Functional Safety with Qt and Qt Safe Renderer

Kimmo Ollila

Presentation Outline

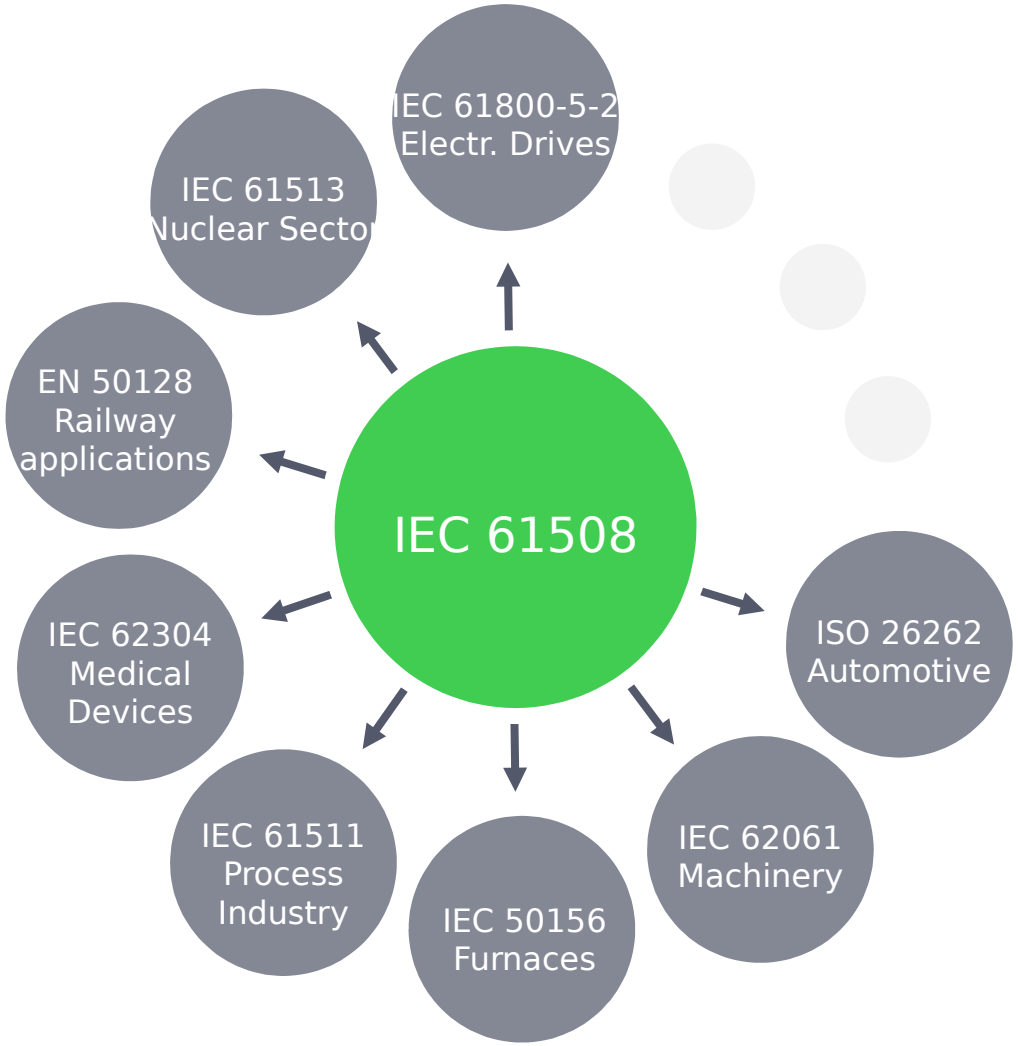
- What is Functional Safety?
- Safety Standards
- Creating a Certified System with Qt
- Qt Safe Renderer
- Summary

What is Functional Safety?

- Protect people from getting hurt
- Active systems: detect & prevent
- Determined considering the system as a whole
- Measured by Safety Integrity Level (SIL, ASIL in Automotive)
- Required level determined based on likelihood of injury or death

Examples of Safety Standards

- Main standard of functional safety is IEC 61508
- Examples of industry specific standards
  - Automotive: ISO 26262
  - Medical Device Software: IEC 62304
  - Railway software: EN 50128
  - Avionics software: DO-178B
  - Machine control: IEC 62061
  - Agricultural machines: ISO 25119
  - Nuclear: IEC 61513



Creating a Certified System with Qt

Creating a Certified System with Qt

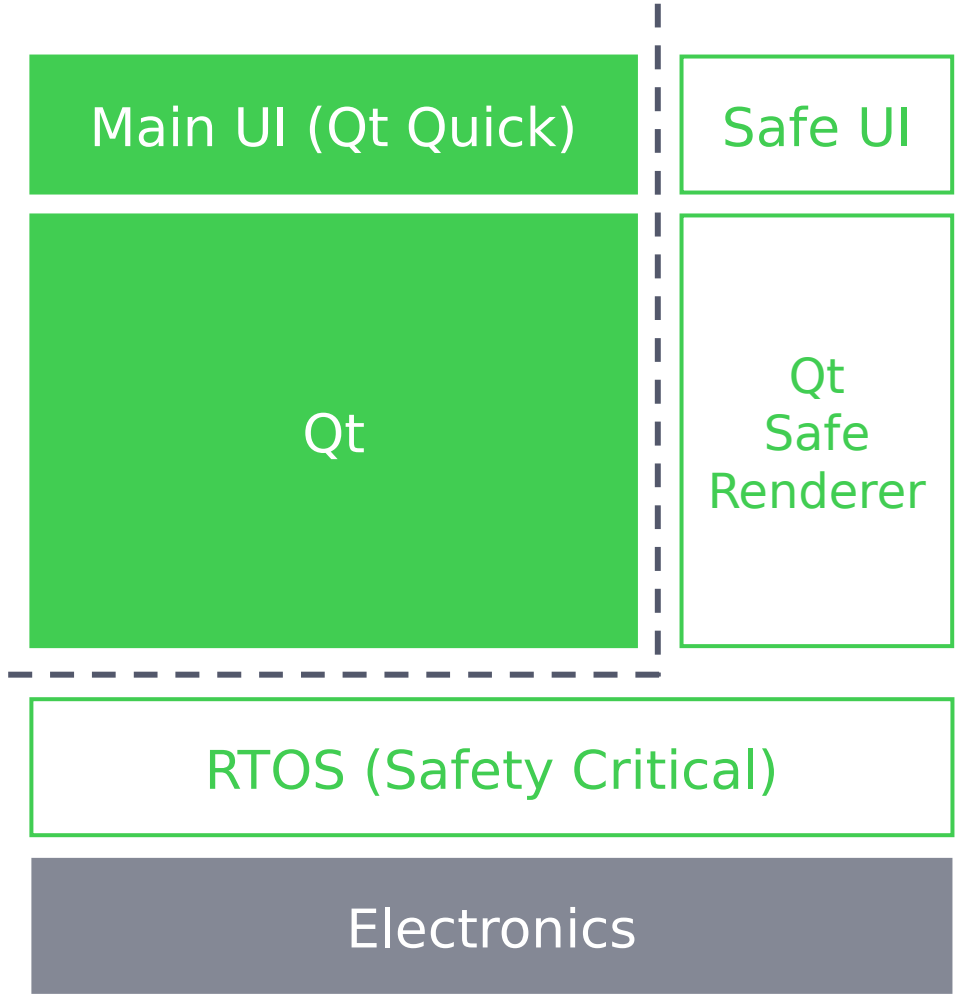
- Complete system needs to be certified
- Using pre-certified tools and components helps achieve certification for new system
- Separation of safety critical functionality from other functionality

