Formal Validation and Verification of Networks-on-Chips

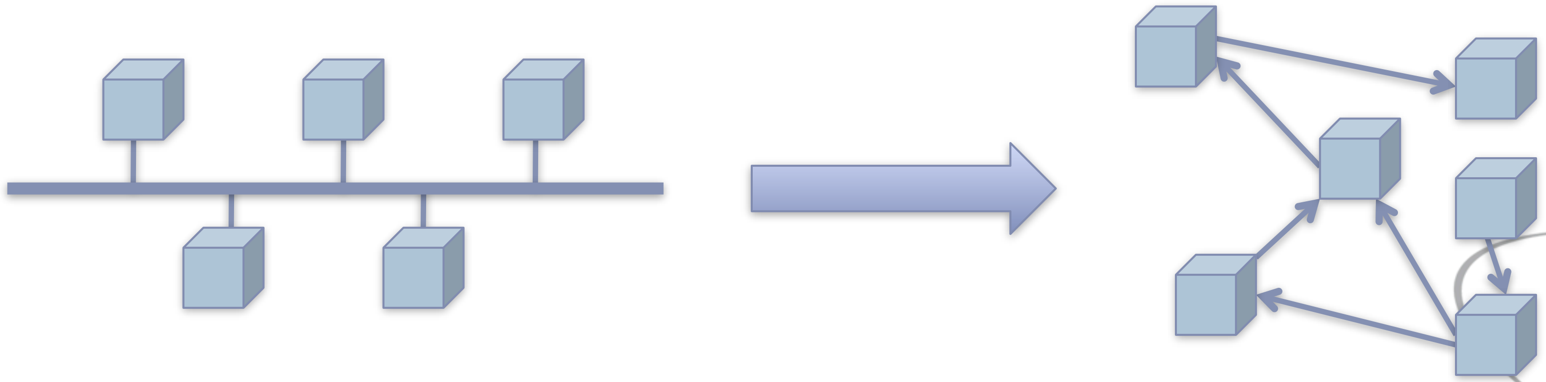
Status and Perspective

Julien Schmaltz
Freek Verbeek
Tom van den Broek

OpenUniversiteitNederland
Radboud University Nijmegen
www.ou.nl

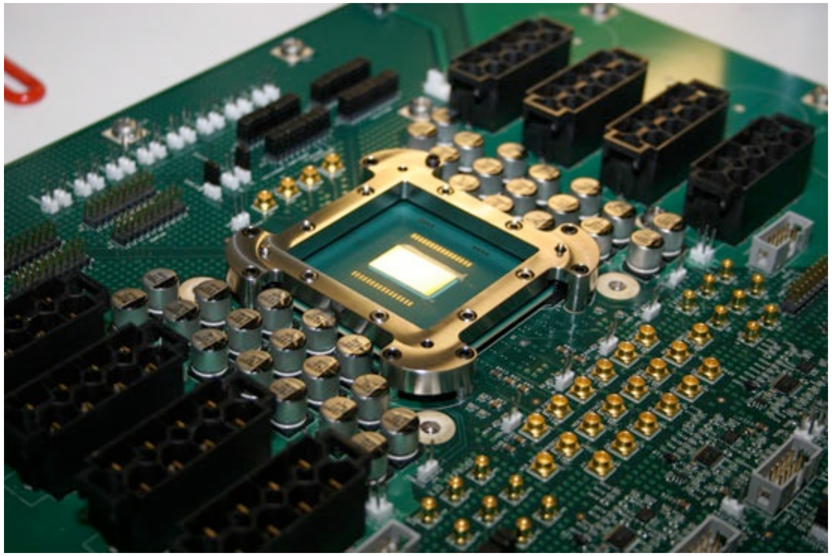
Networks-on-Chips

- MPSoCs - Multi-Processors Systems-on-Chips
- Trend: Networks replace buses



MultiProcessor SoCs

- Intel's 80-core Research Chip
- Teraflops, 62 Watts
- 100 millions transistors, 275 mm2
- 25% node area for router



- ASCI Red Supercomputer
- Teraflops (Dec. 1996)
- 10, 000 Pentium Pro
- 104 cabinets, 230 m2

Motivation

- Networked based SoCs
- Communication infrastructure crucial to system performance and correctness
- NoCs run under constrained environment
  - limited heat budget
  - must work perfectly (e.g. no loss)
- Network architectures key to supercomputing
- NoCs architectures key to on-chip supercomputing ?

Global Objective

- Verified complex on-chip networks
- General methodology to support the design of correct complex on-chip network architectures

In this talk

- Description of our target methodology
- Recent results towards this target
- Next steps towards our goal

Part I

Our target

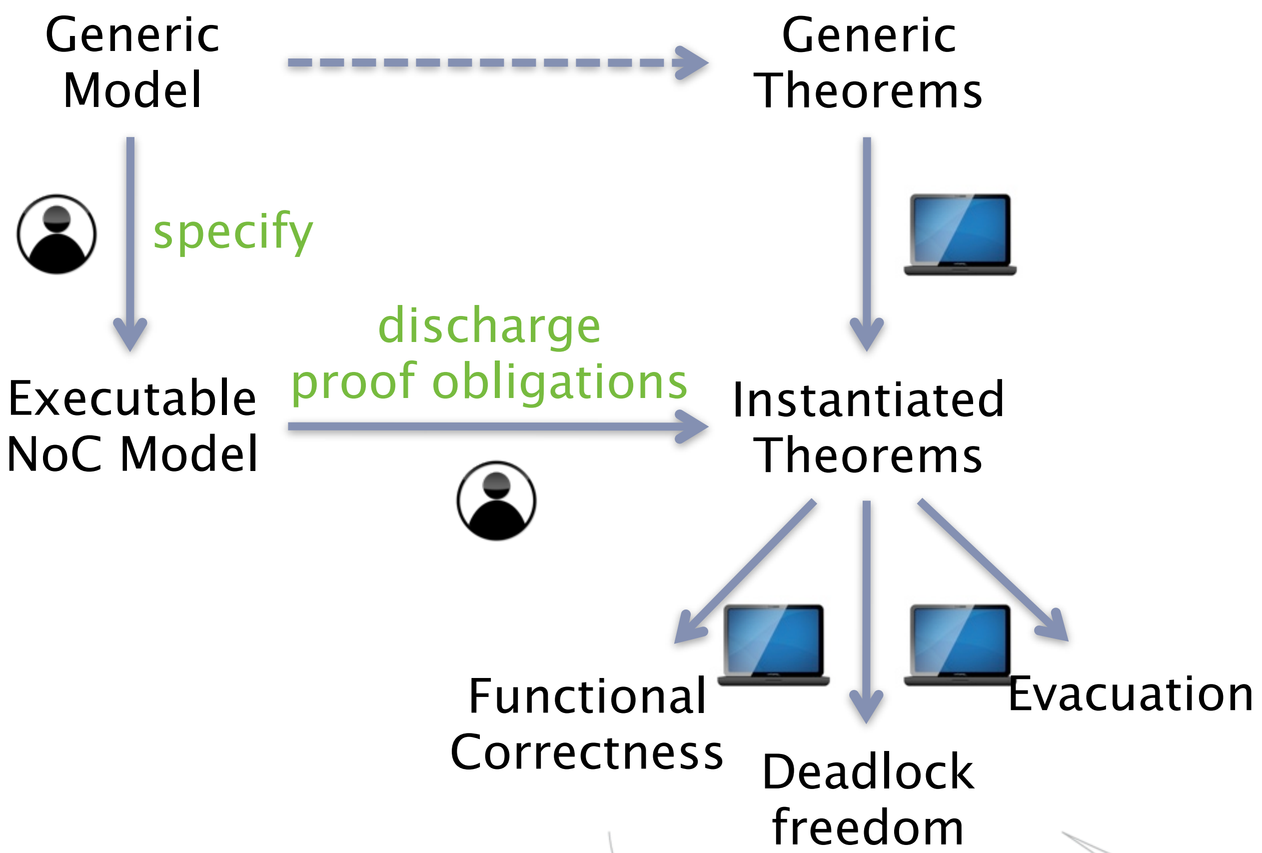
Application domain

- Focus on the communication infrastructure / architecture
- Highly parametric analysis
  - size of the network
  - size of messages
  - topology
  - routing algorithm
  - switching policy
  - injection method
  - ...
- Prove global properties of networks
  - no message loss
  - no deadlock/livelock
  - evacuation
  - performance
  - ...

Method elements

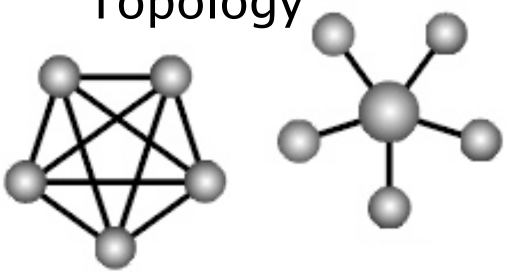
- Generic or meta-model
  - constituents
  - architectures
  - proof obligations and theorems
- Temporal abstractions
  - define maximum travel distance per time unit

The GeNoC approach

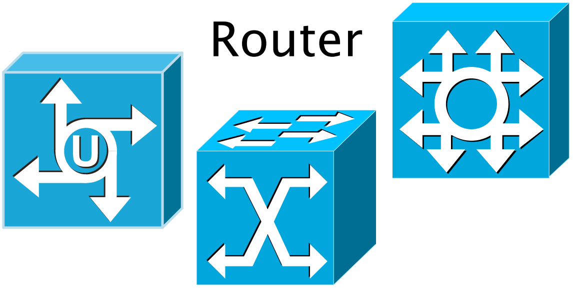


The Generic Model: Constituents

Topology



Router



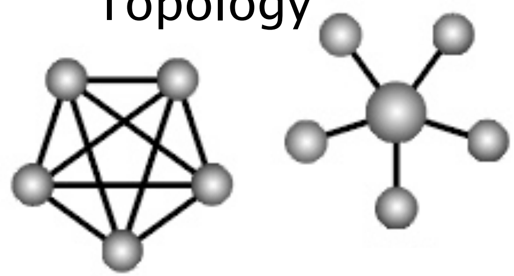
Scheduler

Injection

The Generic Model: Proof obligations (or constraints)

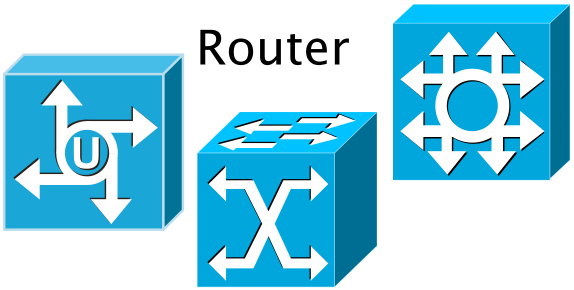
Local constraints sufficient to prove global generic theorems.

Topology



"Sinks have no outgoing edges"

Router



"R:PxP→P"

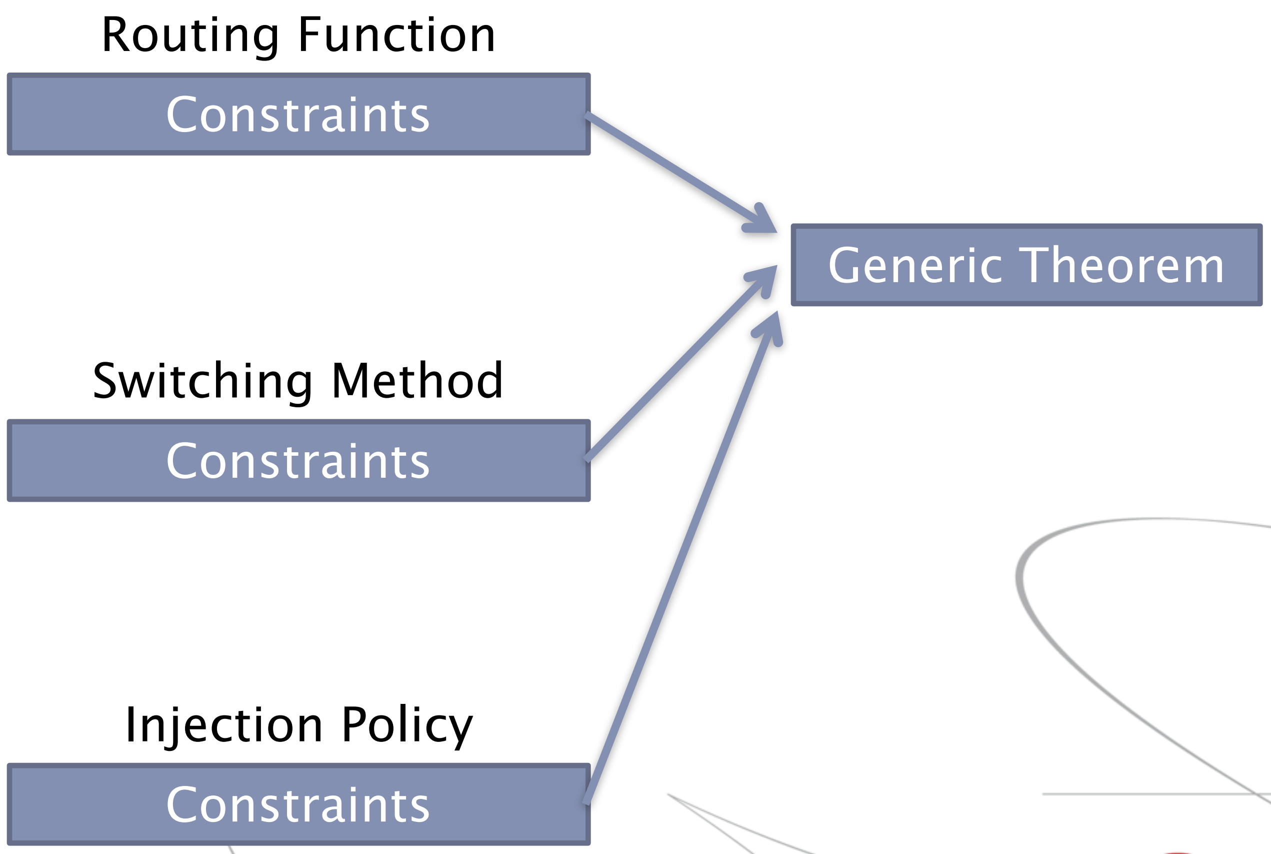
Scheduler

"A message moves, unless it is stuck"

