Embedded Networks

Models of Communication

Summer Term 2008

CO-OPERATIVE SYSTEMS

Which model of communication?
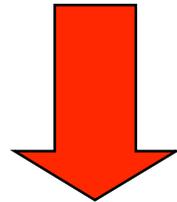
What kind of addressing and routing should be supported by the network?

Which abstractions in the programming model?

Message Passing

**Explicit communication via send and receive: Message passing.**
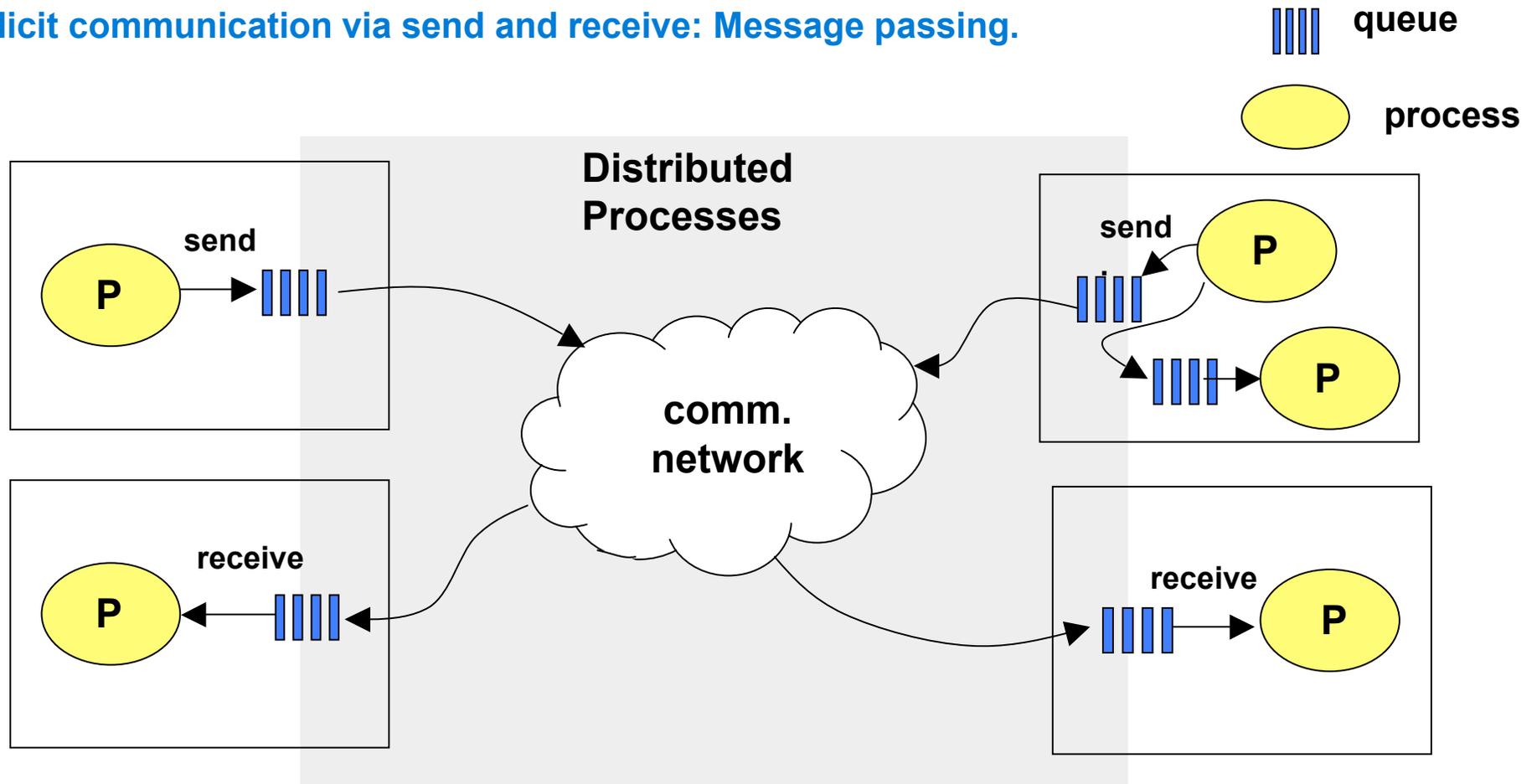
Problem: very low level, very general, poorly defined semantics of communication

