Tomasz Włostowski
CERN BE-CO-HT

White Rabbit Node Core

Introduction, Applications & Status

8th White Rabbit Workshop
Genève, October 6th 2014

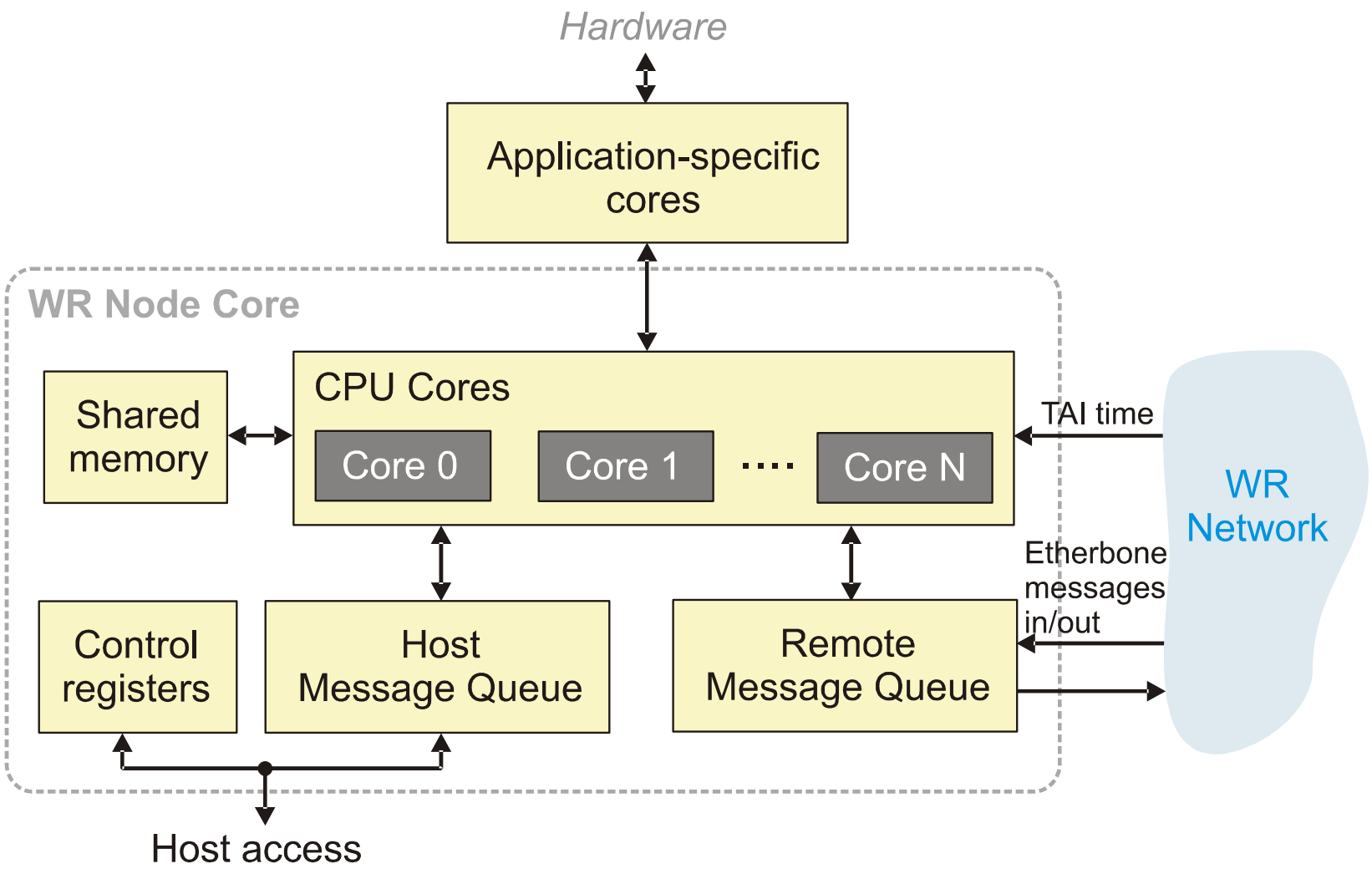
Outline

- Introduction
- HDL and software design
- WR node applications at CERN
- LHC Instability Trigger Distribution project
- Status and plans

What is WRNC?

- A generic, multipurpose White Rabbit Node
- Applications run on one or more deterministic CPU cores
- Communicates with external world over Etherbone
- Extended by connecting Wishbone-compatible user cores and hardware

