A Generic Implementation Model for the Formal Verification of Networks-on-Chips

Tom van den Broek and Julien Schmaltz

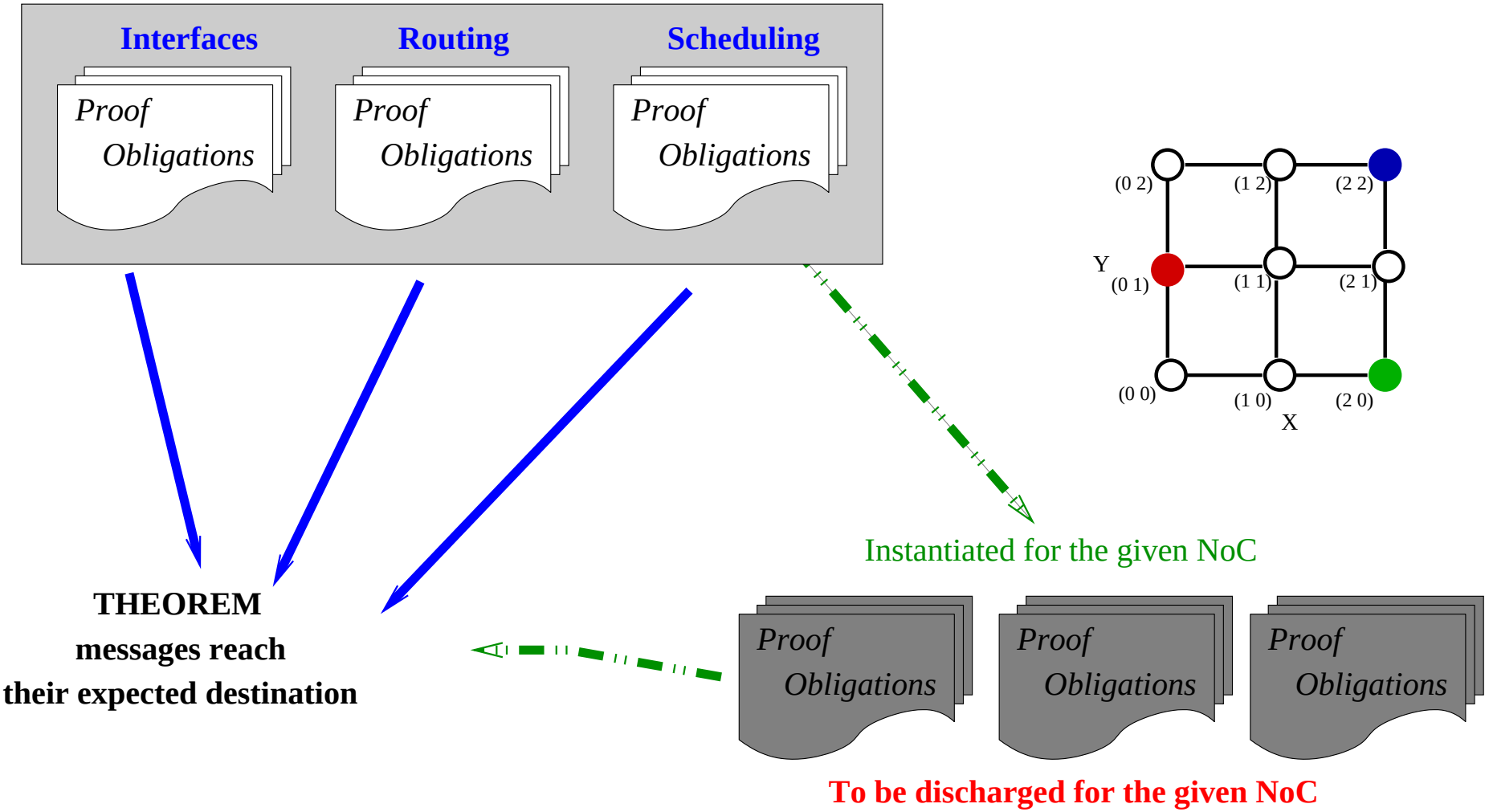
Institute for Computing and Information Sciences
Radboud University Nijmegen
The Netherlands
tombroek@science.ru.nl and julien@cs.ru.nl

May 12, 2009

Platform-Based Design and Networks on a Chip

- Platform-Based Design:
  - Re-use of parameterized modules (*Intellectual Properties*)
  - High-level of abstraction
  - *Communication*-centric: from buses to networks
- System Verification:
  - Proof of each component
  - Proof of their interconnection
- State-of-the-Art:
  - Model checking or theorem proving of instances of systems
  - Often at RTL and below
- The GeNoC Approach
  - A generic model for reasoning about NoCs

The GeNoC Approach

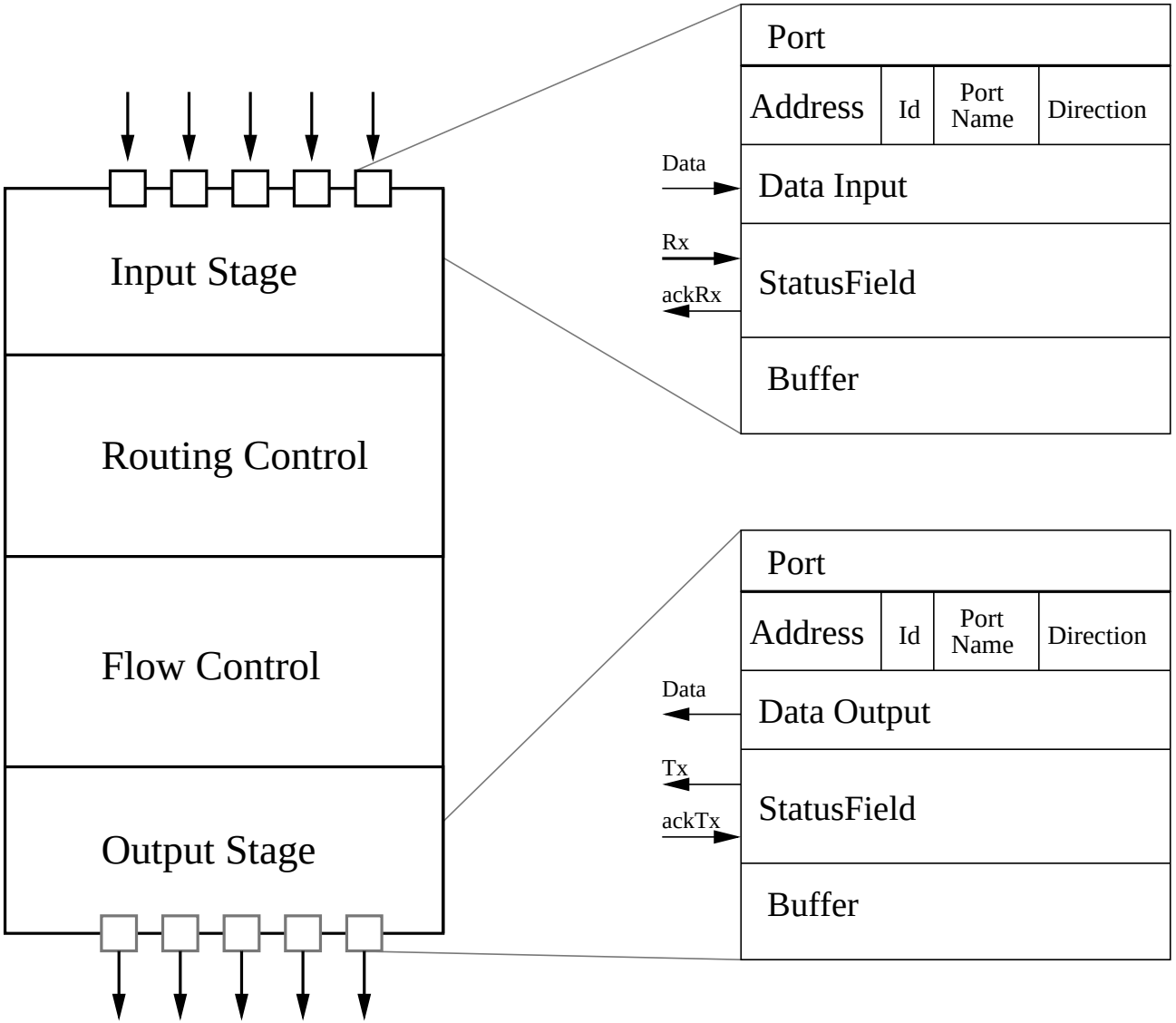


Our Contribution

- Current GeNoC Model
  - Developed together with A. Helmy, L. Pierre and D. Borrione
  - Abstract representation of the communications
  - Global generic properties (functional correctness, deadlock prevention)
  - Efficient framework for NoCs specification
  - Not sure whether it can be related to actual implementations ...
- This talk will be about ...
  - A generic implementation model *à la GeNoC*
  - A preview on a refinement proof

Outline

Implementation level – Generic Router



The Generic Router

```
( defun Router (nst nstmem)
    (mv-let
      (nst nstmem)
      (RouteControl (ProcessInputs nst) nstmem)
      (mv-let (nst nstmem)
              (Flowcontrol nst nstmem)
              (mv (ProcessOutputs nst) nstmem))))
```

