# TRANSFORMERS AND GRAPH NEURAL NETWORKS

AKSHAT GUPTA

# TRANSFORMERS

# CONTENT

- Introduction to Transformers
- Transformers Background
- The Attention Mechanism
- The Transformer Architecture
- GPT and BERT

# INTRODUCTION TO TRANSFORMERS

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $\mathbf{3.3 \cdot 10^{18}}$ | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |

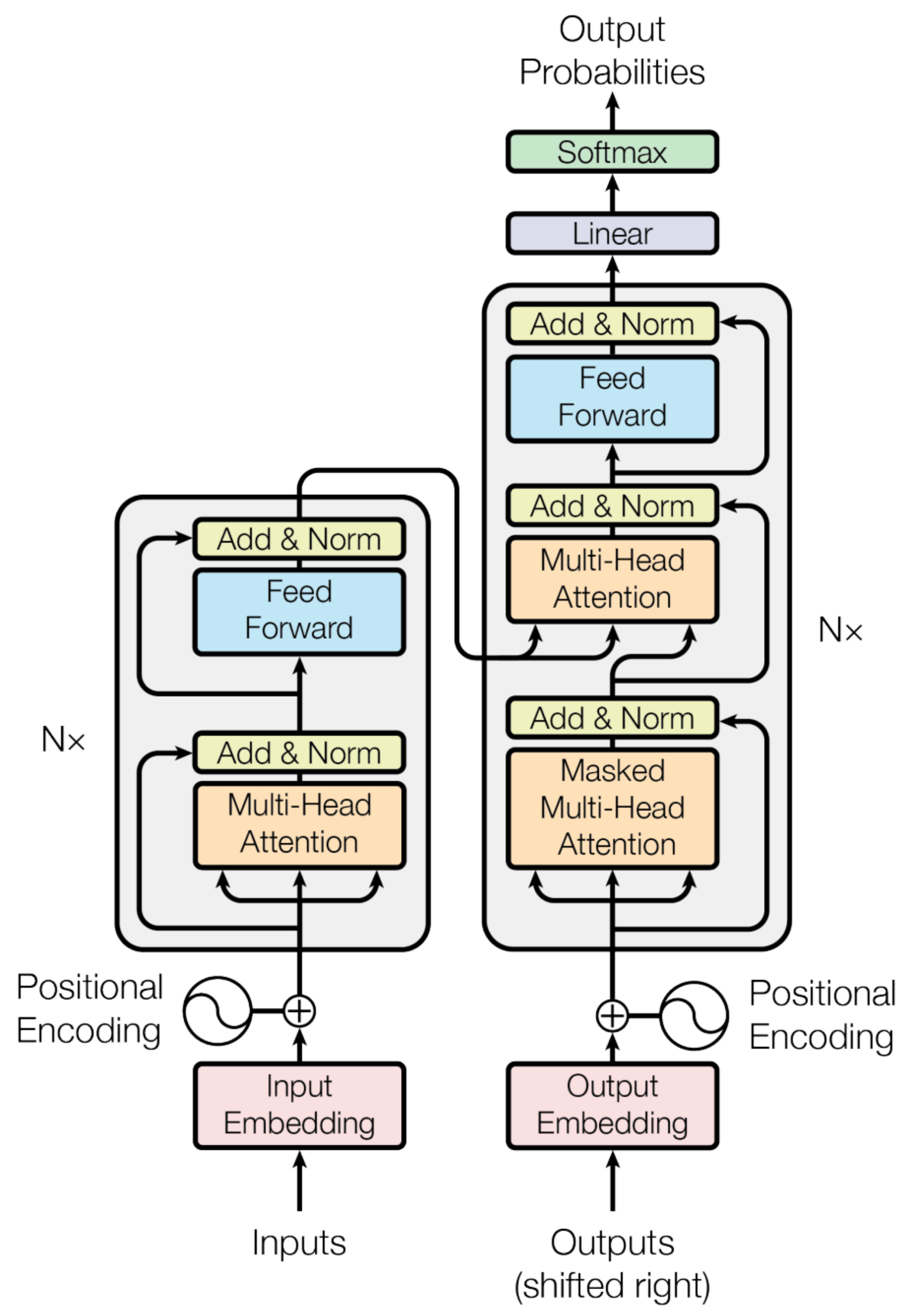
# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $\mathbf{3.3 \cdot 10^{18}}$ | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |



# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI:**
**Improving Language Understanding**
**by Generative Pre-Training**

June 11, 2018

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI:**
**Improving Language Understanding**
**by Generative Pre-Training**

June 11, 2018

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

| Method | Avg. Score | CoLA (mc) | SST2 (acc) | MRPC (F1) | STSB (pc) | QQP (F1) | MNLI (acc) | QNLI (acc) | RTE (acc) |
|---|---|---|---|---|---|---|---|---|---|
| Transformer w/ aux LM (full) | 74.7 | 45.4 | 91.3 | 82.3 | 82.0 | **70.3** | **81.8** | **88.1** | **56.0** |
| Transformer w/o pre-training | 59.9 | 18.9 | 84.0 | 79.4 | 30.9 | 65.5 | 75.7 | 71.2 | 53.8 |
| Transformer w/o aux LM | **75.0** | **47.9** | **92.0** | **84.9** | **83.2** | 69.8 | 81.1 | 86.9 | 54.4 |
| LSTM w/ aux LM | 69.1 | 30.3 | 90.5 | 83.2 | 71.8 | 68.1 | 73.7 | 81.1 | 54.6 |

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper : Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI: Improving Language Understanding by Generative Pre-Training**

June 11, 2018

**First BERT paper by Google: Pre-Training Deep Bidirectional Transformers for Language Understanding**

Oct 11, 2018

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper : Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI: Improving Language Understanding by Generative Pre-Training**

June 11, 2018

**First BERT paper by Google: Pre-Training Deep Bidirectional Transformers for Language Understanding**

Oct 11, 2018

| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

# TRANSFORMERS : BACKGROUND

# TRANSFORMERS : BACKGROUND

- Word Embeddings
- Encoder Decoder Models
- Attention

# TRANSFORMERS : BACKGROUND

- **Word Embeddings**
- Encoder Decoder Models
- Attention

# TRANSFORMERS : BACKGROUND

- Word Embeddings

Representing words in the form of vectors
that incorporate information such as word meaning and context.

# TRANSFORMERS : BACKGROUND

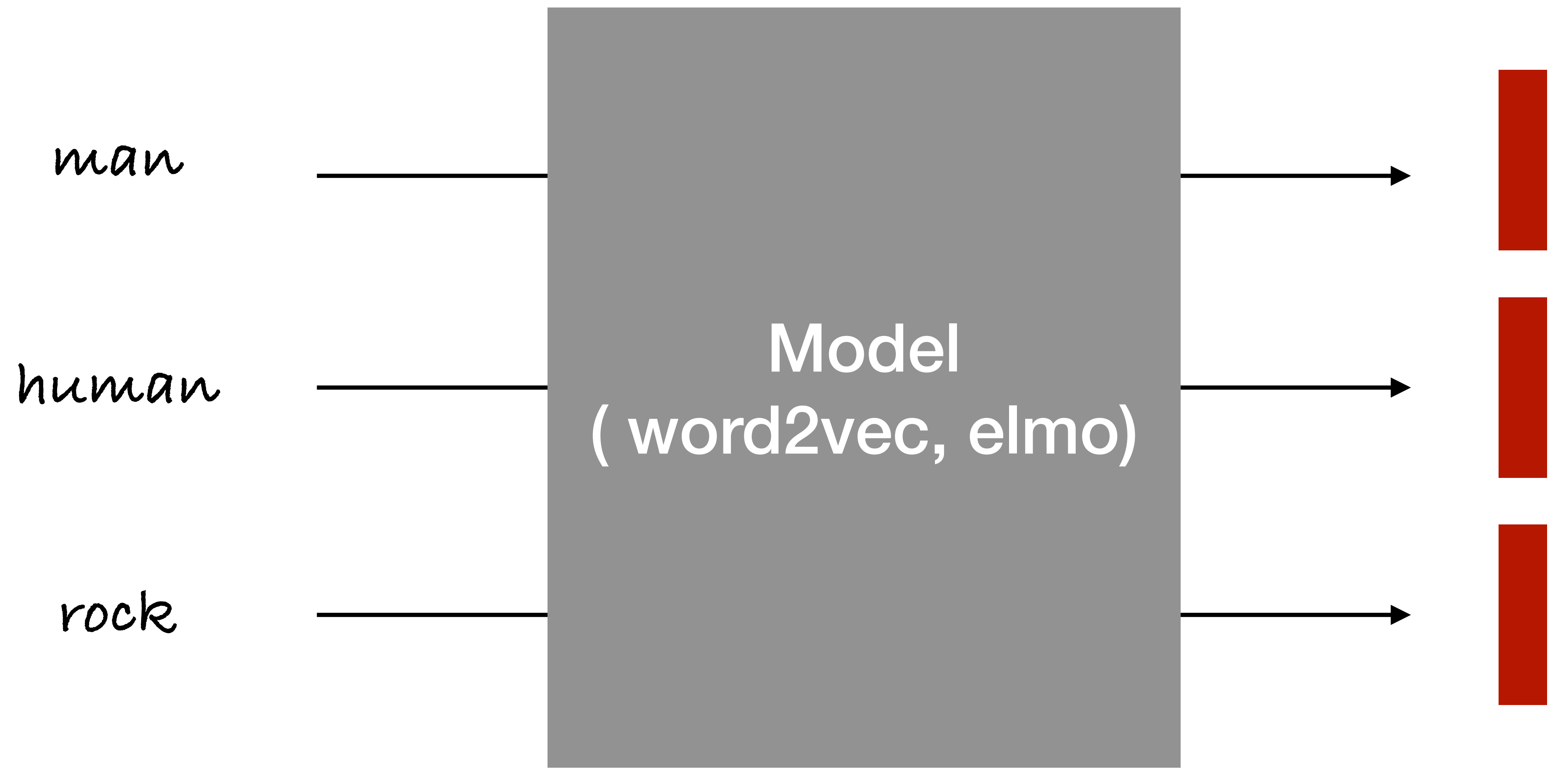
- Word Embeddings

Representing words in the form of vectors
that incorporate information such as word meaning and context.

# TRANSFORMERS : BACKGROUND

- Word Embeddings

Representing words in the form of vectors
that incorporate information such as word meaning and context.



# TRANSFORMERS : BACKGROUND

- Word Embeddings
- **Encoder Decoder Models**
- Attention

# TRANSFORMERS : BACKGROUND

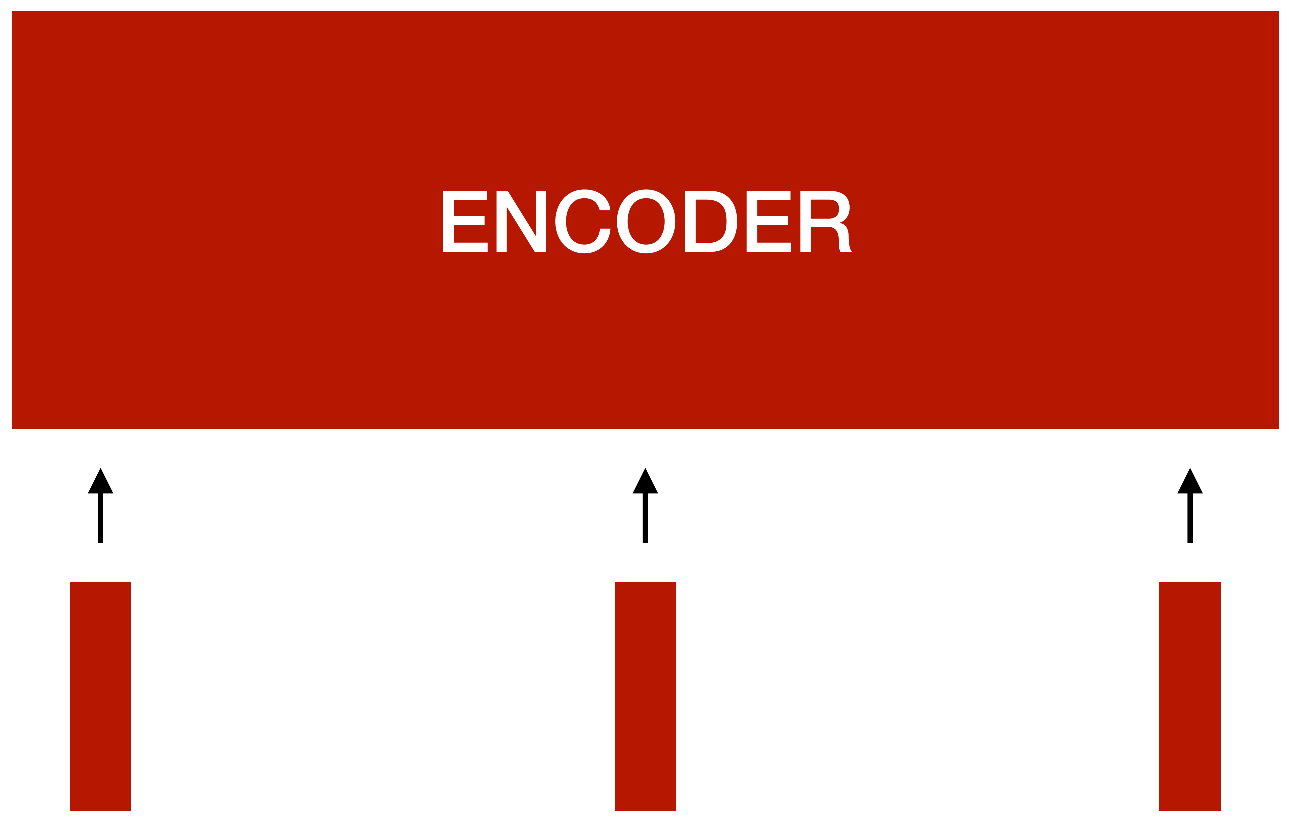
- Encoder Decoder Models

Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output

# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

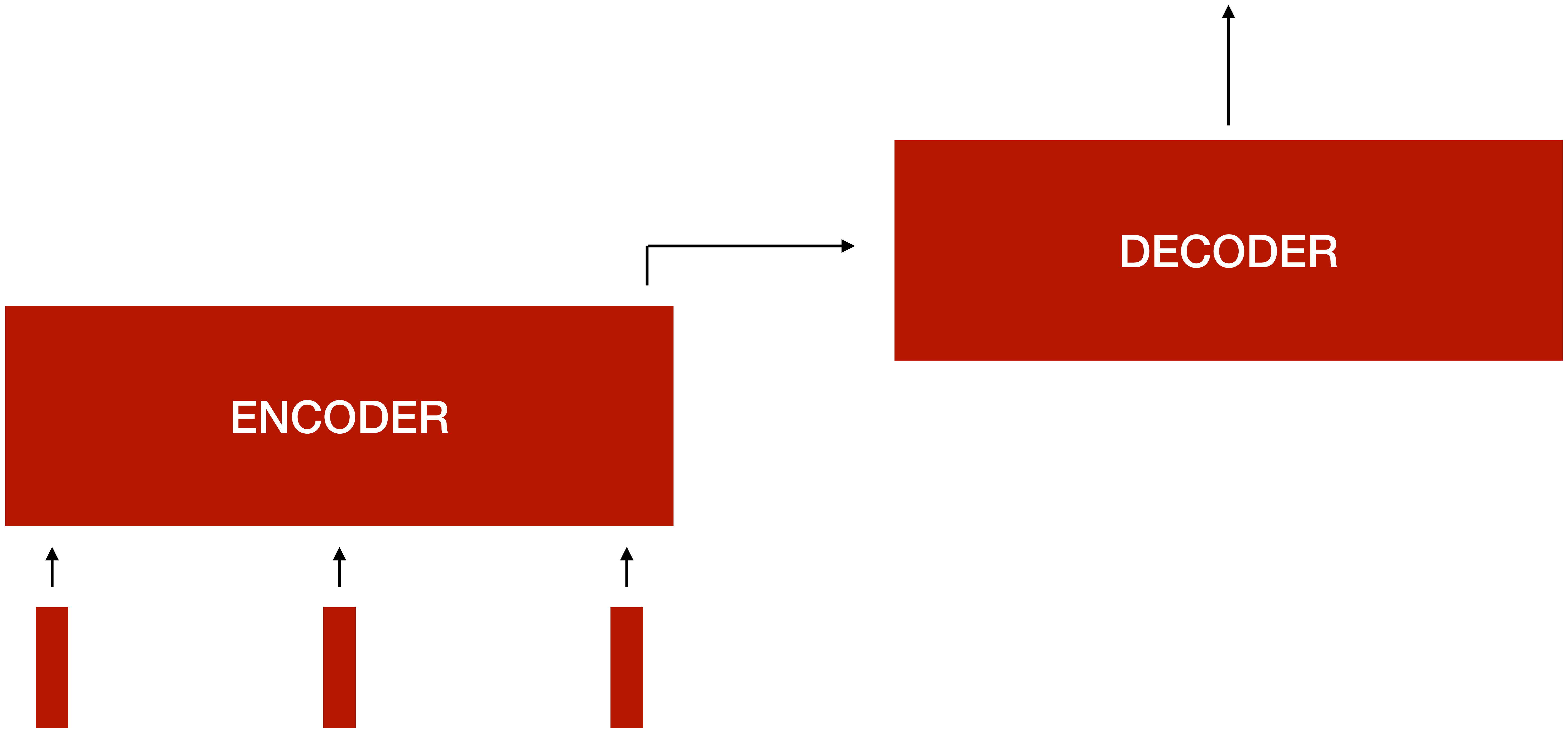
Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output



# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

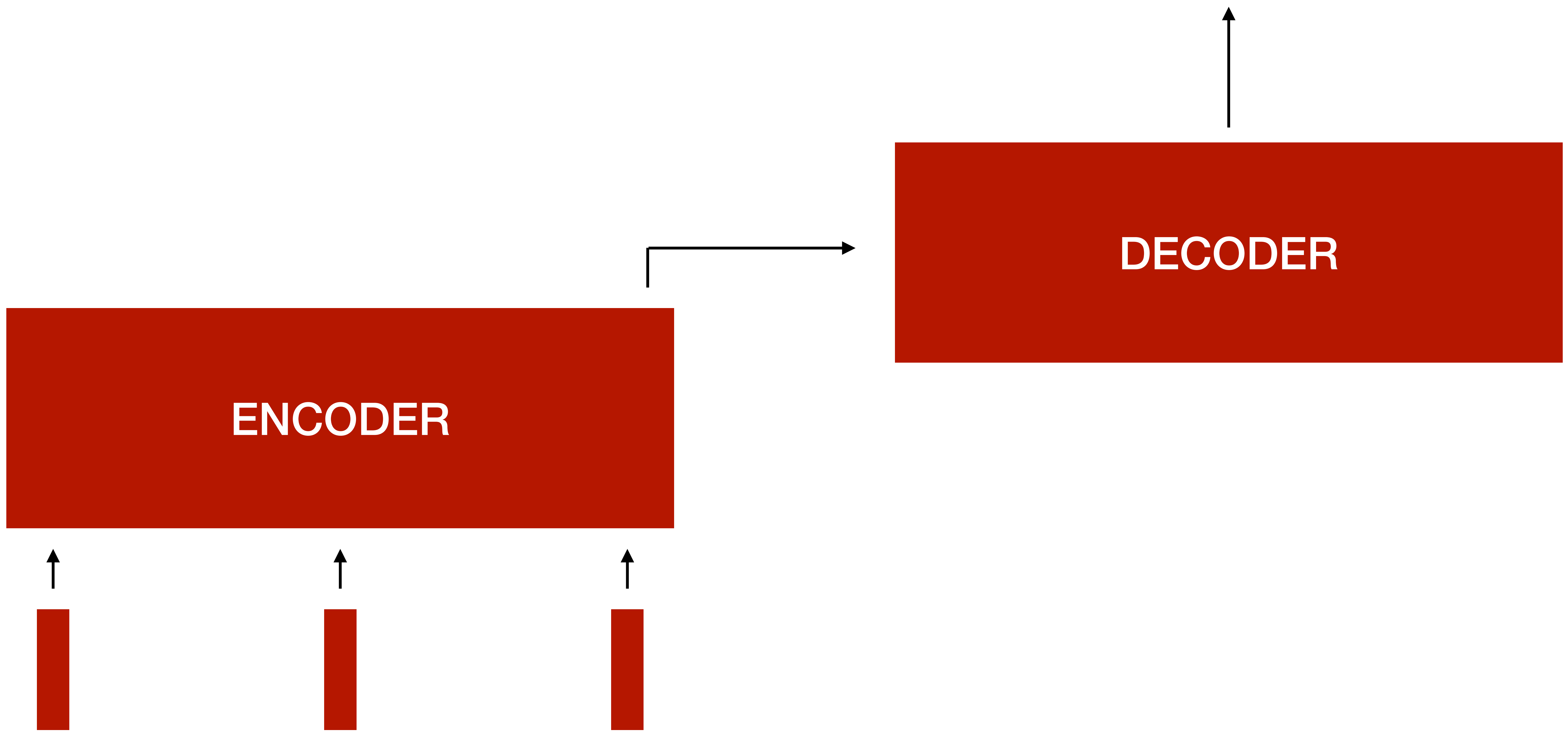
Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output



# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output




**See any problems here?**

# TRANSFORMERS : BACKGROUND

- Word Embeddings
- Encoder Decoder Models
- **Attention**

# TRANSFORMERS : BACKGROUND

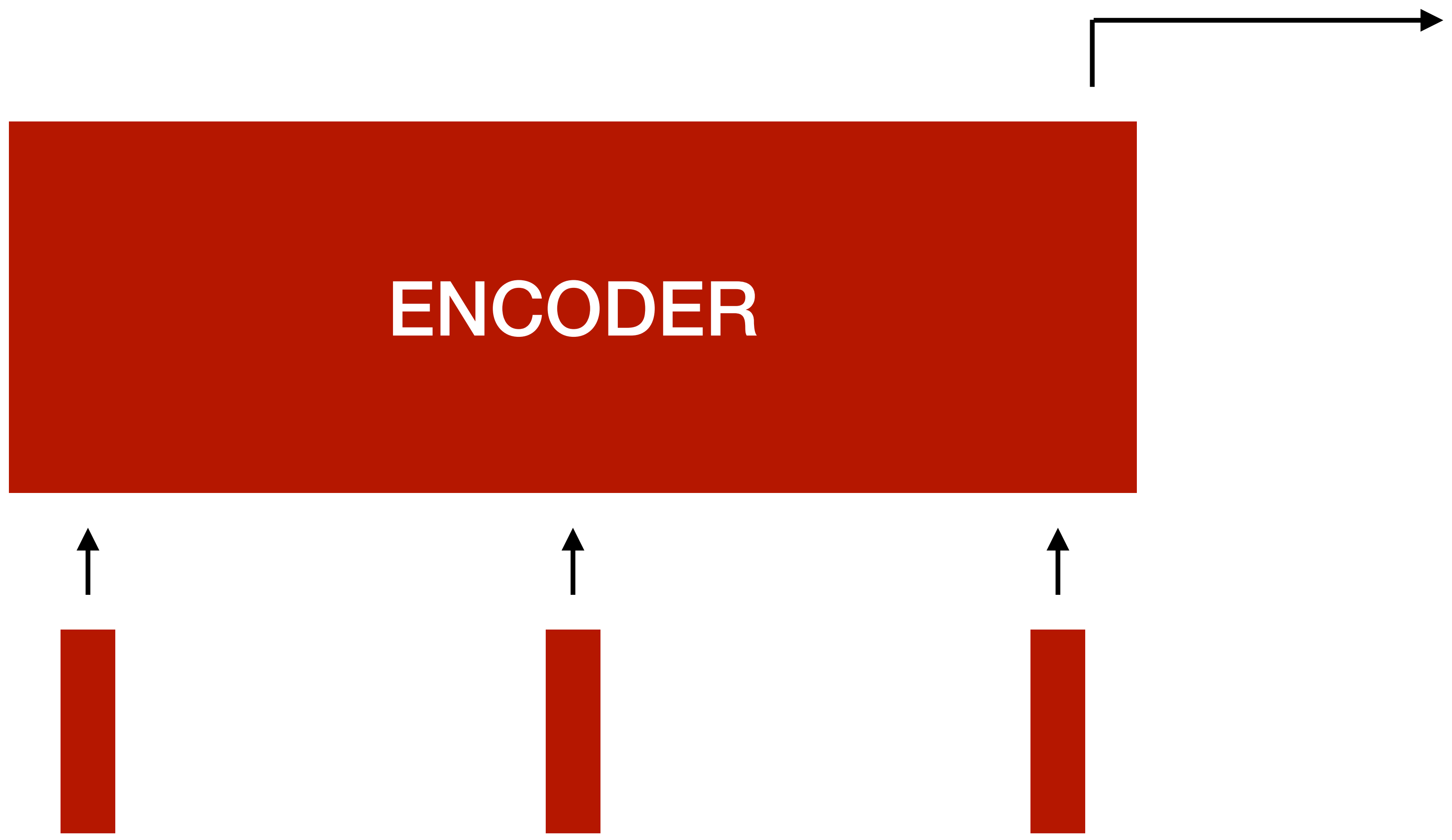
- Attention Mechanism

  A method of dynamically giving weight (attention) to different parts of the input to make a decision.

# TRANSFORMERS : BACKGROUND

- Attention Mechanism

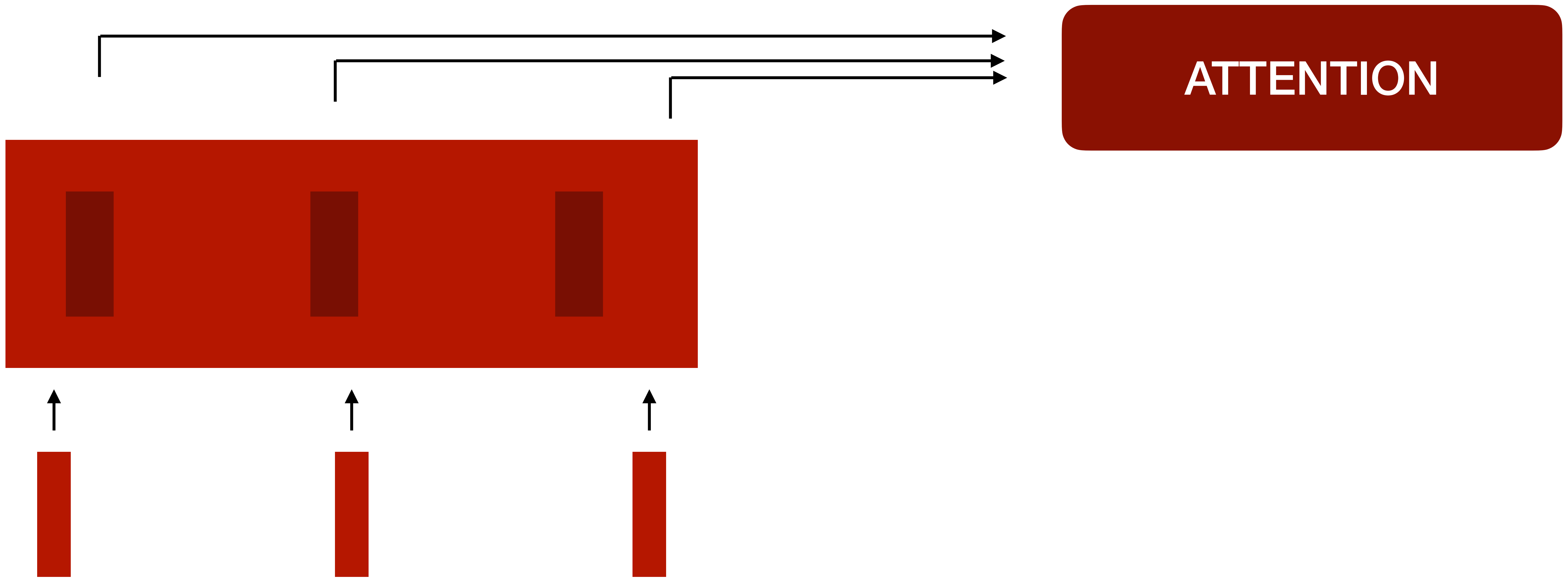
A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

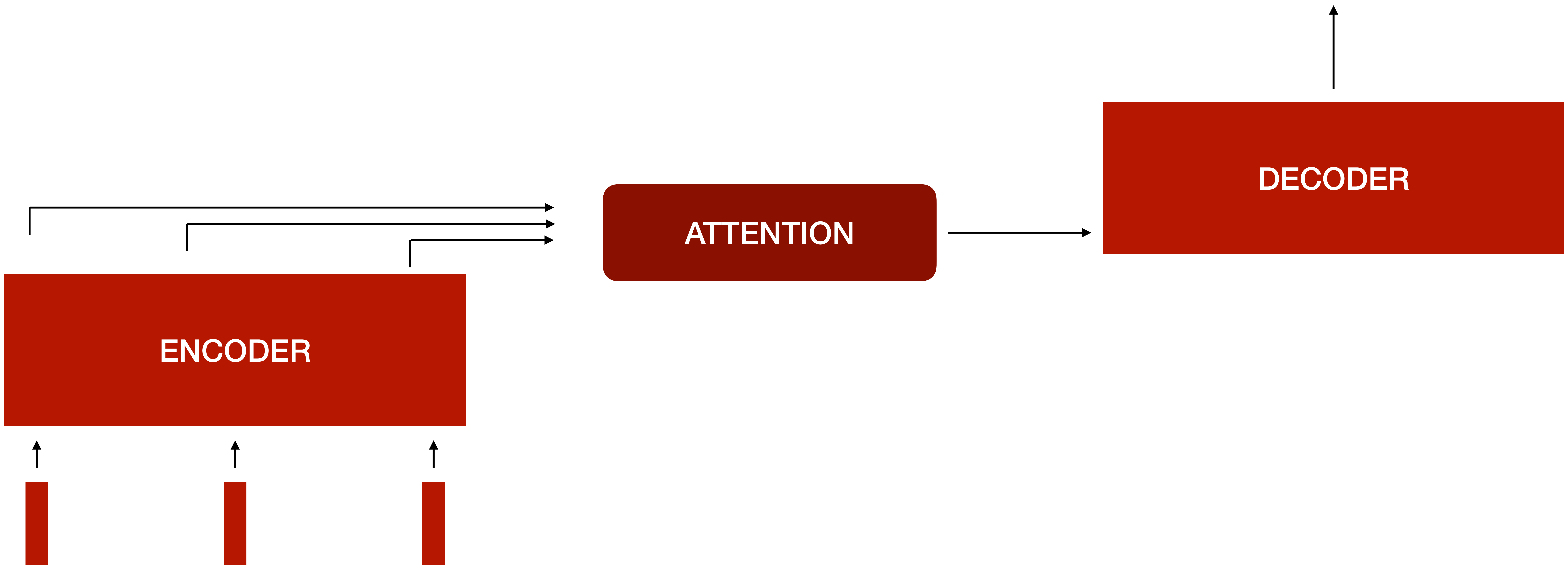
A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

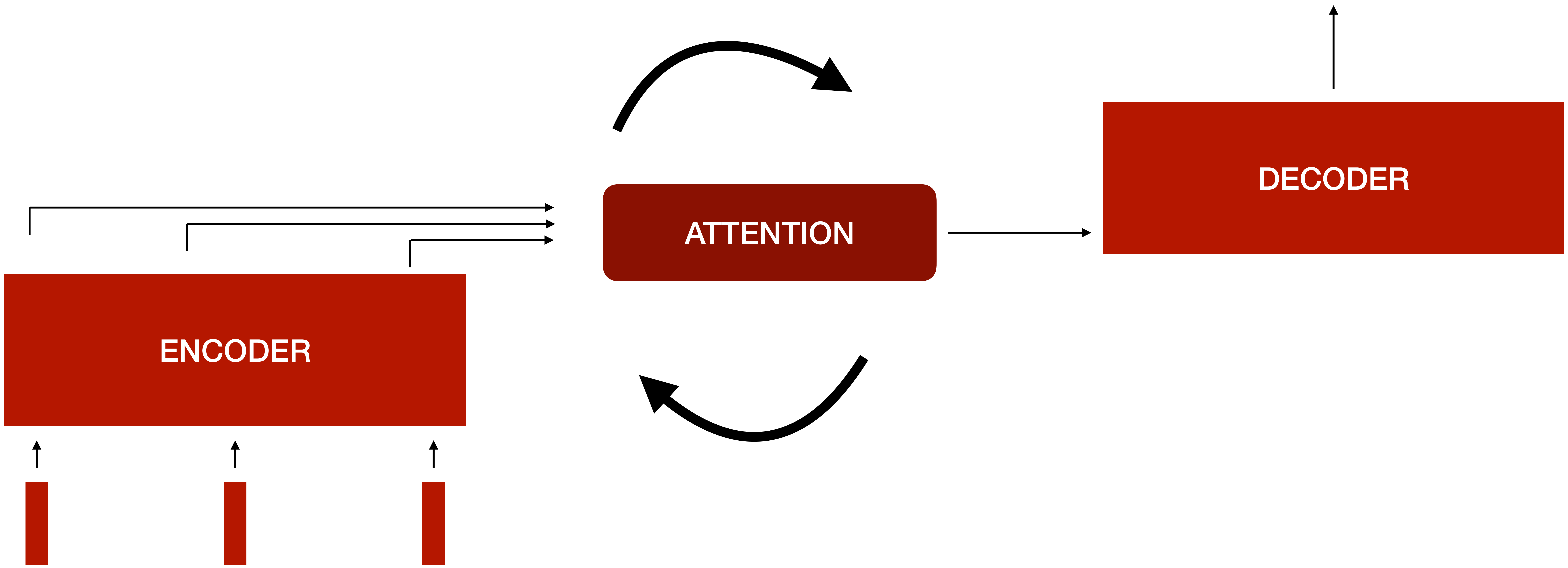
A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS :
# THE ATTENTION MECHANISM

# THE ATTENTION MECHANISM

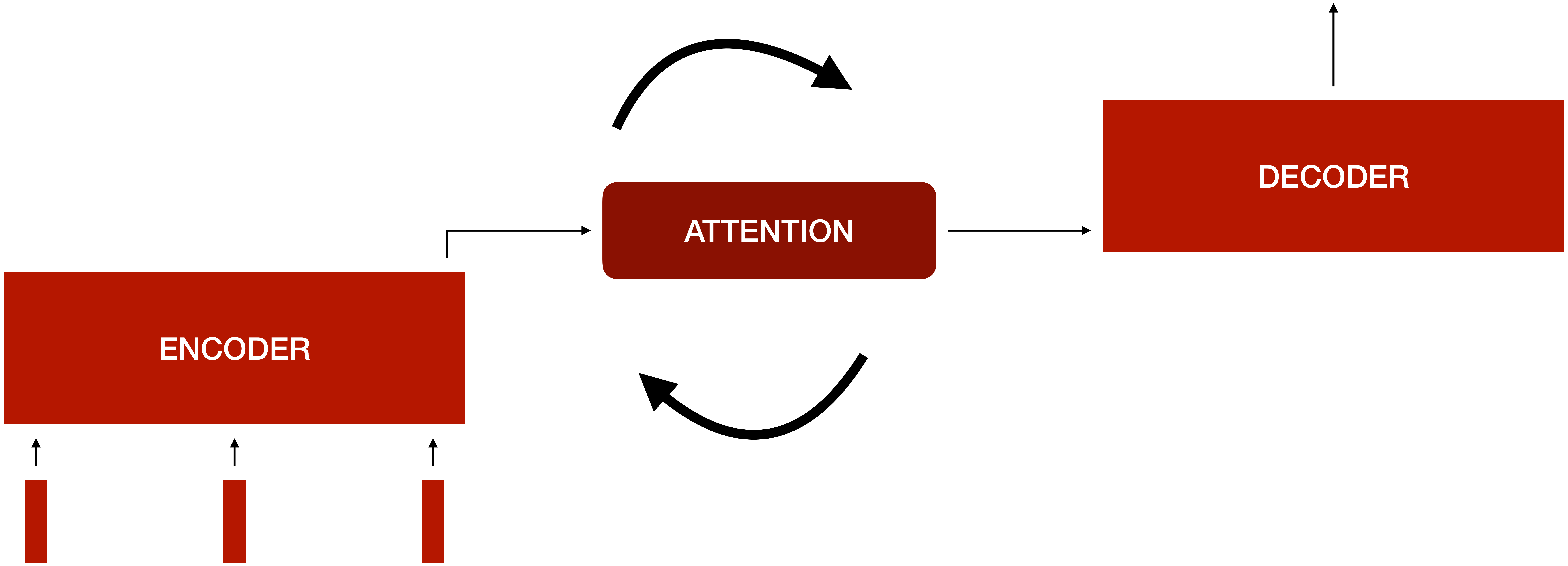
- Attention
- Self-Attention
- Multi-Head Attention

# THE ATTENTION MECHANISM

- **Attention**
- Self-Attention
- Multi-Head Attention

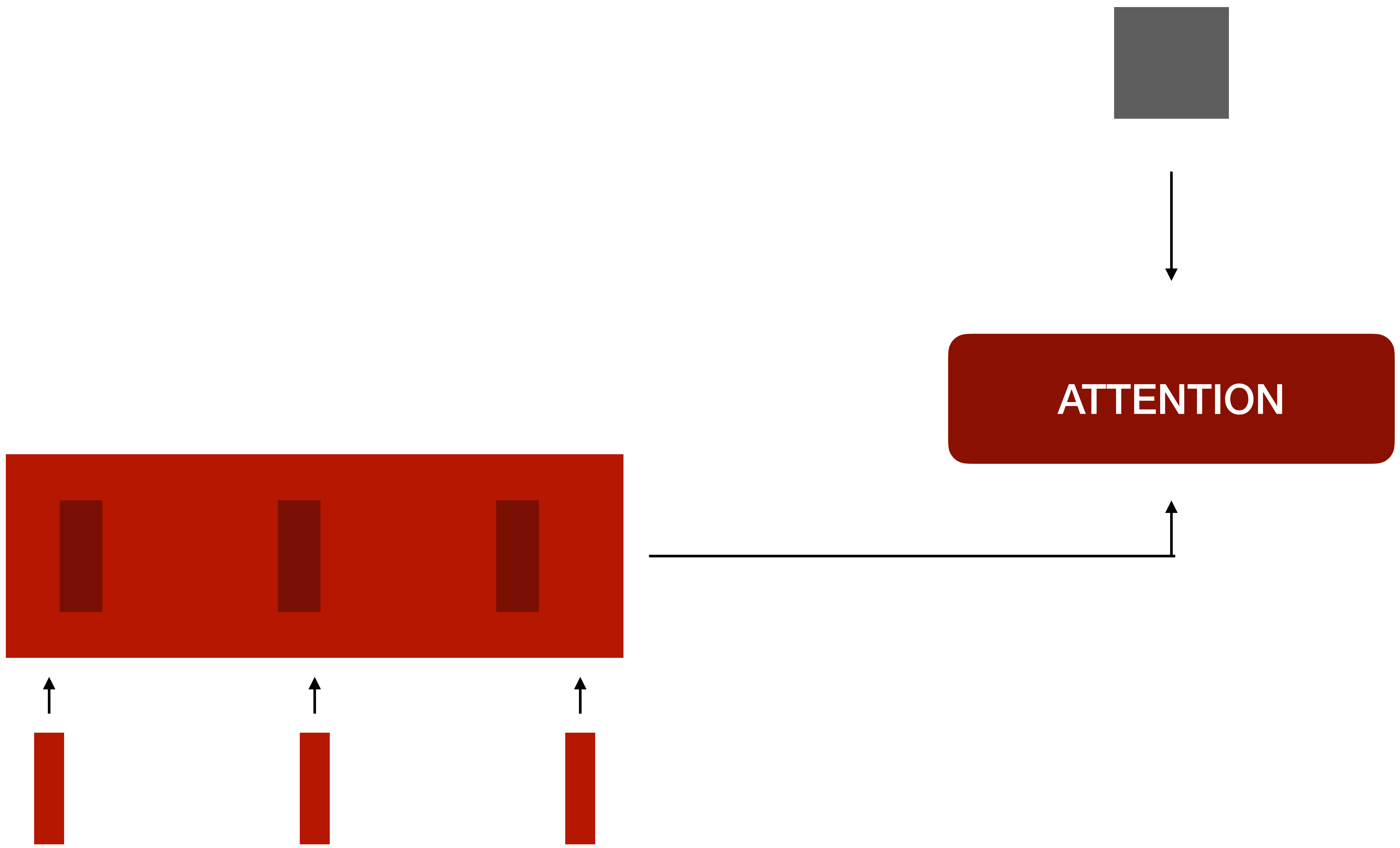
# THE ATTENTION MECHANISM

- **Attention**



# THE ATTENTION MECHANISM

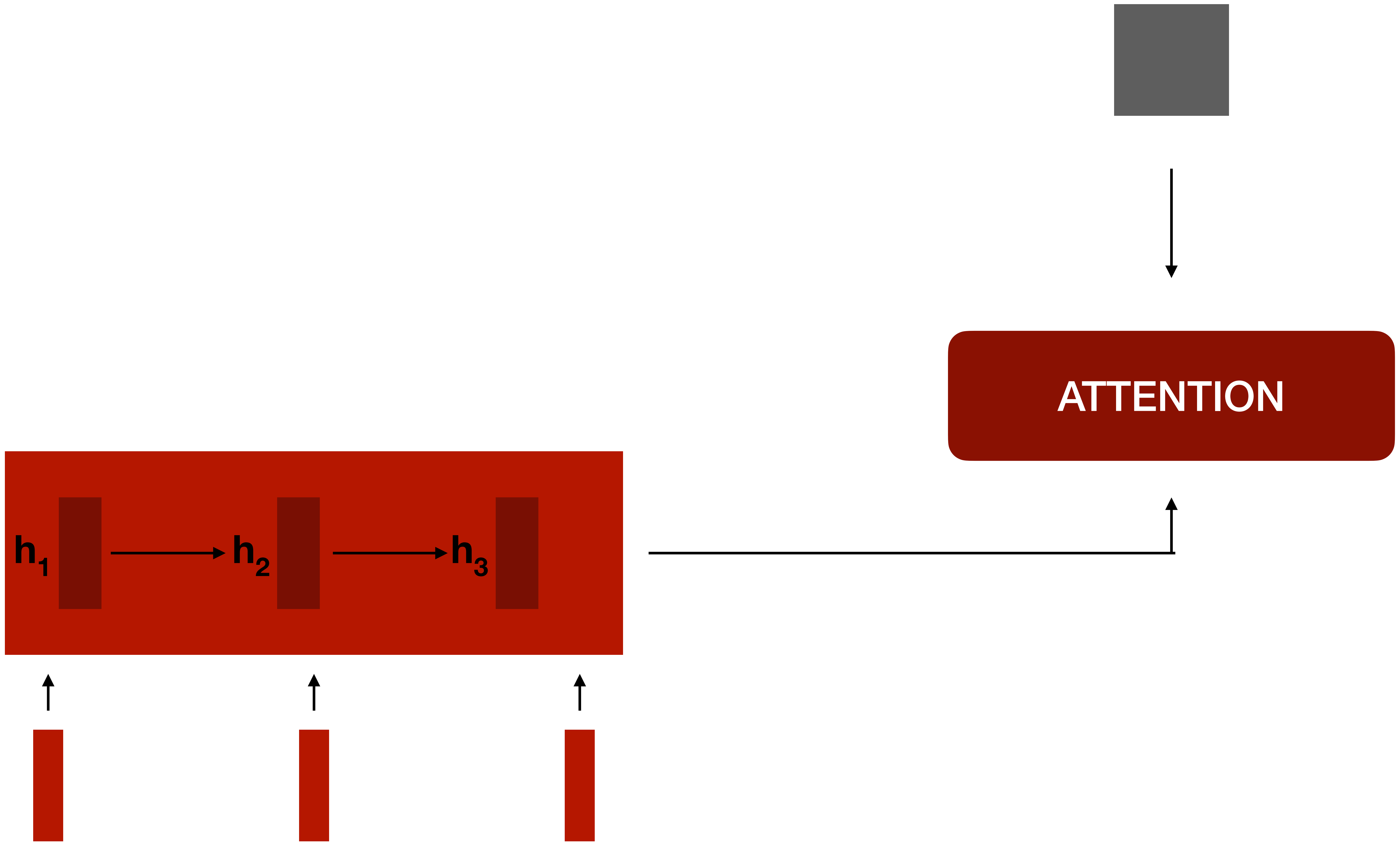
- **Attention**



# THE ATTENTION MECHANISM

- **Attention**

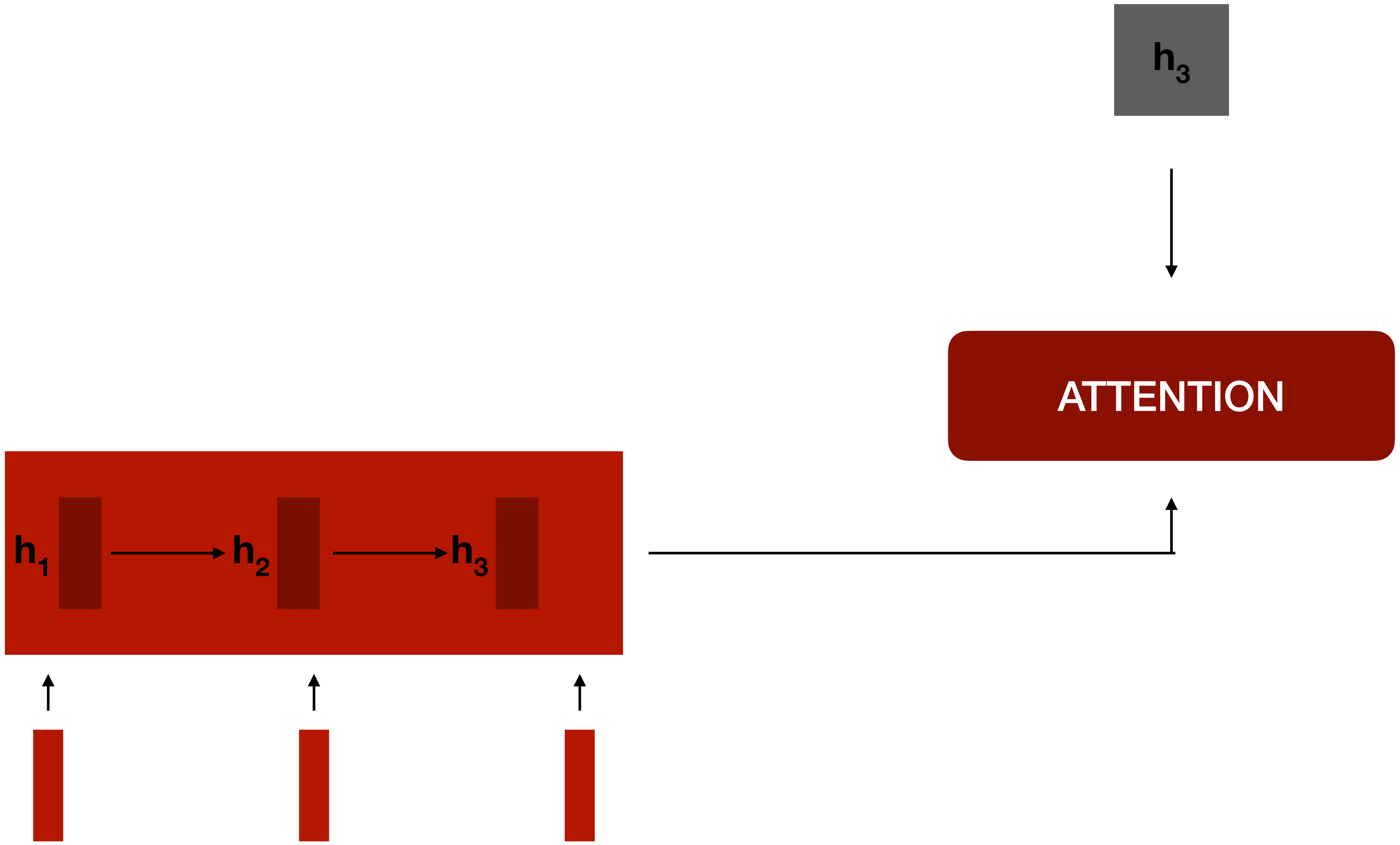
What is the other input to the attention module at the beginning of generation?



