The 1st IEEE South-East Asia Workshop on Circuits and Systems

# Fault-Tolerant Architectures and Algorithms for 3D-Network-on-Chips

Khanh N. Dang, Ph.D.

khanh.n.dang@vnu.edu.vn

SISLAB, University of Engineering and Technology

Vietnam National University Hanoi (VNU)



November 7th, 2017

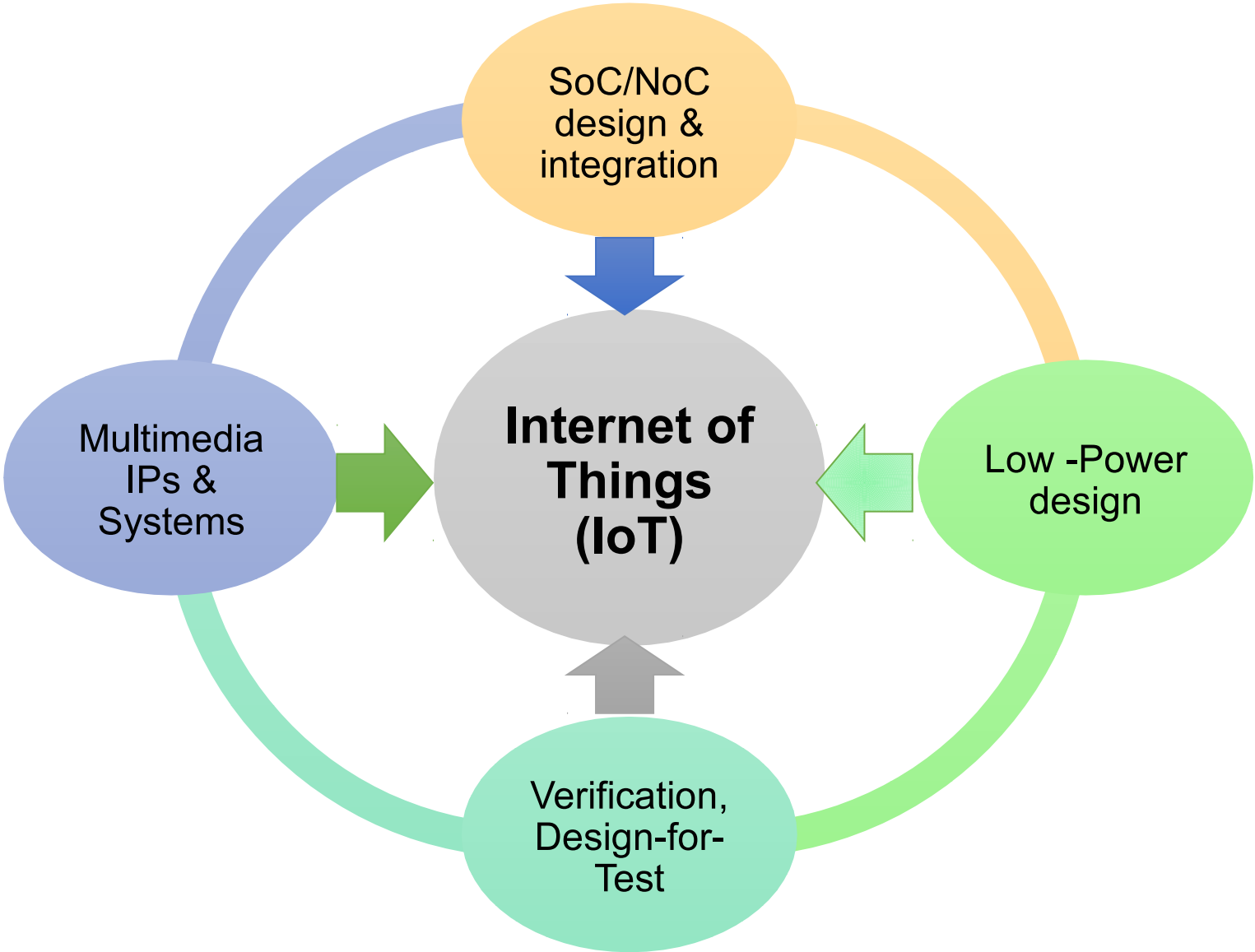
# About SISLAB

One of 7 key R&D laboratories of Vietnam National University, Hanoi (VNU), Vietnam.
Members: 6 PhDs, 8 PhD students, 3 MSc students, and BSc students.

**Campus 1:** University of Engineering and Technology, Vietnam National University Hanoi,
2.1 E4, 144 Xuan Thuy, Cau Giay, Hanoi
**Campus 2:** 2th Floor, High Tech Business Incubator Center (HBI), Hoa Lac High Tech Park, Hanoi

**Contact**
Tel: 04-3754-9664
Fax: 04-3754-7460
Website: *http://sis.uet.vnu.edu.vn*
Email: *sislab@vnu.edu.vn*

# Research and Development @ SISLAB

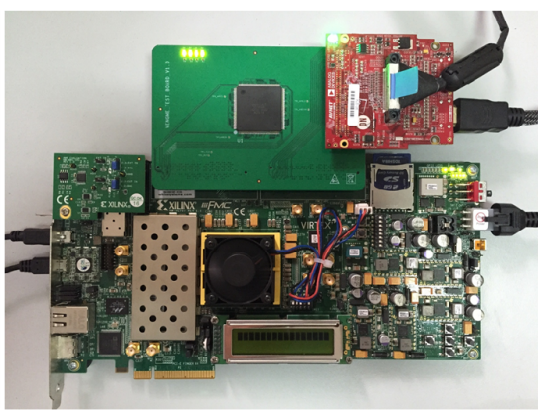
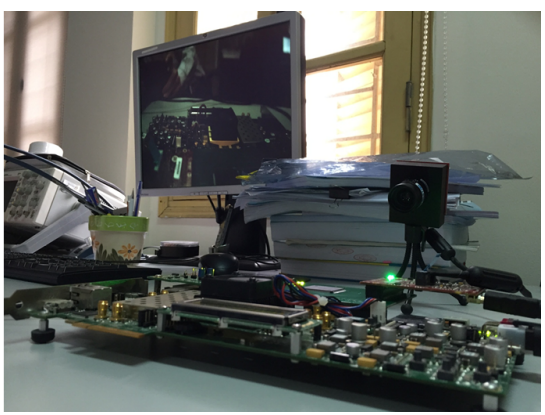


# Products

- CoMoSy: a SoC platform.
- System-C based NoC simulator.
- VENGME: H.264 encoder chip.
- ...

Notable project: VENGME - a H.264/AVC encoder chip

- Technology: Global Foundry CMOS 130nm
- Area: 16mm$^2$
- Complexity: 2 M gates ( 8 M transistors)
- Power consumption: 53mW
- Operating frequency: 100MHz
- Voltage: 1.2 Volt
- Package: QFP256

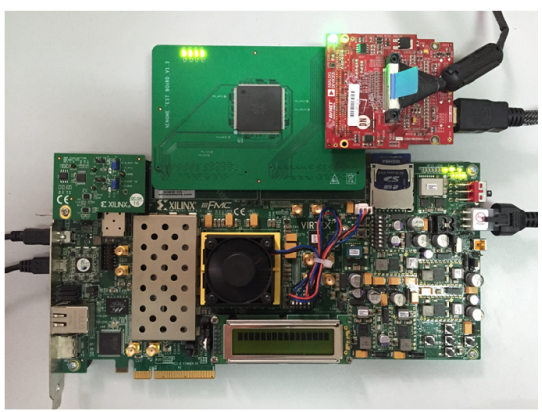
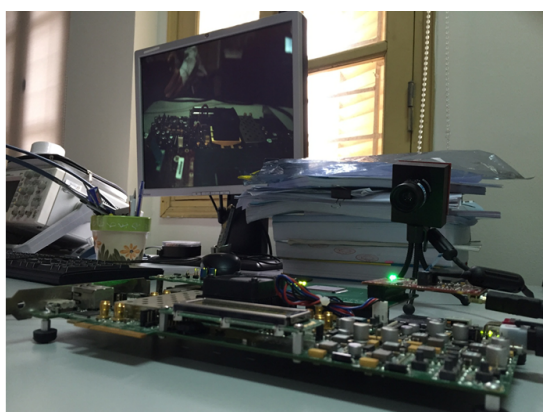
# Products

- CoMoSy: a SoC platform.
- System-C based NoC simulator.
- VENGME: H.264 encoder chip.
- ...

Notable project: VENGME - a H.264/AVC encoder chip

For more details/products/publications:
http://sis.uet.vnu.edu.vn
http://www.uet.vnu.edu.vn/~tutx/

- Voltage: 1.2 Volt
- Package: QFP256

# Table of Contents

# Table of Contents

# Era of Multi/Many-core processing

Constant increase of the number of cores → *multi/many-core processing*.

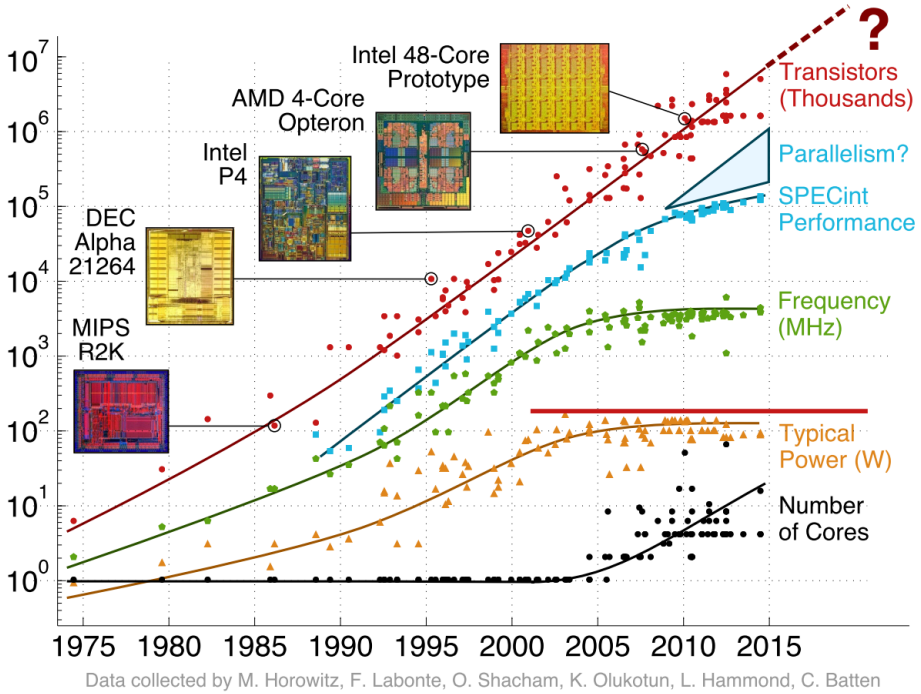



Figure 1: Integrated Circuit Scaling [1].

Interconnect delay becomes the major challenge.

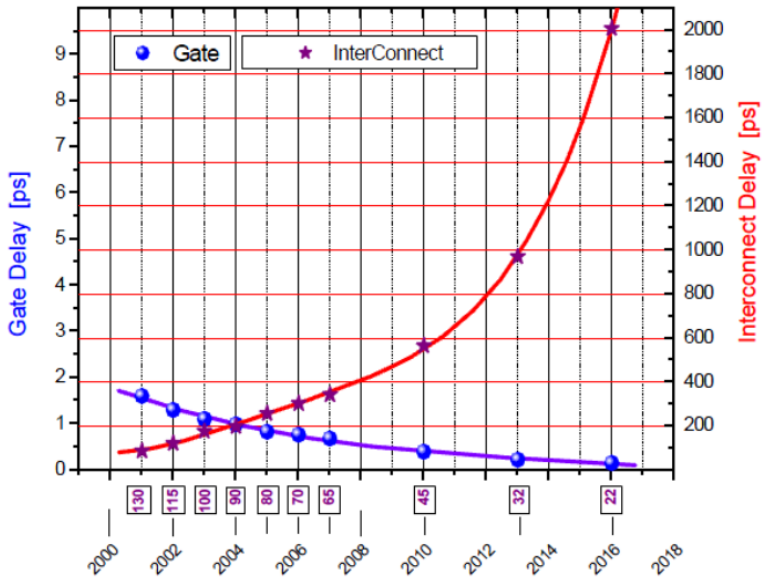



Figure 2: Gate and interconnect delay overtime [2].

To keep up with demands on computational power, we need to:

- Increase *parallelism*.
- Provide an efficient and *low-power interconnect* infrastructure to achieve better *scalability*, *bandwidth*, and *reliability*.

# Design Challenges of Multi/Many-core systems

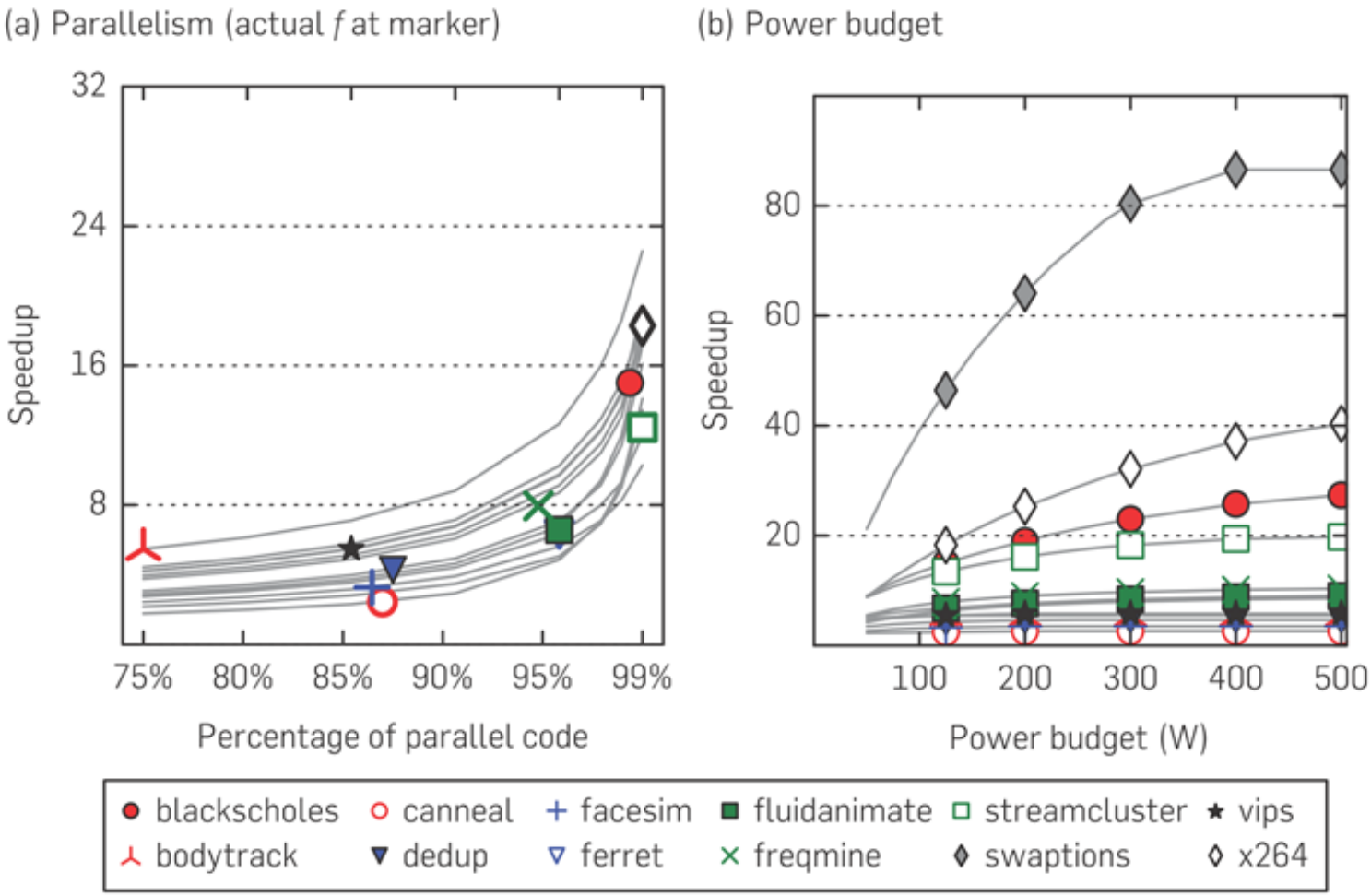



Figure 3: Challenge on *parallelism* and *power budget* on application speedup at 8nm [3].

# Emerging Interconnect Paradigms

- **RF/Wireless**: Replacing on-chip wires by integrated on-chip antennas to communicate with electromagnetic waves, in free space or guided medium.
- **Carbone Nanotube**: Using of carbon-based interconnect to replace the Cu/low-k technology.
- **Photonic**: Using photon instead of electron to transfer data.
- **Network-on-Chips**: Electronic networks were designed on a chip to allow parallel data transmission.
- **3D Integration**: Stacking multiple layers to obtain smaller footprints and shorter intra-layers interconnects.

# 3D Integration Technology

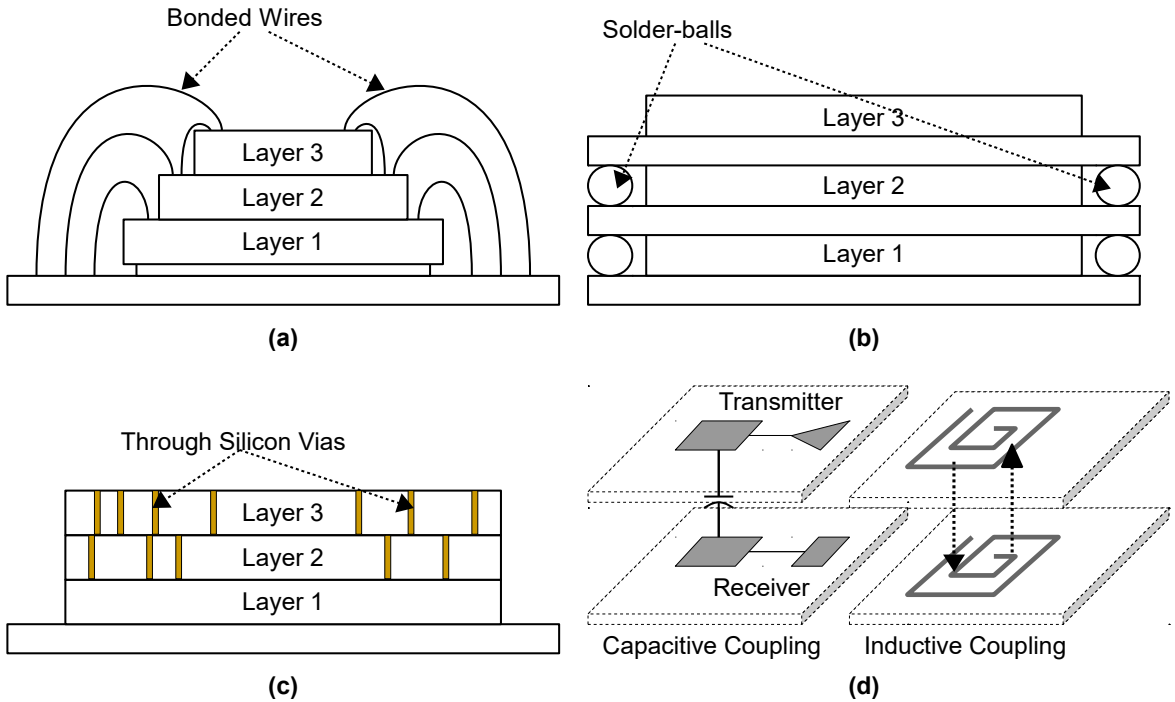
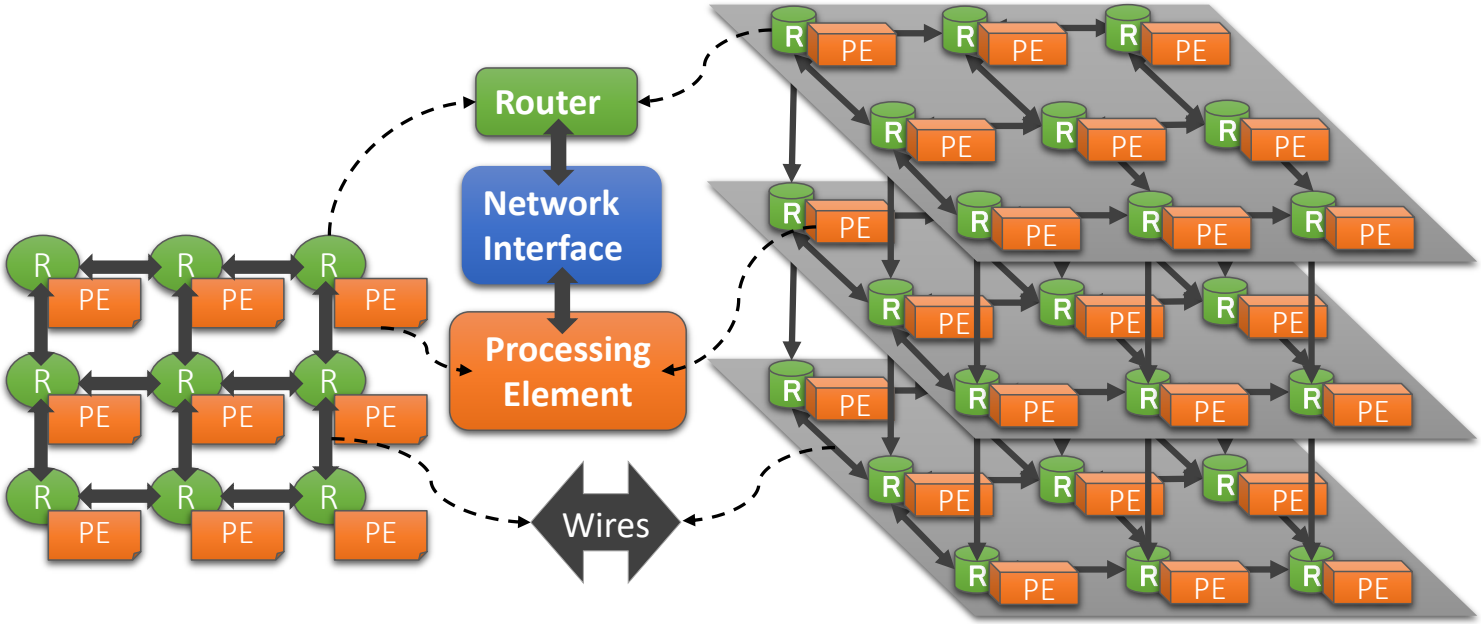



Figure 4: 3D Integration technologies: (a) Wire bonding; (b) Solder balls; (c) Through Silicon Vias (TSVs); (d) Wireless stacking.

Table 1: Performance and power: 3D vs 2D architecture [4].

| # of input bits | Kogge-Stone Adder | | Log shifter 16 | | Log shifter 32 | |
|---|---|---|---|---|---|---|
| | 16-bits | | | | 32-bits | |
| | Delay | Power | Delay | Power | Delay | Power |
| 2 planes | -20.23% | -8% | -13.4% | -6.5% | -28.4% | -8% |
| 3 planes | -23.60% | -15% | - | - | - | - |
| 4 planes | -32.70% | -22% | - | - | - | - |

# On-Chip Communication Network




2D Mesh Network-on-Chip

3D Mesh Network-on-Chip

**Network-on-Chips** is an on-chip communication infrastructure:

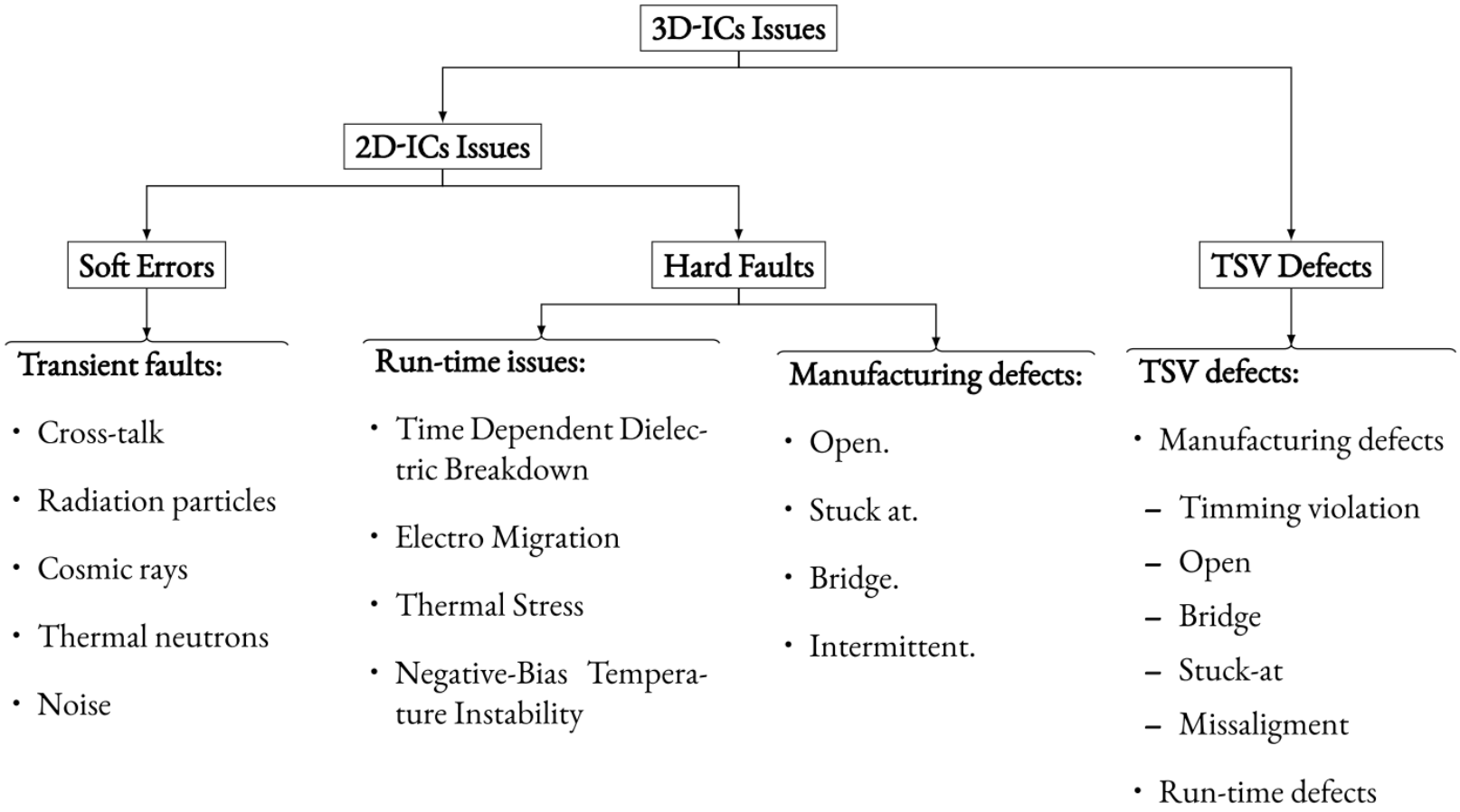
- *Processing Elements* (PEs) are attached to *routers* via *Network Interfaces* (NIs).
- *Network* is established from a set of *routers* in a specific form (topology, size, flit-width) and transaction protocols (node to node, end to end).
- Data (message/packet/flit) transmissions between PEs are handled by *routing* inside the network.

# 3D Network-on-Chip

- Among the existed interconnect infrastructure (e.g. Bus, Point-to-Point), **Network-on-Chips** have offered benefits on *parallelism*, *scalability* and *resource usability*.
- **3D integration** is considered as the future of ICs that can improve the *performance*, reduce the *footprint*, decrease the *power consumption*, and offer *multiple technologies integration*.
- By using **Network-on-Chips** on *3D integration*[1], we obtain *3D-Network-on-Chips* (3D-NoCs) that inherits the benefits from the both technologies.
- Recently, NoCs are widely used for multi/many core processing. Therefor, **3D-NoCs** will be the future paradigm of multi/many core processing 3D-ICs.
- *However,* due to the *vulnerability* of deep sub-micron devices and the *high defect rate* of TSVs, 3D-NoCs are predicted to encounter the **reliability challenge**.

[1] TSVs handle the vertical wires between routers.

# Fault/Error Types of 3D-ICs/3D-NoCs




Beside the benefits, TSV-based 3D-ICs also have challenges on reliability. Especially, the *high defect rates* of TSVs are problematic. *Thermal removal difficulty* and *stress issues* also accelerate the fault rates.

# Research Motivation

1. Future TSV-based 3D-ICs need fault-tolerances in order to deal with their reliability issues.
2. As considered as the backbone of future 3D-ICs, 3D-NoCs also need fault-tolerance methods to ensure the reliability of their communications.
3. There are numerous number of fault-tolerance works on: *soft errors*, *hard faults*, and *TSV defects*; however, there is also a need of a comprehensive work that can handle all type of faults.
4. Beside *fault recovery*, *fault detection and diagnosis* are also important aspects of fault resilience. Handling faults on-line also help reduce the threat giving by the occurred faults.

# Goals and Contributions

## ❶ A highly reliable comprehensive soft-errors and hard-faults resilient architectures, algorithms, and design methodologies

To provide a comprehensive fault-tolerance method that can handle both soft errors and hard faults. Moreover, a *detection, diagnosis and recovery* scheme is also proposed to help in on-line fault/error handling.

## ❷ A scalable cluster-TSV defect tolerance for vertical connections

Because the cluster-defect is a critical issue that cannot be efficiently dealt by using redundancies, this work proposes a cluster-TSV defect tolerance for 3D-NoCs.

# Table of Contents

# Soft Error Hard Fault Tolerant Architectures and Algorithms (1/4)

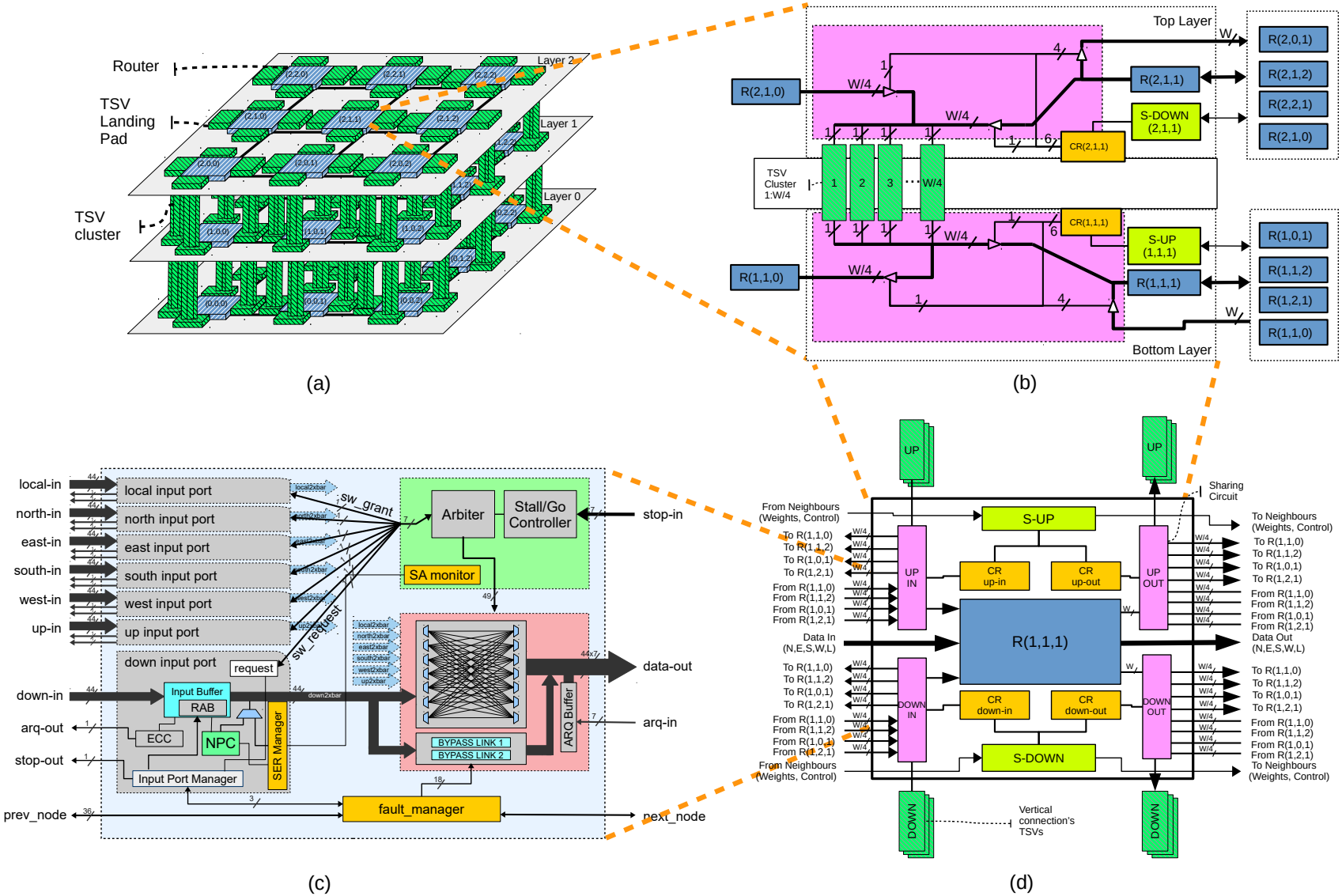



Figure 5: System architecture: (a) 3D NoC, (b) Interface between two routers from adjacent layers; (c) Router architecture; and (d) The wrapped router.

# Soft Error Hard Fault Tolerant Architectures and Algorithms (2/4)

## Proposed Algorithms and Architecture:

- A soft error resilience method, named as **Pipeline Computation Redundancy** (PCR), to handle soft errors on pipeline stage.
  - Multiple executions to detect and correct soft errors.
  - Since *NextPortComputing*/*SwitchAllocation* are the important part inside the network, we handle soft errors using PCR.
- A **detection, diagnosis and recovery mechanism** (DDRM) for handling the possible faults.
- As a summary, a comprehensive design of 3D-NoC system (3D-FETO) that can handle both soft errors and hard faults.

# Soft Error Hard Fault Tolerant Architectures and Algorithms (3/4)

To complete the design, we adopted the following fault-tolerant methods:

- **Error Correction Code**: SECDED (Single Error Correction, Double Error Detection) [5] to protect data path against soft errors.
- **Buffer Slot Fault Tolerance**: Random Access Buffer[6].
- **Crossbar Link Fault Tolerance**: Bypass-Link-on-Demand[6].
- **Intra-router Link Fault Tolerance**: Lookahead-Fault-Tolerant (LAFT) routing algorithm[7].

