RobotCub

Towards Long-Lived Robot Software

Paul Fitzpatrick

Workshop on Humanoid Technologies

Humanoids 2006

Talk Credits:
Some material from Assif Mirza

Cogsys
Cognitive Systems

RobotCub

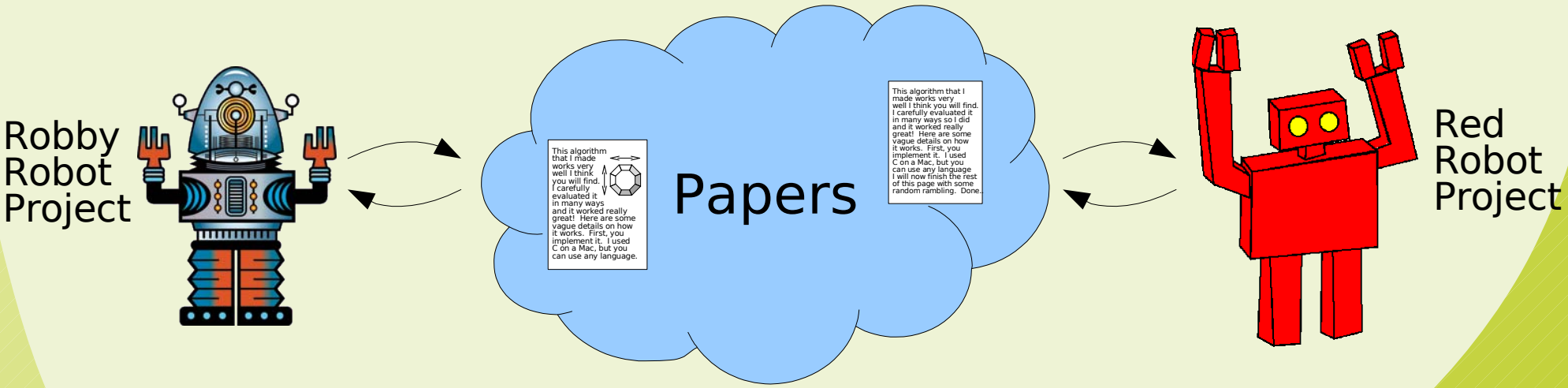
Outline

- The sad fate of most robot software
- Modularity in robotics
- YARP: Yet Another Robot Platform
- Excising communication "plumbing" from code
- Excising device dependencies from code
- Conclusions

Cogsys
Cognitive Systems

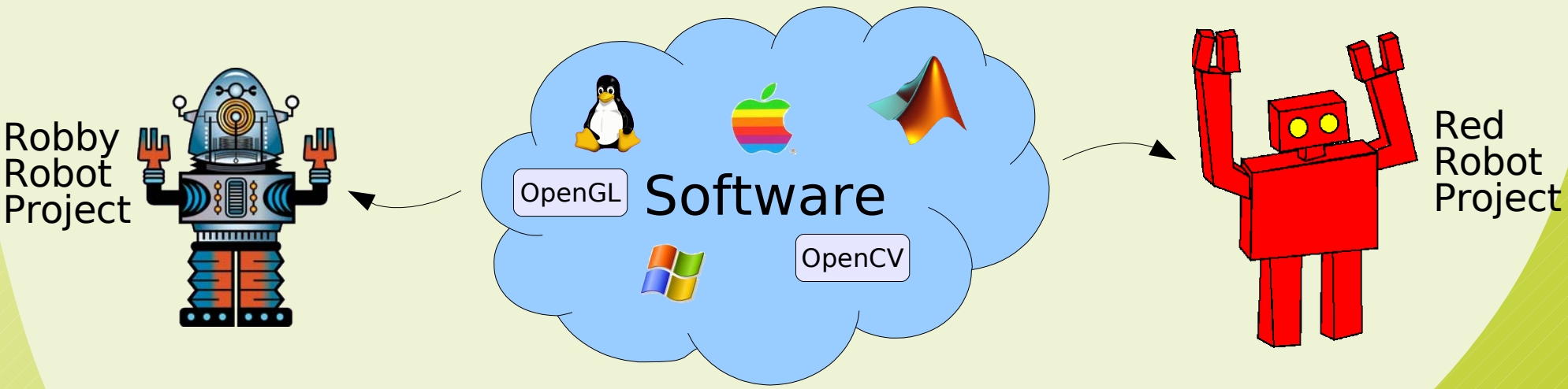
Our Literature

- As a research community, we both read and produce papers, building on each others' work



Our Software

- We also both acquire and produce software
- Our software tends to die with our projects
- Sad! Software collaboration speeds things up



RobotCub

Hardware Diversity

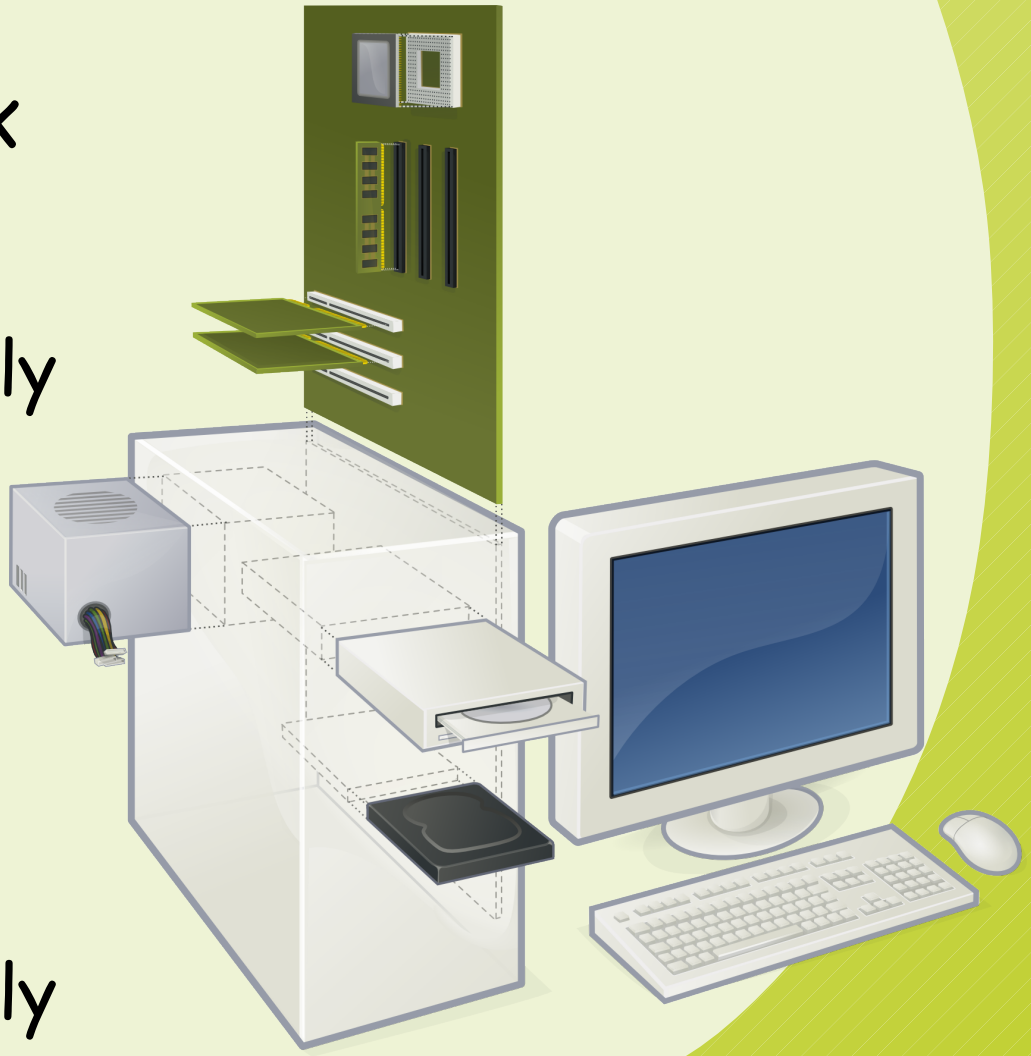
- Research groups that all use a specific robot (Khepera, Pioneer, AIBO, ...) often form a natural software community
  - But each alone is a small subset of robotics
- Groups developing new robots face obstacles
  - Differences in sensors, actuators, bodies...
  - Differences in processors, operating systems, libraries, frameworks, languages, compilers...
  - Big barriers to software collaboration

Cogsys
Cognitive Systems

RobotCub

the PC

- Constant hardware flux
  - Parts change rapidly
  - Interfaces change slowly
- Lots of software grew and evolved alongside the changing hardware
  - Parts change rapidly
  - Interfaces change slowly
  - "Modularity" is rewarded



Cogsys
Cognitive Systems

RobotCub

Modularity Helps

- The way parts interact can last longer than the parts themselves
- E.g. an eternal broom
  - replace broom head
  - replace broom handle

Cogsys
Cognitive Systems

RobotCub

the Ship of Theseus

- Long-lived software is like the Ship of Theseus
  - The mast gets replaced
  - The planks get replaced
  - Over time, everything may get replaced
- In philosophy, this is a "paradox of identity"
- For us, it's just our job