Remote Procedure Call

**Function shipping initiates computations in a remote processing entity.
Example: Remote Procedure call.**
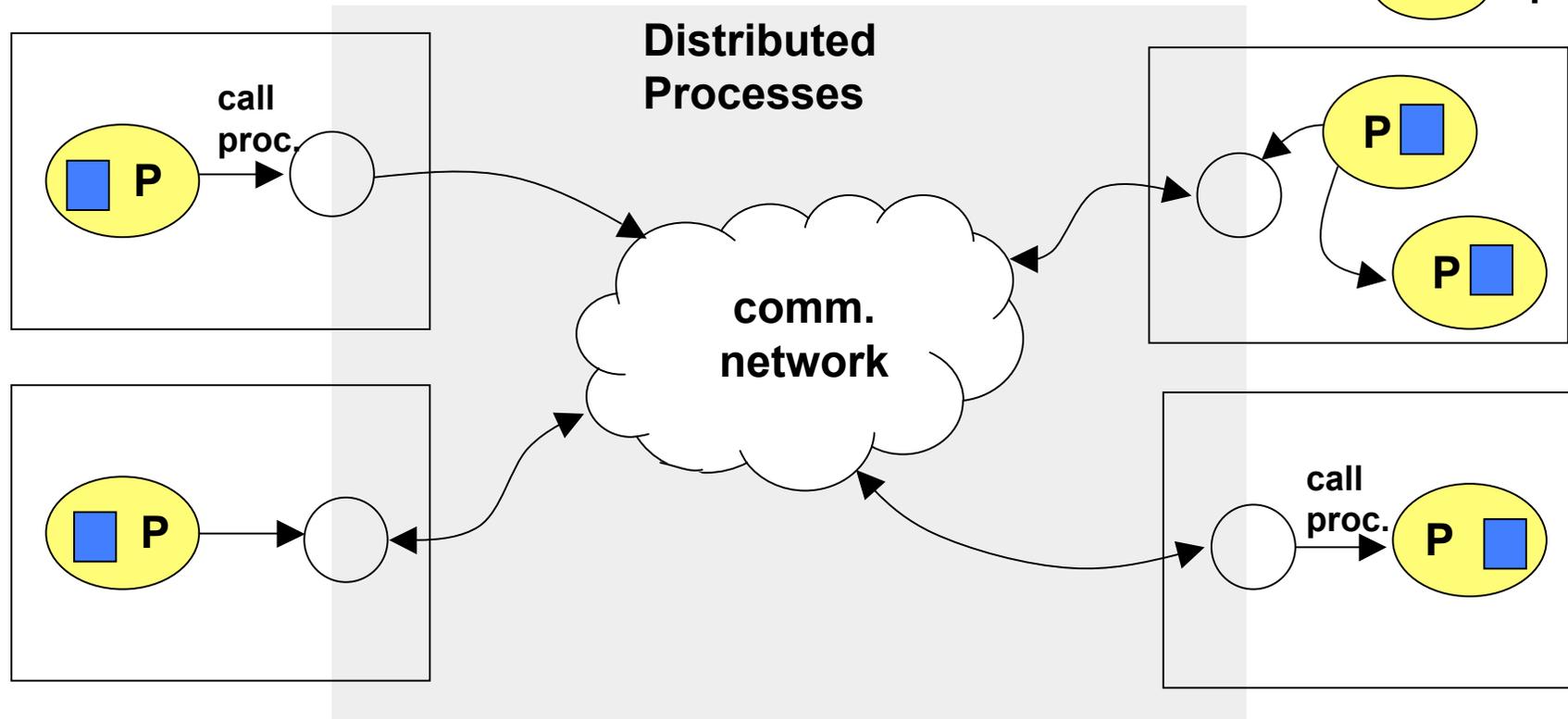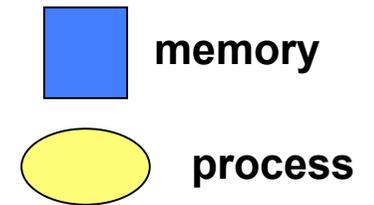
Problem: computation bottlenecks, more complex programming model, references.

Distributed Shared memory

DSM (Data shipping) maintains the read/write semantics of memory
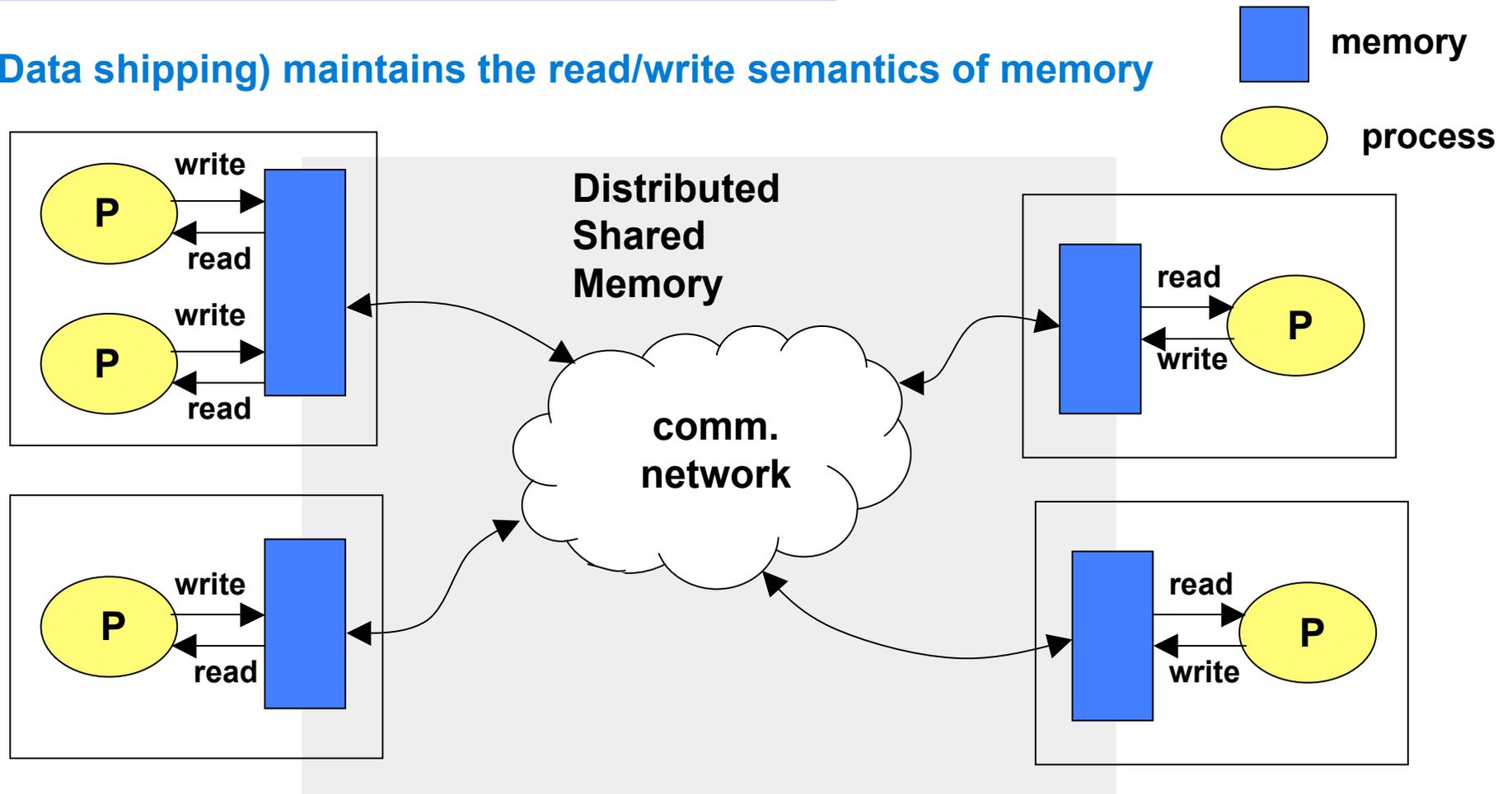

