Code less.
Create more.
Deploy everywhere.

C++11 in Qt 5: Challenges & Solutions

Thiago Macieira, Qt Core Maintainer

Aspen, May 2013

Who am I?

- **Open Source developer for 15 years**
- **C++ developer for 13 years**
- **Software Architect at Intel’s Open Source Technology Center (OTC)**
- **Maintainer of two modules in the Qt Project**
  - QtCore and QtDBus
- **MBA and double degree in Engineering**
- **Previously, led the “Qt Open Governance” project**

Qt 5

First major version in 7 years

Goals:

- New graphics stack
- Declarative UI design with QML
- More modular for quicker releases
- New, modern features
- Mostly source-compatible w/ Qt 4

Release status:

- Qt 5.0.2 released in April
- Qt 5.1.0 beta 1 released in May 14, 2013

The C++11 challenge

- We would have liked to switch

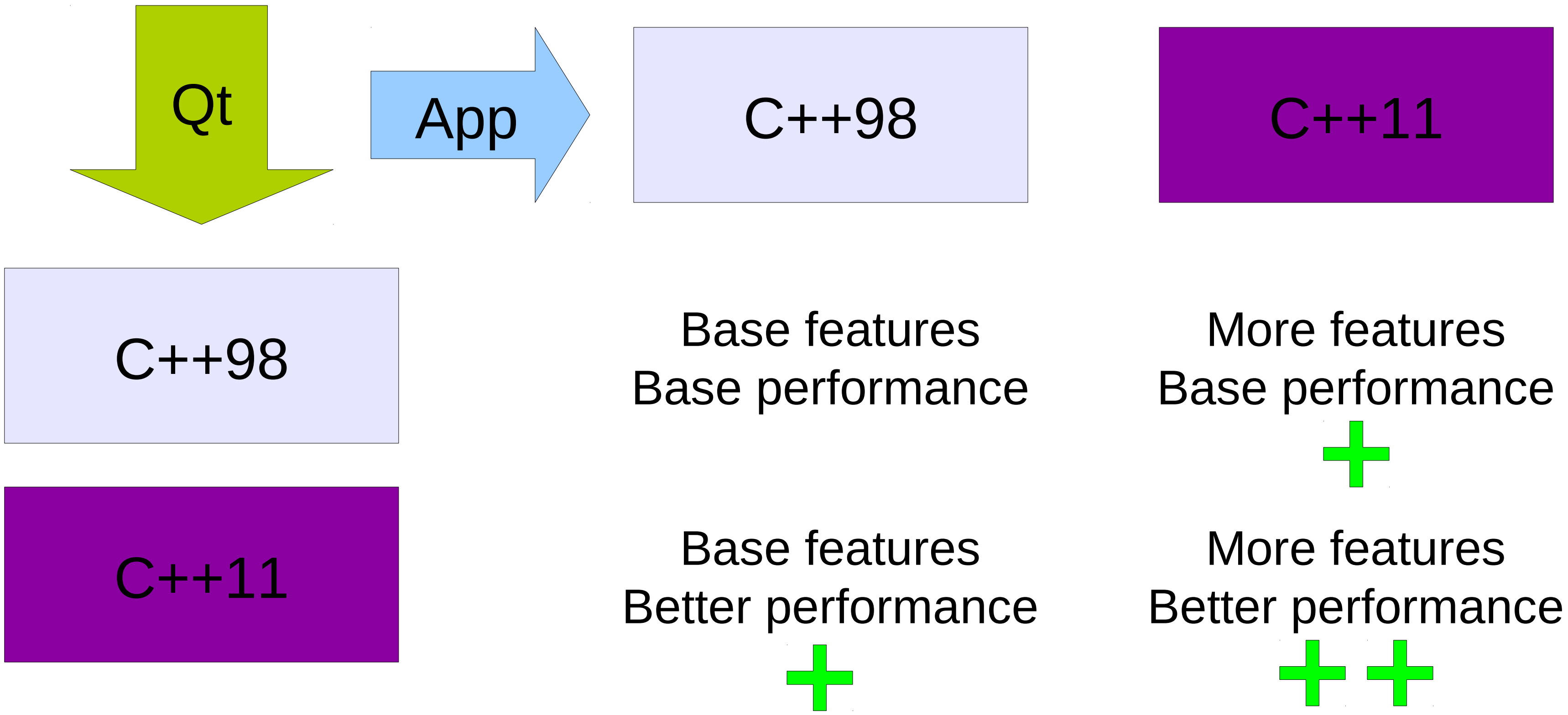
“C++98 costs more”

- But we need to maintain compatibility
  - MSVC 2008
  - GCC 4.2
  - Commercial Unix compilers (AIX and Solaris)

A look at C++11 in Qt 5.1

- Added a lot of C++11 support to Qt 4.8, Qt 5.0 and Qt 5.1
- Lots of C++11 stuff left to do for 5.2:
  - Move Semantics (containers and containees)
  - add constexpr to more classes / functions
  - 'explicit' missing on N-ary ctors, N ≥ 2