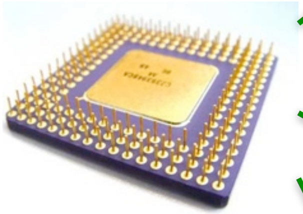
"All messages are injected at time slot 0"

Injection

The Generic Model: Generic theorems



The Generic Model: Generic theorems



Routing Function

Constraints

Instantiated Constraints

Switching Method

Constraints

Instantiated Constraints

Injection Policy

Constraints

Instantiated Constraints

Generic Theorem

Instantiated Theorem

The GeNoC Model: Architecture template

Let σ be a configuration containing a state and messages
Let M be a set of messages to be sent over the NoC

GeNoC (σ) =

σ iff σ.M = ∅ // empty list of messages

σ iff deadlocked(Routing(Injection(σ)))

GeNoC(***take-a-step***(σ))

The Deadlock and Evacuation Theorems - DATE'10

- Deadlock Theorem
  - Routing function R is deadlock-free if and only if there is no cycle in its port dependency graph
- Evacuation Theorem
  - All messages eventually leave the network if and only if function GeNoC terminates
- New constraints
  - Ports dependency graph must be consistent with the routing function (topology as well)
  - Scheduling policy must decrease the termination measure if no deadlock
  - Injection methods injects all messages at time 0
- Application
  - Arbitrary large 2D-mesh with XY routing

The Refinement Theorem - FMCAD'09

- Two architectures and a mapping between them
  - Source routing (route encoded in message)
  - Distributed routing (route computed step-by-step)
  - Function *transform* remove encoded route from messages

- The Refinement Theorem
  - For all states s and message lists m, we have
  - transform(GeNoC_S(s,m)) = GeNoC_D(s,m)

- New constraints
  - Encoded route matches step-by-step computation

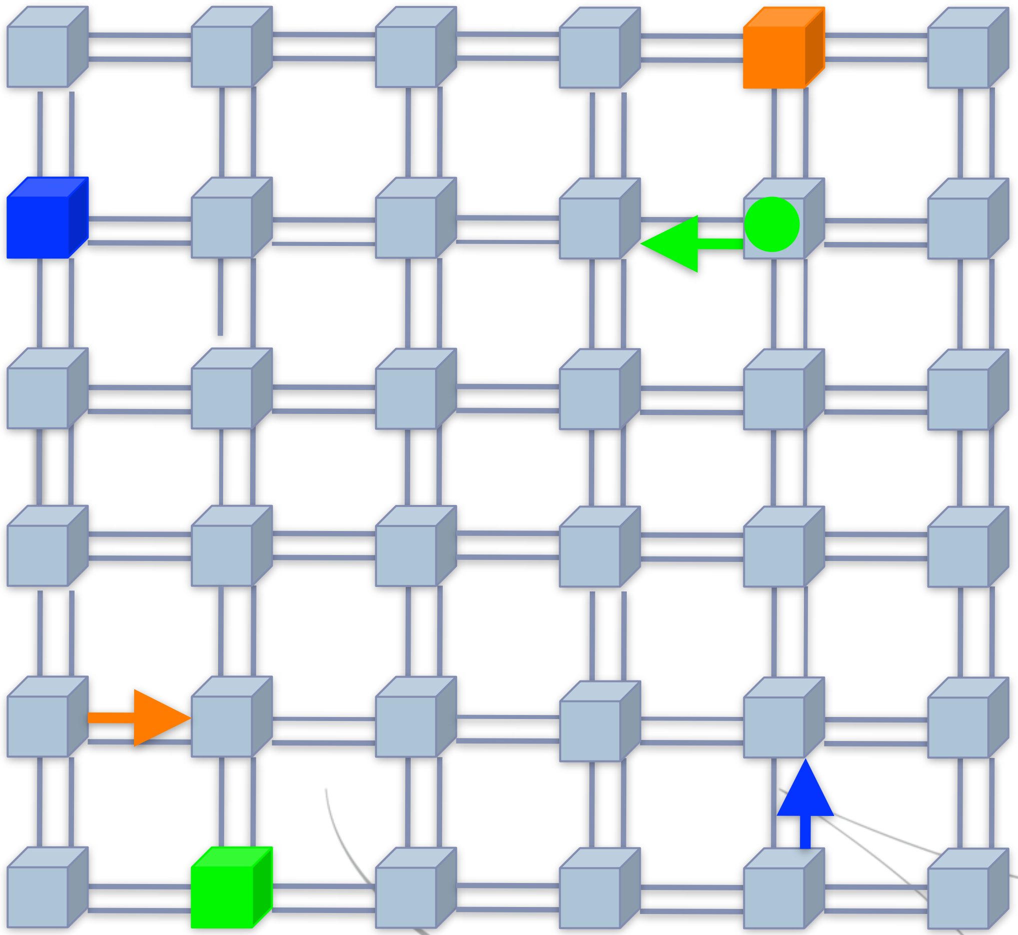
- Application
  - Arbitrary large 2D-mesh with XY routing

- N.B.: use a different definition template !

Method elements

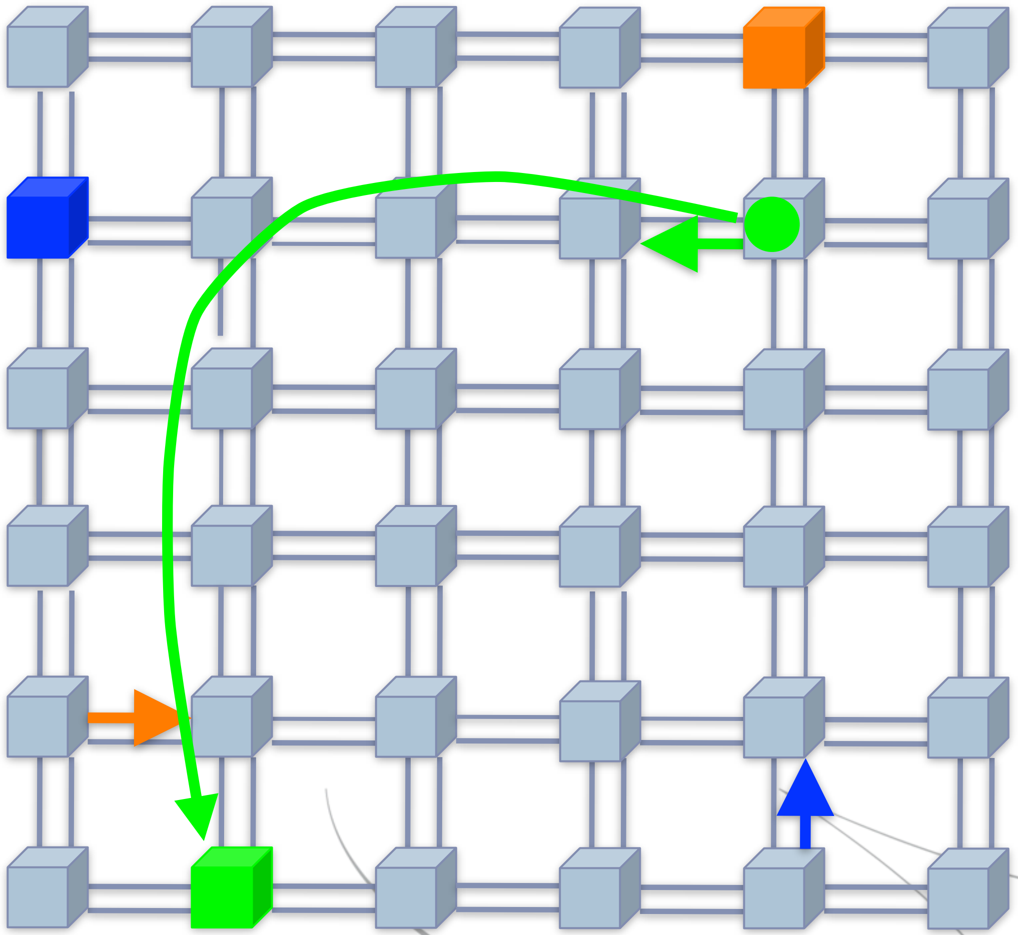
- Generic or meta-model
  - constituents
  - architectures
  - proof obligations and theorems
- Temporal abstractions
  - define maximum travel distance per time unit

The temporal abstractions (1): time = source to destination



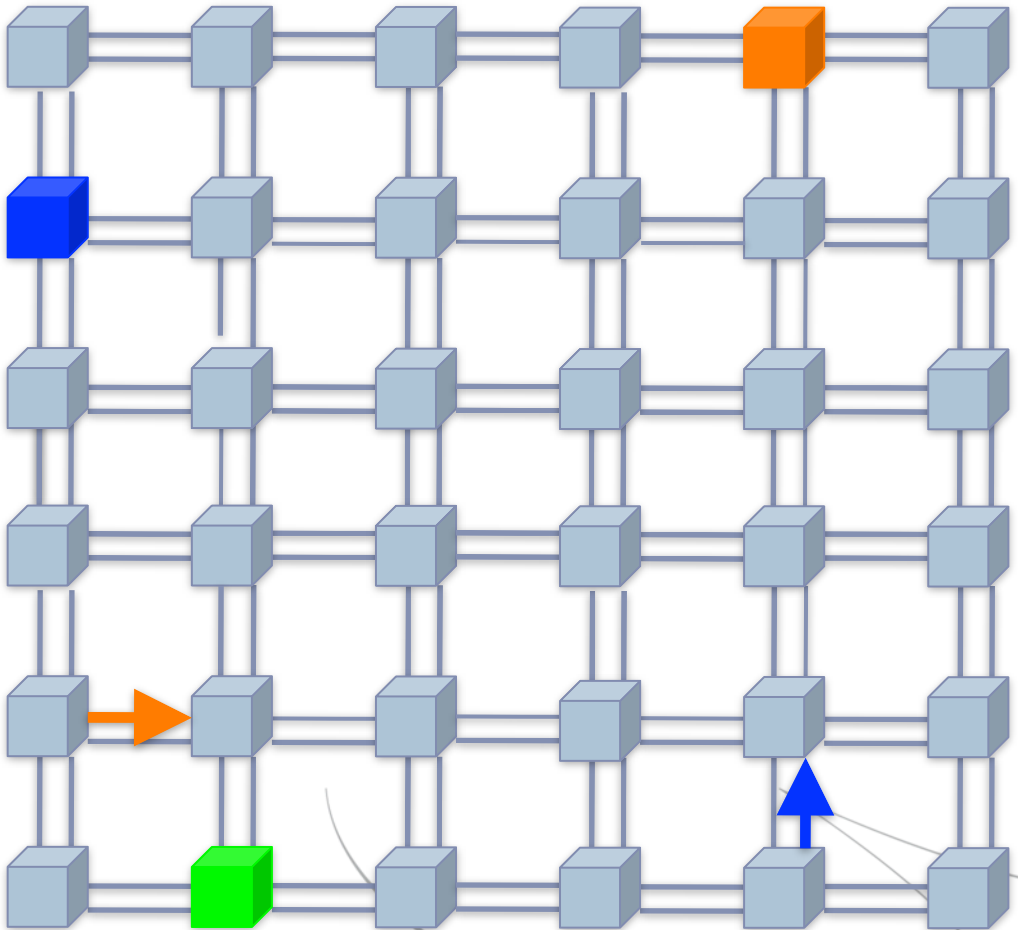
Green chosen first and reaches its destination *in one step*

The temporal abstractions (1): time = source to destination



Green chosen first and reaches its destination *in one step*

The temporal abstractions (1): time = source to destination

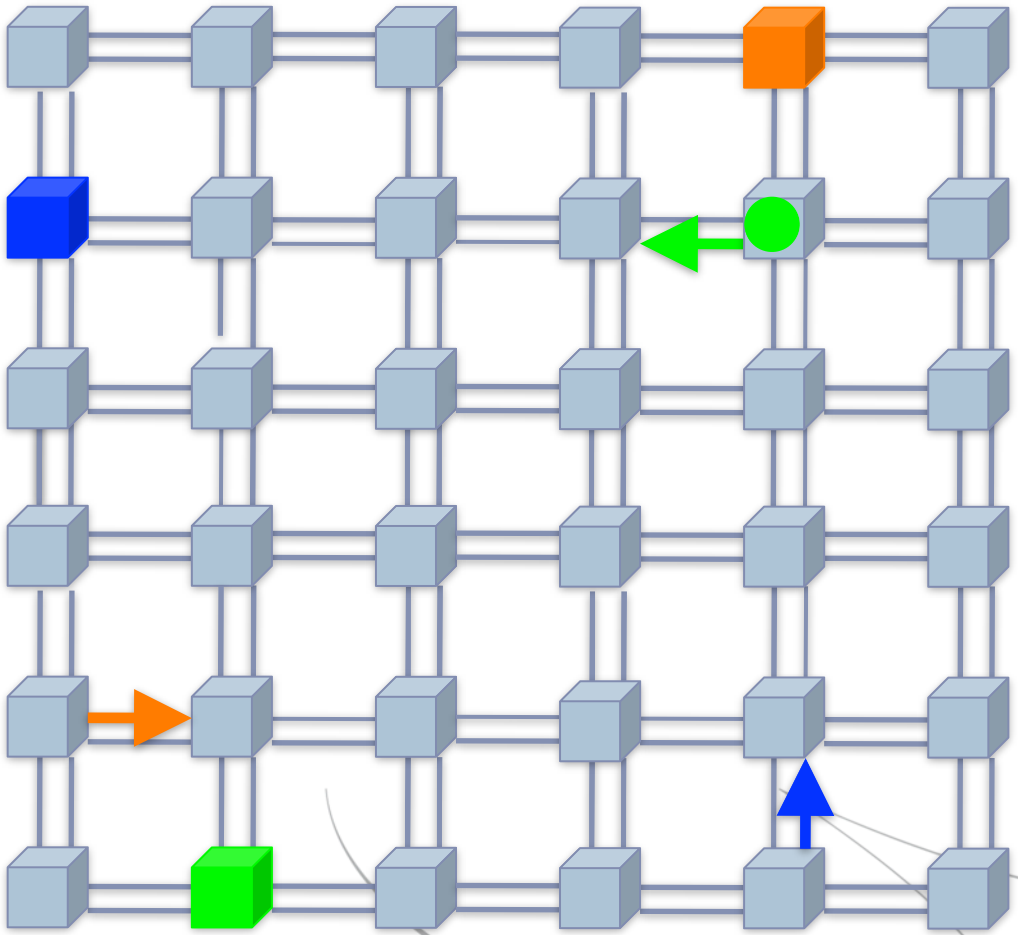


Then blue reaches its destination *in one step*

The temporal abstractions (1)