Problem: Consistency in the presence of concurrency and communcation delays

Distributed system architecture

abstracting from HW

windows	linux	linux	PalmOS
HW1 Pentium	HW2 XScale	HW3 PPC	HW4 68K

Distributed system architecture

network

P   TCP/I   P/IP

appl.1   appl.2   appl.1   appl.3   appl.3

abstracting from local OS

middleware

abstracting from HW

windows   linux   linux   PalmOS

HW1 Pentium   HW2 XScale   HW3 PPC   HW4 68K
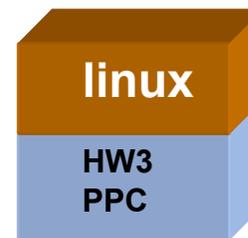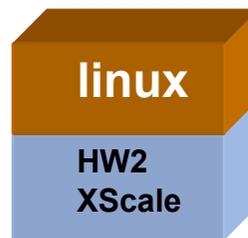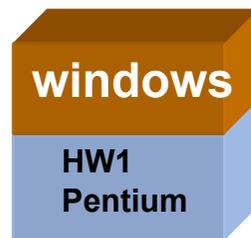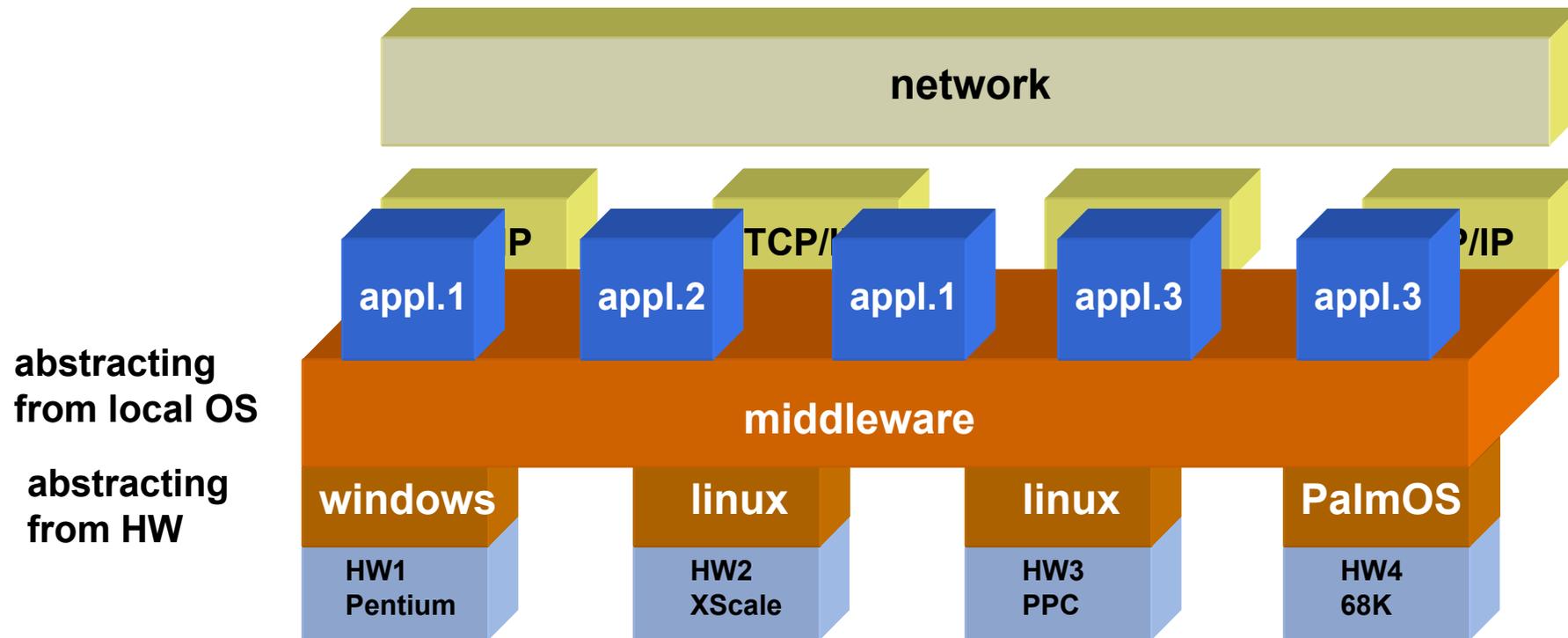
Interaction Structure in Co-operative Systems