# THE ATTENTION MECHANISM

**At t = 0,**
**First time step of generation**

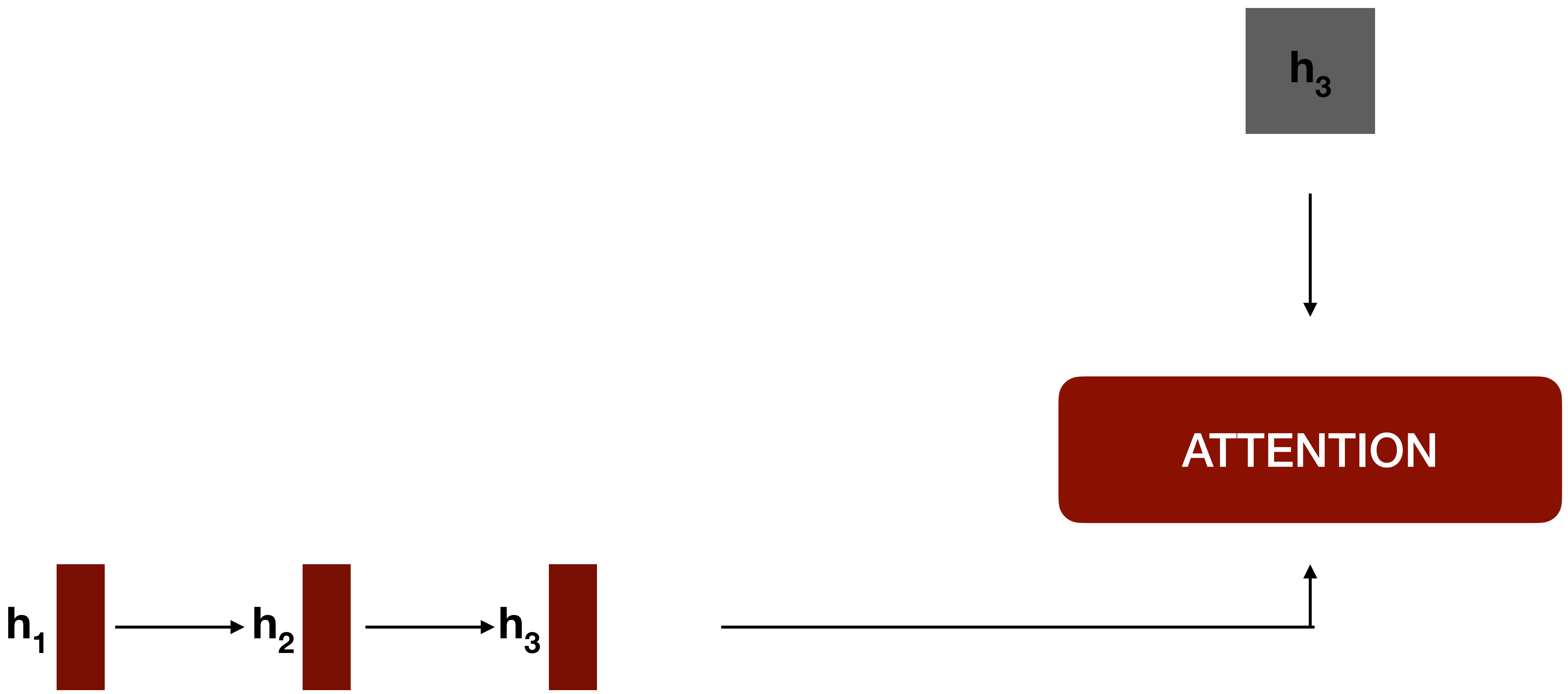
- **Attention**



# THE ATTENTION MECHANISM

At t = 0,
First time step of generation

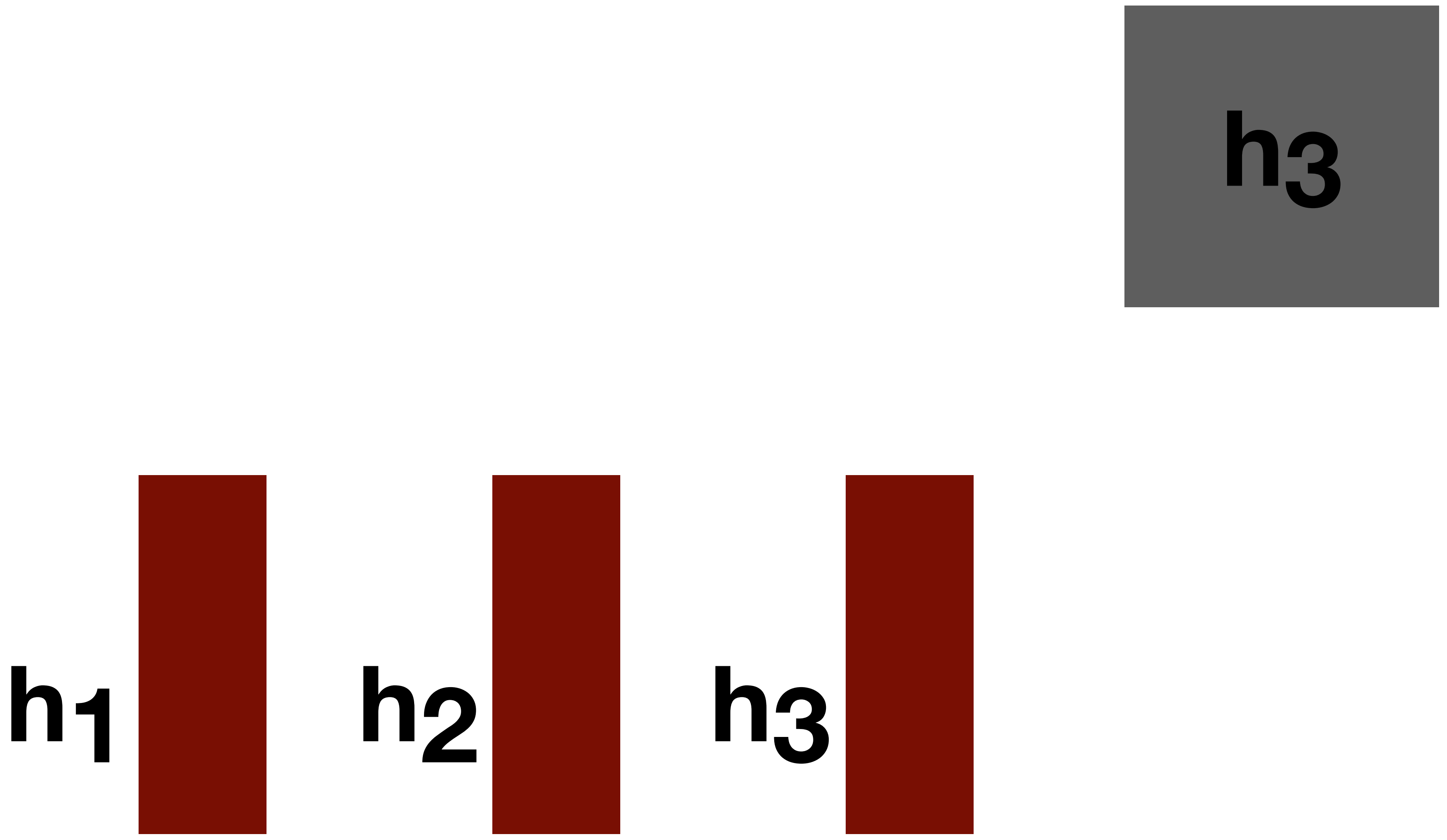
- **Attention**



# THE ATTENTION MECHANISM

**At t = 0,**
**First time step of generation**

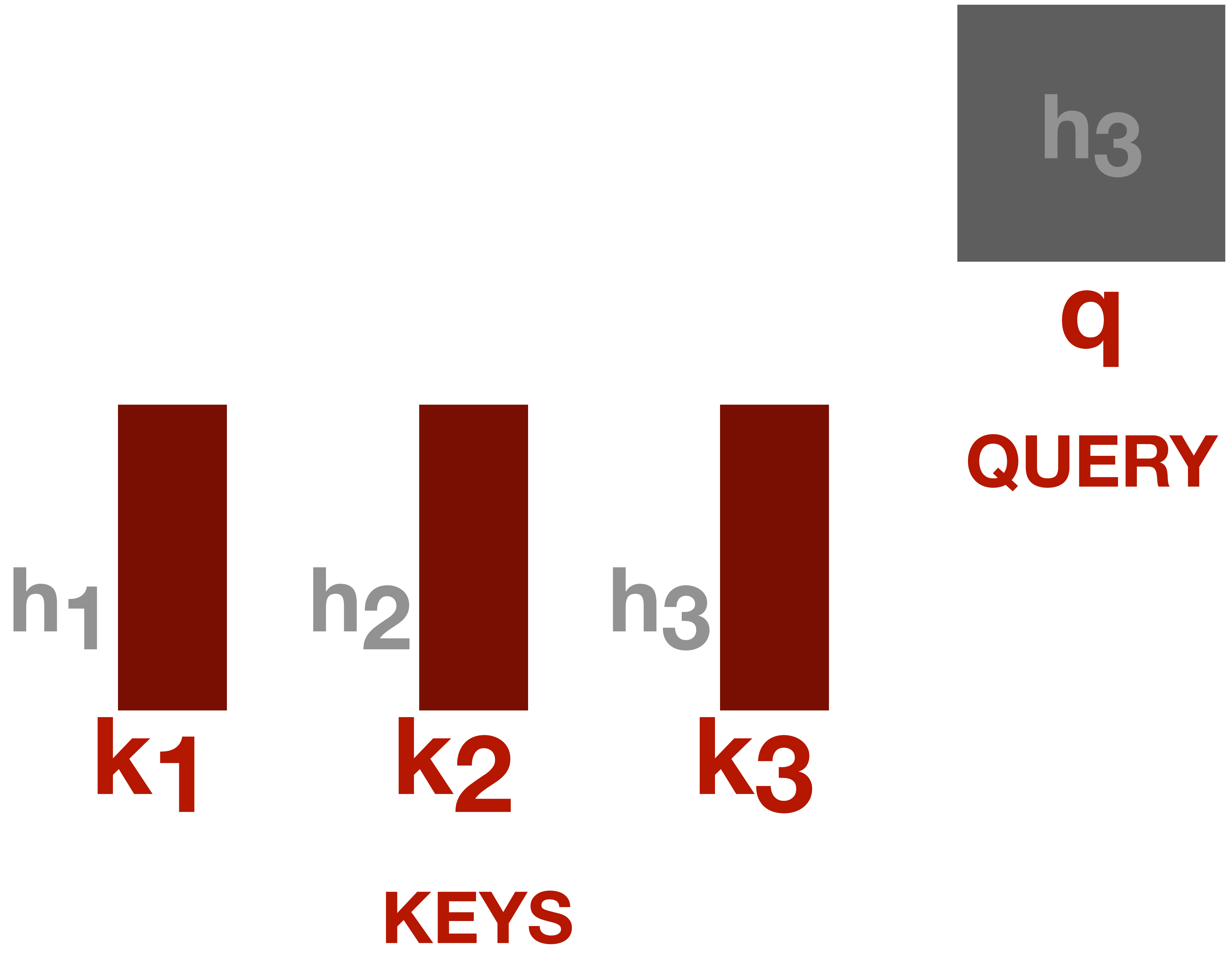
- **Attention : Set Up**



# THE ATTENTION MECHANISM

At t = 0,
First time step of generation

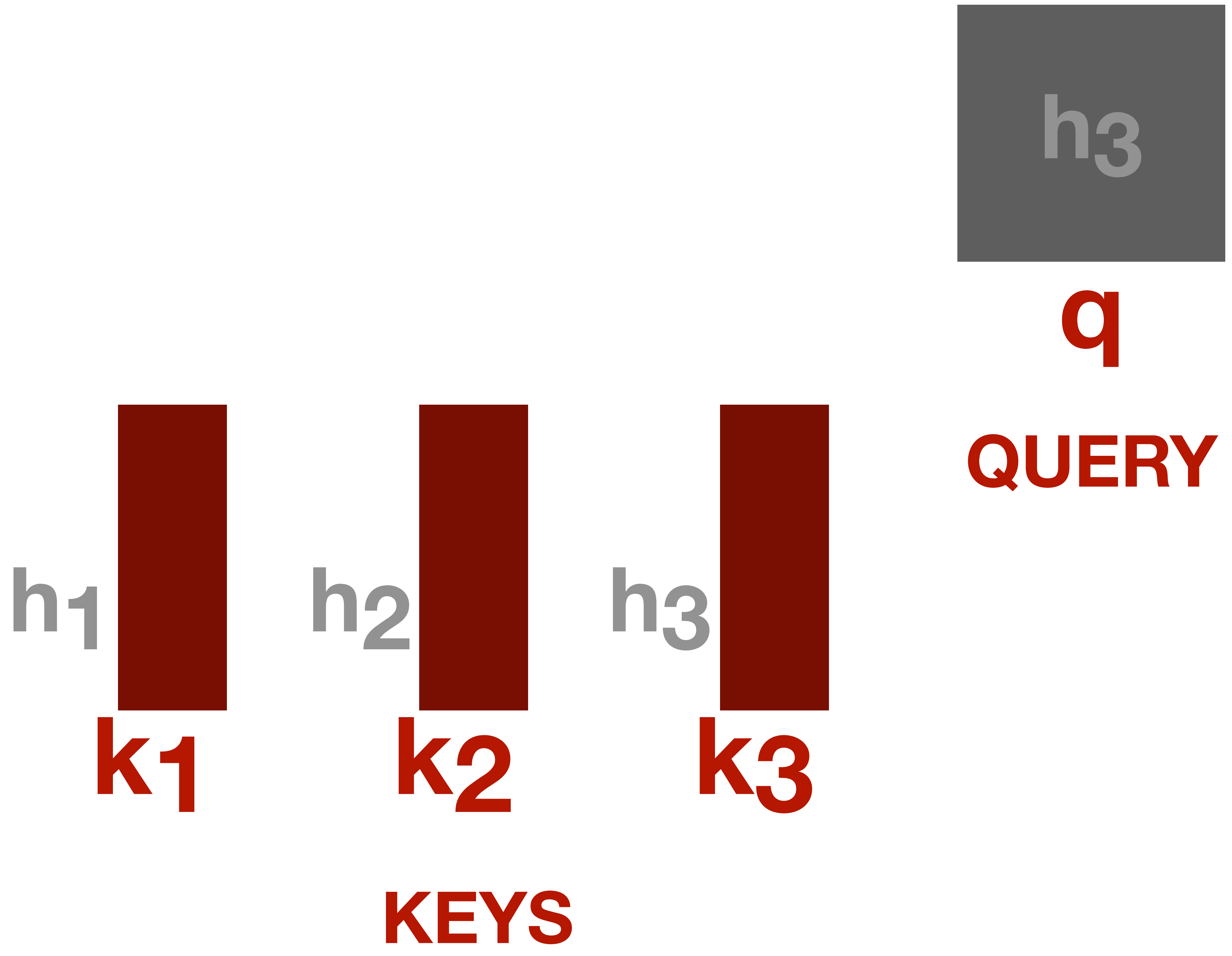
- **Attention : Set Up**



# THE ATTENTION MECHANISM

At t = 0,
First time step of generation

- **Calculating Attention**

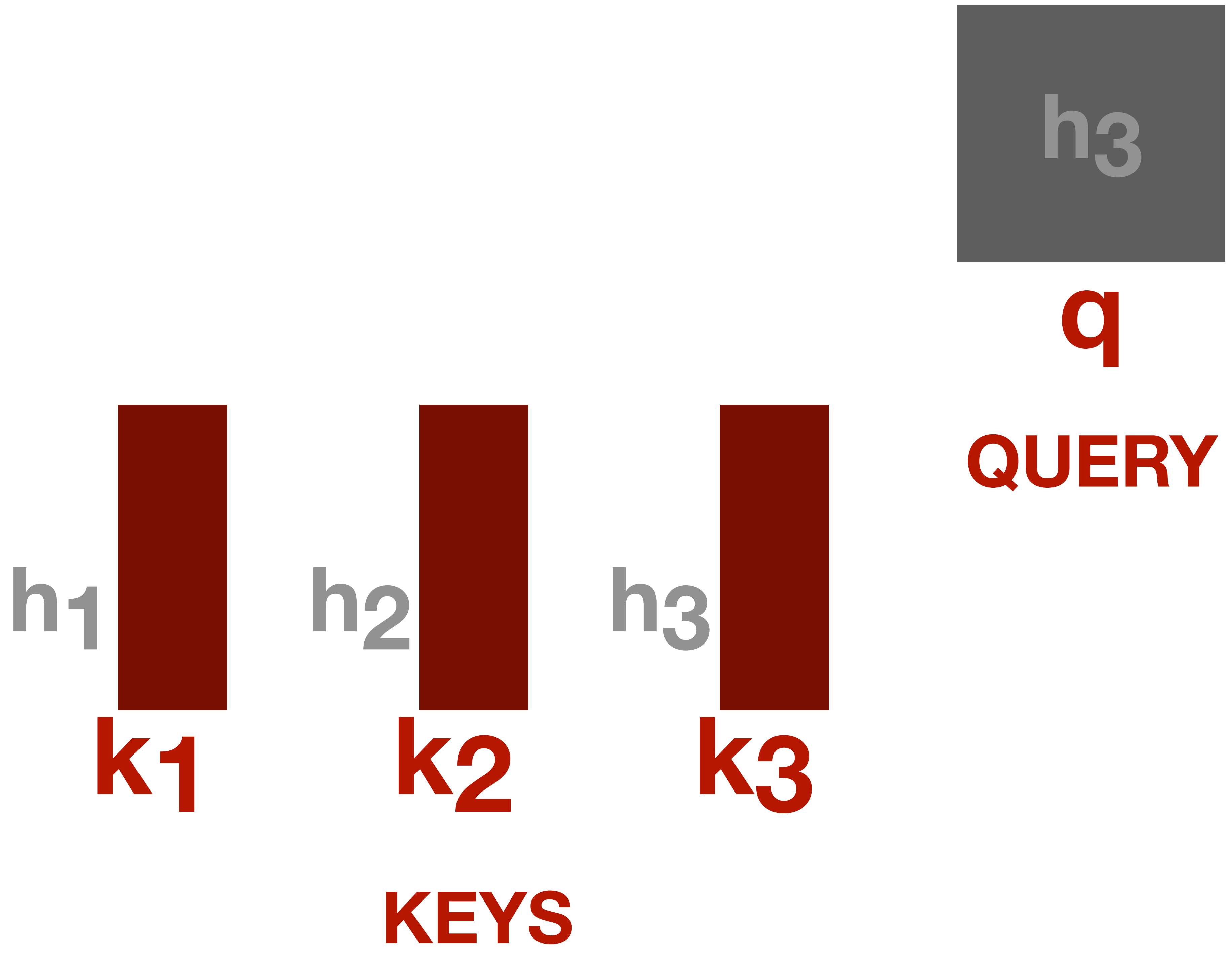


# THE ATTENTION MECHANISM

- **Calculating Attention**

STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

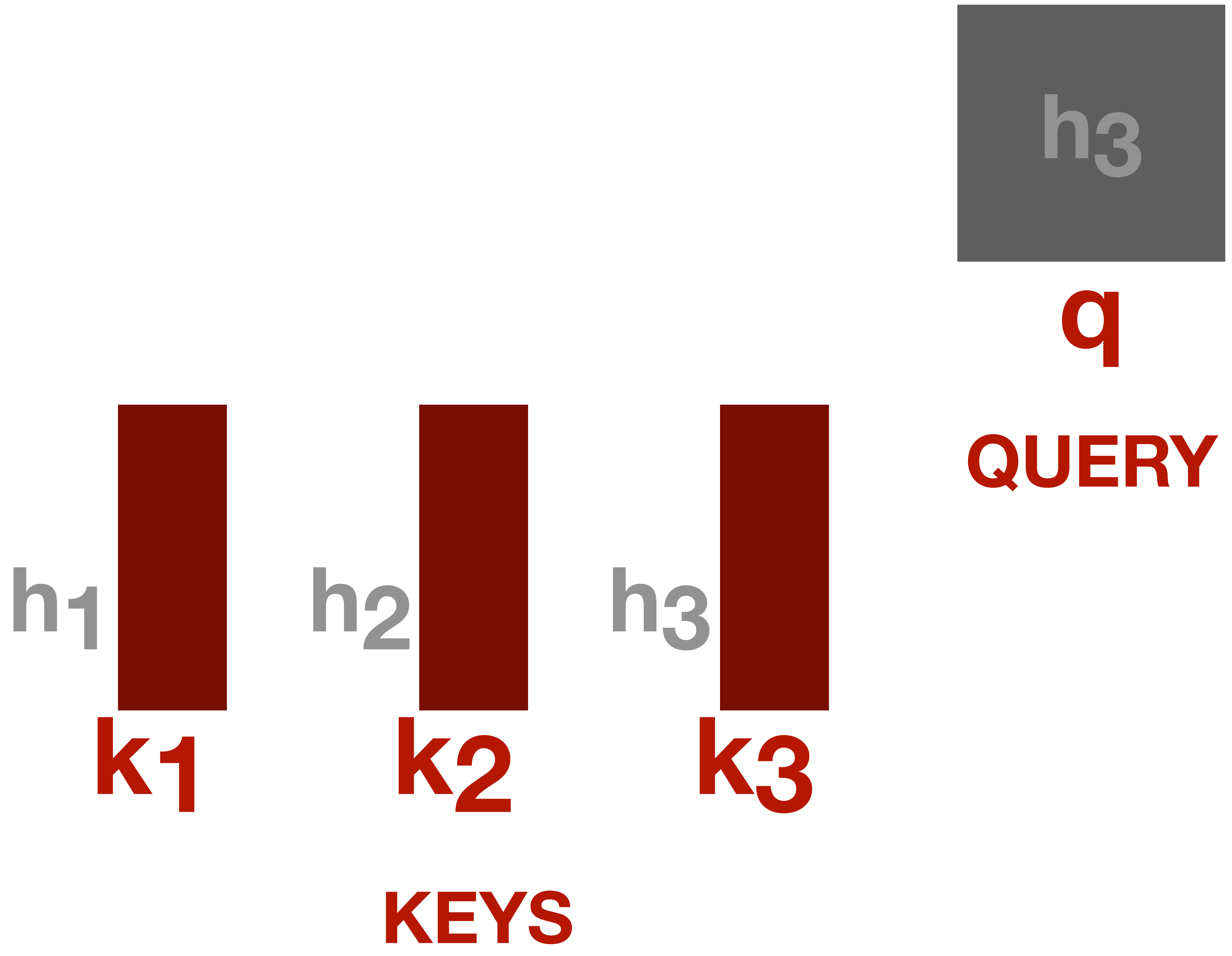


# THE ATTENTION MECHANISM

- **Calculating Attention**

STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

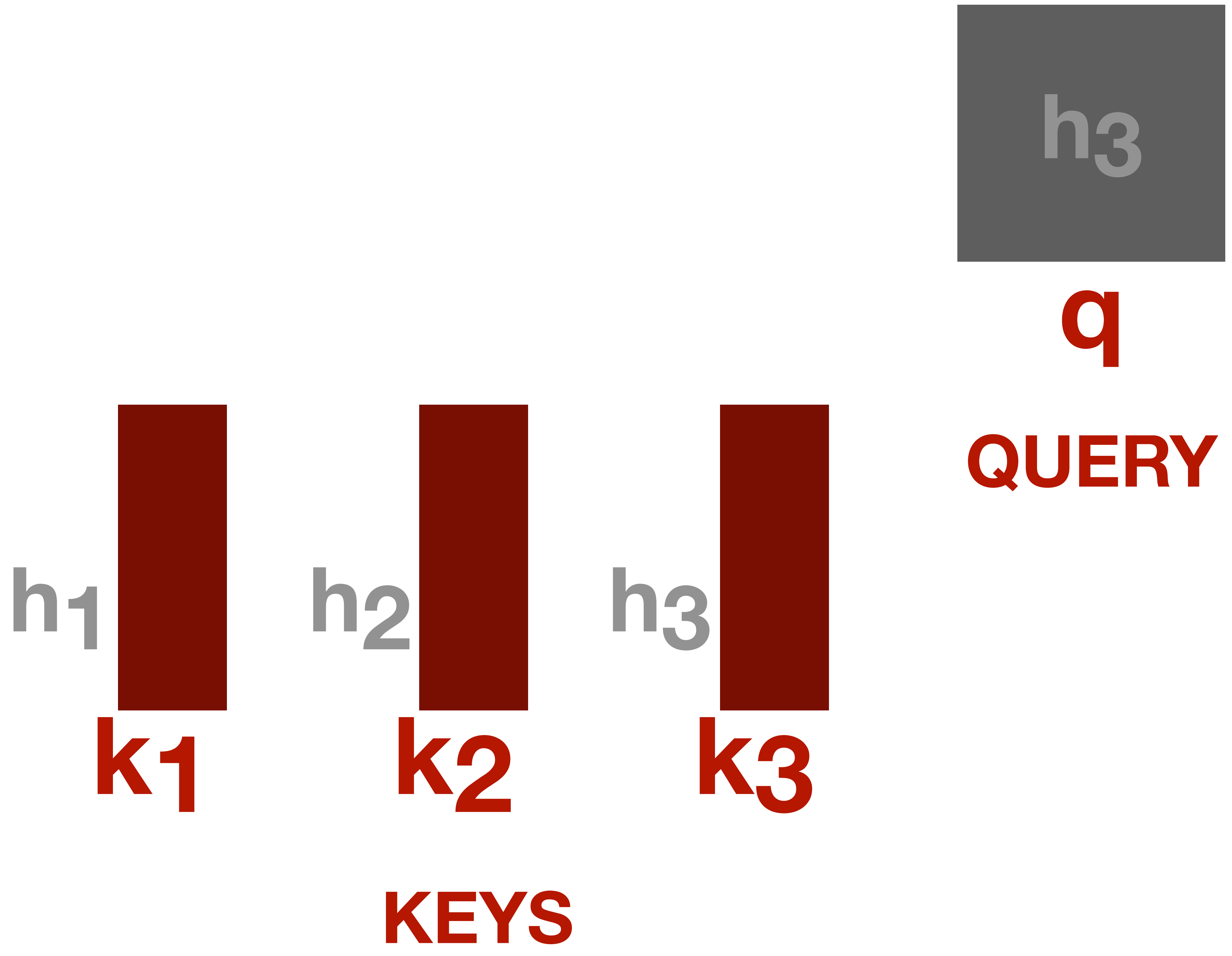



**Examples:**

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^{\mathsf{T}}\boldsymbol{k}_i$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

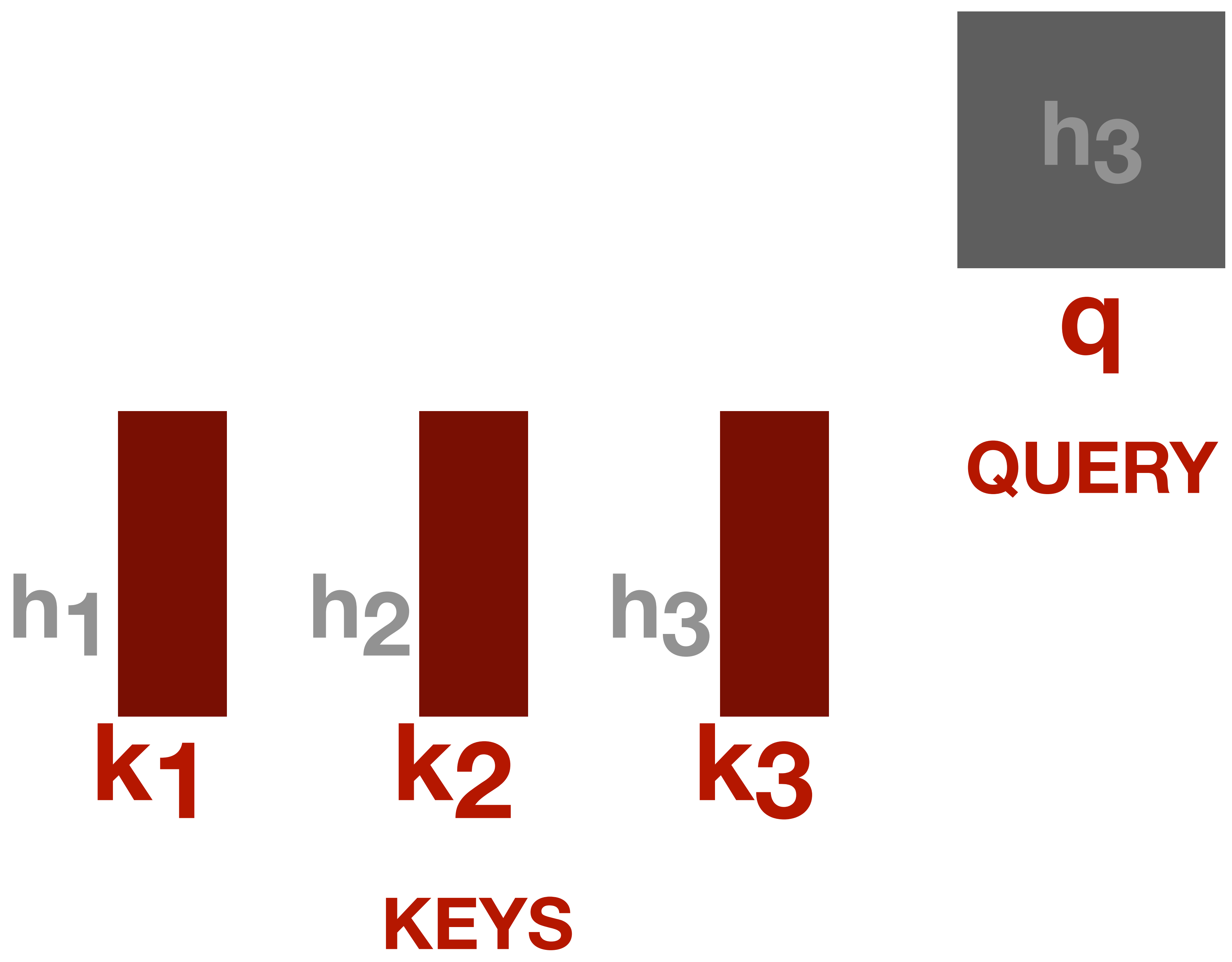
**Examples:**

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^{\mathrm{T}}\boldsymbol{k}_i$$

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^{T}\mathbf{W}\,\boldsymbol{k}_i$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

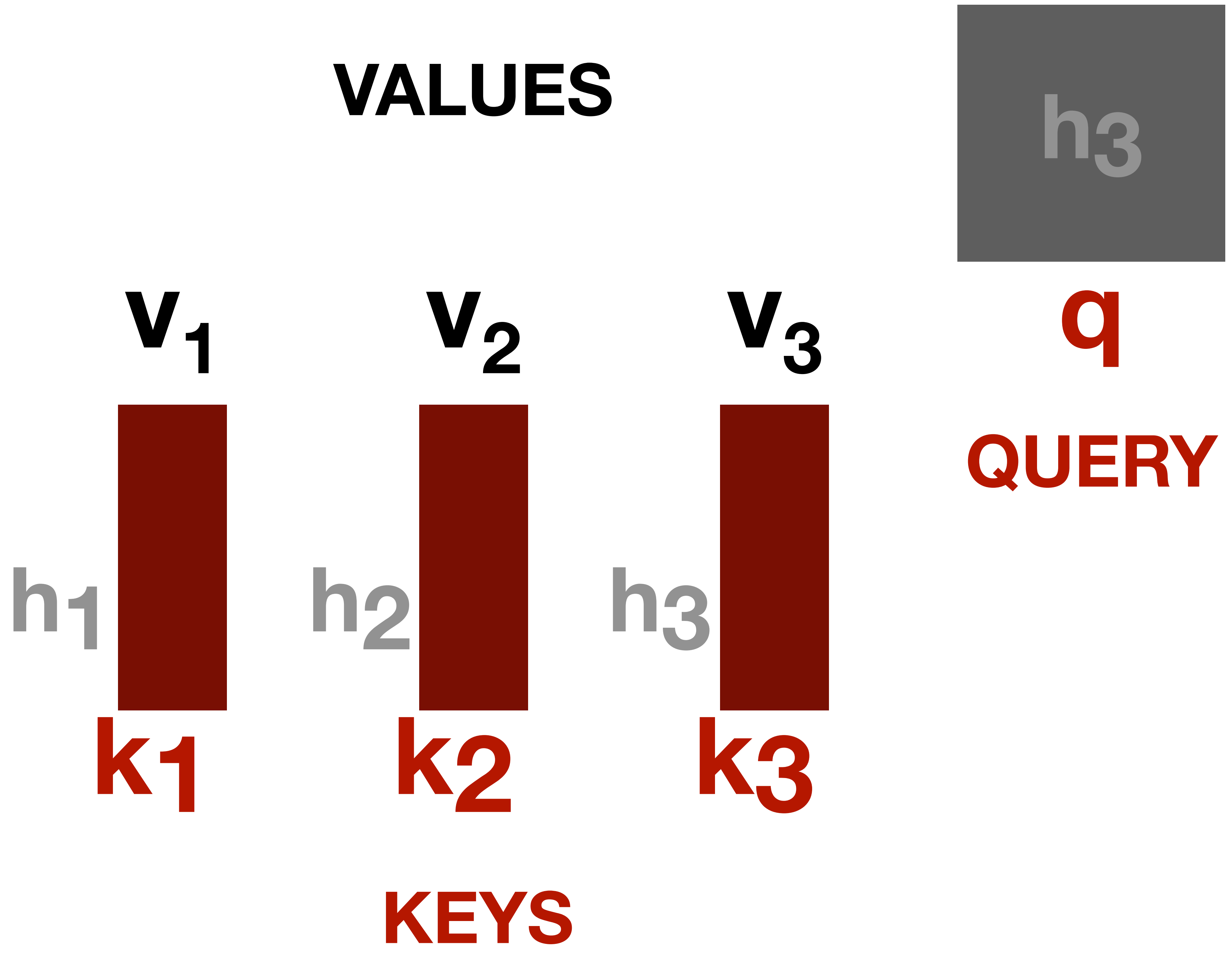
$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

STEP -2 : TAKE SOFTMAX OVER RAW WEIGHTS

$$w_i(t) = \frac{\exp(\, e_i(\boldsymbol{q}(t), \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}(t), \boldsymbol{k}_j)\,)}$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

STEP -2 : TAKE SOFTMAX OVER RAW WEIGHTS

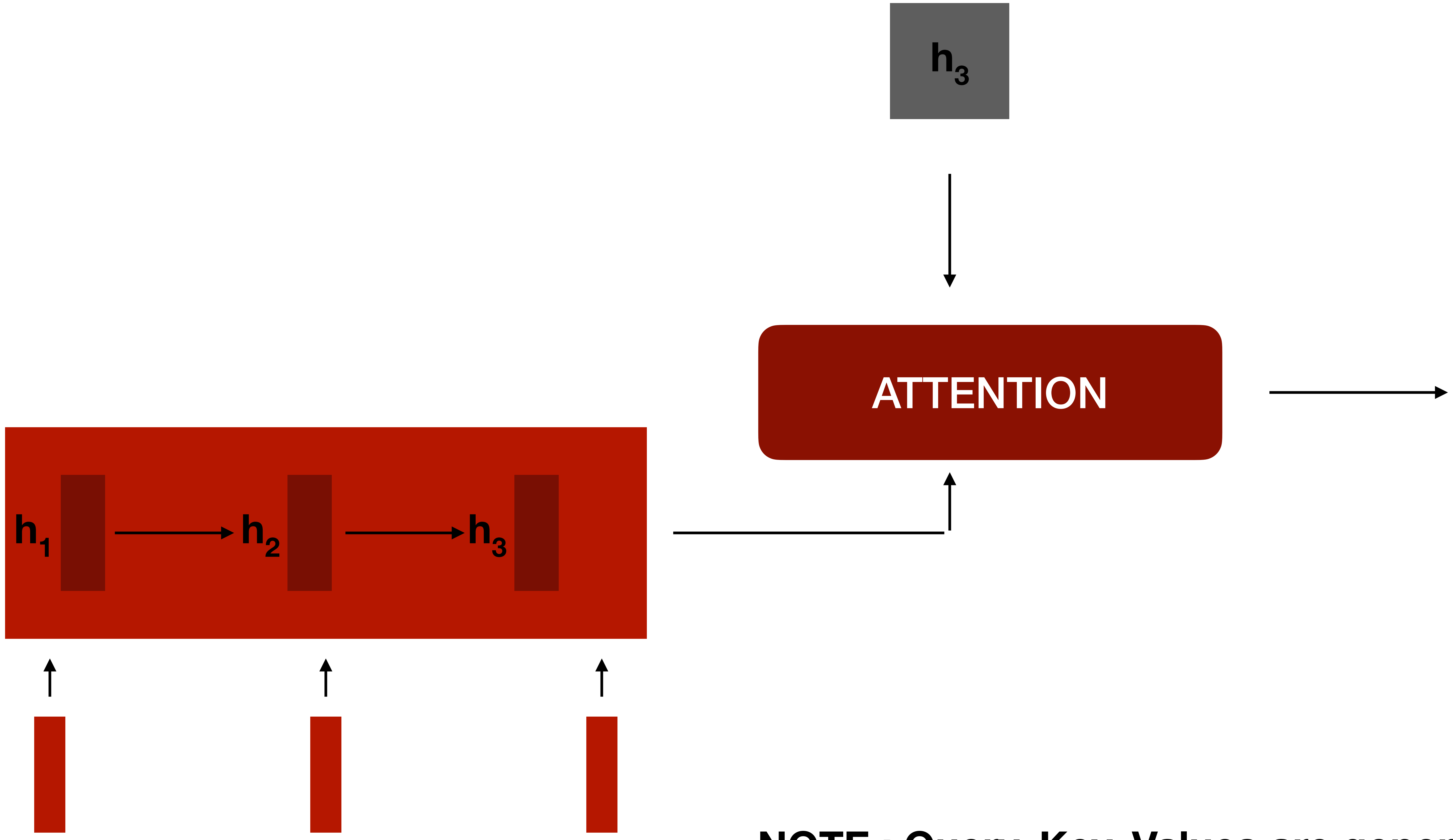
$$w_i(t) = \frac{\exp(\, e_i(\boldsymbol{q}(t), \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}(t), \boldsymbol{k}_j)\,)}$$

STEP -3 : TAKE A LINEAR COMBINATION

$$\boldsymbol{O}(\boldsymbol{k},\boldsymbol{q}(t),\boldsymbol{v}) = \sum_i w_i(t)\, \boldsymbol{v}_i$$

# THE ATTENTION MECHANISM

- **Attention**




$$\boldsymbol{O}(\boldsymbol{k},\boldsymbol{q}(t),\boldsymbol{v}) = \sum_i w_i(t)\, \boldsymbol{v}_i$$

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**

# THE ATTENTION MECHANISM

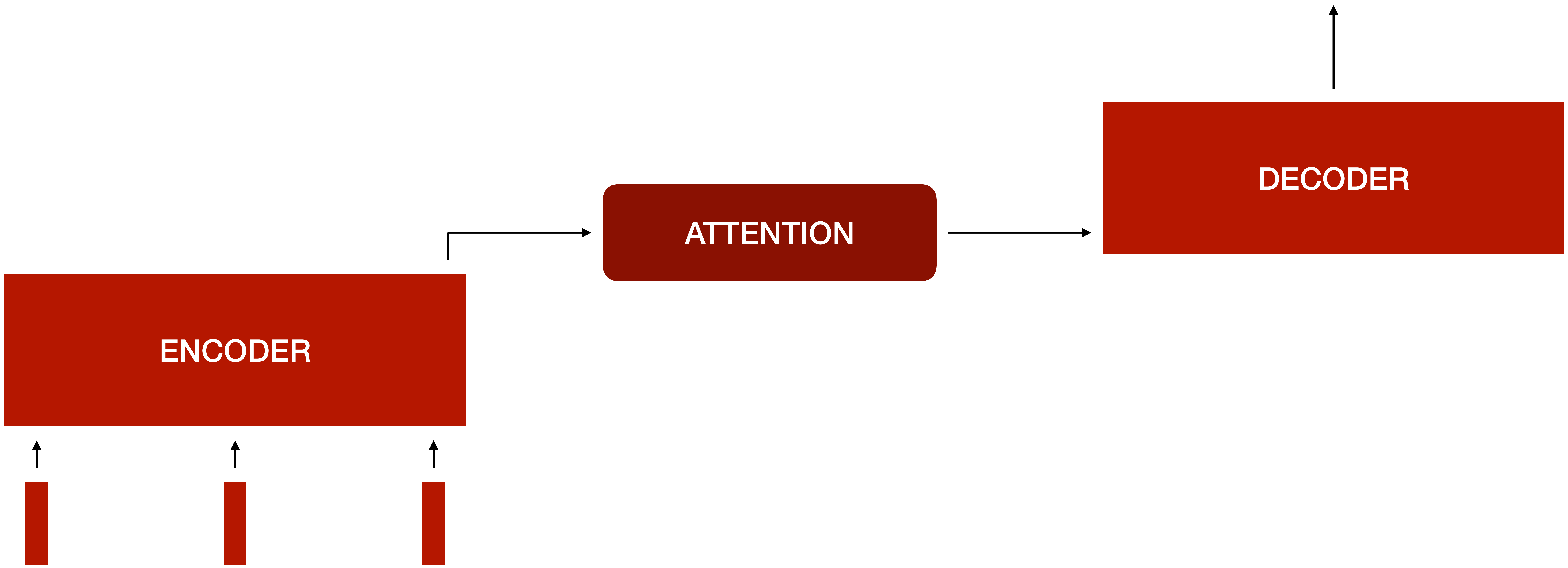
- Attention
- **Self-Attention**
- Multi-Head Attention

# THE ATTENTION MECHANISM

- **Self-Attention**

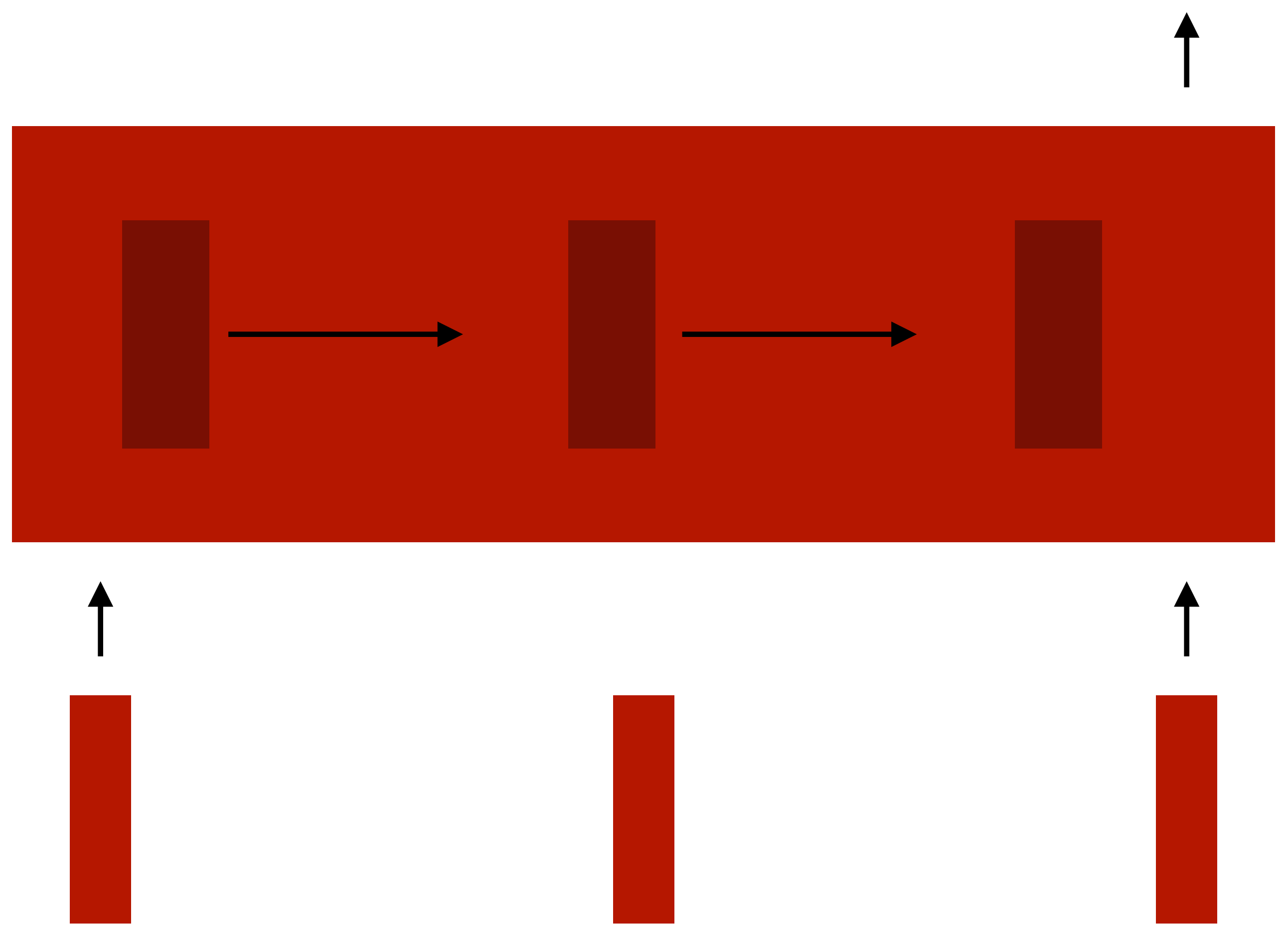
# THE ATTENTION MECHANISM

- **Self-Attention**



# THE ATTENTION MECHANISM

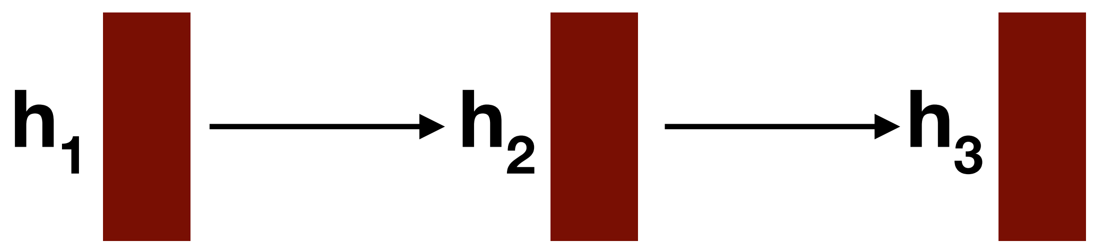
- **Self-Attention**



# THE ATTENTION MECHANISM

- **Self-Attention**

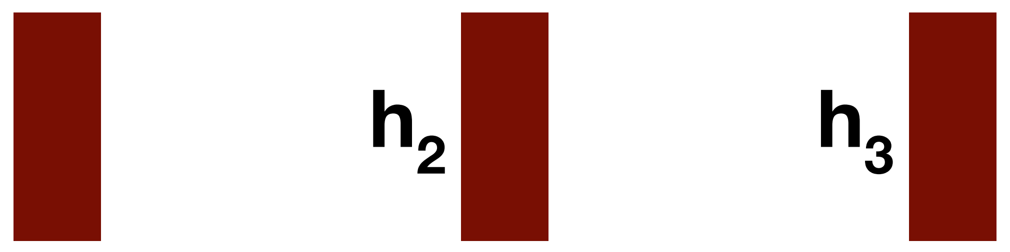
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- **Self-Attention**

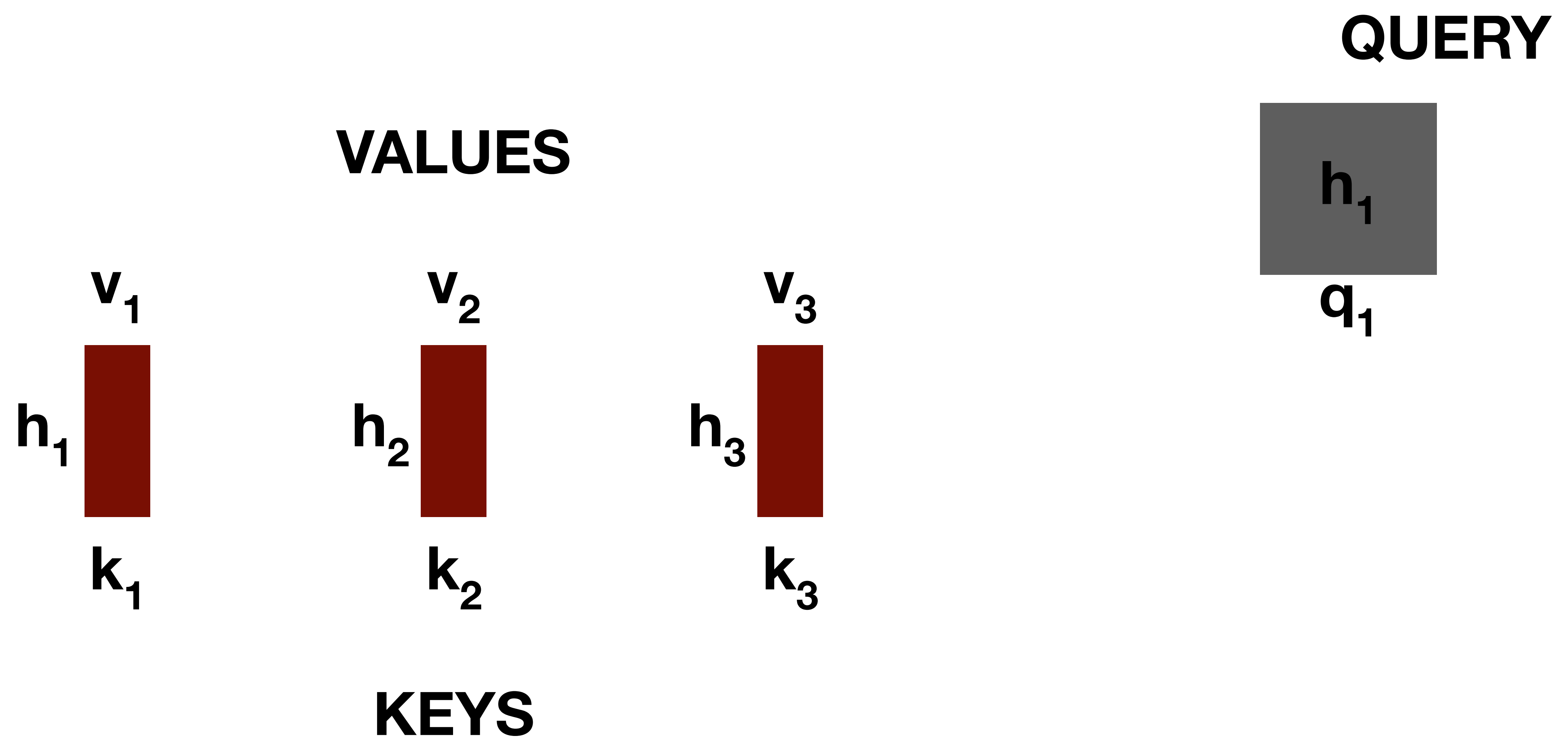
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- **Self-Attention**

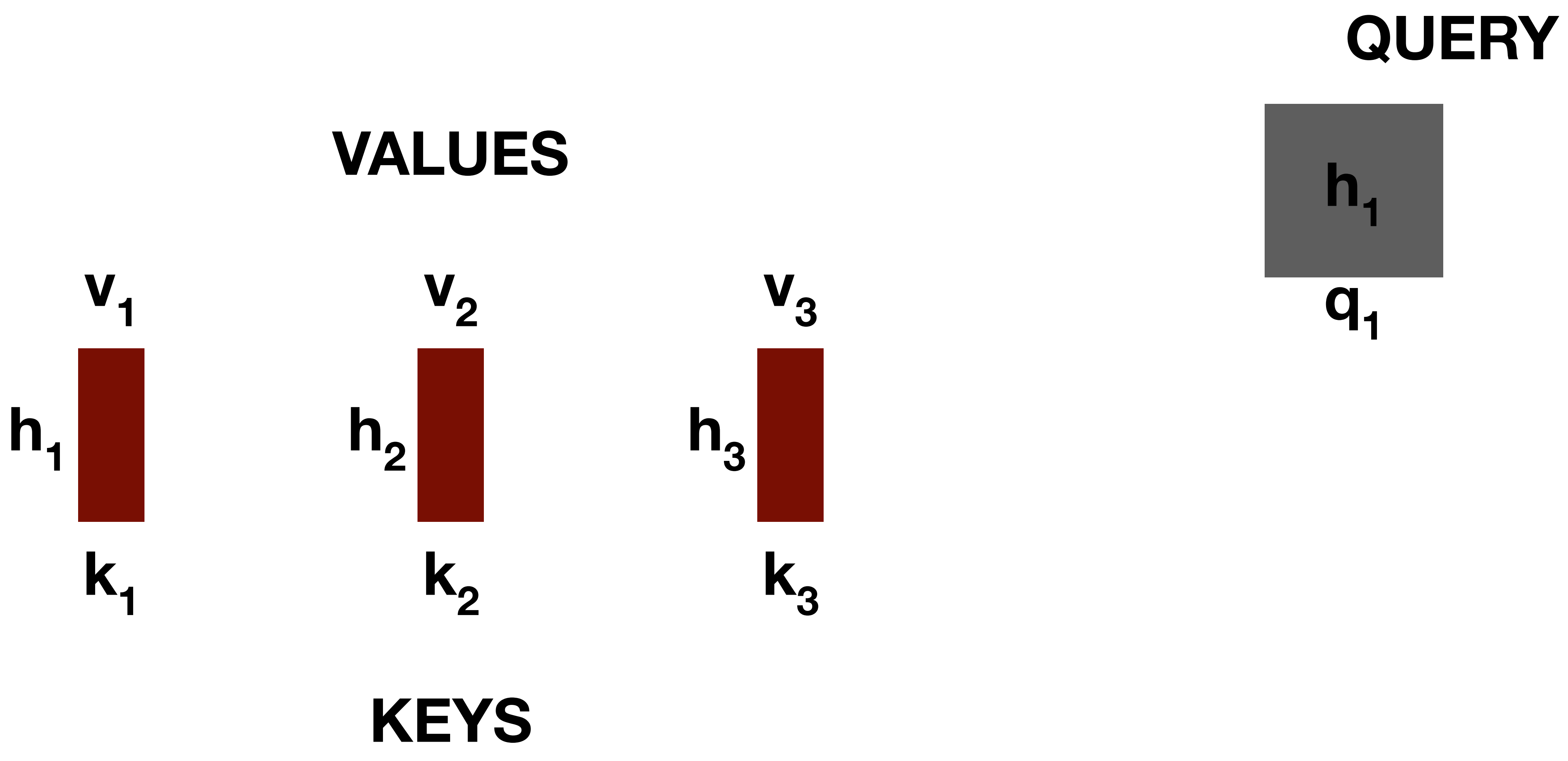
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- **Self-Attention**

