KDAB

Christoph Sterz

QML-Driven HMI Architectures for Rolling Embedded Devices

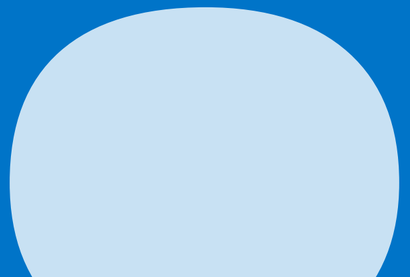
KDAB

1. What's the Difference?
   How Do We Fail Differently?
2. First Know the Rules …
   … then Break Them!
3. Turnkey Programming Setups
   & Fast Turnaround to Win the Game

KDAB

The Qt, OpenGL and C++ experts

So what's the difference with automotive devices?



KDAB

Embedded != Automotive

The Qt, OpenGL and C++ experts

KDAB

Embedded != Automotive

- ⊕ Platform Building
- ⊕ Cross Compilation
- ⊕ Hardware Limits
- ? Custom Hardware
- ? Fast Startup Times
- ? Phone Integration
- ? Multimedia
- ? RT & Certification

KDAB

Embedded != Automotive

- ⊕ Platform Building
- ⊕ Cross Compilation
- ⊕ Hardware Limits
- ? Custom Hardware
- ? Fast Startup Times
- ? Phone Integration
- ? Multimedia
- ? RT & Certification

→ All ?s become ⊕s!

- ⊕ External Applications
- ⊕ Compositing
- ⊕ Hands-Free Interaction
- ⊕ Changing Environment

---

significantly more screens
significantly more settings
significantly larger team
significantly more parties

KDAB

Be aware of the increased size and complexity!

Projects like these have two new major problems:

Cross-Cutting Concerns

- Styling and customization
- Hardware variant modelling
- User & role management
- Searching

Interaction Between Parts

- Notifications and warnings
- Compositing/overlays
- Shared resources
- Data transfer

KDAB

Be aware of the increased size and complexity!

Projects like these have two new major problems:

Cross-Cutting Concerns

- Styling and customization
- Hardware variant modelling
- User & role management
- Searching

Interaction Between Parts

- Notifications and warnings
- Compositing/overlays
- Shared resources
- Data transfer

Sad Fun Fact

These problems are most often revealed at the end of the project

These are all contracting forces and signs for a possibly monolithic architecture

A Word on Styling

Think twice if you want to include styling into your framework!

- Only **temporal changes** should be **done in QML** *... but that's not styling*
- **The rest** should be **done from the C++ side:**
  - Image providers
  - Constants pushed from C++ side: colors | sizes | margins | components
  - Retrigger all relevant bindings! Anchor changes are still a tricky issue

- Do not add styling by creating subclasses of a *"god-styled component"*
  - Rather re*load or instantiate* a different component

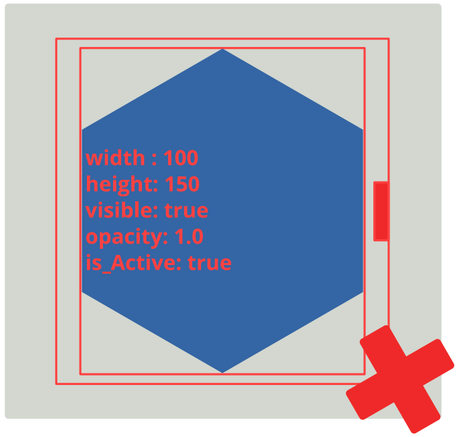
Think again if you want to include styling into your framework!

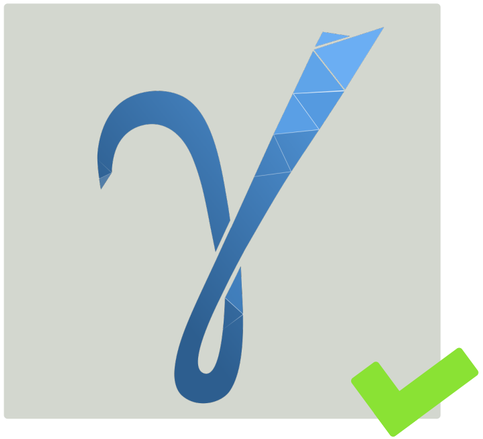
Don’t create an “engineering style”

Do not factor in engineering styles into your QML components framework!

