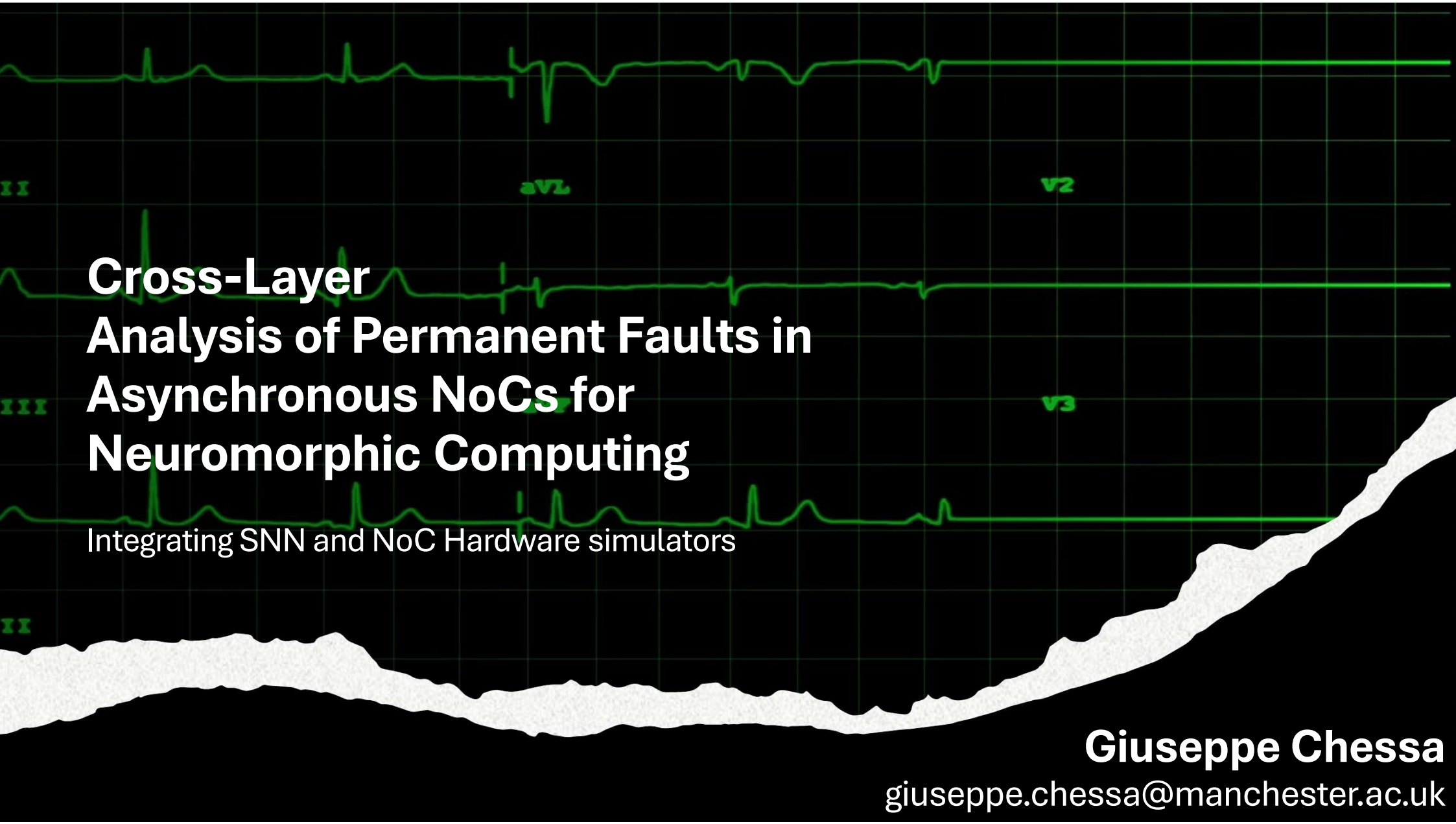
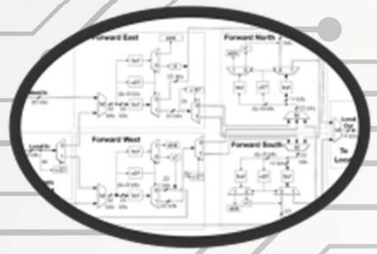


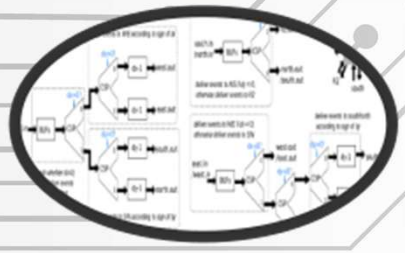

# Cross-Layer Analysis of Permanent Faults in Asynchronous NoCs for Neuromorphic Computing

Integrating SNN and NoC Hardware simulators

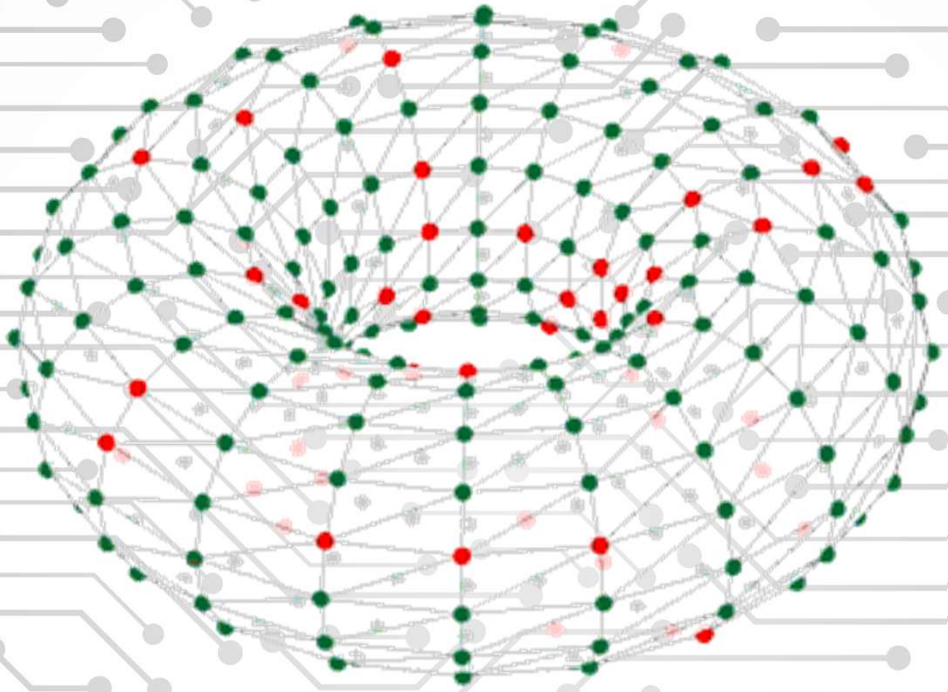
**Giuseppe Chessa**
giuseppe.chessa@manchester.ac.uk

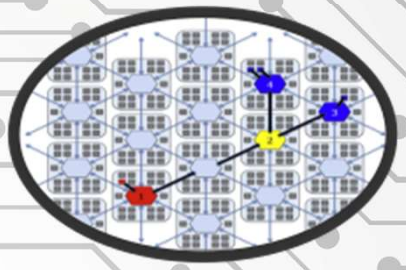


TrueNorth
- Asynchronous NoC -



Dynap
- Asynchronous NoC -



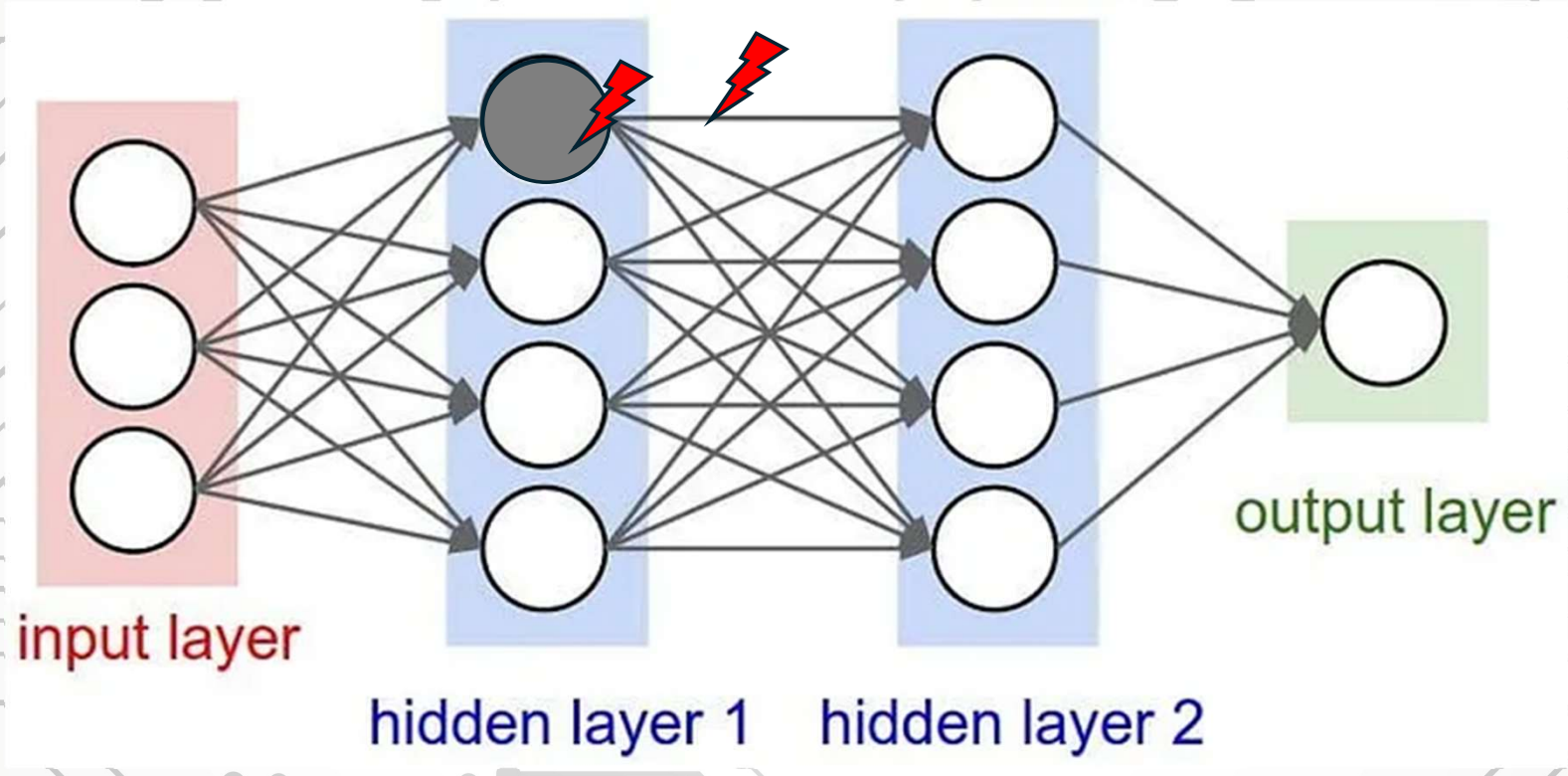


SpiNNaker
- Asynchronous NoC -

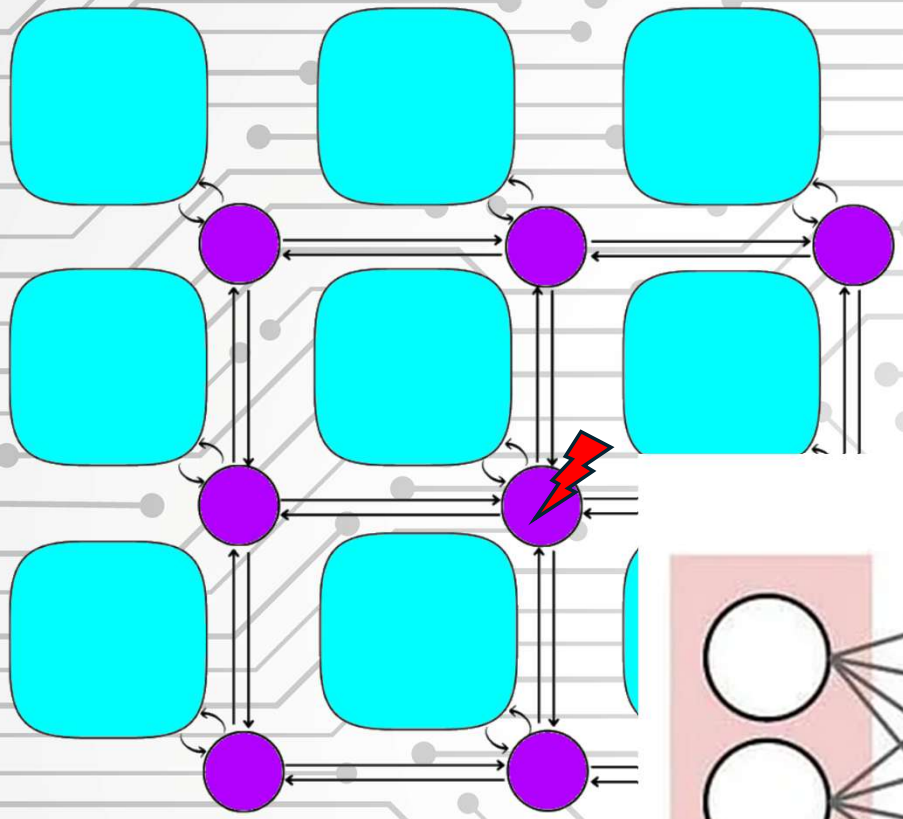

Loihi
- Fully asynchronous -

**At the heart of neuromorphic hardware lies the interconnect, enabling brain-like parallelism and ultra-efficient communication**

