LNLS Brazilian Synchrotron Light Laboratory · sirius
CNPEM Brazilian Center for Research in Energy and Materials
MINISTRY OF SCIENCE, TECHNOLOGY INNOVATION AND COMMUNICATION · BRASIL
ICALEPCS2017 Barcelona · Spain, October 8-13
Palau de Congressos de Catalunya
www.icalepcs2017.org

Software and Gatew are Development for Sirius BPM Ele ctronics Using a Service-Orient ed Architecture

L. M. Russo, Beam Diagnostics Group
Brazilian Synchrotron Light Laboratory, LNLS, Brazil

Introduction

The Brazilian Synchrotron Light Laboratory (LNLS) is in the final stages of developing an open-source BPM system for Sirius, a 4th-generation synchrotron light source under construction in Brazil. The system is based on the MicroTCA.4 standard comprising AMC FPGA boards carrying FMC digitizers and a CPU module. The software is built with the HALCS framework and employs a service-oriented architecture (SOA) to export a flexible interface between the gateware modules and its clients, providing a set of loosely-coupled components favoring reusability, extensibility and maintainability. In the paper, the BPM system will be discussed in detail focusing on how specific functionalities of the system are integrated and developed in the framework to provide SOA services. In particular, two domains will be covered: (i) gateware modules, such as the ADC interface, acquisition engine and digital signal processing; (ii) software services counterparts, showing how these modules can interact with each other in a uniform way, easing integration with control systems.

Generic Hardware Architecture

- Standard communication interfaces: PCIe, UART, etc.
- Hierarchical Design, e.g. based on open-source Wishbone Bus Protocol
- Peripherals with minimal interaction, acting as isolated components
- Desirable to have knowledge about internal components such as: unique ID, name, address range, version, capabilities

