Dependability Techniques for Networks on Chips

Tomohiro Yoneda

National Institute of Informatics

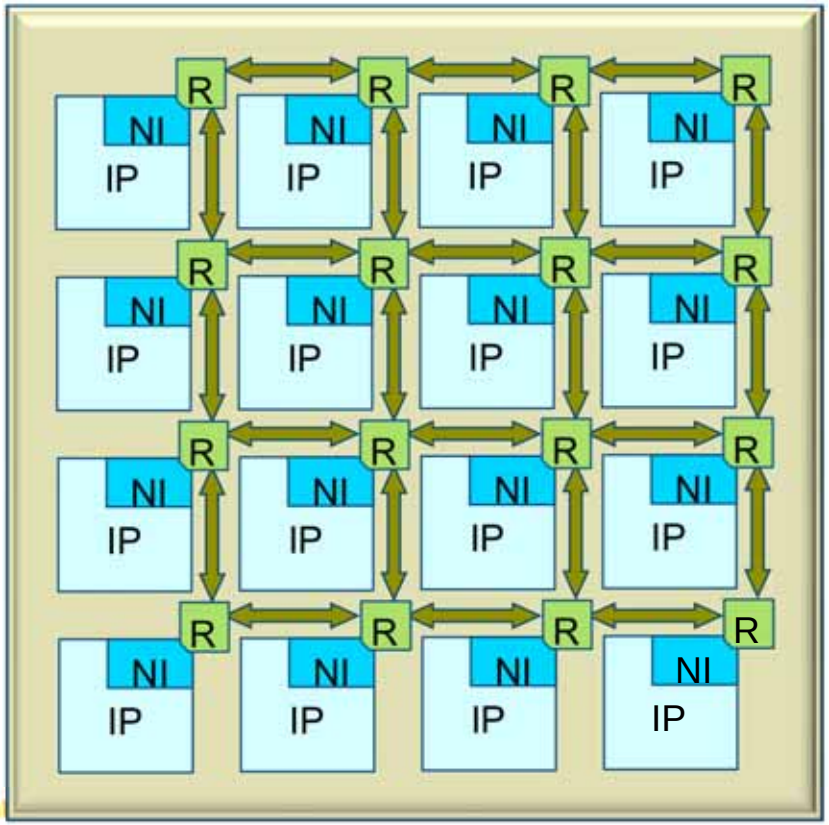
National Institute of Informatics
国立情報学研究所

Agenda

1. Basic notions on NoC
2. Dependable routing
   - Deadlock avoidance
   - On-line process
   - Off-line process
3. Challenges in NoCs
4. Conclusion

What's NoC?

- System-on-Chip with on-chip networks
  - Communication is done by packet switching



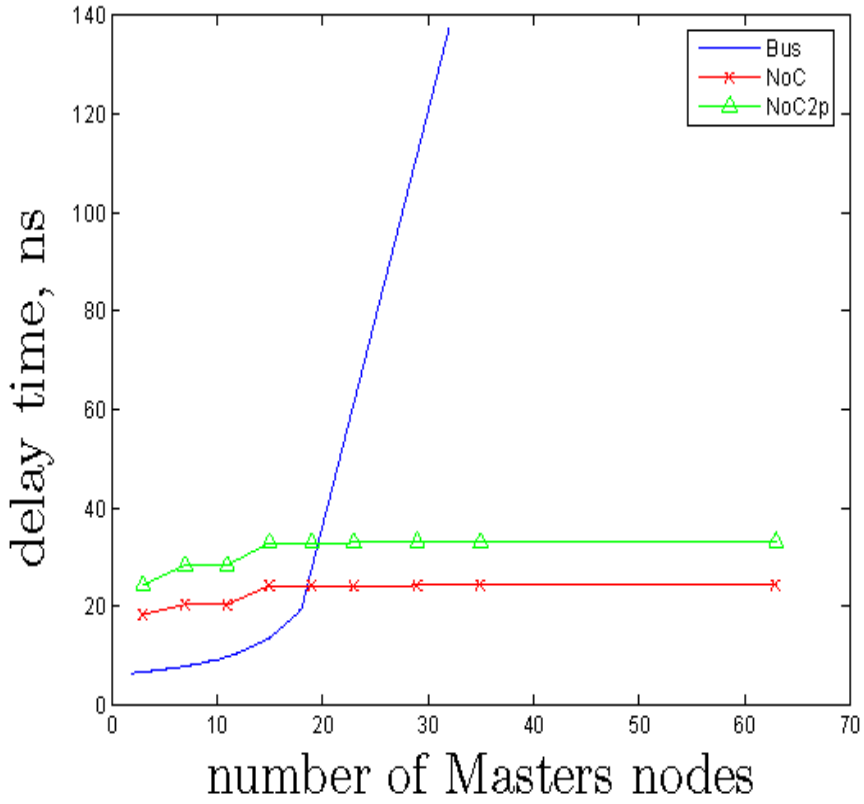
IP : CPU/Accelerator Core
NI: Network Interface
R: Router

Existing Chips

- Intel
  - Polaris: 80 cores
  - SCC (Single-Chip Cloud Computer): 48 cores
- Tilera
  - TILE64: 64 cores
- LETI
  - FAUST: 20 cores
  - ALPIN: 9 cores
  - MAGALI: 15 cores

Why NoC?

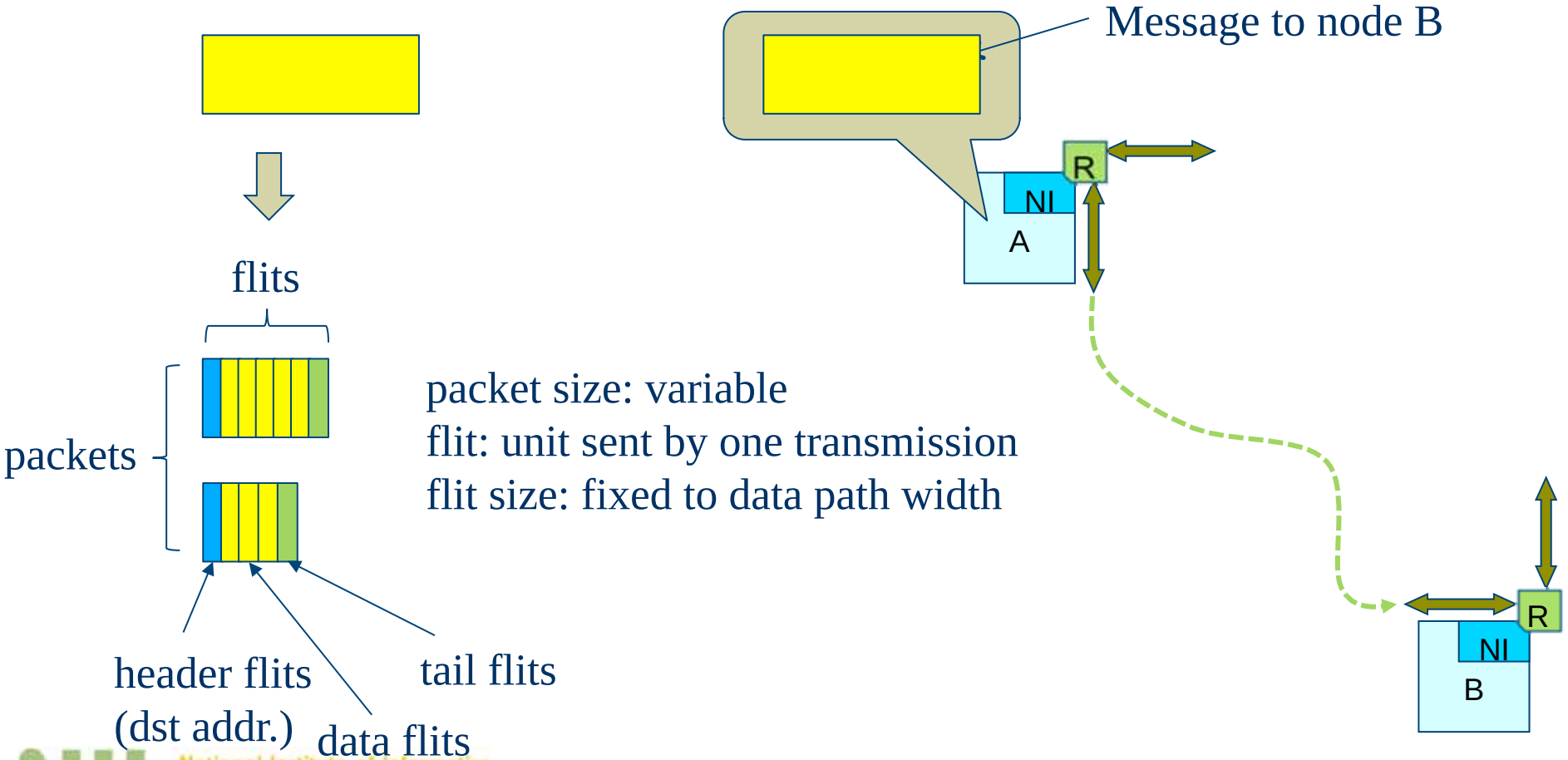
- Many cores need to be integrated on a chip
  - Bus architectures do not scale



Asynchronous Network-on-Chip Communication Architecture Performance Analysis
by
Bas Bijlsma and Rene van Leuken

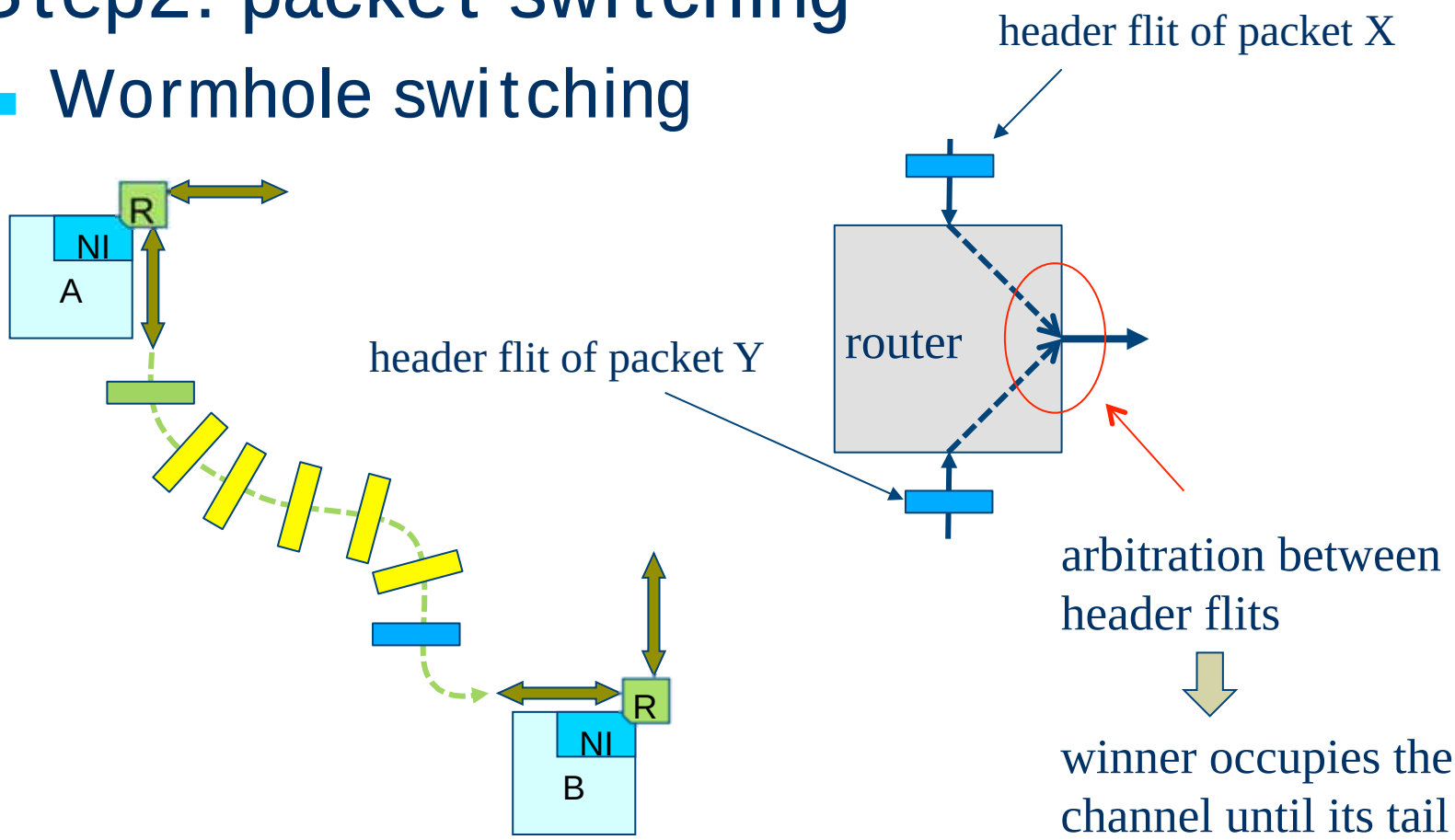
Node to node data communication

- Step1: Packetization



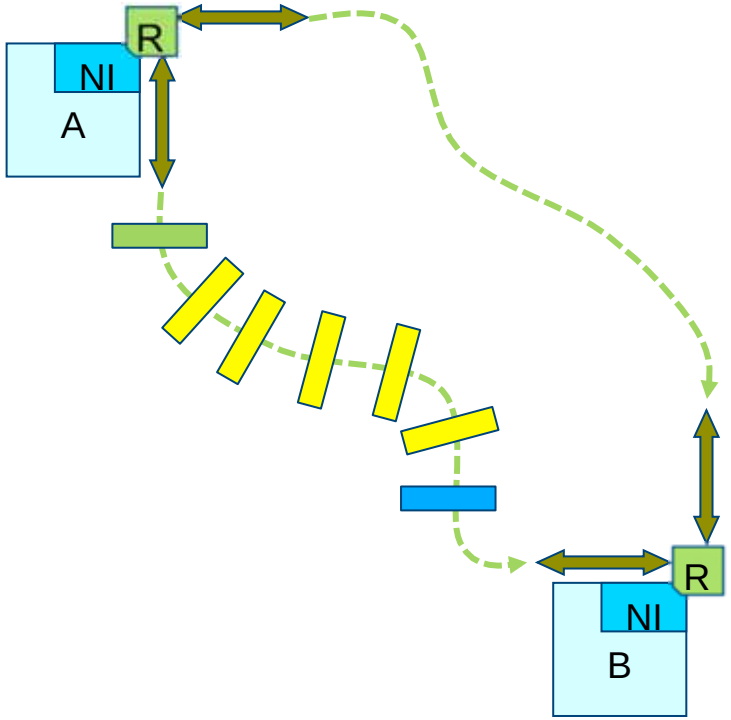
Node to node data communication

- Step2: packet switching
  - Wormhole switching



Node to node data communication

- Step2: packet switching (continued)
  - Path selection: routing algorithm



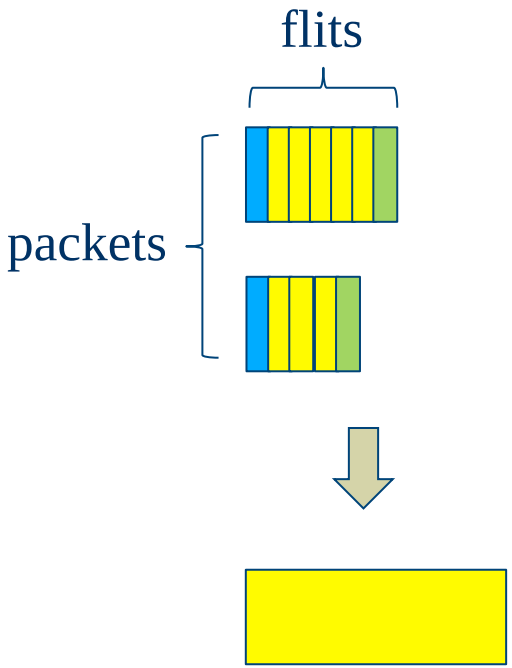
deterministic routing algorithm
→ fixed path for each src-dst pair

