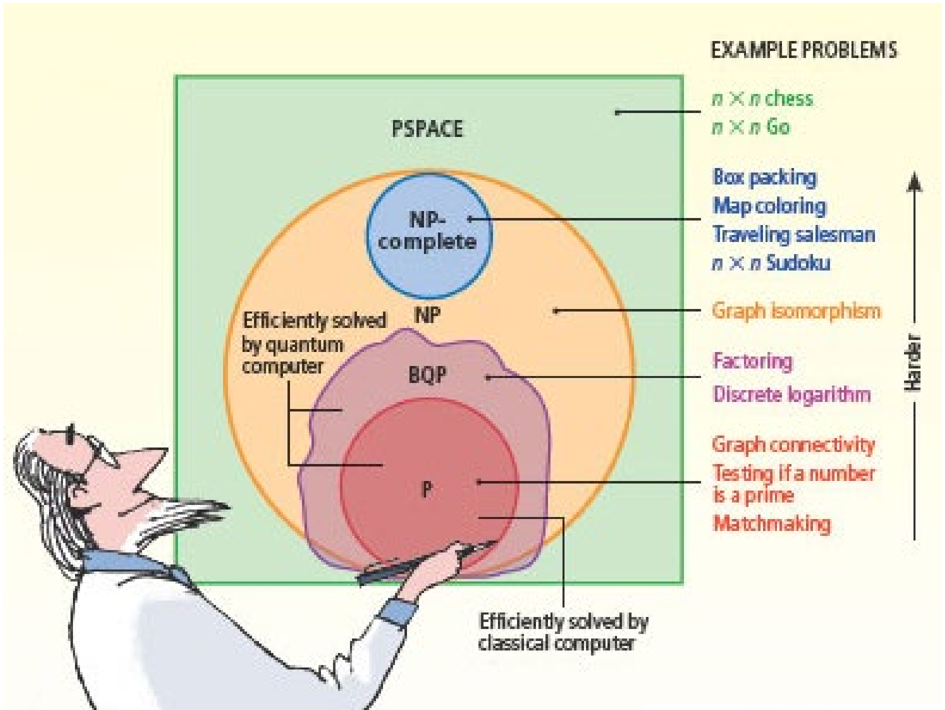


Neuromorphic Complexity Theory

Johan Kwisthout & Nils Donselaar, Donders Institute

Towards neuromorphic complexity analysis

- What kind of problems are efficiently solvable on a neuromorphic computer? Which are not? Are these problems different / the same as the problems efficiently solvable on a Von Neumann architecture?

- Given the nature of neuromorphic architectures, **energy** seems to be a vital resource (not only time)

- Our current models of computation (viz., Turing machines) capture only time and space as relevant resources for computation – not energy!

New computational model is needed

- DoE 2016 workshop report, p. 29:
  *"…likely that an **entirely new computational theory paradigm** will need to be defined in order to encompass the computational abilities of neuromorphic systems*"

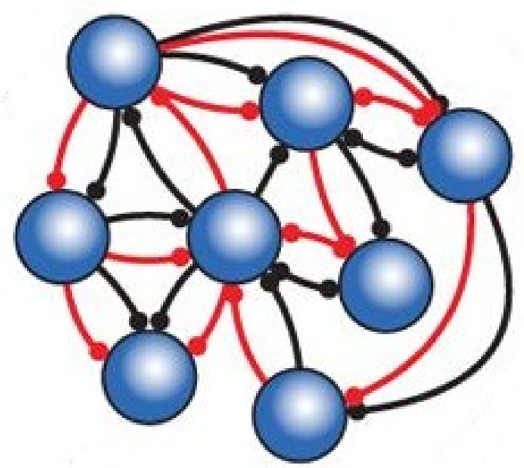
Goal: To describe what sort of **problems** can and cannot be solved **energy-efficiently** on neuromorphic hardware

Needed: New branch of complexity theory with:

1) Formal notion of "computation" in neuromorphic architectures
2) Complexity classes based on resource constraints
3) Hardness criteria and a means to *translate* problems into each other while keeping resources invariant
4) Algorithms to show that a problem is in a specific class