**Find the attention of each hidden state with every other hidden state in the sequence**



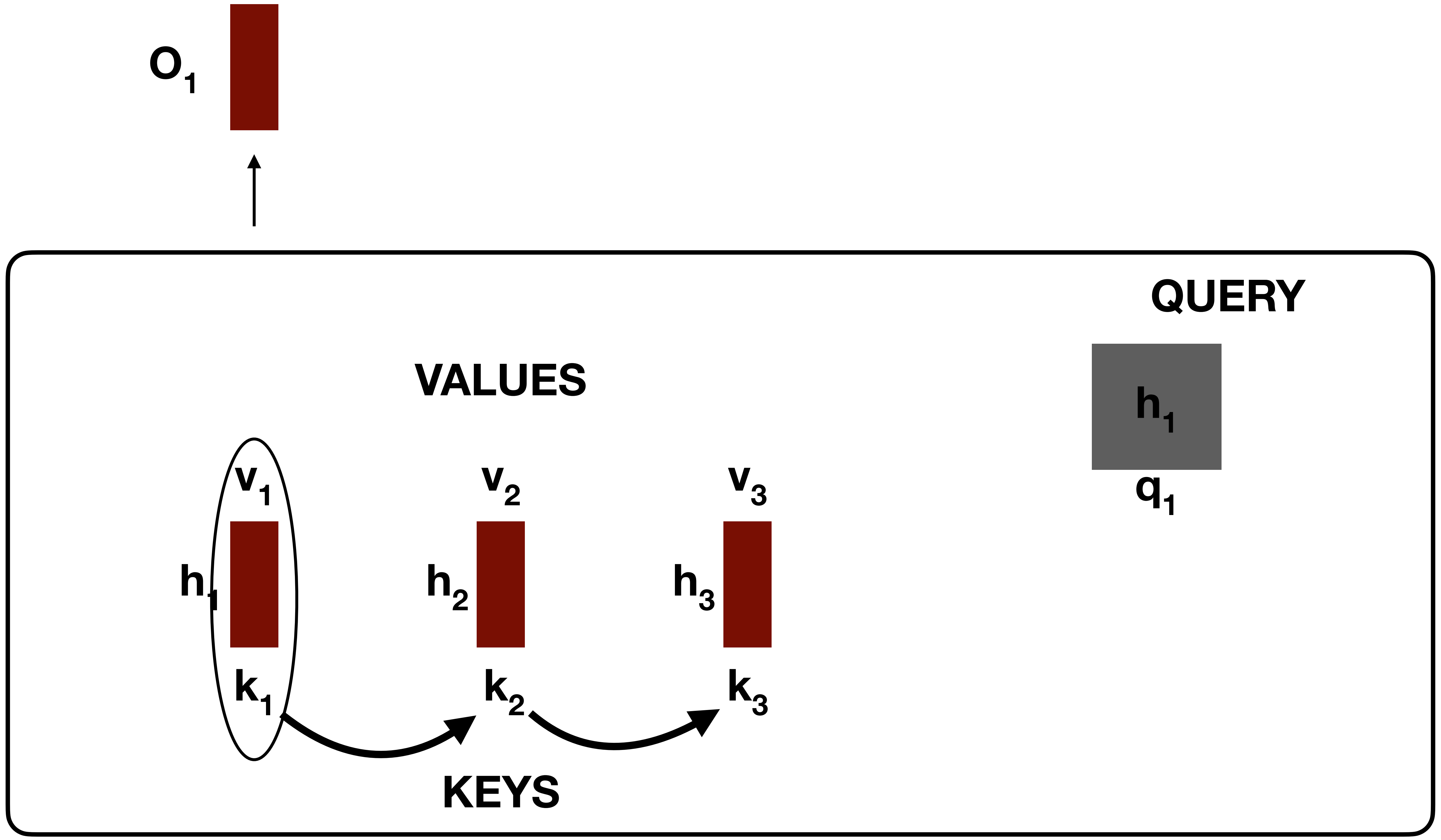

$$\boldsymbol{O}_1\,(\boldsymbol{k}_i, \boldsymbol{q}_1) = \sum w_i\, \boldsymbol{v}_i$$

$$\boxed{w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

**Find the attention of each hidden state with every other hidden state in the sequence**



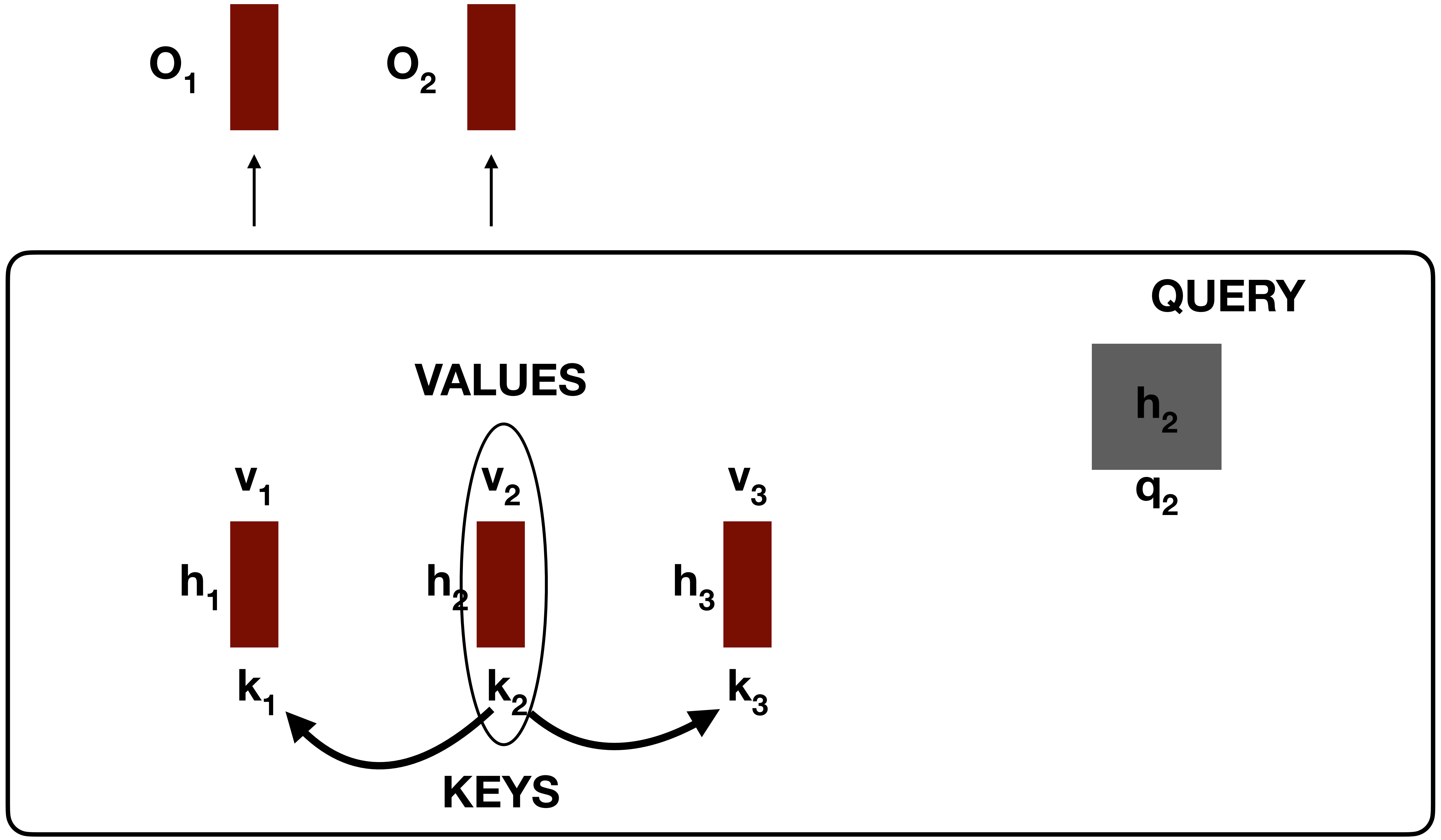

$$\boldsymbol{O}_1\,(\boldsymbol{k}_i, \boldsymbol{q}_1) = \sum w_i\, \boldsymbol{v}_i$$

$$\boxed{w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

**Find the attention of each hidden state with every other hidden state in the sequence**



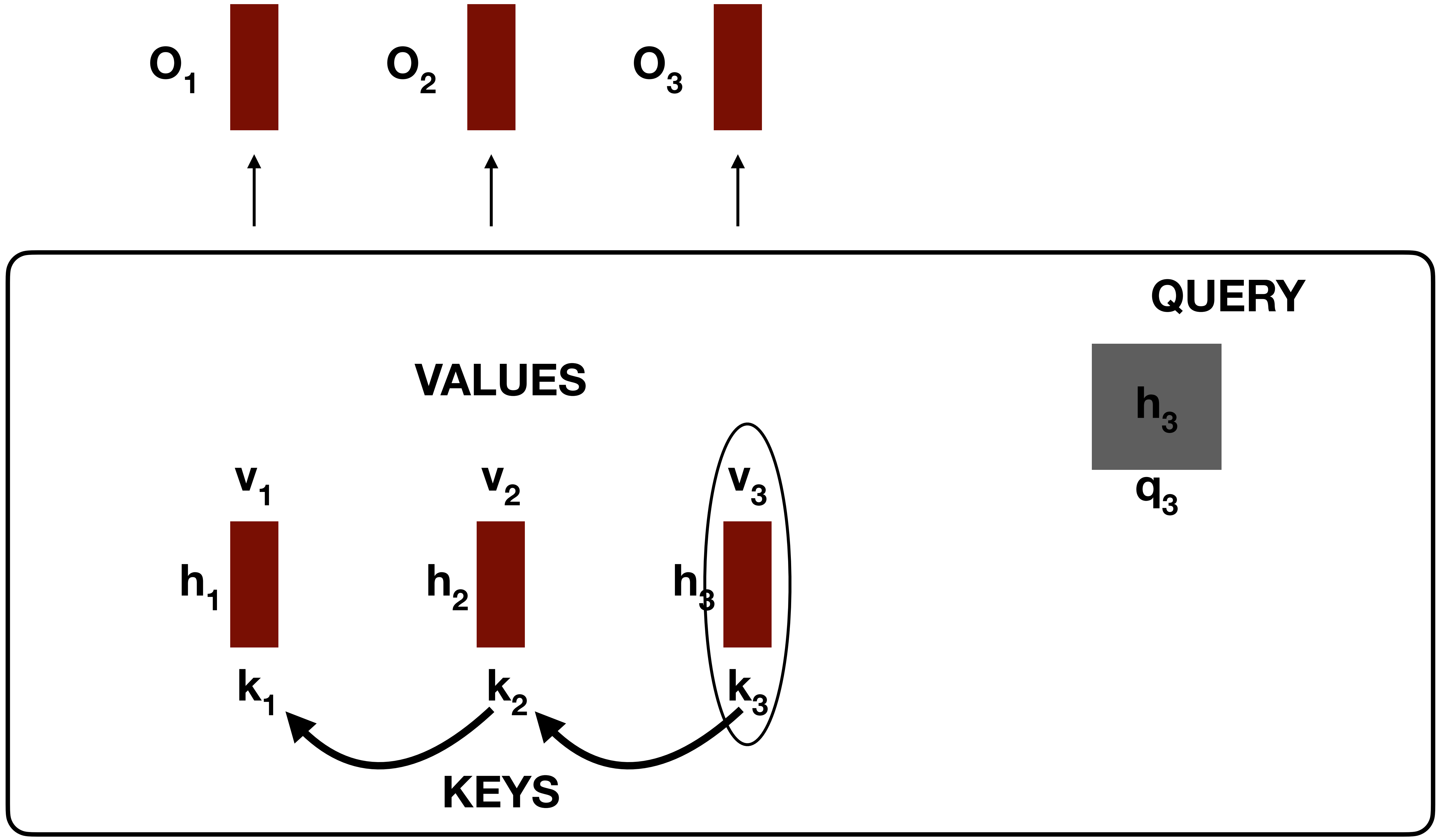

$$\boldsymbol{O}_2\,(\boldsymbol{k}_i,\ \boldsymbol{q}_2) = \sum w_i\ \boldsymbol{v}_i$$

$$\boxed{w_{i(t)} = \frac{\exp(\ e_i(\boldsymbol{q}_{(t)},\ \boldsymbol{k}_i)\ )}{\sum_j \exp(\ e_j(\boldsymbol{q}_{(t)},\ \boldsymbol{k}_j)\ )}}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

**Find the attention of each hidden state with every other hidden state in the sequence**



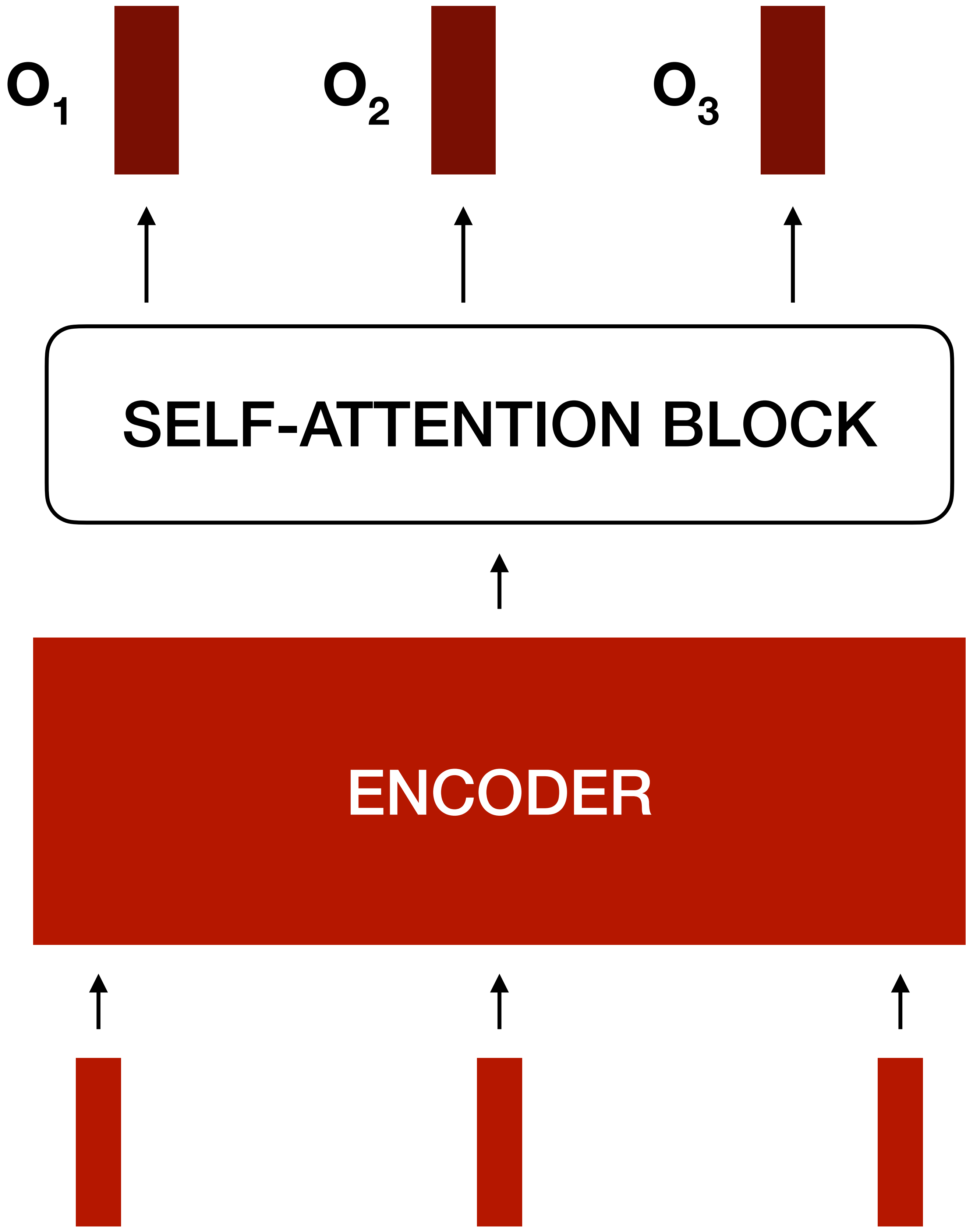

$$\boldsymbol{O}_3\,(\boldsymbol{k}_i,\ \boldsymbol{q}_3) = \sum w_i\, \boldsymbol{v}_i$$

$$w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

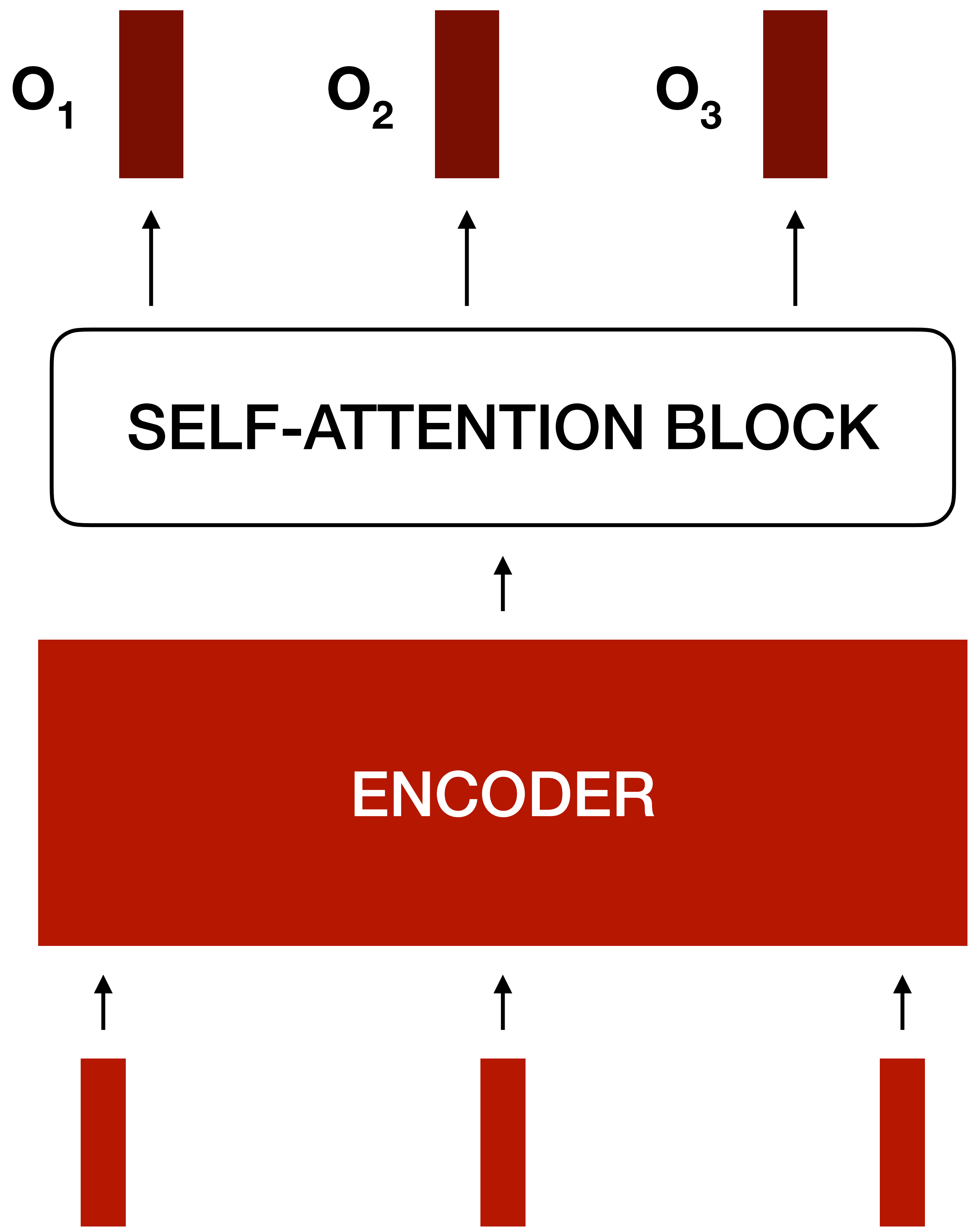
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- **Self-Attention**

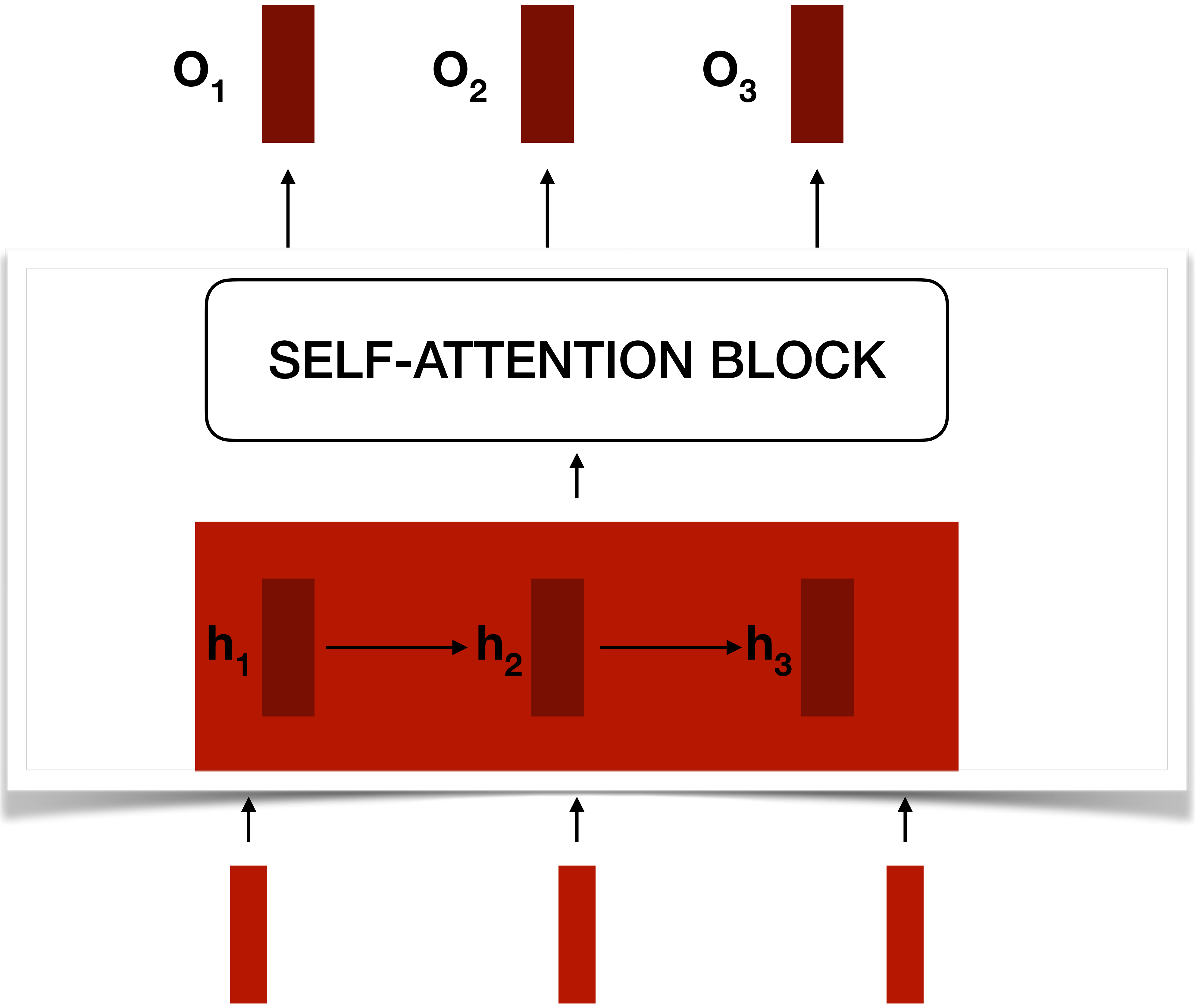
**Find the attention of each hidden state with every other hidden state in the sequence**