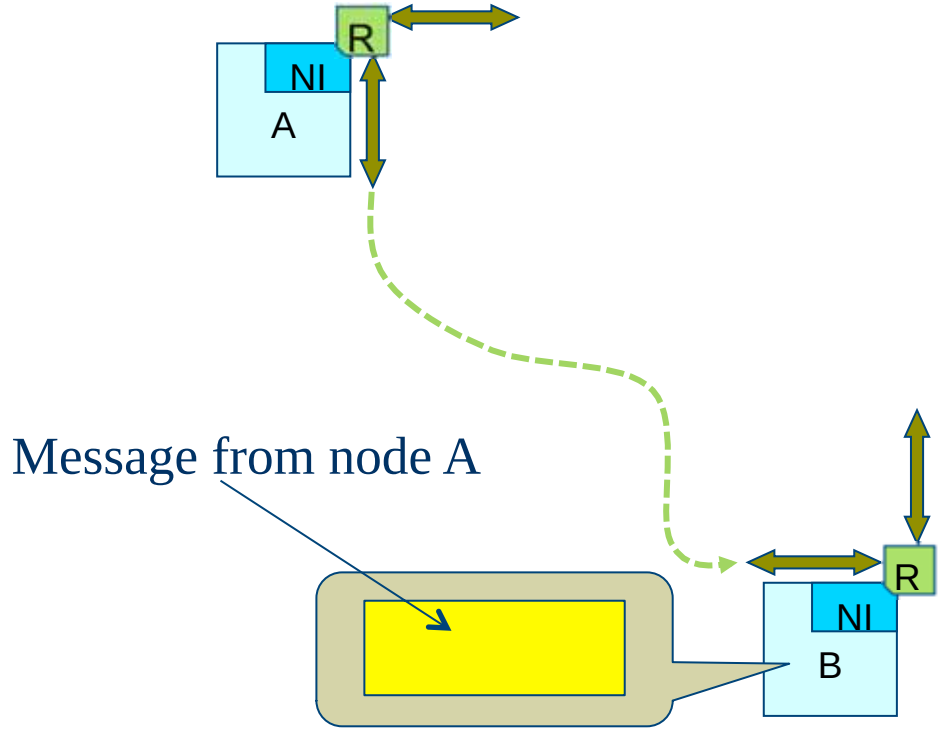
adaptive routing algorithm
→ dynamically selected path based on traffic or other information
(path is fixed by a header flit)

deadlock should be avoided

Node to node data communication

- Step3: Assembling packets to messages

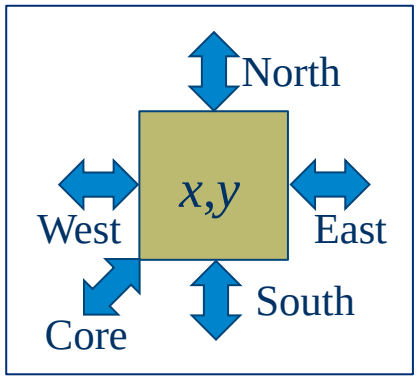


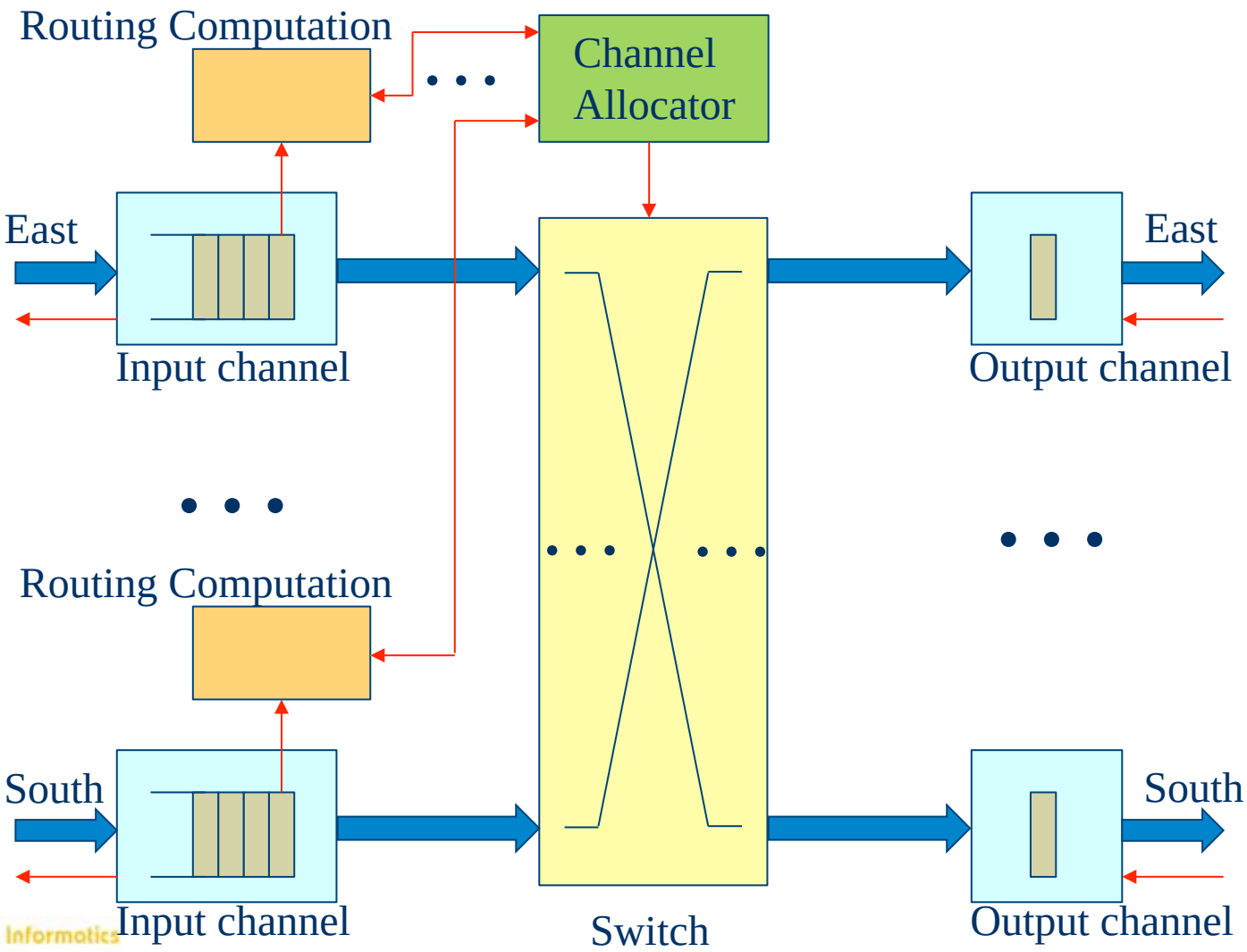


How different is NoC?

	NoC	Computer Network
Communication latency	low	high
Network bandwidth	high	low-medium
Communication granularity	small	large
Reliability	high	low
Routing computation	simple hardware	software/complex hardware
Power consumption	restricted	not restricted
Network topology	simple/regular	complex

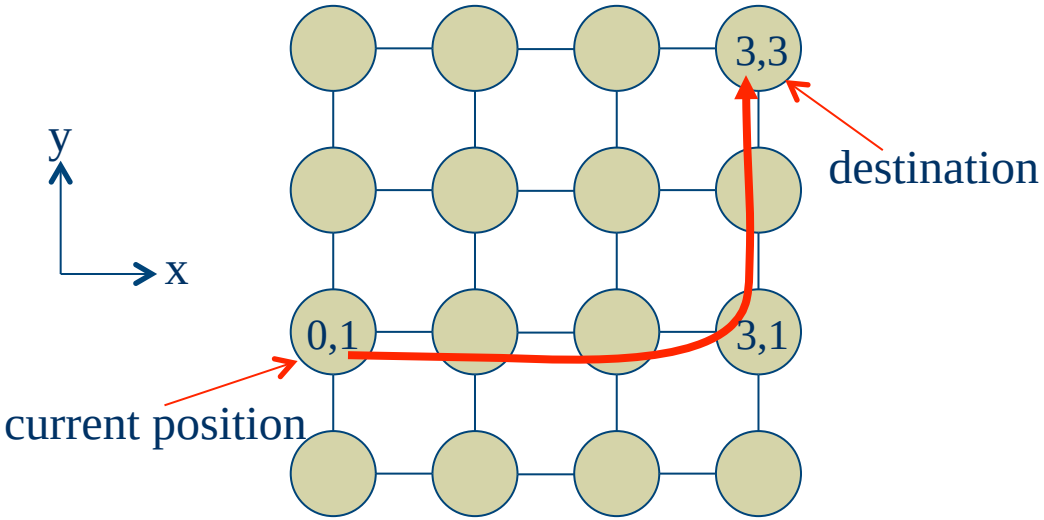
Router Architecture

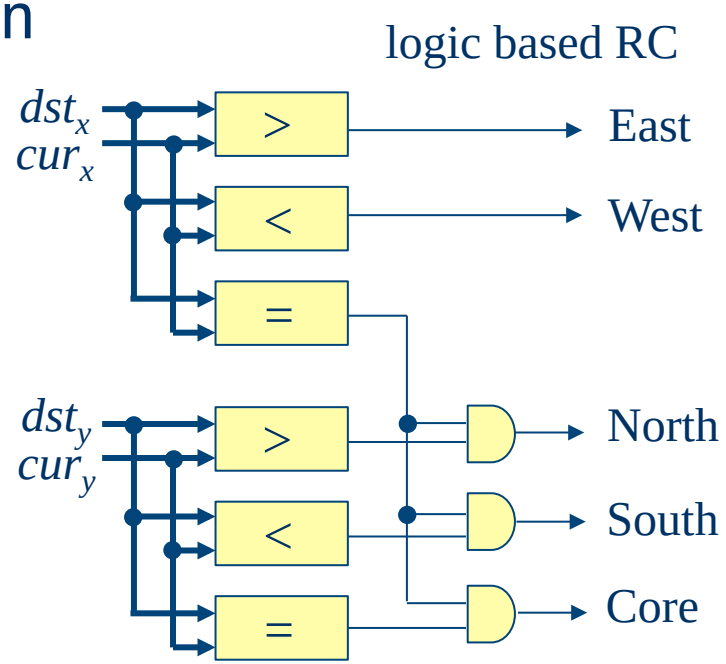




Routing Computation

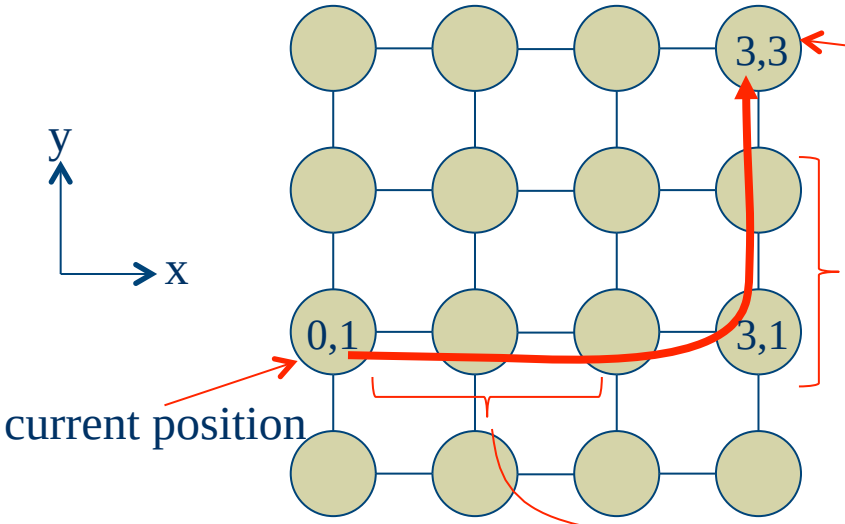
- Implementation of a routing algorithm
  - E.g., XY (dimension-order) routing
    - Send to X direction until $cur_x = dst_x$
    - Then, send to Y direction





Routing Computation

- Implementation of a routing algorithm
  - E.g., XY (dimension-order) routing
    - Send to X direction until $cur_x = dst_x$
    - Then, send to Y direction



Routing table based RC

DST	port
0,0	
0,1	
…	
x,y	
…	
3,3	E

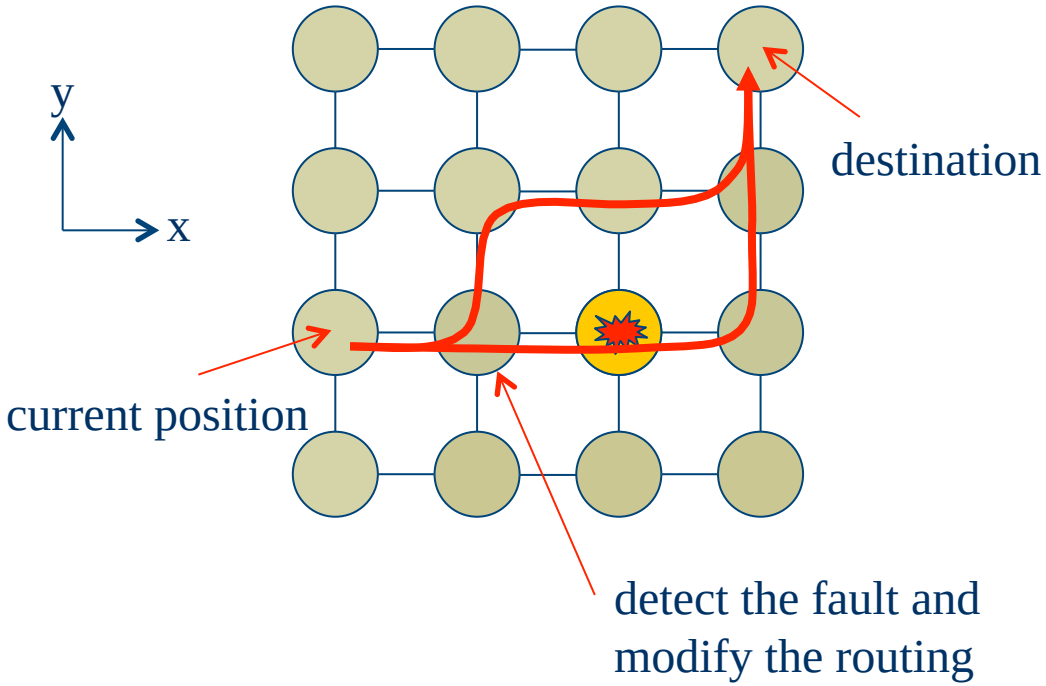
DST	port
0,0	
0,1	
…	
x,y	
…	
3,3	N

Routing Computation

- Implementation of a routing algorithm
  - Logic based RC
    - Small delay
    - Detour routings hard-wired
    - Used for simple routing algorithms
  - Routing table based RC
    - Relatively large delay
    - Easy to update and modify
    - Used for off-line approaches

Fault tolerance

- When a router goes faulty, a detour path should be taken

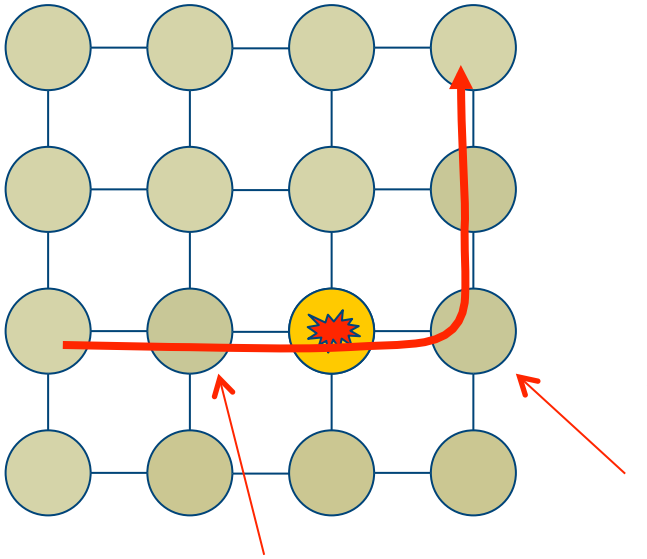


Dependable routing

- Assumptions
  - Distributed (decentralized) algorithms used
  - Faulty routers, links or cores are detected by the neighboring routers
  - Some packets may be lost before detour paths are taken
    - Affected packets should be discarded
- Important issue
  - Deadlock avoidance
  - Connectibity

Transient case

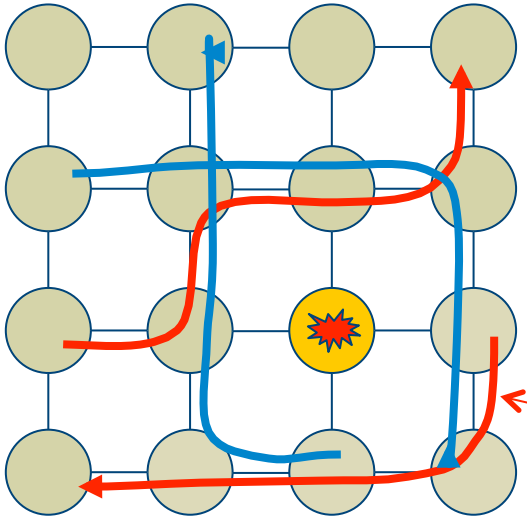
- One way to discard affected packets



fault detection
→ produce a pseudo tail flit from West port to release the occupied path

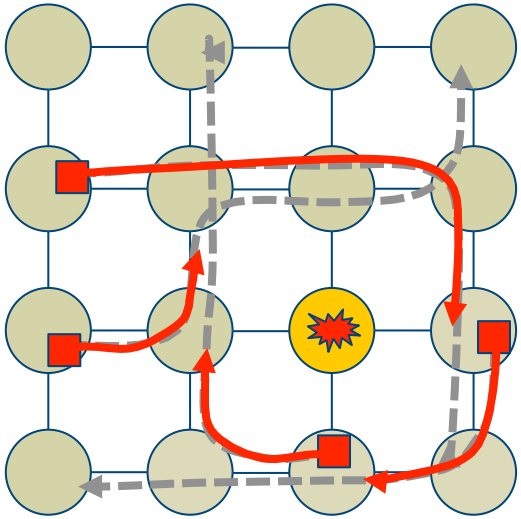
fault detection
→ discard flits for East port until the tail flit is found

Possibility of deadlock!



another detour path

Possibility of deadlock!



