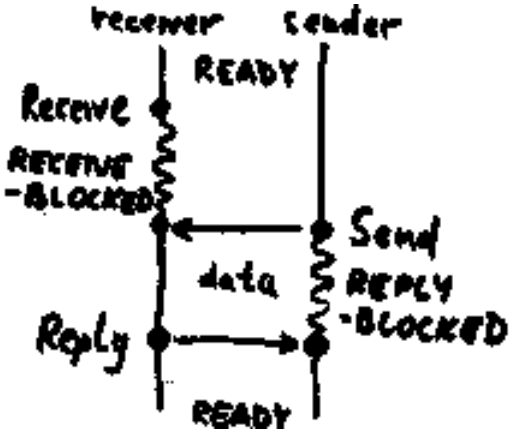
Spaces and Reactive Agents under QNX4

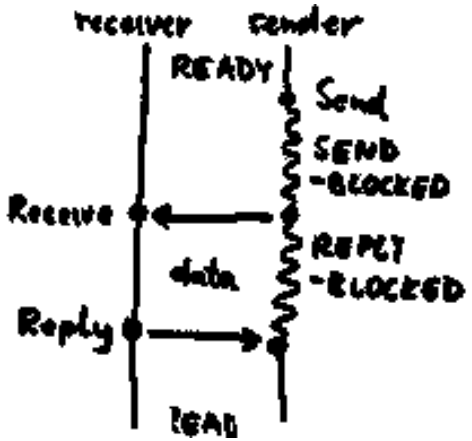
Andrej Lúčny
MicroStep-MIS

QNX Tools and Technologies,
Bratislava 2-4.12.2001

QNX4 IPC

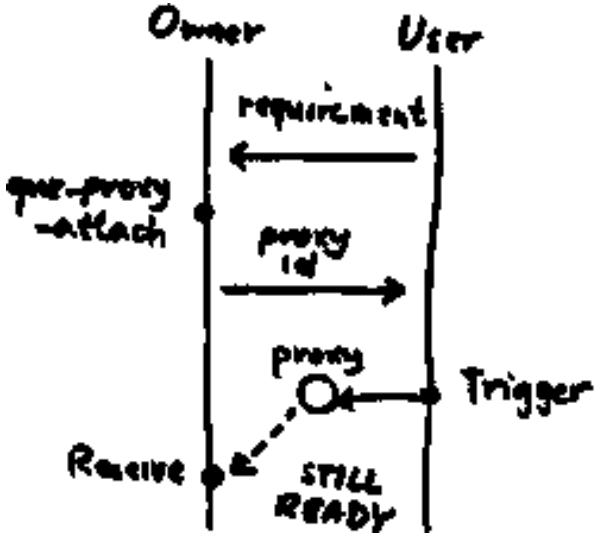
- **blocking message passing - great and almost exclusive means for data exchange and process synchronization**