Cogsys
Cognitive Systems

RobotCub

Modularity

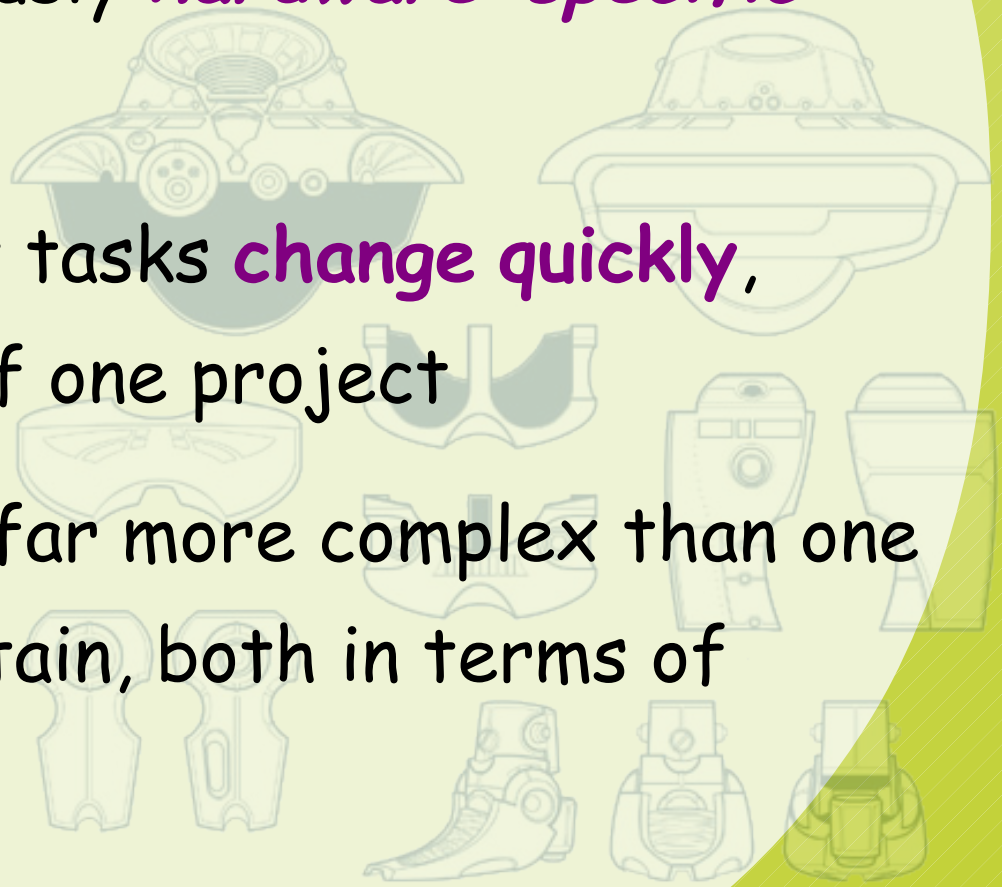
- The opposite of a **modular** system is a **coupled** one.
- In a "coupled" system, changes in one part trigger changes in another.
  - Coupling leads to **complexity**
  - Complexity leads to **confusion**
  - Confusion leads to **suffering**
- This is the path to the Dark Side

Cogsys
Cognitive Systems

RobotCub

Why Modularity for Robots?

- Robot software is notoriously ***hardware-specific*** and ***task-specific***
- Both hardware and target tasks **change quickly**, even within the lifetime of one project
- Our humanoid robots are far more complex than one person can build and maintain, both in terms of hardware and software
- They need to be **modular**



Cogsys
Cognitive Systems

RobotCub

Modular Approaches

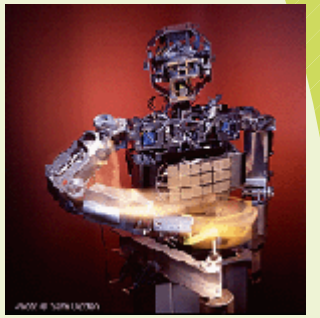
- Modular approaches to robotics:
  - **Player/Stage** (mobile robotics)
    - Robot control (Khepera, Pioneer), simulator
  - **Orocos** (industrial robotics)
    - Real-time control, kinematics library, other libs
  - **YARP** (humanoid robotics)

SOURCE: Chad Jenkins, June 11, 2005, Workshop Introduction
Robotics 2005 Workshop on Modular Foundations for Control and Perception

Cogsys
Cognitive Systems

Yet Another Robot Platform

- YARP is an open-source software library for humanoid robotics



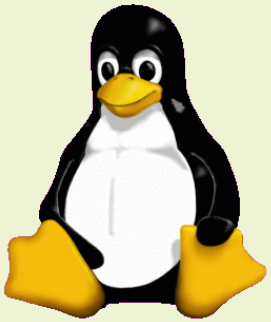
- History
  - An MIT / LIRA-Lab collaboration
  - Born on Kismet, grew on COG
  - With a major overhaul, now used by RobotCub consortium
  - Exists as an independent open source project
  - C++ source code

IDE/OS Portability (CMake)

CMake
Cross-platform Make

Project description (.txt)

LINUX: Makefiles, Kdevelop files, ...



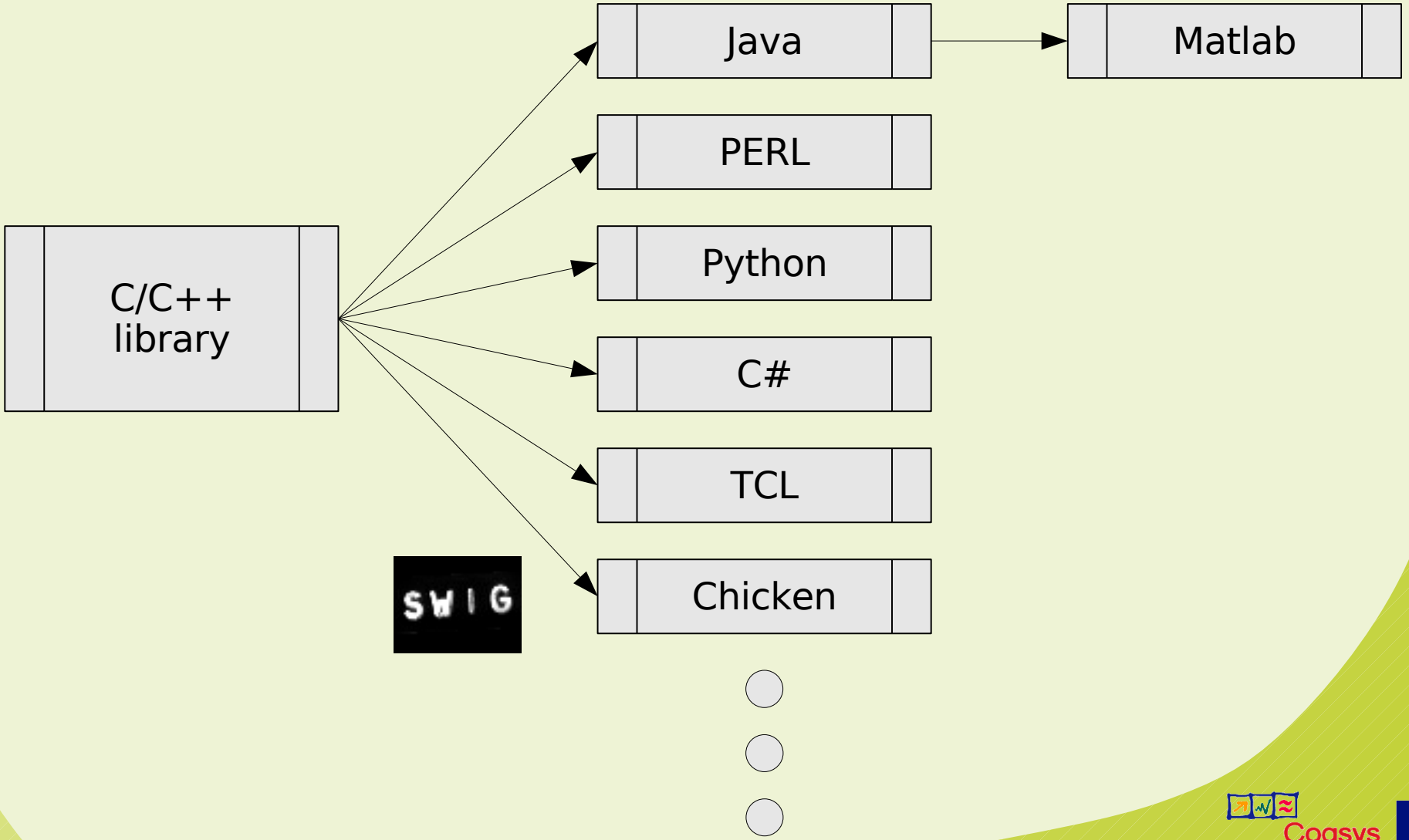
WINDOWS: MSVC files, Borland files, ...

OSX: Makefiles, Xcode files, ...

Cogsys
Cognitive Systems

RobotCub

Language Portability (SWIG)



Cogsys
Cognitive Systems

RobotCub

What is YARP for?

