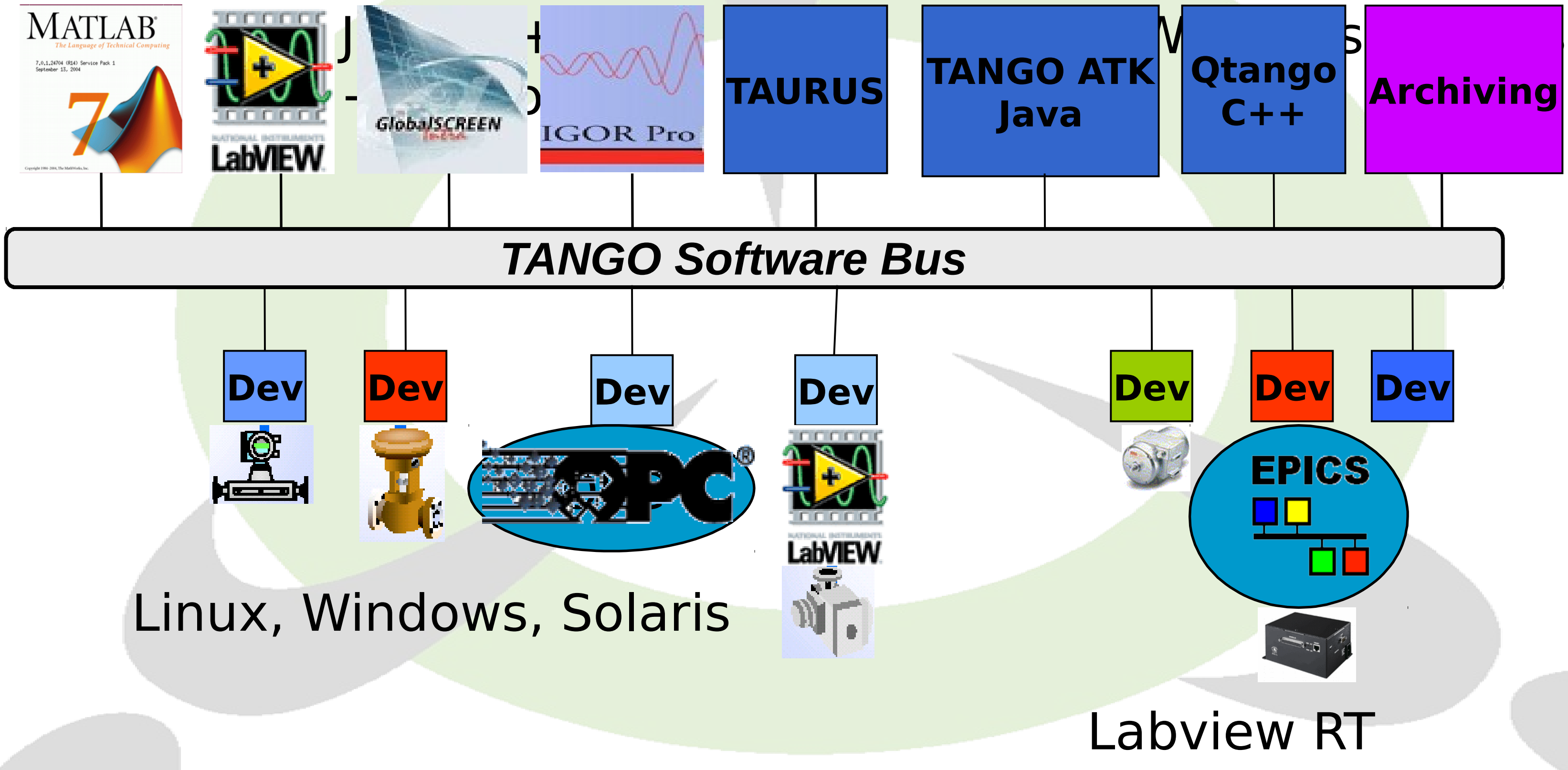
TANGO
Connecting things together

Tango Basics

http://www.tango-controls.org/

What is Tango?

- A software bus for distributed objects



Linux, Windows, Solaris

Labview RT

http://www.tango-controls.org/

What is Tango?

- Provides a unified interface to all equipments, hiding how they are connected to a computer (serial line, USB, sockets….)
- Hide the network
- Location transparency
- Tango is one of the Control Systems available today but other exist (EPICS, Tine, ...)