We want



	C++98	C++11
C++98	Base features Base performance	More features Base performance +
C++11	Base features Better performance +	More features Better performance + +

Solution for Qt’s own code

- Enable C++11 automatically
- **Must** still build under C++98 mode
- **Must** provide the same library ABI in either mode

- **Can** use C++11 features with fallback
- **Can** offer new features in .h files (inlines) under #ifdef

Compiler support in Qt

	C++11 support	Minimum version
GCC	• Automatically enabled	4.4 (except on Mac)
Clang	• Automatically enabled **if** using libc++ – Default as of Qt 5.1	Apple Clang: 4.0 Official: 3.0
ICC	• Automatically enabled	12.0
Visual Studio	• Cannot be disabled	2008

It has not been without problems

- Compiler bugs
- Different implementations
- Implementations of earlier papers / draft standard
- Difficulty in making the changes

Some C++11 features can be used under #ifdef

- Macros for #ifdef: Q_COMPILER_xxx

  Q_COMPILER_CONSTEXPR, Q_COMPILER_RVALUE_REFS, Q_COMPILER_VARIADIC_TEMPLATES, etc.

- All “interesting” C++11 features listed and checked

```
#ifdef Q_COMPILER_RVALUE_REFS
    inline QList(QList<T> &&other) : d(other.d)
    { other.d = const_cast<QListData::Data *>(&QListData::shared_null); }
    inline QList &operator=(QList<T> &&other)
    { qSwap(d, other.d); return *this; }
#endif
#ifdef Q_COMPILER_INITIALIZER_LISTS
    inline QList(std::initializer_list<T> args)
        : d(const_cast<QListData::Data *>(&QListData::shared_null))
    { qCopy(args.begin(), args.end(), std::back_inserter(*this)); }
#endif
```

Some C++11 features don’t require #ifdef

- #ifdef is too ugly
  - Q_DECL_EQ_DELETE
  - Q_DECL_EQ_DEFAULT
  - Q_DECL_CONSTEXPR
  - Q_DECL_NOEXCEPT
  - Q_DECL_NOEXCEPT_EXPR(x)
  - Q_NULLPTR

No #ifdef

```
Q_DECL_CONSTEXPR inline QFlags(Enum f) : i(f) {}
Q_DECL_CONSTEXPR inline QFlags(Zero = 0) : i(0) {}
Q_DECL_CONSTEXPR inline QFlags(QFlag f) : i(f) {}
```

```
template<typename T> inline uint qHash(const T &t, uint seed)
    Q_DECL_NOEXCEPT_EXPR(noexcept(qHash(t)))
{ return (qHash(t) ^ seed); }
```

Some C++11 features are also enabled in C++98

- Macros for C++98 extensions by some compilers
  - Q_ALIGNOF   GCC's `__alignof__`, MSVC's `__alignof`
  - Q_DECL_OVERRIDE   MSVC's `override`
  - Q_DECL_FINAL   MSVC's `sealed`
- Or equivalent behaviour
  - Q_DECL_NOTHROW   MSVC's `nothrow()`, not GCC's
  - Q_DISABLE_COPY   declare copy constructor and assignment op
  - Q_STATIC_ASSERT   uses `sizeof(QStaticAssertFailure<!!(Condition)>)`

```
Q_CORE_EXPORT uint qHash(const QByteArray &key, uint seed = 0) Q_DECL_NOTHROW;
Q_CORE_EXPORT uint qHash(const QString &key, uint seed = 0) Q_DECL_NOTHROW;
```

Some features are almost never used

- Language syntax features that don’t add performance
  - Adding #ifdef would reduce readability
- Examples:
  - Angle bracket for templates without space (>> vs > >)
  - Auto types
  - Class enum
  - Delegating constructors
  - Initialisation of non-static members in the class body
  - Lambdas*
  - New function declaration syntax
  - Range `for`
  - Raw strings & Unicode strings*
  - Thread-safe initialisation of function statics*

* not directly

Close to no use of Standard Library features

Standard Library features

- Features coming too slowly to the Standard Library
- No reasonable way of detecting them
- We end up duplicating, with Qt API (e.g. QSharedPointer, QEnableIf)

Core language features

- Features that cannot be implemented without compiler help:
  - `<initializer_list>`
  - `<type_traits>`
- Trouble for:
  - Clang with GCC’s headers (Mac OS X)
  - GCC with Dinkumware headers (QNX)

The past

Qt and C++98

- C++98 support took a long time
- MS Visual Studio 6 support dropped only with Qt 4.6 (Dec/2009)
  - QT_NO_MEMBER_TEMPLATES
  - QT_NO_PARTIAL_TEMPLATE_SPECIALIZATION
  - QT_NO_TEMPLATE_TEMPLATE_PARAMETERS
  - Q_TYPENAME (no typename support)
- Standard Library is a requirement only with Qt 5.0 (Dec/2012)
- Qt 5 now requires all C++98 features

All C++98 features? No, one remaining...

