

Programowanie w środowiskach RAD

QtCreator, Qt i C++

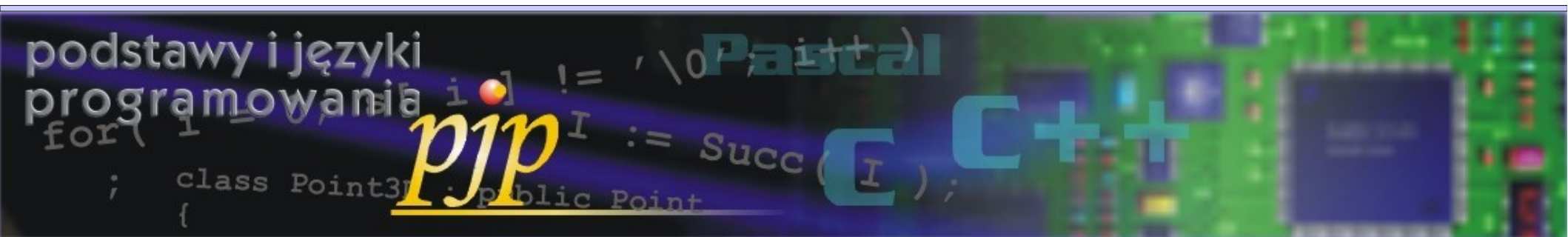
Roman Simiński

roman.siminski@us.edu.pl

www.siminskionline.pl

Wprowadzenie do programowania w C++ z wykorzystaniem biblioteki Qt

Sygnały i sloty

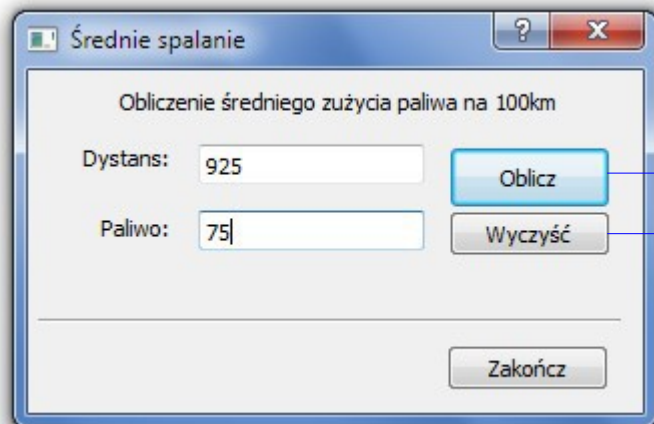


Sygnały i sloty – motywacja

- Programowanie sterowane zdarzeniami wymaga określenia powiązania pomiędzy elementem *generującym zdarzenie* a elementem to zdarzenie *obsługującym*.
- W obrębie pojedynczej aplikacji GUI polega to zwykle na powiązaniu informacji o zdarzeniu związanym z *elementem interfejsu*, a *podprogramem* realizującym obsługę tego zdarzenia.
- W różnych środowiskach i bibliotekach GUI stosuje się różne metody realizacji takich powiązań. Najpopularniejsze jest przypisywanie podprogramów do dedykowanych handler'ów obsługi zdarzeń, co mniej lub bardziej bezpośrednio stanowi realizację mechanizmu *wywołań zwrotnych* (ang. *call back*).
- W Qt zastosowano mechanizm dynamicznego wiązania *sygnałów* (ang. *signals*) generowanych przez nadawcę z podprogramami obsługi, zwanymi *slotami* (ang. *slots*).
- Sloty i sygnały stanowią niestandardowe *rozszerzenie* języka C++ w Qt, są obsługiwane przez MOC.

