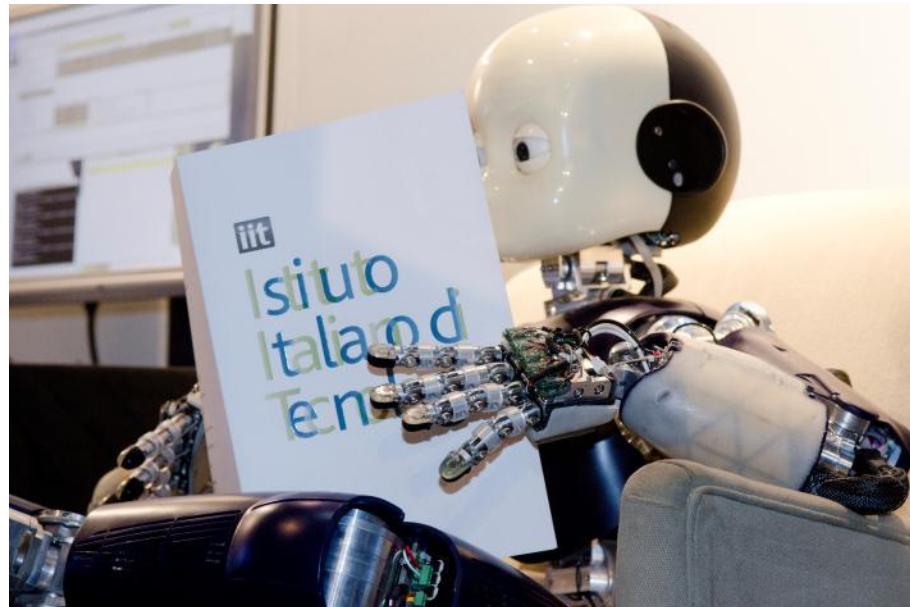


# Robot programming

Lorenzo Natale, Ali Paikan, Alessandro Roncone, Ugo Pattacini, Daniele Domenichelli

iCub Facility  
Istituto Italiano di Tecnologia, Genova



*Genova, July 2015*

# Program

- 2<sup>nd</sup> July, 2015 – h 10:12 – Sala Natta - Lesson 1: *Introduction to the middleware*
- 3<sup>rd</sup> July, 2015 – h 10:12 – Sala Volta - Lesson 2: *Hello world, command line tools. YARP threads and YARP modules*
- 6<sup>nd</sup> July, 2015 – h 10:12 – Sala Natta - Lesson 3: *YARP Manager, port monitor, and realtime computing with YARP*
- 7<sup>nd</sup> July, 2015 – h 14:16 – Sala Dulbecco - Lesson 4: *Image processing and Computer Vision*
- 8<sup>nd</sup> July, 2015 – h 14:16 – Sala Dulbecco - Lesson 5: *The Gaze controller and the Cartesian Controller for efficient inverse kinematics*
- 10<sup>nd</sup> July, 2015 – h 14:16 – Sala Dulbecco - Lesson 6: *Software development with GIT, the GIT workflow*

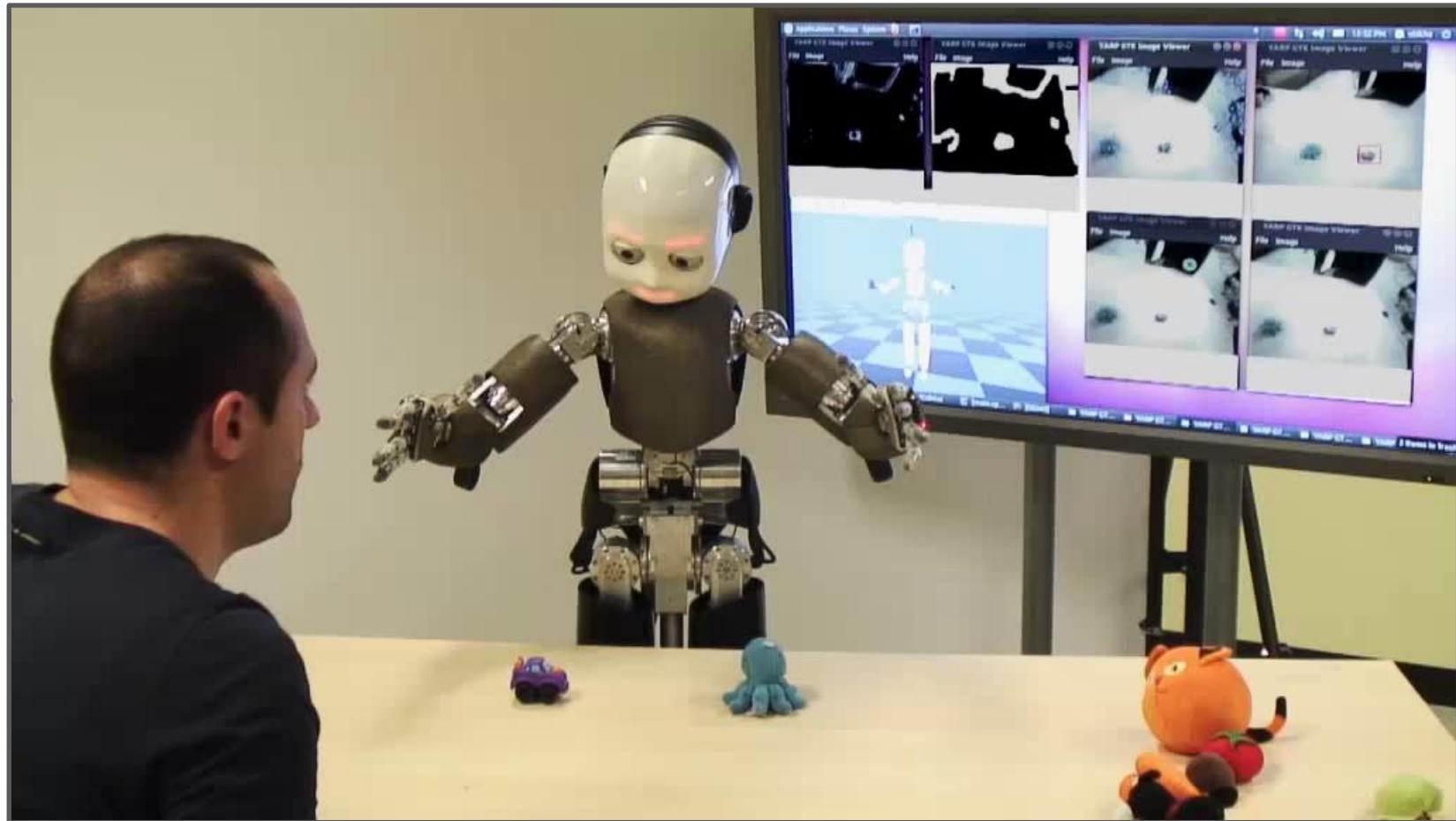
# References

- YARP: [www.yarp.it](http://www.yarp.it)
  - Documentation
  - Papers (see also references in this presentation)
- iCub: <http://wiki.icub.org/wiki/Manual>
- CMake:
  - [www.cmake.org](http://www.cmake.org) → documentation & wiki
  - Mastering CMake, by Kitware Inc.
- C++:
  - Thinking in C++, Bruce Eckel
  - The C++ programming language, Bjarne Stroustrup
  - Anything else

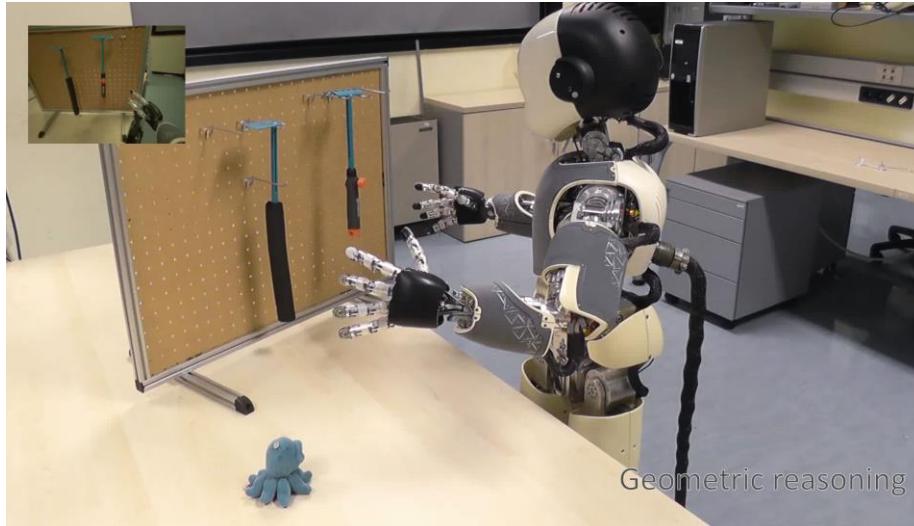
# Motivations



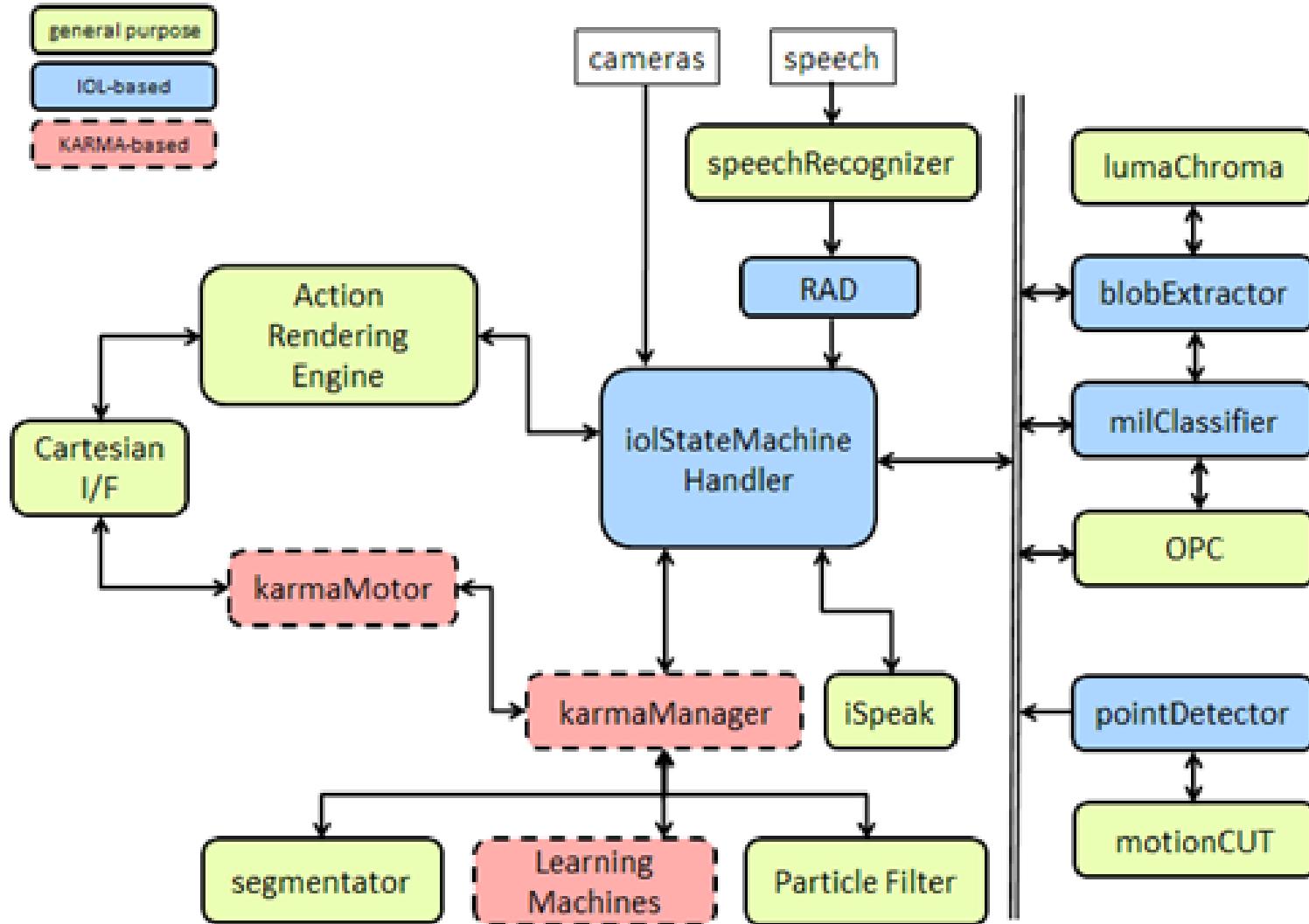
# Complex behaviors



# Integration of software components

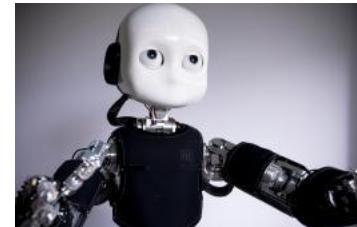
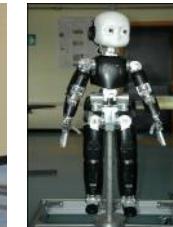
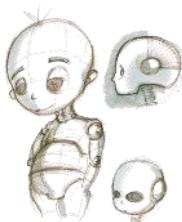


# Complex behaviors



# Key issues

- Asynchronous development
- Variability: various scenarios and platforms
- Lack of standards
- Inherent complexity, distributed processing, lots of sensors, real-time
- Fluctuation in hardware and algorithms, lots of open questions



# Software architecture

- Major cost in software development is debugging, recycling code is key
- Divide and conquer
- Modularity
- Factor out platform specificities
  - Hardware Abstraction Layer
  - Communication Abstraction
  - Operating system
  - Parameters
  - Computing infrastructure

# Separation of concerns (5C)

Goal: separate software components

- Computation ← What we are interested in
  - Communication ← Dependent on the hardware, network topology
  - Configuration
  - Coordination
  - Composition
- { Application dependent



# Component driven software development

```
output myAlgorithm(input)
{
    ...
    out = call alg1(in)
    ...
    out = call alg2(in)
    getImage() // from usb camera
    ...
}
```

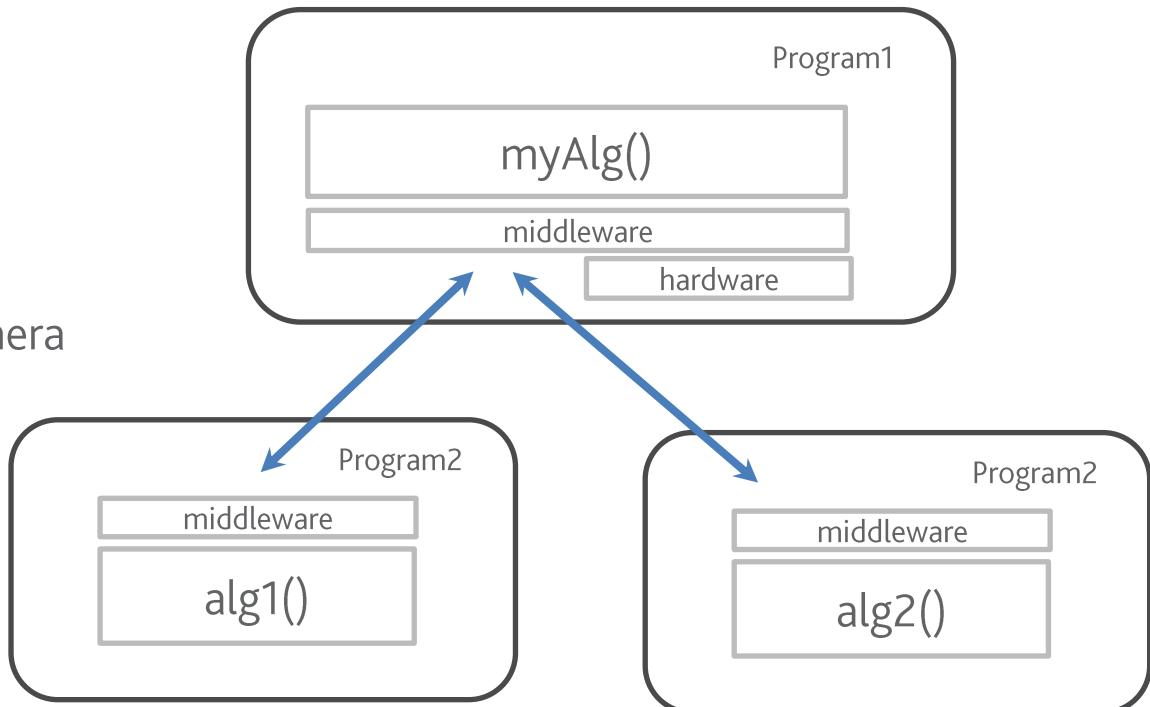
```
output alg1(input)
{code}
```

```
output alg2(input)
{code}
```

# Component driven software development

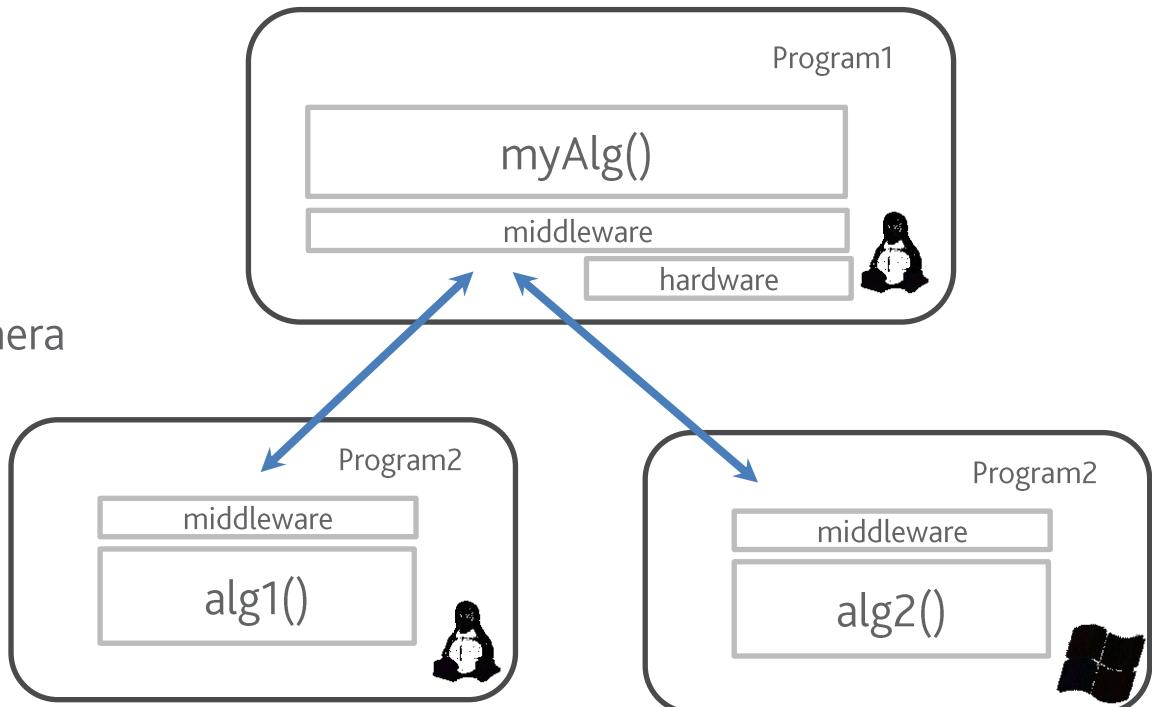
```
output myAlgorithm(input)
{
...
out = call alg1(in)
...
out = call alg2(in)
getImage() // from usb camera
...
}
output alg1(input)
{code}

output alg2(input)
{code}
```



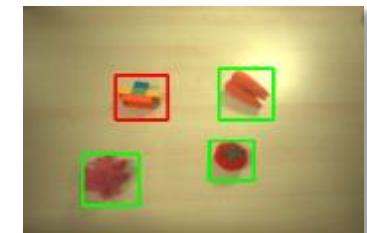
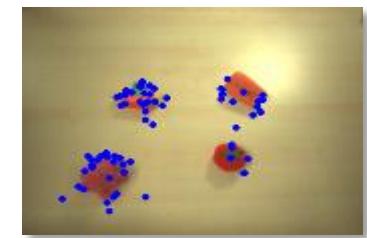
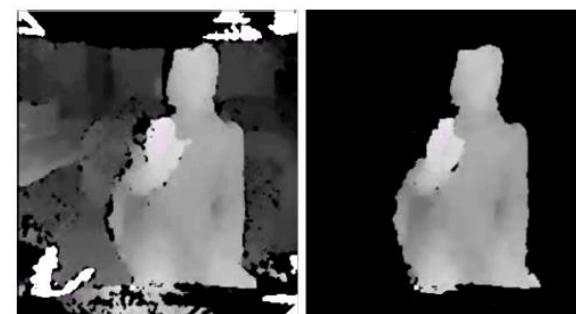
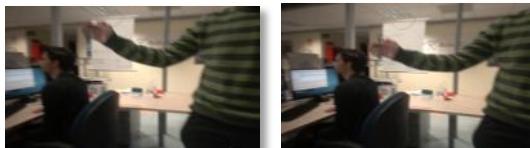
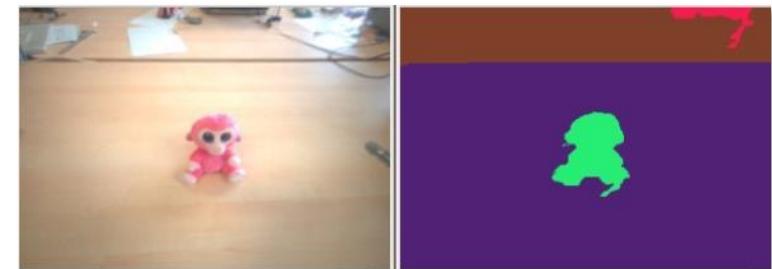
# Component driven software development

```
output myAlgorithm(input)
{
...
out = call alg1(in)
...
out = call alg2(in)
getImage() // from usb camera
...
}
output alg1(input)
{code}
output alg2(input)
{code}
```



