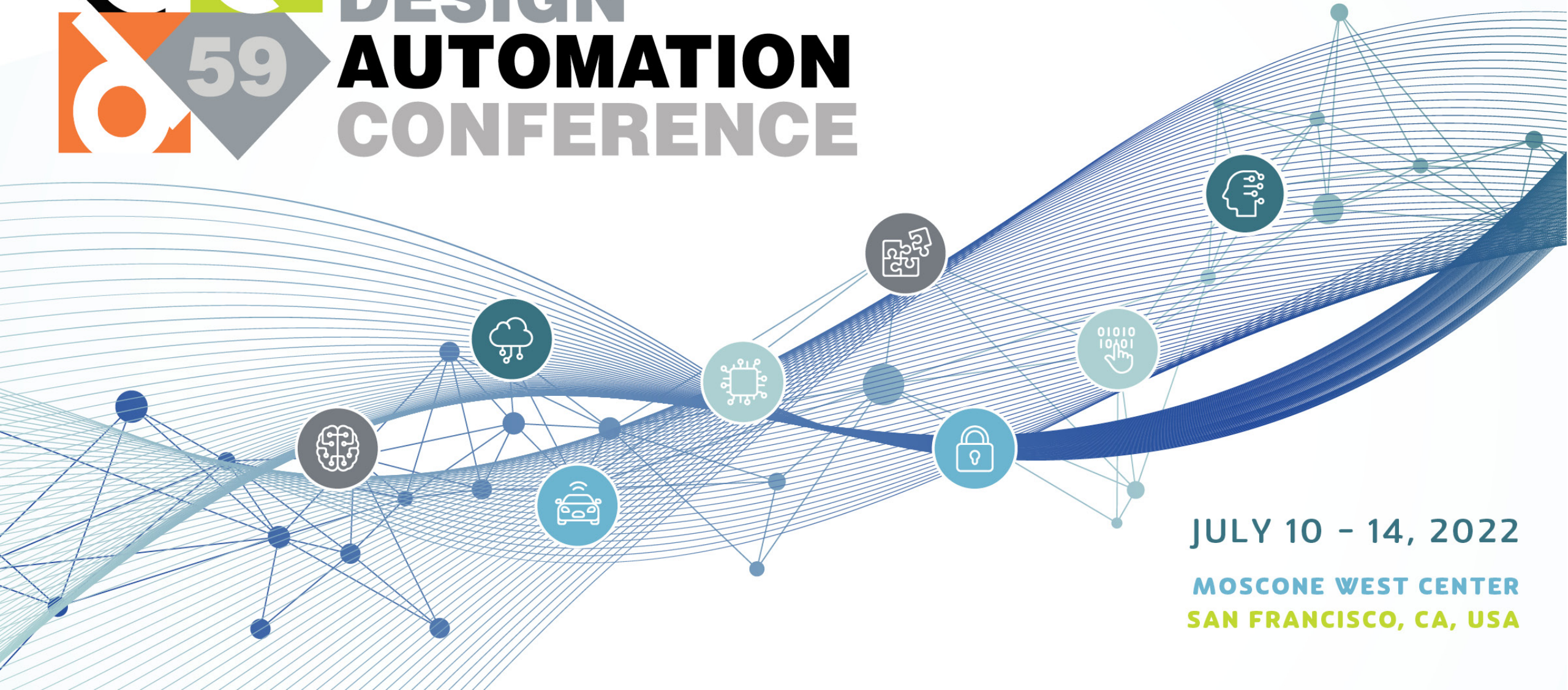




# DESIGN AUTOMATION CONFERENCE



JULY 10 - 14, 2022

MOSCONE WEST CENTER  
SAN FRANCISCO, CA, USA



# SATO: Spiking Neural Network Acceleration via Temporal-Oriented Dataflow and Architecture

**Fangxin Liu (Speaker)**

Wenbo Zhao, Zongwu Wang, Yongbiao Chen, Tao Yang, Zhezhi He, Xiaokang Yang and **Li Jiang\***

Shanghai Jiao Tong University



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY



人工智能研究院

Artificial Intelligence Institute



先进计算机体系结构实验室  
Advanced Computer Architecture Laboratory



# Outline

- Background and motivation
- Proposal: Spiking Neural Network Acceleration via Temporal-Oriented Dataflow and Architecture
- Design and implementation details
- Experiment results
- Conclusion

# DNN Applications in Edge Devices



Automatic Drive



Face Recognition



Speech Recognition



Translation

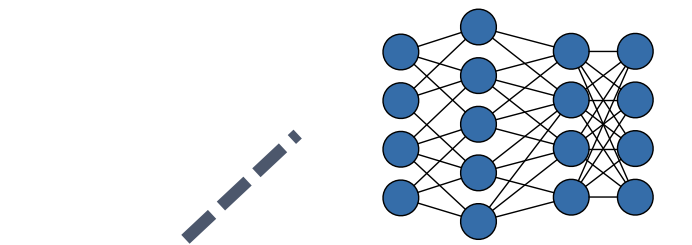
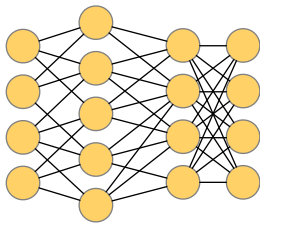
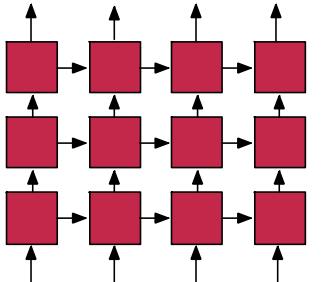


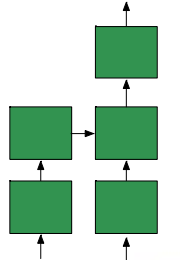
Image DNN



DNN



Speech DNN



Language DNN

Diverse DNN models and use cases on various devices

## Challenges



Energy

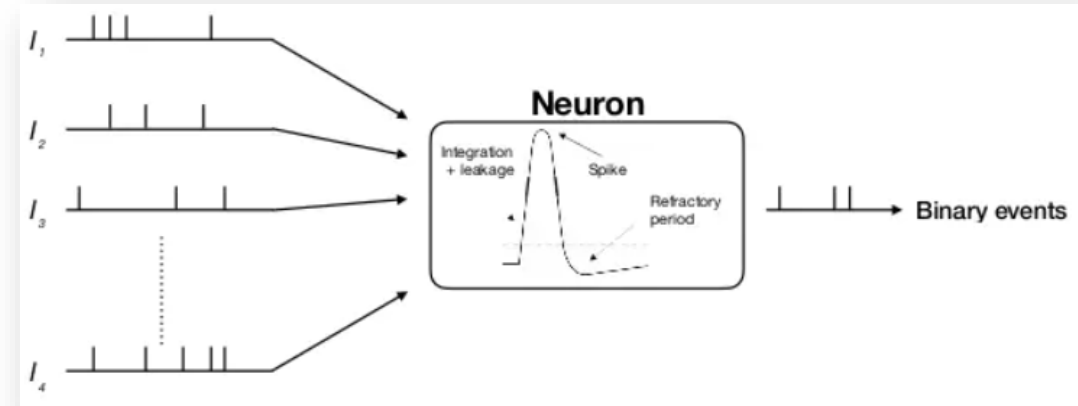


Latency

# Spiking Neural Network

## SNN:

- Increasing the information density due to the spike train over the time window
- Sparse additions



Spiking NN

## Features

### Requirement of repeated processing spike over the time window

- SNNs iterate over all the neurons for each time step
- Accumulating spikes over multiple time steps leads to more operations
- A larger number of time steps represent the longer network latency.

