Formal Verification of Communications in Networks on Chips

Sebastiaan J.C. Joosten
Julien Schmaltz
Freek Verbeek
Bernard van Gastel

Open University of the Netherlands, Heerlen
Radboud University Nijmegen

TU/e Technische Universiteit Eindhoven University of Technology

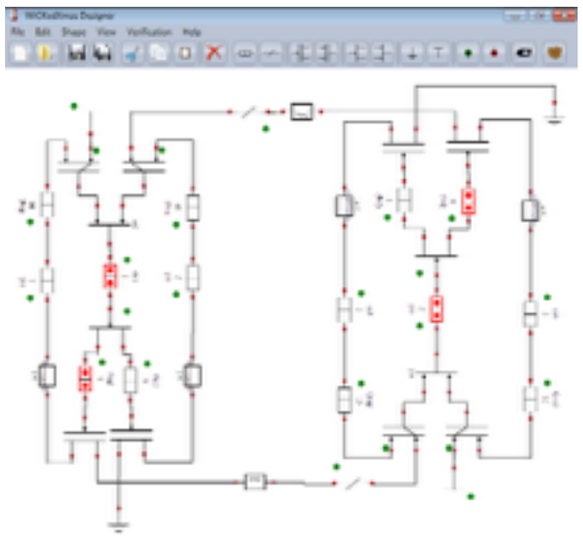
Where innovation starts

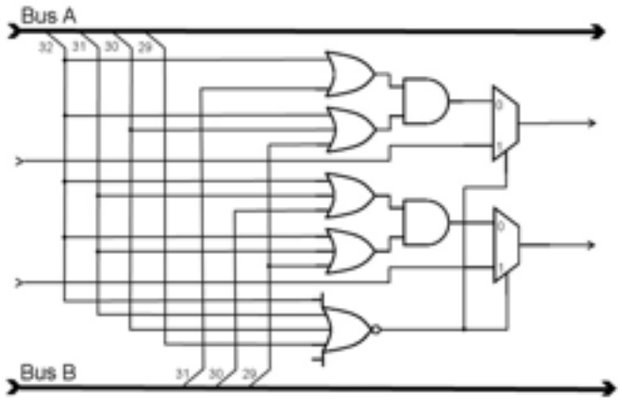
Three Levels of Abstraction

System Level

```
/**************/
/* ROUTING    */
/**************/

/*
Shortest path routing in the Spidergon ring
*/

#include "spidergon_definitions.h"


/*
The routing function. Routes from channel c == (x,p) to processing node n == (dx)
Returns a set of resources. As routing is deterministic, one next hops are returned.
Note that we initialize a list of 2 next hops: one next hop and one enclosing NULL.
*/
ResourceList* inst_routing(const Resource &c, Procnode* d, const Params* dim, ResourceList* frs) {
    ResourceList *hops = new ResourceListN<2>;
    Resource *nexthops = hops->routed;

                        // The coordinates of the destination
                        int dest = d->s;
                        // The coordinates of the processing node at the end of channel c
                        int next = get_end(c).s;

                        // Compute relAd:
                        int relAd = (dest - next + NUM_OF_PROC_NODES) % NUM_OF_PROC_NODES;

                        if (relAd == 0) {
                                        // No next hop
                        }
                        else if (relAd == 1 || relAd == 2)
                                        nexthops[0] = Resource(next, CW);
                        else if (relAd == 6 || relAd == 7)
                                        nexthops[0] = Resource(next, CCW);
                        else
                                        nexthops[0] = Resource(next, ACC);

                        return hops;
}
```

Micro-Architectural Level





Register Transfer Level

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

(Formal) Verification

Testing

Model Checking

Dedicated Algorithms

Theorem Proving

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

NoC Correctness - Productivity

A network is productive if and only if
all pending messages eventually gain access to the network
and eventually reach their expected destination.

- Deadlock freedom
- Livelock freedom
- Functional Correctness

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

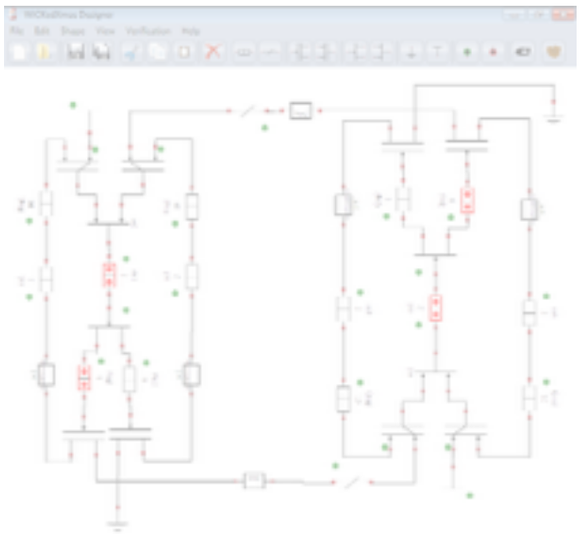
TU/e Technische Universiteit Eindhoven University of Technology

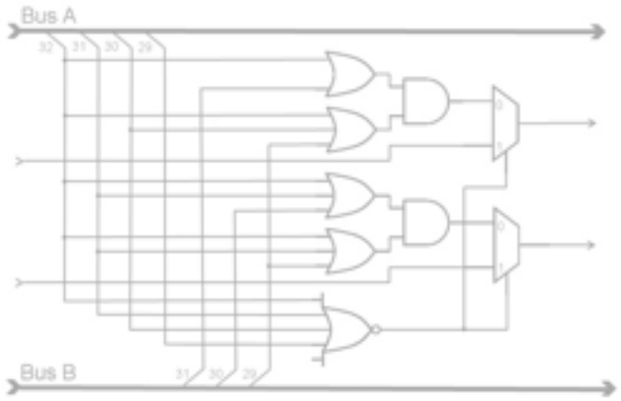
Three Levels of Abstraction

System Level

```
/*************/
/* ROUTING   */
/*************/

/*
Shortest path routing in the Spidergon ring
*/

#include "spidergon_definitions.h"


/*
The routing function. Routes from channel c == (x,p) to processing node n == (dx)
Returns a set of resources. As routing is deterministic, one next hops are returned.
Note that we initialize a list of 2 next hops: one next hop and one enclosing NULL.
*/
ResourceList* inst_routing(const Resource &c, Procnode* d, const Params* dim, ResourceList* frs) {
    ResourceList *hops = new ResourceListN<2>;
    Resource *nexthops = hops->routed;

                    // The coordinates of the destination
                    int dest = d->s;
                    // The coordinates of the processing node at the end of channel c
                    int next = get_end(c).s;

                    // Compute relAd:
                    int relAd = (dest - next + NUM_OF_PROC_NODES) % NUM_OF_PROC_NODES;

                    if (relAd == 0) {
                                    // No next hop
                    }
                    else if (relAd == 1 || relAd == 2)
                                    nexthops[0] = Resource(next, CW);
                    else if (relAd == 6 || relAd == 7)
                                    nexthops[0] = Resource(next, CCW);
                    else
                                    nexthops[0] = Resource(next, ACC);

                    return hops;
}
```