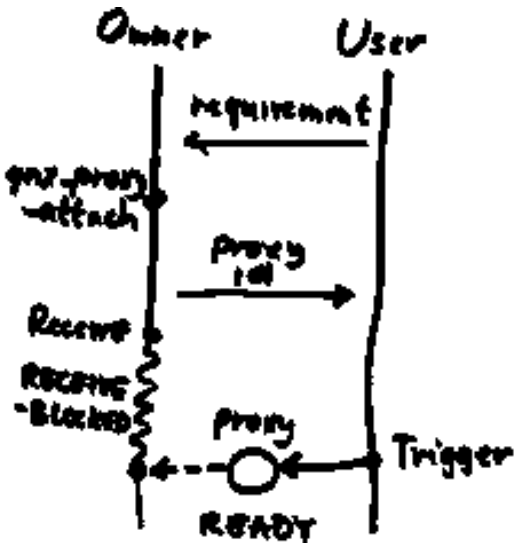




QNX4 IPC

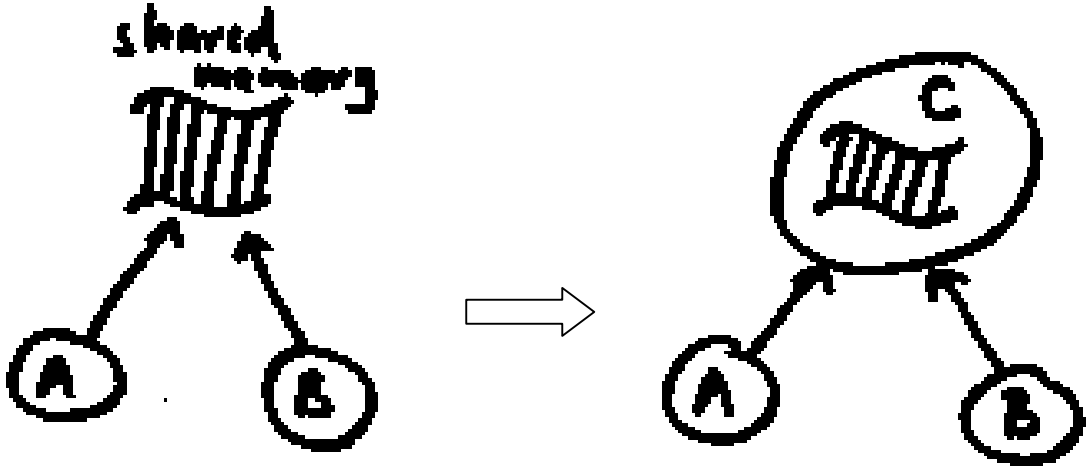
- **non-blocking communication by proxies - means to realize triggers**





QNX4 IPC

- **Shared memory and named pipes are recommend to be turned to message passing (Mqueue)**



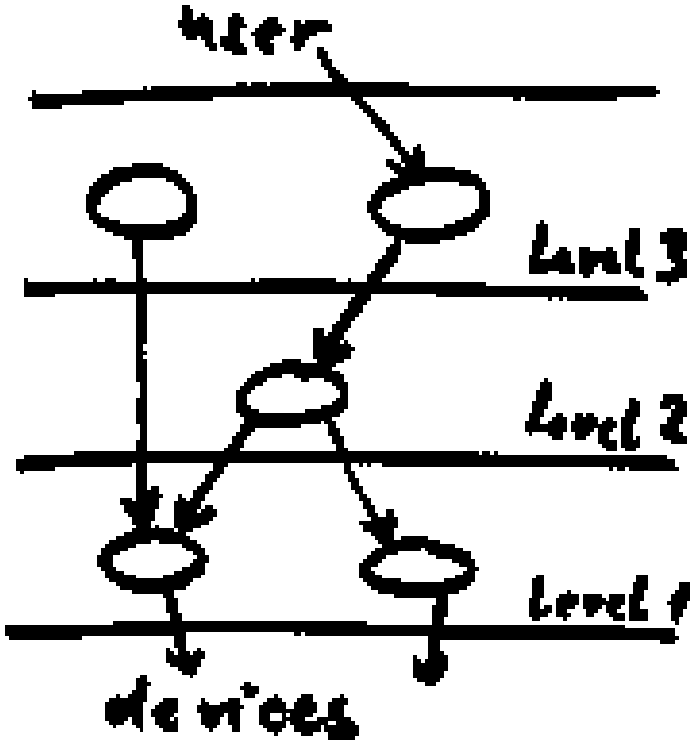
Deadlock problem

- **having couple of communicating processes without any order, deadlock can appear easily**

- **three possible solutions**
  - **to obey an architecture**
  - **to establish non-blocking message passing**
  - **both**

Pyramidal Client-Server Architecture

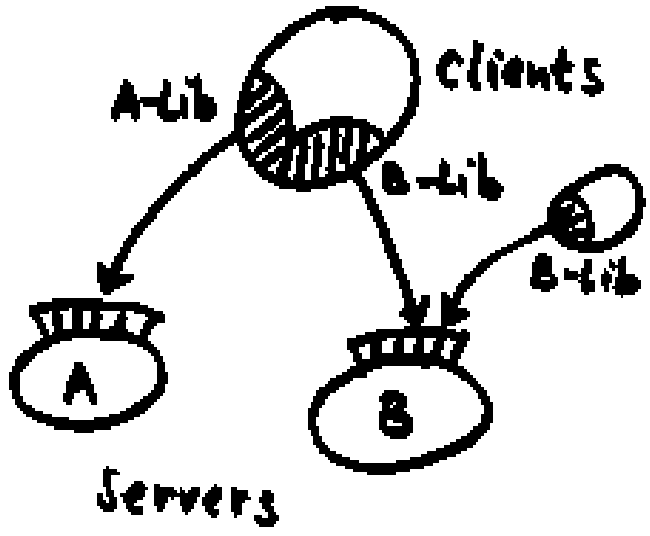
- Client-Server relation between each two communicating processes
- Client must be on higher level than its Server



