

Observe your neighbors and remove your seatbelt

Type introspection and type-unsafety in Qt

Stephen Kelly
stephen.kelly@kdab.com
KDAB

Stephen Kelly

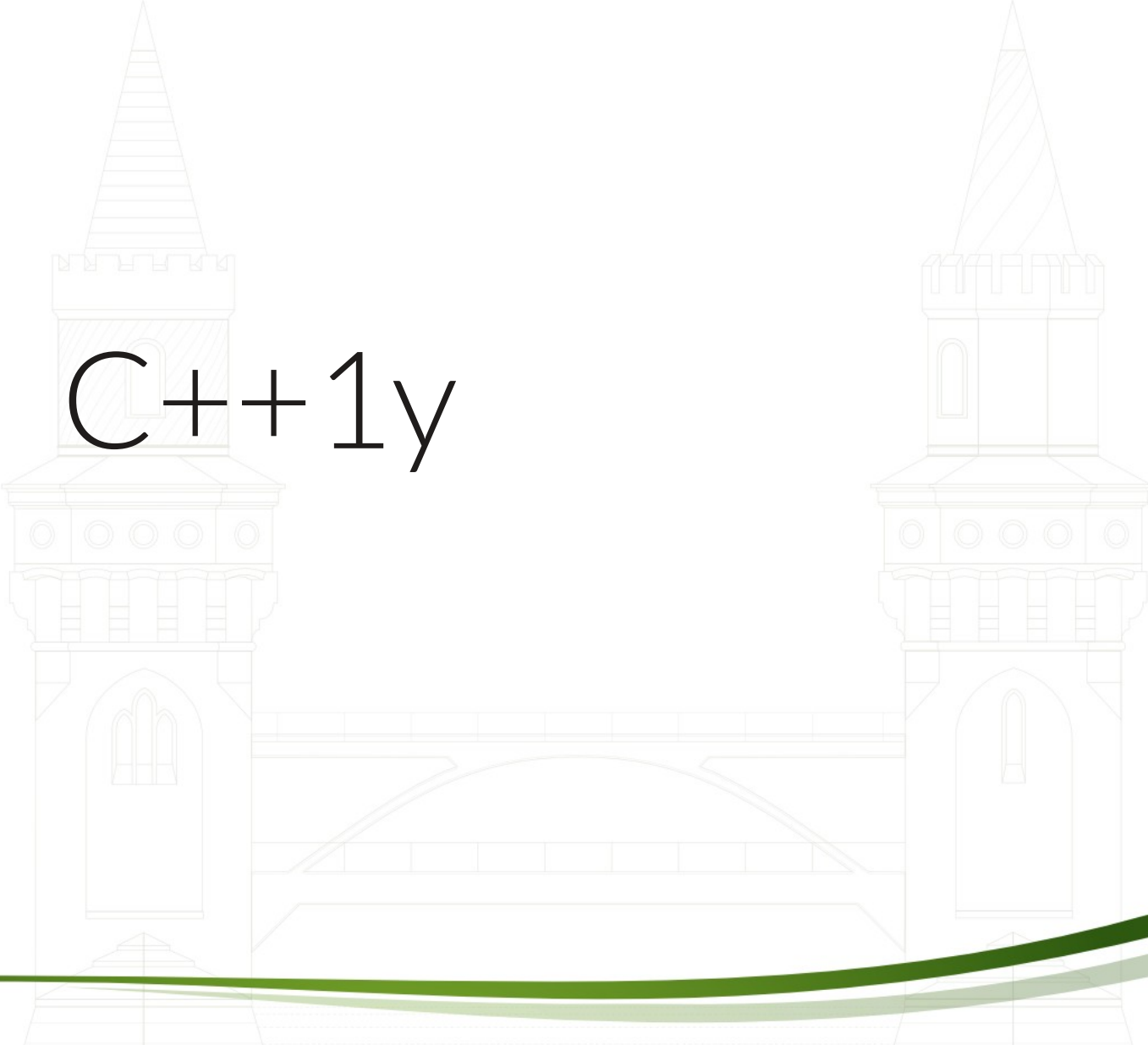
- C++/Qt user since 2006
- KDE contributor since 2007
- Qt contributor since 2009
- CMake contributor since 2011
- Interested in clang tooling
- Living in Berlin



oCaml scala ActionScript
javascript ruby Haskell
scheme Erlango
C C++ Java
python lisp

shauryashaurya



C++1y

A faint, light green line-art illustration of a castle with two towers and a bridge, serving as a background for the text.