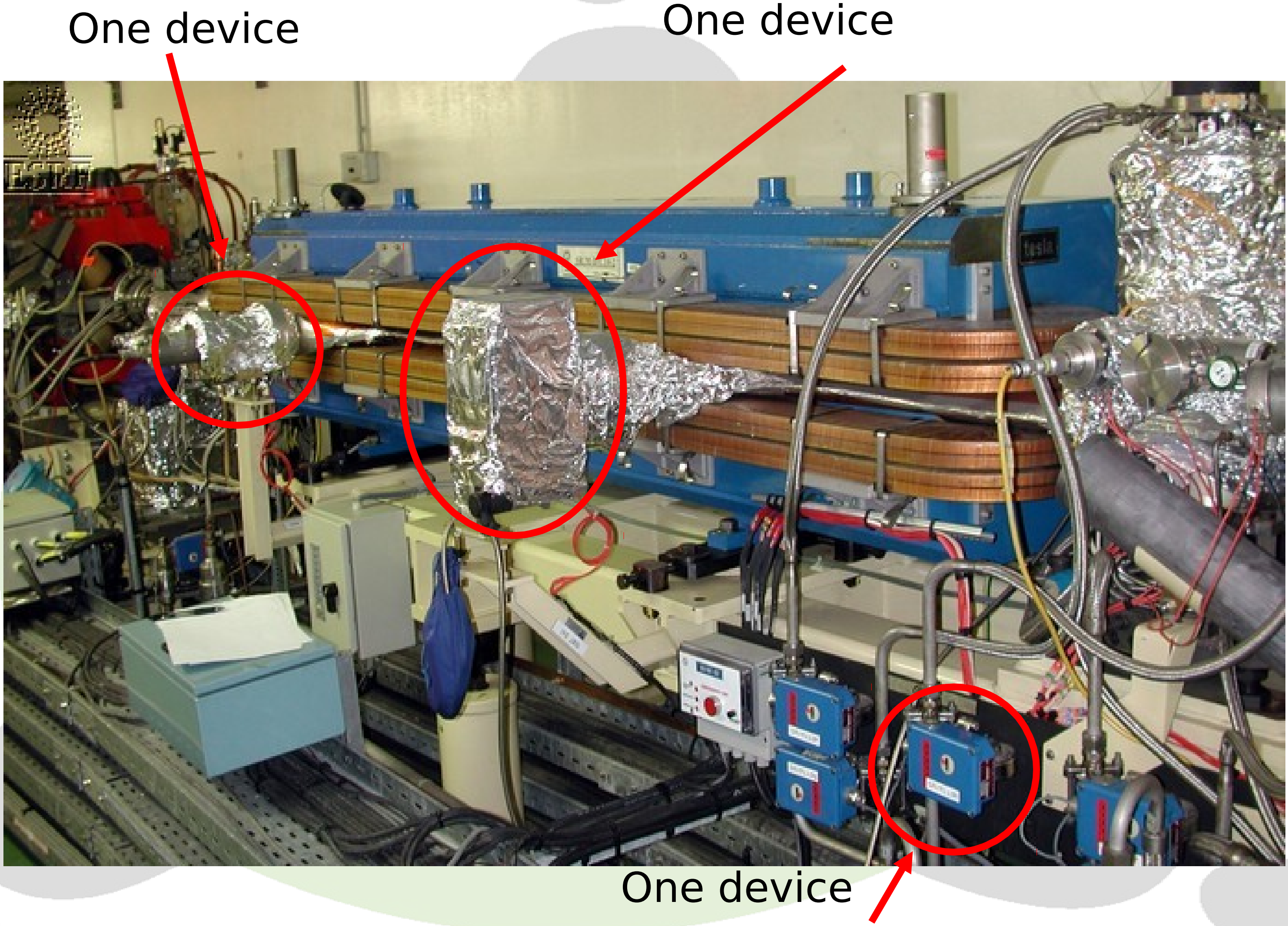
http://www.tango-controls.org/

The Tango Device

- The fundamental brick of Tango is the device!
  - A distributed object exposing an interface
- Everything which needs to be controlled is a “device” from a very simple equipment to a very sophisticated one
- Every device is known by a three field name “domain/family/member”
  - sr/v-ip/c18-1, sr/v-ip/c18-2
  - sr/d-ct/1
  - id10/motor/10, id20/mono/2theta, id20/mirror/exp1

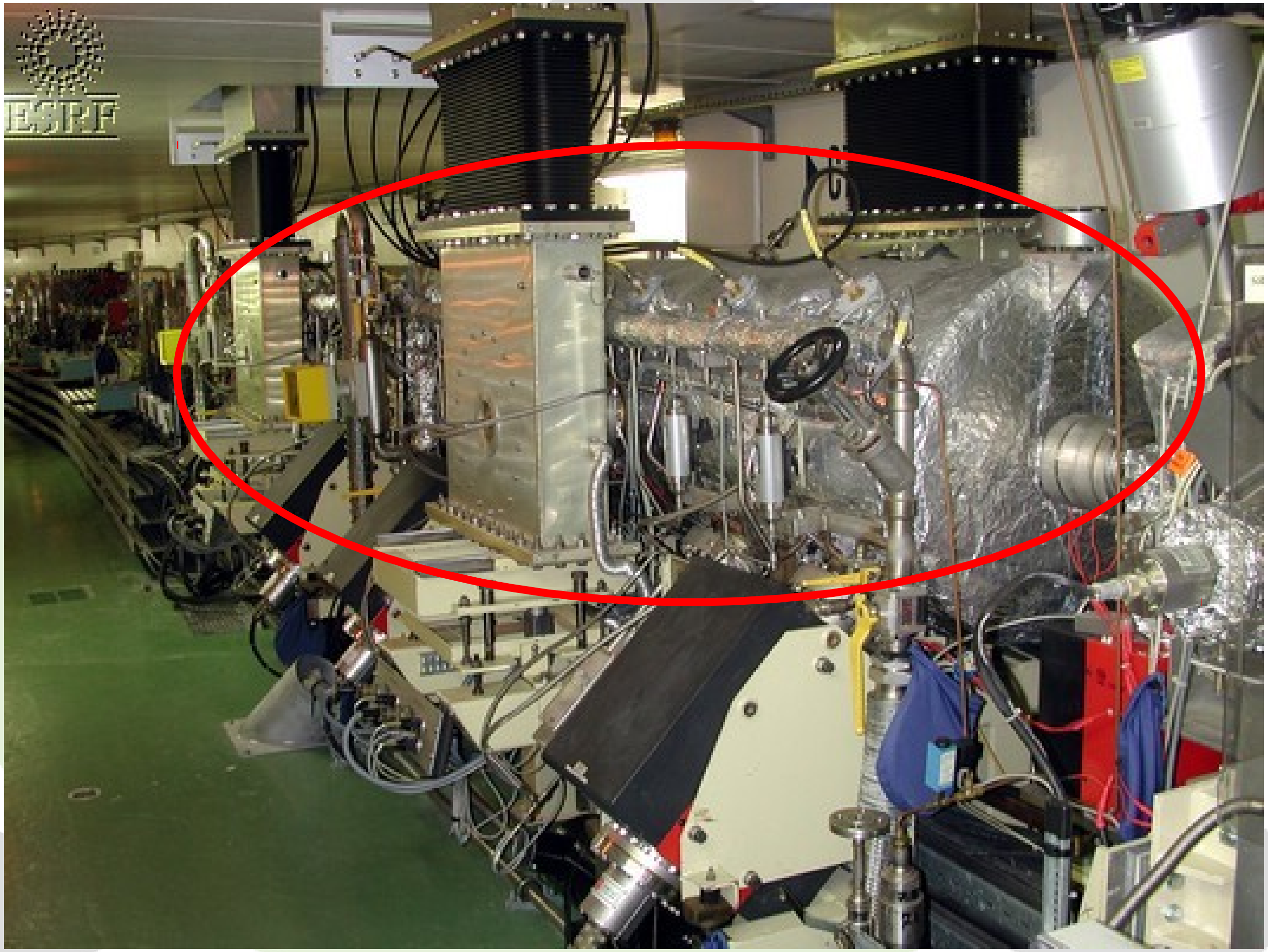
http://www.tango-controls.org/

Some device(s)



http://www.tango-controls.org/

A sophisticated device (RF cavity)



