YARP

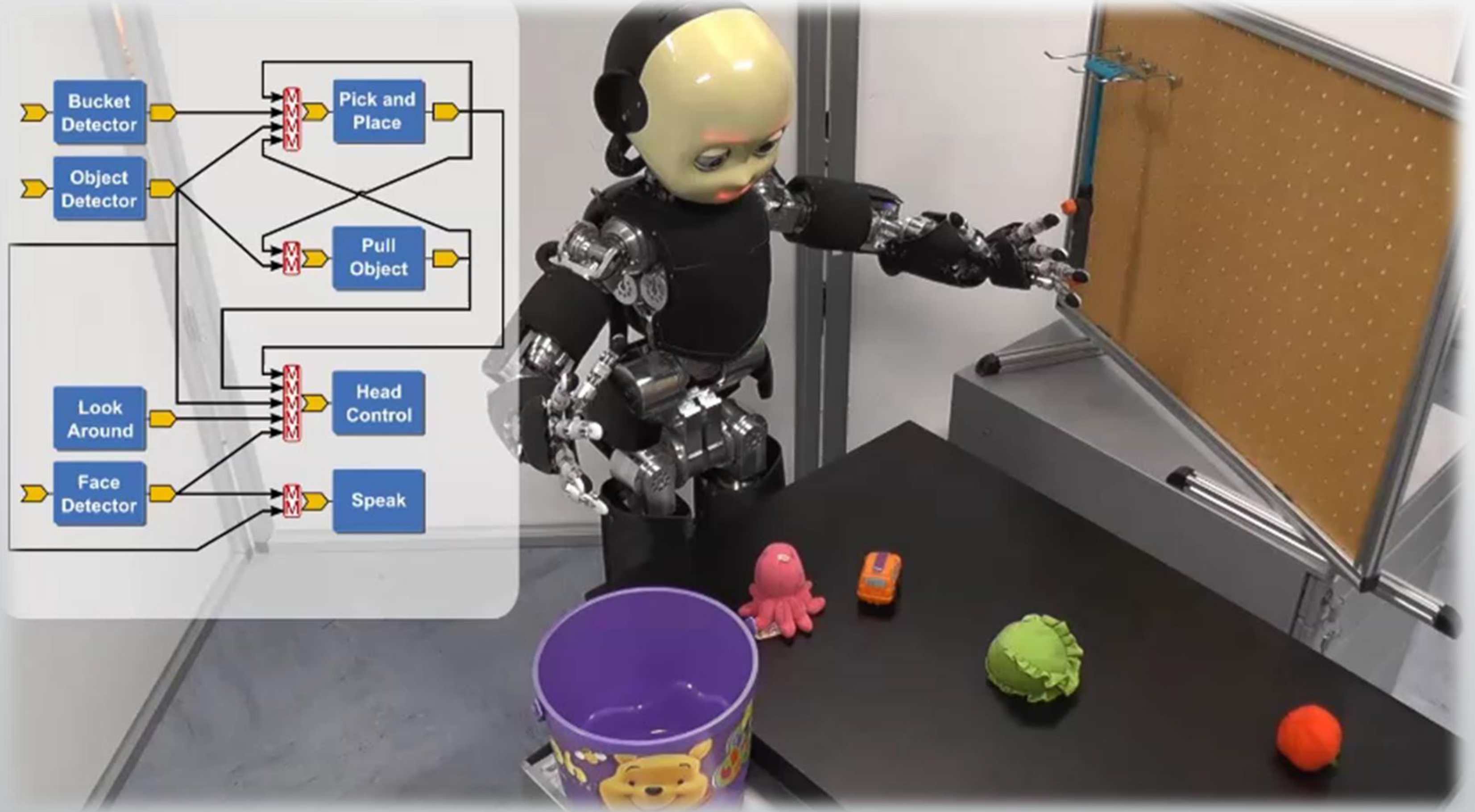
Yet Another Robot Platform

Summary

- What is YARP?
- YARP Ports
- YARP Devices
- YARP Tools
- Other YARP features

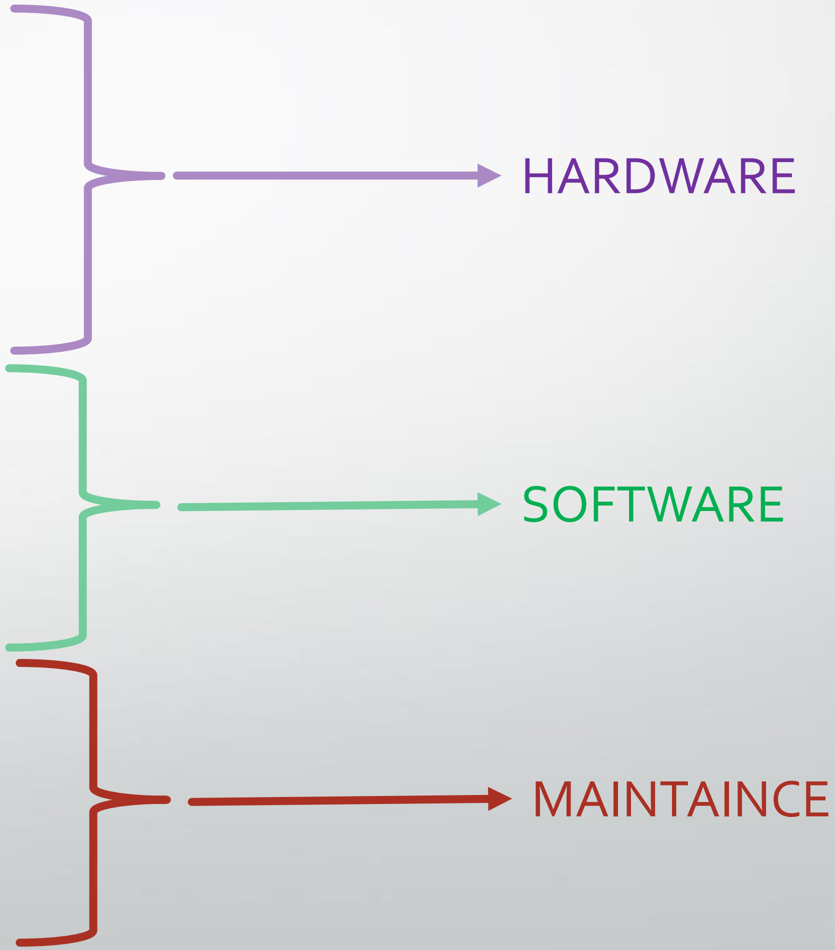
What is YARP?