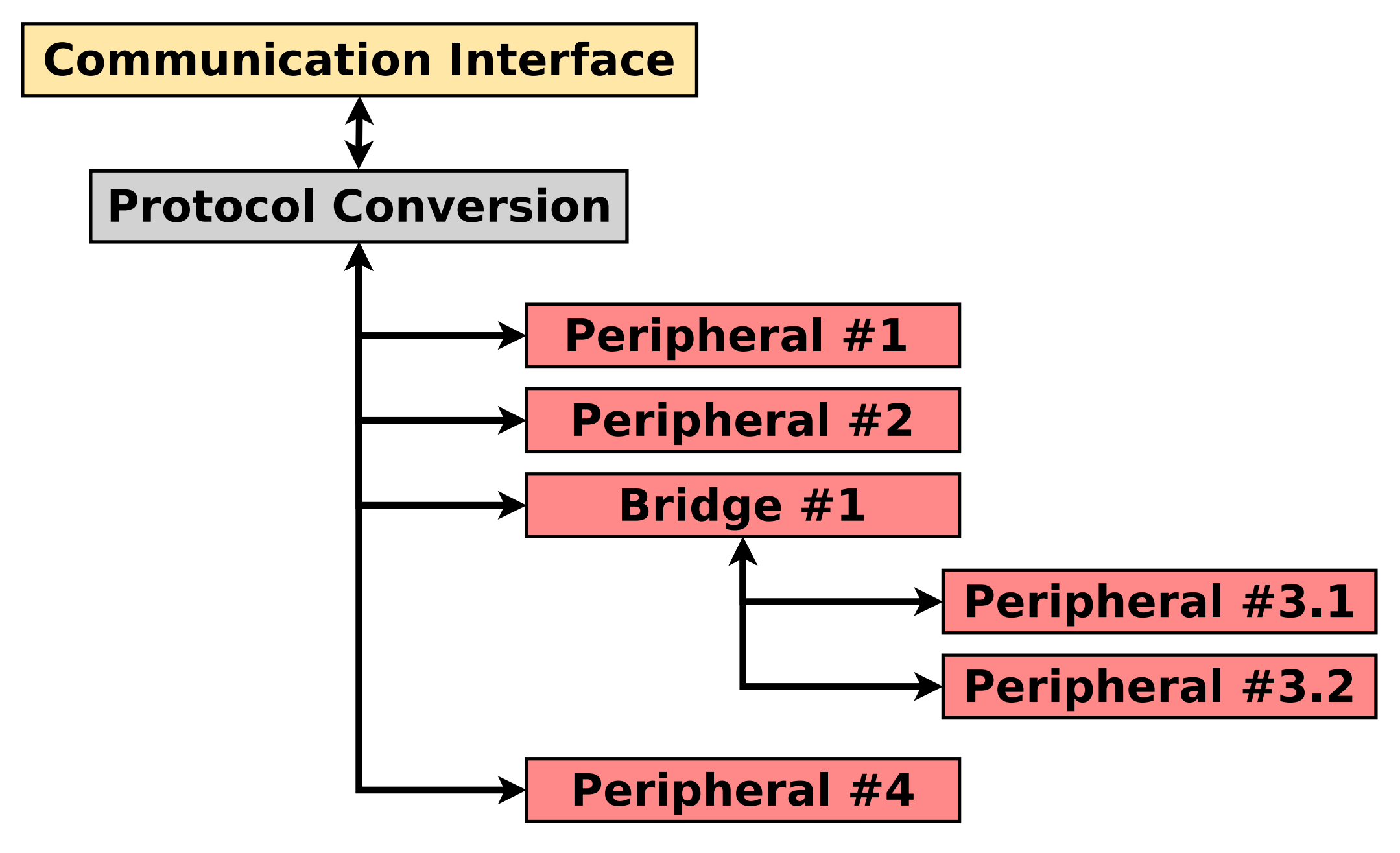

Figure 1. Generic Hardware Architecture.

SOA-based Software Architecture

- Software abstracts hardware components as *services*
- Uses a common protocol to communicate with hardware device
- *Services* can coordinate themselves by using an *Intra-Controller protocol*
- Protocols acts as a flexible/extensible API