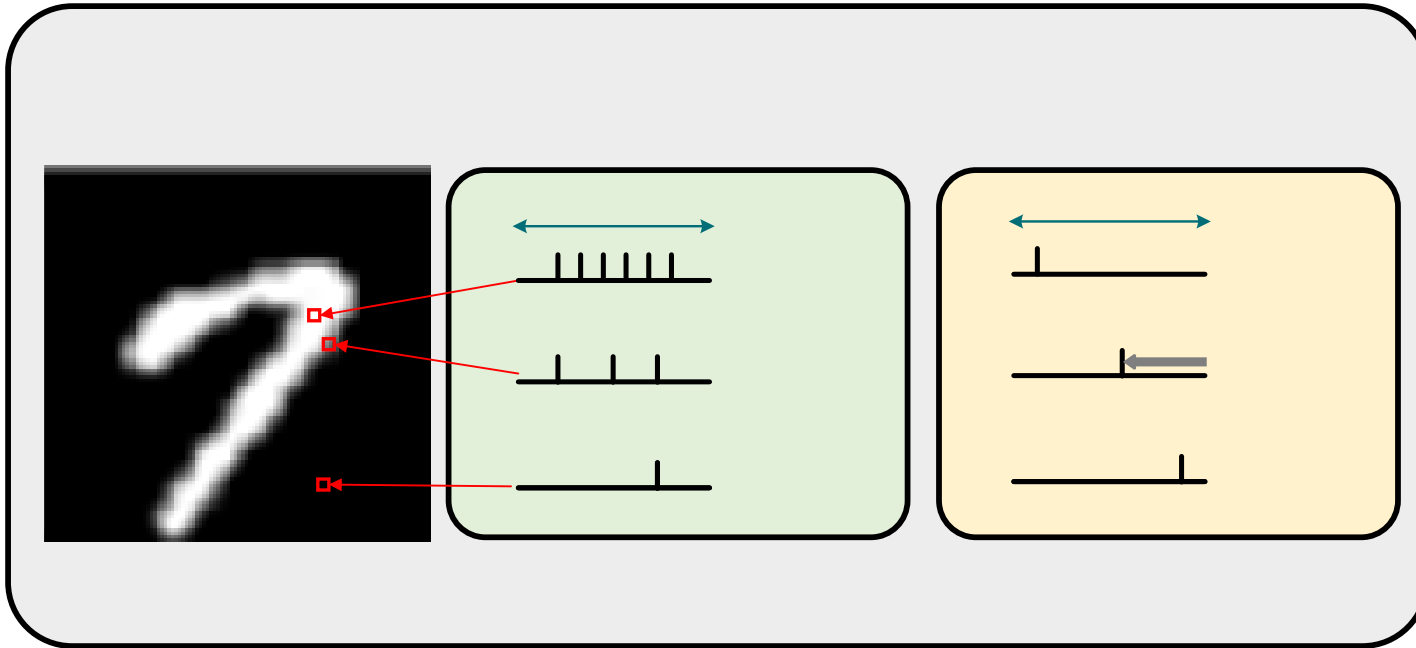
Energy



Latency



# The Types of SNNs



Recent studies have shown temporal-encoded SNNs with their **inherent higher sparsity** and ability to encode temporal information in inputs can match the **accuracy of an ANN, even on large-scale datasets.**



## Rate-encoded SNN:

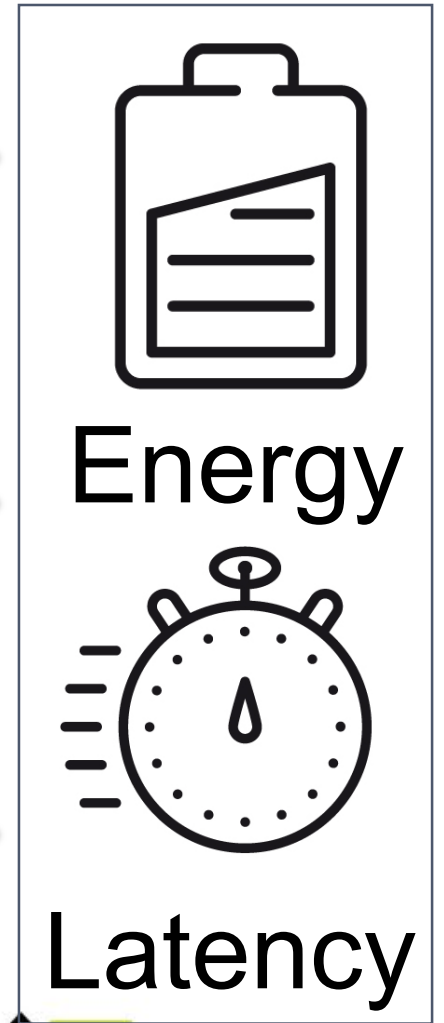
- converts an input pixel value into the spike train consist of **multiple spikes.**



## Temporal-encoded SNN:

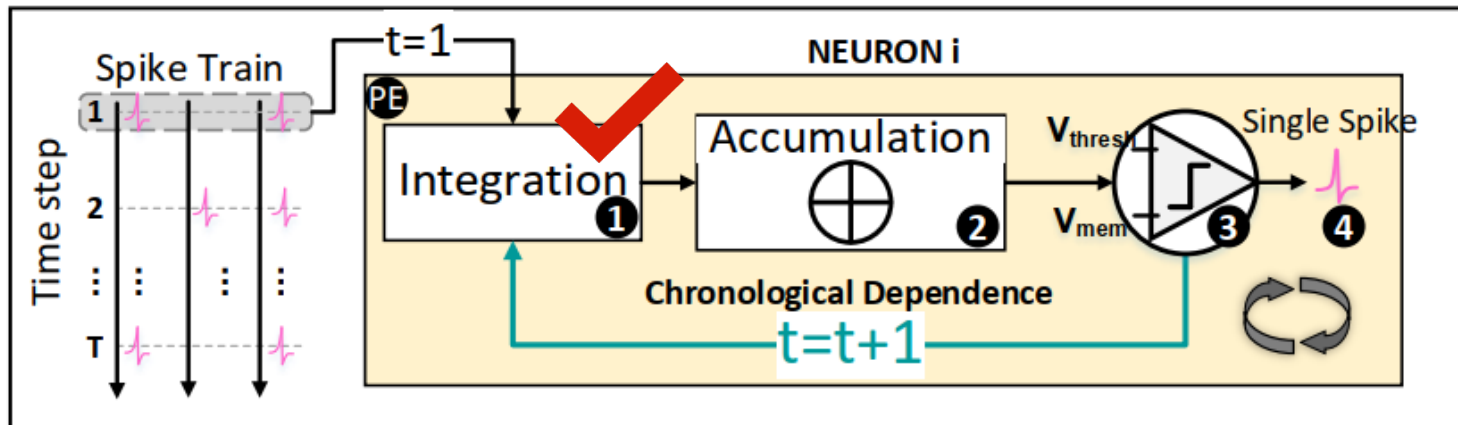
- converts an input pixel value into the spike train consist of **the single spike.**

# What Determines SNN Accelerator Efficiency





# Existing dataflow in SNN accelerators



| Code Snippet  | Module Name      | # Modules | # Cycles |
|---|------------------|-----------|----------|
| for t = 0 to T - 1 {<br>for n = 0 to N - 1 {              | Integrate Spikes | T × M     | $O(T)$   |
| PSUM = 0;   | Comparator       | T         | $O(T)$   |
| for m = 0 to M - 1 {                                      | Accumulator      | T         | $O(T)$   |
| PP <sub>n,t</sub> += TS <sub>t</sub> × W <sub>m,n</sub> ; | Total Cycles     |           | $O(NT)$  |
| }   |                  |           |          |
| PSUM += PP <sub>n,t</sub> ;                               |                  |           |          |
| if PSUM > Threshold {                                     |                  |           |          |
| Fire the spike of neuron n at time-step t;                |                  |           |          |
| }   |                  |           |          |
| }   |                  |           |          |
| }   |                  |           |          |

The accelerator processes the spikes as follows:

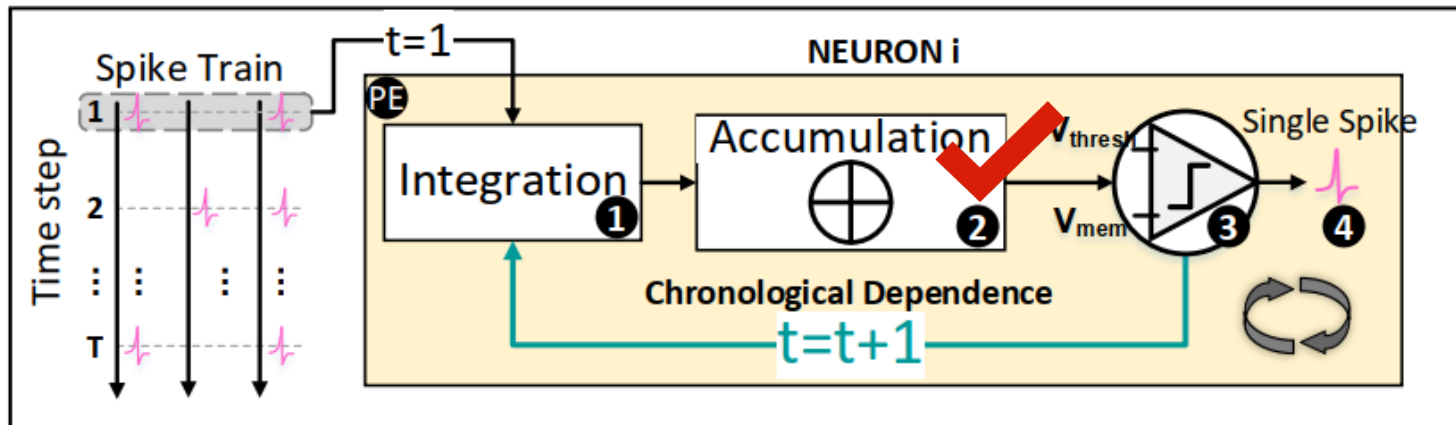
1. integrate the spikes at time step t;

(a) Conventional Design (SpinalFlow, Eyeriss-based, etc.)

Map the post-synaptic neurons calculations onto PEs in parallel and integrate spikes along the time steps in serial.