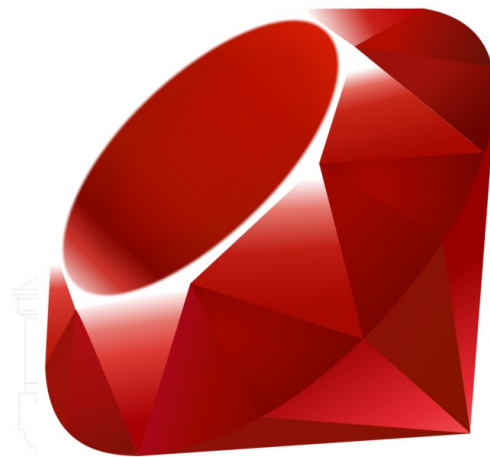
  **JTC1/SC22/WG21 - Papers 2012**

WG21 Number	Title	Author
N3326	Sequential access to data members and base sub-objects	Andrzej Krzemieński
N3340	Rich Pointers	D. M. Berris, M. Austern, L. Crowl
N3390	Any Library Proposal (Revision 1)	B. Dawes, K. Henney
N3403	Use Cases for Compile-Time Reflection	Mike Spertus
N3410	Rich Pointers with Dynamic and Static Introspection	D. M. Berris, M. Austern, L. Crowl, L. Singh
N3437	Type Name Strings For C++	Axel Naumann
N3449	Open and Efficient Type Switch for C++	B. Stroustrup, G. Dos Reis, Y. Solodkyy

Motivation

- Language Binding
 - Domain Specific Languages
- Introspection/Reflection
 - Tooling
 - Testing/Unit tests

Language Bindings



Problems

- Moving types through API boundaries
- Type to string conversion
- String to type conversion
- Finding the capabilities of types
- Introspection

N3437:

Type Name Strings For C++

6 Summary

While C++ is providing more and more compile-time-centric features, string-based type identification enables C++ to cover more use cases.

This paper has shown that type name strings can dramatically simplify problems that are currently impossible to solve without crude external scaffolding.

Scaffolding

Runtime features

- QVariant
- QMetaType
- QObject
- QMetaObject



Scaffolding

Runtime features

- QVariant
- QMetaType
- QObject
- QMetaObject

Compile-time features

- Macros
- Templates

Scaffolding

Runtime features

- QVariant
- QMetaType
- QObject
- QMetaObject

Compile-time features

- Macros
- Templates

?

Qt classes

- QVariant
- QMetaType
- QObject
- QMetaObject



Qt classes

- QVariant
- QMetaType
- QObject
- QMetaObject



QVariant

```
class QVariant {  
    ...  
private:  
    union Data {  
        char c;  
        int i;  
        bool b;  
        double d;  
        qlonglong ll;  
        void *ptr;  
    } data;  
};
```

QVariant



QVariant



```
class QVariant {  
private:  
    union Data {  
        ...  
    } data;  
    int type;  
};
```



Built-in Meta-types

- Implicit constructors

- `QVariant var1 = 42;`
- `QVariant var2 = 3.14158`
- `QVariant var3 = "Hello, world!";`

- Static factory

- `QVariant var4 = QVariant::fromValue<MyClass*>(myObject);`
- `QVariant var5 = QVariant::fromValue<EnumType>(myEnumVal);`

- Accessors

- `int i = var1.toInt();`
- `MyClass *obj = var4.value<MyClass*>();`

MetaTypes

```
QVariant QVariant::fromValue<T>(T t)
{
    int id = QMetaTypeId<T>::qt_metatype_id();
    return QVariant(id, reinterpret_cast<void*>(&t));
}
```

MetaTypes

```
T QVariant::value<T>() const
{
    int id = QMetaTypeId<T>::qt_metatype_id();
    if (this->userType() == id)
        return *reinterpret_cast<T*>(this->data);
    return T();
}
```

User-defined types

```
struct Customer
{
    QString name;
    nsCity *city;
};
Q_DECLARE_METATYPE(Customer)
```

- `QVariant variant = QVariant::fromValue(cust);`
- `Customer cust = variant.value<Customer>();`