Let’s start from the end – Why?



Why do we need a framework?

- Various scenarios and platforms
- Hardware changes in time
- Lots of different sensors
- Lack of standards
- Distributed processing
- Real-time friendly
- Algorithms/libraries/code changes in time
- Inherent complexity
- Distributed development
- Short life span of projects



HARDWARE

SOFTWARE

MAINTAINCE

What is YARP?

YARP is a middleware aimed to ease the development of high level application for robots with a strong focus on modularity, code re-usage, flexibility and hw/sw abstraction.

Homogeneous set of libraries,
GUIs, tools,
debug and run facilities

What is YARP?

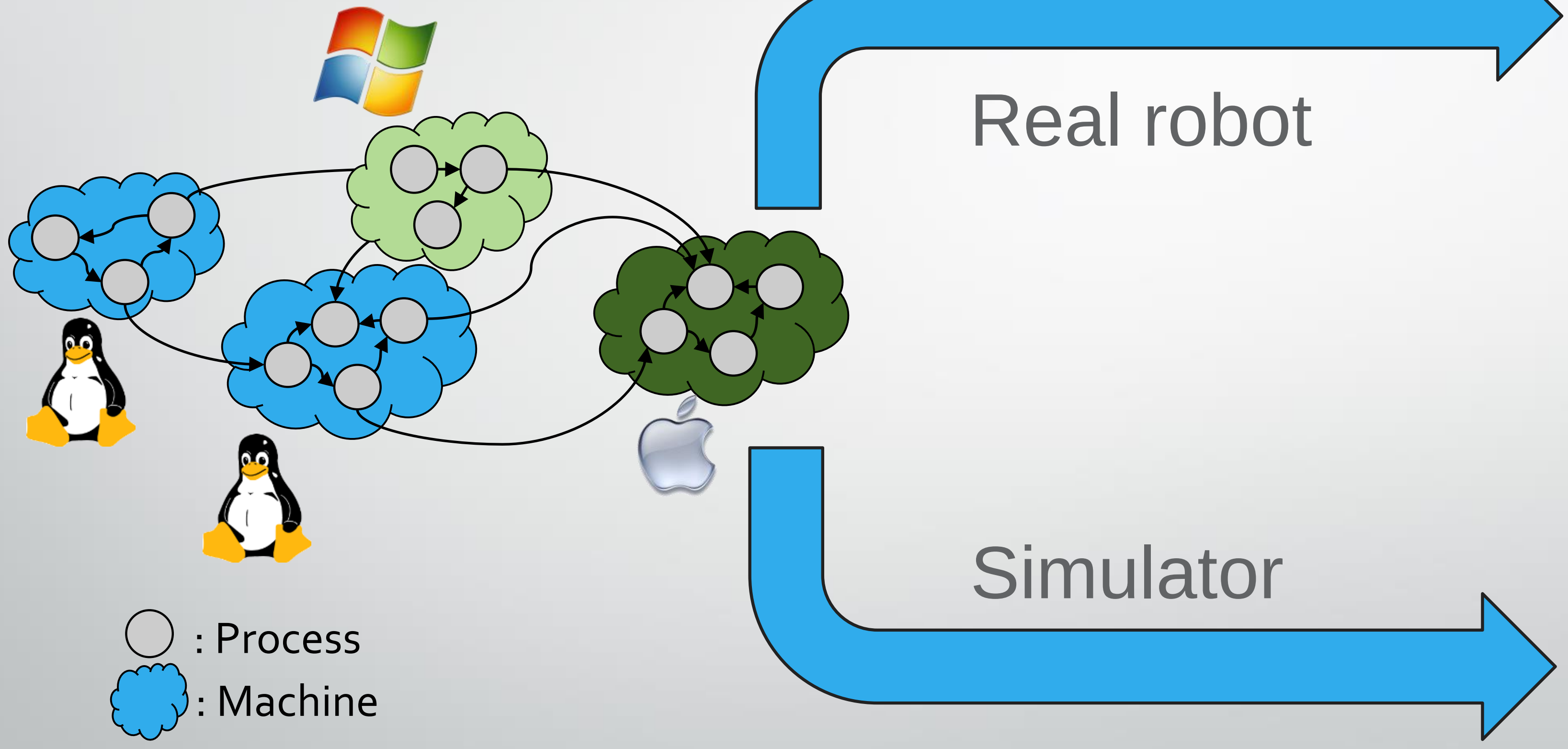
YARP is a middleware aimed to ease the development of high level application for robots with a strong focus on modularity, code re-usage, flexibility and hw/sw abstraction.

YARP has been designed to support building robot control systems as collection of executables communicating in a peer-to-peer way, with an extensible types of connections (tcp, udp, multicast, local, MPI, mjpeg, XML/RPC, tcpros, …).

YARP has been historically a C++ library targeting C++ users, but it has also bindings for high-level languages such as Python.

The strategic goal of this kind of design is to increase the longevity of robot software projects.