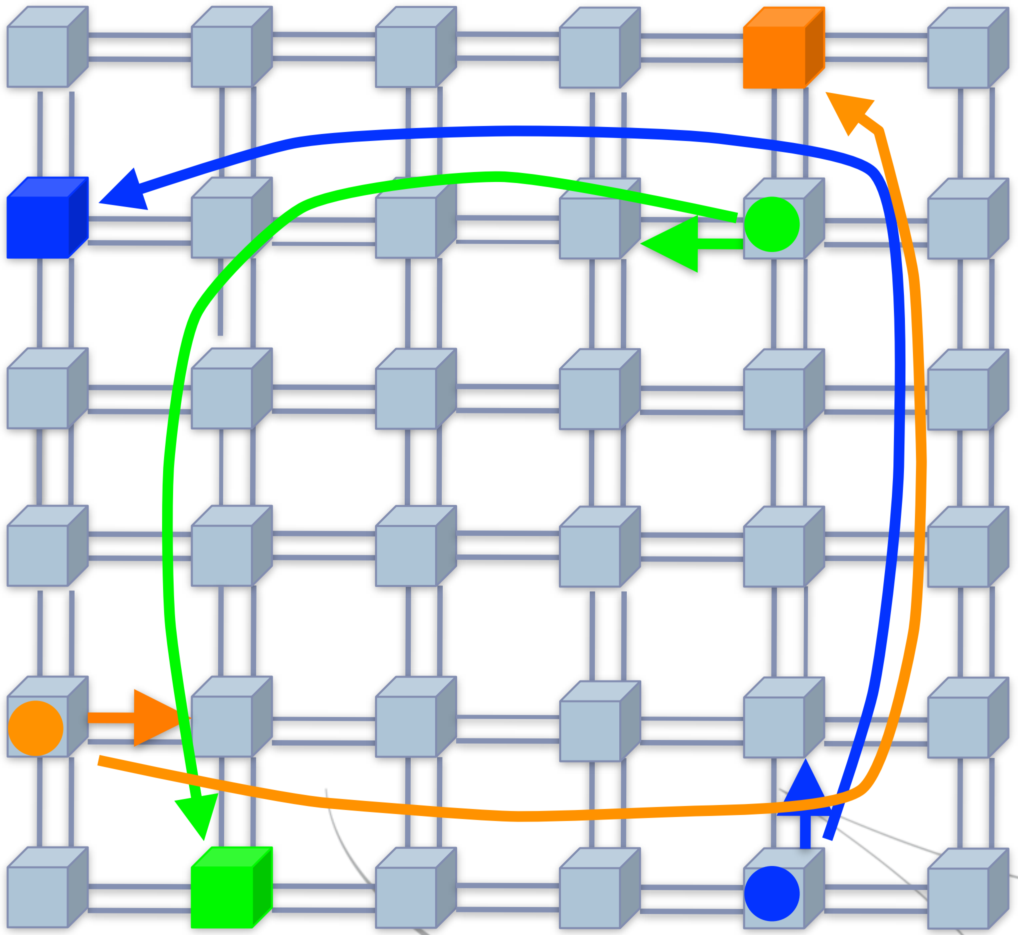
- Restrictions
  - No deadlock possible
  - Non minimal adaptive routing not possible
- Routes computed from source to destination
- Scheduling decision from source to destination
- Global Properties
  - routes are valid
  - messages reach their expected destination
- Example
  - TDMA scheduling of AEthereal from NXP

The temporal abstractions (2): time = current to next



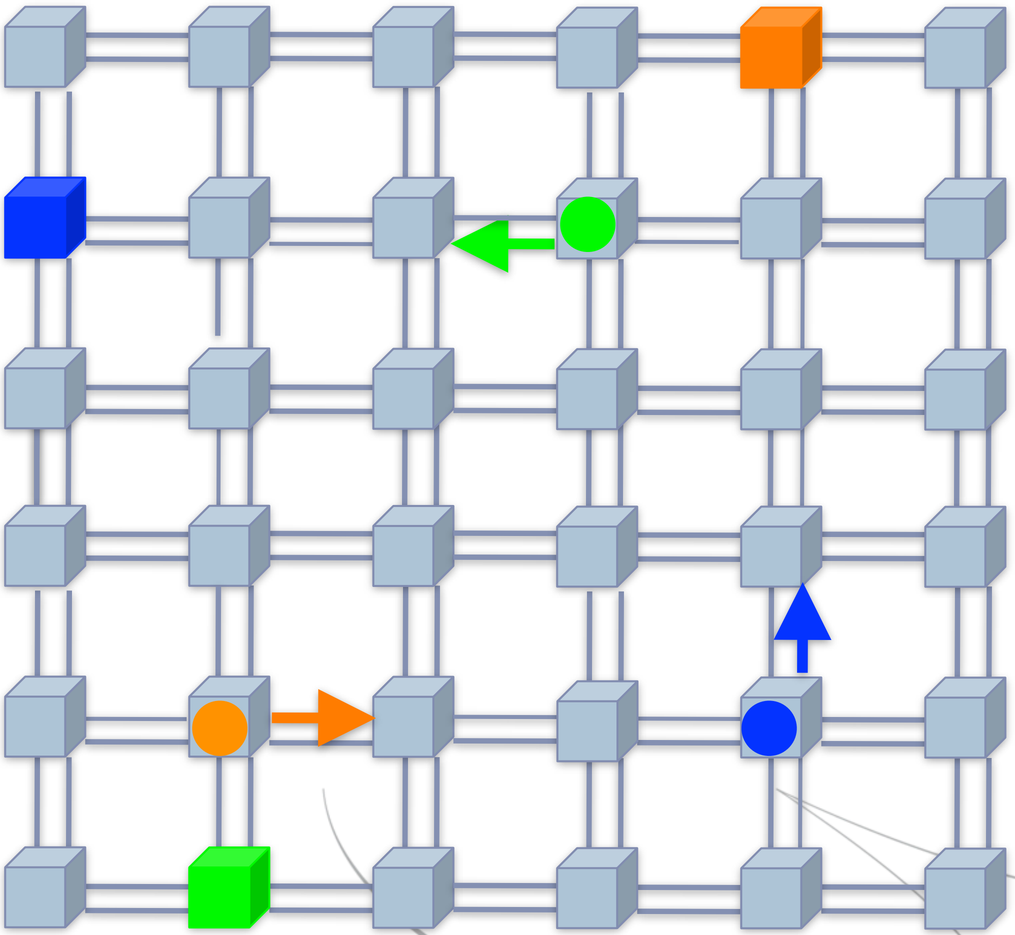
Messages advance of at most one hop *in one step*

The temporal abstractions (2): time = current to next



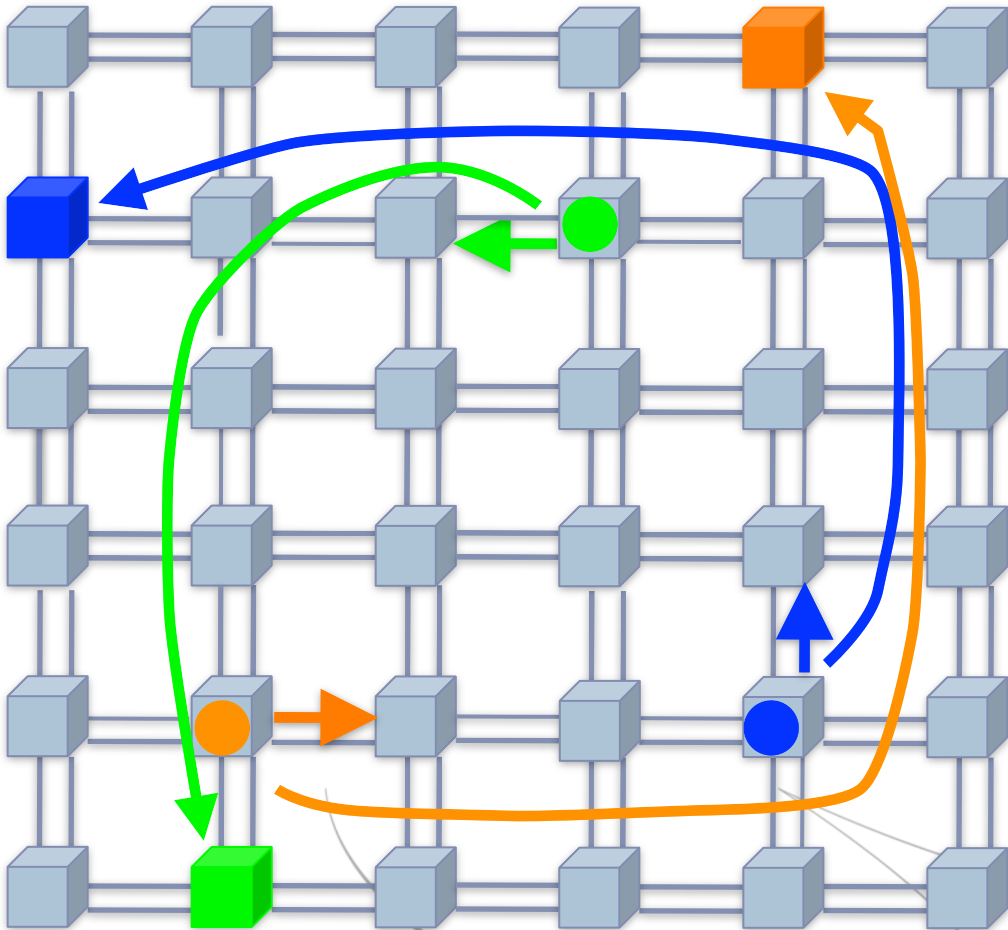
Messages advance of at most one hop *in one step*

The temporal abstractions (2): time = current to next



Messages advance of at most one hop *in one step*

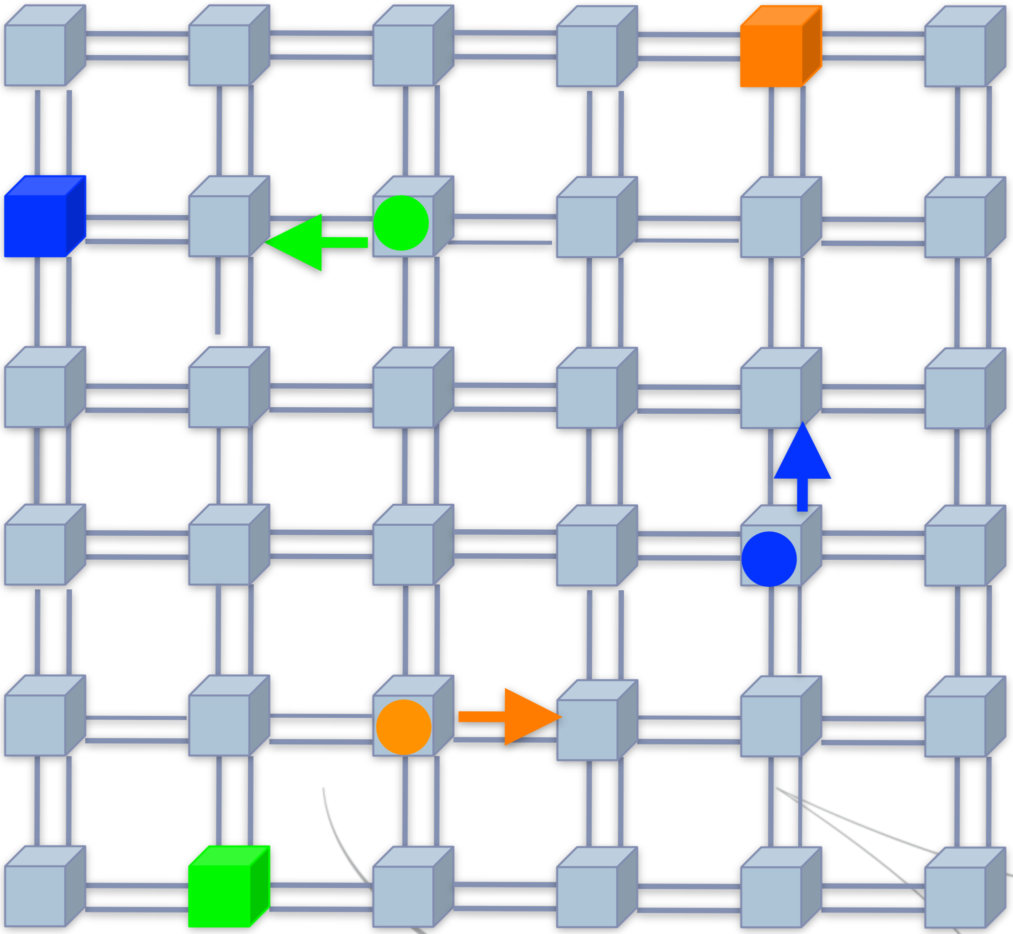
The temporal abstractions (2): time = current to next



