

# Attention mechanisms

CS 685, Fall 2021

Advanced Natural Language Processing

Mohit Iyyer

College of Information and Computer Sciences

University of Massachusetts Amherst

*some slides from Richard Socher & Emma Strubell*

# stuff from last time...

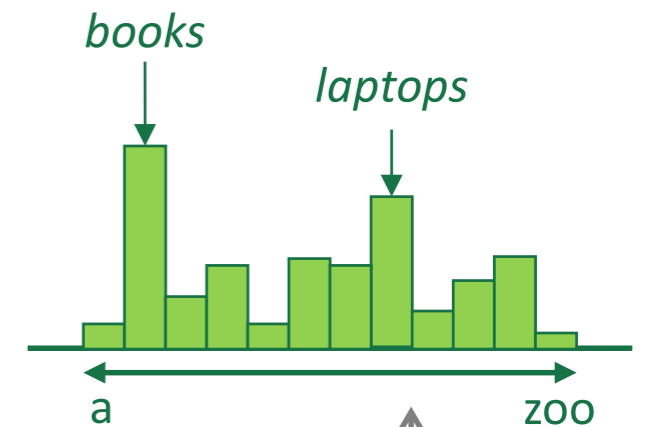
- HW0 grading hopefully done by next week
- First quiz due today
- HW1 will be out within the next 2 weeks
- Project proposals due 9/24, all group assignments have been finalized
- Compute resources for projects?
- Do NNs process text similarly to how we process language in our brains?

# A RNN Language Model

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$

output distribution

$$\hat{y} = \text{softmax}(W_2 h^{(t)} + b_2)$$



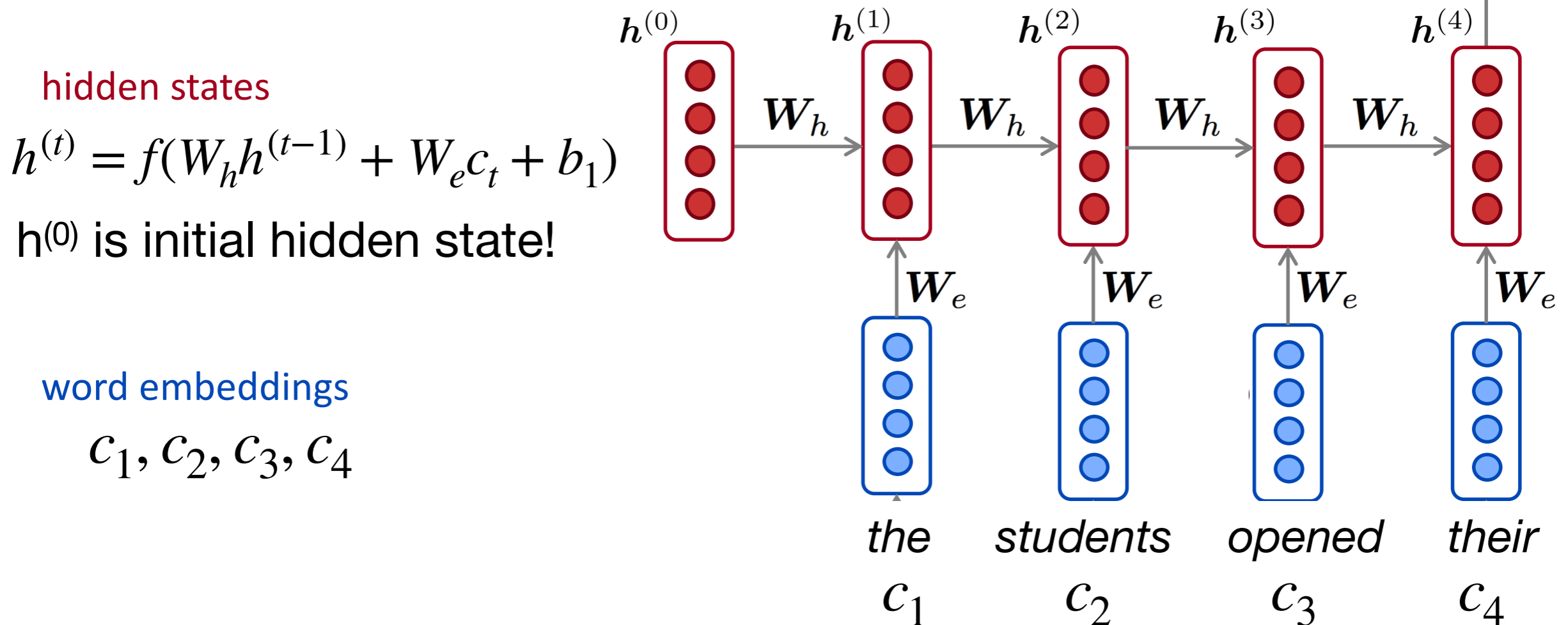
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t + b_1)$$

$h^{(0)}$  is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



## why is this good?

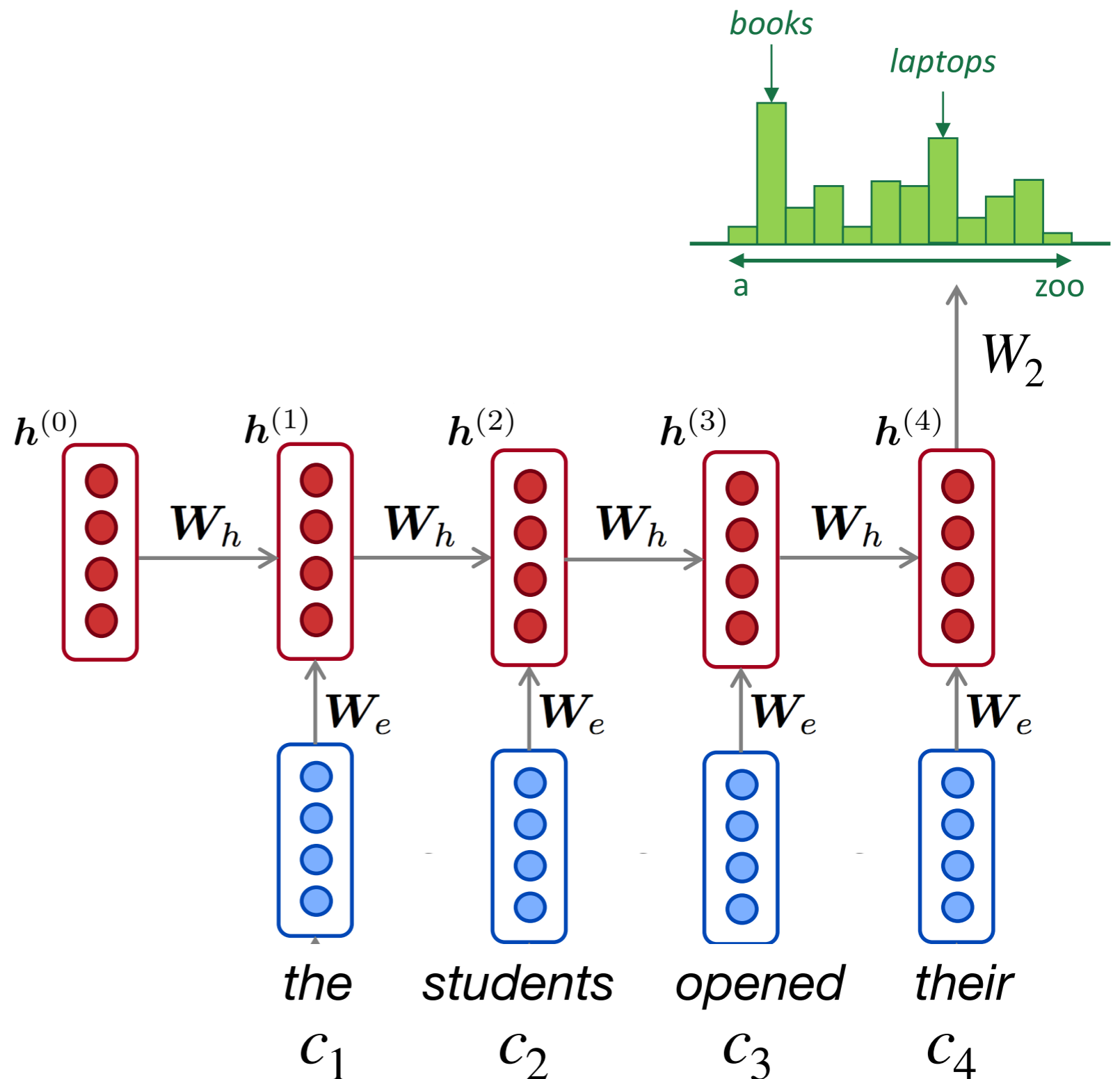
### RNN Advantages:

- Can process **any length** input
- **Model size doesn't increase** for longer input
- Computation for step  $t$  can (in theory) use information from **many steps back**
- Weights are **shared** across timesteps  $\rightarrow$  representations are shared

### RNN Disadvantages:

- Recurrent computation is **slow**
- In practice, difficult to access information from **many steps back**

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$



# Training a RNN Language Model

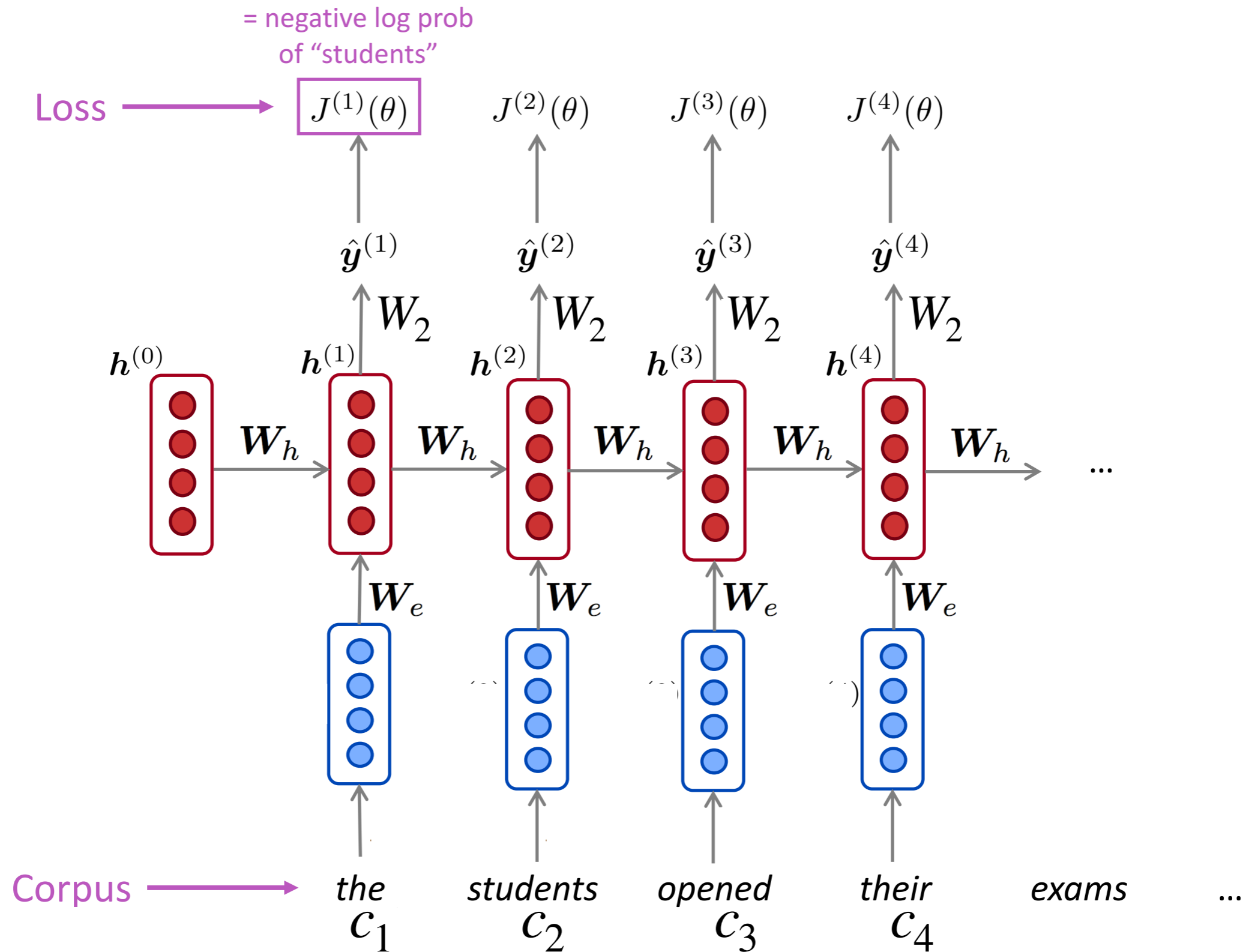
- Get a **big corpus of text** which is a sequence of words  $x^{(1)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution  $\hat{y}^{(t)}$  **for every step  $t$** .
  - i.e. predict probability dist of *every word*, given words so far
- **Loss function** on step  $t$  is usual cross-entropy between our predicted probability distribution  $\hat{y}^{(t)}$ , and the true next word  $y^{(t)} = x^{(t+1)}$ :

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{j=1}^{|\mathcal{V}|} y_j^{(t)} \log \hat{y}_j^{(t)}$$