many-to-many
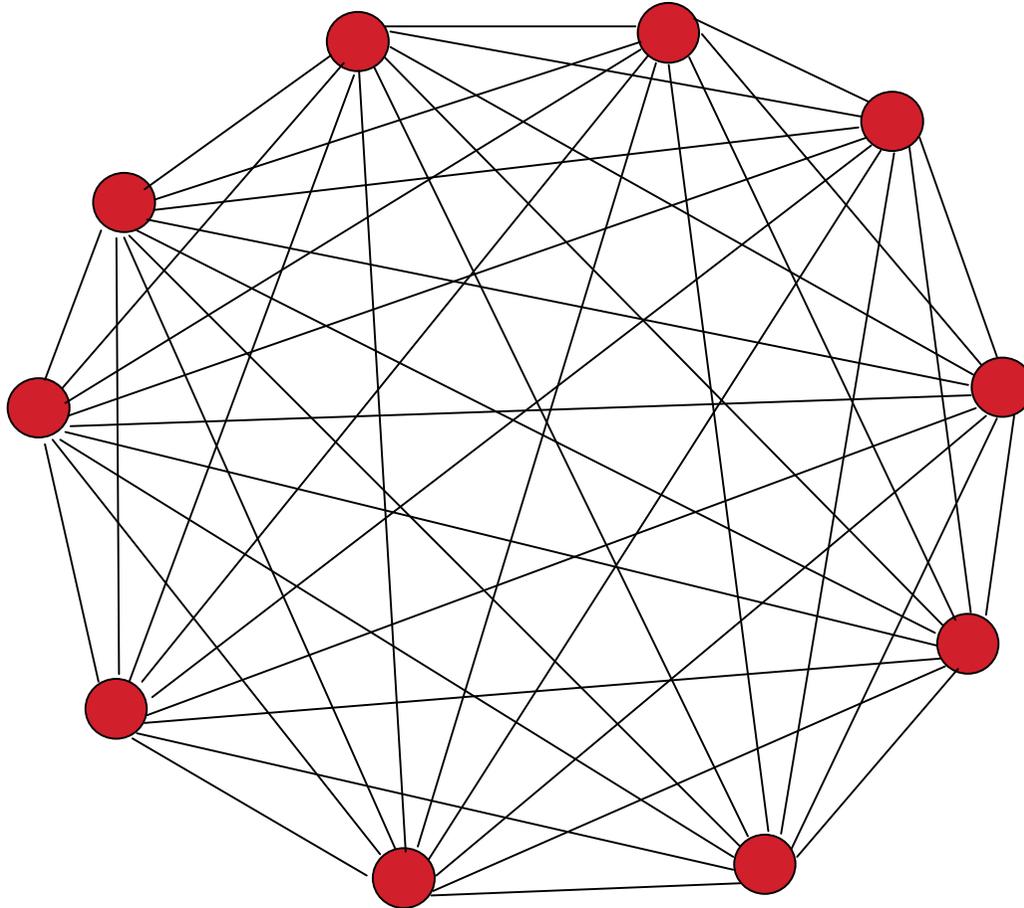
managebility

goal ➡ **sharing information and co-ordinating acitvities**

Abstractions for Communication

➡️ **Message passing**

➡️ **Remote Procedure Call**

➡️ **Remote Object Invocation**

➡️ **Distributed shared memory**

➡️ **Notifications**

➡️ **Publish Subscribe**

➡️ **Shared data spaces**

Abstractions for Communication

Dimensions of Dependencies:

Space Coupling: References must be known
Explicit specification of the destination, i.e. producer must know where to send the message. Message contains an ID specifying an address or name.

Flow coupling: Control transfer with communication
Defines whether there is a control transfer coupled with a message transfer.
E.g. if the sender blocks until a message is correctly received.

Coupling in time: Both sides must be active
Communication can only take place if all partners are up and active.

Message passing

Connected socket, e.g. TCP

logical channel

producer   abstraction   consumer   interface   thread   *
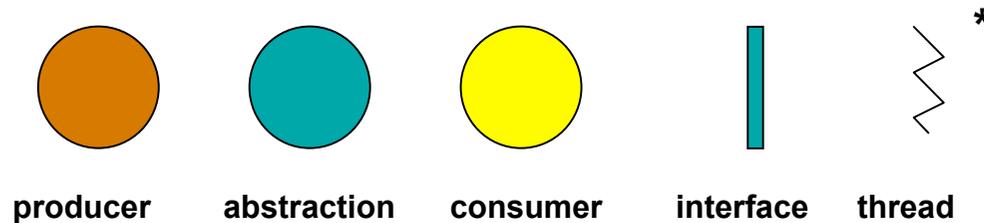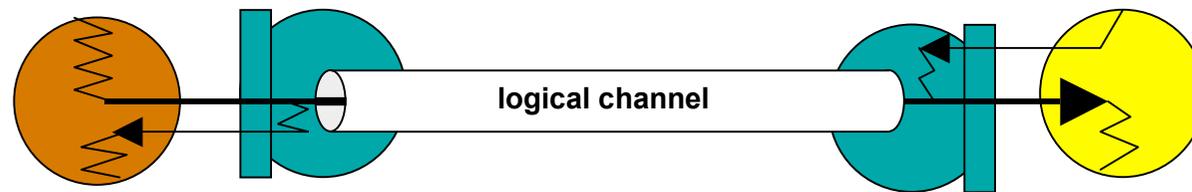
primitives: send (), receive ()

Coupling: flow, space, time

* **Notation acc. P. Eugster: Type-Based Publish Subscribe, PhD-thesis, EPFL, Nr. 2503, 2001**

Remote Procedure Call (RPC)

proxy, stub

skeleton

Relation: one-to-one

Coupling:
Space: destination is explicitly specified
Flow: blocks until message is delivered
Time: both sides must be active

