Scuola Superiore
Sant'Anna
di Studi Universitari e di Perfezionamento

UNIVERSITÀ DI PISA

Master Program (Laurea Magistrale) in Computer Science and Networking

High Performance Computing Systems and Enabling Platforms

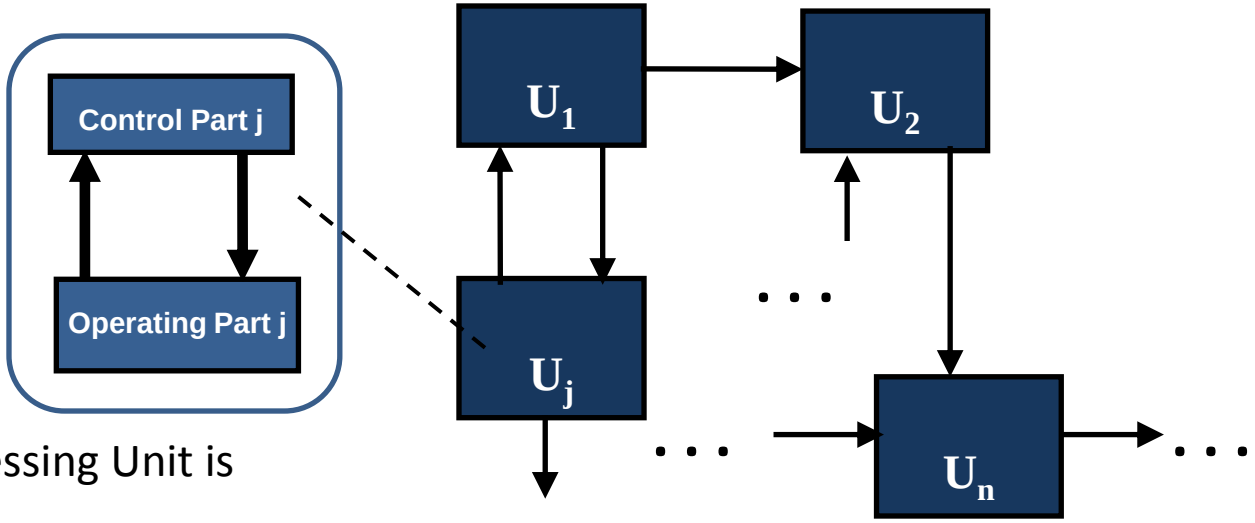
Marco Vanneschi

1. Prerequisites Revisited

1.2. Firmware Structuring

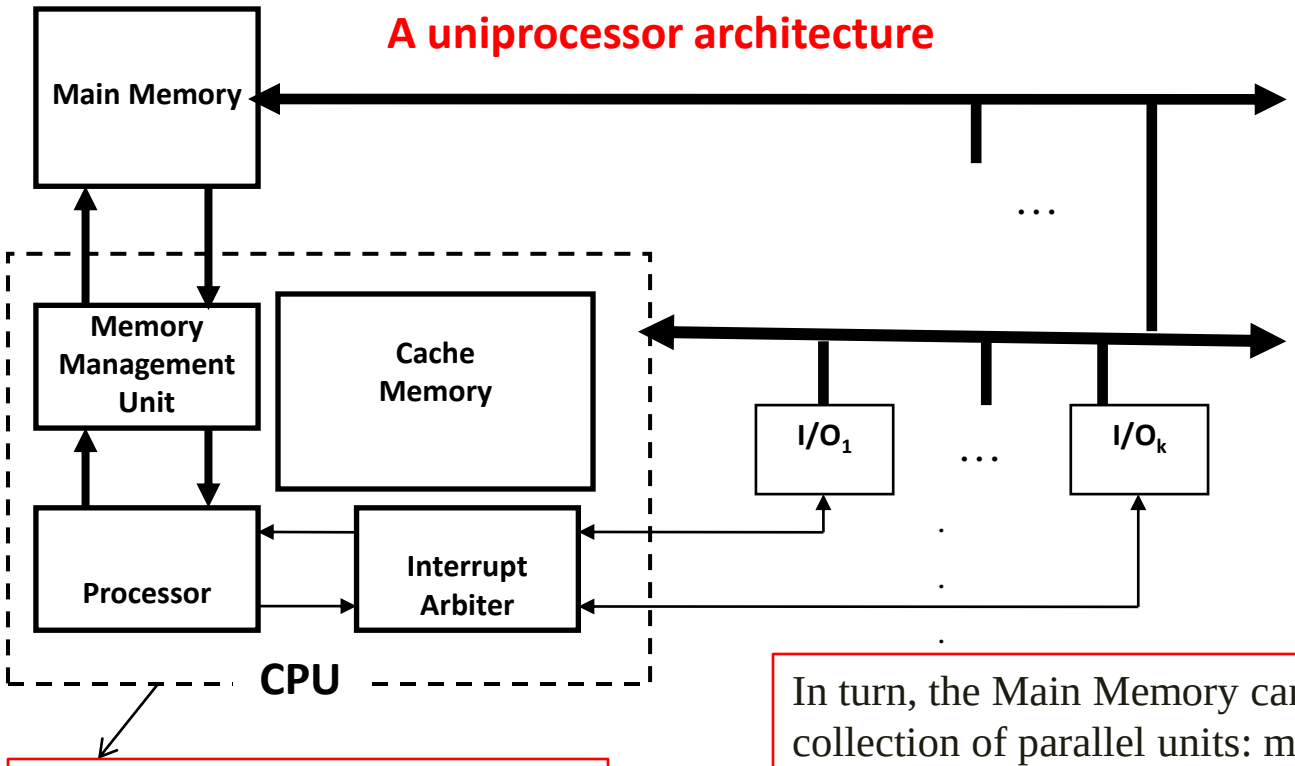
Firmware level

- At this level, the system is viewed as a collection of cooperating modules called PROCESSING UNITS (simply: units).



- Each Processing Unit is
  - autonomous
    - has its own control, i.e. it has self-control capability, i.e. it is an **active computational entity**
  - described by a sequential program, called microprogram.
- Cooperation is realized through COMMUNICATIONS
  - Communication channels *implemented* by physical links and a firmware protocol.
- Parallelism between units.

Examples



In turn, an advanced I/O unit can be a parallel architecture (vector / SIMD coprocessor / GPU / …), or a powerful multiprocessor-based network interface, or ….

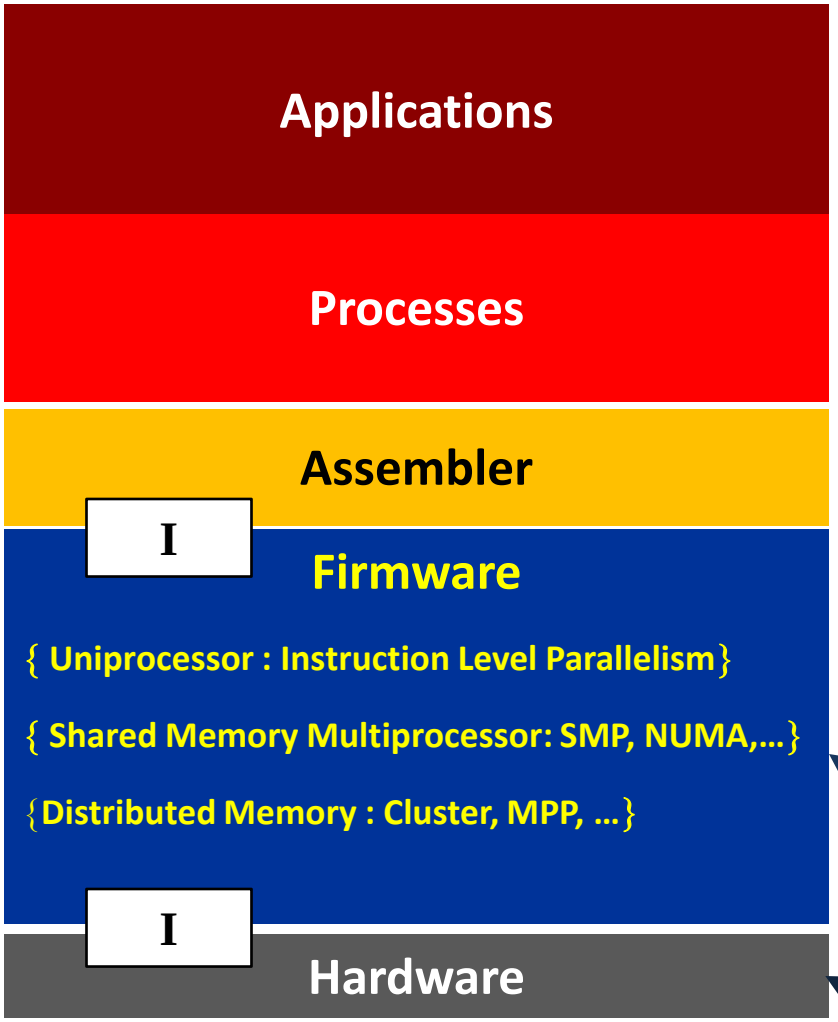
In turn, the Main Memory can be implemented as a collection of parallel units: modular memory, interleaved memory, buffer memory, or …