Paths occupy several resources

↓

Resource dependencies form a loop

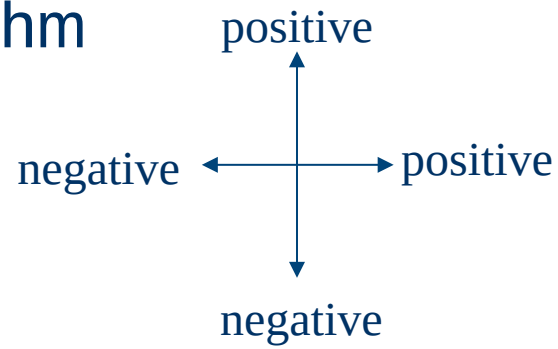
↓

Deadlock occurs

Deadlock avoidance

- Classical problem
  - Requirement for NoCs: simply implementable
- One approach
  - Restrict turns
    - Negative first routing algorithm

Turns disallowed

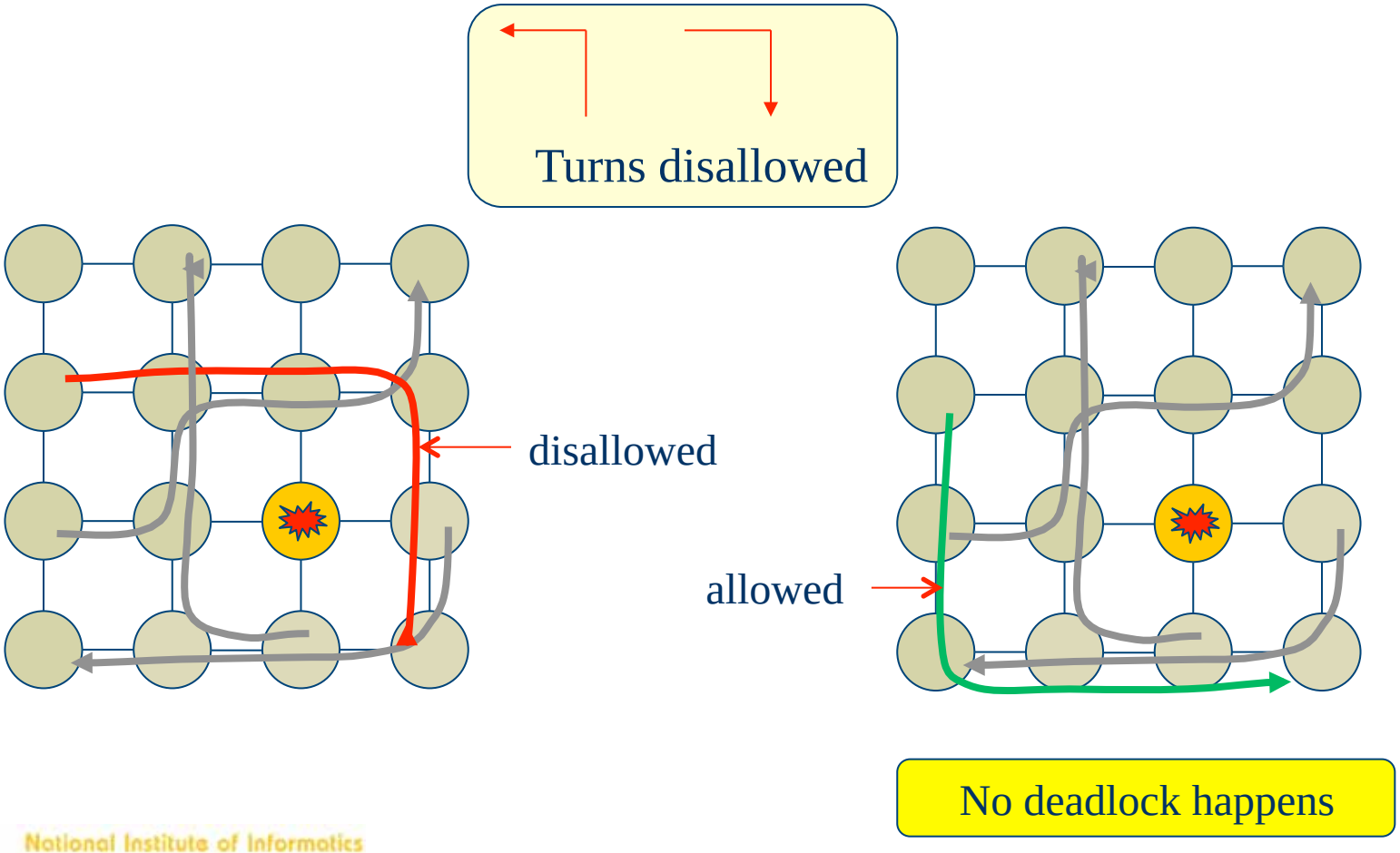


    - Up/Down routing algorithm

Turns disallowed

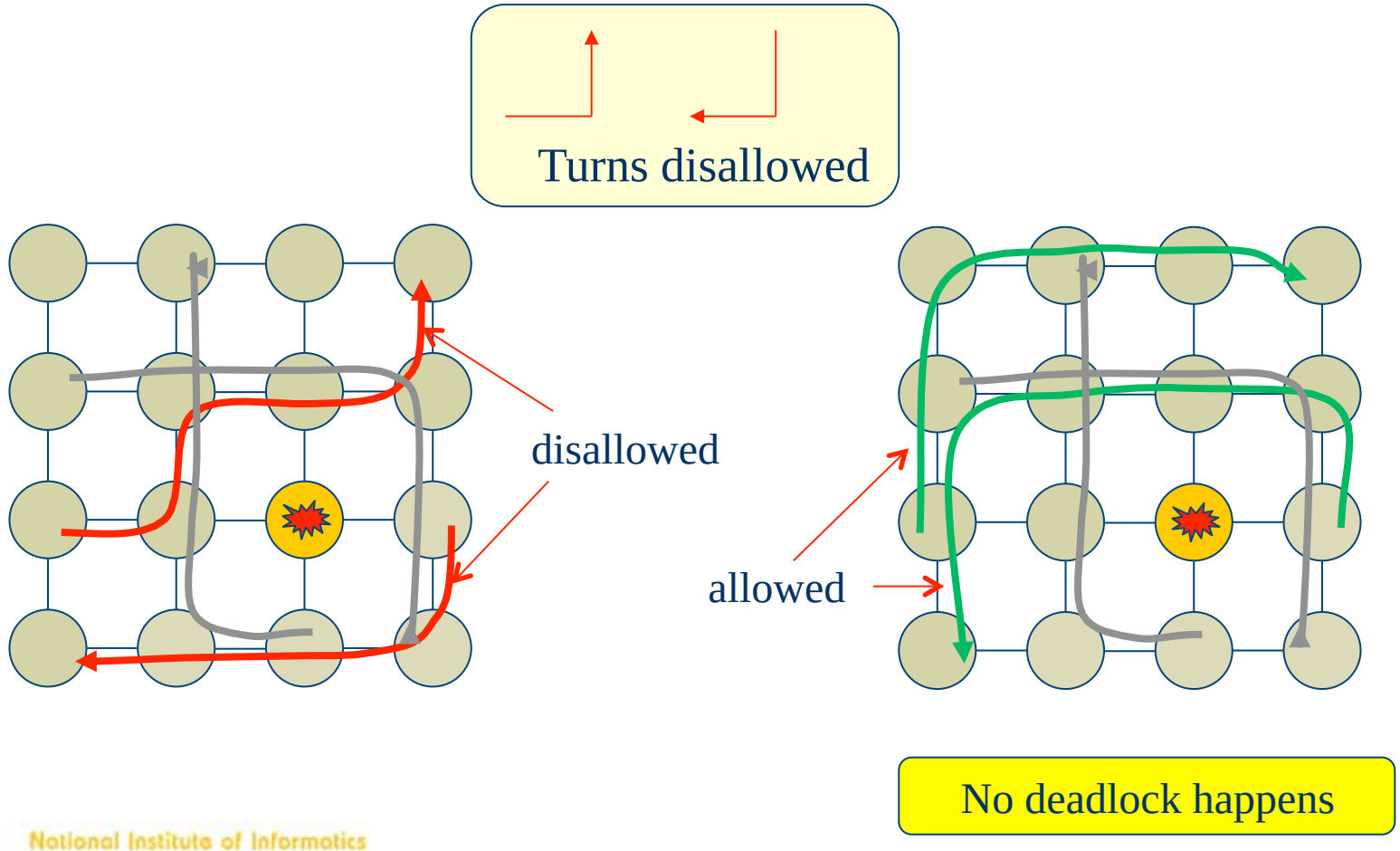
Routing Algorithms

- Negative first routing algorithm



Routing Algorithms

- Up/Down routing algorithm

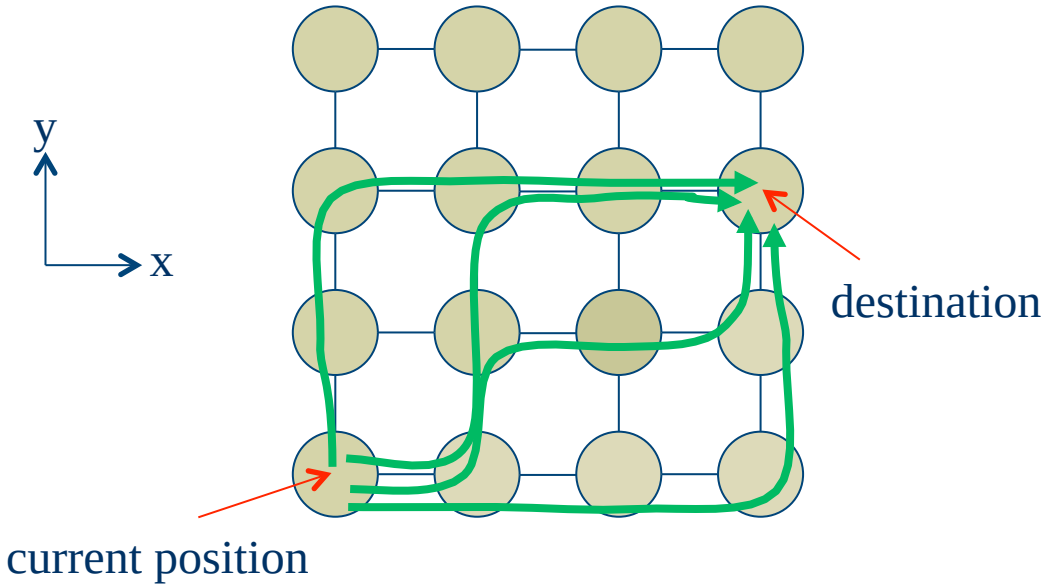


Dependable routing algorithm

- Glass-Ni algorithm [FTCS23]
  - Based on Negative First alg.
    - Any moves allowed unless negative moves occur later
  - Special case handling