- Very tempting to add visual debug information to all components
  - Often seen for layouting components, general item wrappers, grids
  - Set to invisible, activated via keypress

- Semi-good workaround: loader for engineering information
- Solution: Just use GammaRay, please!!! ;)

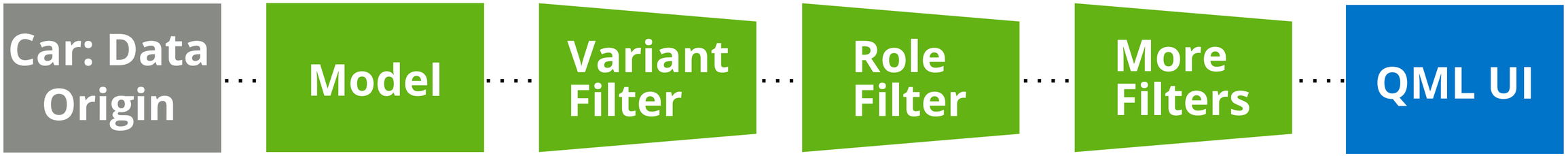




Filtering Functionality

Most of the customizations turn out to be constructible through filters

- Differing user roles and functionalities
  - Think of all stakeholders: manufacturers | retailers | technicians
  - Taxi lights, police radio integration, emergency sirens, …
- Same is true for different car model configurations and software variants
- Use Repeaters, Loaders, ListViews, PathViews, and Instantiators



KDAB

Hardware Limitations

*Expect your hardware to be **slow** … but expect the unexpected!*

Key pain points

- HMI startup
- Screen/view changes
- Background processes spinning up
- Spreading performance mistakes widely – “Death by 1000 Cuts”
  - Classic example: QGraphicalEffects Shadow on all texts

There is no inherent algorithmic complexity in a Car HMI, not even in nav!
What kills you in the end is the footprint of memory, components, screens

KDAB

Follow the path of memory

Trace the way of your visual assets at least one time from file to texture!

