Multi-level Fault Tolerance in 2D and 3D Networks-on-Chip

Claudia Rusu
Vladimir Pasca
Lorena Anghel

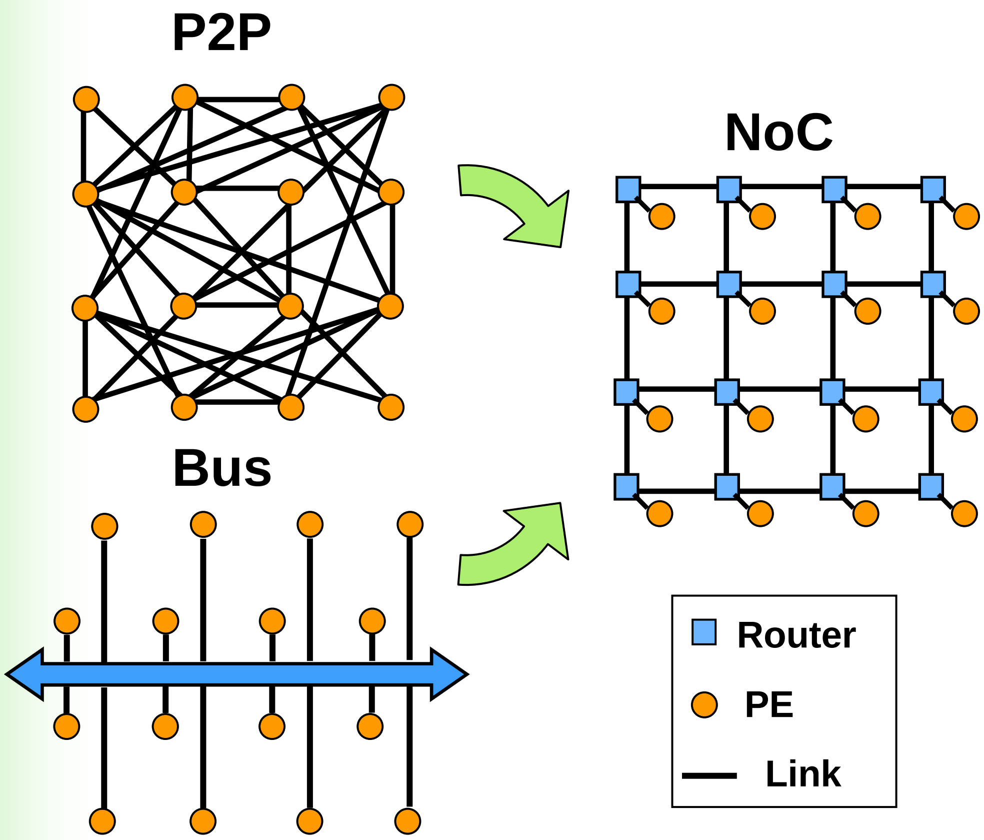
TIMA Laboratory
Grenoble, France

Outline

- **Introduction**
- Link Level
- Routing Level
- Application Level
- Conclusions

Network-on-Chip based Systems

- NoC vs. traditional connection systems



- NoC advantages
  - Efficient sharing of wires
  - Shorter design time, lower effort
  - Scalability

NoC QoS vs. Faults

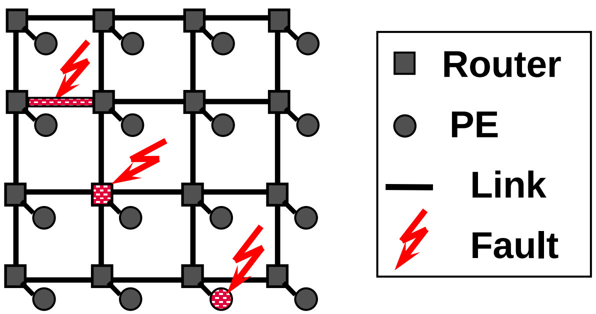
- Quality of service (QoS)
  - reliability, throughput, latency, bandwidth
- Unreliable signal transmission medium
  - timing and data errors
  - process variation induced logic and timing errors, crosstalks, electromagnetic interferences, radiations

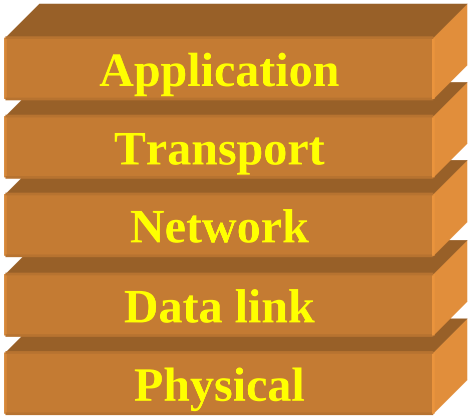
- Technology down scaling
- Increased system complexity

=> **Increased vulnerability to faults**

Fault Tolerance in NoCs

- Faults and Fault Tolerance
  - At different NoC components
    - Links
    - Routers
      - switching blocks
      - buffers
  - At different levels of the communication protocol stack

