Planning with the Assembly Calculus

Francesco d'Amore

COATI

Inria
INVENTEURS DU MONDE NUMÉRIQUE

Joint work with

Pierluigi Crescenzi
Gran Sasso Science Institute

Daniel Mitropolsky
Columbia University

Emanuele Natale
COATI, CNRS

Christos Papadimitriou
Columbia University

COATI'S SEMINAR
19/10/2021

Brain & Mind

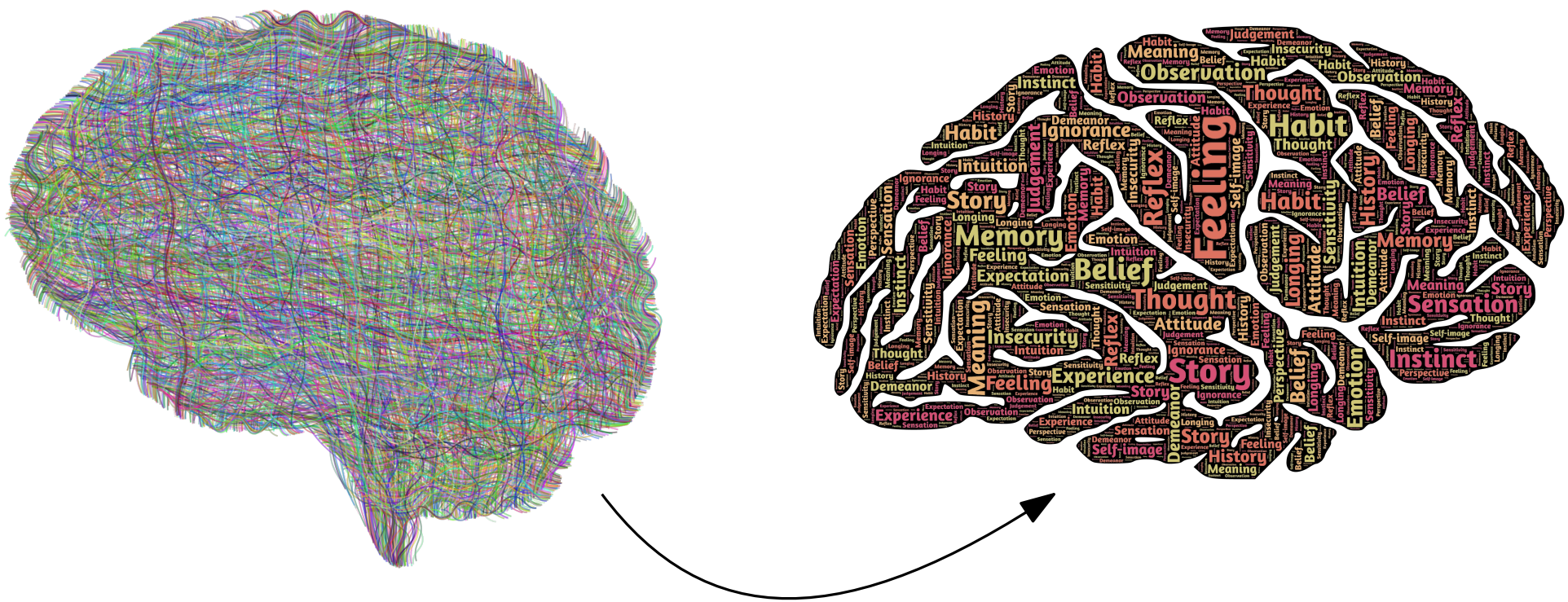
How does the brain beget the mind?

How do the intelligence, reasoning, languange (etc.) emerge from neurons and synapses?

Brain & Mind

How does the brain beget the mind?

How do the intelligence, reasoning, languange (etc.) emerge from neurons and synapses?



Brain & Mind

Despite tremendous advances over the past decades in our understanding of neural mechanisms, still very far from answering

Difficulty: huge gap of scale and methodology between Experimental Neuroscience and Cognitive Science

Brain & Mind

Despite tremendous advances over the past decades in our understanding of neural mechanisms, still very far from answering

Difficulty: huge gap of scale and methodology between Experimental Neuroscience and Cognitive Science

Nobel laureate Richard Axel: *We do not have a* **logic** *for the transformation of neural activity to thought and action. I consider discerning [this* **logic***] as the most important future direction in Neuroscience* [Axel, Neuron 2018]

The Assembly Calculus

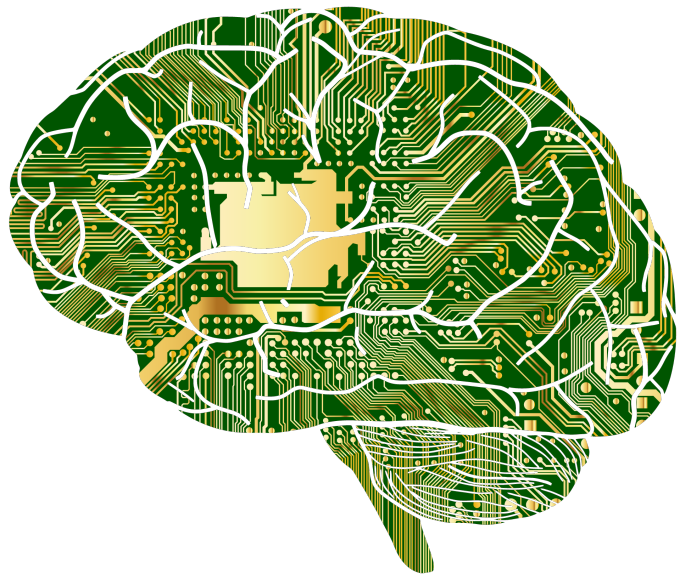
Recently proposed formal computational system [Papadimitriou et al., PNAS 2020]

The Assembly Calculus

Recently proposed formal computational system [Papadimitriou et al., PNAS 2020]

Explicit purpose of fitting the Axel's logic:

- bridging through computation the gap between neurons and intelligence



The Assembly Calculus

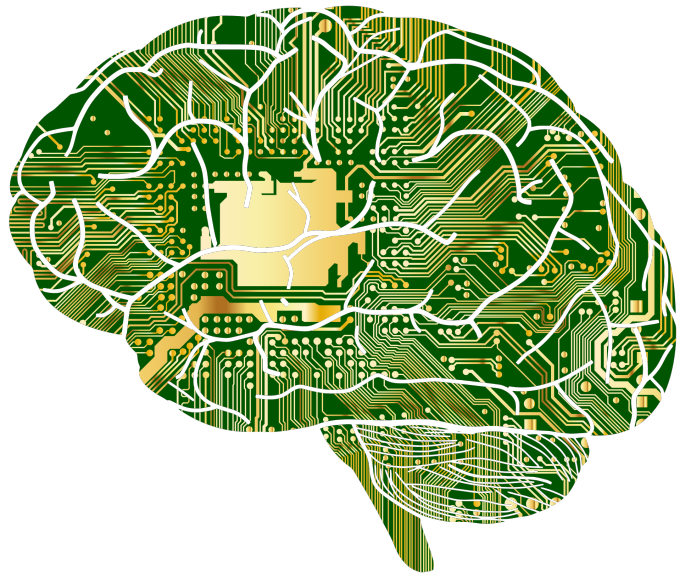
Recently proposed formal computational system [Papadimitriou et al., PNAS 2020]

Explicit purpose of fitting the Axel's logic:

- bridging through computation the gap between neurons and intelligence

Components:

- *assemblies*
- *operations*



Assemblies

Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

Assemblies

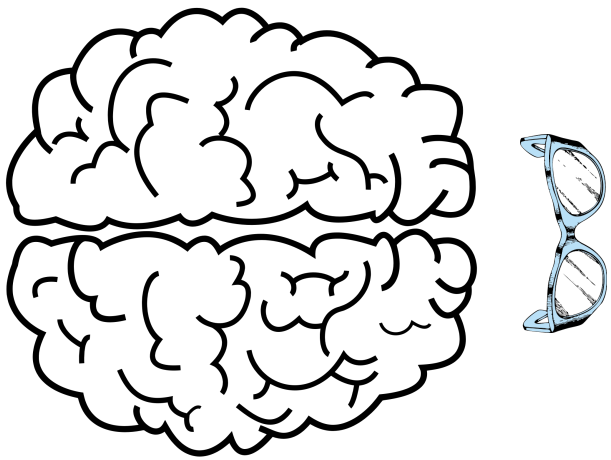
Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]

Assemblies

Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

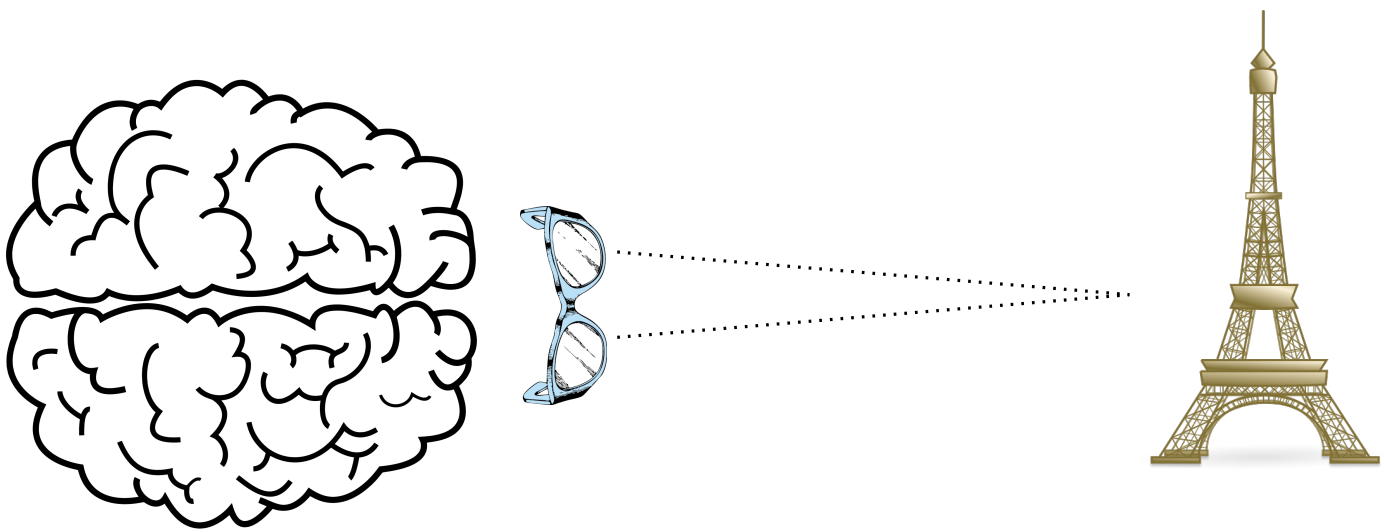
Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]



Assemblies

Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

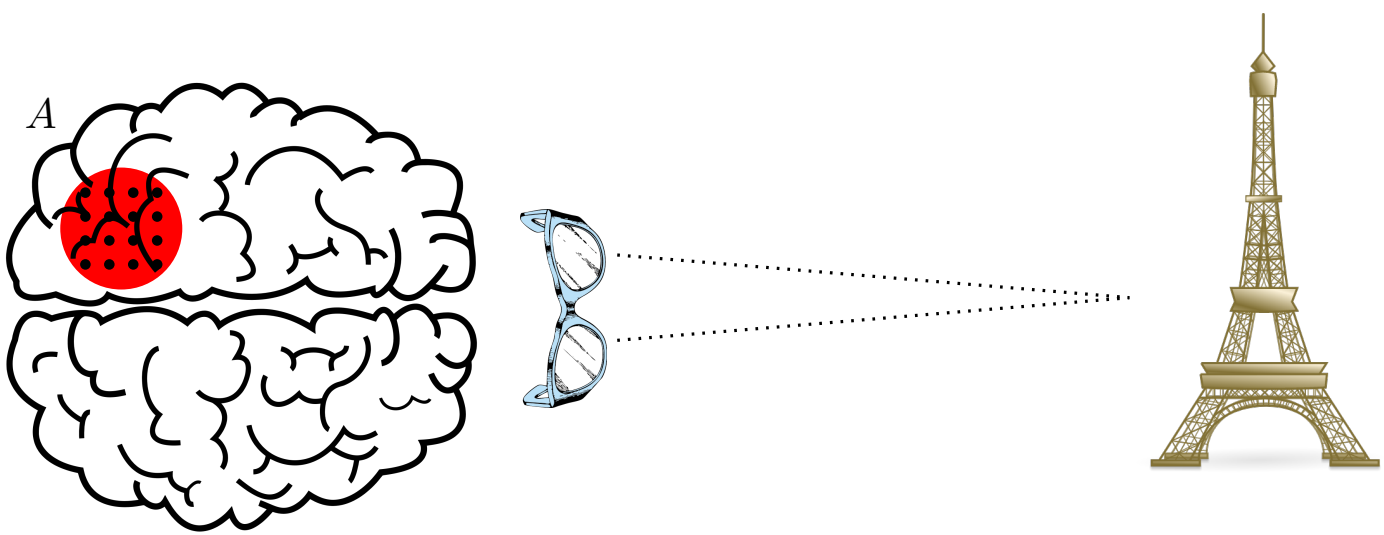
Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]



Assemblies

Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

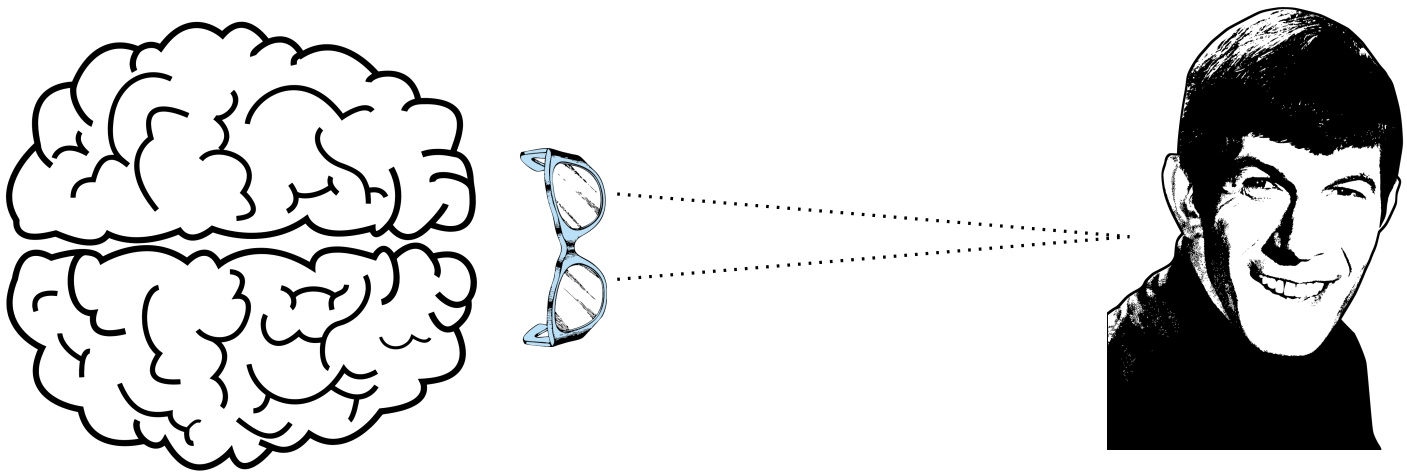
Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]



Assemblies

Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

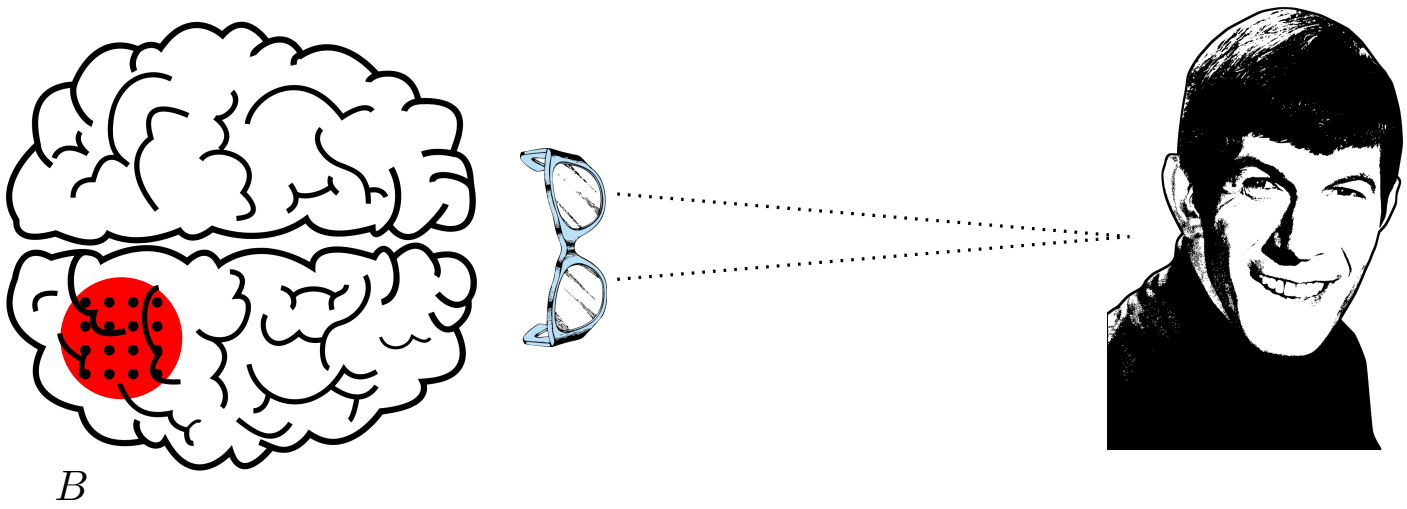
Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]



Assemblies

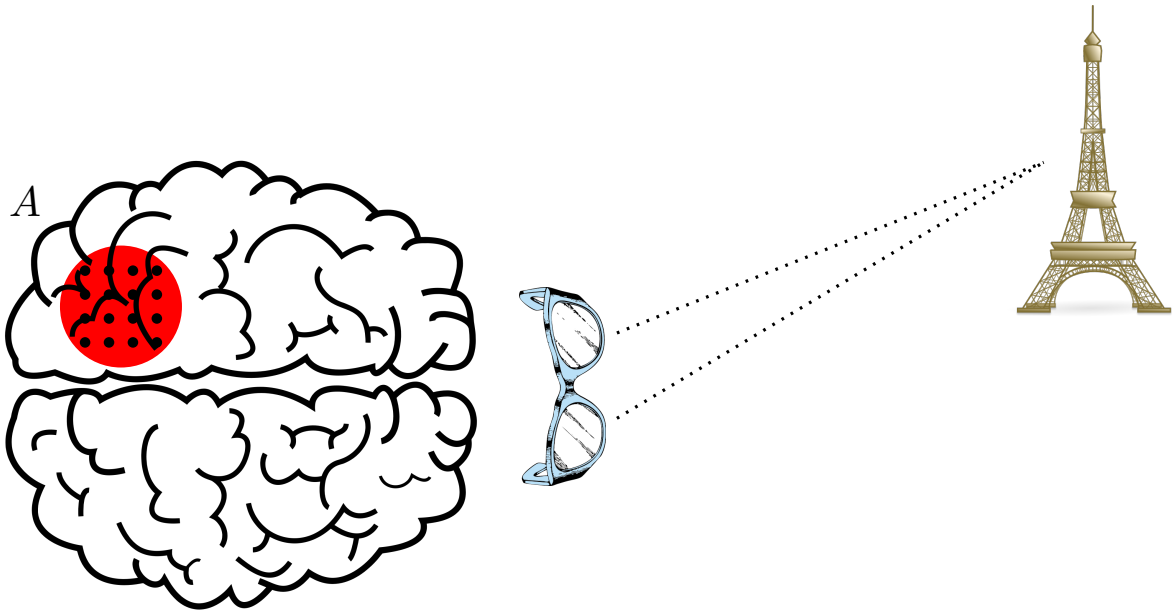
Hypothized by [Hebb, 1949]: densely interconnected sets of neurons whose loosely synchronized firing in a pattern is simultaneous with the subject thinking of a particular **concept** or **idea**

Confirmed by [Harris, Nat. Rev. Neurosci. 2005], [Buzsaki, Neuron 2010]



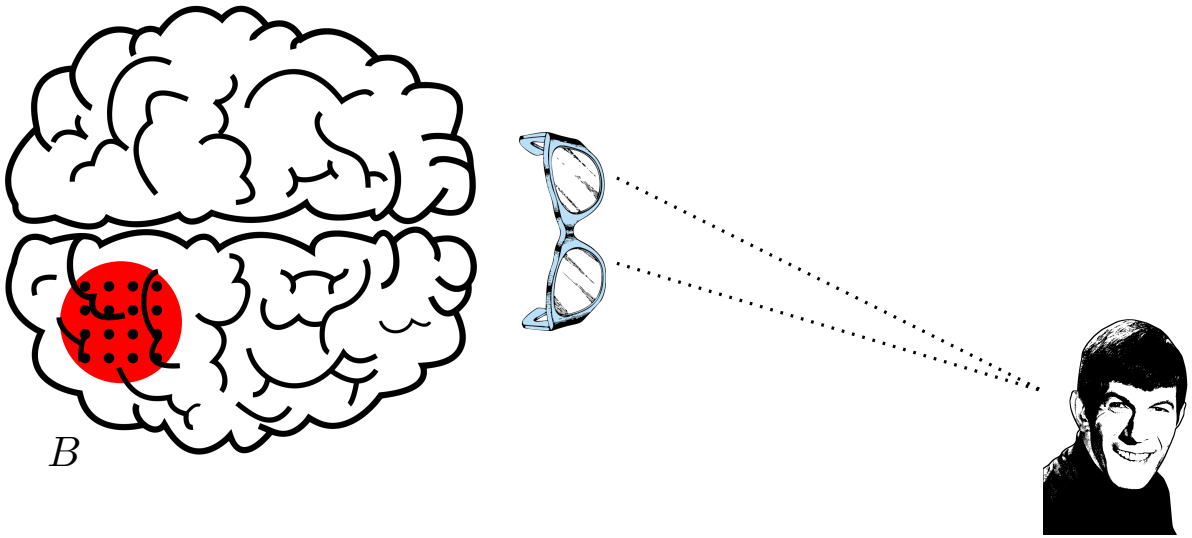
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



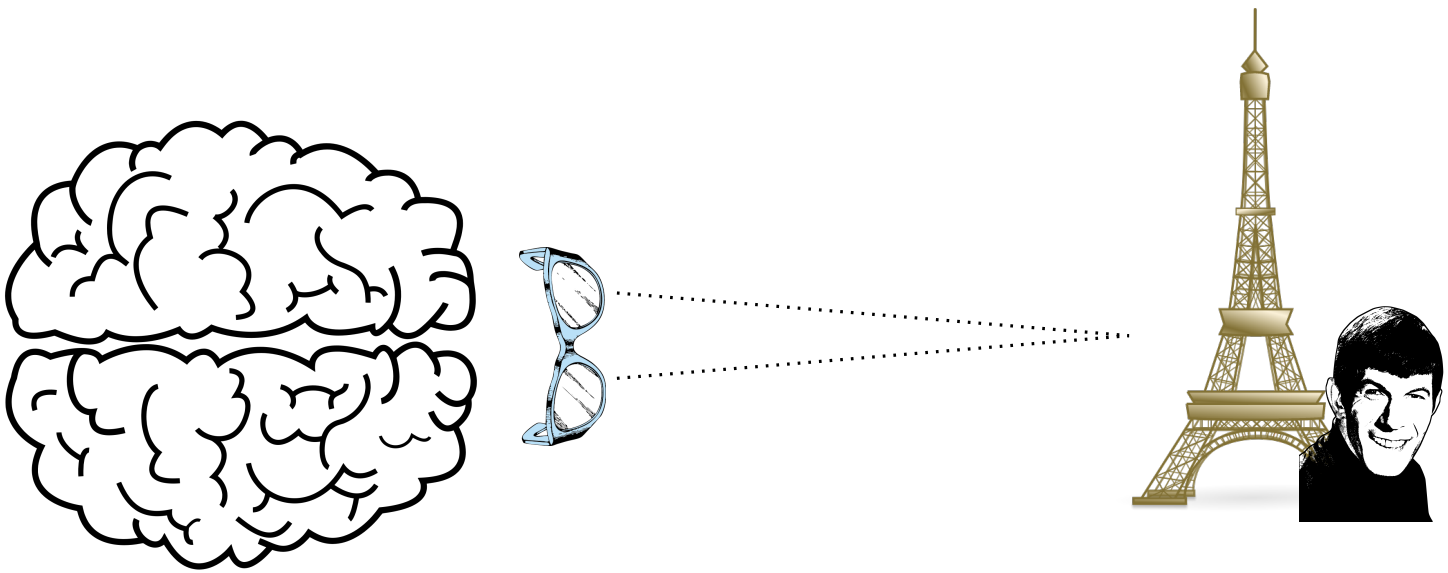
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



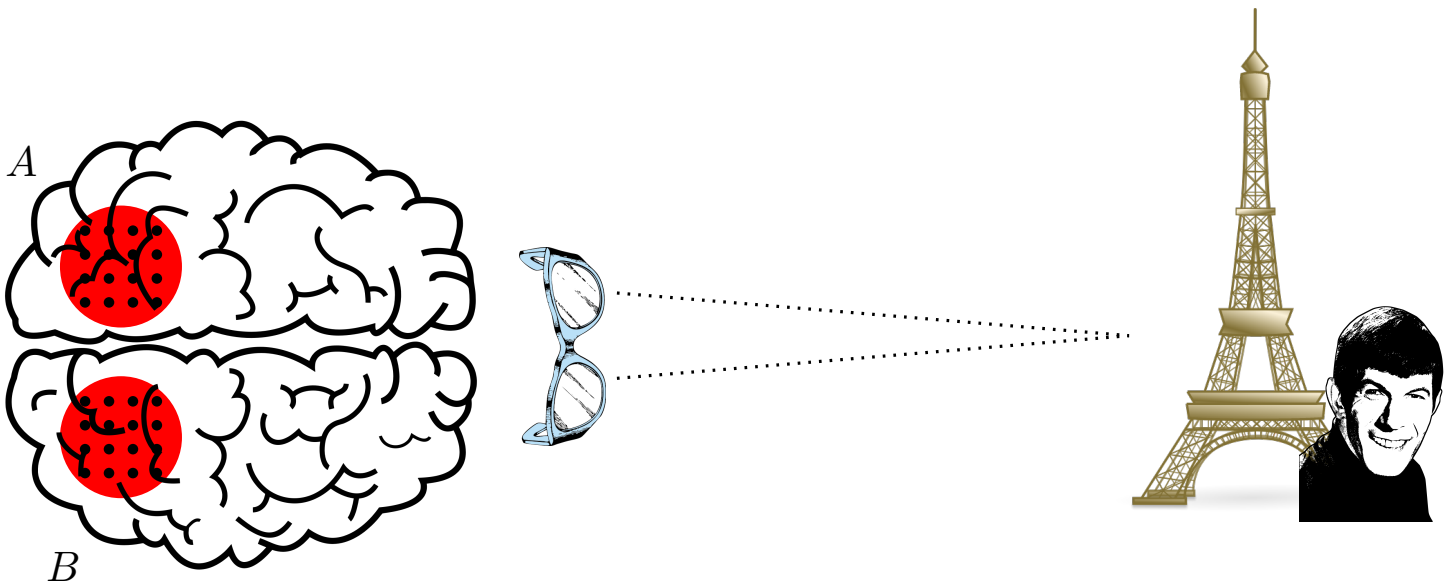
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



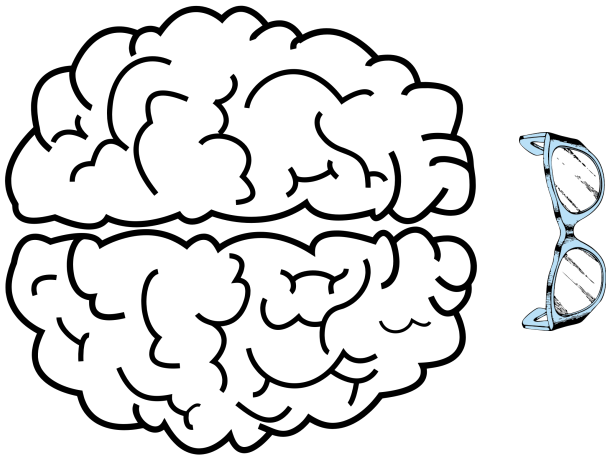
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



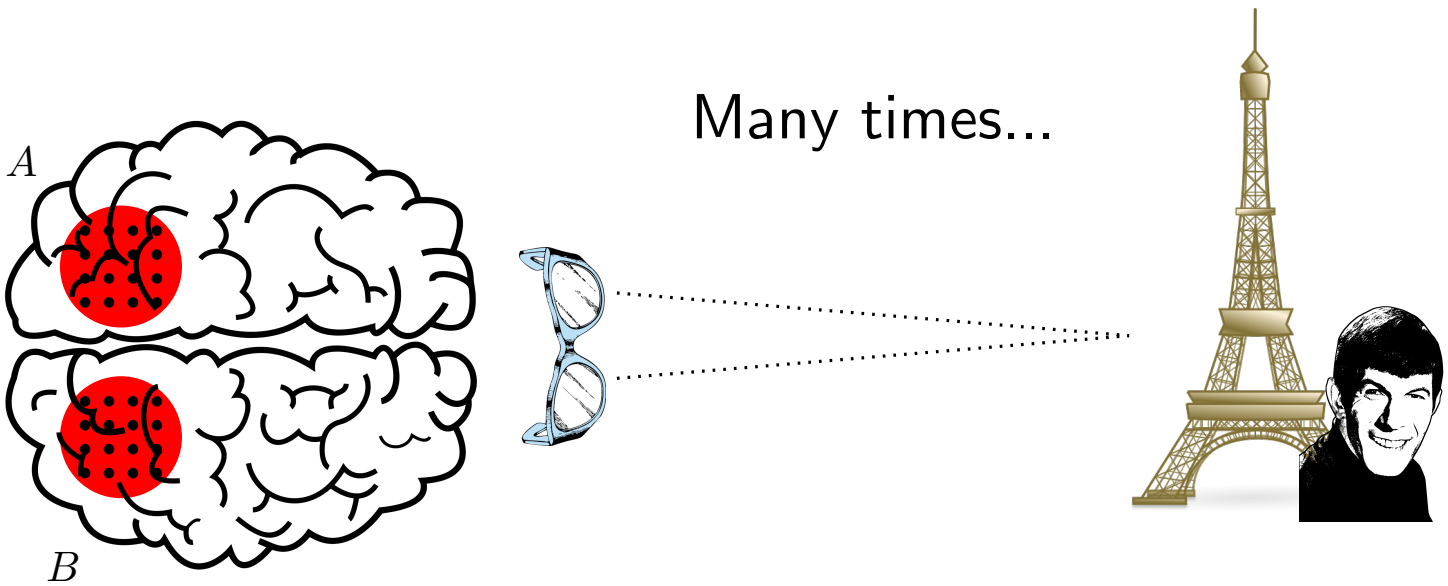
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



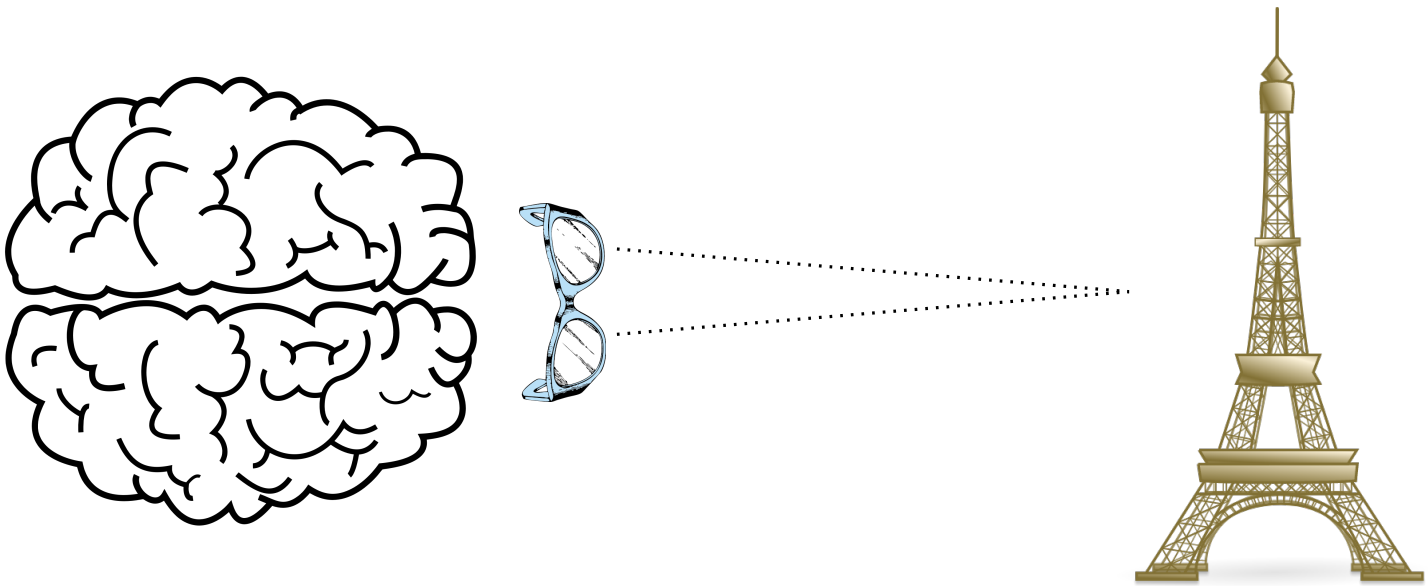
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