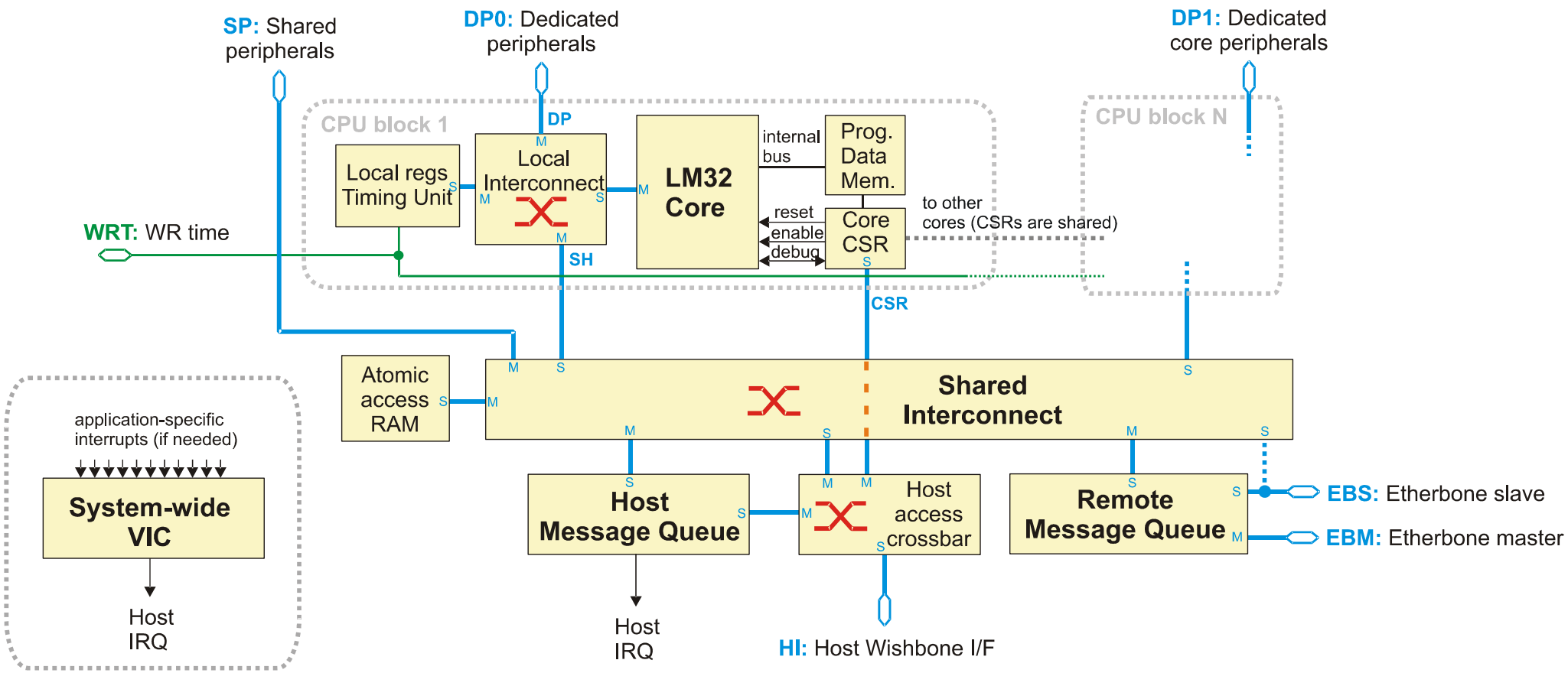
What is WRNC?


Hardware
Application-specific cores
WR Node Core
Shared memory
CPU Cores
Core 0
Core 1
....
Core N
TAI time
WR Network
Etherbone messages in/out
Control registers
Host Message Queue
Remote Message Queue
Host access

Why we designed WRNC?

- Variety of hard real time systems under our responsibility
- Determinism (in most cases) is more important than speed
- Programming in C takes less time and effort than HDL design
- One generic core instead of many custom-designed devices