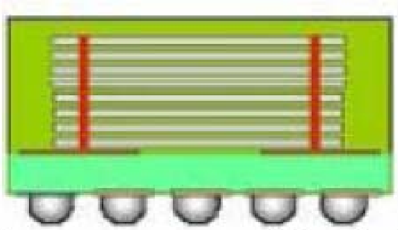


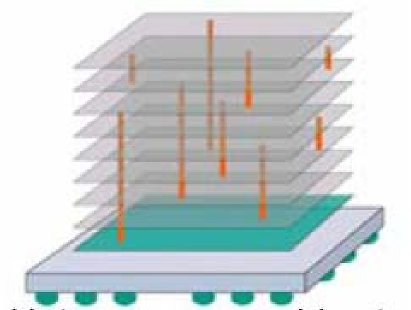


- Fault tolerant solutions
  - EDC, ECC, NMR
  - Fault-tolerant routing
  - Stochastic communication
  - Application checkpoint

3D NoCs

- Manufacturing
  - Stacked silicon active layers
  - Interlayer wires (TSV)
- Topology
  - Vertical and horizontal links
- Challenges
  - Manufacture high density vertical wires (TSV)
  - Place and route for heat removal


Homogeneous chip stack


Heterogeneous chip stack
(Source: Yole Development)

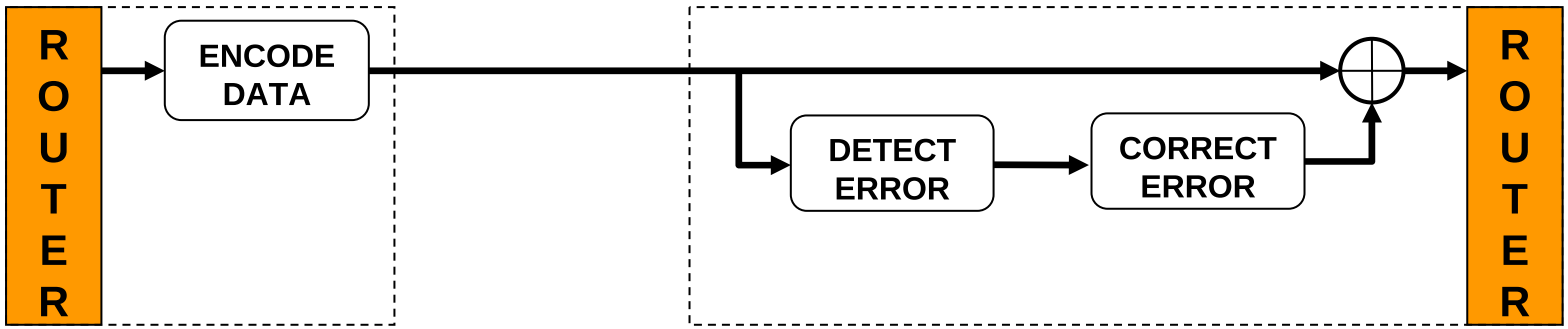
Outline

- Introduction
- **Link Level**
- Routing Level
- Application Level
- Conclusions

Error Control Schemes

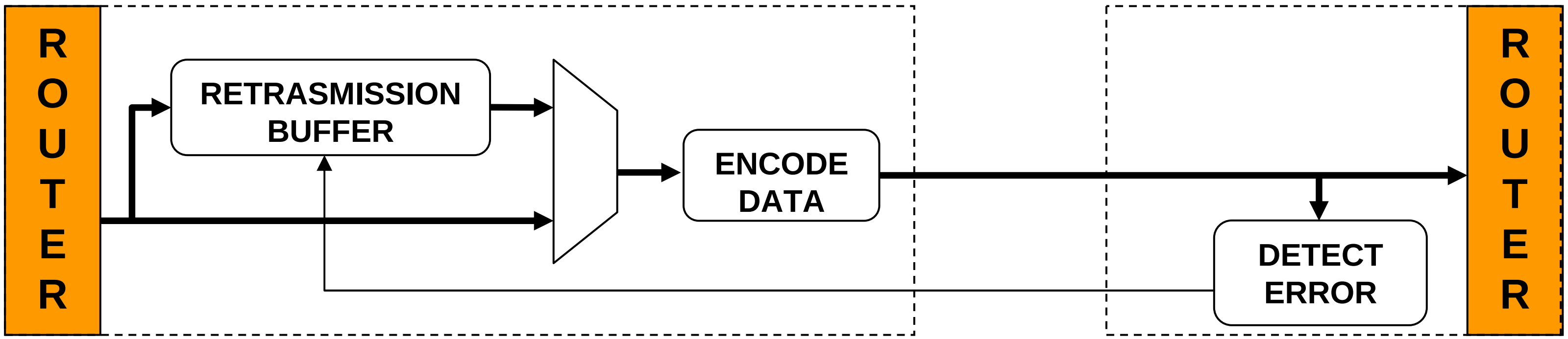
- **Fault Tolerance Mechanism**
  - Goal: Receiver always receives good (error free) data
- **Main classes**
  - **A. Forward Error Correction →** received data is always corrected (use only error correction codes: Hamming SEC/DED, Hsiao).
  - **B. Detection and Retransmission →** resend data when errors are detected (use error detection codes: Parity, Hamming).
  - **C. Hybrid →** correct as many errors as possible and alternatively request retransmission to improve error correction capability (use correction codes).
- **Flow Control**
  - **a. Switch to Switch →** error recovery on each intermediary link
  - **b. End to End →** error recovery at destination
- **Add several blocks to the initial scheme**
  - Encoder, Syndrome decoders and Correctors
  - Dedicated buffers for retransmission
  - Additional logic for the control signals of the scheme