MetaTypes

```
#define Q_DECLARE_METATYPE(TYPE) \
    template <> \
    struct QMetaTypeId< TYPE > \
    { \
        static int qt_metatype_id() \
        { \
            return qRegisterMetaType< TYPE >(#TYPE); \
        } \
    };
```

MetaTypes

```
template<typename T>
int qRegisterMetaType(const char *typeName)
{
    QMetaType::Destructor dtor = qMetaTypeDeleteHelper<T>;
    QMetaType::Constructor ctor = qMetaTypeConstructHelper<T>;
    return QMetaType::registerType(typeName, dtor, ctor);
}
```


MetaTypes

```
class QMetaType
{
    static const char *typeName(int id);
    static int type(const char *typeName);
}

int qRegisterMetaType<T>() {
    return QMetaTypeId<T>::qt_metatype_id();
}
```

Intermediate summary

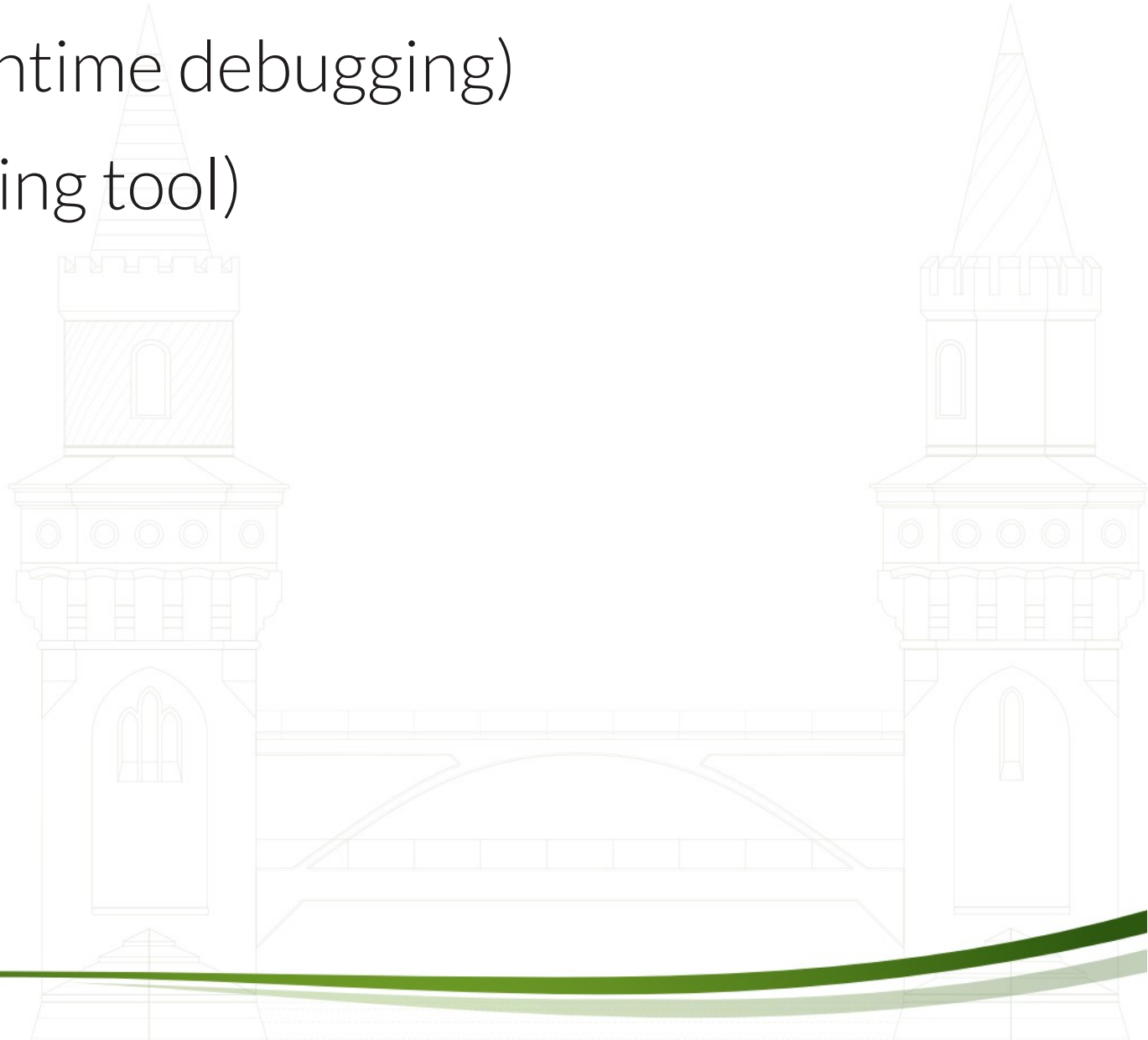
- QMetaType maps integer id ↔ type string
- Strings extracted at compile-time
- Mapping defined at and available at run-time

