Turing Complete Computation with Synfire Ring-Based Neural Networks

DARPA - Lifelong Learning Machines (L2M) PI Meeting

Jérémie Cabessa

Laboratory of Mathematical Economics
Université Paris II, Paris, France

September 22, 2019, Chicago, IL, USA

Project Summary

- We introduce a bio-inspired paradigm for neural computation based on the concept of synfire rings.
- Phase 1: Propose a Turing-complete neural architecture based on synfire rings.
- Phase 2: Develop learning algorithms on this architecture:
  - ⋆ Towards neuromorphic implementation.
  - ⋆ Towards (real) biological implementation.

Project Summary

- We introduce a bio-inspired paradigm for neural computation based on the concept of synfire rings.
- Phase 1: Propose a Turing-complete neural architecture based on synfire rings.
- Phase 2: Develop learning algorithms on this architecture:
  - ⋆ Towards neuromorphic implementation.
  - ⋆ Towards (real) biological implementation.

Project Summary

- We introduce a bio-inspired paradigm for neural computation based on the concept of synfire rings.
- Phase 1: Propose a Turing-complete neural architecture based on synfire rings.
- Phase 2: Develop learning algorithms on this architecture:
  - ⋆ Towards neuromorphic implementation.
  - ⋆ Towards (real) biological implementation.

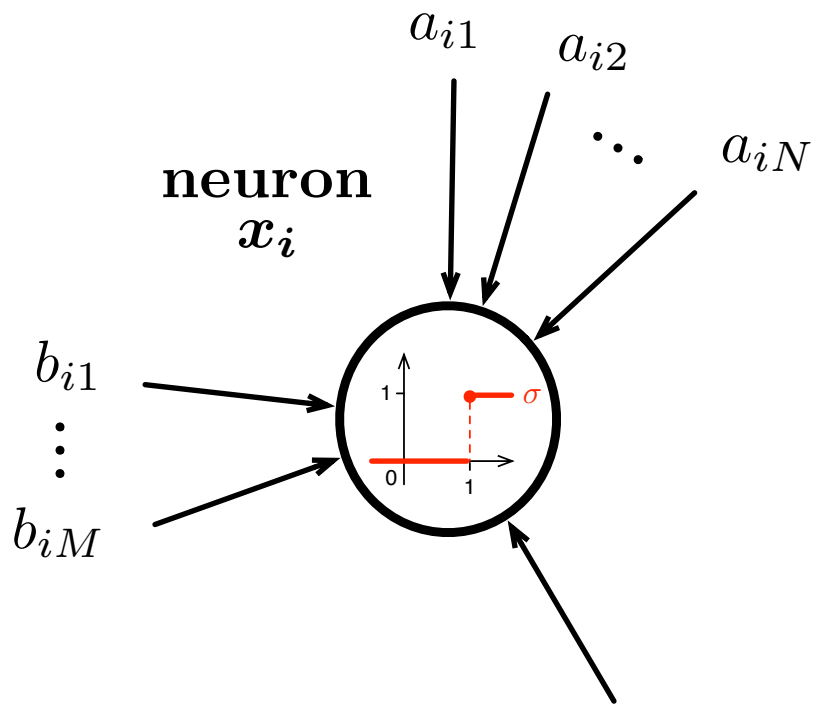
Project Summary

- We introduce a bio-inspired paradigm for neural computation based on the concept of synfire rings.
- Phase 1: Propose a Turing-complete neural architecture based on synfire rings.
- Phase 2: Develop learning algorithms on this architecture:
  - ⋆ Towards neuromorphic implementation.
  - ⋆ Towards (real) biological implementation.

Project Summary

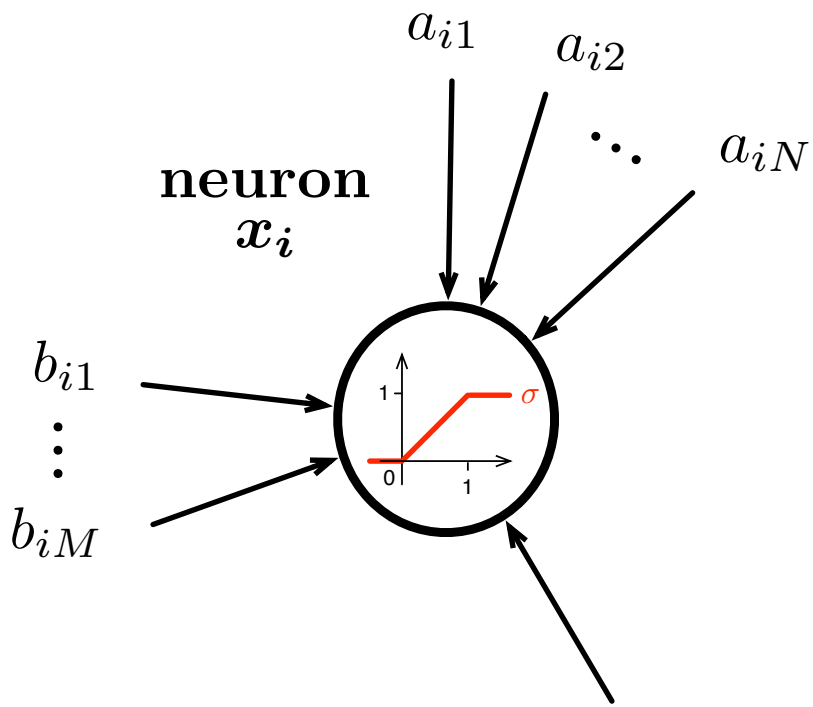
- We introduce a bio-inspired paradigm for neural computation based on the concept of synfire rings.
- Phase 1: Propose a Turing-complete neural architecture based on synfire rings.
- Phase 2: Develop learning algorithms on this architecture:
  - ⋆ Towards neuromorphic implementation.
  - ⋆ Towards (real) biological implementation.

Boolean Recurrent Neural Networks



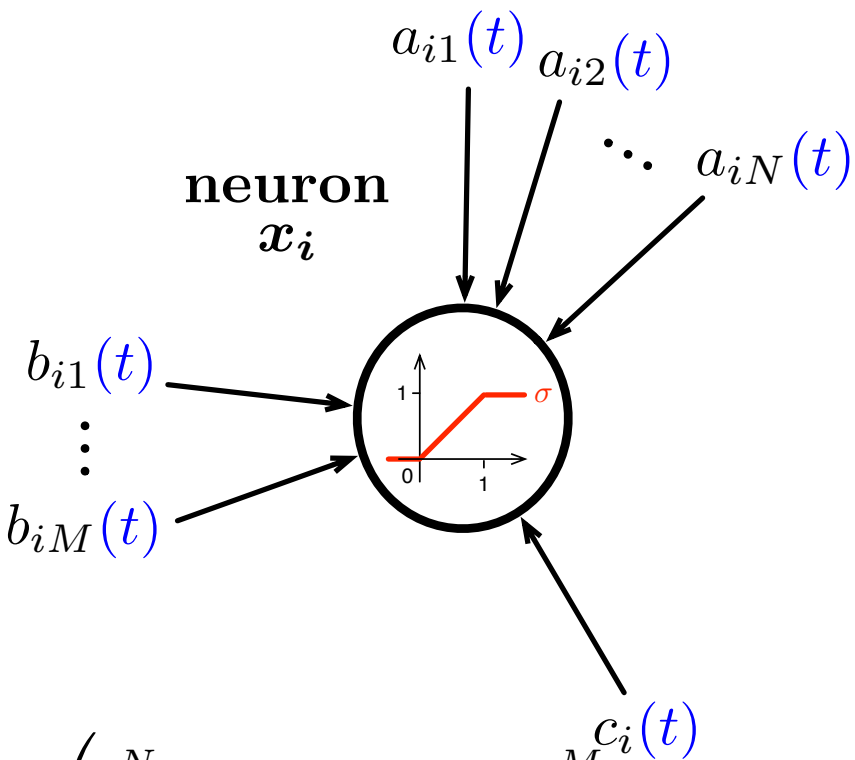
$$x_i(t+1) = \theta\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

Sigmoid Recurrent Neural Networks



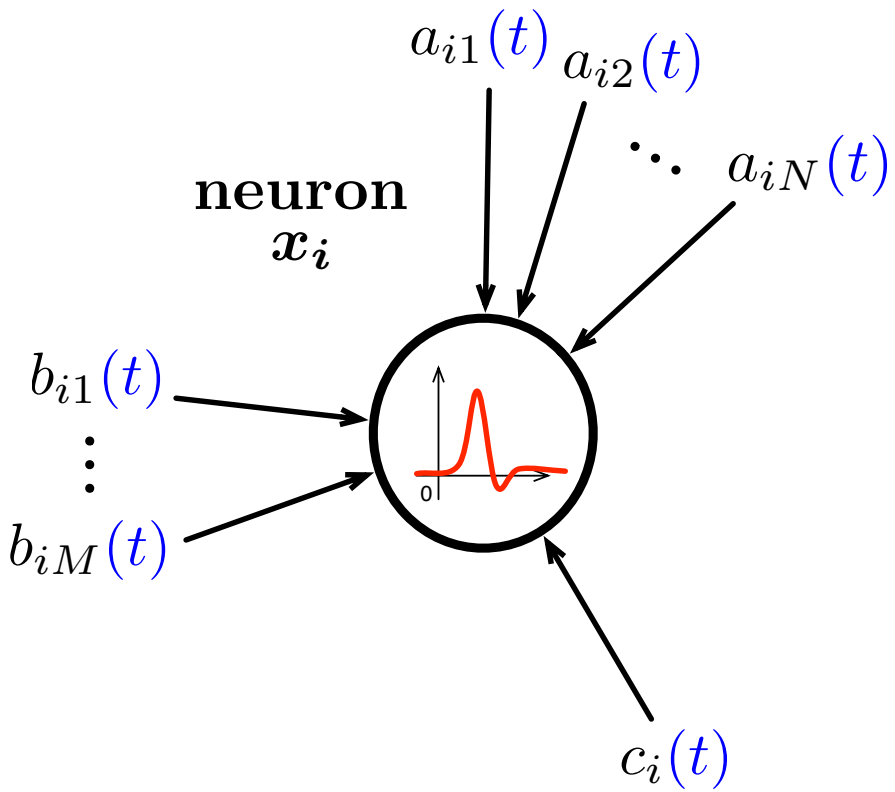
$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij} \cdot x_j(t) + \sum_{j=1}^{M} b_{ij} \cdot u_j(t) + c_i\right)$$

Sigmoid Recurrent Neural Networks



$$x_i(t+1) = \sigma\left(\sum_{j=1}^{N} a_{ij}(t) \cdot x_j(t) + \sum_{j=1}^{M} b_{ij}(t) \cdot u_j(t) + c_i(t)\right)$$

Spiking Recurrent Neural Networks



Neurons' activities are modelled by Izhikevich or Hodgkin-Huxley differential equations.

Hodgkin-Huxley Spiking Neural Networks

$$\alpha_n(V_m) = \frac{0.01(10 - V_m)}{\exp(\frac{10-V_m}{10}) - 1} \qquad \beta_n(V_m) = 0.125 \exp(\frac{-V_m}{80})$$

$$\alpha_m(V_m) = \frac{0.1(25 - V_m)}{\exp(\frac{25-V_m}{10}) - 1} \qquad \beta_m(V_m) = 4 \exp(\frac{-V_m}{18})$$

$$\alpha_h(V_m) = 0.07 \exp(\frac{-V_m}{20}) \qquad \beta_h(V_m) = \frac{1}{\exp(\frac{30-V_m}{10}) + 1}$$

$$\frac{dn}{dt} = \alpha_n(V_m)(1 - n) - \beta_n(V_m)n$$

$$\frac{dm}{dt} = \alpha_m(V_m)(1 - m) - \beta_m(V_m)m$$

$$\frac{dh}{dt} = \alpha_h(V_m)(1 - h) - \beta_h(V_m)h$$

$$C_m \frac{dV_m}{dt} = I - \bar{g}_{\mathsf{K}} n^4 (V_m - V_K) - \bar{g}_{\mathsf{Na}} m^3 h (V_m - V_{Na}) - \bar{g}_l (V_m - V_l)$$

State of the Art Results

	Boolean	Sigmoid		
		Static	Bi-Valued Evolving	Evolving
$\mathbb{Q}$	FSA **REG**	TM **P**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56	SiegelmannSontag95	CabessaSiegelmann14	CabessaSiegelmann14
	Minsky67	CabessaSiegelmann12	CabessaVilla15,16	CabessaVilla15,16
	CabessaVilla14	CabessaVilla15,16	CabessaFinkel18	CabessaFinkel18
$\mathbb{R}$	FSA **REG**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56	SiegelmannSontag94	CabessaSiegelmann14	CabessaSiegelmann14
	Minsky67	CabessaSiegelmann12	CabessaVilla15,16	CabessaVilla15,16
	CabessaVilla14	CabessaVilla15,16	CabessaFinkel18	CabessaFinkel18

State of the Art Results

	Boolean		Sigmoid	
		Static	Bi-Valued Evolving	Evolving
$\mathbb{Q}$	FSA **REG**	TM **P**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56	SiegelmannSontag95	CabessaSiegelmann14	CabessaSiegelmann14
	Minsky67	CabessaSiegelmann12	CabessaVilla15,16	CabessaVilla15,16
	CabessaVilla14	CabessaVilla15,16	CabessaFinkel18	CabessaFinkel18
$\mathbb{R}$	FSA **REG**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56	SiegelmannSontag94	CabessaSiegelmann14	CabessaSiegelmann14
	Minsky67	CabessaSiegelmann12	CabessaVilla15,16	CabessaVilla15,16
	CabessaVilla14	CabessaVilla15,16	CabessaFinkel18	CabessaFinkel18

State of the Art Results

	Boolean		Sigmoid	
		Static	Bi-Valued Evolving	Evolving
$\mathbb{Q}$	FSA **REG**	TM **P**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag95 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18
$\mathbb{R}$	FSA **REG**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag94 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18

State of the Art Results

	Boolean	Sigmoid		
		Static	Bi-Valued Evolving	Evolving
$\mathbb{Q}$	FSA **REG**	TM **P**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag95 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18
$\mathbb{R}$	FSA **REG**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag94 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18

State of the Art Results

	Boolean	Sigmoid		
		Static	Bi-Valued Evolving	Evolving
$\mathbb{Q}$	FSA **REG**	TM **P**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag95 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18
$\mathbb{R}$	FSA **REG**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**	TM/poly(A) **P/poly**
	Kleene56 Minsky67 CabessaVilla14	SiegelmannSontag94 CabessaSiegelmann12 CabessaVilla15,16	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18	CabessaSiegelmann14 CabessaVilla15,16 CabessaFinkel18

Negative Results

- In practice, the effectiveness of ML is mainly due to the increase in computational power of computers.
- In theory, back-propagation is not efficient!

Theorem (Blum & Rivest 1992)

Training a specific 3-node neural network composed of threshold cells is NP-complete.

Theorem (Sima 1995)

Back-propagation training a specific 3-node neural network composed of sigmoidal cells is NP-hard.

Negative Results

- In practice, the effectiveness of ML is mainly due to the increase in computational power of computers.
- In theory, back-propagation is not efficient!

Theorem (Blum & Rivest 1992)

Training a specific 3-node neural network composed of threshold cells is NP-complete.

Theorem (Sima 1995)

Back-propagation training a specific 3-node neural network composed of sigmoidal cells is NP-hard.

Negative Results

- In practice, the effectiveness of ML is mainly due to the increase in computational power of computers.
- In theory, back-propagation is not efficient!

Theorem (Blum & Rivest 1992)

Training a specific 3-node neural network composed of threshold cells is NP-complete.

Theorem (Sima 1995)

Back-propagation training a specific 3-node neural network composed of sigmoidal cells is NP-hard.

Negative Results

- In practice, the effectiveness of ML is mainly due to the increase in computational power of computers.
- In theory, back-propagation is not efficient!

Theorem (Blum & Rivest 1992)

Training a specific 3-node neural network composed of threshold cells is NP-complete.

Theorem (Sima 1995)

Back-propagation training a specific 3-node neural network composed of sigmoidal cells is NP-hard.

Negative Results

- In practice, the effectiveness of ML is mainly due to the increase in computational power of computers.
- In theory, back-propagation is not efficient!

Theorem (Blum & Rivest 1992)

Training a specific 3-node neural network composed of threshold cells is NP-complete.

Theorem (Sima 1995)

Back-propagation training a specific 3-node neural network composed of sigmoidal cells is NP-hard.

Towards Novel Paradigms of Neural Computation

- Computational states of the machines are represented by spiking configurations of the network.
  - ⋆ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
  - ⋆ Network should be robust to *architectural plasticity* and *synaptic noise*.

Towards Novel Paradigms of Neural Computation

- Computational states of the machines are represented by spiking configurations of the network.
- ⋆ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ⋆ Network should be robust to *architectural plasticity* and *synaptic noise*.

Towards Novel Paradigms of Neural Computation

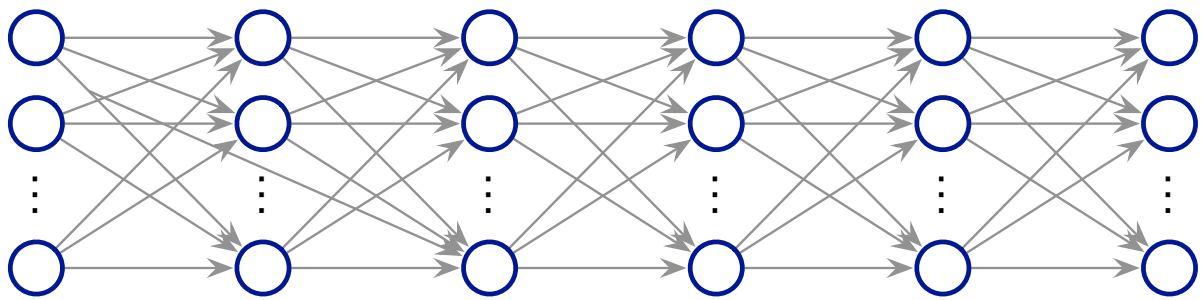
- Computational states of the machines are represented by spiking configurations of the network.
- ⋆ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ⋆ Network should be robust to *architectural plasticity* and *synaptic noise*.

Towards Novel Paradigms of Neural Computation

- Computational states of the machines are represented by spiking configurations of the network.
- ⋆ Computational states should rather be represented by *sustained activities of neural assemblies*, e.g., by *cyclic attractors*.
- Network is not robust to cell death, synaptic plasticity, architectural plasticity in general.
- ⋆ Network should be robust to *architectural plasticity* and *synaptic noise*.

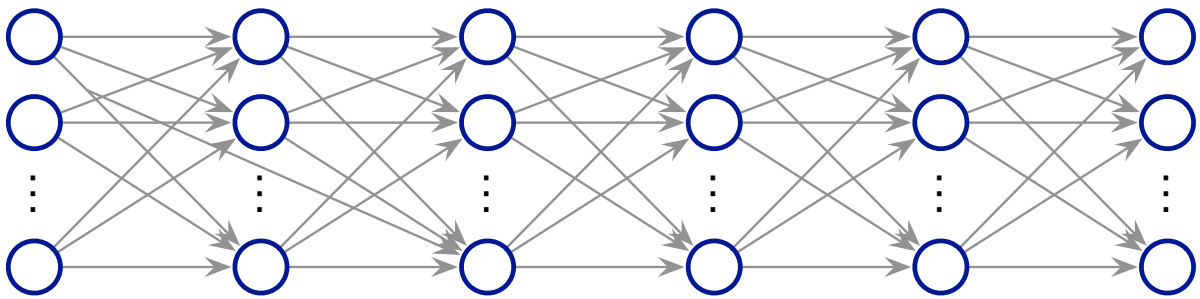
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



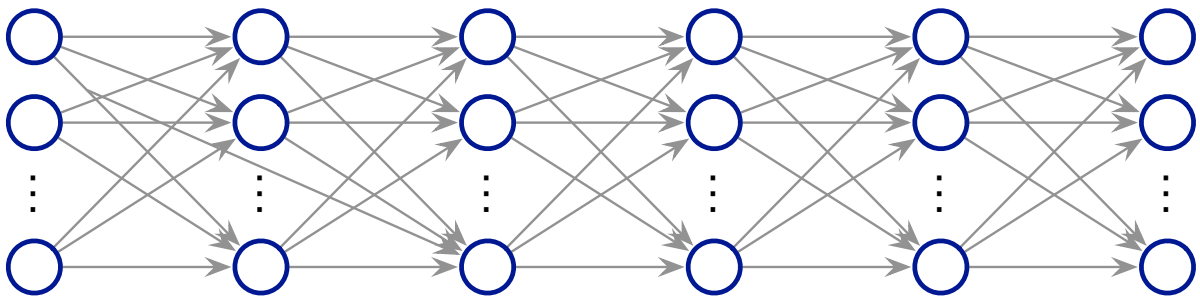
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



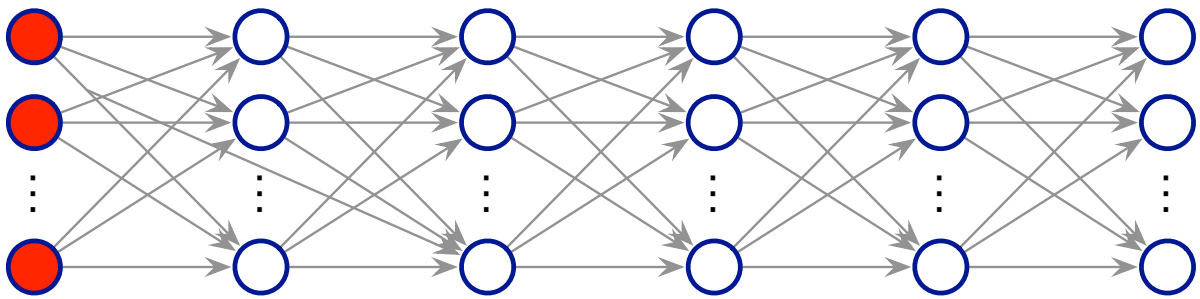
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



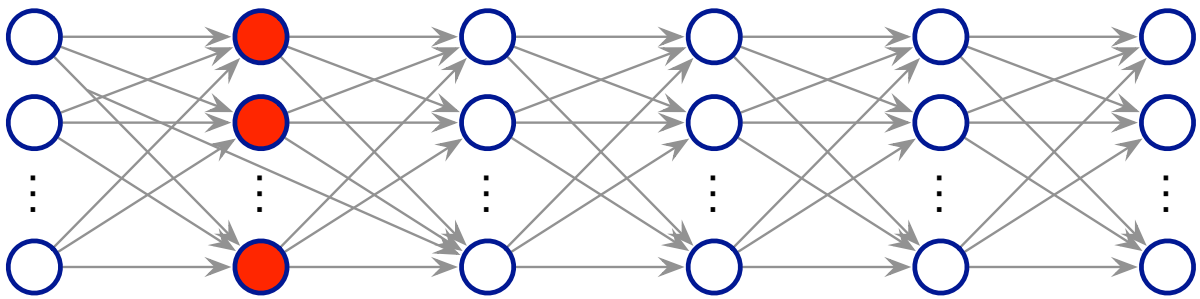
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



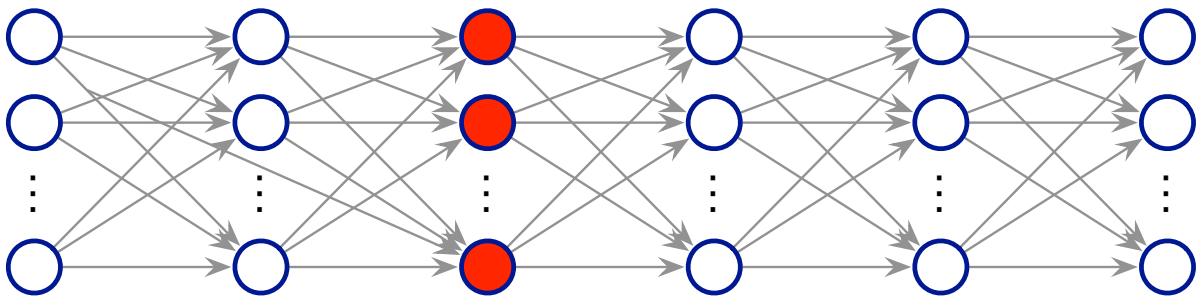
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



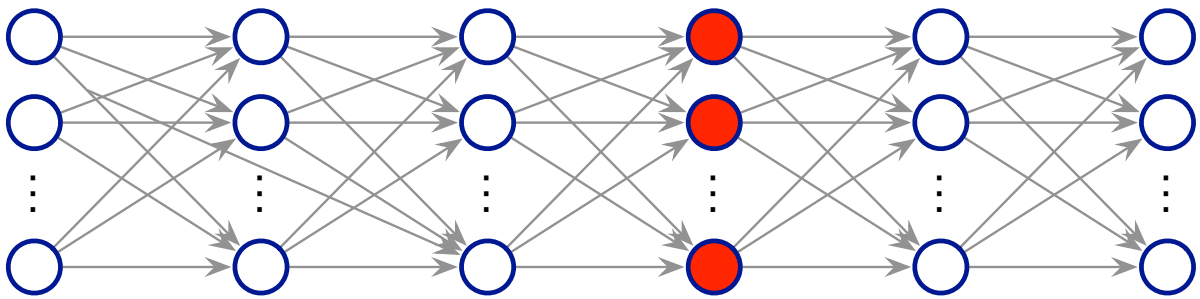
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



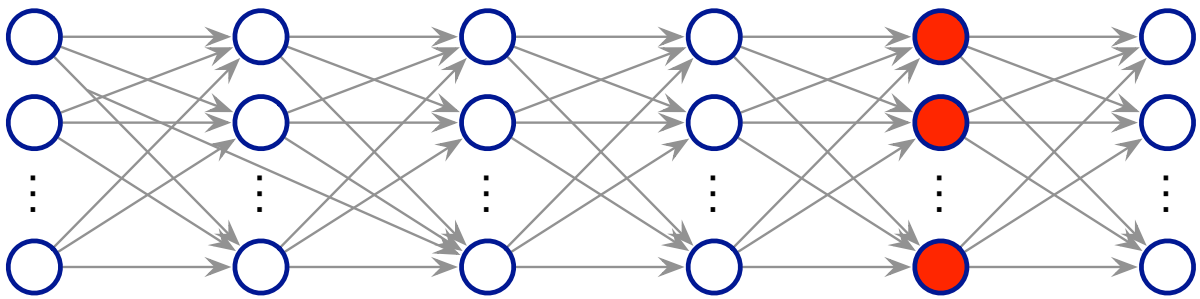
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



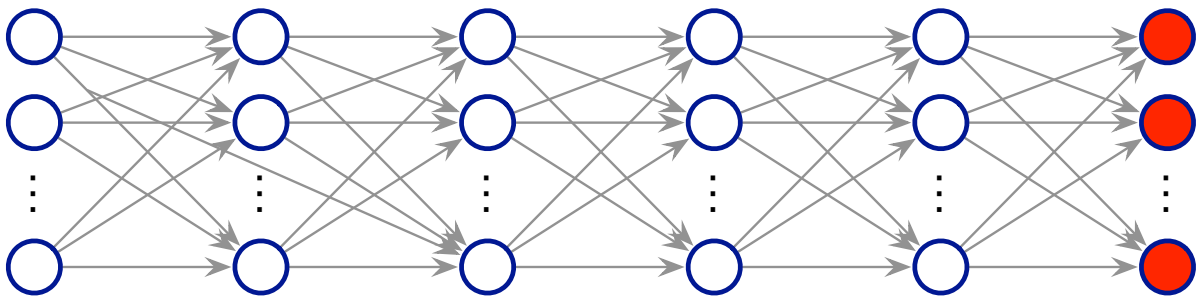
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



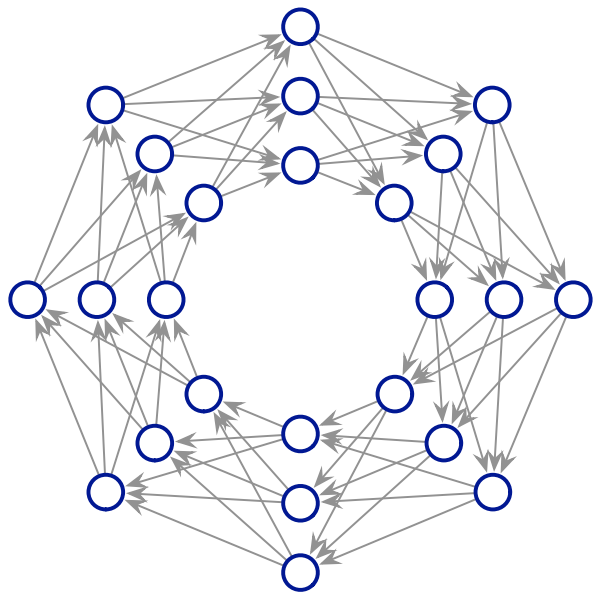
Synfire Chains

- *Synfire chains* allow for robust and highly precise transmission of information in neural networks (Abeles 82).
- *Synfire chains* are likely to be crucially involved in the processing and coding of information in neural networks.



