# Associative memories

9/25/2014

# Memorized associations are ubiquitous

Stimulus

Response

“Bill”

# Memorized associations are ubiquitous

Stimulus

Response

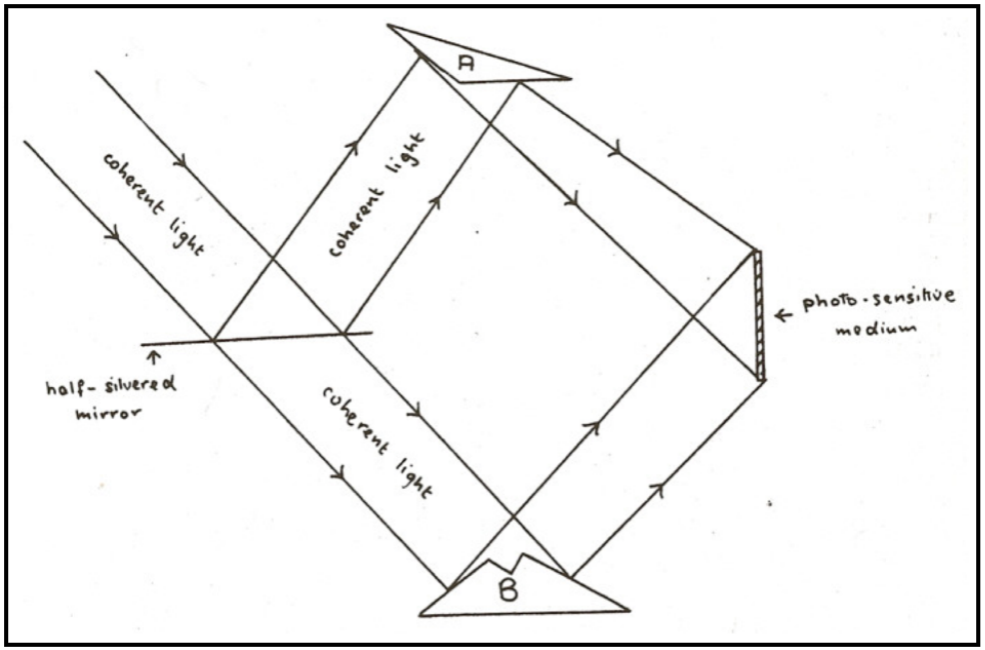
“Bill”

**Key properties:**

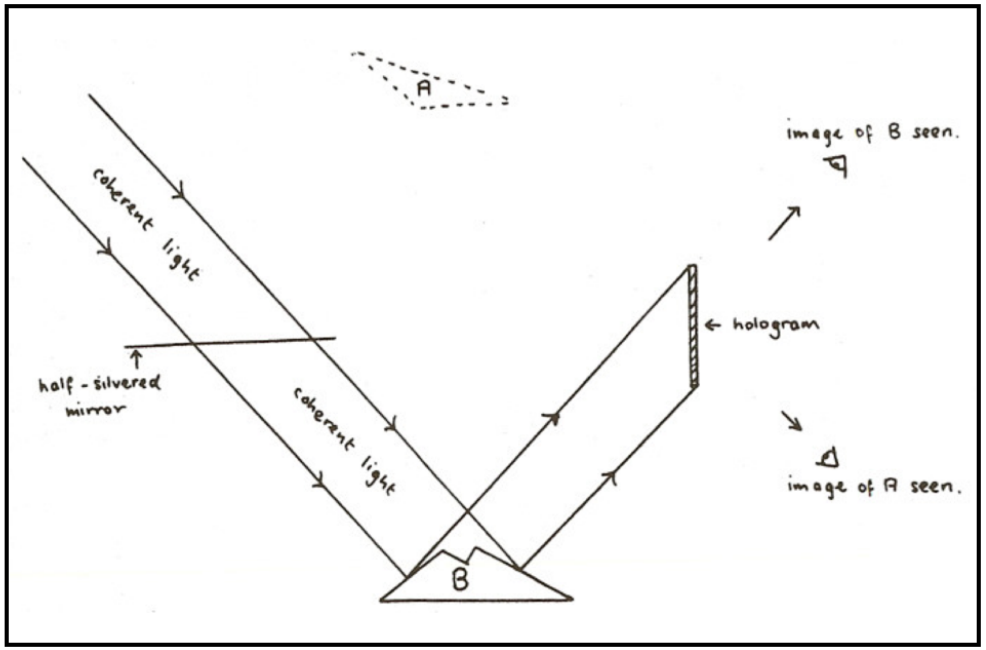
- Noise tolerance (generalization)
- Graceful saturation
- High capacity

# First attempts: Holography

van Heerden, 1963
Willshaw, Longuet-higgins, 1960s




Storage




Retrieval

Mathematically, this is the convolution-correlation scheme from class 4.

$$\phi_{r,s}(x) = r \star s = \int r(\zeta) r(x - \zeta) d\zeta$$

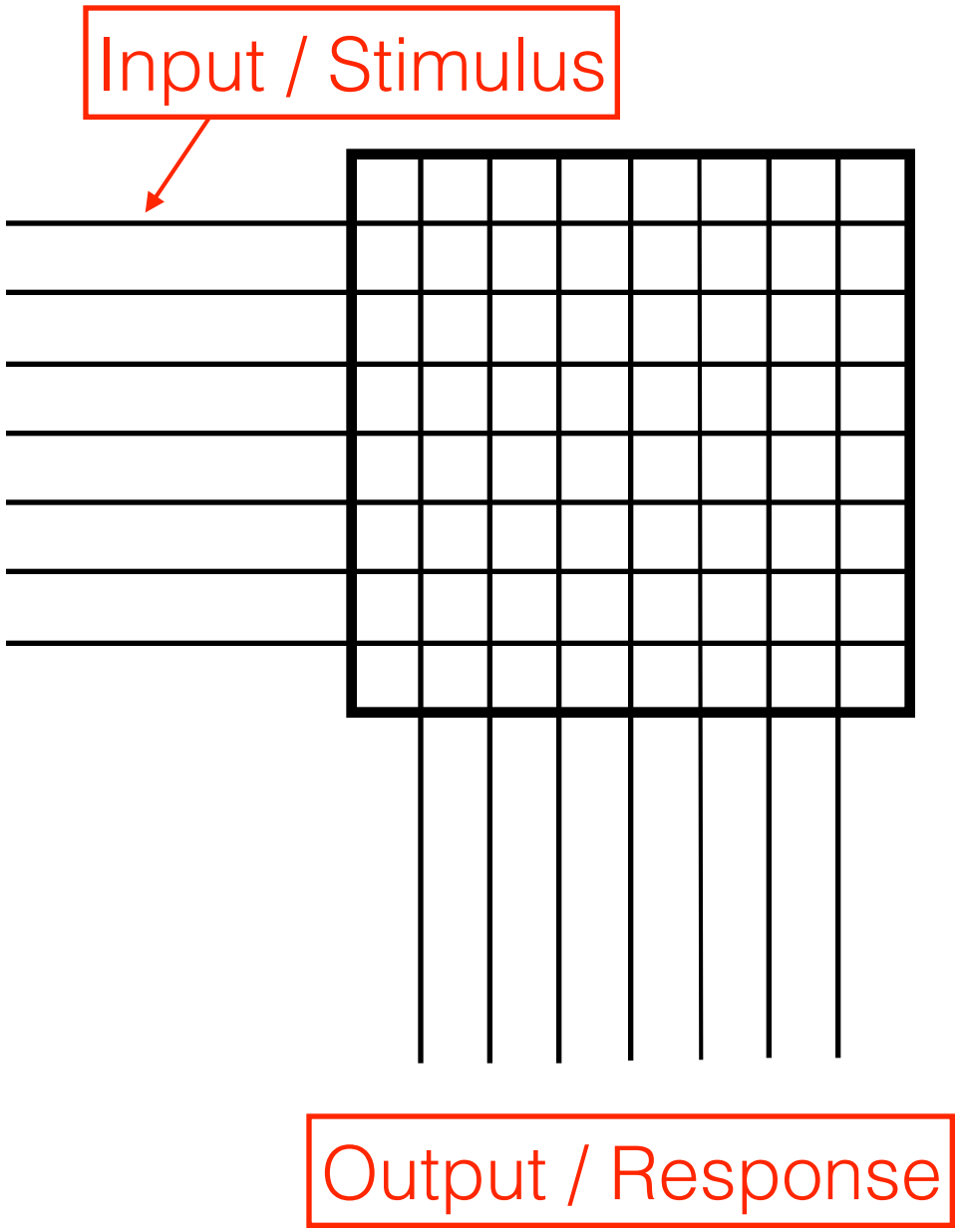
$$s \odot \phi_{r,s}(x) = \int s(\tau) \phi_{r,s} + (\tau + x) d\tau$$

# Matrix memories

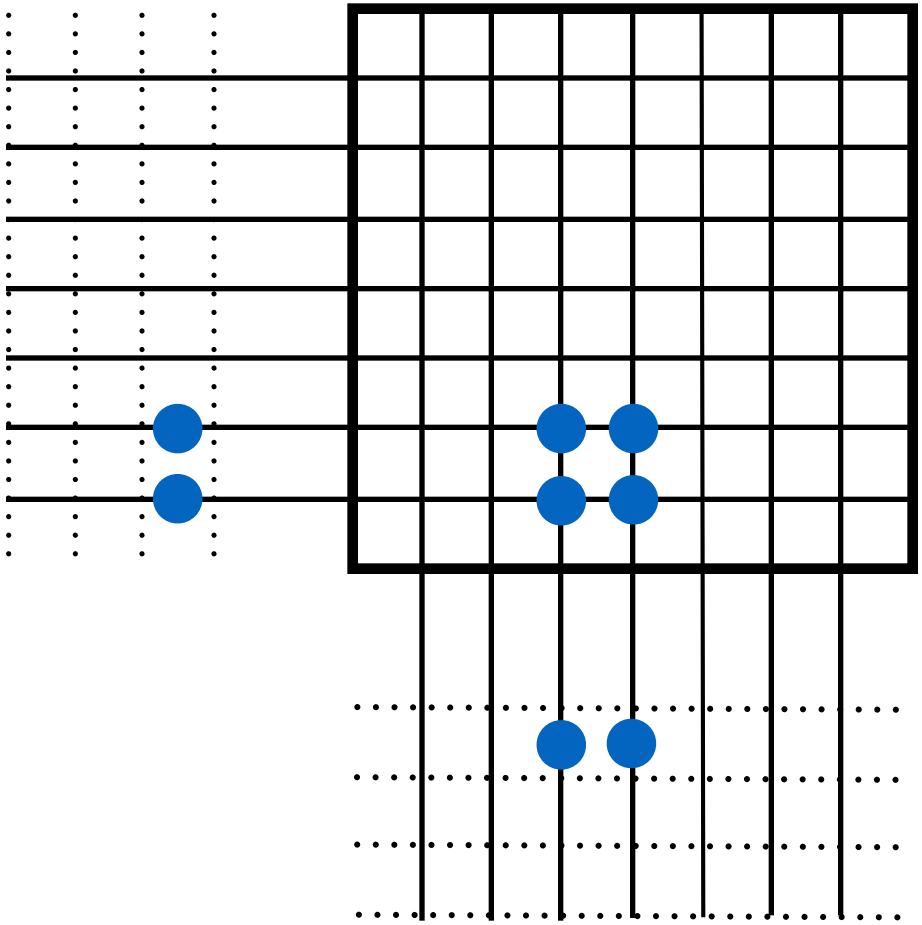
Steinbuch, 1962
Willshaw et al., 1969

Before long, it was realized that better results could be obtained with a simpler, more neurally plausible framework.

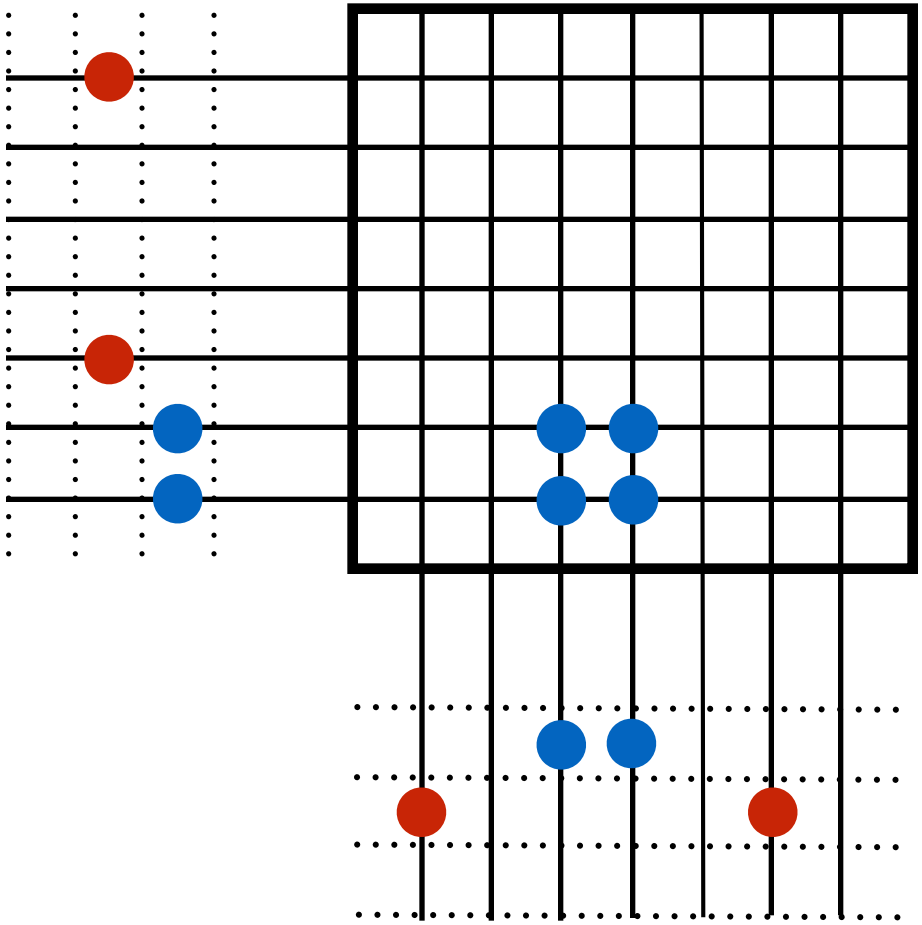
Let's explore a simple Hebbian scheme. We have input and output lines, and we strength synapses when they're on together.



