Formal Validation of Deadlock prevention in Networks-on-Chips

F. Verbeek J. Schmaltz

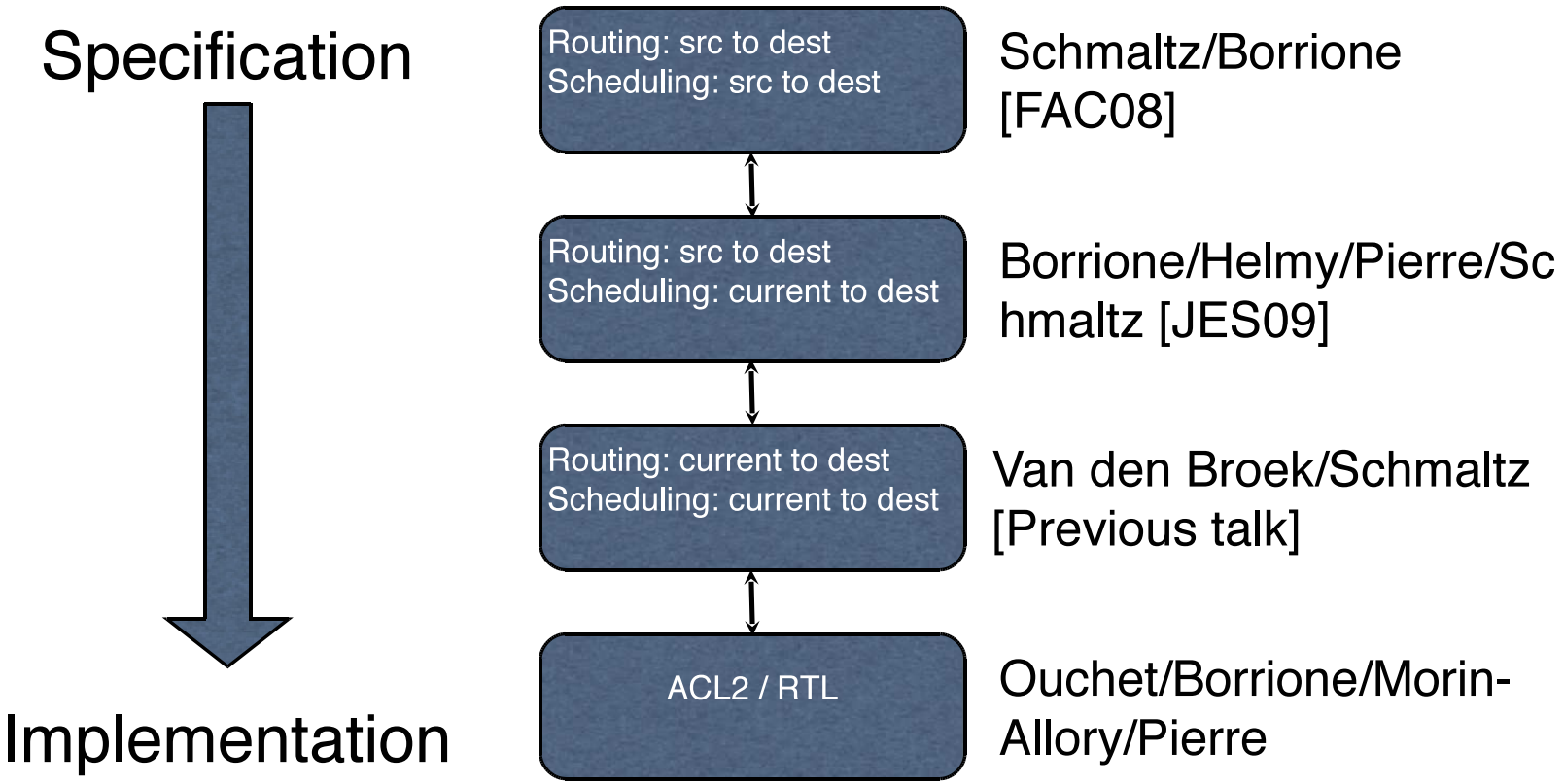
Radboud University, Nijmegen

12-05-2009 / ACL2 Workshop

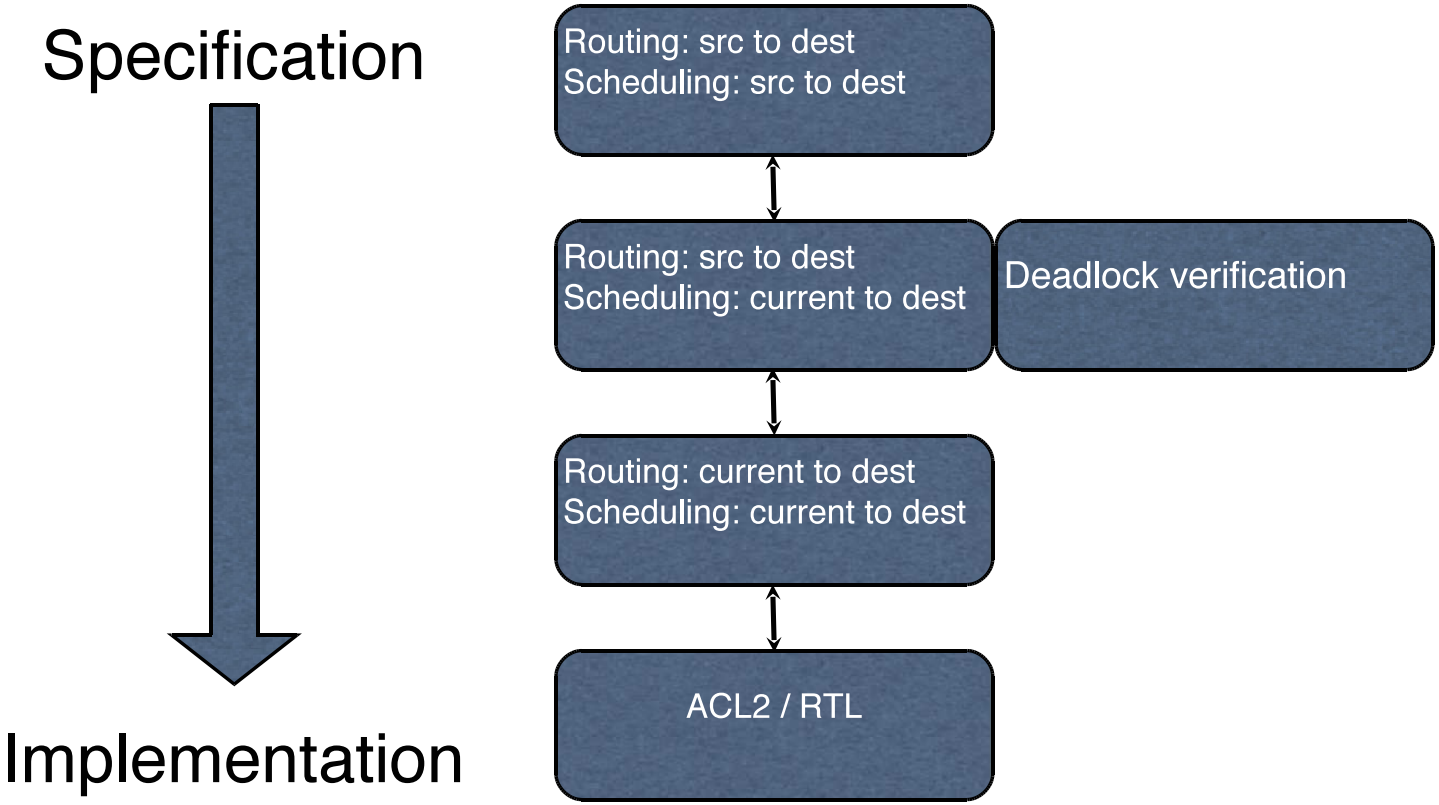
Outline

Radboud University Nijmegen

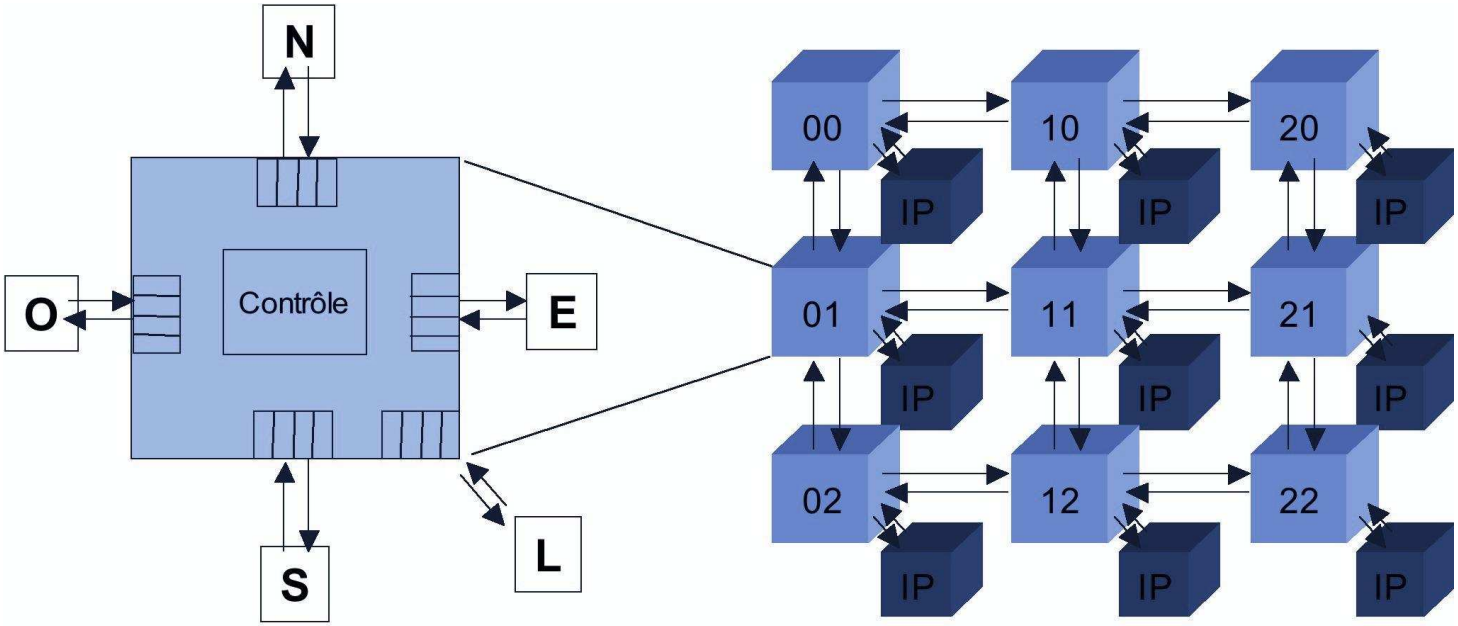
The GeNoC Stack



Contribution



The HERMES Network-on-Chip



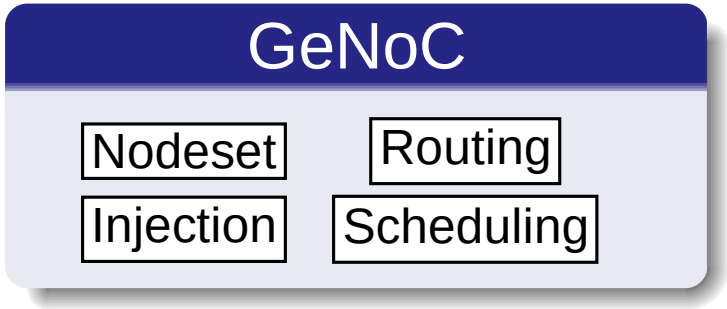
Moreas et al. 2004

Constituents

Nodeset	Routing
Injection	Scheduling

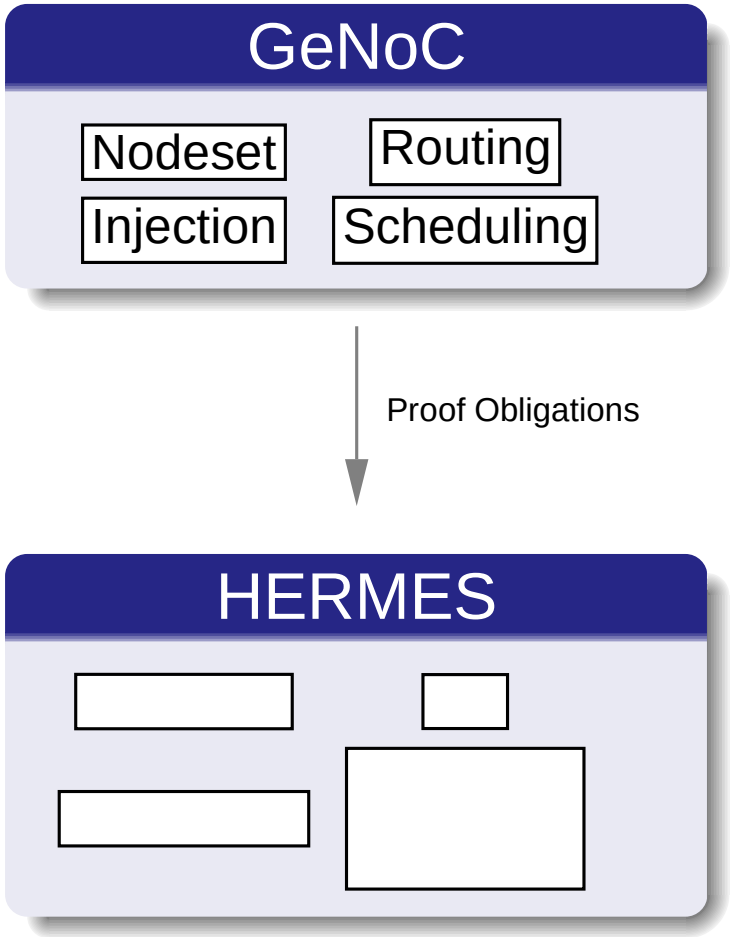
