A (very) simple example: read data to/from a port

[on terminal 1] yarpserver
[on terminal 2] yarp read /read
[on terminal 3] yarp write /write /read


```
$ yarp write /write /read
Port /write listening at tcp://127.0.0.1:10012
yarp: Sending output from /write to /read using tcp
Added output connection from "/write" to "/read"
hello yarp
1 2 3
```

```
$ yarp read /read
Port /read listening at tcp://127.0.0.1:10002
yarp: Receiving input from /write to /read using tcp
hello yarp
1 2 3
```

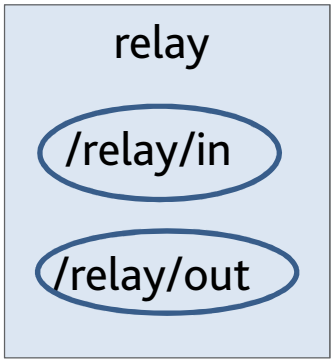
How do we get this?

Let's now to write a simple "relay" executable which takes whatever comes from a port and forwards it to another one.

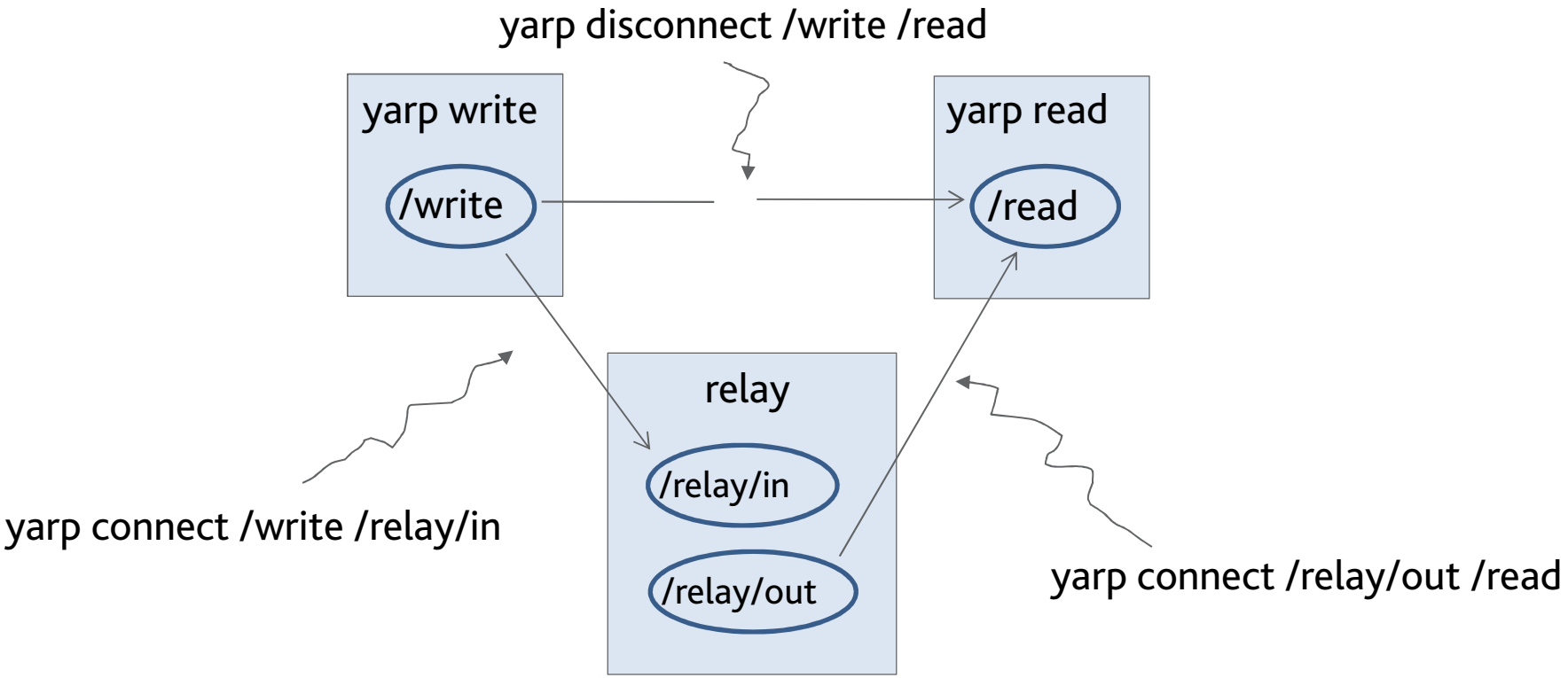
```
int main(int argc, char *argv) {
    Network yarp;
    Port inPort;
    inPort.open("/relay/in");

    Port outPort;
    outPort.open("/relay/out");


    while (true) {
        cout << "waiting for input" << endl;
        Bottle input,output;
        inPort.read(input);
        output=input;
        cout << "writing " << output.toString().c_str() << endl;
        outPort.write(output);
    }
    return 0;
}
```