**Note: Our attention mechanism has no learnable parameter if we use dot product attention**

# THE ATTENTION MECHANISM

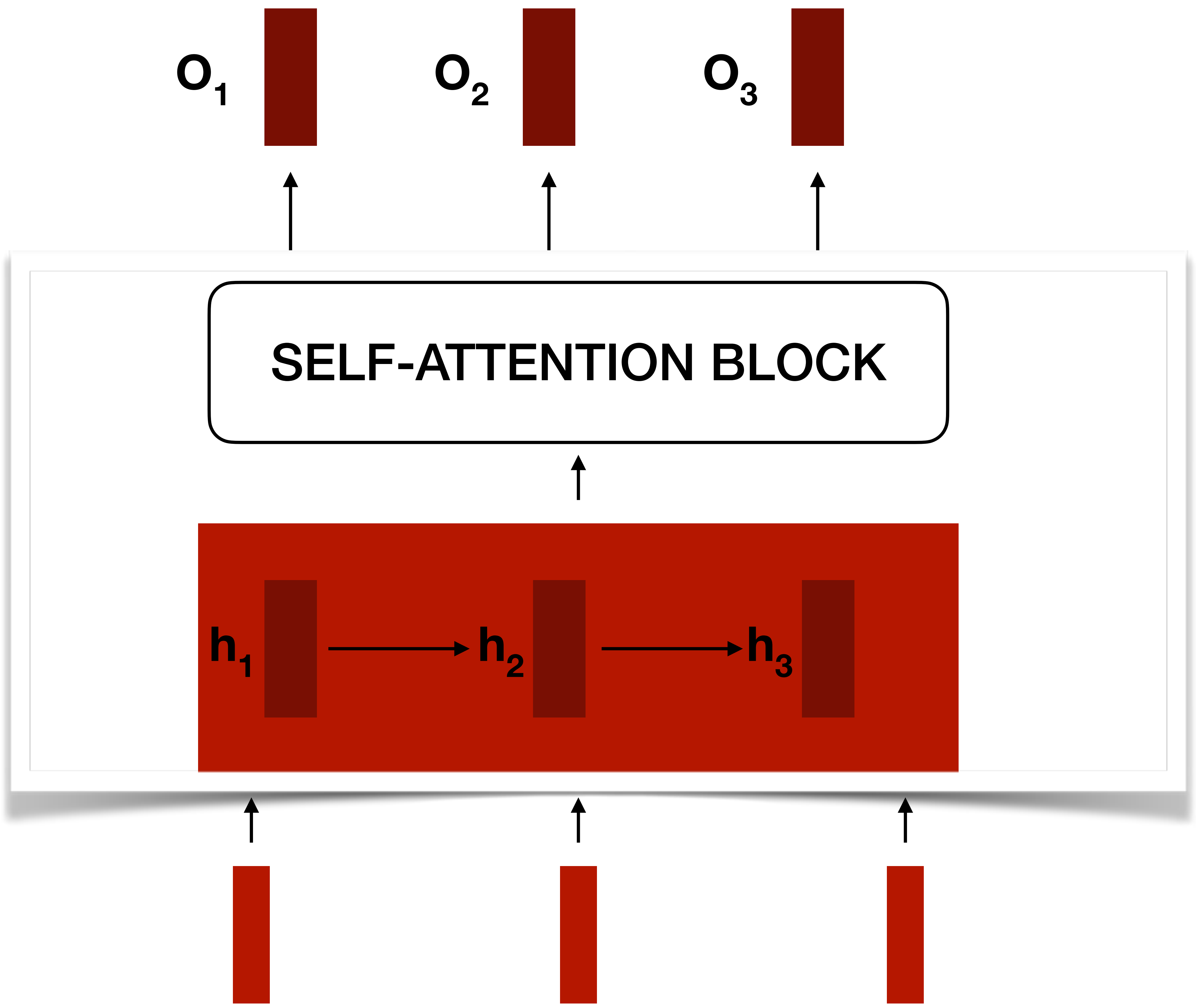
- **Self-Attention**




**ENCODER BLOCK**

# THE ATTENTION MECHANISM

- **Self-Attention**

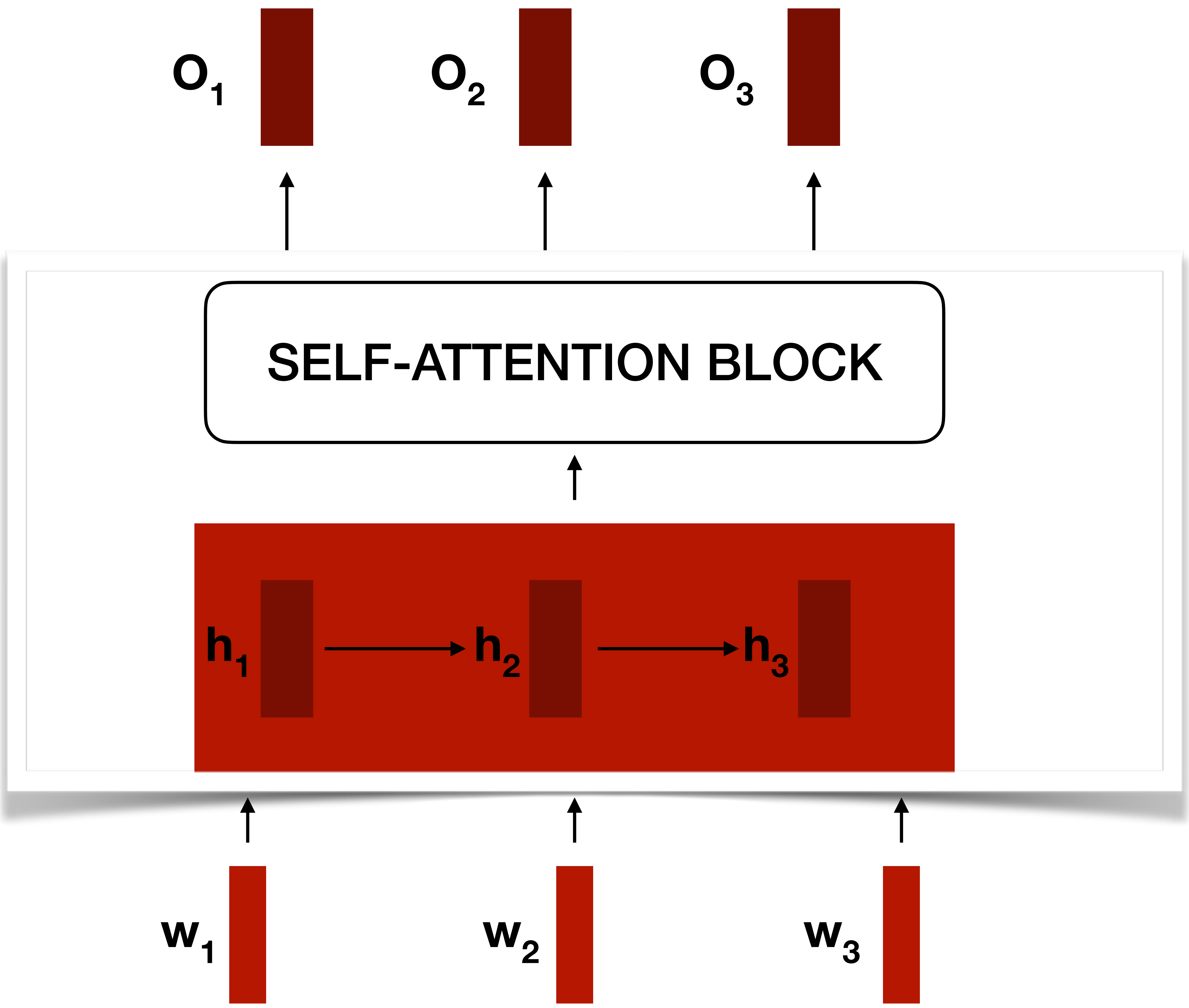



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

- **Self-Attention**

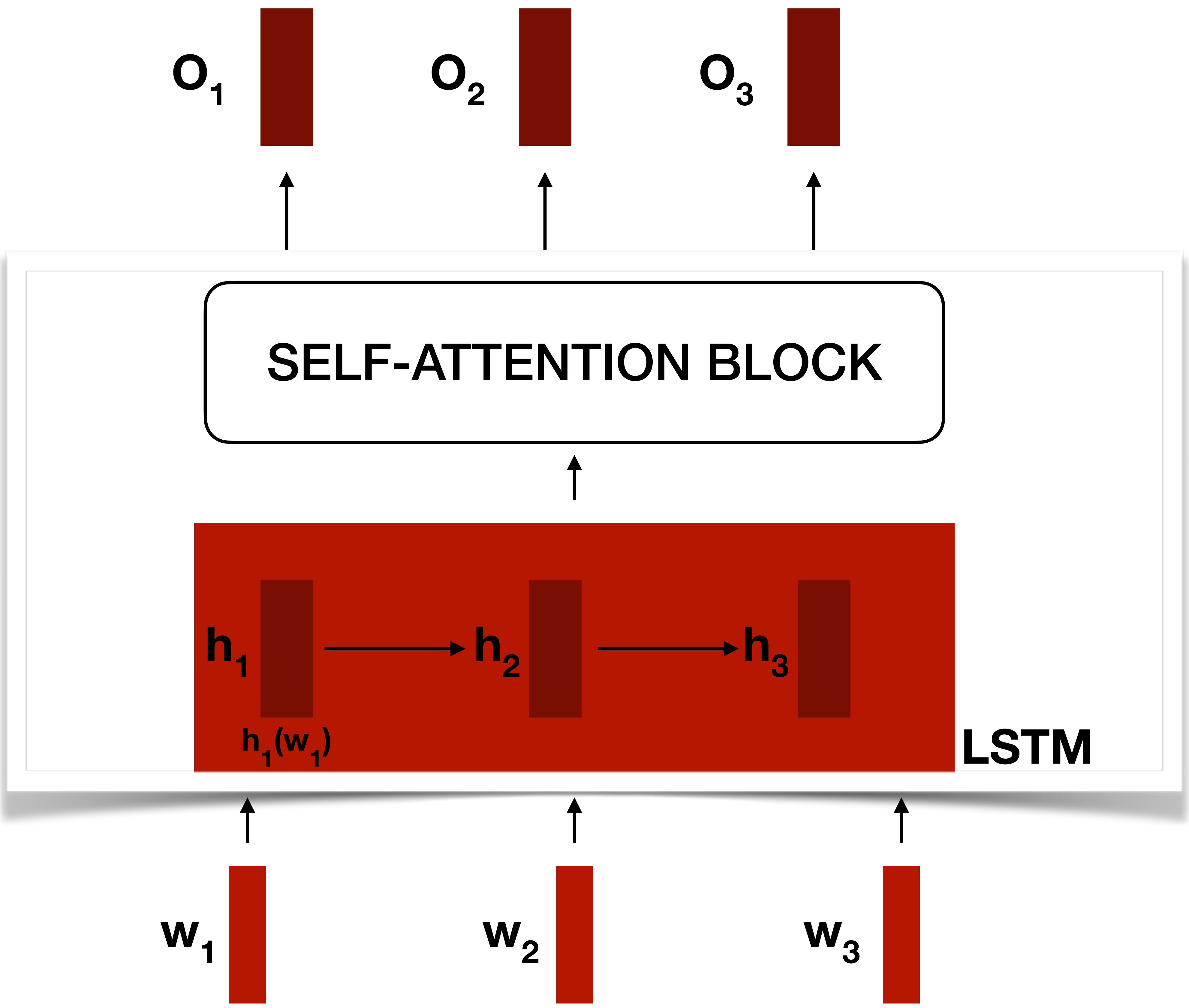



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

- **Self-Attention**

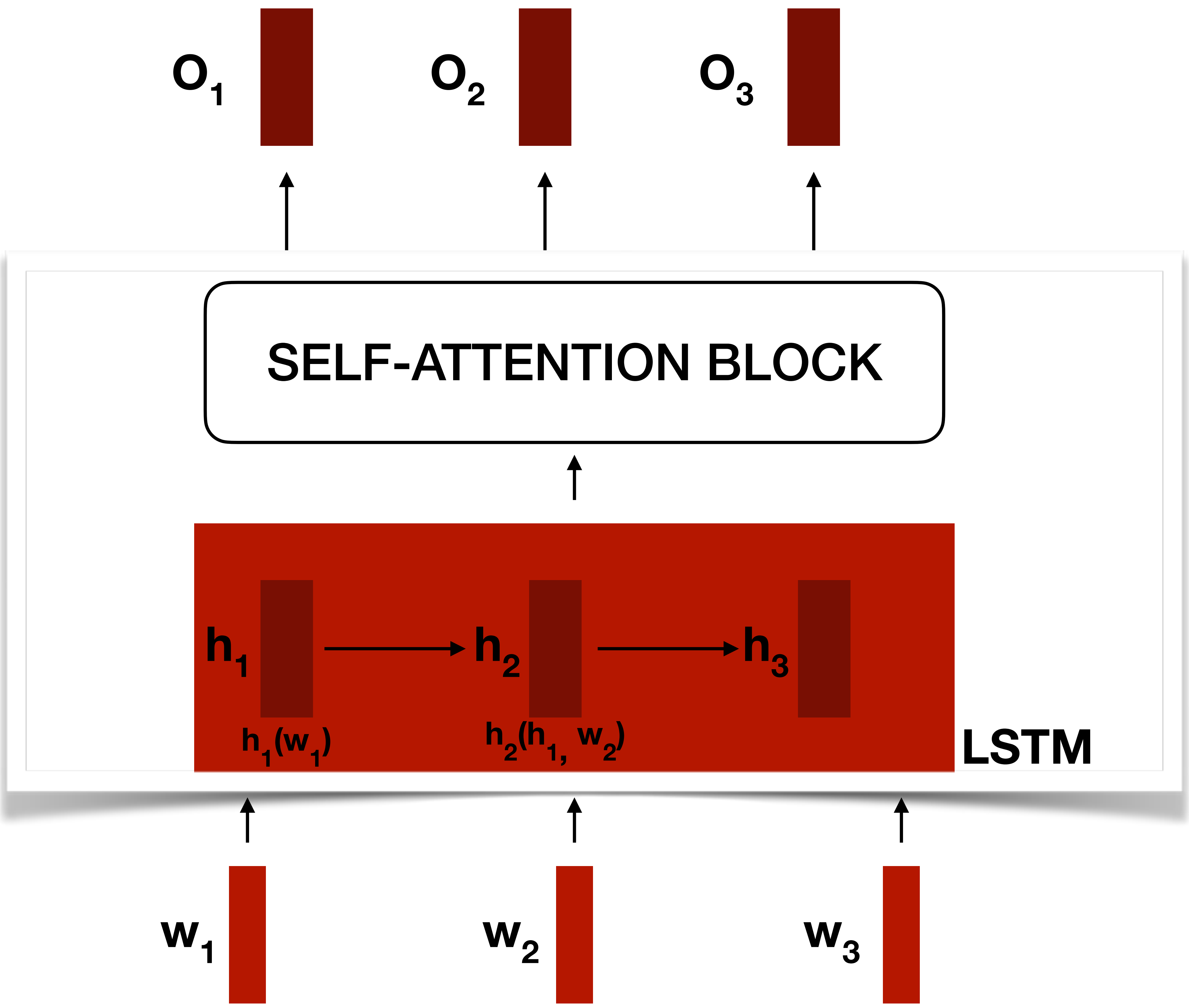



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

- **Self-Attention**

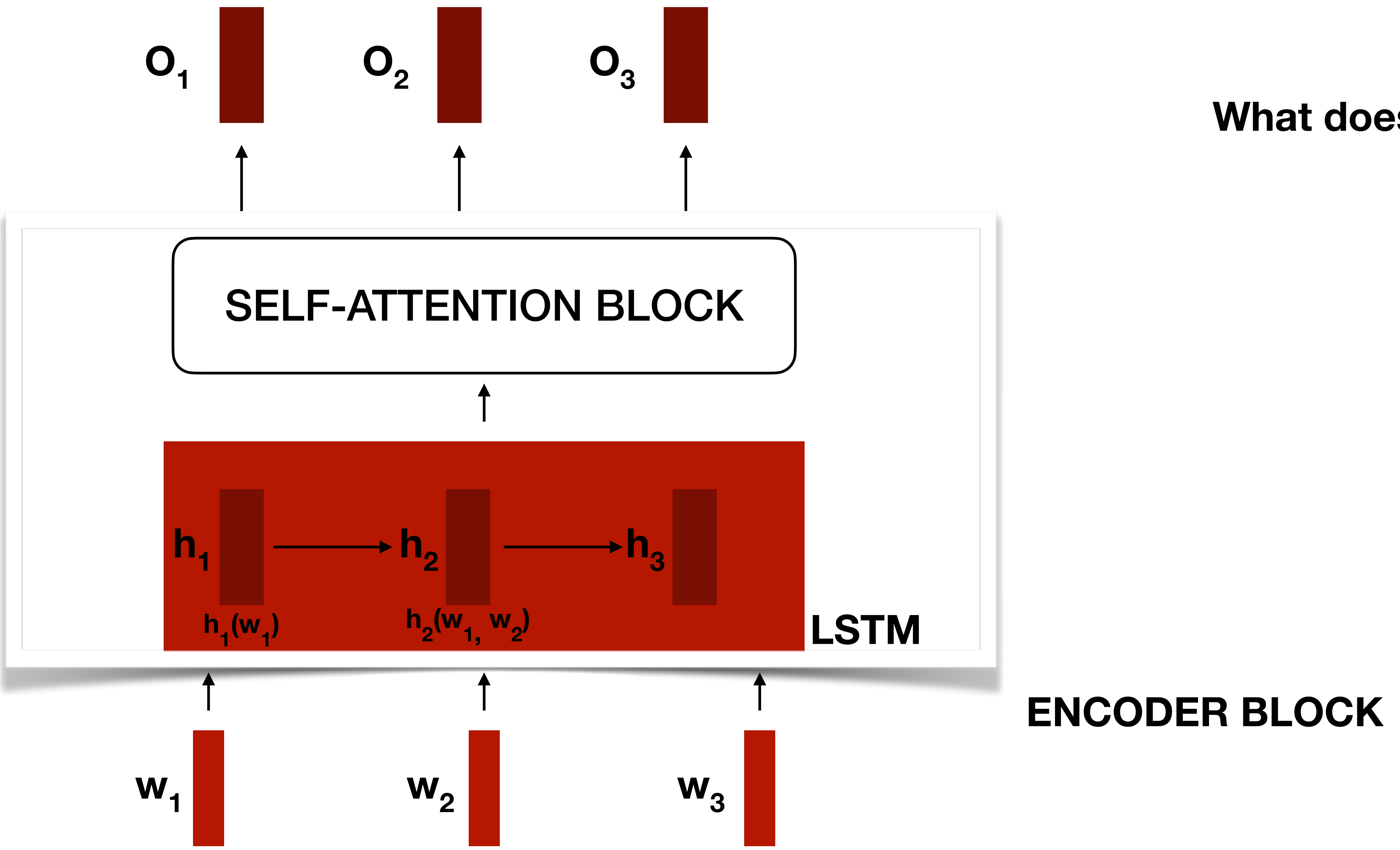



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

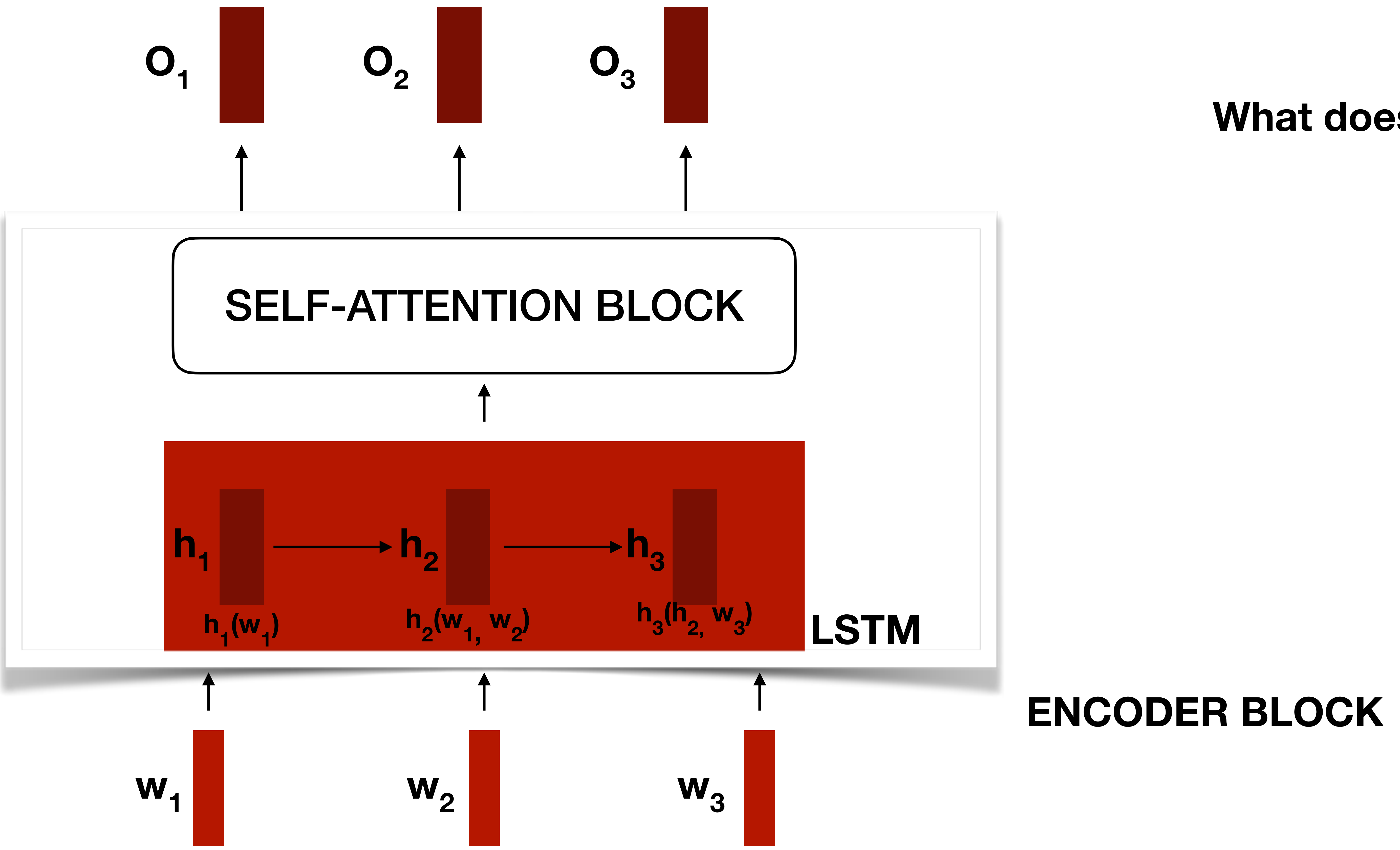
- **Self-Attention**




**What does self attention do here?**

# THE ATTENTION MECHANISM

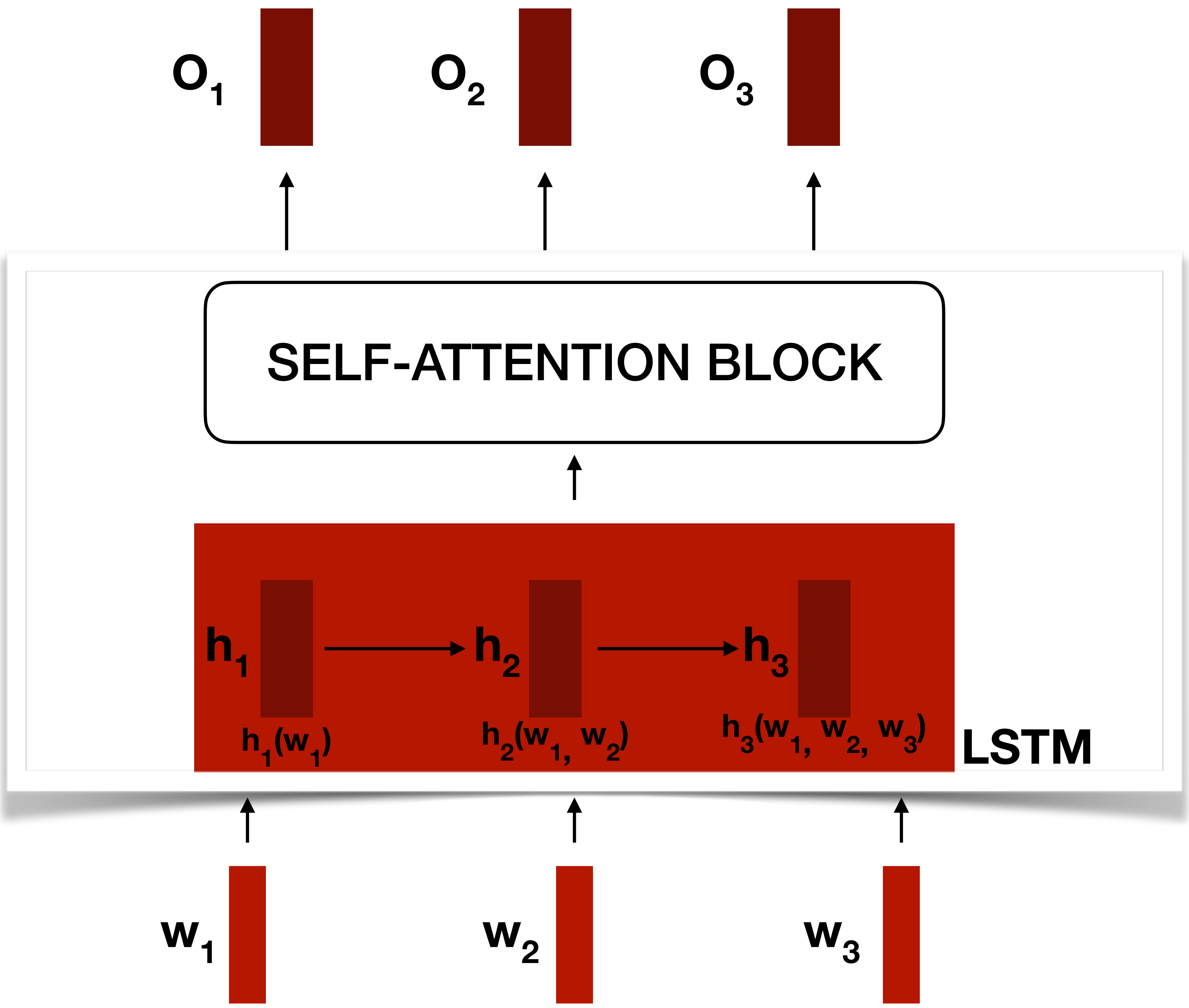
- **Self-Attention**




**What does self attention do here?**

# THE ATTENTION MECHANISM

- **Self-Attention**

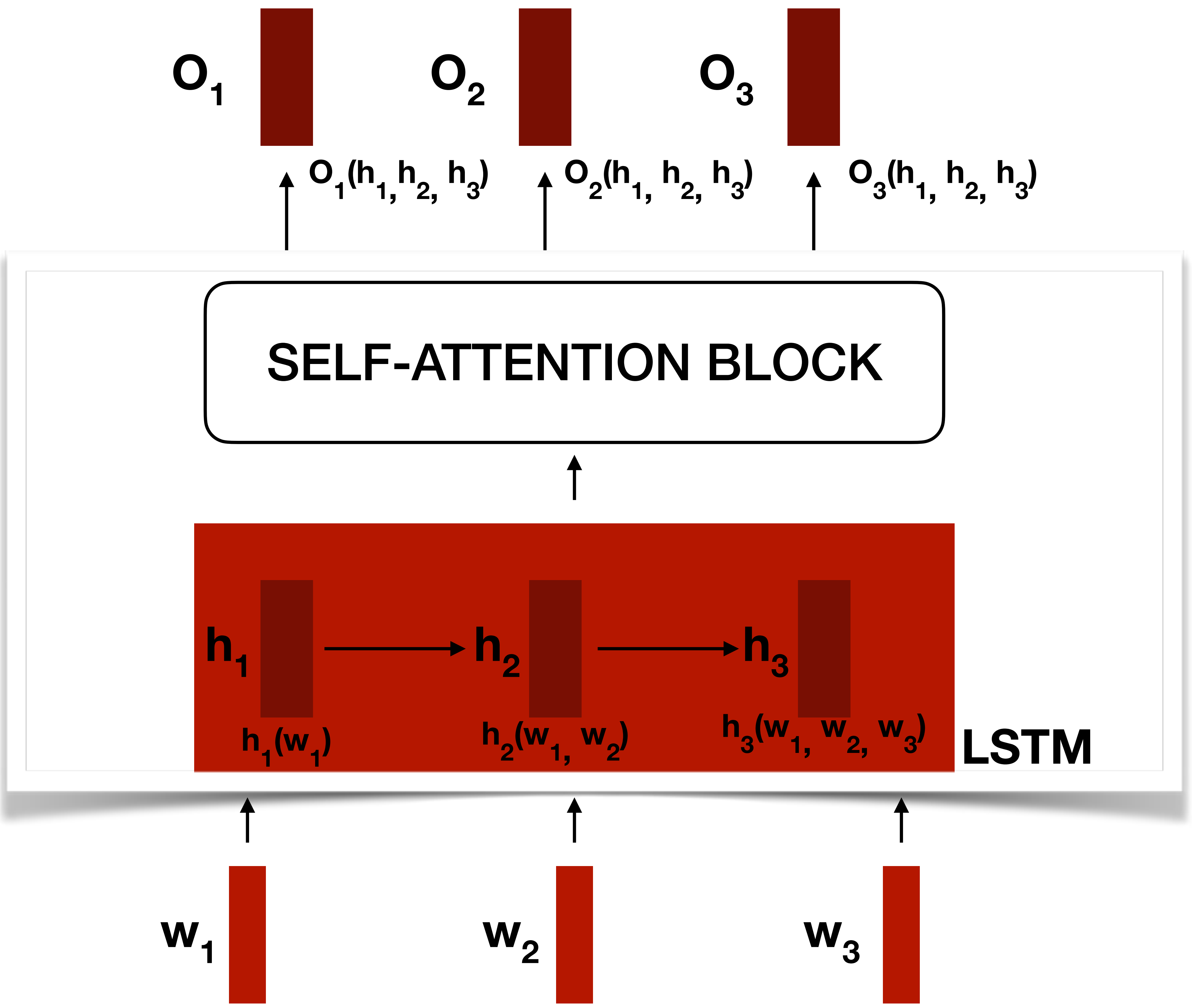



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

- **Self-Attention**

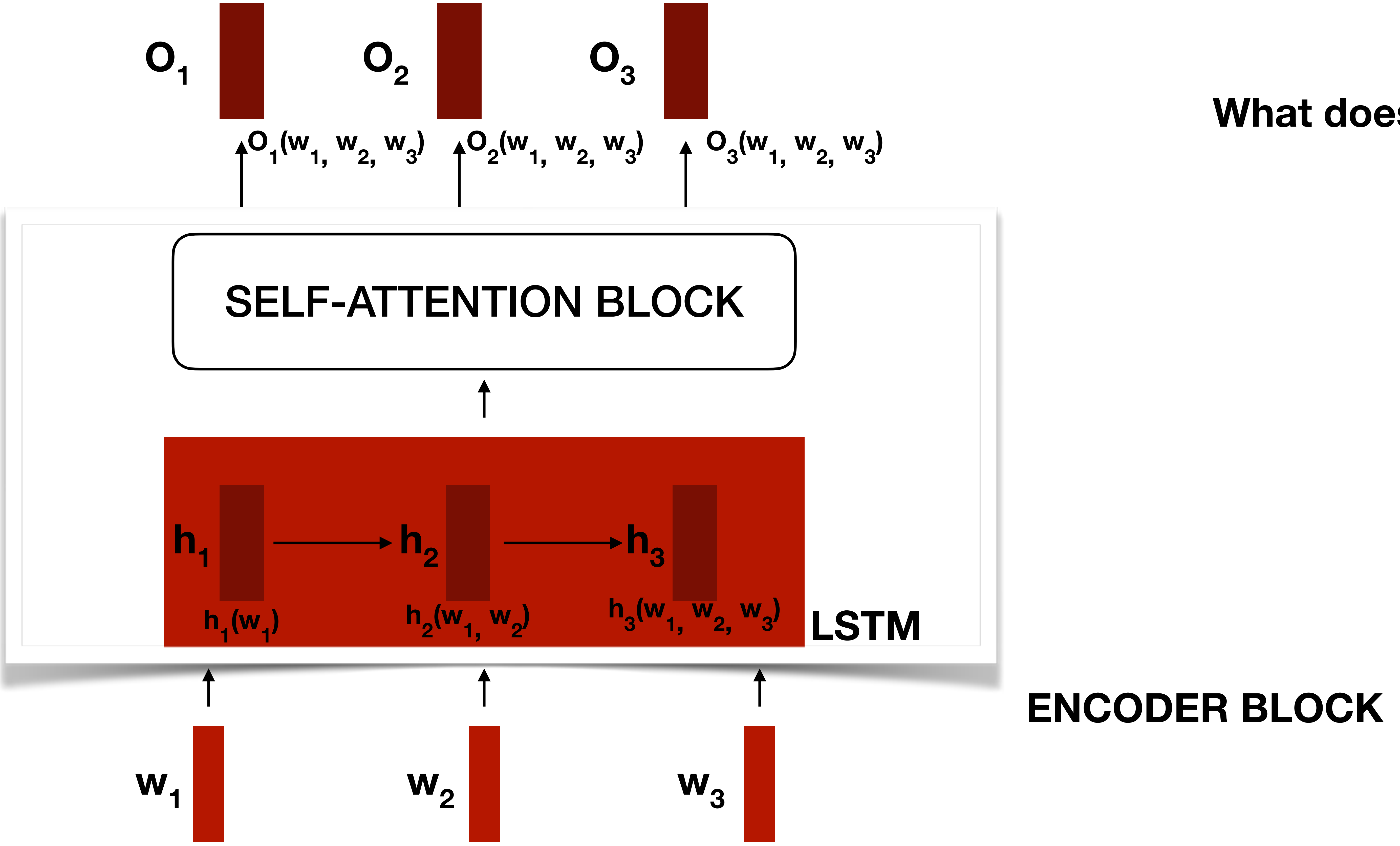



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

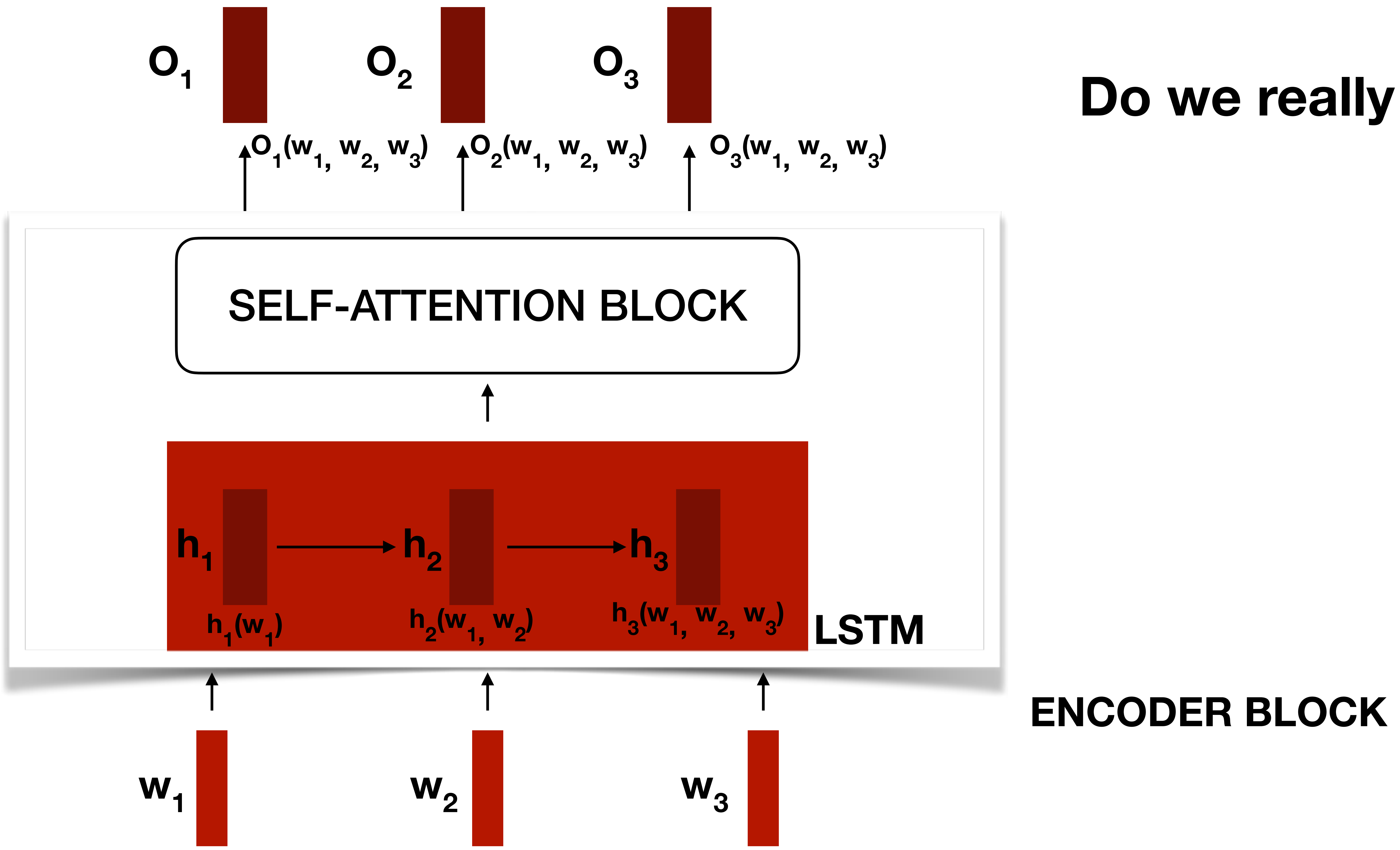
- **Self-Attention**




**What does self attention do here?**

# THE ATTENTION MECHANISM

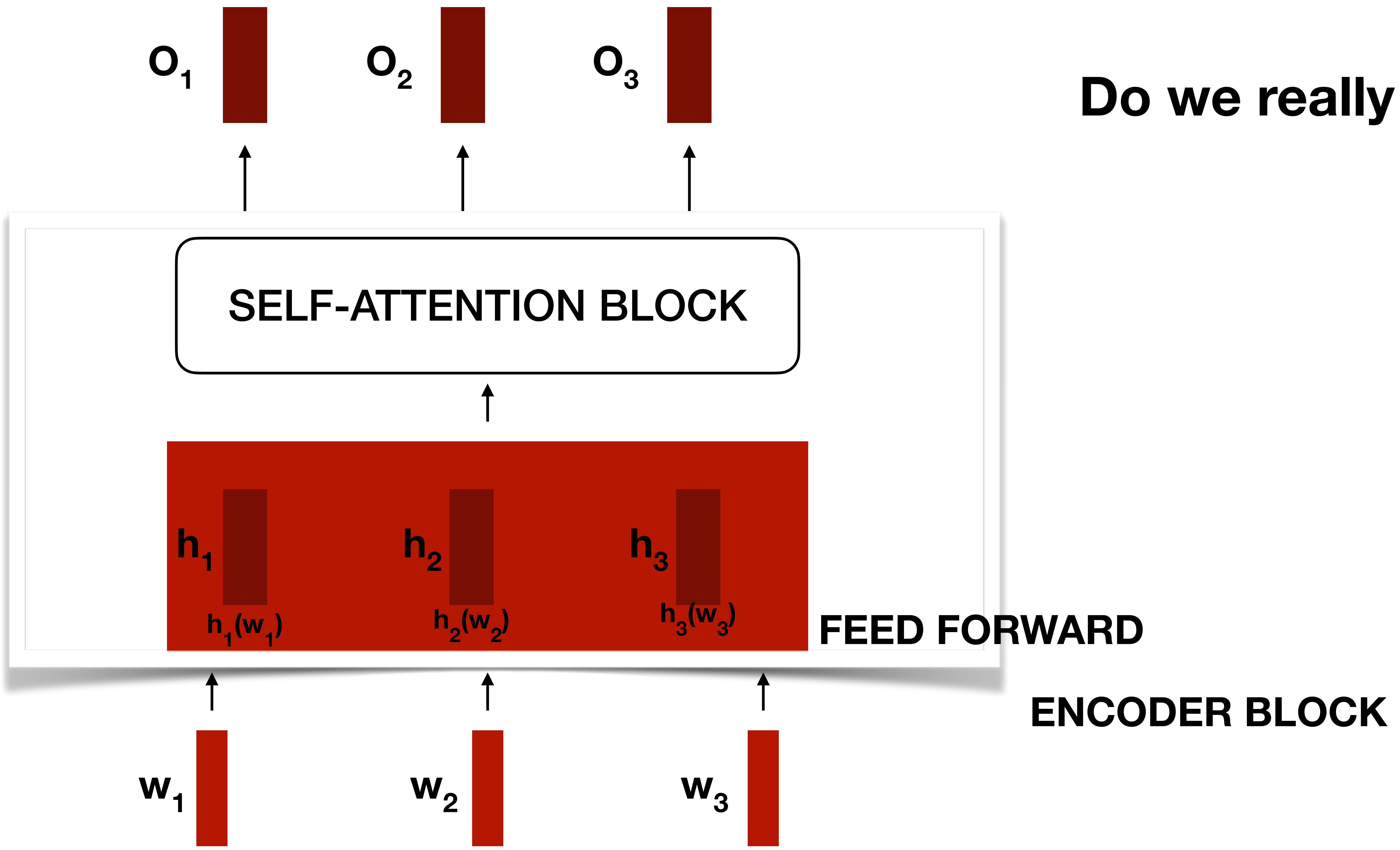
- **Self-Attention**




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

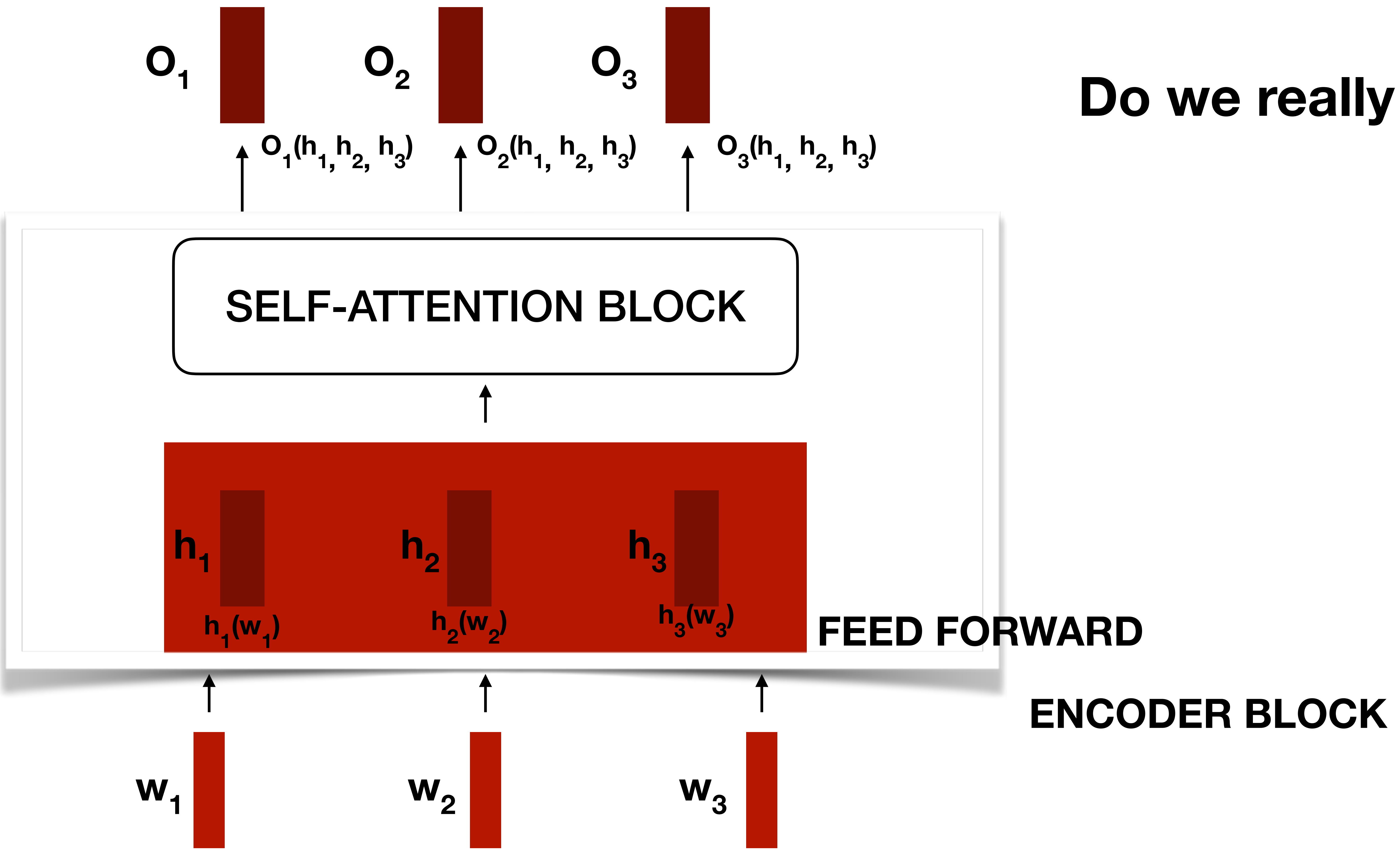
- **Self-Attention**




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

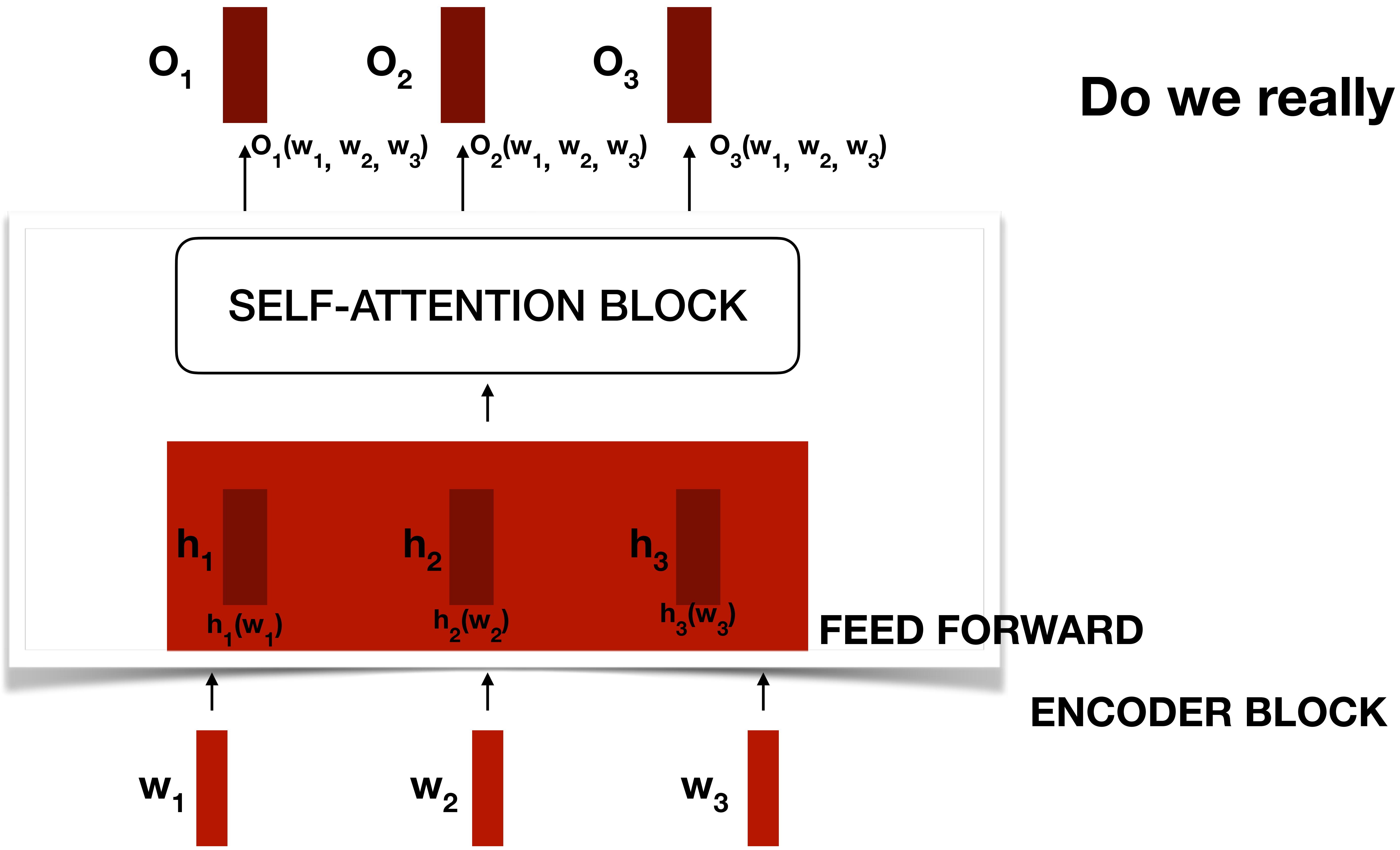
- **Self-Attention**




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

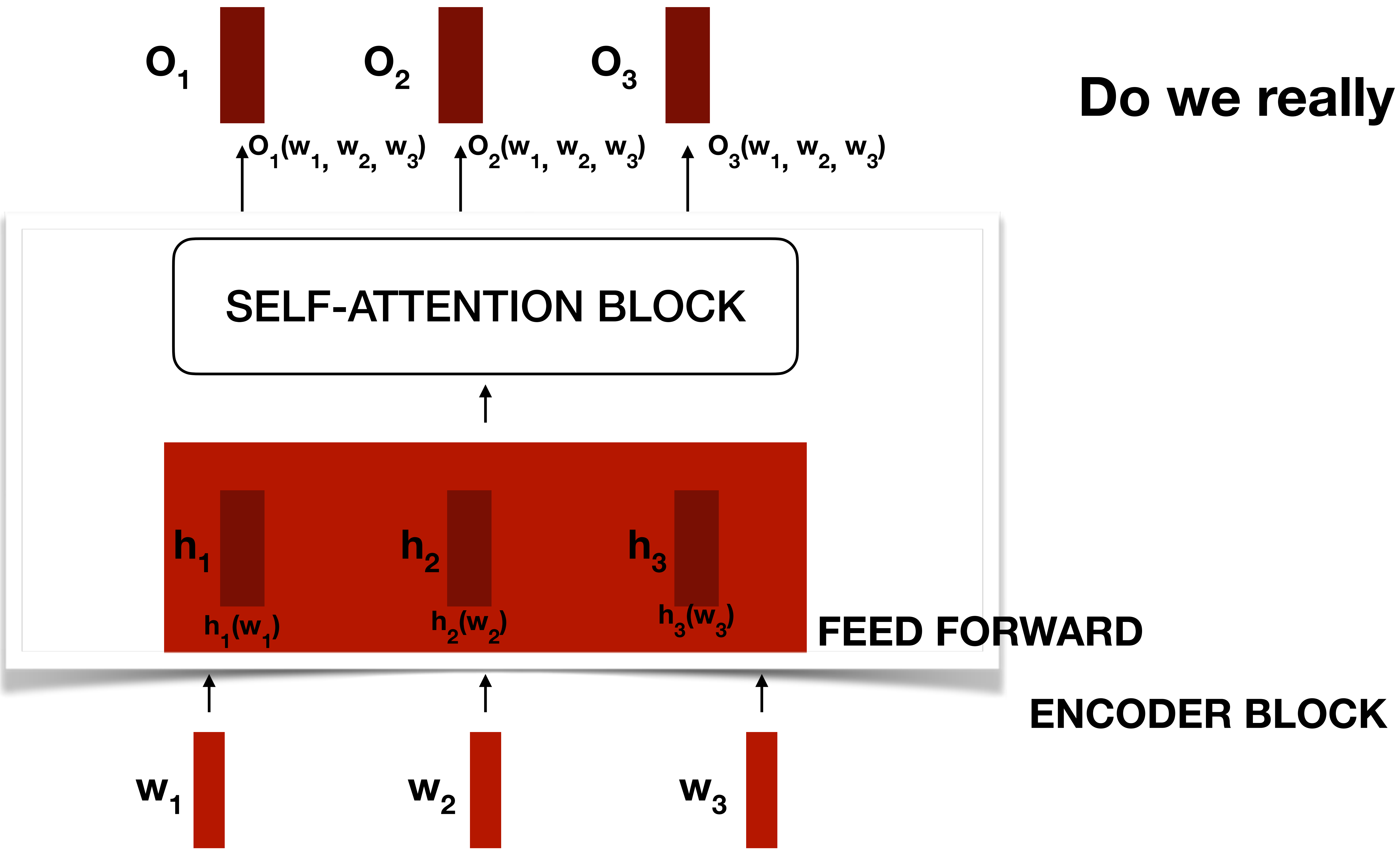
- **Self-Attention**




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

- **Self-Attention**

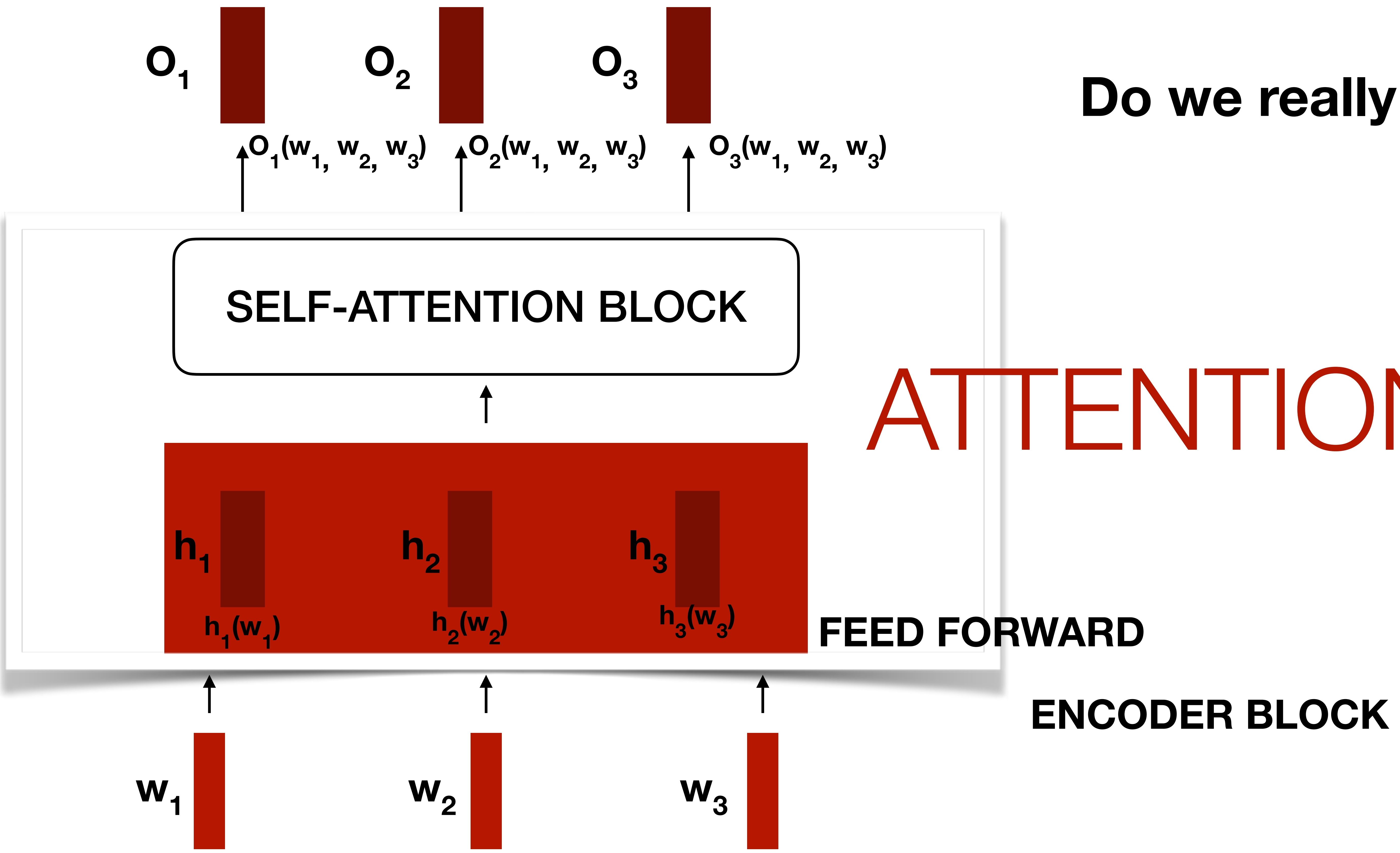



**Do we really need the LSTM to model sequences?**

**NO!**

# THE ATTENTION MECHANISM

- **Self-Attention**




**Do we really need the LSTM to model sequences?**

**NO!**

ATTENTION IS ALL YOU NEED!

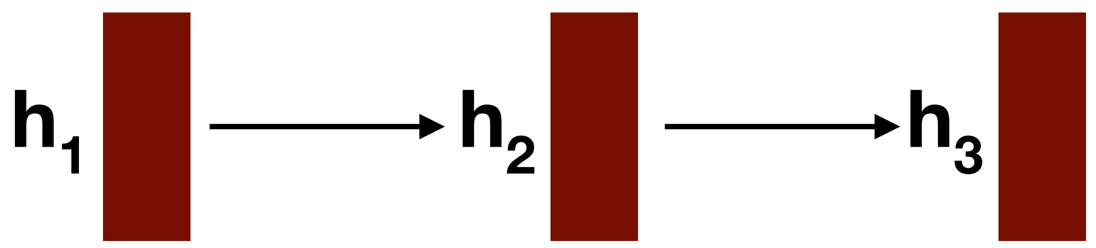
# THE ATTENTION MECHANISM

- Attention
- Self-Attention
- **Multi-Head Attention**

# THE ATTENTION MECHANISM

- **Multi-Head Attention**

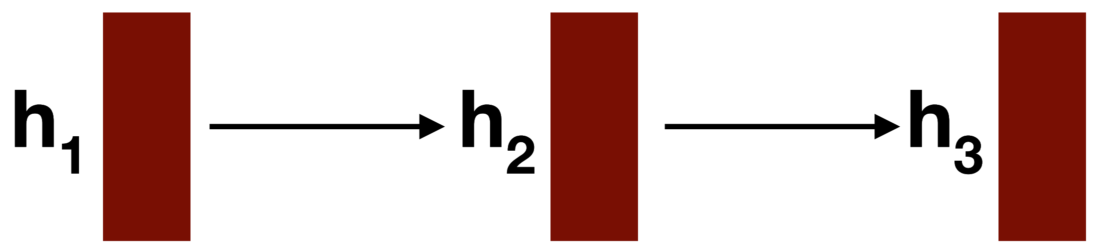
**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**



# THE ATTENTION MECHANISM

- **Multi-Head Attention**

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**




$k_1 = W_k h_1$ $k_2 = W_k h_2$ $k_3 = W_k h_3$

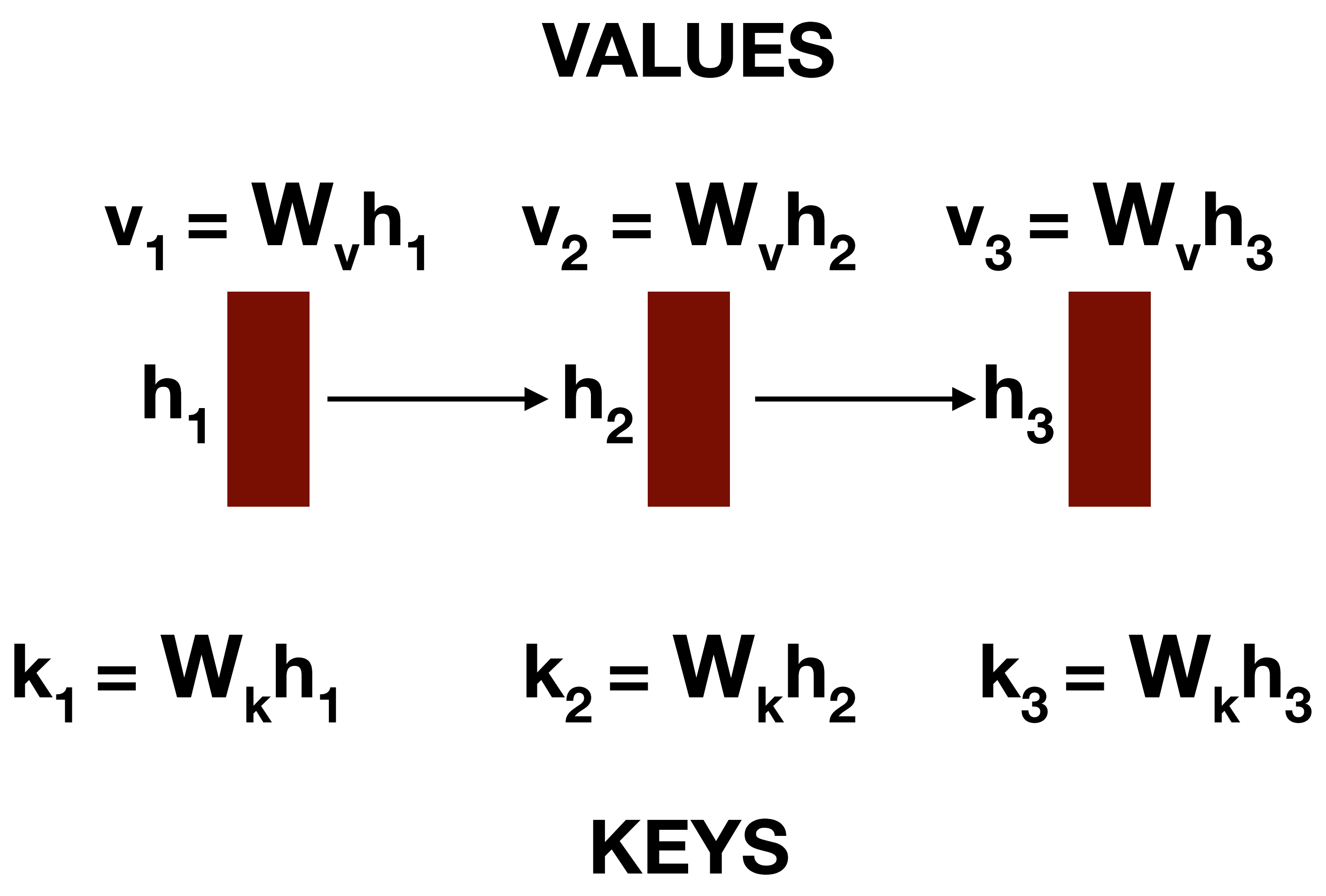
**KEYS**

$W_k$ **=: To convert input sequence to keys**

# THE ATTENTION MECHANISM

- **Multi-Head Attention**

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**



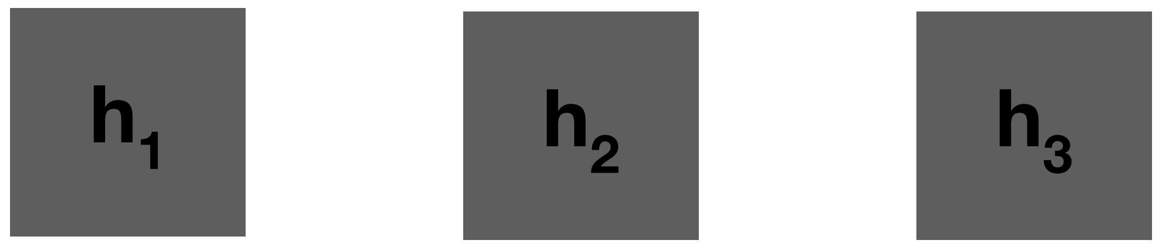

$W_k$ =: To convert input sequence to keys

$W_v$ =: To convert input sequence to values

# THE ATTENTION MECHANISM

- **Multi-Head Attention**

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**

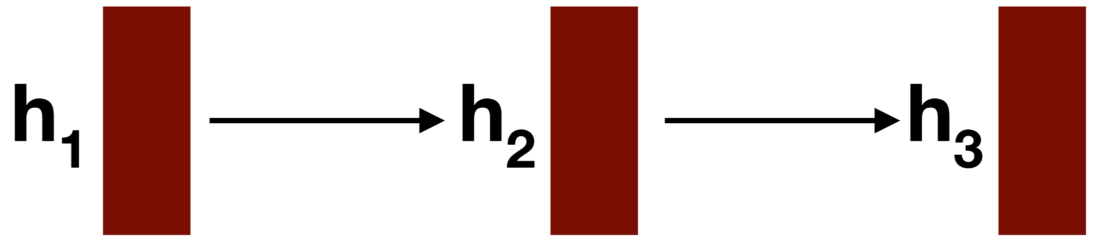



$q_1 = W_q h_1$ $q_2 = W_q h_2$ $q_3 = W_q h_3$

**QUERIES**

**VALUES**

$v_1 = W_v h_1$ $v_2 = W_v h_2$ $v_3 = W_v h_3$




$k_1 = W_k h_1$ $k_2 = W_k h_2$ $k_3 = W_k h_3$

**KEYS**

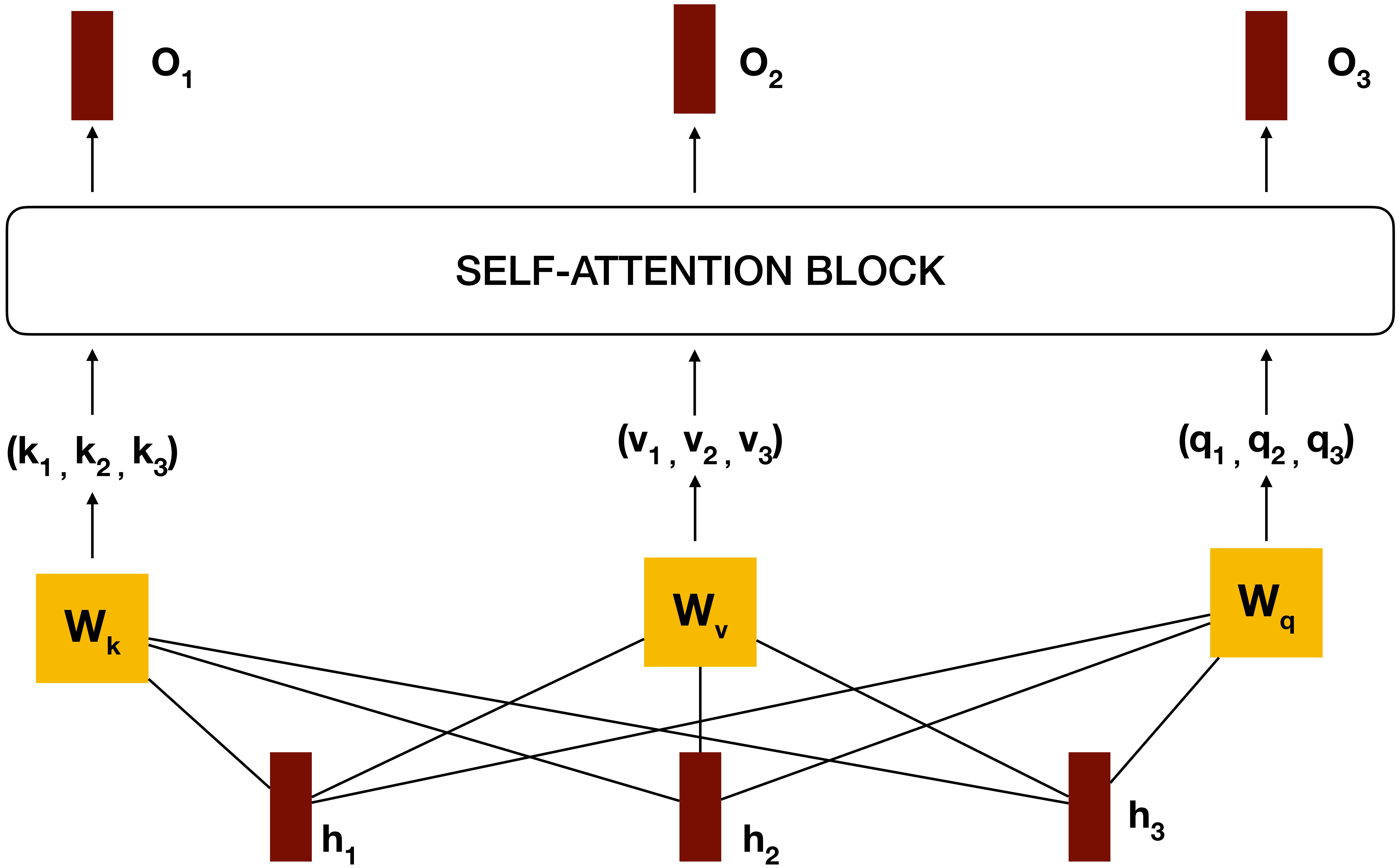
$W_k$ **=: To convert input sequence to keys**

$W_v$ **=: To convert input sequence to values**

$W_q$**=: To convert input sequence to values**

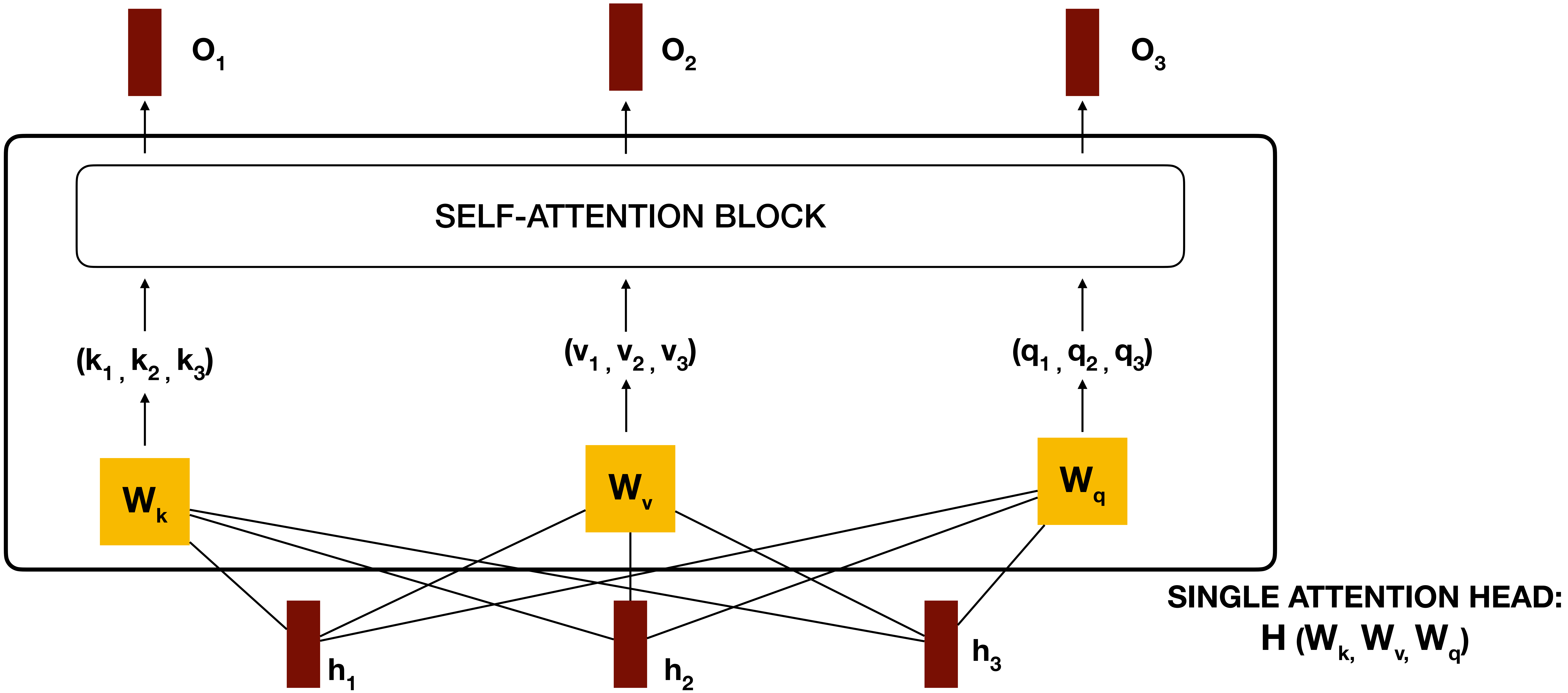
# THE ATTENTION MECHANISM

- **Multi-Head Attention**



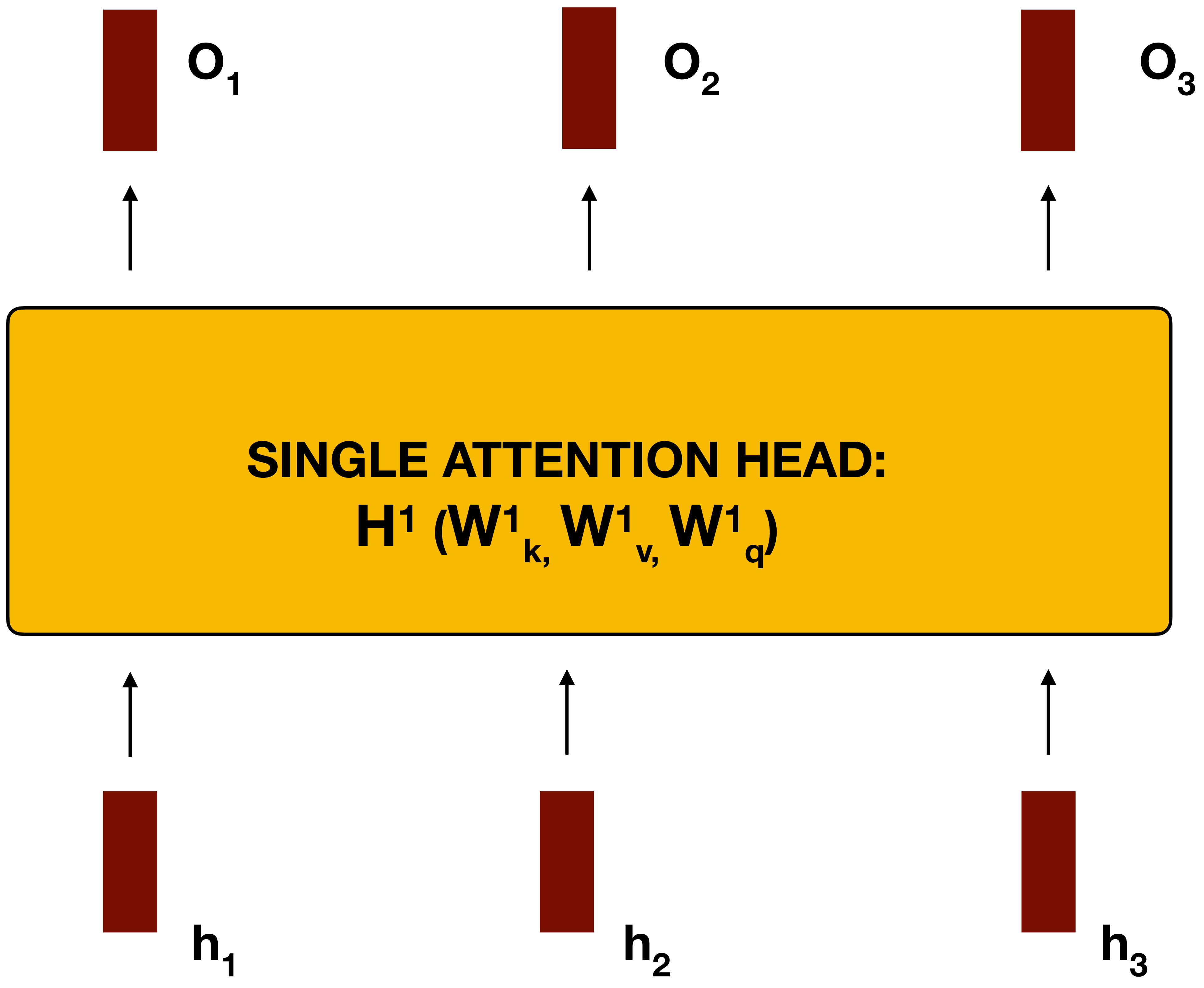
# THE ATTENTION MECHANISM

- **Multi-Head Attention**



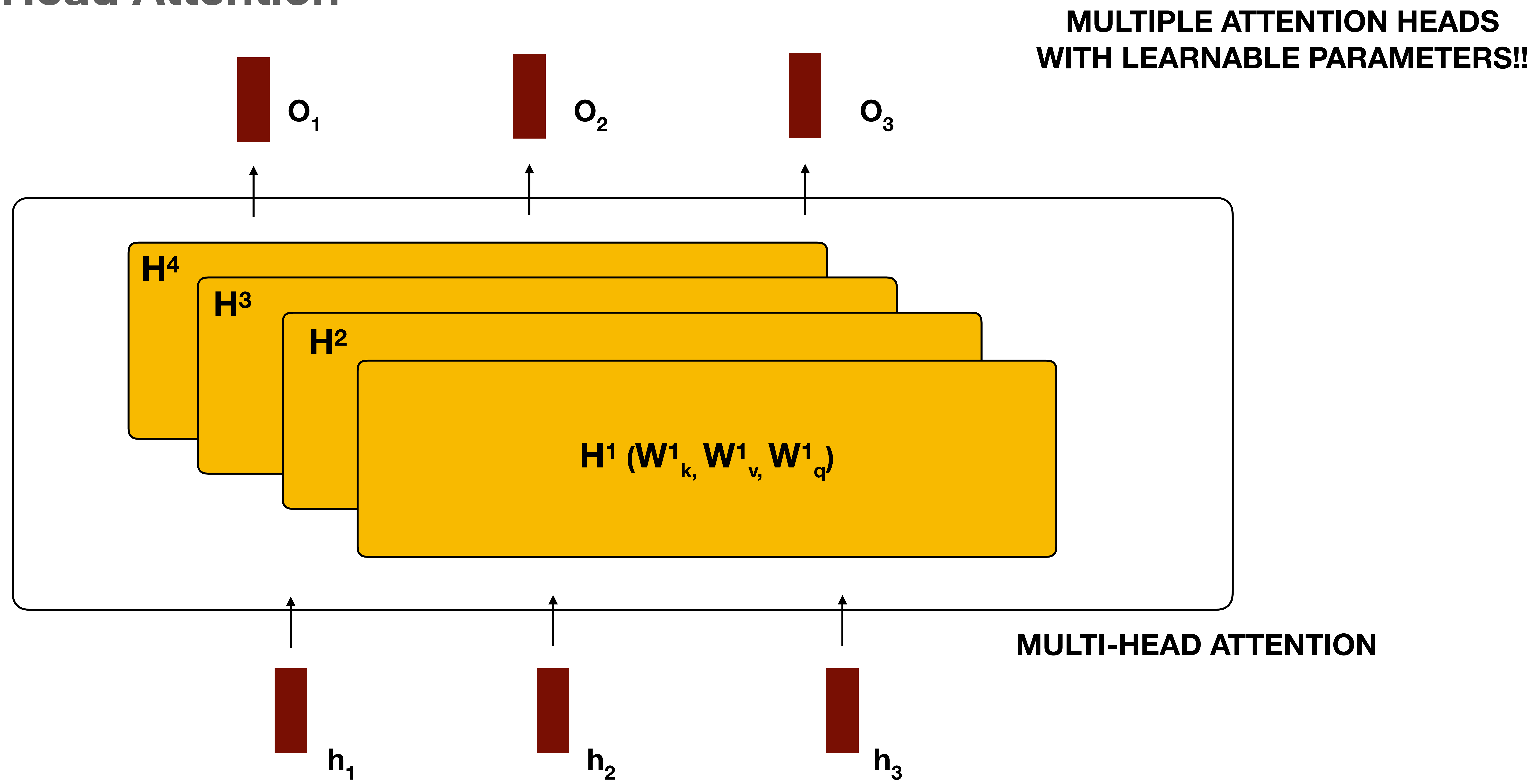
# THE ATTENTION MECHANISM

- **Multi-Head Attention**



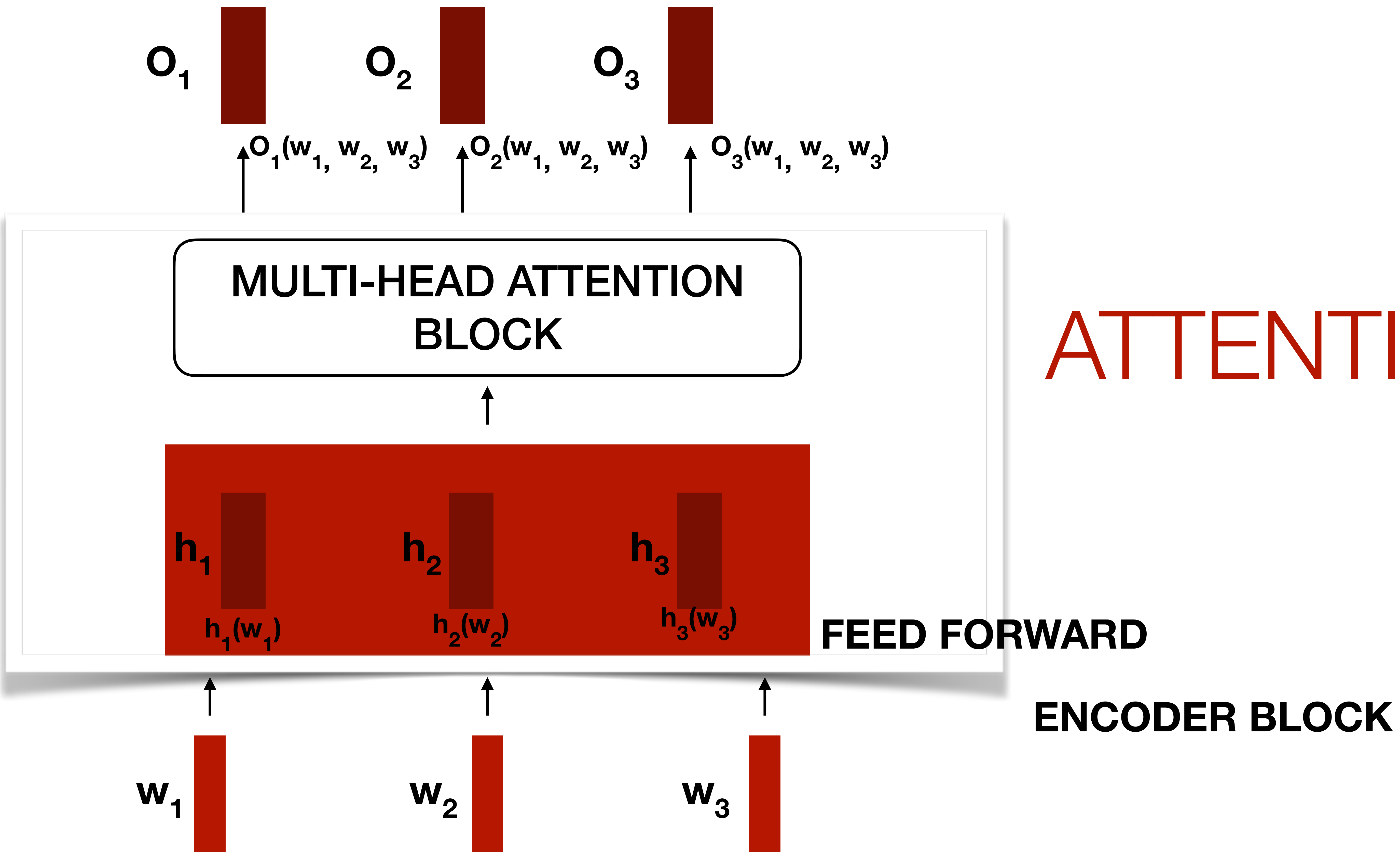
# THE ATTENTION MECHANISM

- **Multi-Head Attention**



# THE ATTENTION MECHANISM

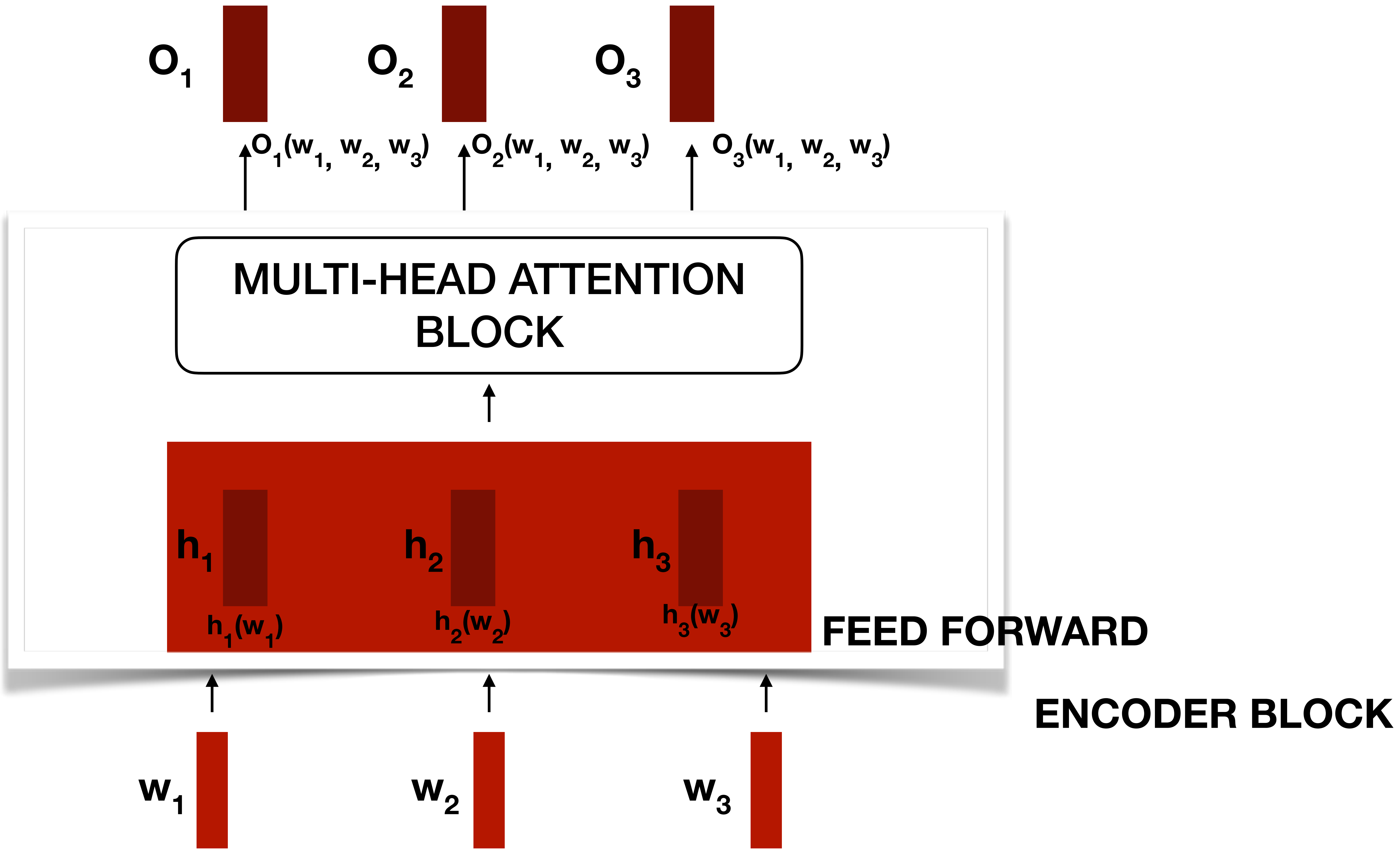
- **Multi-Head Attention**



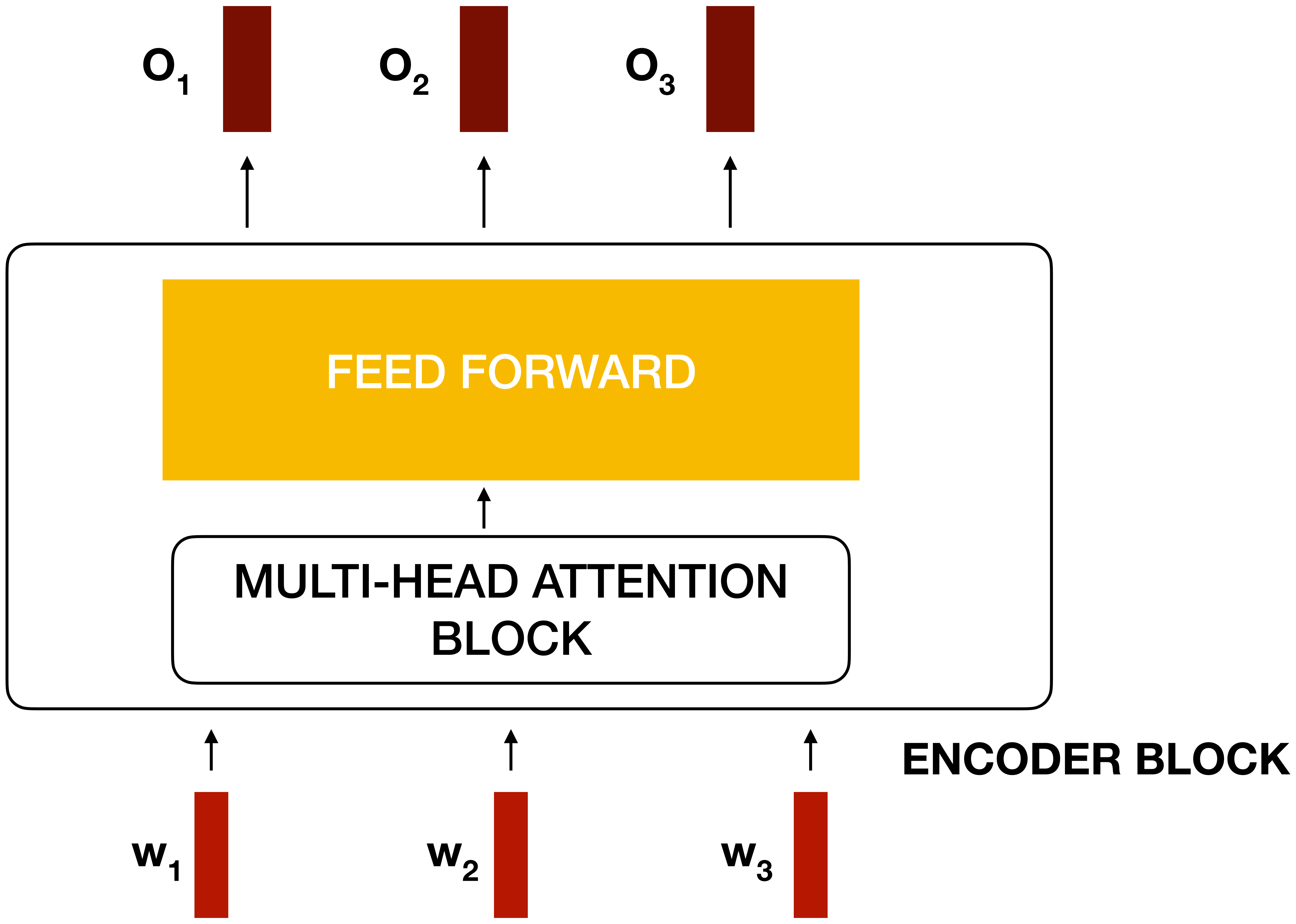

ATTENTION IS ALL YOU NEED!