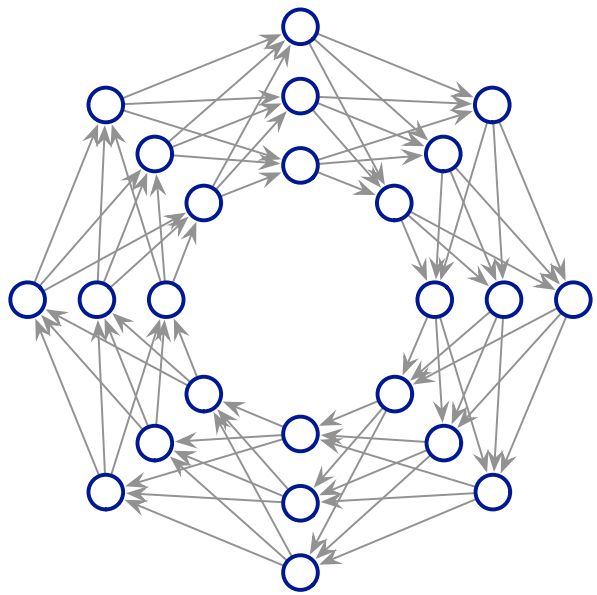
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



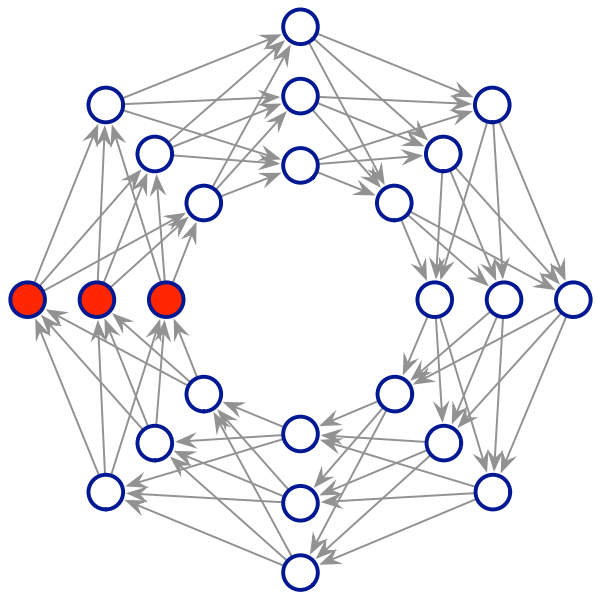
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



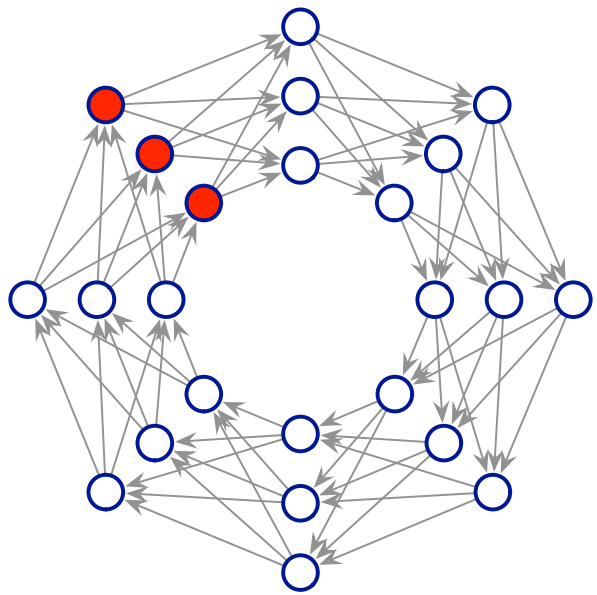
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



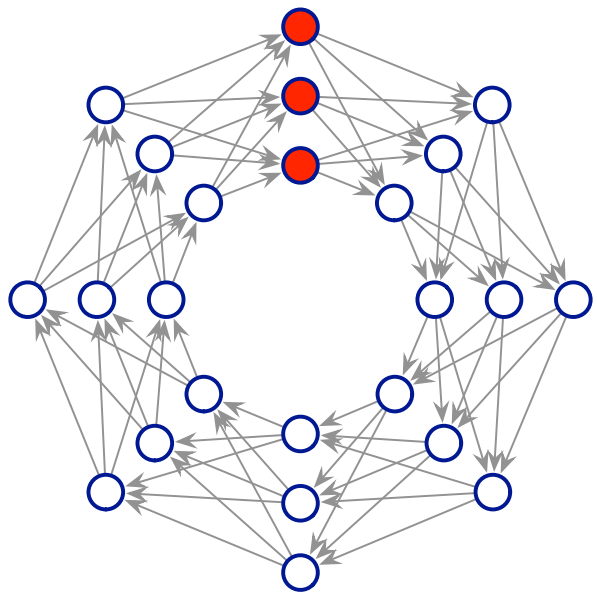
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



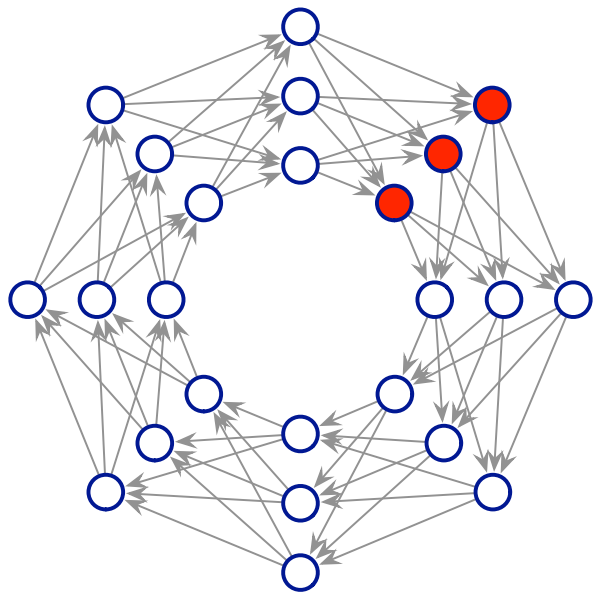
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



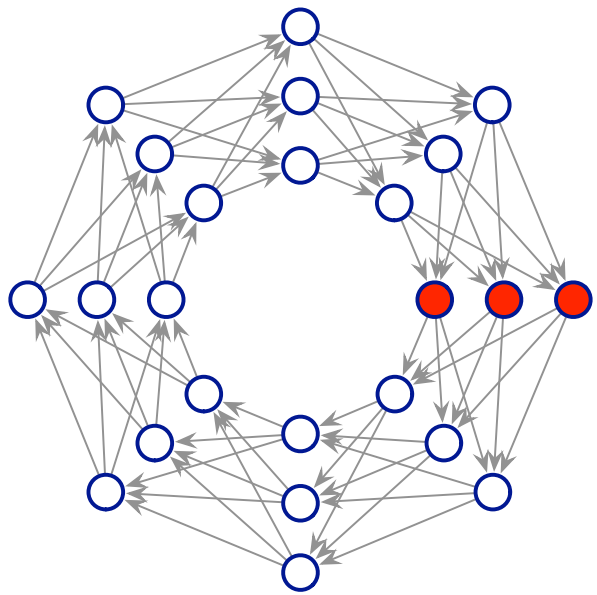
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



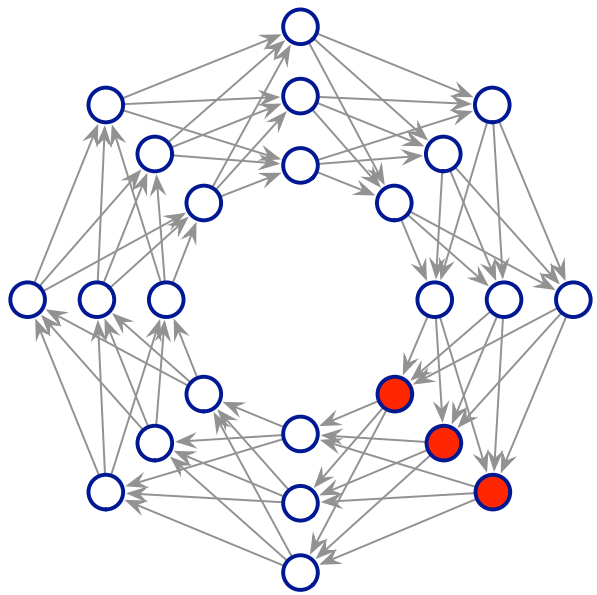
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



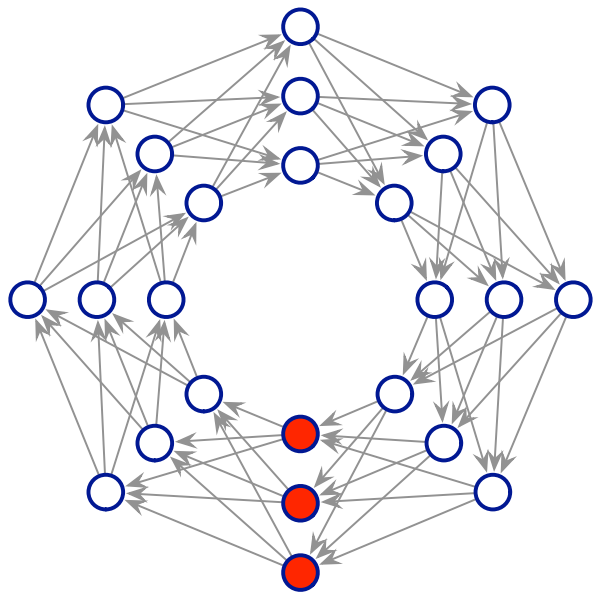
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



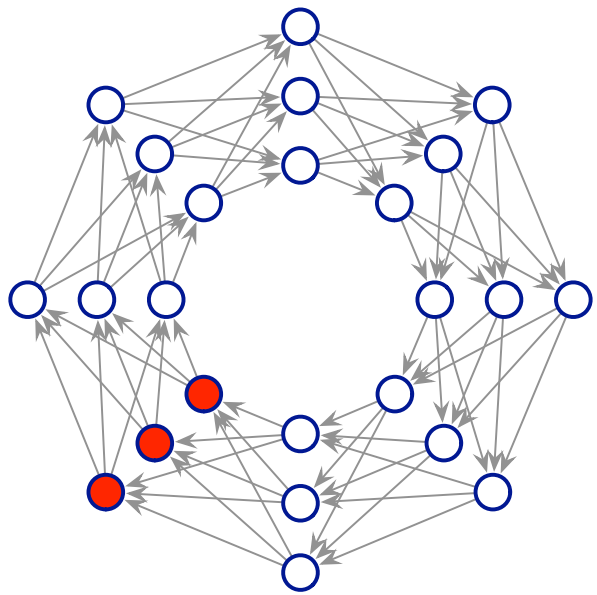
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



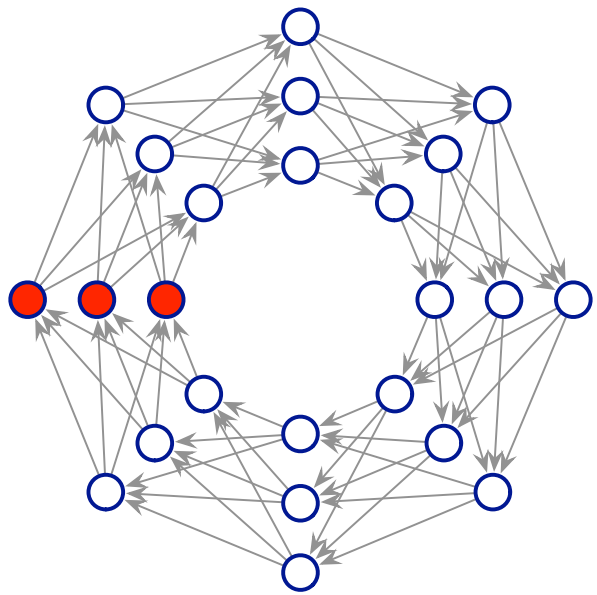
Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.

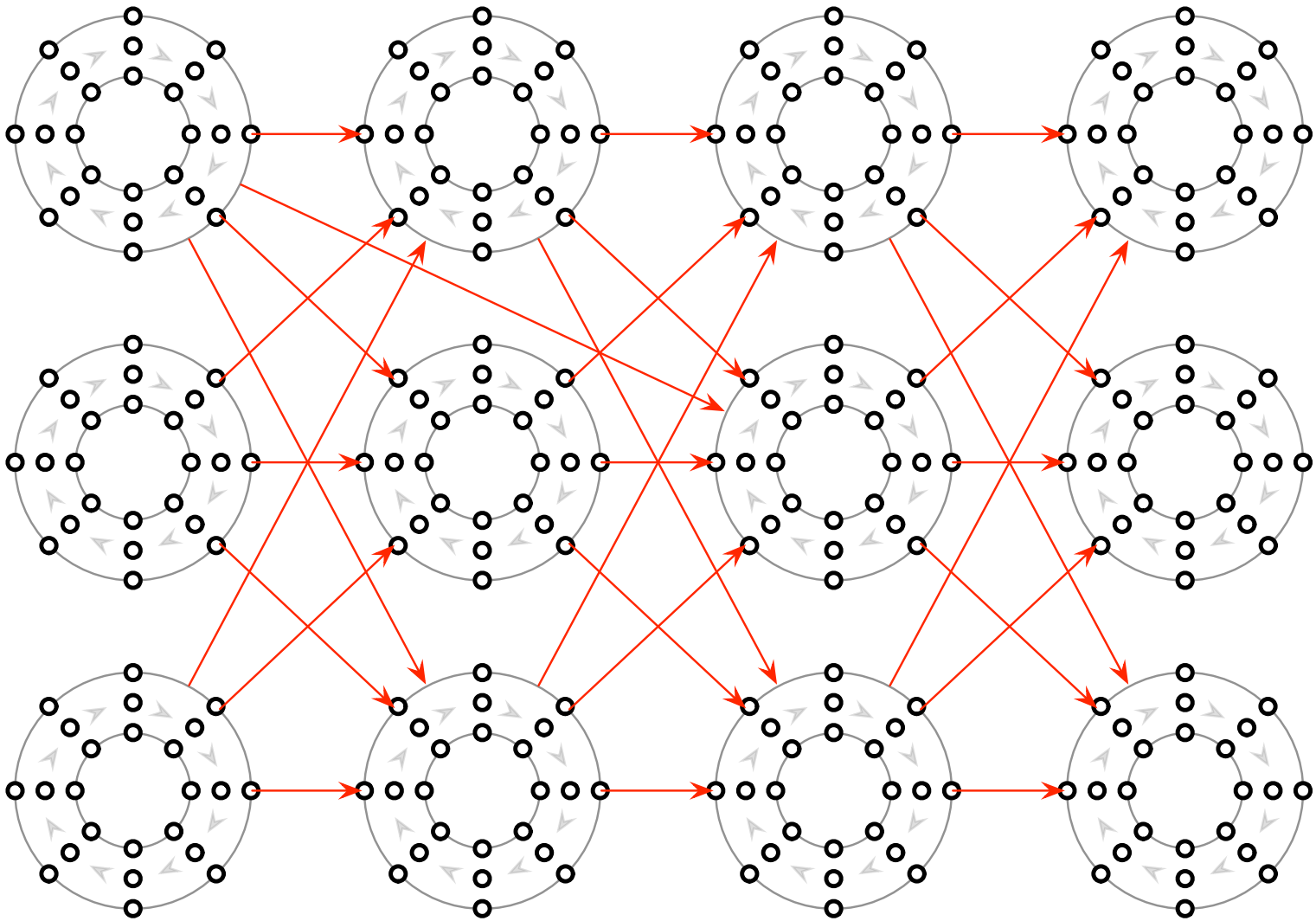


Synfire Rings

- *Synfire rings* are looping synfire chains that have been observed in self-organizing neural networks (Zheng & Triesch 14).
- *Synfire rings* allow for robust and temporally precise *self-sustained activities*.



Synfire Ring Architecture



Neural Computation with Synfire Rings

- We introduce a paradigm of abstract neural computation based on *synfire rings*.
- Computational states are represented by sustained activities of synfire rings, i.e., attractors.
- The transitions between such attractors are perfectly controlled by the inputs.
- The global computational process is robust to some kinds of architectural plasticities and synaptic noises.

Neural Computation with Synfire Rings

- We introduce a paradigm of abstract neural computation based on *synfire rings*.
- Computational states are represented by sustained activities of synfire rings, i.e., attractors.
- The transitions between such attractors are perfectly controlled by the inputs.
- The global computational process is robust to some kinds of architectural plasticities and synaptic noises.

Neural Computation with Synfire Rings

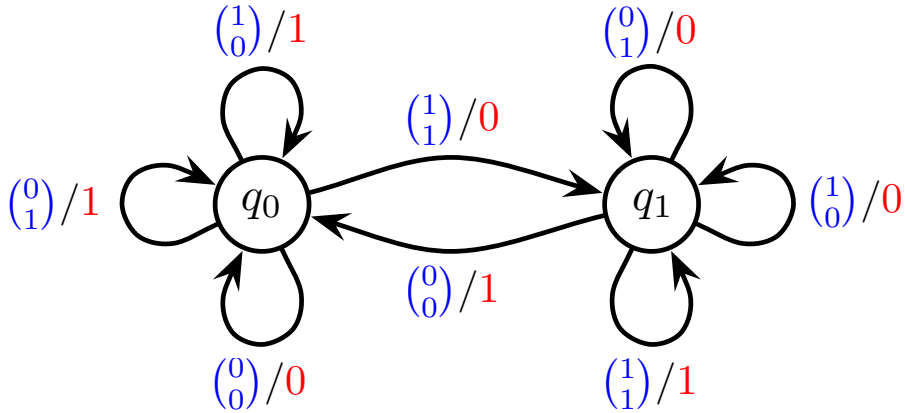
- We introduce a paradigm of abstract neural computation based on *synfire rings*.
- Computational states are represented by sustained activities of synfire rings, i.e., attractors.
- The transitions between such attractors are perfectly controlled by the inputs.
- The global computational process is robust to some kinds of architectural plasticities and synaptic noises.

Neural Computation with Synfire Rings

- We introduce a paradigm of abstract neural computation based on *synfire rings*.
- Computational states are represented by sustained activities of synfire rings, i.e., attractors.
- The transitions between such attractors are perfectly controlled by the inputs.
- The global computational process is robust to some kinds of architectural plasticities and synaptic noises.

Simulation of Finite State Automata

- We simulate finite state automata with synfire ring-based neural networks.

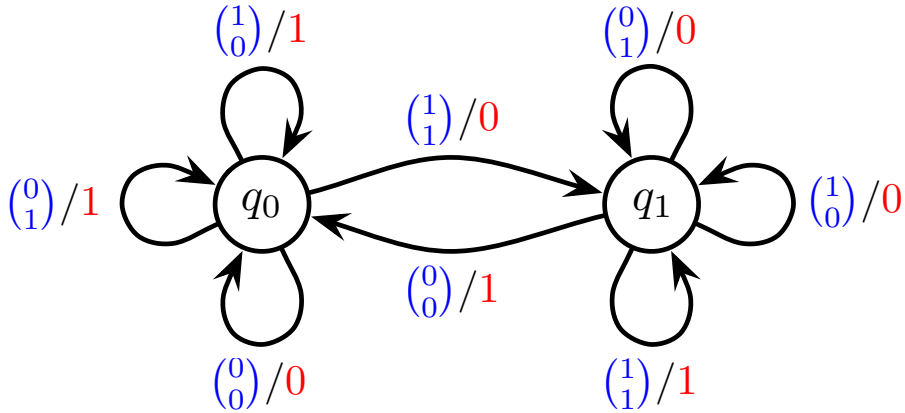


$$\begin{array}{cccccccc} & 1 & 1^{1} & 0^{1} & 1 & 1 & 0^{1} & 1 \\ + & & 1 & 1 & 1 & 0 & 0 & 1 \\ \hline & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

- For this purpose, we need to simulate *transitions* between ring activities.

Simulation of Finite State Automata

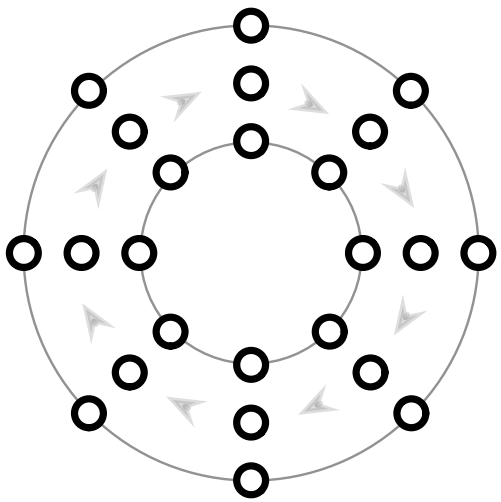
- We simulate finite state automata with synfire ring-based neural networks.

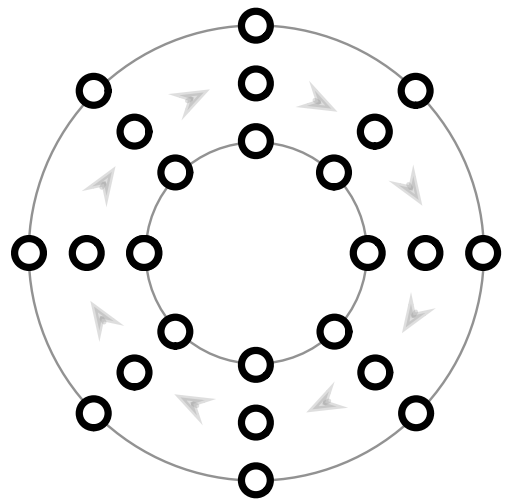


$$\begin{array}{cccccccc} & 1 & 1^{1} & 0^{1} & 1 & 1 & 0^{1} & 1 \\ + & & 1 & 1 & 1 & 0 & 0 & 1 \\ \hline & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

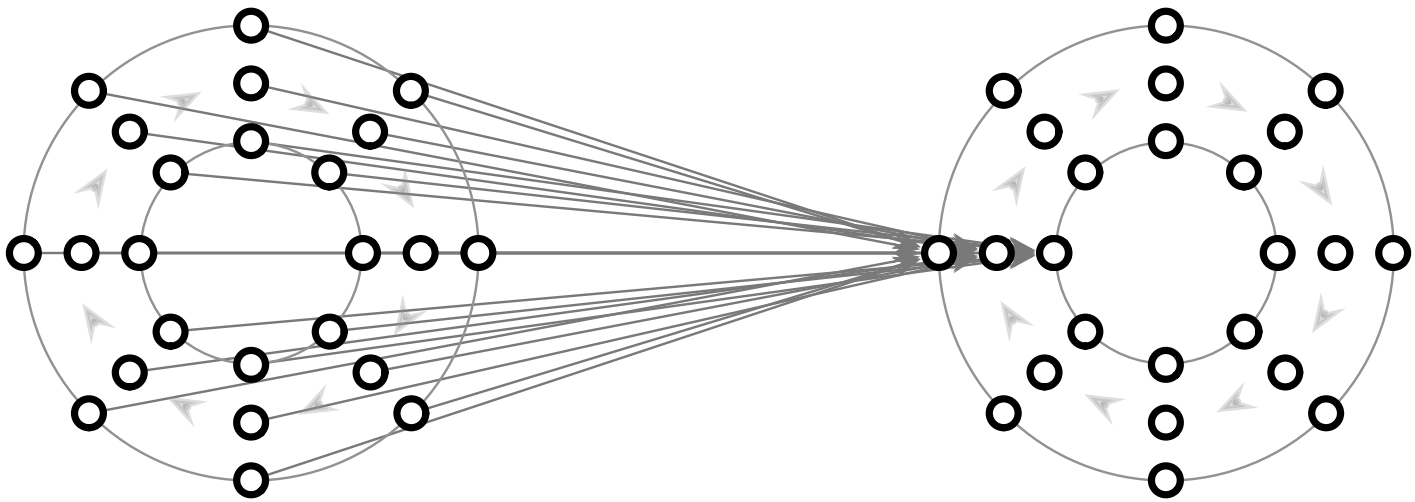
- For this purpose, we need to simulate *transitions* between ring activities.

Ring Transition 1: Excitatory/Inhibitory System

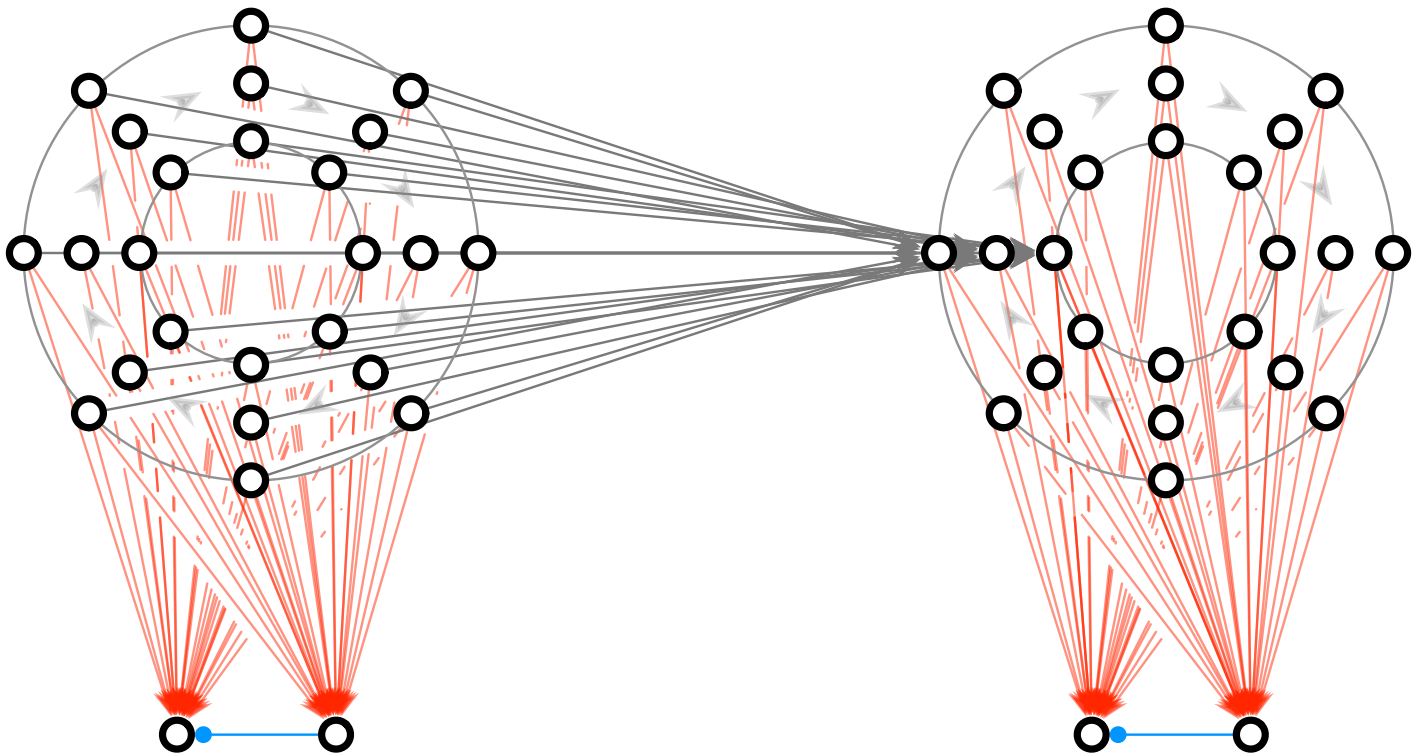




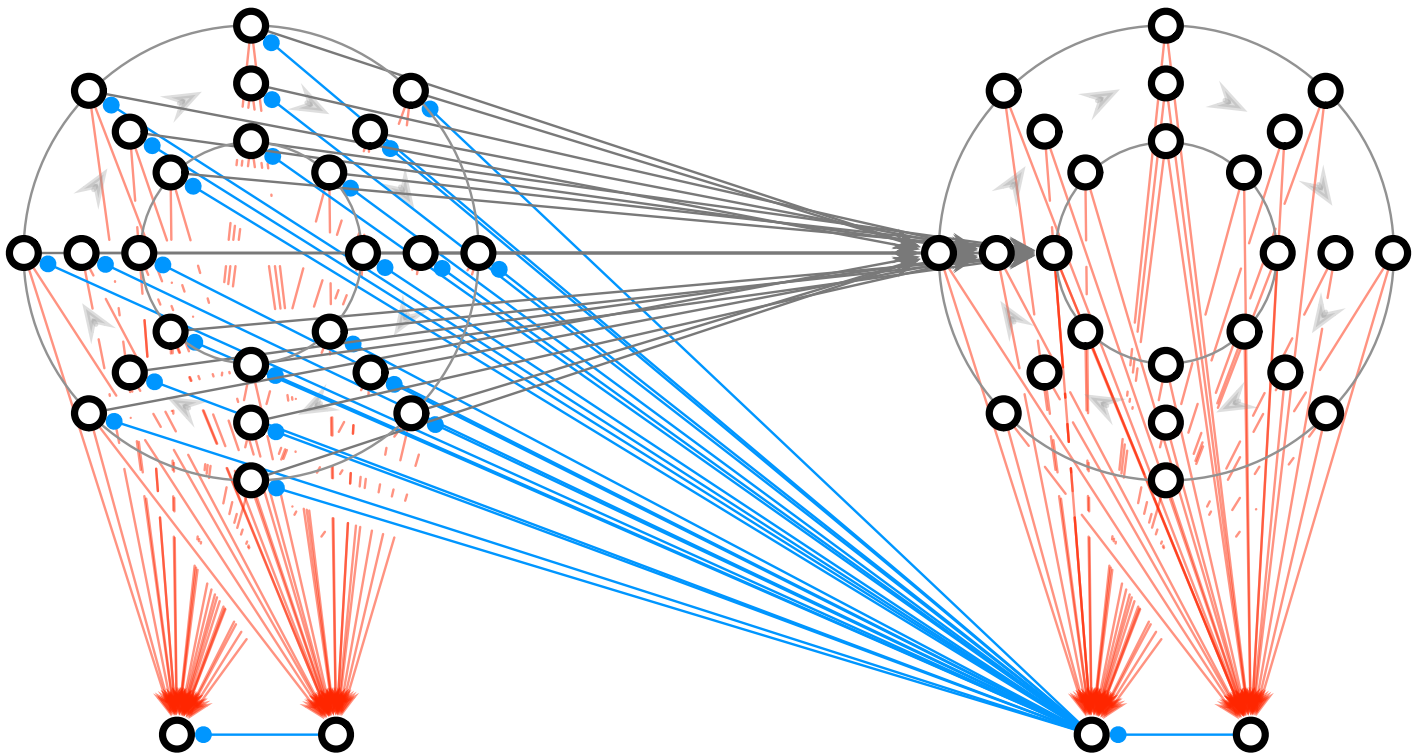
Ring Transition 1: Excitatory/Inhibitory System