- Factor out **details of data flow between programs** from program source code
  - Data flow is very specific to robot platform, experimental setup, network layout, communication protocol, etc.
  - Useful to keep "algorithm" and "plumbing" separate
- Factor out **details of devices used by programs** from program source code
  - The devices can then be replaced over time by comparable alternatives; code can be used in other systems

Cogsys
Cognitive Systems

RobotCub

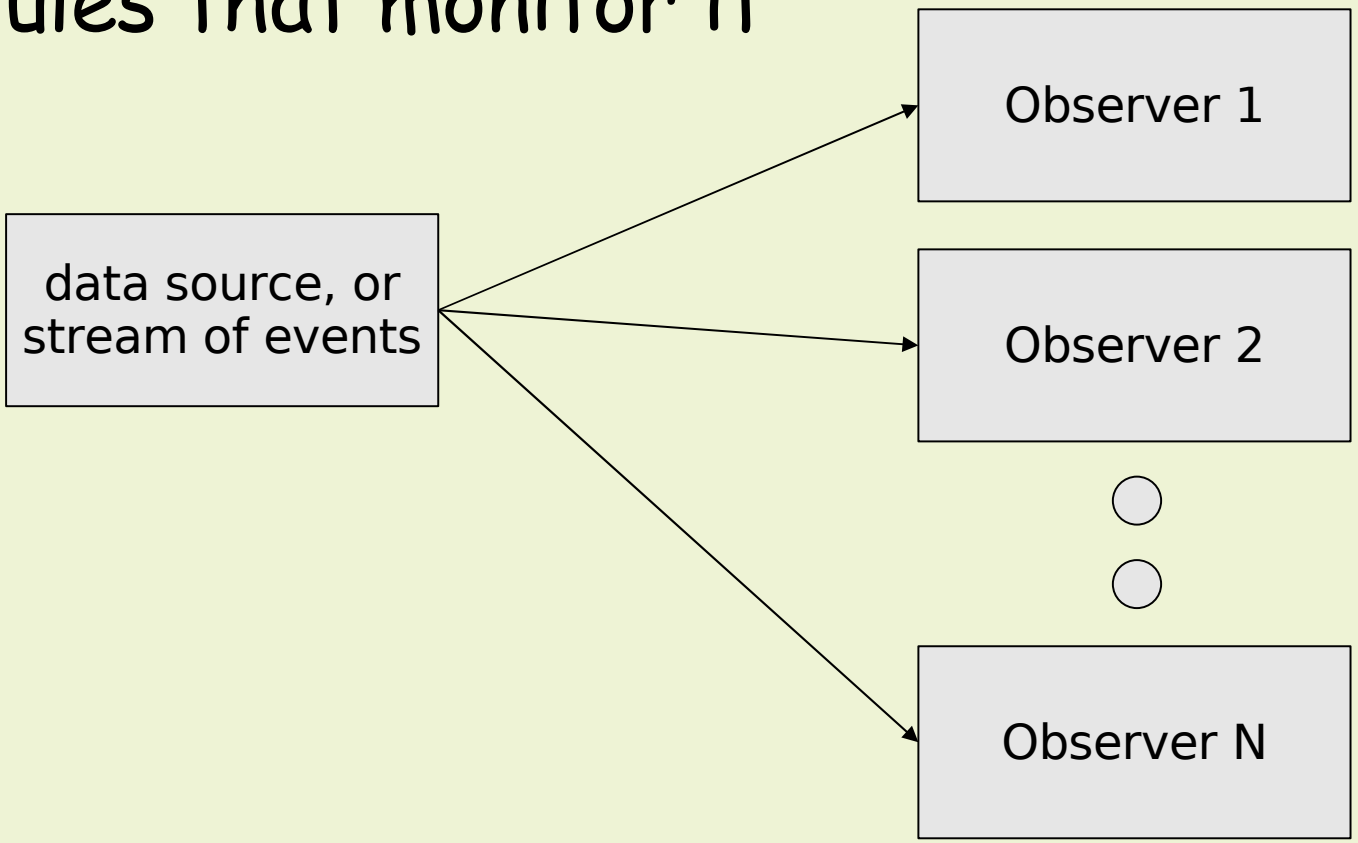
What is YARP for?

- Factor out **details of data flow between programs** from program source code
  - Data flow is very specific to robot platform, experimental setup, network layout, communication protocol, etc.
  - Useful to keep "algorithm" and "plumbing" separate
- Factor out **details of devices used by programs** from program source code
  - The devices can then be replaced over time by comparable alternatives; code can be used in other systems

Cogsys
Cognitive Systems

the Observer pattern

- Data source knows nothing about identity of modules that monitor it



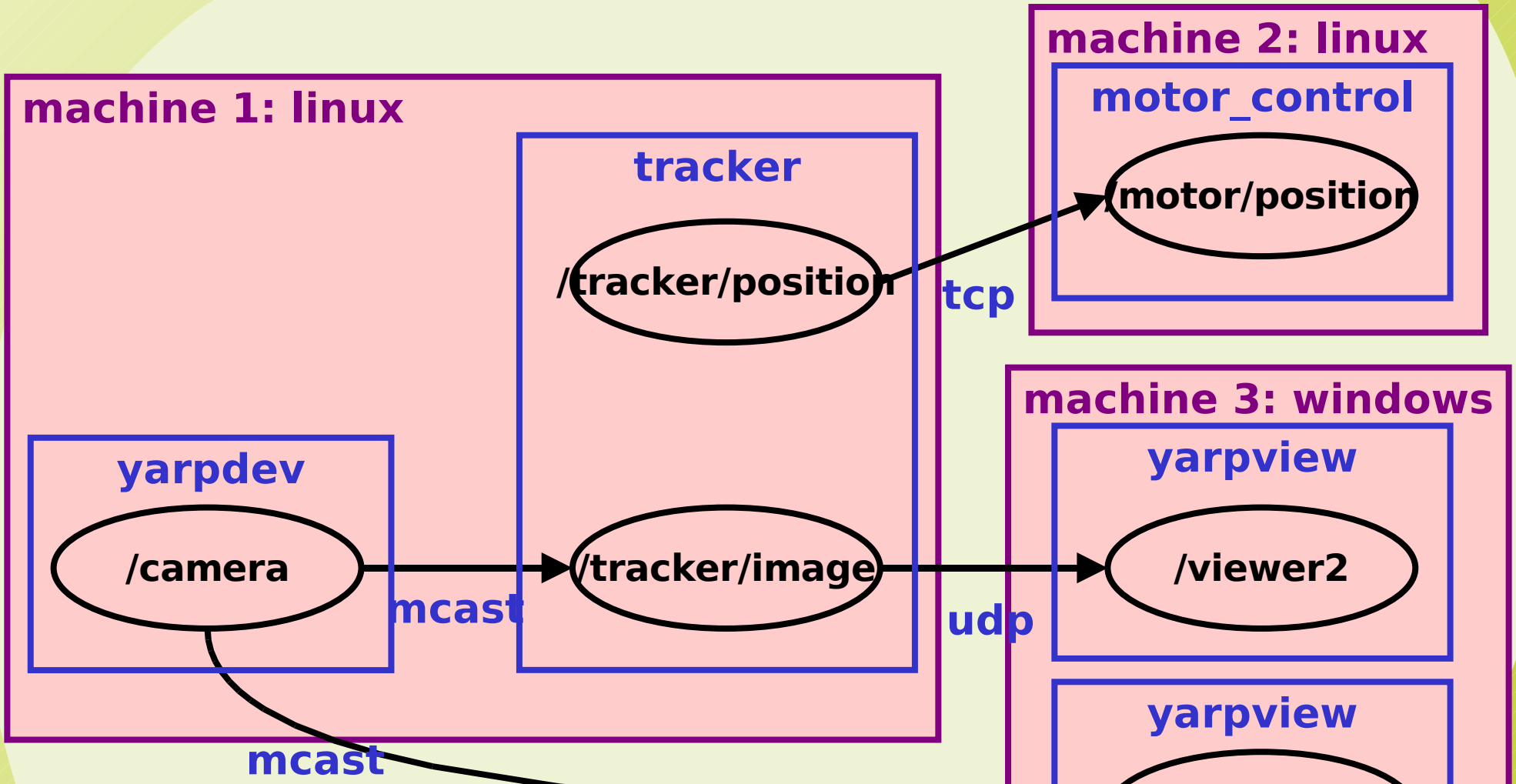
RobotCub

YARP Ports

- We follow the **Observer** design pattern.
- Special "Port" objects deliver data to:
  - Any number of observers (other "Port"s) ...
  - ... in any number of processes ...
  - ... distributed across any number of computers/OSes ...
  - using any of several underlying communication protocols with different technical advantages, streaming or RPC
- This is called the YARP Network