# THE TRANSFORMER ARCHITECTURE

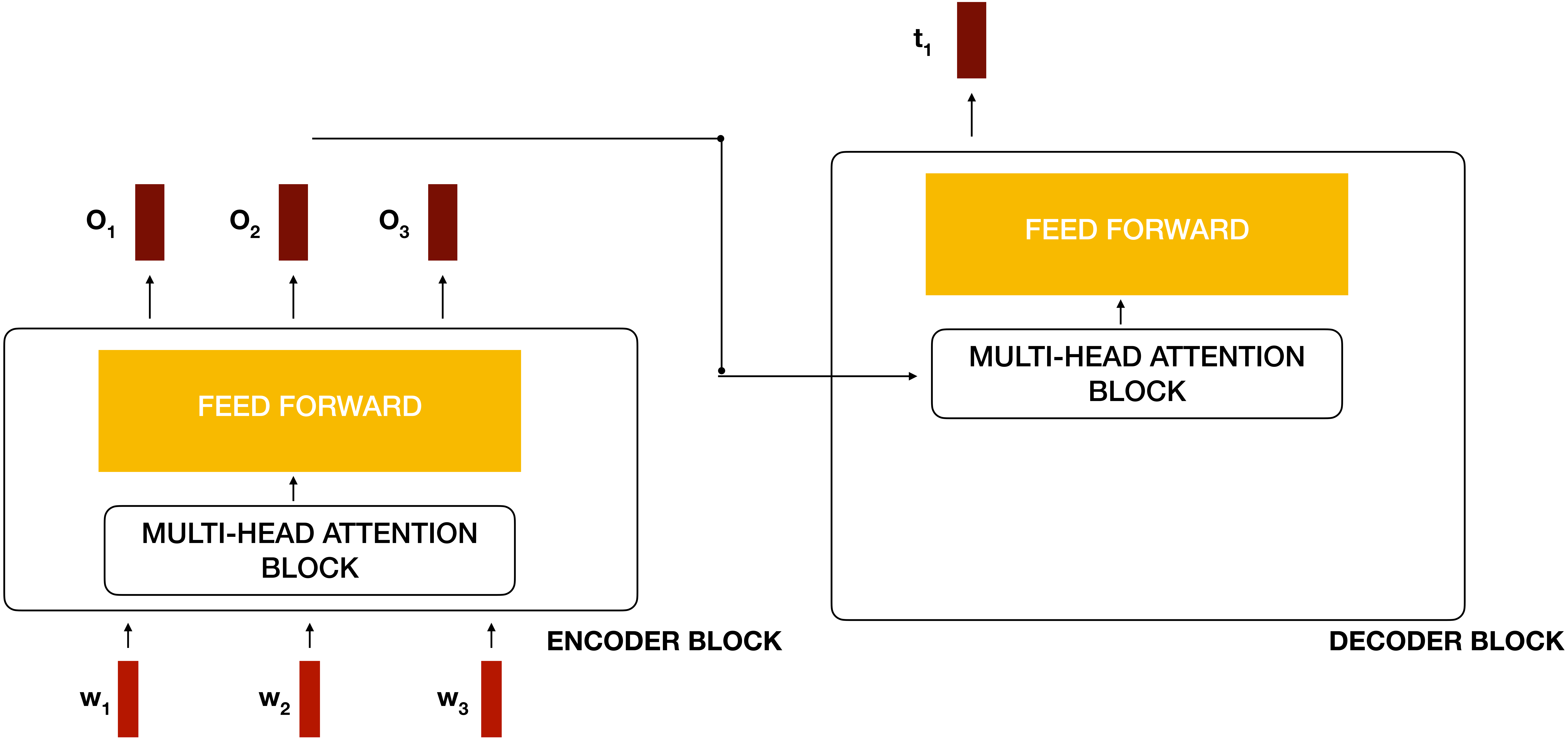
# THE TRANSFORMER ARCHITECTURE



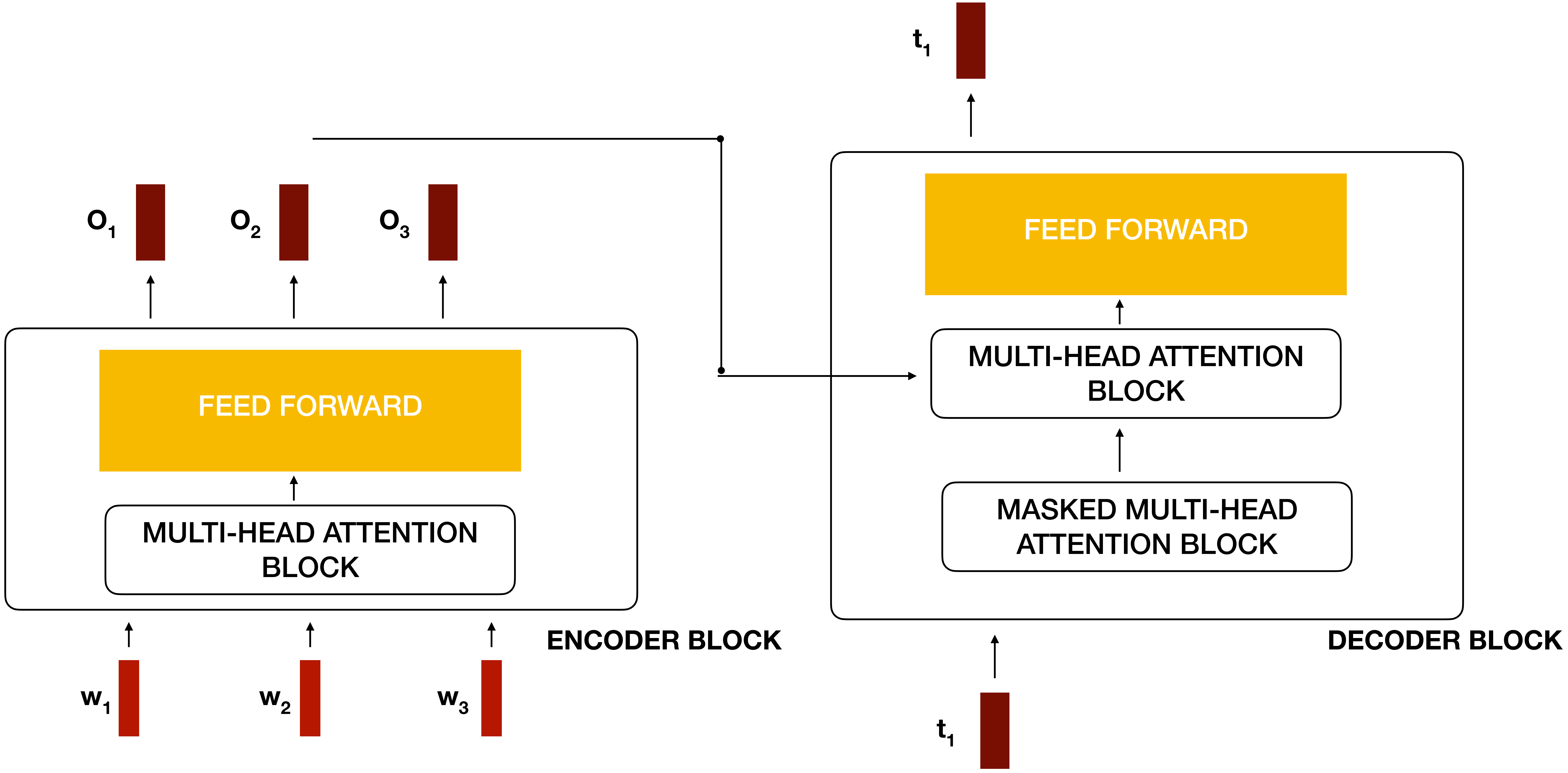
# THE TRANSFORMER ARCHITECTURE



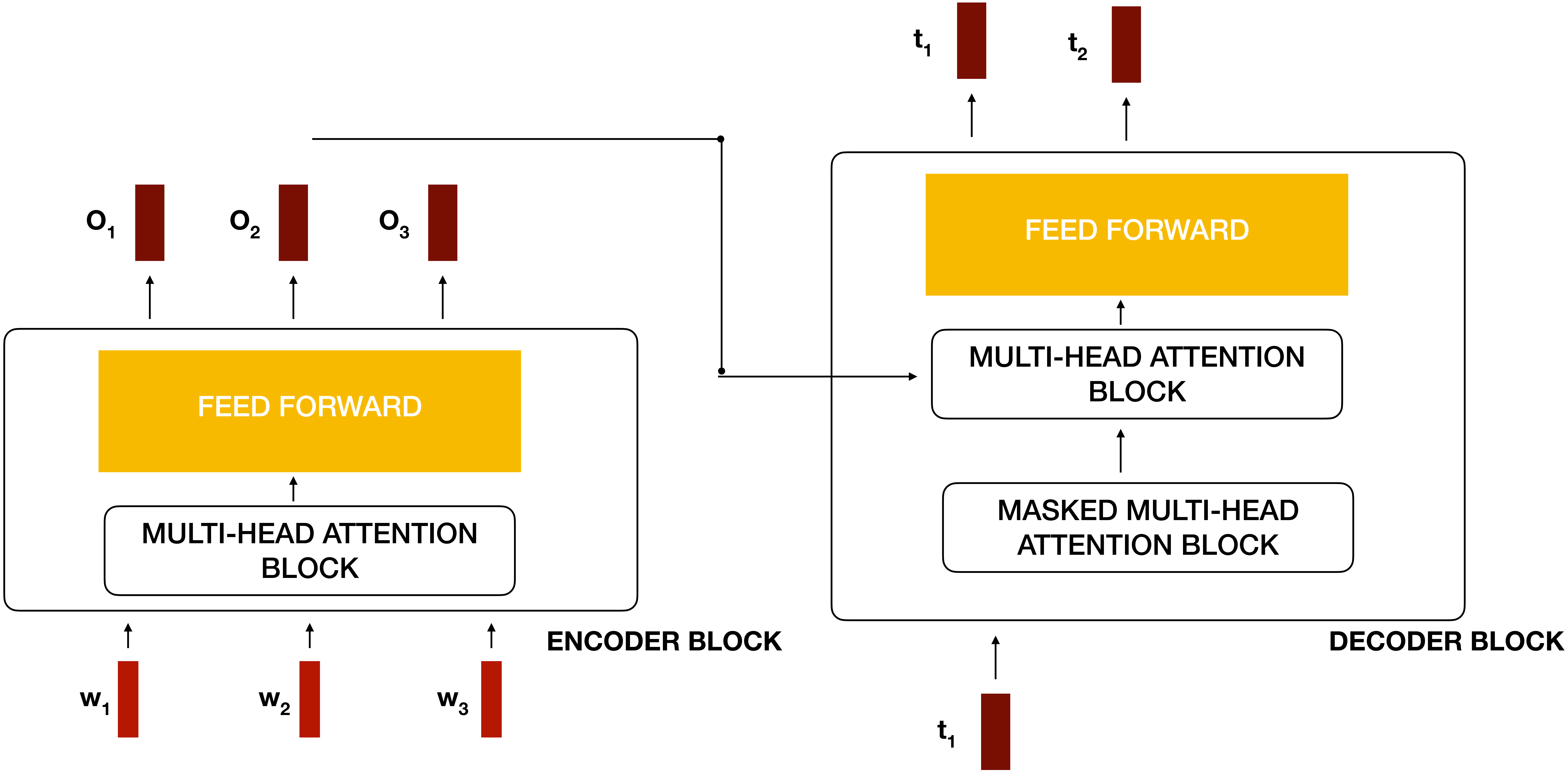
# THE TRANSFORMER ARCHITECTURE



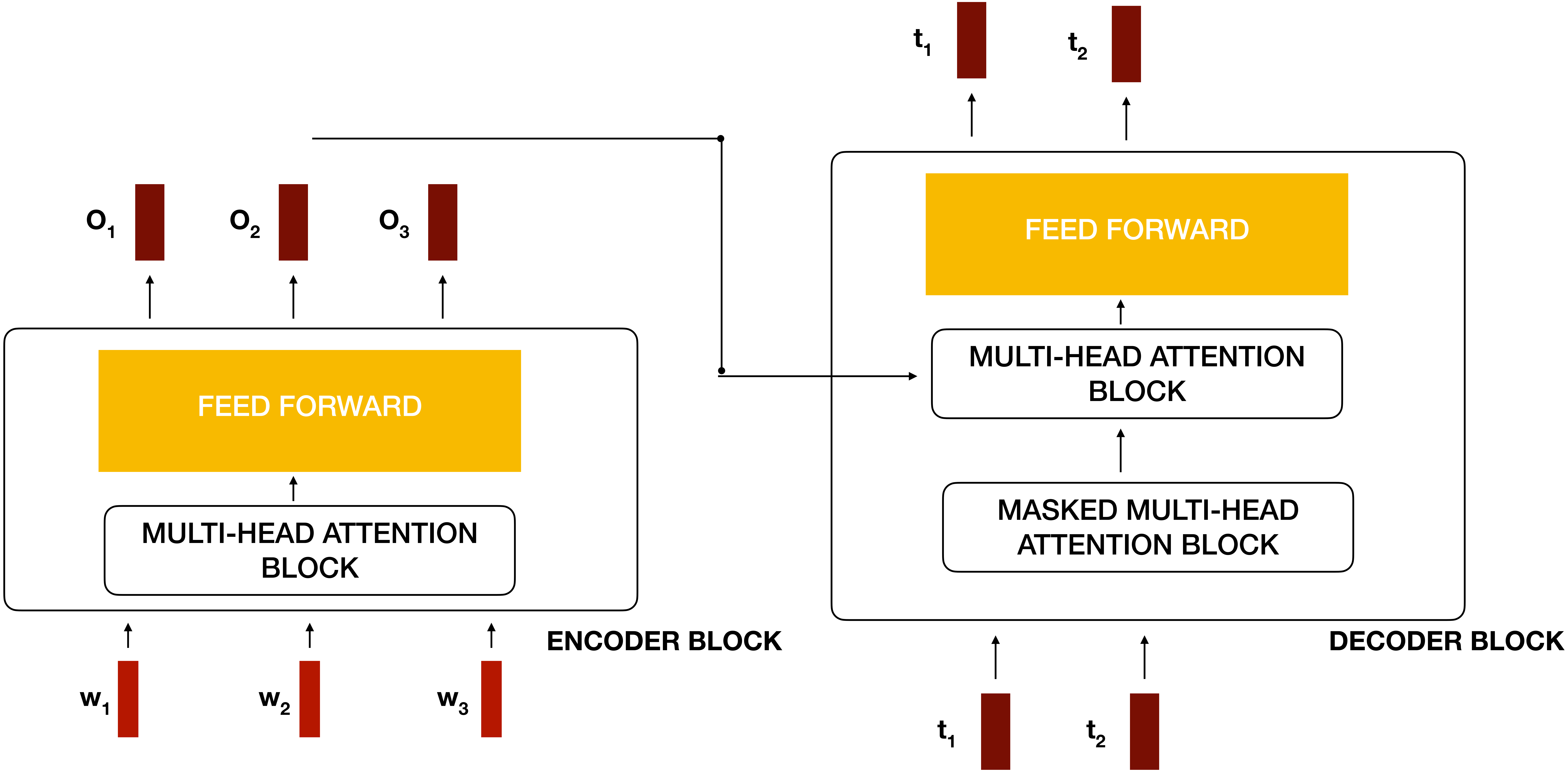
# THE TRANSFORMER ARCHITECTURE



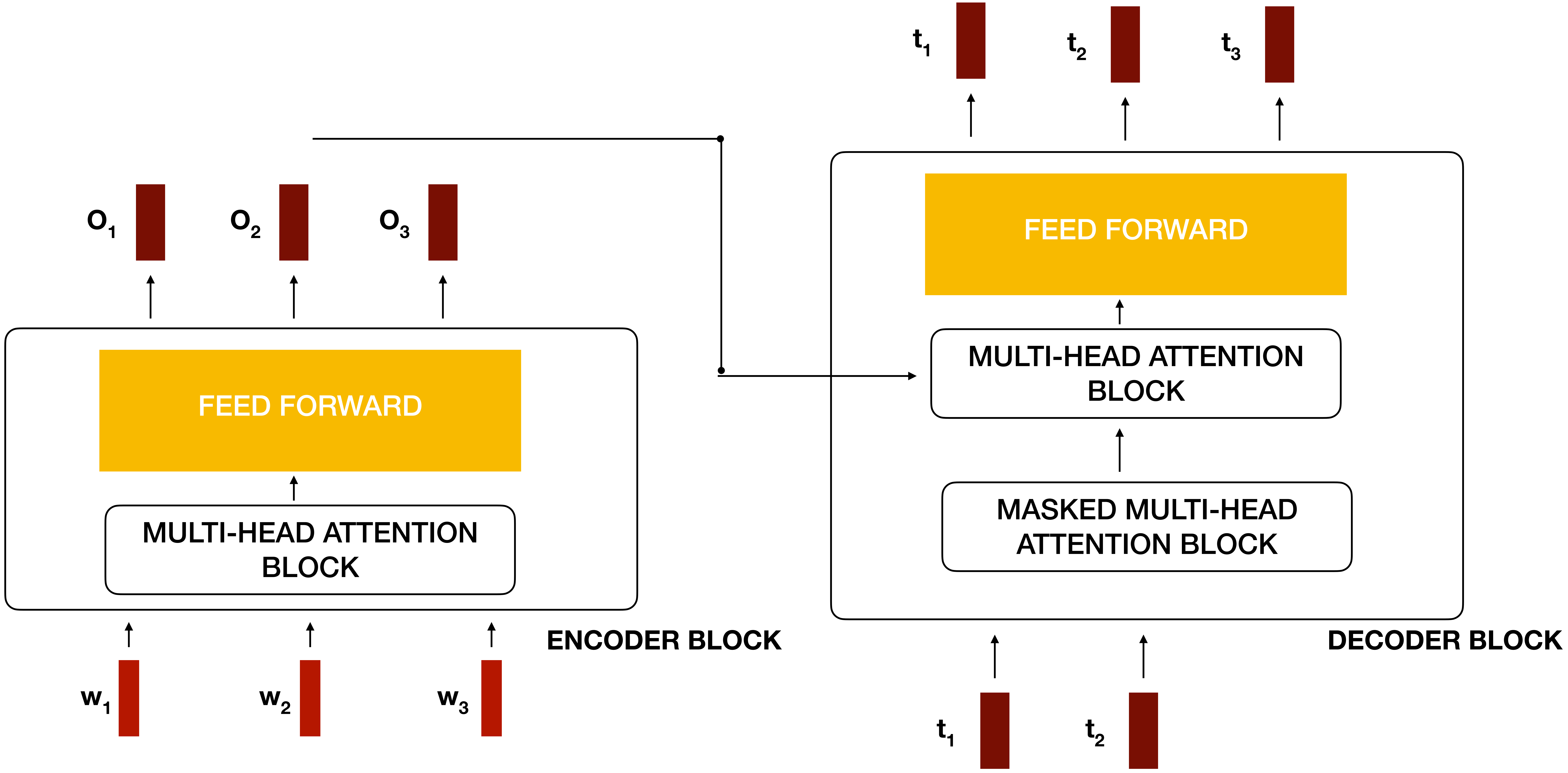
# THE TRANSFORMER ARCHITECTURE



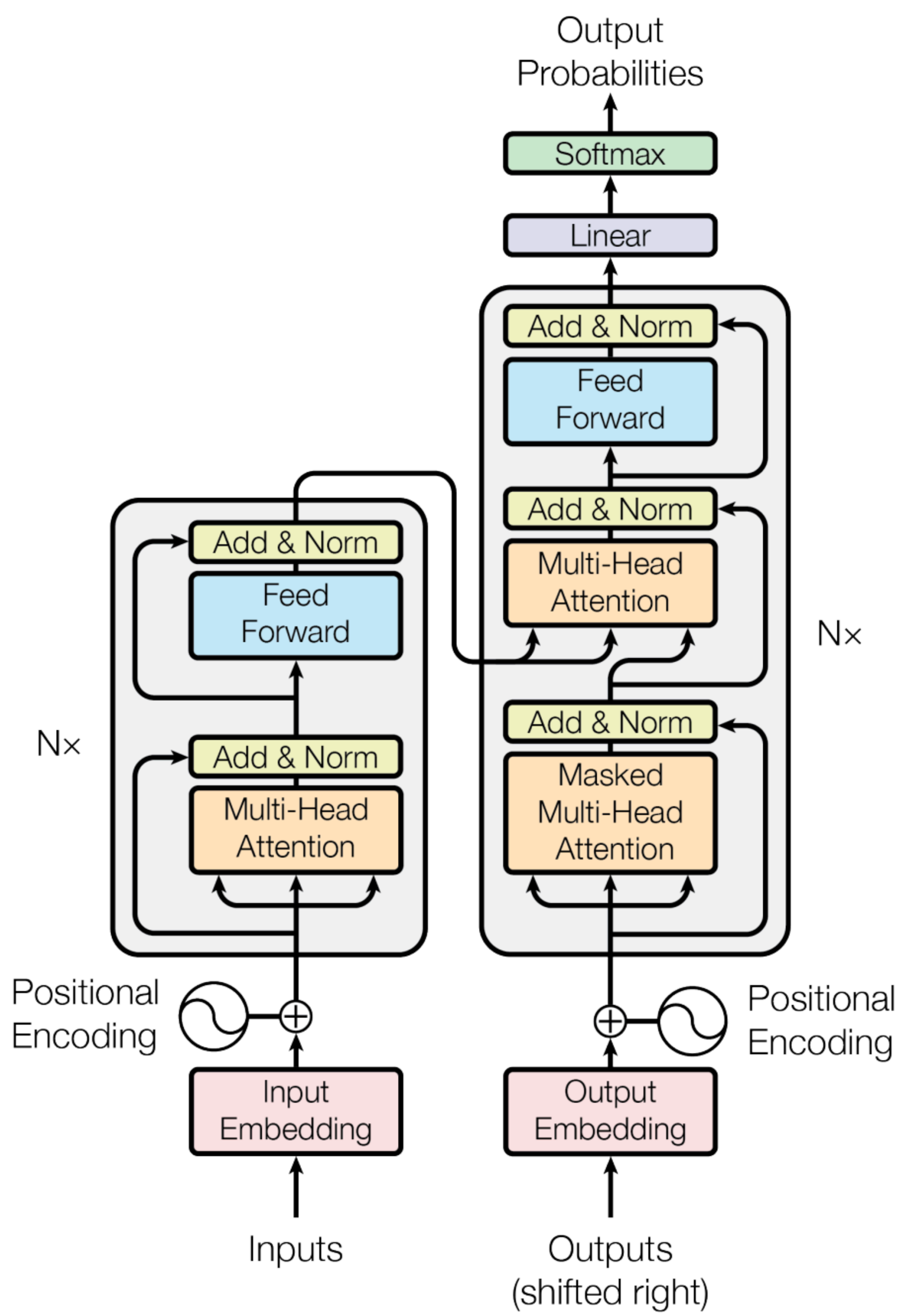
# THE TRANSFORMER ARCHITECTURE



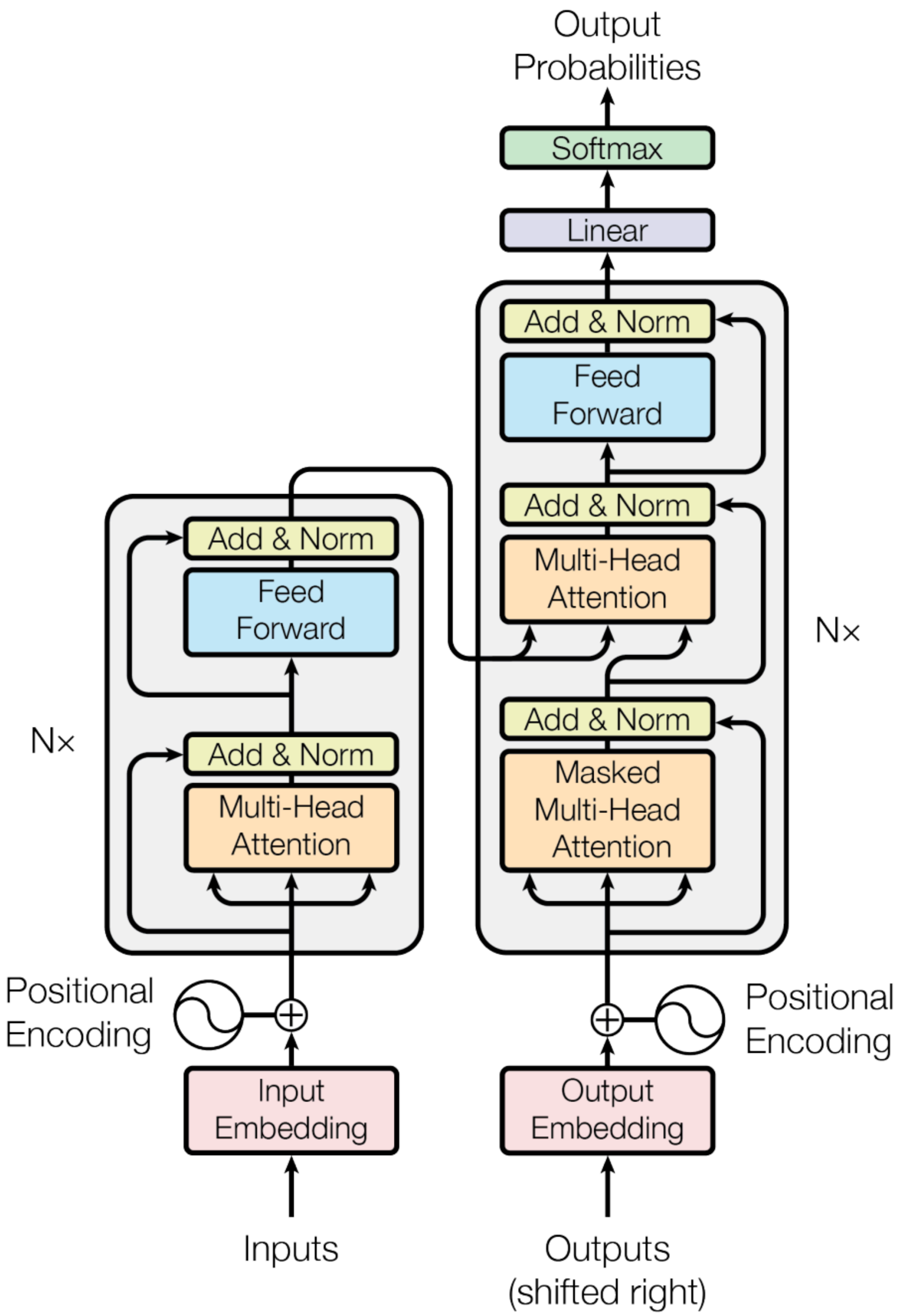
# THE TRANSFORMER ARCHITECTURE



# THE TRANSFORMER ARCHITECTURE

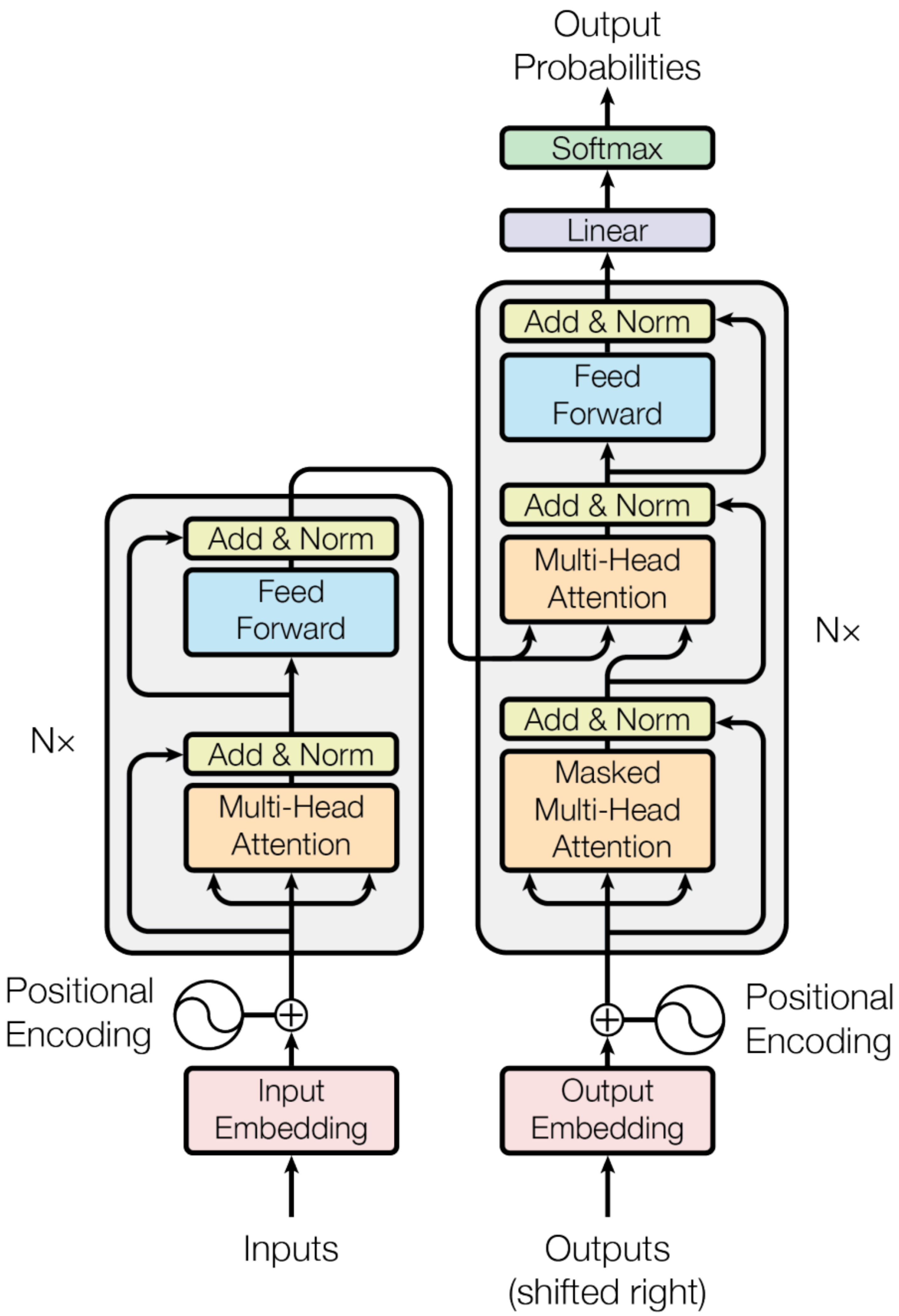


# THE TRANSFORMER ARCHITECTURE




**Multiple stacked encoder and decoder blocks!**

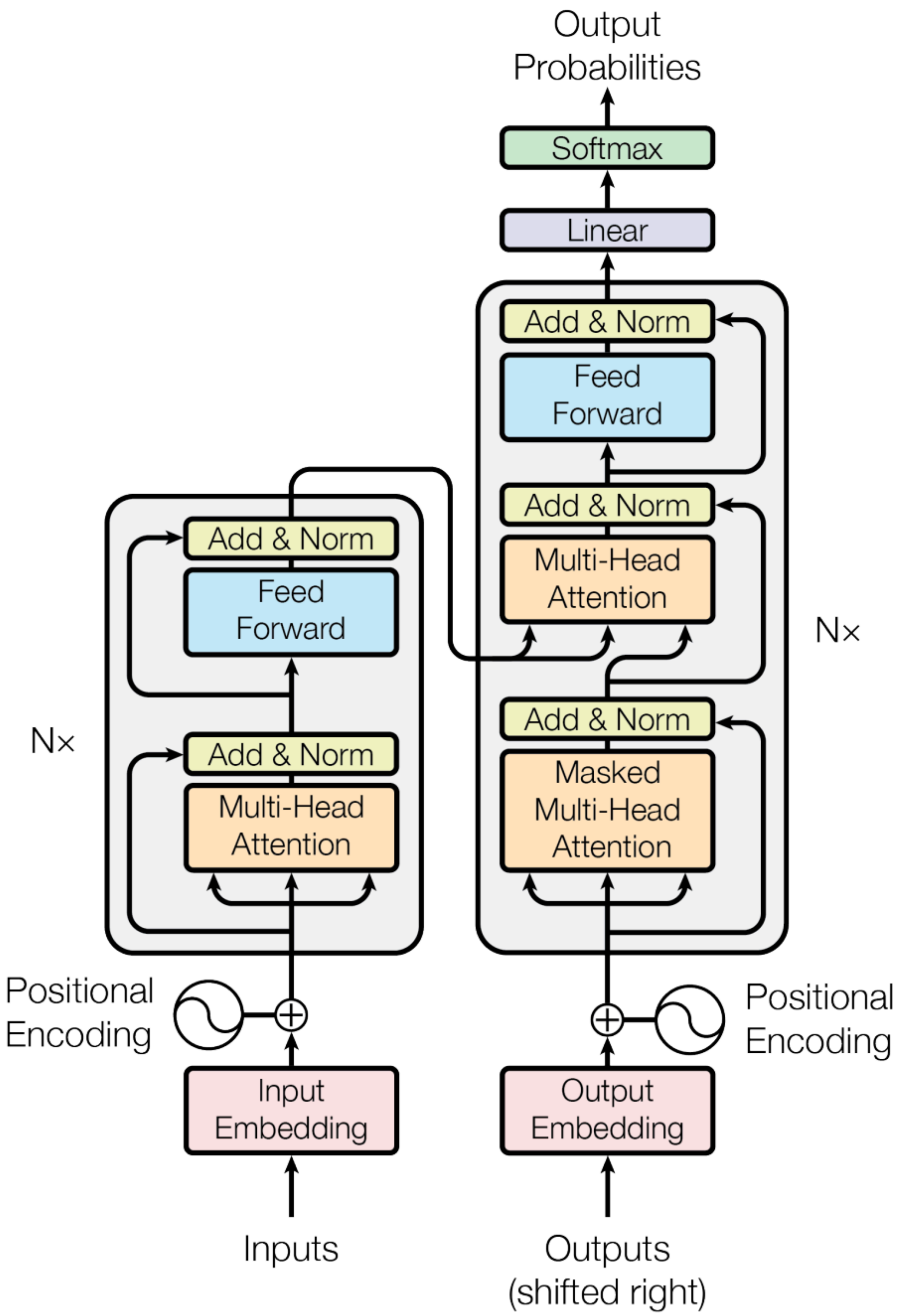
# THE TRANSFORMER ARCHITECTURE




**Multiple stacked encoder and decoder blocks!**

**Layer Normalization!**

# THE TRANSFORMER ARCHITECTURE



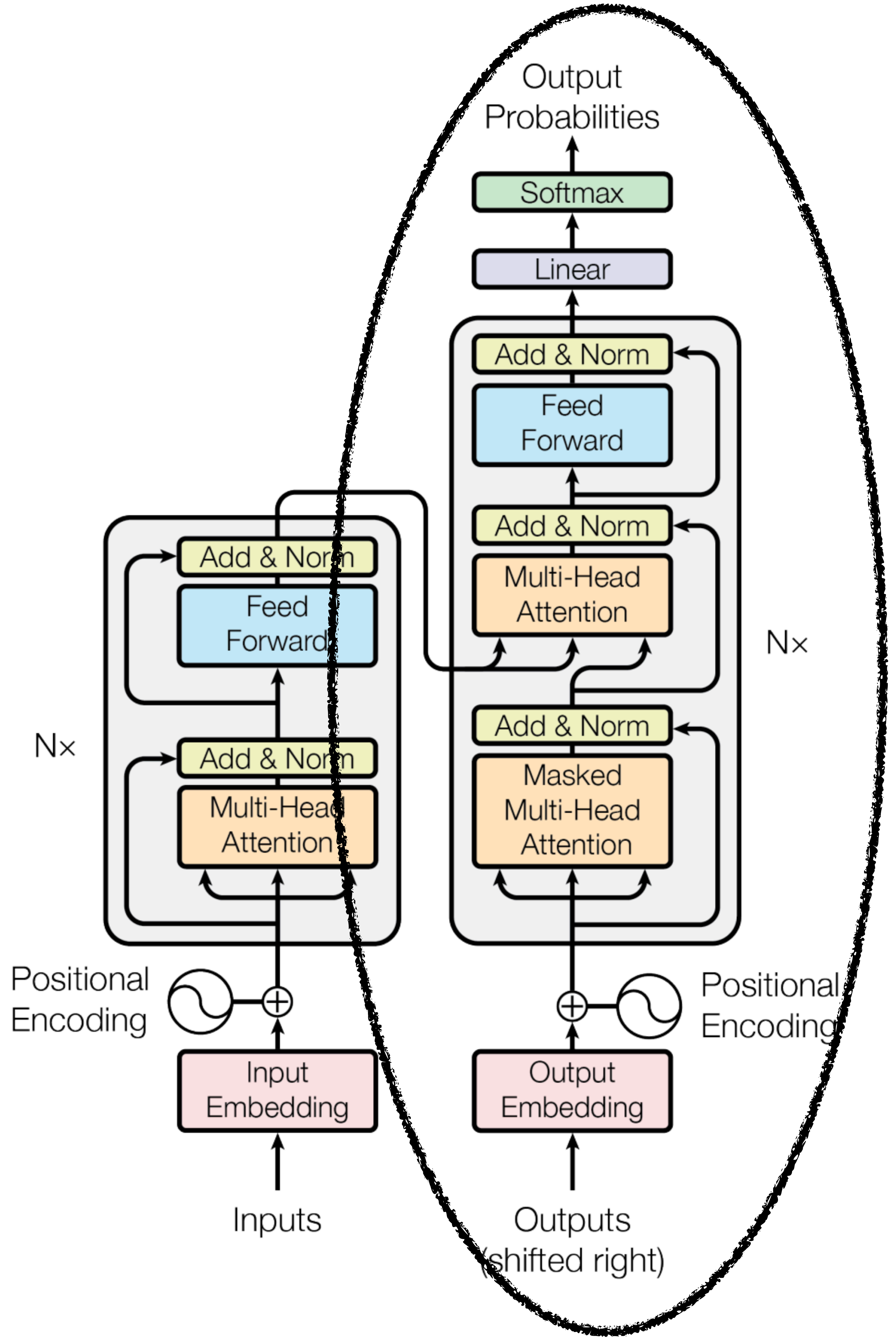

**Multiple stacked encoder and decoder blocks!**

**Layer Normalization!**

**Positional Embeddings!**

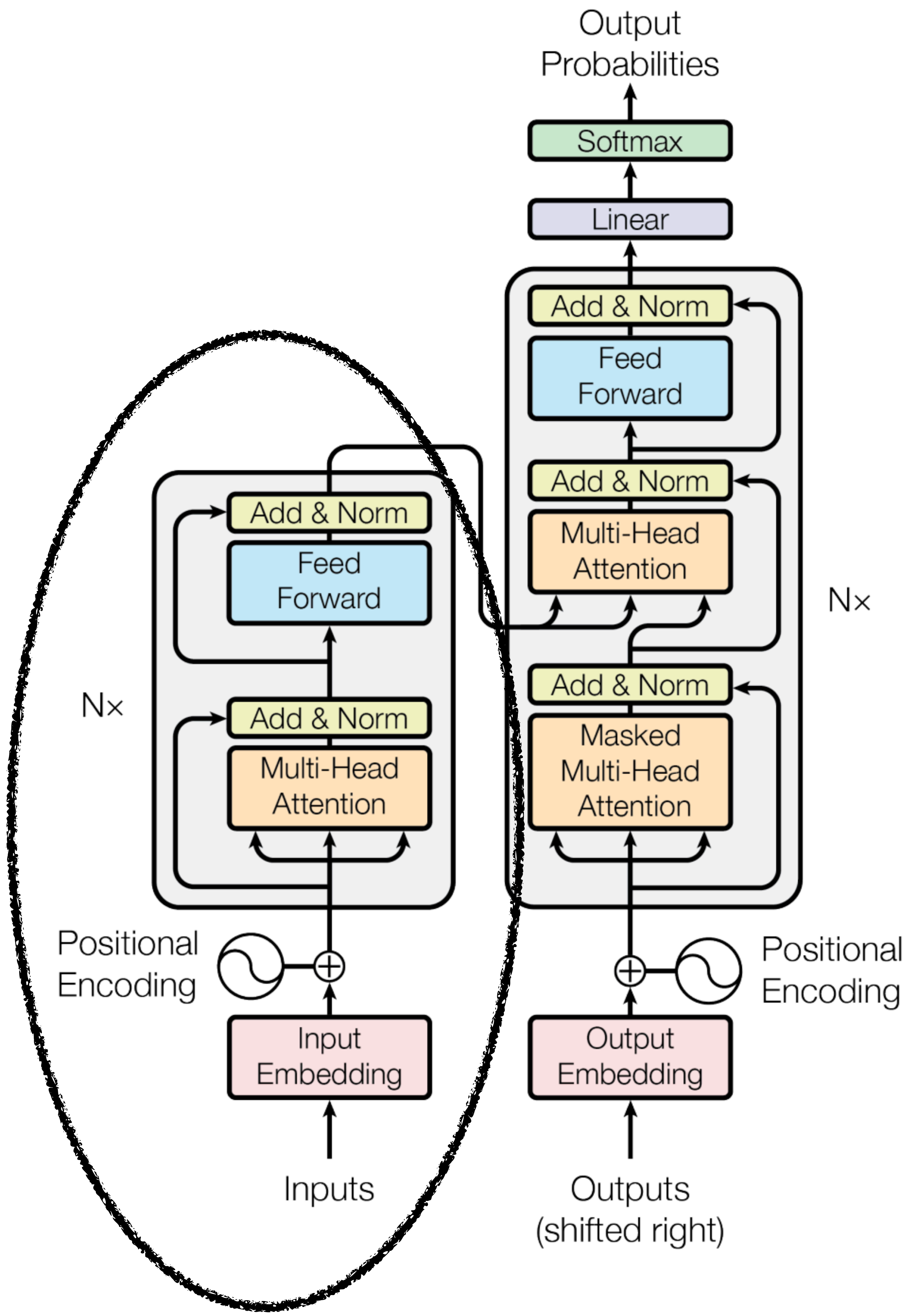
# GPT ARCHITECTURE

# GPT ARCHITECTURE



# BERT ARCHITECTURE

# BERT ARCHITECTURE



# GRAPH NEURAL NETWORKS

# WHY GRAPH NEURAL NETWORKS?

# WHY GNN?

- There are many cases where we represent data as graphs.

# WHY GNN?

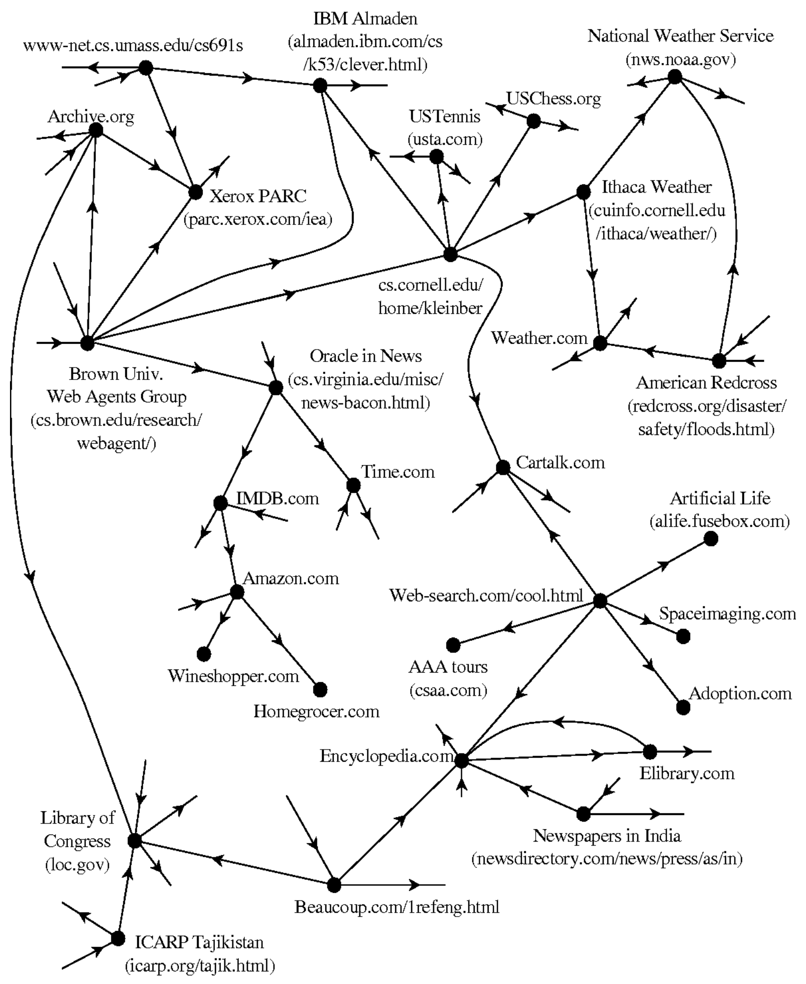
- There are many cases where we represent data as graphs.

**Social Networks**

# WHY GNN?

- There are many cases where we represent data as graphs.

**Social Networks**

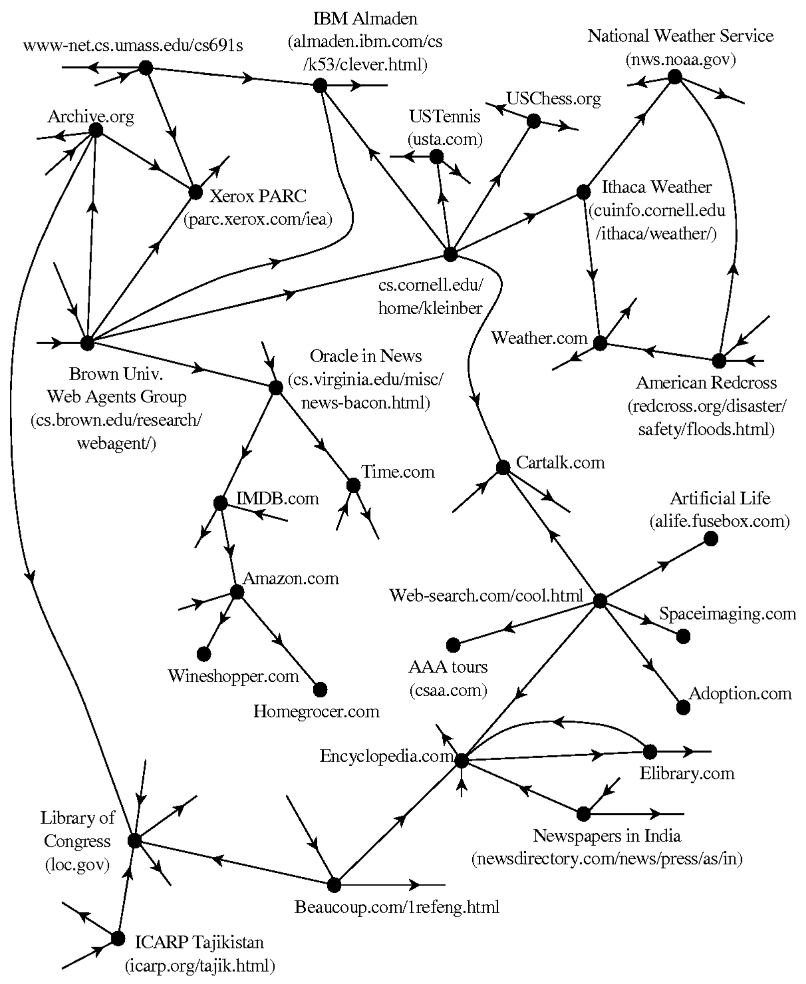



**World Wide Web or Citation Networks**

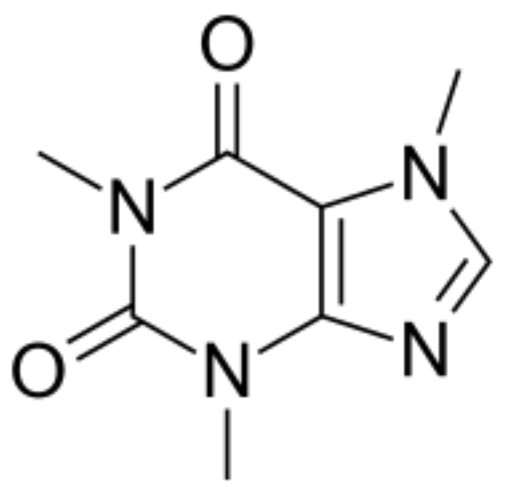
# WHY GNN?