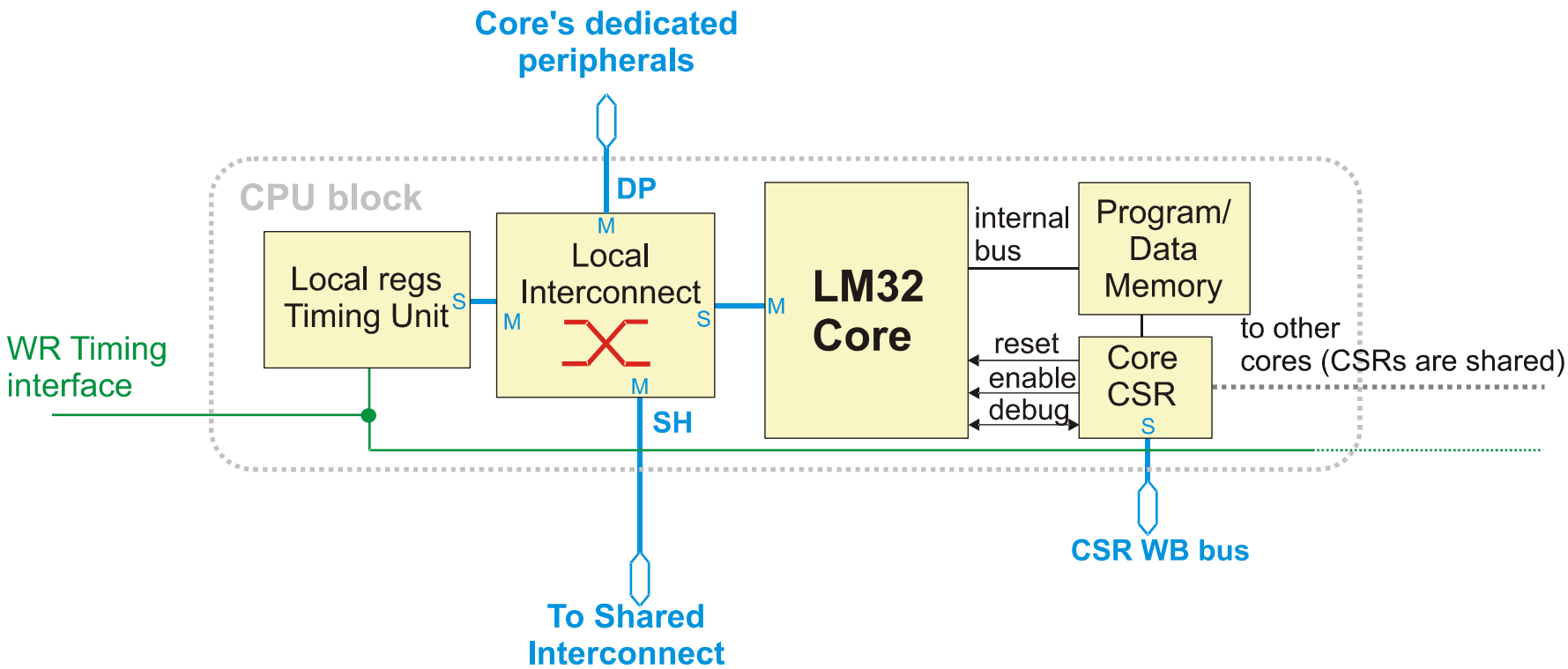
HDL Design


SP: Shared peripherals
DP0: Dedicated peripherals
DP1: Dedicated core peripherals
CPU block 1
CPU block N
DP
Local regs Timing Unit
Local Interconnect
LM32 Core
internal bus
Prog. Data Mem.
reset
enable
debug
Core CSR
to other cores (CSRs are shared)
WRT: WR time
SH
CSR
Atomic access RAM
Shared Interconnect
application-specific interrupts (if needed)
System-wide VIC
Host IRQ
Host Message Queue
Host IRQ
Host access crossbar
HI: Host Wishbone I/F
Remote Message Queue
EBS: Etherbone slave
EBM: Etherbone master

The CPUs

- Up to 8 LM32 cores (can be replaced by any Wishbone CPU)
- Programed in *bare metal* C, using standard GCC tool chain
- No interrupts (to ensure determinism)
- Inter-core communication through shared memory
- Program loading and flow control from the host system

CPU Core block



CPU Core block

- User-configurable amount of code/data memory. Single-cycle access (no caches)
- No direct host access to the memory except for program loading
- Host-accessible CSRs for program loading and debugging
- Timing Unit: a simple delay/interval generator
- Local Registers: WR time counter and message queue polling bits
- Private per-CPU Wishbone Dedicated Peripheral

Shared Memory

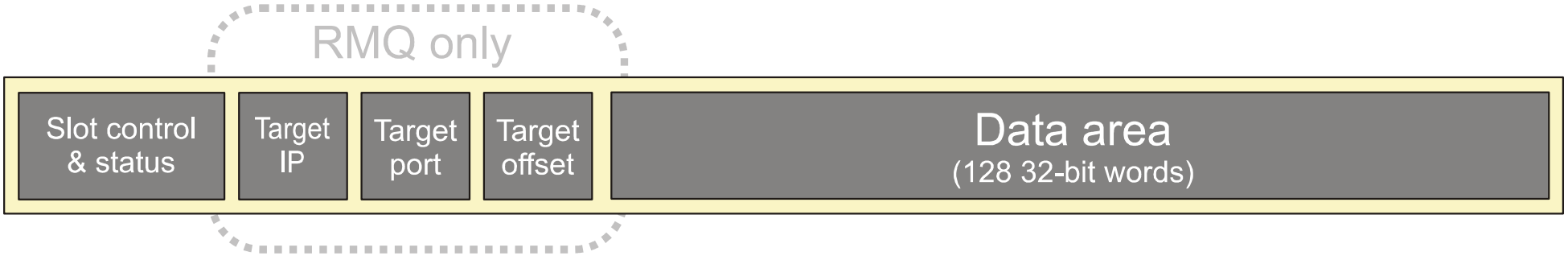
- Foreseen for multiprocessor communication and task synchronization
- Accessible by all cores, Etherbone and host
- 8kB fixed size
- Atomic inc/dec operations (semaphores/queue pointers)
- Atomic set/clear/flip operations (mutexes, flags, events)
- Operations are selected by most significant address bits

Message Queues

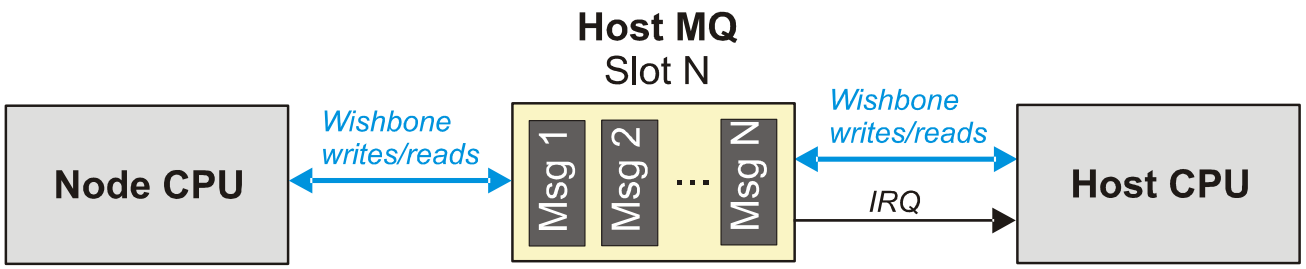
- Organized as multi-word bidirectional FIFOs
- One or more incoming/outgoing slots per MQ
- Each slot can hold multiple messages
- User-configurable slot width and depth
- Message integrity ensured by MQs
- No high-level flow control mechanism
- MQ slots assigned exclusively to CPUs

MQ messages

- Up to 128 32-bit words per message
- Payload format defined by the user
- Sending/receiving controlled by simple commands (**`claim`**, **`ready`**, **`discard`**, **`purge`**) written to the MQ's control register