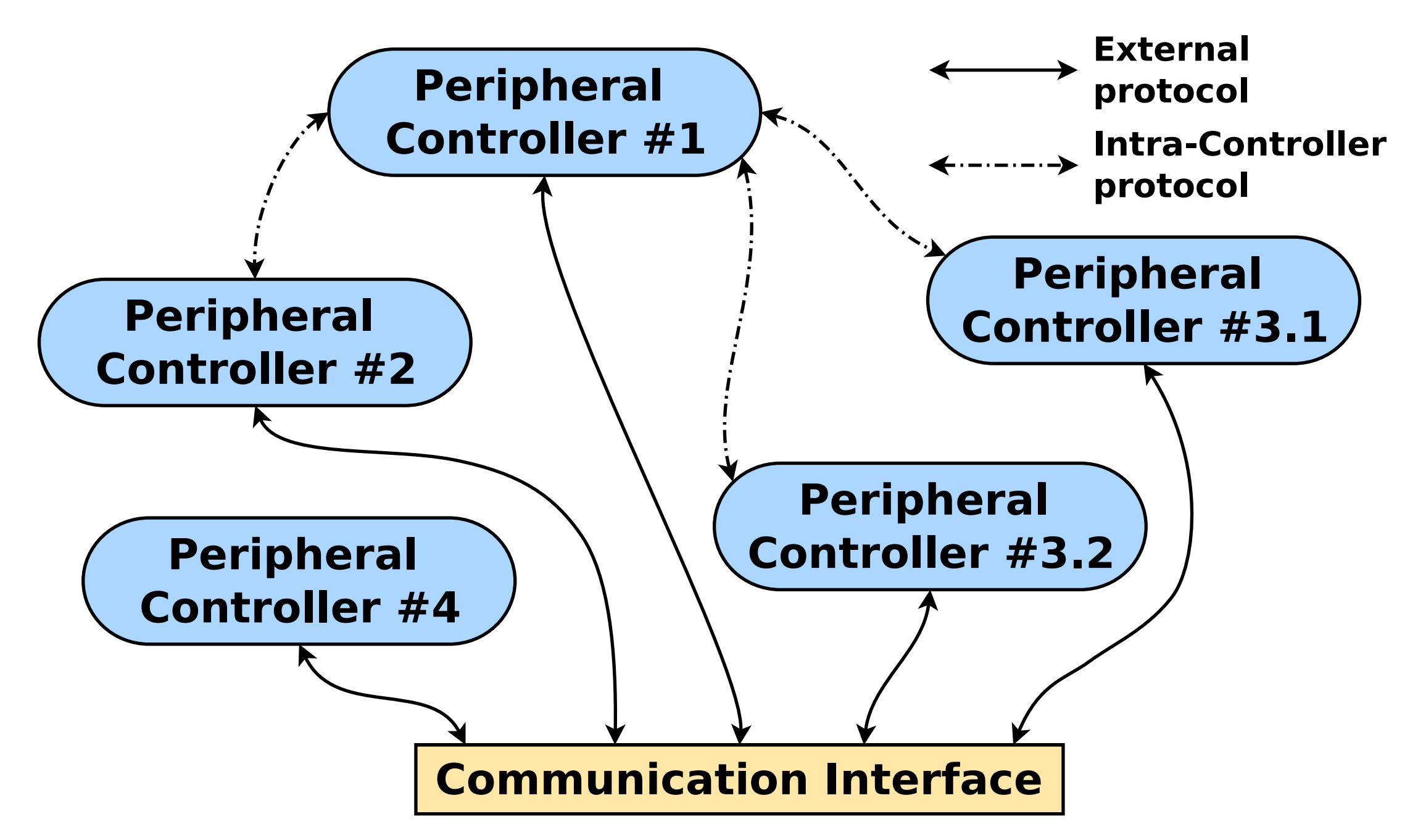

Figure 2. SOA-based Software Architecture.