Proposed computational framework

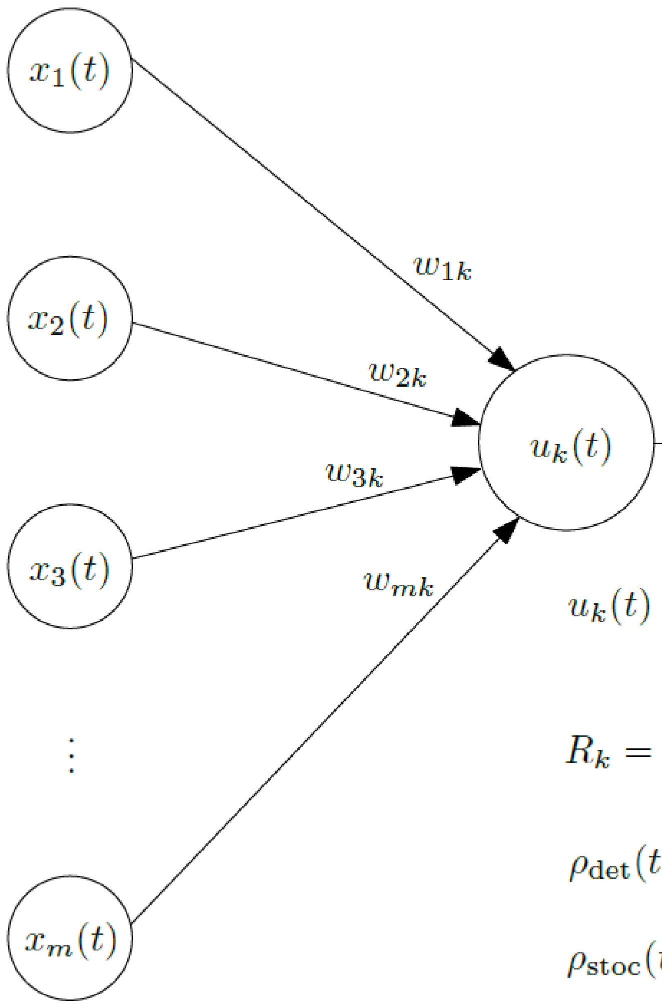
Spiking neural network model



- Key neuromorphic aspects are there:
  - Co-located memory & computation
  - Spiking behavior → energy efficiency
  - Stochastic or deterministic spikes
- Underlying principle of Loihi (& SpiNNaker)

Figure adapted from Habenschuss, S., Jonke, Z., & Maass, W. (2013). Stochastic Computations in Cortical Microcircuit Models. PLoS computational biology, 9(11), e1003311-e1003311.

Neuronal model: basically simple LIF model



- Specific **input** and **readout** neurons to inject / extract information
- Stochasticity or determinism
- Discrete time steps

$$u_k(t) = \begin{cases} R_k, & \text{if } u_k(t-1) \geq T_k; \\ 0, & \text{if } u_k(t-1) \leq 0; \\ m_k u_k(t-1) + \sum_j w_{jk} x_j(t - d_{jk}), & \text{otherwise.} \end{cases}$$

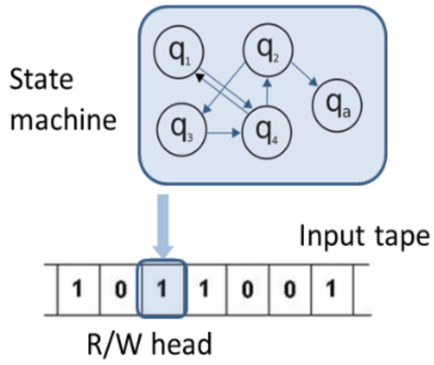
$$R_k = \text{reset voltage}, T_k = \text{threshold}, m_k = \text{leakage constant}$$

$$\rho_{\text{det}}(t) = \begin{cases} 1, & \text{if } u_k(t-1) \geq T_k; \\ 0, & \text{otherwise.} \end{cases}$$

$$\rho_{\text{stoc}}(t) \propto f(u_k(t))$$

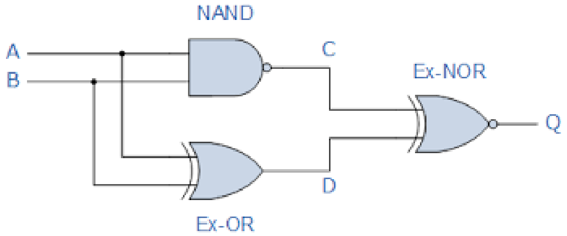
$$x_k(t) = \begin{cases} 1, & \text{if a spike was released in } \langle t - \tau_k, t]; \\ 0, & \text{otherwise.} \end{cases}$$

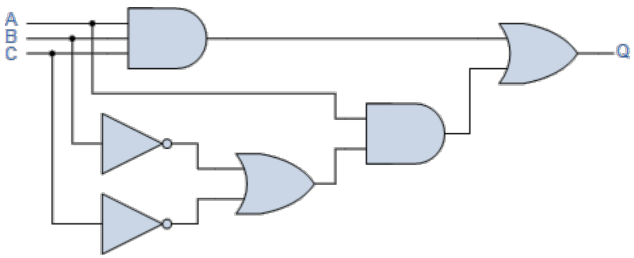
Beyond Turing



- **Turing Machine M_L**
- Input **I** encoded (in binary) on the tape
- State machine $\mathbf{M_L}$ implements algorithm
- Formally: recognizes languages $\mathbf{L} \subset \{0,1\}^*$
- Canonical question: Does $\mathbf{M_L}$ accept $\mathbf{I} \in \mathbf{L}$ using resources (time/space) at most **R**?