Variations of RPC

Asynchronous RPC with pull

Asynchronous RPC with call-back
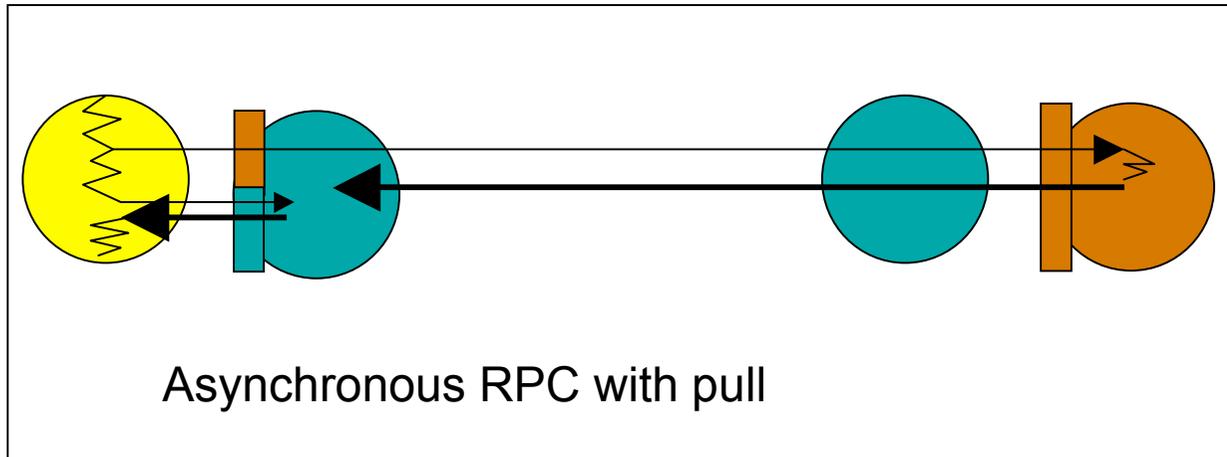
Example: Concurrent Smalltalk

Relation: one-to-one

Coupling:
Space: destination is explicitely specified
Flow: no flow coupling
Time: both sides must be active
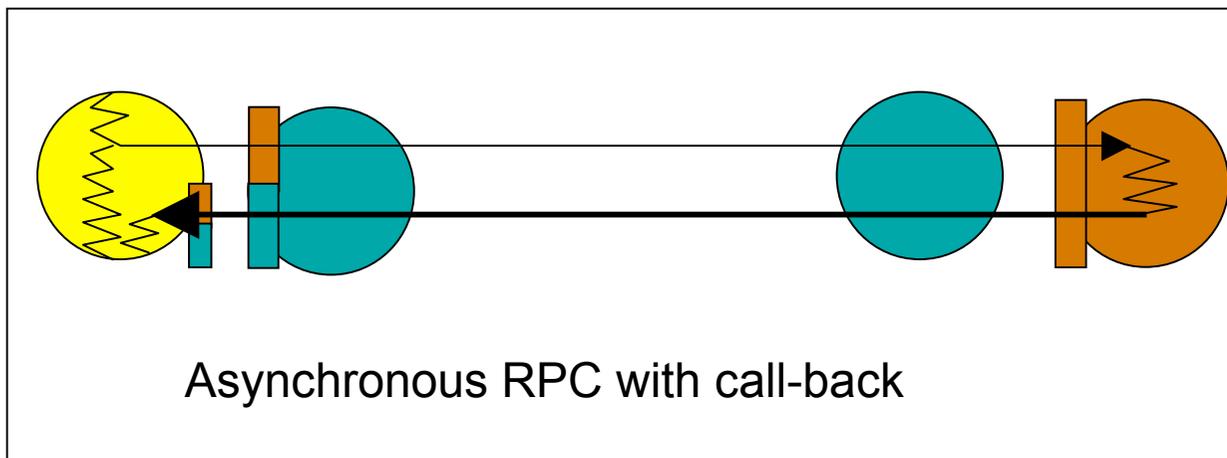
Example: Eiffel

Notification

Observer/ listener

observable

registration
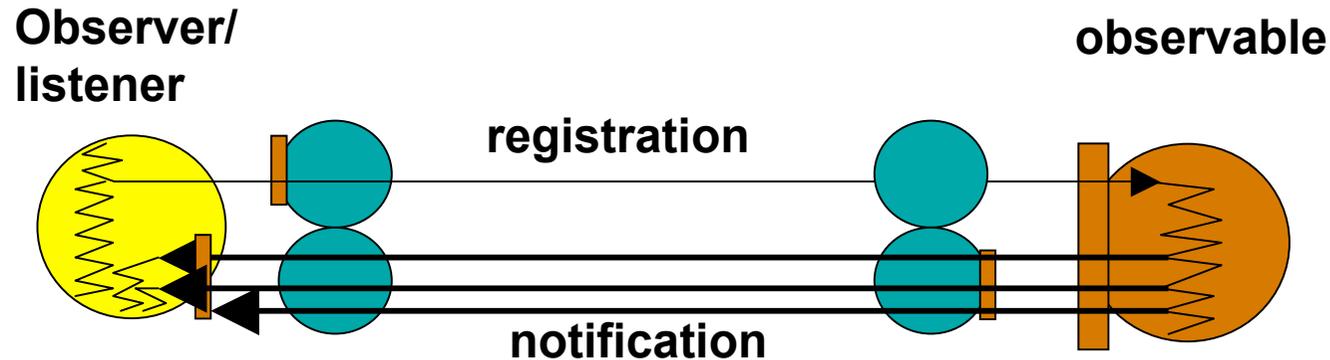
notification

Examles: Java

Relation: one-to-many

Coupling:
Space: Yes (Observable/Observer pattern (delegation))
Flow: none
Time: both sides must be active (notification performed by RMI)

Shared Data Spaces

logical container
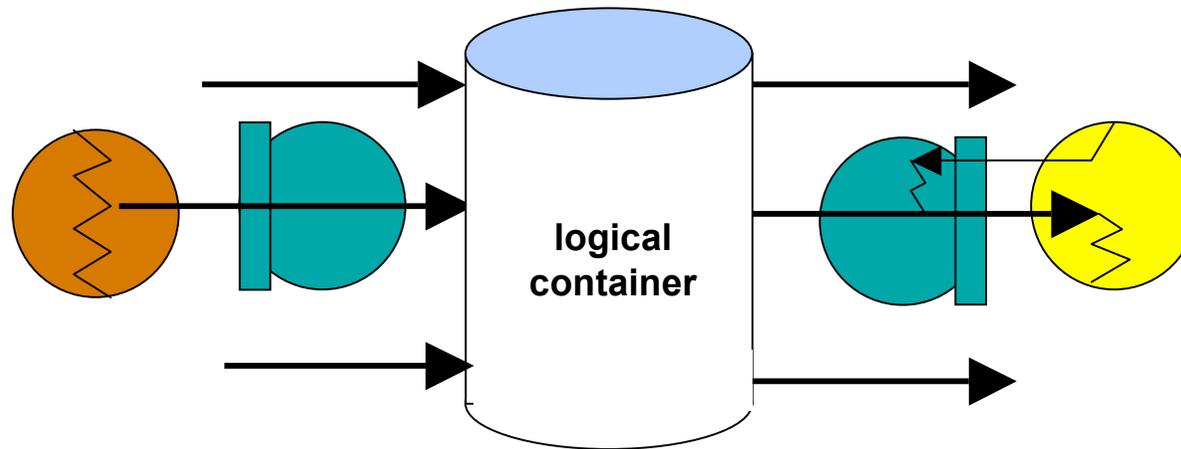
Relation: many-to-many