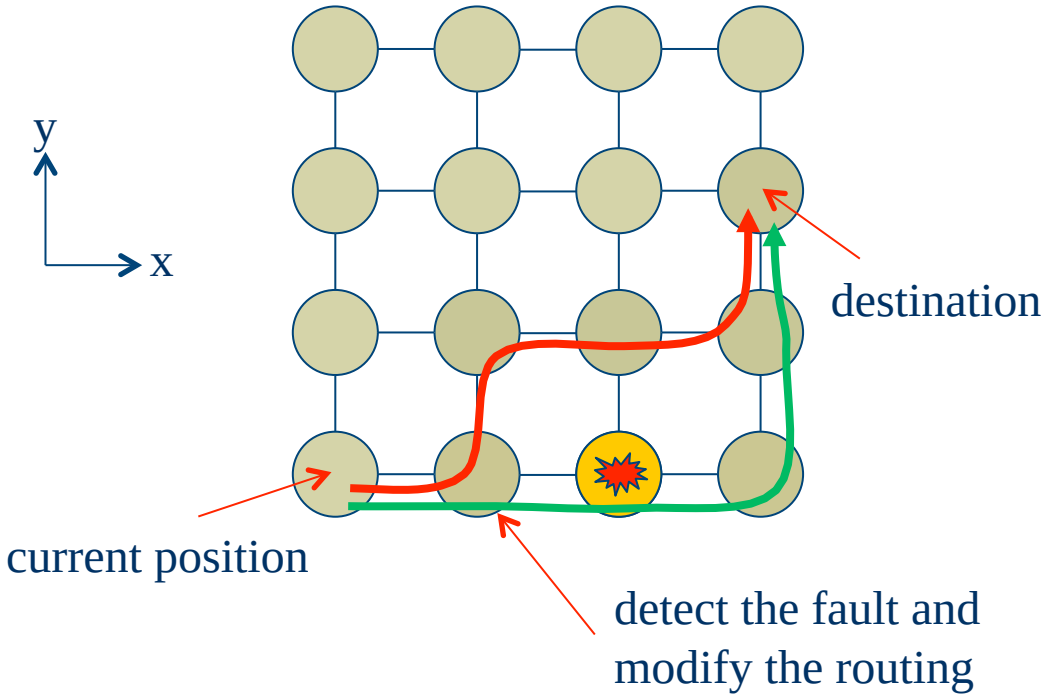
Dependable routing algorithm

- When only positive moves are needed



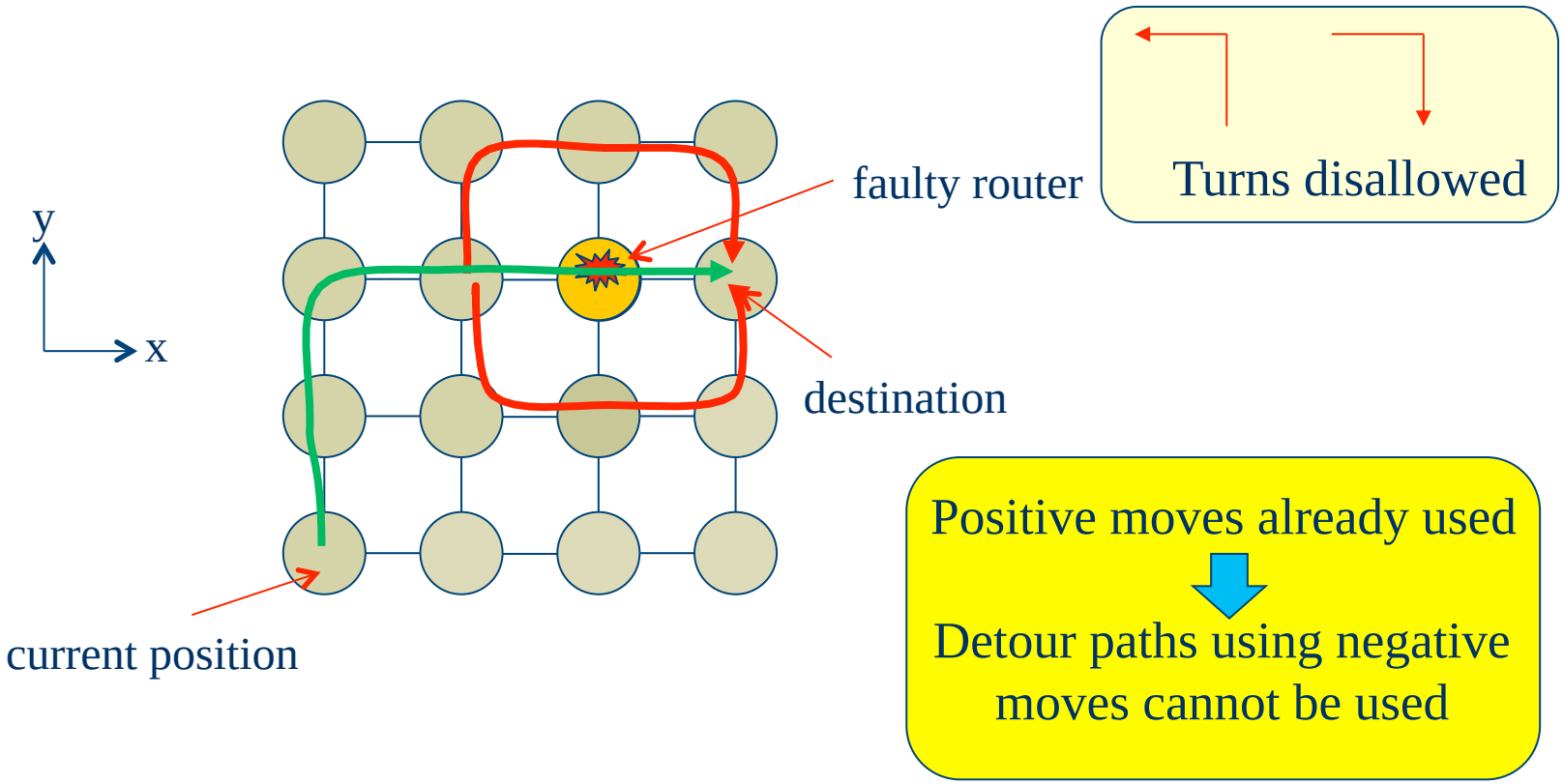
Dependable routing algorithm

- When only positive moves are needed



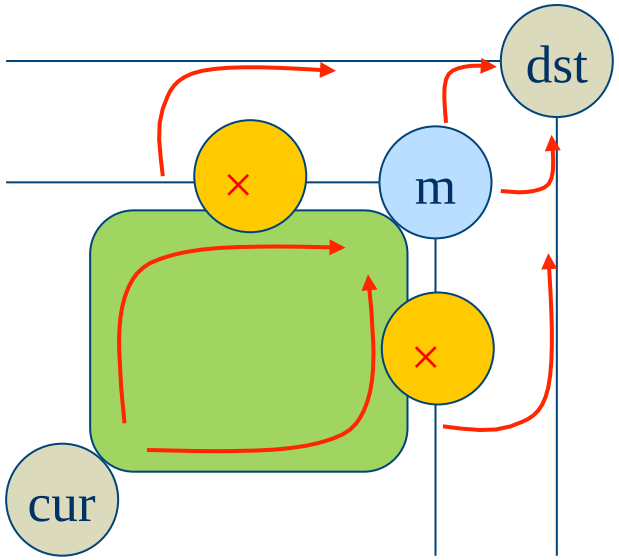
Dependable routing algorithm

- When only positive moves are needed



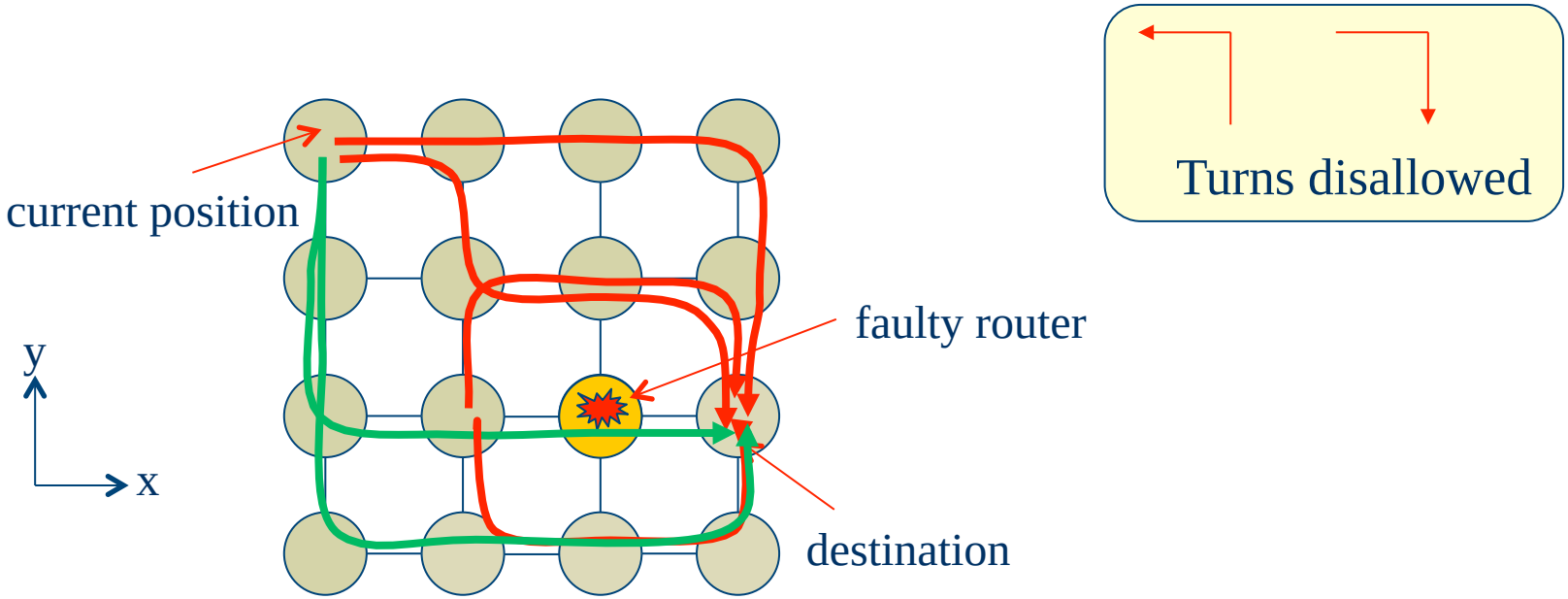
Dependable routing algorithm

- When only positive moves are needed



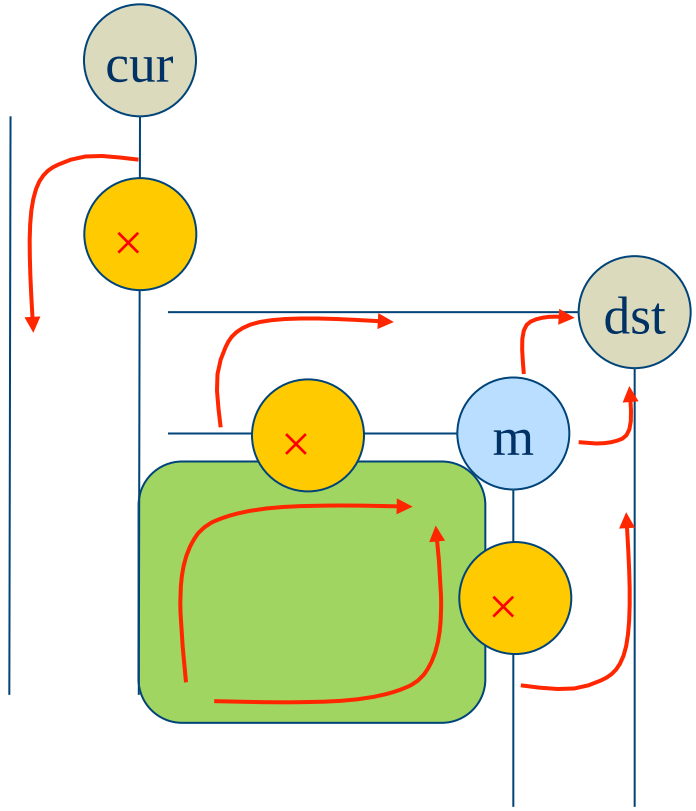
Dependable routing algorithm

- When both positive and negative moves are needed



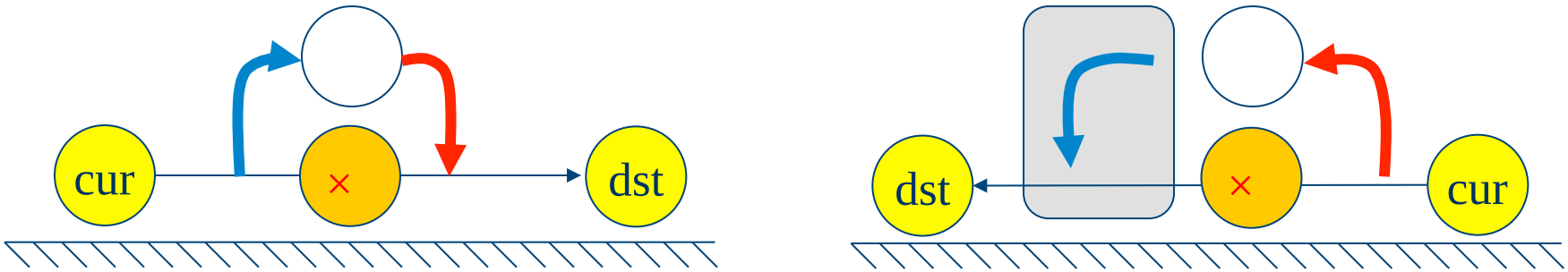
Dependable routing algorithm

- When both positive and negative moves are needed



Dependable routing algorithm

- Special cases: When a faulty node exists on a negative edge
  - Some of disallowed turns can be used without causing deadlock



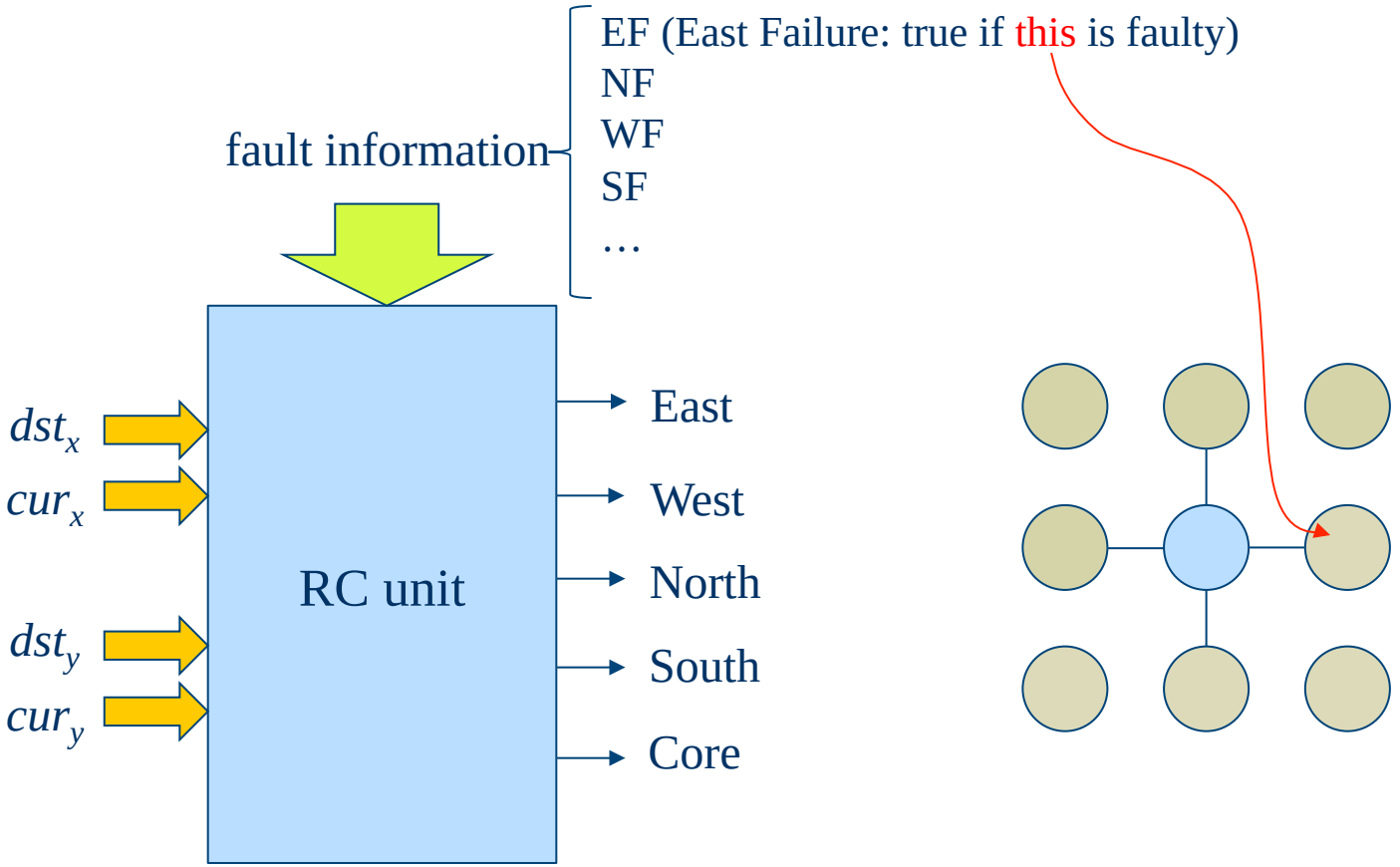
No space for forming loops

Applying dependable routing

- Approaches
  - On-line process
    - Routing function is dynamically modified using faulty information that each router obtains
  - Off-line process
    - An off-line process that gathers faulty information among the whole routers is invoked when a faulty is detected somewhere

On-line approach

- Logic based implementation



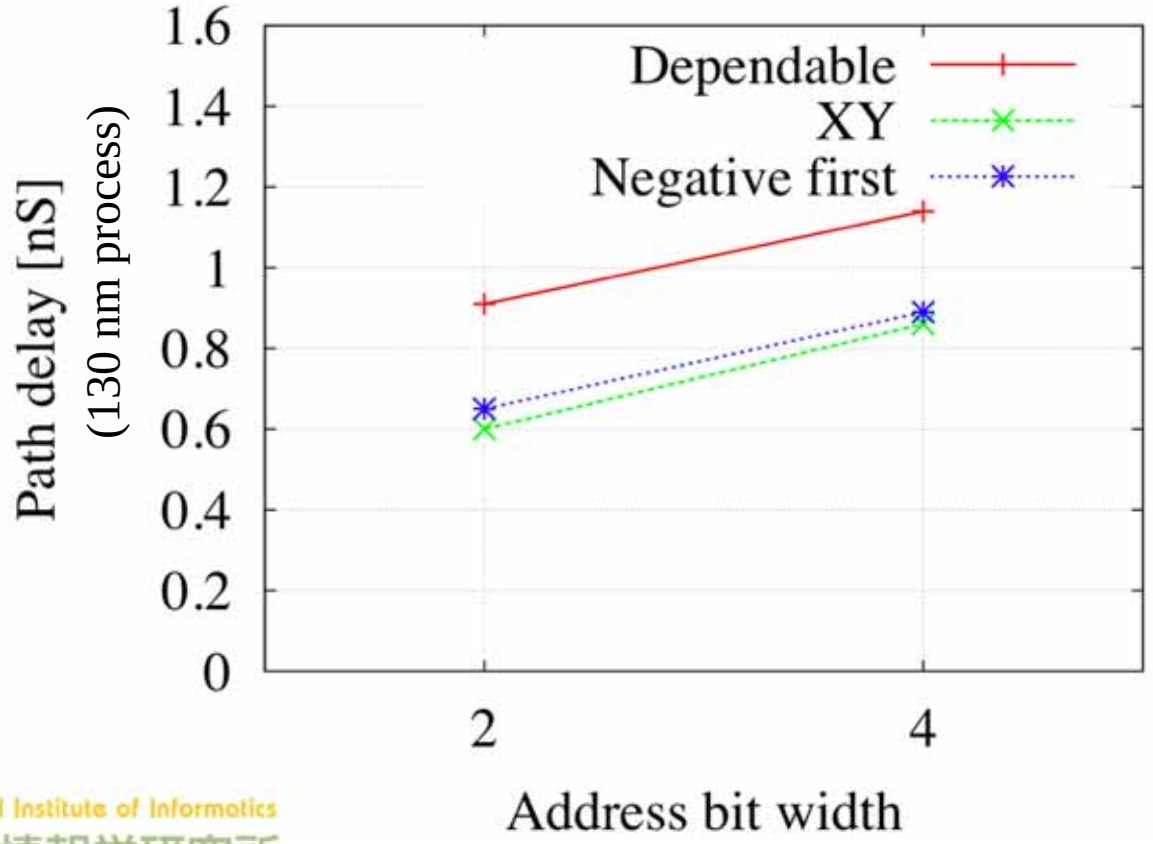
On-line approach

- Logic based implementation

```
function [4:0] rc_org;
    input signed [(`ADR_RNG+1):0] x_off, y_off;
    input [`ADR_RNG:0]     x_dst, y_dst;
    input [`ADR_RNG:0]     x_cur, y_cur;
    input     NF, NCF, NEF;
    input     SF, SEF;
    input     EF, ERF, ENF;
    input     WF, WNF;

    if ((x_off == 0) && (y_off == 1) && (NF == 0)) rc_org = 5'b00100;
    else if ((x_off == 0) && (y_off == -1) && (SF == 0)) rc_org = 5'b00001;
    else if ((y_off == 0) && (x_off == 1) && (EF == 0))  rc_org = 5'b00010;
    else if ((y_off == 0) && (x_off == -1) && (WF == 0)) rc_org = 5'b01000;
    else if ((mod(x_cur) == 1) && (mod(x_dst) == 0) &&  (y_off > 0) &&
                        ((WNF == 1) || (WF == 1))) rc_org = 5'b00100;
    else if ((y_cur == 1) && (y_dst == 0) &&  (x_off > 0) &&
                        ((SEF == 1) || (SF == 1))) rc_org = 5'b00010;
    else if ((mod(x_cur) == 0) && (y_off <= 0) &&  (SF == 1)) rc_org = 5'b00010;
    else if ((mod(x_cur) == 0) && (y_off > 0) && (x_off == 0)) begin
          if (NF == 1) rc_org = 5'b00010;  else rc_org = 5'b00100;
    end
    else if ((y_cur == 0) && (x_off <= 0) && (WF == 1)) rc_org = 5'b00100;
    else if ((y_cur == 0) && (x_off > 0) && (y_off == 0)) begin
          if (EF == 1) rc_org = 5'b00100;  else rc_org = 5'b00010;
     end
    else if ((x_off < 0) && (y_off < 0)) begin
          if ((x_dst+1-x_cur == 0) && (y_dst+1-y_cur == 0)) rc_org = 5'b01001;
          else if (x_dst+1-x_cur == 0) begin
             if (SF == 1) rc_org = 5'b01000; else rc_org = 5'b00001;
          end
          else if (y_dst+1-y_cur == 0) begin
             if (WF == 1) rc_org = 5'b00001; else rc_org = 5'b01000;
          end
          else rc_org = 5'b01001;
    end
    else if ((x_off > 0) && (y_off > 0)) begin
          if ((x_dst-1-x_cur == 0) && (y_dst-1-y_cur == 0)) begin
             if ((NEF == 1) && (ENF == 0)) rc_org = 5'b00010;
             else if ((NEF == 0) && (ENF == 1)) rc_org = 5'b00100;
             else rc_org = 5'b00110;
          end
          else if (x_dst-1-x_cur == 0) begin
             if (NF == 1) rc_org = 5'b00010;
             else rc_org = 5'b00100;
          end
          else if (y_dst-1-y_cur == 0) begin
             if (EF == 1) rc_org = 5'b00100;
             else rc_org = 5'b00010;
          end
          else rc_org = 5'b00110;
    end

    else if ((x_off >= 0) && (y_off <= 0)) begin
          if ((y_off == 0) && (ERF == 0)) rc_org = 5'b00010;
          else if (SF == 1) rc_org = 5'b01000; else rc_org = 5'b00001;
    end
    else if ((x_off <= 0) && (y_off >= 0)) begin
          if ((x_off == 0) && (NCF == 0)) rc_org = 5'b00100;
          else if (WF == 1) rc_org = 5'b00001; else rc_org = 5'b01000;
    end
    else rc_org = 5'b0;

endfunction
```

