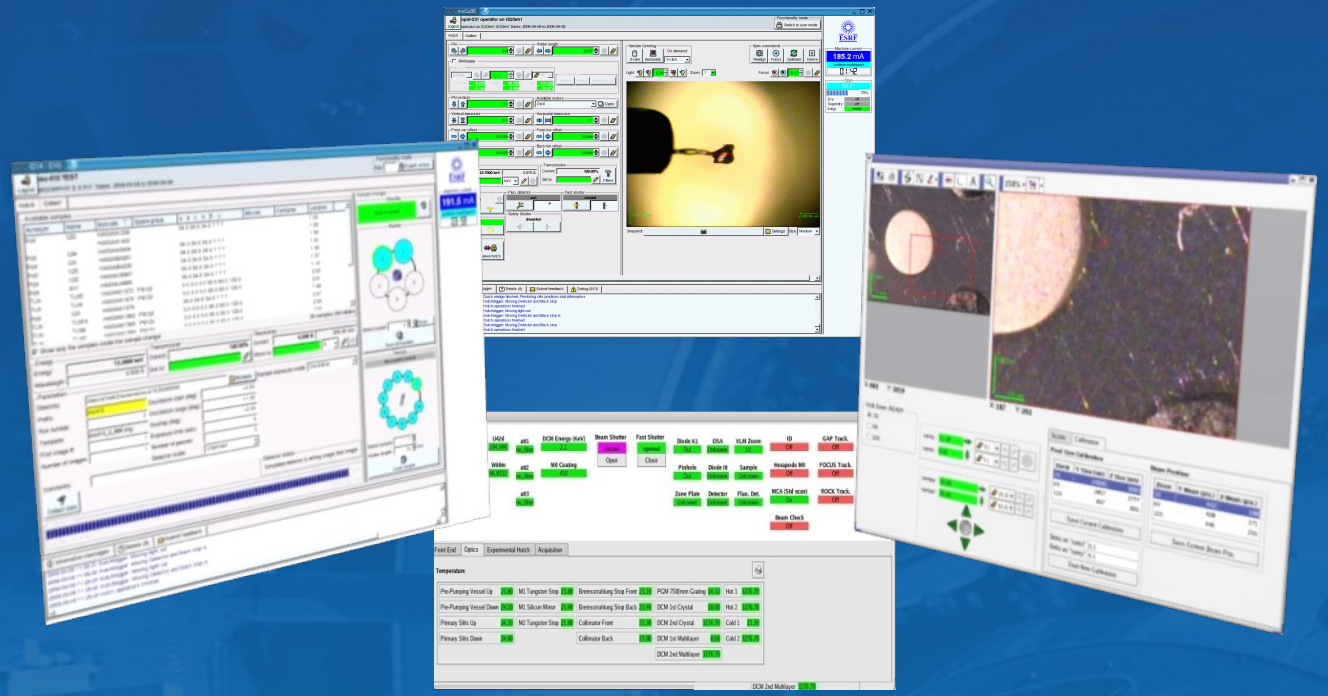
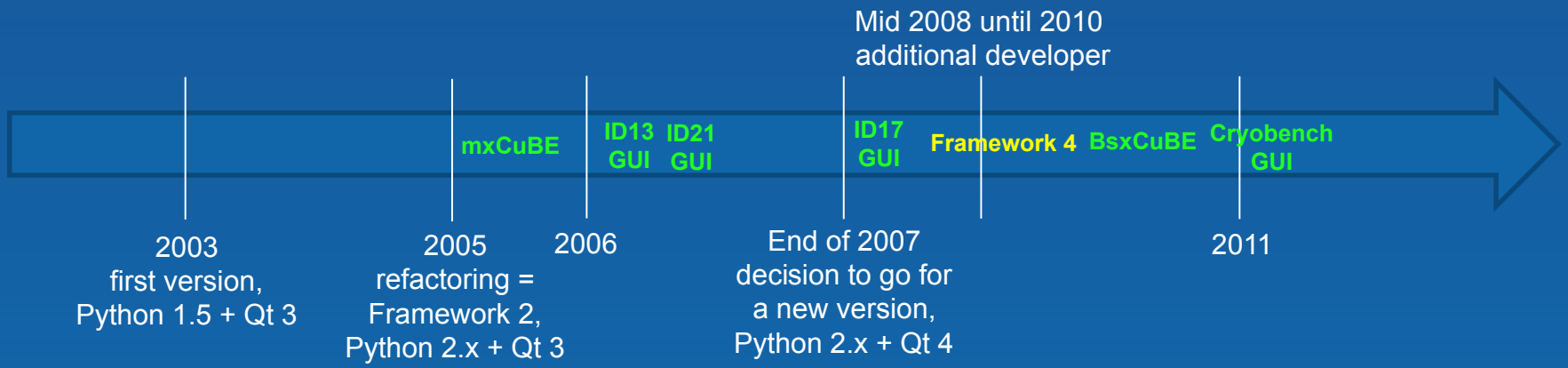
Framework 4 : Control-oriented graphical interfaces for beamlines