In turn, CPU is a collection of communicating units (on the *same chip*)

In turn, the Processor can be implemented as a (large) collection of parallel, pipelined units

In turn, the Cache Memory can be implemented as a collection of parallel units: instruction and data cache, primary – secondary – tertiary… cache

Firmware interpretation of assembly language



The firmware interpretation of any assembly language instructions has a **decentralized organization**, even for simple uniprocessor systems.

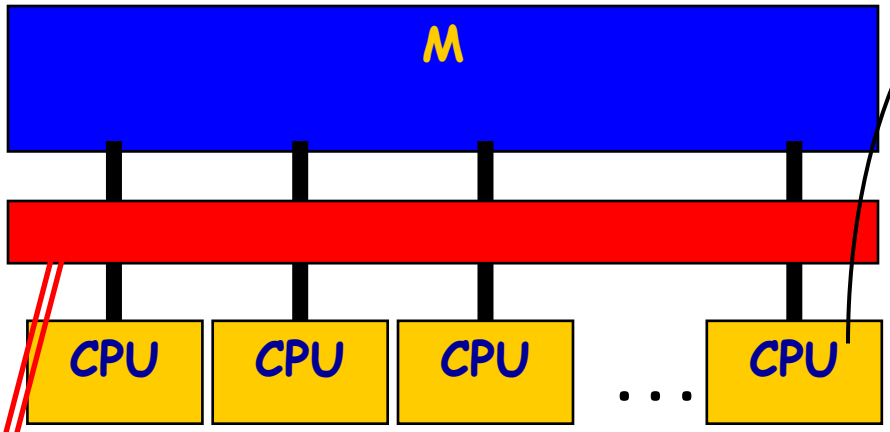
The interpreter functionalities are **partitioned** among the various processing units (belonging to CPU, M, I/O subsystems), *each unit having its own Control Part and its own microprogram*.

Cooperation of units through explicit message passing .

- *The true architectural level.*
- Microcoded interpretation of assembler instructions.
- This level is fundamental (it cannot be eliminated).

- Physical resources: combinatorial and sequential circuit components, physical links

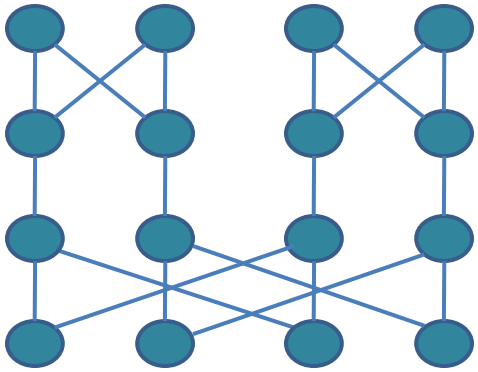
Parallel architectures: (very) large collections of (small) processing units

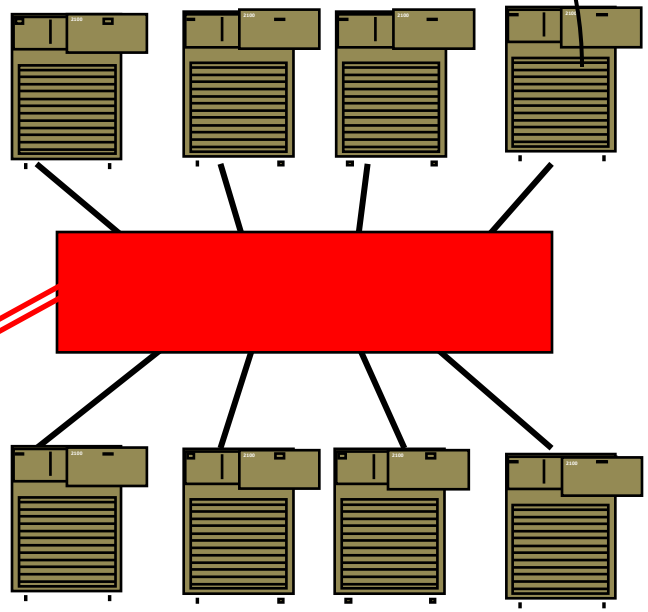


Shared memory multiprocessor

Instruction Level Parallelism CPU (pipeline, superscalar, multithreading, …)

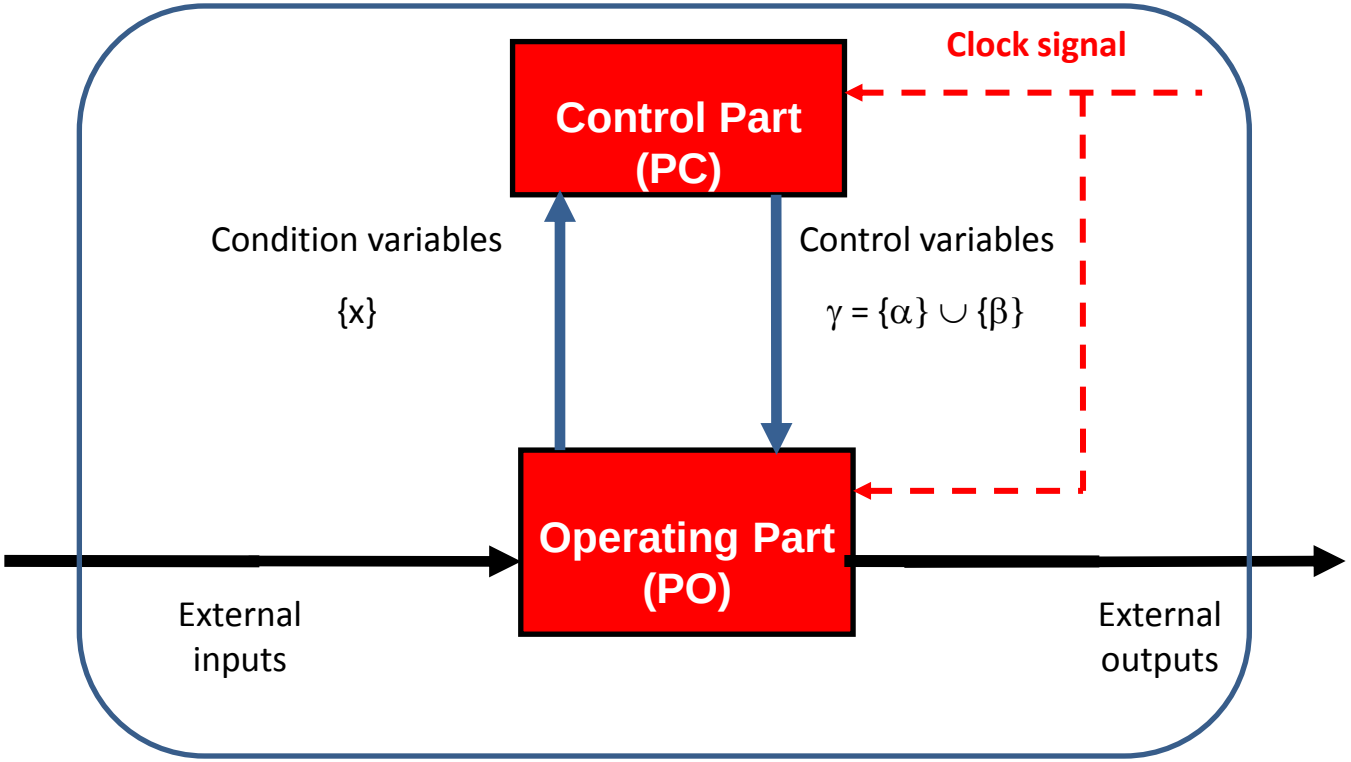
"Limited degree" Interconnection Network ("one-to-few")





