

# Accelerating Neural Models with FPGAs and GPUs

Wayne Luk  
Imperial College London

BIECS'09

# Outline

1. Why FPGA and GPU?
2. Modelling neural network
3. FPGA design
4. GPU design
5. Extensions
6. Summary

Acknowledgement: Andreas Fidjeland, David Thomas, Murray Shanahan

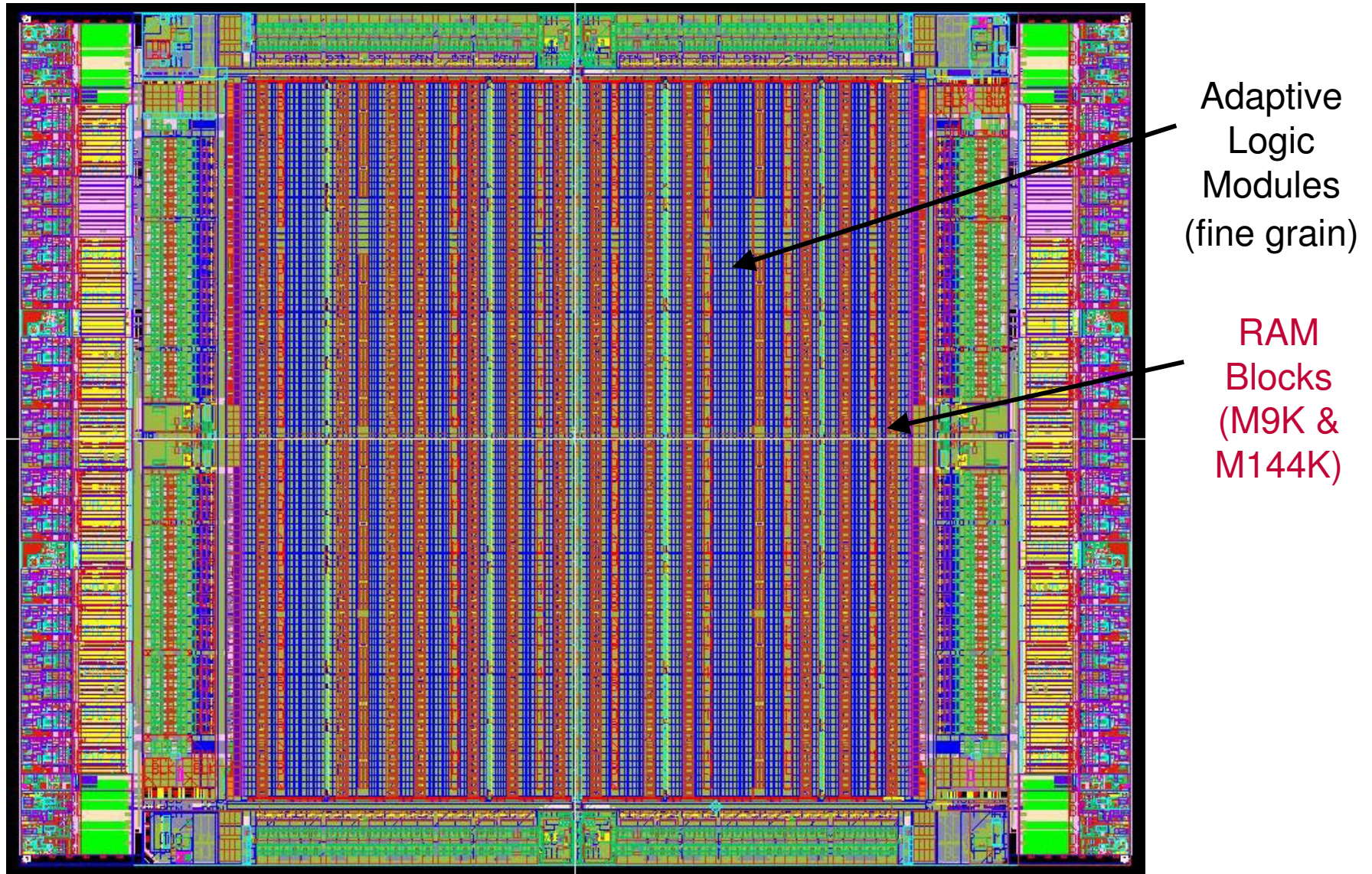
# 1. Why FPGA and GPU?

- Exploit latest off-the-shelf technologies
  - CPU: multi-core parallelism
  - GPU: single-instruction, multi-thread parallelism
  - FPGA: customisable hardware parallelism
- Benefits
  - no fabrication, latest commercial technologies
- Our approach: post-fabrication customisation
  - System architecture: compute + connect + memory + ...
  - Application mapping: neural models
  - Modularity: retarget to different / future technologies

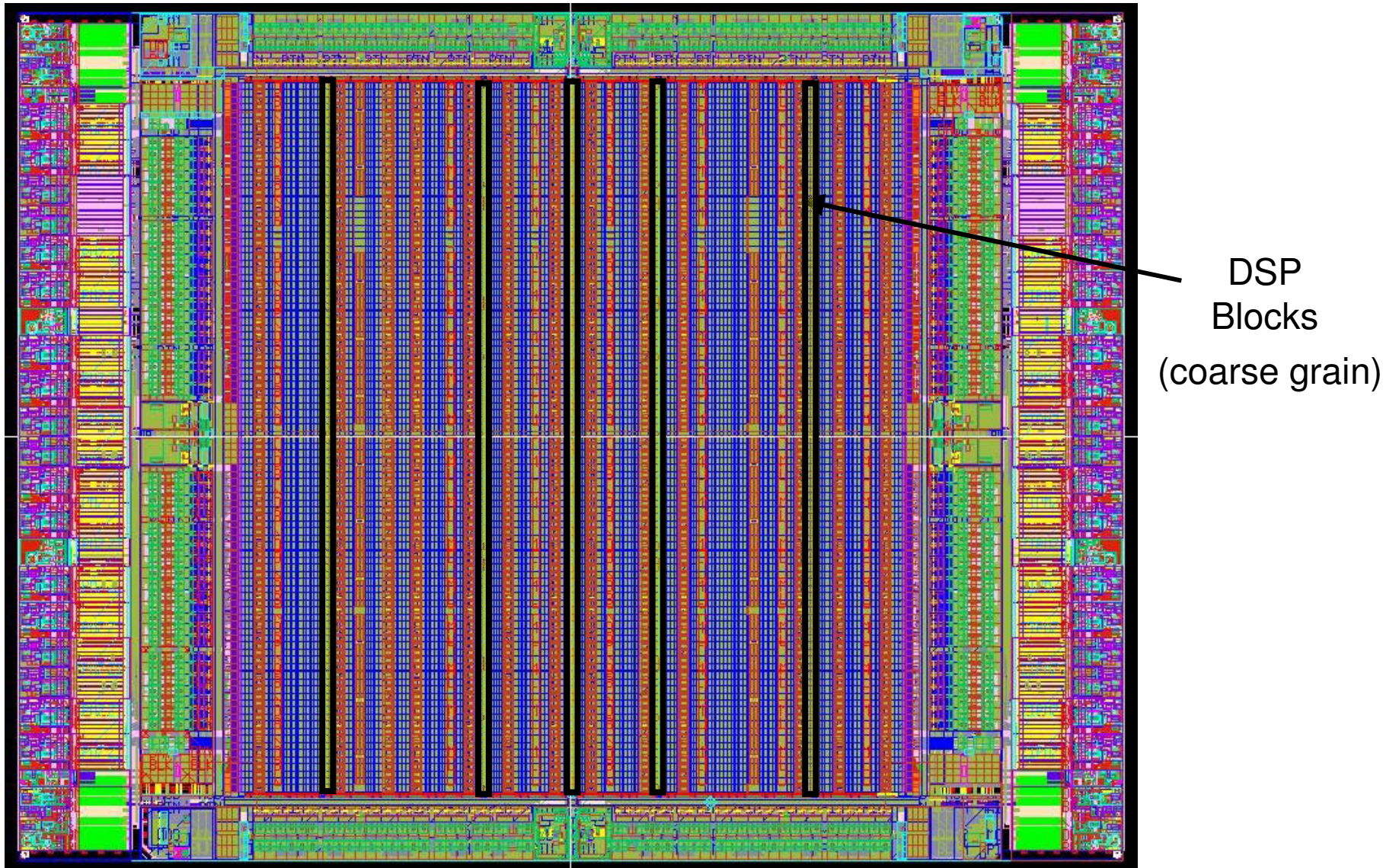
# FPGA Overview

- Field Programmable Gate Array
  - Hardware speed
  - Software flexibility
- Resources customisable by application builders
  - Computation resources: multiple granularities
  - Inter-connections
  - Memories
  - Clocking
  - External interfaces
- Much used in prototyping; growing consumer apps

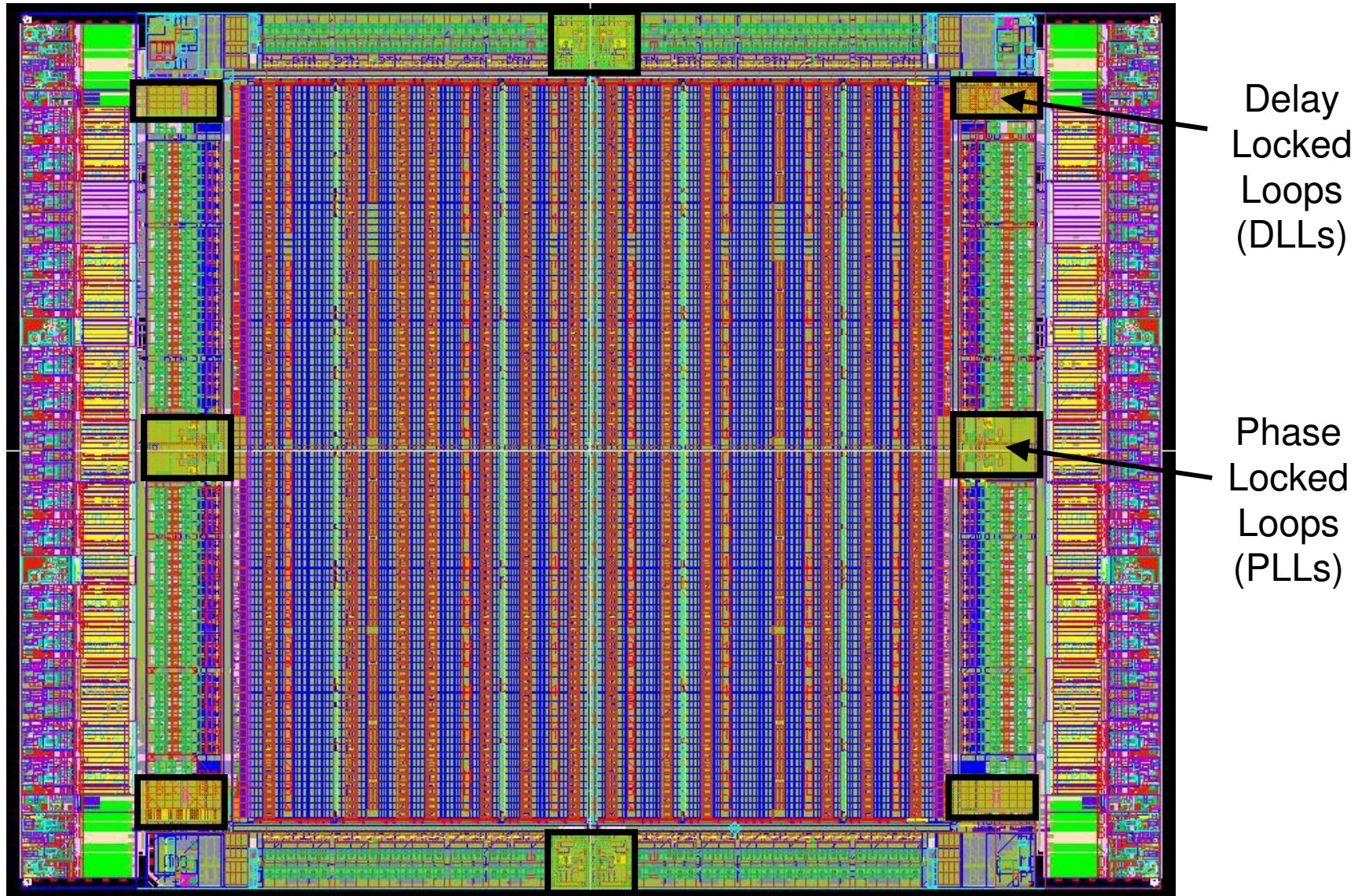
# Stratix IVGX 230 (Mid-Size Device)



# Digital Signal Processing



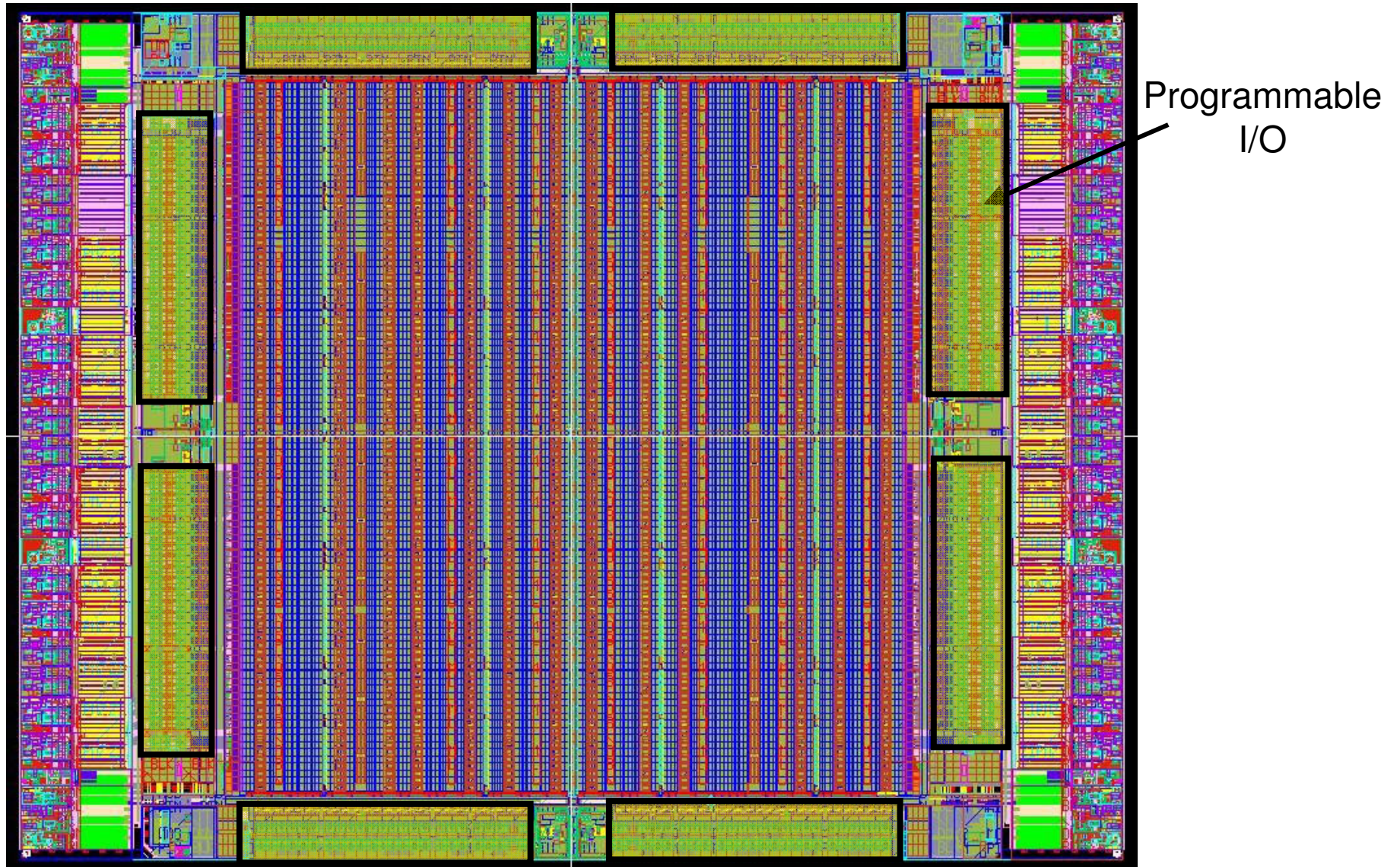
# Clock Management



Delay  
Locked  
Loops  
(DLLs)