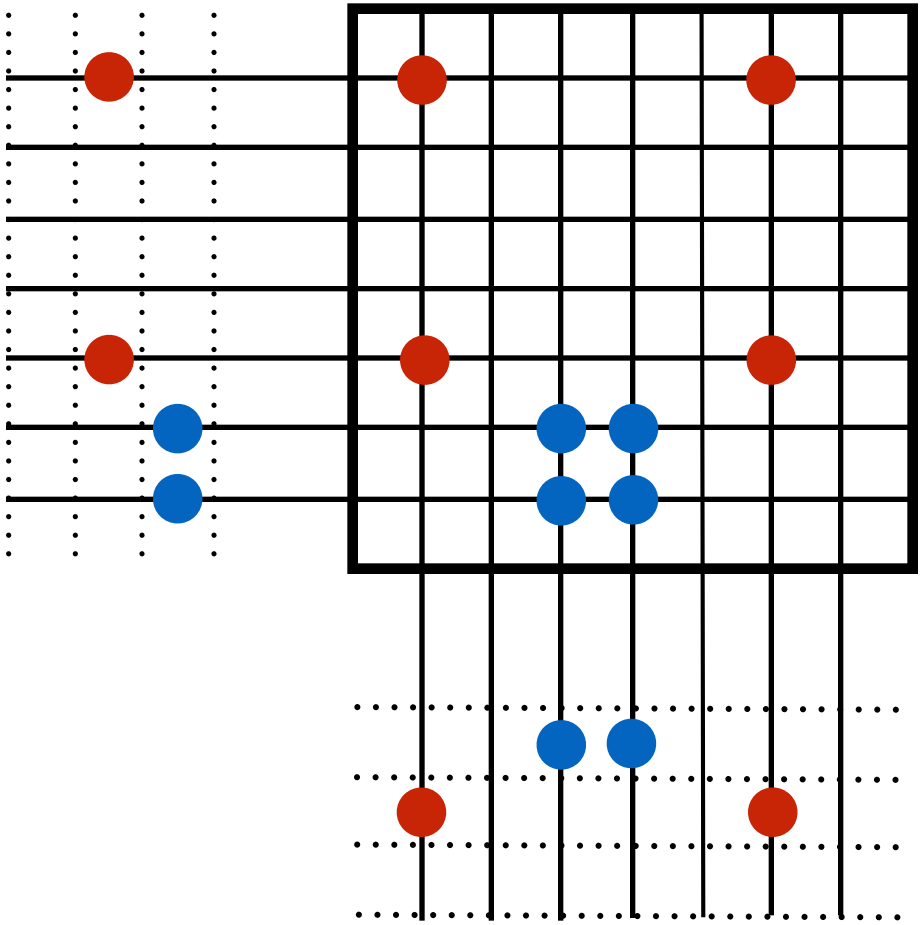
# Storage



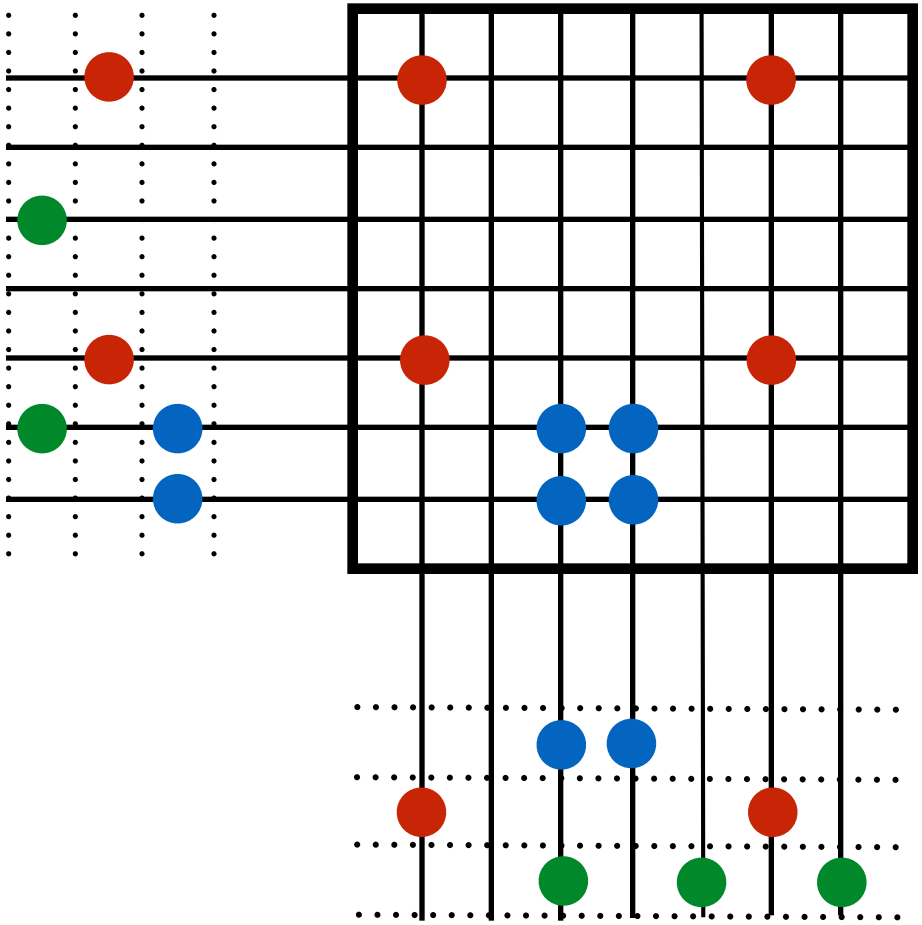
# Storage



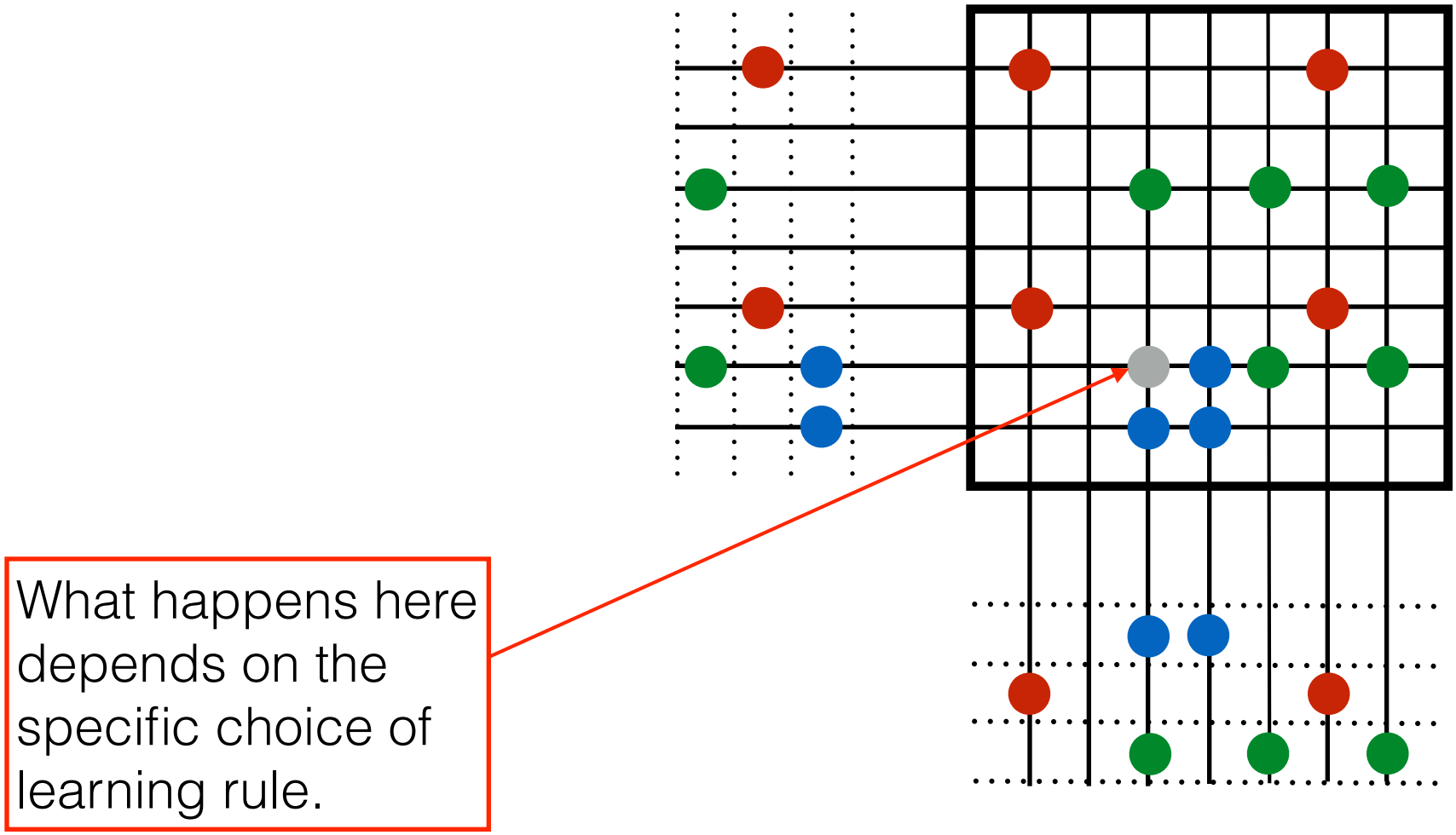
# Storage



# Storage



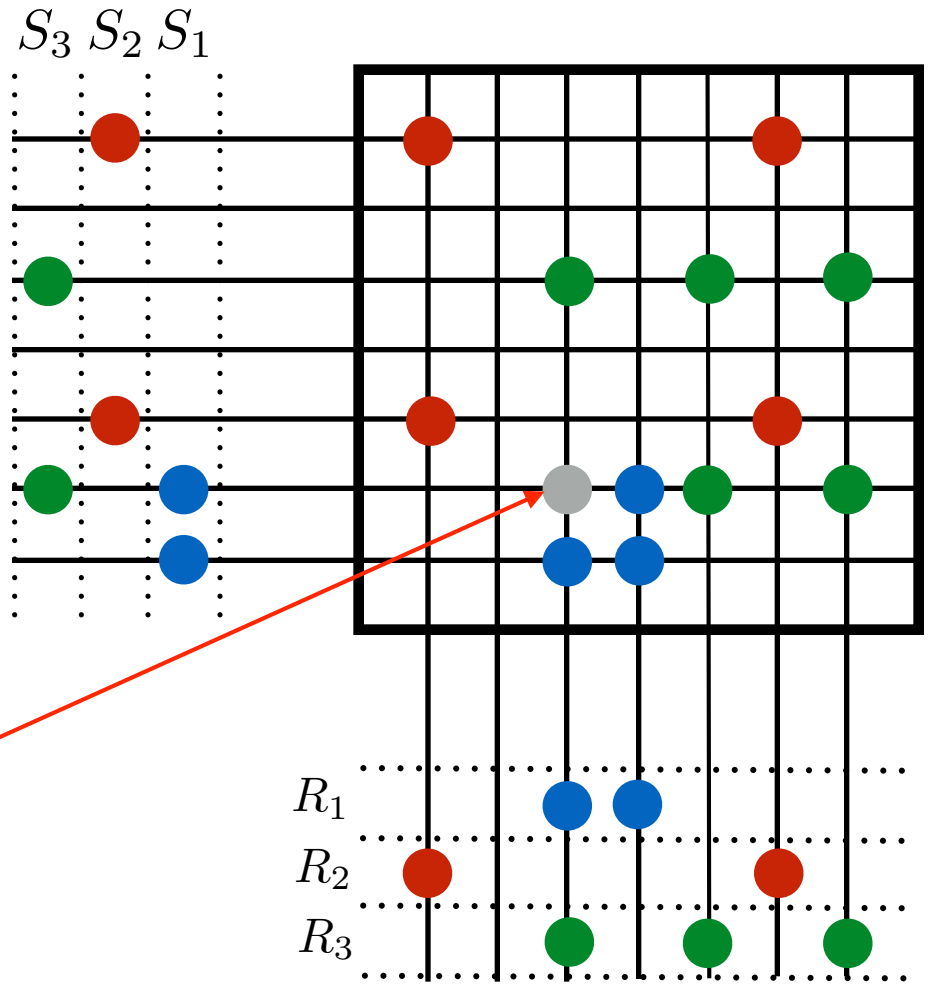
# Storage


What happens here depends on the specific choice of learning rule.

# Storage

Additive Hebb rule

$$M = \sum_{i=1}^{n} R_i S_i^T$$




What happens here depends on the specific choice of learning rule.

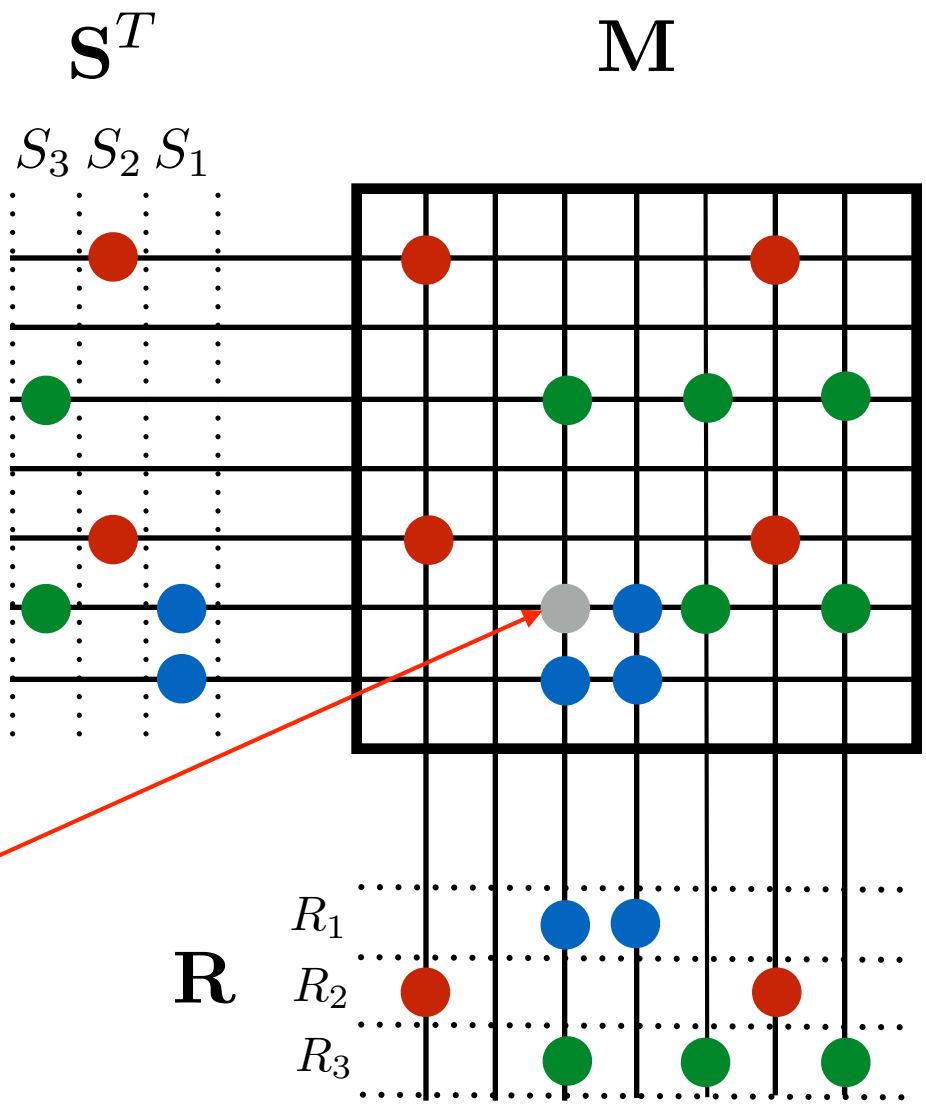
# Storage

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{R}\mathbf{S}^T$$

What happens here depends on the specific choice of learning rule.