# Soft Error Hard Fault Tolerant Architectures and Algorithms (4/4)

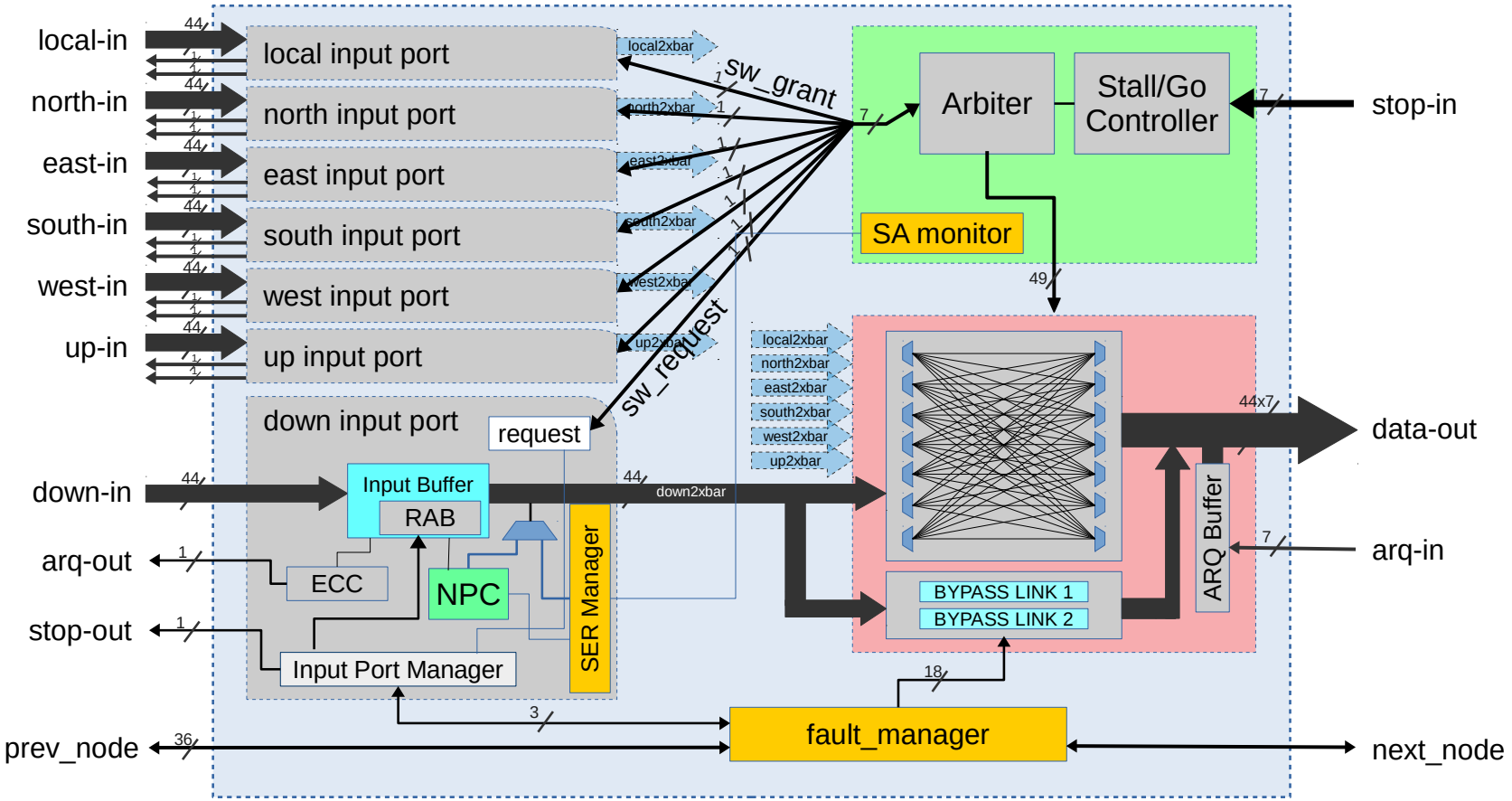
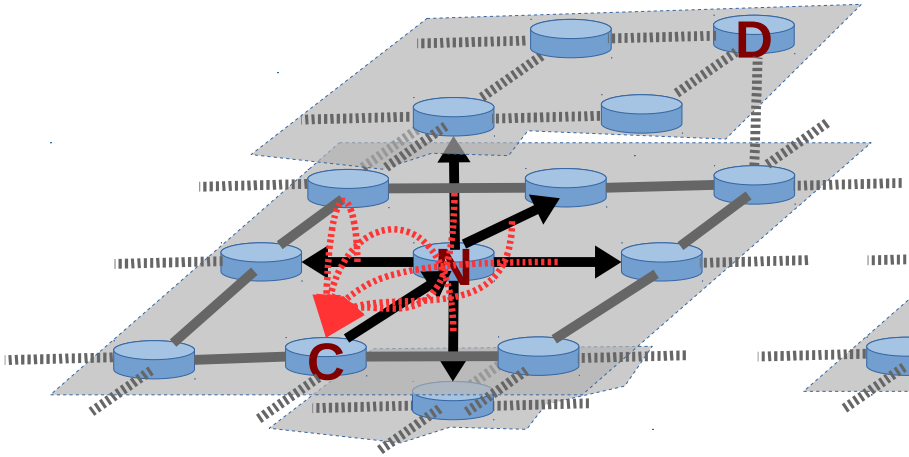
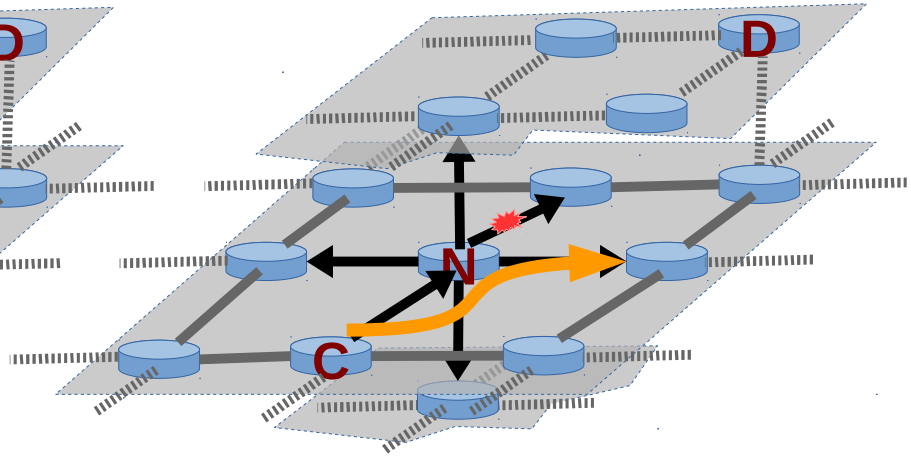



Figure 6: Adaptive 3D-FETO router architecture.

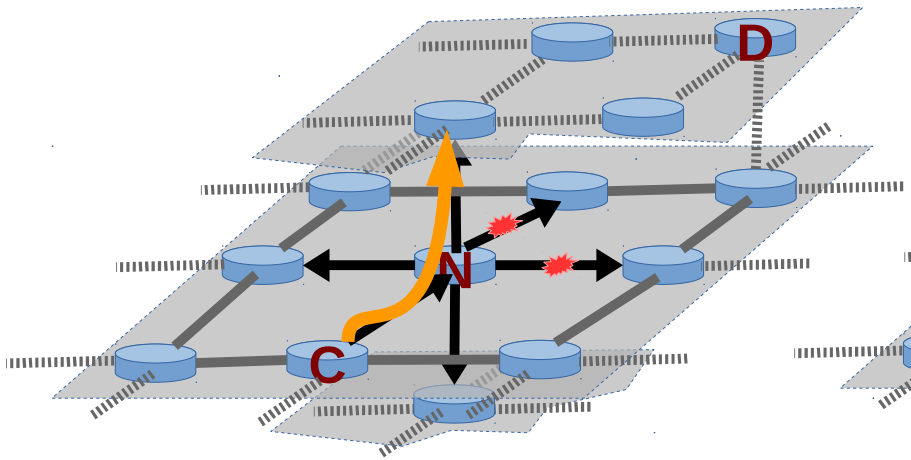
# Hard Fault-Tolerance (1/2)



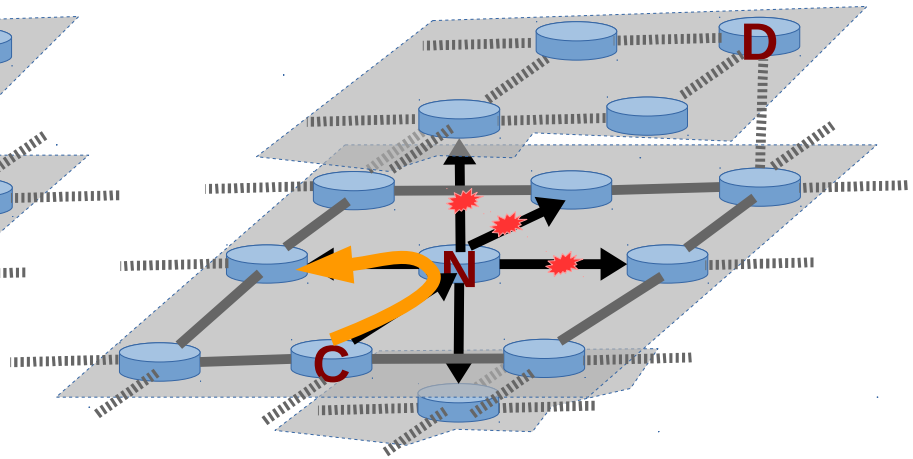

Updating faulty information.




Case1: One faulty link.




Case2: Two faulty links.




Case3: Minimal routing.

Figure 7: Look-Ahead Fault Tolerant Routing[6].

# Hard Fault-Tolerance (2/2)

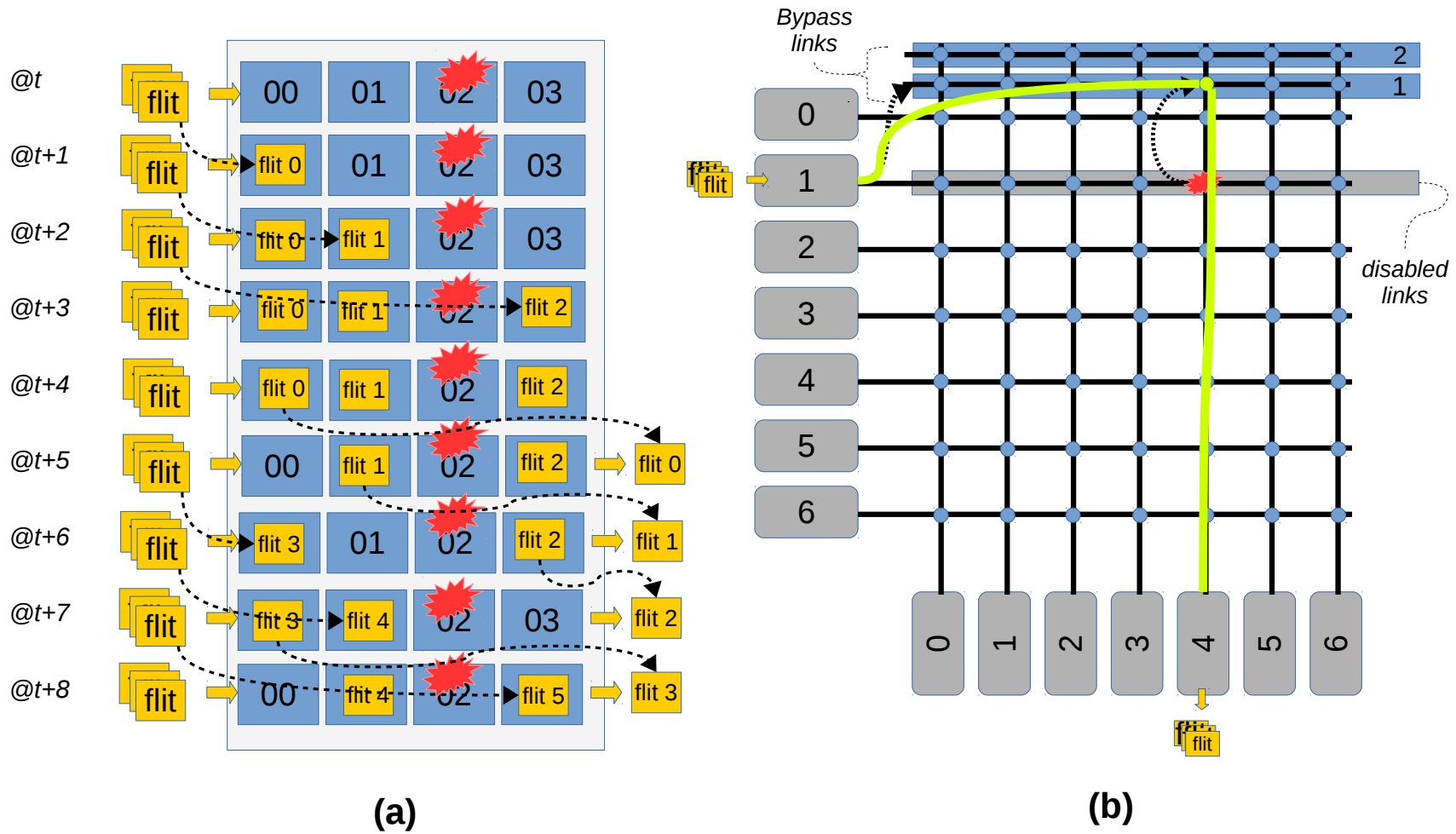



Figure 8: (a) Buffer fault-tolerance[6]; (b) Crossbar fault-tolerance[6].

# Pipeline Computation Redundancy Algorithm

```
   // input flit's data
   Input: in_flit
   // output flit's data
   Output: out_flit

   // Write flit's data into buffers
1  BufferWriting(in_flit)
   // Compute first time of NPC and SA
2  next_port[1] = NextPortComputing(in_flit)
3  grants[1] = SwitchAllocation(in_flit)

   // Compute redundant of NPC and SA
4  next_port[2] = NextPortComputing(in_flit)
5  grants[2] = SwitchAllocation(in_flit)

   // Compare orginal and redundant to
      detect soft-error
   // Soft-error on NPC
6  if (next_port[1] ≠ next_port[2]) then
       // roll-back and recalculate NPC
7      next_port[3] =
        NextPortComputing(in_flit)
8      final_next_port =
        MajorityVoting(next_port[1,2,3]);
9  else
       // No soft-error on NPC
10     final_next_port = next_port[1]

   // Soft-error on SA
11 if (grants[1] ≠ grants[2]) then
       // roll-back and recalculate SA
12     grants[3] = SwitchAllocation(in_flit)
13     final_grants =
        MajorityVoting(grants[1,2,3])
14 else
       // No soft-error on SA
15     final_grants = grants[1]

   // After detection and recovery, the
      algorithm finishes with CT
16 out_flit = CrossbarTraversal(in_flit,
    final_next_port, final_grants);
```

**Algorithm 1:** Algorithm of Pipeline Computation Redundancy (PCR).

# Pipeline Computation Redundancy Timeline

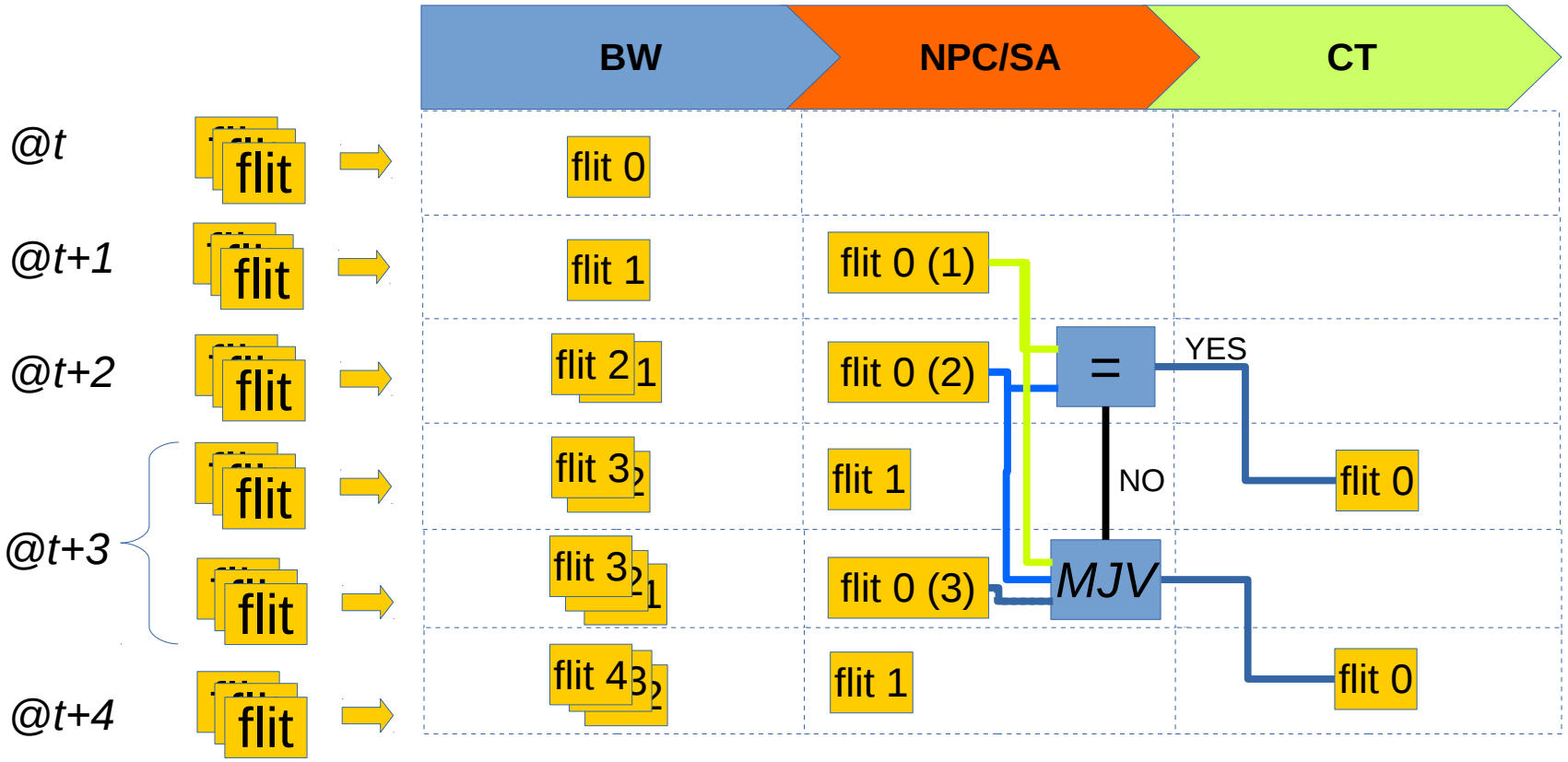



Figure 9: Pipeline Computation Redundancy (PCR).

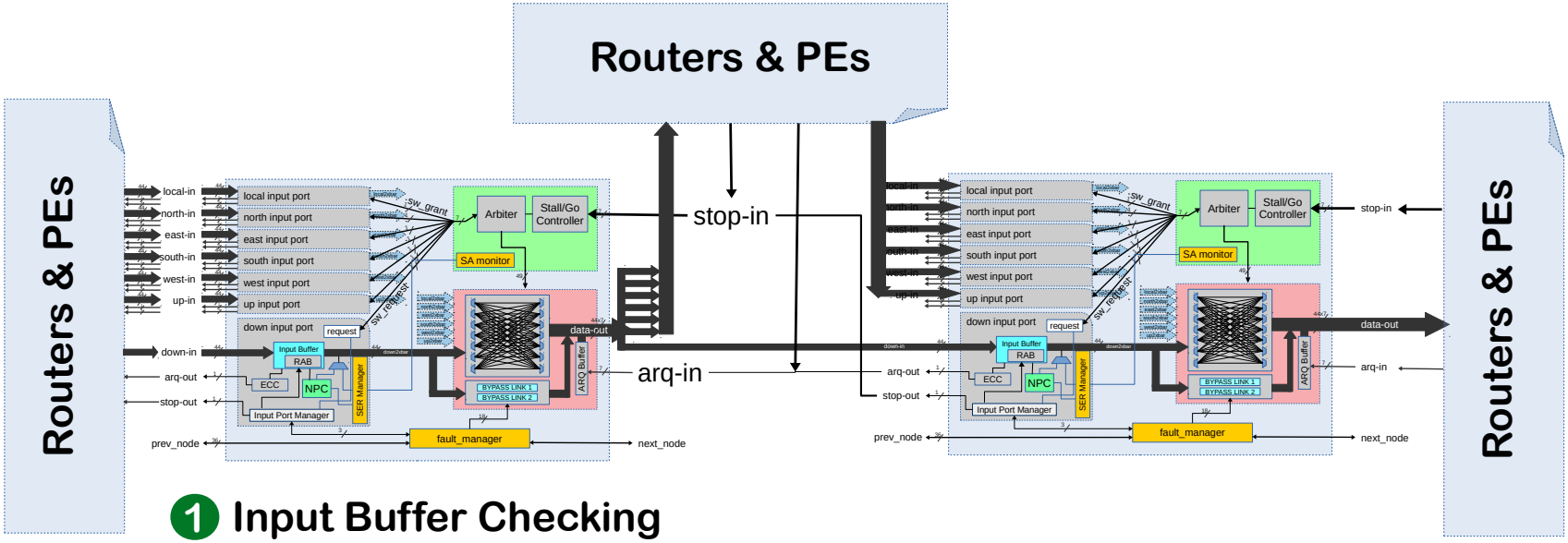
- **BW**: *Buffer Writing*
- **NPC**: *Next Port Computing*
- **SA**: *Switch Allocation*
- **CT**: *Crossbar Traversal*

# Detection, Diagnosis and Recovery Mechanism Algorithm

```
// Automatic Retransmission Request
Input: transmitting_flit
// Transmitted Buffer Position
Input: buffer_position
// Control signal to all Fault-Tolerance
   modules
Output: RAB_control, BLoD_control,
        LAFT_control
// Transmit the flit, get the ECC's feedback
1  Transmit(transmitting_flit);
2  ECC_result = ECC-Decoder(transmitting_flit);
   // DETECTION PHASE:
3  if ECC_result == ARQ then
       // Automatic Retransmission Request
4      increase(ARQ_counter);
5      ARQ(transmitting_flit);
6  else
       // The transmitted flit is non faulty
7      Finish;
   // Check the number of consecutive ARQs
8  if (ARQ_counter == 2) then
       // There is a permanent fault
       // Jump to DIAGNOSIS-RECOVERY PHASE
   // DIAGNOSIS-RECOVERY PHASE:
   // Start with Input Buffer Checking
9  Buffer_Failure ←
     Buffer_Checking(buffer_position);
10 if (Buffer_Failure == Yes) then
       // Random Access Buffer is received the
          position to handle.
11     RAB_Control = buffer_position;
12     Finish;
13 else
       // The buffer slot is non faulty.
       // Move to Crossbar Checking: using a
          Bypass-Link.
14     BLoD_control = enable;
       // Get the ECC's feedback and detect with
          ARQ counter.
15     if (ARQ_counter == 2) then
           // BLoD cannot fix the fault, the link
              is failed.
16         BLoD_control = release;
           // The LAFT routing algorithm handles
              the faulty link.
17         LAFT_control = faulty;
18         Finish;
19     else
           // BLoD already fixed the failure, the
              recovery step is finished.
20         Finish;
```

**Algorithm 2:** Fault Detection, Diagnosis and Recovery.

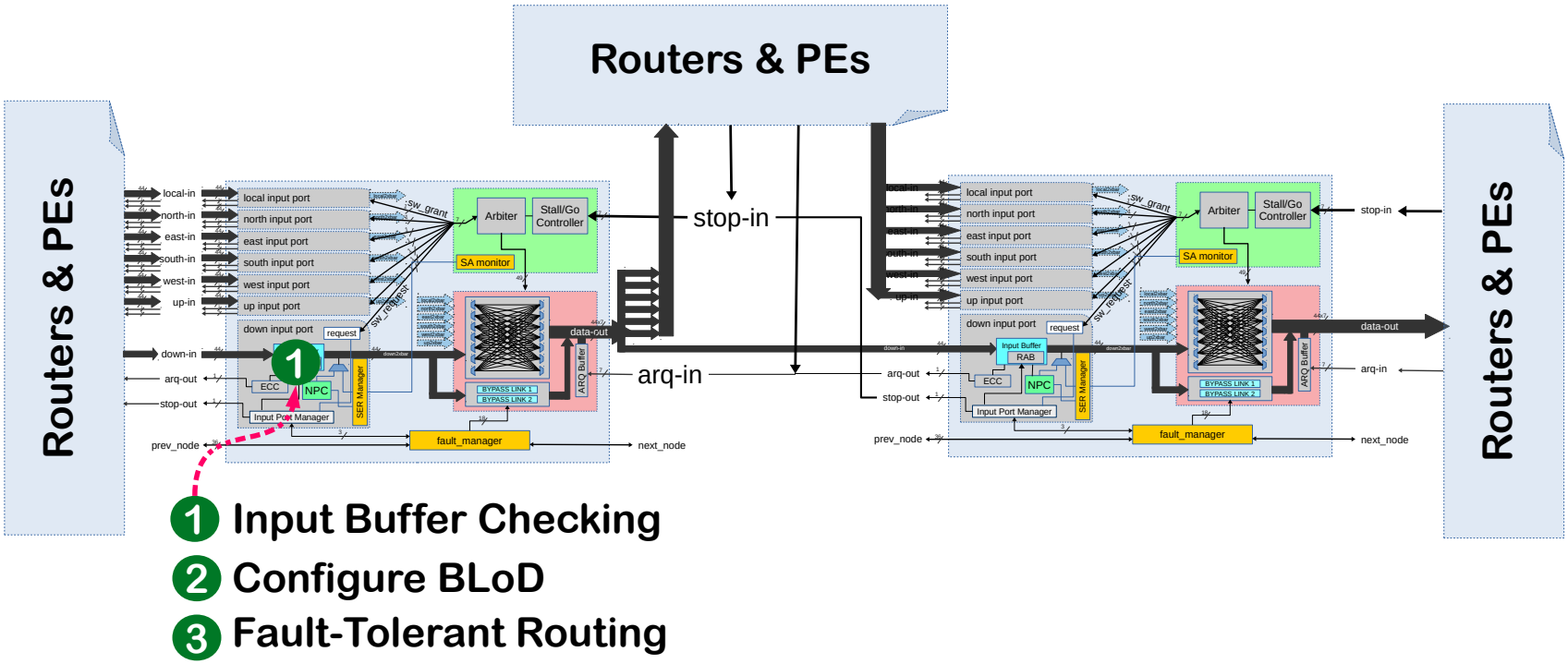
# Detection, Diagnosis and Recovery Mechanism




1. **Input Buffer Checking**
2. **Configure BLoD**
3. **Fault-Tolerant Routing**

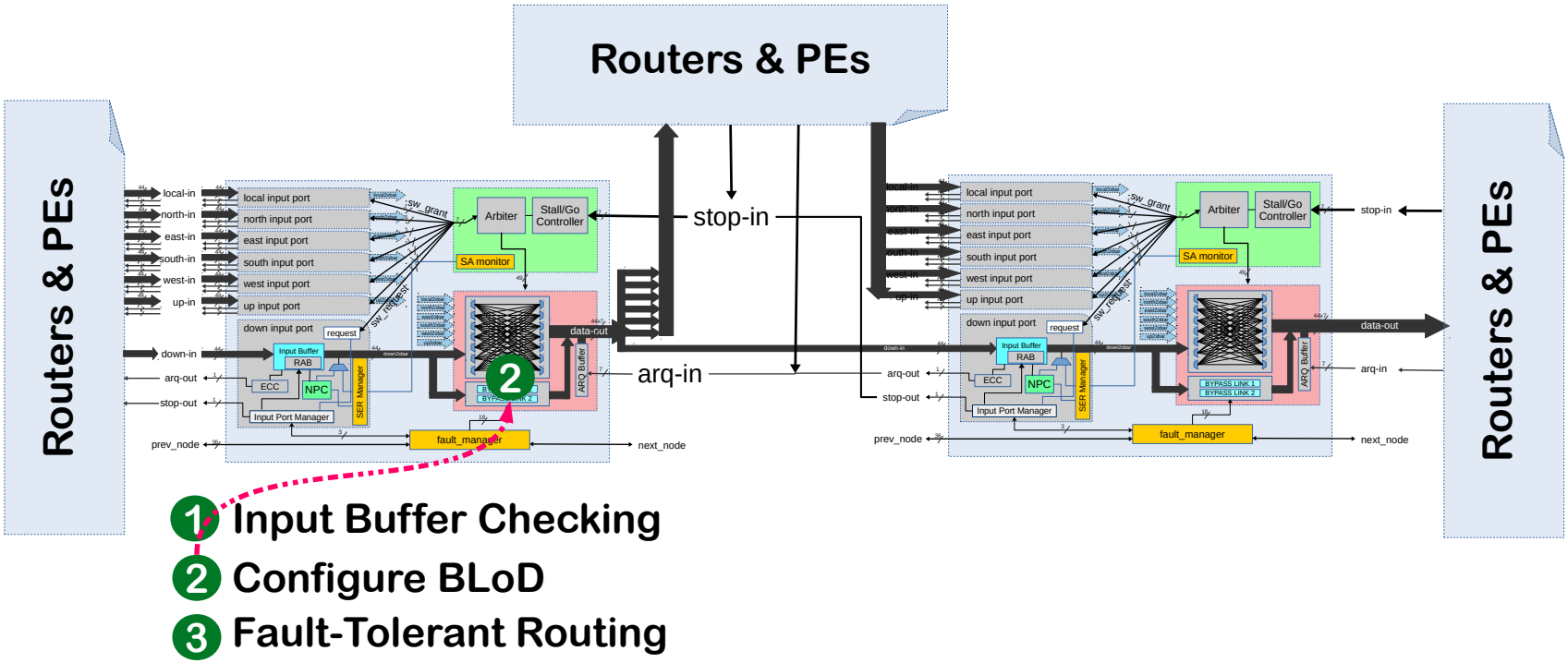
1. Input buffer slot faults are handled by *Random Access Buffer*.
2. Crossbar link faults are handled by *Bypass Link on Demand* (BLoD).
3. Inter-routers channel faults are handled by *Look-Ahead Fault-Tolerant* routing algorithm.

# Detection, Diagnosis and Recovery Mechanism



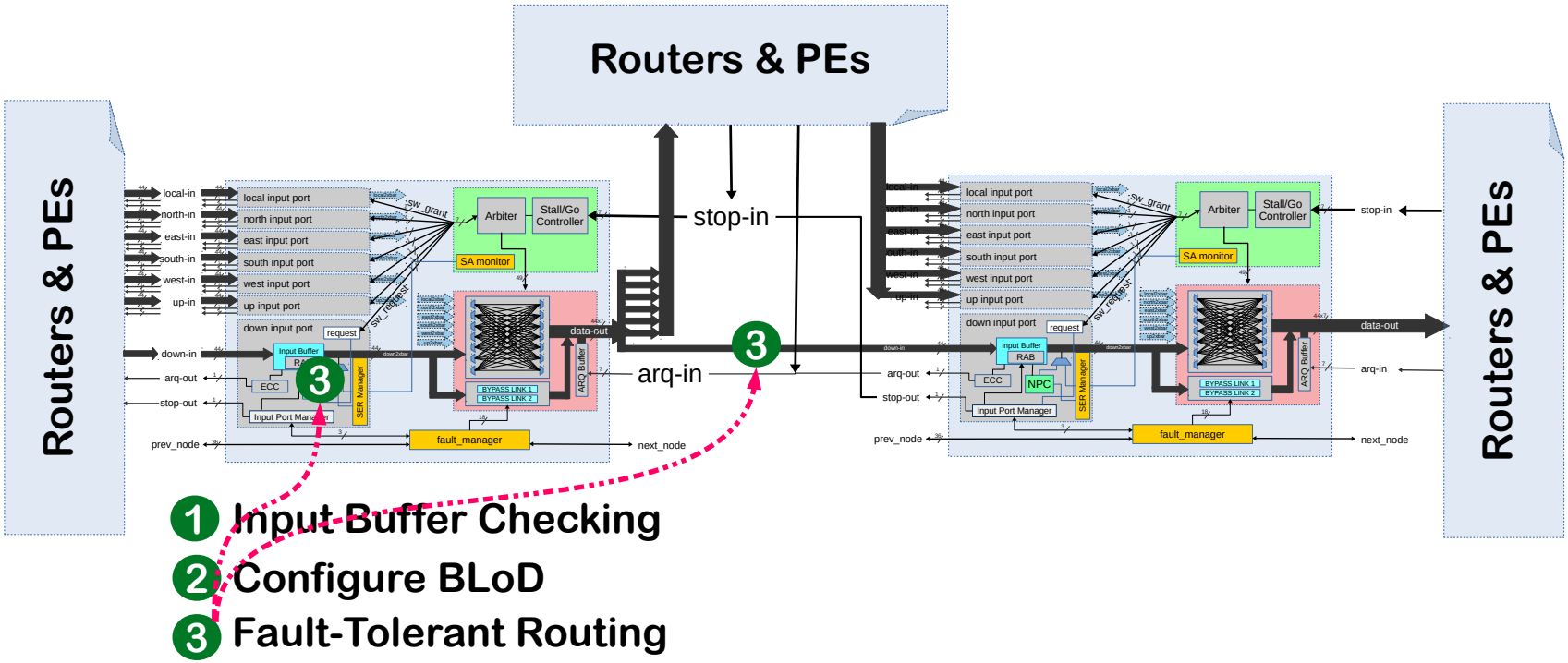

1. Input buffer slot faults are handled by *Random Access Buffer*.
2. Crossbar link faults are handled by *Bypass Link on Demand* (BLoD).
3. Inter-routers channel faults are handled by *Look-Ahead Fault-Tolerant* routing algorithm.

# Detection, Diagnosis and Recovery Mechanism



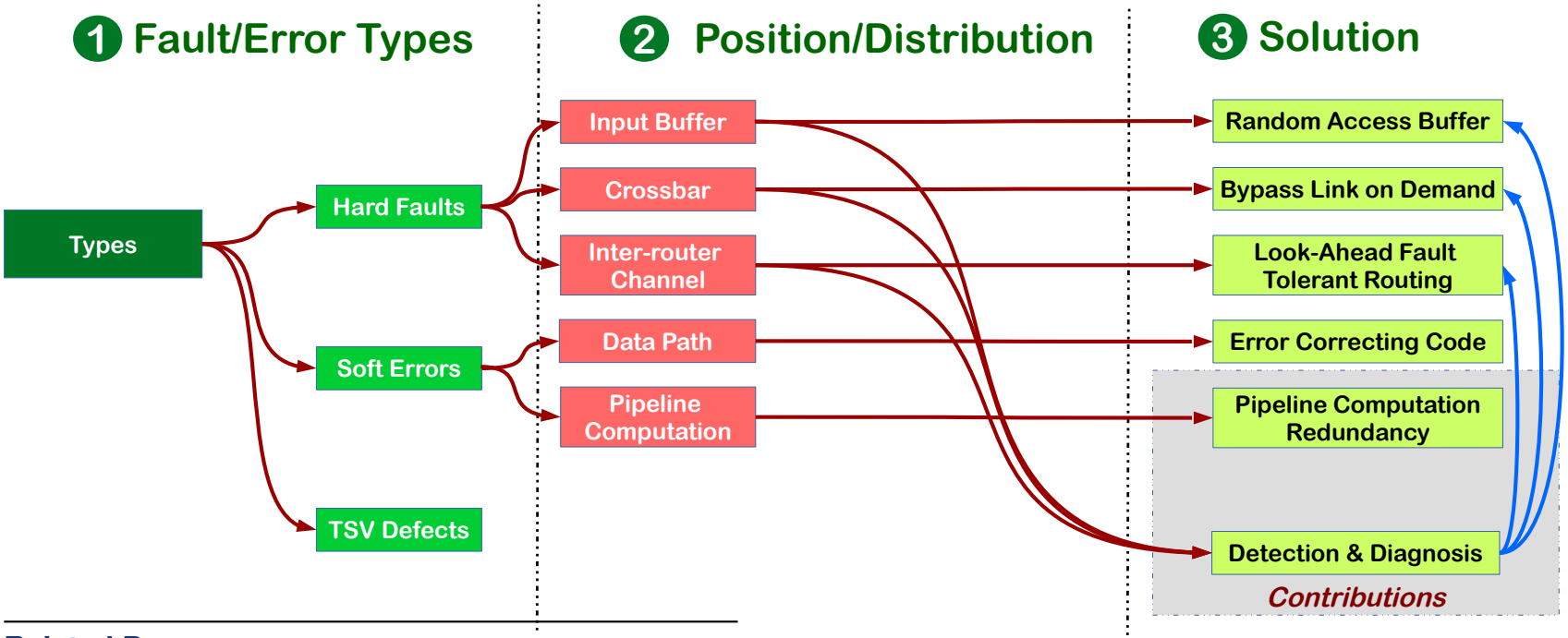

1. Input buffer slot faults are handled by *Random Access Buffer*.
2. Crossbar link faults are handled by *Bypass Link on Demand* (BLoD).
3. Inter-routers channel faults are handled by *Look-Ahead Fault-Tolerant* routing algorithm.