Designer

File

Board Storage

Memory

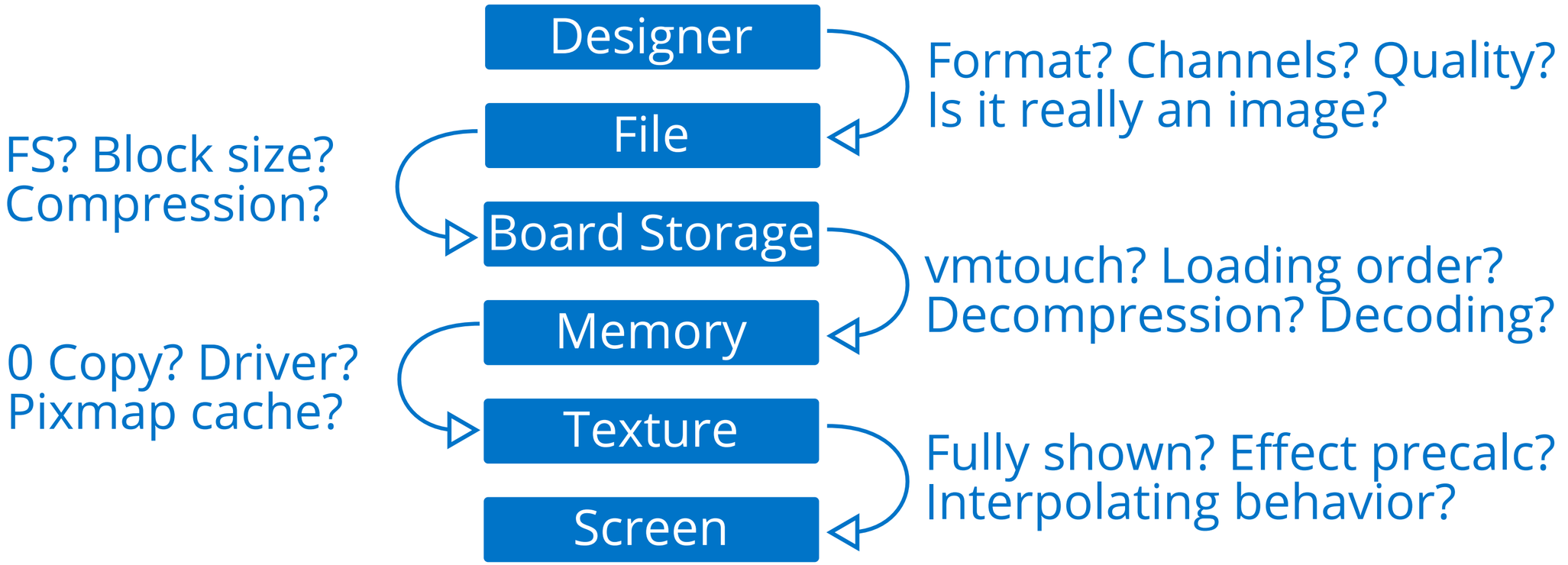
Texture

Screen

KDAB

Follow the path of memory

Trace the way of your visual assets at least one time from file to texture!



KDAB

HMIs have multiple applications

You are probably not going to implement that Japanese navigation by yourself …

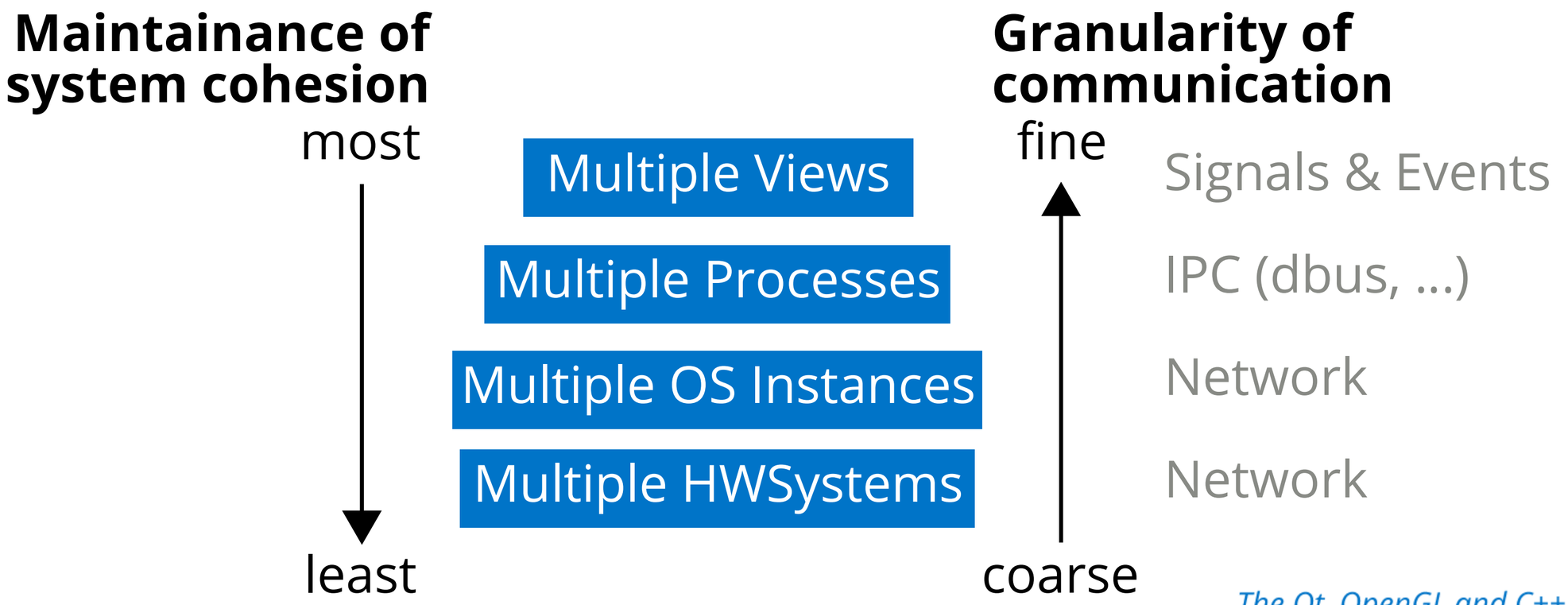
- Best Scenario: Suppliers/other teams also use Qt
- Otherwise: There are different options to compose applications
  - X11
  - Wayland
  - Streaming textures with GPU extensions
  - Weston IVI

If you are on Linux, best use Wayland!