Q_NO_TEMPLATE_FRIENDS

```
#if defined(Q_NO_TEMPLATE_FRIENDS)
public:
#else
    template <class X> friend class QSharedPointer;
    template <class X> friend class QWeakPointer;

#endif
    inline void ref() const { d->weakref.ref(); d->strongref.ref(); }
```

The present

Data alignment (Q_COMPILER_ALIGNOF, Q_COMPILER_ALIGNAS)

- Macro: Q_ALIGNOF
  - Always present (no #ifdef)
- Most compilers support alignof as an extension to C++98
  - MSVC, GCC, Clang, ICC, IBM xlC, Sun CC
  - Don’t need to wait for C++11!
- Emulation for older / exotic compilers

- Macro: Q_DECL_ALIGNED
  - Not always present!
- Difficult to emulate
  - Could be done with an unrestricted union
- No good solution

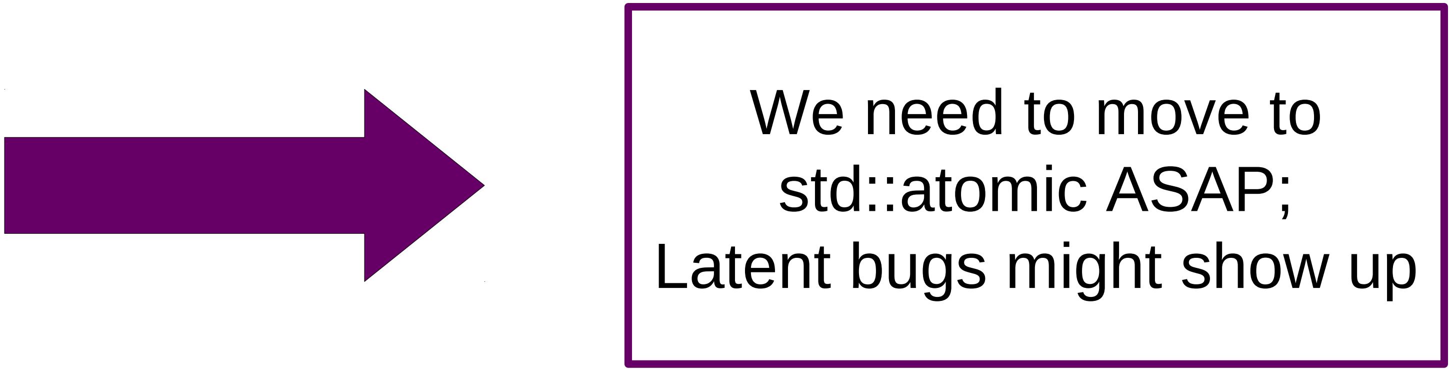
Atomics (Q_COMPILER_ATOMICS)

- Qt has had an atomics API since 4.4 (2008)
  - Has used them since 4.0 (2005)
- Most of it is written in assembly
- Only GCC 4.8 generates decent code for atomics
  - less-than-full memory barriers, no unnecessary locks
  - GCC 4.7 has support, but it’s reasonable only on x86 / x86-64

We need to keep our assembly for the time being

C++11 data races

- C++11 finally has a memory model supporting threads
- Compiler can be more aggressive when std::atomic is not in use
- `volatile` for threading was wrong!
- Qt atomic classes are abusing the compiler



We need to move to std::atomic ASAP; Latent bugs might show up

Example of data races

- Used to be `volatile` variables in the Qt event loop

```
@@ -266,8 +266,8 @@ void QEventLoop::exit(int returnCode)
     if (!d->threadData->eventDispatcher.load())
         return;

-    d->returnCode = returnCode;
-    d->exit = true;
+    d->returnCode.store(returnCode);
+    d->exit.storeRelease(true);
     d->threadData->eventDispatcher.load()->interrupt();
 }

@@ -281,7 +281,7 @@ void QEventLoop::exit(int returnCode)
 bool QEventLoop::isRunning() const
 {
     Q_D(const QEventLoop);
-    return !d->exit;
+    return !d->exit.loadAcquire();
 }
```

