ISTITUTO ITALIANO
DI TECNOLOGIA

Component based software development & software middleware

Lorenzo Natale

iCub Facility

Istituto Italiano di Tecnologia, Genova, Italy

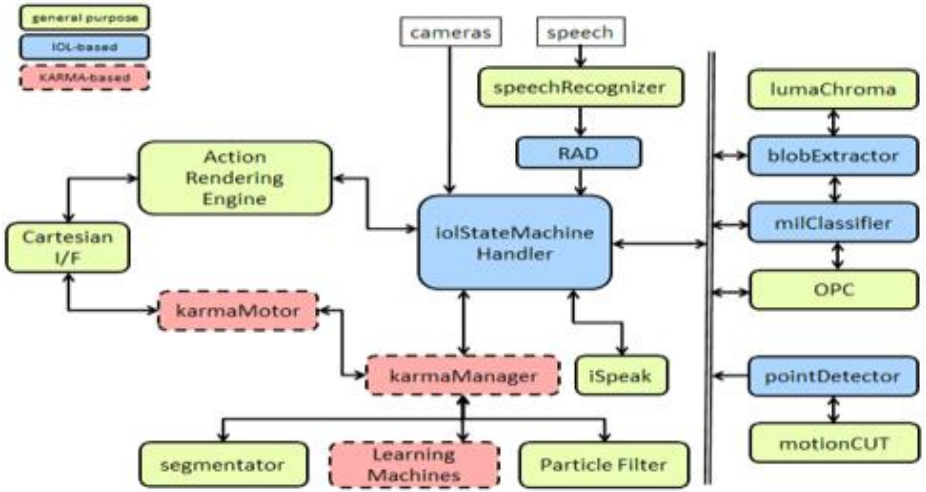
Winter School on Humanoid Robot Programming
Santa Margherita Ligure, 6-15 feb 2018


FAIRPLEX
POLARIS

Due to a human programming error, the robot fell when transitioning from the driving task to the egress task (the foot throttle controller wasn't turned off). This caused the robot to the fall and faceplant out of the car onto the asphalt

source: http://drc.mit.edu/

System Integration



Key issues

Complexity: distributed processing, heterogeneous systems, noise, real-time

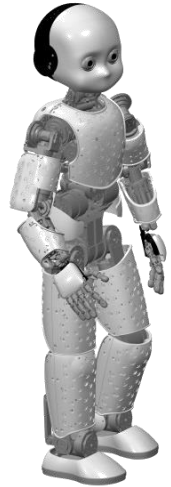
Asynchronous development

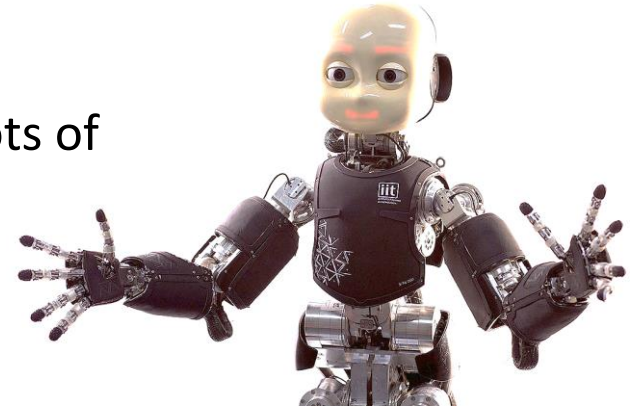
Variability: various scenarios and platforms

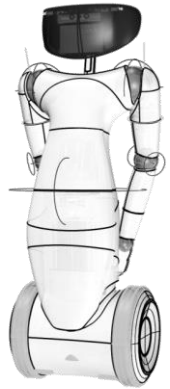
Fast prototyping

Lack of standards

Fluctuation in hardware and algorithms, lots of open questions



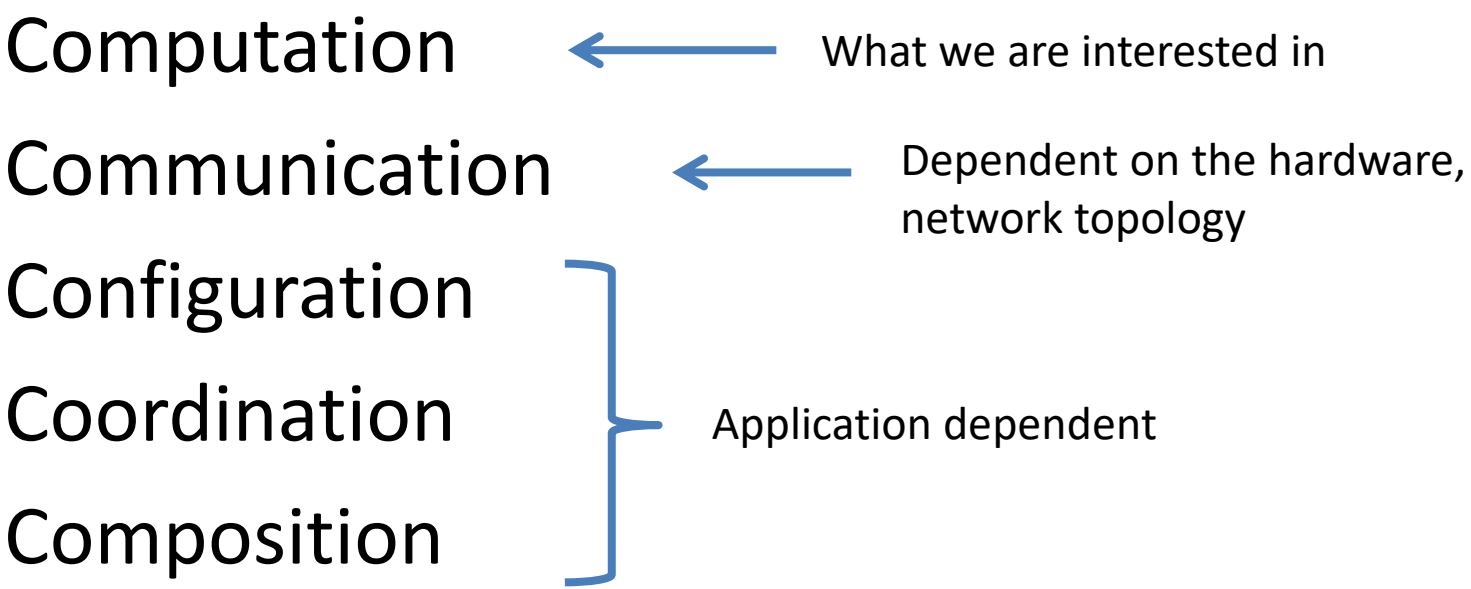




Approach

- Major cost in software development is debugging, recycling code is key
- Component driven software development
- Middleware: factors out platform specificities
  - Hardware (robot abstraction layer)
  - Communication
  - Operating system
  - Parameters
  - Computing infrastructure
- Testing: test driven development