Qt supports you with multi-screen systems

main touch screen | instrument cluster | back seat entertainment

Enabling global animations and events among these screens is hard



KDAB

Car HMIs are multi-input-method applications

- **Touch interaction** is the first-class citizen in modern cars
  - Complex gestures
  - Does not allow for eyes-free interaction

- **Physical input methods** are spread over the interior of a vehicle
  - More extreme in modern utility vehicles (cockpit situation)

- **Voice input** commands invoke UI changes
- In QML, only focus and view stacking support external interaction

→ *Input event handling must enable state changes in the overall UI from the backend! (C++ side)*

KDAB

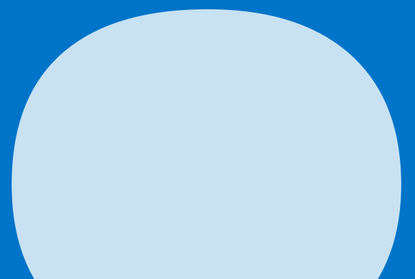
Auto & Embedded: You own and know the system!

- **The backend is formable** to serve the HMI
  - Event rates, types, and notifications
  - There can be intercommunication for deferred services

- **Slim out your OS's**/service's startup sequence
- It is possible to set and alter process priorities
- Boost foreground priorities?

- **systemd** becoming the standard
  - Manage dependencies of services
  - Set group rights and quotas

First Know The Rules ...
... Then Break Them!



KDAB

But first know the rules!

- **Structure your C++ around the data from your backend**
  - Generate and filter
  - Expose models and constants to QML
  - Logical transitions happen here

- **Structure your QML visually**
  - Think in terms of screens and pixels
  - Do not think in terms of functionality here
  - Visually temporal transitions happen here

Reinvent the wheel after you have tried Qt's wheel. And Qt offers many wheels ;)

KDAB

Don’t create what you won’t show!

Everything that takes less than 16 ms to create should be put behind a loader!

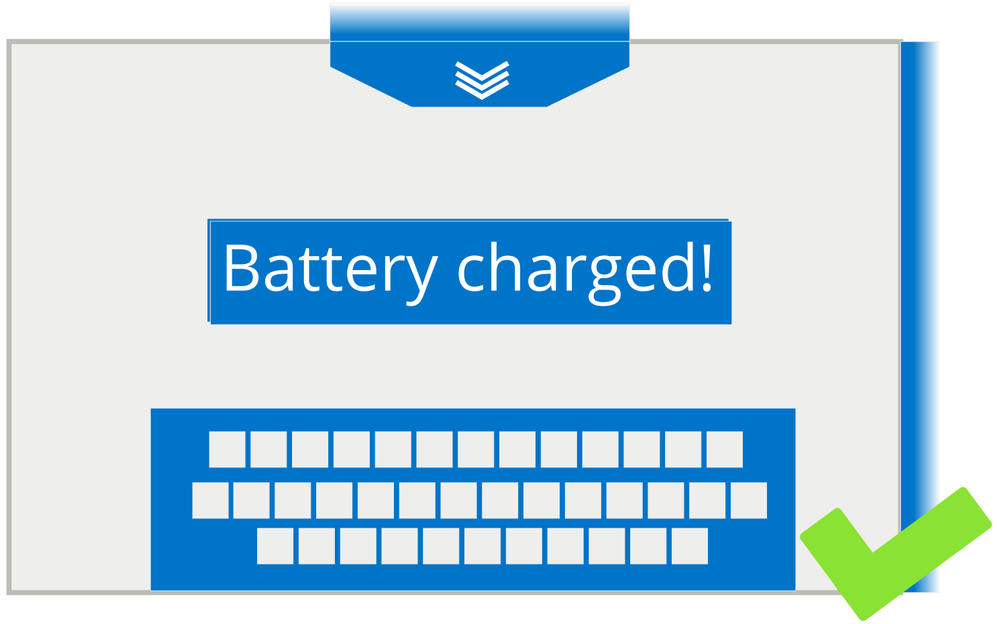
- With **precompiled qmlc,** it becomes tempting to load every screen upfront
- Reduces your times by **about 50 %**, *but it’s still not free!*

- Preloading your screens may trigger C++ code
- QML item creation bears hidden costs for
    - image decompression and texture upload
    - shader configuration and compilation

- Critical startup phase becomes less deterministic, less analyzable

Break the Rule: What to still load upfront

- **Notifications and warnings**
  - Might be on top with negative *z*-ordering
  - Invisible elements existing at all times
- **Slide-Ins and Overlays**
- **One singular virtual keyboard**
- **The “next” screen**
  - In wizard-like situations



KDAB

Take vs. Make

- **QMLShaderEffects are often slow**
  - custom OpenGL-based QQuickItems
    - Don’t cascade ShaderEffects

- Investigate if ParticleEffects can be replaced

- **Trade-off between prerendered images and effects**
- Sometimes even complex effects are faster than multiple images
- Prebaked image sequences can decrease the memory bandwidth

- QML introduced Shapes in 5.10. This can help reduce images further

KDAB

Stable vs. New

- Automotive usually settles on a fixed Qt version
  - Long Term Support | Stable version | Well tested
  - But most projects take more than a year from first plan to SOP!