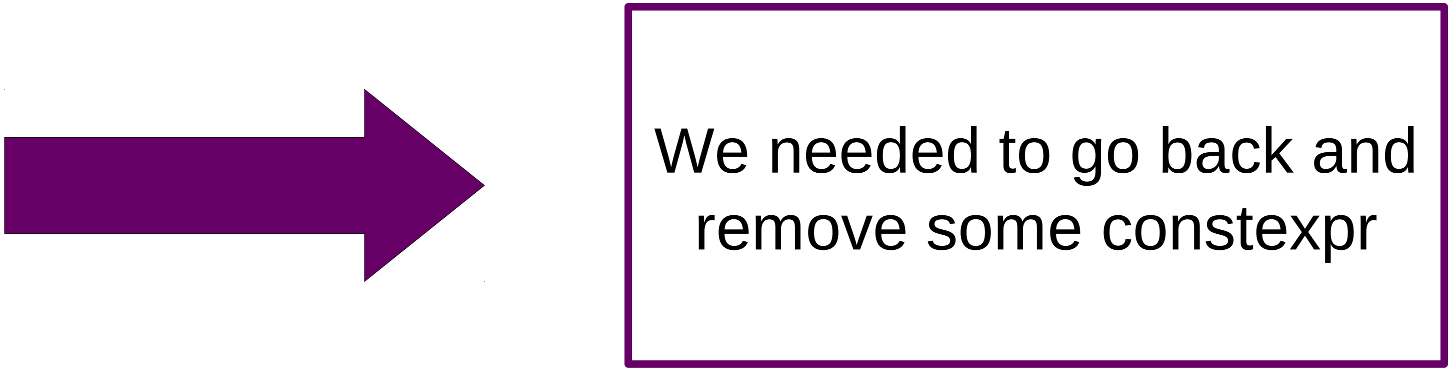
Future of the Qt atomics

- Qt 5.0 saw an overhaul of the code, to simplify
  - Uses CRTP to provide “virtual” methods without virtual tables
- Missing features:
  - Compare-and-swap that returns the current value
    `testAndSet + fetchAndStore = fetchAndTestAndSet ?`
  - `volatile` members
  - Maybe: implicit load, store and operator overloads, like std::atomic

```
T loadAcquire() const Q_DECL_NOTHROW { return Ops::loadAcquire(_q_value); }
void storeRelease(T newValue) Q_DECL_NOTHROW { Ops::storeRelease(_q_value, newValue); }
operator T() const Q_DECL_NOTHROW { return loadAcquire(); }
T operator=(T newValue) Q_DECL_NOTHROW { storeRelease(newValue); return newValue; }
```

`constexpr` support (Q_COMPILER_CONSTEXPR)

- We added Q_DECL_CONSTEXPR almost everywhere
- GCC and Clang did not implement full spec
  - Code broke with stricter, newer Clang
- Found compiler bugs...



We needed to go back and remove some constexpr

`constexpr` and static initialisation

- No load-time overhead
  - Objects can be static-initialised if they have a `constexpr` constructor (3.6.2 [basic.start.init] p2)
- Only used for QBasicAtomicInt and QBasicAtomicPointer
  - and QBasicMutex, but shhhh...
- For all other types, the recommendation is to avoid statics

Initialiser lists (Q_COMPILER_INITIALIZER_LISTS)

- Feature is provided in the Qt containers
- But never used by Qt itself...
- And it requires a header to be present

Feature is for the users, not for the library itself...

Brace initialisation

- Language syntactic sugar in most cases...
- Except where it allows us to do something new
  - Like a `constexpr` constructor for a class containing an array

```
#if defined(Q_COMPILER_INITIALIZER_LISTS) && !defined(Q_QDOC)
    Q_DECL_CONSTEXPR QUuid() : data1(0), data2(0), data3(0), data4{0,0,0,0,0,0,0,0} {}

    Q_DECL_CONSTEXPR QUuid(uint l, ushort w1, ushort w2, uchar b1, uchar b2, uchar b3,
                           uchar b4, uchar b5, uchar b6, uchar b7, uchar b8)
        : data1(l), data2(w1), data3(w2), data4{b1, b2, b3, b4, b5, b6, b7, b8} {}
#else
```

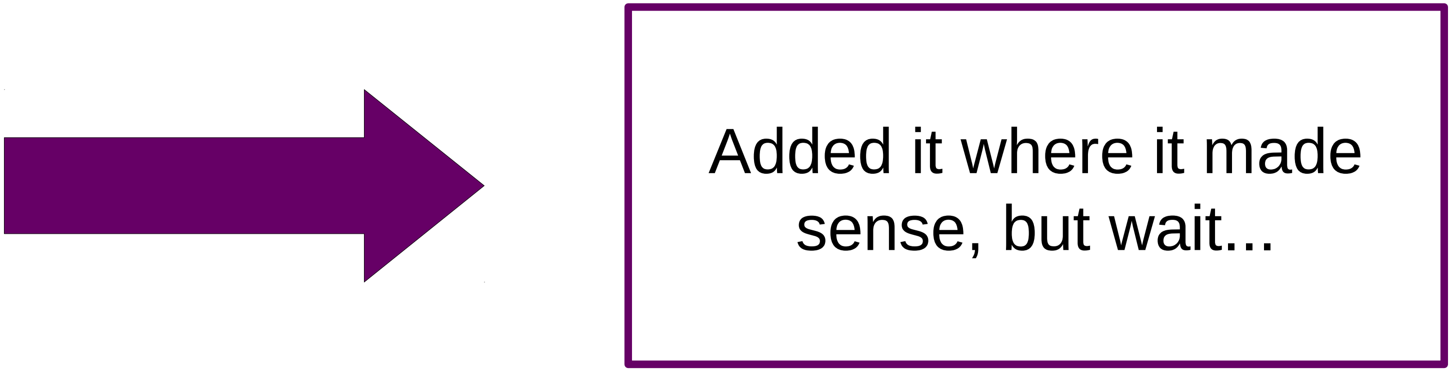
Lambdas (Q_COMPILER_LAMBDA)

- Support for use of lambdas added to:
  - `QObject::connect`
  - QtConcurrent (requires `decltype` and the new function syntax)
- Need to add to other slot-type functions
- No lambda use in Qt itself...