Micro-Architectural Level



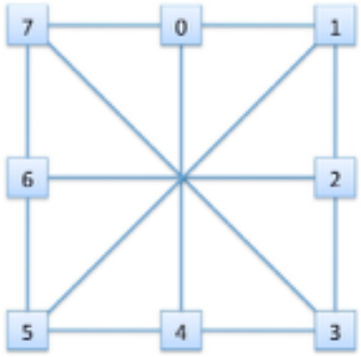


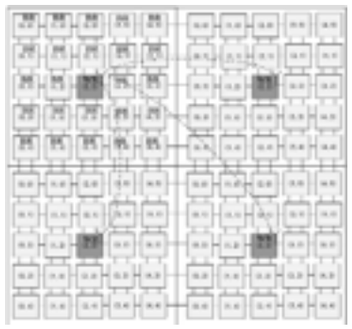
Register Transfer Level

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

System Level - Deadlock-free routing





topology.top

routing.c

Fast

Proven correct

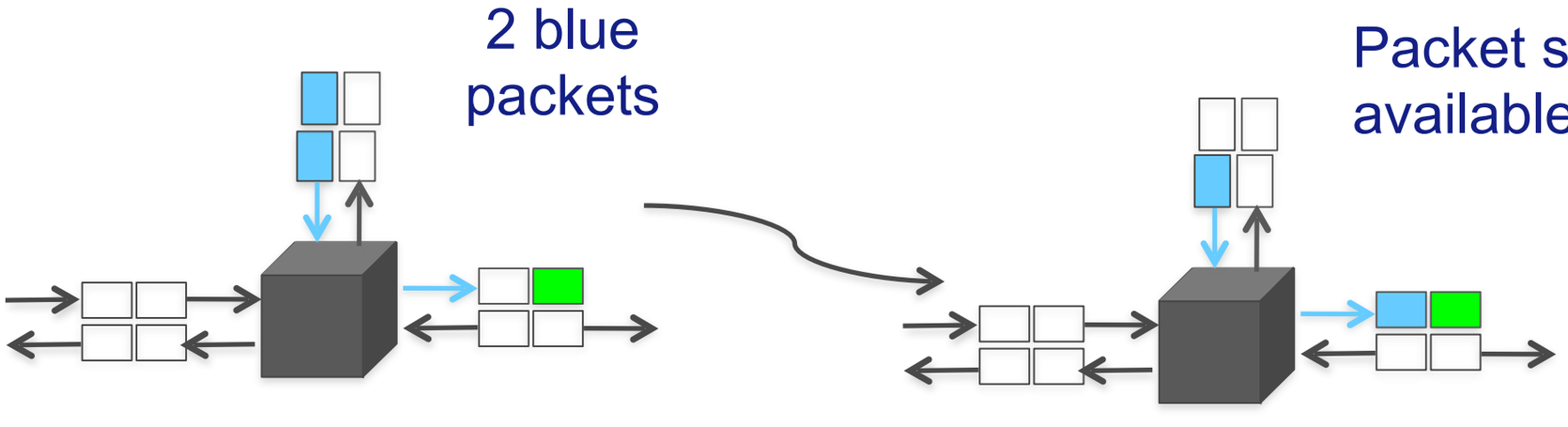
Minimal deadlock configuration

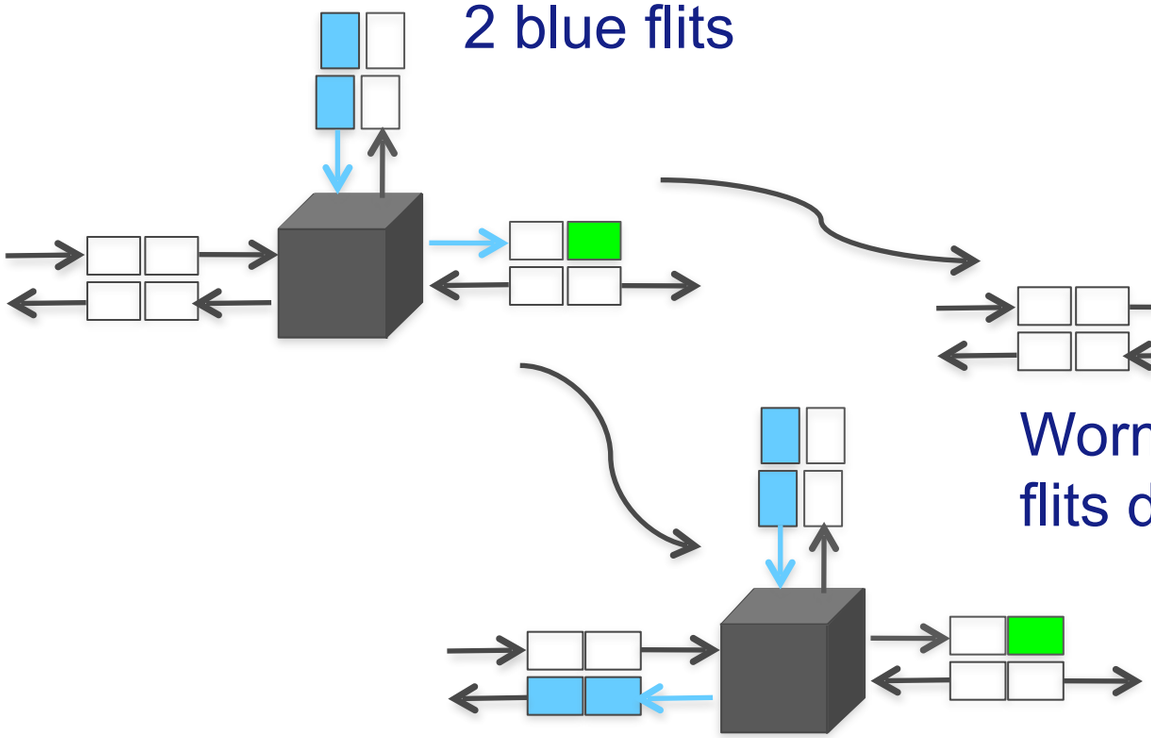
No deadlock!

Technische Universiteit Eindhoven University of Technology

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

Two semantics - Wormhole & Packet

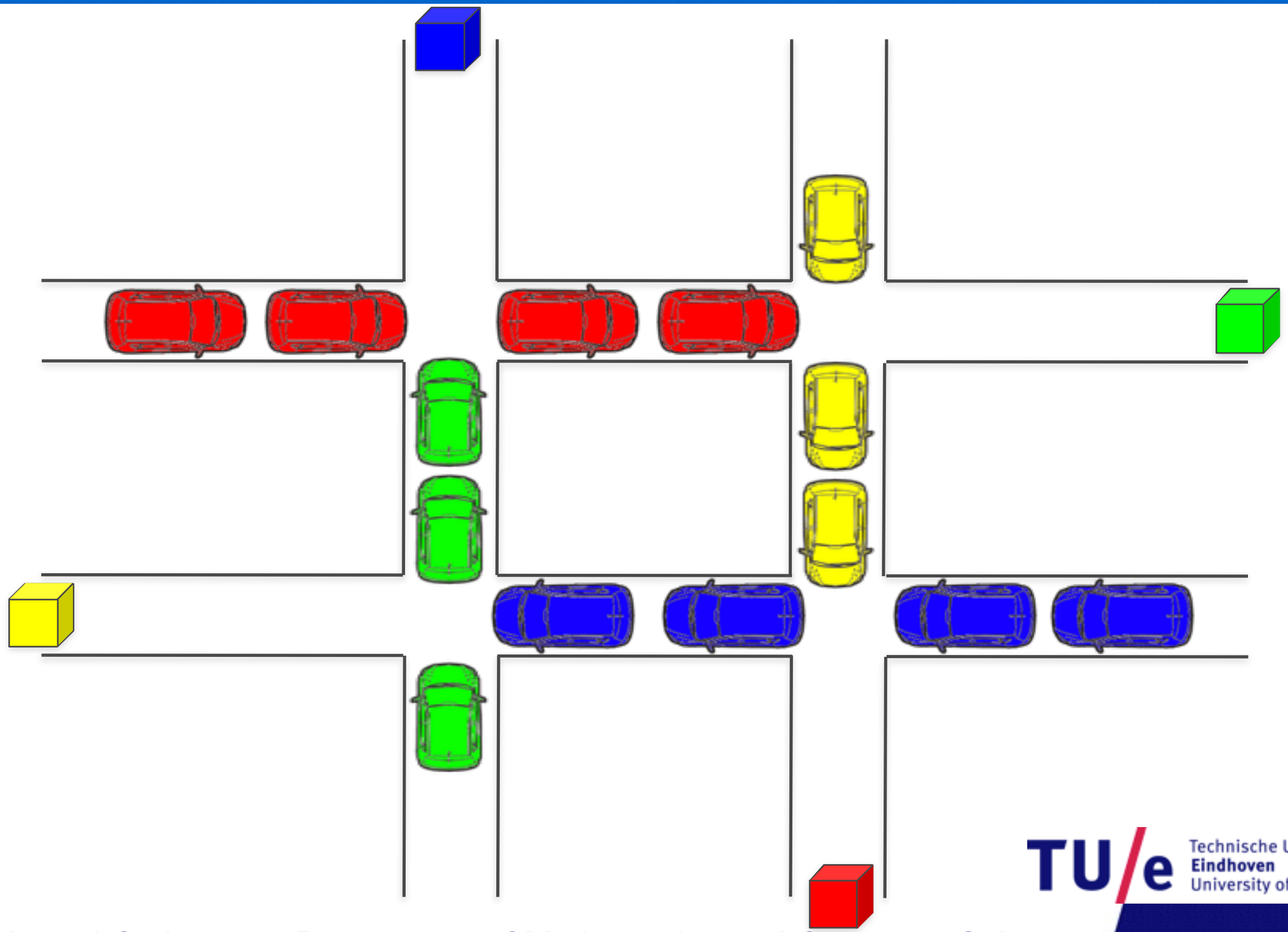




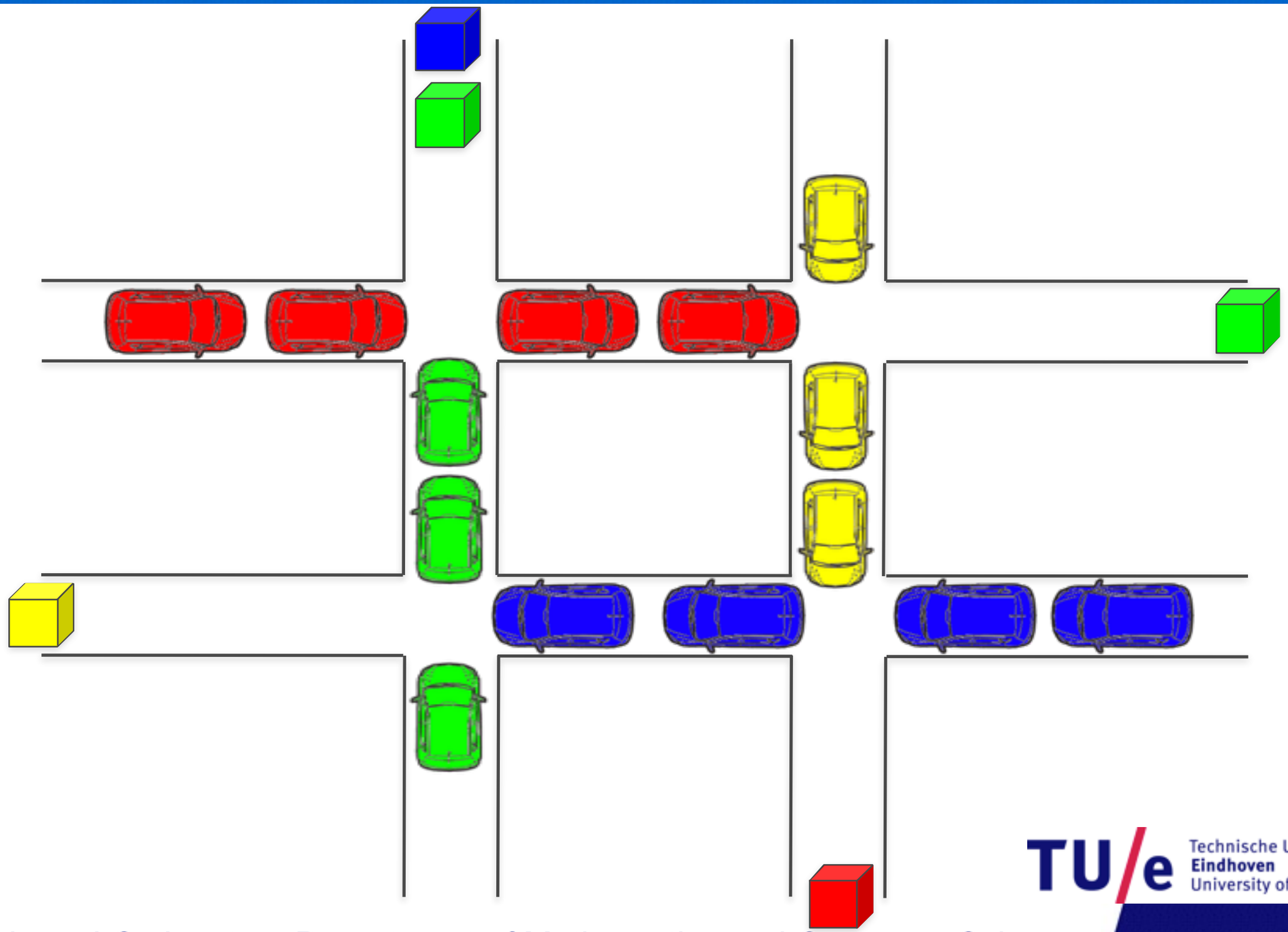
Wormhole switching:
flits don't have to belong to the same message