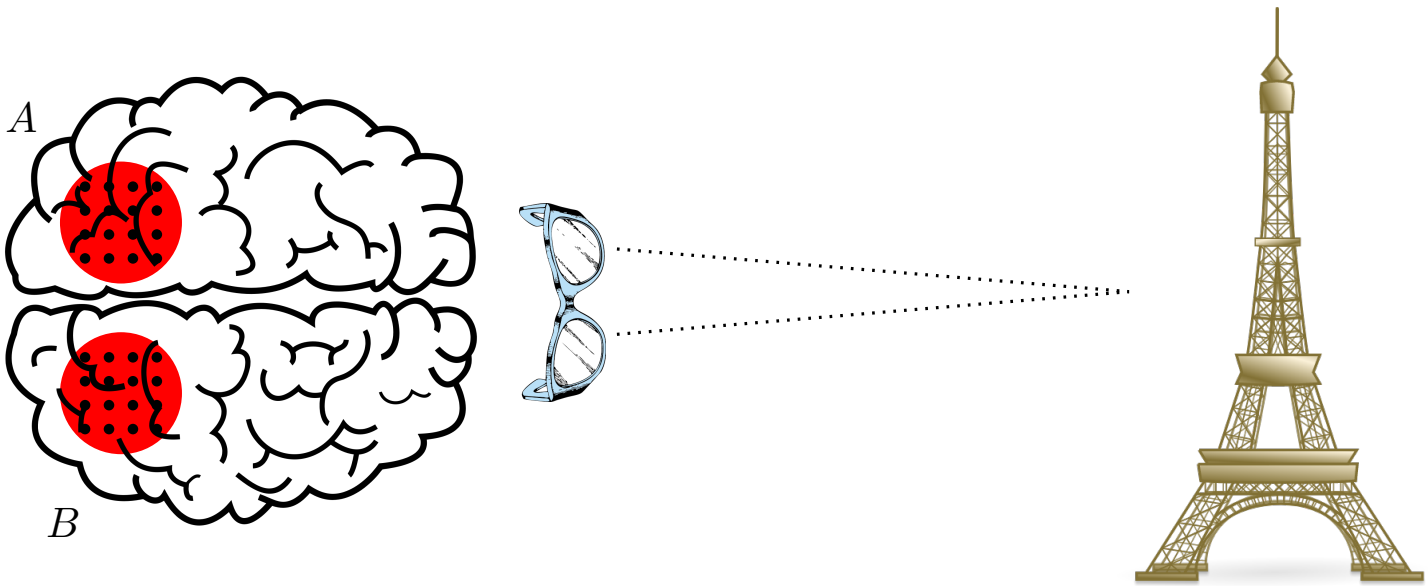
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



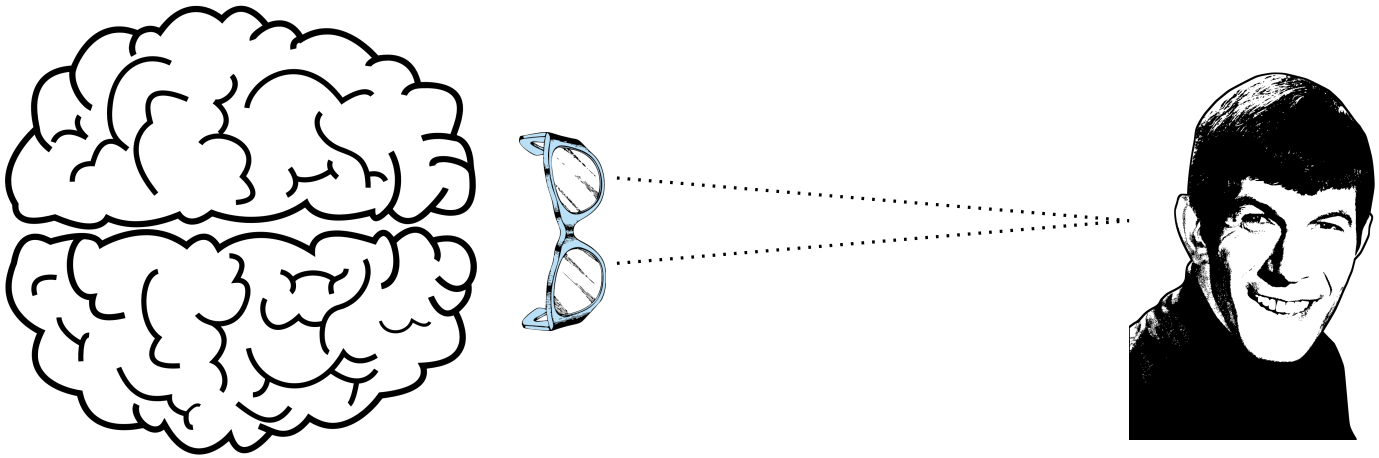
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



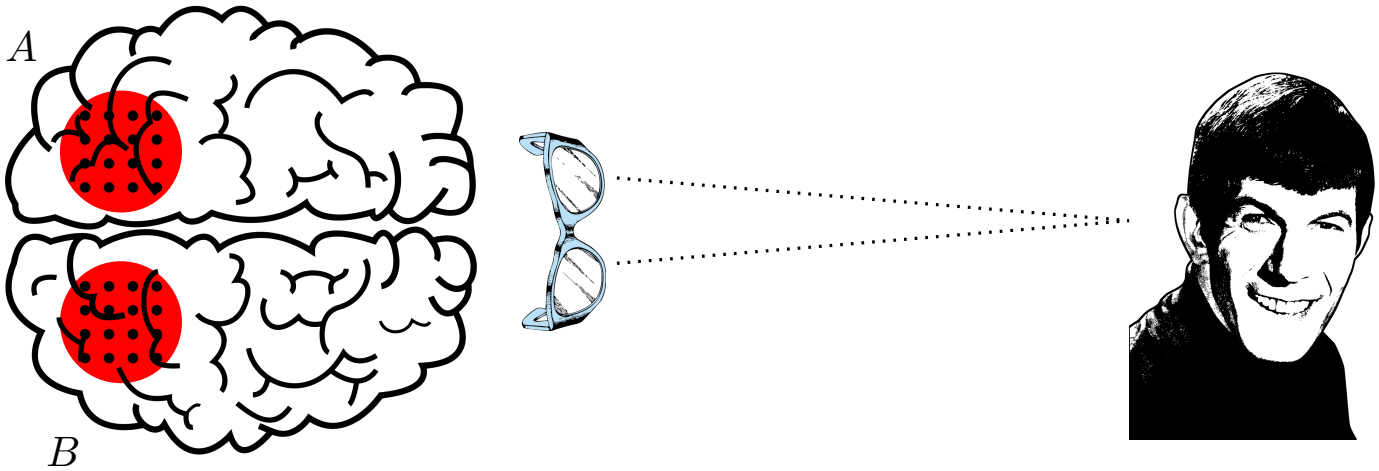
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



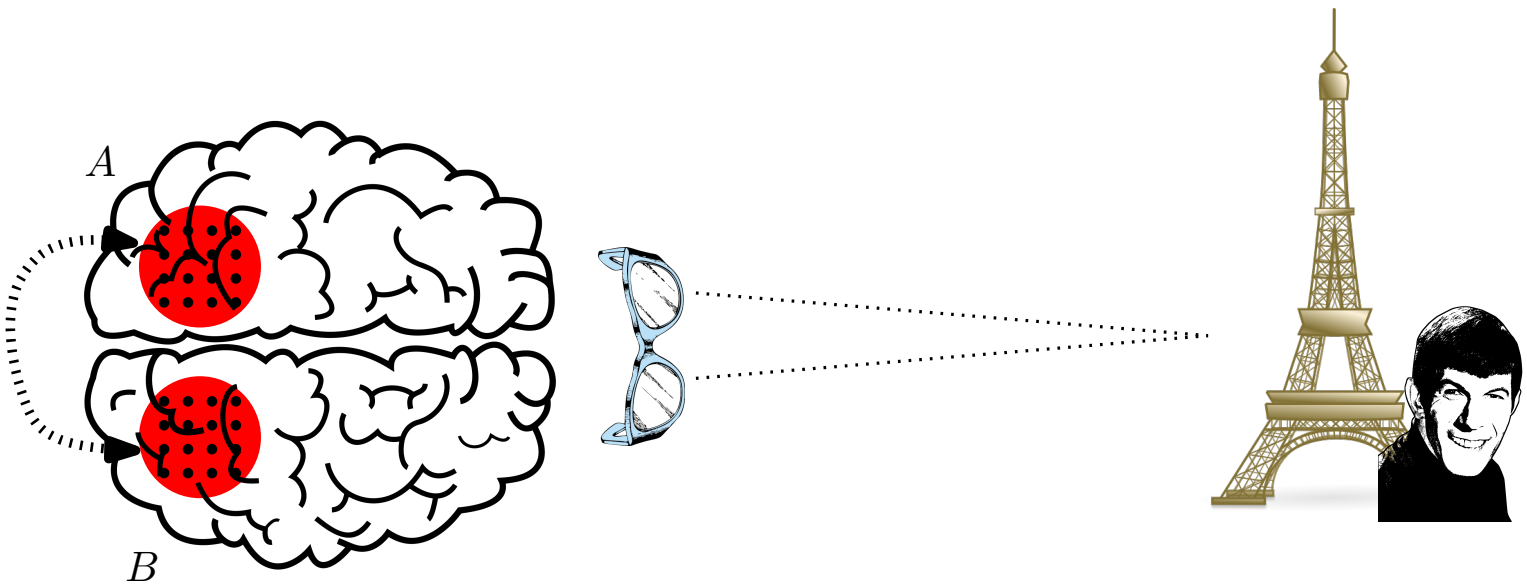
Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



Operations with Assemblies

Experiment by [Ison et al., Neuron 2015]



Association of assemblies (and others...)

The Model

- **Brain area**: $A = G_{n_A, p_A}$ Erdös Rényi graph

- *Recall* $G_{n,p} = (V, E)$:
  - $|V| = n$
  - $\forall x, y \in V$, $(x, y) \in E$ with probability p

The Model

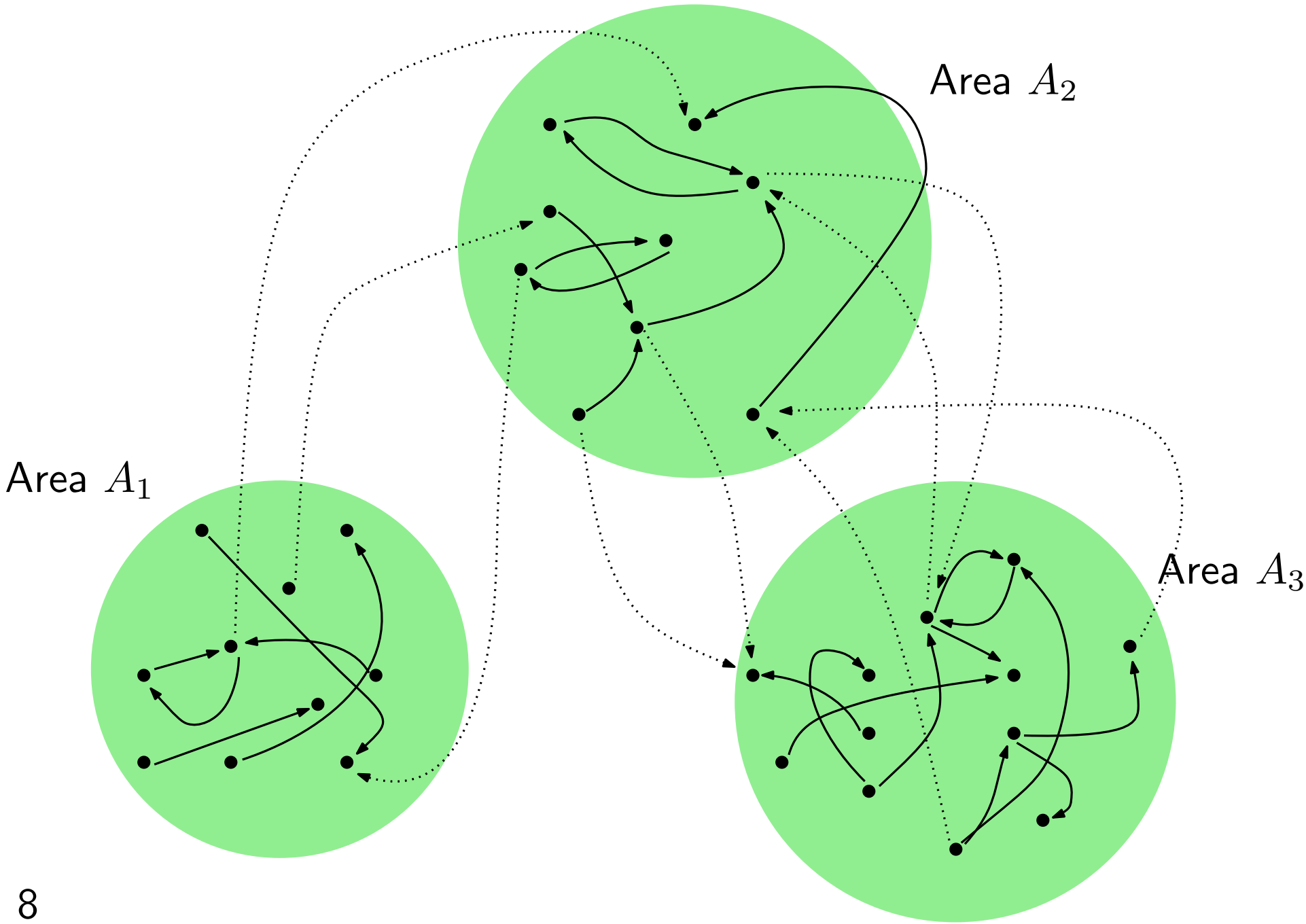
- **Brain area**: $A = G_{n_A, p_A}$ Erdös Rényi graph

- *Recall* $G_{n,p} = (V, E)$:
  - $|V| = n$
  - $\forall x, y \in V$, $(x, y) \in E$ with probability p

- set of brain areas $\mathcal{S} = \{A_1, A_2, \ldots, A_m\}$
- $\mathcal{C} \subseteq \mathcal{S} \times \mathcal{S}$ set of ordered pairs of brain regions

- **The Brain**: $\mathcal{B} = (V_{\mathcal{B}}, E_{\mathcal{B}})$ with
  - $V_{\mathcal{B}} = V_{A_1} \cup \cdots \cup V_{A_m}$
  - $E_{\mathcal{B}} = E \cup E_{A_1} \cup \ldots E_{A_m}$
  - E: $\forall (A, B) \in \mathcal{C}$, $\forall x \in A, y \in B$, $(x, y) \in E$ with probability $p_{A,B}$

Example of a Brain



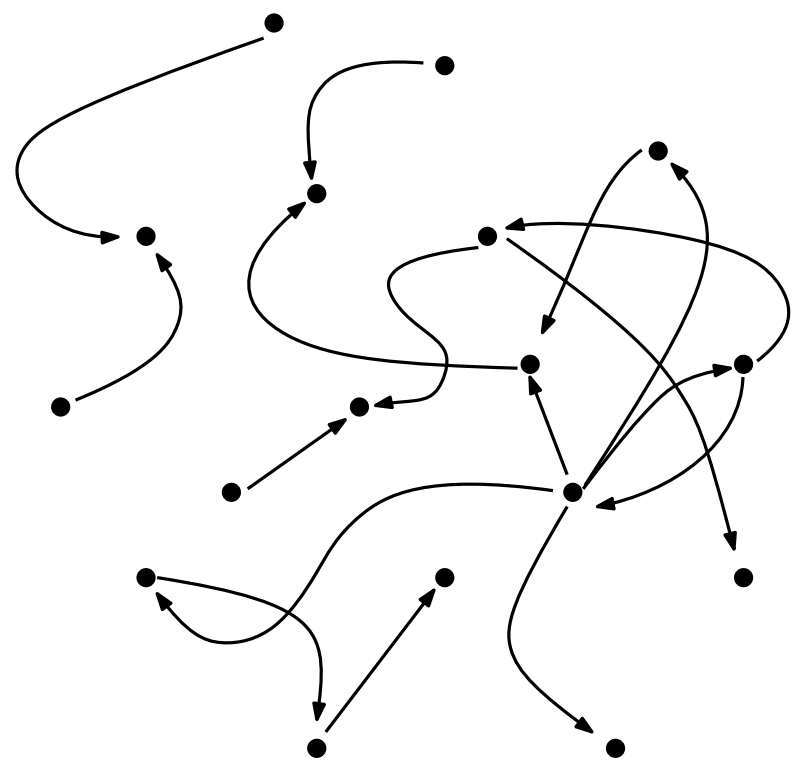
The Dynamics

Neuron = vertex **Fiber** = edge

All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired



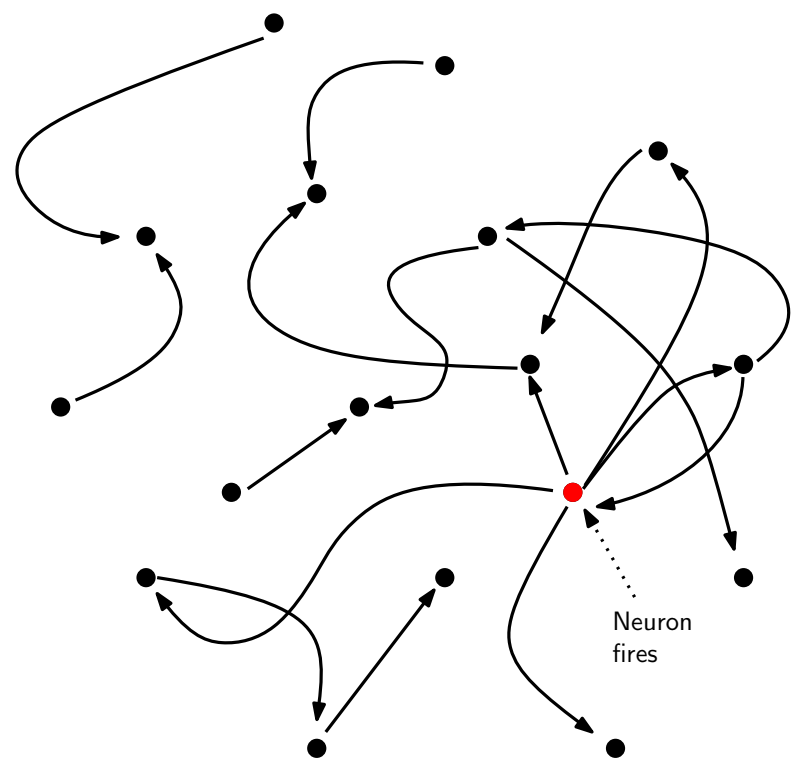
The Dynamics

Neuron = vertex **Fiber** = edge

All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired



The Dynamics

Neuron = vertex **Fiber** = edge

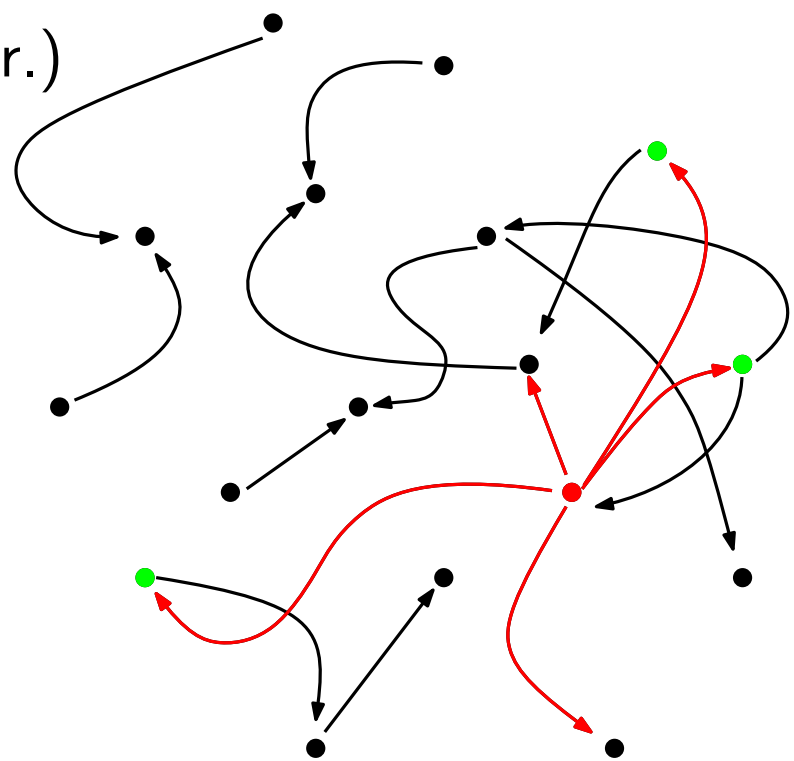
All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)



The Dynamics

Neuron = vertex **Fiber** = edge

All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

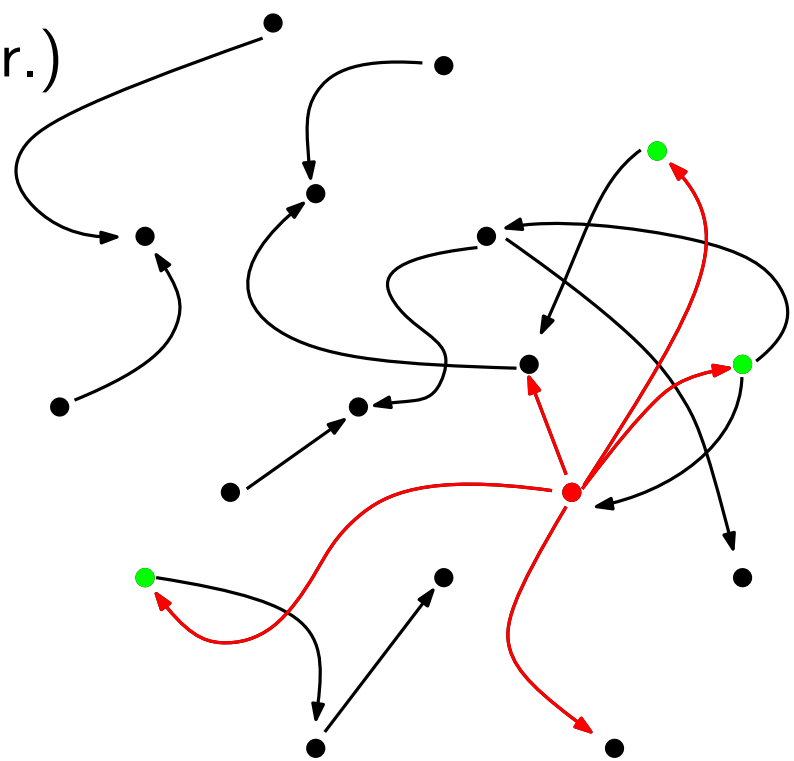
Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)

Upon fiber endpoint activation:

weight = old_weight $\cdot (1 + \beta)$

Hebbian plasticity



The Dynamics

Neuron = vertex **Fiber** = edge

All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

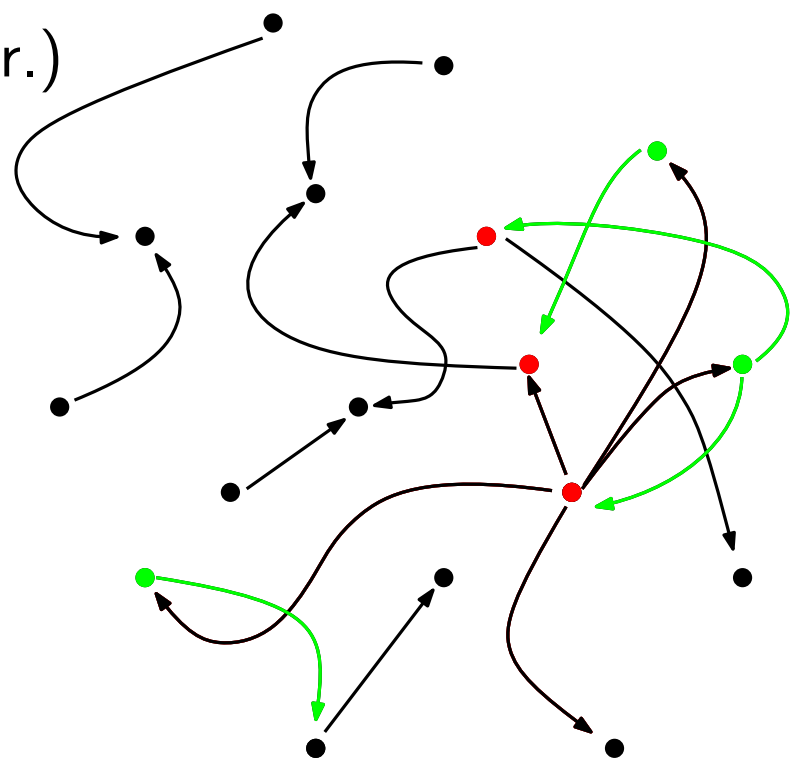
Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)

Upon fiber endpoint activation:

weight = old_weight $\cdot (1 + \beta)$

Hebbian plasticity



The Dynamics

Neuron = vertex **Fiber** = edge

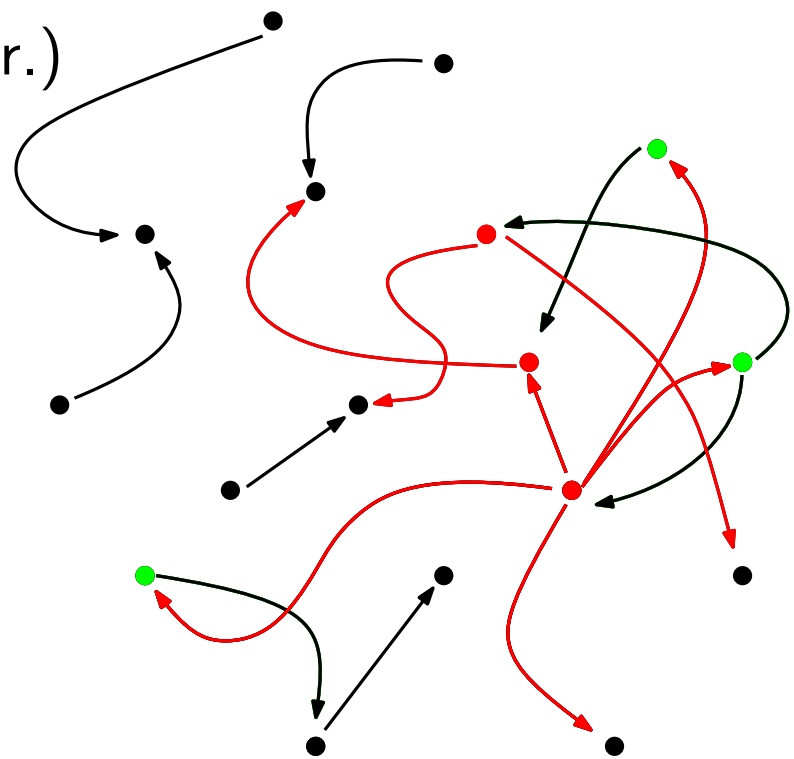
All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)



Upon fiber endpoint activation:

weight = old_weight $\cdot (1 + \beta)$

Hebbian plasticity

The Dynamics

Neuron = vertex **Fiber** = edge

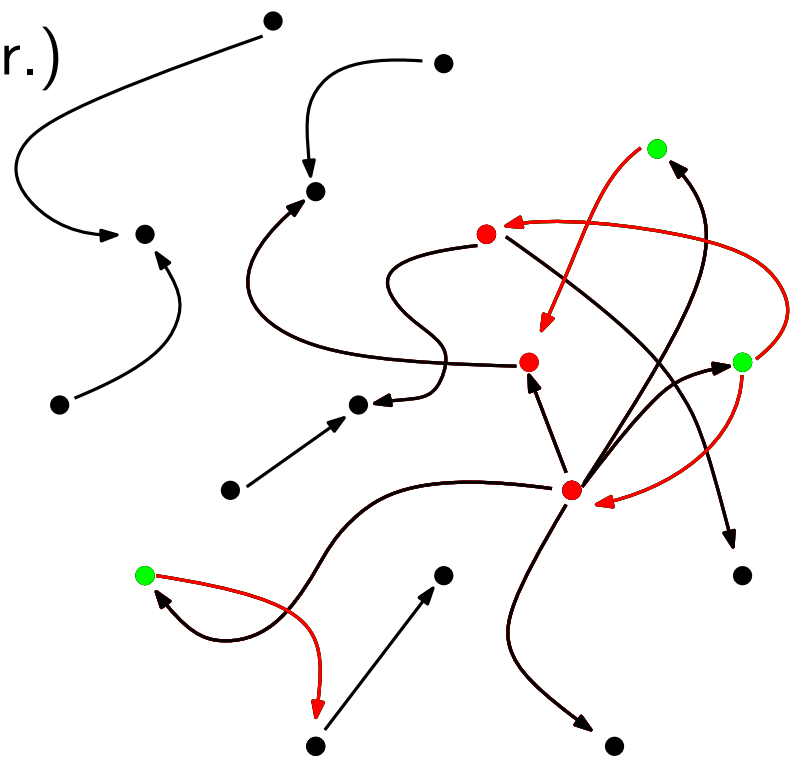
All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)



Upon fiber endpoint activation:

weight = old_weight $\cdot (1 + \beta)$

Hebbian plasticity

The Dynamics

Neuron = vertex **Fiber** = edge

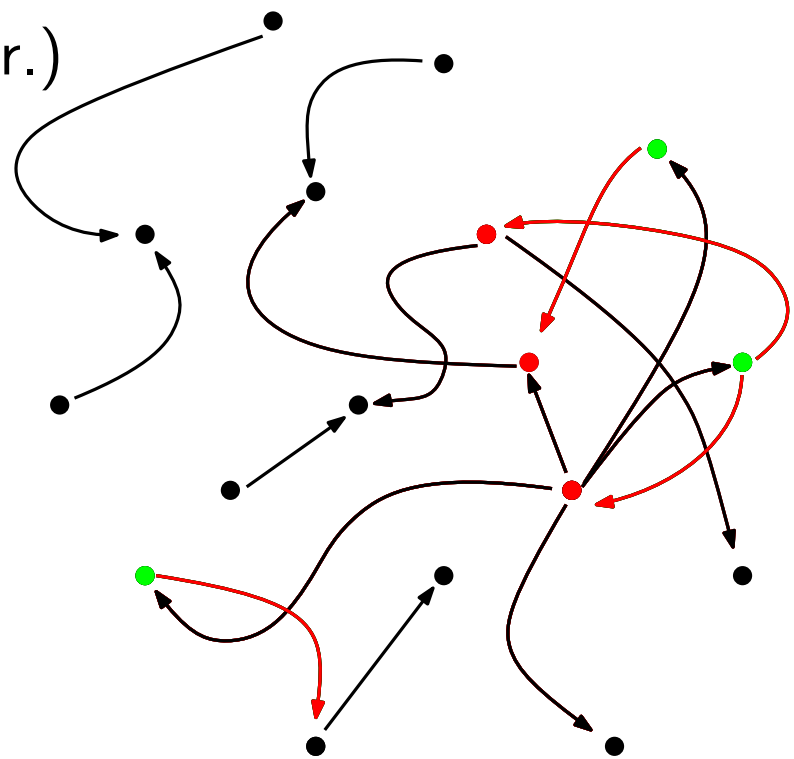
All fiber weights initialized to be 1

Input for a neuron:

- external input (number)
- sum of incoming fiber weights whose sources have fired

Neuron fires if it is among the k neurons with the highest input (**CAP**)

Here (toy): $k = 3$ (ties broken u.a.r.)