a.A. Switch to Switch Error Correction



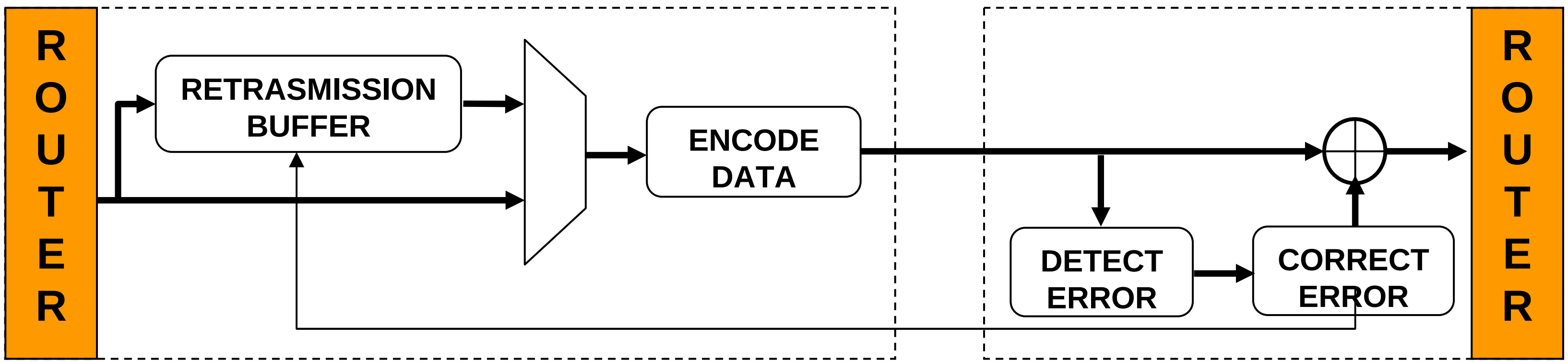
- Protect data lines with error correction codes
- Alternative protection (e.g. TMR) schemes for control signals
- Encode before sending flits on the communication lines (Flit)
- Check for errors on the received flits ➔ *code syndrome*
- Decode syndrome to identify errors ➔ *correction vector*
- Correct flit (received flit xor correction vector)

a.B. Switch to Switch Detection and Retransmission



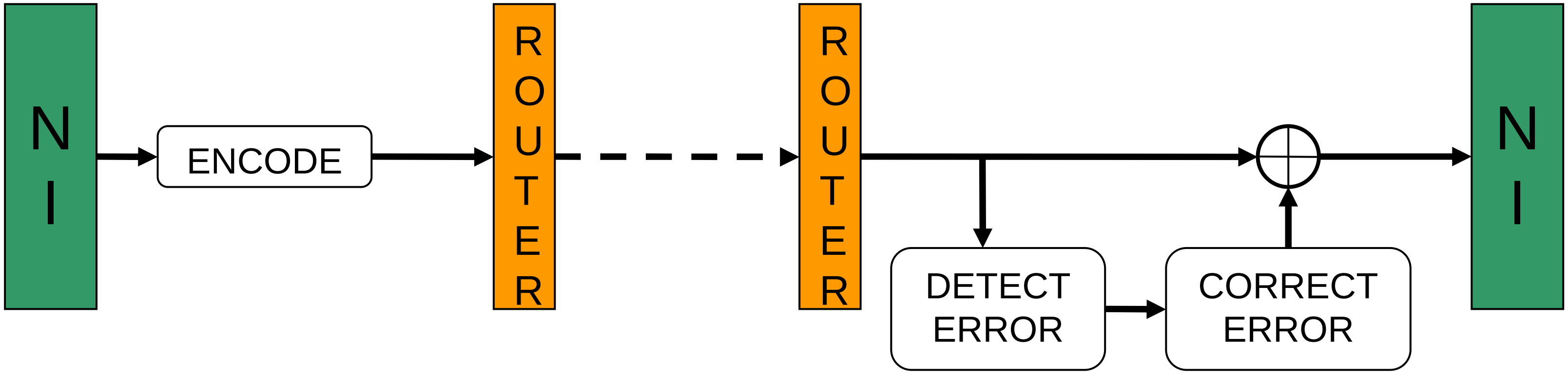
- Protect data lines with error detection codes
- Alternative protection (e.g. TMR) schemes for control signals
- Store transmitted flits in dedicated buffers
- Error detected during transmission → Retransmission from dedicated buffers
- Retransmission
  - Detect Error → Start retransmission
  - During Retransmission → Read Buffer & Disable link
  - Empty Buffer → Stop Retransmission & Enable link