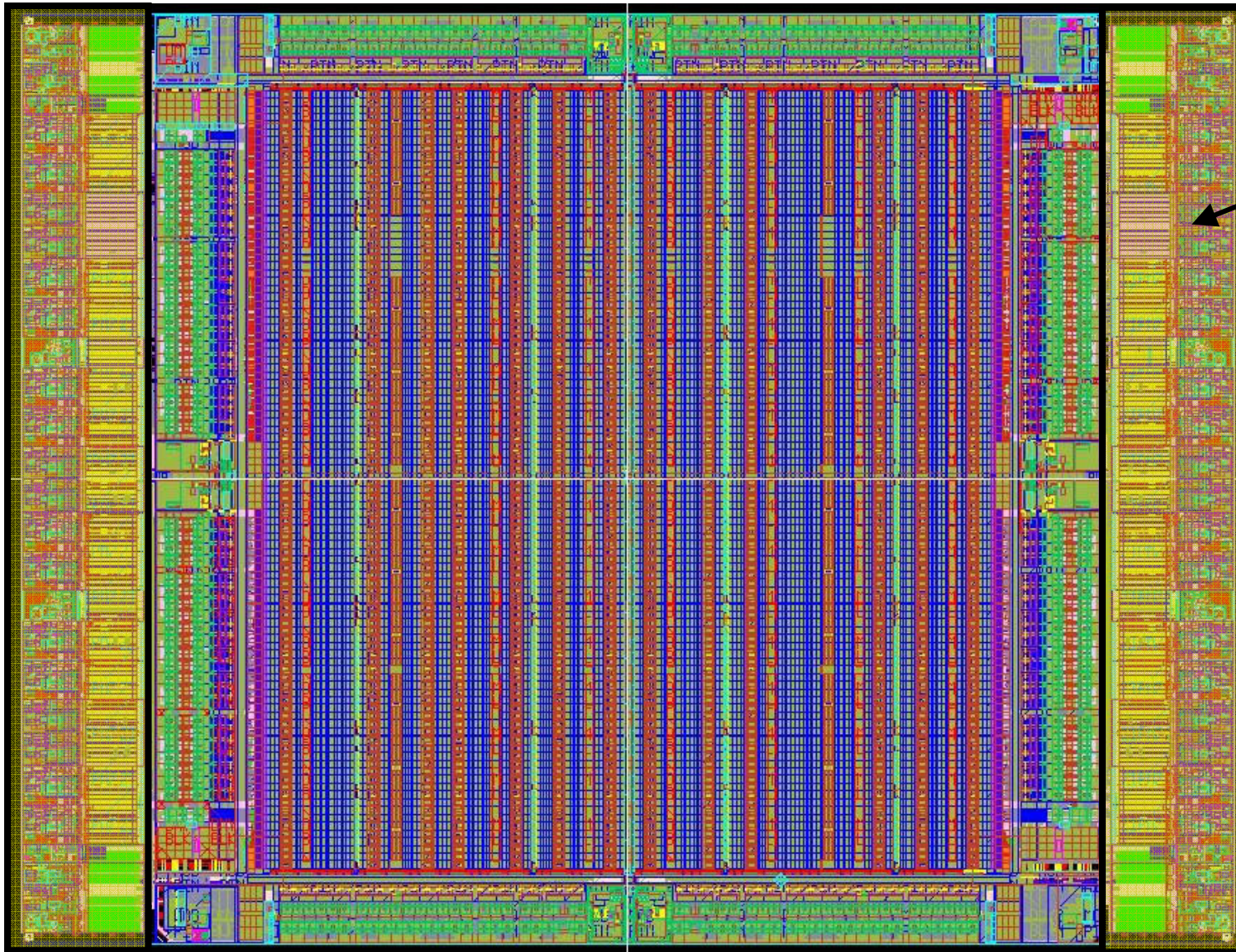
Phase  
Locked  
Loops  
(PLLs)

# General I/O



(from V. Betz, Altera)

# Serial Interfaces

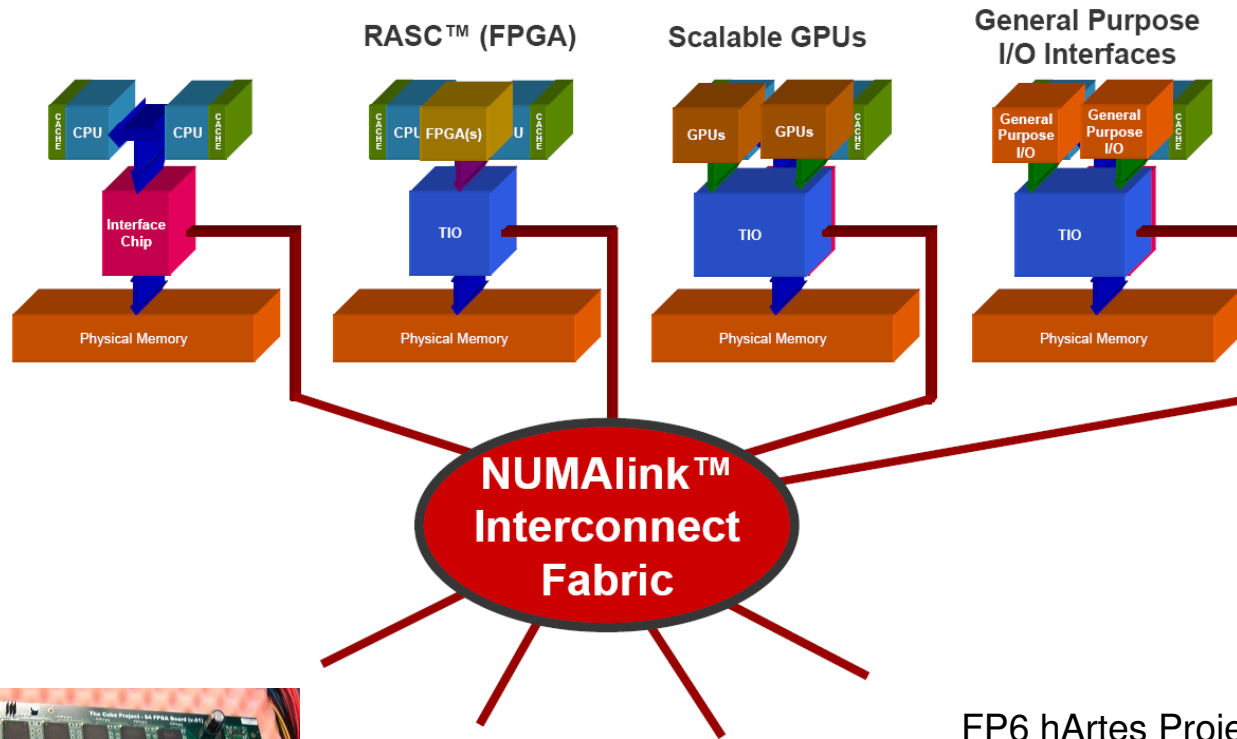


High Speed Serial Interfaces:  
eg connect multiple FPGAs

# Stratix IV Overview

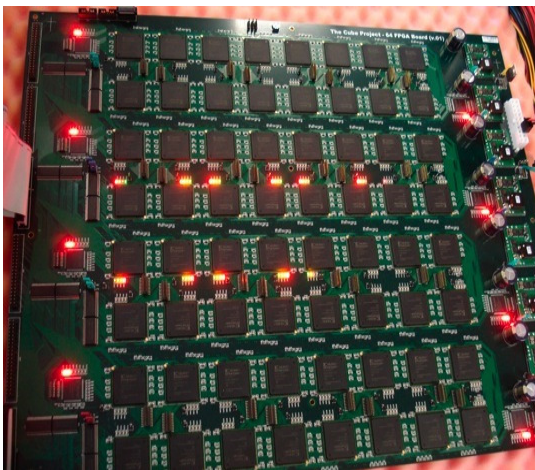
Feature	Stratix III (65 nm)	Stratix IV (40 nm)
Logic Elements	340k	680k
RAM bits	16 Mb + 4 Mb	33 Mb + 8.5 Mb
18x18 multipliers	768	1360
General I/O	1104	1104
High-speed serial links	0	48 transmit + 48 receive @ 11.3 Gb/s
Hard PCIe blocks	0	4
Clock generation	12 PLL(x10)	12 PLL(x10) + 32 serial recovered + 24 serial transmit
Clock distribution	16 Global + 88 Quadrant + 132 PCLK	16 Global + 88 Quadrant + 132 PCLK

# Heterogeneous Multicore Systems



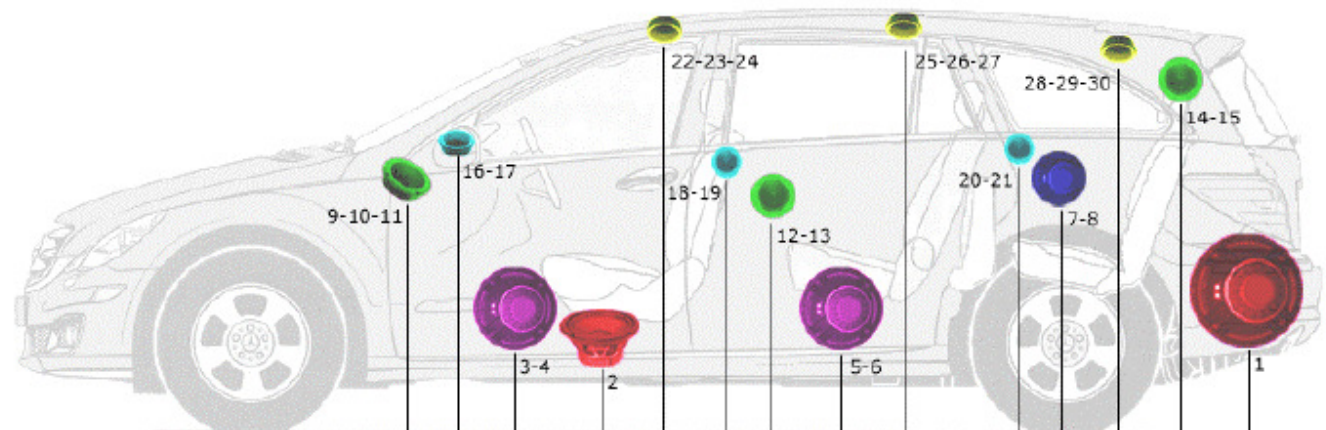
SGI Altix 4700  
Multi-Paradigm  
ccNuma

FPGA Cube

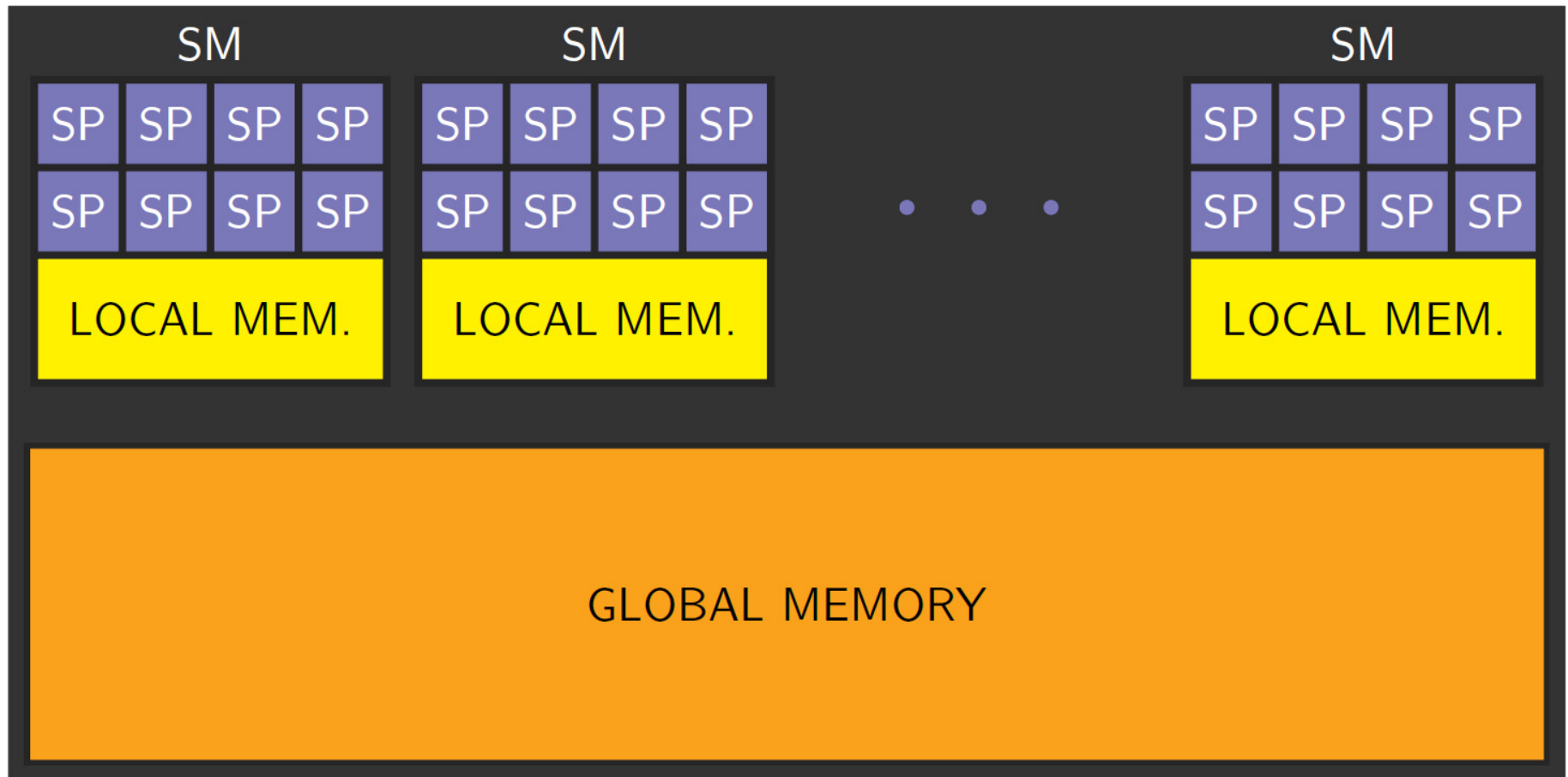


wl.11

FP6 hArtes Project: [hartes.org](http://hartes.org)