Messages advance of at most one hop *in one step*

www.ou.nl

The temporal abstractions (2): time = current to next

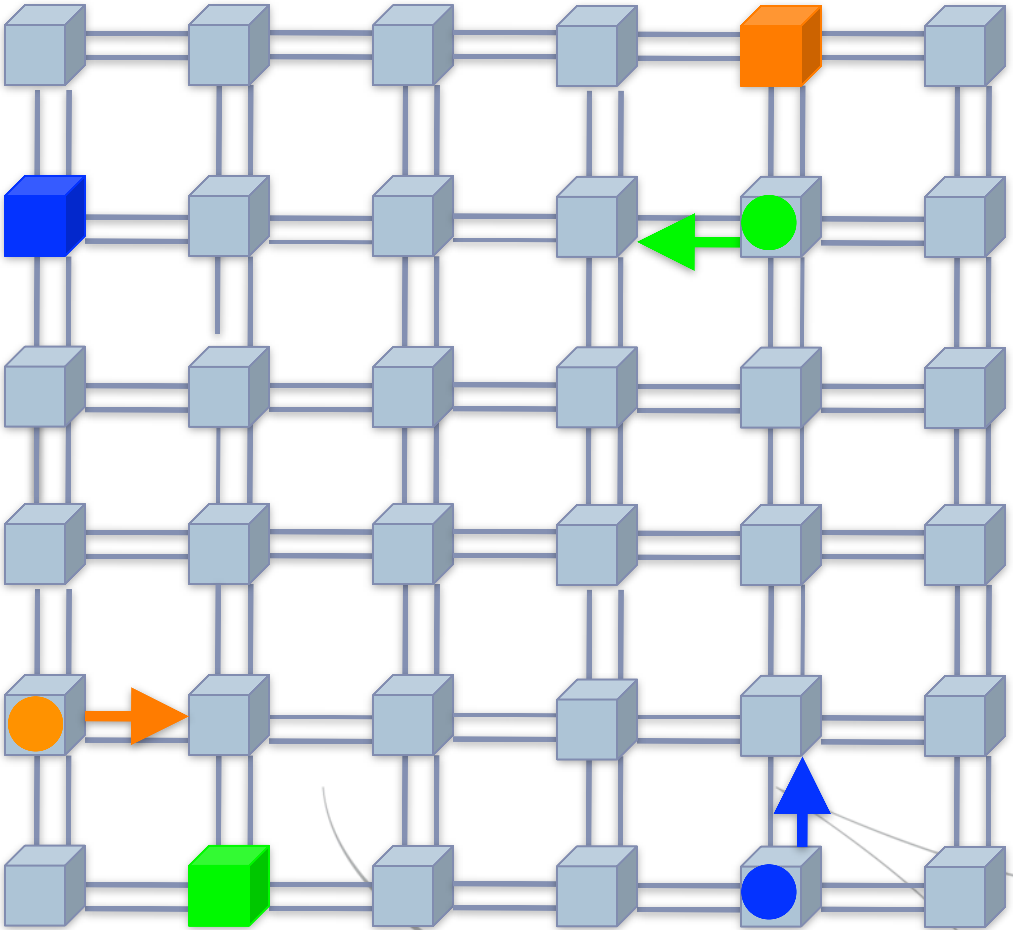


Messages advance of at most one hop *in one step*

The temporal abstractions (2): time = current to next

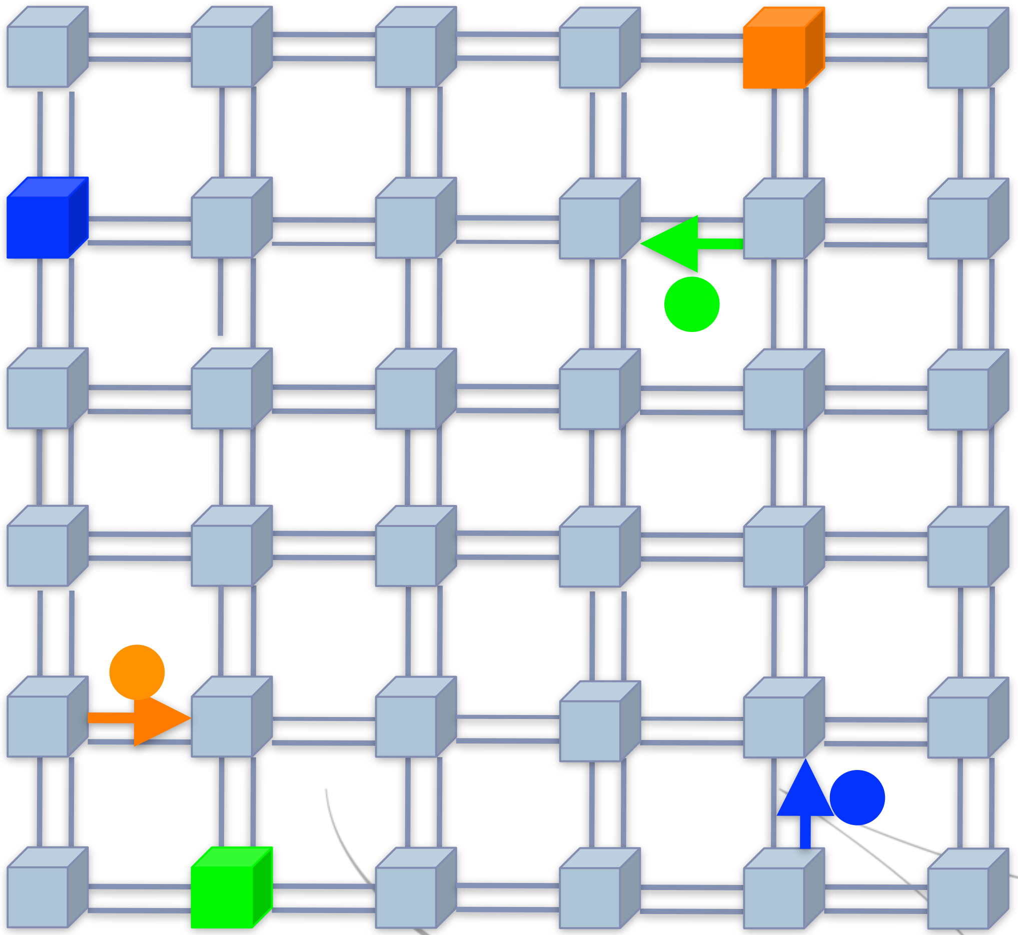
- Deadlock possible
- Routes computed from current to destination
- Scheduling decisions from current to next
- Global Properties (preserved)
  - routes are valid
  - messages reach their expected destination
- Global properties (new)
  - no deadlock
  - no livelock
  - evacuation
- Still ...
  - Complete routes known at all times
  - Node reads neighbour to check available space

The temporal abstractions (3): time = one router



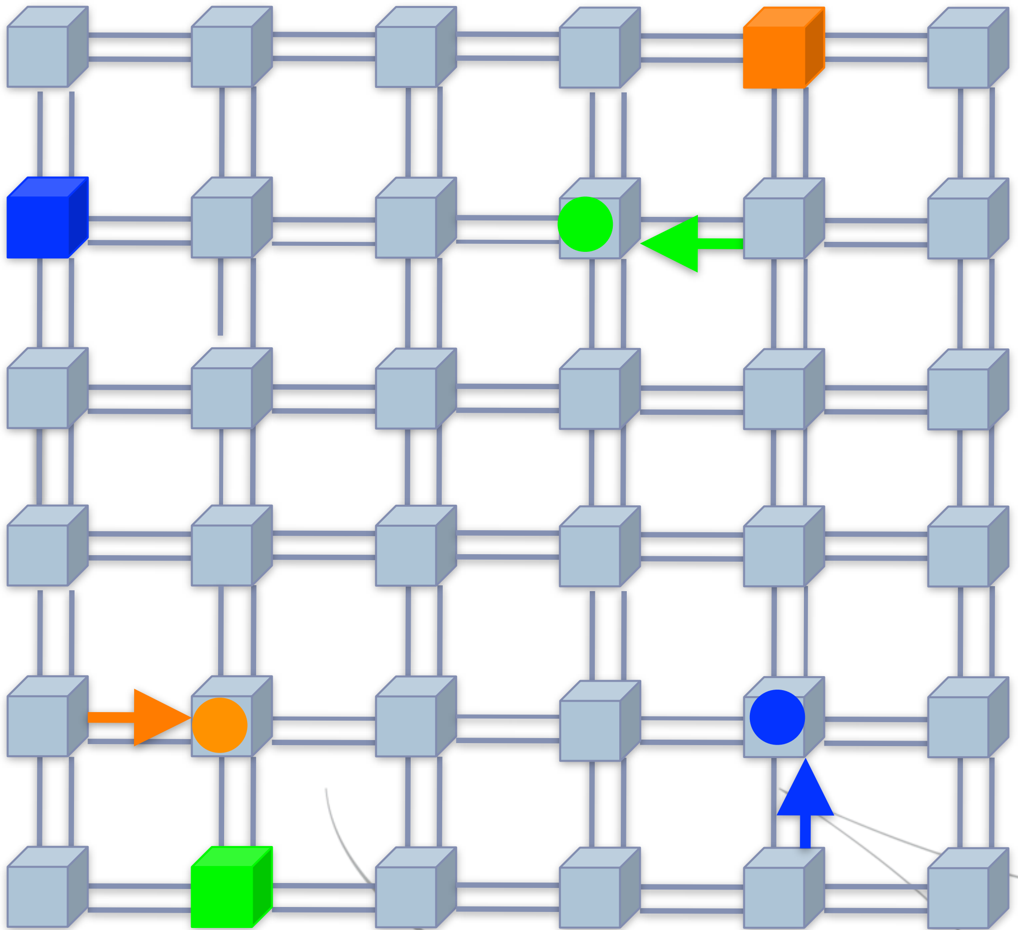
Messages cross **at most one node** *in one step*

The temporal abstractions (3): time = one router



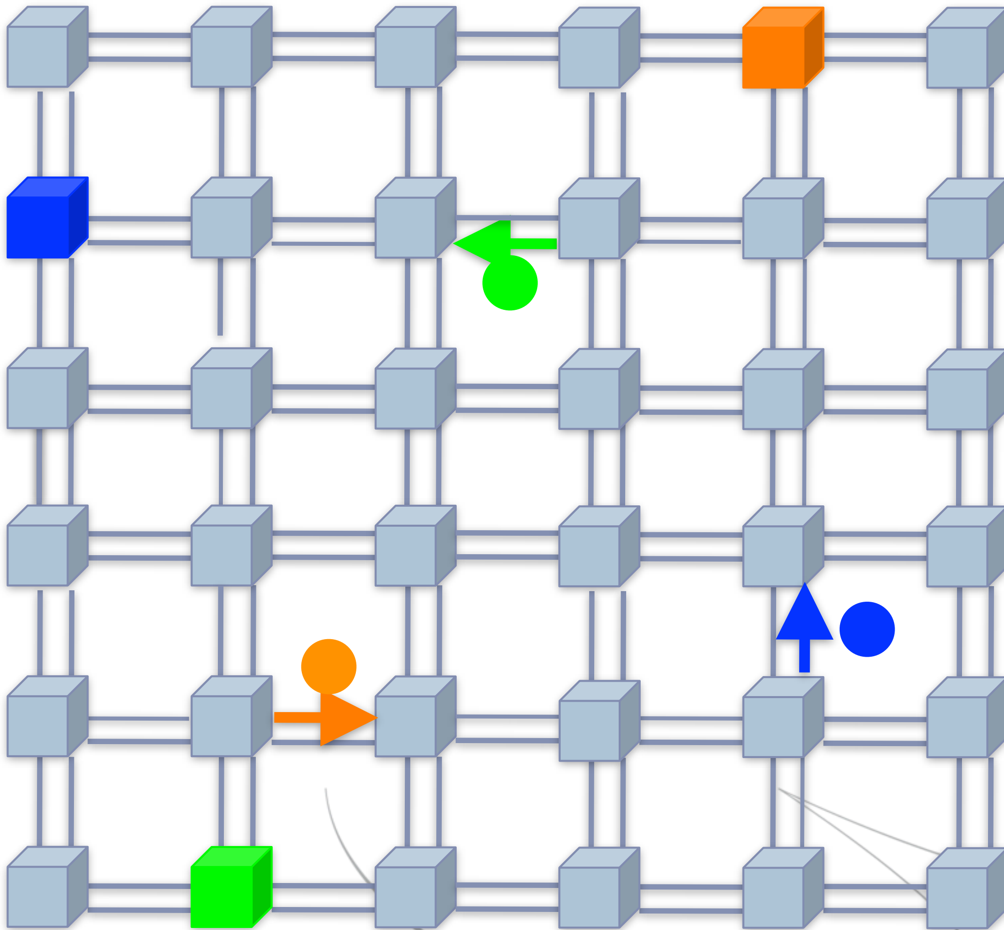
Messages cross **at most one node** *in one step*

The temporal abstractions (3): time = one router



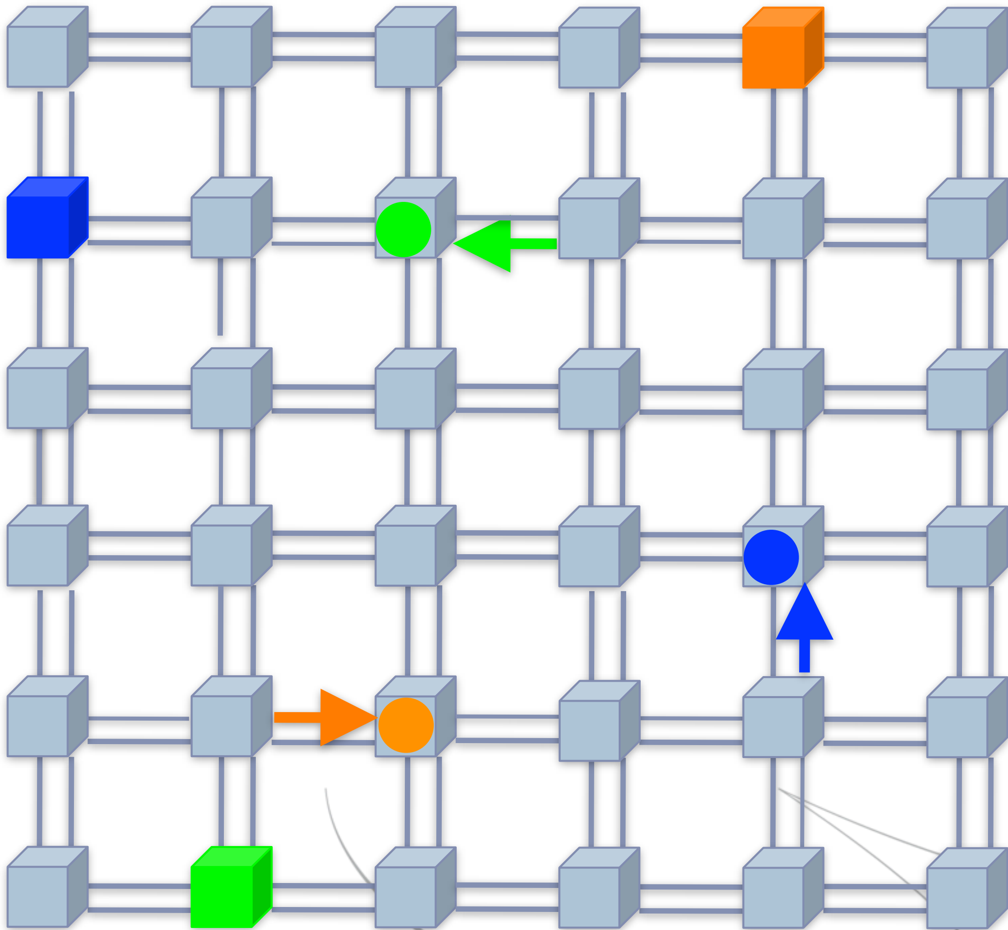
Messages cross **at most one node** *in one step*