Network Interpreter

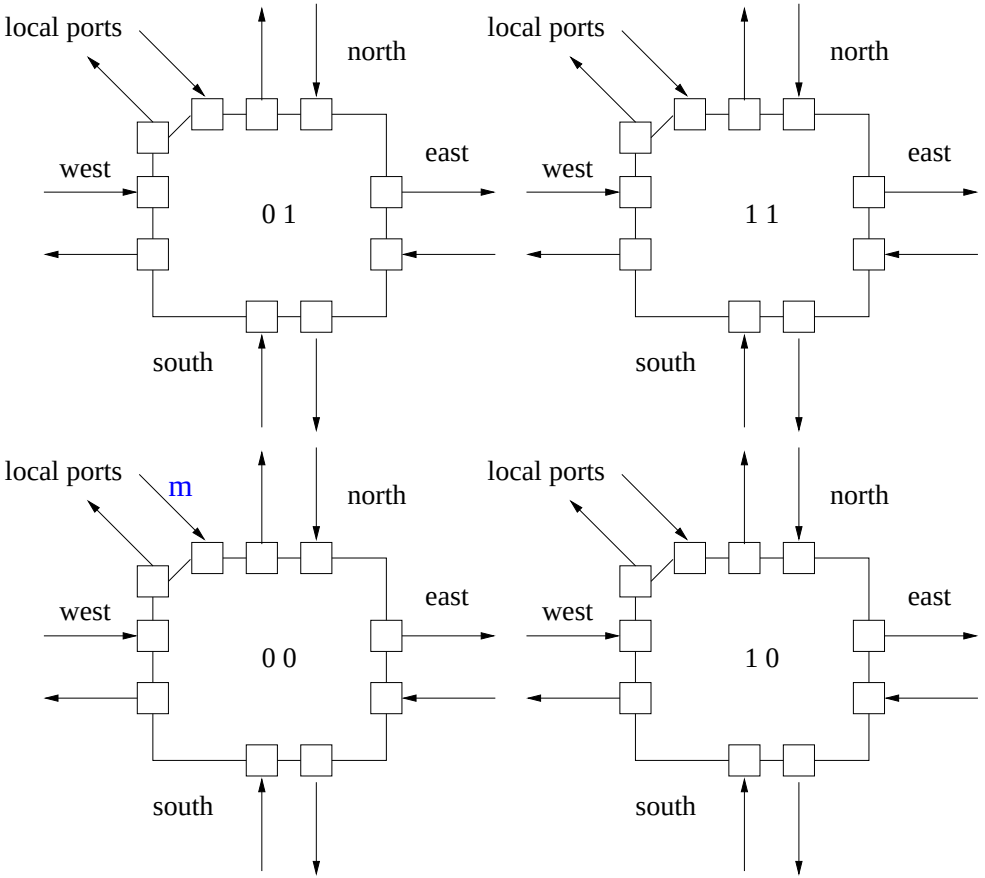
- Basic components
  - Basic elements are ports
  - A node is a set of ports (with same coordinate/address)
  - Global network state = current state of all ports
- Update each node of the network one by one
- Copy output data line to input data line
- Copy signals (e.g., "Tx", "Rx","AclkTX")

Message injection – depart (1)

```
(mv-let
  (dep del) ;; dep = new value of ntkst
  (depart ntkst m z)
```

- A list of messages (m)
- A network state (ntkst)
- Current simulation step (z)
- Update network state with injected messages
- Rest of messages are delayed (del)

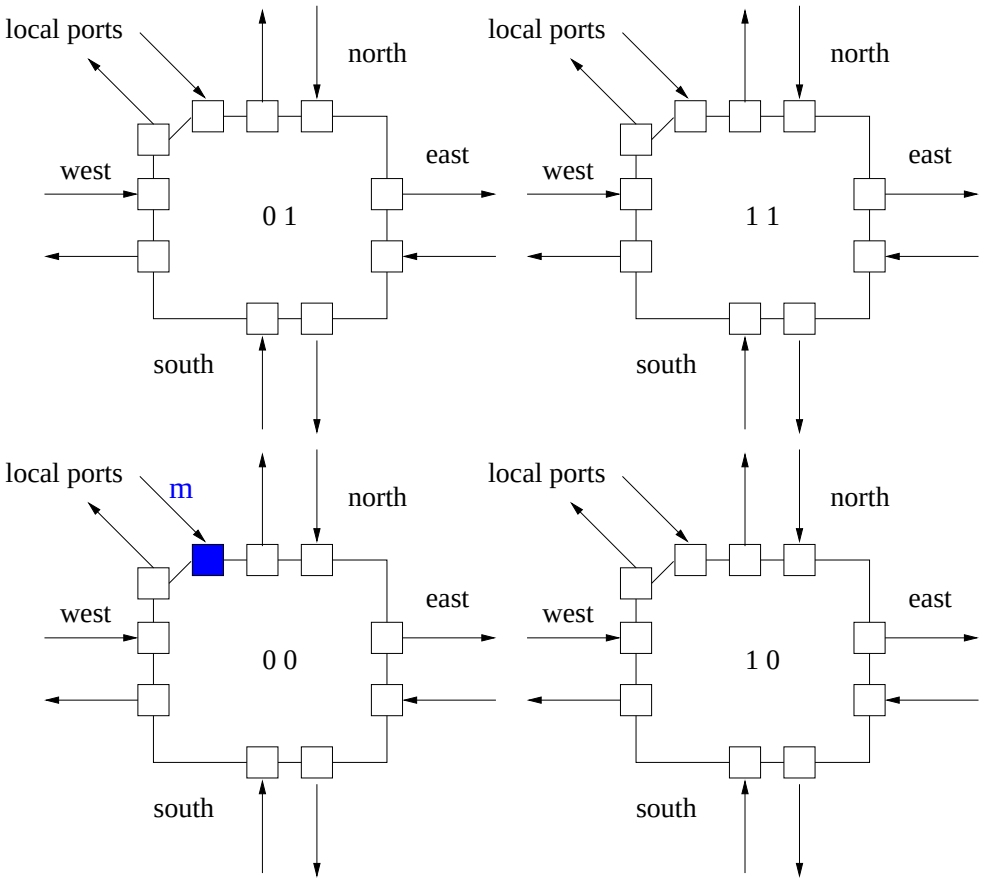
Message injection – depart (2)




```
(mv-let
  (dep del)
  (depart ntkst m z)
```

- m.d = 11
- m.z = 0

Message injection – depart (2)




```
(mv-let
   (dep del)
   (depart ntkst m z)
```

- m.d = 11
- m.z = 0

Network step – step-ntk

```
(mv-let
  (dep del) ;; dep = new value of ntkst
  (depart ntkst m z)
  (let
     ((newntkst (step-ntk dep topo))
```

- A network state (dep)
- The topology (topo)
- Update each node using function router
- Update data line and signals (updateNeighbours)

A network step (1)

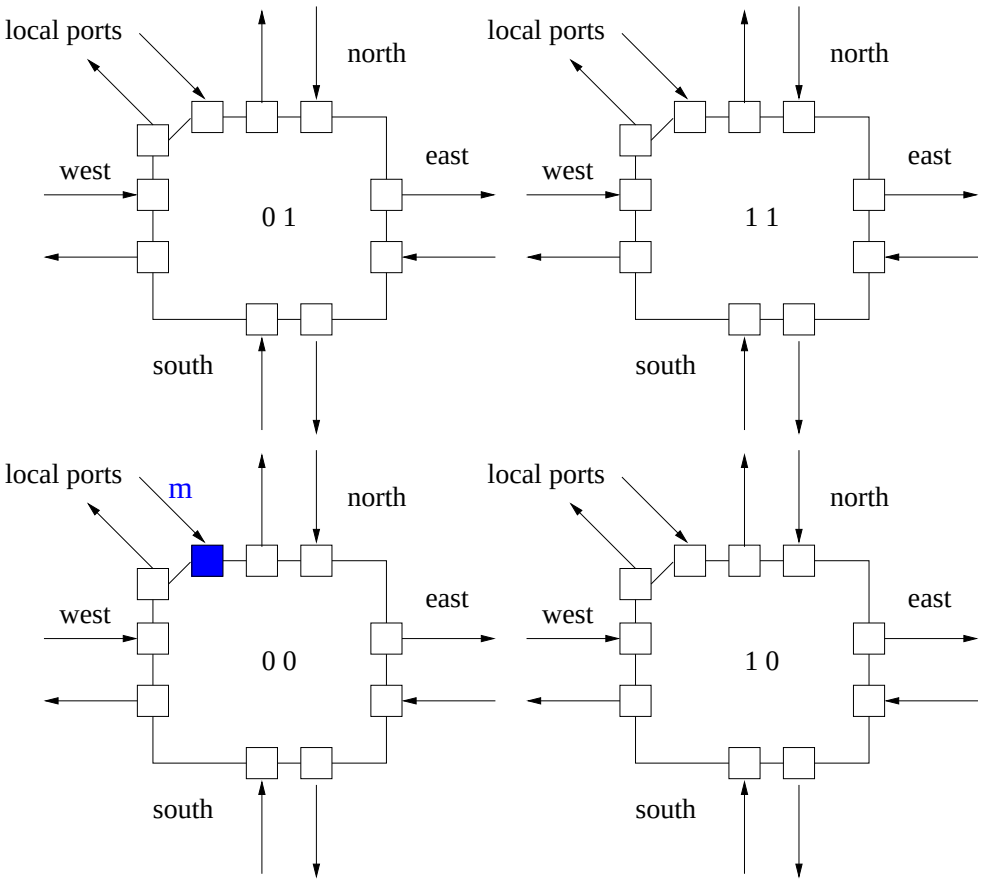
```
(defun step-ntk (ntkst topology)
  (let
   ((newntkst
    (step-ntk1 (ports-nodelist nktst nil)
               ntkst)))
   (updateNeighbours newntkst topology)))
```

A network step (2)

where

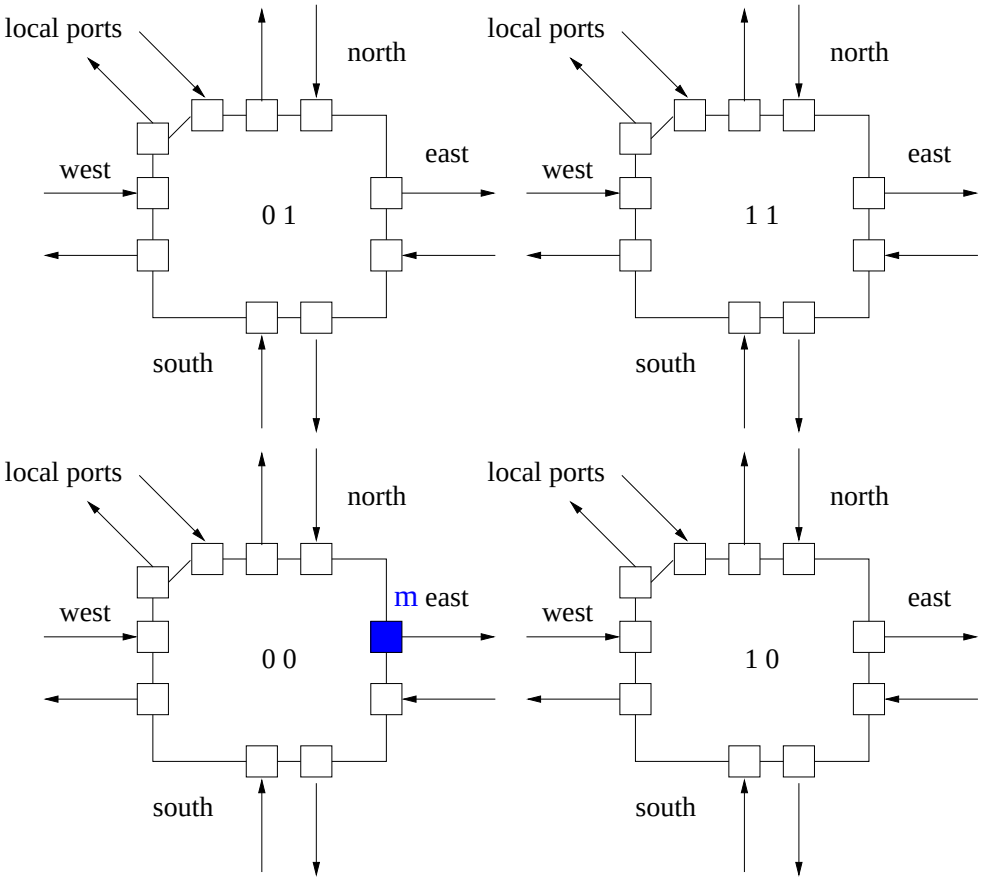
```
(defun step-ntk1 (ntslist ntkst)
  (if (endp ntslist)
    ntkst
    (let*
     ((newnst  (router (car ntslist)))
      (newntkst
       (step-ntk1 (cdr ntslist) ntkst)))
     (ports-update newntkst newnst))))
```

