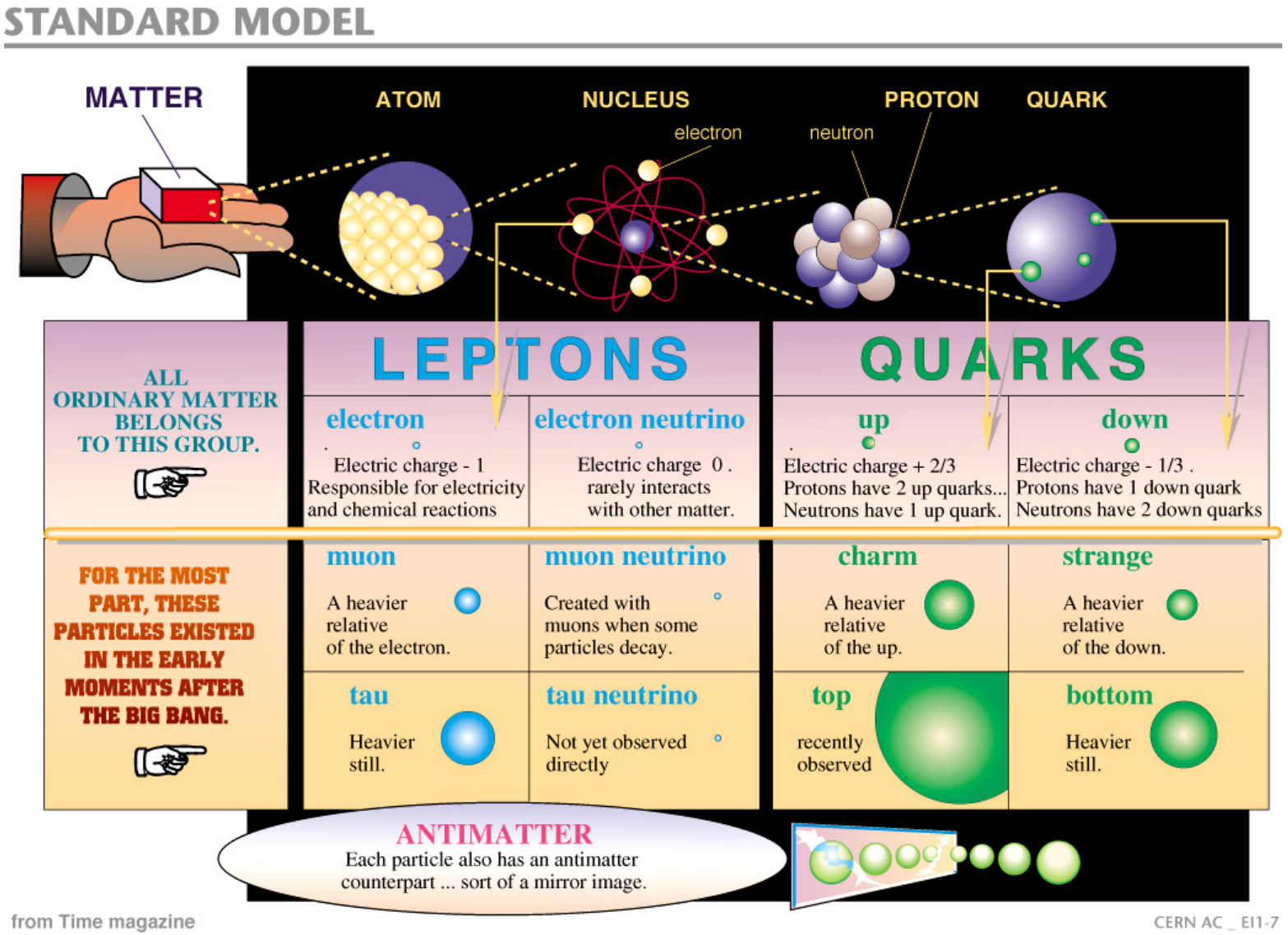
A strategy for managing diverse equipment in the CERN controls group

Javier Serrano, Juan David González Cobas

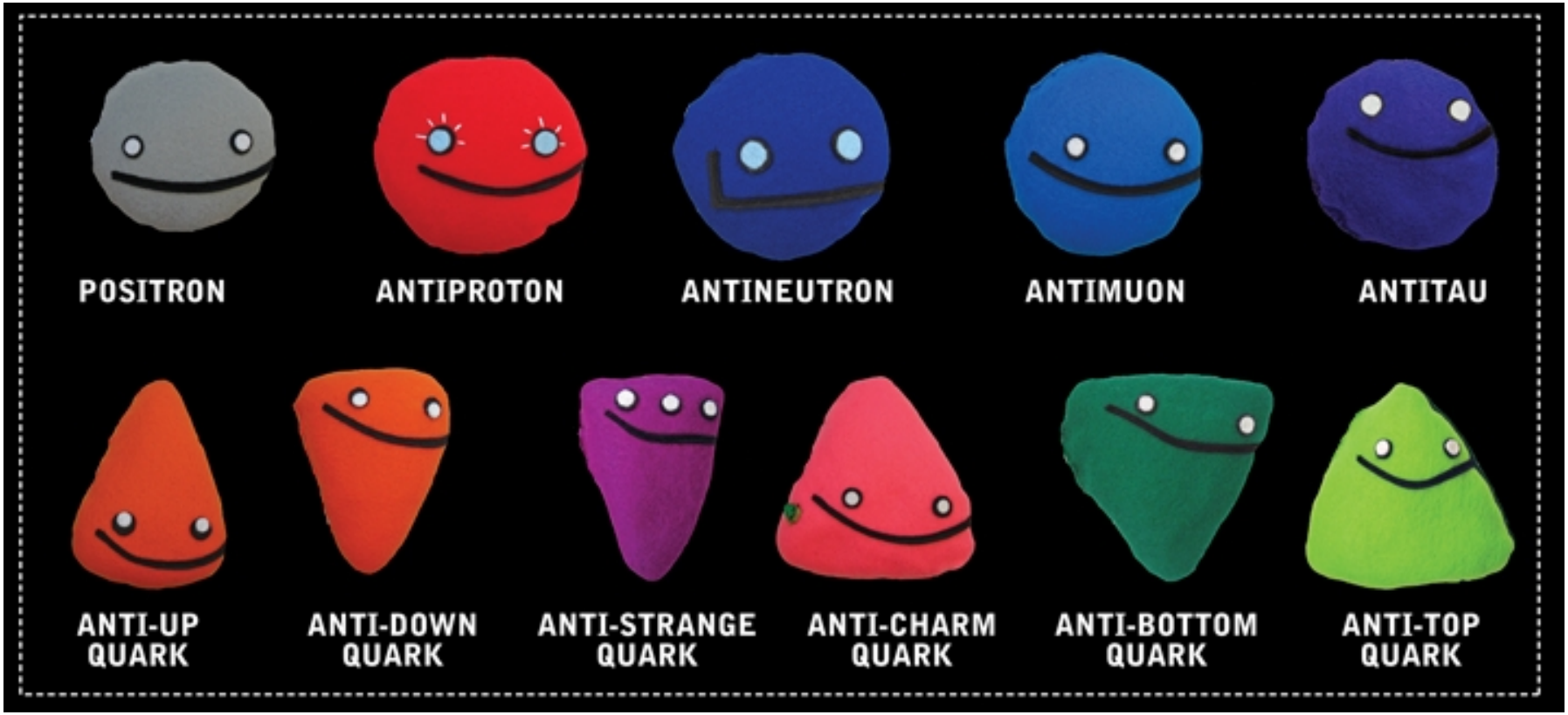
FOSDEM 2012

Outline

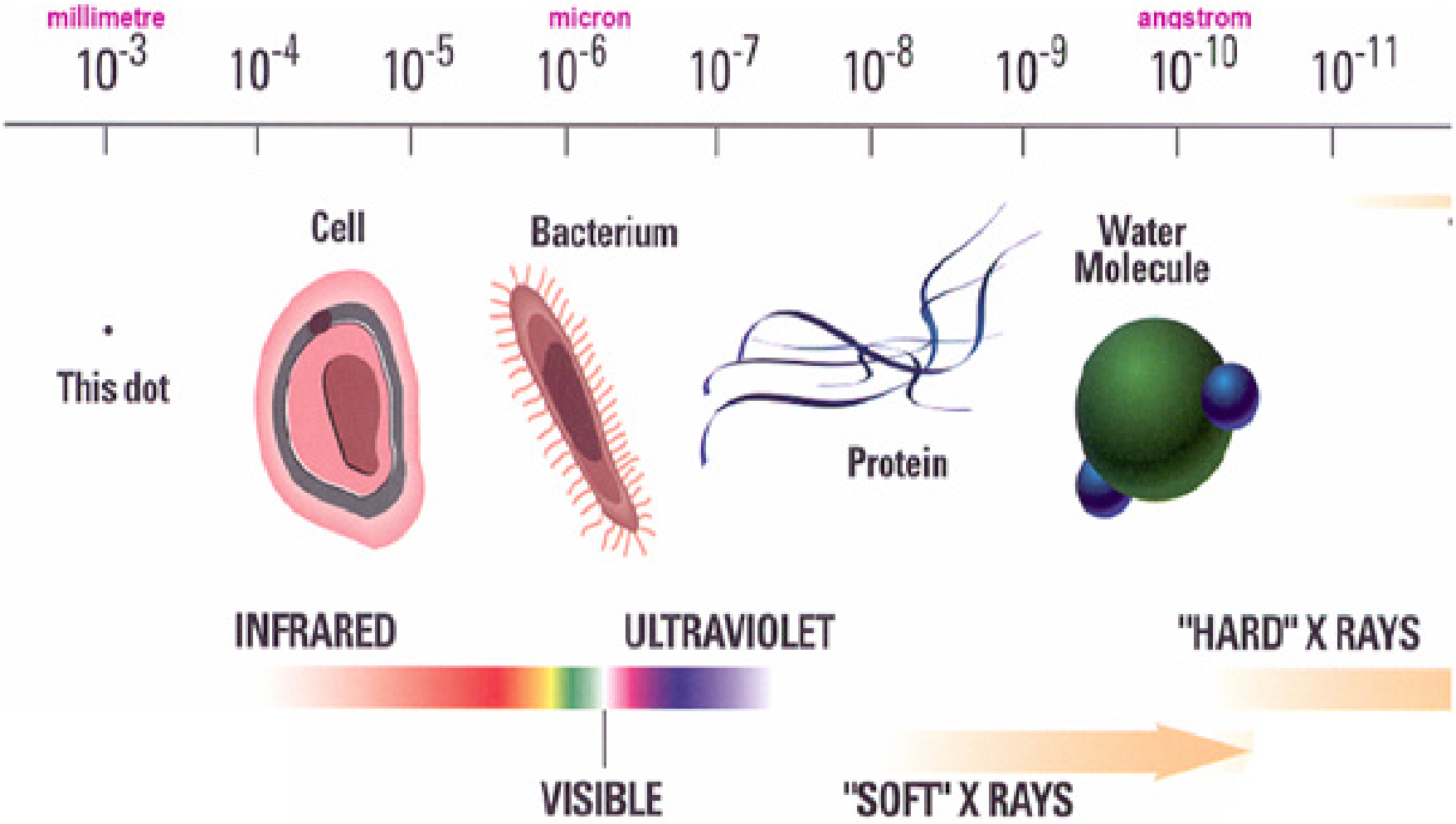
Find out about what the world is made of...



. . . and what other worlds might be made of!

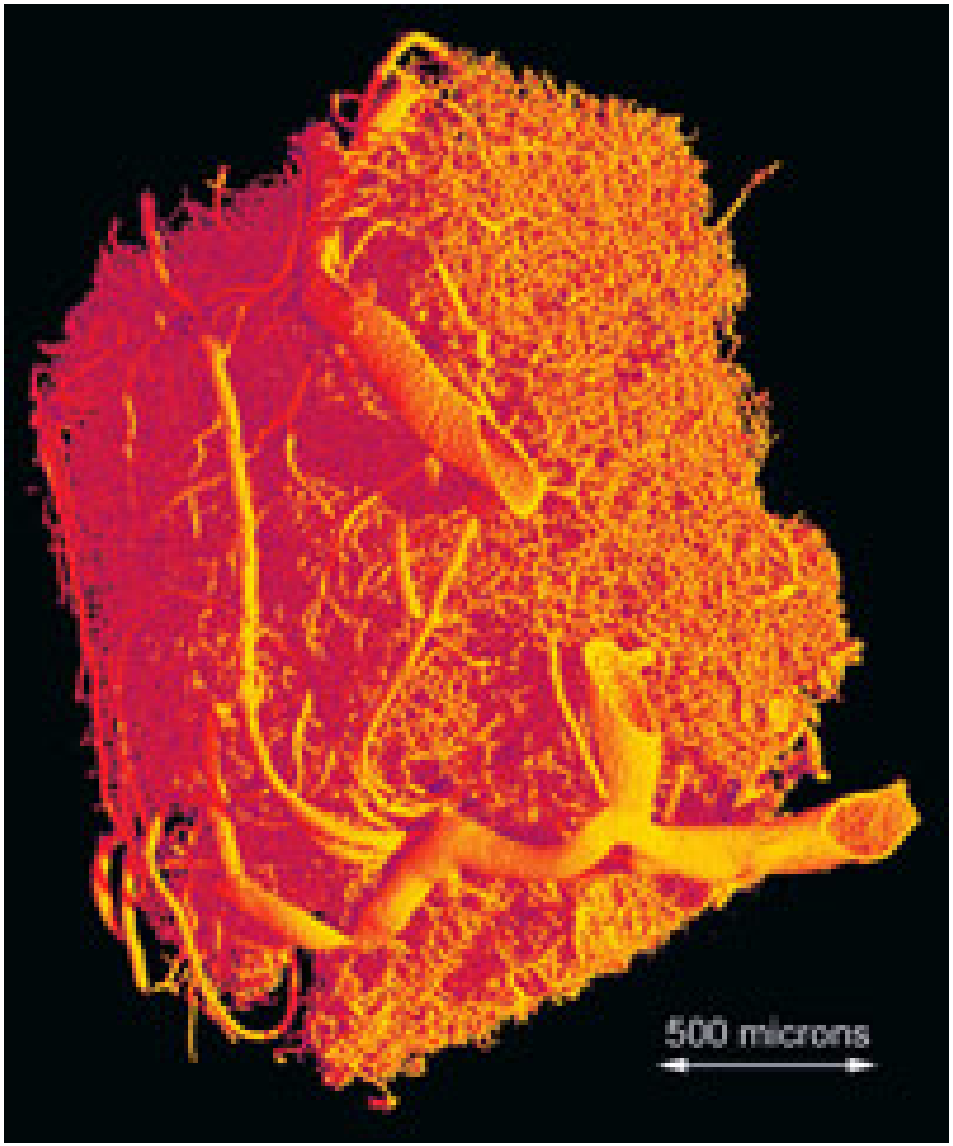


Better microscopes for biologists and other scientists



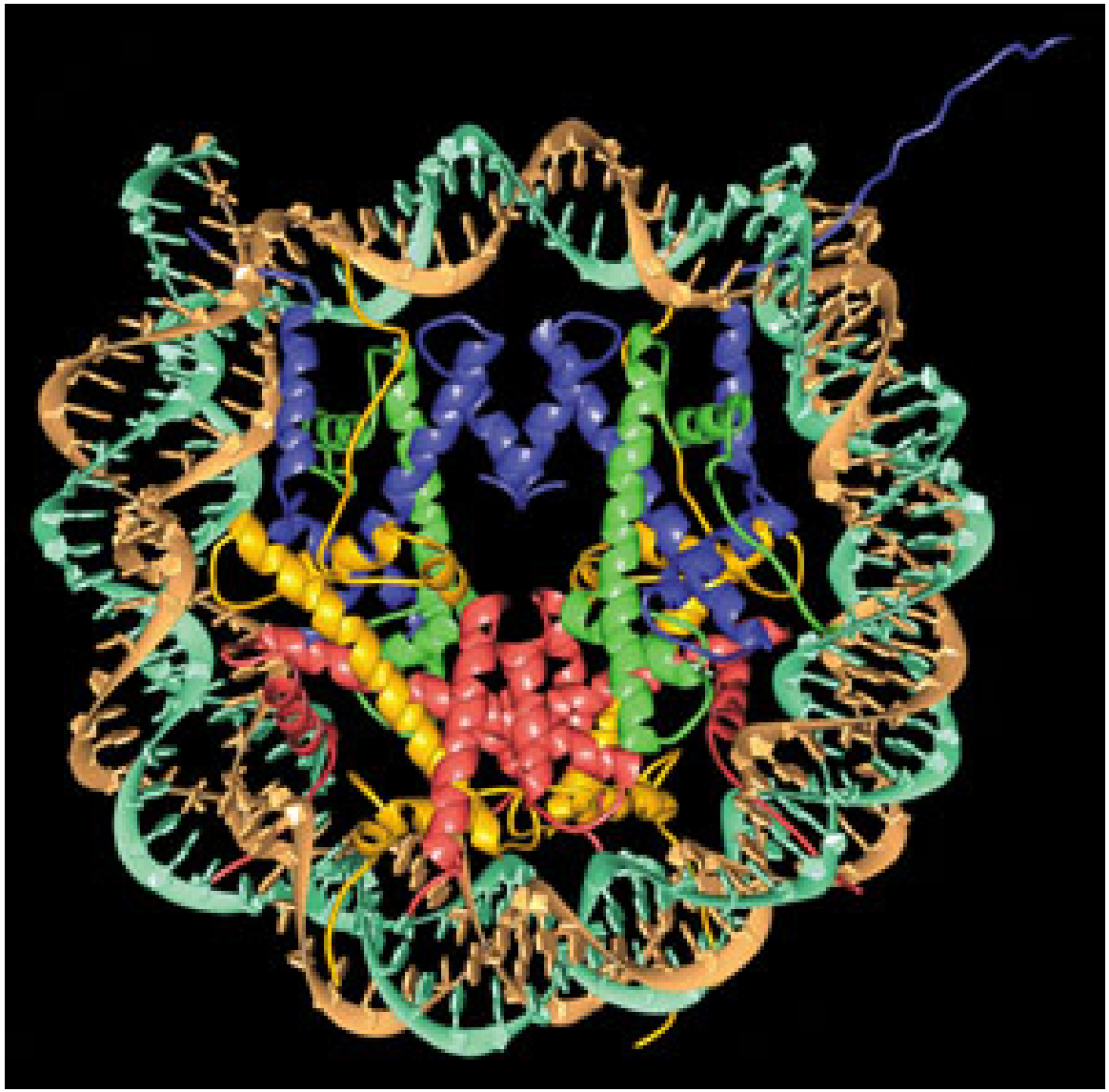
Better microscopes for biologists and other scientists

