Validation of a Parameterized Bus Architecture Model

SCHMALTZ Julien and BORRIONE Dominique

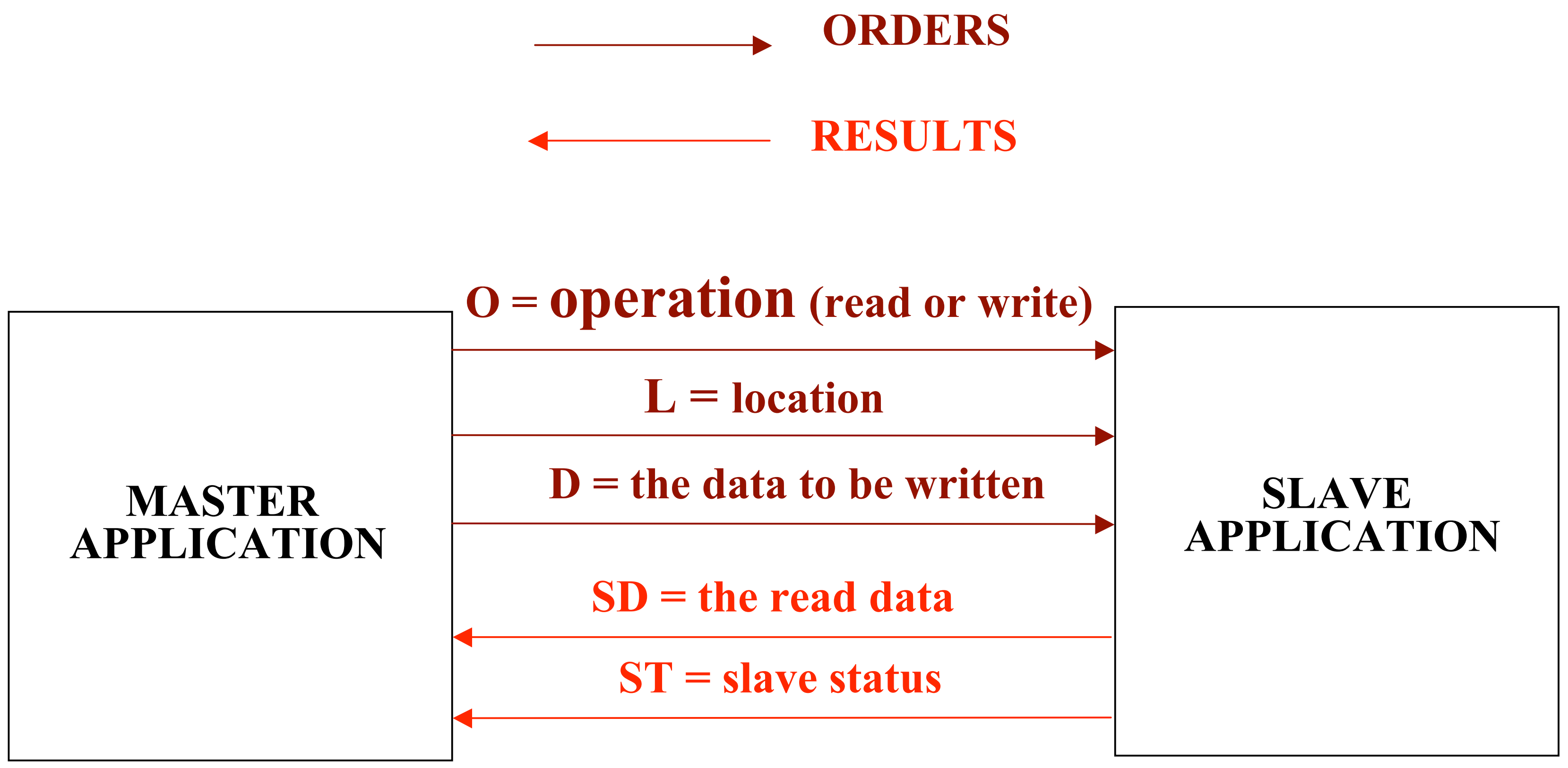
TIMA Laboratory -VDS Group, Grenoble, France