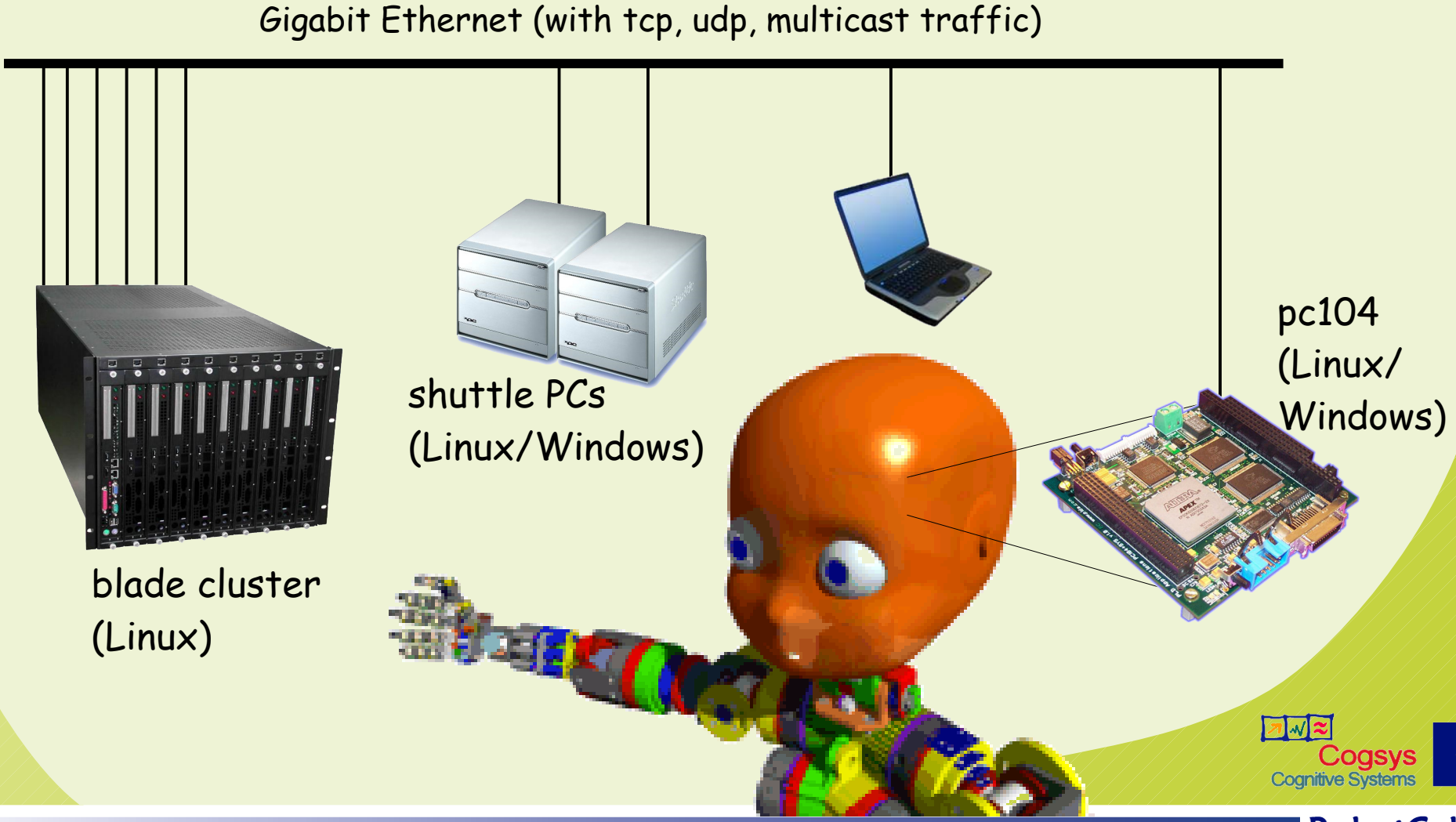
Cogsys
Cognitive Systems

Typical YARP Network



- Connections can use different protocols
- Ports belong to processes
- Processes can be on different machines/OS

Physical Network Example: RobotCub



Simple example: a countdown

```
#include <yarp/os/all.h>
#include <stdio.h>
using namespace yarp::os;

int main(int argc, char *argv[]) {
    if (argc!=2) return 1;
    Network::init();

    BufferedPort<Bottle> out;
    out.open(argv[1]);

    for (int i=10; i>=0; i--) {
        printf("at %d\n", i);
        Bottle& msg = out.prepare();
        msg.clear();
        msg.addString("countdown");
        msg.addInt(i);
        out.write();
        Time::delay(1);
    }

    Network::fini();
    return 0;
}
```

make_count.cpp

```
#include <yarp/os/all.h>
#include <stdio.h>
using namespace yarp::os;

int main(int argc, char *argv[]) {
    if (argc!=2) return 1;
    Network::init();

    BufferedPort<Bottle> in;
    in.open(argv[1]);

    int count = 1;
    while (count>0) {
        Bottle *msg = in.read();
        count = msg->get(1).asInt();
        printf("at %d\n", count);
    }

    Network::fini();
    return 0;
}
```

view_count.cpp

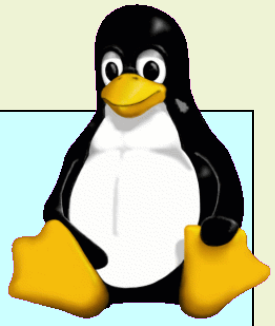
RobotCub

Simple example: a countdown

- make_count generates a countdown message
  - countdown 10
  - countdown 9
  - ...
- view_count views a countdown message
- Neither program mentions the other
- Neither program dictates how the message will be transmitted

Cogsys
Cognitive Systems

Simple example: a countdown



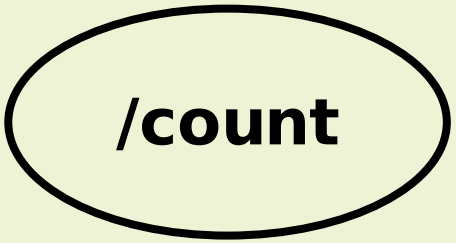
```
$ ./make_count /count
yarp: Port /count listening
at 10
at 9
...
```

```
C:\> view_count /view
yarp: Port /view listening
```

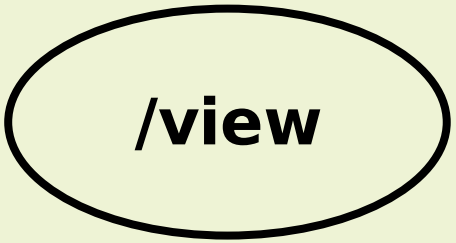
```
$
```

YARP Network

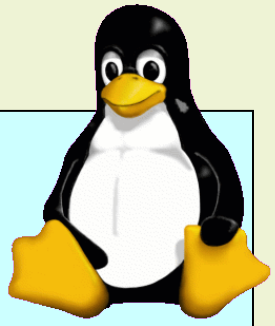
`make_count /count`



`view_count /view`



Simple example: a countdown

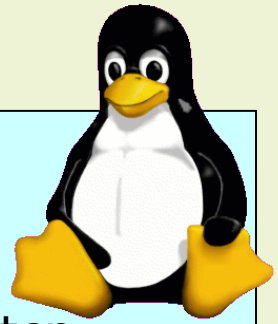


```
$ ./make_count /count
yarp: Port /count listening
at 10
at 9
...
```

```
C:\> view_count /view
yarp: Port /view listening
```

```
$
```

Simple example: a countdown



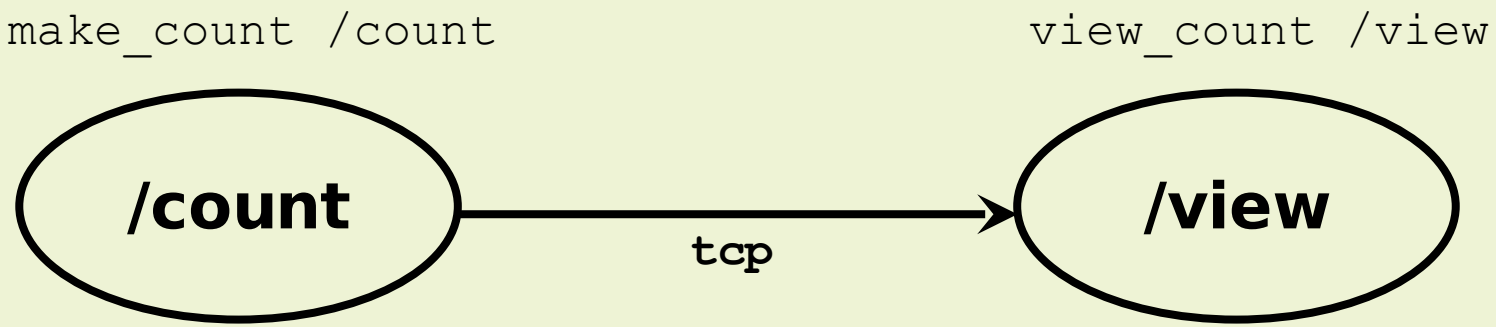
```
$ ./make_count /count
yarp: Port /count listening
at 10
at 9
yarp: output /count to /view, tcp
at 8
at 7
..
```

```
C:\> view_count /view
yarp: Port /view listening
yarp: Input /count to /view, tcp
at 8
at 7
...
```

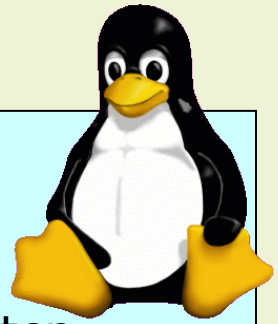
```
$ yarp connect /count /view
Added output: /count to /view, t

$
```

YARP Network



Simple example: a countdown



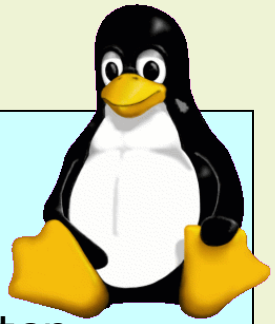
```
$ ./make_count /count
yarp: Port /count listening
at 10
at 9
yarp: output /count to /view, tcp
at 8
at 7
..
```

```
C:\> view_count /view
yarp: Port /view listening
yarp: Input /count to /view, tcp
at 8
at 7
...
```

```
$ yarp connect /count /view
Added output: /count to /view, t

$
```

Simple example: a countdown



```
$ ./make_count /count
yarp: Port /count listening
at 10
at 9
yarp: output /count to /view, tcp
at 8
at 7
yarp: output /count to /view2, udp
at 6
...
```

```
C:\> view_count /view
yarp: Port /view listening
yarp: Input /count to /view, tcp
at 8
at 7
at 6
at 5
...
```