# Existing dataflow in SNN accelerators



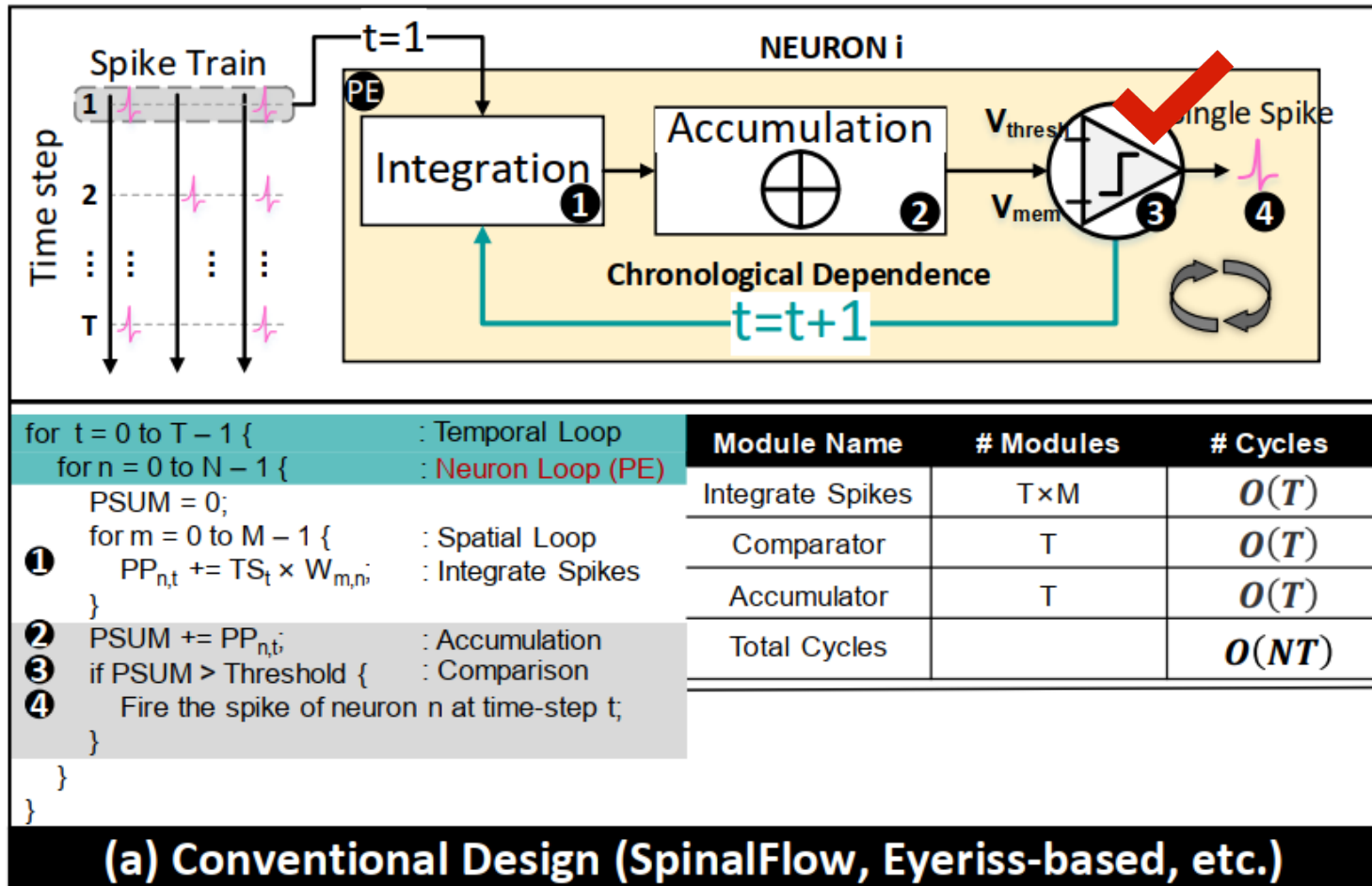
| Code Snippet  | Module Name      | # Modules | # Cycles |
|---|------------------|-----------|----------|
| for t = 0 to T - 1 {<br>for n = 0 to N - 1 {              | Integrate Spikes | T × M     | $O(T)$   |
| PSUM = 0;   | Comparator       | T         | $O(T)$   |
| for m = 0 to M - 1 {                                      | Accumulator      | T         | $O(T)$   |
| PP <sub>n,t</sub> += TS <sub>t</sub> × W <sub>m,n</sub> ; | Total Cycles     |           | $O(NT)$  |
| }   |                  |           |          |
| PSUM += PP <sub>n,t</sub> ;                               |                  |           |          |
| if PSUM > Threshold {                                     |                  |           |          |
| Fire the spike of neuron n at time-step t;                |                  |           |          |
| }   |                  |           |          |
| }   |                  |           |          |
| }   |                  |           |          |

The accelerator processes the spikes as follows:

1. integrate the spikes at time step t;
2. accumulate the integrated spike to the membrane potentials

Map the post-synaptic neurons calculations onto PEs in parallel and integrate spikes along the time steps in serial.

# Existing dataflow in SNN accelerators

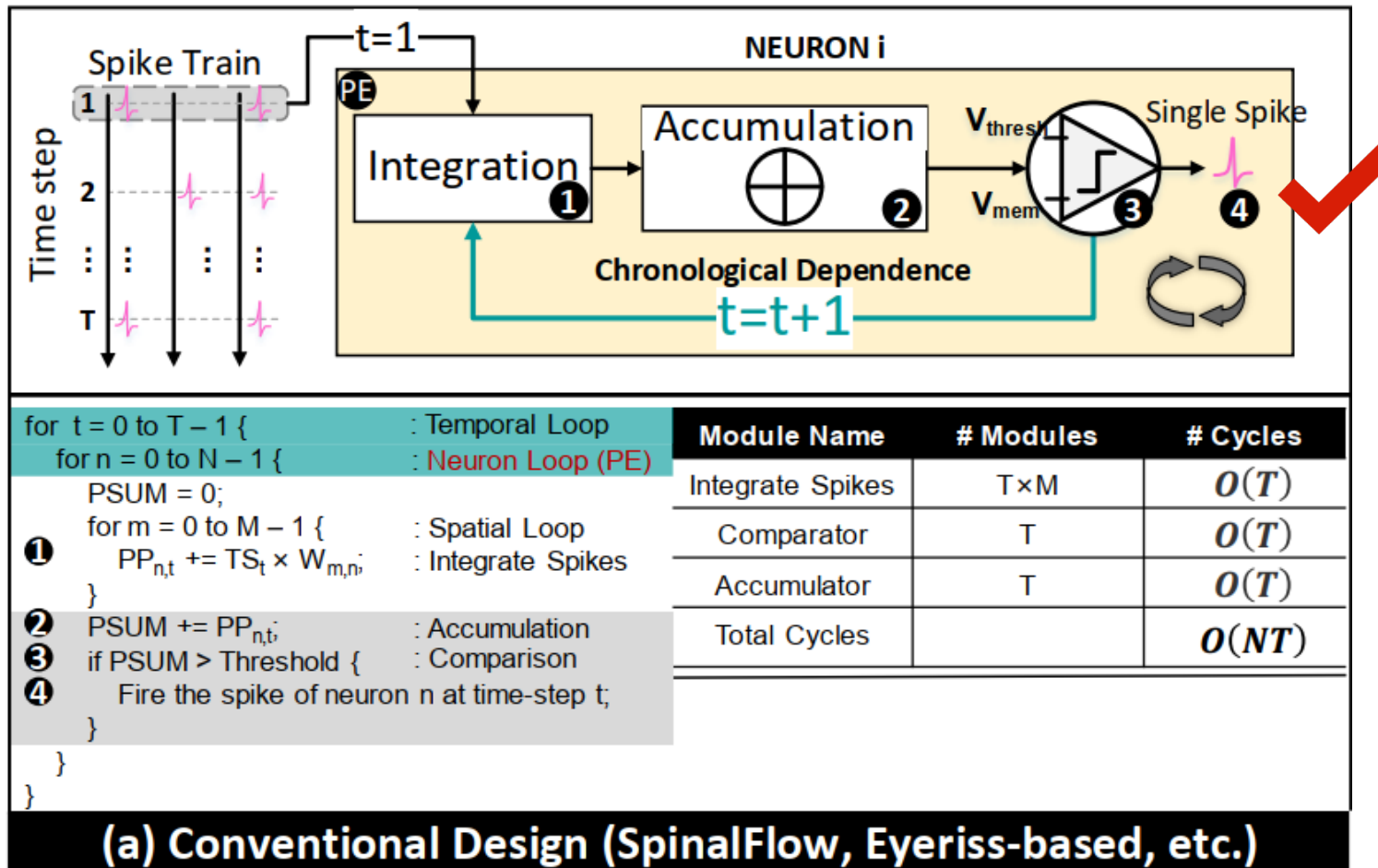


The accelerator processes the spikes as follows:

1. integrate the spikes at time step  $t$ ;
2. accumulate the integrated spike to the membrane potentials;
3. compare the current membrane potential to the prescribed firing threshold

Map the post-synaptic neurons calculations onto PEs in parallel and integrate spikes along the time steps in serial.

