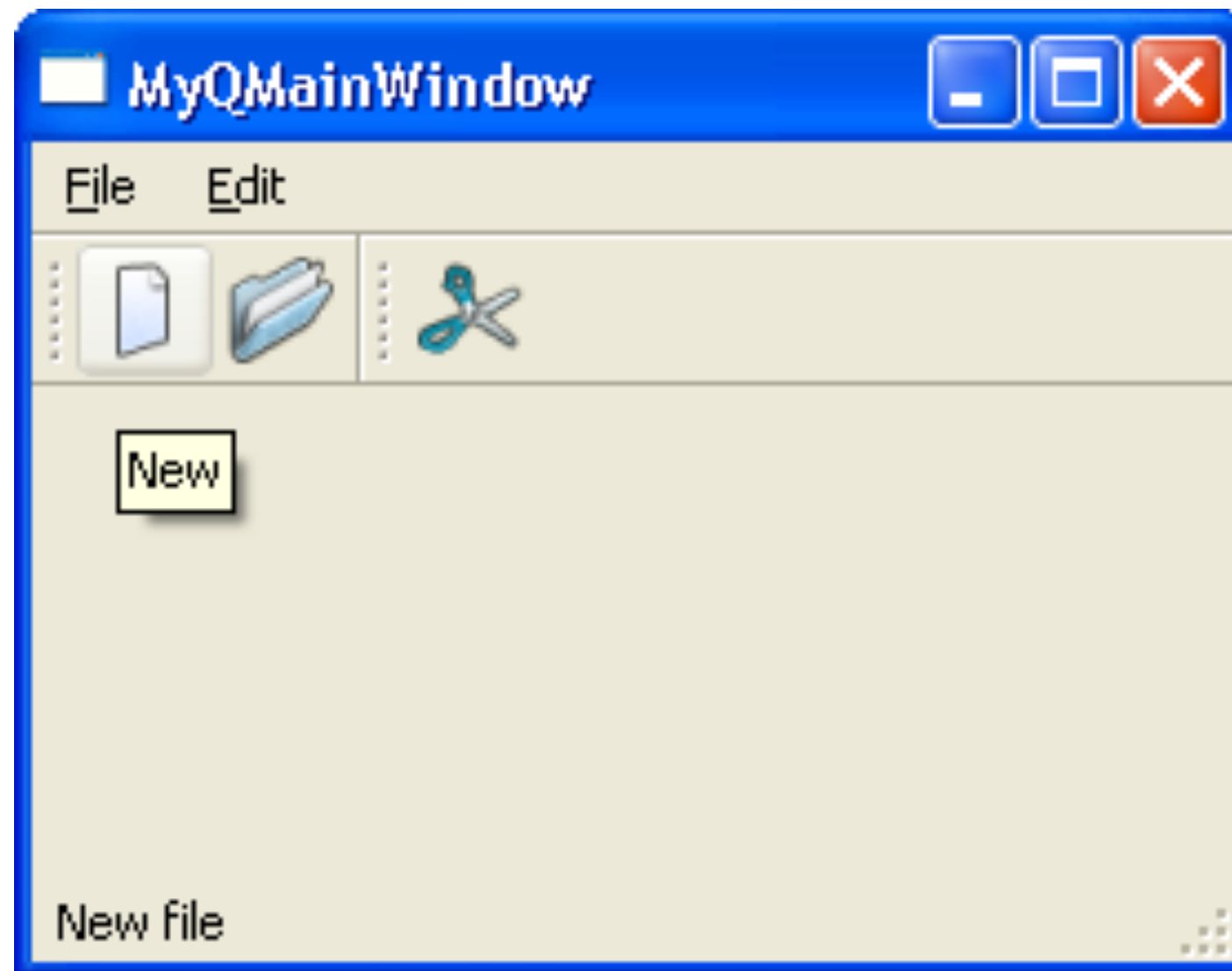
Arbre d'héritage



Arbre d'instanciation

Hiérarchie d'instance (=objets)