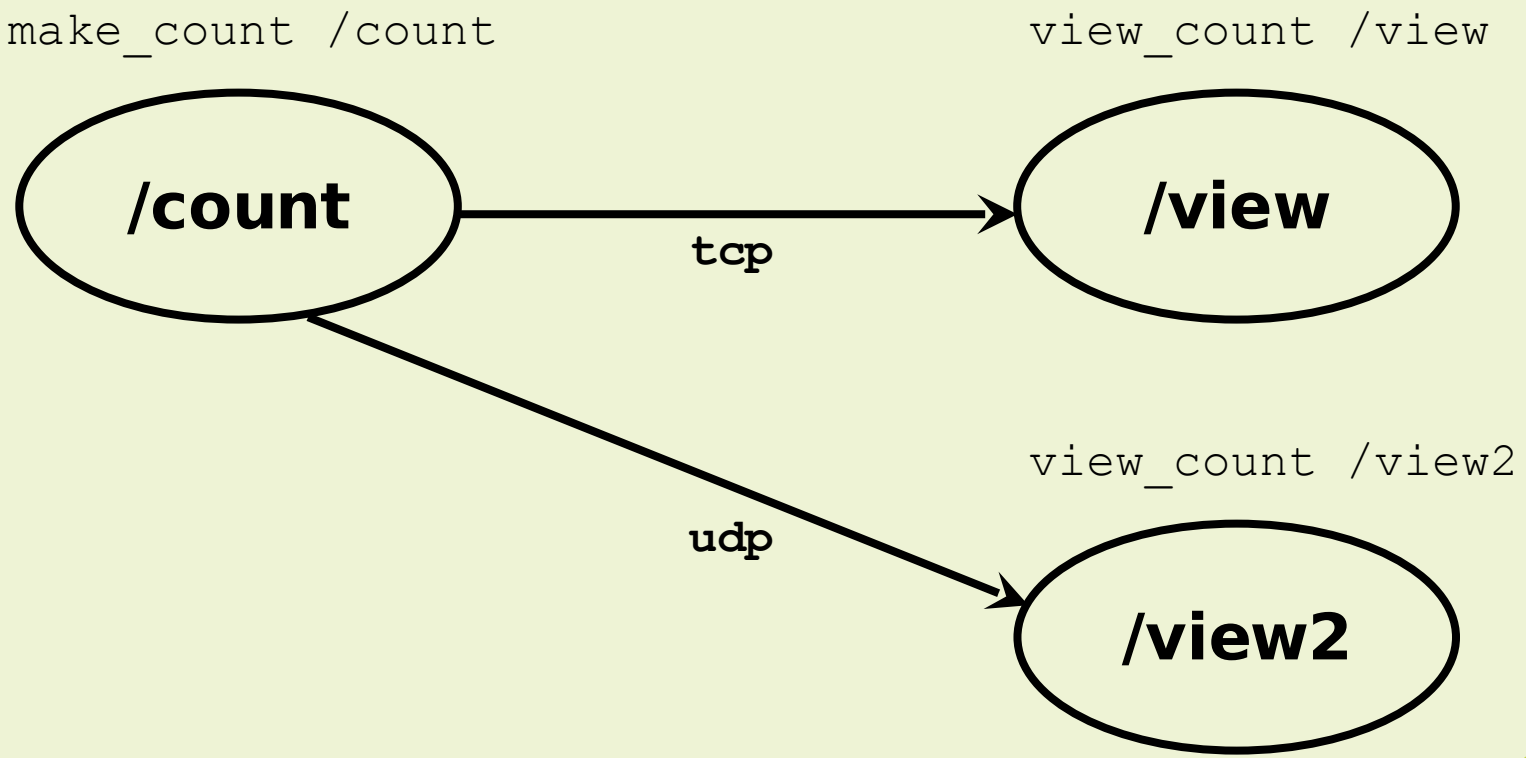
```
$ yarp connect /count /view
Added output: /count to /view, t

$ yarp connect /count /view2 udp
Added output: /count to /view, udp
```

```
$ view_count /view2
yarp: Port /view2 listening
yarp: Input /count to /view2, udp
at 6
at 5
...
```

Cogsys
Cognitive Systems

YARP Network



RobotCub

Why is all this useful?

- We've separated out most of the plumbing
- We get to change it dynamically (handy)
- More importantly, we have better modularity
  - Programs can be moved around as load and OS/device/library dependencies dictate
  - Fundamental protocol for communication can be changed without affecting programs
  - Better chance that your code can be used by others (even just within your group)

Cogsys
Cognitive Systems

RobotCub

What is YARP for?

- Factor out **details of data flow between programs** from program source code
  - Data flow is very specific to robot platform, experimental setup, network layout, etc.
  - Useful to keep "algorithm" and "plumbing" separate
- Factor out **details of devices used by programs** from program source code
  - The devices can then be replaced over time by comparable alternatives; code can be used in other systems

Cogsys
Cognitive Systems

RobotCub

What is YARP for?

- Factor out **details of data flow between programs** from program source code
  - Data flow is very specific to robot platform, experimental setup, network layout, etc.
  - Useful to keep "algorithm" and "plumbing" separate
- Factor out **details of devices used by programs** from program source code
  - The devices can then be replaced over time by comparable alternatives; code can be used in other systems

Cogsys
Cognitive Systems

YARP Devices

- There are three separate concerns related to devices in YARP:
  - Implementing **specific drivers** for particular devices
  - Defining interfaces for **device families**
  - Implementing **network wrappers** for interfaces
- Basic idea: if you view your devices through well thought out interfaces, the impact of device change can be minimized.

