Automatic certification and interactive theorem proving:

An impossible combination ?

Julien Schmaltz

Radboud University Nijmegen

Open Universiteit
www.ou.nl

Be provocative - Disclaimer

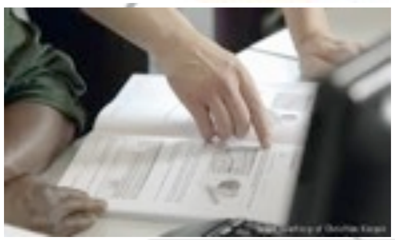
- Provocative statements - take them with a pinch of salt

About the speaker

- Limited knowledge about certification
- Interactive theorem proving systems
  - mainly ACL2
  - Isabelle on one project (1 year)
  - sharing office with Coq user
- Application domains
  - mainly on-chip interconnects
  - time-triggered hardware
- Other research projects
  - model-based testing with (Timed) LTS
  - real-time model-checking using UPPAAL
  - application to Wireless Sensor Networks

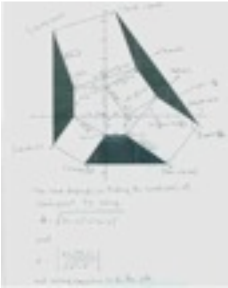
Automatic certification

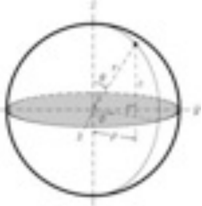
- Automatic
  - tools - easy to use and efficient
  - no human interaction - scalability
- Certification
  - high-quality design process
  - less bugs at the end

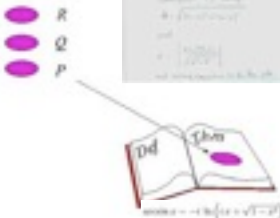


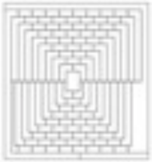
Interactive Theorem Proving

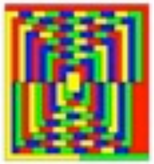
- Interactive
  - hard thinking
  - complex tools
- Theorem Proving
  - complex, tedious proofs
  - bug free but expensive
  - deep insight in products
  - true correctness

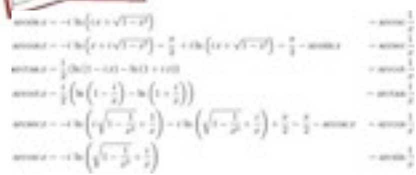




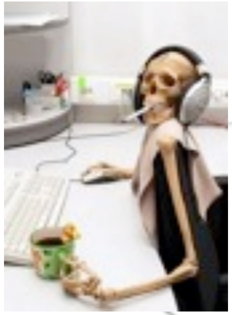


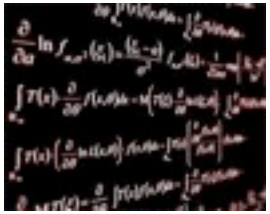






$$\frac{\bigvee_{i\in A} L_i \qquad \bigvee_{i\in B} L_i}{\bigvee_{i\in C} Subst(\theta, L_i)}$$





www.ou.nl

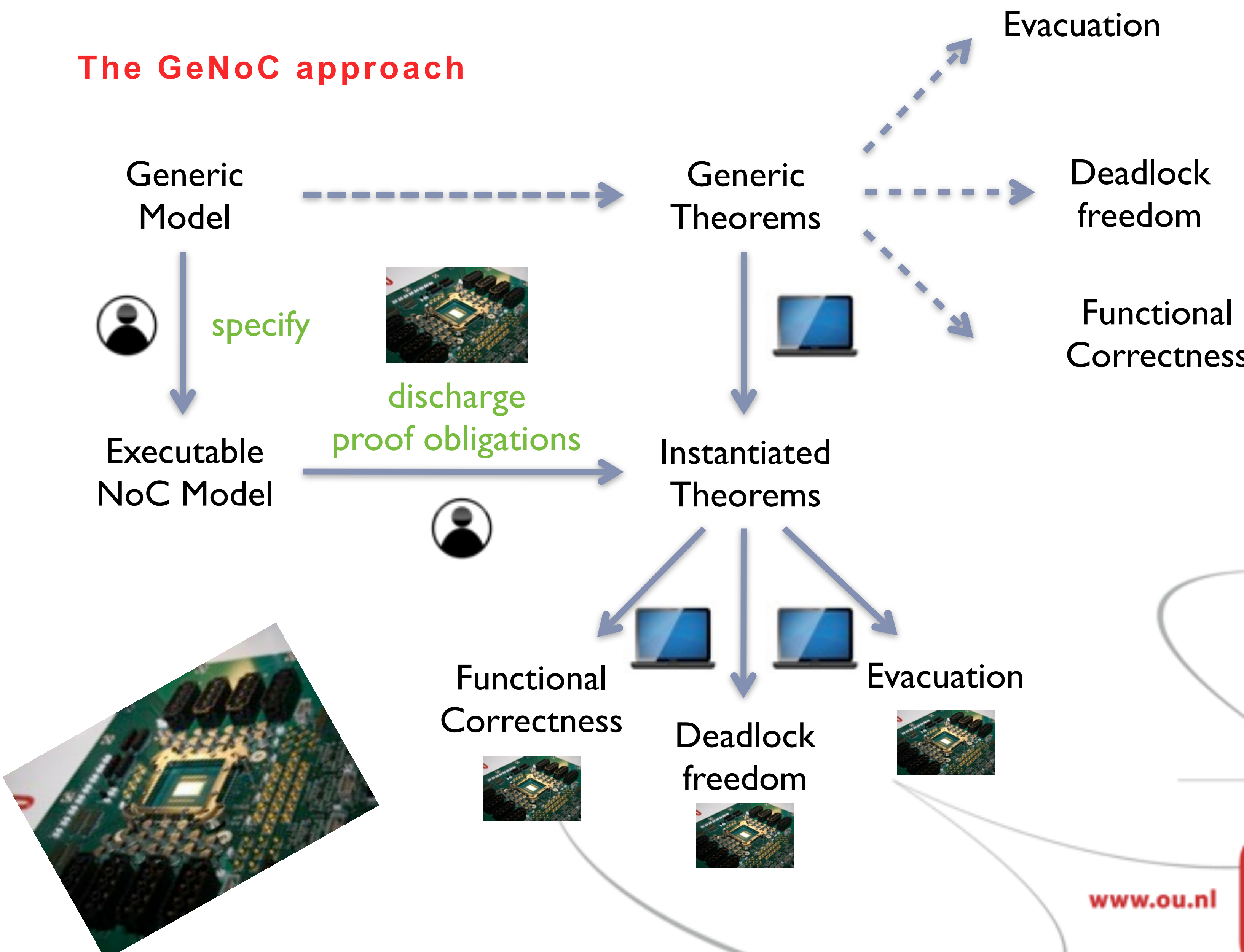
Certification vs. Theorem Proving

- Automatic Certification
  - scalability, ease of use
  - stamp about system quality
  - bug removal by good design process
  - low injection + good hunting
- Interactive Theorem proving
  - tedious proofs, complex tools, "intelligence required"
  - proof of (total) correctness
  - about systems not their design process
  - can prove tools correct
    - tools with insight and true correctness