HALCS Architecture

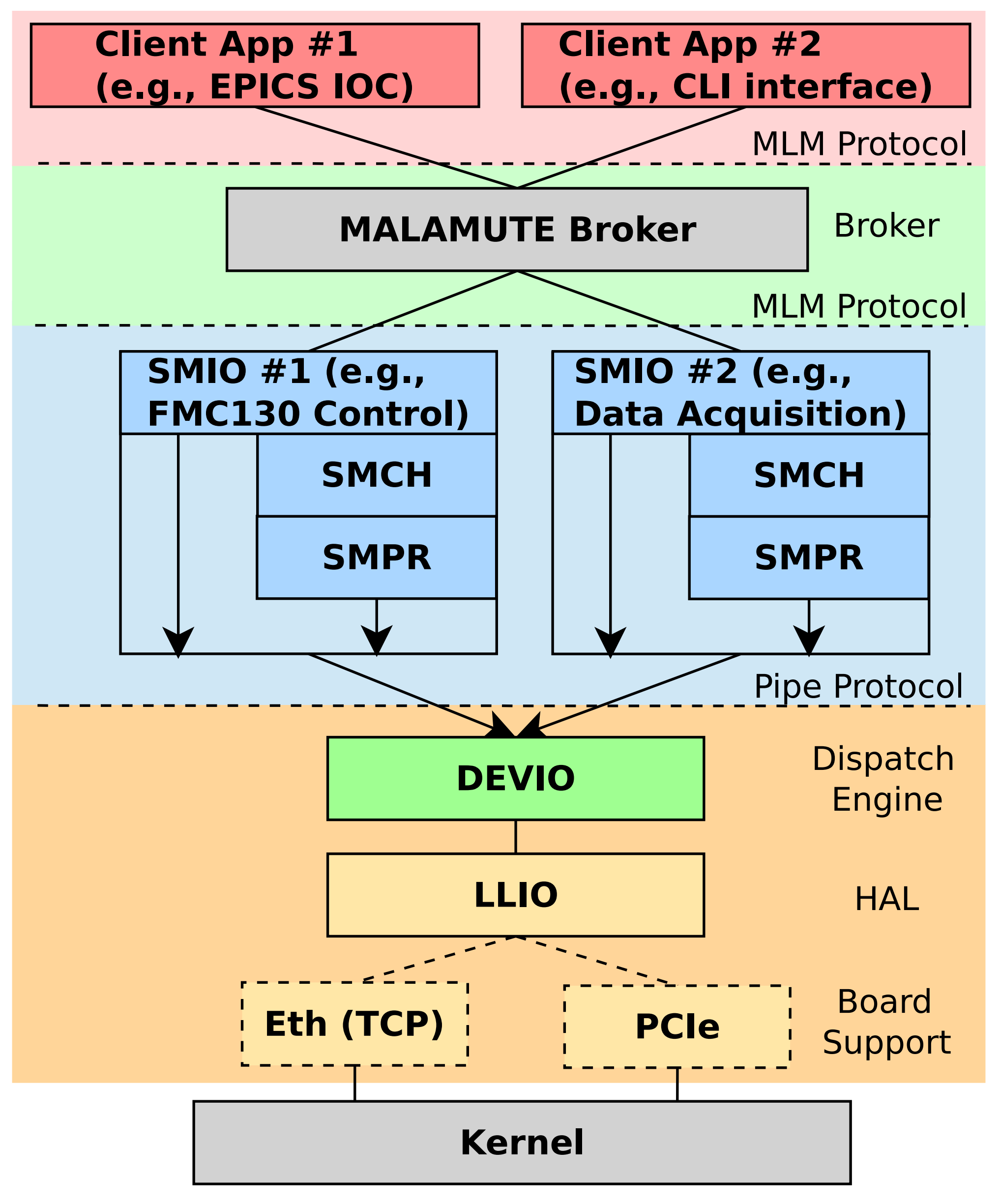

Figure 3. HALCS Framework Architecture.

- *HALCS* Framework implements SOA principles, using an *Inversion of Control* design paradigm
- Uses a common RPC protocol to expose *services* functionalities
- **Broker** provides *discoverability* and *reliability* to *services* using *Mailbox messaging pattern*
- *Services* (**SMIO**) register functions
- *Services* can use additional abstractions:
  - **SMCH** for external chips: AD9510 clock distributor and PLL, Si57x clock oscillator, etc.
  - **SMPR** for external protocols: SPI, I2C, etc.
- **DEVIO:** Event-driven reactor engine
- **LLIO:** Hardware Abstaction Layer

HALCS Dataflow

1. Caller instantiates **LLIO**
2. Caller instantiates **DEVIO**
3. Caller registers **SMIOs** manually or via **SDB**
4. **SMIOs** registers into Malamute Broker
5. **Clients** may send/receive messages to any SMIO