Distributed memory multicomputer: PC cluster, Farm, Data Centre, …

Processing Unit model: Control + Operation

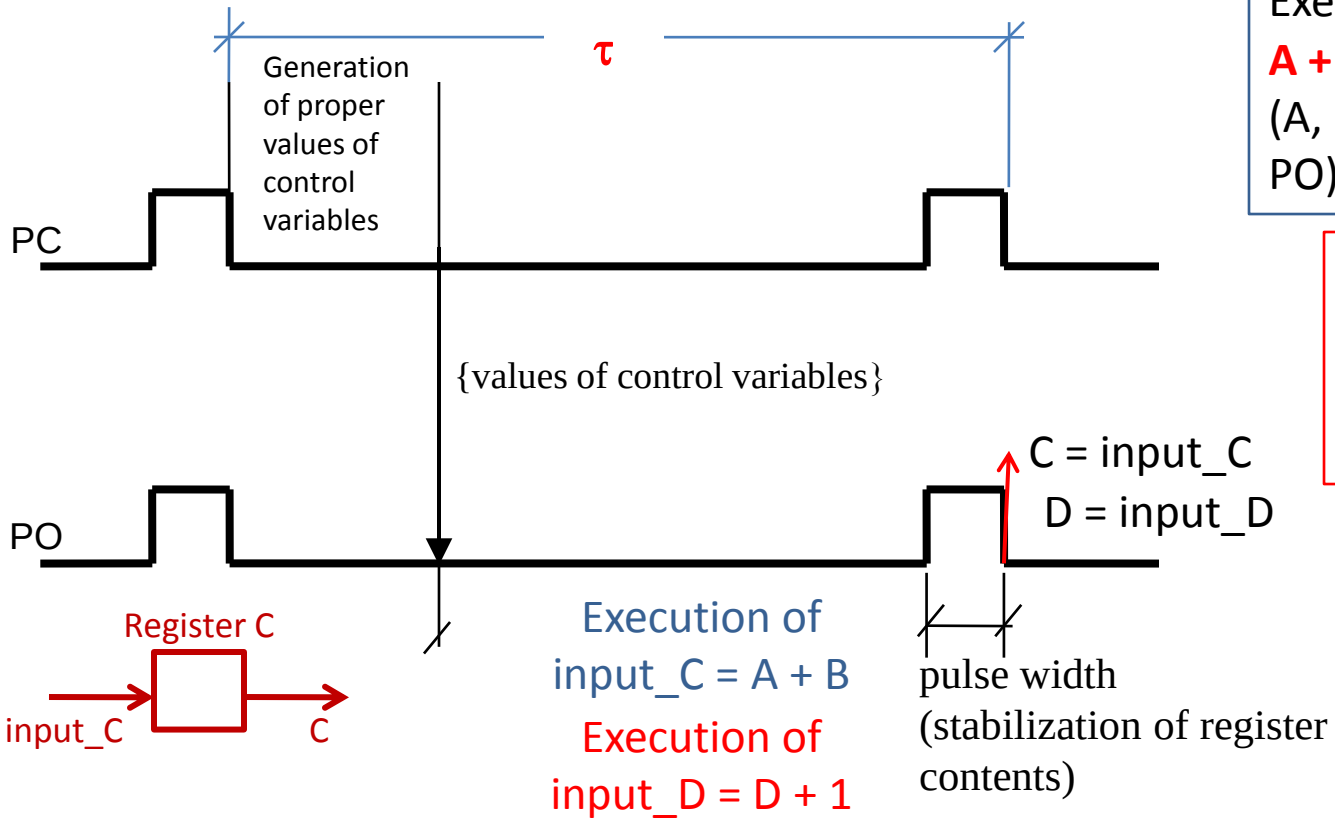


PC and PO are ***automata***,
implemented by *synchronous sequential networks* (sequential circuits)

Clock cycle: time interval for executing a (any) microinstruction.

Synchronous model of computation.

Clock cycle: informal meaning



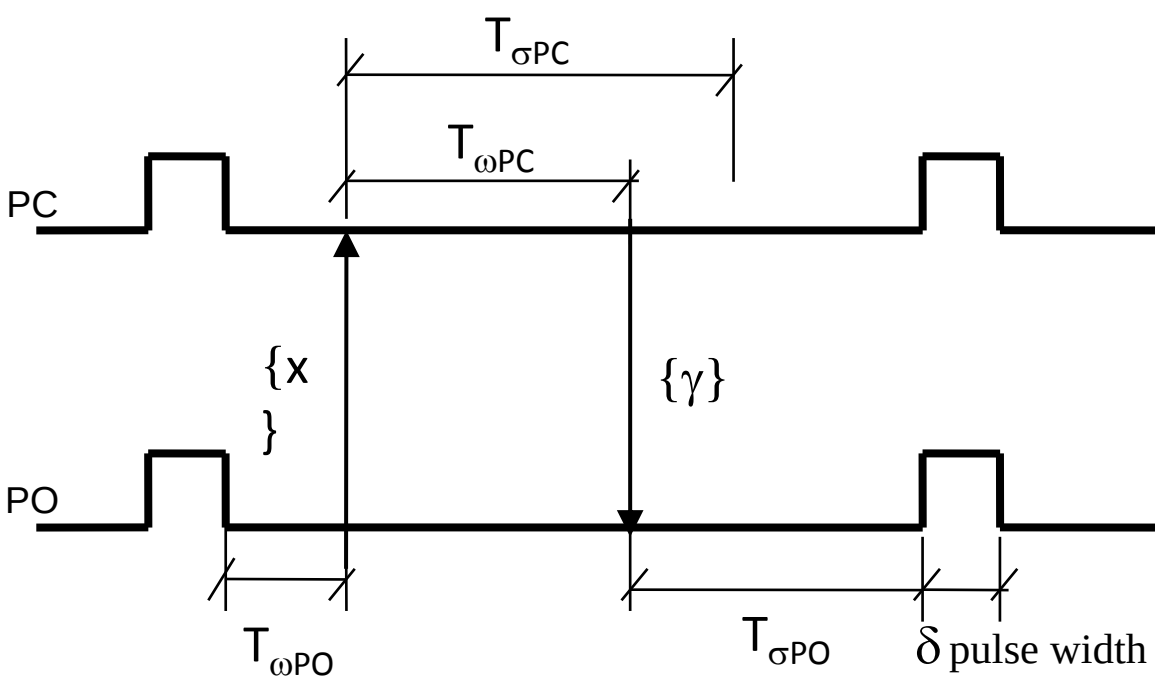
Execution of
A + B → C
(A, B, C: registers of PO)