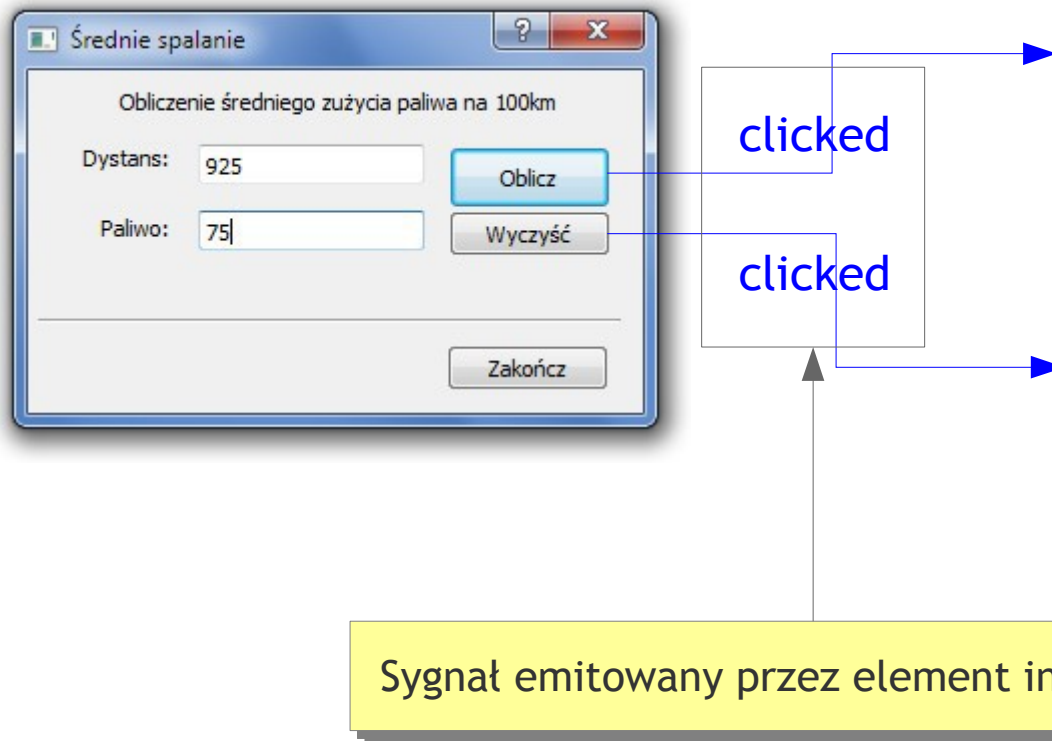
Sygnały i sloty – koncepcja

- Sloty i sygnały stanowią podstawę działania GUI, jednak ich zastosowanie nie ogranicza się tylko do zadań związanych z interfejsem.
- Sygnały mogą być emitowane przez elementy interfejsu (obiekty Qt), każdy element może emitować wiele sygnałów o różnych nazwach.



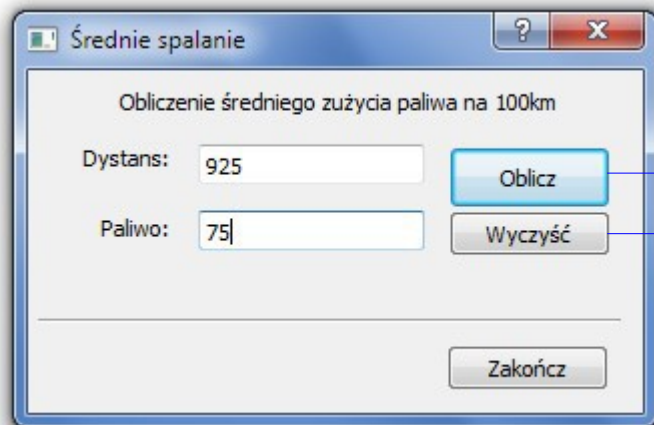
Sygnały i sloty – koncepcja

- Każdy sygnał ma swoją nazwę, nazwy sygnałów mogą się powtarzać w różnych elementach.
- Przykładowy sygnał dla klasy *QPushButton*: *clicked*



Sygnały i sloty – koncepcja

- Sygnały wyemitowane przez elementy interfejsu powinny dotrzeć do odpowiednich podprogramów obsługi.
- Aby tak się stało, należy *powiązać sygnały z podprogramami obsługi*.



clicked

clicked

Sygnał emitowany przez element interfejsu

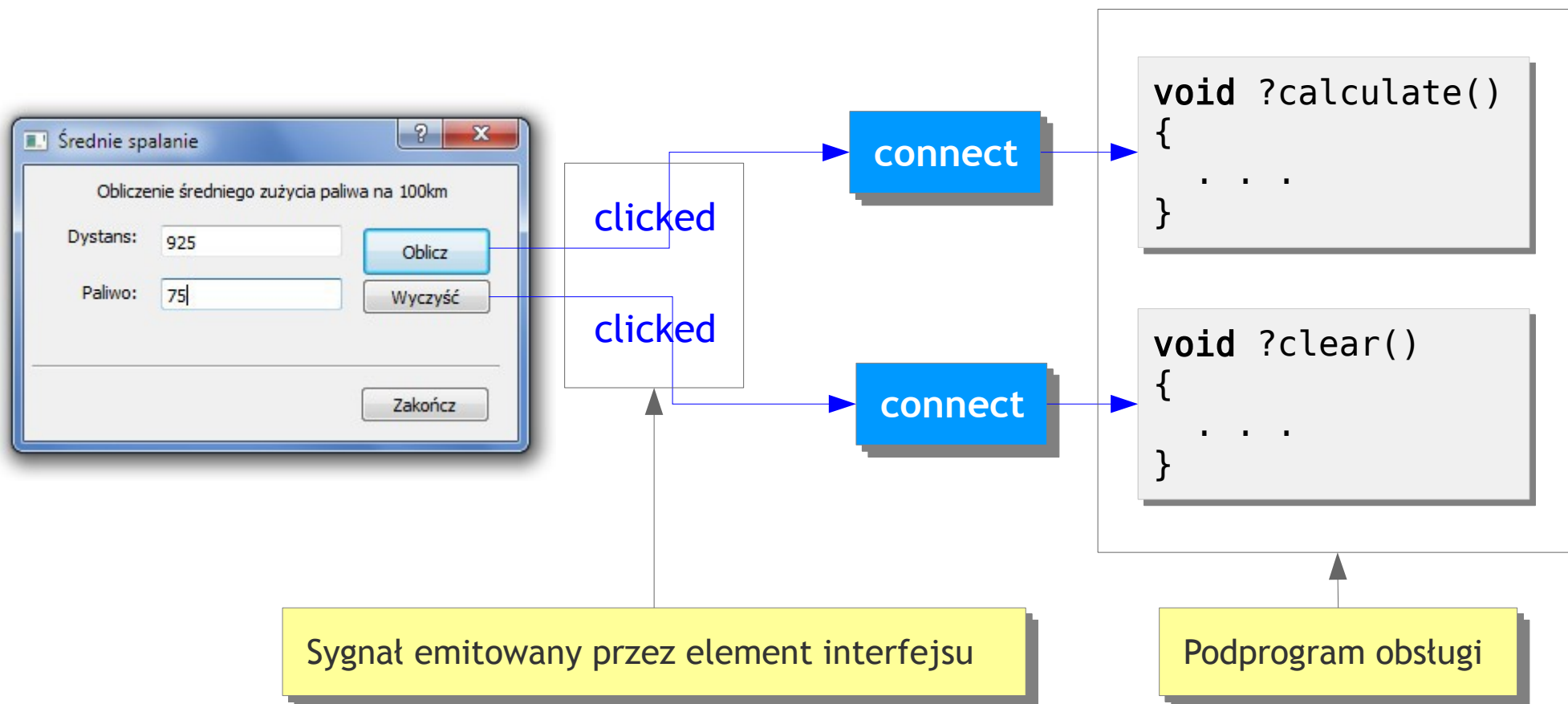
```
Void ?calculate()  
{  
    . . .  
}
```

```
void ?clear()  
{  
    . . .  
}
```