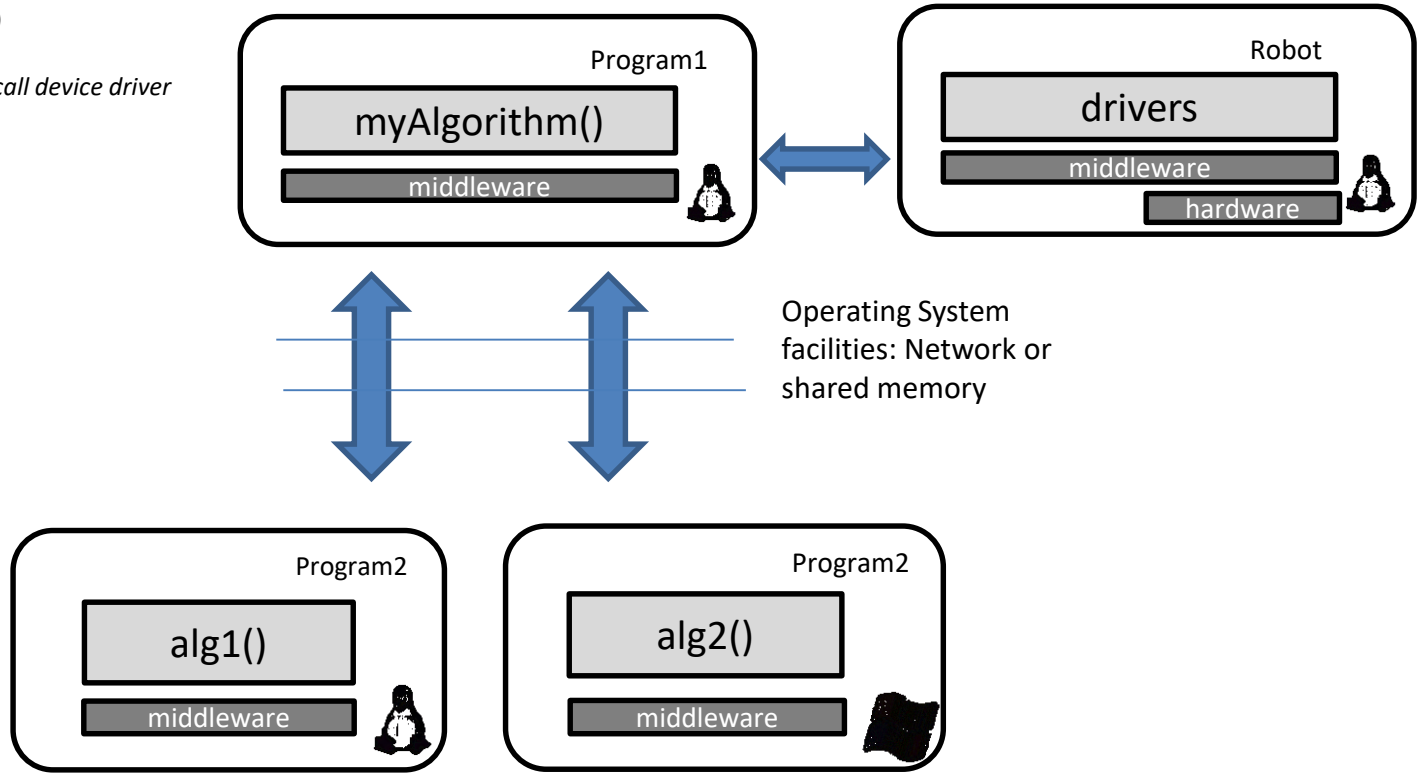
Component driven software development



BRICS Component Model, 2013

```
output myAlgorithm(input)
{
   ...
   data=readSensor()  //call device driver
   out = call alg1(data)
   …
   out = call alg2(data)

   controlMotors(out)  //call device driver
   …
}

output alg1(input)
{code}

output alg2(input)
{code}
```

```
output myAlgorithm(input)
{
  data=readSensor()  //call device driver
  out = query(alg1, data)
  …
  out = query(alg2, data)

  controlMotors(out)  //call device driver
  …
}

output alg1(input)
{code}

output alg2(input)
{code}
```



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)
- Language independent (provided interfaces are compatible)

Middleware: general concepts

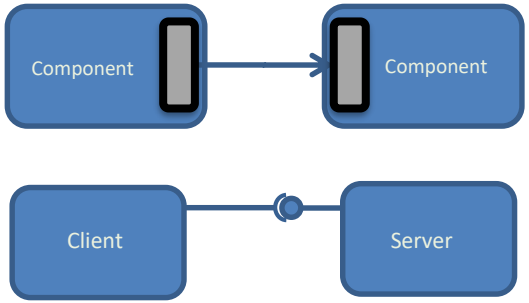
- Information sharing model
- Communication model, timing
- Serialization, Interface Definition Language
- Data persistency
- Quality of Service control
- Hardware abstraction layer

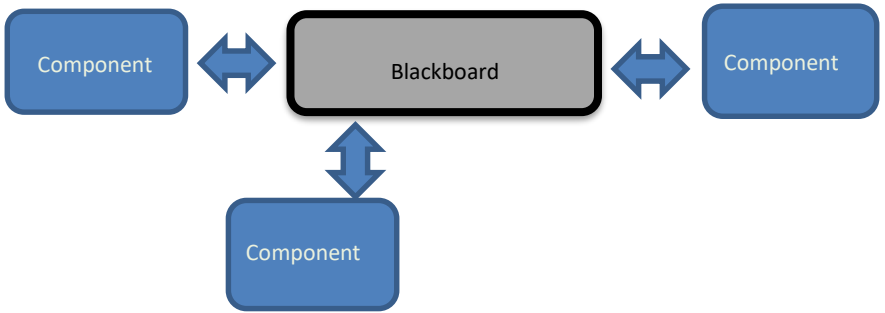
Middleware: general concepts

- Information sharing model
- Communication model, timing
- Serialization, Interface Definition Language
- Data persistency
- Quality of Service control
- Hardware abstraction layer

Information sharing model

- Data ports
- Services
- Data centric (blackboard)