# Detection, Diagnosis and Recovery Mechanism




1. Input buffer slot faults are handled by *Random Access Buffer*.
2. Crossbar link faults are handled by *Bypass Link on Demand* (BLoD).
3. Inter-routers channel faults are handled by *Look-Ahead Fault-Tolerant* routing algorithm.

# Soft Error Hard Fault Tolerant Architectures and Algorithms




**Related Paper**


- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, "A Low-overhead Soft-Hard Fault Tolerant Architecture, Design, and Management Scheme for Reliable High-performance Many-core 3D-NoC Systems", *The Journal of Supercomputing*, Volume 73, Issue 6, pp 2705–2729, 2017.
- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, "Reliability Assessment and Quantitative Evaluation of Soft-Error Resilient 3D Network-on-Chip Systems", The IEEE 25th Asian Test Symposium (ATS), pp. 161-166, Hiroshima, Japan, November 21-24, 2016.
- **Khanh N. Dang**, Yuichi Okuyama, and Abderazek Ben Abdallah, "Soft-error resilient network-on-chip for safety-critical applications", *The 2016 International Conference on IC Design and Technology (ICICDT)*, pp. 1-4, Ho Chi Minh City, Vietnam, June 27-29, 2016.
- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama, Abderazek Ben Abdallah, and Xuan-Tu Tran, "Soft-error resilient 3d network-on-chip router", *The 2015 IEEE 7th International Conference on Awareness Science and Technology (iCAST)*, pp. 84-90 Qinhuangdao, China, September 22-24, 2015.

# Table of Contents

# Scalable Cluster-TSV Defect Tolerance (1/4)

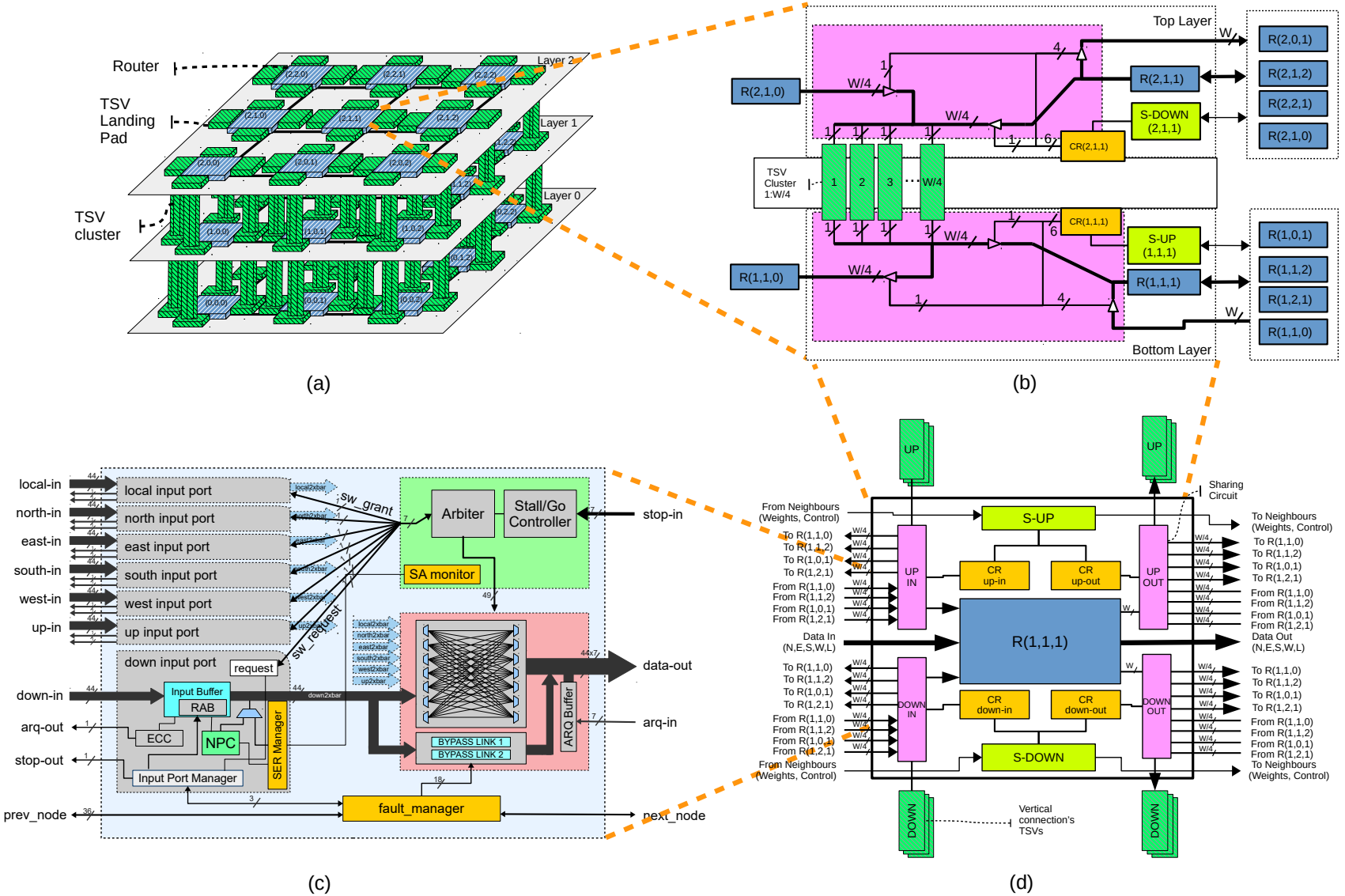



Figure 10: System architecture: (a) 3D NoC, (b) Interface between two routers from adjacent layers; (c) Router architecture; and (d) The wrapped router.

# Scalable Cluster-TSV Defect Tolerance (2/4)

**Approach**:

- A method to organize the TSVs in 3D-NoC systems to handle the cluster defect[2].
- A cluster-TSV defect recovery method without adding TSV redundancies.
- An adaptive online algorithm to handle the cluster-TSV defect.

[2] In fact, in this design, ECC code can handle random failed TSVs.

# Fault Assumption

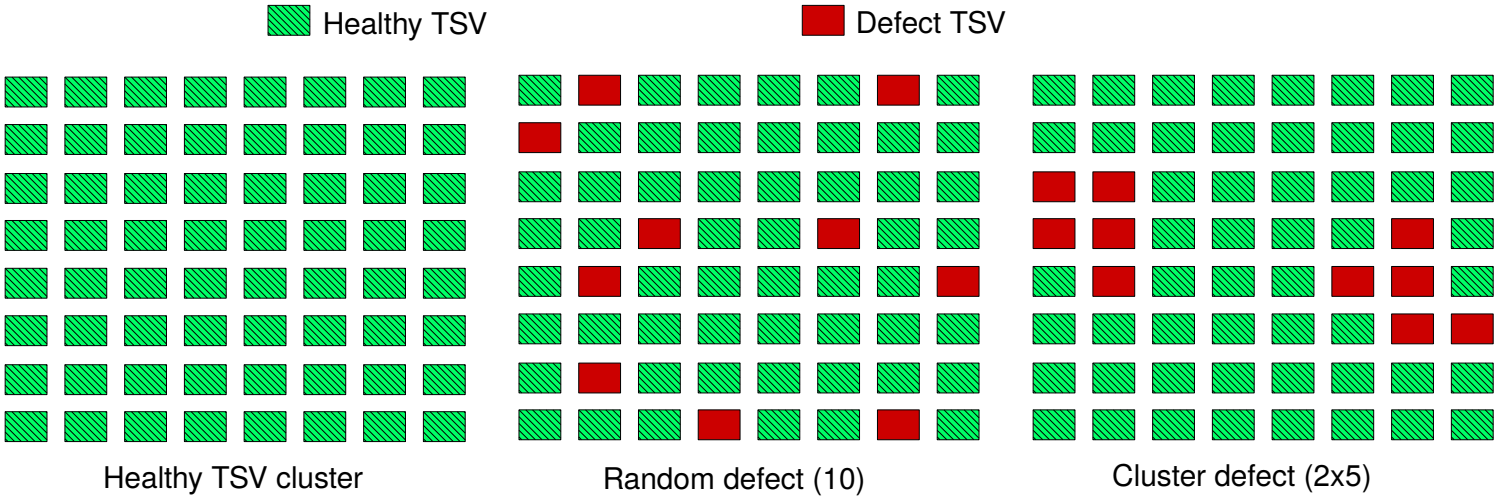



Figure 11: TSV fault assumption.

- This work only focuses on cluster defect. No random defects are considered.
- Detection is assumed to be done by a dedicated module[3].

[3] *DDRM* module can help detect the fault occurence; however, it does not support the diagnosis phase.

# Scalable Cluster-TSV Defect Tolerance (3/4)

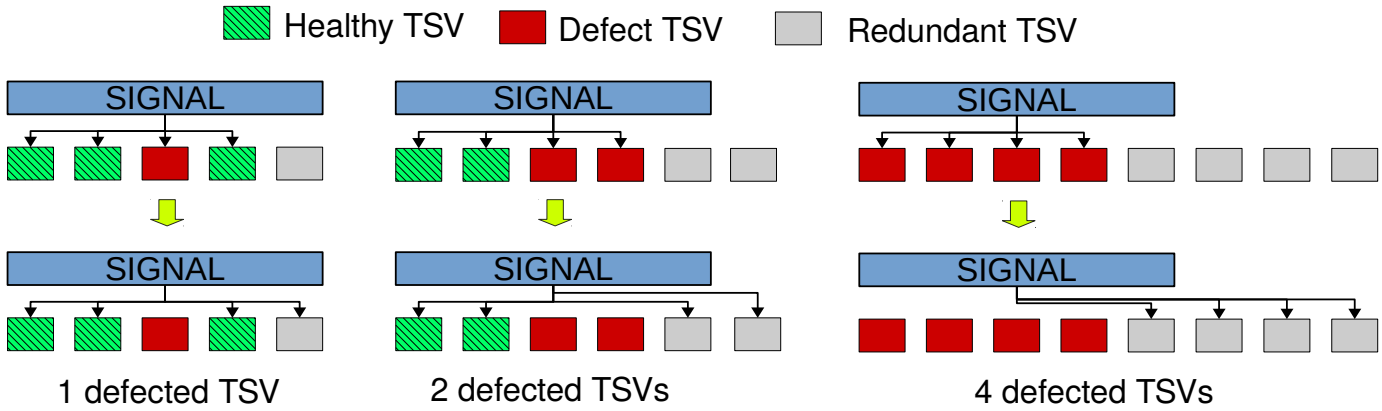



Figure 12: Conventional TSV fault-tolerant method.

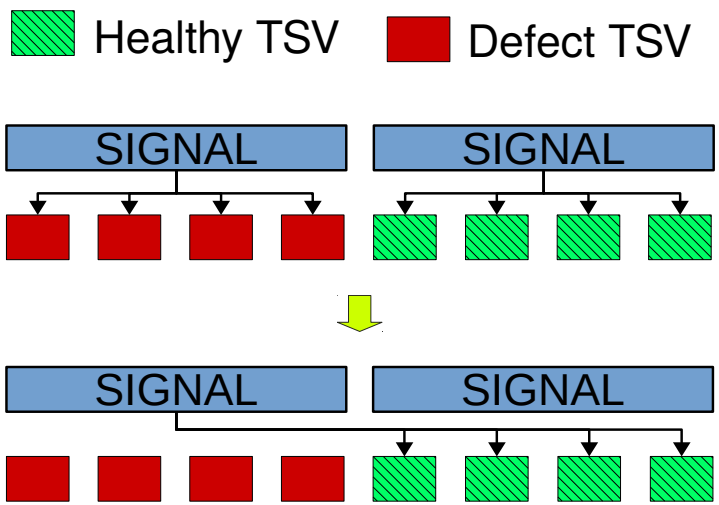



Figure 13: The proposed technique.

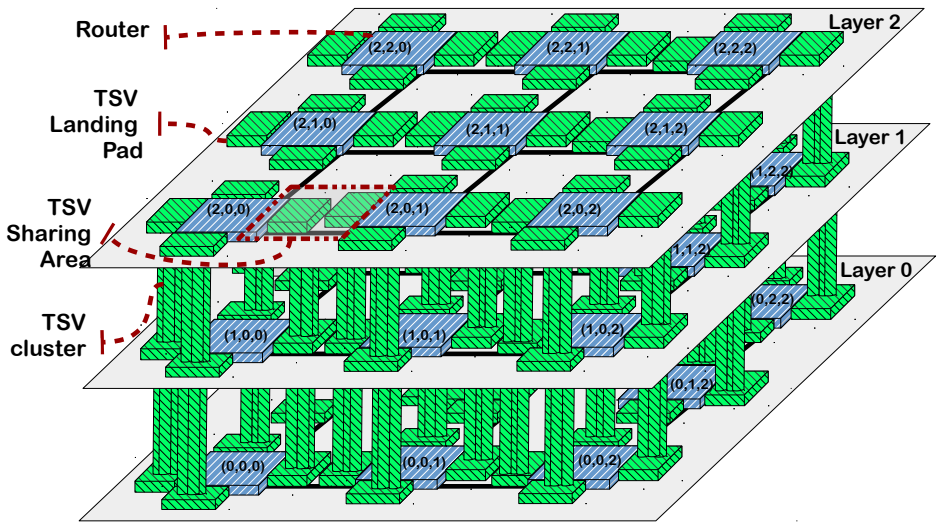
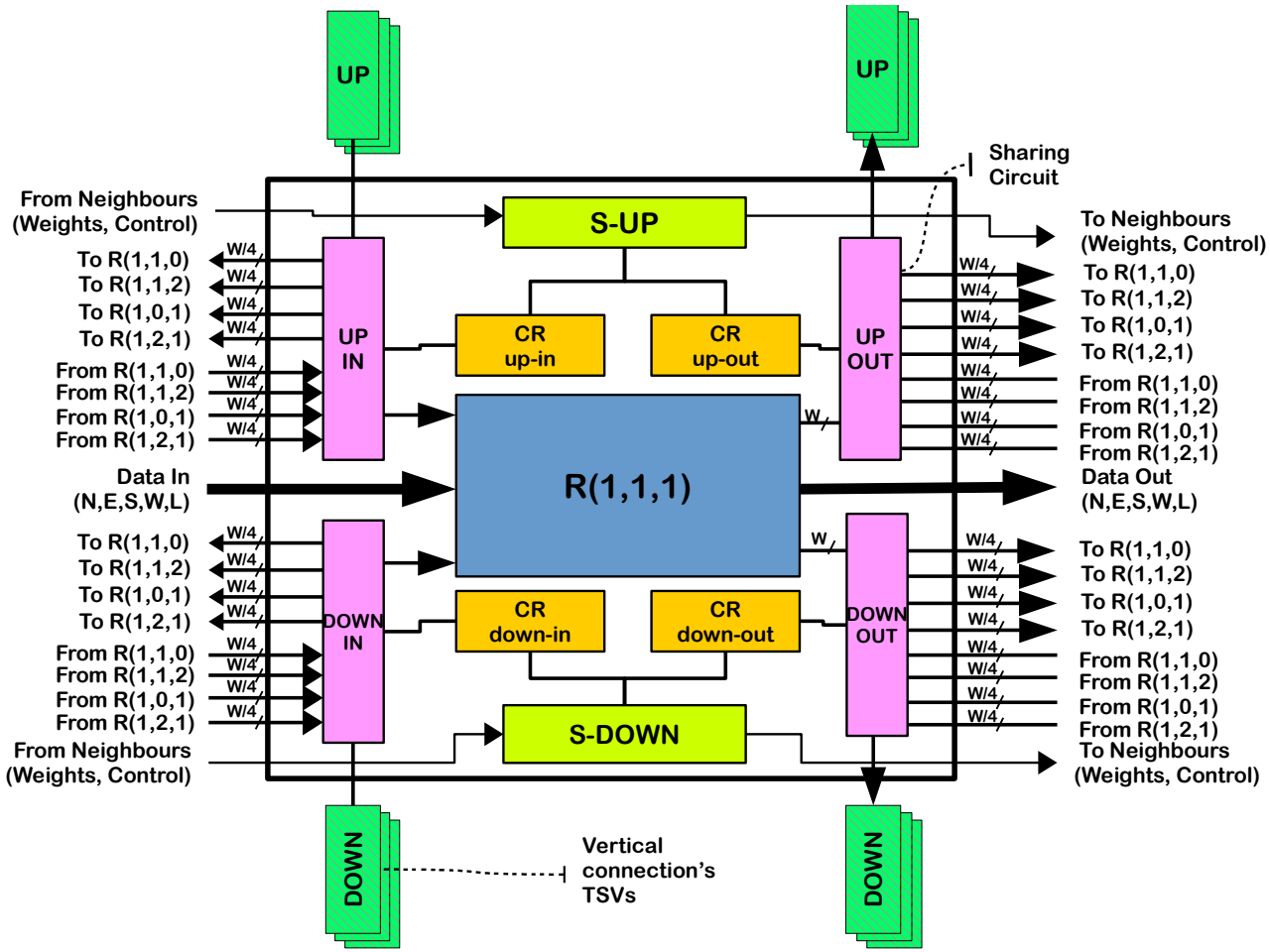



Figure 14: Simplified block diagram of the proposed system with configuration $3 \times 3 \times 3$.

# Scalable Cluster-TSV Defect Tolerance (4/4)

## Router Architecture




The TSV Router wrapper fault-tolerance architecture. *S-UP* and *S-DOWN* are the sharing arbitrators which manage the proposed mechanism. *CR* stands for configuration register and *W* is the flit width.

# Inter-Layer Connection

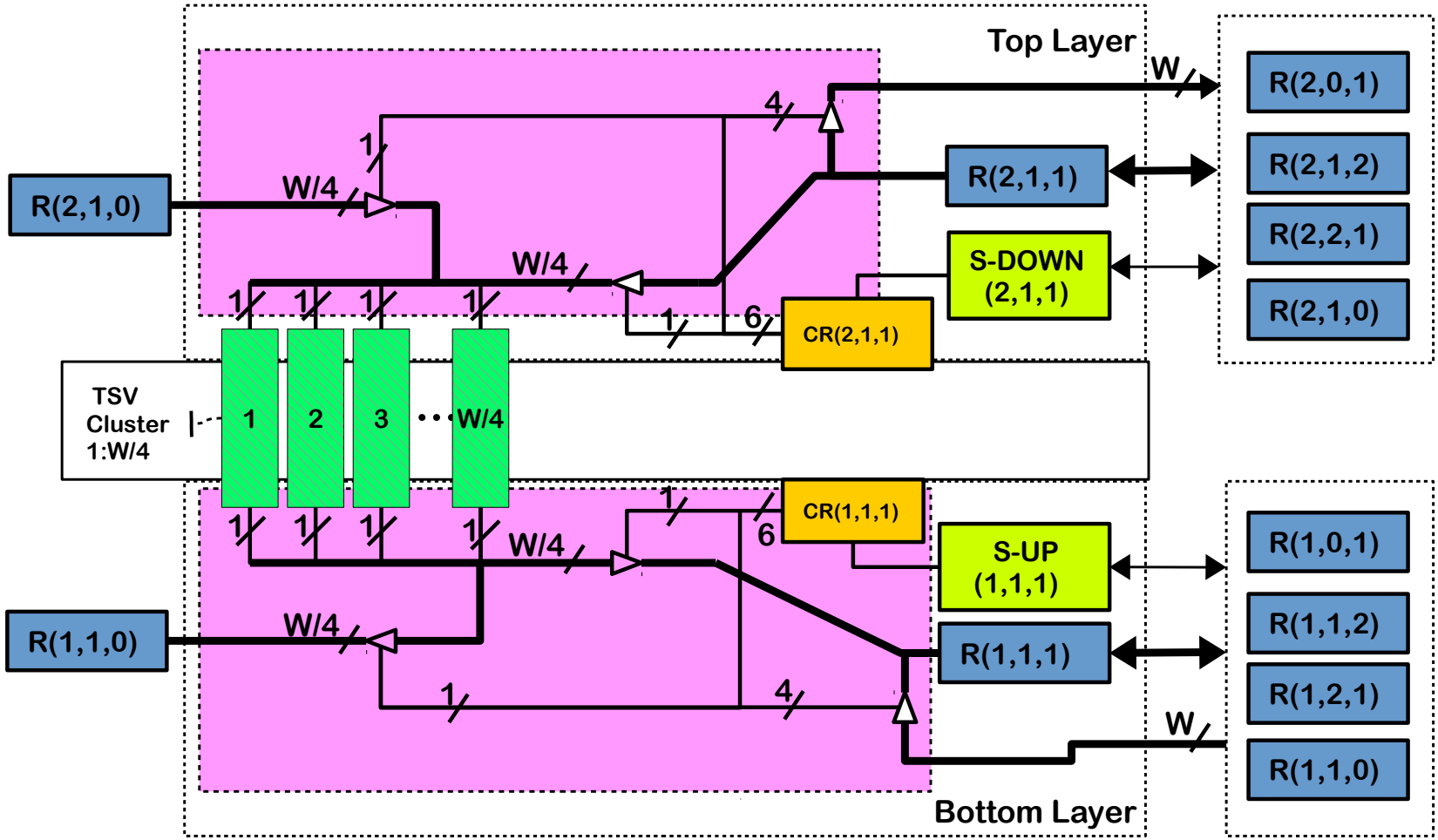
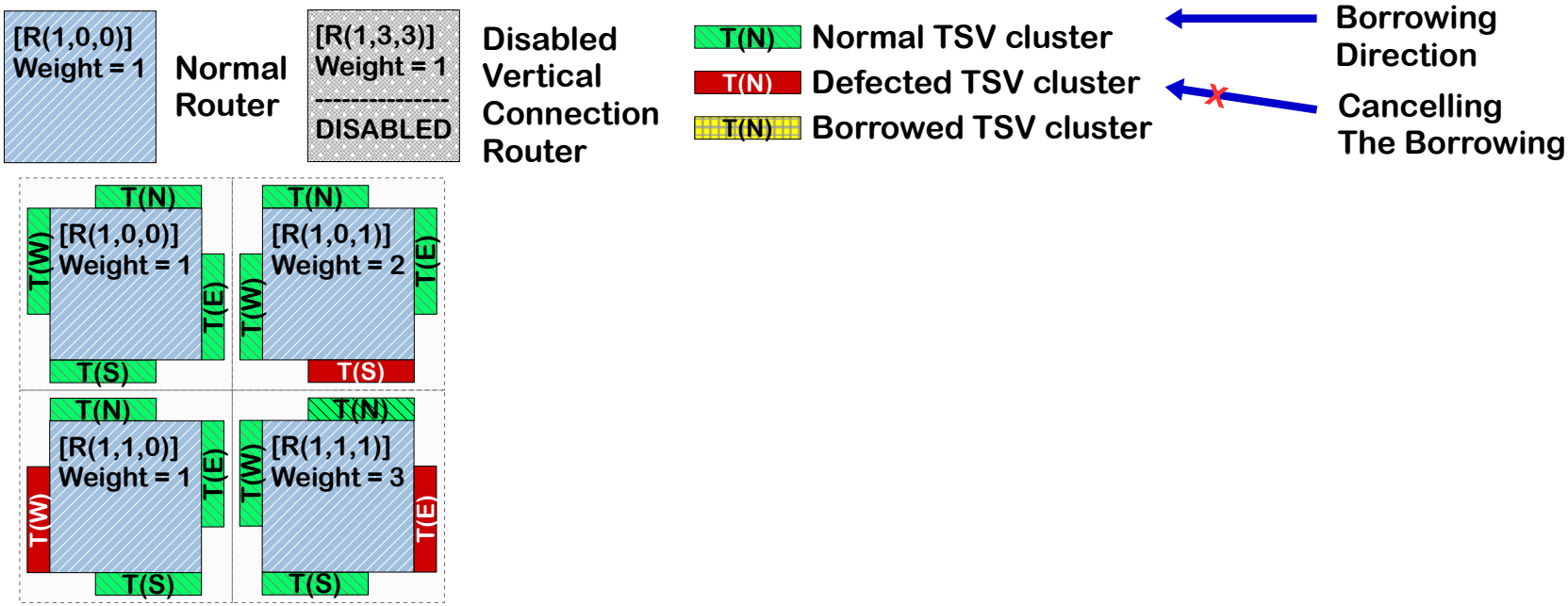



Figure 15: Cluster-TSV connection between two layers.

