

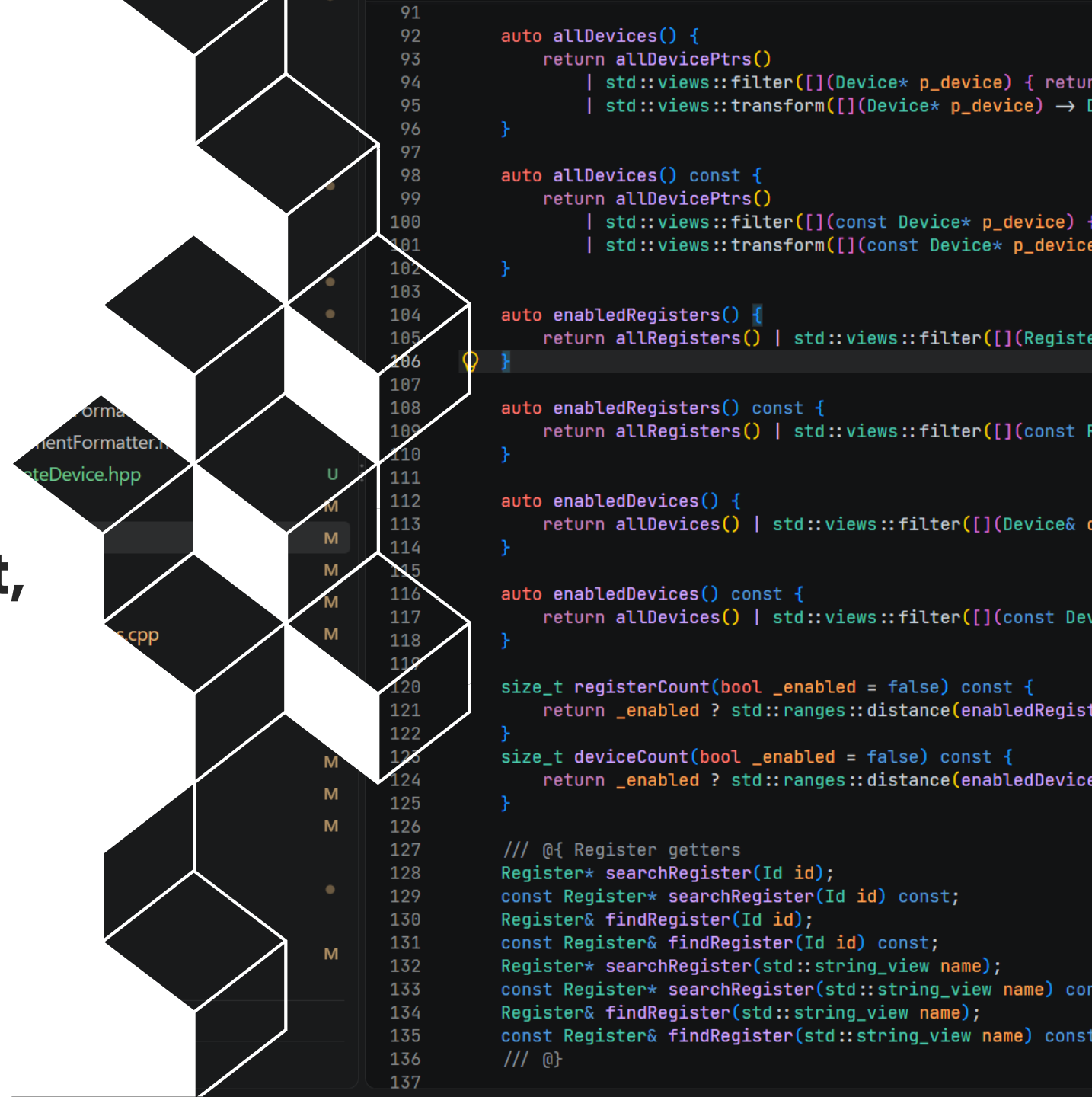


irfu

High-Performance Test Benches for Development, Validation & Production

DEDIP Workshop on
Computing Strategy

Shebli ANVAR
2026-09-24



Data Acquisition and Processing Stream

Embedded onboard real-time software

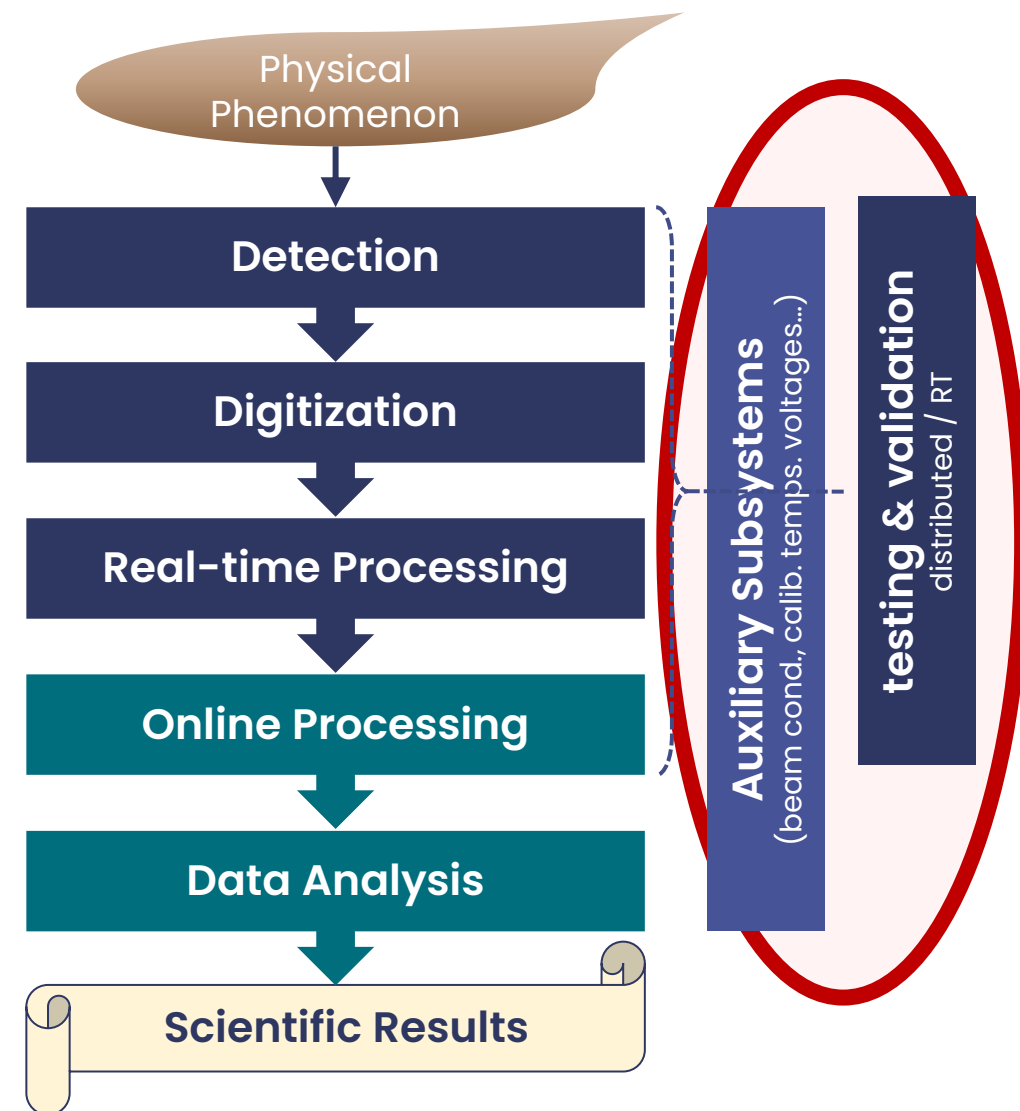
- Control and configuration of electronics