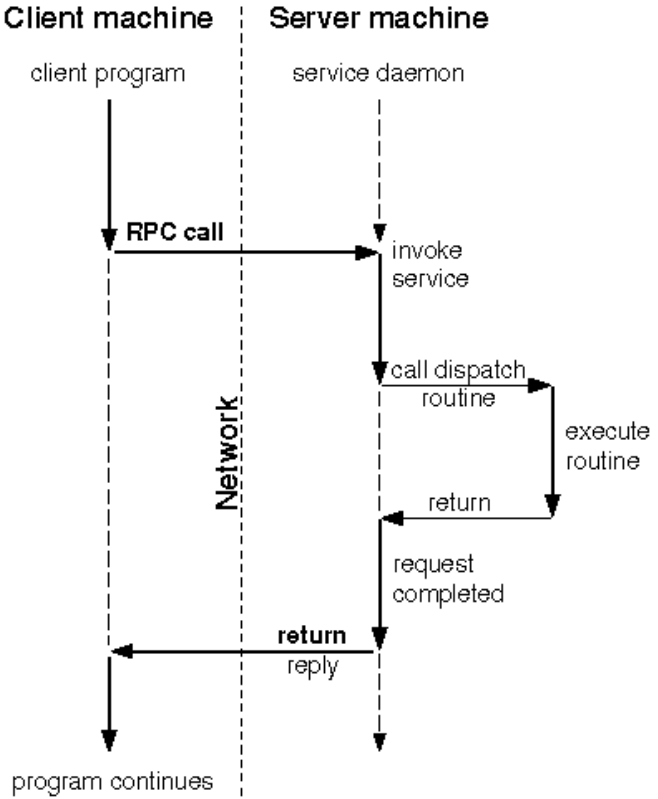




Communication model

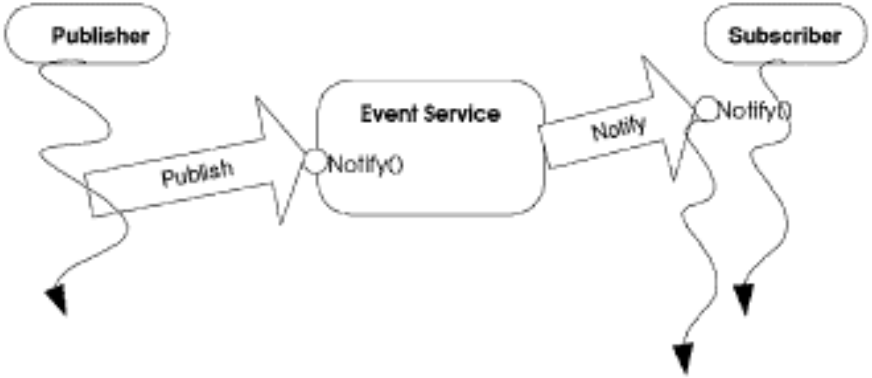
- Remote Procedure Calls (RPC)
  - Remote invocation of an object
  - Synchronous nature (although variant exists)
- Publish/Subscribe
  - Space, time, synchronization decoupling

RPC: Timing


Client machine
Server machine
client program
service daemon
RPC call
invoke service
call dispatch routine
execute routine
Network
return
request completed
return
reply
program continues

Publish-subscribe

Space decoupling

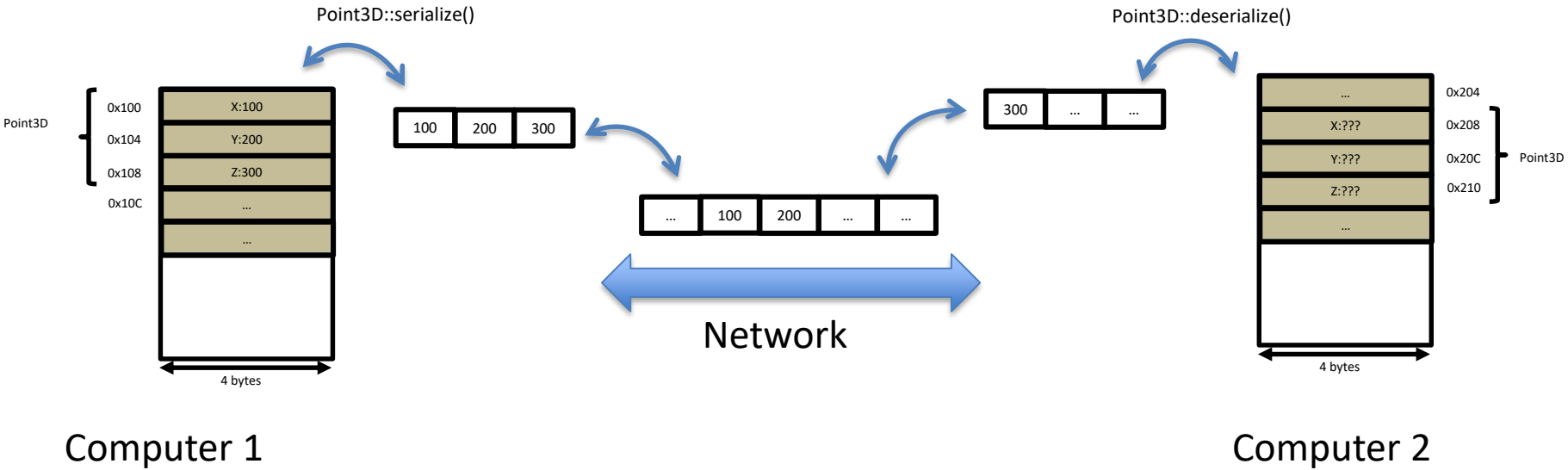


Synchronization decoupling

Eugster, et al., The Many Faces of Publish/Subscribe, 2003.

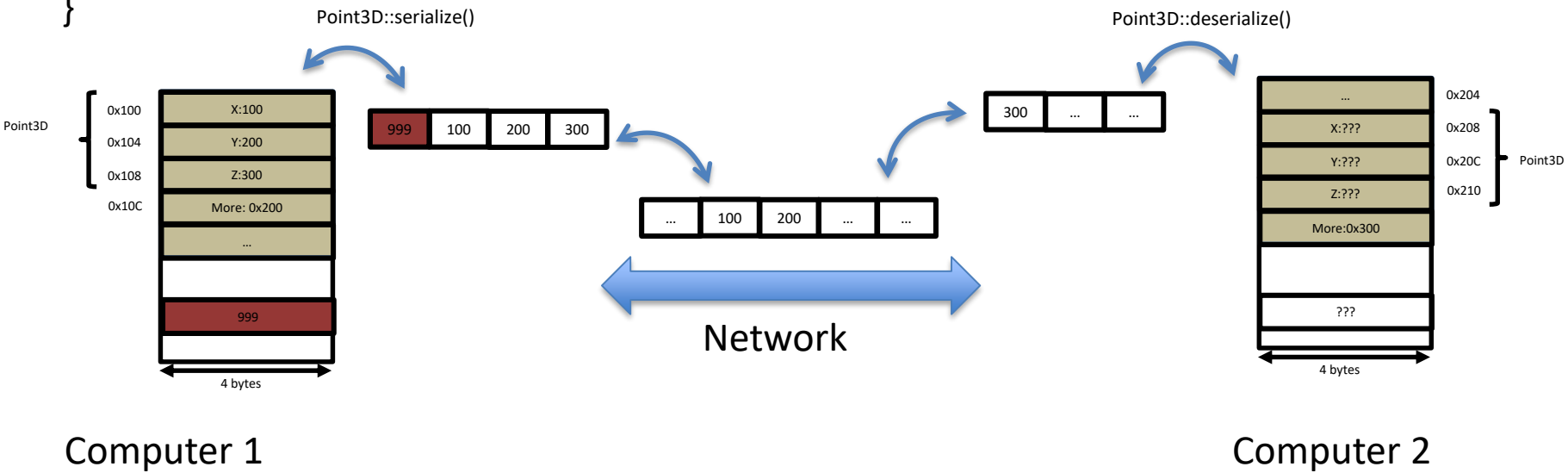
Serialization

```
Class Point3D
{
        int x;
        int y;
        int z;
}
```