E.g. Mouse brain to study Alzheimer



Better microscopes for biologists and other scientists

E.g. Nucleosome core

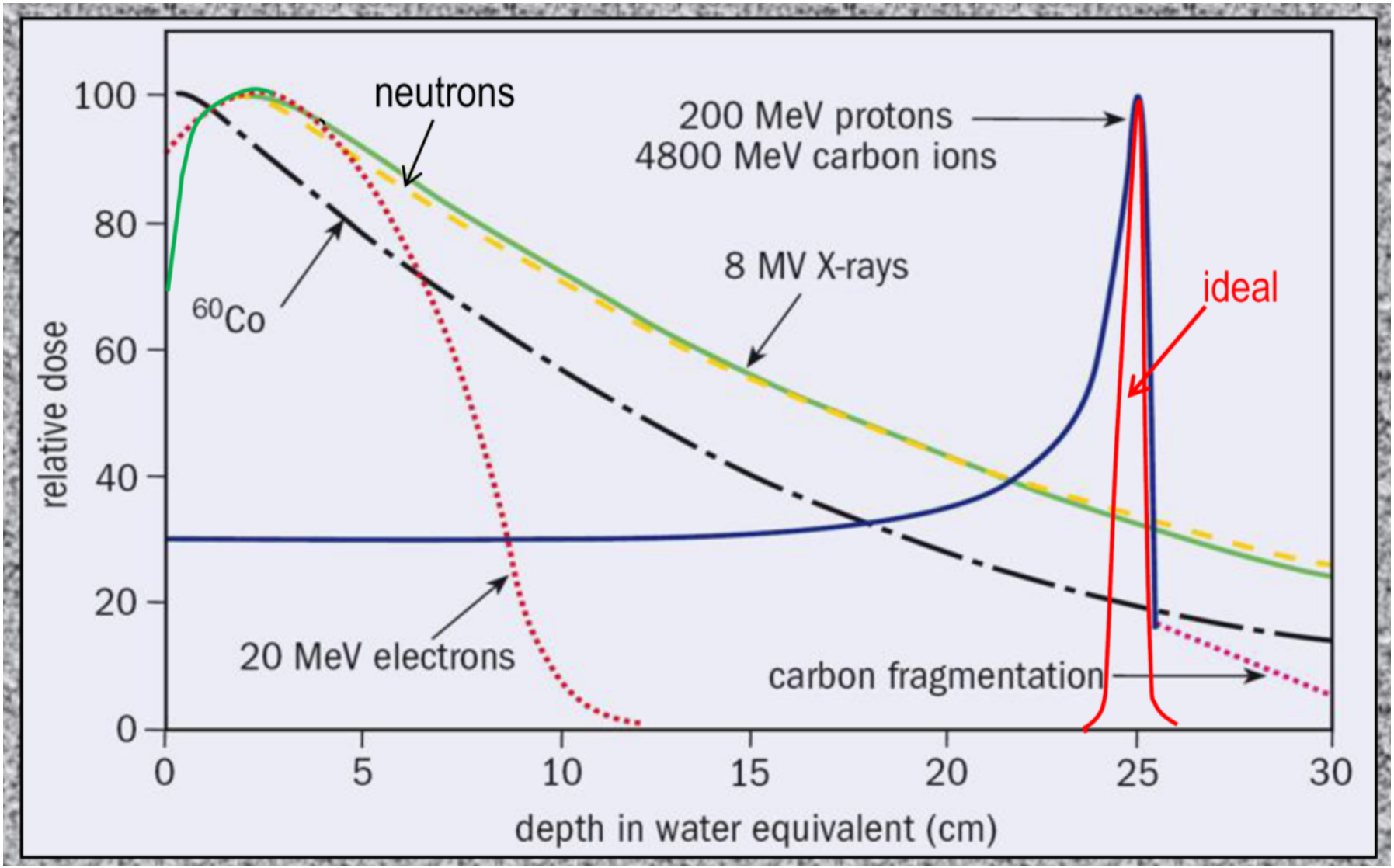


Fight tumors more effectively

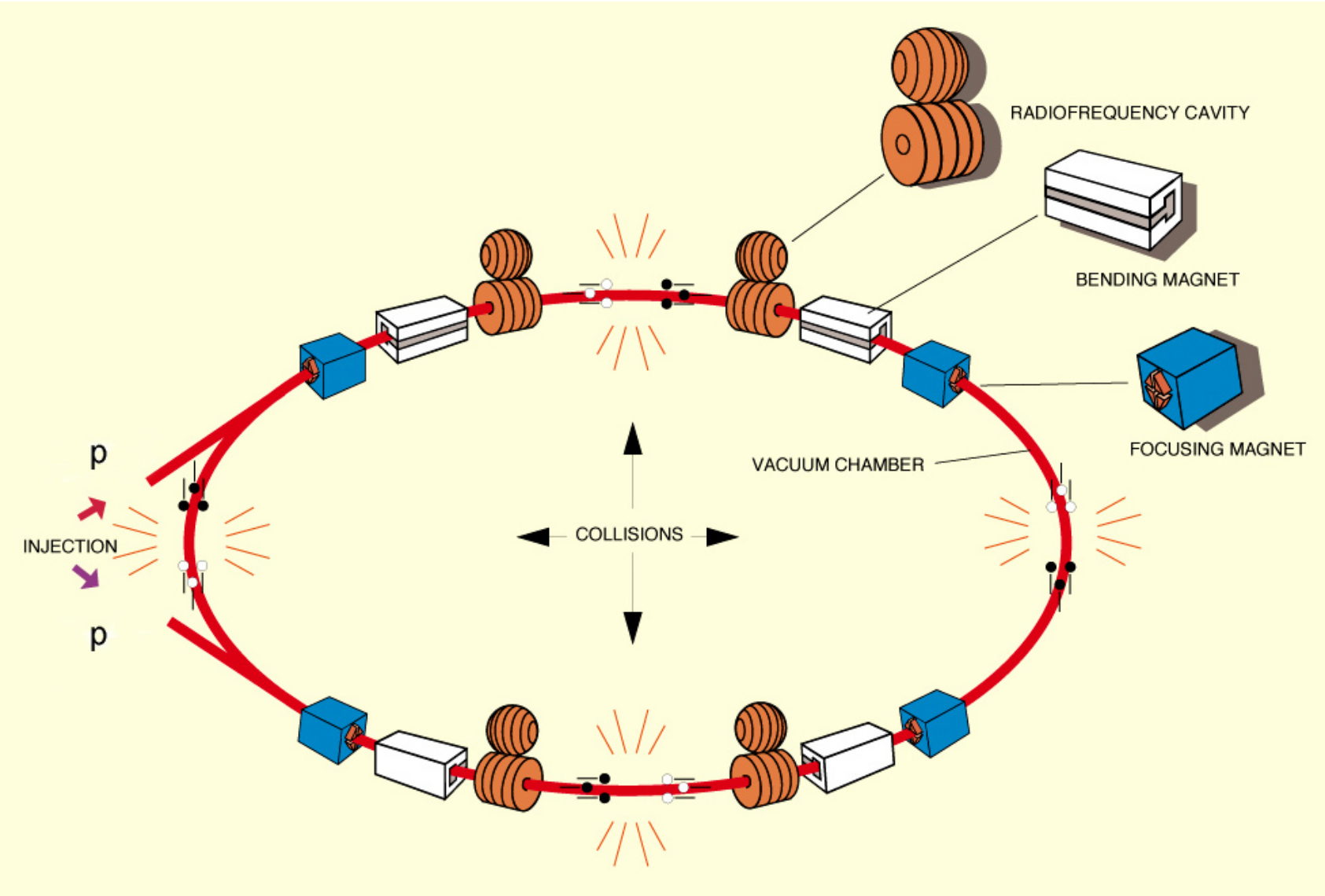
Hadron therapy

Fight tumors more effectively

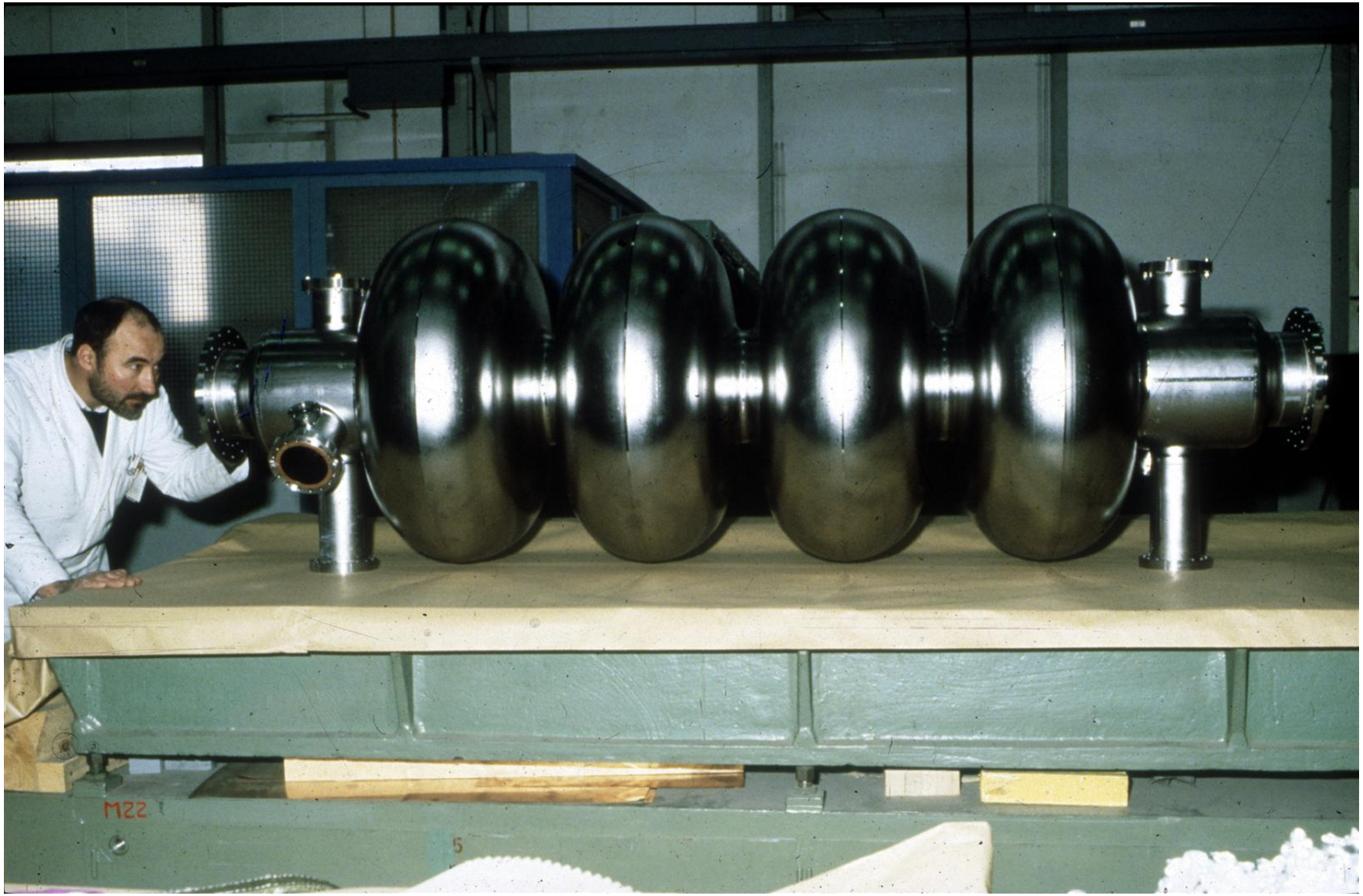
Better energy deposition than X-rays and traditional radiotherapy



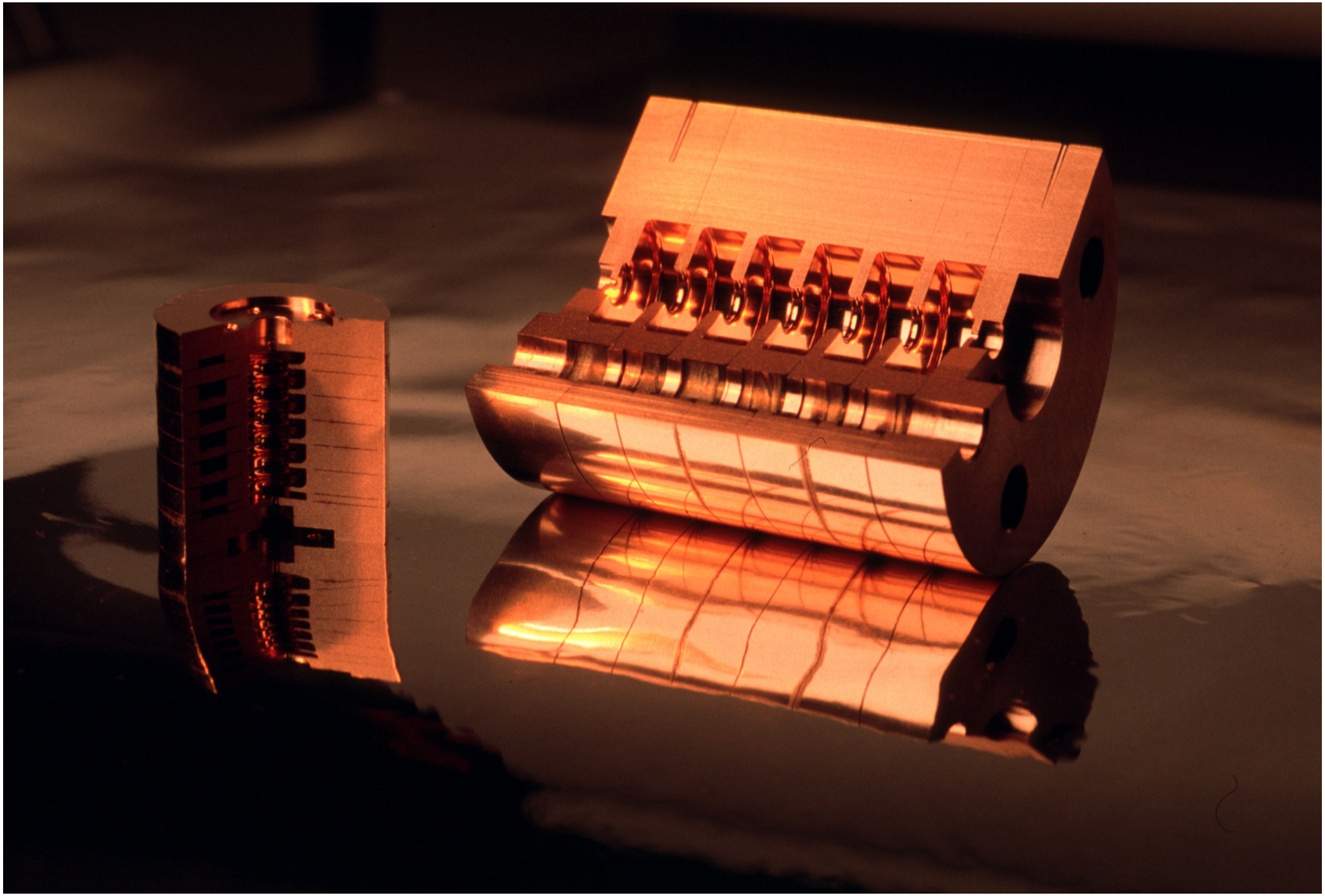
A simple synchrotron



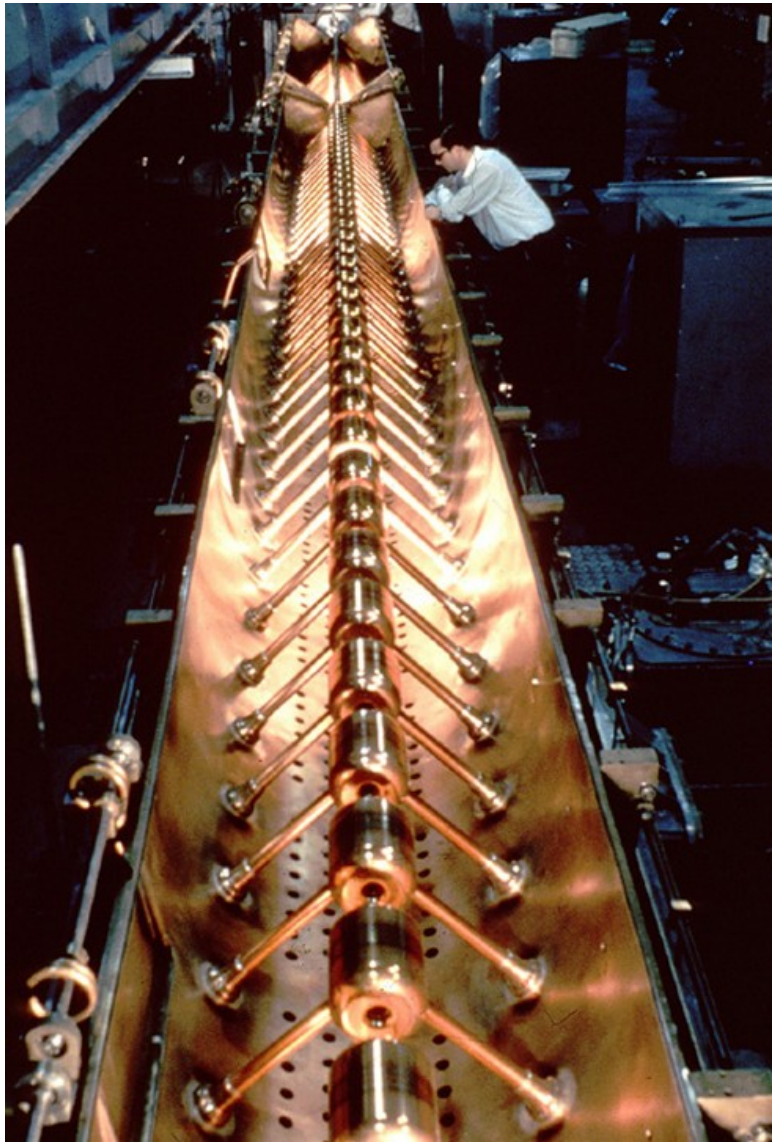
LEP superconducting cavity



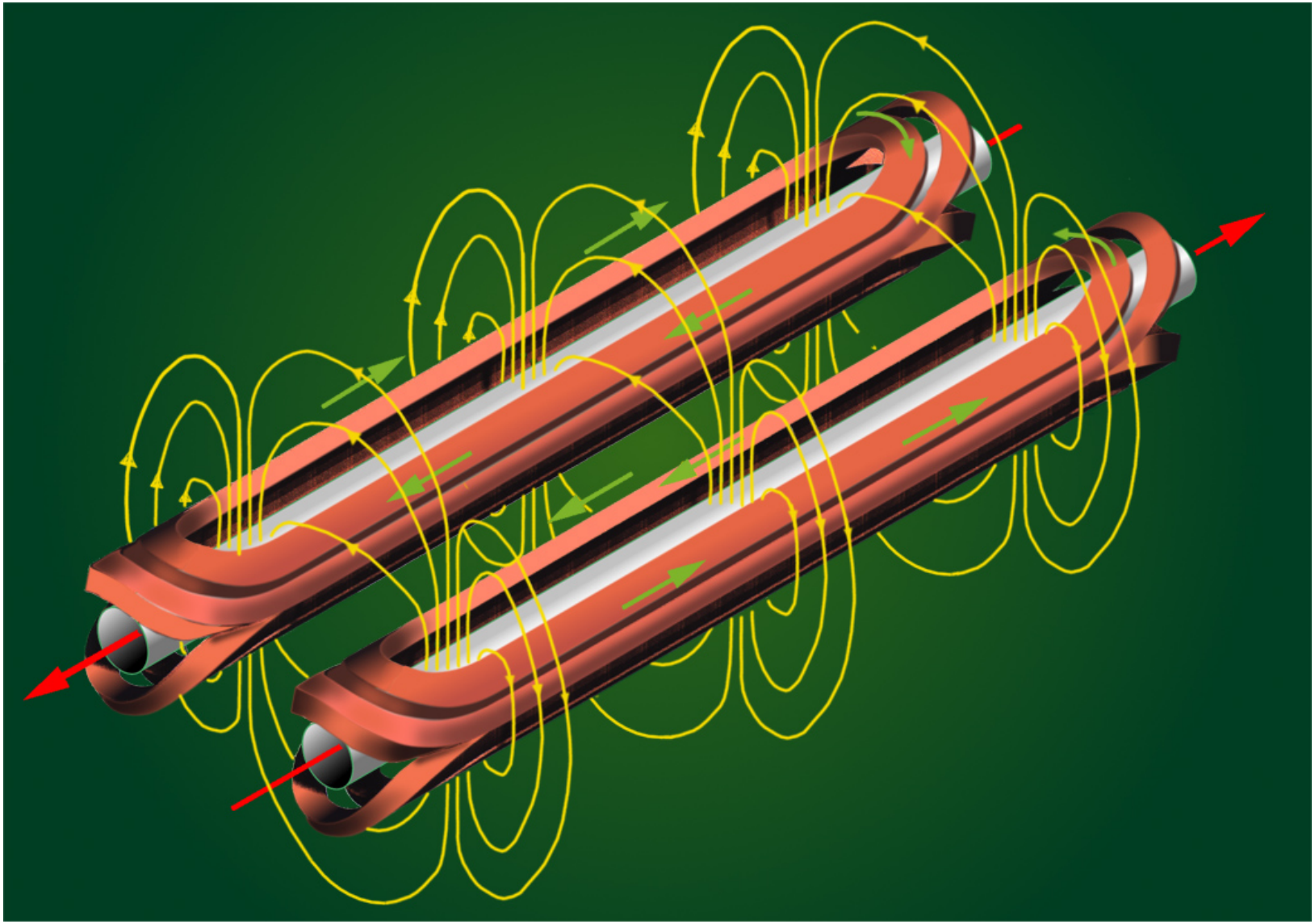
CLIC cavity



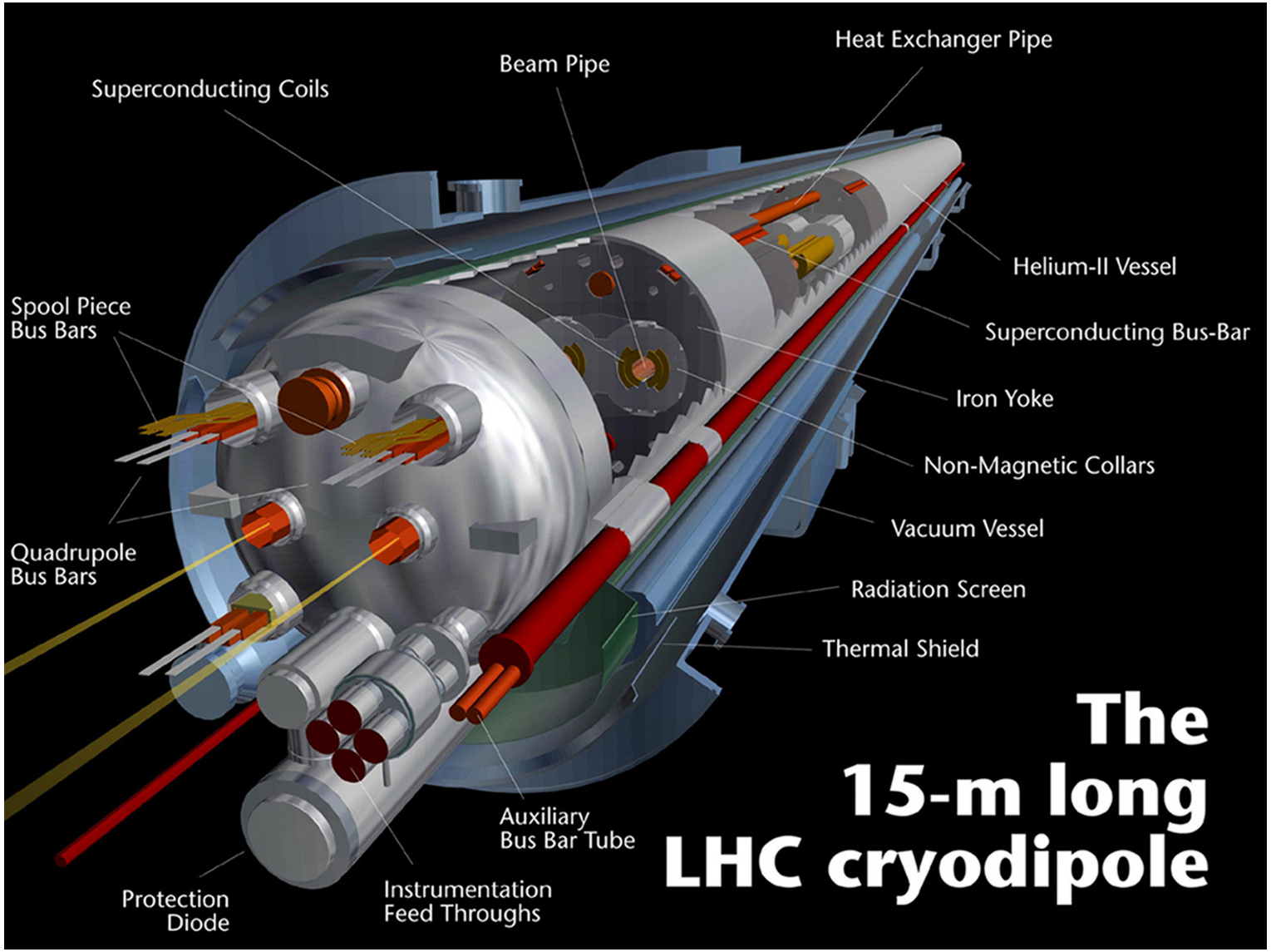
Drift Tube Linac (DTL)