- Data formatting, routing & flow control
- Scientific processing (e.g. trigger algos)

Data processing pipelines

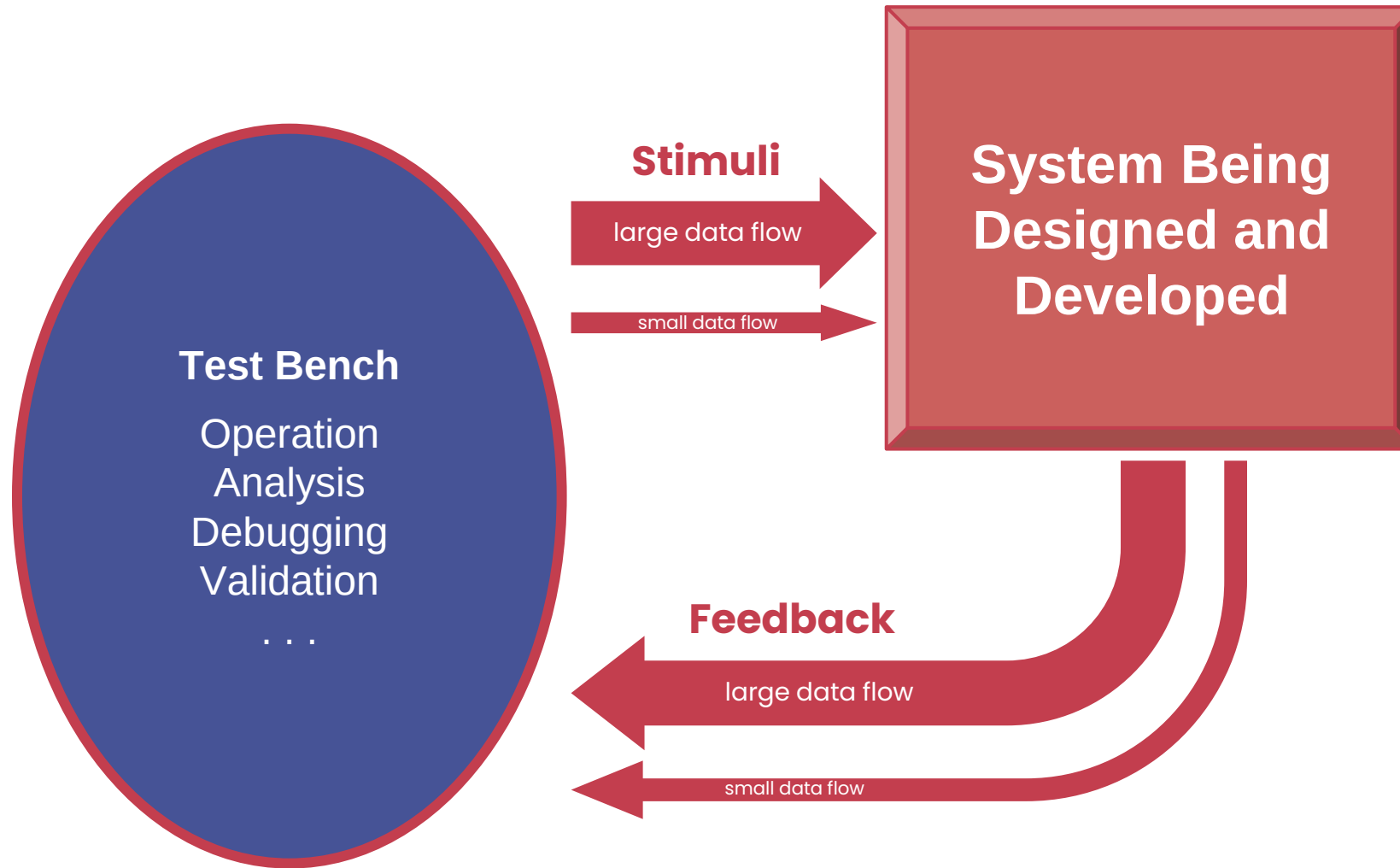
- Data Stream Management
- Computation chain of data analysis
- Software tools
 - scientific analysis
 - information retrieval
 - visualization
 - documentation