Serialization

```
Class Point3D
{
        int x;
        int y;
        int z;
        char *more;
}
```



Computer 1

Computer 2

Interface Definition Language

- Allows describing types
- A compiler then generates required code to define the type, including serialization functions
- Language independent, if the middleware is cross-language

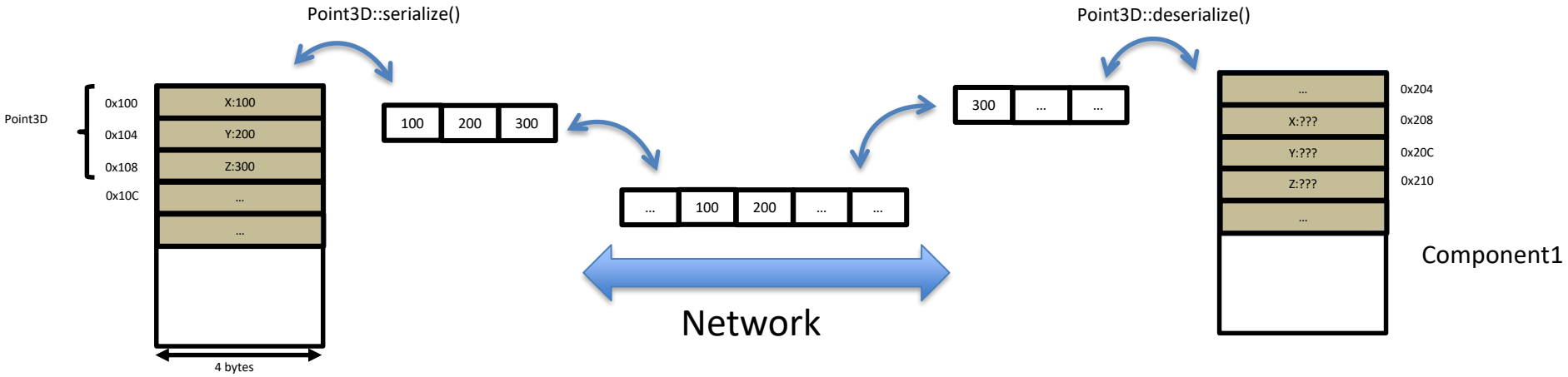
```
point3d.thirft
struct Point3D {
      1: i32 x;
      2: i32 y;
      3: i32 z;
}
```


```
Point3D.h/Point3D.cpp
class Point3D
{
      int x;
      int y;
      int z;

      serialize();
      deserialize();

}
```

Data storage (persistency)

- What happens when the sender is faster than the receiver?
- Buffering:
  - First In First Out (messages are queued, at the cost of latency)
  - Oldest Packet Drop: minimize latency, drop messages if required