# Components: some examples from the iCub repository

- Algorithms for motion computation and egomotion compensation
- Machine learning for vision
- Disparity map
- Action recognition
- Segmentation
- See <https://github.com/robotology>



# Component driven development

- Modular software: simple structure, data encapsulation, interface
- Reconfigurable components
- Reduced coupling: interaction between components happens through pre-defined standards (no direct inclusion of header files)

# Communication paradigm

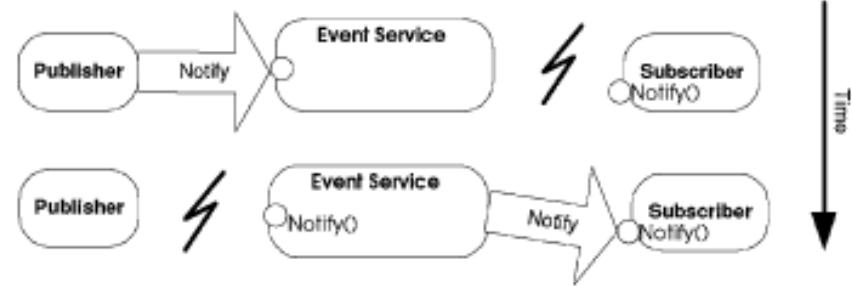
- Publish/Subscribe
  - Space, time, synchronization decoupling
- Remote Procedure Calls (RPC)
  - Remote invocation of object
  - Synchronous nature (although variant exists)

# Publish/Subscribe

- **Space decoupling**: the interaction parties do not need to know each other; publisher publish events through an event service and the subscribers get these events indirectly through the event service
- **Time decoupling**: interaction parties do not need to be actively participating in the interaction at the same time, publisher might publish events while subscribers is disconnected and subscribers might get notified of an event while the original publisher is disconnected
- **Synchronization decoupling**: publishers are not blocked while producing events and subscribers can get asynchronously notified (through callback) of the occurrence of the event while performing concurrent activity



Space decoupling



Time decoupling



Synchronization decoupling

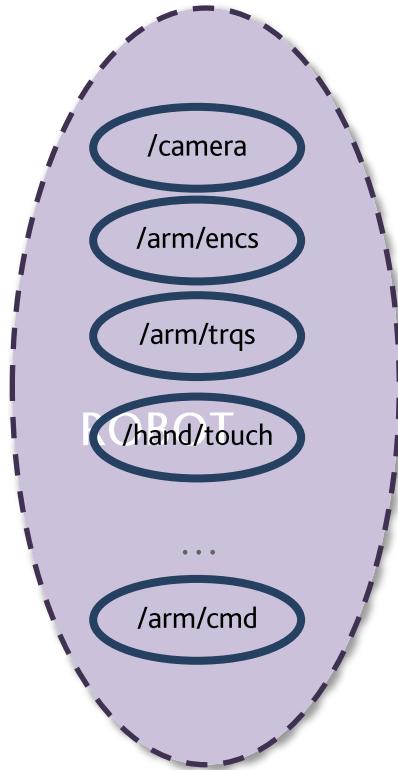
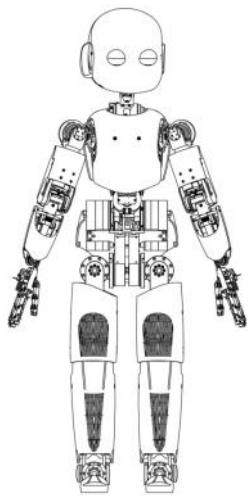
# YARP approach

- Limited form of publish-subscribe
- *Observer pattern*: subscribers register their interest directly with publishers, which manage subscriptions and sends events
- Publishers notify subscribers synchronously or asynchronously
- Space decoupling (from user perspective)
- Time coupling
- Dynamic (re)configuration

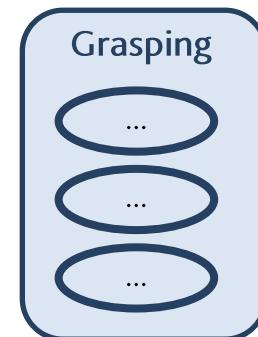
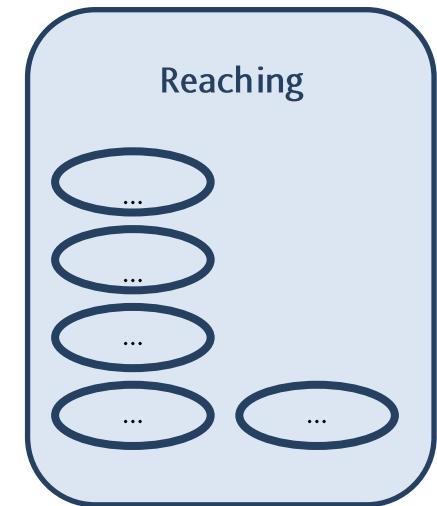
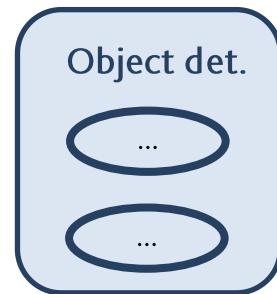
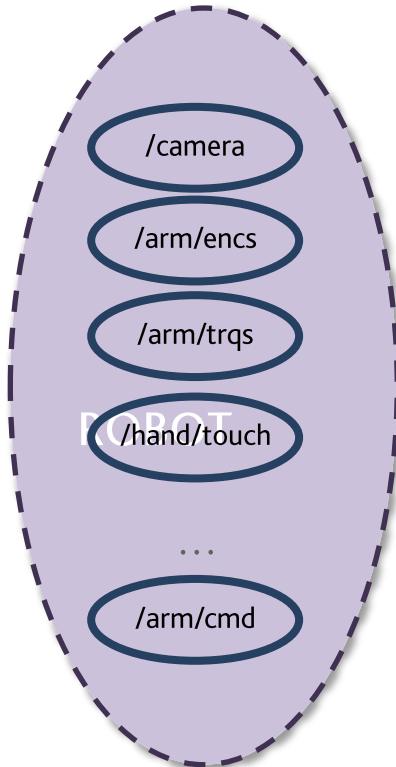
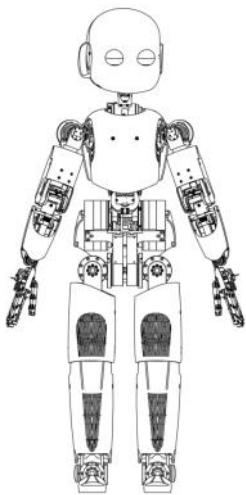
See also:

*YARP: Yet Another Robot Platform, G. Metta, P. Fitzpatrick, L. Natale, 2006*  
*Stewart et al., Design of Dynamically Reconfigurable Real-time Software Using Port-Based Objects, 1997*

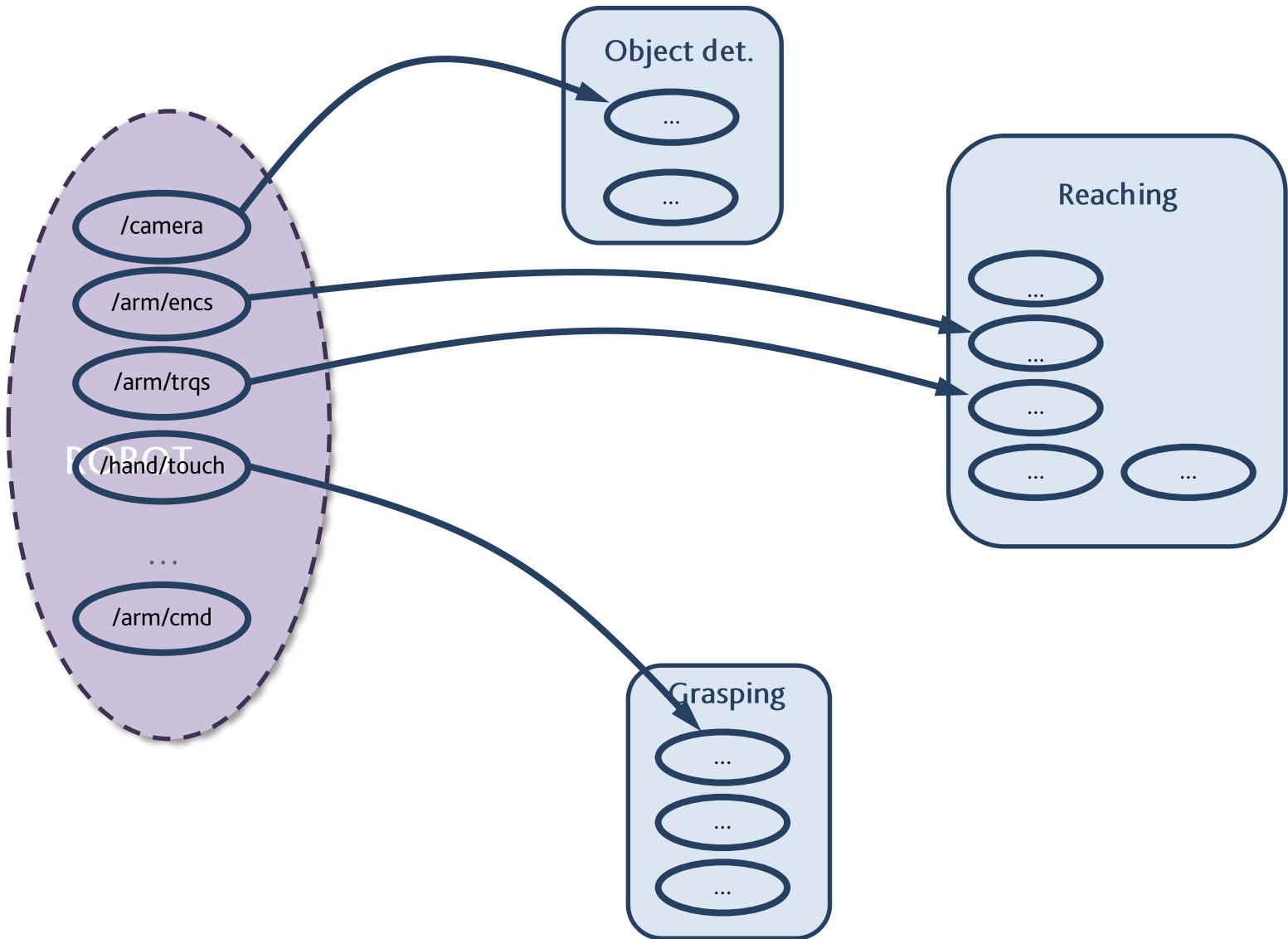
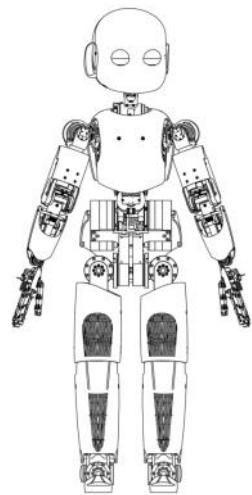
# iCub software architecture



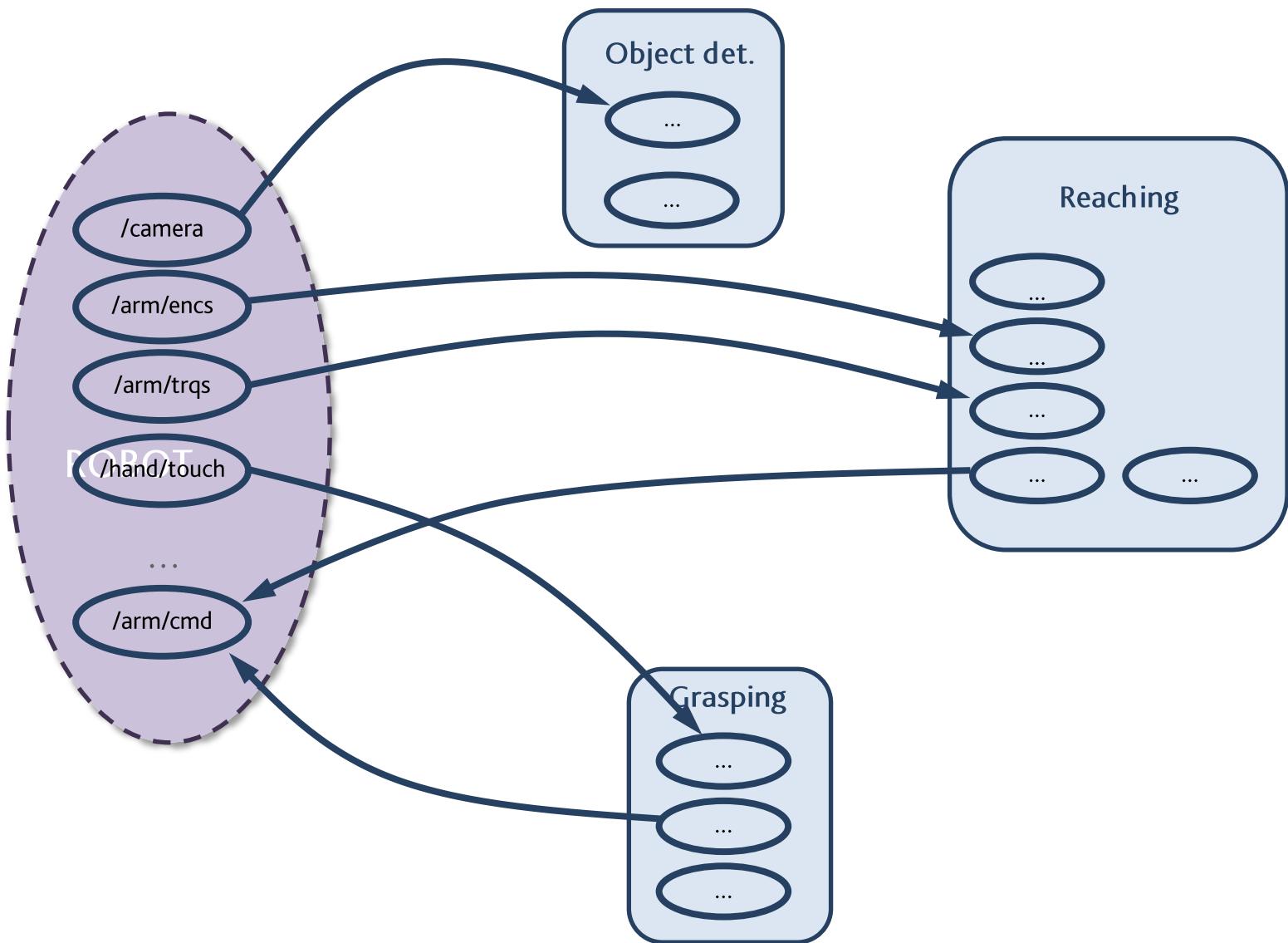
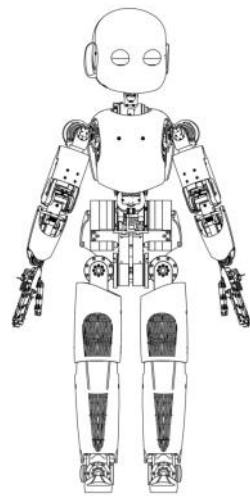
# iCub software architecture



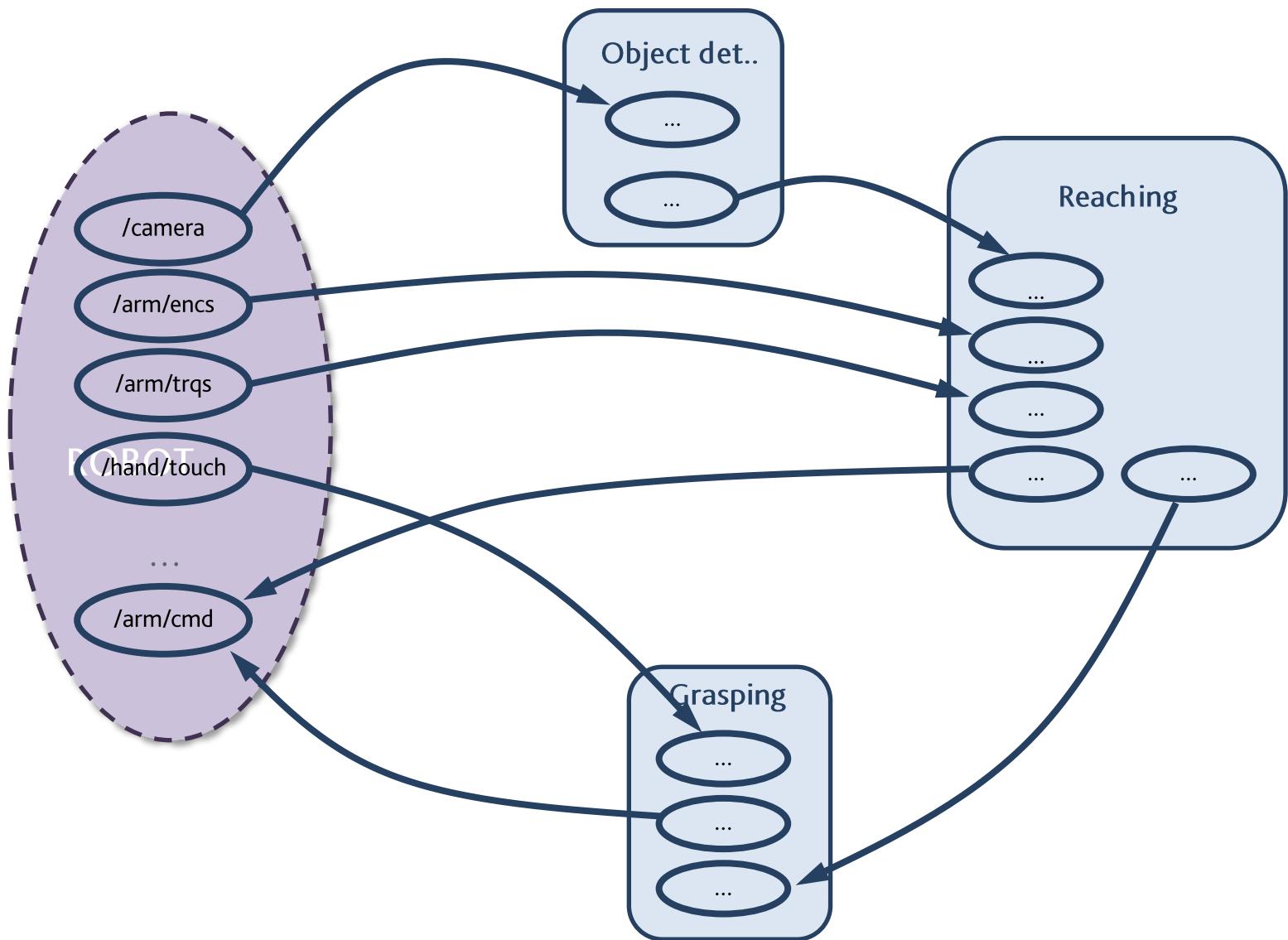
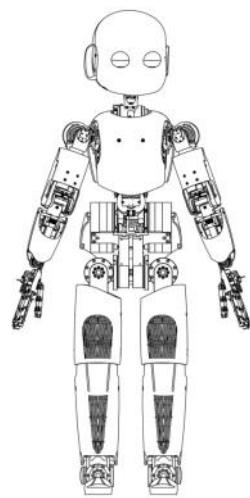
# iCub software architecture



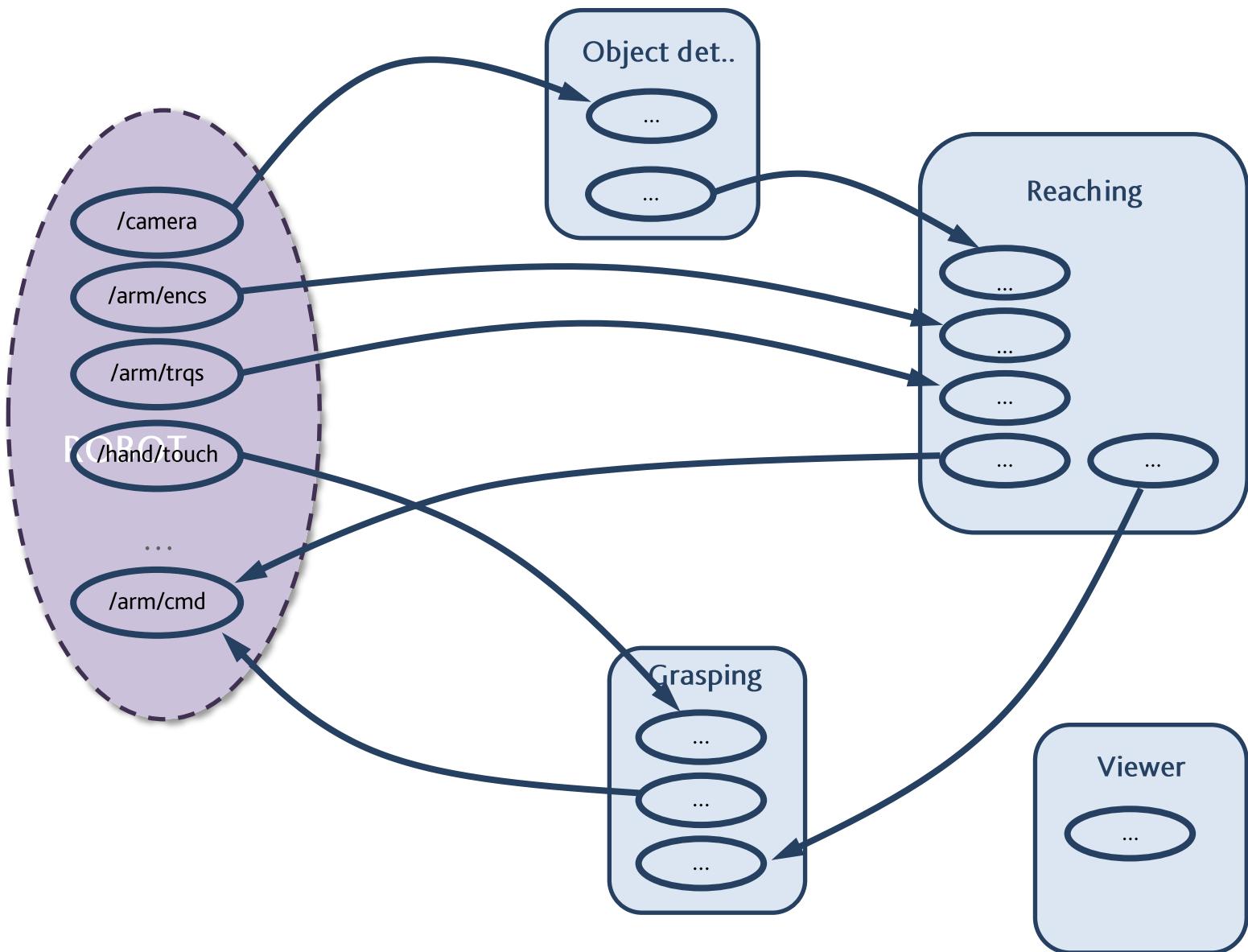
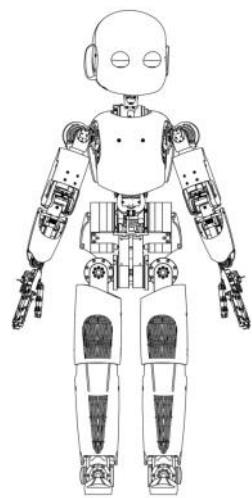
# iCub software architecture