What is the Framework ?

- A library and a set of tools for ESRF BCU staff to deliver graphical applications on beamlines for data acquisition and experiment control
- Python + Qt
- Control system agnostic : works on top of spec, Taco, Tango, …
- Design principles
  - bricks instead of widgets
  - MVC architecture
- A long-running project within BCU



What is the Framework ?

A threefold project

- Control system abstraction
- GUI bricks
- Application builder

Control system abstraction

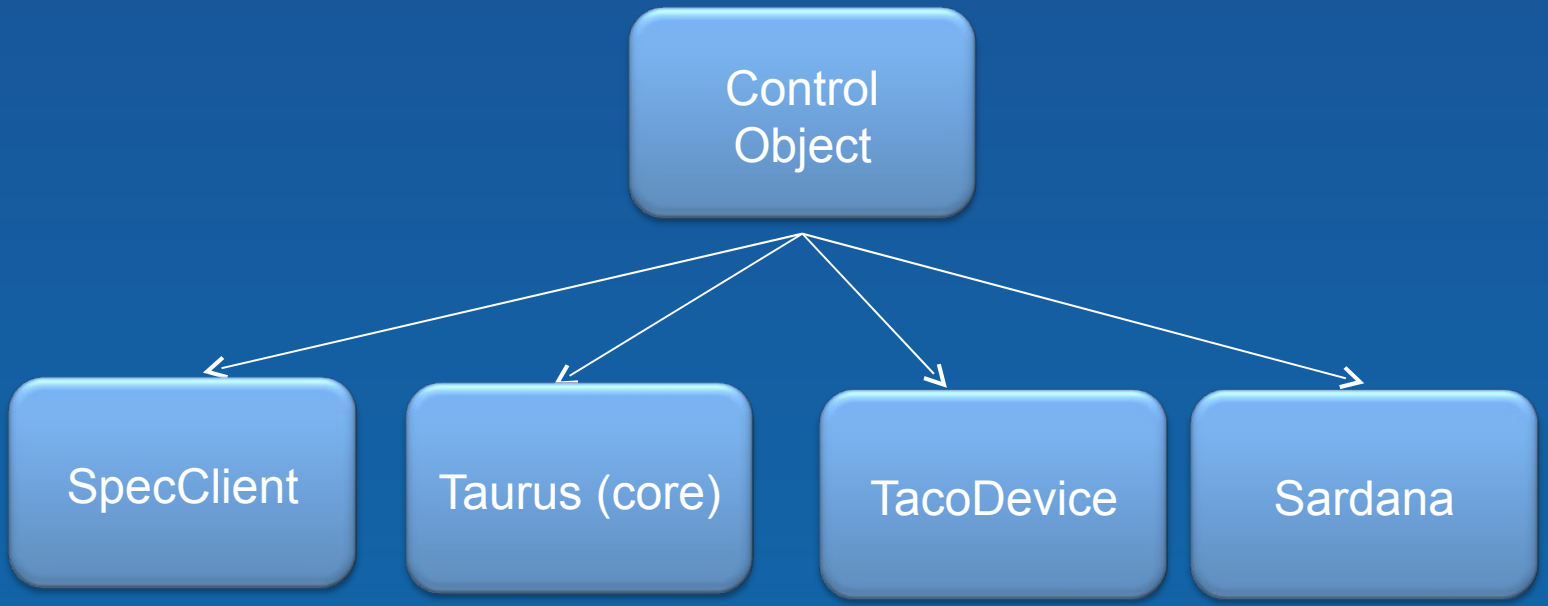
Control Objects

- Control Objects are Python objects containing Command and Channel objects
- Command objects corresponds to :
  - Spec macros
  - Taco commands
  - Tango commands
  - Sardana Macro Server macros