# GPU Overview

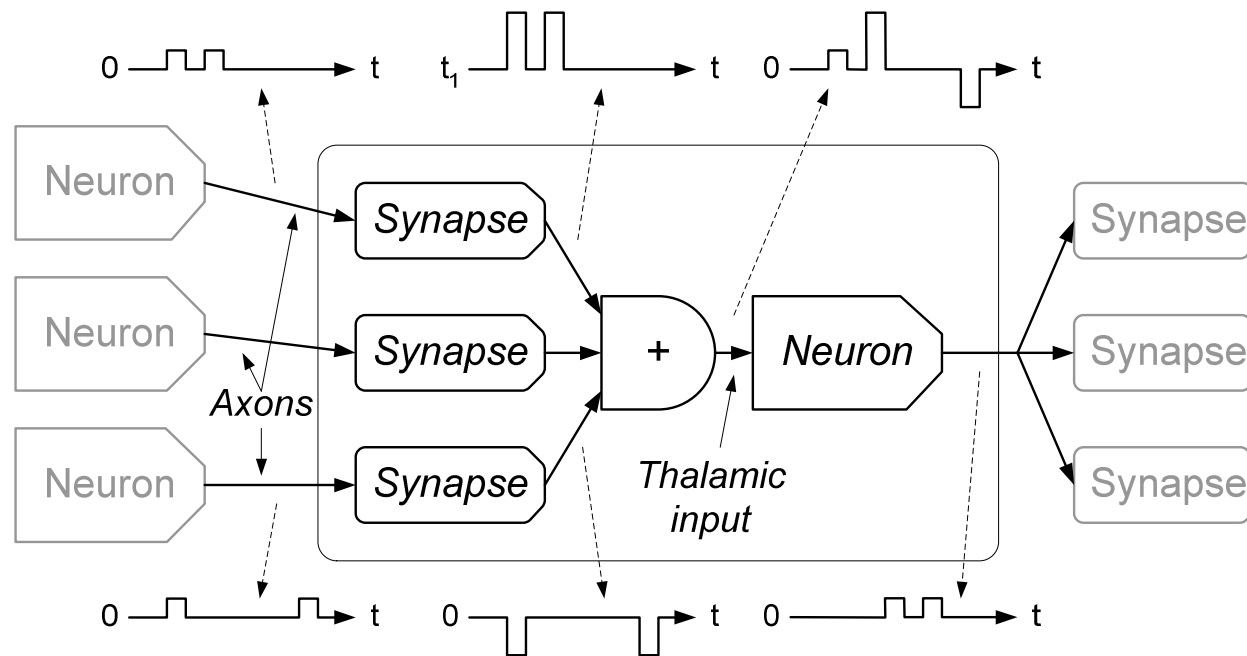


SP: Scalar Processor, SM: Streaming Multiprocessors

# GPU Features

- Increasingly adapted for general-purpose computing
- Parallel streaming multiprocessor
  - large number of scalar processors SP: 240 in C1060
  - SP groups into Streaming Multiprocessors SM
  - 2-level memory hierarchy
- Achieve high performance by
  - Parallel execution
  - Use local memory where possible
  - Exploit spatial locality of reference for global access

## 2. Modelling Spiking Neurons



- Spiking neural networks
  - Spikes: electrical binary pulses generated by neurons
  - Synapses: assign weight to spikes arriving at neuron
  - Neurons: state-ful; respond to and generate spikes

# Goals

- Not sure what features are important
  - Understand low-level components, but not high-level
  - No clue about global behaviour – start local
- Stochastic system: random noise at all levels
  - Identical networks starting in same state will diverge
  - Look for statistical similarities between different runs
- Goal: hyper real-time biologically plausible simulator
  - Small fully-connected networks: local phenomena
  - Run same network many times: statistical behaviour
  - Biologically plausible simulator: emulate real neurons

# Problem Statement

- Neural network structure:
  - $\mathbf{c} \in \mathbb{P}^n$  : vector of  $n$  neuron parameters
  - $\mathbf{W} \in \mathbb{R}^{n \times n}$  :  $n \times n$  matrix of synapse weights
  - $f : (\mathbb{P}, \mathbb{S}, \mathbb{R}) \rightarrow (\mathbb{S}, [0, 1])$  : neuron update function
- Dynamic state:
  - $\mathbf{s} \in \mathbb{S}^n$  : current state of neurons
  - $\mathbf{f} \in [0, 1]^n$  : did each neuron fire in previous time-step
- Advance network state one time-step
  - $\mathbf{i} \leftarrow \mathbf{W}\mathbf{x}\mathbf{f}$  : Calculate thalamic input vector  $\mathbf{i}$
  - $(\mathbf{s}, \mathbf{f}) \leftarrow f(\mathbf{c}, \mathbf{s}, \mathbf{i})$  : Update neuron state and firings

# Software Approach: fan-out

- Average firing rate is low
  - around 1-10% firings per time-step
  - optimise for average case
- Retain stimulus, not firings
  - accumulate into temporary vector  $\mathbf{t}$
  - set  $\mathbf{i}$  to  $\mathbf{t}$  after each time-step
- Each spike is fanned out
  - Vector add if neuron fires
    - Column  $\mathbf{W}[:,i]$  from synapse weights
  - No add if neuron doesn't fire
- Neuron updates can conflict
  - Vector add must be protected

```
 $\mathbf{t} = 0$   
for  $i=1..n$  do  
     $(\mathbf{s}_i, \mathbf{f}_i) = f(\mathbf{c}_i, \mathbf{s}_i, \mathbf{i}_i)$   
    if  $\mathbf{f}_i$  then  
         $\mathbf{t} = \mathbf{t} + \mathbf{W}[:,i]$   
    end if  
end for  
 $\mathbf{i} = \mathbf{t}$ 
```

# Hardware Approach: fan-in

- Use spatial parallelism
  - Target worst case
- Calculate thalamic input in one cycle
  - $n$  element pipelined dot-product
  - $[0,1]^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ 
    - Reduces to  $n-1$  additions
- Retain firings, not stimulus
  - Vector of booleans: much smaller
- Exposes parallelism
  - Fine-grain: dot-product
  - Coarse-grain: between iterations
  - One synch-point: final statement  $\mathbf{f}=\mathbf{t}$

```
t = 0
for  $i=1..n$  do
     $\mathbf{i}_i = \mathbf{W}[i,:] \times \mathbf{f}$ 
     $(\mathbf{s}_i, \mathbf{t}_i) = f(\mathbf{c}_i, \mathbf{s}_i, \mathbf{i}_i)$ 
end for
f = t
```

# 3. FPGA design

- FPGA implementation of spiking neural network
  - Biologically plausible: same behaviour as a brain
  - Hyper real-time: runs 100 times faster than “wet” network
- Algorithm optimised for FPGA resources
  - Synapses distributed across block-RAMs
  - Pipelined neuron update core: 1 neuron per cycle
- 2.26 GFlop/s double-precision (sustained)
  - 16x times faster than 3GHz Core-2 CPU
  - 1.1x times *single-precision* GPU

# Stage one: calculate thalamic input

$$\begin{bmatrix} - \\ - \\ - \\ - \\ - \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -3 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -2 \\ 0 & 1 & 1 & -2 & 1 \\ -1 & 2 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

**i**                      **w**                      **f**  
Thalamic                      Synapse                      Neurons that fired  
input                      weights                      last time-step

# Stage one: calculate thalamic input

$$\begin{bmatrix} -4 \\ - \\ - \\ - \\ - \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -3 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -2 \\ 0 & 1 & 1 & -2 & 1 \\ -1 & 2 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

**i**  
Thalamic input

**w**  
Synapse weights

**f**  
Neurons that fired last time-step

# Stage one: calculate thalamic input

$$\begin{bmatrix} -4 \\ 2 \\ - \\ - \\ - \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -3 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -2 \\ 0 & 1 & 1 & -2 & 1 \\ -1 & 2 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

**i**                      **w**                      **f**  
Thalamic                      Synapse                      Neurons that fired  
input                      weights                      last time-step

# Stage one: calculate thalamic input

$$\begin{bmatrix} -4 \\ 2 \\ 2 \\ - \\ - \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -3 & 1 \\ 0 & 2 & 1 & 0 & -1 \\ 1 & 1 & -1 & 1 & -2 \\ 0 & 1 & 1 & -2 & 1 \\ -1 & 2 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

**i**                      **w**                      **f**  
Thalamic input              Synapse weights              Neurons that fired last time-step

# Stage two: update neurons

$$\left( \begin{array}{c} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{array}, \begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \right) = f \left( \begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array}, \begin{array}{c} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{array}, \begin{array}{c} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{array} \right)$$

**s**                      **f**                      **c**                      **s**                      **i**

Neuron state      Fired neurons                      Neuron constants      Neuron state      Thalamic input

# Stage two: update neurons

$$\left( \begin{array}{c} \boxed{s_1} \\ - \\ - \\ - \\ - \end{array}, \begin{array}{c} \boxed{-4} \\ - \\ - \\ - \\ - \end{array} \right) = f \left( \begin{array}{c} \boxed{c_1} \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{array}, \begin{array}{c} \boxed{s_1} \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{array}, \begin{array}{c} \boxed{-4} \\ 2 \\ 2 \\ 1 \\ 2 \end{array} \right)$$

**s**                      **f**                      **c**                      **s**                      **i**

Neuron state      Fired neurons                      Neuron constants      Neuron state                      Thalamic input

# Stage two: update neurons

$$\left( \begin{array}{c} s_1 \\ \boxed{s_2} \\ - \\ - \\ - \end{array}, \begin{array}{c} -4 \\ \boxed{2} \\ - \\ - \\ - \end{array} \right) = f \left( \begin{array}{c} c_1 \\ \boxed{c_2} \\ c_3 \\ c_4 \\ c_5 \end{array}, \begin{array}{c} s_1 \\ \boxed{s_2} \\ s_3 \\ s_4 \\ s_5 \end{array}, \begin{array}{c} -4 \\ \boxed{2} \\ 2 \\ 1 \\ 2 \end{array} \right)$$

**s**
**f**
**c**
**s**
**i**

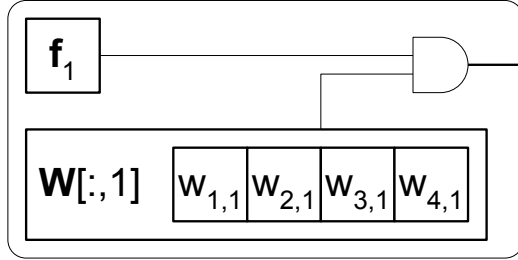
Neuron state
Fired neurons
Neuron constants
Neuron state
Thalamic input