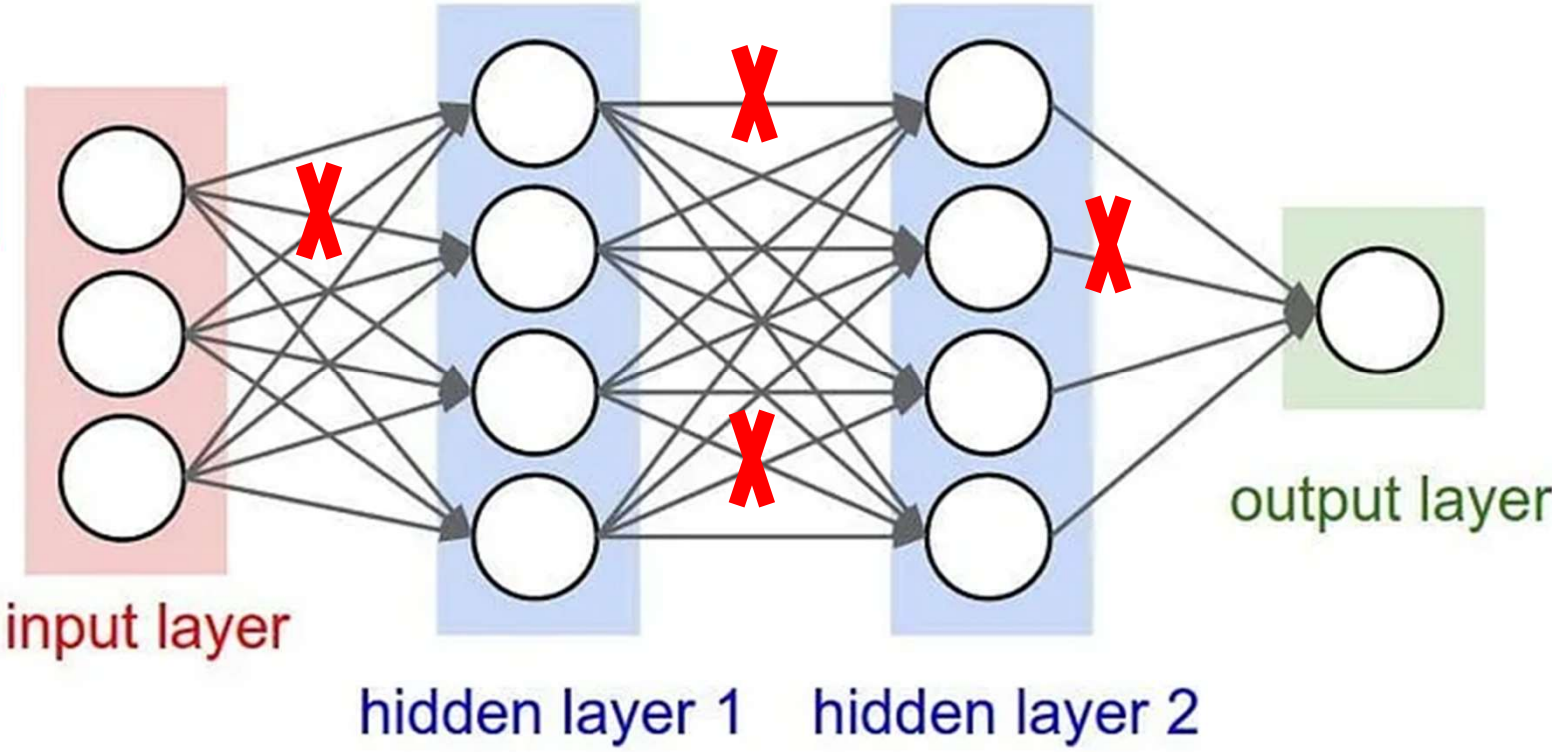
# SNN fault models




**Saturated neurons, dead neurons, spike timing variation, weight variation, zeroing**



**Faults in the interconnect can not be simplified as a fault in the single neuron/synapses**



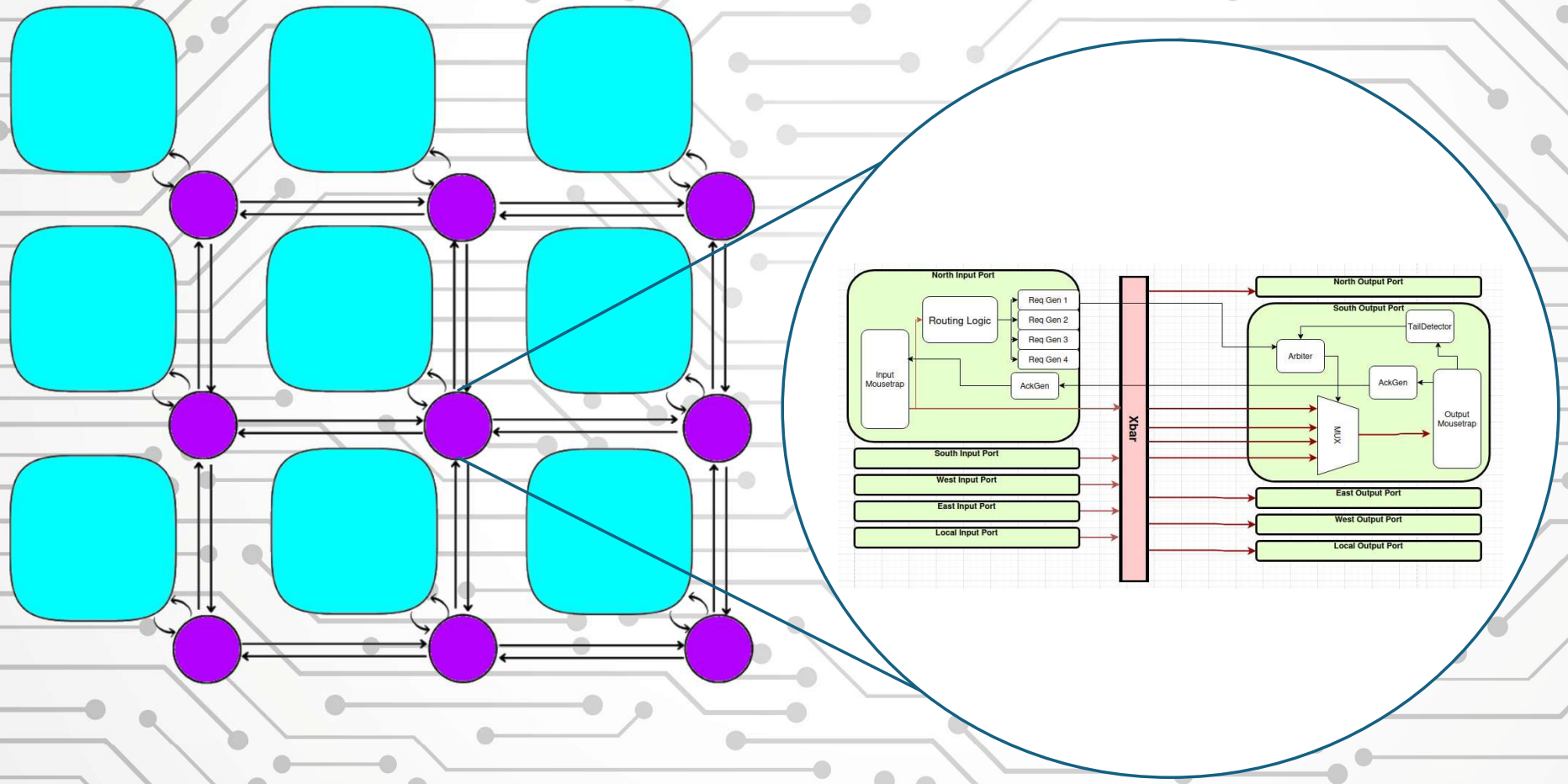
# Contributions

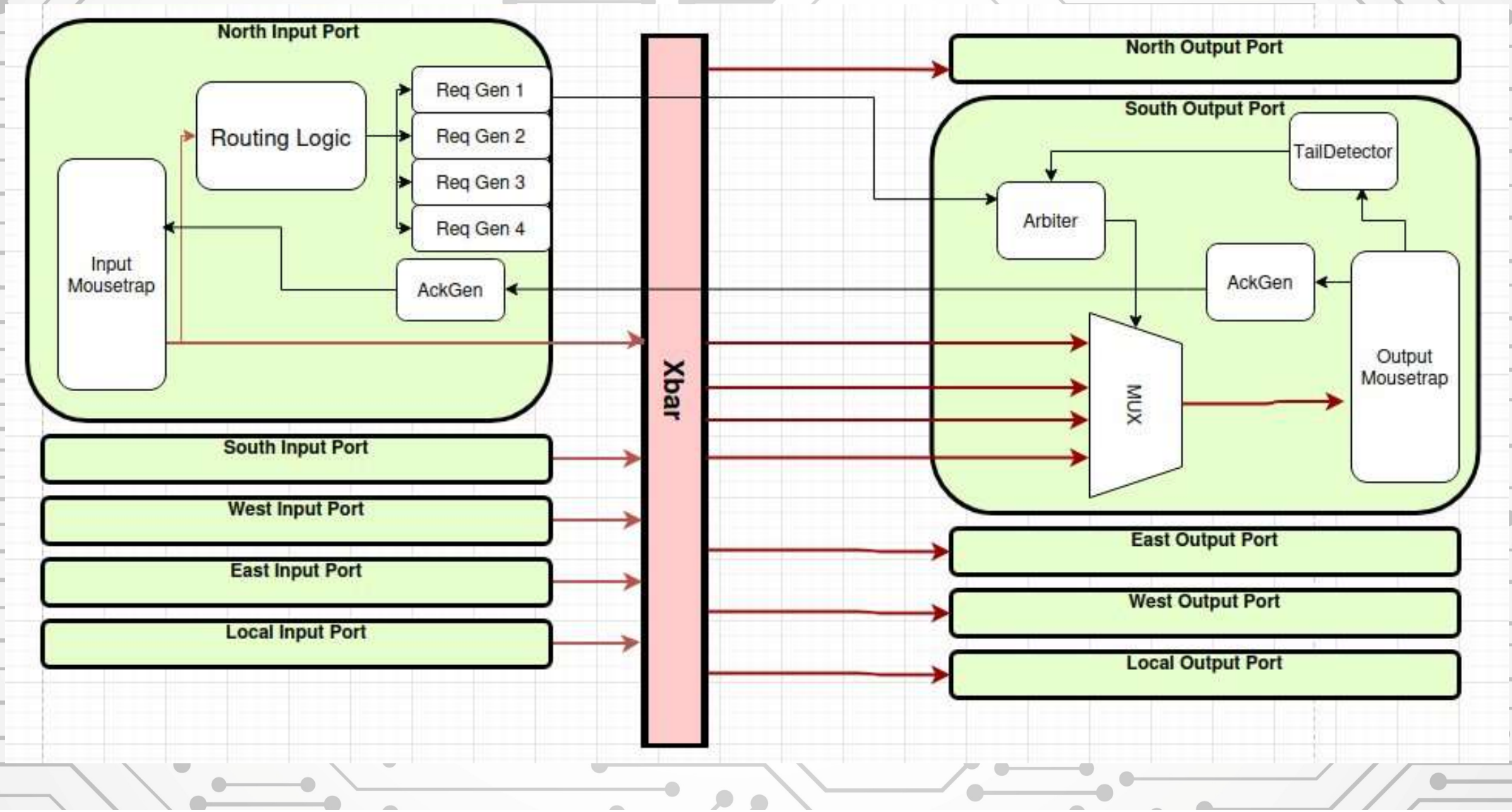
**We study how permanent faults in the AER routing fabric and SNN performance are linked**

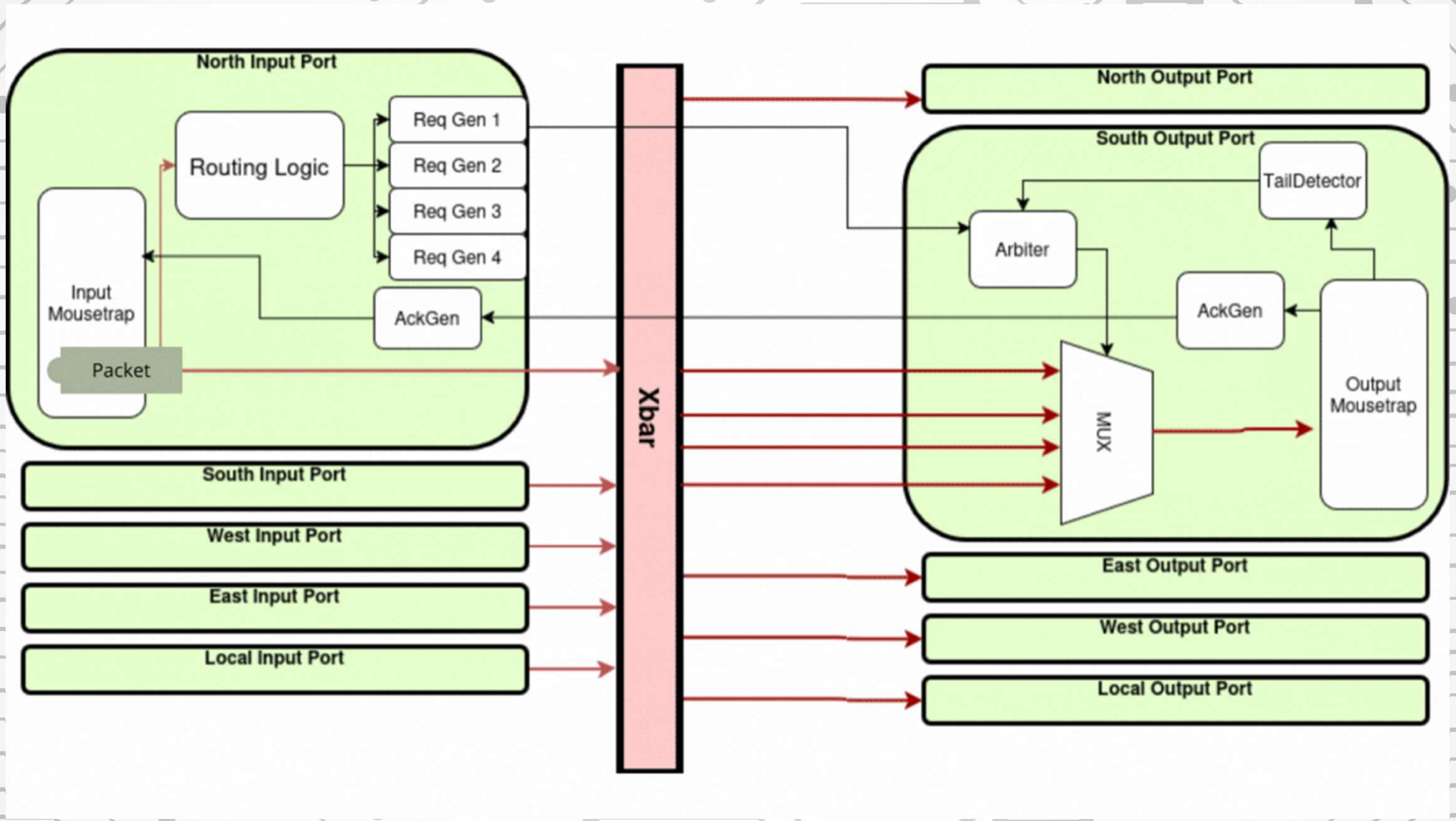
**We evaluate SNN robustness based on the integrity of the underlying asynchronous NoC**