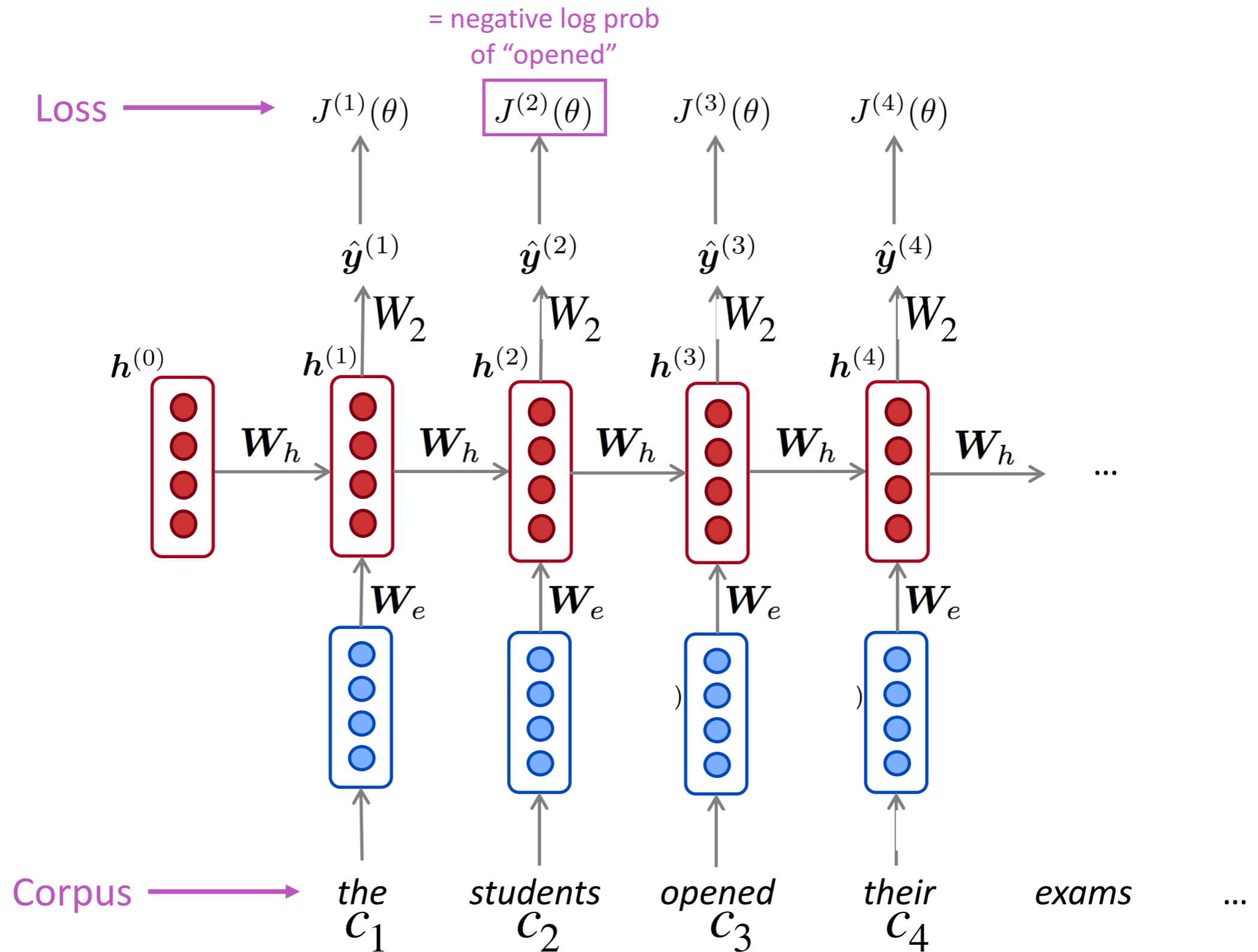
- Average this to get **overall loss** for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

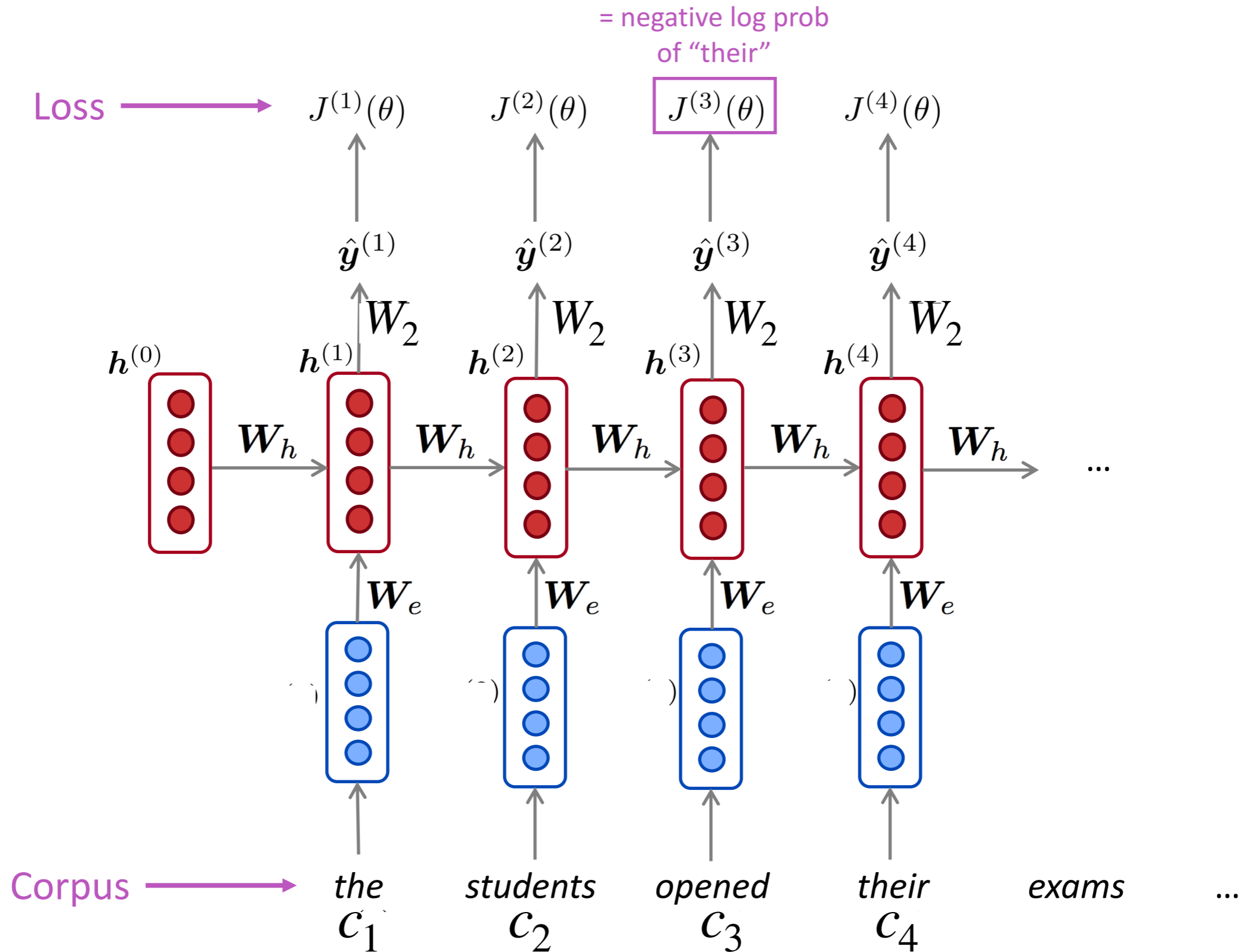
# Training a RNN Language Model



# Training a RNN Language Model

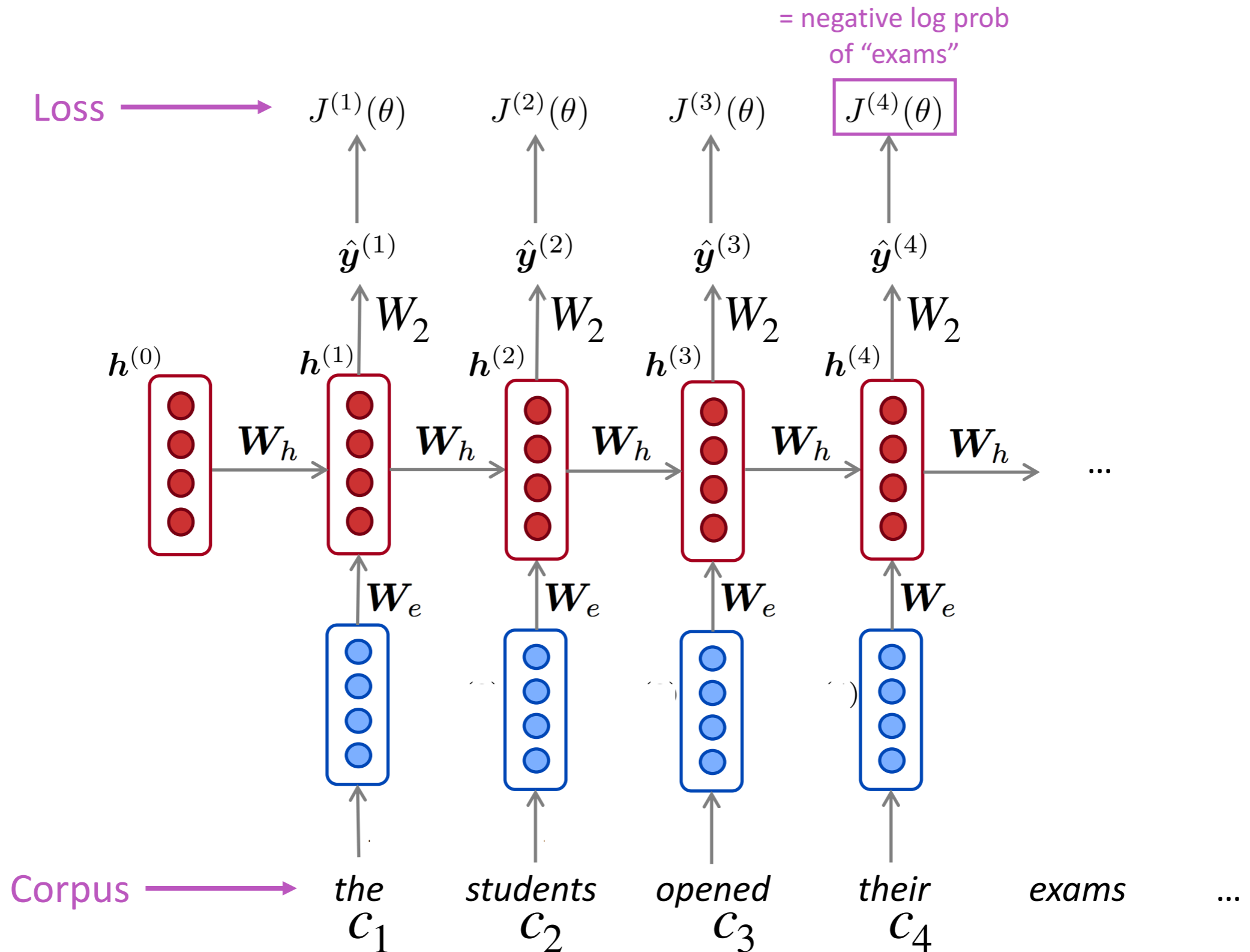


# Training a RNN Language Model





# Training a RNN Language Model

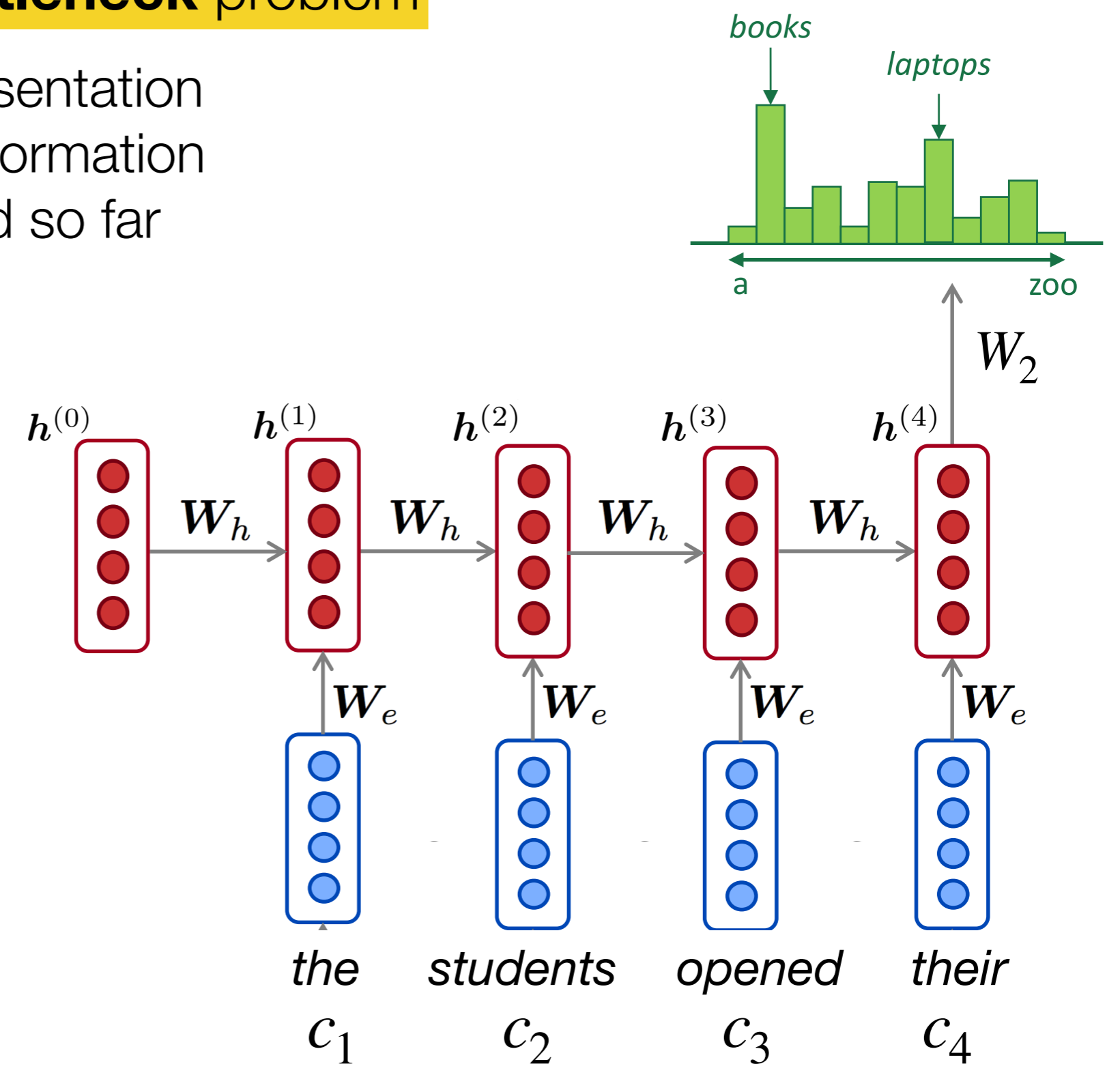




# RNNs suffer from a **bottleneck** problem

The current hidden representation must encode all of the information about the text observed so far