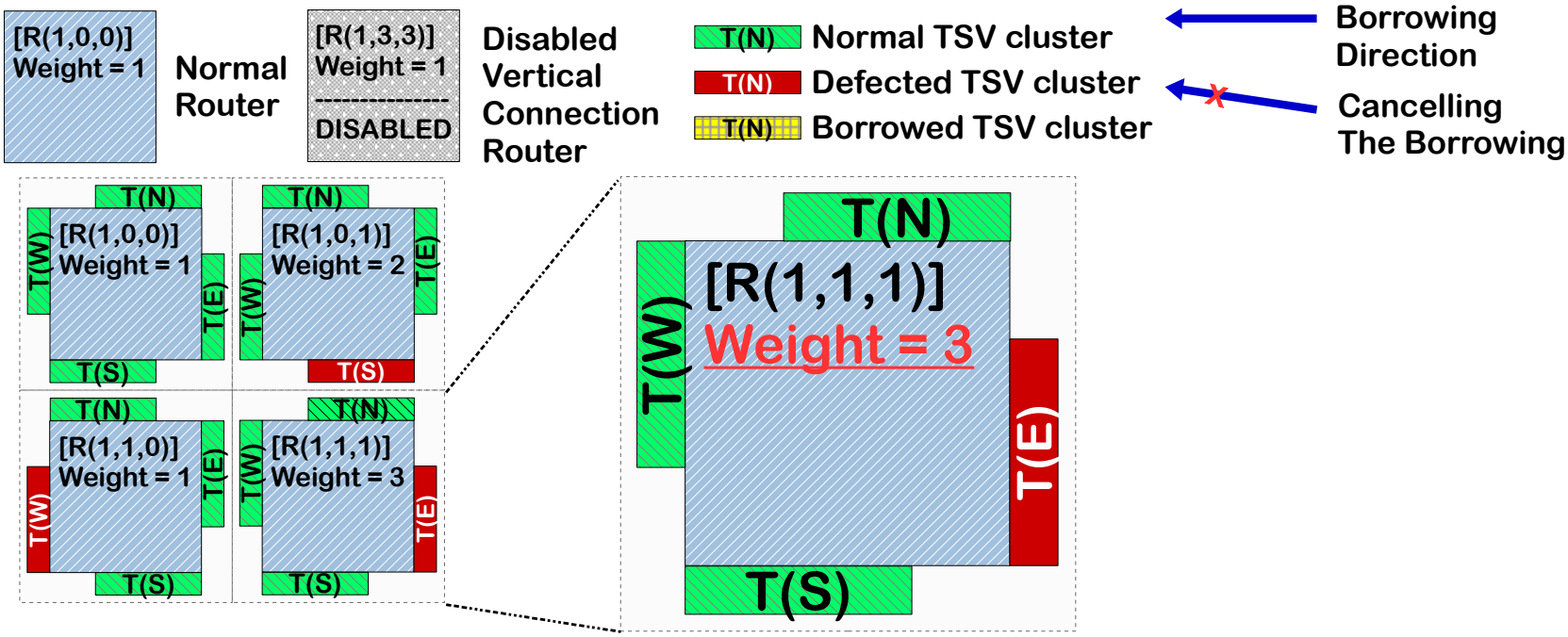
```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is minimal
8          and TSV_Status_neighbor[c] == "NORMAL";
9          if (c==NULL) then
10             return RQ_link[i] = "NULL"
11             return Router_Status = "DISABLE"
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

**Algorithm 3:** TSV Sharing Algorithm.

# TSV Sharing Algorithm



# TSV Sharing Algorithm

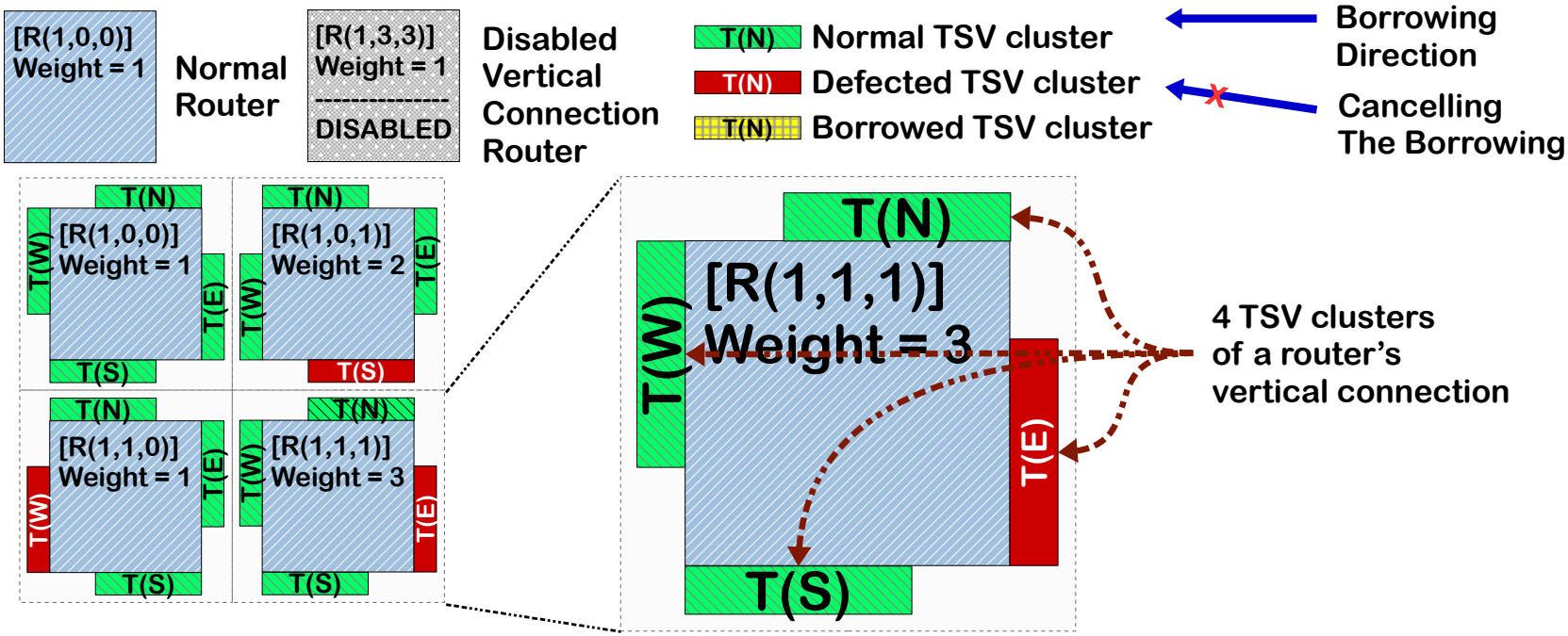



- Every router is assigned a weight value.

*Note:* The weight values can be generated based on traffic of the vertical connection of the router. In this work, we generate higher weights for the middle routers and lower weights for the border routers:

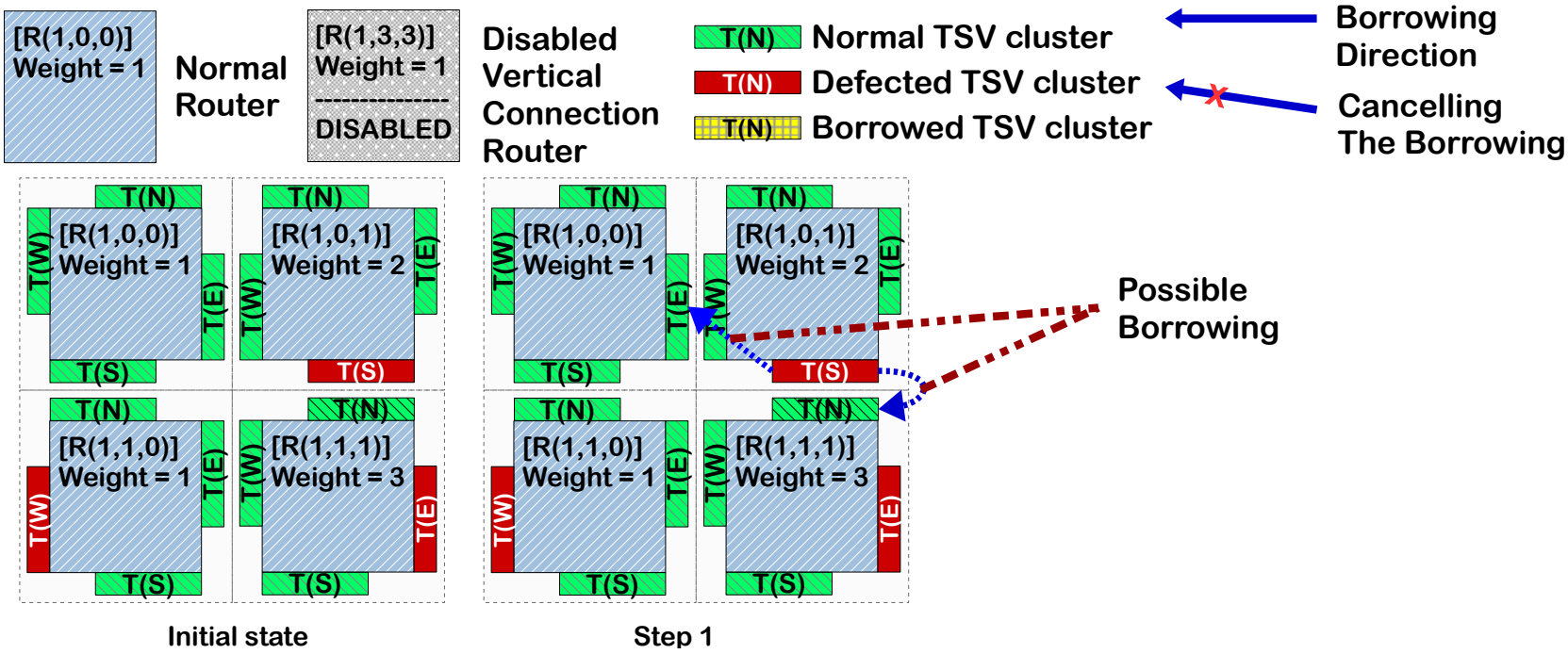
$$\text{Weight}_{\text{router}}(x, y) = \min(x, \text{cols} - x) + \min(y, \text{rows} - y) + 1 \qquad (1)$$

# TSV Sharing Algorithm



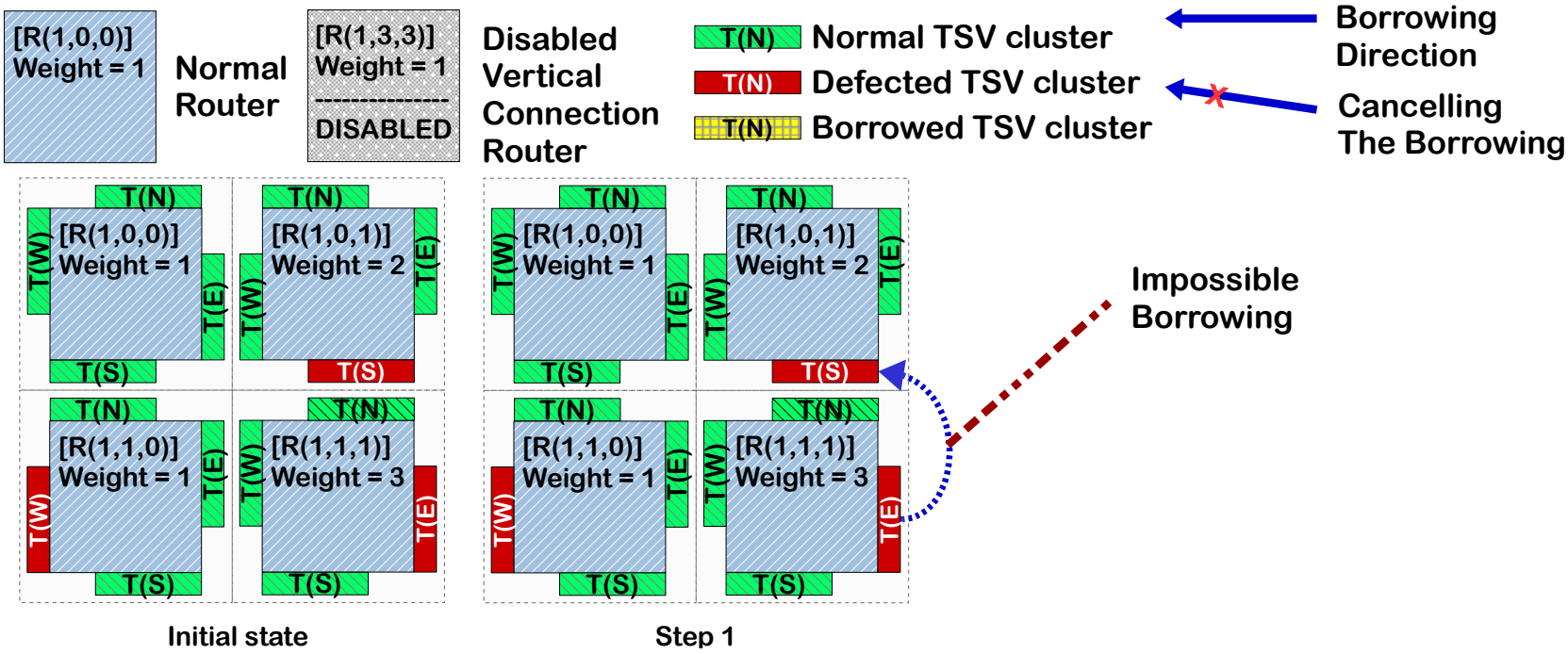

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.

# TSV Sharing Algorithm



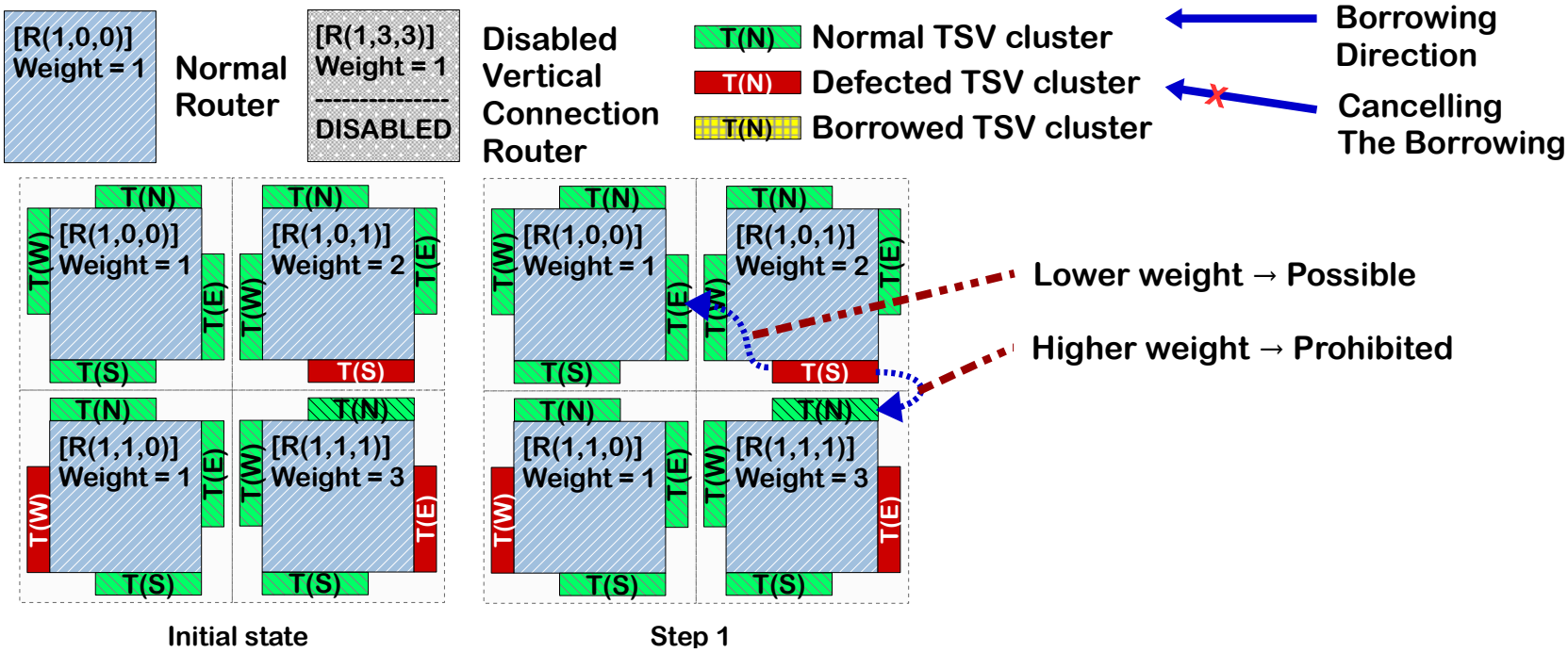

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.
- Each router having defected/borrowed TSV cluster (red/yellow) can borrow from one of its neighbors.

# TSV Sharing Algorithm



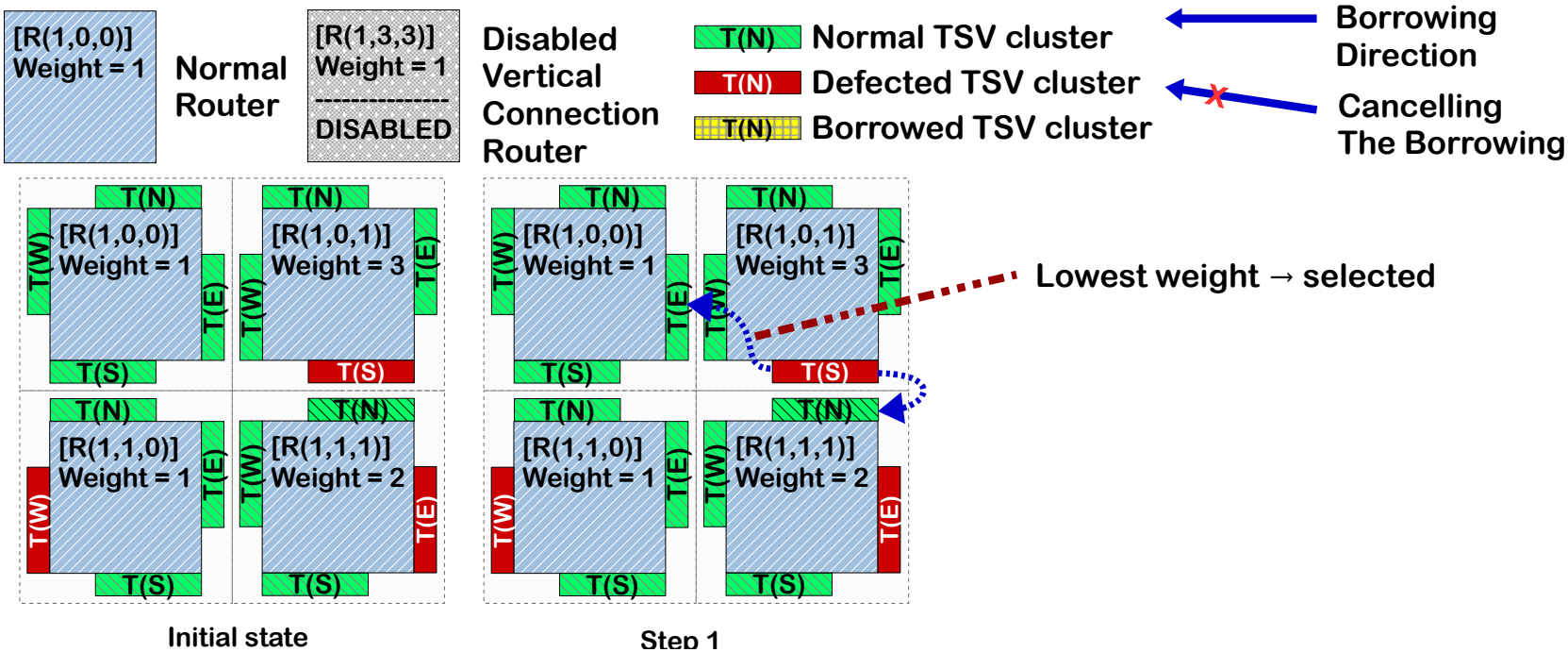

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.
- Each router having defected/borrowed TSV cluster (red/yellow) can borrow from one of its neighbors.
- The borrowed cluster must be healthy.

# TSV Sharing Algorithm



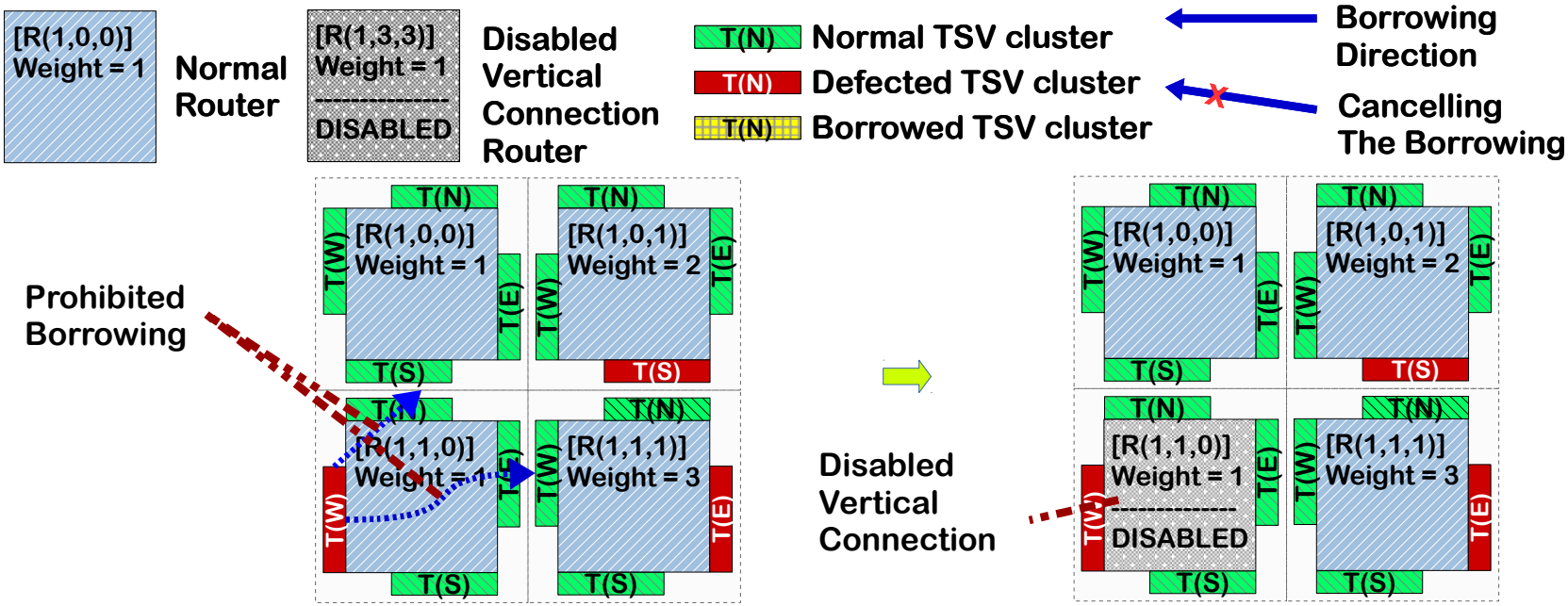

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.
- Each router having defected/borrowed TSV cluster (red/yellow) can borrow from one of its neighbors.
- The borrowed cluster must be healthy.
- The borrowed cluster must belong to the router having lower weight than the current router.

# TSV Sharing Algorithm



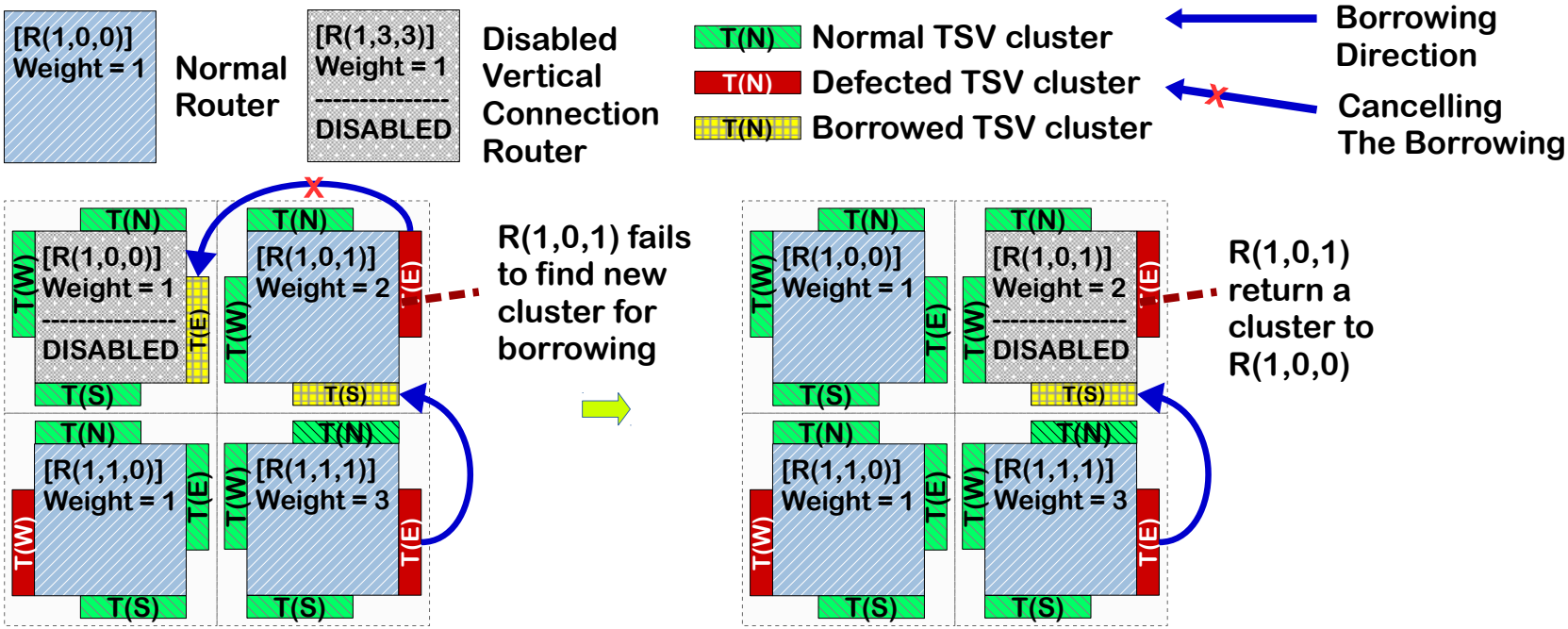

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.
- Each router having defected/borrowed TSV cluster (red/yellow) can borrow from one of its neighbors.
- The borrowed cluster must be healthy.
- The borrowed cluster must belong to the router having lower weight than the current router.
- The borrowed router must have the lowest weight than all possible candidates.

# TSV Sharing Algorithm



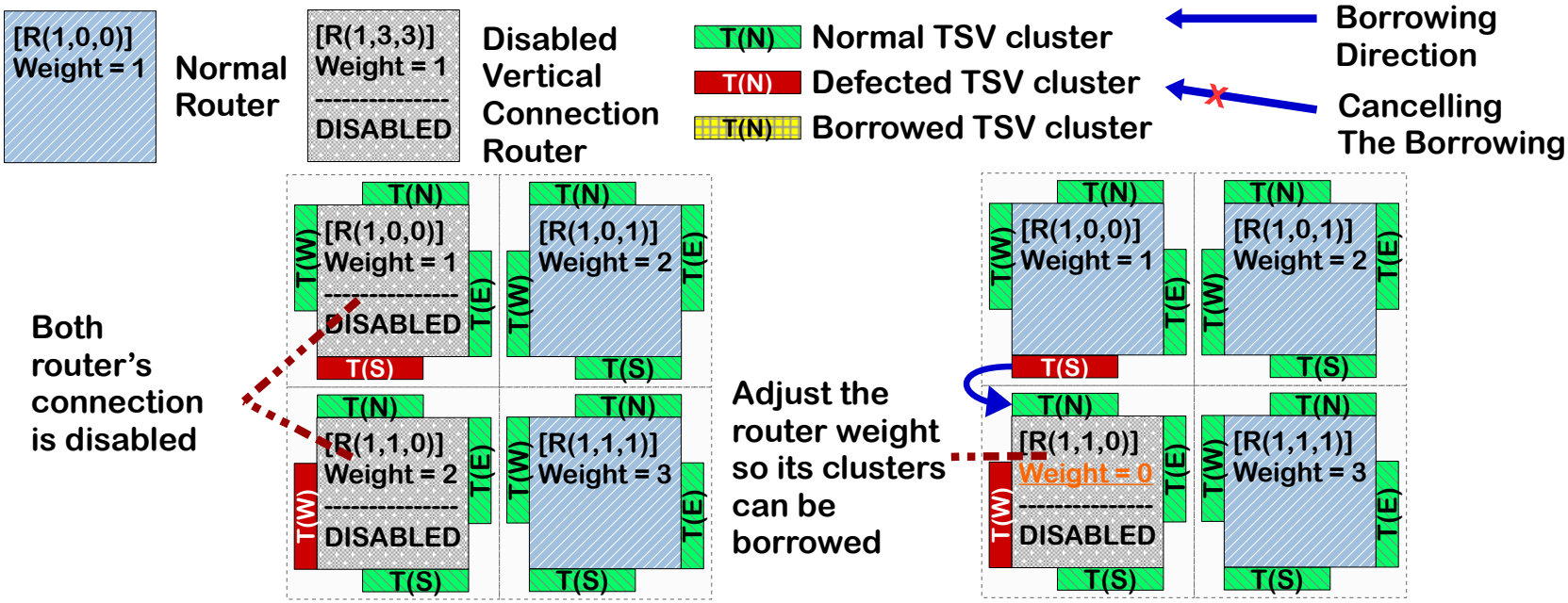

- Every router is assigned a weight value.
- TSVs of a router are organized in four clusters around it.
- Each router having defected/borrowed TSV cluster (red/yellow) can borrow from one of its neighbors.
- The borrowed cluster must be healthy.
- The borrowed cluster must belong to the router having lower weight than the current router.
- The borrowed router must have the lowest weight than all possible candidates.
- If a router fails to find a cluster to maintain its connection, its vertical connection is disabled.

# TSV Sharing Algorithm: Optimization



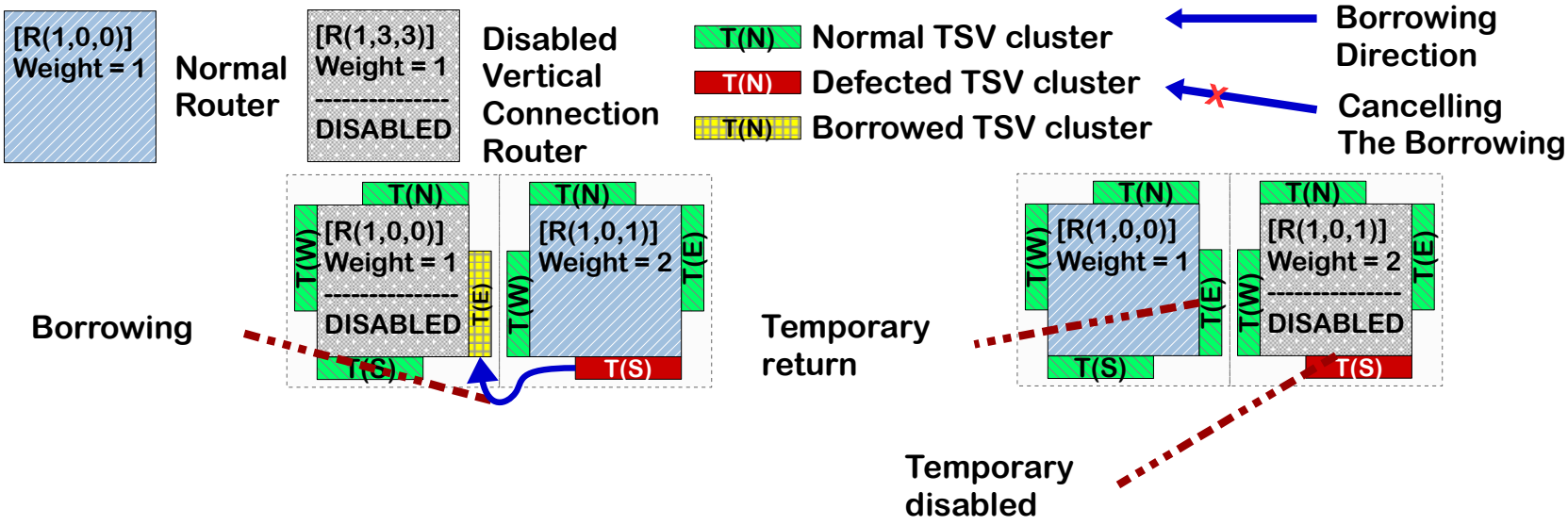

- New disabled router will return its borrowing cluster.

# TSV Sharing Algorithm: Optimization



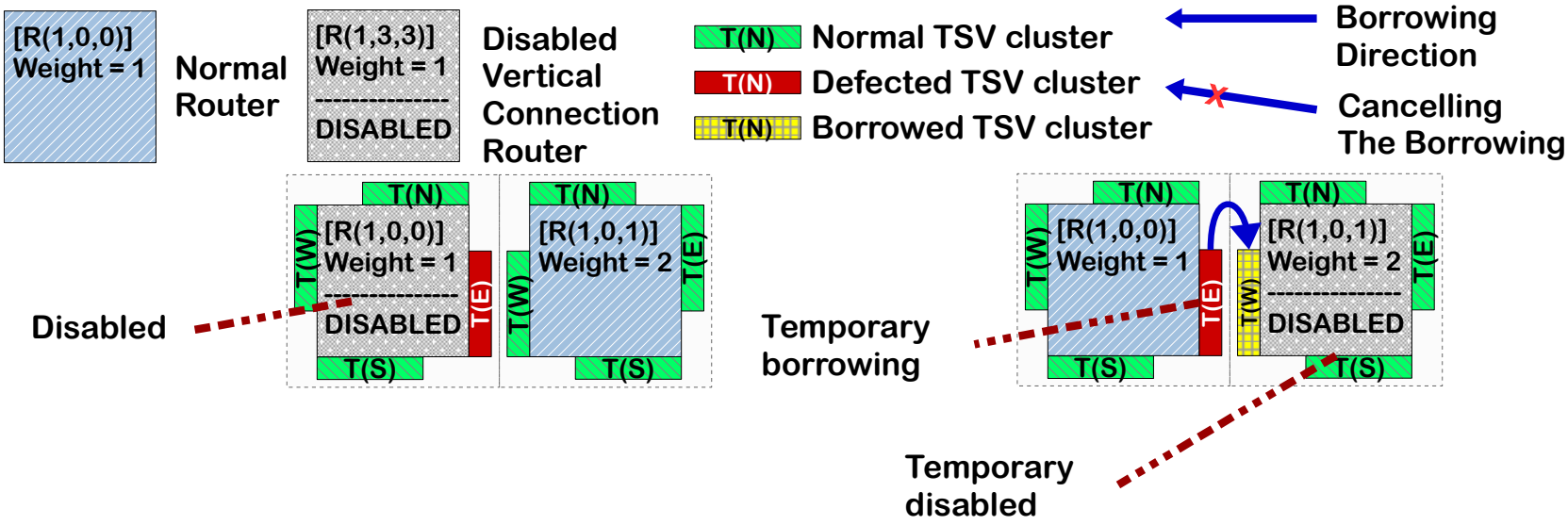

- New disabled router will return its borrowing cluster.
- Disabled router's weights are considered to be adjusted. This helps lower weight routers can gain their operations.

# TSV Sharing Algorithm: Optimization



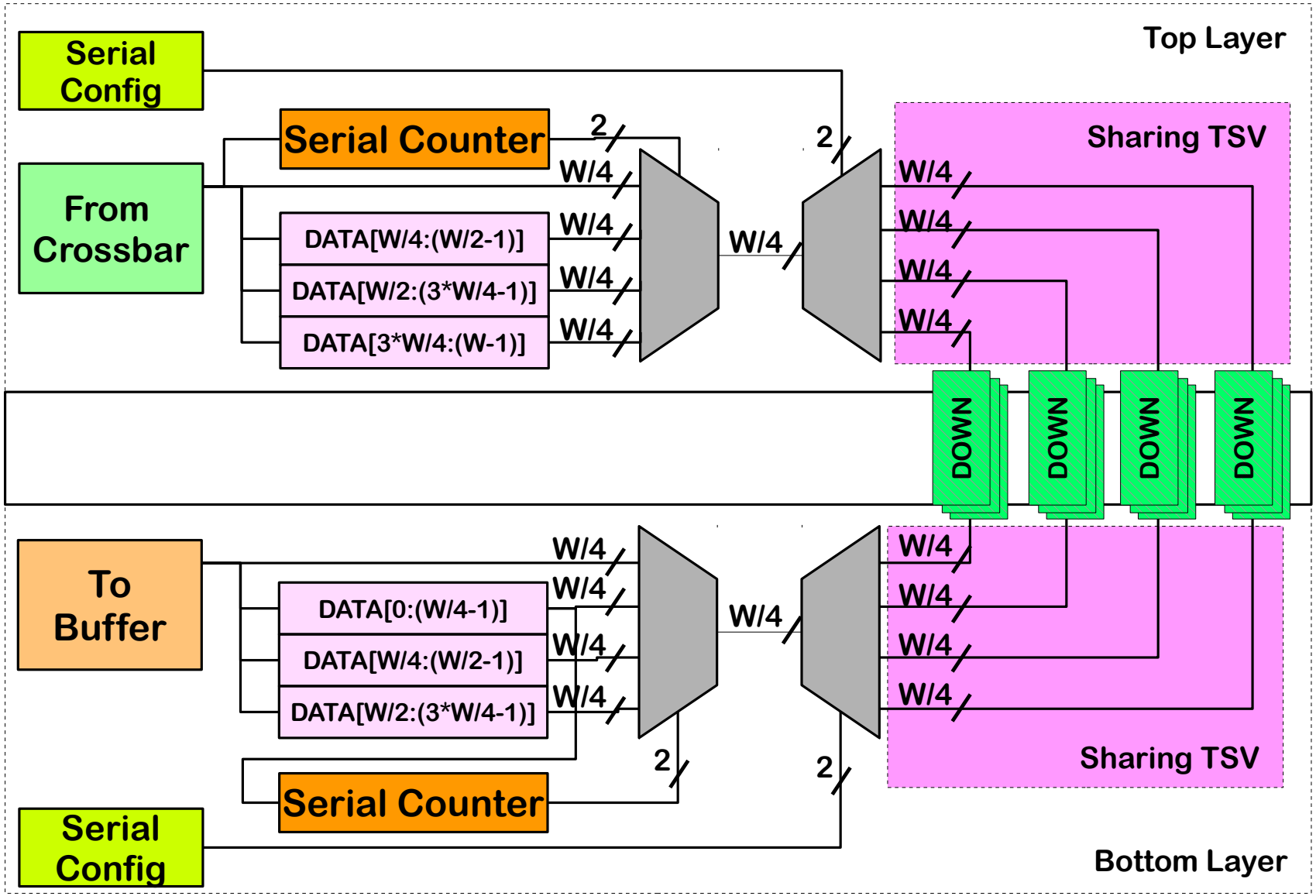

- New disabled router will return its borrowing cluster.
- Disabled router's weights are considered to be adjusted. This helps lower weight routers can gain their operations.
- **Virtual TSV:** Even being disabled, router can temporarily borrow a cluster to transmit data. *Case 1: return the cluster to its origin.*

# TSV Sharing Algorithm: Optimization



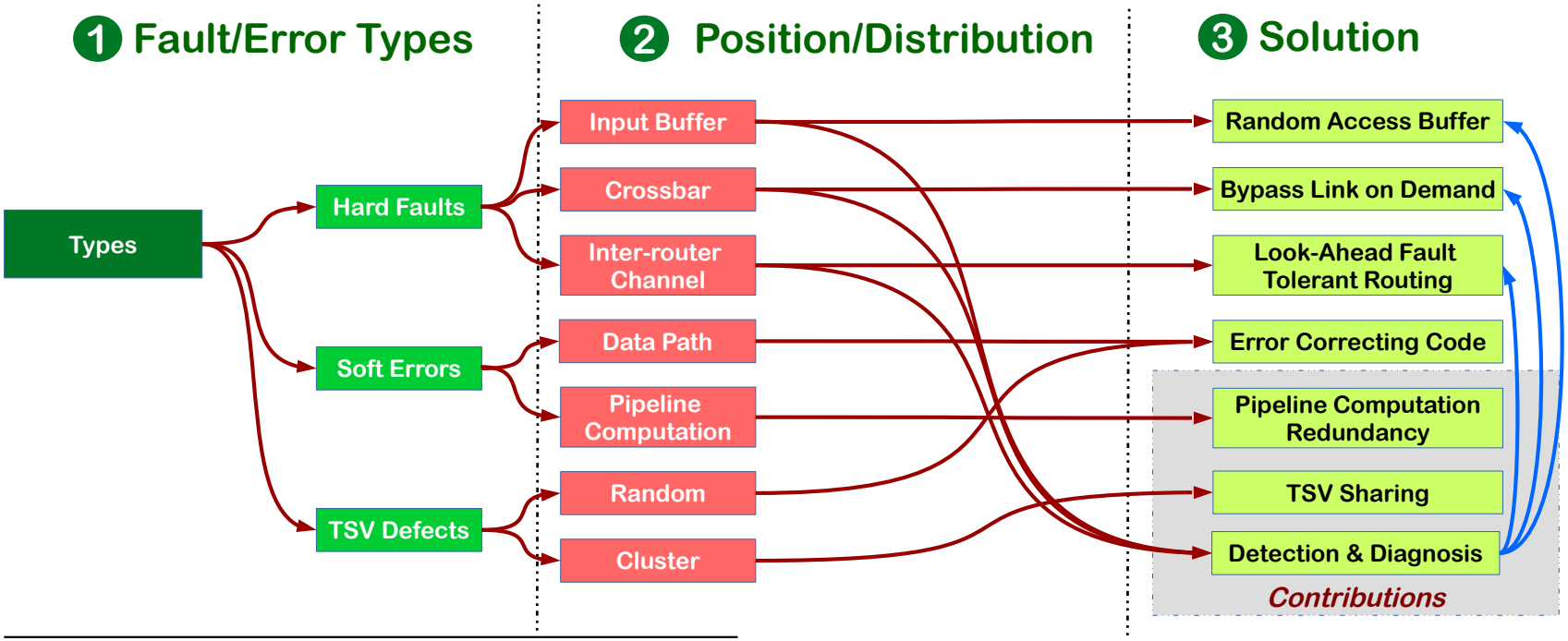

- New disabled router will return its borrowing cluster.
- Disabled router's weights are considered to be adjusted. This helps lower weight routers can gain their operations.
- **Virtual TSV:** Even being disabled, router can temporarily borrow a cluster to transmit data. *Case 2: borrow from a higher weight router.*

# TSV Sharing Algorithm: Optimization




- **Serialization:** When it is impossible to have 4 clusters, serialization technique is adopted to maintain the connection.

# Scalable Cluster-TSV Defect Tolerance

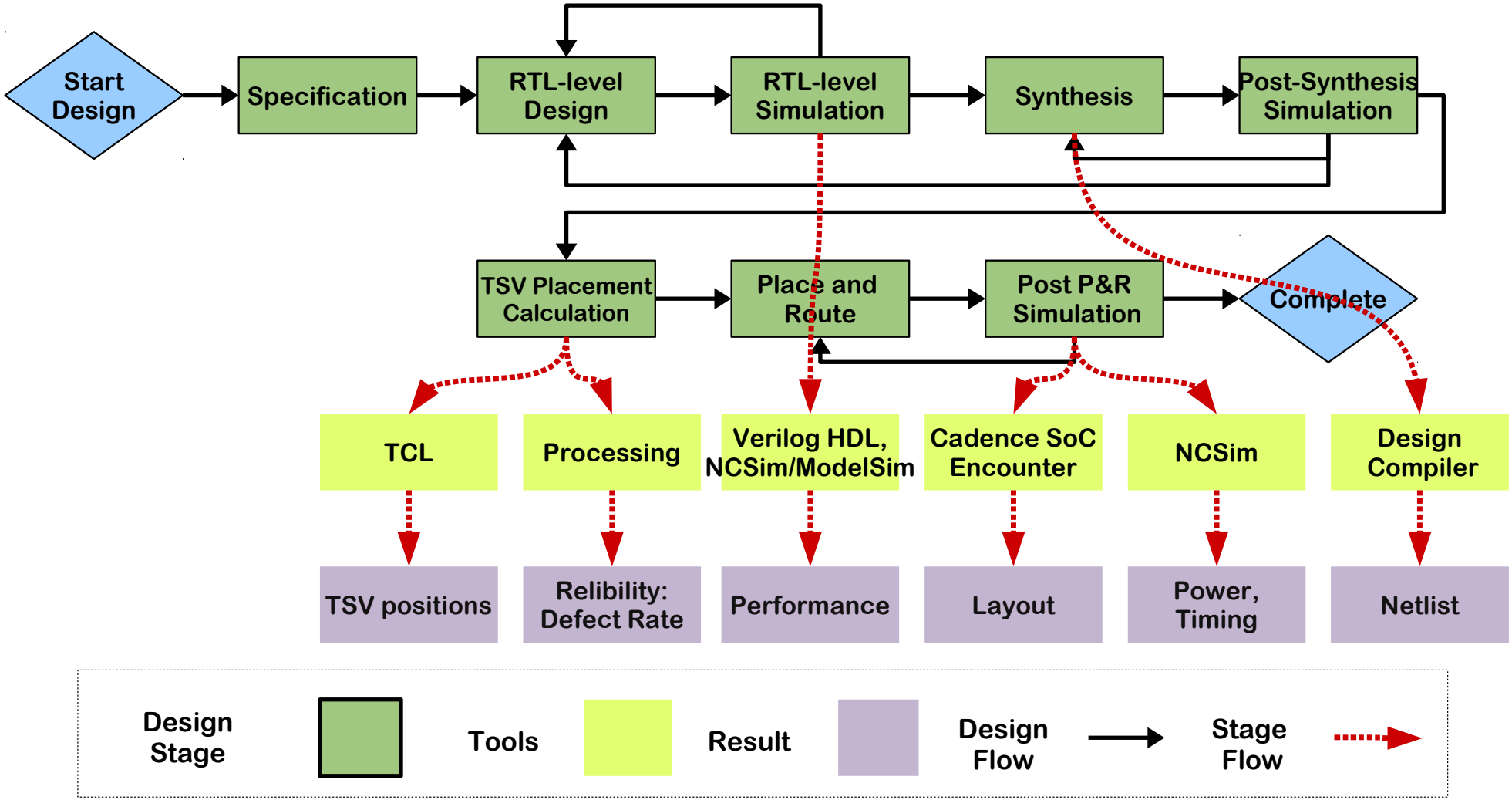



**Related paper**

- **Khanh N. Dang**, Akram Ben Ahmed, Yuichi Okuyama and Abderazek Ben Abdallah, "Scalable design methodology and online algorithm for TSV-cluster defects recovery in highly reliable 3D-NoC systems", IEEE Transactions on Emerging Topics in Computing, 2017.

# Table of Contents

# Evaluation Methodology



# Evaluation Configuration and Assumption

Table 2: Technology parameters.

| Parameter | Value |
|---|---|
| Technology | Nangate 45 $nm$ FreePDK3D45 |
| Voltage | 1.1 V |
| TSV's size | $4.06\mu m \times 4.06\mu m$ |
| TSV pitch | 10 $\mu m$ |
| Keep-out Zone | 15 $\mu m$ |

Table 3: System configurations.

| Parameter | Value |
|---|---|
| # ports | 7 |
| Topology | 3D Mesh |
| Routing Algorithm | Look-ahead routing |
| Flow Control | Stall-Go |
| Forwarding mechanism | Wormhole |
| Input buffer | 4 |
| Flit width | 44 |

Fault-rate of hard faults:

- Percentage of routers having faults.
- There is no local link is failed.

Fault-rate of soft errors:

- For ECC, percentage of ARQ per flit.
- For PCR, percentage of error per executing clock cycle.

Fault-rate of TSV-cluster defects:

- No random defect.
- TSVs randomly fail in clusters.

Table 4: Simulation configurations.

| Parameter/System | | Value |
|---|---|---|
| Network Size ($x \times y \times z$) | Matrix | $6 \times 6 \times 3$ |
| | Transpose | $4 \times 4 \times 4$ |
| | Uniform | $4 \times 4 \times 4$ |
| | Hotspot 10% | $4 \times 4 \times 4$ |
| | H.264 | $3 \times 3 \times 3$ |
| | VPOD | $3 \times 2 \times 2$ |
| | MWD | $2 \times 2 \times 3$ |
| | PIP | $2 \times 2 \times 2$ |
| Total Injected Packets | Matrix | 1,080 |
| | Transpose | 640 |
| | Uniform | 8,192 |
| | Hotspot 10% | 8,192 |
| | H.264 | 8,400 |
| | VPOD | 3,494 |
| | MWD | 1,120 |
| | PIP | 512 |
| Packet's Size | Hotspot 10% | 10 flits+10% on hostpot nodes |
| | Others | 10 flits |

# Average Packet Latency of realistic benchmarks

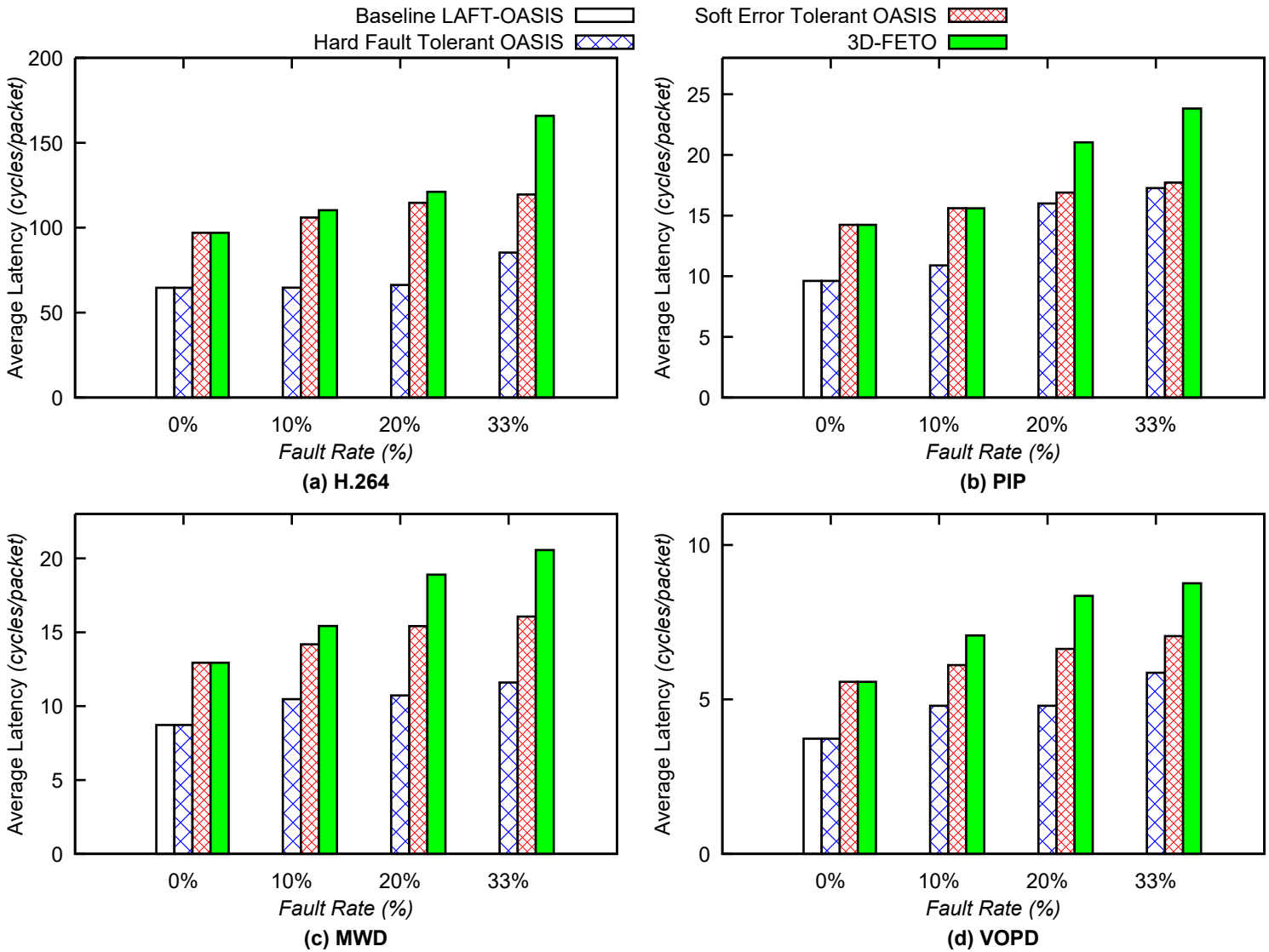



Figure 16: **Average packet latency** evaluation of the *realistic* benchmarks.

# Throughput of synthetic benchmarks

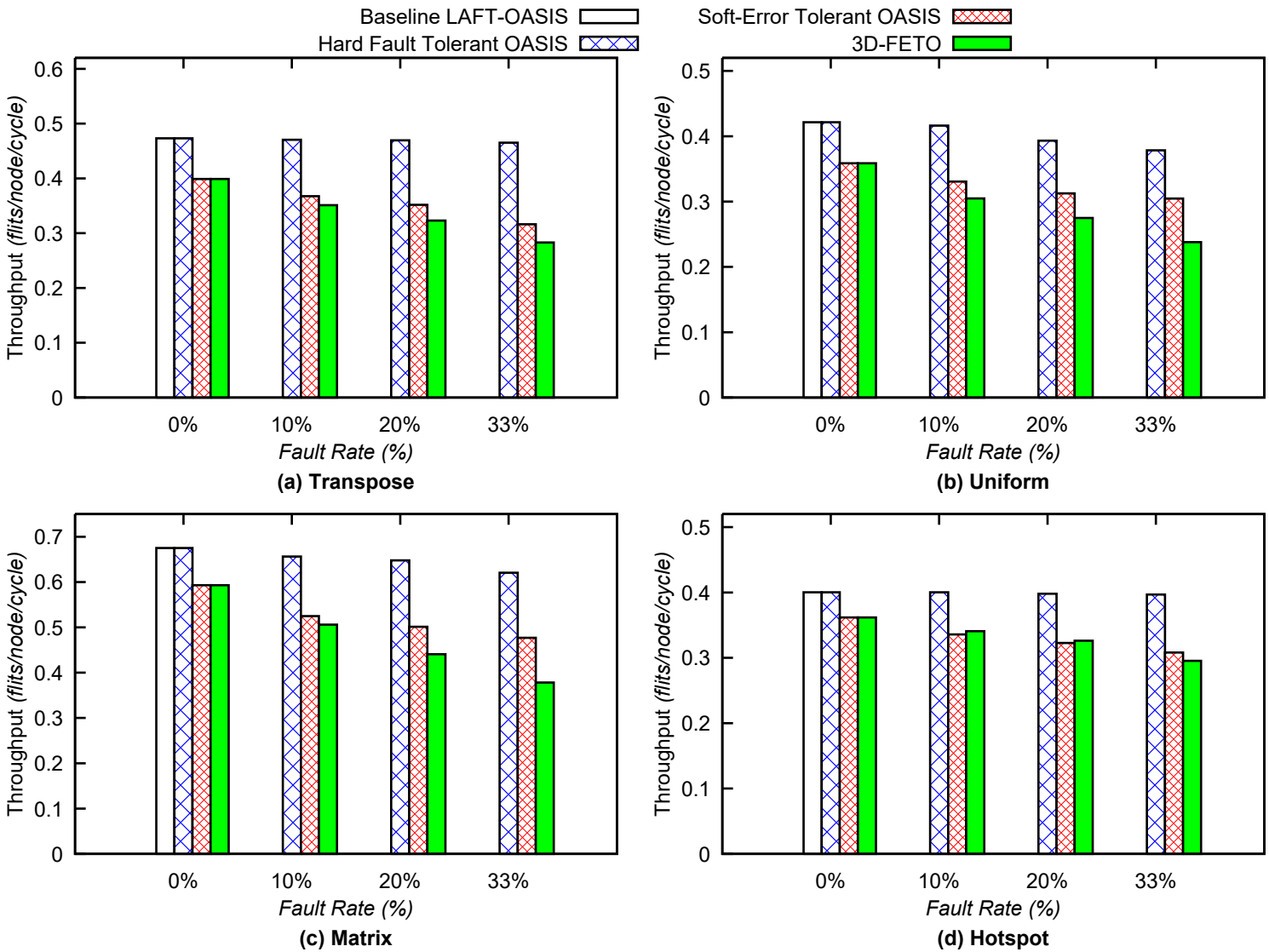



Figure 17: **Throughput** evaluation of the *synthetic* benchmarks.

# Comparison of Soft Error Tolerance

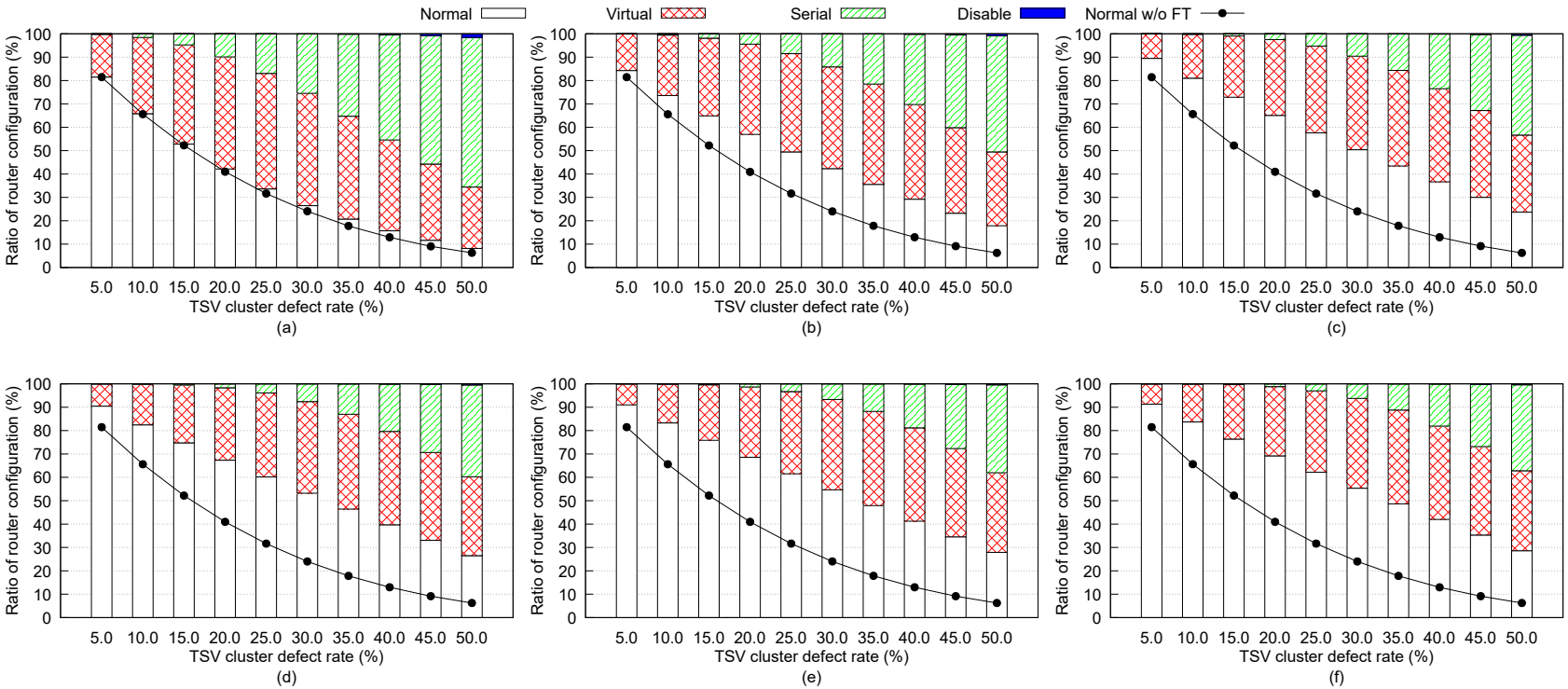
| **Model** | TMR-OASIS[4] | [8] | [9] | PCR |
|---|---|---|---|---|
| Mechanism | Majority Voting | Monitor | Monitor | Monitor |
| Area Overhead | 204.33% | 9% | 3% (average) | 54.46% |
| RAF | $\simeq 1.33$ | $\simeq 11.11$ | $\simeq 1$ (only detection) | 1.84 |
| Delay (cycle) | +0 | +0 (no fault)<br>+1 (recovery) | 0% (only detection) | +1 (redudancy)<br>+2 (recovery) |
| Fault Coverage | 100% of hard faults<br>and soft errors | design specific<br><br>(7 faults) | design specific<br><br>(13 faults) | 100% soft errors |
| Reovery method | immediately | re-execution | unsupport | re-execution |

Summary:

- Pipeline Computation Redundancy (PCR) coverage the maximum soft errors (100%) under the assumption.
- The RAF value of PCR is smaller than the technique by Yu et al. [8].
- The area overhead of PCR is still smaller than the TMR method.

[4] Triple Modular Redundancy for SA and RC

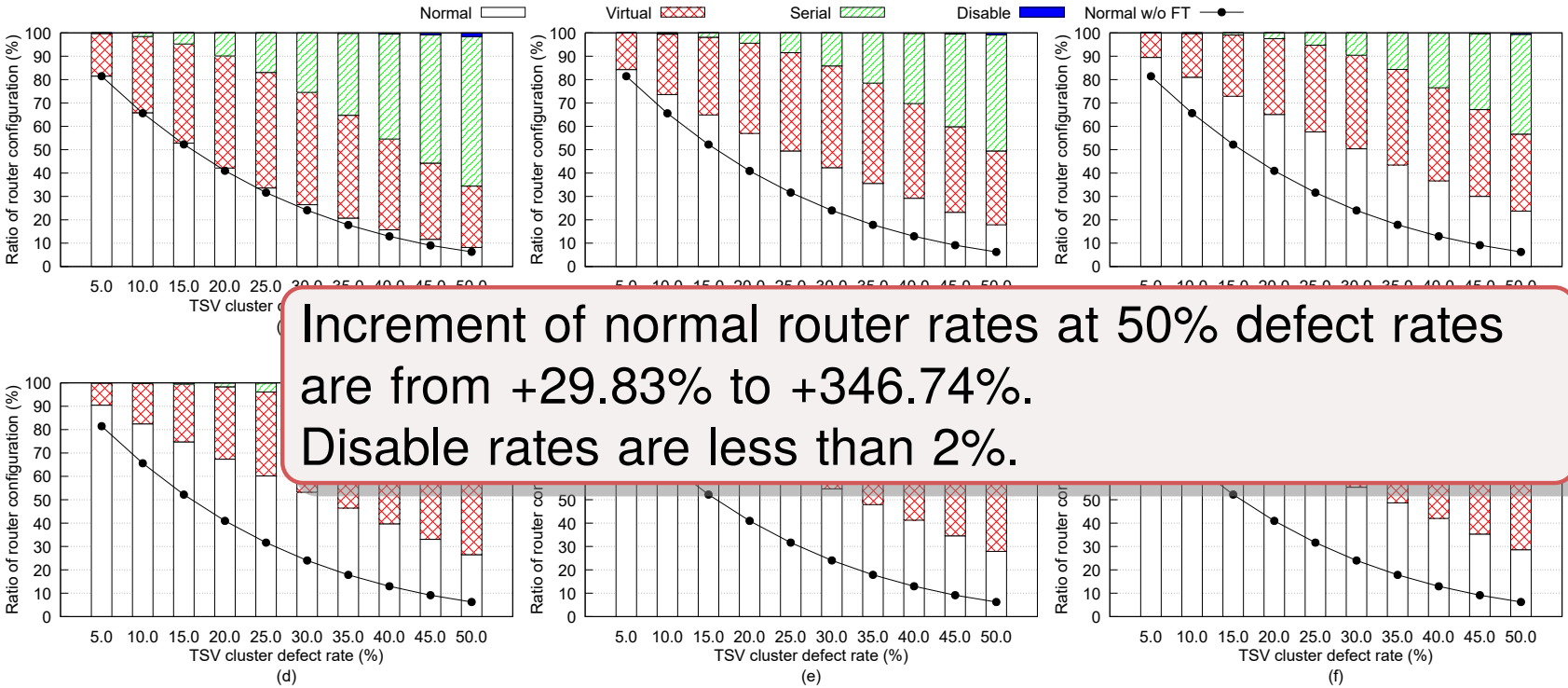
# TSV-cluster: Reliability Evaluation




Defect-rate evaluation[5]: (a) Layer size: $2 \times 2$ (4 routers, 16 TSV clusters); (b) Layer size: $4 \times 4$ (16 routers, 64 TSV clusters); (c) Layer size: $8 \times 8$ (64 routers, 256 TSV clusters); (d) Layer size: $16 \times 16$ (256 routers, 1024 TSV clusters); (e) Layer size: $32 \times 32$ (1024 routers, 4096 TSV clusters); (f) Layer size: $64 \times 64$ (4096 routers, 16384 TSV clusters).

[5]We generate 100K cases and calculate the average value.

# TSV-cluster: Reliability Evaluation




Increment of normal router rates at 50% defect rates are from +29.83% to +346.74%.
Disable rates are less than 2%.

Defect-rate evaluation[5]: (a) Layer size: $2 \times 2$ (4 routers, 16 TSV clusters); (b) Layer size: $4 \times 4$ (16 routers, 64 TSV clusters); (c) Layer size: $8 \times 8$ (64 routers, 256 TSV clusters); (d) Layer size: $16 \times 16$ (256 routers, 1024 TSV clusters); (e) Layer size: $32 \times 32$ (1024 routers, 4096 TSV clusters); (f) Layer size: $64 \times 64$ (4096 routers, 16384 TSV clusters).

[5]We generate 100K cases and calculate the average value.

# TSV Fault Tolerance Performance Evaluation

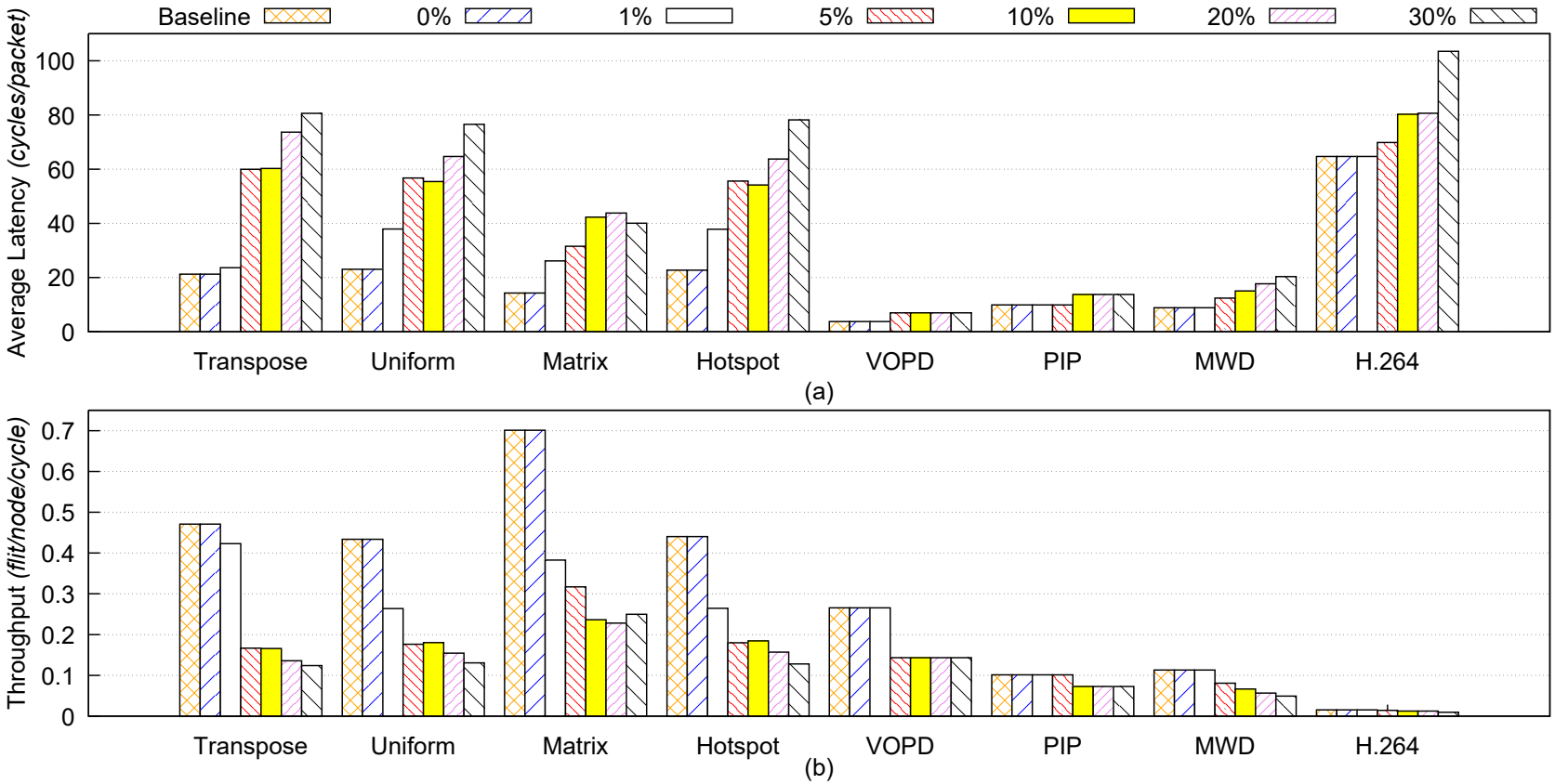



Figure 18: Evaluation result: (a) Average Packet Latency; (b) Throughput.

- Use the same configuration as 3D-FETO.
- Only cluster-TSV defects are randomly injected.

# Reliability and Area Cost Comparison of TSV Fault Tolerance

| **Model** | *TSV Network* [10] | | | | |
|---|---|---|---|---|---|
| **Technology** | 65 *nm* | | | | |
| **#TSV** | 1000 | | | | |
| **Configuration** | 4:2 | 8:2 | $4 \times 4 : 8$ | $8 \times 8 : 16$ | $16 \times 16 : 32$ |
| **#Spare TSV** | 512 | 256 | 512 | 256 | 128 |
| **45$nm$ Arbiter Area ($\mu m^2$)** | 372 [2] | 744 [2] | 1,116 [2] | 1,116 [2] | 1,116 [2] |
| **Average Area/TSV ($\mu m^2$)** | 151.572 | 126.244 | 152.316 | 126.716 | 128.03 |
| **Reliability** | 100% | 99% | 100% | 100% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 0.01\%, \alpha = 2)$ [4] | | | | |

| **Model** | *TSV Grouping* [11] | | | *This work* | |
|---|---|---|---|---|---|
| **Technology** | N/A | | | 45 *nm* | |
| **#TSV** | 6000 | | | 8448 | |
| **Configuration** | 4:4 | 8:4 | 20:5 | $11 \times 4 \times 4$:0 | |
| **#Spare TSV** | 6000 | 3000 | 1500 | 0 | |
| **45$nm$ Arbiter Area ($\mu m^2$)** | 11,160 [1] | 11,160 [1] | 12,555 [1] | 434,784 [3] | |
| **Average Area/TSV ($\mu m^2$)** | 113.916 | 151.86 | 127.09 | 151.47 | |
| **Reliability** | 100% | | | 98.11% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 1\%, \alpha = 2)$ [4] | | | $\delta_c = 50\%$ [4] | $\delta_c = 1\%$ [4] |

[1] The authors use 2:1 multiplexers [11]. For comparison, we use the area cost of multiplexer from Nangate 45$nm$ [12] (MUX2_X1: 0.186$\mu m^2$) .

[2] The authors use 1-to-3 multiplexers [10] which consists of two MUX2_X1 multiplexers ($2 \times 0.186 \mu m^2$ [12]).

[3] For fair comparisons, our arbiter only consists of the TSV sharing and serialization modules as shown in Table 6.

[4] $\delta$: defect-rate. $\alpha$: parameter of Poisson distribution [10, 11]. $\delta_c$: cluster fault rate, $\delta_{TSV}$: TSV fault rate.

# Reliability and Area Cost Comparison of TSV Fault Tolerance

| **Model** | *TSV Network* [10] | | | | |
|---|---|---|---|---|---|
| **Technology** | 65 *nm* | | | | |
| **#TSV** | 1000 | | | | |
| **Configuration** | 4:2 | 8:2 | $4 \times 4 : 8$ | $8 \times 8 : 16$ | $16 \times 16 : 32$ |
| **#Spare TSV** | 512 | 256 | 512 | 256 | 128 |
| **45*nm* Arbiter Area ($\mu m^2$)** | 372 [2] | 744 [2] | 1,116 [2] | 1,116 [2] | 1,116 [2] |
| **Average Area/TSV ($\mu m^2$)** | 151.572 | 126.244 | 152.316 | 126.716 | 128.03 |
| **Reliability** | 100% | 99% | 100% | 100% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 0.01\%, \alpha = 2)$[4] | | | | |

| **Model** | *TSV Grouping* [11] | | | *This work* | |
|---|---|---|---|---|---|
| **Technology** | N/A | | | 45 *nm* | |
| **#TSV** | 6000 | | | 8448 | |
| **Configuration** | 4:4 | 8:4 | 20:5 | $11 \times 4 \times 4$:0 | |
| **#Spare TSV** | 6000 | 3000 | 1500 | 0 | |
| **45*nm* Arbiter Area ($\mu m^2$)** | 11,160 [1] | 11,160[1] | 12,555[1] | 434,784[3] | |
| **Average Area/TSV ($\mu m^2$)** | 113.916 | 151.86 | 127.09 | 151.47 | |
| **Reliability** | 100% | | | 98.11% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 1\%, \alpha = 2)$[4] | | | $\delta_c = 50\%$[4] | $\delta_c = 1\%$[4] |

[1] The authors use 2:1 multiplexers [11]. For comparison, we use the area cost of multiplexer from Nangate 45*nm* [12] ($0.186\mu m^2$ [12]).

[3] as shown in Table 6.

SV fault rate.

The average area of this work is similar with some cases but worse than the best case. However, this work consists of an online arbitration.

# Reliability and Area Cost Comparison of TSV Fault Tolerance

| **Model** | *TSV Network* [10] | | | | |
|---|---|---|---|---|---|
| **Technology** | 65 *nm* | | | | |
| **#TSV** | 1000 | | | | |
| **Configuration** | 4:2 | 8:2 | $4 \times 4 : 8$ | $8 \times 8 : 16$ | $16 \times 16 : 32$ |
| **#Spare TSV** | 512 | 256 | 512 | 256 | 128 |
| **45*nm* Arbiter Area ($\mu m^2$)** | 372 [2] | 744 [2] | 1,116 [2] | 1,116 [2] | 1,116 [2] |
| **Average Area/TSV ($\mu m^2$)** | 151.572 | 126.244 | 152.316 | 126.716 | 128.03 |
| **Reliability** | 100% | 99% | 100% | 100% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 0.01\%, \alpha = 2)^4$ | | | | |

| **Model** | *TSV Grouping* [11] | | | *This work* | |
|---|---|---|---|---|---|
| **Technology** | N/A | | | 45 *nm* | |
| **#TSV** | 6000 | | | 8448 | |
| **Configuration** | 4:4 | 8:4 | 20:5 | $11 \times 4 \times 4$:0 | |
| **#Spare TSV** | 6000 | 3000 | 1500 | 0 | |
| **45*nm* Arbiter Area ($\mu m^2$)** | 11,160 [1] | 11,160[1] | 12,555[1] | 434,784[3] | |
| **Average Area/TSV ($\mu m^2$)** | 113.916 | 151.86 | 127.09 | 151.47 | |
| **Reliability** | 100% | | | 98.11% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 1\%, \alpha = 2)^4$ | | | $\delta_c = 50\%^4$ | $\delta_c = 1\%^4$ |

[1] The authors use 2:1 multiplexers [11]. For comparison, we use the area cost of multiplexer from Nangate 45*nm* [12] (MUX2_X1: 0.186$\mu m^2$) .

0.186$\mu m^2$ [12]).
as shown in Table 6.
SV fault rate.

In term of number of TSV, this work doesn't requires any redundancies.

# Reliability and Area Cost Comparison of TSV Fault Tolerance

| **Model** | *TSV Network* [10] | | | | |
|---|---|---|---|---|---|
| **Technology** | 65 *nm* | | | | |
| **#TSV** | 1000 | | | | |
| **Configuration** | 4:2 | 8:2 | $4 \times 4 : 8$ | $8 \times 8 : 16$ | $16 \times 16 : 32$ |
| **#Spare TSV** | 512 | 256 | 512 | 256 | 128 |
| **45*nm* Arbiter Area ($\mu m^2$)** | 372 [2] | 744 [2] | 1,116 [2] | 1,116 [2] | 1,116 [2] |
| **Average Area/TSV ($\mu m^2$)** | 151.572 | 126.244 | 152.316 | 126.716 | 128.03 |
| **Reliability** | 100% | 99% | 100% | 100% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 0.01\%, \alpha = 2)$[4] | | | | |

| **Model** | *TSV Grouping* [11] | | | *This work* | |
|---|---|---|---|---|---|
| **Technology** | N/A | | | 45 *nm* | |
| **#TSV** | 6000 | | | 8448 | |
| **Configuration** | 4:4 | 8:4 | 20:5 | $11 \times 4 \times 4$:0 | |
| **#Spare TSV** | 6000 | 3000 | 1500 | 0 | |
| **45*nm* Arbiter Area ($\mu m^2$)** | 11,160 [1] | 11,160[1] | 12,555[1] | 434,784[3] | |
| **Average Area/TSV ($\mu m^2$)** | 113.916 | 151.86 | 127.09 | 151.47 | |
| **Reliability** | 100% | | | 98.11% | 100% |
| **Fault Assumption** | $(\delta_{TSV} = 1\%, \alpha = 2)$[4] | | | $\delta_c = 50\%$[4] | $\delta_c = 1\%$[4] |

[1] The authors use 2:1 multiplexers [11]. For comparison, we use the area cost of multiplexer from Nangate 45*nm* [12] ($0.186\mu m^2$ [12]).

[3] as shown in Table 6.

SV fault rate.

In term of reliability, this work provide extremely high working rate: 98.11% of routers even with 50% of clusters are defected..

# Hardware Design Result (1/2)

Table 5: Design parameters.

| Parameter | Value |
|---|---|
| Technology | Nangate 45 $nm$ FreePDK3D45 |
| Voltage | 1.1 V |
| Chip's size | $865\mu m \times 865\mu m$ |
| TSV's size | $4.06\mu m \times 4.06\mu m$ |
| TSV pitch | 10 $\mu m$ |
| Keep-out Zone | 15 $\mu m$ |

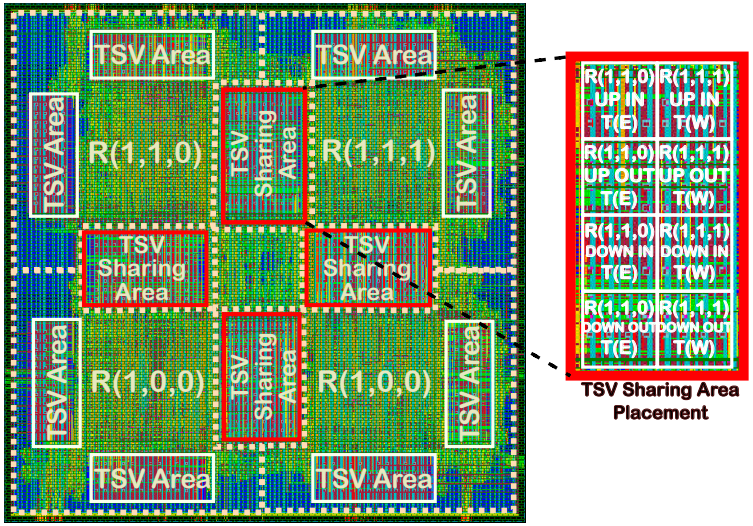



Figure 19: Single layer layout illustrating the TSV sharing areas (red boxes). The layout size is $865\mu m \times 865\mu m$. The sharing TSV area are the red boxes. Each sharing area has 8 clusters for 4 ports and 2 routers.

- Estimated 3D-NoC router layout area: $423.5\mu m \times 423.5\mu m$.
- Estimated 3D-NoC ($X \times Y \times Z$) layout area: $Z$ layers $\times$ $X \times 423.5\mu m \times Y \times 423.5\mu m$
- E.g.: *MWD* ($X = 2, Y = 2, Z = 3$) needs 3D-NoC with layout of $3 \times 865\mu m \times 865\mu m$

# Hardware Design Result (2/2)

Table 6: Hardware complexity of a single router.

| Model | | Area ($\mu m^2$) | Power (mW) | | | Speed (Mhz) |
|---|---|---|---|---|---|---|
| | | | Static | Dynamic | Total | |
| Baseline router [13] | | 18,873 | 5.1229 | 0.9429 | 6.0658 | 925.28 |
| 3D-FTO [6] router[6] | | 19,143 | 6.4280 | 1.1939 | 7.6219 | 909.09 |
| Soft Error Tolerance router[7] | | 27,457 | 9.7314 | 2.6710 | 12.4024 | 625.00 |
| 3D-FETO router[8] | | 29,516 | 10.0819 | 2.7839 | 12.8658 | 613.50 |
| Final router[9] | Router | 29,780 | 10.017 | 2.2574 | 12.3144 | 613.50 |
| | Serialization | 3,318 | 0.9877 | 0.2807 | 1.2684 | - |
| | TSV Sharing | 5,740 | 0.7863 | 0.2892 | 1.0300 | - |
| | Total | 38,838 | 11.7910 | 2.8273 | 14.6128 | 537.63 |

[6]This router consists of RAB, BLoD, and LAFT.

[7]This router consists of ECC and PCR.

[8]This router consists all soft error and hard fault tolerant techniques: RAB, BLoD, LAFT, ECC and PCR.

[9]This router is 3D-FETO with TSV management.

# Table of Contents

# Conclusion

## Conclusion:

- We provide a set of on-chip communication fault-resilient adaptive architectures and algorithms for 3D-NoC IC technologies.

## Future Work:

- Further research is needed about the **thermal awareness** in terms of design, routing and reliability.
- An in-depth study on **stress issues** is also necessary to understand the potential defects.
- Fault-tolerance also need to be covered in application layers with a cross-layers protocol.