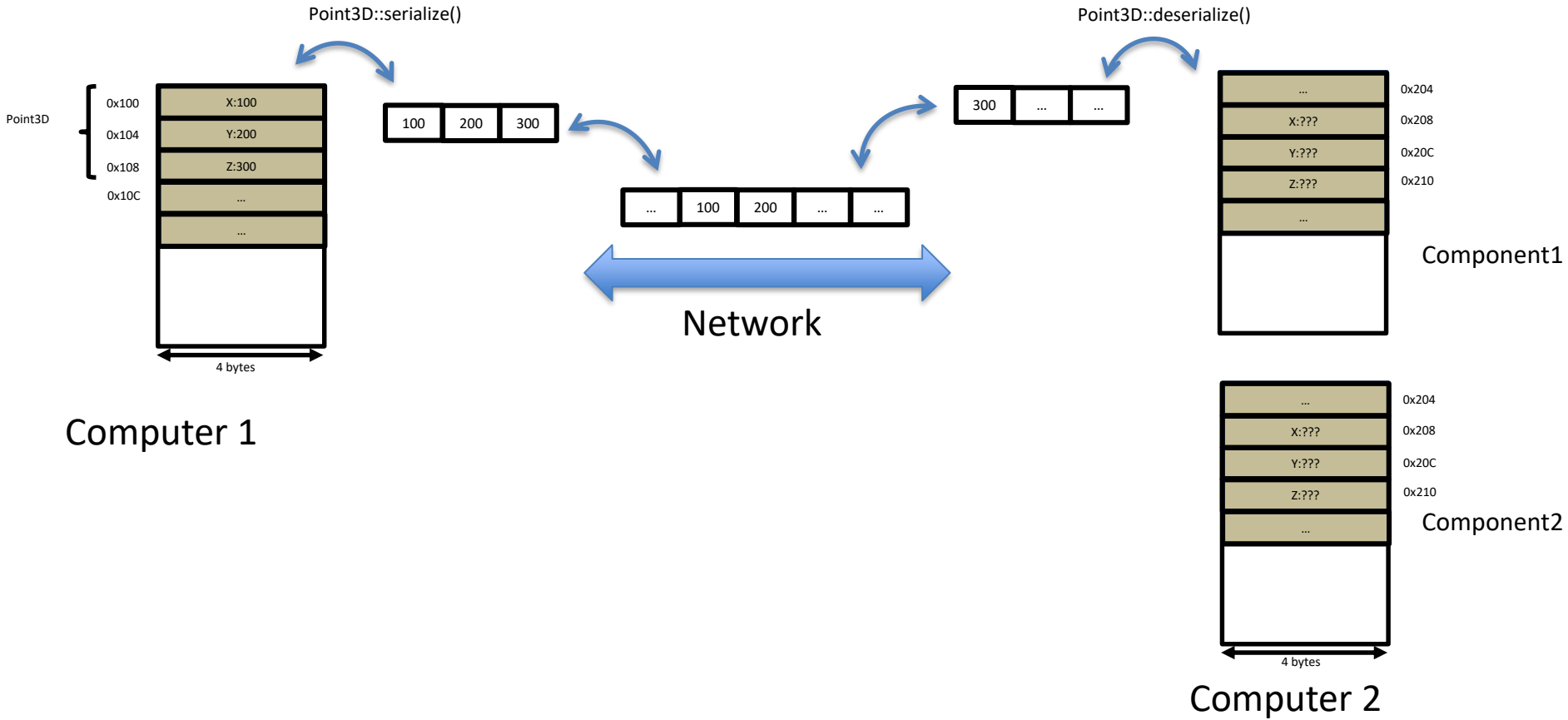
Reducing latency/memory copies



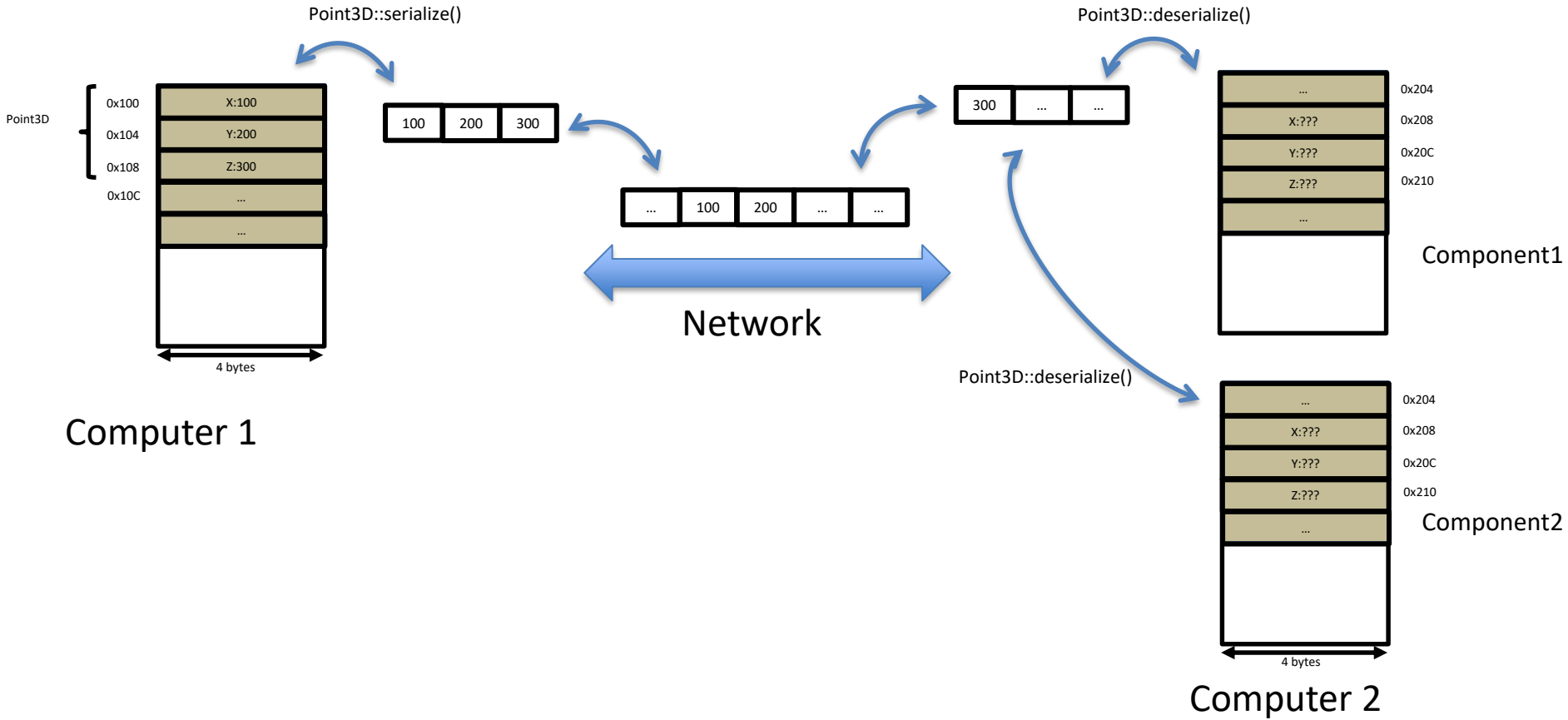
Computer 1

Computer 2

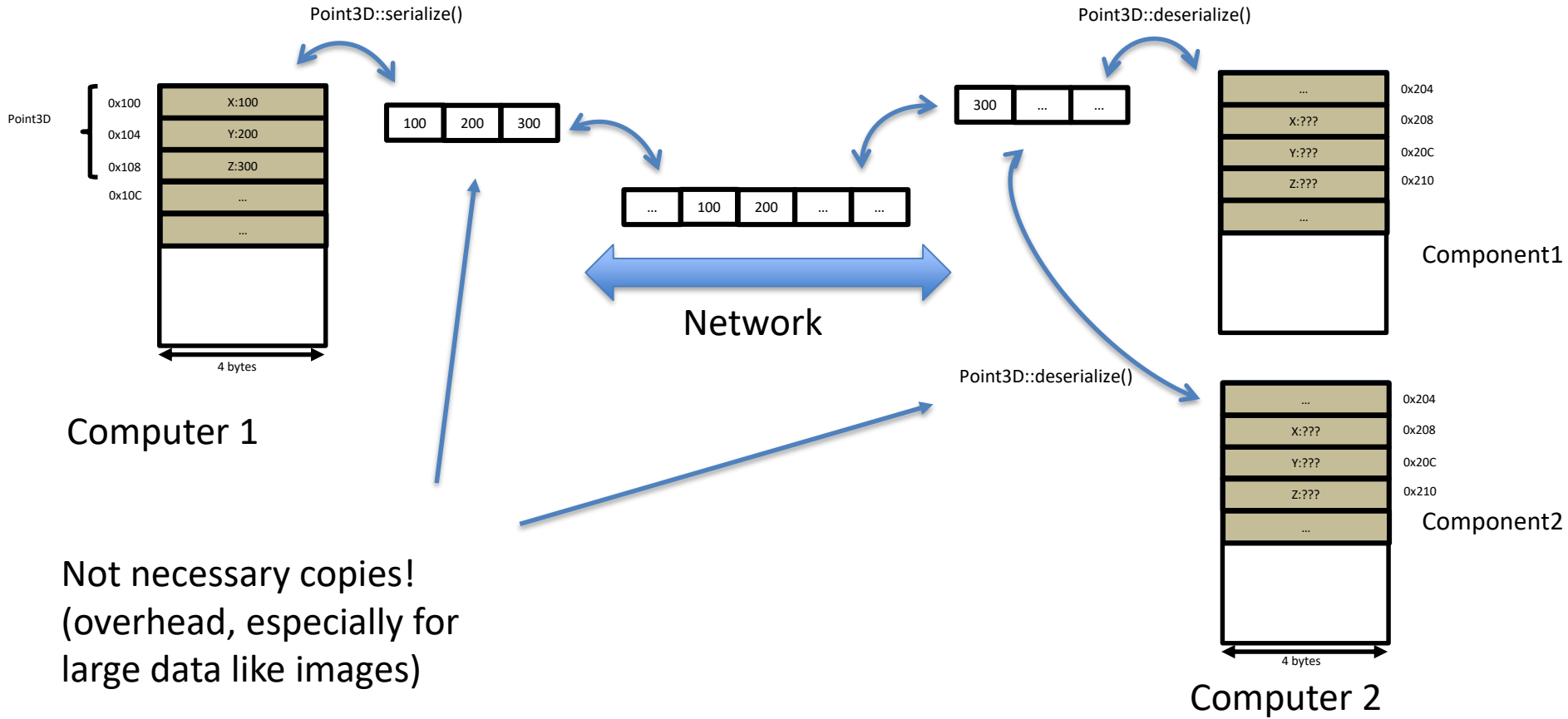
Reducing latency/memory copies