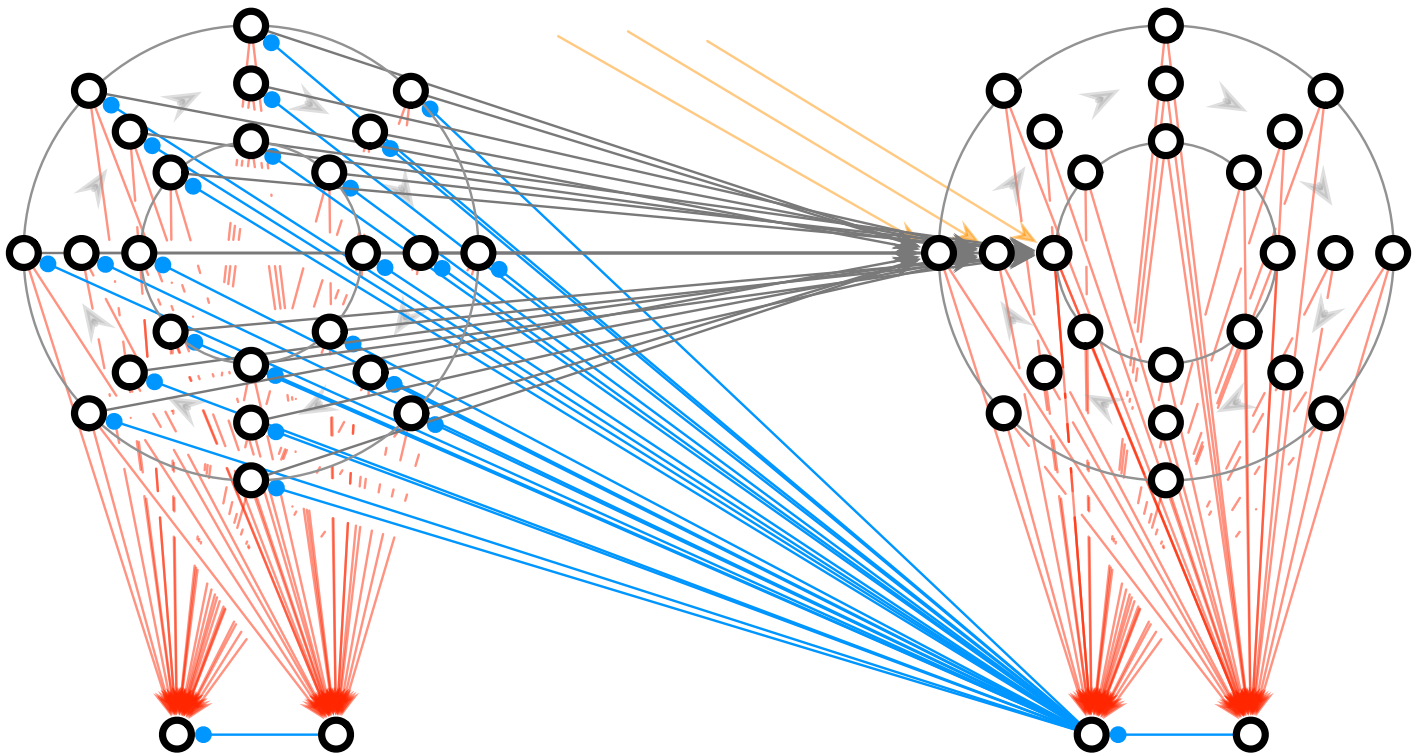
Ring Transition 1: Excitatory/Inhibitory System



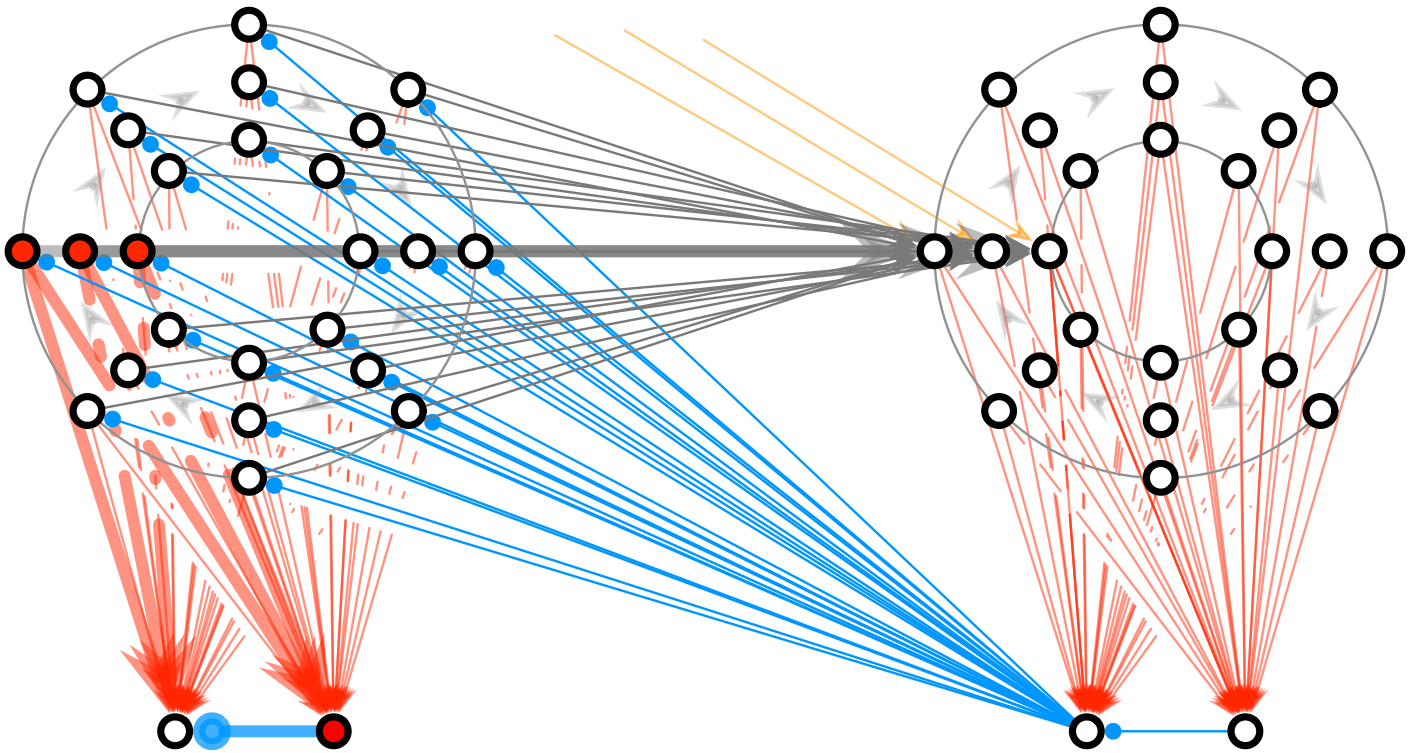
Ring Transition 1: Excitatory/Inhibitory System



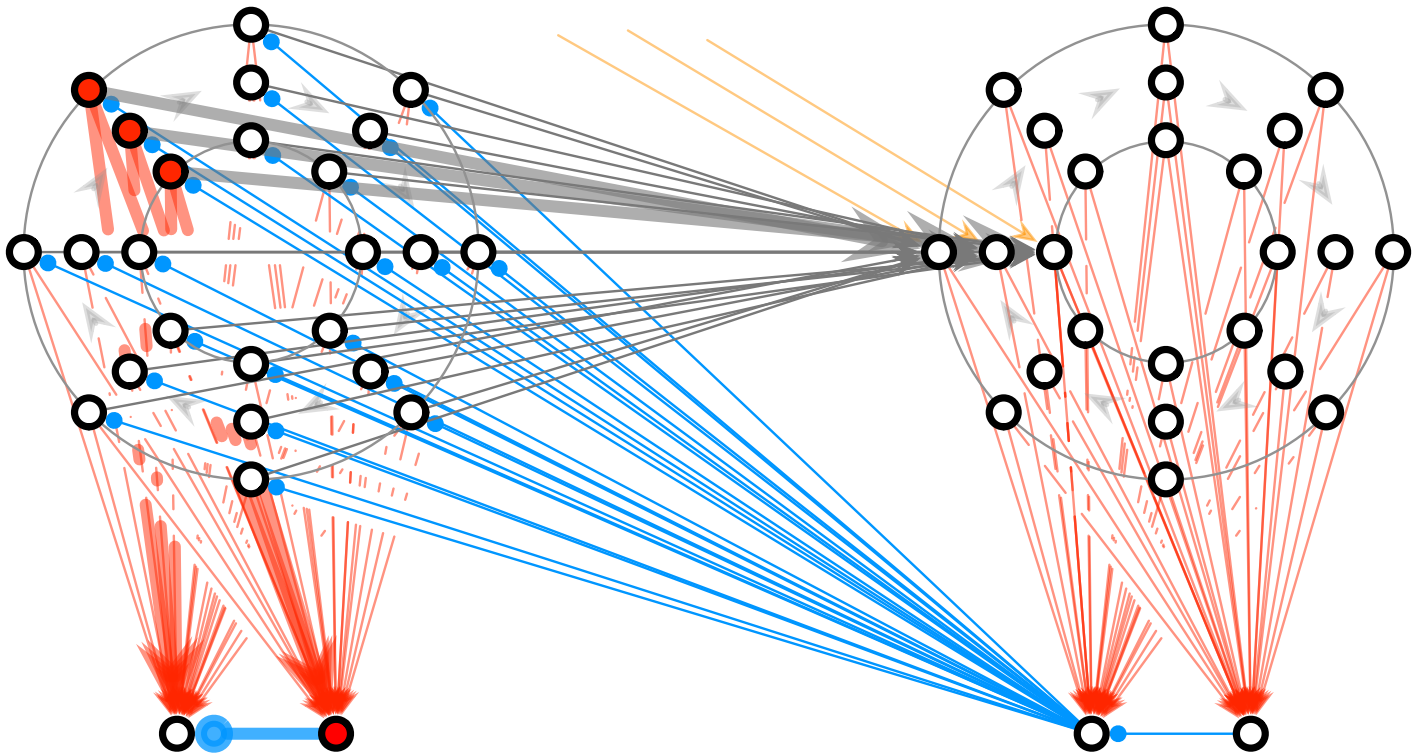
Ring Transition 1: Excitatory/Inhibitory System



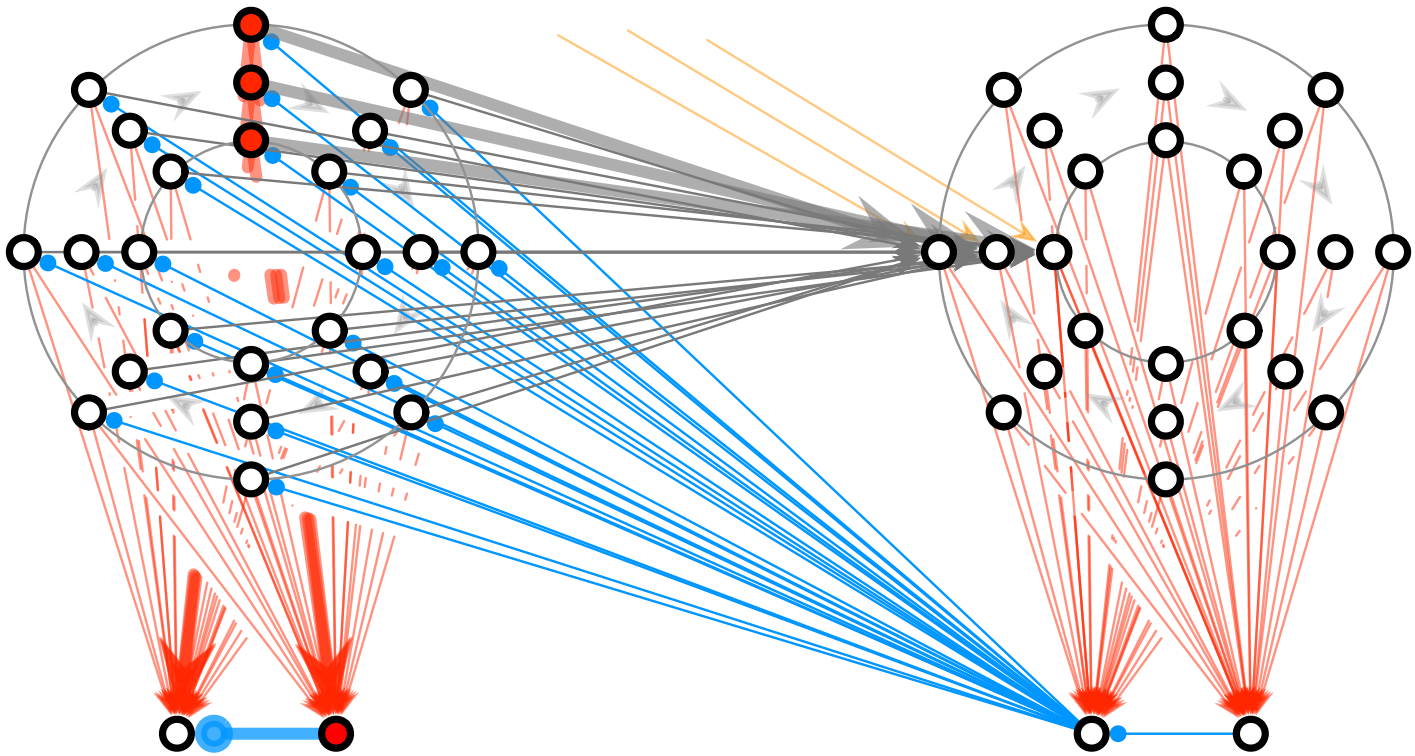
Ring Transition 1: Excitatory/Inhibitory System



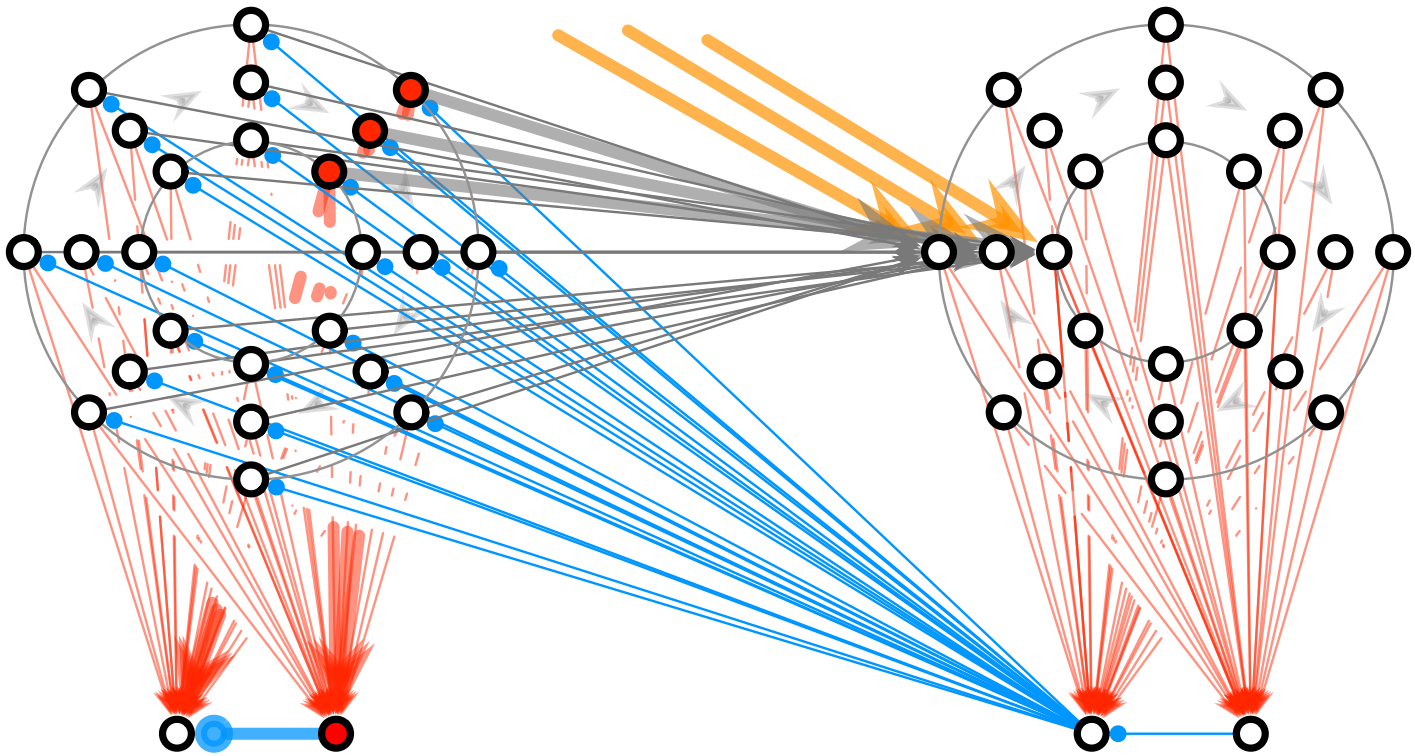
Ring Transition 1: Excitatory/Inhibitory System



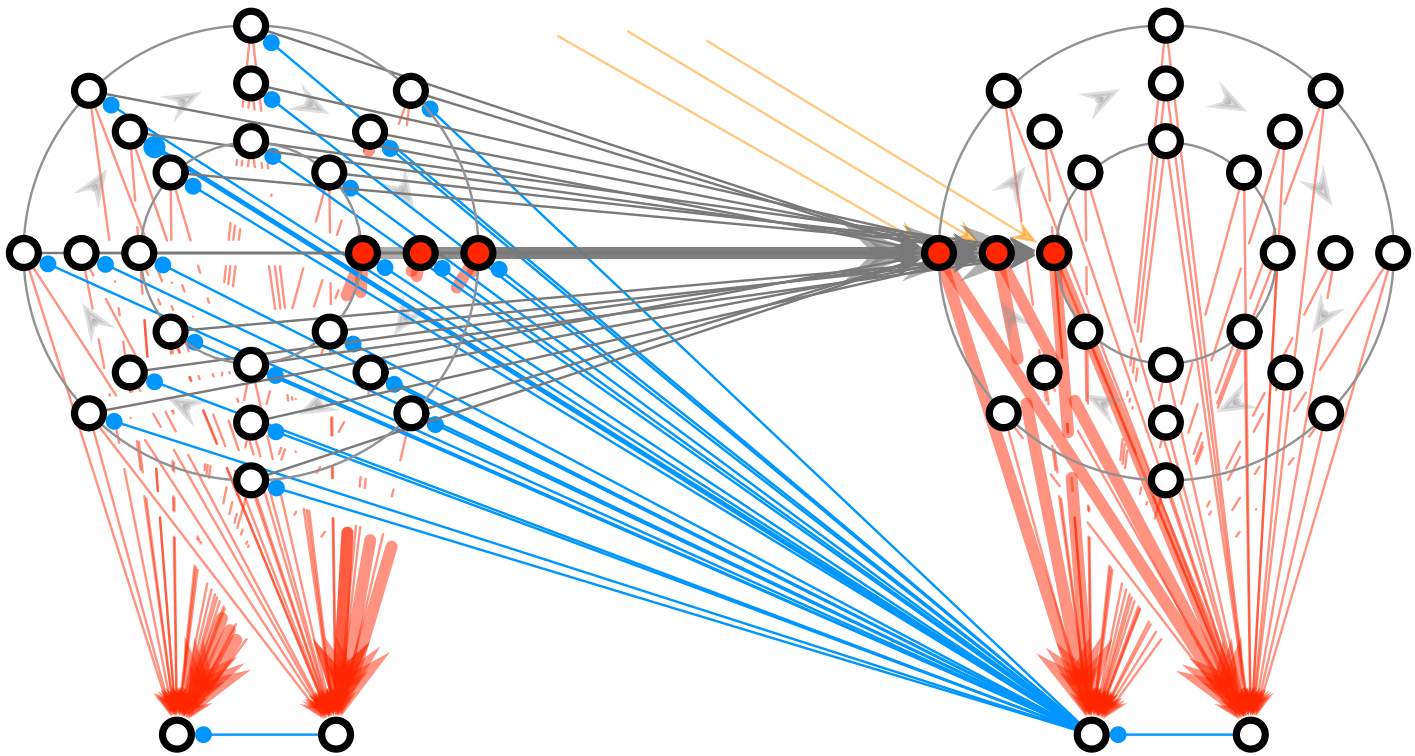
Ring Transition 1: Excitatory/Inhibitory System



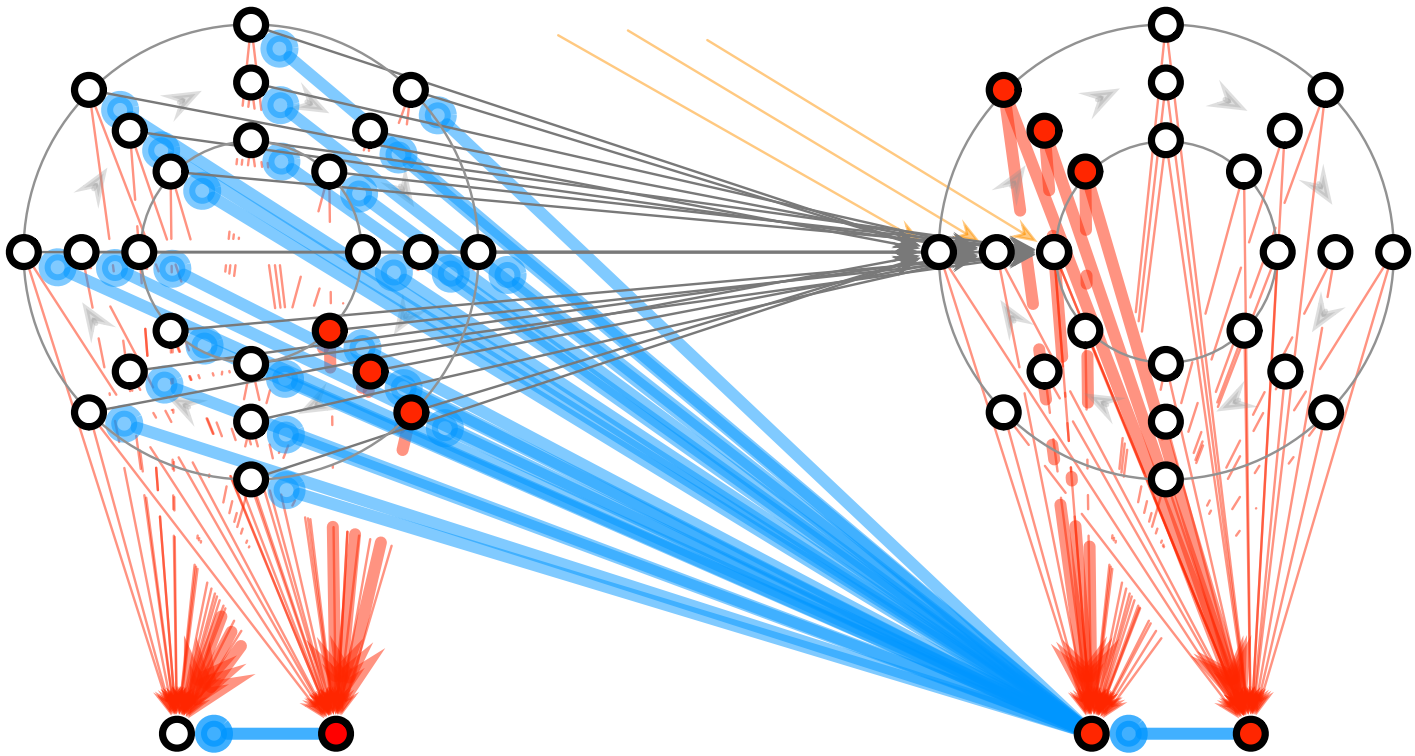
Ring Transition 1: Excitatory/Inhibitory System



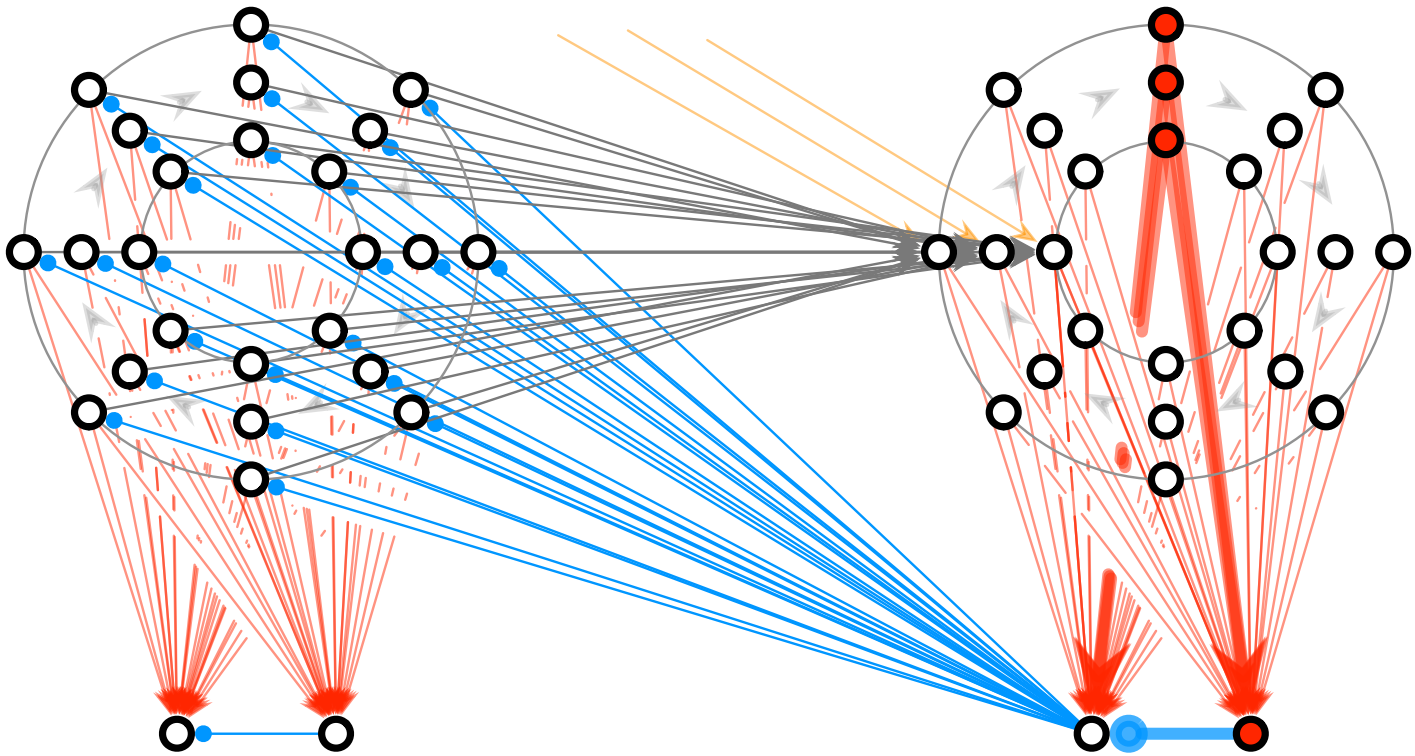
Ring Transition 1: Excitatory/Inhibitory System