Upon fiber endpoint activation:

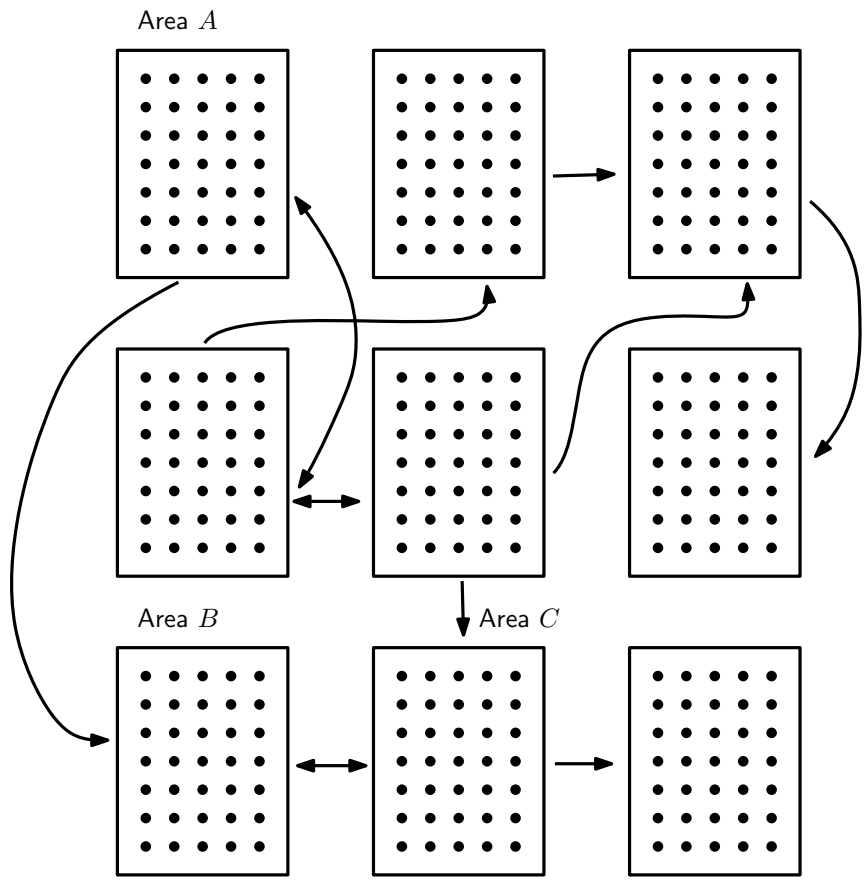
weight = old_weight $\cdot (1 + \beta)$

Hebbian plasticity

Assembly: *stable* set of k neurons

Operations

The brain and its areas

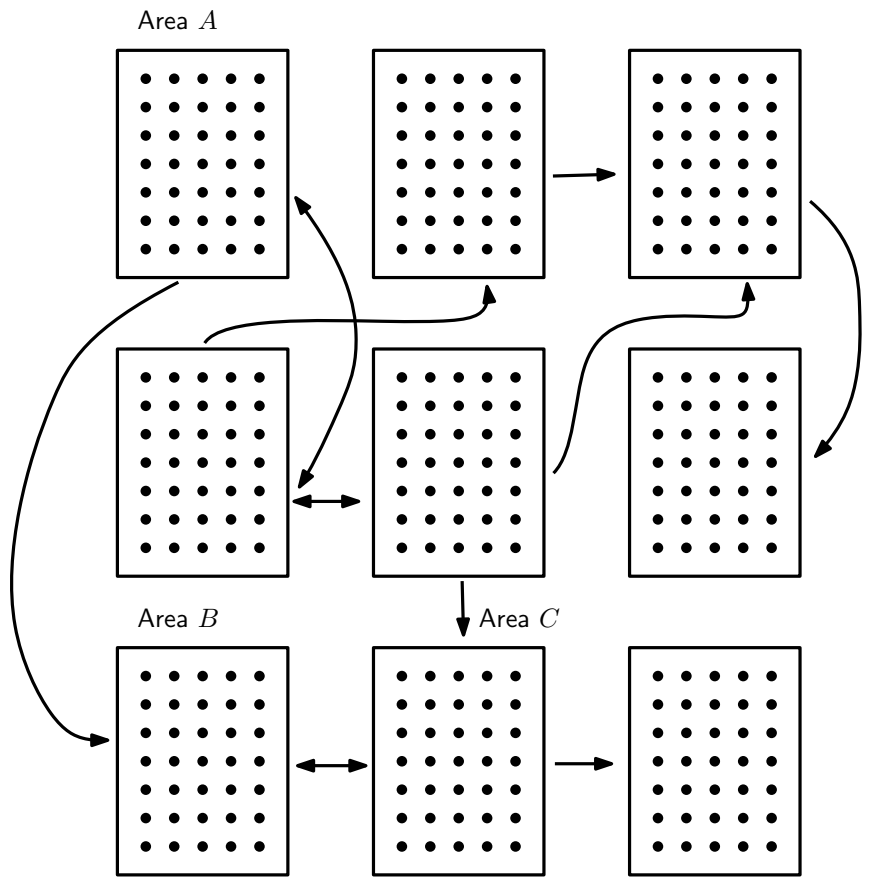


Operations

- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

The brain and its areas



Operations

- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

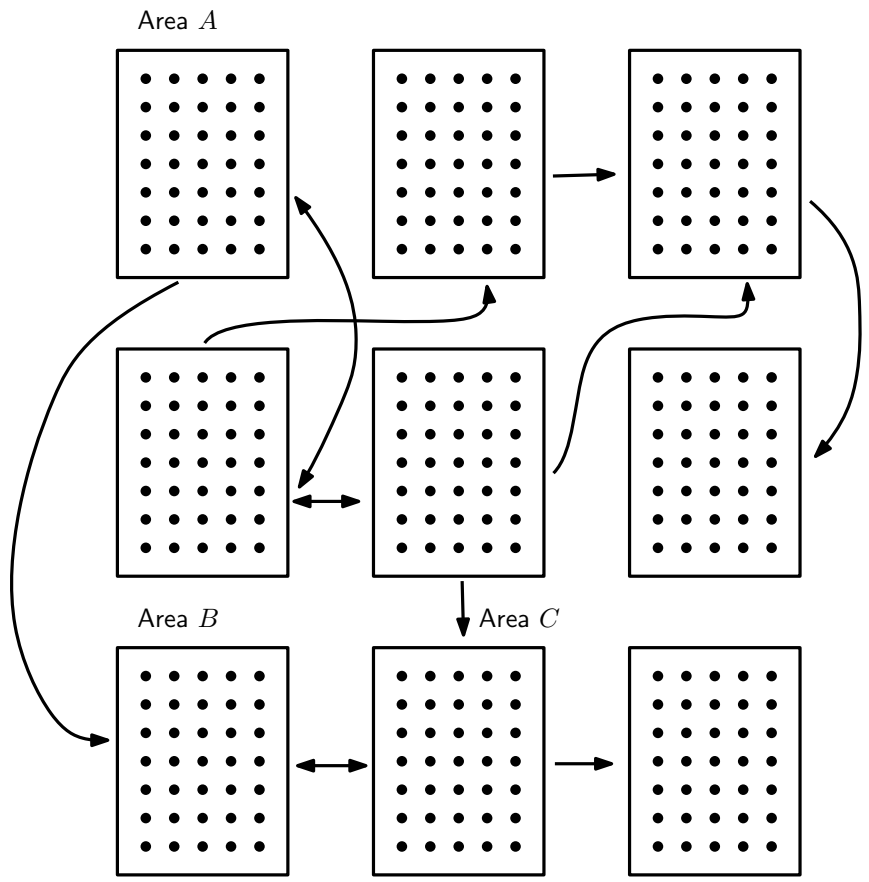
Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

- **Disinhibition**: an inhibited area can be disinhibited, that is, its neurons can now fire

Disinhibition inhibits a population of inhibitory neurons, which currently inhibit the area [Mitropolsky et al., TACL 2021]

We assume we can do the same with fibers

The brain and its areas



Operations

- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

  Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

- **Disinhibition**: an inhibited area can be disinhibited, that is, its neurons can now fire

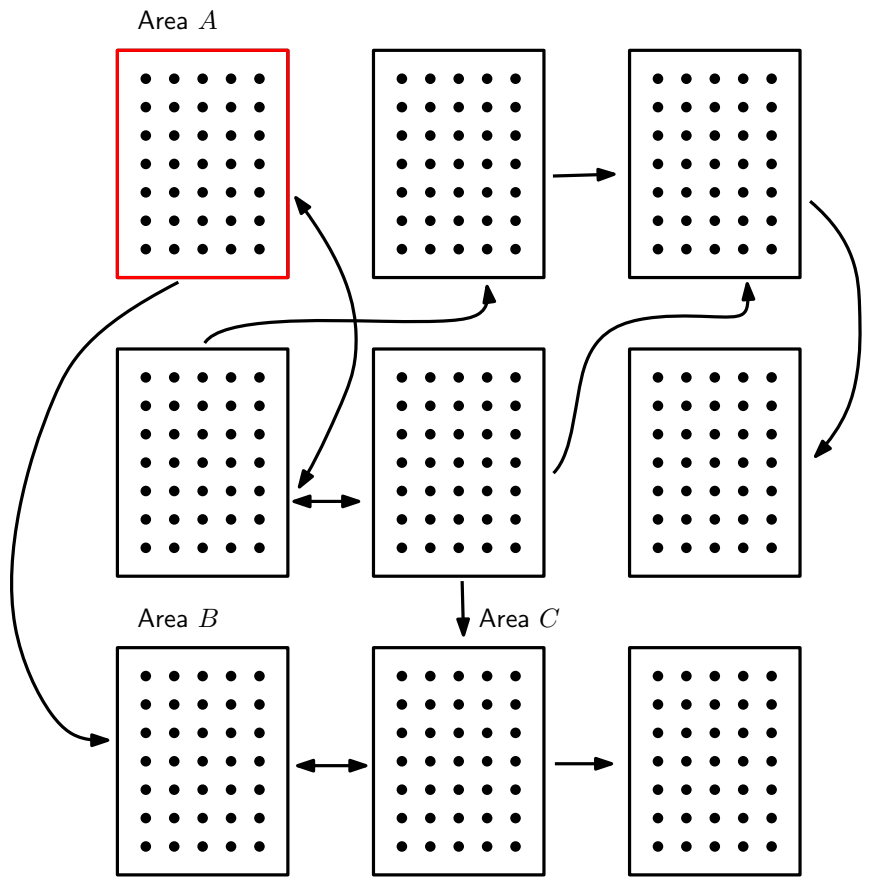
  Disinhibition inhibits a population of inhibitory neurons, which currently inhibit the area [Mitropolsky et al., TACL 2021]

We assume we can do the same with fibers

For areas A, B, C:

- `inhibitArea` (A)

The brain and its areas



Operations

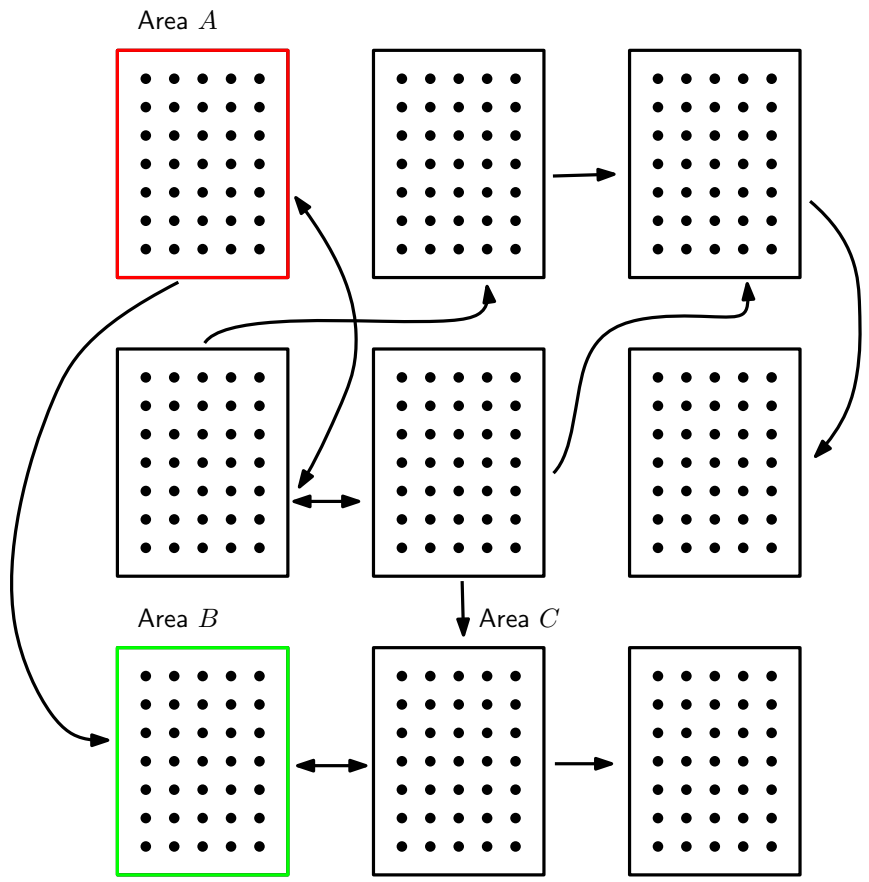
- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

  Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

- **Disinhibition**: an inhibited area can be disinhibited, that is, its neurons can now fire

  Disinhibition inhibits a population of inhibitory neurons, which currently inhibit the area [Mitropolsky et al., TACL 2021]

The brain and its areas



We assume we can do the same with fibers

For areas A, B, C:

- `inhibitArea` (A)
- `disinhibitArea` (B)

Operations

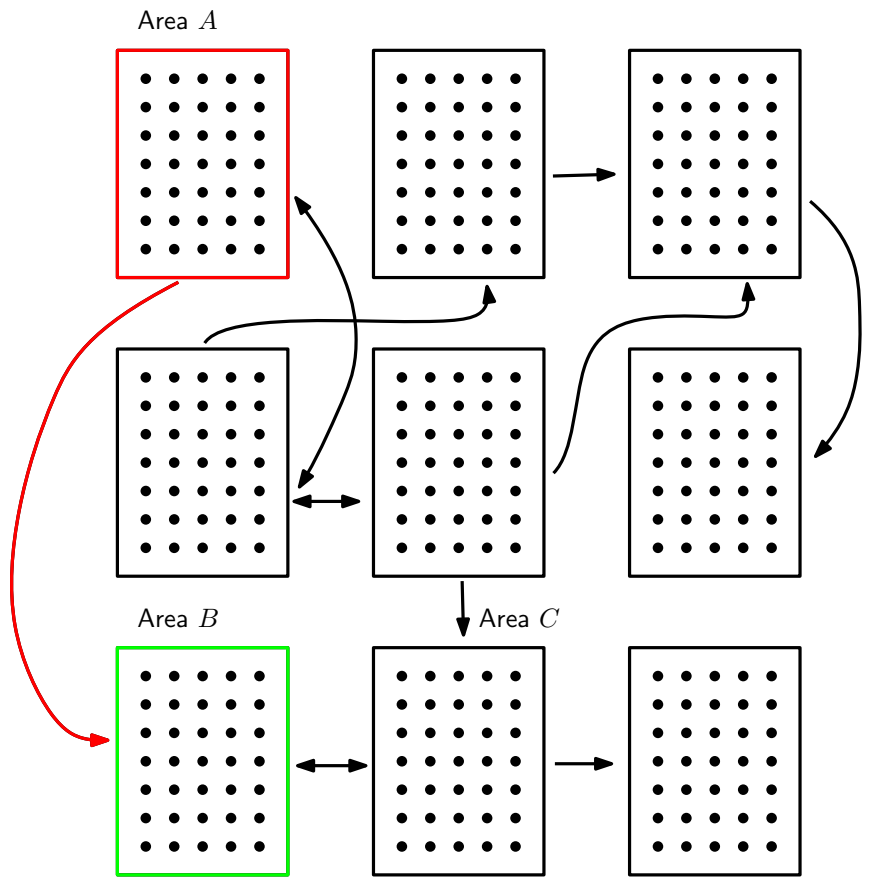
- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

- **Disinhibition**: an inhibited area can be disinhibited, that is, its neurons can now fire

Disinhibition inhibits a population of inhibitory neurons, which currently inhibit the area [Mitropolsky et al., TACL 2021]

The brain and its areas



We assume we can do the same with fibers

For areas A, B, C:

- `inhibitArea` (A)
- `inhibitFiber` (A, B)
- `disinhibitArea` (B)

Operations

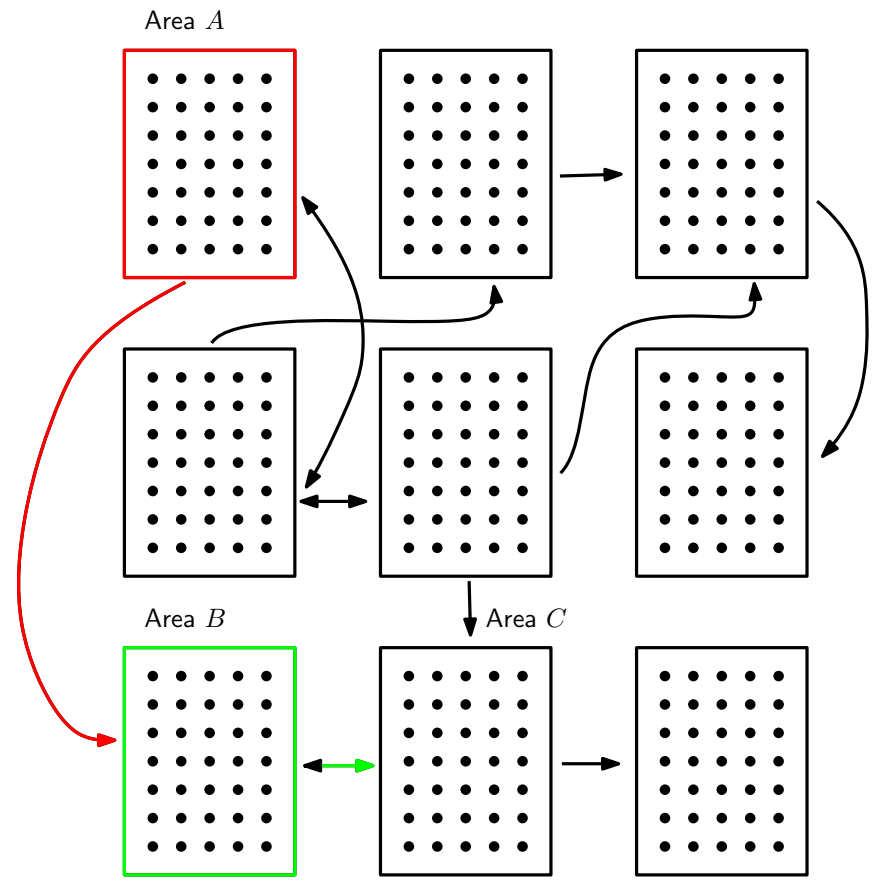
- **Inhibition**: an area can be inhibited, that is, its neurons cannot fire

Inhibition is accomplished through populations of inhibitory neurons, whose firing prevents other neurons from firing [Mitropolsky et al., TACL 2021]

- **Disinhibition**: an inhibited area can be disinhibited, that is, its neurons can now fire

Disinhibition inhibits a population of inhibitory neurons, which currently inhibit the area [Mitropolsky et al., TACL 2021]

The brain and its areas



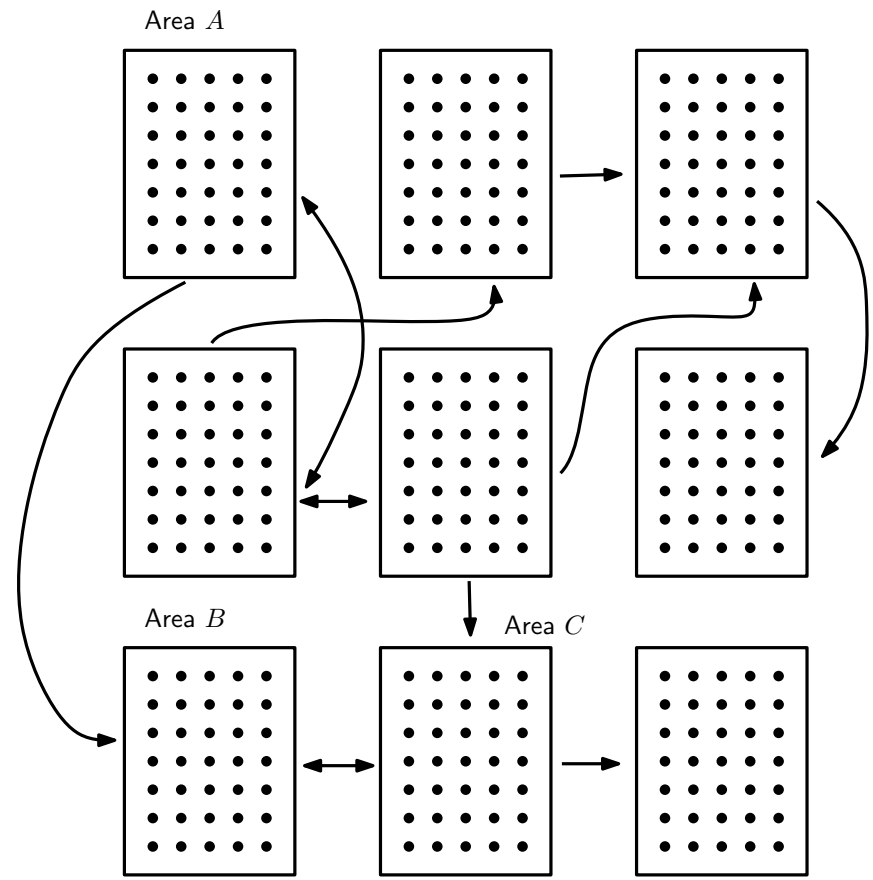
We assume we can do the same with fibers

For areas A, B, C:

- `inhibitArea` (A)
- `inhibitFiber` (A, B)
- `disinhibitArea` (B)
- `disinhibitFiber` (B, C)

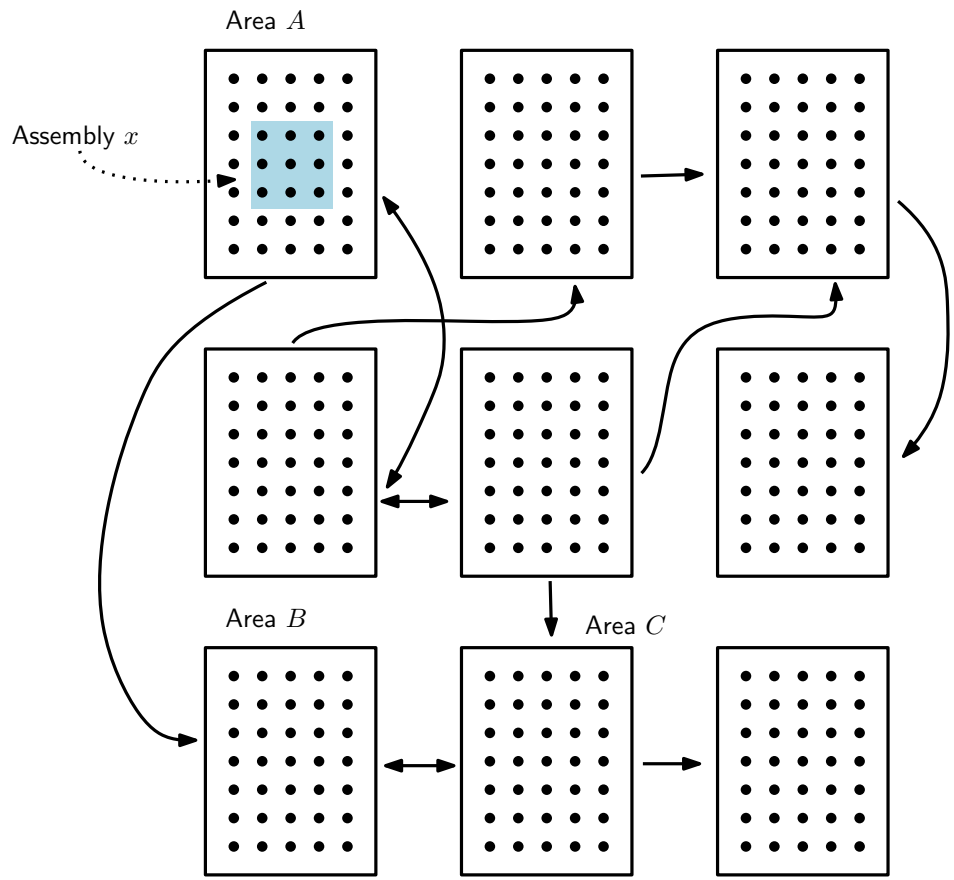
Operations

The brain and its areas



Operations

The brain and its areas

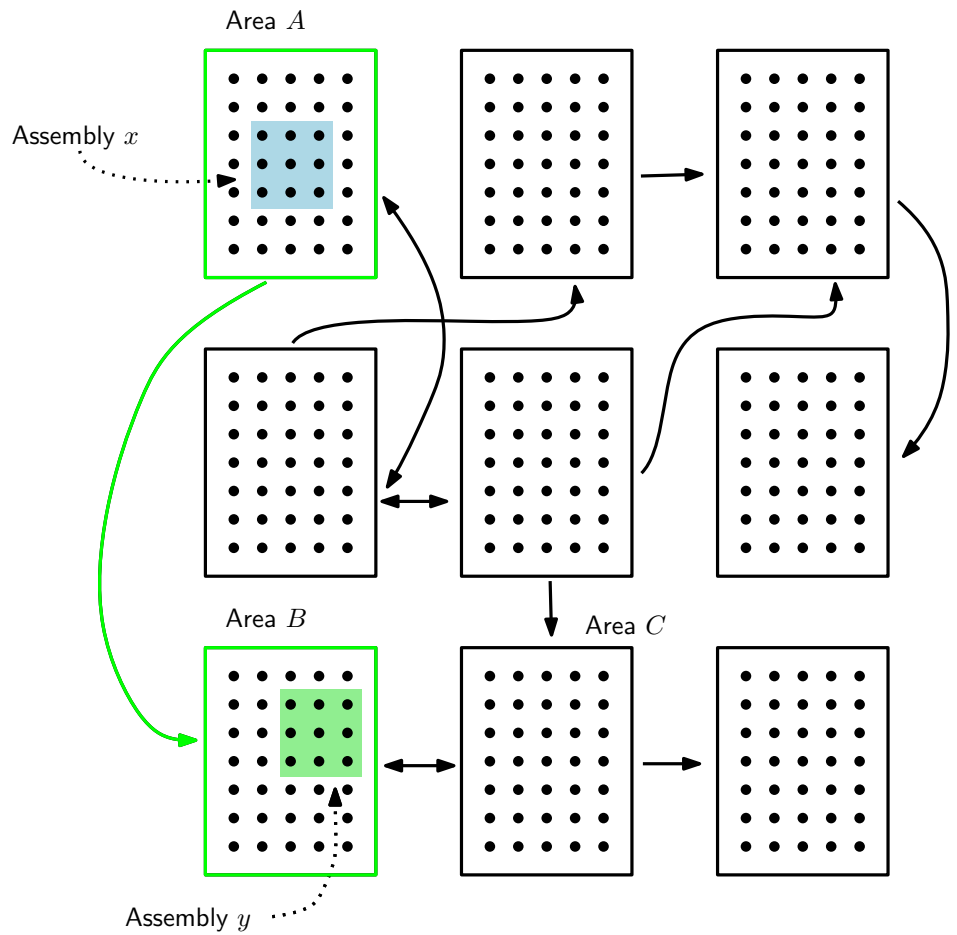


Operations

- **Projection**: the last active assembly x in area A starts firing repeatedly into area B until an assembly y is formed

This process eventually converges under a wide range of parameters w.h.p. [Papadimitriou et Vempala, ITCS 2019]

The brain and its areas

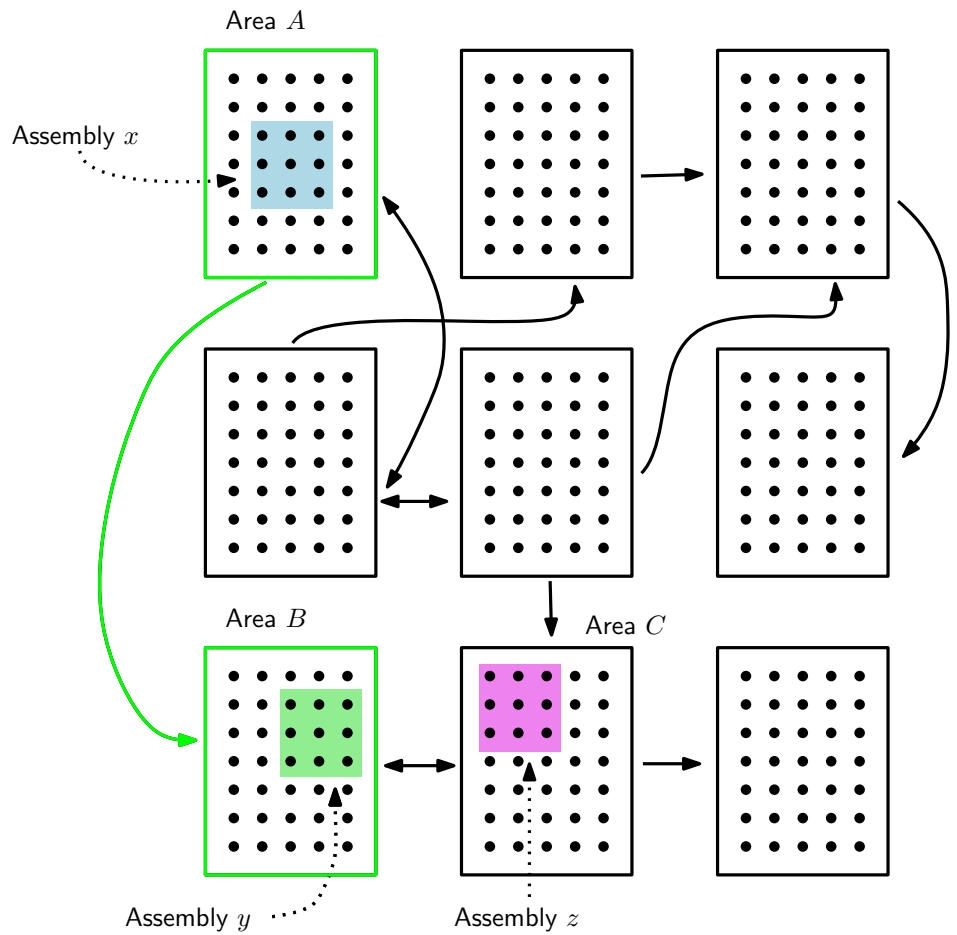


Operations

- **Projection**: the last active assembly x in area A starts firing repeatedly into area B until an assembly y is formed

This process eventually converges under a wide range of parameters w.h.p. [Papadimitriou et Vempala, ITCS 2019]

The brain and its areas



Operations

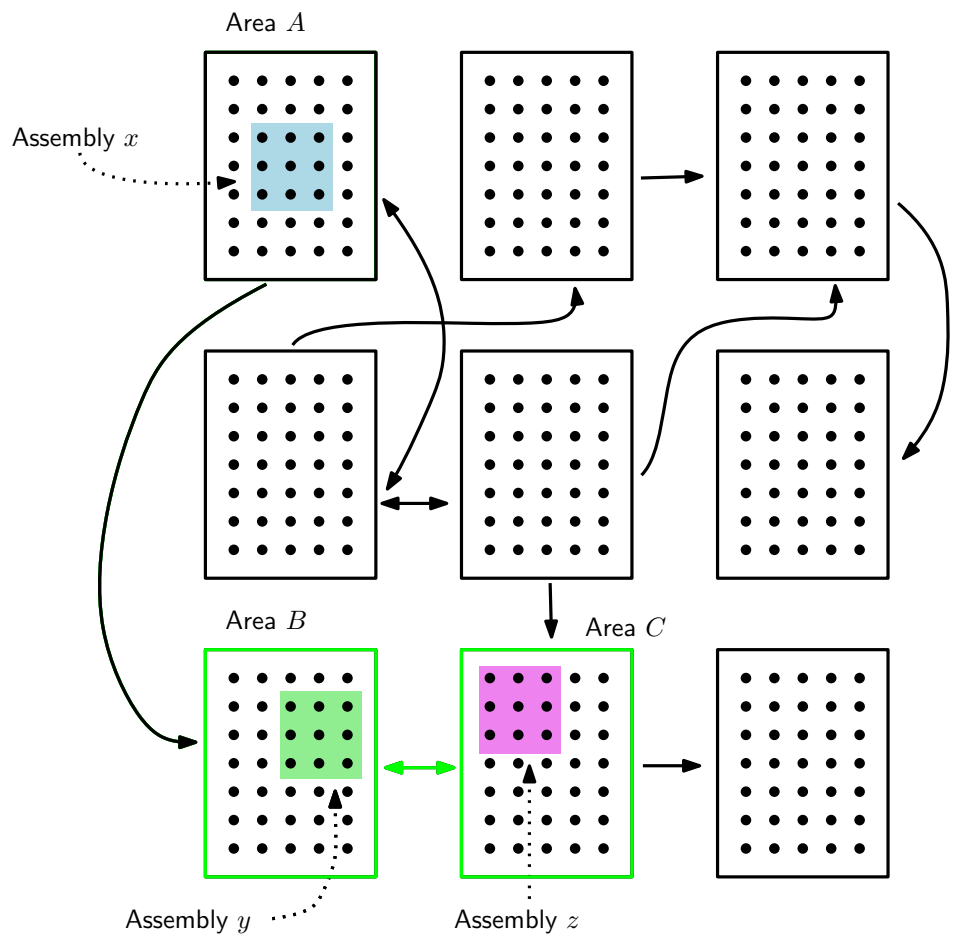
- **Projection**: the last active assembly x in area A starts firing repeatedly into area B until an assembly y is formed

This process eventually converges under a wide range of parameters w.h.p. [Papadimitriou et Vempala, ITCS 2019]

- **Association**: last active assemblies y in area B and z in area C fire simultaneously until they're linked

Linked: the firing of one results into the firing of the other

The brain and its areas



Remember?

Operations

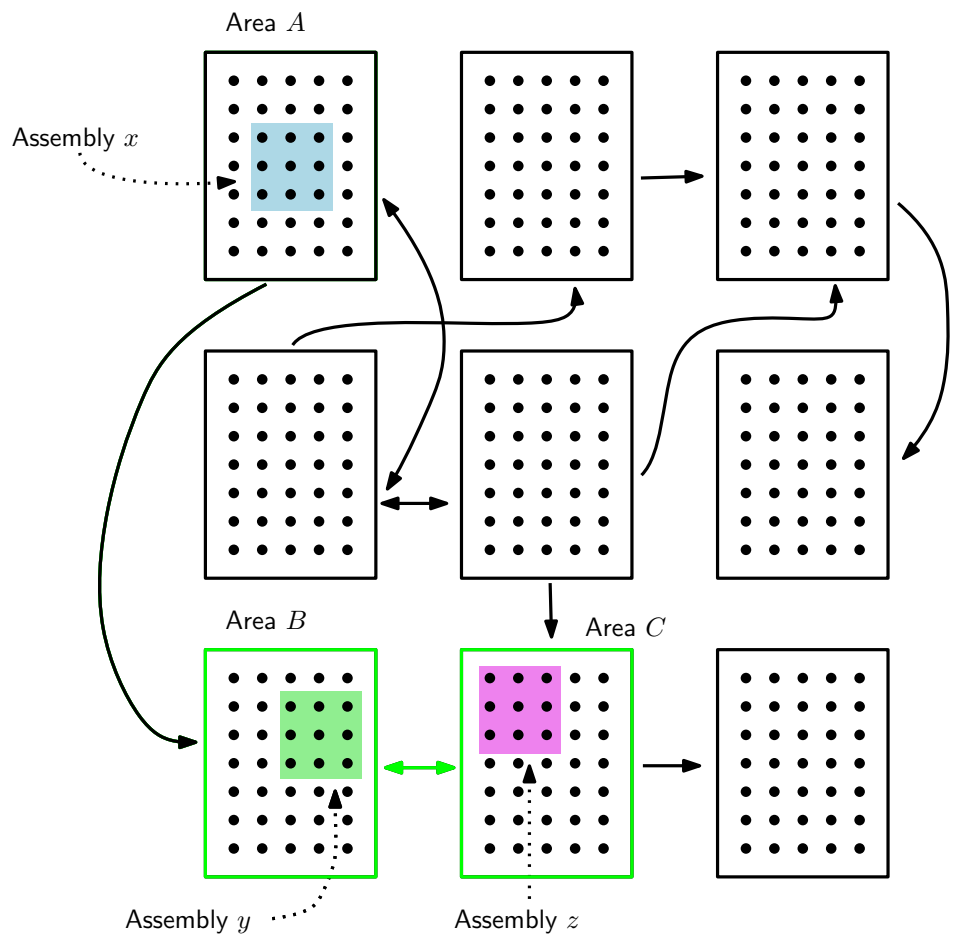
- **Projection**: the last active assembly x in area A starts firing repeatedly into area B until an assembly y is formed

This process eventually converges under a wide range of parameters w.h.p. [Papadimitriou et Vempala, ITCS 2019]

- **Association**: last active assemblies y in area B and z in area C fire simultaneously until they're linked

Linked: the firing of one results into the firing of the other

The brain and its areas



For areas A, B, C:

- `project` (A, B)
- `associate` (B, C)

Remember?