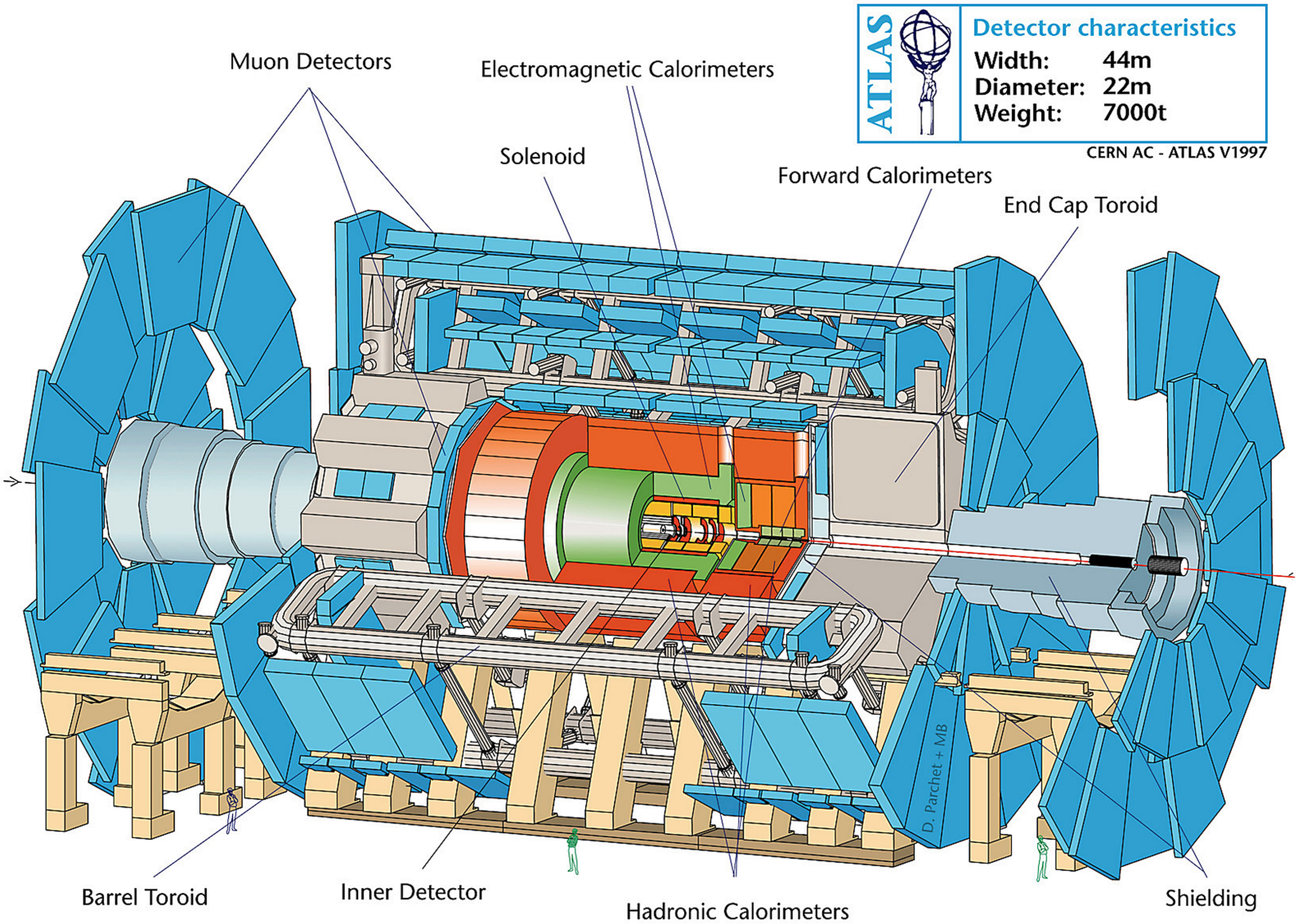
A simple dipole electromagnet



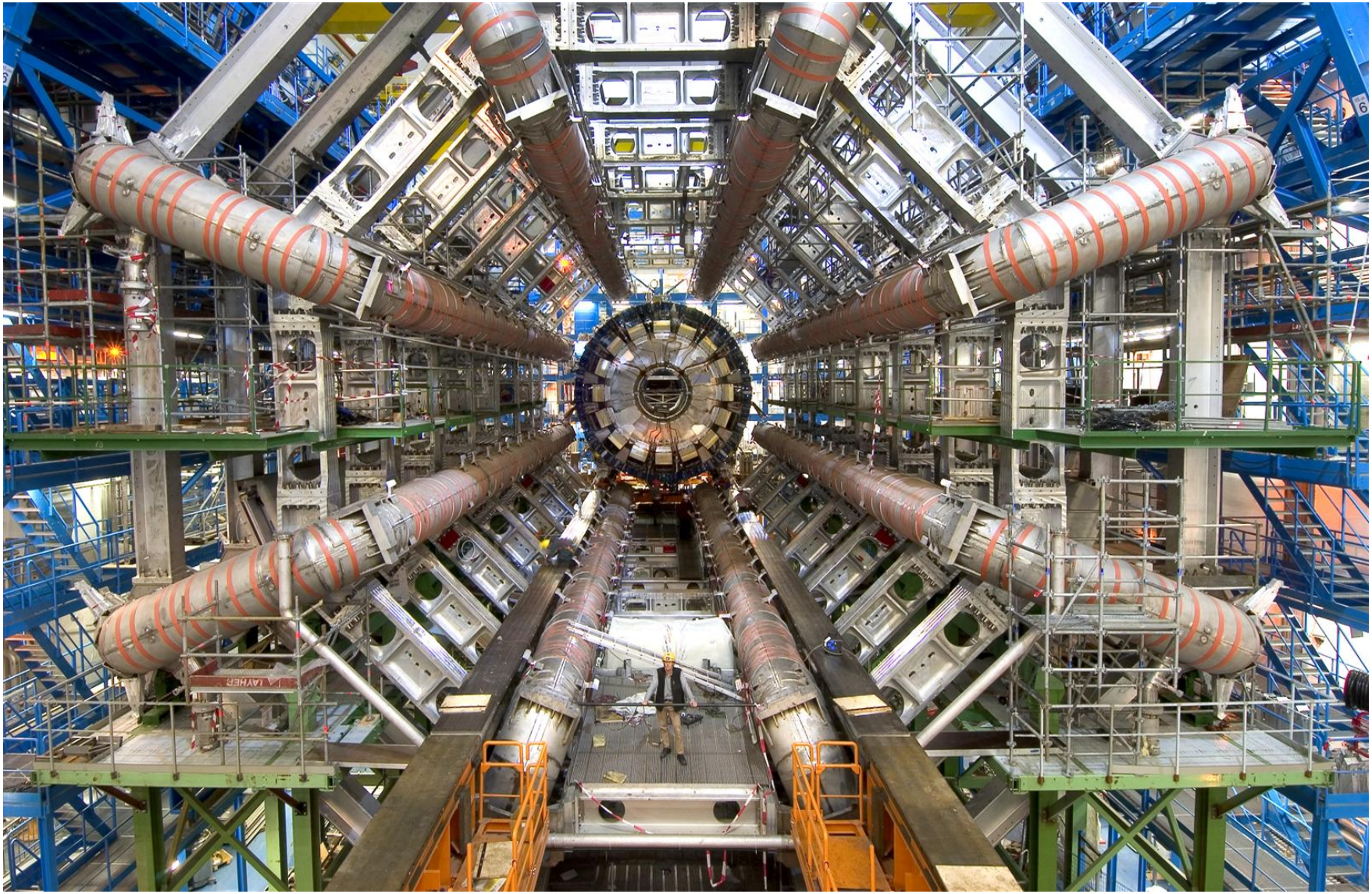
LHC cryodipole



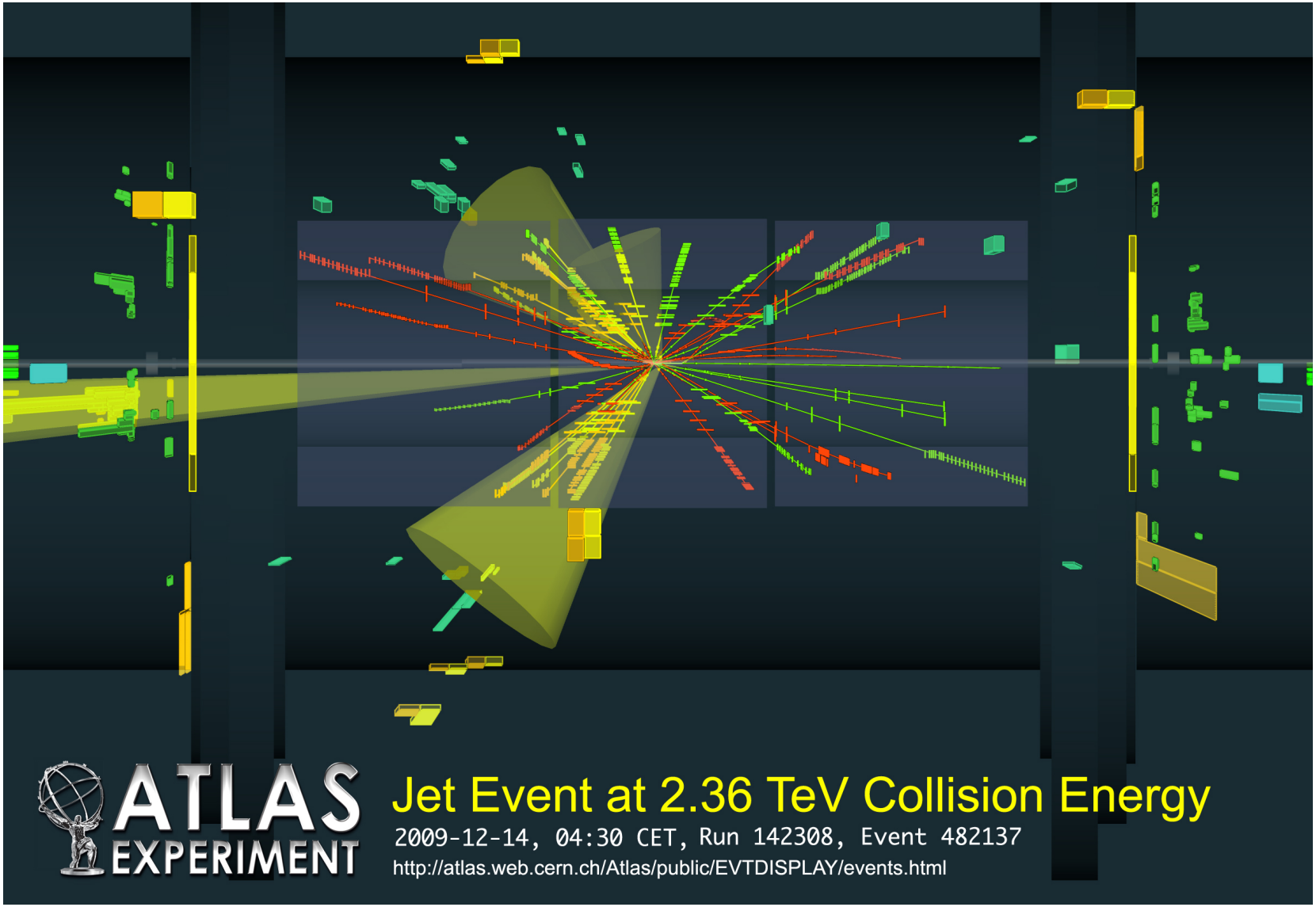
ATLAS on paper



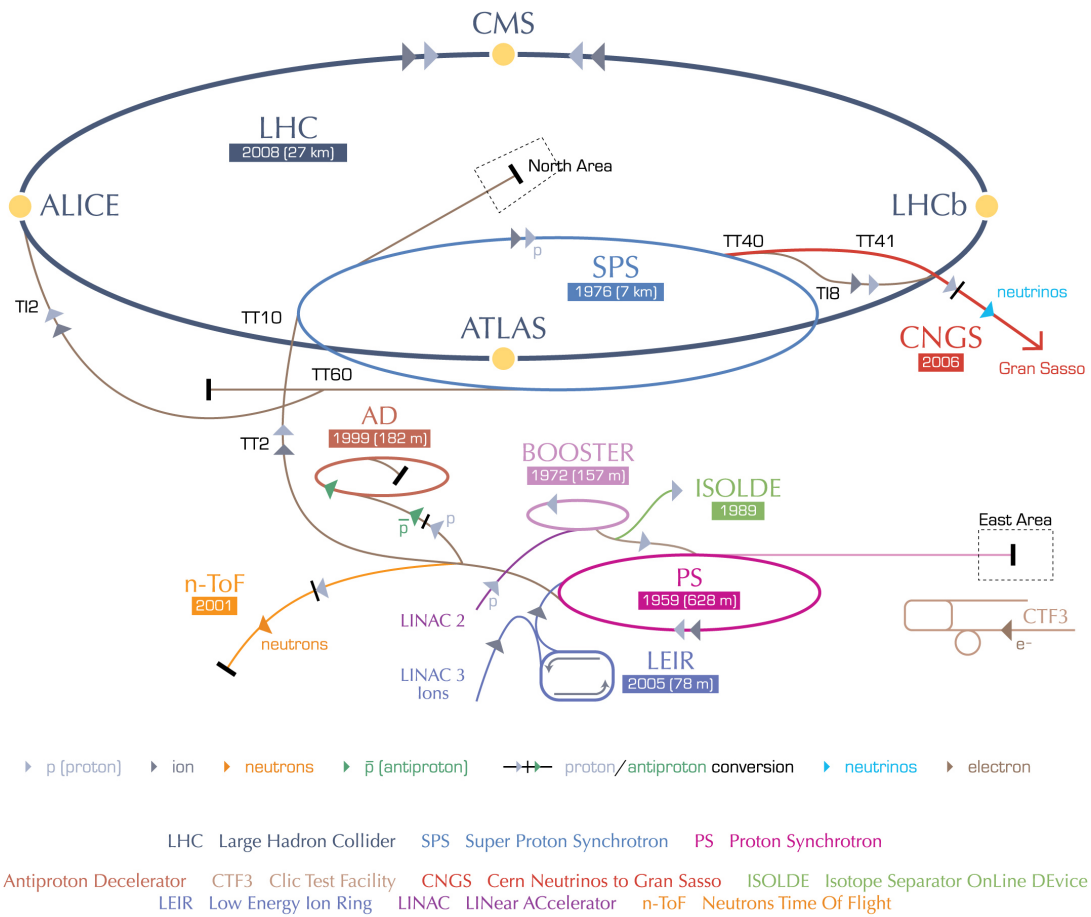
ATLAS in reality



ATLAS event example



CERN Accelerator Complex



European Organization for Nuclear Research | Organisation européenne pour la recherche nucléaire

CERN Beams Controls group

Responsible for

- Controls infrastructure for all CERN accelerators, transfer lines and experimental areas
- General services such as synchronization and analog signal acquisition/display
- Specification, design, procurement, integration, installation, commissioning and operation

Supports

- beam instrumentation, cryogenics, power converters etc.

Beams Controls standard kit

Hardware kit

- analog and digital I/O
- level converters, repeaters
- serial links, timing modules

Software

- Linux device drivers, C/C++ libraries, test programs

Bus standards for new designs

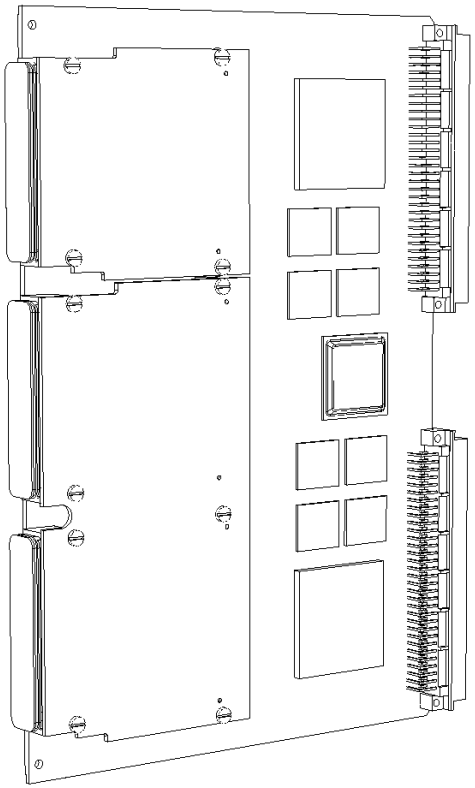
Two bus standards

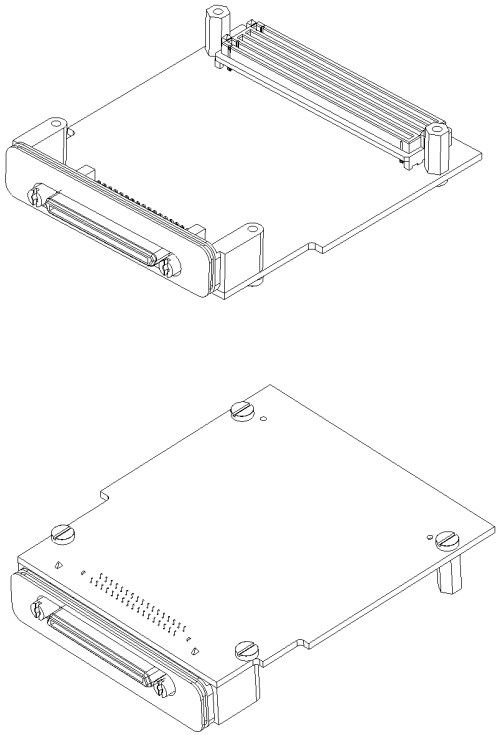
- VME64x
  - 6U, large front-panel space, may use rear transition module
- PICMG 1.3
  - Industrial type PC with the processor on a plug-in board
  - Internal buses PCI Express and PCI

Need for a mezzanine approach

- Functions (e.g. ADC, TDC) are needed for both buses
- Would need twice as many designs, more if additional standards are needed (PXIe, xTCA)

Carriers and mezzanines





Courtesy of VITA: `http://www.vita.com/fmc.html`

Advantages of the carrier/mezzanine approach

Re-use

- One mezzanine can be used in VME and PCIe carriers.
- People know standards, more likely to re-use or design for it.