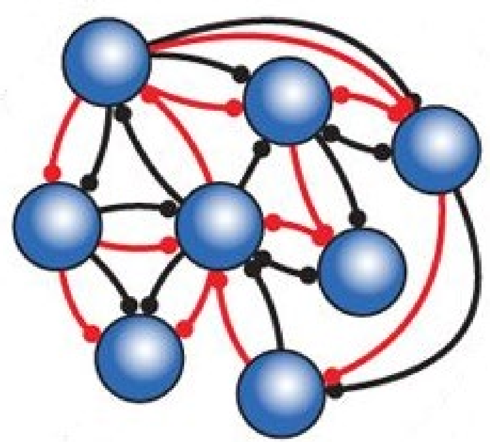
- ***Family* of Boolean Circuits $C_{L,|I|}$**
- Input **I** encoded as special input gates
- Circuit (different circuit per input size |I|) implements algorithm
- Formally: recognizes languages $\mathbf{L} \subset \{0,1\}^*$
- Canonical question: Does, for every I, the corresponding circuit $\mathbf{C_{L,|I|}}$ accept $\mathbf{I} \in \mathbf{L}$ using resources (time/space) at most **R**?





Beyond Turing

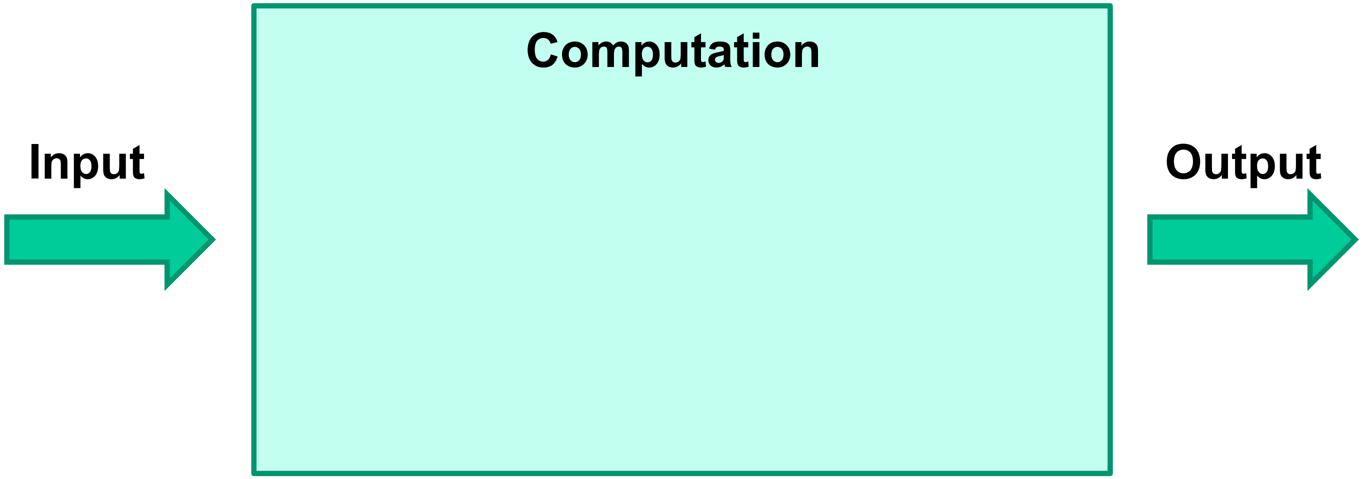
- In SNNs, input **I** and algorithm **A** are **co-located!**
- We take the circuit idea *to the extreme*…



- ***Collection* of SNNs $\mathbf{S_{L,I}}$**
- *One* network for *every* input **I** (or set of inputs {**I**})
- Input and 'algorithm' operating on it are encoded in the network structure
- Formally: recognizes languages **L** $\subset$ {0,1}*
- Accept / reject by special neurons firing
- Canonical question: Is there a resource-bounded Turing machine $\mathbf{M_L}$ that, given **I**, generates $\mathbf{S_{L,I}}$ which decides **I** using resources at most $\mathbf{R_S}$?
- **Agnostic** about how $\mathbf{S_{L,I}}$ is generated (trained / programmed / configured)