Network interpreter – Example




```
(mv-let
  (dep del)
  (depart ntkst m z)
  (let
     ((newntkst
      (step-ntk dep topo))
```

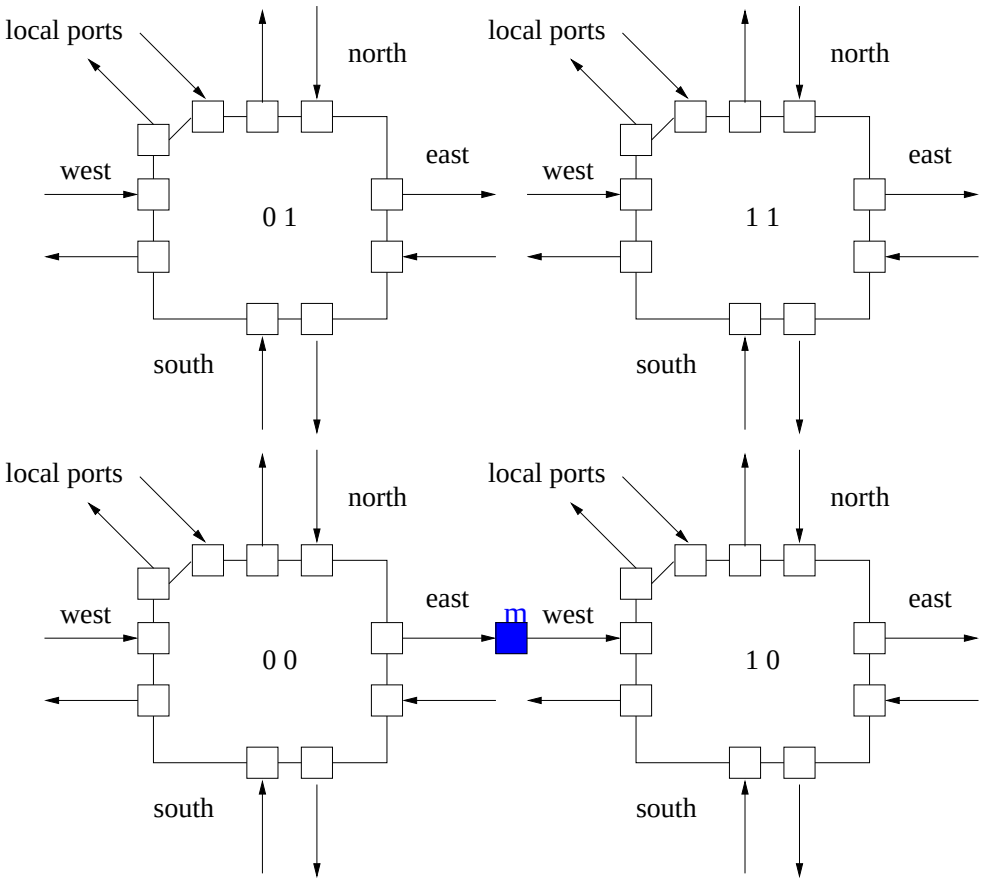
Network interpreter – Example




```
(mv-let
  (dep del)
  (depart ntkst m z)
  (let
     ((newntkst
      (step-ntk dep topo))
```

router

Network interpreter – Example

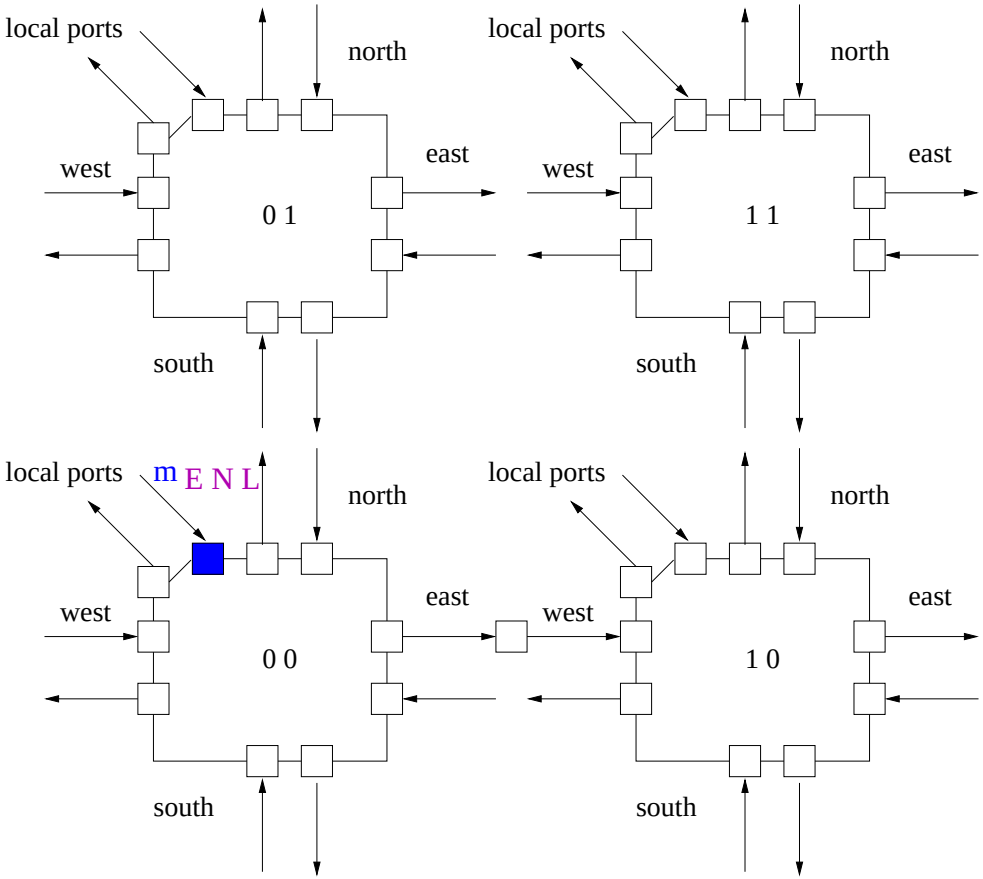



```
(mv-let
  (dep del)
  (depart ntkst m z)
  (let
     ((newntkst
      (step-ntk dep topo))
```