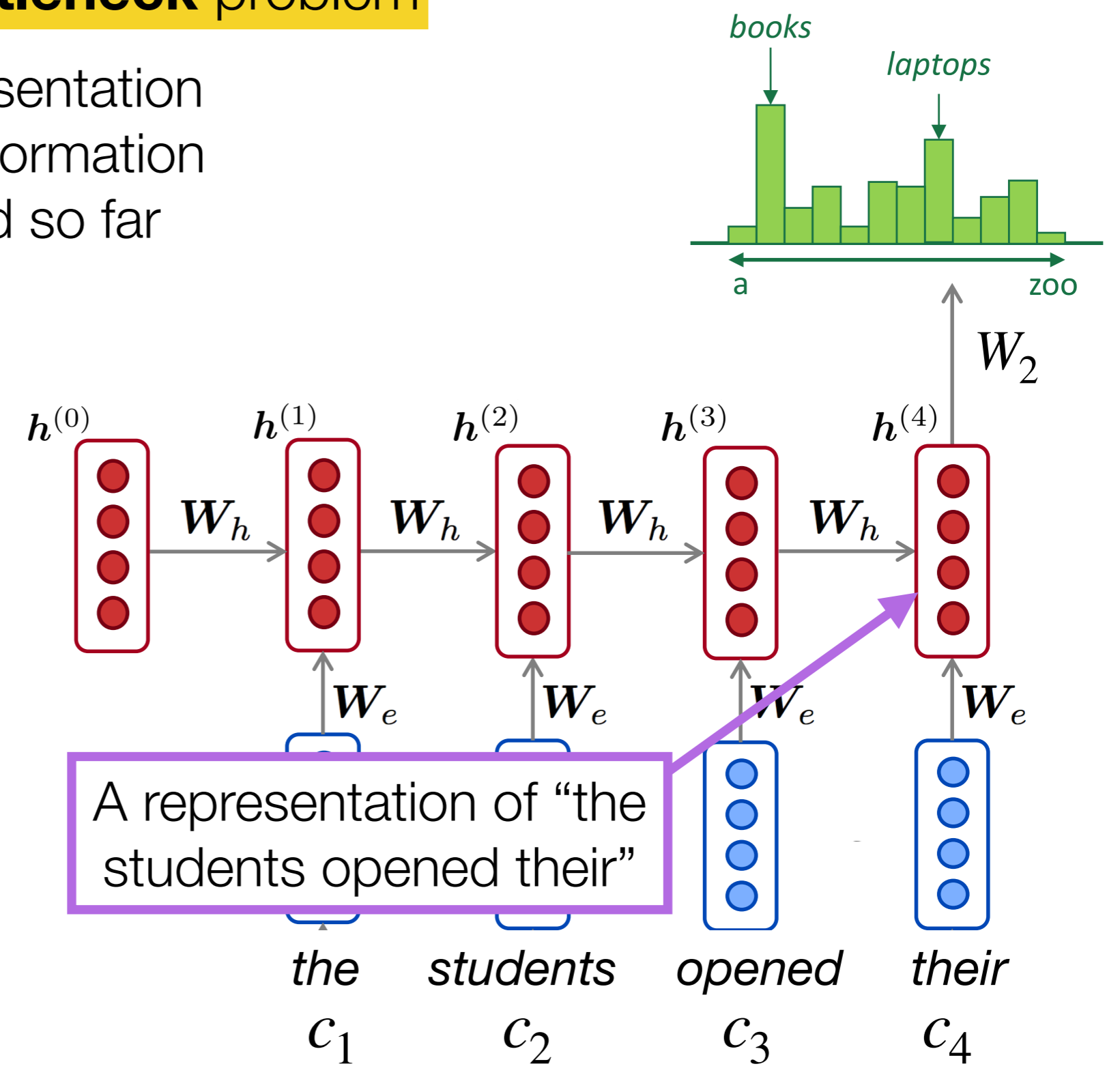
$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$



# RNNs suffer from a **bottleneck** problem

The current hidden representation must encode all of the information about the text observed so far

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$

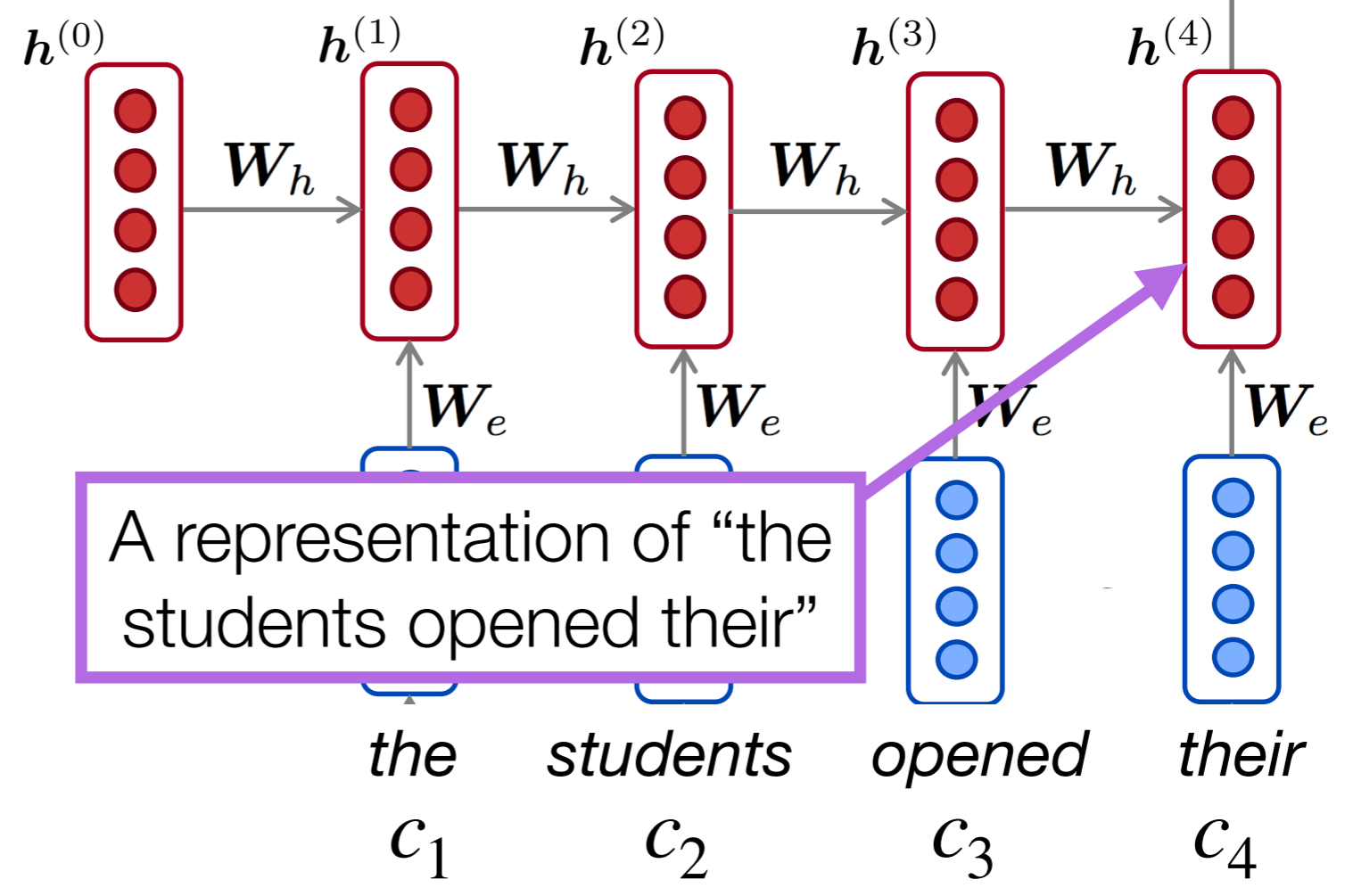
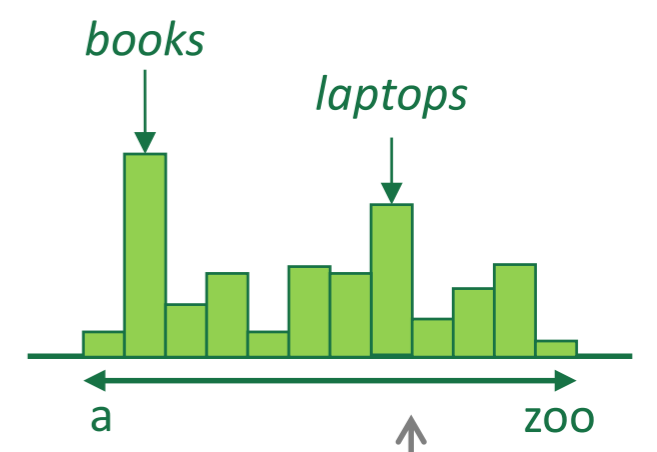


# RNNs suffer from a **bottleneck** problem

The current hidden representation must encode all of the information about the text observed so far

This becomes difficult especially with longer sequences

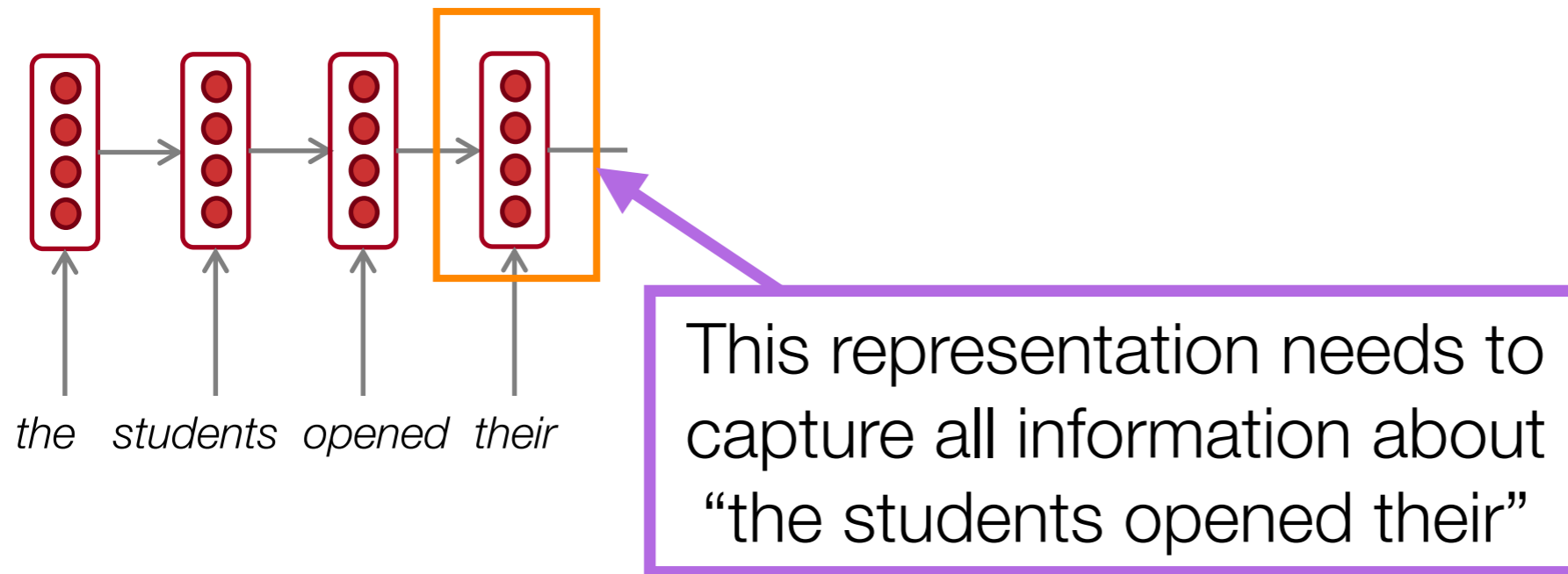
$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$



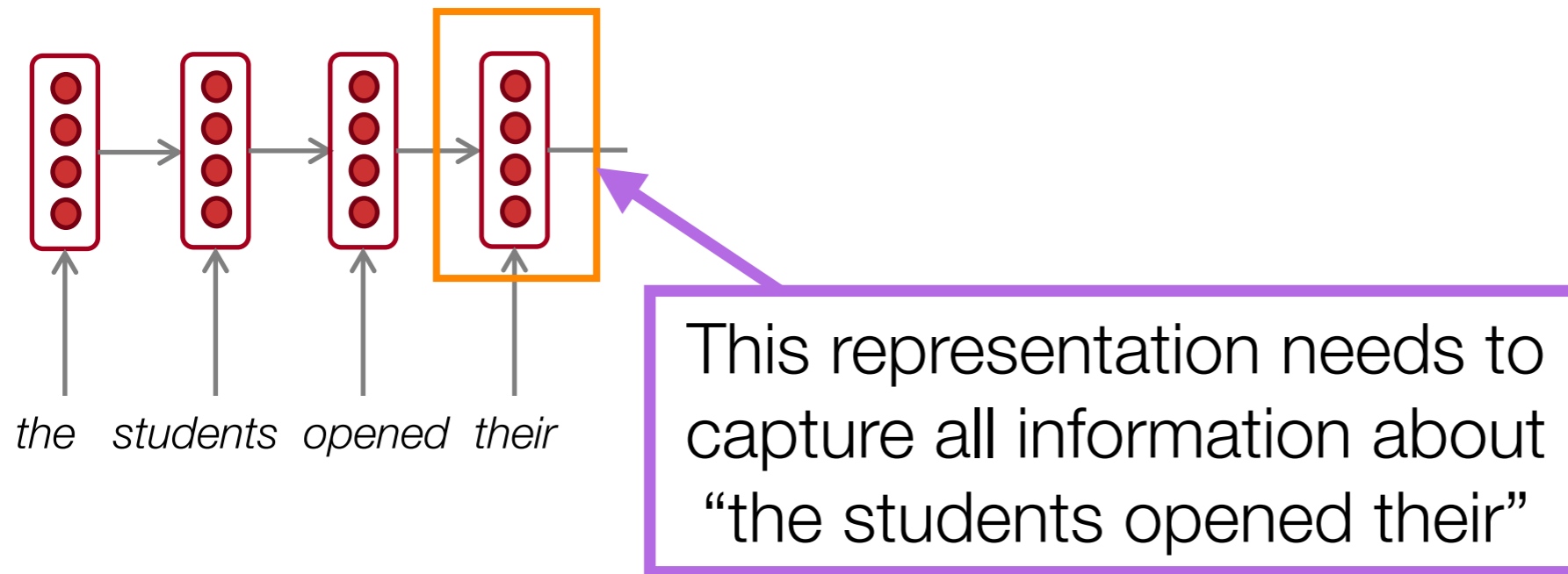
“you can’t cram the meaning  
of a whole %&@#&ing  
sentence into a single  
\$\*(&@ing vector!”