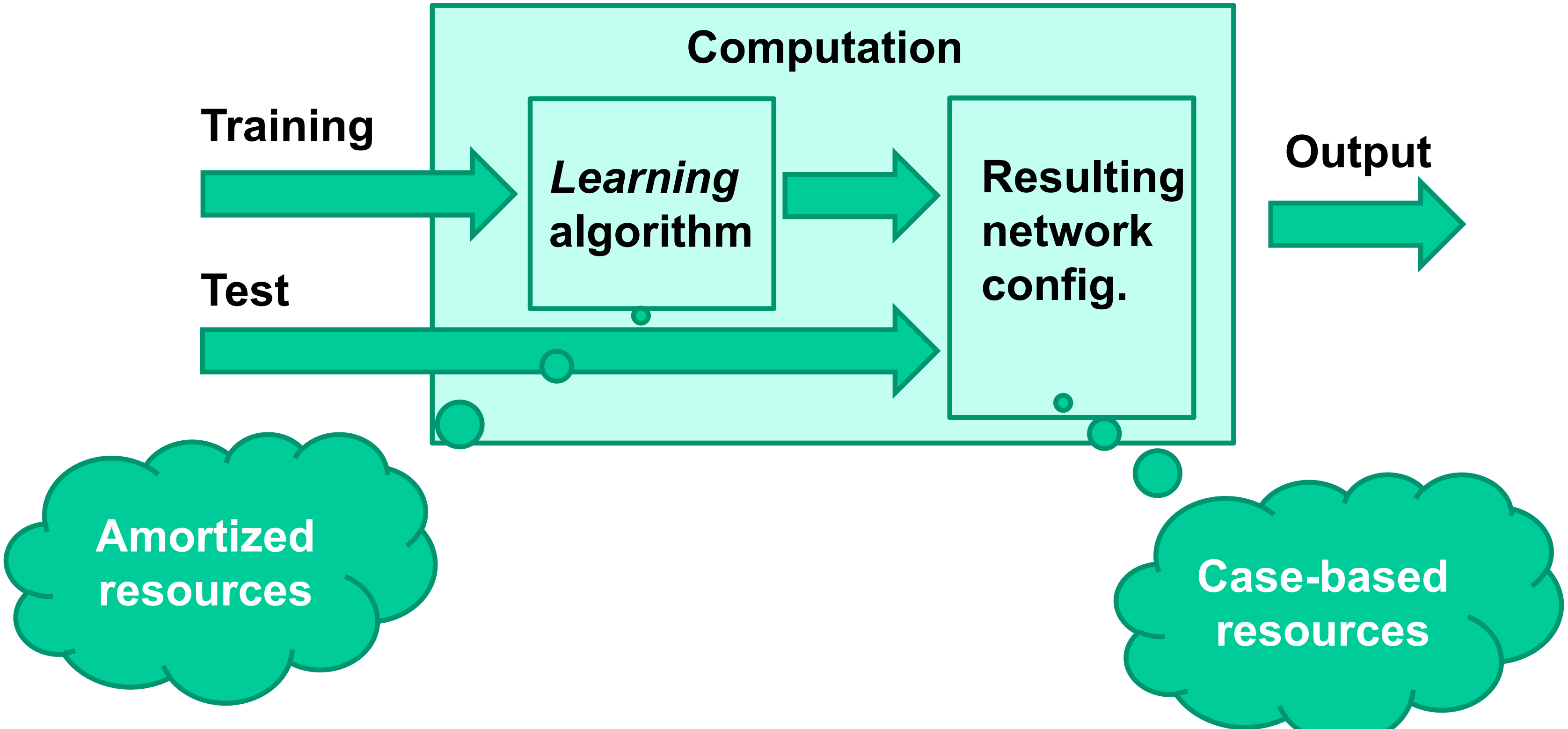
Towards neuromorphic complexity analysis

- No **cheating**: constructing / configuring / training the network should be part of the computational model and count for towards resource usage