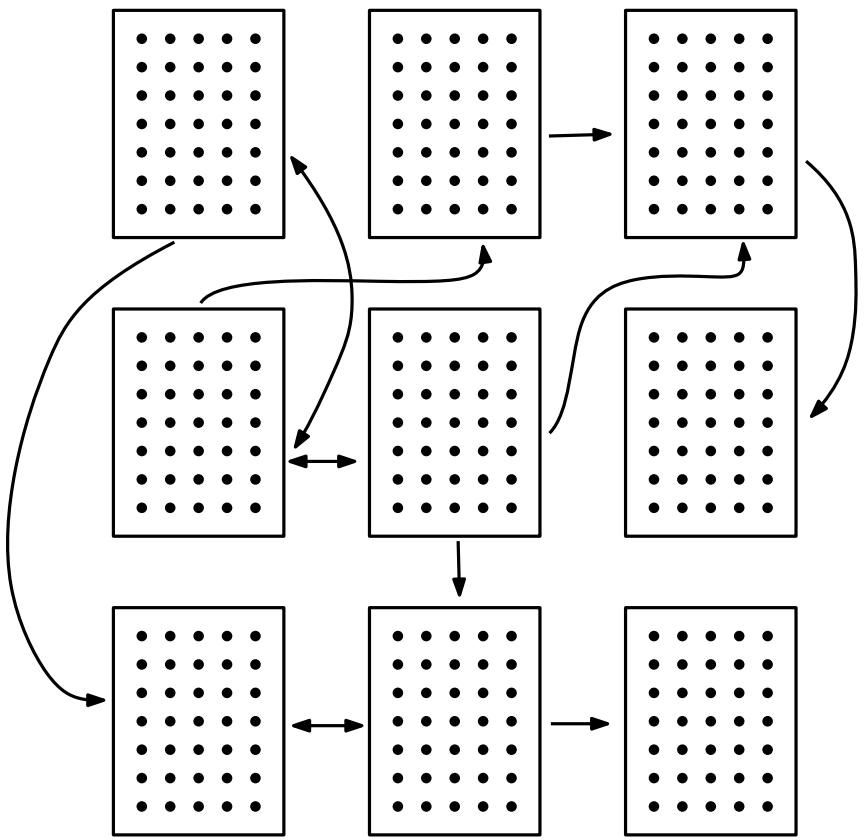
Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

The brain and its areas



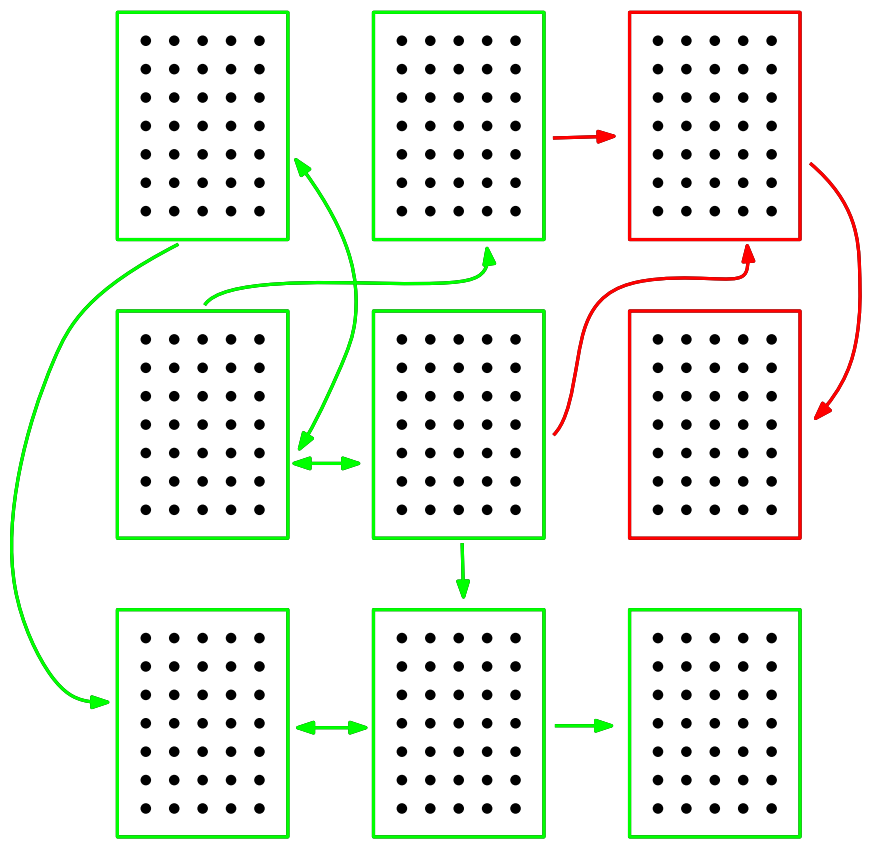
Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

The brain and its areas



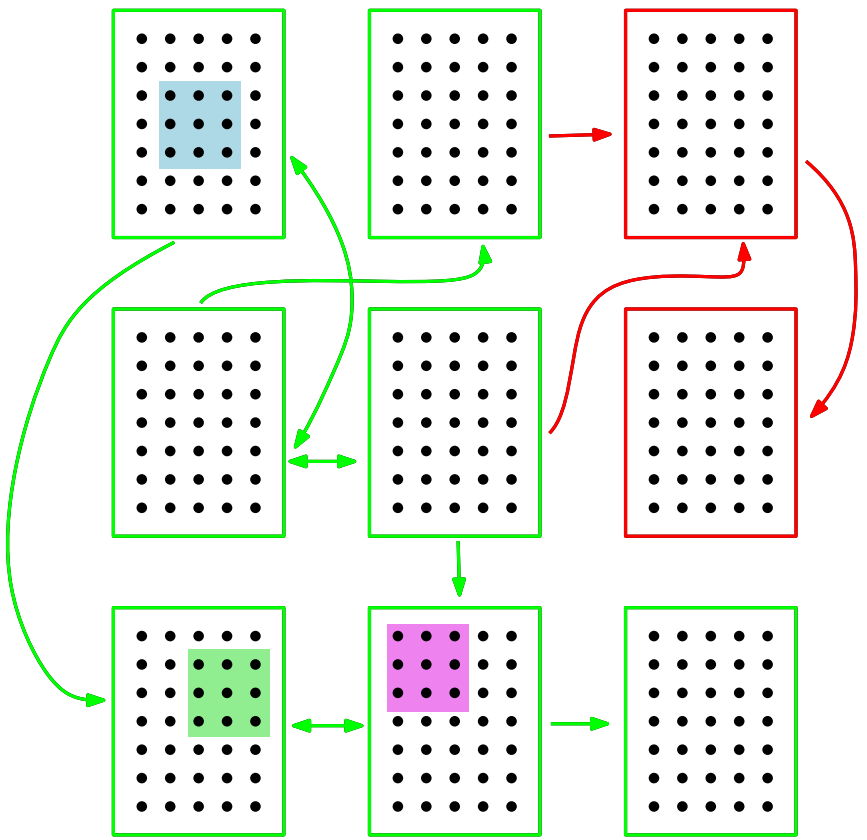
Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

The brain and its areas



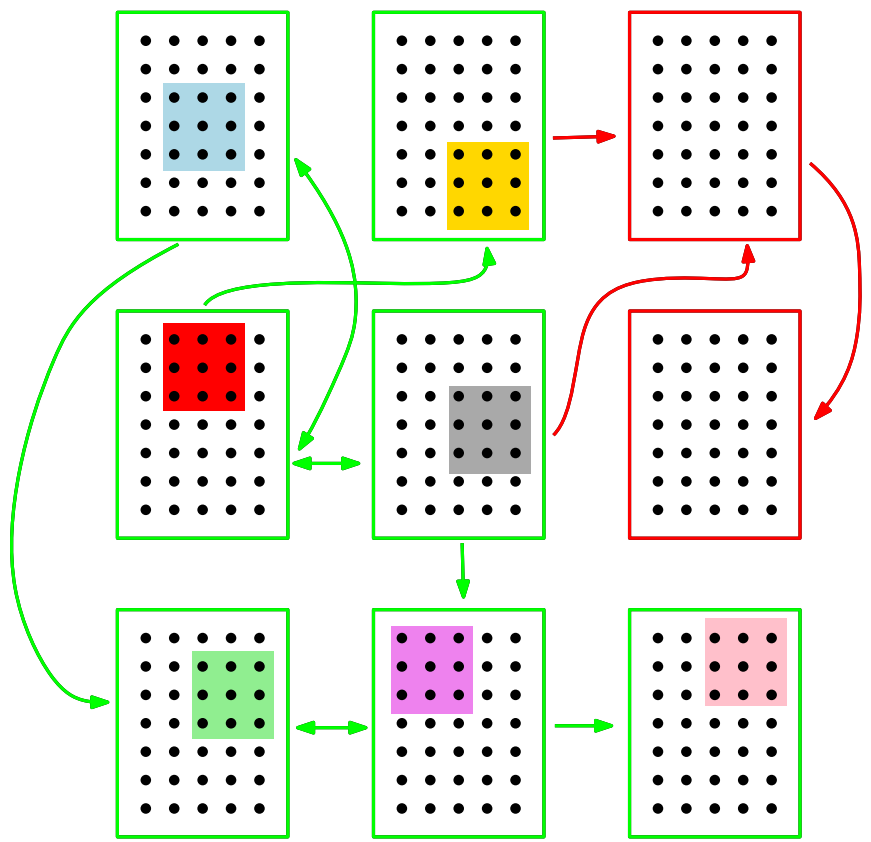
Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

The brain and its areas



Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

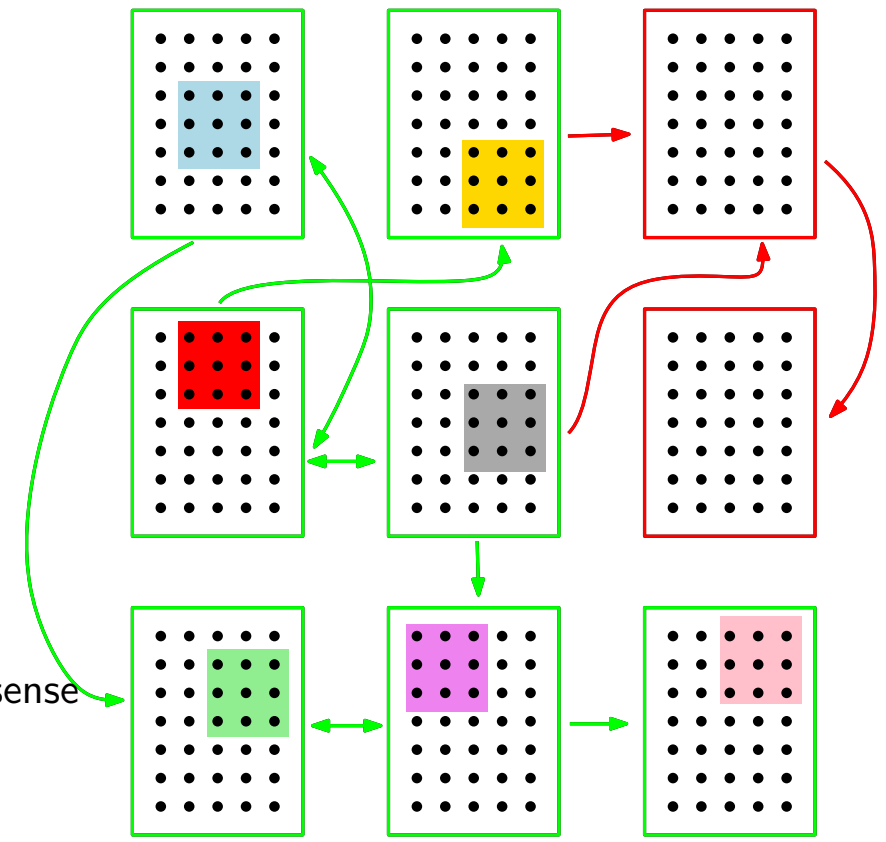
- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

- **Read**: identifies whether in a given area an assembly has fired

[Buzsáki, Neuron 2010] proposes that, for assemblies to be functionally useful, readout mechanisms must exist that sense the current state of the assembly system and trigger appropriate further action

- returns a boolean

The brain and its areas



Operations

- **Strong projection**: enhancement of the projection operation

Dynamics on the subgraph induced by all disinhibited areas and fibers:

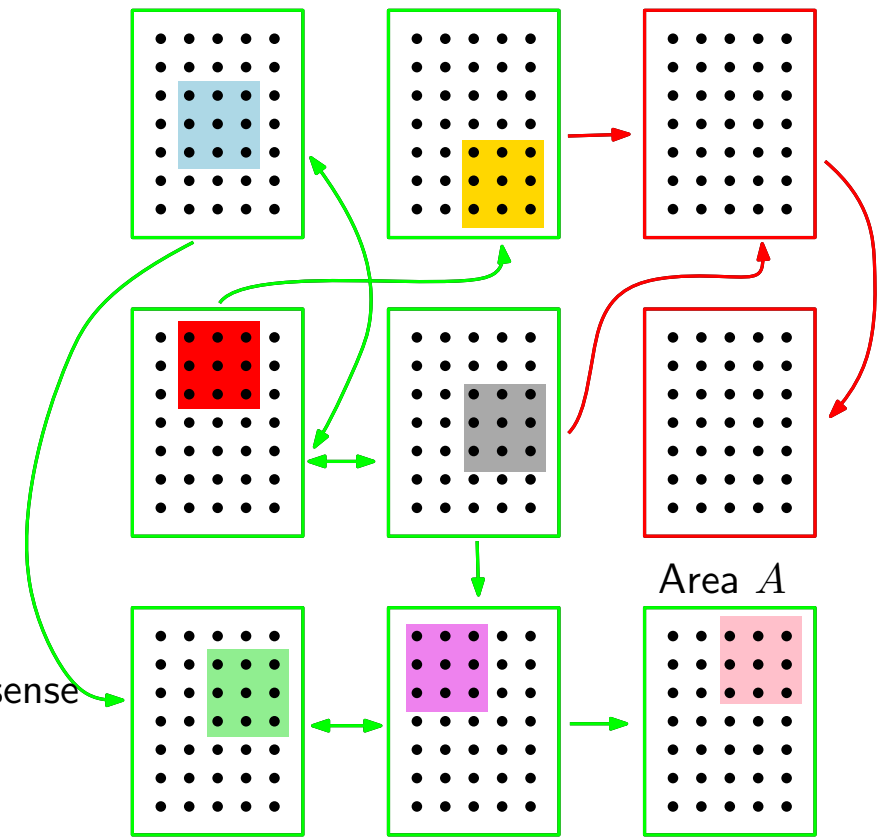
- all active assemblies start projecting in adjacent areas, forming new assemblies, and so on, until stability

- **Read**: identifies whether in a given area an assembly has fired

[Buzsáki, Neuron 2010] proposes that, for assemblies to be functionally useful, readout mechanisms must exist that sense the current state of the assembly system and trigger appropriate further action

- returns a boolean

The brain and its areas



We write `strongProject()` and `read`(A) for a brain area A

Related Works

- [Legenstein et al., ITCS 2018] shows analytically how assemblies emerge from stimula

- [Papadimitriou et Vempala, ITCS 2019] analyzes the convergence of processes defined by operations with assemblies

- [Papadimitriou et al., PNAS 2020] introduces the formal model of the assembly calculus and proves it's Turing complete
  - gives a Python code which simulates the model
  - leaves open:
    - language representation
    - planning strategies
    - reasoning

Related Works

- [Legenstein et al., ITCS 2018] shows analytically how assemblies emerge from stimula

- [Papadimitriou et Vempala, ITCS 2019] analyzes the convergence of processes defined by operations with assemblies

- [Papadimitriou et al., PNAS 2020] introduces the formal model of the assembly calculus and proves it's Turing complete
  - gives a Python code which simulates the model
  - leaves open:
    - ~~language representation~~
    - planning strategies
    - reasoning

- [Mitropolsky et al., TACL 2021] proposes a parser for the English language in the assembly calculus

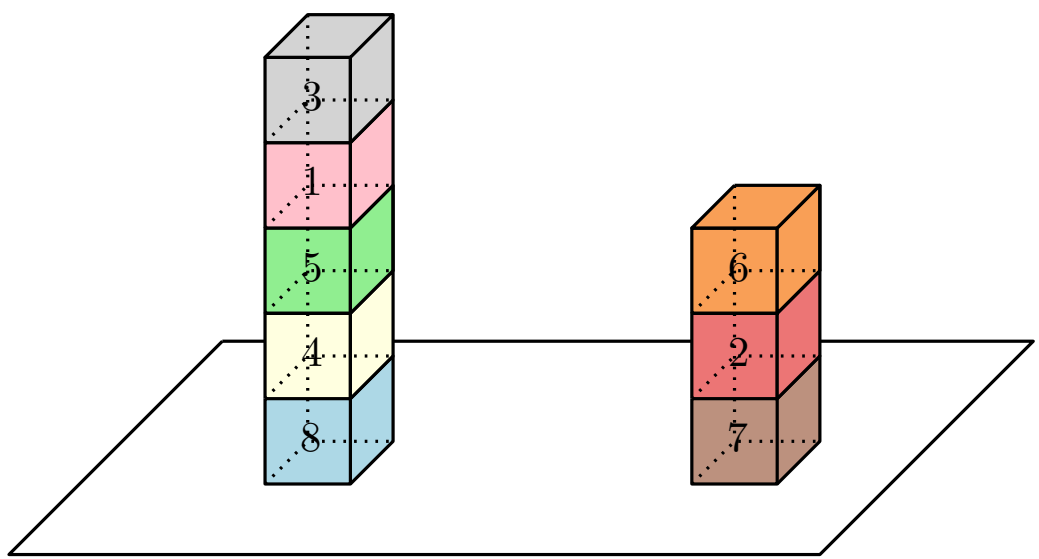
Our Contribution

We employ the assembly calculus to implement (through neurons and synapses) planning strategies in the blocks world (**Julia** programming language)

Our Contribution

We employ the assembly calculus to implement (through neurons and synapses) planning strategies in the blocks world (**Julia** programming language)

Blocks world: n unique blocks labelled in $\{1, \ldots, n\}$, organized as follows

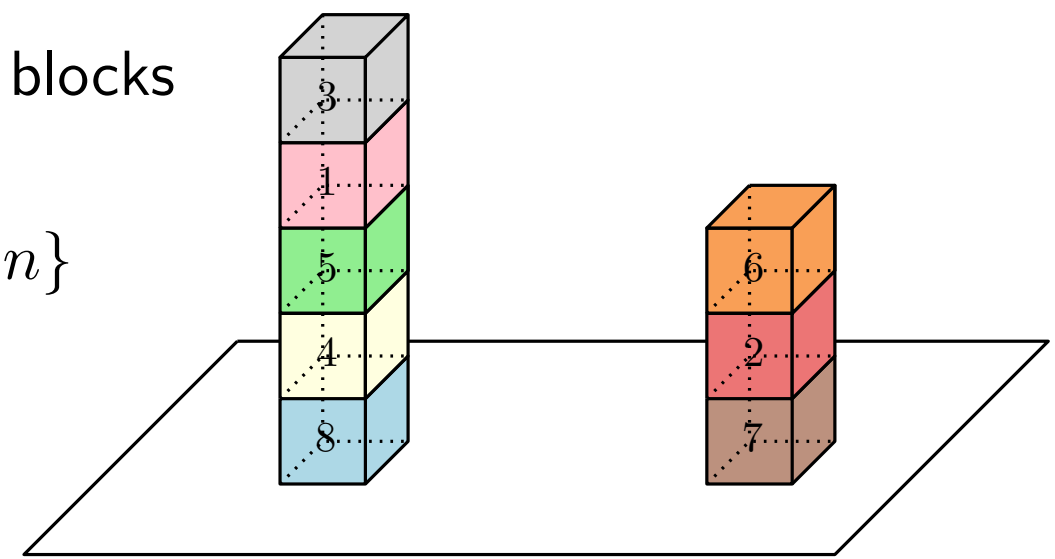


Our Contribution

We employ the assembly calculus to implement (through neurons and synapses) planning strategies in the blocks world (**Julia** programming language)

Blocks world: n unique blocks labelled in $\{1, \ldots, n\}$, organized as follows

- set of stacks $\{S_i\}_i$
- each stack is a sequence of blocks $S_i = (b_1^{(i)}, \ldots, b_{k_i}^{(i)})$
- $\cup_i \{b_1^{(i)}, \ldots, b_{k_i}^{(i)}\} = \{1, \ldots, n\}$



The Planning Task

Possible moves:

- move a block from the top of a stack to the table
- move a block from the top of a stack to the top of another stack
- move a block from the table to the top of a stack

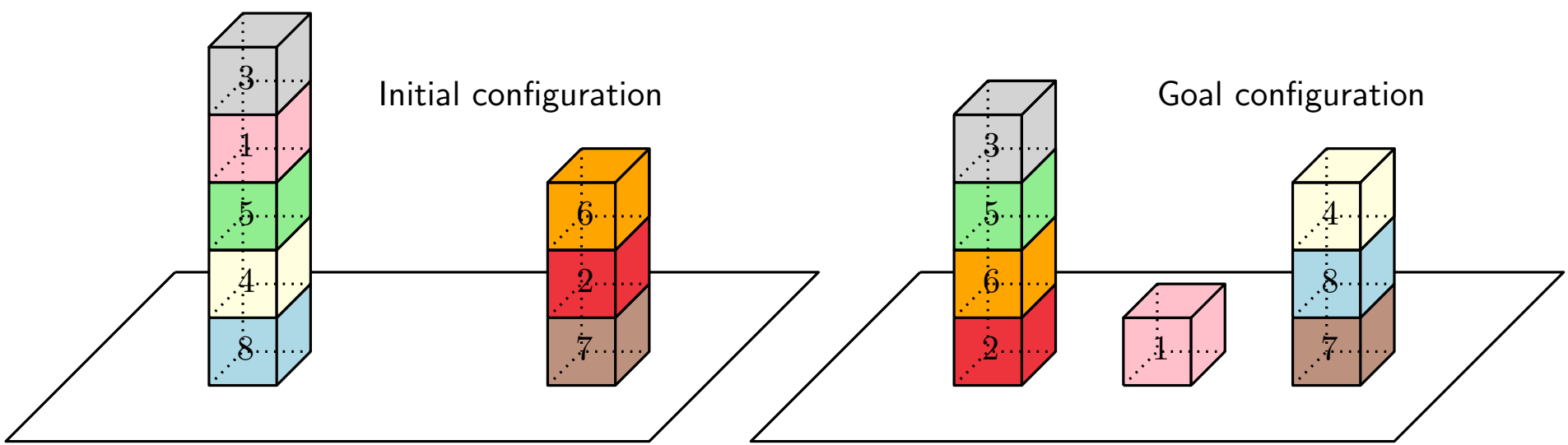
The Planning Task

Possible moves:

- move a block from the top of a stack to the table
- move a block from the top of a stack to the top of another stack
- move a block from the table to the top of a stack

Input:

- set of initial stacks $\{S_i^{(in)}\}_i$, $S_i^{(in)} = (b_1^{(i)}, \ldots, b_{k_i}^{(i)})$
- $\cup_i \{b_1^{(i)}, \ldots, b_{k_i}^{(i)}\} = \{1, \ldots, n\}$
- set of goal stacks $\{S_i^{(goal)}\}_i$, $S_i^{(goal)} = (c_1^{(i)}, \ldots, c_{k_i}^{(i)})$
- $\cup_i \{c_1^{(i)}, \ldots, c_{k_i}^{(i)}\} = \{1, \ldots, n\}$



The Planning Task

Possible moves:

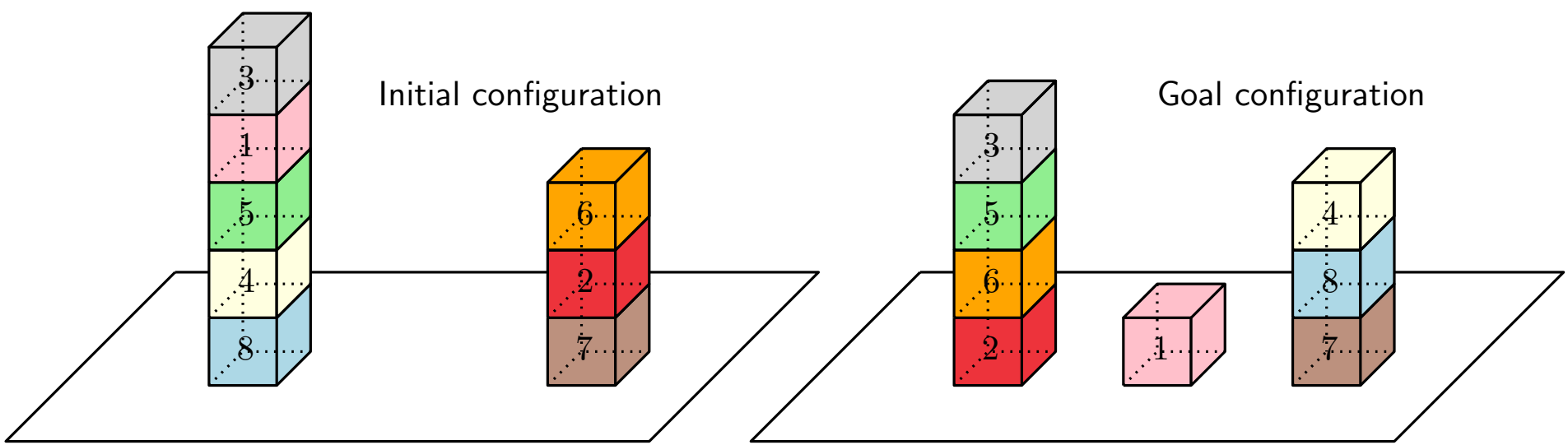
- move a block from the top of a stack to the table
- move a block from the top of a stack to the top of another stack
- move a block from the table to the top of a stack

Input:

- set of initial stacks $\{S_i^{(in)}\}_i$, $S_i^{(in)} = (b_1^{(i)}, \ldots, b_{k_i}^{(i)})$
- $\cup_i \{b_1^{(i)}, \ldots, b_{k_i}^{(i)}\} = \{1, \ldots, n\}$
- set of goal stacks $\{S_i^{(goal)}\}_i$, $S_i^{(goal)} = (c_1^{(i)}, \ldots, c_{k_i}^{(i)})$
- $\cup_i \{c_1^{(i)}, \ldots, c_{k_i}^{(i)}\} = \{1, \ldots, n\}$

Output:

- the sequence of moves



Strategies

Brute-force strategy: *move all blocks to the table and place them correctly, one by one*

Strategies

Brute-force strategy: *move all blocks to the table and place them correctly, one by one*

[Gupta et Nau, AI 1992] proves the problem of finding the optimum is NP-complete, and provides a 2-apx algorithm

> *Move each block which is not in its final position to the table, and then place the blocks (which are on the table) correctly one by one*

Strategies

Brute-force strategy: *move all blocks to the table and place them correctly, one by one*

[Gupta et Nau, AI 1992] proves the problem of finding the optimum is NP-complete, and provides a 2-apx algorithm

> *Move each block which is not in its final position to the table, and then place the blocks (which are on the table) correctly one by one*

We implement both of them...