Radboud University Nijmegen

GeNoC for HERMES

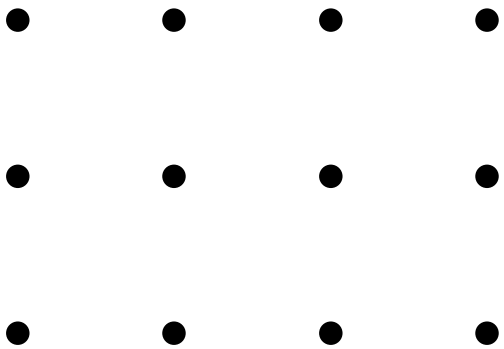


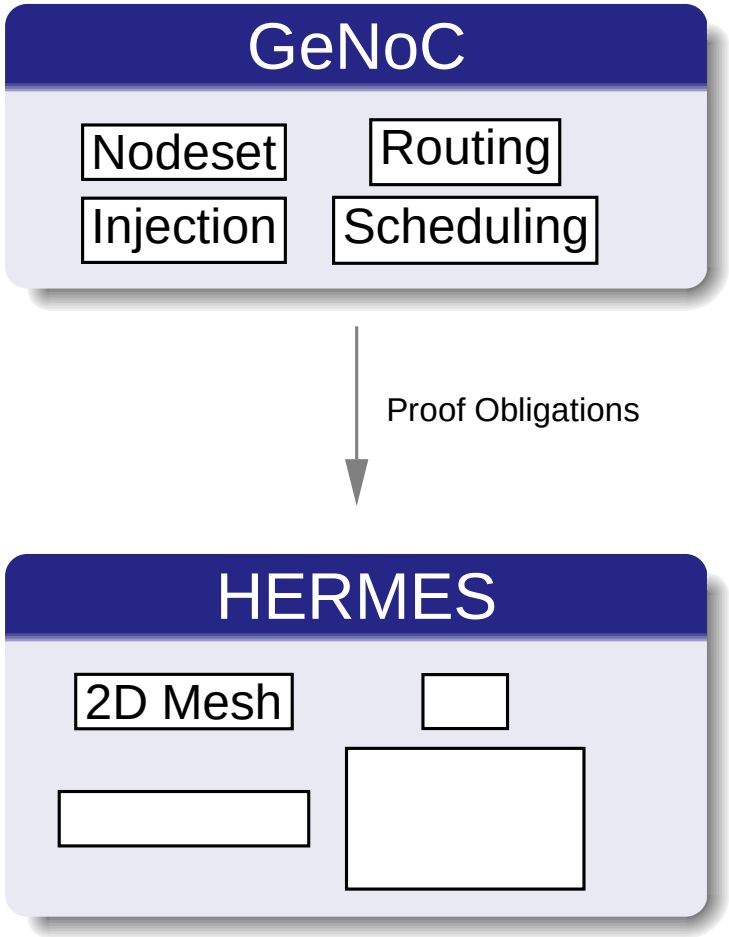
Radboud University Nijmegen

GeNoC for HERMES



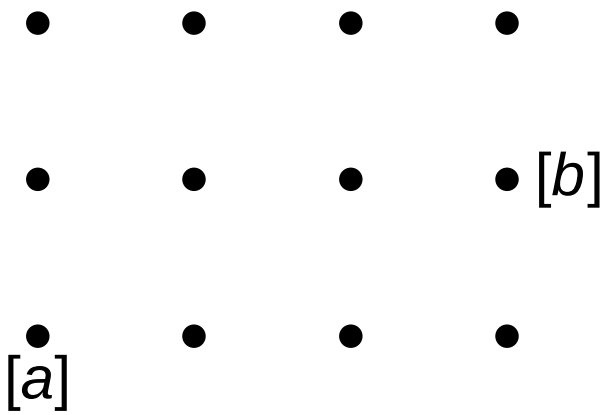
GeNoC for HERMES

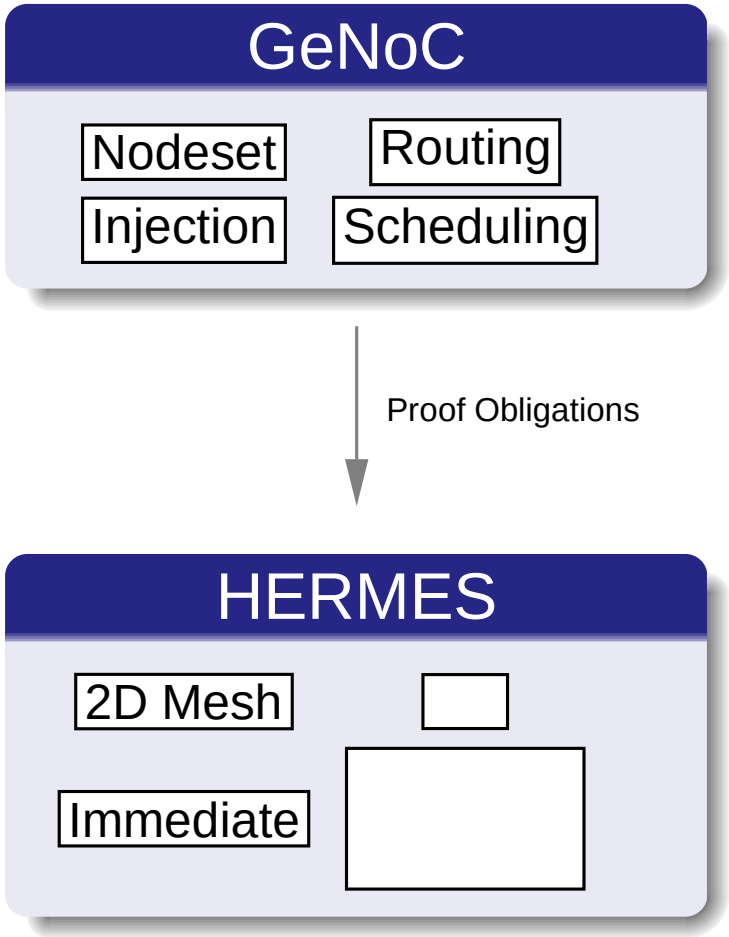




Radboud University Nijmegen

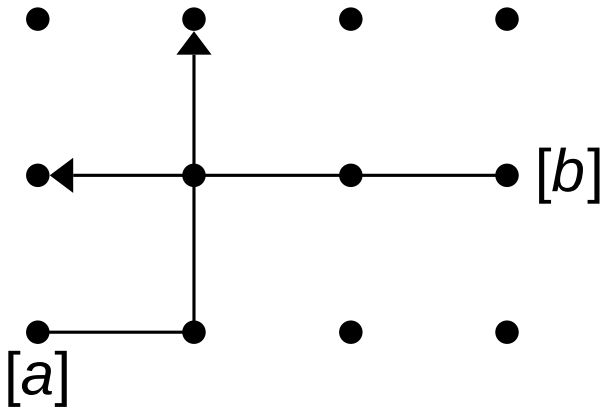
GeNoC for HERMES

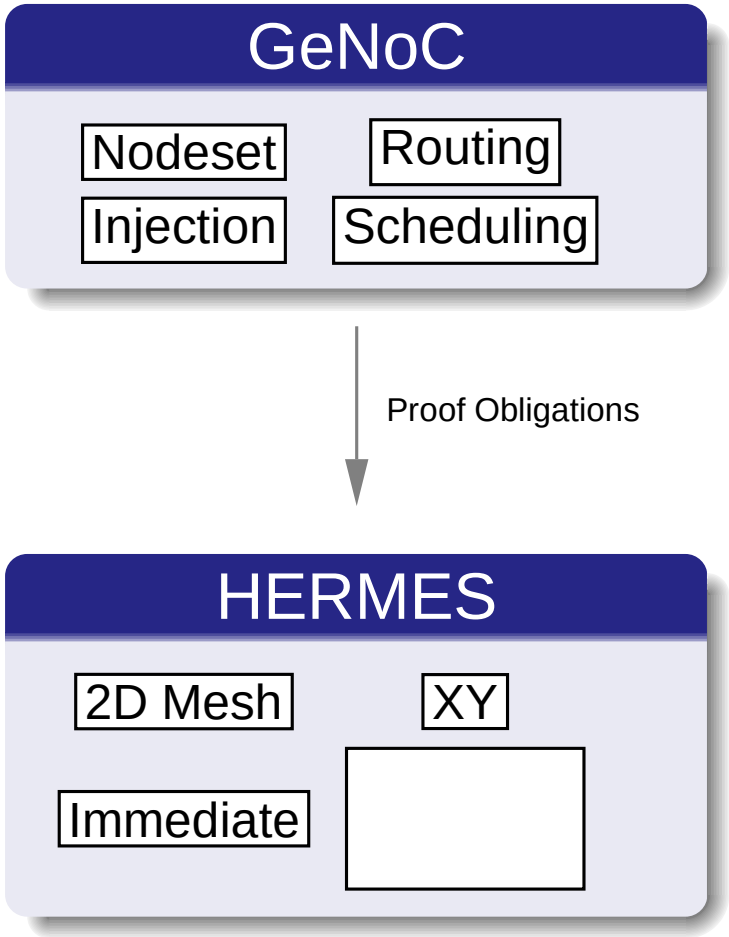




Radboud University Nijmegen

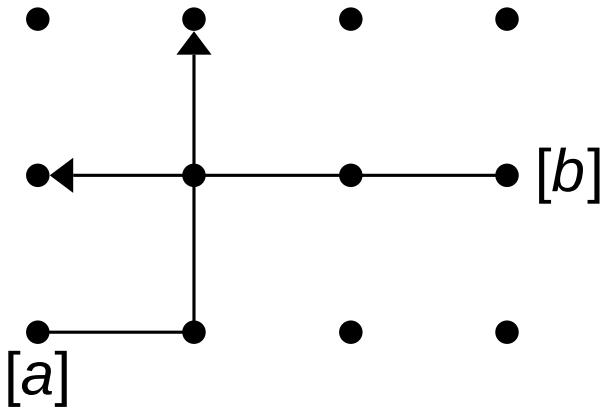
GeNoC for HERMES





Radboud University Nijmegen