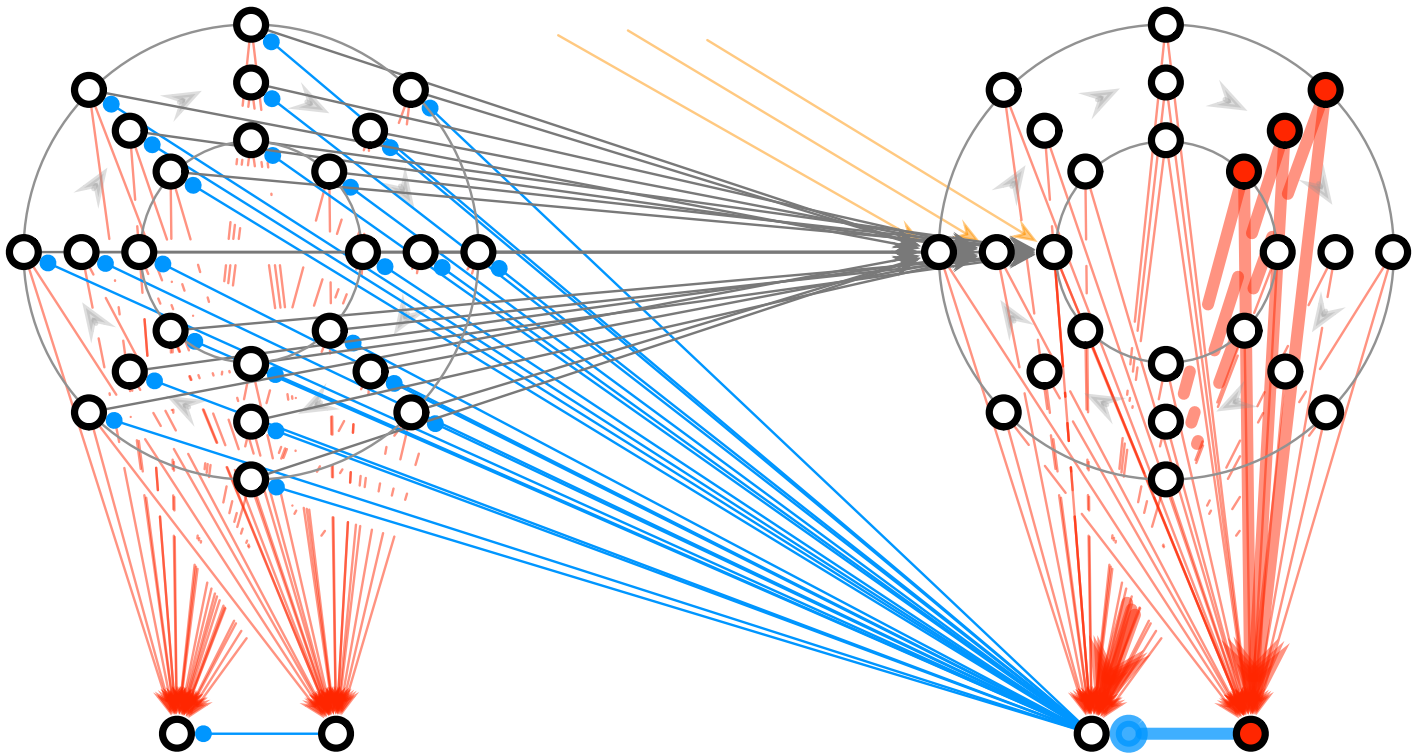
Ring Transition 1: Excitatory/Inhibitory System



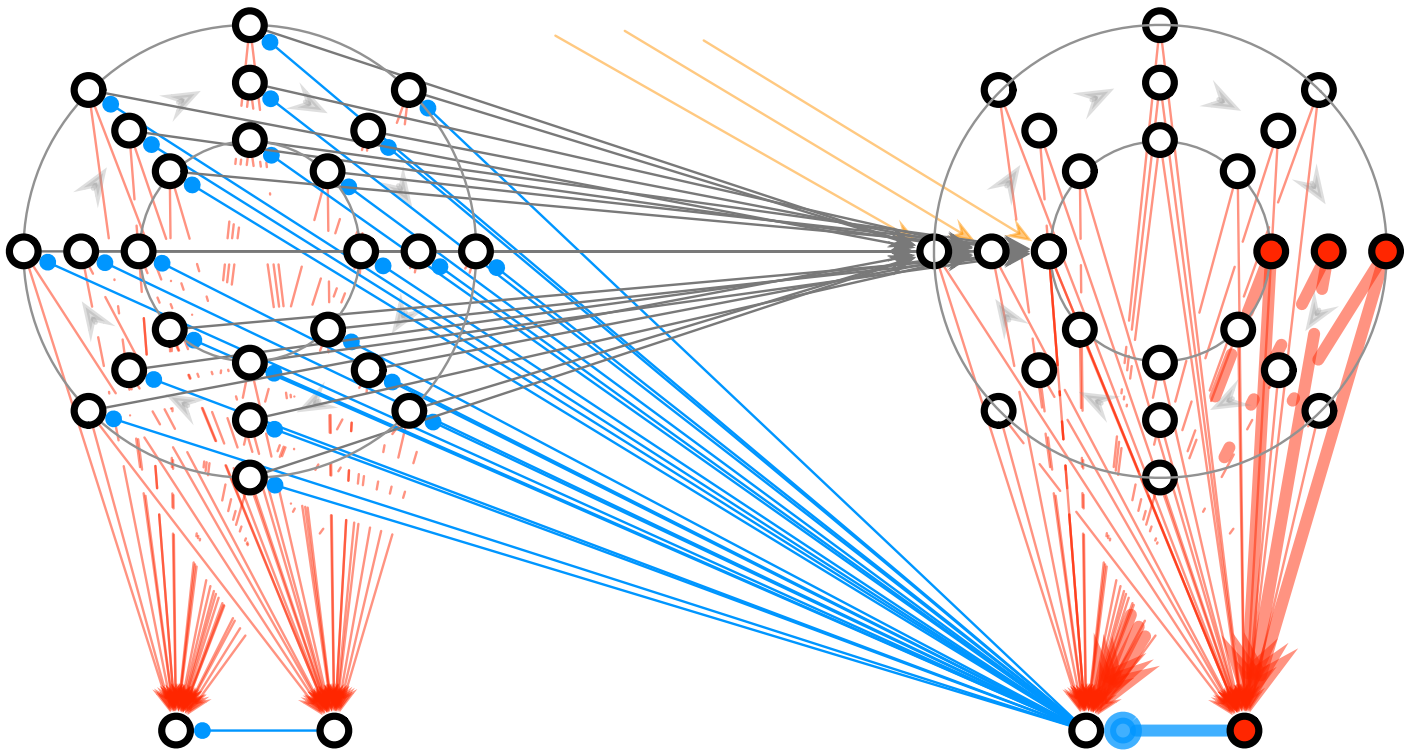
Ring Transition 1: Excitatory/Inhibitory System



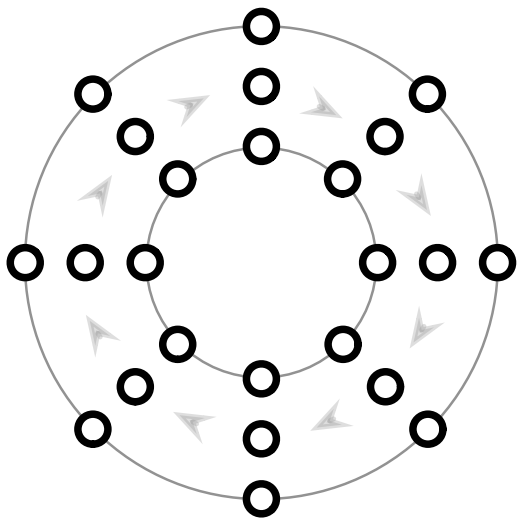
Ring Transition 1: Excitatory/Inhibitory System

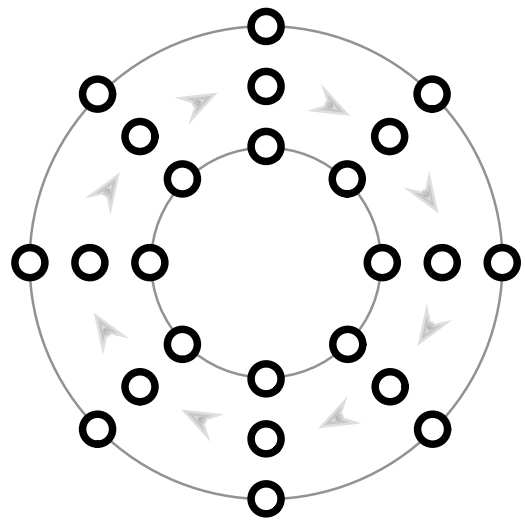


Ring Transition 1: Excitatory/Inhibitory System

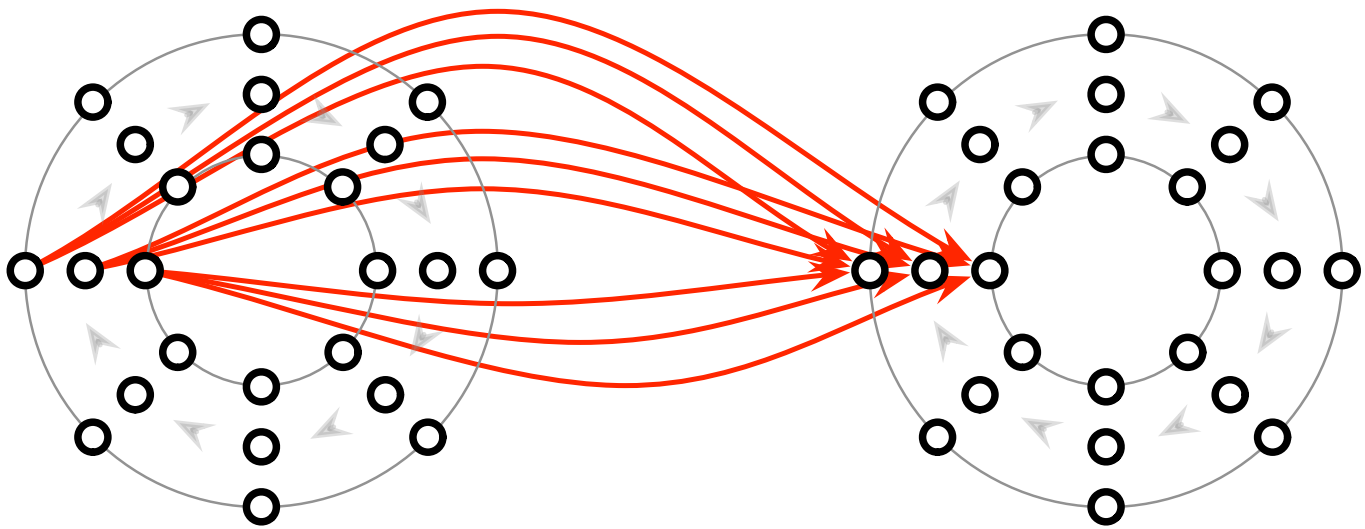


Ring Transition 2: Excitatory/Inhibitory System

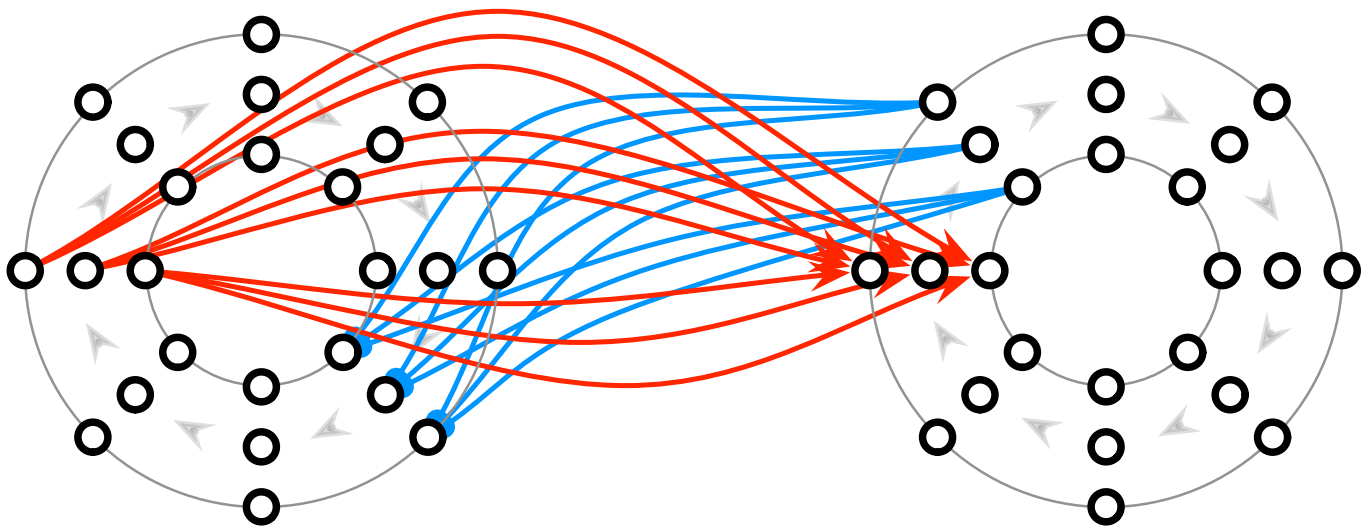




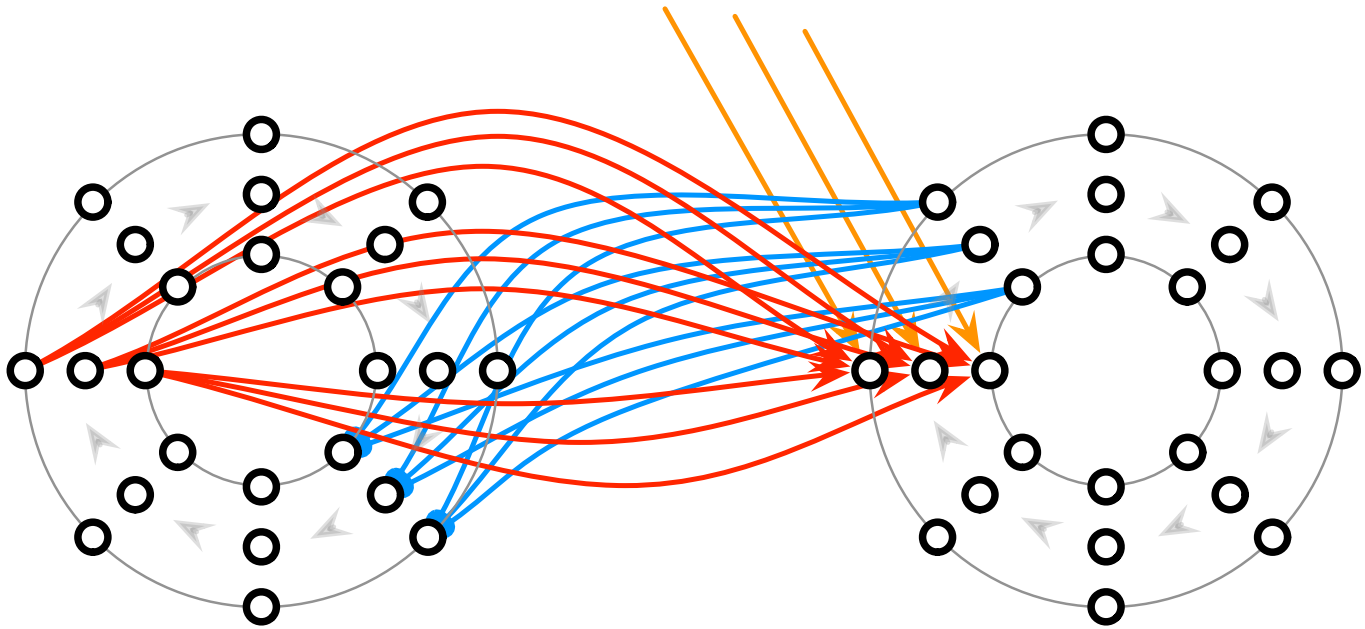
Ring Transition 2: Excitatory/Inhibitory System



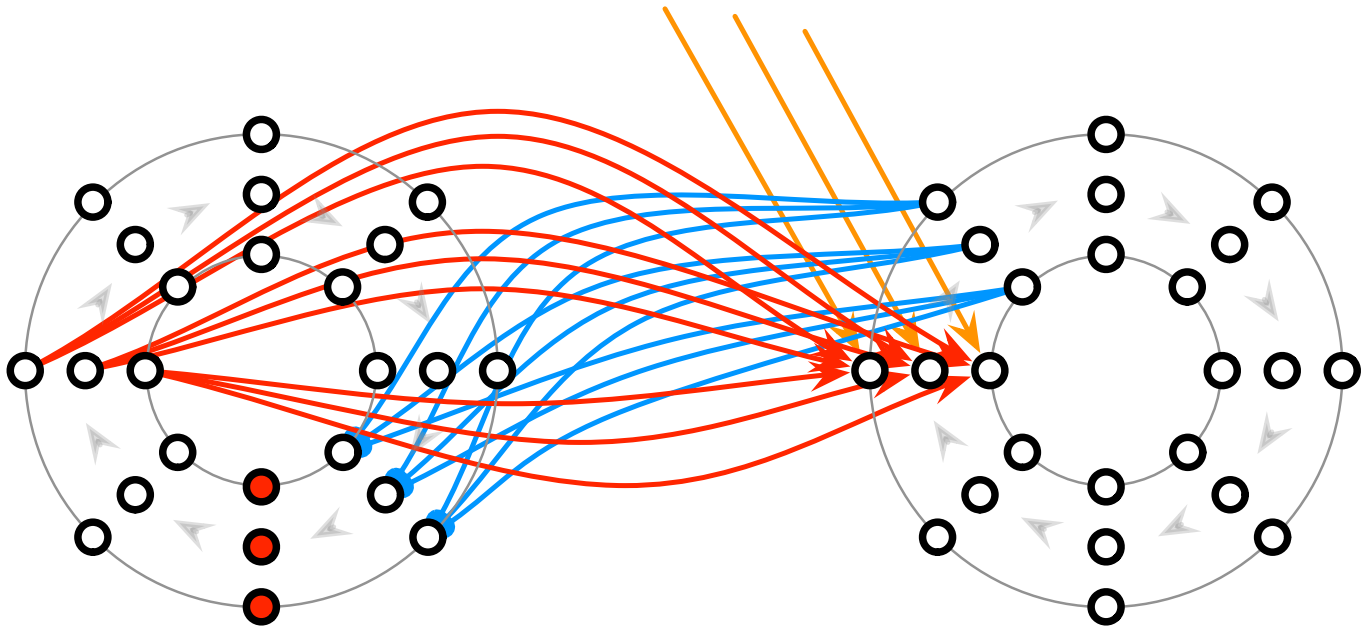
Ring Transition 2: Excitatory/Inhibitory System



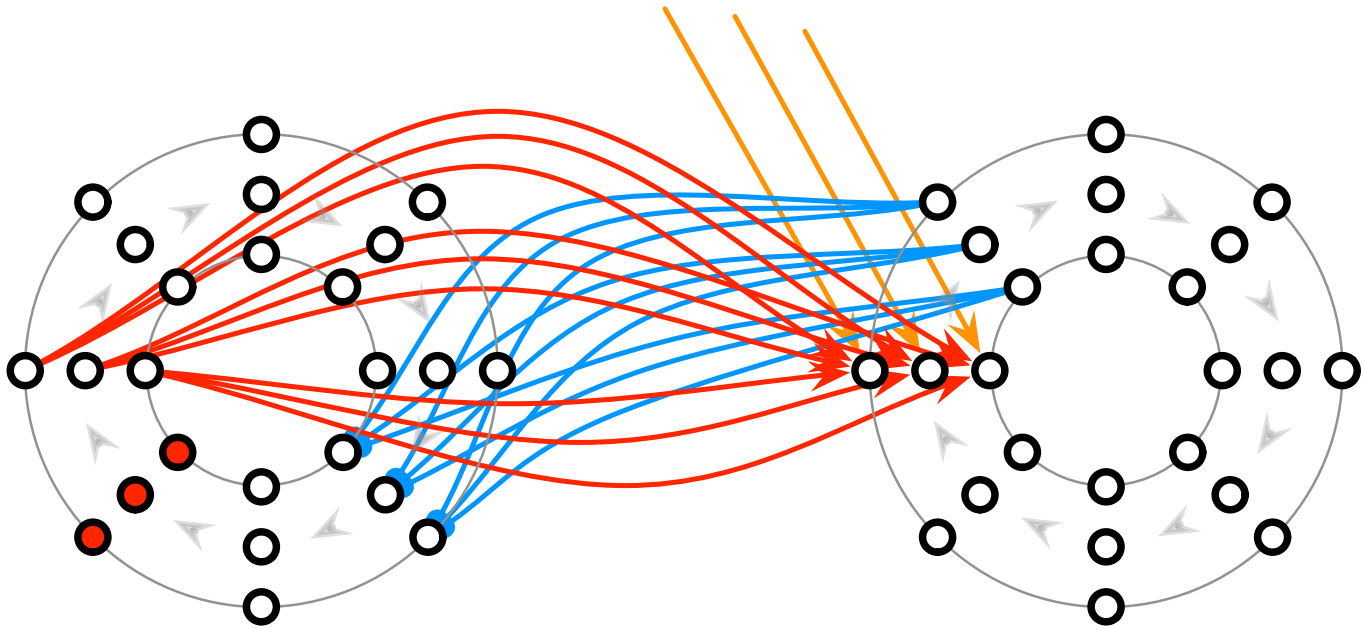
Ring Transition 2: Excitatory/Inhibitory System



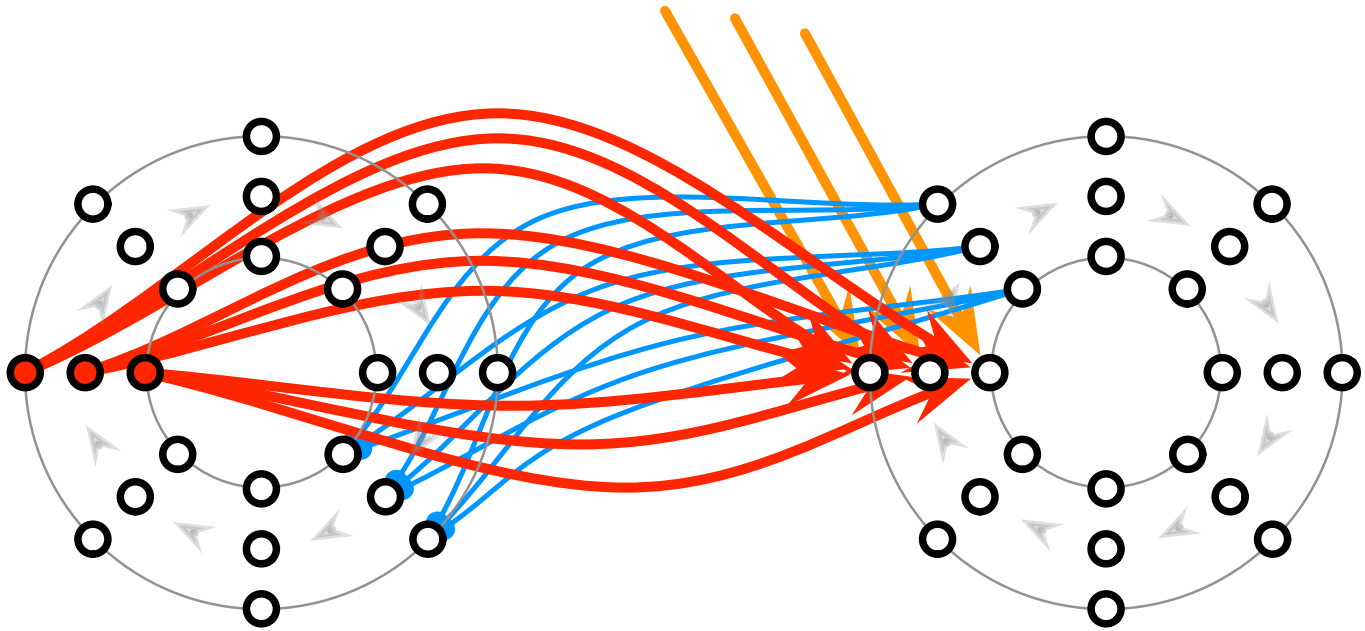
Ring Transition 2: Excitatory/Inhibitory System



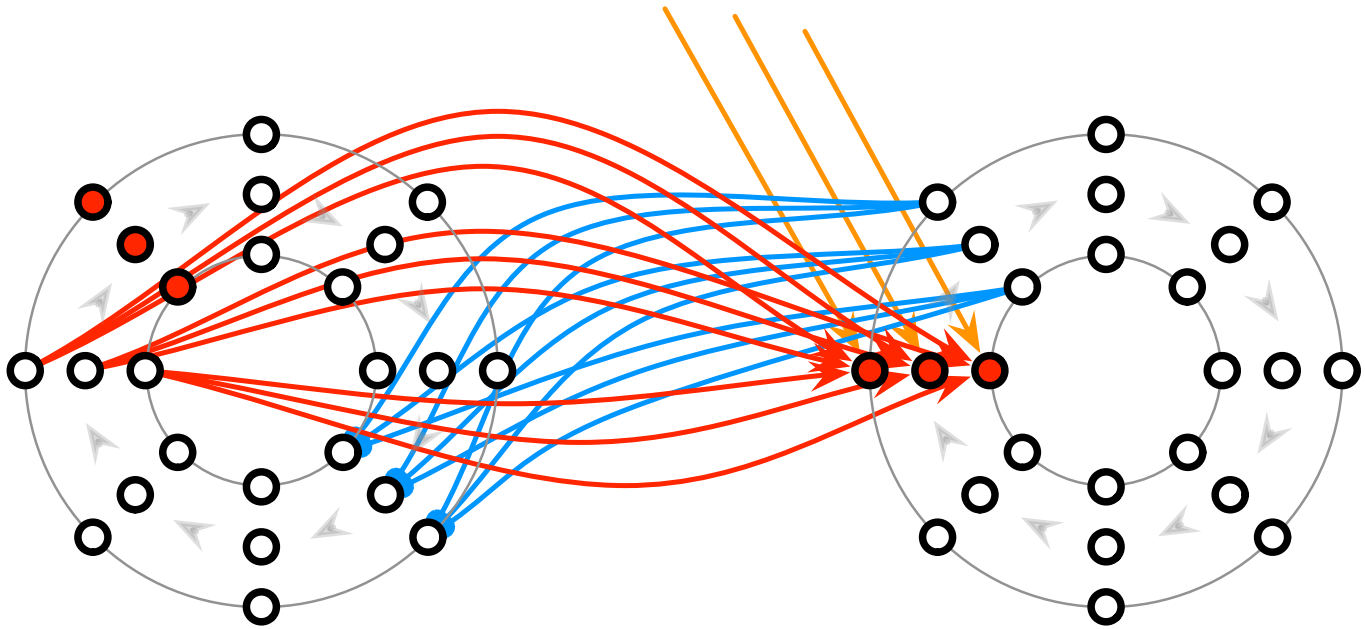
Ring Transition 2: Excitatory/Inhibitory System



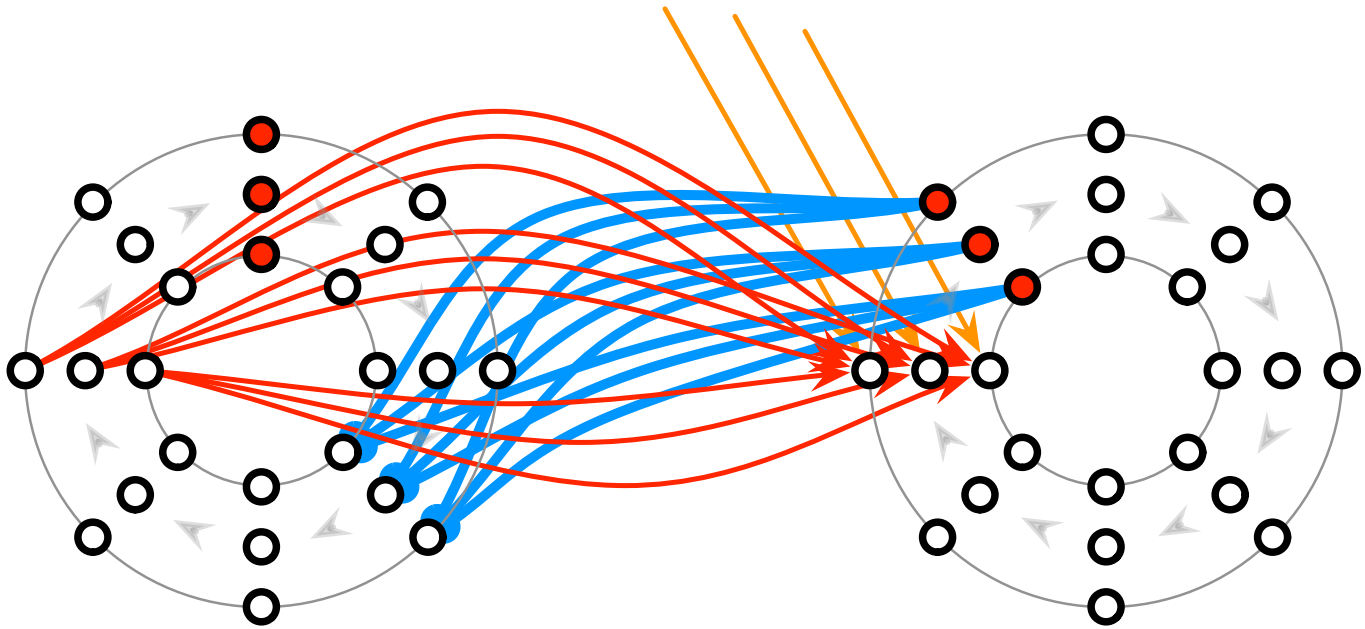
Ring Transition 2: Excitatory/Inhibitory System