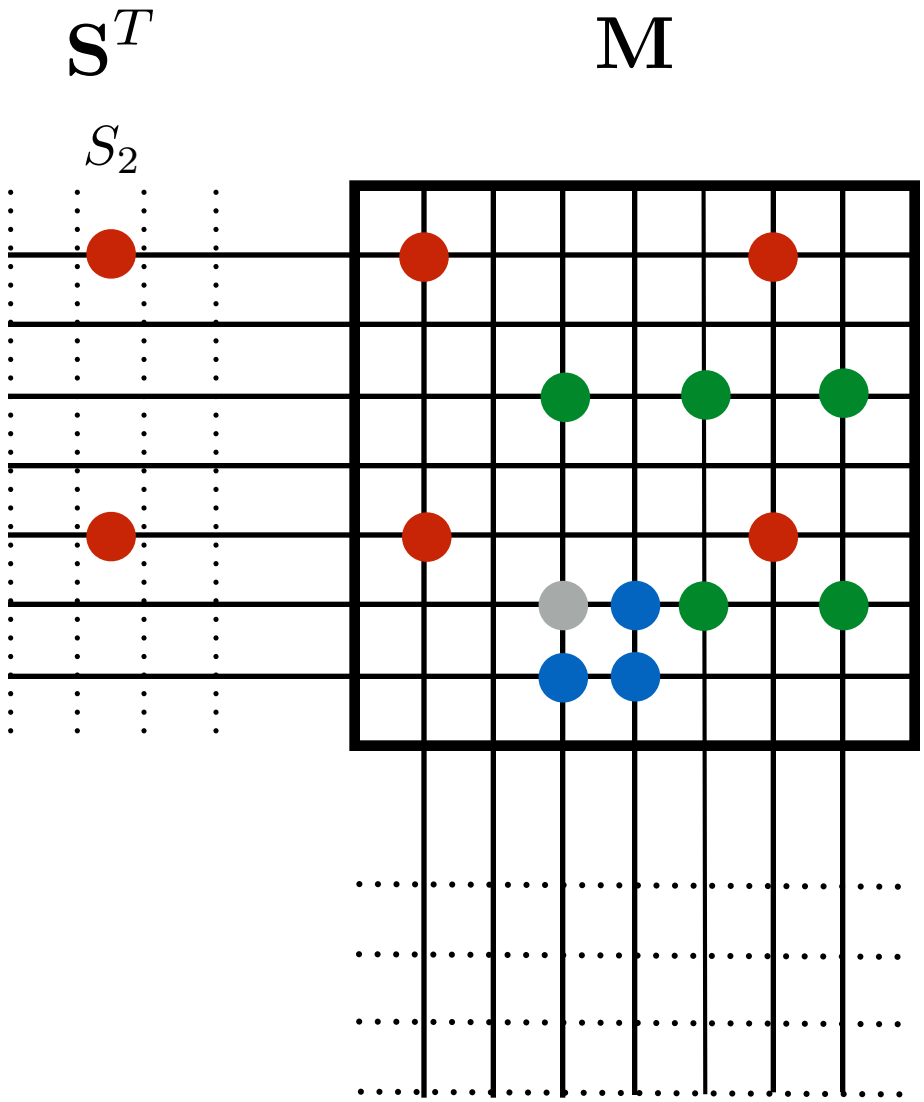


# Storage

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{R}\mathbf{S}^T$$

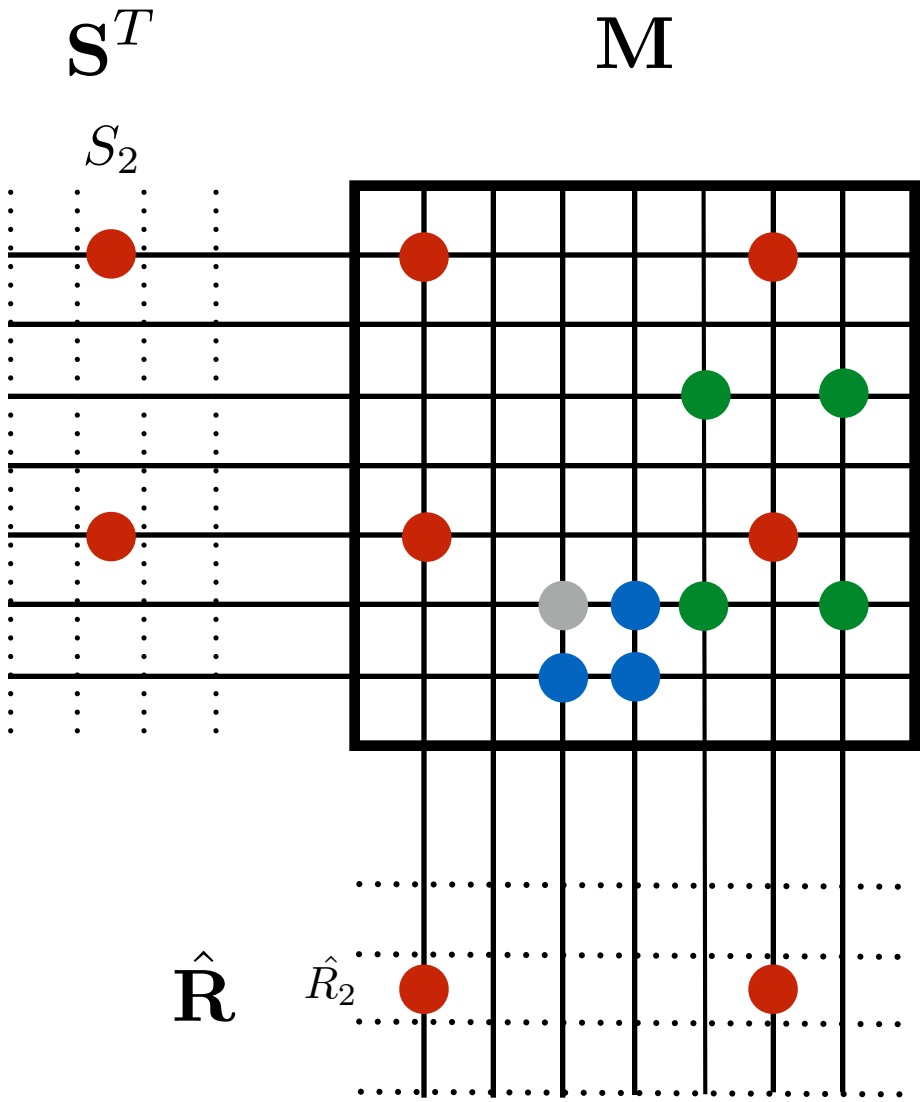


# Storage

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{R}\mathbf{S}^T$$

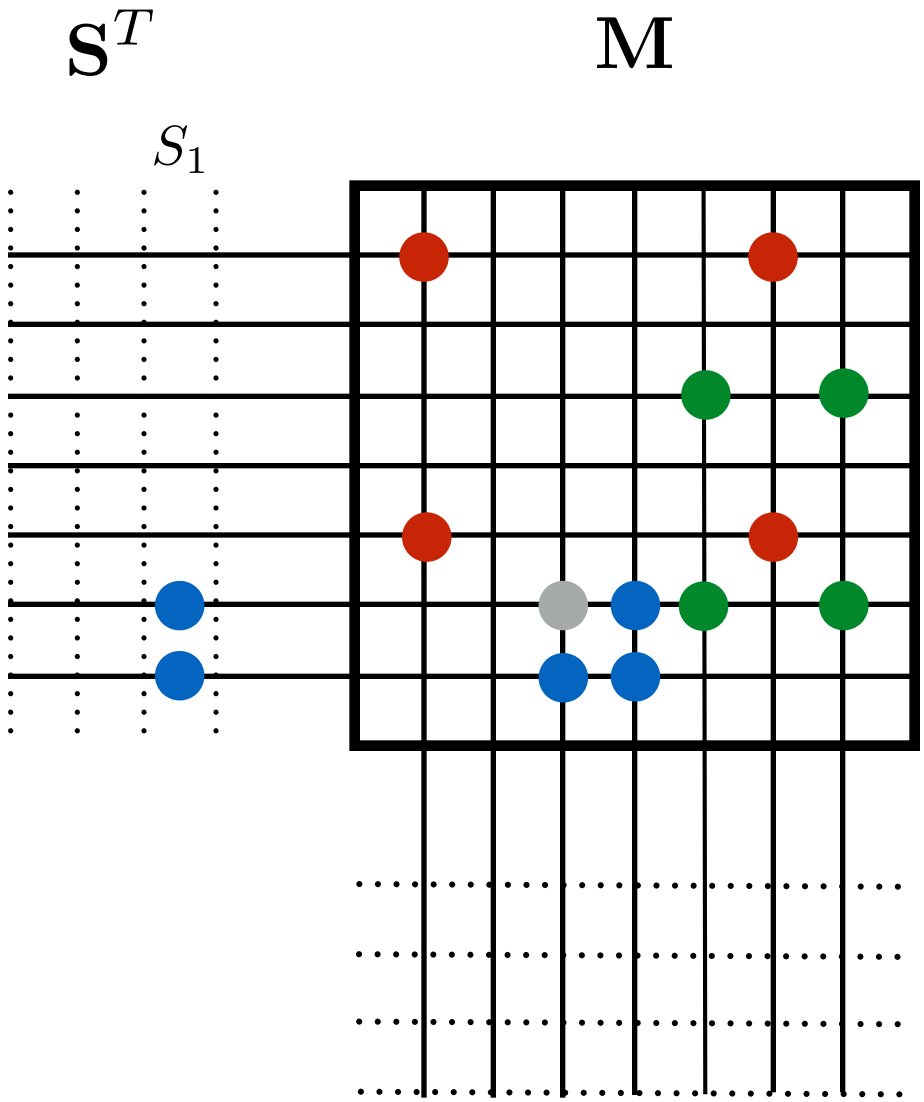


# Retrieval

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{R}\mathbf{S}^T$$

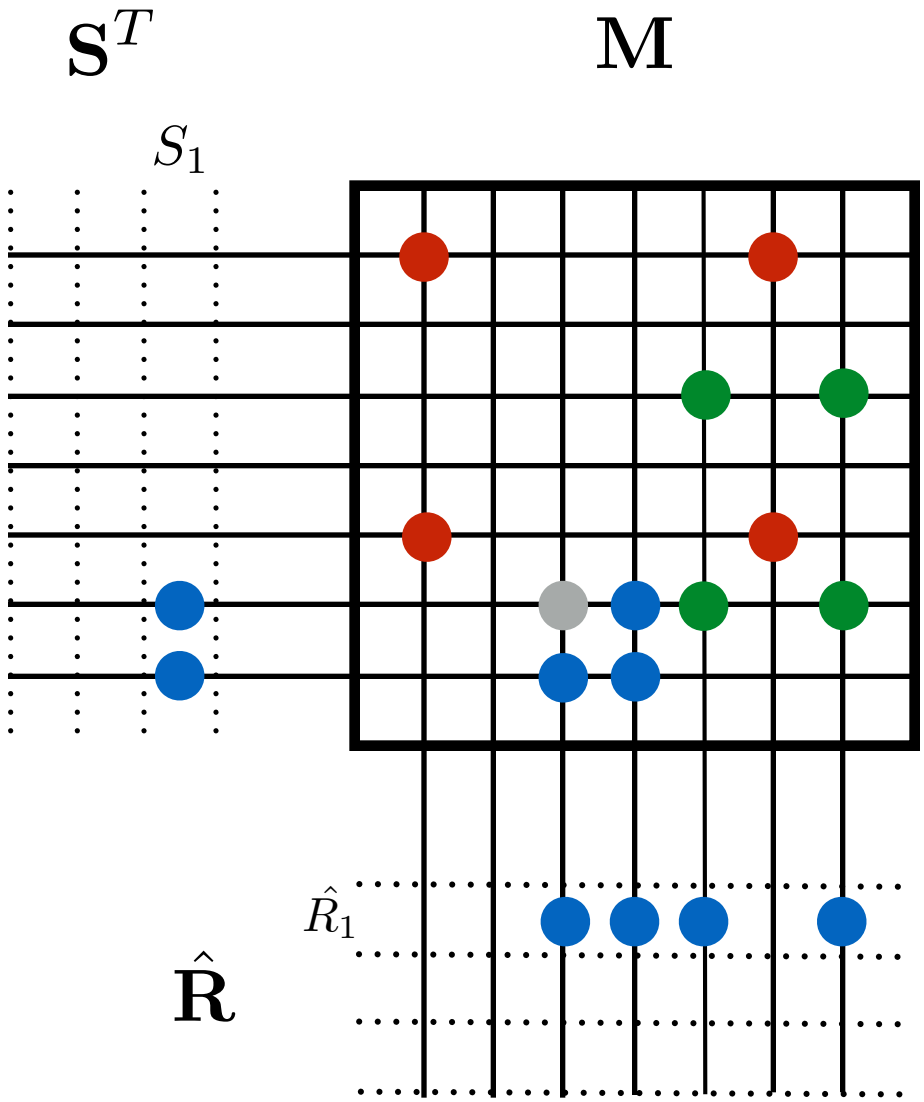


# Retrieval

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\hat{R}_j = \sum_{i=1}^{n} R_i S_i^T S_j$$



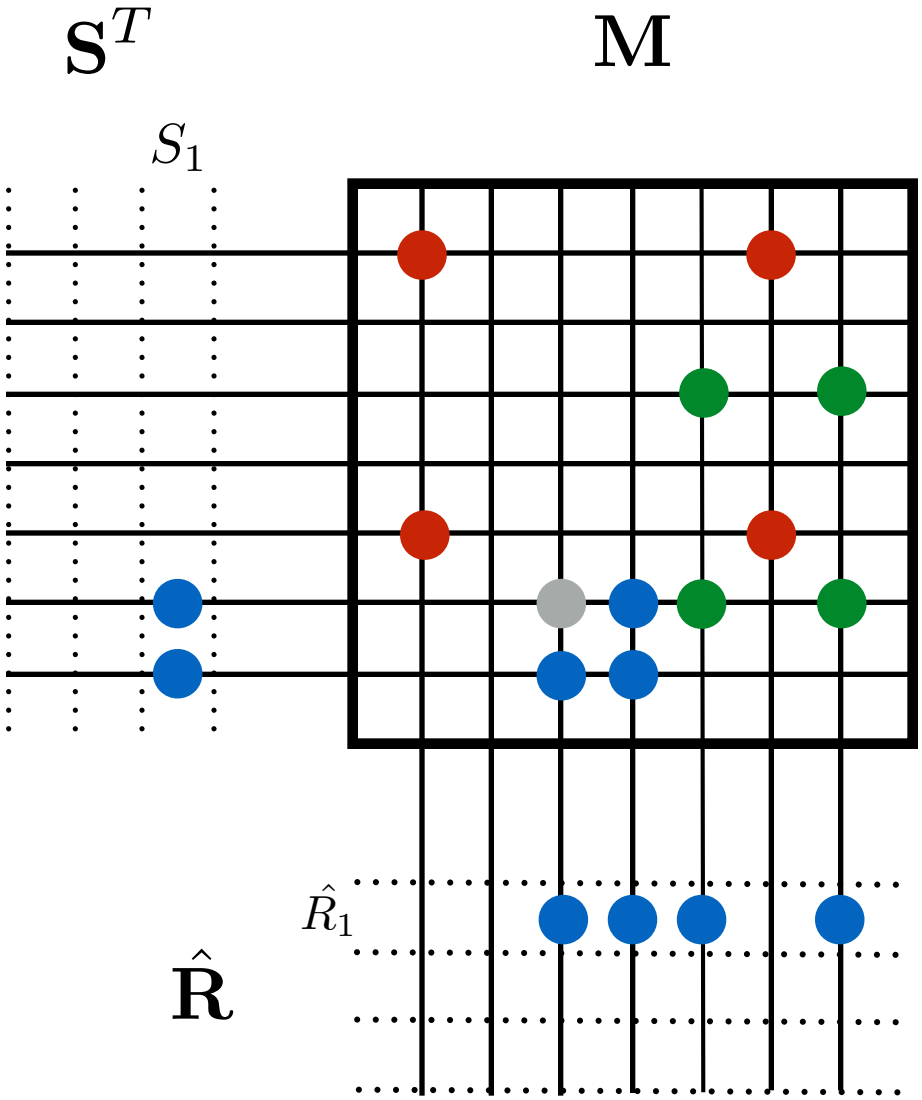
# Retrieval

Additive Hebb rule

$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\hat{R}_j = \sum_{i=1}^{n} R_i S_i^T S_j$$

$$\hat{R}_j = \sum_{i \neq j}^{n} R_i S_i^T S_j + R_j ||S_j||^2$$



# Retrieval

Additive Hebb rule

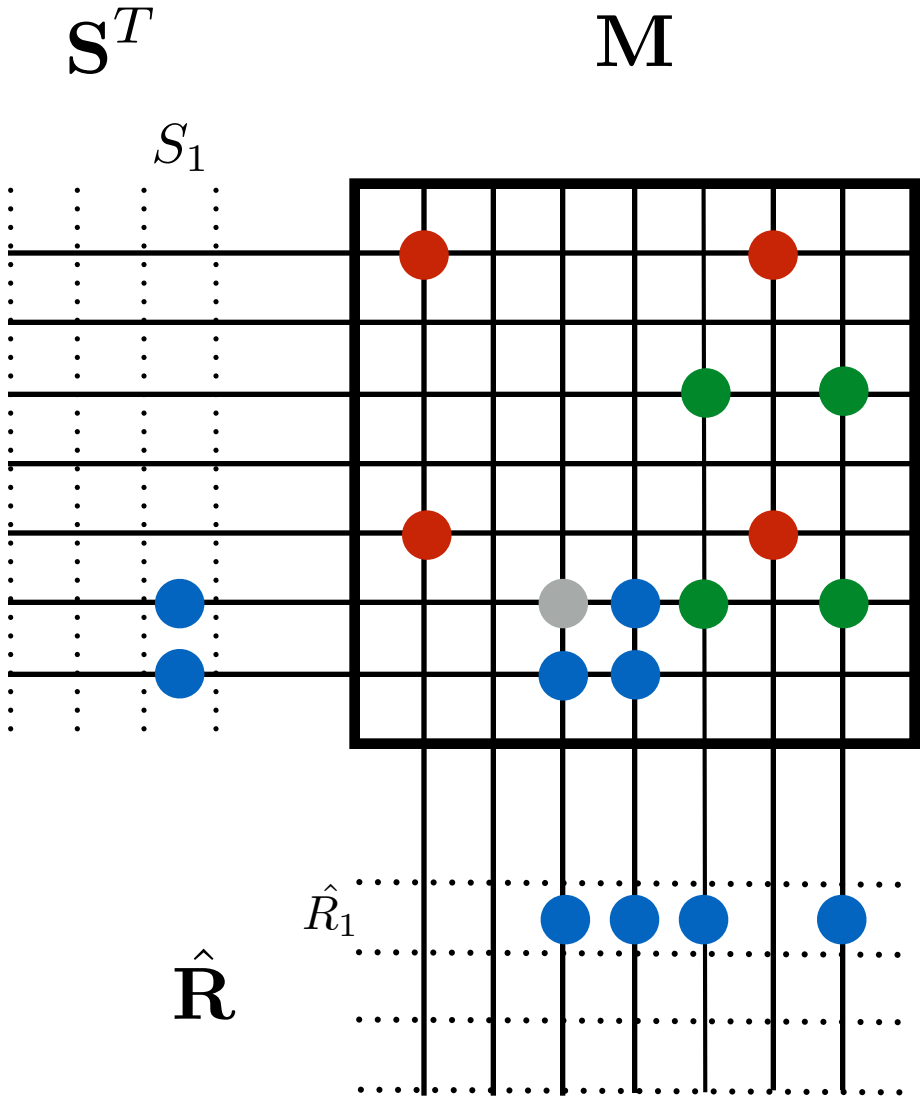
$$\mathbf{M} = \sum_{i=1}^{n} R_i S_i^T$$

$$\hat{R}_j = \sum_{i=1}^{n} R_i S_i^T S_j$$

$$\hat{R}_j = \sum_{i \neq j}^{n} R_i S_i^T S_j + R_j ||S_j||^2$$

0

1




If the $S_k$ are **orthonormal**, then retrieval is exact.

# Another perspective

Recall the the *optimal* memory matrix is

$$\mathbf{M}^\star = \mathbf{R}\mathbf{S}^\dagger$$

If the columns of $\mathbf{S}$ are linearly independent, then

$$\mathbf{X}^\dagger = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T,$$

giving

$$\mathbf{M}^\star = \mathbf{R}(\mathbf{S}^T\mathbf{S})^{-1}\mathbf{S}^T.$$

So if the columns of $\mathbf{S}$ (the $S_i$) are orthonormal,

$$\mathbf{M}^\star = \mathbf{R}\mathbf{S}^T,$$

which is exactly what we got for the simple Hebb rule.

# Capacity

How much information can a matrix memory store?

Model:

*P* patterns of size *N*

1. $|\mathbf{S}| = |\mathbf{R}| = N \times P$

2. Each input pattern (column of $\mathbf{S}$) has $m_S$ nonzeros, and each output pattern (column of $\mathbf{R}$) has $m_R$ nonzeros.

3. **Binary** Hebb rule: $\mathbf{M} = \max(\mathbf{R}\mathbf{S}^T, \mathbf{1})$ (Each entry is clipped at one.)

4. Threshold recall:

$$\hat{R}_{jk} = \begin{cases} 1 & [\mathbf{M}S_j]_k > \tau \\ 0 & \text{else} \end{cases}$$

# Capacity

How much information can a matrix memory store?

Model:

*P* patterns of size *N*

1. $|\mathbf{S}| = |\mathbf{R}| = N \times P$

2. Each input pattern (column of $\mathbf{S}$) has $m_S$ nonzeros, and each output pattern (column of $\mathbf{R}$) has $m_R$ nonzeros.

3. $\mathbf{I}$ $T$, $\mathbf{1}$)

Parameters we can pick

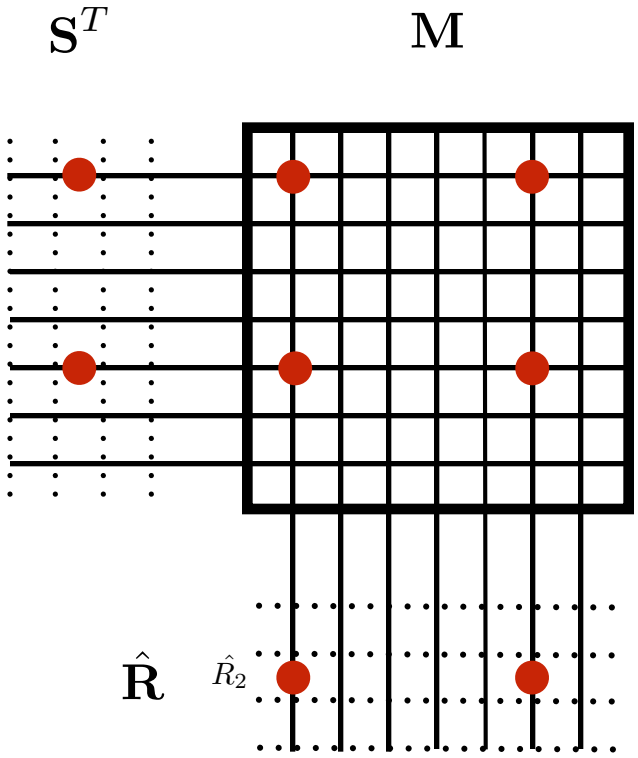
4. [illegible]

$$\hat{R}_{jk} = \begin{cases} 1 & [\mathbf{M}S_j]_k > \tau \\ 0 & \text{else} \end{cases}$$

# Capacity

To choose the threshold $\tau$, note that in the absence of noise, a column of $\mathbf{M}$ has exactly $m_S$ ones.

So in order to recover all the ones in $\mathbf{R}$, we better set $\tau = m_S$.



# Capacity

## Sparsity parameters

The chance of given weight in $\mathbf{M}$ remaining zero throughout the learning process is

$$(1 - \frac{m_S m_R}{N^2})^P \approx e^{\frac{-P m_S m_R}{N^2}} = (1 - q)$$

The probability of a spurious one in $\hat{\mathbf{R}}$ is the probability of exceeding $\tau$ purely by chance. This is $Nq^{m_S}$. So the highest value for $m_S$ we can choose before make the first error is:

$$Nq^{m_S} = 1 \quad \Rightarrow \quad m_S = -\frac{\log N}{\log q}$$

# Capacity

Sparsity parameters

From previous slide

$$e^{\frac{-Pm_S m_R}{N^2}} = (1-q) \qquad m_S = -\frac{\log N}{\log q}$$

There are $\binom{N}{m_R}$ possible output patterns, meaning that the information in each one is $\log\binom{N}{m_R}$, making in total:

$$I = P\binom{N}{m_R} \approx \underline{Pm_R}\underline{\log N} \text{ nats}$$

$$-\frac{N^2\log(1-q)}{m_S} \qquad -m_S \log q$$

$$I = N^2 \log(q)\log(1-q)$$

# Capacity

## Sparsity parameters

$$I = N^2 \log(q)\log(1-q)$$

The most information that could be stored in $\mathbf{M}$ is $N^2$ bits. The factor $\log(q)\log(1-q)$ reaches it maximum value of 0.69 when $q = 0.5$.

This is quite good!

In order to achieve this value, we need *sparse* patterns, and the more there, the sparser they must be.

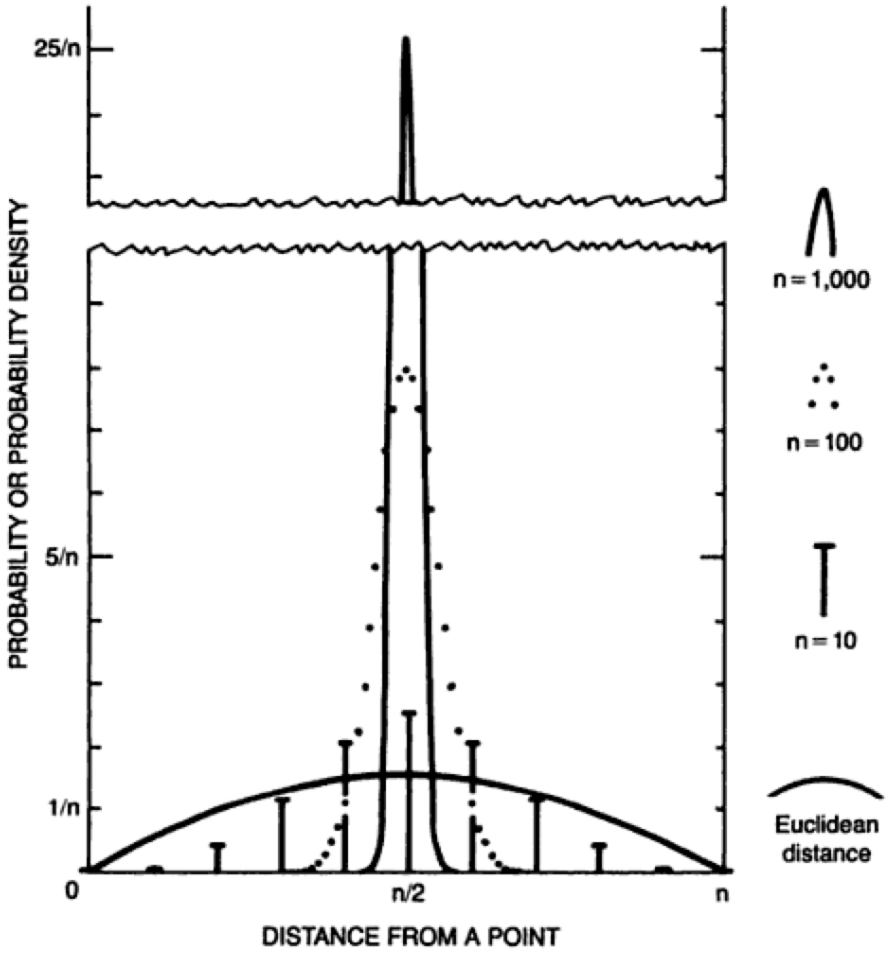
Makes sense w.r.t. earlier analysis.

$$(1-q) = e^{\frac{-P m_S m_R}{N^2}} \quad \Rightarrow \quad \frac{m_S m_R}{N^2} = \frac{\log(2)}{N^2}$$

# Memory space

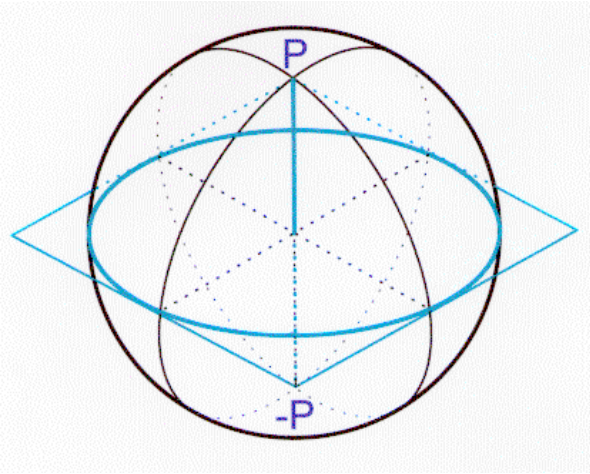
Kanerva, 1988

The decision to represent memory items as sparse, high-dimensional vectors has some interesting consequences. *High-dimensional spaces are counterintuitive.*




In 1000 dimensions, 0.001 of all patterns are within 451 bits of a given point, and all **but** 0.001 are within 549 bits.