With separation, Qt can be used in a system requiring certification without changing the Qt libraries.
Safety critical UI rendered with Qt Safe Renderer.

- Suitable means and level of separation depending upon the required SIL/ASIL level
- Qt Safe Renderer as safety critical process, Main UI with Qt Quick as regular process

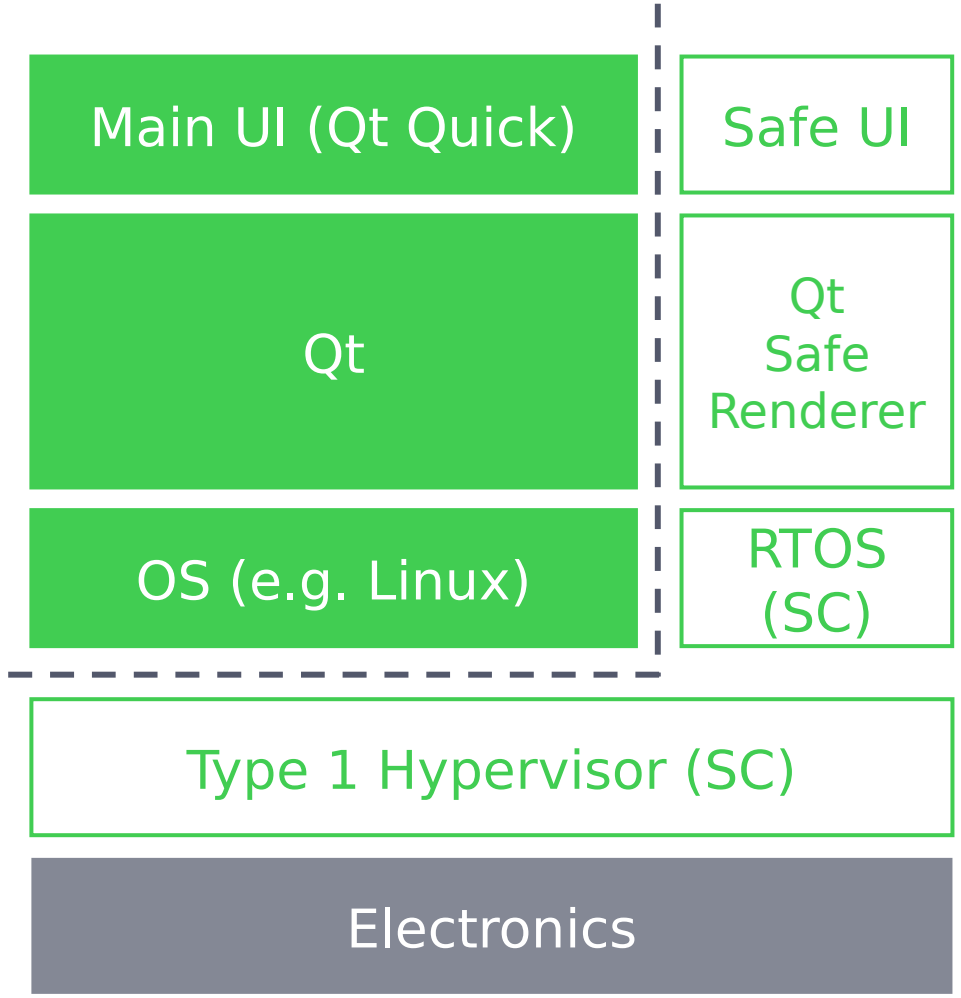
Certified RTOS for Separation

- Real-Time Operating System (RTOS) to separate Safety Critical and other processes
- Certified RTOS and toolchain saves time and effort in system level certification
- Certification requirements applied for the safety critical parts
- UI elements can be separated using HW layers or by the RTOS compositor
- In some designs, a safety critical UI may not be necessary at all, or can be arranged using a separate display or a warning light