# Related publications

- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, “A Low-overhead Soft-Hard Fault Tolerant Architecture, Design, and Management Scheme for Reliable High-performance Many-core 3D-NoC Systems”, *The Journal of Supercomputing*, Volume 73, Issue 6, pp 2705–2729, 2017.
- **Khanh N. Dang**, Akram Ben Ahmed, Xuan-Tu Tran, Yuichi Okuyama and Abderazek Ben Abdallah, “A Comprehensive Reliability Assessment of Fault-Resilient Network-on-Chip Using Analytical Model”, *IEEE Transactions on Very Large Scale Integration Systems*, Volume 25, Issue 11, pp 3099-3112, 2017.
- **Khanh N. Dang**, Akram Ben Ahmed, Yuichi Okuyama and Abderazek Ben Abdallah, “Scalable design methodology and online algorithm for TSV-cluster defects recovery in highly reliable 3D-NoC systems”, *IEEE Transactions on Emerging Topics in Computing*, 2017.
- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, “Reliability Assessment and Quantitative Evaluation of Soft-Error Resilient 3D Network-on-Chip Systems”, *The IEEE 25th Asian Test Symposium (ATS)*, pp. 161-166, Hiroshima, Japan, November 21-24, 2016.
- **Khanh N. Dang**, Yuichi Okuyama, and Abderazek Ben Abdallah, “Soft-error resilient network-on-chip for safety-critical applications”, *The 2016 International Conference on IC Design and Technology (ICICDT)*, pp. 1-4, Ho Chi Minh City, Vietnam, June 27-29, 2016.
- **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama, Abderazek Ben Abdallah, and Xuan-Tu Tran, “Soft-error resilient 3d network-on-chip router”, *The 2015 IEEE 7th International Conference on Awareness Science and Technology (iCAST)*, pp. 84-90 Qinhuangdao, China, September 22-24, 2015.

# References (1/2)


[1] C. Batten, “Energy-efficient parallel computer architecture,” 2014.

[2] M. A. El-Moursy and E. G. Friedman, “Shielding effect of on-chip interconnect inductance,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 3, pp. 396–400, 2005.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, “Power challenges may end the multicore era,” *Communications of the ACM*, vol. 56, no. 2, pp. 93–102, 2013.

[4] B. Vaidyanathan, W.-L. Hung, F. Wang, Y. Xie, V. Narayanan, and M. J. Irwin, “Architecting microprocessor components in 3d design space,” in *VLSI Design, 2007. Held Jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, pp. 103–108, IEEE, 2007.

[5] M.-Y. Hsiao, “A class of optimal minimum odd-weight-column sec-ded codes,” *IBM Journal of Research and Development*, vol. 14, no. 4, pp. 395–401, 1970.

[6] A. B. Ahmed and A. B. Abdallah, “Adaptive fault-tolerant architecture and routing algorithm for reliable many-core 3D-NoC systems,” *Journal of Parallel and Distributed Computing*, vol. 93-94, pp. 30–43, 2016.

[7] A. Ben Ahmed and A. Ben Abdallah, “Architecture and design of high-throughput, low-latency, and fault-tolerant routing algorithm for 3D-network-on-chip (3D-NoC),” *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1507–1532, 2013.

[8] Q. Yu, M. Zhang, and P. Ampadu, “Addressing network-on-chip router transient errors with inherent information redundancy,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 4, pp. 105:1–105:21, 2013.

[9] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, “NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures,” in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60–71, December 2012.

[10] L. Jiang, F. Ye, Q. Xu, K. Chakrabarty, and B. Eklow, “On effective and efficient in-field TSV repair for stacked 3D ICs,” in *Proceedings of the 50th Annual Design Automation Conference*, p. 74, ACM, 2013.

# References (2/2)


[11] Y. Zhao, S. Khursheed, and B. M. Al-Hashimi, “Cost-effective TSV grouping for yield improvement of 3D-ICs,” in *Asian Test Symposium (ATS)*, pp. 201–206, IEEE, 2011.

[12] NanGate Inc., “Nangate Open Cell Library 45 nm,” 2016.

[13] A. Ben Ahmed and A. Ben Abdallah, “LA-XYZ: low latency, high throughput look-ahead routing algorithm for 3D network-on-chip (3D-NoC) architecture,” in *IEEE 6th International Symposium on Embedded Multicore Socs (MCSoC)*, pp. 167–174, IEEE, September 2012.

[14] J. P. Hayes, “A graph model for fault-tolerant computing systems,” *IEEE Transactions on Computers*, vol. 100, no. 9, pp. 875–884, 1976.

[15] A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, J. Hu, and G. Chen, “A reliable routing architecture and algorithm for NoCs,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 5, pp. 726–739, 2012.

[16] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, “Bulletproof: A defect-tolerant CMP switch architecture,” in *The Twelfth International Symposium on High-Performance Computer Architecture*, pp. 5–16, IEEE, 2006.

[17] F. Ye and K. Chakrabarty, “TSV open defects in 3D integrated circuits: Characterization, test, and optimal spare allocation,” in *Proceedings of the 49th Annual Design Automation Conference*, pp. 1024–1030, ACM, 2012.

[18] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, *et al.*, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *Proceedings 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pp. 7–18, IEEE, 2003.

**Thank you for your attention!**

# Backup slides

# Fault Assumption

## Hard Fault Assumption

- Hard faults only occur in the following positions: *input buffer*, *crossbar* and *inter-router channel*.
- Hard faults are modeled as stuck-at faults where the output values of faulty gates are always ‘0’ or ‘1’.
- This type of faults occurs permanently.

## Soft Error Assumption

- Soft errors can occur in data path or in the routing arbitrator (Next Port Computing and Switch Allocator).
- Soft errors are modeled as stuck-at faults where the output values of faulty gates are always ‘0’ or ‘1’.
- This type of faults only occurs in a single clock cycle.

# Reliability Assessment Methodology

**Dividing:**

1. A Network-on-Chip consists of $N_R$ routers.
2. A router is divided into several modules.

**Conquering:**

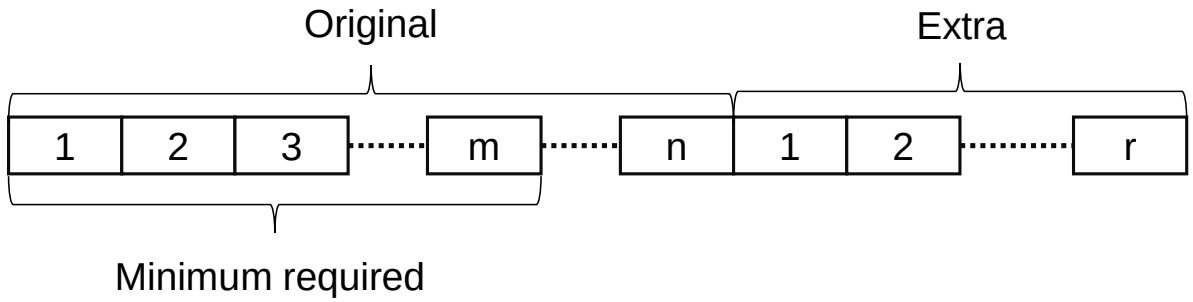
1. For each module of a router, analyze it using one of the following model:
   - *Model 0*: non fault-tolerant module - use the fault assumption (Eq. 10).
   - *Model 1*: spare or reconfiguration module.
   - *Model 2*: fault reduced module.
   - *Model 3*: module with fault-tolerance support.

**Merging:**

1. A router reliability is obtained by *Router Merging*.
2. A network reliability is obtained by *Network Merging*.

# Model 1: spare/reconfiguration

This strategy handles faults using spare modules or by reconfiguring.




- Module has $n$ separate identical parts.
- Module can function with at least $m$ parts.
- Extra $r$ spare parts are added in the design stage.
- $f$ is the number of parts that are faulty in a state.

**Lemma 1:** The RAF values can be calculated as follows:

$$\mathrm{RAF}_{conv.} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} = \sum_{i=m}^{n+r} \frac{n}{i} = 1 + \sum_{i=m}^{n-1} \frac{n}{i} + \sum_{i=n+1}^{n+r} \frac{n}{i} \quad (1)$$

# Model 2: fault reduced

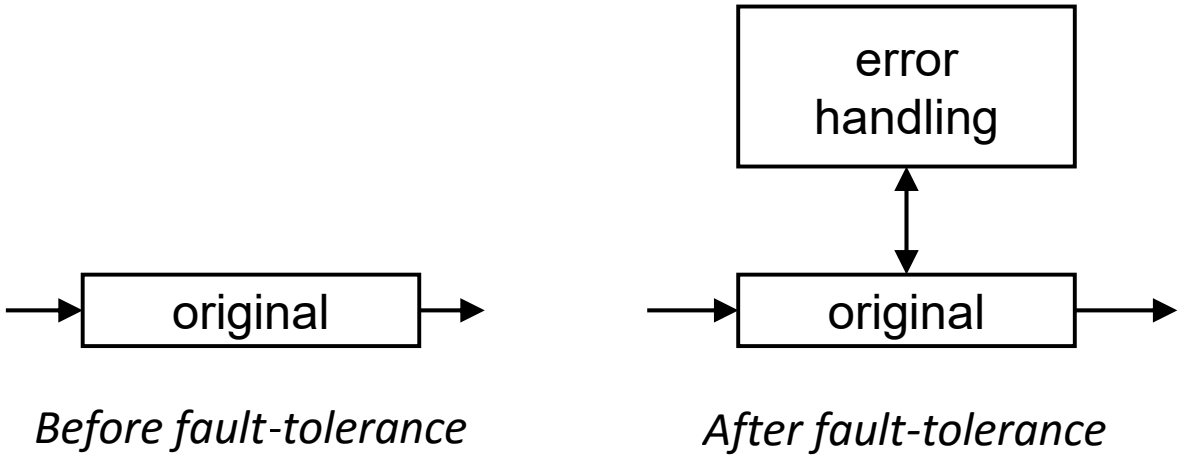
- For helping the platform being compatible with other reliability assessments, this model can integrate them together.
- With a fault reduction value $f_{FT}$ given by the other technique, the new fault rate is obtained by Eq. 2.

$$\lambda_{FT} = f_{FT}\lambda_{original} \tag{2}$$

The RAF value can be obtained by:

$$\mathrm{RAF}_{FT} = 1/f_{FT} \tag{3}$$

# Model 3: module with fault-tolerance




Because the fault-tolerance technique may require additional modules for checking and correcting faults. These correction modules also add fault-rates.

# Model 3: module with fault-tolerance

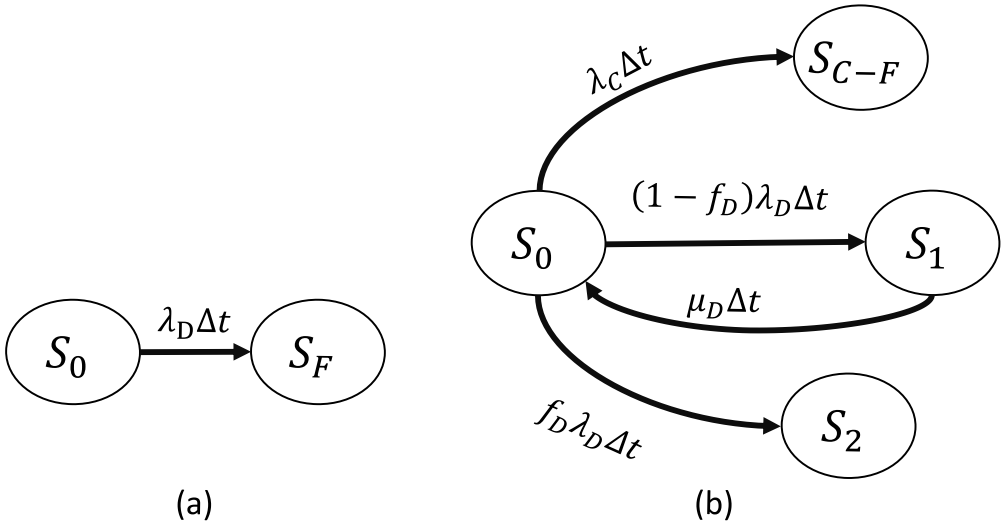



Figure 20: A simplified Markov-state reliability model for (a) the original system; (b) the fault-tolerant (FT) system.

**Lemma 2:** The RAF value can be then expressed as:

$$\text{RAF}_{FT} = f_D + \frac{\lambda_C}{\lambda_D} \tag{4}$$

Where

- $\lambda_D$ is the fault-rate of the original system (D).
- $\lambda_C$ is the fault-rate of the repair module of the FT system.
- $f_D$ is the fault reducing value by applying the fault-tolerance mechanism.

# Summary of Models

In this proposal, we analyze the reliability for four types of model:

- Model 0: no fault-tolerance.
- Model 1: spare or reconfiguration model which is the most common fault-tolerant method.
- Model 2: fault reduced model which this platform compatible with other methods.
- Model 3: a typical fault-tolerant module. Most of the works ignore the additional fault rates given by the correction module. So, we consider them for more accurate result.

For each sub-module in the NoC systems, it reliability is analyzed by using one of four model. Later, the system reliability is synthesized (*Merging*).

# Reliability Assessment Methodology

**Dividing:**

1. A Network-on-Chip consists of $N_R$ routers.
2. A router is divided into several modules.

**Conquering:**

1. For each module of a router, analyze it using one of the following model:
   - *Model 0*: non fault-tolerant module - use the fault assumption (Eq. 10).
   - *Model 1*: spare or reconfiguration module.
   - *Model 2*: fault reduced module.
   - *Model 3*: module with fault-tolerance support.

**Merging:**

1. A router reliability is obtained by *Router Merging*.
2. A network reliability is obtained by *Network Merging*.

# Merging

## Router

The fault rate of a router is summarized from its own $N$ sub-modules ($M_i$):

$$\lambda_{router} = \sum_{i=1}^{N} f_{M_i} \lambda_{M_i} \tag{5}$$

## Network

The fault rate of a network is summarized from three parts: (1) the local connection (router-PE), (2) the routing paths inside network and (3) other modules inside routers:

$$\lambda_{network} = \lambda_{local} + \lambda_{transmitting-path} + \lambda_{others} \tag{6}$$

# Network Merging

Three main parts of network are:

- $\lambda_{local} = N_R \times (2\lambda_{1-channel} + \lambda_{input-buffer})$ is the fault-rate of all local connections.
- $\lambda_{transmitting-path} = \lambda_{RTR} \times N_{RTR}$. $\lambda_{RTR}$ is the fault-rate of all router-to-router (RTR) connections. $N_{RTR}$ is the number of used RTR connections.
- $\lambda_{others}$ is given by the fault-rates of other parts (non-routing parts) of routers.

# Reliability of transmitting path

- A router-to-router (RTR) connection consists of: an input buffer, a crossbar link and an intra-router channel[10].
- For transmitting path reliability, we use the *k-failure* [14] model: a router is disconnected at the presence of $k$ failures[11].
- For 3D-NoCs, we the $k$ value depends on the position of the router and the efficiency of the fault-tolerant algorithm.[12].

---

[10] The control logic is counted as other modules in the network equation (Eq. 6).

[11] Note: Not only the *k-failure* model, any reliability network assessment can be applied to obtain the $\lambda_{transmitting-path}$.

[12] Conner routers: k=3, middle routers: k= 6

# Pipeline Computation Redundancy Timeline

| Cycle | BW | NPC/SA | (PCR) | CT |
|---|---|---|---|---|
| 1st | $flit(1)$ | $idle$ | $idle$ | $idle$ |
| 2nd | $flit(2)$ | $flit(1), time(1)$ | $wait$ | $idle$ |
| 3rd | $flit(3)$ | $flit(1), time(2)$ | $\mathbf{c(1)}$ | $c(1) = T$: $flit(1)$<br>$c(1) = F$: $idle$ |
| 4th : $c(1) = T$ | $flit(4)$ | $flit(2)$ | $idle$ | $idle$ |
| 4th : $c(1) = F$ | $flit(4)$ | $flit(1), time(3)$ | $\mathbf{mjv(1)}$ | $flit(1)$ |

$flit(n)$ : flit $n^{th}$ in a packet.
$time(m)$ : the flit's computation at the $m^{th}$ time.
$c(n)$ : flit $n^{th}$ comparison. $T = True; F = False$
$mjv(n)$ : flit $n^{th}$ finalization based on majority voting.

⇢ : *Input direction*
**BW** : *Buffer Writing*
**NPC:** *Next Port Computing*
**SA** : *Switch Allocation*
**CT** : *Crossbar Traversal*
**PCR:** *Pipeline Computation Redundancy*

1. **1st cycle**: the flit arrives and is handled by BW.
2. **2nd cycle**: the *first time* routing/arbitrating is started in NPC/SA.
3. **3rd cycle**: the *second time* routing/arbitrating is started in NPC/SA. PCR compares two consecutive results to find out whether a soft error occurred. If there is no soft error, CT finishes the flit's transmission. Otherwise, a recovery is need.
4. **4th cycle**: the *third time* routing/arbitrating is started in NPC/SA. A majority voting is used to find the correct result. Later, CT completes the transmission.

# Pipeline Computation Redundancy Timeline

| Cycle | BW | NPC/SA | (PCR) | CT |
|---|---|---|---|---|
| 1st | $flit(1)$ | $idle$ | $idle$ | $idle$ |
| 2nd | $flit(2)$ | $flit(1), time(1)$ | $wait$ | $idle$ |
| 3rd | $flit(3)$ | $flit(1), time(2)$ | $\mathbf{c(1)}$ | $c(1) = T$: $flit(1)$<br>$c(1) = F$: $idle$ |
| 4th : $c(1) = T$ | $flit(4)$ | $flit(2)$ | $idle$ | $idle$ |
| 4th : $c(1) = F$ | $flit(4)$ | $flit(1), time(3)$ | $\mathbf{mjv(1)}$ | $flit(1)$ |

*Summary*: PCR requires one additional clock cycle for detecting and one more clock cycle for recovery.

$fli$
$tim$
$c(n)$ : flit $n^{th}$ comparison: $T = True, F = False$
$mjv(n)$ : flit $n^{th}$ finalization based on majority voting.

**CT** : *Crossbar Traversal*
**PCR:** *Pipeline Computation Redundancy*

1. **1st cycle**: the flit arrives and is handled by BW.
2. **2nd cycle**: the *first time* routing/arbitrating is started in NPC/SA.
3. **3rd cycle**: the *second time* routing/arbitrating is started in NPC/SA. PCR compares two consecutive results to find out whether a soft error occurred. If there is no soft error, CT finishes the flit's transmission. Otherwise, a recovery is need.
4. **4th cycle**: the *third time* routing/arbitrating is started in NPC/SA. A majority voting is used to find the correct result. Later, CT completes the transmission.

# Optimizations for TSV Sharing

Although TSV sharing significantly enhances the reliability of the vertical connection, there are additional optimizations:

1. **Weight Adjustment**: after finishing the sharing process, there is a chance that a *disabled lower weight* router can borrow a *disabled higher weight router* a cluster to obtain 4 clusters. Therefore, **Weight adjustment** will reduce the weights of the disabled and higher weight routers to optimize.
2. **Virtual TSV**: when a router does not have 4 TSV-clusters, it can *temporarily* borrow one of its neighbors for communication. This only happens when the neighbor is free to be borrowed.
3. **Serialization**: when even borrowing cannot help the router to obtain 4 clusters, it can perform a serialization mode. Instead of 1 flit/clock cycle, it takes 2 or 4 cycles.
4. **Fault-tolerant routing**: at a high fault rate, a router even has no cluster for communication.

# Reliability Assessment Configuration

In order to assess the reliability, we first select a random weight where more faults are injected in the protected module. In addition, we extracted the ratio of area cost from hard ware complexity[13].

Table 7: Router's Weight and Gate Ratio.

| Module | Submodule | Weight | Gate Ratio |
|---|---|---|---|
| Network | Network | 100% | 100% |
| | Routers | 70% | 100% |
| | Channels | 30% | 0% |
| Router | Router | 100% | 100% |
| | Input Buffer | 69.72% | 7.90% |
| | Crossbar | 8.00% | 11.43% |
| | Switch-Allocator | 7.00% | 16.97% |
| | Others | 15.28% | 63.7% |

[13]The used router is 3D-FETO (without TSV fault-tolerance).

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is minimal
8          and TSV_Status_neighbor[c] == "NORMAL";
9          if (c==NULL) then
10             return RQ_link[i] = "NULL"
11             return Router_Status = "DISABLE"
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

**Algorithm 4:** TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is mini
8          and TSV_Status_neighbor[
9          if (c==NULL) then
10             return RQ_link[i] =
11             return Router_Stat
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

*Initialization*: the inputs are the weights and TSV-cluster status of current and neighboring routers.

**Algorithm 5:** TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is mini
8          and TSV_Status_neighbor[
9          if (c==NULL) then
10             return RQ_link[i] =
11             return Router_Stat
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

*Initialization*: the output are the router status and the request signals to neighboring routers.

**Algorithm 6:** TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
// Status of current and neighboring TSV-clusters
Input: TSV_Status_current[1 : N], TSV_Status_neighbor[1 : N]
// Request to link TSV-clusters to neighbors
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "NORMAL" then
           // It is a healthy TSV-cluster
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is mini
8          and TSV_Status_neighbor[
9          if (c==NULL) then
10             return RQ_link[i] =
11             return Router_Stat
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

*Checking the current TSV status*: if all TSV clusters are normal, there is nothing to do.

**Algorithm 7:** TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbo
// Status of current and neighb
Input: TSV_Status_current[1 : N], T
// Request to link TSV-clusters
Output: RQ_link[1 : N]
// Current router status
Output: Router_Status
1  foreach TSV_Status_current[i] do
2      if TSV_Status_current[i] == "N
           // It is a healthy TSV
3          RQ_link[i] = "NULL"
4      else
           // It is a faulty or borrowed TSV-cluster
5          find c in 1:N with:
6          Weight_neighbor[c] < Weight_current
7          Weight_neighbor[c] is minimal
8          and TSV_Status_neighbor[c] == "NORMAL";
9          if (c==NULL) then
10             return RQ_link[i] = "NULL"
11             return Router_Status = "DISABLE"
12         else
13             return RQ_link[i] = c
14             return Router_Status = "NORMAL"
```