another device

TANGO
Connecting things together

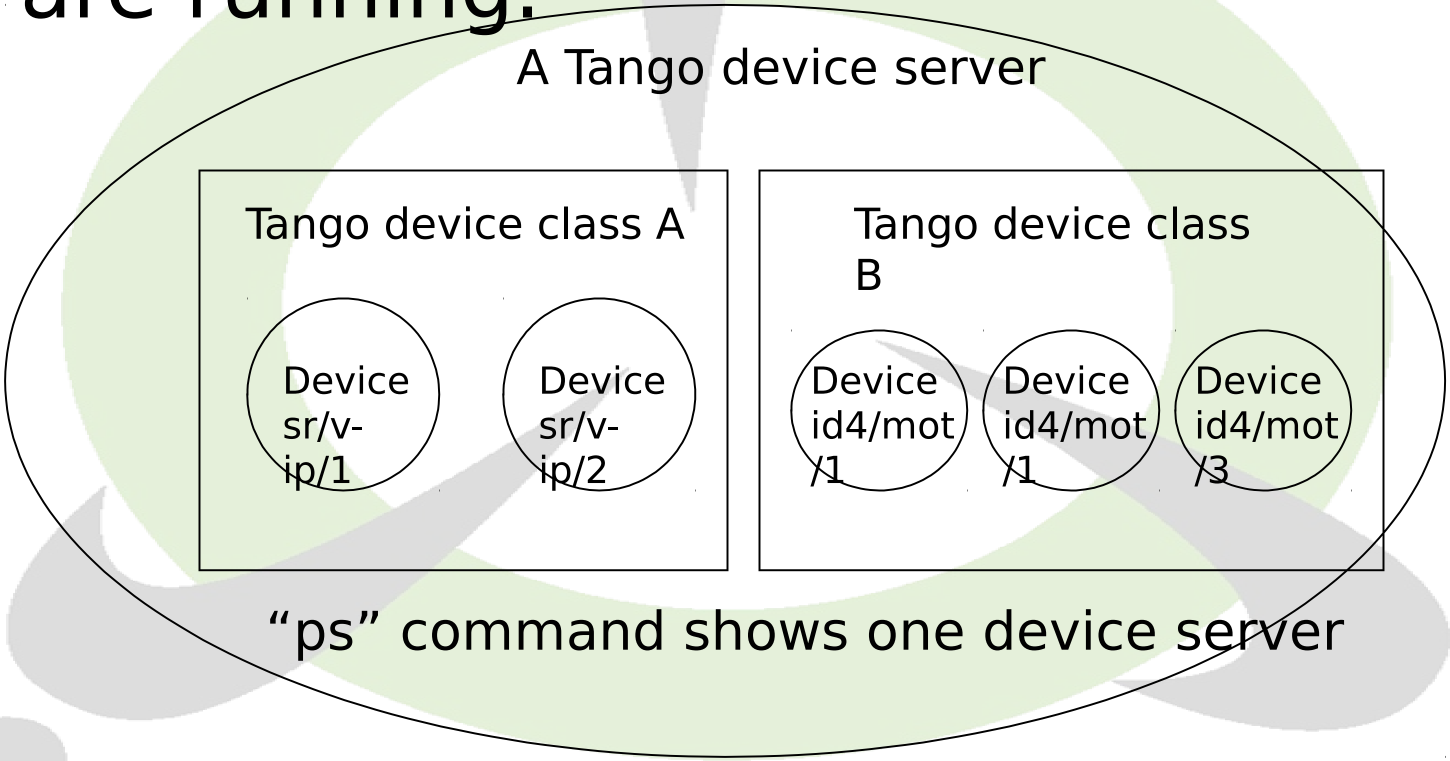
The Tango Class

- Every device belongs to a Tango class (not a computing language class)
- Every device inherits from the same root class (DeviceImpl class)
- A Tango class implements the necessary features to control one kind of equipment
  - Example : The Agilent 4395a spectrum analyzer controlled via its GPIB interface

http://www.tango-controls.org/

The Tango Device Server

- A Tango device server is the process where the Tango class(es) are running.



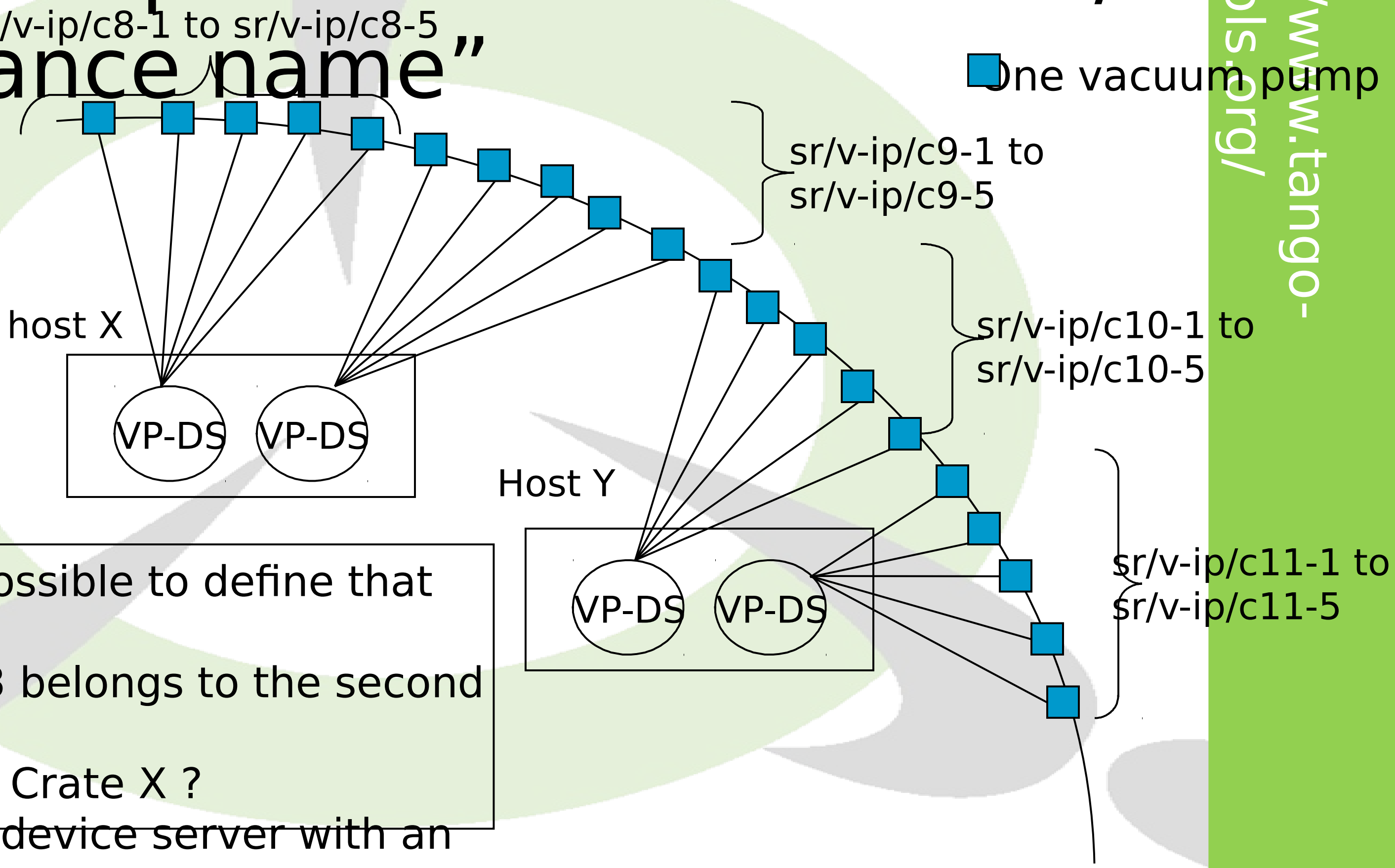
The Tango Device Server

- Tango uses a database to configure a device server process
- Device number and names for a Tango class are defined within the database **not in the code**.
- Which Tango class(es) are part of a device server process is defined in the database but also in the code
  - Classes have to be linked in the executable

http://www.tango-controls.org/

The Tango Device Server

- Each device server is defined by the couple “executable name / instance name”



How is it possible to define that device
sr/v-ip/c9-3 belongs to the second VP-DS
running on Crate X ?
Start each device server with an **INSTANCE NAME**