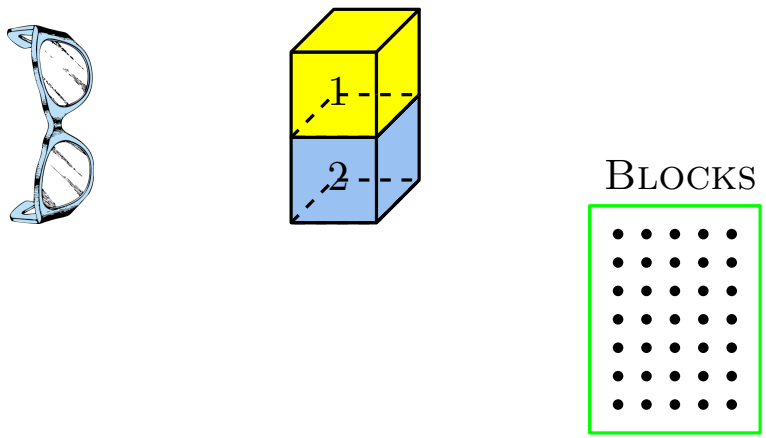
Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

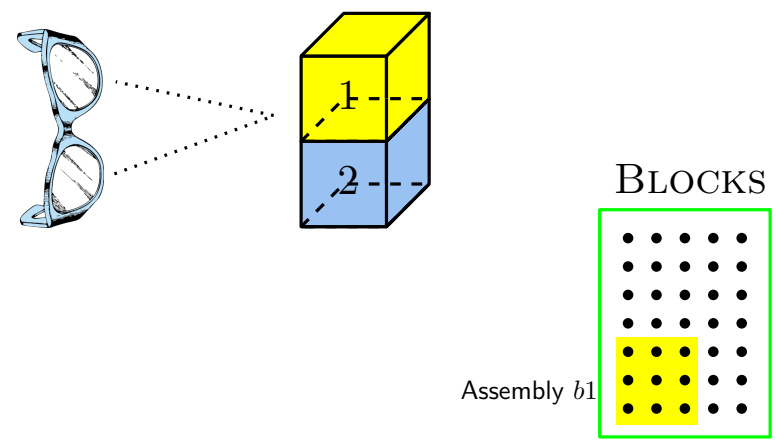
The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)



Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

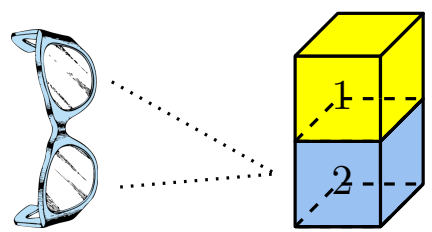
The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)

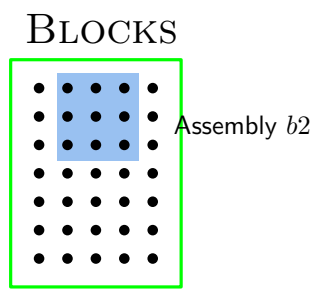


Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)



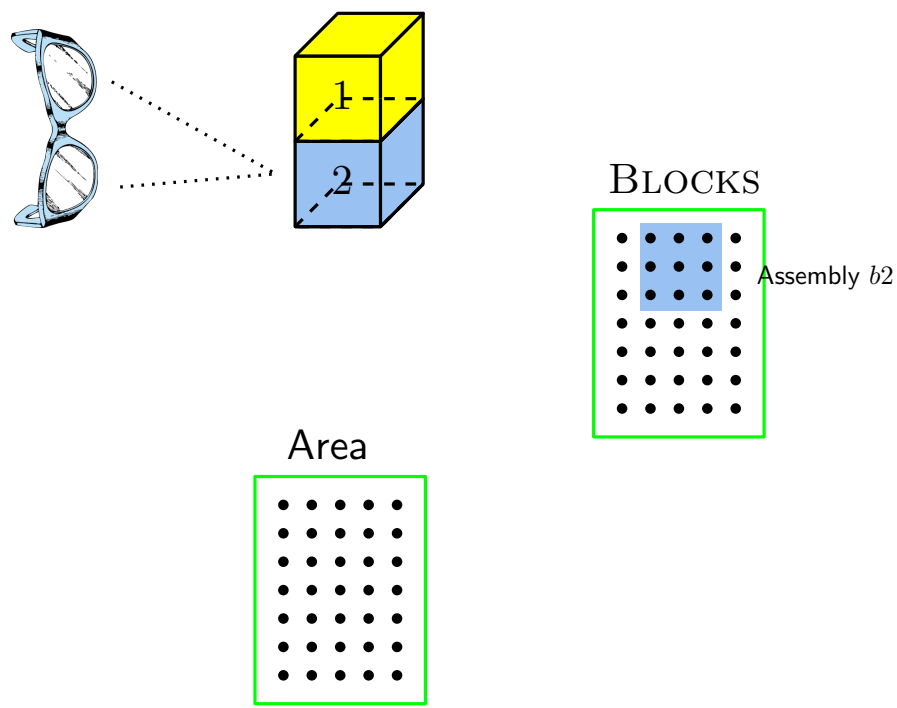


Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)

Before and after each activation, the brain makes some operations, such as projection, inhibition, disinhibition

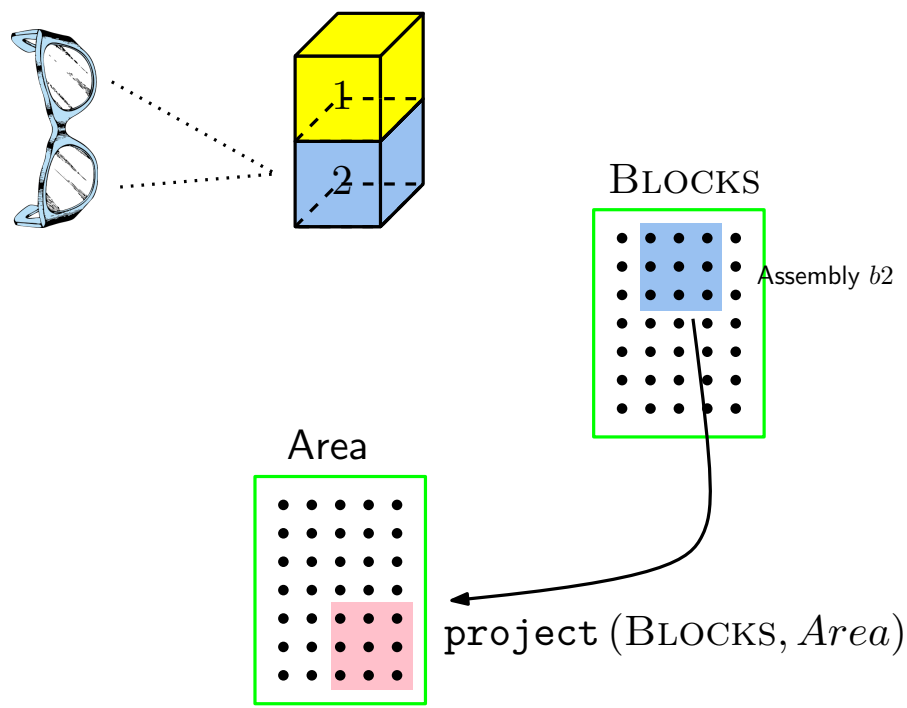


Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)

Before and after each activation, the brain makes some operations, such as projection, inhibition, disinhibition



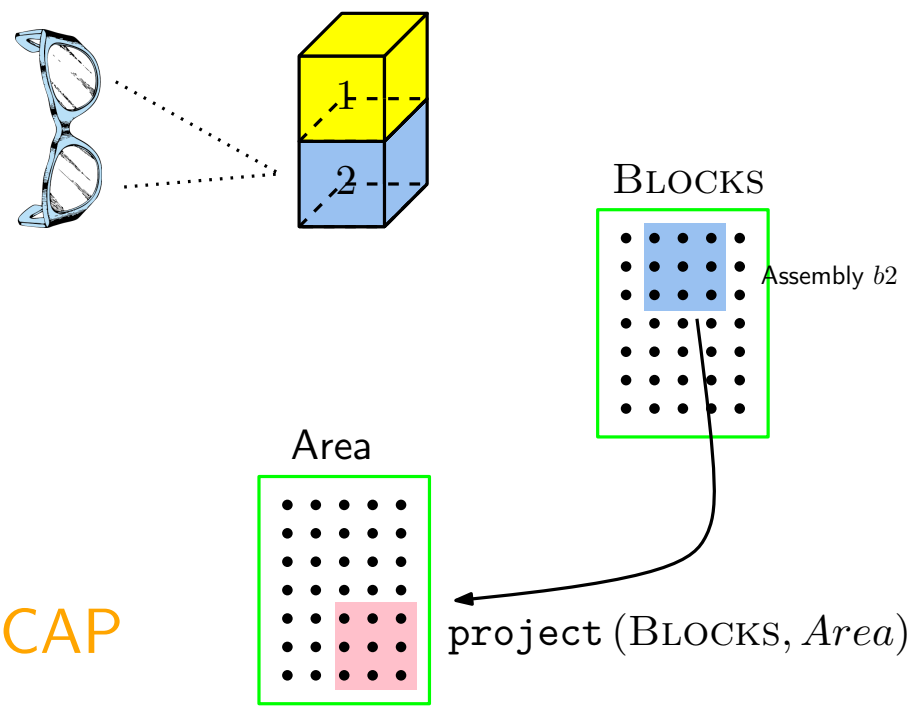
Back to the Brain: Assumptions

The individual "scans" the block world configuration by processing each stack from the top block to the bottom block, one at a time

The scanning of each block activates an unique corresponding assembly in the brain area BLOCKS (if disinhibited)

Before and after each activation, the brain makes some operations, such as projection, inhibition, disinhibition

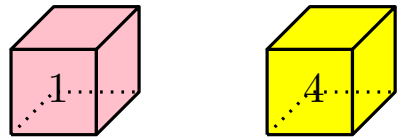
Each area has n neurons, the same Erdös-Renyi prob. p, the same k as the CAP

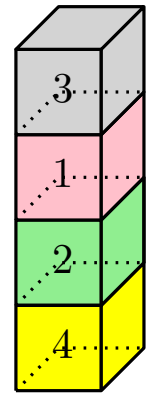


A Toy Example

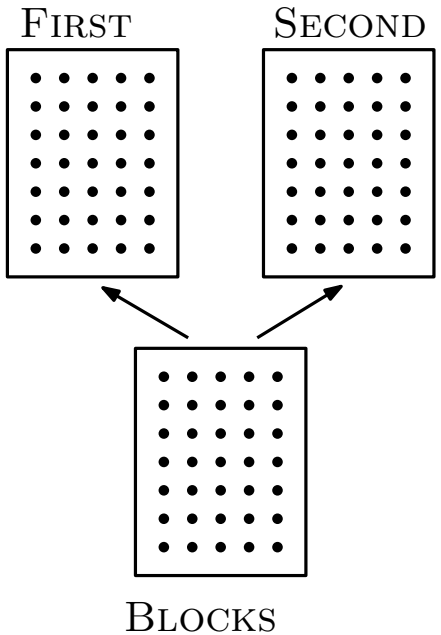
Given a stack of blocks and two of its blocks a, b*, is* a **above** b*?*

E.g., is block 1 above block 4?





Assumptions: 3 brain areas, Blocks, First, Second



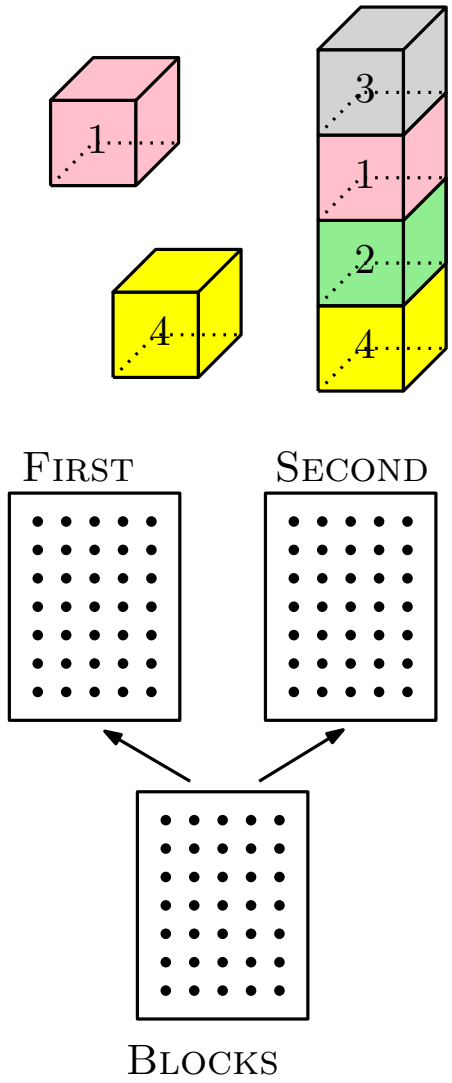
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



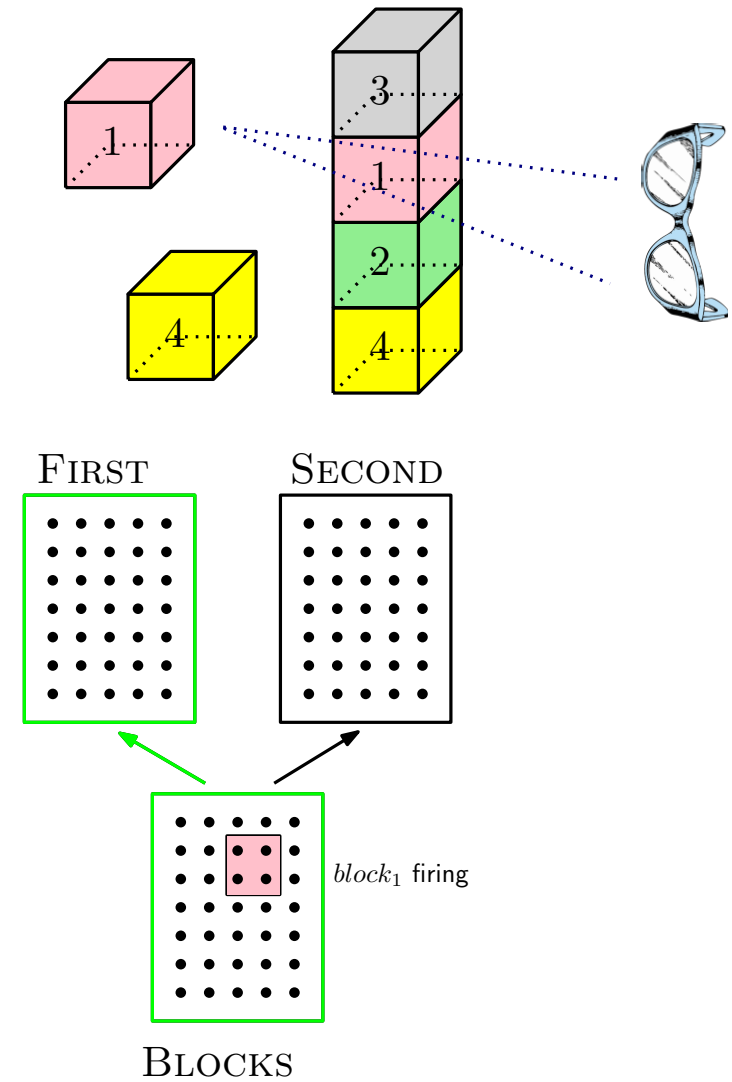
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



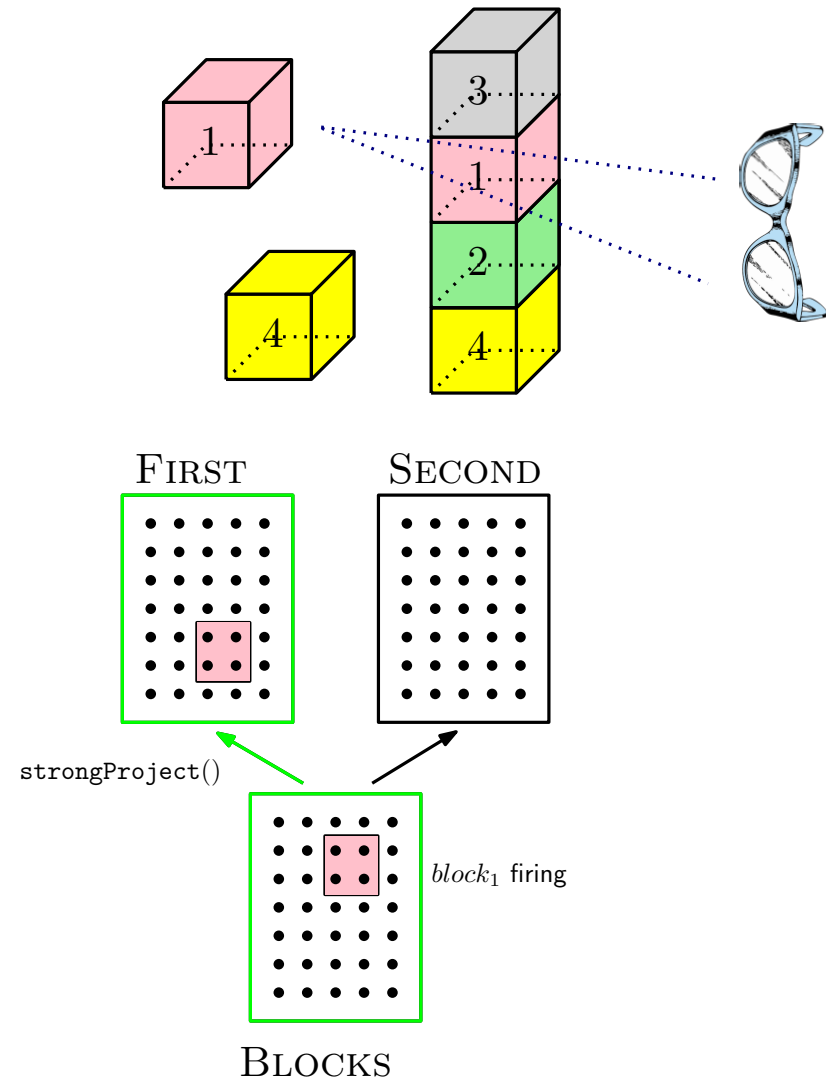
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



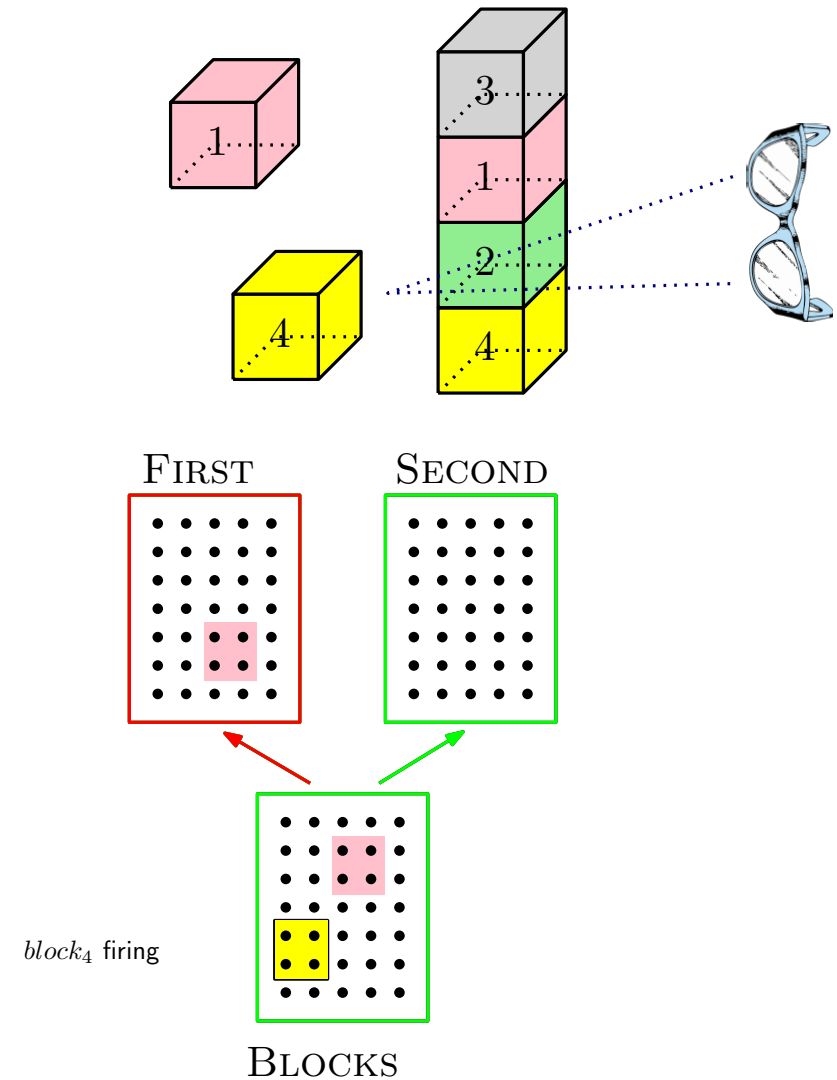
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



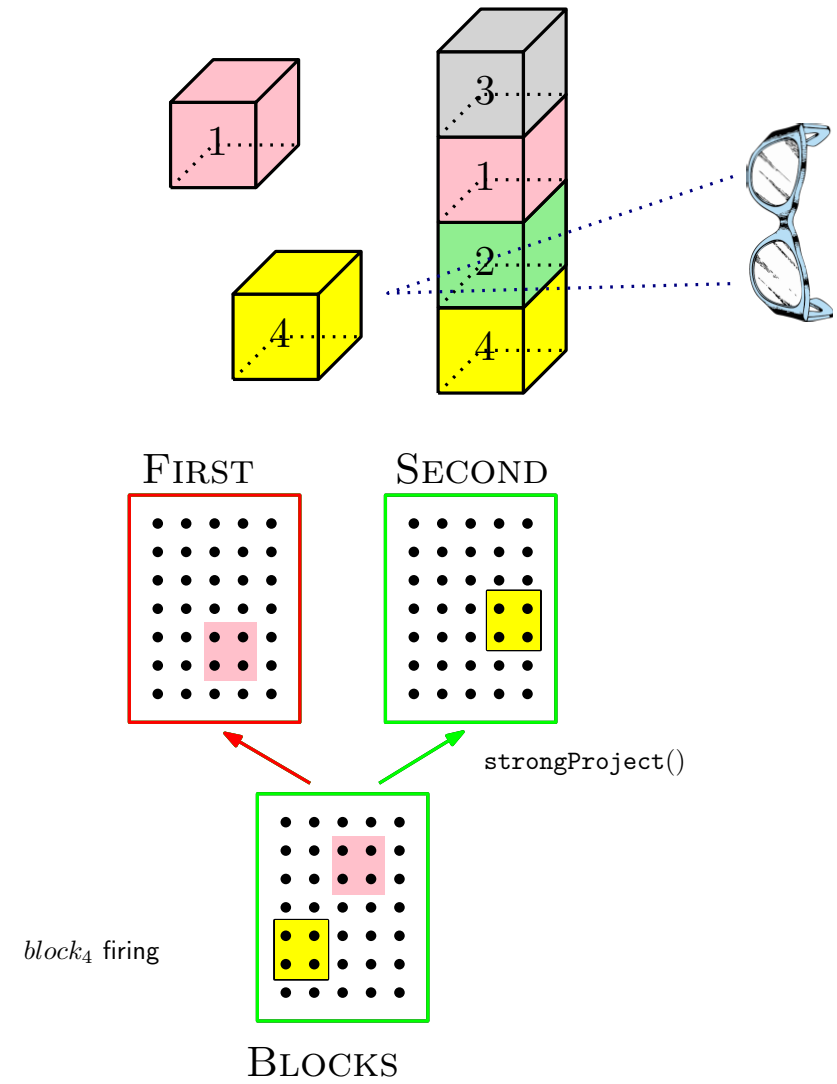
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



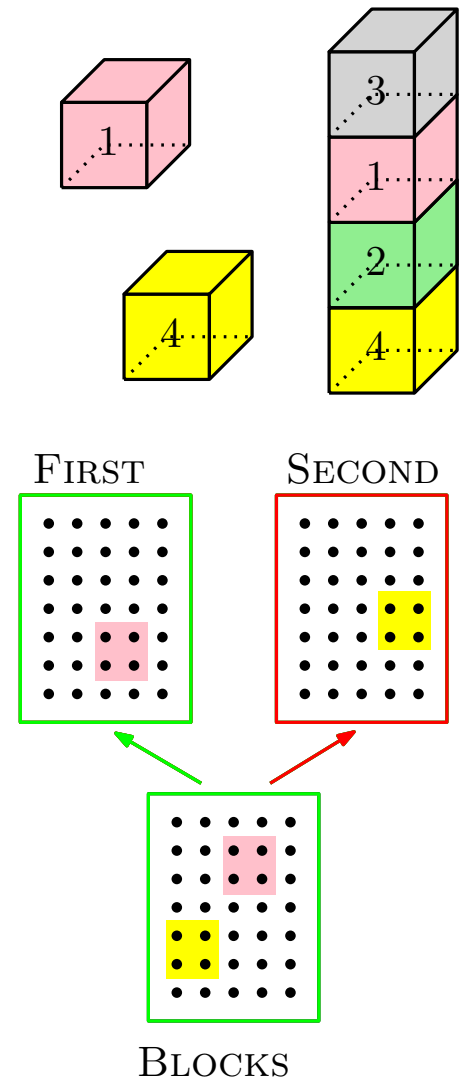
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



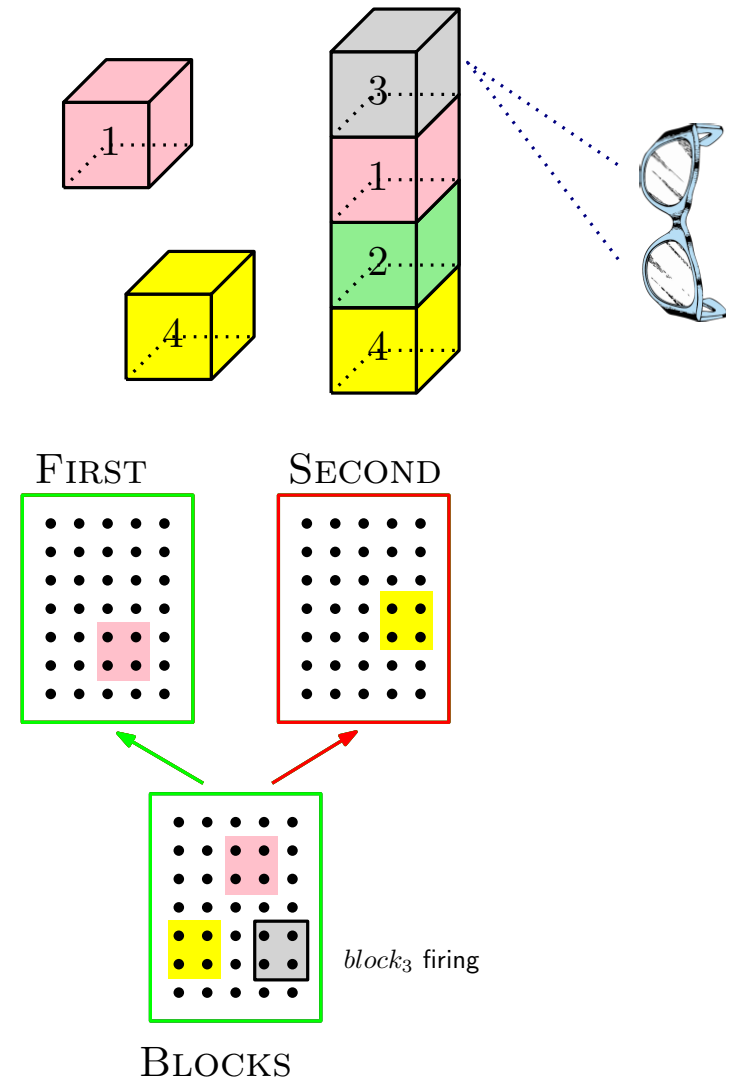
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



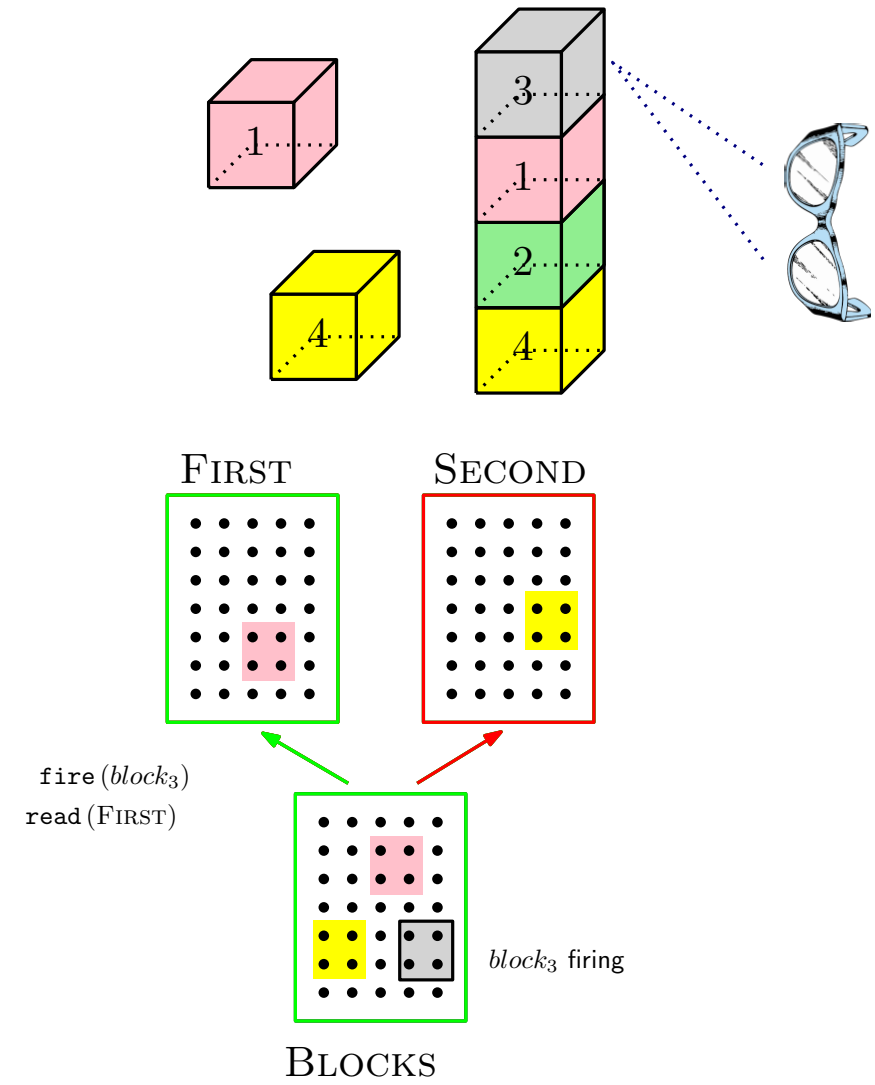
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



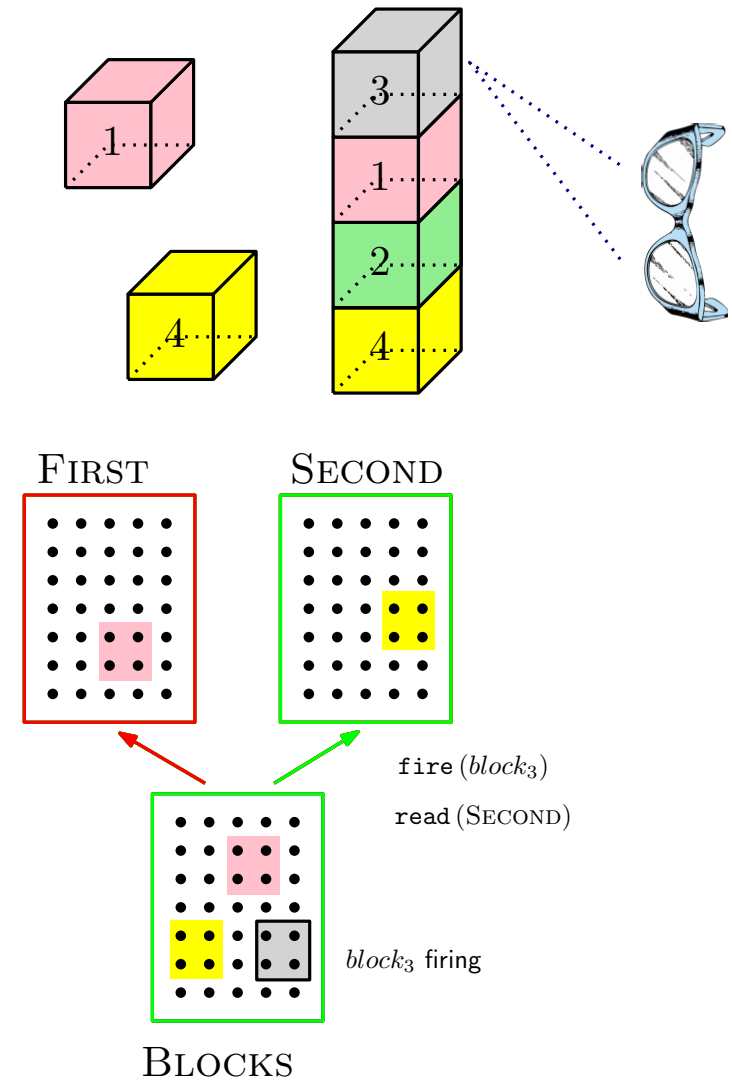
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



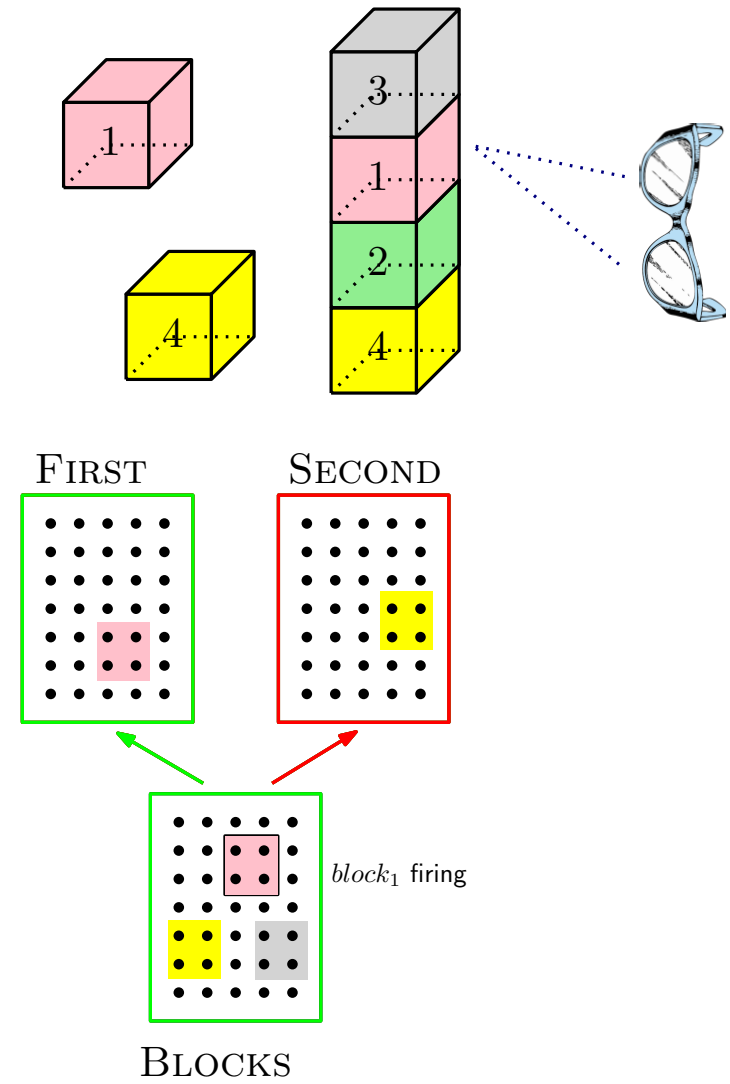
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



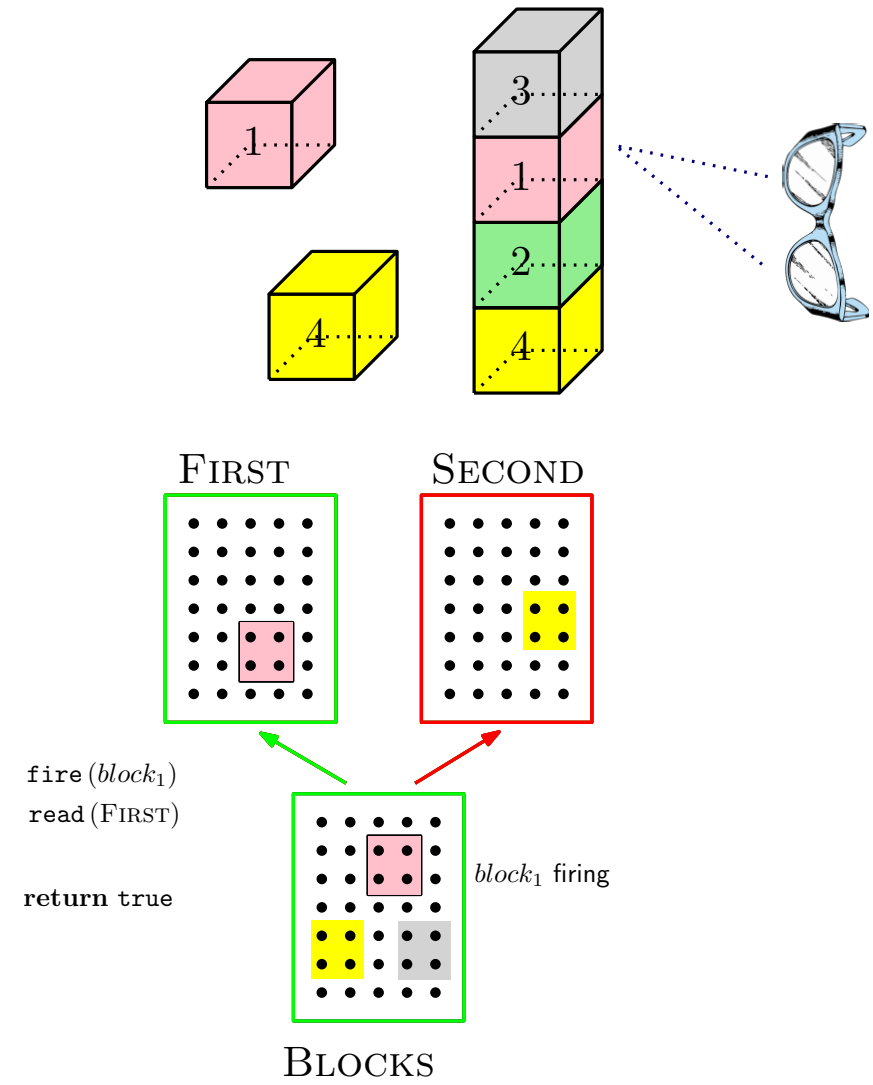
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