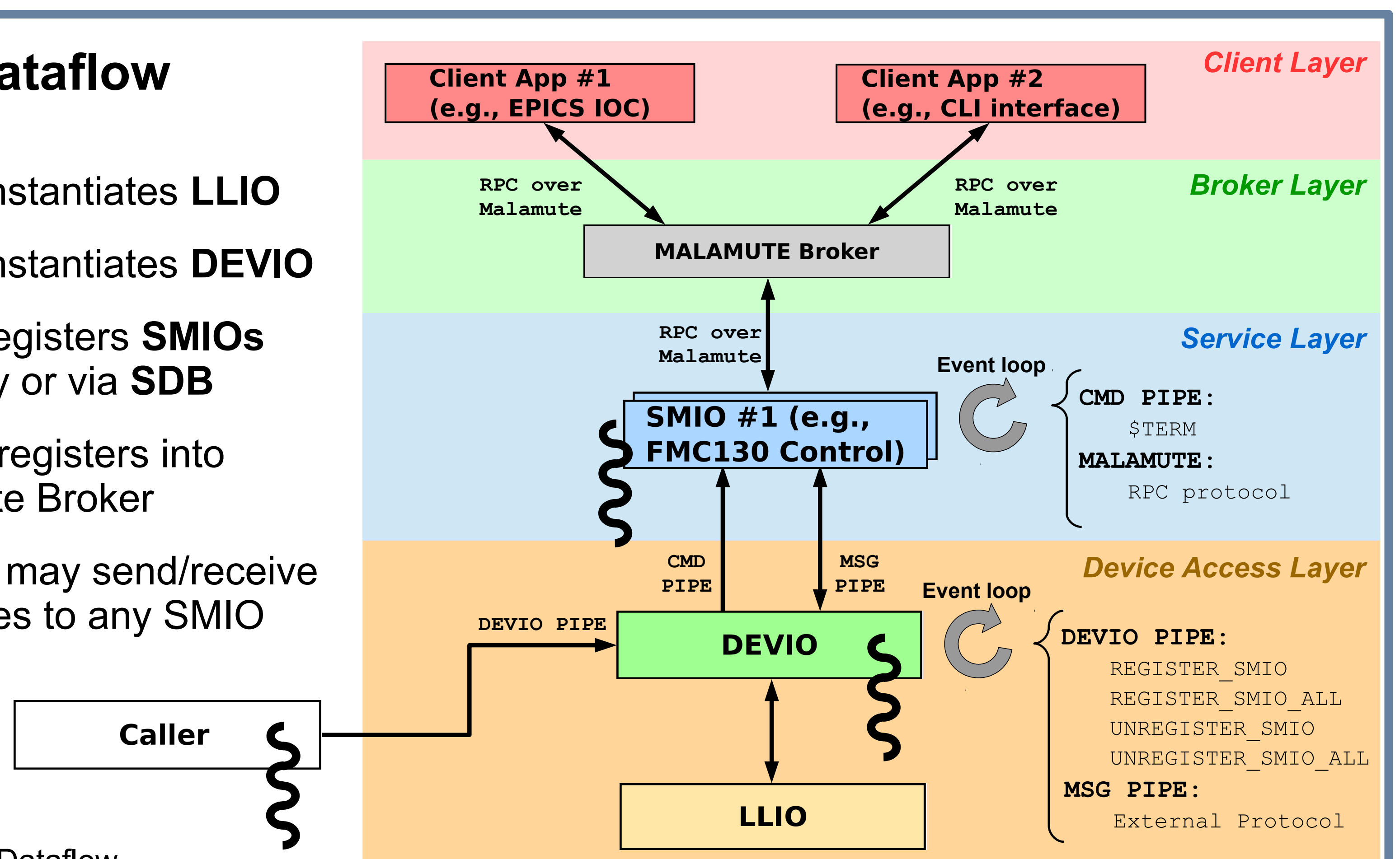
Client Layer
Client App #1 (e.g., EPICS IOC)
Client App #2 (e.g., CLI interface)
Broker Layer
RPC over Malamute
MALAMUTE Broker
Service Layer
SMIO #1 (e.g., FMC130 Control)
Event loop
CMD PIPE: $TERM
MALAMUTE: RPC protocol
Device Access Layer
CMD PIPE
MSG PIPE
DEVIO PIPE
DEVIO
Event loop
DEVIO PIPE: REGISTER_SMIO REGISTER_SMIO_ALL UNREGISTER_SMIO UNREGISTER_SMIO_ALL
MSG PIPE: External Protocol
Caller
LLIO

Figure 4. HALCS Dataflow.