- Channel objects corresponds to :
  - Spec channels
  - Tango attributes
- Control Objects can emit signals/events to notify listeners something happened
- Starting with Framework 4, Control Objects are really independent of Qt 

Control system abstraction

Control Objects

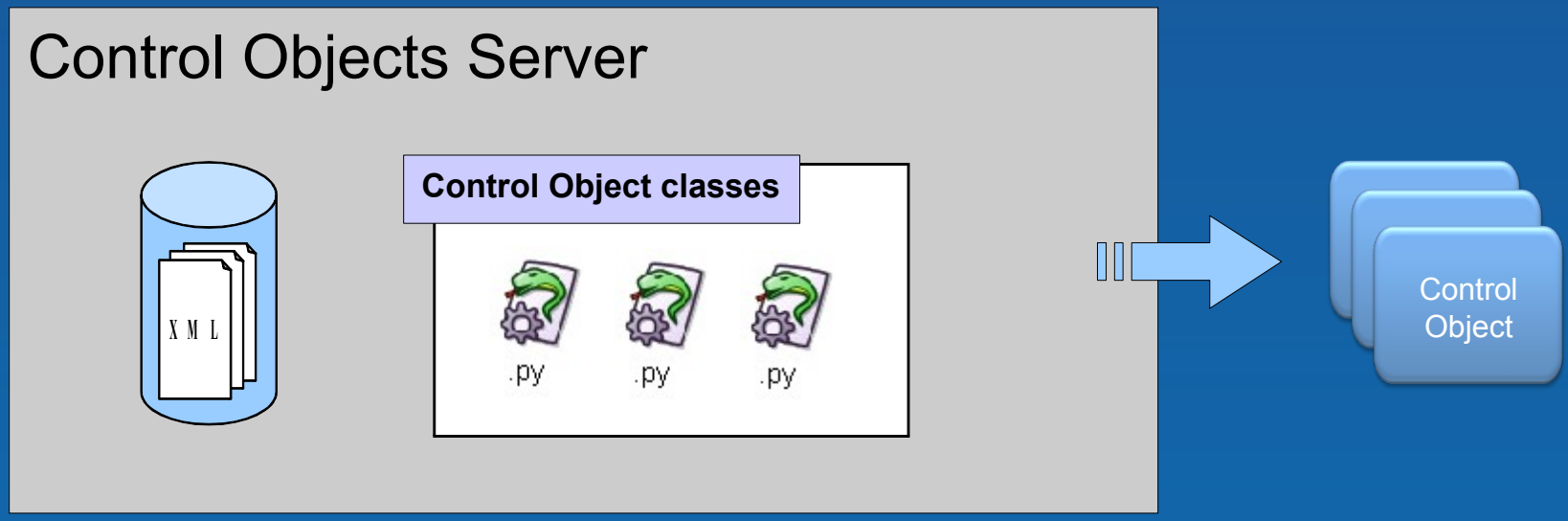
- Control Objects provide a unified interface to middleware and other control systems



Control system abstraction

Control Objects Server

- There is one Control Objects Server per beamline
- The Control Objects Server takes care of configuration for Control Objects
- Configuration is stored in XML files
- A Control Object can run within the Control Objects Server or within the client GUI application