ACL2 Workshop 2003 Boulder, CO

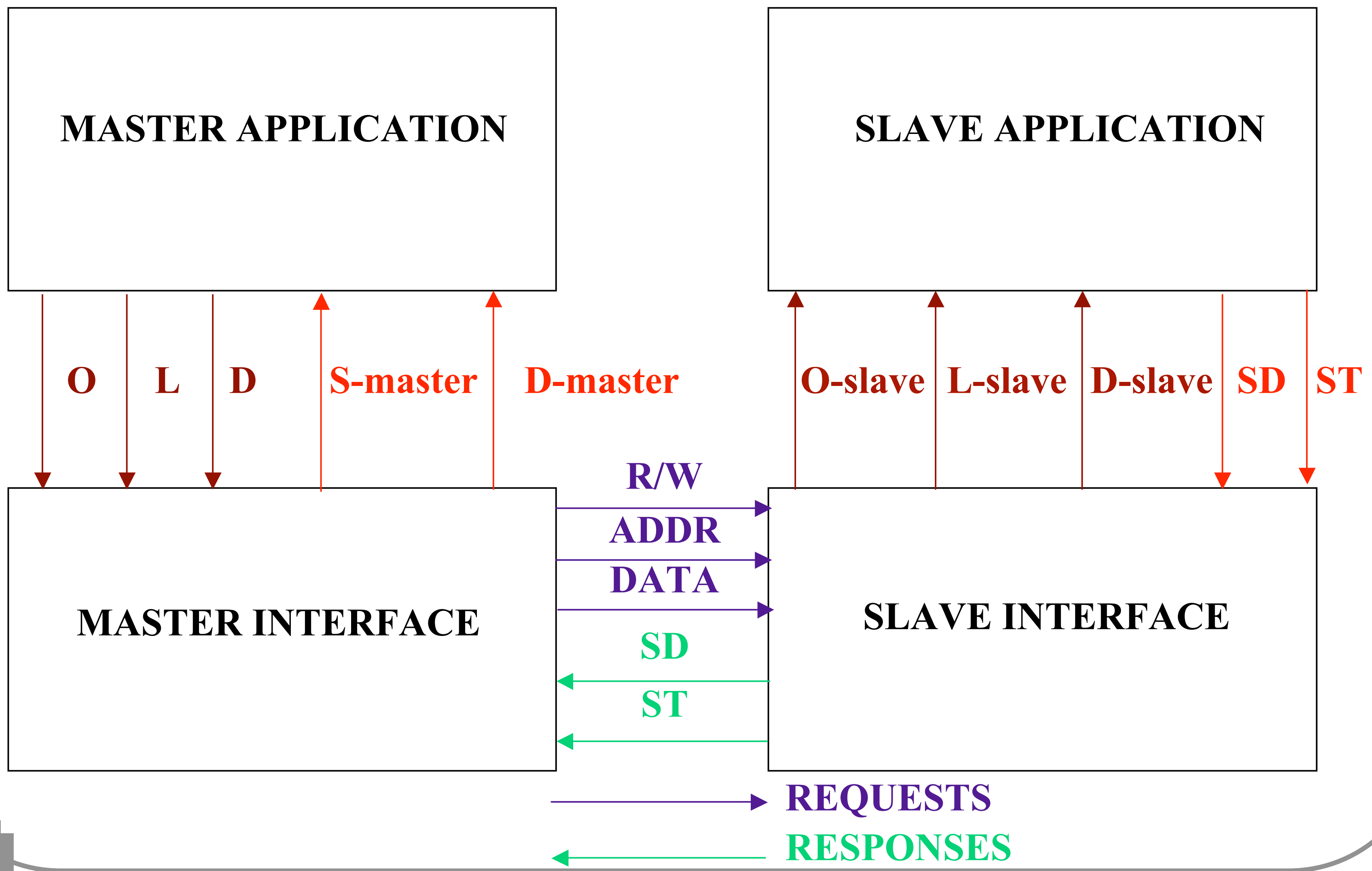
Outline

- The point to point connection
- Functional modeling
- Validation of the model
- One step forward: the bus case