Typical application

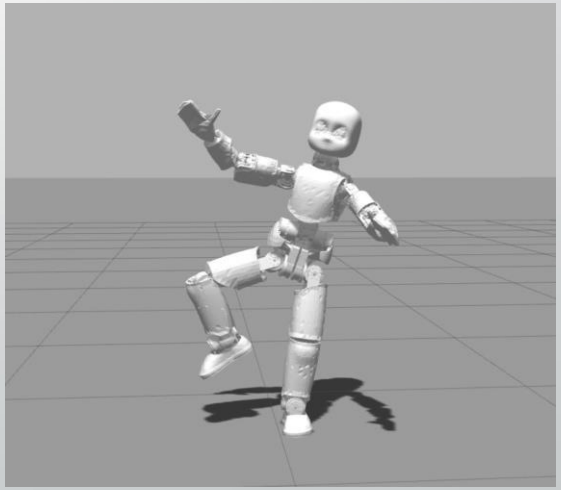


Real robot

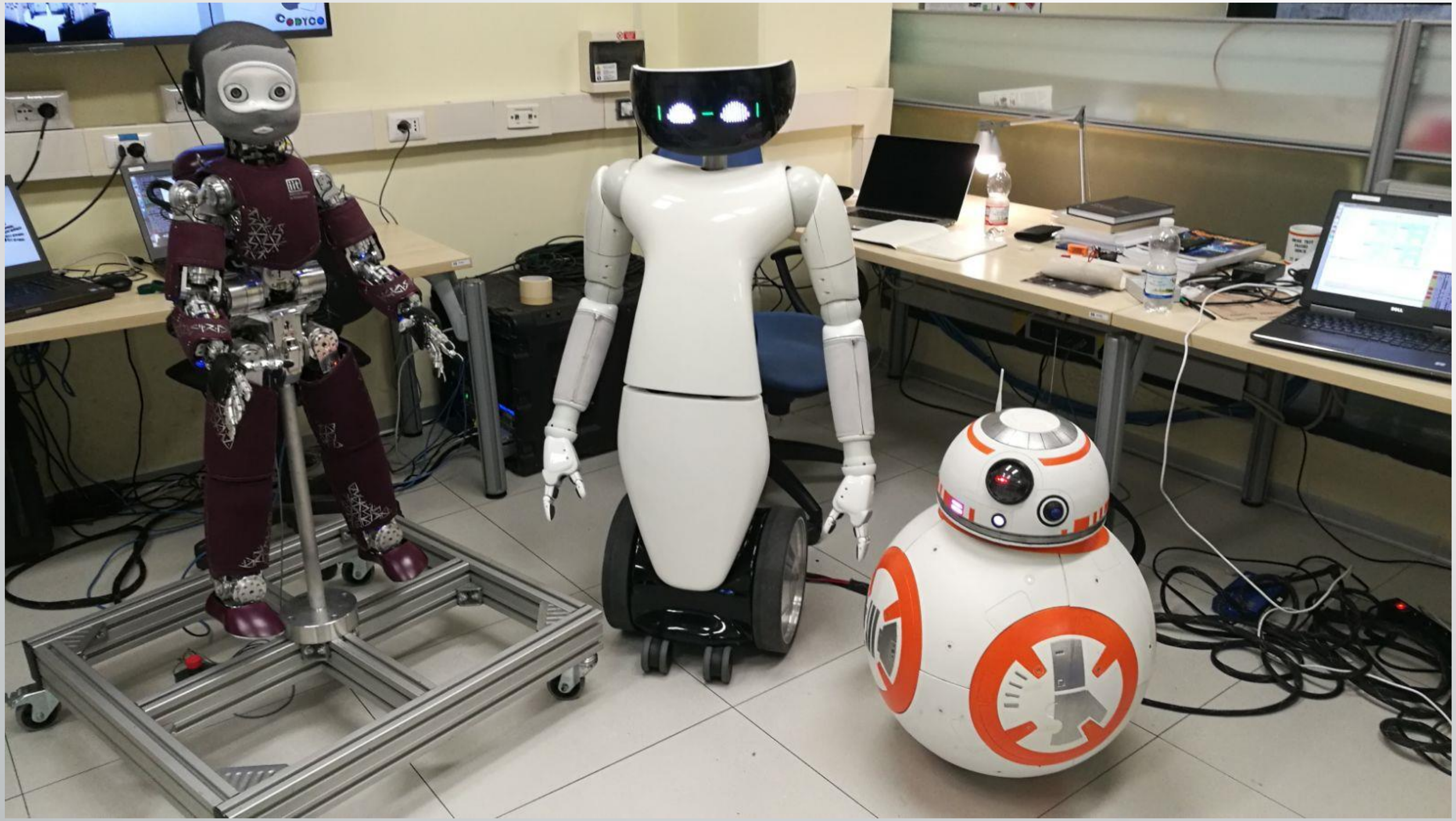
Simulator

: Process

: Machine

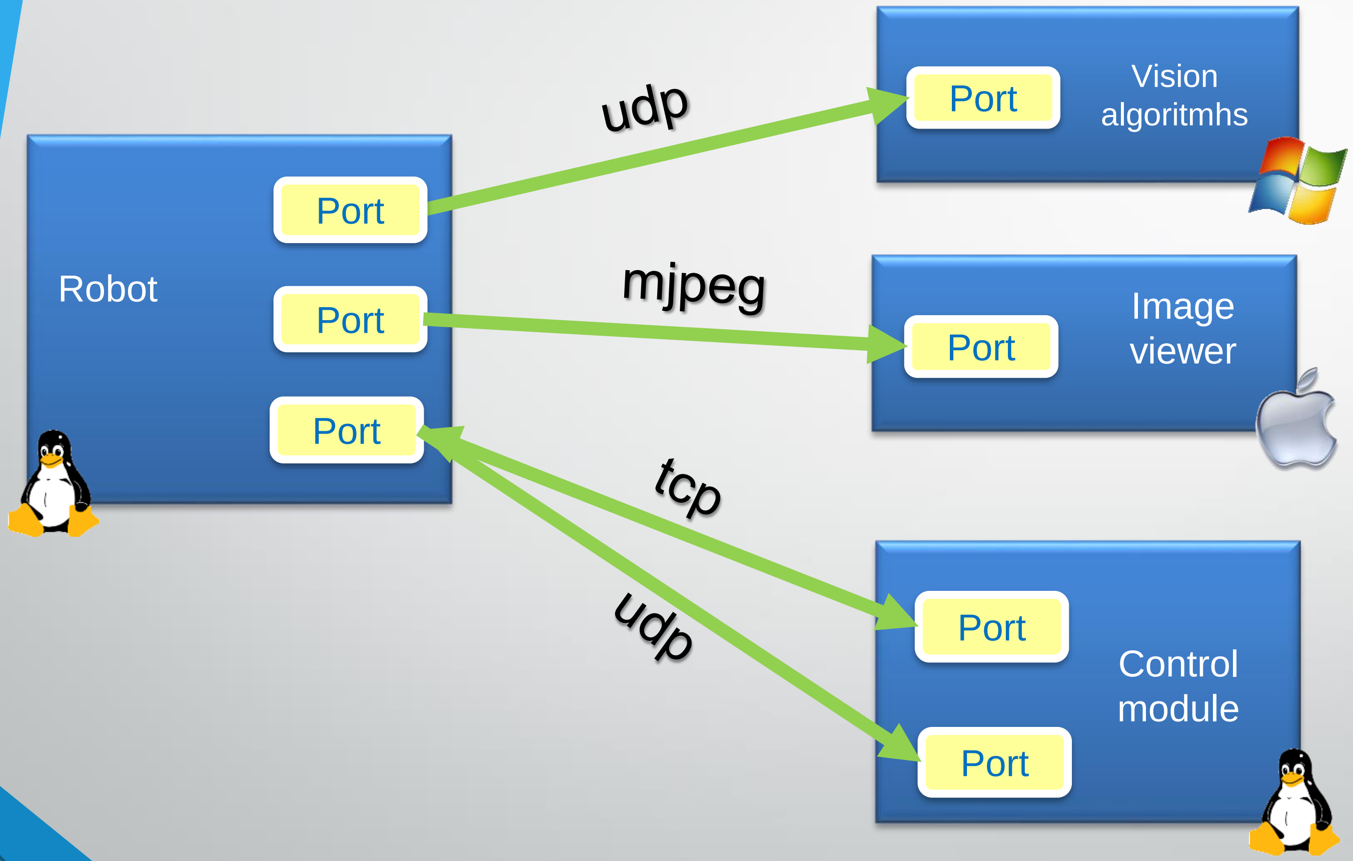


Who uses YARP



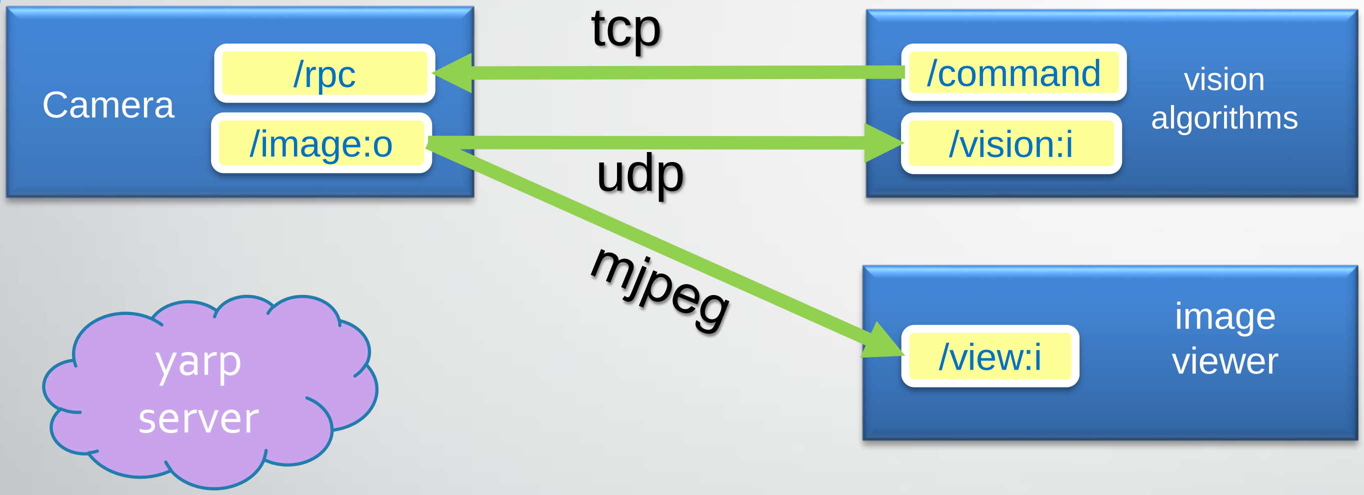
YARP Ports

YARP Ports: How YARP communicates



- YARP ports are the communication entry point.
- A port is a bi-directional communication entity.
- Many clients can connect to a port.
- Each connection can use different protocols or custom carrier to manipulate data on the fly.

YARP Ports: How YARP communicates



YARP server acts as a DNS, resolving yarp port names into system sockets

```
$ yarp name list

/image:o    192.168.1.1:10001
/vision:i   192.168.1.2:10002
/view:i     192.168.1.3:10003
/command    192.168.1.2:10004
/rpc        192.168.1.3:10005
```

yarp connect <source> <receiver> <carrier>(tcp)

```
$ yarp connect  /command   /rpc
$ yarp connect  /image:o   /vision:i  udp
$ yarp connect  /image:o   /view:i    mjpeg
```

Data types

Data in YARP are Portable classes with read and write capabilities. This kind of classes can travel through the YARP network.

```
class MyData : public yarp::os::Portable
{
    // Portable interface toward YARP
    read(…);
    write(…);

    // Custom user methods for data handling
    fill_me();
    getData();

    // Usually for readability
    toString();
}
```

yarp::os::Property

Dictionary type of data

```
Property prop;
prop.clear();
```

Works in pair <key, data>, where

- Key is a string
- Data is a `yarp::os::Value`

```
prop.put(“myInt”, 5);
prop.put(“myString”, “Hello World”);
prop.put(“myPi”, 3.14);
```