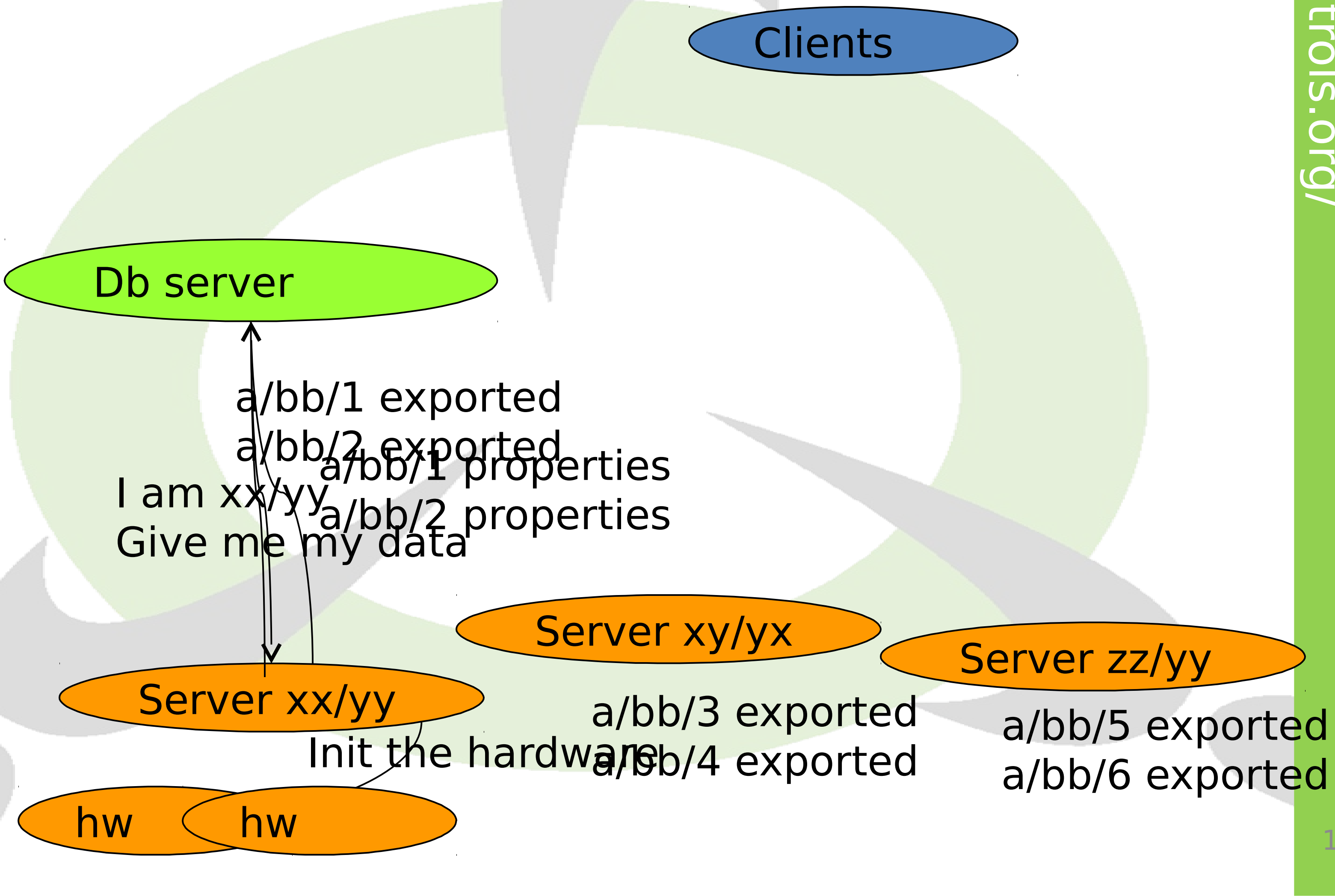
http://www.tango-controls.org/

The Tango Device Server

- During its startup sequence, a Tango device server asks the database which devices it has to create and to manage (number and names)
- Device servers are started like
  - ➢VP-DS c8
  - ➢VP-DS c10

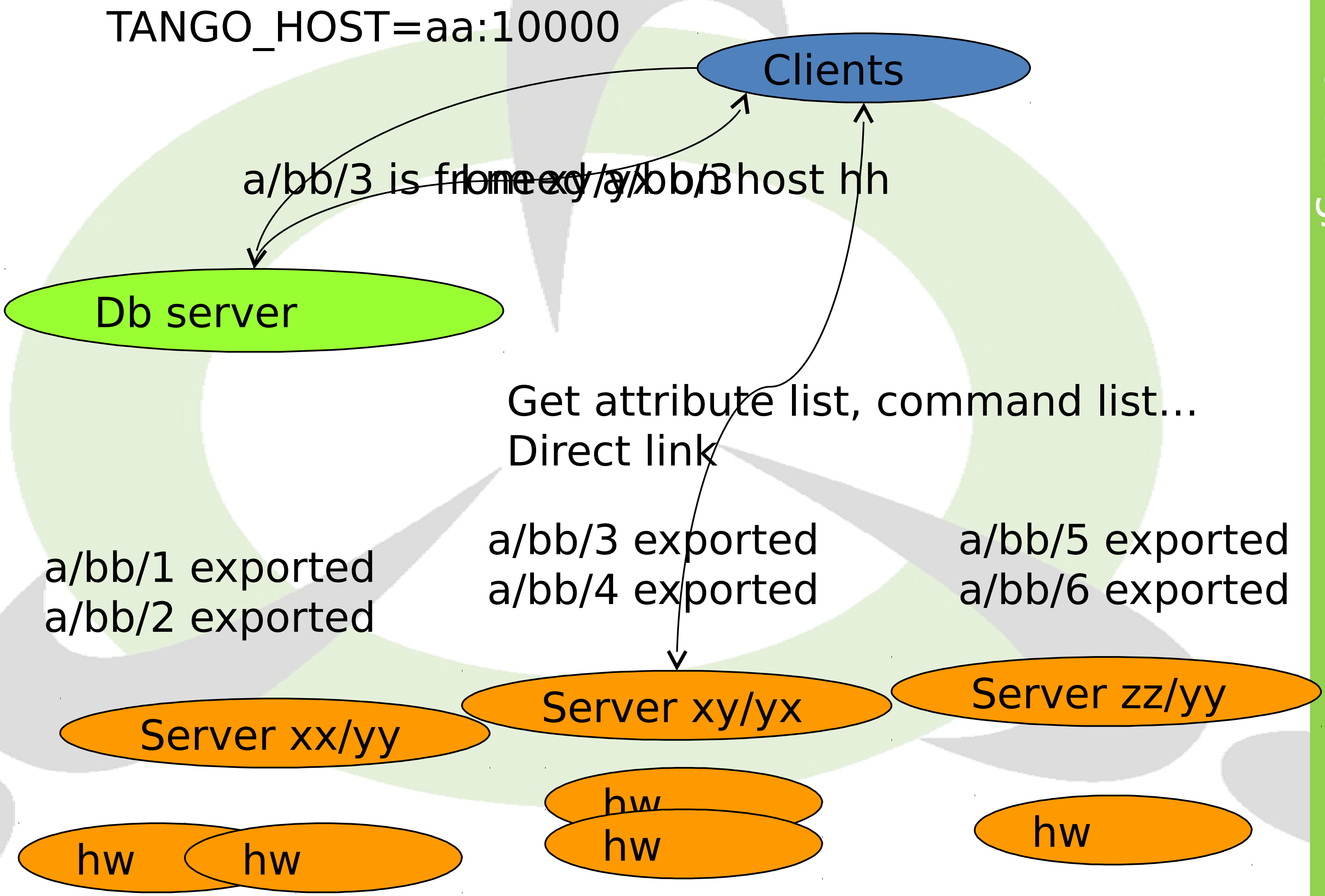
DS exec name	Inst name	Class name	Device name
VP-DS	c8	RibberPump	sr/v-ip/c8-1
VP-DS	c8	RibberPump	sr/v-ip/c8-2
VP-DS	c8	RibberPump	sr/v-ip/c8-3