BPM Project Gateware

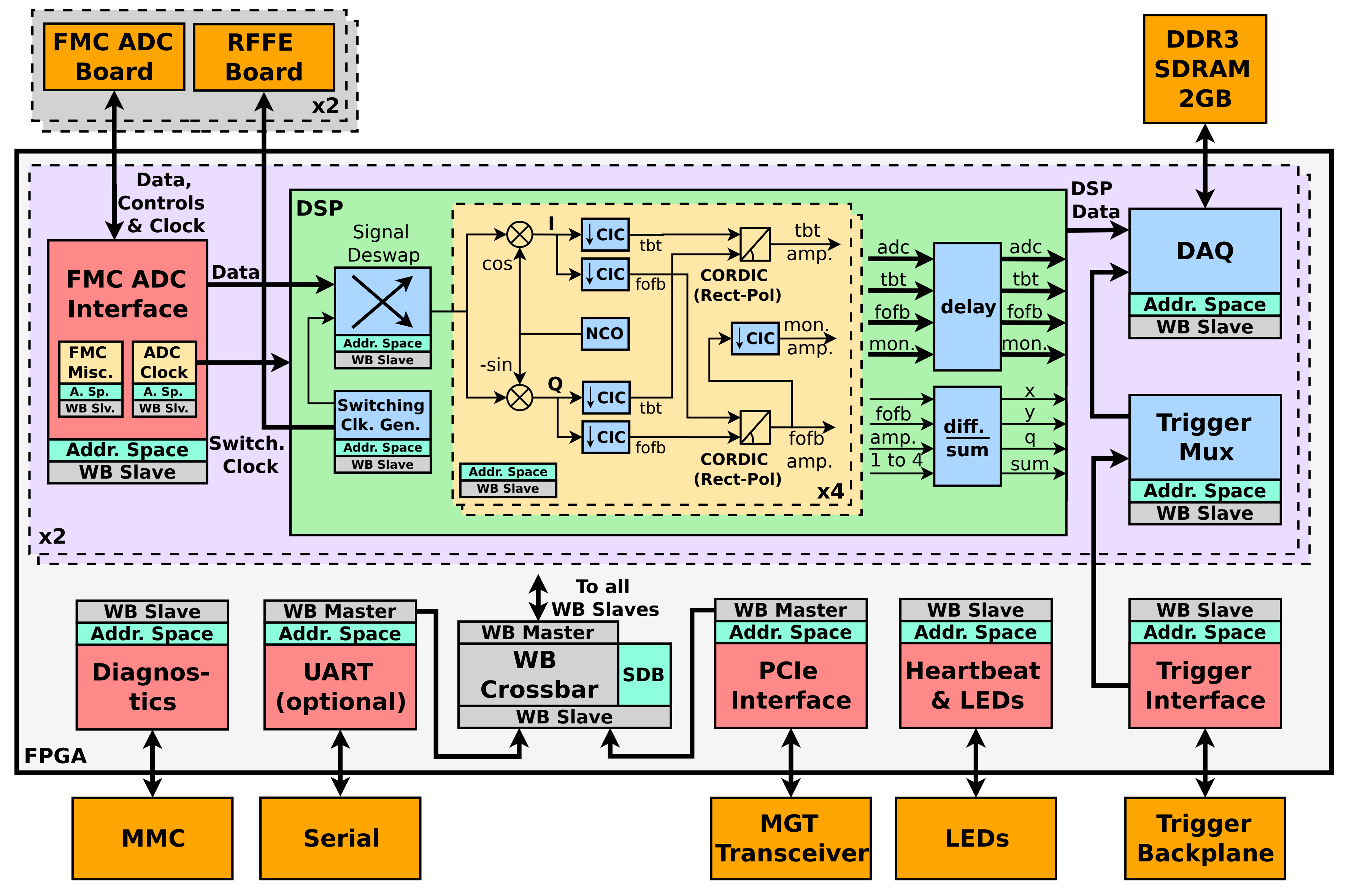

Figure 5. BPM Hardware Architecture.

- Reusable gateware components, based on **Wishbone B4**
- Self-Describing Bus (**SDB**) aware components:
  - Easier for software to be **gateware-agnostic**: dynamic offsets, version, capabilities
  - Software can act as a userspace driver
  - Application logic pushed to **Client Layer**