GeNoC for HERMES



GeNoC

Nodeset | Routing | Injection | Scheduling

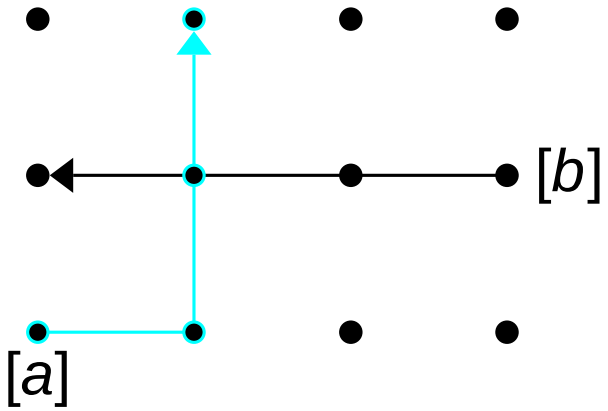
Proof Obligations

HERMES

2D Mesh | XY | Immediate | Circuit Switching

Radboud University Nijmegen

GeNoC for HERMES



GeNoC

Nodeset | Routing | Injection | Scheduling

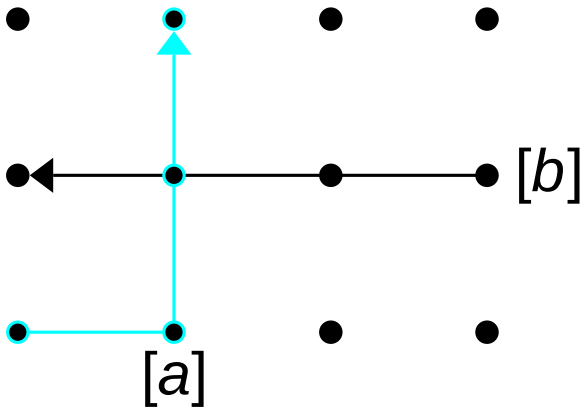
Proof Obligations

HERMES

2D Mesh | XY | Immediate | Circuit Switching

Radboud University Nijmegen

GeNoC for HERMES



GeNoC

Nodeset | Routing

Injection | Scheduling

Proof Obligations