Execution of
A + B → C, D + 1 → D
in parallel in the same clock cycle

Clock cycle, τ = maximum stabilitation delay time for the execution of a (any) microinstruction, i.e. for the stabilization of the inputs of all the PC and PO registers

clock frequency $f = 1/\tau$ e.g. τ = 0,25 nsec f = 4 GHz

Clock cycle: formally



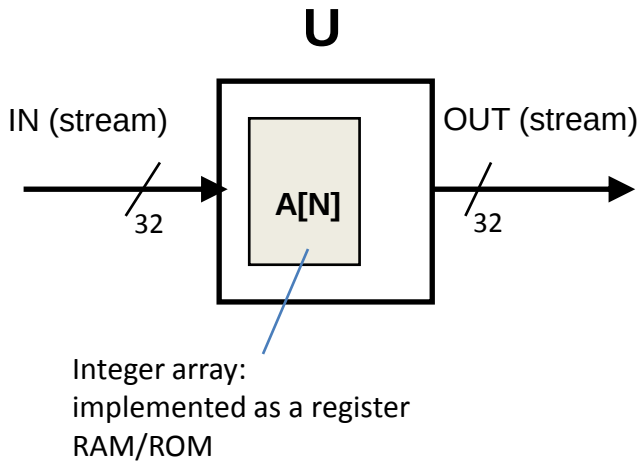
ω_A: output function of automata A

σ_A: state transition function of automata A

T_F: stabilization delay of the combinatorial network implementing function F

clock cycle $\tau = T_{\omega PO} + \max (T_{\omega PC} + T_{\sigma PO}, T_{\sigma PC}) + \delta$

Example of a simple unit design



Specification ∀ **IN: OUT = number_of_occurrences (IN, A)**

Pseudo-code of interpreter:

```
while (true) do
        { B = IN; C = 0;
          for (J = 0; J < N; J ++)
                  if ( A[ J ] == B) then C = C + 1;
          OUT = C
        }
```

Microprogram (= interpreter implementation)

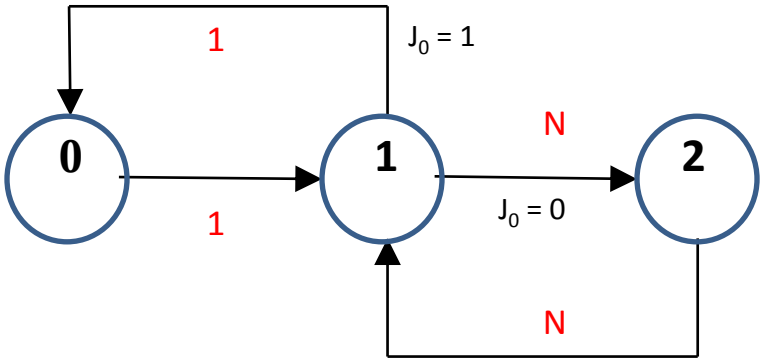
0. IN → B, 0 → C, 0 → J, **goto** 1

1. **if** (J_0 = 0) **then** {*zero* (A[J] – B) → Z, **goto** 2}
   else {C → OUT, **goto** 0}

2. **if** (Z = 0) **then** {J + 1 → J, **goto** 1}
   else {J + 1 → J, C + 1 → C, **goto** 1}

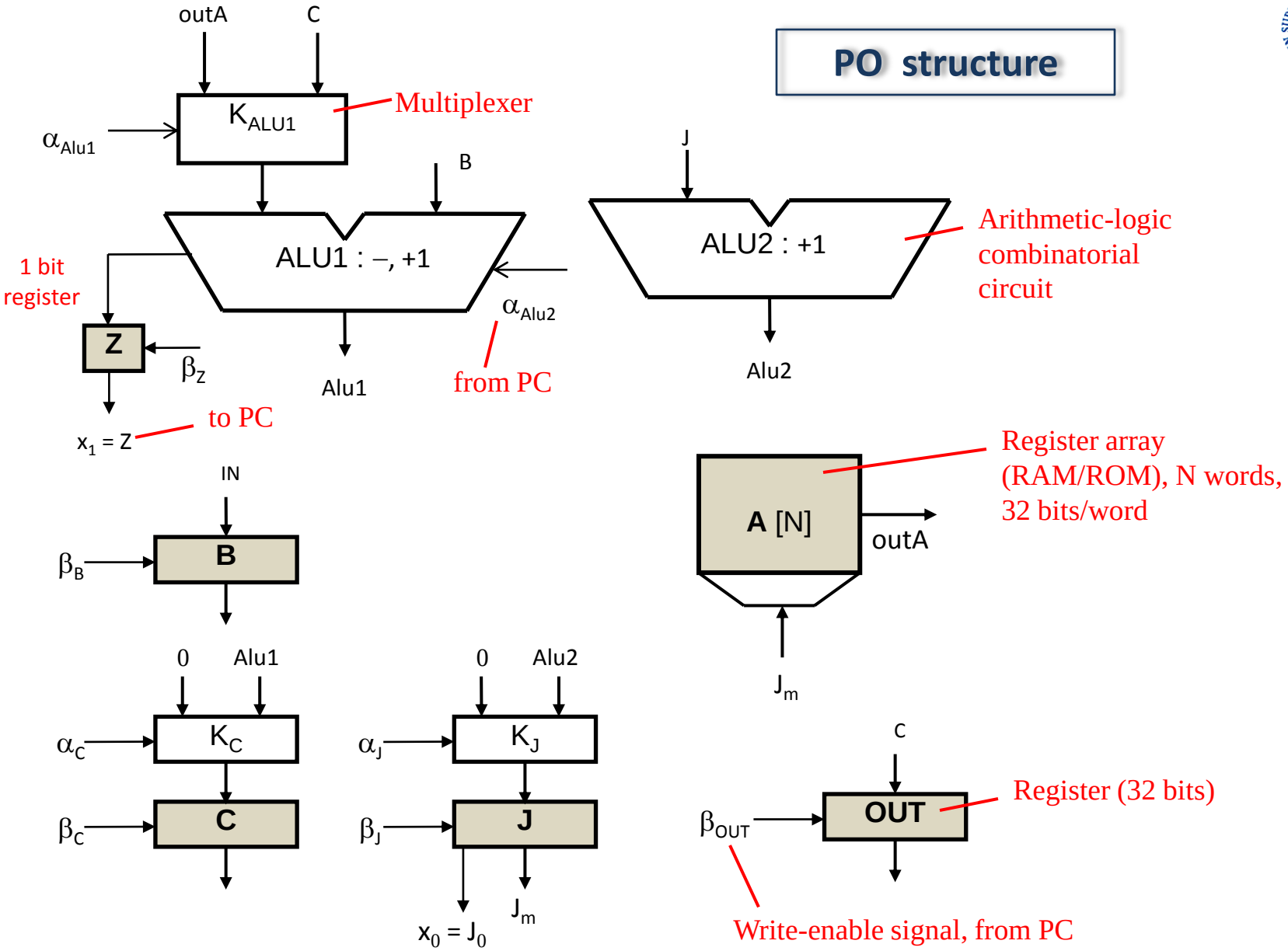
// to be completed with synchronized communications //

Control Part state graph (automaton):