Pyramidal Client-Server Architecture

- **many libraries for wrapping and marshalling**

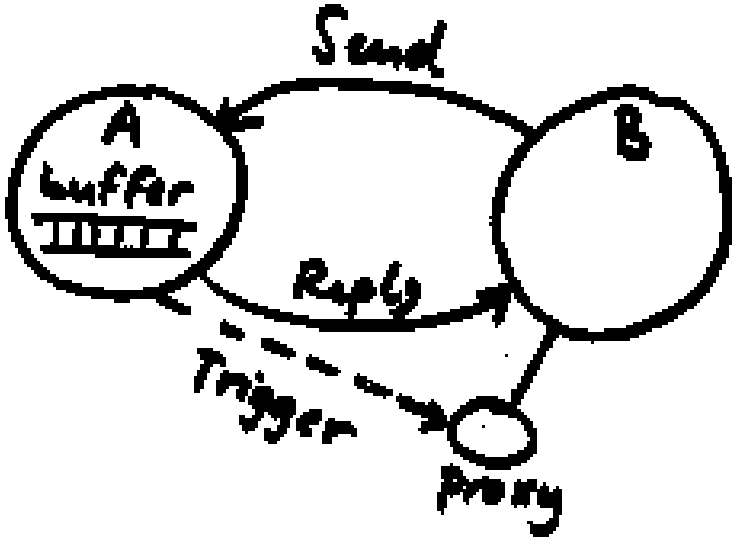
```
void  SayHello()
{
     MY_MSG  msg;
     strcpy ( msg.text, "hallo");
     Send (myfriend, &msg, &msg,
          sizeof(msg), sizeof(msg));
}
```

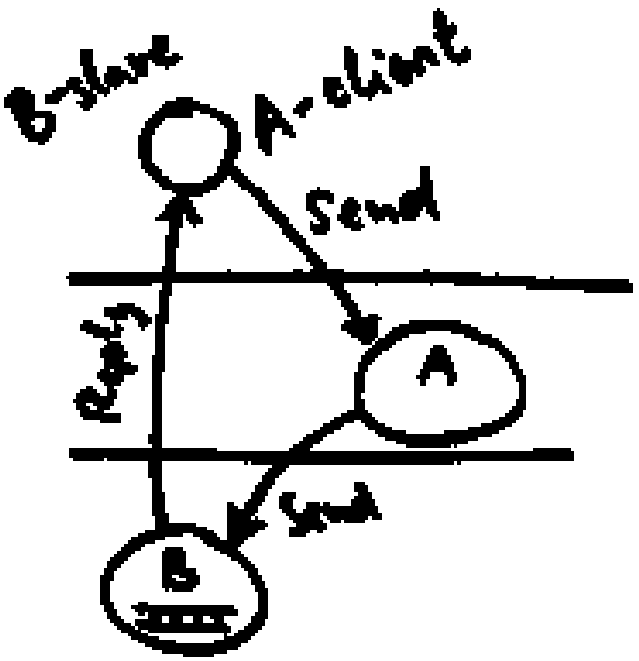


Pyramidal Client-Server Architecture

- **difficult design for cycled data flow**

message buffers, ferrymen, named pipes, ...