Feature is for the users, not for the library itself...

`noexcept` support (Q_COMPILER_NOEXCEPT)

- Improves code generation of callers!
- MSVC’s nothrow() has the semantic of noexcept
  - But not GCC’s! It implements the C++98 standard



Added it where it made sense, but wait...

Does C code throw?

- The C language has no support for exceptions...
- Unless you’re called Microsoft:
  - Windows has exceptions in C mode
  - In fact, crashes are thrown as exceptions!
- Unless you’re using Linux:
  - POSIX asynchronous cancellations are implemented with exceptions
  - Possible C++1y feature

noexcept macros

- Helper macros:

– Q_DECL_NOTHROW	noexcept if supported, nothrow() on MSVC, empty otherwise
– Q_DECL_NOEXCEPT	**really** noexcept if supported, empty otherwise
– Q_DECL_NOEXCEPT_EXPR(x)	for use in noexcept expressions

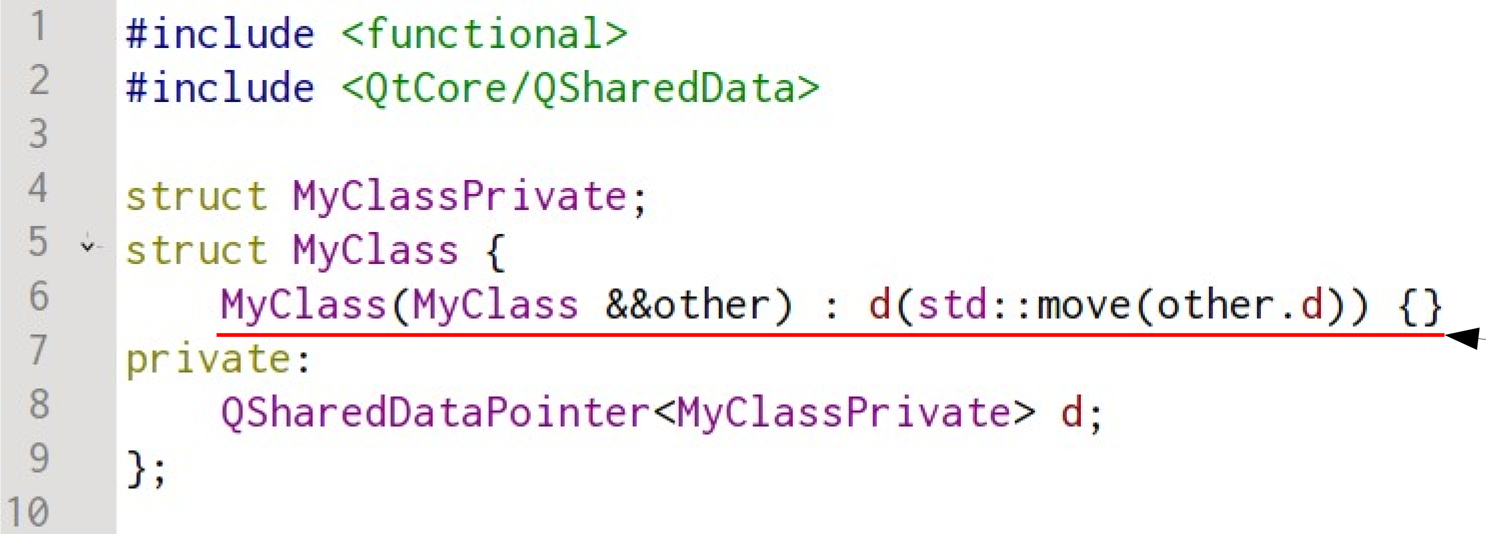
```
Q_CORE_EXPORT uint qHash(const QByteArray &key, uint seed = 0) Q_DECL_NOTHROW;
template<typename T> inline uint qHash(const T &t, uint seed)
    Q_DECL_NOEXCEPT_EXPR(noexcept(qHash(t)))
{ return (qHash(t) ^ seed); }
```

Move constructors (Q_COMPILER_RVALUE_REFS)

- Look deceptively easy
- Question: what state is a moved object left in?
- Can’t use them if using smart pointers and d-pointer / pimpl
  - Constructor needs to implement destruction for exceptional case