updateNeighbours

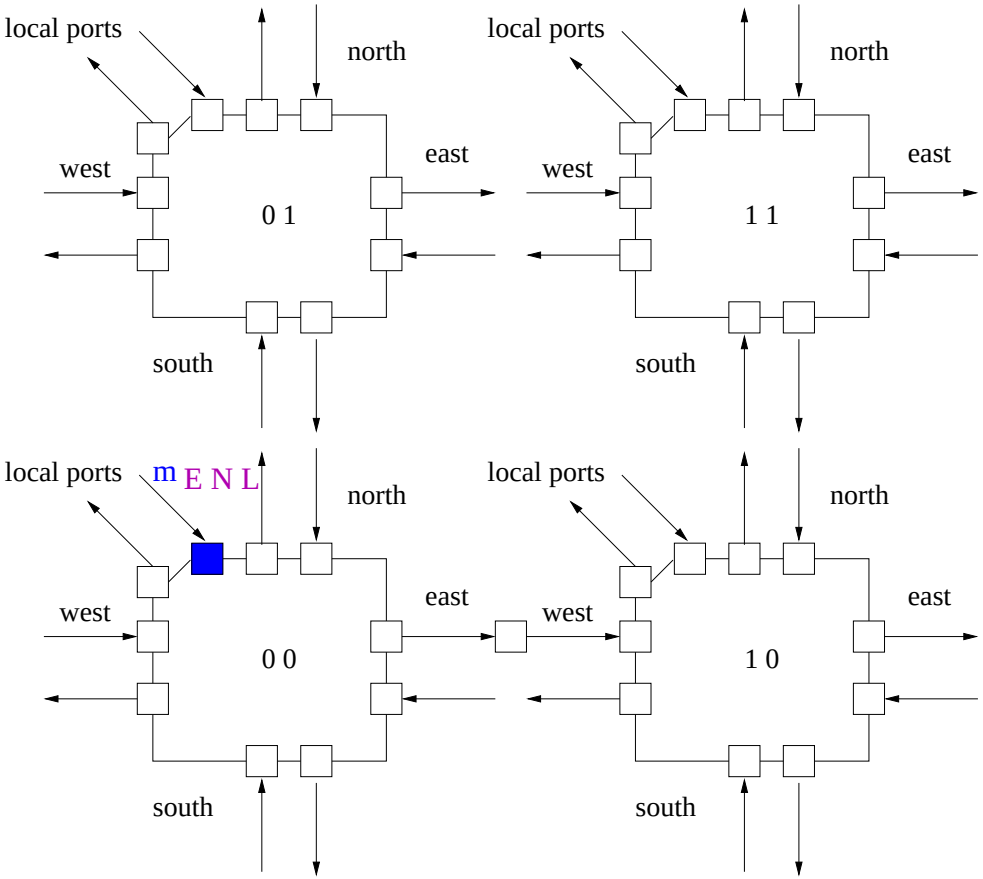
Outline

Specification level



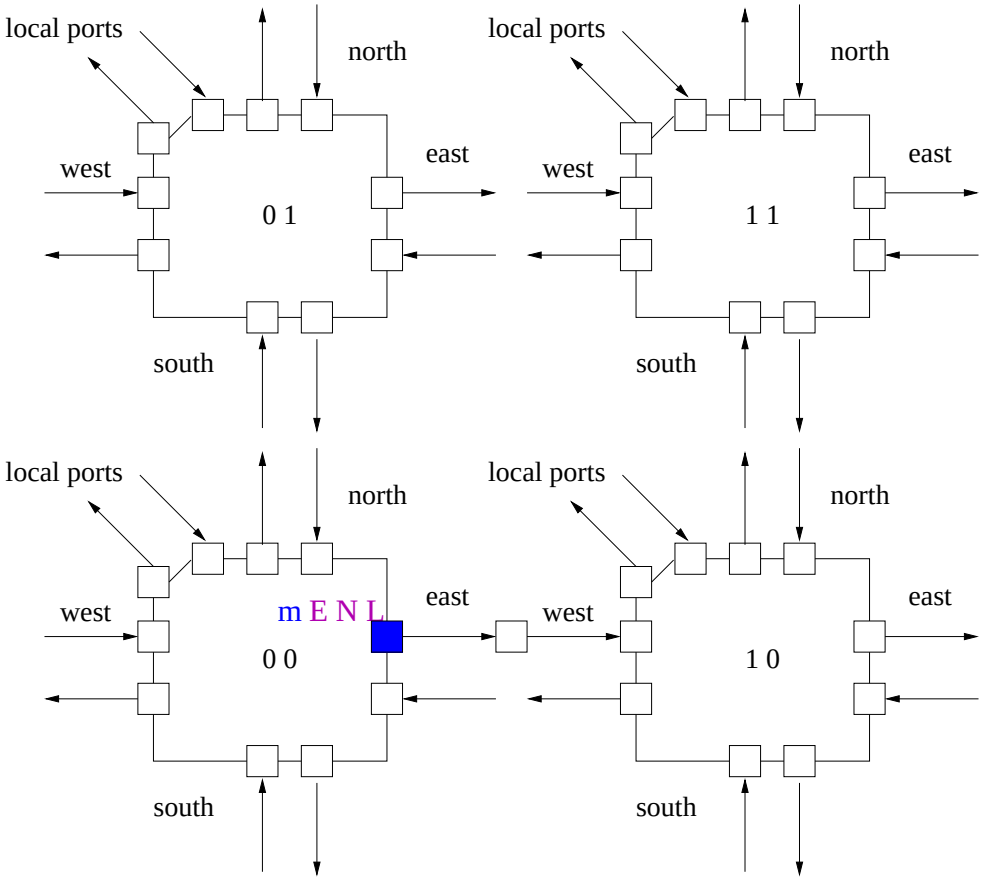
Route computed at injection time

Specification level – RouteControl



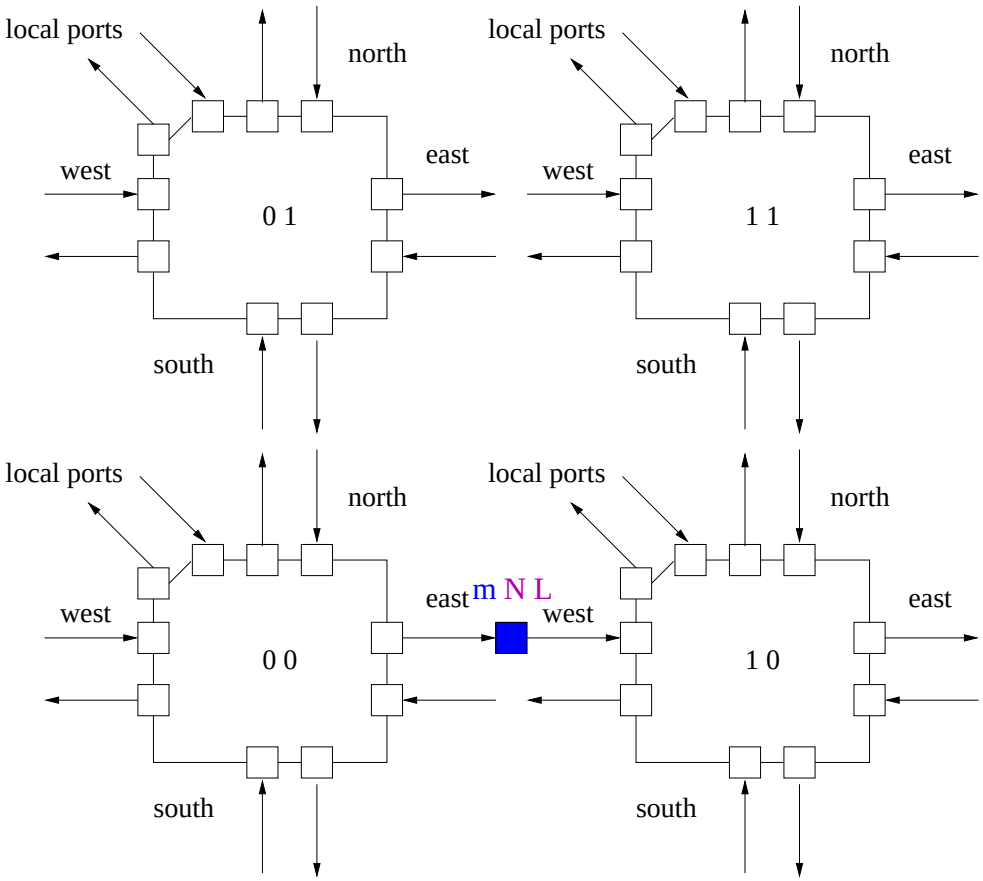
Routing = "(car R)"

Specification level – RouteControl



Routing = "(car R)"

Specification level – updateNeighbours



new route = "(cdr R)"

Refinement theorem (1)

```
(defthm eq-genoc
  (implies (and (buffer-p transactions)
                (natp param))
           (correct-genoc
             (fake-genoc transactions param timelimit)
             (serial-genoc transactions param timelimit))))
```

Refinement theorem (2) – transform

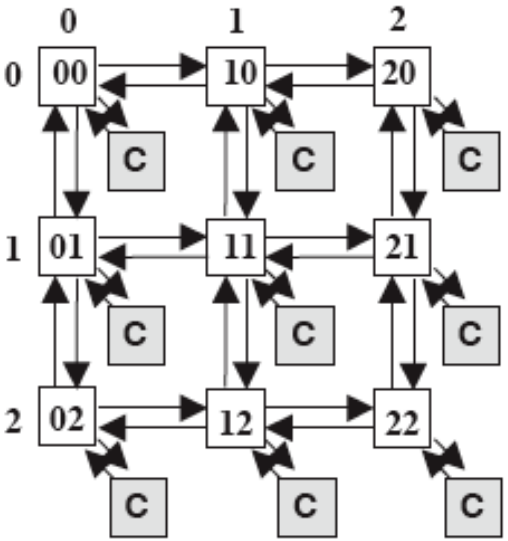
```
(defun correct-genoc (x y)
  (and
       ;; arrived messages are the same
       (equal (trans-arrived (mv-nth 0 x)) (mv-nth 0  y) )
       ;; network state is the same
       (equal (mv-nth 1 x) (mv-nth 1 y))
       ;; simulation outputs are the same
       (equal (trans-accup (mv-nth 2 x)) (mv-nth 2  y))))
```

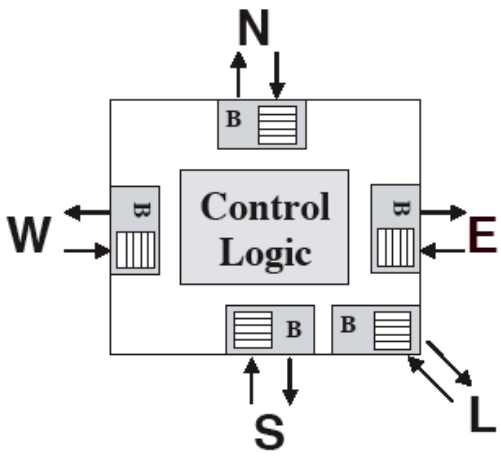
Conclusion and Future Work

- "Done"
  - A realistic generic router model (implementation level)
  - Demonstrated several instances of this generic router
    - Packet, circuit, and wormhole switching
    - XY and Spidergon routing
- "To Do"
  - Relation with initial GeNoC specification model
  - We already have initial results for one particular instance
  - Link with RTL designs
  - Link with our work on deadlock prevention (see next talk)