Bugs and NoCs

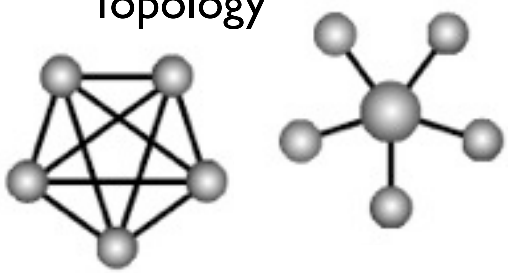
- Bug hunting - Model Checking&Co
  - algorithmic technique - automation
  - routine in HW industry
  - find subtle bugs
  - state-explosion problem - small, fixed size systems
- A mosquito-net for NoCs - The GeNoC approach
  - a *generic* model for reasoning about NoCs
  - highly *parametric*
  - generic definition of *correctness theorems*
  - identify *constraints* sufficient to prove the theorems
  - only need to check constraints on *particular instances*

The GeNoC approach

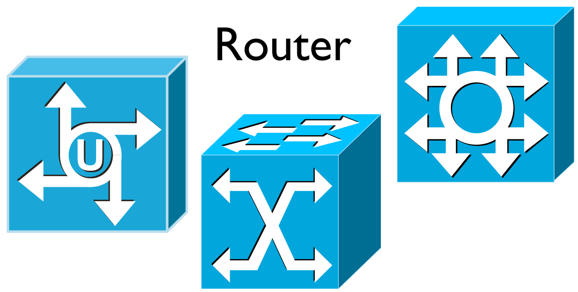


The Generic Model: Constituents

Topology



Router



Scheduler

Injection

Formal model of network architectures

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

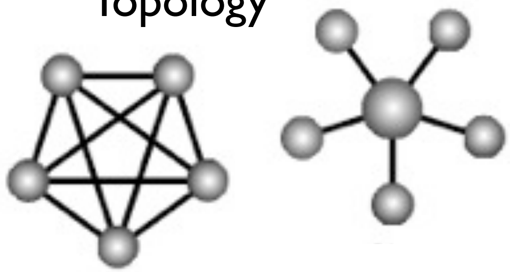
Routes from current to destination

inject messages

The Generic Model: Proof obligations (or constraints)

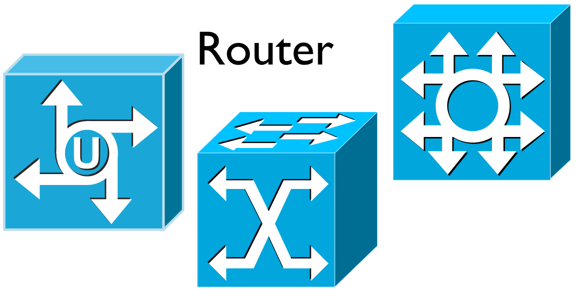
Local constraints sufficient to prove *global* generic theorems.

Topology



"Sinks have no outgoing edges"

Router



Type: "R:PxP→P"

Acyclic dependency graph

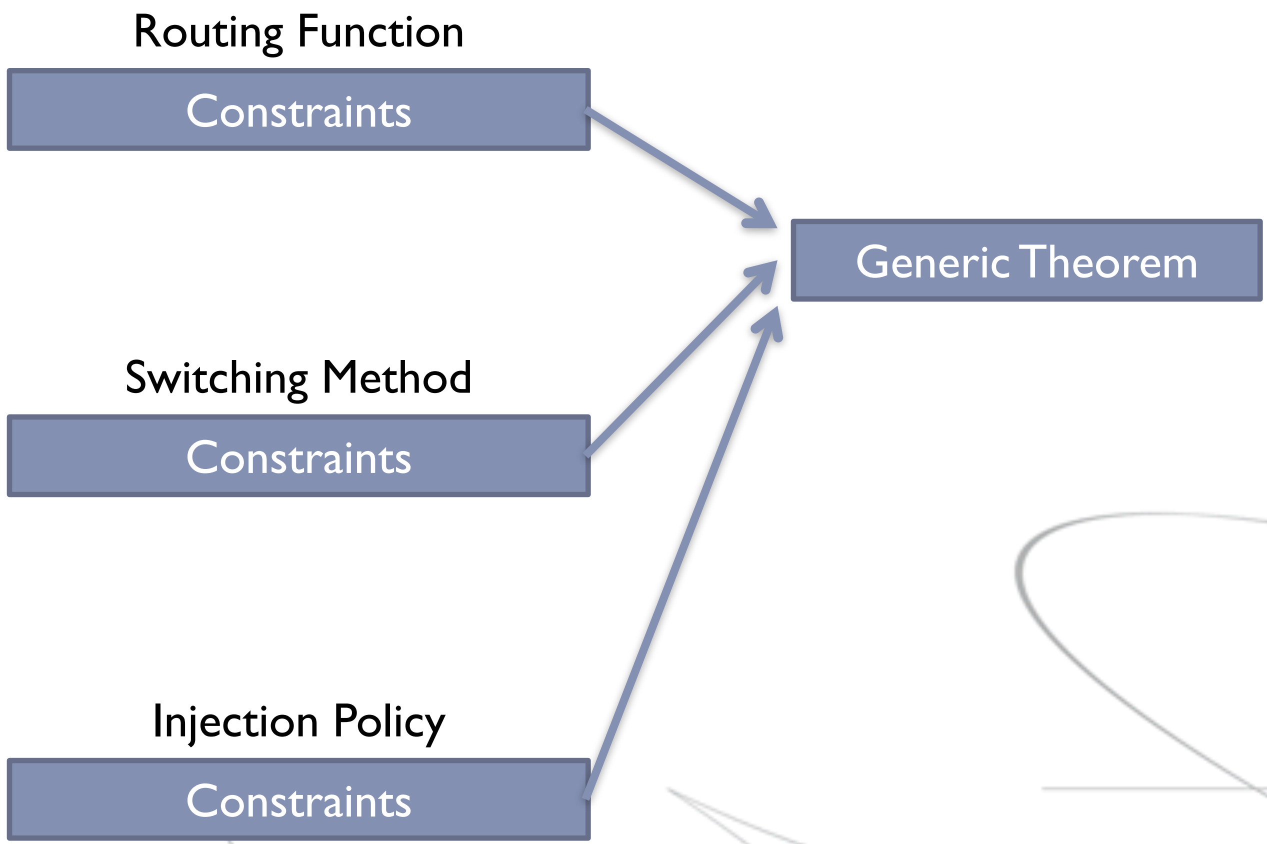
Scheduler

"A message moves, unless it is stuck"