The direct point to point connection



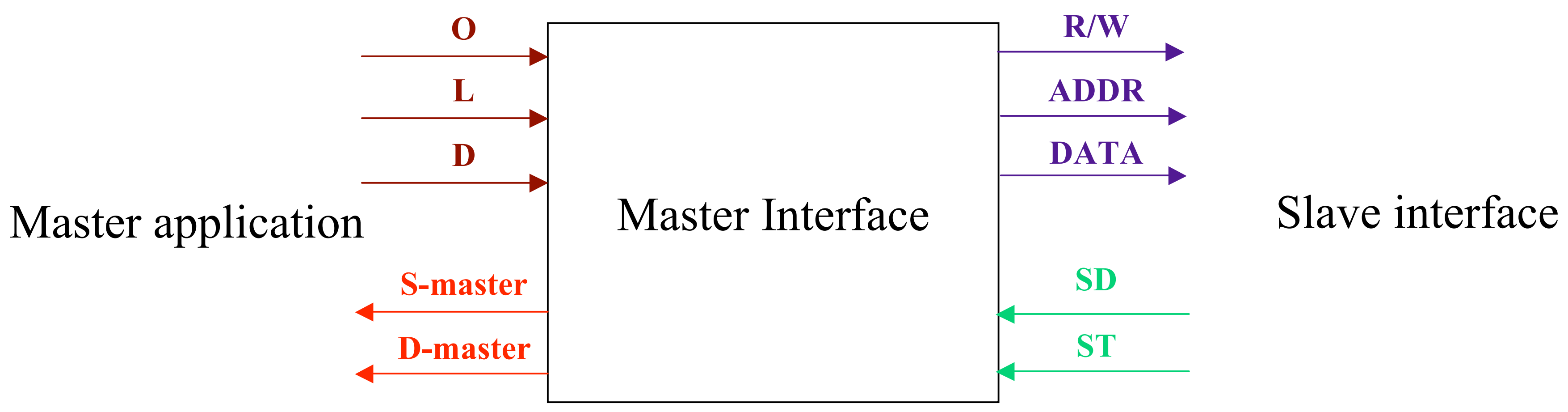
Introduction of the interfaces



Functional modeling

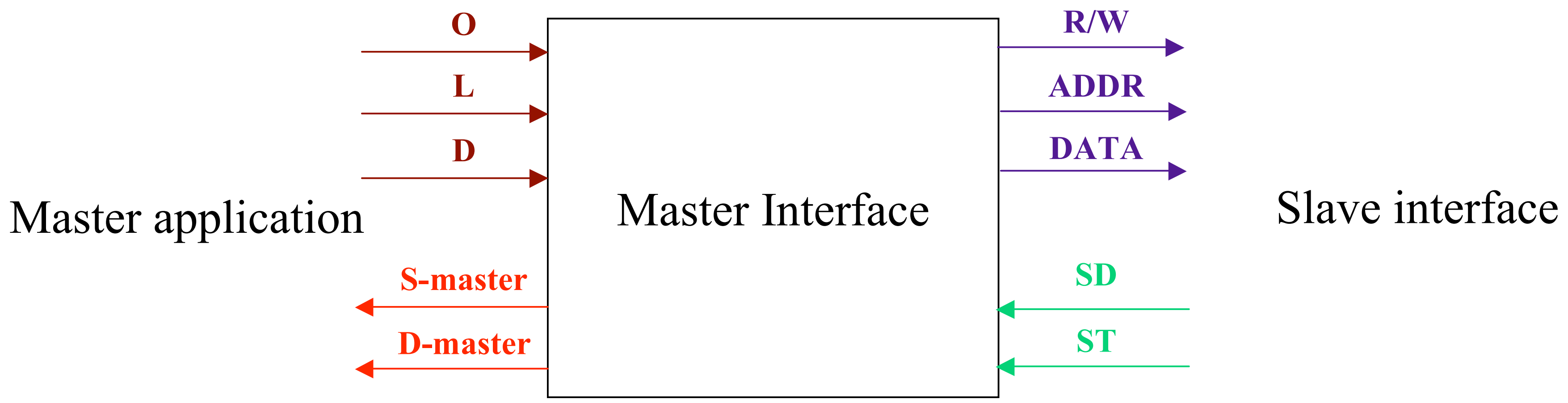
- Each component is represented by a function
- Time is abstracted away
- Functional composition is used to express sequential communication events

Modeling the master interface



```
(defun master_interface (O L D   SD ST)
    (if  (equal  O  ‘read)
        (list  (list  1 L D)  (list SD ST))
        (list  (list  0 L D)  (list SD ST))))
```

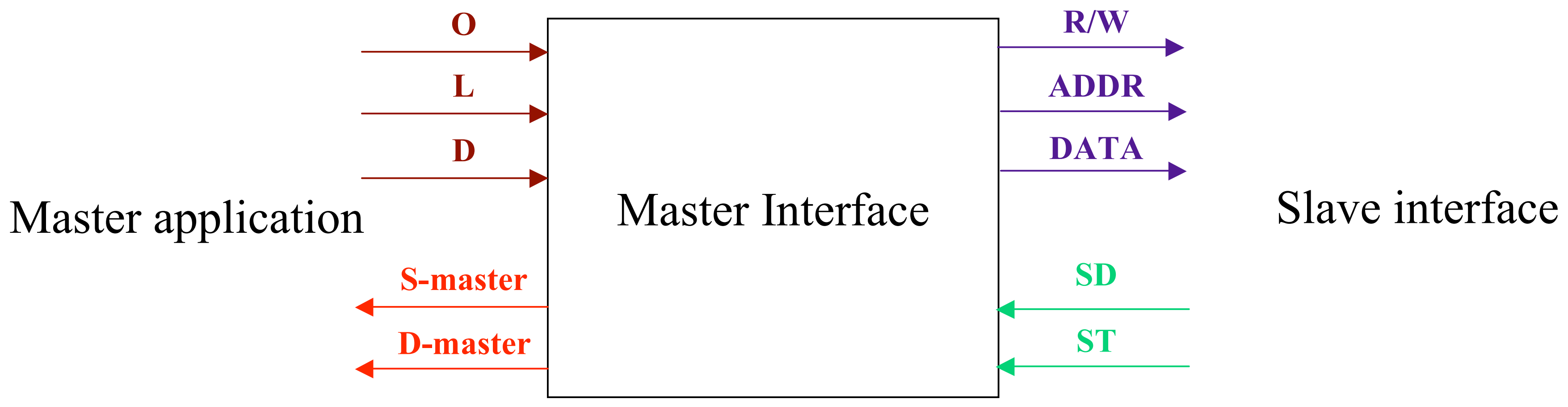
Modeling the master interface



```
(defun master_interface (O L D   SD ST)
    (if  (equal  O  'read)
        (list  (list  1  L D)  (list SD ST))
        (list  (list  0  L D)  (list SD ST))))
```

(R/W x)

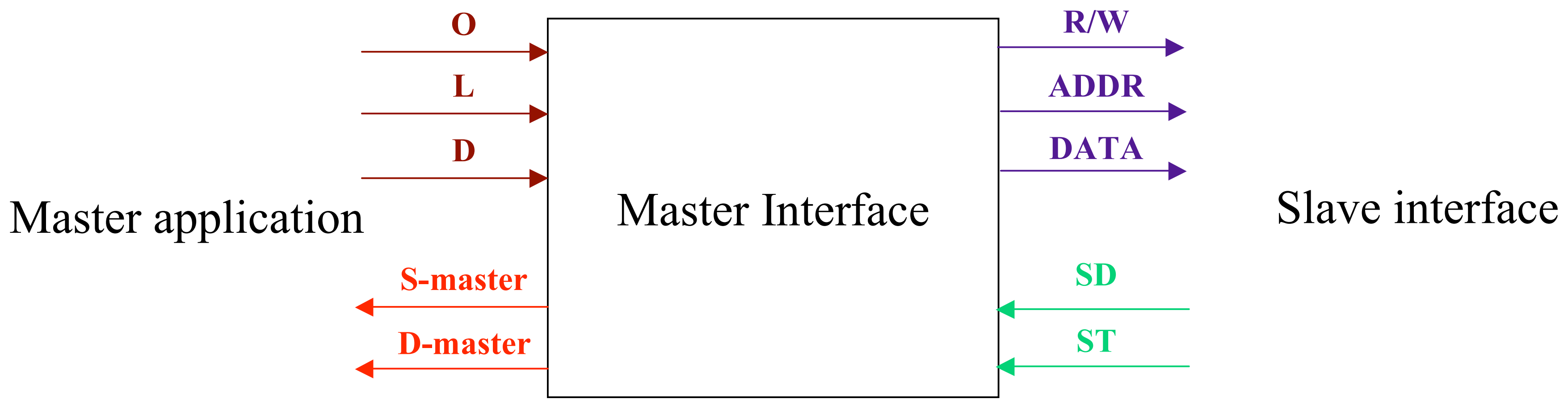
Modeling the master interface



```
(defun master_interface (O L D   SD ST)
   (if  (equal  O  ‘read)
        (list  (list  1 L D)  (list SD ST))
        (list  (list  0 L D)  (list SD ST))))
```