- There are many cases where we represent data as graphs.

**Social Networks**




**World Wide Web or Citation Networks**

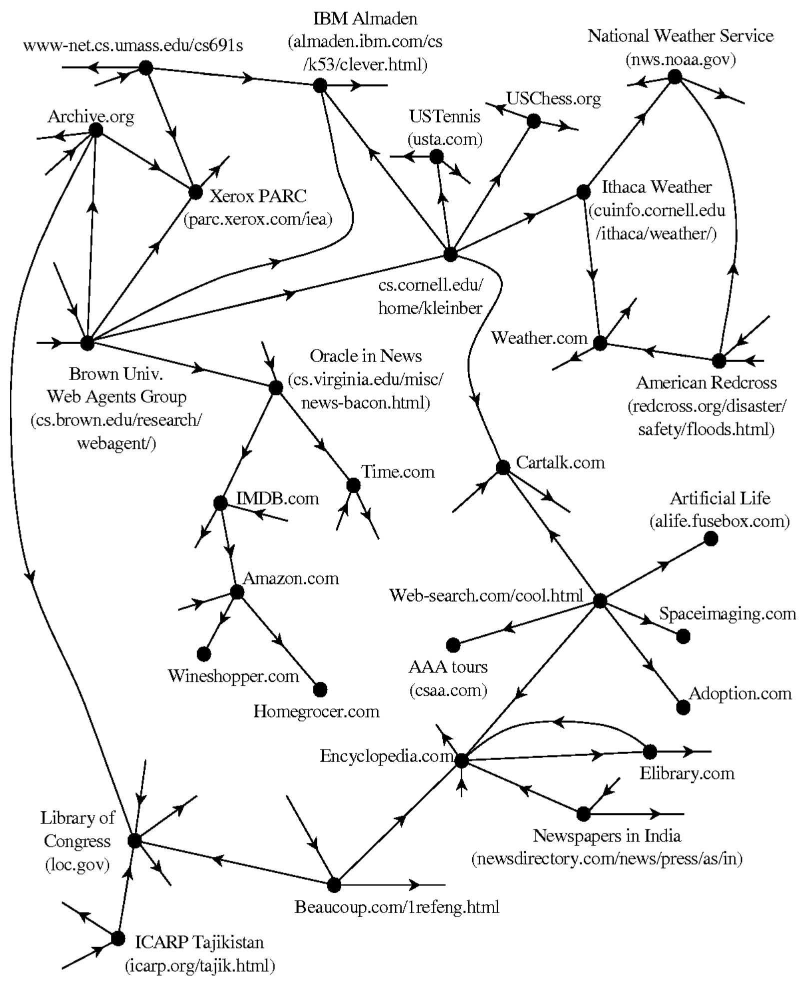



**Molecules**

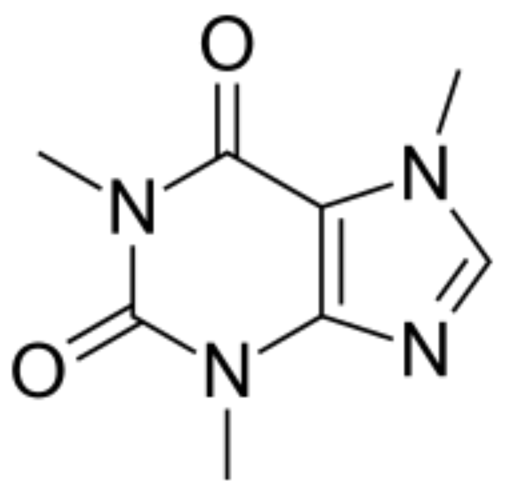
# WHY GNN?

- There are many cases where we represent data as graphs.

**Social Networks**




**World Wide Web or Citation Networks**



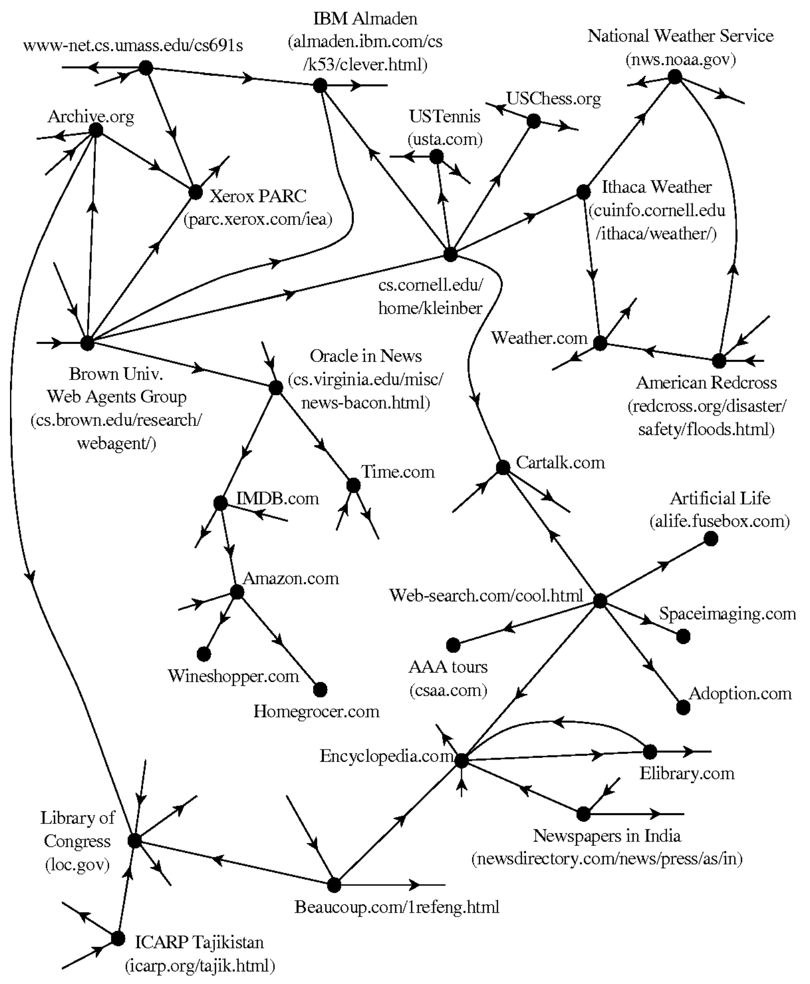

**Molecules**

Question: When can we model data as graphs?

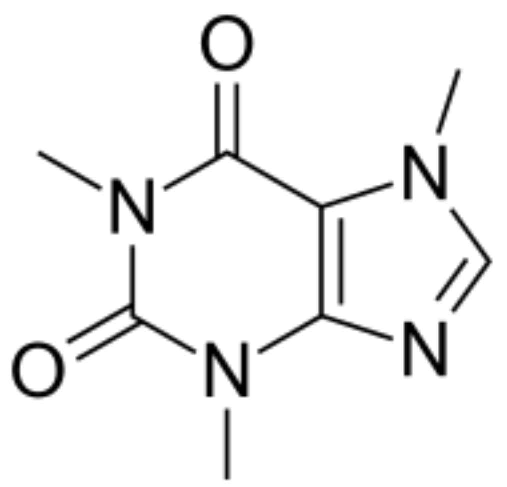
# WHY GNN?

- There are many cases where we represent data as graphs.

**Social Networks**




**World Wide Web or Citation Networks**

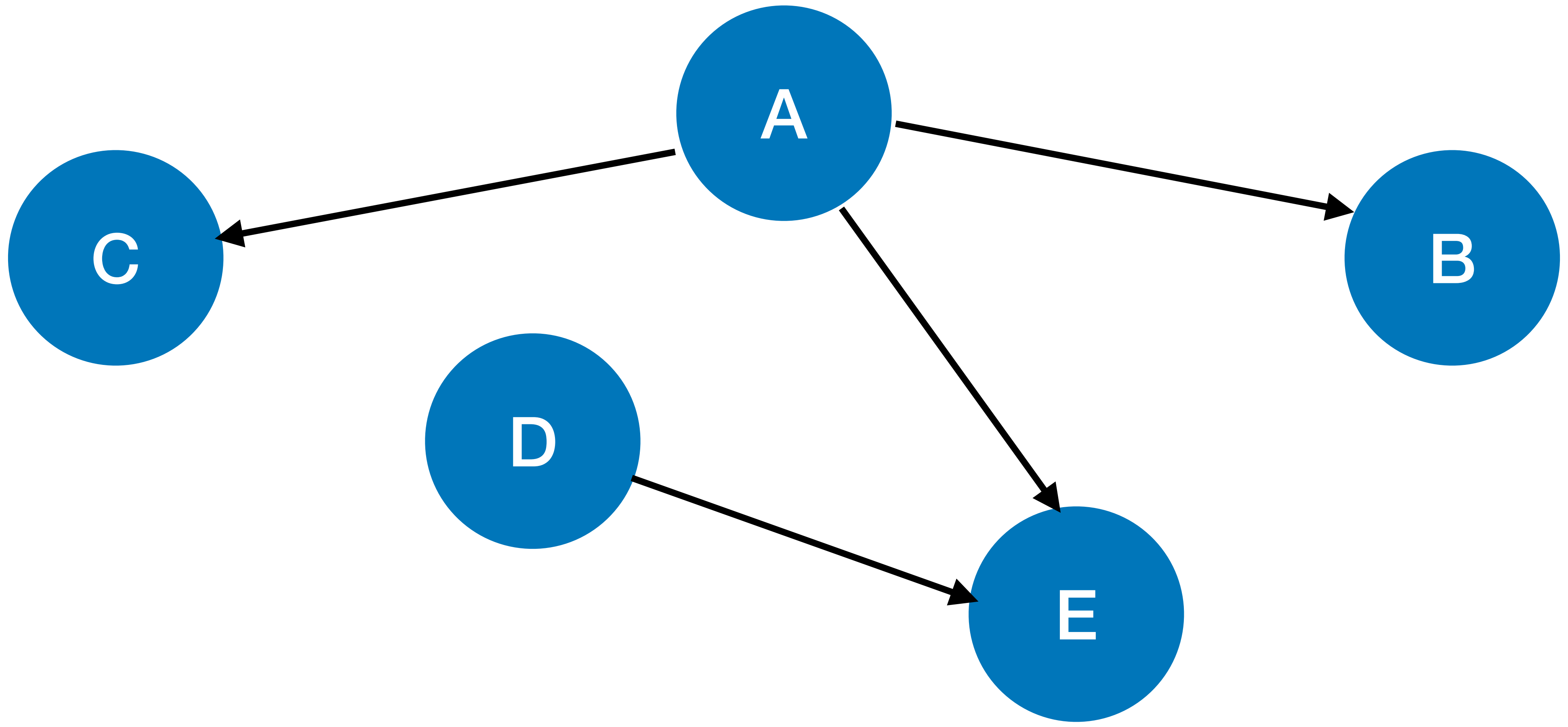



**Molecules**

When our domain has data points with a relational structure.

# WHY GNN?

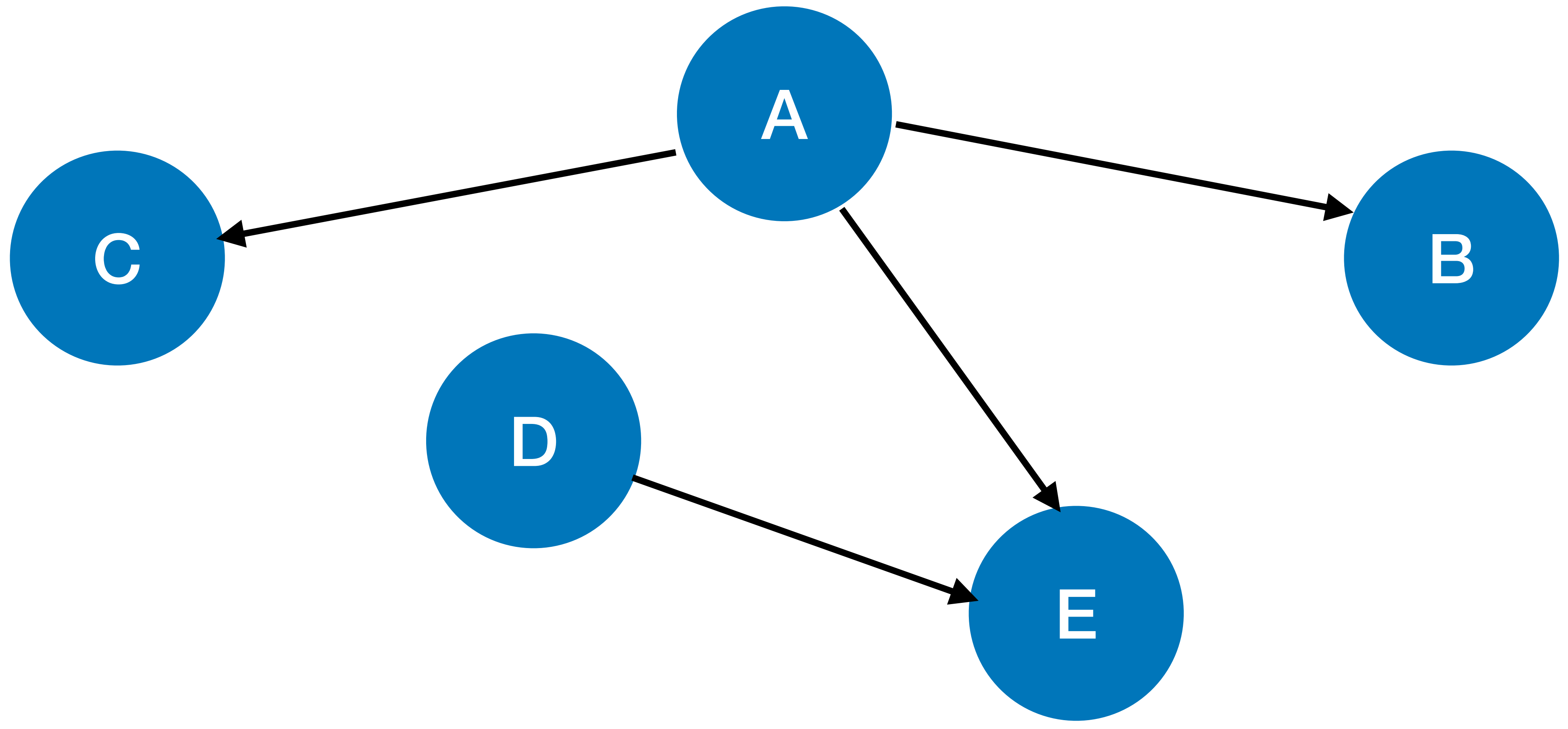
- There are many cases where we represent data as graphs.



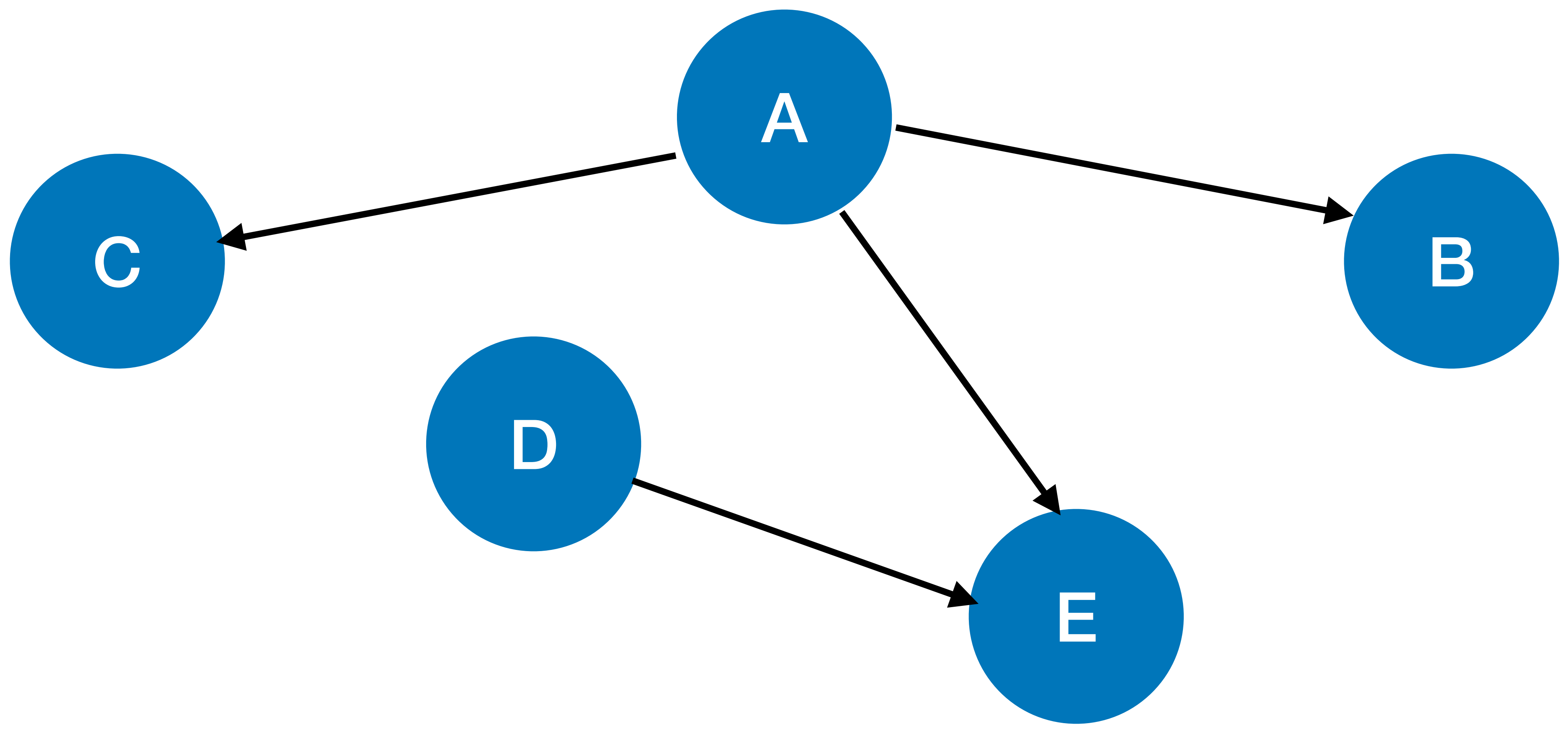

When our domain has data points with a relational structure.

# A QUICK INTRODUCTION TO GRAPHS

# A QUICK INTRODUCTION TO GRAPHS

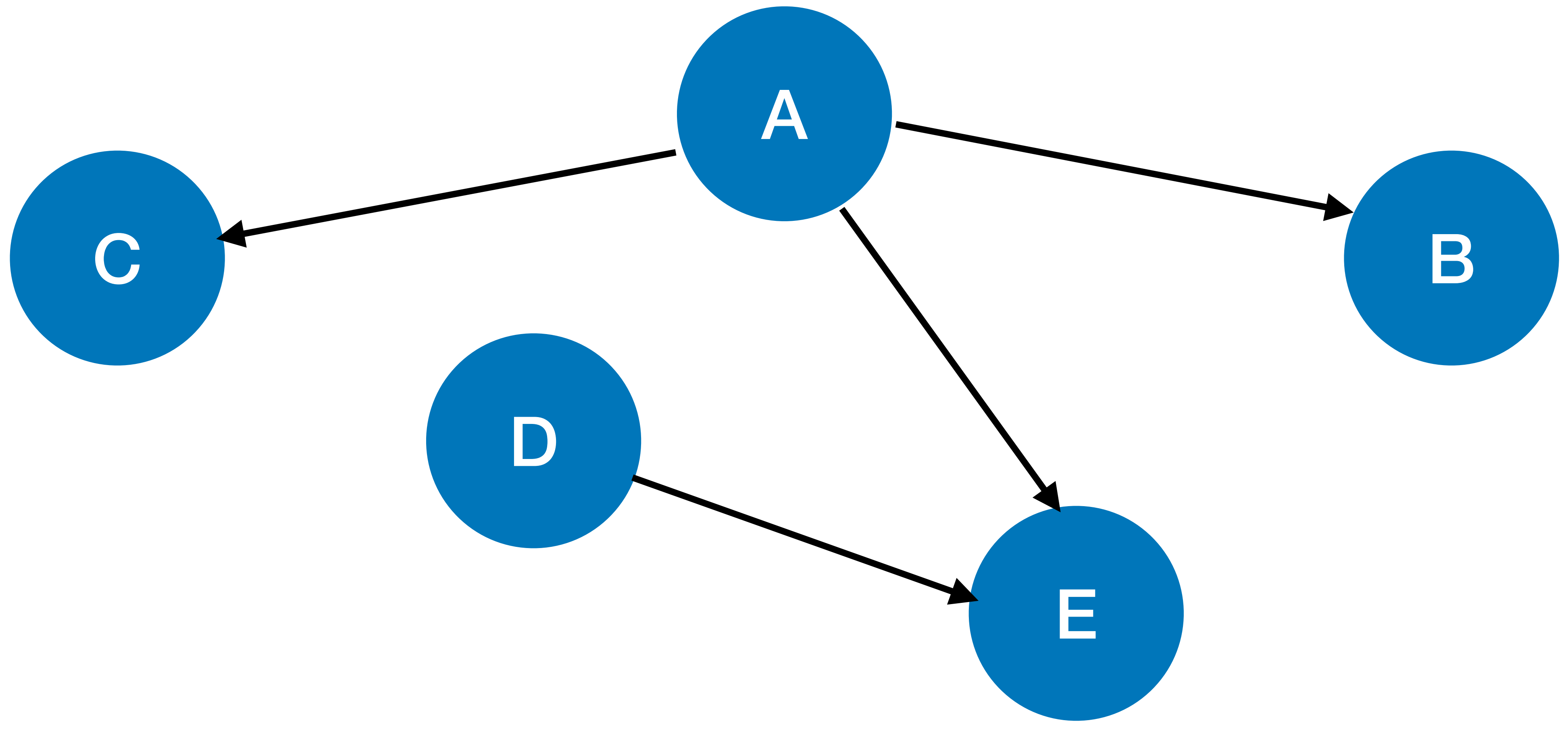


# A QUICK INTRODUCTION TO GRAPHS



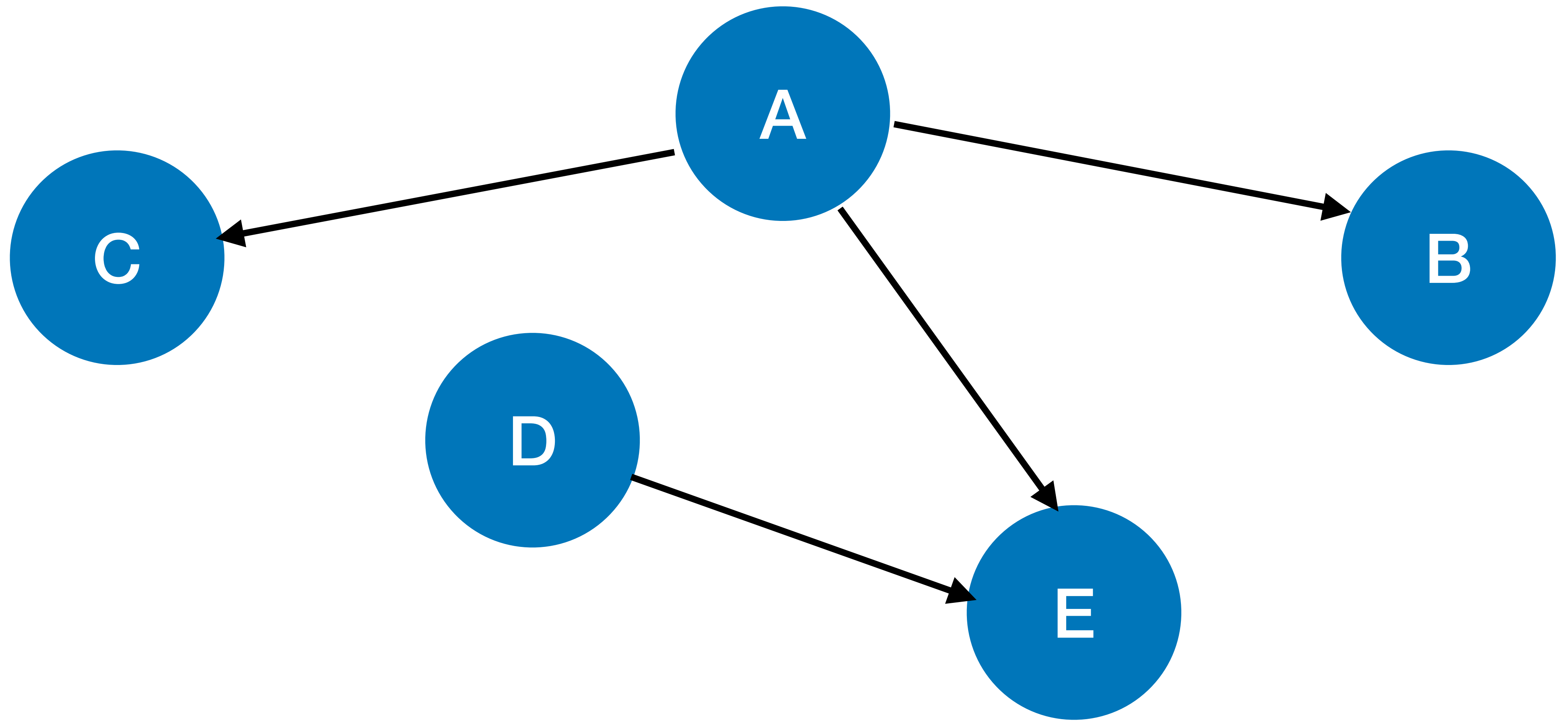

- OBJECTS / DATA POINTS : Nodes, Vertices (V)
- INTERACTIONS / RELATIONS : Links, Edges (E)
- SYSTEM : Network, Graphs (G)
- NODE ATTRIBUTES : Feature vectors (X)

# A QUICK INTRODUCTION TO GRAPHS



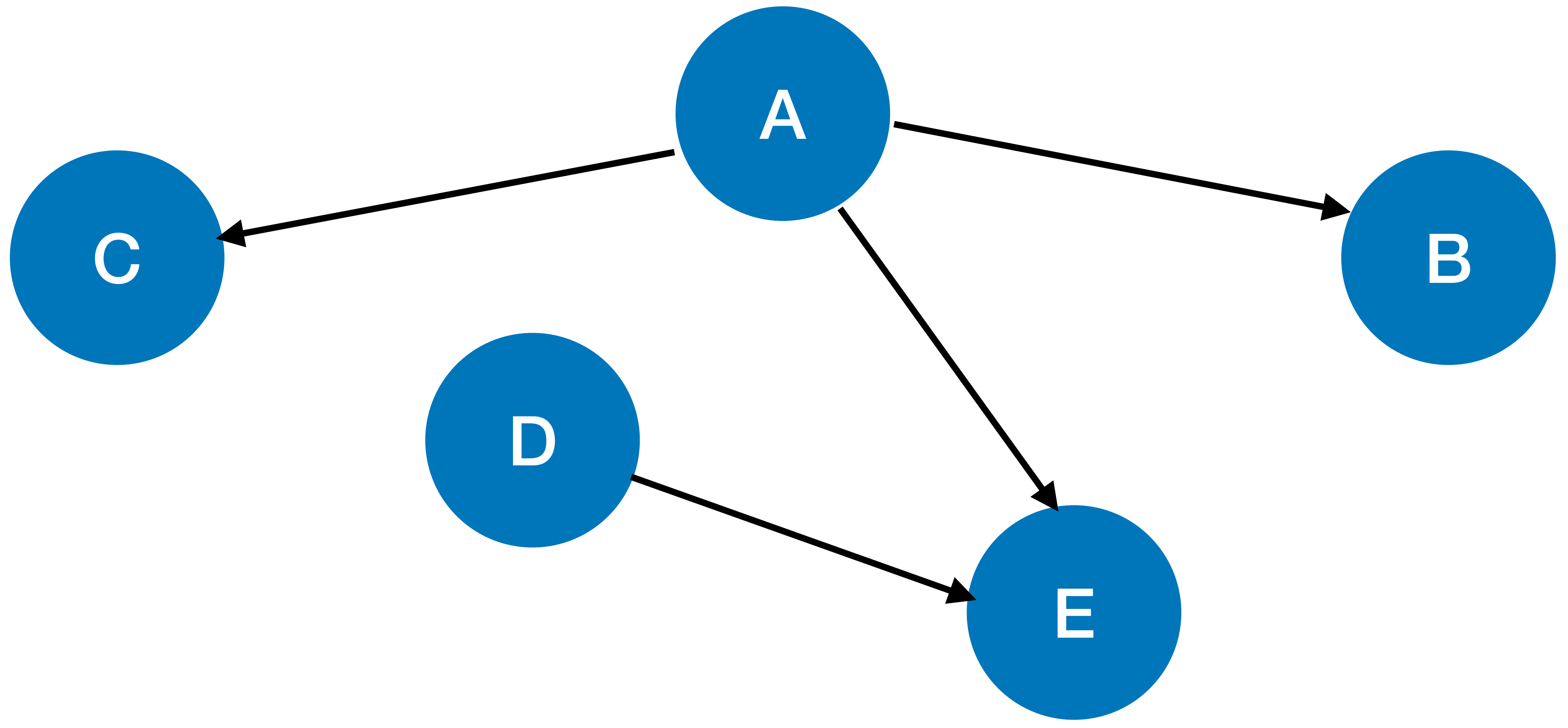

- OBJECTS / DATA POINTS : Nodes, Vertices (V)
- INTERACTIONS / RELATIONS : Links, Edges (E)
- SYSTEM : Network, Graphs (G)
- NODE ATTRIBUTES : Feature vectors (X)

# A QUICK INTRODUCTION TO GRAPHS




- OBJECTS / DATA POINTS : Nodes, Vertices (V)
- INTERACTIONS / RELATIONS : Links, Edges (E)
- SYSTEM : Network, Graphs (G)
- NODE ATTRIBUTES : Feature vectors (X)

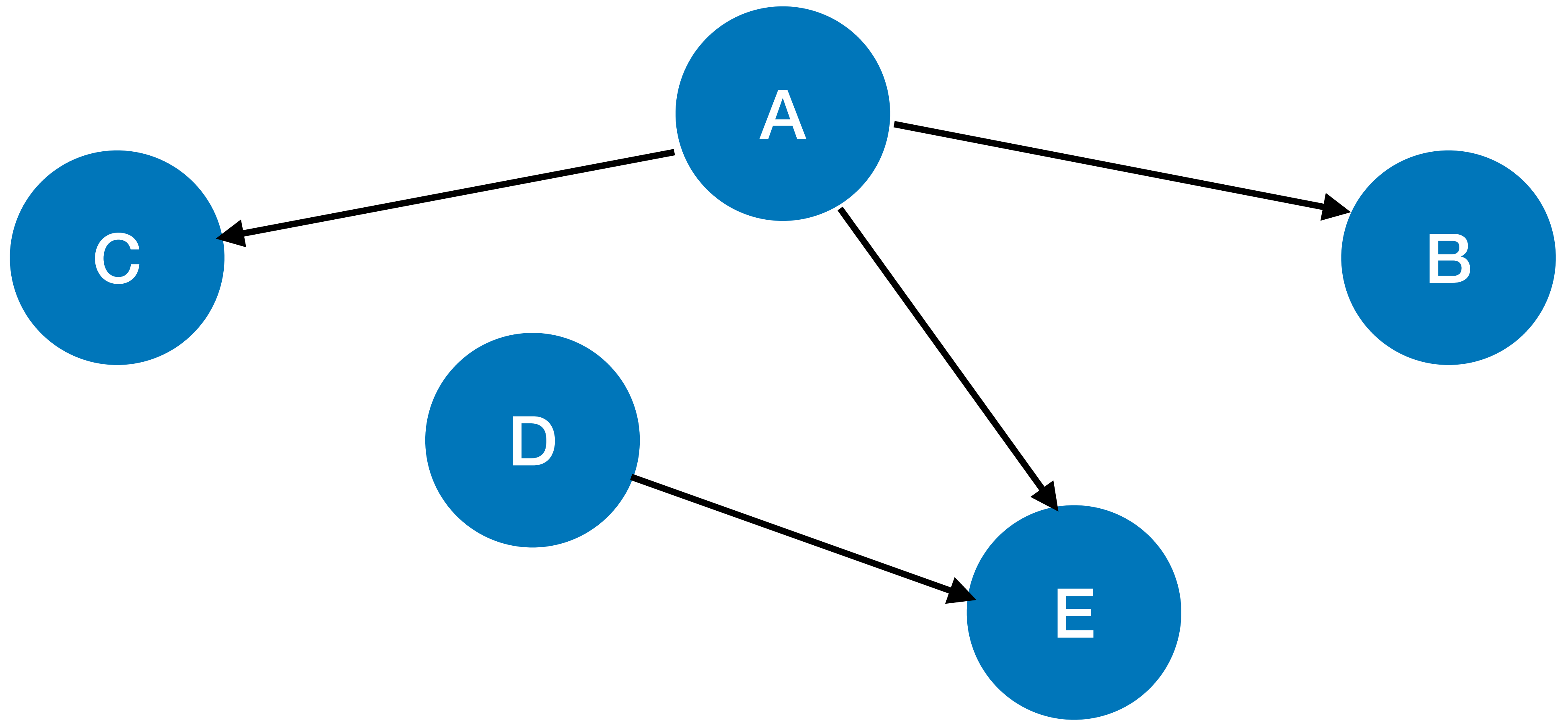
# A QUICK INTRODUCTION TO GRAPHS




$G = (V, E)$

- OBJECTS / DATA POINTS : Nodes, Vertices (V)
- INTERACTIONS / RELATIONS : Links, Edges (E)
- SYSTEM : Network, Graphs (G)
- NODE ATTRIBUTES : Feature vectors (X)

# A QUICK INTRODUCTION TO GRAPHS

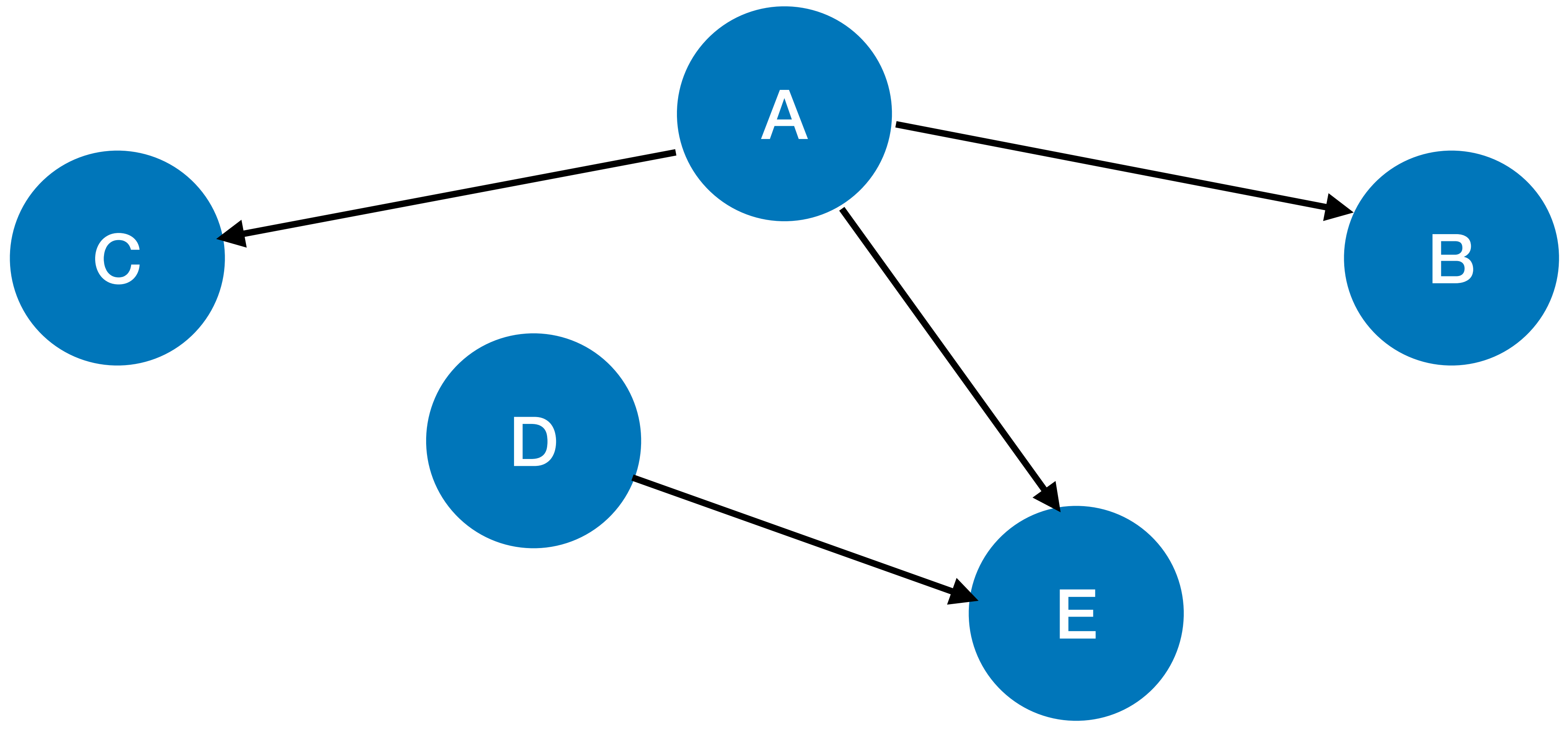



$G = (V, E)$

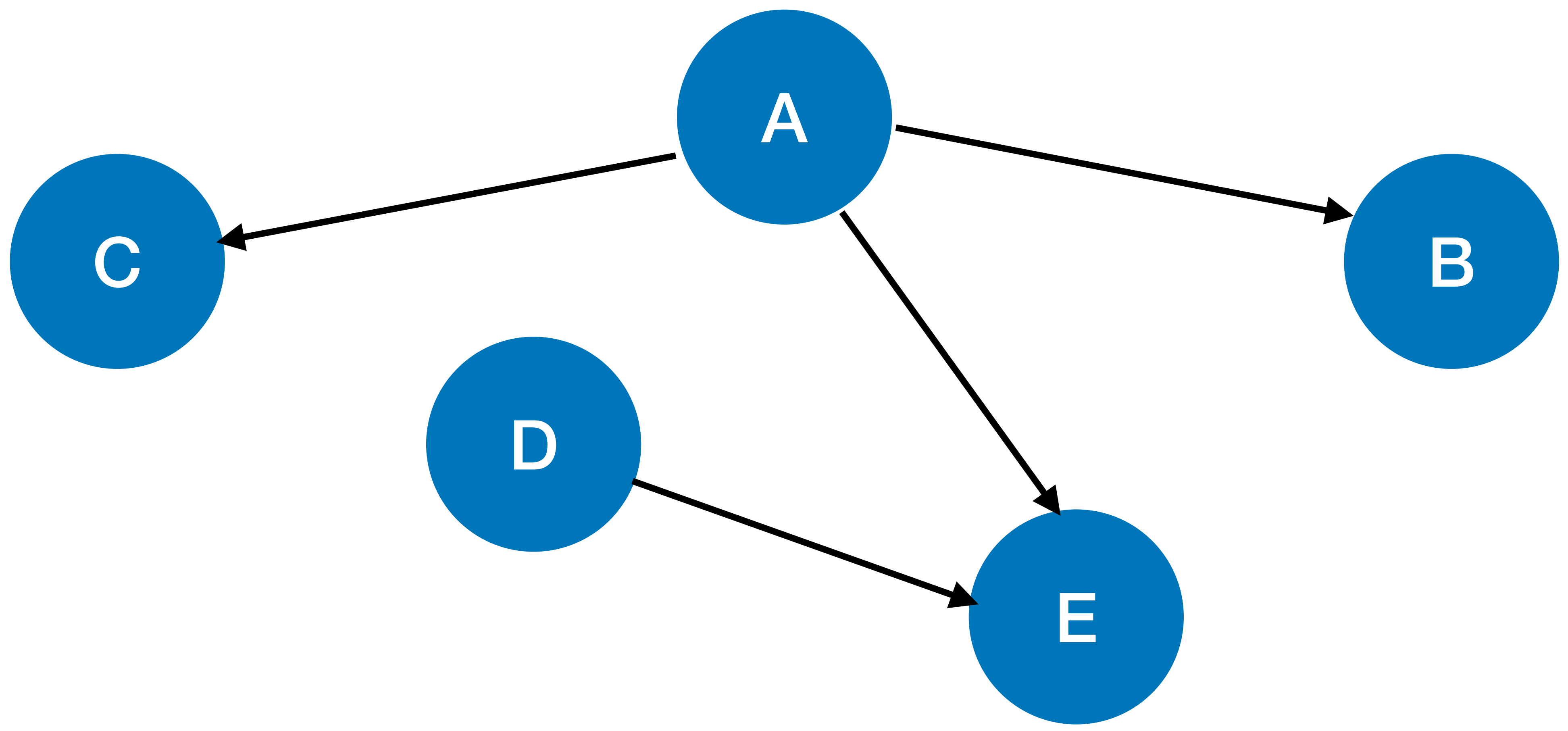
- OBJECTS / DATA POINTS : Nodes, Vertices (V)
- INTERACTIONS / RELATIONS : Links, Edges (E)
- SYSTEM : Network, Graphs (G)
- NODE ATTRIBUTES : Feature vectors (X)