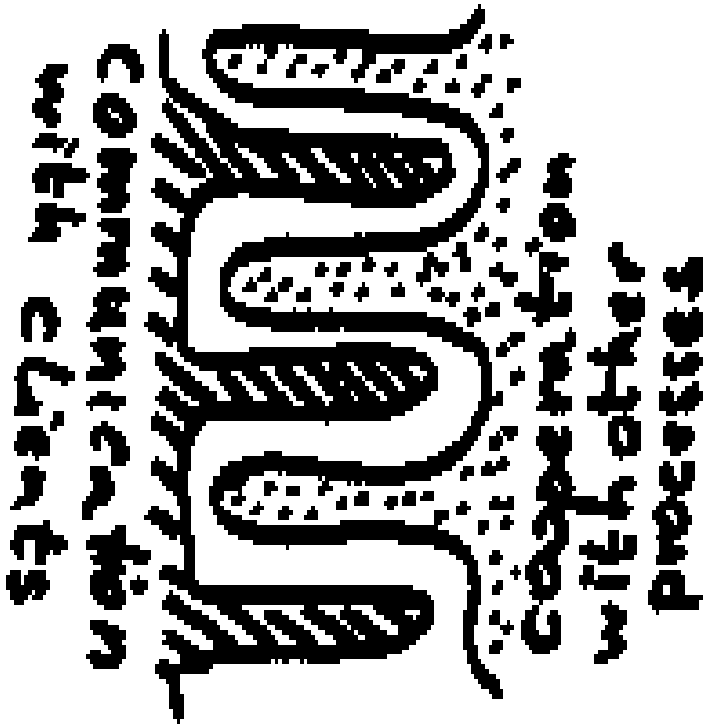




Pyramidal Client-Server Architecture

- **tangled code of any server (lack of threads)**

```
:
for (;;) {
    pid= Receive (0, &msg,...);
    switch (msg.action) {
        case 'QT':
            ... msg2...
    }
    Reply (pid, &msg2,...)
}
```



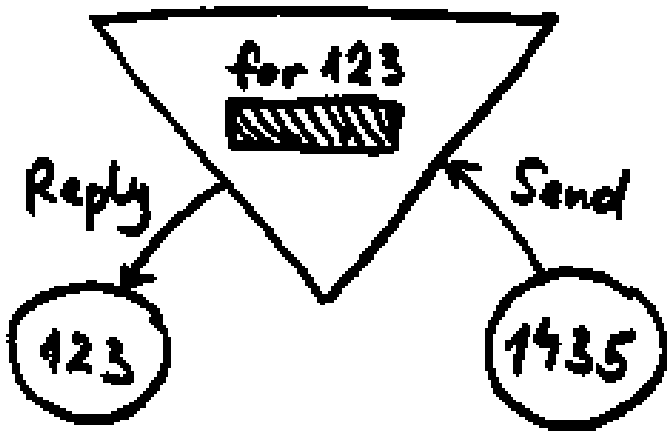
server must be able to provide services also during longer-term cooperation

Non-blocking message passing

- **can be established over blocking message passing by adding a process which is able receive a message from sender and store it until its recipient is ready to take it out.**

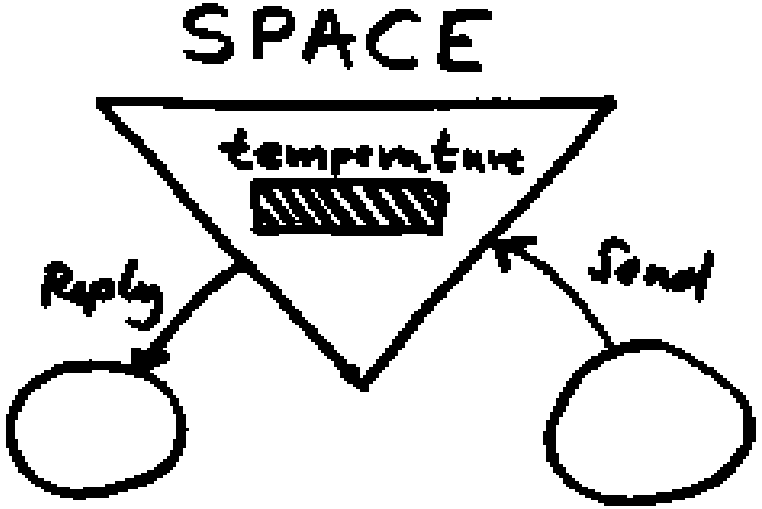
Non-blocking message passing

- **message stored in the added process is usually referenced by recipient id or channel name (address communication)**



Space

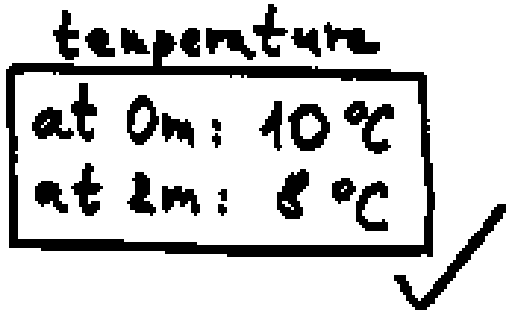
- **if the stored message is referenced by stigma of its content, the added process is called space (stigmergic communication)**



in space, a reference specifies not only data format, but also data content and meaning