Wormhole switching:
in a channel all flits belong to the same message

Deadlocks - Circular wait

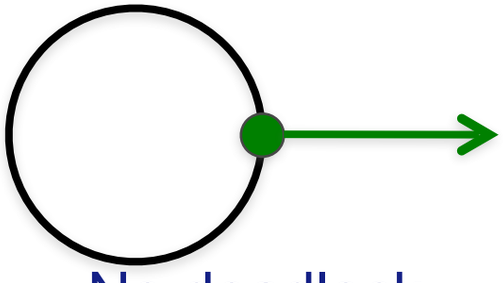


An escape

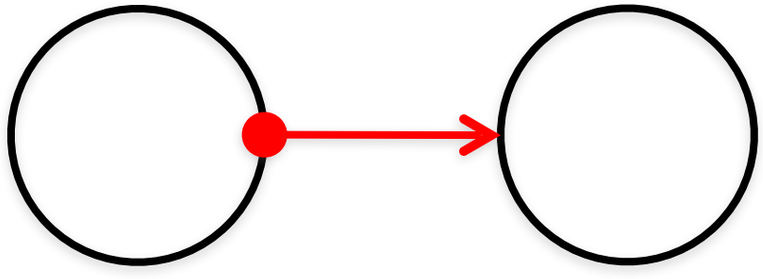


Necessary and sufficient condition (packet networks)

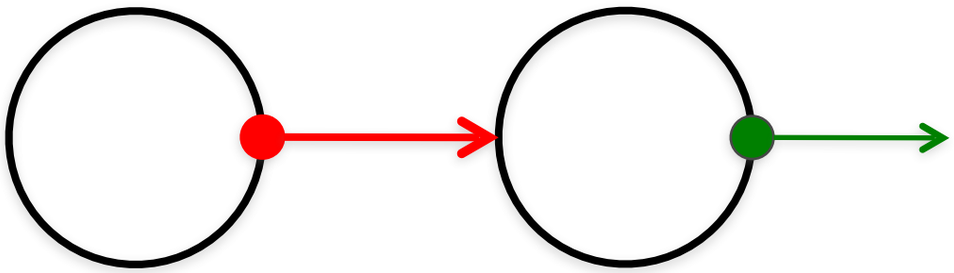
There exists a deadlock iff there all sets of cycles have an escape.