Appendix

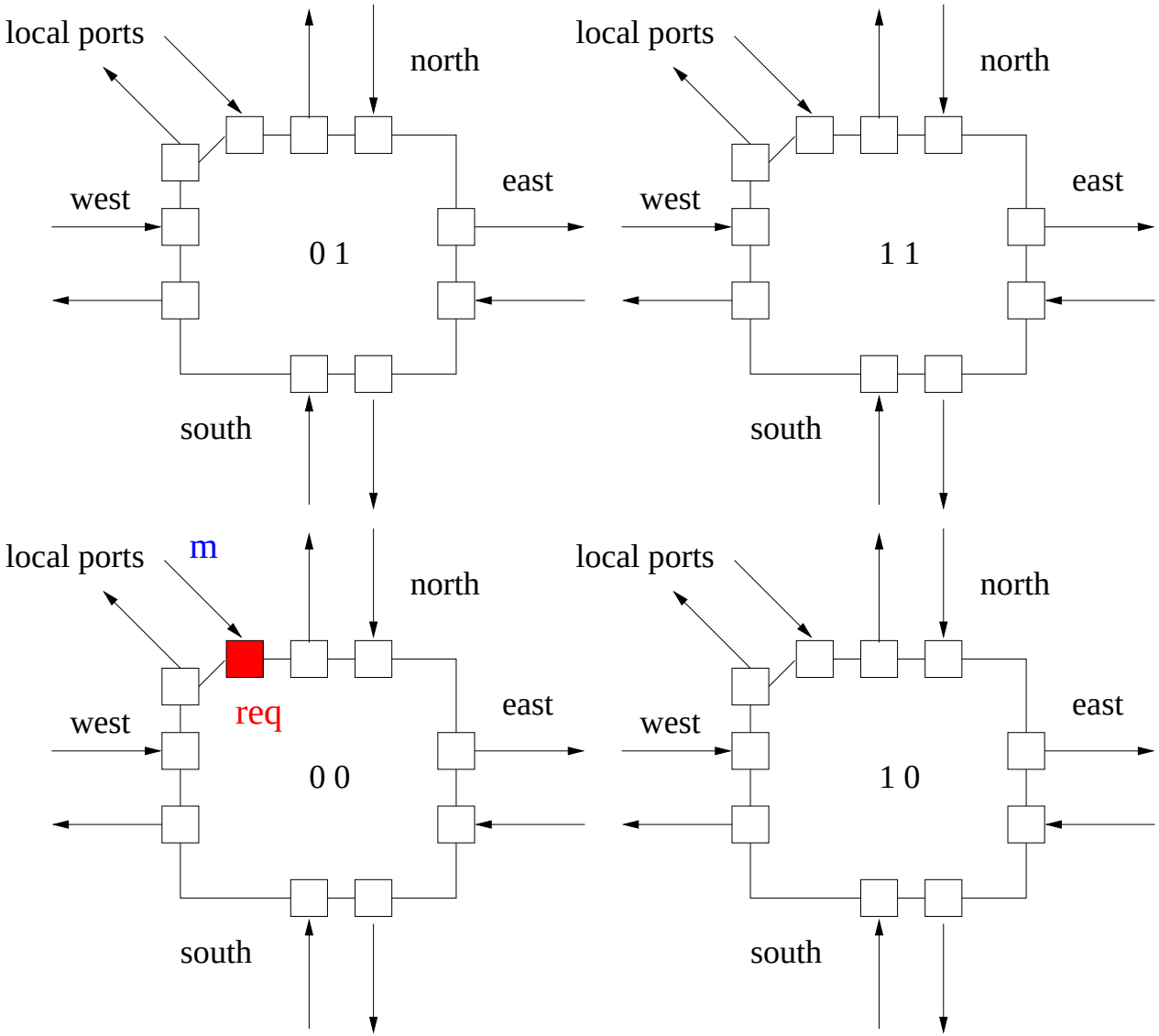
Networks-on-Chips: Hermes



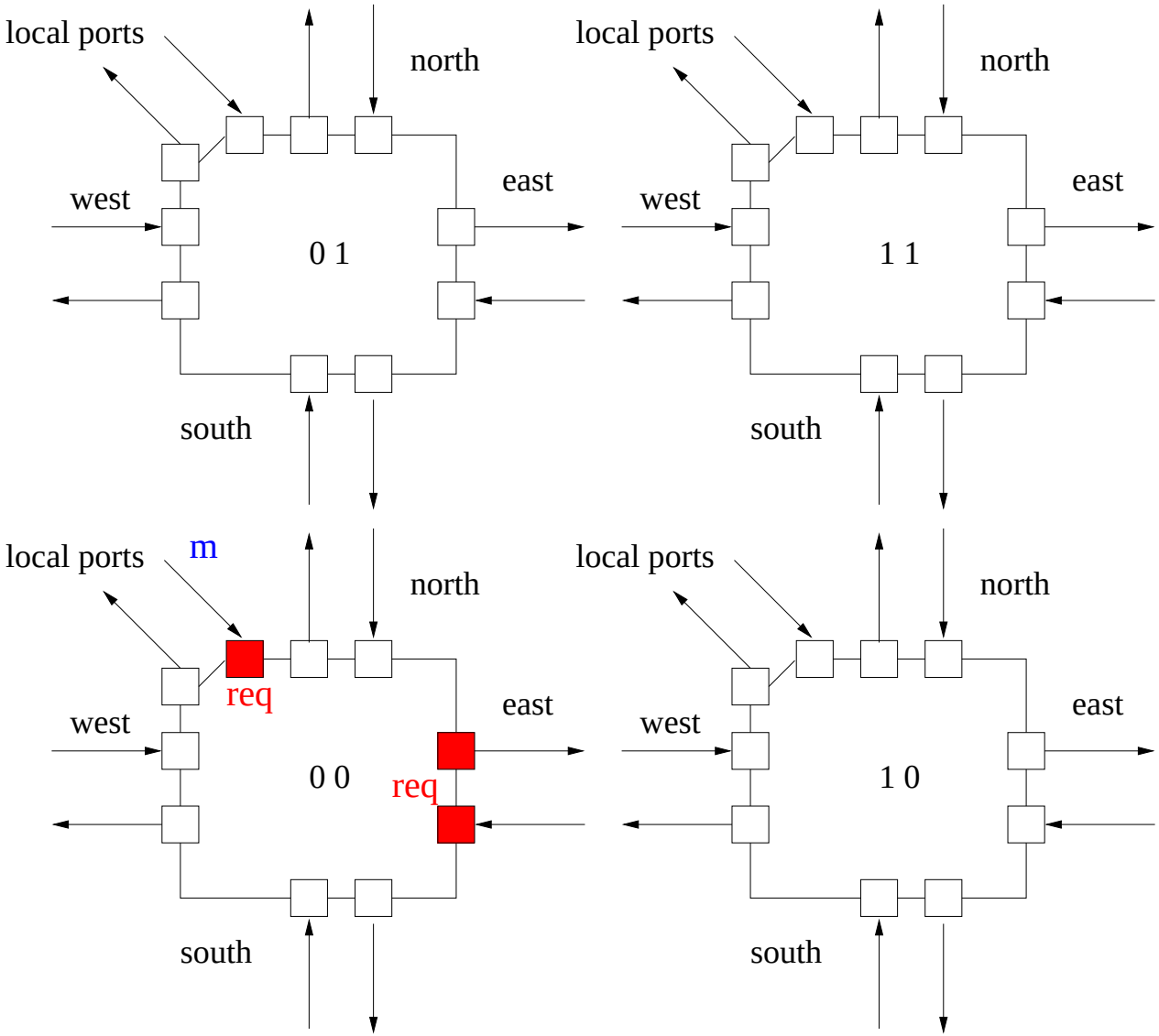


- XY minimal deterministic routing
- Wormhole switching
- Frame structure:
  - Header flit (Route Information)
  - Control flit (Number of Flits)
  - Data flits (Payload)