The temporal abstractions (3): time = one router



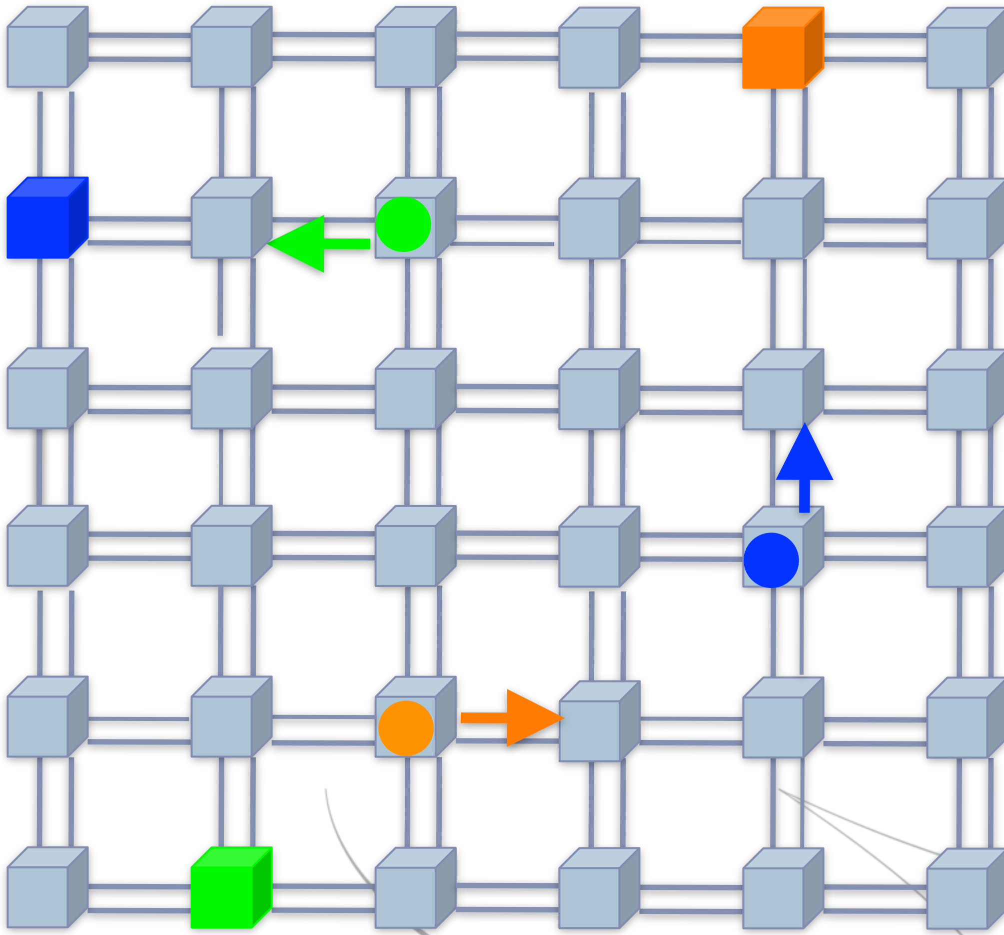
Messages cross **at most one node** *in one step*

The temporal abstractions (3): time = one router



Messages cross **at most one node** *in one step*

The temporal abstractions (3): time = one router



Messages cross **at most one node** *in one step*

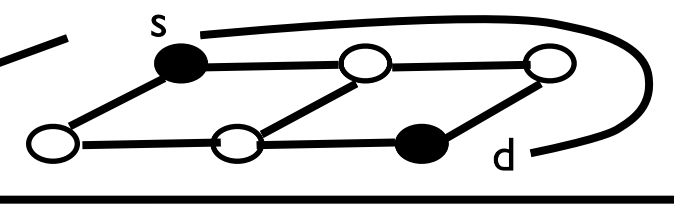
The temporal abstractions (3): time = one router

- Messages on links can be lost
- Read signals to take decision
- Routes computed from current to next
- Scheduling decisions from current to next
- Global properties (preserved)
  - routes are valid
  - messages reach their expected destination
  - no deadlock
  - no livelock
  - evacuation
- Global properties (new)
  - no message lost
  - correctness of handshake protocol

The temporal abstractions: Summary

Routing: source to destination

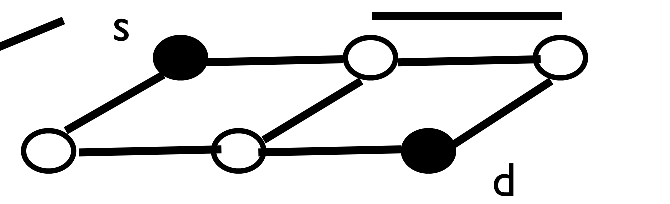
Scheduling: source to destination



in one step from source to destination

Routing: source to destination

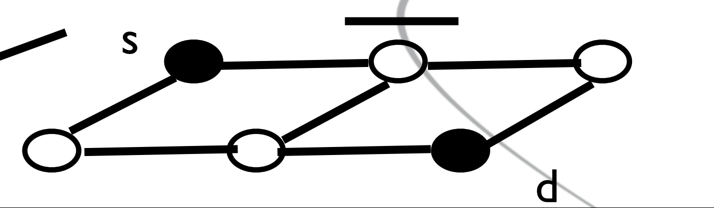
Scheduling: current to next



in one step from current to next

Routing: current to next

Scheduling: current to next



in one step from input to output

The temporal abstractions: objective

Find proof obligations sufficient to maintain stuttering (bi)simulations between the architecture templates of two abstraction levels

Part II

Temporal abstraction (2): time = one hop

Architecture template at temporal abstraction (2)

Let σ be a configuration containing a state and messages
Let M be a set of messages to be sent over the NoC

GeNoC (σ) =

σ iff σ.M = ∅ // empty list of messages

σ iff deadlocked(Routing(Injection(σ)))

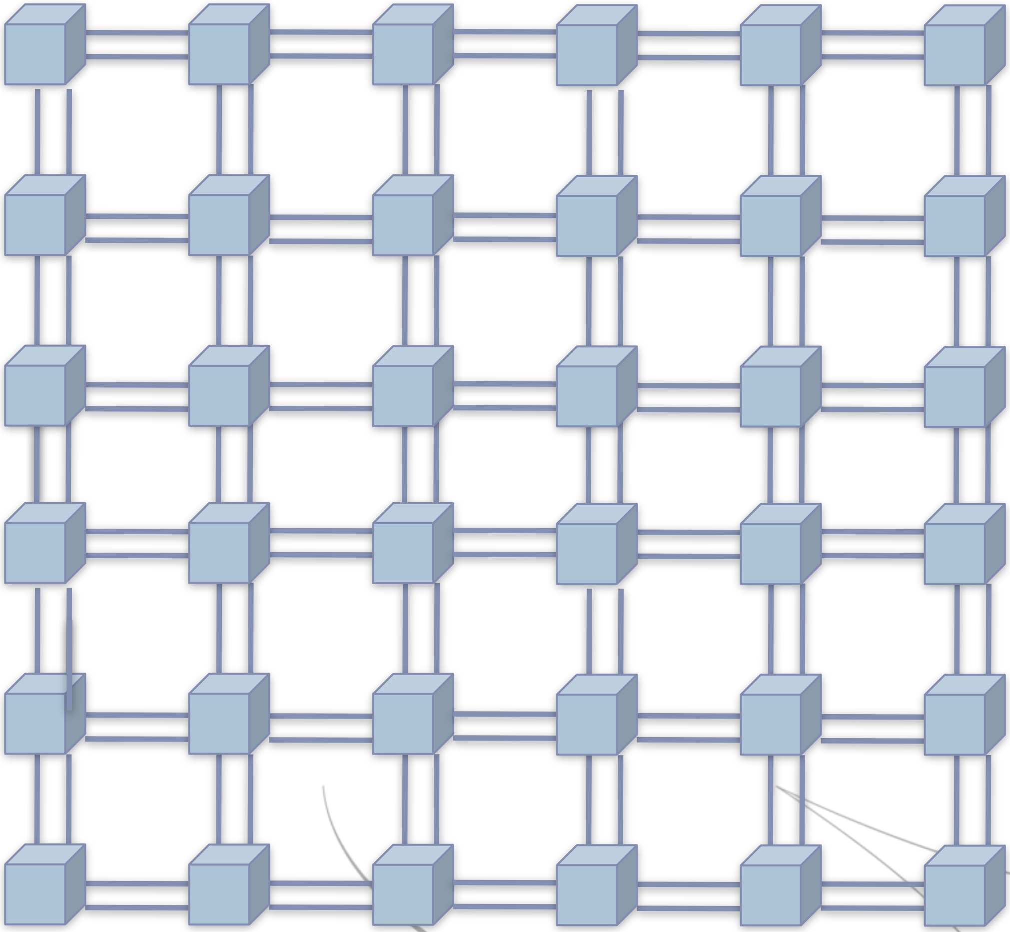
GeNoC(***Scheduling*(*Routing*(*Injection*(σ))))**

Advance of one hop if possible → *Scheduling*

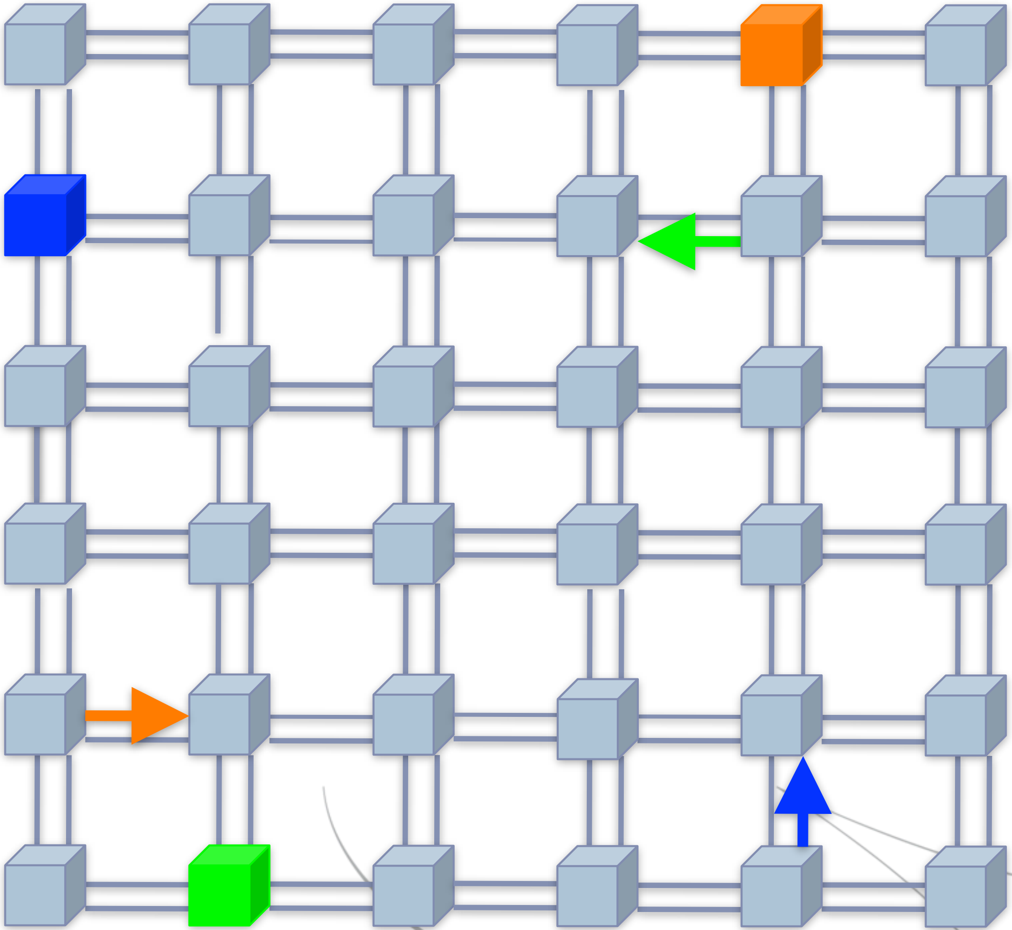
Routes from current to destination → *Routing*