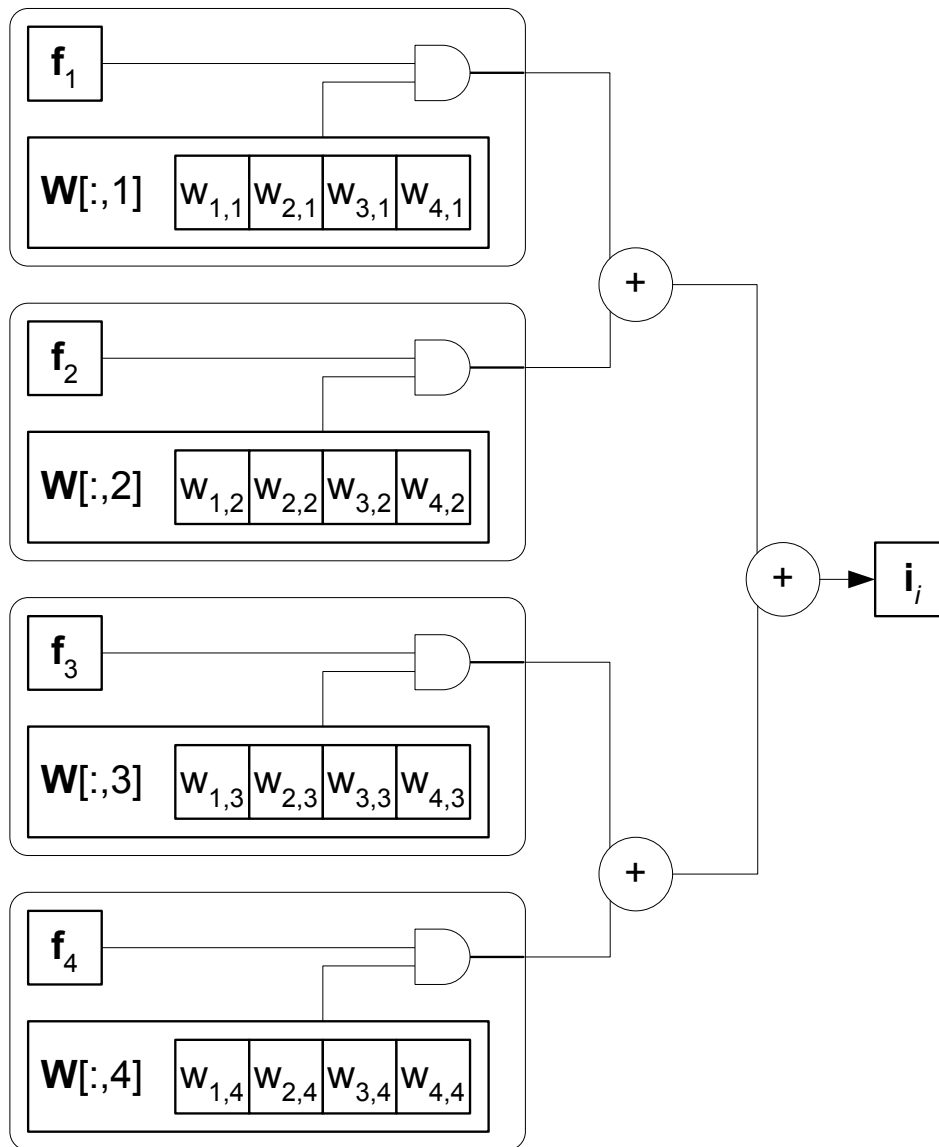
# Stage two: update neurons

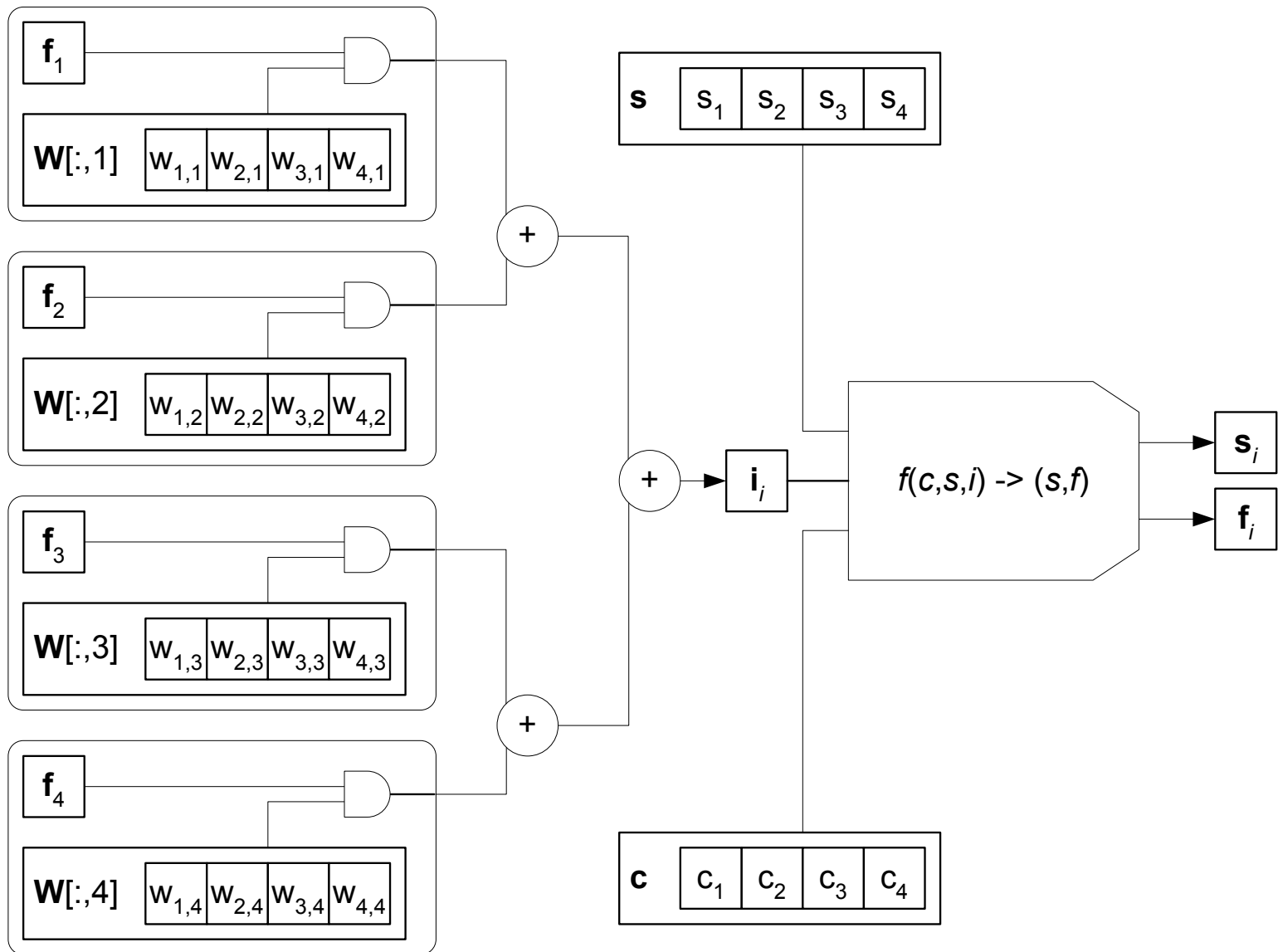
$$\left( \begin{array}{c} s_1 \\ s_2 \\ \boxed{s_3} \\ - \\ - \end{array} , \begin{array}{c} -4 \\ 2 \\ \boxed{2} \\ - \\ - \end{array} \right) = f \left( \begin{array}{c} c_1 \\ c_2 \\ \boxed{c_3} \\ c_4 \\ c_5 \end{array} , \begin{array}{c} s_1 \\ s_2 \\ \boxed{s_3} \\ s_4 \\ s_5 \end{array} , \begin{array}{c} -4 \\ 2 \\ \boxed{2} \\ 1 \\ 2 \end{array} \right)$$

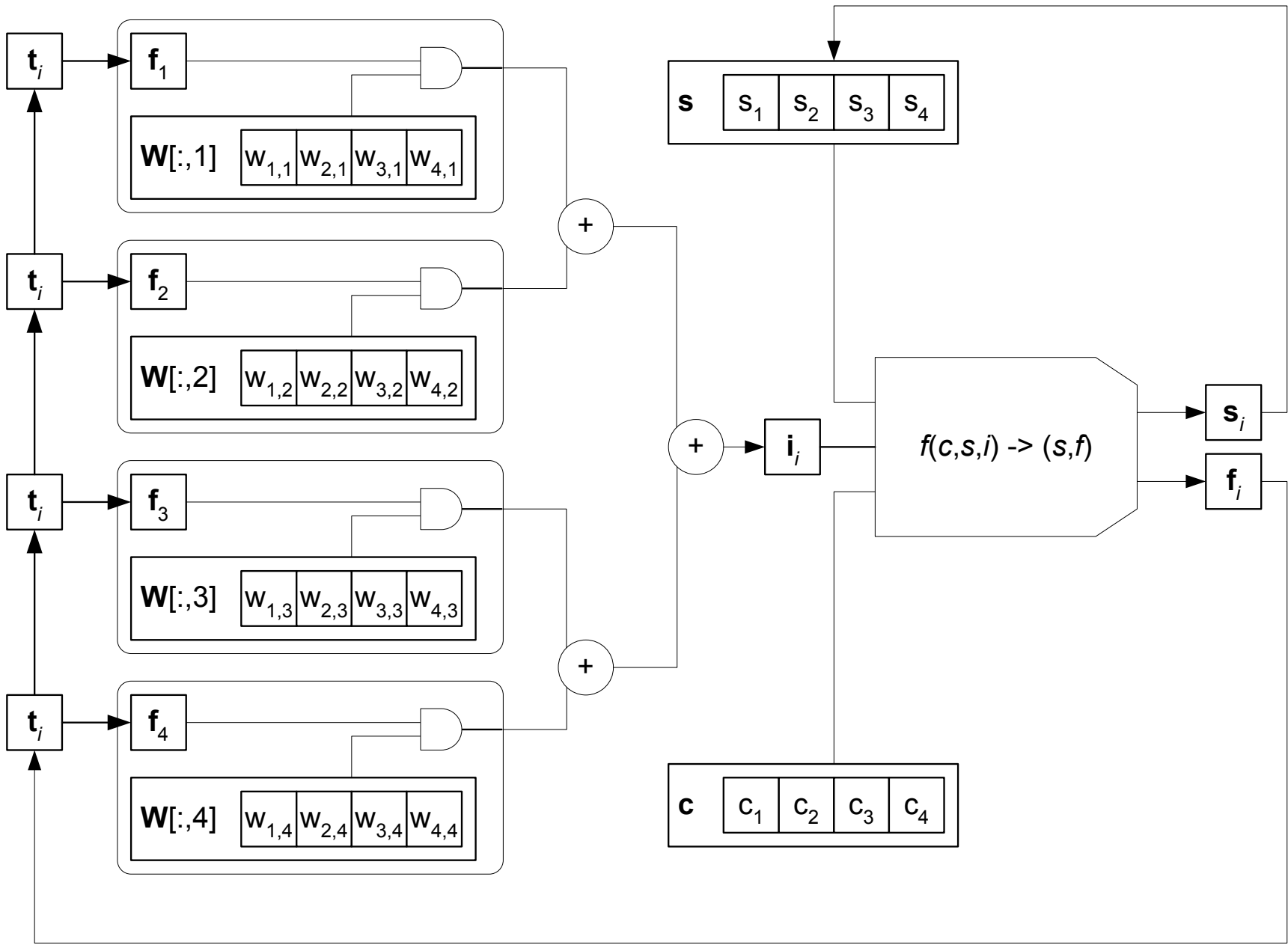
**s**                      **f**                      **c**                      **s**                      **i**

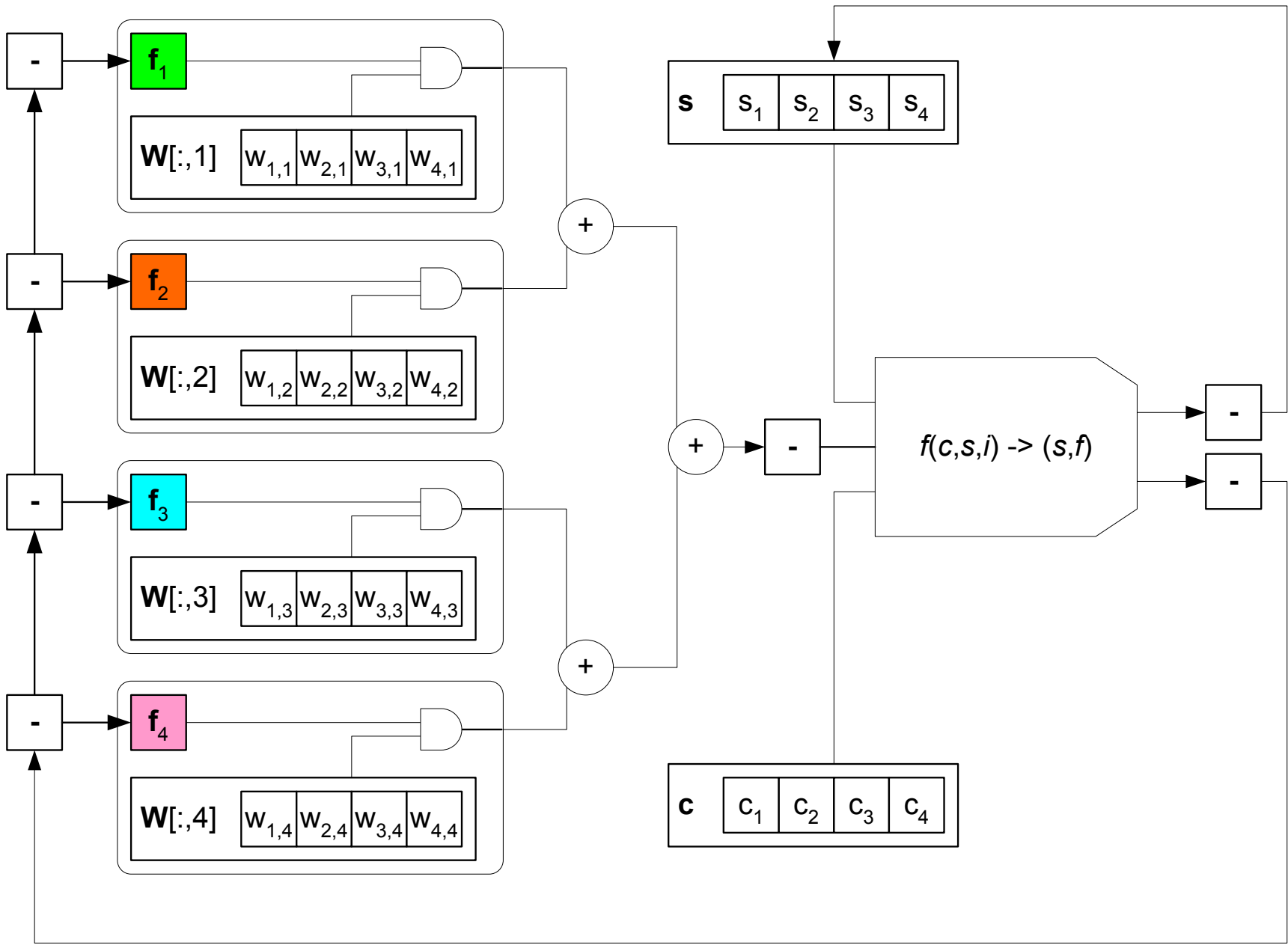
Neuron state      Fired neurons                      Neuron constants      Neuron state                      Thalamic input