Podprogram obsługi

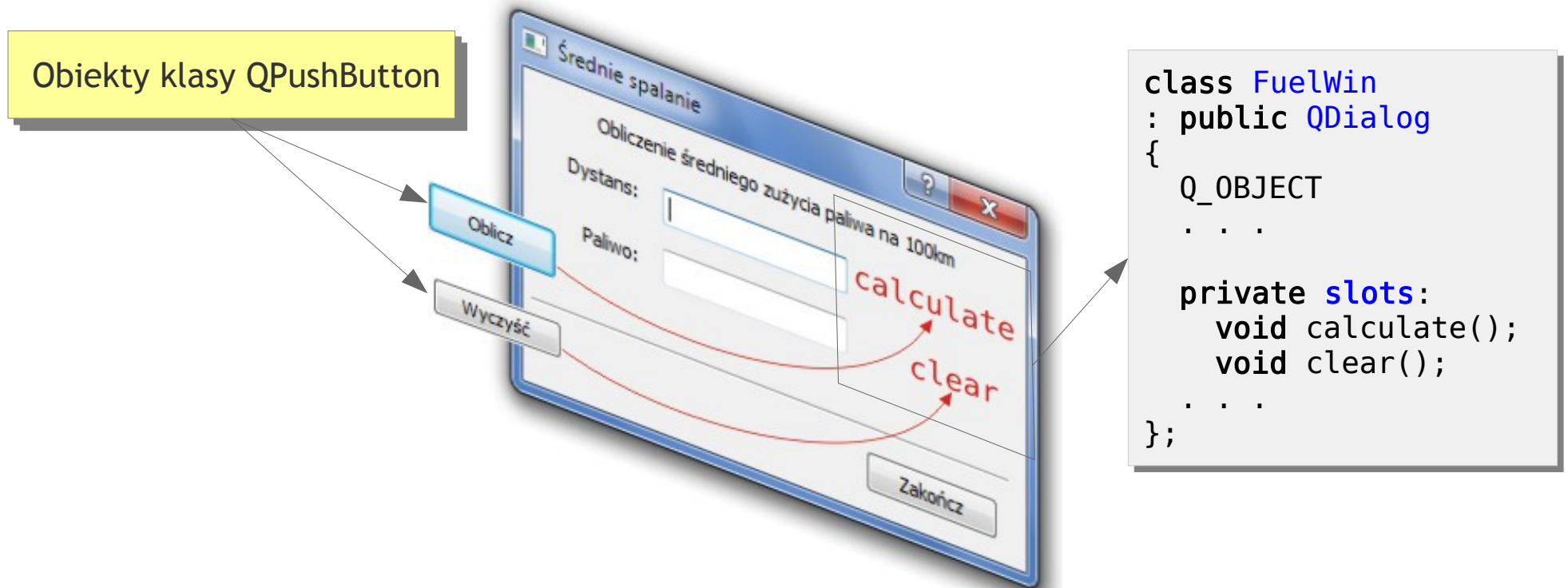
Sygnały i sloty – koncepcja

- Powiązanie sygnał-podprogram obsługi realizuje funkcja *connect*.
- Aby takie powiązanie zadziało, podprogram obsługi musi *slotem* — specjalnie zdefiniowaną funkcją składową pewnej klasy.