*Find a replacement for the failed/borrowed clusters*: if there are failed/borrowed TSV clusters , the router finds replacement and sends request signal. If it cannot find, it turns into 'DISABLE' mode.

**Algorithm 8:** TSV Sharing Algorithm.

```
// Weight values of the current router and its N neighbors
Input: Weight_current, Weight_neighbor[1 : N]
```

Conditions:

1. Candidate $c$ should have smaller weight than the current router.
2. Candidate $c$ should have the smallest weight among the possible ones.
3. The TSV cluster from $c$ should be NORMAL.

```
1
2
3
4   else
        // It is a faulty or borrowed TSV-cluster
5       find c in 1:N with:
6       Weight_neighbor[c] < Weight_current
7       Weight_neighbor[c] is minimal
8       and TSV_Status_neighbor[c] == "NORMAL";
9       if (c==NULL) then
10          return RQ_link[i] = "NULL"
11          return Router_Status = "DISABLE"
12      else
13          return RQ_link[i] = c
14          return Router_Status = "NORMAL"
```

**Algorithm 9:** TSV Sharing Algorithm.

# Overview of PCR

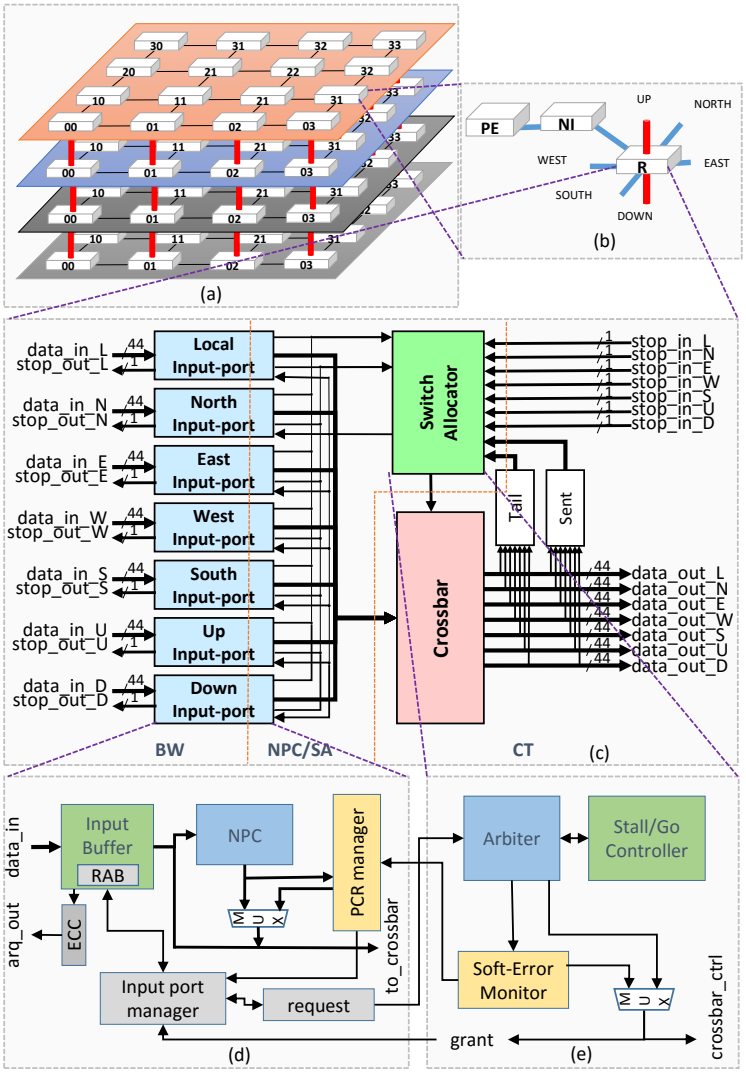



Figure 21: High-level view of the soft-hard error recovery approach: (a) 3D-Mesh based NoC configuration; (b) Tile organization; (c) SHER-3DR router organization; (d) Input-Port; (e) Switch allocation unit.

# Speedup of the assessment method

Table 8: Reliability Assessment Speedup.

| Evaluated module | A MTTF simulation | Proposed method | Speedup |
|---|---|---|---|
| A router | 11 hours | 0.090 second | 440,000 |
| A $2 \times 2 \times 2$ network | 20 hours | 0.091 second | 791,209 |
| A $3 \times 3 \times 3$ network | 2 days | 0.092 second | 1,878,261 |
| A $4 \times 4 \times 4$ network | 3.5 days | 0.109 second | 2,774,312 |

# Scaling of network reliability

The network reliability doesn't scale up with network size:

- Router vs network: Since router reliability doesn't have the flexibility as inside the network, its reliability is less than the network. E.g. router inside a network can choose an alternative routing path
- The verification of network reliability is uniform, where each node has to send to ever node.
- So, the most critical routing path is actually two neighboring router instead of long routing path.
- Therefore, the fault-tolerant algorithm benefit the reliability but not extremely enhance.

# Scaling of network reliability (cnt.)

- We use *k-failure* model to represent that with k failures, a node is disconnected. The k value is either 3 (corner routers), 4(edge routers), 5 (side routers) or 6 (middle routers). There is an improvement while increasing the network size.
- There is a domination of local (router-PE) failure that reduce the overall reliability.

$$\lambda_{RTR} = P_{corner} \times \lambda_{corner} + P_{edge} \times \lambda_{edge} + P_{side} \times \lambda_{side} + P_{middle} \times \lambda_{middle} \quad (7)$$

$\lambda_{corner}$, $\lambda_{edge}$, $\lambda_{side}$, and $\lambda_{middle}$ are calculated using model 1: m=1, n = k and r= 0. The based element is $\lambda_{conn.}$:

$$\lambda_{conn.} = \lambda_{1-input-buffer} + \lambda_{1-crossbar-link} + \lambda_{1-router-channel} \quad (8)$$

# Throughput of realistic benchmarks

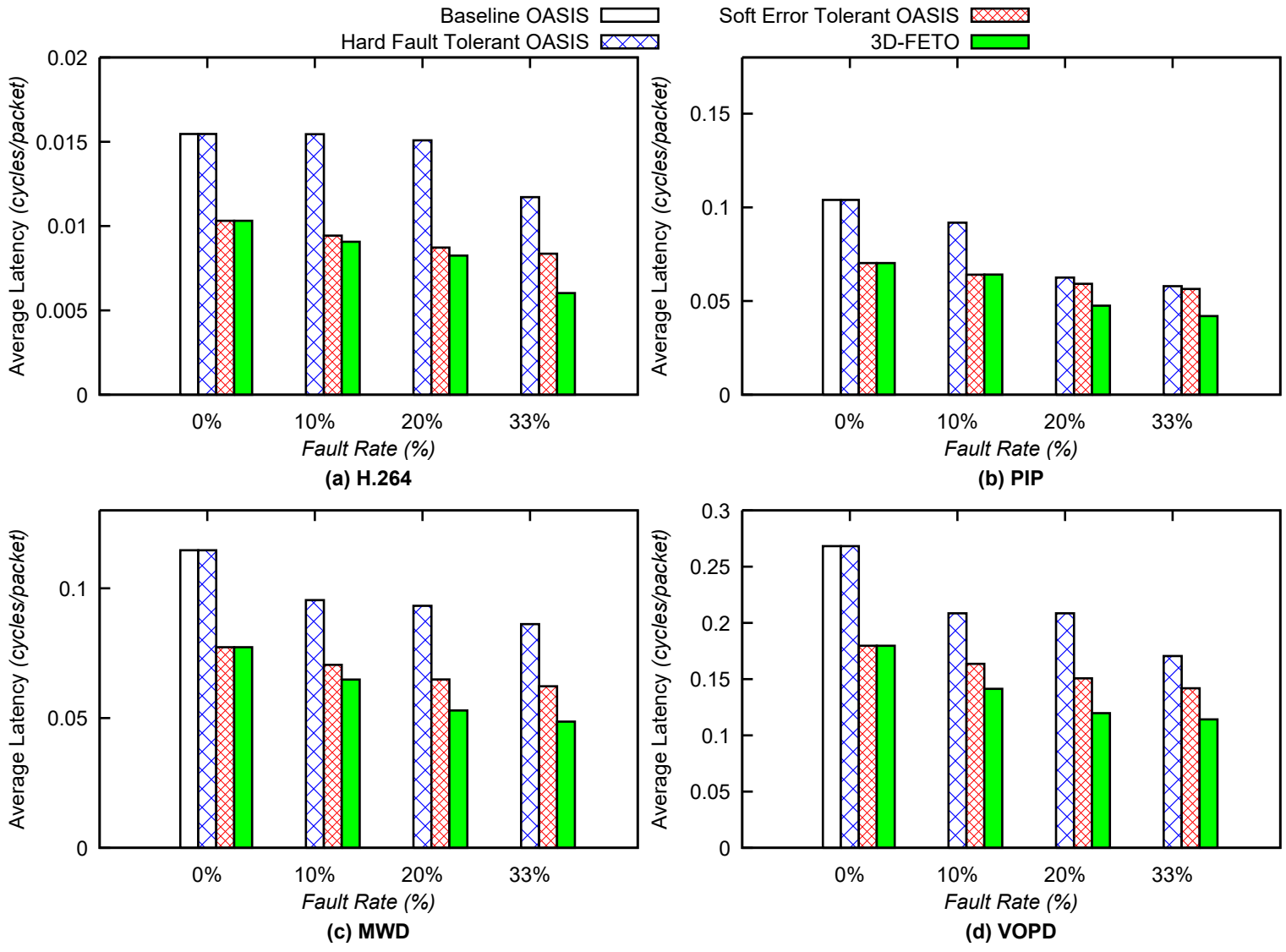



Figure 22: **Throughput** evaluation of the *synthetic* benchmarks.

# Reliability Assessment Accuracy Comparison (2)

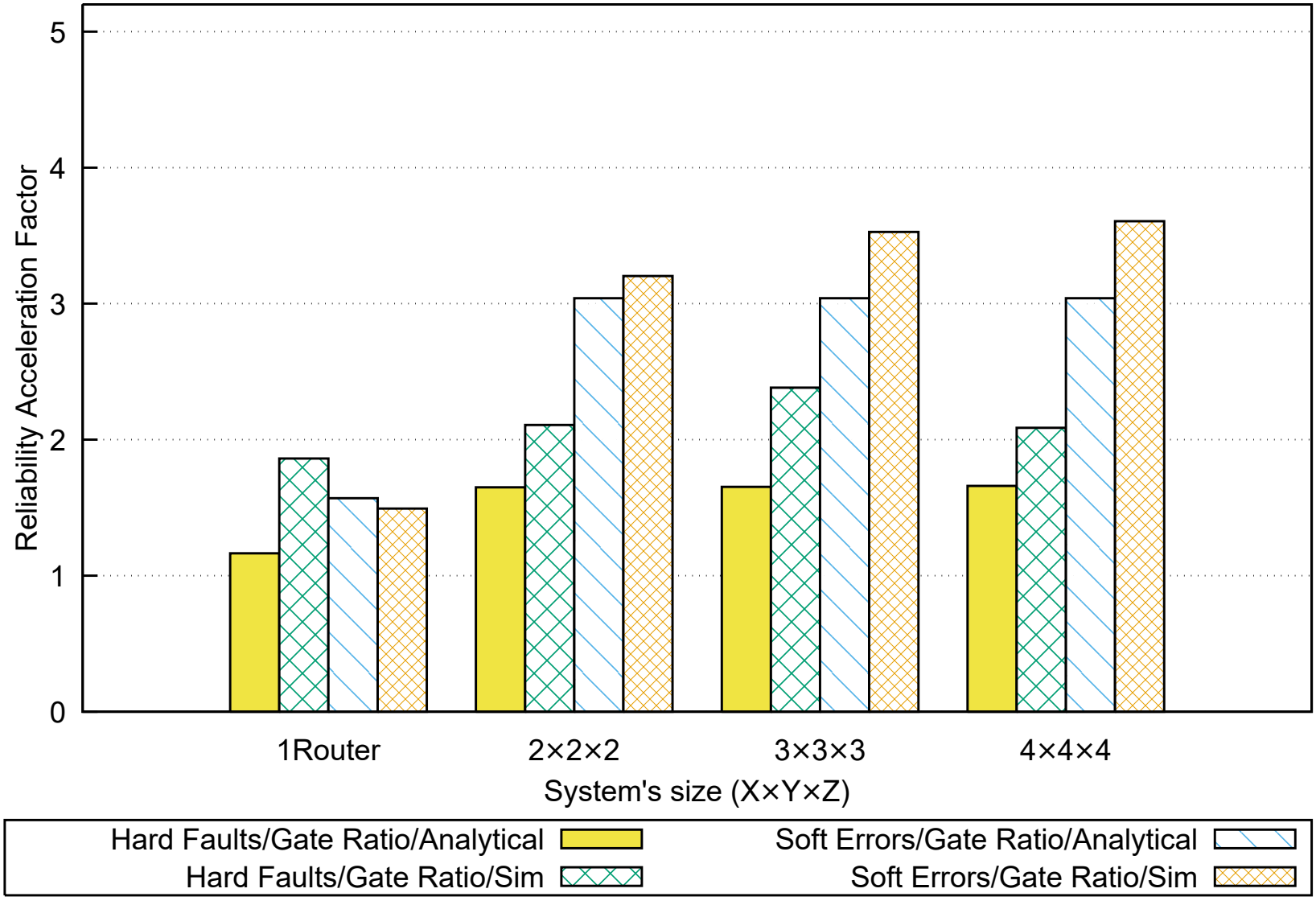



Figure 23: Comparison results between analytical assessment and Monte-Carlo MTTF simulation (cnt.).