a.C. Switch to Switch Hybrid Error Detection/Correction



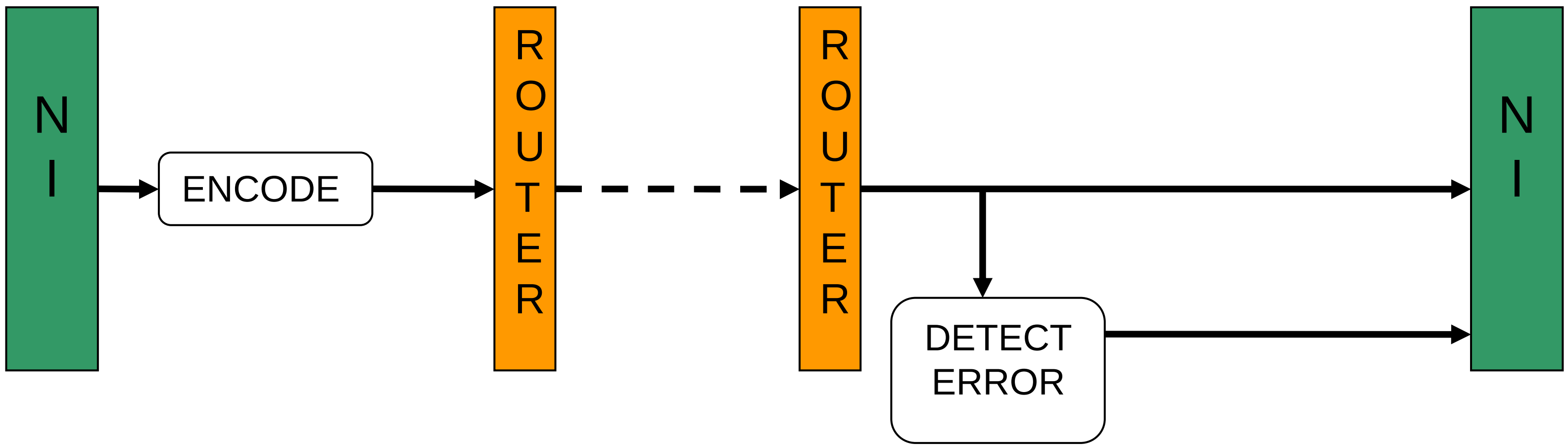
- Protect data lines with error detection codes
- Alternative protection (e.g. TMR) schemes for control sign
- Store transmitted flits in dedicated buffers
- Error detected during transmission → Decode syndrome and correct error
- Unable to correct detected errors → Retransmission from dedicated buffers
    - Uncorrectable Error → Start retransmission
    - During Retransmission → Read Buffer & Disable link
    - Empty Buffer → Stop Retransmission & Enable link

b.A. End to End Error Correction



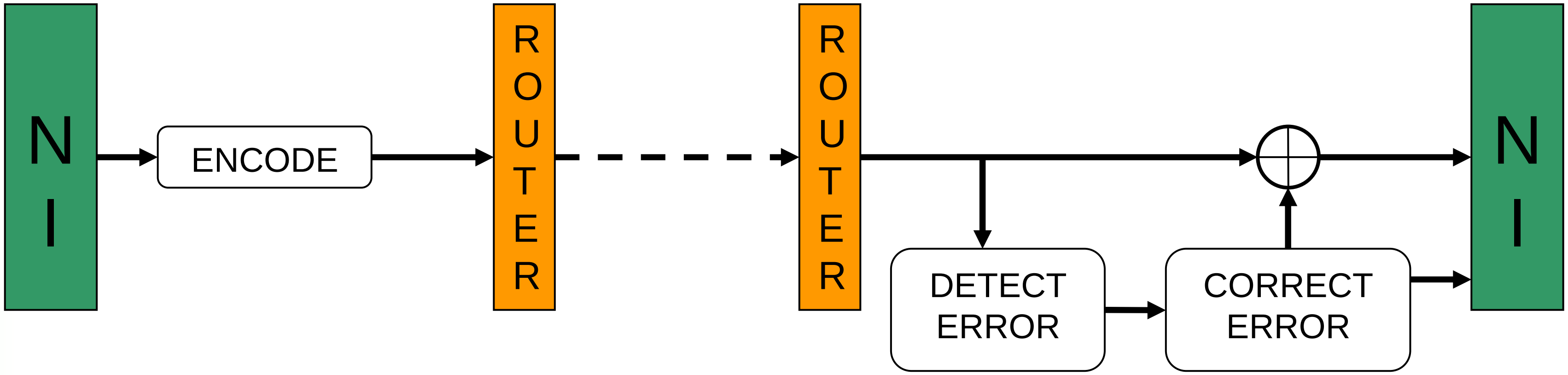
- Protect data lines with error detection codes
- Alternative protection (e.g. TMR) schemes for control signals
- Encode flits **before** transmission over the network (at the Network Interface)
- Intermediate buffers store encoded flits
- Flit arrives at destination
    - Detect and correct errors
    - Forward corrected data to the NI

b.B. End to End Detection and Retransmission



- Protect data lines with error detection codes
- Alternative protection (e.g. TMR) schemes for control signals
- Encode flits before transmission over the network (at the Network Interface)
- Transmission buffers in intermediate nodes store encoded flits
- Flit arrives at destination
  - Detect errors
  - Signal errors to NI
  - NI makes retransmission requests for detected errors

b.C. End to End Hybrid Error Detection / Correction



- Protect data lines with error detection codes
- Alternative protection (e.g. TMR) schemes for control signals
- Encode flits before transmission over the network (at the Network Interface)
- Transmission buffers of intermediate nodes store encoded flits
- When the flit arrives at destination
  - Detect and correct errors
  - Signal detected, but uncorrected errors to NI
  - NI makes a retransmission request for detected, but uncorrected errors

Implementation Overhead

- 3x3 Mesh Network on Chip
  - 9 Nodes and 21 Bi-directional Links
  - **Switch to Switch Error Control**
    - 42 Encoding, Detection, Correction Modules
    - 42 Retransmission Buffers
  - **End to End Switch Control**
    - 9 Encoding, Detection, Correction Modules
    - 9 Retransmission Buffers