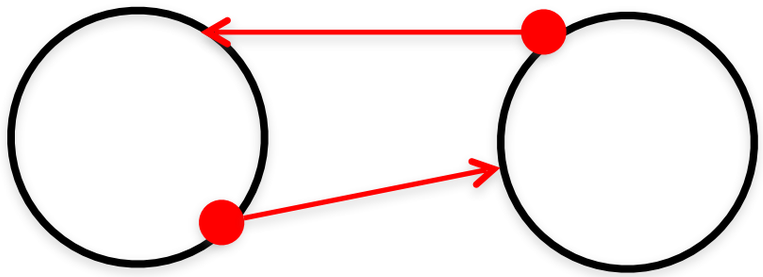
No deadlock



Deadlock



No deadlock



Deadlock

Technische Universiteit Eindhoven University of Technology

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

Wormhole networks

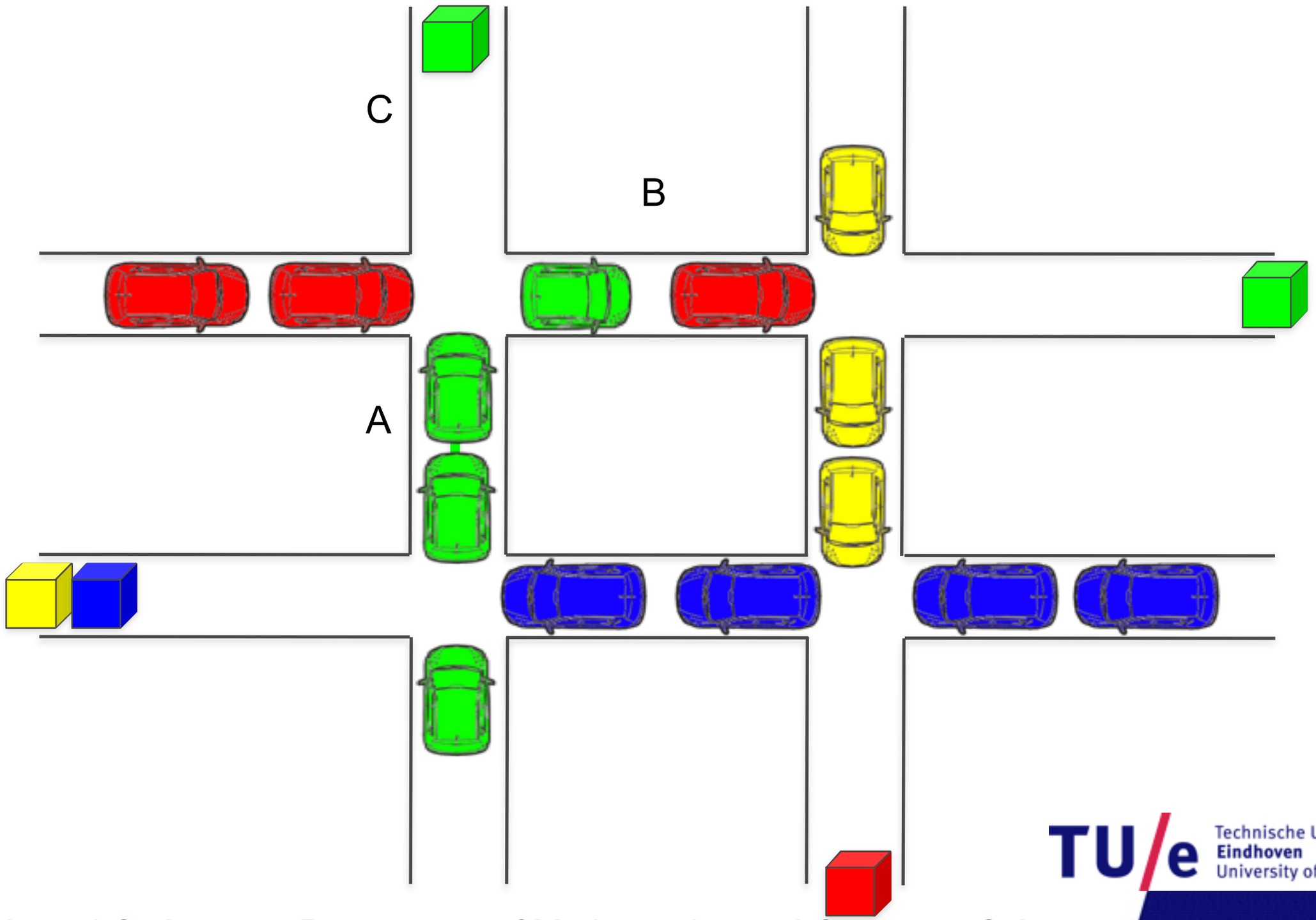
Many more subtleties…

co-NP-complete

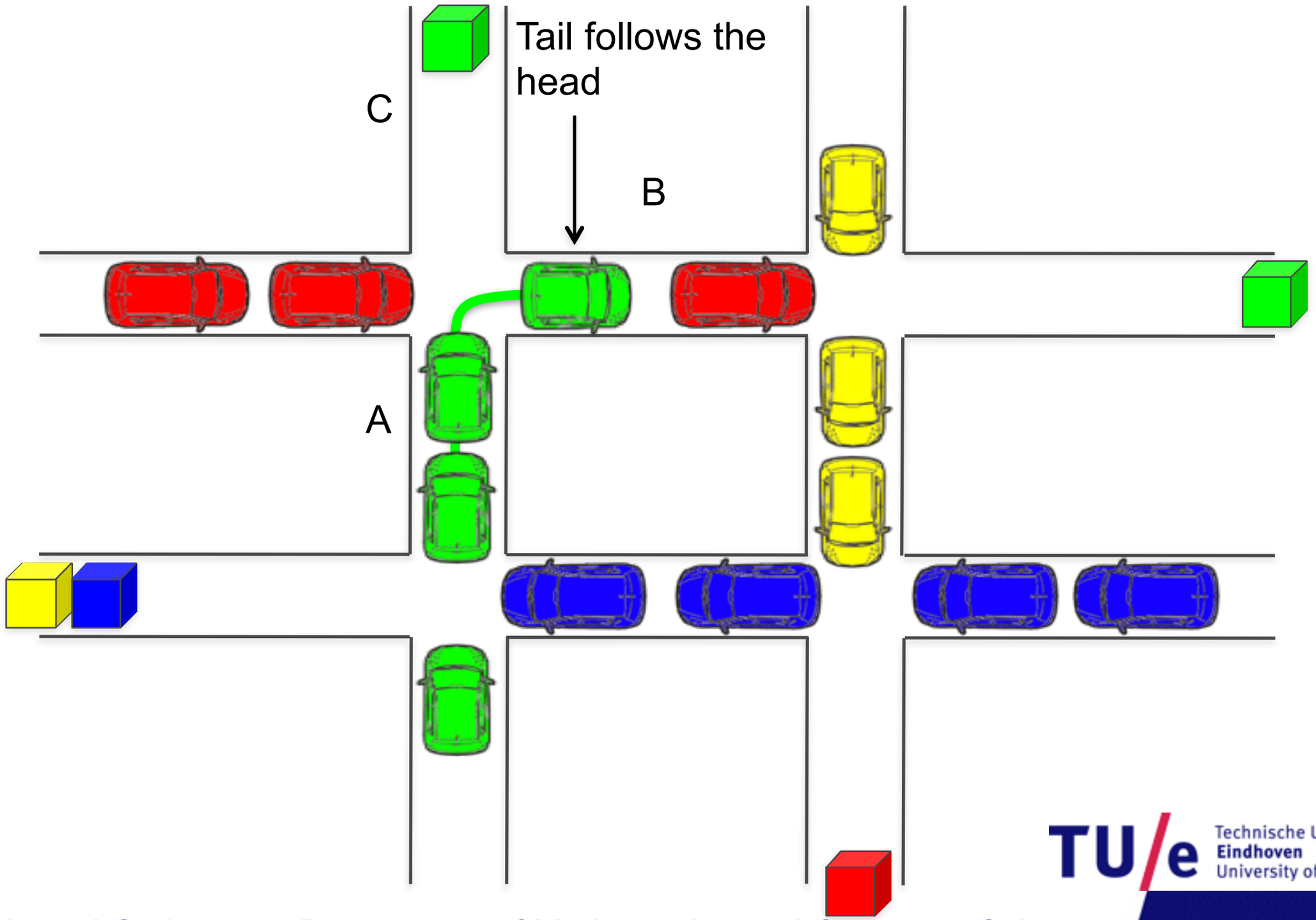
Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

Technische Universiteit Eindhoven University of Technology

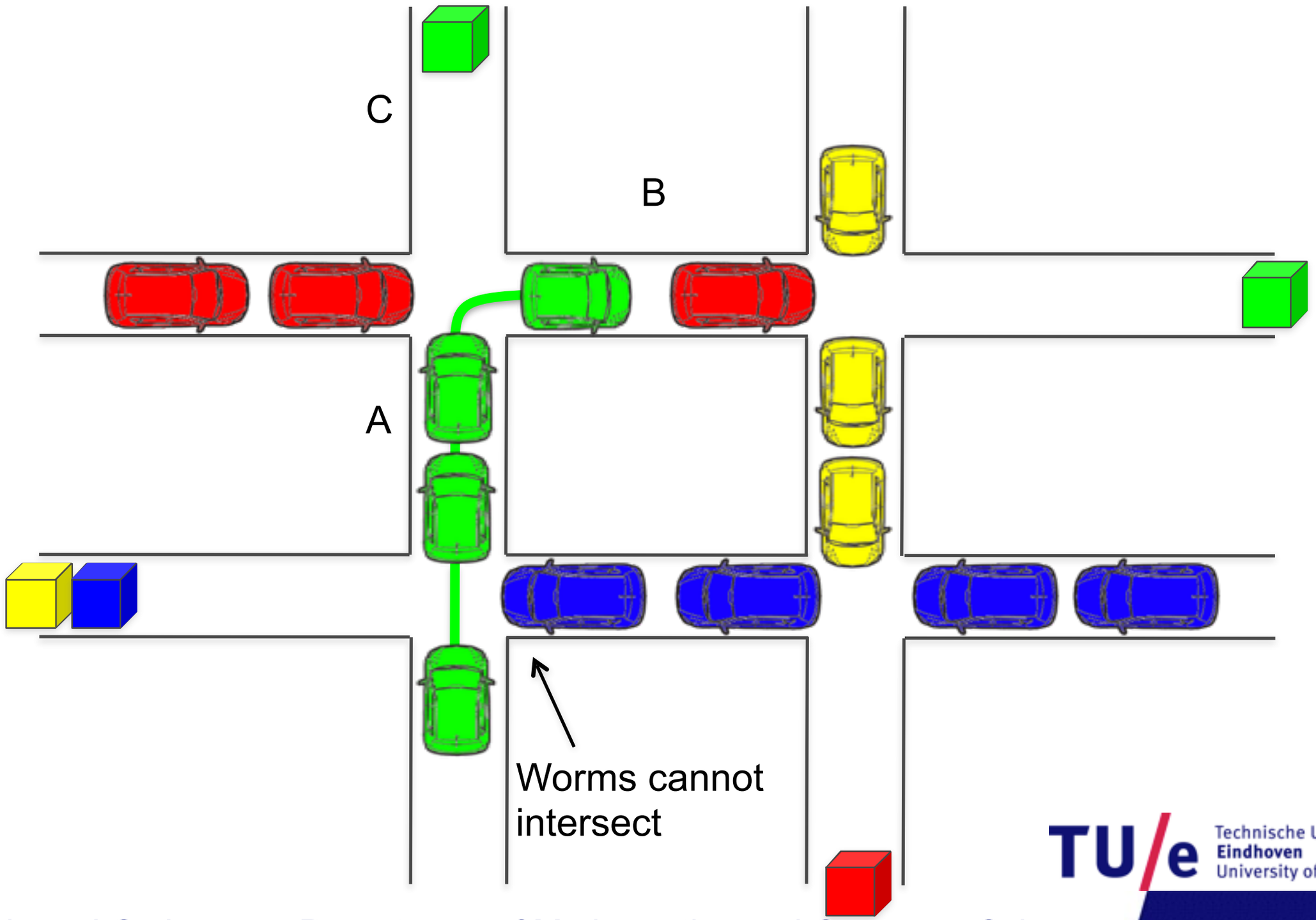
Two escapes



One escape

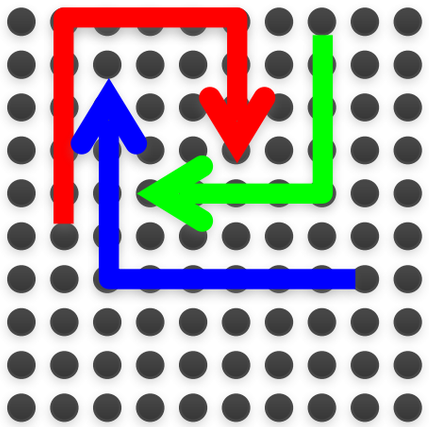


No escape



Necessary and sufficient condition

There exists a deadlock iff there exists a set of routing paths such that:



The set is non-empty	(A)
The set is pairwise disjoint	(B)
The set has no escape	(C)

Tool - DCI2

DCI2 Website
http://www.cs.ru.nl/~freekver/DCI2/index.html

DCI2: Detecting routing deadlocks

For any question or remark please email f.verbeek -- at -- cs.ru.nl.

DCI2 stands for Deadlock Checker In Designs of Communication Interconnects

Step 1: installation

Download

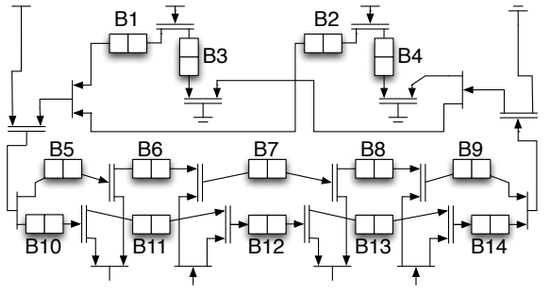
Download the algorithms here.
Download the IBM CPLEX solver (see here for instructions). It is free of charge for academic purposes.
Note that some optional features also require Zchaff. If you are not using these options, it is not necessary to download it. Otherwise, please ask me for further instructions.

Unpack algorithm and install CPLEX

- Install CPLEX according to the installation instructions.
- Unpack the DCI2.tar.gz file in the directory where the algorithm will be installed (from now on referred to with ".").
- Go to "." and make a sub directory called "./ilcplex".
- Copy the ".h" files that came with CPLEX (they are located in the "icplex/include/ilcplex" sub directory of your CPLEX installation) to your new "./ilcplex" sub directory.
- Make a subdirectory -- either "./ilcplex/mac" or "./ilcplex/x86_64" -- depending on your OS.
- Copy the files "libcplex.a" and "libilocplex.a" located in your CPLEX installation (they are located in a directory similar to "cplex/lib/x86-64_darwin9_gcc4.0/static_pic") to this location.

How will you verify your NoC?