On-line approach

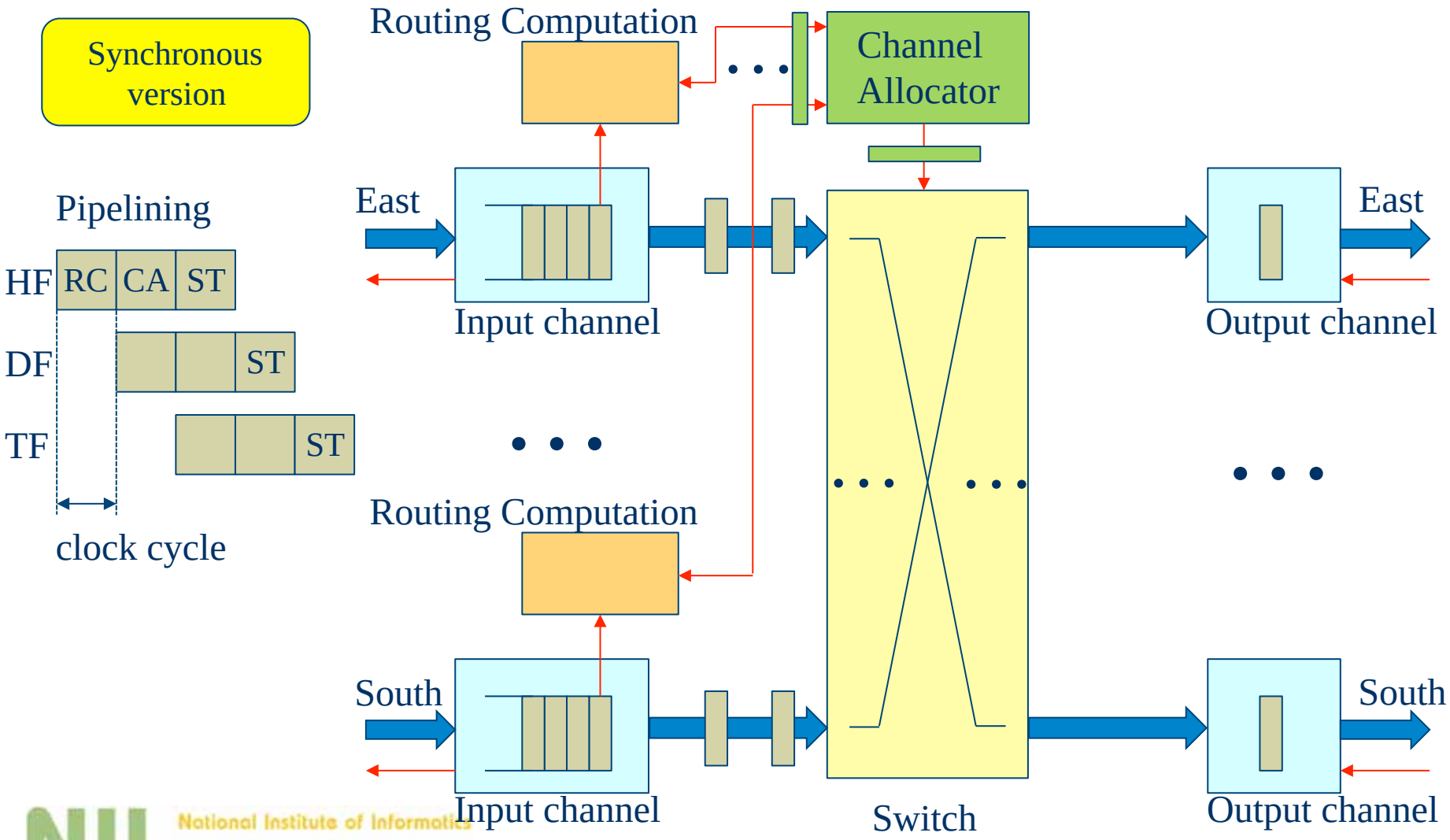
- Critical path delays of RC units for different routing algorithm



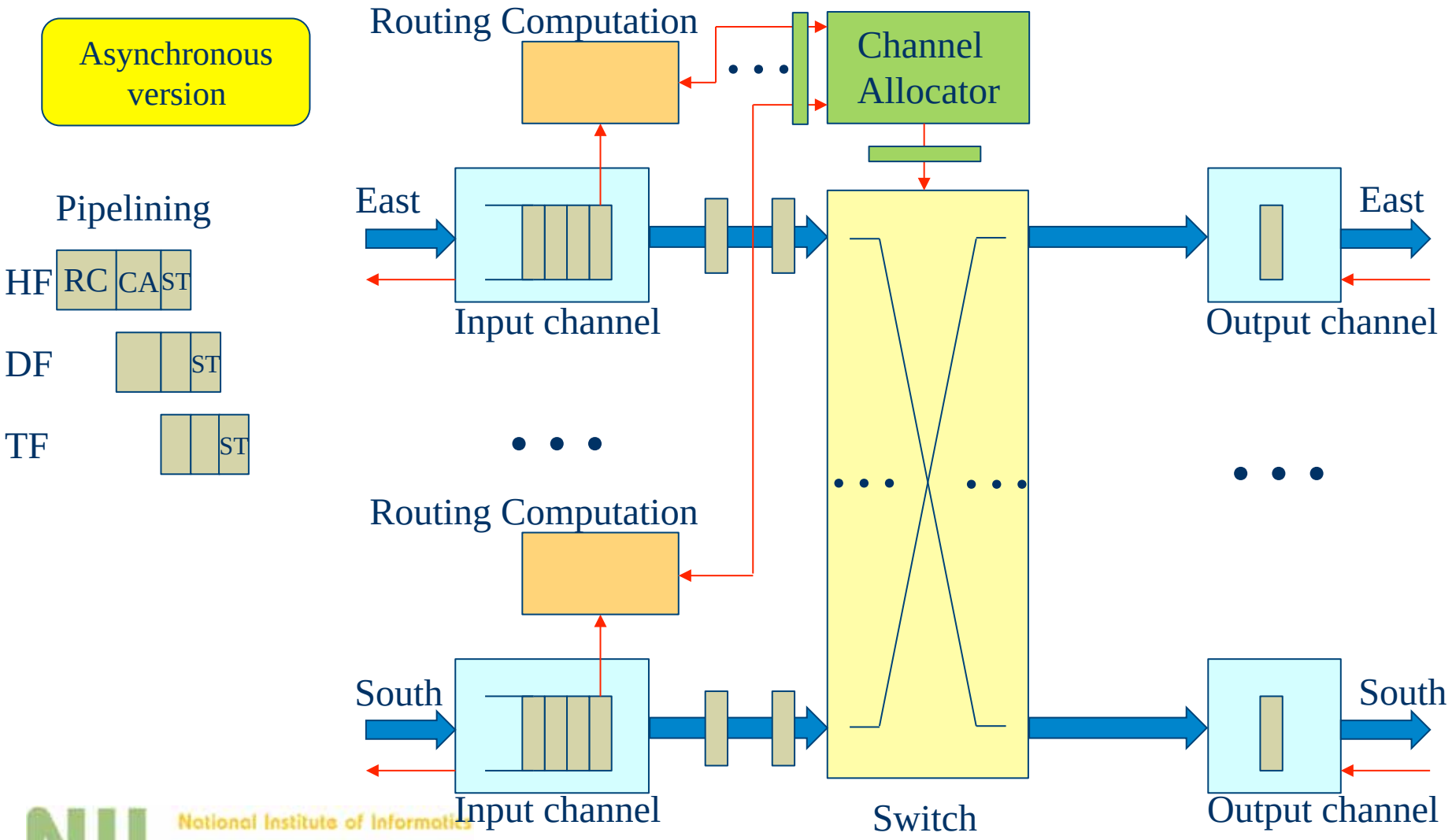
On-line approach

- Logic based implementation
  - Complicated conditions make the routing computation circuit large
    - Critical path delay for some specific cases becomes large
    - Synchronous implementation
      - Large pipeline stage → low clock frequency
    - Asynchronous implementation
      - Self-timed circuits
      - Performance depends on delays of the sub-circuits actually activated for the given cases

On-line approach

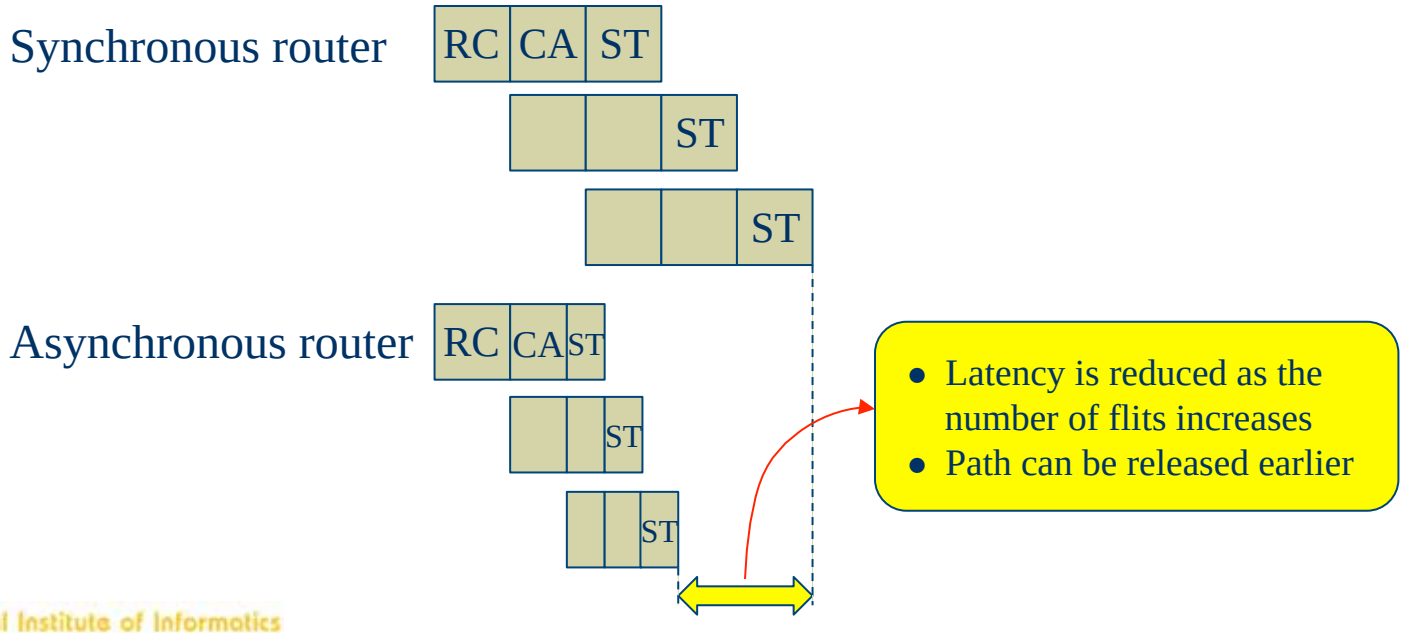


On-line approach



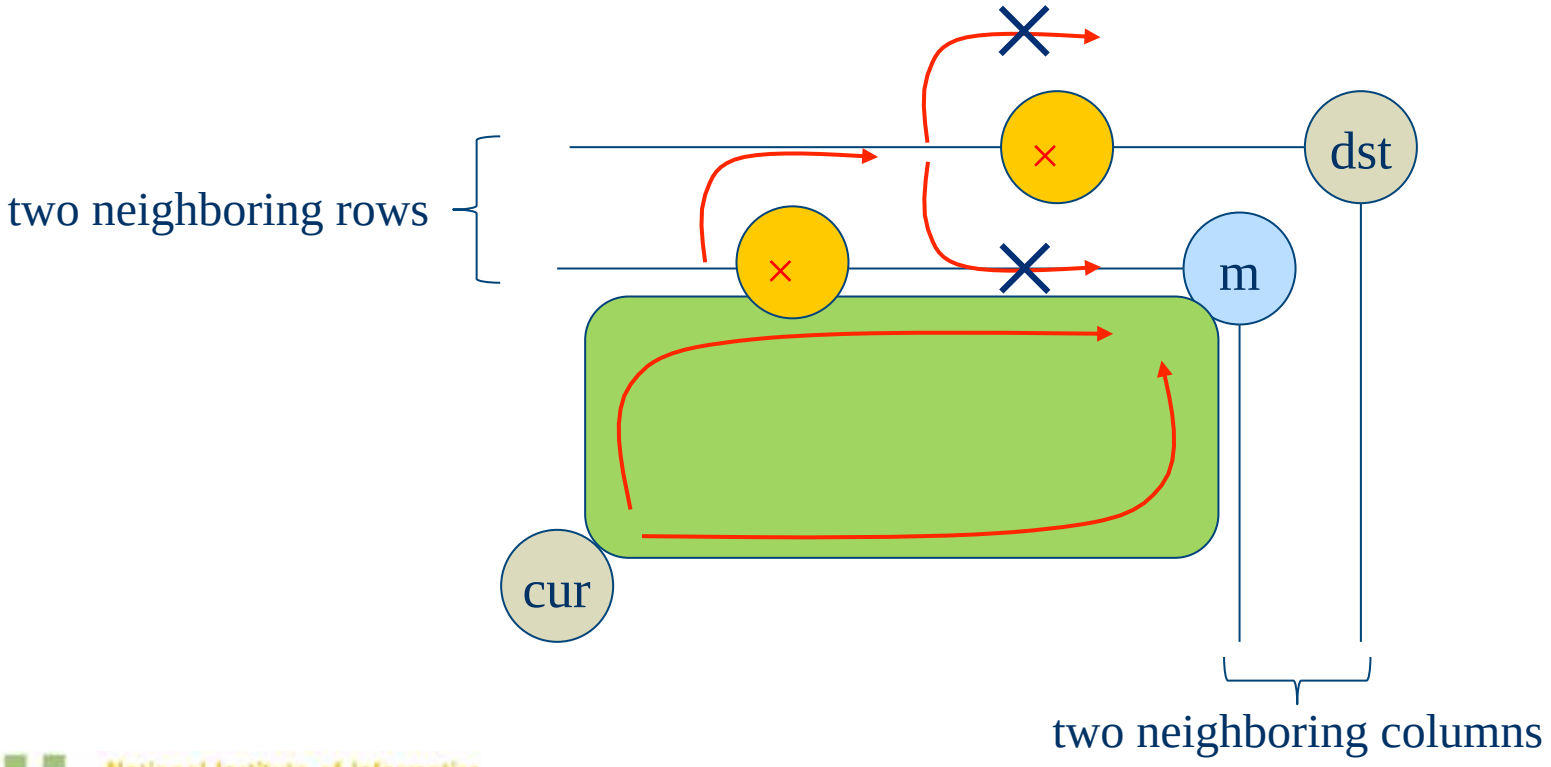
On-line approach

- Handling head flits takes longer time
  - Synchronous router
    - Unnecessary slacks are given for the other flits
  - Asynchronous router
    - Those flits can go quickly