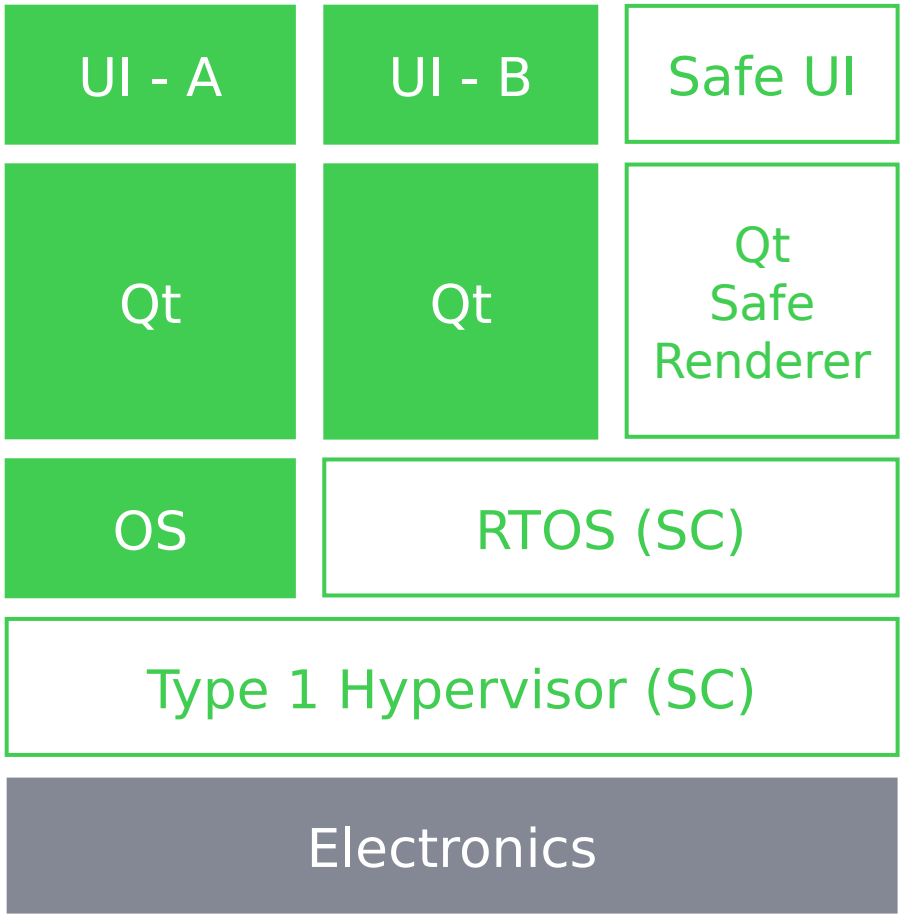
Hypervisor for Separation

- A Hypervisor to run separate OS for safety critical and other functionality
- Safety critical functionality on a certified RTOS
- Other functionality for example on embedded Linux
- Operating systems can share resources and data as long as the separation guarantees integrity of the safety critical software
- Safety critical functionality can be assigned to a dedicated CPU core
- Shared resources controlled by hypervisor (e.g. GPU)



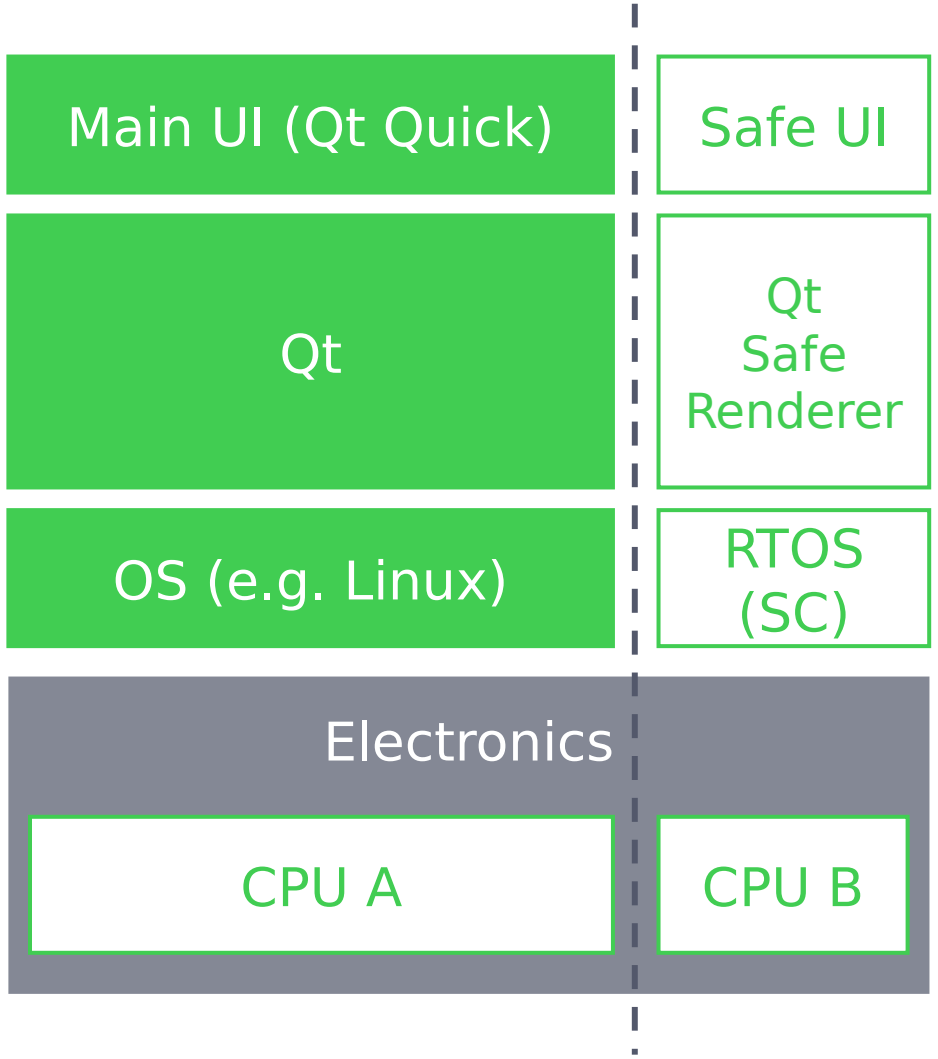
Hypervisor for Separation – with Multiple Domains

- A Hypervisor to separate OS domains, as well as safety critical and other functionality
- Safety critical functionality on a certified RTOS
- Other functionality for example on embedded Linux
- Two different regular Qt UIs + Safety Critical UI
  - UI - A running on a regular OS, e.g. Linux
  - UI - B running on a safety critical RTOS
  - Qt Safe Renderer for safety critical UI functionality
- Otherwise similar as previously shown hypervisor architecture



Separate Processors

- Separate processors or a single SoC with separate CPUs to run different OS for safety critical and other functionality
- Similar to hypervisor, but separation done directly with physical hardware
- Safety critical functionality can run on a simple RTOS
- A microcontroller CPU may be enough for safety critical functionality
- Other functionality can run for example on Linux OS
- Operating systems can share resources and data as long as the separation guarantees integrity of the safety critical software