**We study how permanent fault afftect bundled-data NoCs, an underexplored area compared to testability and soft-errors**

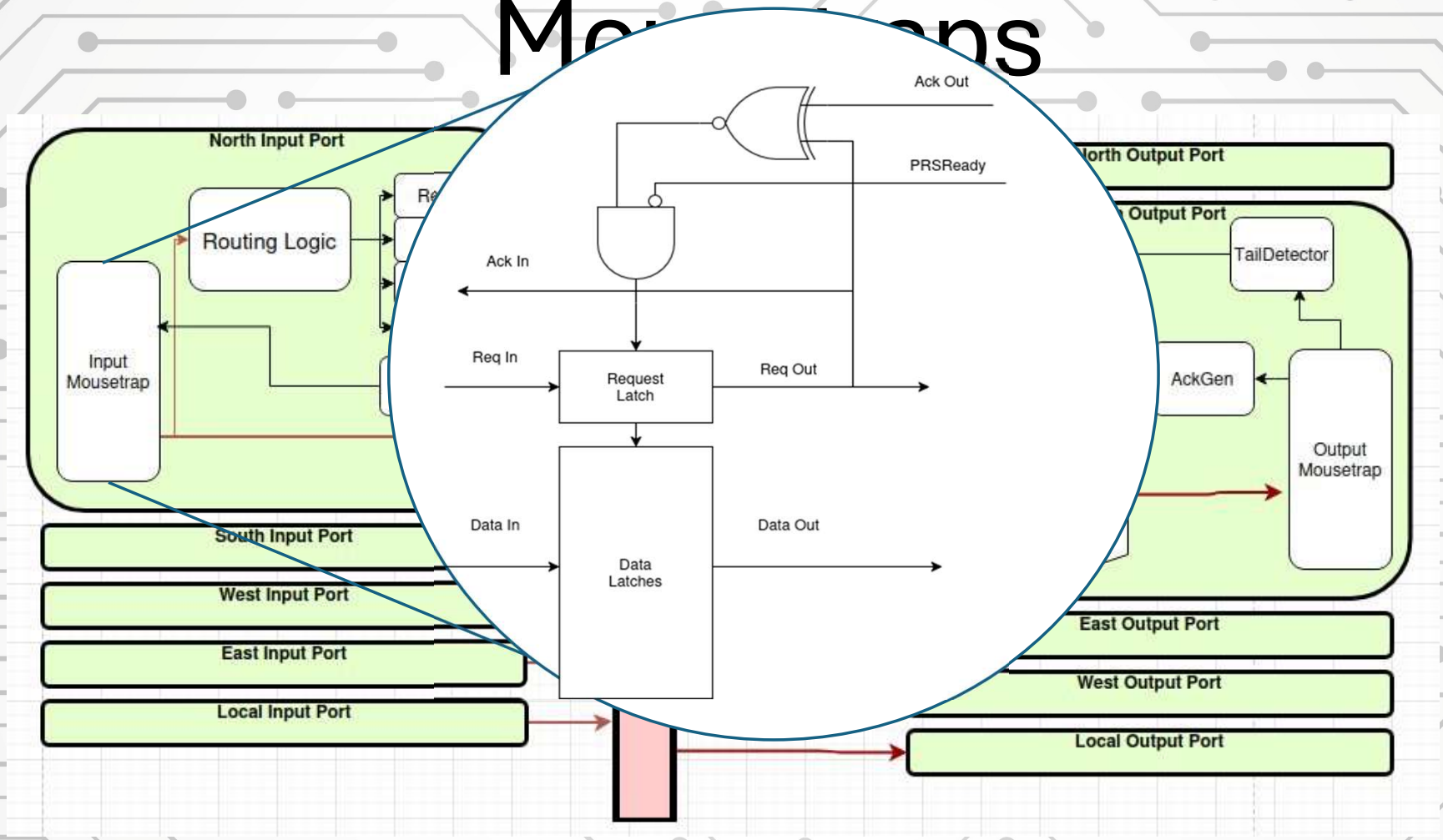
# Hardware NoC architecture



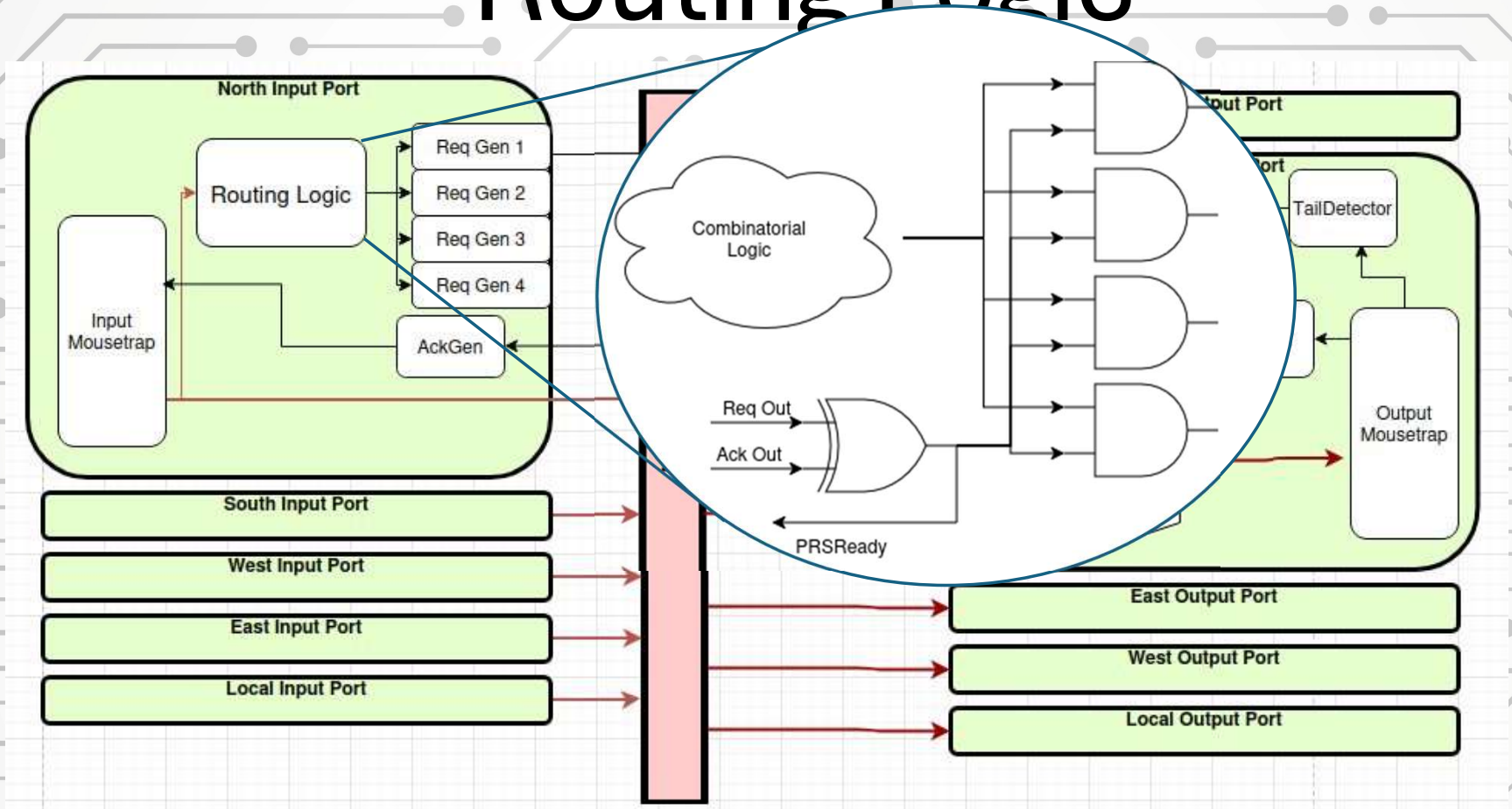

North Input Port
Routing Logic
Req Gen 1
Req Gen 2
Req Gen 3
Req Gen 4
Input Mousetrap
AckGen
South Input Port
West Input Port
East Input Port
Local Input Port
Xbar
North Output Port
South Output Port
TailDetector
Arbiter
AckGen
MUX
Output Mousetrap
East Output Port
West Output Port
Local Output Port


North Input Port
Routing Logic
Req Gen 1
Req Gen 2
Req Gen 3
Req Gen 4
Input Mousetrap
AckGen
Packet
South Input Port
West Input Port
East Input Port
Local Input Port
Xbar
North Output Port
South Output Port
TailDetector
Arbiter
AckGen
MUX
Output Mousetrap
East Output Port
West Output Port
Local Output Port

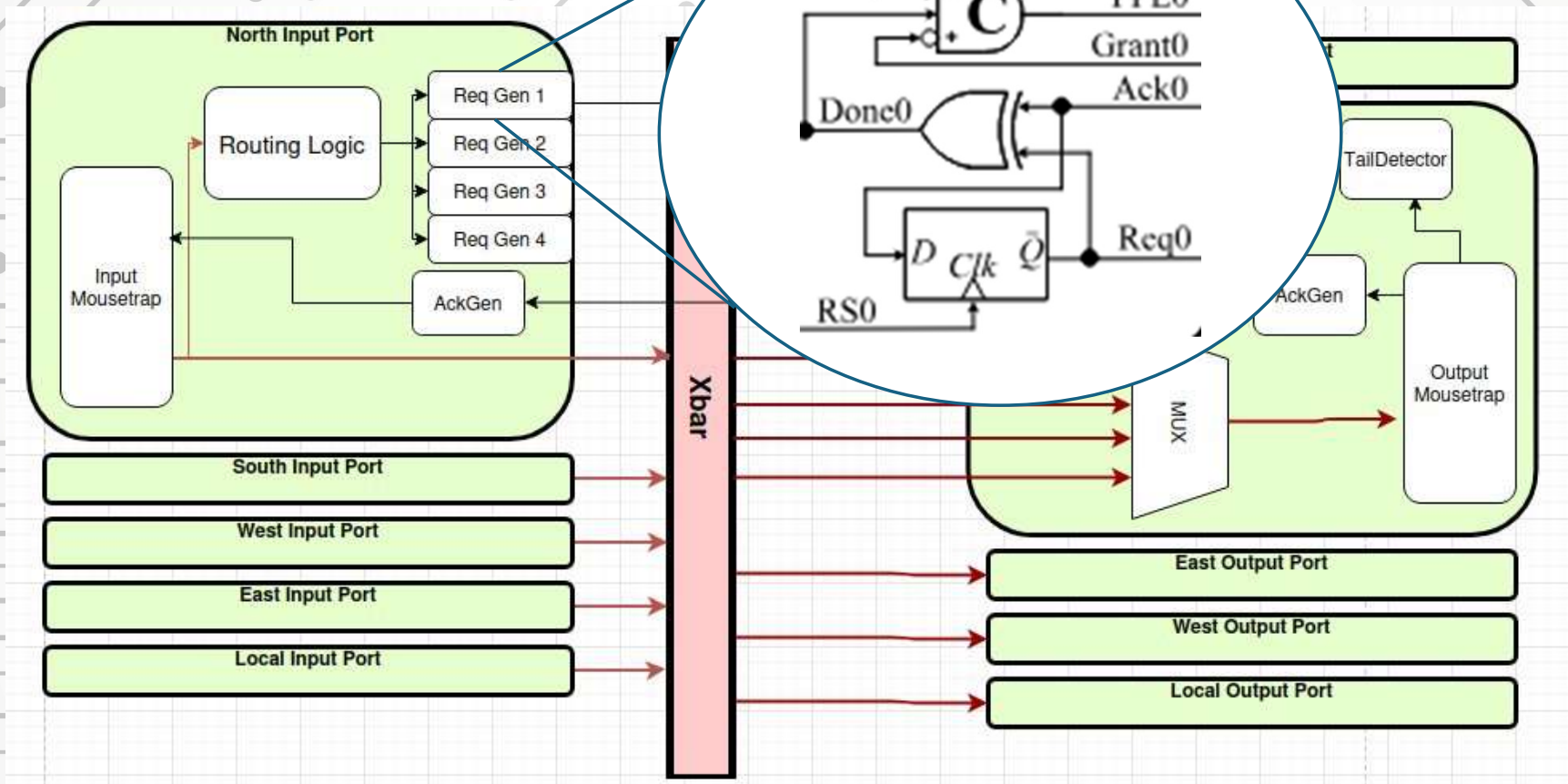
# Mo ns


North Input Port
Routing Logic
Input Mousetrap
South Input Port
West Input Port
East Input Port
Local Input Port
Ack Out
PRSReady
Ack In
Req In
Request Latch
Req Out
Data In
Data Latches
Data Out
TailDetector
AckGen
Output Mousetrap
East Output Port
West Output Port
Local Output Port