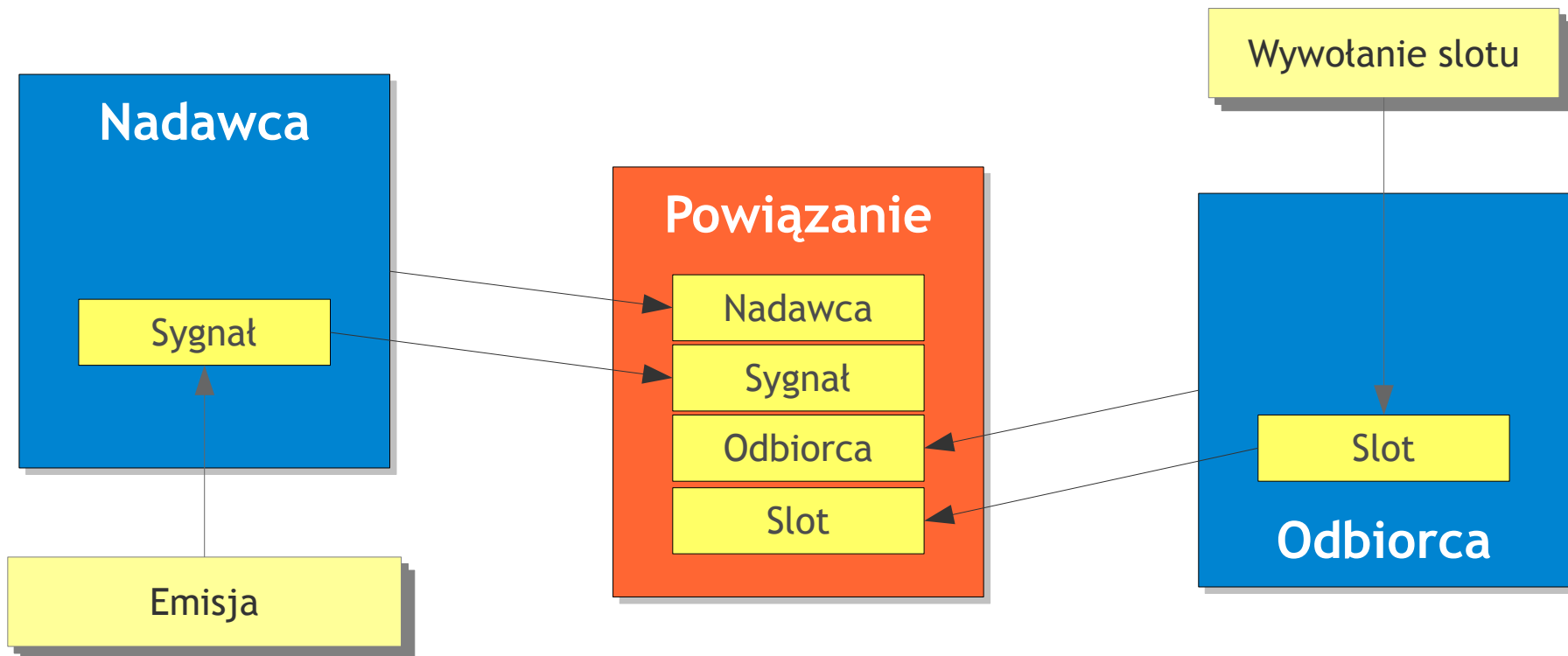
Sygnały i sloty – koncepcja

- Sloty i sygnały są zgodne z koncepcją programowania obiektowego.
- *Sygnał* jest emitowany przez pewien obiekt Qt — najczęściej jest to obiekt klasy interfejsowej, np. klasy *QPushButton*.
- *Slot* jest funkcją składową pewnego obiektu Qt — często klasy interfejsowej, np. klasy reprezentującej okno, ale również klasy niezwiązanej z GUI.



Sygnały i sloty – elementy

- **Nadawca sygnału** — obiekt klasy Qt, emituje sygnał o określonej nazwie.
- **Odbiorca sygnału** — obiekt klasy Qt, odbiera sygnał poprzez aktywowanie odpowiedniego slotu.
- **Powiązanie** — para *sygnał-slot*, kojarząca *sygnał* nadawcy z *odbiorcą*.



Czym jest slot?

- Jest *funkcją składową* pewnej klasy, zwykle jej rezultatem jest *void*.
- Jest *implementowany* i może być *wywoływany* jak zwykła funkcja składowa.
- Jest *deklarowany* w odmienny sposób — w sekcji *slots* klasy:

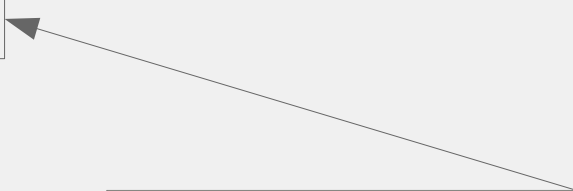
```
class JakasKlasa : public QObject
{
    Q_OBJECT
    . . .

    public slots:
        void publicznySlot();

    protected slots:
        void zabezpieczonySlot();

    private slots:
        void prywatnySlot();

    . . .
};
```



Mechanizm slotów i sygnałów działa dla obiektów klas pochodnych względem QObject

Dlaczego używamy slotów?

- Slot jest normalną funkcją C++, i może być wywoływany jak każda inna funkcja.
- Slot tym różni się od zwykłej funkcji, że może być wiązany z sygnałem (sygnałami), a wywołanie slotu jest wynikiem emisji sygnału (sygnałów).
- Aby mechanizm slotów i sygnałów działał, konieczne jest wykorzystanie klasy bazowej *QObject* lub jej pochodnej.
- Program *qmake* aktywuje *moc* i realizuje odpowiednie przetworzenie slotów i sygnałów do postaci standardowych konstrukcji C++.

Czym jest sygnał?

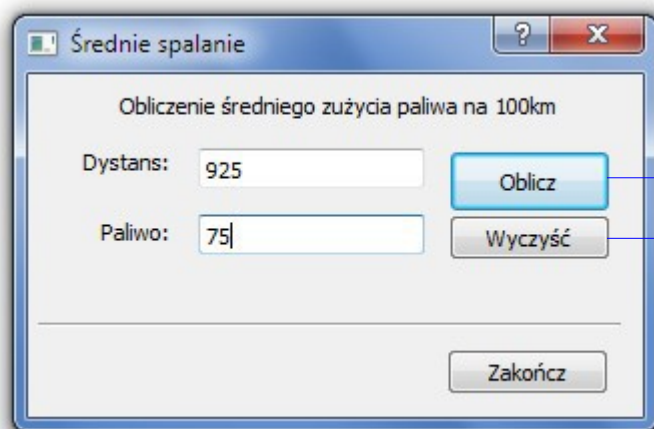
- Sygnał jest *komunikatem* reprezentowanym w definicji klasy jak *deklaracja funkcji* o rezultacie *void*.
- Sygnał ma tylko *deklarację*, programista *nie definiuje* sygnału.
- Programista *pisze nagłówek* funkcji-sygnału — nazwa sygnału oraz listę parametrów sygnału.
- Programista *nie pisze ciała* funkcji-sygnału — ciało jest *generowane automatycznie* przez *moc*.
- Sygnały definiuje się w sekcji *signals*:

```
class Jakasklasa : public QObject
{
    Q_OBJECT
    . . .
signals:
    void signalBezParametrow();
    void signalZParametrami( int p );
    . . .
};
```