Link Level - Conclusions

Criterion/scheme	**S2S**	**E2E**
Deal with cumulated errors along the path	*yes*	*no*
Area and power overhead	*higher*	*smaller*
Intermediate buffer width	*narrower*	*wider*

Latency	**S2S**	**E2E**
Retransmission	*3 clock cycles / link*	*depends on the path, traffic load etc.*
Correction	*1 clock cycle / link*	*1 clock cycle*

Area overhead	
Retransmission	*higher*
Correction	*lower*

Efficiency	**S2S**	**E2E**
Retransmission	*- Many errors uniformly distributed along the path*	*- Many errors not uniformly distributed along the path*
Correction	*- Few errors uniformly distributed along the path*	*- Few errors not uniformly distributed along the path*

Outline

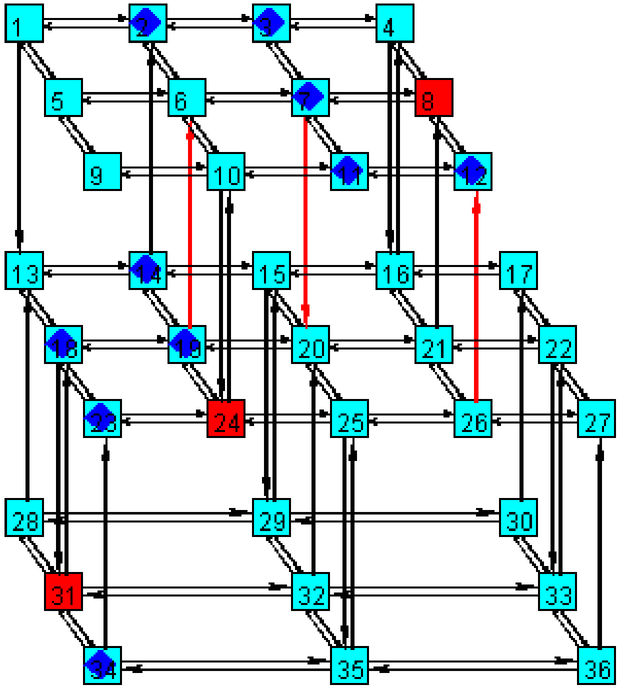
- Introduction
- Link Level
- **Routing Level**
- Application Level
- Conclusions

2D FT Routing

- Virtual networks
  - N Last and S Last virtual networks
    - if Dst is N of Crt, use S Last
    - if Dst is S of Crt, use N Last
  - The VN changes on the path when the location (N or S) of the Dst from the Crt changes
- Numbering of nodes (only increasing or only decreasing) -> avoid deadlocks

3D Fault-tolerant Routing

- Extends the 2D FT routing
  - Folded 2D FT routing
- Assumptions
  - All layers are reachable
  - All nodes in the same layer are reachable

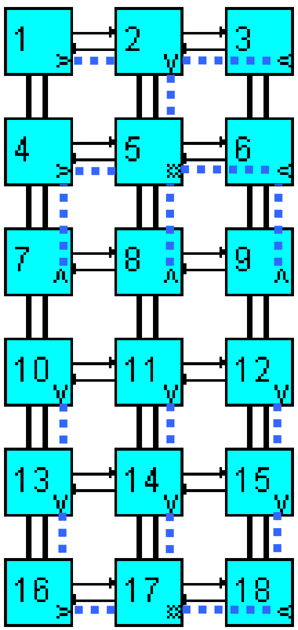


Algorithm

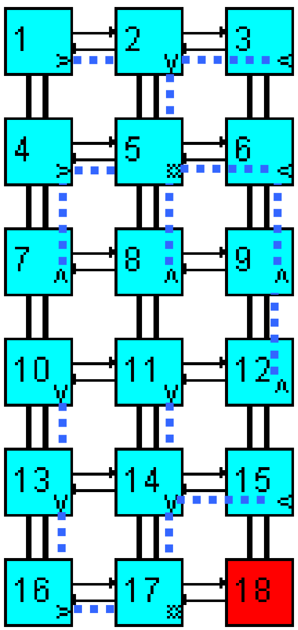
```
while Crt layer <> Dst layer
    if Dst layer < Crt layer
        then V = Up
        else V = Down
    end if
    2D_FT_routing to V
    route to next layer
end while
2D_FT_routing in Dst layer
```

Determining V

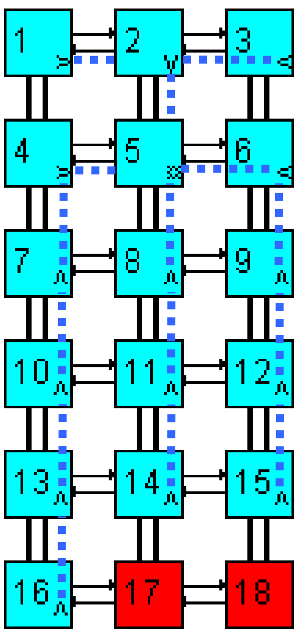
- By gossiping
- Reconfiguration in case of failure



Initial



Node failure (18)



V failure (17)