# Existing dataflow in SNN accelerators



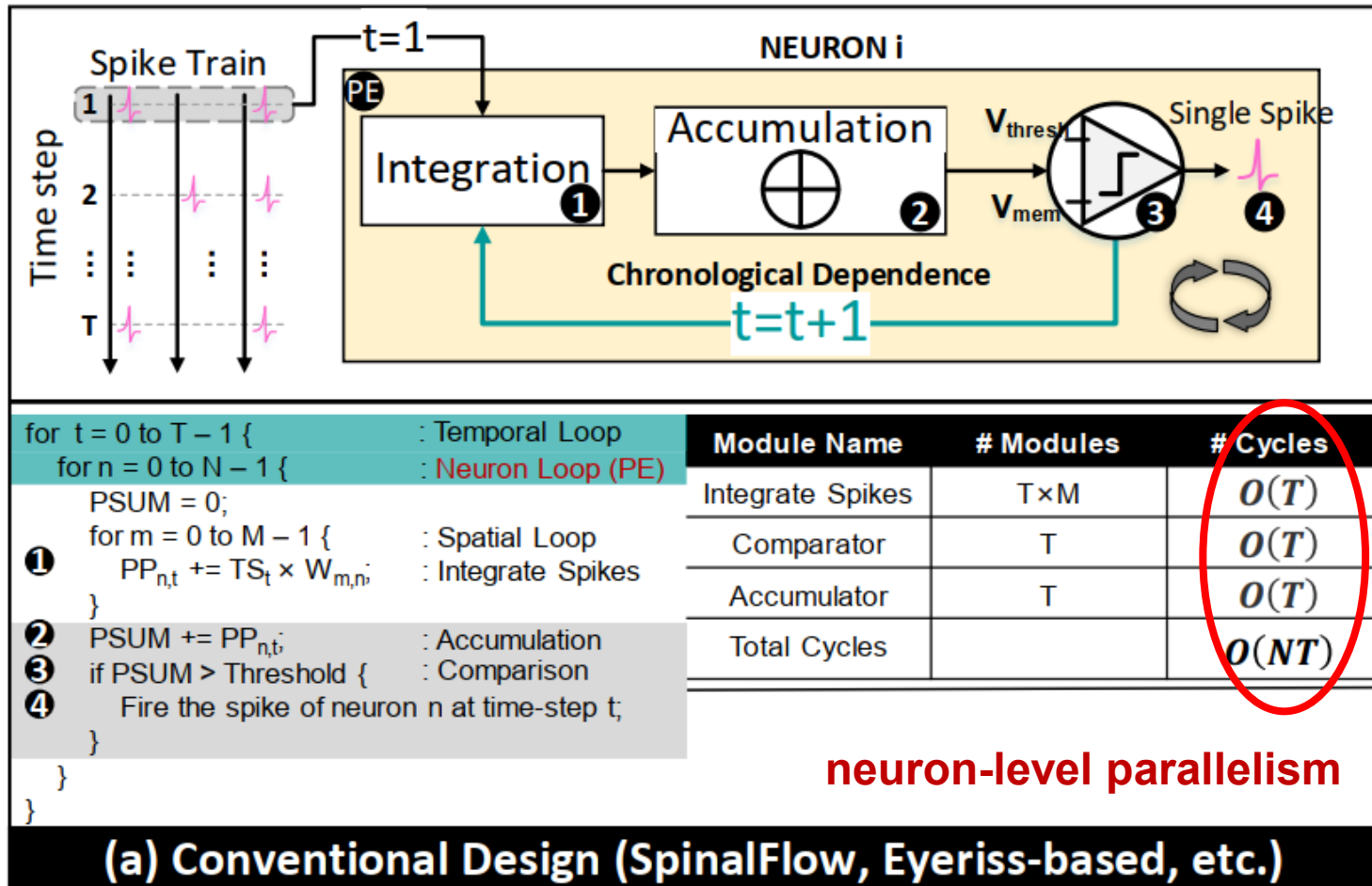
The accelerator processes the spikes as follows:

1. integrate the spikes at time step  $t$ ;
2. accumulate the integrated spike to the membrane potentials;
3. compare the current membrane potential to the prescribed firing threshold;
4. fire the output spike at time step  $t$  and reset the membrane potential if the potential exceeds the threshold.

Map the post-synaptic neurons calculations onto PEs in parallel and integrate spikes along the time steps in serial.



# Existing dataflow in SNN accelerators

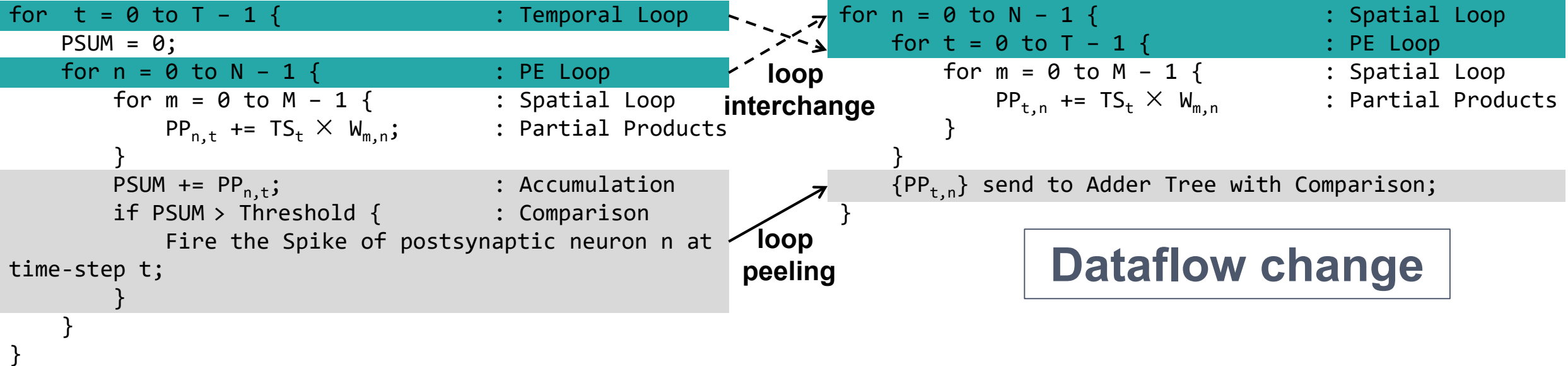


## Limitations

- The accumulation of membrane potential depends on the accumulated membrane potential of previous time steps, making SNNs require sequential computation across multiple time steps of the spike train.

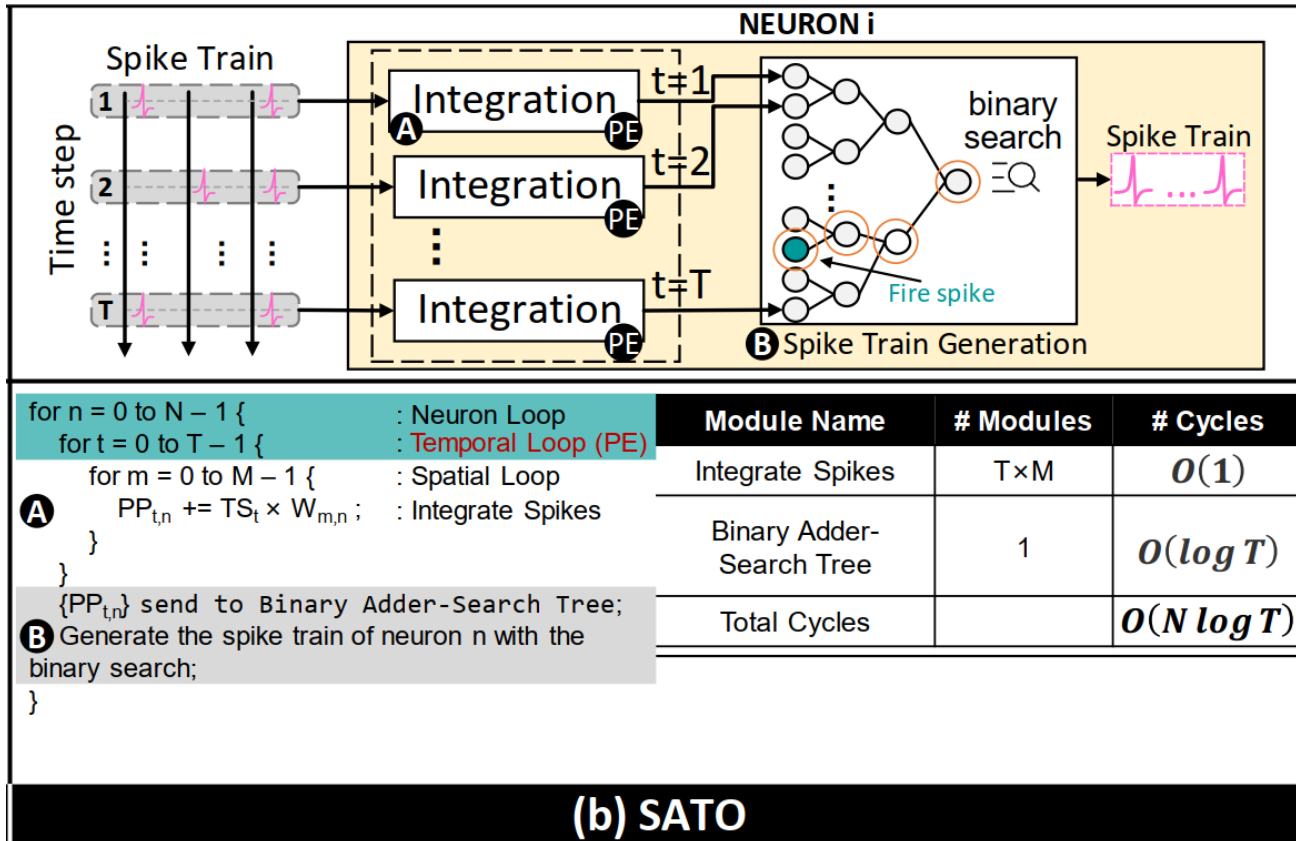
- The number of cycles required to perform an inference is **at least the number of time steps**.
- The parallelism of PE is limited by the number of neurons.