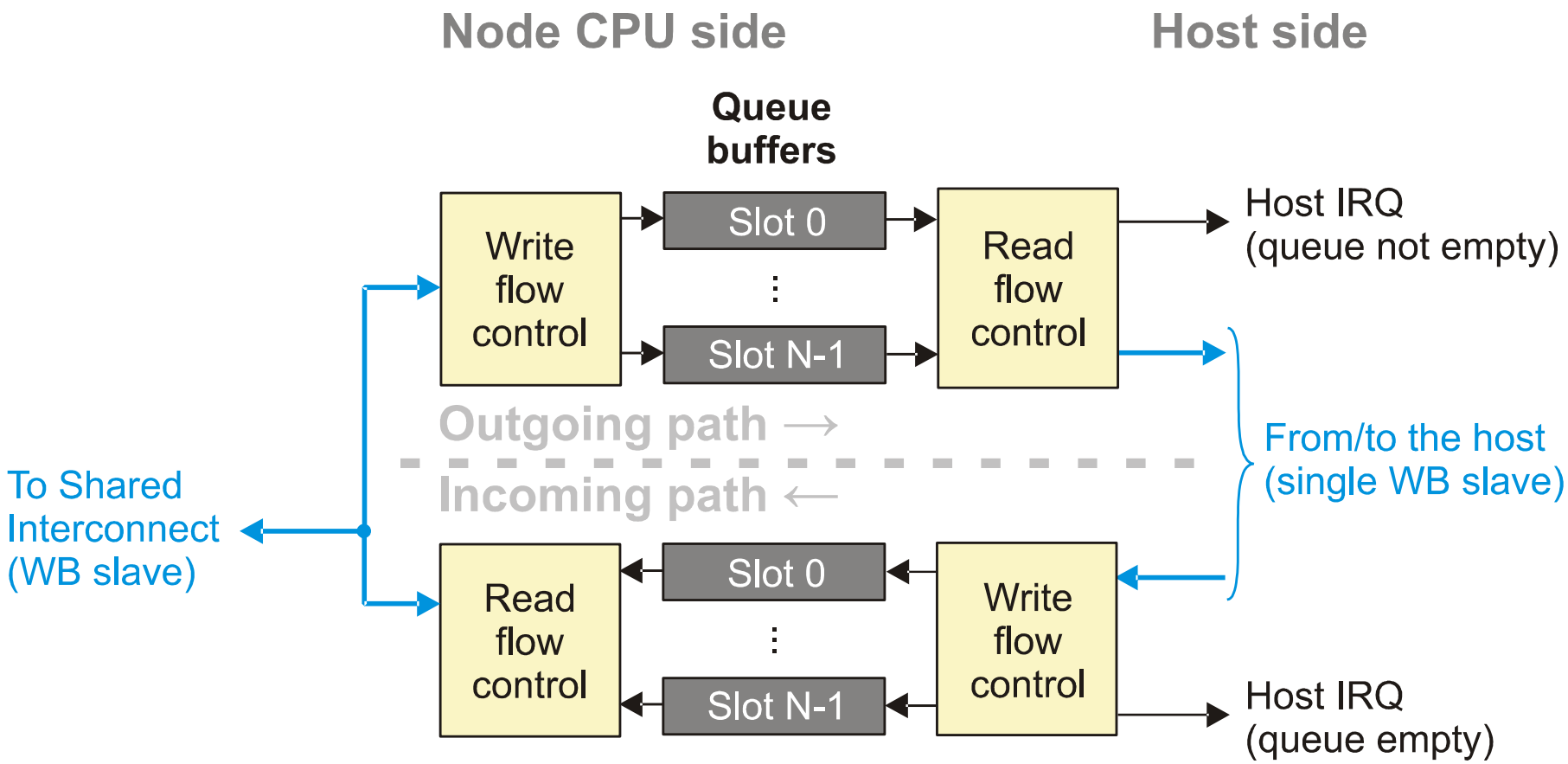


Host Message Queue

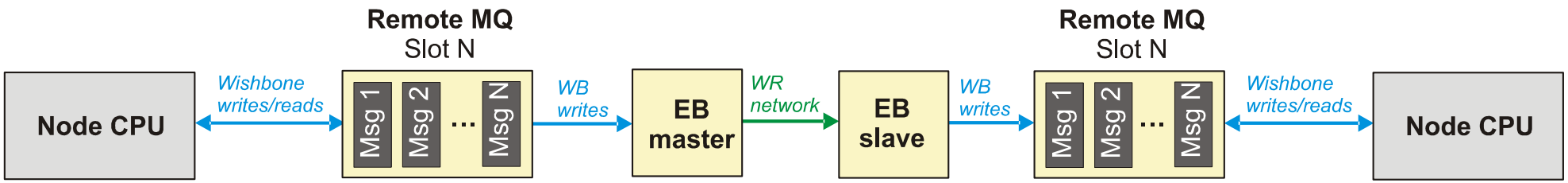


- Main route of communication between host software and the node's CPUs
- Nothing but a fancy FIFO
- Can generate interrupts for the host to avoid polling