# TASKS ON
# A GRAPH

# TASKS ON A GRAPH

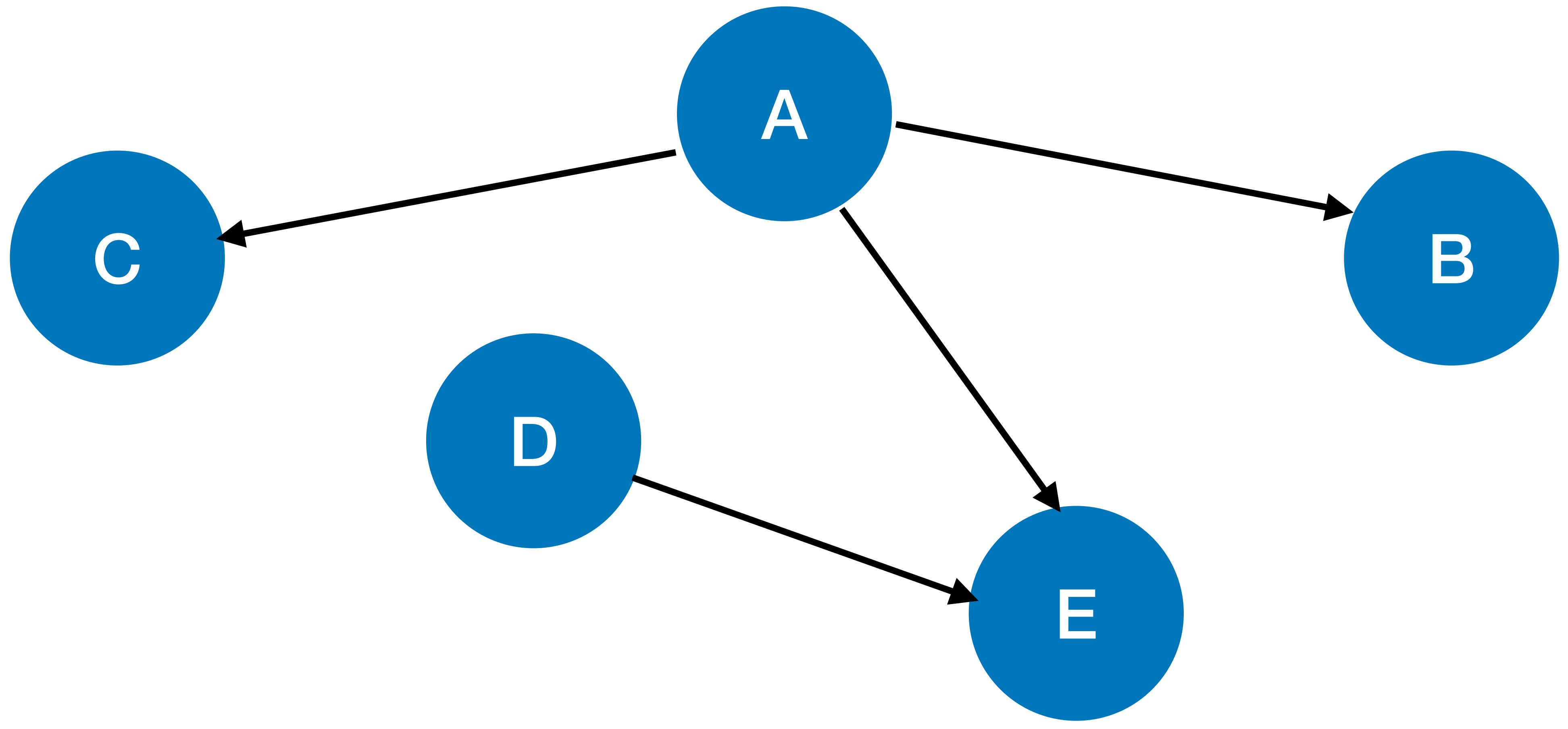


# TASKS ON A GRAPH



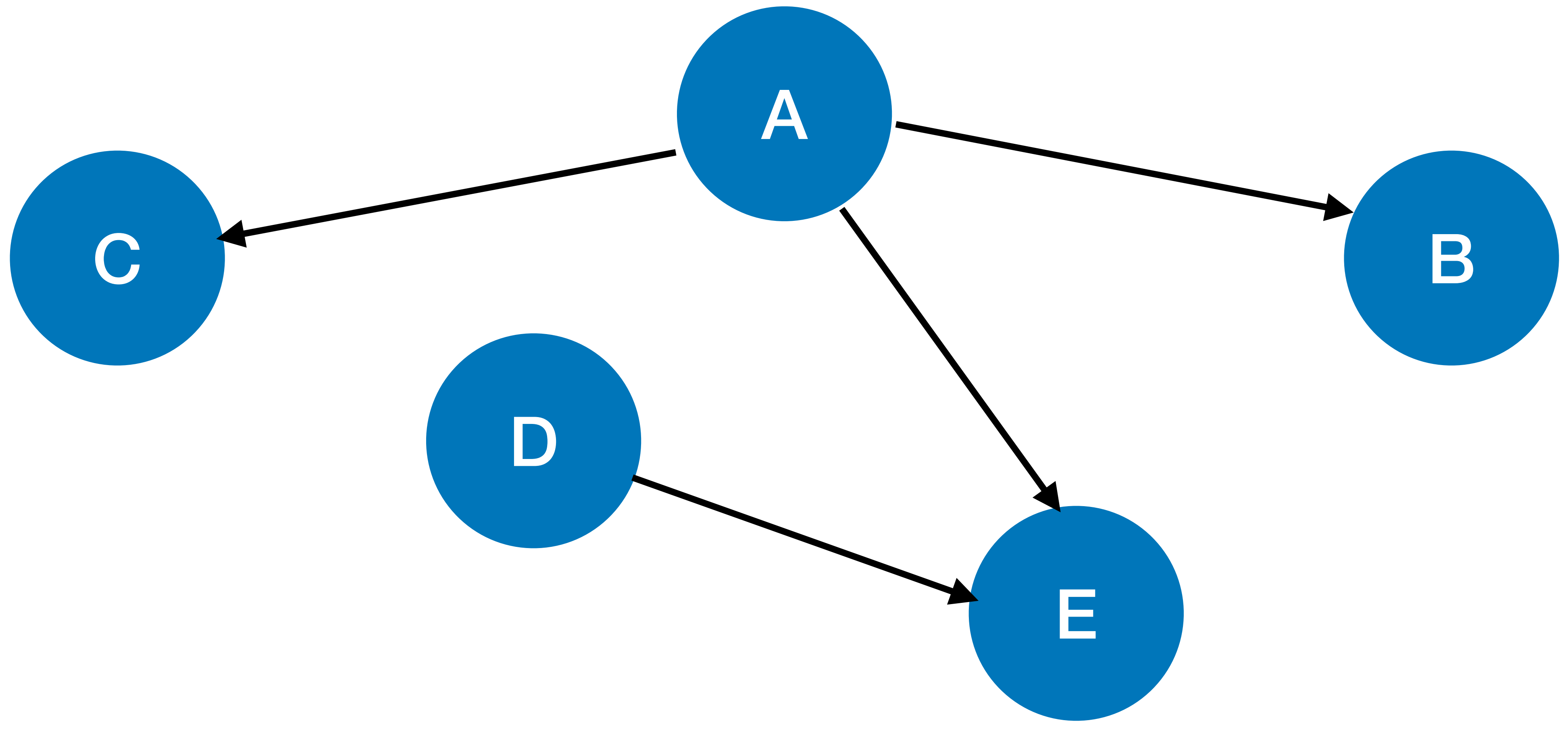

- NODE CLASSIFICATION
- LINK PREDICTION
- GRAPH CLASSIFICATION

# TASKS ON A GRAPH



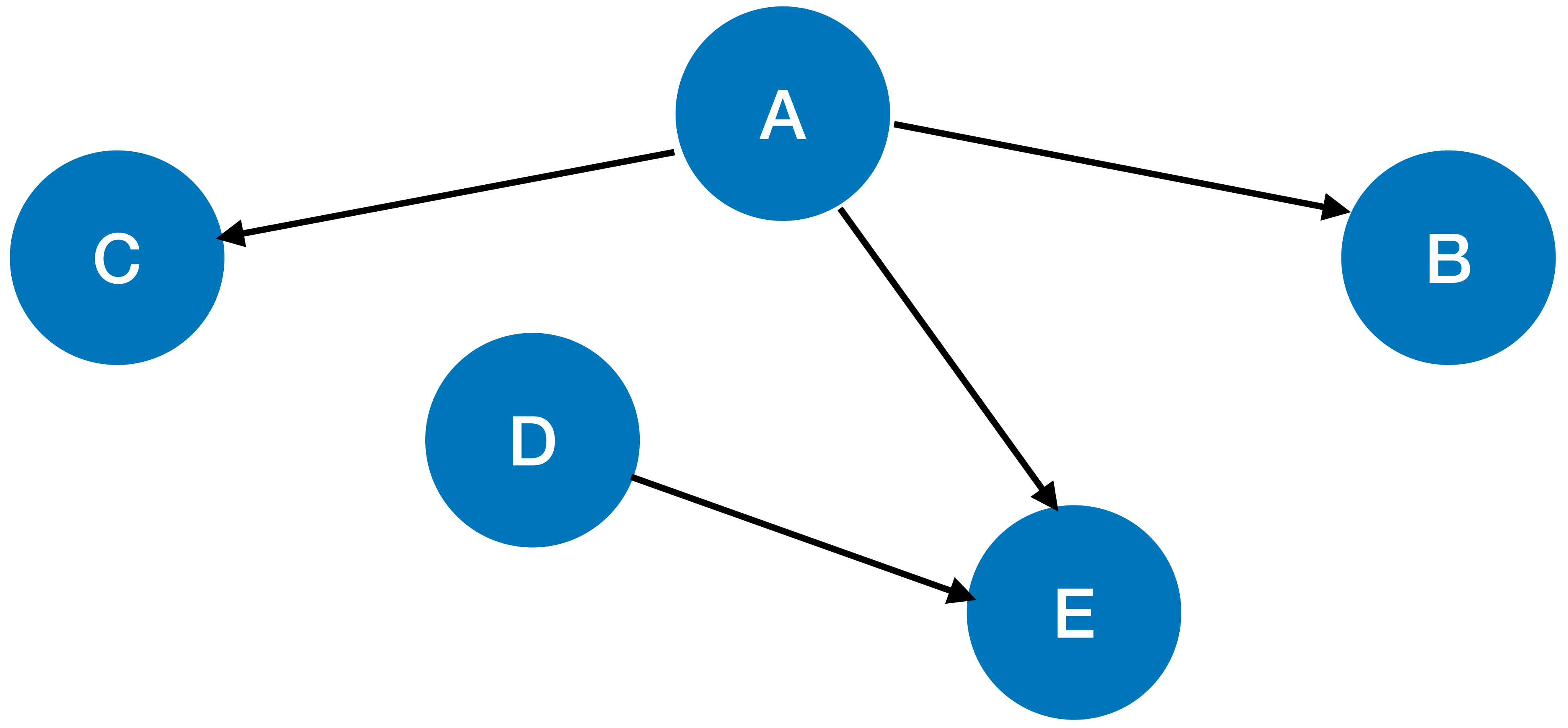

- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION
- GRAPH CLASSIFICATION

# TASKS ON A GRAPH



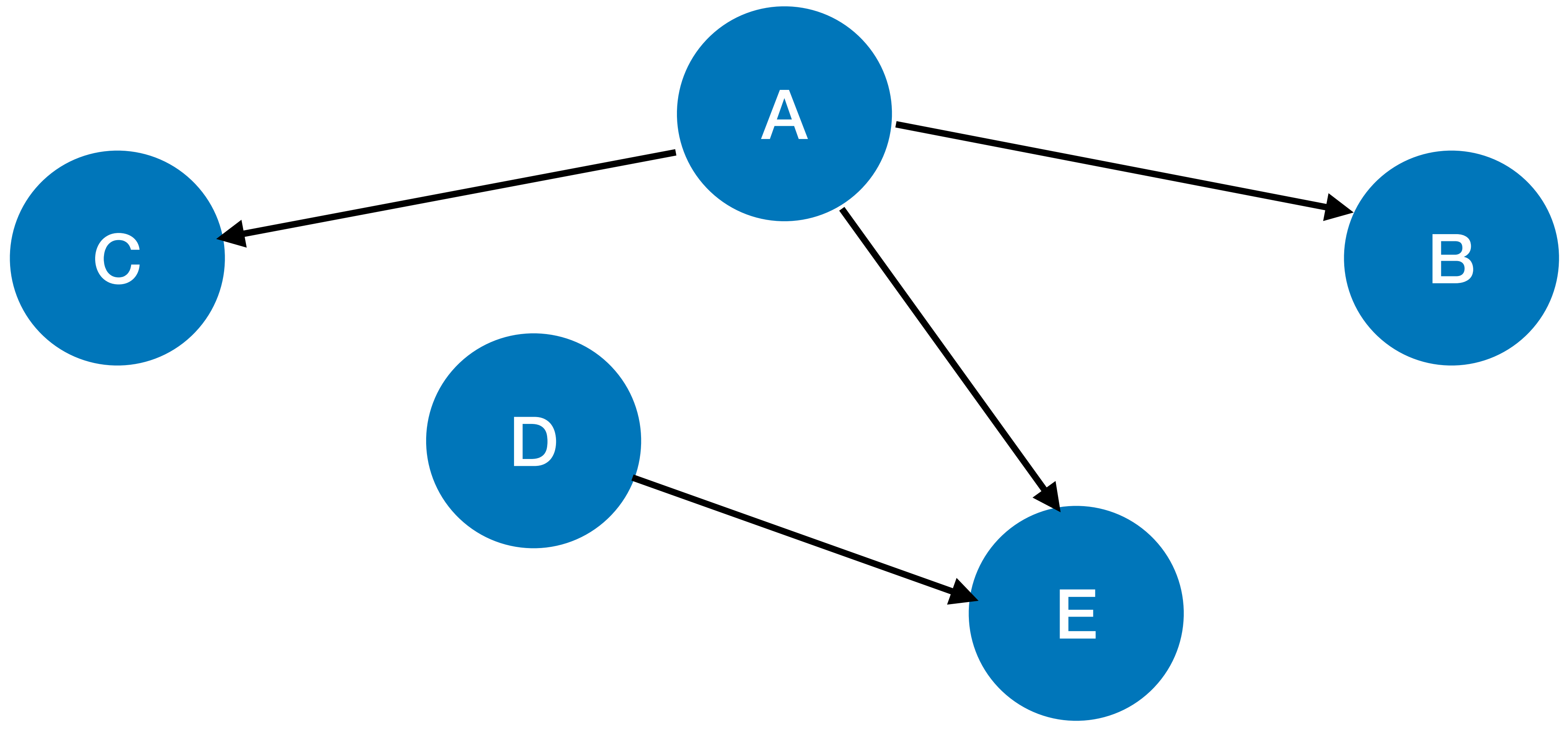

- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION
- GRAPH CLASSIFICATION

# TASKS ON A GRAPH



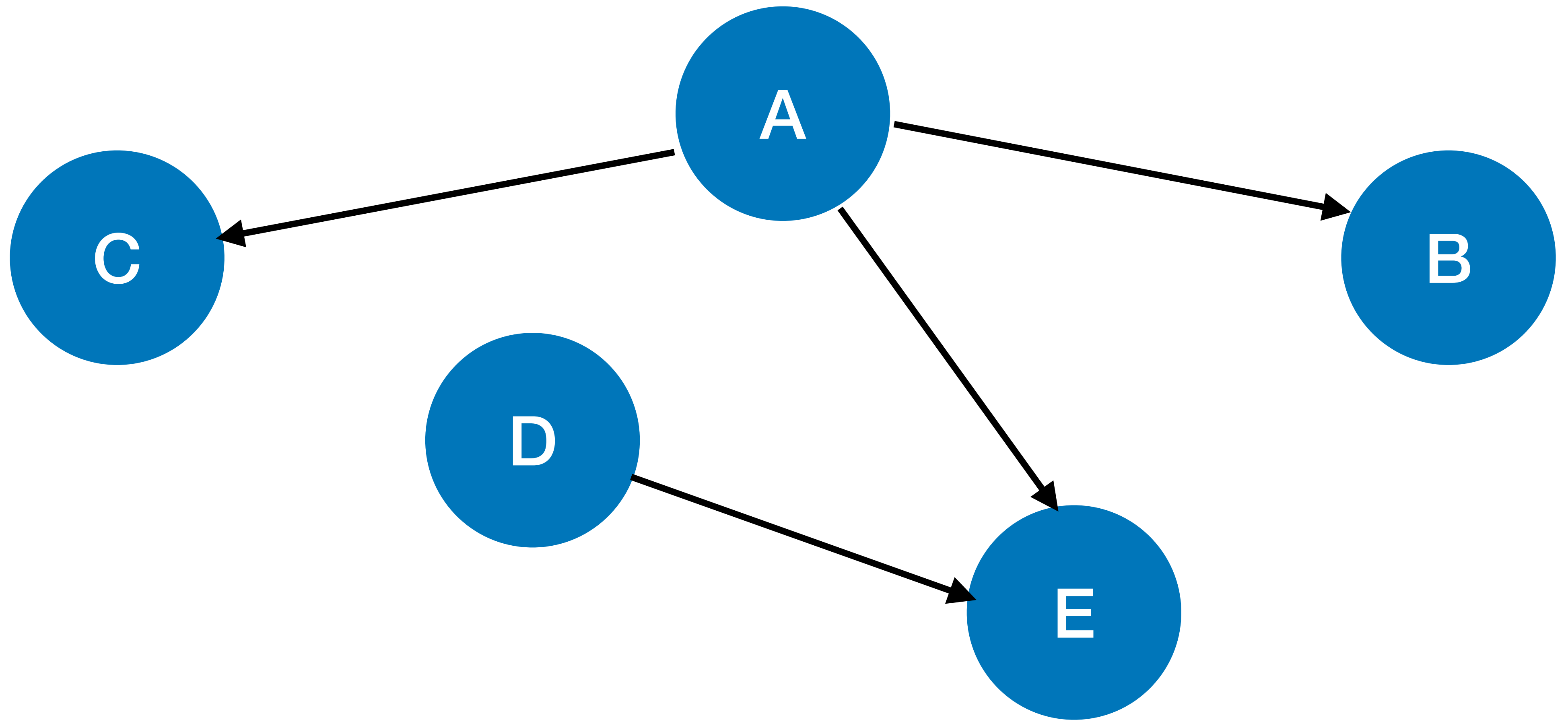

- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION : Recommendation Systems
- GRAPH CLASSIFICATION

# TASKS ON A GRAPH



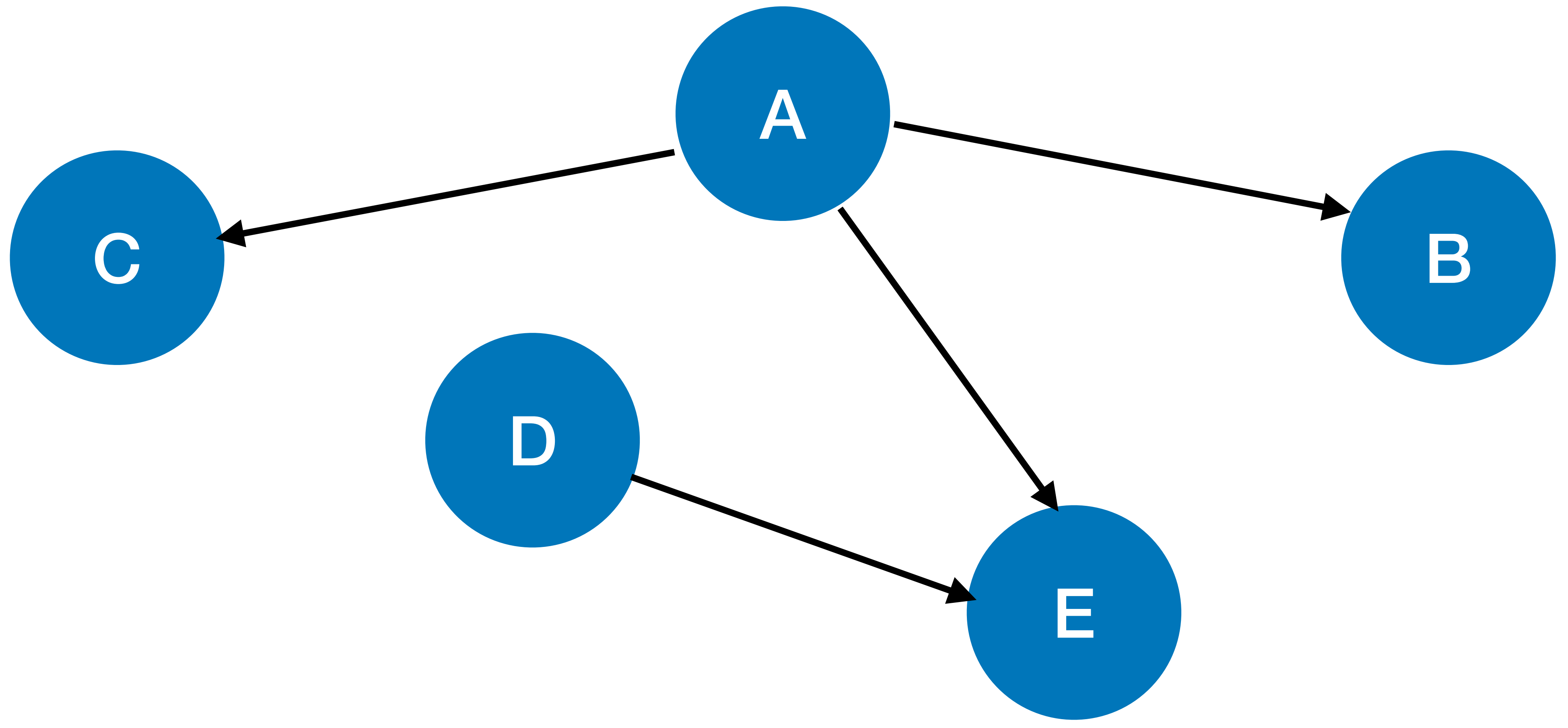

- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION : Recommendation Systems
- GRAPH CLASSIFICATION

# TASKS ON A GRAPH




- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION : Recommendation Systems
- GRAPH CLASSIFICATION : Image Classification

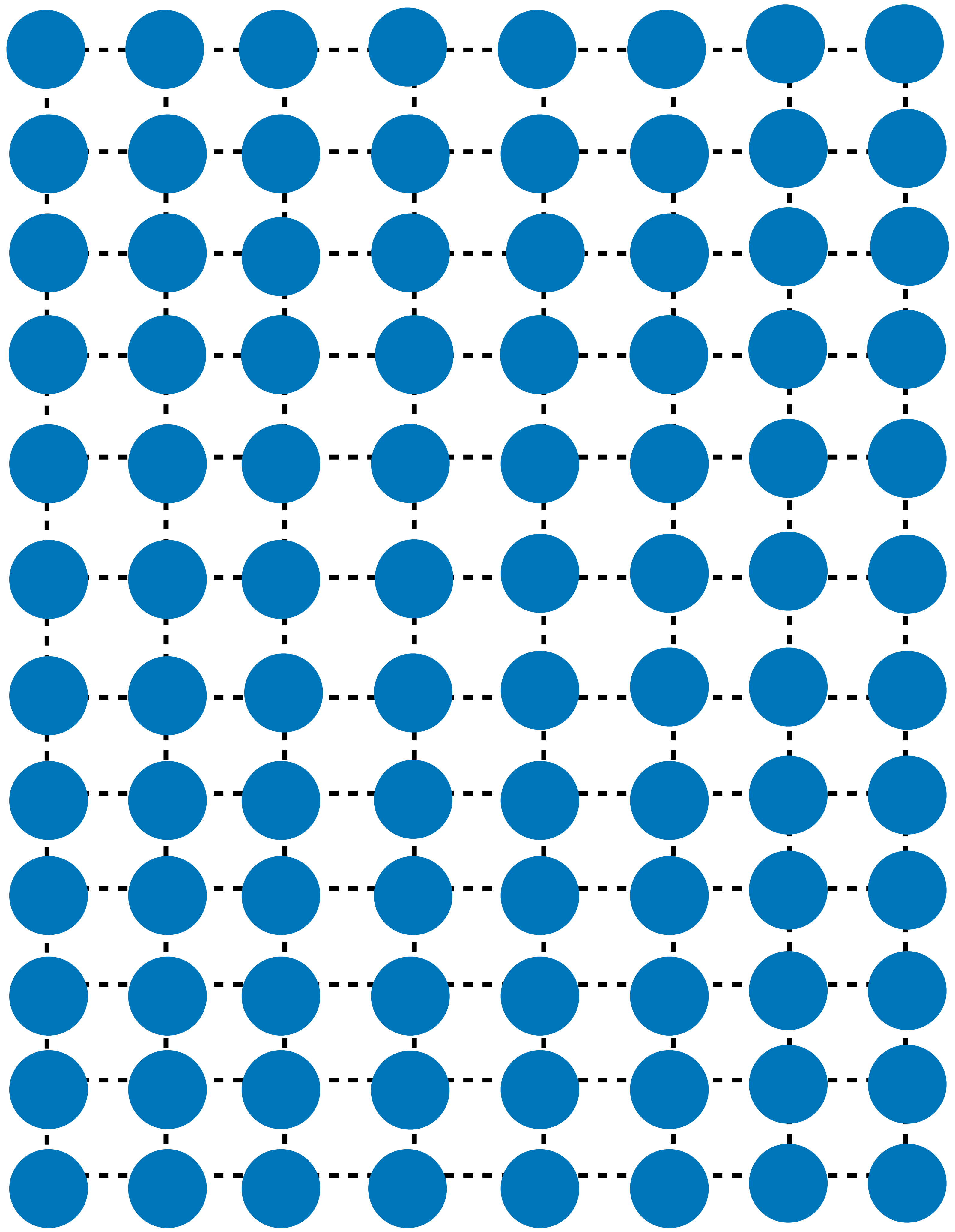
# TASKS ON A GRAPH



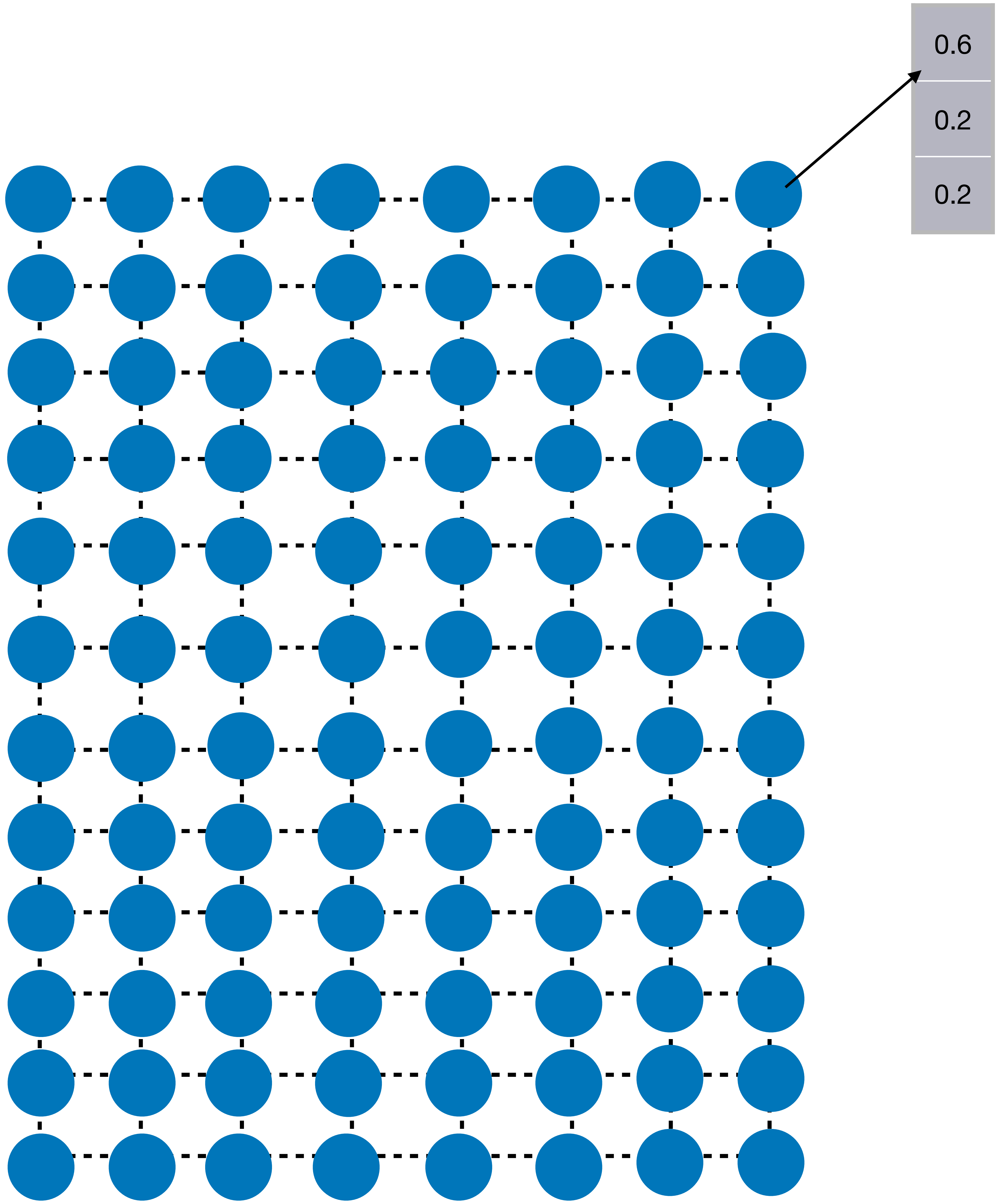

**Question : Can we represent images as graphs?**

- NODE CLASSIFICATION : Topic Classification
- LINK PREDICTION : Recommendation Systems
- GRAPH CLASSIFICATION : Image Classification


Carnegie Mellon
Scarlett
C++
SIGNALS & SYSTEMS



Fixed sized grids!


0.6
0.2
0.2

# SOLVING A PROBLEM USING GNN : Node Classification

# PROBLEM SET UP : Citation Network

Given a citation network, classify a paper topic into either Natural Language Processing (NLP) or Computer Vision (CV) paper.

# PROBLEM SET UP : Citation Network

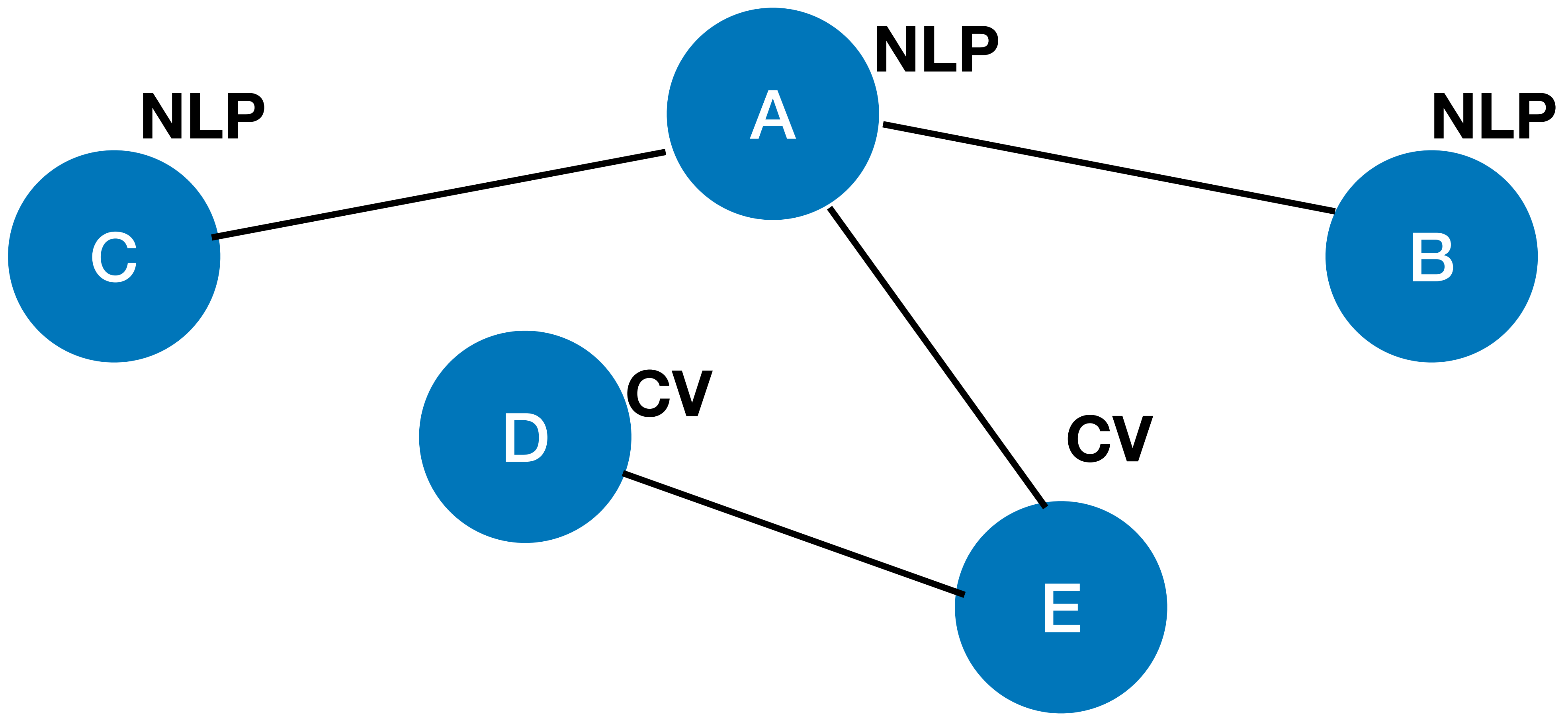
Given a citation network, classify a paper topic into either Natural Language Processing (NLP) or Computer Vision (CV) paper.

- Node Classification
- Binary Classification - NLP, CV

# STEP - 1 : INITIALIZATION

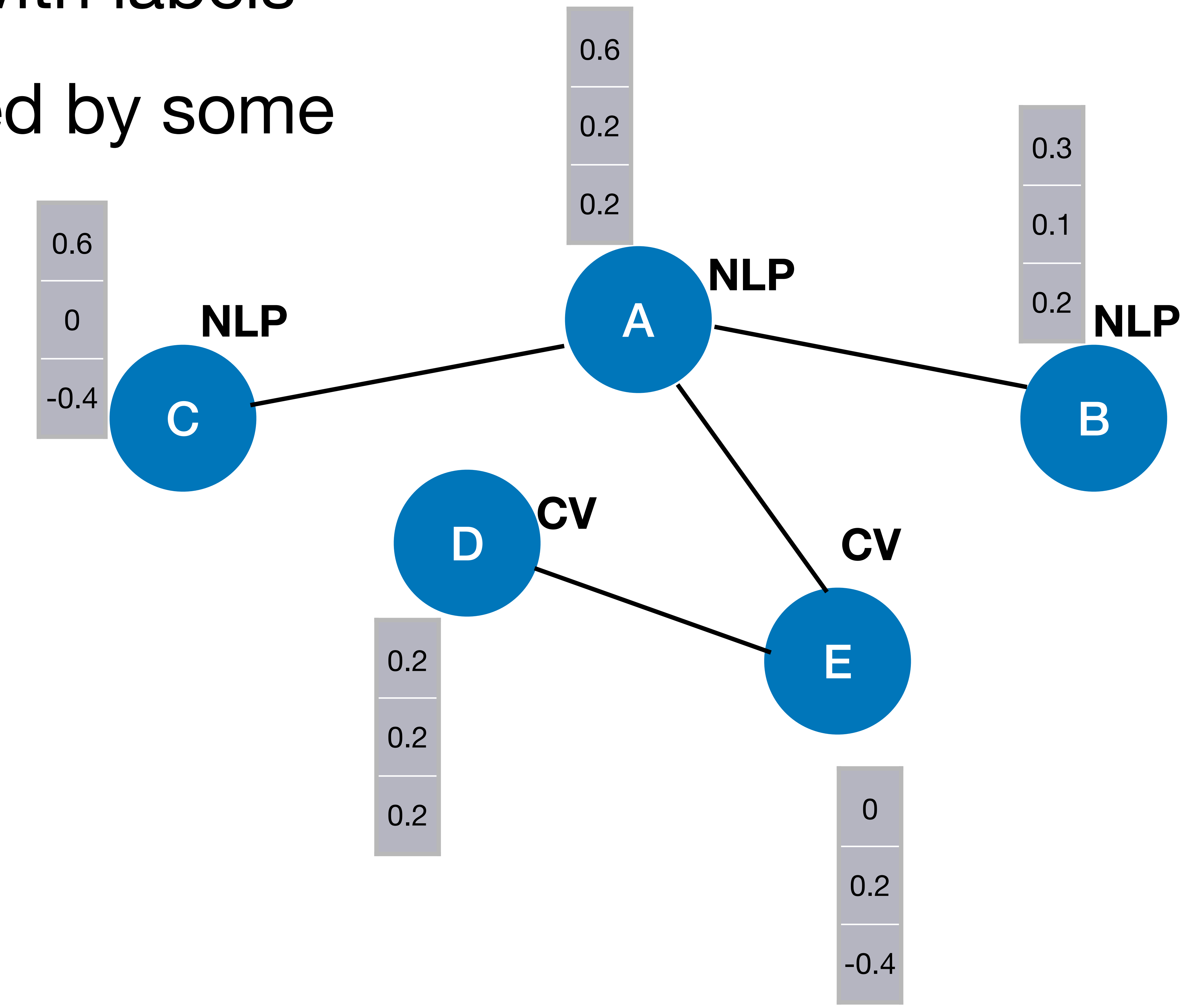
# STEP - 1 : INITIALIZATION

- Graph representing a citation network with labels

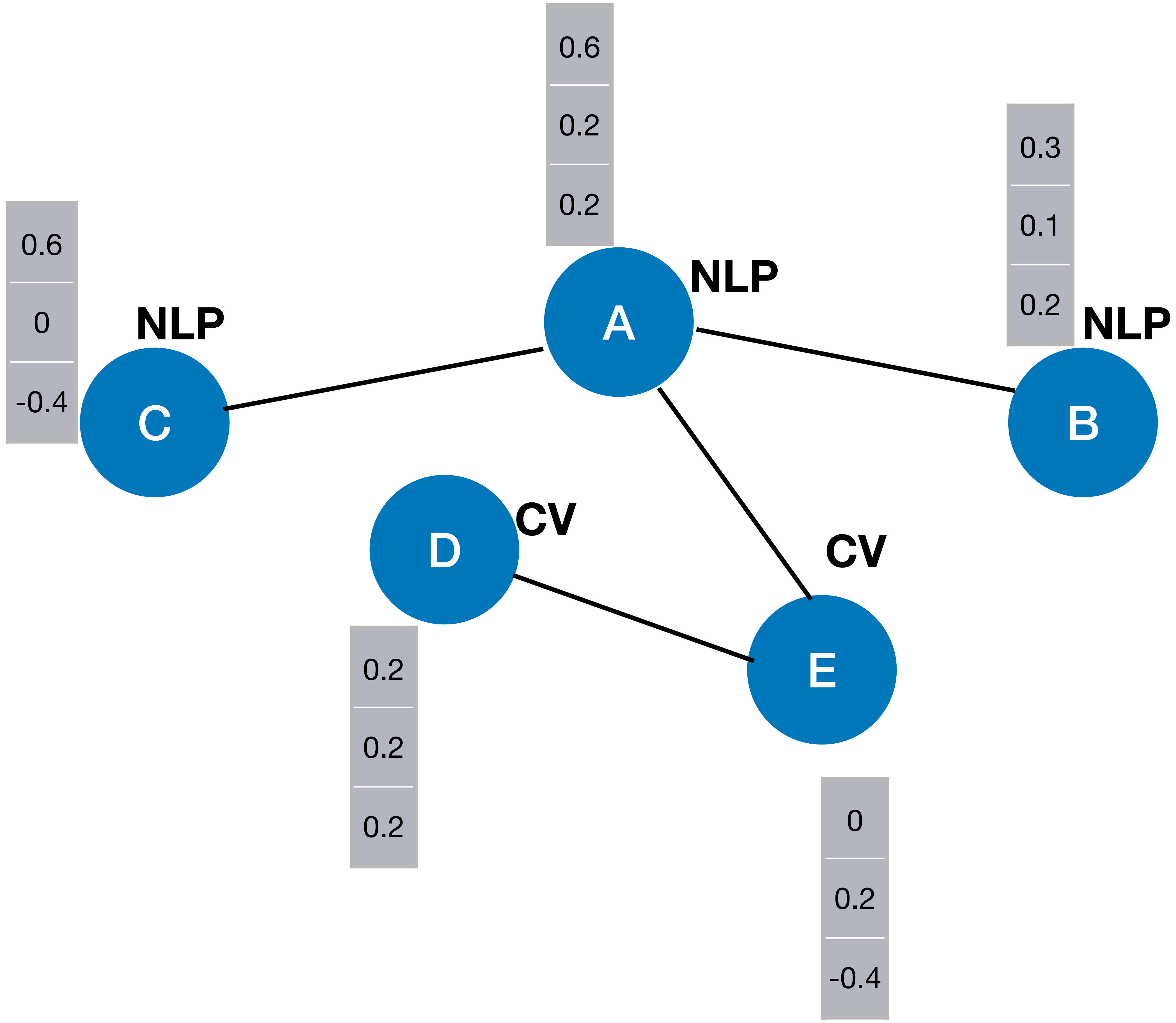


# STEP - 1 : INITIALIZATION

- Graph representing a citation network with labels
- Each node has a feature vector initialized by some heuristic

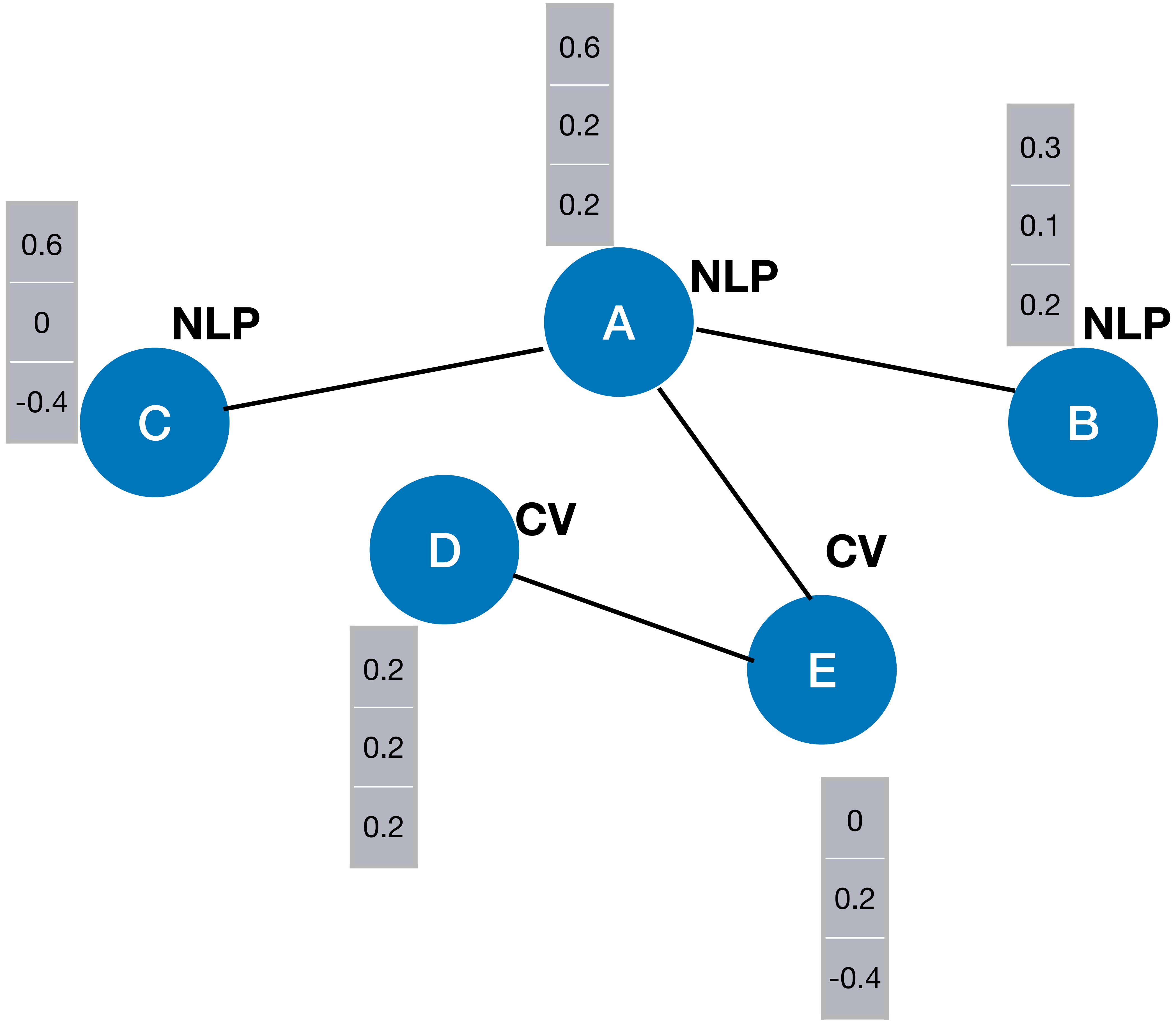


# QUESTION : WHAT DO WE DO NEXT?



QUESTION : WHAT DO WE DO NEXT?

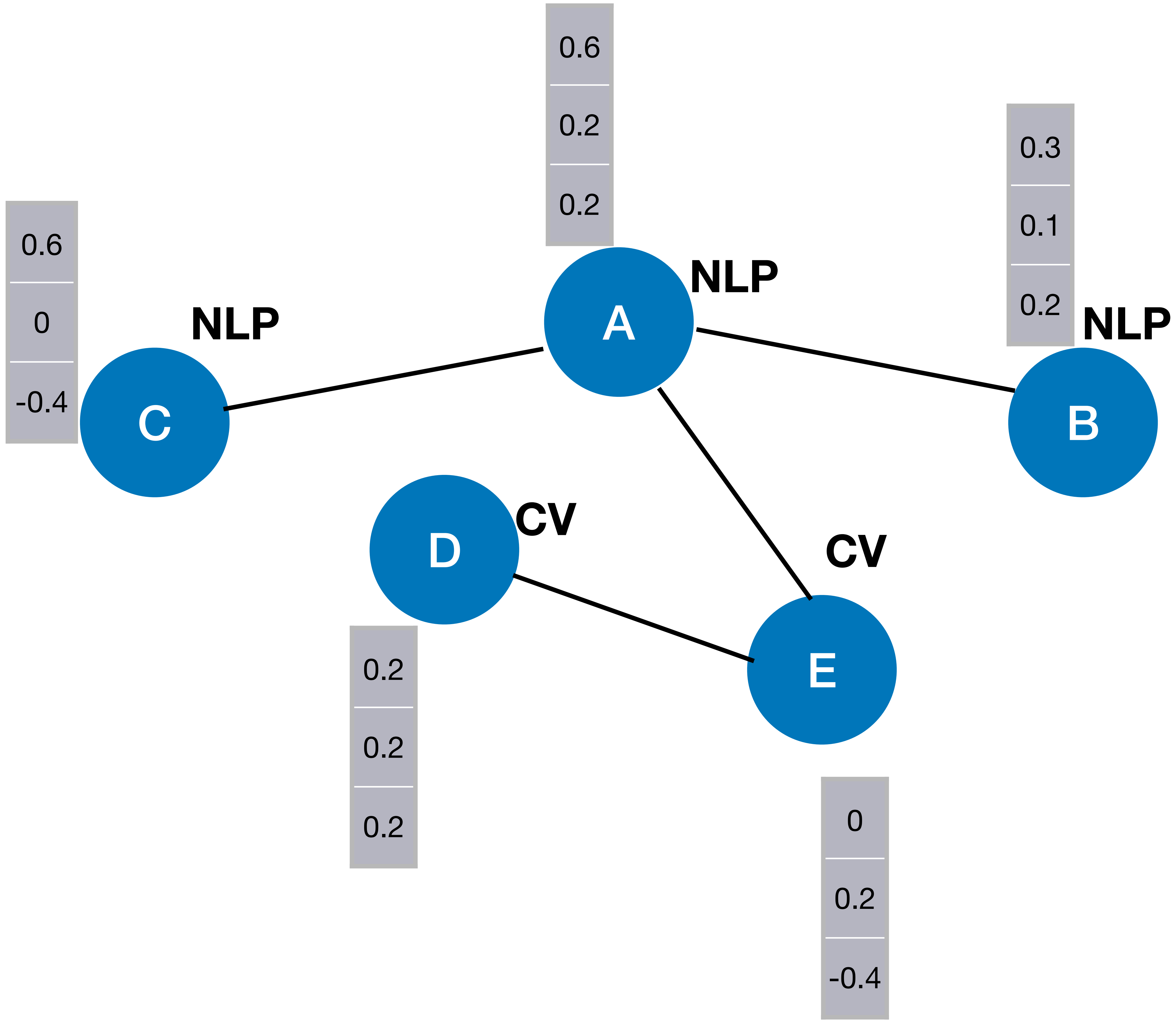
HINT : Take inspiration from traditional CV or NLP models.



# QUESTION : WHAT DO WE DO NEXT?

Currently the node feature vectors describe local information about each paper.

Place each node in context of the rest of the graph.



# STEP - 2.1 : AGGREGATION

# STEP - 2.1 : AGGREGATION

Aggregate neighboring features to place every node in context of the graph.

# STEP - 2.1 : AGGREGATION

Aggregate neighboring features to place every node in context of the graph.

- Model the relations of each node

# STEP - 2.1 : AGGREGATION

Aggregate neighboring features to place every node in context of the graph.

- Model the relations of each node
- Create node embeddings incorporating the context of the graph

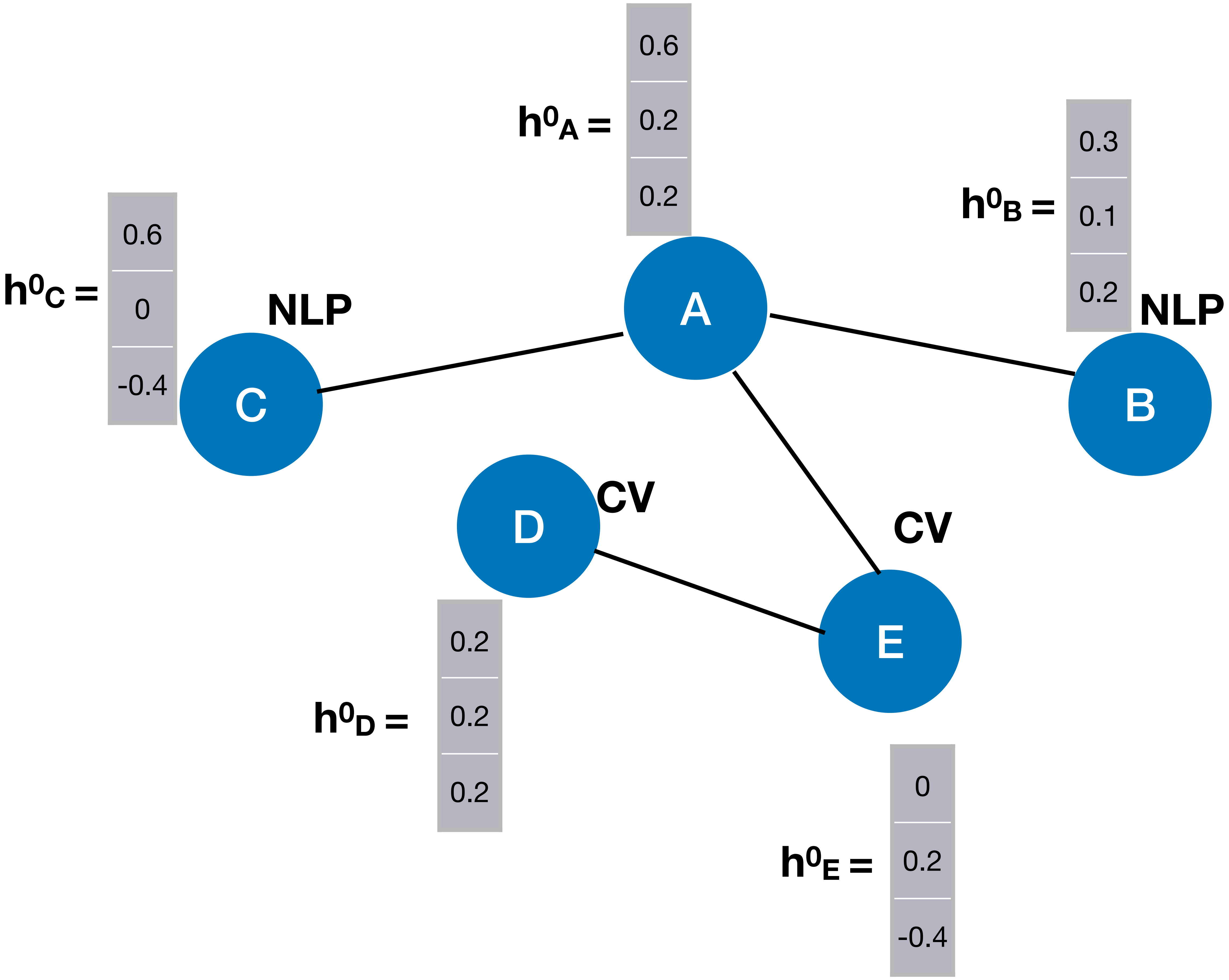
# STEP - 2.1 : AGGREGATION

Aggregate neighboring features to place every node in context of the graph.

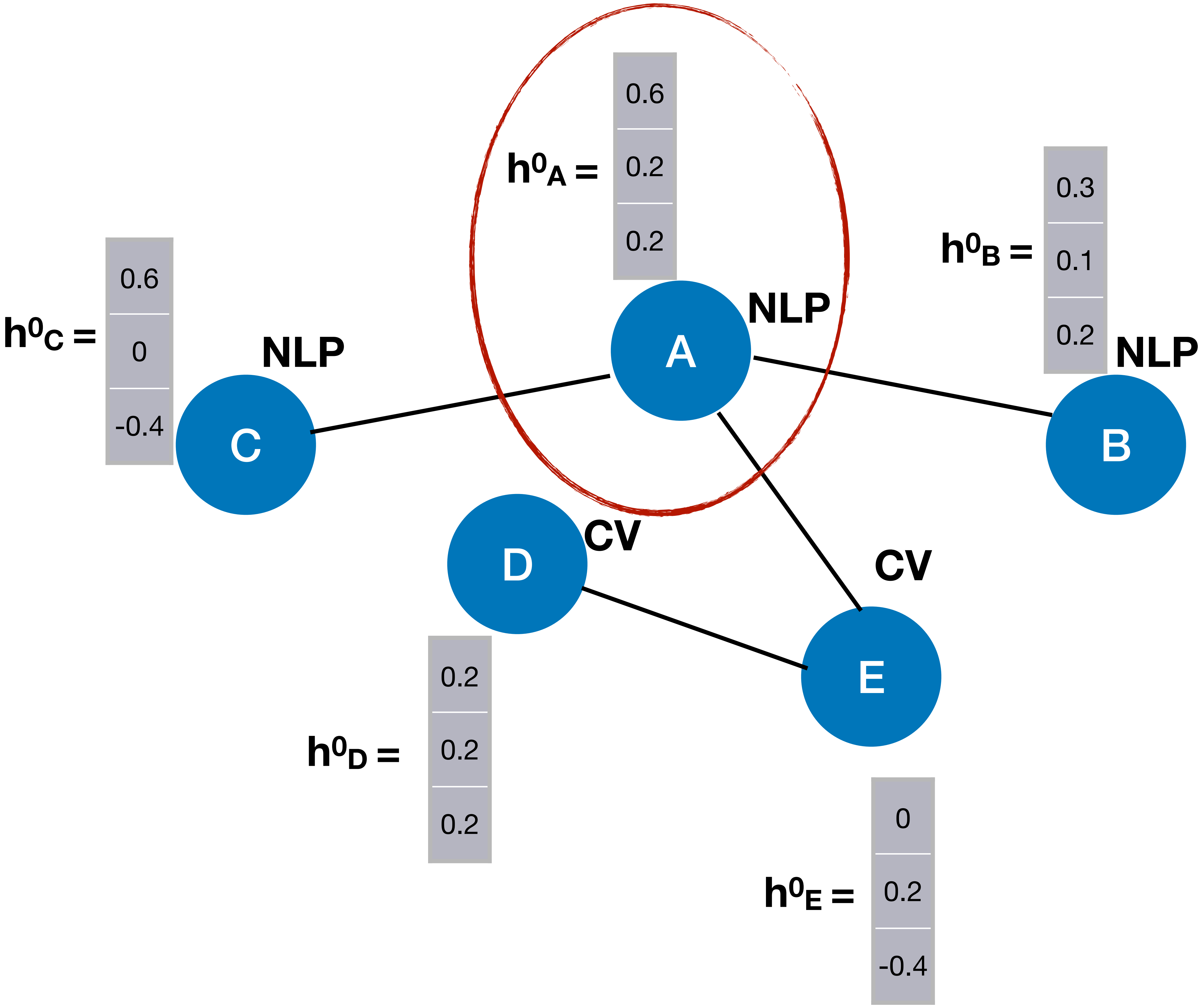
- Model the relations of each node
- Create node embeddings incorporating the context of the graph

QUESTION : HOW DO WE DO AGGREGATION?