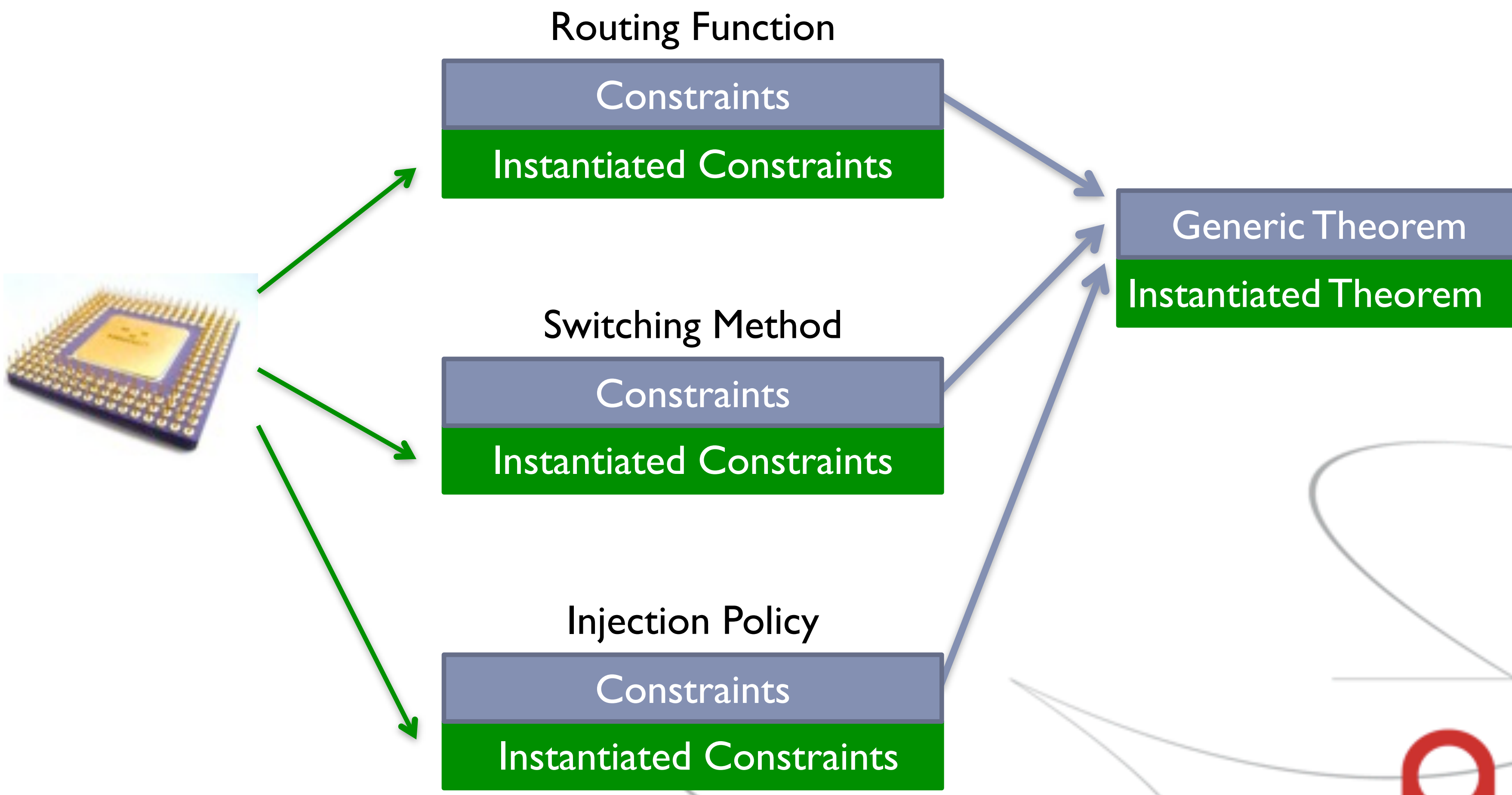
Inject if network is empty

Injection

The Generic Model: Generic theorems



The Generic Model: Generic theorems



GeNoC Theorem (1): Functional correctness

- Functional correctness
  - *if a message reaches a destination, it reaches its expected destination without modification of its content*
  - Note: trivially holds if no message reach a destination

- Main proof obligations on routing
  - last of route from s to d is d
  - route computation terminates
  - length of routes (opt)

- Proof obligation on scheduling
  - mutual exclusion of scheduled and delayed messages
  - union of scheduled and delay contains exactly all messages (no spontaneous generation of new messages)

GeNoC Theorem (2): Deadlock freedom

- A network is deadlock-free iff
  - *there is no reachable deadlocked configuration*
  - deadlocked configuration = configuration where all messages are stuck
- Main proof obligations on routing
  - acyclic resource dependency graph (deterministic)
  - escape for all cycles (adaptive)
  - consistency between dependency graph and routing function
- Main proof obligation on scheduling
  - next-hop based scheduling policy

GeNoC Theorem (3): Evacuation

GeNoC (σ) =

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

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

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

- Evacuation theorem
  - *all messages eventually leave the network*

- Main proof obligations on function GeNoC
  - function GeNoC terminates
  - generic termination measure

- Main proof obligation on scheduling
  - decreases measure if no deadlock

- Main proof obligation on routing
  - deadlock-free routing

- Main proof obligation on injection
  - decreases measure if when network is empty