(ADDR x)

Modeling the master interface

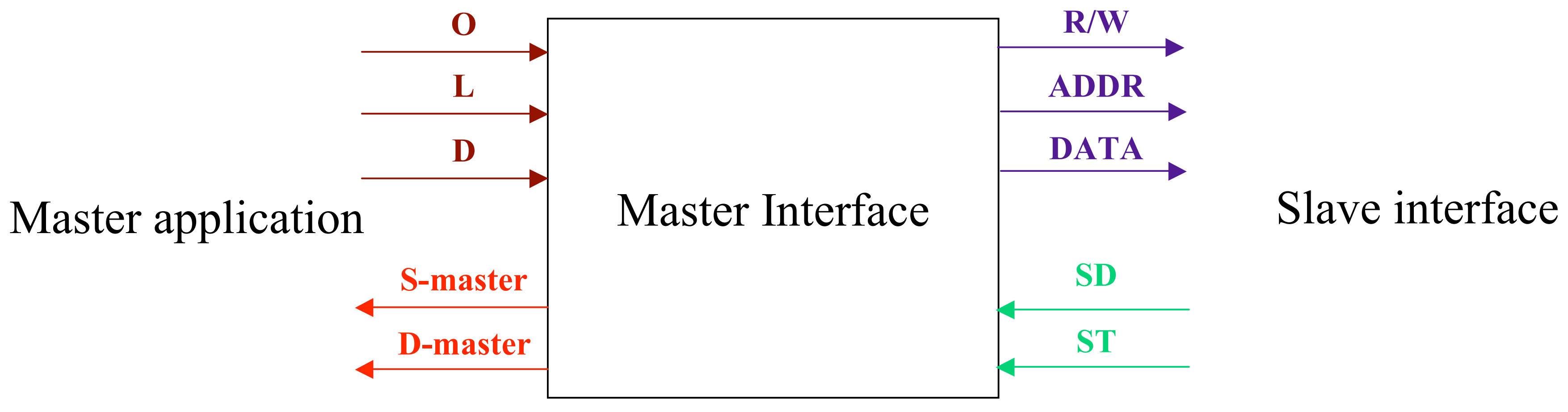


```
(defun master_interface (O L D   SD ST)
    (if  (equal  O  'read)
         (list  (list  1 L D)  (list SD ST))

         (list  (list  0 L D)  (list SD ST))))
```

(DATA x)

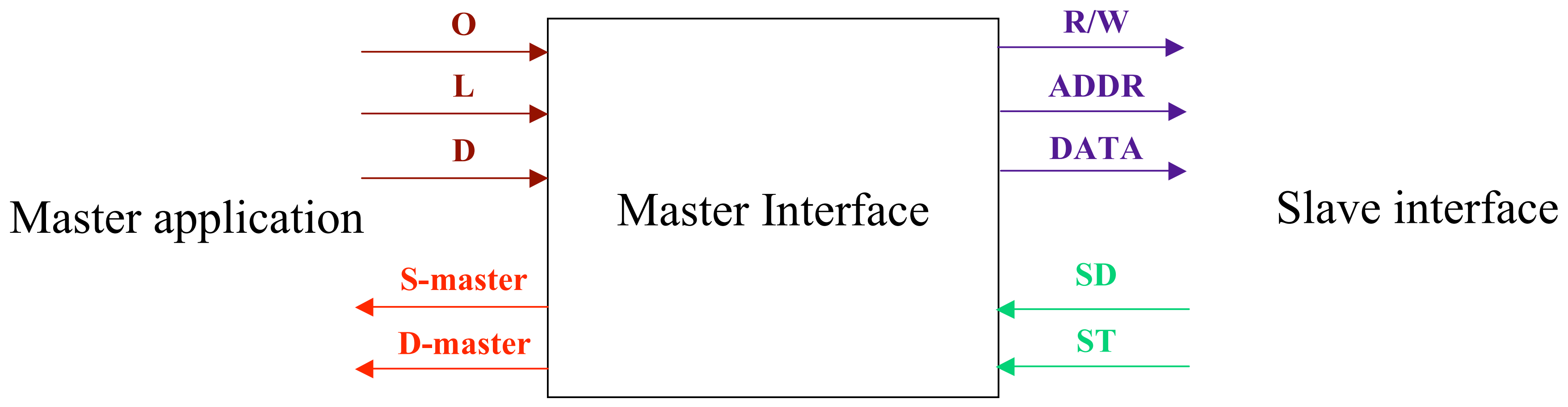
Modeling the master interface



```
(defun master_interface (O L D   SD ST)
    (if  (equal  O  'read)
         (list  (list  1 L D)  (list SD ST))
         (list  (list  0 L D)  (list SD ST))))
```