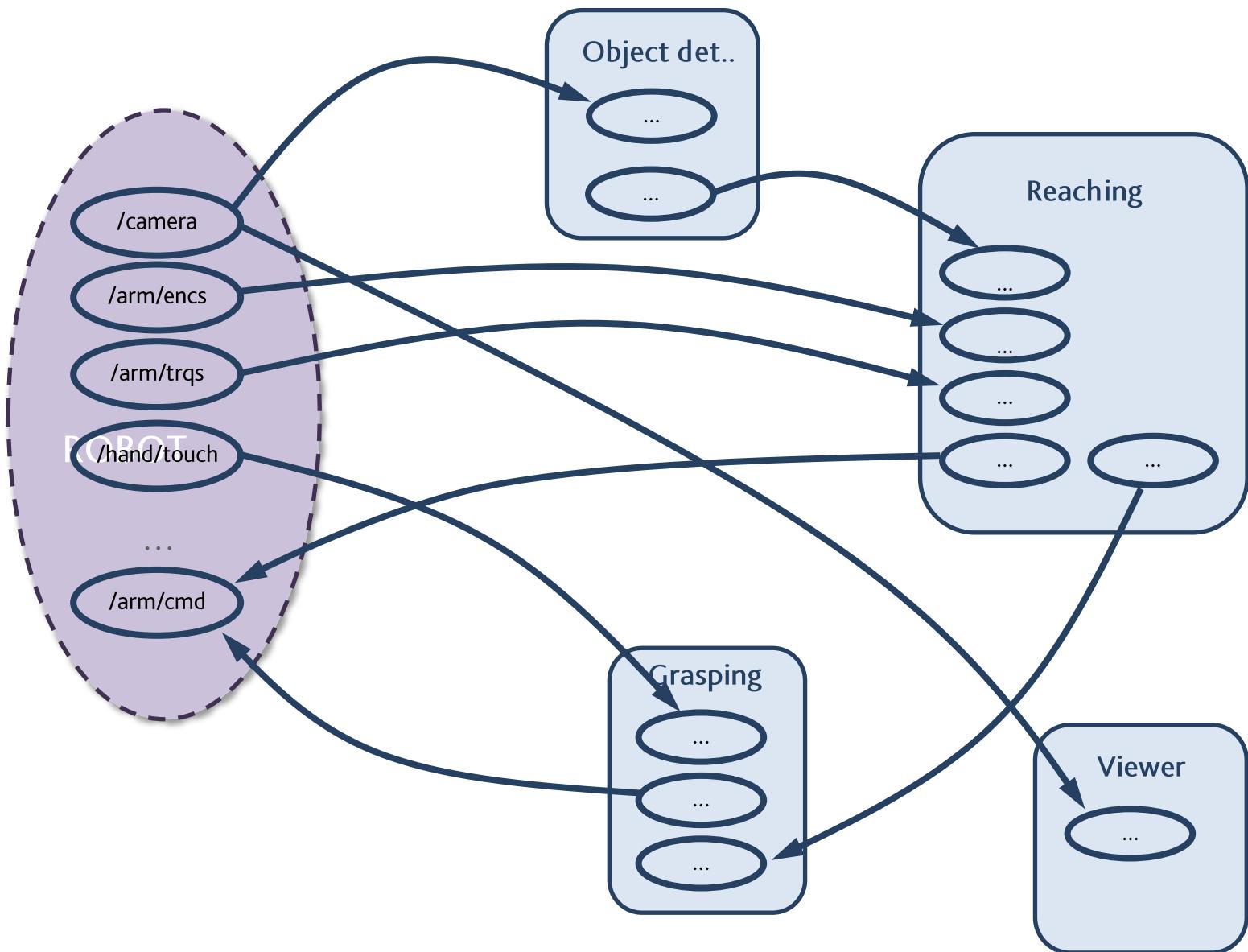
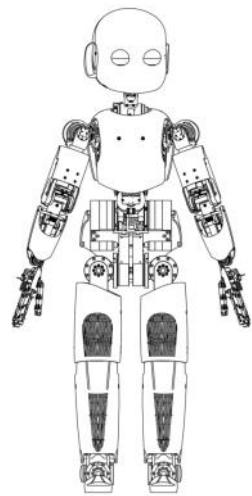
# iCub software architecture



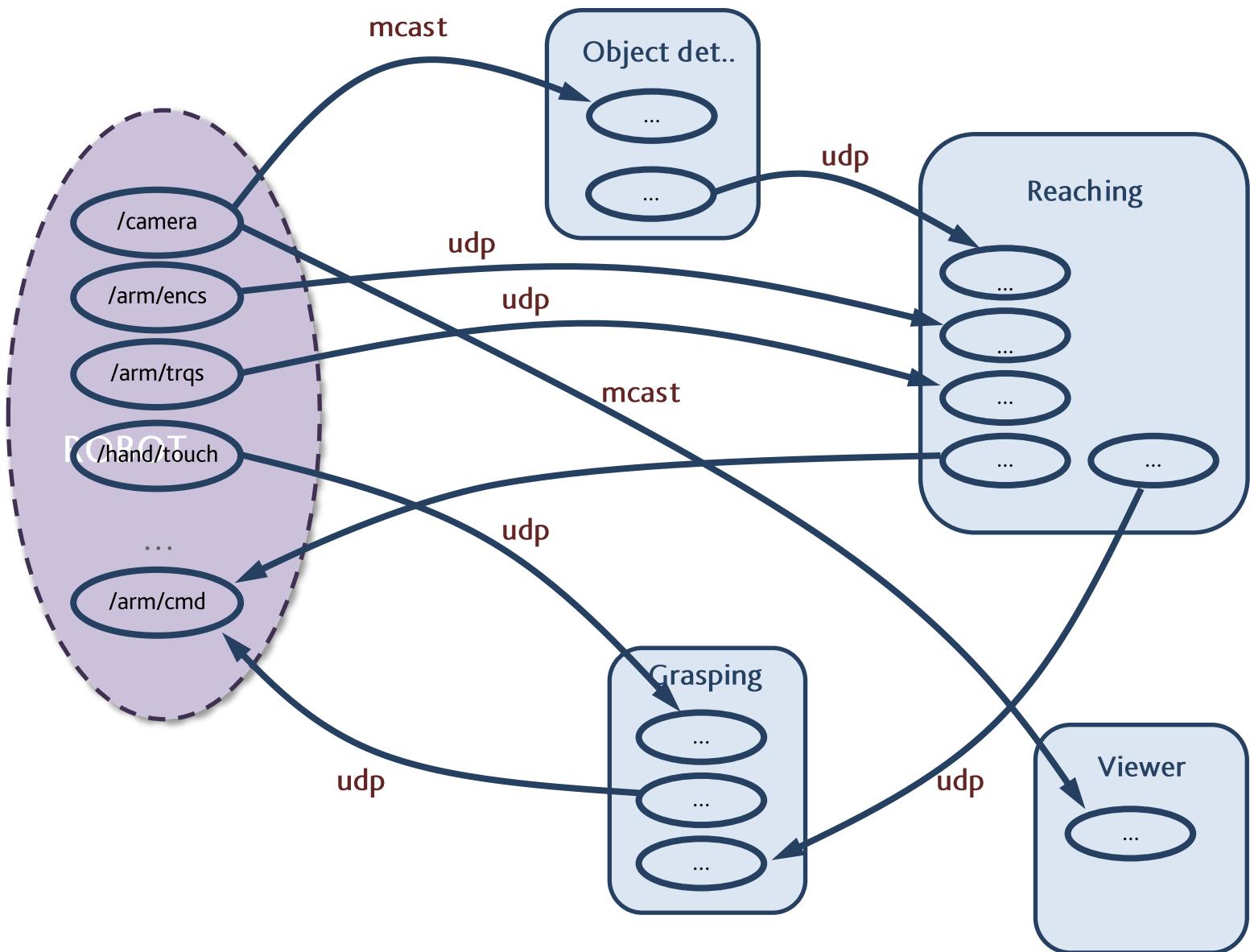
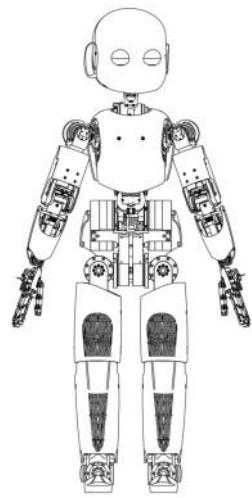
# iCub software architecture



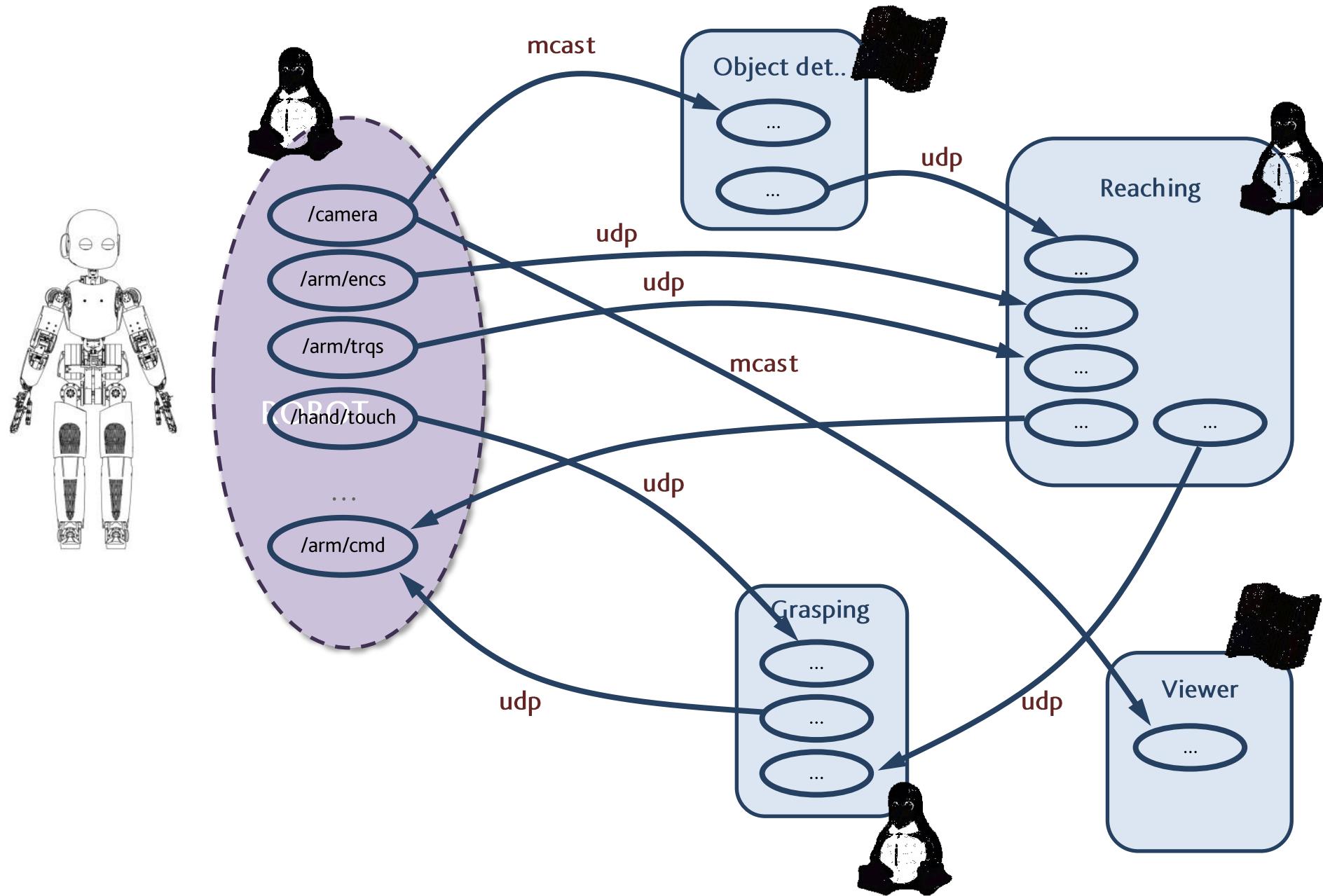
# iCub software architecture



# iCub software architecture



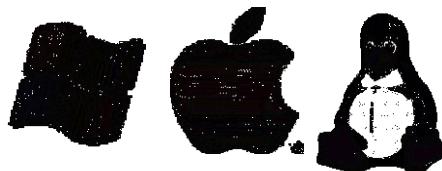
# iCub software architecture



# YARP: main features



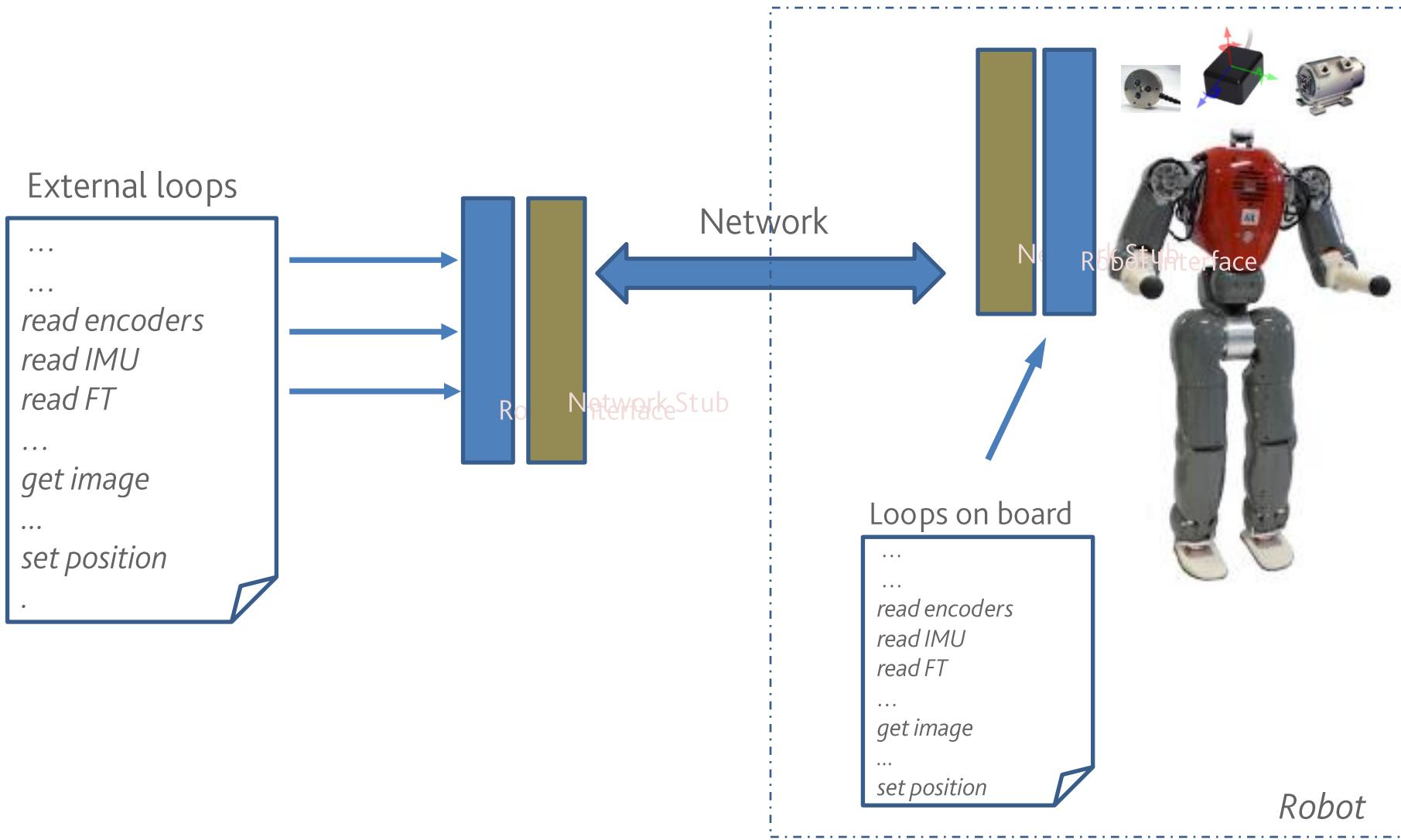
- Peer-to-peer, **loosely coupled**, communication
- Very stable code base >10 years old
- **Flexibility** and minimal **dependencies**, fits well with other systems
- Easy install with **binaries** on many OSes/distributions (Ubuntu, Debian, Windows, MacOs)
- Recently added: **channel prioritization** with QoS and thread priorities
- Many **protocols**:
  - Built-in: tcp/udp/mcast
  - Plug-ins: ROS tcp, xml rpc, jpg etc..



# Interfaces

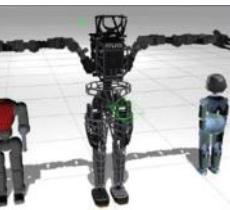
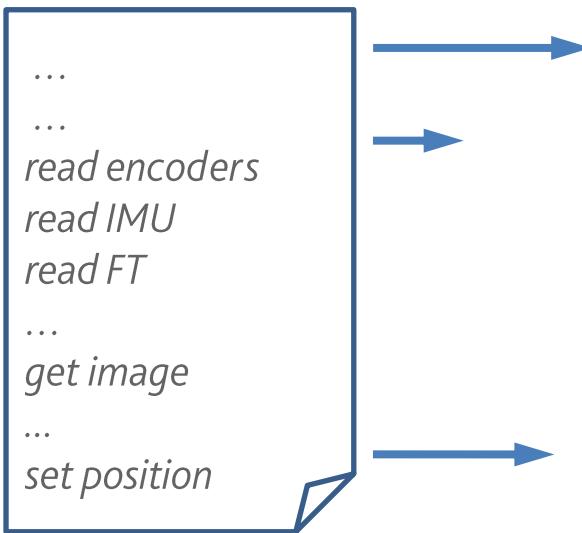
- Define interfaces to **motors** and **sensors** so to minimize the impact of changes in the hardware
- Also: network stubs allow **remotization**

# Interfaces

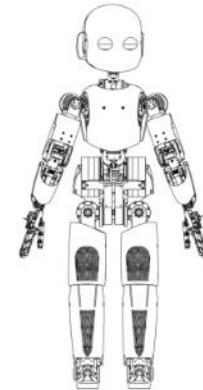


# Interfaces

## Control loops



Gazebo



iCub



COMAN

Recycle code across different robots and simulators (testing/fast prototyping)

Robot interface



Walkman

# YARP plugins

- YARP includes a plugin system for drivers and protocols (carriers)
- Interchangeable carriers allow:
  - interfacing existing software with ports (without bridges)
  - change significantly port behavior
- Examples:
  - ROS, jpeg, xml rpc, etc...
  - bayer carrier, priority based communication

# Examples



```
yarp connect /camera /receiver rec.bayer
```

# Examples



```
yarp connect /camera /receiver rec.bayer
```



```
yarp connect /65.52.88.202:5159 /receiver mjpeg
```

# Examples



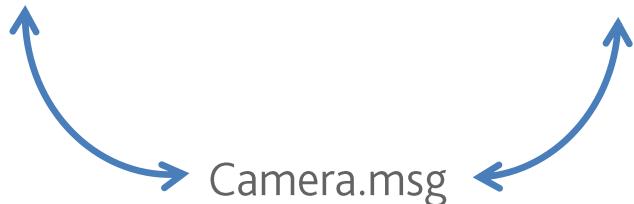
```
yarp connect /camera /receiver rec.bayer
```