Coupling:
Space: none
Flow: consumer side
Time: none

Examples:
Linda Tuple Space
Java Spaces
ADS Data field

Shared Data Spaces

Processes communicate via the "Tuple" Space,
A tuple is only data, no address, no identifier,
A tuple is a data structure similar to a struct in C,

Examples: ("3numbers", 3, 6, 7), ("matrix" , 1, 5, 3.23, 8),
("faculty", "is_member_of", "franz", "maria", "otto")

Primitives (operations) in LInda:
op. in: takes (and removes) an element from the tuple space
op. read: reads an element from the tuple space
op. out: puts a tuple into the tuple space
op. eval: allows to eveluate the fields of a tuple, results are put in the
tuple space [example: ("product", mult(4,7))]

No Tuple is ever (over-) written! "out" always put a new item in the space.

Shared Data Spaces

Content-Based Addressing by Tuple matching:

All fields in a template are compared to all tuples.
A match of a template occurs if:

     tuple has the same number of fields
  AND **types of fileds are equivalent**
  AND **contents corresponds**

Example:
<"distance´_sensor", "N", 23>
<"distance´_sensor", "E", 127>
<"distance´_sensor", "S", 127>
<"distance´_sensor", "W", 12>

in(<"distance_sensor", " ", ?i> : reads all distance sensors and removes their
           values from the space.
read(<"distance_sensor", S, ?i>: subsequent read blocks until new S-value has been
           put to the Space.

Shared Data Spaces

P_k

Co-ordination

\<s.count\> := myTS.take (\<"counter", ?i\>);
myTS.write (\<"counter", count+1\>);

\<"counter", 64\>

P_m

\<s.count\> := myTS.take (\<"counter", ?i\>);
myTS.write (\<"counter", count+1\>);
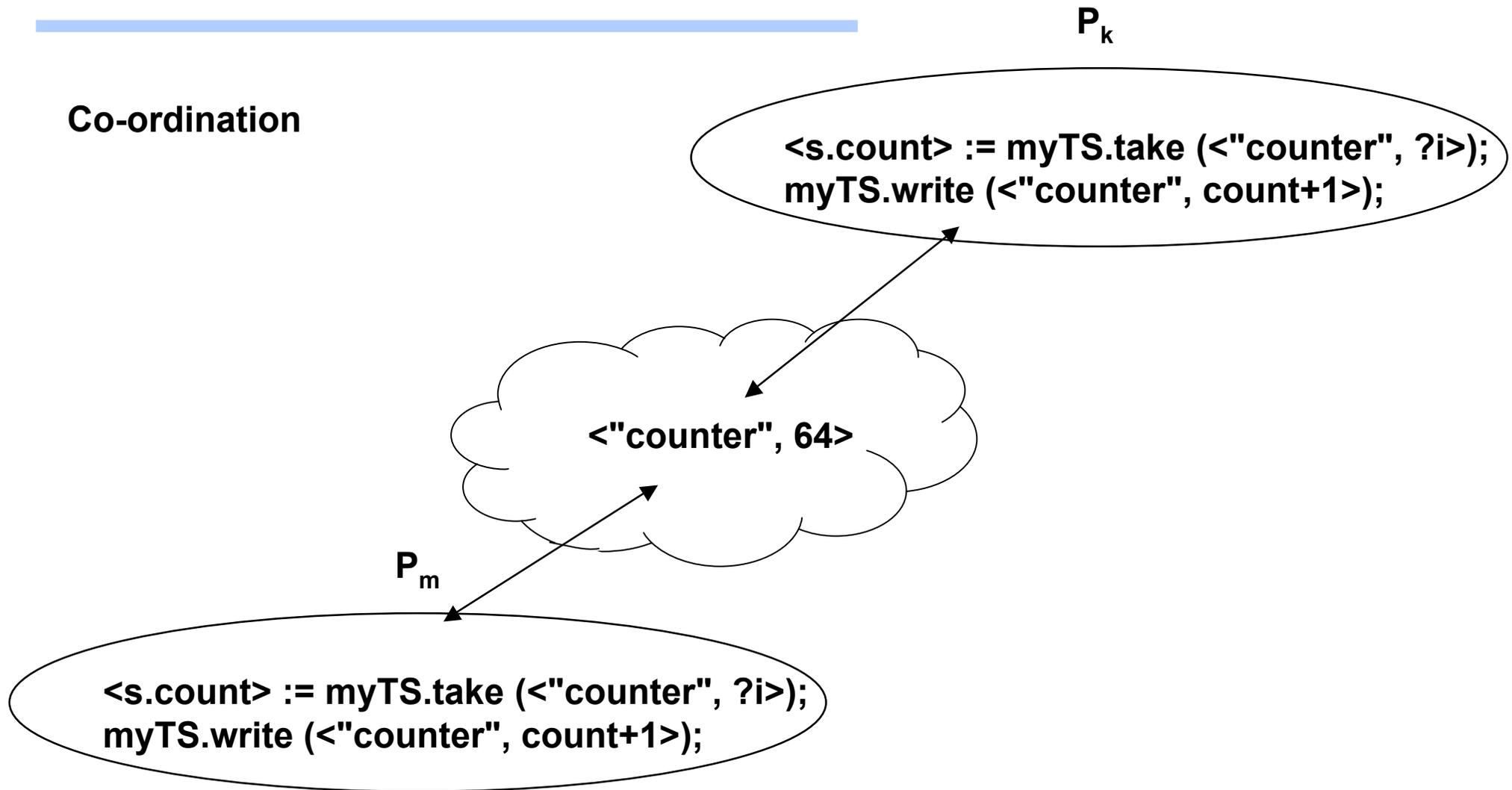
Shared Data Spaces Summary