System level



Properties

- Deadlock freedom
- Livelock freedom
- No packet loss
- Correct destinations
- Correct payload

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

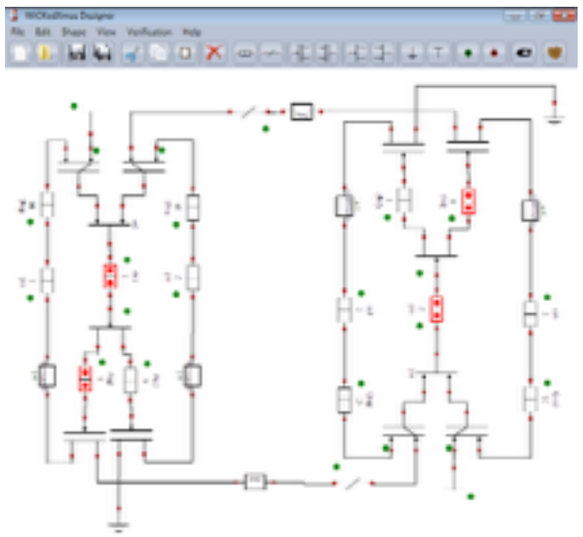
TU/e Technische Universiteit Eindhoven University of Technology

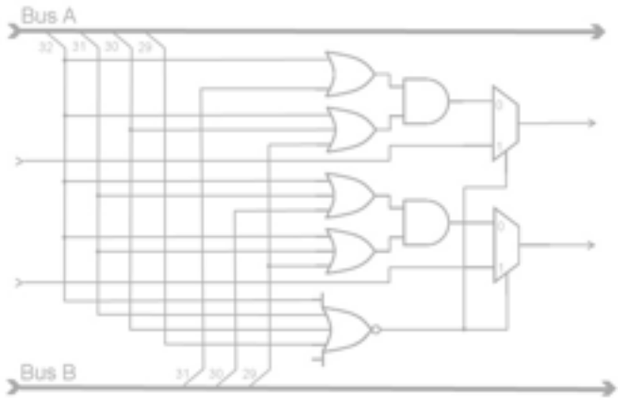
Three Levels of Abstraction

System Level

```
/*************/
/* ROUTING   */
/*************/

/*
Shortest path routing in the Spidergon ring
*/

#include "spidergon_definitions.h"


/*
The routing function. Routes from channel c == (x,p) to processing node n == (dx)
Returns a set of resources. As routing is deterministic, one next hops are returned.
Note that we initialize a list of 2 next hops: one next hop and one enclosing NULL.
*/
ResourceList* inst_routing(const Resource &c, Procnode* d, const Params* dim, ResourceList* frs) {
    ResourceList *hops = new ResourceListN<2>;
    Resource *nexthops = hops->routed;

                // The coordinates of the destination
                int dest = d->s;
                // The coordinates of the processing node at the end of channel c
                int next = get_end(c).s;

                // Compute relAd:
                int relAd = (dest - next + NUM_OF_PROC_NODES) % NUM_OF_PROC_NODES;

                if (relAd == 0) {
                                // No next hop
                }
                else if (relAd == 1 || relAd == 2)
                                nexthops[0] = Resource(next, CW);
                else if (relAd == 6 || relAd == 7)
                                nexthops[0] = Resource(next, CCW);
                else
                                nexthops[0] = Resource(next, ACC);

                return hops;
}
```

Micro-Architectural Level





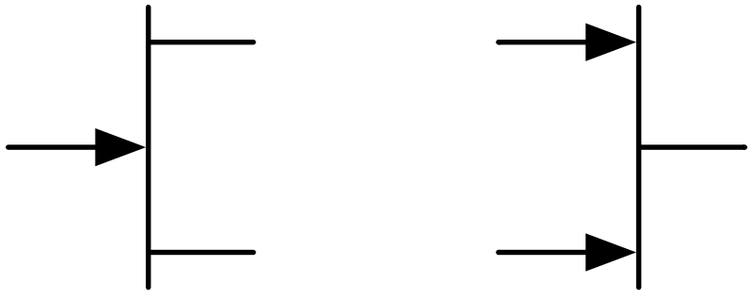
Register Transfer Level

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

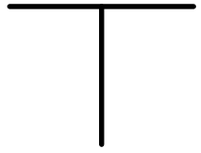
What is xMAS?

- Packets in queues. Other components determine routing.

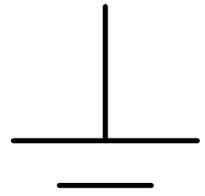


switch

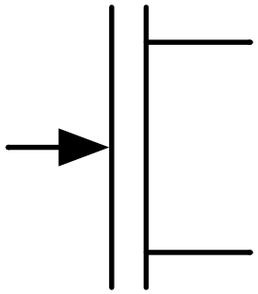
merge



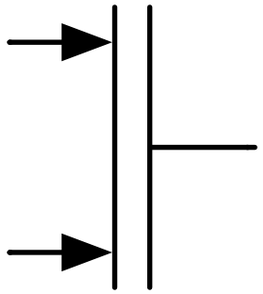
source



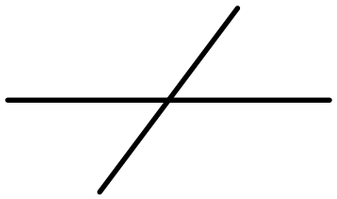
sink



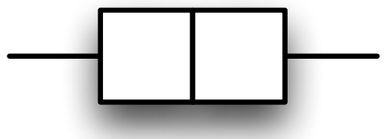
fork



join



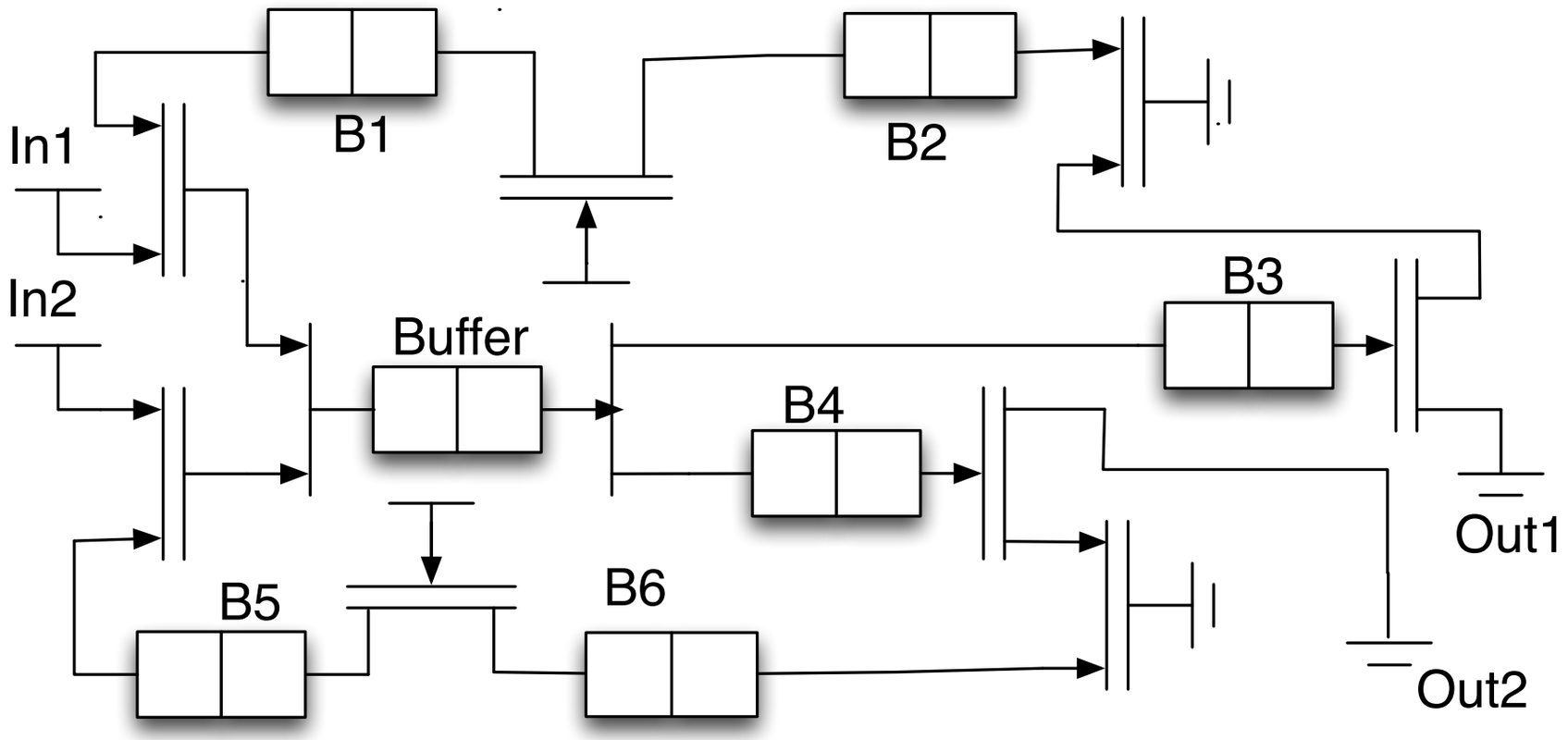
function



queue

What is xMAS?