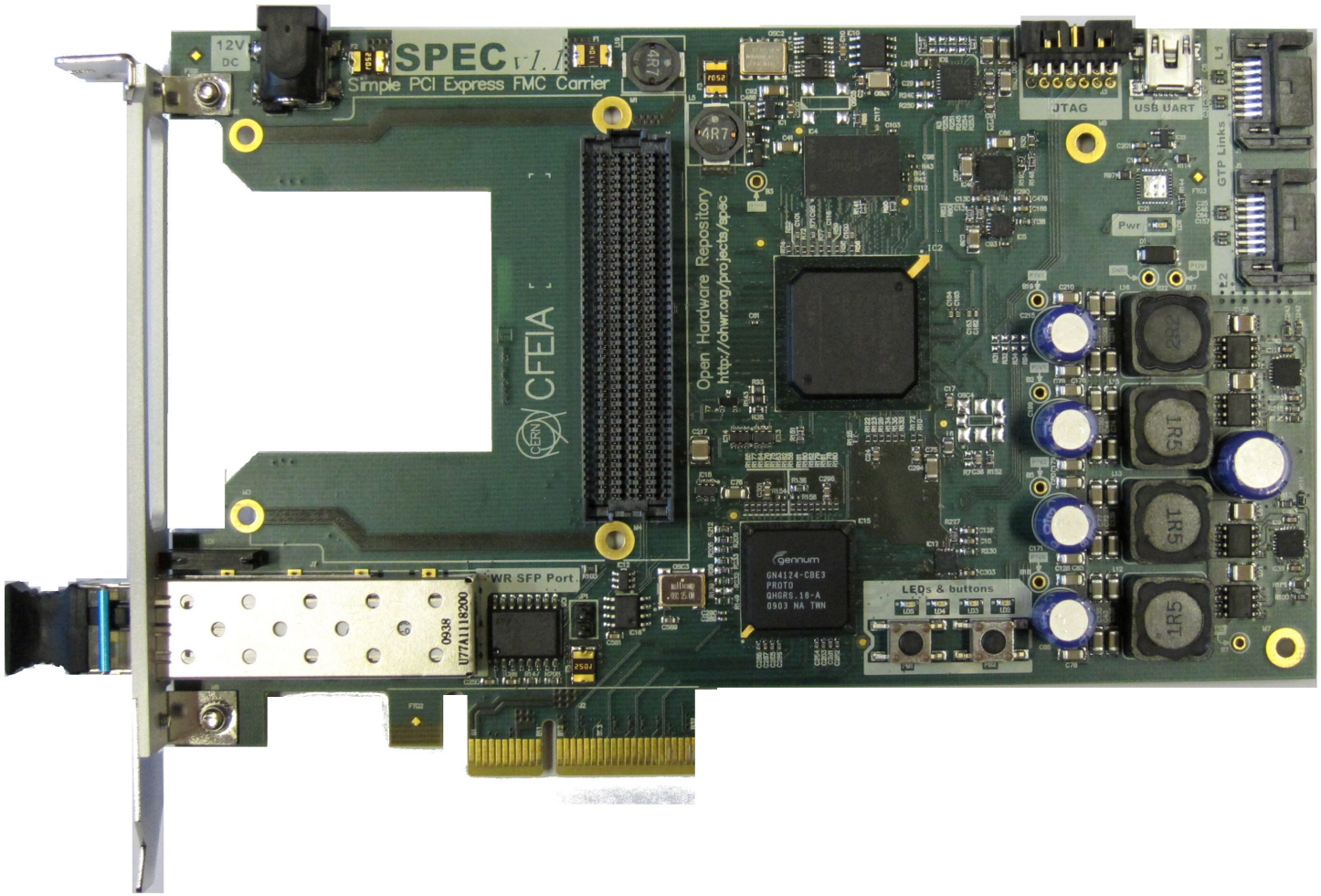
Reactivity

- Carrier: place and route a complex FPGA/Memory PCB once.
- Mezzanine: small and easier to route cards, easy assembly.

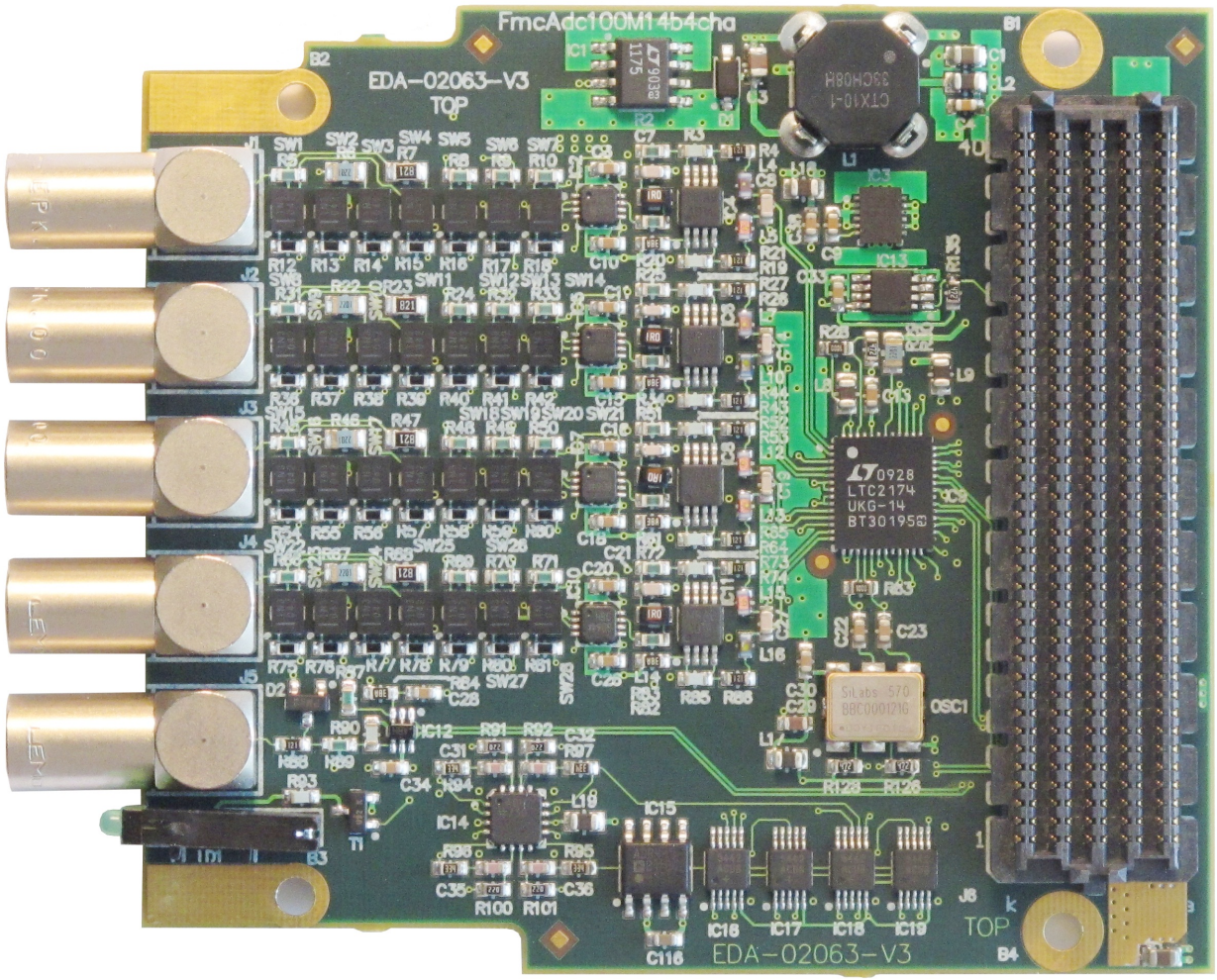
Rational split of work

'Controls' can design the carrier, 'Instrumentation' an ADC mezzanine, 'RF' a DDS one, etc.

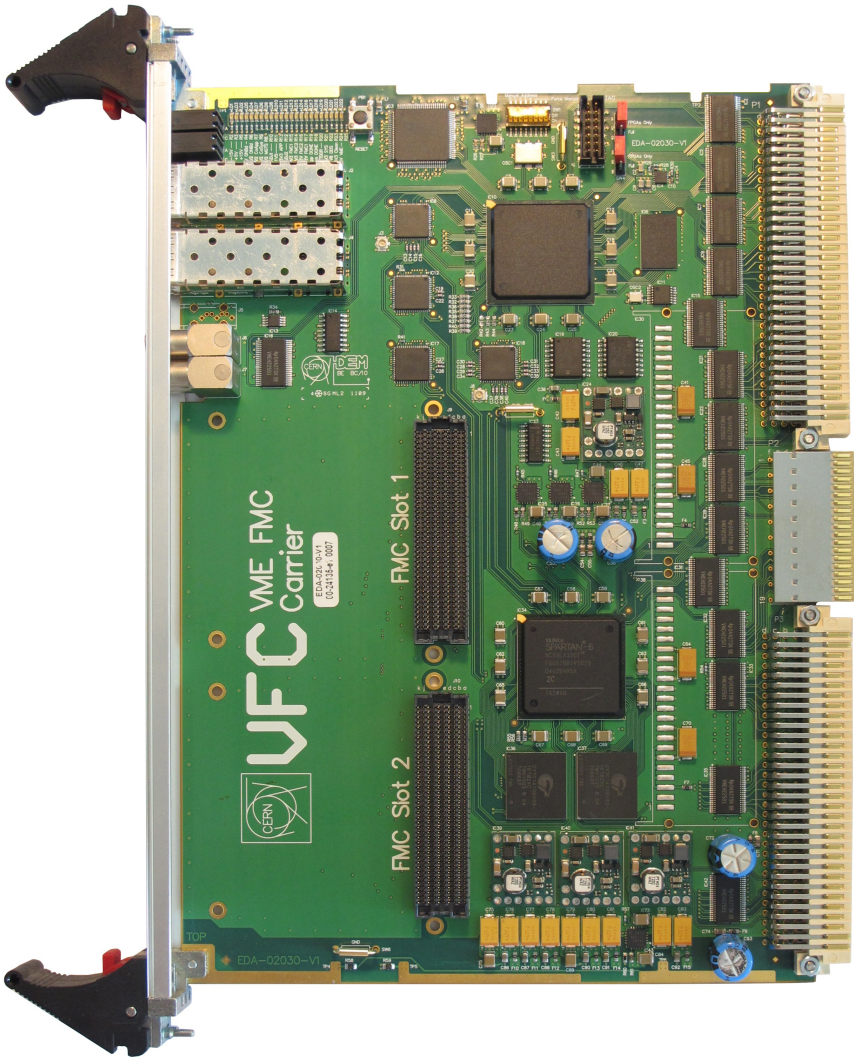
Example of a PCI Express FMC carrier (SPEC)



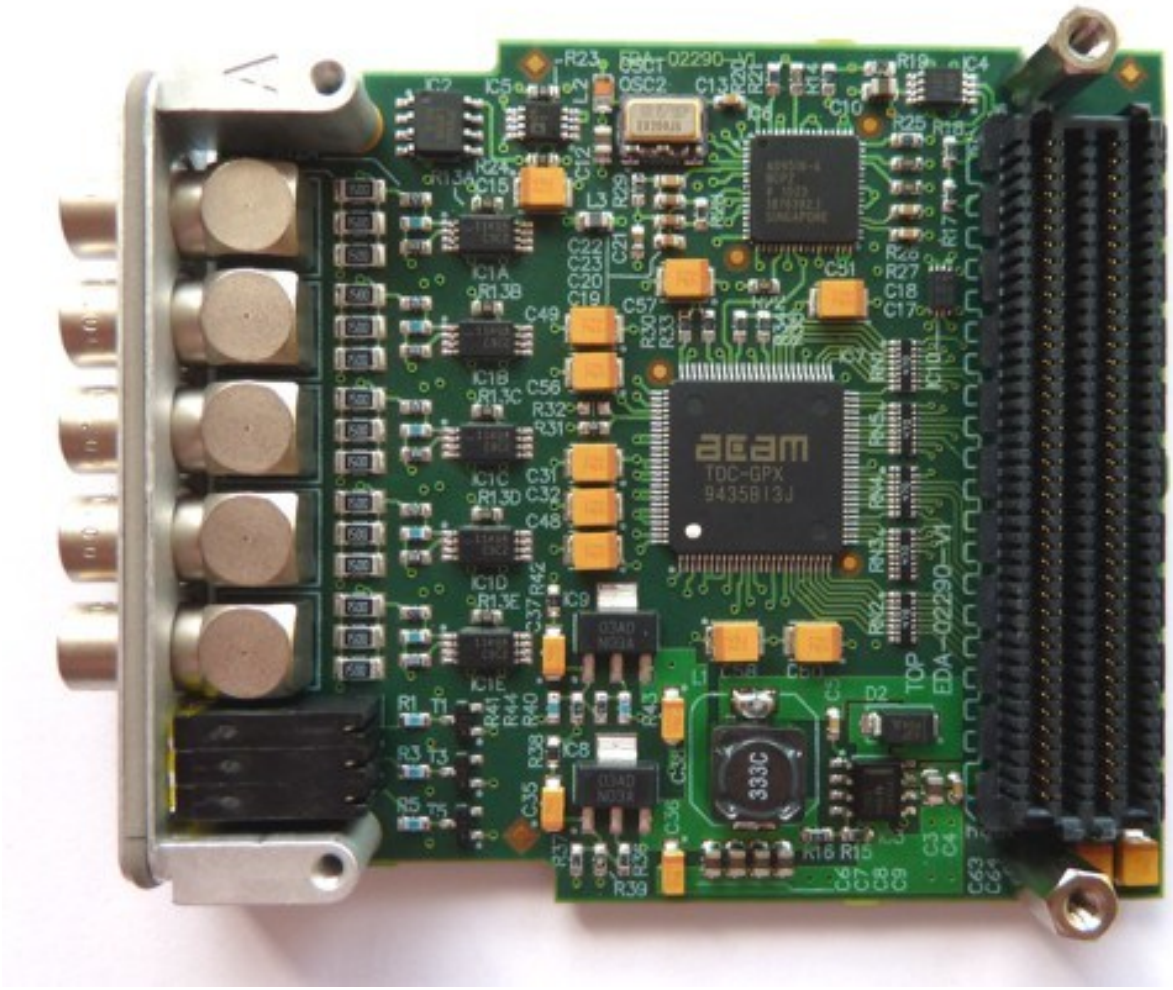
Example of FMC mezzanine: 100 MSPS 14-bit 4-channel ADC



VME64x FMC carrier



Another example of FMC mezzanine: 5-channel 1ns TDC



Inside the FPGA: Wishbone

- System becomes pretty complex: System-on-a-chip
- Build up from re-usable HDL cores
- Connect blocks with Wishbone bus
  - open standard
  - simple address/data bus
  - extended with pipelined mode
  - many cores already available
- We developed a design infrastructure
  - WBGen: scripts to automatically generate Wishbone slave HDL and documentation
  - SDWB: IP blocks with descriptors to enable software re-use
  - HDLMake: support to synthesize and simulate designs with distributed sources
  - Etherbone: a bridge between Ethernet and Wishbone (made by GSI)

There is an OSHW definition!

Check out http://freedomdefined.org/OSHW

- Inspired by the Open Source definition for software.
- Focuses on ensuring freedom to study, modify, distribute, make and sell designs or hardware based on those designs.
- Now we know exactly what we mean when we say OSHW!

Why we use Open Hardware

Peer review

Get your design reviewed by experts all around the world, including companies!

Design re-use

When it's Open, people are more likely to re-use it.

Healthier relationship with companies

No vendor-locked situations. Companies selected solely on the basis of technical excellence, good support and price.

Dissemination of knowledge

One of the core missions of CERN.

Open Hardware Repository – ohwr.org

A web-based collaborative tool for electronics designers

- Wiki, News
- File repository
- Issues management
- Mailing list

Fully open access

- All information readable by everyone, without registration

Server made itself of FOSS

- ChiliProject (a fork of Redmine)
- SVN/GIT for version management, integrated in OHR

CERN FMC projects in OHR – some examples

FMC Carriers

- VME64x, PCIe, AMC, VXS
- PXIe in the pipeline

FMC Mezzanines

- ADC's, sampling speeds: 200 kSPS, 100 MSPS
- TDC and Fine delay (resolution 1 ns)
- Digital I/O: 5 channels, 16 channels

Important: all of these are or will be **commercially** available.

The best of both worlds

	Commercial	Non-commercial
Open	Winning combination, best of both worlds.	Whole support burden falls on developers. Not scalable.
Propietary	Vendor lock-in.	?!

CERN Open Hardware License – ohwr.org/cernohl

Provides a solid legal basis

- Developed with Knowledge Transfer Group at CERN
- Inspired by FOSS licences
- Focused on both design and products
- Persistent as LGPL

Practical: makes it easier to work with others

- Upfront clear that anything you give will be available to everyone
- Makes it clear that anyone can use it for free

CERN Open Hardware License – ohwr.org/cernohl

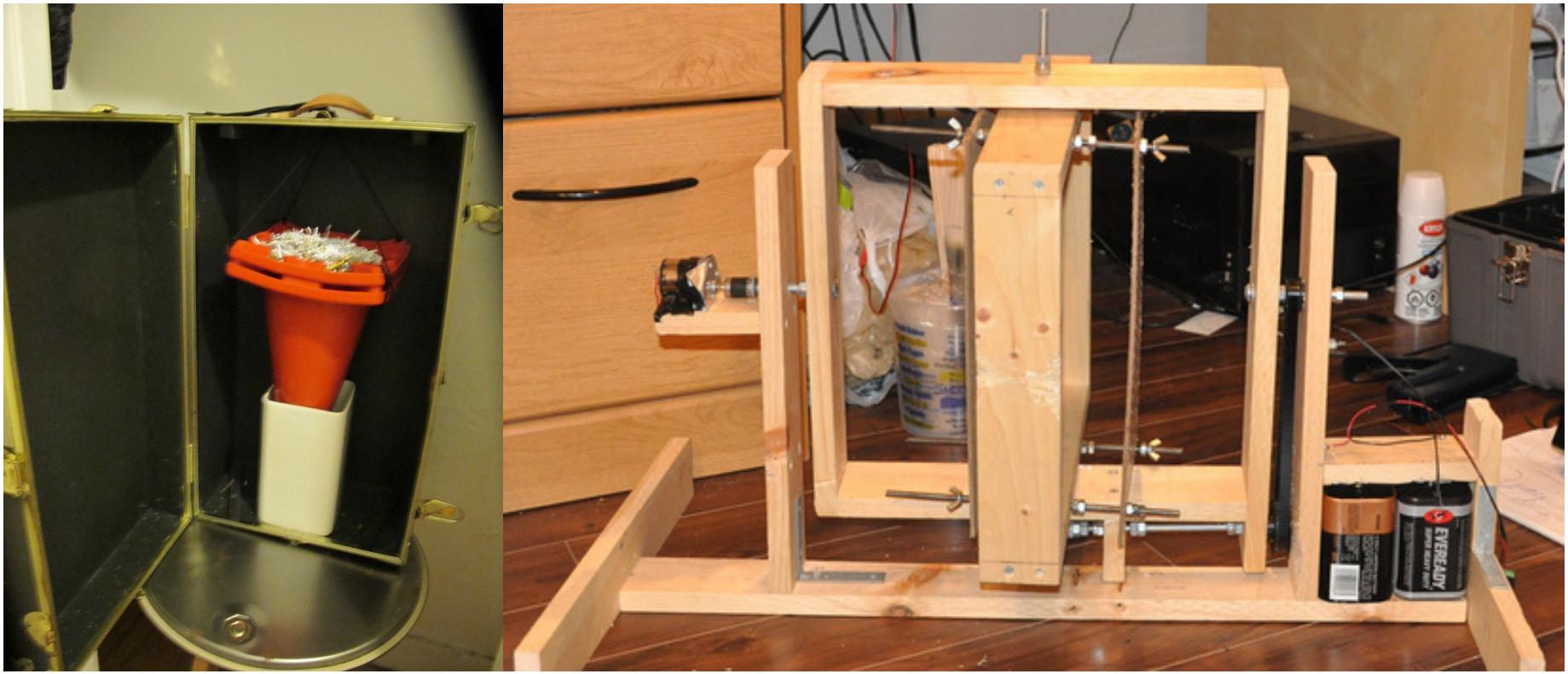
Same principles as FOSS

- Anyone can see the source (design documentation)
- Anyone is free to study, modify and share
- Any modification and distribution under same licence
- Persistence makes everyone profit from improvements

Hardware production

- When produce: licensee is invited to inform the licensor

Example of mechanics licenced with the CERN OHL



Worm farm and rotocaster

Try to use FOSS tools for development

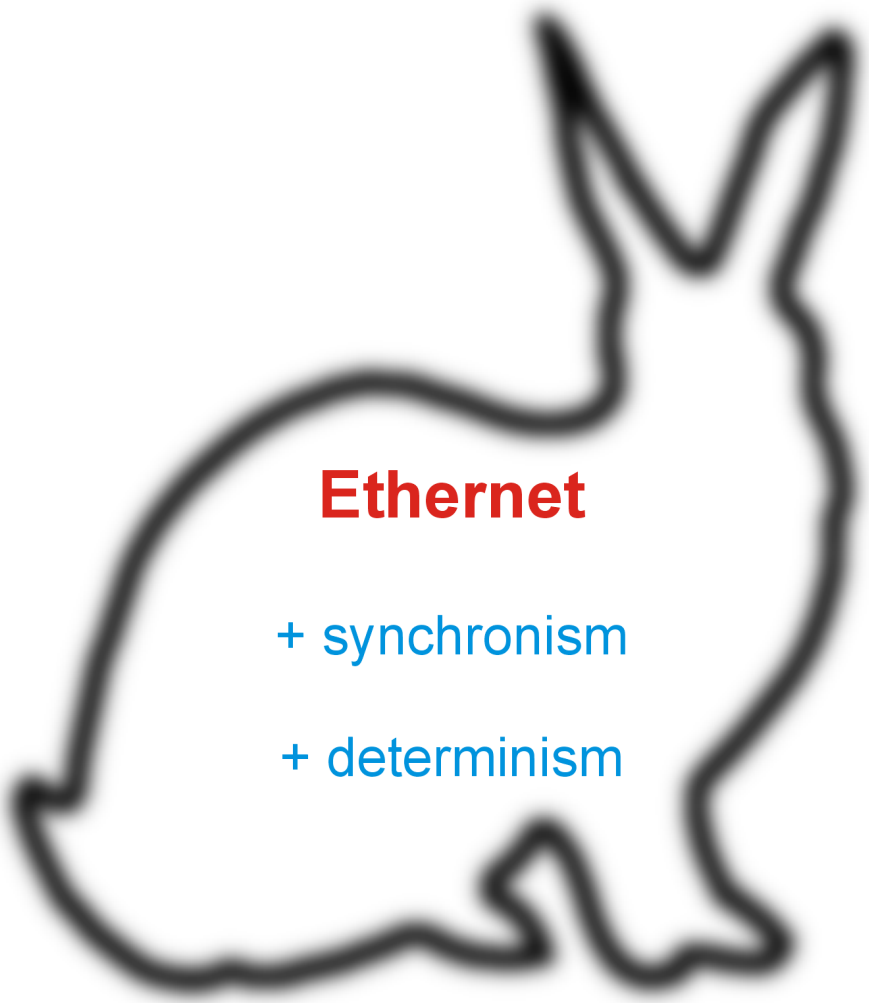
Tools: the last hurdle to sharing

- We already have a forge and a licence.
- Current proprietary CAD tools make it hard to share designs.