We can’t provide move constructors everywhere

Move constructor + smart d-pointer problem



```
 1  #include <functional>
 2  #include <QtCore/QSharedData>
 3
 4  struct MyClassPrivate;
 5  struct MyClass {
 6      MyClass(MyClass &&other) : d(std::move(other.d)) {}
 7  private:
 8      QSharedDataPointer<MyClassPrivate> d;
 9  };
10
```

```
qshareddata.h: In instantiation of
‘QSharedDataPointer<T>::~QSharedDataPointer() [with T = MyClassPrivate]’:
/tmp/test.cpp:6:52:   required from here
qshareddata.h:87:36: error: invalid use of incomplete type ‘struct
MyClassPrivate’
```

Move constructor: state of moved-from object

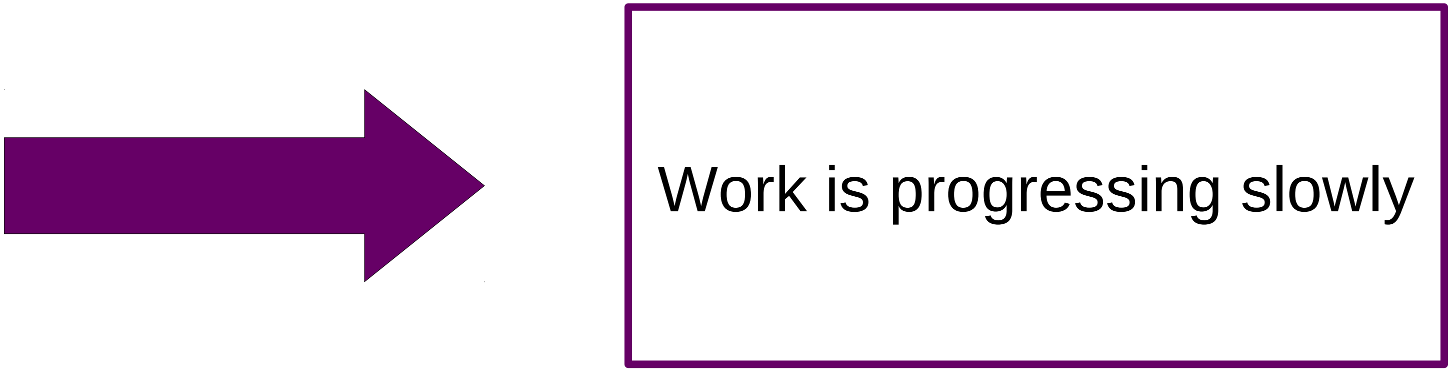
- What can you do with v?

```
MyClass v;



other = std::move(v);
// v?
```

- It must:
  - Be destructible
  - Be moved onto (swap implementation by triple-move)
  - What else?

Move semantics

- Would like to add support everywhere
- Huge amount of work
- Need to be careful about behaviour compatibility



Work is progressing slowly

Ref qualifiers in member functions (Q_COMPILER_REF_QUALIFIERS)

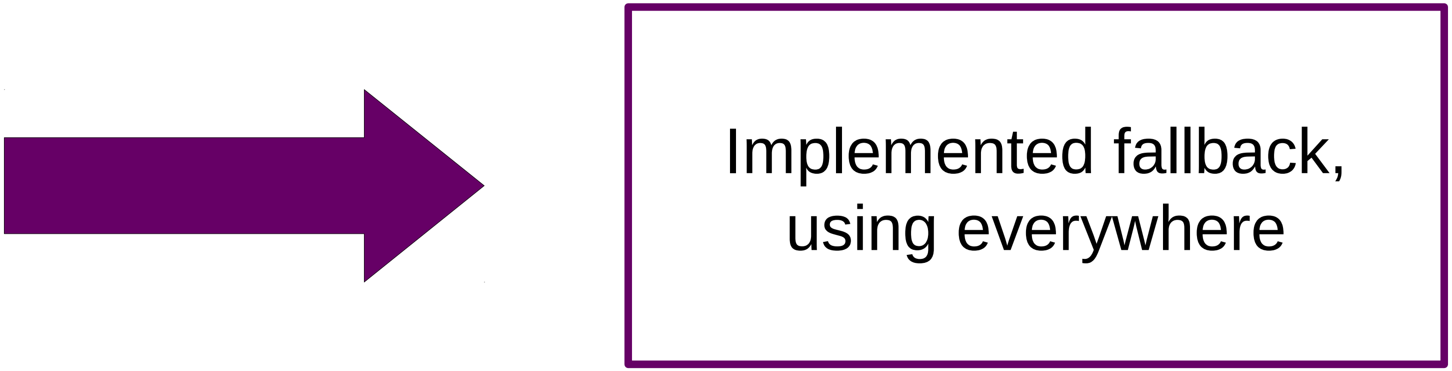
- Still investigating

```
QString time(int hrs, int mins)
{
    return QString("%1:%2").arg(hrs).arg(mins, 2, 10, QChar('0'));
}
```

  - Can we avoid the temporaries?
- Problems:
  - http://gcc.gnu.org/bugzilla/show_bug.cgi?id=57064 - FIXED in 4.8.2
  - Maintaining binary compatibility