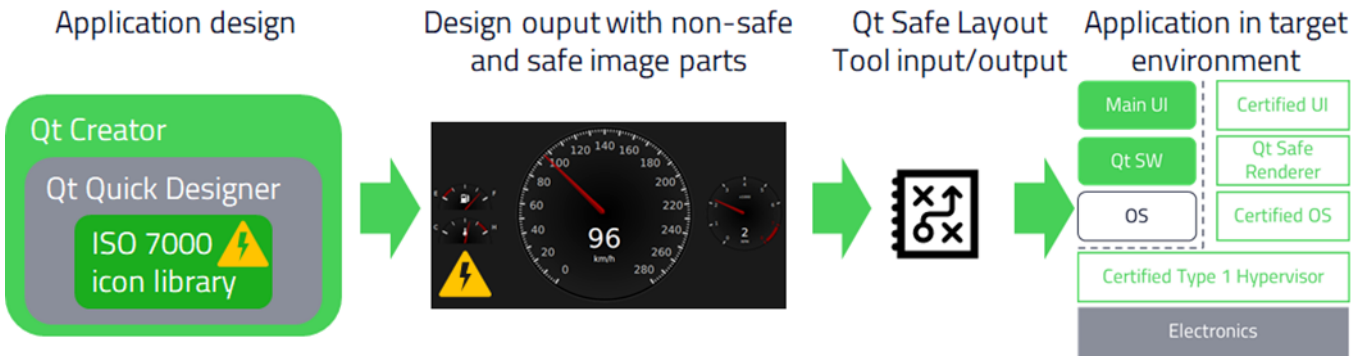


Qt Safe Renderer

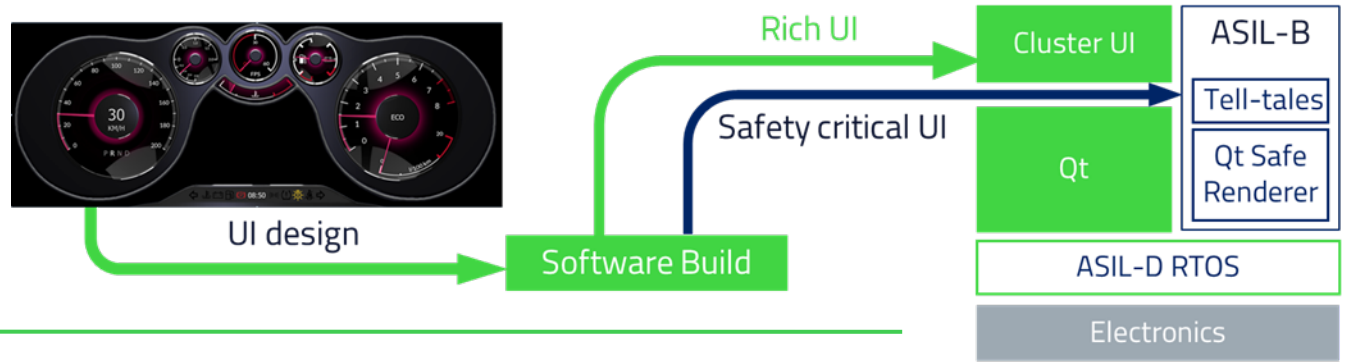
- Certification for: IEC 61508, ISO 26262, EN 50128 and ISO 62304

Qt Safe Renderer – Product Overview

1. Qt Safe Renderer SW consists of
   - Qt Creator plug-in
   - Layout generator
   - Runtime component



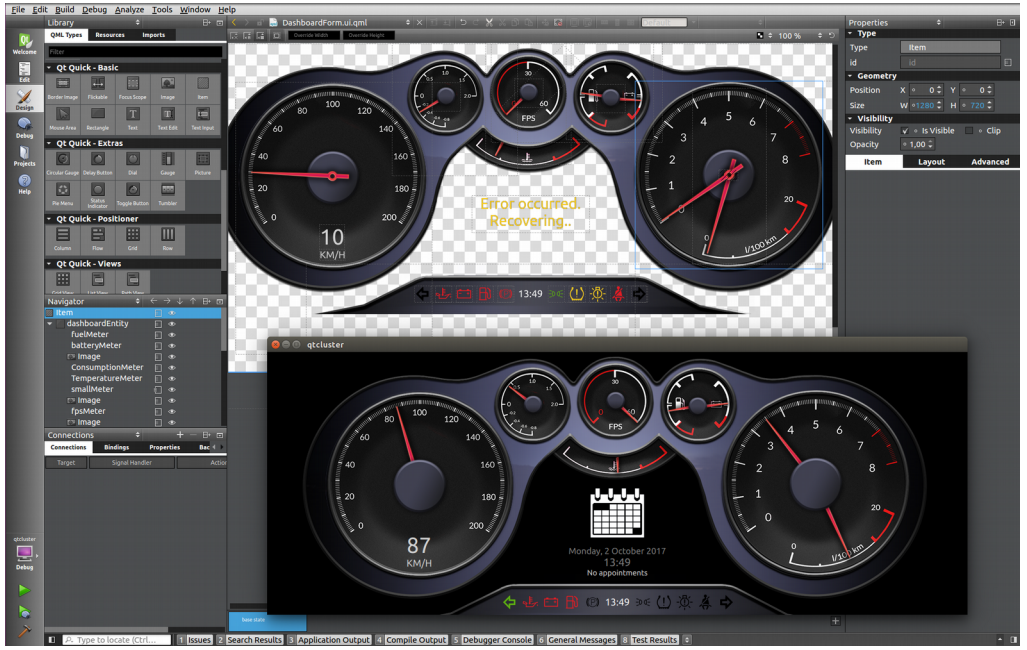
2. Common tool chain for designing both safe and non-safe UI