Current efforts

- Icarus verilog: help in adding VHDL and SystemVerilog support. See http://iverilog.icarus.com/
- Kicad: help bring it on par with proprietary tools in terms of features and quality. See http://www.ohwr.org/projects/ohr-meta/wiki/Foss-pcb.

What is White Rabbit?



What is White Rabbit?

An **extension** to **Ethernet** which provides:

- **Synchronous mode** (Sync-E) - common clock for physical layer in entire network, allowing for precise time and frequency transfer.
- **Deterministic routing** latency - a guarantee that frame transmission delay between two stations will never exceed a certain boundary.

A multi-lab, multi-company project

- based on Open Hardware and FOSS
- with expanding user base

Design goals

Scalability

Up to 2000 nodes.

Range

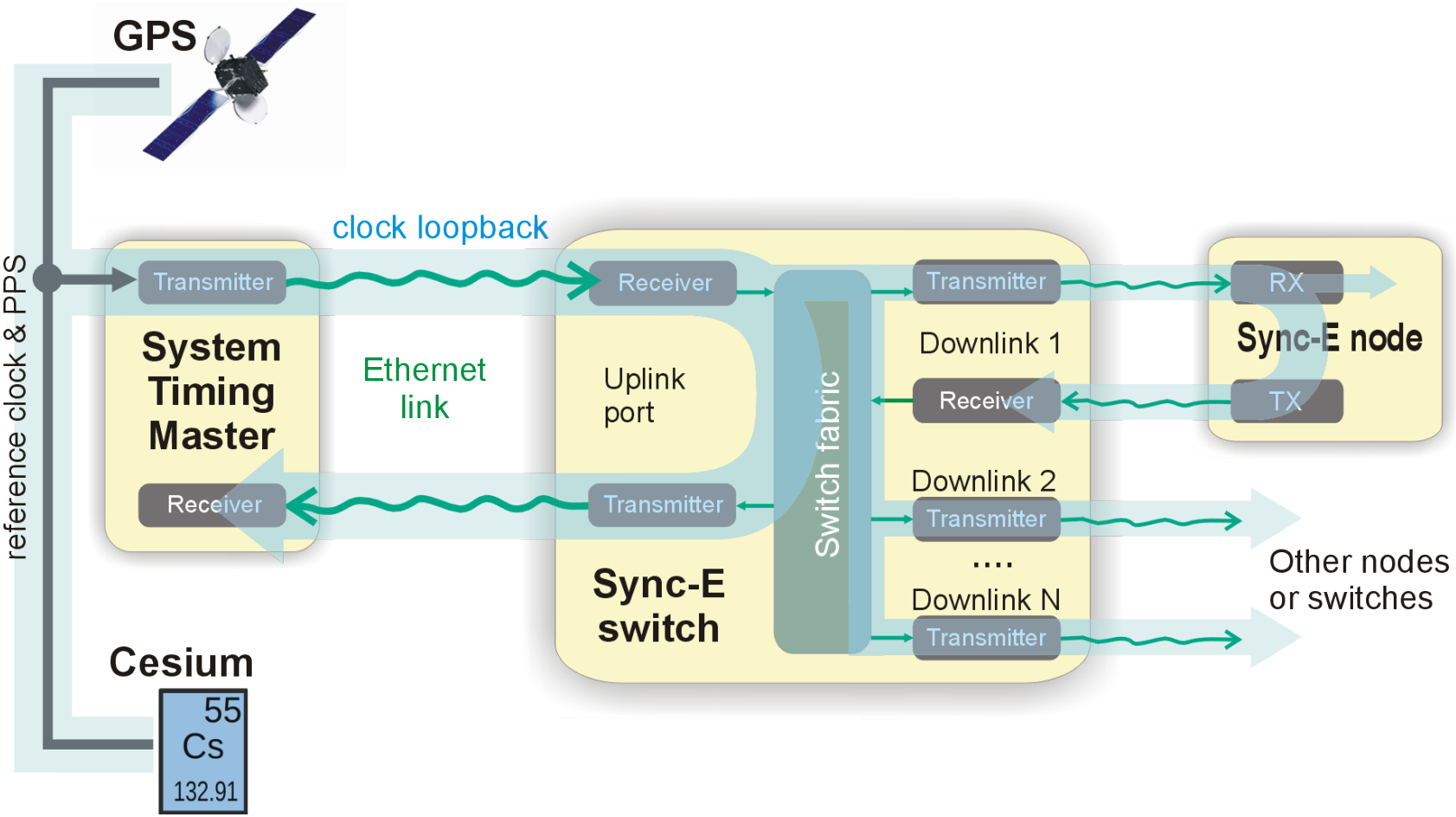
10 km fiber links.

Precision

1 ns time synchronization accuracy, 20 ps jitter.

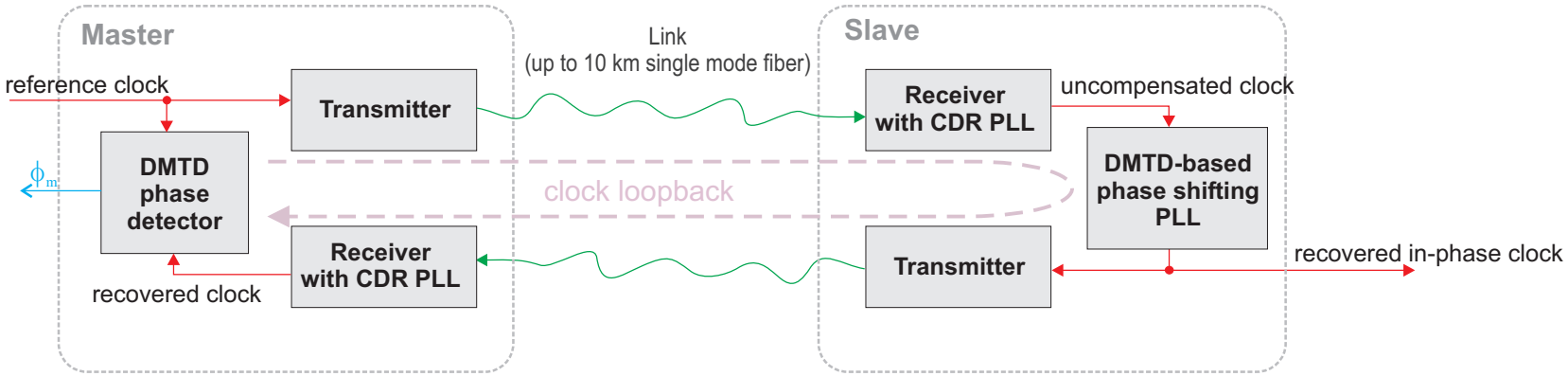
White Rabbit Synchronism

General architecture



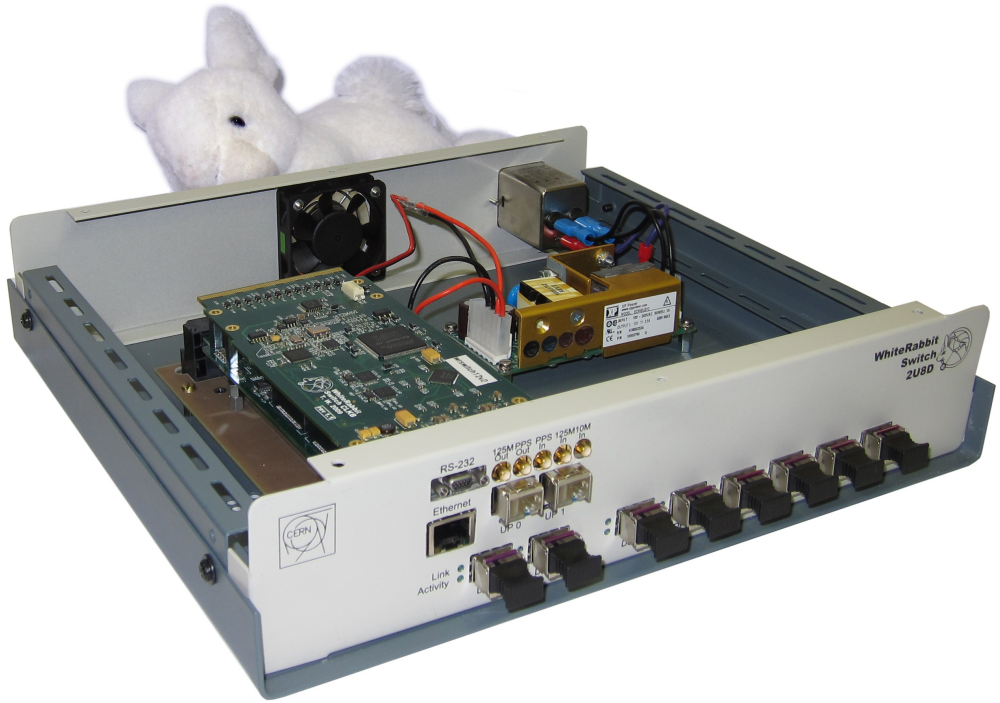
White Rabbit Synchronism

Phase tracking



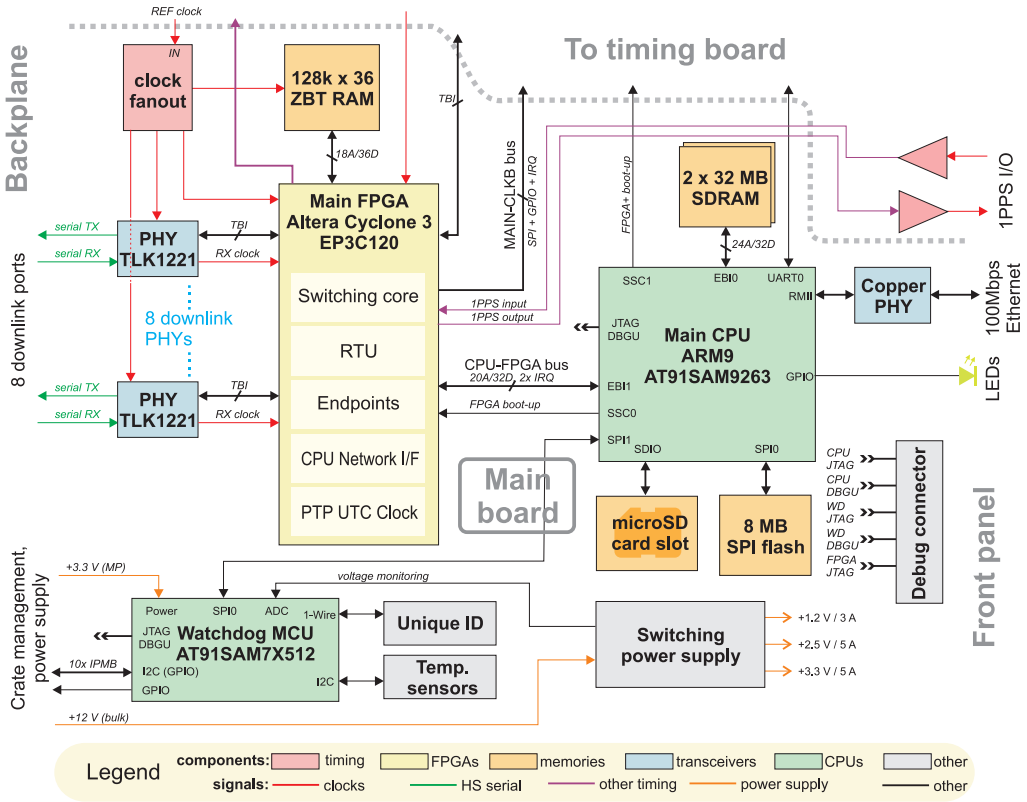
- Monitor phase of bounced-back clock continuously.
- Phase-locked loop in the slave follows the phase changes measured by the master.

White Rabbit Switch



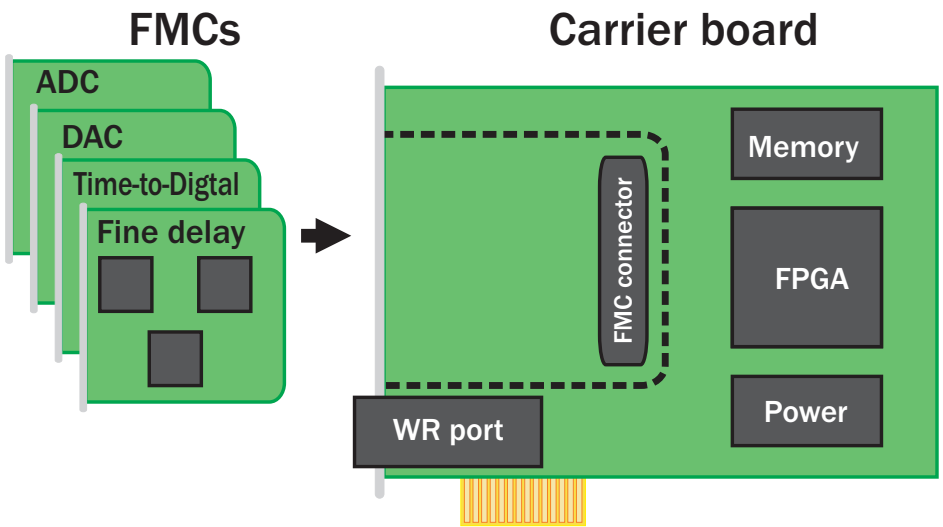
- Central element of WR network.
- Fully custom design, done from scratch at CERN.
- 10 1000Base-X ports, capable of driving 10+ km of SM fiber.
- 200 ps synchronization accuracy.

Switch block diagram - main part



- System FPGA handles all frame processing.
- Implements PTP stack and management functions (SNMP, Spanning Tree) inside a Linux embedded platform.

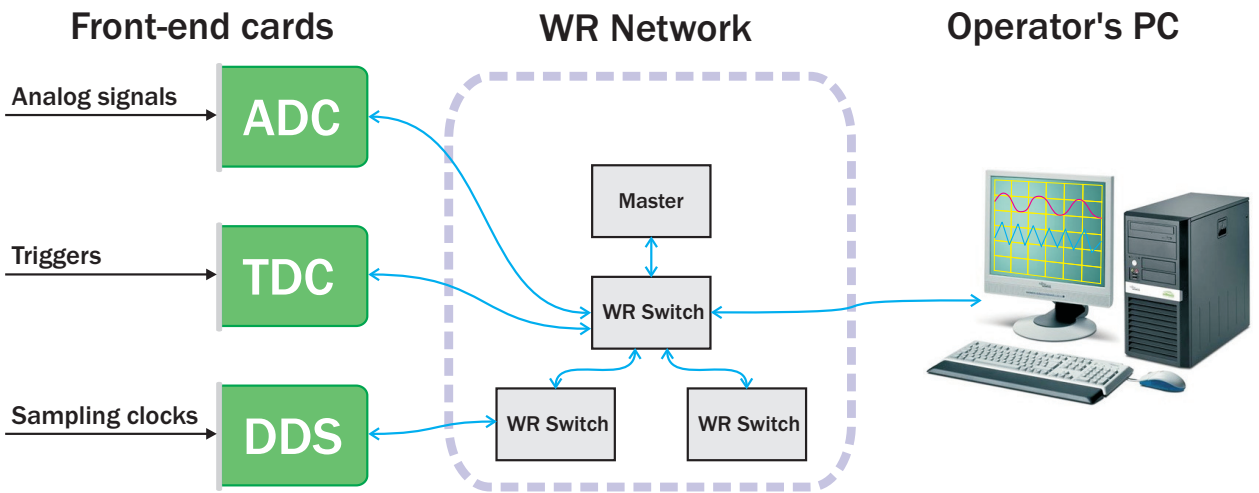
WR in CERN's Hardware Kit



CERN's FMC-based Hardware Kit:

- FMCs (FPGA Mezzanine Cards) with ADCs, DACs, TDCs, fine delays, digital I/O.
- Carrier boards in PCI-Express and VME formats.
- All carriers are equipped with a White Rabbit port.

Distributed oscilloscope



- Common clock in the entire network: no skew between ADCs.
- Ability to sample with different clocks via Distributed DDS.
- External triggers can be time tagged with a TDC and used to reconstruct the original time base in the operator's PC.

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst...

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry,

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)
- many others

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)
- many others

...of diverse breed

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)
- many others

...of diverse breed

- shiny brand new FMC boards

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)
- many others

...of diverse breed

- shiny brand new FMC boards
- legacy COTS and in-house stuff

Software for Diverse Equipment

We got ourselves a rather diverse hardware ecosyst... eeehh, sorry, zoo

Different species of hardware animals...

- analog I/O (ADCs, DACs, waveform generators)
- digital I/O
- timing boards (FD, TDC)
- many others

...of diverse breed

- shiny brand new FMC boards
- legacy COTS and in-house stuff

Some order has to be imposed in the zoo.