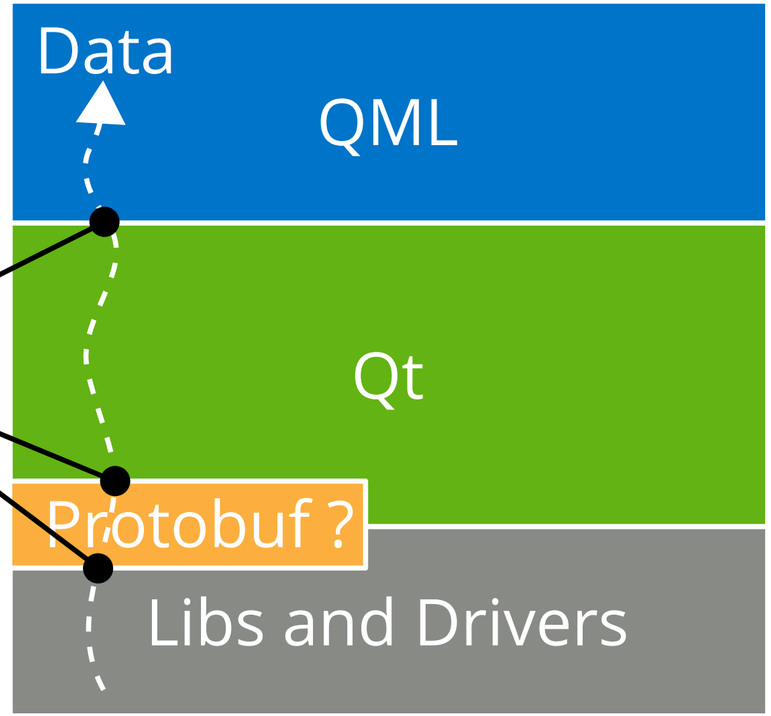
- If there is the option to switch to a new LTS, switch!
  - Improved performance
  - Reduce workarounds and pain points
  - Fairly backwards-compatible: easily portable within major version

There are successful embedded projects with open source GPU drivers!

Not everything will be Qt

But Qt IVI helps you cope with that

Checks
Type conversions
Polling



QT IVI

- “Rich” types
  - Value ranges
  - Defaults
- Eases mocking
- Designed for QML use

