Managing Hard Real Times
(28 Years Later)

Peter Welch (**phw@kent.ac.uk**)

CPA 2015, University of Kent, 24th. August, 2015

Copyright P.H.Welch

Hard Real Times

Received wisdom about ***hard real-time*** systems ***(where lateness in response means system failure)*** is that a currently running process must be ***pre-empted*** by a higher priority process that becomes runnable (e.g. by the assertion of a processor event pin or timeout). Otherwise worst-case response times cannot be guaranteed.

Copyright P.H.Welch

Hard Real Times

Further, if a higher priority process needs to synchronise with one of lower priority, the latter must automatically *inherit* the priority of the former. If this does not happen, the opposite happens and the former effectively inherits the lower priority of the latter as it waits for it to be scheduled *(priority inversion)* — again, worst-case response times fail.

Copyright P.H.Welch

Hard Real Times

The CCSP multicore scheduler for occam-pi (part of the KRoC package) is, *possibly*, the fastest and most scalable (with respect to processor cores) multicore scheduler on the planet. [Some say … ☺]

However, its scheduling is *cooperative* (not *pre-emptive*) and it does not implement *priority inheritance* (and *cannot do so*, given the nature of CSP synchronisation, where processes do not know the identities of the other processes involved).

Therefore, despite its performance, received wisdom would seem to rule it out for hard real-time applications. [☹]

Copyright P.H.Welch

Hard Real Times (28 years later)

This talk reviews a paper from OUG-7 proceedings (1987) that discusses these ideas with respect to Transputers. No change is needed for modern multicore architectures. [☺]

Peter H. Welch. Managing Hard Real-Time Demands on Transputers. In: Traian Muntean, ed., *Proceedings of OUG-7 Conference and International Workshop on Parallel Programming of Transputer Based Machines*. LGI-IMAG, Grenoble, France: IOS Press. 1987.

One minor fix, that simplifies the logic and improves behaviour, can be made. [☺]

Copyright P.H.Welch

Hard Real Times

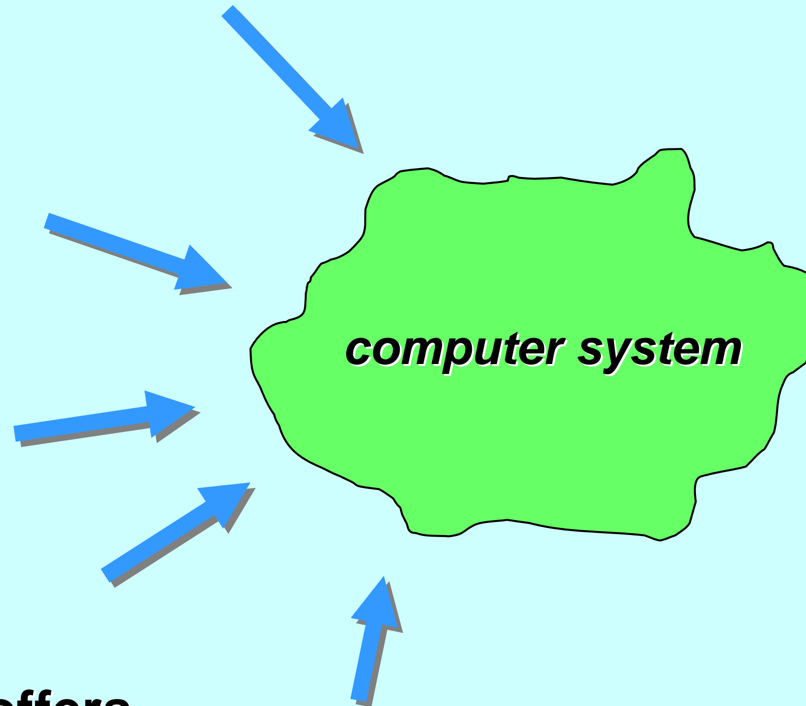
Let's start with Conclusions:

- **pre-emptive scheduling is *not required* for hard real-time; [☺]**
- **priority inheritance is a *design error* (dealt with by correct design, not the run-time system); [☺]**
- **the occam-pi/CCSP scheduler can be made to work *even more efficiently* for hard real-time systems than it presently does for soft real-time (e.g. complex system modelling). [☺]**

Hard Real Times

real world

real world



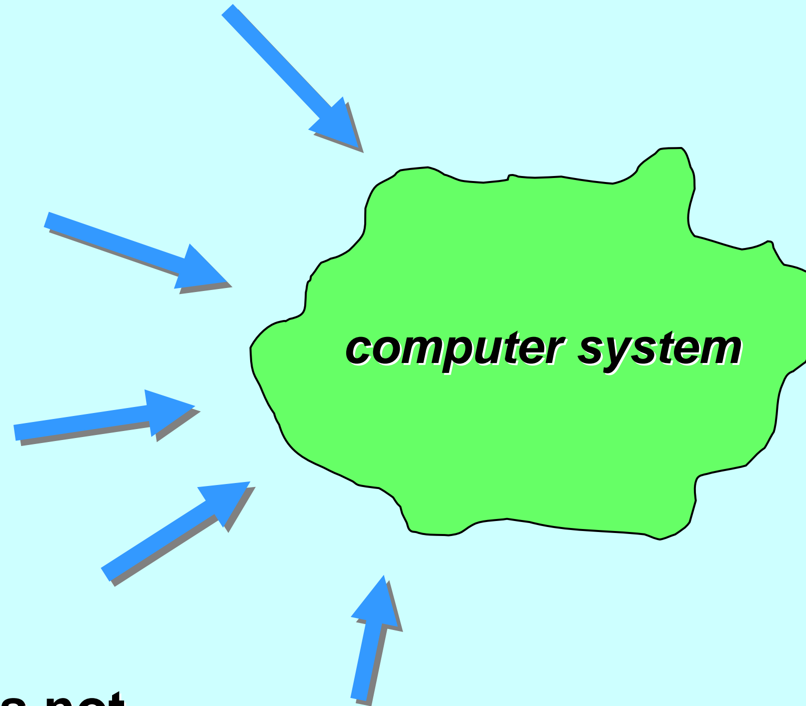
The real world offers lots of information, some of which the computer system needs to gather.