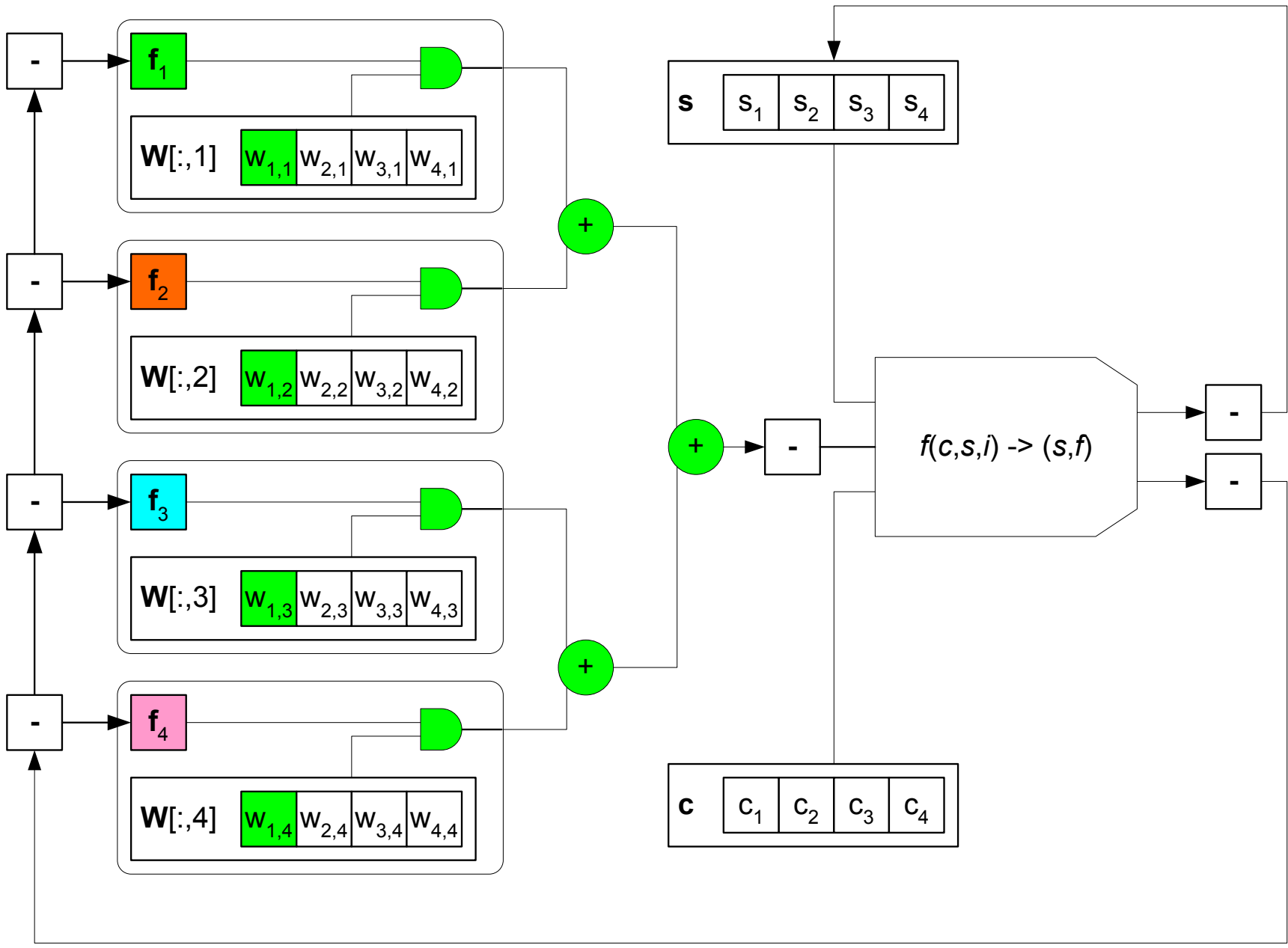


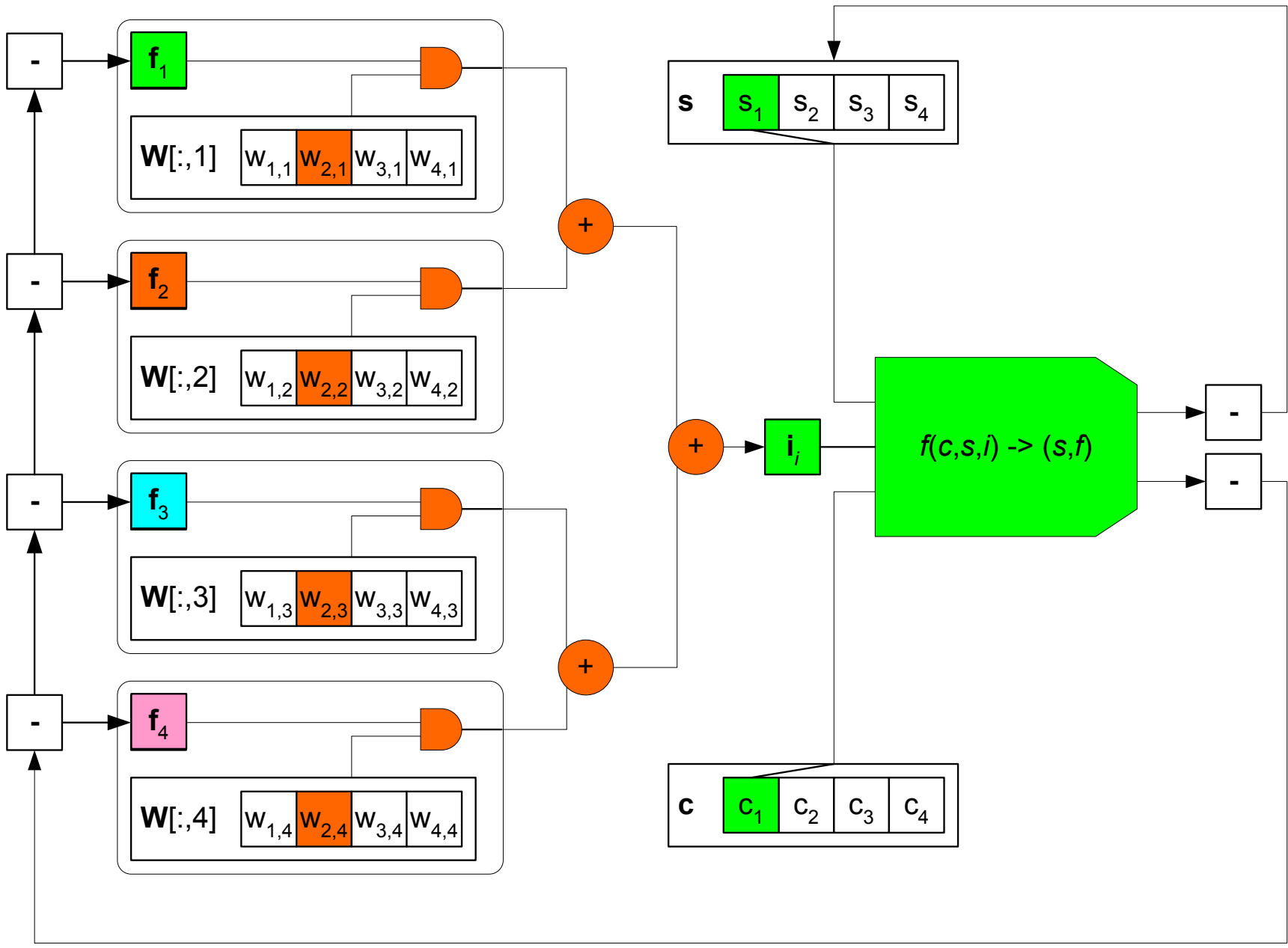


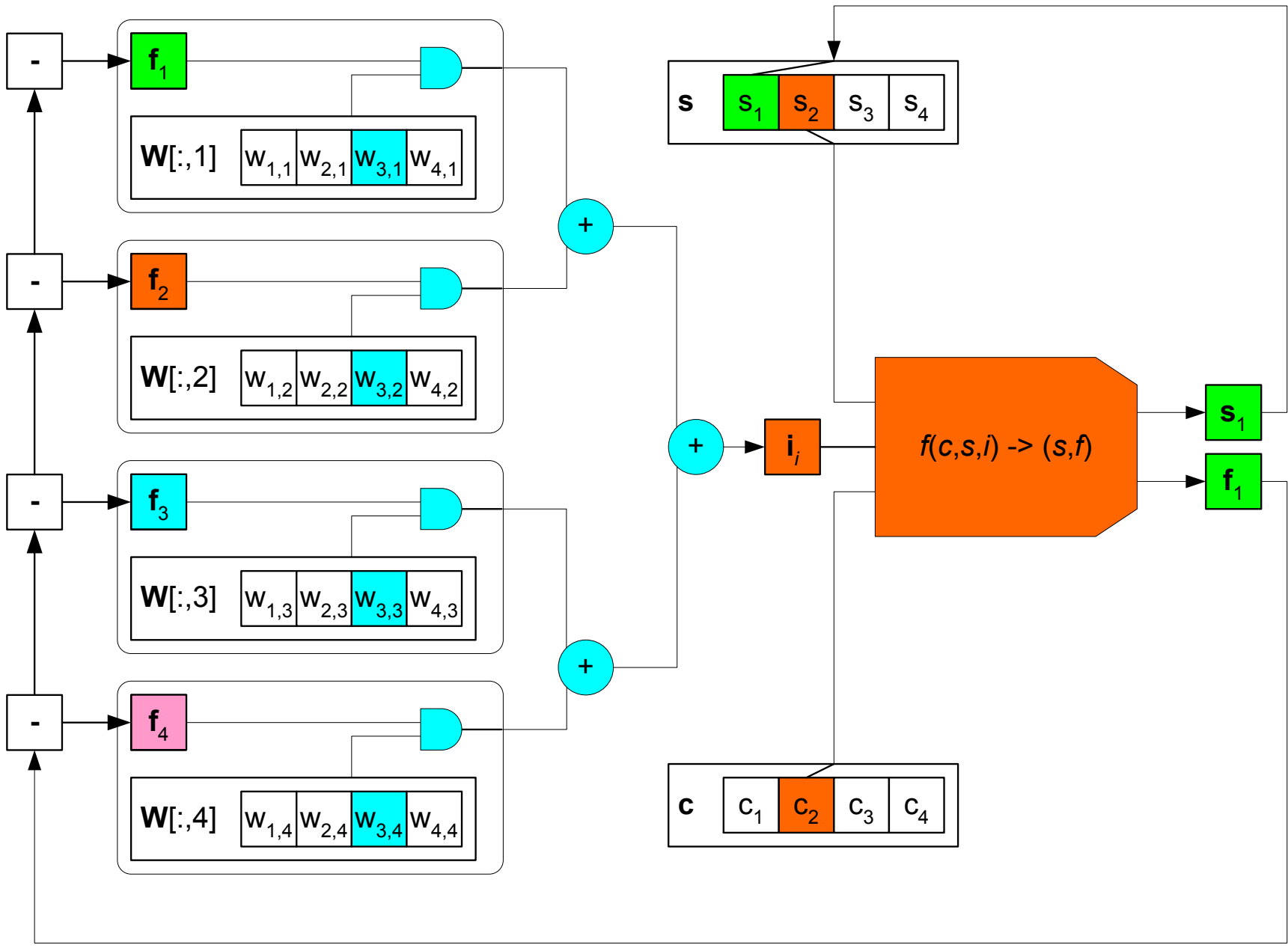


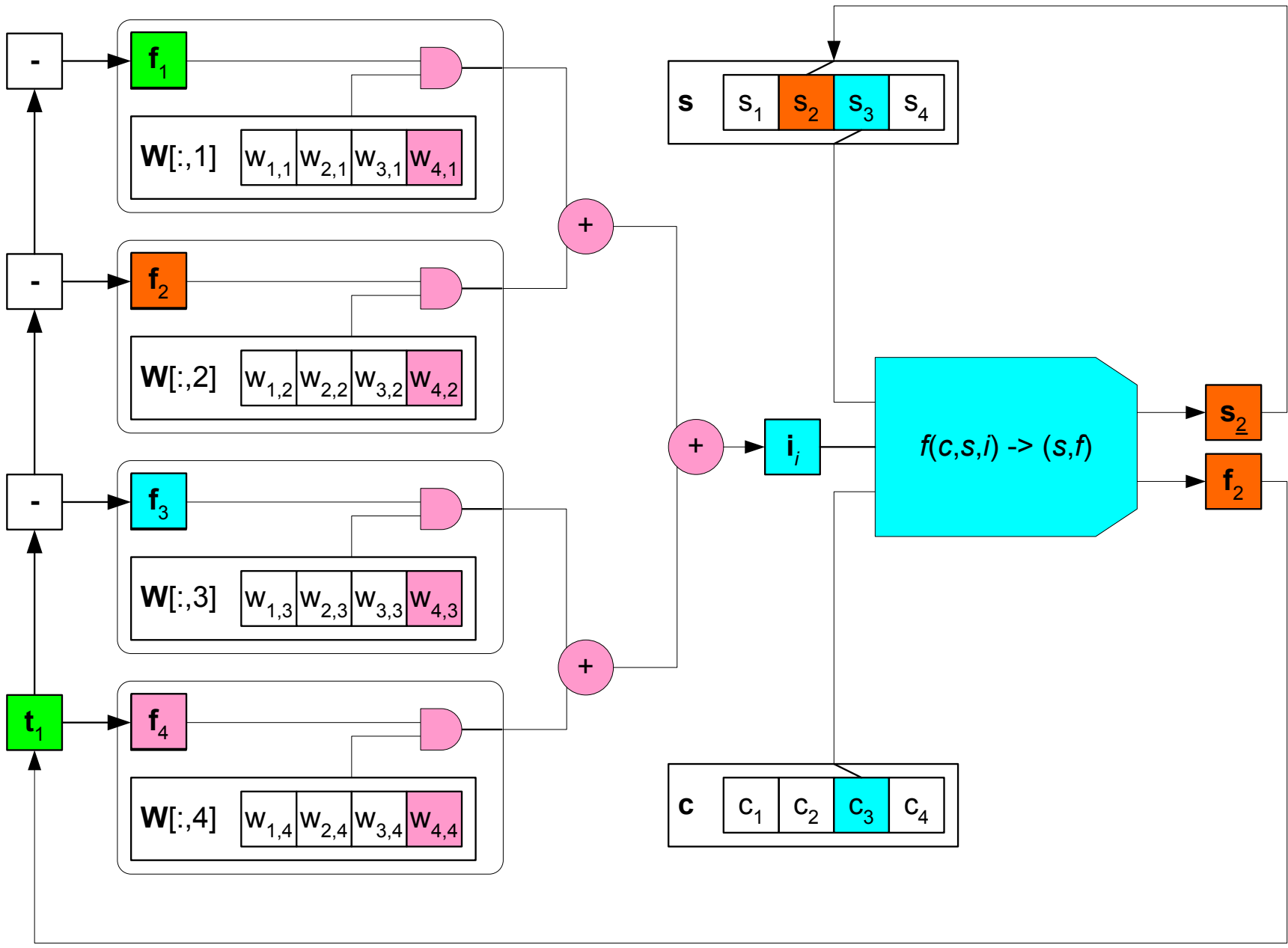


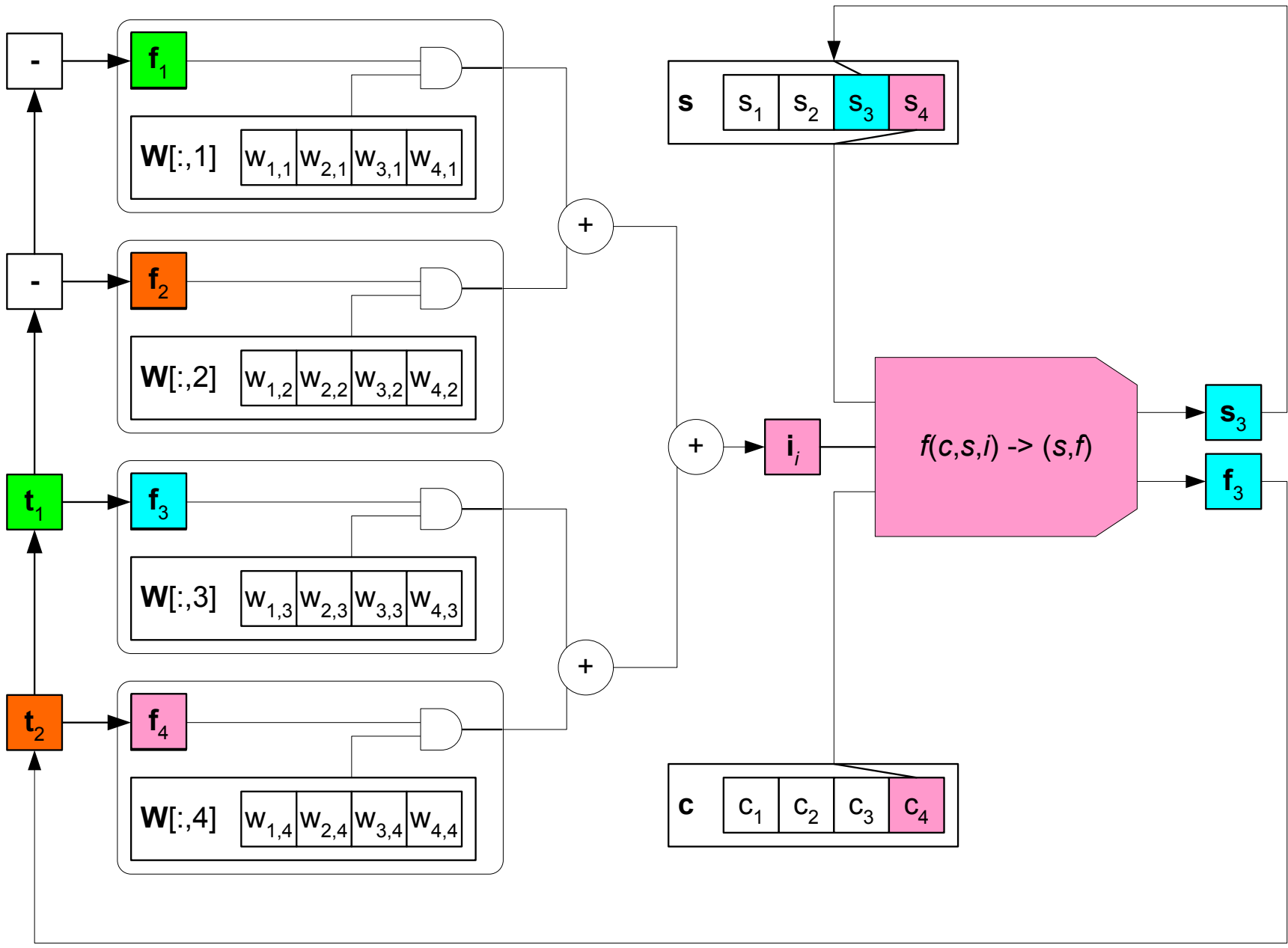


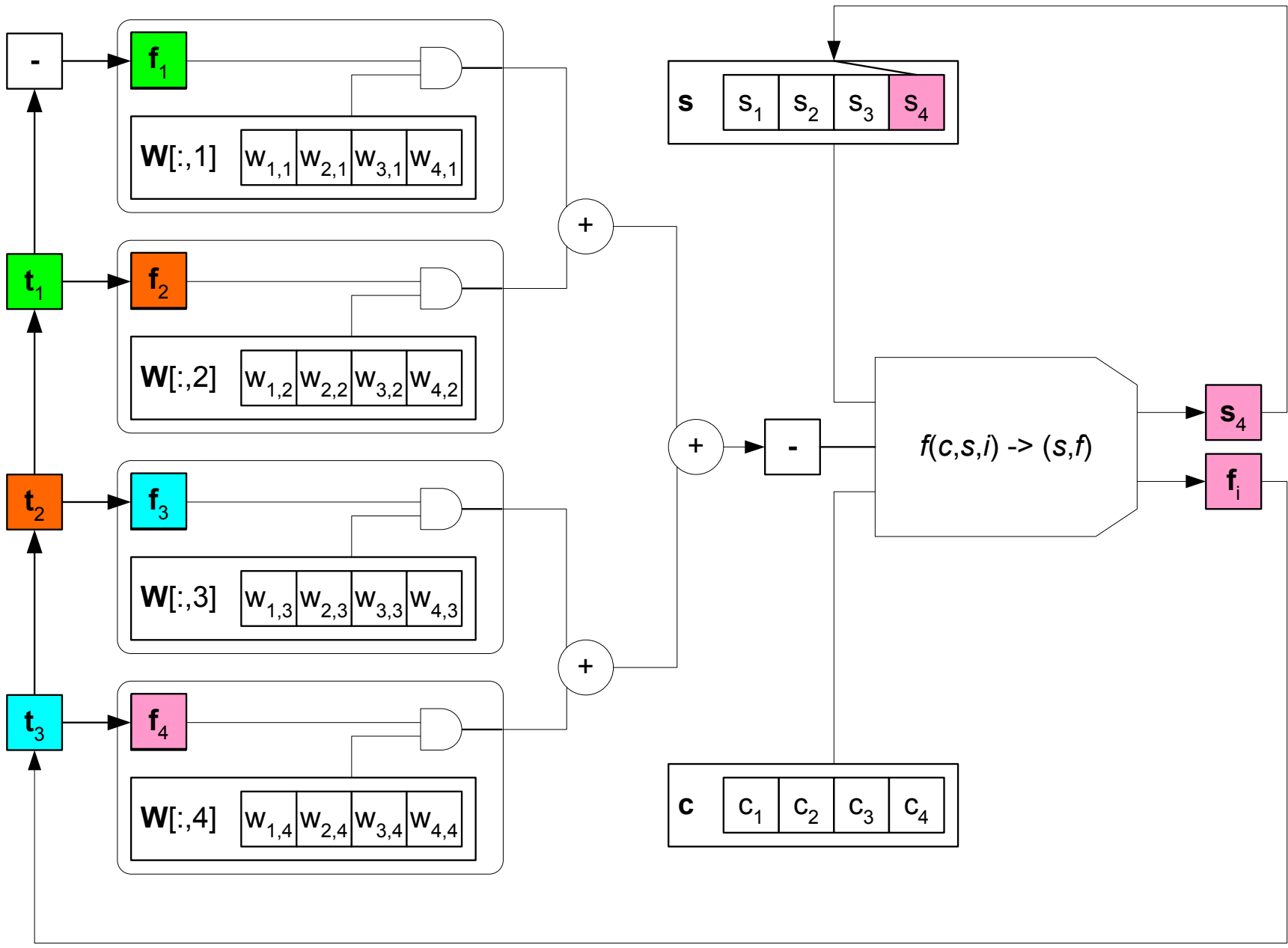


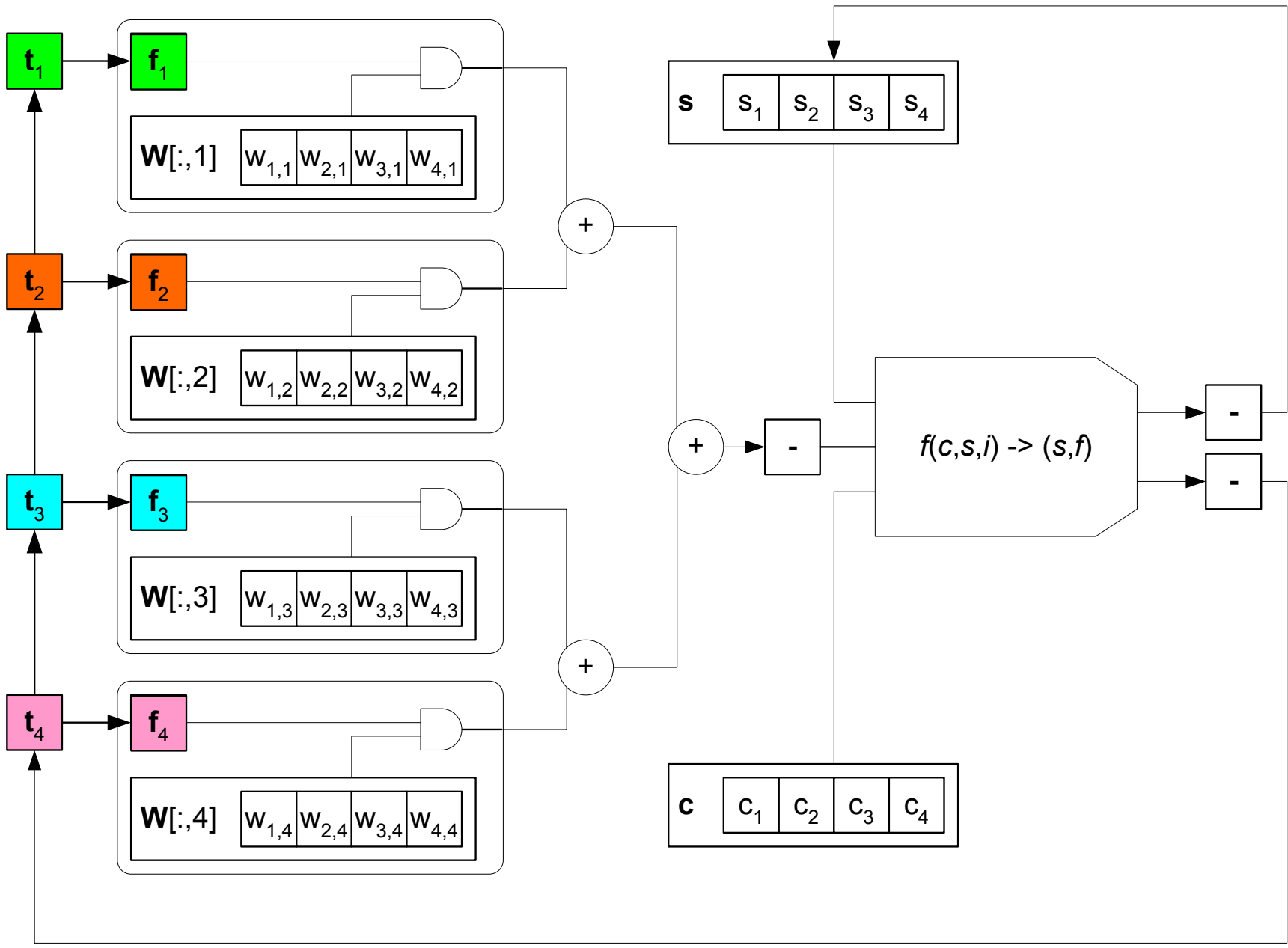






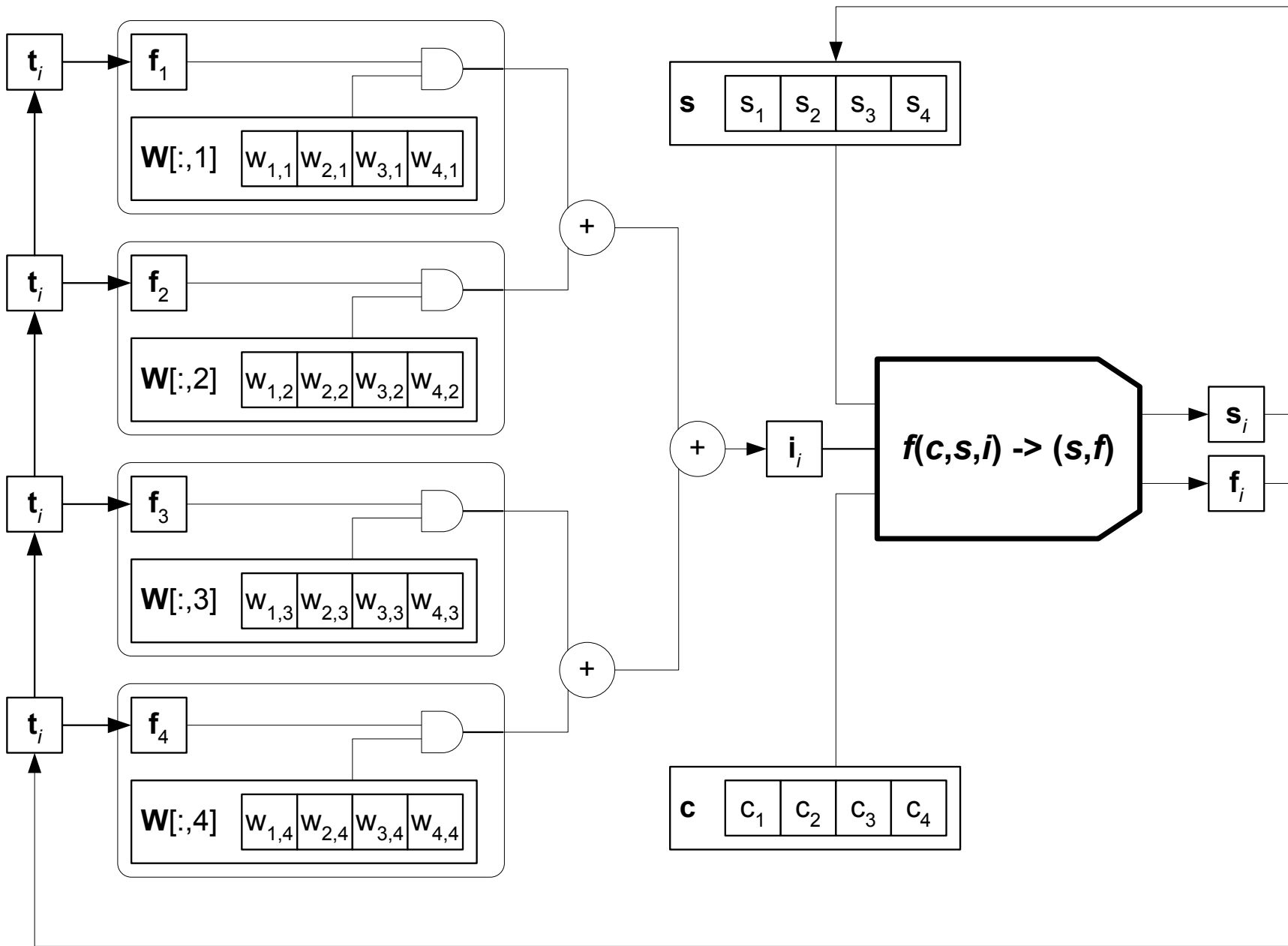






# Fan-in approach: advantages

- Fixed performance: time-step takes  $n+K$  cycles
  - $K$  is pipeline latency for acc. tree and neuron update
  - Performance independent of firing rate
- Exposes lots of spatial and temporal parallelism
  - Can freely pipeline accumulator and neuron update
  - No dependencies between neurons
- Separates stimulus acc. from neuron update
  - Can use any neuron update function
  - Neuron update can be deeply pipelined