→ *visit Mike Krus’s Talk!*

Turnkey Programming Setups & Fast Turnaround to Win the Game

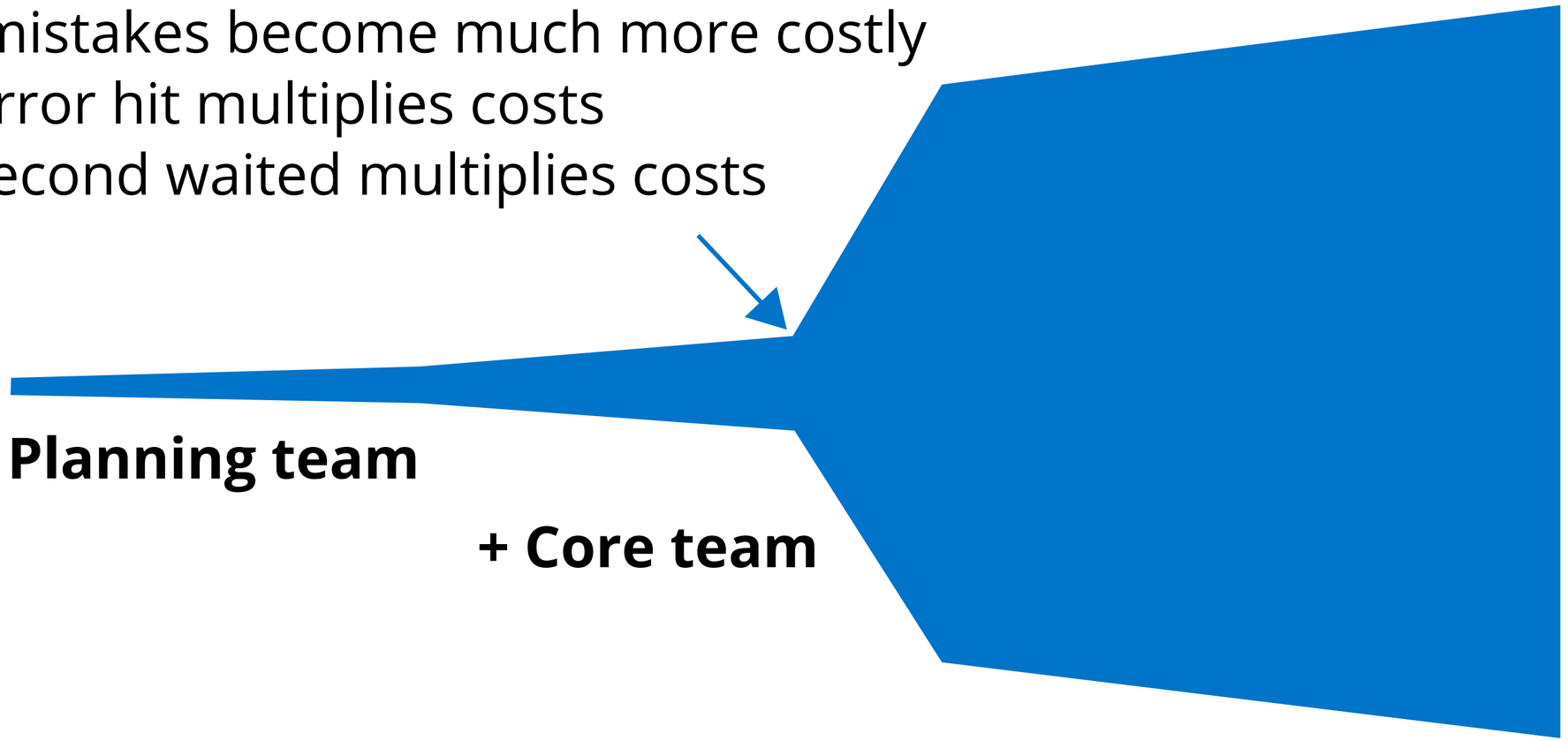
KDAB

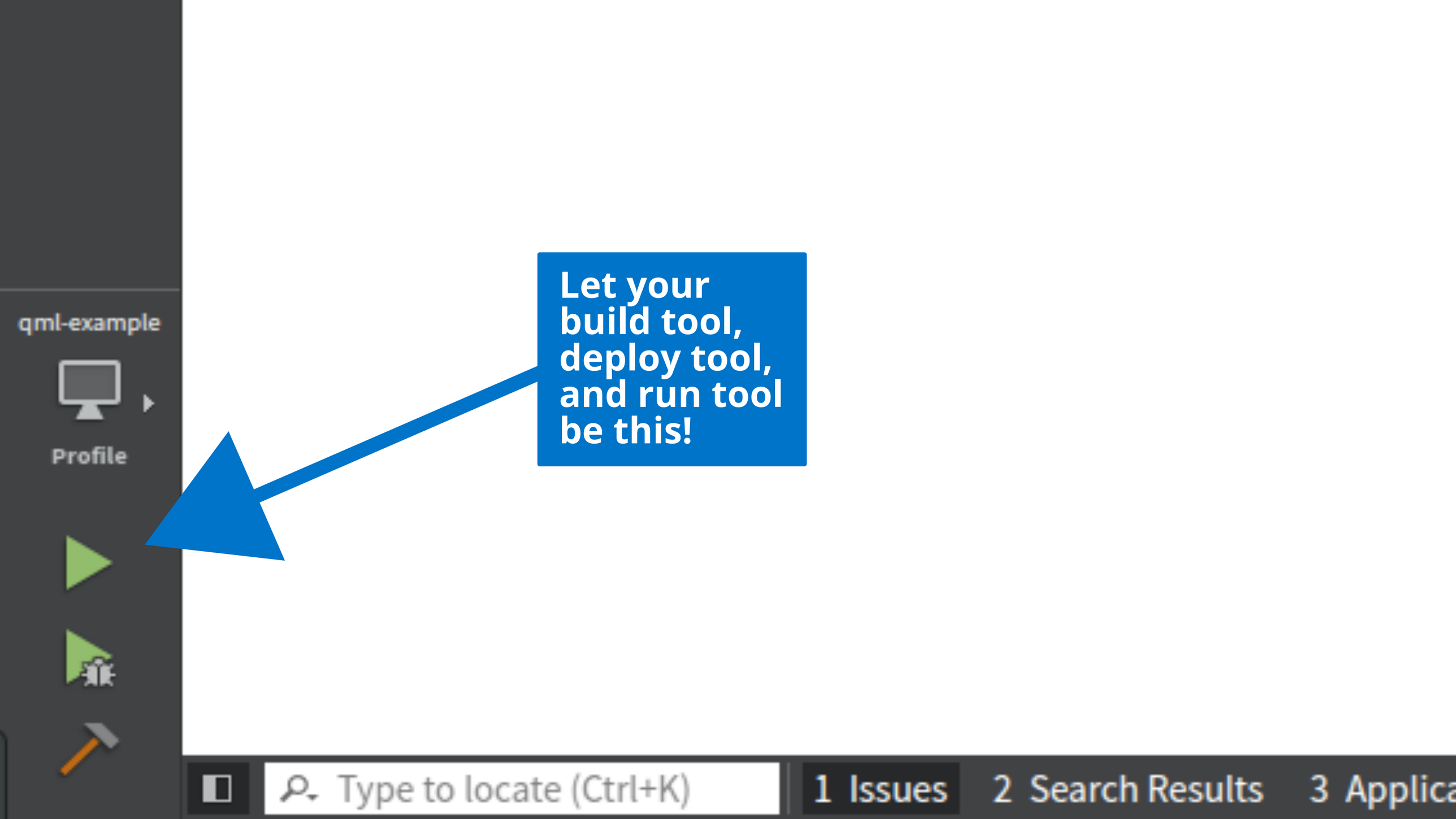
The Qt, OpenGL and C++ experts

KDAB

Have your tools ready at the “fanout”

- Here, all mistakes become much more costly
  - Every error hit multiplies costs
  - Every second waited multiplies costs




qml-example
Profile
Let your build tool, deploy tool, and run tool be this!
Type to locate (Ctrl+K)
1 Issues
2 Search Results
3 Applica

KDAB

A typical 2017 automotive SDK & setup

Usually contains this

C++/Qt

(custom) Qt
QtCreator
Static checkers
Valgrind
Emulator
GammaRay

↓

Visit Volker Krause's Talk!

Linux

(Yocto-built)
Dev version
Trace version
Product version

perf + hotspot

VCS/CI

Git (∞ submods)
Squish CI
Regular CI

No hardware available (yet)? No Problem!

With QtAutomotiveSuite's emulator!

- Most drastic hardware limitation is to have no hardware
- Qt is your friend for cross-platform development
  - Develop on desktop, deploy to target
- Supports emulating the target even with simulated hardware knobs

- Tooling Like QtCreator's DebugMode and GammaRay work from desktop!
  - In principle: Connecting to shared or remote HW prototypes is possible

- QtAutomotiveSuite comes with 20 ready-to-run images for common boards

KDAB

Have fast deployment cycles

Every second waited here is worth eliminating

- When finding a complex bug, deployment will be done hundreds of times
  - Differential updates
  - Instant navigation to the problems screen
  - Have a backend-enabled boilerplate application
  - Evaluate reloading components live or try out QmlLive
- Delays hinder willingness to try something new
- This is also true for device flashing
  - Find a fast and reliable mechanism to flash hardware images

KDAB

Make quick experiments possible to everyone

- Enable all devs to quickly request or create environments to test upon
- Have a **fixed naming scheme** for these variants:

Linux 4.13, Backend version 12, Qt 5.9 + Patches, HMI nightly

- Result is a hardware/emulator image!
- With Yocto, this can even be automated!

- Allows for pretests for upcoming backend versions

Getting/Making a new image should take 1 day max.

Don’t Invent Your Own Build System!

Use something widely used and cross-platform!

The Qt, OpenGL and C++ experts

KDAB

1. What's the Difference?
   How Do We Fail Differently?
2. First Know the Rules …
   … then Break Them!
3. Turnkey Programming Setups
   & Fast Turnaround to Win the Game

KDAB

Thank you!

christoph.sterz@kdab.com

The Qt, OpenGL and C++ experts