real world

Hard Real Times

real world

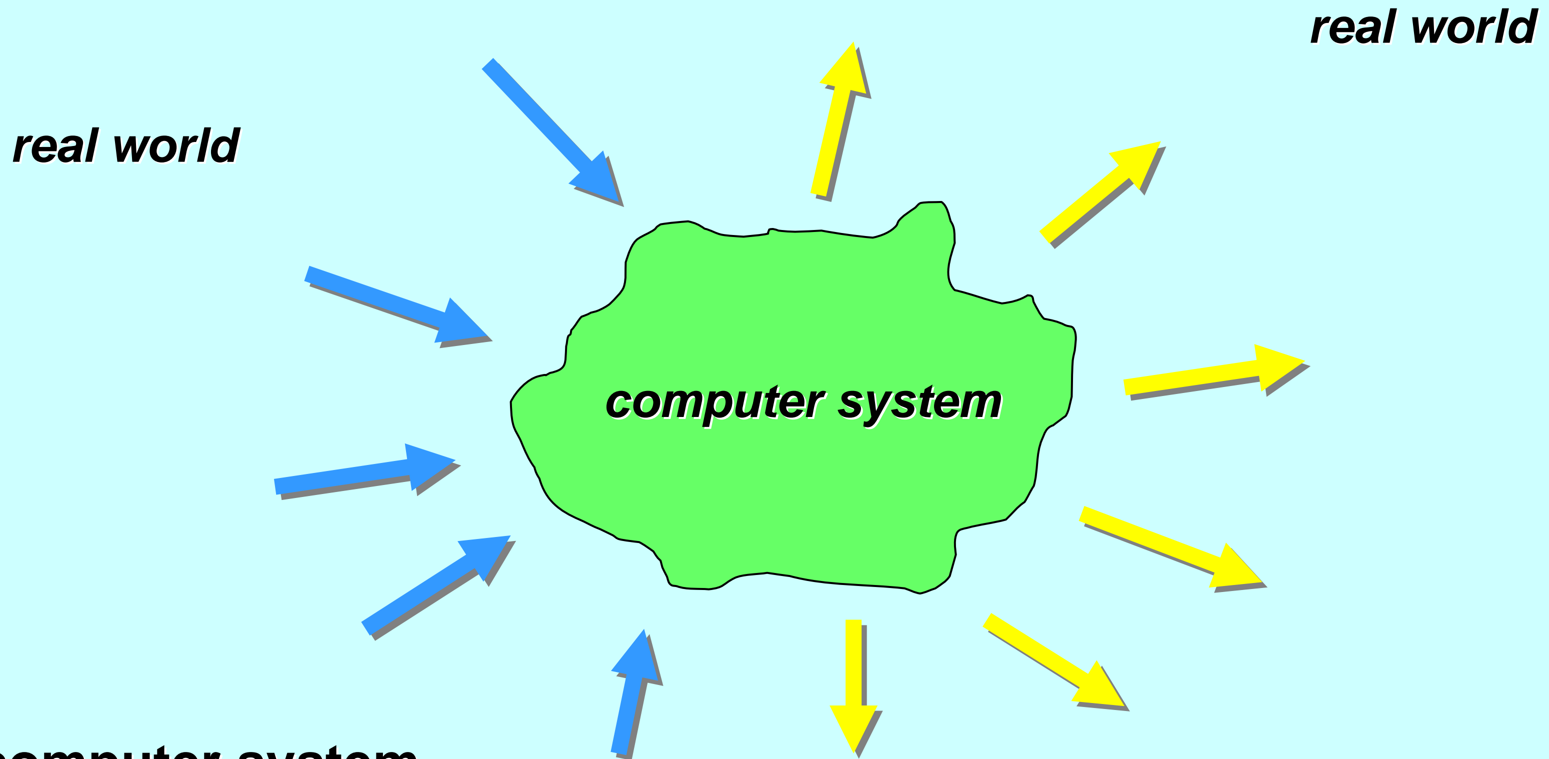
real world



real world

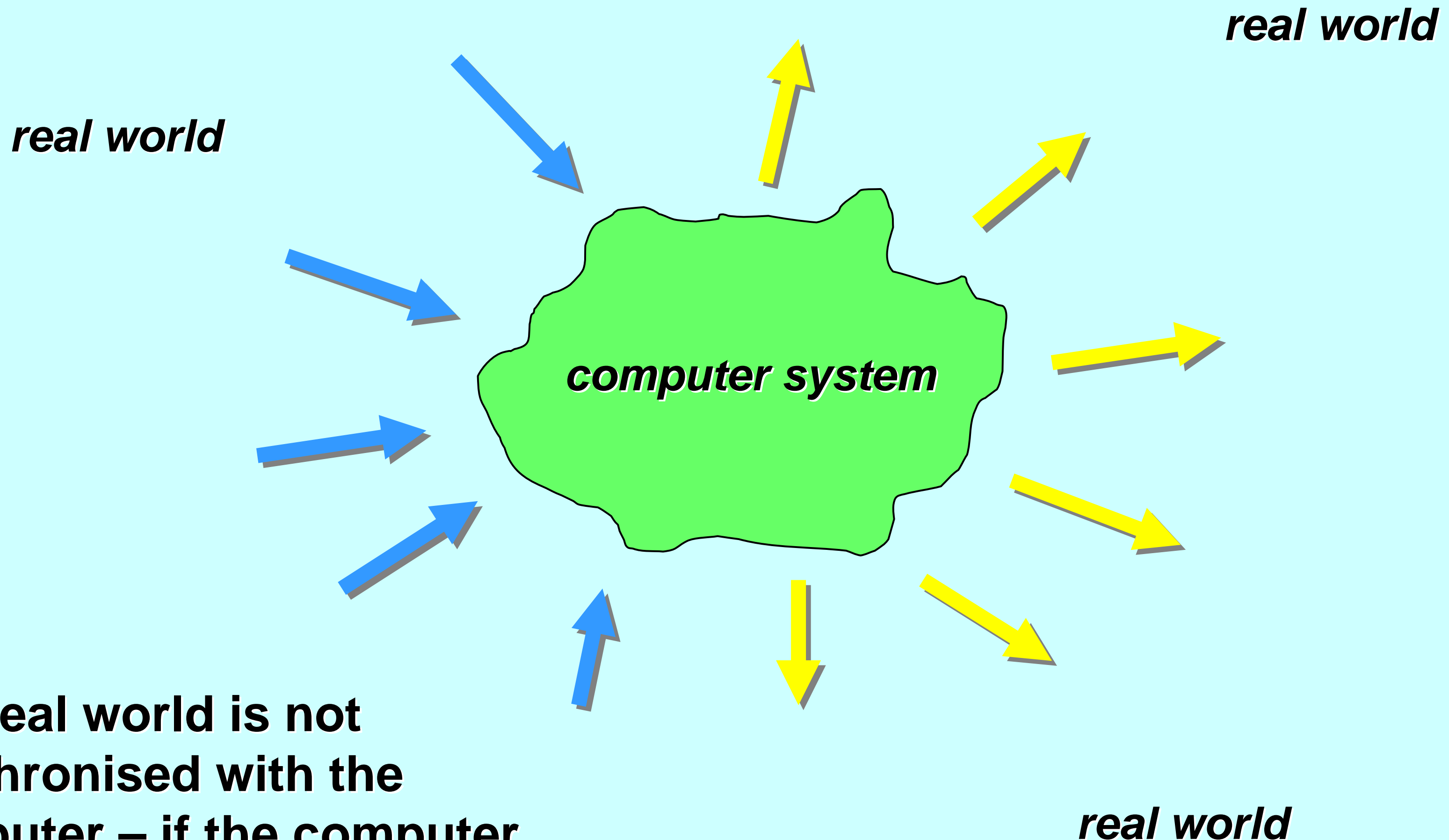
The real world is not synchronised with the computer – if the computer does not take the information, the real world does not freeze!

Hard Real Times



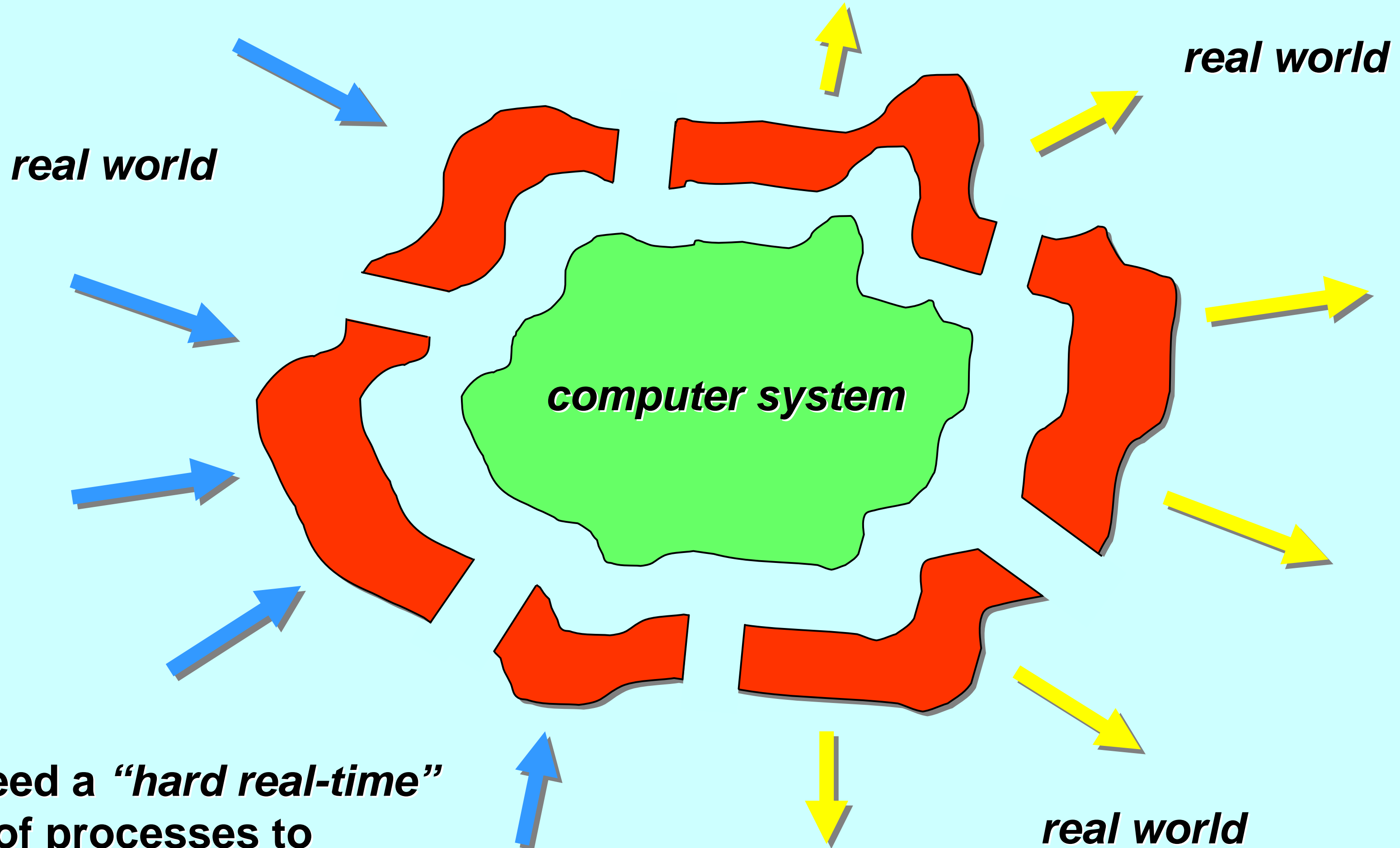
The computer system offers information to the real world, in an attempt to control a small part of it.