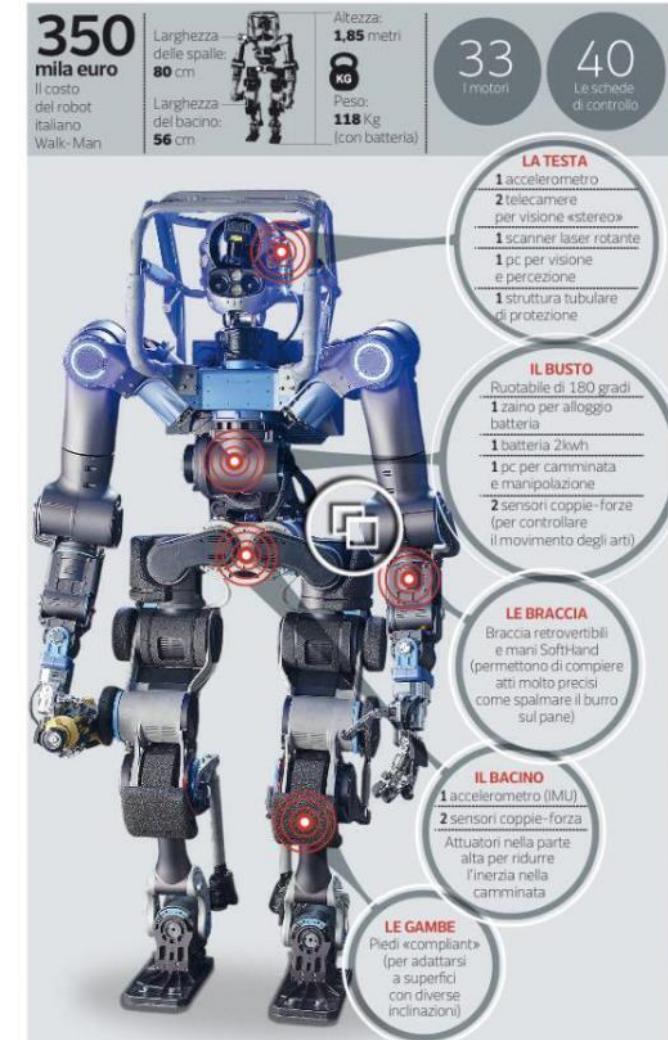
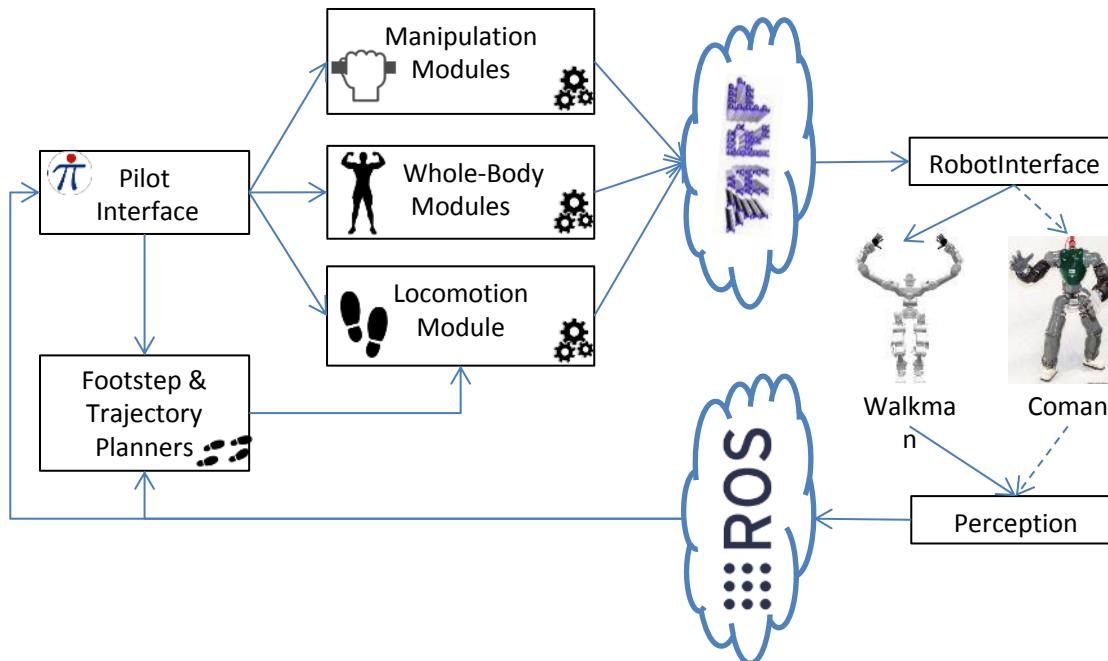
```
yarp connect /65.52.88.202:5159 /receiver mjpeg
```



```
yarp connect /image@/camera /receiver
```

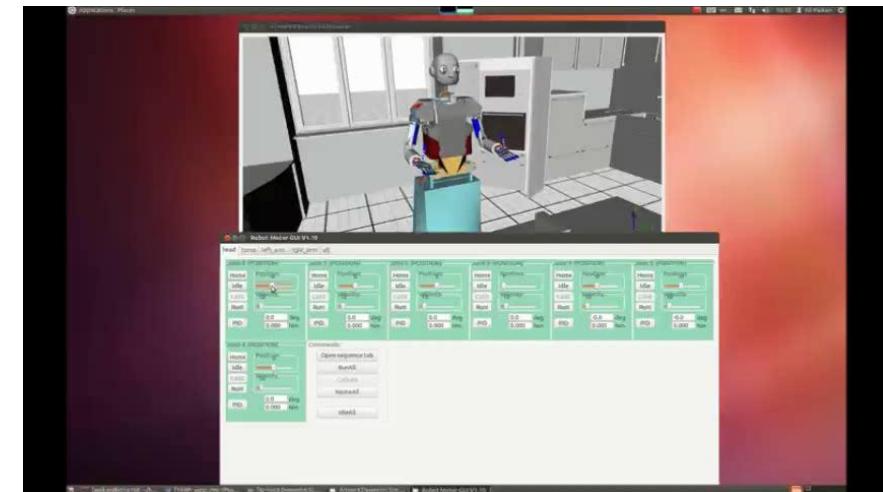
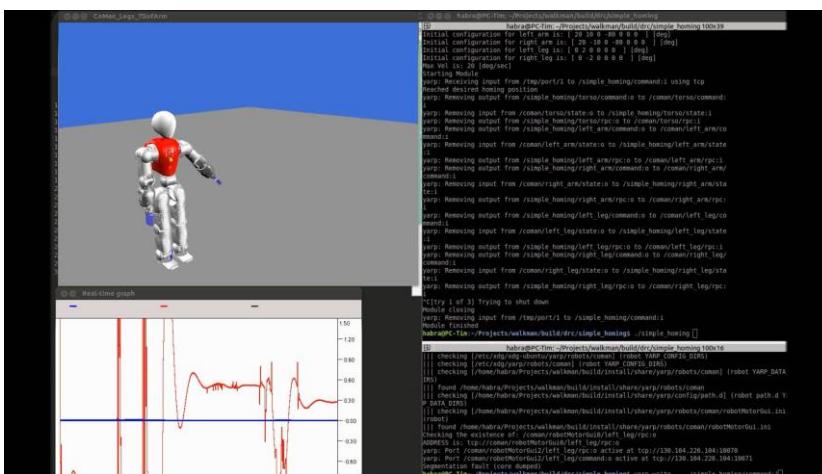


# Success story



# Simulators & robots

- iCub, COMAN, Walk-man, ARMAR III
- iCub\_SIM (in the iCub main repository)
- Gazebo (<https://github.com/robotology/gazebo-yarp-plugins>)
- Robotran



*Robotran-Yarp interface: a framework for real-time controller development based on multibody dynamics simulation, T. Habra, et al., 2015*

*Paikan et al., Transferring Object Grasping Skills and Knowledge Across Different Robotic Platforms, ICAR 2015*

# Managing complexity

In a modular system integration becomes an issue:

- Execution and monitoring
- Development
- Coordination



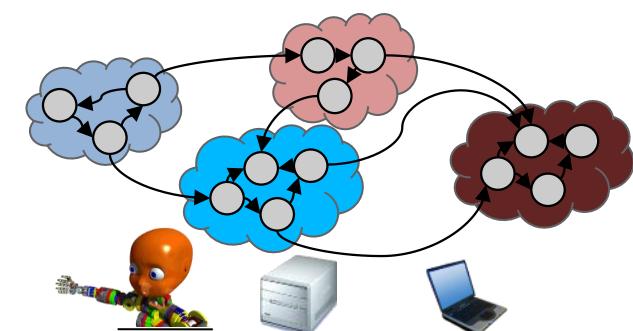
# Execution and monitoring: YARP manager

Required  
modules  
connections  
nodes  
resources

Available  
resources

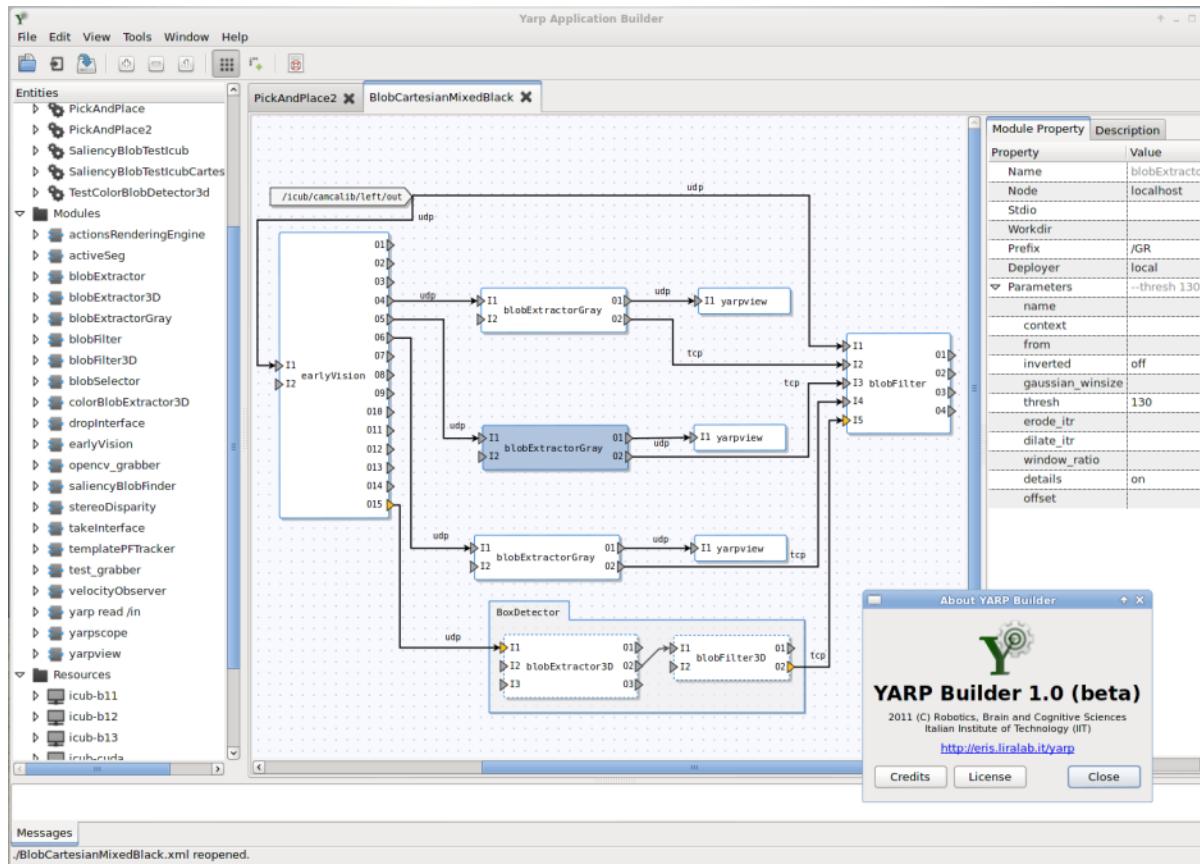
The screenshot shows the Yarp module manager window with the following details:

- Entities:** A tree view listing various applications and modules, including Crawling\_(EPFL)(1), Crawling\_(EPFL)(2), dataSetCollector\_dumper, dataSetPlayerColVision, dataSetPlayer\_example, dataSetPlayer\_recording\_example, DemoYI\_DARWIN, Demo\_Force\_Control, Demo\_Grasping\_with\_Impedance\_C, Demo\_Two\_Balls, dump\_Attention\_Application, Early\_Attention\_Mechanism, eMorph\_Application, eMorph\_Application\_Bottle, eMorph\_AttentionDemo\_Main, eMorph\_AttentionMechanism, eMorph\_AttentionMechanism\_GUI, eMorph\_AttentionMechanism\_Left, eMorph\_AttentionMechanism\_Main, eMorph\_Attention\_Test, eMorph\_Static\_Replay, eventLUOptFlwApp, eventOptFlwApp, Face, and Face\_Tracker.
- Calibrated\_Cameras X:** A tabbed panel showing the status of various modules. Modules listed include iCubmoddev (ID 0, running, host pc104), iCubmoddev (ID 1, running, host pc104), camCalib (ID 2, running, host icub-b14), camCalib (ID 3, running, host icub-b15), yarpview (ID 4, stopped, host icub14), yarpview (ID 5, stopped, host icub14), frameGrabberGui2 (ID 6, running, host icub14), and frameGrabberGui2 (ID 7, running, host icub14).
- Connections:** A table showing internal connections between modules. The table includes columns for Connection, ID, Status, From, To, Carrier, Resource, ID, and Status. Connections include Internal 0 (disconnected, /icub/cam/left to /icub/camcalib/left/in, udp, pc104 0 available), Internal 1 (disconnected, /icub/cam/right to /icub/camcalib/right/in, udp, icub-b14 1 available), Internal 2 (disconnected, /icub/camcalib/left/out to /icub/view/left, udp, icub-b15 2 available), and Internal 3 (disconnected, /icub/camcalib/right/out to /icub/view/right, udp, icub14 3 available).
- Messages:** A log window showing an error message: [ERR]: (Face) cannot stop yarpdev on pc104 : Timeout! Cannot stop yarpdev on /pc104.

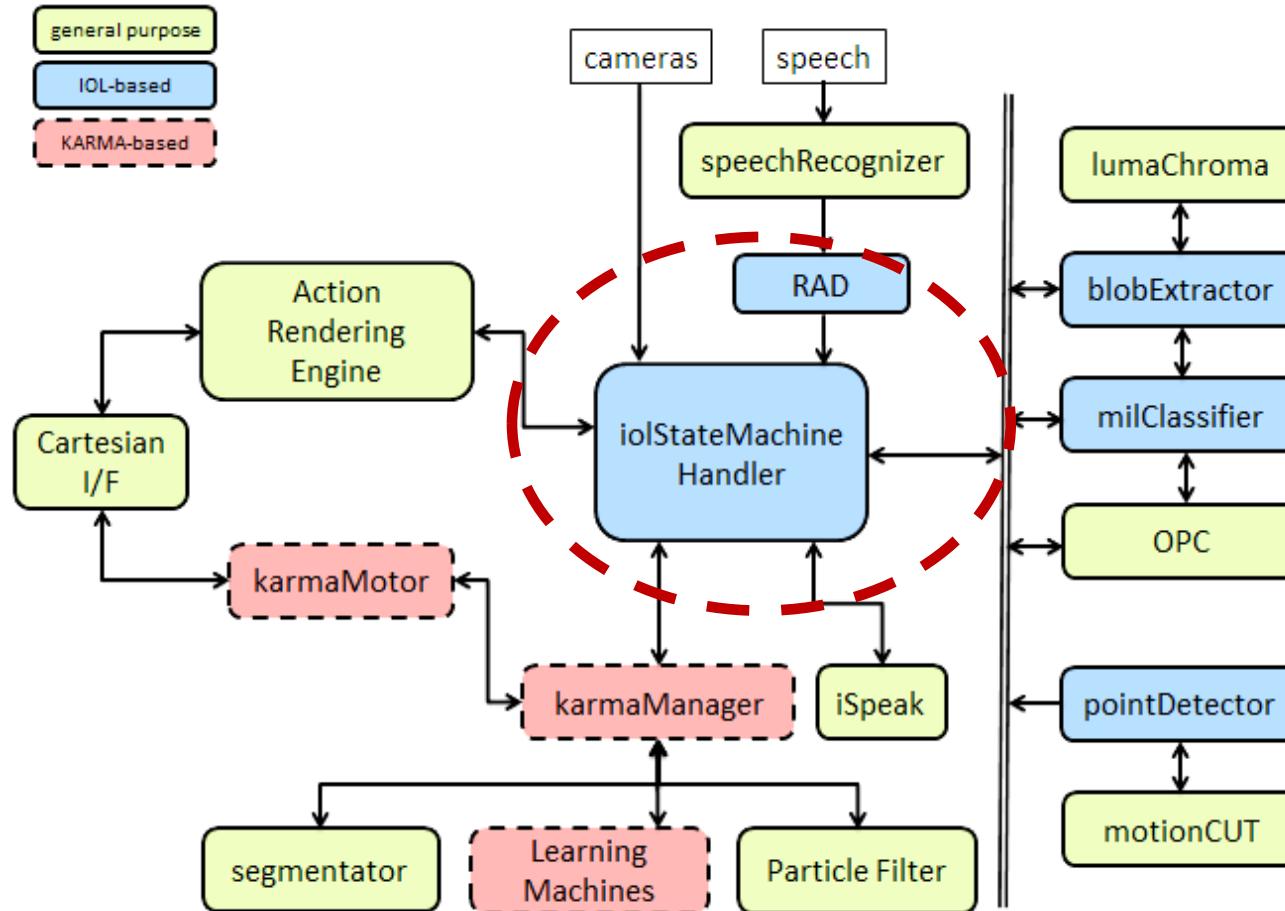


# Tools for rapid development

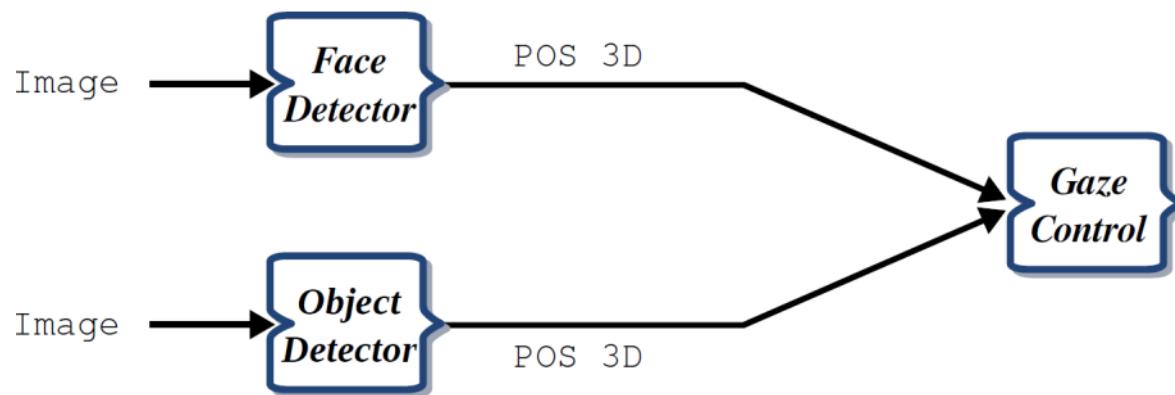
- YARP builder: graphical tool to design application
- Interface Definition Language (IDL):
  - formalization of types and interfaces between modules
  - automatic generation of message handlers



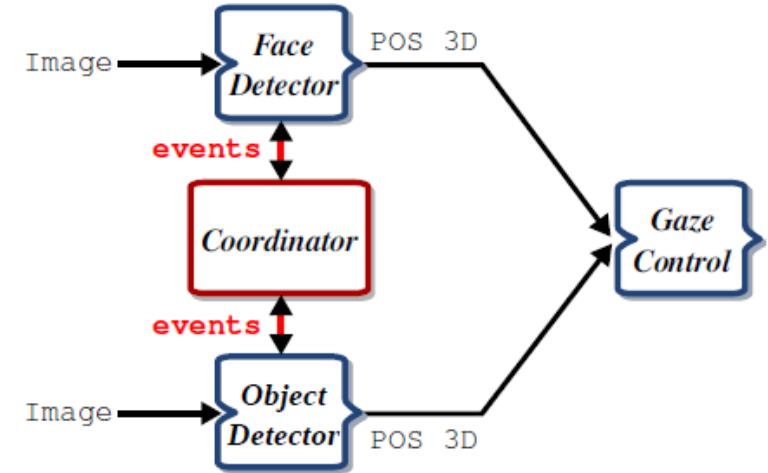
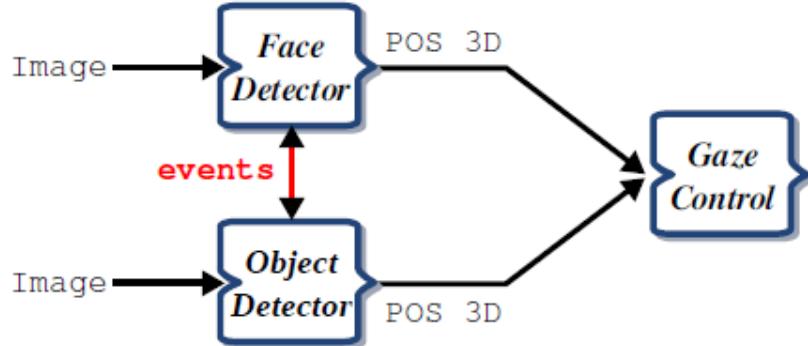
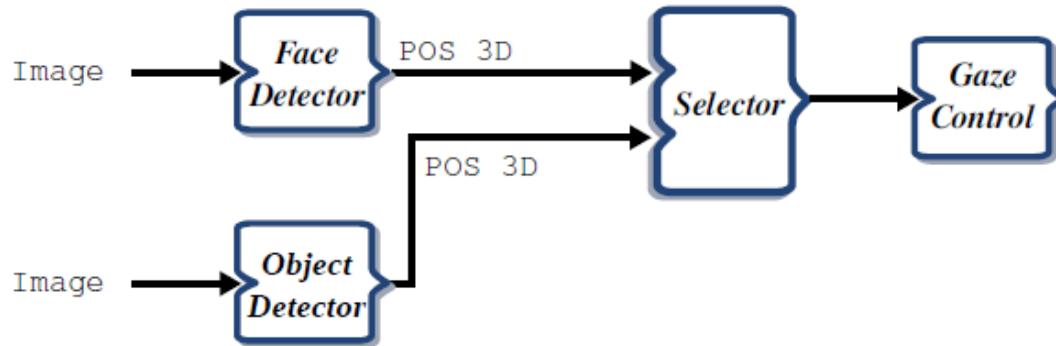
# Coordinating modules