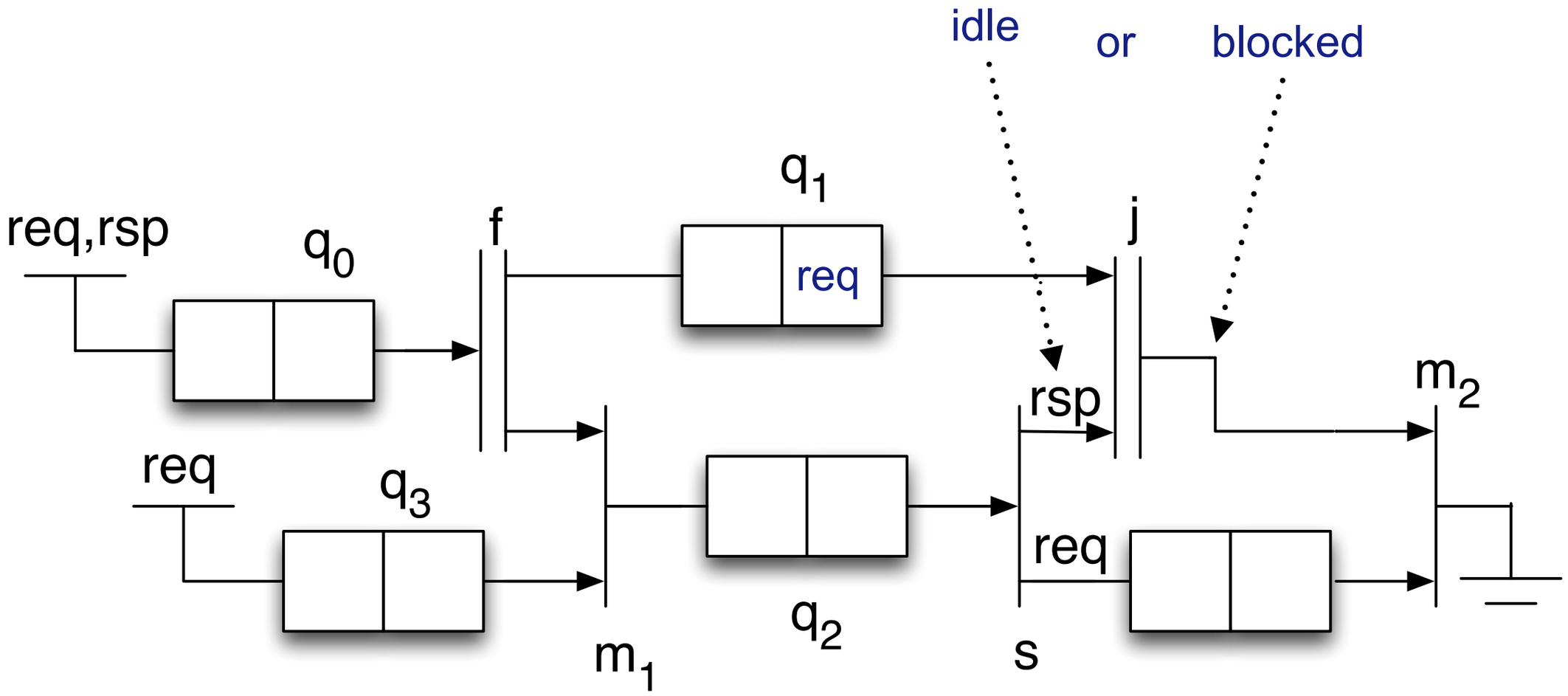
- Virtual channel using xMAS.



- Arbitration at: merge, switch, source, sink
- Data-functions at: join, function, source

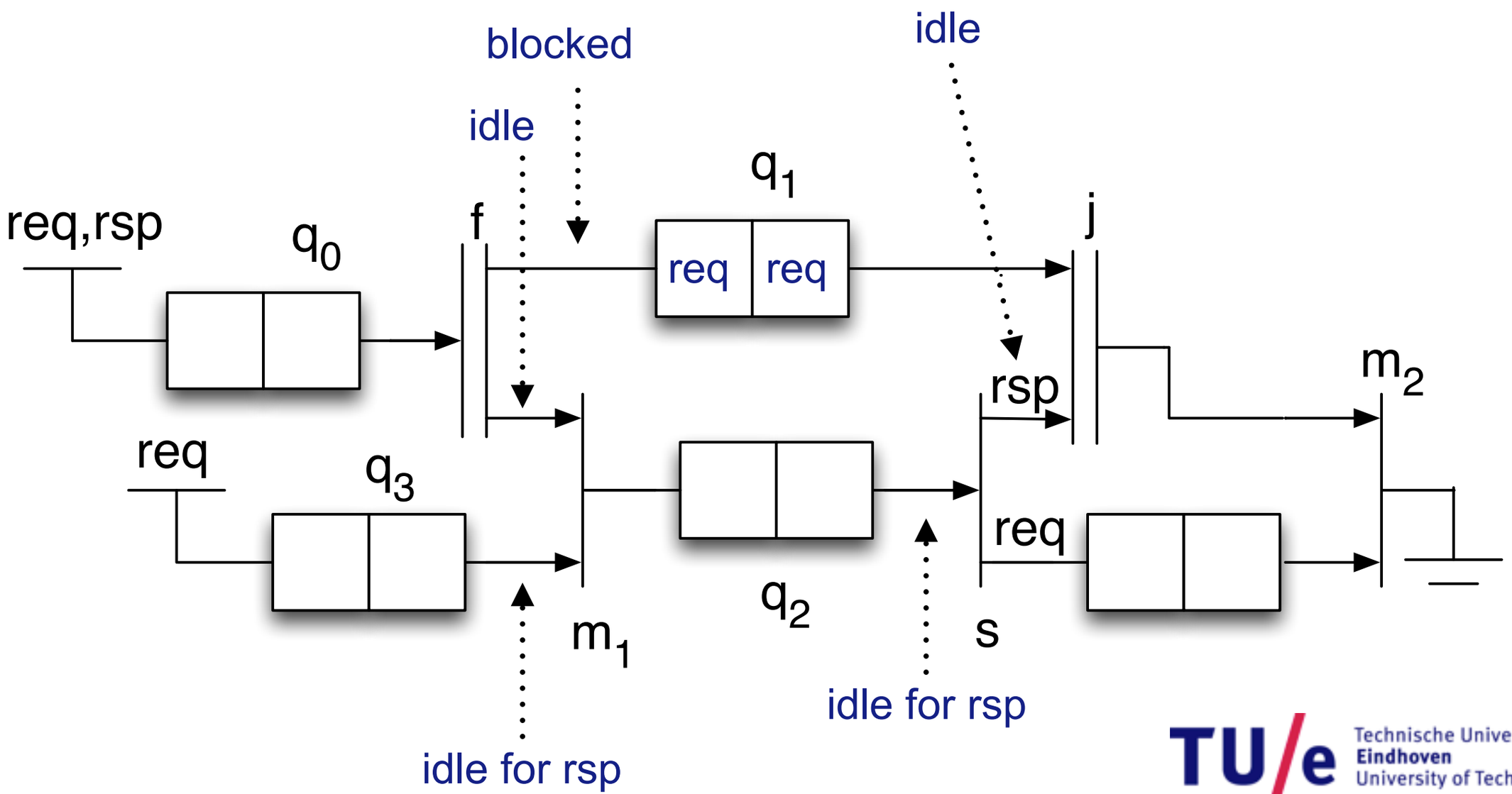
Liveness for xMAS

- Can $q_{1\text{-out}}$ be blocked for a request?



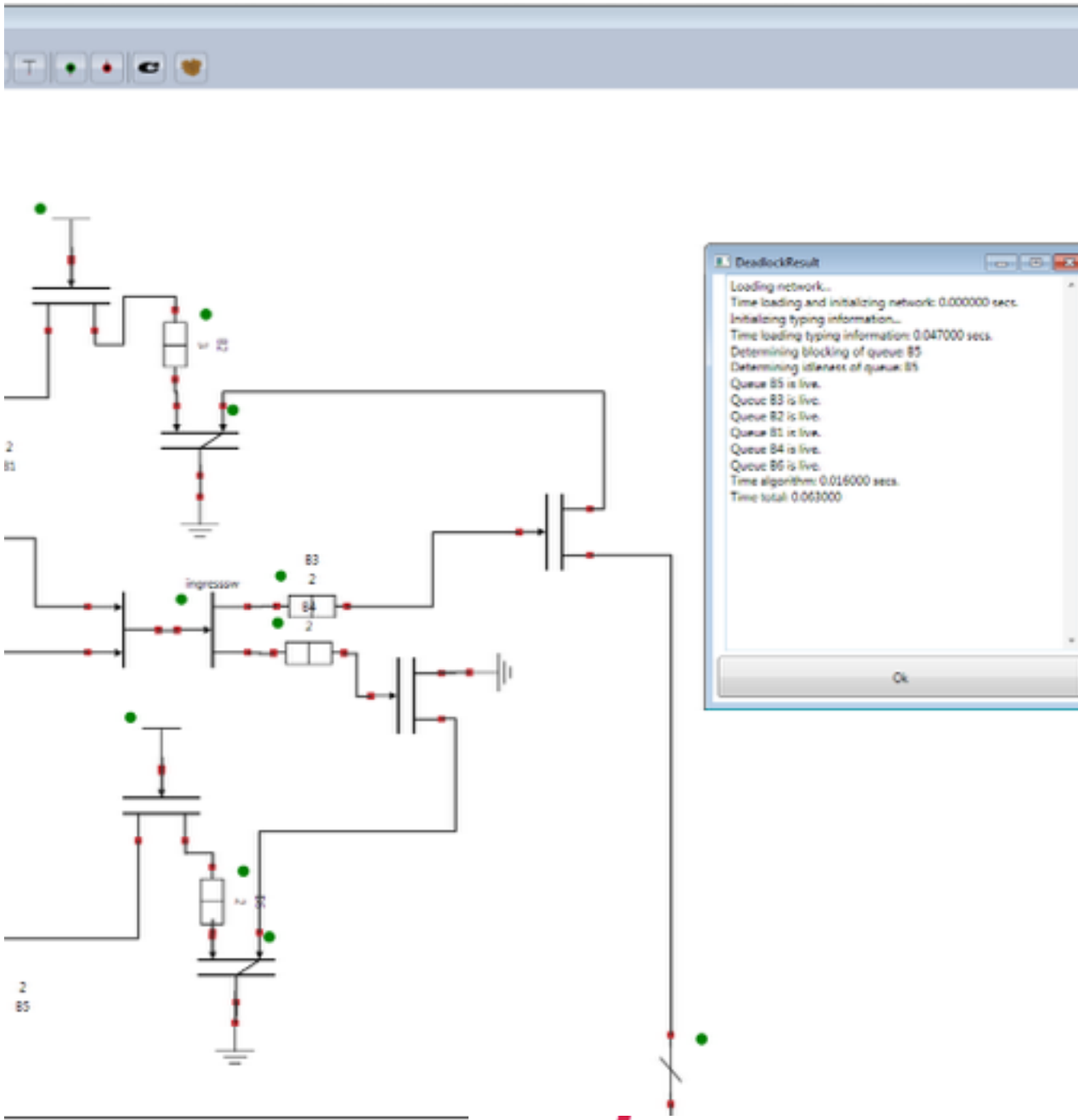
Liveness for xMAS

- $q_{1\text{-out}}$ is blocked for a request!
  The network has a local deadlock!