Overview of applications of GeNoC

Topology

2D-mesh

Octagon

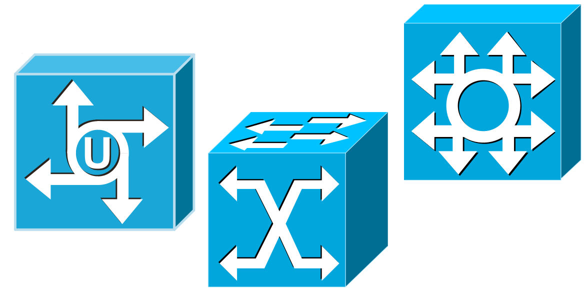
Router

xy routing

double Y routing

Octagon routing

Hot potato routing



“R:PxP→P”

Injection

time dependent

immediate

Scheduler

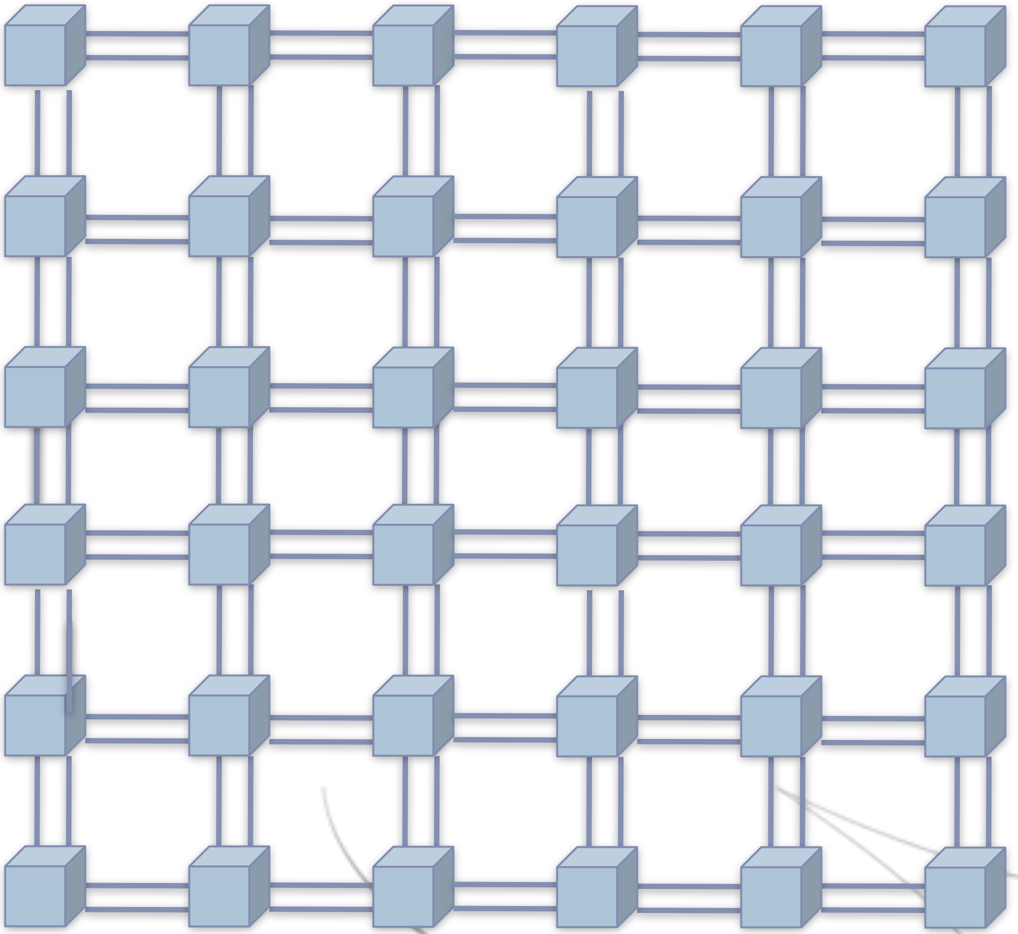
circuit switching

packet switching

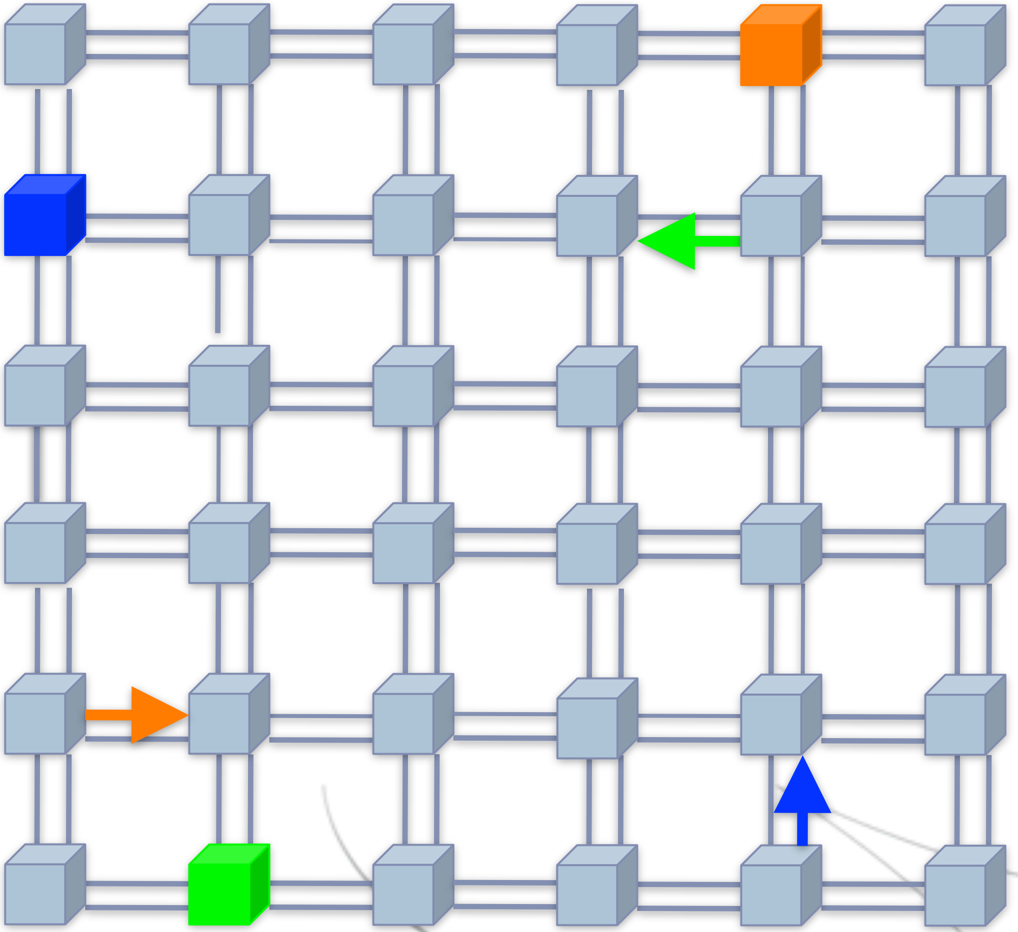
wormhole switching

bus arbitration