Mechanizm slotów i sygnałów działa dla obiektów klas pochodnych względem QObject

Emisja sygnałów

- Emisja sygnału realizowana jest przez słowo kluczowe *emit*, stanowiące rozszerzenie Qt obsługiwane przez *moc*.
- *Nadawca* emitujący sygnał zazwyczaj *nie wie*, kto jest jego *odbiorcą*.
- Emisja sygnału jest pewnego rodzaju *rozgłoszeniem* (ang. *broadcast*) informacji o zaistniałym zdarzeniu.
- Emisja sygnału ma *konsekwencje* — powoduje *uaktywnienie slotu*, o ile jakiś został do sygnału *przywiązany*.



```
{  
    . . .  
    emit clicked();  
    . . .  
}
```

```
{  
    . . .  
    emit clicked();  
    . . .  
}
```

Powiązanie sygnał-slot

- Powiązanie sygnał-slot realizowane poprzez wywołanie funkcji *connect*.
- Ta funkcja zdefiniowana jest w klasie *QObject*.
- Dla elementów GUI projektowanych w systemie QtDesigner, powiązania zapisywane są w odpowiednim pliku opisu interfejsu *.ui.
- Powiązania mogą być ustanawiane i zrywane dynamicznie w czasie działania programu.

```
QObject::connect( nadawca, SIGNAL( sygnatura ), odbiorca, SLOT( sygnatura ) );
```

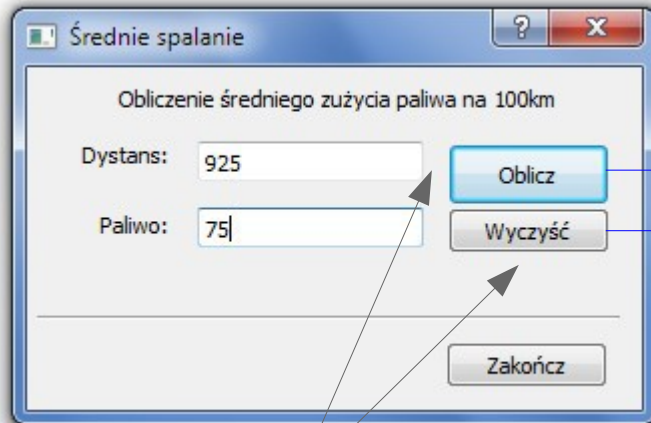
- Ponieważ syntaktycznie sygnały i sloty deklarowane są jak funkcje, ich sygnatury mają postać:

```
nazwaSygnałuLubSlotu( <lista argumentów> )
```

Powiązanie sygnał-slot

```
connect( ui->pushButtonCalculate, SIGNAL( clicked() ), this, SLOT( calculate() ) );  
connect( ui->pushButtonClear, SIGNAL( clicked() ), this, SLOT( clear() ) );
```

```
class FuelWin : public QDialog  
{  
    Q_OBJECT  
private slots:  
    void calculate();  
    void clear();  
    . . .  
};
```



ui->pushButtonCalculate
ui->pushButtonClear

clicked()

clicked()

Sygnały

connect

connect

Sloty

```
void FuelWin::calculate()  
{  
    . . .  
}
```

```
void FuelWin::clear()  
{  
    . . .  
}
```

Powiązanie sygnał-slot, parametry

```
bool QObject::connect (
    senderQObjectPtr,
    SIGNAL( signalName(argumentList)),
    receiverQObjectPointer,
    SLOT(slotName(argumentList))
    OptionalConnectionType
);
```

- Powiązanie sygnał-slot mogą zawierać parametry:

```
QLabel *label = new QLabel;
QScrollBar *scrollBar = new QScrollBar;

QObject::connect( scrollBar, SIGNAL( valueChanged( int ) ),
    label, SLOT( setNum( int ) ) );
```

- Parametry mogą tylko zawierać typy argumentów, nie mogą zawierać nazw:

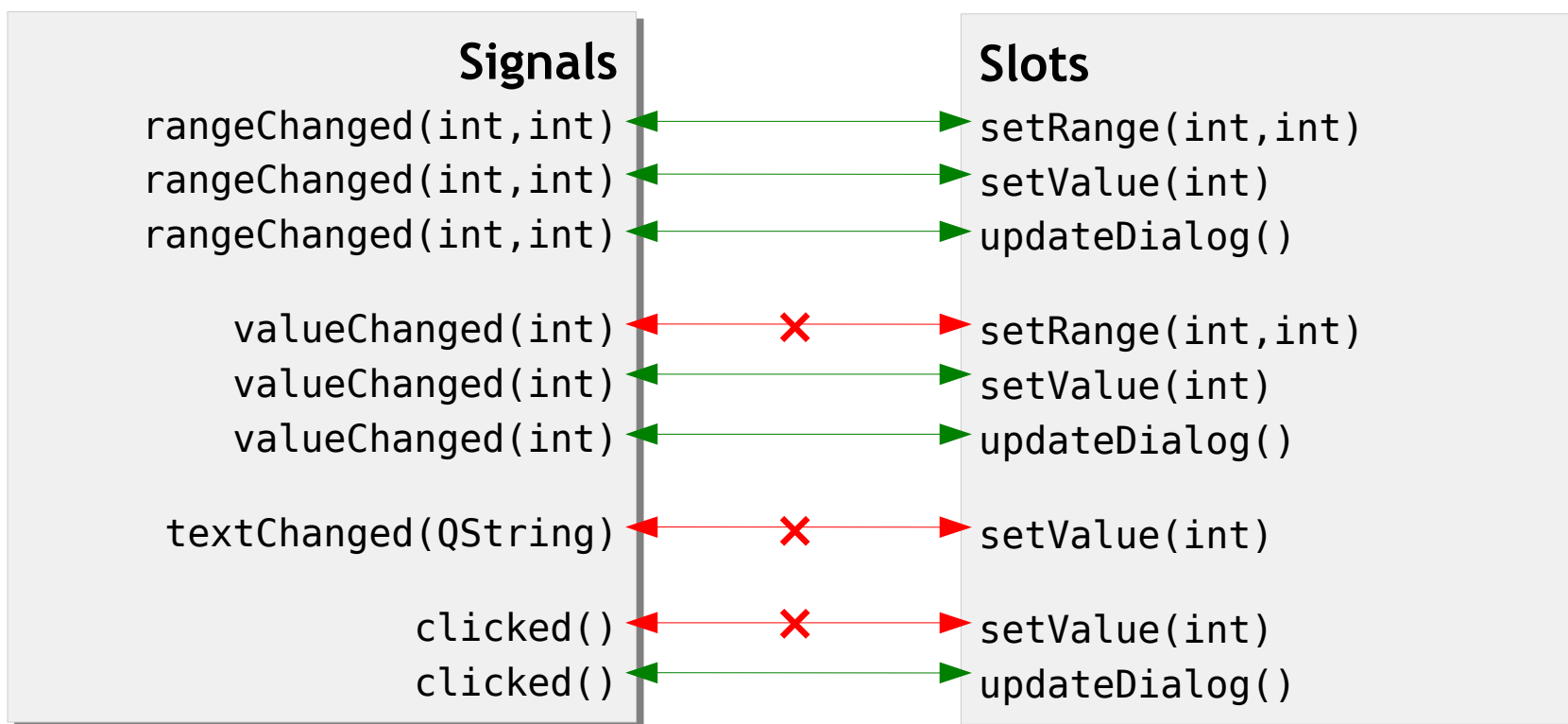
```
QLabel *label = new QLabel;
QScrollBar *scrollBar = new QScrollBar;

QObject::connect( scrollBar, SIGNAL( valueChanged( int n ) ),
    label, SLOT( setNum( int n ) ) );
```

Błędny zapis

Powiązanie sygnał-slot, parametry

- Listy parametrów sygnału i slotu powinny sobie *odpowiadać* — *liczbą i typami*.
- Dozwolone jest, gdy sygnał ma więcej parametrów, są one przy aktywowaniu slotu ignorowane.
- Sytuacja odwrotna jest niedozwolona — jak mówi dokumentacja: „*Qt can ignore arguments, but not create values from nothing*”.



Rozłączanie powiązania – funkcja disconnect

```
bool QObject::disconnect (
    senderQObjectPtr,
    SIGNAL( signalName(argumentList)),
    receiverQObjectPointer,
    SLOT(slotName(argumentList))
    OptionalConnectionType
);
```

Szczególne przypadki:

- Odłączenie odłączenie powiązań od wszystkich sygnałów:

```
disconnect( senderQObjectPtr, 0, 0, 0 );
```

- Odłączenie odłączenie powiązań od pewnego sygnału:

```
disconnect( senderQObjectPtr, SIGNAL( mySignal() ), 0, 0 );
```

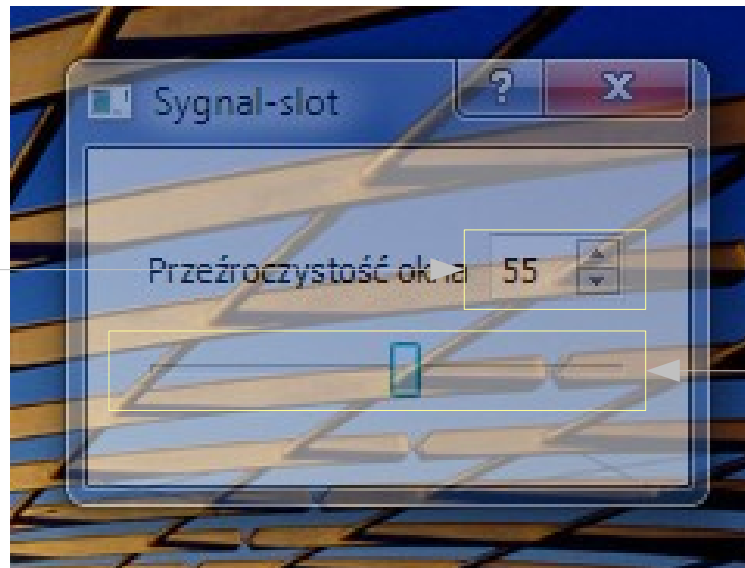
- Odłączenie konkretnego odbiorcy od nadawcy sygnału:

```
Disconnect( senderQObjectPtr, 0, receiverQObjectPointer, 0 );
```

Przykład wykorzystania standardowych sygnałów i slotów

Wykorzystanie standardowych sygnałów i slotów do powiązania elementu *QSlider*, *QSpinBox* jako elementów sterujących przezroczystością okna.