3. Safety Manual and Certification Artifacts

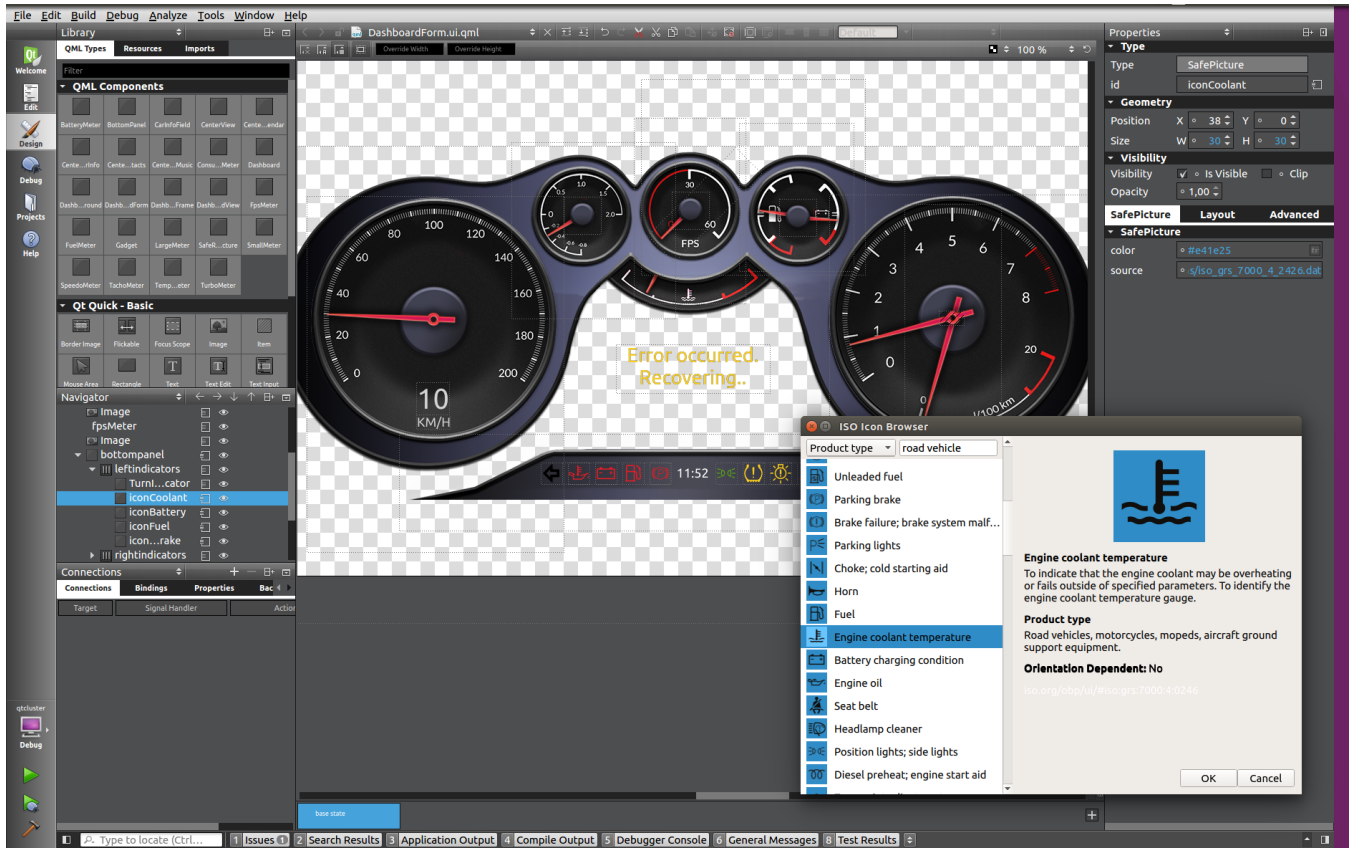
Qt Safe Renderer – Overview

- Two certified components:
  - Development tooling with visual designer
  - Rendering component for safety critical UI
- Integration to RTOS:
  - QNX 7.0 or later
  - INTEGRITY 11.4.4 or later
- Examples of supported HW:
  - NXP i.MX6, Renesas R-Car H3, Qualcomm Snapdragon 820, NVIDIA Tegra X1, ...



Qt Safe Renderer Tooling – Convenience for Safety Critical UI Creation

- Easy to define safety critical parts with Qt UI design tools
- Flexibility in UI design without need to modify safety critical SW components
- Integration to Qt Quick Designer visual UI creation tooling in Qt Creator IDE
  - Drag and drop safety critical items to UI
  - Full set of ISO standard icons included
  - No code changes needed due to new UI design
- Run safety critical UI in host during development and deploy to target hardware from Creator IDE



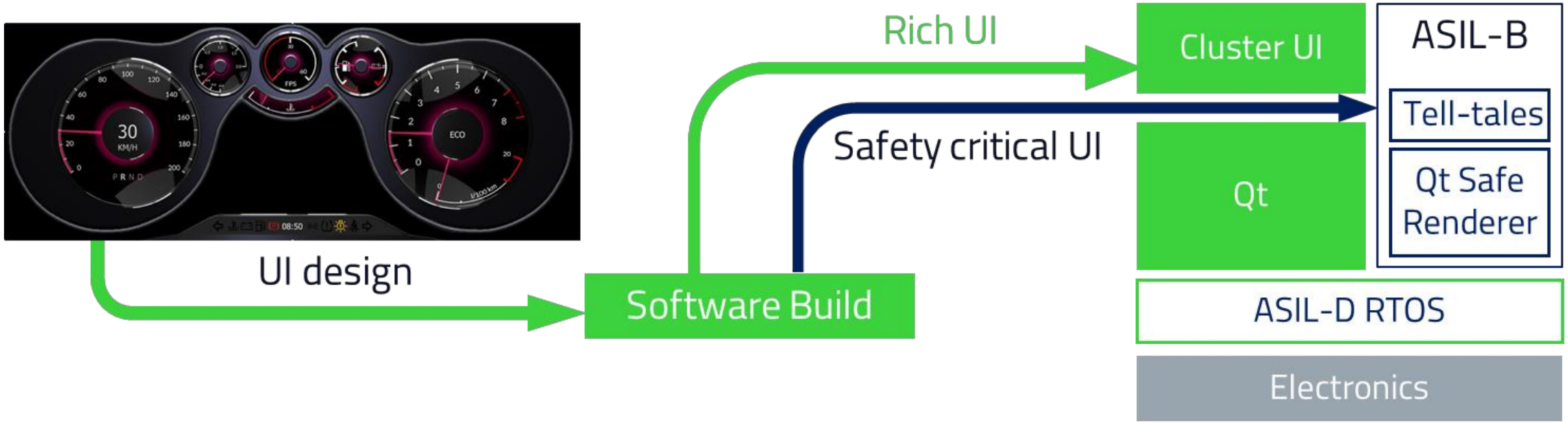
Three “Safe” QML Items

```
SafeImage {
    id: safeImage1
    objectName: "safeImage1"
    source: "indicator1.png"
    width: 64
    height: 64
    x: 321
    y: 123
 }
```

```
SafePicture {
    id: iconCoolant
    objectName: "iconCoolant"
    width: 30
    height: 30
    color: "#e41e25"
    source: "qrc:/iso-icons/iso_grs_7000_4_2426.dat"
  }
```

```
SafeText {
    id: safeText
    x: 256
    y: 8
    text: "Safe text.."
    font.pointSize: 12
 }
```