Host Message Queue

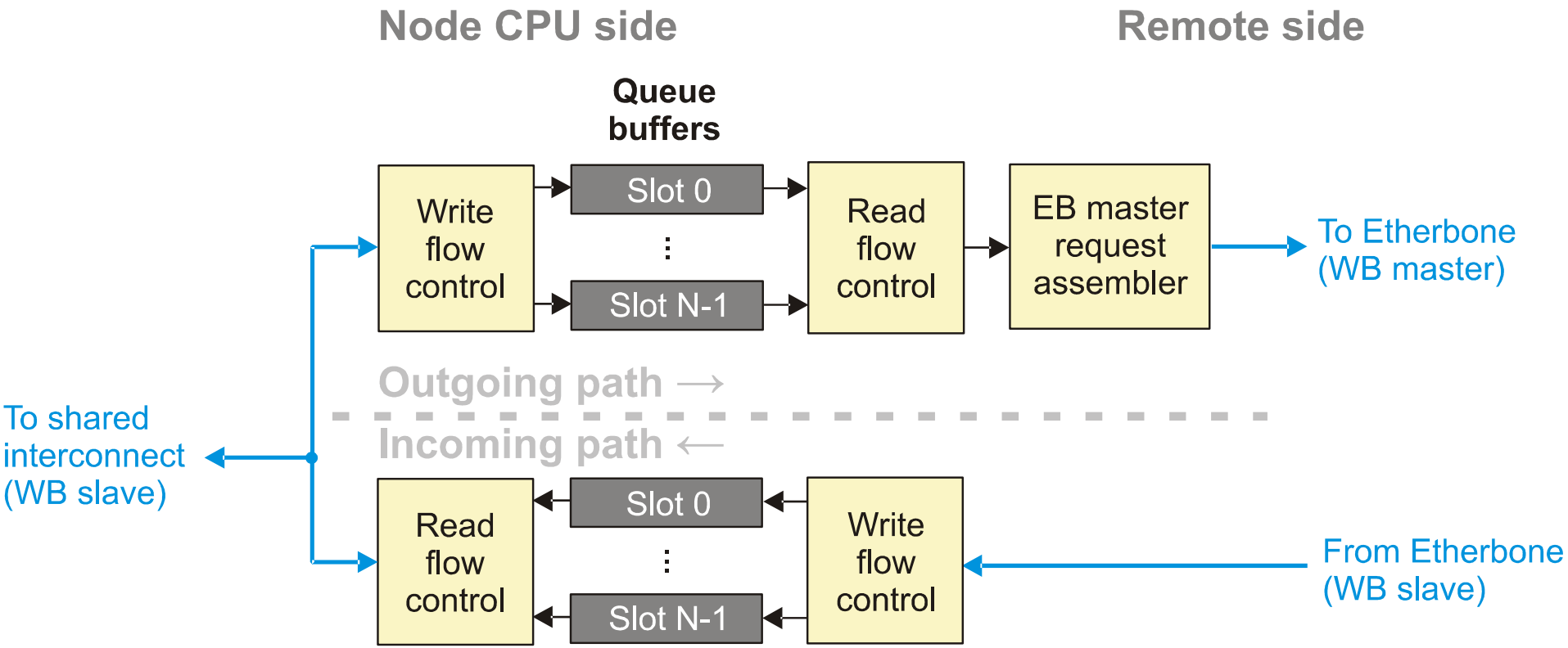


Remote Message Queue



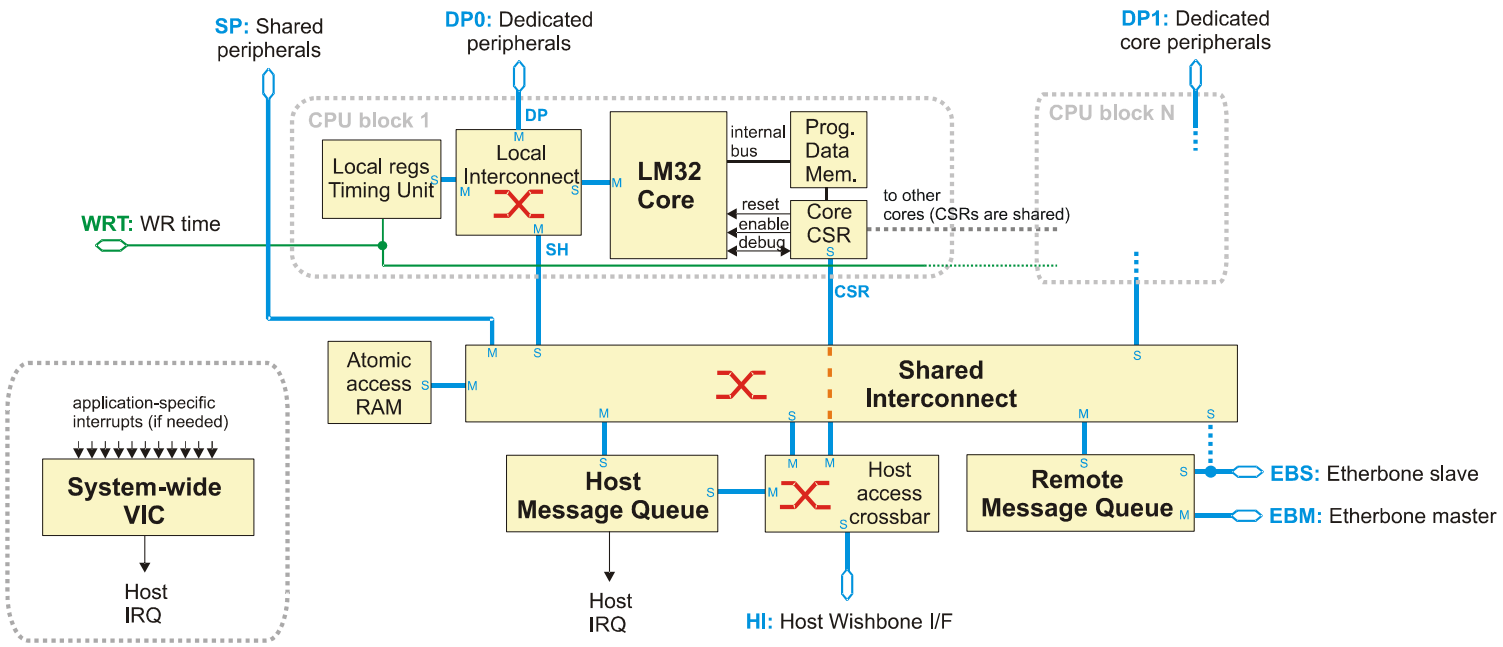
- Communication between remote WR nodes
- Outgoing path generates Etherbone requests to an external EB master core
- Incoming path same as in HMQ, driven by an external EB slave
- Slots multiplexed by Round-Robin scheduler

Remote Message Queue



Shared Interconnect

- Wishbone crossbar allowing the cores to talk with the Shared Memory and Message Queues
- Raw Etherbone slave access to SI possible
- Shared Peripheral WB port accessible by all CPUs (e.g. a big DDR memory)



Software

- Generic kernel driver + user space C library
- Applications can be written in user space only → easier development
- Host Message Queue and Shared Memory access
- Loading CPU applications and controlling each core's execution flow
- Python API (future)