Entry can be grouped together, with a key

```
Property &myGroup = prop.addGroup(“group1”);
group1.put(“g1”, 2.5);
group1.put(“g2”, “We have cookies”);
```

Entry and group can be searched by the key

```
prop.check(“myInt”);
Value myInt = prop.find(“myInt”);
double myPi = prop.find(“myPi”).asFloat64();
Bottle &group = prop.findGroup(“myGroup”)
```

yarp::os::Bottle

Most flexible (but inefficient) type of data.

```
Bottle bot;
void clear();
```

Can hold variable number of Value.

```
bot.addInt32(5);
bot.addString("hello");
```

Bottle can be appended or nested one into another.

```
Bottle& b1 = addList();
b1.addFloat64(10.2);
```

A Property can be an element of a Bottle

```
Property &prop = bot.addDict();
prop.put(“pib”, “Help me”);
```

Bottle can be accessed using indexes.
Size is the number of element you can get()

```
Value &v0 = bot.get(0);
Value &v1 = bot.get(1);
```

yarp::sig::ImageOf<PixelType>

Container for image type

Template working with many different pixel types

Full documentation here:
http://www.yarp.it/classyarp_1_1sig_1_1ImageOf.html

```
ImageOf<PixelRgb> yarpImage;
yarpImage.resize(300,200);
PixelRgb  rgb;
rgb = yarpImage.pixel(10, 20);
```

yarp::sig::PointCloud<DataType>

Container for point cloud type.

Template working with many different point types.

Moreover, it has been implemented to be compatible with Point Cloud Library (PCL) and with an interoperability between different point types.

Full documentation here: http://www.yarp.it/yarp_pointcloud.html

```
PointCloud<DataXYZRGBA> yarpPointCloud;
yarpPointCloud.resize(300,200);
DataXYZRGBA point;
point = yarpPointCloud(10, 20);
```

Working with Ports – Client/Server

Ports are identified by their name.

Constraints:

- Names must be unique
- Names must start with '/' character
- No '@' character allowed

Ideal for client/server pattern

```
yarp::os::Port myPort;
myPort.open(“/port”);

Bottle b;
port.read(b);
int n = b.get(0).asInt32();
n++;
b.clear();
b.addInt32(n);
myPort.write(b);

myPort.close();
```

Working with Ports -- Streaming

In case of continuously broadcasted data (e.g. video streaming), a yarp::os::BufferedPort<T> can be used.

Main differences:

- Data type is fixed for port lifetime
- Memory creation/destruction is handled by the port
- Buffering policy can be set (default latest message is kept)
- A dedicated thread handles the read/write operations optimizing user thread cycle

```
BufferedPort<Bottle> port;

port.open("/out");

// Get memory to write into.
Bottle& b = port.prepare();

b.clear();

b.addString(“Hello world”);

port.write();

port.close();
```

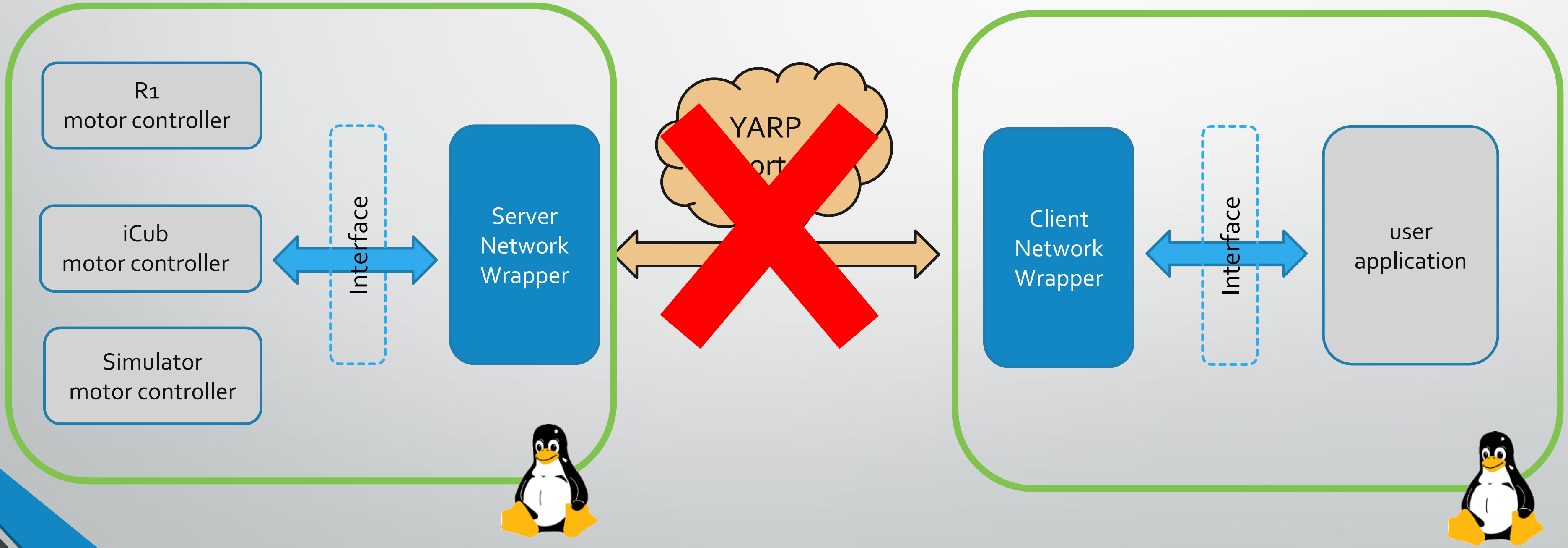
YARP Devices

YARP Devices: Hardware abstraction

- YARP devices are dynamically loaded C++ classes, that expose their functionalities
- They are used to model functionalities under common interfaces, such as sensors (cameras, IMUs, Force-Torques), low-level joint motor control, even if the under the hood the implementation is different
- When you launch a robot like iCub, you launch a program `yarprobotinterface` that creates and run several YARP devices to communicate with the low-level aspects of the robot.