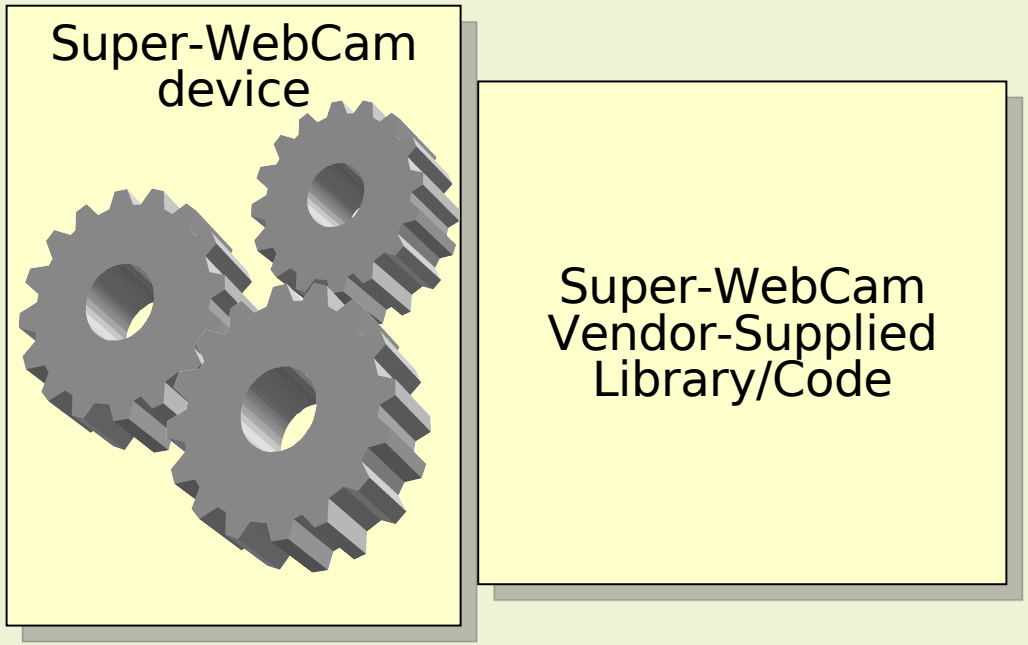
RobotCub

A Light Touch

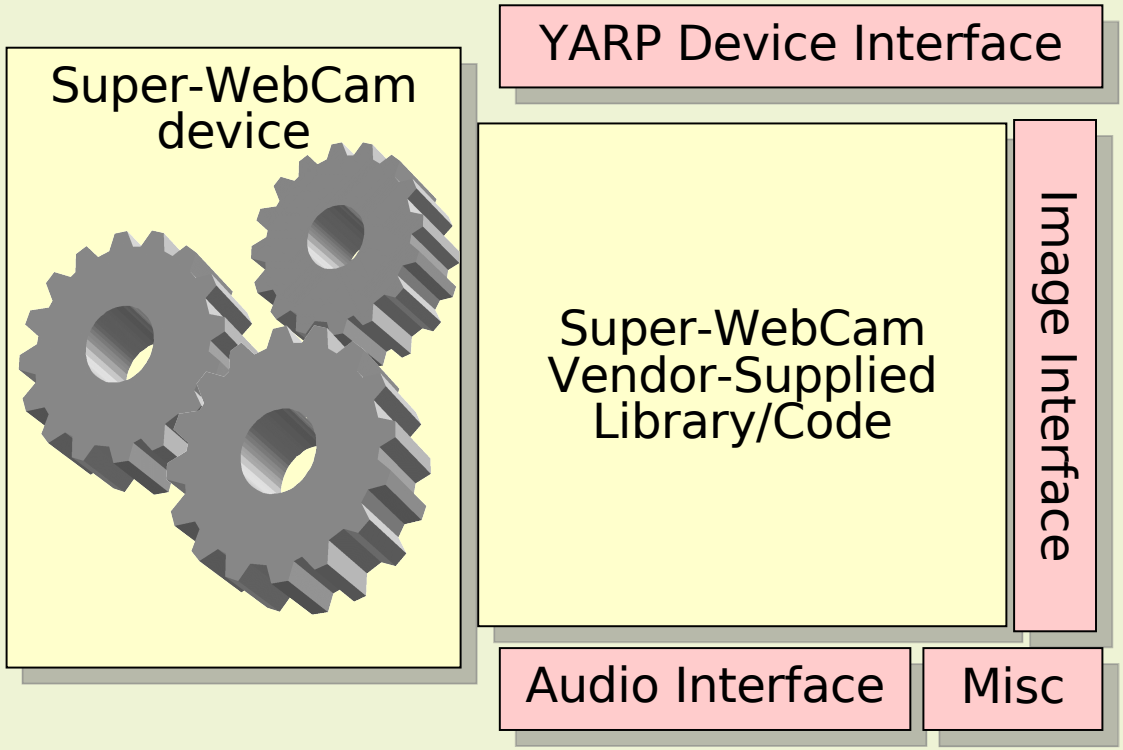
- New devices come out all the time – needs to be easy to connect them to existing code
- YARP needs a minimal "wrapper" class to match vendor-supplied library with relevant interfaces that capture common capabilities
- YARP encourages separating configuration from source code – separating the "plumbing"
- Devices and communications remain distinct concerns