Points tend to be orthogonal - most point-pairs are “noise-like.”

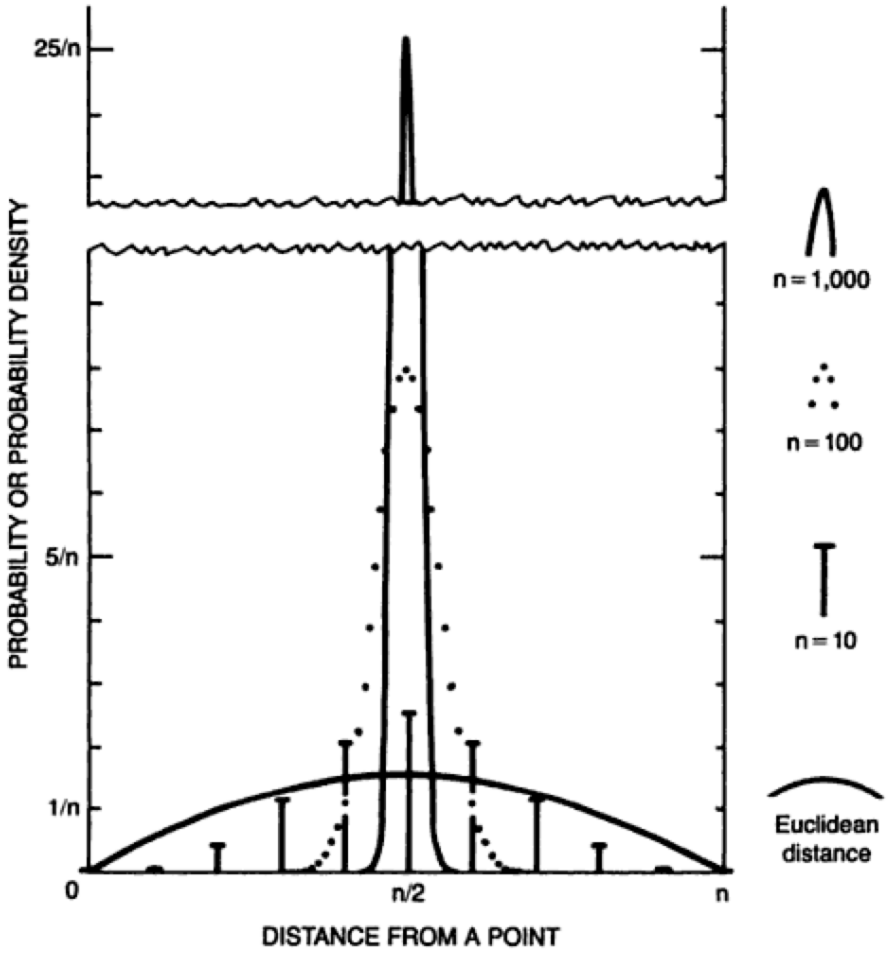


# Memory space

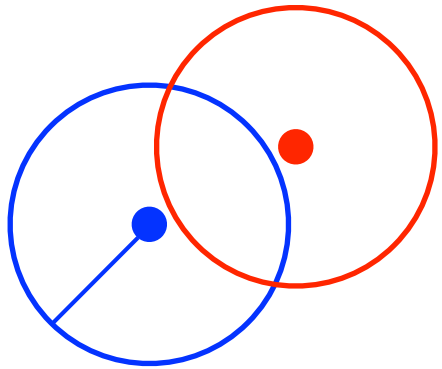
Kanerva, 1988

The decision to represent memory items as sparse, high-dimensional vectors has some interesting consequences.

*High-dimensional spaces are counterintuitive.*




## Linking concepts



Almost all pairs of points are far apart, but there are multiple“linking” points that are close to both.

# Memory space

Kanerva, 1988

The decision to represent memory items as sparse, high-dimensional vectors has some interesting consequences.

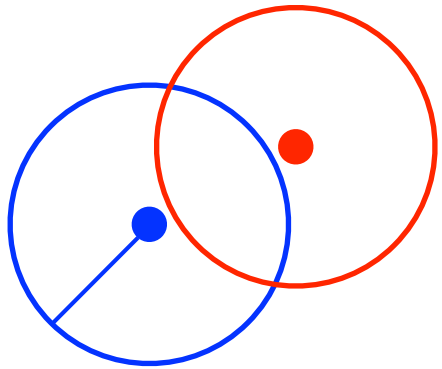
*High-dimensional spaces are counterintuitive.*

Why are fire engines painted red?
Firemen's suspenders are red, too.
Two and two are four.
Four times three is twelve.

Introduction

Twelve inches in a foot.
A foot is a ruler.
Queen Mary was a ruler.
Queen Mary sailed the sea.
The sea has sharks.
Sharks have fins.
The Russians conquered the Finns.
The Russians' color is red.
Fire engines are always rushin'.
So that's why they're painted red!

## Linking concepts



Almost all pairs of points are far apart, but there are multiple"linking" points that are close to both.

# Matrix memories in the brain: Marr's model of the cerebellum

Marr, 1967
Albus, 1971

*J. Physiol.* (1969), **202**, *pp.* 437–470
*With* 1 *plate and* 2 *text-figures*
*Printed in Great Britain*



## A THEORY OF CEREBELLAR CORTEX

BY DAVID MARR*
*From Trinity College, Cambridge*

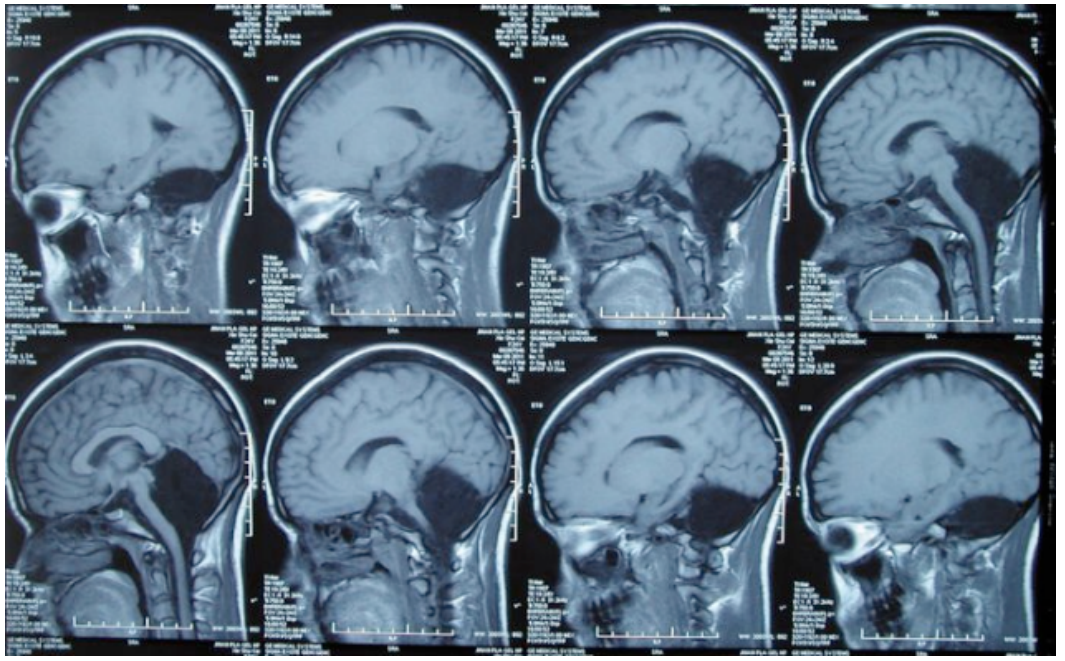
(*Received* 2 *December* 1968)

**Short, live experiment**

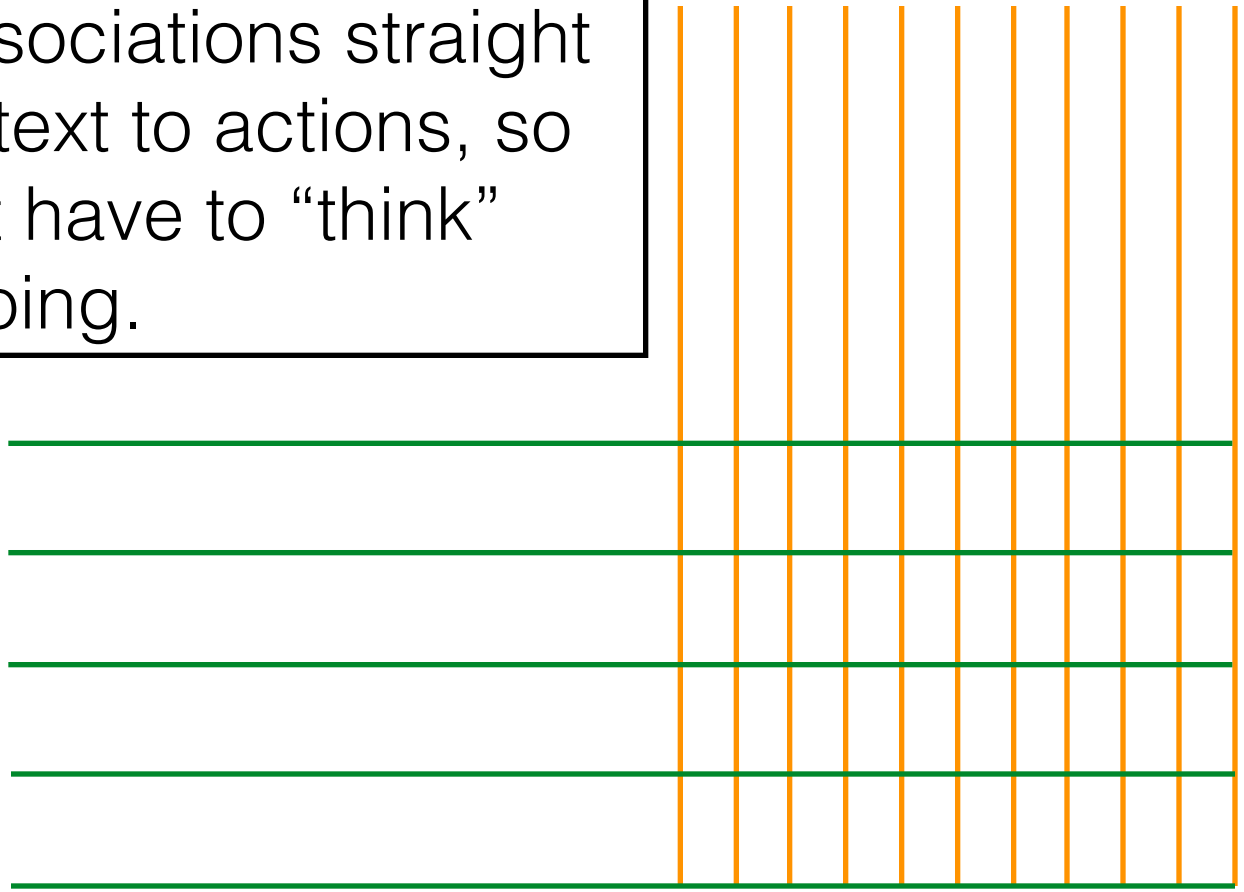
# Matrix memories in the brain: Marr's model of the cerebellum

The cerebellum produces *smooth*, *coordinated* motor movements.
(And may be involved in cognition as well).

**24-year old Chinese woman without a cerebellum**



Learn associations straight from context to actions, so you don't have to "think" before doing.

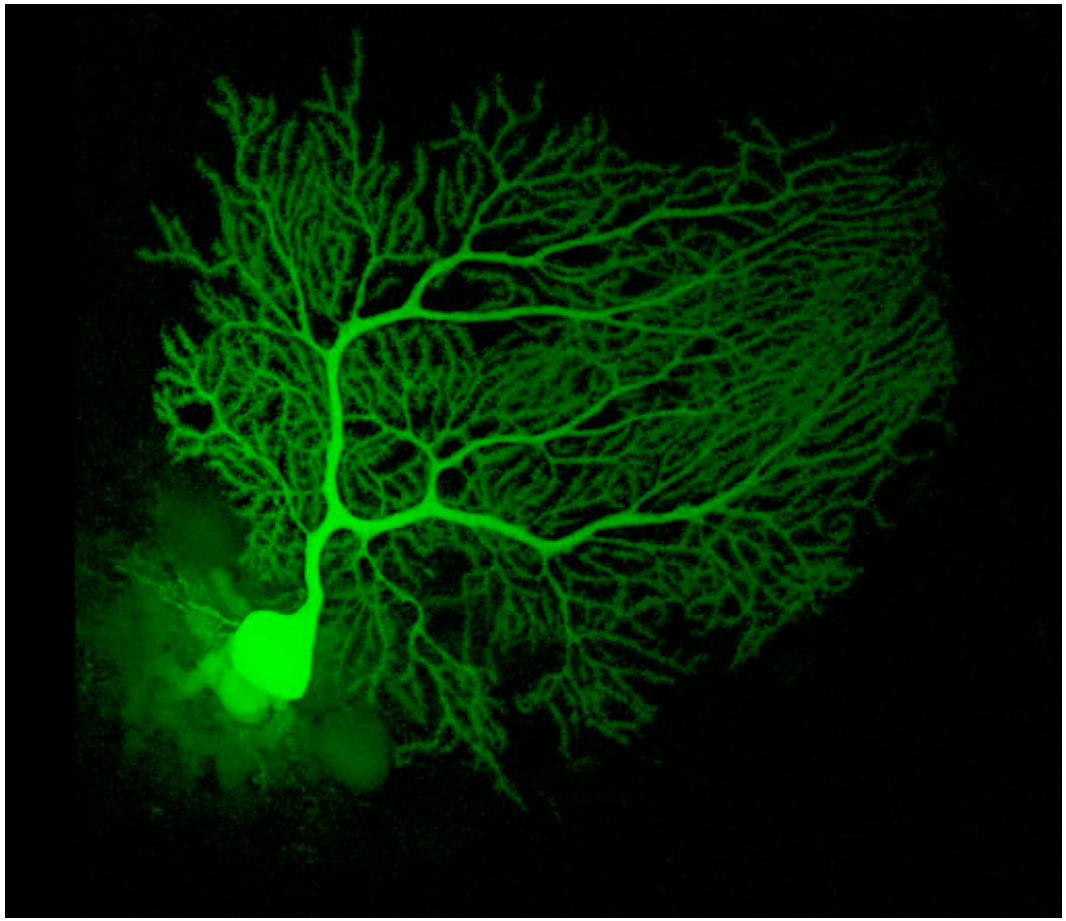


Purkinje Axons
Motor Output

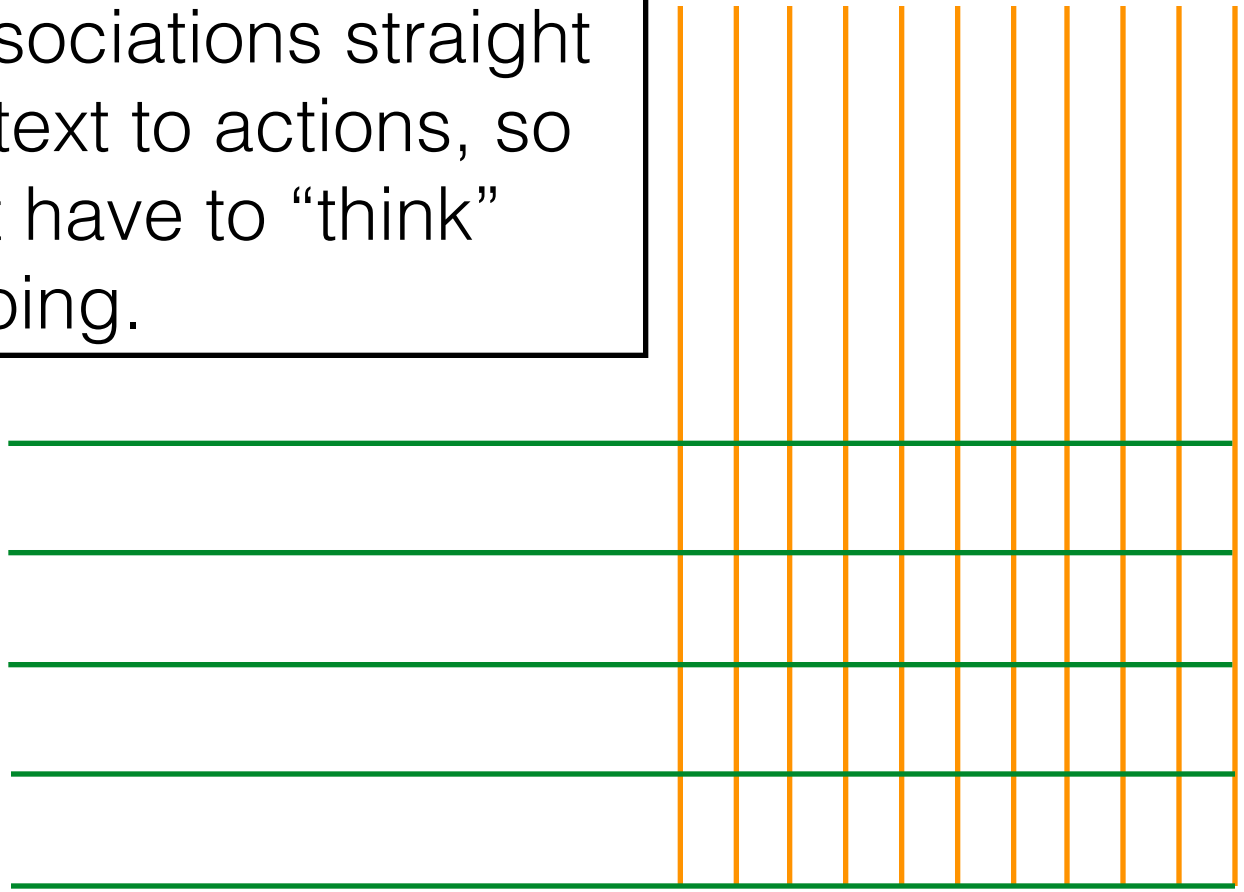
Mossy Fiber
Contextual Input

Learn associations straight from context to actions, so you don't have to "think"