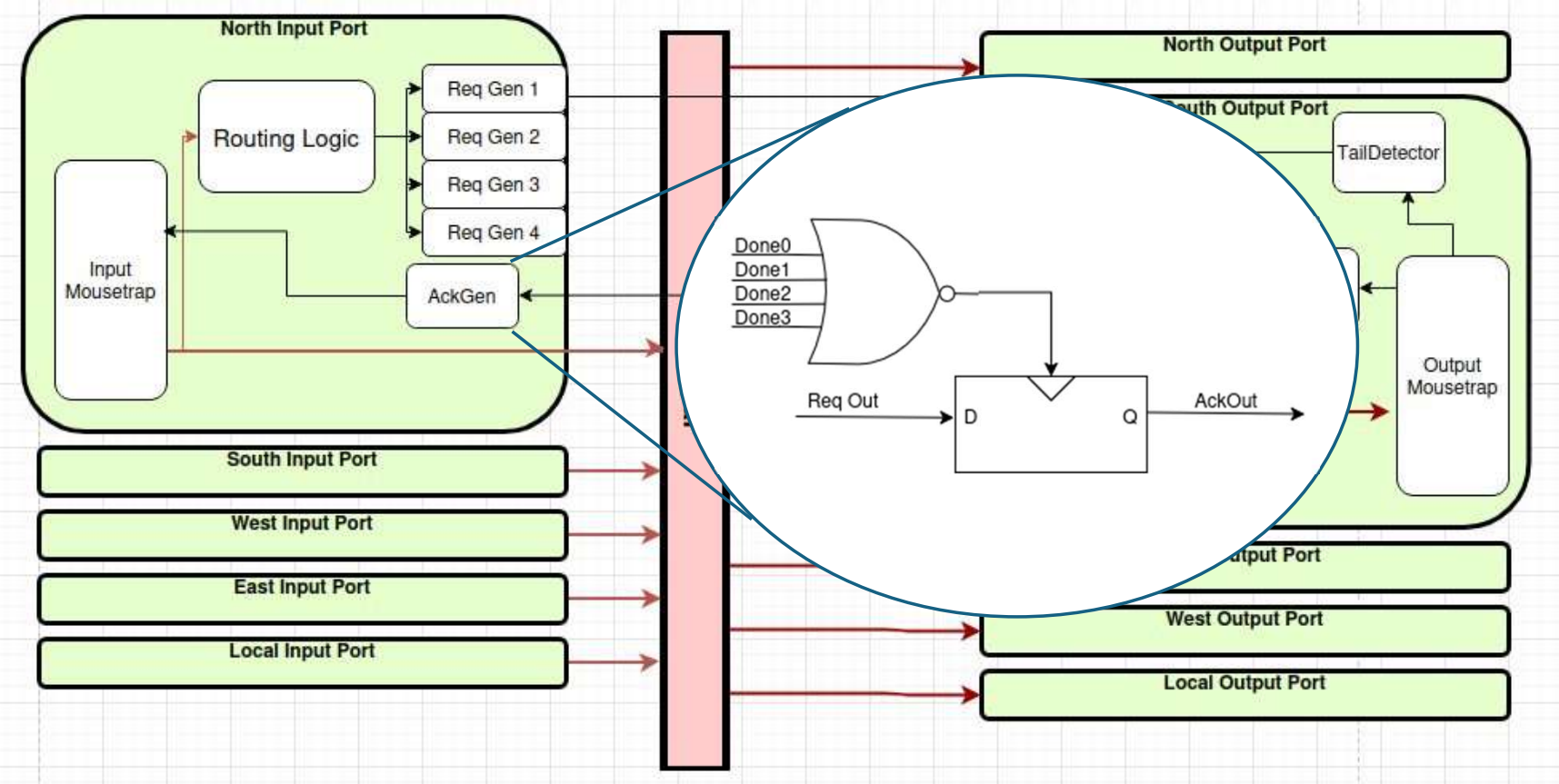
# Routing Logic



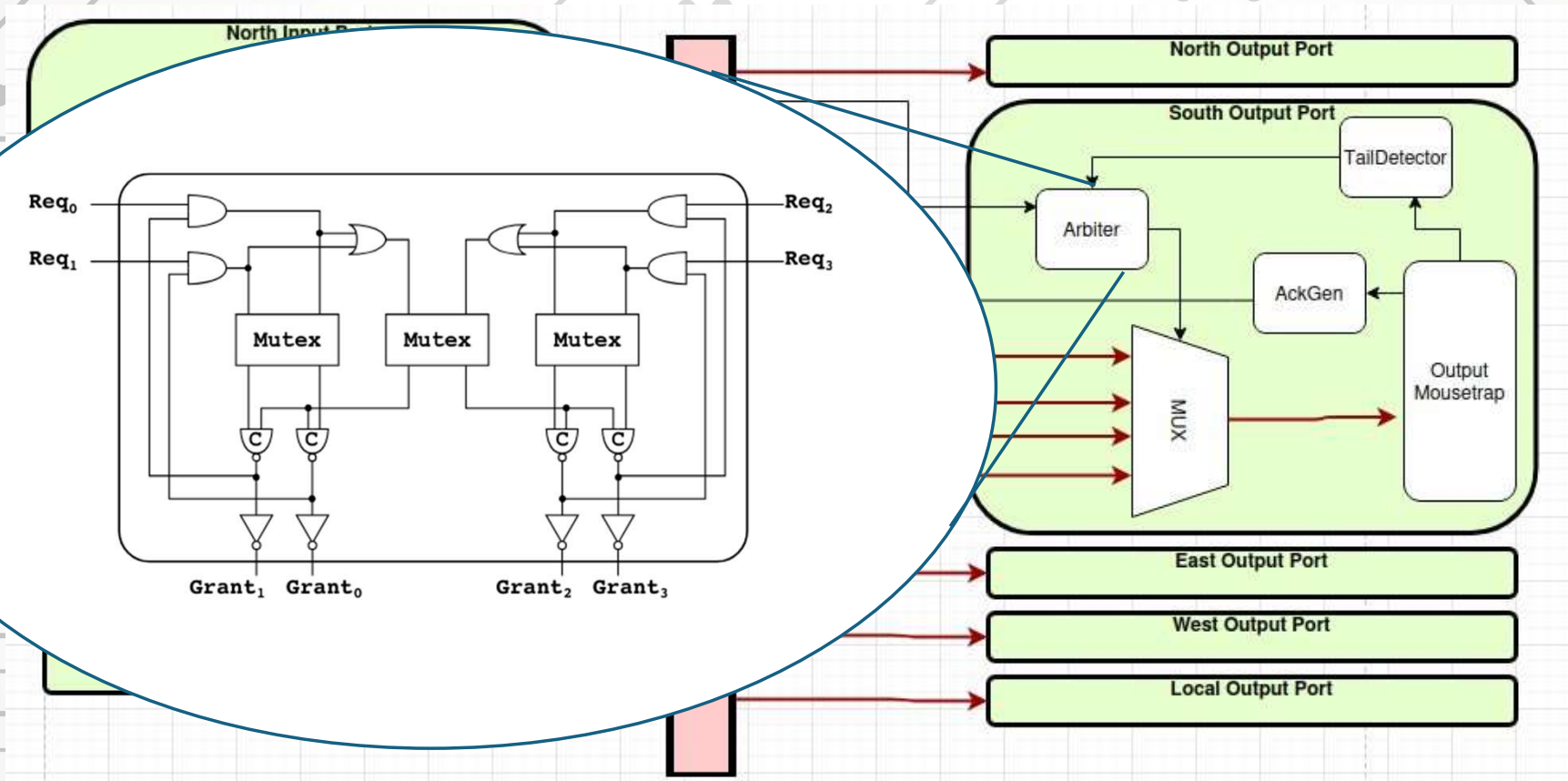
# Request G ors



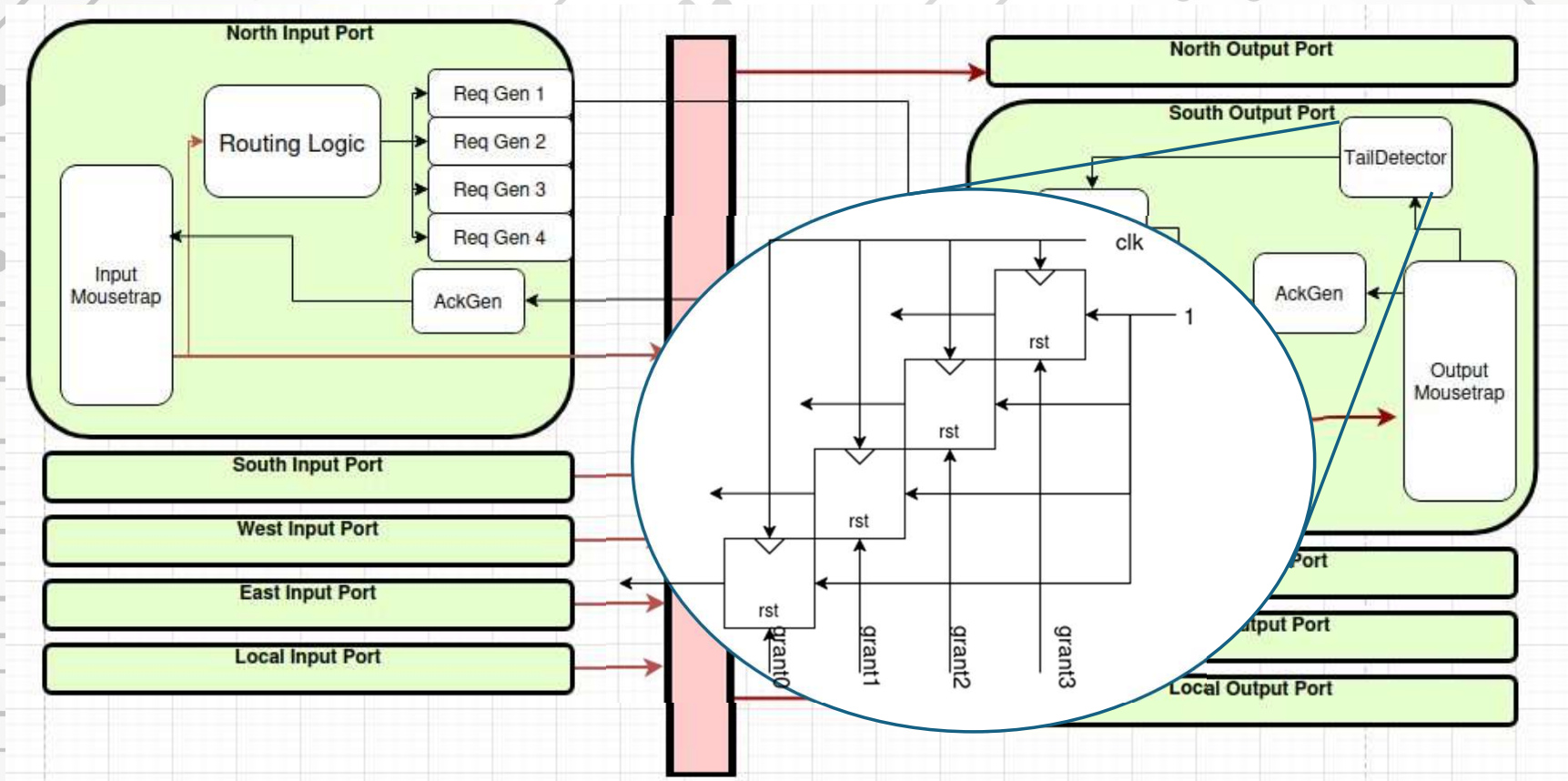
# AckGen

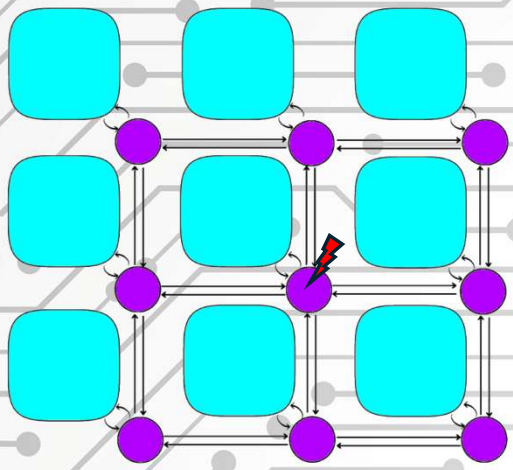

North Input Port
Routing Logic
Req Gen 1
Req Gen 2
Req Gen 3
Req Gen 4
Input Mousetrap
AckGen
South Input Port
West Input Port
East Input Port
Local Input Port
North Output Port
TailDetector
Output Mousetrap
West Output Port
Local Output Port
Done0
Done1
Done2
Done3
Req Out
D
Q
AckOut

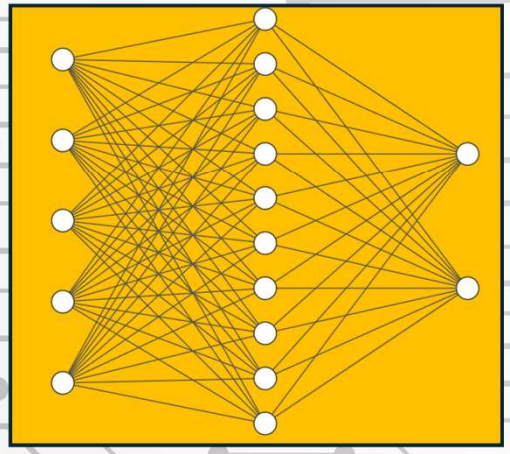
# Arbiter


North Output Port
South Output Port
TailDetector
Arbiter
AckGen
Output Mousetrap
MUX
East Output Port
West Output Port
Local Output Port
Req0
Req1
Req2
Req3
Mutex
Mutex
Mutex
C
C
C
C
Grant1
Grant0
Grant2
Grant3

# TailDetector



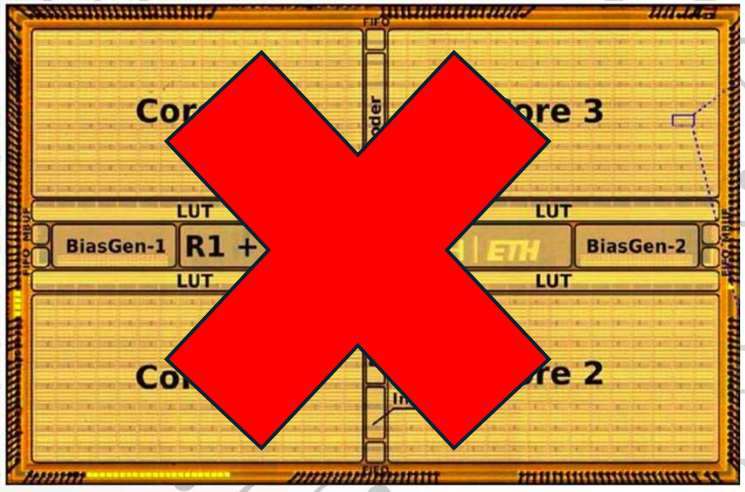




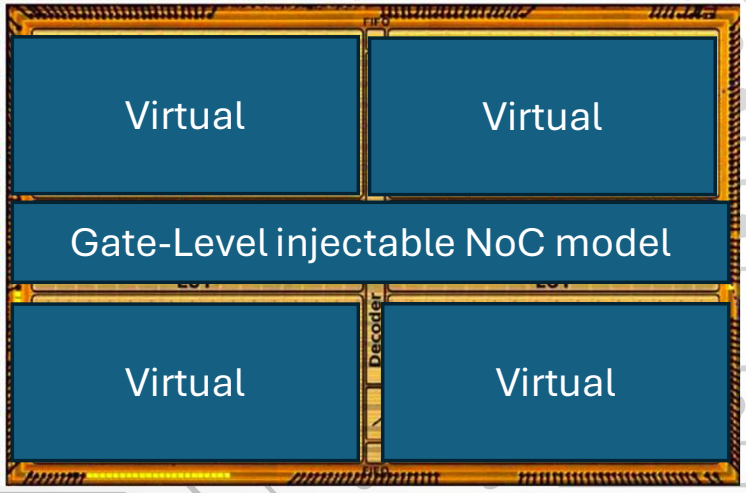
**Goal:Focus on NoC hardware faults**
(initial focus on permanent Faults)

How: not in isolation, but throught end-to-end reliability analysis.

In practice: how to capture the effect of NoC hw faults over SNN simulation?




The simulation of the whole neuromorphic processor woulb be overly costly, with accurate modeling of components(neurocores) that fall outside of the scope of this analysis.