- Support fully connected neural networks
  - Every neuron connected to every other neuron



# Neuron Update Function

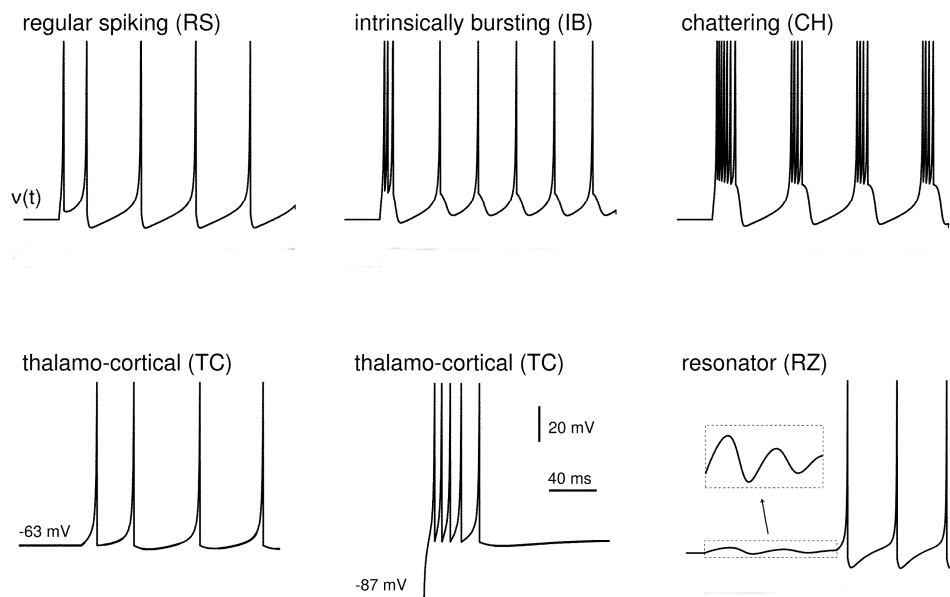
- Neuron update function  $f(\mathbf{c}_i, \mathbf{s}_i, \mathbf{i}_i) \rightarrow (\mathbf{s}_i, \mathbf{f}_i)$ 
  - In: neuron parameters, current state, thalamic input
  - Out: new state, whether neuron fired
  - Can pick any function and plug it in
- Many possible functions: simplicity vs. plausibility
  - Leaky integrate and fire: capacitor-like model
  - Hodgkin-Huxley: incredibly detailed biological model
- Our choice: Izhikevich model
  - Artificial model - captures all types of neuron behaviour
  - Requires 20 floating-point ops. per neuron update
  - Incorporates RNG: simulate stochastic features