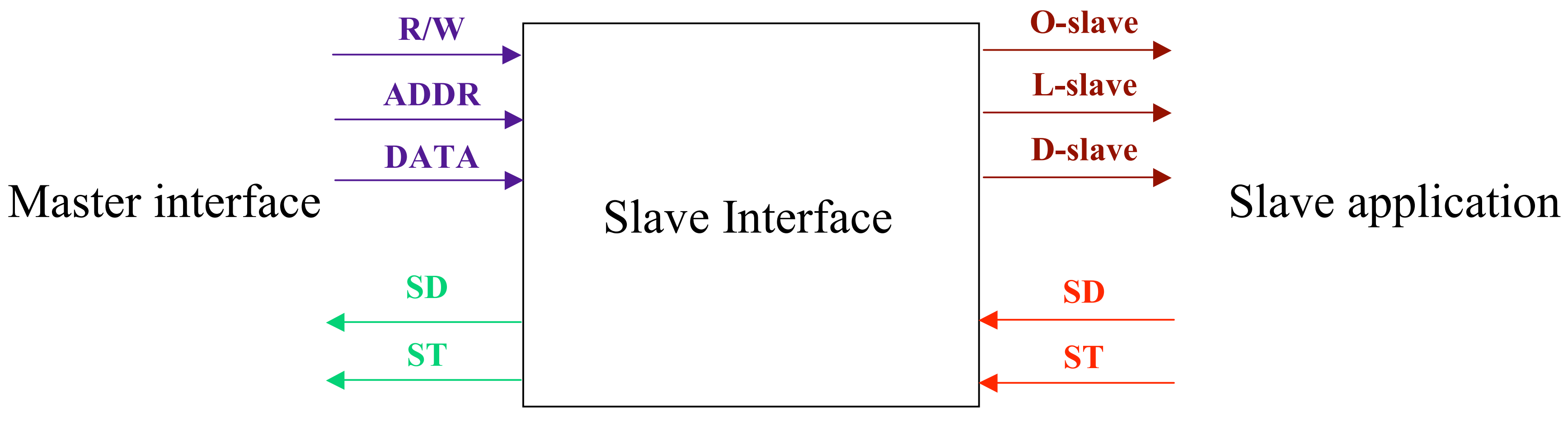
(D-master x)

Modeling the master interface



```
(defun master_interface (O L D   SD ST)
    (if  (equal  O  ‘read)
         (list  (list  1 L D)  (list SD ST))
         (list  (list  0 L D)  (list SD ST))))
                         (S-master x)
```

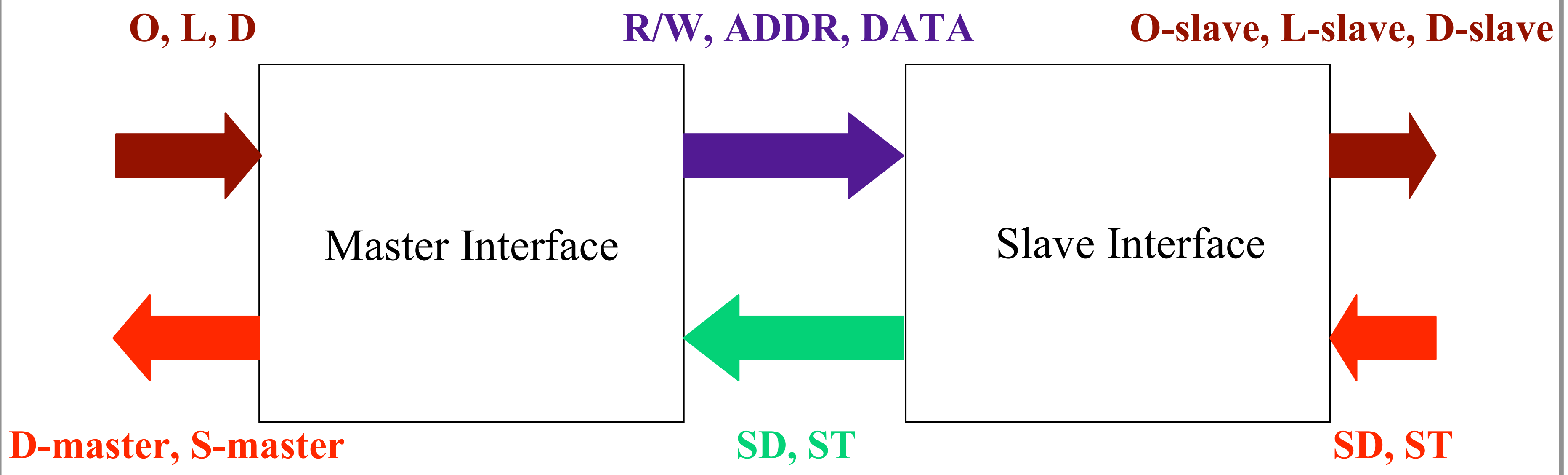
Modeling the slave interface



```
(defun slave_interface (R/W ADDR DATA   SD ST)
   (if  (equal  R/W  1)
        (list  (list  'read  ADDR DATA)  (list SD ST))
        (list  (list  'write ADDR DATA)  (list SD ST))))
```