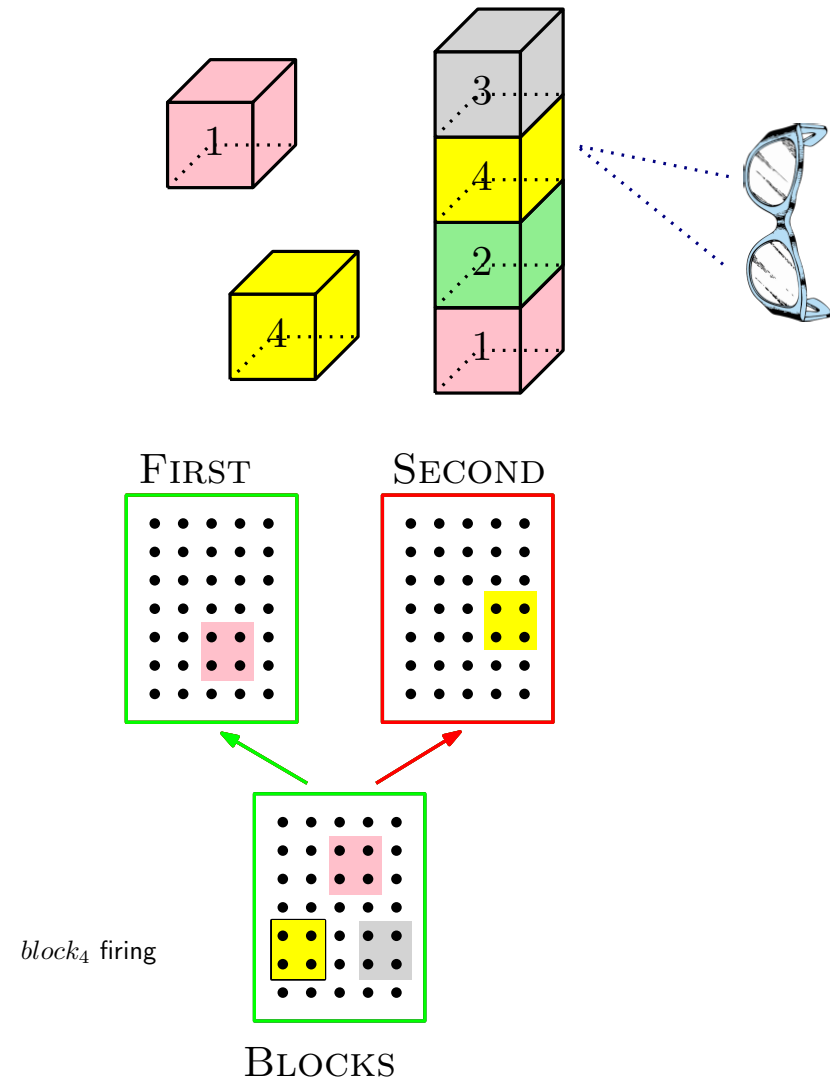
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



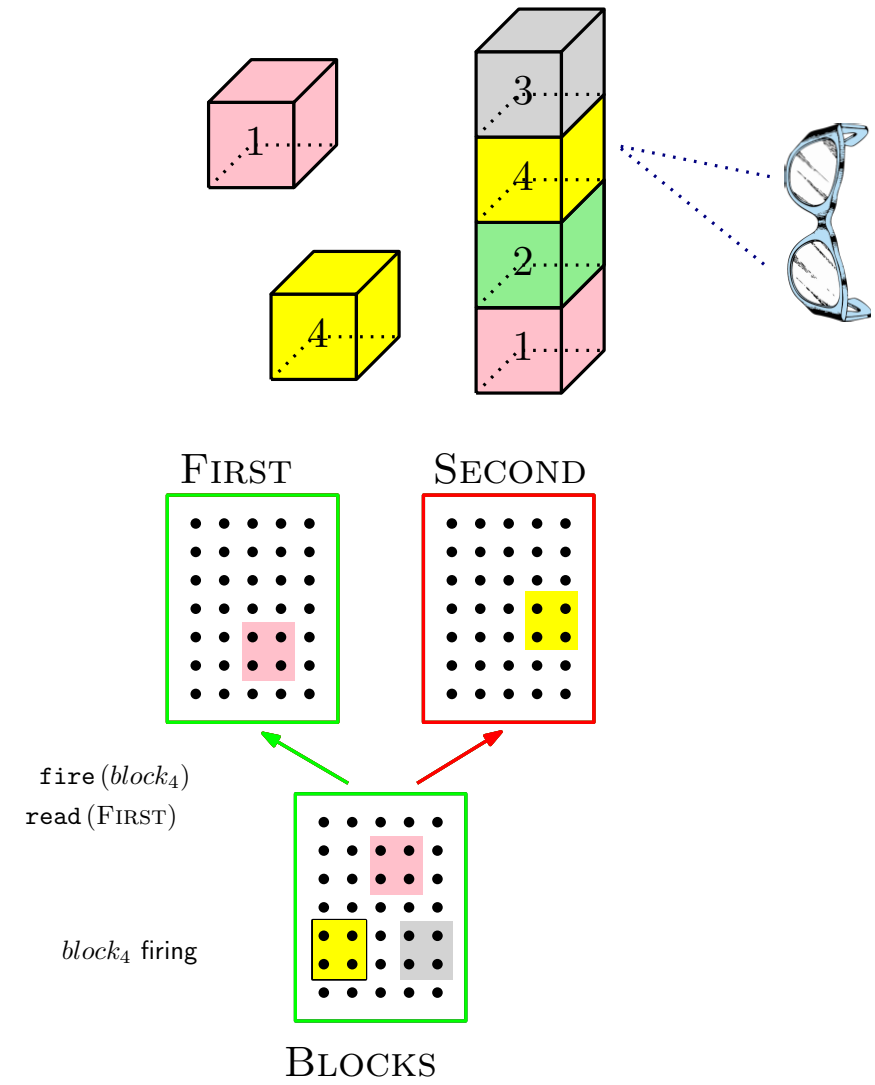
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



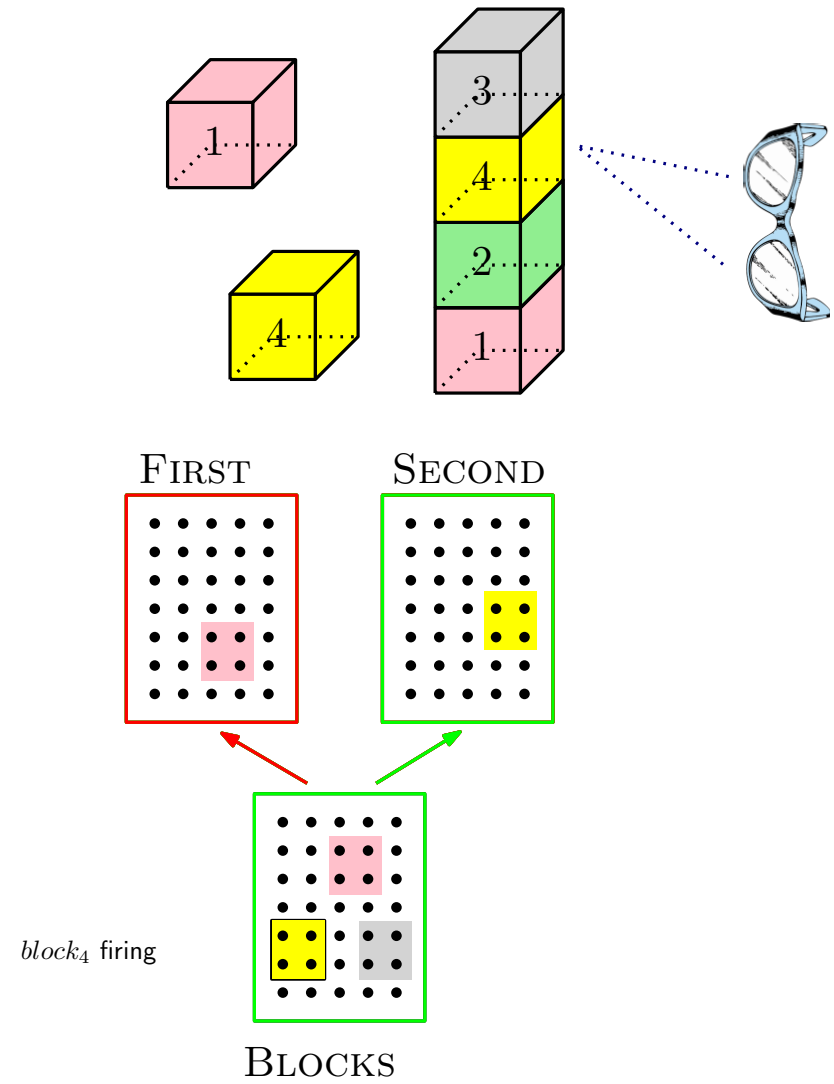
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11     inhibitArea({Second});
12     disinhibitArea({First});
13     fire(b_i);
14     if read(First) then return true ;
15     inhibitArea({First});
16     disinhibitArea({Second});
17     fire(b_i);
18     if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



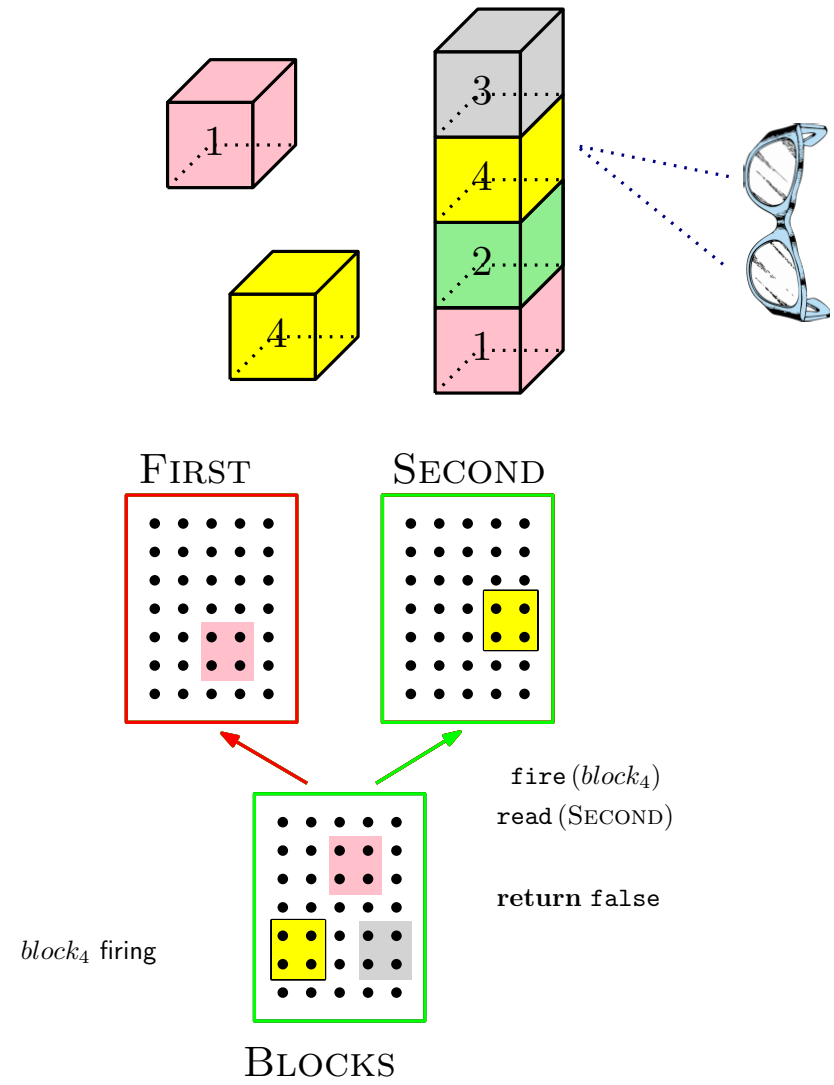
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

E.g., is block 1 above block 4?



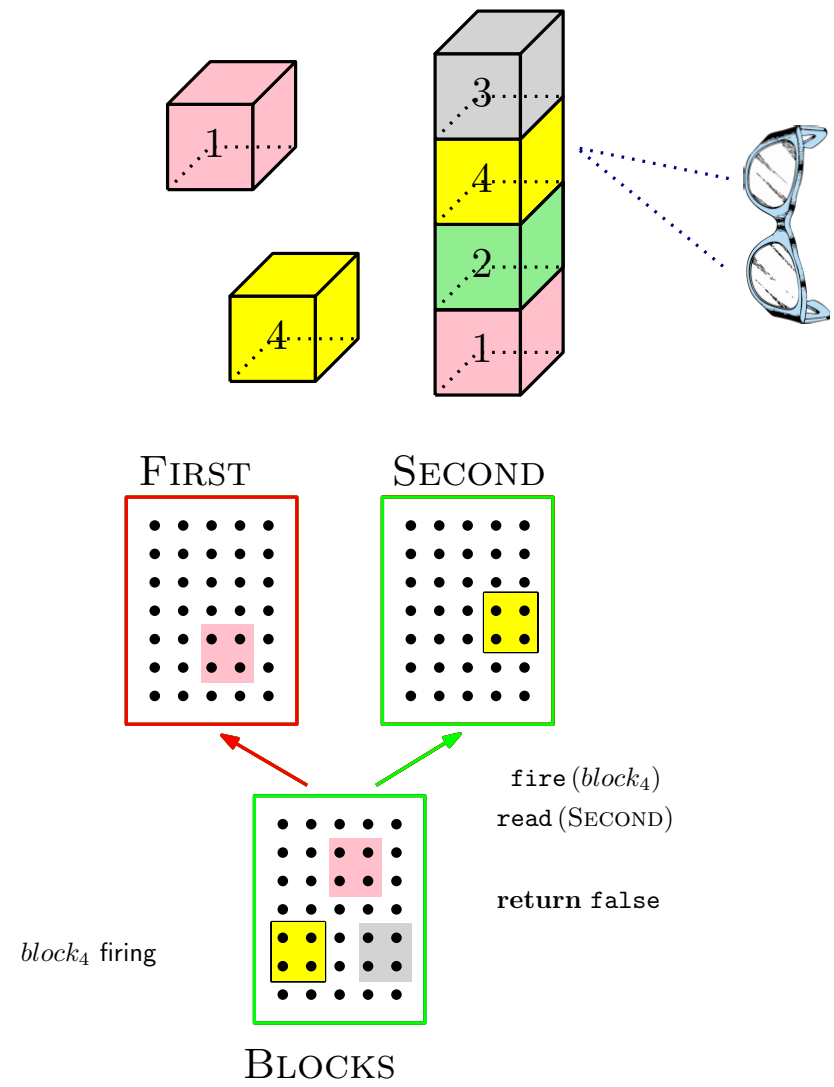
IsAbove Algorithm

Algorithm 1: IsAbove (S, x, y)

input: a stack S of blocks $b_1, b_2, \ldots, b_s$;
two blocks x, y with $x, y \in S$.

```
1  disinhibitArea({Blocks, First});
2  disinhibitFiber({(Blocks, First)});
3  fire(x);
4  strongProject();
5  inhibitArea({First});
6  disinhibitArea({Second});
7  disinhibitFiber({(Blocks, Second)});
8  fire(y);
9  strongProject();
10 foreach i with 2 ≤ i ≤ s do
11 |   inhibitArea({Second});
12 |   disinhibitArea({First});
13 |   fire(b_i);
14 |   if read(First) then return true ;
15 |   inhibitArea({First});
16 |   disinhibitArea({Second});
17 |   fire(b_i);
18 |   if read(Second) then return false ;
19 end
```

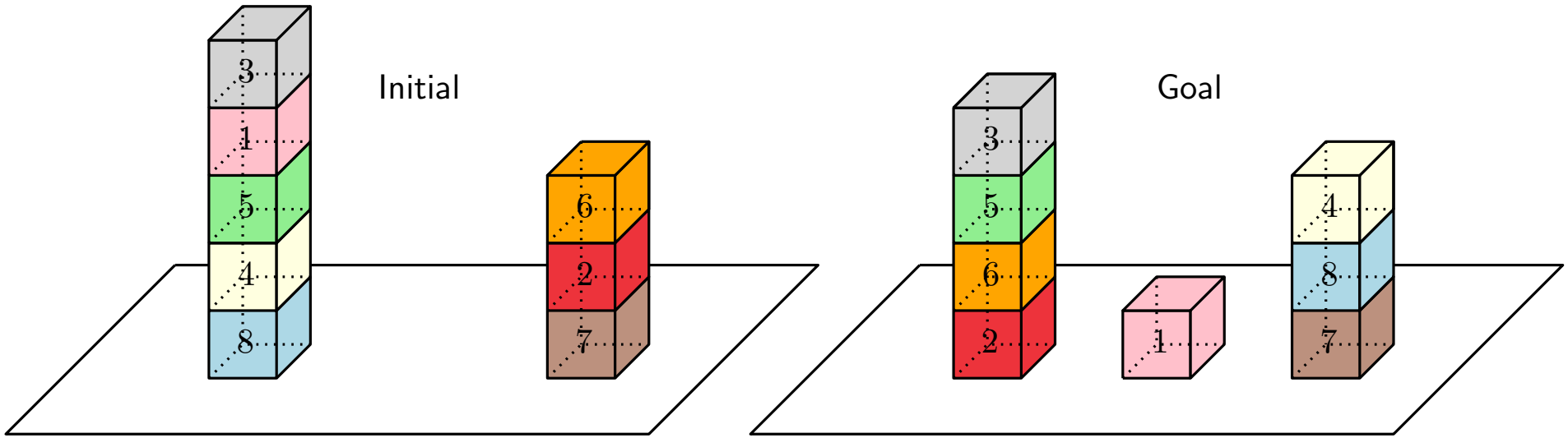
E.g., is block 1 above block 4?



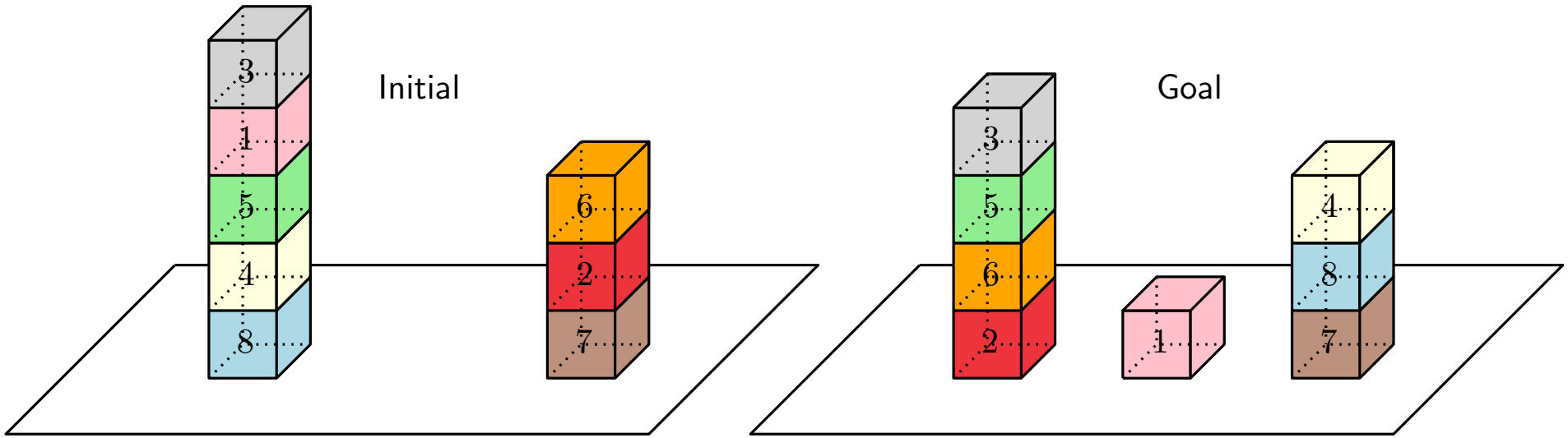
How does the brain understand a positive (negative) answer?

The answer corresponds to an assembly activating in a particular brain area

Back to the Planning Problem

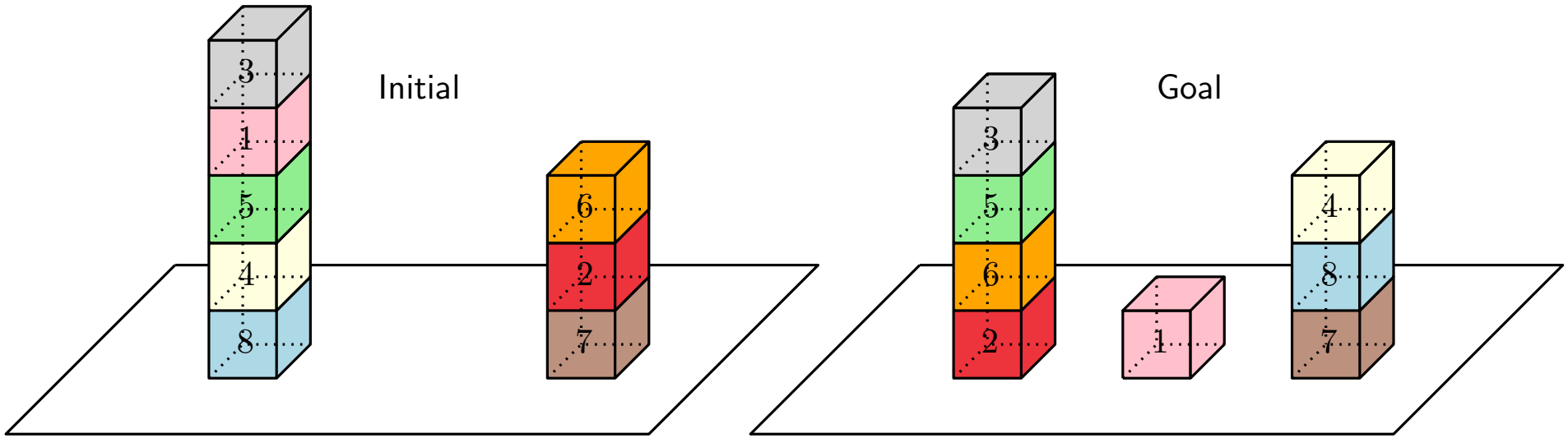


Back to the Planning Problem



- **First**: way to represent the initial and the goal stacks in the brain

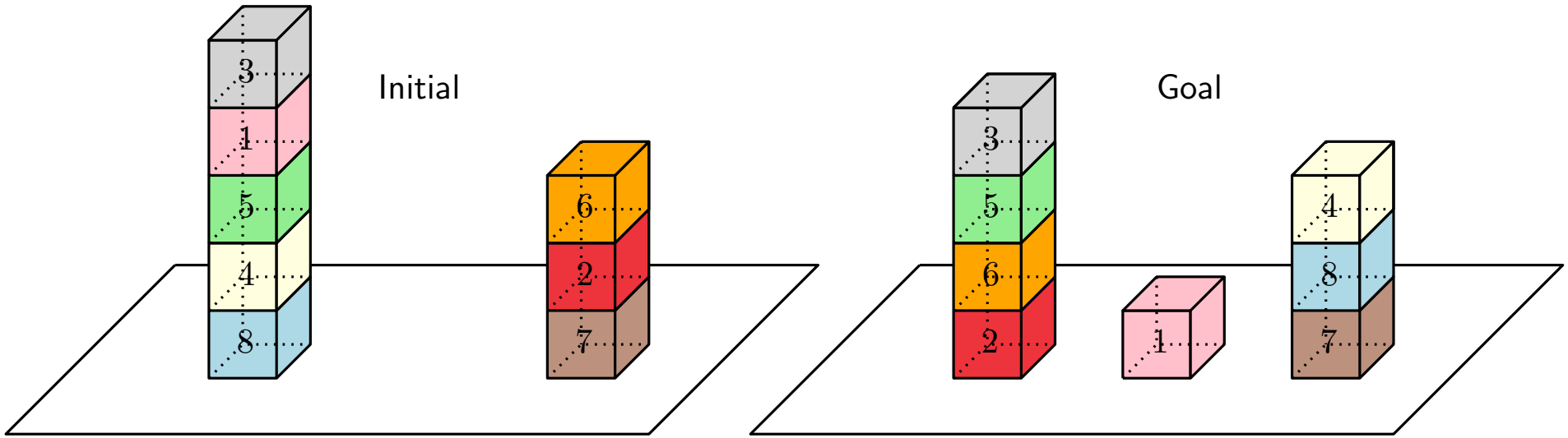
Back to the Planning Problem



- **First**: way to represent the initial and the goal stacks in the brain

A Parser

Back to the Planning Problem



- **First**: way to represent the initial and the goal stacks in the brain

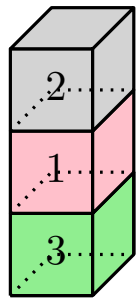
A Parser

- **Second**: way to implement actions and represent them

The Parser

Assumptions:

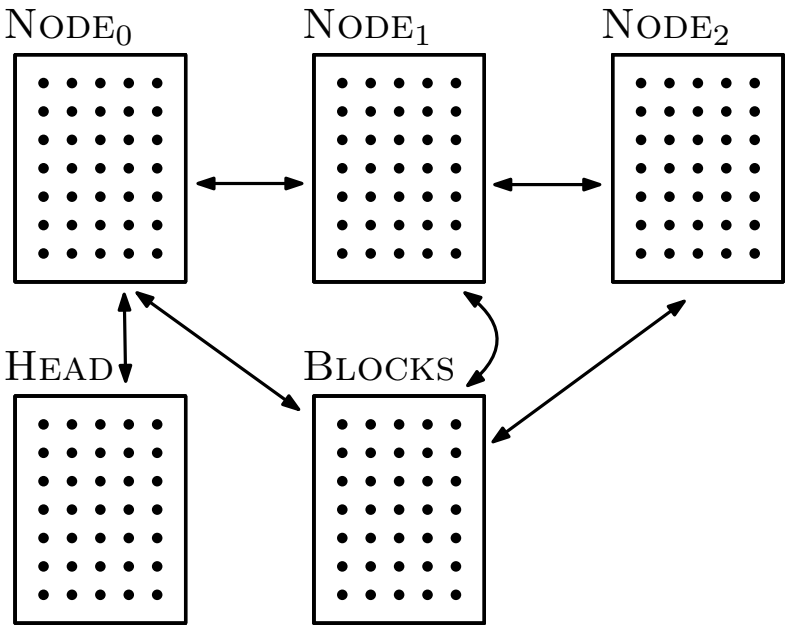
- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE_0, NODE_1, NODE_2, HEAD

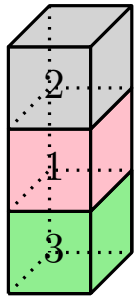


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

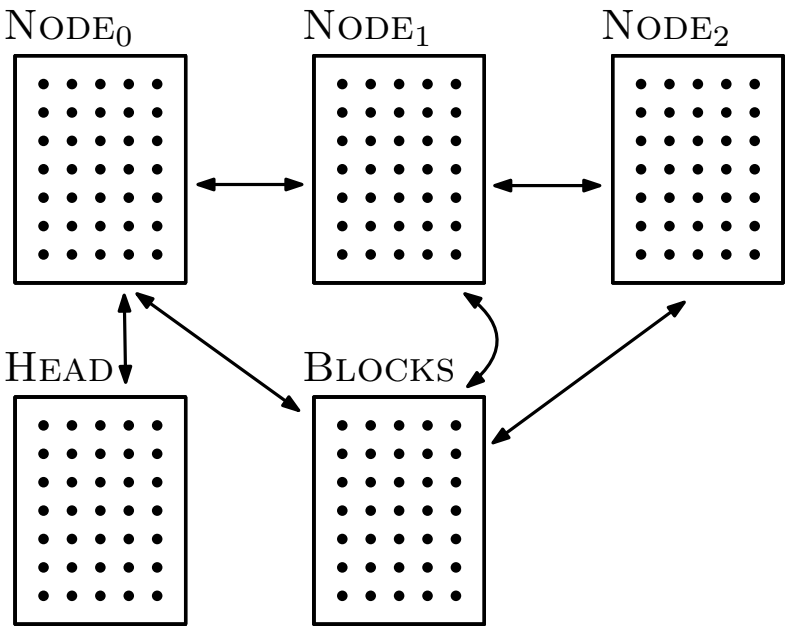


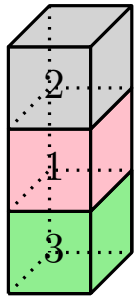


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

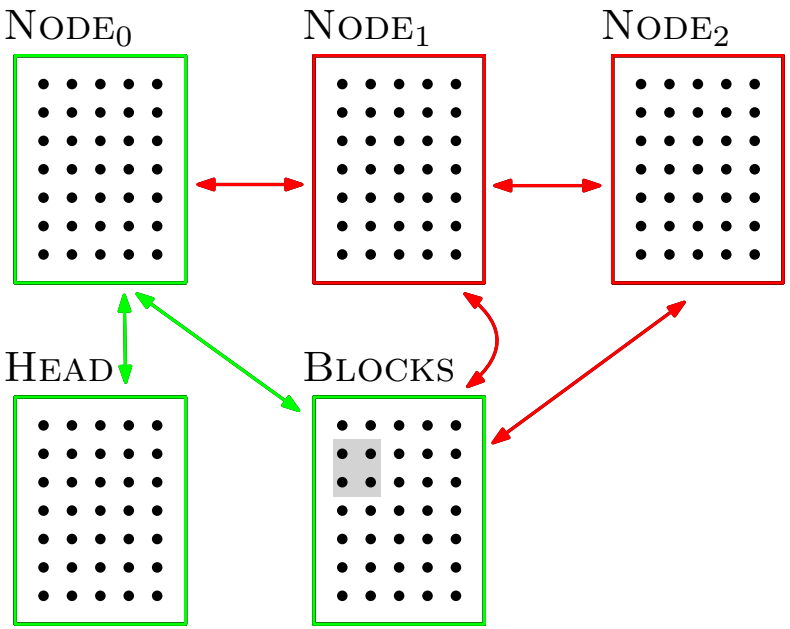


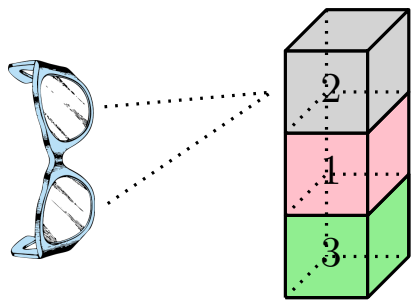


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE_0, NODE_1, NODE_2, HEAD

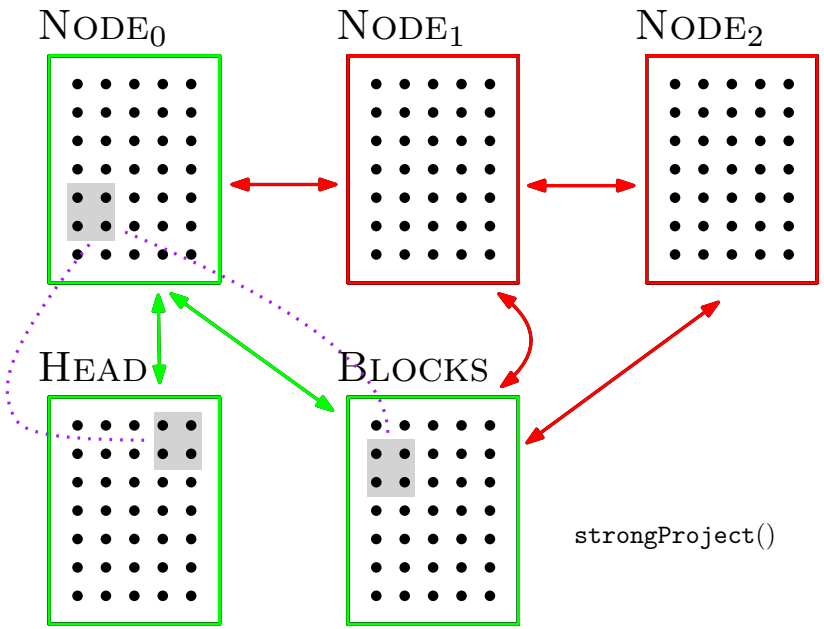


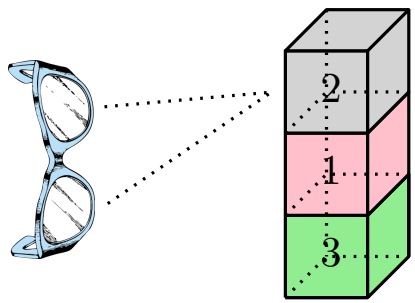


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

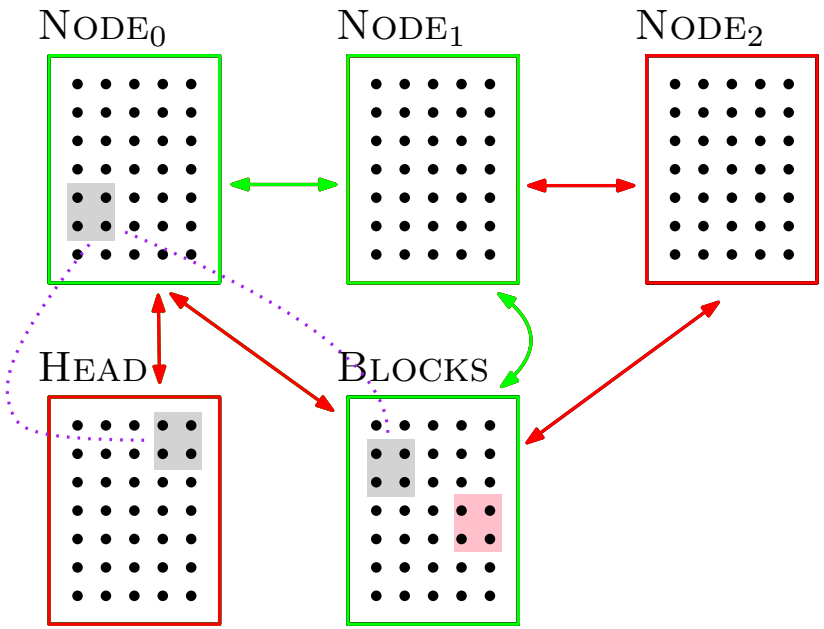


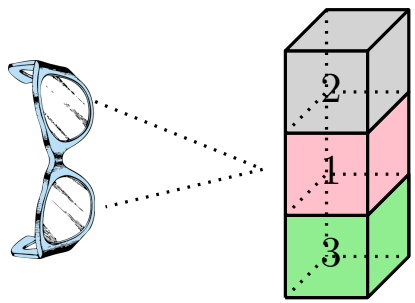


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

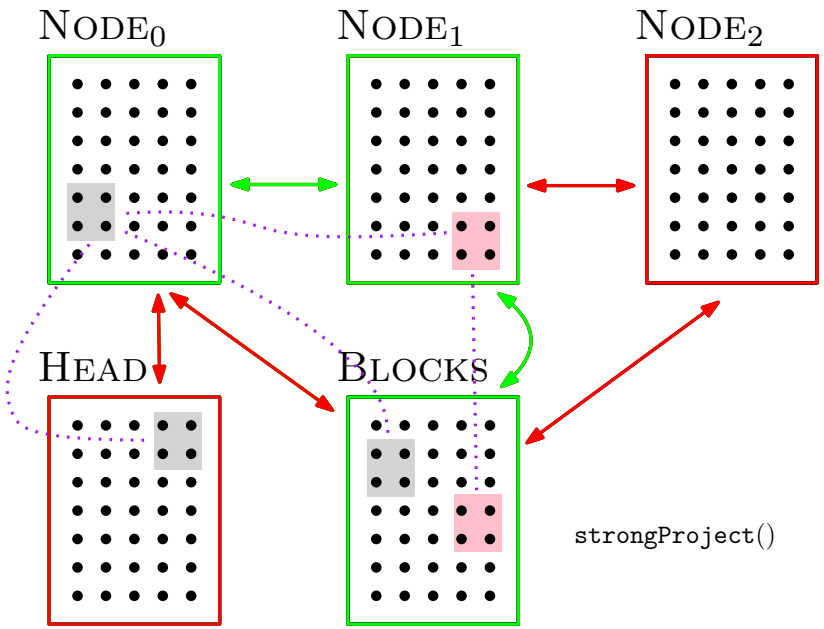


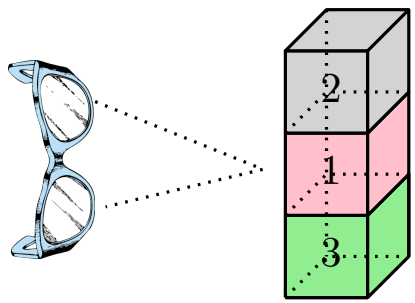


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, Blocks, Node$_0$, Node$_1$, Node$_2$, Head