— Ray Mooney (NLP professor at UT Austin)

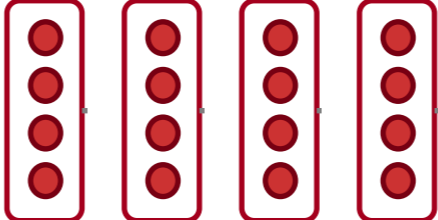
idea: what if we use multiple vectors?



idea: what if we use multiple vectors?



Instead of this, let's try:

the students opened their =  (all 4 hidden states!)



# The solution: **attention**

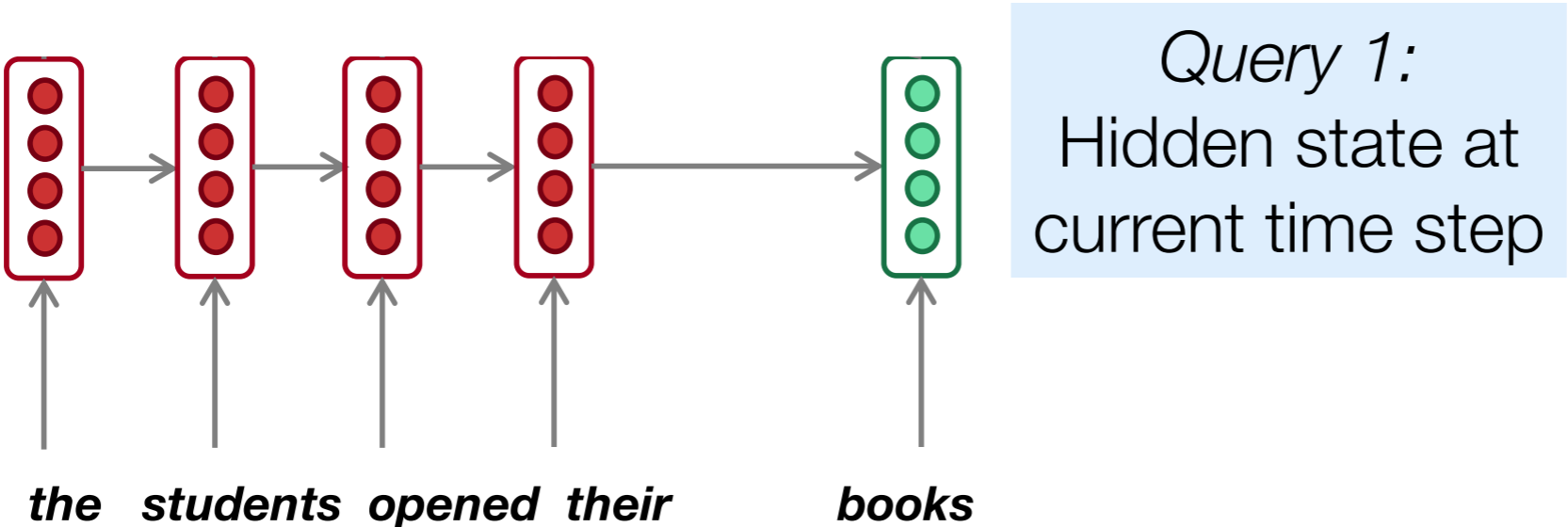
- **Attention mechanisms** (Bahdanau et al., 2015) allow language models to focus on a particular part of the observed context at each time step
  - Originally developed for machine translation, and intuitively similar to *word alignments* between different languages

# How does it work?

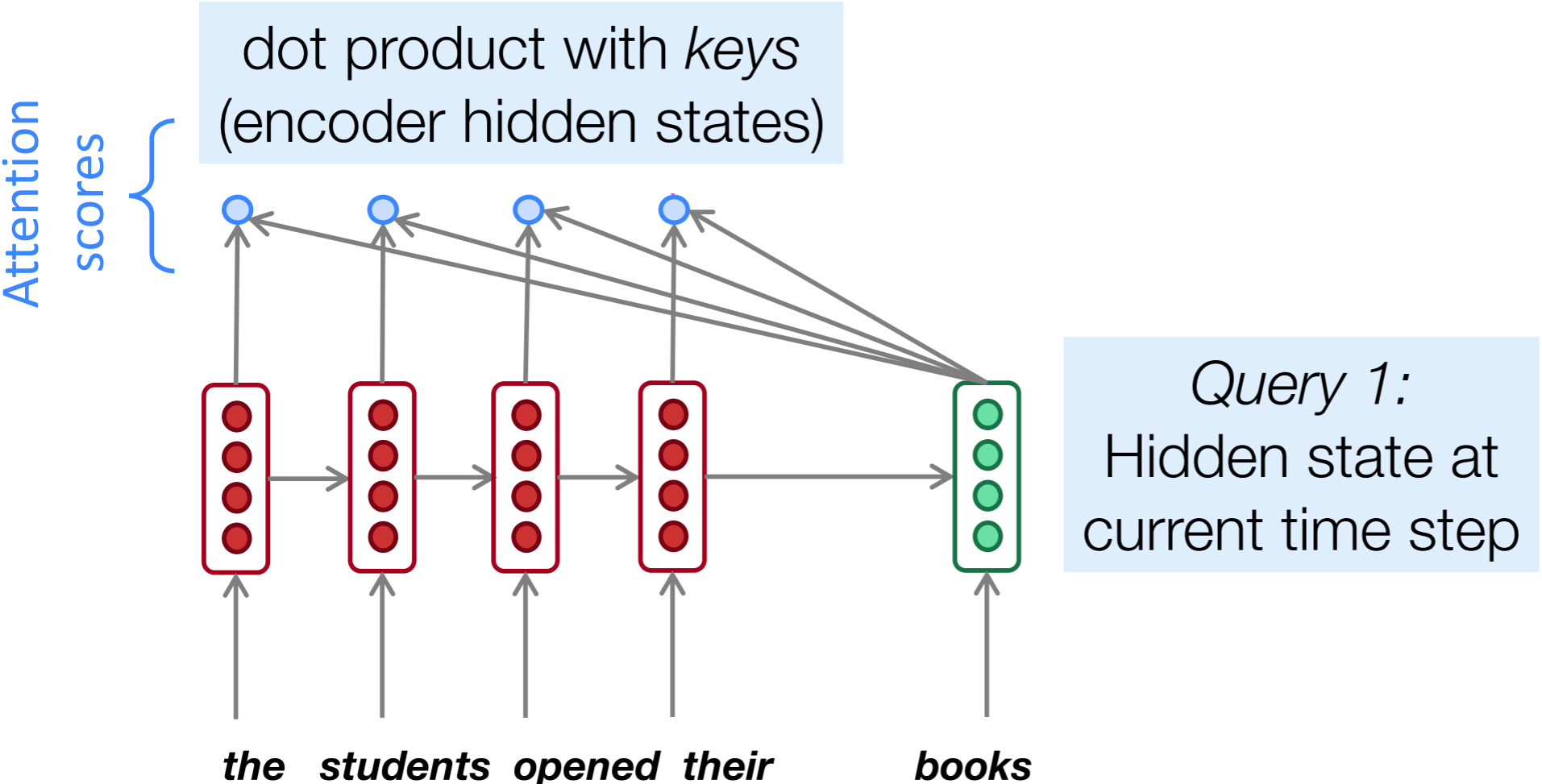
- in general, we have a single *query* vector and multiple *key* vectors. We want to score each query-key pair

in a neural language model, what are the queries and keys?

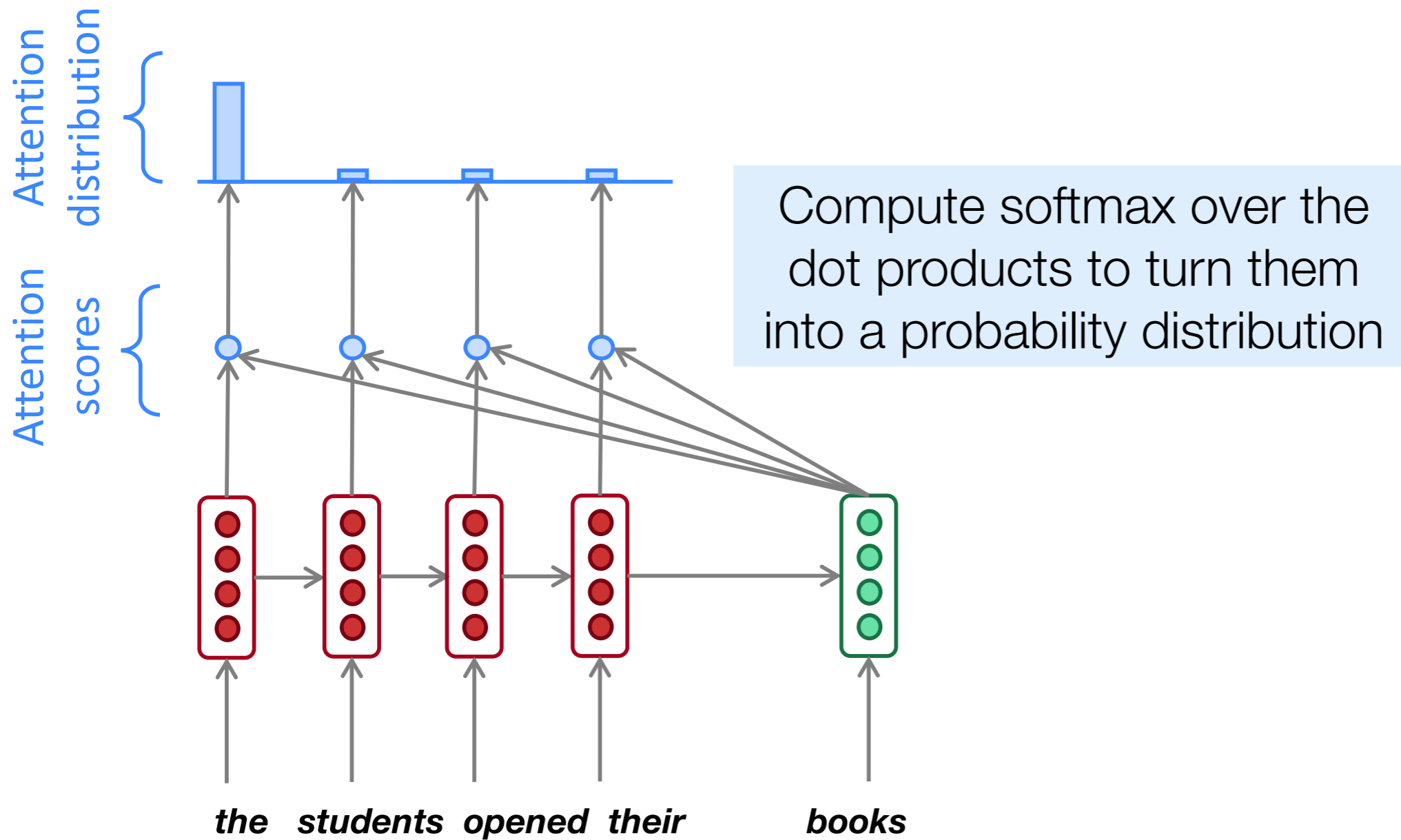
# Attention mechanisms in neural language models