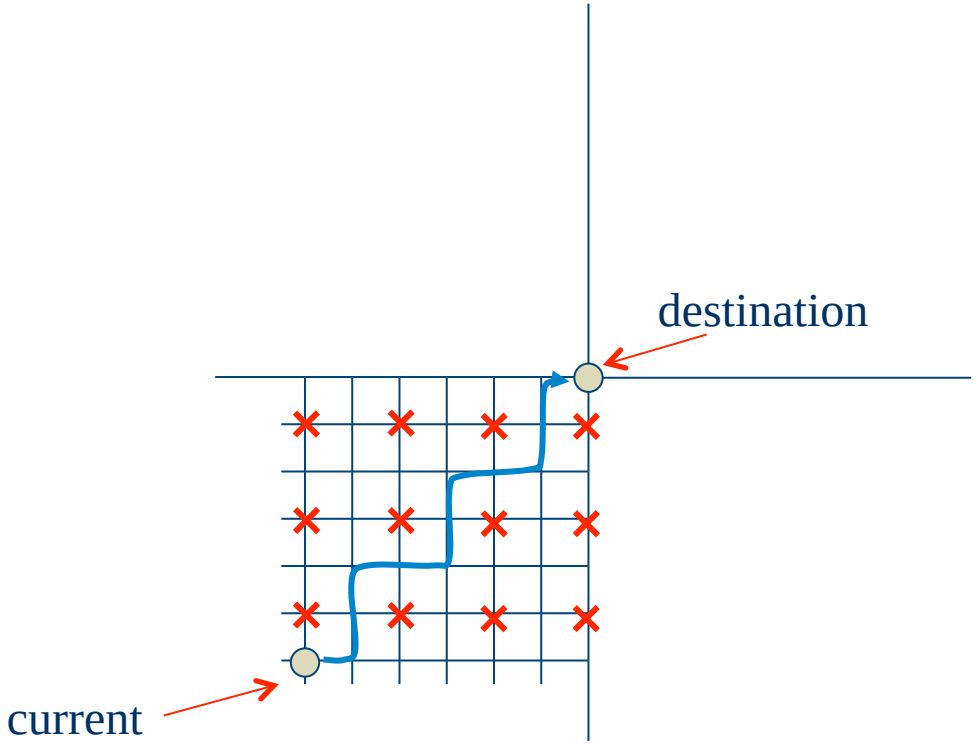
On-line approach

- Original algorithm assumes a single fault
  - Double-fault pattern cannot be handled, if



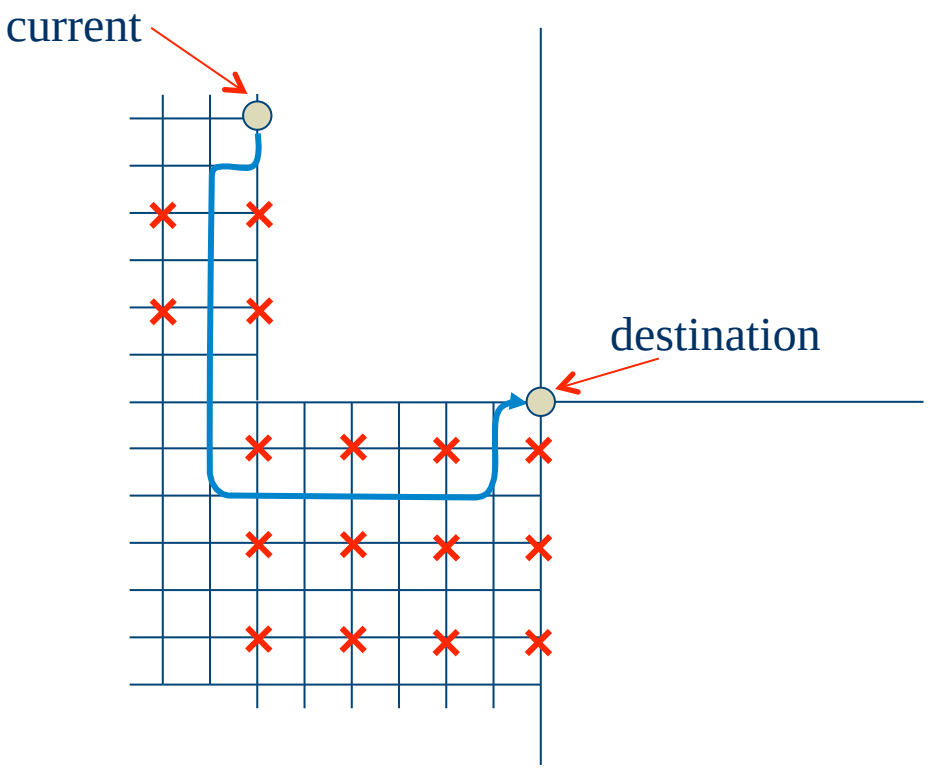
On-line approach

- If any two neighboring rows or columns do not have faults, multiple faults can be tolerated



On-line approach

- If any two neighboring rows or columns do not have faults, multiple faults can be tolerated



On-line approach

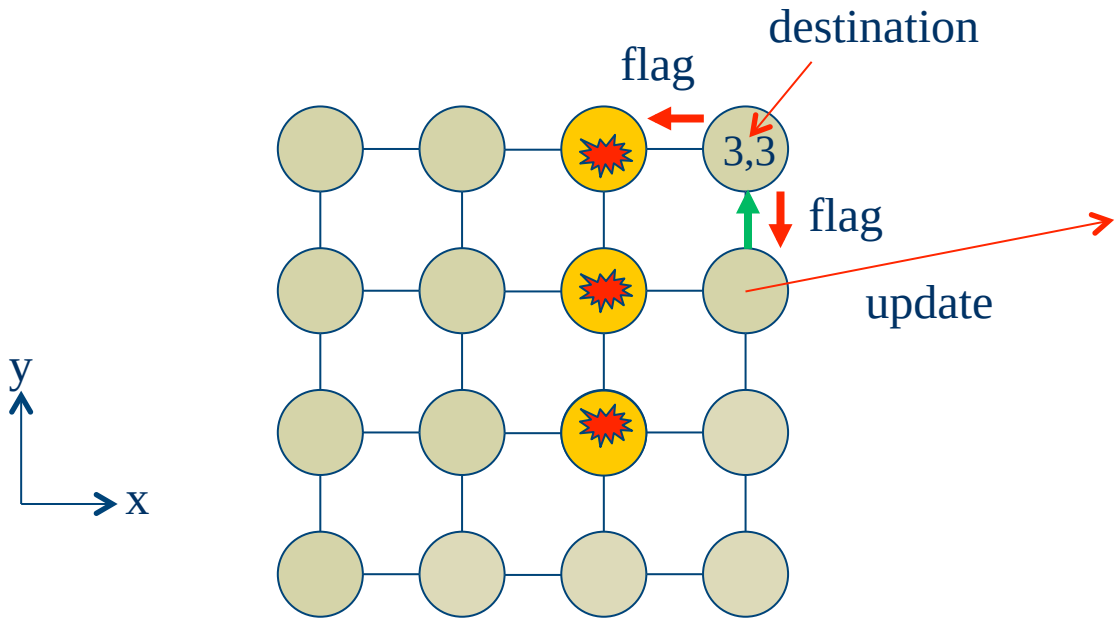
- Algorithms with different disallowed turns
  - different circuit delay
  - different fault-tolerance

Off-line approach

- Routing table based approach [Fick, et al.: DATE'09]
  - Off-line process
    - When a new fault has been detected, the following steps are repeated, for each destination node
      - Flag transmission
      - Routing entry update

Off-line approach

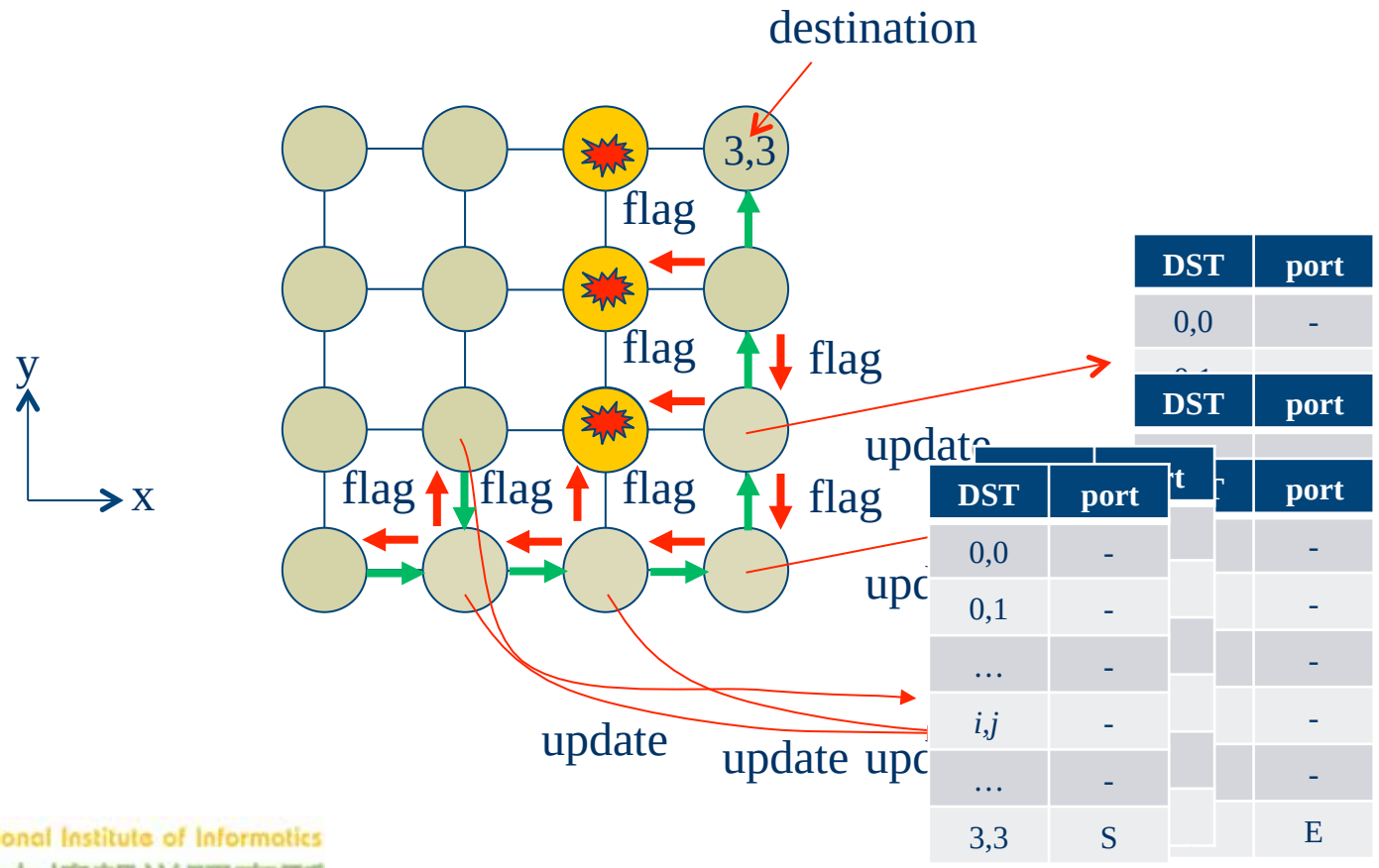
- A flag is initially sent from dest. node



DST	port
0,0	-
0,1	-
…	-
i,j	-
…	-
3,3	N

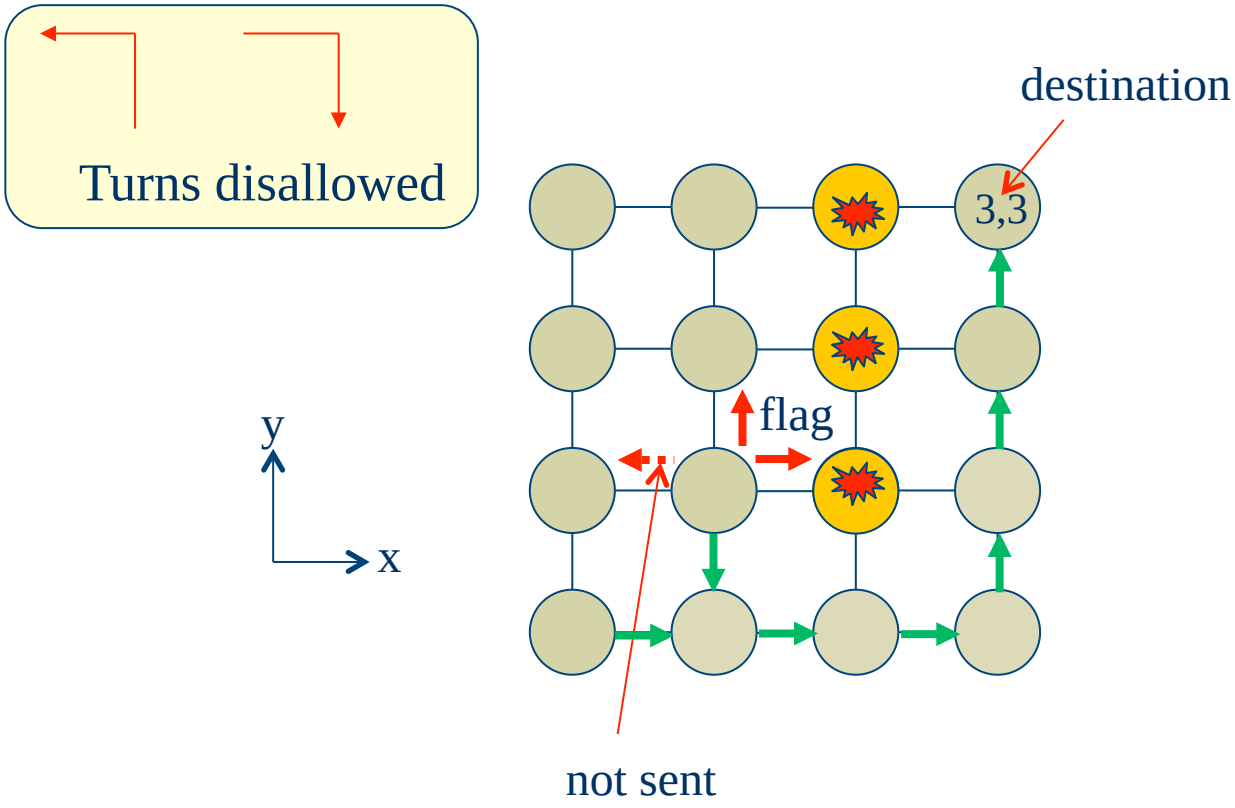
Off-line approach

- Routers with valid dest. entry send flags



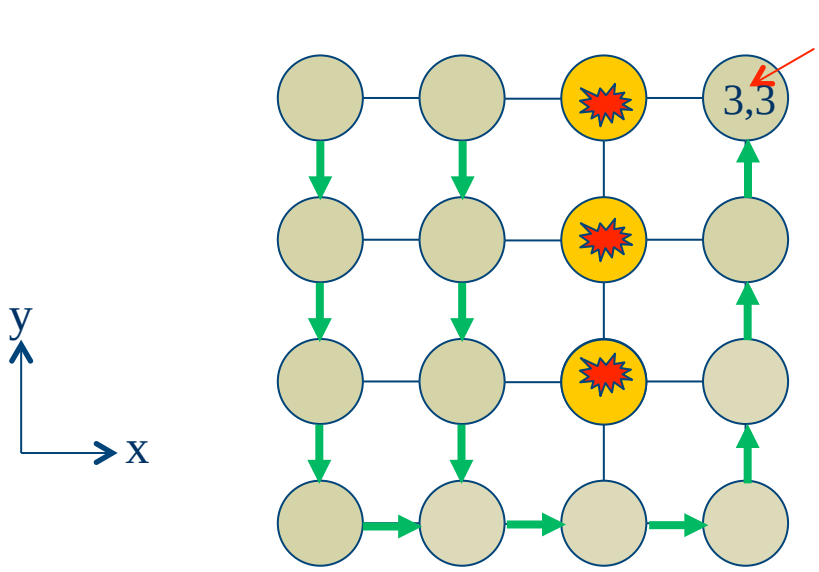
Off-line approach

- Flags for disallowed turns not be sent



Off-line approach

- All routing tables updated for (3,3)
  - Repeated for each dest. node



On-line algorithm cannot handle this fault-pattern

more fault-tolerant

Off-line approach

- Pros
  - Highly tolerant against multiple faults
- Cons
  - Off-line procedure needed for updating routing tables
    - It takes time
    - During the time, the new fault cannot be handled
  - Routing table based RC unit is relatively slow