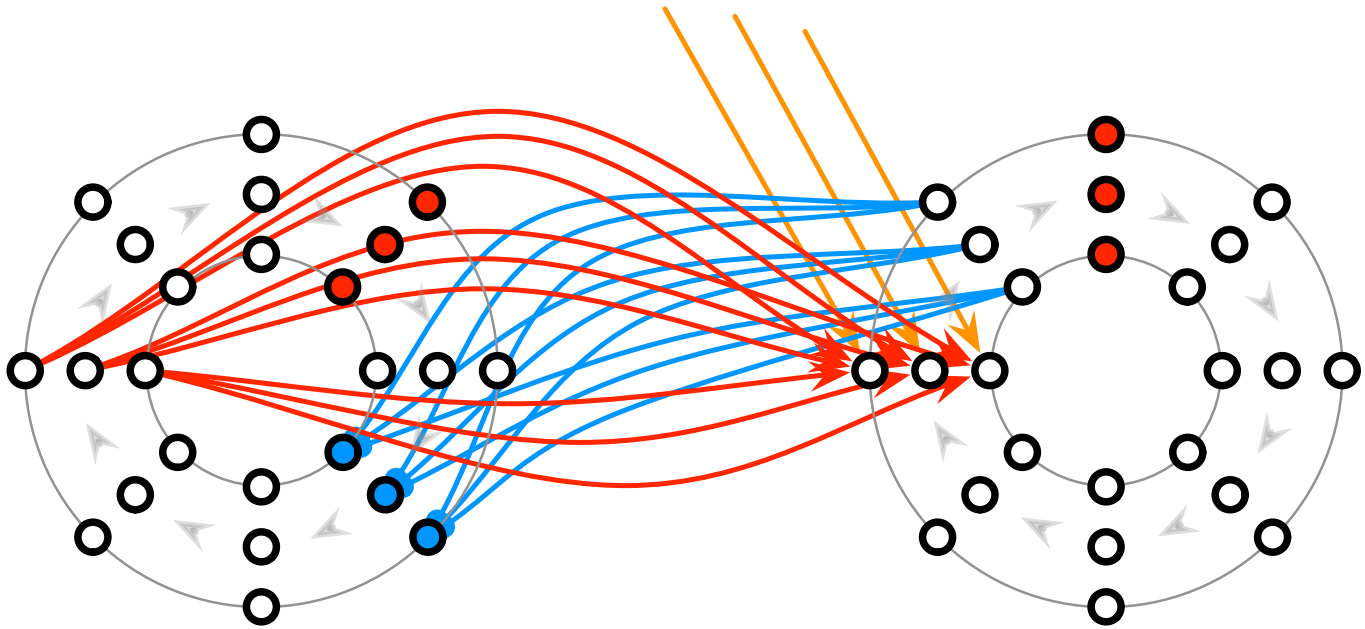
Ring Transition 2: Excitatory/Inhibitory System



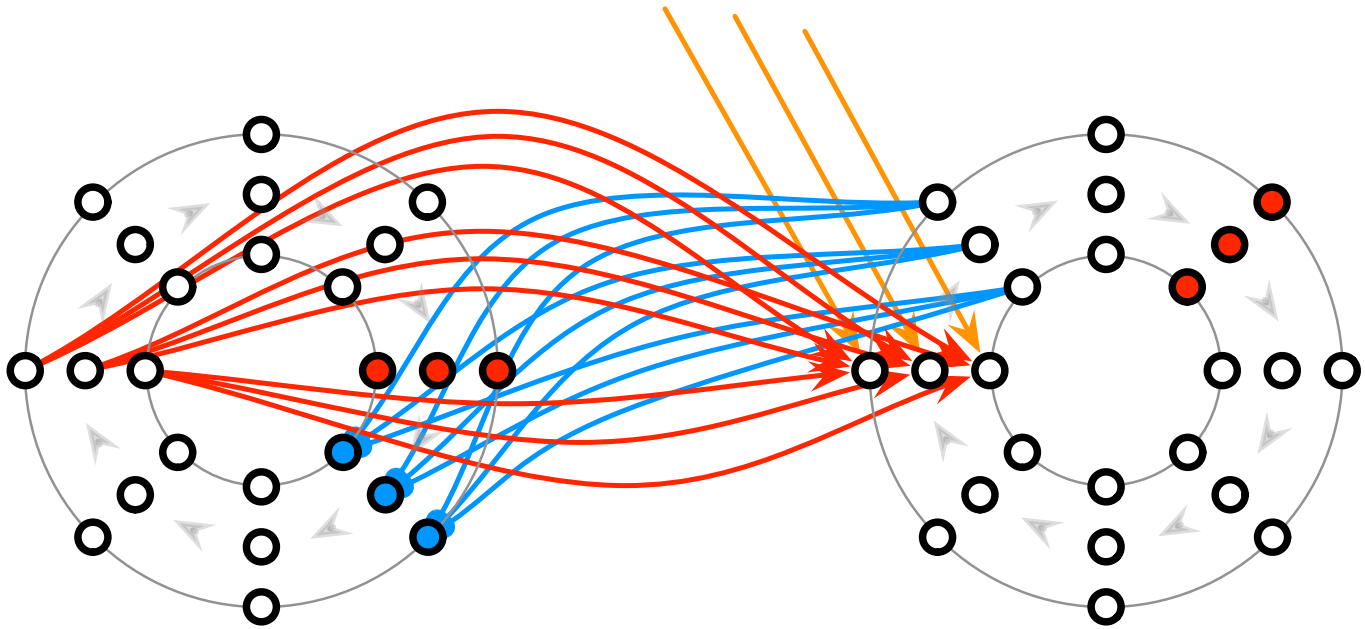
Ring Transition 2: Excitatory/Inhibitory System



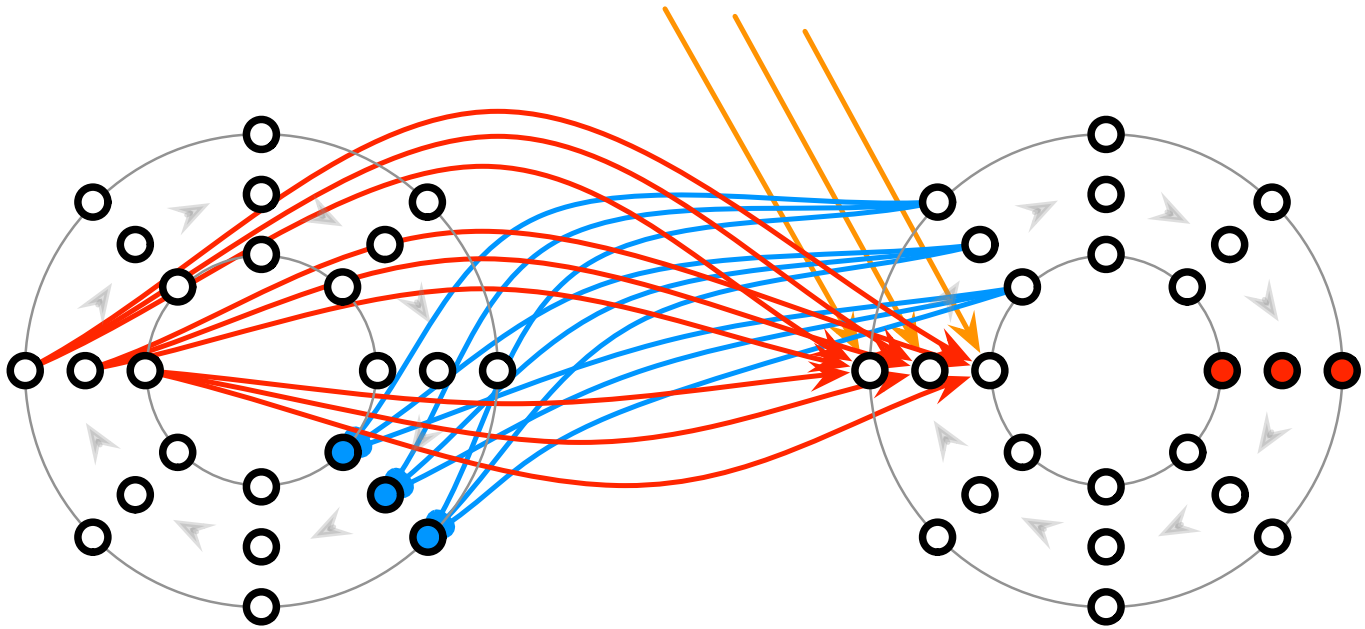
Ring Transition 2: Excitatory/Inhibitory System



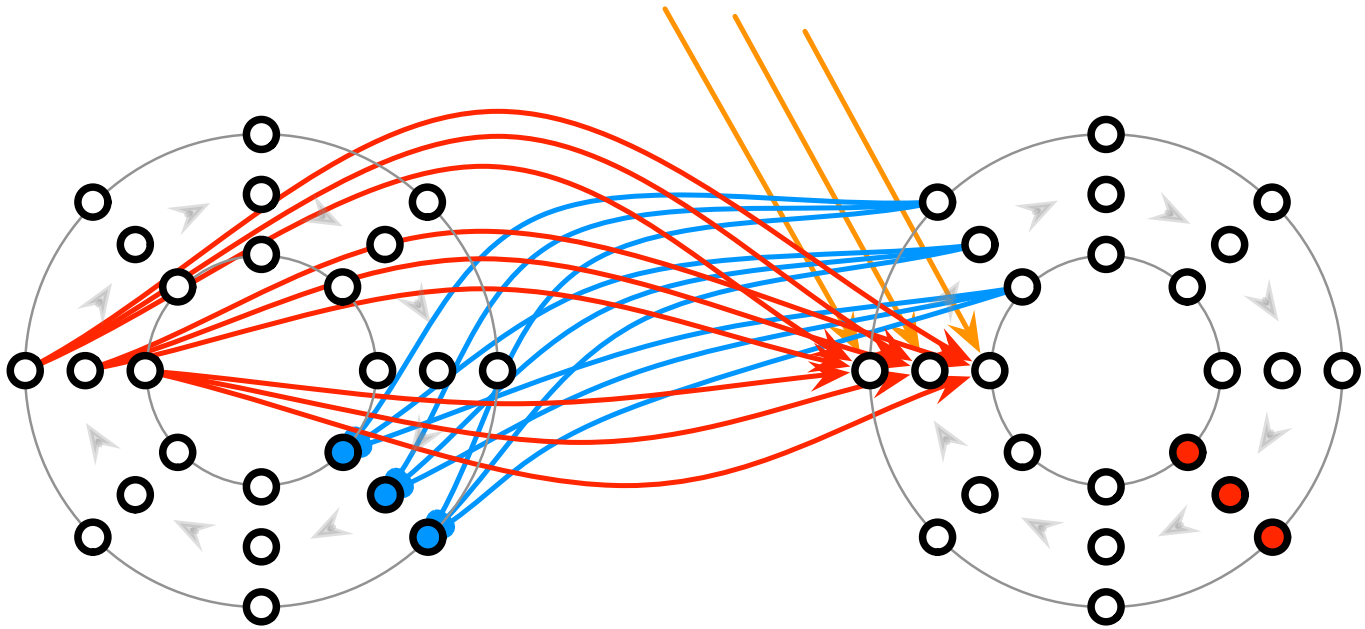
Ring Transition 2: Excitatory/Inhibitory System



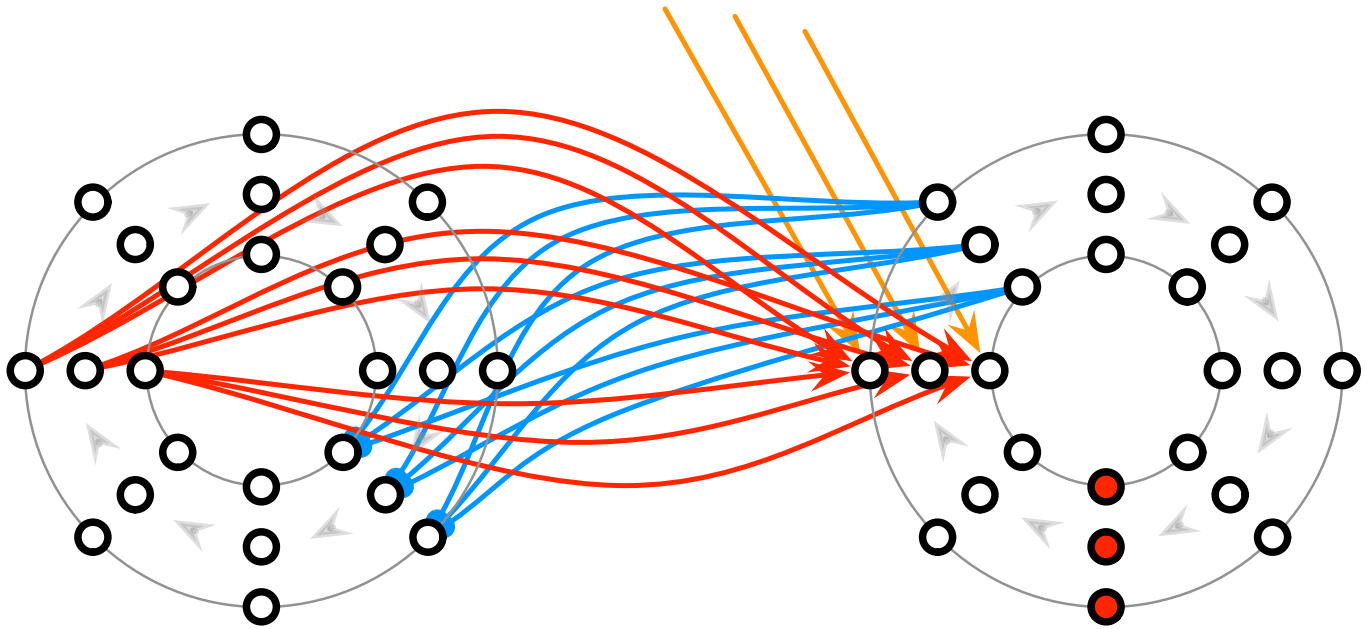
Ring Transition 2: Excitatory/Inhibitory System



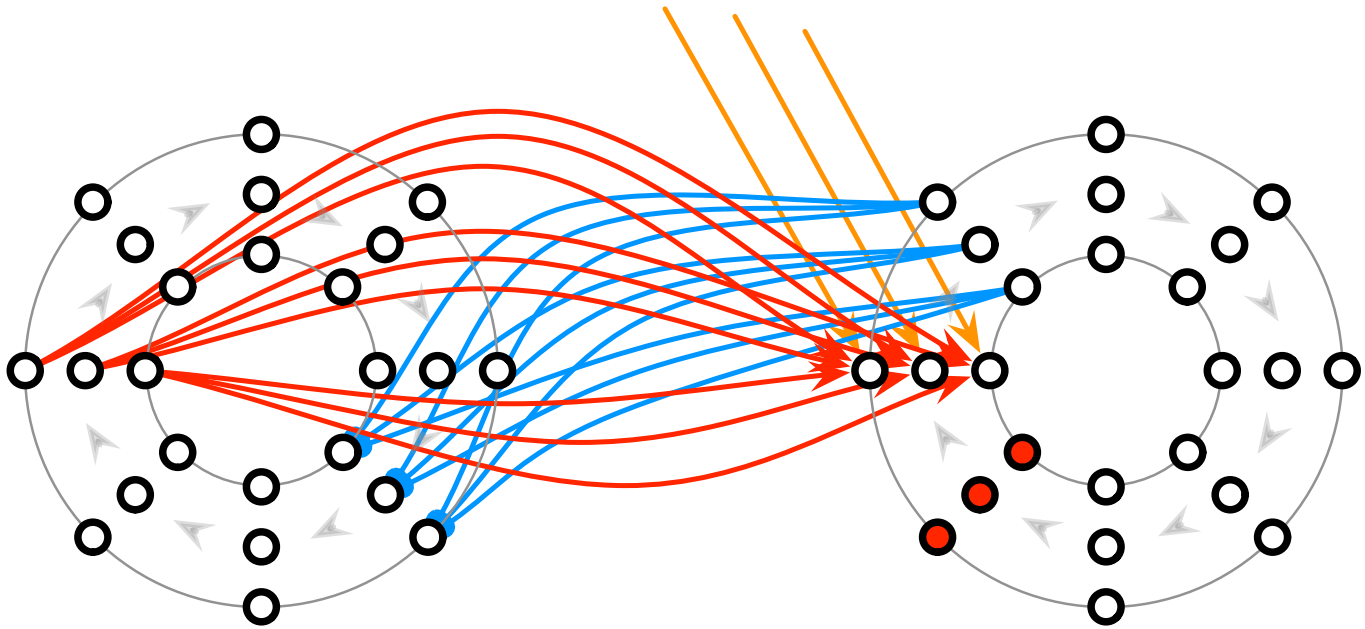
Ring Transition 2: Excitatory/Inhibitory System



Ring Transition 2: Excitatory/Inhibitory System



Ring Transition 2: Excitatory/Inhibitory System



Automata & RNNs with Synfire Rings

The construction is generic, therefore, the following results hold:

Theorem (Cabessa & Masulli 17, Cabessa et al. 17, Cabessa & Tchaptchet 18, Cabessa & Sima 19)

- *Any finite state automaton can be simulated by some Boolean neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some (noisy) Izhikevich spiking neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some Hodgkin-Huxley spiking neural network composed of synfire rings.*

Automata & RNNs with Synfire Rings

The construction is generic, therefore, the following results hold:

Theorem (Cabessa & Masulli 17, Cabessa et al. 17, Cabessa & Tchaptchet 18, Cabessa & Sima 19)

- *Any finite state automaton can be simulated by some Boolean neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some (noisy) Izhikevich spiking neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some Hodgkin-Huxley spiking neural network composed of synfire rings.*

Automata & RNNs with Synfire Rings

The construction is generic, therefore, the following results hold:

Theorem (Cabessa & Masulli 17, Cabessa et al. 17, Cabessa & Tchaptchet 18, Cabessa & Sima 19)

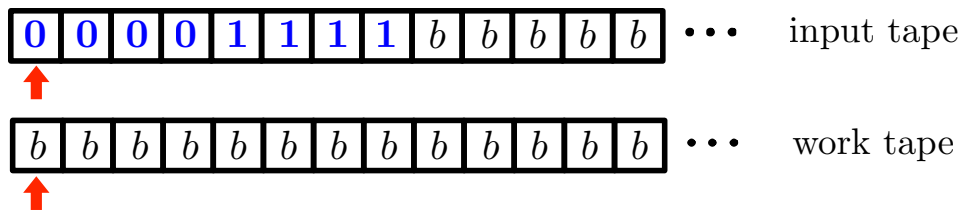
- *Any finite state automaton can be simulated by some Boolean neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some (noisy) Izhikevich spiking neural network composed of synfire rings.*
- *Any finite state automaton can be simulated by some Hodgkin-Huxley spiking neural network composed of synfire rings.*

Simulations

Play movies...

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

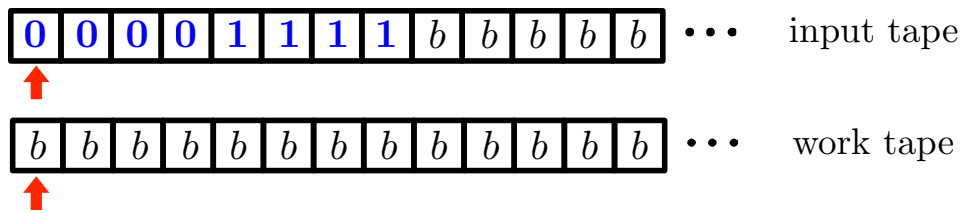


Program *current state:* q_{in}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

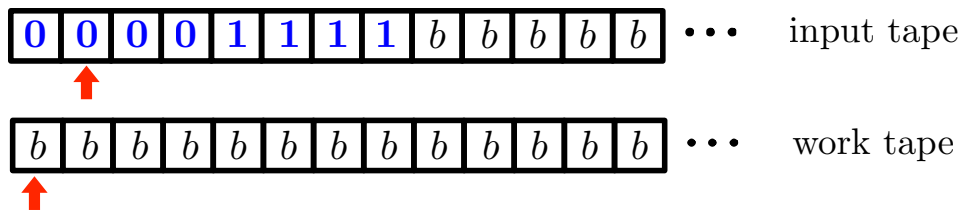


Program *current state:* q_{in}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

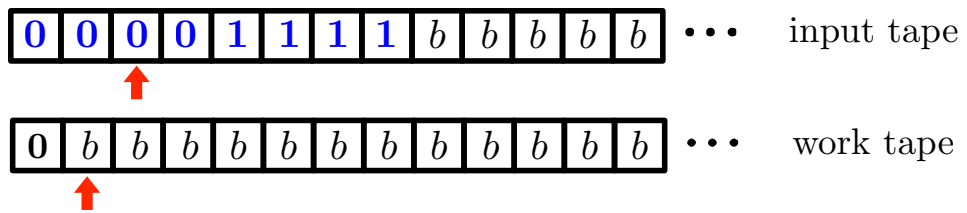


Program *current state:* q_0

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)					
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$					
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$					
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$		
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$		
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)		
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$		
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$		
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$		
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$		
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$		

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

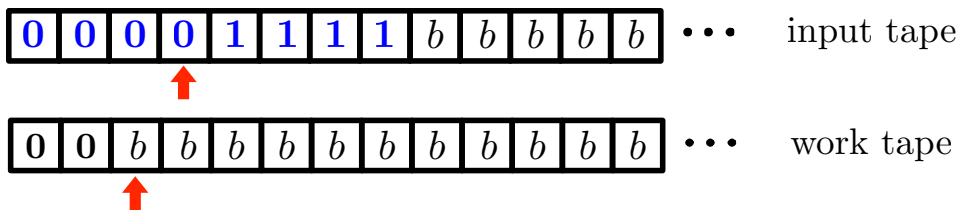


Program *current state:* q_{0bis}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

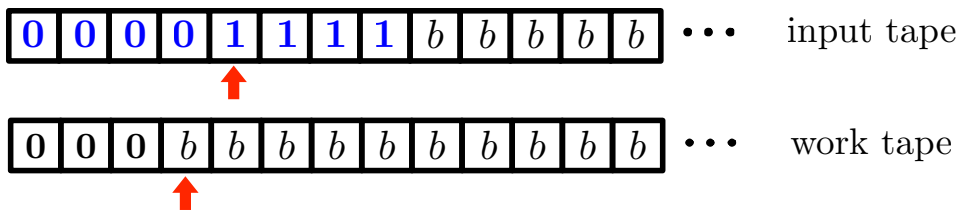


Program *current state:* q_0

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

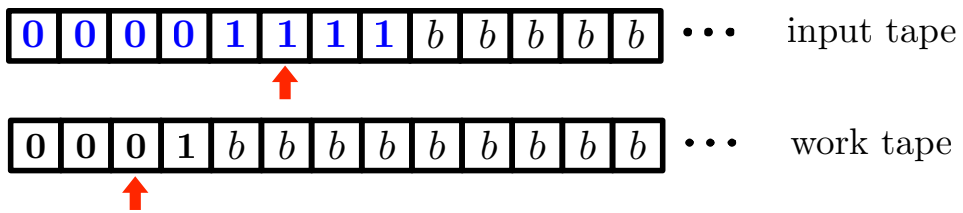


Program *current state:* q_{0bis}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

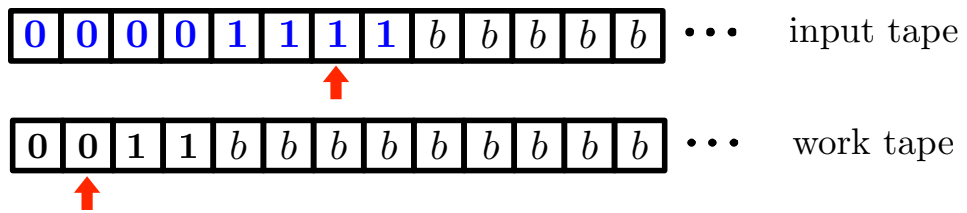


Program *current state:* q_1

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

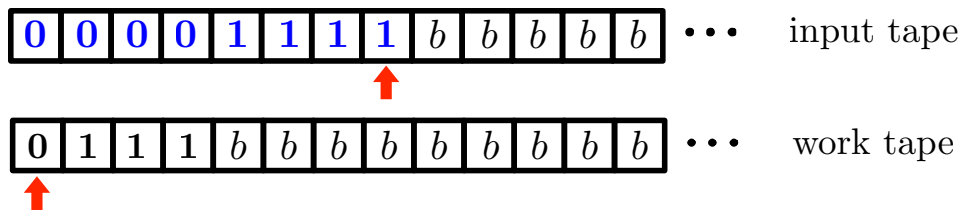


Program *current state:* q_{1bis}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

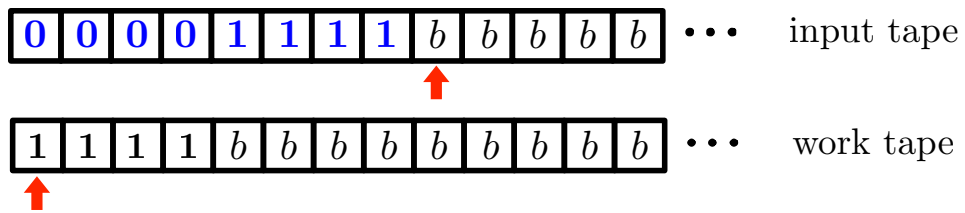


Program *current state:* q_1

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

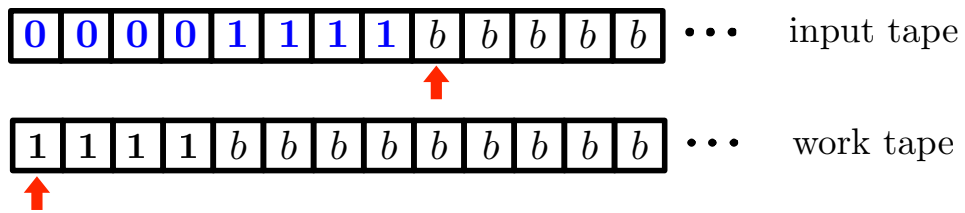


Program *current state:* q_{1bis}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

Simulation of Bounded-Space Turing Machines

- We move to the simulation of bounded-space Turing machines.
- Turing machine recognizing the non regular and non context-free language $\{0^n 1^n : n \geq 0\}$

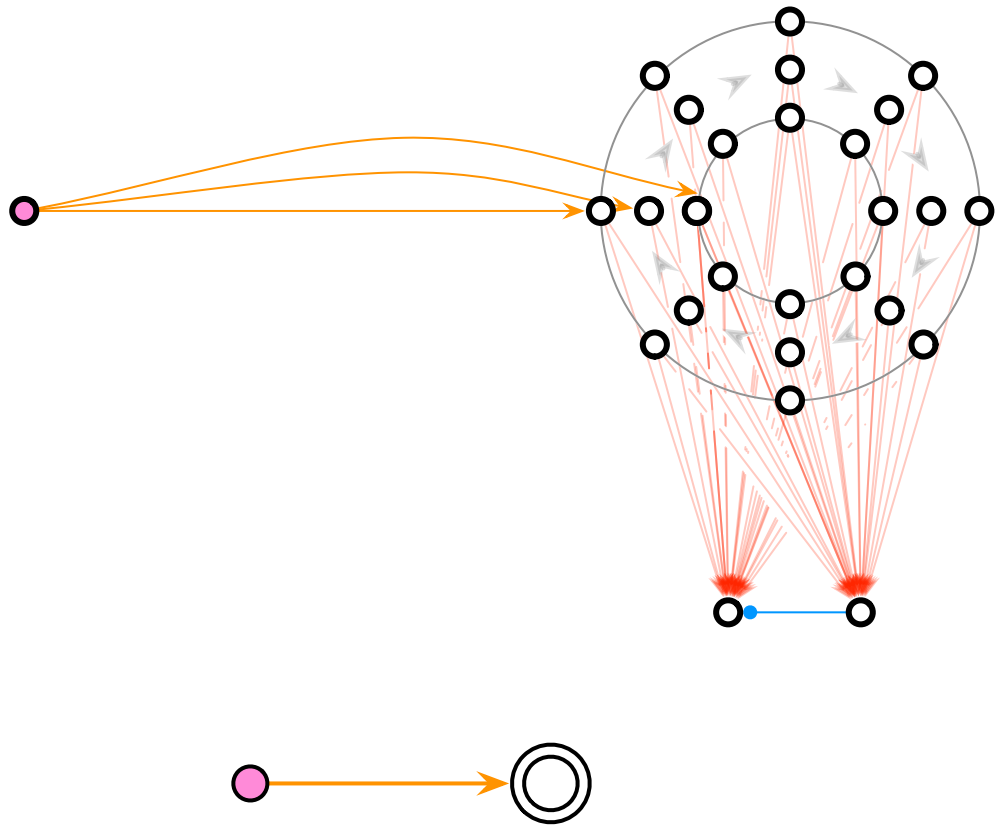


Program *current state:* q_{acc}

(q_{in}, b, b)	$\mapsto$	(q_{acc}, b, b, S, S)			
$(q_{in}, 0, b)$	$\mapsto$	$(q_0, 0, b, R, S)$			
$(q_{in}, 1, b)$	$\mapsto$	$(q_{rej}, 1, b, S, S)$			
$(q_0, 0, b)$	$\mapsto$	$(q_{0bis}, 0, 0, R, R)$	$(q_{0bis}, 0, b)$	$\mapsto$	$(q_0, 0, 0, R, R)$
$(q_0, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$	$(q_{0bis}, 1, b)$	$\mapsto$	$(q_1, 1, 1, R, L)$
(q_0, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)	(q_{0bis}, b, b)	$\mapsto$	(q_{rej}, b, b, S, S)
$(q_1, 1, 0)$	$\mapsto$	$(q_{1bis}, 1, 1, R, L)$	$(q_{1bis}, 1, 0)$	$\mapsto$	$(q_1, 1, 1, R, L)$
$(q_1, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$	$(q_{1bis}, b, 1)$	$\mapsto$	$(q_{acc}, b, 1, S, S)$
$(q_1, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$	$(q_{1bis}, b, 0)$	$\mapsto$	$(q_{rej}, b, 0, S, S)$
$(q_1, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$	$(q_{1bis}, 1, 1)$	$\mapsto$	$(q_{rej}, 1, 1, S, S)$
$(q_1, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$	$(q_{1bis}, 0, 1)$	$\mapsto$	$(q_{rej}, 0, 1, S, S)$