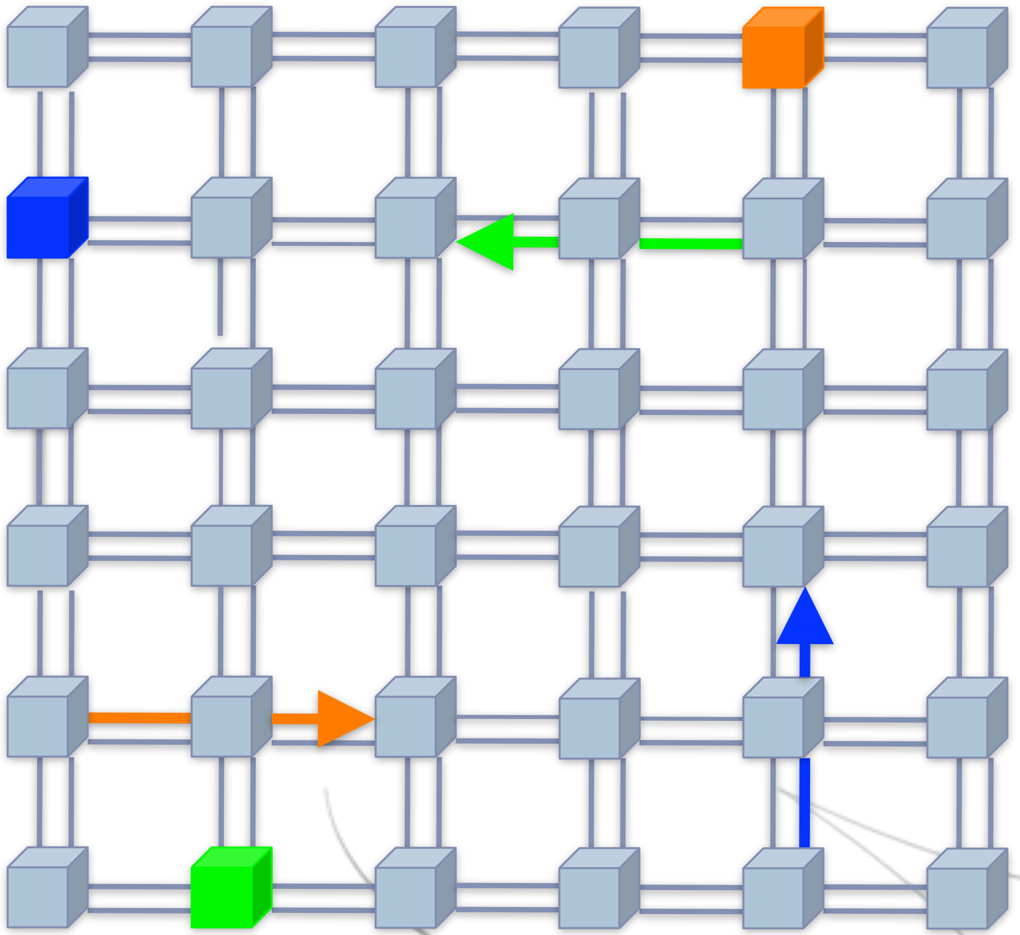
Deadlock



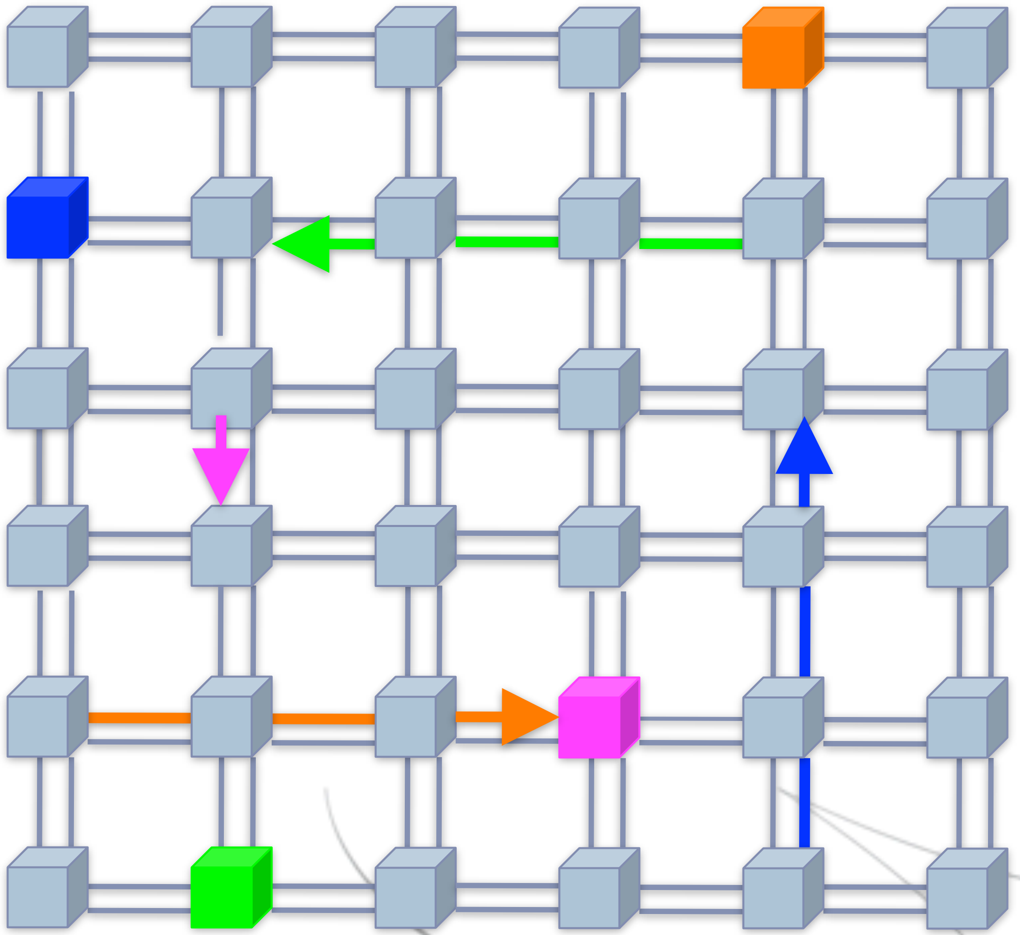
Deadlock



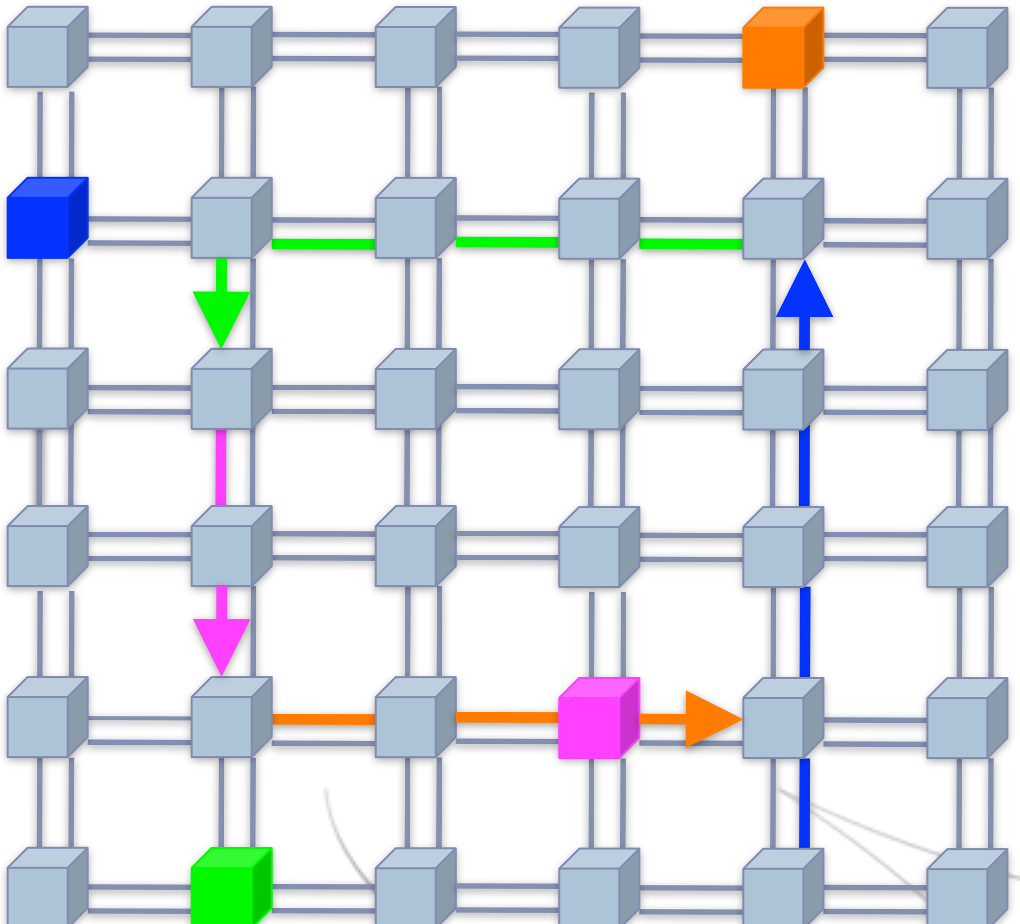
Deadlock



Deadlock

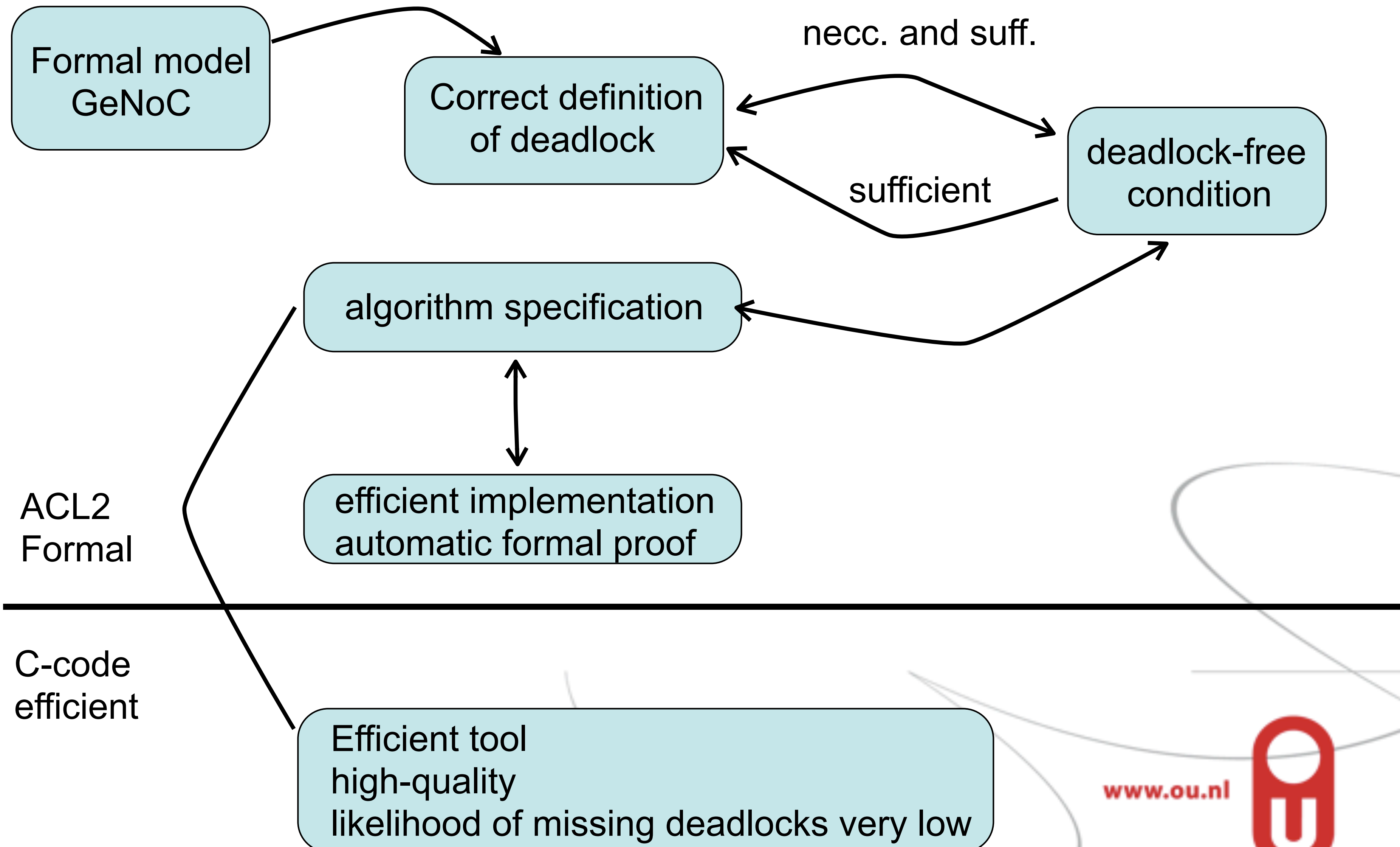