Unifying Themes

Unifying Themes

Wishbone-based drivers

- covering the FMC family of boards
- designed around the internal Wishbone bus
- introducing enumeration of device cores

Unifying Themes

Wishbone-based drivers

- covering the FMC family of boards
- designed around the internal Wishbone bus
- introducing enumeration of device cores

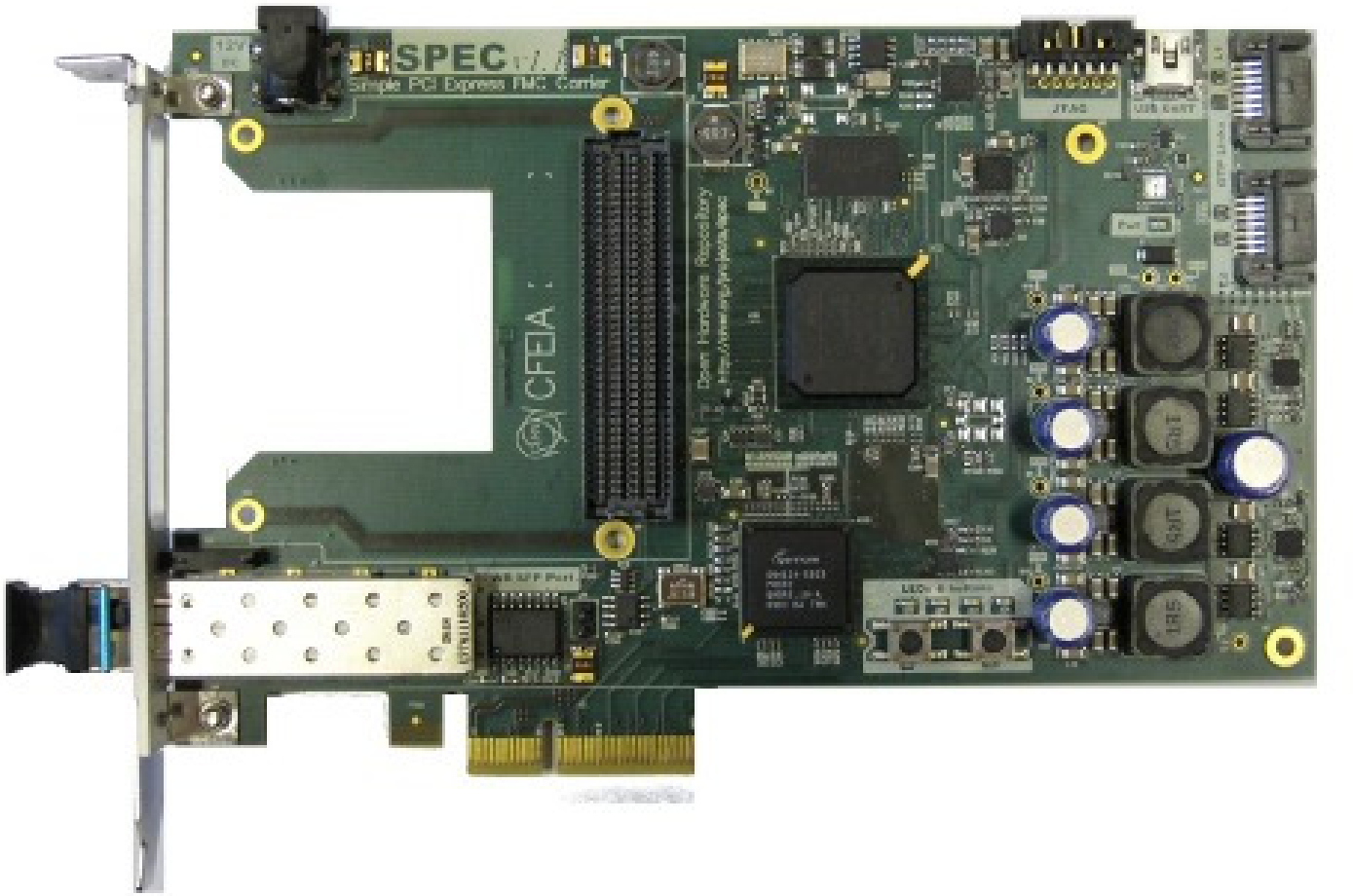
`zio`: The Ultimate I/O

- Linux kernel framework for I/O devices
- oriented to high bandwidth applications

The FMC family of boards

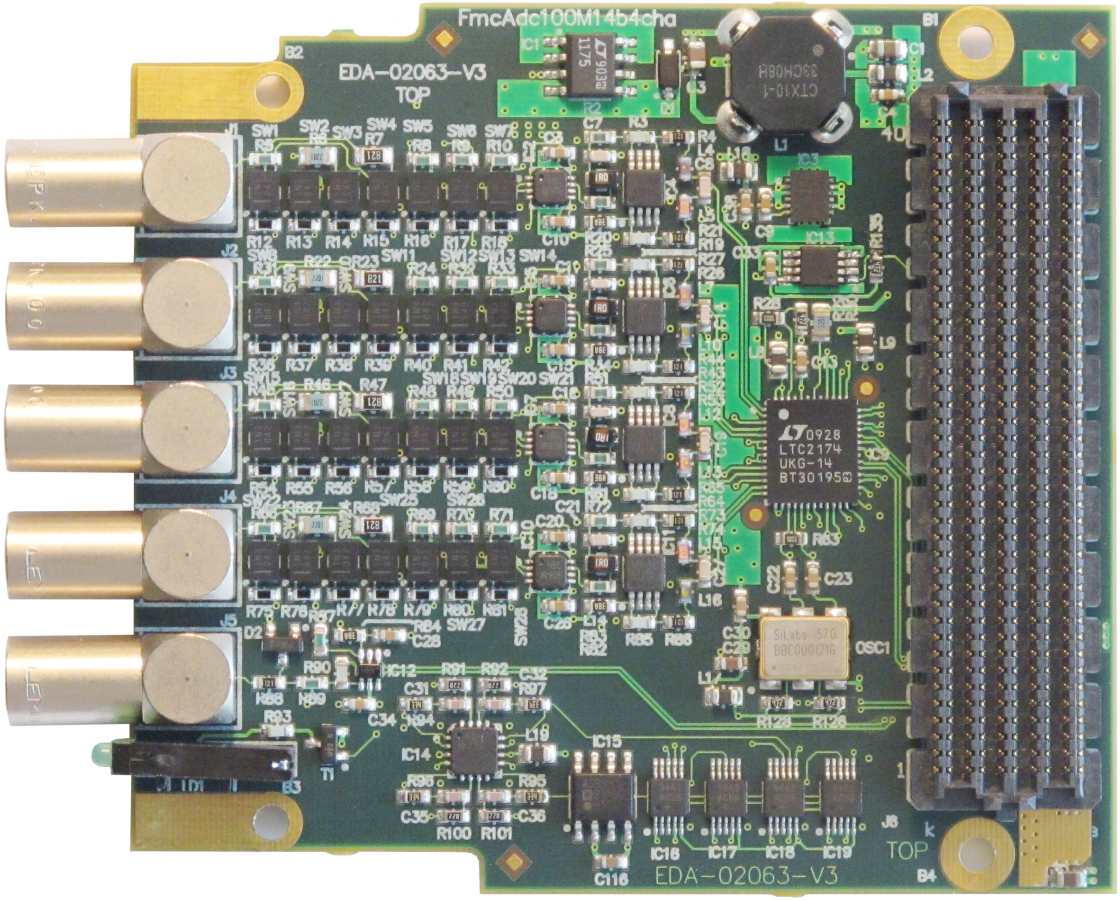
- carriers in PCIe and VME form factors
- simple mezzanines with electronics for ADCs, DACs, DIO and endless other applications
- circuitry in the mezzanine
- FPGA application logic in the carrier
- *logic in the FPGA is organized as a set of IP cores interconnected through an internal Wishbone* bus

A typical data acquisition application: carrier



SPEC carrier board (`http://www.ohwr.org/projects/spec`)

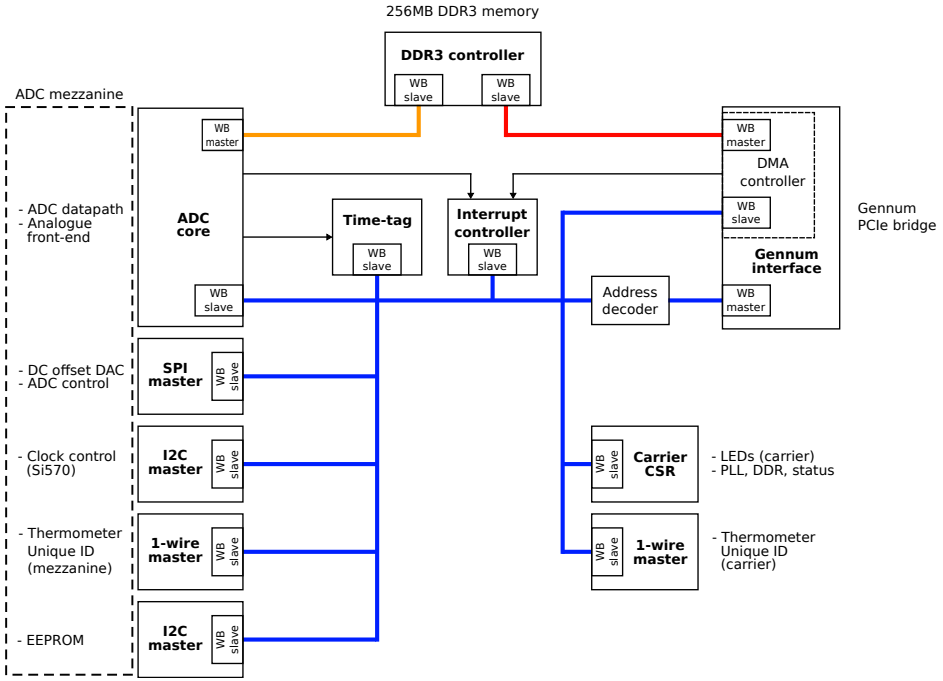
A typical data acquisition application: mezzanine



FMC 100M4ch14b ADC

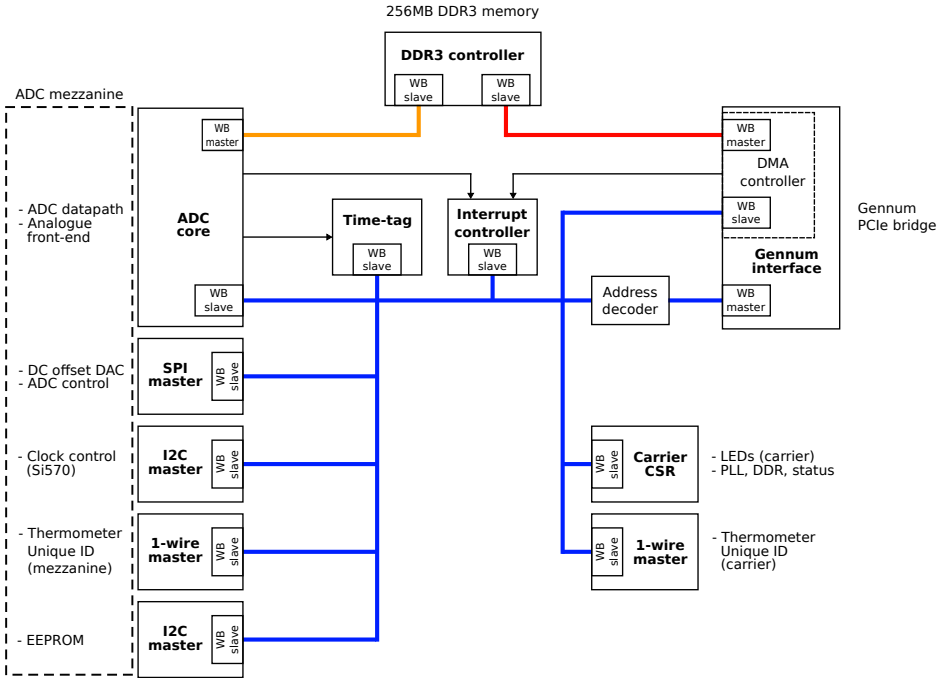
(`http://www.ohwr.org/projects/fmc-adc-100m14b4cha`)

Parts of the whole



Block diagram of the FMC ADC application.

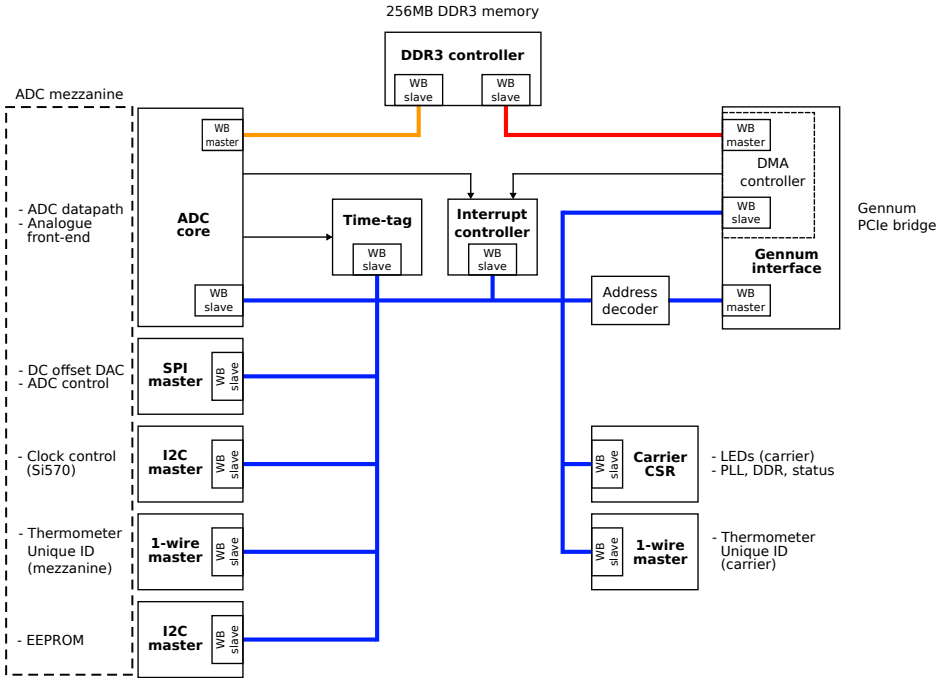
Parts of the whole



Block diagram of the FMC ADC application.

- Basic I^2C interfacing to the mezzanine board.

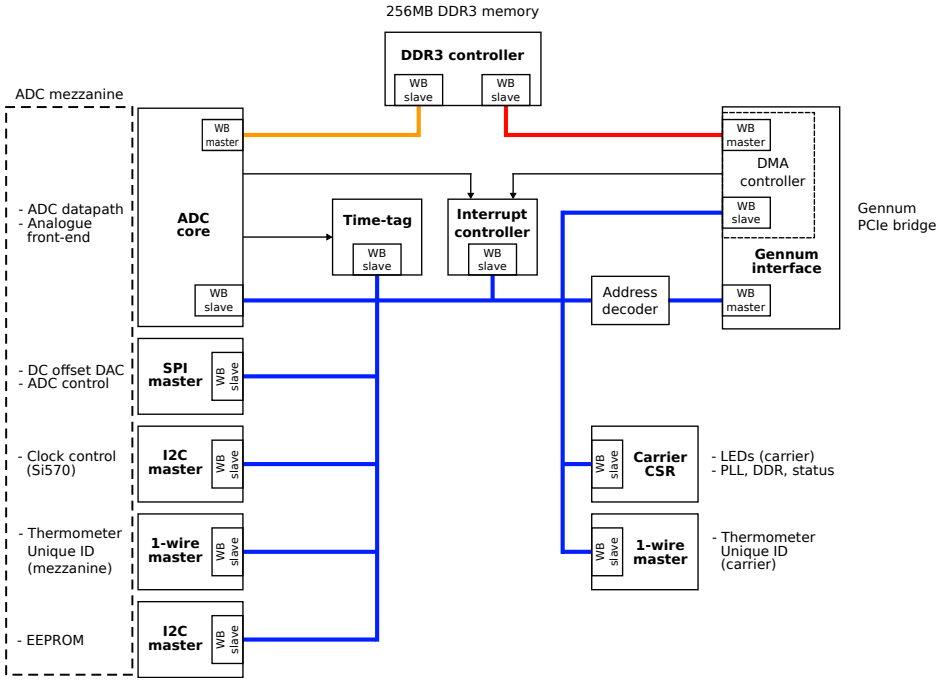
Parts of the whole



Block diagram of the FMC ADC application.

- Basic I^2C interfacing to the mezzanine board.
- Wishbone mastering.

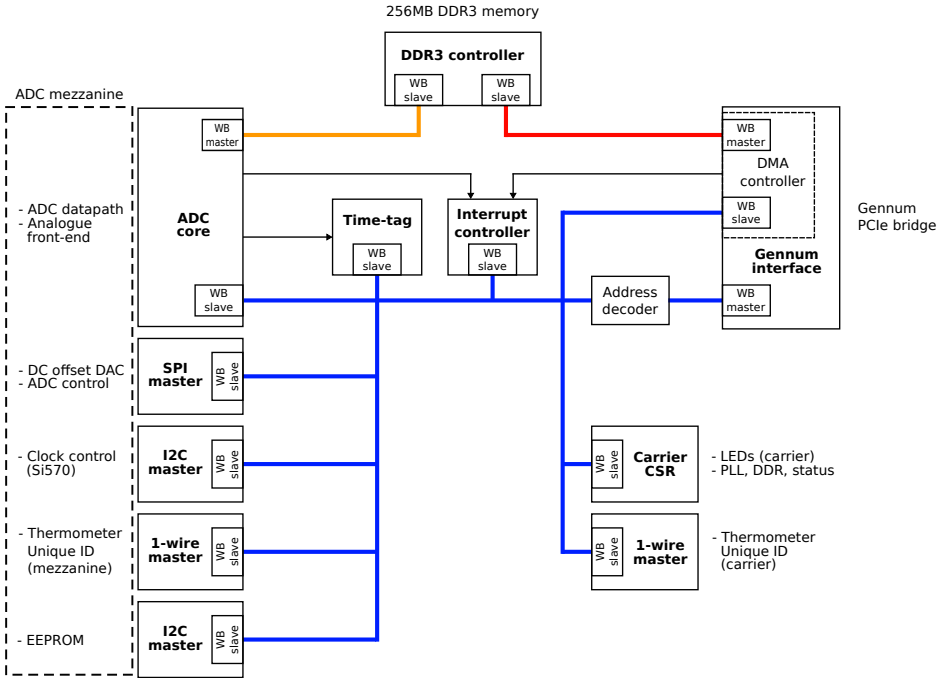
Parts of the whole



Block diagram of the FMC ADC application.

- Basic I^2C interfacing to the mezzanine board.
- Wishbone mastering.
- DMA access to DDR3 memory in the carrier board.

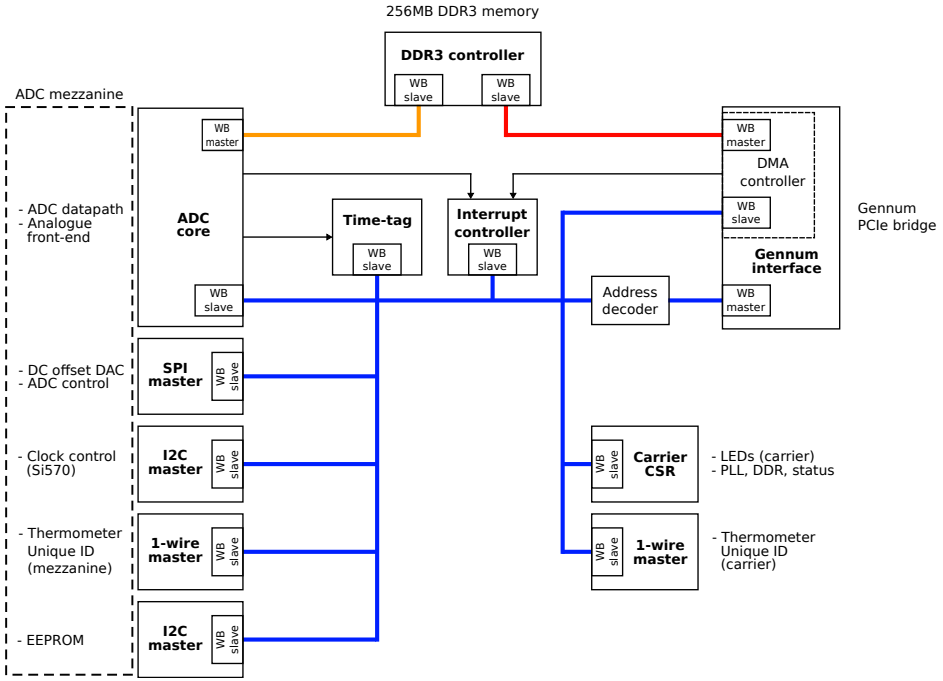
Parts of the whole



Block diagram of the FMC ADC application.

- Basic I^2C interfacing to the mezzanine board.
- Wishbone mastering.
- DMA access to DDR3 memory in the carrier board.
- Mezzanine-specific control logic (*e.g.* ADC programming/setup).

Parts of the whole



Block diagram of the FMC ADC application.

- Basic I^2C interfacing to the mezzanine board.
- Wishbone mastering.
- DMA access to DDR3 memory in the carrier board.
- Mezzanine-specific control logic (*e.g.* ADC programming/setup).
- Interrupt control.

Drivers for the FMC family

Design concepts

Drivers for the FMC family

Design concepts

- modular structure that reflects the core structure of the firmware

Drivers for the FMC family

Design concepts

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver $\leftrightarrow$ core (usually)

Drivers for the FMC family

Design concepts

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver $\leftrightarrow$ core (usually)
- ability to dynamically load bitstreams by application

Drivers for the FMC family

Design concepts

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver $\leftrightarrow$ core (usually)
- ability to dynamically load bitstreams by application

Nothing new under the FOSS sun

Drivers for the FMC family

Design concepts

- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver $\leftrightarrow$ core (usually)
- ability to dynamically load bitstreams by application

Nothing new under the FOSS sun

- On the whole, the driver for the carrier board acts as a basic firmware loader and a bridge driver (with device enumeration *à la PCI)* between the host bus (PCIe, VME) and the FPGA interconnection bus

Drivers for the FMC family

Design concepts

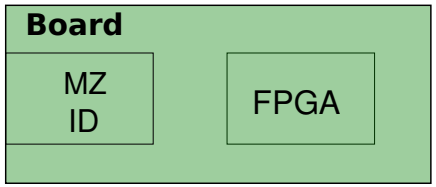
- modular structure that reflects the core structure of the firmware
- one-to-one mapping driver $\leftrightarrow$ core (usually)
- ability to dynamically load bitstreams by application

Nothing new under the FOSS sun

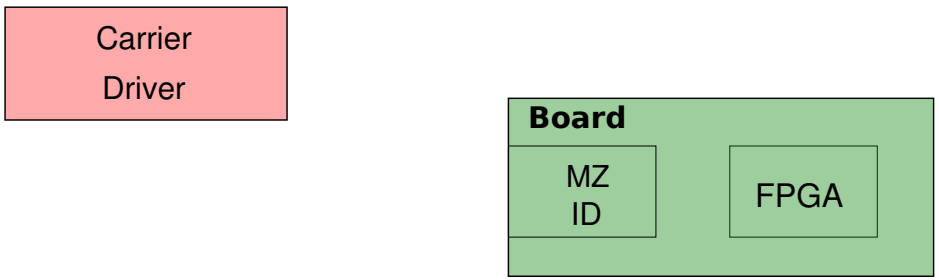
- On the whole, the driver for the carrier board acts as a basic firmware loader and a bridge driver (with device enumeration *à la PCI)* between the host bus (PCIe, VME) and the FPGA interconnection bus
- It will be (we hope) the first Wishbone bus driver in the mainstream kernel

Architecture of the FMC drivers

Architecture of the FMC drivers

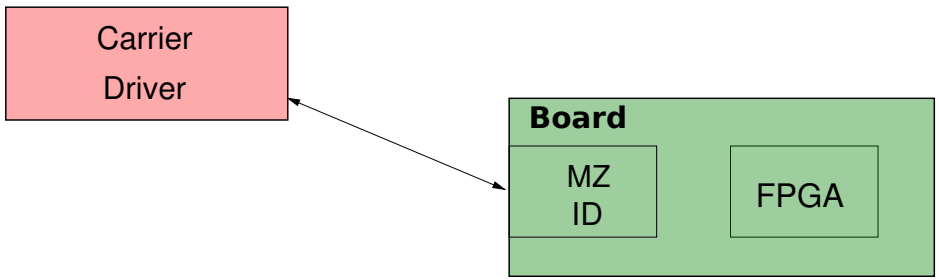


Architecture of the FMC drivers



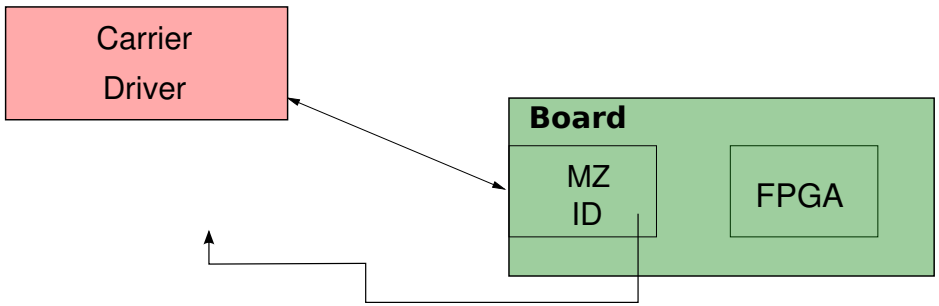
Architecture of the FMC drivers

- Identify the carrier board and initialize it.