Device server startup sequence

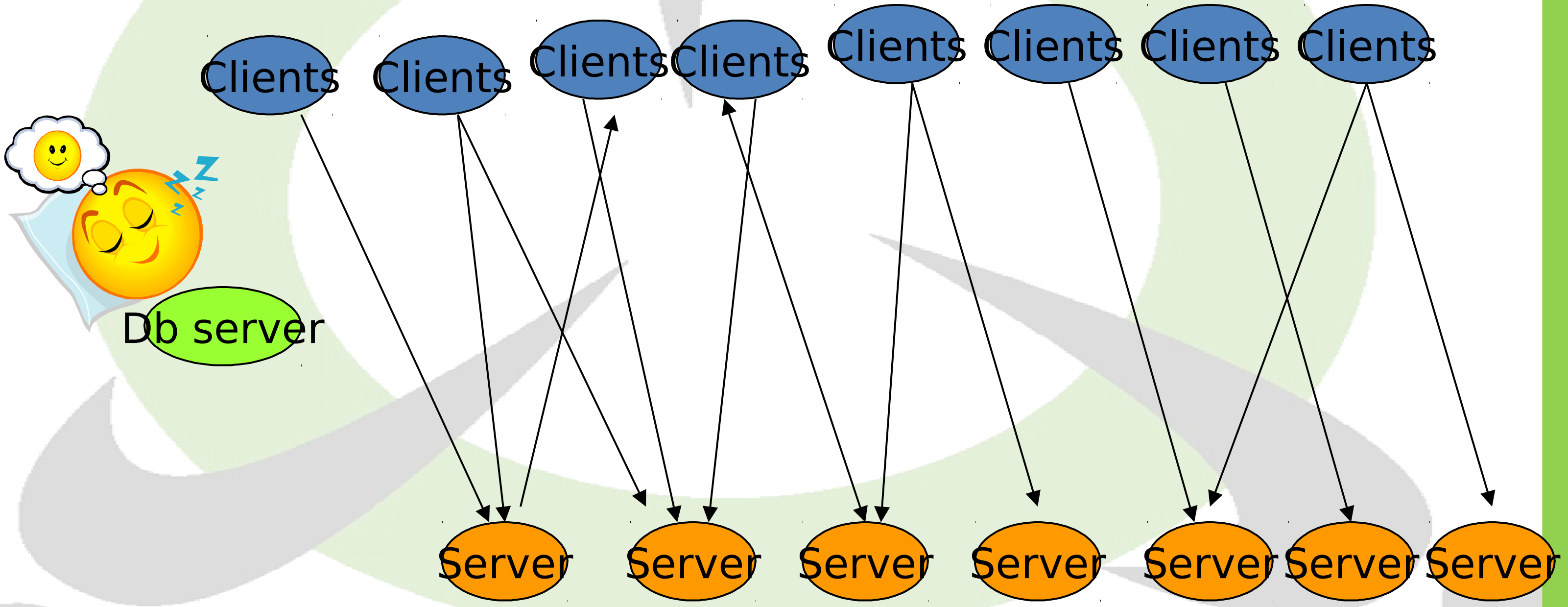


http://www.tango-controls.org/

Device server startup sequence



Steady state situation



http://www.tango-controls.org/

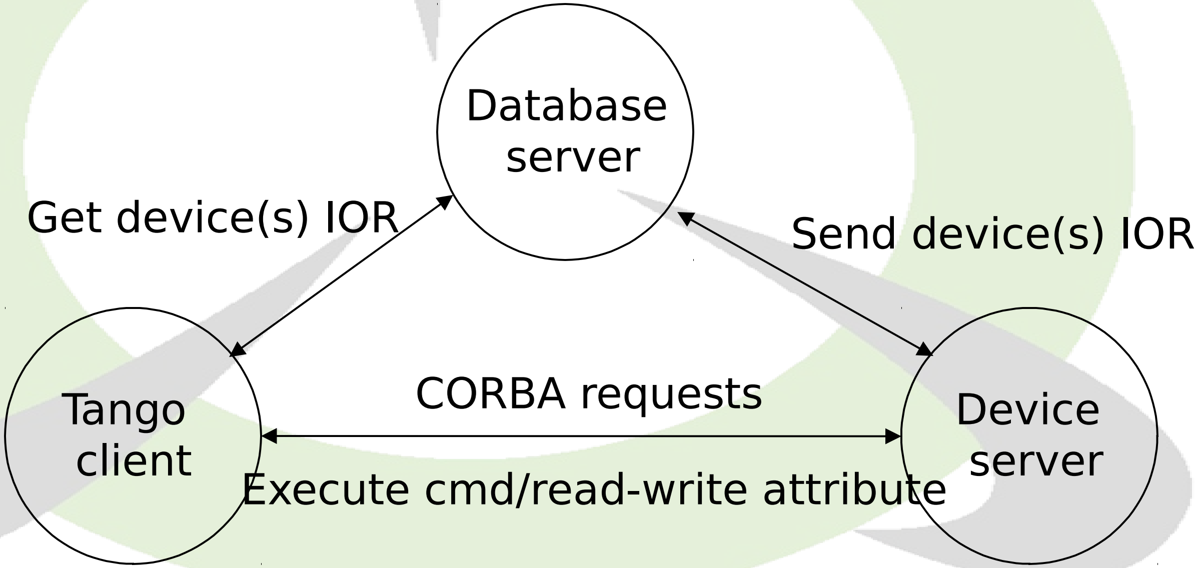
A minimum Tango System

- To run a Tango control system, you need
  - A running MySQL database
  - The Tango database server
    - It is a C++ Tango device server with one device
- To start the database server on a fixed port
- The environment variable **TANGO_HOST** is used by client/server to know
  - On which **host** the database server is running
  - On which **port** it is listening

A minimum Tango System

DataBaseds 2 –ORBendPoint giop:tcp:host:10000

TANGO_HOST=host:port (Ex : TANGO_HOST=orion:10000)



http://www.tango-controls.org/

Demo jive

- Device servers
- Devices
- Classes
- Admin devices

STARTER

- Watch admin slides
- Demo astor

http://www.tango-controls.org/

Tango Basics:

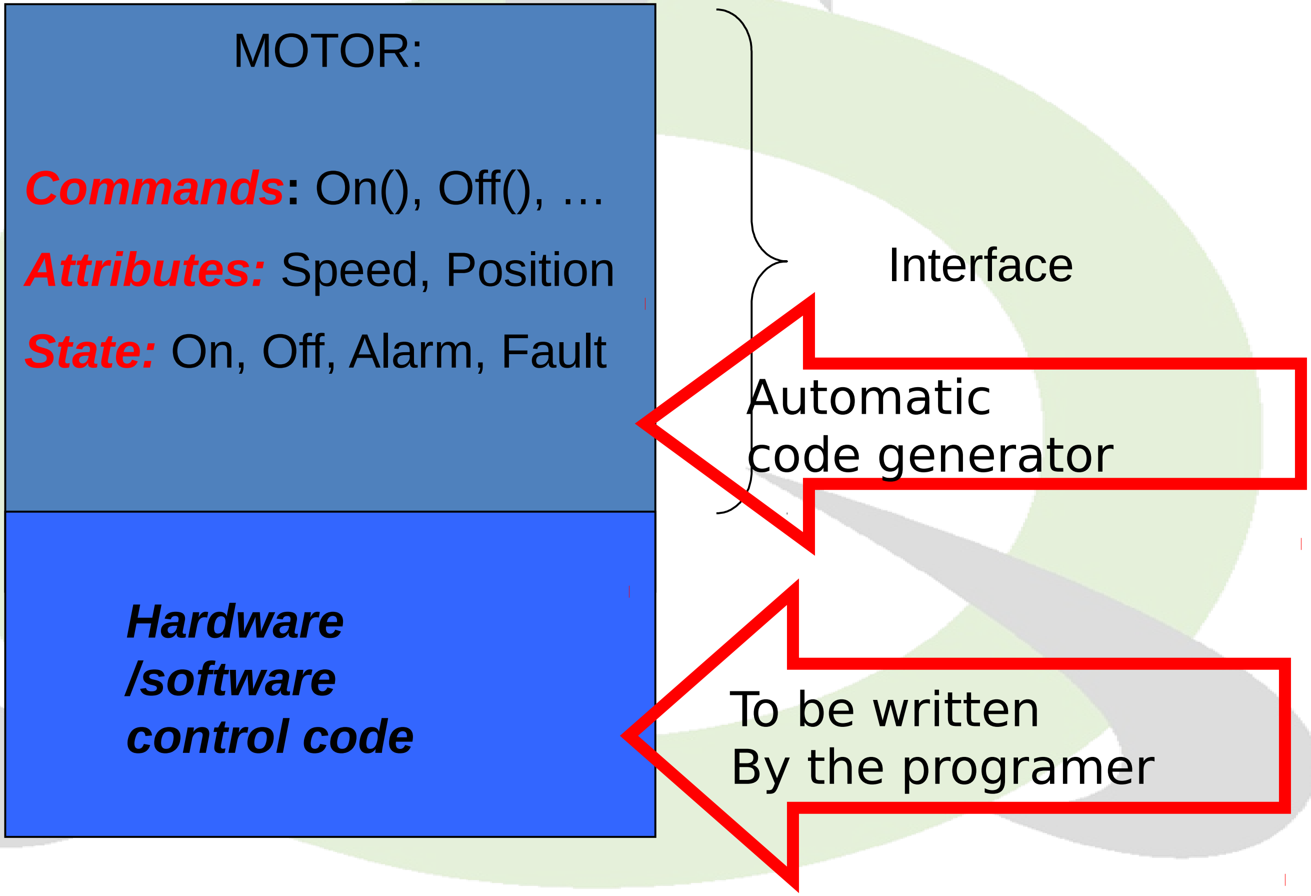
a device server

- Commands
- Attributes
- States
- Properties

http://www.tango-controls.org/

TANGO devices

Example: motor interface:



MOTOR:

***Commands*: On(), Off(), …**

Attributes: Speed, Position

State: On, Off, Alarm, Fault

Hardware /software control code

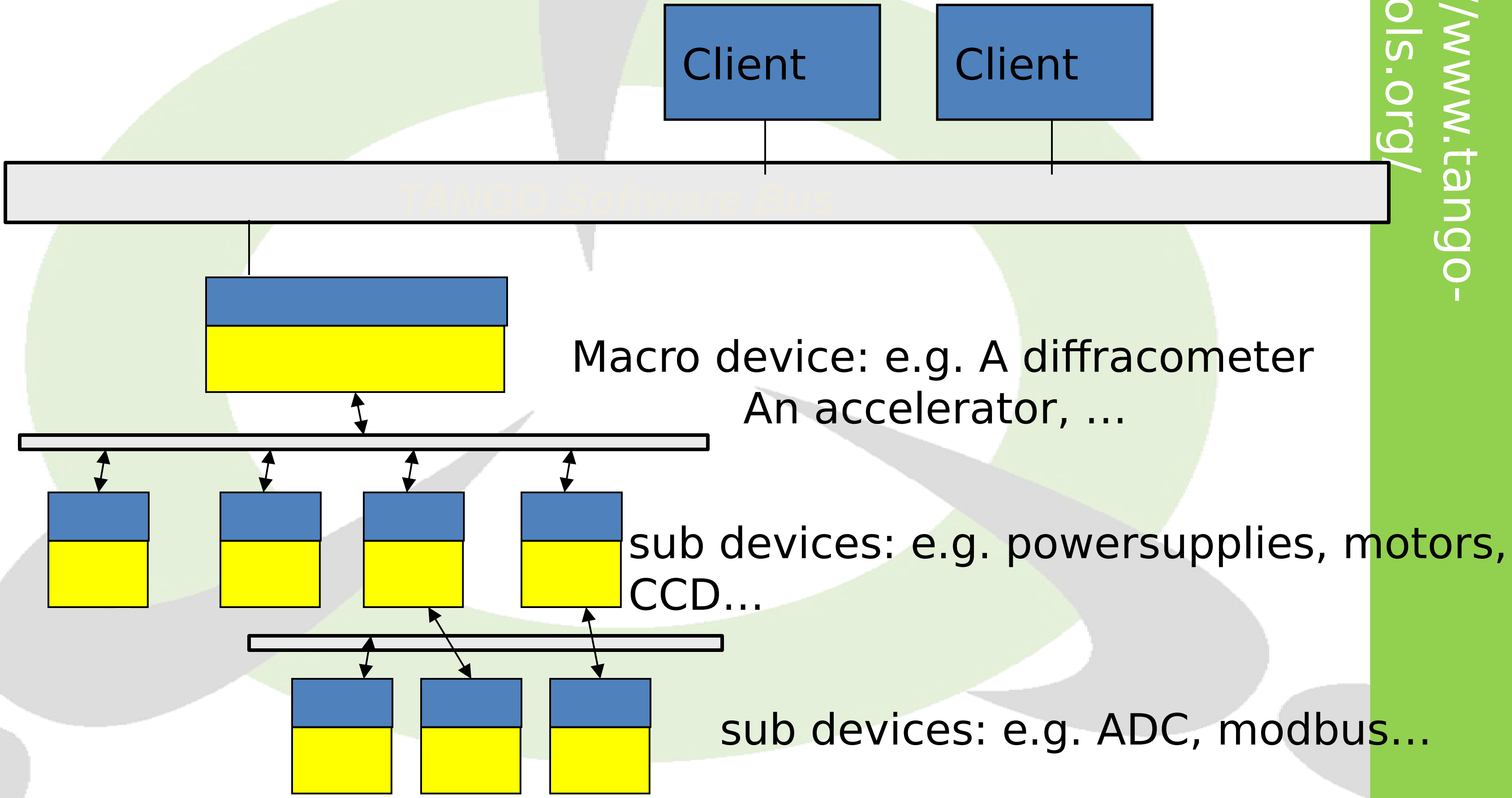
Interface

Automatic code generator

To be written
By the programer

TANGO devices

- 1 Device can also interface complex systems
  - Hierarchical structure



Macro device: e.g. A diffracometer
An accelerator, ...

sub devices: e.g. powersupplies, motors, CCD…

sub devices: e.g. ADC, modbus…

Commands & Attributes

- On the network a Tango device mainly has
  - **Command**(s): Used to implement “action” on a device (switching ON a power supply)
  - **Attribute**(s): Used for physical values (a motor position, a temperature, a spectrum, an matrix)
- Clients ask Tango devices to execute a command or read/write one of its attributes
- A Tango device also has a **state** and a **status** which are available using command(s) or as attribute(s)

Commands

- A command may have one input and one output argument.
- A limited set of argument data types are supported
  - Boolean, short, long, long64, float, double, string, unsigned short, unsigned long, unsigned long64, array of these, 2 exotic types and State data type

http://www.tango-controls.org/

Attributes

- Self describing data via a configuration
- Thirteen data types supported:
  - Boolean, unsigned char, short, unsigned short, long, long64, unsigned long, unsigned long64, float, double, string, state and DevEncoded data type
- Three accessibility types
  - Read, write, read-write
- Three data formats
  - Scalar (one value), spectrum (an array of one dimension), image (an array of 2 dimensions)

http://www.tango-controls.org/

Attributes

- When you read an attribute you receive:
  - The attribute data (luckily…)
  - An attribute quality factor
    - ATTR_VALID, ATTR_INVALID, ATTR_CHANGING, ATTR_ALARM, ATTR_WARNING
  - The date when the attribute was acquired by the server (number of seconds and usec since EPOCH)
  - Its name
  - Its dimension, data type and data format
- When you write an attribute, you send
  - The attribute name
  - The new attribute data

DEMO test device

- Attributes
- Attribute properties, quality factors...
- Pure software devices.