upstream

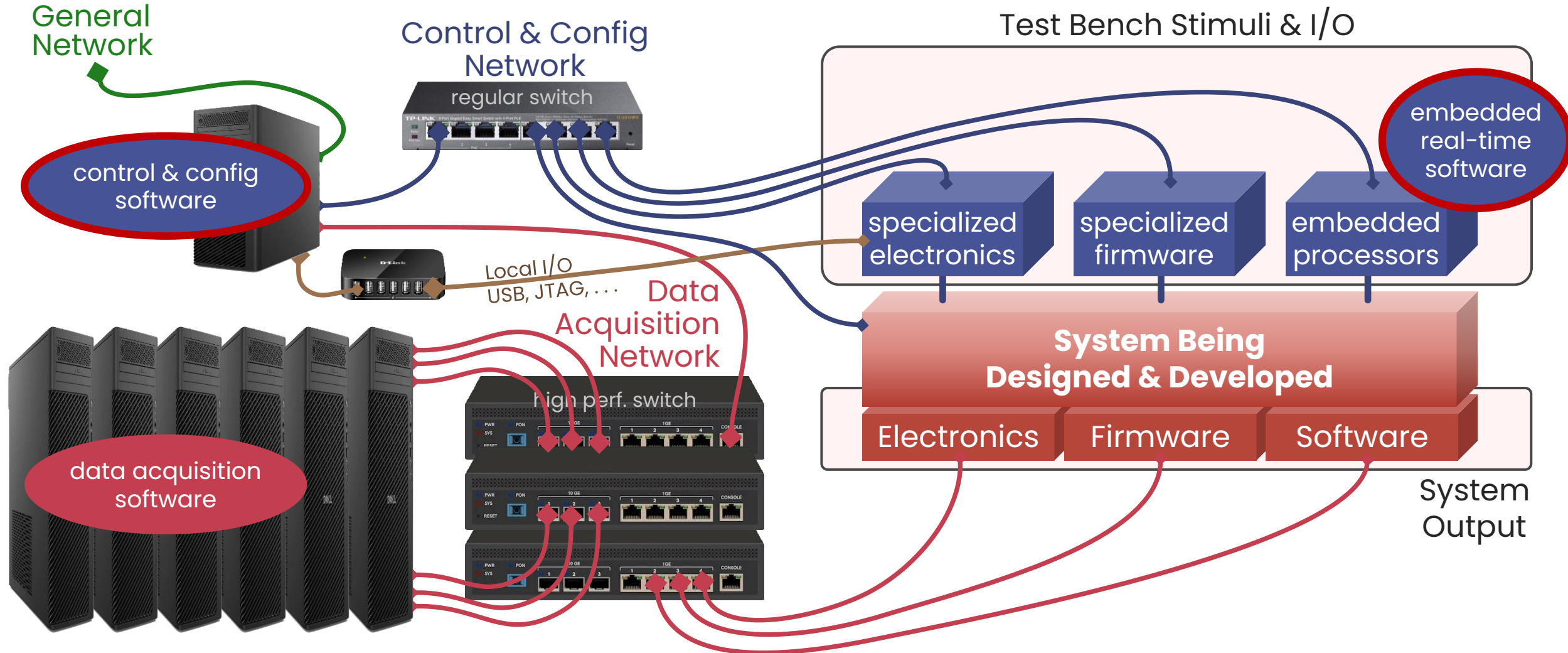
downstream



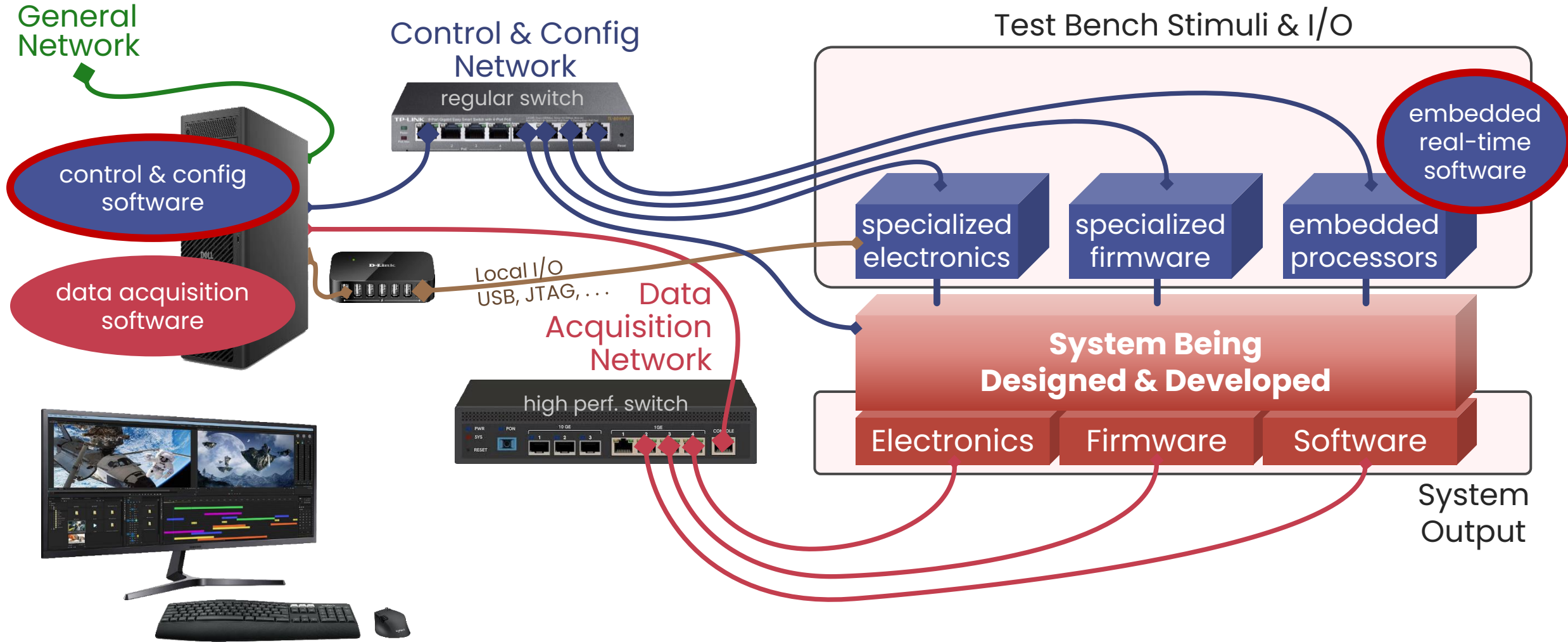
Basic Principle