Routing Level - Conclusions

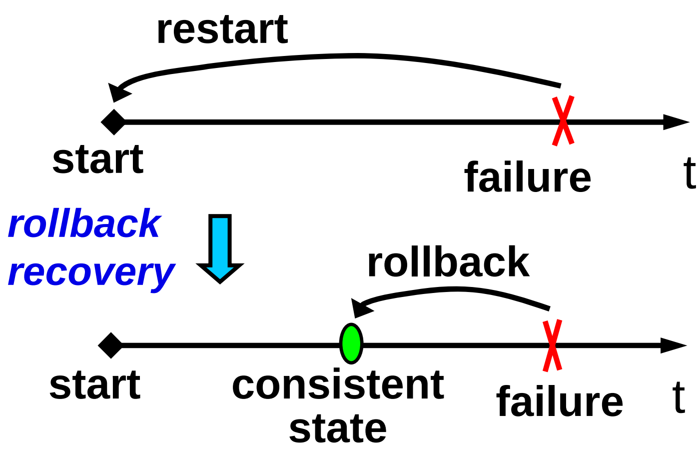
- Fault-tolerant routing in 3D NoC
  - Deadlock free
  - Tolerate multiple node and link failures
  - Topology independent
    - Heterogeneous stacking
  - Compatible with any 2D FT routing
    - reusability of specific 2D FT routing of each layer

Outline

- Introduction
- Link Level
- Routing Level
- **Application Level**
- Conclusions

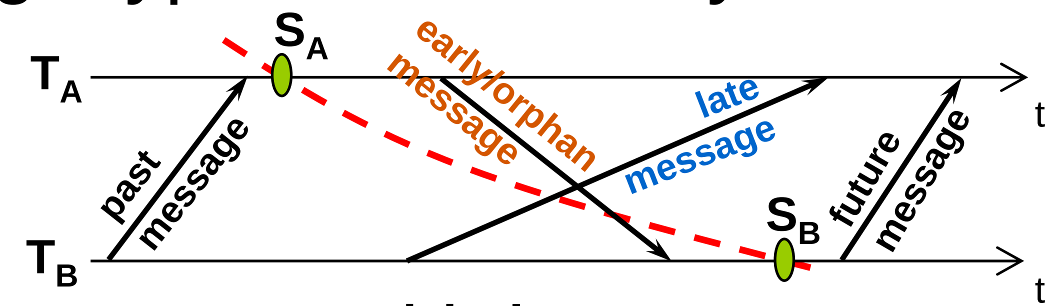
Checkpoint and Rollback Recovery. Principle

- No failure tolerance
  - – Failure => Restart
- Checkpoint and rollback recovery
  - – Failure => Resume from a more recent state
  - – Principle
    - ❖ Failure-free
      - – periodically store states on stable storage
    - ❖ Failure
      - – rollback to the last consistent stored state