Space

- **Place, where a message is stored in space, is called block**
- **block has a name which represents its content and it should contain only data which logically belong one to each other**

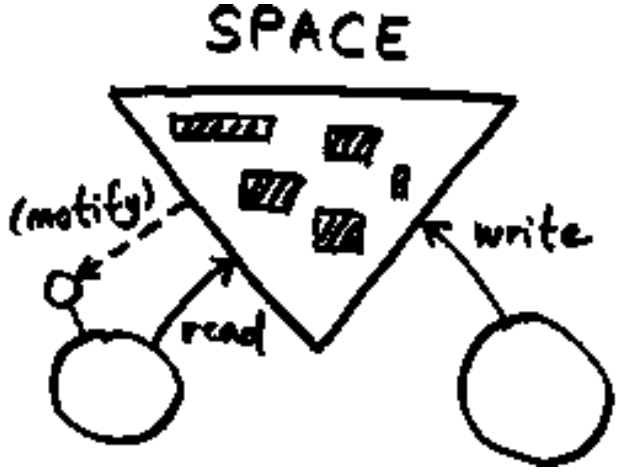


Space

- **space is a server providing to its clients**
  - **write to a block**
  - **read from a block**

  } **non-blocking operations**

  - **(notification that a block is changed)**

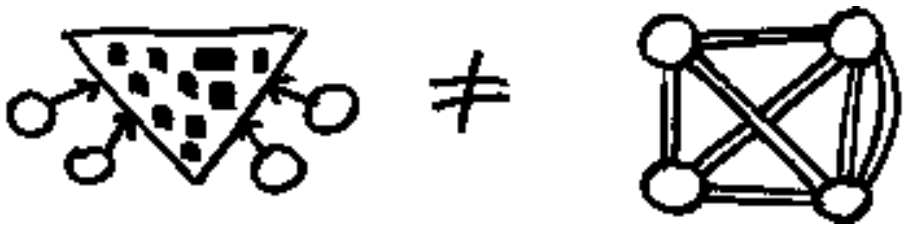


Space

- **block (unlike its content) does not depend on its writers and readers**
- **nobody has to create it**
- **it can be empty**
- **it can store message of arbitrary size**
- **it can be read before it is written**

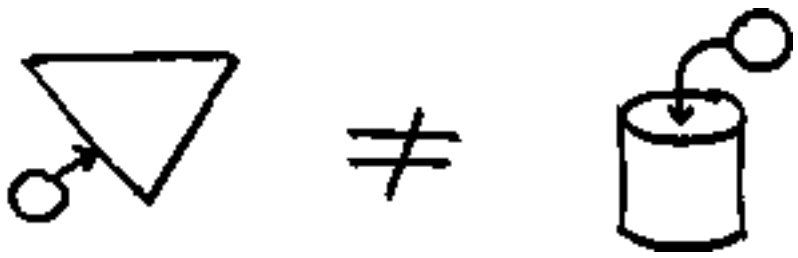
Space

- **usually, blocks are not queues, their content is overwritten by write operation**
- **their number corresponds to number of logical units which clients have dealt with**
- **consecutively, space is not a message queue**



Space

- **usually, read and write operation deals with just one particular block**
- **consecutively, space is not a database**