# Izhikevich model

- Two-dimensional ODE
  - Two state variables :  $\mathbf{s}_i = (u, v)$
  - Four parameters :  $\mathbf{c}_i = (a, b, c, d)$
- Incorporates Gaussian RNG
  - Noise added to thalamic input
- Approximately 20 FLOP/step

```

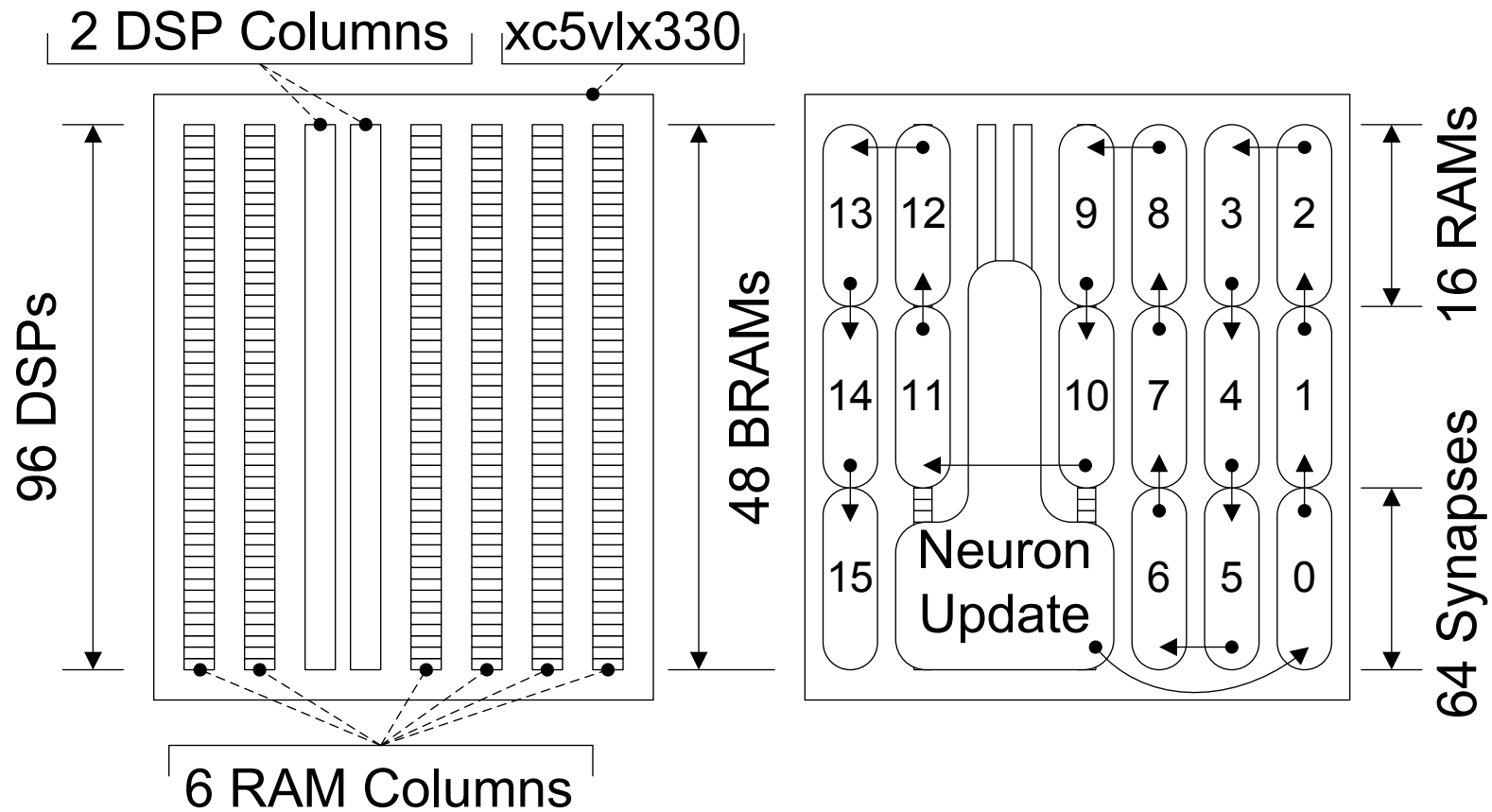
f(  $\mathbf{c}_i$  ,  $\mathbf{s}_i$  ,  $\mathbf{i}_i$ )  $\rightarrow$  (  $\mathbf{s}_i$  ,  $\mathbf{f}_i$ )
f( [a,b,c,d], [u,v], i)  $\rightarrow$  ( [u',v'], f)
begin
  i = i + N(0,1)
  v' = 0.04v2 + 5v + 140 - u + i
  u' = a (b v - u)
  if v' > 30 then
    f = 1
    v' = c
    u' = u + d
  else
    f = 0
  end if
end
    
```



# FPGA Implementation

- Implemented for Virtex-5 (xc5vlx330t)
  - VHDL; XST (ISE 9.2); 133MHz target; Alpha-Data board
- Stimulus accumulation
  - Groups of 4 synapse weights stored in one block-RAM
  - Fixed-point (8-bit) synapse weights
  - 64 synapse (16 RAM) group: relatively-placed
  - Entire accumulation tree: absolutely placed
- Neuron update function
  - Single- and double-precision FP cores from Xilinx
  - Fully pipelined: single - 96 cycles; double - 147 cycles

# FPGA Placement

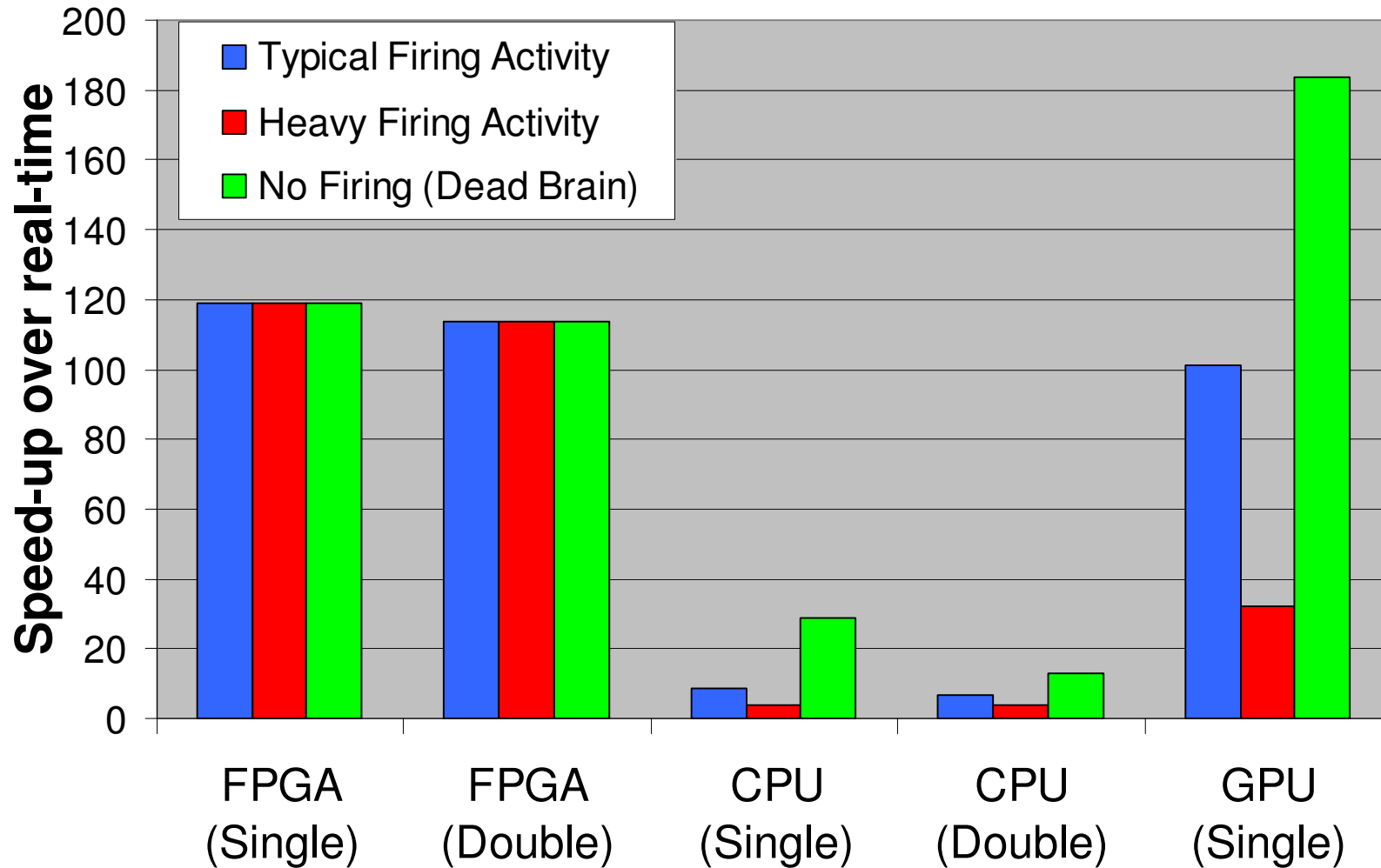


# Resource Utilisation

	Format	LUT-FF Pairs <sup>1</sup> (%)	DSP (%)	RAM (%)	MHz
<b>Synapses</b>	8-bit	24226 (11.7)	-	256 (79)	311
<b>Neuron Update</b>	Single	8727 (4.2)	16 (8.3)	8 (2.5)	307
	Double	18147 (8.8)	96 (50.0)	15 (4.6)	214
<b>Entire System</b>	Single	33452 (16.1)	16 (8.3)	264 (81.5)	133
	Double	43263 (20.9)	96 (50.0)	271 (83.6)	133

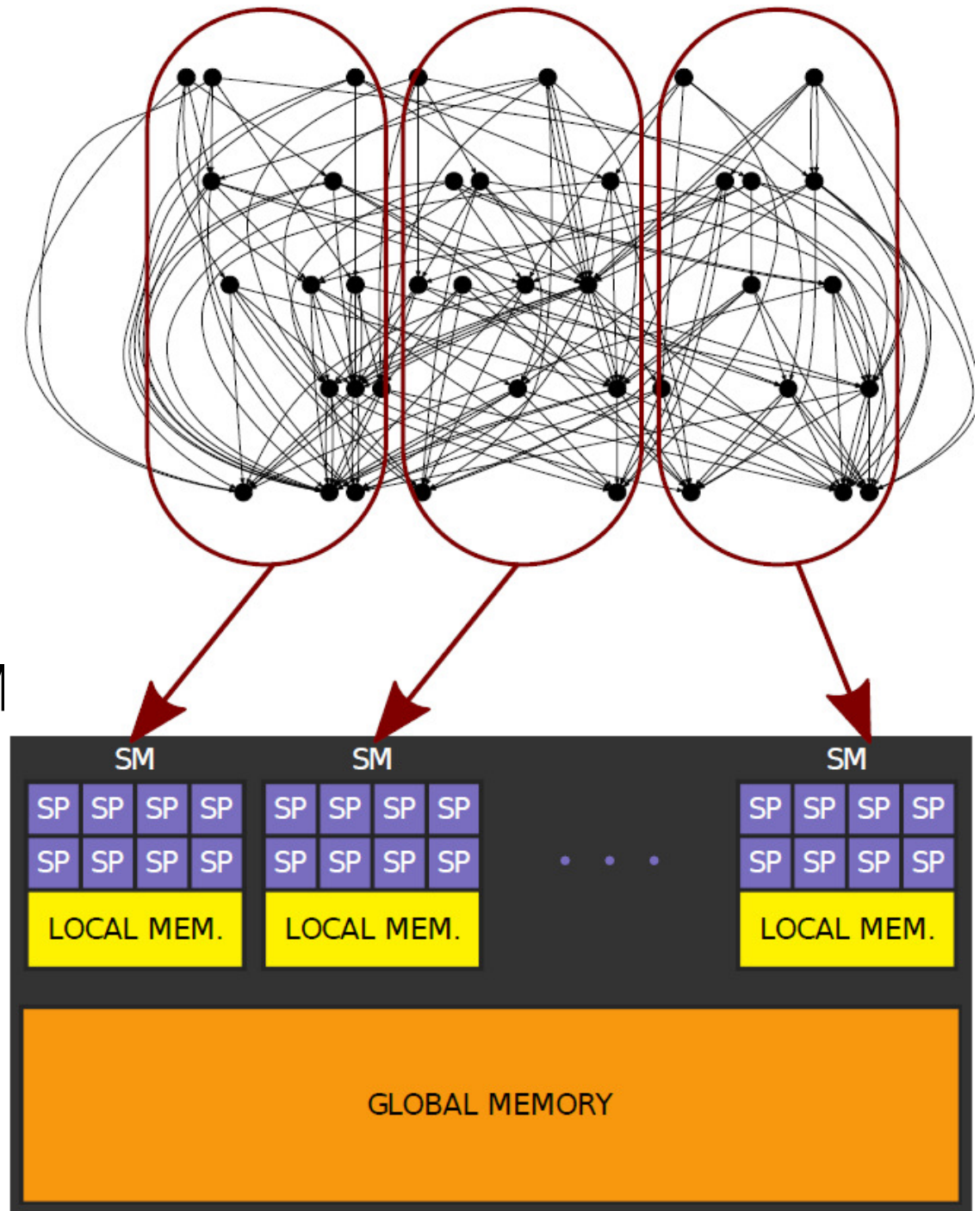
[1] – LUT-FF Pair = Fully or partially utilised LUT-FF pair