# Overview of Our SATO Dataflow



Decouple the chronological dependence and parallelize the integration of received spikes at each time step for boosting SNN efficiency.

# Overview of Our SATO Dataflow



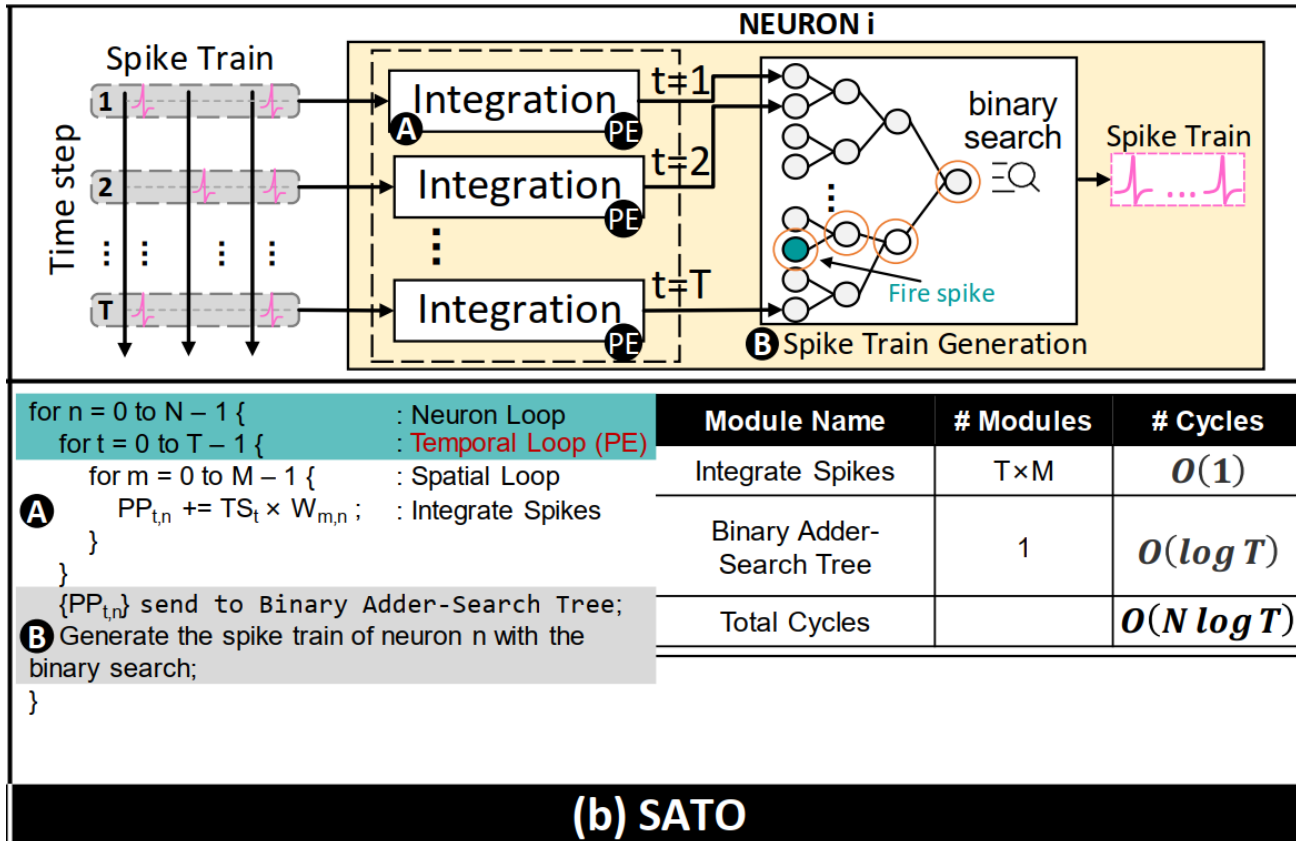
SATO processes the spikes as follows:

- A. map the integration of spikes of all time steps on PEs without accumulating membrane potential

Expand the neuron-level parallelism to additional temporal-level parallelism.



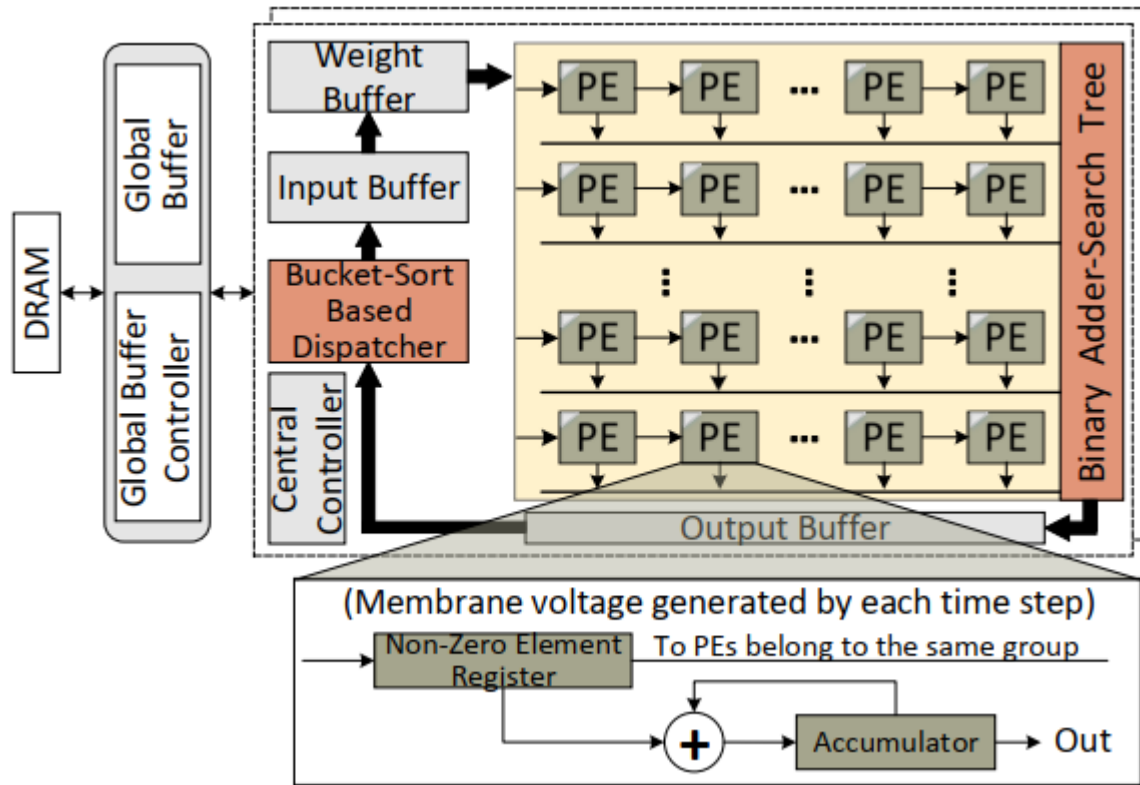
# Overview of Our SATO Dataflow



SATO processes the spikes as follows:

- B. perform spike train generation in a PE array, we orderly feed the integration results located at each time step to the adder tree and combine them with the binary search to determine the time step that the first fired spike.