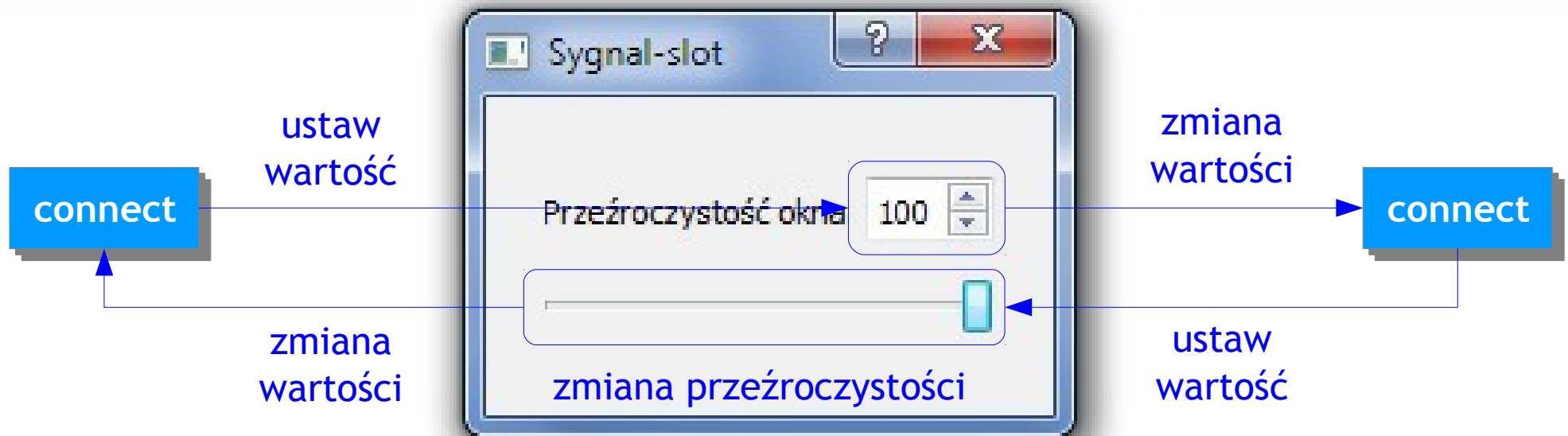
QSpinBox *spinBox



QSlider *horizontalSlider

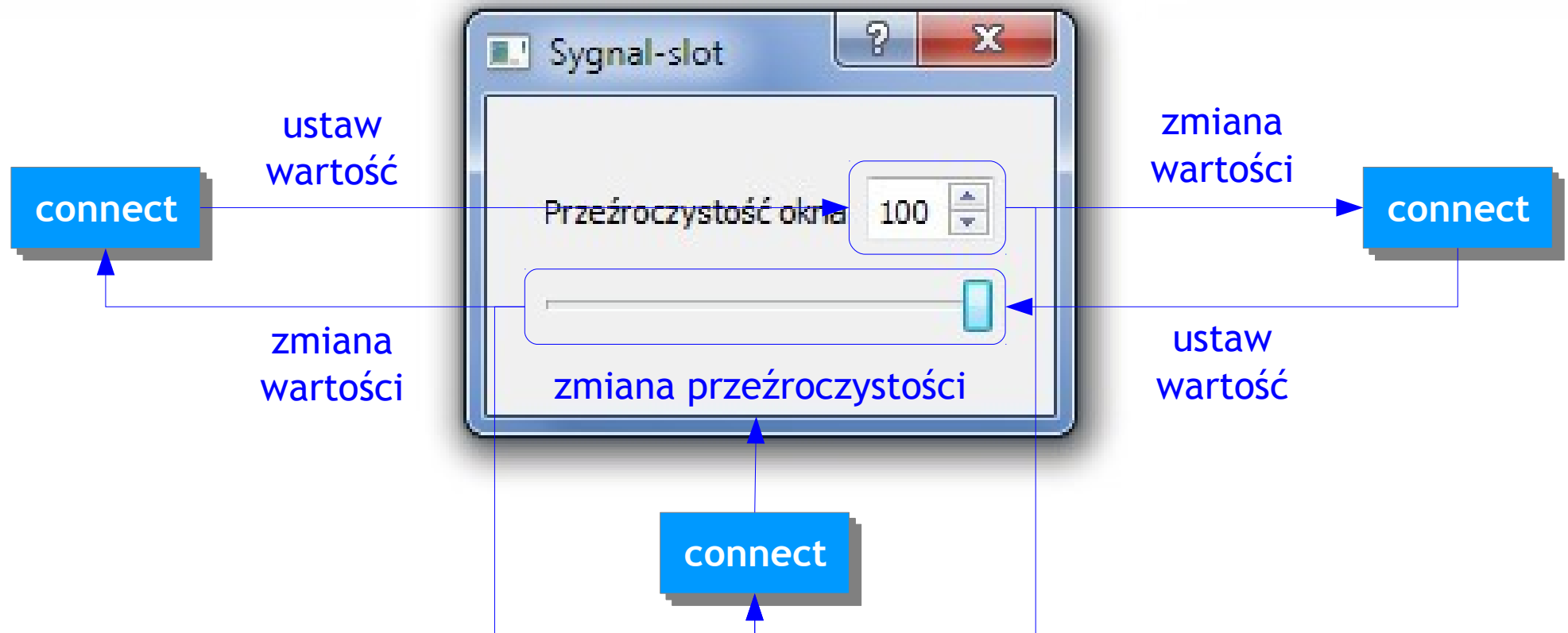
Przykład wykorzystania standardowych sygnałów i slotów

Zmiana wartości w obiekcie *QSpinBox* ma spowodować zmianę pozycji *QSlider*, zmiana pozycji obiektu *QSlider* ma spowodować zmianę wartości *QSpinBox*.