# Average Packet Latency of synthetic benchmarks

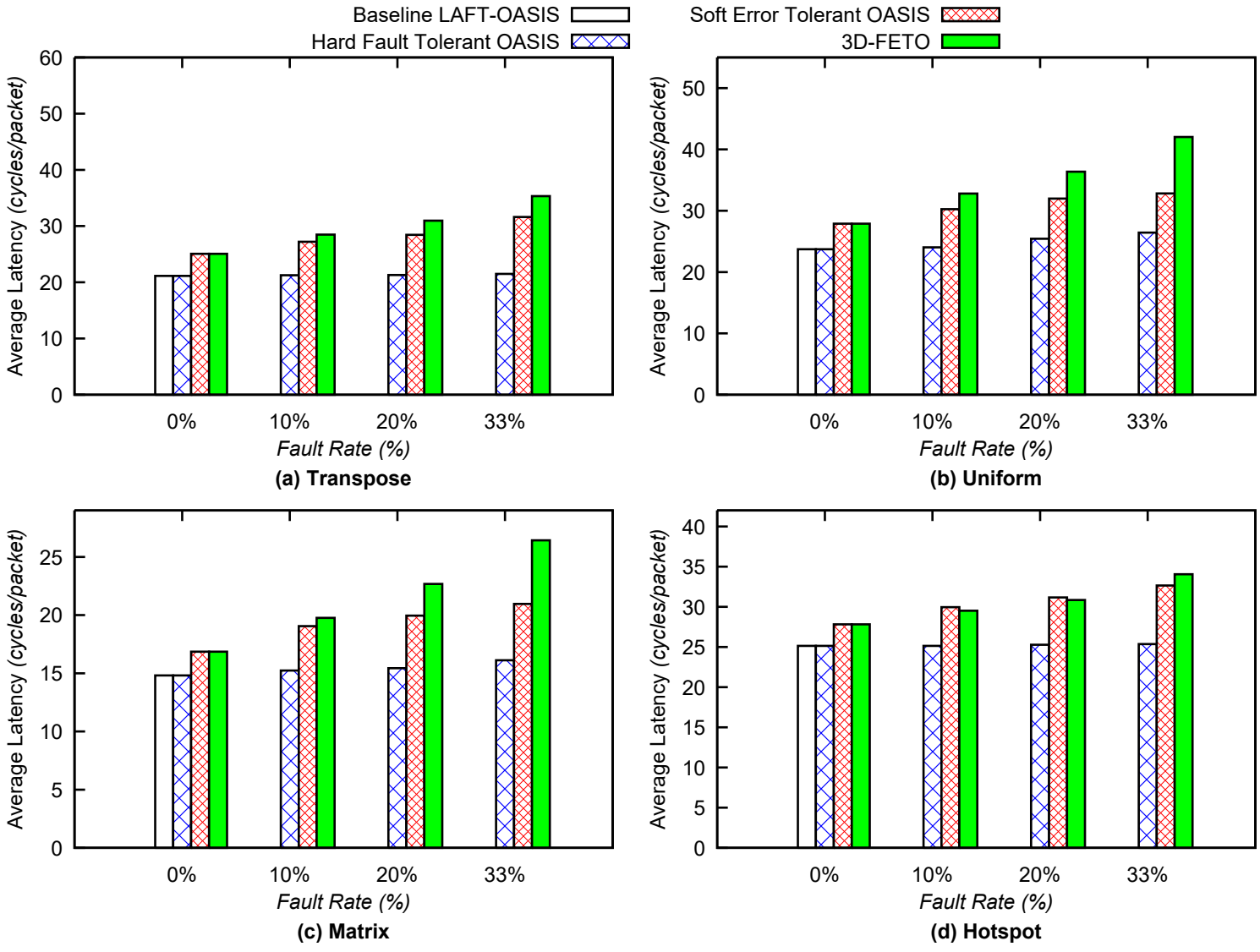



Figure 24: **Average packet latency** evaluation of the *synthetic* benchmarks.

# Average Packet Latency of synthetic benchmarks

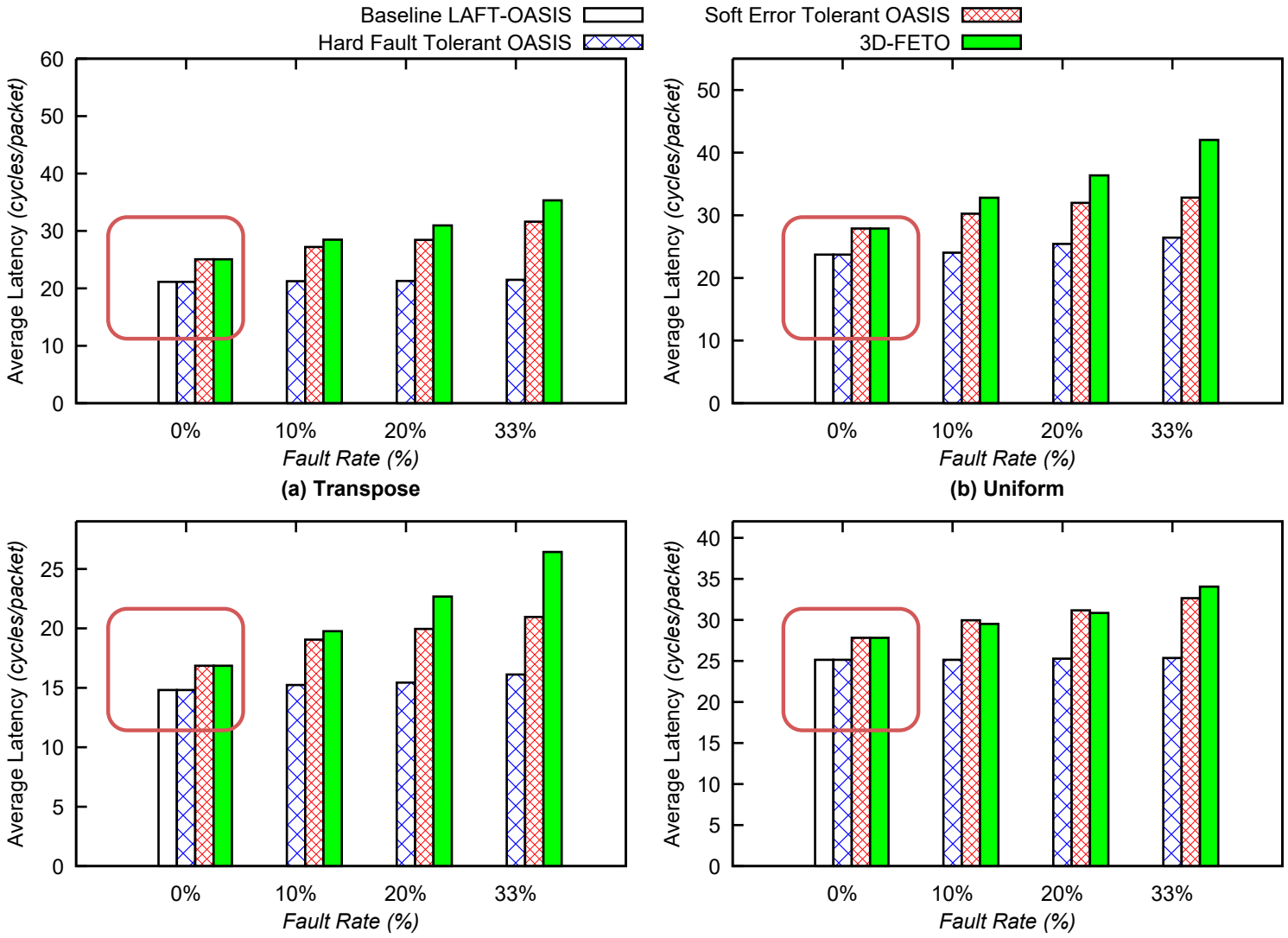



Figure 24: **Ave**

APL at **0%** of error rates, Soft Error Tolerant OASIS $\simeq$ 20% + the baseline.

# Average Packet Latency of synthetic benchmarks

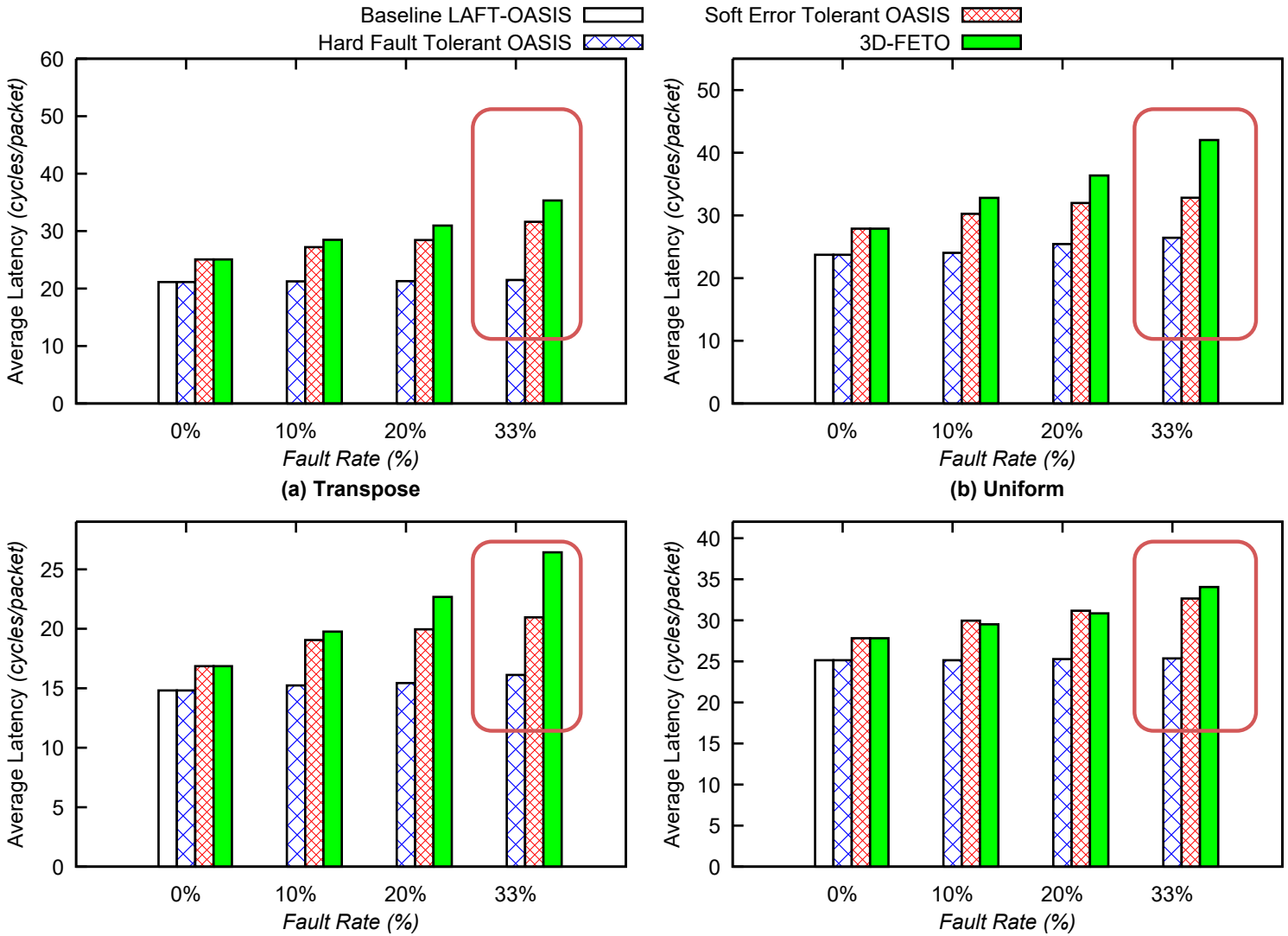



Figure 24: **Ave**

APL at **33%** of error rates, Soft Error Tolerant OASIS $\leq$ 50% + the baseline.

# Average Packet Latency of synthetic benchmarks

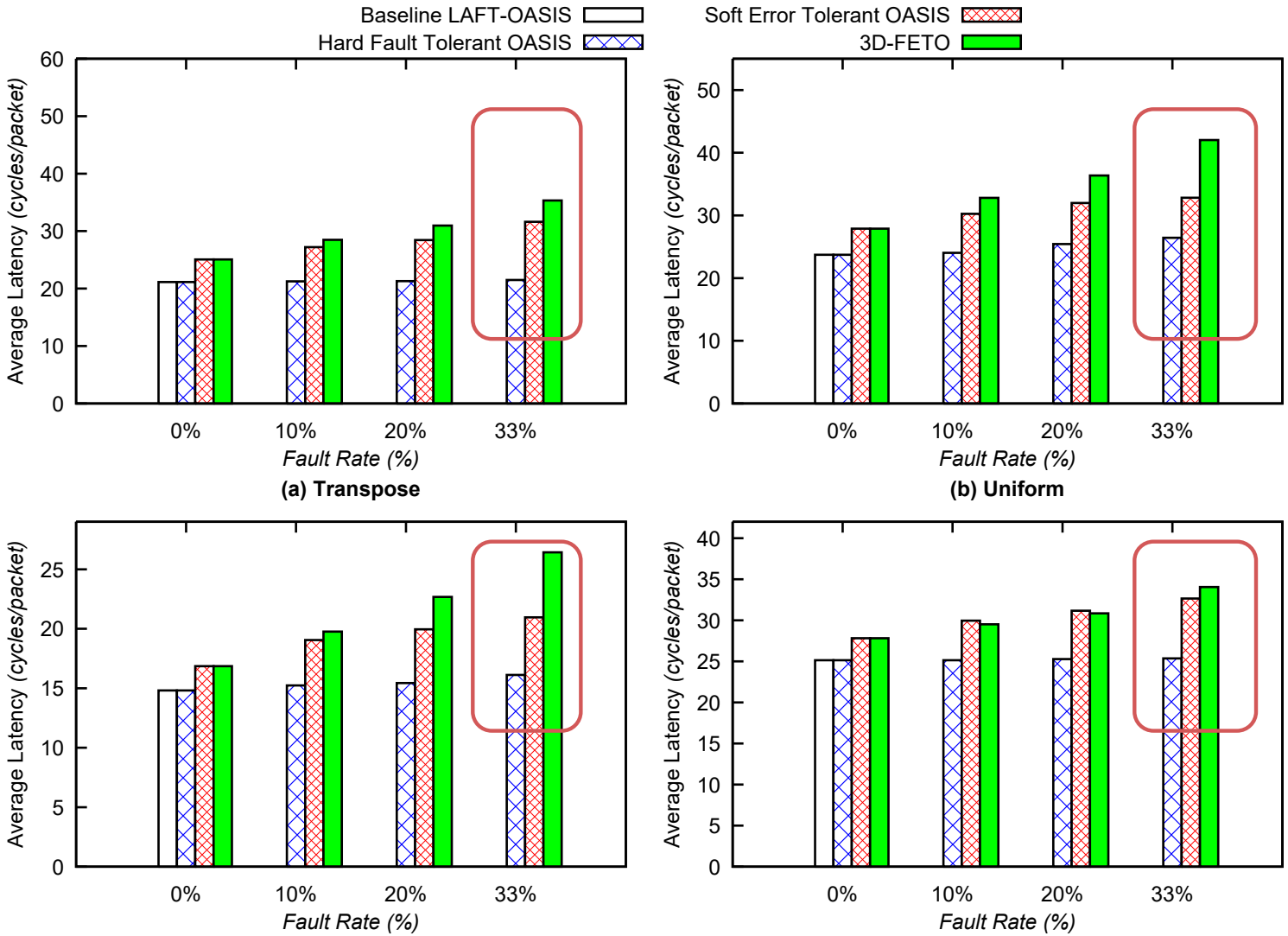



Figure 24: **Ave**

APL at **33%** of error rates, 3D-FETO $\leq$ 79% + the baseline.

# Detail of Benchmark

| Benchmark | Description | |
|---|---|---|
| Transpose | Each node (a,b,c) in a network with (X,Y,Z) sends packets to node (X-a, Y-b, Z-c) | |
| Uniform | Each node in a network sends packets to all nodes | |
| Matrix-multiplication | Performs C=A*B. Matrix A is stored in layer-1, is sent to layer-2 which has matrix B. The final values are accumulated in layer-3 as matrix C. | |
| Hotspot 10% | Each node in a network sends packets to all nodes. X (X=1 or 2 or more) nodes have additional 10% amount of traffic. | |
| Realistic Traffic Pattern | Generate from task graphs which provide the connections (e.g: node A$\rightarrow$B) and the traffic (e.g: 100 packets). | |

$\Rightarrow$ The following slides will explain these benchmarks in details.

# Transpose algorithm

```
// Network
Input: Network(X, Y, Z)
// Amount of data for each communication
Input: D
// Communication set
Output: C = {c_i : (source → destination, amount of data)}
1 foreach node (a,b,c) in Network(X, Y, Z) do
2     add ((a, b, c) → (X − a − 1, Y − b − 1, Z − c − 1), D packets) to C
3 return C
```

**Algorithm 10:** Transpose Algorithm.

# Uniform algorithm

```
// Network
Input: Network(X, Y, Z)
// Amount of data for each communication
Input: D
// Communication set
Output: C = {c_i : (source → destination, amount of data)}
1 foreach node (a,b,c) in Network(X, Y, Z) do
2     foreach node (m,n,p) in Network(X, Y, Z) do
3         add ((a, b, c) → (m, n, p), D packets) to C
4 return C
```

**Algorithm 11:** Uniform Algorithm.

# Matrix-multiplication algorithm

**Input:** $layerA(n, n)$, $layerB(n, n)$, $layerC(n, n)$,
**Input:** $A(n, n)$, $B(n, n)$
**Output:** $C(n, n)$

```
1  foreach node (i,j) in layerA(n, n) do
2      send A(i,j) → layerB(j,i)
3  foreach node (i,j) in layerB(n, n) do
4      receive A(j,i)
5      R(i, j) = A(j, i) × B(i, j)
6      foreach k in 1:n do
7          send R(i,j) → layerC(i,k)
8  foreach node (i,j) in layerC(n, n) do
9      foreach k in 1:n do
10         send C(i,j) = C(i,j) + R(k,i)
11 return C(n, n) from layerC(n, n)
```

**Algorithm 12:** Matrix-multiplication Algorithm.

# Hotspot algorithm

```
// Network
Input: Network(X, Y, Z)
// Amount of data for each communication
Input: D
// Extra percentage of hotspot node
Input: E
// Communication set
Output: C = {c_i : (source → destination, amount of data)}
1 foreach node (a,b,c) in Network(X, Y, Z) do
2     foreach node (m,n,p) in Network(X, Y, Z) do
3         if node (m,n,p) is hotspot node then
4             add ((a, b, c) → (m, n, p), (D+D*E/100) packets) to C
5         else
6             add ((a, b, c) → (m, n, p), D packets) to C
7 return C
```

**Algorithm 13:** Hotspot Algorithm.

# Algorithm of Realistic Benchmark

**Input:** $Network(X, Y, Z)$
// Communication set
**Input:** $C = \{c_i : (source \rightarrow destination, D, O)\}$
1 ProgramCounter = 0;
2 **foreach** *node (i,j,k) in* $Network(X, Y, Z)$ **do**
3  **foreach** $c_i$ *in C* **do**
4  **if** $c_i(source) == (i, j, k)$ *and ProgramCounter ==* $O$ **then**
5  **send** (i,j,k) $\rightarrow$ $c_i(destination)$ with $c_i(D)$ packets.
6  **if** $c_i(destination) == (i, j, k)$ *and ProgramCounter ==* $O$ **then**
7  **receive** $c_i(D)$ packets.
8 **if** *all destinations completedly receive their own* $c_i(D)$ *packets* **then**
9  ProgramCounter++;

**Algorithm 14:** Realistic Benchmark Algorithm.

# Task mapping (1/5)

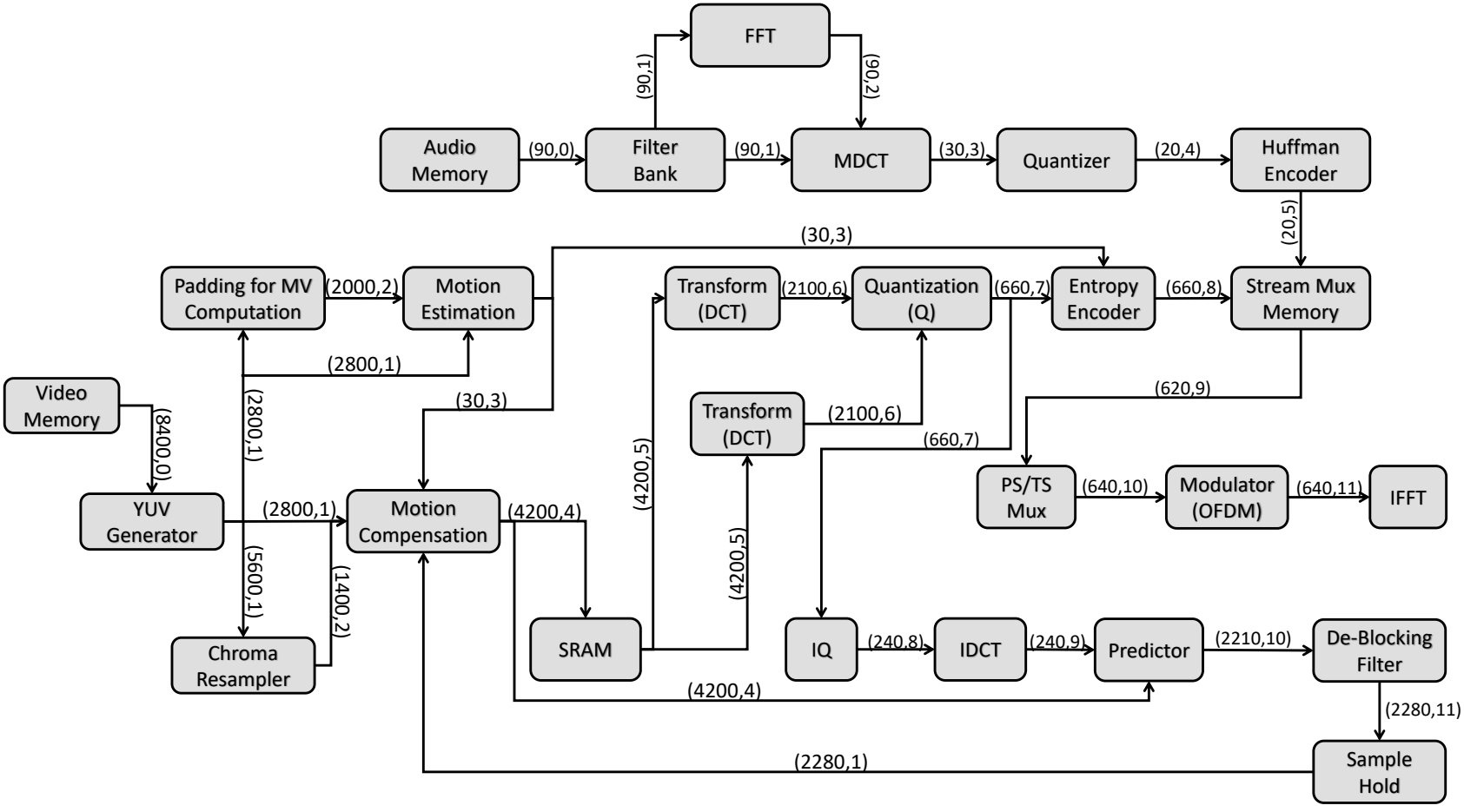



Figure 25: H.264 Task Graph.

# Task mapping (2/5)

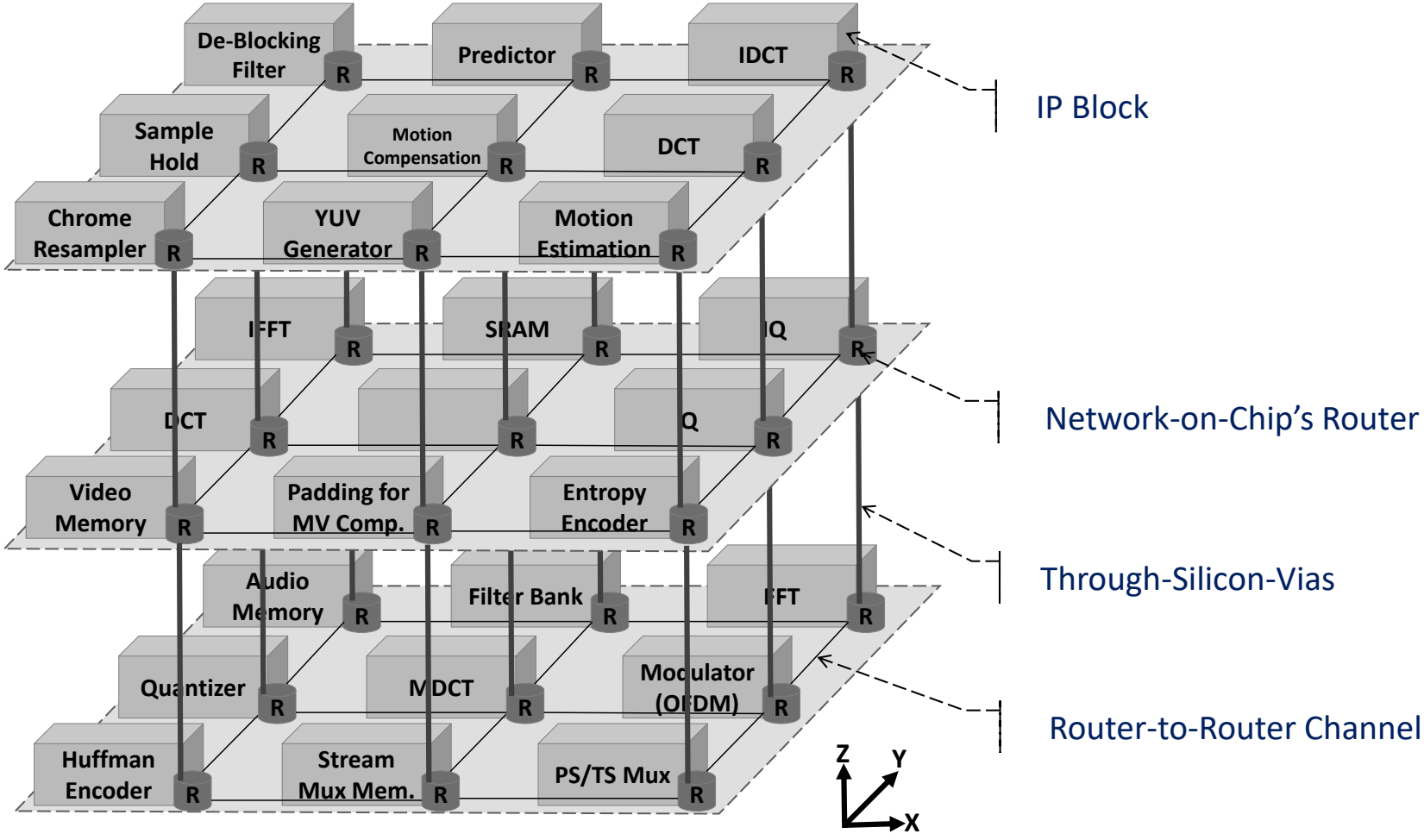
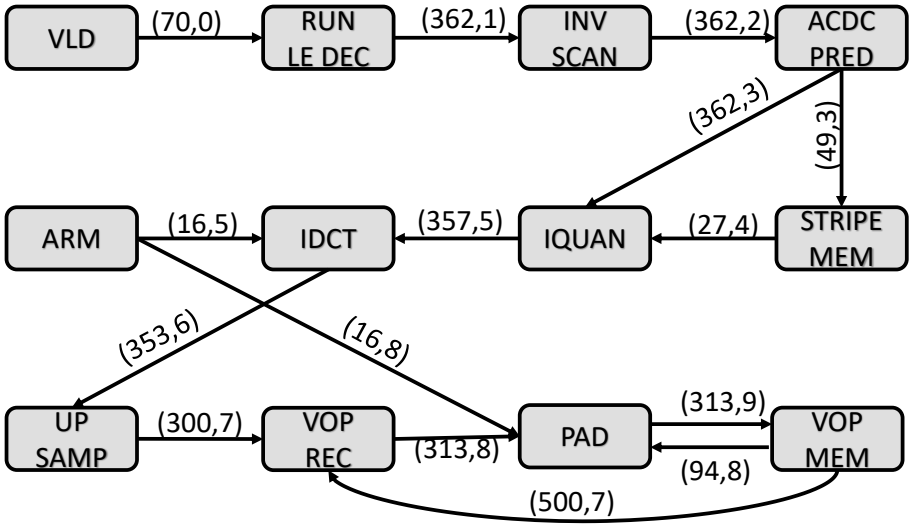



Figure 26: H.264 Task Map.

# Task mapping (3/5)



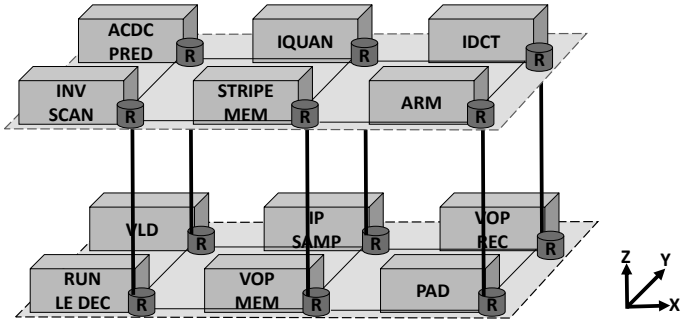
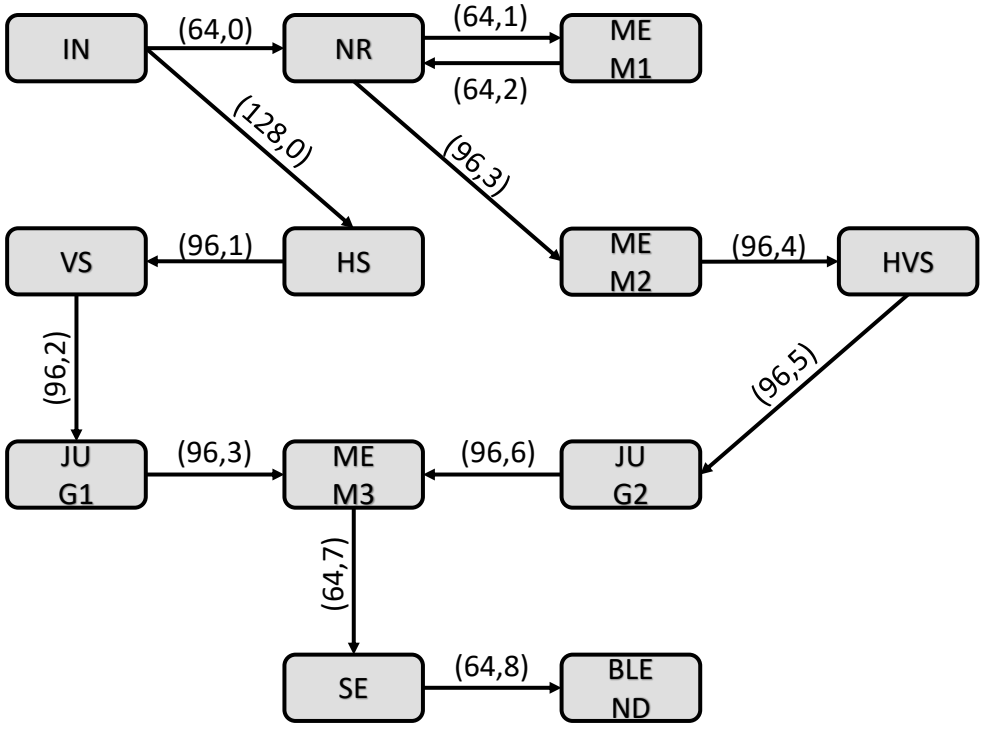




Figure 27: VOPD Task Map.

# Task mapping (4/5)



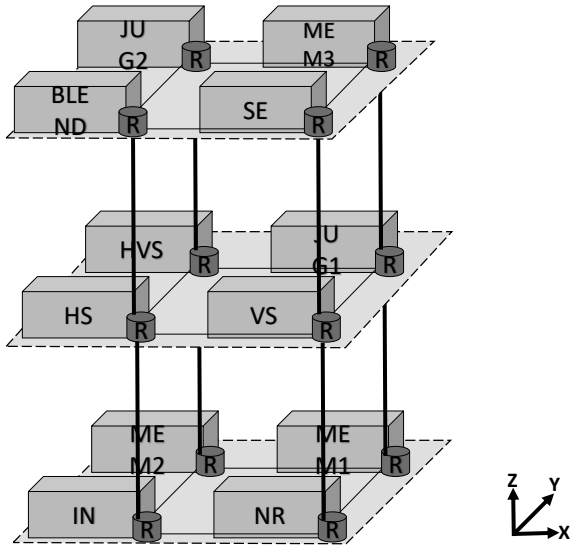
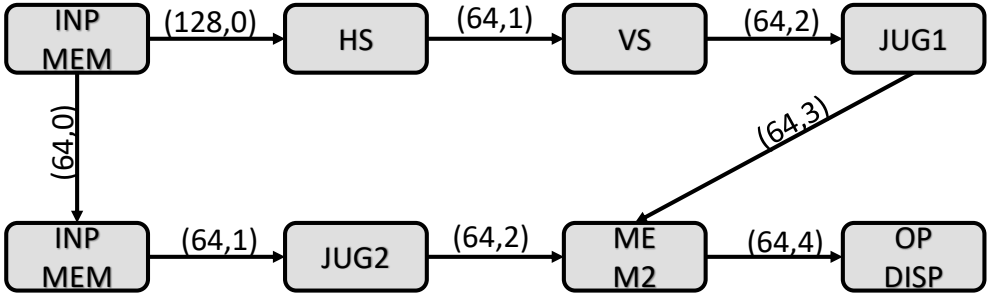




Figure 28: MWD Task Map.

# Task mapping (5/5)



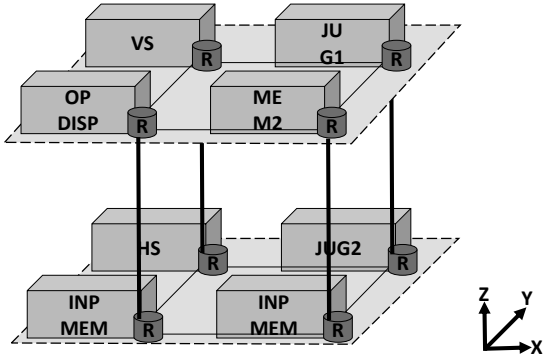




Figure 29: PIP Task Map.