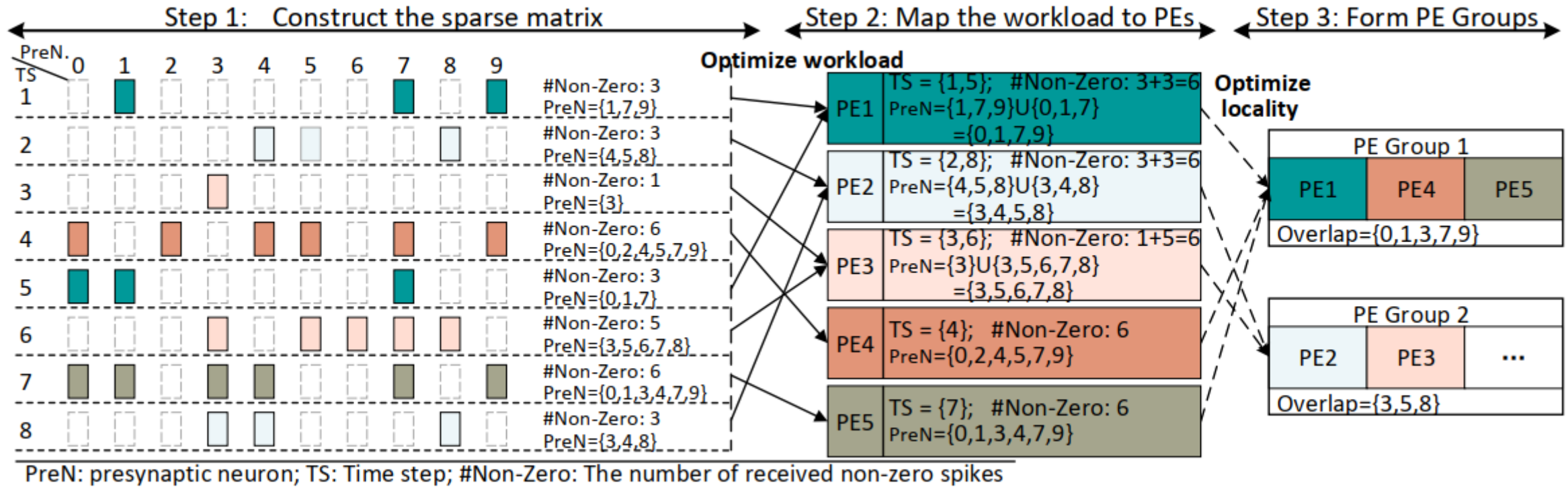
# Overview of Our SATO Architecture



- Performs the integration operation of received spikes in parallel.
- Exploit both temporal-parallelism and neuron-level parallelism, increasing the scalability of the accelerator,
- Once PEs complete the integration of received spikes of each time step, the results are fed to a novel binary adder-search tree to generate the spike train.

In temporal-encoded SNN, the spike train only has a single spike, which can be realized by adopting binary search

# Overview of Our SATO Architecture

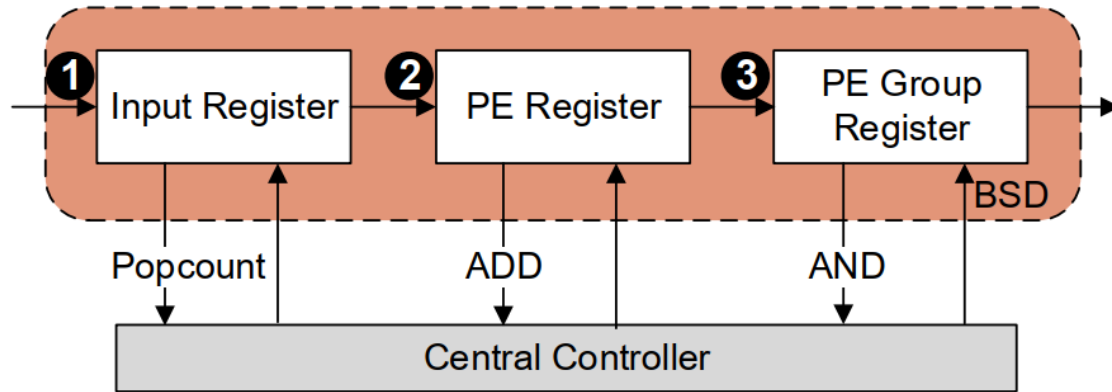


- Designs a workload dispatch strategy and exploits the inherent sparsity of the spike train and the data locality to optimize the workload and overhead of the PEs.

**Balancing workloads among PEs**




# Overview of Our SATO Architecture

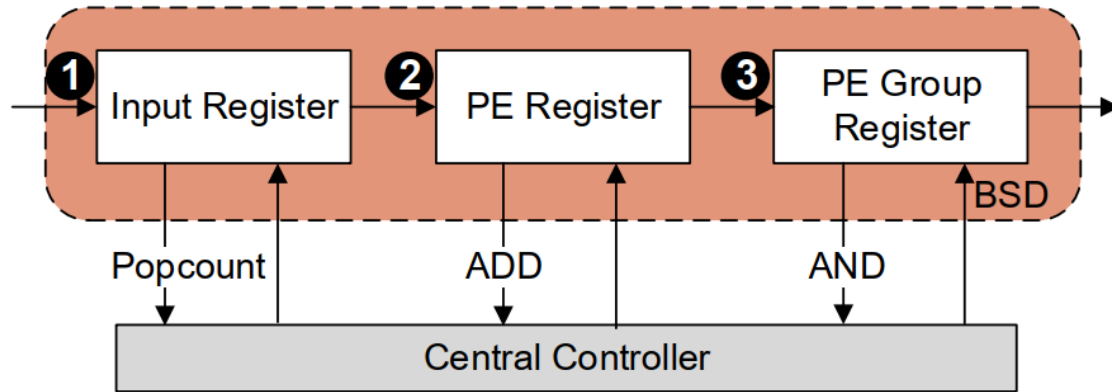


Bucket-Sort Based Dispatcher (BSD) is the key to balance the workload among PEs and maximize the data locality.

BSD design as follows:

-  Step 1: load the input by rows into the input register and conduct the count for each row (i.e., counting the `1` at each time steps)

# Overview of Our SATO Architecture

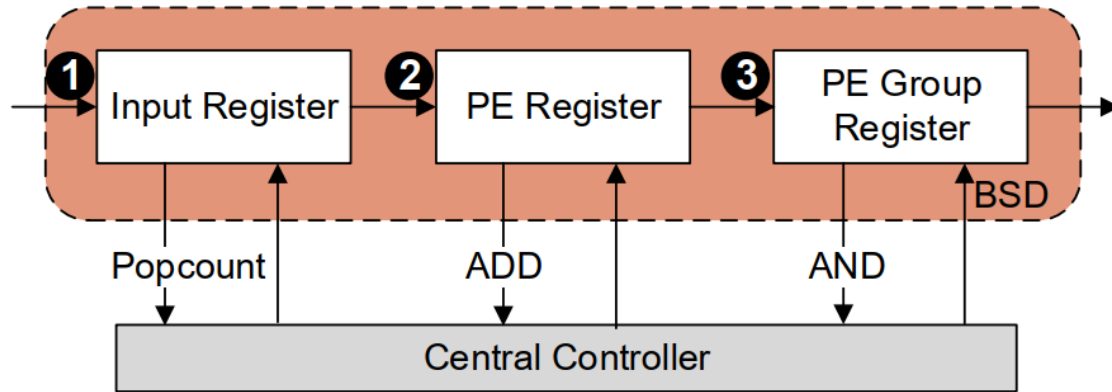


Bucket-Sort Based Dispatcher (BSD) is the key to balance the workload among PEs and maximize the data locality.

BSD design as follows:

- Step 2: map the workload to PE based on the number of spikes processed by PE and save results into the PE register

# Overview of Our SATO Architecture



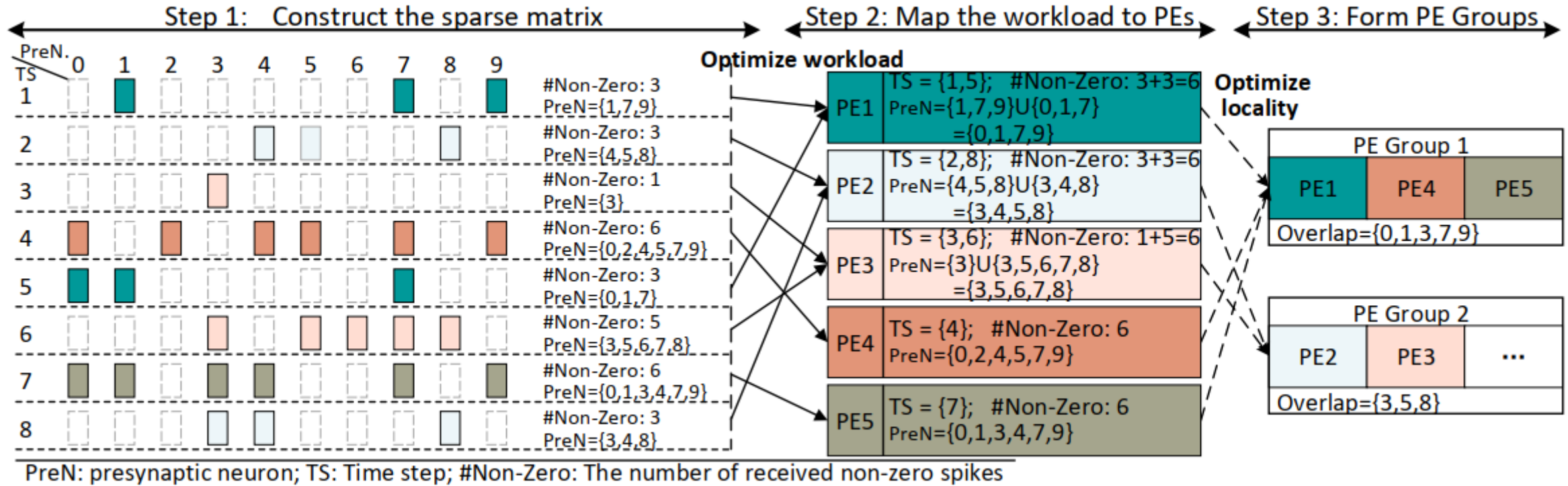
Bucket-Sort Based Dispatcher (BSD) is the key to balance the workload among PEs and maximize the data locality.