Hard Real Times



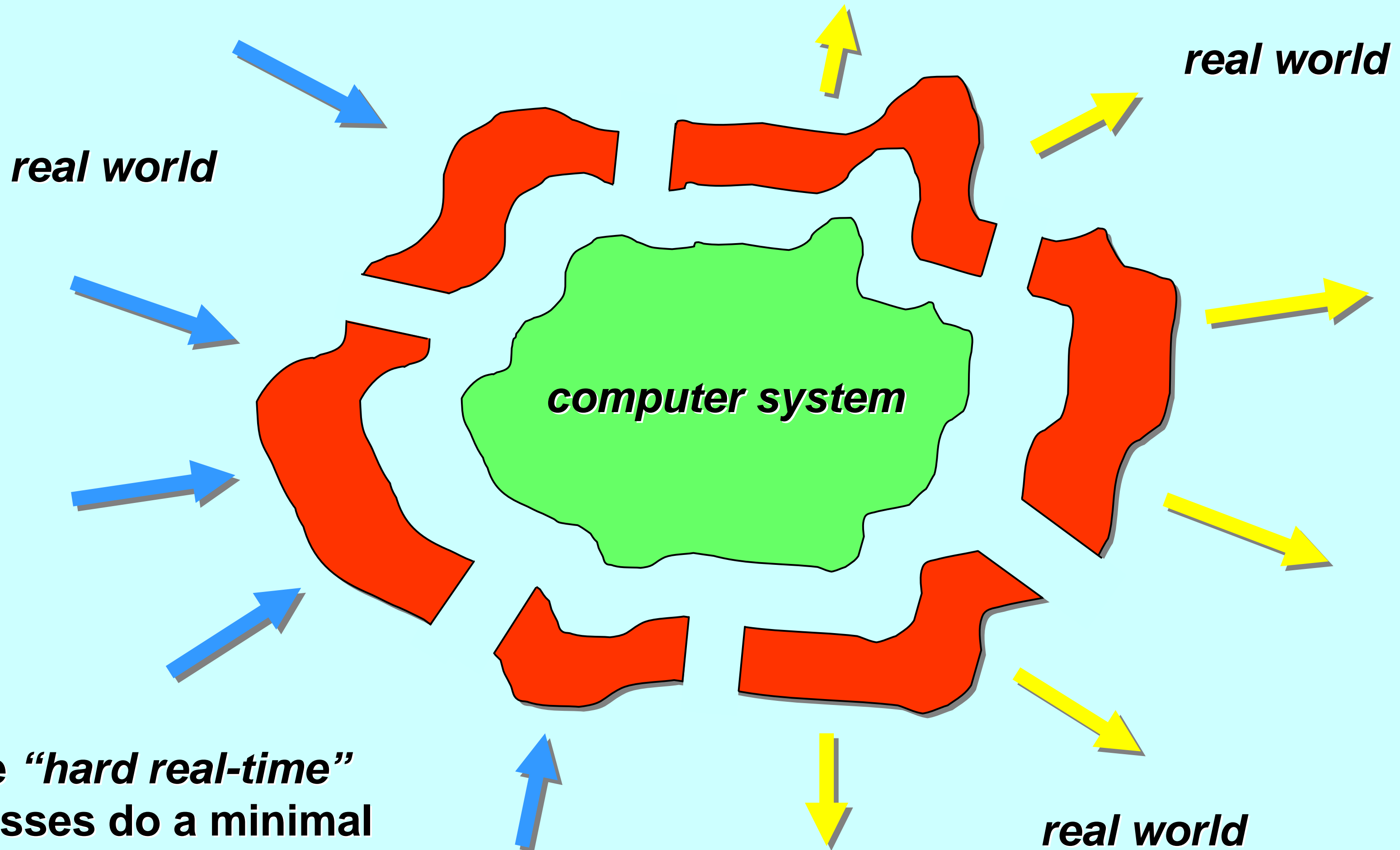
The real world is not synchronised with the computer – if the computer is late with the information, the real world does not wait!

Hard Real Times



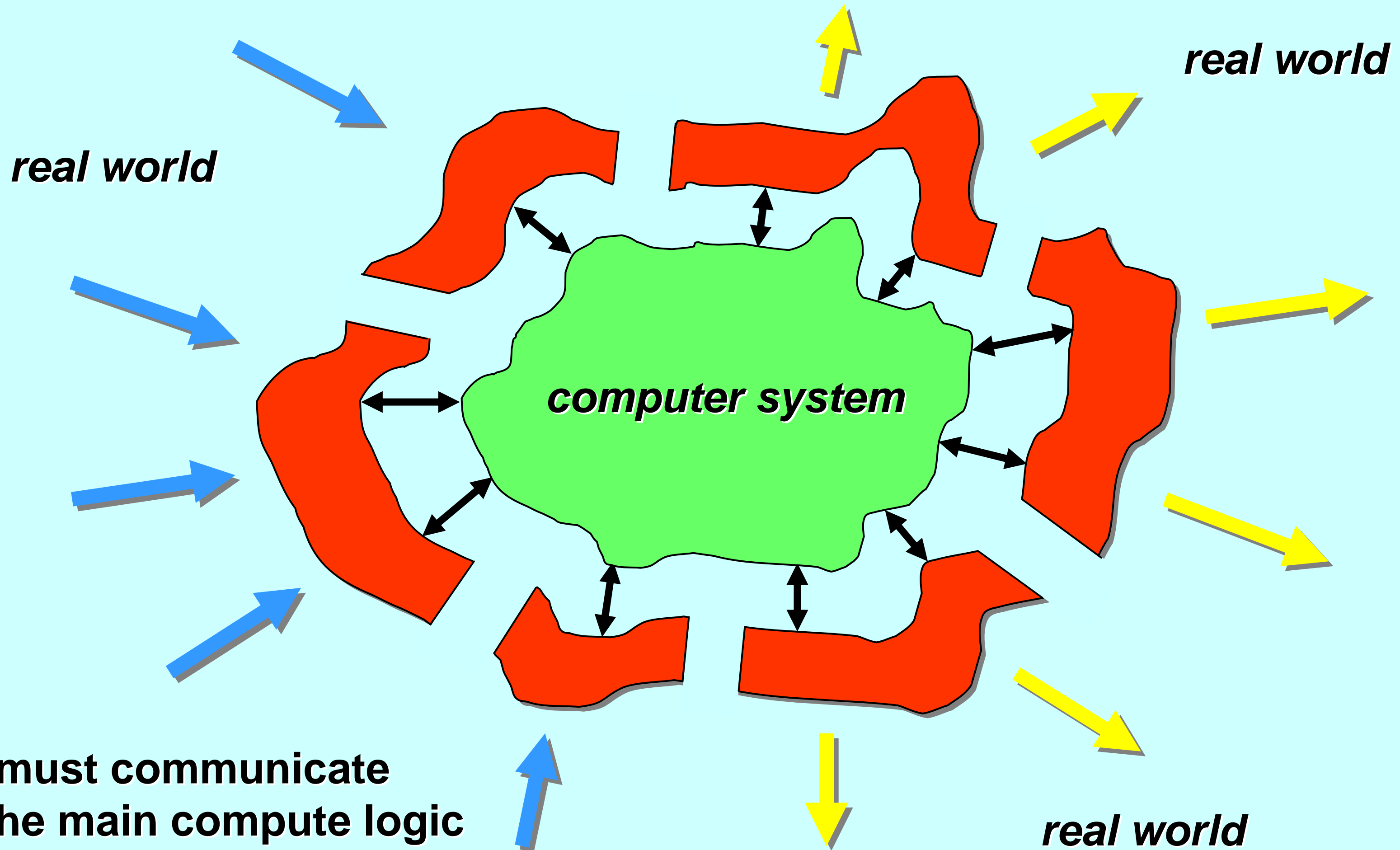
We need a *"hard real-time"* shell of processes to interface between the real world and the main compute logic.