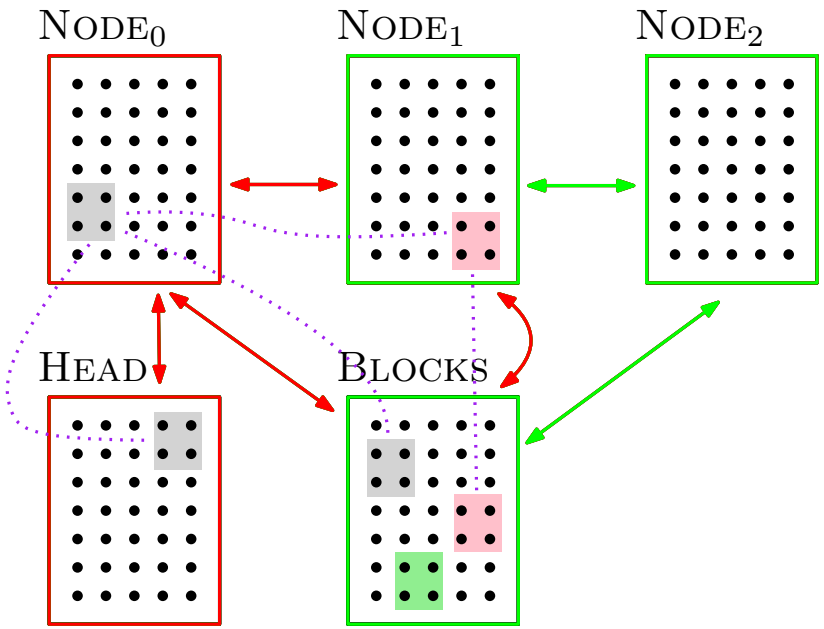


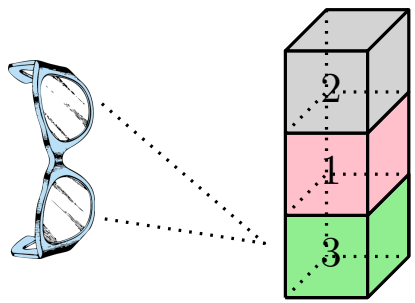


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

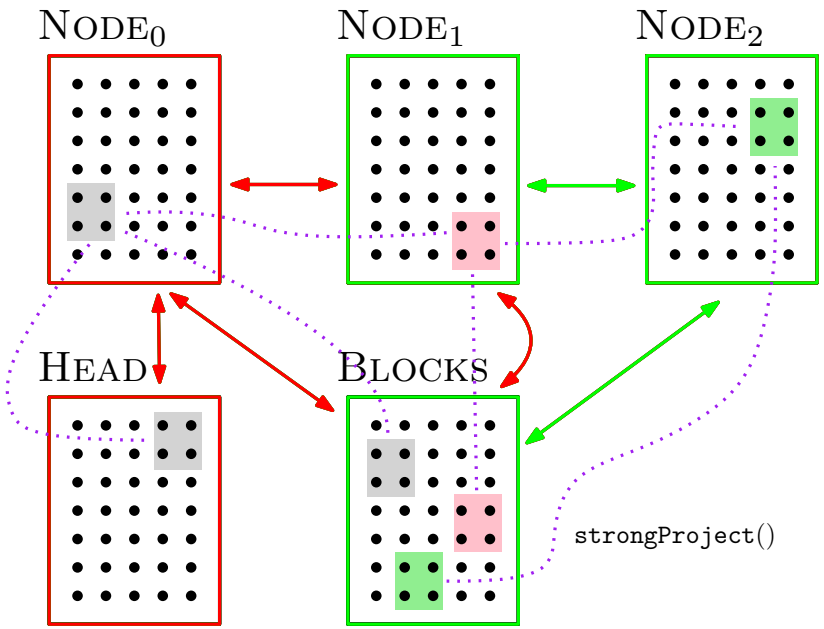


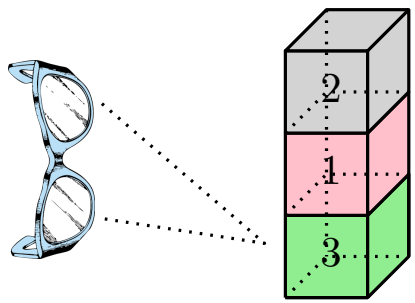


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, Blocks, Node$_0$, Node$_1$, Node$_2$, Head

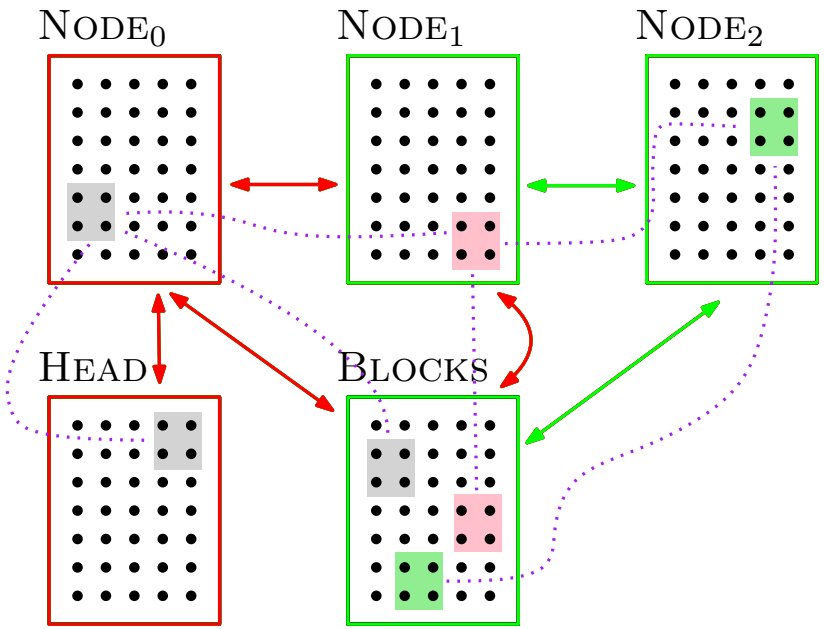


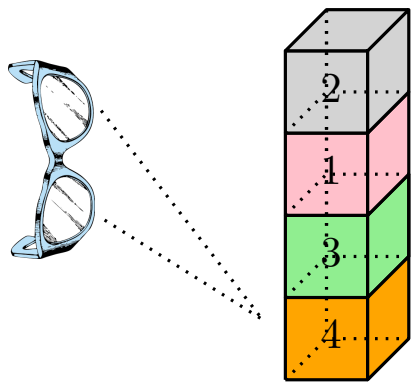


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

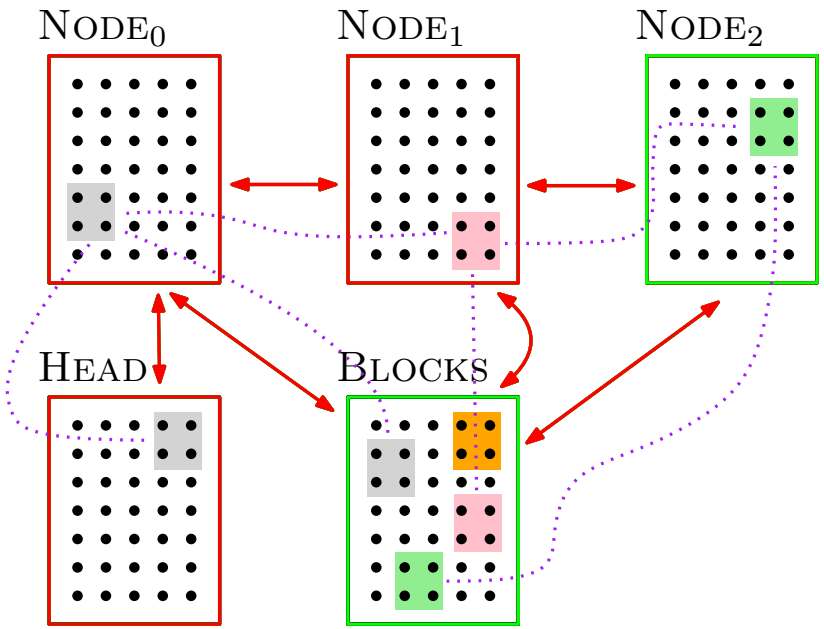


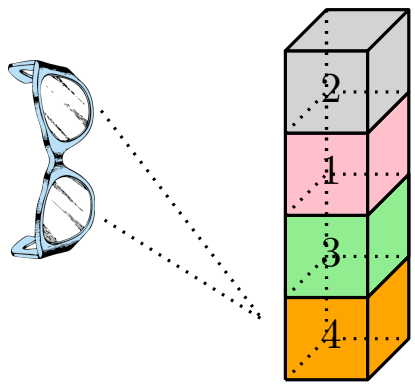


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

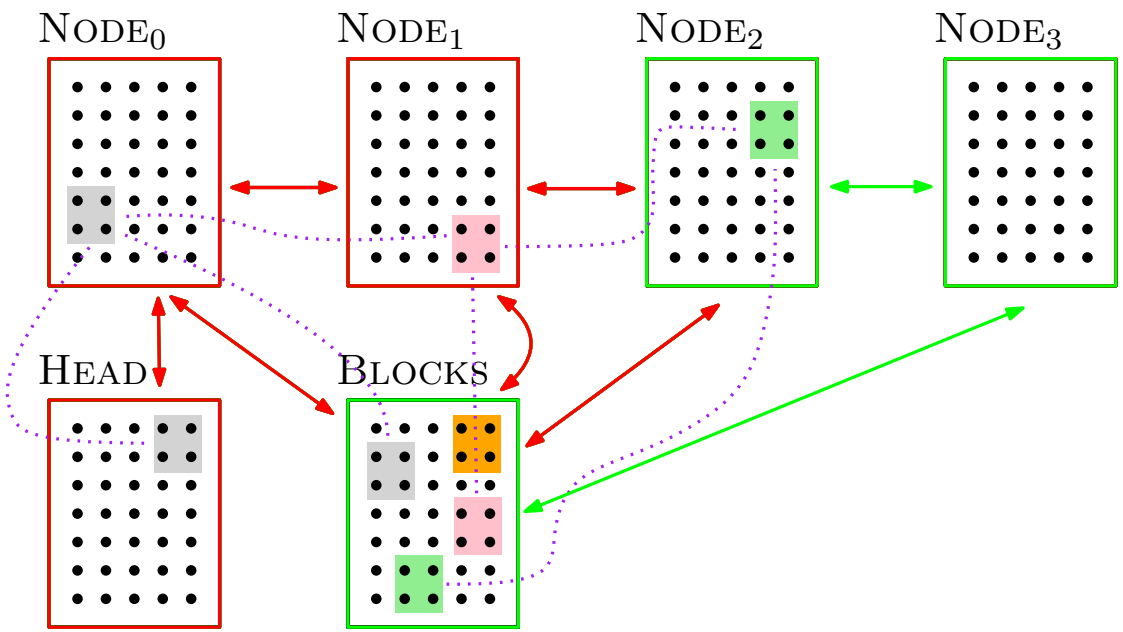


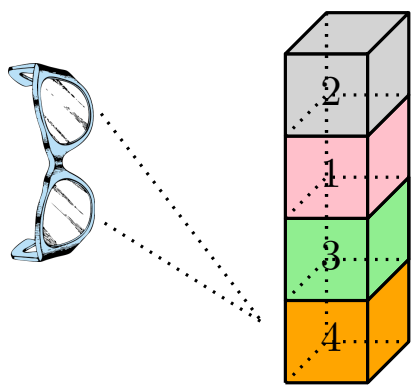


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE_0, NODE_1, NODE_2, HEAD



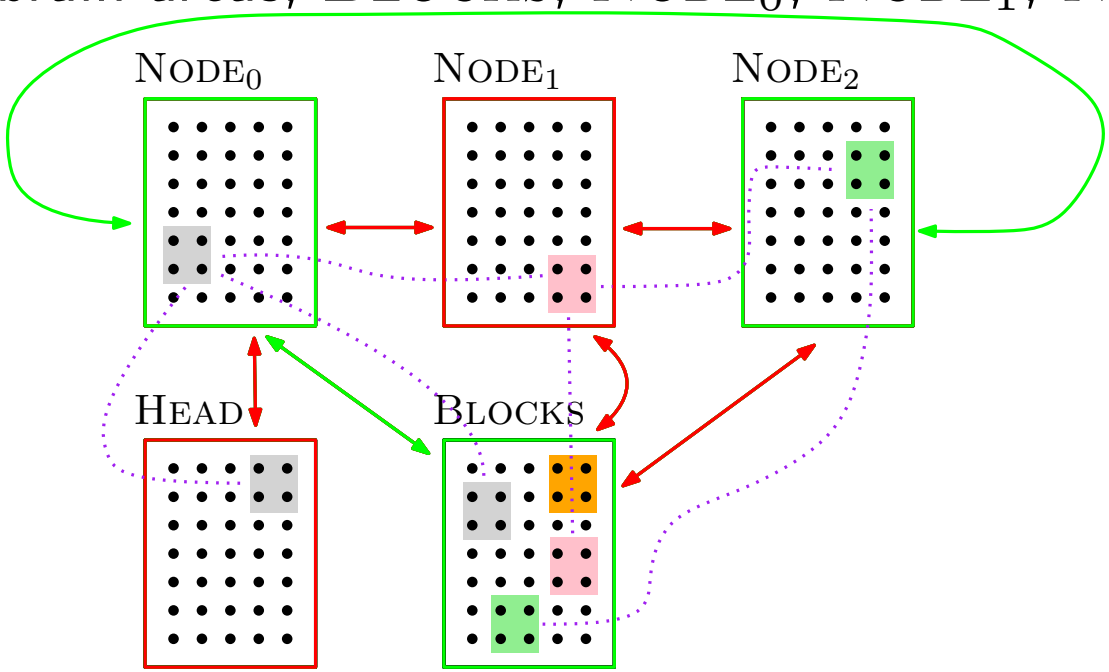


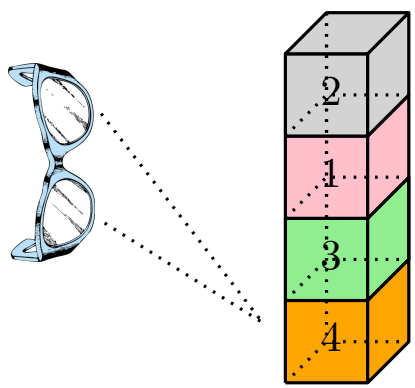
As many areas as the number of blocks... :(

The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

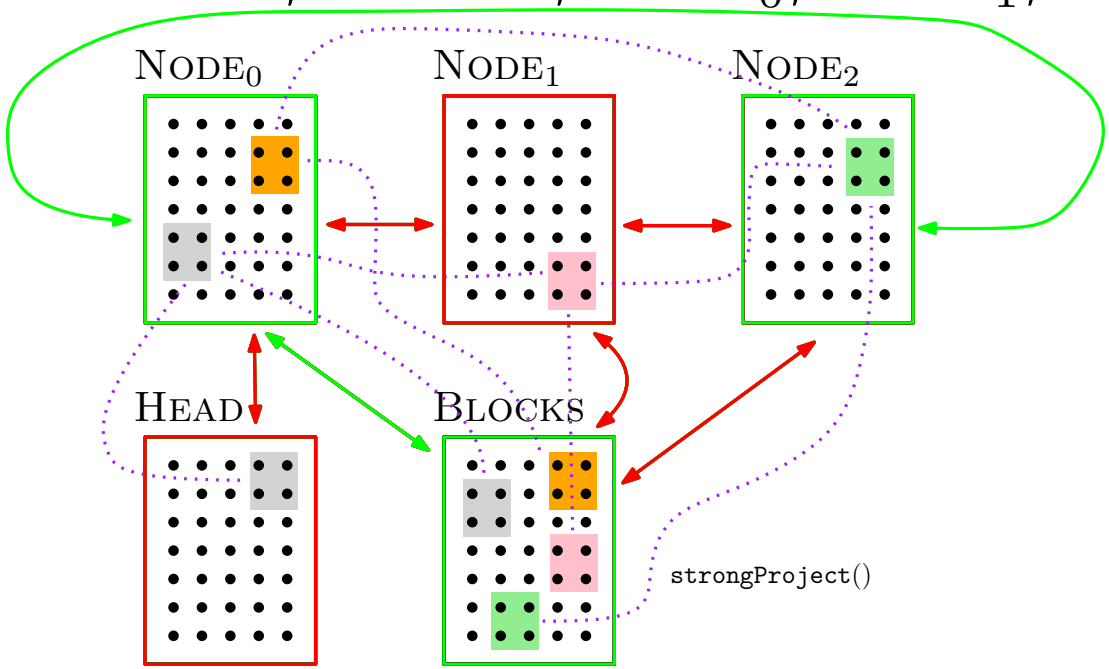


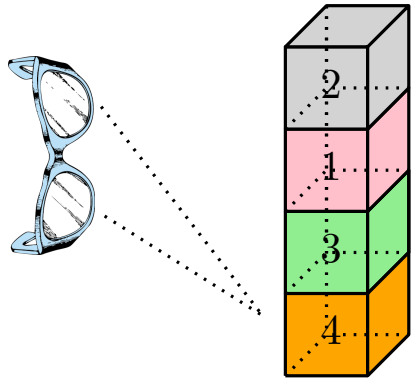


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD

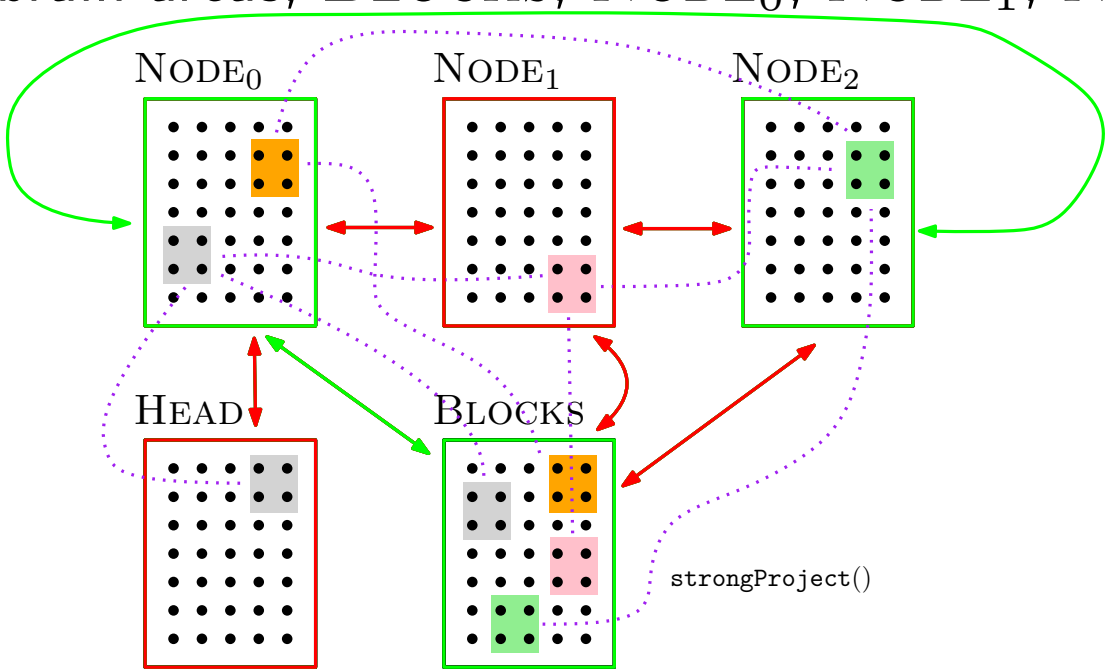


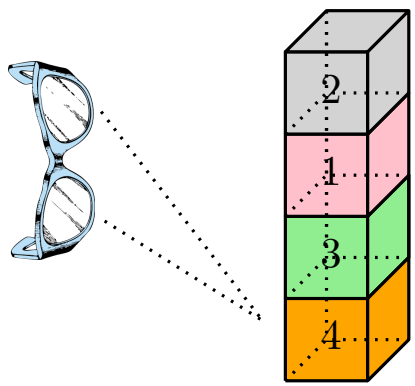


The Parser

Assumptions:

- single stack
- brain as undirected graph
- 5 brain areas, BLOCKS, NODE$_0$, NODE$_1$, NODE$_2$, HEAD





Chaining: novelty. Number of blocks represented depends only on n (number of neurons per area) and k (the CAP)

Parser Code

Algorithm 2: PARSER (S)

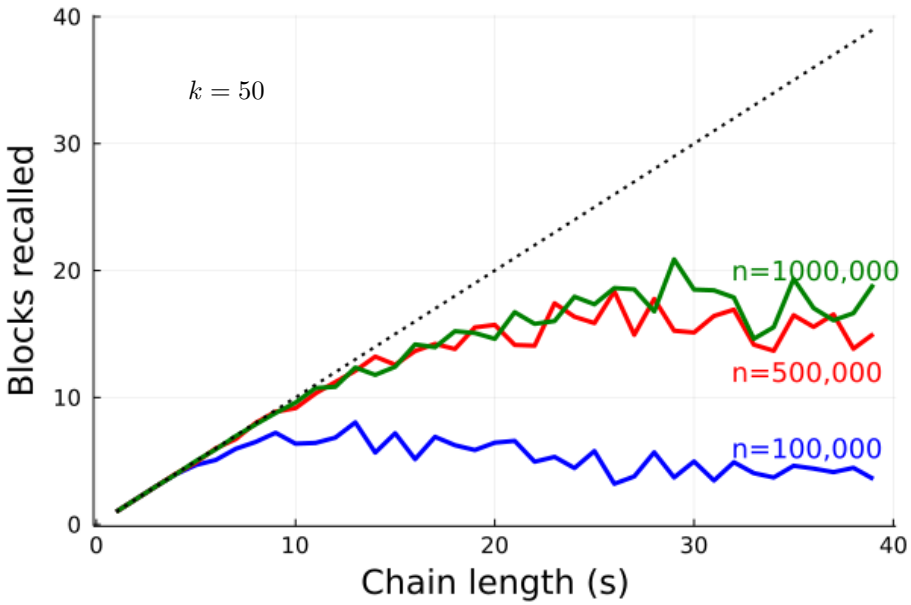
input: a stack S of blocks $b_1, b_2, \ldots, b_s$.

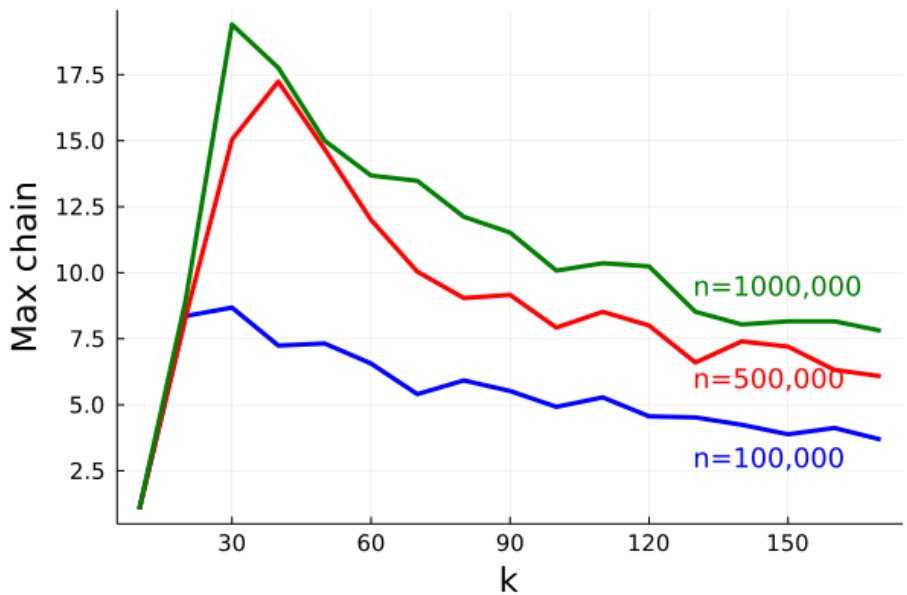
```
1  disinhibitArea({BLOCKS, HEAD, NODE_0});
2  disinhibitFiber({(HEAD, NODE_0), (NODE_0, BLOCKS)});
3  fire(b_1);
4  strongProject();
5  inhibitArea({HEAD});
6  inhibitFiber({(HEAD, NODE_0), (NODE_0, BLOCKS)});
7  foreach i with 2 ≤ i ≤ s do
8      p = (i − 2) mod 3; c = (i − 1) mod 3;
9      disinhibitArea({NODE_c});
10     disinhibitFiber({(NODE_p, NODE_c), (NODE_c, BLOCKS)});
11     fire(b_i);
12     strongProject();
13     inhibitArea({NODE_p});
14     inhibitFiber({(NODE_p, NODE_c), (NODE_c, BLOCKS)});
15 end
16 inhibitArea({BLOCKS, NODE_(s−1) mod 3});
```

Chaining Experiments

Due to overlap of assemblies, chaining may fail

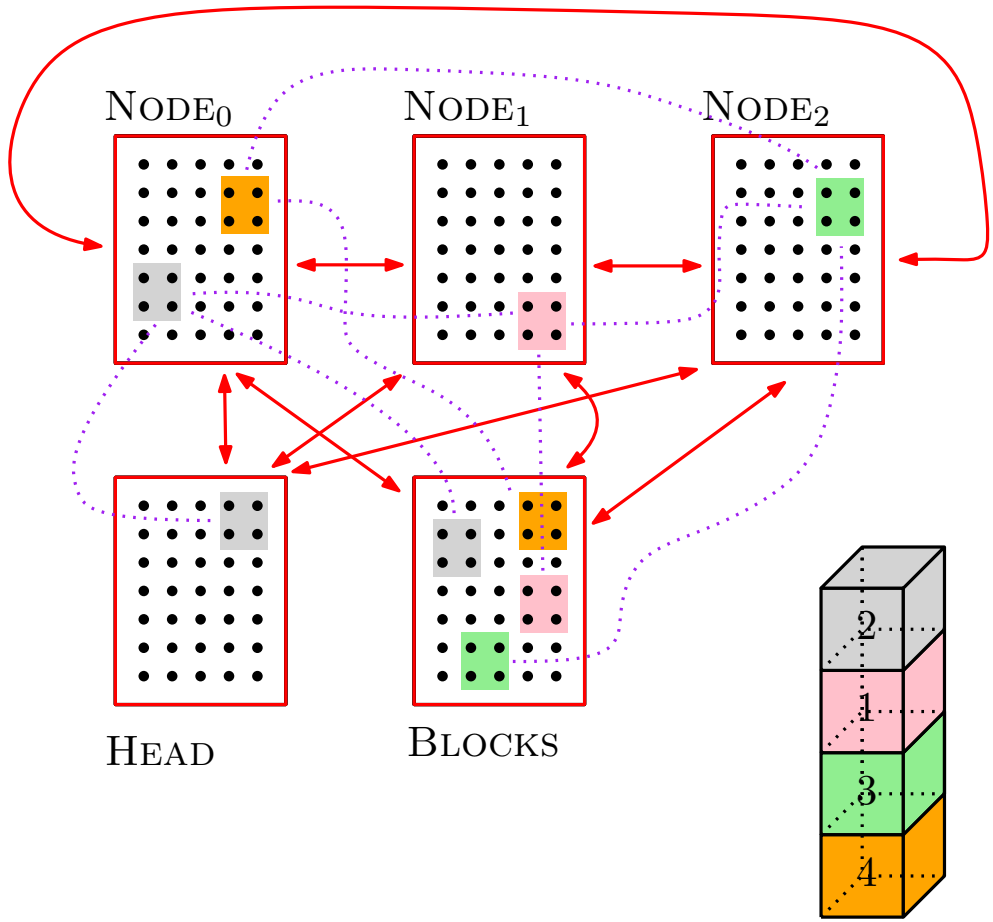
Experiments conducted averaging over 50 trials