Control system abstraction

Example : a Shutter Control Object

- Defines 2 commands (open, close) and 2 channels (state, status)

```
<object class="Shutter"  username="Safety Shutter">
       <channel name="status"  uri="//basil/id23/bsh/1"  type="taco" call="DevStatus" />
       <channel name="state"  uri="//basil/id23/bsh/1"  type="taco" call="DevState" />
       <command name="open"  uri="//basil/id23/bsh/1"  type="taco" call="DevOpen" />
       <command name="close"  uri="//basil/id23/bsh/1"  type="taco" call="DevClose" />
</object>
```

- Logic is in the Python code (Shutter class)

Control system abstraction

Example : a Shutter Control Object

```
from Framework4.Control.Core.CObject import CObjectBase, Signal, Slot

class Shutter(CObjectBase):
    signals = [Signal('statusChanged'), Signal('stateChanged')]
    slots = [ Slot("open"), Slot("close") ]

    def init(self):
        self.channels["state"].connect("update", self.stateChanged)
        self.channels["status"].connect("update", self.statusChanged)

    def open(self):
        self.commands['open']()

    def close(self):
        self.commands['close']()

    def statusChanged(self, status):
        self.emit("statusChanged", status)

    def stateChanged(self, state):
        self.emit("stateChanged", state)
```

GUI Bricks

Building blocks for an application

- Meta-widgets
- Python classes inheriting from Framework4.GUI.Core.BaseBrick
- Brick objects are not Qt objects – bricks contain a *brick_widget* member, being Qt's container for the brick
- Bricks contain properties