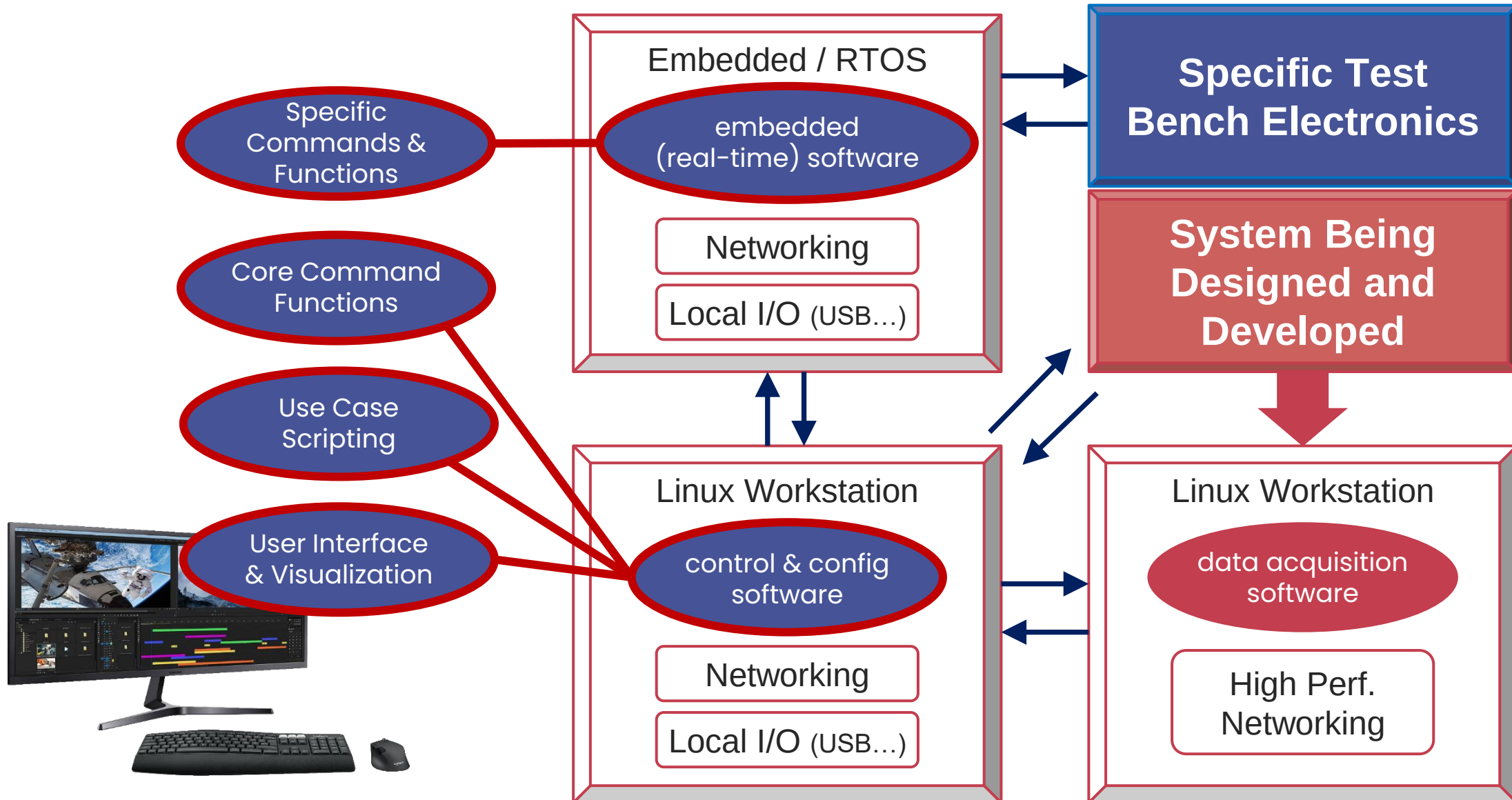
Typical Setup for Development of Control, Configuration, and Data Acquisition