Immutable Data Storage

➡️ **no write operation!**

➡️ **"out" always adds a data element to the storage**

➡️ **destructive "in" and non-destructive "read"**

➡️ **consistency is preserved by ordering accesses**

➡️ **examples: Linda, JavaSpaces**

Publish/Subscribe

logical channel
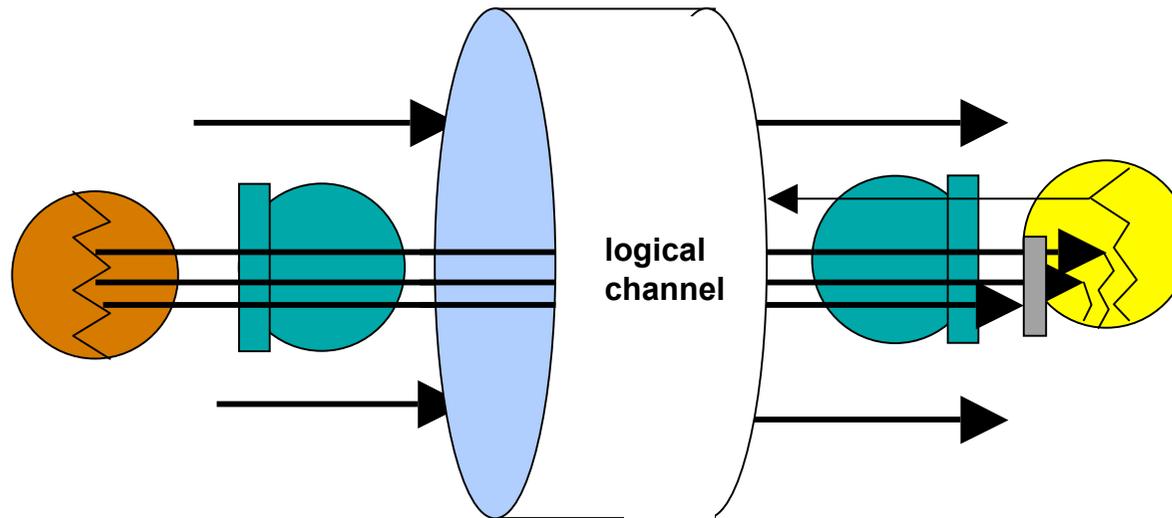
Relation: many-to-many

Coupling:
Space: none
Flow: none
Time: none

Examples:
Information Bus
NDDS
Real-Time P/S
COSMIC

....

....

The Publisher/Subscriber Model

Principle: Keep control local and link systems via event channels
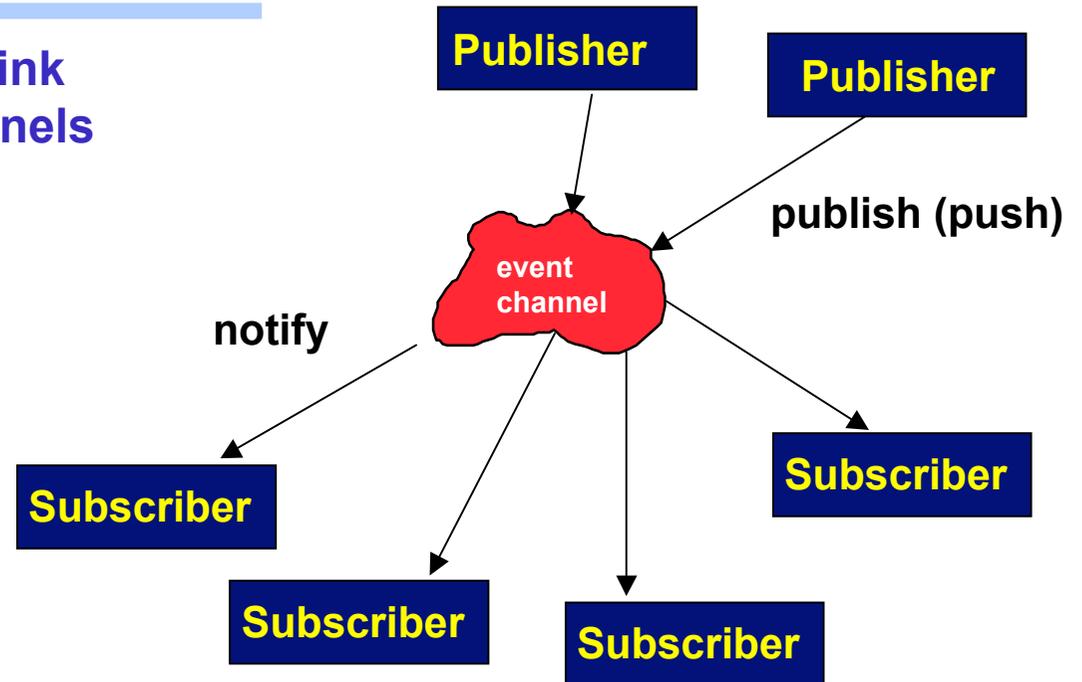
Information Bus *(Oki, Pfluegl, Siegel, Skeen)*
iBus *(Maffeis)*
Real-Time P/S *(Rajkumar, Gagliardi, Sha)*
NDDS *(Real-Time Innovations, Inc.)*
SIENA *(Carzanoga, Rosenblum, Wolf)*
Directed Diff. *(Intanagonwiwat, Govindan, Estrin)*

Publisher

Publisher

publish (push)

event channel

notify

Subscriber

Subscriber

Subscriber

**Subscriber**

Subscriber

Many-to-many communication

Support for event-based spontaneous (generative) communication

Anonymous communication

P/S in a smart sensor application

acceleration sensor

acceleration measurement & crash detection

event: acceleration

publish (subject,attr., [acceleration]);

event: alarm

publish (subject, attr., [alarm]);

configuration parameters
subscribe (subject-uid, ...)