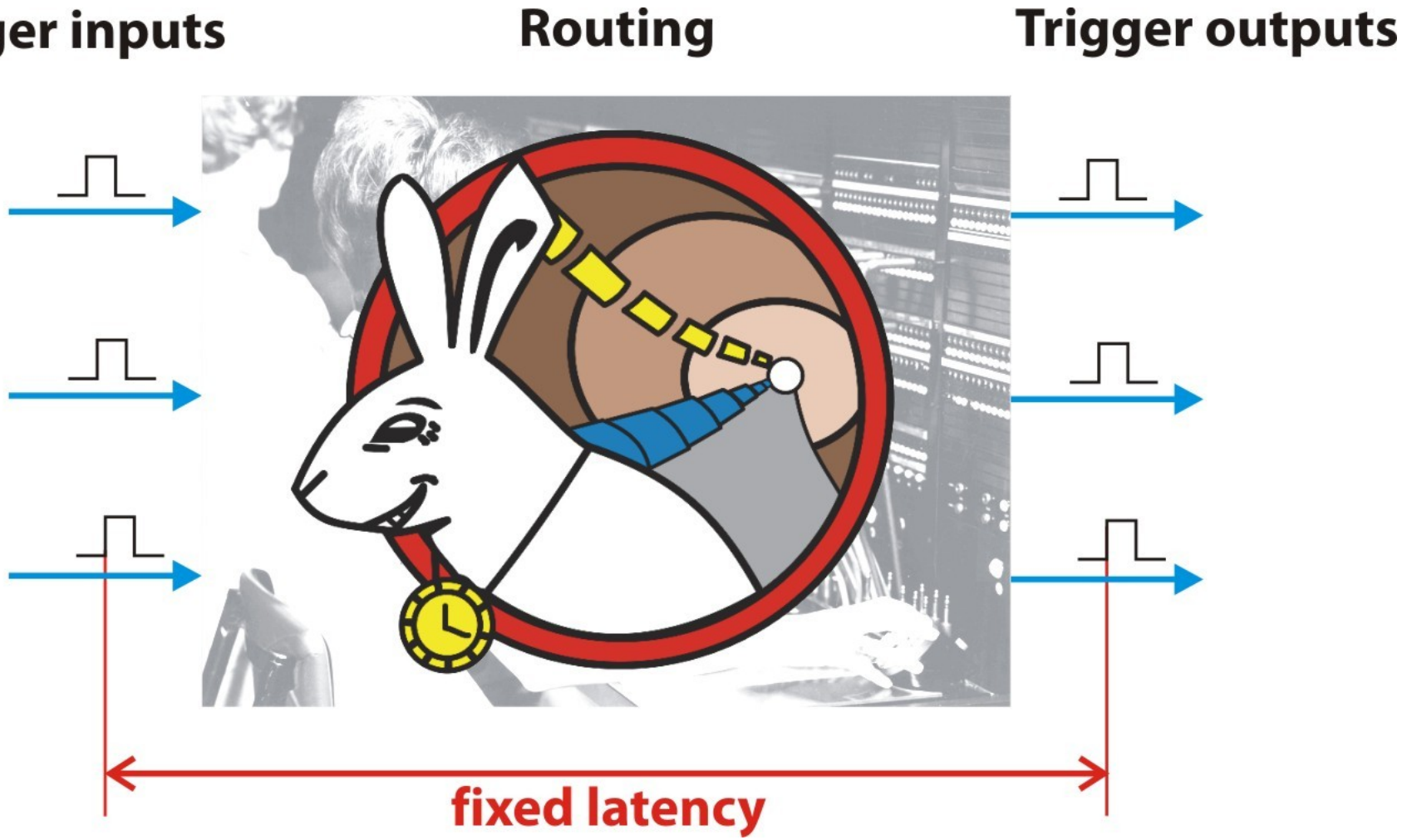
Applications @ CERN

- LHC Instability Trigger Distribution (a.k.a. LIST) – a flexible trigger distribution system
- Future CERN timing master & receiver
- GMT from/to WR conversion
- Distributed DDS (RF over WR) node