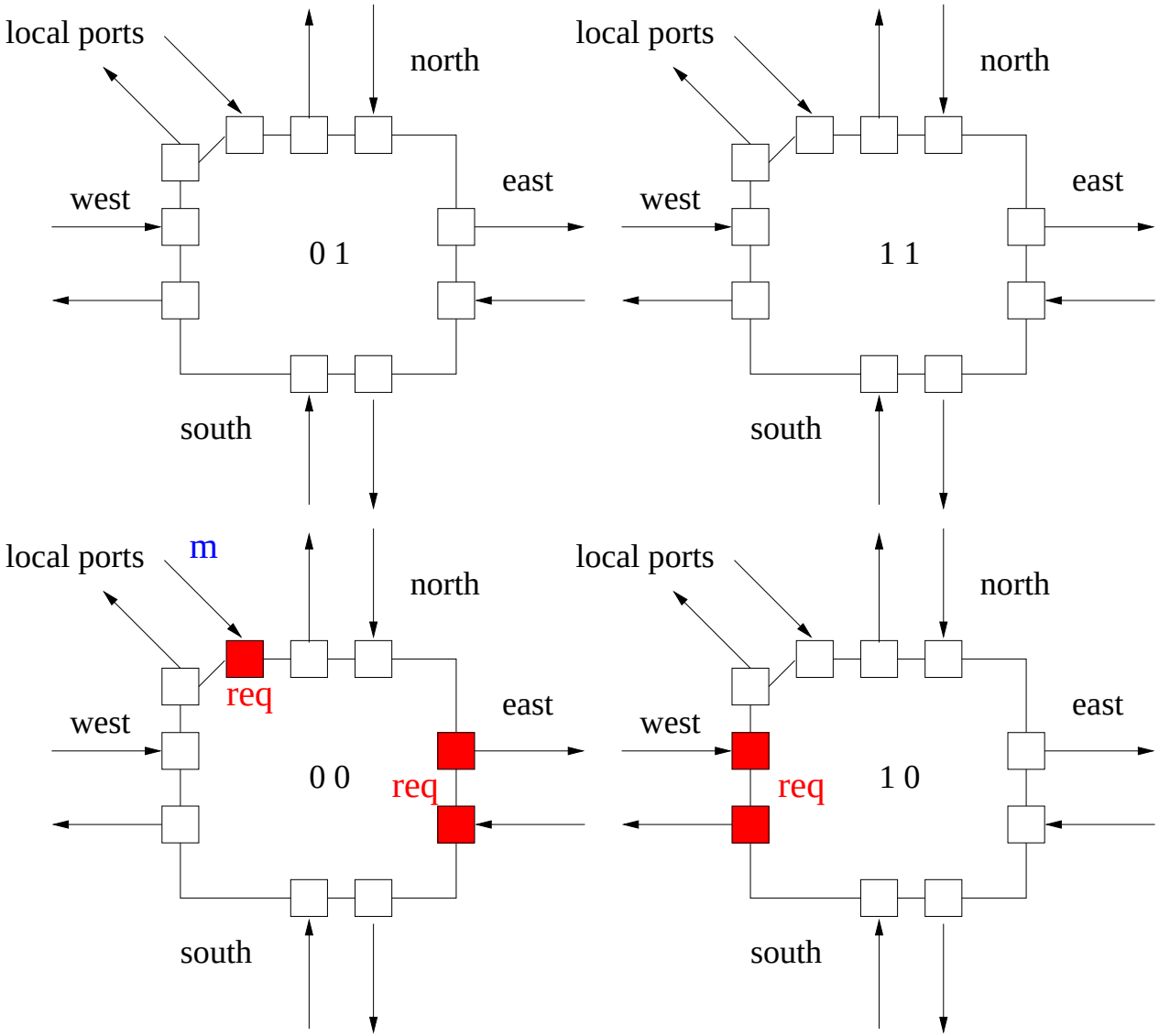
Circuit Switching and XY Routing



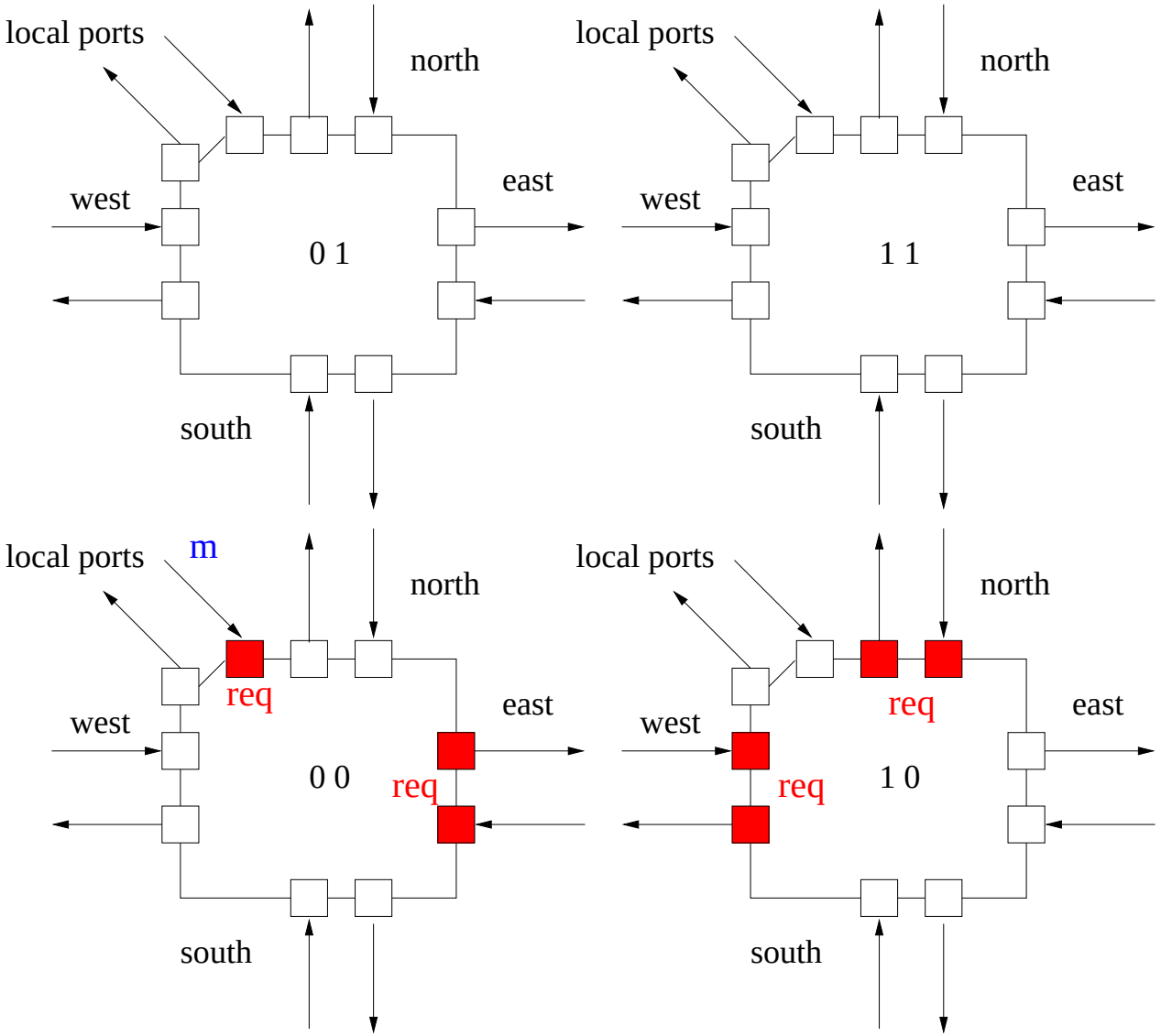
Circuit Switching and XY Routing



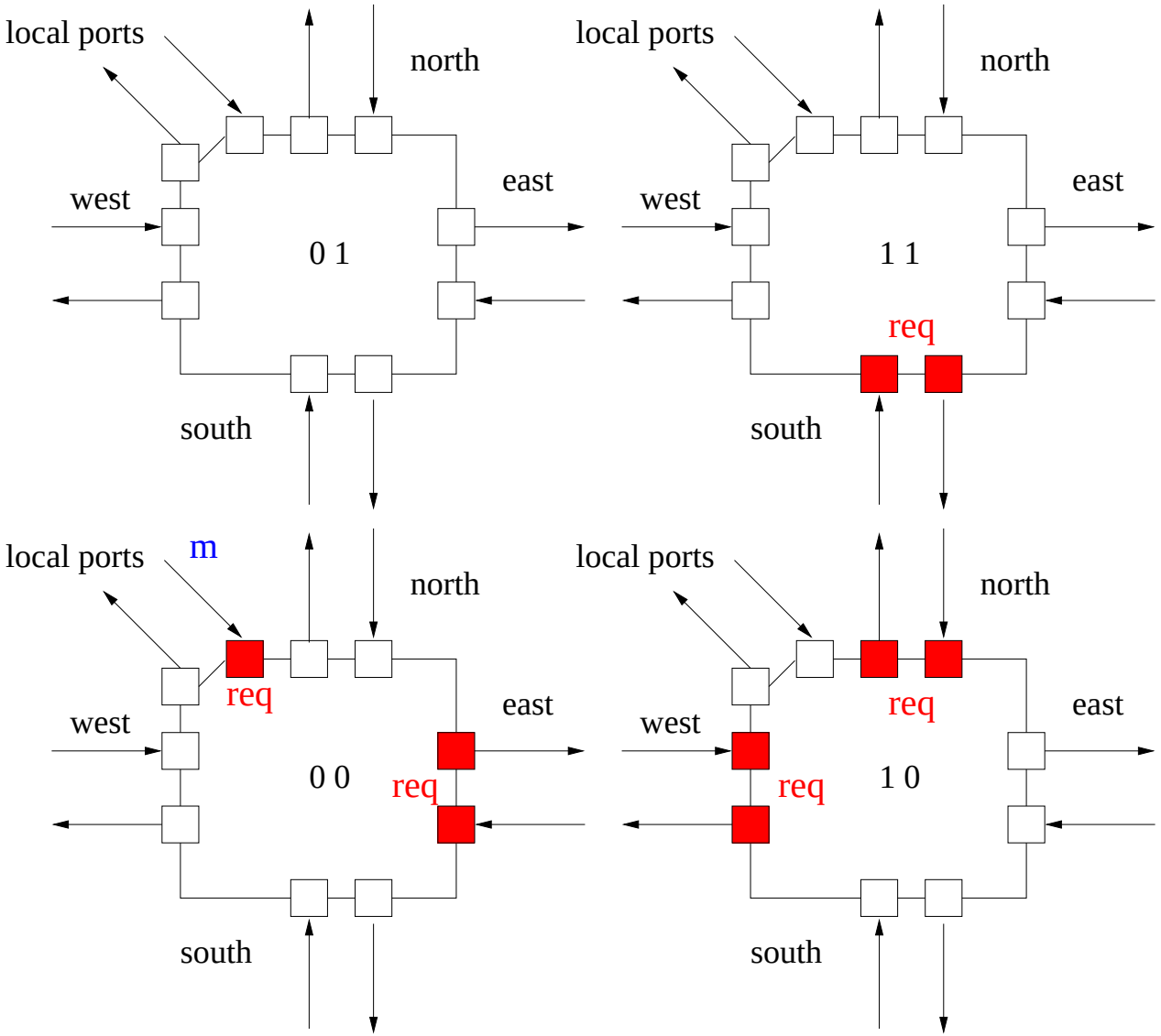
Circuit Switching and XY Routing



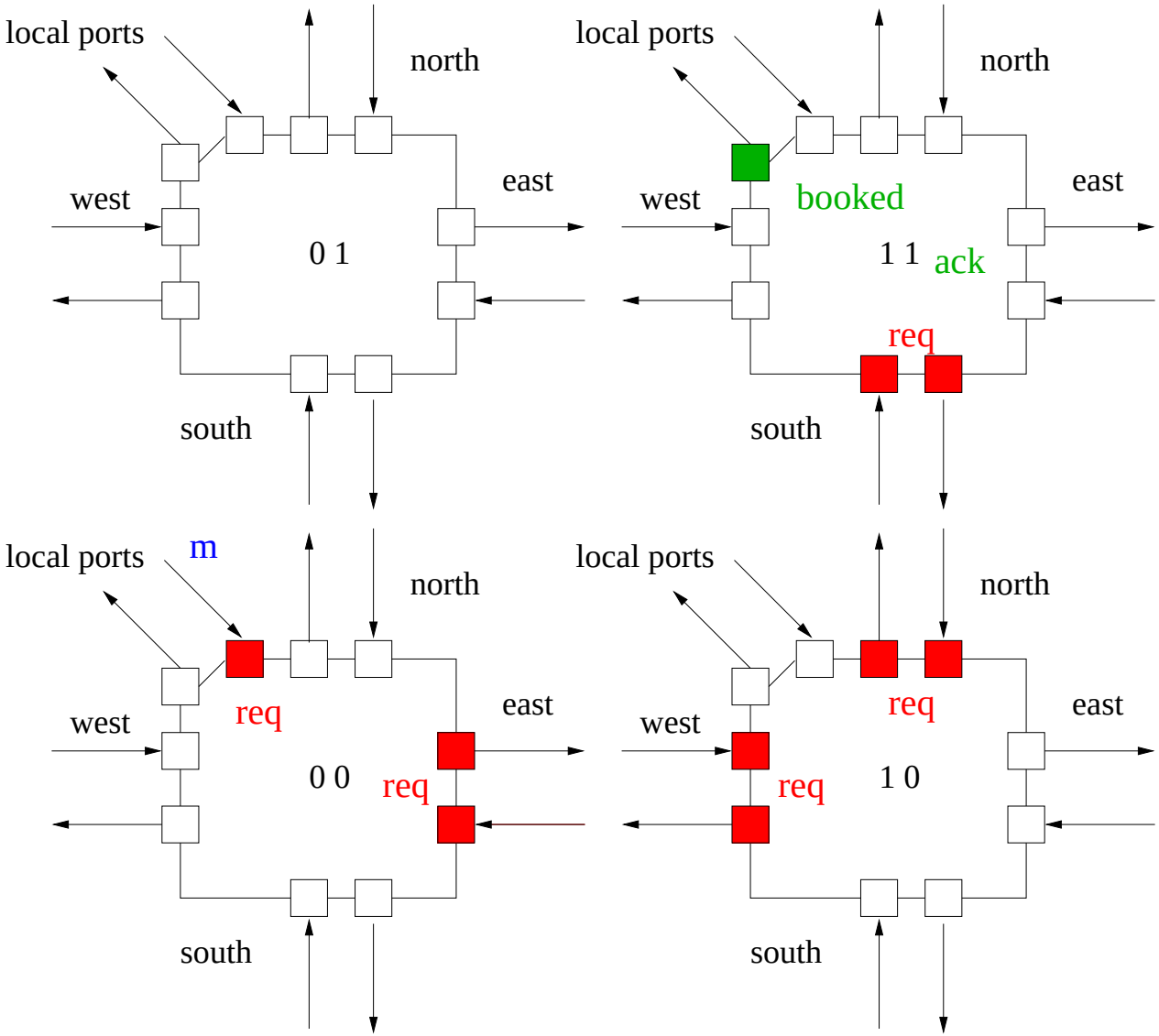
Circuit Switching and XY Routing



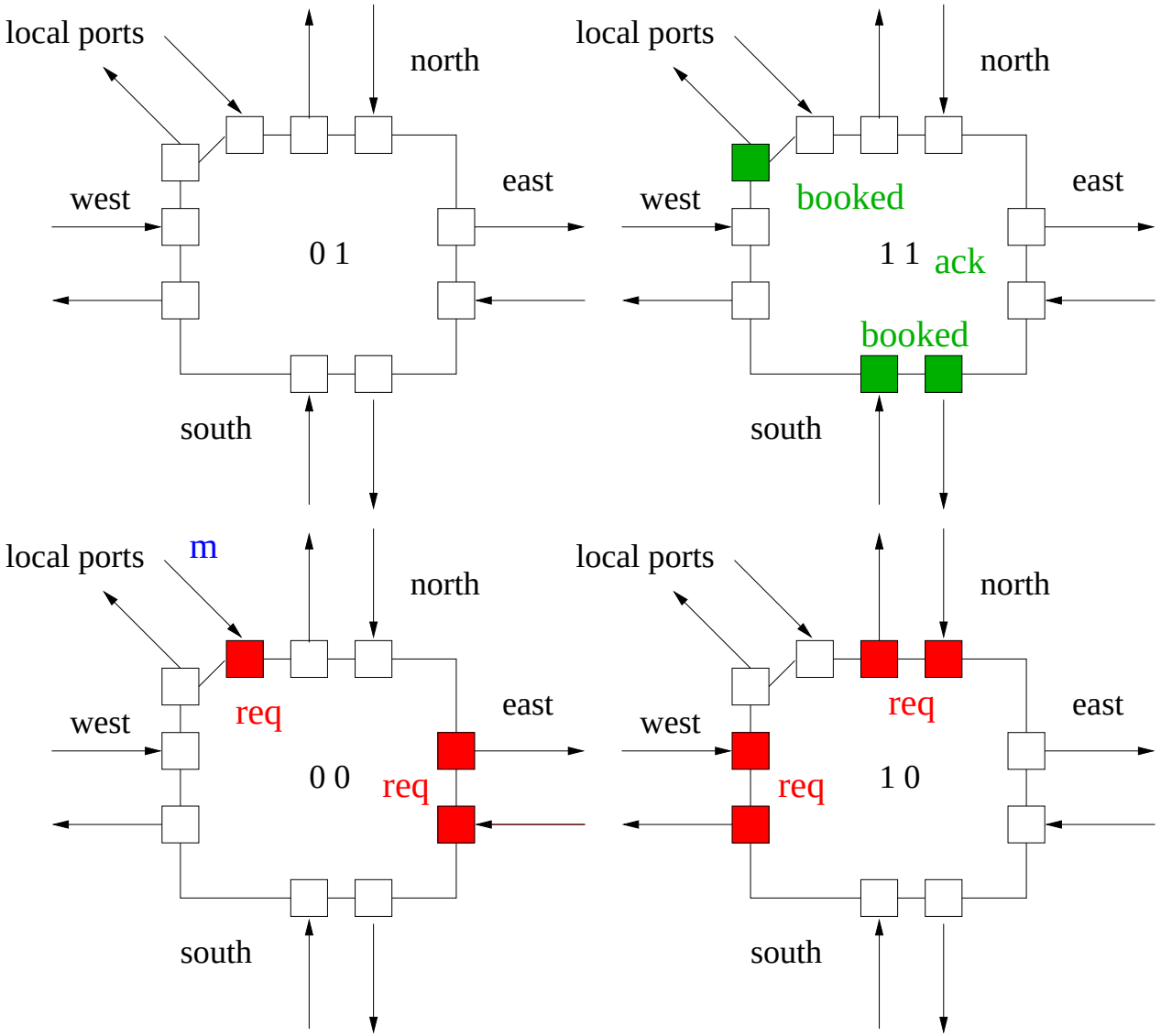
Circuit Switching and XY Routing