Qt classes

- QVariant
- QMetaType
- QObject
- QMetaObject



- GammaRay (Runtime debugging)
- Squish (Gui testing tool)



Scaffolding

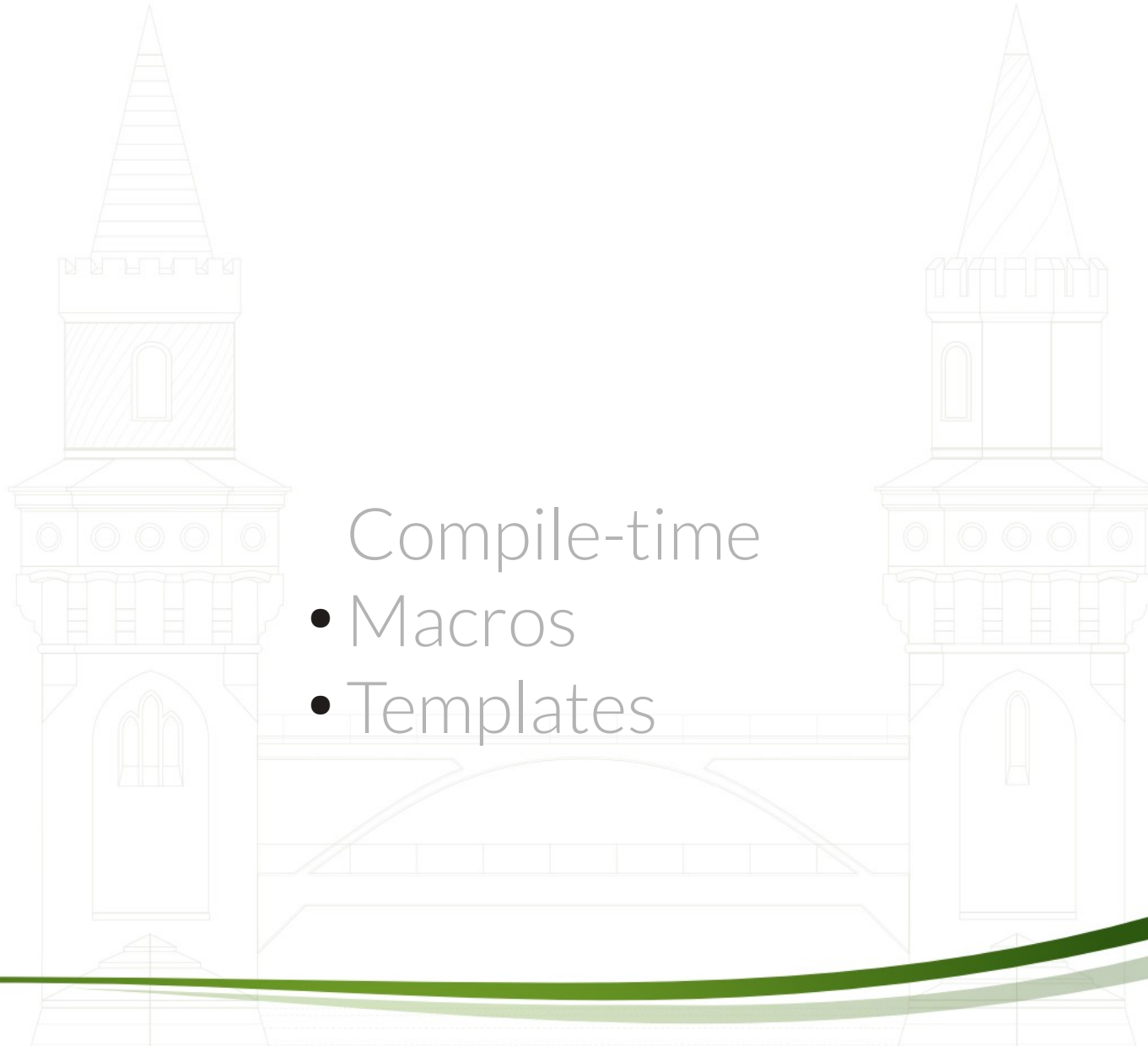
Runtime

- QVariant
- QMetaType
- QObject
- QMetaObject

Compile-time

- Macros
- Templates

?