Cogsys
Cognitive Systems

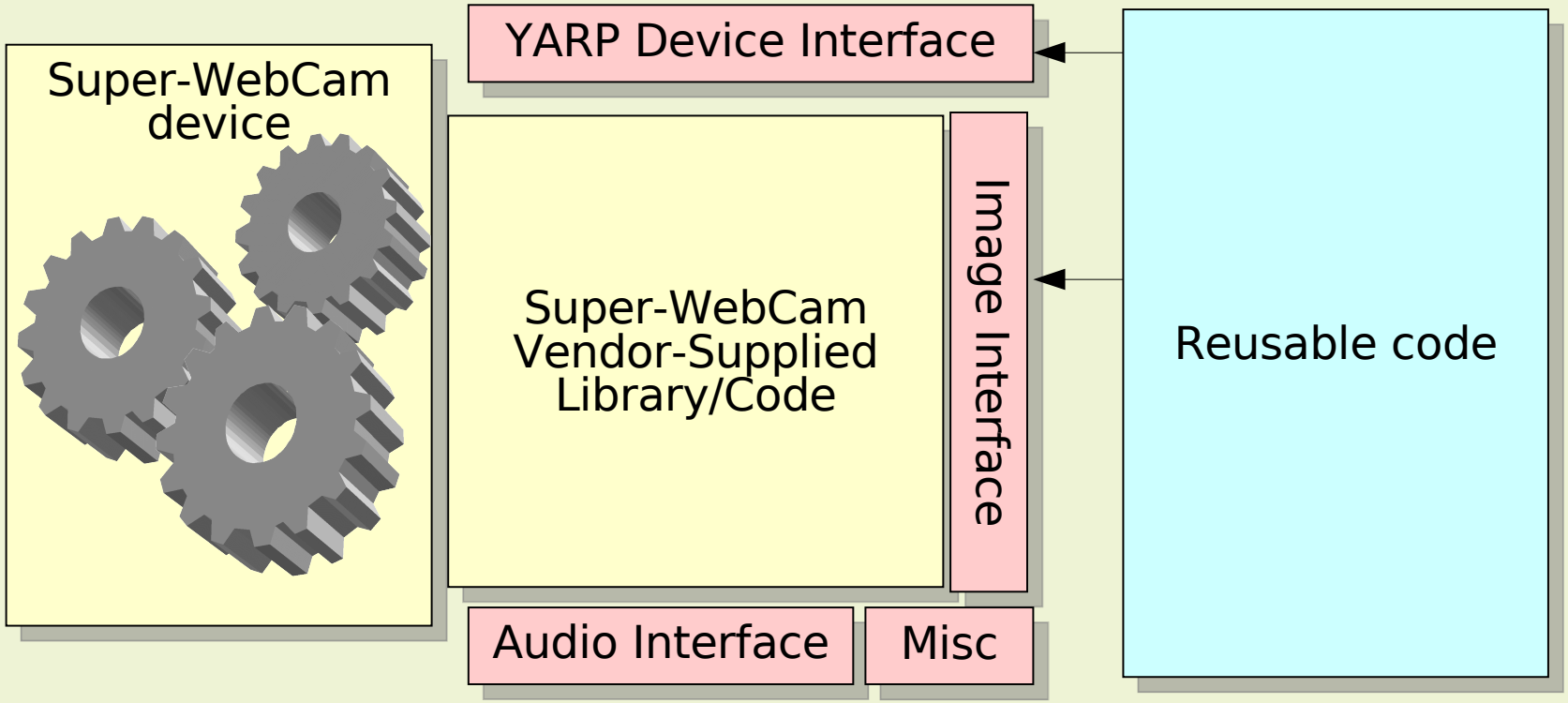
Example: New WebCam



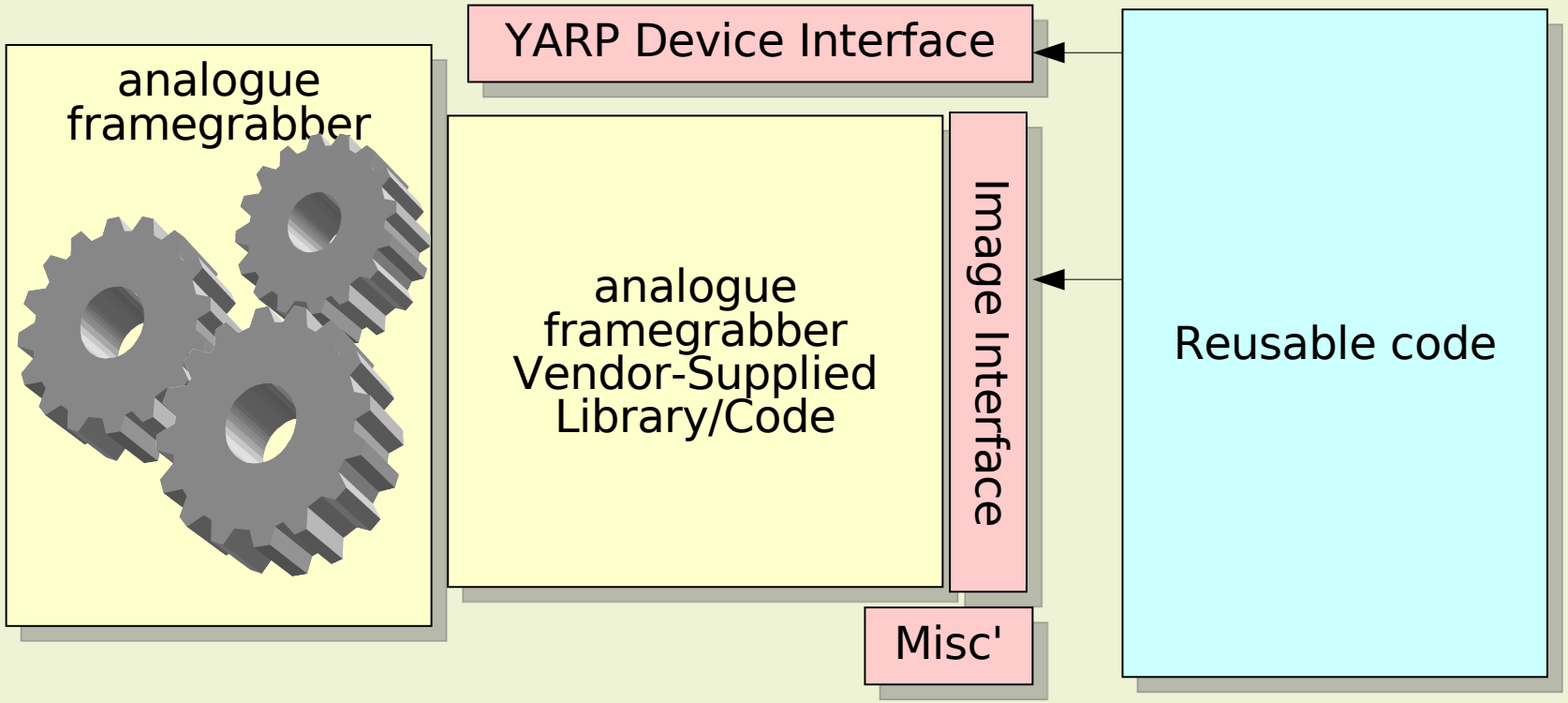
Example: New WebCam



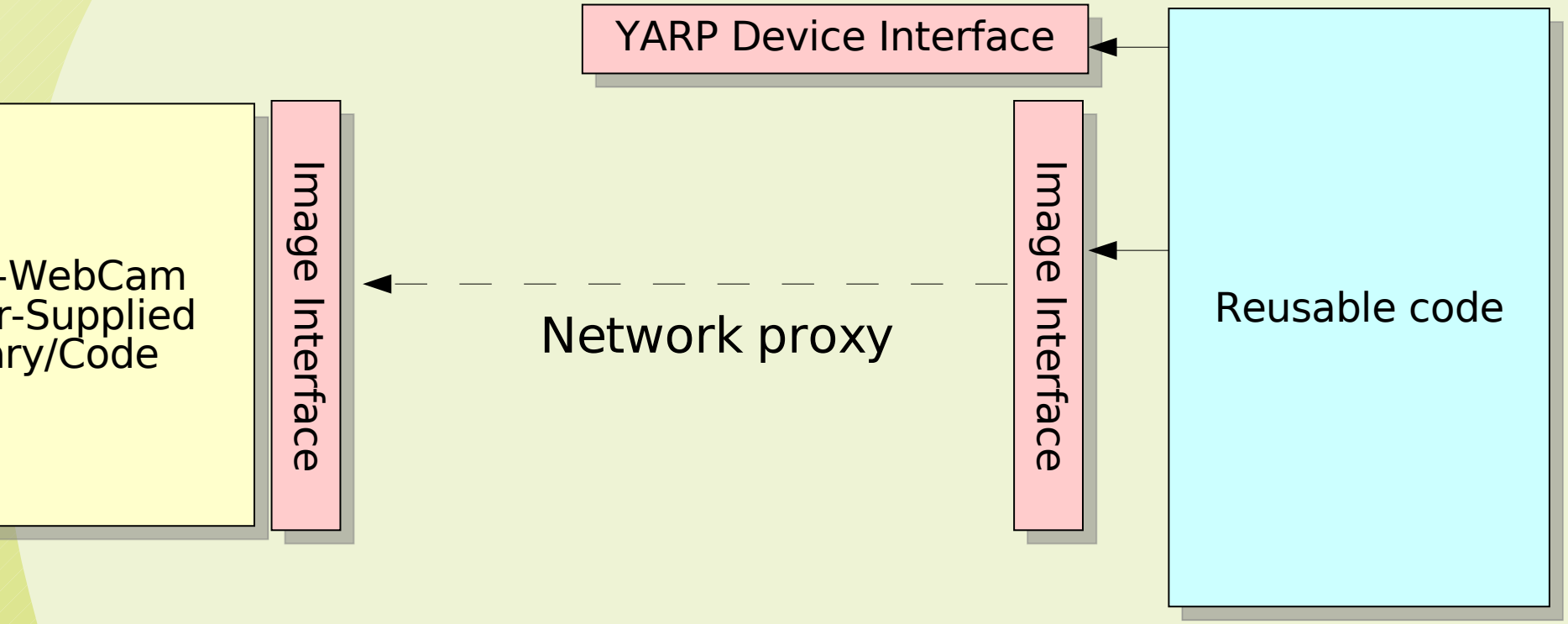
Example: New WebCam



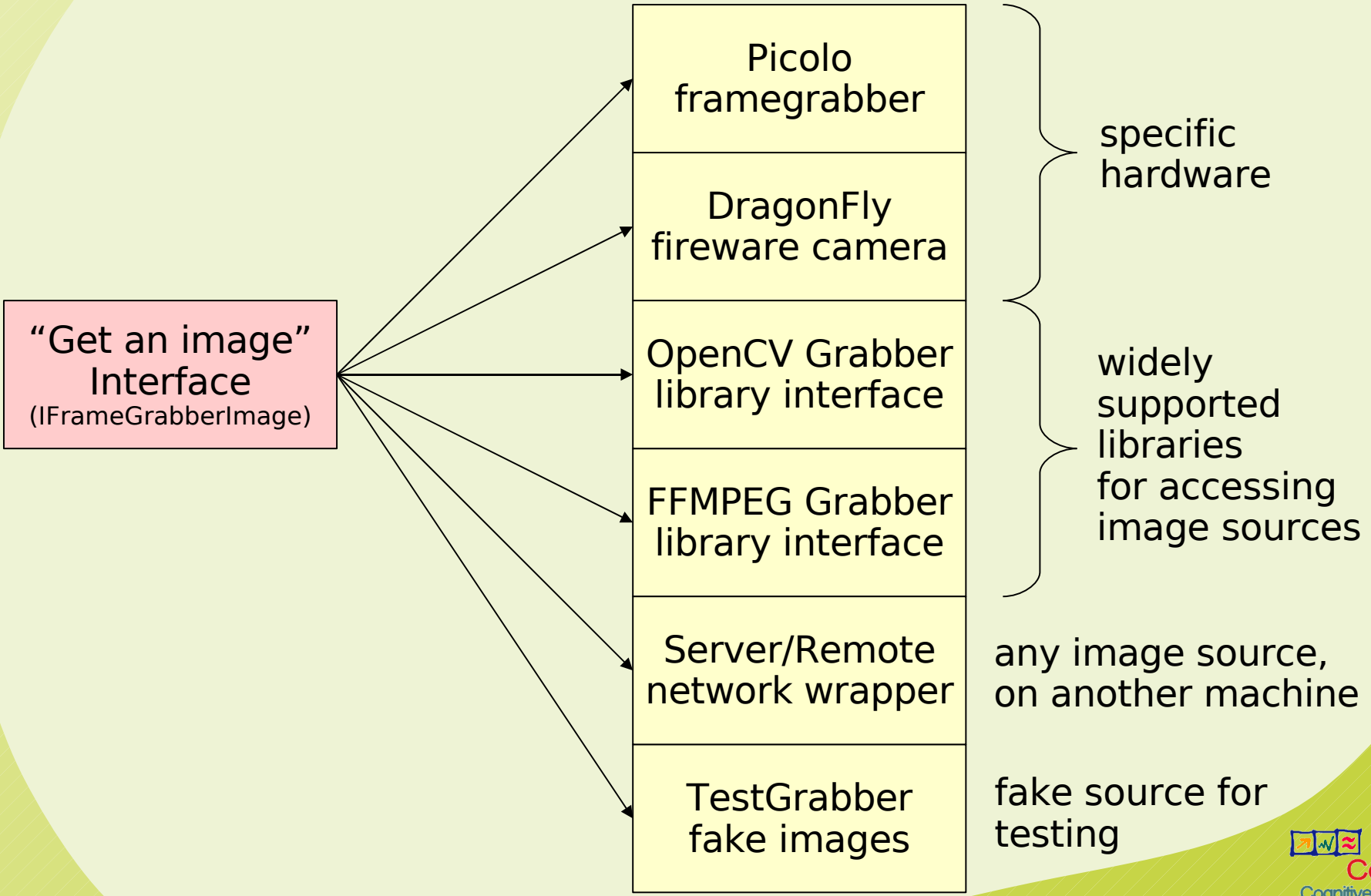
Example: Old Framegrabber



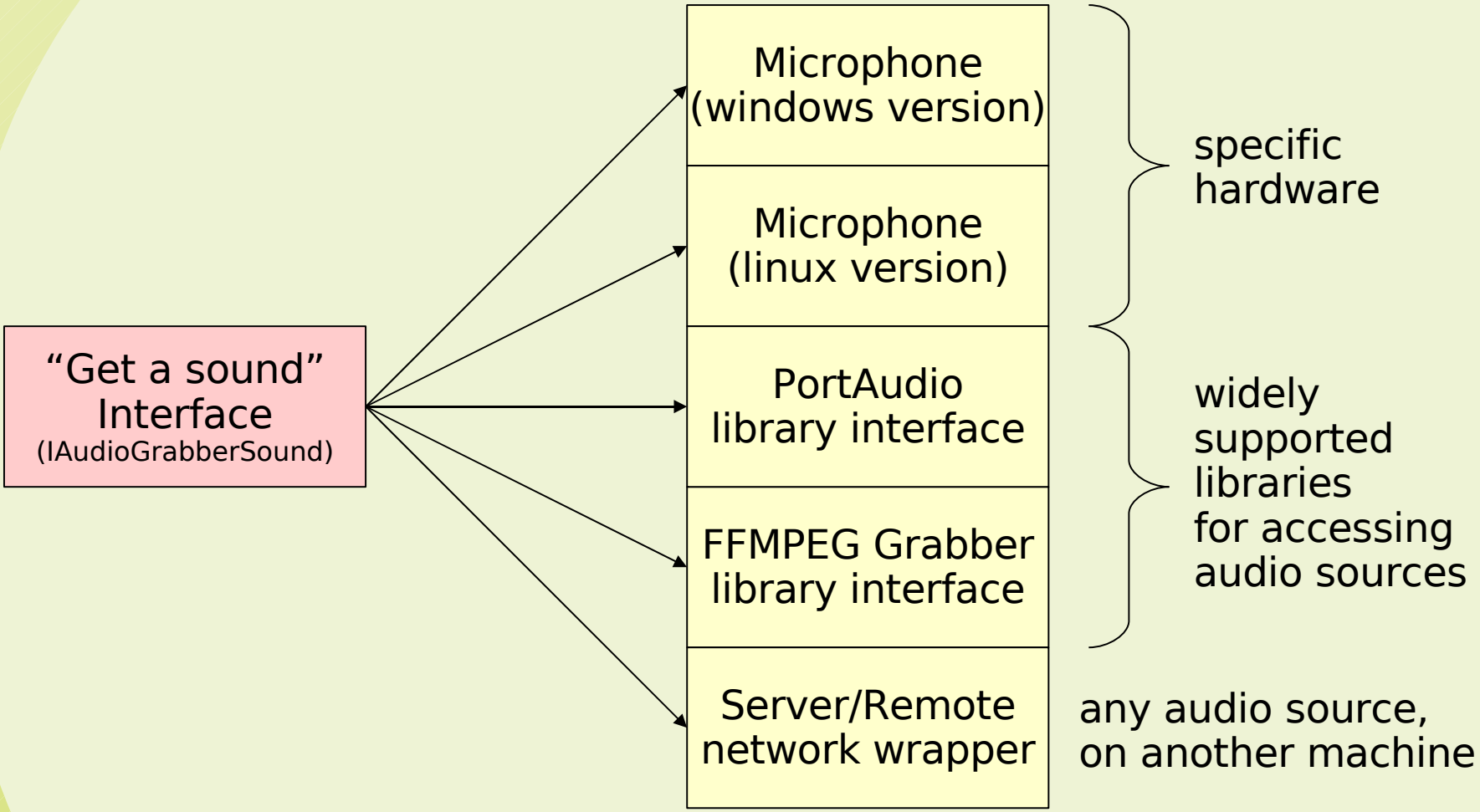
Example: Networking



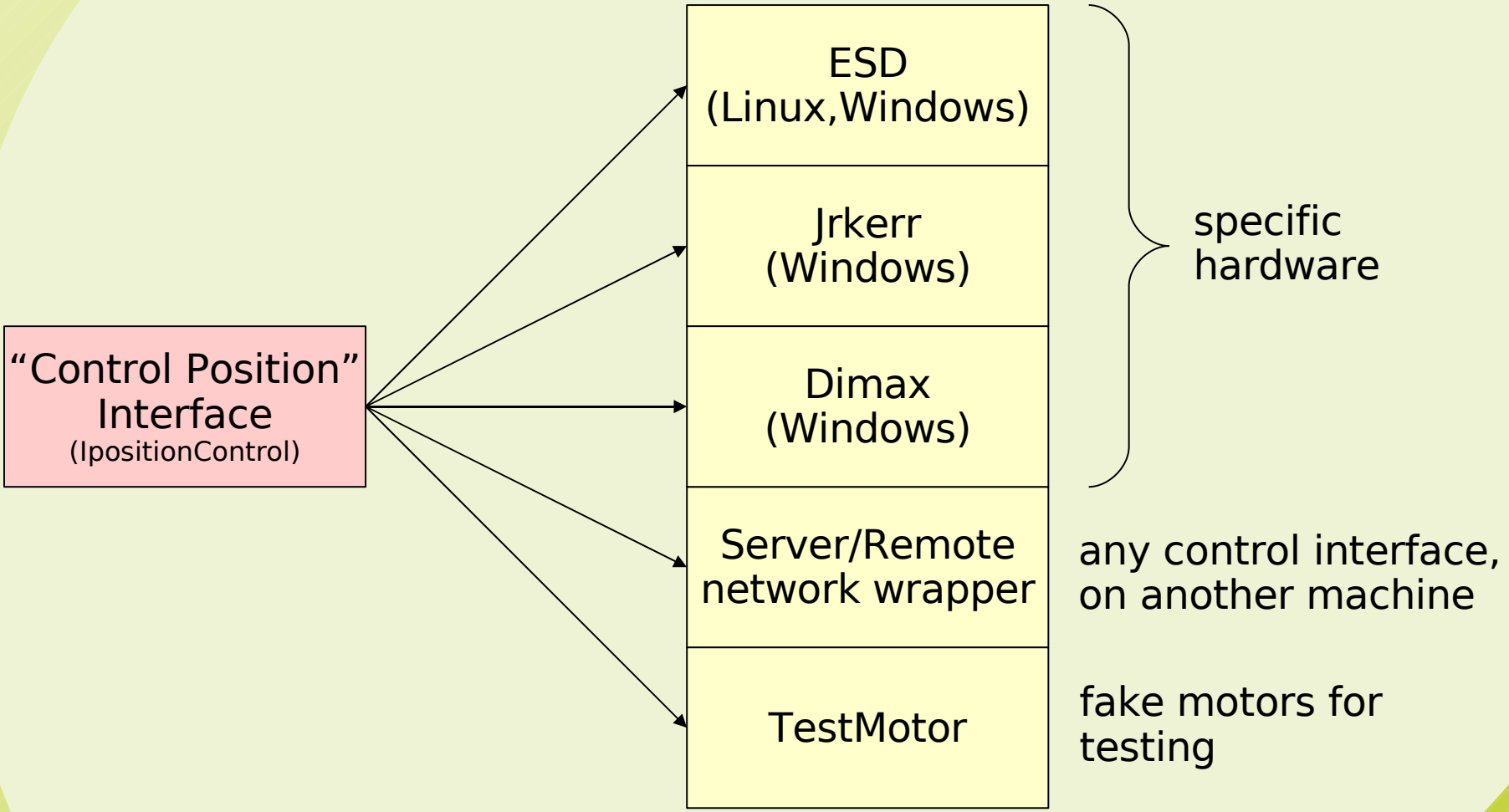
Family of Image Sources



Family of Audio Sources



Family of Motor Controllers



RobotCub

Why is this useful?

- Allows collaboration between groups whose robots have different devices
- Makes device changes less painful
- Devices and communications are orthogonal features