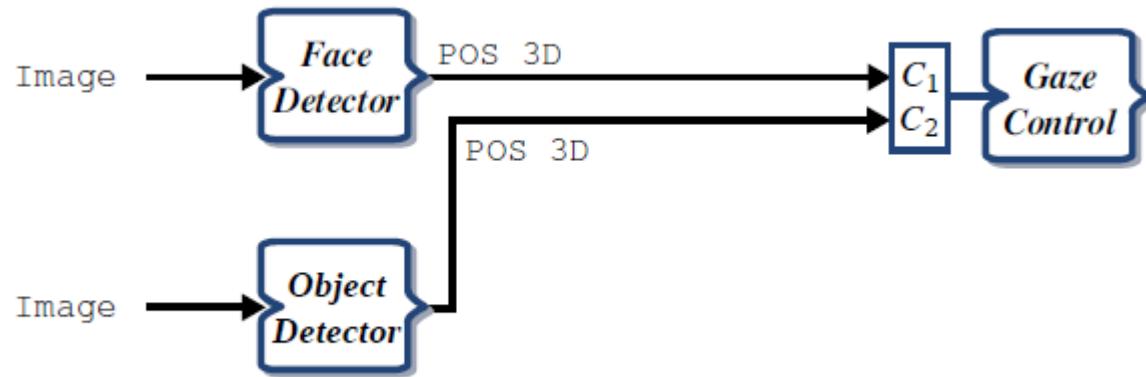
# Arbitration and coordination



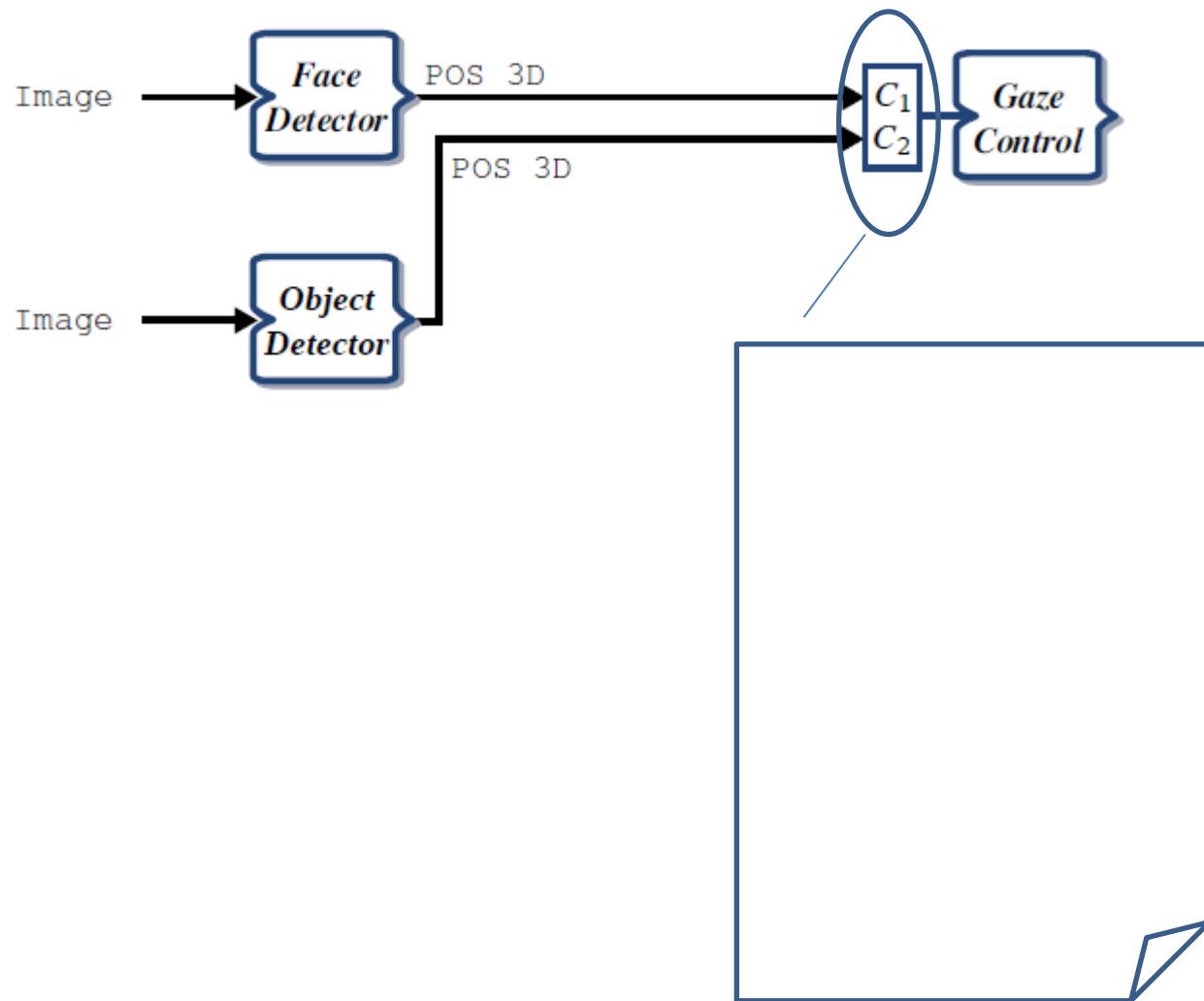
# Arbitration and coordination



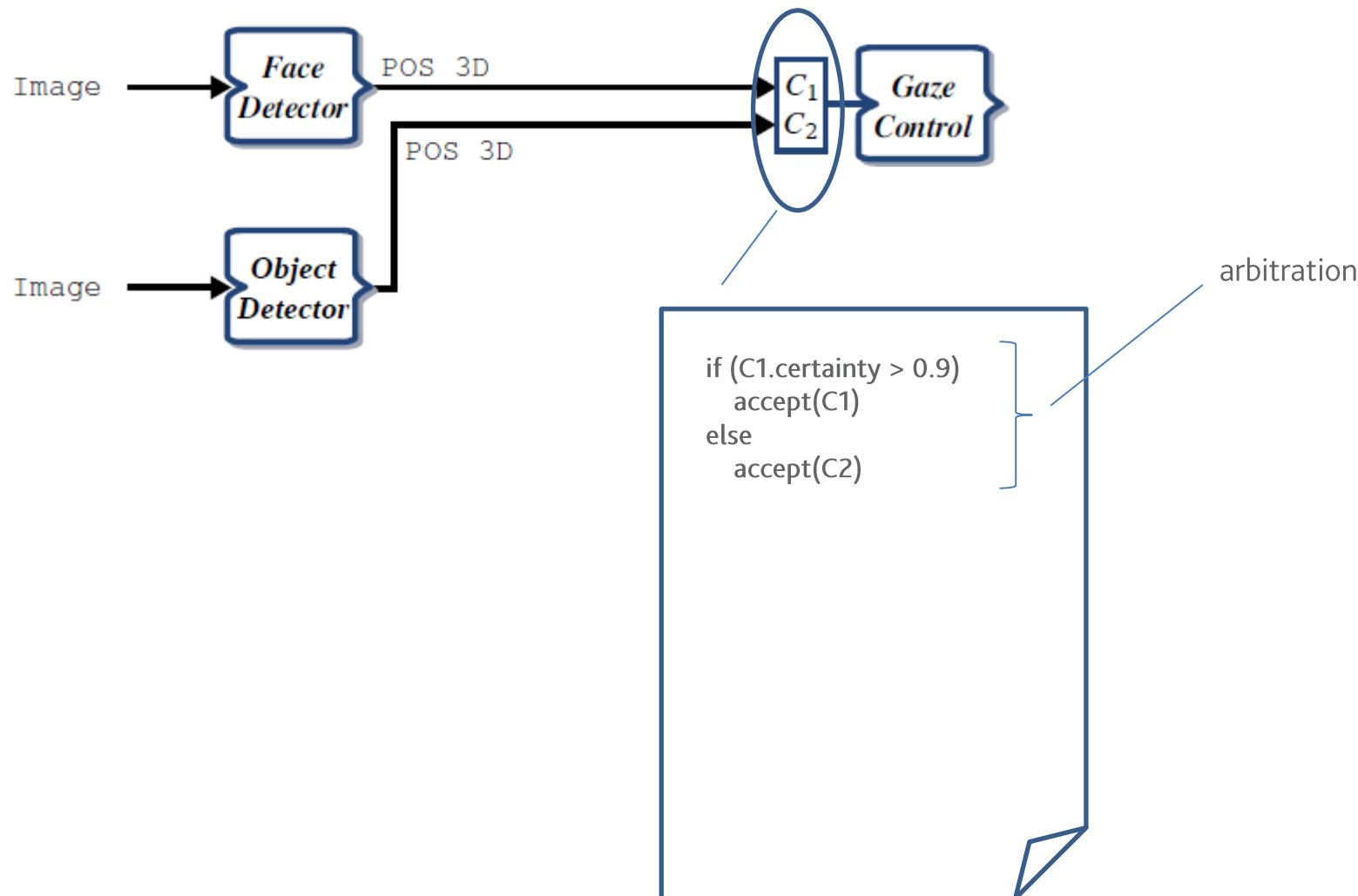
# Our solution: Port Monitor



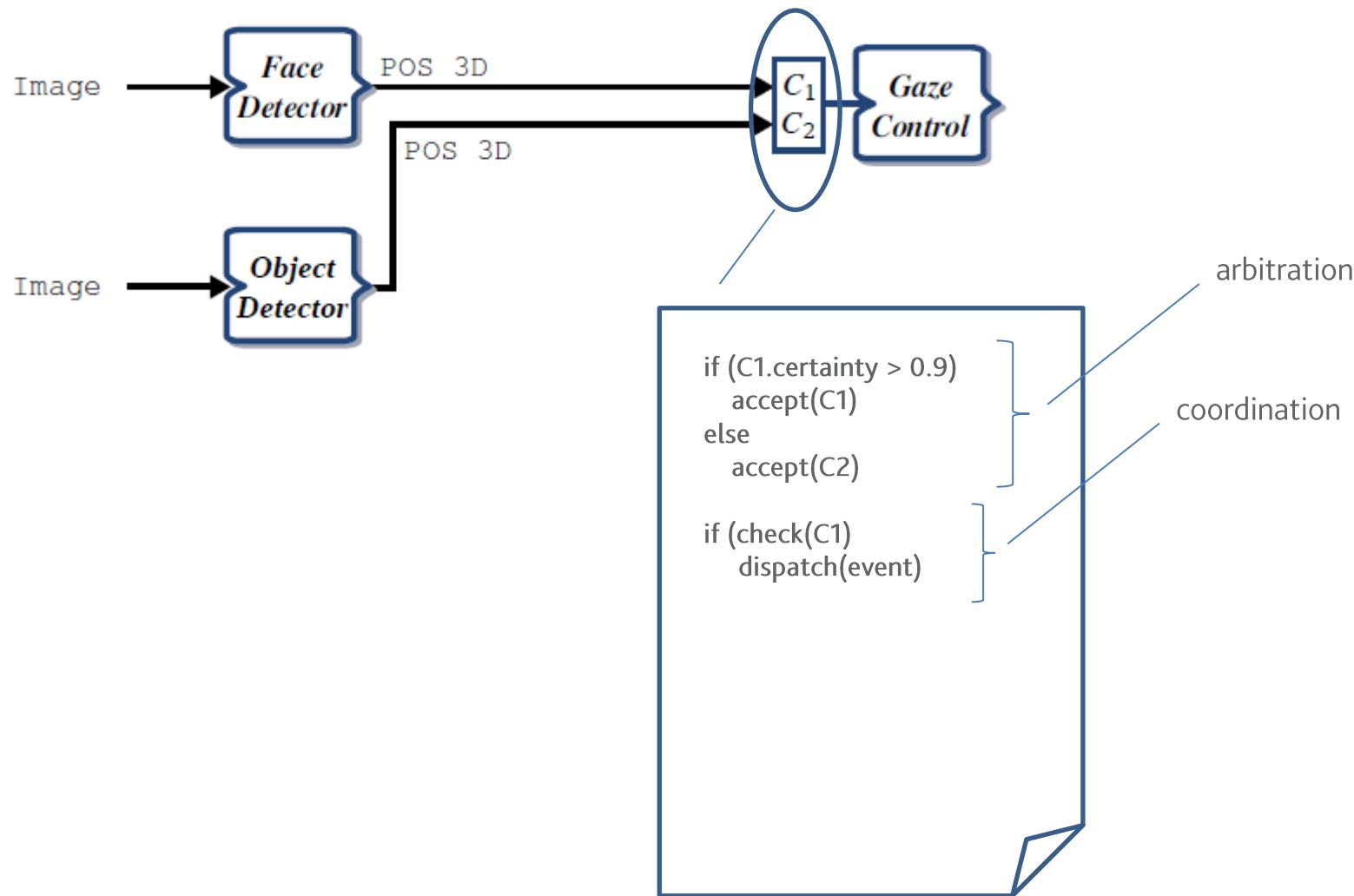
# Our solution: Port Monitor



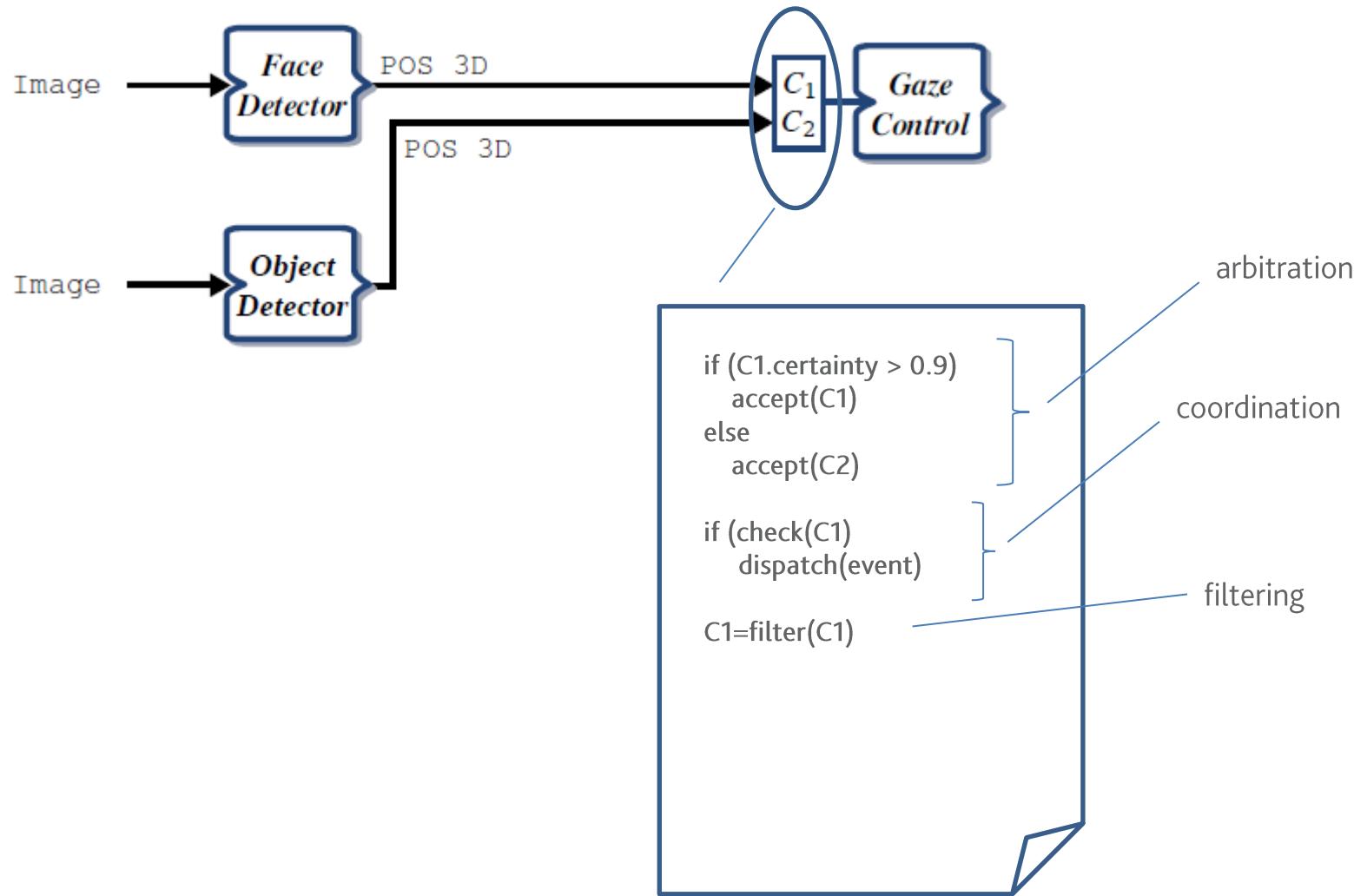
# Our solution: Port Monitor



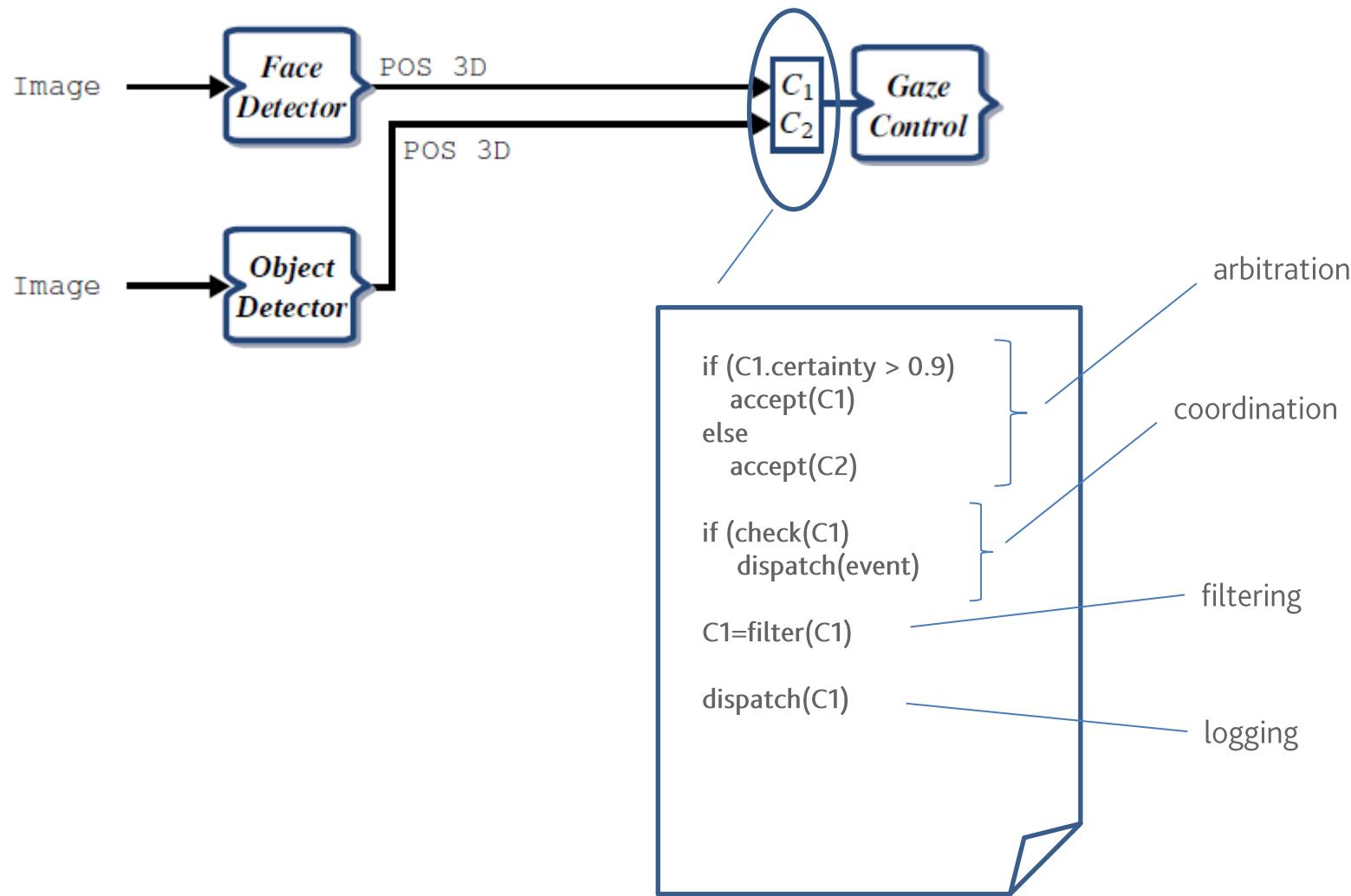
# Our solution: Port Monitor



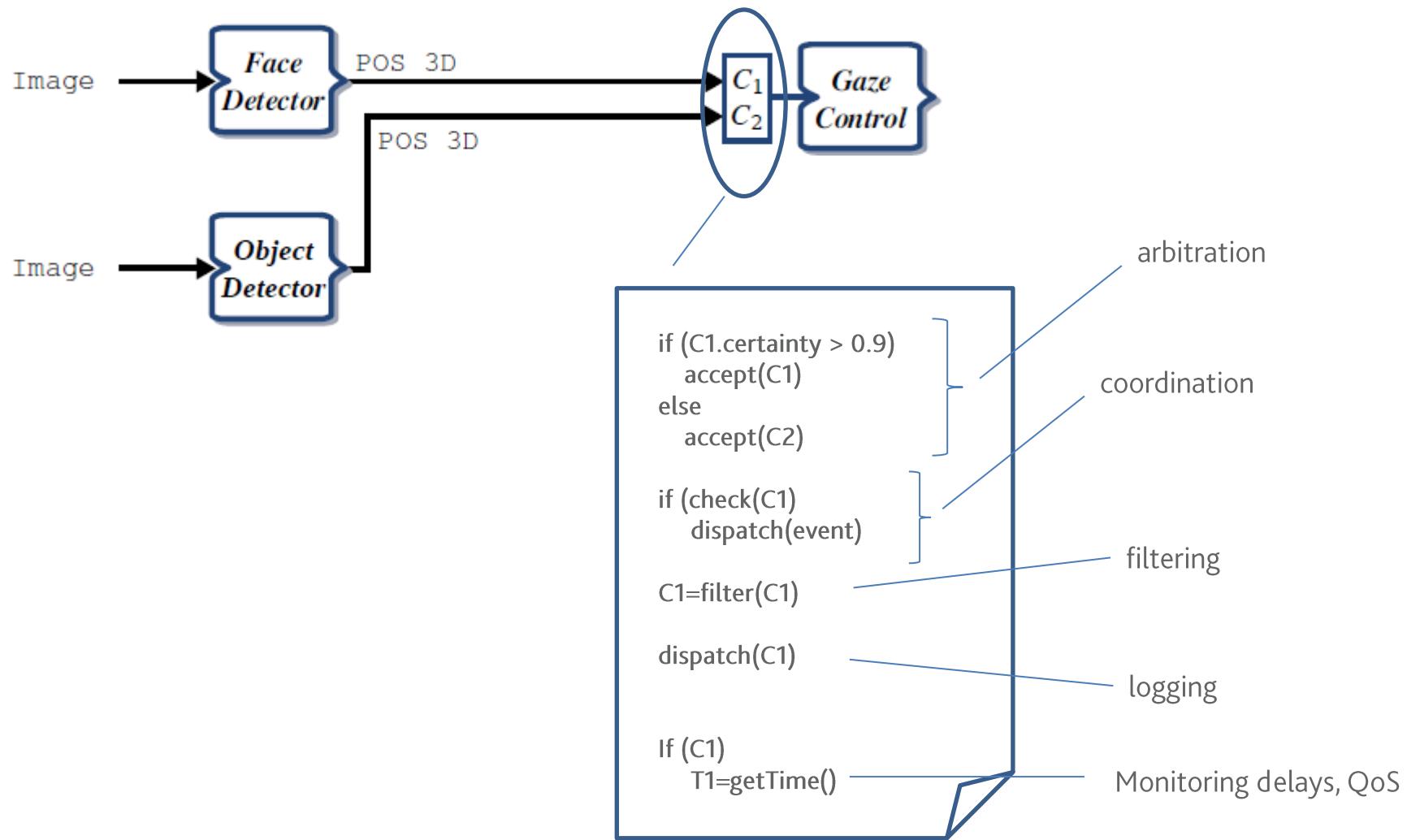
# Our solution: Port Monitor

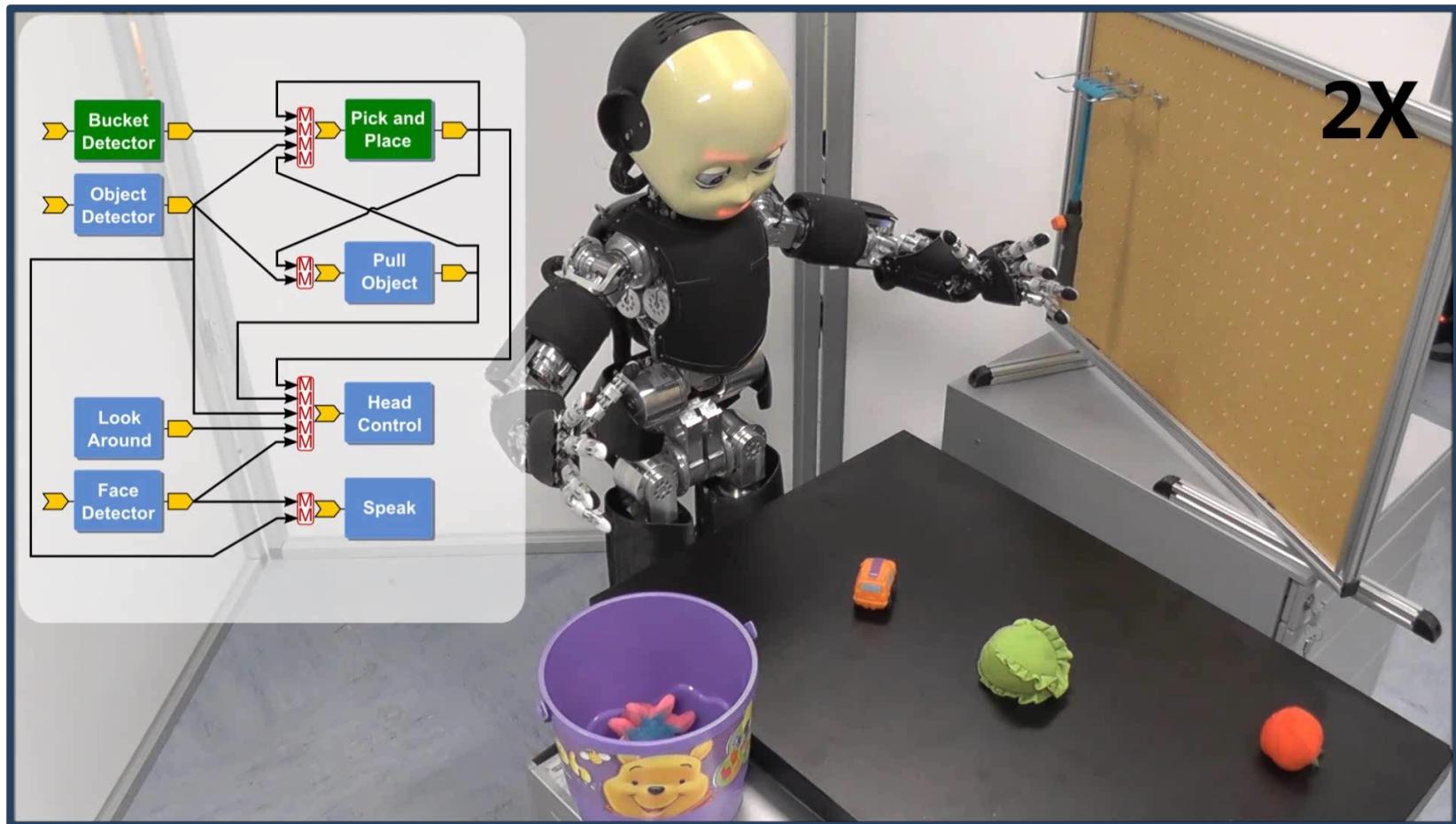


# Our solution: Port Monitor



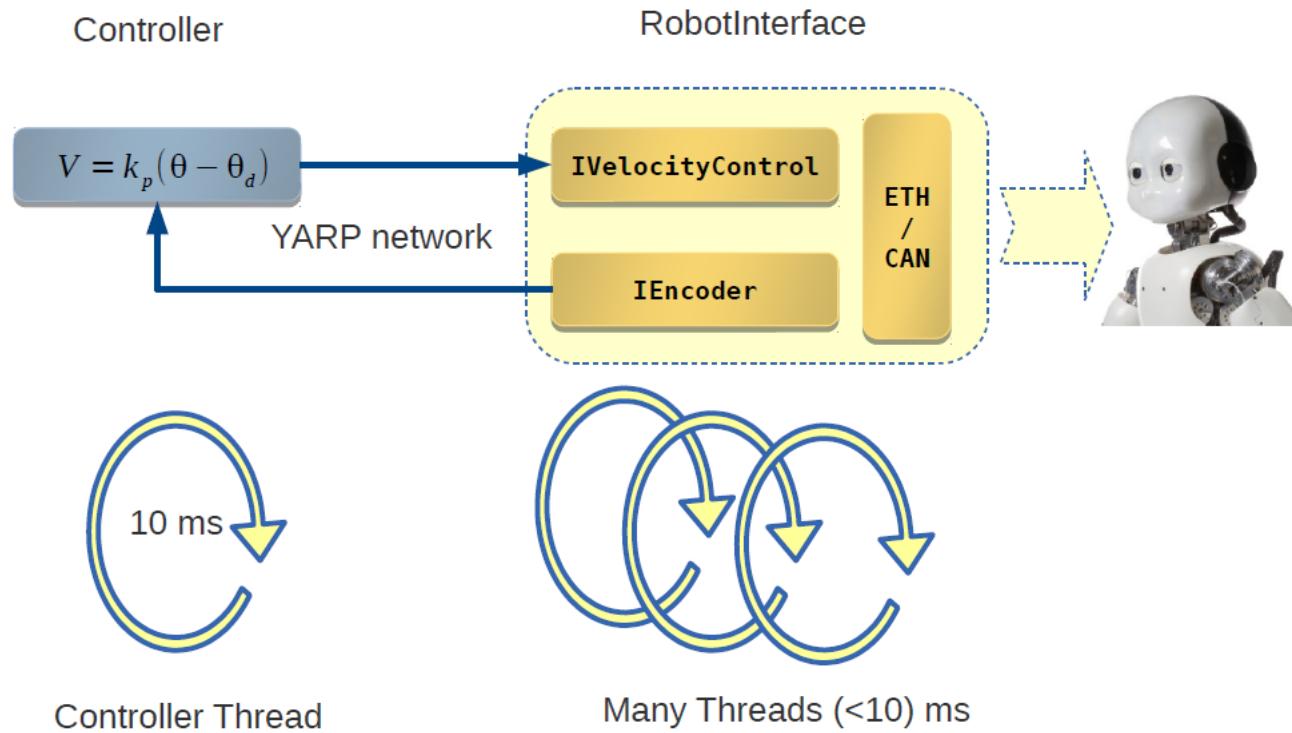
# Our solution: Port Monitor





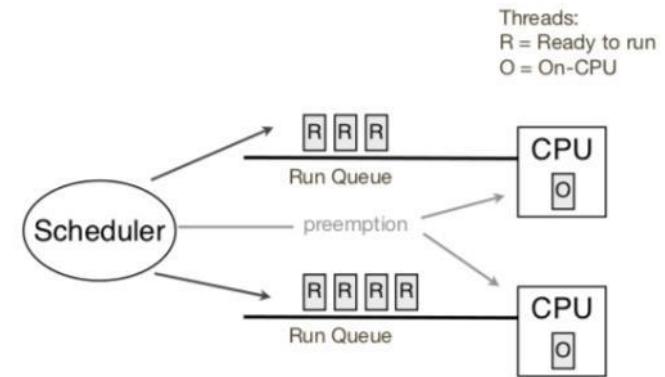
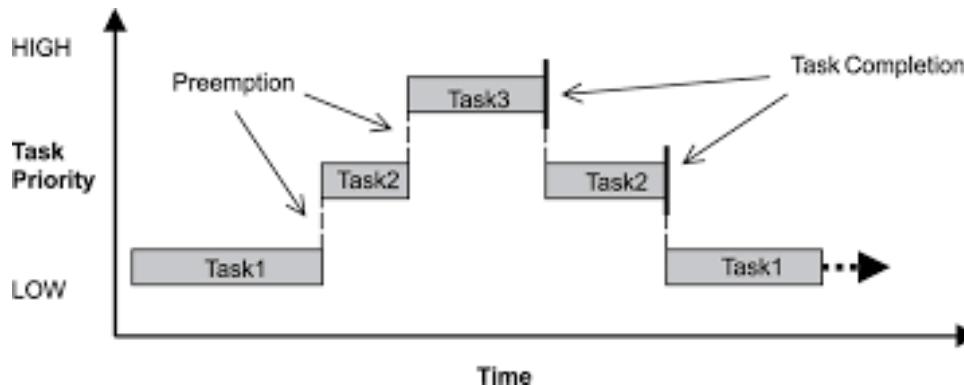