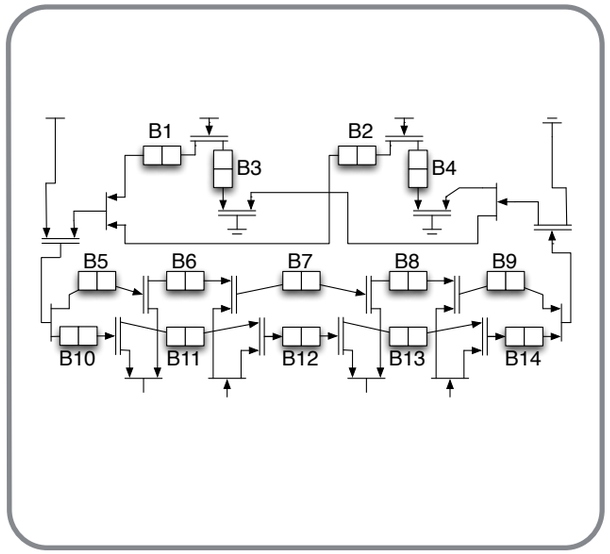
Packet type inferences for xMAS

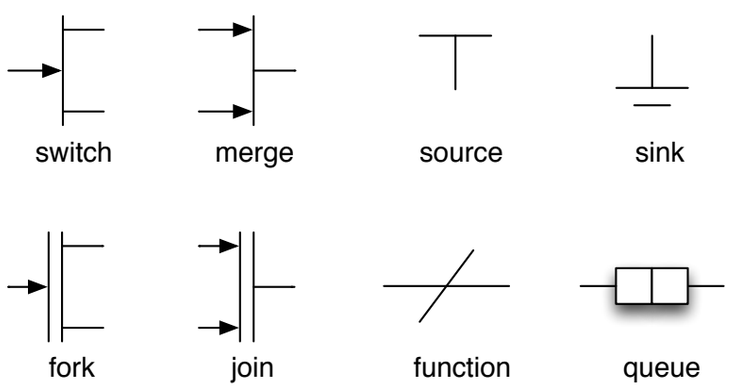
- Implemented xMAS editor with liveness algorithm:
  - http://www.cs.ru.nl/~freekver/algo_xmas/index.html



How will you verify your NoC?

xMAS





Properties

- Deadlock freedom
- Invariants
- No packet loss
- Correct destinations
- Correct payload

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

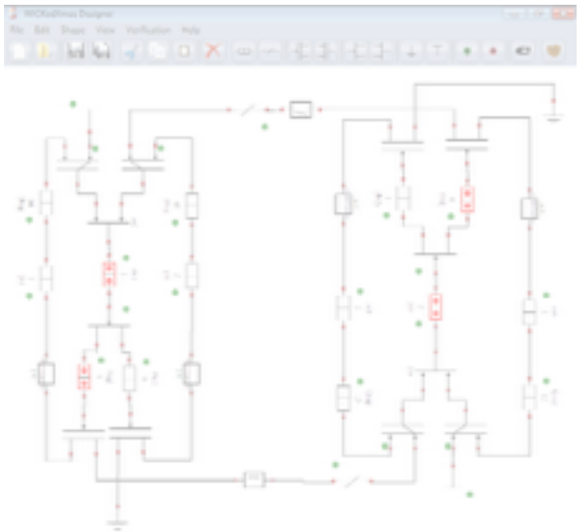
TU/e Technische Universiteit Eindhoven University of Technology

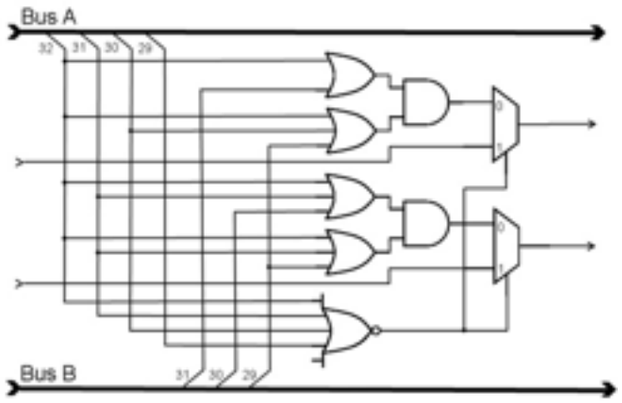
Three Levels of Abstraction

System Level

```
/*************/
/* ROUTING   */
/*************/

/*
Shortest path routing in the Spidergon ring
*/

#include "spidergon_definitions.h"


/*
The routing function. Routes from channel c == (x,p) to processing node n == (dx)
Returns a set of resources. As routing is deterministic, one next hops are returned.
Note that we initialize a list of 2 next hops: one next hop and one enclosing NULL.
*/
ResourceList* inst_routing(const Resource &c, Procnode* d, const Params* dim, ResourceList* frs) {
    ResourceList *hops = new ResourceListN<2>;
    Resource *nexthops = hops->routed;

                // The coordinates of the destination
                int dest = d->s;
                // The coordinates of the processing node at the end of channel c
                int next = get_end(c).s;

                // Compute relAd:
                int relAd = (dest - next + NUM_OF_PROC_NODES) % NUM_OF_PROC_NODES;

                if (relAd == 0) {
                                // No next hop
                }
else if (relAd == 1 || relAd == 2)
                                nexthops[0] = Resource(next, CW);
else if (relAd == 6 || relAd == 7)
                                nexthops[0] = Resource(next, CCW);
else
                                nexthops[0] = Resource(next, ACC);

return hops;
}
```