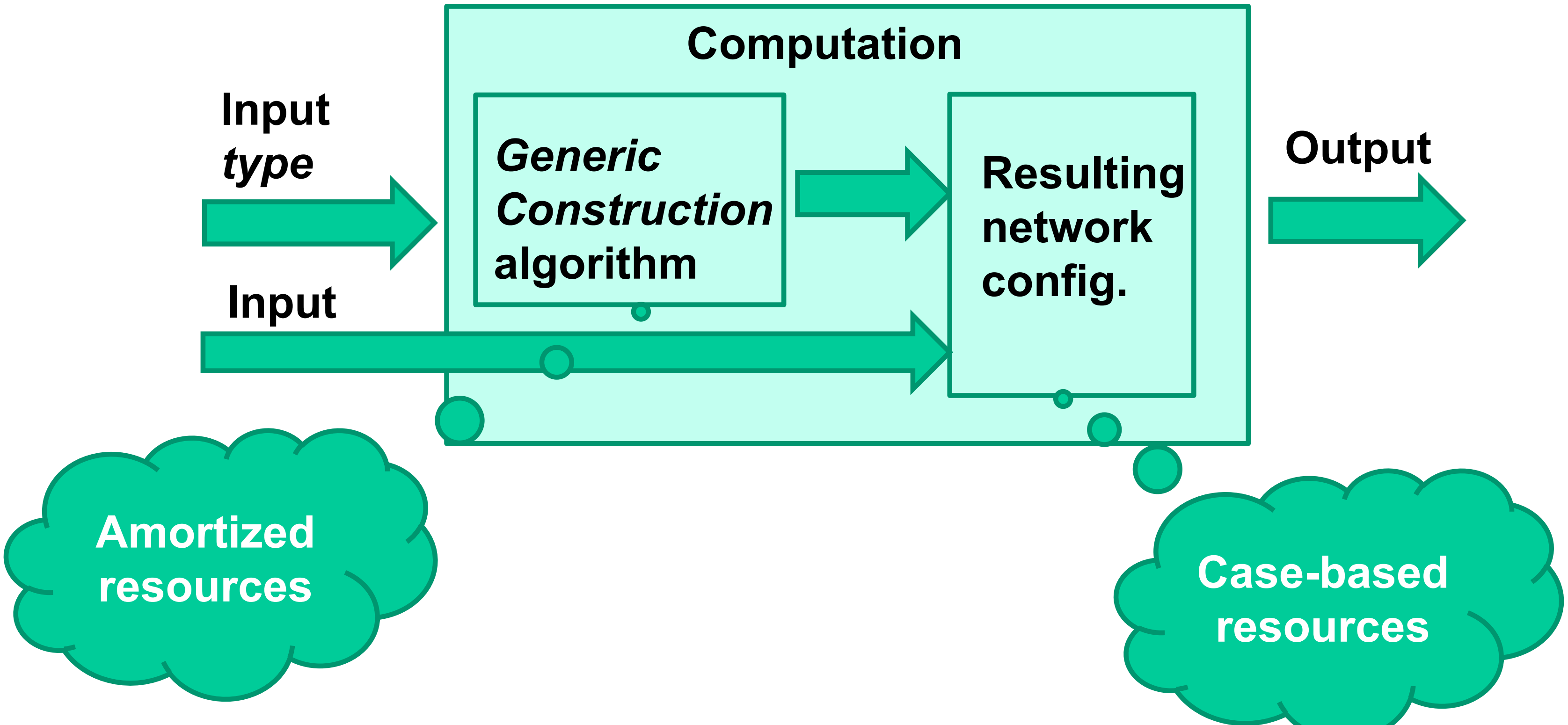
Towards neuromorphic complexity analysis

- Traditional **Machine-Learning view**
  (e.g. train a network for pattern recognition)



