QML version ??

A retrospective after 5 years

Ulf Hermann, ~~November 19, 2019~~ September 5th, 2024

Goals in 2019

- Improve Performance and memory consumption
- Simplify Maintenance of large scale QML projects
- Address Language problems
- Facilitate better Tooling

Achieved by 2024

- Improve Performance and memory consumption
  - Performance: Some via compilation to C++
  - Memory consumption: No
- Simplify Maintenance of large scale QML projects: Some
  - via more rigid (CMake) project structure
- Address Language problems: No
- Facilitate better Tooling: Some – qmllint, qmlformat, qmlls

Performance and memory consumption (**2019**)

- QML requires a full JS engine and garbage collector
  - Not well suited for very low-end devices
  - GC leads to some unpredictable performance characteristics
- Object model built on top of Qt Object model
  - Duplicated data structures
  - Huge amount of mallocs
  - Large initialization overhead (runtime, not compile time)
- Binding propagation is immediate, not synced with consumers
  - Leads to duplicated binding evaluations
- Weak typing and runtime resolution of dependencies
  - Generated code must be generic

Performance and memory consumption (**2024**)

- QML requires a full JS engine and garbage collector: unchanged
  - Not well suited for very low-end devices
  - GC leads to some unpredictable performance characteristics
- Object model built on top of Qt Object model: unchanged
  - Duplicated data structures
  - Huge amount of mallocs
  - Large initialization overhead (runtime, not compile time)
- Binding propagation is immediate, not synced with consumers: invalid
  - But: delayed binding propagation does not push changes into the scene graph!
  - Leads to duplicated binding evaluations: unchanged
- Weak typing and runtime resolution of dependencies: improved
  - Generated code must be generic
  - Type annotations, automatic generation of complete qmltypes, focus on QML modules

Maintaining large scale QML projects (**2019**)

- Weak typing makes refactoring difficult
- "distributed" versioning (in each QML file)
- QML scoping rules can lead to unexpected side effects
- Integration with C++ type system can be cumbersome

Maintaining large scale QML projects (**2024**)

- Weak typing makes refactoring difficult: improved
  - qmlls can exhaustively analyze well-written qml
  - No actual refactoring tools, yet
- "distributed" versioning (in each QML file): improved
  - Versioning is optional and discouraged now
- QML scoping rules can lead to unexpected side effects: improved
  - qmllint, qmlls warn about unqualified access
  - The actual scoping rules are still a mess
- Integration with C++ type system can be cumbersome: improved
  - qt_add_qml_module, declarative type registration

Language problems (**2019**)

- QML Scoping rules difficult to understand
- No C++ API for bindings
- Binding updates done immediately, not batched
- Complicated/Unnecessary versioning system
- C++ based context properties
- Can’t bind to JavaScript properties
- No private properties
- Grouped properties fundamentally broken

Language problems (**2024**)