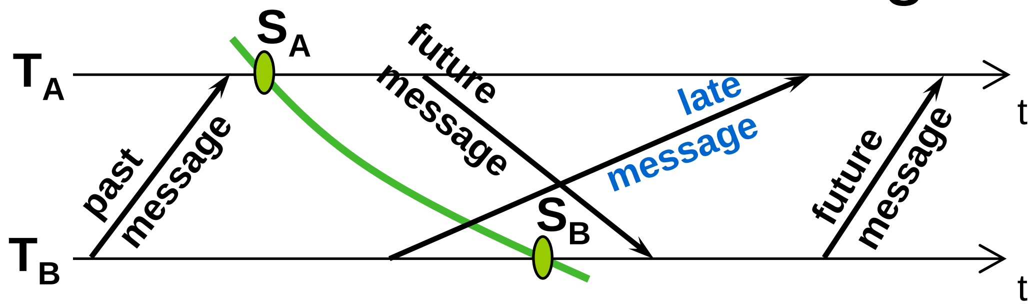


Checkpoint and Rollback Recovery. Consistent State of Several Tasks

- Message types vs. recovery line



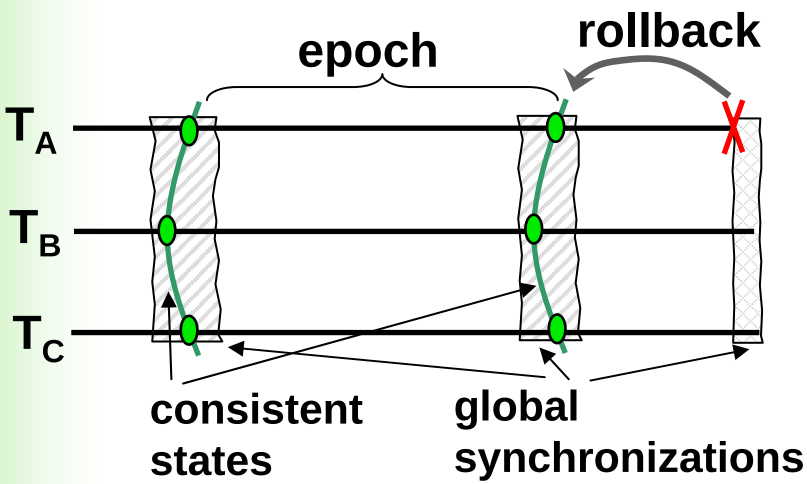
- Consistent state with late messages



  - early messages are avoided
  - late messages are to be replayed after rollback

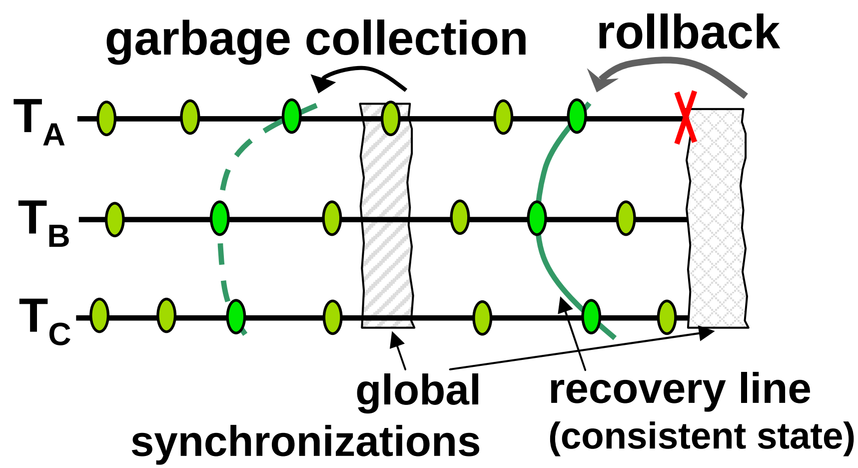
Coordinated vs. Uncoordinated

- Coordinated



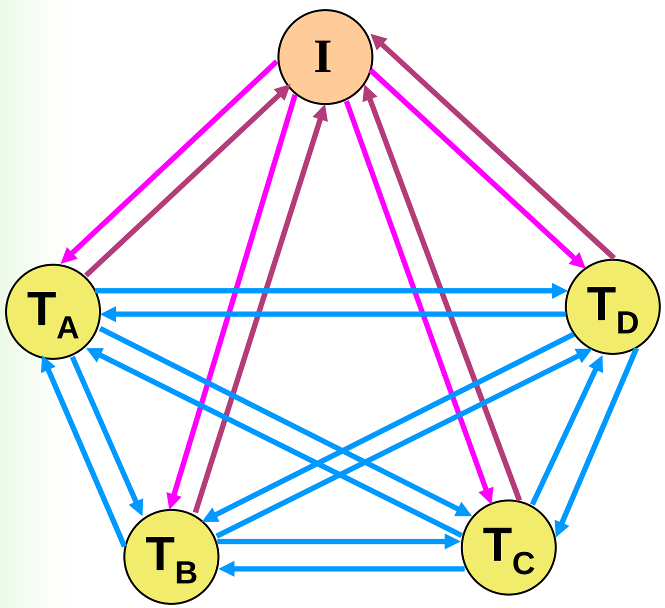
- Failure-free
  - synchronization –> consistent state
- Failure
  - rollback to the last consistent state

- Uncoordinated



- Failure-free
  - individual task checkpoints
- Failure
  - synchronization –> consistent state
  - rollback to consistent state

Blocking and Non-blocking Coordinated Checkpointing




```
Initiator

- broadcast
  CK_REQ

- when CK_TAKEN
  received from
  all tasks
  - validate
    global
    checkpoint
```

```
Non-initiator
(blocking or not)

- on CK_REQ receipt
  - broadcast
    CK_START
  - when CK_START
    received from
    all tasks
    - take local
      checkpoint
    - send to
      initiator
      CK_TAKEN
```

- Messages in NoC during checkpointing
  - Blocking
    - synchronization messages
  - Non-blocking
    - synchronization messages
    - application messages

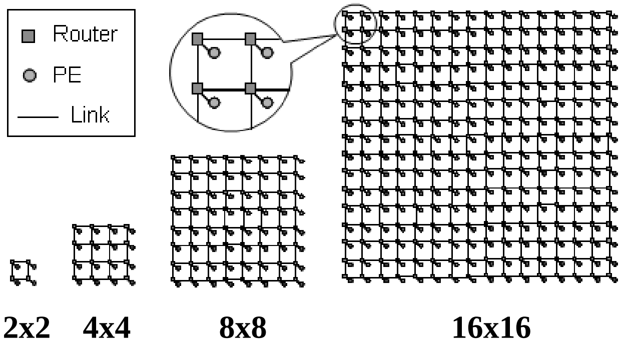
- Unique coordination
  - Allows blocking of a task set and the non-blocking of another

Coordinated Checkpointing Scalability Improvements

- Coordinated checkpointing improvements
  - – Reduce number of broadcasts
  - – Smart broadcast

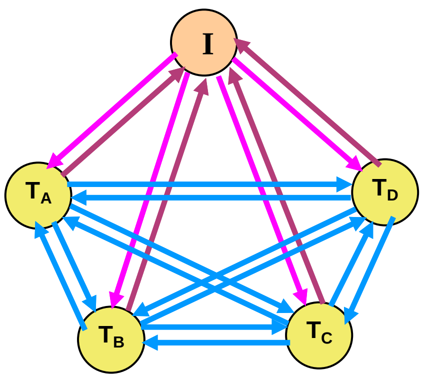
  => Significant reduction of checkpointing overhead and latency, especially for higher NoC sizes