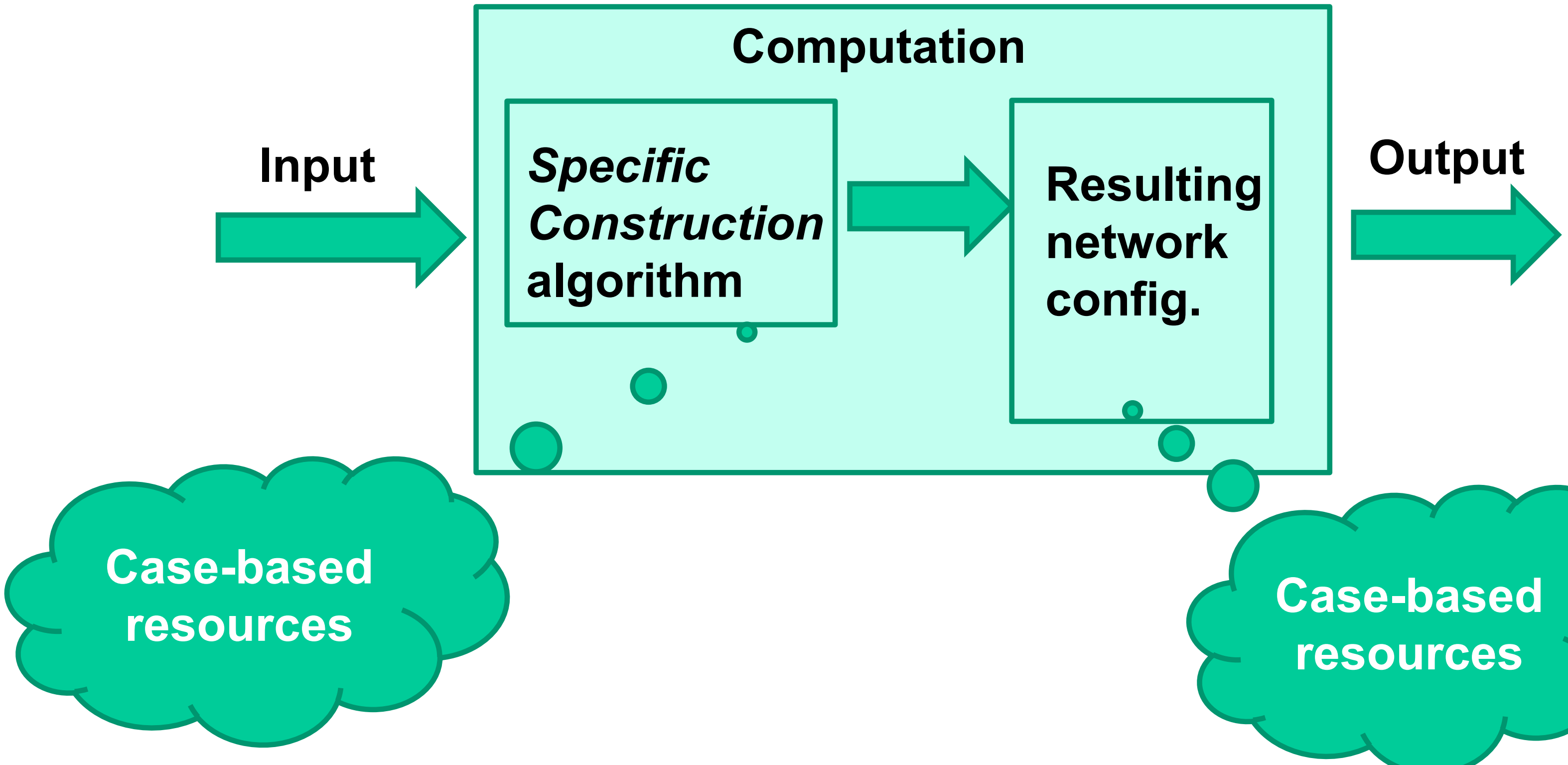
Towards neuromorphic complexity analysis

- **Configuration view** (e.g. construct a ‘generic’ network for solving graph optimization problems)



Towards neuromorphic complexity analysis

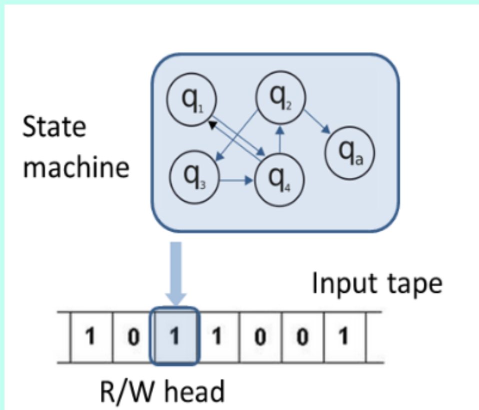
- **Programming view** (on the basis of input, construct a network that computes [more efficiently] on that input, e.g. shortest path in a graph)



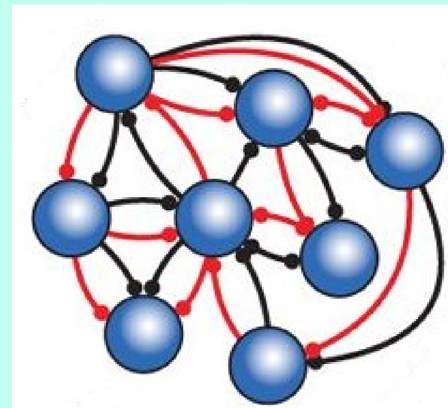
Beyond Turing: preprocessing + computation