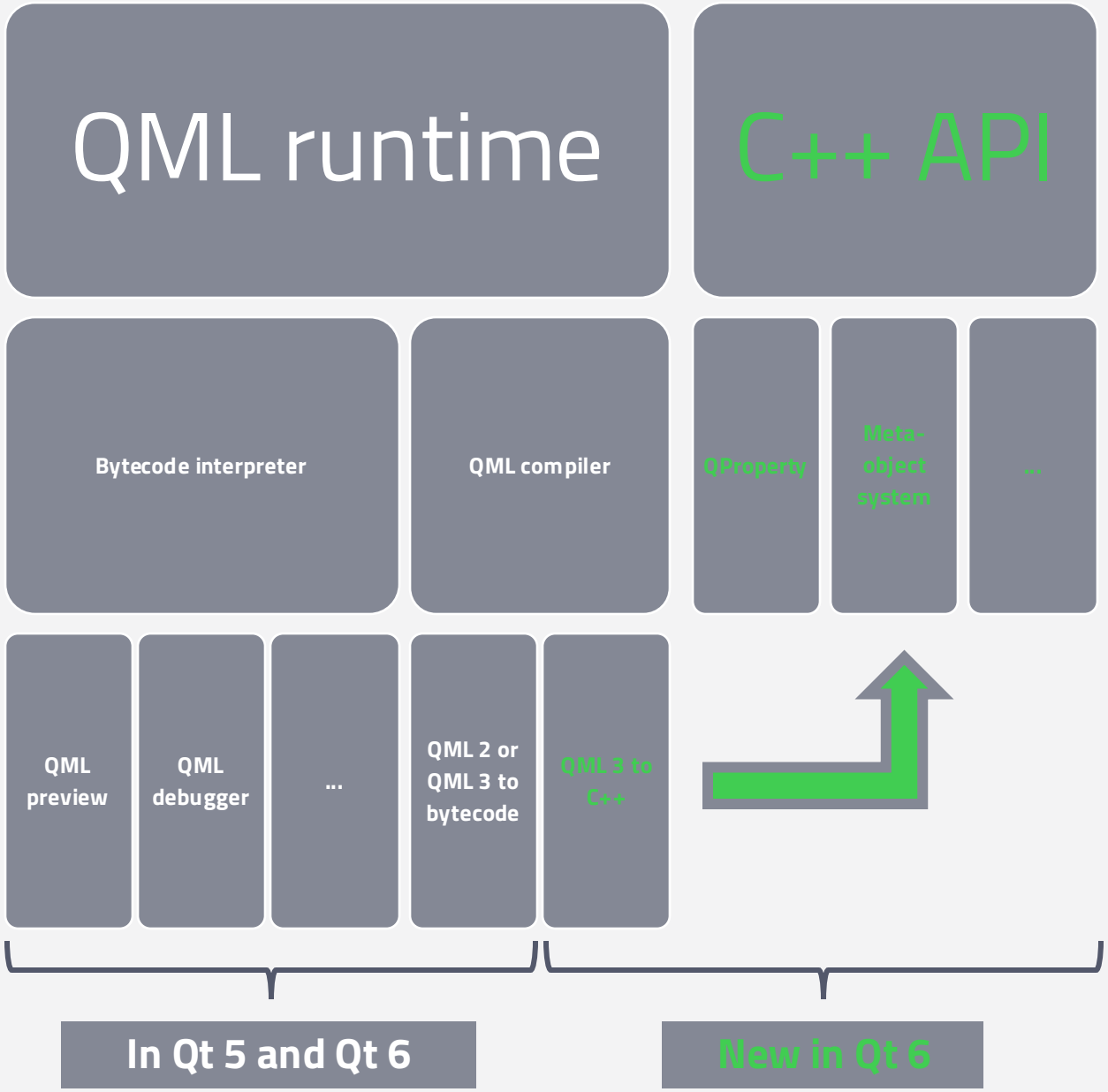
- QML Scoping rules difficult to understand: unchanged
- No C++ API for bindings: improved (but not well received)
- Binding updates done immediately, not batched: unchanged
  - May get fixed by exposing binding update groups to QML
- Complicated/Unnecessary versioning system: improved
- C++ based context properties: improved
  - Context properties are discouraged, but sometimes unavoidable
- Can’t bind to JavaScript properties: unchanged
- No private properties: unchanged
- Grouped properties fundamentally broken: invalid
  - Inconsistencies could be fixed, but only with behavior change

Tooling (**2019**)

- Refactoring tools difficult to implement
- Code completion buggy
- C++ types only visible to tooling when separately declared in .qmltypes
- Context properties and dynamic type registration invisible to tooling
- Grammar ported manually from Qt to Qt Creator