Hard Real Times



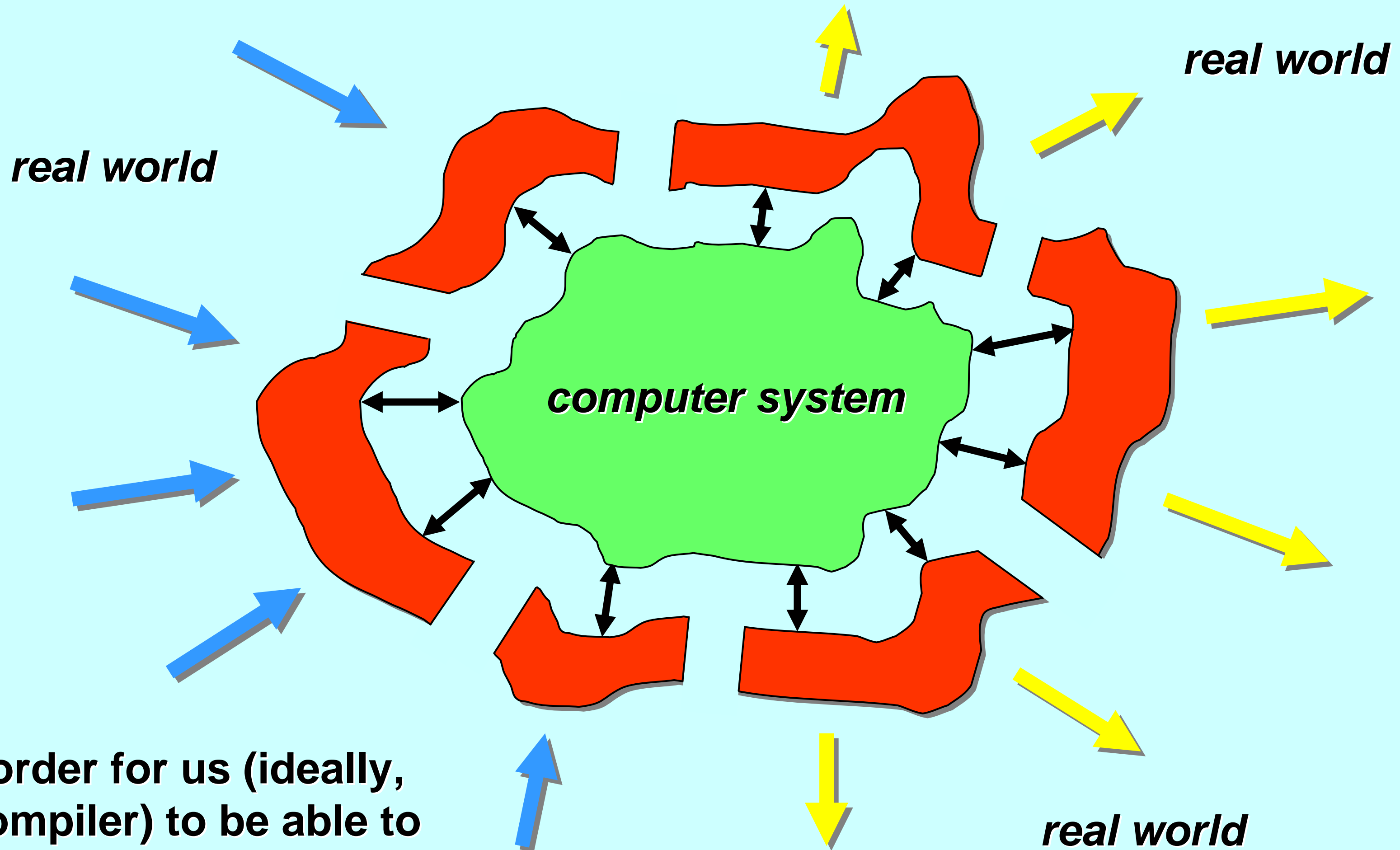
These *"hard real-time"* processes do a minimal amount of work: data gather, lose data (deliberate), keep records, actuate and emergency actuate.

Hard Real Times



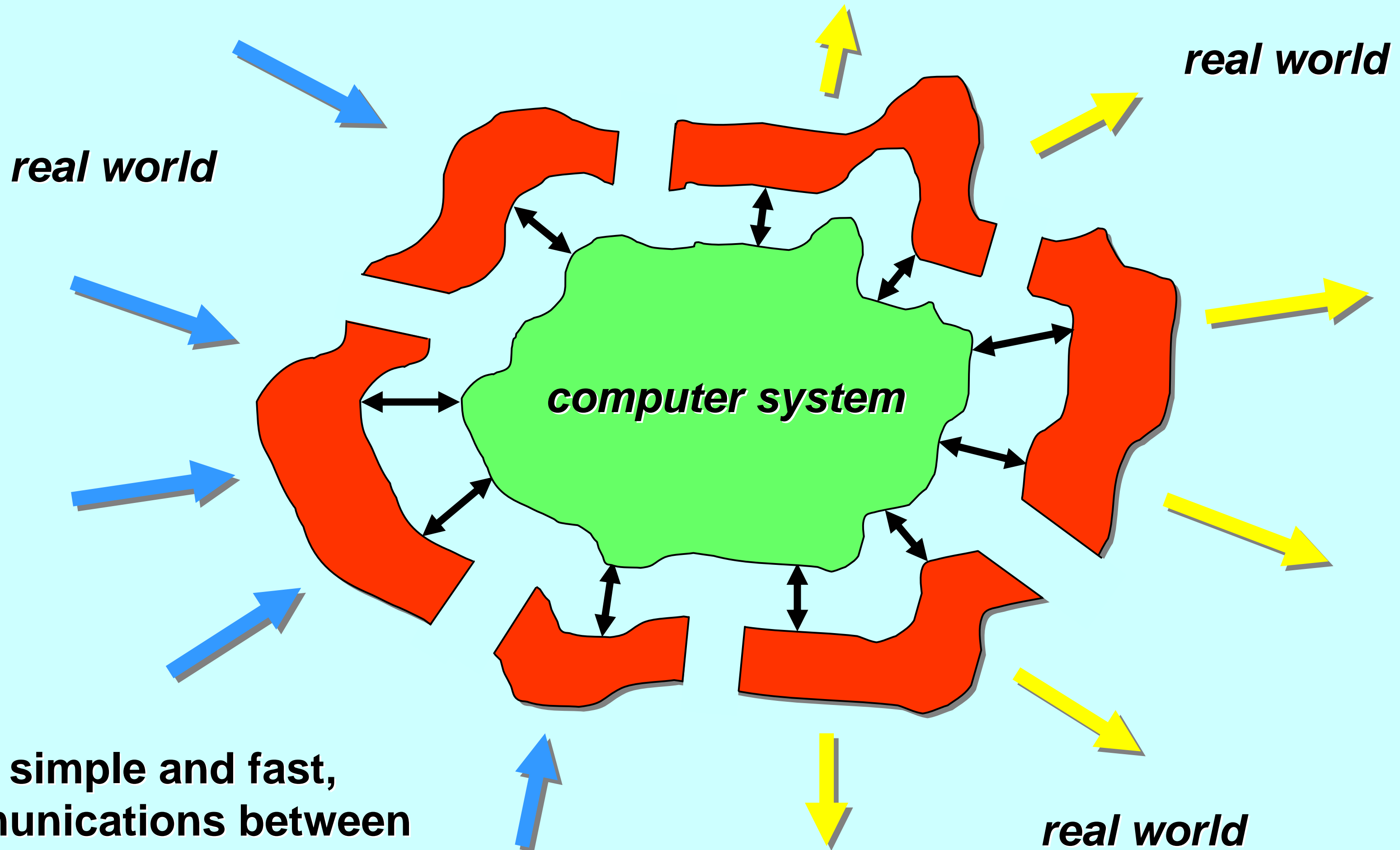
They must communicate with the main compute logic without getting blocked (like the *"real world"*) …