Reducing latency/memory copies

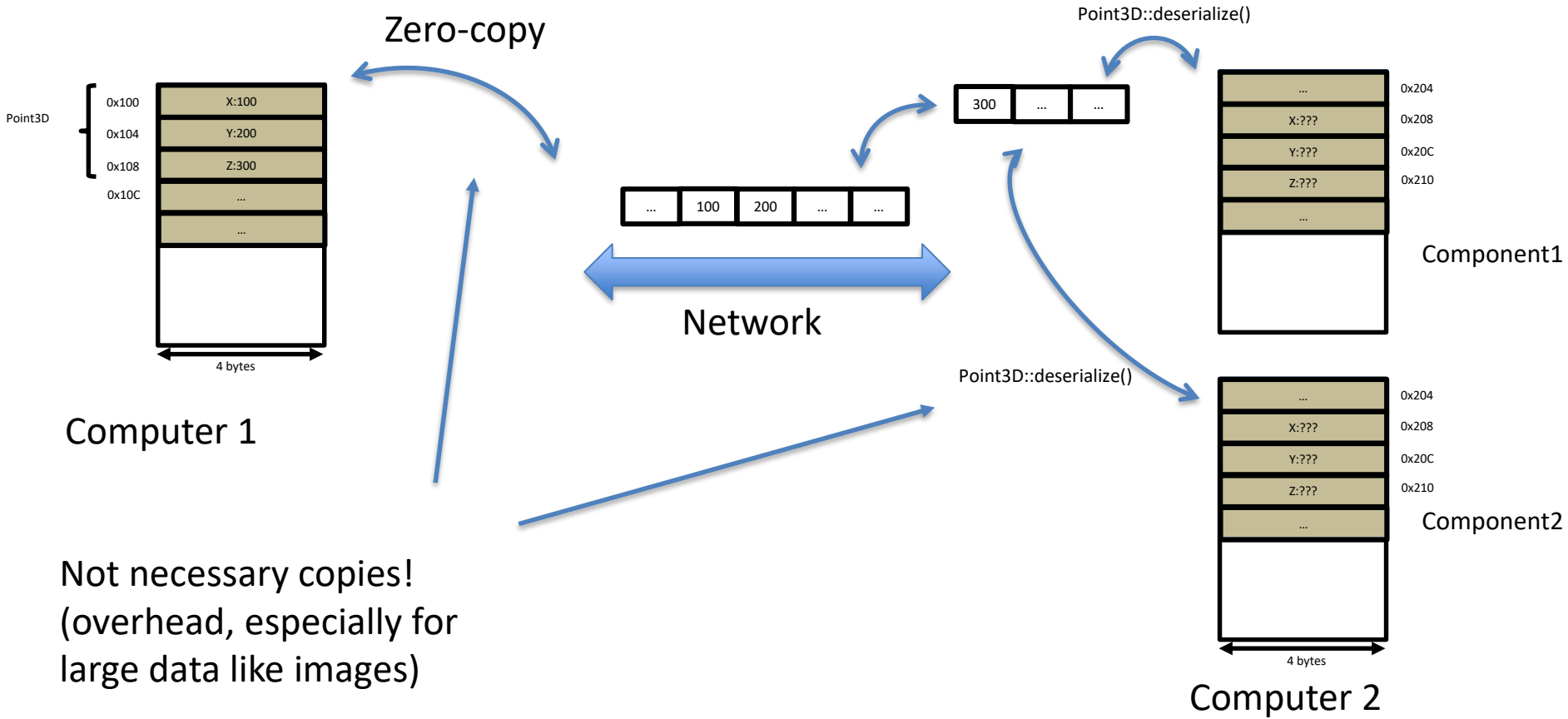


Reducing latency/memory copies



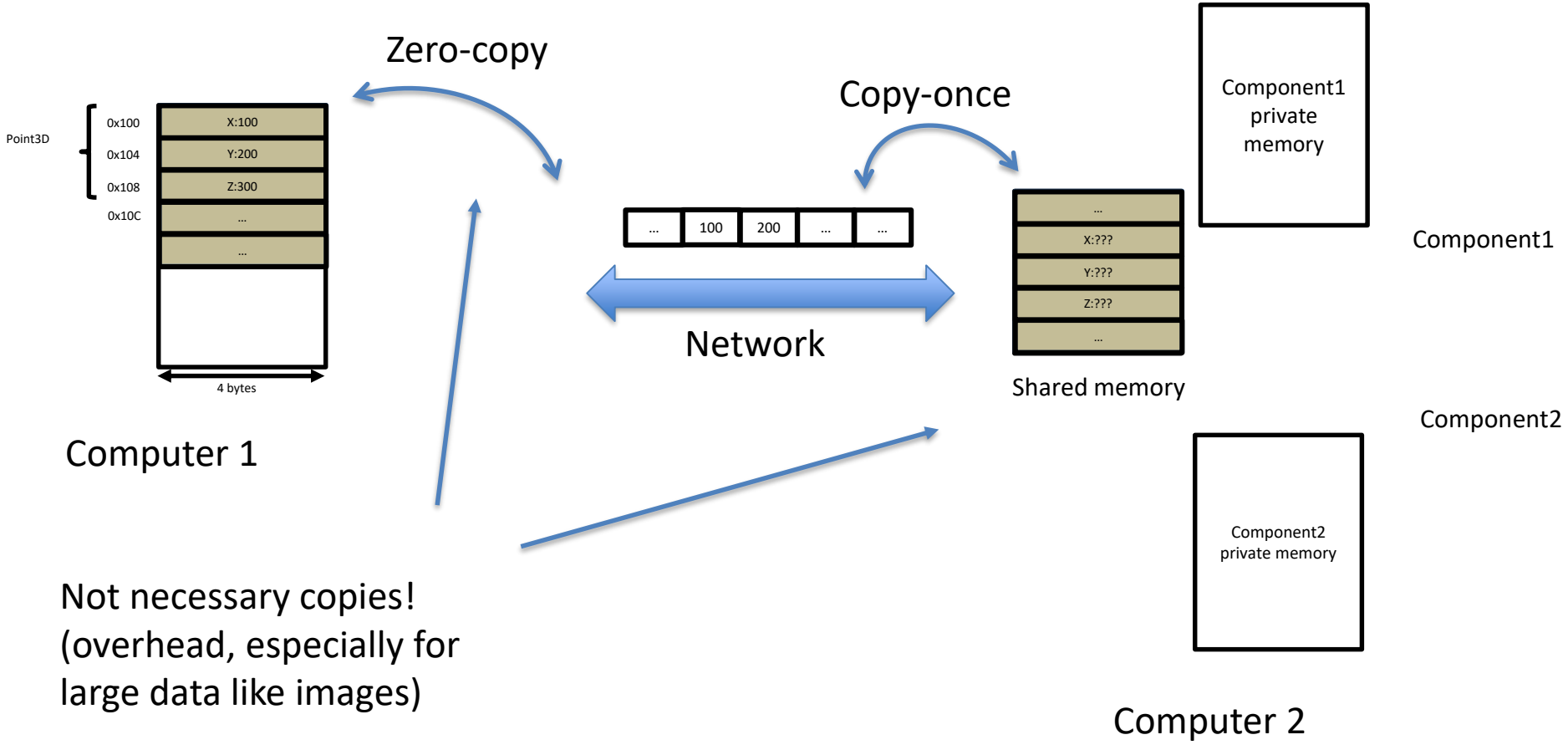
Not necessary copies! (overhead, especially for large data like images)