# Attention mechanisms in neural language models

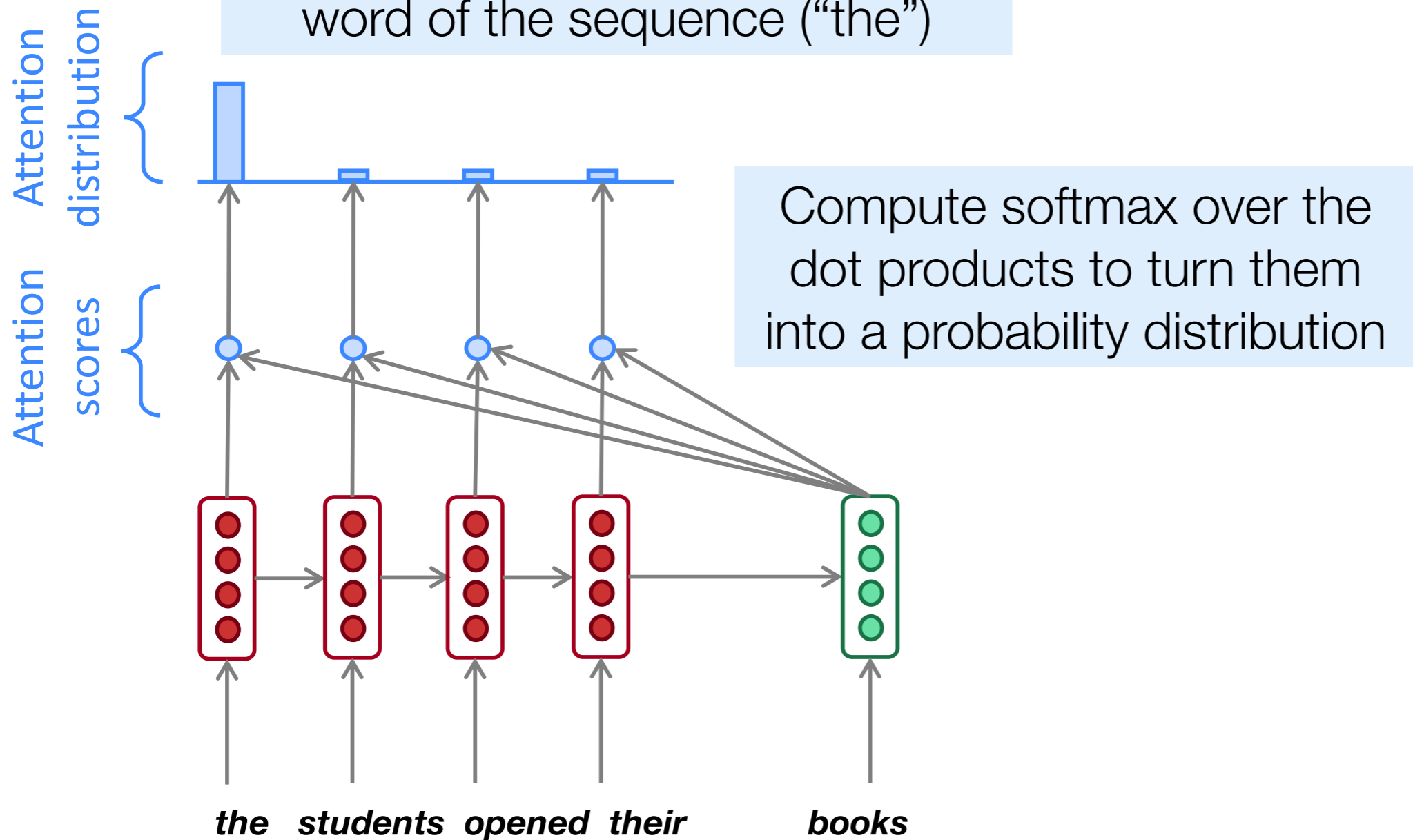


# Attention mechanisms in neural language models

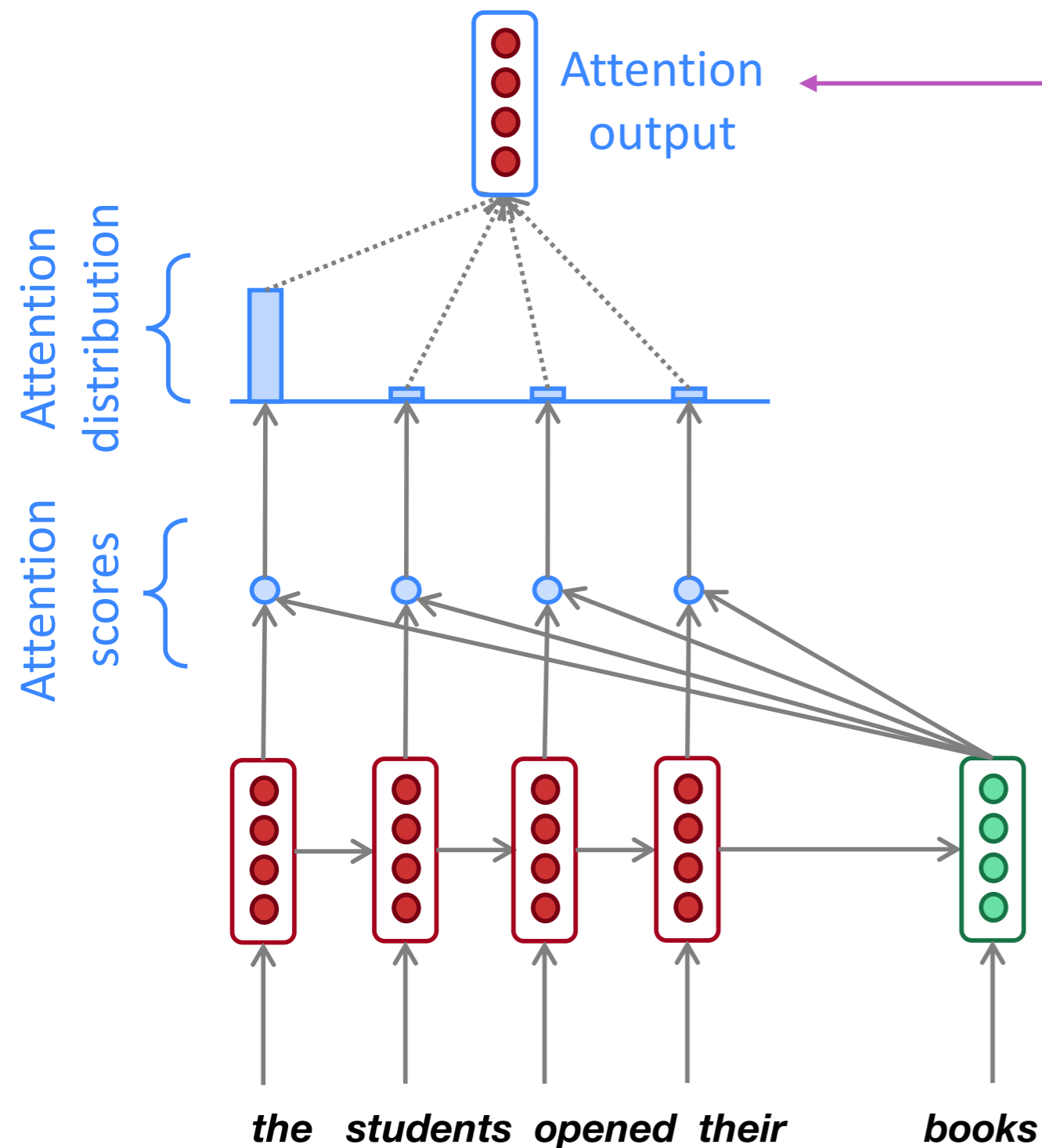


# Attention mechanisms in neural language models

At this time step, the attention distribution is focused on the first word of the sequence ("the")



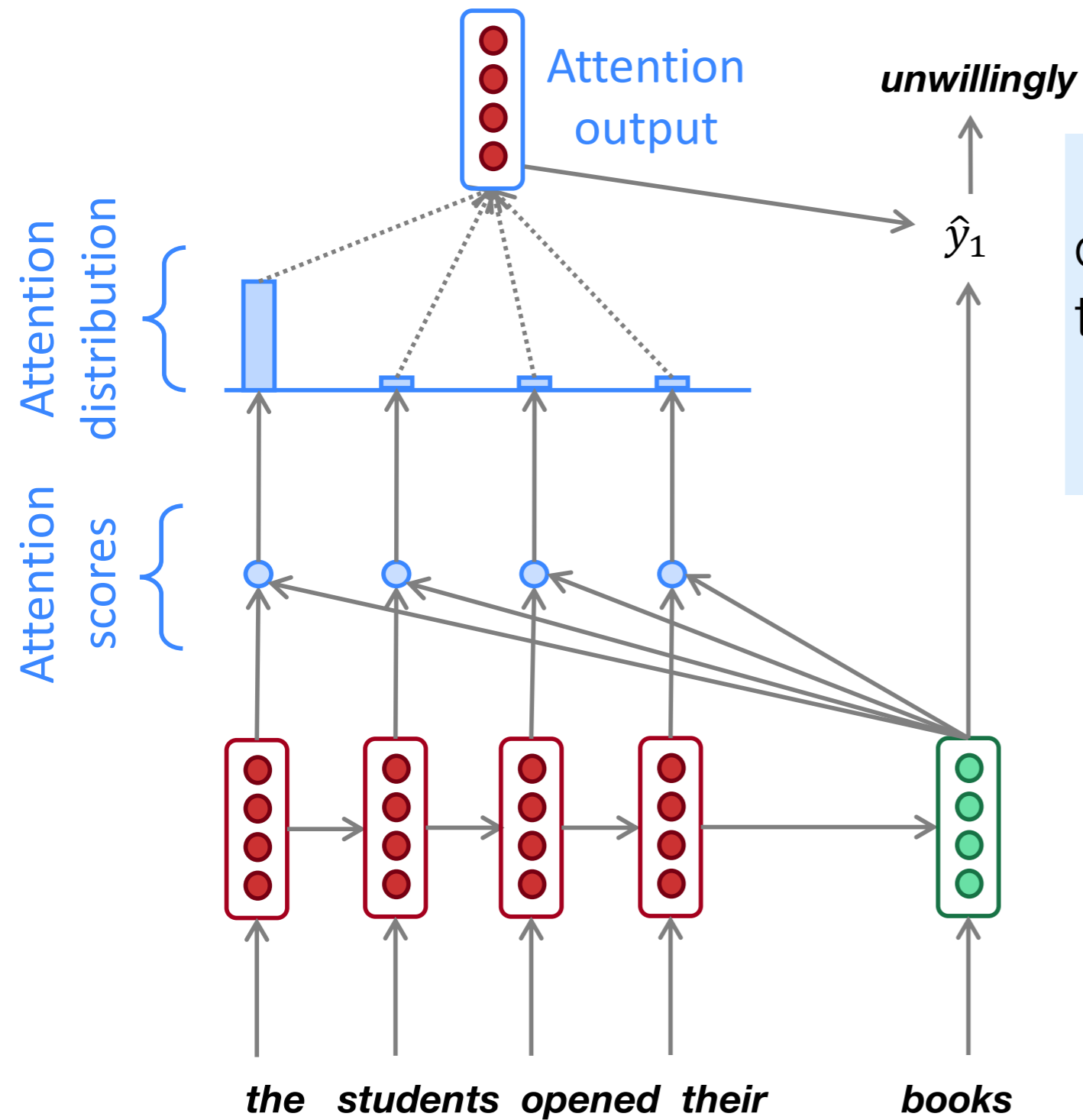
# Attention mechanisms in neural language models



We use the attention distribution to compute a weighted average of the hidden states.

Intuitively, the resulting attention output contains information from hidden states that received high attention scores

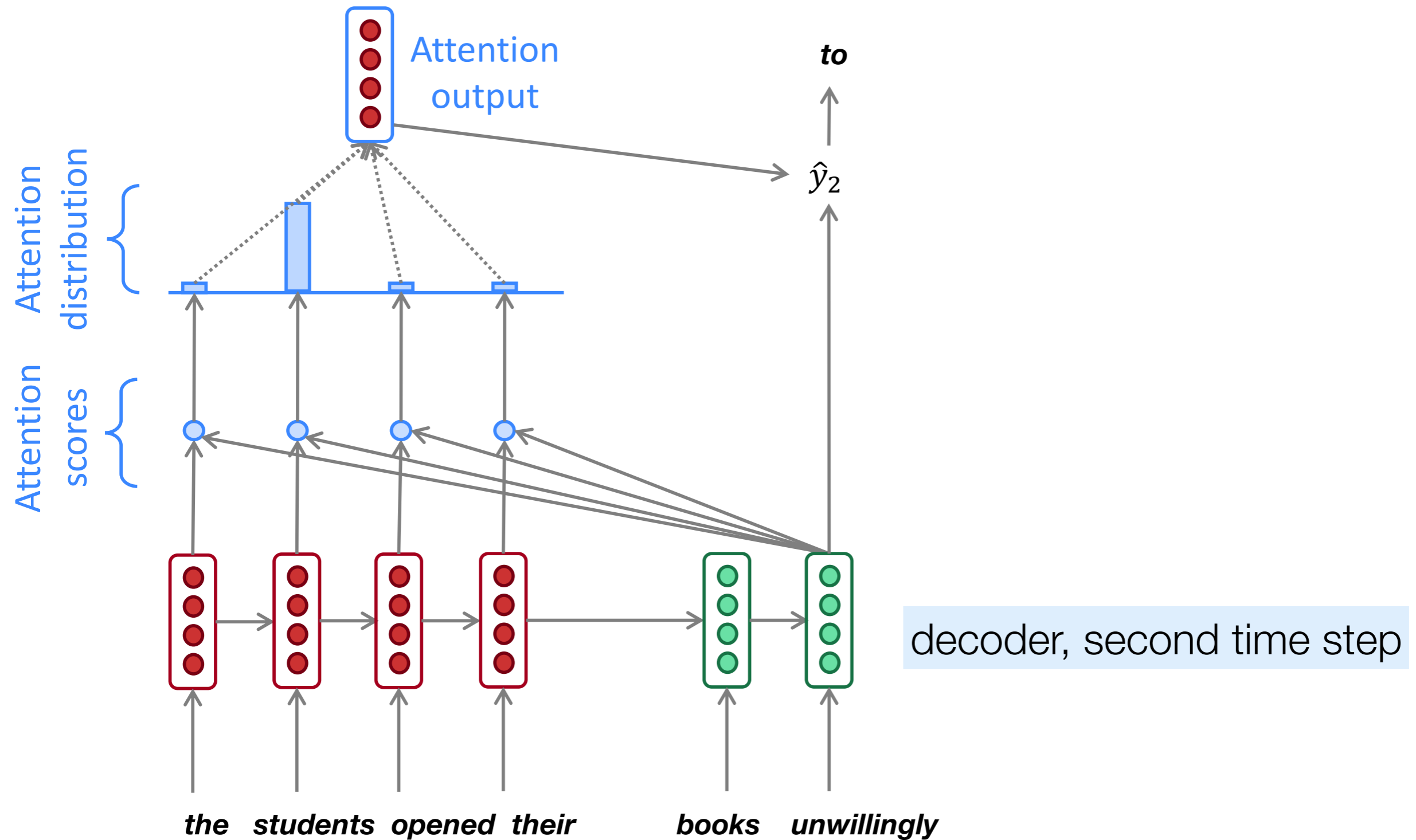
# Sequence-to-sequence with attention




Concatenate (or otherwise compose) the attention output with the current hidden state, then pass through a softmax layer to predict the next word