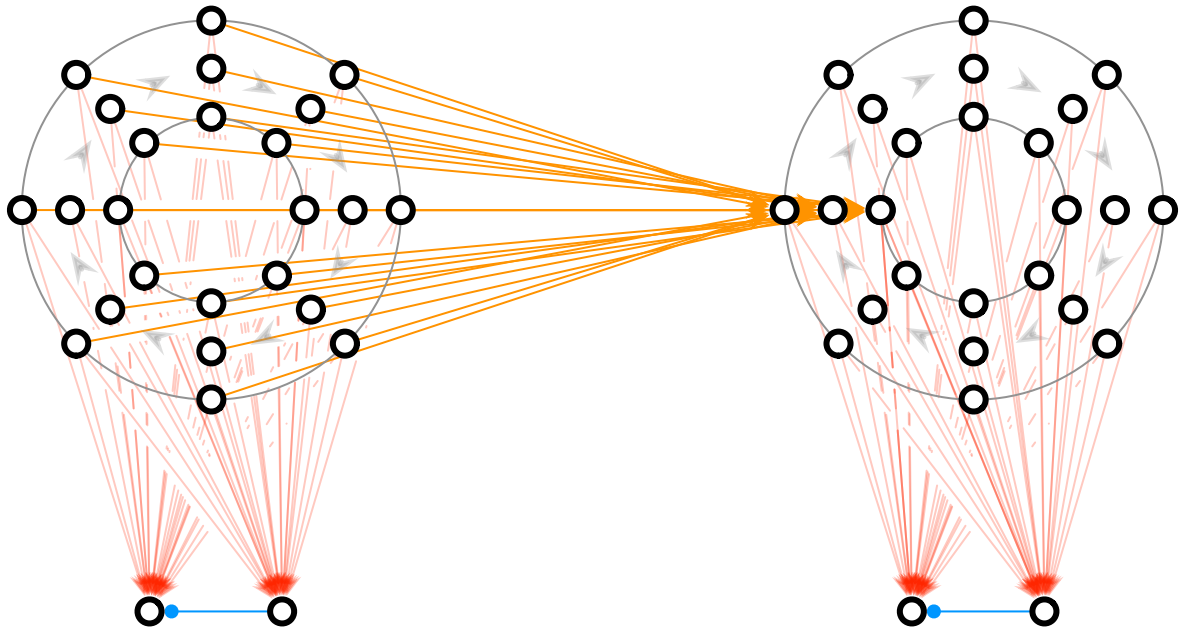
Fibres of connections & Inhibitory System

- cell to ring excitatory



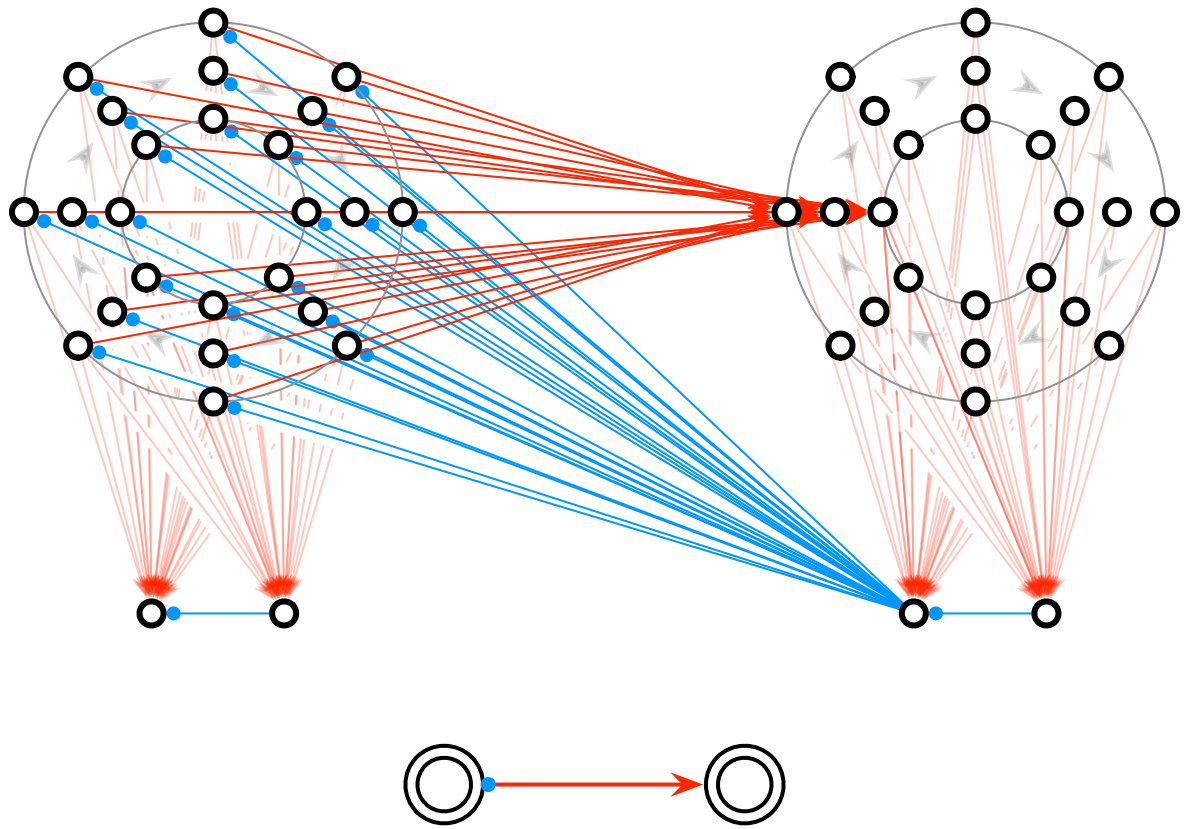
Fibres of connections & Inhibitory System

- constant excitatory



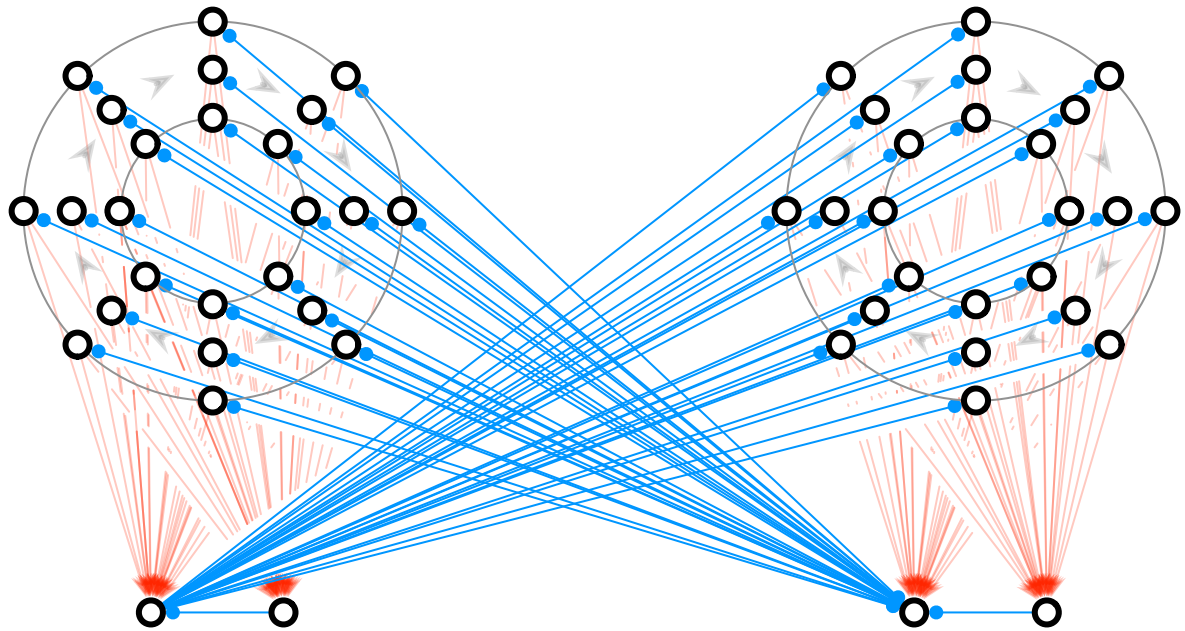
Fibres of connections & Inhibitory System

- constant excitatory / one-shot inhibitory

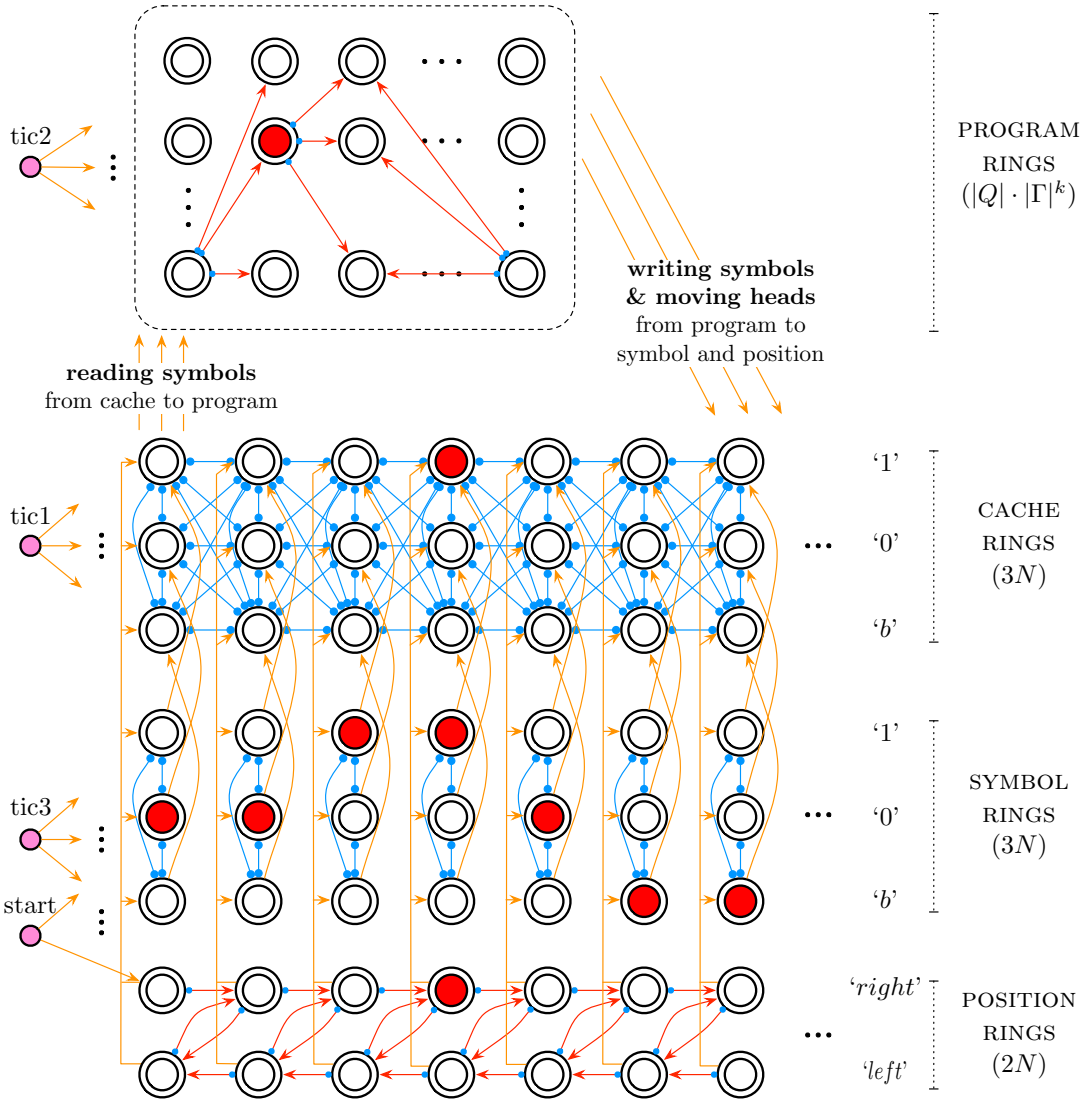


Fibres of connections & Inhibitory System

- one-shot inhibitory / one-shot inhibitory

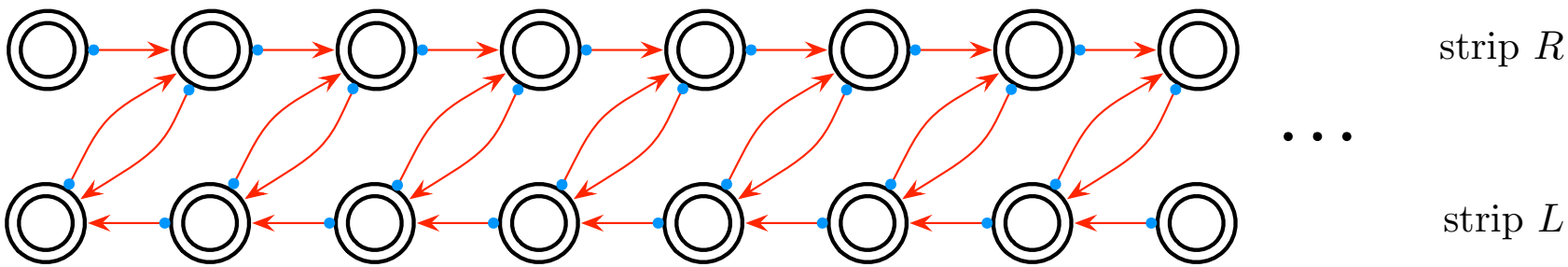


General Architecture



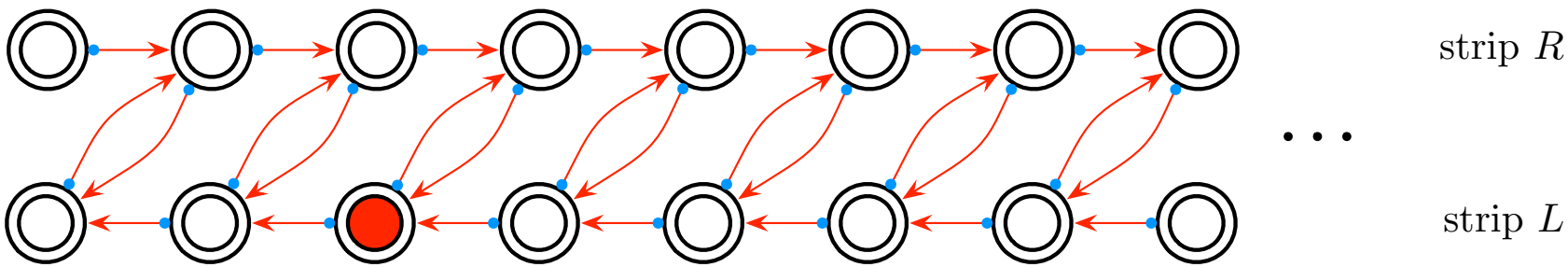
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



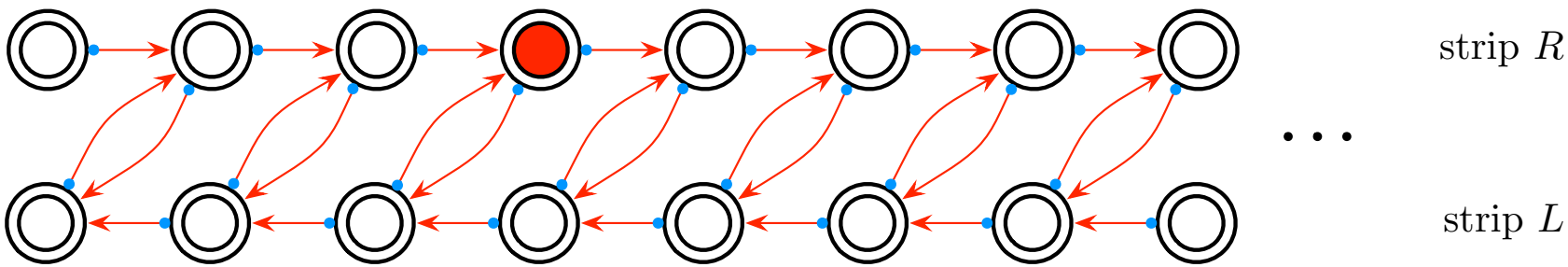
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



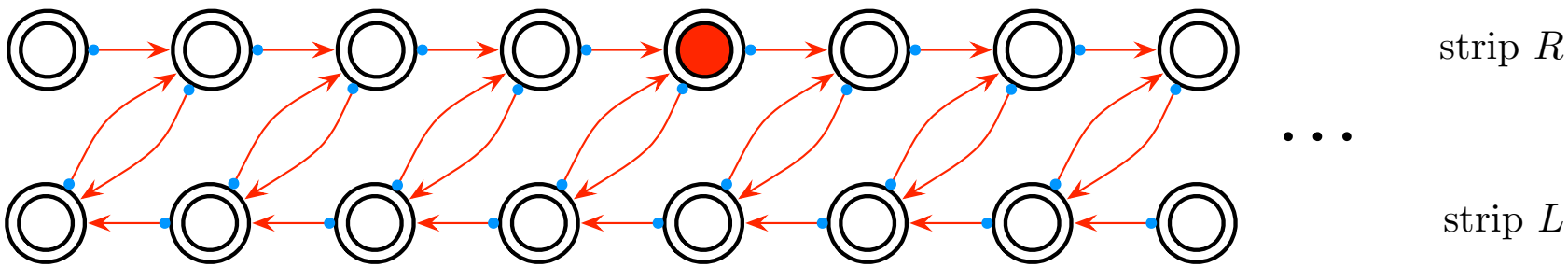
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



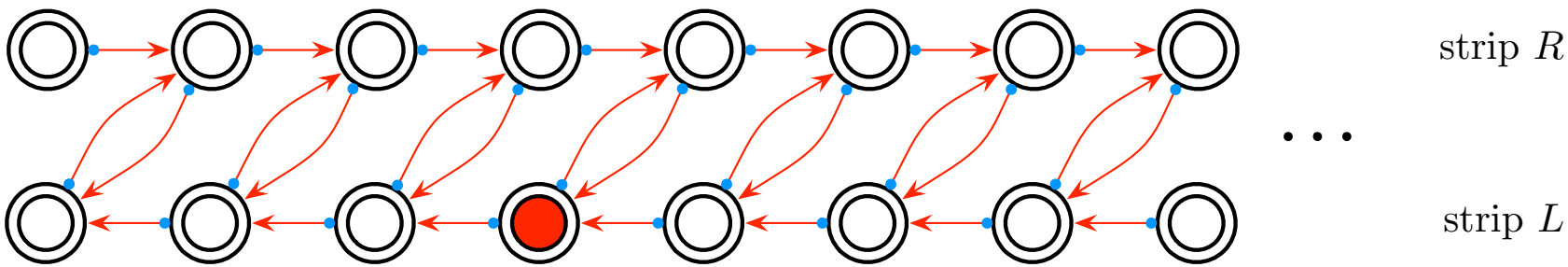
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



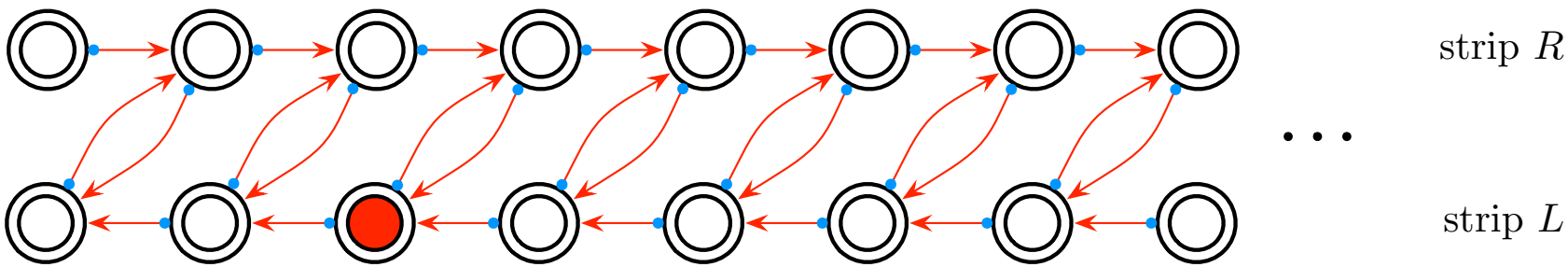
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



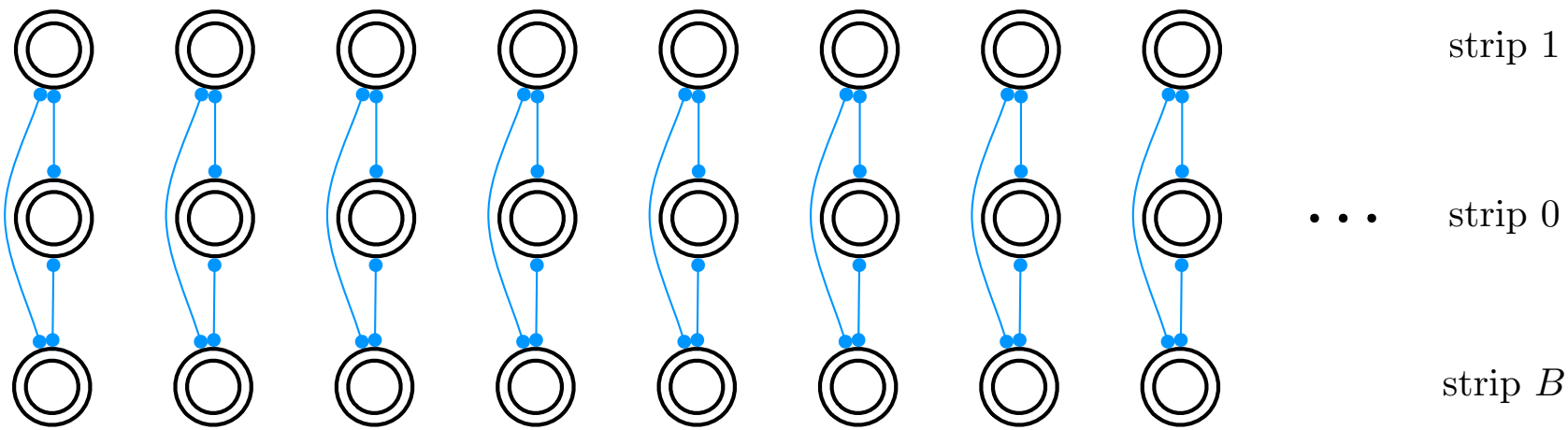
Position Rings

- Used to store the current position of the head.
- Composed of a "left" and a "right" strip.
- excitatory/inhibitory connections.



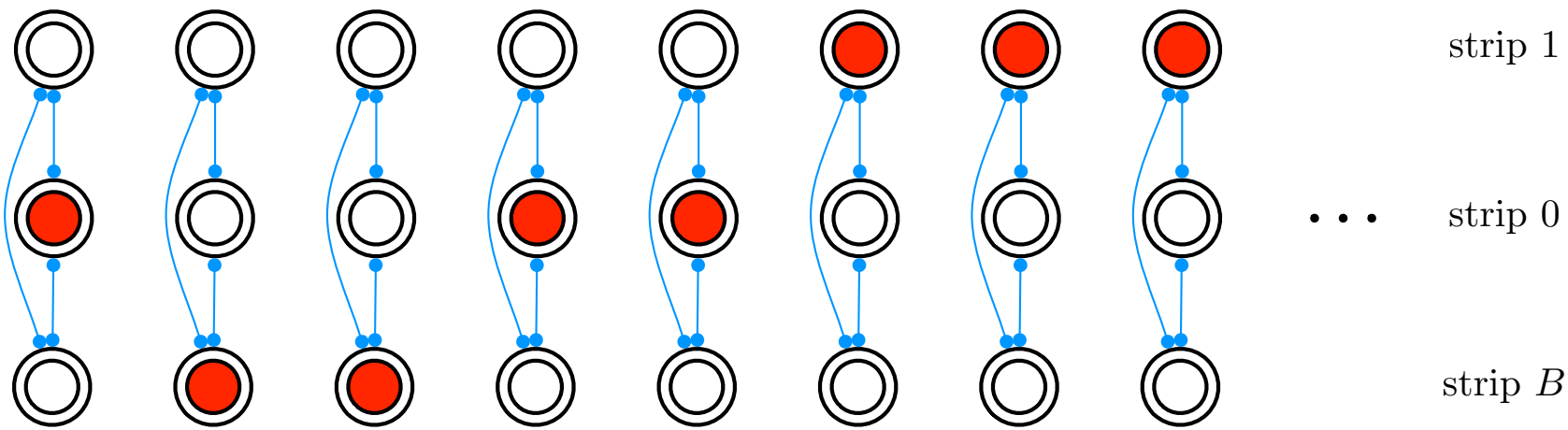
Symbol Rings

- Used to store the symbols written on the tape.
- Composed of a "blank", a "0" and a "1" strip.
- inhibitory/inhibitory connections.



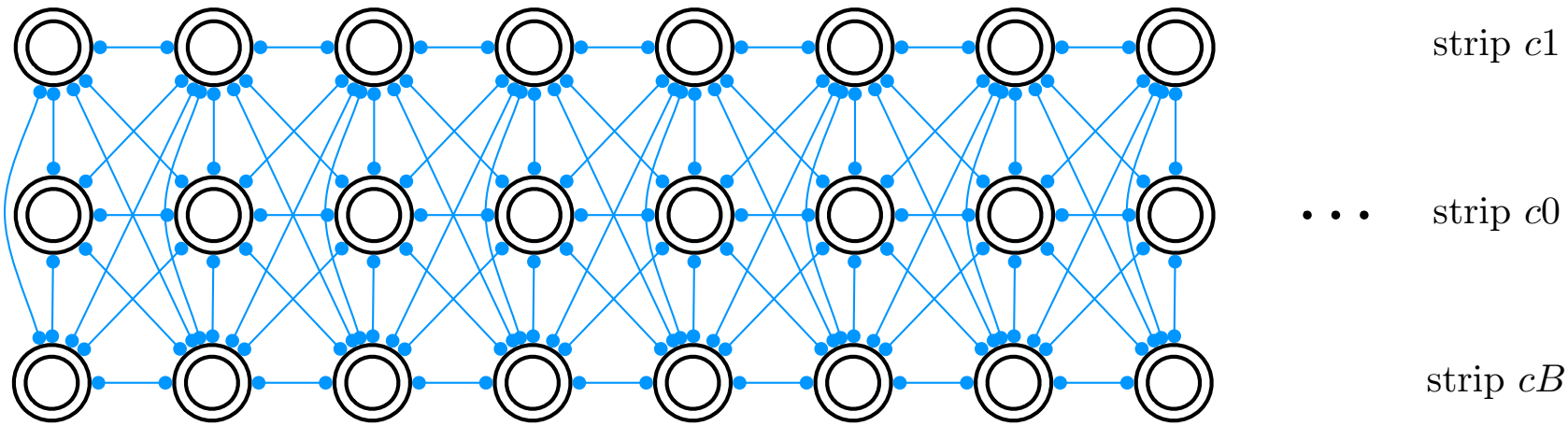
Symbol Rings

- Used to store the symbols written on the tape.
- Composed of a "blank", a "0" and a "1" strip.
- inhibitory/inhibitory connections.