inject messages → *Injection*

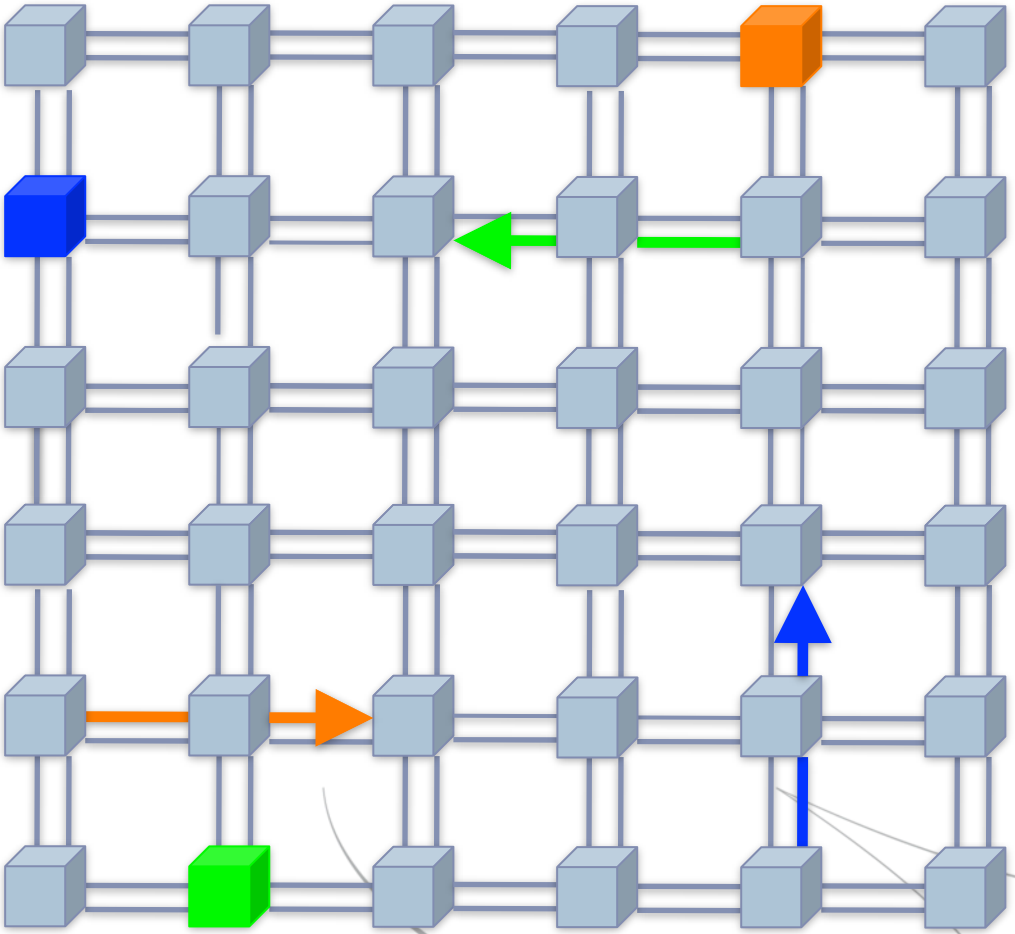
Deadlock



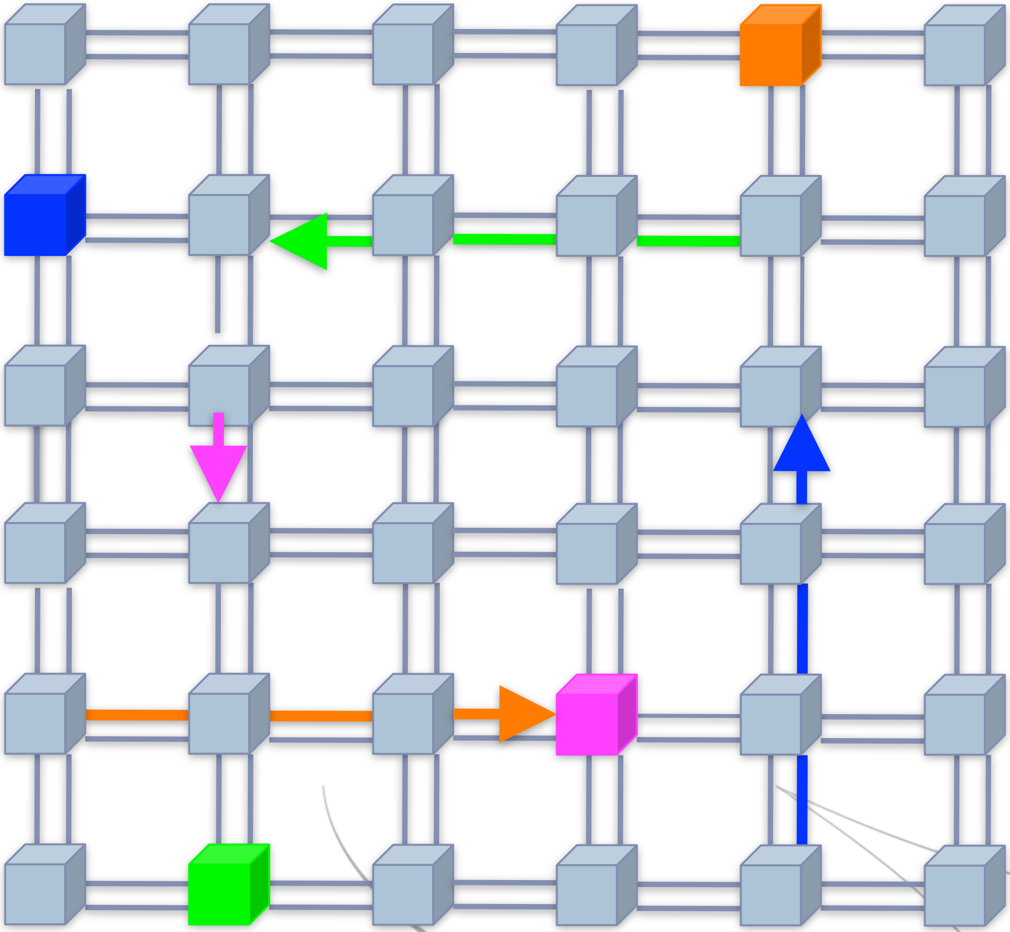
Deadlock



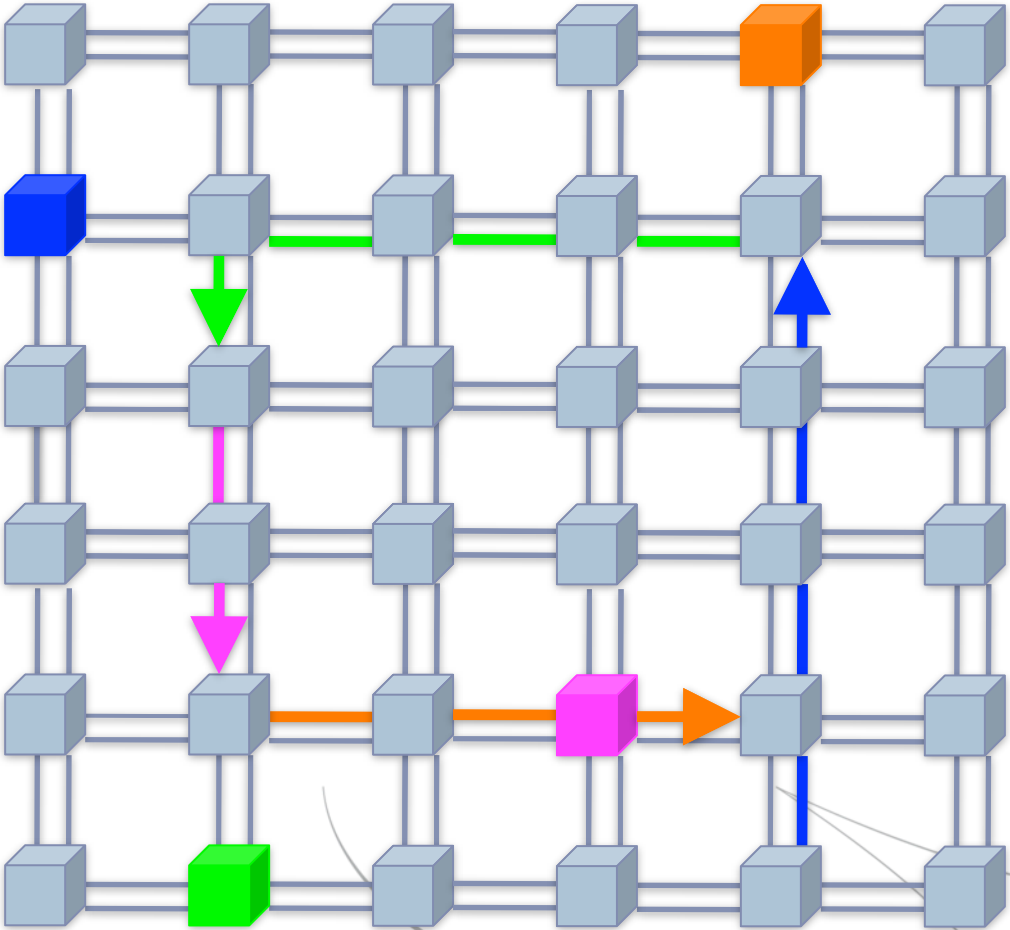
Deadlock



Deadlock



Deadlock



Deadlock is an emerging property

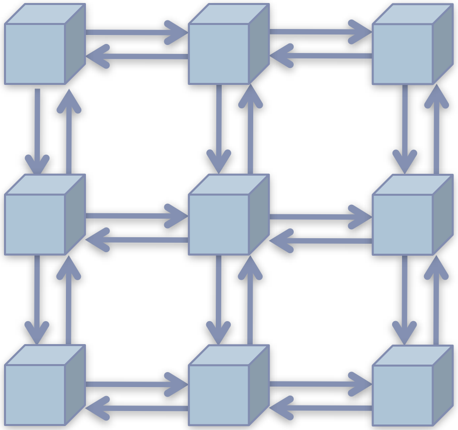
Deadlock generic theorem

- There is no deadlock iff there is no configuration where all messages are blocked
- A message is blocked iff its next hop is full
- Deadlock-free theory:
  - there is no deadlock configuration iff the routing function has an acyclic dependency graph
- New proof obligations about the dependency graph (G)

1. Each dependency in the network is an edge in G
2. All edges of G are dependencies
3. Graph G is acyclic.

NoC example: 2D-mesh Hermes NoC

Topology



Routing:

XY Routing

Injection

Immediate

Switching policy

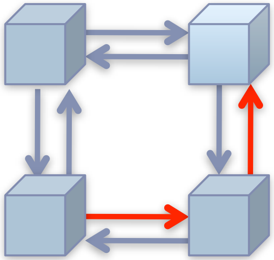
Wormhole

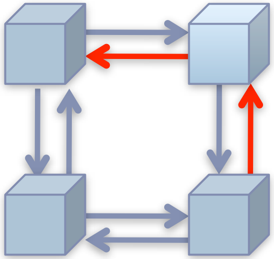
Packet

Specifying routing algorithm and dependency graph

- Specify XY routing algorithm
- Specify dependency graph

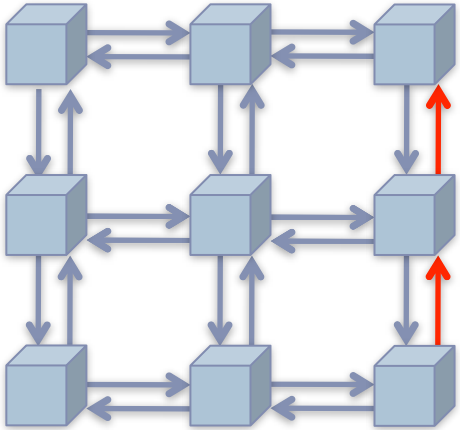
Dependency: No dependency:

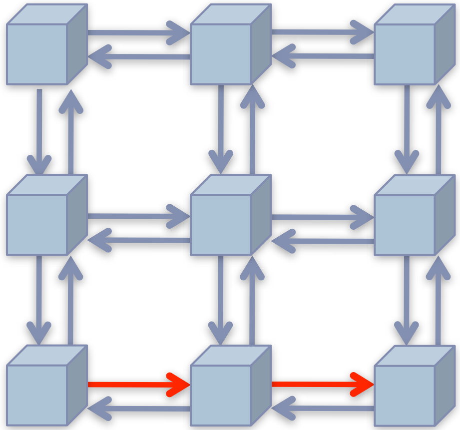




Proving dependency graph acyclic (arbitrary large network)

- Prove graph acyclic
  - proof based on the concept of flows





East flow:
• Increase the x-coordinate
or
•Go into a vertical flow
or
• Stop

North flow:
• Increase the y-coordinate
or
• Stop

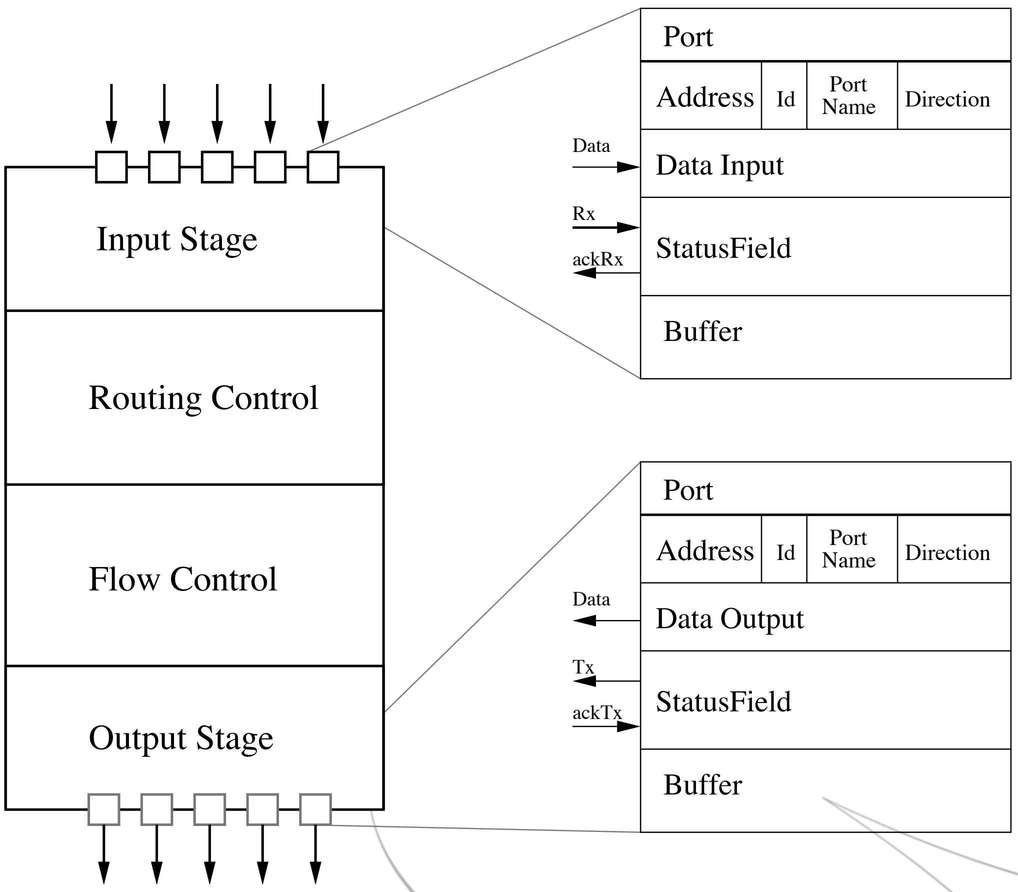
Part III

Temporal abstractions (3) / Time = one router

Temporal abstraction (3)