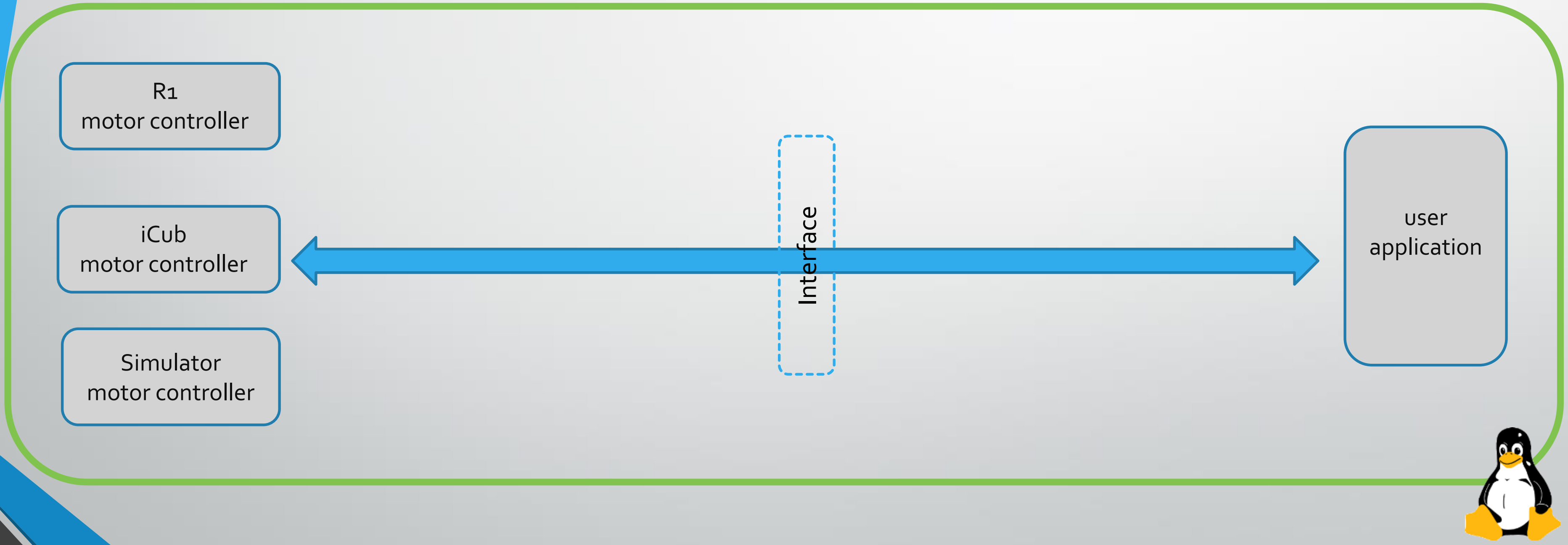
YARP Devices: Hardware abstraction

Client & Server on the same machine



YARP Devices: Hardware abstraction

Client & Server on the same machine



R1
motor controller

iCub
motor controller

Simulator
motor controller

Interface

user
application

Interfaces

A class with pure virtual methods.

Servers provide functionalities by implementing required methods.

Clients use the functionalities by calling provided methods.

```
IPositionControl::getAxes() = 0;
IPositionControl::positionMove(…) = 0;
IPositionControl::relativeMove(…) = 0;
IPositionControl::checkMotionDone(…) = 0;
IPositionControl::setRefSpeed(…) = 0;
IPositionControl::setRefAcceleration(…) = 0;
IPositionControl::getRefSpeed(…) = 0;
IPositionControl::getRefAcceleration(…) = 0;
IPositionControl::getTargetPosition(…) = 0;
IPositionControl::stop(…) = 0;
```

Opening a device

Devices are opened by mean of a special class called “PolyDriver”.

PolyDriver is a polymorphic class which can turn into any device.

Keyword “device” tell YARP which device we really want to open.

All other parameters will be propagated to the specified device.

```
PolyDriver mystica;

Property config;

config.put(“device”, “device_type”);
config.put(“deviceParam1”, paramValue1);
config.put(“deviceParam2”, paramValue2);
...

mystica.open(config);
```

Remote Control Board

Device devoted to provide remote access to the robot motor control is the “remote_controlboard”

Required parameter to configure it are:

- Remote port prefix: remote

- Local port name: local

```
PolyDriver poly;

Property config;

config.put(“device”, “remote_controlboard”);
config.put(“remote”, “/icub/head”);
config.put(“local”, “/<myApplication>”);
...

poly.open(config);
```

CONTINUE

Remote Control Board

Once opened, we need to specify which interface we want to work with.

To get a specific view of the device:

- create a pointer to the interface we want to use
- fill it by calling the `.view(…)` function

In case the device does not implement that interface, the pointer will be `nullptr`!

A device can implement more than one interface.

```
IPositionControl *posControl = nullptr;

poly.view(posControl);

if(!posControl)   // handle error
   ...

posControl->getAxes(…);
posControl->positionMove(…);

IVelocityControl *velControl = nullptr;
poly.view(velControl);

velControl->velocityMove(…);
```

IPositionControl

Give access to main position control commands.

Used to send high level targets, with a velocity & acceleration profile.

For getters, memory must be allocated by user.