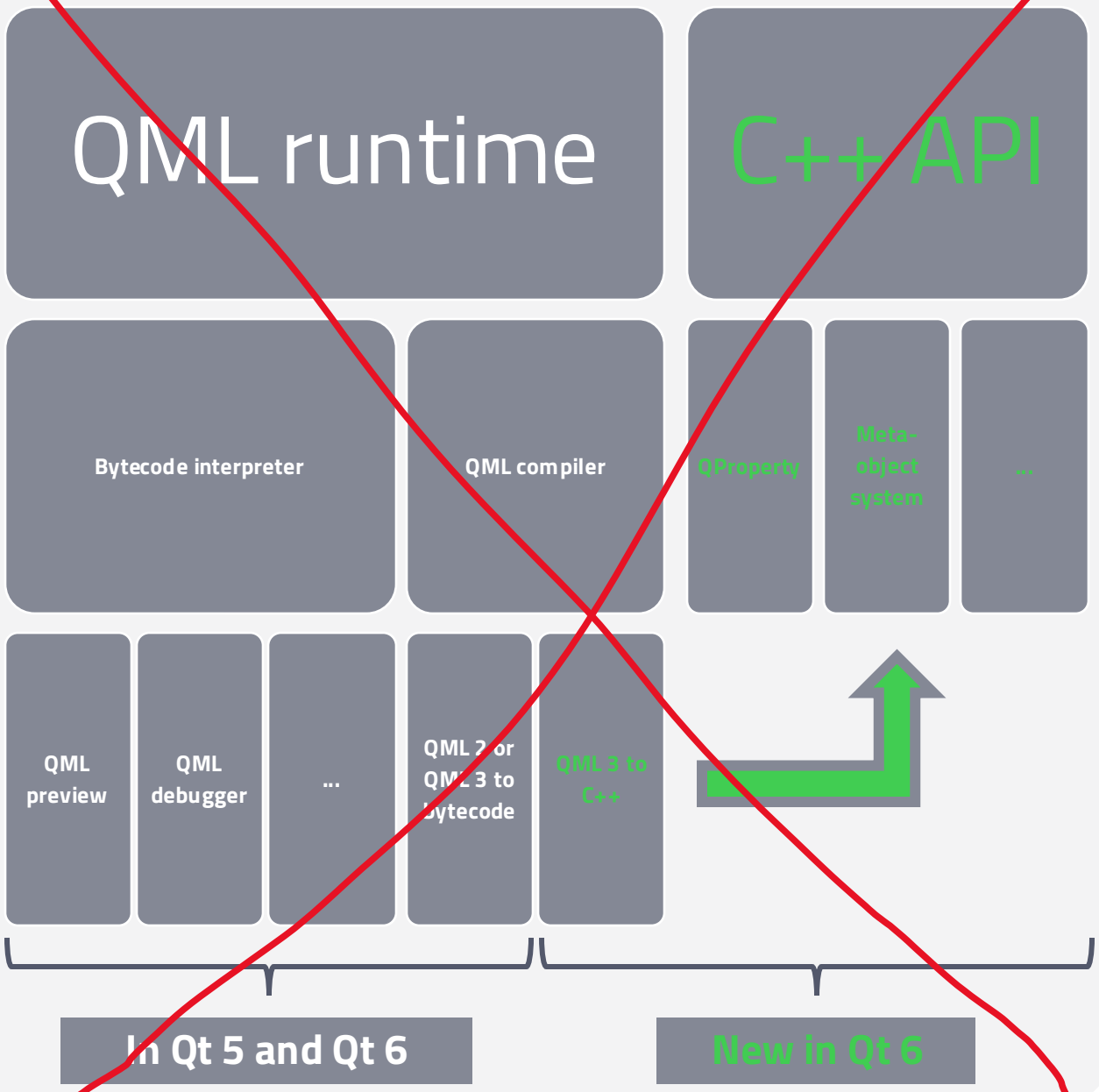
Tooling (**2024**)

- Refactoring tools difficult to implement: Improved
  - QmlCompiler, QmlDom
- Code completion buggy: Improved via qmlls
- C++ types only visible to tooling when separately declared in .qmltypes:
  - Improved: qmltypes auto-generated by qmltyperegistrar
- Context properties and dynamic type registration invisible to tooling
  - Improved: Both are discouraged
- Grammar ported manually from Qt to Qt Creator: unchanged