BSD design as follows:

- Step 3: calculate the number of extra spikes when inserting PE into the group with logic AND operation and generate a group PE table stored in the register

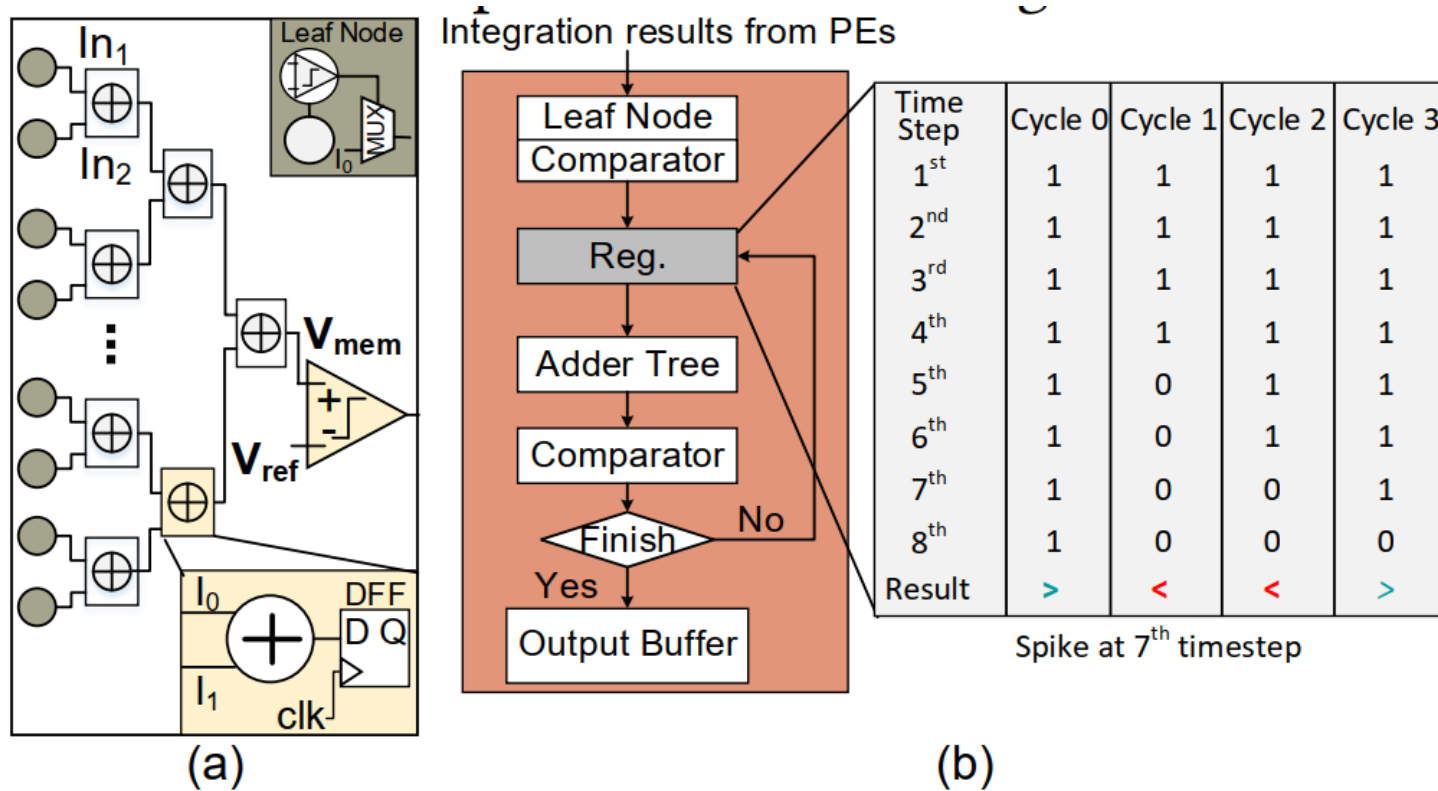


# Overview of Our SATO Architecture



- Step 3: calculate the number of extra spikes when inserting PE into the group with logic AND operation and generate a group PE table stored in the register

# Overview of Our SATO Architecture

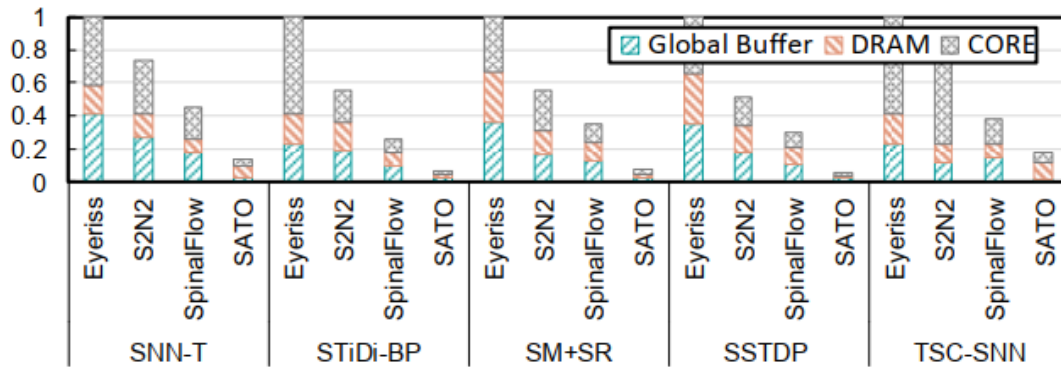


Binary adder-search tree is responsible for processing the results from the PE array and generating the spike train.

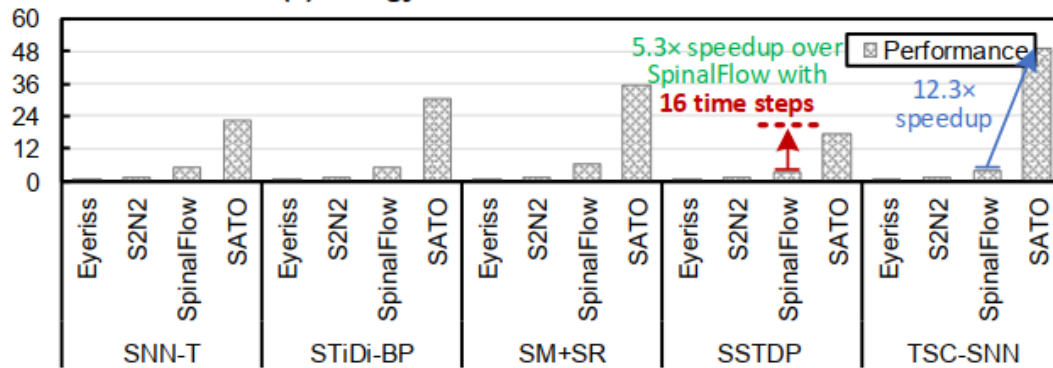


We can determine the position of a unique spike in the spike train by binary search.

# Experiment Results — Energy



(a) Energy breakdown of different SNNs



(b) Performance of different SNNs

Table 1: The characteristic of SNNs for verifying SATO.