Certified Separation of Safety Critical UI



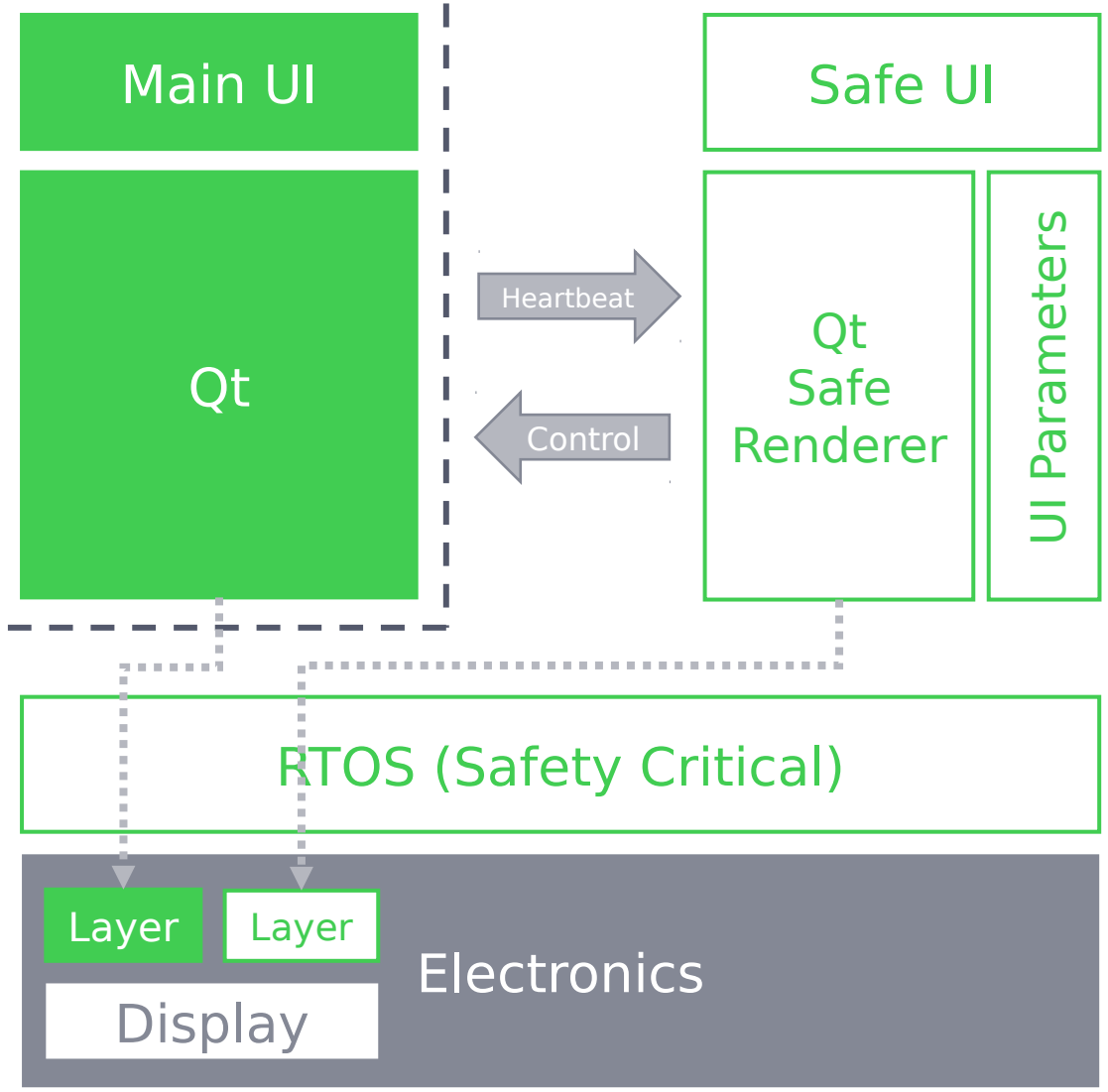
› Complete UI designed with Qt QML and tooling, including the Safety critical UI
› Tooling automatically separates the Safety critical UI parts from the other UI
› Safety critical UI rendered by Qt Safe Renderer

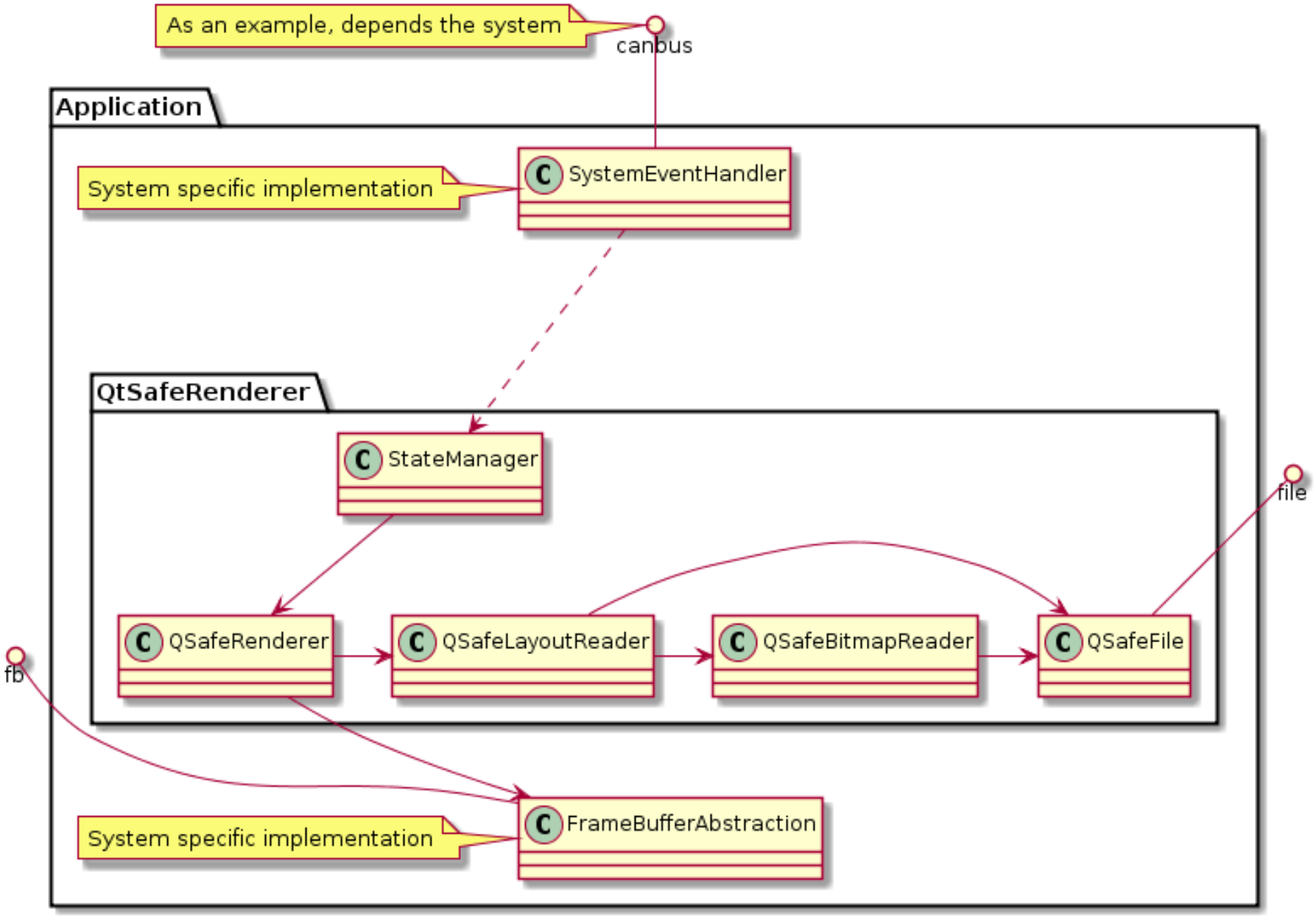
Qt Safe Renderer Runtime – Renderer for Safety Critical UI