Attribute Configuration

- Attribute configuration defined by its properties
  - Five type of properties
    - Hard-coded
    - Modifiable properties
      - GUI parameters
      - Max parameters
      - Alarm parameters
      - Event parameters
- A separate network call allows clients to get attribute configuration (get_attribute_config)

Attribute Configuration

- The hard coded attribute properties (5)
  - name
  - data_type
  - data_format
  - writable
  - display level

Attribute Configuration

- The GUI attribute properties (6)
  - Description
  - Label
  - Unit
  - Standard_unit
  - Display_unit
  - Format (C++ or printf)
- The Maximum attribute properties (used only for writable attribute) (2)
  - min_value
  - max_value

http://www.tango-controls.org/

Attribute Configuration

- The alarm attribute properties (6)
  - min_alarm, max_alarm
  - min_warning, max_warning
  - delta_t, delta_val
- The event attribute properties (6)
  - period (for periodic event)
  - rel_change, abs_change (for change event)
  - period, rel_change, abs_change (for archive event)

http://www.tango-controls.org/

Demo atkpanel

- Get attribute list
- Get attribute config
- Get command list
- Etc...

http://www.tango-controls.org/

States

- A limited set of 14 device states is available.
  - ON, OFF, CLOSE, OPEN, INSERT, EXTRACT, MOVING, STANDBY, FAULT, INIT, RUNNING, ALARM, DISABLE and UNKNOWN

Properties

- Properties are stored in the MySQL database
- No file – Use Jive to create/update/delete properties
- You can define properties at
  - Class level, device level and attribute level
- Property data type
  - Basic data types as scalar or array values

http://www.tango-controls.org/

Demo jive

- Device properties
- Class properties

Automatically added Commands & Attributes

- Three commands are automatically added
  - **State** : In = void Out = DevState
    - Return the device state and check for alarms
    - Overwritable
  - **Status** : In = void Out = DevString
    - Return the device status
    - Overwritable
  - **Init** : In = void Out = void
    - Re-initialise the device (delete_device + init_device)
- Two attributes are automatically added
  - State and Status

Design a device DEMO Pogo icepap

- Alarm level
- Attribute properties
- Expert/operator
- Memorized attribute
- Inheritance
- ...

Demo debug

- Compile
- Add in Starter
- wizard
- Log viewer

http://www.tango-controls.org/

Tango Basics: The Client API

- Synchronous Calls
- Error management
- Asynchronous Calls
- Group Calls
- Events

Synchronous Calls

- On the client side, each Tango device is an instance of a **DeviceProxy** class
- DeviceProxy class
  - Hide connection details
  - Manage re-connection
- The DeviceProxy instance is created from the device name

```
C++ : Tango::DeviceProxy dev("id13/v-pen/12");
```

```
Python : PyTango.DeviceProxy dev("id13/v-pen/12");
```

http://www.tango-controls.org/

Synchronous Calls

- The DeviceProxy *command_inout()* method sends a command to a device
- The class DeviceData is used for the data sent/received to/from the command.

DeviceData DeviceProxy::command_inout (const char *, DeviceData &);

DeviceProxy.command_inout (name, cmd_param)

```
Tango::DeviceProxy dev(“sr/v-pen/c1”);
Tango::DeviceData d_in,d_out;
vector<long> v_in,v_out;

d_in << v_in;
d_out =
dev.command_inout(“MyCommand”,d_in);
d_out >> v_out;
```

```
dev = PyTango.DeviceProxy(“sr/v-pen/c1”)

dev.command_inout(‘On’)
dev.on()

print dev.command_inout(‘EchoShort’,10)
print dev.EchoShort(10)
```

http://www.tango-controls.org/

Synchronous Calls

- The DeviceProxy *read_attribute()* method reads a device attribute (or *read_attribute**s**()*)
- The class DeviceAttribute is used for the data received from the attribute.

```
DeviceAttribute DeviceProxy::read_attribute(string &);
```

```
DeviceAttribute DeviceProxy.read_attribute(name);
```

```
Tango::DeviceProxy dev(“sr/v-pen/c1”);
Tango::DeviceAttribute da;
float press;
string att_name(“Pressure”);

da = p_dev->read_attribute(att_name);
da >> press;
```

```
dev = PyTango.DeviceProxy(‘sr/v-pen/c1’)
da = dev.read_attribute(‘Pressure’)
print da.value

print dev[‘SpecAttr’].value
seq_da =
dev.read_attributes([‘SpecAttr’,’Pressure’])
```

Synchronous Calls

- The DeviceProxy *write_attribute()* method writes a device attribute (or *write_attribute**s**()*)

```
void DeviceProxy::write_attribute(DeviceAttribute &);
```

```
DeviceProxy.write_attribute(name, value)
```

```
Tango::DeviceProxy
dev(“id2/motor/1);
long spe =  102;
Tango::DeviceAttribute
da(“Speed”, spe);

dev.write_attribute(da);
```

```
dev = PyTango.DeviceProxy(‘et/s_lift/1)
dev.write_attribute(‘SpecAttr’,[2,3])

dev.write_attribute(‘SpecAttr’,
numpy.array([6,7]))

dev[‘SpecAttr’] = [3,4]
dev.write_attributes(([‘Speed’,5],
[‘SpecAttr’,[2,3]]))
```

http://www.tango-controls.org/

Synchronous Calls

- Many methods available in the DeviceProxy class
  - ping, info, state, status, set_timeout_millis, get_timeout_millis, attribute_query, get_attribute_config, set_attribute_config.....
- If you are interested only in attributes, use the **AttributeProxy** class

http://www.tango-controls.org/

Error Management

- All the exception thrown by the API are PyTango.DevFailed exception
- One catch (except) block is enough
- Ten exception classes (inheriting from DevFailed) have been created
  - Allow easier error filtering
- These classes do not add any new information compared to the DevFailed exception

Error Management

- An example

```
try {
     Tango::AttributeProxy
ap("id18/pen/2/Press");
     Tango::DeviceAttribute da;

      da = ap.read();
      float pre;
      da >> pre;
}
catch (Tango::WrongNameSyntax &e) {
      cout << "Et couillon, faut 3 / !" << endl;
}
catch (Tango::DevFailed &e) {
     Tango::Except::print_exception(e);
}
```

```
try:
        att = PyTango.AttributeProxy('d18/pen/2/Pres')
        print att.read()
except PyTango.WrongNameSyntax:
                print 'Et couillon, faut 3 / !'
except PyTango.DevFailed,e:
                PyTango.Except.print_exception(e)
```

http://www.tango-controls.org/

Asynchronous Calls

- Asynchronous call :
  - The client sends a request to a device and does not block waiting for the answer.
  - The device informs the client process that the request has ended
- Does not request any changes on the server side
- Supported for
  - command_inout
  - read_attribute(s)
  - write_attribute(s)