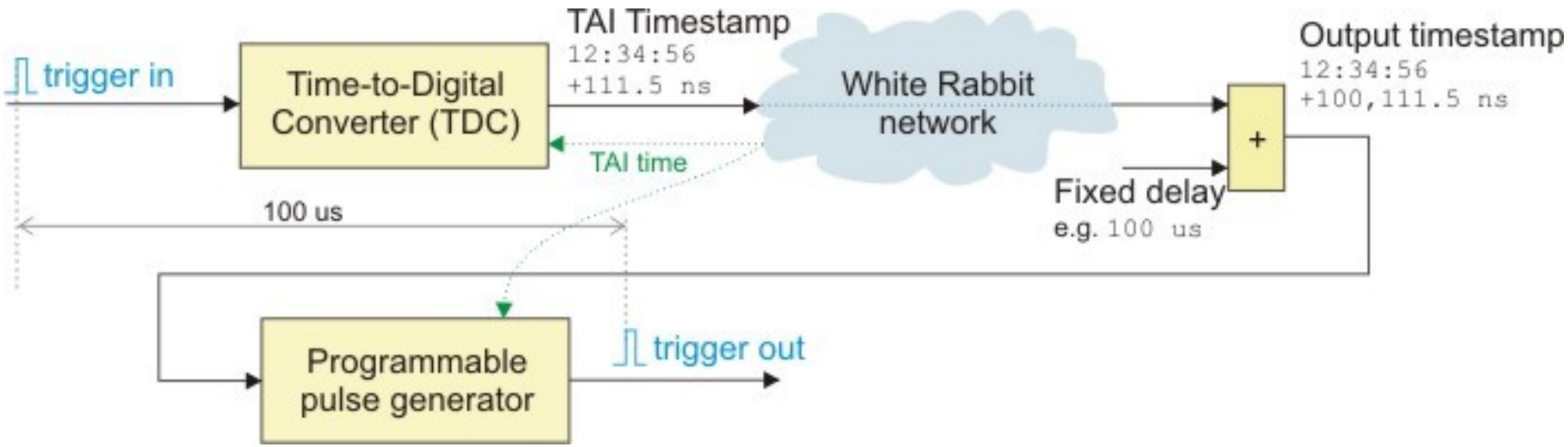
LIST project

Trigger inputs **Routing** **Trigger outputs**

LIST project



Trigger distribution idea



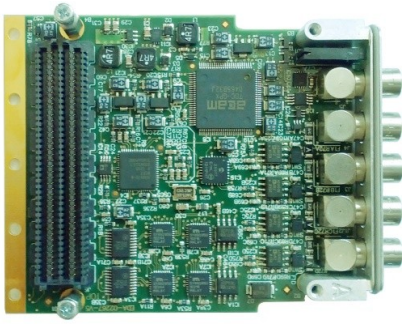
- A trigger pulse comes in and gets timestamped.
- The timestamp is broadcast in a UDP packet with metadata identifying the trigger source.
- Any number of devices can subscribe to the trigger and reproduce it with a fixed delay thanks to network-wide synchronization provided by White Rabbit.

Functional requirements

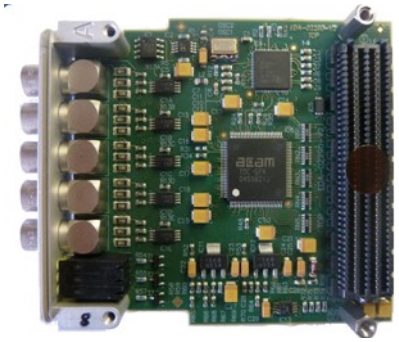
- **Accuracy**: < 1 ns network-wide, jitter < 100 ps rms.
- **Throughput**: 1 trigger every 80 us per each input/output (capable of recovering the *FRev* transmitted as a series of pulses).
- **Latency:** < 270 us
- **Delay** configurable independently for each in/out.
- Each output can subscribe to one or more triggers.
- **Logging** of each sent and executed trigger.
- Standard network **diagnostic tools** (Wireshark).

LIST hardware

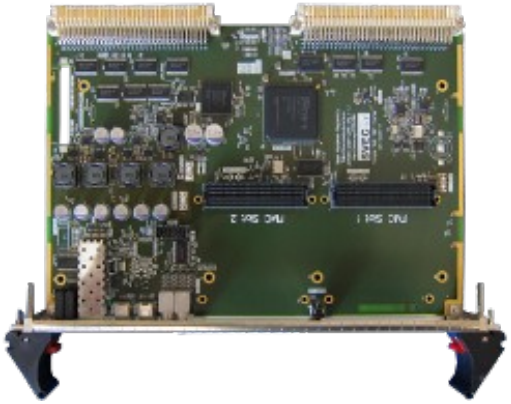
FMC Fine Delay
(trigger output)



FMC TDC
(trigger input)



Simple VME64x FMC carrier
(VME host for mezzanines)