Challenges in NoCs

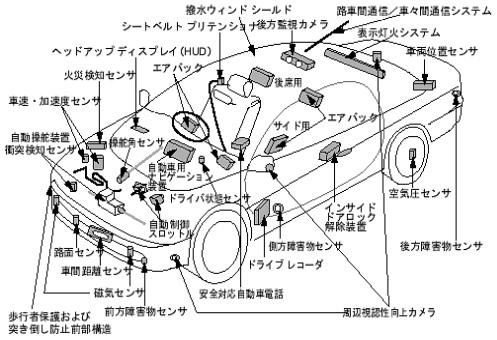
- How to use redundancy for dependability
  - Duplication, TMR, etc. can be dynamically and easily configured
- How to map applications onto a NoCs
  - Flexible, but overhead in packet switching
  - Task allocation on CPU/Accel. cores
- How to implement efficient NoCs
  - Advanced technologies
    - 3D Topologies, Optical interconnects, …
  - GALS (Globally Asynchronous Locally Synchronous)
    - fully asynchronous on-chip networks

Some of GALS systems

- LETI group
  - FAUST (ISSCC'07, JSSC'08)
    - ST 130nm, 20 nodes
    - SISO OFDM and CDMA
  - MAGALI (ISSCC'10, NoCS'09)
    - ST 65nm, 15 nodes
    - MIMO 4G Telecom
- Our group
  - Automotive application

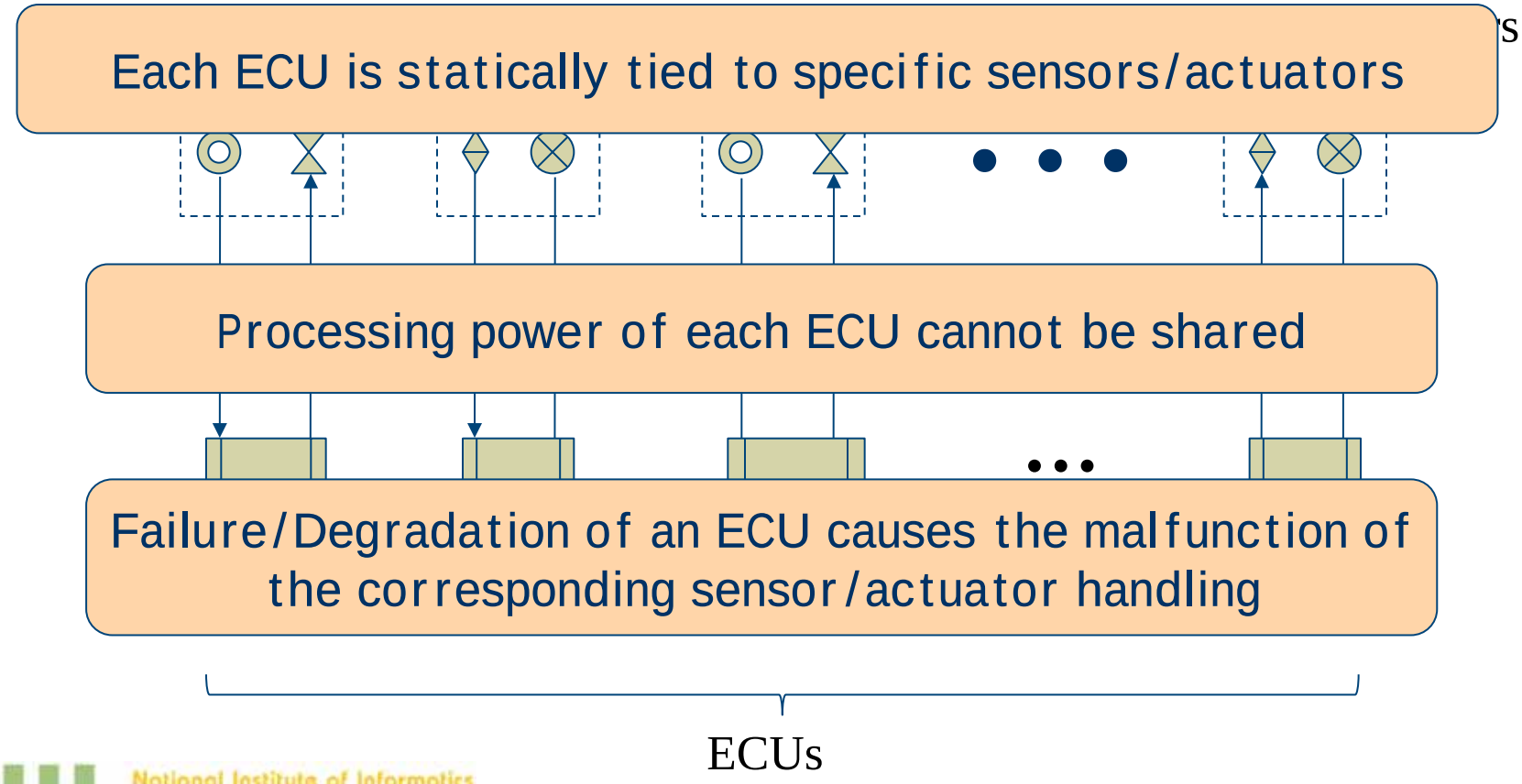
Background

- Demands for integrating more and more cores into a chip
  - Eg. Automotive electronic systems
    - More than 50 ECUs are used in an automobile
    - Many problems in connecting them
    - New approach
      - Centralized architecture where many ECUs are contained in one chip