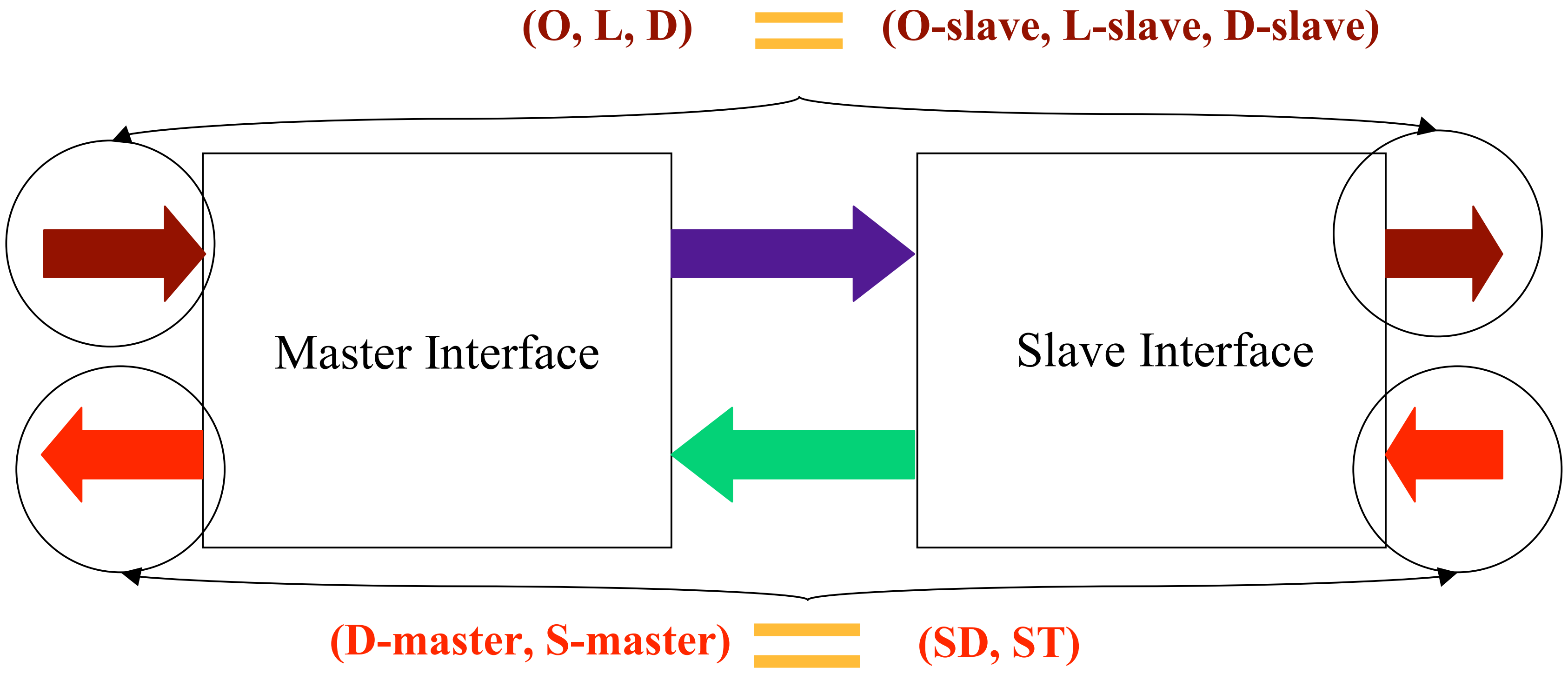
(O-slave x) (L-slave x) (D-slave x) (SD x) (ST x)

Transfers modeling



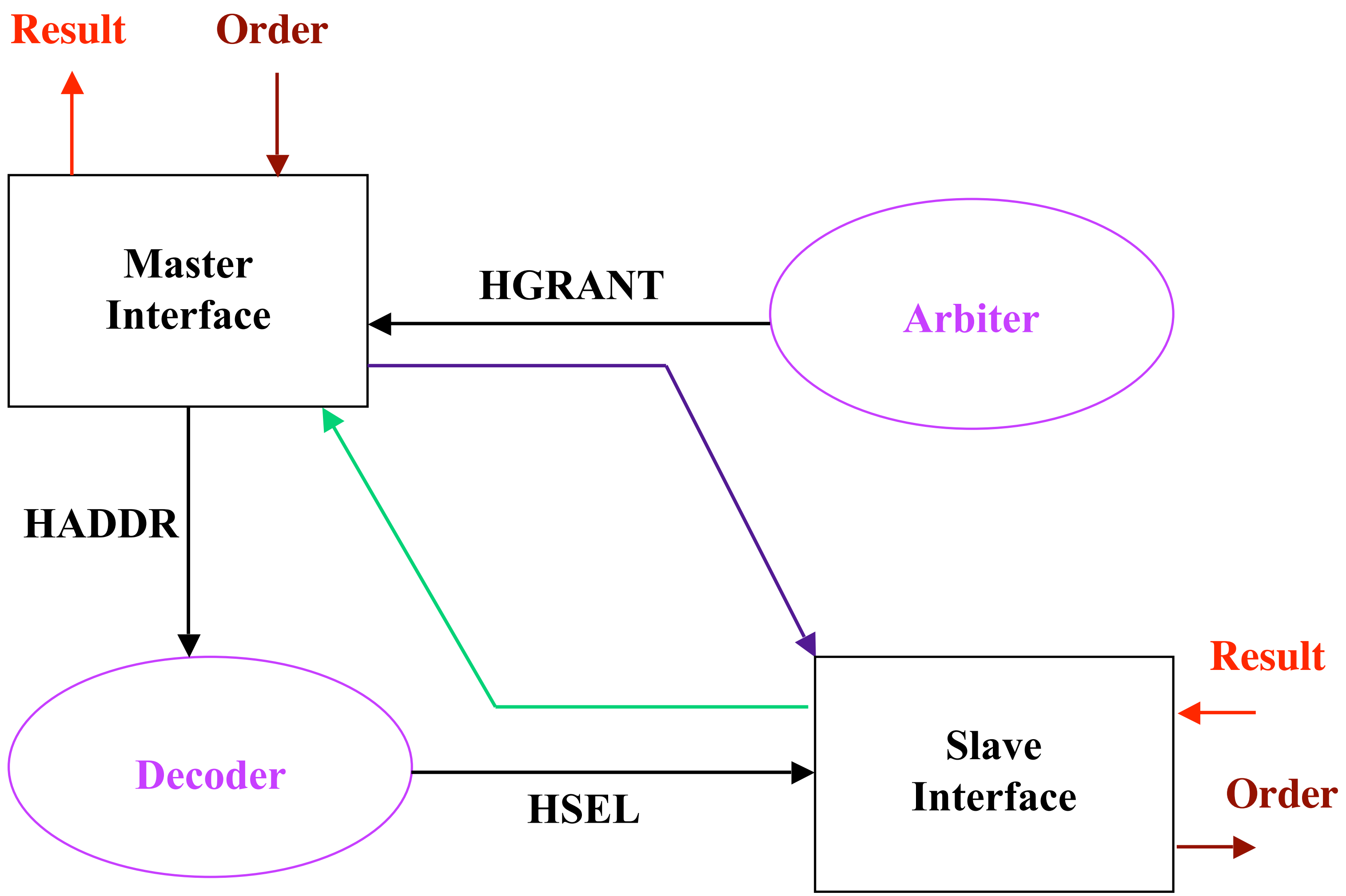
- Each transfer is modeled by a functional composition
  - trans_M_to_S (O L D) = slave_interface ● master_interface(O L D)
  - trans_S_to_M(SD ST) = master_interface ● slave_interface (SD ST)