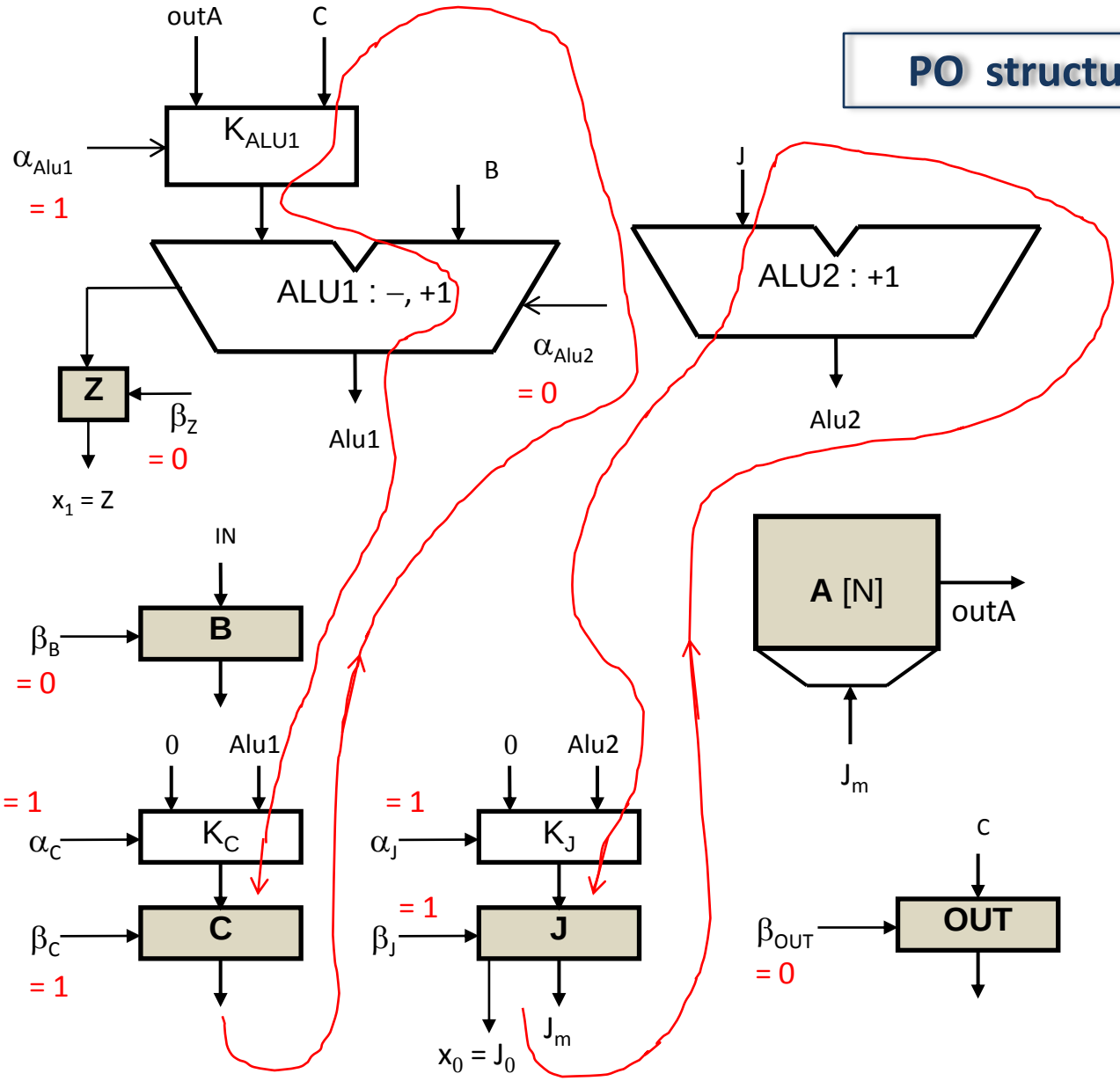


Service time of U: $T = k\,\tau = (2 + 2N)\,\tau \sim 2N\,\tau$

PO structure



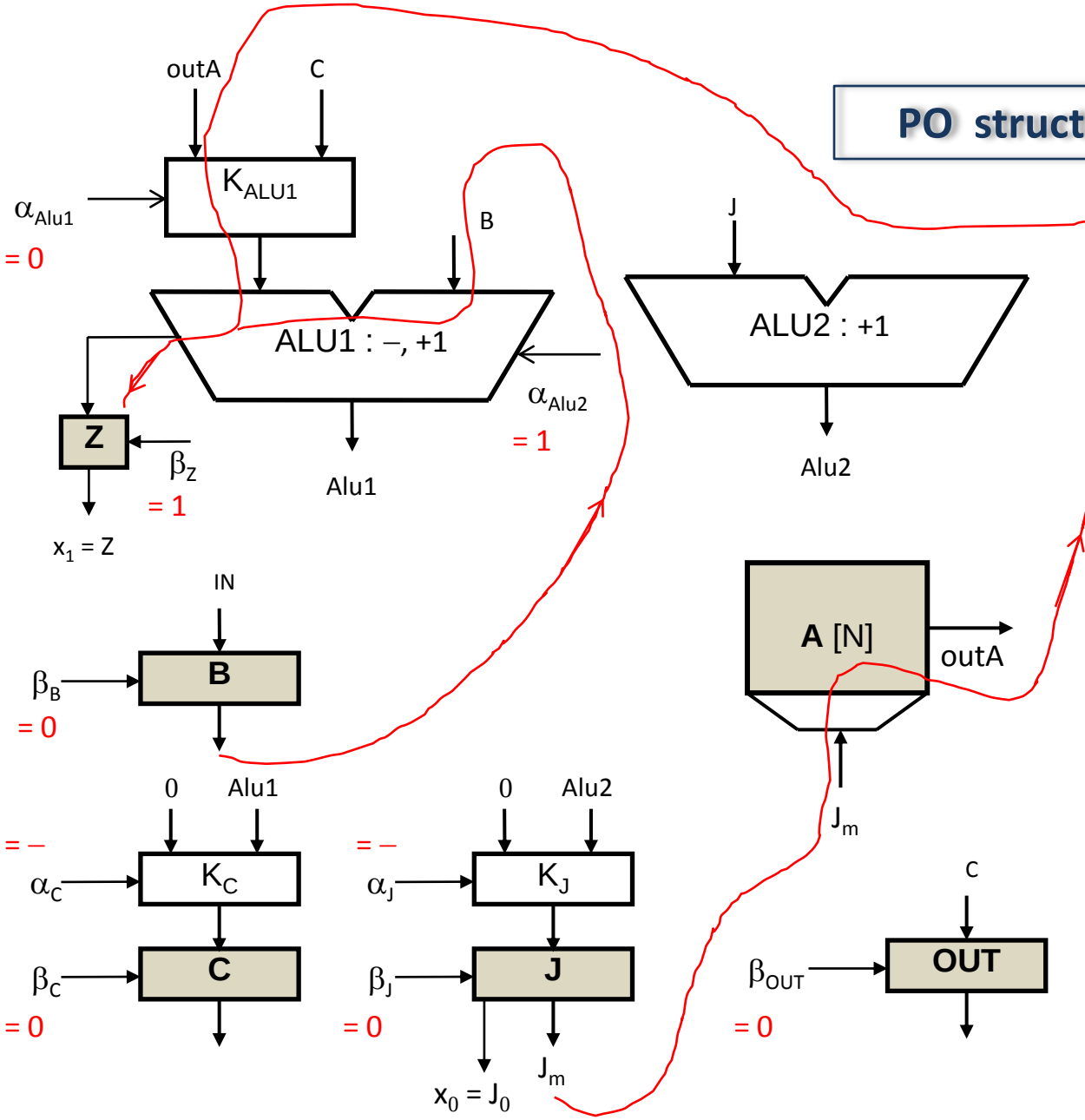
PO structure



Data path
for the execution of

J + 1 → J, C + 1 → C

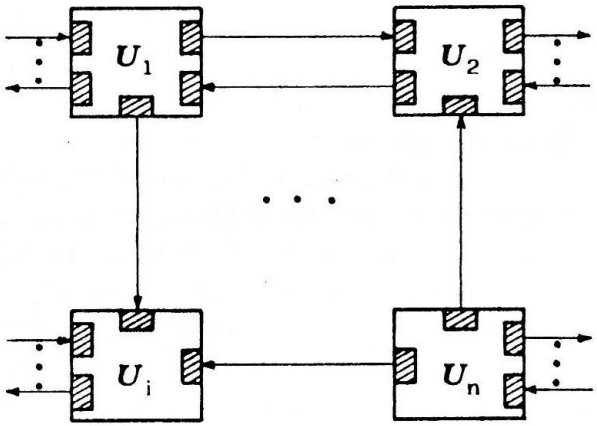
PO structure



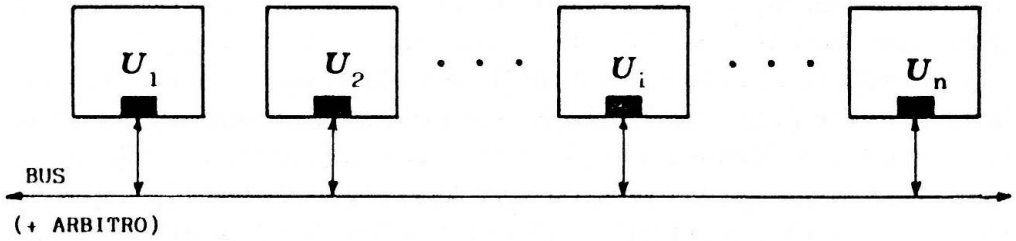
Data path
for the execution of

***zero* (A[J] – B) → Z**

Communications at the firmware level



Dedicated links



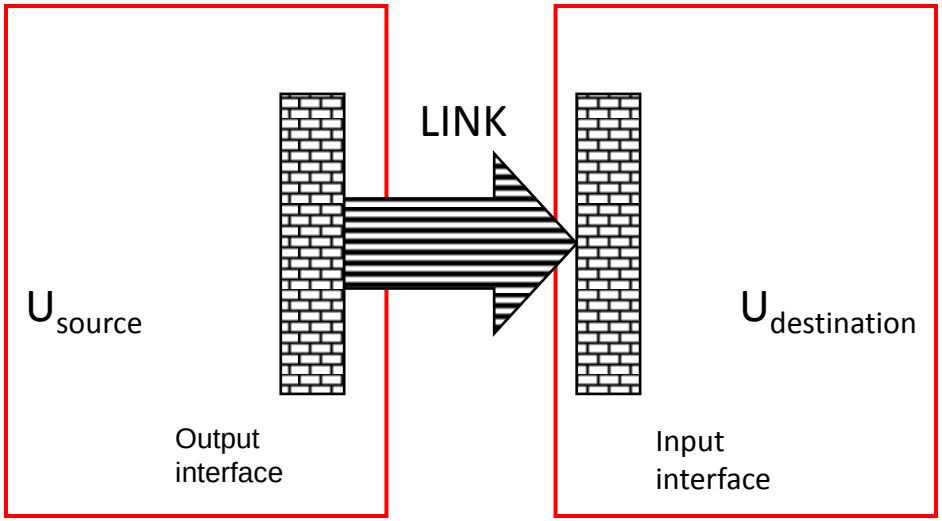
Shared link: bus

In general, for computer structures : **parallel links** (e.g. one or more words)

Valid for
- a uniprocessor computer,