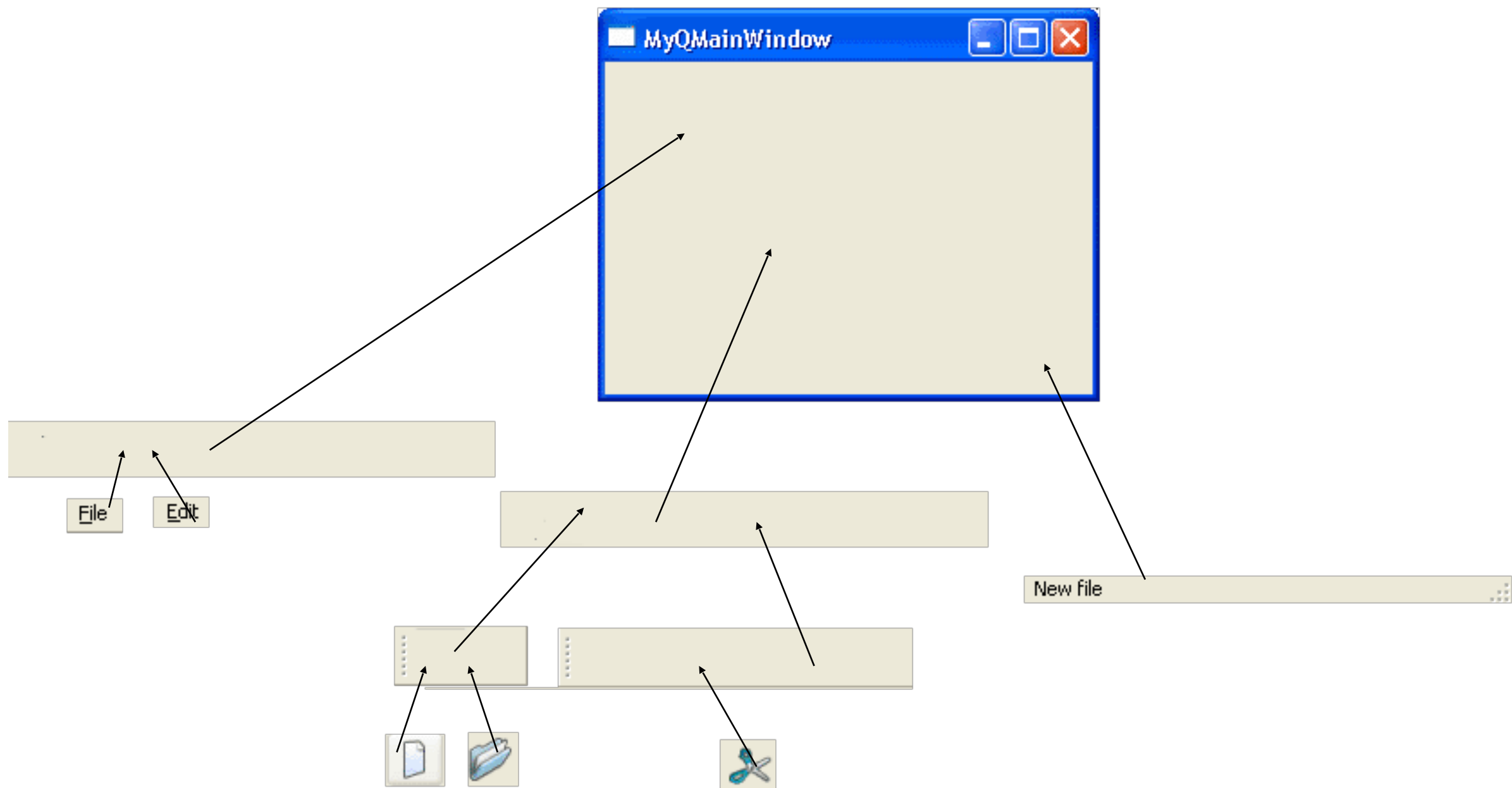
- Arbre de filiation des objets



Arbre d'instanciation

Hiérarchie d'instance (=objets)

- Arbre de filiation des objets



Arbre d'insanciation

Les enfants se déclarent auprès de son parent (\neq java)

- ▶ `label = QLabel("Hello", parent);`
- ▶ Exceptions
 - `QFile`, `QApplication`...

Si le parent d'un Widget est nul, le Widget est une fenêtre (Window).

Que font les parents ?