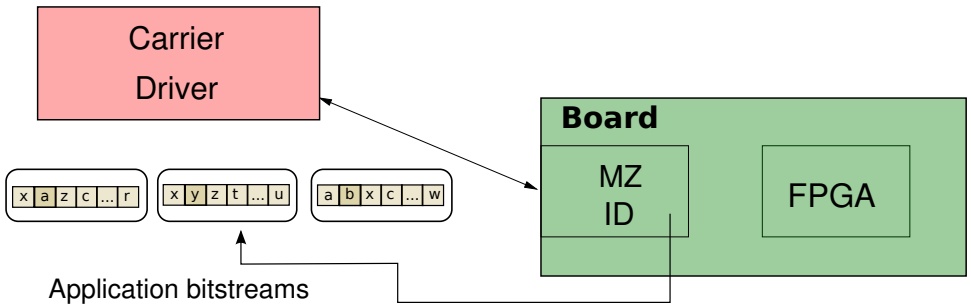
Architecture of the FMC drivers

- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.



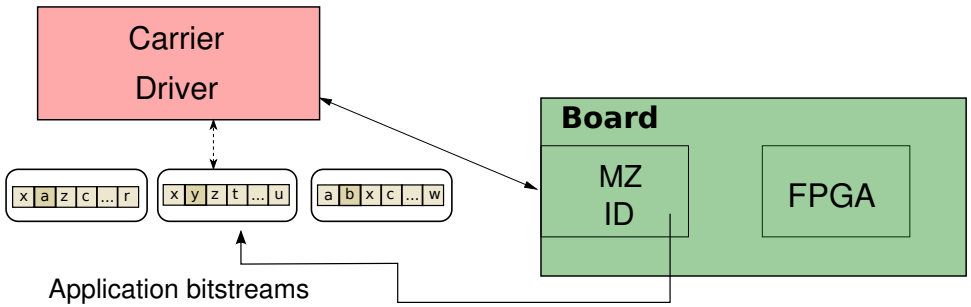
Architecture of the FMC drivers

- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.

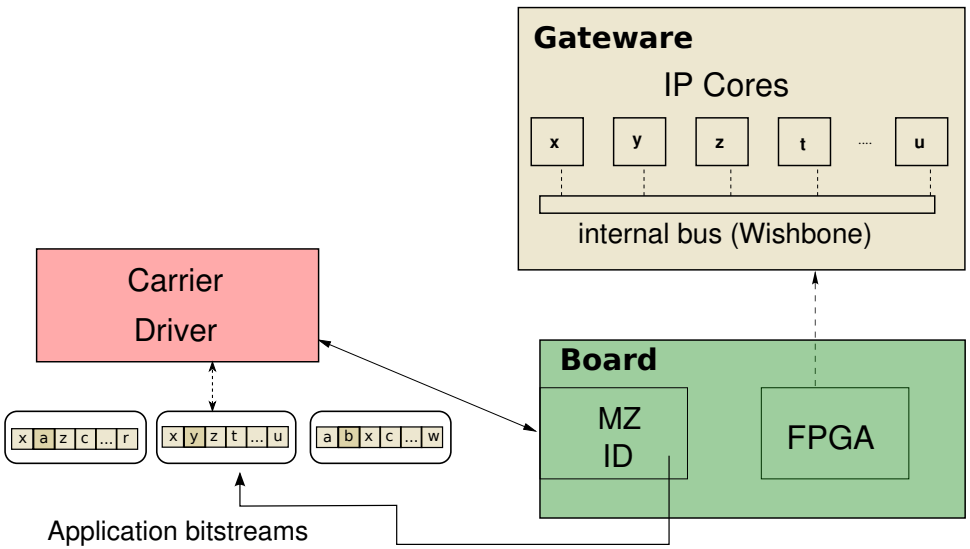


Architecture of the FMC drivers

- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.

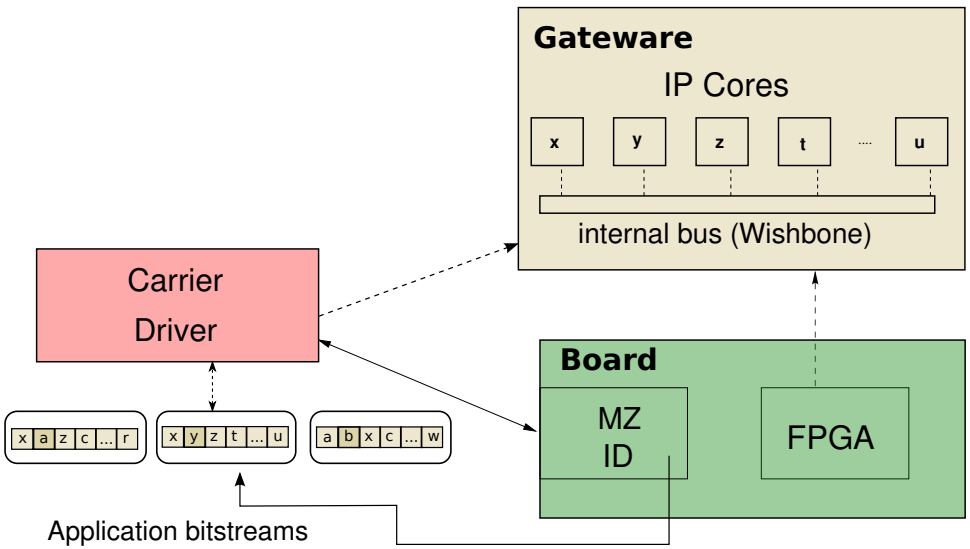


Architecture of the FMC drivers



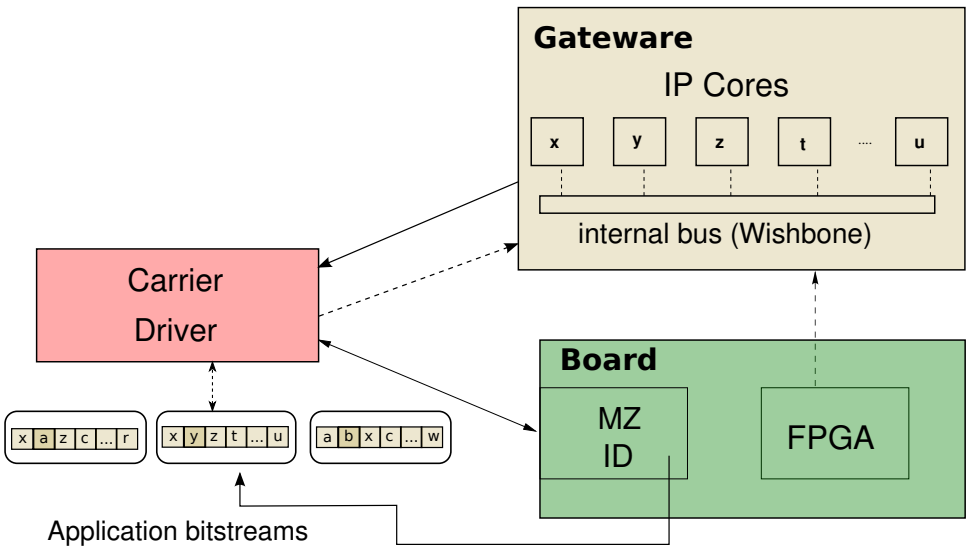
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.

Architecture of the FMC drivers



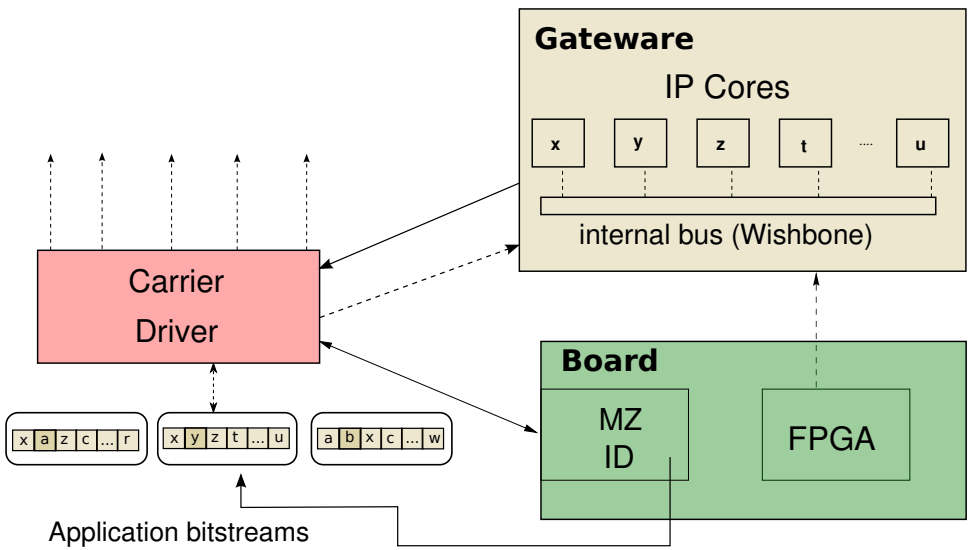
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.

Architecture of the FMC drivers



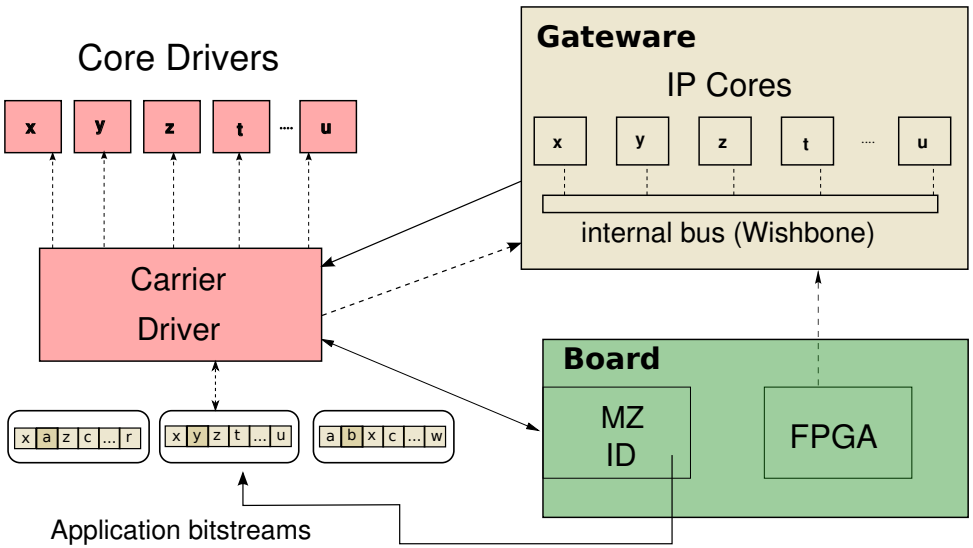
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.

Architecture of the FMC drivers



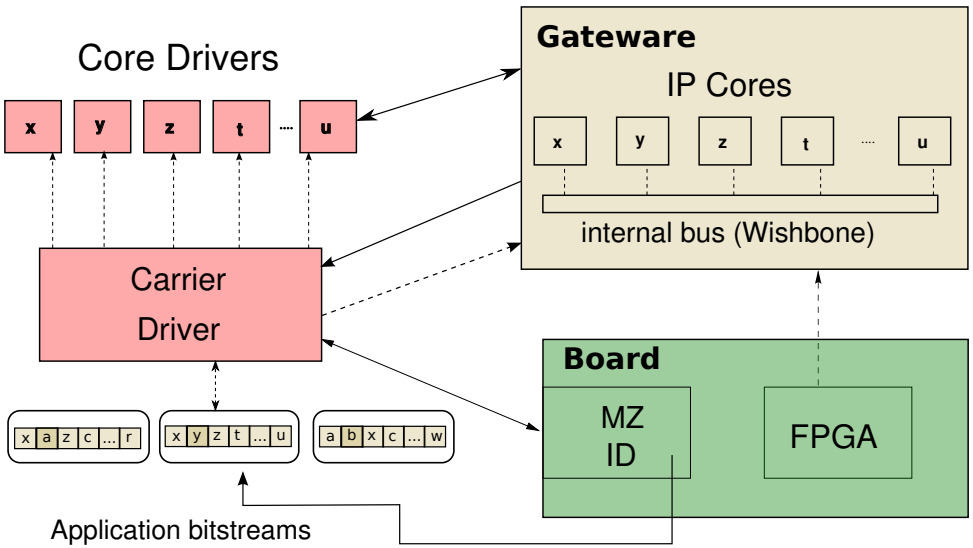
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.
- Enumerate the cores in that firmware.

Architecture of the FMC drivers



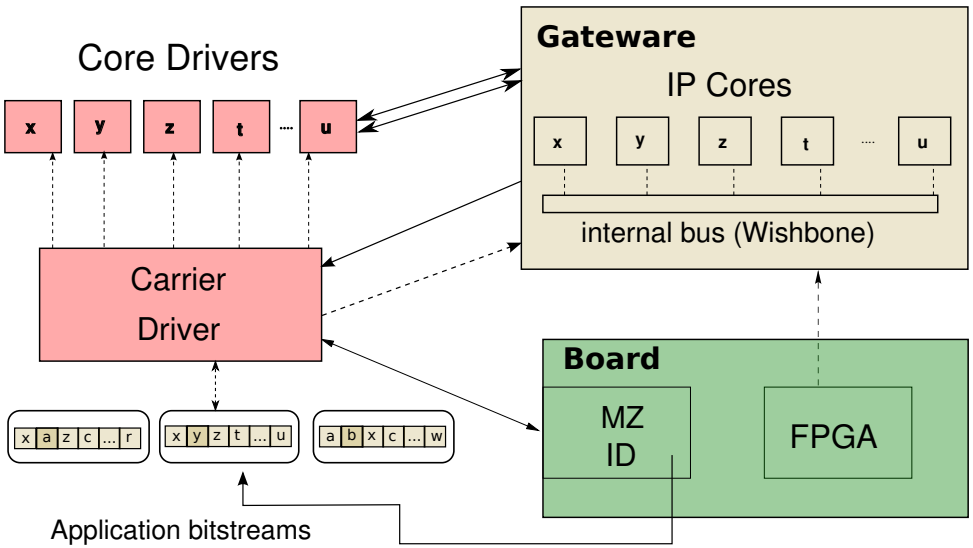
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.
- Enumerate the cores in that firmware.
- Register the devices those cores implement and install the drivers associated to them.

Architecture of the FMC drivers



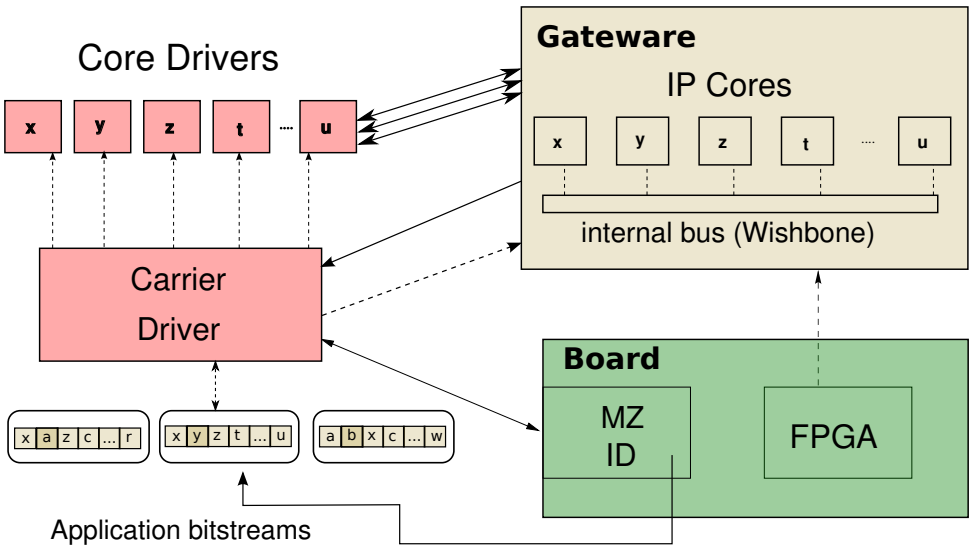
- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.
- Enumerate the cores in that firmware.
- Register the devices those cores implement and install the drivers associated to them.

Architecture of the FMC drivers



- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.
- Enumerate the cores in that firmware.
- Register the devices those cores implement and install the drivers associated to them.

Architecture of the FMC drivers



- Identify the carrier board and initialize it.
- Perform a basic identification of the mezzanine(s) installed in the FMC slot(s), and their configured applications.
- Load the application firmware into the carrier FPGA.
- Register a Wishbone bus with the kernel.
- Enumerate the cores in that firmware.
- Register the devices those cores implement and install the drivers associated to them.

Linux Kernel I/O frameworks

Linux Kernel I/O frameworks

In Linux staging area

- Comedi
- IIO

Linux Kernel I/O frameworks

In Linux staging area

- Comedi
- IIO

Drawbacks (for CERN applications)

- interfaces are cumbersome (Comedi)
- our use cases are far more complicated

Linux Kernel I/O frameworks

In Linux staging area

- Comedi
- IIO

Drawbacks (for CERN applications)

- interfaces are cumbersome (Comedi)
- our use cases are far more complicated

Then `zio` comes

- Alessandro Rubini and Federico Vaga, main developers
- Designed *ab initio* for upstream integration
- See under `http://www.ohwr.org/projects/zio`

`zio` features

The `zio` framework is designed to flexibly support the following aspects:

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution,

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate,

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration,

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset and gain.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset and gain.
- Bit grouping in digital I/O.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset and gain.
- Bit grouping in digital I/O.
- Triggering of acquisition/output.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset and gain.
- Bit grouping in digital I/O.
- Triggering of acquisition/output.
- Support for DMA.

zio features

The zio framework is designed to flexibly support the following aspects:

- Digital and analog input and output.
- One-shot and streaming (buffered) data acquisition or waveform play.
- Buffer management and timing for streaming conversion.
- Resolution, sampling rate, timestamping.
- Calibration, offset and gain.
- Bit grouping in digital I/O.
- Triggering of acquisition/output.
- Support for DMA.
- *Potential to integrate in the main tree.*

zio concepts

zio abstractions

zio concepts

zio abstractions

devices victims of our device drivers, contain channels

zio concepts

zio abstractions

devices victims of our device drivers, contain channels

channels basic I/O units

zio concepts

zio abstractions

devices victims of our device drivers, contain channels

channels basic I/O units

channel sets group channels to be triggered (acquire, output) simultaneously

zio concepts

zio abstractions

devices	victims of our device drivers, contain channels
channels	basic I/O units
channel sets	group channels to be triggered (acquire, output) simultaneously
buffers	for input and output buffering, of course

zio concepts

zio abstractions

devices	victims of our device drivers, contain channels
channels	basic I/O units
channel sets	group channels to be triggered (acquire, output) simultaneously
buffers	for input and output buffering, of course
triggers	cause acquisitions to occur

Next candidates for (`zio`) integration

And for mainstream integration as well...

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)
- . . . and so forth.

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)
- . . . and so forth.

CERN-developed drivers for good old beasts

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)
- . . . and so forth.

CERN-developed drivers for good old beasts

- Struck SIS33xx ADCs

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)
- . . . and so forth.

CERN-developed drivers for good old beasts

- Struck SIS33xx ADCs
- Tews TPCI200/TVME200 carries plus IPOCTAL serial boards

Next candidates for (zio) integration

And for mainstream integration as well...