**CoSimulation Framework:**

1. **Use virtual neuro-cores from software SNN simulators**
2. **Integrate such simulators with an HDL gate level model of the NoC**

# Software SNN simulator

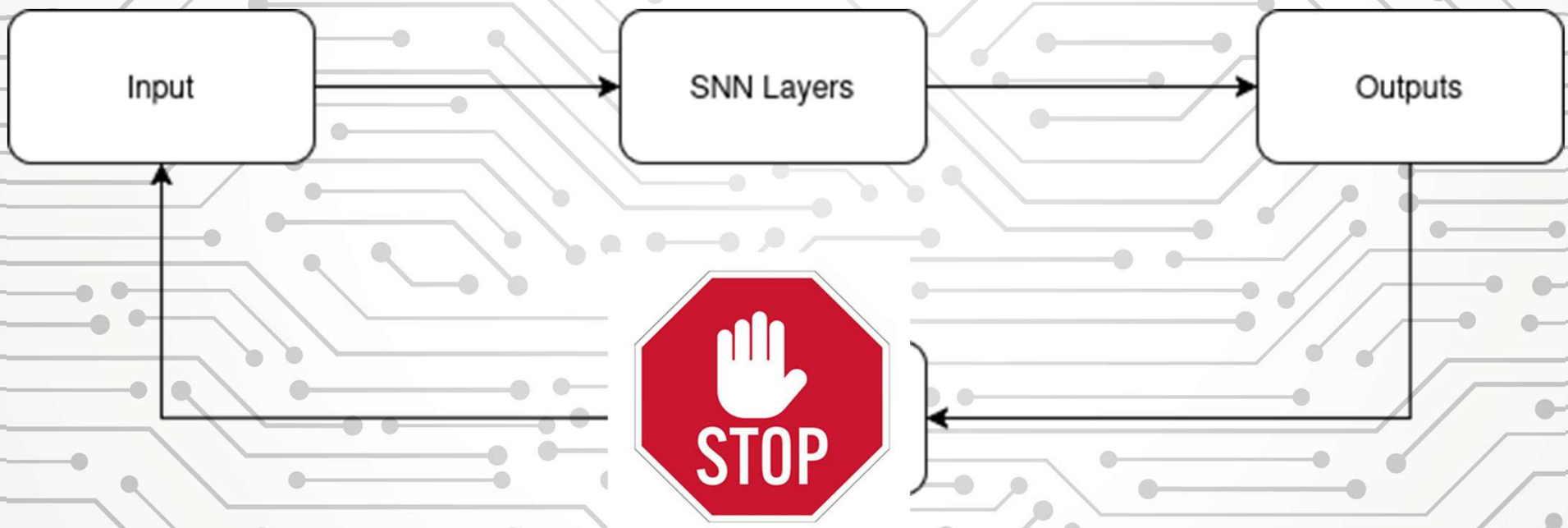
- It uses timestep driven approach
- Integrates with pyTorch
- Supports GPU acceleration and deep learning workflows

# Hardware SNN simulator

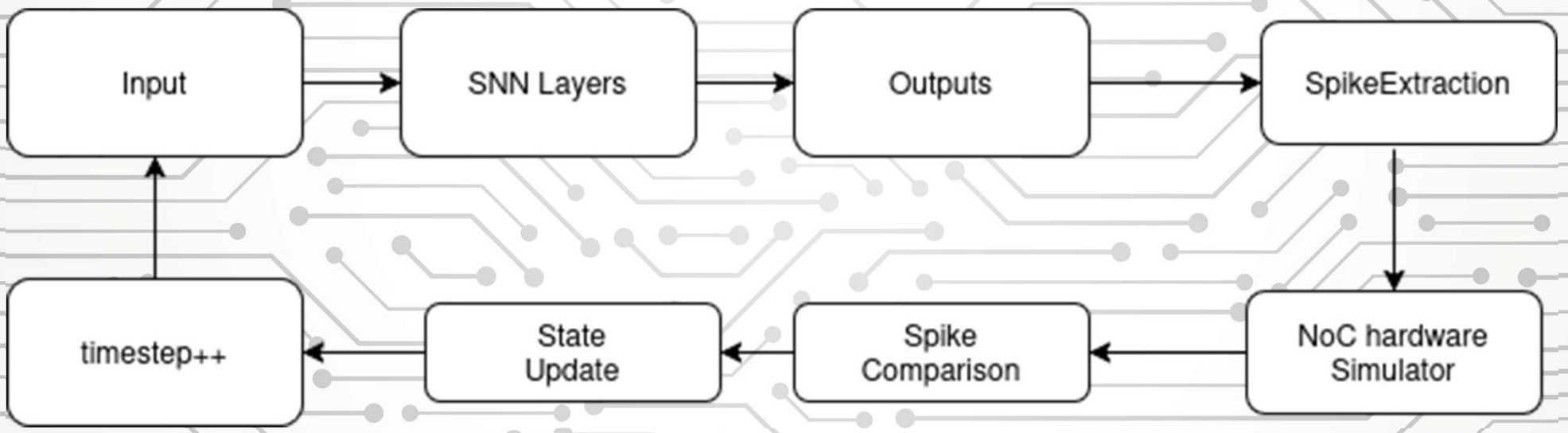
- Event driven
- Lower power consumption
- Still organized with discrete alghoritmic timesteps