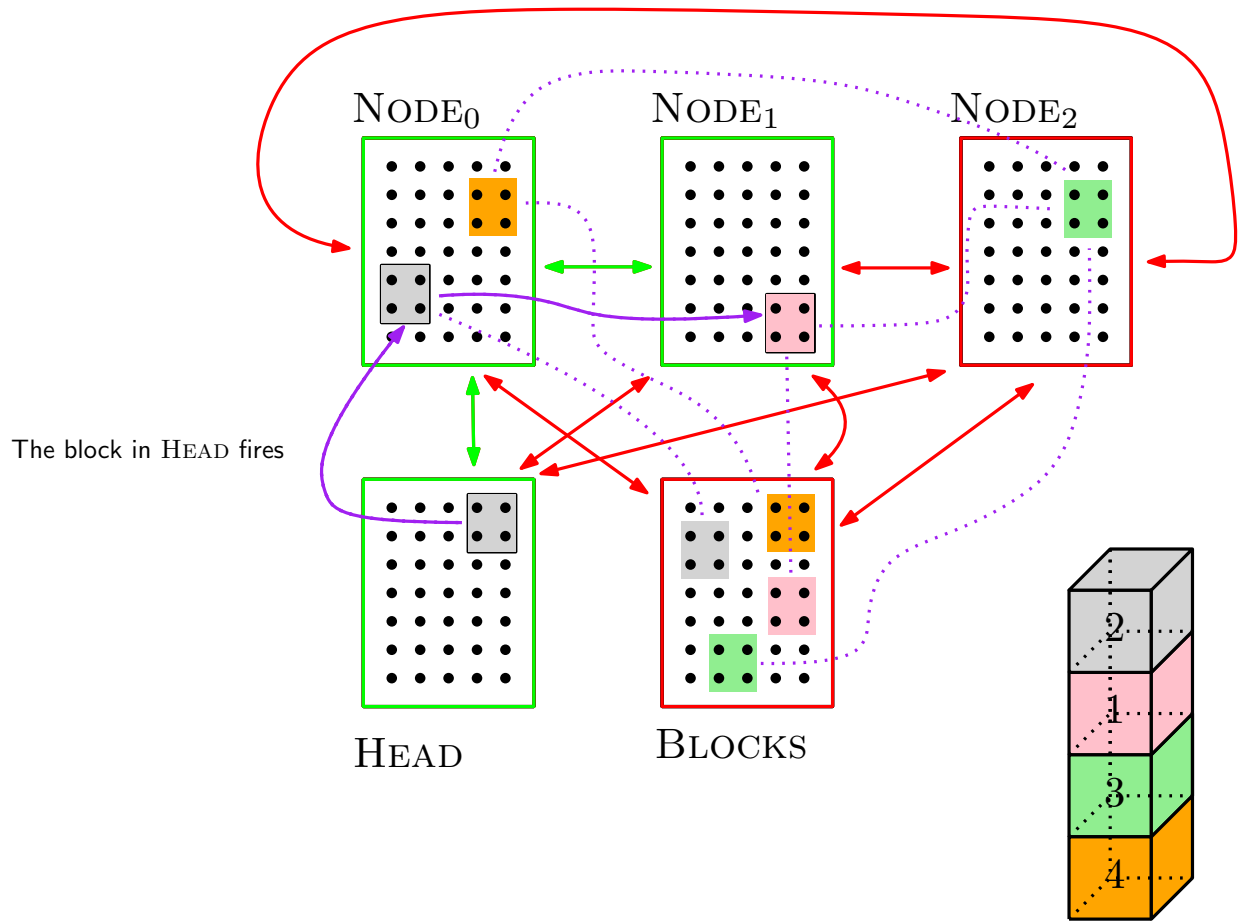
Actions

- *Removing the top block*



Actions

- *Removing the top block*



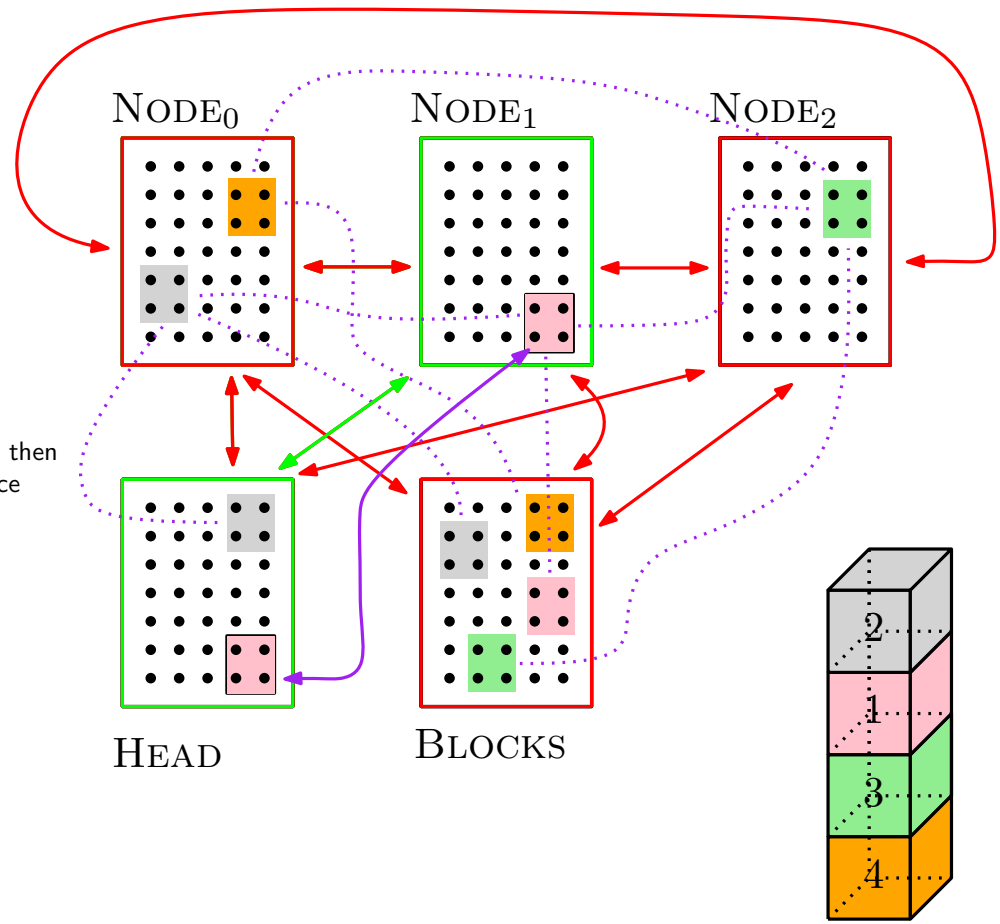
Actions

- *Removing the top block*

The last active assembly in HEAD will point to the new top

The model doesn't forget old connections, it simply doesn't use them again

Now, `project` $(\text{NODE}_1, \text{HEAD})$ and then `project` $(\text{HEAD}, \text{NODE}_1)$ to reinforce



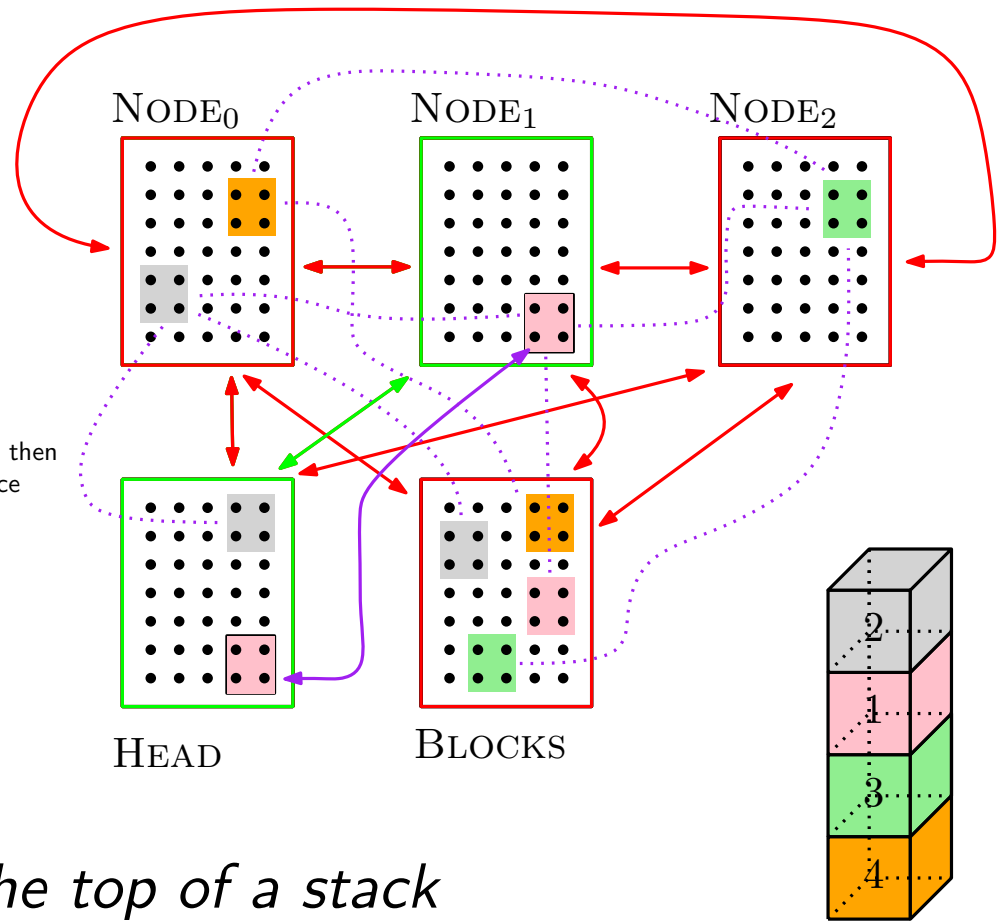
Actions

- *Removing the top block*

The last active assembly in HEAD will point to the new top

The model doesn't forget old connections, it simply doesn't use them again

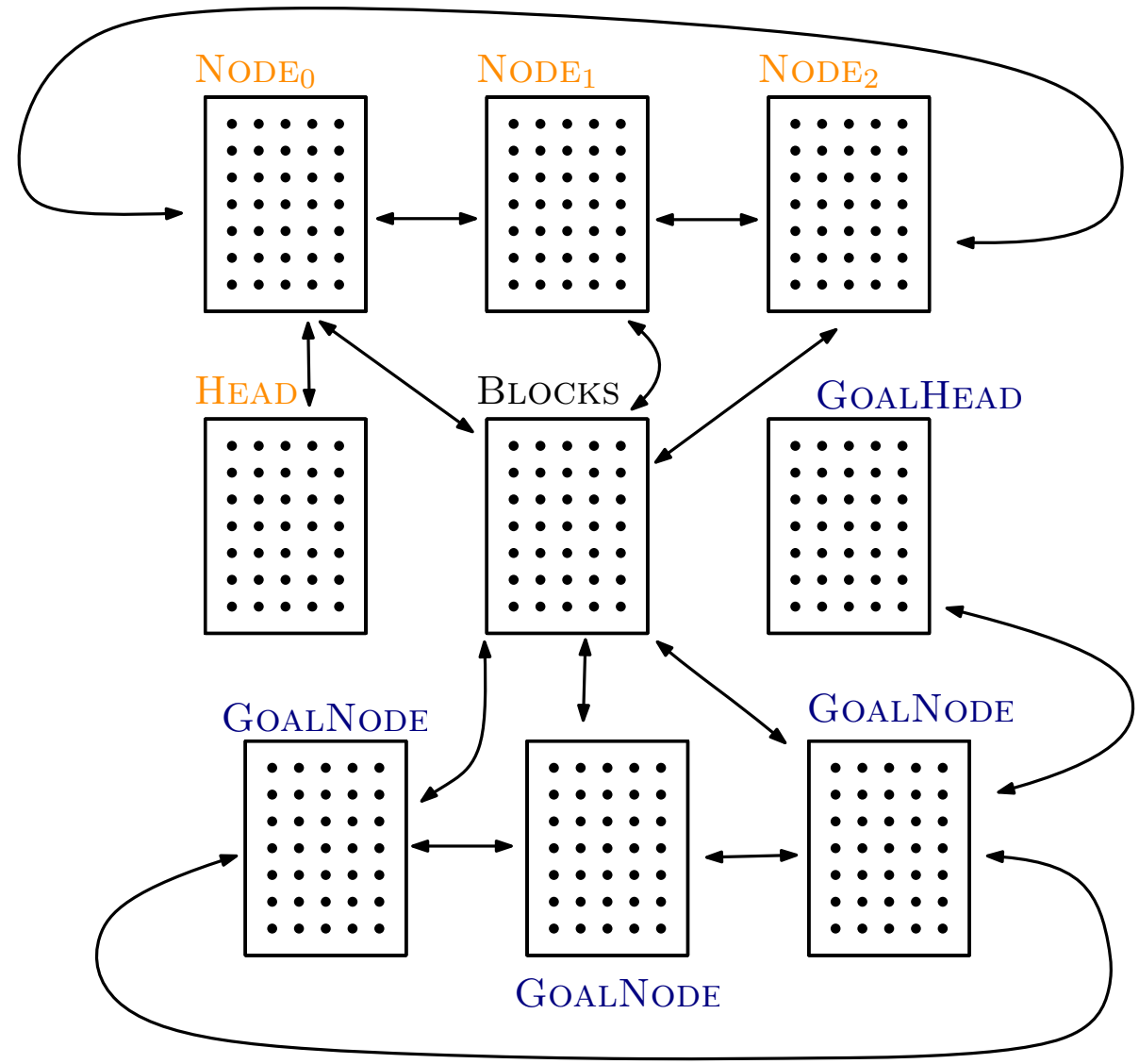
Now, `project` $(\text{NODE}_1, \text{HEAD})$ and then `project` $(\text{HEAD}, \text{NODE}_1)$ to reinforce



- *Put a block from the table to the top of a stack*

Comparisons

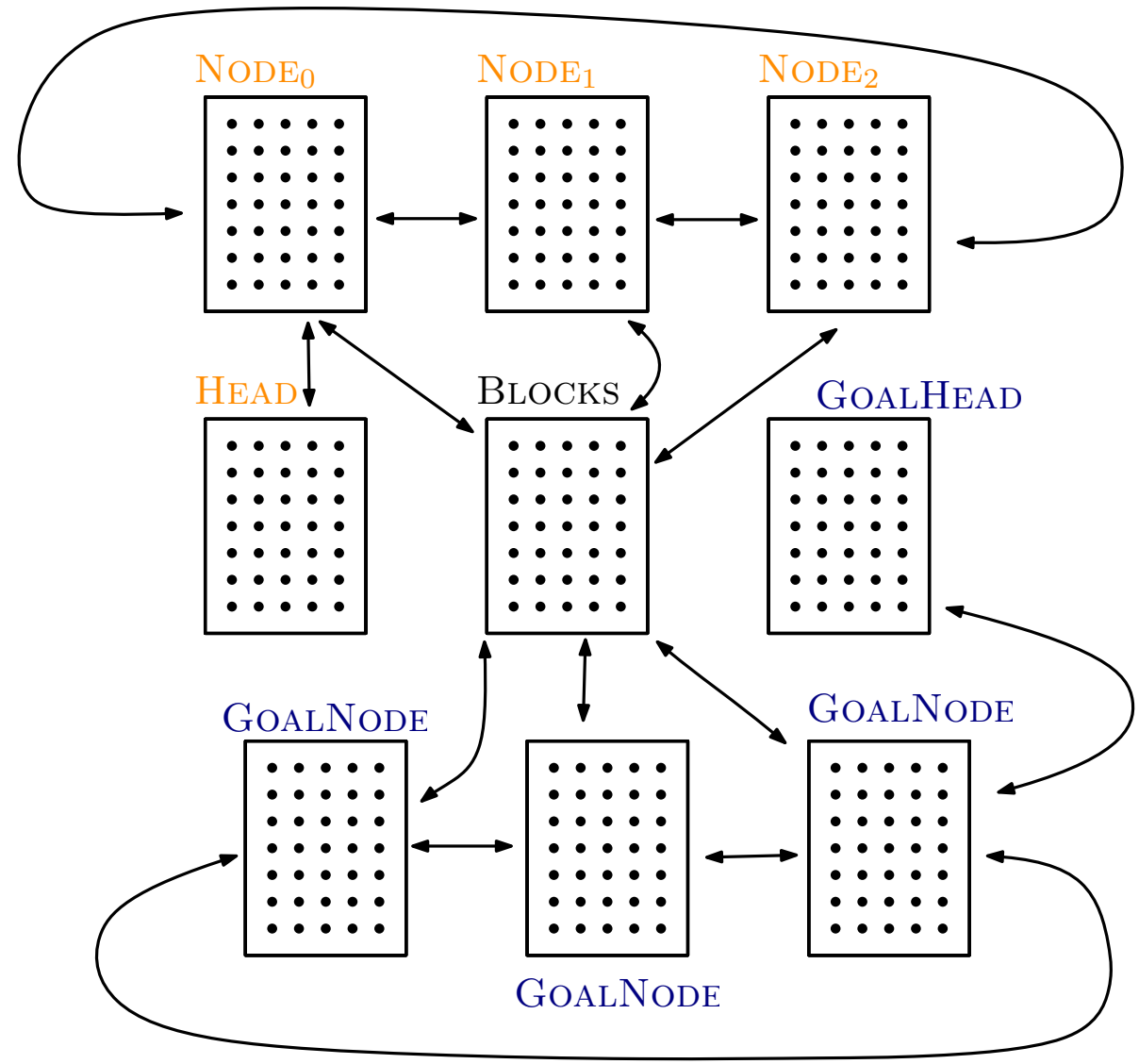
We have to save both the initial configuration and the goal configuration



Comparisons

We have to save both the initial configuration and the goal configuration

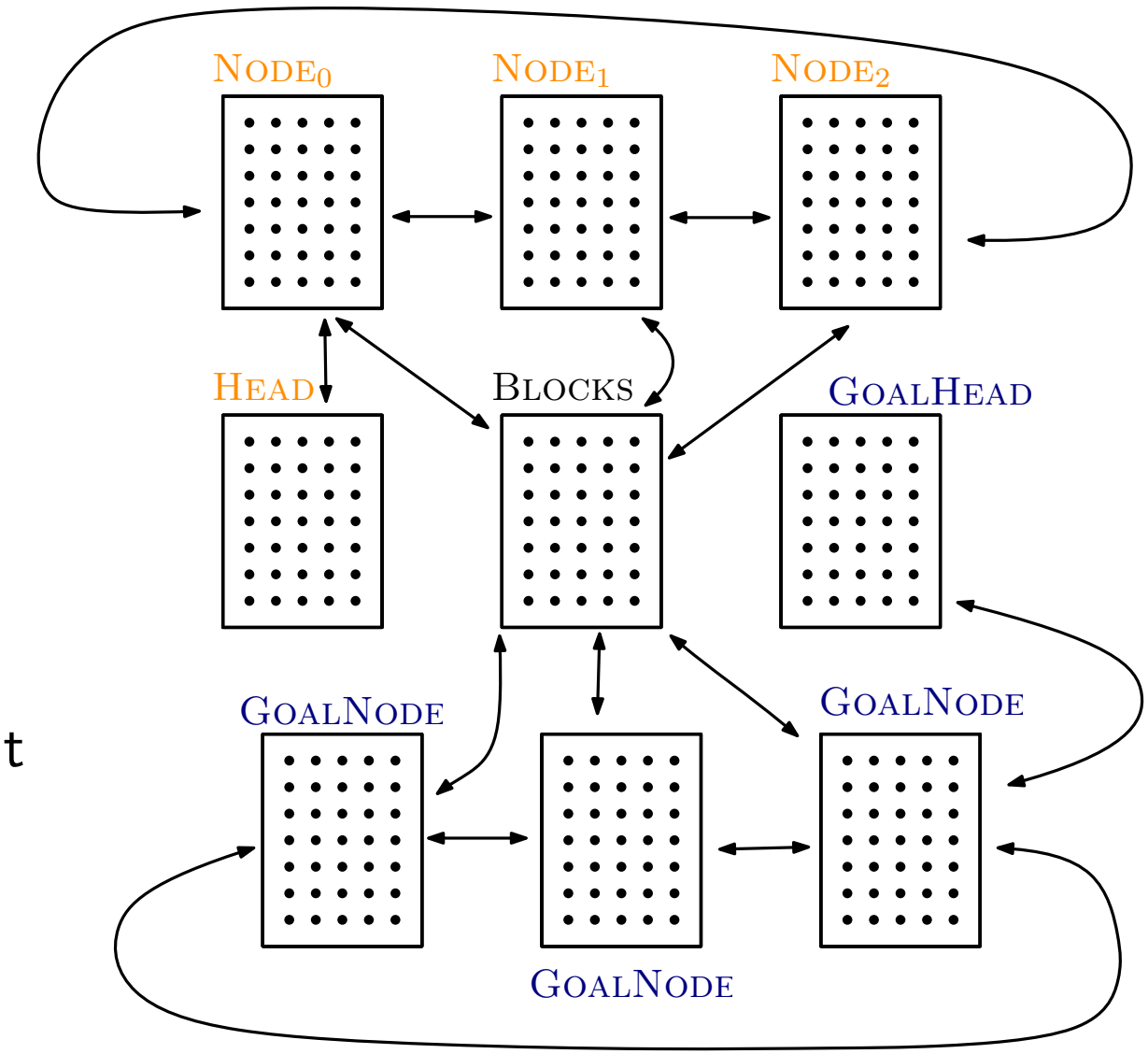
- *Intersect*: scrolls down the two stacks until reaching the last block, then starts comparing each pair of blocks by scrolling up



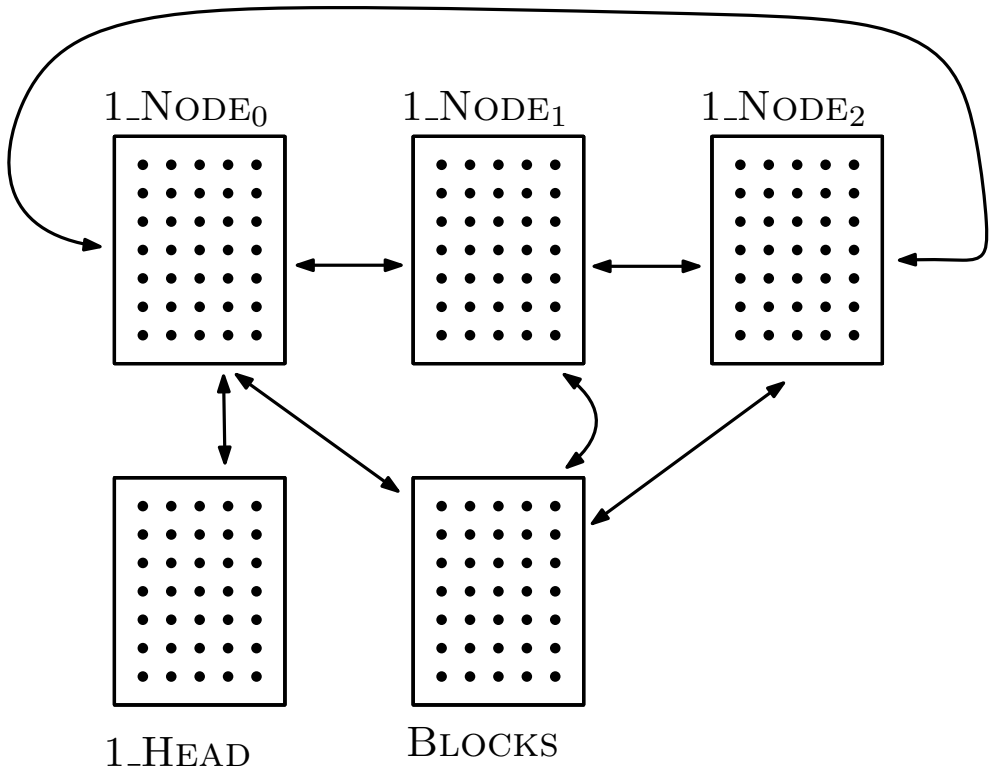
Comparisons

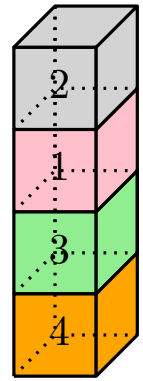
We have to save both the initial configuration and the goal configuration

- *Intersect*: scrolls down the two stacks until reaching the last block, then starts comparing each pair of blocks by scrolling up
- By combining these operations we implement the planning strategies

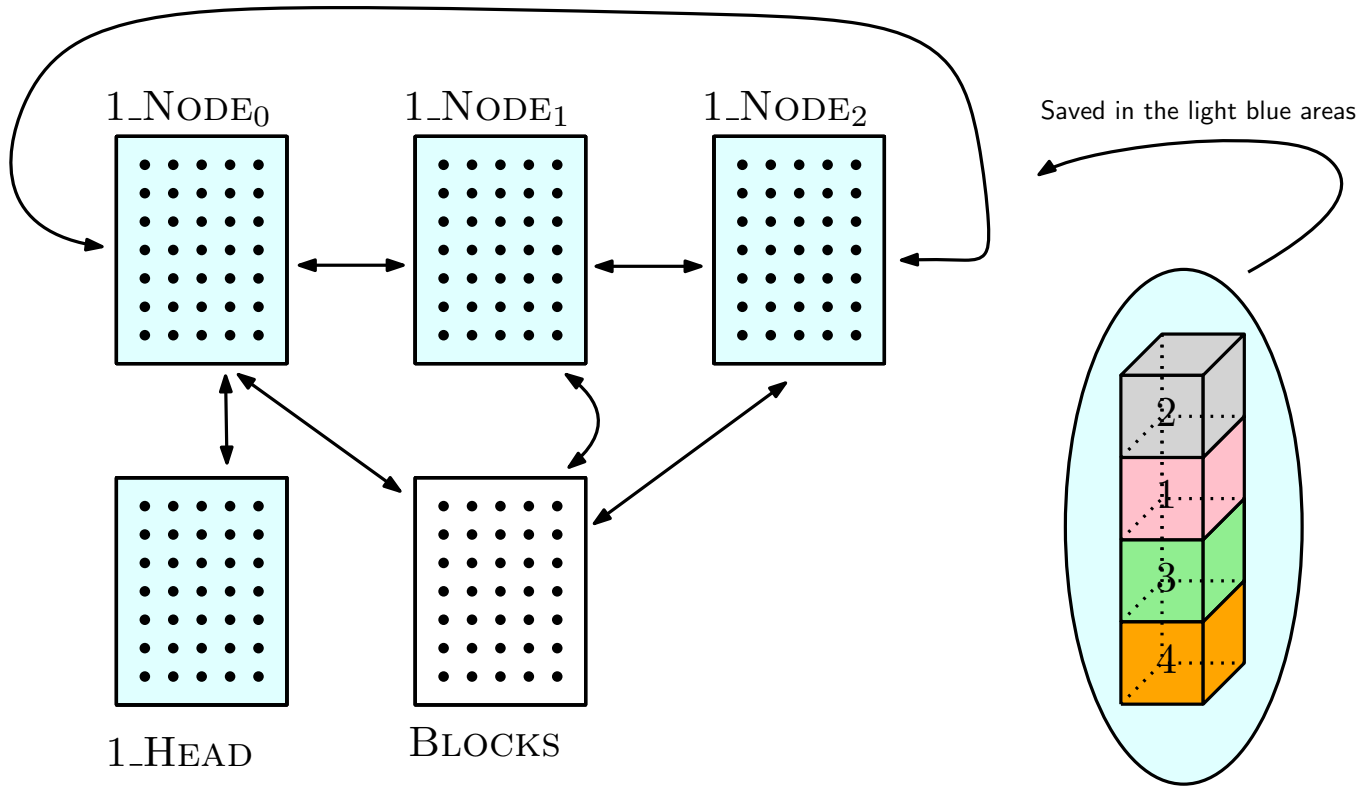


Multiple Stacks Case

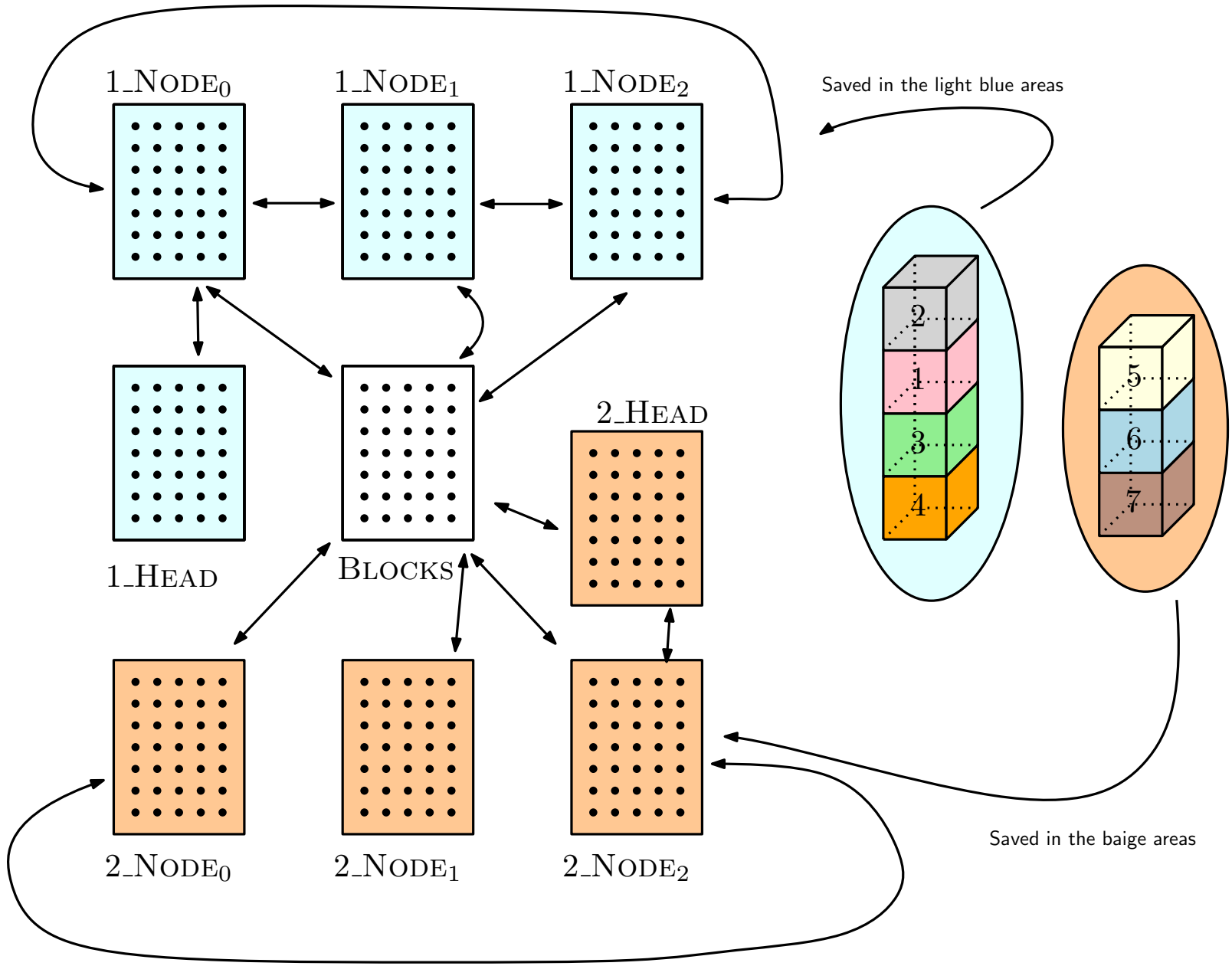




Multiple Stacks Case

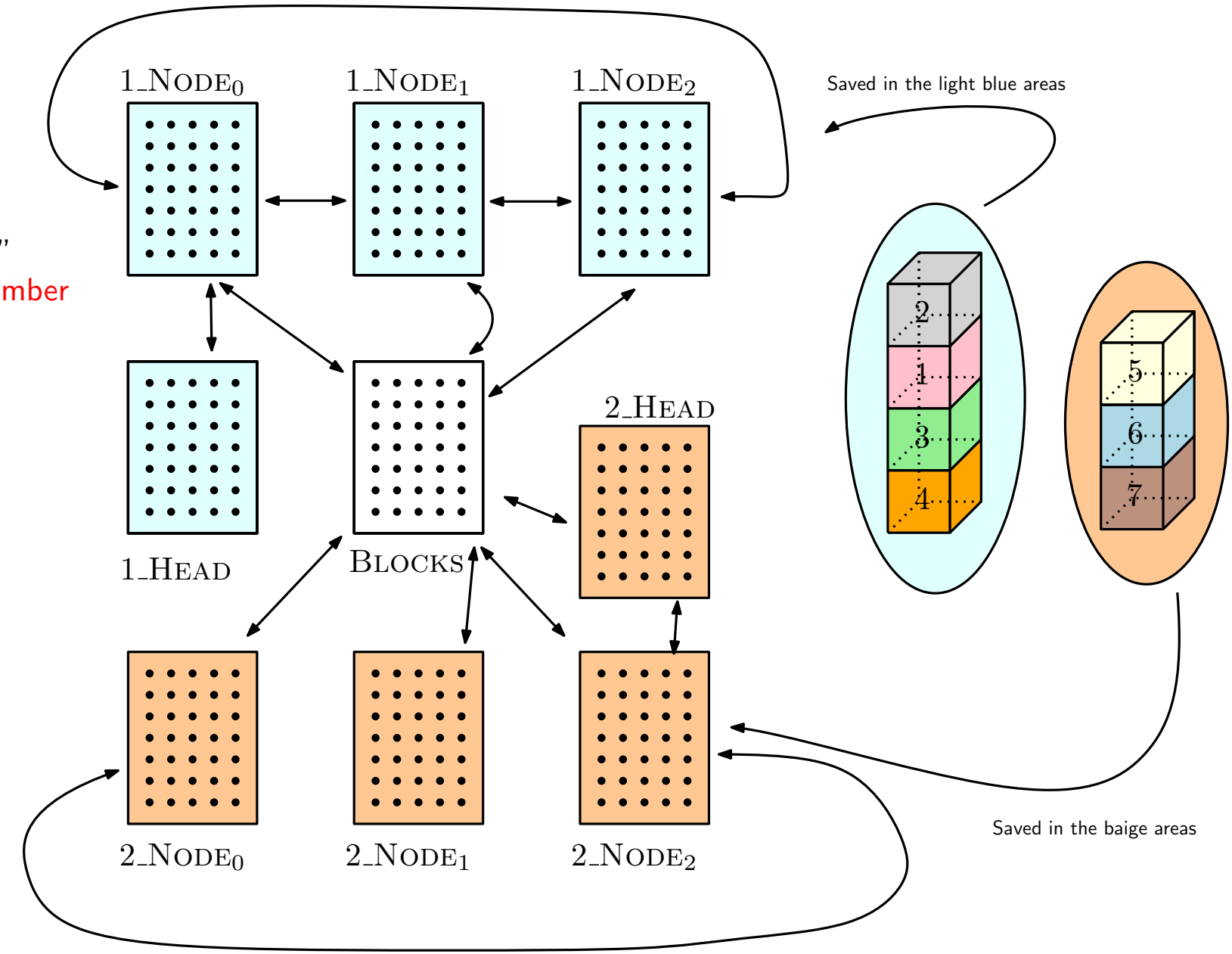


Multiple Stacks Case



Multiple Stacks Case

As many "saving" regions as the number of stacks: **to be improved** (like chaining)



Discussion

- Demonstrated experimentally that reasonably large and complex programs in the assembly calculus can execute correctly and reliably
- Shown the realization of a *list-like data structure* which makes use of a *constant* number of brain regions
- Shown how simple manipulations of the data structure (removing or putting a block) can be realized by making use of a constant number of brain regions
- **Bottleneck**: the parsing. Its reliability depends on the ratio between the number of neurons and the size of the assemblies in each region
  - Must be the object of further investigation

Future Directions

After syntactic analysis in language [Mitropolsky et al., TACL 2021] and blocks world planning, what comes **next** as a compelling stylized cognitive function, which could be implemented in the AC?

- *Reasoning*
- *Planning and problem solving* in less specialized domains
- *Deductive tasks* in the context of logical and constraint-based formalisms

The End

Questions?