Hard Real Times



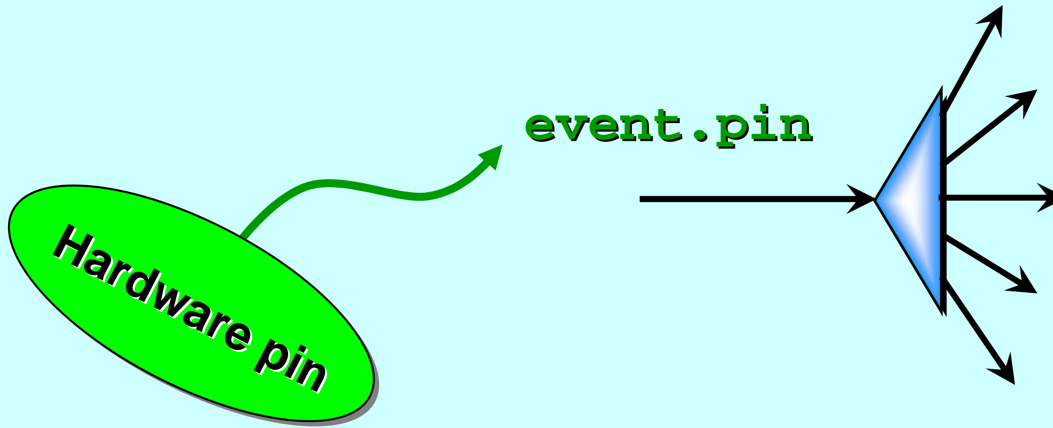
… in order for us (ideally, the compiler) to be able to compute *worst-case* response times to clear the sensors and drive the actuators.

Hard Real Times



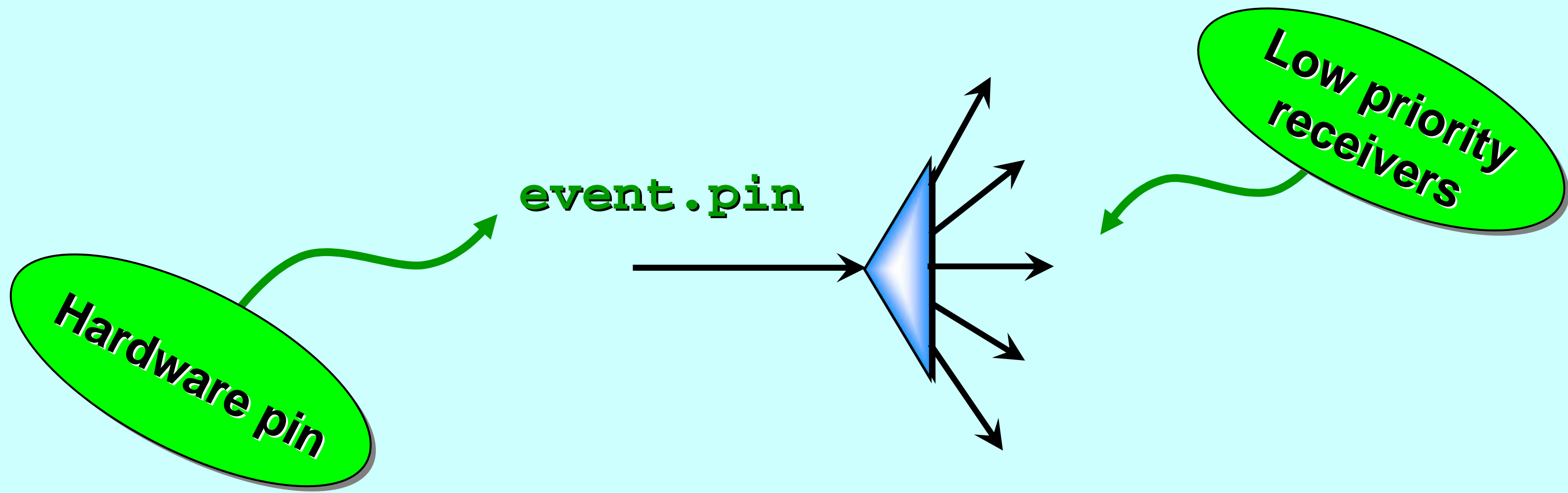
To be simple and fast, communications between *"hard real-time"* processes and the main compute logic are (CSP) synchronised. ☺ ☺ ☺

Event Manager



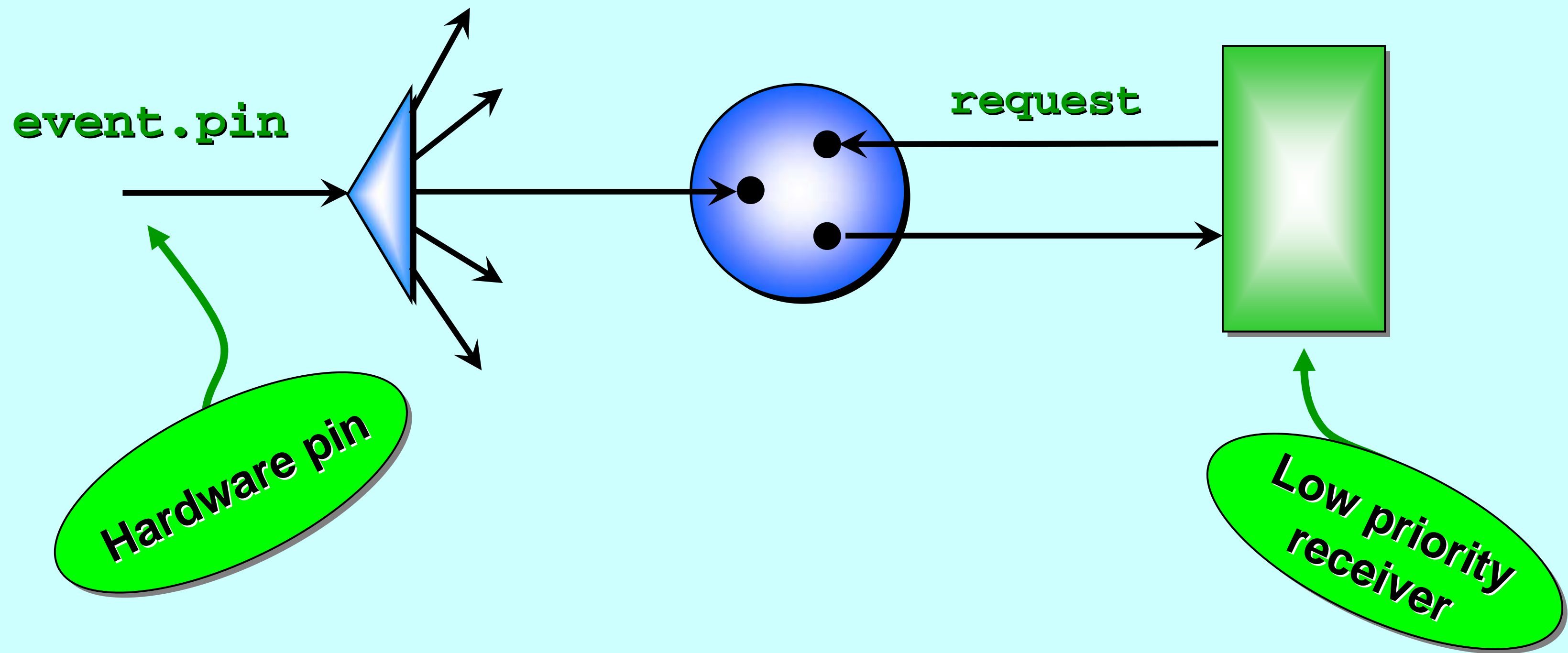

```
PROC event.manager (CHAN SIGNAL event.pin?, []CHAN BYTE out!)
  WHILE TRUE
    SEQ
      SIGNAL any:
      event ? any
      ...   find who pulled the pin (from status registers)
      ...   extract data (from data register 'i')
      out[i] ! data
:
```