**At each timestep of the SNN simulation the software simulator creates a TB for the NoC**

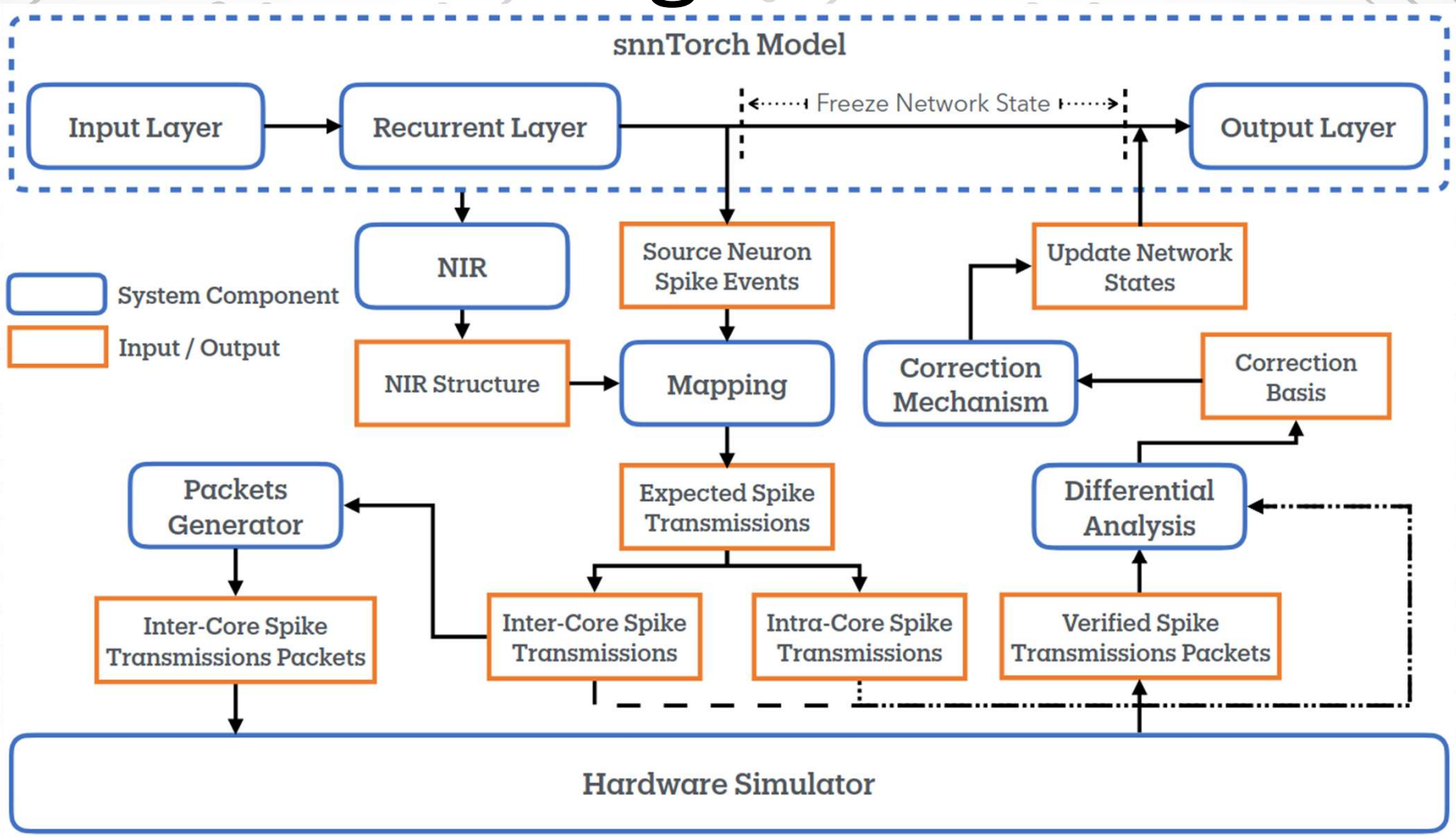
# Snntorch Simulation



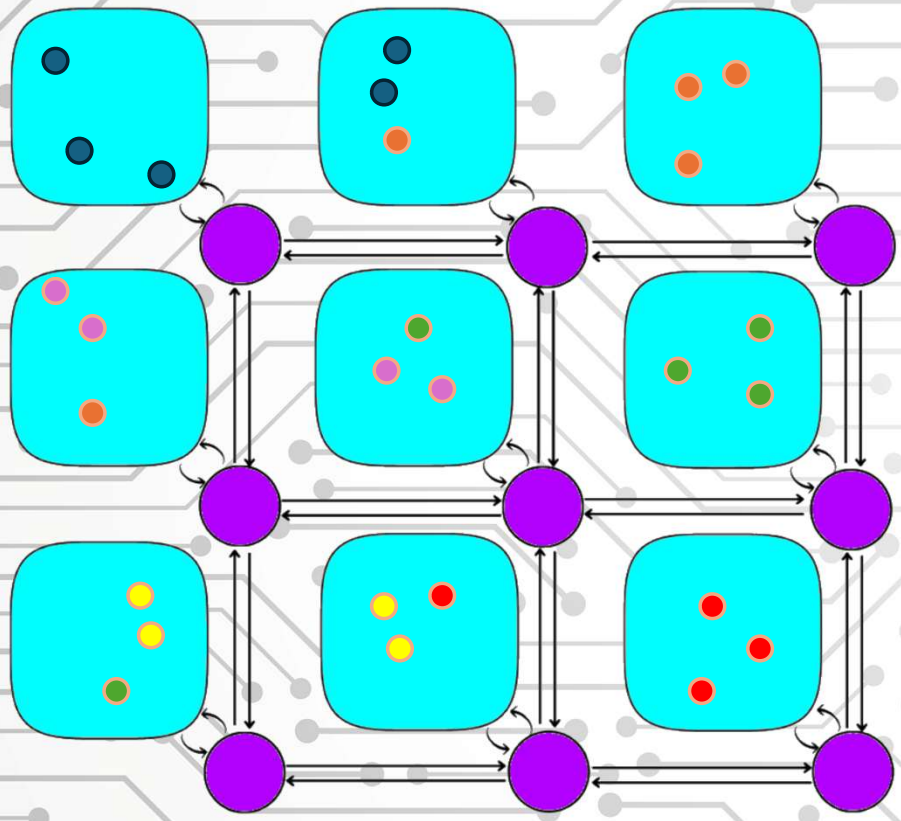
# Snntoch + NoC Simulation



# Detailed bridge frontend view



# Mapping

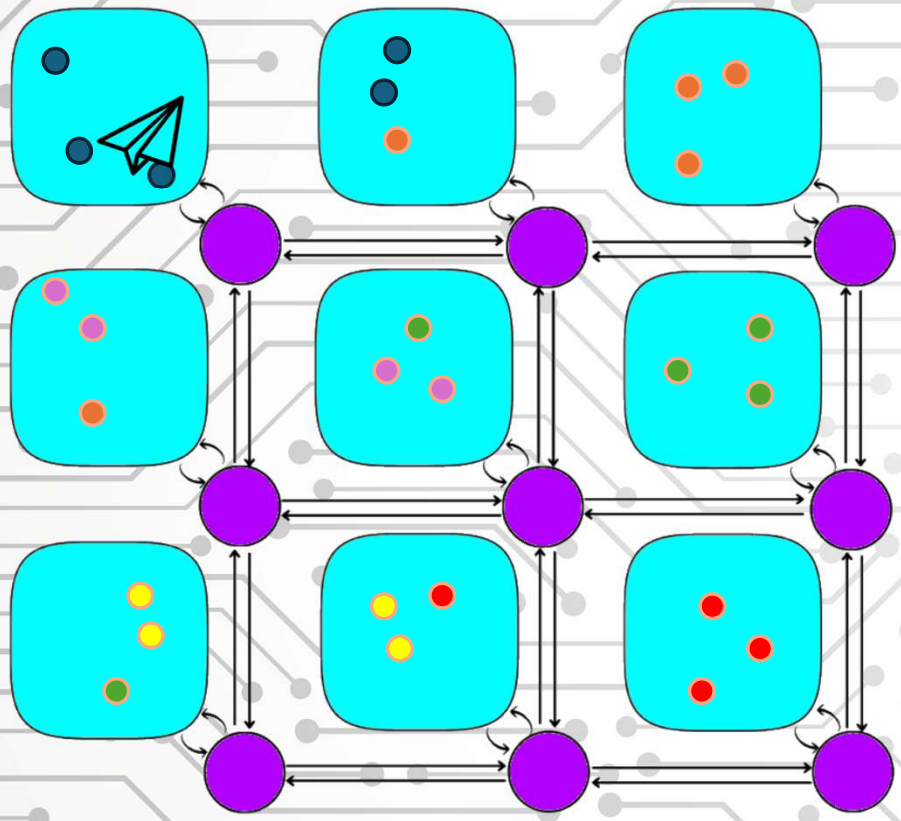


We assign neurons to phisical neuro-cores

This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Mapping

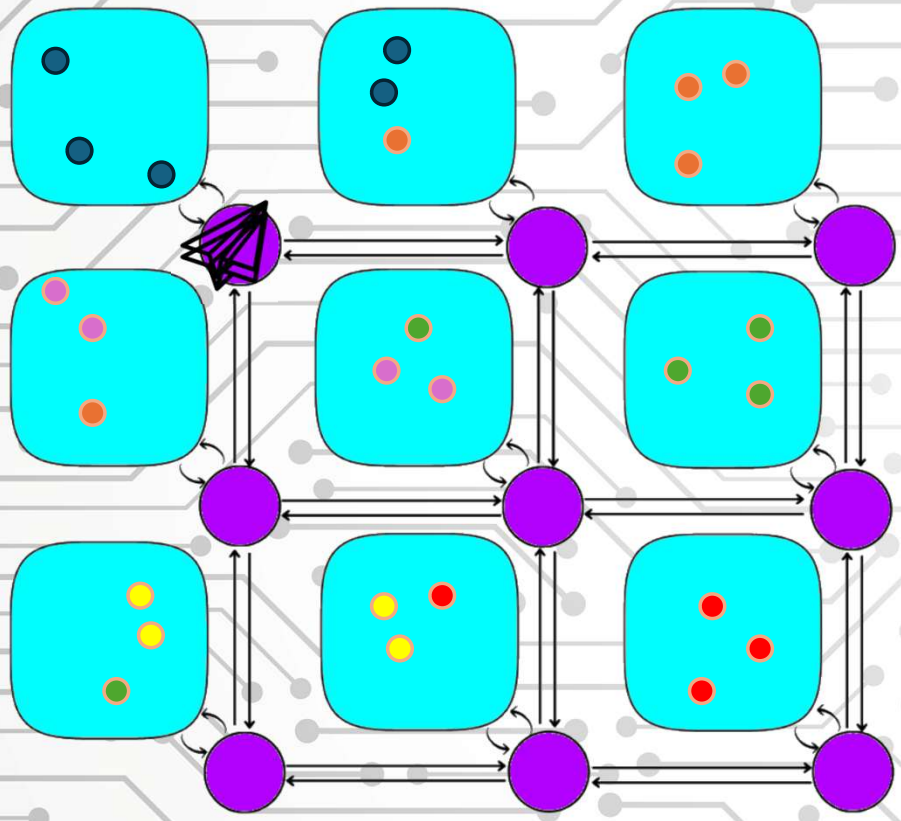


We assign neurons to phisical neuro-cores

This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Mapping

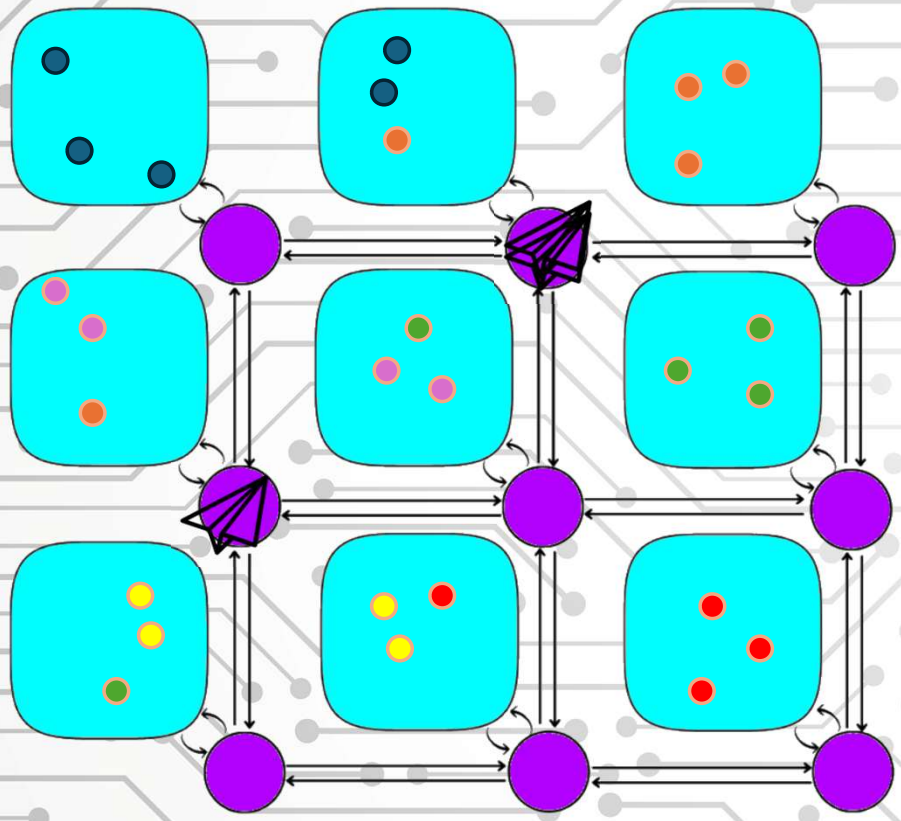


We assign neurons to phisical neuro-cores

This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Mapping

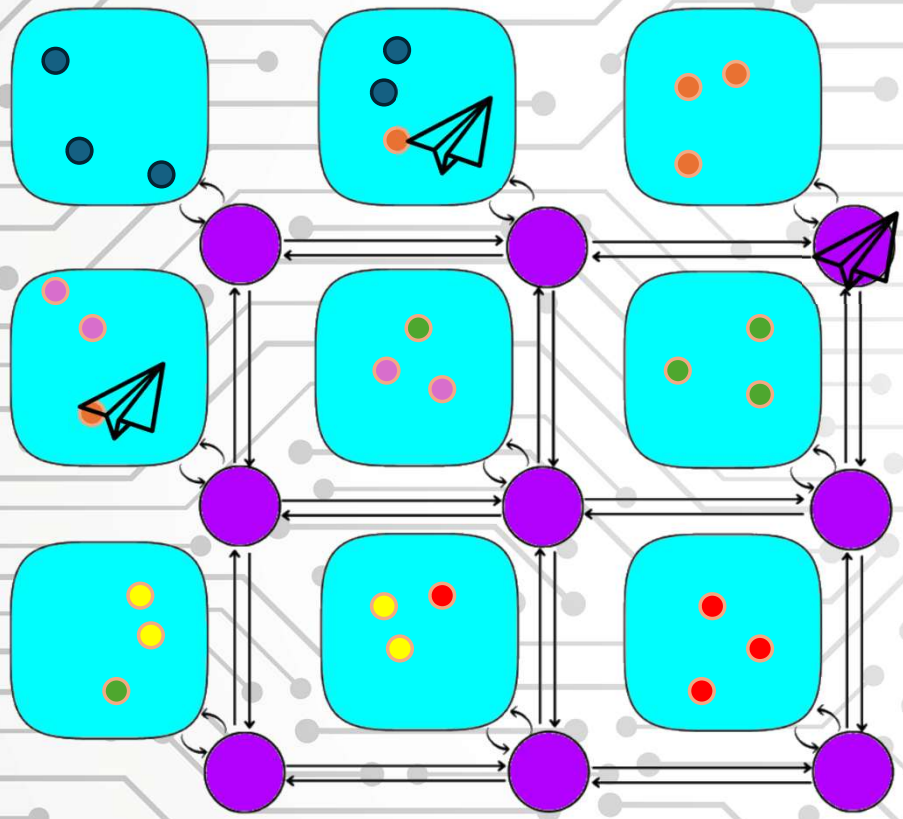


We assign neurons to phisical neuro-cores

This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Mapping

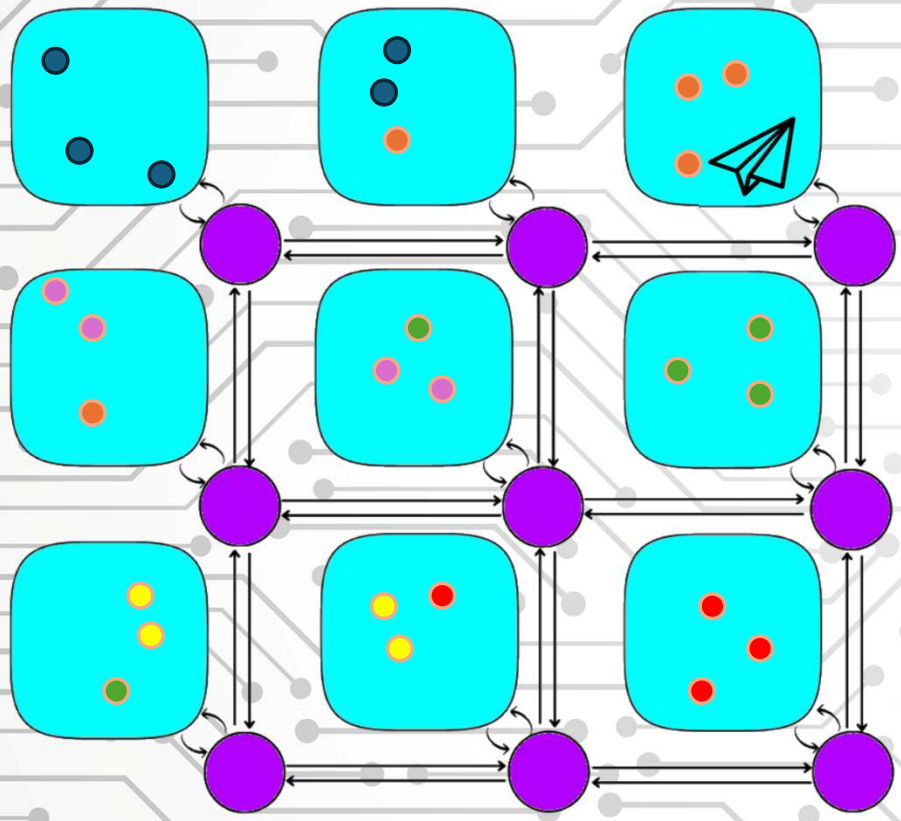


We assign neurons to phisical neuro-cores

This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Mapping

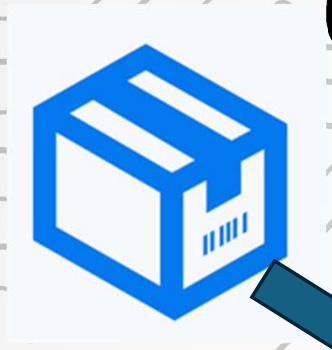


We assign neurons to phisical neuro-cores

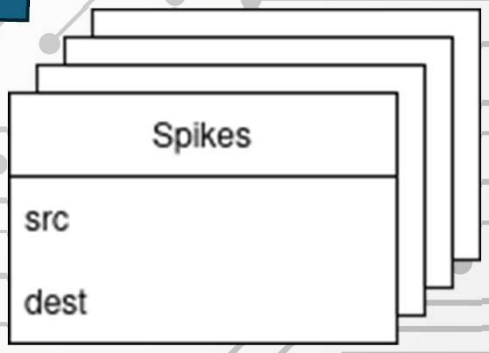
This process is necessary to generate routing informations that can be sent to the NoC simulator

Intra-core spike bypass the NoC and are handled automatically within the cores.

# Content Addressable Memories



| | Core | Source | Destinations |
|---|---|---|---|
| | Filter | Filter | Filter |
| 79 | 0 | 98 | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] |
| 80 | 0 | 99 | [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19] |
| 81 | 1 | 0 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |
| 82 | 1 | 1 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |
| 83 | 1 | 2 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |
| 84 | 1 | 3 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |
| 85 | 1 | 4 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |
| 86 | 1 | 5 | [20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39] |



# Differential analysis

| SPIKE-ID | THEORETICAL | EFFECTIVE | DIFF |
|---|---|---|---|
| 15 | ✅ | ✅ | Correrct Tx |
| 20 | ✅ | ❌ | Spike Lost |
| 22 | ❌ | ✅ | Spike Added |
| 36 | ✅ | ✅ | Correrct Tx |
| 84 | ❌ | ✅ | Spike Added |
| 85 | ✅ | ❌ | Spike Lost |

# Correction Mechanism

$$I_{\text{syn}}[t+1] = \alpha I_{\text{syn}}[t] + V(S_{\text{out}}[t]) + I_{\text{in}}[t+1]$$
$$U[t+1] = \beta U[t] + I_{\text{syn}}[t+1] - RU_{\text{thr}}$$

$$I_{\text{syn}}^{(j)}[t+1] = I_{\text{syn}}^{(j)}[t+1] \pm \sum_{i \in T_{\text{faulty}}} w_{i,j}$$

To correct the synaptic current we need to add the weight of the added spikes, and subtract the weight of the lost ones

# Correction Mechanism

$$I_{\text{syn}}^{(j)}[t+1] = I_{\text{syn}}^{(j)}[t+1] \pm \sum_{i \in T_{\text{faulty}}} w_{i,j}$$

$$I_{\text{syn}}[t+1] = \alpha I_{\text{syn}}[t] + V(S_{\text{out}}[t]) + I_{\text{in}}[t+1]$$

$$U[t+1] = \beta U[t] + I_{\text{syn}}[t+1] - RU_{\text{thr}}$$

# Correction Mechanism

$$I_{\text{syn}}^{(j)}[t] = I_{\text{syn}}^{(j)}[t] \pm \sum_{i \in T_{\text{faulty}}} w_{i,j}/\alpha$$