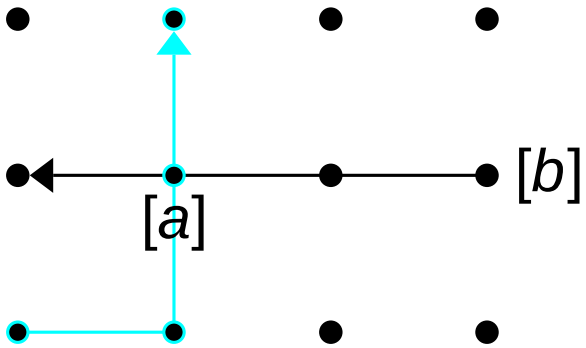
HERMES

2D Mesh | XY

Immediate | Circuit Switching

Radboud University Nijmegen

GeNoC for HERMES



GeNoC

Nodeset | Routing | Injection | Scheduling

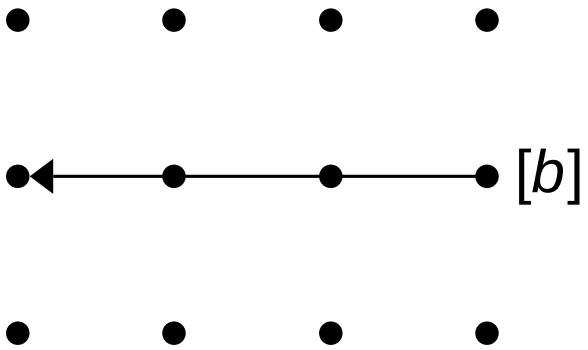
Proof Obligations

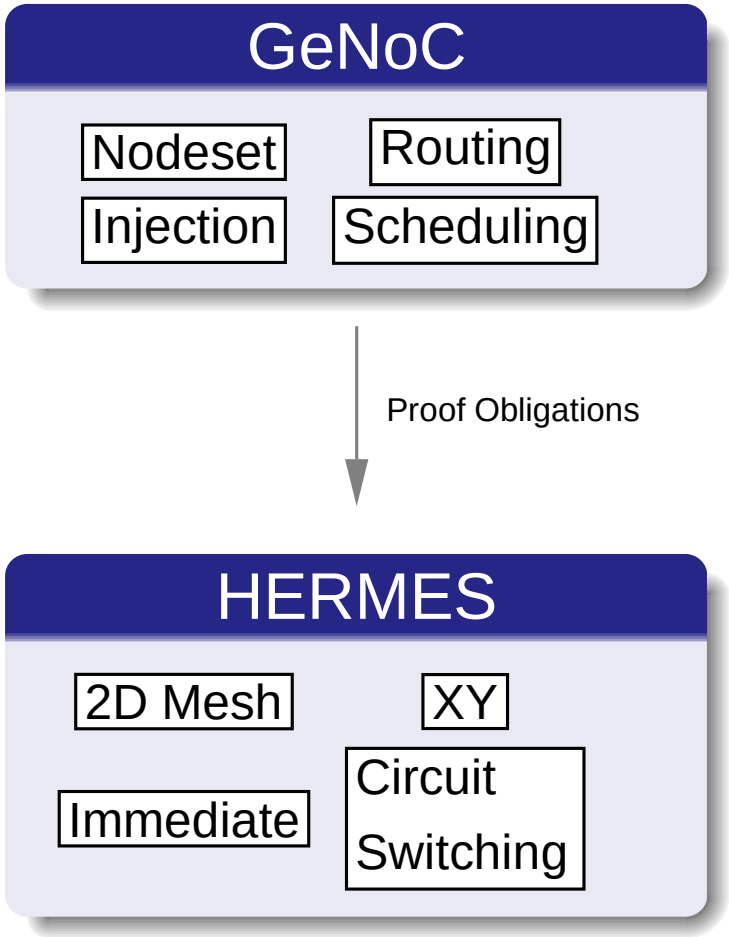
HERMES

2D Mesh | XY | Immediate | Circuit Switching

Radboud University Nijmegen

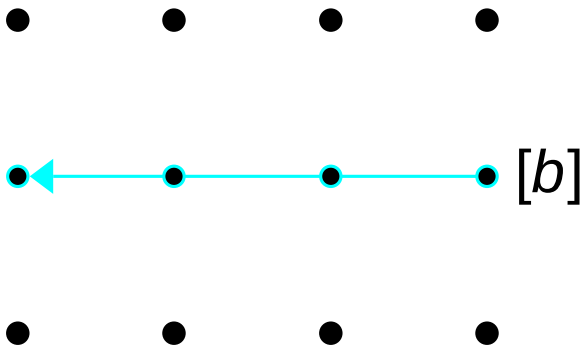
GeNoC for HERMES

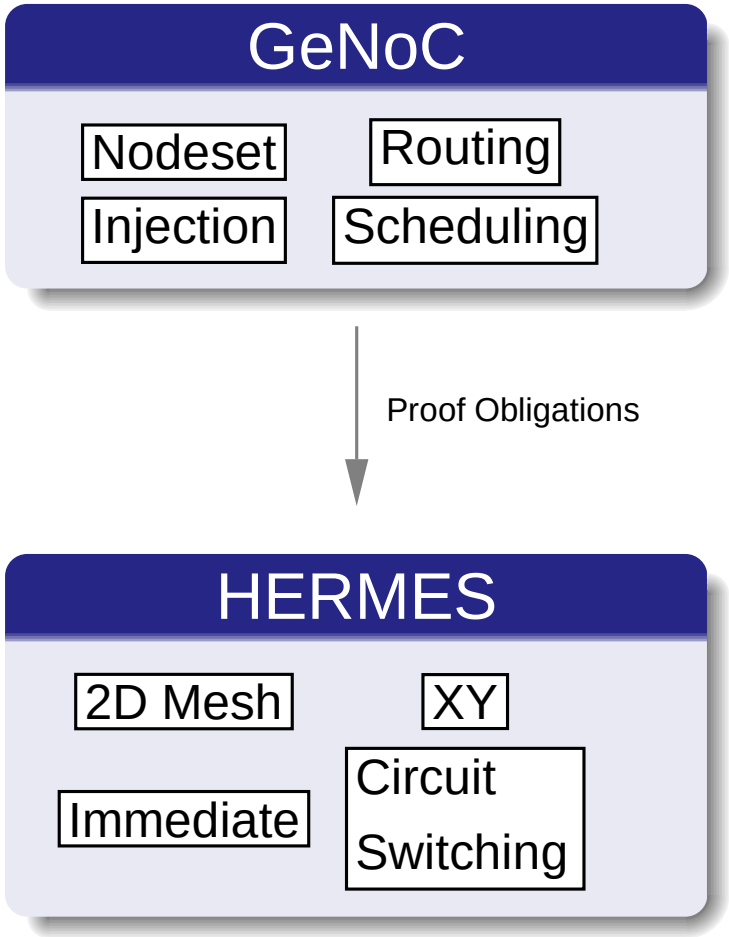




Radboud University Nijmegen