Scaffolding

Runtime

- QVariant
- QMetaType
- QObject
- QMetaObject

Code-generation

- moc
- QMetaObject
- qt_metacall()

Compile-time

- Macros
- Templates

Qt Properties

```
class Customer : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name READ name)
public:
    QString name() const;
}

QObject *o = new Customer;
QVariant var = o->property("name");
QString name = var.value<QString>();
```

QMetaObject

```
static const uint qt_meta_data_Customer[] = {
    // content:
        6,      // revision
        0,      // classname
        0,    0, // classinfo
        0,    0, // methods
        1,  14, // properties
        0,    0, // enums/sets
        0,    0, // constructors
        0,      // flags
        0,      // signalCount

    // properties: name, type, flags
        17,    9, 0x0a095001,

        0      // eod
};
```


QMetaObject

```
static const char qt_meta_stringdata_Customer[] = {  
    "Customer\0QString\0name\0"  
};
```

```
const QMetaObject Customer::staticMetaObject = {  
    { &QObject::staticMetaObject, // Base class  
      qt_meta_stringdata_Customer,  
      qt_meta_data_Customer  
    }  
};
```

QMetaObject

```
int Customer::qt_metacall(QMetaObject::Call _c, int _id, void **_a)
{
    _id = QObject::qt_metacall(_c, _id, _a);
    if (_id < 0)
        return _id;
    if (_c == QMetaObject::ReadProperty) {
        void *_v = _a[0];
        switch (_id) {
            case 0: *reinterpret_cast< QString*>(_v) = name();
                    break;
        }
        ...
    }
```

Bindings



```
function myFunc(customer) {  
    var name = customer.name;  
}
```



```
def myFunc(customer):  
    name = customer.name
```

```
Shopping list for {{ person.name }}  
{% for item in itemlist %}  
  * {{ item.name }} (${{ item.cost }})  
{% endfor %}
```

Qt 5 Improvements

Built-in `qobject_cast`

```
QVariant v1 = QVariant::fromValue(new QLabel);  
Q_ASSERT(v1.canConvert<QWidget*>());  
QWidget *w1 = v1.value<QWidget*>();  
Q_ASSERT(v1.canConvert<QObject*>());  
QObject *o1 = v1.value<QObject*>();  
QString s1 =  
o1->property("text").value<QString>();
```

Qt 5 Improvements

TU 1

```
#include <QLabel>
#include <QVariant>
```

```
QVariant getVar()
{
    return
        QVariant::fromValue(
            new QLabel);
}
```

TU 2

```
#include <QVariant>
```

```
{
    QVariant v = getVar();
    setVar(v);
}
```

TU 3

```
#include <QObject>
#include <QVariant>
```

```
void setVar(QVariant v)
{
    QObject *o
        = v.value<QObject*>();
    o->property("text");
}
```

Qt 5 Improvements

TU 1

```
#include <QLabel>
#include <QVariant>
```

```
QVariant getVar()
{
    return
        QVariant::fromValue
            <QObject*>
            (new QLabel);
}
```

TU 2

```
#include <QVariant>
```

```
{
    QVariant v = getVar();
    setVar(v);
}
```

TU 3

```
#include <QObject>
#include <QVariant>
```

```
void setVar(QVariant v)
{
    QObject *o
        = v.value<QObject*>();
    o->property("text");
}
```

Qt 5 Improvements

Automatic MetaType declaration

- No need for **Q_DECLARE_METATYPE**
 - QObject subclasses
 - Qt Containers
 - Smart (Qt) pointers