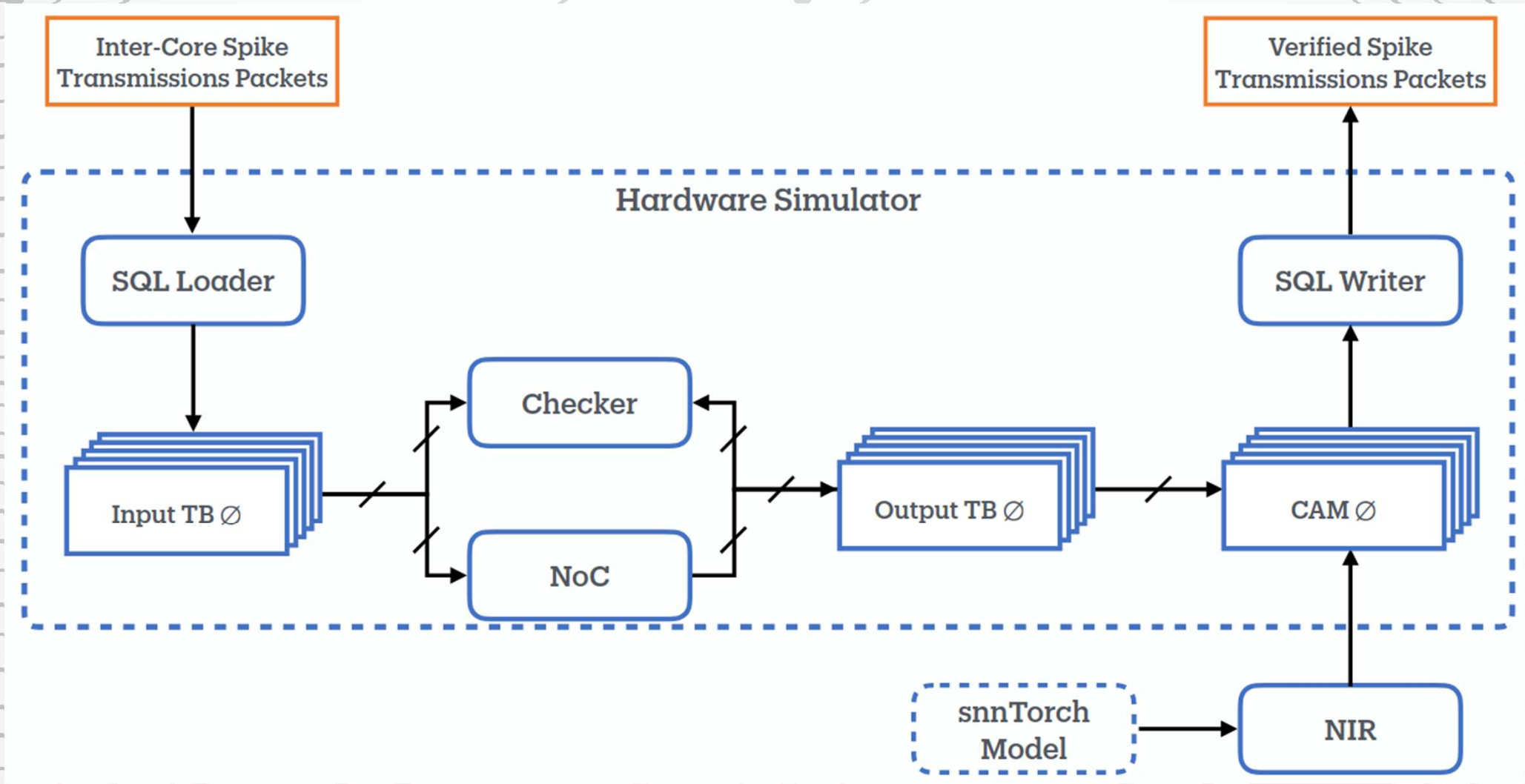
$$I_{\text{syn}}[t+1] = \alpha I_{\text{syn}}[t] + V(S_{\text{out}}[t]) + I_{\text{in}}[t+1]$$

$$U[t+1] = \beta U[t] + I_{\text{syn}}[t+1] - RU_{\text{thr}}$$

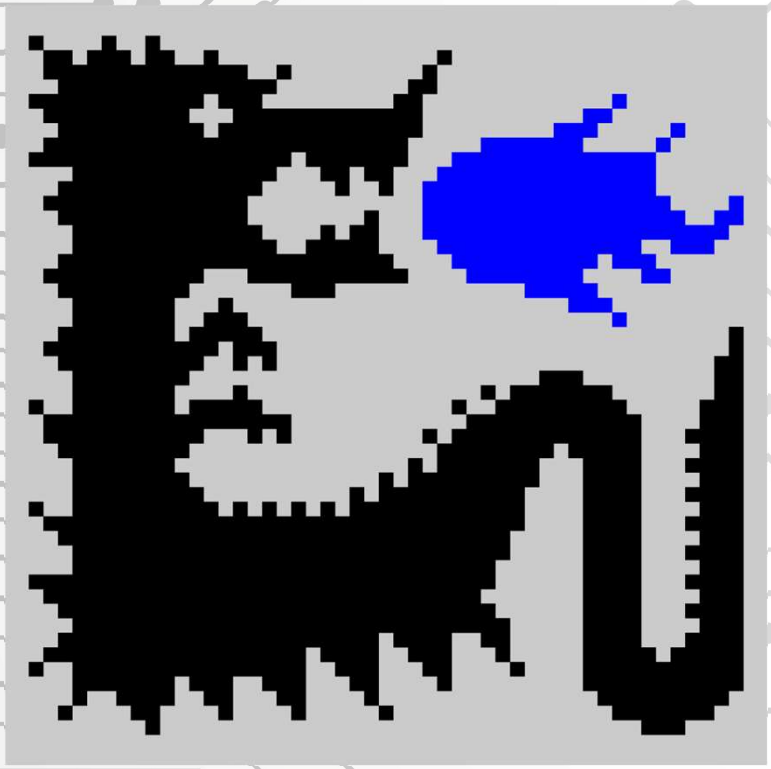
By changing the synaptic current before executing the new timestep, we ensure that the synaptic current at time t+1 is immediately correct, and this means that the membrane potential is also correctly calculated

**We do not need to correct the membrane potential!**

# Detailed bridge backend view



# The NoC model



A gate level C++ model with annotated cell timing

- average gate delays extracted via STA on a synthetized RTL using a  40nm industrial technology library

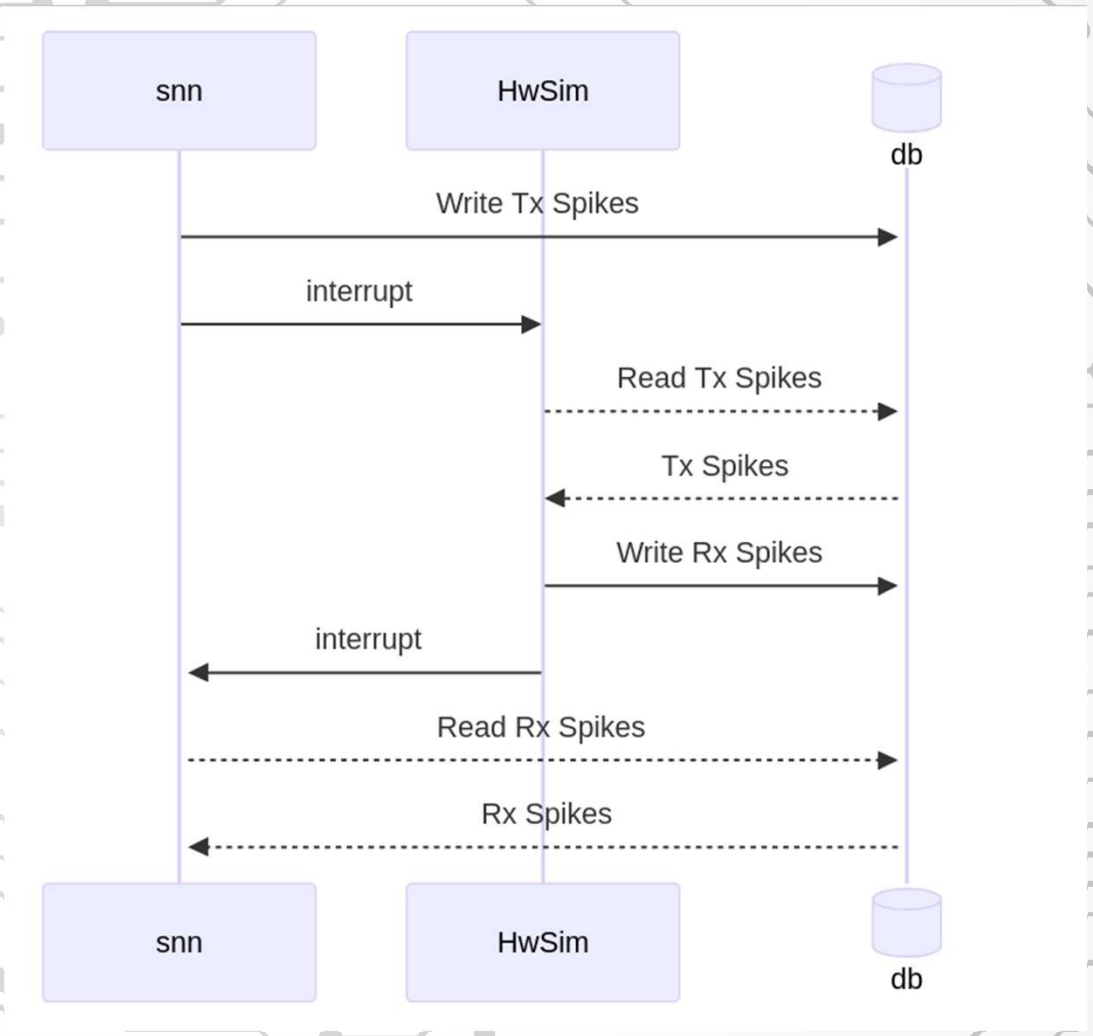
It enables:

- fast development and debugging cycles
- easier snnTorch integration
- simplified fault injection

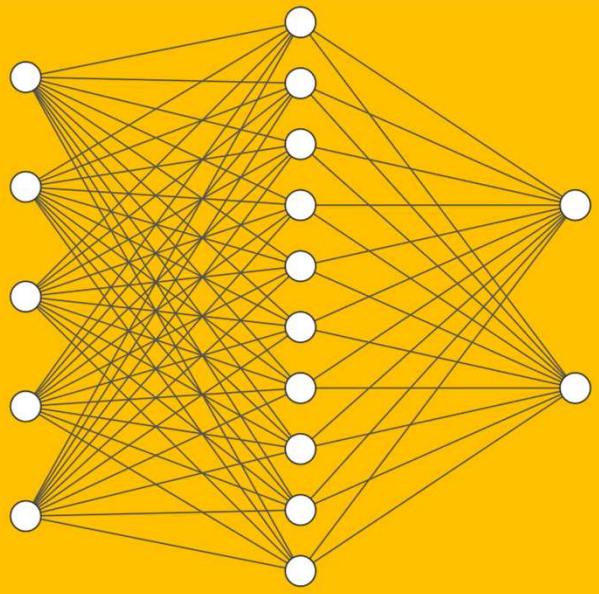
A customized version of Ember has been used has the modeling, simulation and fault injection platform

- event-driven simulation restored for async modeling
- saboteurs  philosophy preserved
- enhanced support for modularization

# Bridge architecture



# Experimental setup- snnTorch



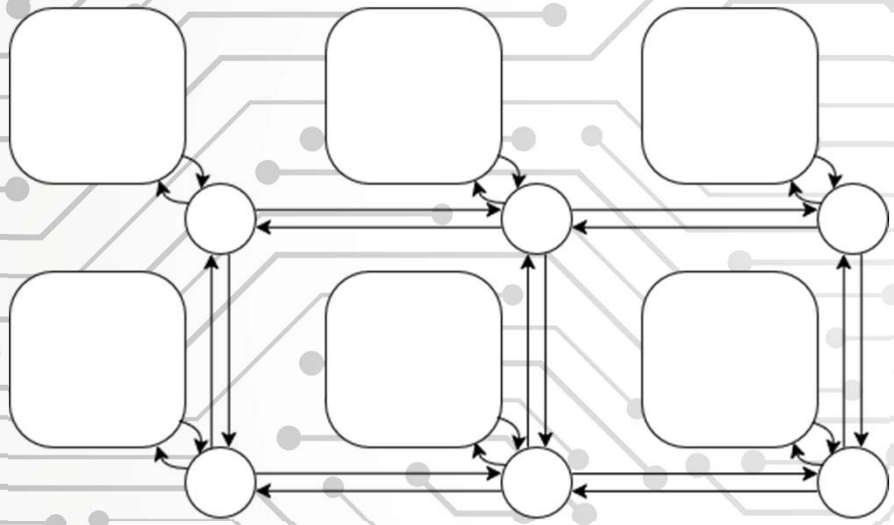
Input: 20 external neurons
Hidden layer: 100 Rsynaptic neurons
output : 2 neurons