Event Manager

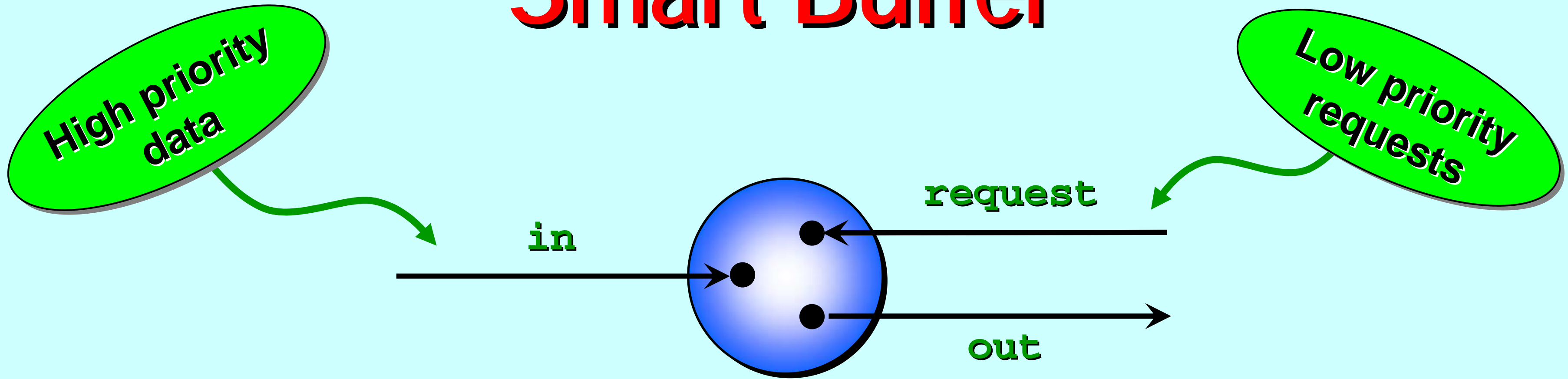


- Must guarantee *never* to miss an assertion of the event pin.
- Worst case: an event signal has just been taken when another arrives. We have to find who was responsible for the *just-taken* signal, extract the relevant data and send to the relevant server process for that signal. How long for this?
- Need to have a max delay for outputting the data. ✗✗✗

Event Manager & Smart Buffer

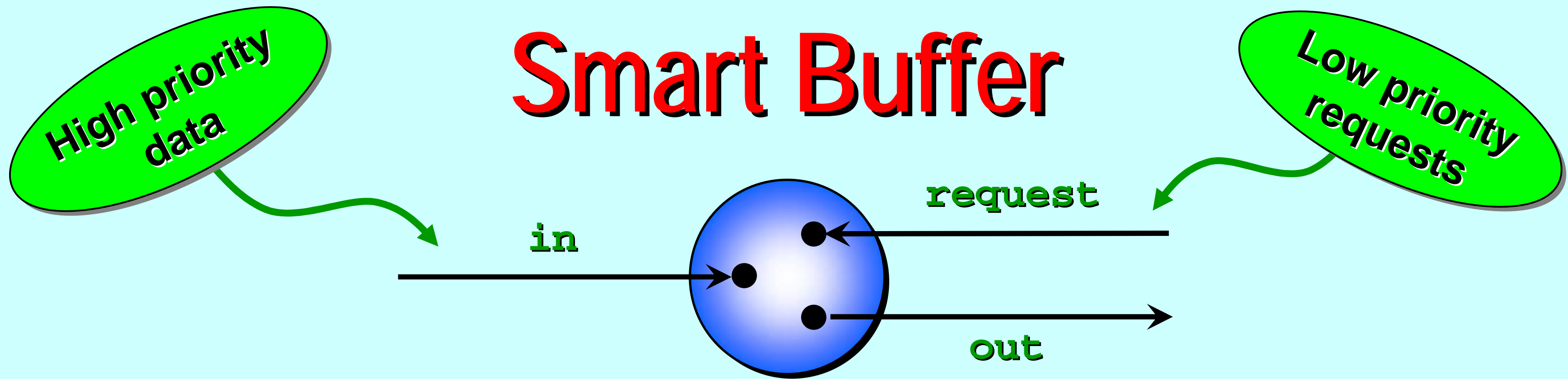


Smart Buffer



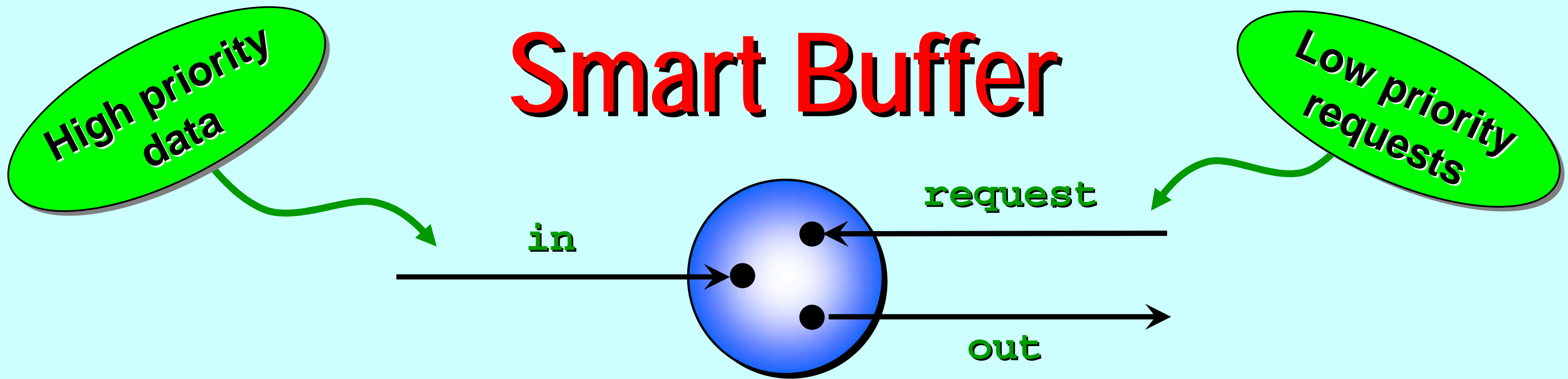

```
PROTOCOL TAGGED.BYTE IS INT; BYTE:    -- n.missed; data

PROC smart.buff (CHAN BYTE in?, CHAN BOOL request?,
                 CHAN TAGGED.BYTE out!)
  ...  local variabes
  WHILE TRUE
    PRI ALT
      ...  deal with an input
      ...  deal with a request
:
```

Smart Buffer



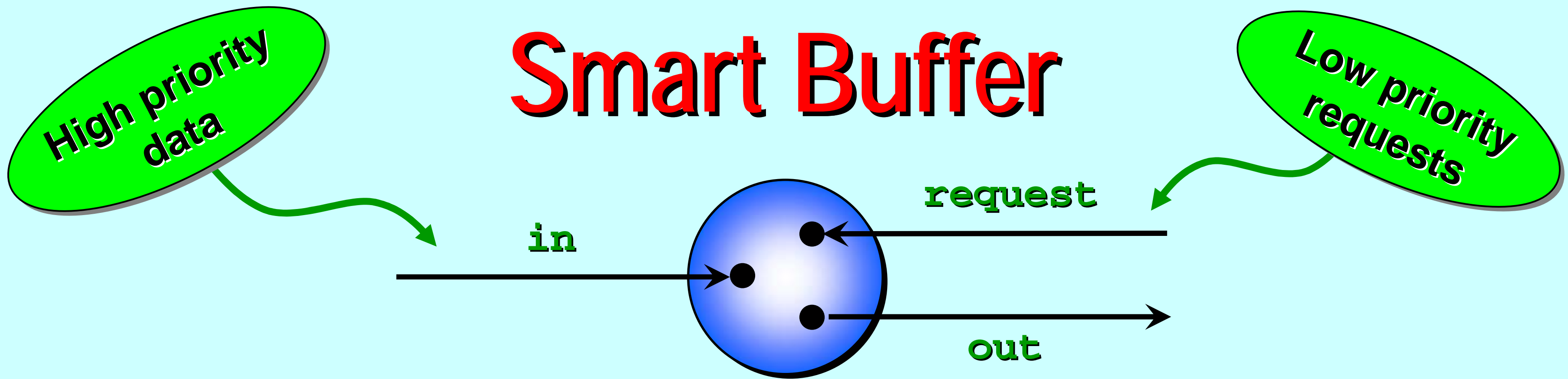

```
PROC smart.buff (CHAN BYTE in?, CHAN BOOL request?,
                 CHAN TAGGED.BYTE out!)
  INITIAL INT n.missed IS 0:
  INITIAL BOOL loaded IS FALSE:
  BYTE data:
  WHILE TRUE
    PRI ALT
      in ? data
        ...  process the data
      BOOL any:
      loaded & request ? any
        ...  process the request
:
```