- Rendering of safety critical UI by Qt Safe Renderer
  - Bitmaps
  - Text baked into bitmaps
- Fully MISRA C++ 2008 compliant
- Safe UI created with tooling – no changes to safety critical code
- Independent from Main UI
- Can react to Main UI failures and restart Main UI

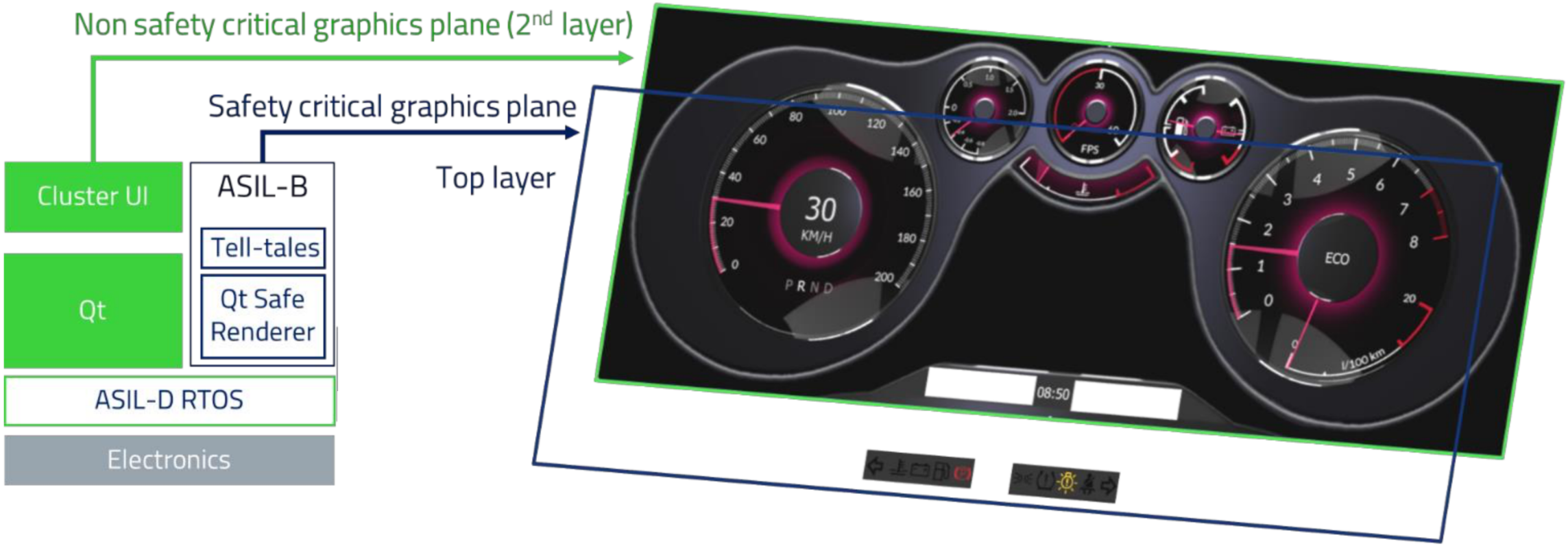
Qt Safe Renderer – Runtime Architecture

- Qt Safe Renderer runs as a safety critical process
  - No dependency to Main UI
  - Process separation by RTOS
- Safety critical UI drawn to a separate (topmost) HW graphics layer
  - Other processes can not overdraw it
- Qt Safe Renderer listens to heartbeat from Main UI and controls Main UI
- UI configuration information generated with build-time tooling
  - No changes to safety critical source code due to changes in UI design