Typical Test Bench Setup

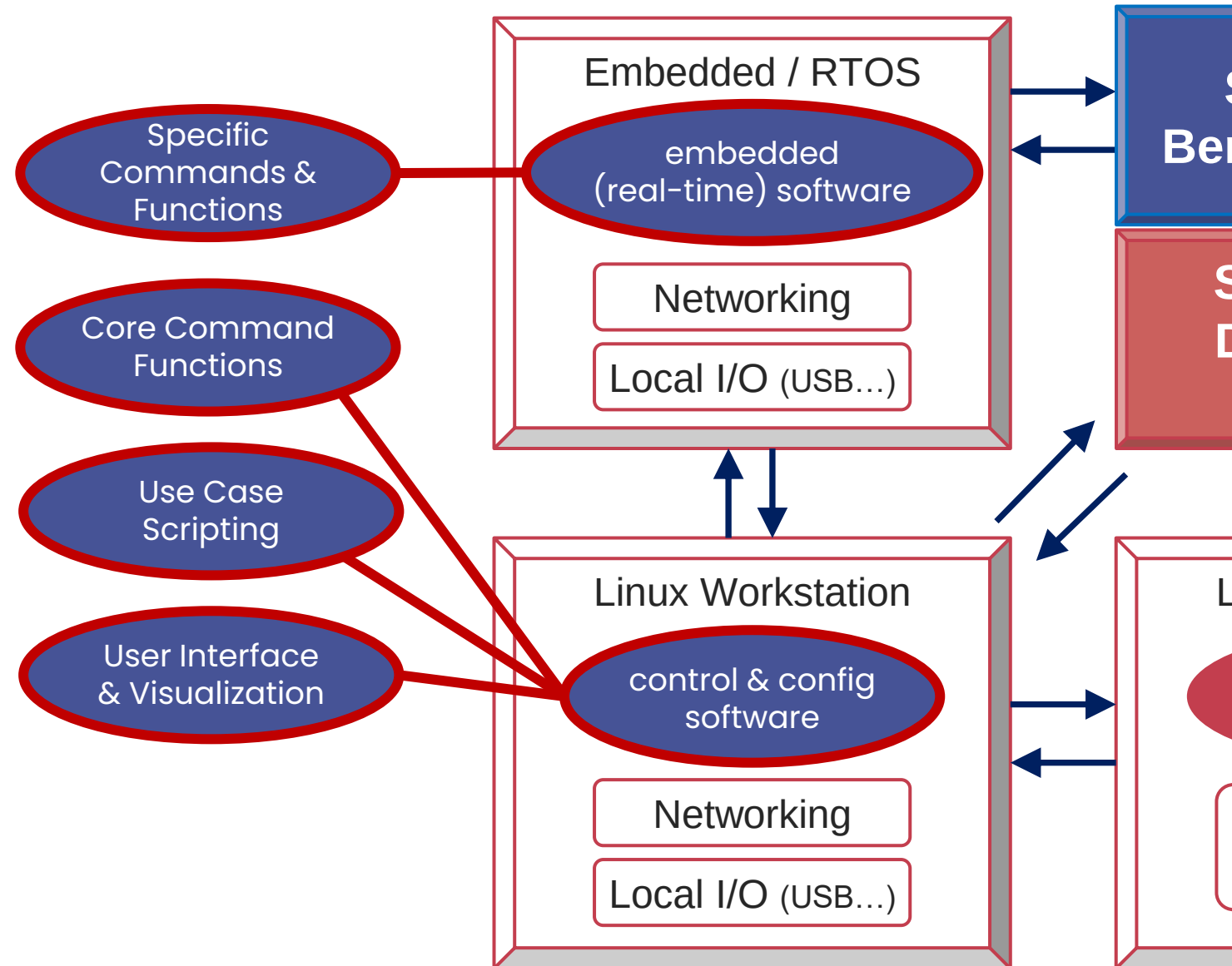


Software Context

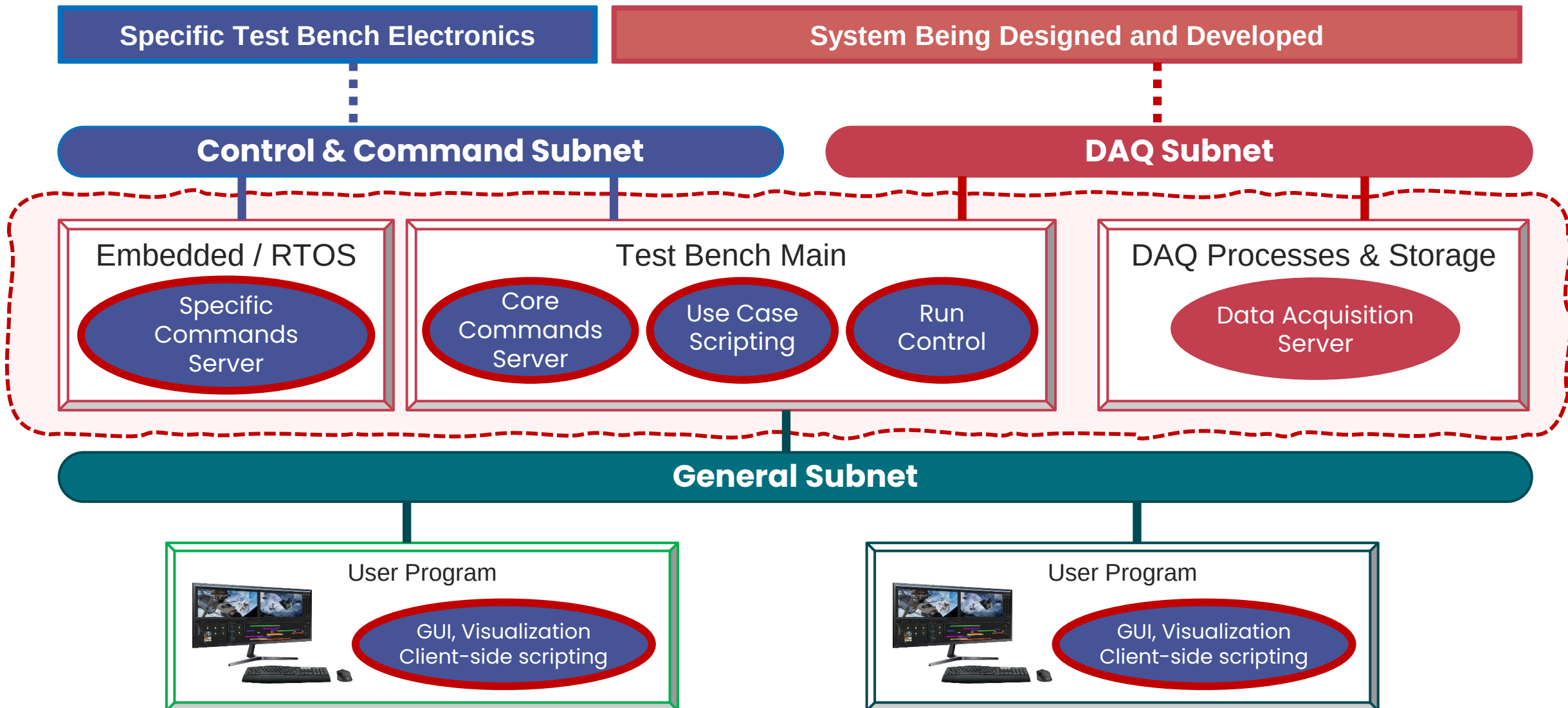


User needs

- Main operations
 - apply commands
 - command & output feedback
 - run data acq & display analyses
 - monitor system
- Multiple users
 - one at a time → R/W “sessions”
 - user traceability
 - group permissions
- Remote access
 - remote user interfaces
 - middleware
 - API & server
- Multiple instances
 - hardware configurations
 - deployment automation
 - organizational templates
- Run control
 - run configurations
 - run conditions
 - acquired data management



Distributed application over heterogeneous platforms



Software Reuse and Development Best Practices

Historical Overview

Why reuse?