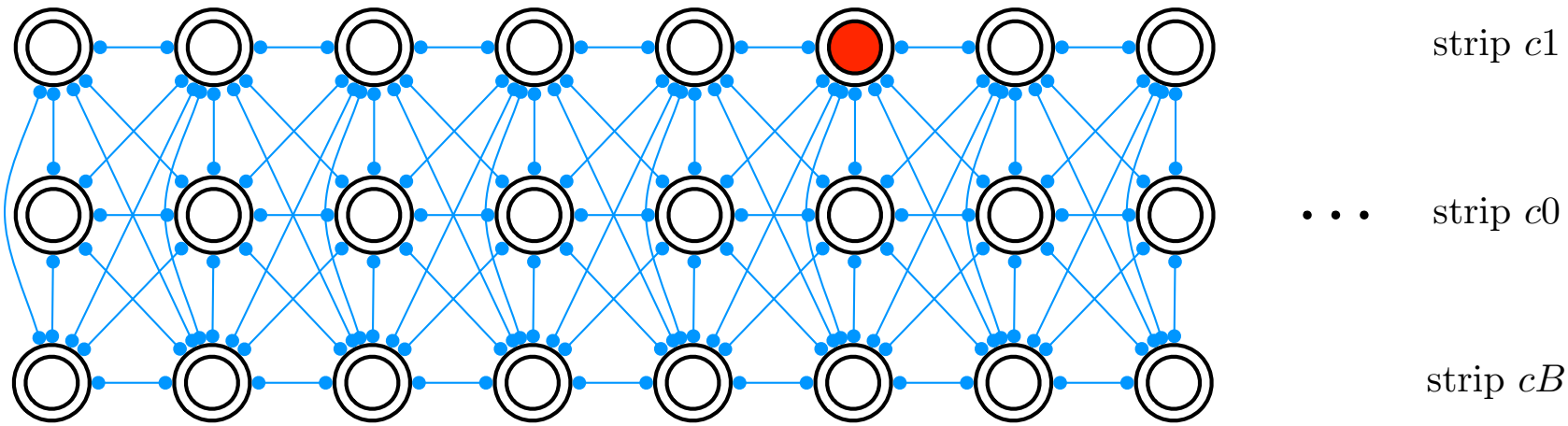
Cache Rings

- Used to store the symbol under the current head's position.
- Composed of a "blank", a "0" and a "1" strip.
- inhibitory/inhibitory connections.

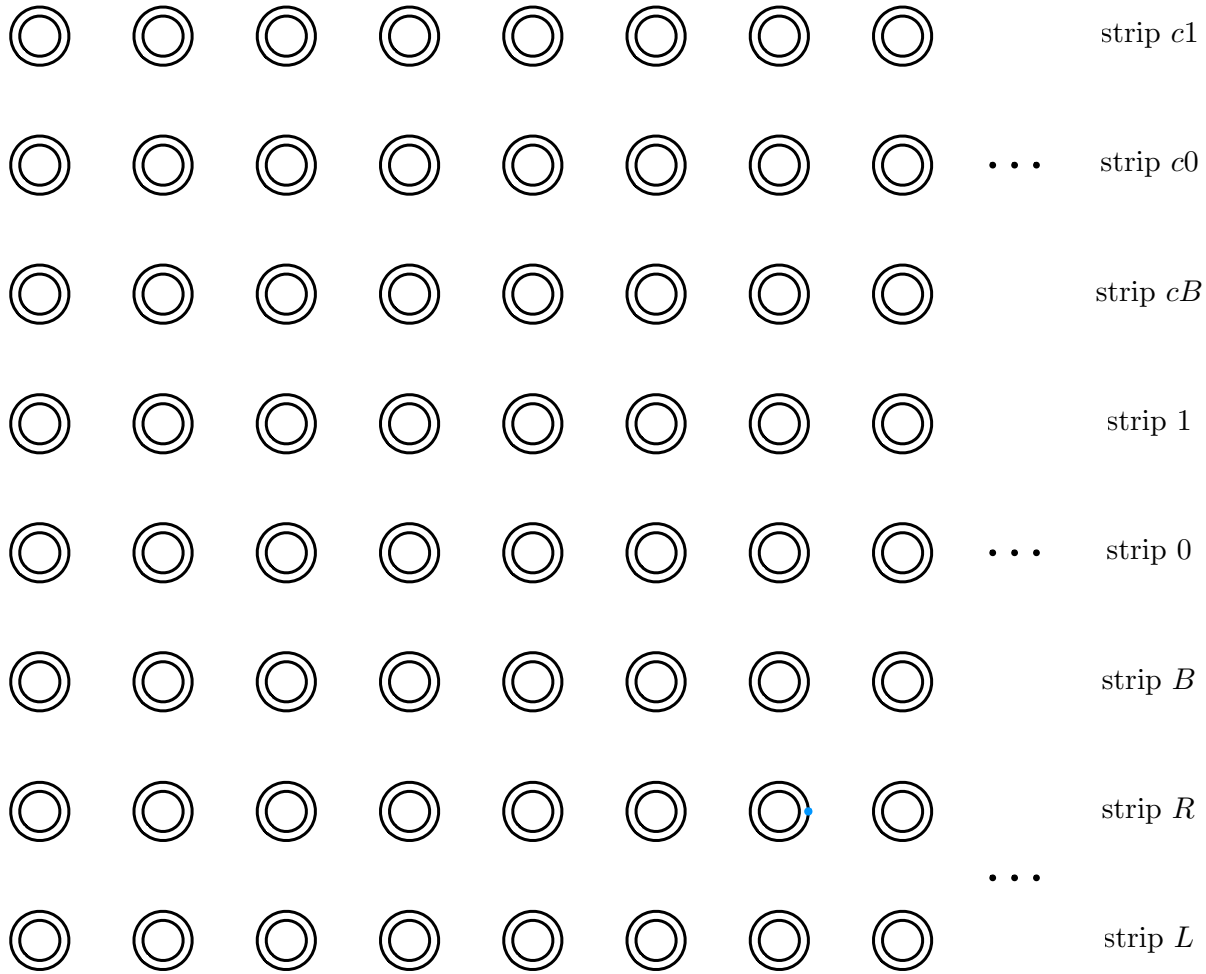


Cache Rings

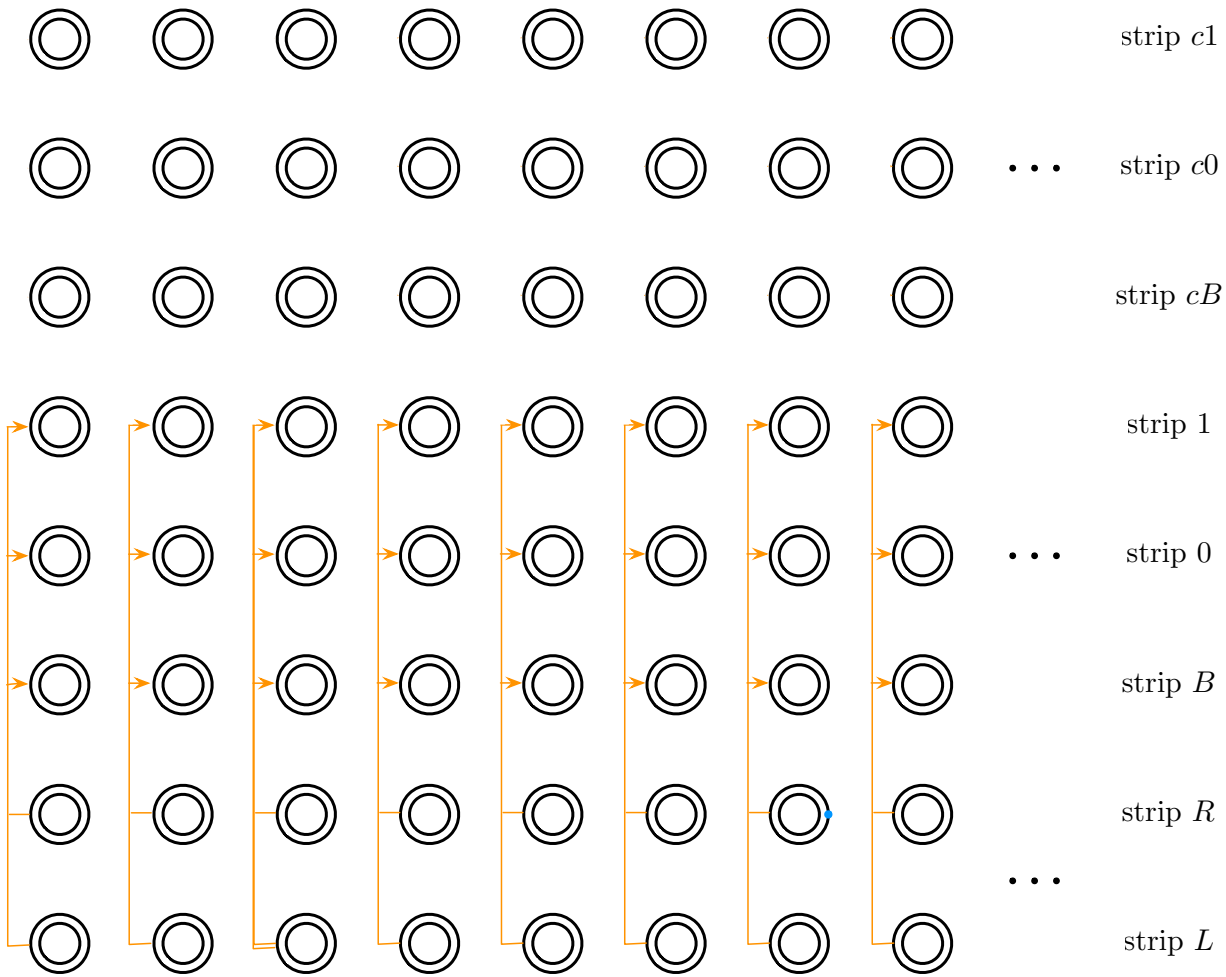
- Used to store the symbol under the current head's position.
- Composed of a "blank", a "0" and a "1" strip.
- inhibitory/inhibitory connections.