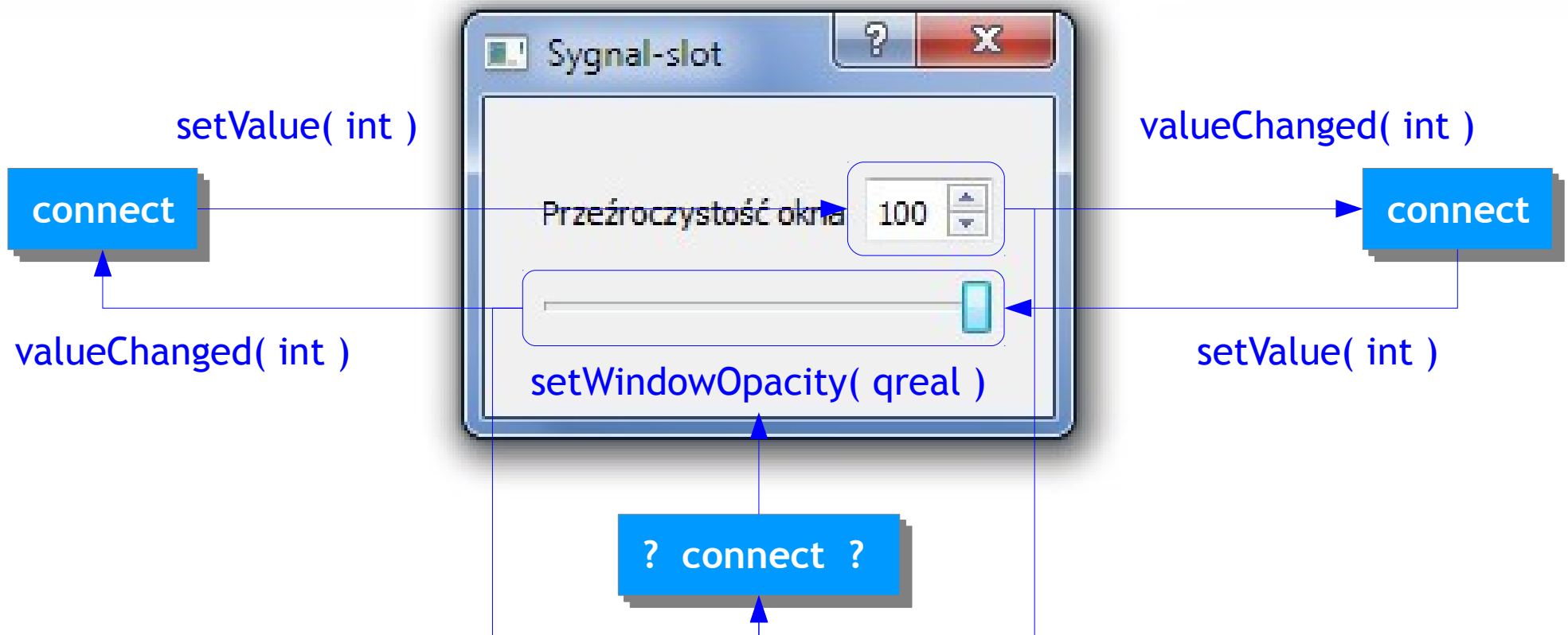
Przykład wykorzystania standardowych sygnałów i slotów

Zmiana wartości w obiekcie *QSpinBox* lub pozycji *QSlider* ma spowodować zmianę przeźroczystości okna (funkcja *setWindowOpacity*).



Przykład wykorzystania standardowych sygnałów i slotów

Zmiana wartości w obiekcie *QSpinBox* lub pozycji *QSlider* ma spowodować zmianę przeźroczystości okna (funkcja *setWindowOpacity*).



Przykład wykorzystania standardowych sygnałów i slotów

```
Dialog::Dialog(QWidget *parent) : QDialog(parent), ui(new Ui::Dialog)
{
    ui->setupUi(this);

    // Powiazanie: zmiana pozycji suwaka → zmiana wartosci spinbox'a
    connect( ui->horizontalSlider, SIGNAL( valueChanged( int ) ),
            ui->spinBox, SLOT( setValue( int ) ) );

    // Powiazanie: zmiana wartosci spinbox'a → zmiana pozycji suwaka
    connect( ui->spinBox, SIGNAL( valueChanged( int ) ),
            ui->horizontalSlider, SLOT( setValue( int ) ) );

    // ??? Powiazanie: zmiana wartosci spinbox'a → zmiana przezroczystosci
    connect( ui->spinBox, SIGNAL( valueChanged( int ) ),
            this, SLOT( setWindowOpacity( qreal ) ) );
}
```

Niezgodność typów

Przykład wykorzystania standardowych sygnałów i slotów

```
class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog( QWidget *parent = 0 );
    ~Dialog();

public slots:
    void setTransparency( int val );

private:
    Ui::Dialog *ui;
};

. . .

void Dialog::setTransparency( int val )
{
    setWindowOpacity( ( qreal )val / 100 );
}
```

Przykład wykorzystania standardowych sygnałów i slotów

```
Dialog::Dialog(QWidget *parent) : QDialog(parent), ui(new Ui::Dialog)
{
    ui->setupUi(this);

    // Powiazanie: zmiana pozycji suwaka → zmiana wartosci spinbox'a
    connect( ui->horizontalSlider, SIGNAL( valueChanged( int ) ),
            ui->spinBox, SLOT( setValue( int ) ) );

    // Powiazanie: zmiana wartosci spinbox'a → zmiana pozycji suwaka
    connect( ui->spinBox, SIGNAL( valueChanged( int ) ),
            ui->horizontalSlider, SLOT( setValue( int ) ) );

    // Powiazanie: zmiana wartosci spinbox'a → zmiana przezroczystosci
    connect( ui->spinBox, SIGNAL( valueChanged( int ) ),
            this, SLOT( setTransparency( int ) ) );
}
```

Dziękuję za uwagę

Pytania? Polemiki?
Teraz, albo:
roman.siminski@us.edu.pl