Purkinje Axons
Motor Output



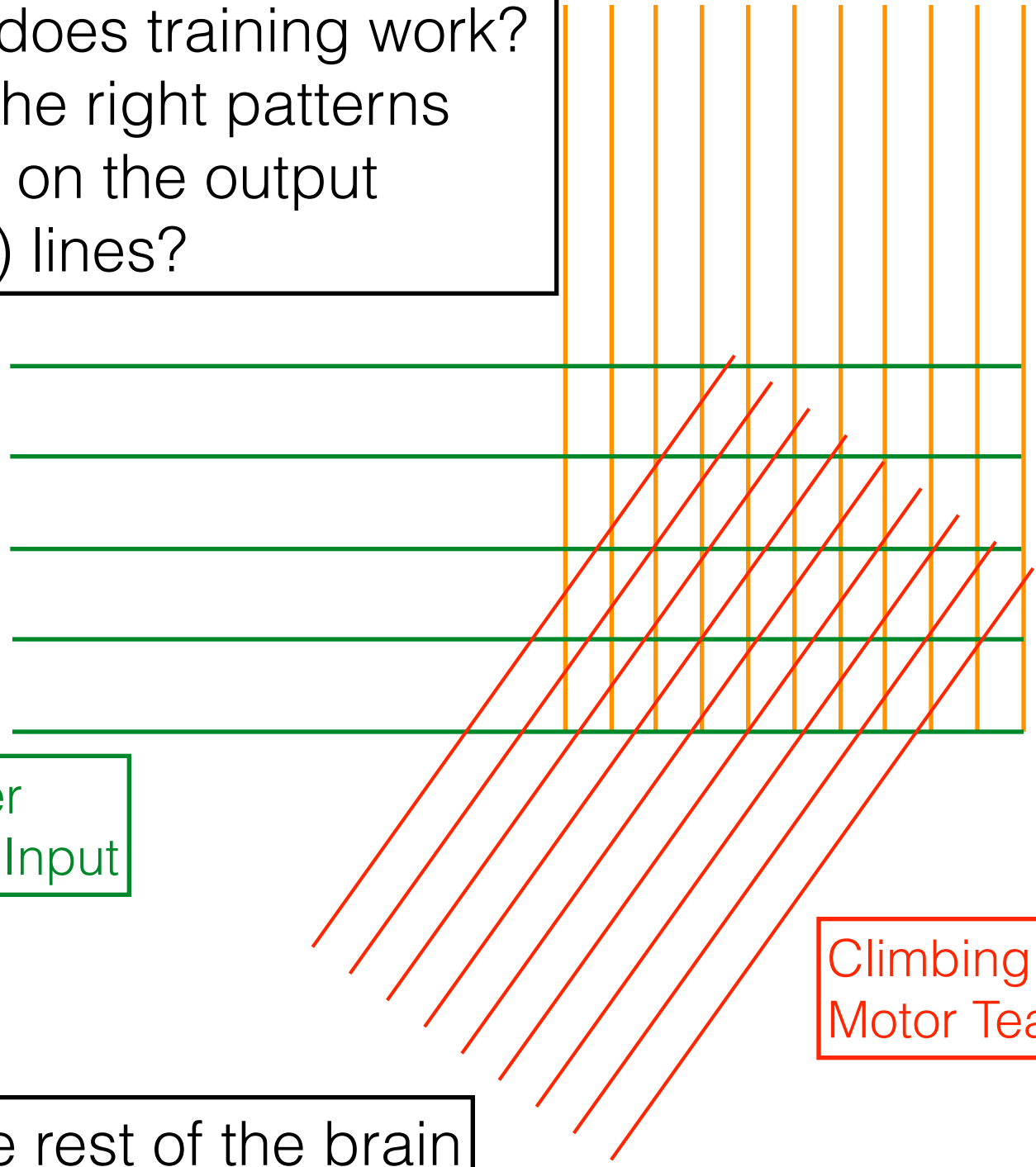
Learn associations straight from context to actions, so you don't have to "think" before doing.



Purkinje Axons
Motor Output

Mossy Fiber
Contextual Input

But how does training work? How do the right patterns "appear" on the output (Purkinje) lines?



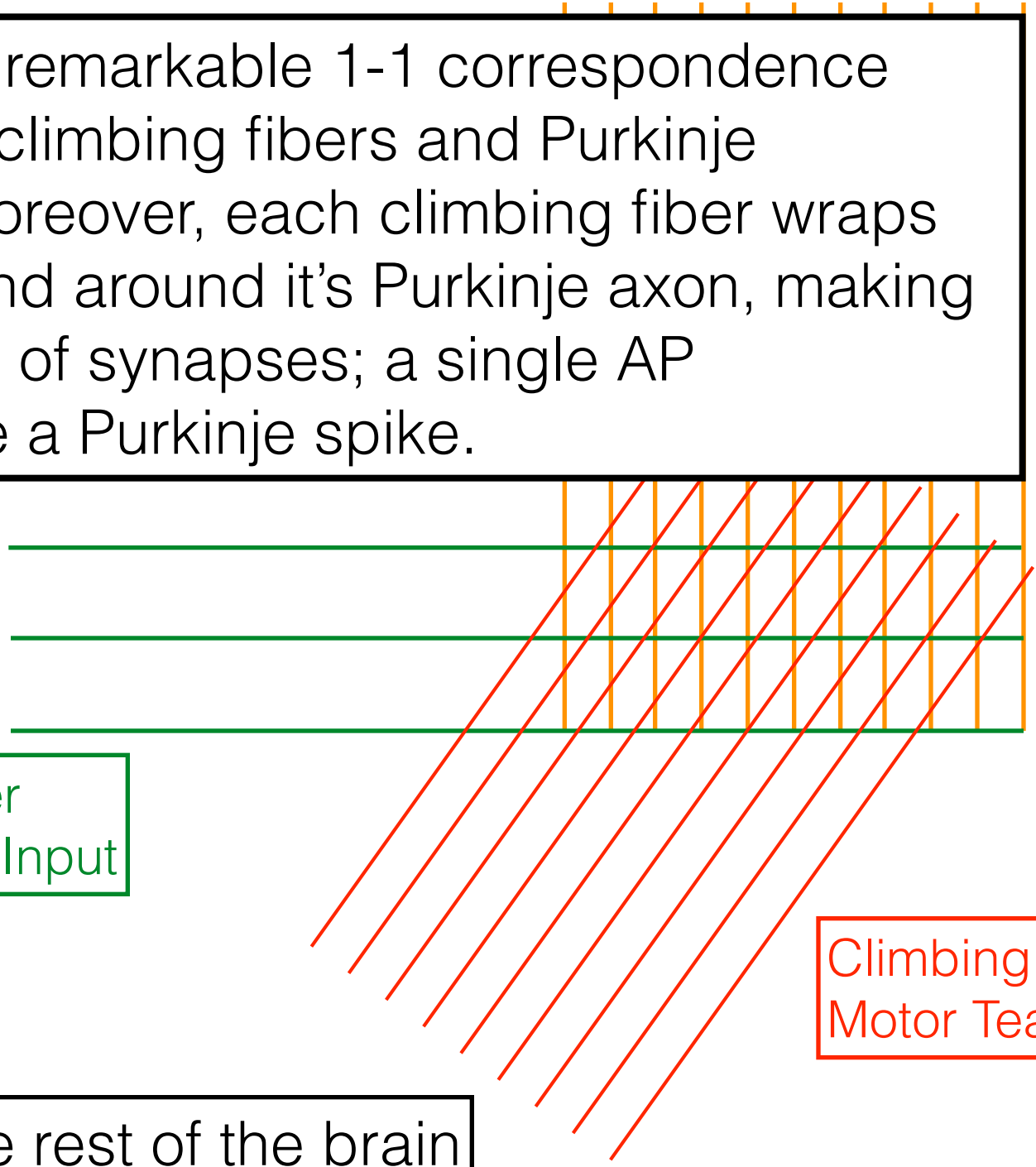
Purkinje Axons
Motor Output

Mossy Fiber
Contextual Input

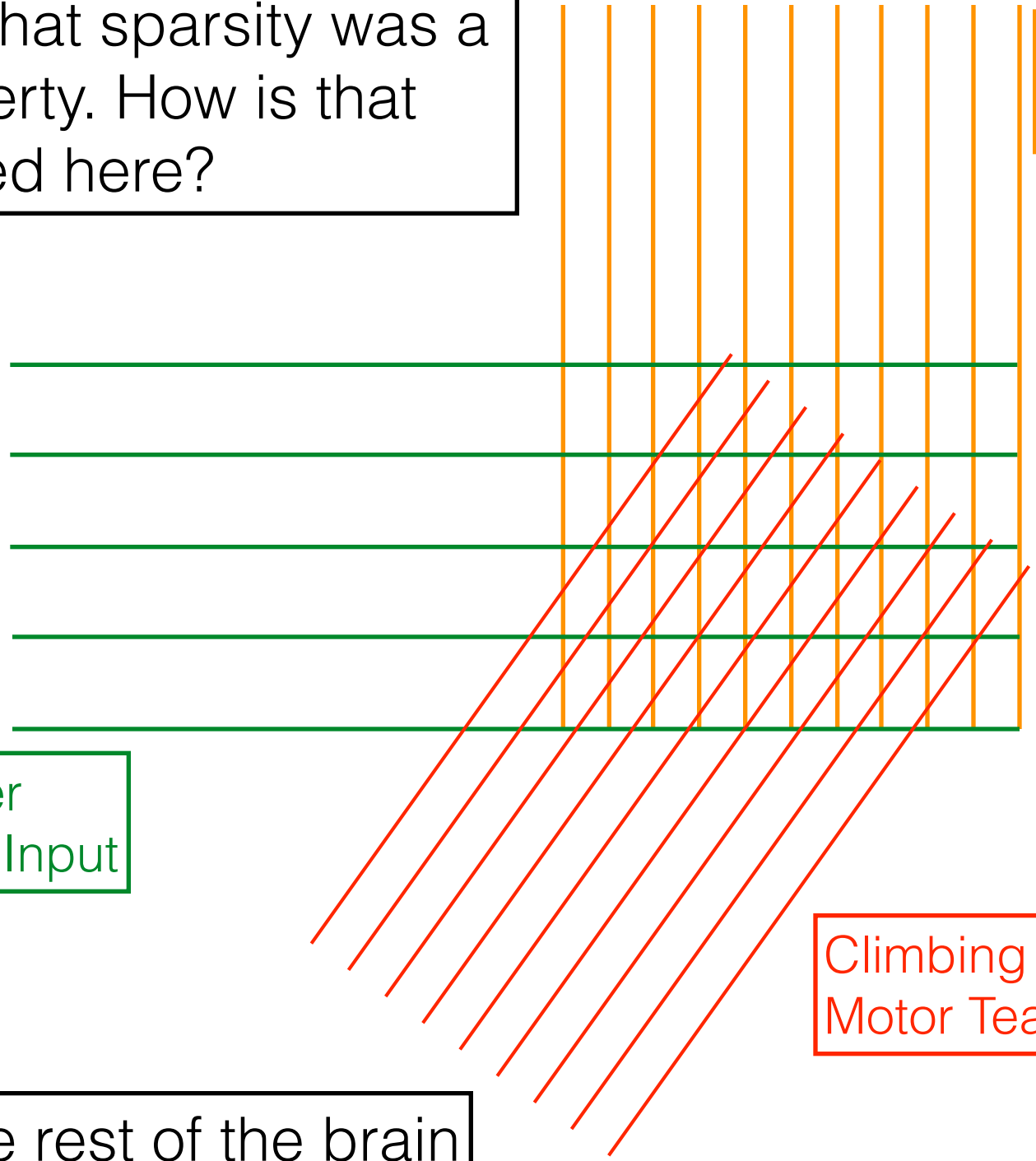
Climbing Fibers
Motor Teaching Input

The rest of the brain

There's a remarkable 1-1 correspondence between climbing fibers and Purkinje axons. Moreover, each climbing fiber wraps around and around it's Purkinje axon, making hundreds of synapses; a single AP can make a Purkinje spike.



We said that sparsity was a key property. How is that manifested here?