BPM Project Software using *HALCS* framework

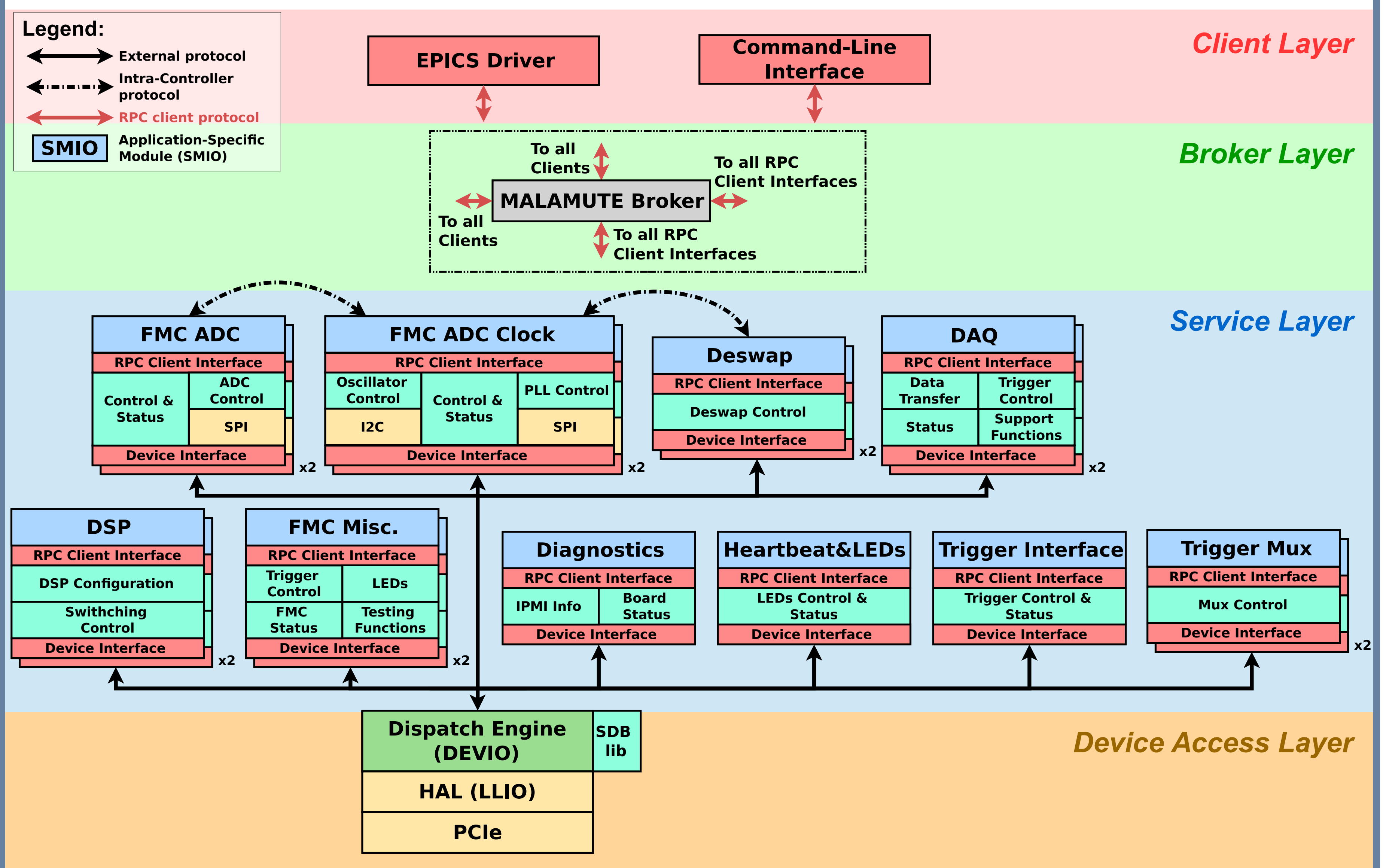

Figure 6. BPM Software Architecture.

Summary

- SOA principles applied to low-level software maximize reuse of commonly used functionalities
- Best used in conjunction with isolated hardware components with minimal interaction among each other
- Succesfully deployed in the BPM and the upcoming MicroTCA.4 Timing Receiver projects

Links

HALCS Framework:	`https://github.com/lnls-dig/halcs`
BPM EPICS IOC:	`https://github.com/lnls-dig/bpm-epics-ioc`
BPM Gateware:	`https://github.com/lnls-dig/bpm-gw`
DSP Cores:	`https://github.com/lnls-dig/dsp-cores`
Infra Cores:	`https://github.com/lnls-dig/infra-cores`
Timing Receiver:	`https://github.com/lnls-dig/timing-receiver-gw`