Units in YARP are SI compliant, except angles for controlboard, which are in degrees, degrees/s

IPositionControl

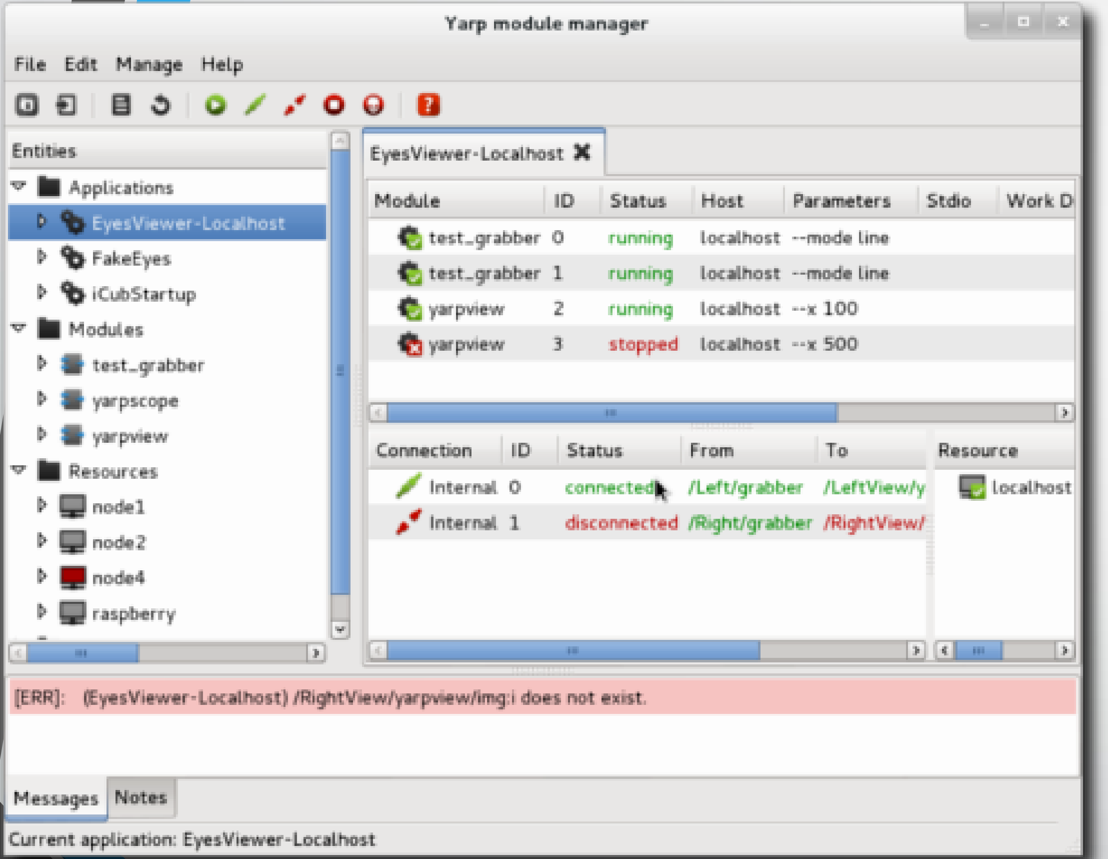
```
int joints;
posControl->getAxes(&joints);        // Get number of joints

posControl->setRefSpeed(0, 5);       // set a speed of 5 degrees/s for joint 0
posControl->positionMove(0, 30);     // move the joint 0 to +30 degrees

bool done = false;
do
{
    checkMotionDone(&done);          // this function checks the movement completion
}
while(!done);

posControl->positionMove(0, 0);      // reset joint position to 0
```

YARP Command Line and GUI tools

YARP Command Line tools

- `yarpserver`: Launch the name server used to register YARP port names
- `yarp`: command-line utility "yarp" performs a set of useful operations for a YARP network.
  - `yarp name list`: list all known YARP ports.
  - `yarp connect <src> <dst>:` Connect the two specified YARP ports.
  - `yarp detect`: Searches for an activate yarpserver in the network.
  - See https://www.yarp.it/latest/yarp.html for all the available functionalities of `yarp` command
- `yarprobotinterface`: Launch a group of devices as a single process, tipically used when you launch a robot
- `yarpdatadumper`: Dump the data connected to a port on a file.
- See https://www.yarp.it/latest/#yarp_command_line_tools for a the complete list of tools

YARP GUI: YARP manager



Module	ID	Status	Host	Parameters	Stdio	Work D
test_grabber	0	running	localhost	--mode line		
test_grabber	1	running	localhost	--mode line		
yarpview	2	running	localhost	--x 100		
yarpview	3	stopped	localhost	--x 500		

Connection	ID	Status	From	To
Internal	0	connected	/Left/grabber	/LeftView/y
Internal	1	disconnected	/Right/grabber	/RightView/

Resource
localhost

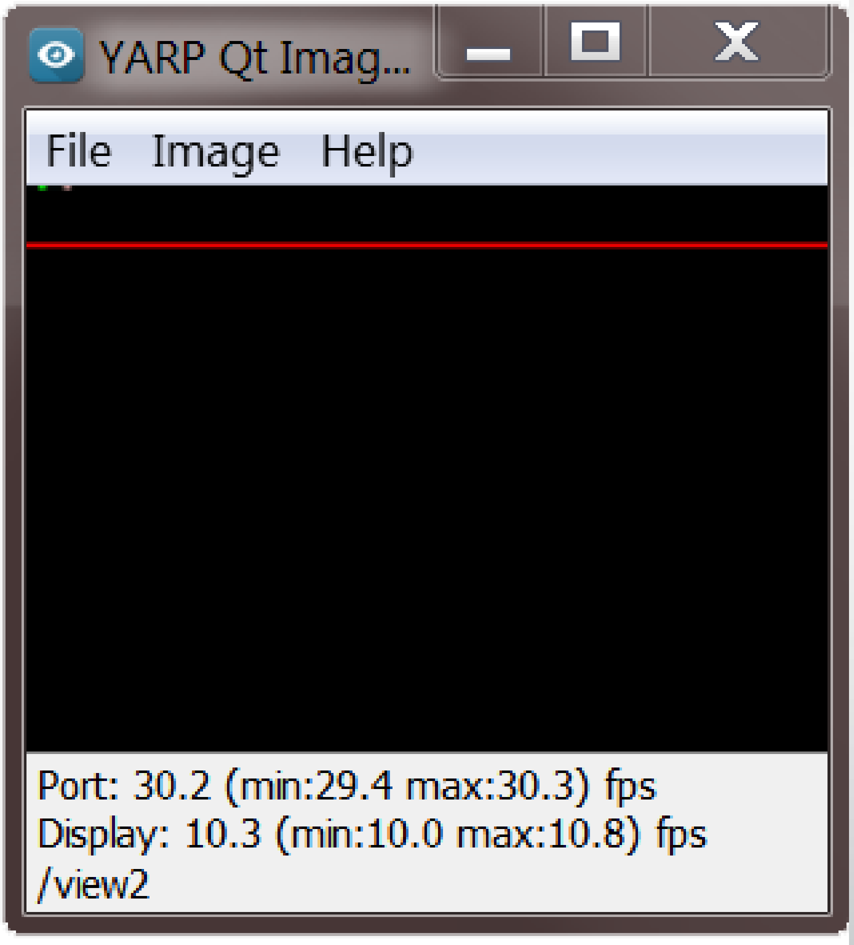
```
[ERR]:   (EyesViewer-Localhost) /RightView/yarpview/img:i does not exist.
```