Reducing latency/memory copies

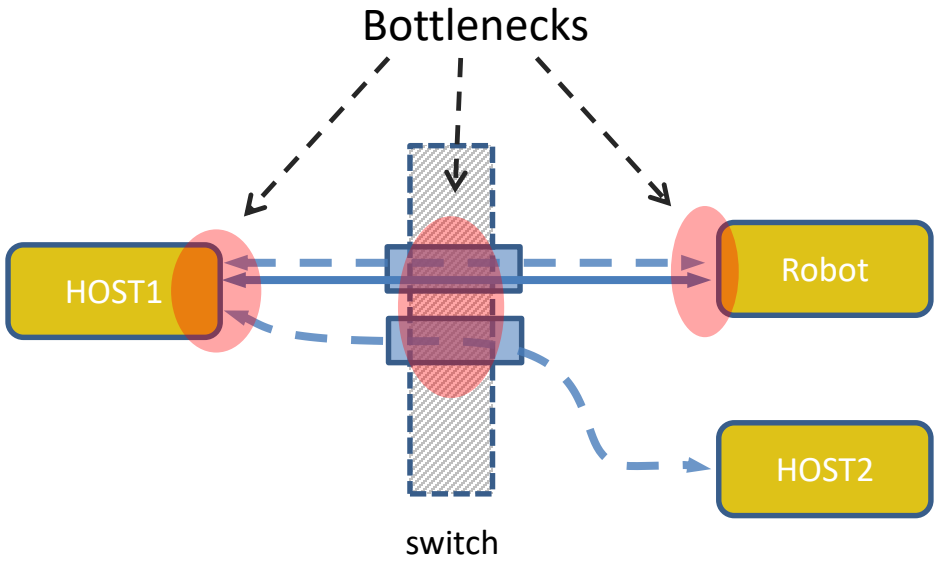


Not necessary copies!
(overhead, especially for large data like images)

Reducing latency/memory copies



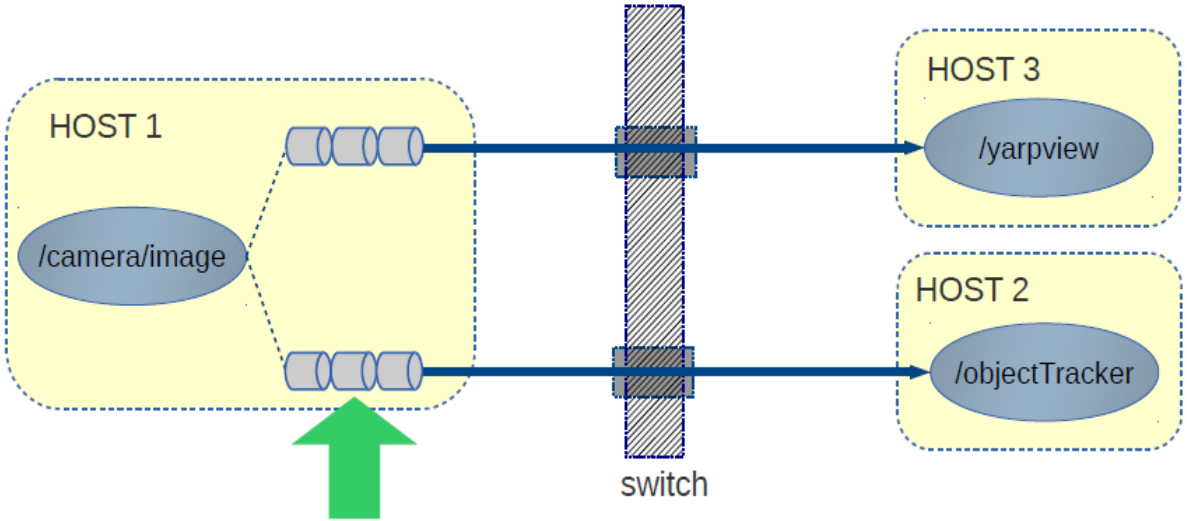
Quality of Service Control



Determinism is affected by:

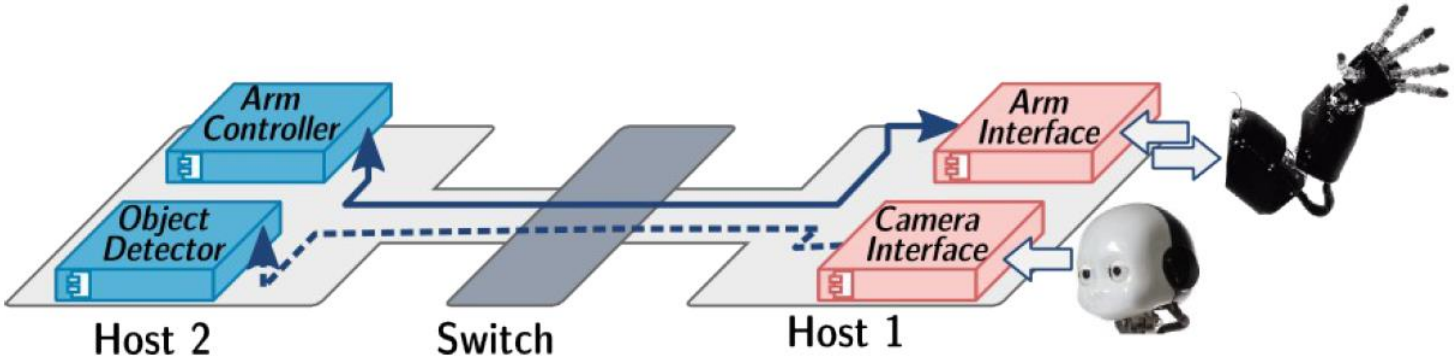
- Thead scheduling (CPU usage)
- Packet conflicts (network usage)