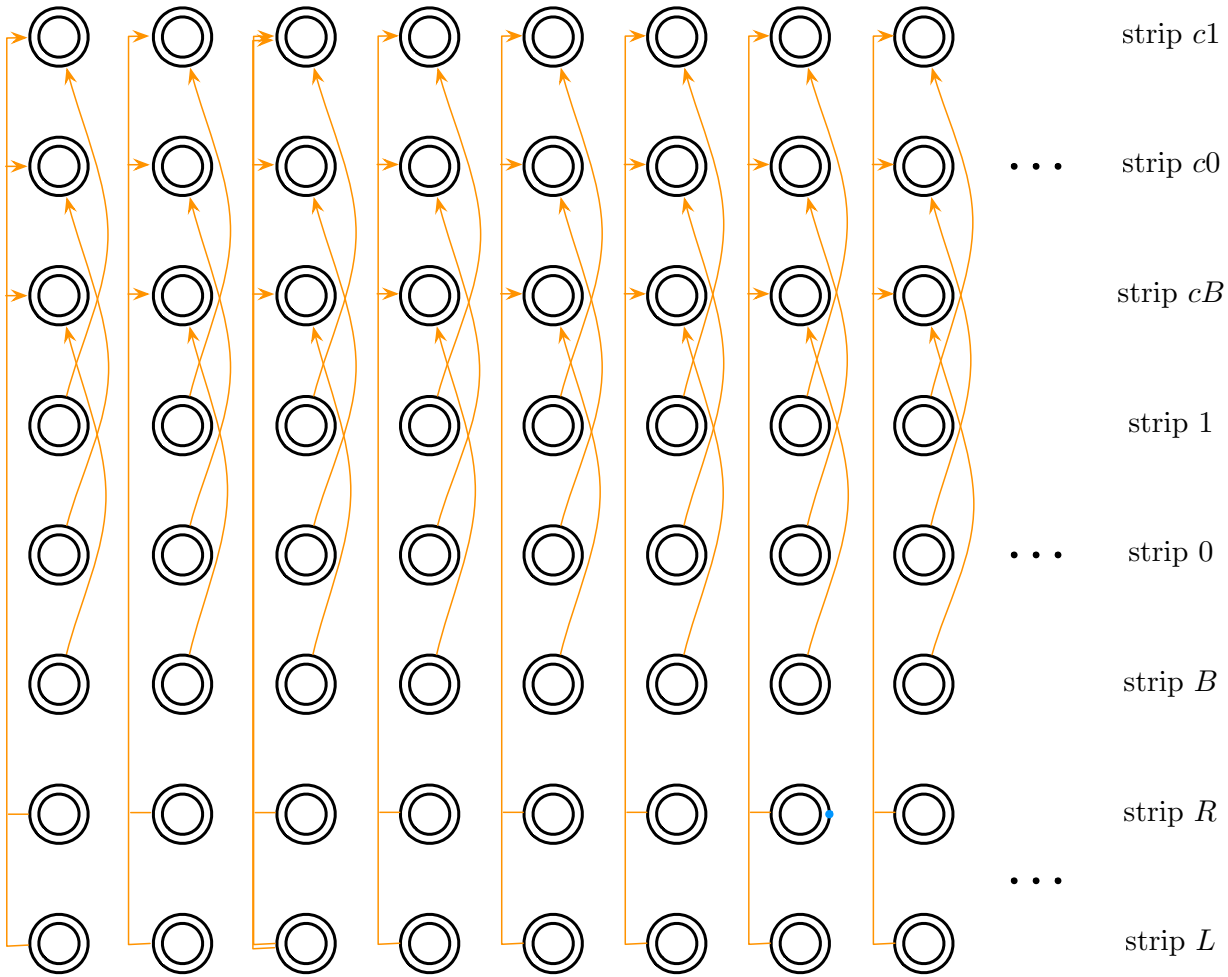
Position-Symbol-Cache Connections



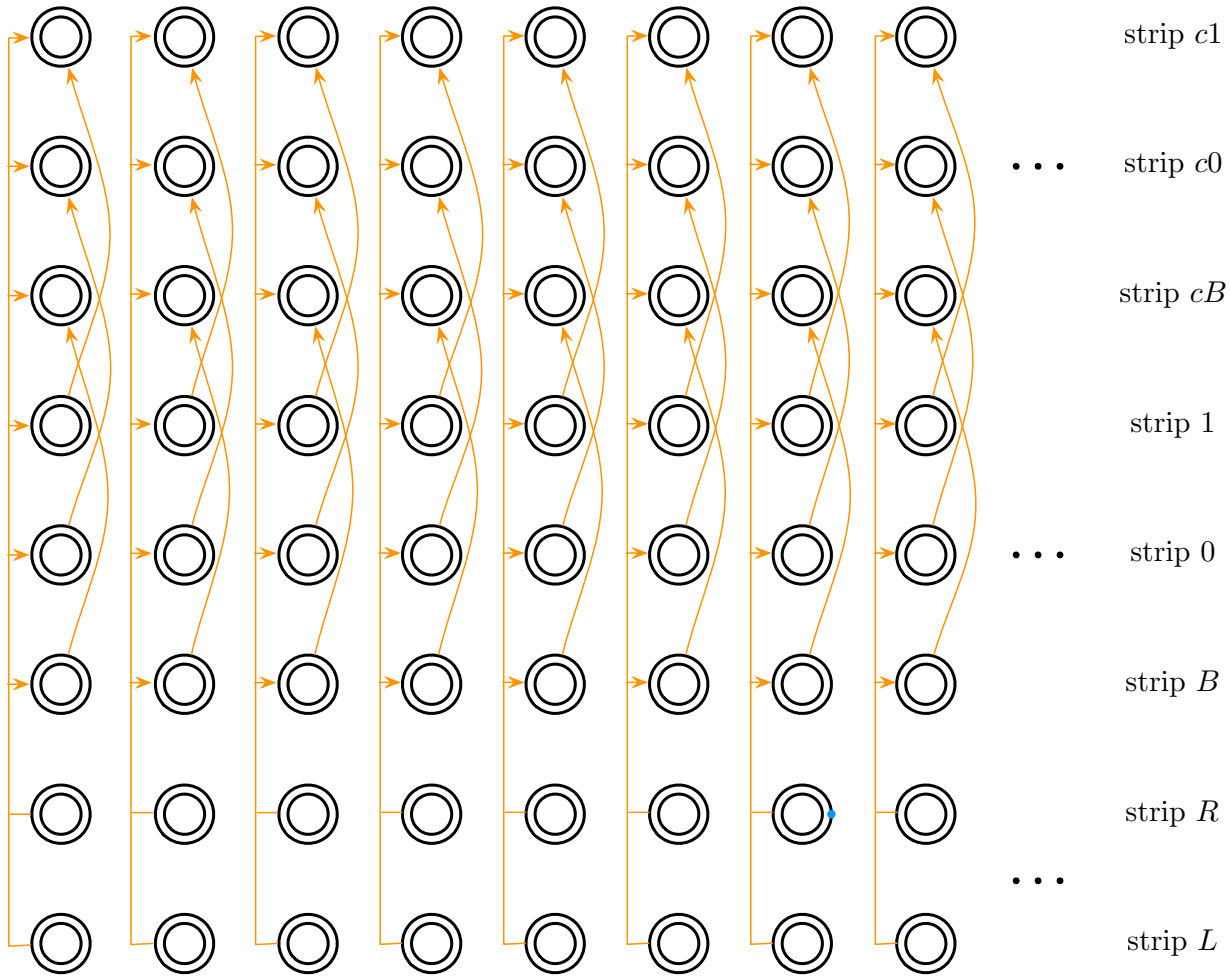
Position-Symbol-Cache Connections



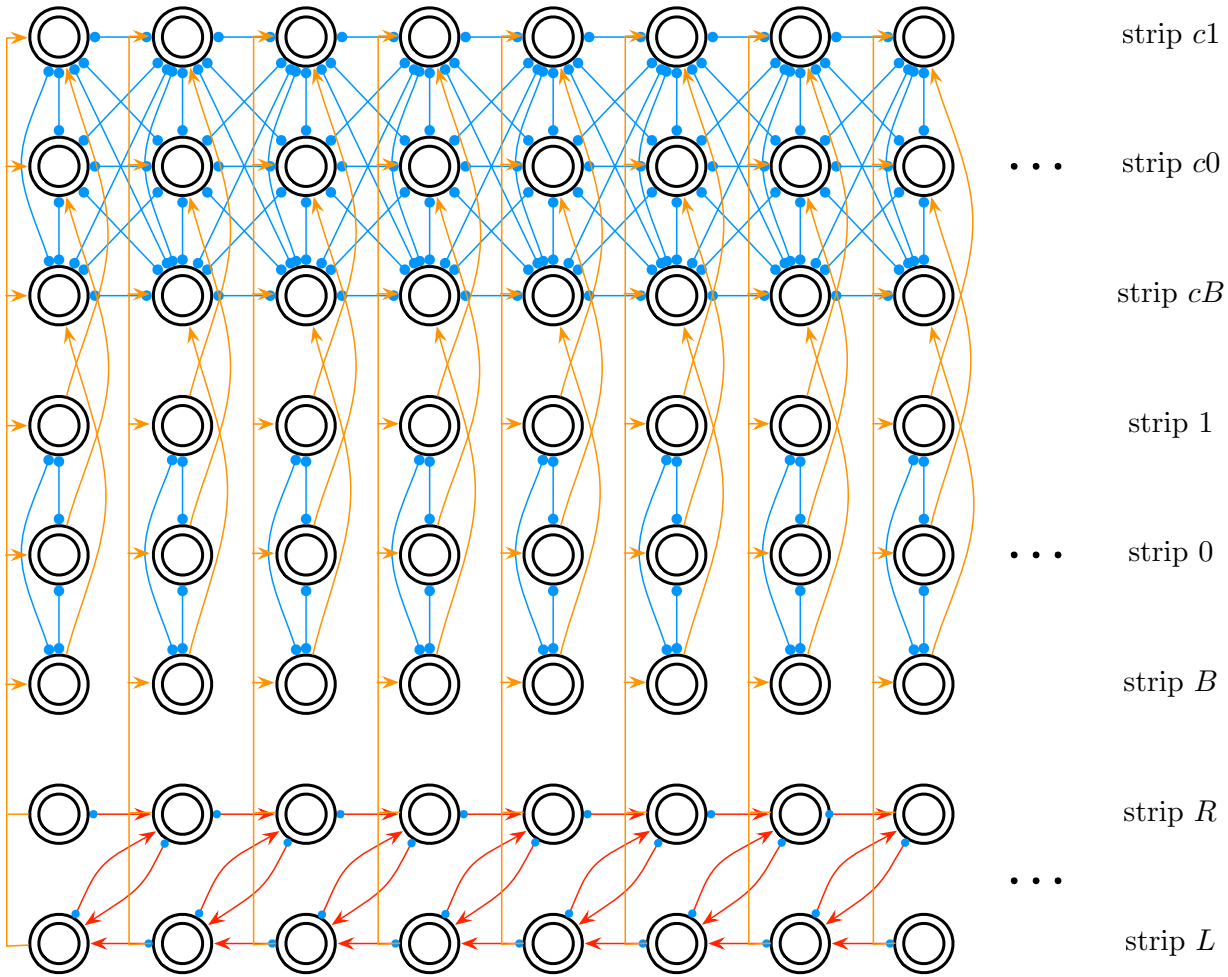
Position-Symbol-Cache Connections



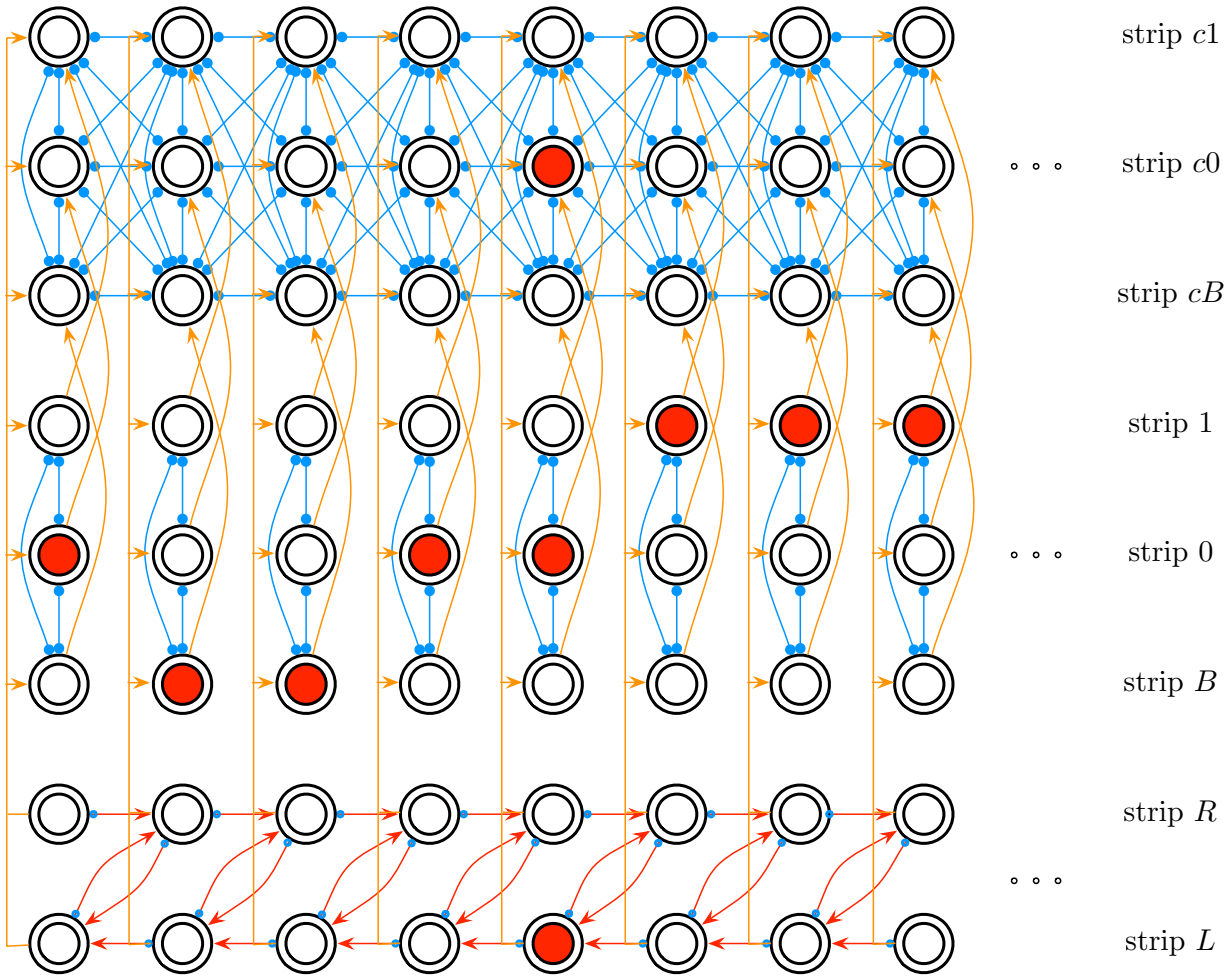
Position-Symbol-Cache Connections



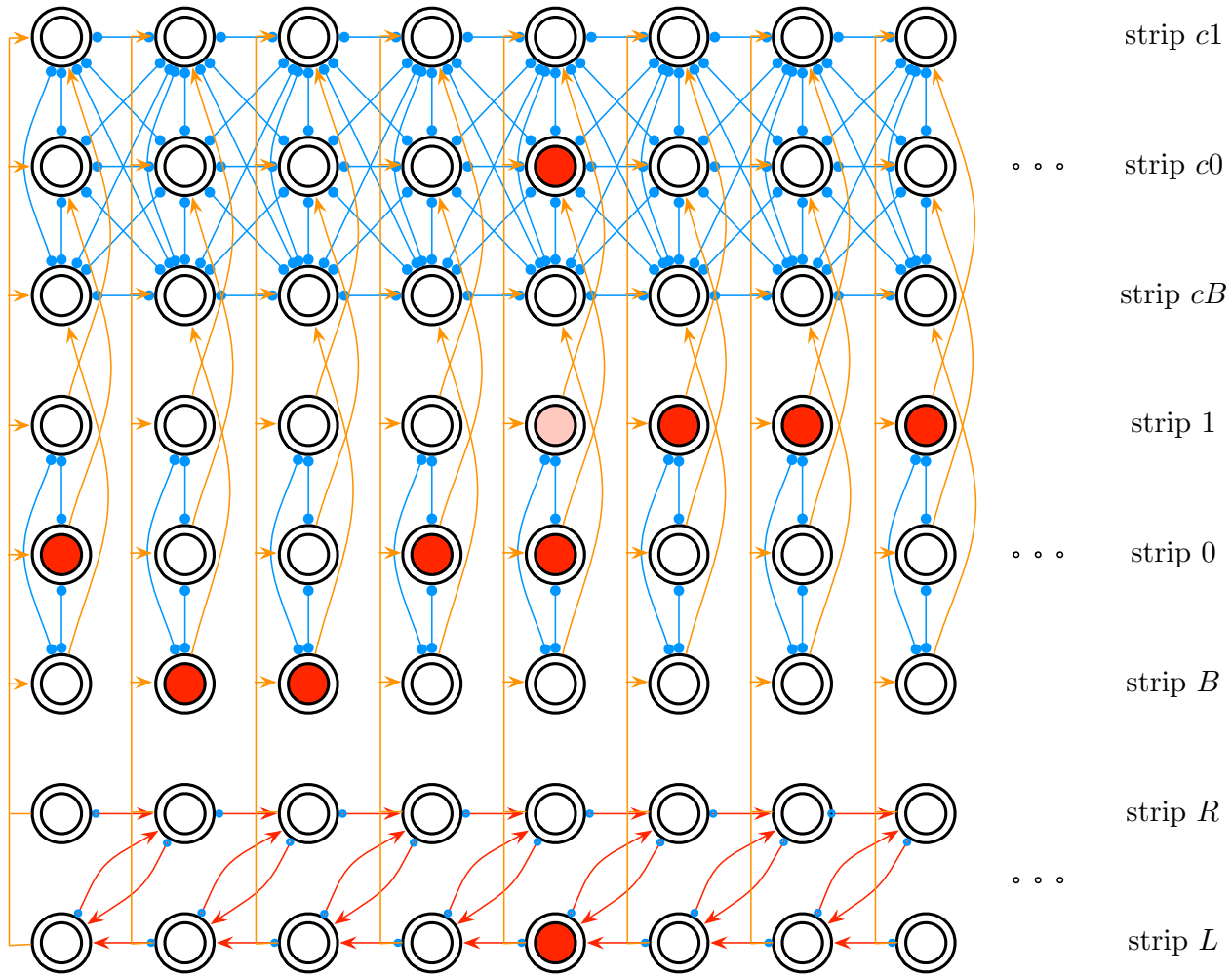
Position-Symbol-Cache Connections



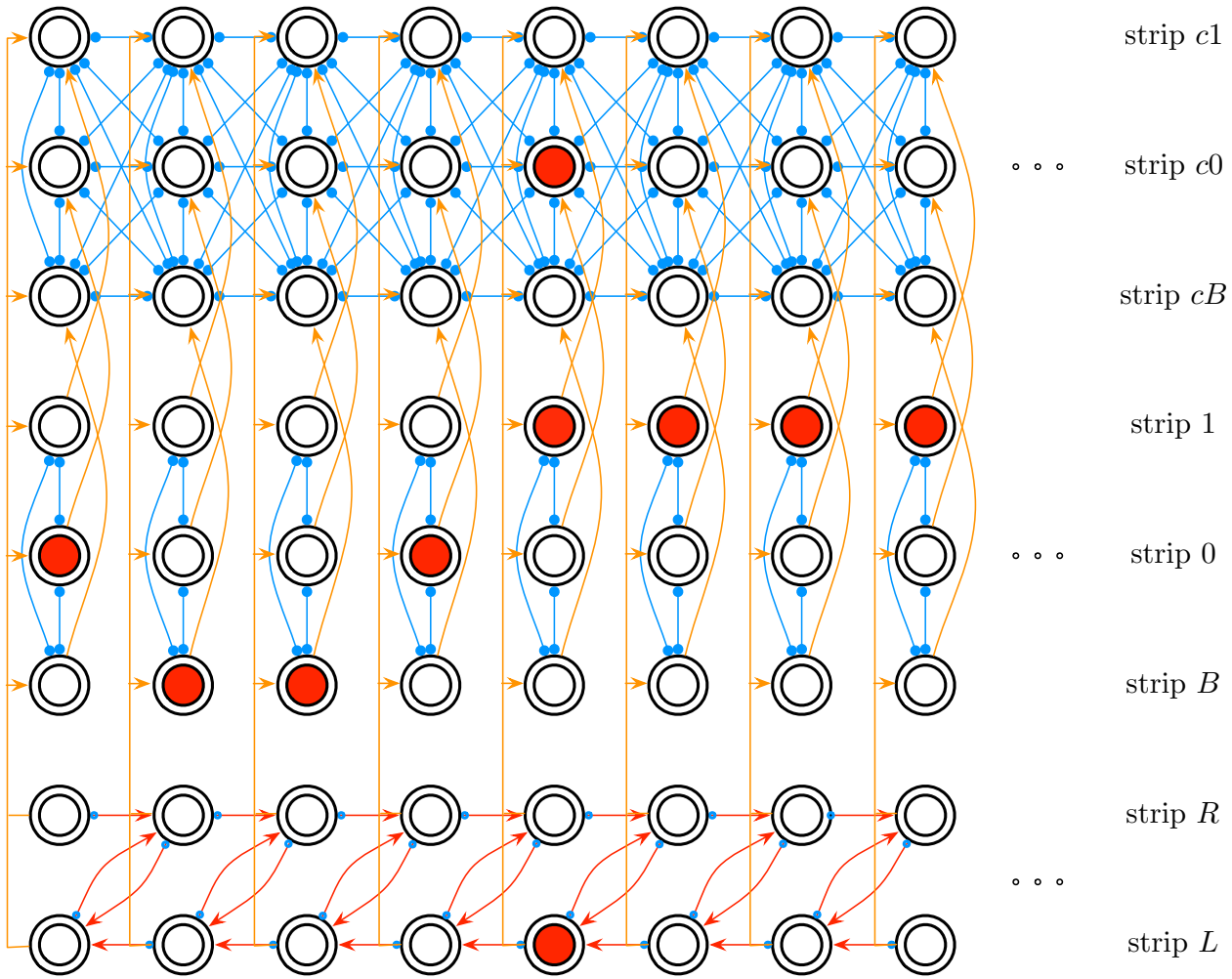
Position-Symbol-Cache Connections



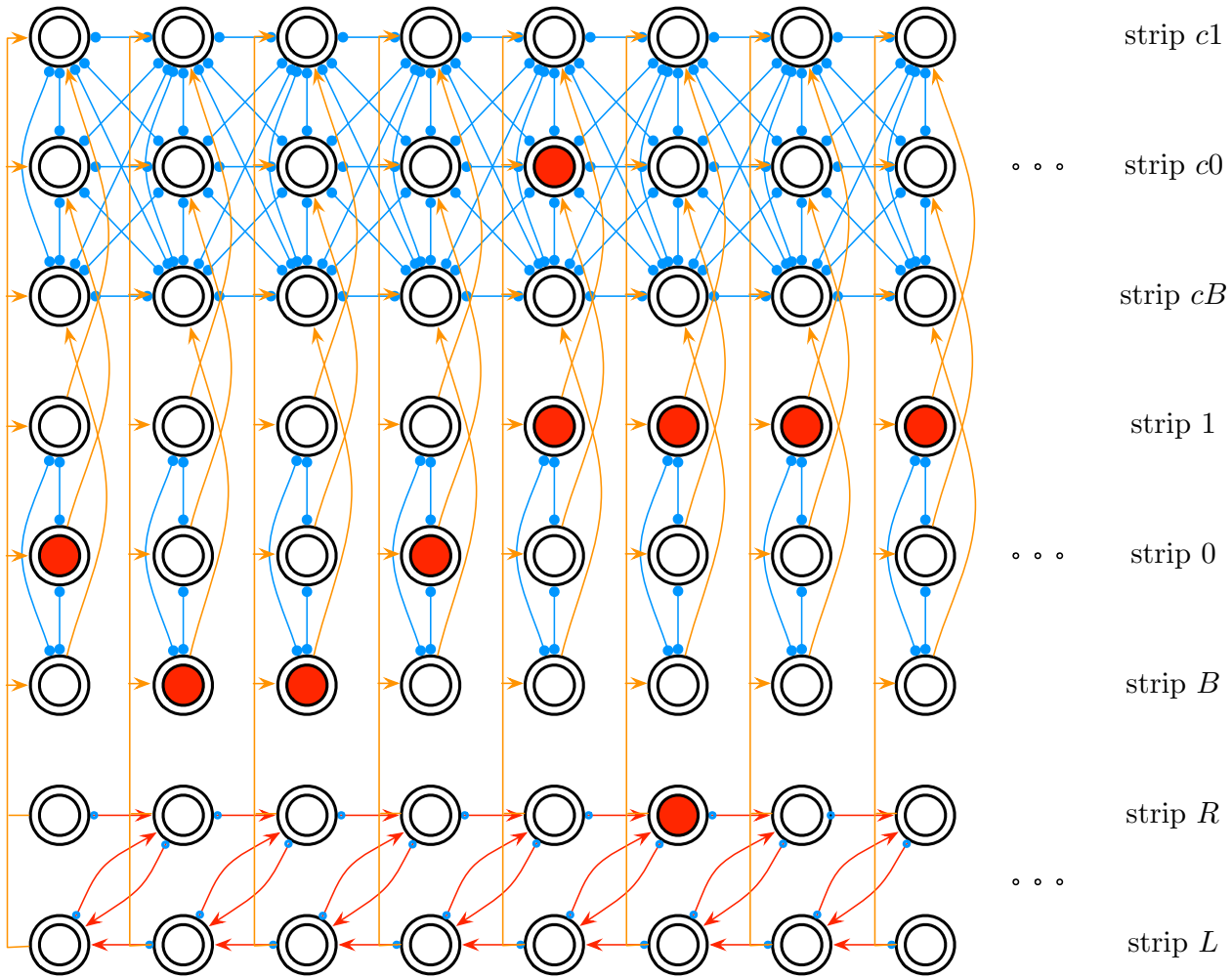
Position-Symbol-Cache Connections



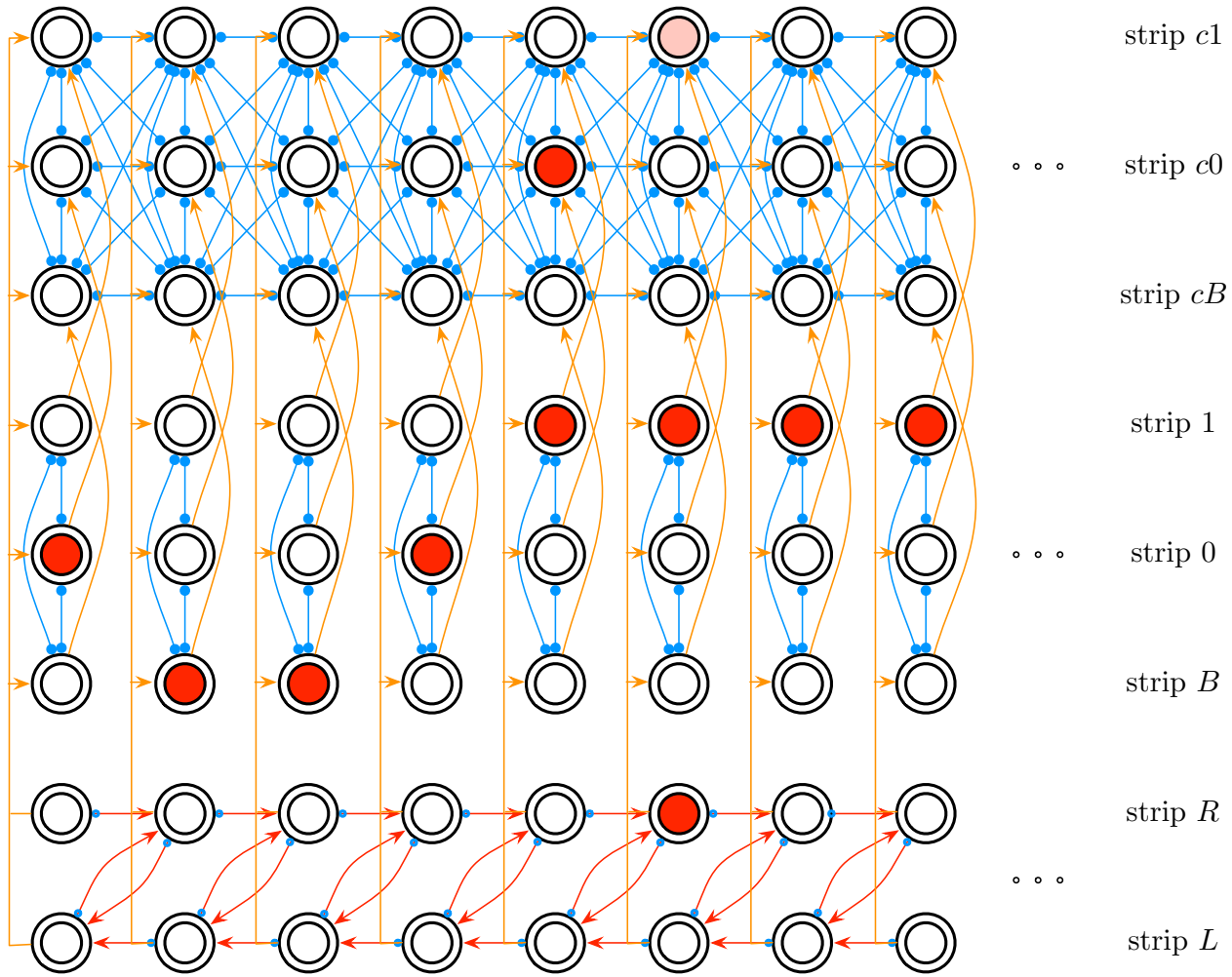
Position-Symbol-Cache Connections



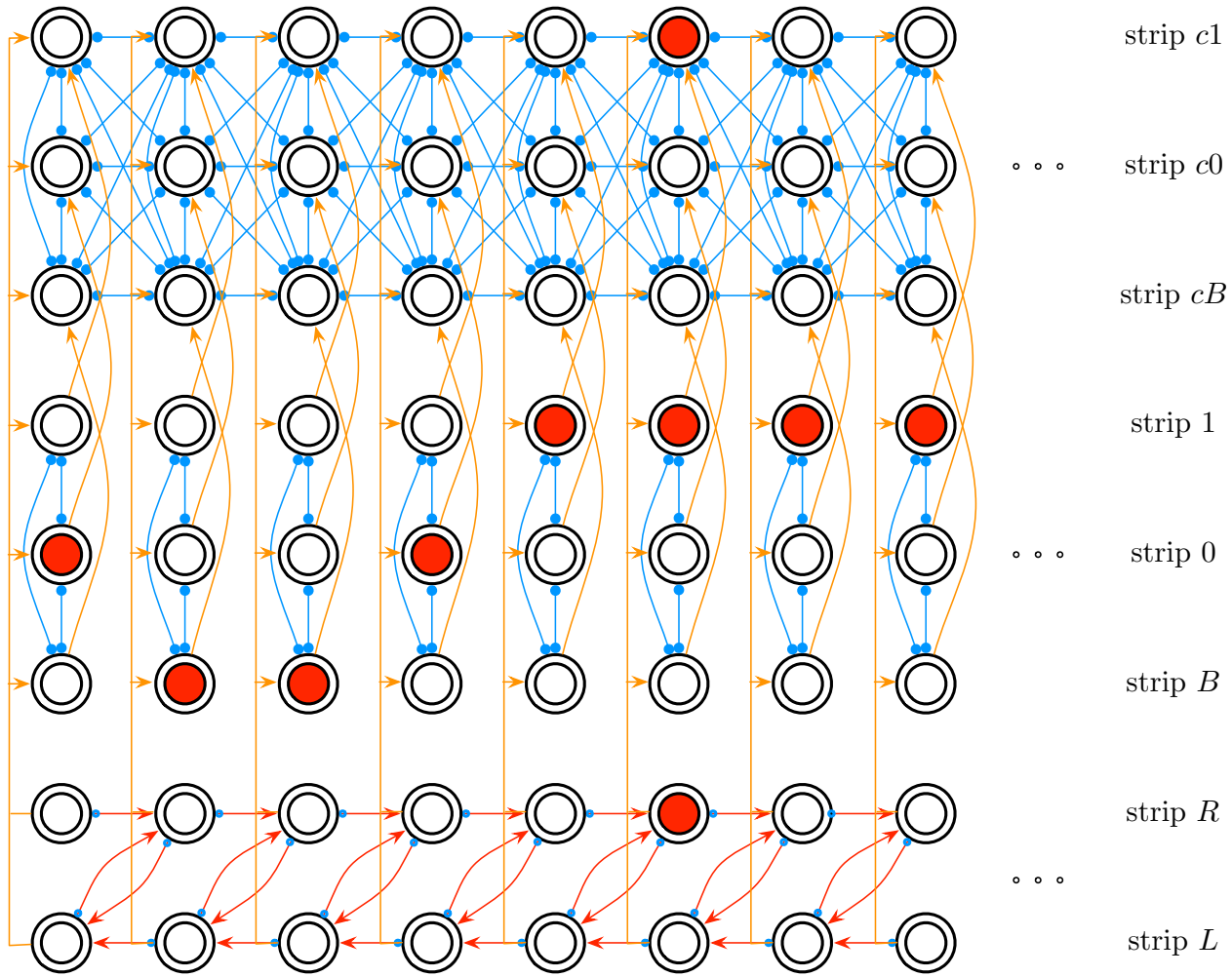
Position-Symbol-Cache Connections



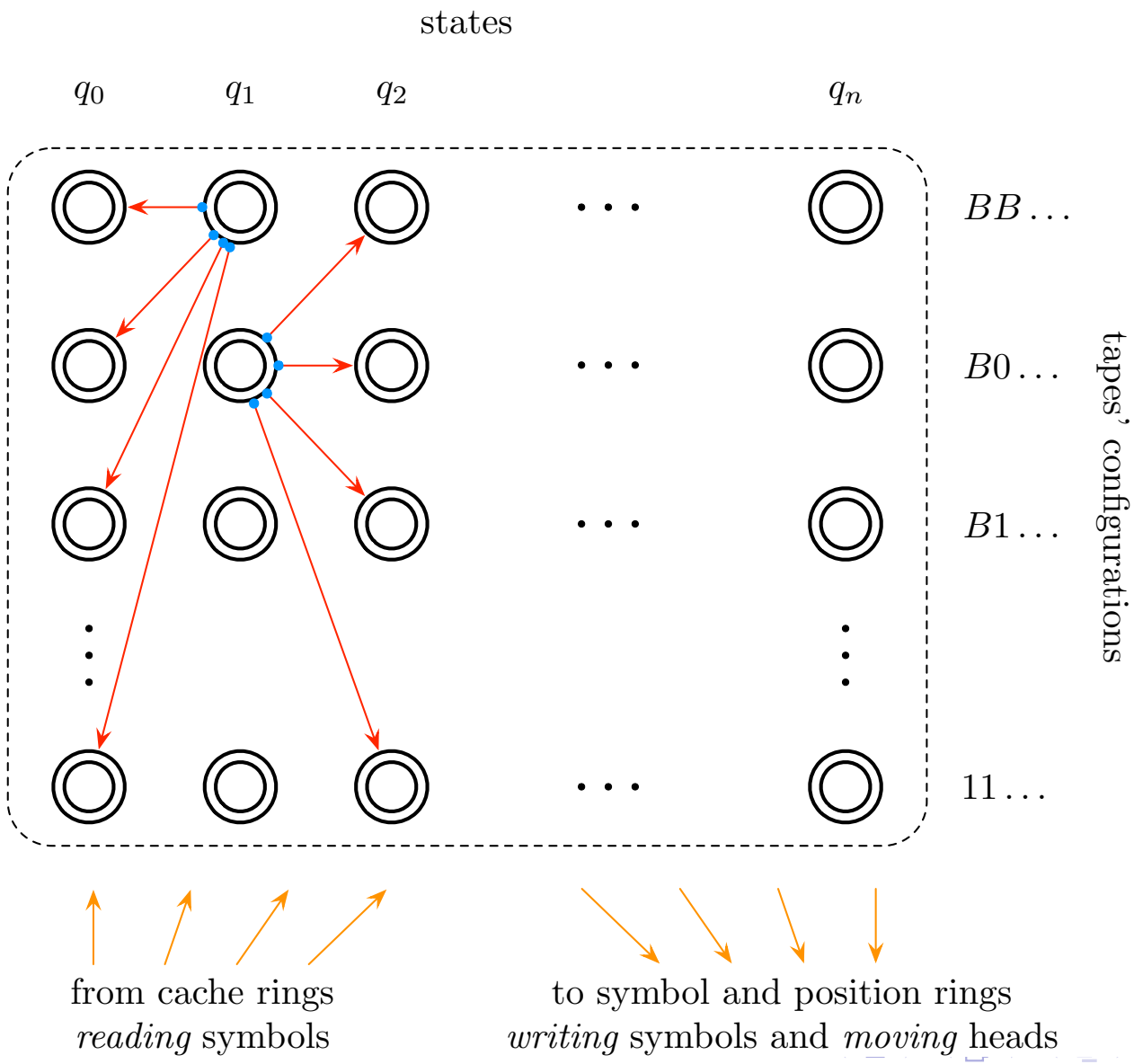
Position-Symbol-Cache Connections



Position-Symbol-Cache Connections



Program Rings



Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

Program Rings

- One ring per event "state q_i & current symbols $a_1, \ldots, a_k$".
- Program to program: excitatory/inhibitory:

$\Rightarrow$ change state

- Cache to program: excitatory

$\Rightarrow$ read current symbol

- Program to symbol: excitatory

$\Rightarrow$ write new symbol

- Program to position: excitatory

$\Rightarrow$ move head

start-tic1-tic2-tic3 procedure

- "start" cell: sets the initial configuration of the tape(s).
- "tic1" cell: triggers the *cache rings update* $\Rightarrow$ read symbol
- "tic2" cell: triggers the *program rings update* $\Rightarrow$ switch state
- "tic3" cell: triggers the *tape rings update* $\Rightarrow$ write symbol and move heads

start-tic1-tic2-tic3 procedure

- "start" cell: sets the initial configuration of the tape(s).
- "tic1" cell: triggers the *cache rings update* $\Rightarrow$ read symbol
- "tic2" cell: triggers the *program rings update* $\Rightarrow$ switch state
- "tic3" cell: triggers the *tape rings update* $\Rightarrow$ write symbol and move heads

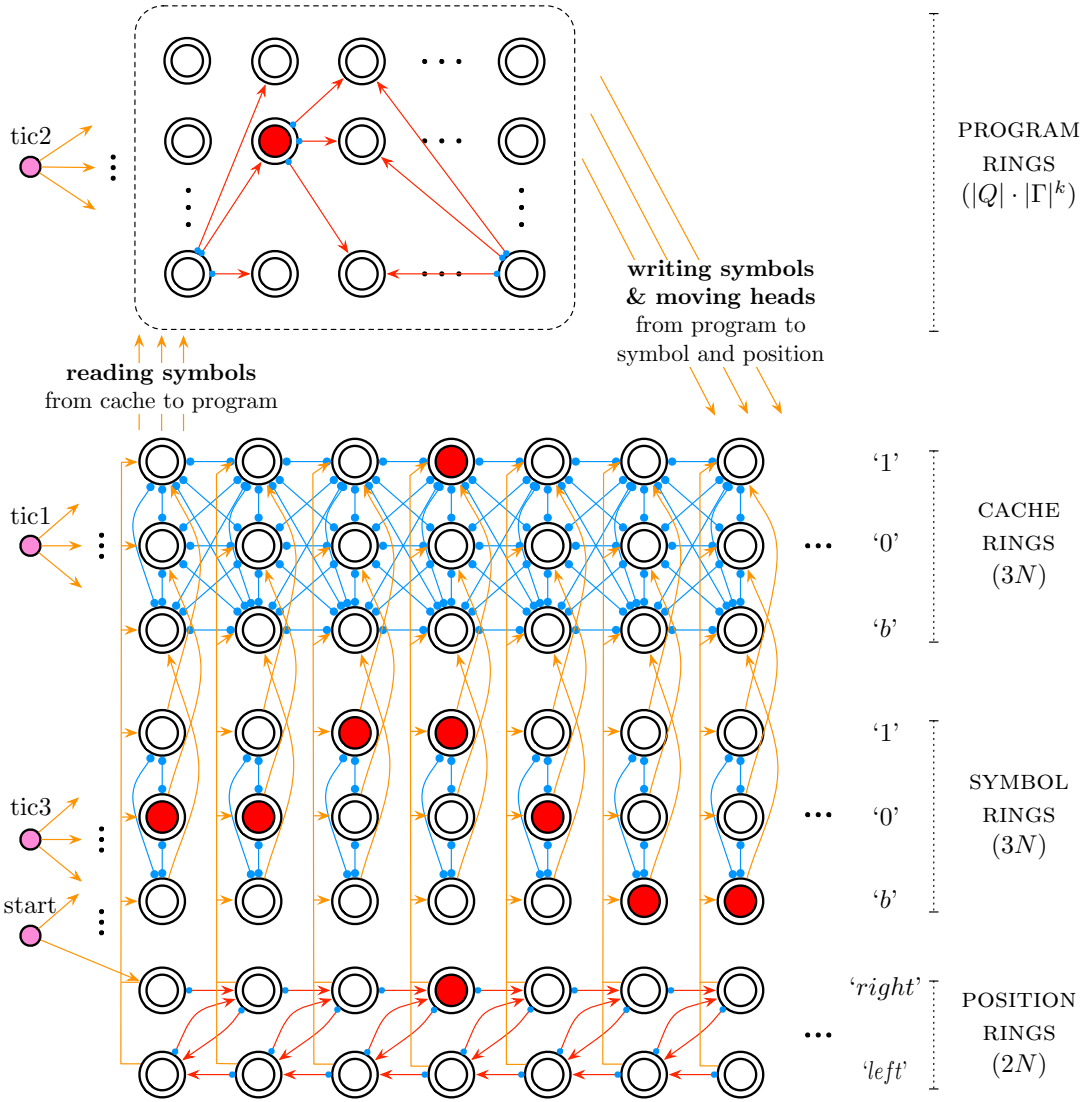
start-tic1-tic2-tic3 procedure

- "start" cell: sets the initial configuration of the tape(s).
- "tic1" cell: triggers the *cache rings update* $\Rightarrow$ read symbol
- "tic2" cell: triggers the *program rings update* $\Rightarrow$ switch state
- "tic3" cell: triggers the *tape rings update* $\Rightarrow$ write symbol and move heads

start-tic1-tic2-tic3 procedure

- "start" cell: sets the initial configuration of the tape(s).
- "tic1" cell: triggers the *cache rings update* $\Rightarrow$ read symbol
- "tic2" cell: triggers the *program rings update* $\Rightarrow$ switch state
- "tic3" cell: triggers the *tape rings update* $\Rightarrow$ write symbol and move heads

Putting everything together...



Turing machines & Boolean RNNs with Synfire Rings

The construction is generic, hence, one has the following result:

Theorem

- *Any bounded-space Turing machine can be simulated by some Boolean neural network composed of synfire rings.*
- *The family of all such synfire ring-based neural networks computes the (super-Turing) complexity class P/poly.*

Turing machines & Boolean RNNs with Synfire Rings

The construction is generic, hence, one has the following result:

Theorem

- *Any bounded-space Turing machine can be simulated by some Boolean neural network composed of synfire rings.*
- *The family of all such synfire ring-based neural networks computes the (super-Turing) complexity class P/poly.*

Simulation

Play movie...

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach: Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

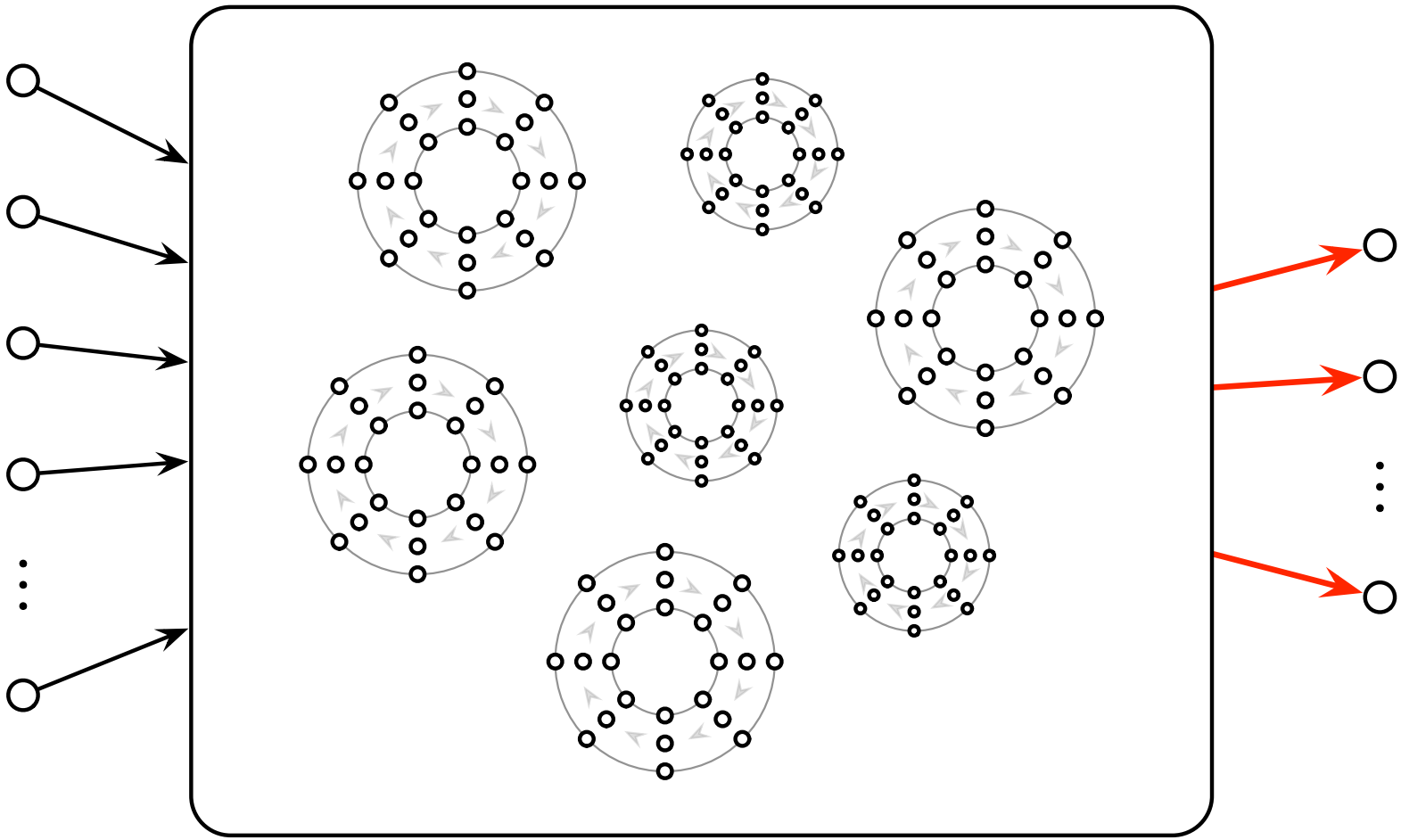
Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work

1. Generalization of the Turing-complete synfire ring architecture to more biological neural models (Hodgkin-Huxley).
2. Develop learning algorithms on this architecture:
   - ⋆ Reservoir computing approach:
     Echo State Networks / Liquid State Machines
   - ⋆ Evolutionary computation approach
   - ⋆ Biologically-oriented approach
3. Towards neuromorphic implementation.
4. Towards (real) biological implementation.

Future Work



Conclusion

Thank you!