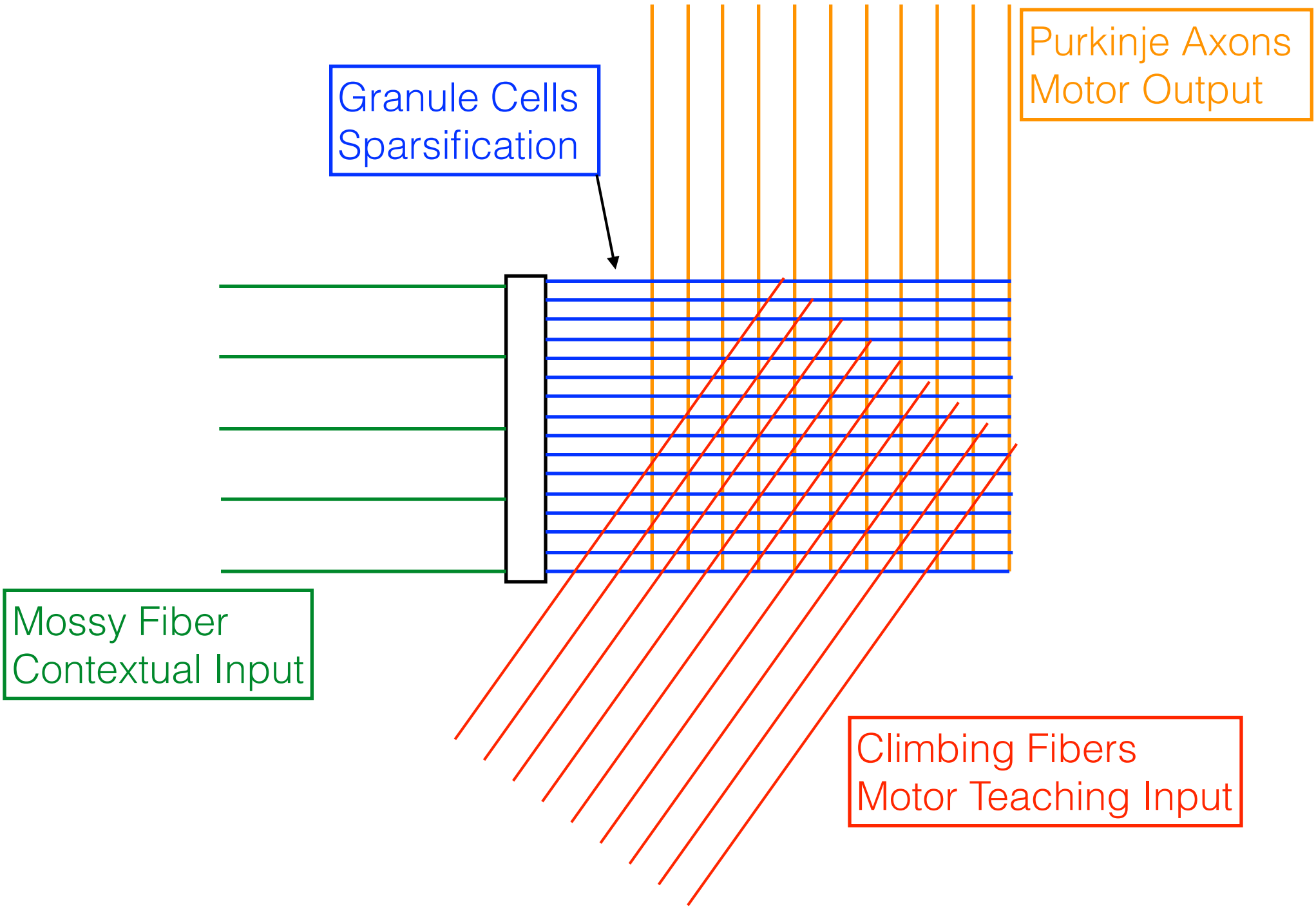


Purkinje Axons
Motor Output

Mossy Fiber
Contextual Input

Climbing Fibers
Motor Teaching Input

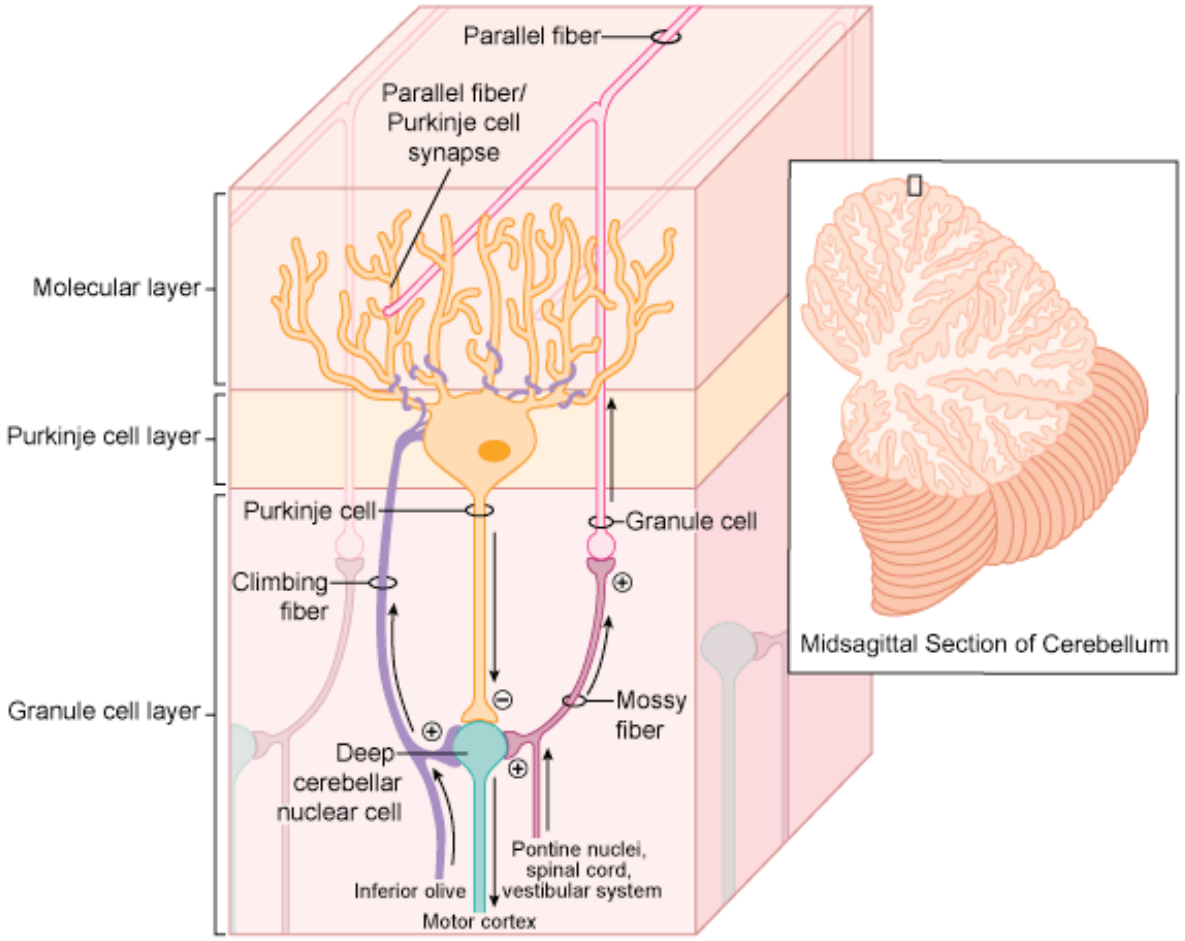
The rest of the brain

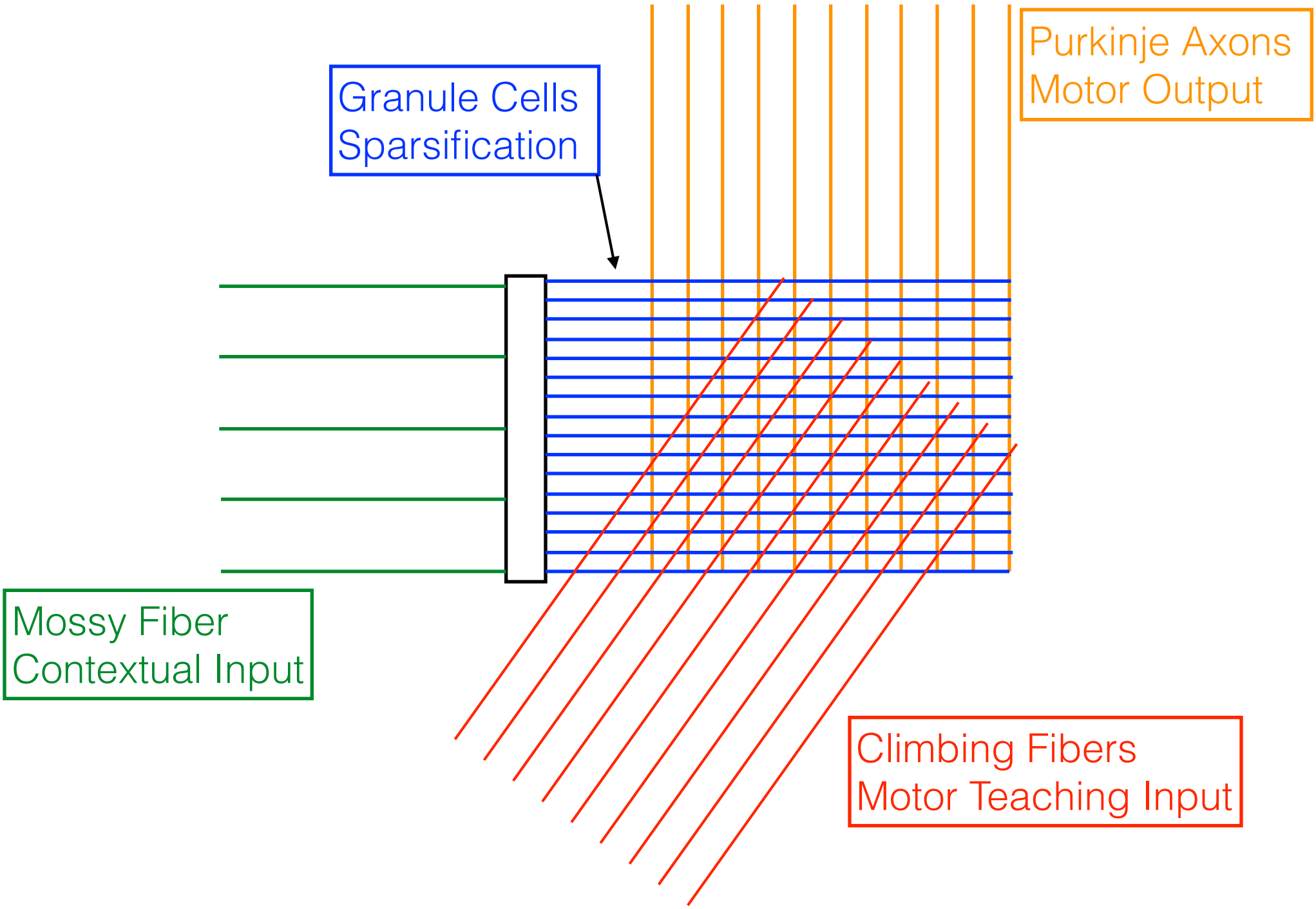

Purkinje Axons
Motor Output
Granule Cells
Sparsification
Mossy Fiber
Contextual Input
Climbing Fibers
Motor Teaching Input

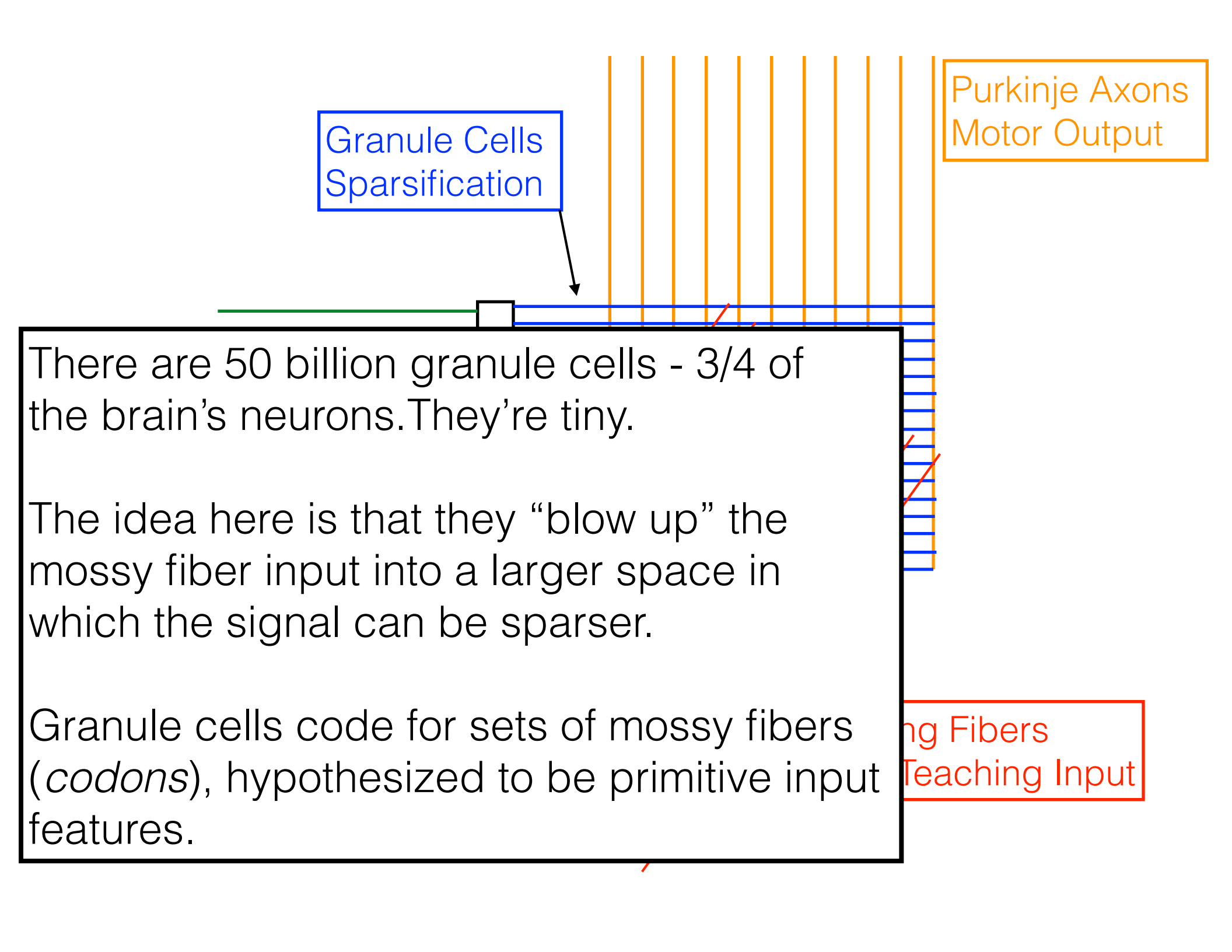
Granule Cells

Purkinje Axons
Motor Output

Mossy Fiber
Contextual In




Purkinje Axons
Motor Output
Granule Cells
Sparsification
Mossy Fiber
Contextual Input
Climbing Fibers
Motor Teaching Input

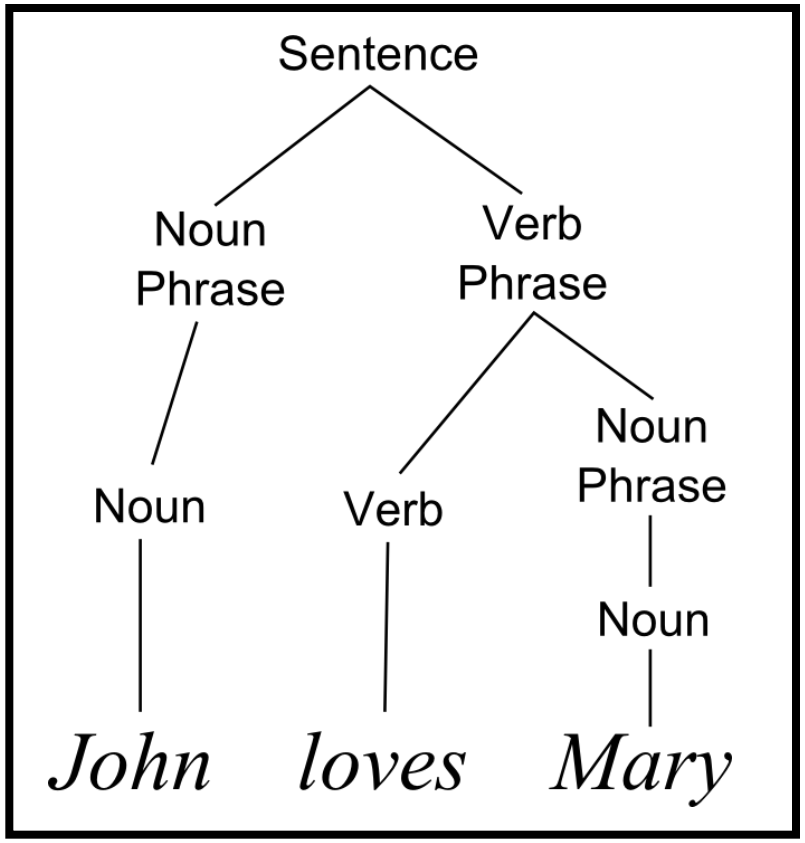

Granule Cells
Sparsification
Purkinje Axons
Motor Output
There are 50 billion granule cells - 3/4 of the brain's neurons.They're tiny.
The idea here is that they "blow up" the mossy fiber input into a larger space in which the signal can be sparser.
Granule cells code for sets of mossy fibers (*codons*), hypothesized to be primitive input features.
ng Fibers
Teaching Input

# Storing structured information

We've discussed how to store S-R pairs, but human cognition goes way beyond this.

**Relations**

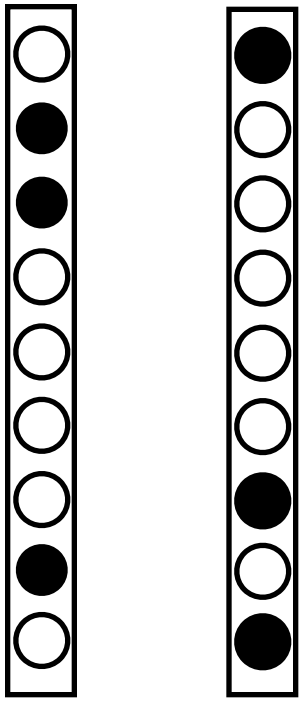
- The kettle is on the table.
- The kettle is to the right of the mug.




# Storing structured information

As before, “concepts” are activation vectors.

Jar Kettle

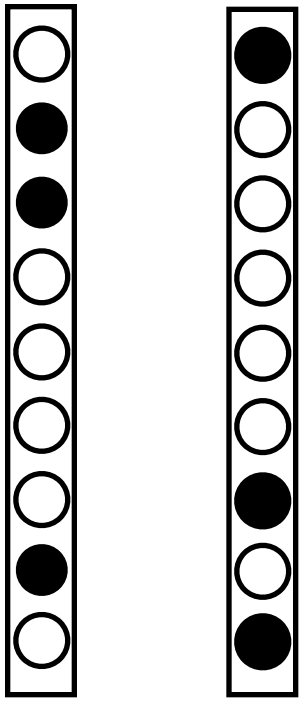


# Storing structured information

As before, “concepts” are activation vectors.

How to represent this?
Green(Jar) & Gray(Kettle)

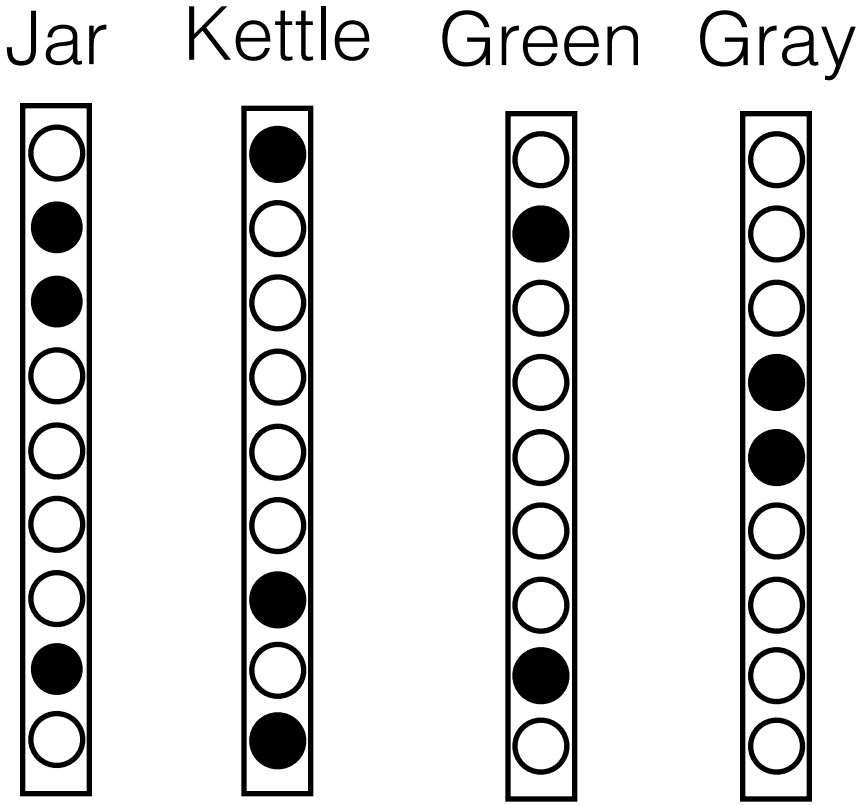
Jar Kettle



# Storing structured information

As before, “concepts” are activation vectors.

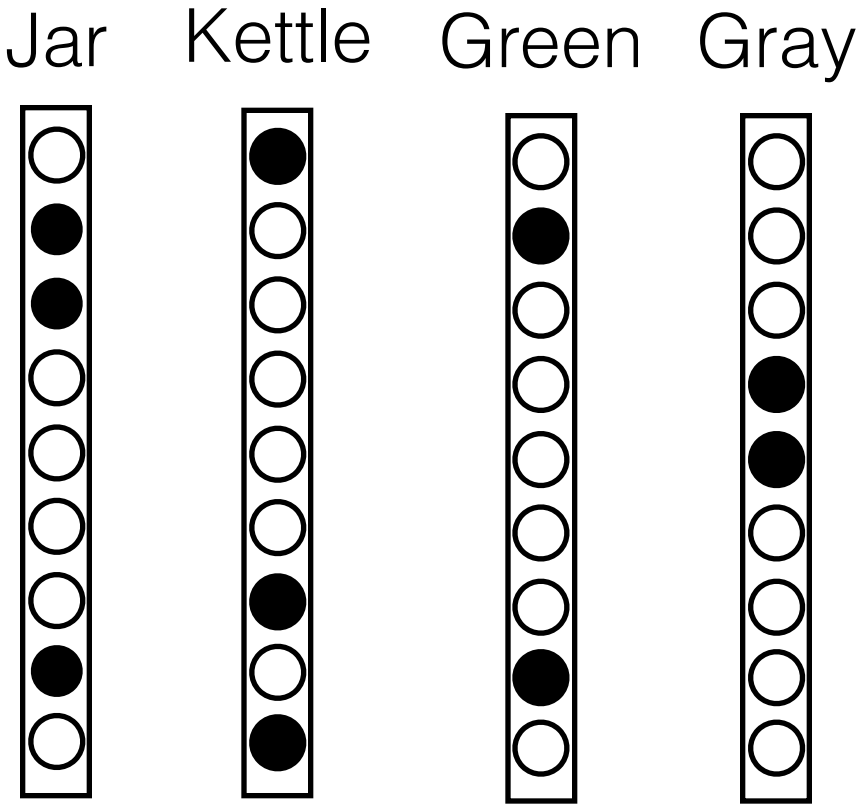
How to represent this?
Green(Jar) & Gray(Kettle)




# Storing structured information

As before, “concepts” are activation vectors.

How to represent this?
Green(Jar) & Gray(Kettle)




**Now what?**

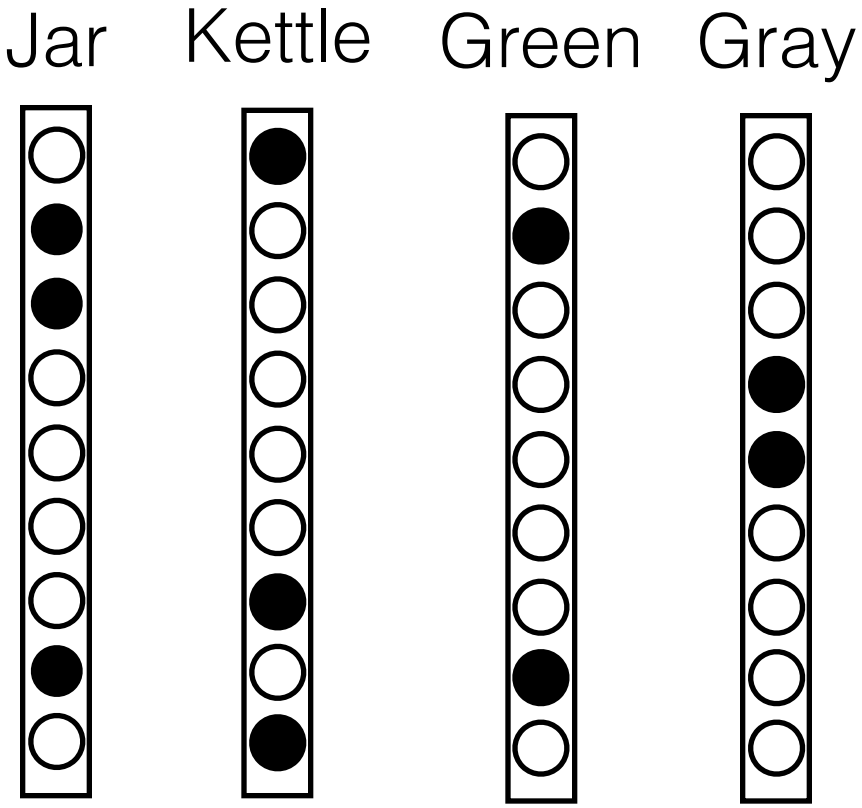
Maybe we should just have
all of these patterns fire at once?