- ▶ Ils ont une liste des enfants
- ▶ Ils détruisent automatiquement les enfants quand ils sont détruits
- ▶ Enable/disable les enfants quand ils enable/disable eux memes
- ▶ Pareil pour Show/Hide

Arbre d'instanciation

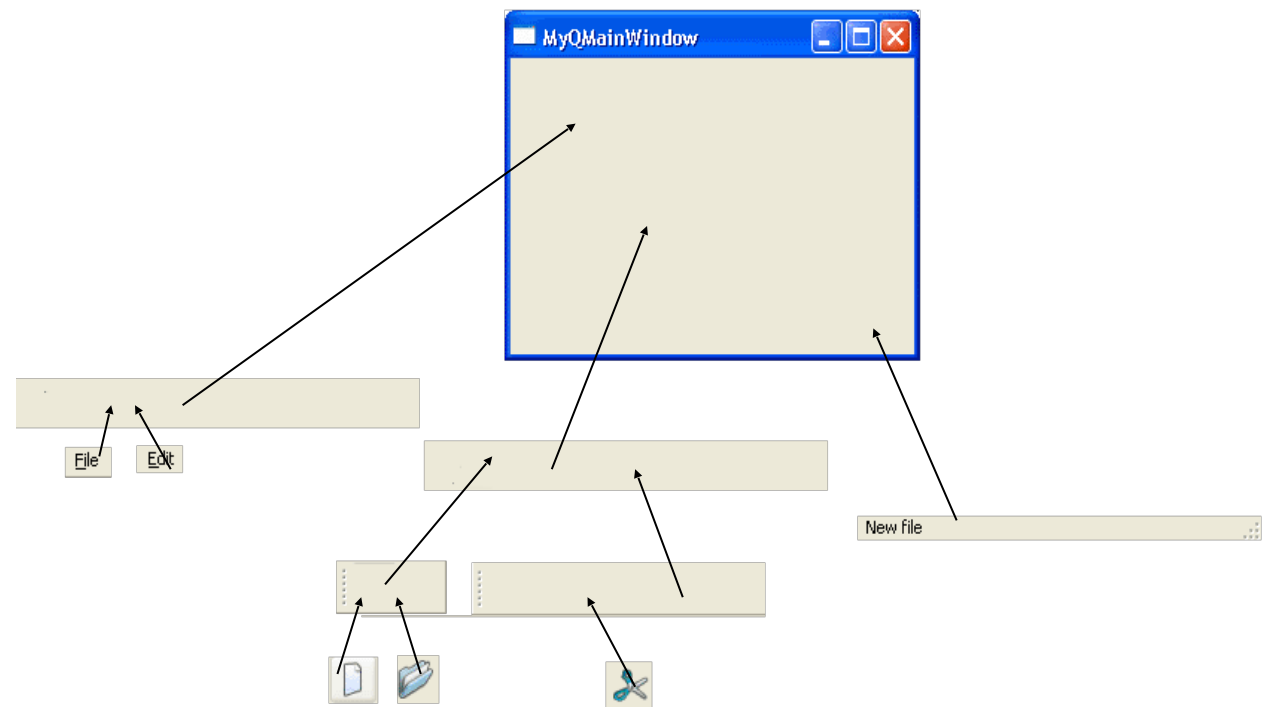
Hiérarchie d'instance (=objets)

- ▶ Arbre de filiation des objets

Chaque objet contient ses enfants

- ▶ Clipping : enfants inclus dans parents
- ▶ Superposition : enfants au dessus des parents

Un objet n'a qu'un seul parent



Arbre d'instanciation

- Les enfants se déclarent auprès de son parent (≠ java)
 - **label = QLabel("Hello", parent);**
 - Exceptions
 - QFile, QApplication...
- Si le parent d'un Widget est nul, le Widget est une fenêtre (Window).
- Que font les parents ?
 - Ils ont une liste des enfants
 - Ils détruisent automatiquement les enfants quand ils sont détruits
 - Enable/disable les enfants quand ils enable/disable eux memes
 - Pareil pour Show/Hide

Arbre d'instanciation

- Hiérarchie d'instance (=objets)
 - Arbre de filiation des objets
- Chaque objet contient ses enfants
 - Clipping : enfants inclus dans parents
 - Superposition : enfants au dessus des parents
- Un objet n'a qu'un seul parent