- A brick can connect to one or several Control Objects
- Each brick specifies its own **connection definitions,** containing a list of expected signals/slots
- Taurus widgets can be used as bricks

GUI Bricks

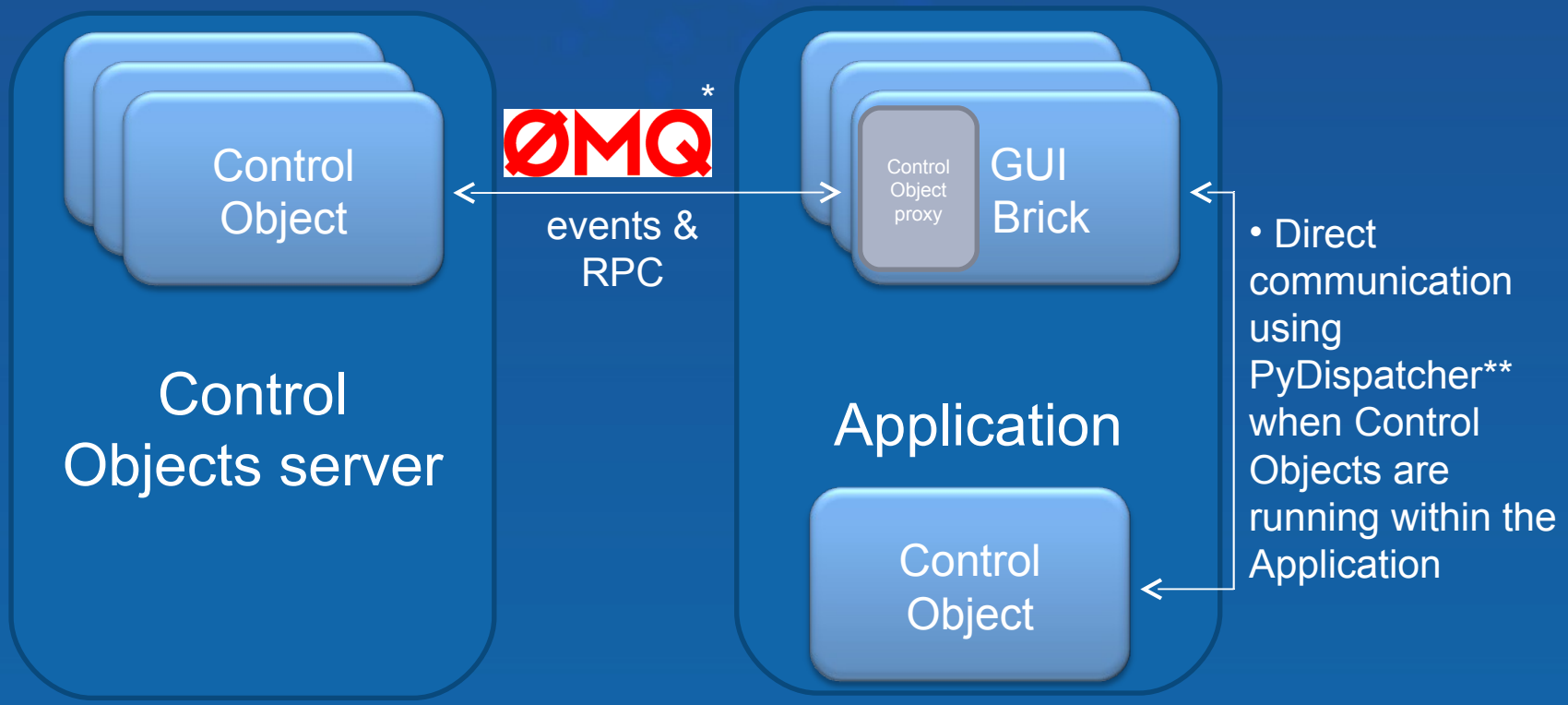
Shutter brick example

```
from Framework4.GUI import Core
from Framework4.GUI.Core import Property, Connection, Signal, Slot

from PyQt4 import Qt, QtGui
import logging

__author__ = "Matias Guijarro"
__version__ = 1.0
__category__ = "General"

class ShutterBrick(Core.BaseBrick):
    description = 'Simple class to display and control a shutter'
    url = ''
    properties = { 'show_button': Property('boolean',
                                           'Show button',
                                           'Allow the user to control the shutter',
                                           'showButtonChanged',
                                           True)
                 , 'orientation': Property('combo',
                                           'Orientation',
                                           onchange_cb = 'orientationChanged',
                                           default = 'Portrait',
                                           choices = [ 'Portrait','Landscape'])}



    connections = {"shutter": Connection("Shutter object",
                                         [ Signal("stateChanged", "shutter_state_changed") ],
                                         [ Slot("open"), Slot("close") ],
                                         "connectionStatusChanged")}
```