- a multiprocessor with shared memory,
- some multicomputers with distributed memory (MPP)

Not valid for
- distributed architectures with standard networks,
- some kind of simple I/O.

Basic firmware structure for dedicated links



U_{source}::

while (true) **do**

{ < production of message MSG >;

< *send MSG to* $U_{destination}$ >

}

$U_{destination}$::

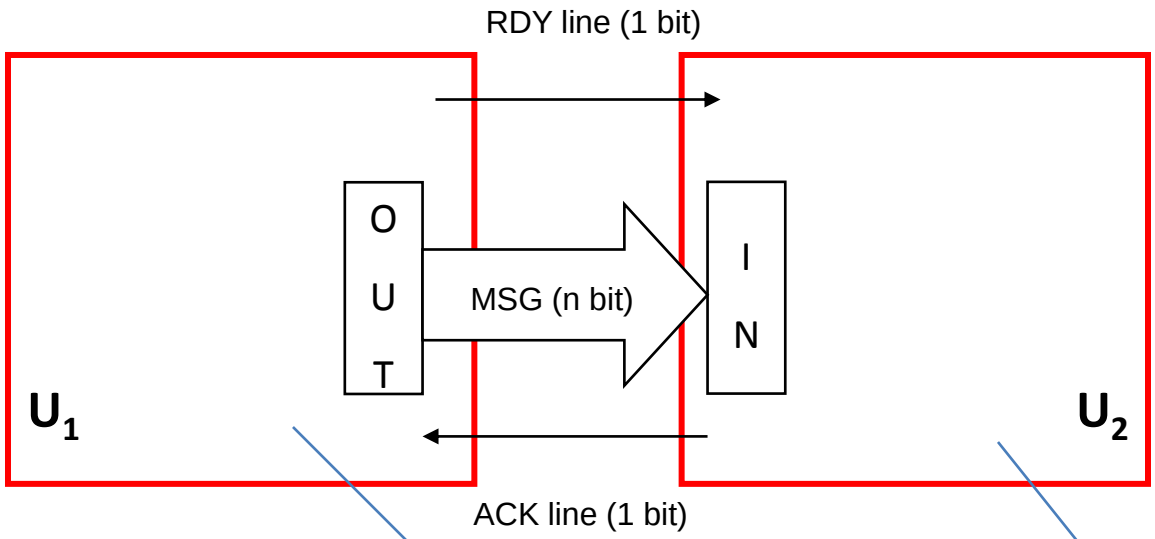
while (true) **do**

{ < *receive MSG from* U_{source} >;

< process MSG >

}

Ready – acknowledgement protocol



< *send MSG* > ::

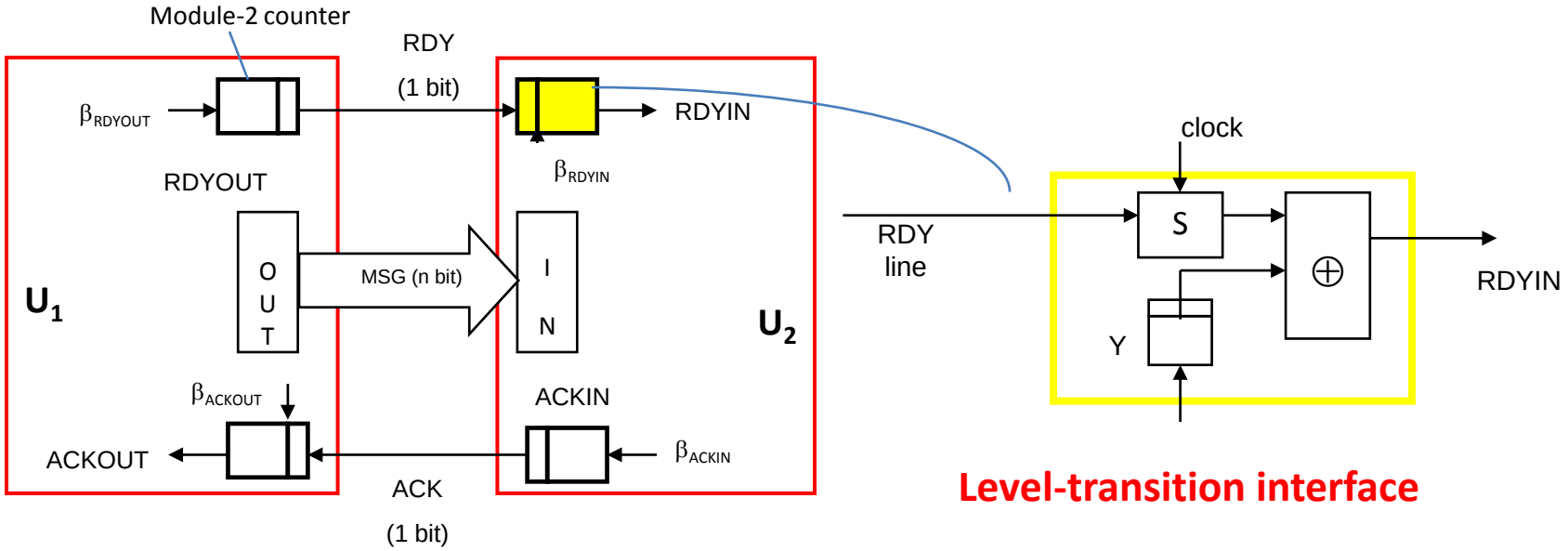
{ wait ACK;
  write MSG into OUT, signal RDY
}

< *receive MSG* >::