Paikan, A., et al., *Enhancing software module reusability using port plug-ins: an experiment with the iCub robot*, IROS 2014.

# Improving determinism



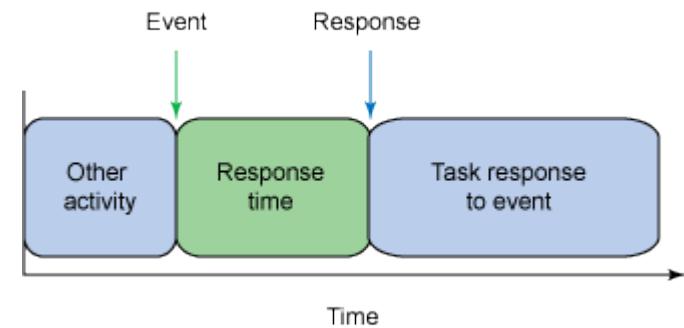
Quality of Service

# Threads and scheduling, quick overview



## Issues:

- Scheduling and interrupt latency
- Certain operations may be *blocking* i.e. not preemptable (usually system calls)
- System calls: file management, process management, I/O, memory requests, communication (shared memory), error handling



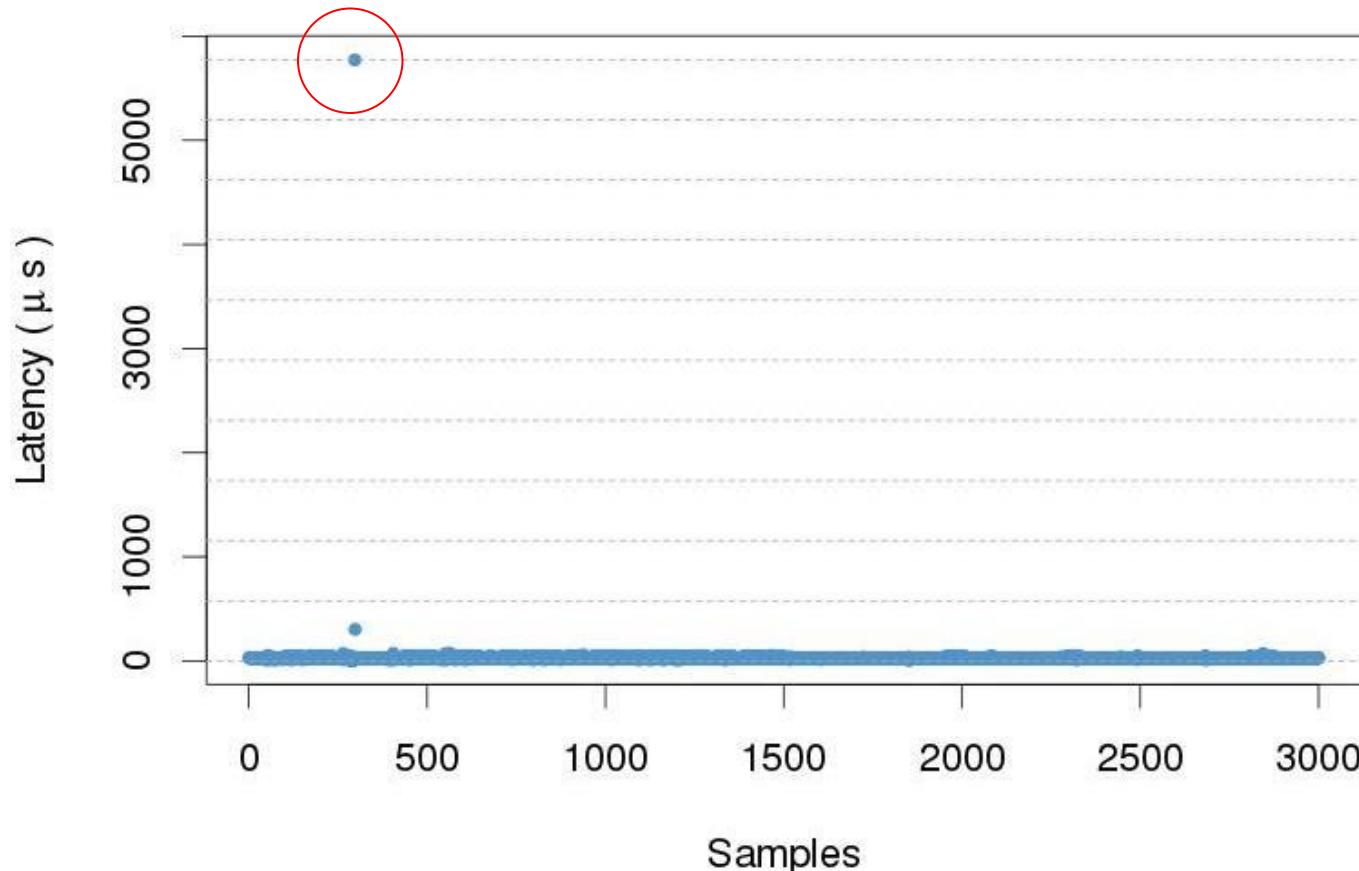
# Which RT solution

- Standard Linux
- Linux with RT patch
- Xenomai/RTAI

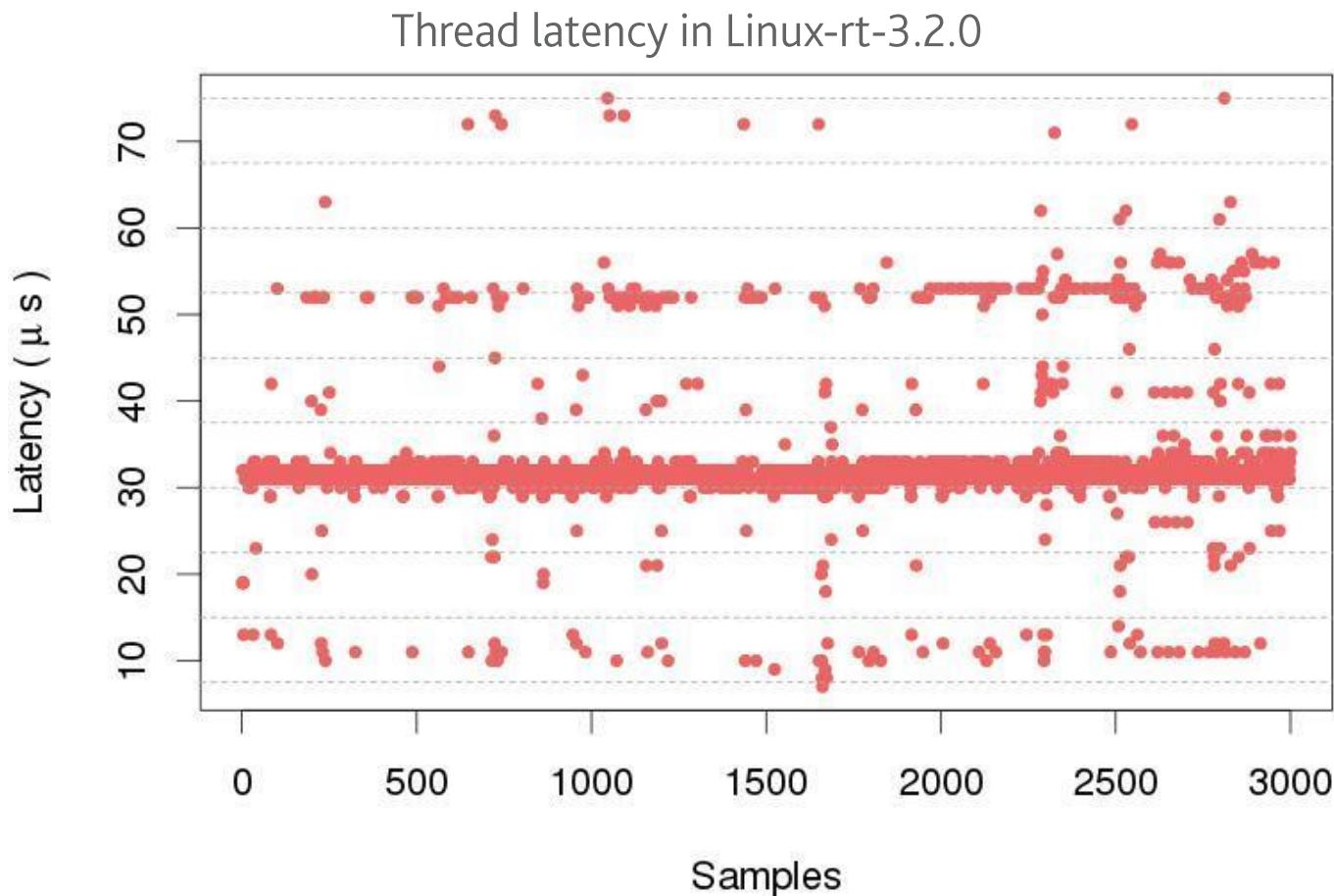
# The latency of a Linux thread with the highest priority (period 1ms)

(period: 1ms, PC104 / 1Ghz )