# Sequence-to-sequence with attention



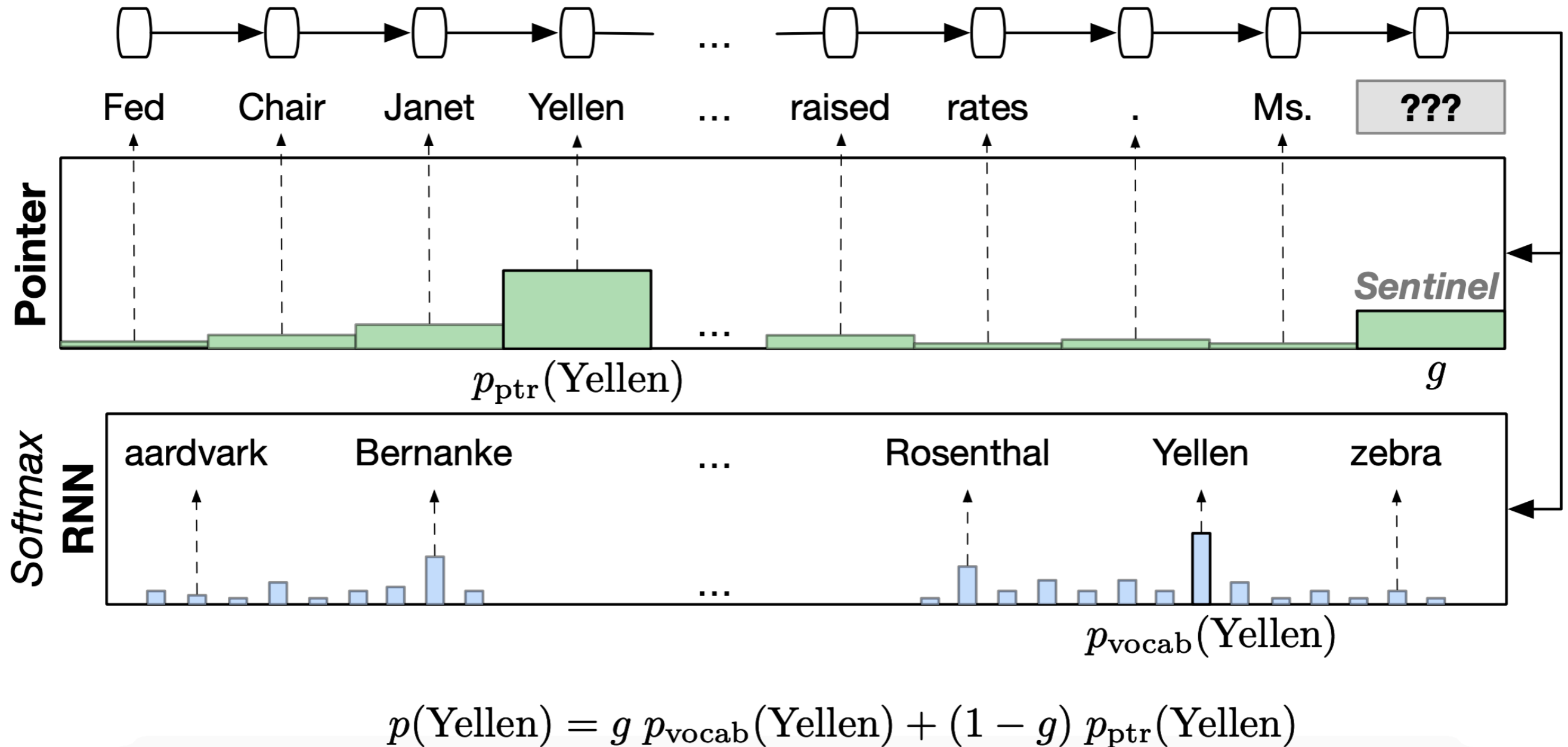
- Attention **solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention **helps with vanishing gradient problem**
  - Provides shortcut to faraway states
- Attention provides **some interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on 
  - We get **alignment for free!**
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

	Les	pauvres	sont	démunis
The				
poor				
don't				
have				
any				
money				

# Many variants of attention

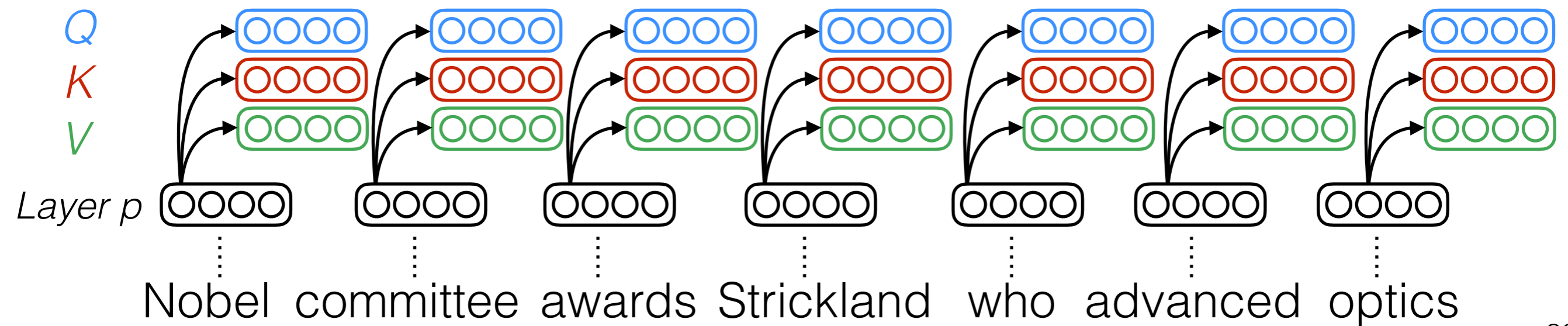
- Original formulation:  $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product:  $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$  Luong et al., 2015
- Dot product:  $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$  Luong et al., 2015
- Scaled dot product:  $a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$  Vaswani et al., 2017