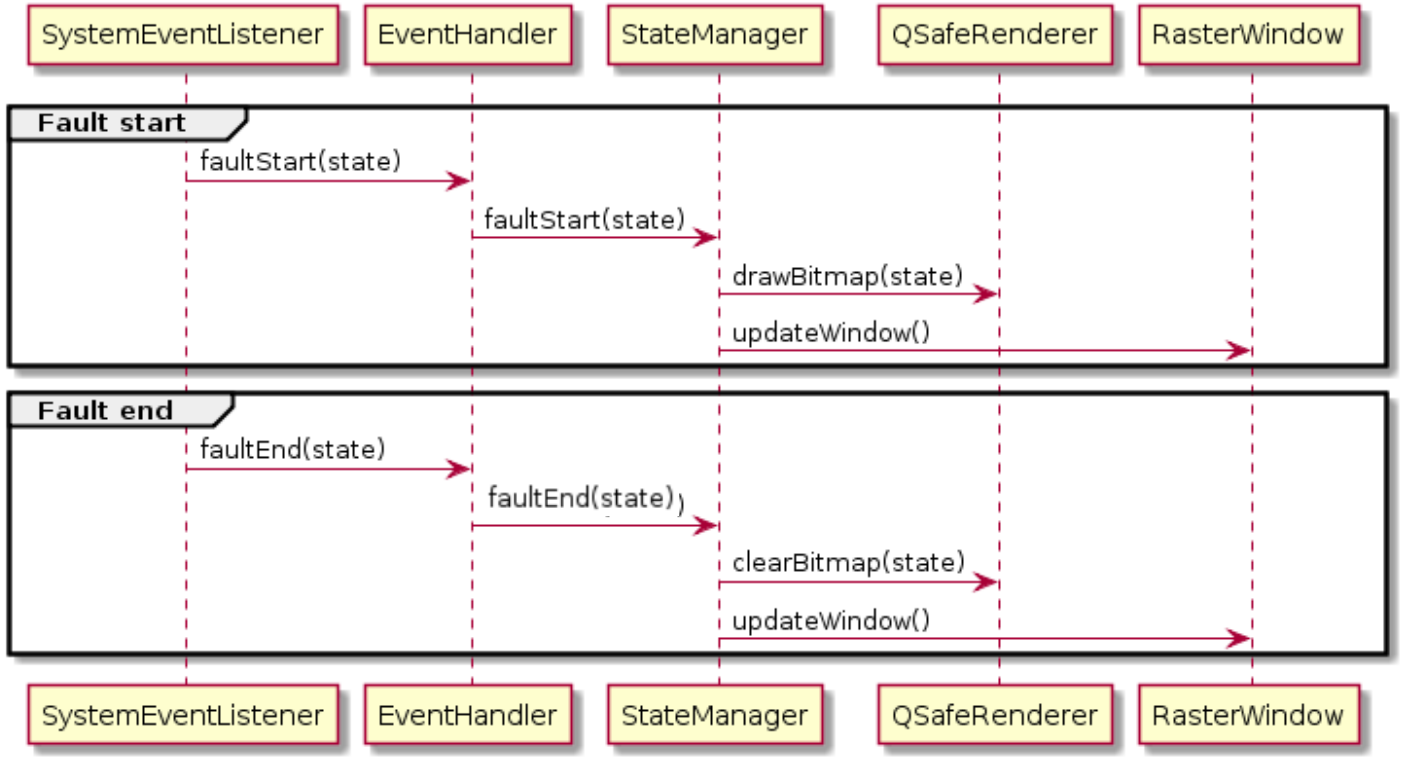


Qt Safe Renderer – Leveraging the Graphics HW Layers



Qt Safe Renderer – Typical Operation

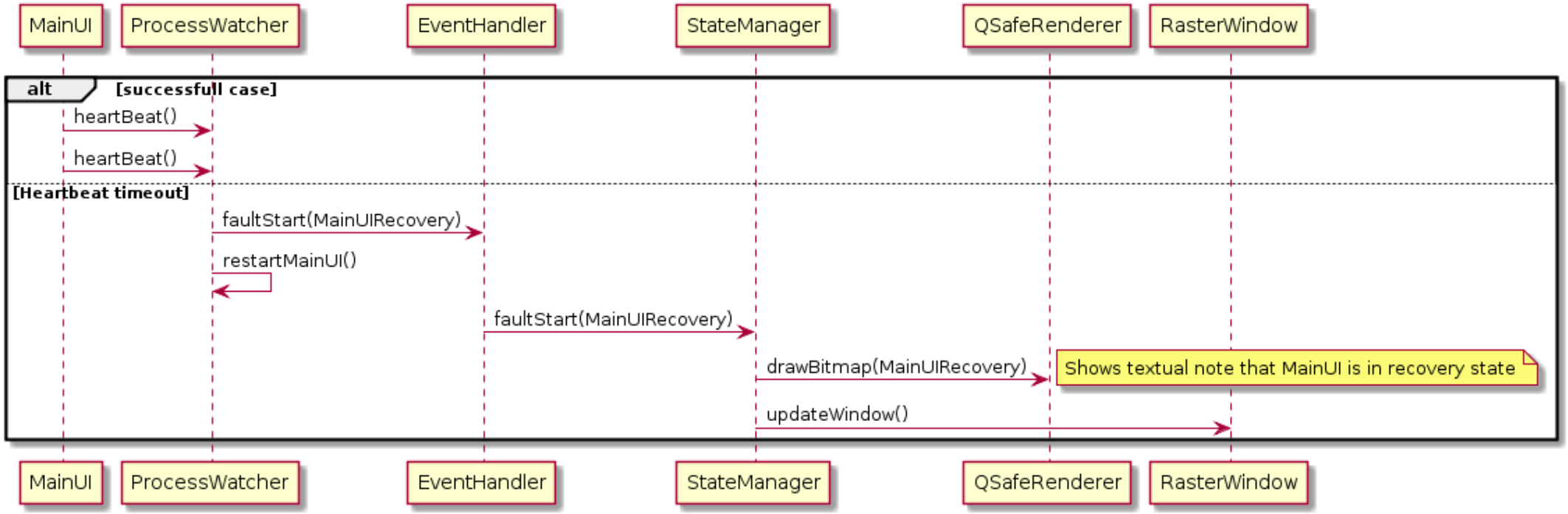
- Communications between Qt Safe Renderer and Main UI minimized
  - No dependency to each other
  - Qt Safe Renderer can control Main UI
- Qt Safe Renderer draws pre-defined bitmaps to screen based on system events
  - Fully independent operation even in case of failure in Main UI



Qt Safe Renderer – Main UI Recovery

› Qt Safe Renderer can listen to a heartbeat from Main UI

› In case of Main UI failure Qt Safe Renderer can restart the Main UI



Summary

Summary

- Objective of Functional Safety: avoid unacceptable risk of injury or damage to the health of people
- Multiple industry specific standards and local legislation set the framework
- Complete final product is certified
  - Use Qt Safe Renderer for safety critical UI
  - Using pre-certified RTOS and hypervisor is beneficial
- Qt is well suited as UI and application technology to create a certified system
  - Safety critical functionality needs to be adequately separated
- Certified systems for multiple different industries have been created with Qt
  - Qt Safe Renderer provides certified easy-to-use tooling and renderer
- Certification of Qt Safe Renderer: IEC 61508, ISO 26262, EN 50128 and ISO 62304

Thank You

kimmo.ollila@qt.io