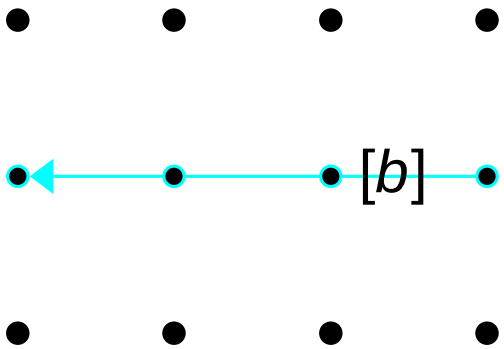
GeNoC for HERMES

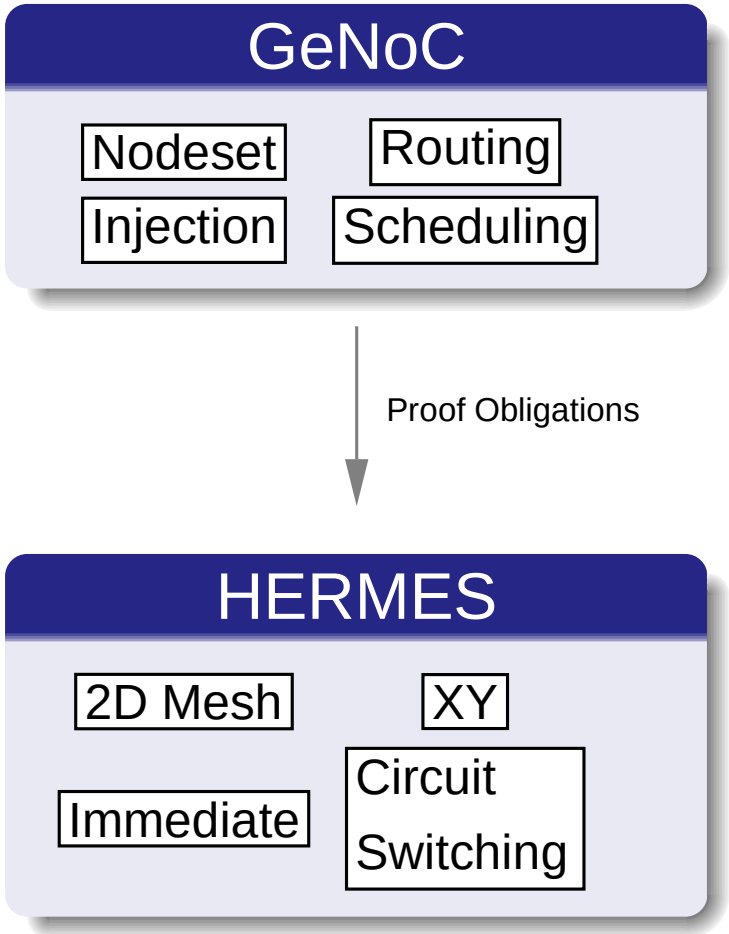




Radboud University Nijmegen

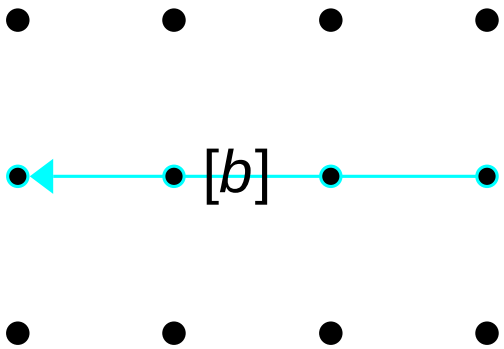
GeNoC for HERMES

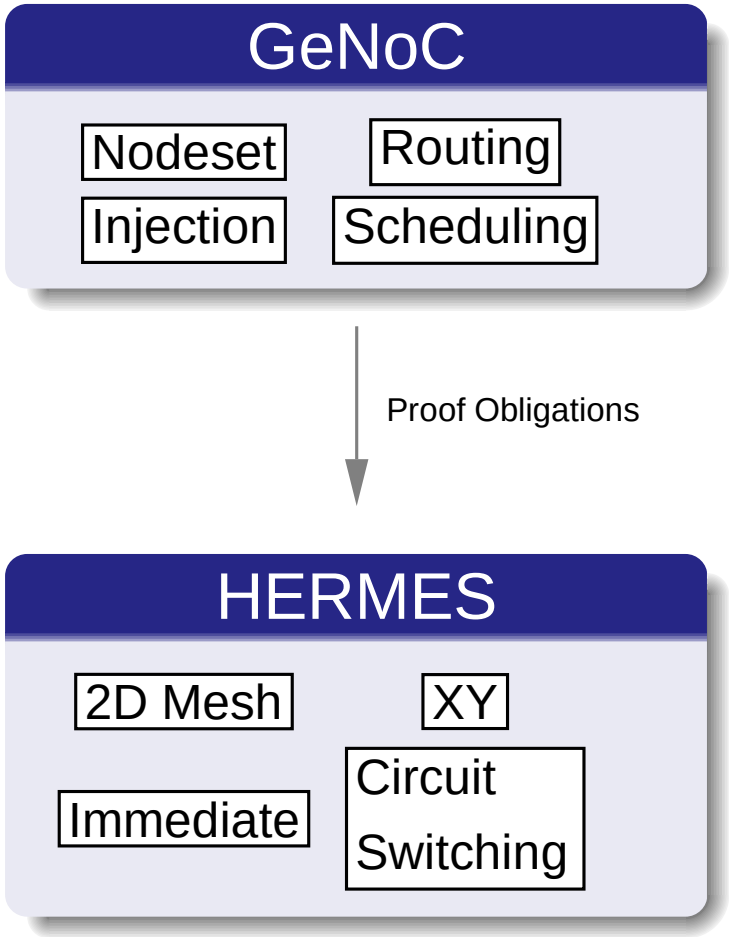




Radboud University Nijmegen

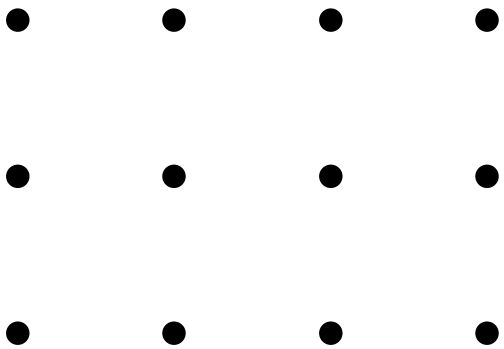
GeNoC for HERMES

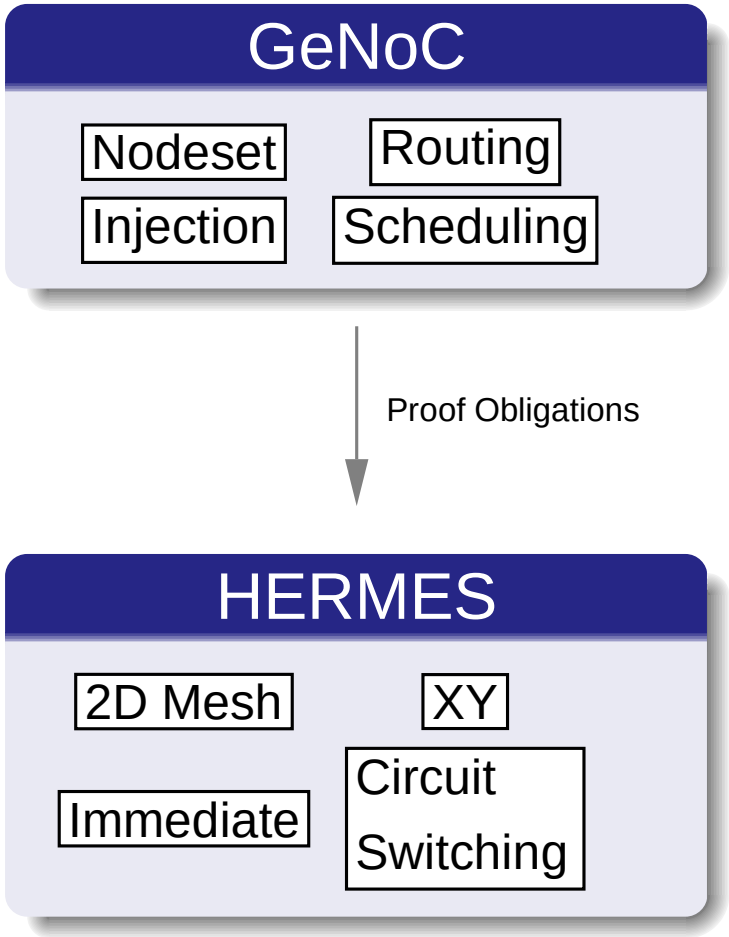




Radboud University Nijmegen

GeNoC for HERMES





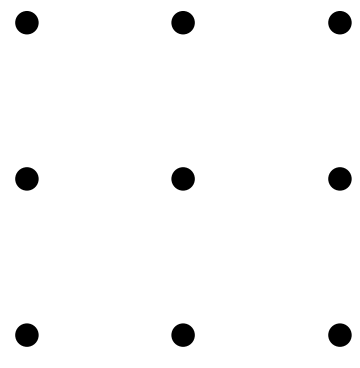
Radboud University Nijmegen

GeNoC

`trlst` denotes the list of travels
`st` denotes the current state of the network
`arr` stores arrived travels

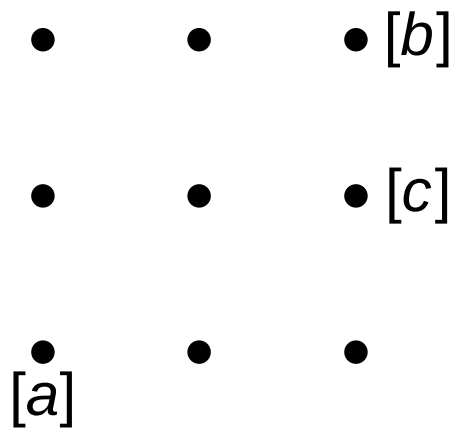
```
(defun GeNoC (trlst st arr)
  (if (endp trlst)
    (list arr nil)
    (mv-let (delayed enroute)
            (injection trlst st)
            (let ((enroute' (routing enroute)))
              (mv-let (st' arr')
                      (scheduling st enroute')
                      (GeNoC (append delayed enroute')
                             st'
                             (append arr' arr)))))))
```