Micro-Architectural Level



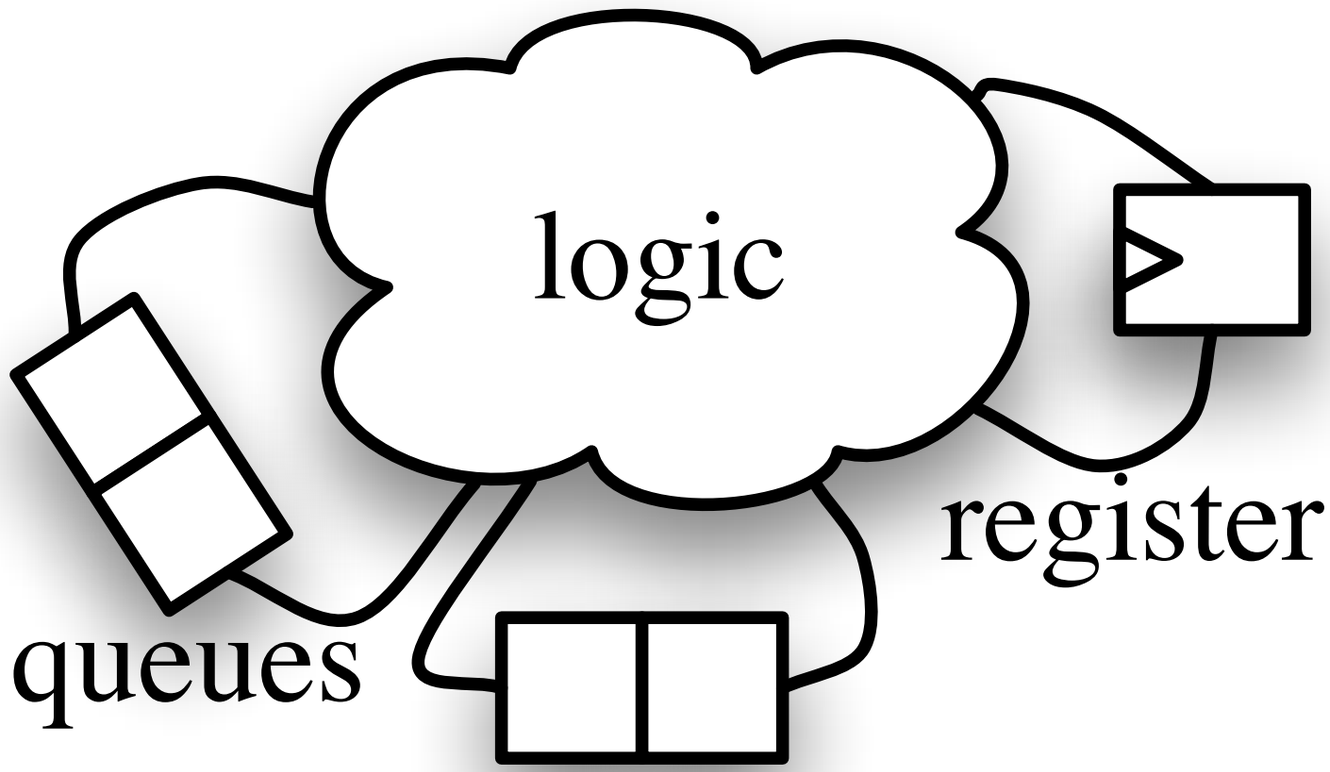


Register Transfer Level

Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

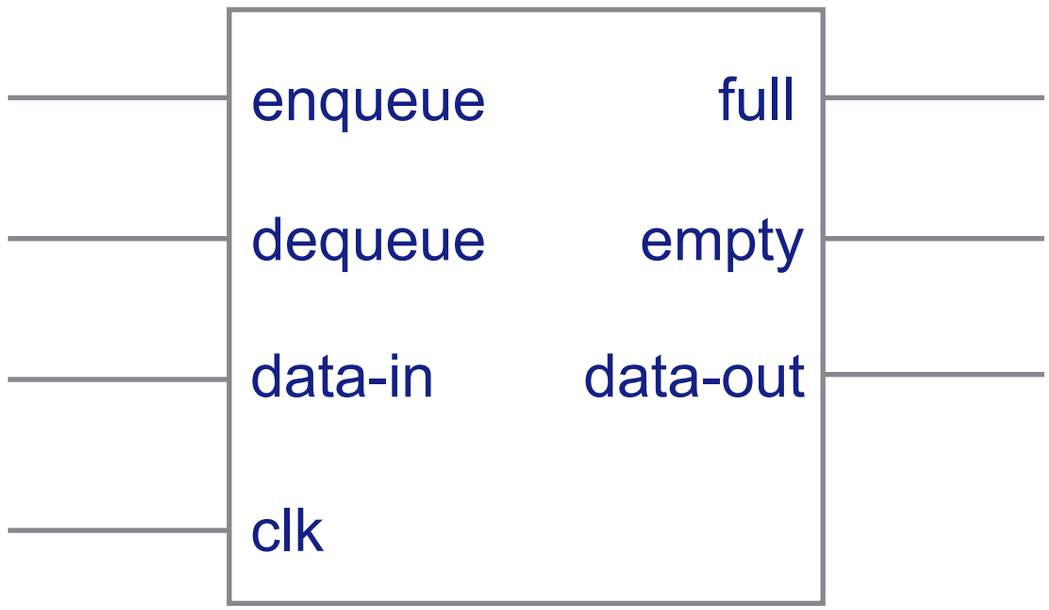
Network as queues and connections



Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

Network as queues and connections



Network as queues and connections

- Identify queues by inputs and outputs

```
port
  direction output;
  module fwft_fifo;
  data dout;
  ready !full;
  transfer wr_en && !full;
endport
```

Network as queues and connections

- For every queue:
  - When does a packet enter
  - When does a packet leave

- #packets in queue = #enter events – #leave events

TU/e Technische Universiteit Eindhoven University of Technology

How will you verify your Netlist?

- Inductive invariants for netlist
  - Sebastiaan J.C. Joosten, Julien Schmaltz
    Generation of Inductive Invariants from Register Transfer Level Designs of Communication Fabrics. MEMOCODE 2013
- Deadlock freedom / liveness for netlist
  - Sebastiaan J.C. Joosten, Julien Schmaltz
    Scalable Liveness Verification for Communication Fabrics. DATE 2014
- Model checking for liveness-like properties
  - Sayak Ray
    Effective Abstraction for Response Proof of Communication Fabrics. NOCS 2014

How will you verify your Netlist?

Register Transfer Level

```
\cpu_core(VC_NUM_PER_PORT=4,PYLD_WI
        .nios_reset(nios_reset), .c
        .flit_out_wr(\router_wr_in_
        .flit_in({\router_flit_out_
        .credit_out({\router_credit_
\router(VC_NUM_PER_PORT=4,BUFFER_NU
        \router_wr_in_en_array[3] [
        .flit_in_array({38'b0000000
        38'b000000000000000000000000
        .credit_out_array({Open_258
```

Properties

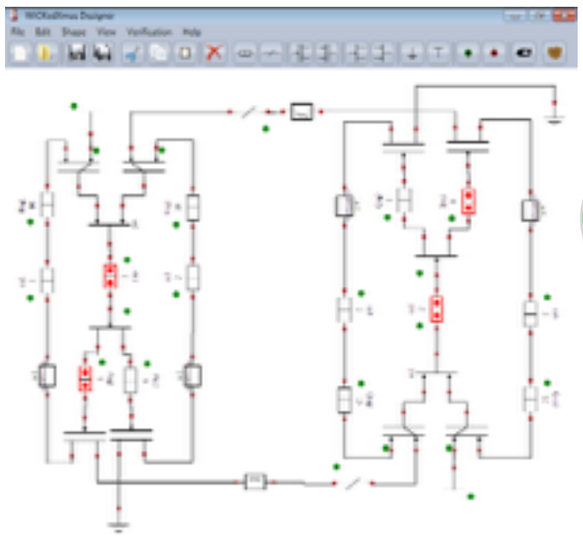
- Deadlock freedom
- Invariants

Conclusion

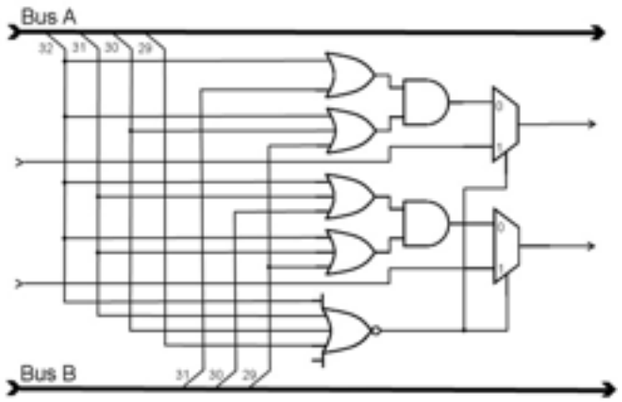
System Level

```
/**************/
/* ROUTING    */
/**************/

/*
Shortest path routing in the Spidergon ring
*/

#include "spidergon_definitions.h"


/*
The routing function. Routes from channel c == (x,p) to processing node n == (dx)
Returns a set of resources. As routing is deterministic, one next hops are returned.
Note that we initialize a list of 2 next hops: one next hop and one enclosing NULL.
*/
ResourceList* inst_routing(const Resource &c, Procnode* d, const Params* dim, ResourceList* frs) {
    ResourceList *hops = new ResourceListN<2>;
    Resource *nexthops = hops->routed;

                // The coordinates of the destination
                int dest = d->s;
                // The coordinates of the processing node at the end of channel c
                int next = get_end(c).s;

                // Compute relAd:
                int relAd = (dest - next + NUM_OF_PROC_NODES) % NUM_OF_PROC_NODES;

                if (relAd == 0) {
                                // No next hop
                }
                else if (relAd == 1 || relAd == 2)
                                nexthops[0] = Resource(next, CW);
                else if (relAd == 6 || relAd == 7)
                                nexthops[0] = Resource(next, CCW);
                else
                                nexthops[0] = Resource(next, ACC);

                return hops;
}
```

Micro-Architectural Level



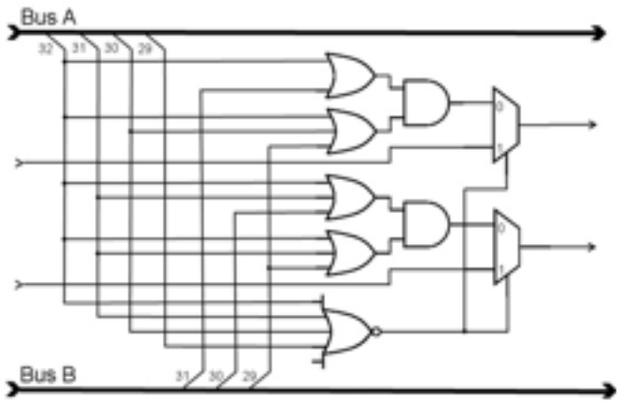
Register Transfer Level



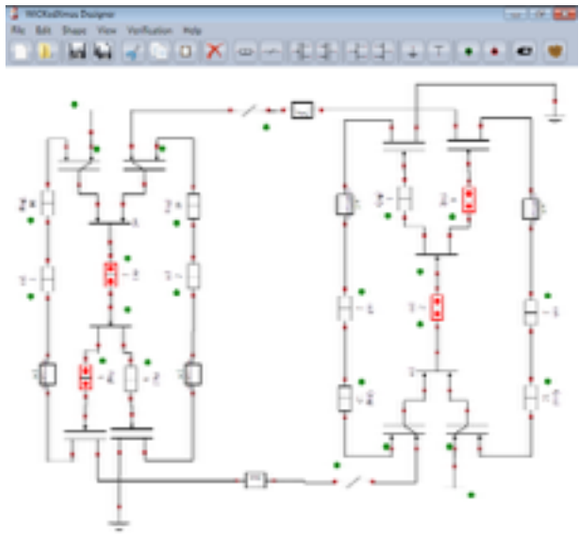
Sebastiaan J.C. Joosten, Department of Mathematics and Computer Science

TU/e Technische Universiteit Eindhoven University of Technology

Register Transfer Level



Micro-Architectural Level



What can be verified?

- Deadlock freedom / liveness for xMAS (and some other DSLs)
  - Freek Verbeek, PhD thesis 2013, Chapter 9
    Formal Verification of On-Chip Communication Fabrics.
- Correct payload for xMAS
  - Bernard van Gastel, Freek Verbeek, Julien Schmaltz
    Inference of channel types in micro-architectural models of on-chip communication networks. VLSI-SoC October 2014
- Inductive invariants for eMOD
  - Sebastiaan J.C. Joosten, Julien Schmaltz
    Generation of Inductive Invariants from Register Transfer Level Designs of Communication Fabrics. MEMOCODE 2013
- Deadlock freedom / liveness for eMOD
  - Sebastiaan J.C. Joosten, Julien Schmaltz
    Scalable Liveness Verification for Communication Fabrics. DATE 2014
- Generating xMAS from eMOD and Verilog
  - Not yet published. Email me: s.j.c.joosten@tue.nl

TU/e Technische Universiteit Eindhoven University of Technology