  - Can switch from remote use of device to local use and vice versa without pain
  - Local use can be very efficient, just an extra virtual method call

Cogsys
Cognitive Systems

RobotCub

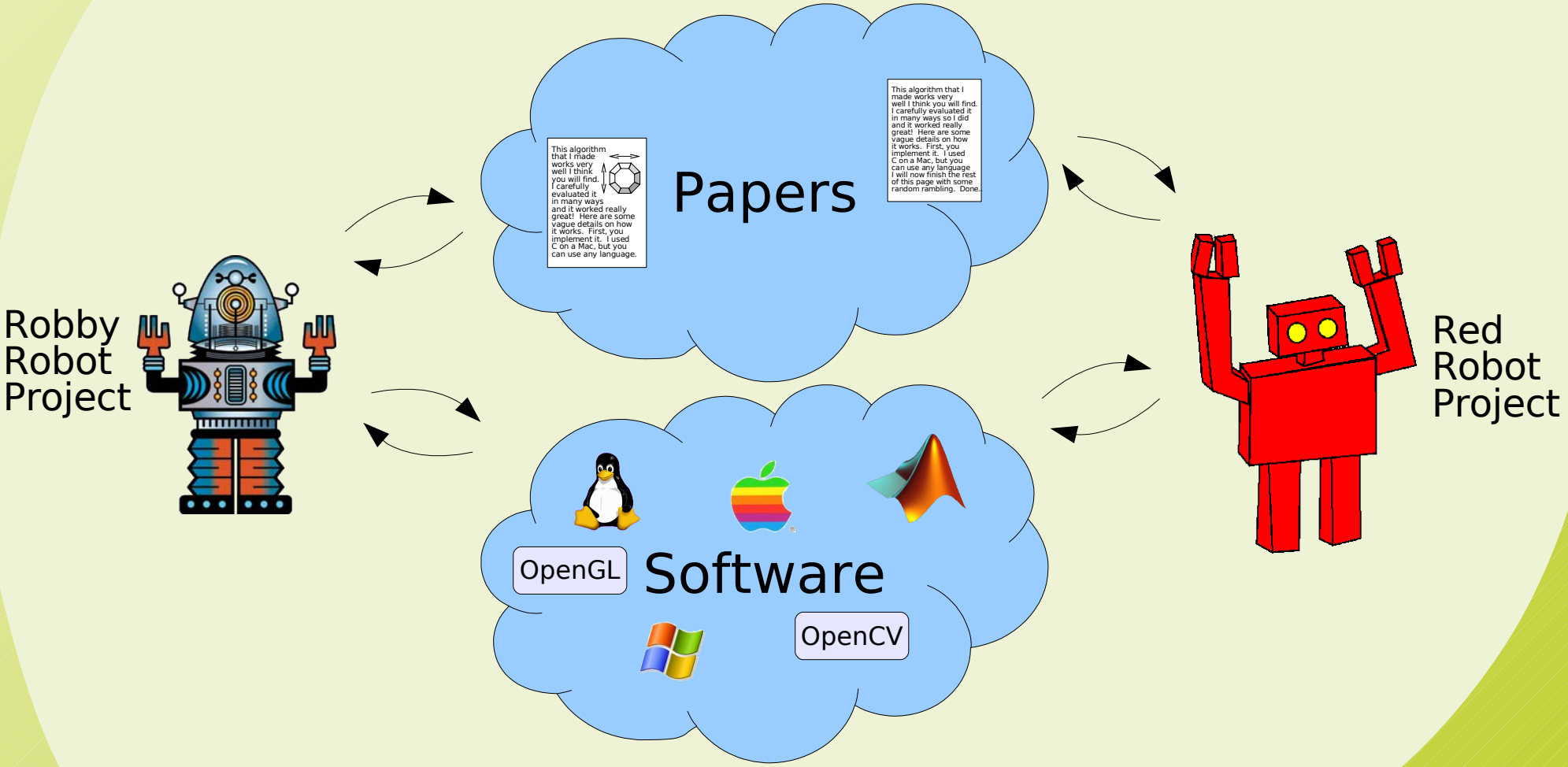
Conclusions

- The literature of a research community both expresses its ideas, and aids in their evolution
  - Published ideas are read, evaluated, and built upon
  - Useful advances get published
- Publication of software can speed progress
  - Facilitates evaluating and comparing approaches
  - Brings new research topics into reach

Cogsys
Cognitive Systems

RobotCub

Publish or Perish!



Cogsys
Cognitive Systems