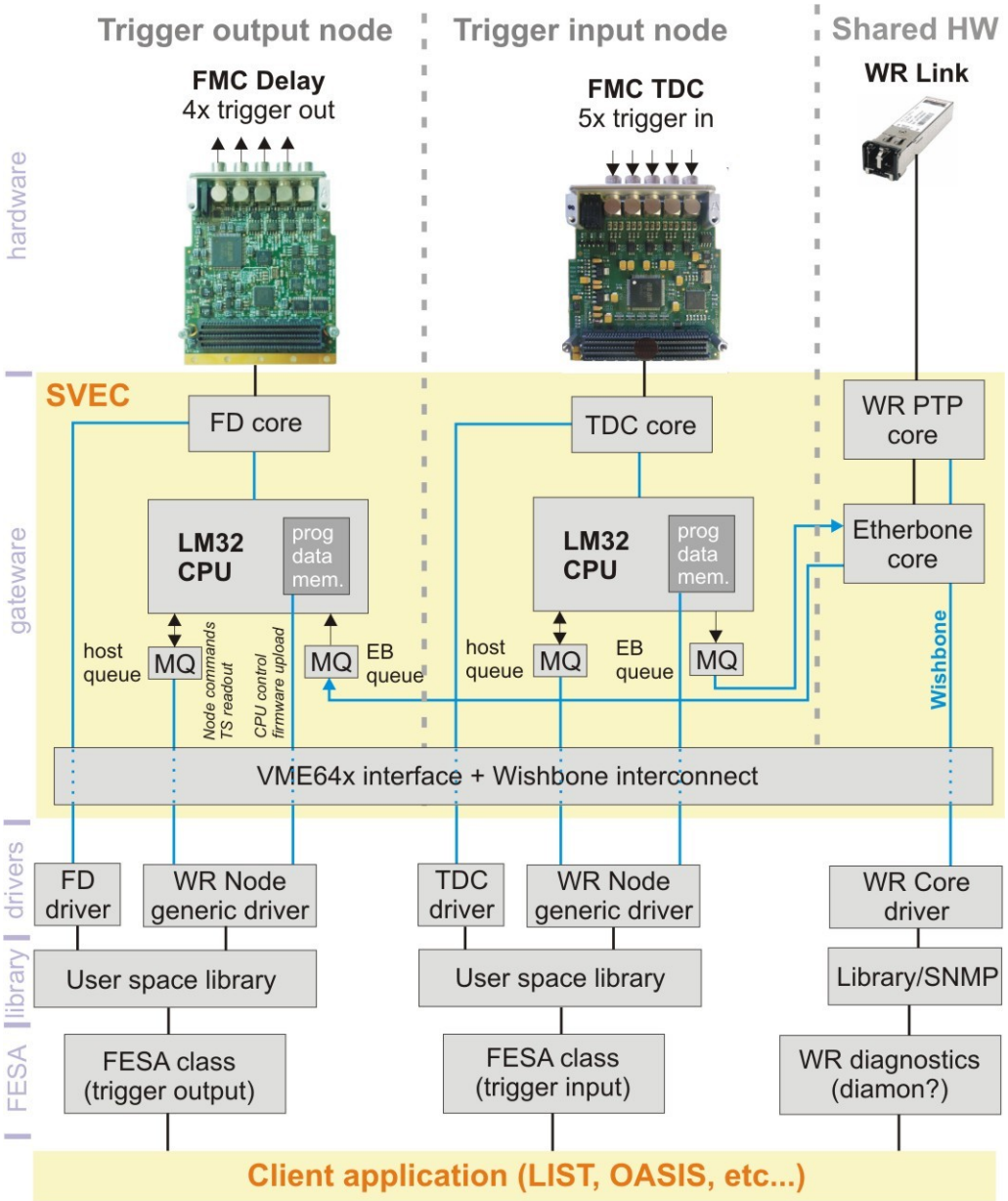


White Rabbit Switch V3.4

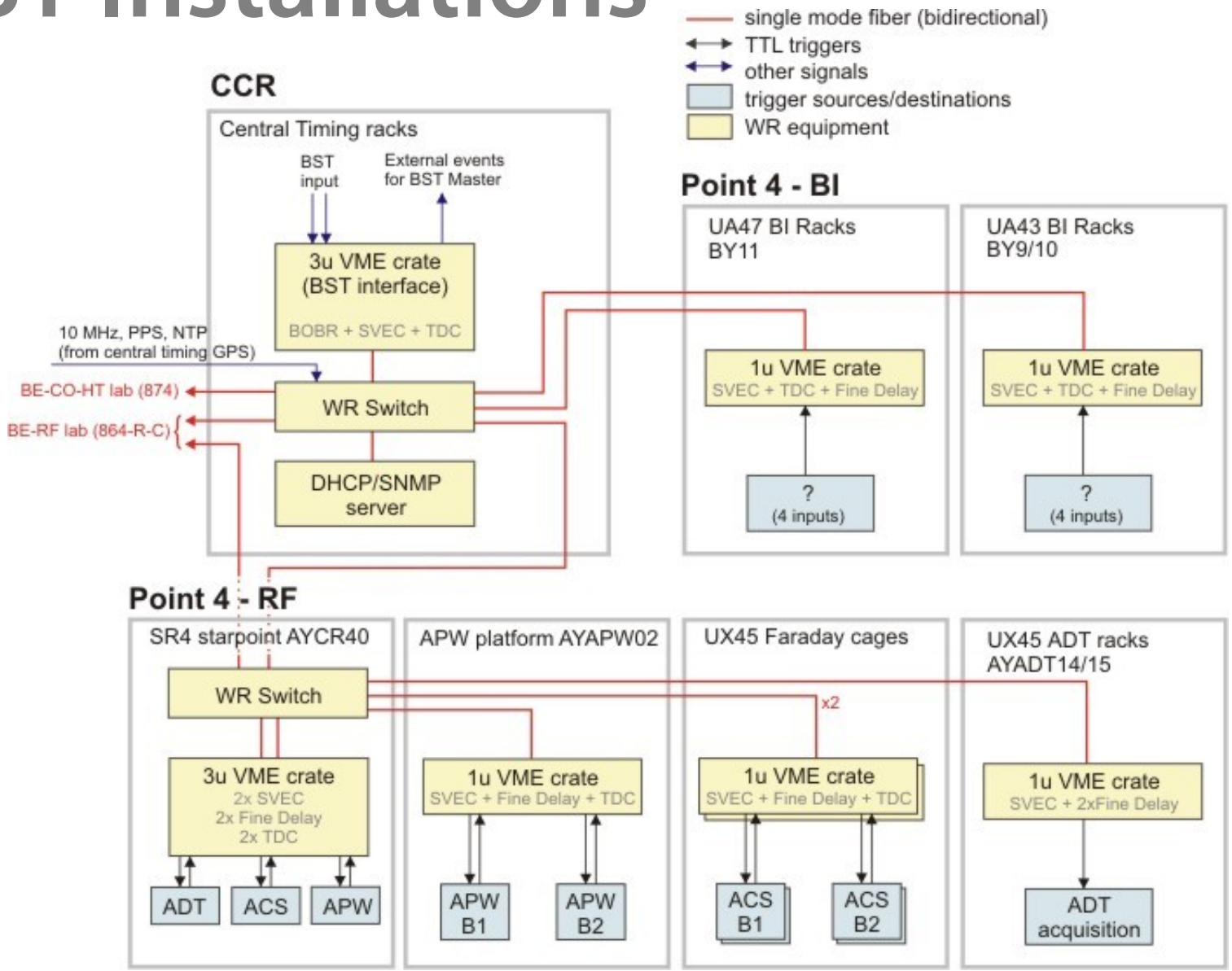
- Based on the BE-CO-HT's standard FMC Hardware Kit (VME).
- Hardware already available & verified.

LIST WR node

- 2 CPU cores
- Core 1 in charge of the outputs (FMC Fine Delay as the DP)
- Core 2 handling trigger inputs (FMC TDC as the DP)
- 2 HMQ slots per core (1 for control, 1 for logging)
- 2 RMQ slots
- Shared Memory for loopback



LIST Installations



Status and plans

- Initial WR node HDL design available (missing some features)
- Drivers currently under development
- LIST: nodes working in the lab, HW installed in the LHC and CCR
- LIST operation planned end of Q1 2015
- Test drive with distributed DDS towards end of 2014

Questions?