# Attention can also be used to copy tokens from the context!

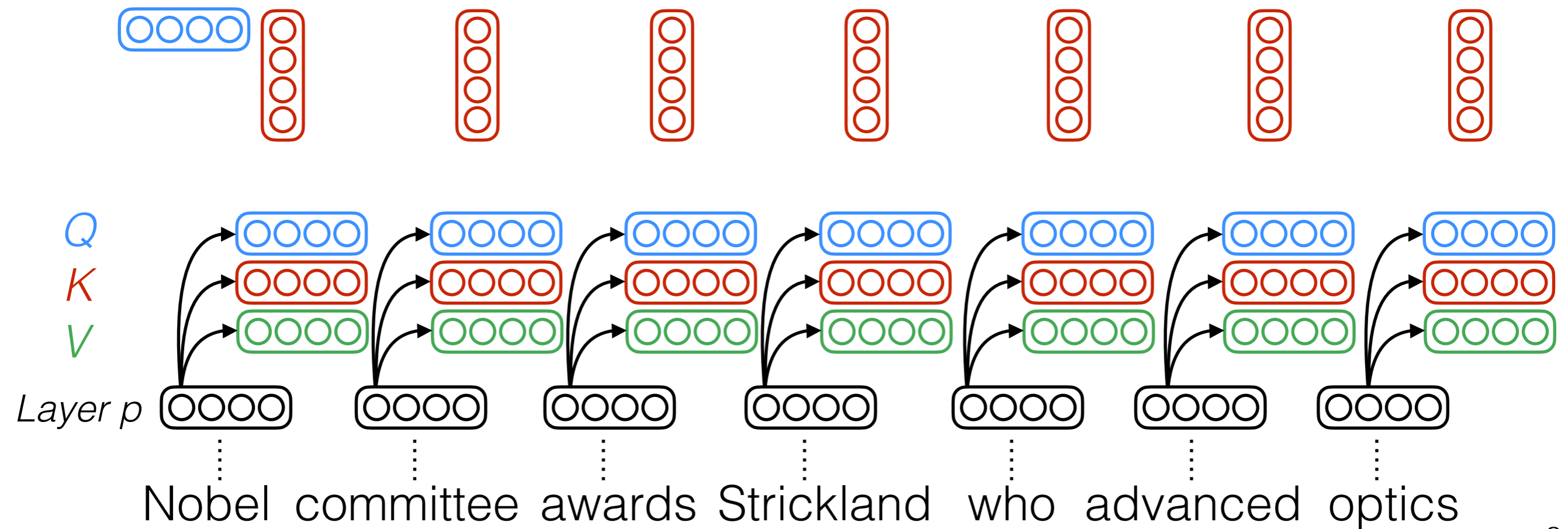


# iPad

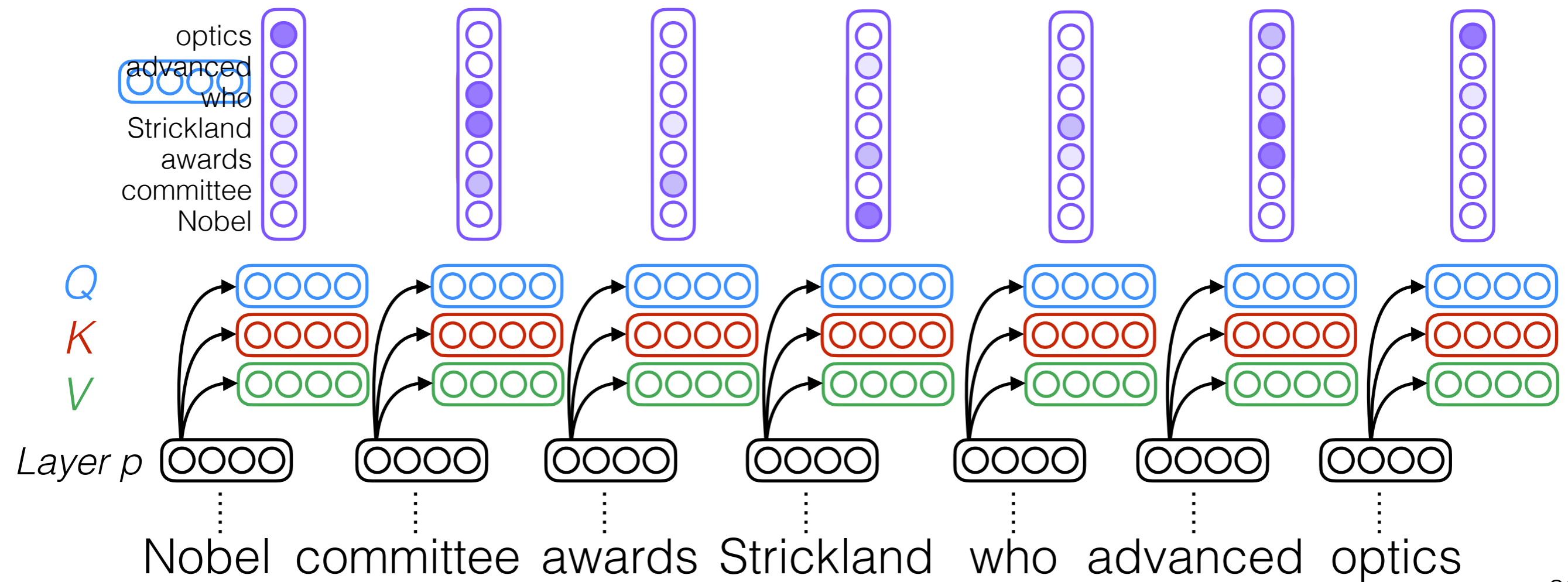
# Self-attention



# Self-attention

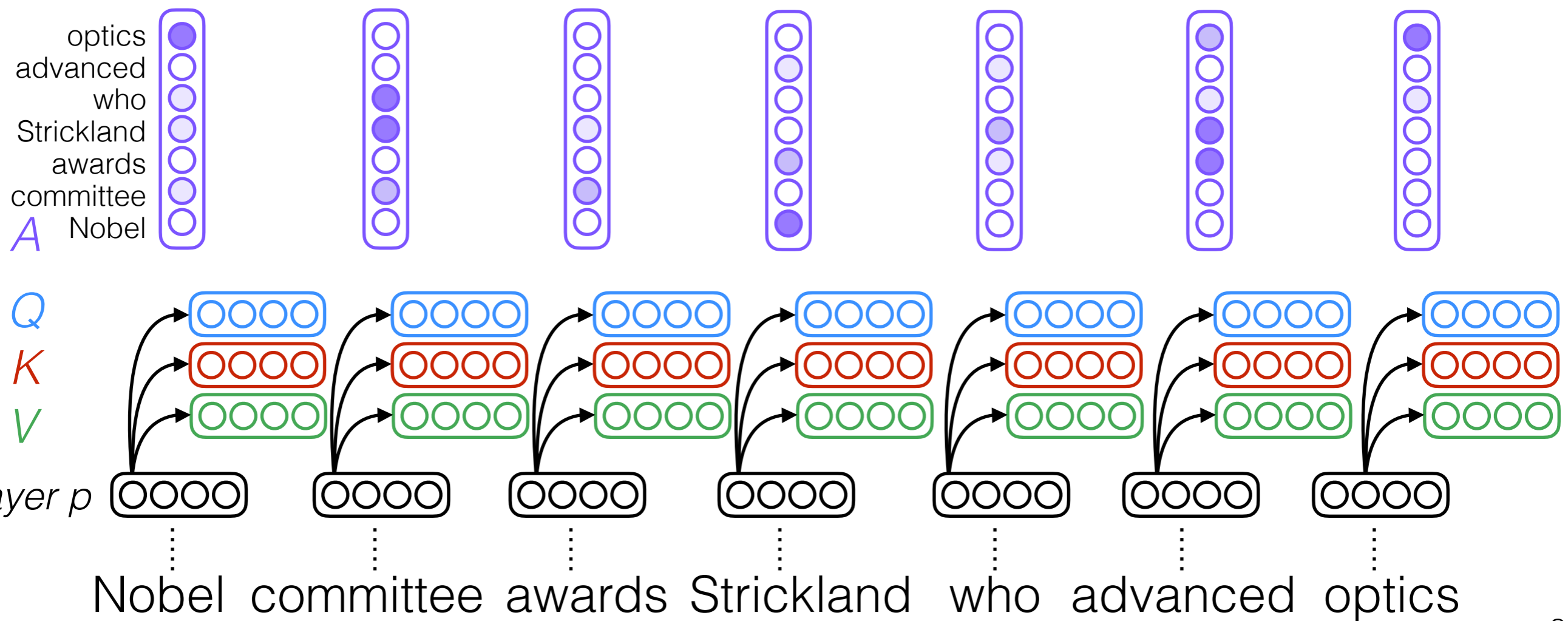


# Self-attention

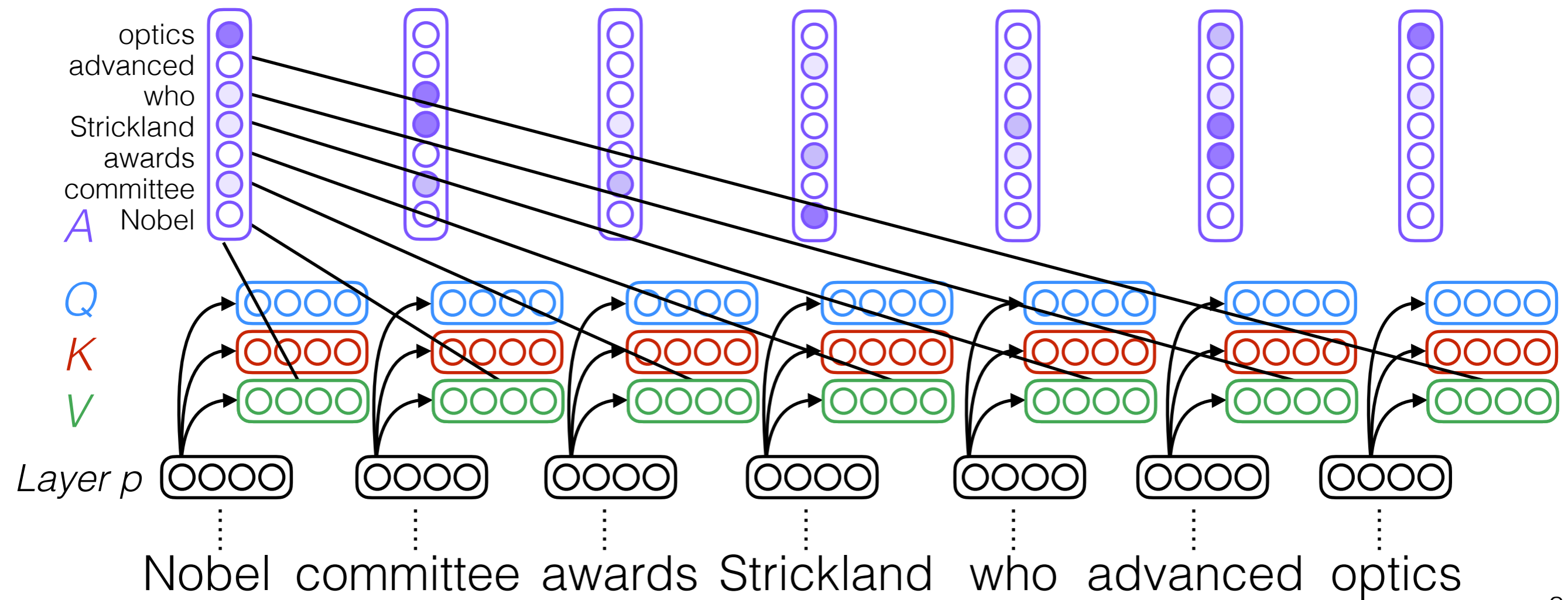




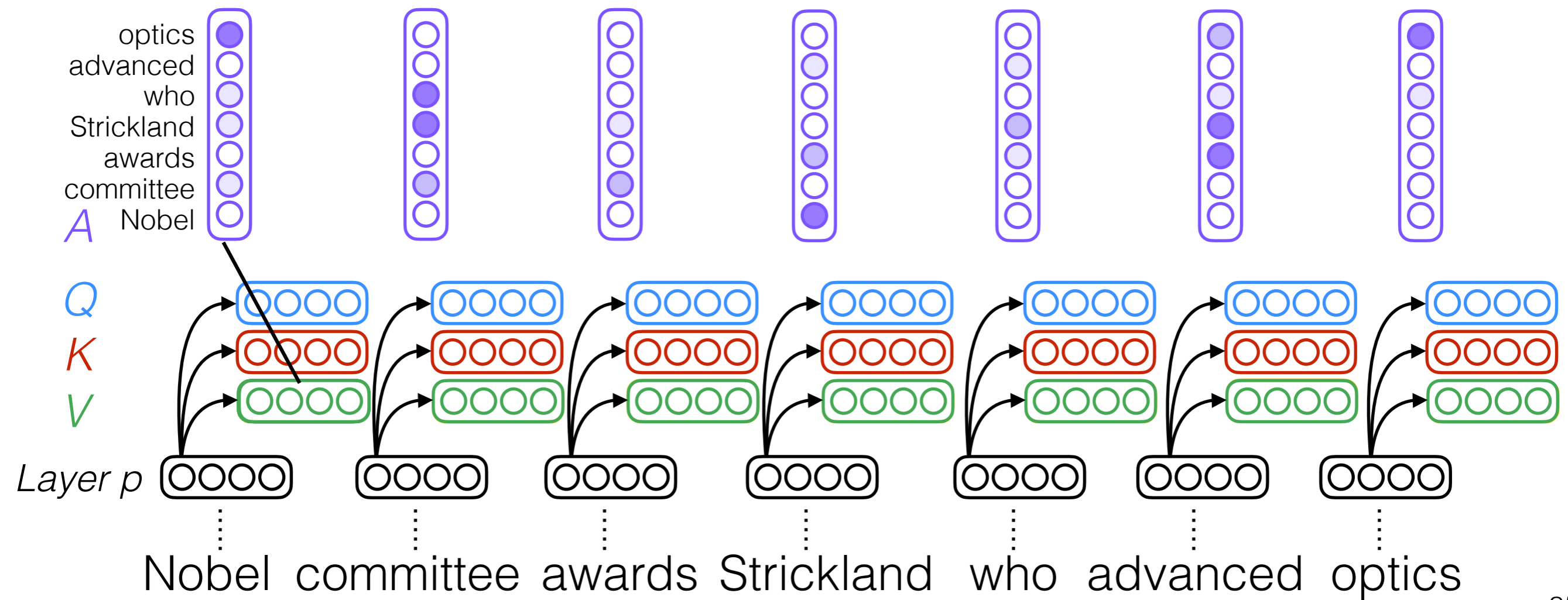
# Self-attention



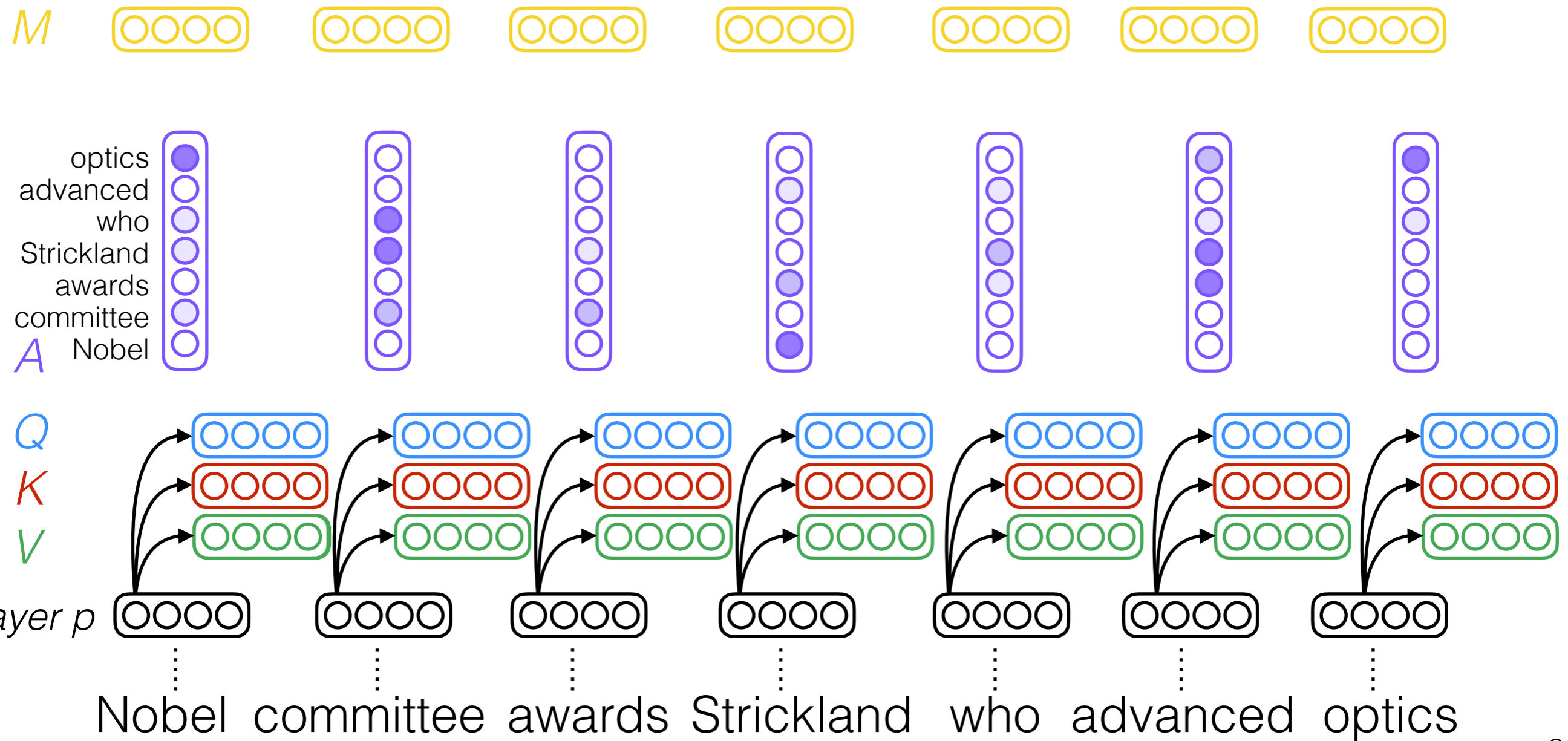