But then how do we know we don’t have
Gray(Jar) & Green(Kettle) ?
Or, worse,
Jar(Kettle) & Green & Gray?

# Storing structured information

As before, “concepts” are activation vectors.

How to represent this?
Green(Jar) & Gray(Kettle)




**Now what?**

Maybe we should just have all of these patterns fire at once?

But then how do we know we don’t have

Gray(Jar) & Green(Kettle) ?

Or, worse,

Jar(Kettle) & Green & Gray?

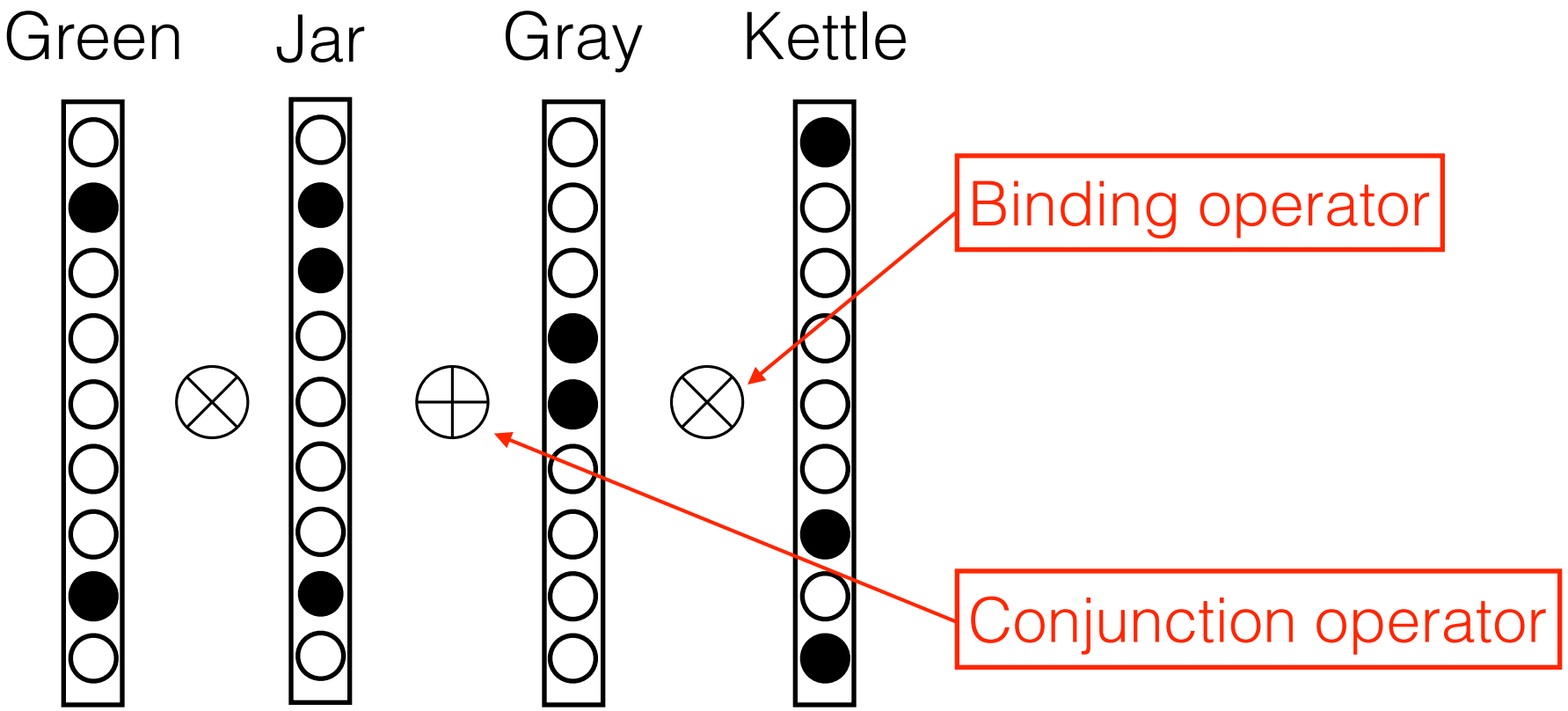
We need a way to *bind* predicates to arguments.

# Storing structured information

As before, “concepts” are activation vectors.

How to represent this?
Green(Jar) & Gray(Kettle)

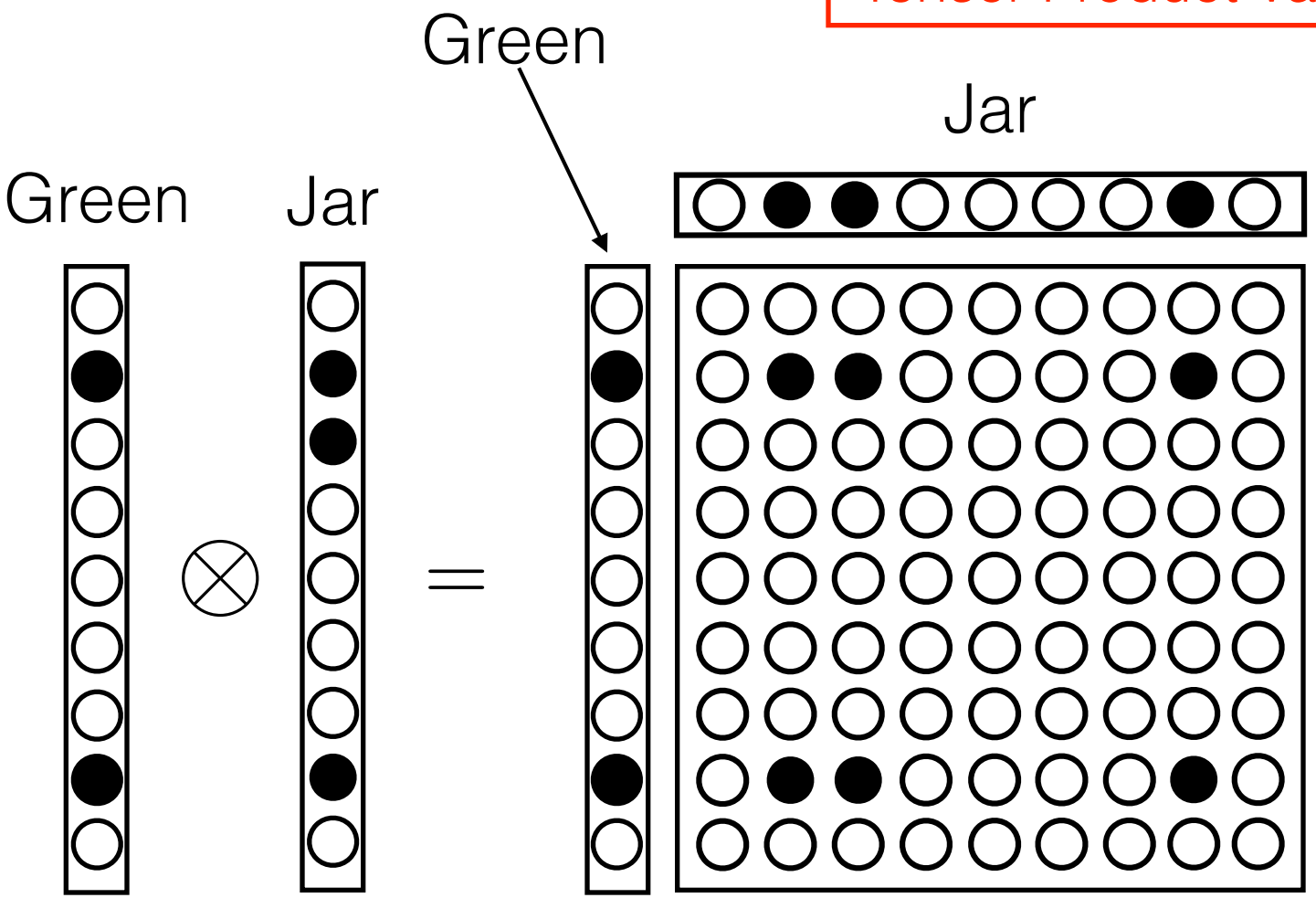
We need a way to *bind* predicates to arguments.



# Storing structured information

How should we choose $\otimes$ and $\oplus$?

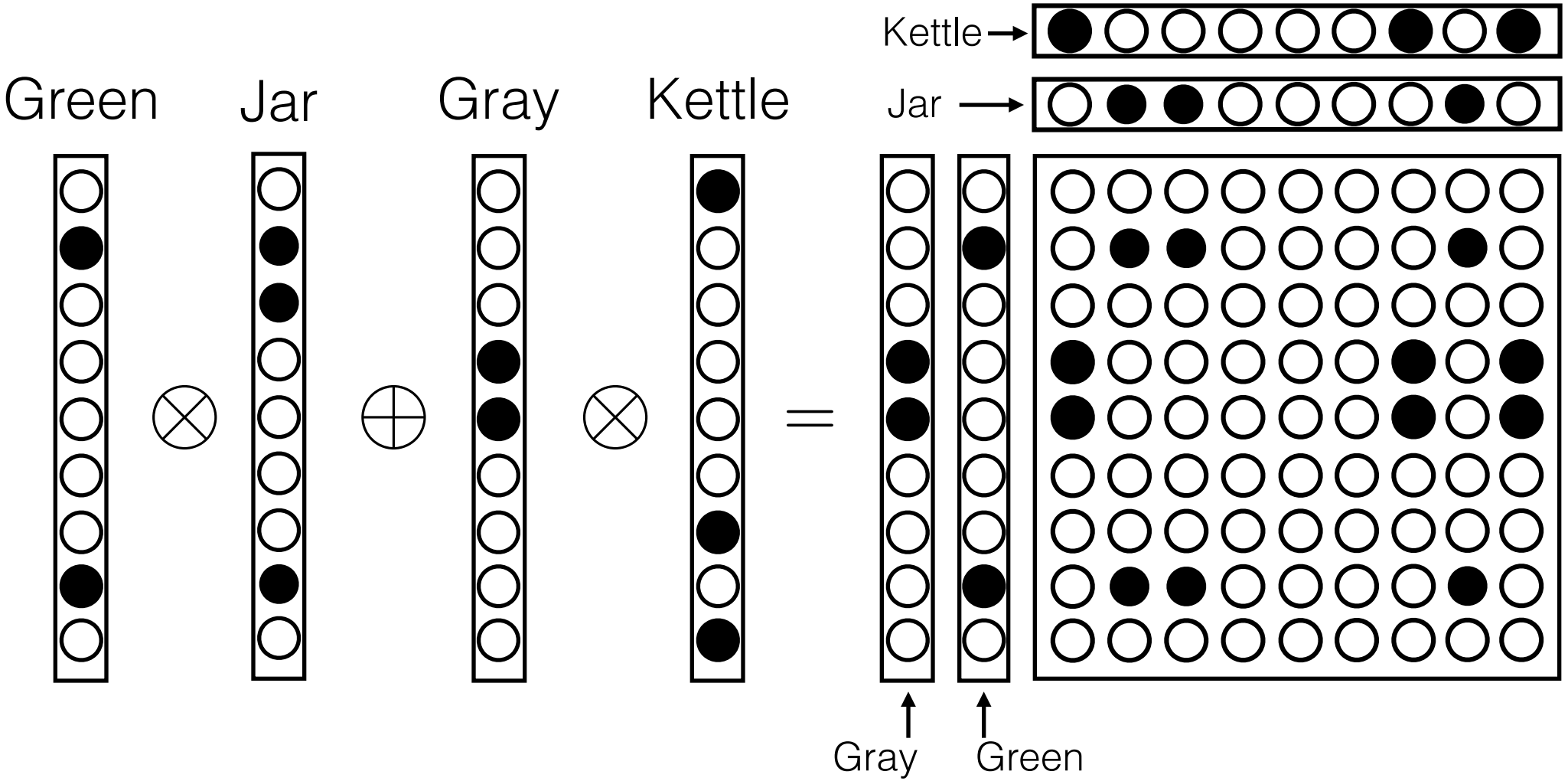
Paul Smolensky,
"Tensor Product Variable Binding", 1990



# Storing structured information

How should we choose $\otimes$ and $\oplus$?

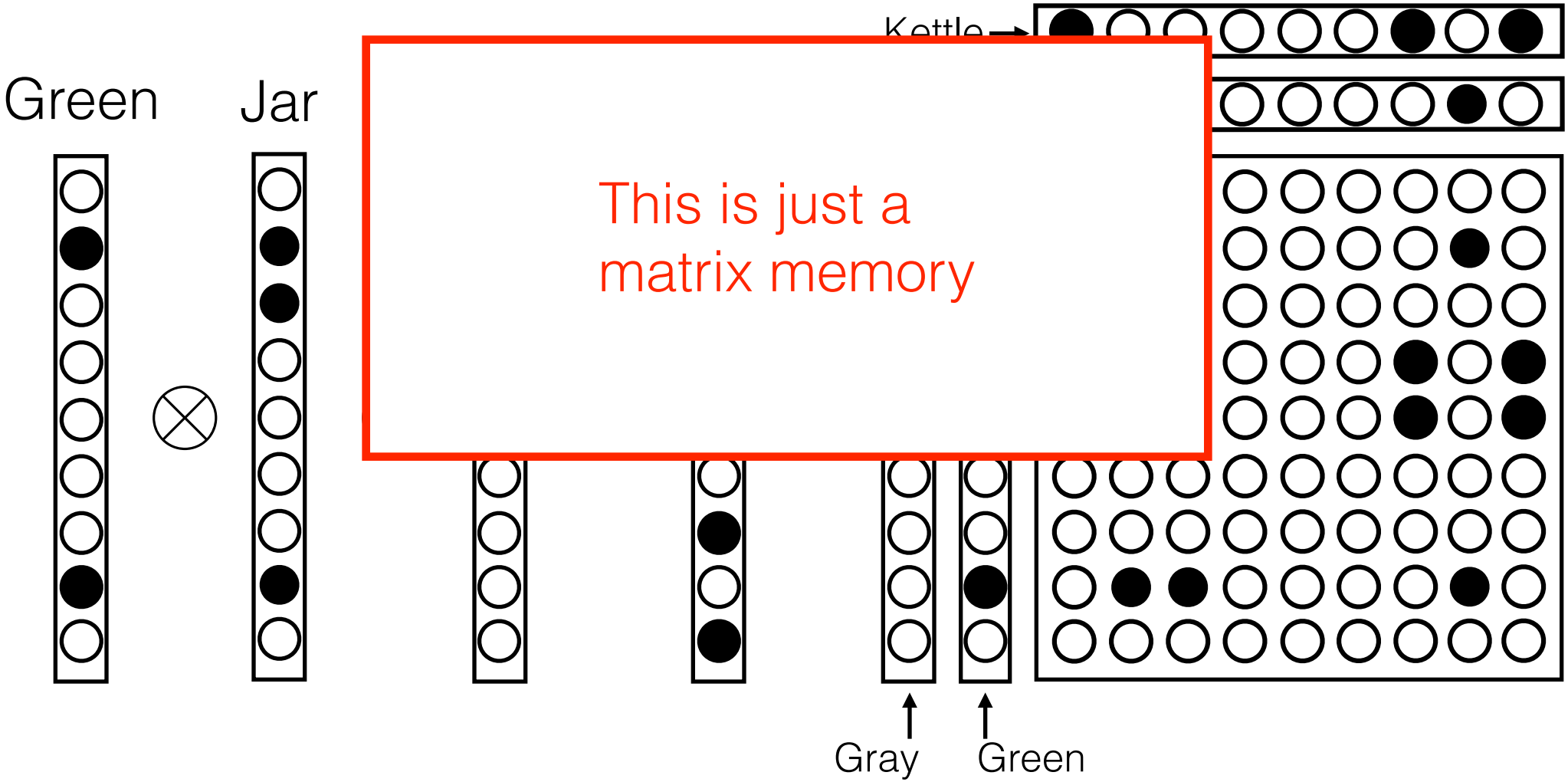
Paul Smolensky,
"Tensor Product Variable Binding", 1990



# Storing structured information

How should we choose $\otimes$ and $\oplus$?

Paul Smolensky,
"Tensor Product Variable Binding", 1990



# Storing structured information

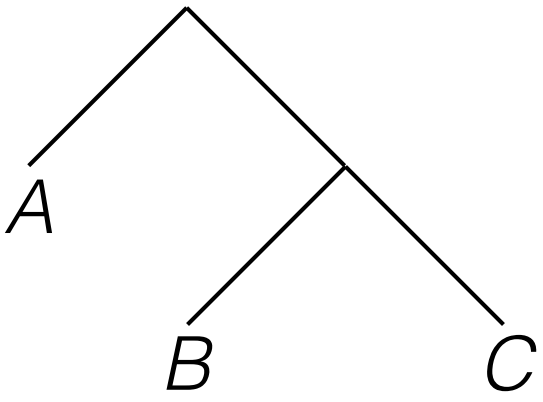
We can store some interesting "data structures," like a stack:

$$S = \sum_i f_i \otimes r_i$$

Some alphabet "fillers"

Linearly independent "indexing roles"

Or a tree:



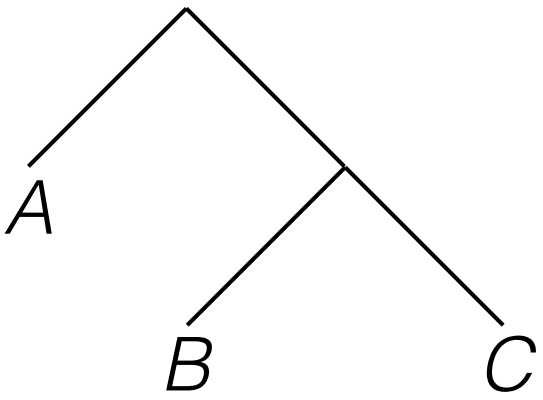

$$T = A \otimes r_0 \otimes +[B \otimes r_0 + C \otimes r_1] \otimes r_1$$

$$= A \otimes r_0 \otimes +B \otimes r_0 \otimes r_1 + C \otimes r_1 \otimes r_1$$

$$= A \otimes r_0 \otimes +B \otimes r_{10} + C \otimes r_{11}$$

Note the dimensions wrt these sums

# Storing structured information




$$T = A \otimes r_0 \otimes + [B \otimes r_0 + C \otimes r_1] \otimes r_1$$
$$= A \otimes r_0 \otimes + B \otimes r_0 \otimes r_1 + C \otimes r_1 \otimes r_1$$
$$= A \otimes r_0 \otimes + B \otimes r_{10} + C \otimes r_{11}$$

Even better, we can do symbolic operations with matrices:

Extraction (**car**, **cdr**)

$$A = \mathbf{W}_{\text{ex0}} T$$

$$B = \underline{\mathbf{W}_{\text{ex0}} \mathbf{W}_{\text{ex1}}} T$$

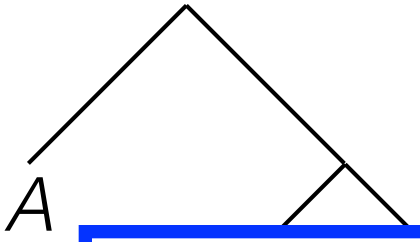
$$\mathbf{W}_{\text{ex01}}$$

One matrix

Construction (**cons**)

$$T = [A, T']$$
$$= \mathbf{W}_{\text{cons0}} A + \mathbf{W}_{\text{cons1}} T'$$

# Storing structured information




$$T = A \otimes r_0 \otimes + [B \otimes r_0 + C \otimes r_1] \otimes r_1$$

$$= A \otimes r_0 \otimes + B \otimes r_0 \otimes r_1 + C \otimes r_1 \otimes r_1$$

So you can build a whole LISP
in linear algebra!