{ wait RDY;
  utilize IN, signal ACK
}

Asynchronous communication

Implementation



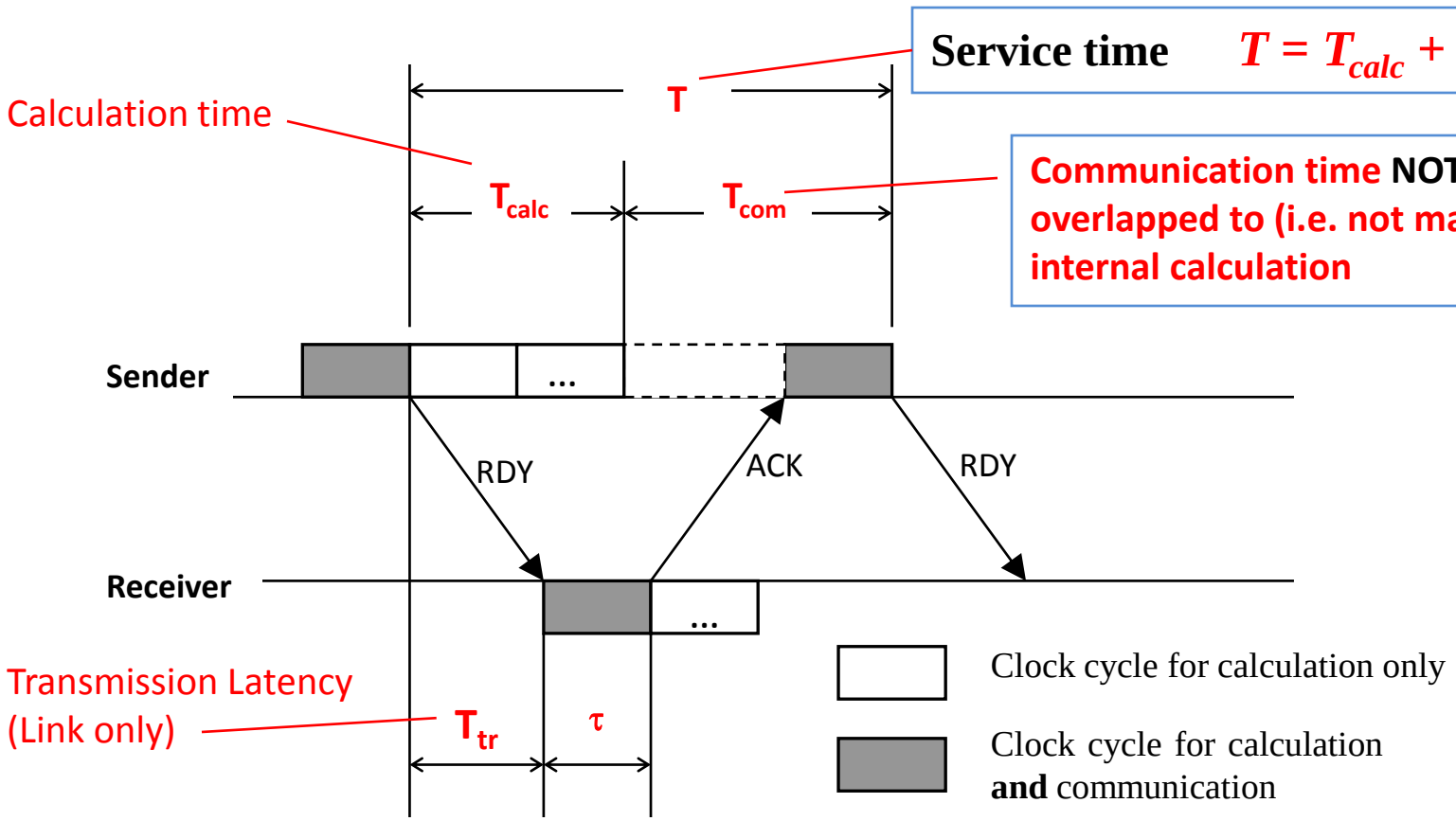
< *send MSG* > ::

{ **wait until** ACKOUT;
MSG → OUT, **set** RDYOUT,
reset ACKOUT;
}

< *receive MSG* >::

{ **wait until** RDYIN;
f(IN, …) …… , **set** ACKIN,
reset RDYIN;
}

Communication latency and impact on service time



Communication Latency

$$L_{com} = 2\,(T_{tr} + \tau)$$

$$T_{calc} \geq L_{com} \quad \Leftrightarrow \quad T_{com} = 0$$

Communication latency and impact on service time

Service time $T = T_{calc} + T_{com}$

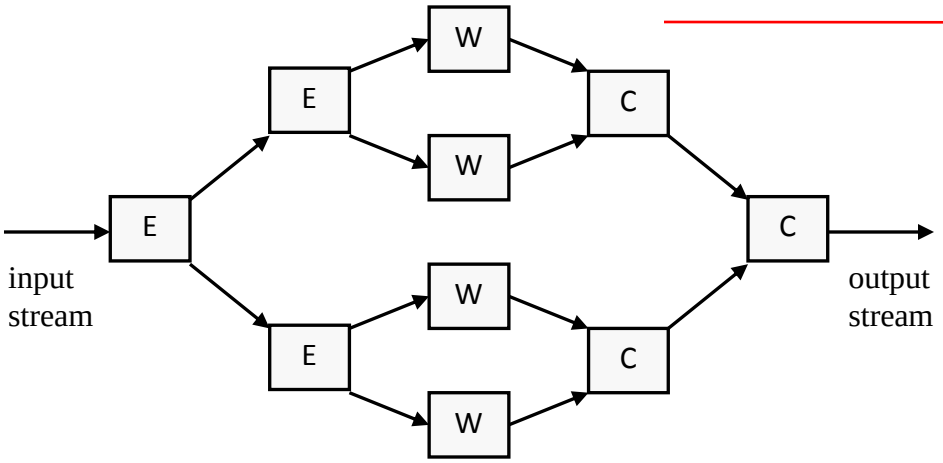
Comm. Latency $L_{com} = 2\,(T_{tr} + \tau)$

$T_{calc} \geq L_{com} \quad \Leftrightarrow \quad T_{com} = 0$

efficiency $\varepsilon \sim 1$,
~ ideal computation bandwidth (throughput)

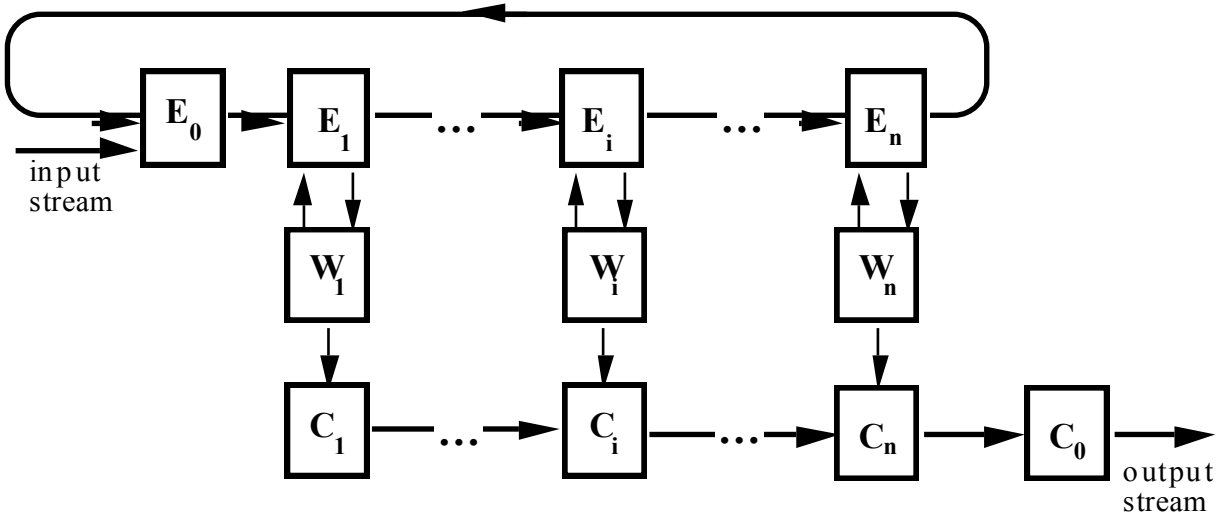
- Impact of T_{tr} on service time T is significant
  - e.g. memory access time = response time of "external" memory to processor requests ("external" = outside CPU-chip)
- **Inside the same chip**: $T_{tr} \sim 0$,
  - e.g. between processor – MMU – cache
  - **except for** on-chip interconnection structures with "long wires" (e.g. buses, rings, meshes, …)
- With $T_{tr} \sim 0$, **very "fine grain" computations** are possible without communication penalty on service time: $L_{com} = 2\tau \quad \Rightarrow \quad$ if $T_{calc} \geq 2\tau$ then $T = T_{calc}$
  - Important application in **Instruction Level Parallelism** processors and in **Interconnection Network** switch-units.