Group Calls

- Provides a single point of control for a Group of devices
- **Group calls are executed asynchronously!**
- You create a group of device(s)
- You execute a command (or R/W attribute) on the group

http://www.tango-controls.org/

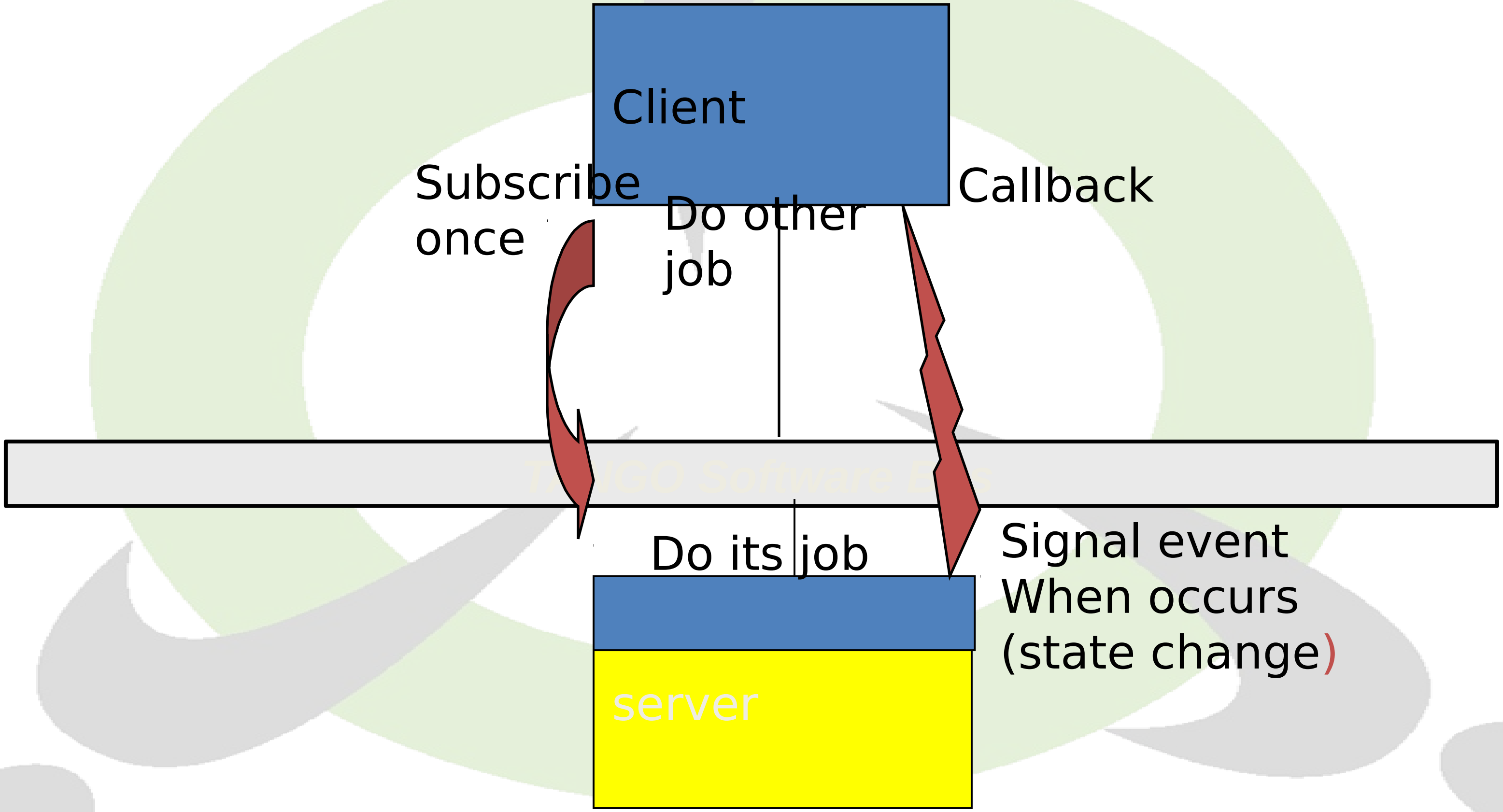
Group Calls

- Using groups, you can
  - Execute one command
    - Without argument
    - With the same input argument to all group members
    - With different input arguments for group members
  - Read one attribute
  - Write one attribute
    - With same input value for all group members
    - With different input value for group members
  - Read several attributes

http://www.tango-controls.org/

TANGO Communication

- Event Driven



Events

- Another way to write applications
  - Applications do not poll any more
  - The device server informs the applications that “something” has happened
- Polling done by the device server polling thread(s)

http://www.tango-controls.org/

Events

- Until tango v7 One Notification service daemon (notifd) running on each host
- Event propagation
  - The event is sent from the server to the notification service
    - When detected by the polling thread(s)
    - On request in the code (push_event() call family)
  - The notification service sends the event to all the registered client(s)
- Since V8 Server sends itself events via ZMQ

http://www.tango-controls.org/

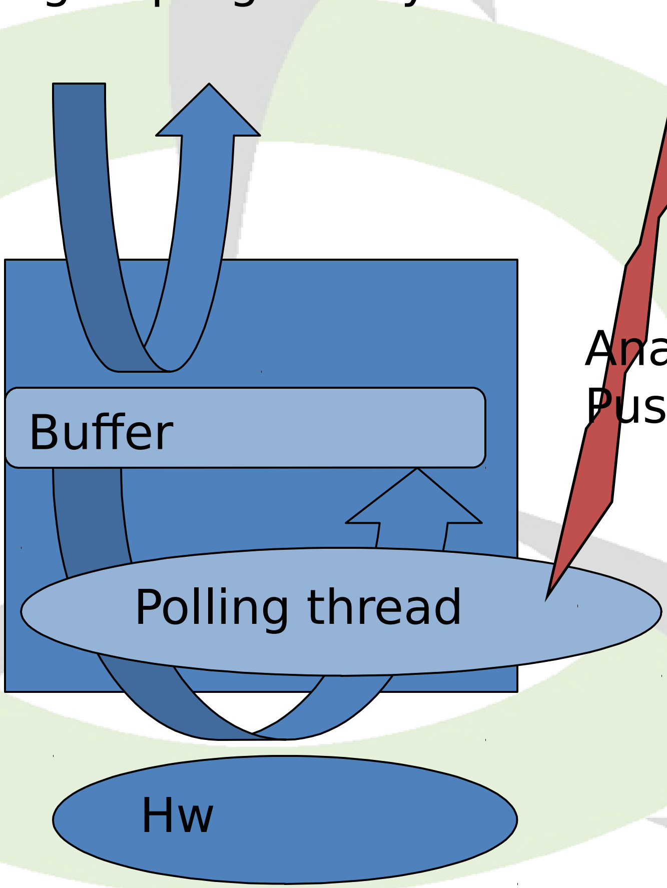
Events

- Only available on attributes!
- Does not requires any changes in the device server code
- Based on callbacks. The client callback is executed when an event is received
  - Event data or an error stack in case of an exception
- 6 types of events
  - Periodic, Change, Archive
  - Attribute configuration change, Data ready
  - User defined

http://www.tango-controls.org/

Events: inside the server

Nothing to program by the server developer



Analyze buffer history
Push Event

http://www.tango-controls.org/

Demo jive

- Polling
- Events
- Properties
- Attribute config
- atkpanel
- Device test

http://www.tango-controls.org/

Events (client side)

- Event subscription with the *DeviceProxy.subscribe_event()* method
- Event un-subscription with the DeviceProxy.*unsubscribe_event()* method
- Call-back idem to asynchronous call
- Already implemented in ATK and TAURUS

http://www.tango-controls.org/