# Self-attention



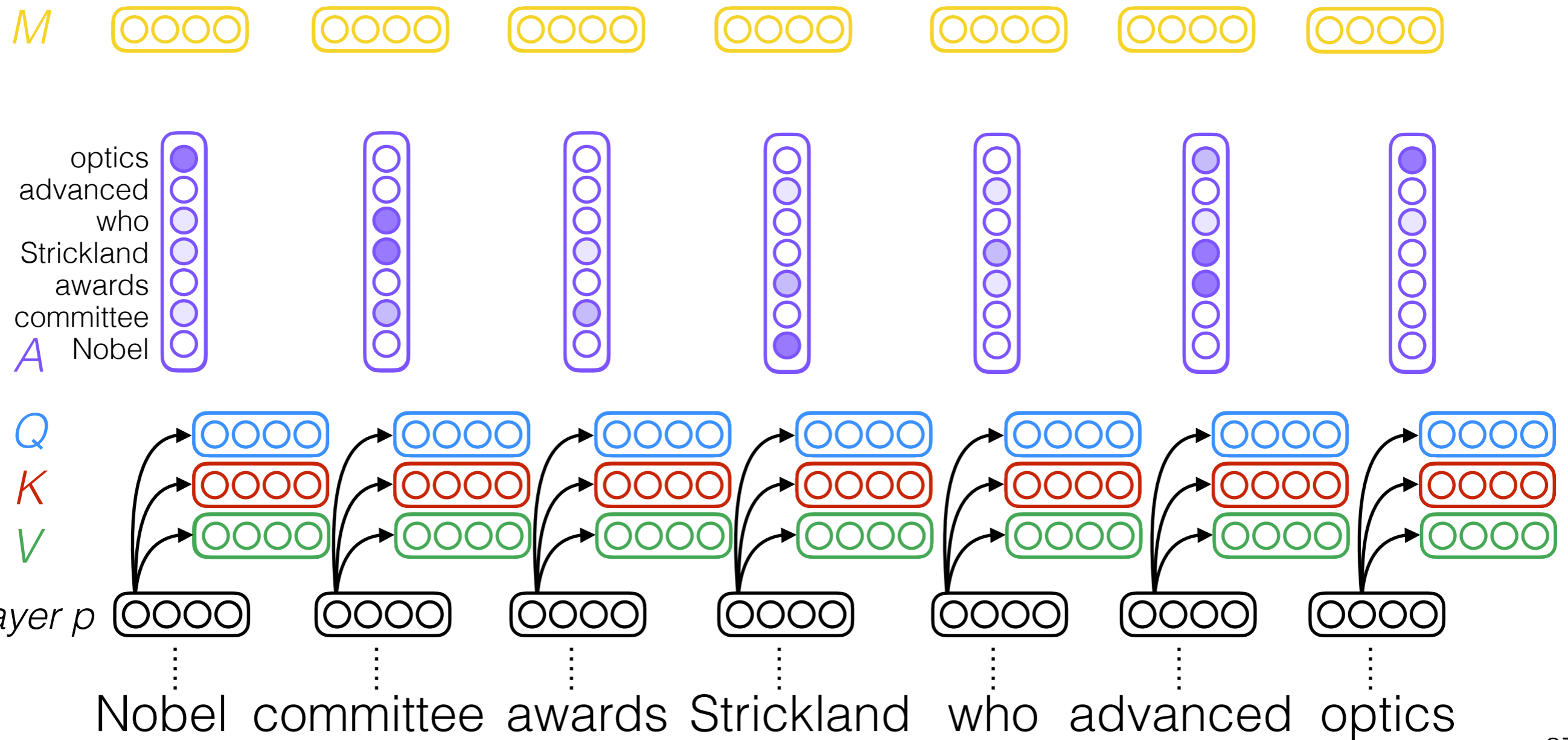
# Self-attention



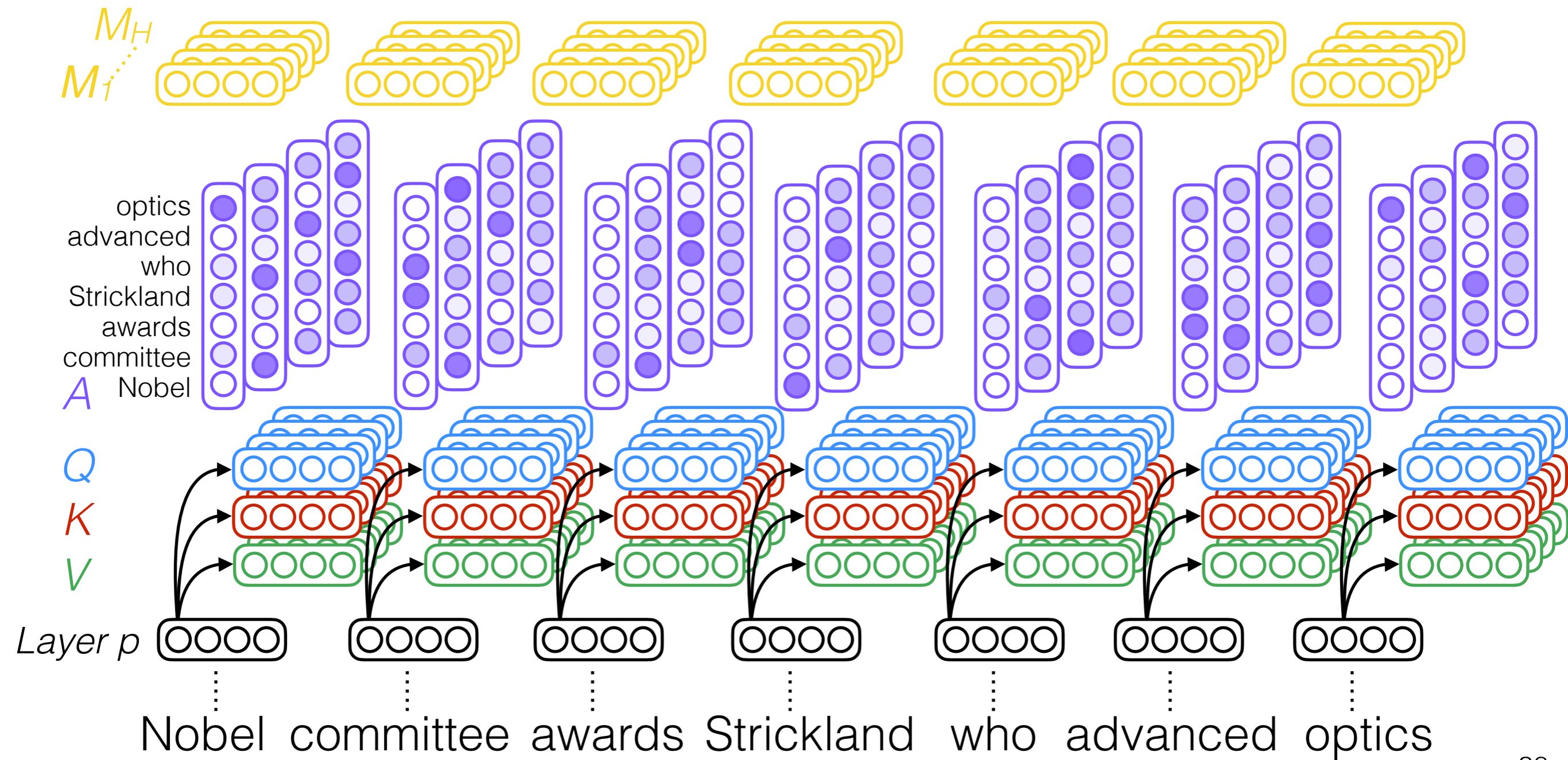
# Self-attention



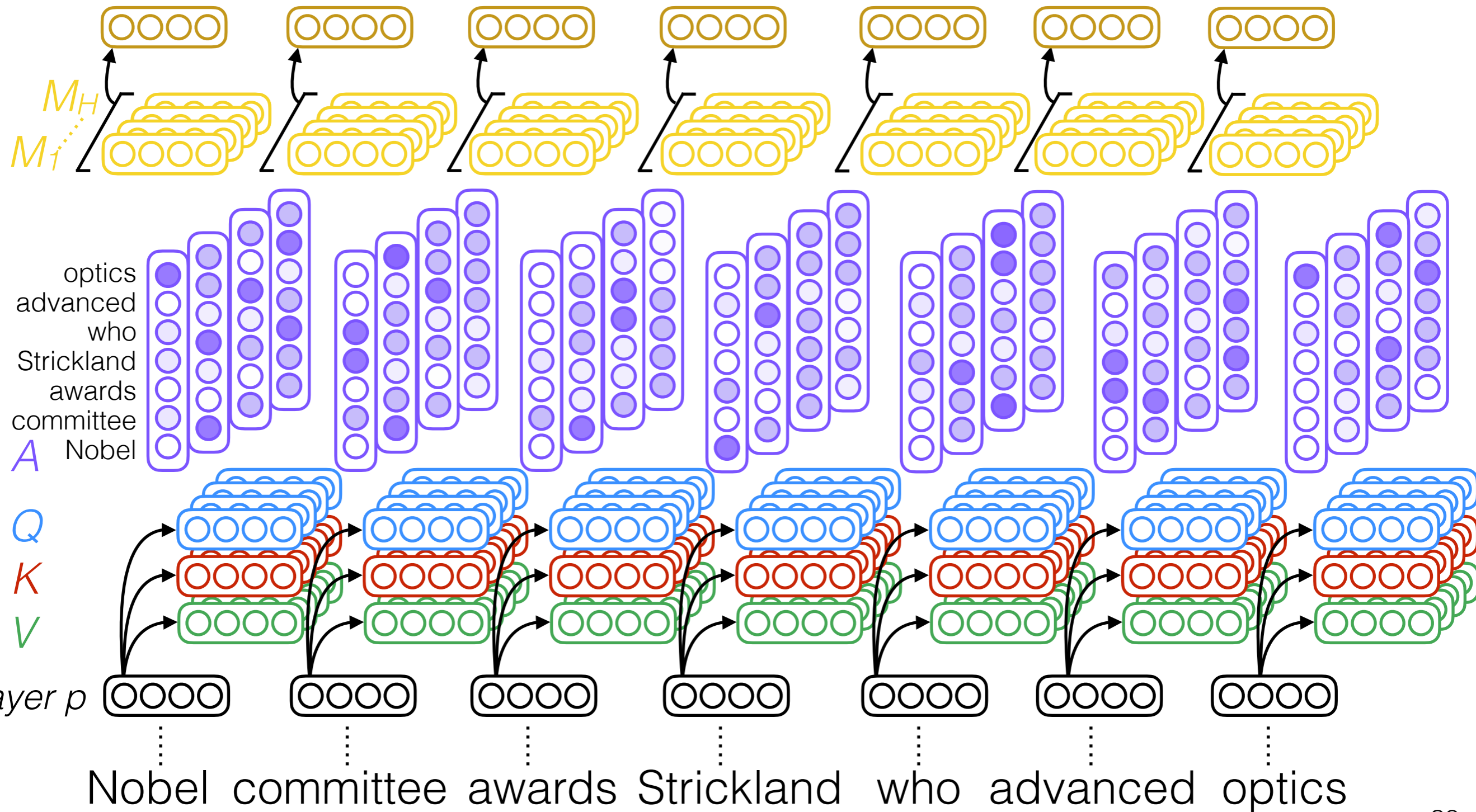
# Self-attention



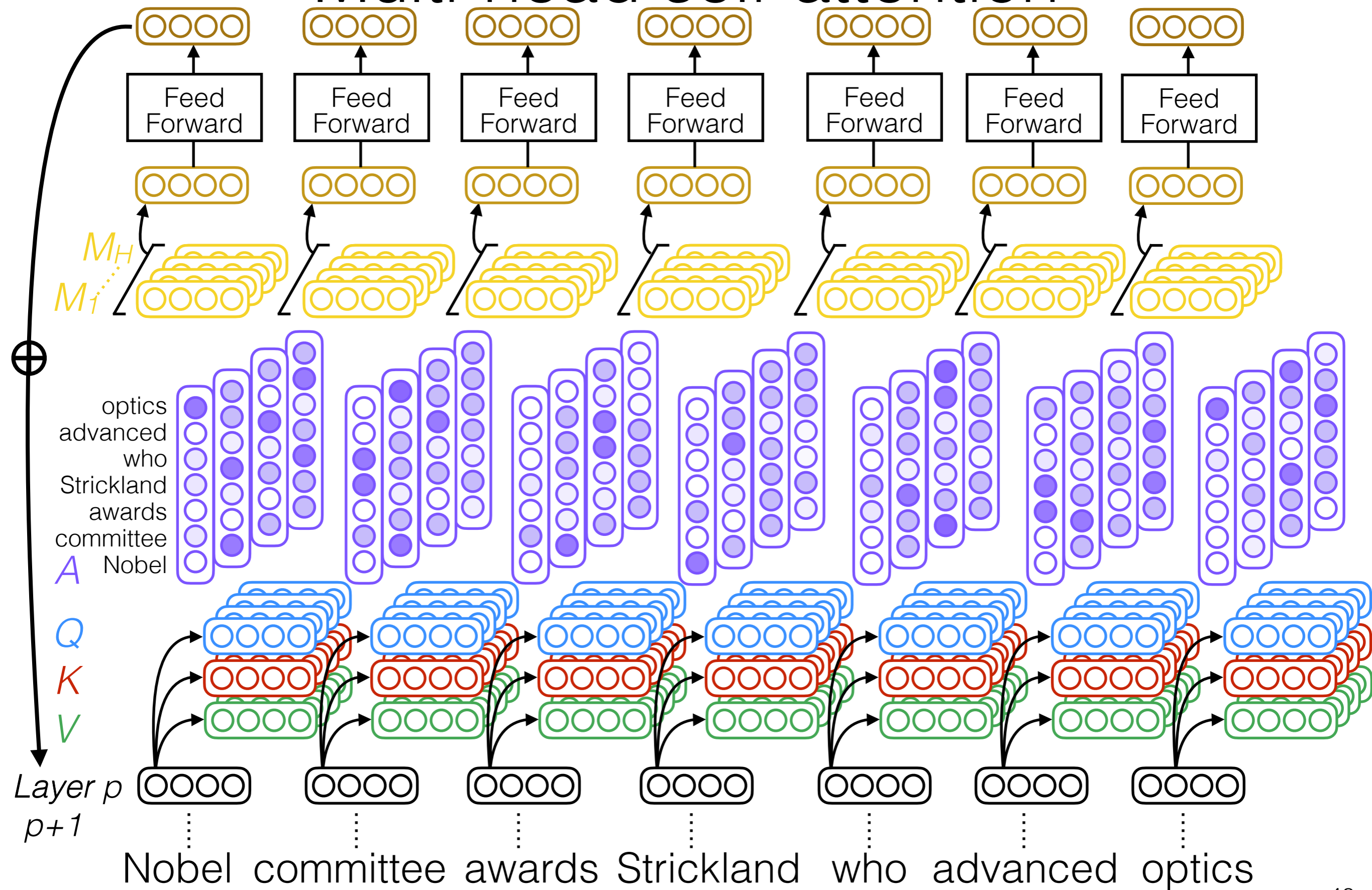
# Multi-head self-attention



# Multi-head self-attention

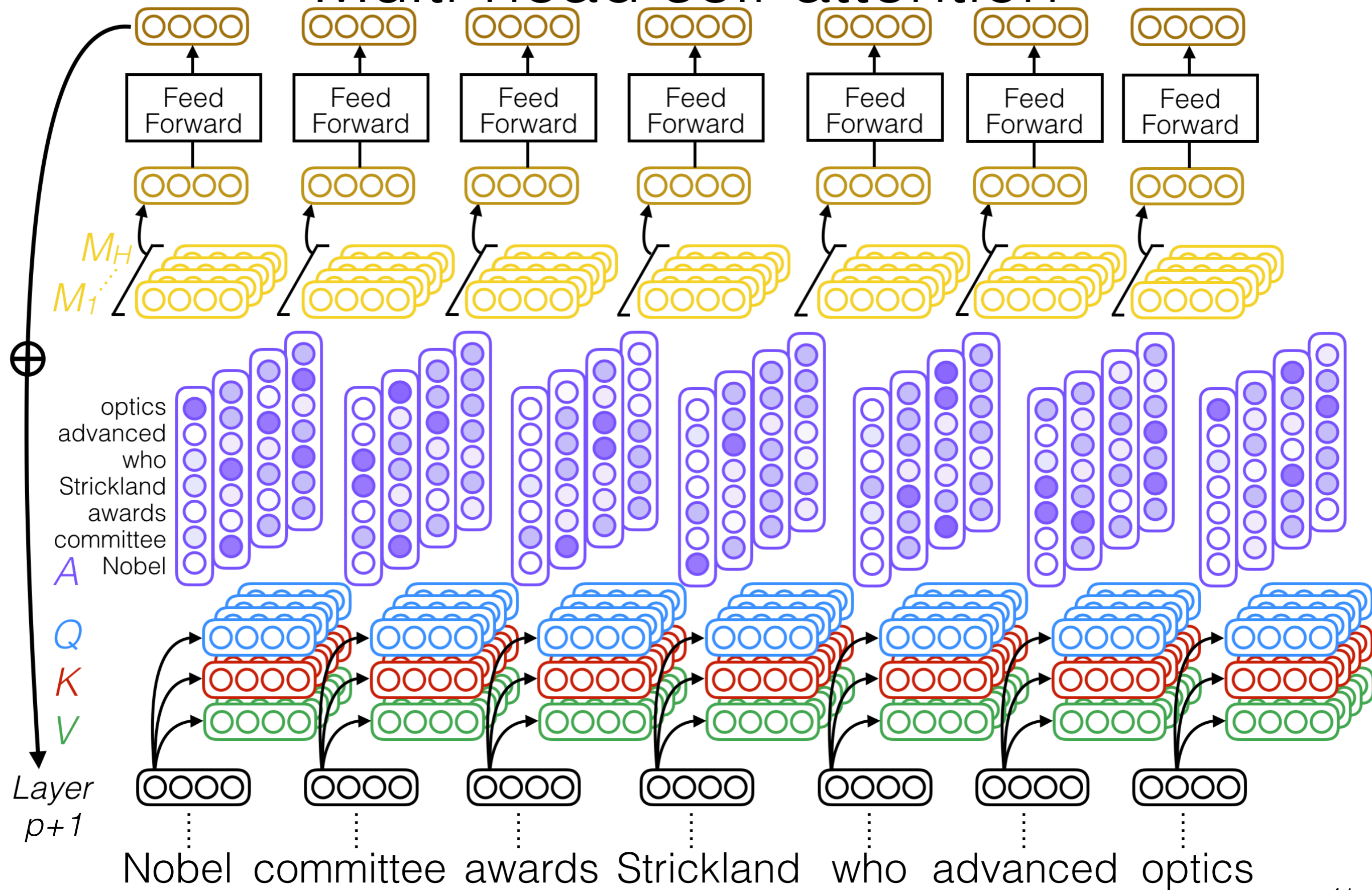


# Multi-head self-attention





# Multi-head self-attention



# Multi-head self-attention