  => Scalability improvement



Protocol Improvement

- No optimization

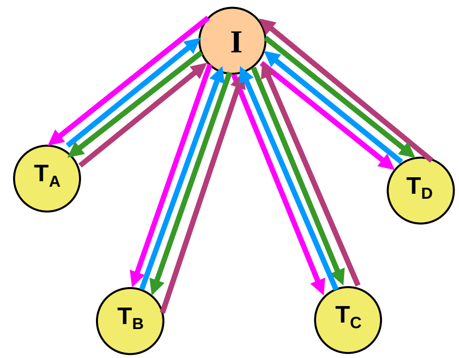


n nodes

- CK_REQ n
- CK_START $n*(n-1)$
- CK_TAKEN n

} $O(n^2)$

- Reduced number of broadcasts



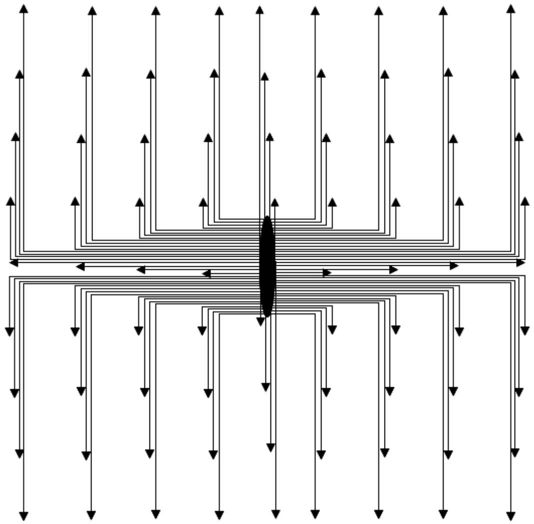
n nodes

- CK_REQ n
- CK_START n
- CK_START_ALL n
- CK_TAKEN n

} $O(n)$

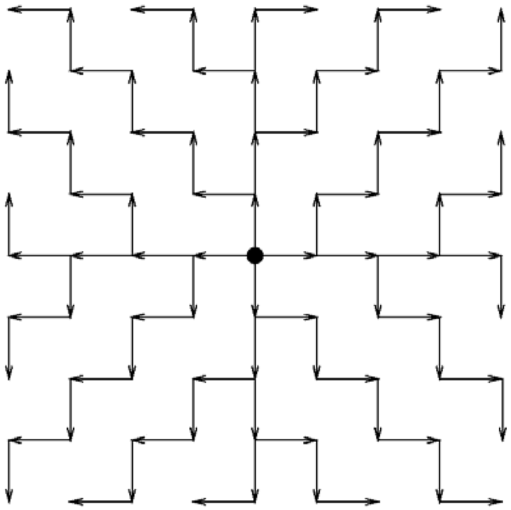
Smart Broadcast

- Classical broadcast



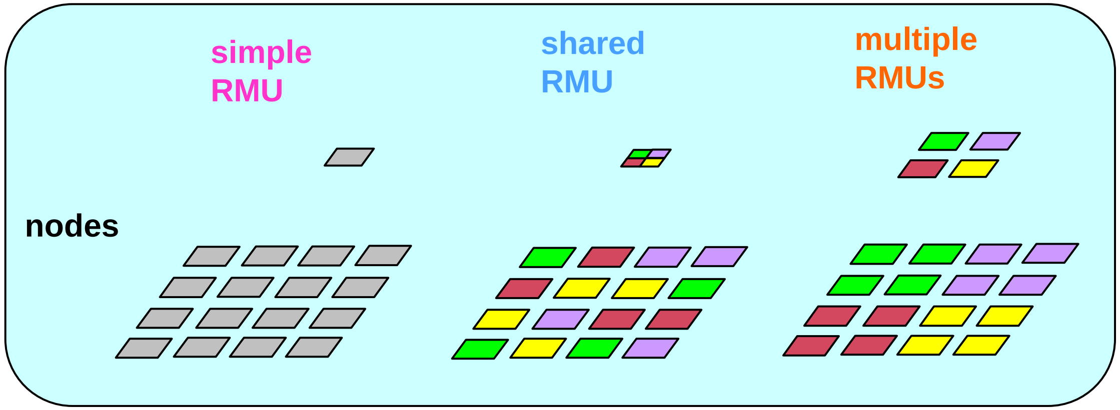
  - simplicity
  - redundancy
  - unequal link utilization; high utilization of certain links –> bottlenecks

- Smart broadcast



  - uniform link utilization
  - no redundancy
  - no bottlenecks
  - adjust routing

Optimized Configurations



- Application partitioning
  - Reduce number of participating nodes
  - Reduce total distance among nodes
- Results
  - Partition configurations assure lower overhead and lower latency

Application Level - Conclusions

- Coordinated vs. Uncoordinated
  - Lower overhead induced by coordinated checkpointing
  - Traffic and failure rate influence
    - Lower traffic load and lower failure rate
      - Coordinated checkpointing is more efficient
    - Higher traffic load and/or higher failure rate
      - Uncoordinated checkpointing becomes relevant
- Blocking vs. Non-blocking
  - Checkpointing duration increases with the traffic load
    - Non-blocking: significantly
    - Blocking: lesser
  - Application latency increases with the traffic load and the failure rate
    - Non-blocking: significantly
    - Blocking: lesser
  - For higher traffic loads and higher failure rates, the blocking approach becomes indispensable
  - Unique blocking and non-blocking protocol
- Scalability Improvements
- Optimized Configurations

Fault-tolerance and Reconfiguration

- Link Level
  - Reconfigurable error correction scheme
  - Self-healing capabilities
- Routing Level
  - Dynamic reconfiguration of the path in case of failures
- Application Level
  - Node failure
    - Use checkpoint and reallocate task

Outline

- Introduction
- Link Level
- Routing Level
- Application Level
- **Conclusions**

Conclusions

- Multi-level fault-tolerant mechanisms
  - link, routing, application
- Complementary mechanisms
- Efficiency – cost trade-off
  - Error rate
  - Application characteristics
  - Required QoS
  - Overhead