Transfer validation



- Correctness Criteria
  - The introduction of the interfaces does not modify the orders nor the results

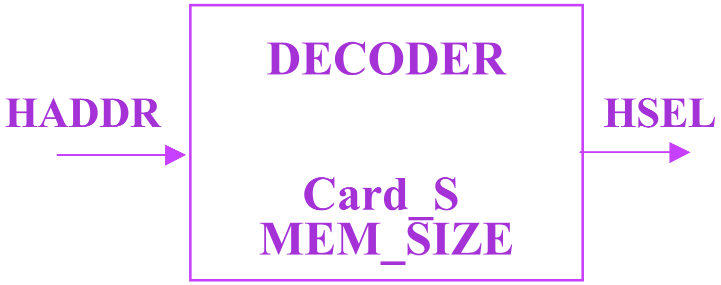
One step forward: the bus case



The proof strategy

- 1st: prove the decoder and the arbiter correct
  - Prove that the decoder selects the slave possessing the data required for the transfers
  - Prove that the arbiter grants the master with the highest priority
- 2nd: prove the point to point connection correct
  - This takes place between a well chosen master interface and a well chosen slave interface

Modeling the address decoder



HADDR < Card_S x MEM_SIZE

UNADDR = HADDR mod MEM_SIZE

SEL = HADDR / MEM_SIZE

```
(defun  select (Card_S SEL)
  (cond ((not (integerp Card_S)) nil)
               ((<= Card_S 0) nil)
               ((equal SEL 0)
                (cons 1
                (select (1- Card_S) (1- SEL))))
               (t
                (cons 0
                (select (1- Card_S) (1- SEL))))))
```

```
(defun decoder (MEM_SIZE Card_S  HADDR)
  (select Card_S (floor HADDR MEM_SIZE)))
```

Validation of the decoder function

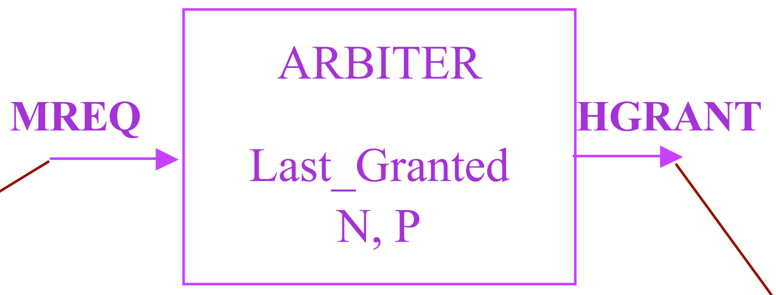
Selection of the right slave

```
(defthm ith_select_=_1
  (implies (and (integerp i) (integerp Card_S)
                           (>= i 0) (> Card_S i))
                 (equal (nth i (select Card_S i )) 1)))
```

Uniqueness of the selection

```
(defthm pth_select_=_0
  (implies (and (integerp p) (integerp Card_S)
                           (<= 0 p) (< p Card_S)
                           (not (equal p i)))
                 (equal (nth p (select Card_S i)) 0))
  :hints (("GOAL"
                 :induct (function_hint_th2_select p Card_S i))))
```