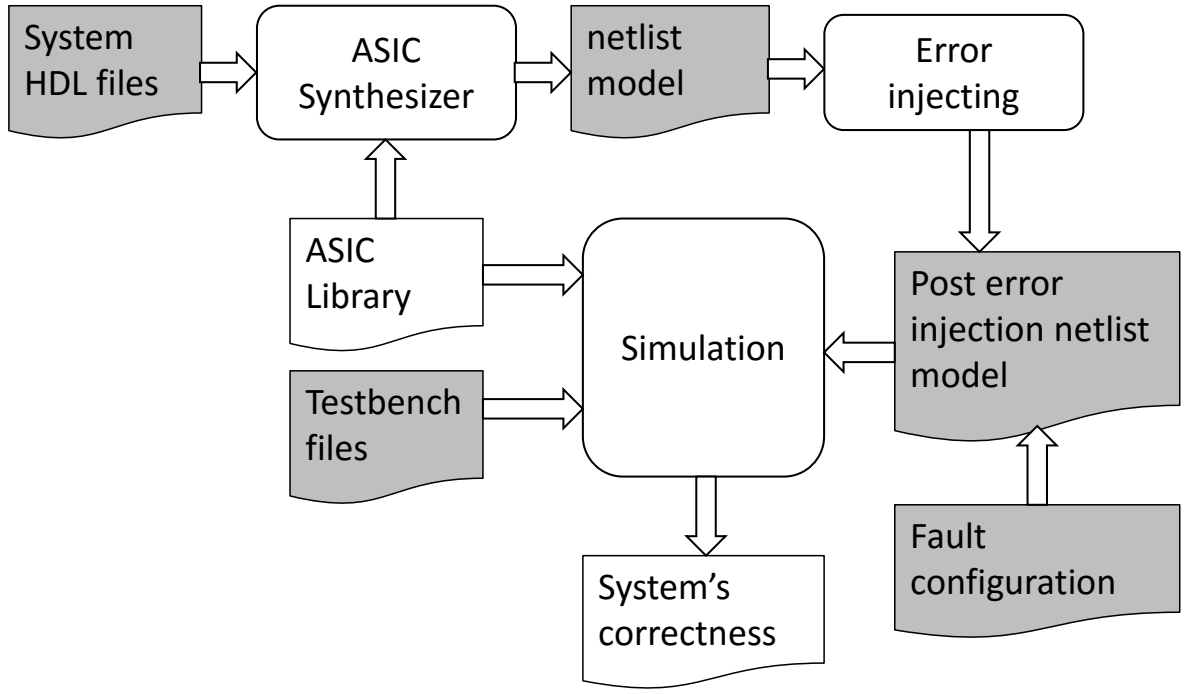



Figure 30: Monte-Carlo setting up flow.

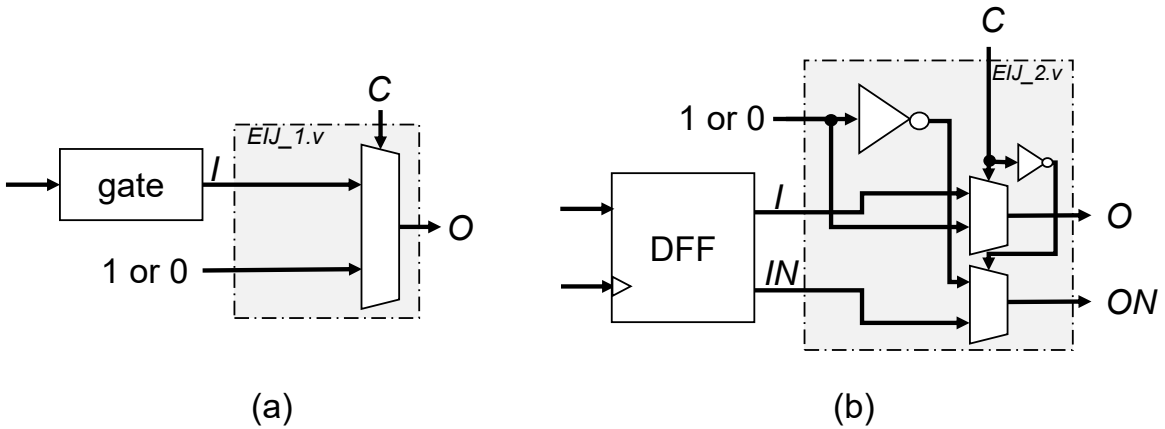



Figure 31: Error Injector architecture (a) Single output gate, (b) Flip-flop with two outputs.

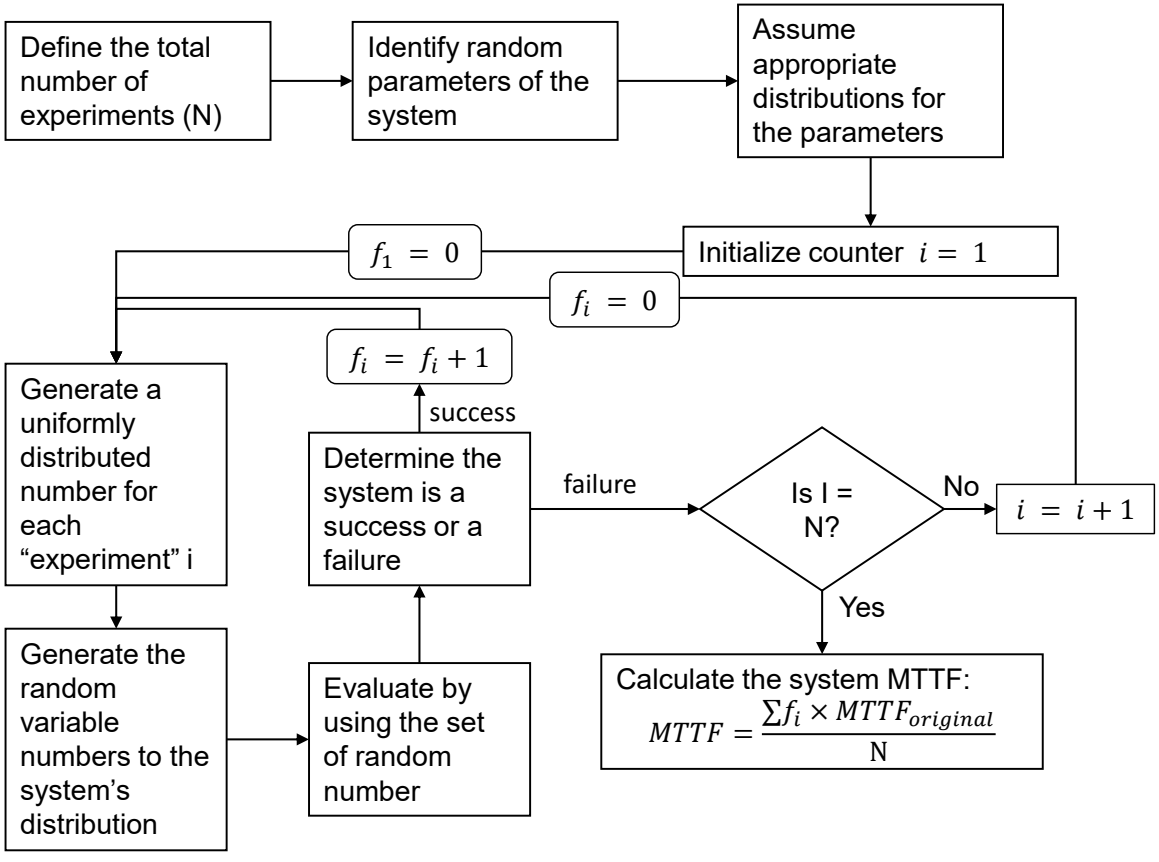



Figure 32: MTTF Monte-Carlo simulation process.

# Netlist simulation

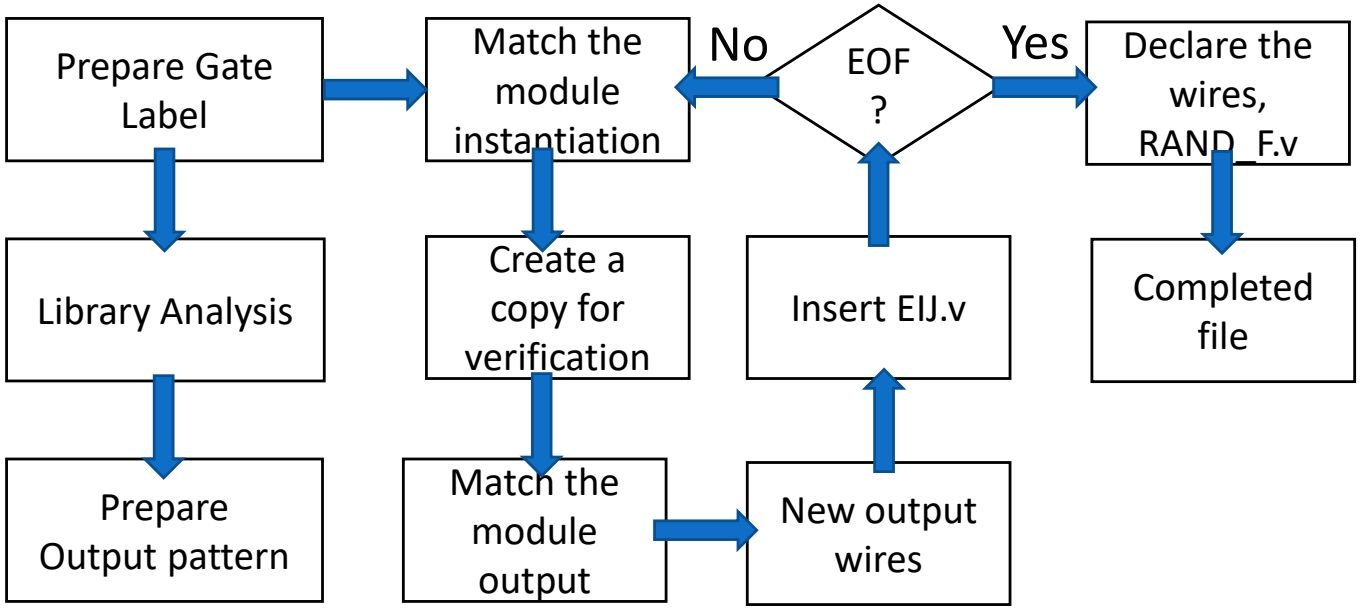



Figure 33: Flow chart of error injector inserting.

# Netlist simulation

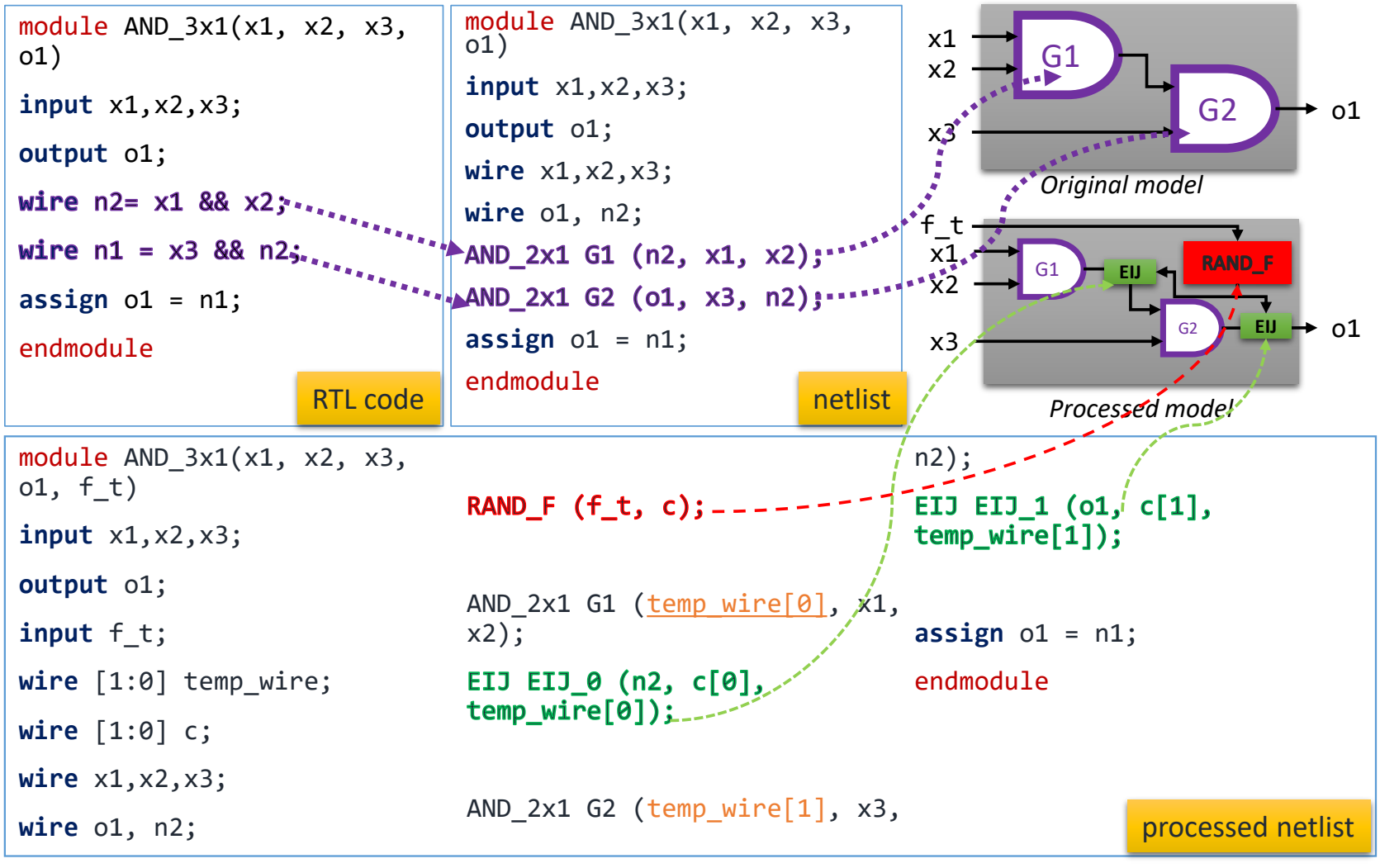
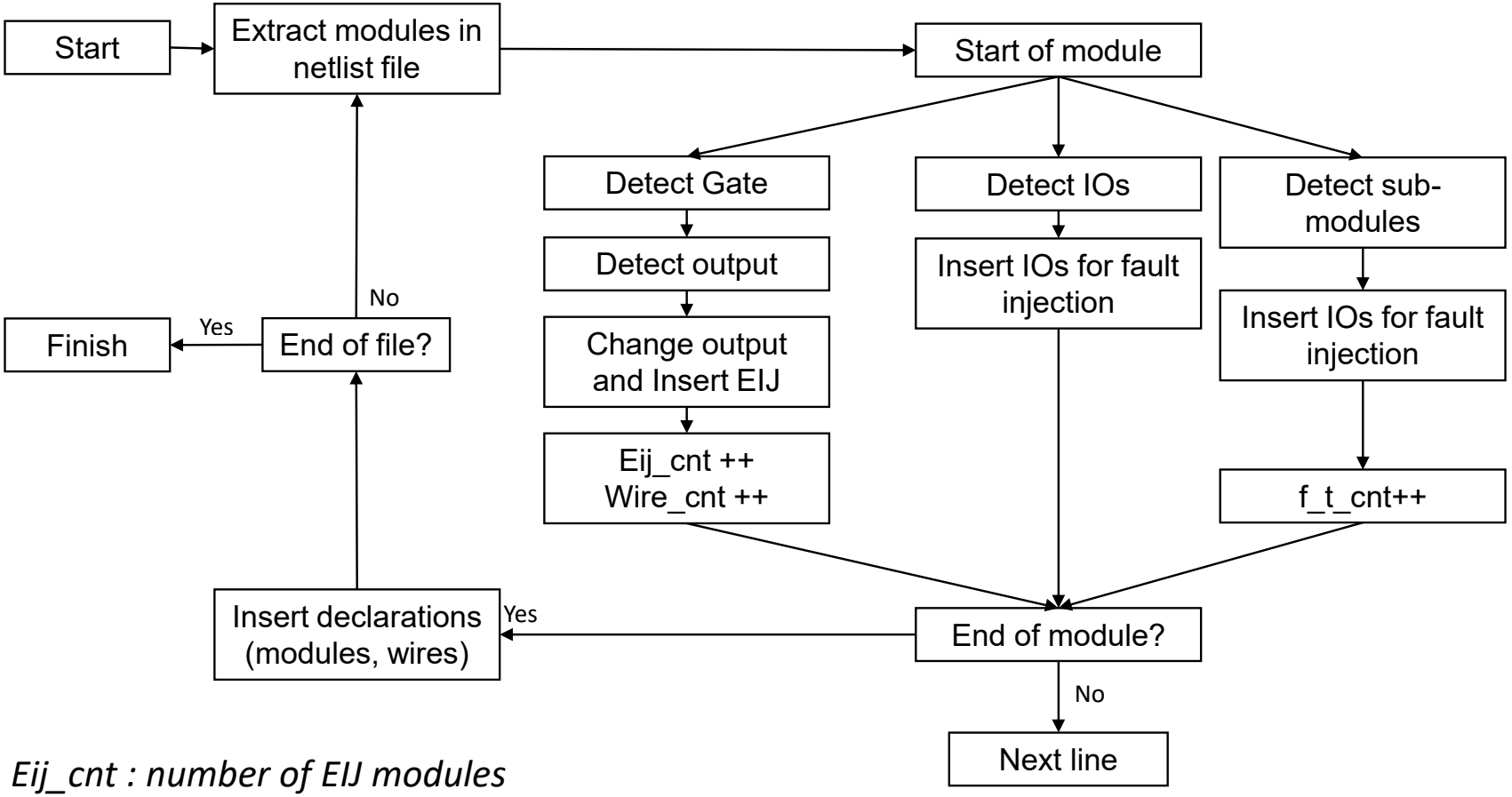



Figure 34: An example of input and output of the netlist processing.

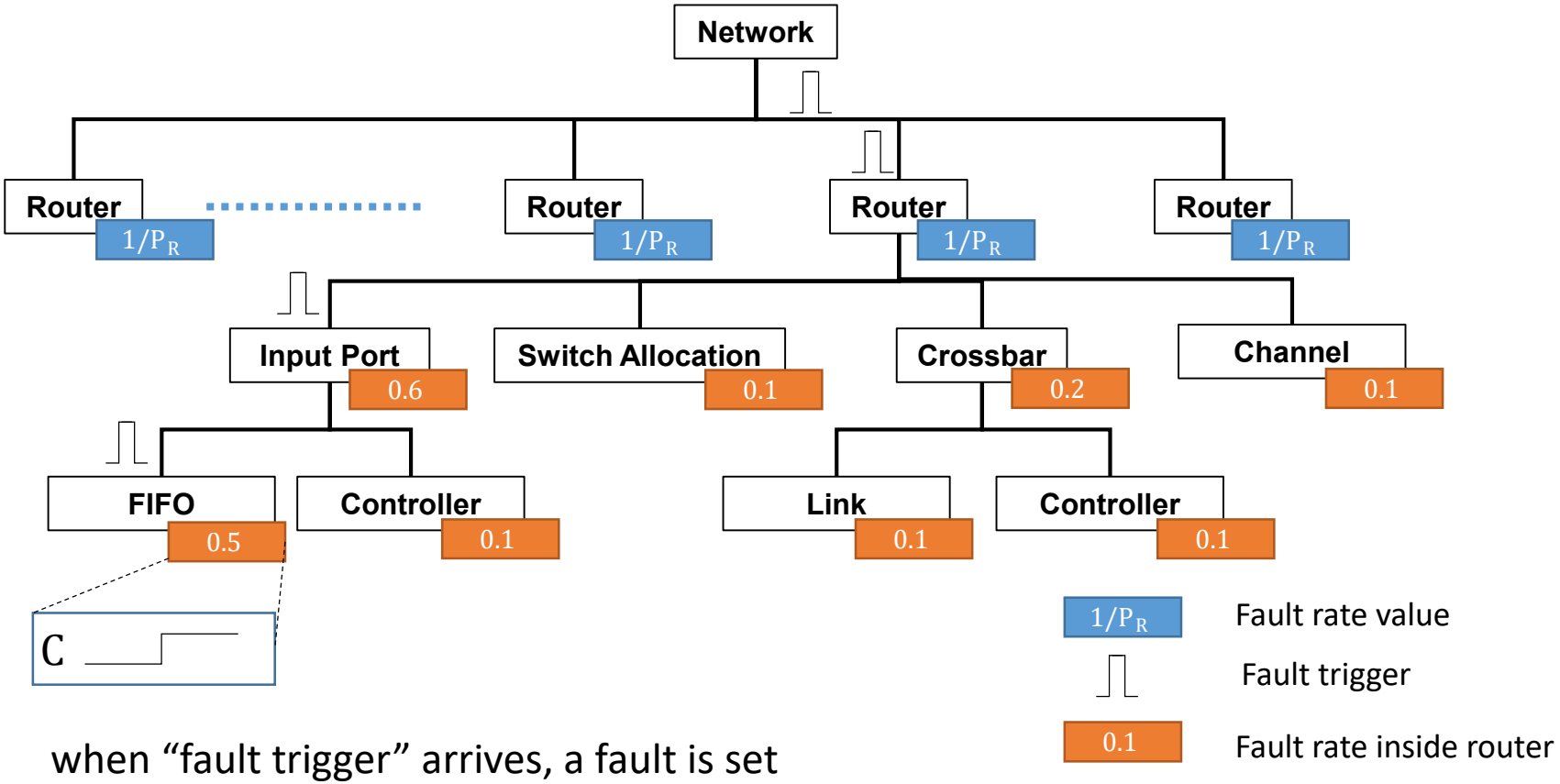
# Netlist simulation




*Eij_cnt : number of EIJ modules*
*Wire_cnt: number of temporal wires*
*f_t_cnt: number of trigger signals*

Figure 35: Netlist processing for multiple modules file.

# Netlist simulation




when “fault trigger” arrives, a fault is set in a randomized indexed gate.
For example: C =1 makes the corresponded EIJ injects a fault

The values 0.1 are loaded from configuration files.

Figure 36: Fault trigger.

# Fault-tolerance

Table 9: Taxonomy of different error recovery protocols and architectures in 3D-NoCs. Classification: A: architecture, S: software and I: integration.

| Fault Type | Position/Type | Fault Tolerant Method | Approach |
|---|---|---|---|
| Soft Errors | Data Path | Automatic Re-transmission Request | S |
| | | Error Detecting/Correcting Code | S |
| | Control Logic | Logic/Latch Hardening | A,I |
| | | Pipeline Redundancy | S |
| | | Monitoring and Correcting model | S |
| Hard Faults | Routing Technique | Spare wire | A |
| | | Split transmission | A |
| | | Fault-Tolerant routing algorithm | S |
| | Architecture-based Technique | Hardware Redundancy | A |
| | | Reconfiguration architectures | A |
| TSV Defects | Redundancy | Shifting | A,I |
| | | Crossbar | A,I |
| | | Network | A,S,I |
| | Management | Design awareness | I |
| | | Randomly distributed redundancy | A,I |

# Fault Tolerance Approaches (1/2)

1. **Architecture approach**: adding redundancies or self configuring the system to handle the task of failed module.
   - Example: failed buffer slot isolation[15], router's module triple modular redundancy [16], TSV redundancies [10, 11].
   - **Drawback**: either having high area overhead (redundancy) or degrading the performance (self-configuration).
2. **Software approach**: creating a check-point and roll-back when a fault occurs.
   - Example: pipeline stage redundancy [8].
   - **Drawback**: creating bottleneck by re-executing the failed task.

# Fault Tolerance Approaches (2/2)

3. **Integration approach**: hardening the systems by using protection or improving the reliability of backbone devices.
   - Example: TSV placement awareness [17],Logic/Latch Hardening [18].
   - **Drawback**: This type of approach leads to a highly complex design process.
4. **Hybrid approach**: combining multiple approaches to handle the fault.

# Reliability Assessment Methodology (1/2)

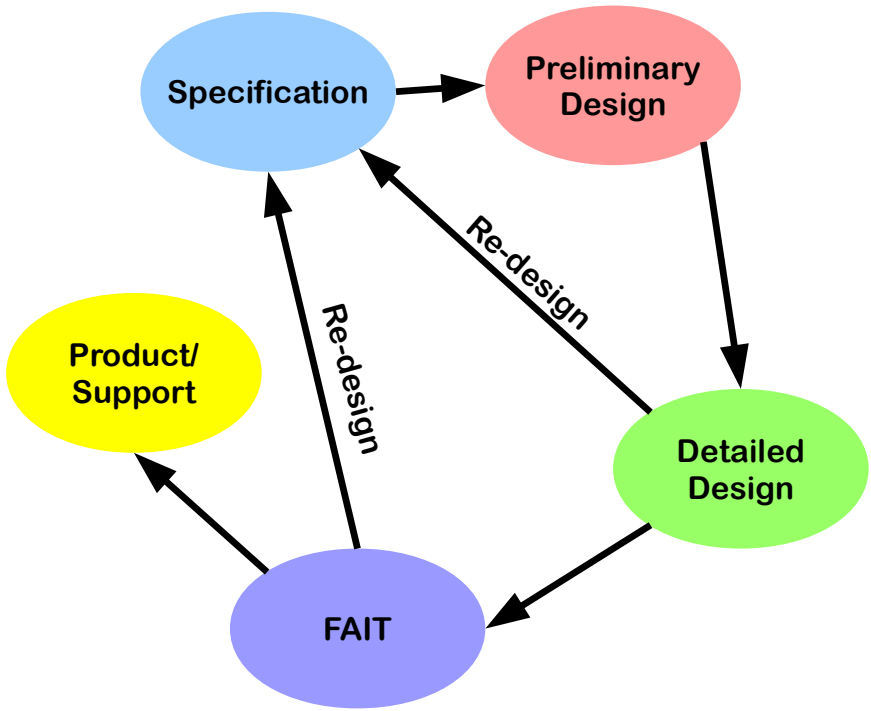



Figure 37: Stages of reliability assessment[14].

- To alleviate the risk of redesign, early assessment is essential, especially the **first three stages**.
- Reliability prediction for NoC systems is still immature.

[14] FAIT:Fabrication, Assembly, Integration and Test

# Reliability Assessment Methodology (2/2)

**Approach:**

- A quantitative factor, which is called as **Reliability Acceleration Factor**, to represent the efficiency of the fault tolerant mechanism.
- A fault assumption to help calculate the failure rate of a system.
- An analytical model to assess the reliability of NoC systems using Markov-state model.

# RAF: Reliability Acceleration Factor

RAF (Reliability Acceleration Factor), which represent the efficiency of the applied fault-tolerances, is given by the following equation:

$$\text{RAF} = \frac{\lambda_{original}}{\lambda_{FT}} = \frac{\mathbf{MTTF}_{FT}}{\mathbf{MTTF}_{original}} \geq 1 \tag{9}$$

Where:

- $\lambda$ is the fault rate and it is the inverse value of Mean Time to Failure (*MTTF*).
- $\mathbf{MTTF}_{original}$ is the MTTF of the original system.
- $\mathbf{MTTF}_{FT}$ is the MTTF of the fault-tolerant system.

# Fault Rate Assumption

For a system with $k$ components, its fault rate is given by:

$$\lambda_{system} = \frac{1}{\mathbf{MTTF}_{system}} = \sum_{i=1}^{k} f_i \pi_i \lambda_{unit} \tag{10}$$

Where:

- $unit$ is a selected module as a reference for calculation.
- $\pi_i$ is the fault-rate ratio between the component and the *unit*.
- $f_i$ is the fault-rate ratio after attaching the component to the system.

# Markov-state Model

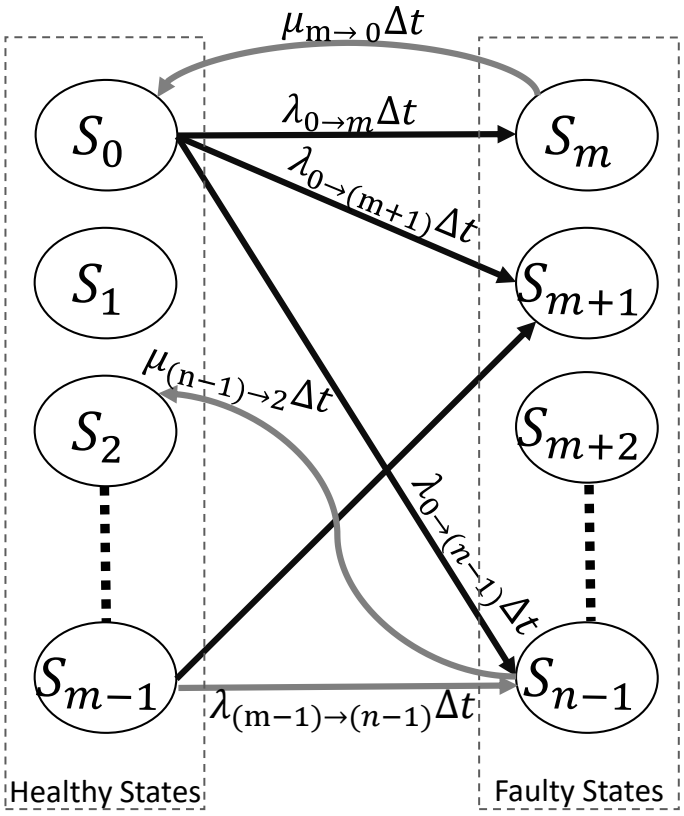



Figure 38: A Markov-state reliability model for an $n$ states system with $m$ non-faulty states.

Mean Time To Failure (MTTF) calculation as follows:

$$\mathbf{MTTF} = \int_{t=0}^{\infty} R(t) = \lim_{s\to 0}(R^*(s)) \quad (11)$$

where $R(t)$ is the reliability function and $R^*(s)$ is its Laplace form.
Assume a system has $n$ states of failure/healthy. $\mathbb{H}$ is the set of healthy states. $\mathbb{F}$ is the set of failed states.

$$R^*(s) = P(\mathbb{H}) = \sum_{S_i\in\mathbb{H}} P(S_i) \quad (12)$$

By calculating the probability of each state (in Laplace domain), we can obtain the MTTF value.

# Markov-state Model (cnt.)

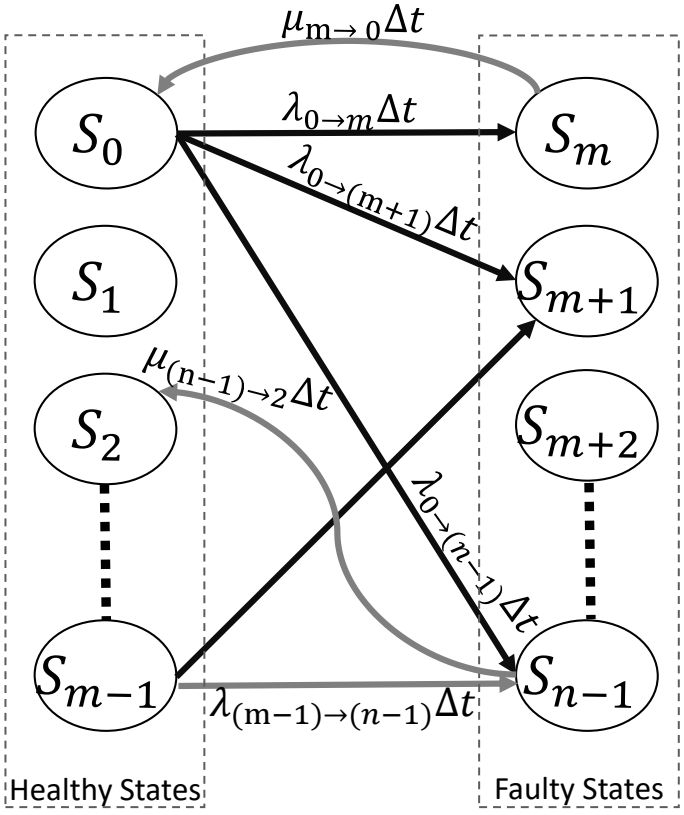



Figure 39: A Markov-state reliability model for an *m* states system with *n* non-faulty states.

Inside a Markov-state model, the transitions between states are indicated with:

- **Fault-rate of a sub-module** ($\lambda$): when a sub-module is failed, the state of the system may change to another state.
- **Repair-rate of a sub-module** ($\mu$): when a sub-module is repaired, the state of the system may change to another state.

# Reliability Assessment Methodology

**Dividing:**

1. A Network-on-Chip consists of $N_R$ routers.
2. A router is divided into several modules.

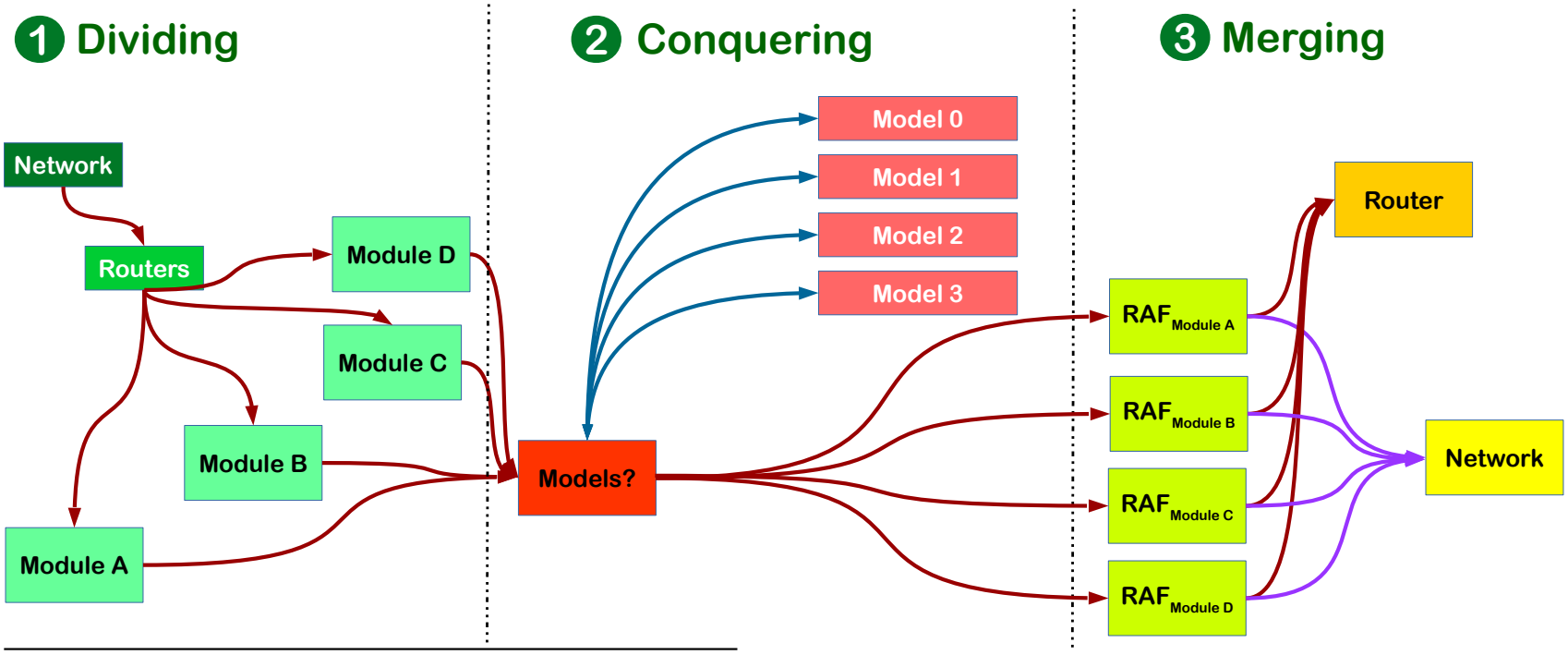
**Conquering:**

1. For each module of a router, analyze it using one of the following model:
   - *Model 0*: non fault-tolerant module - use the fault assumption (Eq. 10).
   - *Model 1*: spare or reconfiguration module.
   - *Model 2*: fault reduced module.
   - *Model 3*: module with fault-tolerance support.

**Merging:**

1. A router reliability is obtained by *Router Merging*.
2. A network reliability is obtained by *Network Merging*.

# Reliability Assessment Methodology




**Related journal paper**

1. **Khanh N. Dang**, Akram Ben Ahmed, Xuan-Tu Tran, Yuichi Okuyama and Abderazek Ben Abdallah, "A Comprehensive Reliability Assessment of Fault-Resilient Network-on-Chip Using Analytical Model", *IEEE Transactions on Very Large Scale Integration Systems*, 2017. DOI: 10.1109/TVLSI.2017.2736004.

**Related conference paper**

1. **Khanh N. Dang**, Michael Meyer, Yuichi Okuyama and Abderazek Ben Abdallah, "Reliability Assessment and Quantitative Evaluation of Soft-Error Resilient 3D Network-on-Chip Systems", *The IEEE 25th Asian Test Symposium (ATS)*, pp. 161-166, Hiroshima, Japan, November 21-24, 2016.