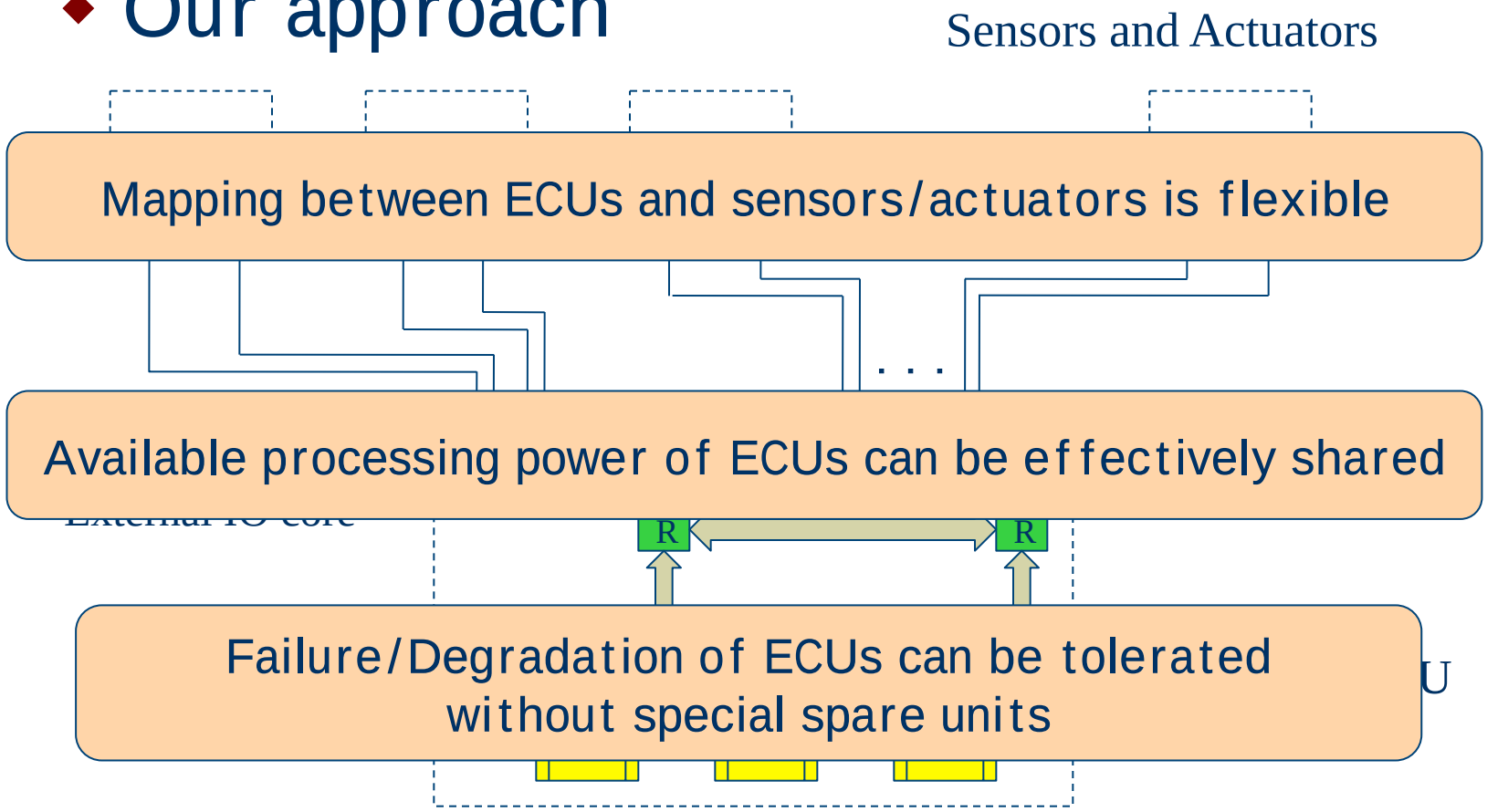
Ideas

- Current implementation



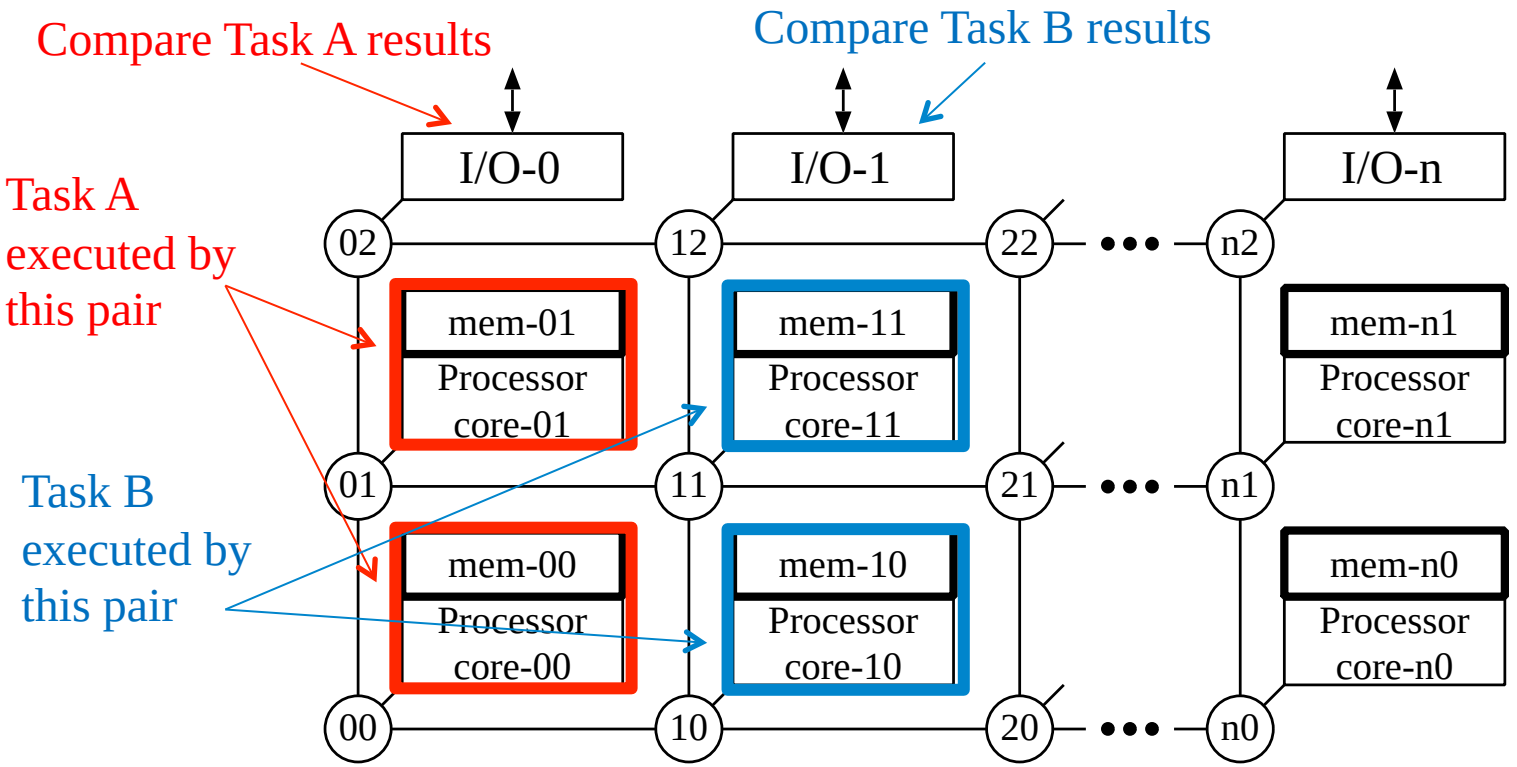
Ideas

- Our approach



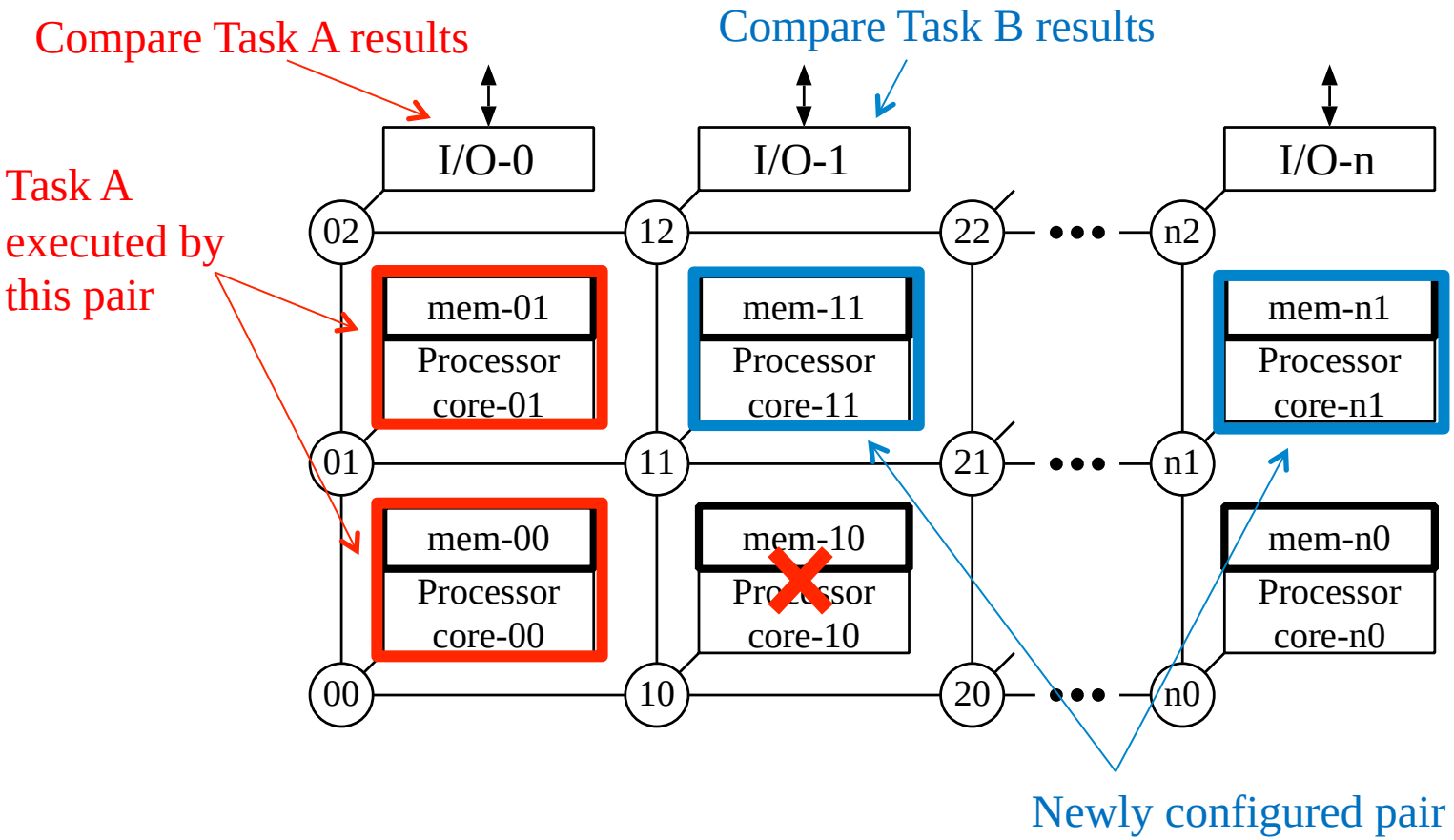
Ideas

- Flexible with fault detection & reconfiguration



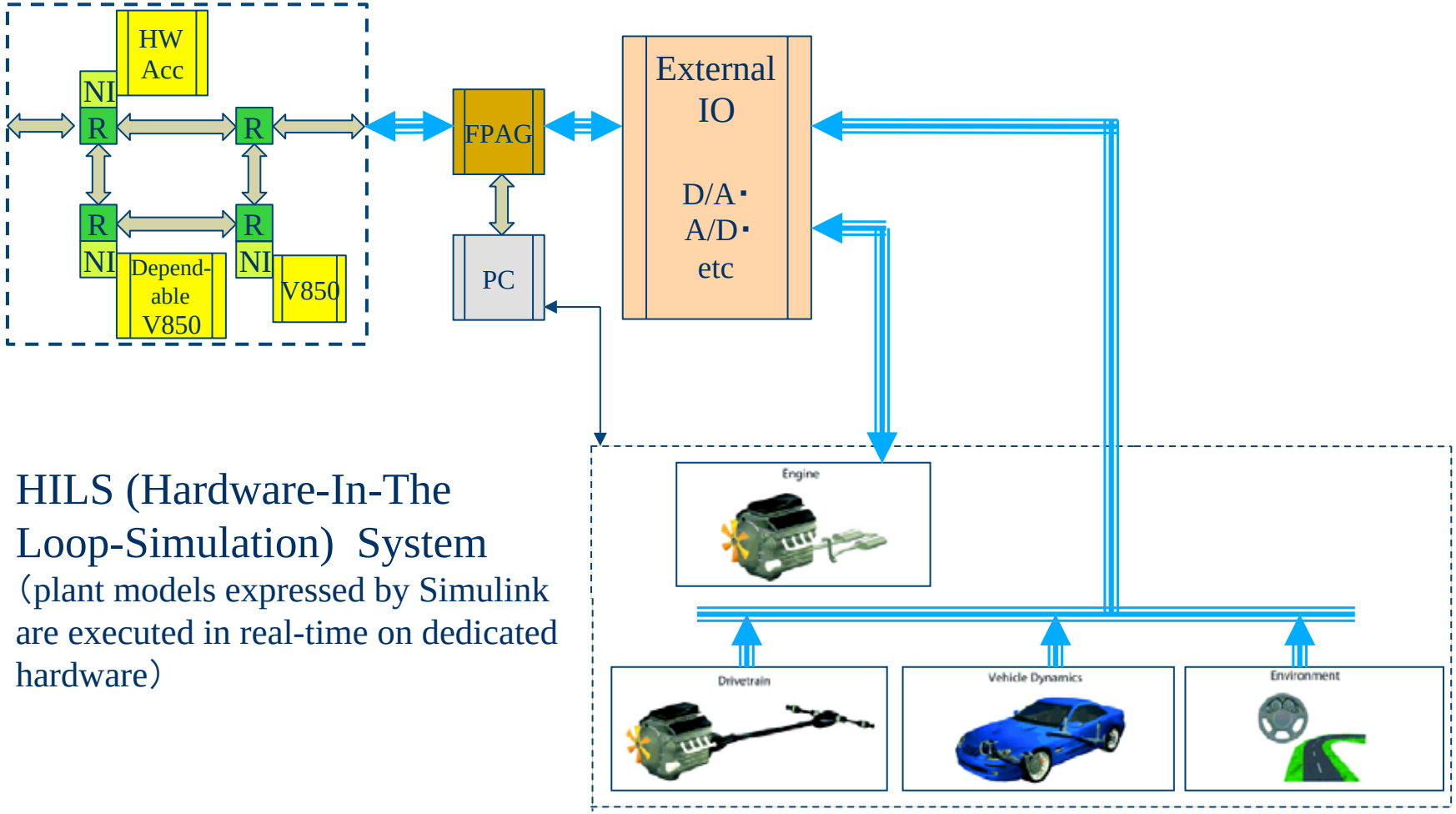
Ideas

- Flexible with fault detection & reconfiguration



First experimental platform

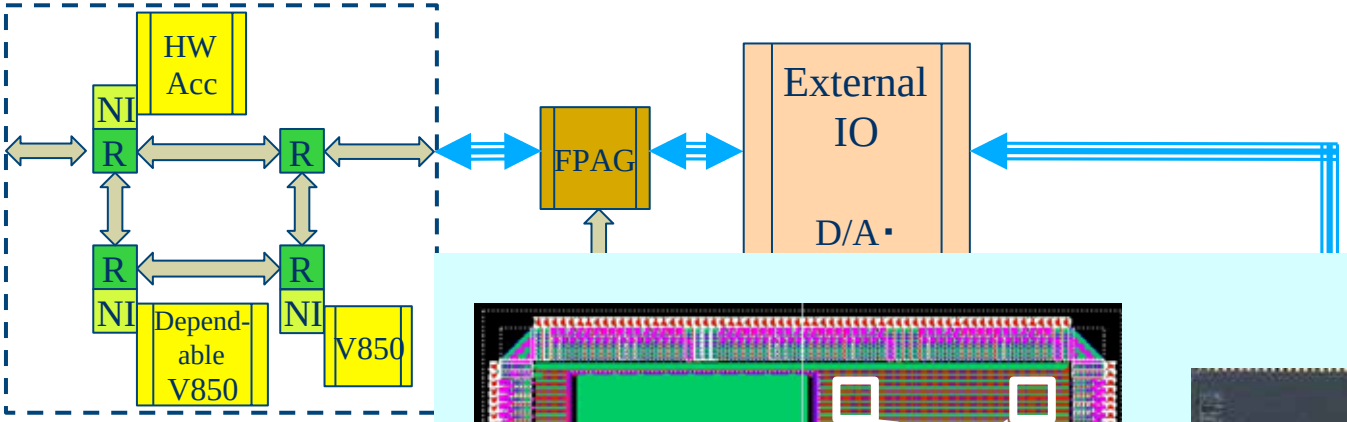
NoC chip



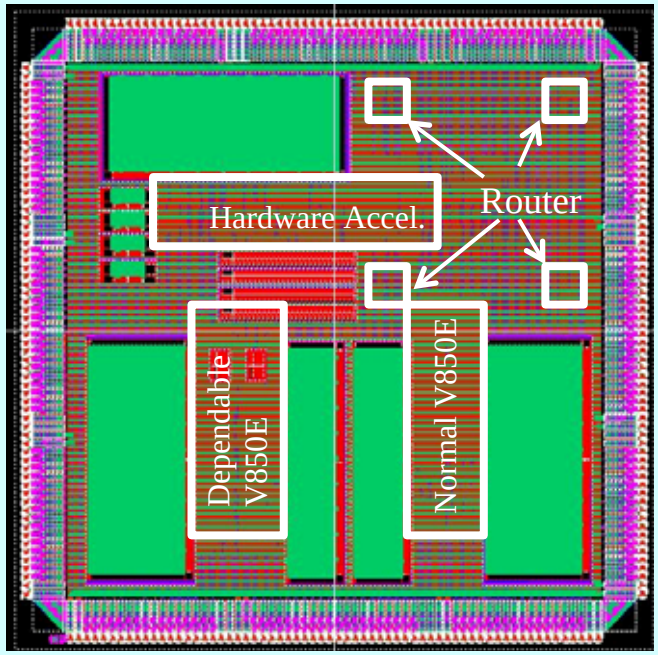
HILS (Hardware-In-The Loop-Simulation) System
（plant models expressed by Simulink are executed in real-time on dedicated hardware）

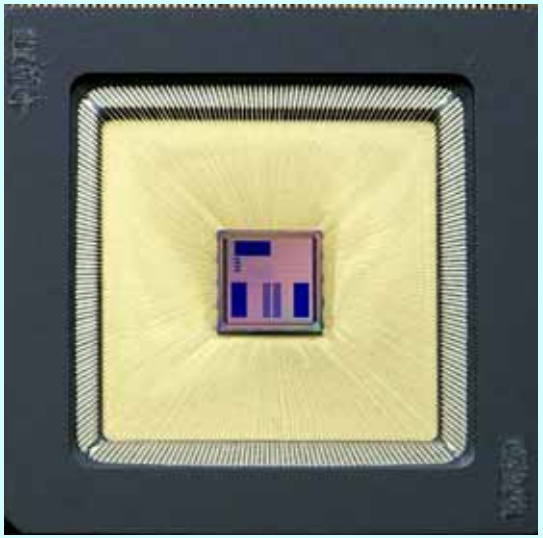
First experimental platform

NoC chip

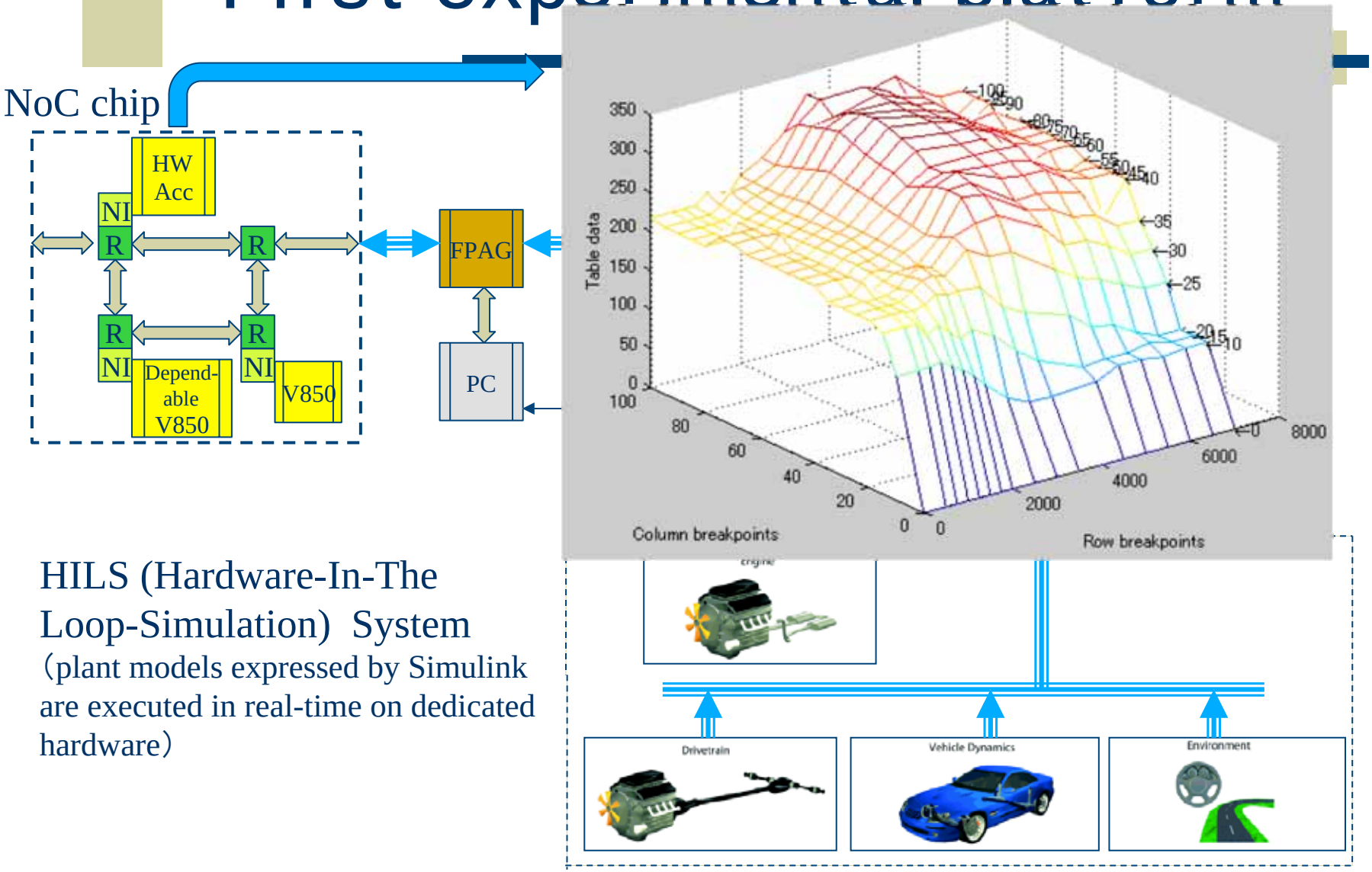


HILS (Hardware-In
Loop-Simulation) S
（plant models expressed
are executed in real-time
hardware）



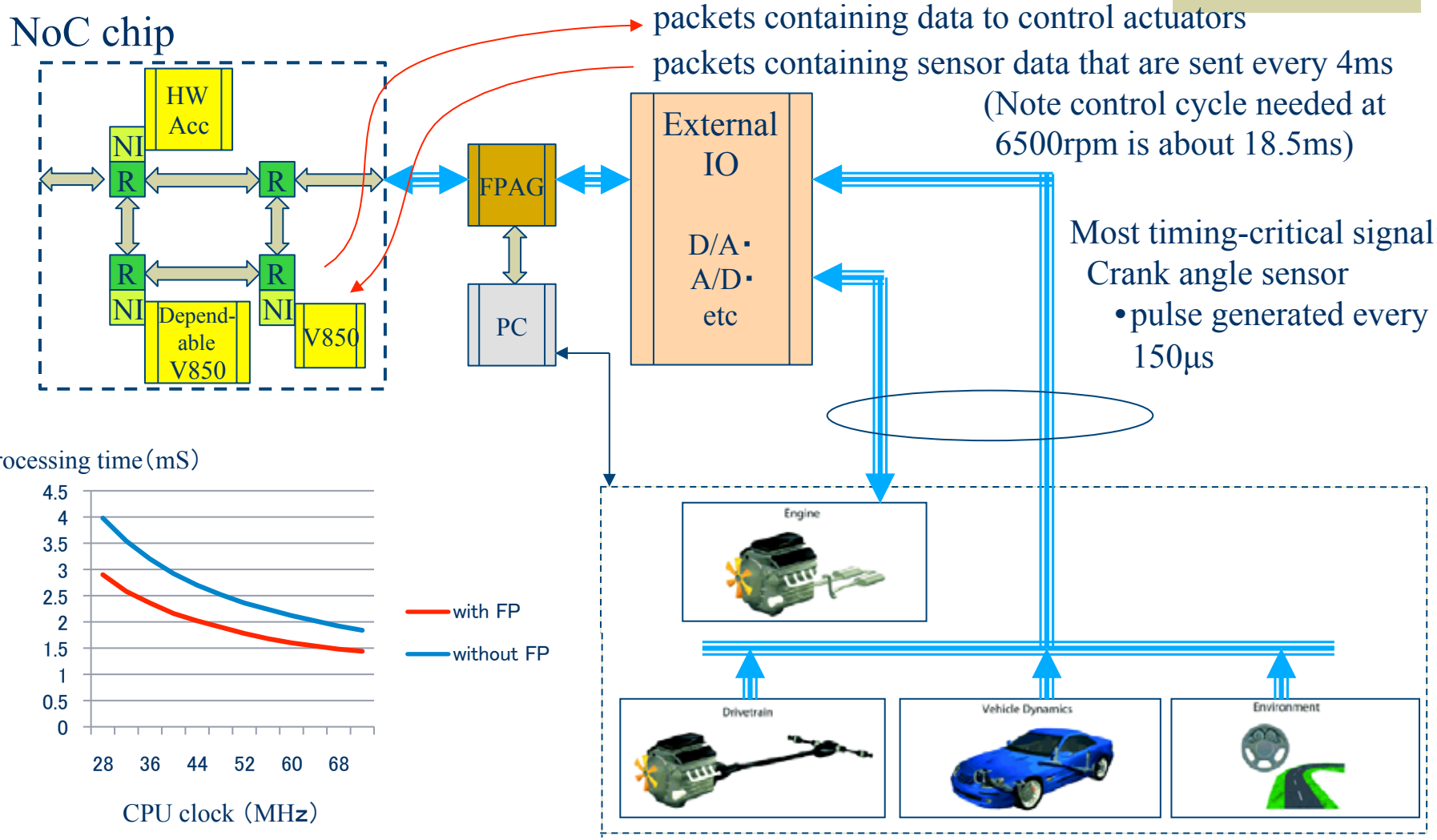


First experimental platform



HILS (Hardware-In-The Loop-Simulation) System
（plant models expressed by Simulink are executed in real-time on dedicated hardware）

First experimental platform



Conclusion

- Techniques for dependable routing
  - Implementations of Deadlock-free Routing
    - Requirements different from conventional computer networks
  - On-line approach
    - logic based
      - low overhead
      - limited fault-tolerance
  - Off-line approach
    - routing table based
      - high fault-tolerance
      - relatively high overhead

Acknowledgment

- Many thanks to
  - Masashi Imai, Hiroshi Saito, Atsushi Matsumoto, Yuichi Nakamura, Naoya Onizawa
  - Daihan Wang, Chammika Mannakkara, Vijay Holimath
  - JST (Japan Science and Technology Agency)