- Architecture template based on generic router
- Source routing architecture
- Distributed routing architecture
- Refinement theorem

Architecture template at temporal abstraction (3): router model



Architecture template at temporal abstraction (3) *

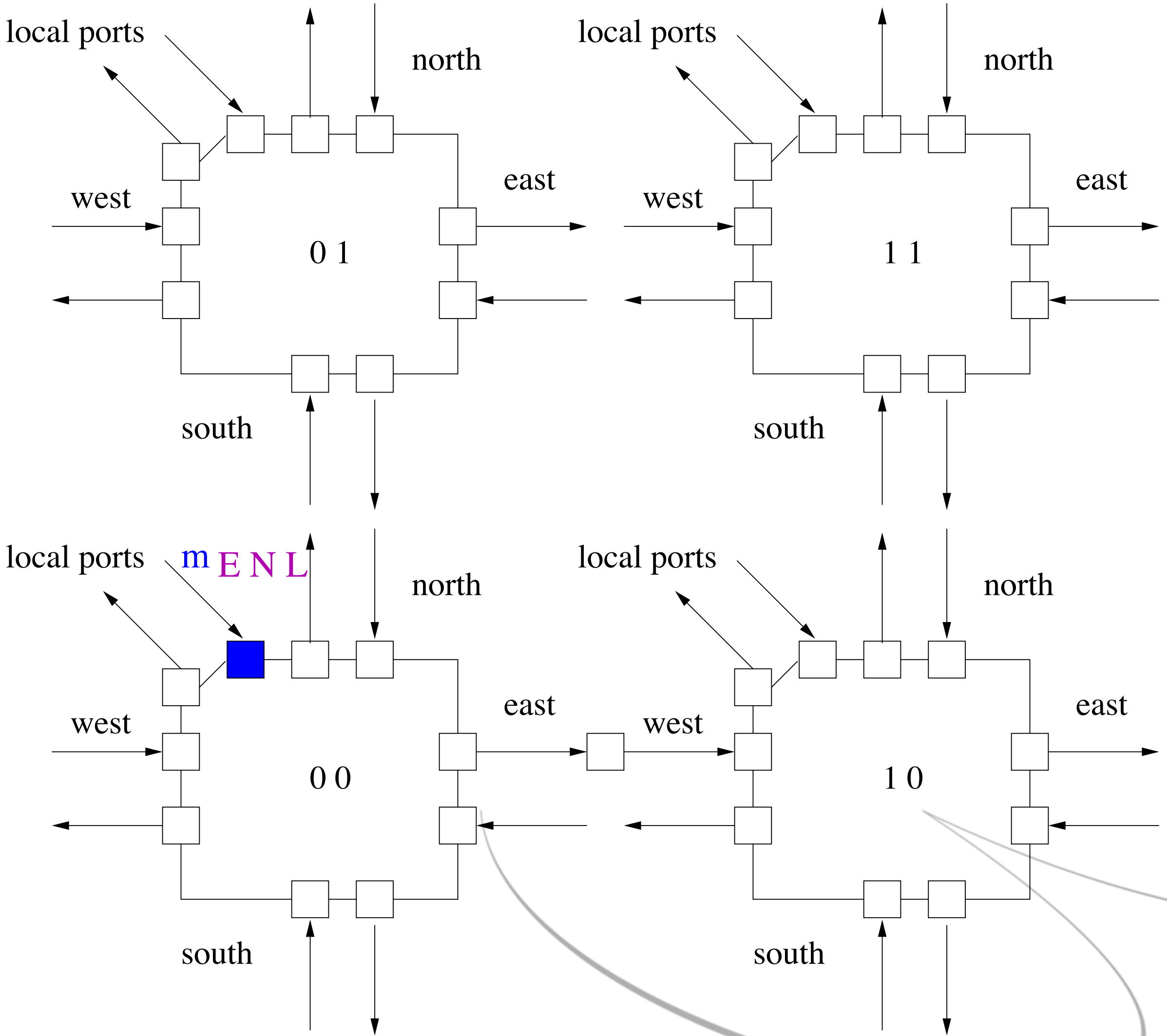
Let σ be a configuration containing a state and messages
Let M be a set of messages to be sent over the NoC

	σ iff σ.M = ∅ // empty list of messages
GeNoC (σ) =	σ iff deadlocked(Routing(Injection(σ)))
	GeNoC(***Apply-Router-All***(σ))))

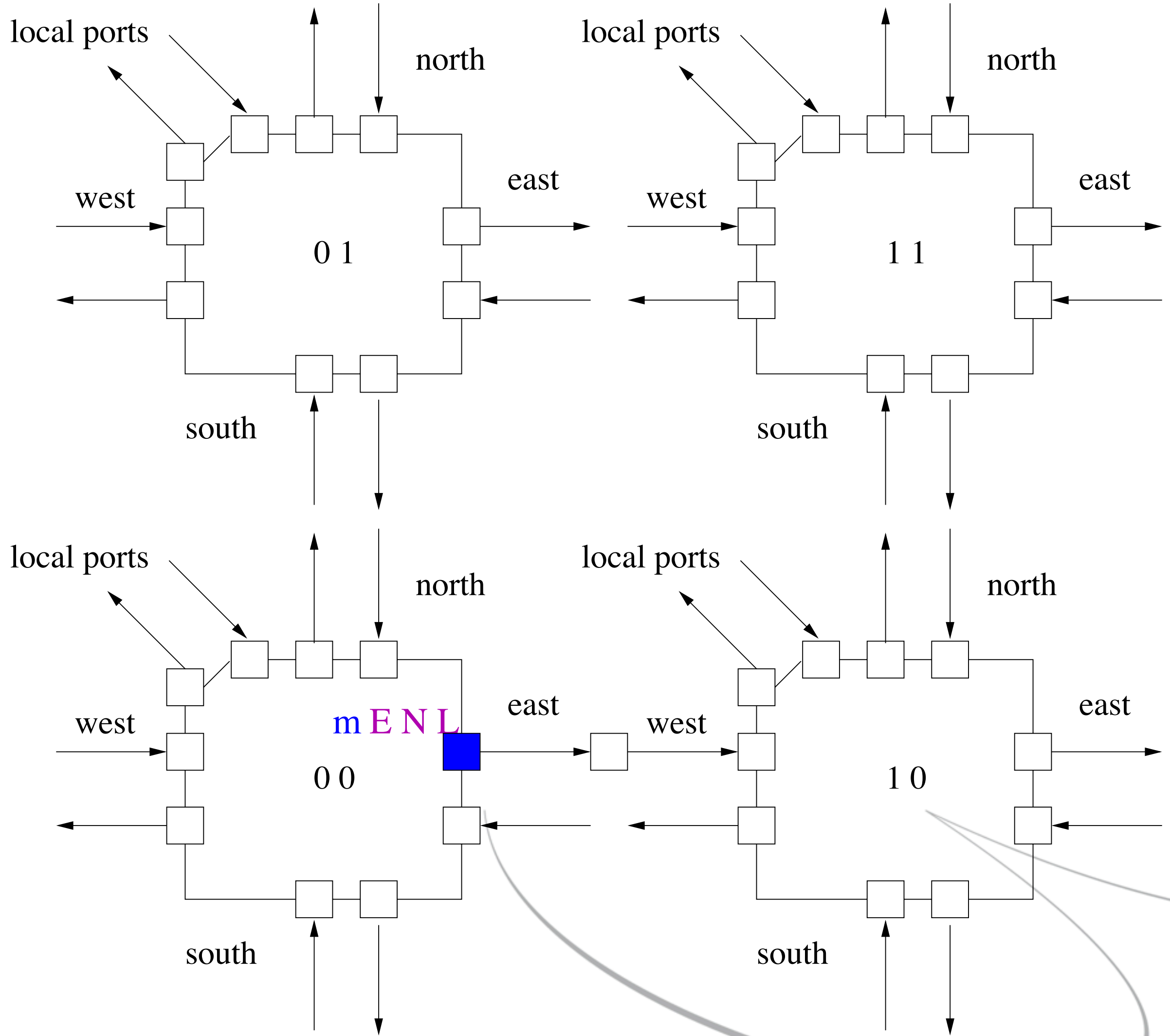
↑ Execute the router model at all nodes

* We do not have exactly that model yet

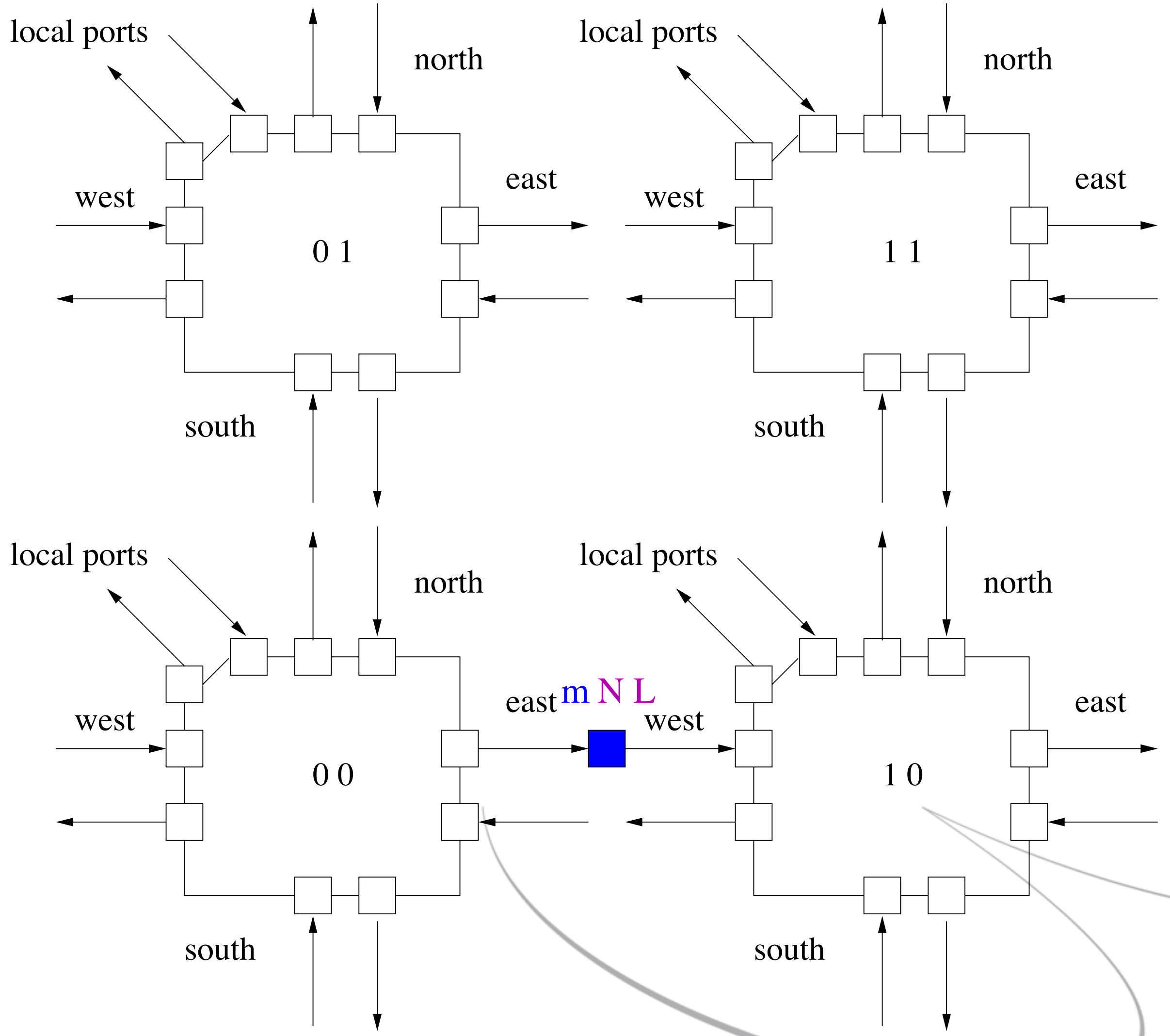
Architecture (1): Source routing / route encoded in messages