Deadlock

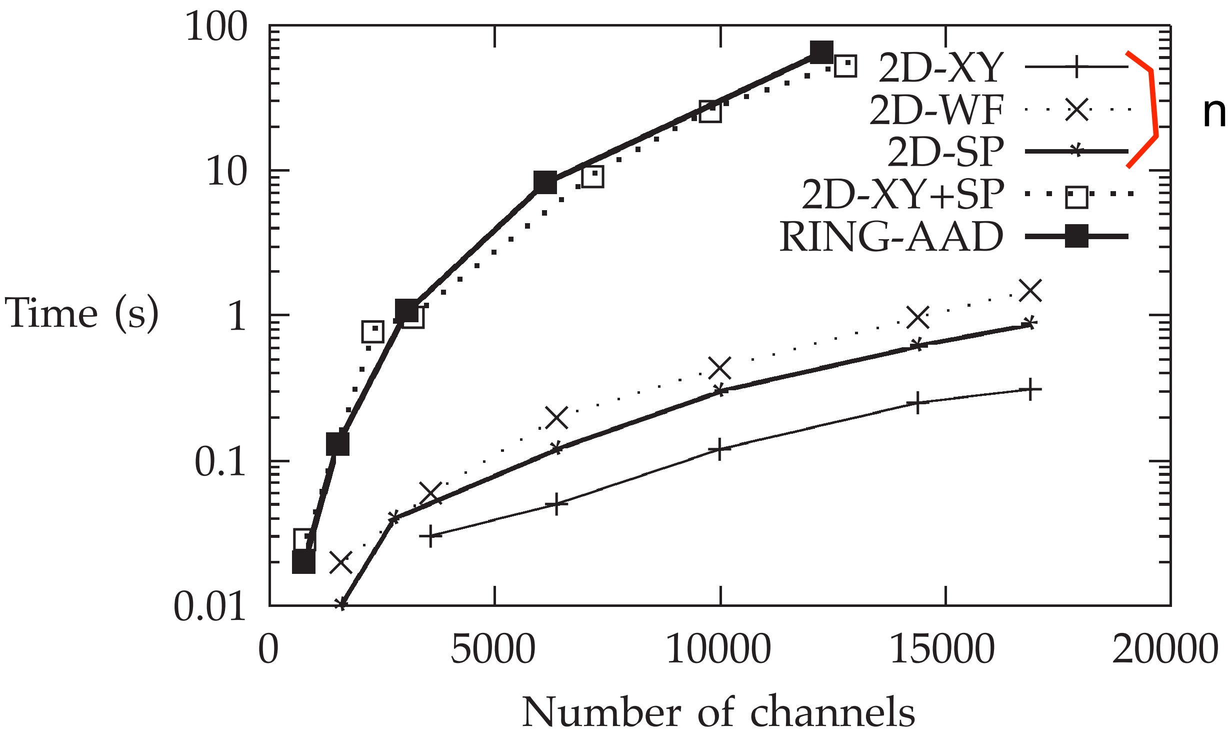


Deadlock is an emerging property

Deadlock verification - The big picture



Automatically checking sufficient condition (C-code)