Static assertions (Q_COMPILER_STATIC_ASSERT)

- Really, really useful
- Qt provides a fallback for C++98:
  - Check happens even in C++98
  - But misses error message



Implemented fallback, using everywhere

Thread-local storage (Q_COMPILER_THREAD_LOCAL)

- Some compilers provide support in C++98 (and C):

MSVC	`__declspec(thread)`
GCC, ICC, Clang	`__thread`

- Qt provides a fallback (QThreadStorage)

Investigate adding an unconditional macro

Thread-safe function statics (macro missing)

Q_GLOBAL_STATIC

- Two problems solved with one solution:
  - Thread-safety of function (local-scope) statics
  - Load-time overhead of global statics
- It uses a function (local-scope) static if it’s thread-safe
  - All compilers adhering to the IA-64 C++ ABI
- Otherwise, it uses a mutex and a guard variable

Q_GLOBAL_STATIC features

```
Q_GLOBAL_STATIC_WITH_ARGS(MyClass, cachedData, (42))
int data()
{
    if (!cachedData.exists()) {
        // don't create the static if it doesn't exist yet
        return 42;
    }
    return cachedData->i;
}

// function possibly called during application shutdown
int dangerousData()
{
    MyClass *c = cachedData;
    return c ? c->i : -1;    // c is null if it has been already destroyed
    // also:
    return cachedData.isDestroyed() ? -1 : cachedData->i;
}
```

Unicode strings (Q_COMPILER_UNICODE_STRINGS)

String literals

- Very useful and welcome
- But never used directly...

QStringLiteral

- Always available:
  - Better with lambdas and UTF-16 string literals
  - Otherwise, falls back to QString::fromUtf8
- Enforces that all source code **must** be encoded in UTF-8

QStringLiteral goals

- Returns a QString
- No memory allocation → internal data stored in .rodata

```
auto s = QStringLiteral("Hello");
```

expands to something like...

```
auto s = []() -> QString {
    enum { Size = sizeof(u"" "Hello") / 2 - 1 };
    static const QStaticStringData<Size> literal = {
        Q_STRINGDATA_HEADER(Size),
        u"" "Hello"
    };
    return const_cast<QArrayData *>(&literal.header);
}
```

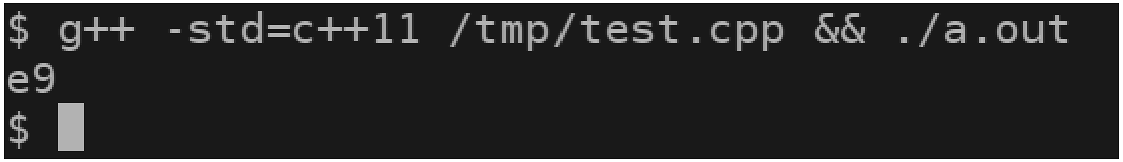
The standard committee stopped short...

- I wrote this on Linux:

```
u16string s = u"Résumé";
cout << hex << s.at(1) << endl;
```

How do I print the string?

- It printed:

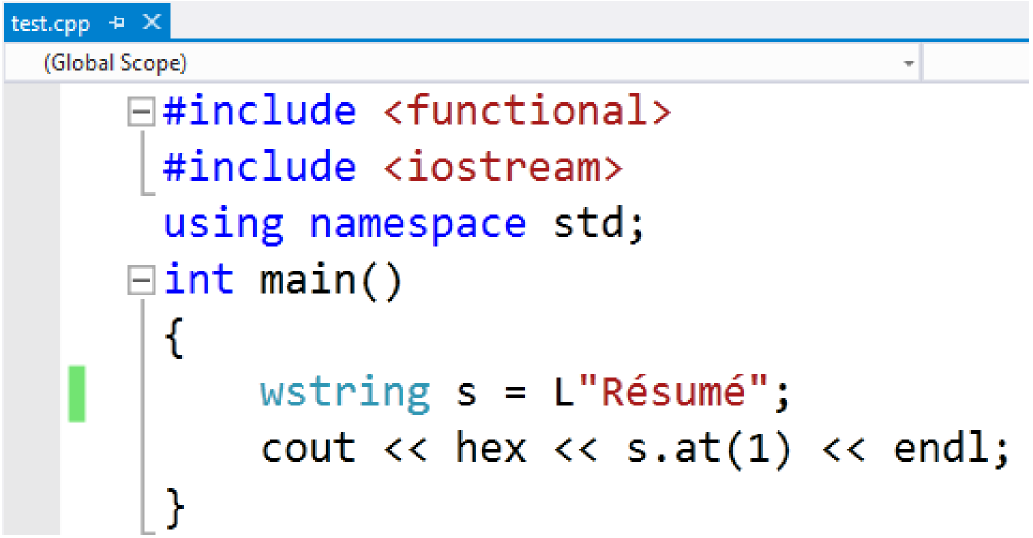


```
$ g++ -std=c++11 /tmp/test.cpp && ./a.out
e9
$ 
```