Eve

Ext

$A =$

$B = \mathbf{W}_{\text{ex0}} \mathbf{W}_{\text{ex1}} T$

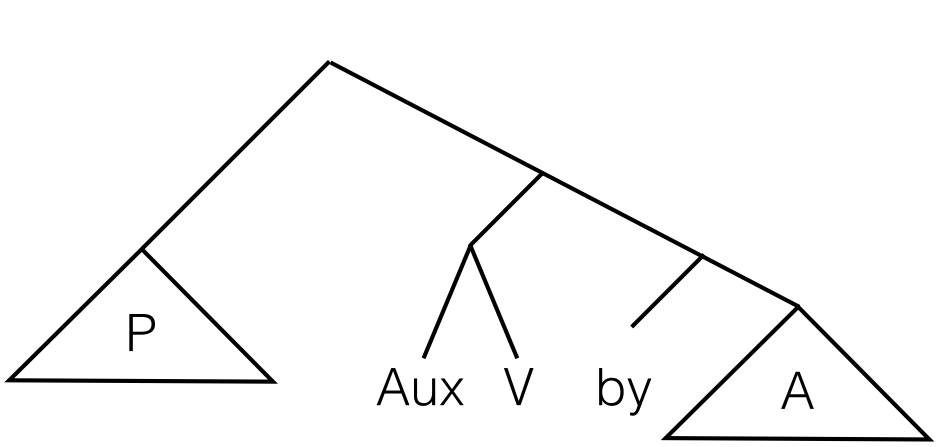
$= \mathbf{W}_{\text{cons0}} \mathbf{T} + \mathbf{W}_{\text{cons1}} \mathbf{T}$

$\mathbf{W}_{\text{ex01}}$

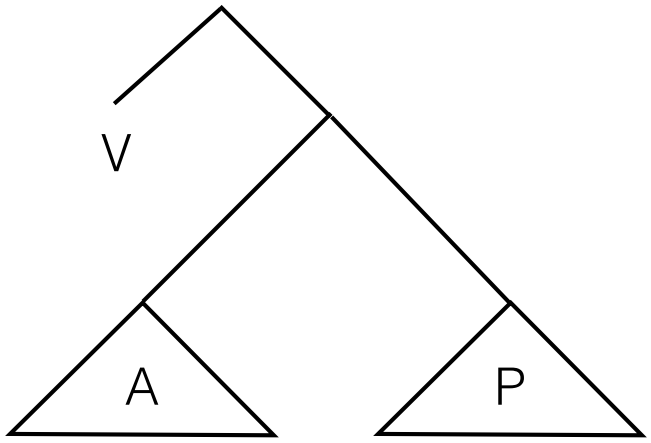
One matrix

# Storing structured information

One more example: Go from passive-voice sentences to predicate representation




$F$




[Few movies] are admired by [skeptical critics]

Admire( [Few movies], [skeptical critics] )

$$F(T) = \mathbf{W}T$$

$$\mathbf{W} = \mathbf{W}_{\text{cons0}}[\mathbf{W}_{\text{ex1}}\mathbf{W}_{\text{ex0}}\mathbf{W}_{\text{ex1}}] + \mathbf{W}_{\text{cons1}}[\mathbf{W}_{\text{cons0}}(\mathbf{W}_{\text{ex1}}\mathbf{W}_{\text{ex1}}\mathbf{W}_{\text{ex1}}) + \mathbf{W}_{\text{cons1}}\mathbf{W}_{\text{ex0}}]$$

**This is just one matrix multiplication**

# Encoding grammars

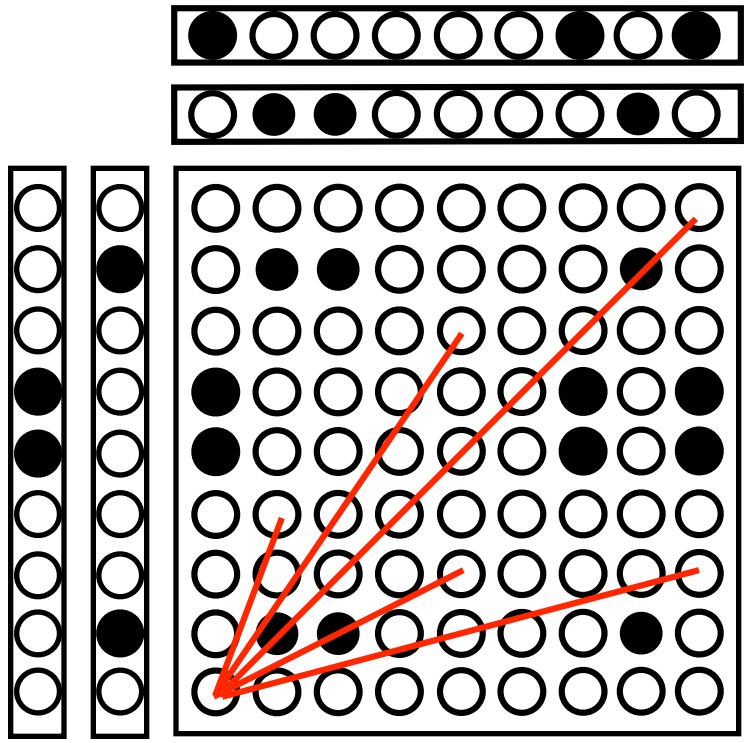
Grammaticality can be represented as constraints on which role/filler pairs.

A singular subject requires a correctly conjugated verb, e.g.

Such constraints can be encoded as weights in the “binding matrix.” (Note the departure from matrix memories, in which the entries in the matrix were synapses.)

These weights give us an energy function, and evolution dynamics.

Over time, the network will settle into low energy (=high grammaticality) states.



# Optimization and Quantization in Gradient Symbol Systems: A Framework for Integrating the Continuous and the Discrete in Cognition

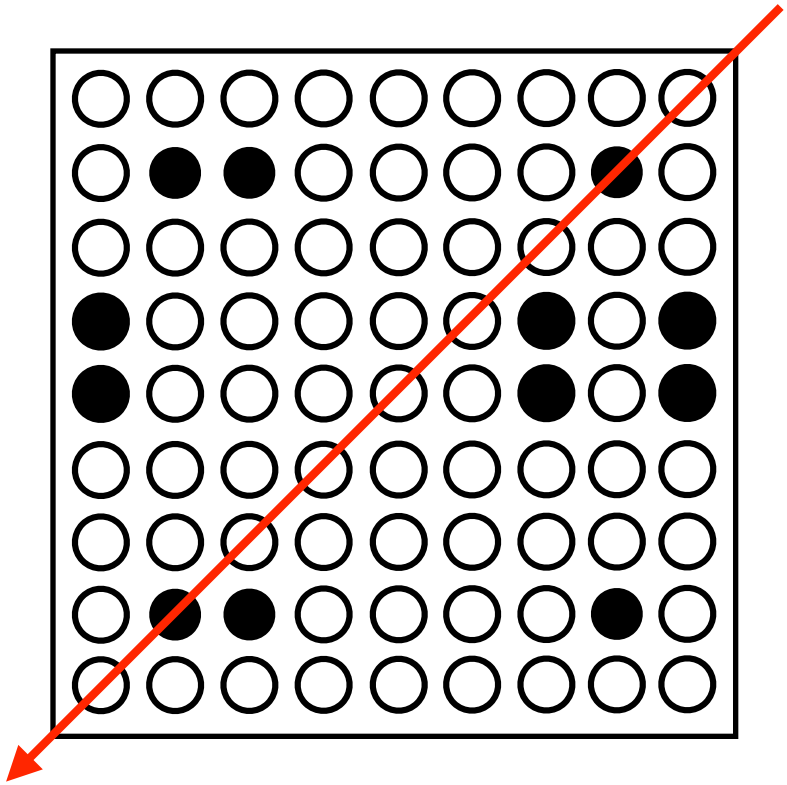
Paul Smolensky*, Matthew Goldrick†, and Donald Mathis*
*Department of Cognitive Science, Johns Hopkins University
†Department of Linguistics, Northwestern University

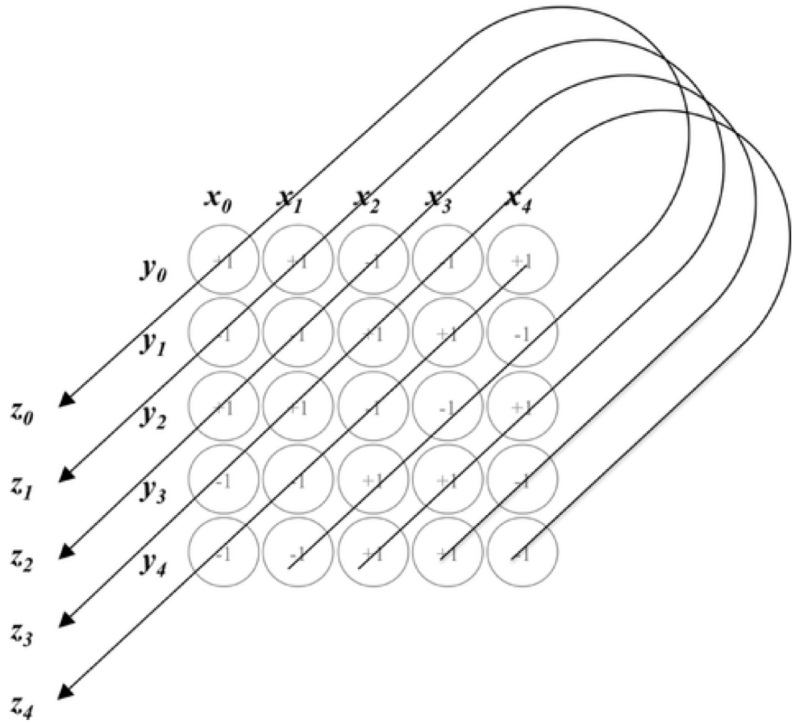
## Abstract

Mental representations have continuous as well as discrete, combinatorial properties. For example, while predominantly discrete, phonological representations also vary continuously; this is reflected by gradient effects in instrumental studies of speech production. Can an integrated theoretical framework address both aspects of structure? The framework we introduce here, Gradient Symbol Processing, characterizes the emergence of grammatical macrostructure from the Parallel Distributed Processing microstructure (McClelland & Rumelhart, 1986) of language processing. The mental representations that emerge, Distributed Symbol Systems, have both combinatorial and gradient structure. They are processed through Subsymbolic Optimization-Quantization, in which an optimization process favoring representations that satisfy well-formedness constraints operates in parallel with a distributed quantization process favoring discrete symbolic structures. We apply a particular instantiation of this framework, λ-Diffusion Theory, to phonological production. Simulations of the resulting model suggest that Gradient Symbol Processing offers a way to unify accounts of grammatical competence with both discrete and continuous patterns in language performance.

# Other frameworks: Vector-symbolic architectures

**Idea:** represent role/filler bindings as vectors, rather than tensors; *everything* is the same size.



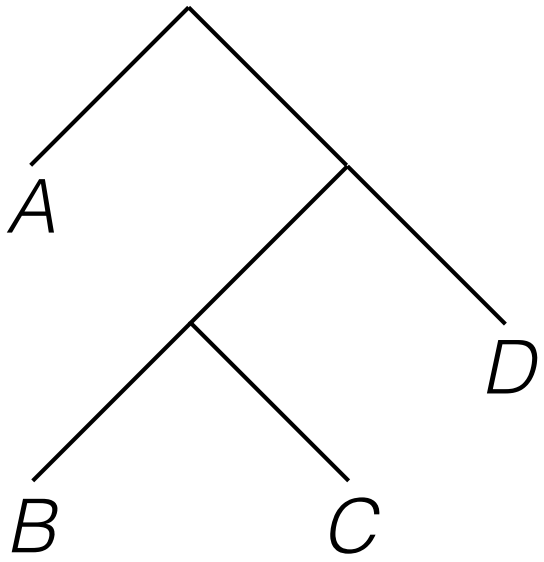



Circular covolution (Plate)

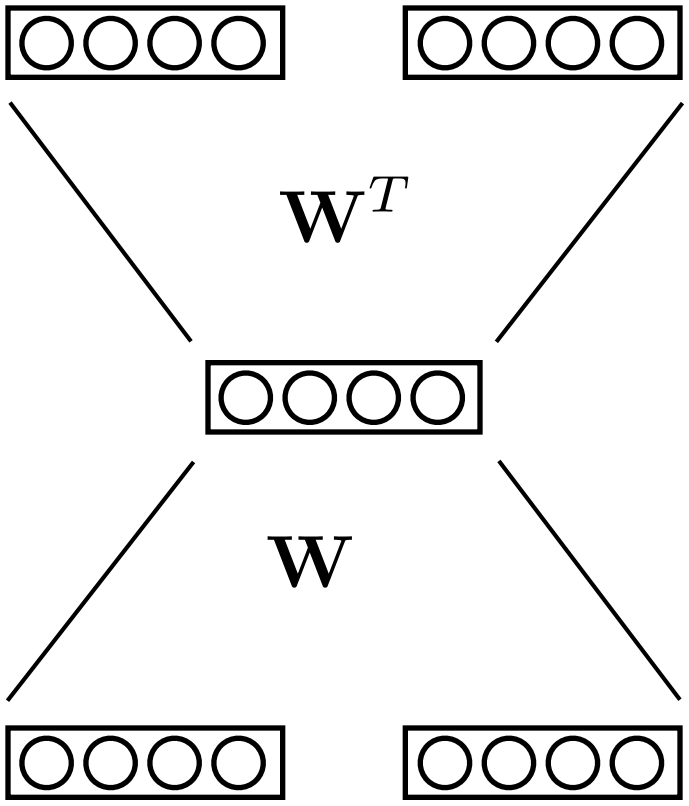
Pollack (1990)

**Recursive Auto-Associative Memory (RAAM)**

*Learn* the compression function.



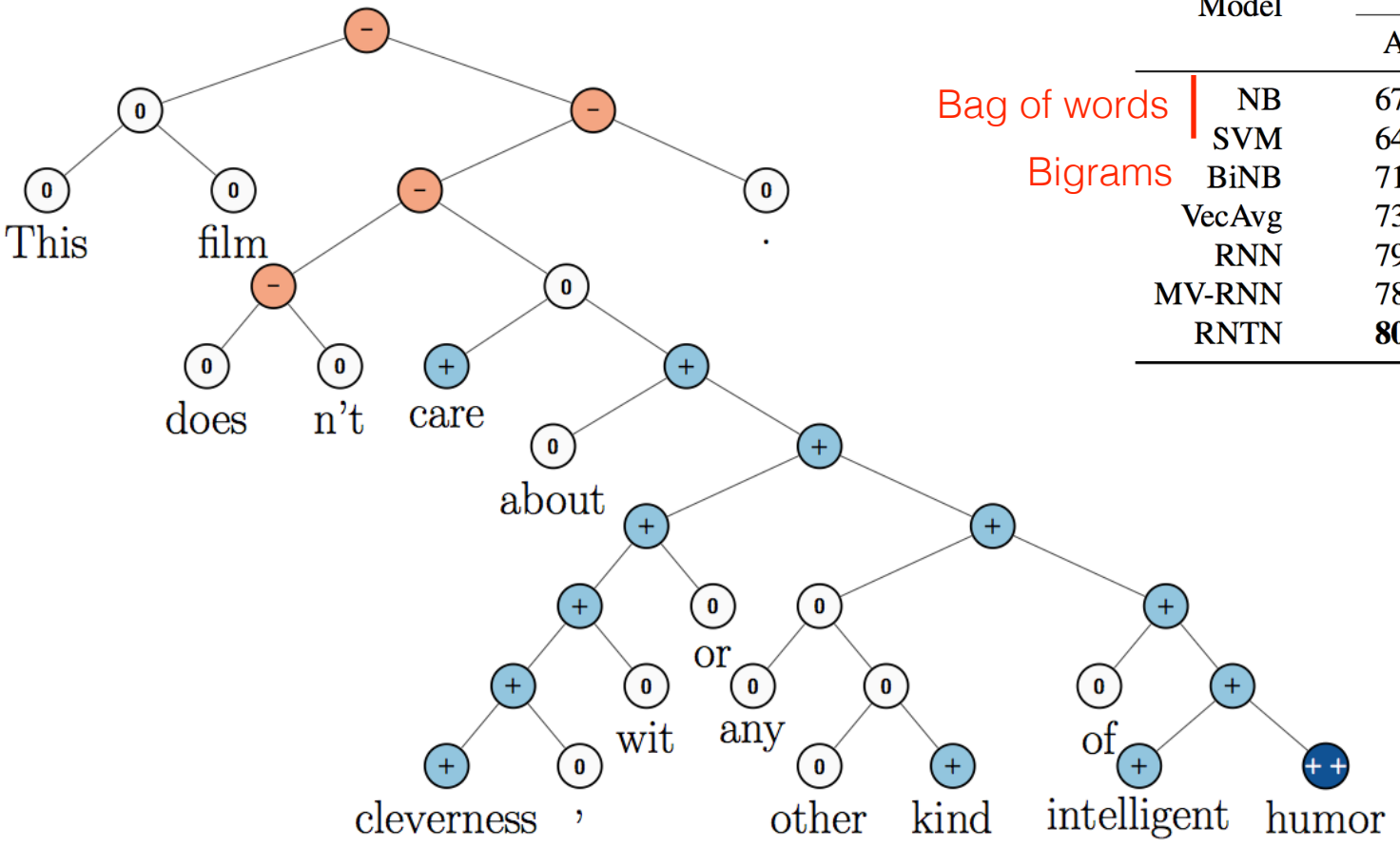

$$\mathbf{W}[\mathbf{W}[\mathbf{W}[B, C], D], A]$$




Just like in an ordinary autoencoder, we learn the same weights for encoding and decoding.

# Modern applications: **Socher et al. 2013**

Similar to RAAMs, that the model learns a parametrized compression function; details are different.




| | Model | Fine-grained | | Positive/Negative | |
|---|---|---|---|---|---|
| | | All | Root | All | Root |
| Bag of words | NB | 67.2 | 41.0 | 82.6 | 81.8 |
| | SVM | 64.3 | 40.7 | 84.6 | 79.4 |
| Bigrams | BiNB | 71.0 | 41.9 | 82.7 | 83.1 |
| | VecAvg | 73.3 | 32.7 | 85.1 | 80.1 |
| | RNN | 79.0 | 43.2 | 86.1 | 82.4 |
| | MV-RNN | 78.7 | 44.4 | 86.8 | 82.9 |
| | RNTN | **80.7** | **45.7** | **87.6** | **85.4** |

End