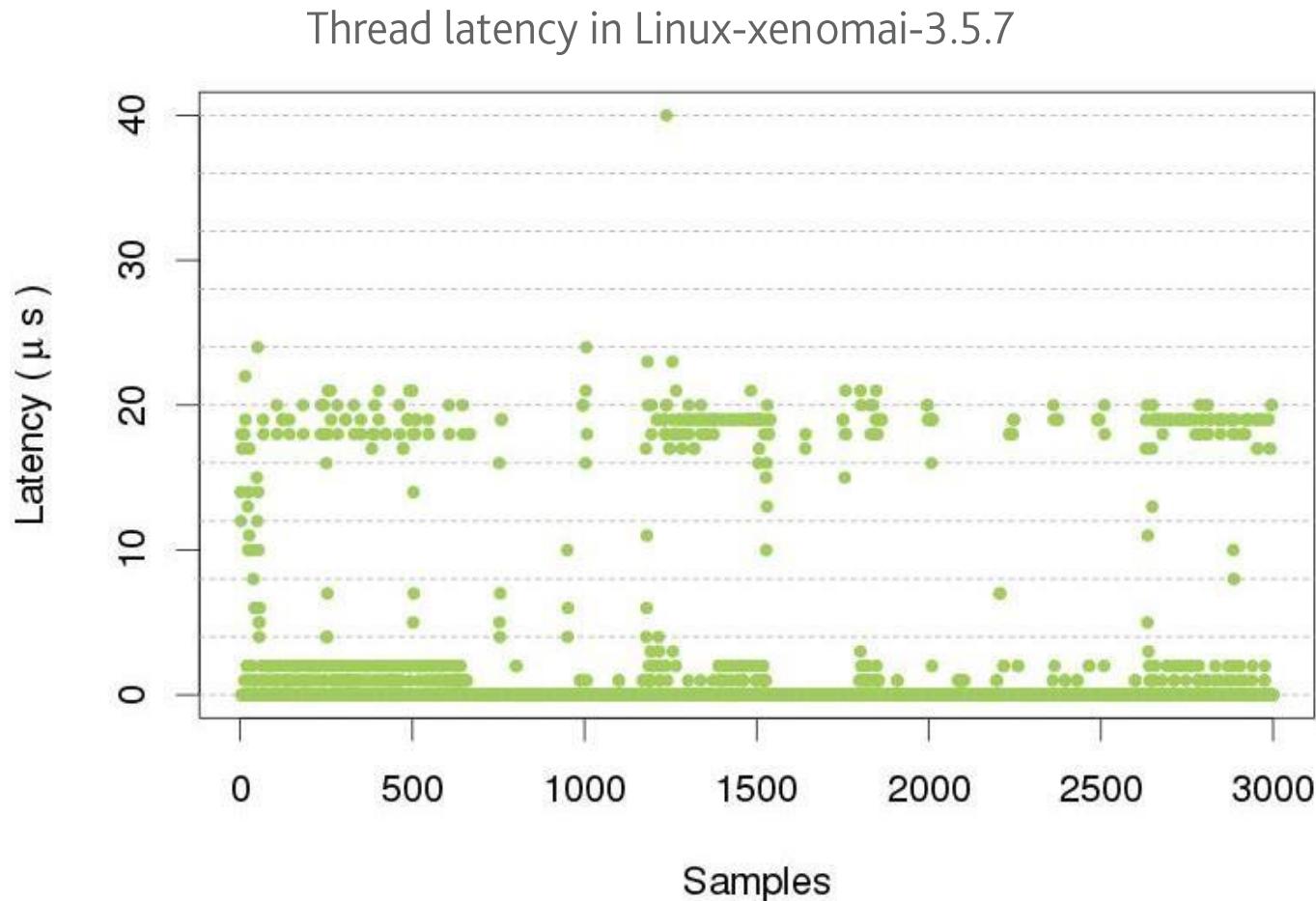
Thread latency in Linux-pae-3.2.0 (SCHED\_FIFO)



# The latency of a Linux thread with preemption patch (period: 1ms, PC104 / 1Ghz )



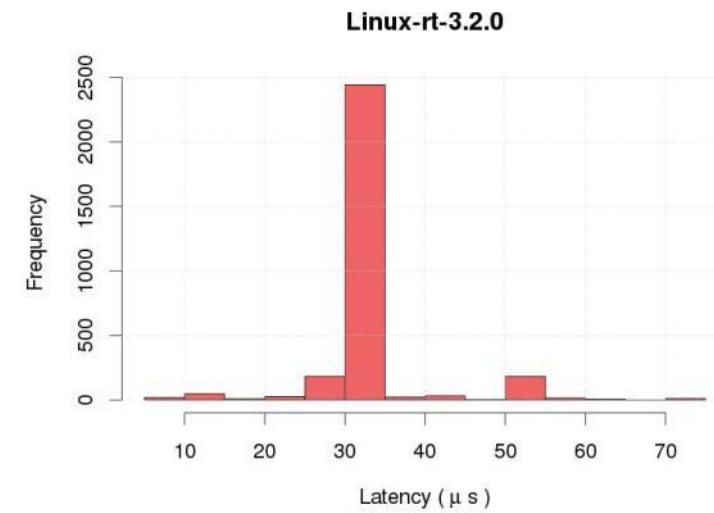
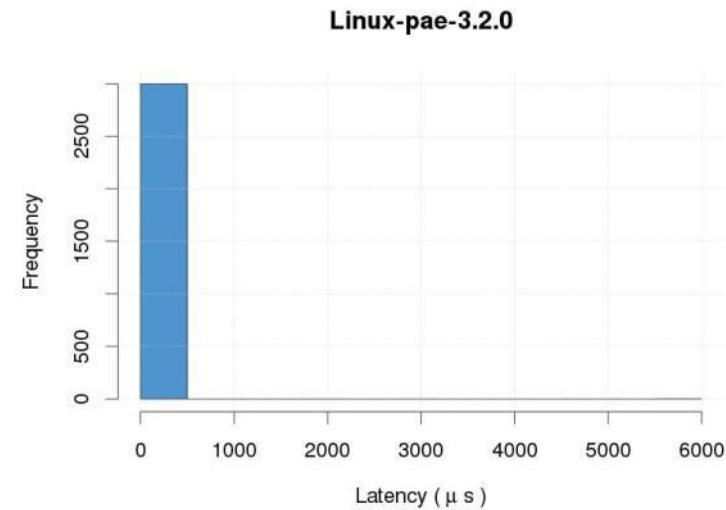
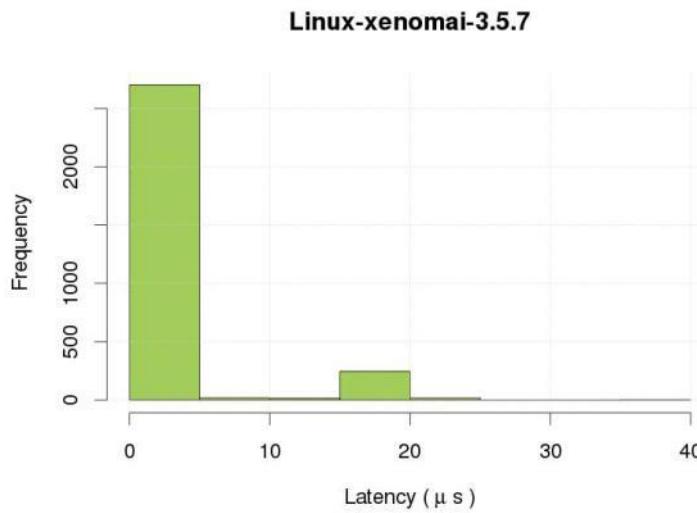
# The latency of a Linux thread with Xenomai patch (period: 1ms, PC104 / 1Ghz )



# Which RT solution?

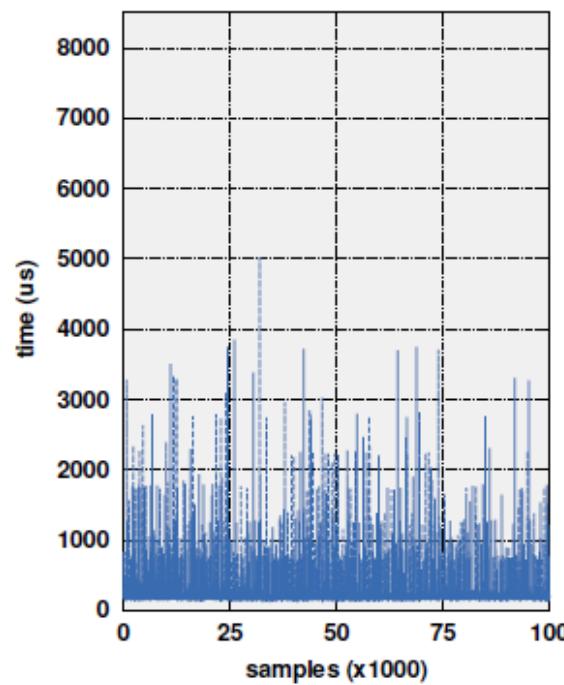
Standard Linux  
Linux with RT patch  
Xenomai/RTAI

After preliminary evaluation we decided to adopt *Linux with RT patch*

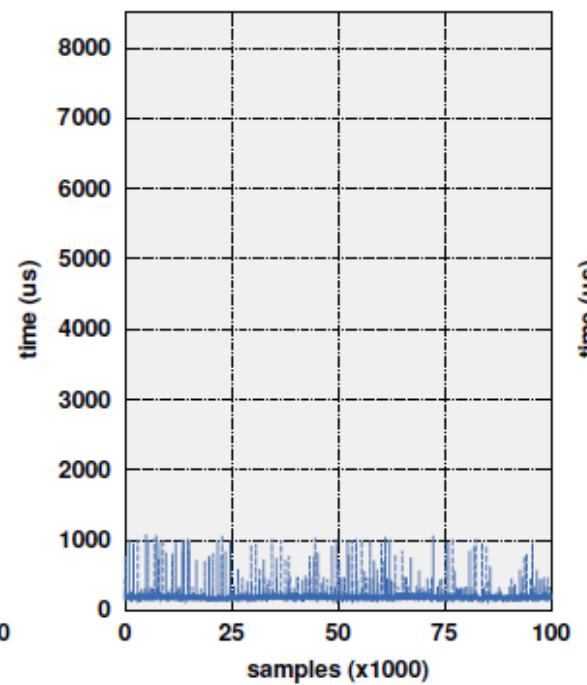




Standard YARP

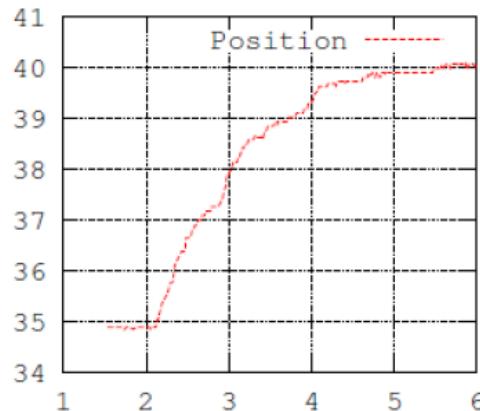


With priority

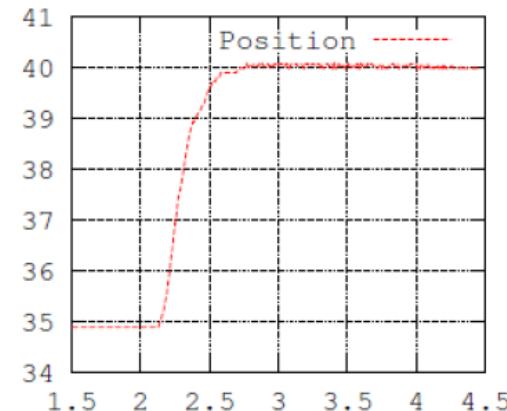


# System under heavy load

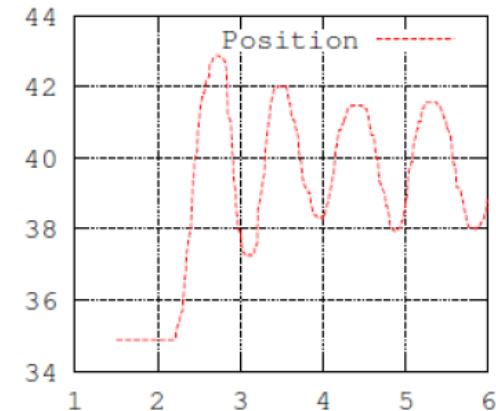
$V = 1.0 (\theta - \theta_d)$   
max delay: 44.4 ms



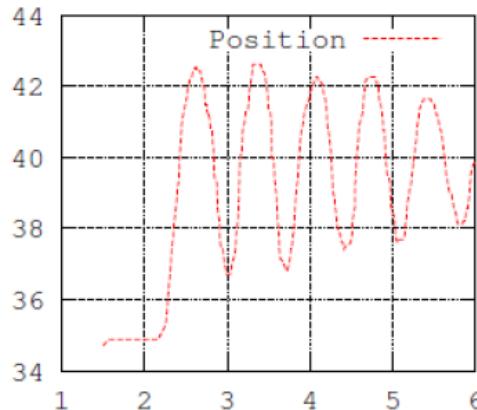
$V = 5.0 (\theta - \theta_d)$   
max delay: 41.4 ms



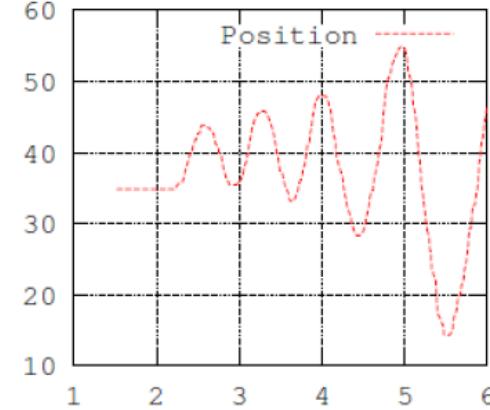
$V = 6.0 (\theta - \theta_d)$   
max delay: 65.9 ms



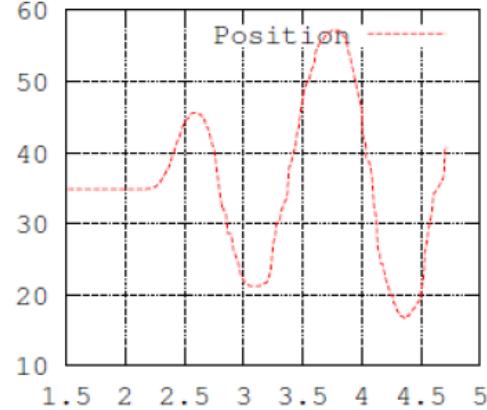
$V = 8.0 (\theta - \theta_d)$   
max delay: 53.6 ms



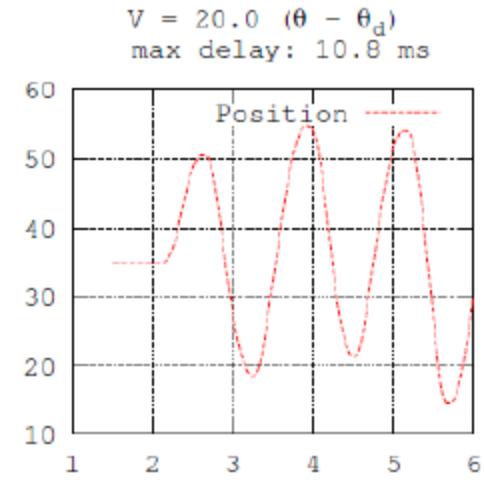
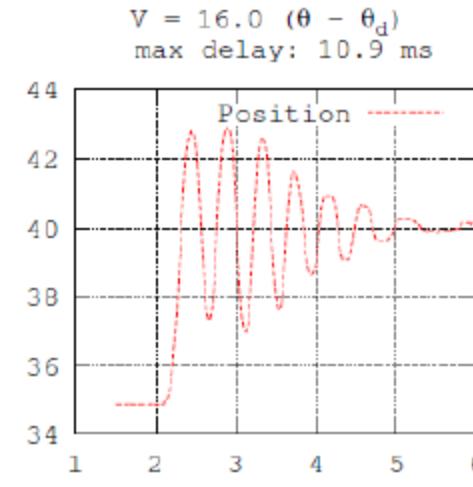
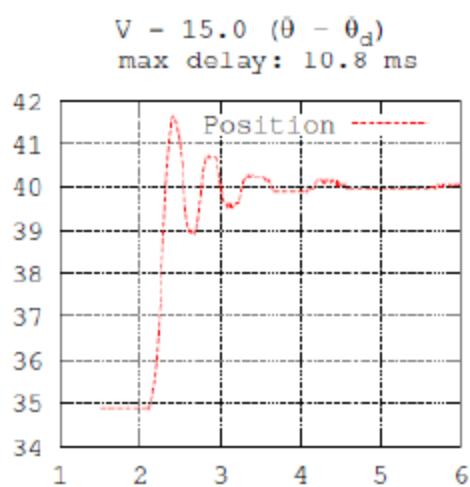
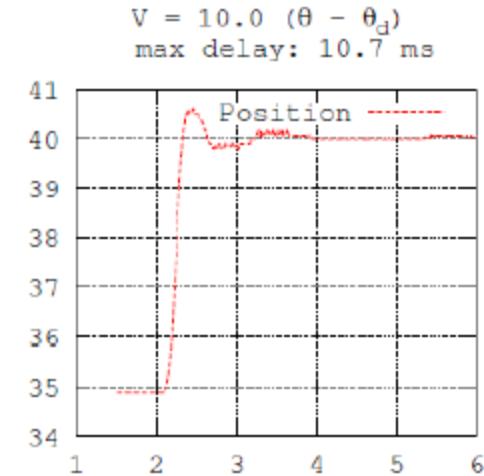
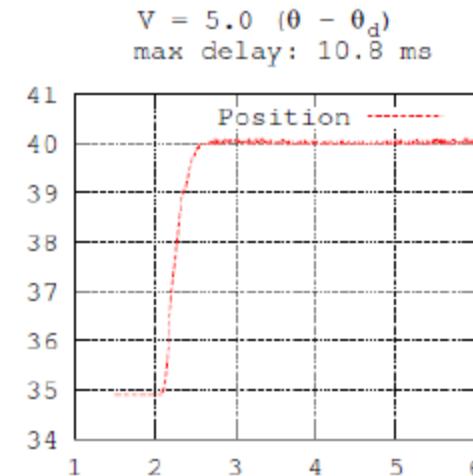
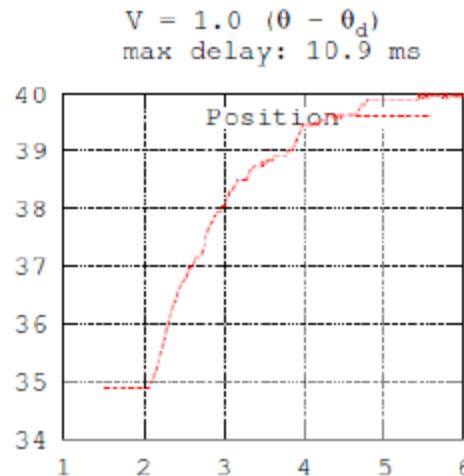
$V = 10.0 (\theta - \theta_d)$   
max delay: 48.9 ms



$V = 15.0 (\theta - \theta_d)$   
max delay: 46.3 ms



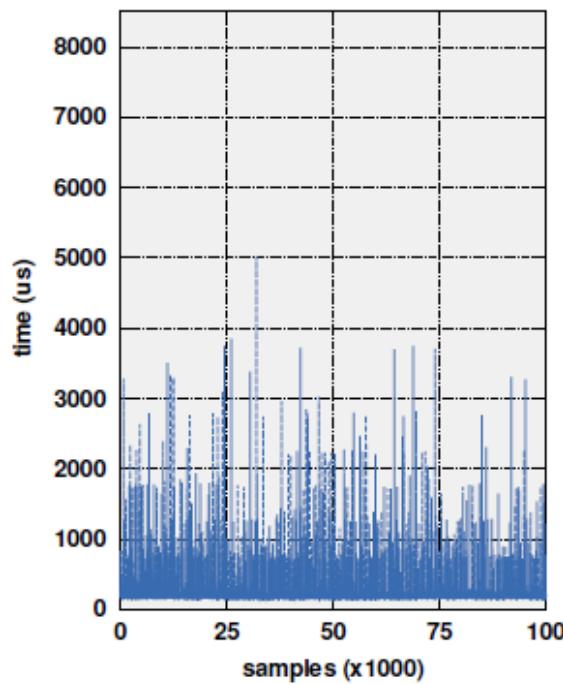
# After adjusting thread priorities



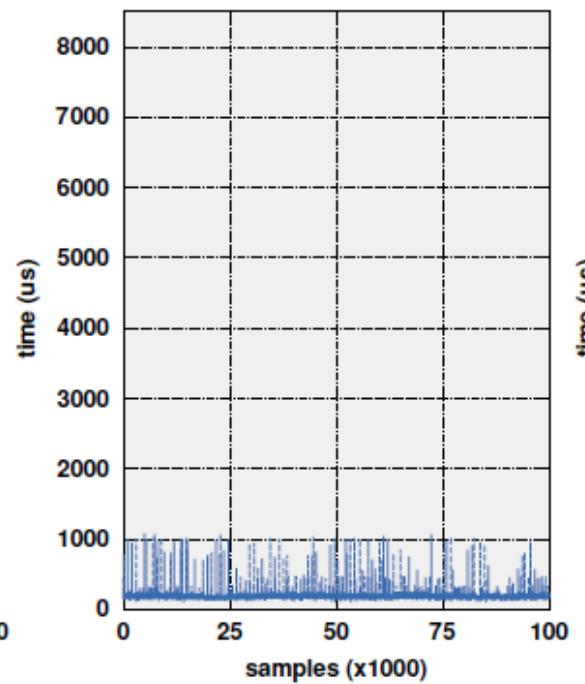


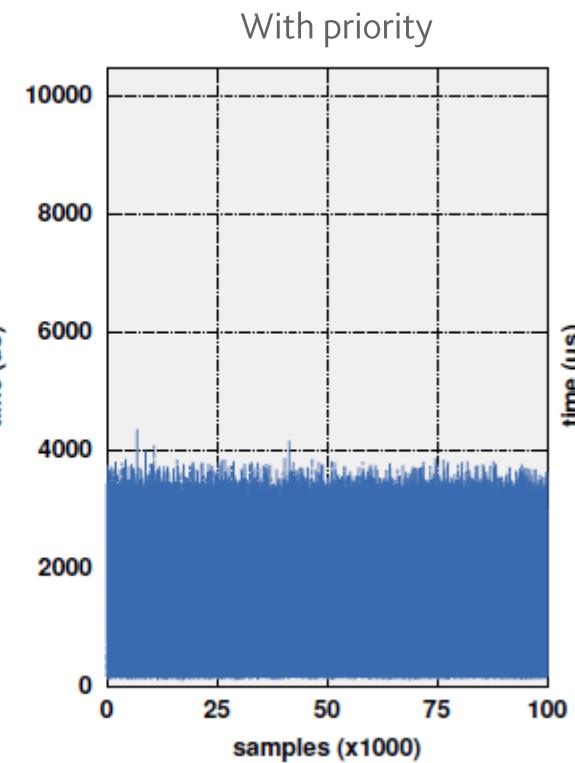
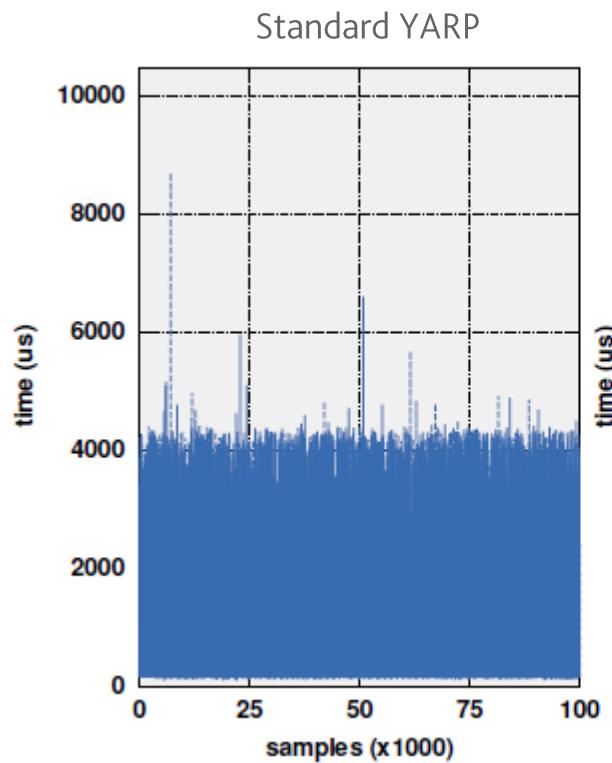
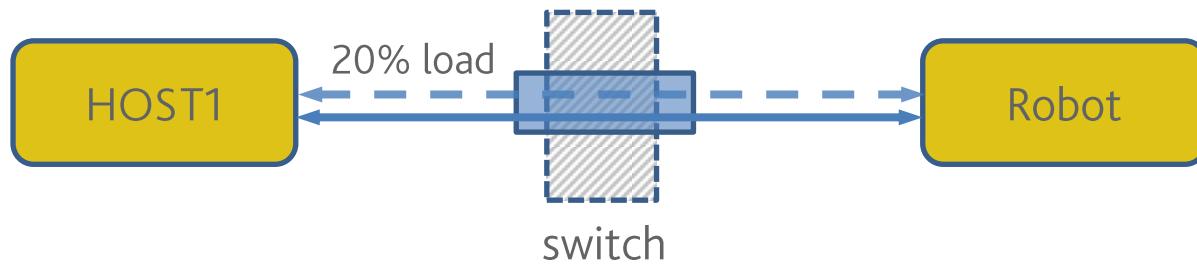
switch