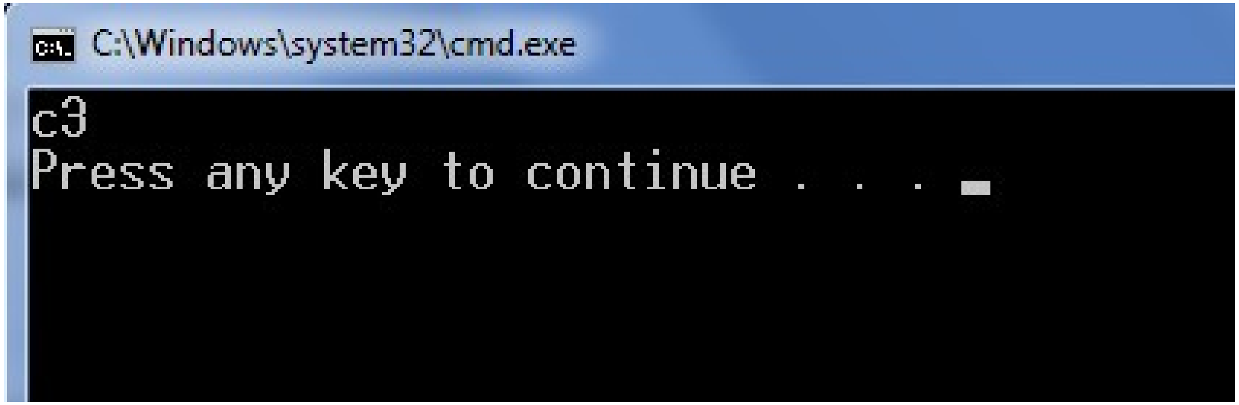
- If I **copy the file** to Windows and compile with MSVC[1], what will it print?

[1] once it supports Unicode strings

Let’s try...



```
#include <functional>
#include <iostream>
using namespace std;
int main()
{
    wstring s = L"Résumé";
    cout << hex << s.at(1) << endl;
}
```



```
c3
Press any key to continue . . .
```

User-defined literals (Q_COMPILER_UDL)

- Neat, but we haven’t found use in Qt yet
- Will be better in C++1y (see N3599)

```
template<char16_t... c> QString operator "" _q()
{
    static const QStaticStringData<sizeof...(c)> literal = {
        Q_STRINGDATA_HEADER(sizeof...(c)),
        { c... } // UTF-16 string
    }
    return &literal.header;
}
```

  - *“indeed [...] this form of literal operator has been requested more frequently than any of the forms which C++11 permits”* - N3599

N3599: http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2013/n3599.html

Latent bugs

- Some code is almost never compiled in C++11 mode
  - e.g., Windows code, due to MSVC and older GCC versions in MinGW
- Errors show up when the user upgrades (or downgrades!)

→ We need to keep an eye for bug reports

The future

C++1y auto function with no return type (Q_COMPILER_AUTO_RETURN_TYPE)

- Proposed by N3386
- Implemented in GCC 4.8 with -std=c++1y
  - No way to detect that flag
- Will most likely not use in Qt for a long time

Future directions

Finish what we started

- Move semantics
- Template export control
- Standard Library feature detection
- “Play” with compiler features

What we'd like to see in the language

- Complaints from previous slides
- Very little in terms of language
  - C++11 was very good
  - Probably things we don't know we need
- Concepts & more meta-programming
- Modules
- Reflection – get rid of moc

How about the Standard Library?

- We don’t use much of the library
- But we’ll keep an eye out and contribute experience
  - e.g., std::networking::uri (N3420, N3484, N3507, N3625)
  - Event loop
- Would like to see simplification of common use-cases
  - Converting int to std::string / std::u16string
  - Dealing with user’s locale codec

We really need more from compilers and OSes

- realloc_inplace (N3495)
- futex (Linux) or WaitOnAddress (Windows 8)
- Support for SIMD with intrinsics
- Support for targeting multiple processor architectures
- Tooling like valgrind, helgrind, perf
- Something between all-or-nothing debugging symbols
- Tighter control over binary compatibility

Conclusion

Conclusions (1/2)

What most developers want:

- Put old version into maintenance mode
- Require C++11 for newer versions

If you can't afford that:

- Target both C++98 and C++11 simultaneously

In any case:

- Familiarise yourself with the C++11 memory model

Conclusions (2/2)

When targeting C++11 & C++98:

- Determine which minimum compiler versions you require
- Focus on features that require no client code changes
- Hide differences in macros
- Try to resist NIH, re-use Qt or Boost config macros
- Try to keep BC between C++11 and C++98 builds

Questions?

Thiago Macieira
thiago.macieira@intel.com

Links:
Website: http://qt-project.org
Mailing lists: http://lists.qt-project.org
IRC: #qt and #qt-labs on Freenode