Smart Buffer



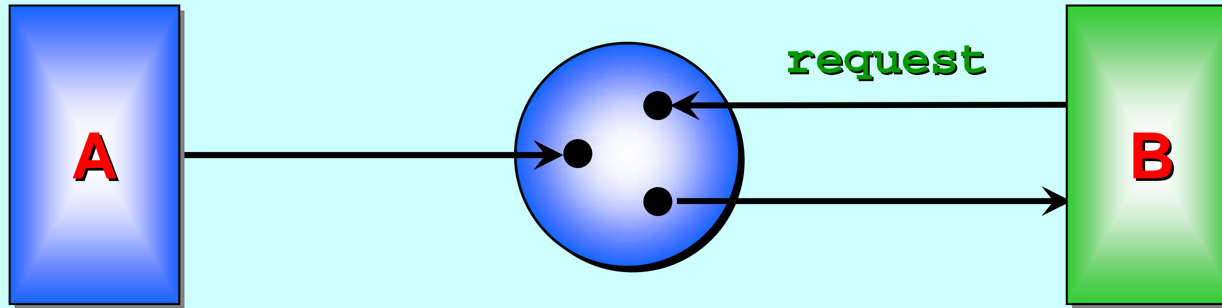

```
WHILE TRUE
  PRI ALT
    in ? data
      IF
        loaded
          n.missed := n.missed + 1
        NOT loaded
          loaded := true
    BOOL any:
    loaded & request ? any
      ...  process the request
```

Smart Buffer



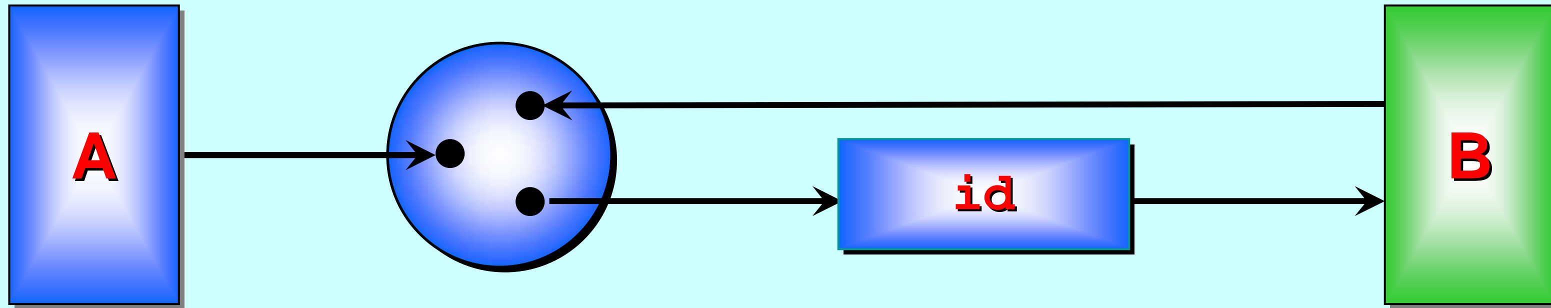

```
WHILE TRUE
  PRI ALT
    in ? data
      ...  process the data
    BOOL any:
    loaded & request ? any
      SEQ
        out ! n.missed; data
        loaded := FALSE
        n.missed := 0
```

Asynchronous Communication



- A (high priority) sends information to B (low priority).
- A can send at any time and must never be blocked by B not being ready to receive *(even when B has made a request for data previously saved in the smart buffer and not yet taken it)*. ✗✗✗
- B can receive data at any time but, first, it has to make a **`request`**. Such requests will be blocked if there is nothing loaded in the buffer.

Asynchronous Communication

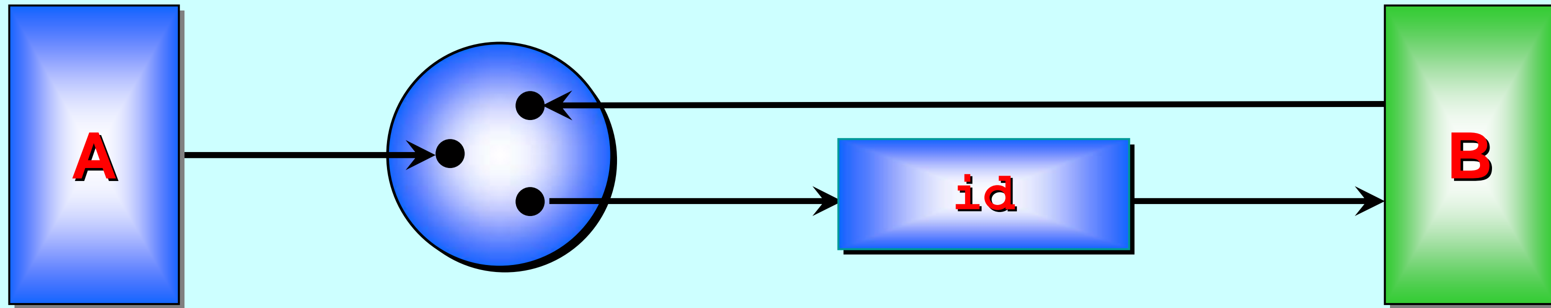


- Insert an **id** process (high priority). Let this get stuck communicating with the low priority one. That block only happens when it does not have to guarantee service to the smart buffer – so no problem! ☺

```
PROC id (CHAN TAGGED.BYTE in?, out!)
  WHILE TRUE
    INT n.missed:
    BYTE data:
    SEQ
      in ? n.missed; data
      out ! n.missed; data
:
```

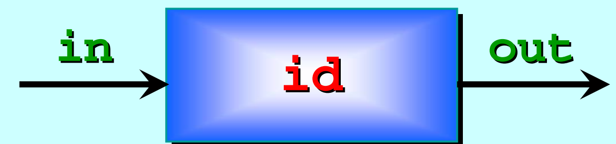
Copyright P.H.Welch

Asynchronous Communication

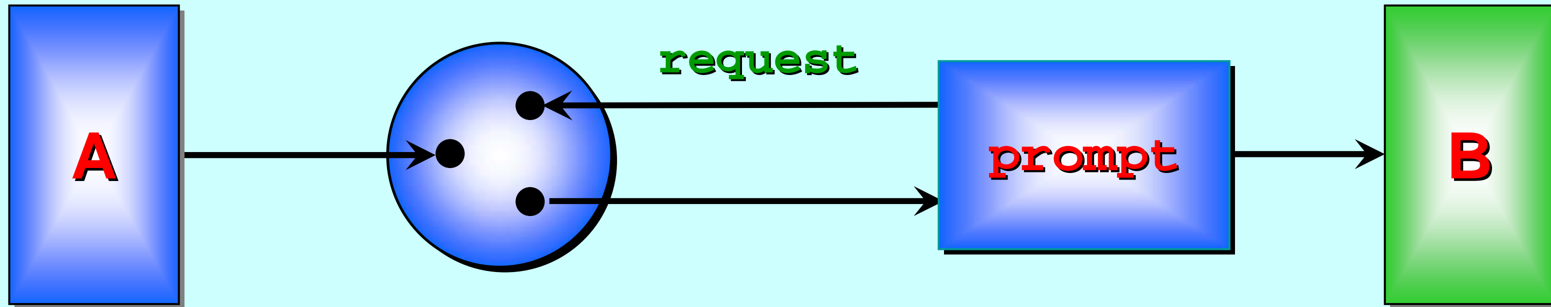


- Initially, it must guarantee worst-case time to accept an input. Subsequently, no input will happen until its output has been taken. *Only then,* must it guarantee service on its input.

```
PROC id (CHAN TAGGED.BYTE in?, out!)
  WHILE TRUE
    INT n.missed:
    BYTE data:
    SEQ
      in ? n.missed; data
      out ! n.missed; data
:
```