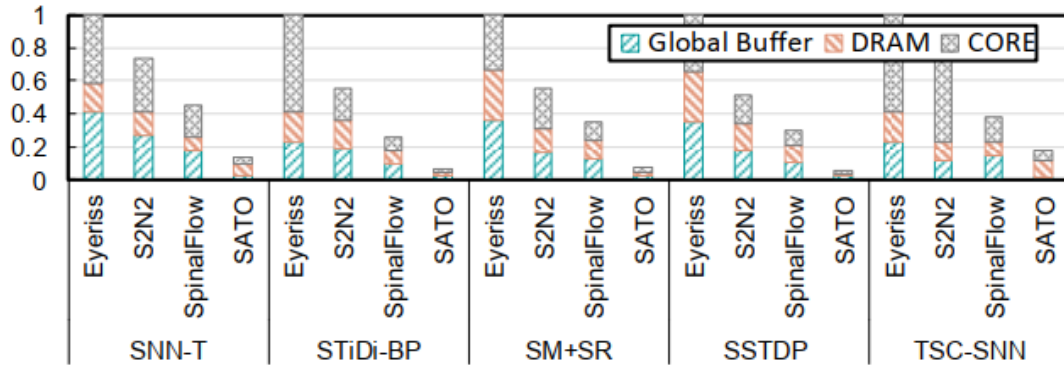
| Model         | Structure  | Coding   | Timestep | DataSet  | Acc.   |
|---------------|------------|----------|----------|----------|--------|
| SNN-T [4]     | 784-340-10 | Temporal | 800      | MNIST    | 97.9%  |
| STiDi-BP [12] | 784-500-10 | Temporal | 600      | MNIST    | 97.4%  |
| SM+SR [15]    | VGG-7      | Temporal | 544      | CIFAR-10 | 91.05% |
| SSTDP [11]    | VGG-7      | Temporal | 16       | CIFAR-10 | 91.31% |
| TSC-SNN [8]   | VGG-16     | Temporal | 2480     | ImageNet | 69.96% |



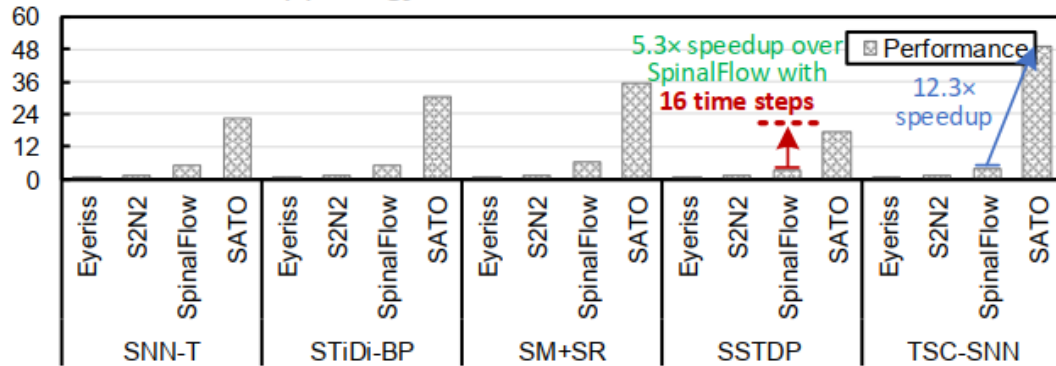
Compared to Eyeriss, S2N2 and SpinalFlow, SATO consumes 91.3%, 83.4% and 69.7% less energy, respectively.



# Experiment Results — Performance



(a) Energy breakdown of different SNNs



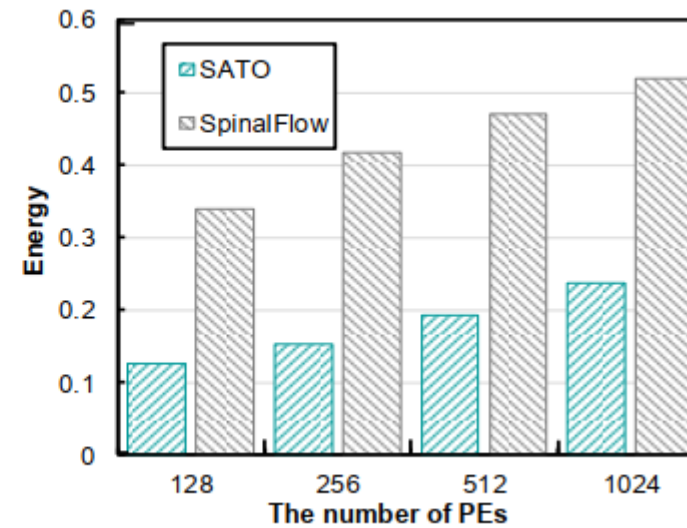
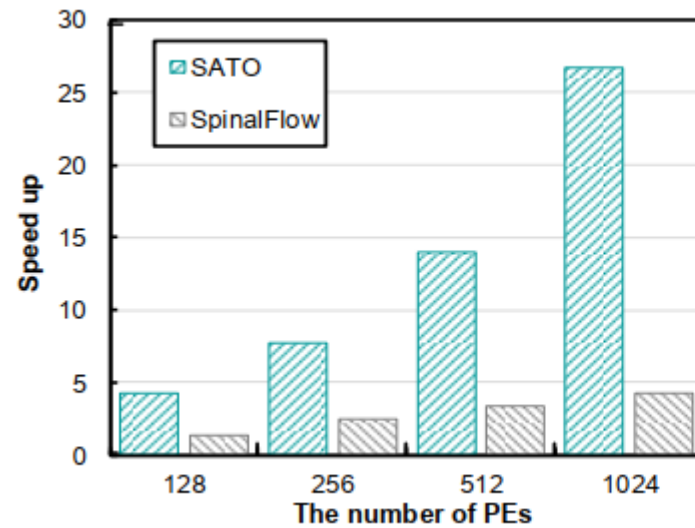
(b) Performance of different SNNs

**Table 1: The characteristic of SNNs for verifying SATO.**

| Model         | Structure  | Coding   | Timestep | DataSet  | Acc.   |
|---------------|------------|----------|----------|----------|--------|
| SNN-T [4]     | 784-340-10 | Temporal | 800      | MNIST    | 97.9%  |
| STiDi-BP [12] | 784-500-10 | Temporal | 600      | MNIST    | 97.4%  |
| SM+SR [15]    | VGG-7      | Temporal | 544      | CIFAR-10 | 91.05% |
| SSTDP [11]    | VGG-7      | Temporal | 16       | CIFAR-10 | 91.31% |
| TSC-SNN [8]   | VGG-16     | Temporal | 2480     | ImageNet | 69.96% |

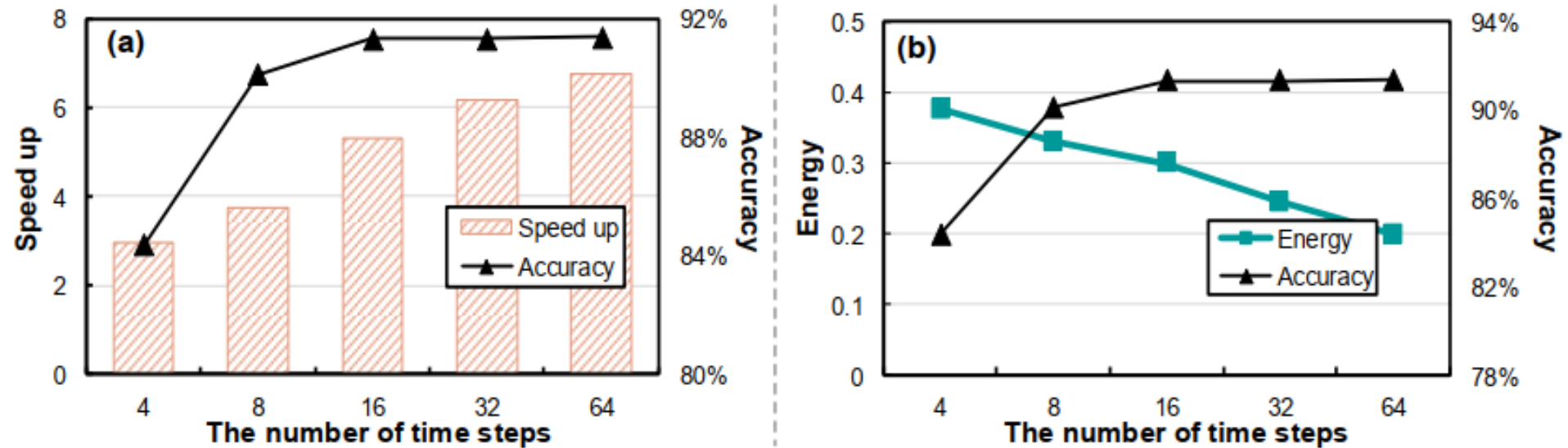
- Compared to Eyeriss, S2N2 and SpinalFlow, SATO consumes 91.3%, 83.4% and 69.7% less energy, respectively.
- Compared with Eyeriss, S2N2, and SpinalFlow, SATO achieves an average 30.9 $\times$ , 22.1 $\times$ , and 6.4 $\times$  performance improvement

# Experiment Results — PEs



- ⊙ The number of time steps is generally much greater than 128, so the number of PEs is related to performance gains and energy consumption.
- ⊙ As the number of PEs increases, we calculate more time steps in parallel to obtain more performance benefits.

# Experiment Results — time steps



- As the number of time steps increases, the accuracy of the SNN increases, and the performance gains of SATO are also improving.
- For the SNN with 16 time steps, compared to SpinaFlow, SATO consumes nearly 70% less energy.



# Conclusion

- ① A novel redesign of the SNN dataflow
  - Decouple the chronological dependence
  - Parallelize the integration of received spikes at each time step
- ① An efficient SNN architecture
  - workload allocation strategy
  - Bucket-Sort Based Dispatcher
  - Binary adder-search tree
- ① Keep high energy efficiency while gaining large performance improvement

# Thank you !

## SATO: Spiking Neural Network Acceleration via Temporal-Oriented Dataflow and Architecture

Fangxin Liu (Speaker)

Wenbo Zhao, Zongwu Wang, Yongbiao Chen, Tao Yang, Zhezhi He, Xiaokang Yang and **Li Jiang\***  
Shanghai Jiao Tong University