Current application: EyesViewer-Localhost

`yarpmanager` is a tool for running and managing multiple programs on a set of machines.

- The programs/executables that can be launched are called "modules" and are grouped in "applications", that are specified by XML files.
- Specific demonstration on the iCub are launched via appropriate yarpmanager applications
- The programs launched by yarpmanager do not need to use YARP to be used via yarpamanager, you can launch YARP independent programs, Bash scripts or Python commands.
- https://www.yarp.it/latest/yarpmanager.html

YARP GUI: YARP view



```
Port: 30.2 (min:29.4 max:30.3) fps
Display: 10.3 (min:10.0 max:10.8) fps
/view2
```

`yarpview` is a graphical interface for viewing images transmitted on the YARP network.

- A typical use of yarpview is to spawn two of them via yarpmanager to visualize the two eyes cameras of iCub.
- http://www.yarp.it/latest/yarpview.html

Other YARP features

- ResourceFinder
  - Infrastructure that specifies where configuration and data files are installed and searched, to permit to easily have different configuration files for different experiments or robots.
  - http://www.yarp.it/git-master/yarp_resource_finder_tutorials.html
  - https://github.com/vvv-school/tutorial_RFModule-simple
- Carriers:
  - Communicate across ports via mjpeg, h264, unix socket, portmonitor, shared memory, ROS
- Bindings:
  - Support via SWIG for Python, Lua, Ruby, C#, MATLAB/Octave.
  - http://www.yarp.it/latest/yarp_swig.html

Other middleware

Cool!

“But what about ROS?”

YARP

Ports can be typed or not

Multi-platform (also mobile)

Run-time reconfiguration of connections

Different carriers, user custom

QoS, channel prioritization

Smaller community

Rich set of libraries and tools

Binary packages for all supported distributions

ROS

Both topic and service are strongly typed

Mainly Ubuntu (ROS2 Linux, macOS and Windows)

Connections from a topic use the same protocol

No concept of carrier (DDS on ROS2)

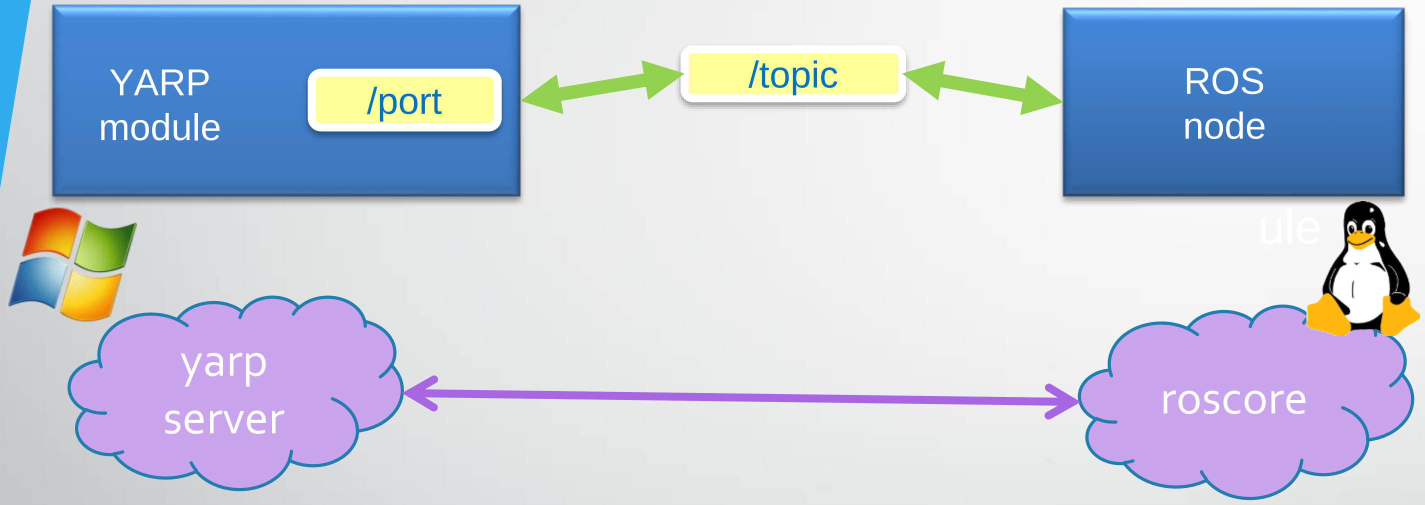
QoS on ROS2

Huge and very active community

Much more rich set of libraries and tools

Distribution-like facilities

YARP - ROS compatibility



/topic@/yarpNode

YARP ask roscore to establish a new connection

YARP loads a specific carrier to convert data into ROS-like type on the fly

No need to have ROS installed

https://www.yarp.it/latest/yarp_with_ros.html

THANKS FOR THE ATTENTION!