- Reduce development time and costs.
- Improve reliability and maintainability.
- Standardize best practices across projects.
- Capitalize on development experience.

Possible Solutions at that Time

- Frameworks as early as 2000.
- Focus on C/C++ (dominant in embedded & scientific).

Mordicus Framework

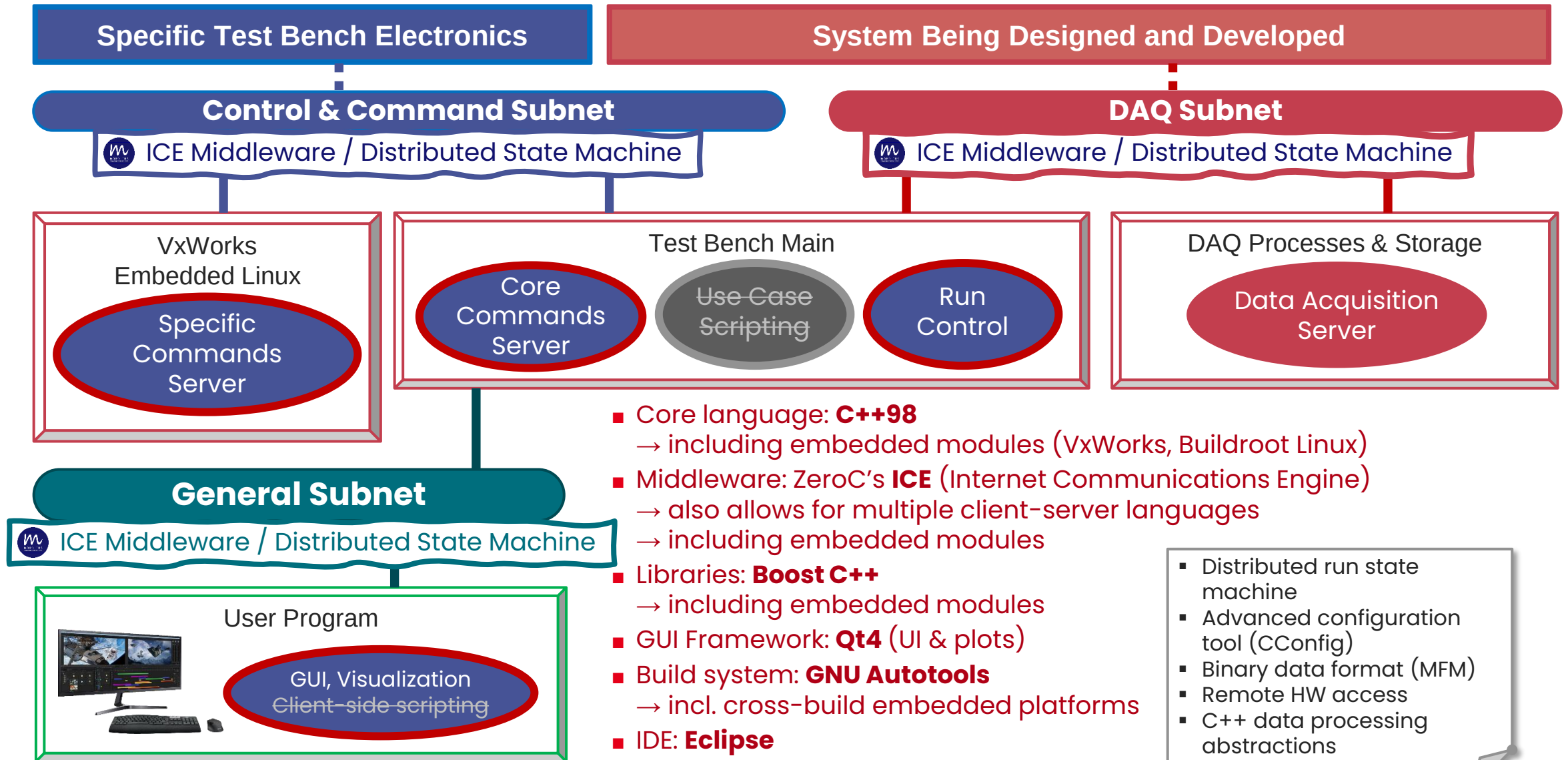
- **Goal:** Standardize test bench & DAQ development across projects.
- Developed for **Antares/KM3NeT** (~2008).
- Used in **Minos, Asacusa, ForFire, GET, Svom** (UGTS test bench)

successful deployments & adaptations over many years





Mordicus FW for Test Bench & small/midsize DAQ





Maintenance Challenges



OS and toolchain evolution

- Linux distros: Full maintenance became very difficult.
- Windows/MacOS/VxWorks: Eventually abandoned due to resource constraints.
- Eclipse IDE: Maintained, but still challenging.