# Performance Comparison



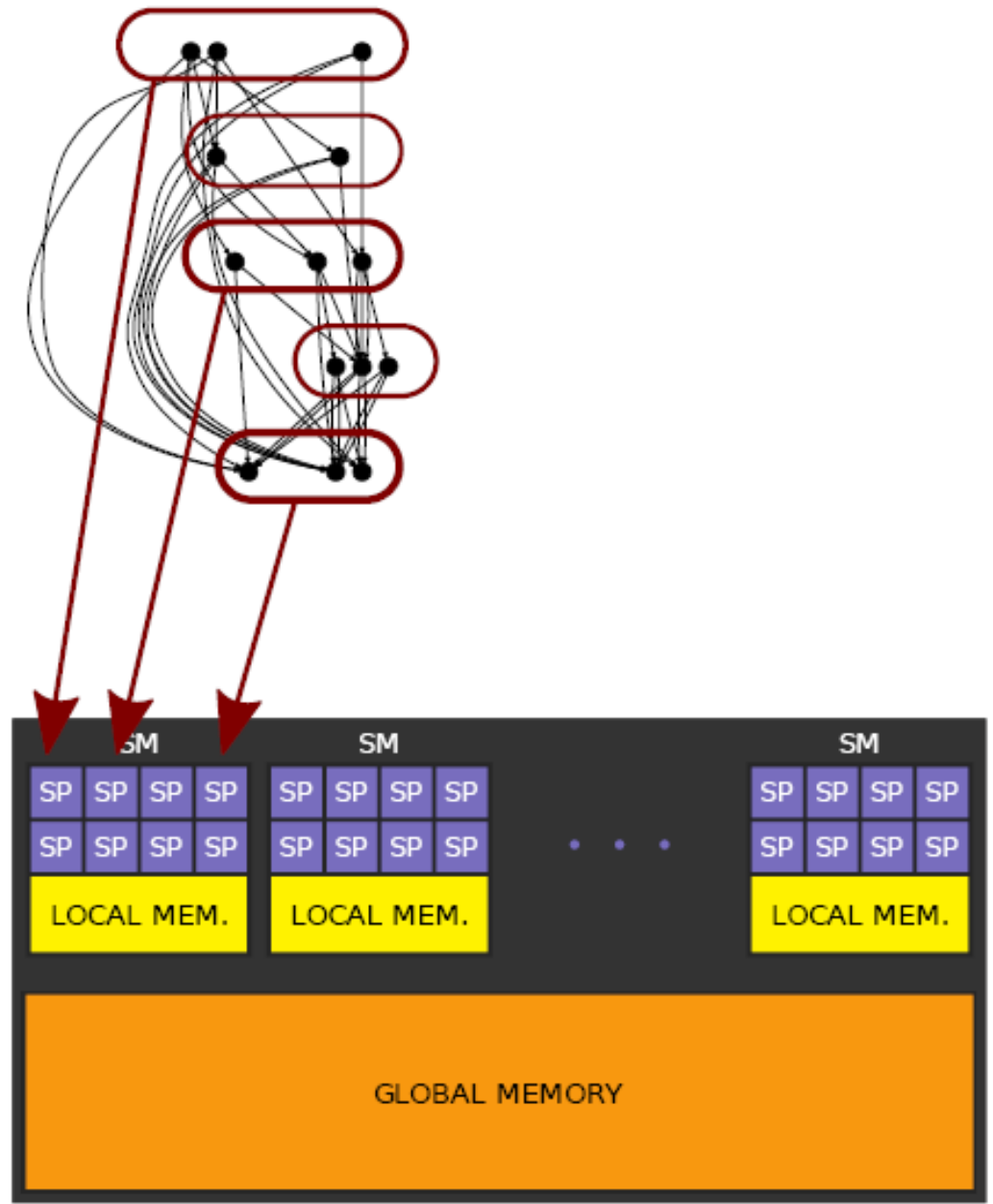
# 4. GPU Design

- Parallel simulation kernel
- Conduction delays
- Partition network
- Each partition to SM

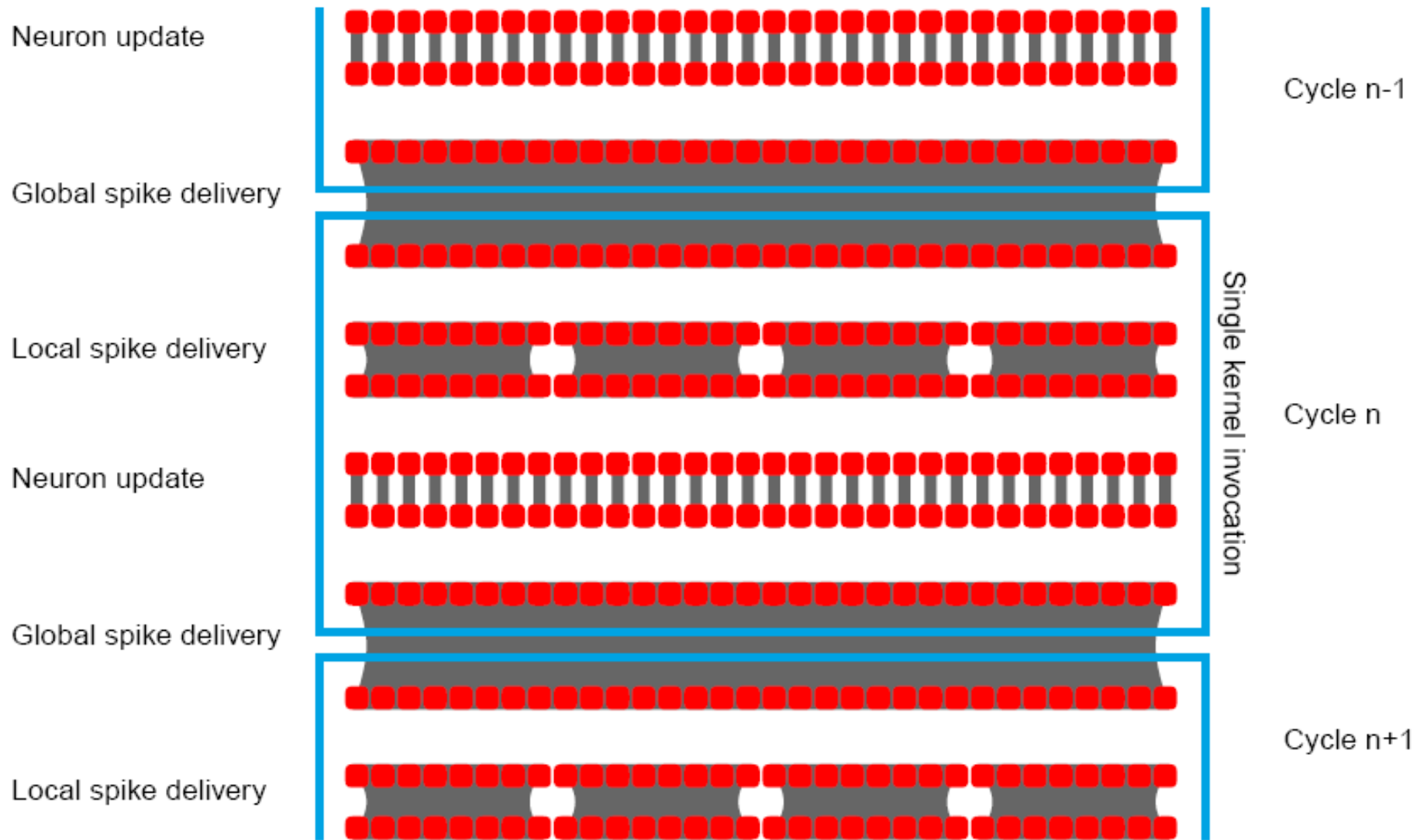


# Sub-division

- Sub-divide partition
- Each processor handles a small group of neurons
- Connection types:
  - local
  - global



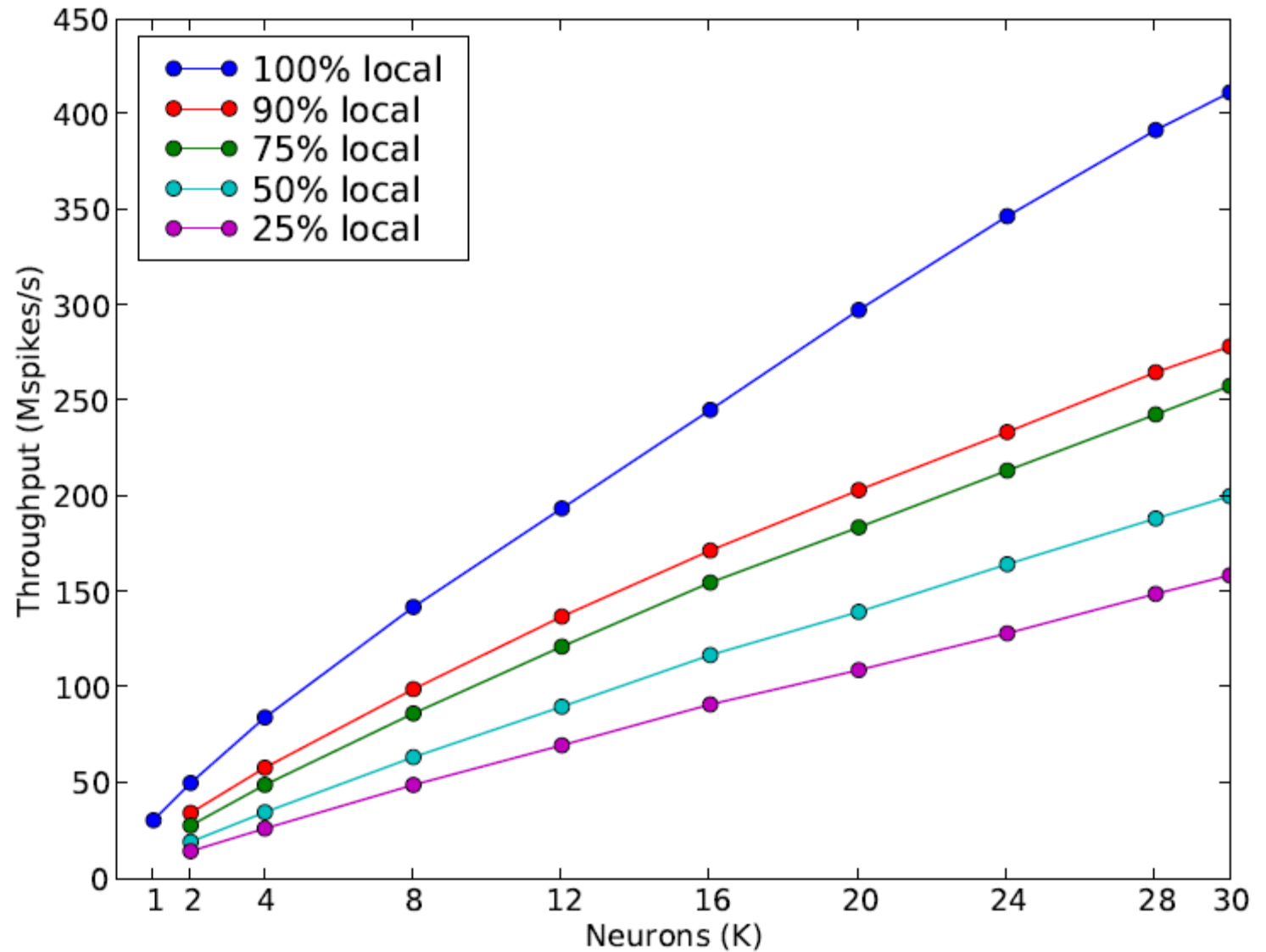
# NeMo: Kernel overview



# Spike delivery

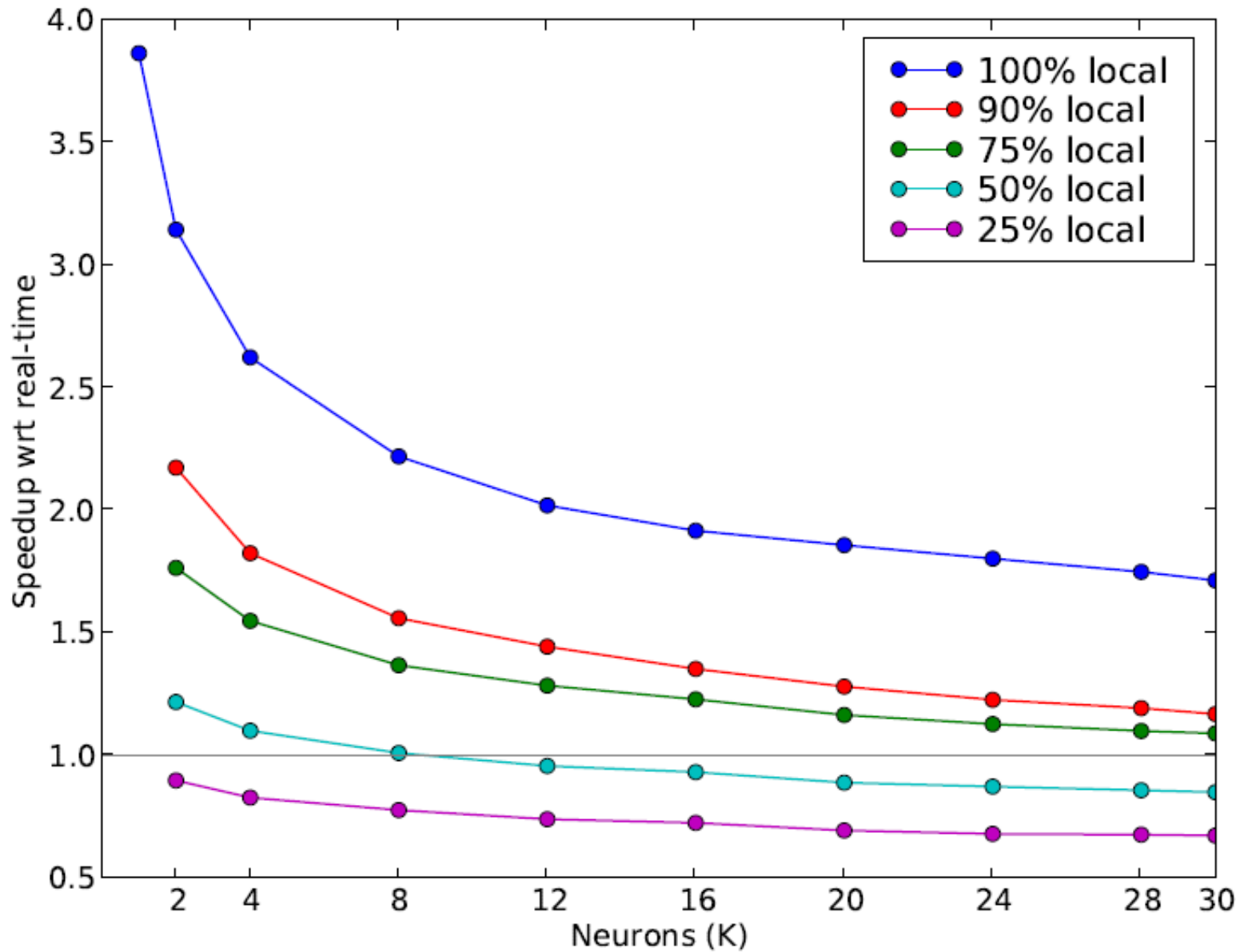
- Local
  - Firing history + delay bits: what spikes to deliver
  - Conduction delay: just-in-time delivery
  - Expensive scatter: local memory
- Global
  - Non-coalesced global memory access: scatter/gather
  - 2D grid of queues, 1 entry per partition pair
  - Synchronisation: single MP local memory

# Results: Throughput



Best case:  
40,000 neurons  
40M synapses  
10Hz firing

# Results: Speedup



Networks with 75% locality: simulate faster than real-time

# Comparison

Source	Type	Device	Clock	TP	Remarks
NeMo	GPU	Tesla C1060	1.3GHz	400M	
Jayram et al.	GPU	GTX280	1.3GHz	85M	STDP
Izhikevich	CPU	Phenom 9650	2.3GHz	15M	single core, n = 1k
Thomas/Luk	FPGA	XC5VLX330	133MHz	700M	no delays, n ≤ 1k

# 5. Future Work

- Improve or maintain performance
  - higher model complexity (eg gap junction), neuron number
- FPGA Architectures: sparse networks
  - Compression: connectivity matrix, synapse weights
- Incorporate plasticity models
  - Change synapse weights over time – learning ability
- Investigate impact of fixed-point weights
  - Explore appropriate precision
- Cluster of nodes, each with FPGA + GPU + CPU
  - Optimal mapping, design automation, scalability

# 6. Summary

- Exploit latest off-the-shelf parallelism
- FPGA
  - Synapses: on-chip block memory
  - Pipelined neuron update core: 1 neuron per cycle
  - 1000 fully-connected neurons, 700M spikes/second
- GPU
  - Parallel simulation kernel, support conductance delay
  - Effective computation + memory partitioning
  - 32000 neurons, 400M spikes/second
- Initial single-chip results... watch this space!