Event Handler

network interface

<network address [acceleration]>

<network address [crash alarm]>

<network address, [config.params]>
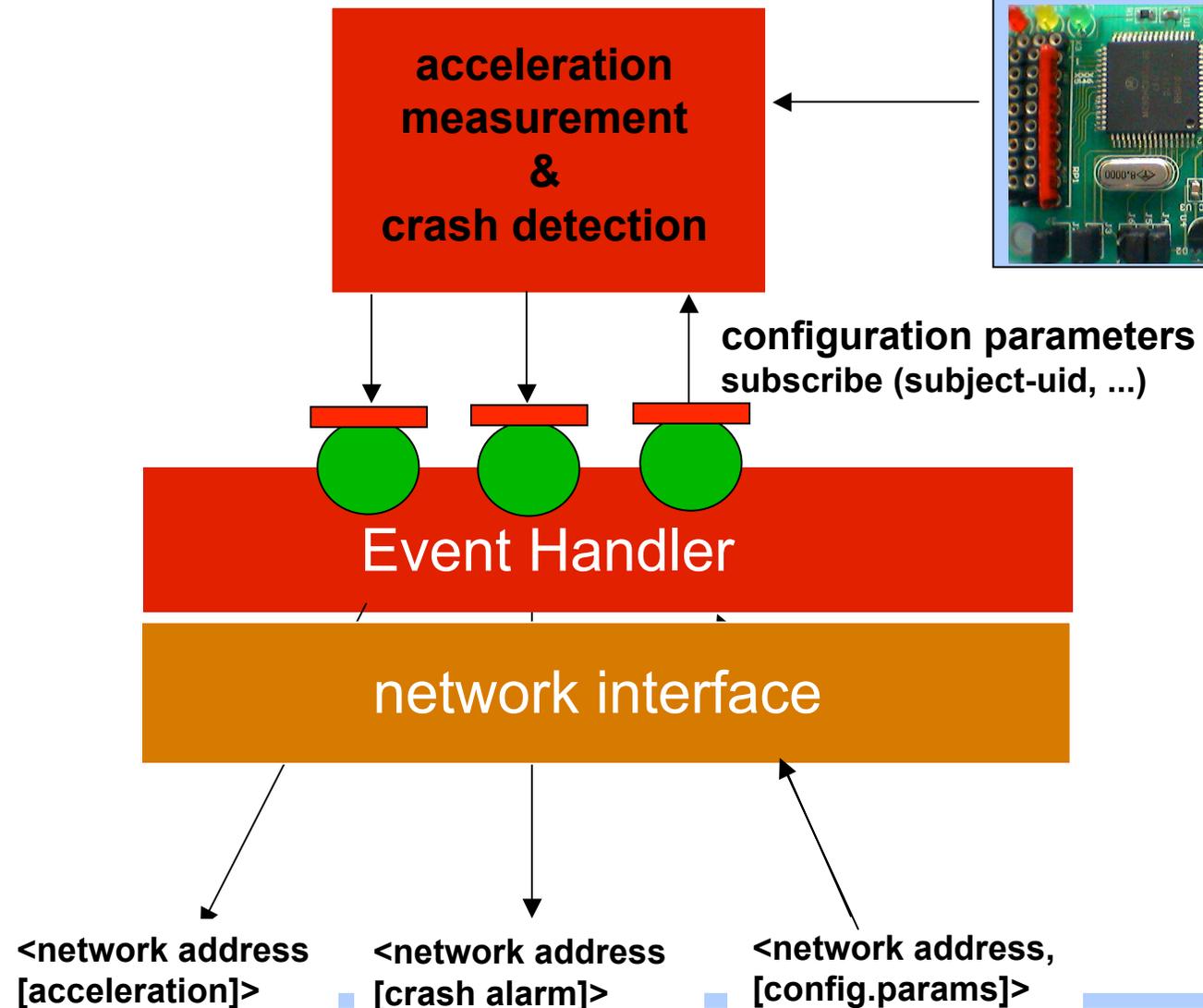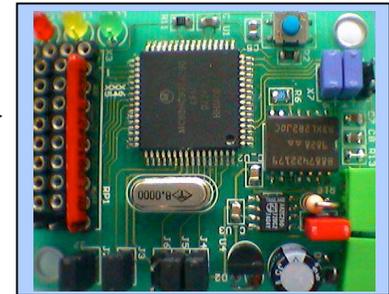
Overview

Abstraction	Space Coupling	Time Coupling	Flow Coupling
• Connected Sockets	Yes	Yes	Yes
• Unconnected Sockets	Yes	Yes	Consumer
• RPC	Yes	Yes	Consumer
• Oneway RPC	Yes	Yes	No
• async (Pull)	Yes	Yes	No
• async (Callback)	Yes	Yes	No
• Implicit Future	Yes	Yes	No
• Notications	Yes	Yes	No
(Observer Design Pattern)			
• Tuple Spaces (Pull)	No	No	Consumer
• Message Queues (Pull)	No	No	Consumer
• Subject-Based P/S	No	No	No
• Content-Based P/S	No	No	No

What are the options?

Communication model	Communication abstraction	Communication relation	Routing mechanism	Binding Time
message based	message	symmetric	address	design time
Remote procedure Call	invocation	client-server	address	design time
Distributed shared memory	memory cell	central	address	design time
Shared Data Spaces	object,tupel	central	contents	run time
Publish-Subscribe	event	Producer-consumer	contents/ subject	run time