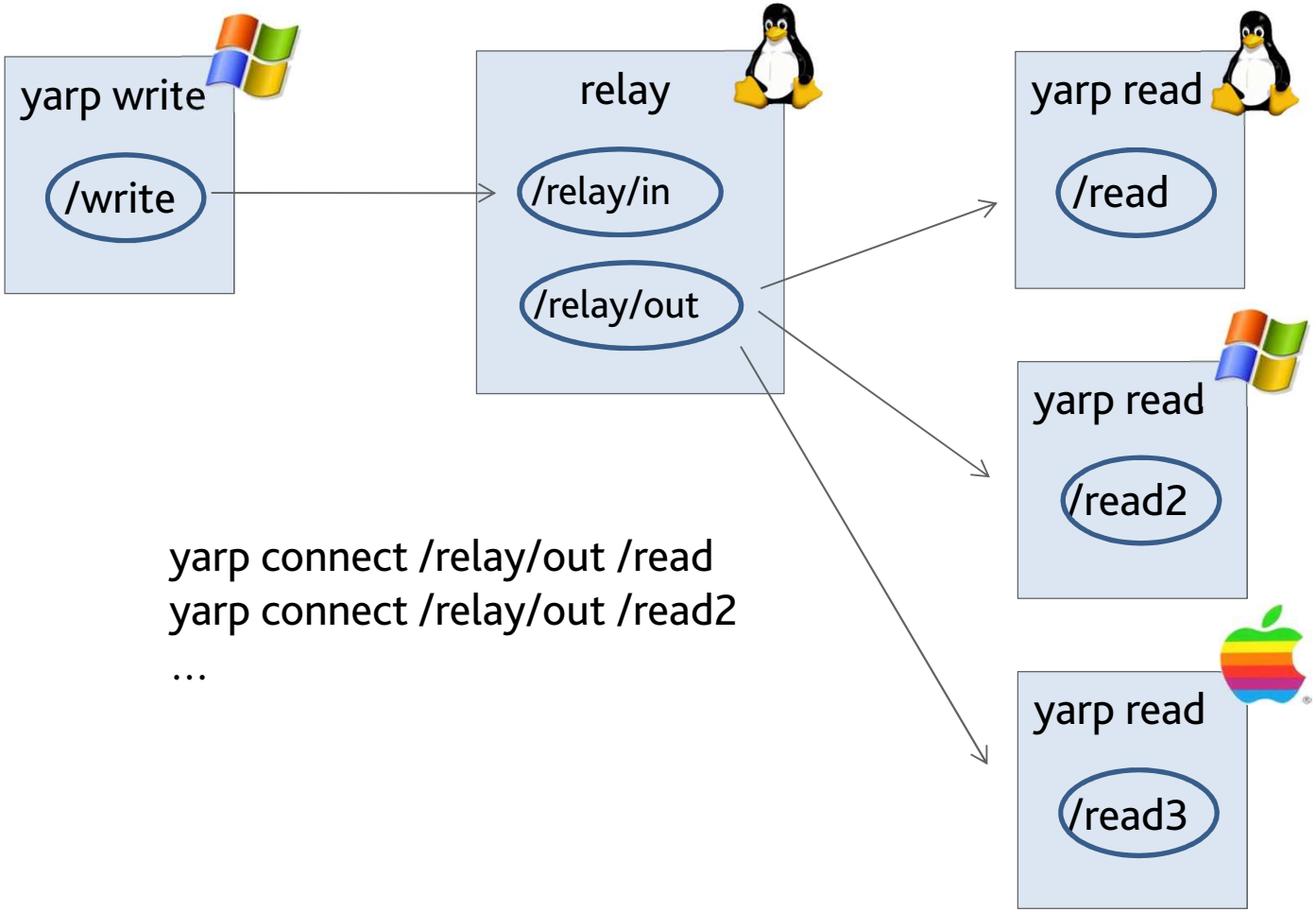


Connect the new module to our network



how the network grows

It is easy to add, for example, another reader…
Processes can run on different machines, with different OS



yarp connect /relay/out /read
yarp connect /relay/out /read2
…

yarp name list

BufferedPort

- In the previous example timing between ports is coupled:
  - The reader waits until data arrives to the port
  - The writer waits until data is transmitted
- Buffered ports allow decoupling time:
  - non blocking read
  - non blocking write
- May loose messages

- Read:

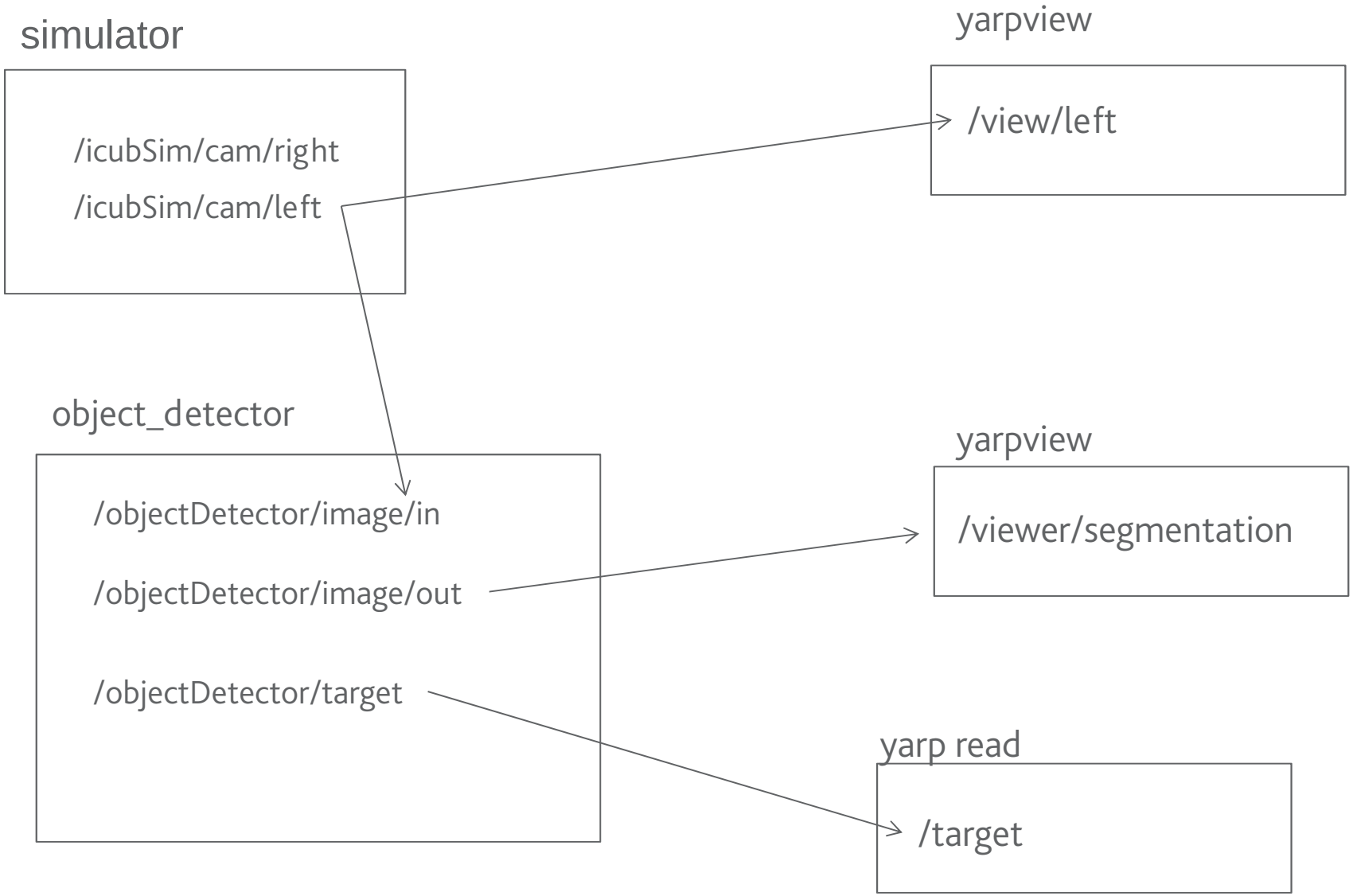
```
BufferedPort<Bottle> p;          // Create a port.
p.open("/in");                   // Give it a name on the network.
while (true) {
      Bottle *b = p.read();      // Read/wait for until data arrives. ...
      // Do something with data in *b
 }
```

- Write:

```
BufferedPort<Bottle> p;          // Create a port.
p.open("/out");                  // Give it a name on the network.
while (true) {
      Bottle& b = p.prepare();   // Get a place to store things. ...
      // Generate data.
      p.write();                 // Send the data.
}
```

- Polling: when you do not want to wait for input data:

```
BufferedPort<Bottle> p;
...
Bottle *b = p.read(false);
if (b!=NULL) {
  // data received in *b
}
```


simulator
/icubSim/cam/right
/icubSim/cam/left
yarpview
/view/left
object_detector
/objectDetector/image/in
/objectDetector/image/out
/objectDetector/target
yarpview
/viewer/segmentation
yarp read
/target


iit