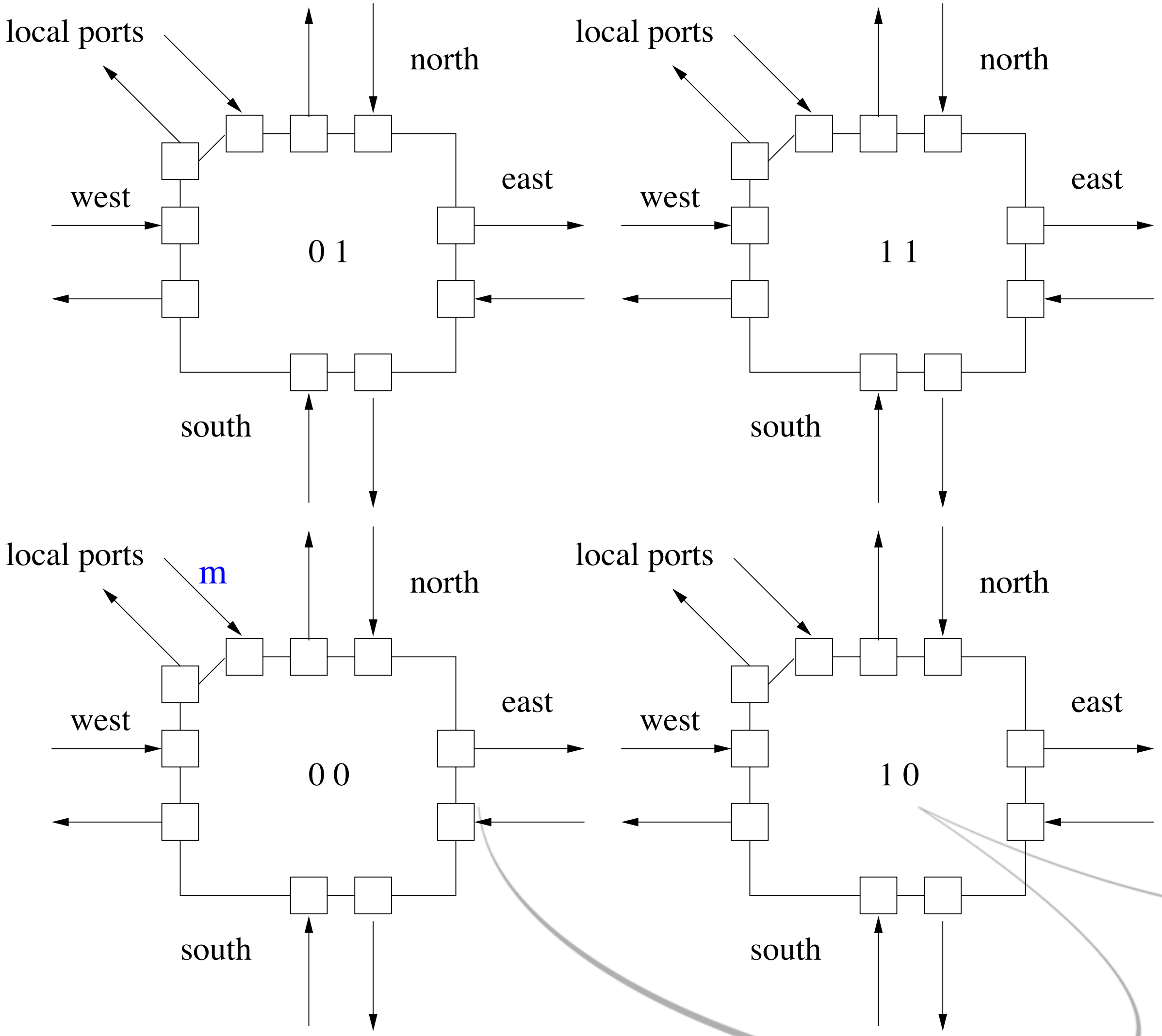
Architecture (1): routing decision read in messages



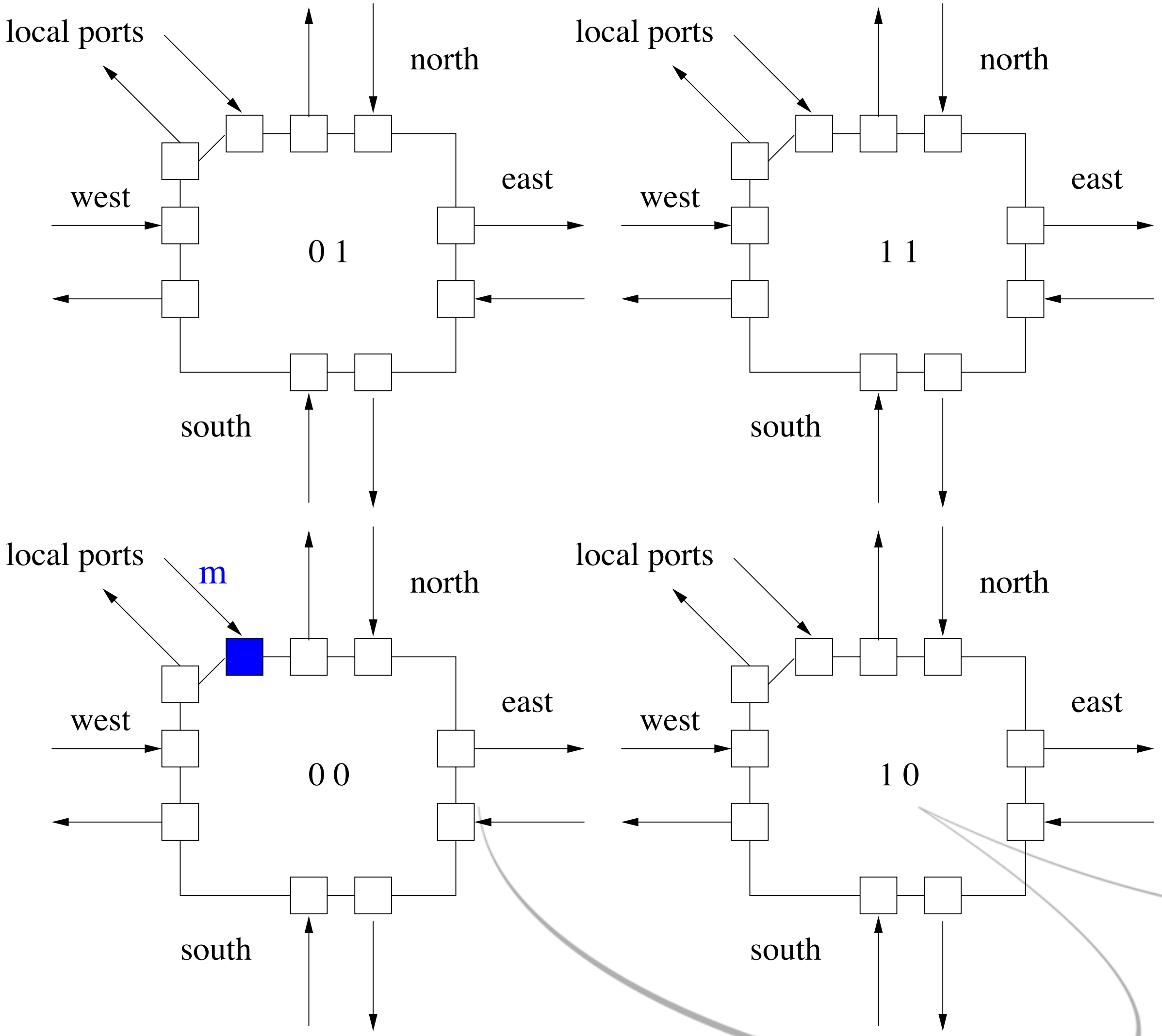
Architecture (1): new head of routes



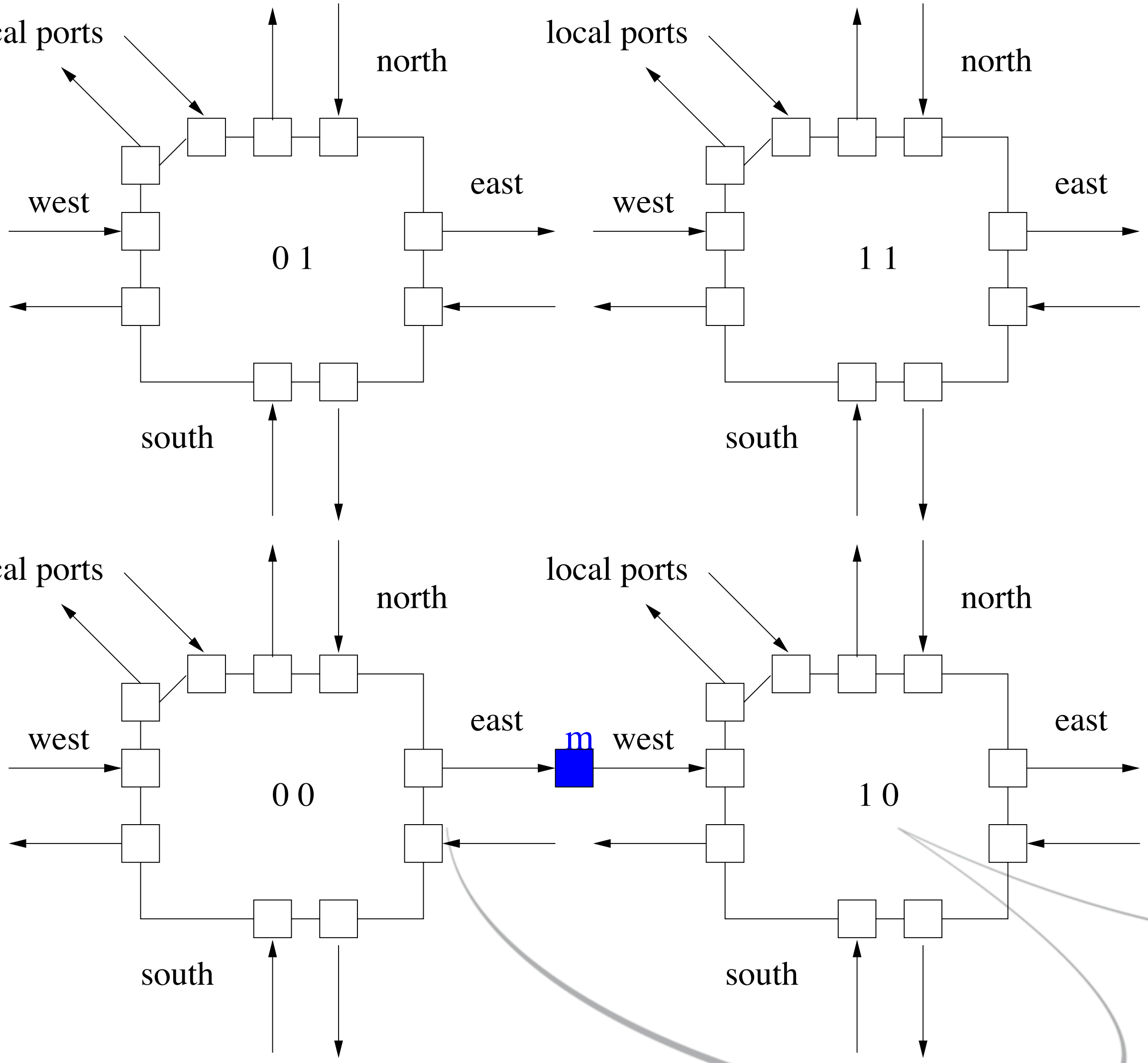
Architecture (2): Distributed routing



Architecture (2): Distributed routing / local routing logic

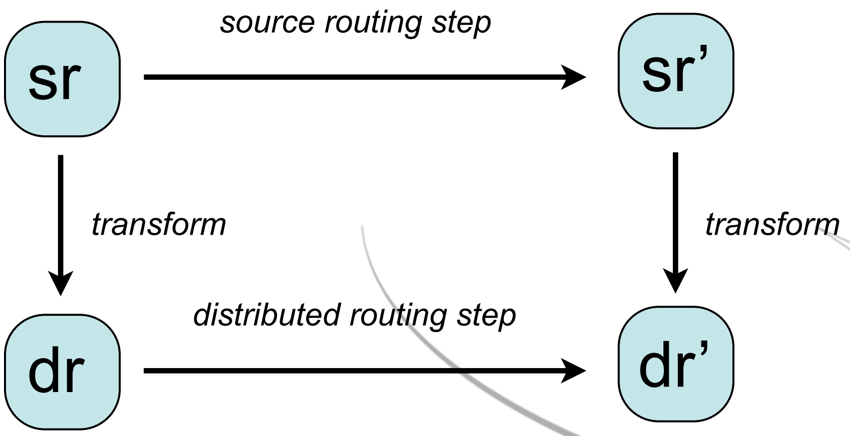


Architecture (2): Distributed routing / local routing logic



Refinement theorem (1)

- Distributed routing architecture refinement of source routing one
  - Given the same inputs, produces same outputs
  - Messages use identical paths
- Function *transform* removes encoded route from messages
- Proved following diagram



Refinement theorem (2)

- Proof between *generic models*
- Main proof obligation
  - head of the pre-computed route = local calculation

Part IV

Next steps / current work

Formalizing theories for deadlock-free routing

- Deterministic routing
  - acyclic dependency graph iff no deadlock
  - Dally and Seitz (87')
- Adaptive routing
  - exists subrouting function with acyclic extended dependency graph iff no deadlock
  - adaptive routing function with cyclic dependencies are OK
  - Duato (93')
- Formalized in ACL2 our own conditions
  - based on port dependency graphs
  - deterministic and adaptive routing
  - wormhole and store-and-forward networks
  - used temporal abstraction (2) / (S) = current to next

Verified algorithm to prove these conditions

- Deterministic routing
  - acyclic dependency graph iff no deadlock
  - linear search for cycles is enough
- Adaptive routing in store-and-forward networks
  - exists routing subfunction with acyclic extended dependency graph iff no deadlock (Duato)
  - all subgraphs have an escape iff no deadlock (our condition)
  - algorithm with time complexity in O(|E|)
- Adaptive routing in wormhole networks
  - algorithm for sufficient condition in O(|E|)
  - checking necessary and sufficient condition is NP-complete ?
- Implementations of algorithms still not efficient enough

Part III

General Conclusion and Future Work

Summary

- A meta-model
  - Local proof obligations imply global properties
  - Functional correctness, deadlock and evacuation
  - Generic constituents and concrete instances

- Expressed in math and in the logic of ACL2
  - Executable instances
  - Same models for proofs and simulations
  - Simulation traces comparable to RTL

- Wide range of applications
  - The HERMES NoC: academic design
  - Spidergon: industrial design
  - Nostrum (Grenoble VDS group): non minimal adaptive routing

Future Work

- Unification of definition templates

- Link with RTL
  - Refine until RTL (cycle accurate)
  - Relate temporal layers (next hop full = handshake fails)

- Deadlock
  - Implement algorithms to automatic analysis of instances
  - These algorithms are verified and efficient
  - Weaken injection method's proof obligation
  - Refine termination measure (bound on evacuation time)

THANKS !!

www.ou.nl