Space

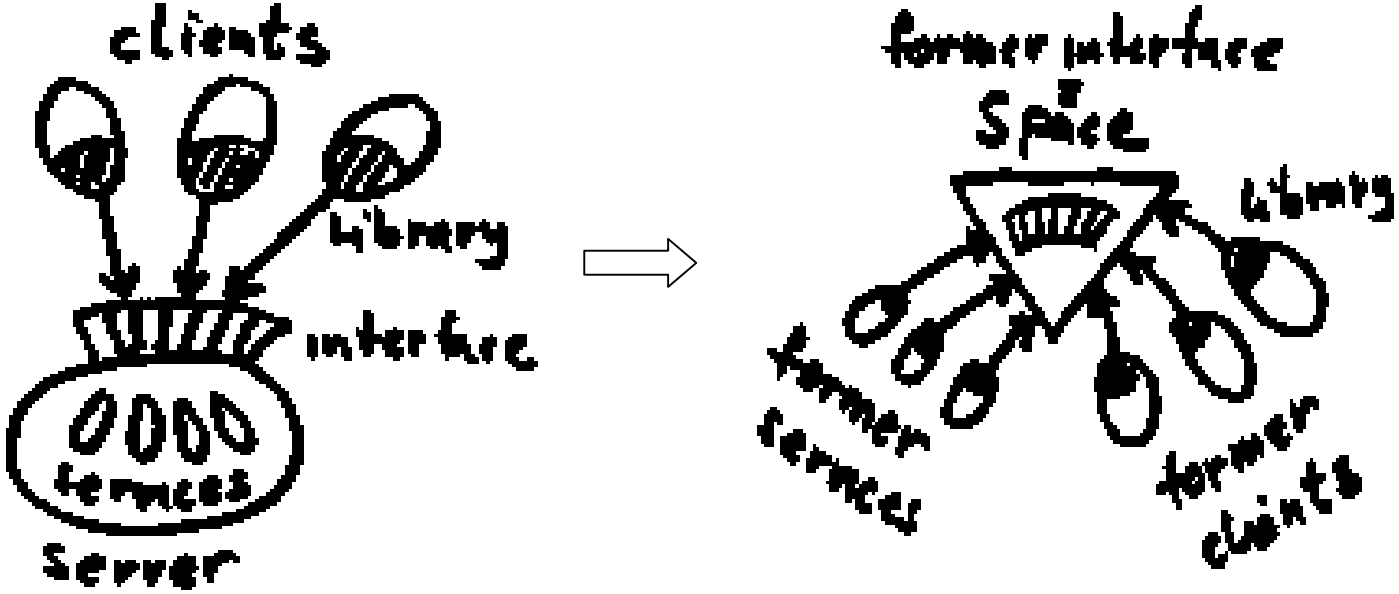
- content can be written to a block with a specified validity
- after its expiration, block becomes empty regardless somebody has read it or not

Space

- **is quite a difficult program relaying on sophisticated algorithms**
- **it must be powerful enough**
- **it must not contain serious errors**
- **BUT! it is same for all clients within all projects, so we can concentrate on it to meet all these requirements**

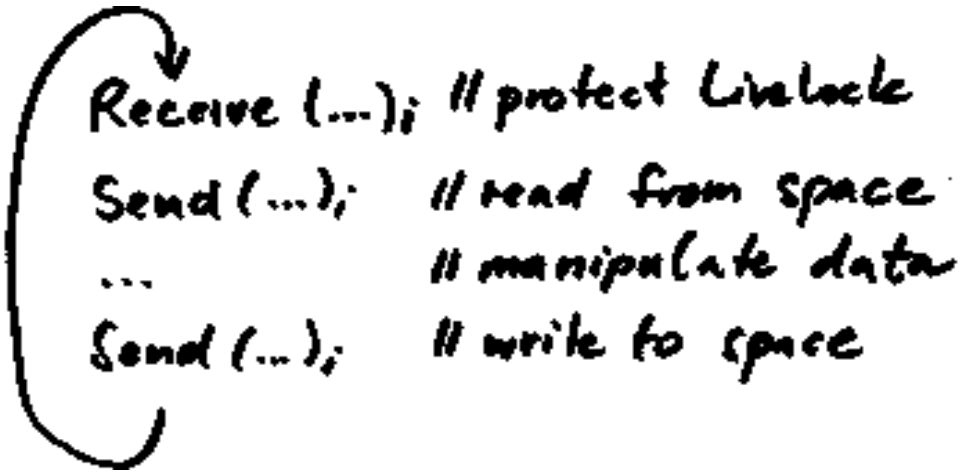
Space

- **can be taken as result of a server decomposition: it corresponds to that part of the decomposed server which realizes communication with clients**