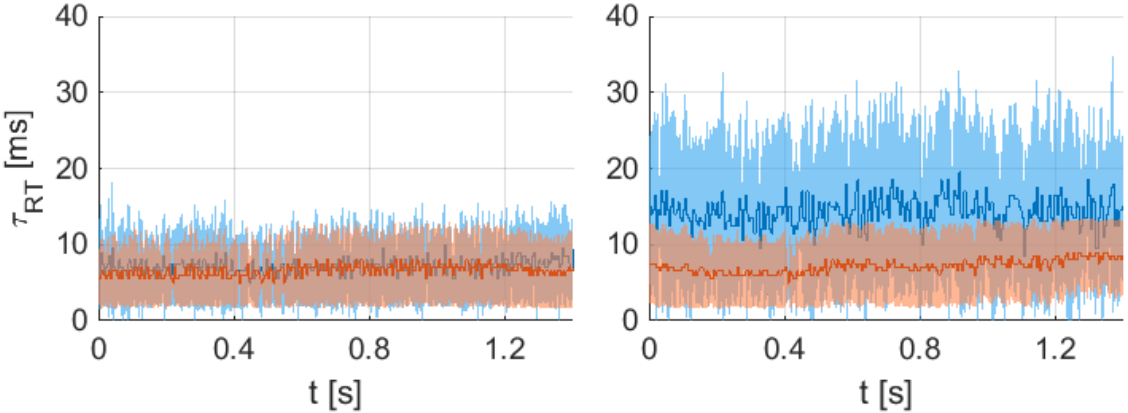
Quality of Service Control allows to attach priority levels to individual connections and data, for example by **increasing thread priorities** and reducing network bottlenecks using packet **QoS**




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

Paikan et al., A Best-Effort Approach for Run-Time Channel Prioritization in Real-Time Robotic Application IROS 2015


Arm Controller
Object Detector
Arm Interface
Camera Interface
Host 2
Switch
Host 1

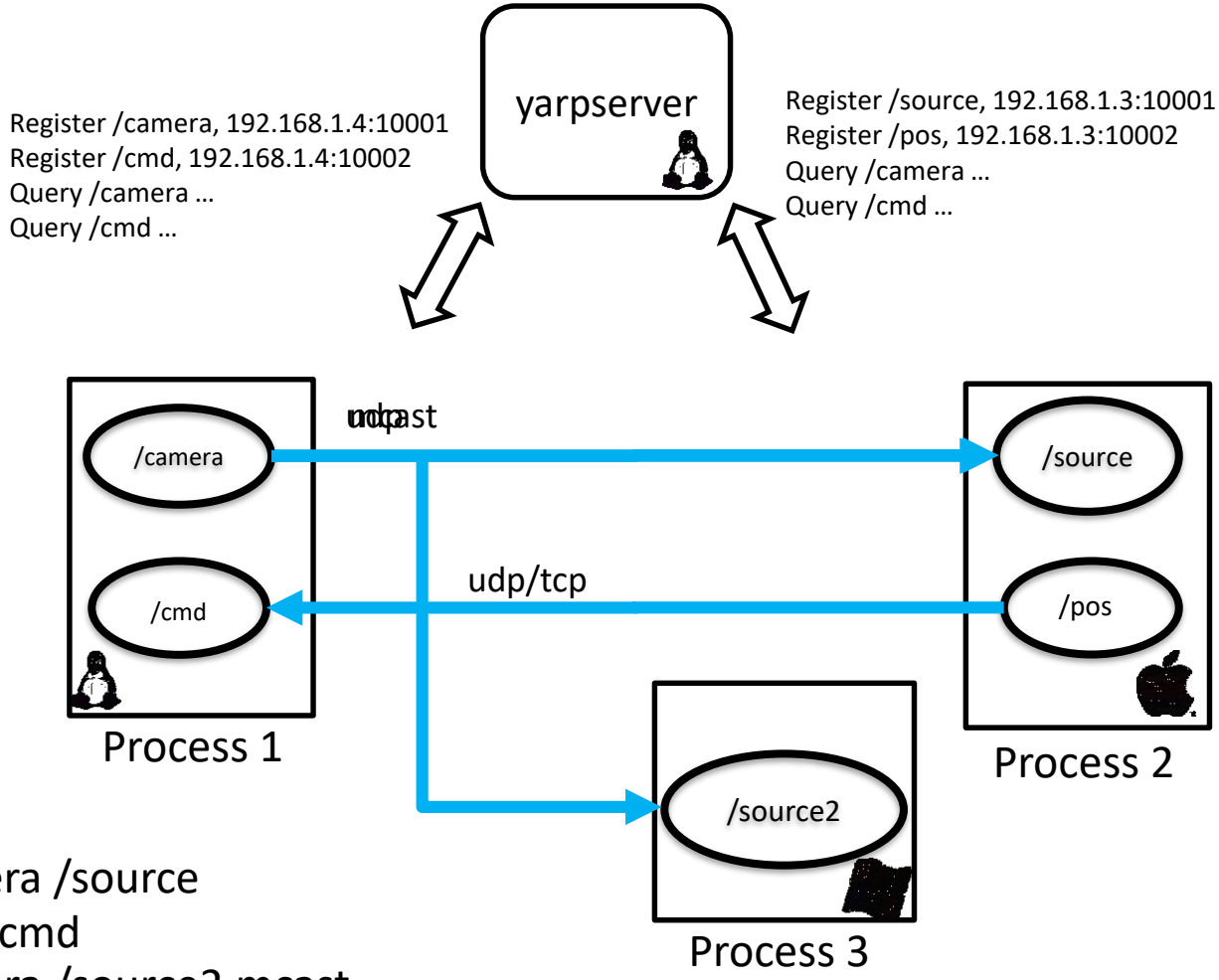

40
30
20
10
0
τ_{RT} [ms]
0
0.4
0.8
1.2
t [s]
40
30
20
10
0
0
0.4
0.8
1.2
t [s]

Which Middleware

- Robot Operating System (ROS)
- YARP
- OROCOS
- SmartSoft
- CORBA
- ICE
- OMG DDS
- Many others: OpenRDK, Mira…

YARP/ROS

- Components are executables which communicate through named objects called Ports (YARP), Topics (ROS)
- A central server (yarpserver/roscore) keeps tracks of names and allows Ports/Topics to be reachable
- Communication is peer-to-peer between Ports/Topics
- Publish-subscribe and client-server (although emphasis is on the former)



connect /camera /source
connect /pos /cmd
connect /camera /source2 mcast

YARP/ROS comparison

YARP

Run-time reconfiguration of connections

Pluggable protocols and **devices**

Multicast for efficient one-to-many communication

Multi-platform

QoS, channel prioritization

LGPL/GPL

Smaller community

No packet management

ROS compatible protocol

ROS

Strongly **typed**

Rich set of **libraries** and **tools**

Eco-system, very active community

Packet management

BSD license

Ubuntu based

Restricted set of protocols

All connections from a topic use the same protocol

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.
- D. Brugali, Model-Driven Software Engineering in Robotics, IEEE Robotics and automation magazine, Sept. 2015

Questions?