Qt 6 and QML 3 (2019)

- QML 3 will be a **subset** of QML 2
- Qt 6 will support **both**
  - QML 2
  - QML 3
- Qt 5 runs QML 3 as QML 2
- Qt 6 will get **extra features** for QML 3
  - compile to C++
  - lower memory footprint
  - no JavaScript interpretation / JIT compilation necessary
  - no garbage collector necessary



Qt 6 and QML 3 (2024)

- QML 3 will be a **subset** of QML 2
- Qt 6 will support **both**
  - QML 2
  - QML 3
- Qt 5 runs QML 3 as QML 2
- Qt 6 will get **extra features** for QML 3
  - compile to C++
  - lower memory footprint
  - no JavaScript interpretation / JIT compilation necessary
  - no garbage collector necessary

None of this has happened!

Further Qt 6 roadmap (**2019**)

- Make JavaScript / Garbage Collector optional
  - Trim down the scripting language, disallow closures, cyclic references
  - Avoid most heap allocations
  - Use reference counting instead of garbage collection for remaining memory management
- Move Property system to Qt Core
  - Properties evaluated on access, not on change
  - Avoid redundant evaluation of related properties
  - Add C++ API
- QML compiler library
  - Compile QML 3 to C++, QML 2 to bytecode
  - Improve tooling
  - Facilitate QML language server

Further Qt 6 roadmap (**2024**)

- Make JavaScript / Garbage Collector optional: unchanged
  - Trim down the scripting language, disallow closures, cyclic references
  - Avoid most heap allocations
  - Use reference counting instead of garbage collection for remaining memory management
- Move Property system to Qt Core:
  - Properties evaluated on access, not on change: invalid
  - Avoid redundant evaluation of related properties: unchanged
  - Add C++ API: done but not well received
- QML compiler library
  - Compile QML 3 to C++, QML 2 to bytecode: improved (qmlcachegen)
  - Improve tooling: done
  - Facilitate QML language server: done

Architecture (**2019**)

- Object model / QProperty
  - Qt Core
- Item models
  - Own plugin (done)
- Debugging/profiling framework
  - Partly Qt Core
- Compiler and runtime for dynamic QML
  - QtQml, **not** needed at runtime for QML3
- Code Model
  - Own library (links against QtQml)
- QtQuick must not depend on QtQml anymore!
  - Split out further things, like Canvas
  - Avoid JavaScript-y constructs in C++

Architecture (**2024**)

- Object model / QProperty
  - Qt Core: done but hardly used!
- Item models
  - Own plugin (done)
- Debugging/profiling framework
  - Partly Qt Core: unchanged
- Compiler and runtime for dynamic QML
  - QtQml, **not** needed at runtime for QML3: unchanged
- Code Model
  - Own library (links against QtQml): done (does not link against QtQml, though)
- QtQuick must not depend on QtQml anymore! unchanged!
  - Split out further things, like Canvas
  - Avoid JavaScript-y constructs in C++

Summary

Done

- Better **module** and **type** system
  - qt_add_qml_module
  - declarative type registration
  - versioning optional
- Partial **compilation** to C++
  - Only simple bindings and functions so far
- QML language server
  - And qmllint, qmlformat

Unchanged

- QML is built on **QObject**
- QML uses (GC'ed) **JavaScript**
- **Redundant** evaluation of properties
- Incomprehensible scoping (and lookup) rules
- No private properties
- Lack of **Modularization**
  - Detaching QtQuick from QtQml
  - Splitting QtQml into smaller pieces

Qt

Fundamental Problems

QML is built on **QObject**

- Bad data **locality**
- Memory **overhead**
- Memory **fragmentation**
- Causes **value/object** type duality
- **Single inheritance** requires interfaces, "extended" types

QML uses **JavaScript**

- Requires **garbage collector**
- Introduces **weak, insane, third (!)** type system
- Is **hard** to analyze/compile ahead of time
- Requires value type instances to be JavaScript objects!