**Binary Navigation Task**, 2-class classification

3 layer, recurrent layer**(RSNN)**

**Trained with Back propagation through time**

# Experimental setup - NoC



used 5 NeuroCores

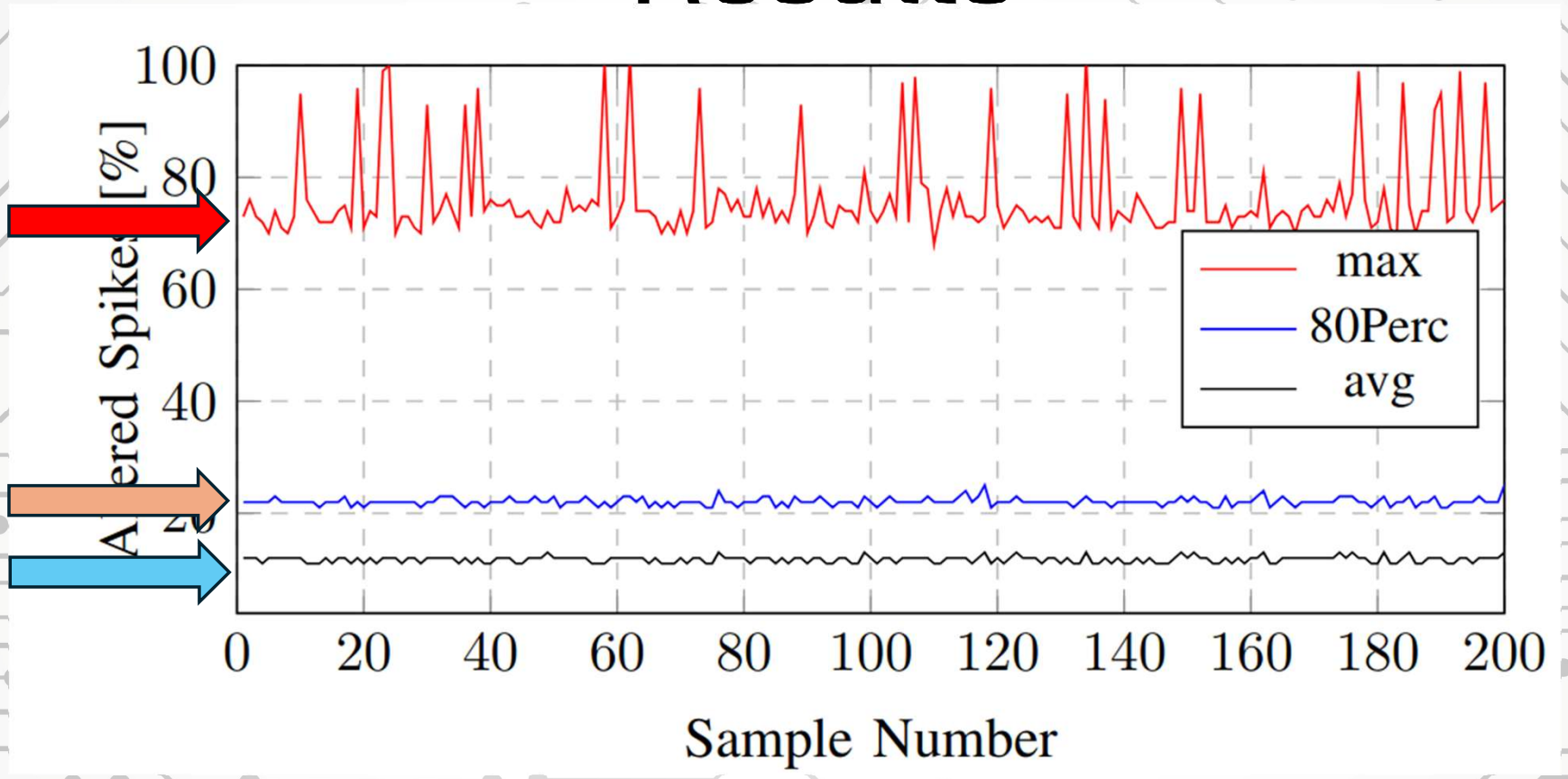
25 recurrent neurons per core

Dedicated neurocores for outputs

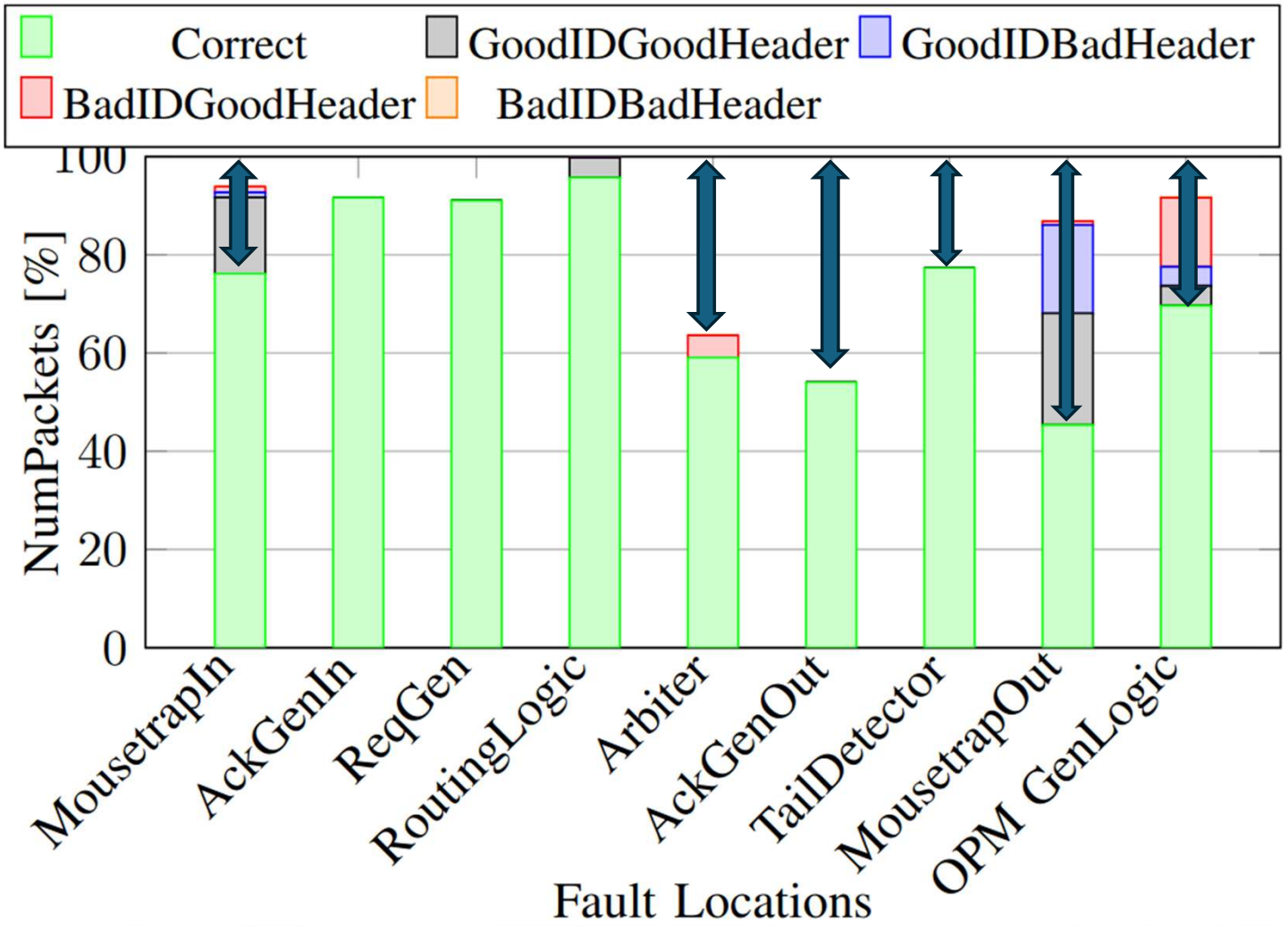
**Random** stuck-at faults injected in **NoC**

Fault injection focused on **switch control path**

# Results

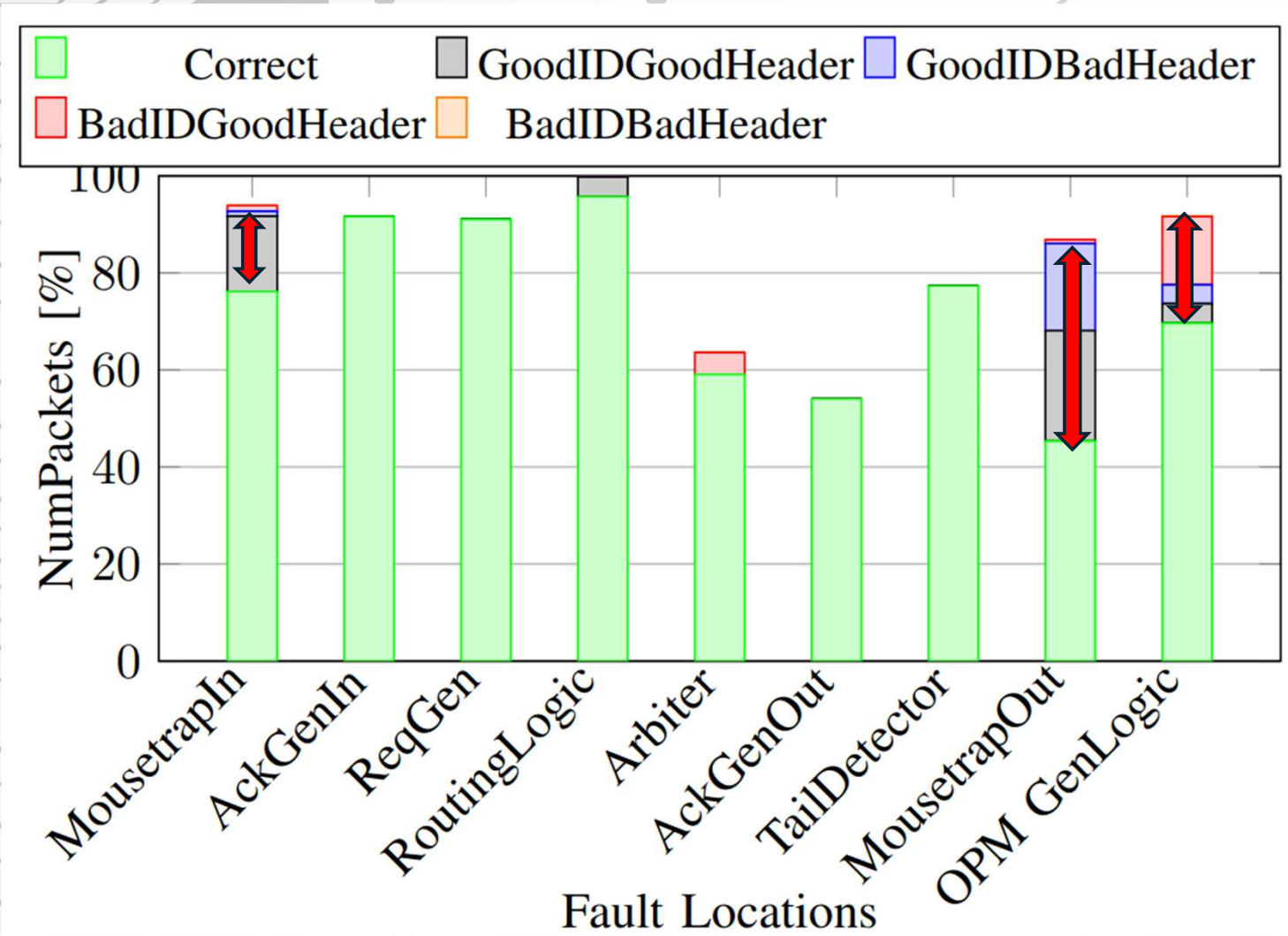



The average error introduces alterations to 10%(10000) of the total spikes.
The 80th percentile of errors introduces alterations to 20% of the spikes.
The worst case fault introduces alterations to 80/90% of the spikes

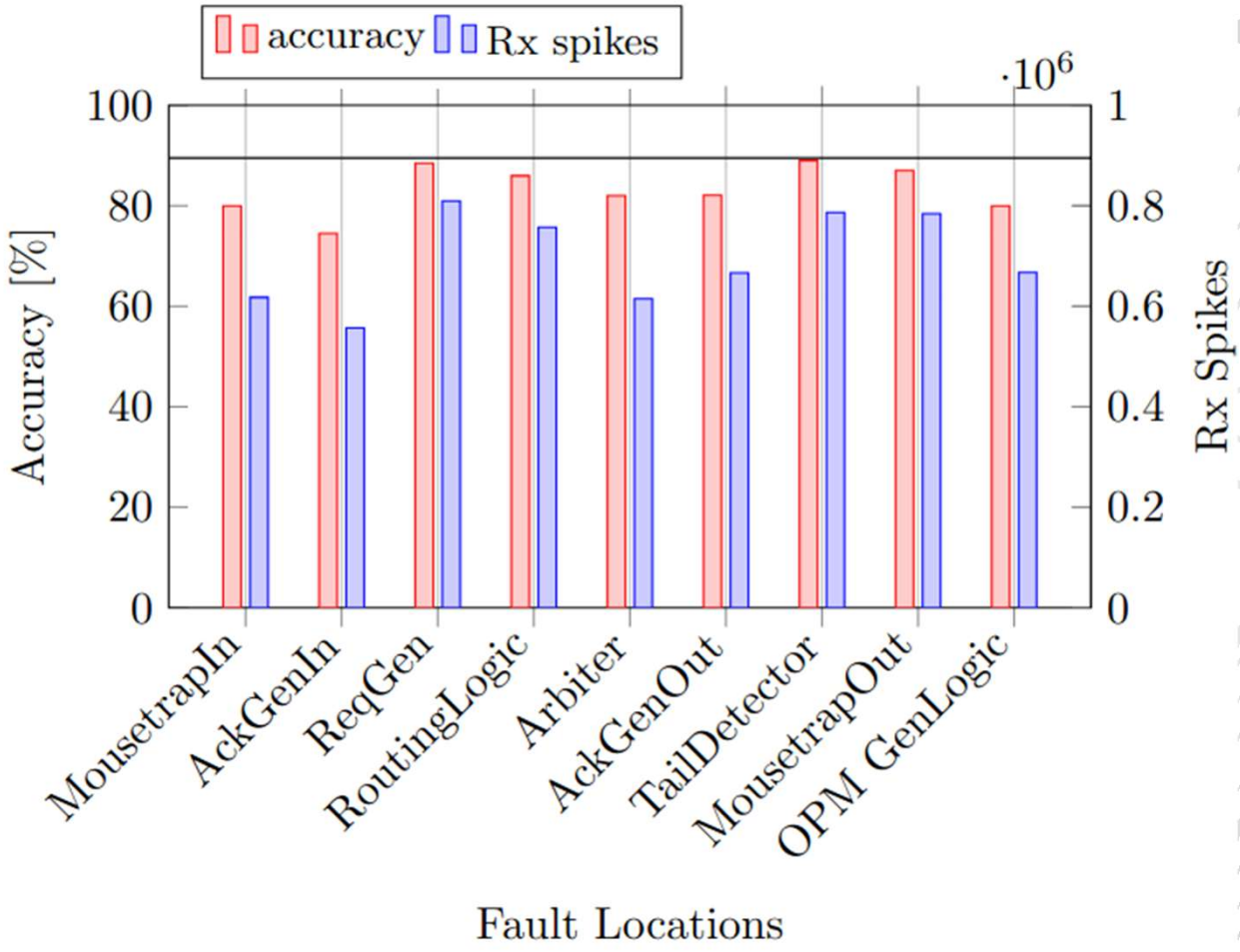



The green bars denote successful transmission

the difference between the green bars and the 100% level corresponds to spike loss.




The remaining bars correspond to packet insertions, where color variations reflect the level of correctness of the inserted packets




Loss in accuracy largely reflects data obtained in previous experiments, with a couple of notable exceptions(ie MousetrapIn and AckGenIn), probably caused by temporal interdependecies in packets transactions

The average loss in accuracy is 5%, driven by the large number of faults in the routing logic, that do not have any ripercussions

Finally we found a correlation between accuracy and number of spikes received

# Conclusions

We found that even localized faults can significantly alter the SNN spiking pattern

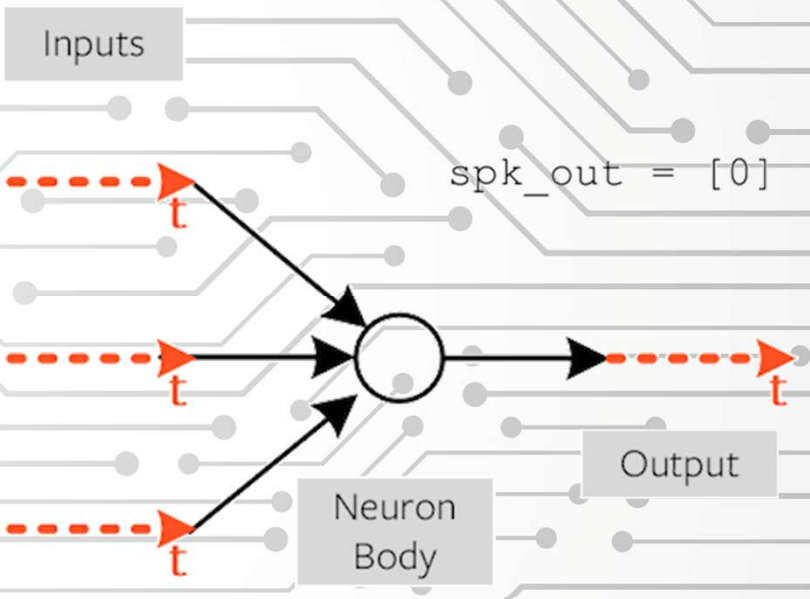
The alteration of the spiking pattern can lead to notable accuracy deviations depending on fault location

Future work will use the developed framework to abstract behavioral models of routing errors

Thanks you!
Questions?

# Neuromorphic computing

The field that aims to design systems inspired by the brain




*It developed Spiking neural networks(SNNs) where neurons communicate through discrete spikes, mimicking brain-like processing*

https://snntorch.readthedocs.io/en/latest/