Some interesting structures



A **FARM** structure at the firmware level:
Workers interconnected by **tree** structures (emitter – collector trees) …

… or by **ring and/or linear chain** structures



Some interesting structures

- In general, all the most interesting interconnection structures for
  - shared memory multiprocessors
  - distributed memory multicomputer, cluster, MPP

  are “limited degree” networks
  - based on **dedicated** links
- Examples (Course Part 2):
  - n-dimension cubes
  - n-dimensione butterflys
  - n-dimension Trees / Fat Trees
  - rings and multi-rings
- Currently the best commercial networks belong to these classes:
  - Infiniband
  - Myrinet
  - …

(Next) future

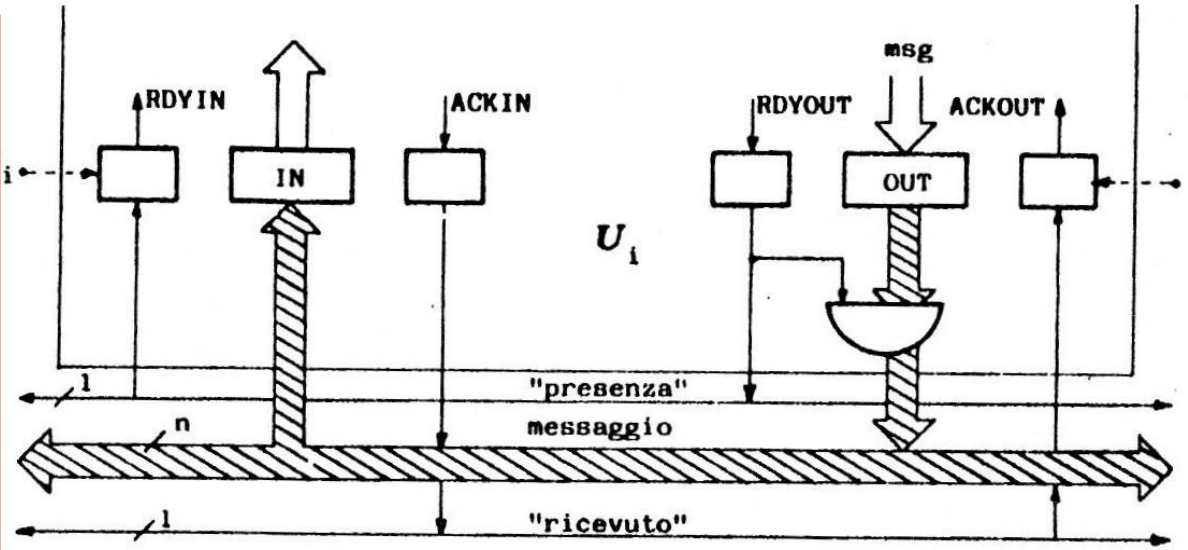
- *On chip interconnection networks*
- *On chip **optical** interconnection networks*

- *Clear trend towards strong inter-relationship between HPC and Telecommunications*
  - *A unique feature of this Master Program*

(Traditional, old-style) Bus

Serious problems of bandwidth (**contention**) for bus-based parallel structures.

Trend: replace buses with **limited degree networks** based on dedicated links.



Sender behaviour

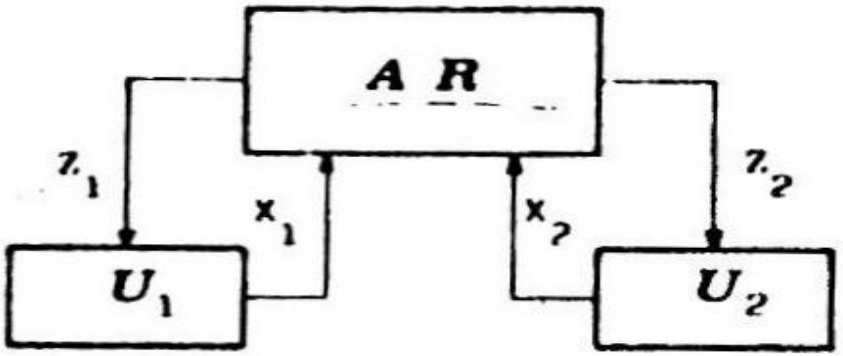
```
while DIS = 1 do;
1→RIC;
while DIS = 0 or ACKOUT = 1 do;
msg→OUT;
1→RDYOUT;
while ACKOUT = 0 do;
0→RDYOUT;
0→RIC;
```

Receiver behaviour

```
while RDYIN = 0 or
  IN[DEST] ≠ «mio nome»
  do;
f(IN)→vtg;
1→ACKIN;
while RDYIN = 1 do;
0→ACKIN
```

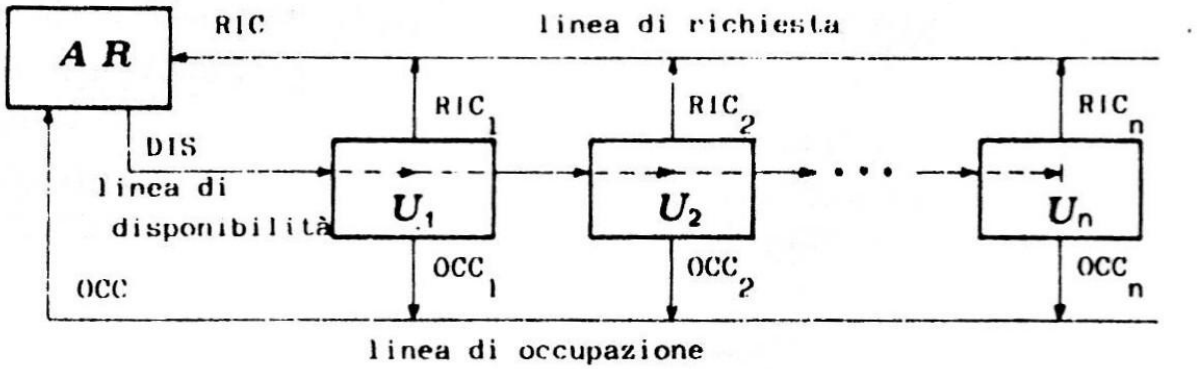
Arbiter mechanisms (for bus writing)

Independent Request



Arbiters

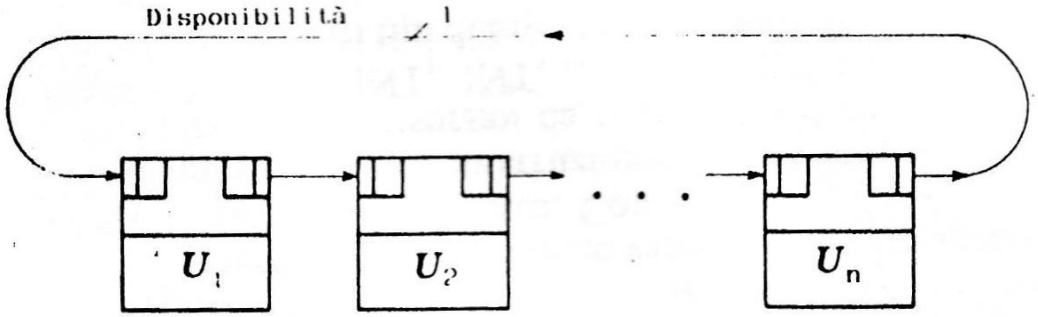
- Daisy Chaining



- Polling (similar)

Decentralized arbiters

- Token ring



- Non-deterministic with collision detection