CERN-developed OHWR FMC boards

- FD (2012 Q1)
- 100Msps ADC (2012 Q2)
- TDC (2012 Q2)
- . . . and so forth.

CERN-developed drivers for good old beasts

- Struck SIS33xx ADCs
- Tews TPCI200/TVME200 carries plus IPOCTAL serial boards
- timing receivers, White Rabbit, etc.

Let’s summarize

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices...

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share
- Open Hardware makes for a much more manageable design and production way

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share
- Open Hardware makes for a much more manageable design and production way
- It is proven to work in practice

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share
- Open Hardware makes for a much more manageable design and production way
- It is proven to work in practice
- Both for a big project like White Rabbit and for the myriad of heterogeneous devices linked to it

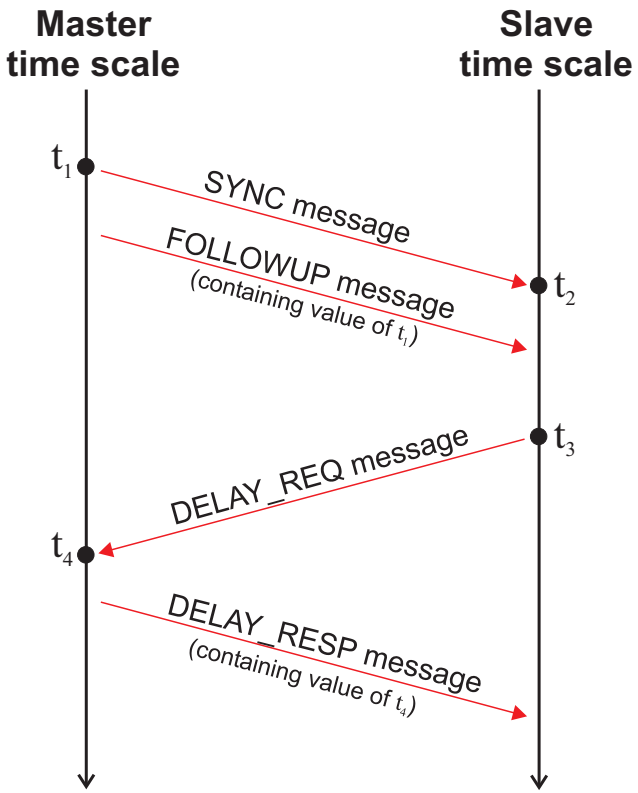
Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share
- Open Hardware makes for a much more manageable design and production way
- It is proven to work in practice
- Both for a big project like White Rabbit and for the myriad of heterogeneous devices linked to it
- The software side cannot simply live without the FOSS model

Let's summarize

- Accelerator control systems require managing a complex and diverse zoo of hardware devices. . . a problem many people share
- Open Hardware makes for a much more manageable design and production way
- It is proven to work in practice
- Both for a big project like White Rabbit and for the myriad of heterogeneous devices linked to it
- The software side cannot simply live without the FOSS model
- Especially the Linux kernel! Thanks for the device model and the I/O frameworks

Two-way delay compensation schemes



Having the values of t_1, t_2, t_3 and t_4, the slave can calculate the one-way link delay:

$$\delta_{ms} = \frac{(t_4 - t_1) - (t_3 - t_2)}{2}$$

Millisecond timing clients

Millisecond timing

Example: Network Time Protocol (NTP)

Used in general-purpose computers

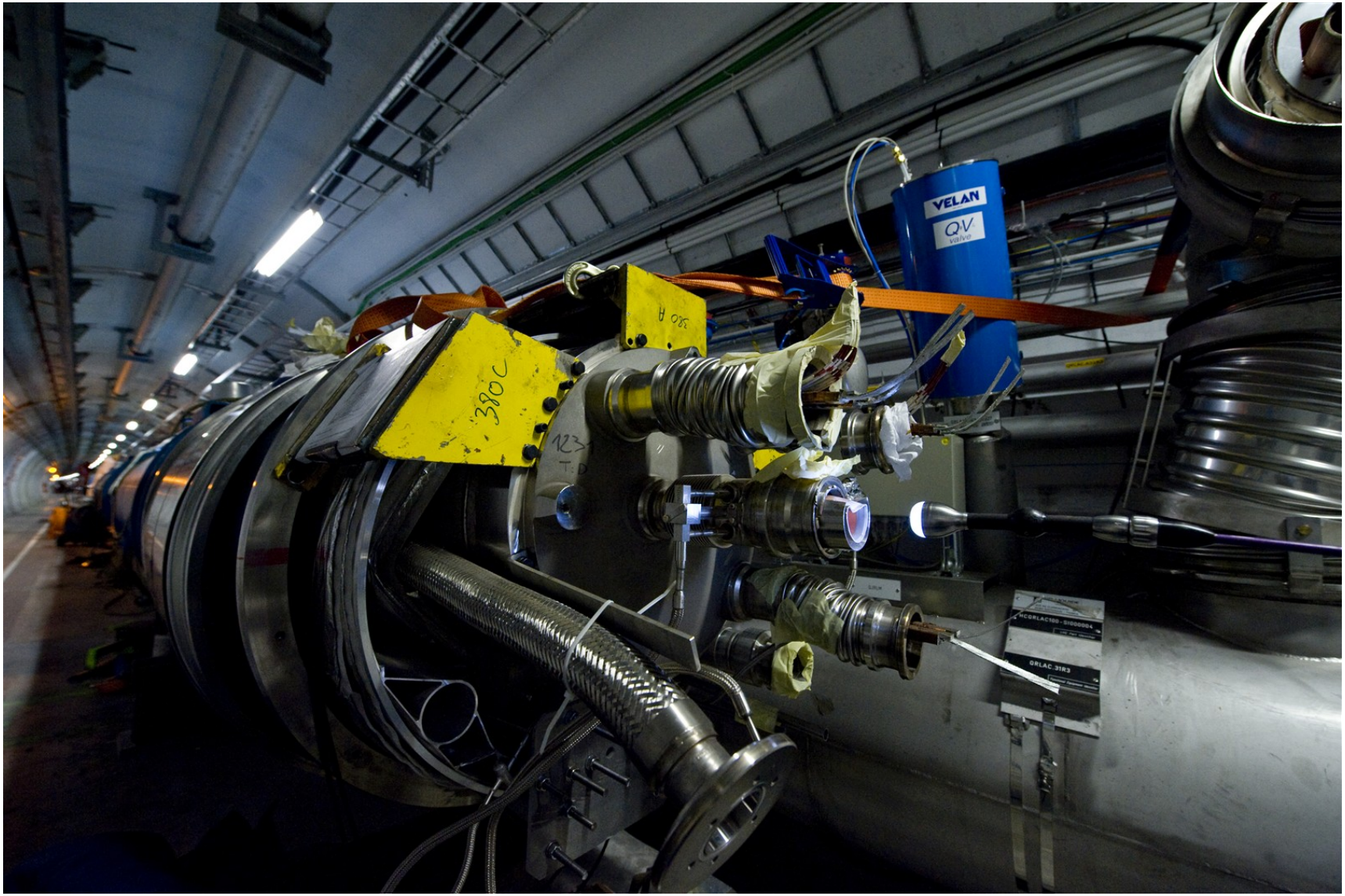
- Works across the Internet.
- Each client (slave) gets synchronized to one or more servers.

Cannot do better than 1 ms

- Asymmetries in network, switches and routers.
- Non-determinism due to OS scheduler (time tags done in SW).
- Requires strong statistics artillery to average over many measurements.

Microsecond timing clients

Microsecond timing clients



Microsecond timing

Example: Precision Time Protocol (PTP, IEEE1588)

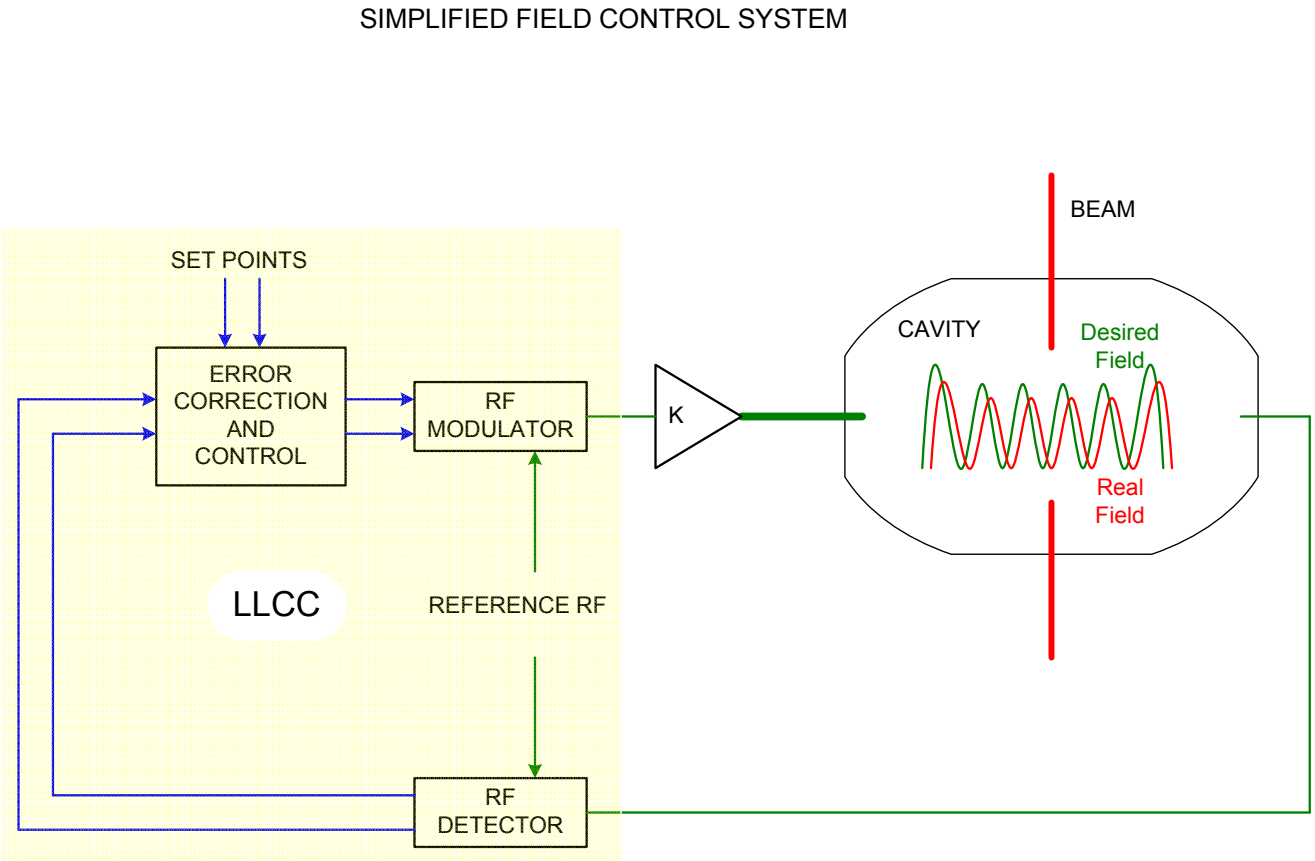
Acts on both of NTP's shortcomings

- Time-tagging can be done in HW.
- Special PTP switches ensure no loss in precision.

Has a hard time doing better than 1μs

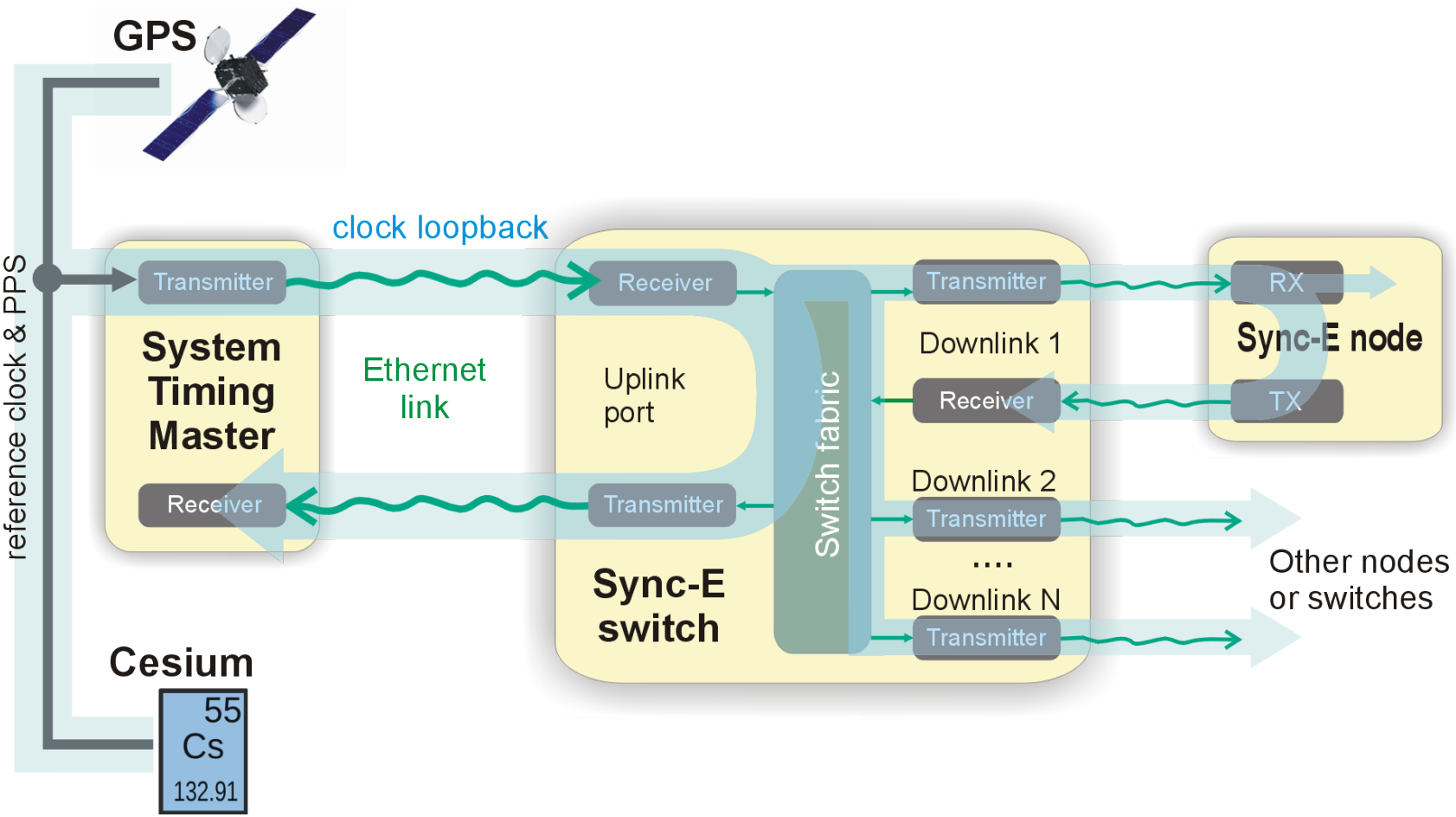
- Typical nodes use a free-running oscillator.
- Frequency offset (and drift) compensation generates extra traffic.

Nanosecond and picosecond timing clients



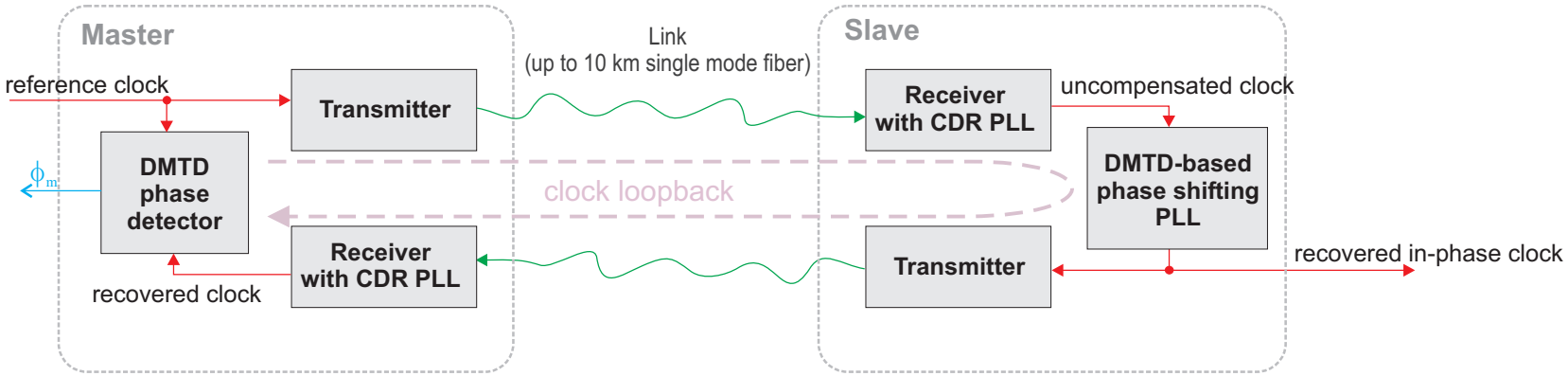
Nanosecond and picosecond timing

Example: White Rabbit



Nanosecond and picosecond timing

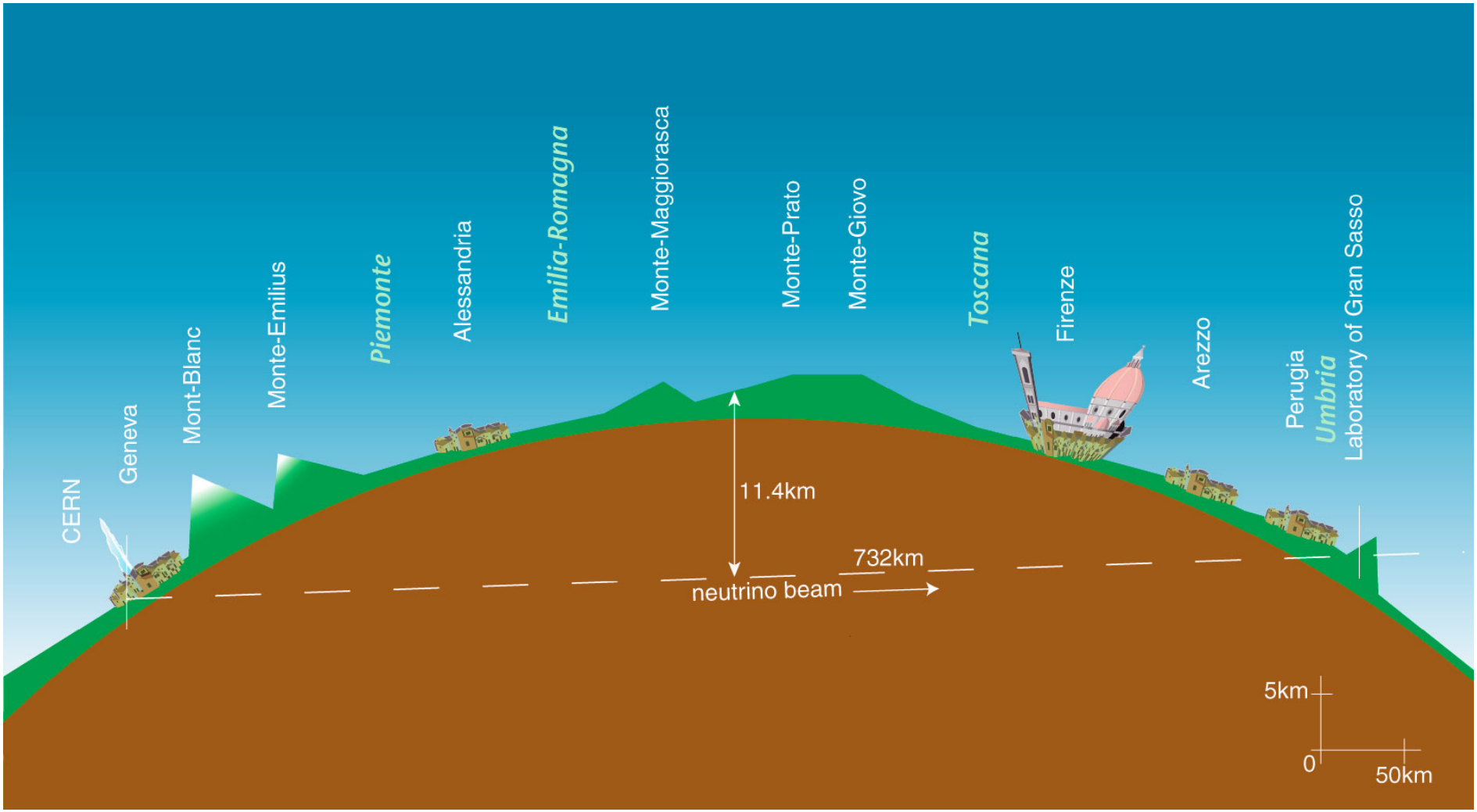
Phase tracking



- Monitor phase of bounced-back clock continuously.
- Phase-locked loop in the slave follows the phase changes measured by the master.

Nanosecond and picosecond timing

Another example: neutrino oscillation experiments



Nanosecond and picosecond timing

Another example: neutrino oscillation experiments