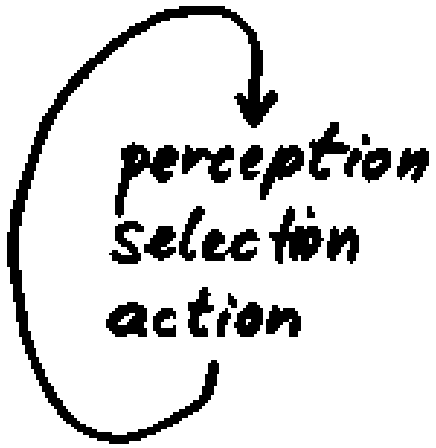
Reactive Agent

- **By this decomposition, the former clients and codes related to the former services become much simpler and can get a structure:**



Reactive Agent

- **In this way we have met concept of reactive agent, what is a process which regularly selects and performs actions as reaction to perception of its environment.**



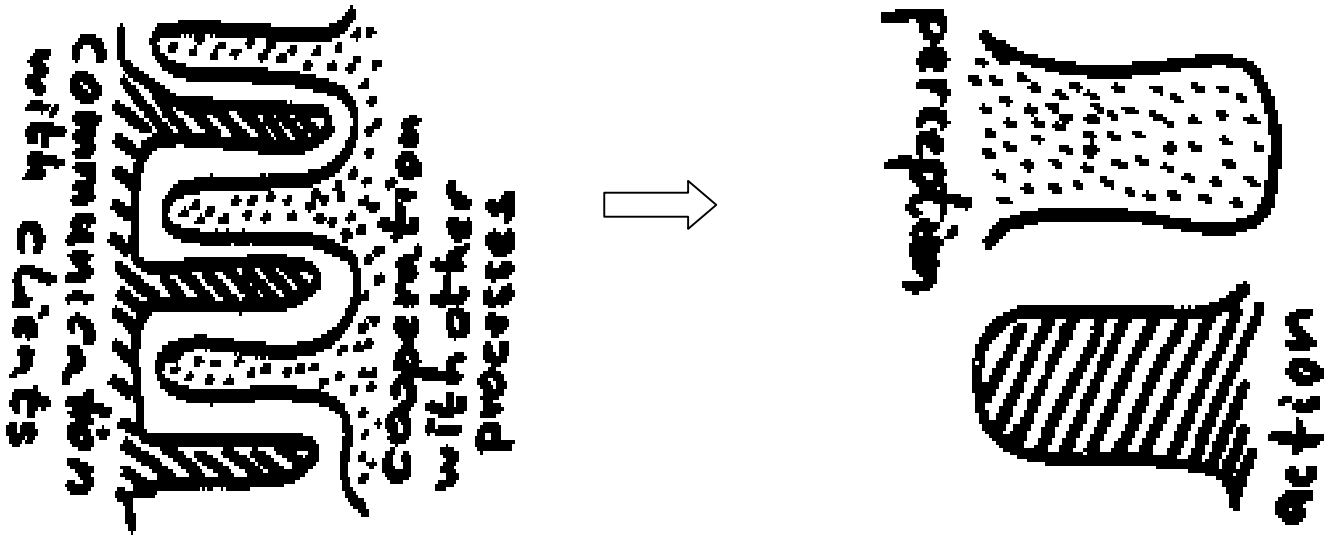
reactive agent pursue a goal built-in its reactions

Reactive Agent

```
void main ()
{
    // initialization
    for (;;) {
        Receive (proxy,...);                // timer or trigger
        ReadFromSpace('a',&a); ...// perception
        ... Compute b form a ...            // selection
        WriteToSpace('b',&b); ...    // action
    }
}
```

Reactive Agent

- **reactive agent is simple enough to write it within one thread without tangling of code**