Qt 5 Improvements

Automatic MetaType declaration

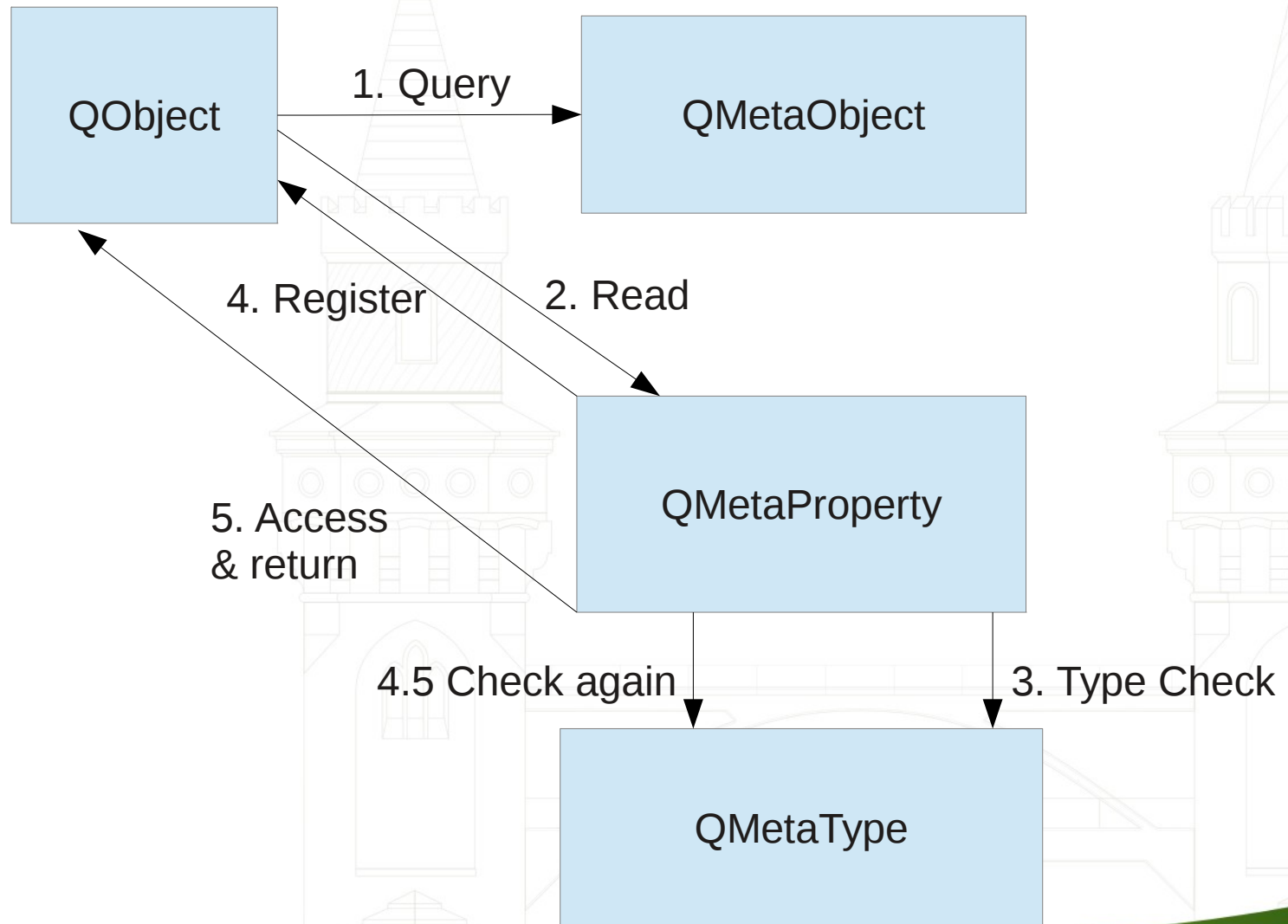
- No need for **Q_DECLARE_METATYPE**
 - `QVariant::fromValue(myWidget);`
 - `QVariant::fromValue(QList<int>());`
 - `QVariant::fromValue(QList<MyWidget* >());`
 - `QVariant::fromValue(QSharedPointer<MyWidget>());`
 - `QVariant::fromValue(QVector<QSharedPointer<MyWidget>>());`

Qt 5 Improvements

Automatic MetaType registration

- No more need for `qRegisterMetaType` (almost)
- Code generated by moc to register types

Reading properties



Summary

- Runtime type introspection in Qt
- Language bindings and tools
- Runtime registration
- Type conversions
- Inspecting properties, signals, slots

Thank You

Questions ?

stephen.kelly@kdab.com