Modeling the bus arbiter



MREQ is a matrix with
P lines and N columns

HGRANT is a list of bits computed according to a priority scheme

	N0	N1	N2	N3
P0	0	0	0	0
P1	1	0	1	1
P2	1	1	1	1

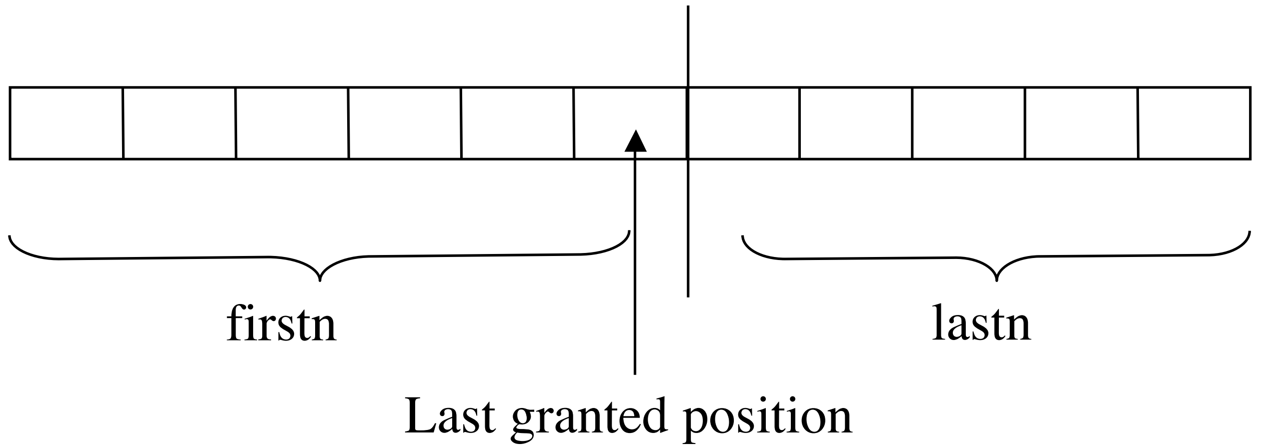
First Step: RLINE search

```
(defun stage_P (L)                       ; returns the line number
  (cond ((endp L) 0)                     ; for the highest priority request
        ((no_requestp_matrix L) 0)       ; if empty list or no request returns 0
        ((not (no_requestp (car L))) 0) ; we count the number of stages containing
        (t                               ; no request until we meet a stage with
         (+ 1 (stage_P (cdr L))))))      ; at least one request


(defthm prior_scheme                     ; we prove that each stage j prior to the
  (implies (and (equal (stage_P L) i) ; returned one i contains no request
                (< j i) (<= 0 j))
           (no_requestp (nth j L))))
```

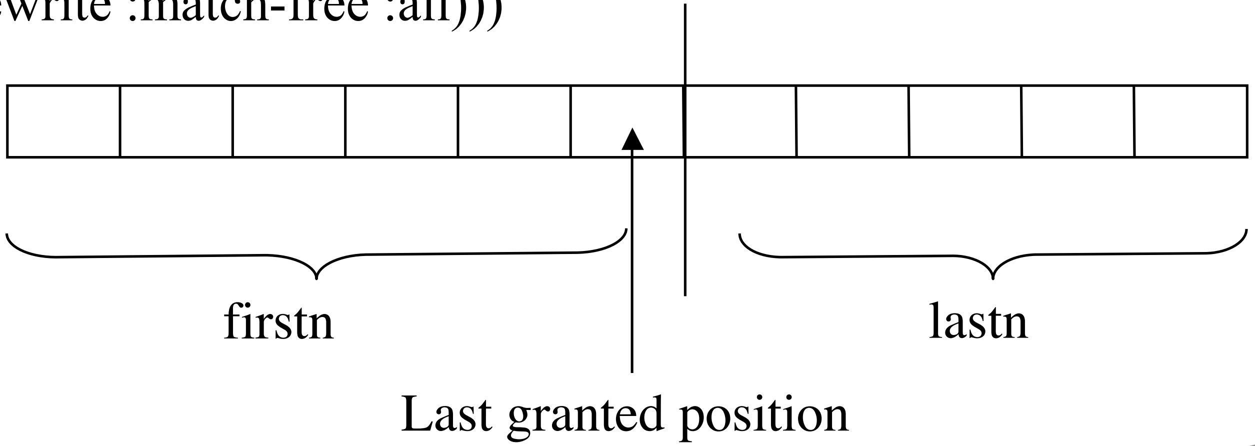
2nd Step: who is the next owner?

```
(defun round_robin (RLINE Last_Granted)
 (cond ((no_requestp RLINE) 0)
           ((no_requestp (lastn (1+ Last_Granted) RLINE))
            (find_next_1 (firstn (1+ Last_Granted) RLINE)))
           (t
            (+ (1+ Last_Granted)
                 (find_next_1 (lastn (1+ Last_Granted) RLINE))))))
```



2nd Step: validation

```
(defthm no_deadlock
  (implies (and (integerp i)
                      (<= 0 i)
                      (equal (nth Last_Granted RLINE) 1)
                      (list_of_1_and_0 RLINE)
                      (not (equal Last_granted i)))
                (implies (equal (nth i RLINE) 1)
                     (not (equal (round_robin RLINE Last_Granted) Last_Granted))))
  :hints (("GOAL" :use lemma1_no_deadlock
                          :in-theory (disable lemma1_no_deadlock firstn)))
  :rule-classes ((:rewrite :match-free :all)))
```



3rd Step: compute and build

	N0	N1	N2	N3
P0	0	0	0	0
P1	1	0	1	1
P2	1	1	1	1

Compute the number of the granted master

```
(defun master_num (MREQ N Last_Granted)
  (+ (* (stage_P MREQ) N)
       (round_robin (nth (stage_P MREQ) MREQ) Last_Granted)))
```

Build the output list HGRANT

```
(defun arbiter (N P MREQ Last_Granted)
      (select (* N P) (master_num MREQ N Last_Granted)))
```

Proof of transfers

- 1st: decoder and arbiter, OK

- 2nd: Prove the point to point connection correct
  - Trans_M_to_S(O L D) = (O (mod L MEM_SIZE) D)
  - Trans_S_to_M(SD ST) = (SD ST)

Conclusion and Future Work

- **Conclusion**
  - 20 functions, 60 theorems, proof time about 30 seconds
  - Protocol proven correct for an arbitrary number of masters and slaves
- **Future Work**
  - Modeling on chip networks
  - Test implementations against the formal specification