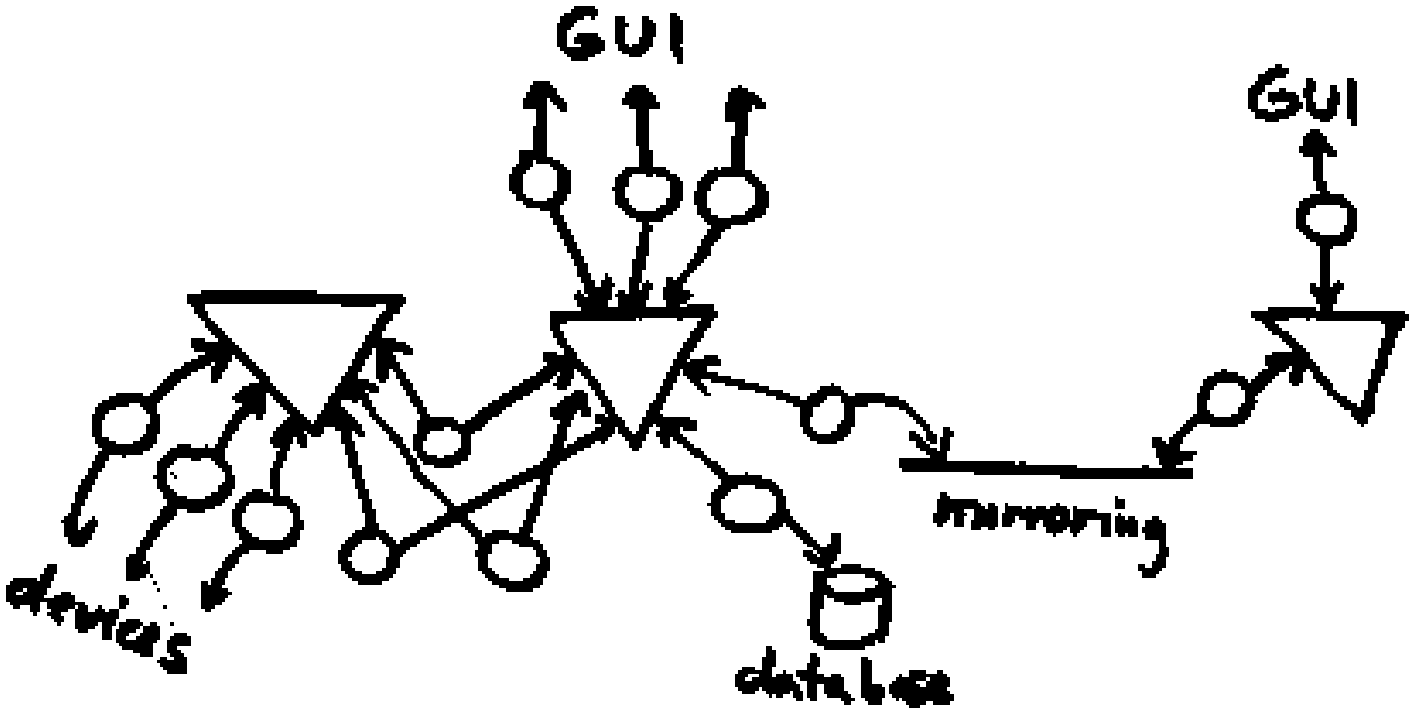
Reactive Agent

- each reactive agent uses only one library for inter-process communication
- the library is quite simple
- in this way wrapping is normalized
- the norm is very compact

Agent-Space Architecture

- **only two kinds of processes are allowed within a system: spaces and reactive agents**
- **all space processes correspond to the same program**
- **any code related to application domain is concentrated in reactive agents**

Agent-Space Architecture



Agent-Space Architecture

- **Advantages:**
  - **easy to design system**
  - **easy to code agents**
  - **easy to modify system**
  - **easy to start system**
  - **easy to restart any agent**
  - **easy to recover from errors in agents**
  - **normalization of communication interfaces**

Agent-Space Architecture

- **Disadvantages:**
  - **less efficient solution**
  - **spaces must be reliable**
  - **communicated data can be potentially lost**
    (in practice, it is overcome by real-time)
    (on the other hand, it supports real-time)
  - **no profit from threads**

Agent-Space Architecture

- **agent-space architecture and real-time operating system support each other**
- **the idea is very suitable mainly for QNX4 where we have no threads**
- **Under QNX4 the idea is already applied on data-server for slinger (SSI technology)**
- **possible extensions: normalization of marshalling, representation languages, XML, mobile code**

Agent-Space Architecture

- **Applications:**
  - **Technology**
    - **monitoring systems**
    - **control systems**
    - **simulators**
  - **Science**
    - **simulation of any reactive behavior**
    - **mobile robotics**
    - **computational etology**

Thank you!

Spaces and Reactive Agents under QNX4

Andrej Lúčny
MicroStep-MIS

QNX Tools and Technologies,
Bratislava 2-4.12.2001

www.microstep-mis.sk

andy@microstep-mis.sk