Input

Computation



Pre-processing step taking resources $\mathbf{R_A}$



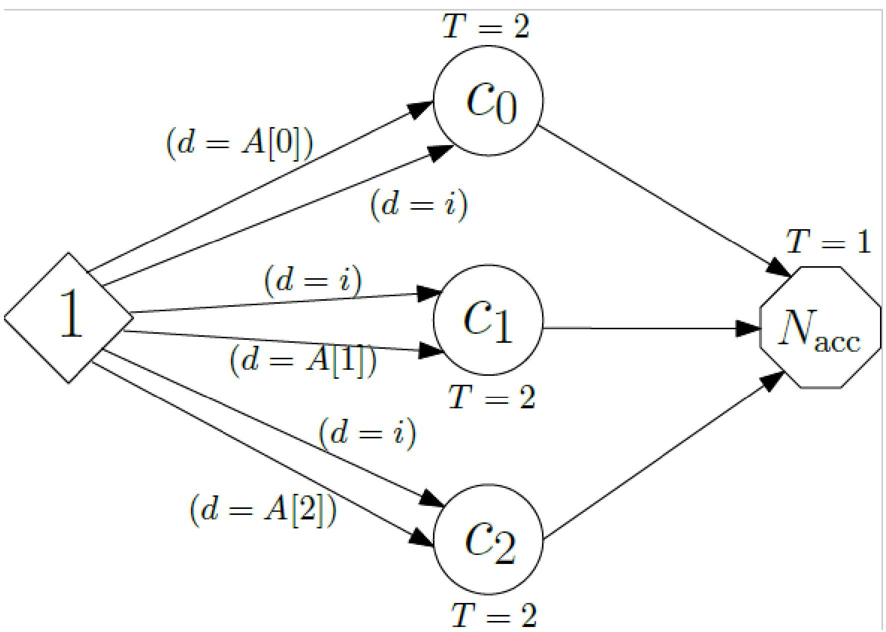
Computing step taking resources $\mathbf{R_S}$

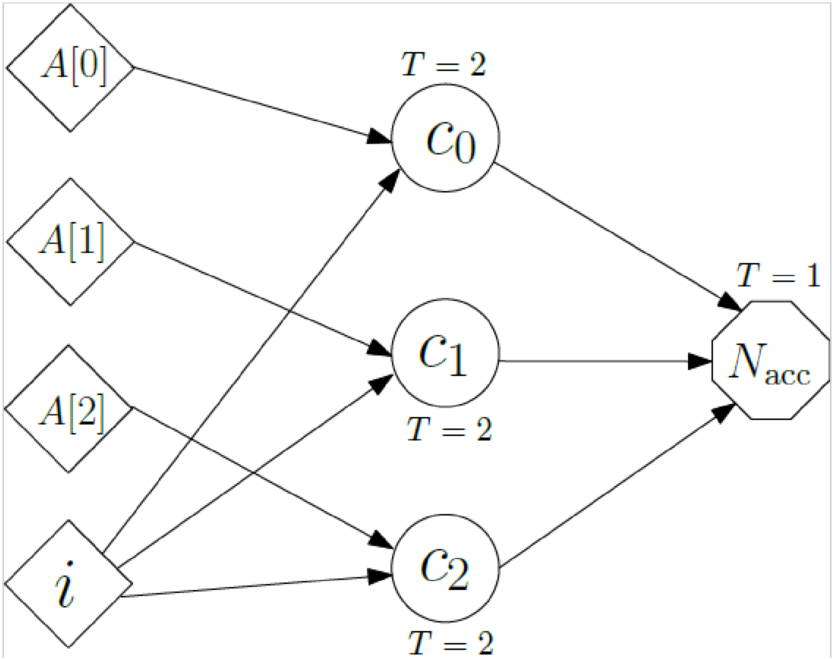
Output

- **Hierarchy** of complexity classes defined by choices for $\mathbf{R_A}$(time, space) and $\mathbf{R_S}$(time, space, energy)
- E.g., $\mathbf{R_A}$(poly time, log space), $\mathbf{R_A}$(poly time, space, energy)
- TM-preprocessing-then-SNN-computation-model: [M ° S]

Trade-off network generality vs efficiency

Deciding whether array A[n] contains integer *i* (O(n) on CPU)