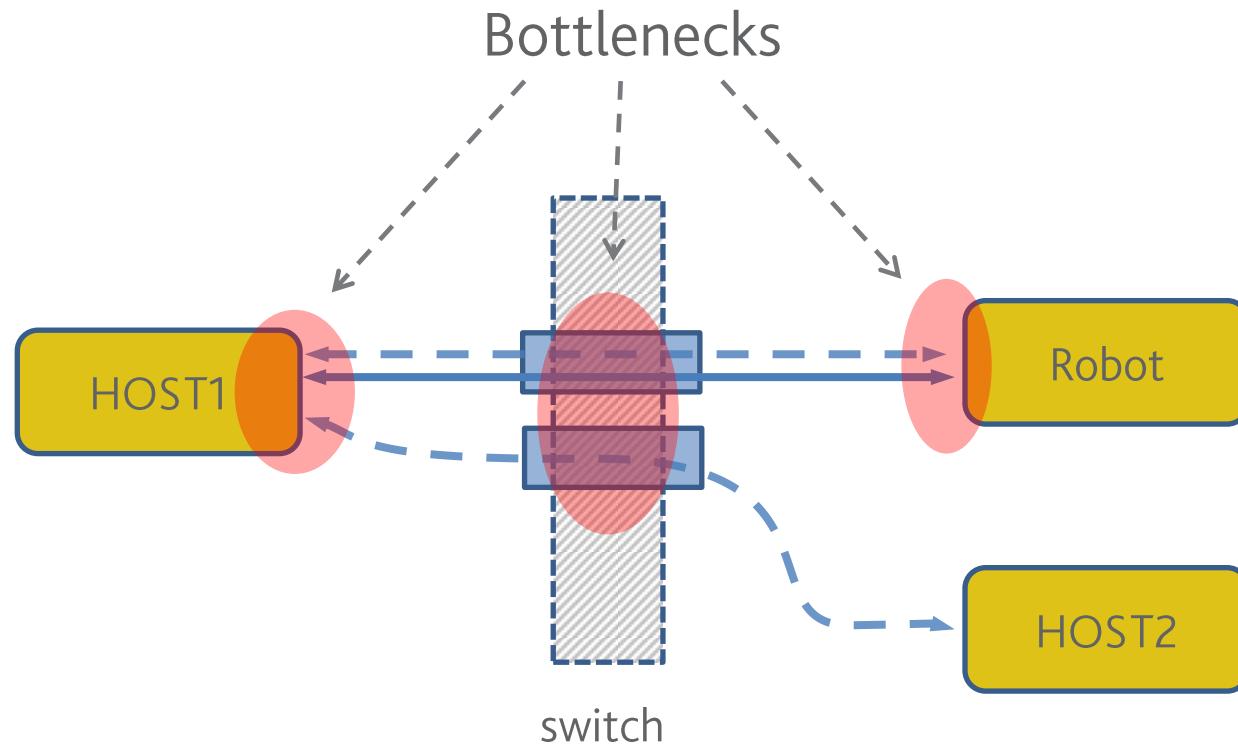
Standard YARP



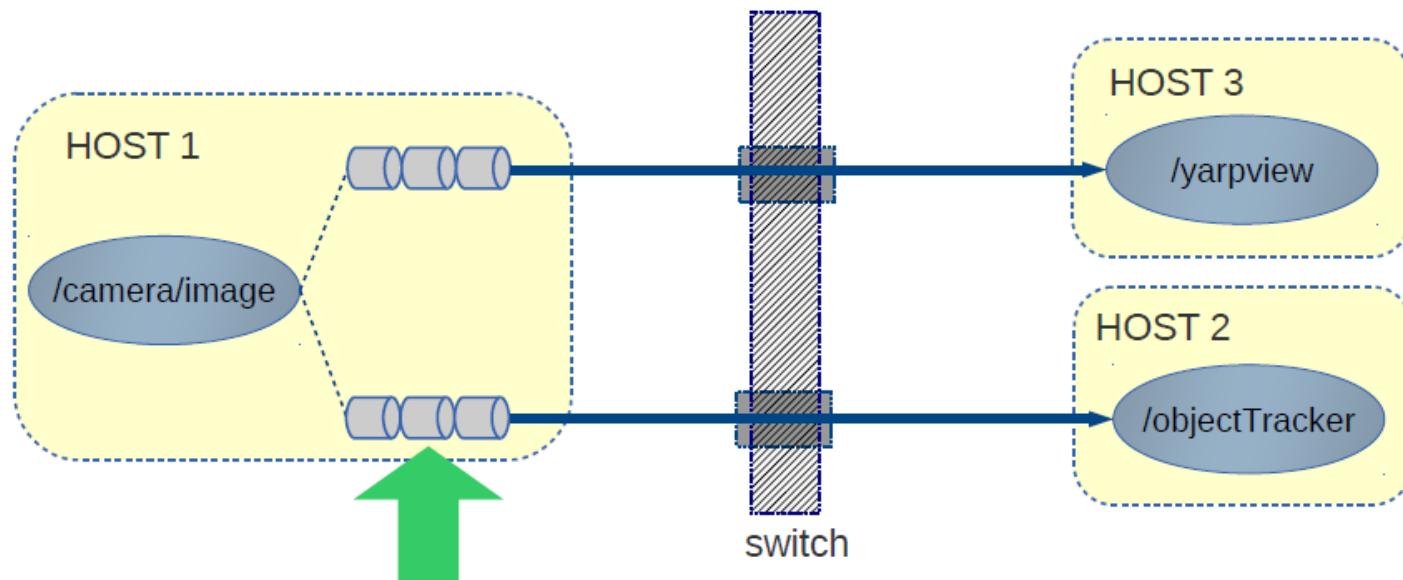
With priority



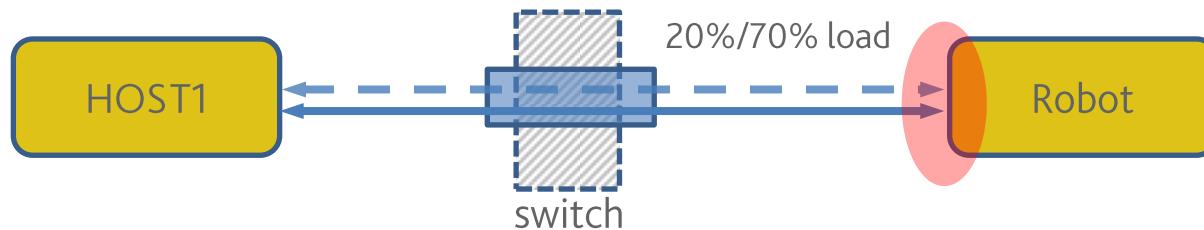




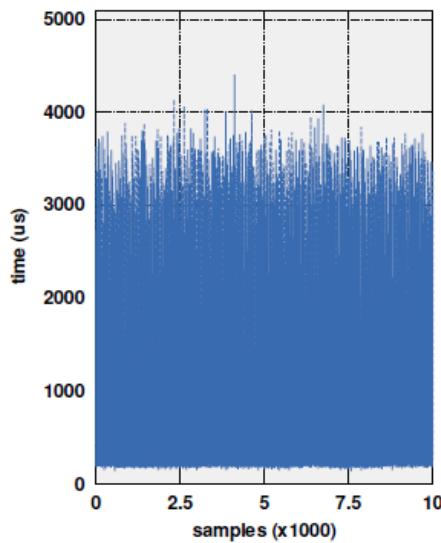
- Approach: improve determinism by increasing **thread priorities** and reducing network bottlenecks using **QoS**



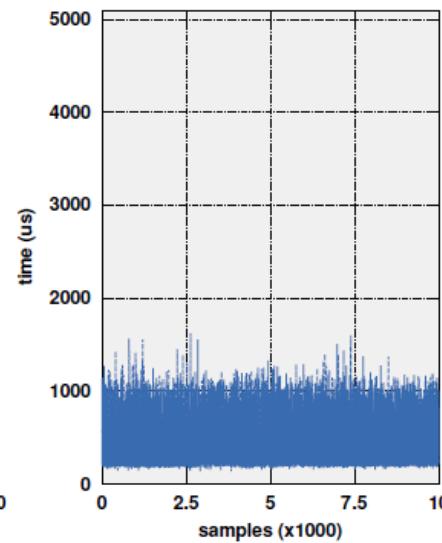
```
> prop sched policy 1 priority 30  
> prop set qos priority HIGH
```



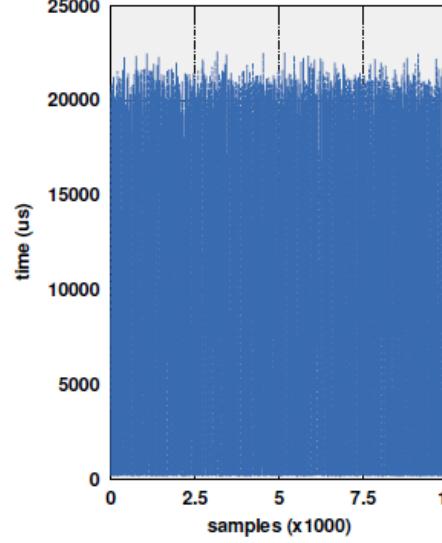
Standard YARP (20%)



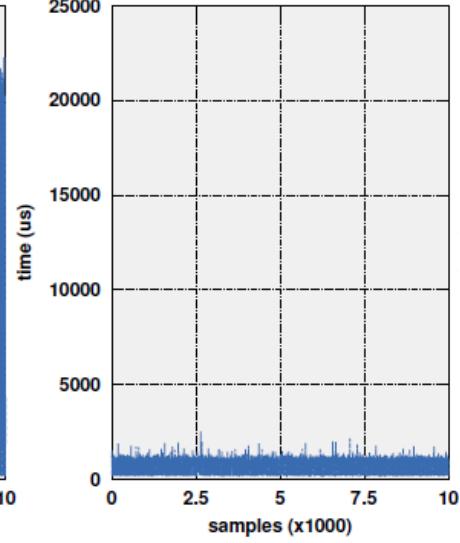
With priority (20%)

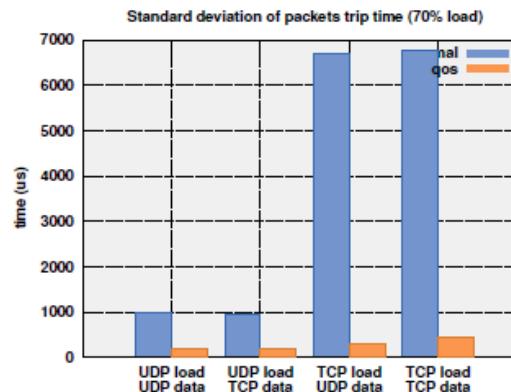
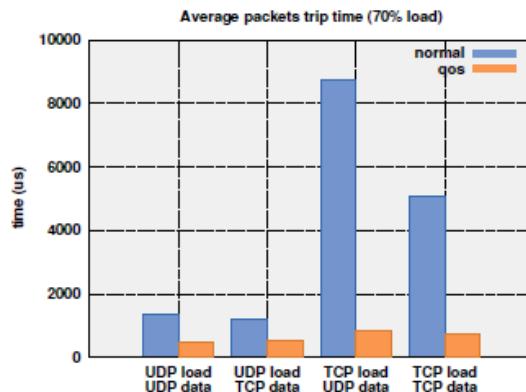
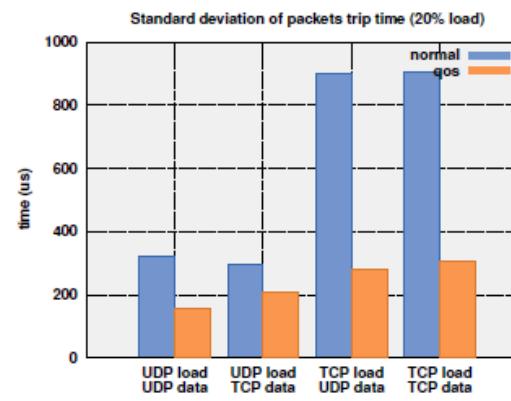
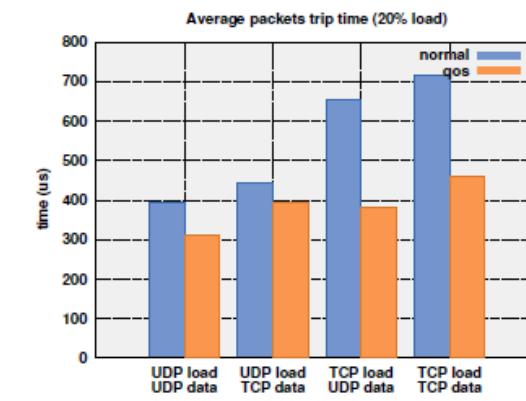
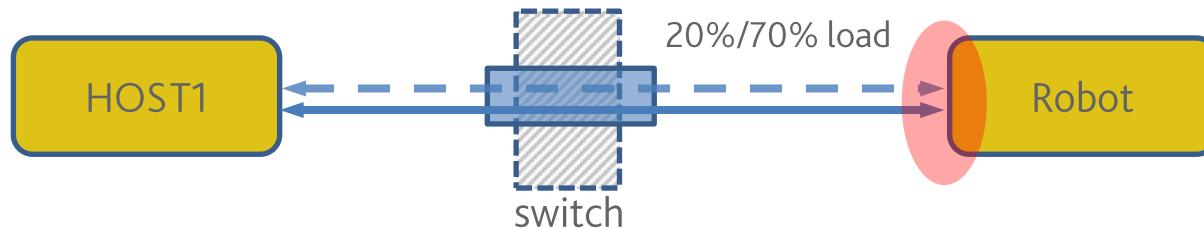


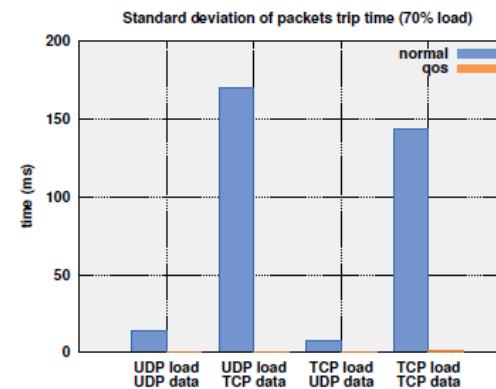
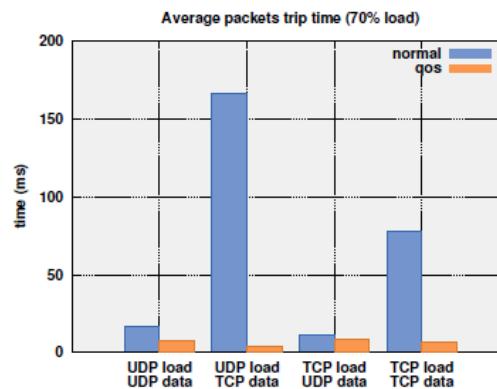
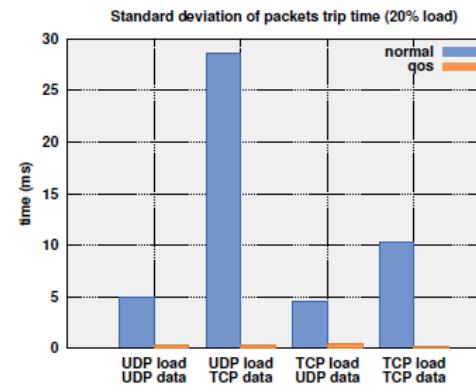
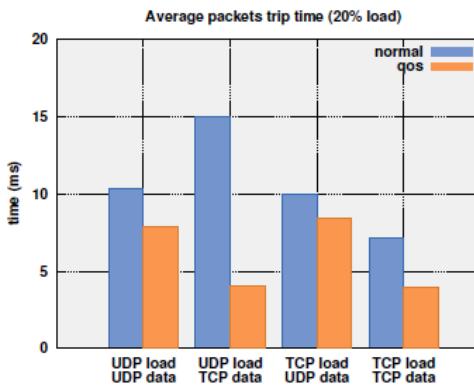
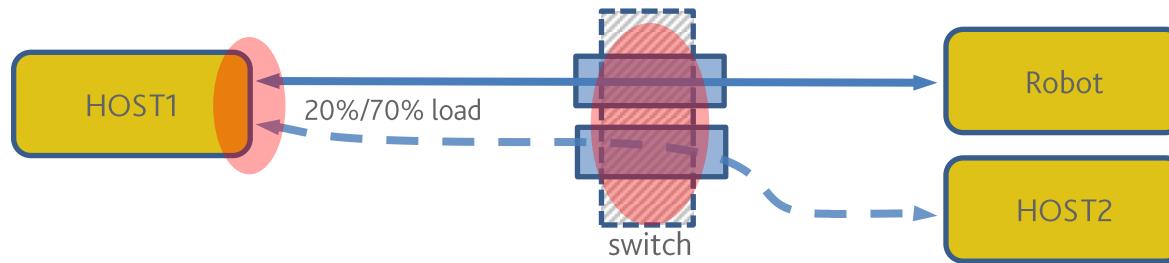
Standard YARP (70%)



With priority (70%)







# References

- Ali Paikan, Silvio Traversaro, Francesco Nori and Lorenzo Natale, "A Generic Testing Framework for Test Driven Development of Robotic Systems", In Lecture Notes in Computer Science (MESAS15), Springer, pp. , 2015
- Paikan, A., Domenichelli, D., and Natale, L., *Communication channel prioritization in a publish-subscribe architecture*, in Proc. Software Engineering and Architectures for Realtime Interactive Systems Workshop, Arles, France, 2015.
- Paikan, A., Tikhanoff, V., Metta, G., and Natale, L., *Enhancing software module reusability using port plug-ins: an experiment with the iCub robot*, in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, Illinois, 2014.
- Fitzpatrick, P., Ceseracciu, E., Domenichelli, D., Paikan, A., Metta, G., and Natale, L., *A middle way for robotics middleware*, Journal of Software Engineering for Robotics, vol. 5, no. 2, pp. 42-49, 2014.
- Paikan, A., Fitzpatrick, P., Metta, G., and Natale, L., *Data Flow Port's Monitoring and Arbitration*, Journal of Software Engineering for Robotics, vol. 5, no. 1, pp. 80-88, 2014
- Metta, G., Fitzpatrick, P., Natale, L., *YARP: Yet Another Robot Platform*, International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics, Volume 3, Issue 1, pp. 43-48, March 2006
- Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 2 (June 2003), 114-131.
- David, B. Stewart, Richard, A. Volpe, Pradeep, K. Khosla, Design of Dynamically Reconfigurable Real-Time Software Using Port-Based Objects, *IEEE Trans. On Software Engineering*, 23(12), 1997
- Dušan Bálek, František Plášil, Software Connectors and Their Role in Component Deployment, *New Developments in Distributed Applications and Interoperable Systems*, 70, 2002, pp. 69-84.
- Farhad Arbab: Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science* 14(3):329--366, 2004.