Definition of connection : brick expects an object emitting stateChanged signal and with 2 slots « open » and « close »

GUI bricks

Communication between Control Objects and GUI Bricks

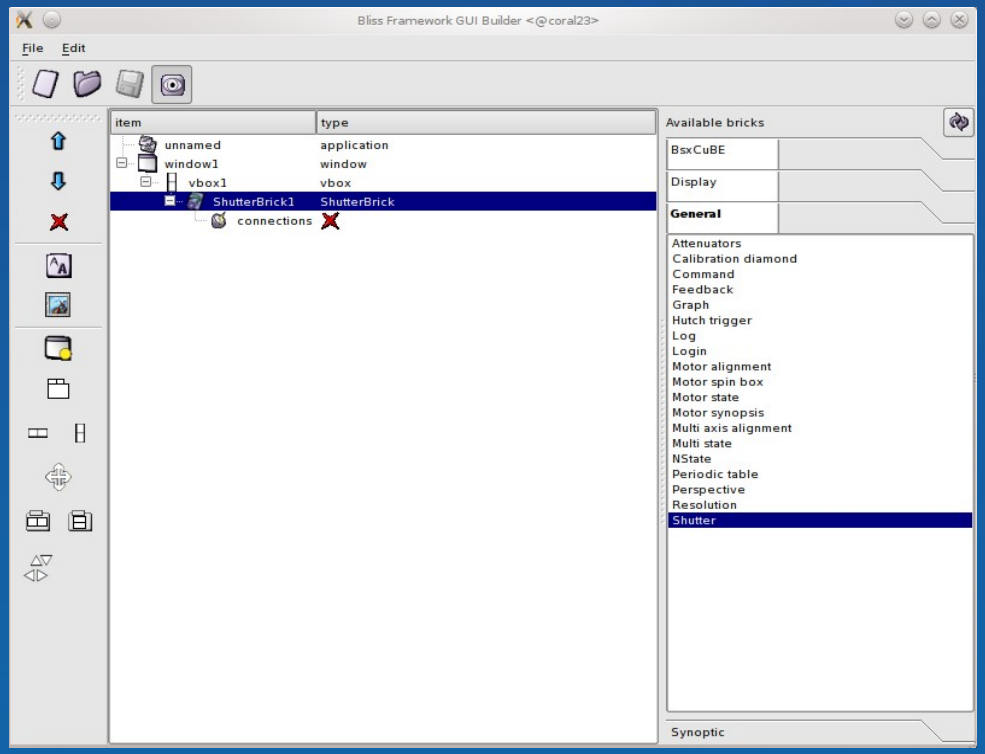


Serialization is done using standard Pickle module from Python

*: http://www.zeromq.org ; **: http://pydispatcher.sourceforge.net/

Application Builder

GUI Builder



- Minimal tree-based GUI editor
- Simple and light
- Allows to create an application, to create a layout and to put bricks into it
- Allows to establish connections between bricks and control objects

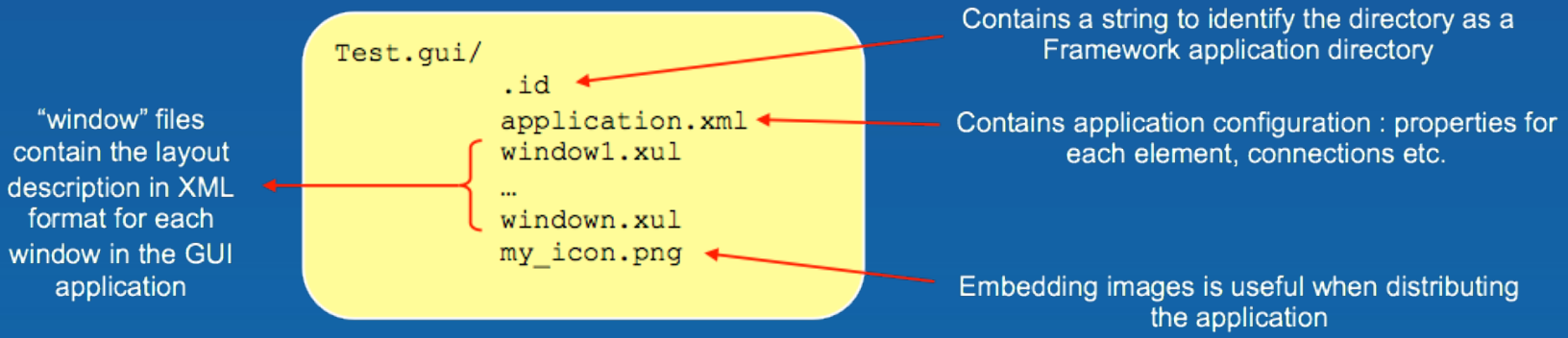
Application Builder

GUI Builder : short demo

Application Builder

Saving an application

- A directory is created for each GUI application
- Layout is separated from properties, connections
- Everything is stored in a human-readable XML format
- Resources (images, icons) can be copied into the directory to stay with the GUI application



Conclusion, future

A tool tailored to our needs

Many Framework 2 applications, only 3 Framework 4 applications : missing new style bricks and control objects

Porting of big applications to be done (mxCuBE...)

New architecture is cleaner, design is better : even if the GUI part changes one day, the Control Objects will remain

Future perspectives

- Control Objects used in Framework GUI applications should really be the same objects used within the beamline experiment sequences (=> towards an integrated beamline software platform, FP7 WP10 goal)
- Remote Access exists in Framework 2 ; for Framework 4, what about having web based applications ?
- Ideally, bricks should not depend on a particular GUI toolkit

The Framework home

Homepage on EPN Forge :

https://forge.epn-campus.eu/projects/show/132

Git repository :

git.epn-campus.eu/repositories/Framework4

Wiki (will be transfered to the EPN Forge page) :

http://fwk.blissgarden.org/

Thanks for your attention

Questions ?