Asynchronous Communication

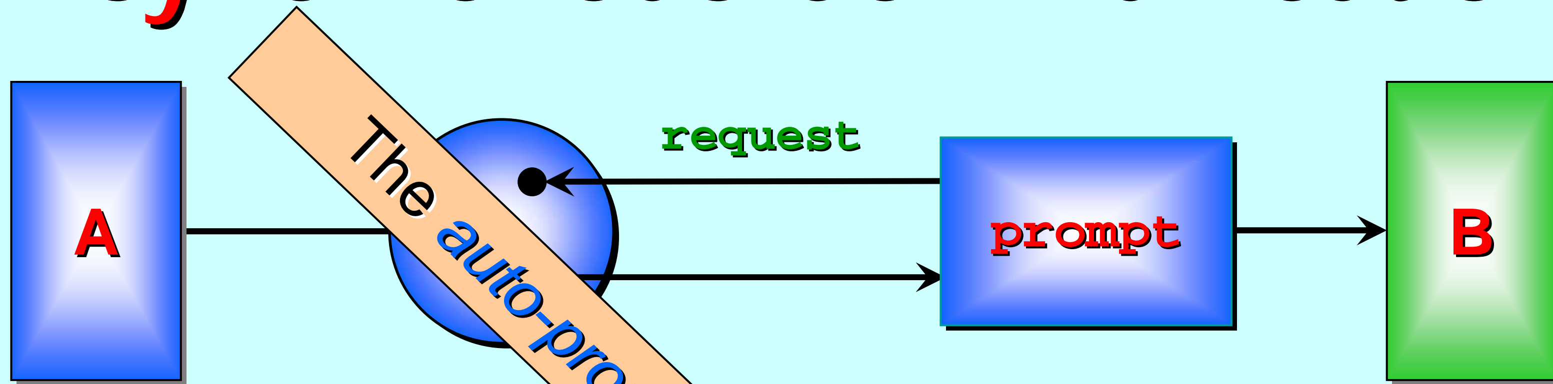


- Alternatively, we *could* relieve B from having to make requests by combining an *auto-prompter* with the memory cell.

```
PROC prompt (CHAN BOOL request!, CHAN INT in?, out!)
  WHILE TRUE
    INT n.missed: BYTE data:
    SEQ
      request ! TRUE
      in ? n.missed; data
      out ! n.missed; data
:
```

Copyright P.H.Welch

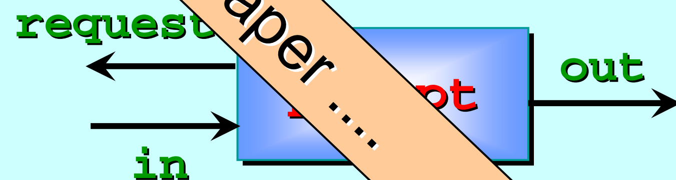
Asynchronous Communication



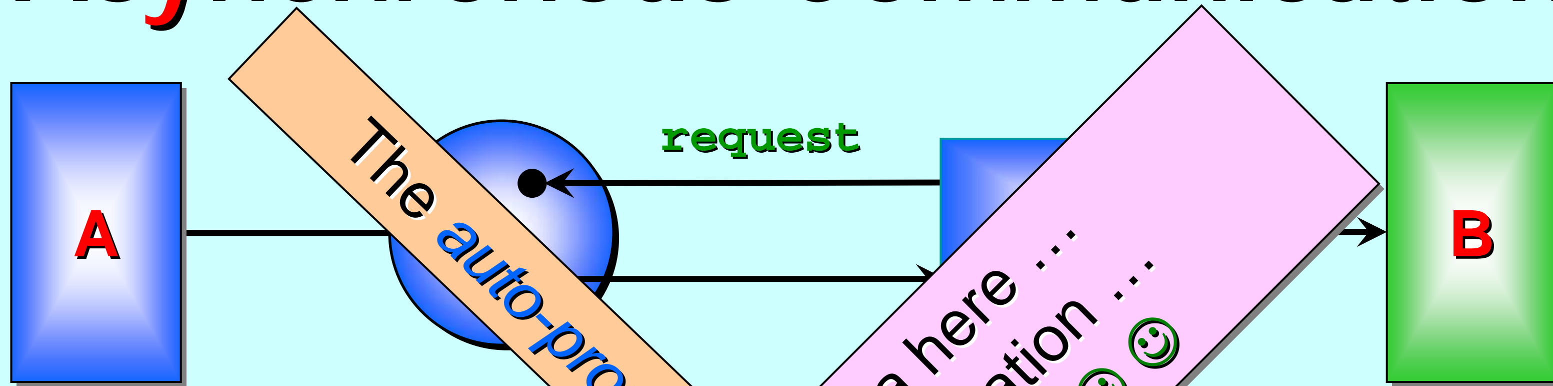
The *auto-prompter* was in the 1987 paper ...

- Alternatively, we *could* reliev[illegible]om having to make requests by combining an *auto-prompter* wi[illegible] memory cell.

```
PROC prompt (CHAN BOOL request!, [illegible] INT in?, out!)
  WHILE TRUE
    INT n.missed: BYTE data:
    SEQ
      request ! TRUE
      in ? n.missed; data
      out ! n.missed; data
:
```



Asynchronous Communication

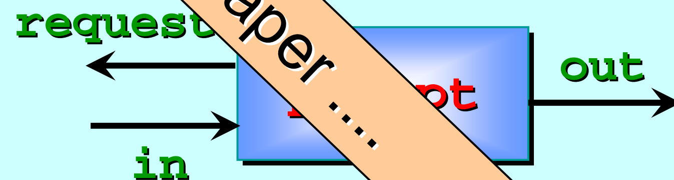


The *auto-prompte*

But it's not a good idea here …
it may deliver stale information …
the id process is better. ☺ ☺ ☺

- Alternatively, we *could* reli… …ving to make requests by combining an *auto-pro*… …memory cell.

```
PROC prompt (CHAN …!, … INT in?, out!)
  WHILE TRUE
    INT n.mis…a:
    SEQ
      req…
      in ? … data
      out ! …ed; data
:
```



…he 1987 paper …

Hard Real Times

Let's start with Conclusions:

- **pre-emptive scheduling is *not required* for hard real-time; [☺]**
- **priority inheritance is a *design error* (dealt with by correct design, not the run-time system); [☺]** ✓
- **the occam-pi/CCSP scheduler can be made to work *even more efficiently* for hard real-time systems than it presently does for soft real-time (e.g. complex system modelling). [☺]**

Hard Real Times

- **pre-emptive scheduling is *not required* for hard real-time; [☺]**

“Imagine each process running on its own silicon …” [1987 paper]

Using this analogy, the paper showed how worst-case response times can be calculated – for a transputer, relying on *pre-emption* of low by high priority processes scheduled on a single processing core.

With multcore, we don’t have to imagine … and we don’t have to pre-empt. ☺ ☺ ☺

Hard Real Times

- pre-emptive scheduling is ***not required*** for hard real-time; [☺]

> **“Imagine each process running on its own silicon …” [1987 paper]**

For XMOS Xcores, this is exactly what happens! Each process, when it has to guarantee response time, waits *(“like a greyhound”)* in its own silicon engine for the signal to be unleashed.

We still need the discussed techniques to not be blocked by another process while on duty! ☺ ☺ ☺

Copyright P.H.Welch

Hard Real Times

- pre-emptive scheduling is ***not required*** for hard real-time; [☺]

"Imagine each process running on its own silicon …" [1987 paper]

For occam-pi, processes may be confined to run on any subset of cores (rather than all).

Set those needing to guarantee hard real-time to run on one set (a singleton is good) and the rest on the rest of the cores.

The above techniques and analysis just work and no pre-emption is needed. ☺ ☺ ☺

Hard Real Times

- the occam-pi/CCSP scheduler can be made to work ***even more efficiently*** for hard real-time systems than it presently does for soft real-time (e.g. complex system modelling). [☺]

We don’t do this yet but ...

Run different versions of the CCSP scheduler on different cores …

On the cores running non-real-time processes, don’t check for interrupts (event pins, links, timeouts) every scheduling point … *faster!*

On the real-time core, run the single core version … *faster!*

Hard Real Times

- **the occa /CCSP scheduler can be made to work *even m efficiently* for hard real-time systems than i esently does for soft real-time (e.g. complex sy modelling). [☺]** ✓

WIN - WIN

We don't do this yet but …

Run different versions of th CSP scheduler on different cores …

On the cores running non-real-ti rocesses, don't check for interrupts (event pins, lin imeouts) every scheduling point … *faster!*

On the real-time core, run the single core rsion … *faster!*

Hard Real Times

- the occa /CCSP scheduler can be de to work *even m efficiently* for hard l-time systems than it presently does f oft real-time (e.g. complex sys modelli [☺]

We don't do this yet but …

Run different version th CSP scheduler on different cores …

On the cores ru g non-real-ti rocesses, don't check for int pts (event pins, lin imeouts) every schedulin nt … *faster!*

On the r -time core, run the single core sion … *faster!*

WIN

WIN

Any questions … ???