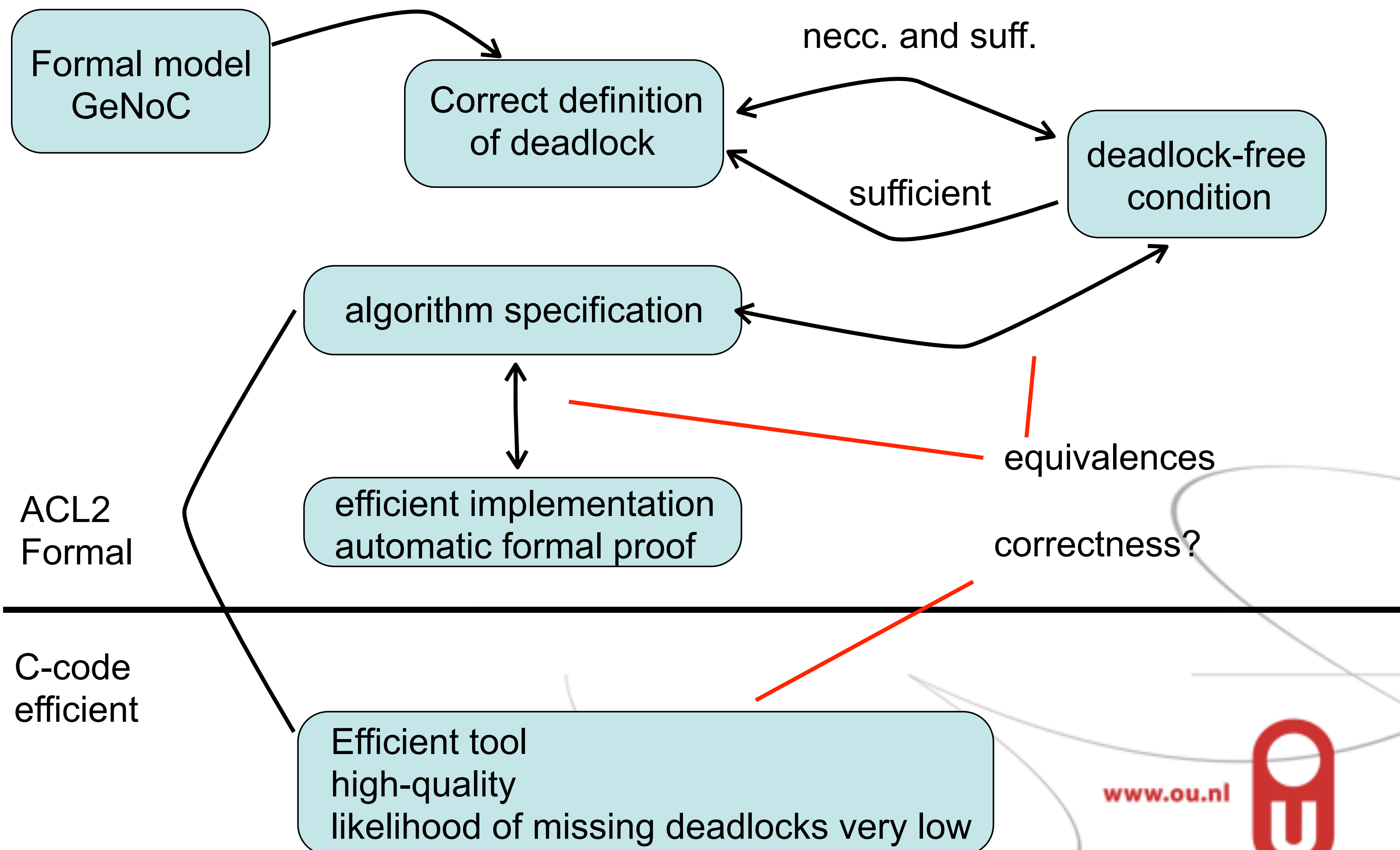
no cyclic dependencies

Condition as strong as Duato's one and its variations

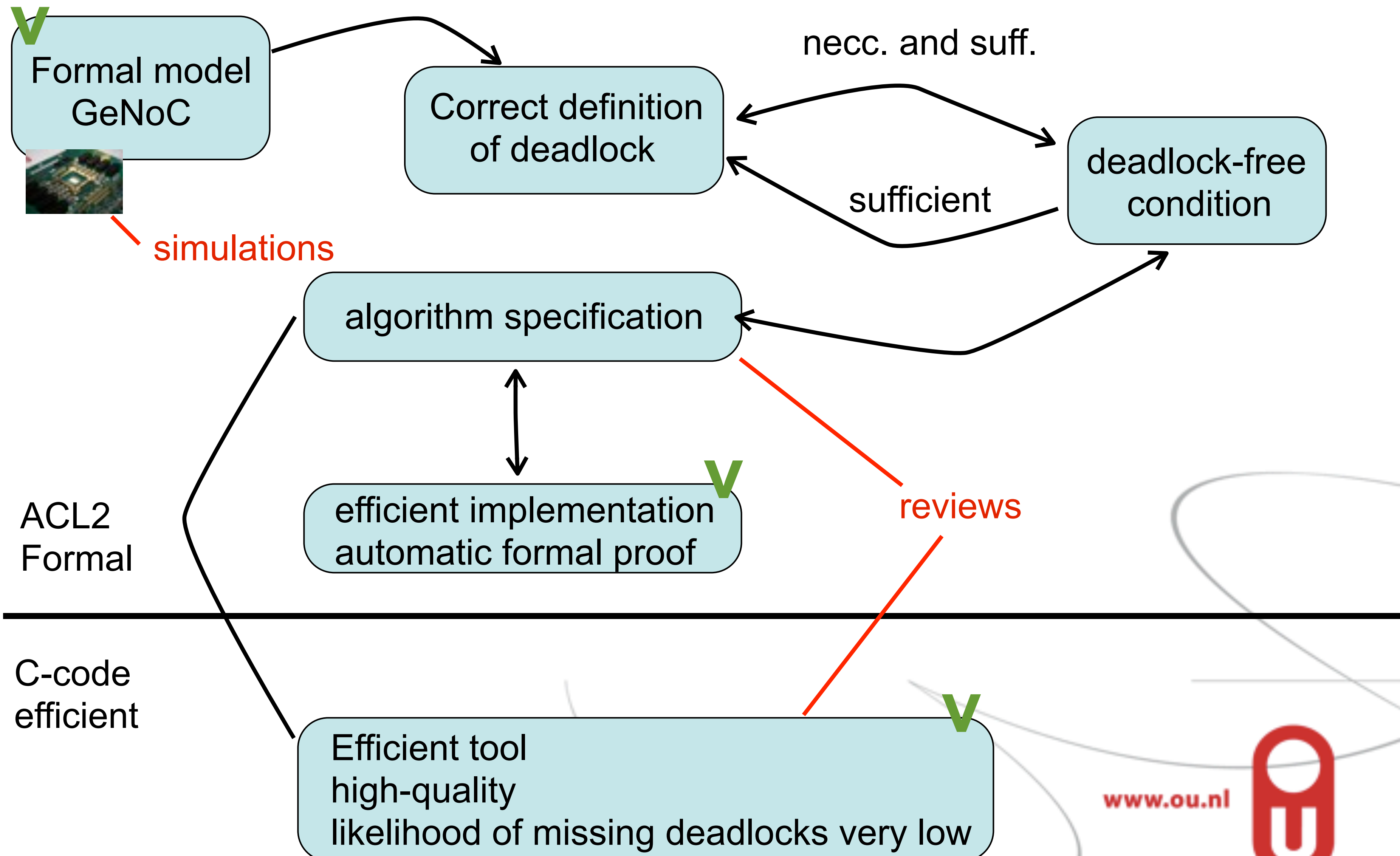
10000s of channels in 100 seconds

Time complexity in O(N^3) where N = number of nodes

Deadlock verification - Unintended functionality



Deadlock verification - Conservative representation



www.ou.nl

Our approach

- Develop formal theory of the domain (e.g. NoCs)
  - identify components and their interactions
- Prove general theorems in this theory
  - what are the interesting global properties (no deadlock)
- Extract proof obligations on the components
  - what is important to know about each component
- Develop verified algorithms checking the POs
- Implement these algorithms
  - within the logic of an ITP (e.g. ACL2)
  - every run of the algorithm is a ***formal*** proof
  - in standard languages (e.g. C)
  - high-quality "bug hunter"

Reflection and side effects of formal efforts

- Found a (small) flaw in seminal paper of Duato
  - work was a breakthrough
  - paper 250 cites on GS
  - flaw in other paper with 630 cites and book with 1450 cites
  - flaw in many papers inspired by Duato's work

- Correcting the flaw makes problem co-NP-complete
  - previous work claimed polynomial solution
  - made same mistake as Duato

- Theorem proving ensure correctness of algorithms
  - lots of corner cases
  - hard to debug when 1 single incorrect trace

- *In-depth understanding* of the issue

Conclusion

- Verified certifiers
- ITP is used to develop general theories and verified algorithms
- Verified algorithms implemented as high-quality "bug hunters"
  - likelihood of bugs after running the certifier
  - formal proof when running verified code (ACL2)
- Domain specific
  - static (on-chip) interconnection networks
- Very efficient
  - proven correct (sound)
  - linear or polynomial when possible
  - boundary to co-NP-complete well-defined