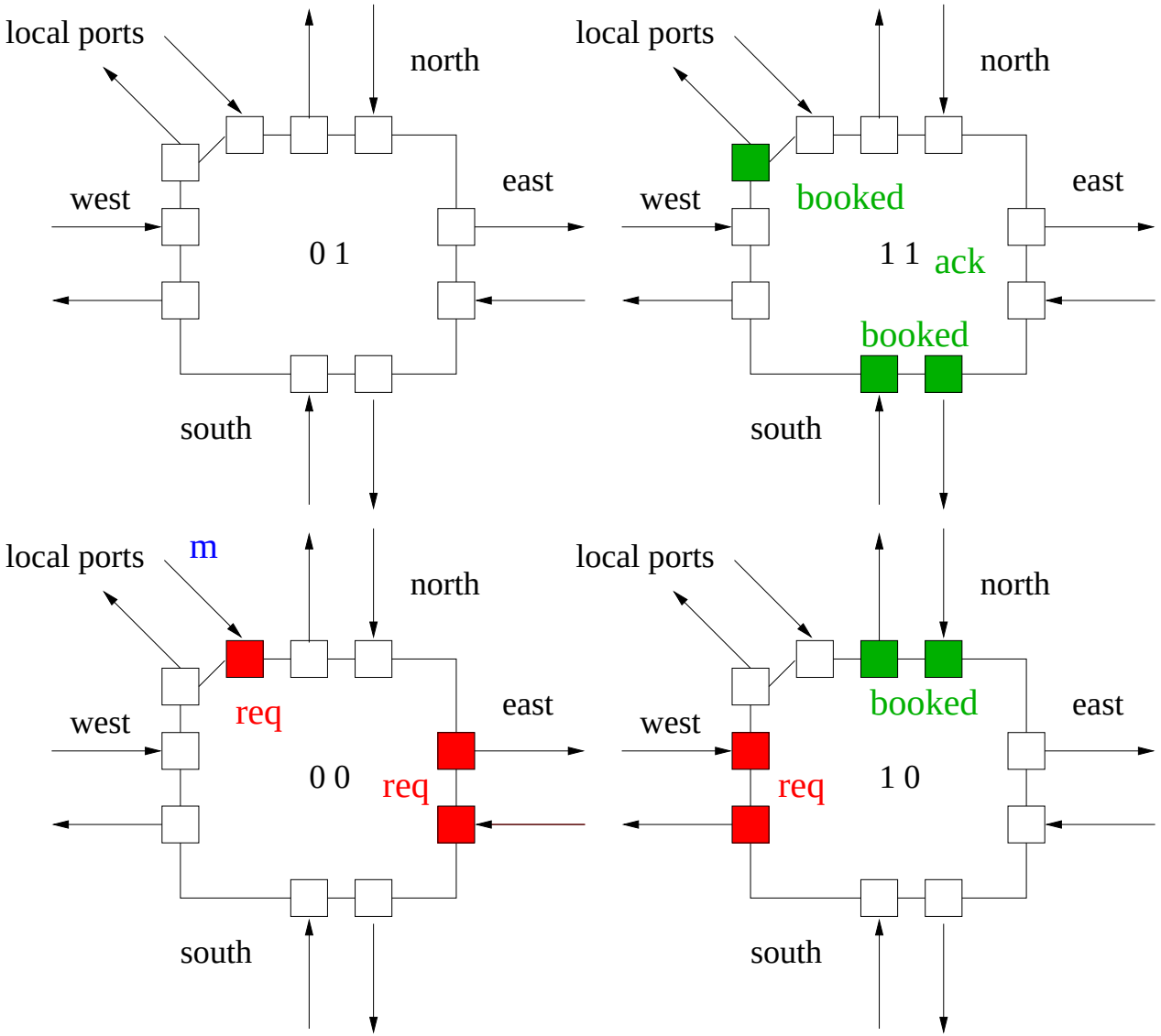
Circuit Switching and XY Routing



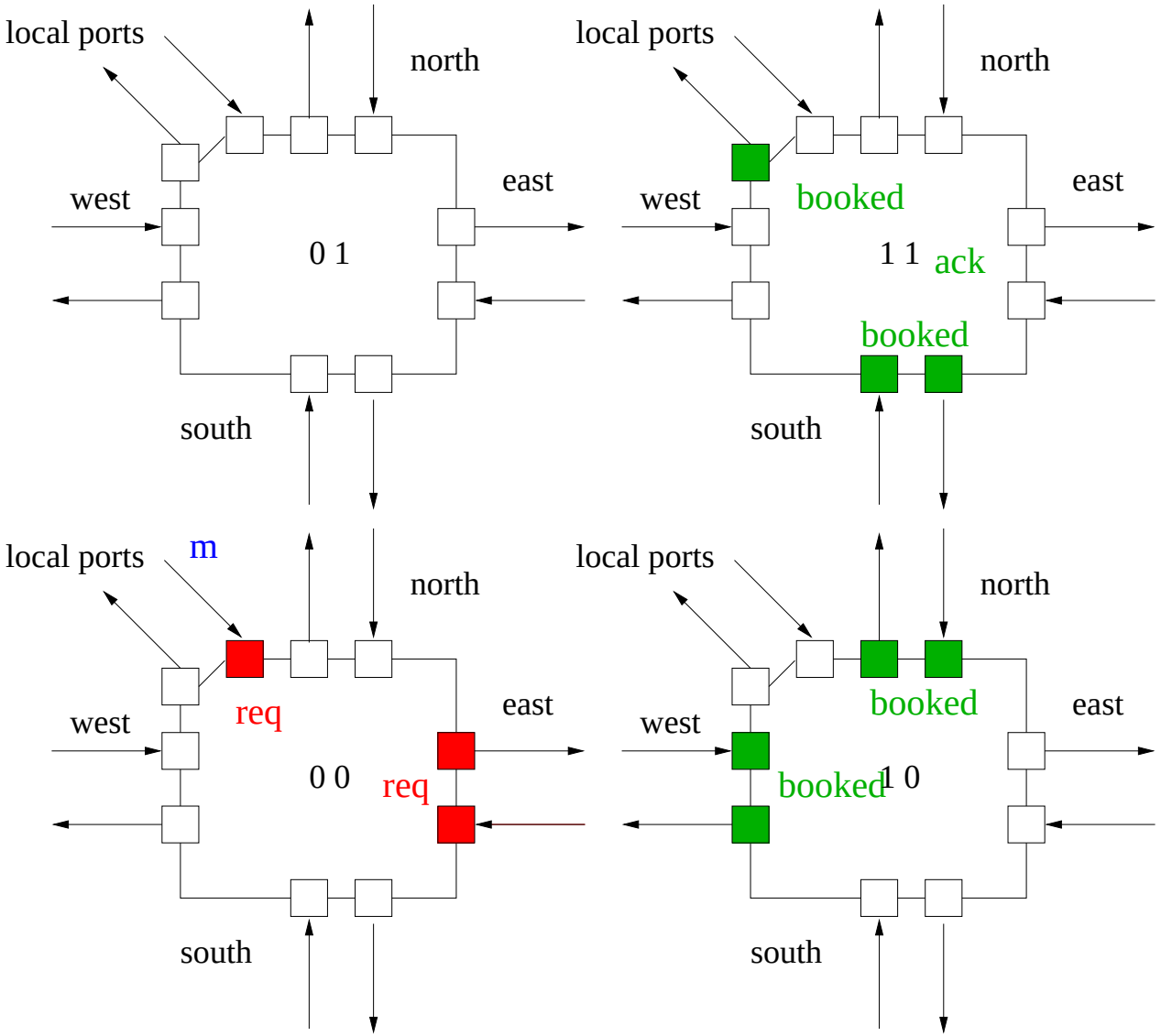
Circuit Switching and XY Routing



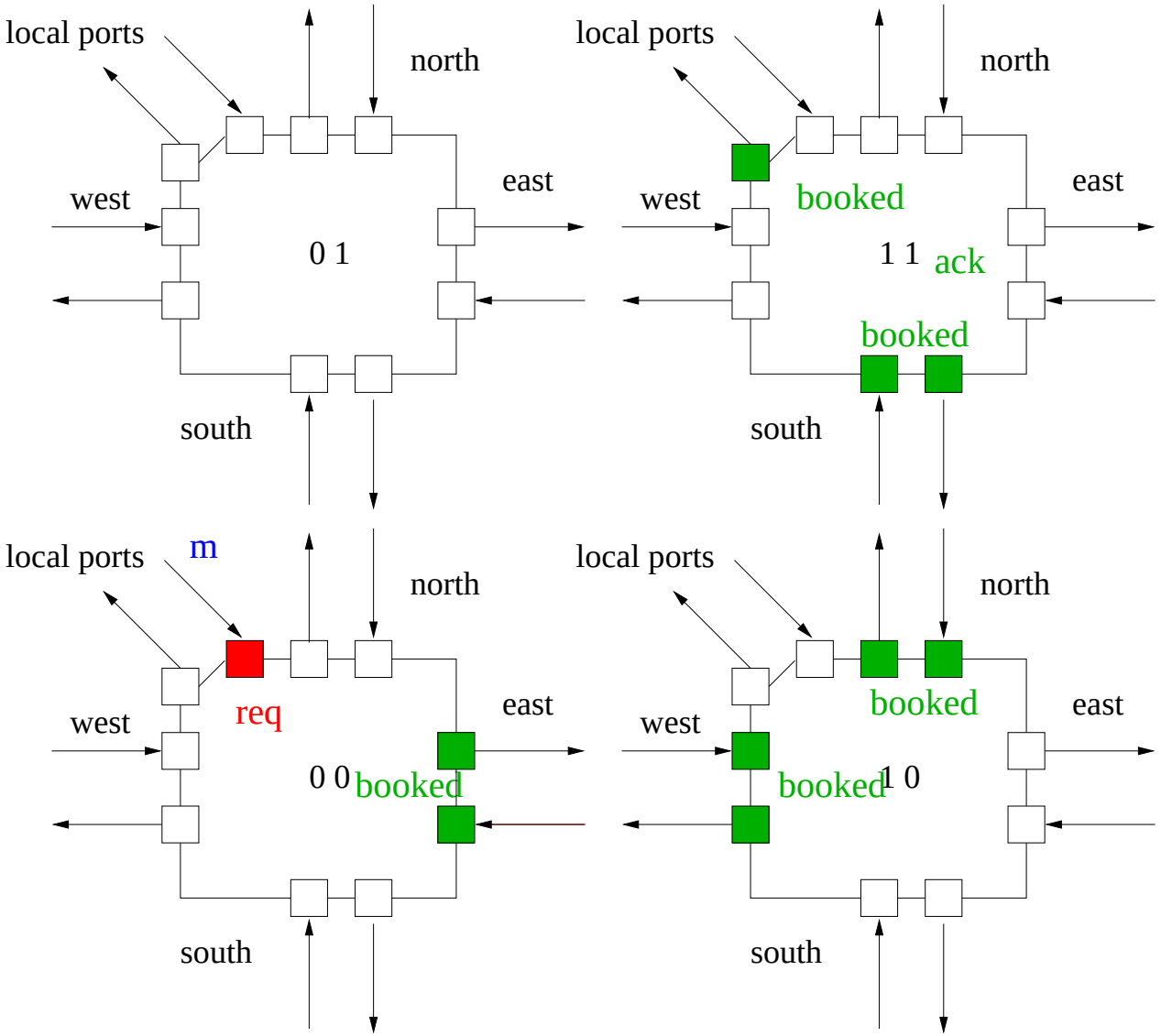
Circuit Switching and XY Routing



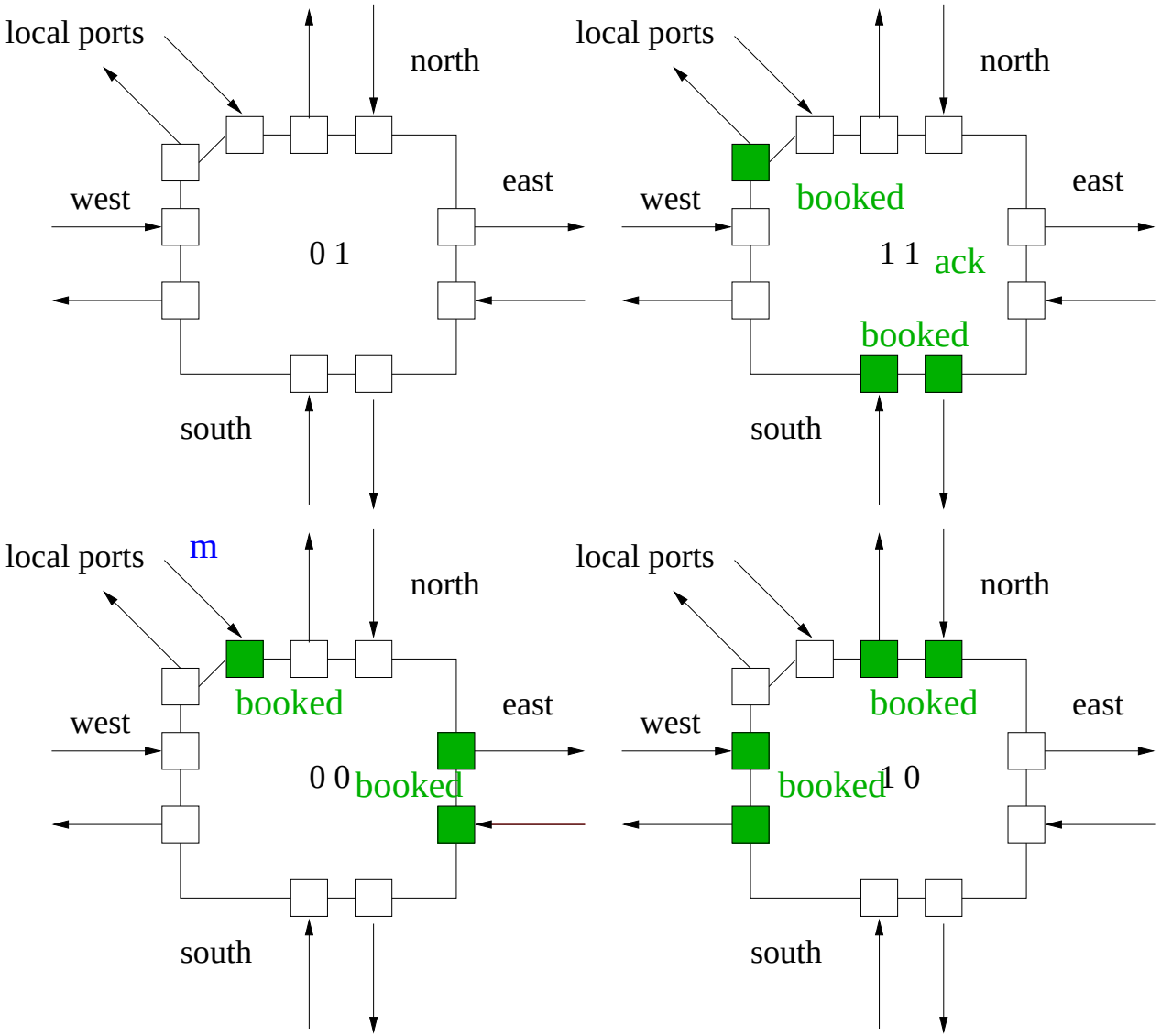
Circuit Switching and XY Routing



Circuit Switching and XY Routing



Circuit Switching and XY Routing



Recursive call

```
(defun genoc_t (m ntkst arr z topo simL)
  (if (zp simL)
    (mv arr m ntkst)
  (mv-let
    (dep del) ;; dep = new value of ntkst
    (depart ntkst m z)
    (let
       ((newntkst (step-ntk dep topo))
       (genoc_t
         del newntkst
         (append
           (list (list 'TIME z
                       (arrive newntkst)))
            arr)
         (1+ z) topo (1- simL)))))))
```