Technical Debt and Programming Paradigms

- **C++ evolution:** New standards as of C++11 include conceptual leaps (e.g. move semantics)
 - adds difficulty adapting to C++ based FW (ICE, Boost)
 - makes code updating very time-consuming
- **Complex build system:** GNU Autotools + customizations for Ice, Qt, Embedded modules
- **Multiple deprecations / evolutions:** APIs, data formats



→ Still managed to maintain but drift from new software technologies



Upgrading & adapting to present & future projects does not meet agility expectations

New Software Technologies

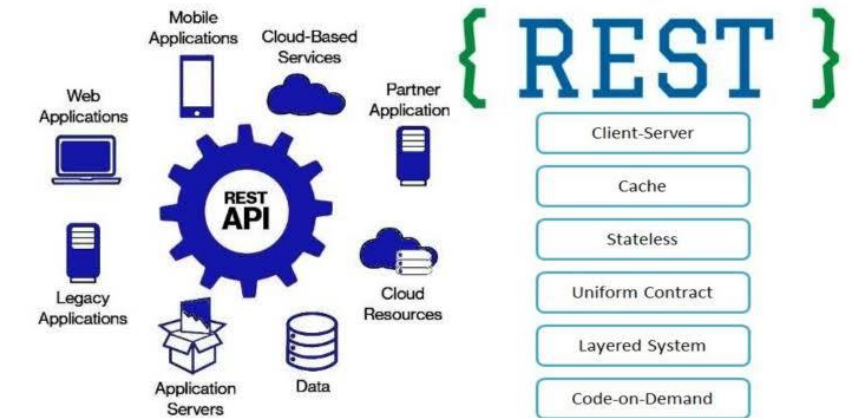
Python Programming

- rapid & easy development of client & server wrapper
- for « human speed » apps
- bound to C++ modules for CPU/GPU intensive parts
- **Allows for natural use case scripting**



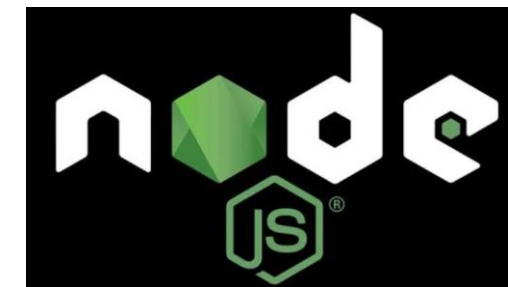
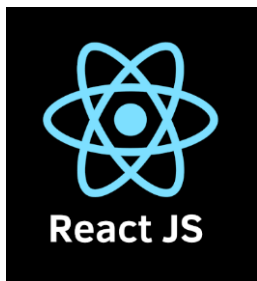
REST APIs and functional programming

- database-driven
- replacing stateful OOP



Web-based Applications

- vs. traditional C++ frameworks like Qt
- main language: JavaScript + framework



Modern Web-based Applications

Pros

- Client only requires a modern browser
 - OS independence
no specific support for Linux/Windows/MacOS
 - no deployment of GUI/client application
→ no maintenance for clients
 - cross-browser incompat. a thing of the past
- Backend server development is independent
 - Most adapted soft. tech. & languages for the server
 - e.g. C++ core with Python bindings as API
 - can be containerized (maintenance & deployment)
 - can be integrated in cloud/service environment
 - Architecture favors modular design
 - reuse of already developed software is improved
 - loose coupling favors different evolution paces
 - All test bench functionalities concentrated in server
- REST API & database persistence
 - Standard web middleware for remote user access
 - Operations are immune to client crash
 - Traceability of operations
 - Mutualized database services

Cons

- Learning curve: Switching from trad. UI (Qt, MVC) to Web: skill gap
- JS framework churn → long-term maintenance is difficult
- Needs engineering skills and large enough team over time
- Python not ideal for asynchronous / mutlitasking

Which skills & manpower?

Core Server Functionalities

- Embedded Systems, RTOS, driver dev.
- Specialized I/O: TCP/UDP, USB, Spacewire, etc.
- Modern C++ & build sys. (native and cross-dev.)
- Python → C++ Binding

Web applications: server side

- Database persistence and programmatic access
- Python programming

Web GUI

- Javascript front-end frameworks
- Advanced graphics for data visualization

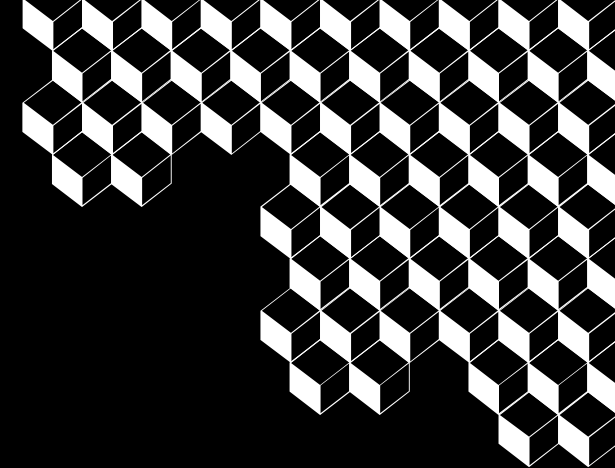
Containers & DevOps

- Active in project team (not as support only)

There may be overlaps but one engineer must focus on a single aspect at a time



Team dedicated to test bench, data acquisition sub-projects



Thank you.

High-Performance Test Benches for
Development, Validation & Production

Shebli ANVAR
2026-09-24