Radboud University Nijmegen



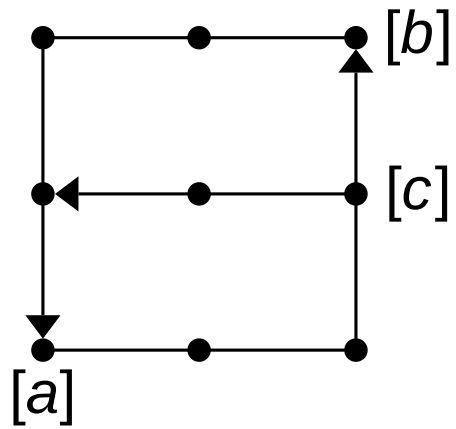
```
trlst = (a, b, c)
arr   = nil
del   =
er    =
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



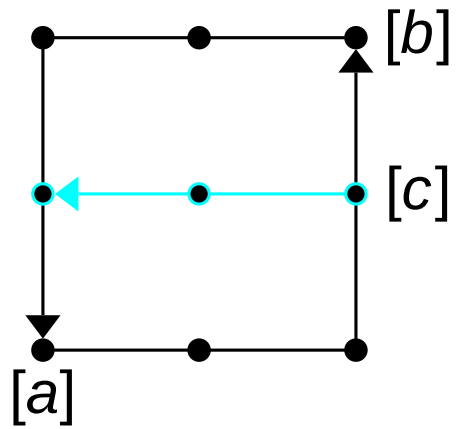

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



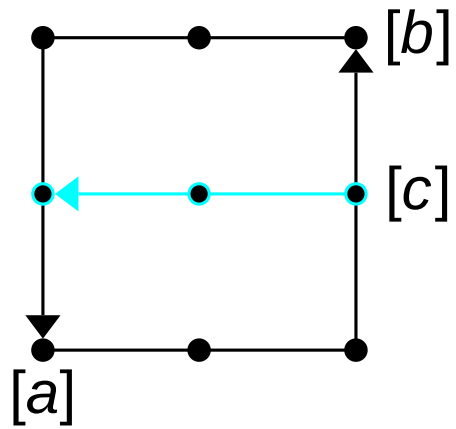

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



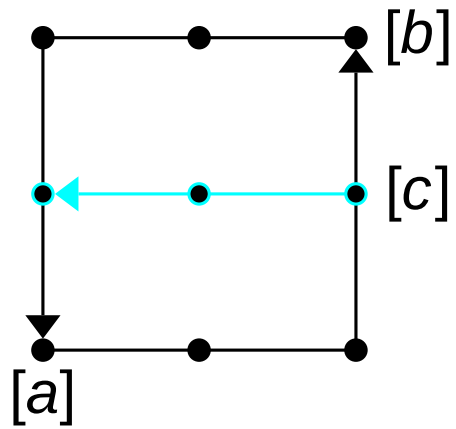

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b, c)
arr'  = nil
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



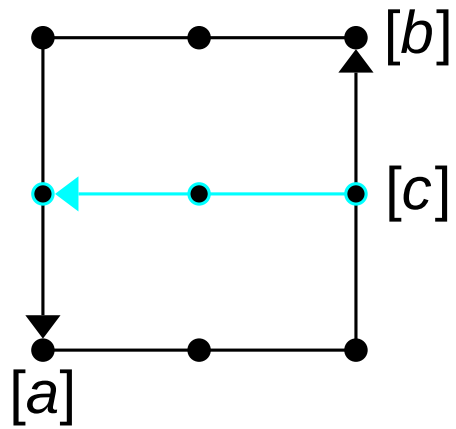

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b, c)
arr'  = nil
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



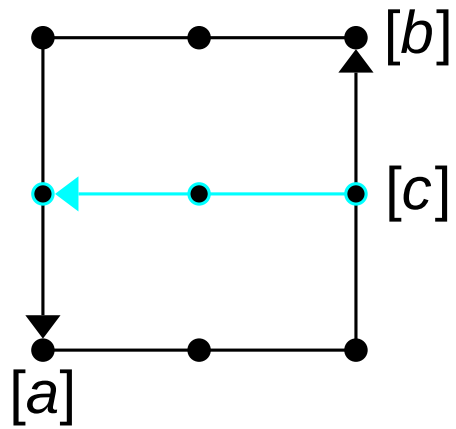

```
trlst = (a, b, c)
arr   = nil
del   =
er    =
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



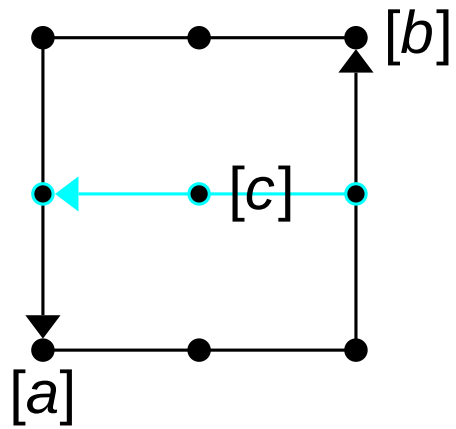

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



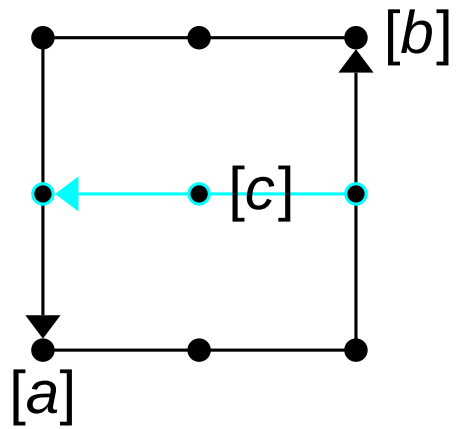

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



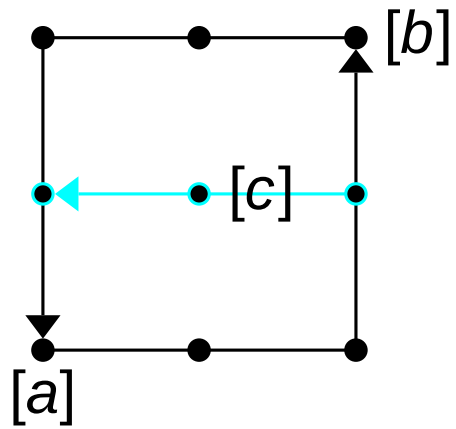

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b, c)
arr'  = nil
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



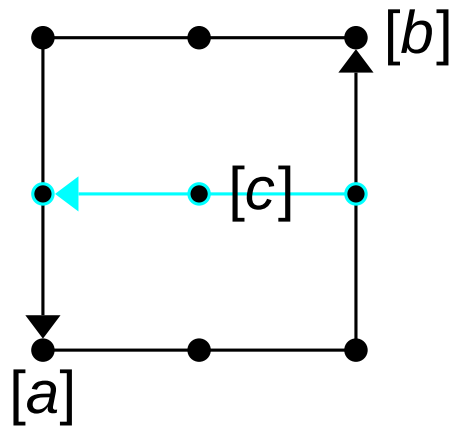

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b, c)
arr'  = nil
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



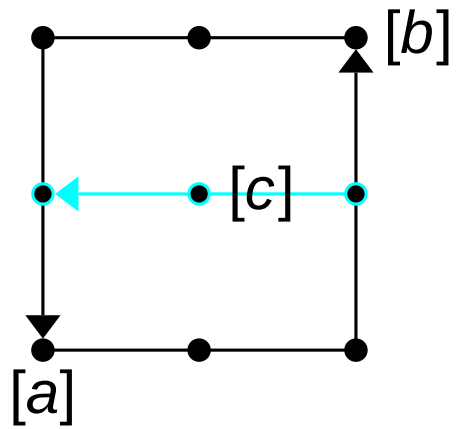

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    =
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



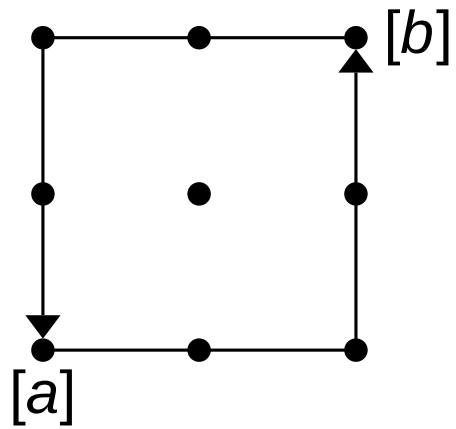

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```



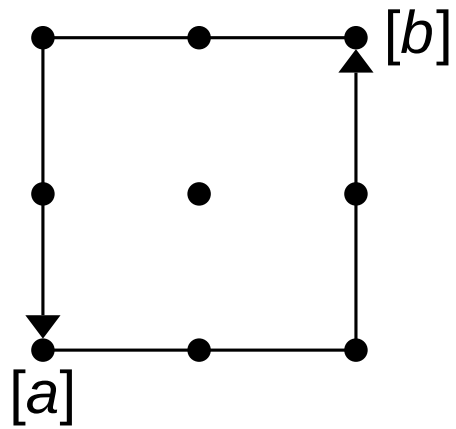

```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   =
arr'  =
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```