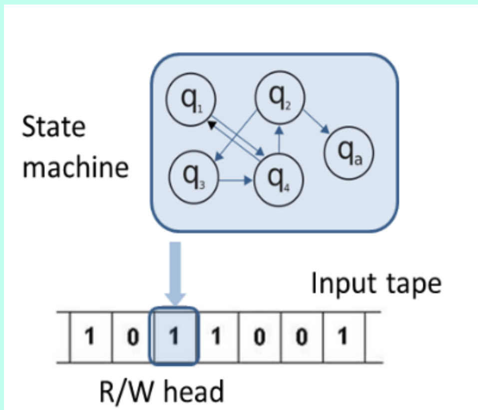
≤ |N| + 2 spikes
i + 1 time steps

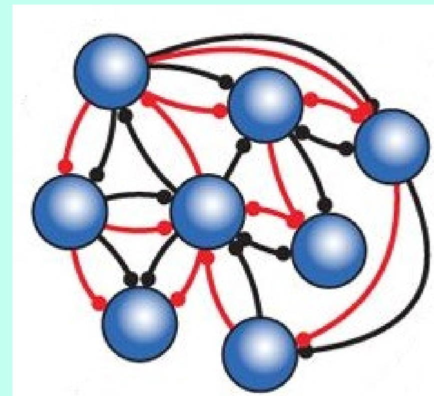
≤ 2|N| + 2 spikes
i + 1 time steps

Computing with neuromorphic oracle

Input

Computation





Computation on M_L
taking resources $\mathbf{R_A}$

Computing on S
taking resources $\mathbf{R_S}$

Output

- More powerful alternative: use S as co-processor
- Formally: S is an *oracle* for Turing machine M_L
- TM-using-SNN-oracle-model: $[M^S]$

Some first theoretical results

- **“Porting” of traditional complexity apparatus**
- Resource-preserving **reductions** from problem A to problem B (like polynomial many-one reductions)
- Limiting resources: clock, ruler → also **meter**
- **Classes** based on resources for TM and SNN
- **Canonical hard** problems relative to constraints on time, energy, and space

Some first theoretical results

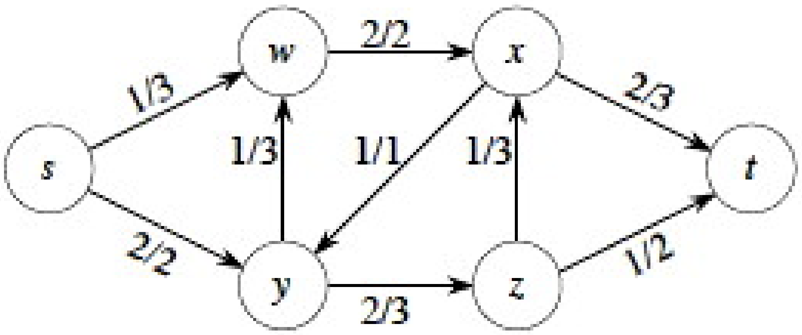
- Canonical complete problems for deterministic Turing Machines:
  - **Time-constrained halting**:
    Given a TM, an input *i* for that machine, and a number *T*, does that machine halt on that input within the first *T* steps?
    (P-completeness if TM is deterministic and *T* in unary notation; if *T* is in binary: EXP-completeness)

- Canonical complete problem for SNNs [M ° S]
  - **M ° S-Halting**
    Given an M ° S-machine, an input *i* for that machine, and resource limits *t* and *e* in unary notation; in the network S_i constructed by M on input i, does N_{acc} fire before time step *t* using energy at most *e*?

  - For oracle machines [M^S] proving a complete problem is more difficult (needs generic Cook-like reduction)!
    - Describe behaviour of M as a satisfiability variant and include in the formula the string of actual oracle answers (the “oracle prophesy”)

A more natural problem

Max network flow problem



This problem is P-complete, meaning that it cannot be efficiently parallellized (use only logarithmic space) on traditional machines!

MAX NETWORK FLOW is in $\mathsf{L}^{\mathsf{SNN}(\mathcal{O}(n),\mathcal{O}(n),\mathcal{O}(n))}$

It can be solved in Logspace when allowed to use a neuromorphic co-processor!

MSc project Abdullahi Ali

https://arxiv.org/abs/1911.13097

THRESHOLD NETWORK FLOW WITH RESERVOIRS *is* $\mathcal{O}(1) \circ \mathsf{SNN}(\mathcal{O}(1), \mathcal{O}(n), \mathcal{O}(n))$*-hard.*

But not efficiently on a neuromorphic system alone!

Relevance for NICE research field

- Contribute formal apparatus / theory helps neuromorphic systems to **mature**
- Examples of **programming** (rather than training) SNNs, temporal computation design patterns, in the future: abstraction to programming paradigm
- Provide a formal means of assessing **hardness** or **tractability** of problems (in addition to benchmarks)
- Show the relation (or mismatch...) between formal **theory and practice**! (e.g. keeping a neuron at sub-threshold potential is not free…)

Future work

- Build a bigger arsenal of motivs / examples for basic building blocks (searching, sorting, selecting etc.)

- New problems: genetic algorithms, dominating set

- Outreach to programming education / learning how to design network circuits, “think temporally”

- Investigate stochastic models of computation
- Investigate amortized costs (create-and-use)
- Investigate ‘local changes’ in the network

- Include costs of silence, communication / readout, and compare theory with hardware implementation