```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b)
arr'  = (c)
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```




```
trlst = (a, b, c)
arr   = nil
del   = nil
er    = (a, b, c)
er'   = (a, b)
arr'  = (c)
```

```
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
            (append del er')
            st'
            (append arr' arr)
                        )))))
```

Termination of GeNoC

Current situation:

- Measure is sum of attempts of each travel
- Termination of GeNoC $=$ Attempts have been exhausted

Desired situation:

- Termination of GeNoC $=$ 1.) All travels have arrived
  or
  2.) deadlock

Approach

- A *generic measure* is added as parameter to GeNoC:
  1. Generic function `deadlock-statep`
  2. Generic function `measurep`
- New proof obligation:

```
(defthm measure-decreases
  (implies (and (measurep (trlst st meas))
                (not (deadlock-statep trlst st)))
           (O< (acl2-count (mv-nth 2
                                   (scheduling trlst st)))
               (acl2-count meas))))
```

Measure for Circuit Switching

- Measure:

```
(defun ct-measurep (trlst st meas)
  (equal meas
         (sum-of-lst (route-lengths trlst))))
```

- Deadlock-state:

```
(defun ct-deadlock-statep (trlst st)
  (if (endp trlst)
    t
    (and (not (free-nodes (route (car trlst)) st))
         (ct-deadlock-statep (cdr trlst) st))))
```

Radboud University Nijmegen

```
(if (endp trlst)
  (list arr trlst)
  (mv-let (del er)
          (injection trlst st)
          (let ((er' (routing er)))
            (if (deadlock-statep er' st)
                (list arr trlst)
                (mv-let (st' arr' meas')
                        (scheduling st er')
                        (GeNoC
                             (append del er')
                             st' meas'
                             (append arr' arr)
                                     ))))))
```

Generic proof structure

To prove that a NoC is deadlockfree:

1. Define set of conditions P, such that:

$$P \Longrightarrow \texttt{(not (deadlock-statep))}$$

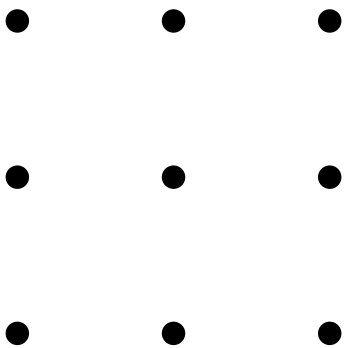
2. Prove that P is inductive for GeNoC

Instantiation

Network:

Nodeset:	$x \times y$ 2D Mesh, b buffers per node
Injection:	Immediate
Routing:	XY Routing
Scheduling:	Circuit Switching with bookings

Deadlock prevention



$b = 3$

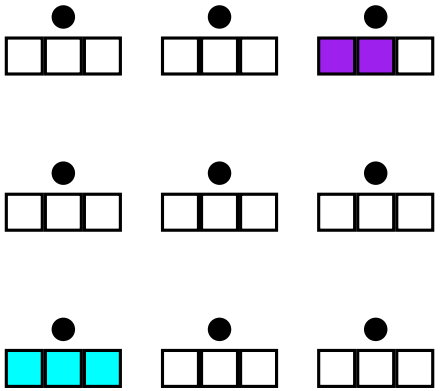
=

Theorem:

Let each node have b buffers.

$$(\texttt{len trlst}) < 2b \implies \text{no deadlock}$$

Deadlock prevention



$b = 3$

$(\texttt{len trlst}) = 5$

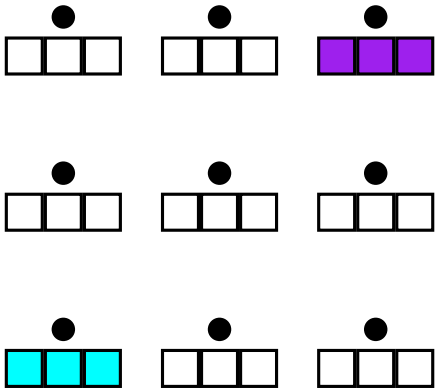
Theorem:

Let each node have b buffers.

$$(\texttt{len trlst}) < 2b \implies \text{no deadlock}$$

Radboud University Nijmegen

Deadlock prevention



$b = 3$

$(\texttt{len trlst}) = 6$

Theorem:

Let each node have b buffers.

$$(\texttt{len trlst}) < 2b \implies \text{no deadlock}$$

Radboud University Nijmegen

Proof structure

Generic:

1. Define set of conditions P, such that:

$$P \implies \texttt{(not (deadlock-statep))}$$

2. Prove that P is inductive for GeNoC

Theorem:

$$(\texttt{len trlst}) < 2b \implies \text{no deadlock}$$

Radboud University Nijmegen

Proof structure

Instantiated:

1. Prove:

$$(\texttt{len trlst}) < 2b \implies$$
at least one travel can advance

2. Prove:

$$(\texttt{len trlst}) < 2b \implies$$
$$(\texttt{len}\,(\texttt{scheduling trlst st}).\texttt{trlst})) < 2b$$

Theorem:

$$(\texttt{len trlst}) < 2b \implies \text{no deadlock}$$

Radboud University Nijmegen

Theorem

$(\texttt{len}\ \texttt{trlst}) < 2b \implies$ *at least one travel can advance*

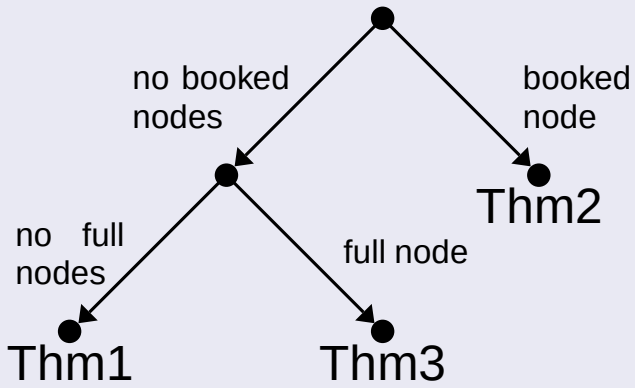
Proof

- A travel can advance if
  1. all nodes of the route are booked, or
  2. all nodes of the route can be booked.
- A node can be booked if
  1. it isn't booked, and
  2. it isn't full.

Theorem

$(\texttt{len}\ \texttt{trlst}) < 2b \implies$ *at least one travel can advance*

Proof



Thm3

No booked nodes and a full node implies any travel currently in the full node can advance.

Proof

Let n be the full node and v be a travel in n:

Thm3.1 The bound on the number of travels implies that for all $n' \neq n$, n' is not full.

Thm3.2 No cycles in routing implies $n \notin \texttt{route}(v)$

Conclusions

- Proven deadlock-prevention theorem by bounding the number of messages.
- Proof consists of 3157 lines: 39 functions and 321 theorems.
- Theorem is proven for any routing, any topology, any nodeset (e.g. ports), and for both circuit and packet switching.
- Defined proof pattern for deadlock-related proofs.

Future work

- Define better bound/set of restrictions.
- Define a more accurate measure than route lengths.
- Prove theorem for wormhole switching.
- Varying injection methods.

Questions?