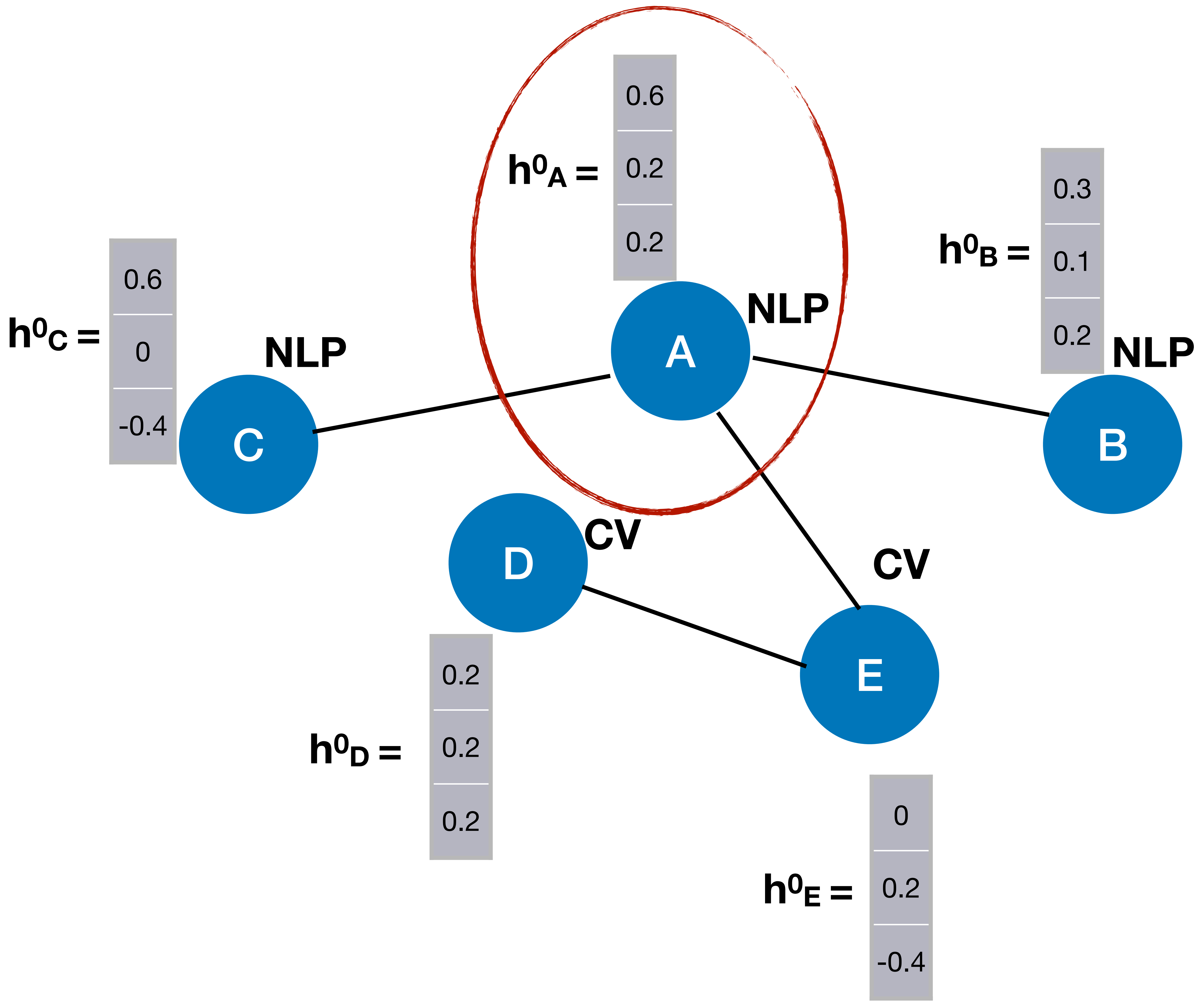
# STEP - 2.1 : AGGREGATION



# STEP - 2.1 : AGGREGATION

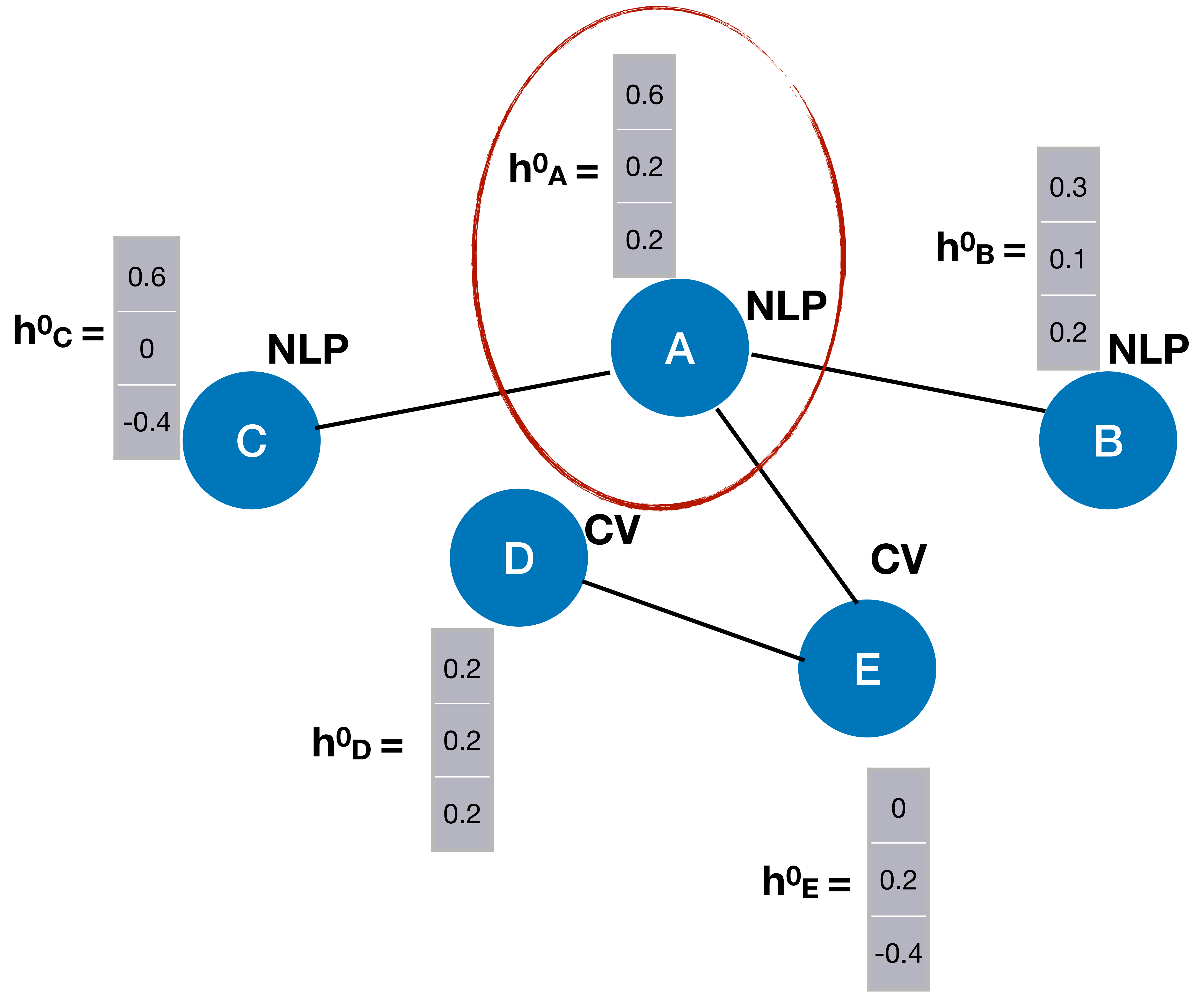


# STEP - 2.1 : AGGREGATION




$$t^1_A = f\ (h^0_E, h^0_B, h^0_C\ )$$

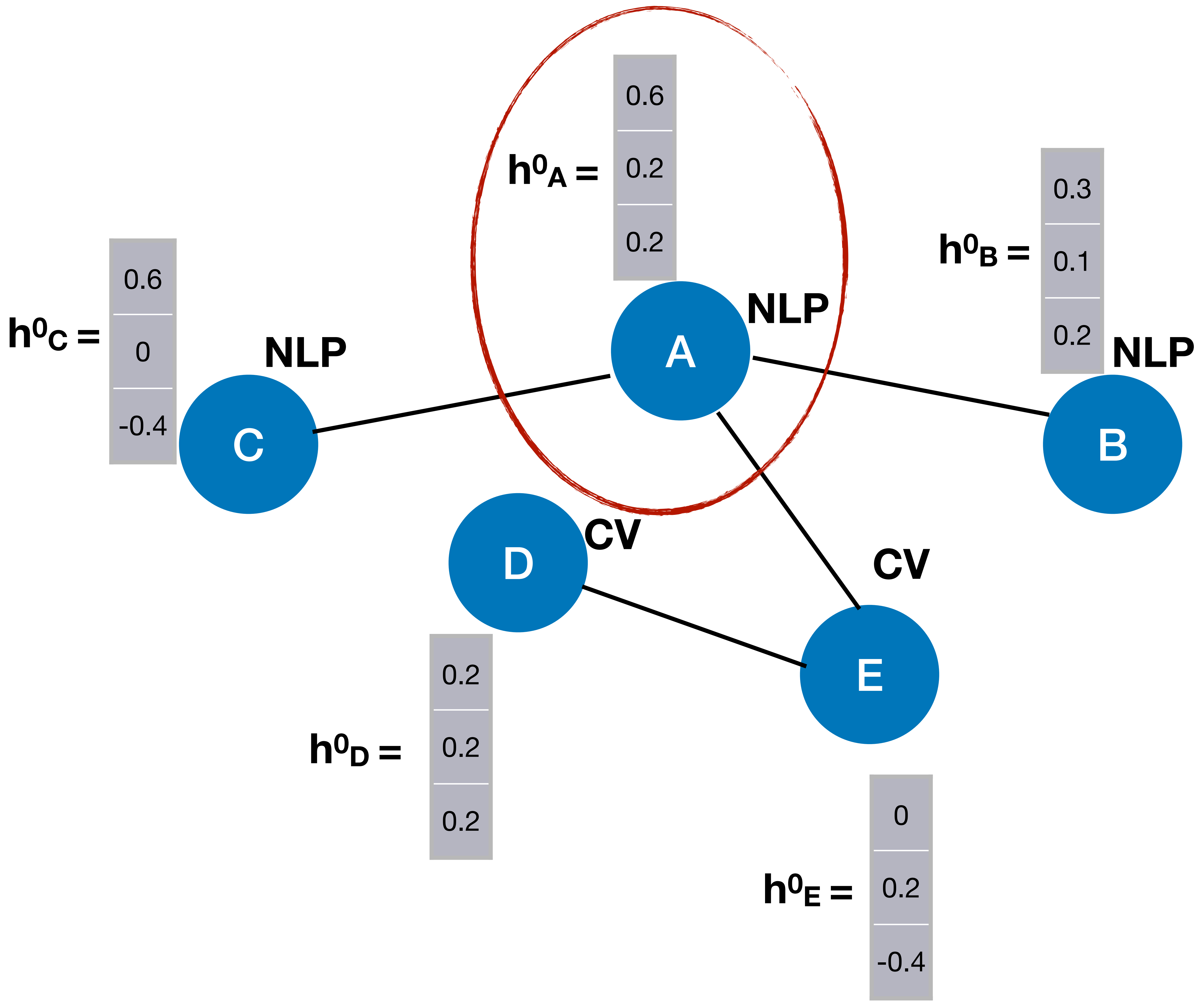
# STEP - 2.1 : AGGREGATION




$$t^1_A = f\,(h^0_E, h^0_B, h^0_C\,)$$

Aggregation Function : Order Invariant!

# STEP - 2.1 : AGGREGATION



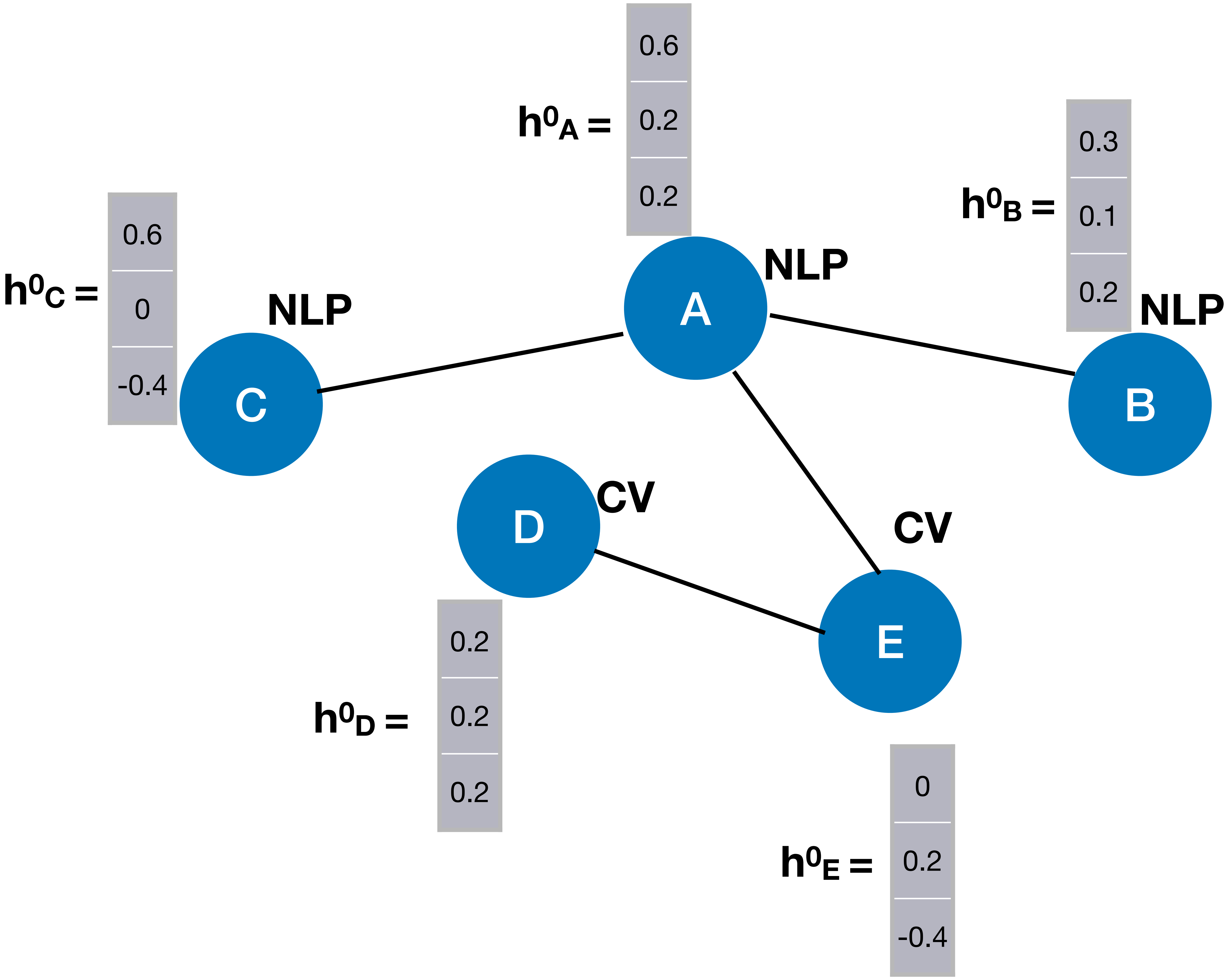

$$t^1_A = f\,(h^0_E, h^0_B, h^0_C\,) + g(h^0_A)$$

Aggregation Function : Order Invariant!

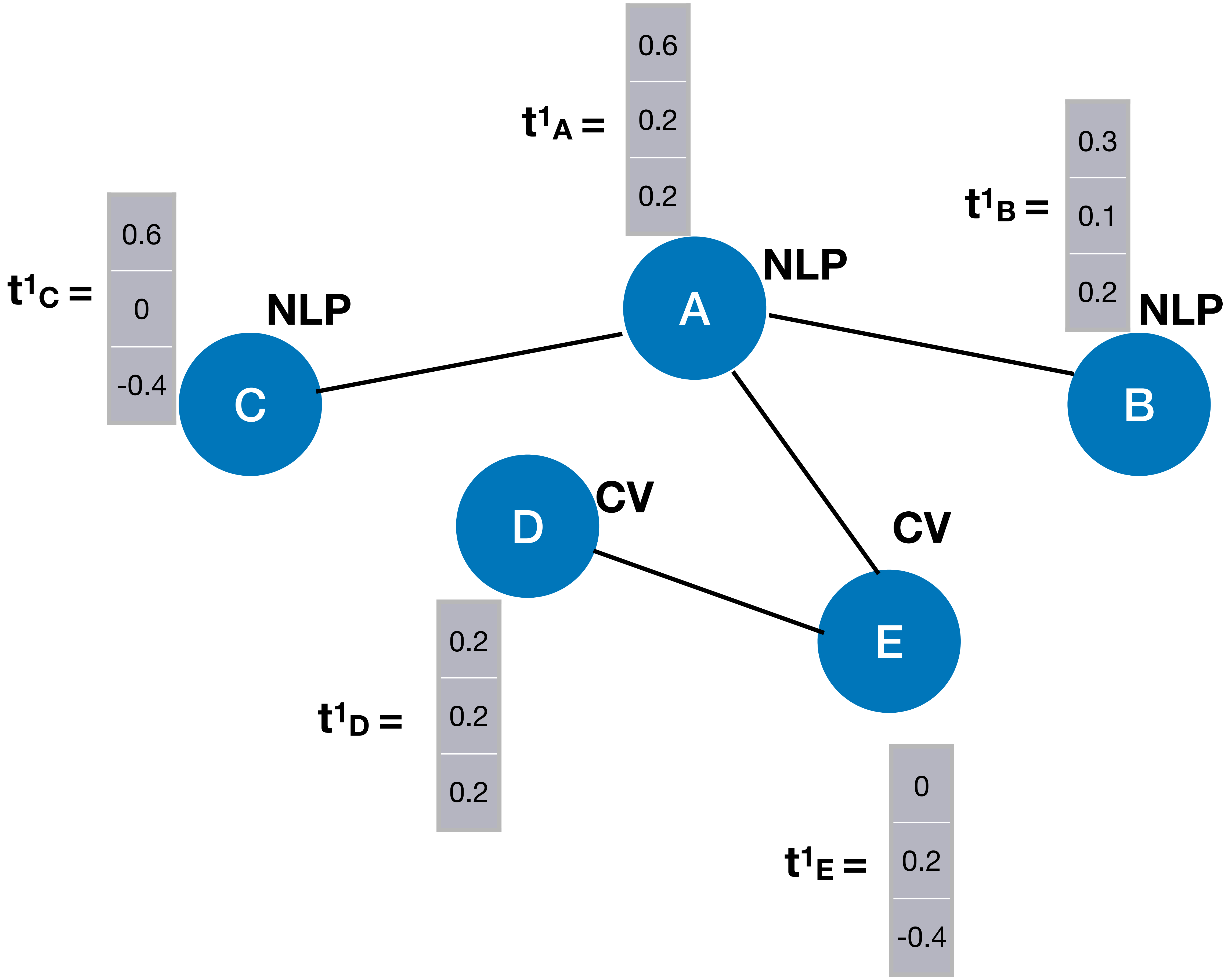
Aggregation Function : Summation, Average, Max

# STEP - 2.2 : LINEAR LAYER

# STEP - 2.2 : LINEAR LAYER



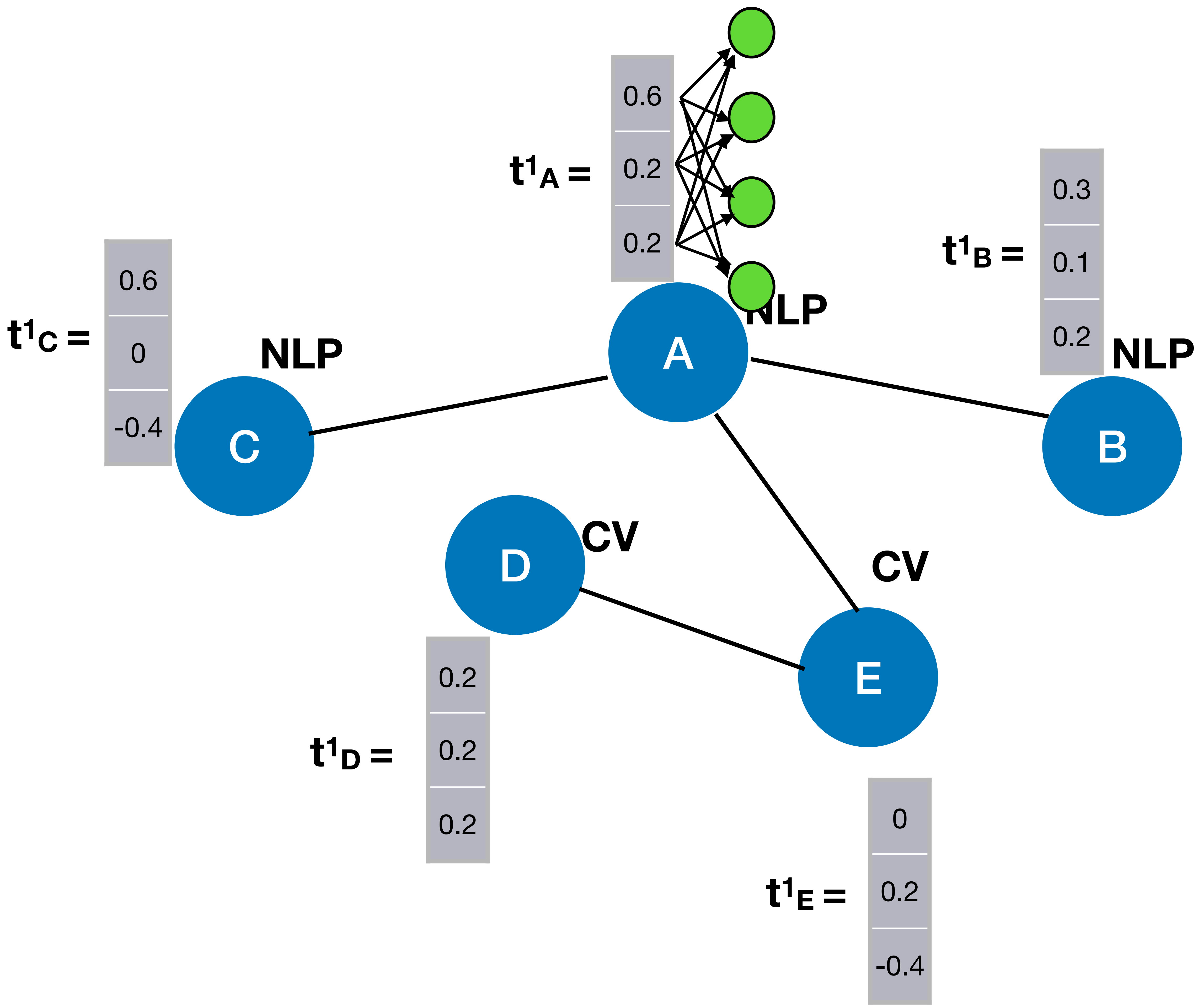
# STEP - 2.2 : LINEAR LAYER




Aggregation:

$$t^1_v = f\,(h^0_{u_1}, h^0_{u_2}, h^0_{u_3}, \dots) + g\,(h^0_v)\ ,\ \forall\ u_i\ \in N(v)$$

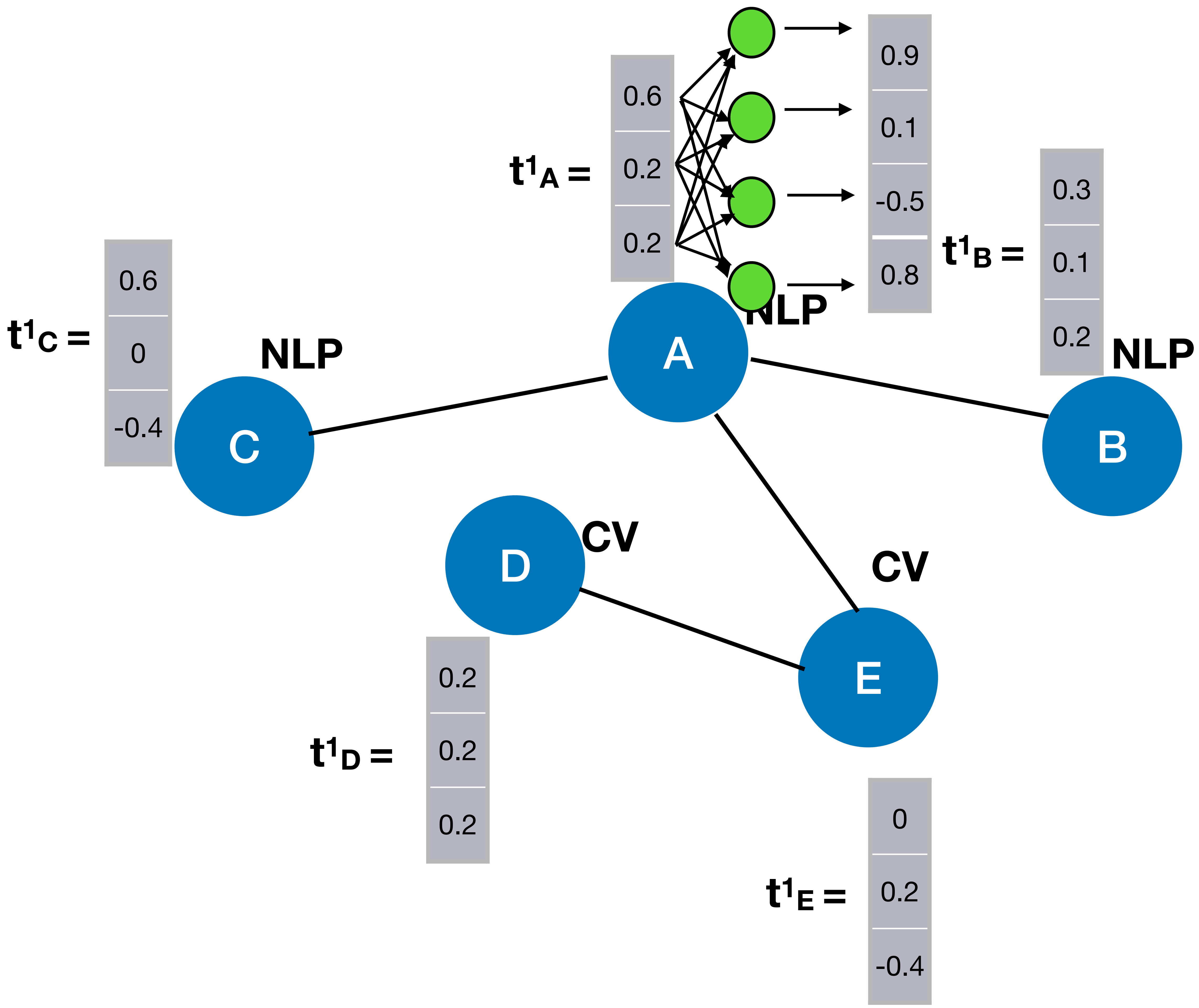
# STEP - 2.2 : LINEAR LAYER




Aggregation:

$$t^1_v = f\,(h^0_{u_1}, h^0_{u_2}, h^0_{u_3}, \ldots)\ ,\ \forall\ u_i\ \in N(v)$$

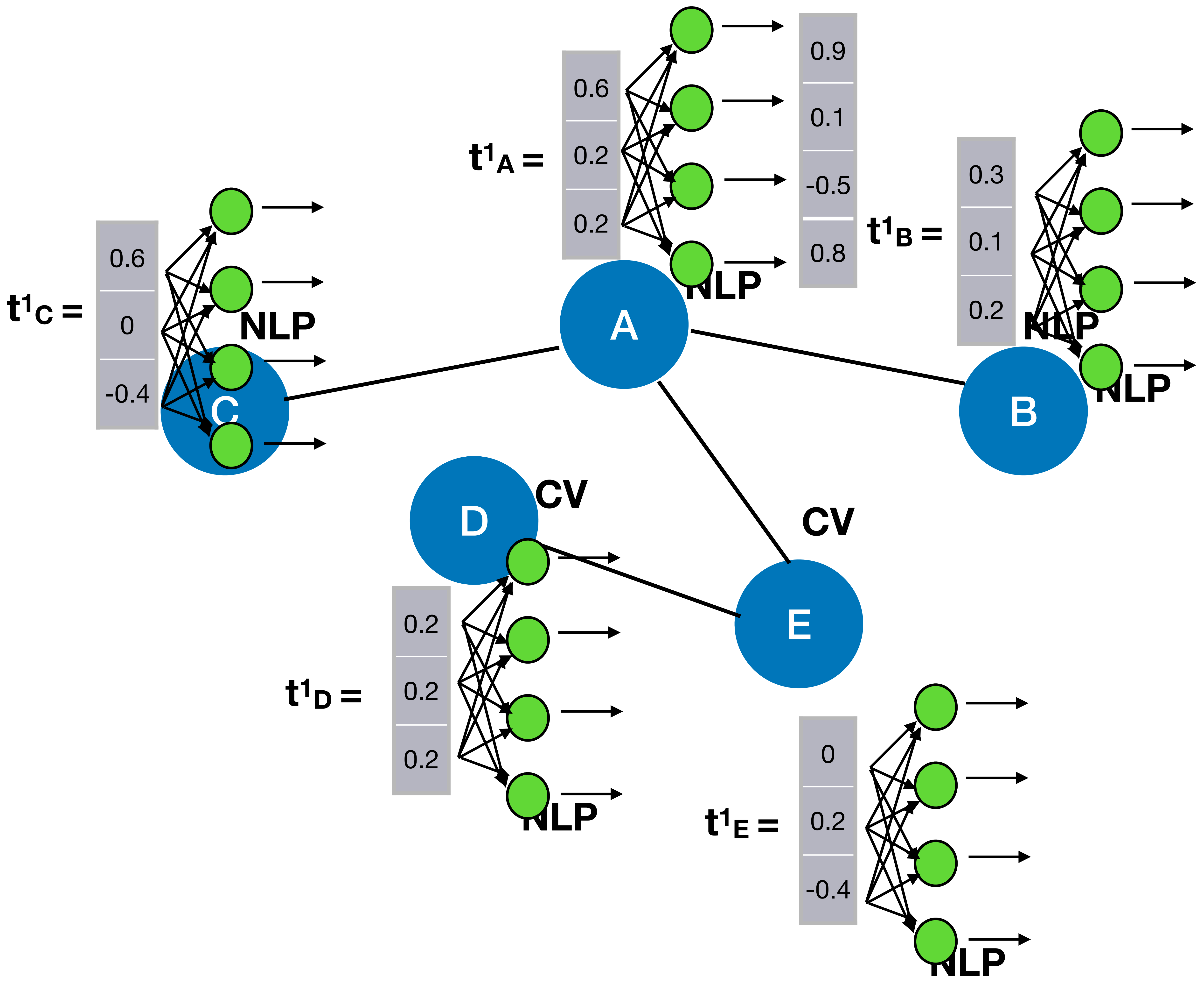
# STEP - 2.2 : LINEAR LAYER




Aggregation:

$$t^1_v = f\,(h^0_{u_1}, h^0_{u_2}, h^0_{u_3}, \ldots)\ ,\ \forall\ u_i\ \in N(v)$$

# STEP - 2.2 : LINEAR LAYER



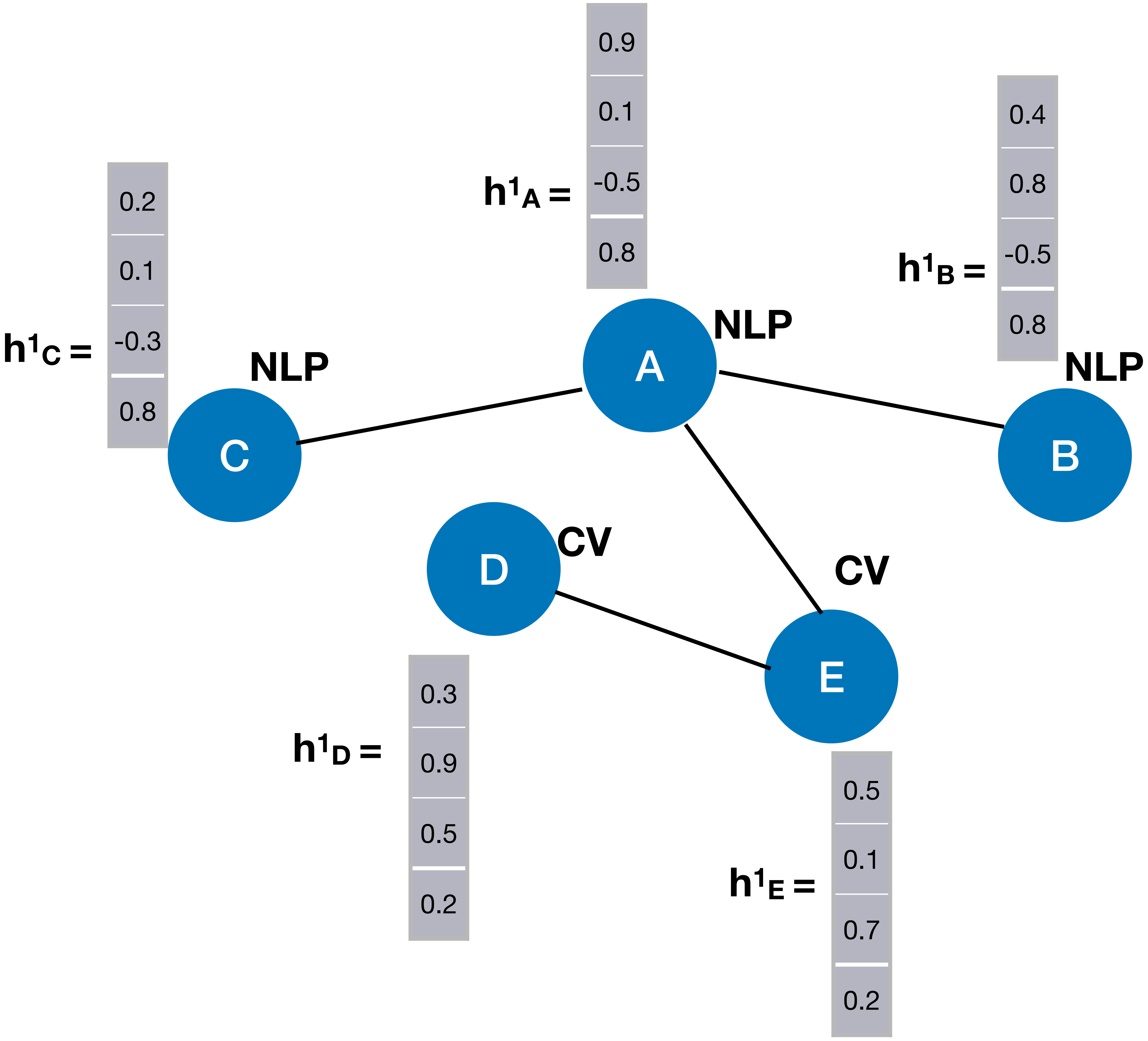

Aggregation:

$$t^1{}_v = f\,(h^0{}_{u_1}, h^0{}_{u_2}, h^0{}_{u_3}, \ldots)\;,\; \forall\, u_i \in \boldsymbol{N(v)}$$

FC Layer and Activation:

$$h^1{}_v = \sigma\,(\,g\,(t^1{}_{u_1}, t^1{}_{u_2}, t^1{}_{u_3}, \ldots)\,),\; \forall\, u_i \in \boldsymbol{N(v)}$$

# STEP - 2.2 : LINEAR LAYER



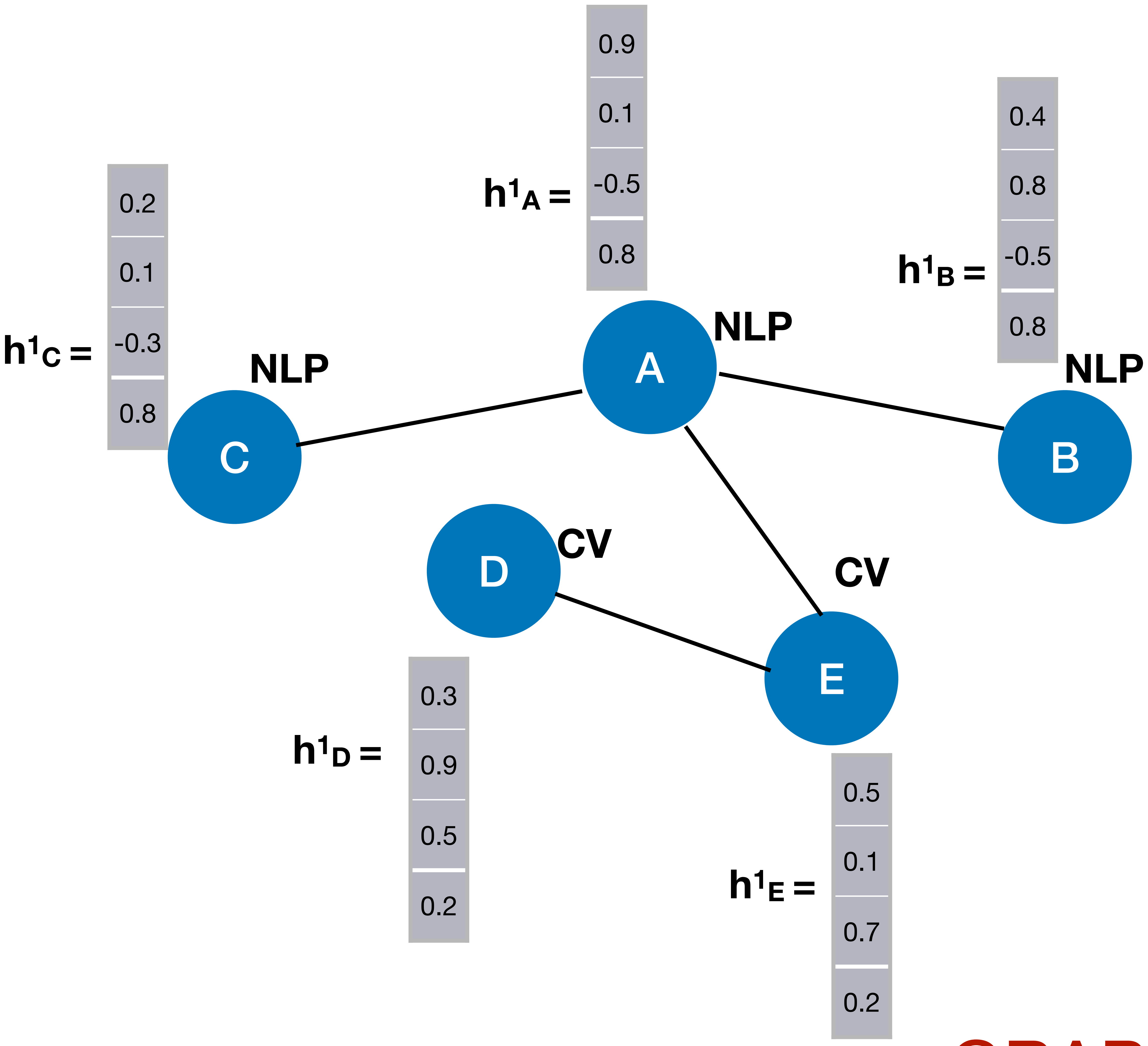

Aggregation:

$$\mathbf{t^1_v = f\,(h^0_{u_1,}\, h^0_{u_2,}\, h^0_{u_3,} \ldots)\ ,\ \forall\ u_i\ \in \boldsymbol{N(v)}}$$

FC Layer and Activation:

$$\mathbf{h^1_v = \sigma\ (\ g\ (t^1_{u_1,}\, t^1_{u_2,}\, t^1_{u_3,} \ldots)\ ),\ \forall\ u_i\ \in \boldsymbol{N(v)}}$$

# STEP - 2.2 : LINEAR LAYER




Aggregation:

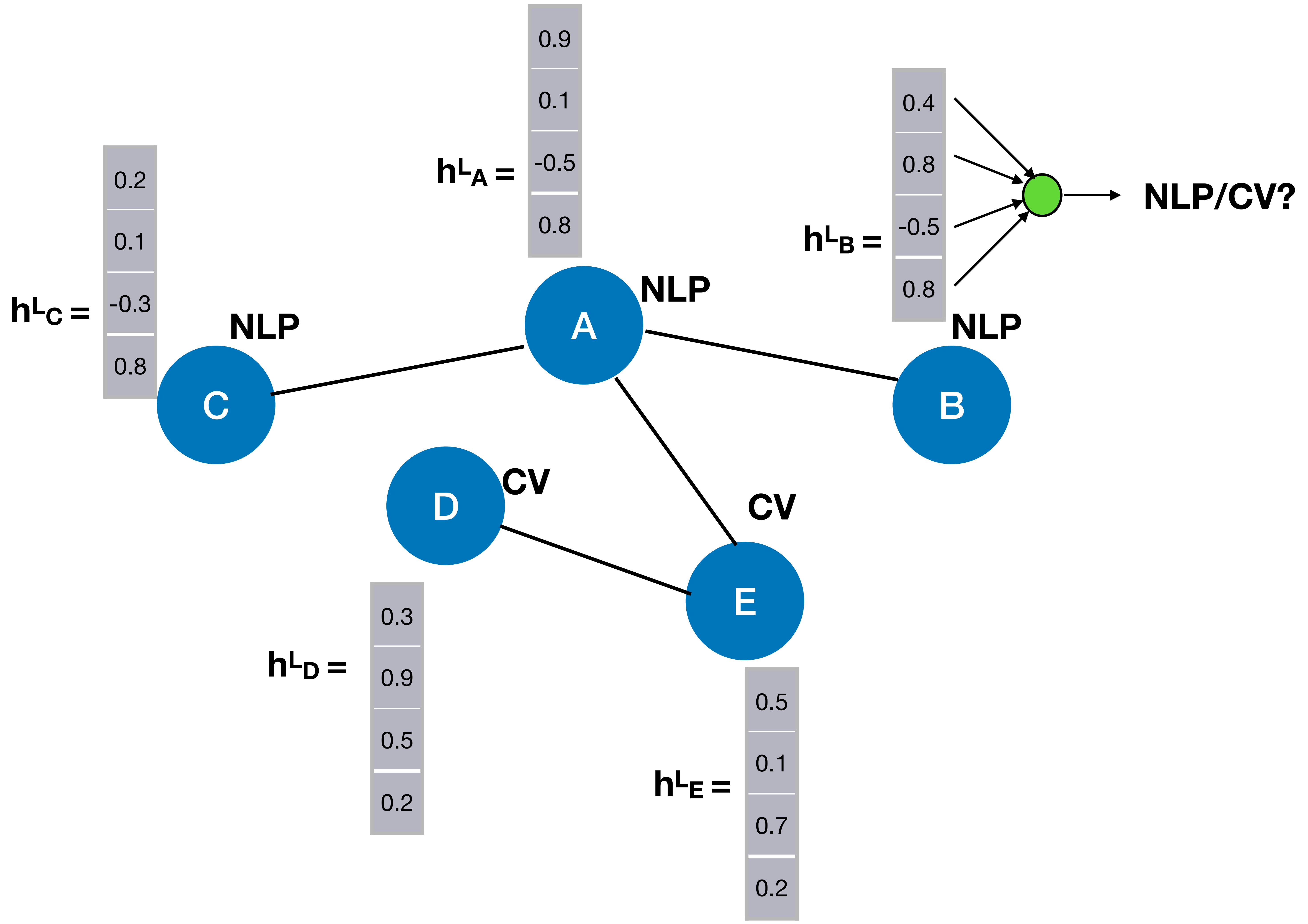
$$t^1_v = f\ (h^0_{u_1}, h^0_{u_2}, h^0_{u_3}, \ldots.)\ ,\ \forall\ u_i\ \in \boldsymbol{N(v)}$$

FC Layer and Activation:

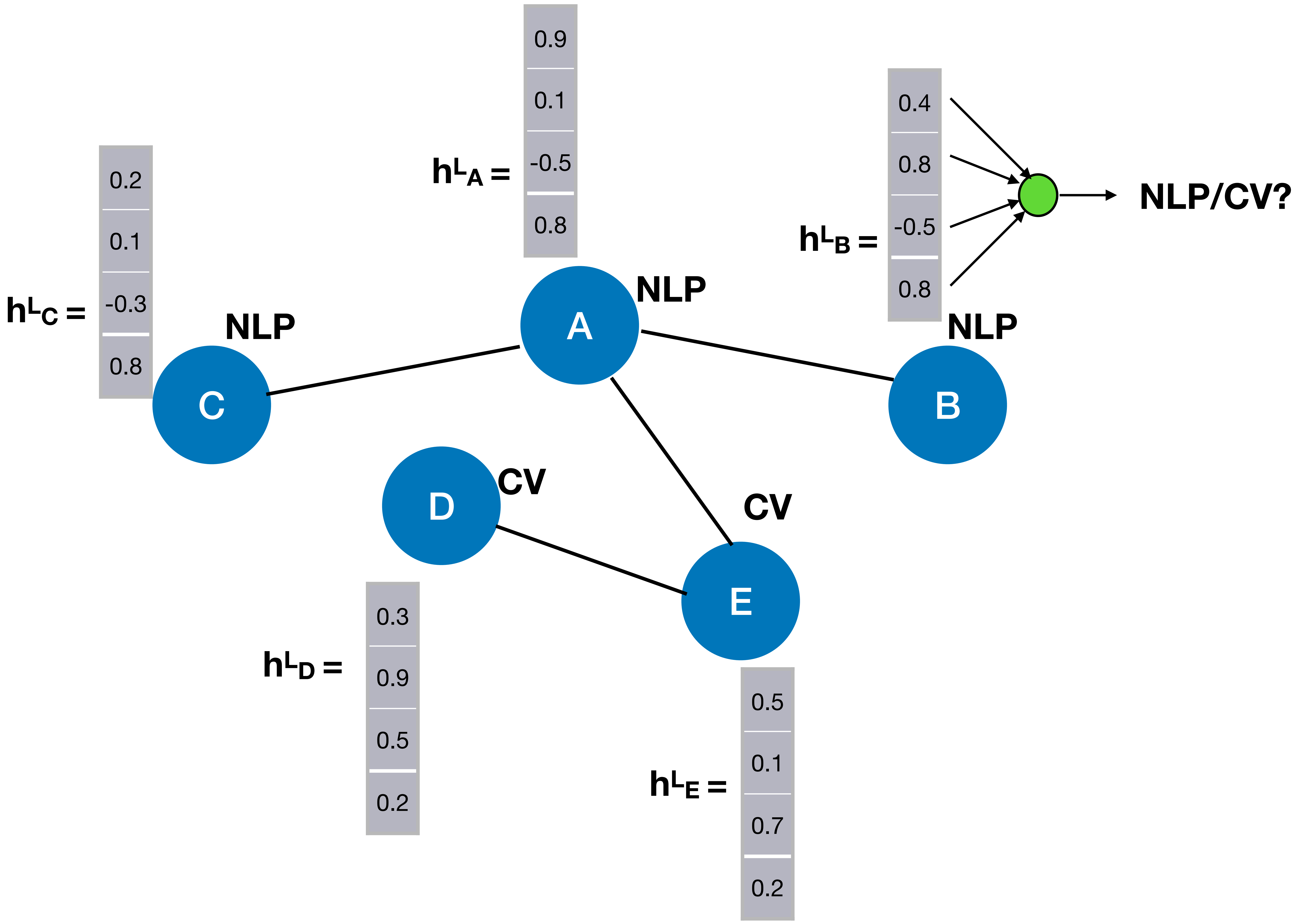
$$h^1_v = \sigma\ (\ g\ (t^1_{u_1}, t^1_{u_2}, t^1_{u_3}, \ldots.)\ ),\ \forall\ u_i\ \in \boldsymbol{N(v)}$$

GRAPH CONVOLUTIONAL NETWORK (GCN)!

# STEP - 3: CLASSIFICATION LAYER

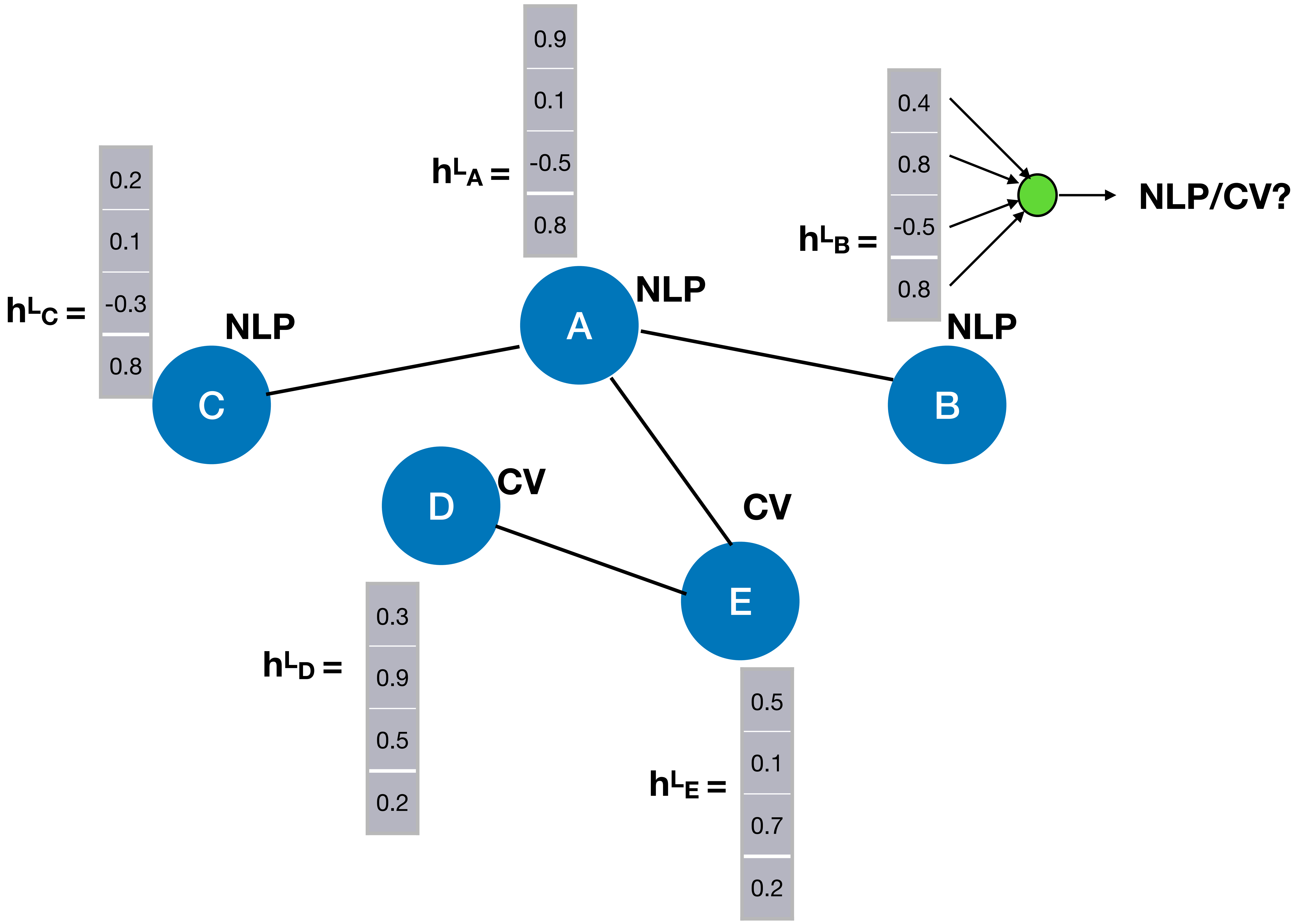


# STEP - 3: CLASSIFICATION LAYER




BINARY CROSS ENTROPY LOSS!

# STEP - 3: CLASSIFICATION LAYER




BINARY CROSS ENTROPY LOSS!

TRAIN!

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

One layer of GCN is applied in two steps:

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

One layer of GCN is applied in two steps:

- Aggregation

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

One layer of GCN is applied in two steps:

- Aggregation
- Linear layer application followed by non-linearity

# GRAPH CONVOLUTIONAL NETWORKS (GCN)

One layer of GCN is applied in two steps:

- Aggregation
- Linear layer application followed by non-linearity

Do this for every node!

# CNN VS GCN

# CNN VS GCN

## CNN

# CNN VS GCN

## CNN

STEP -1 : MATRIX MULTIPLICATION

# CNN VS GCN

## CNN

STEP -1 : MATRIX MULTIPLICATION

STEP-2 : CONTINUE FOR EACH PIXEL
WITH SHARED WEIGHTS

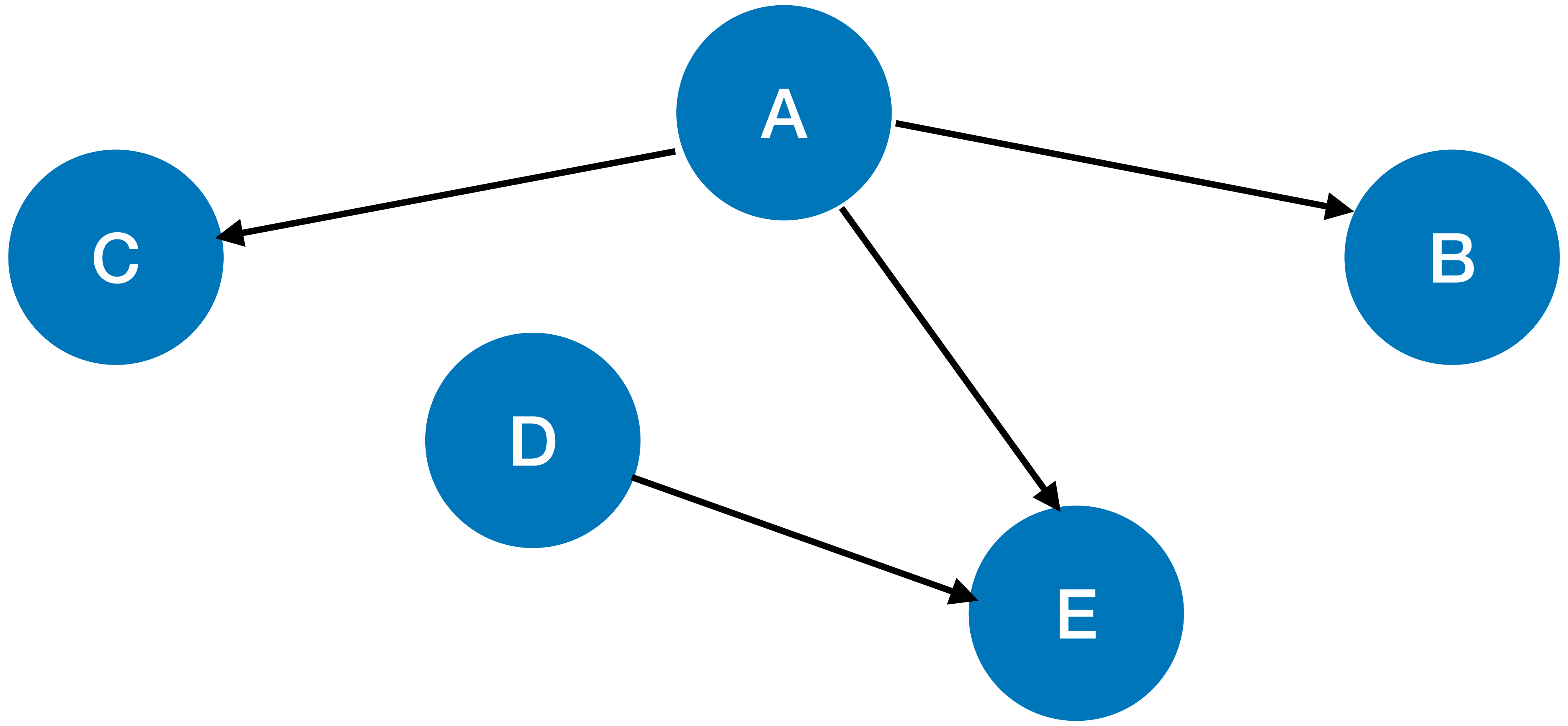
# CNN VS GCN

## CNN

STEP -1 : MATRIX MULTIPLICATION

STEP-2 : CONTINUE FOR EACH PIXEL
WITH SHARED WEIGHTS

## GCN



# CNN VS GCN

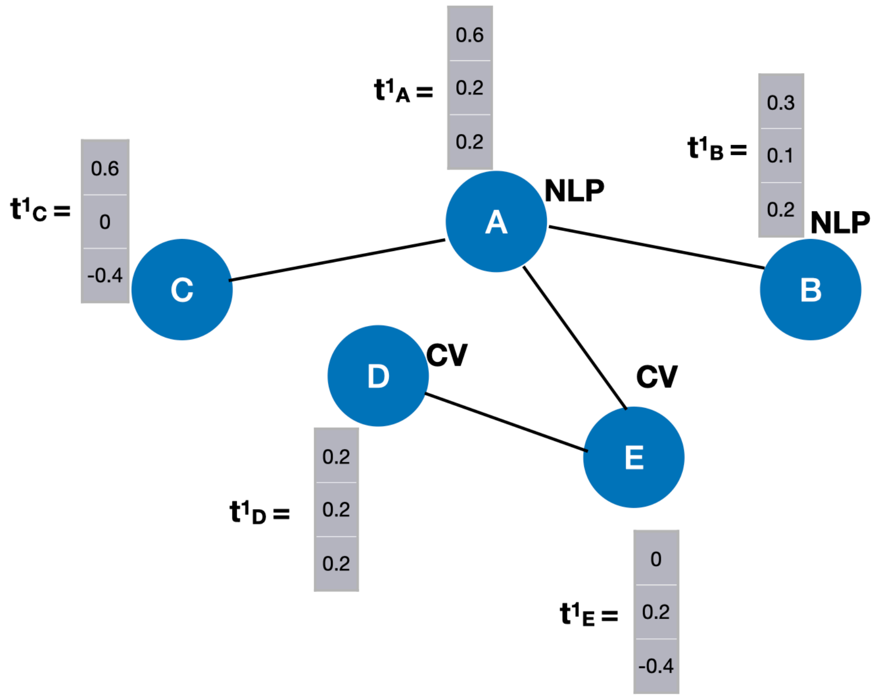
## CNN

STEP -1 : MATRIX MULTIPLICATION

STEP-2 : CONTINUE FOR EACH PIXEL
WITH SHARED WEIGHTS

## GCN

STEP -1 : AGGREGATION




Aggregation:

$t^1_v = f\,(h^0_{u_1}, h^0_{u_2}, h^0_{u_3}, \ldots.)\ ,\ \forall\ u_i \in N(v)$

# CNN VS GCN

## CNN

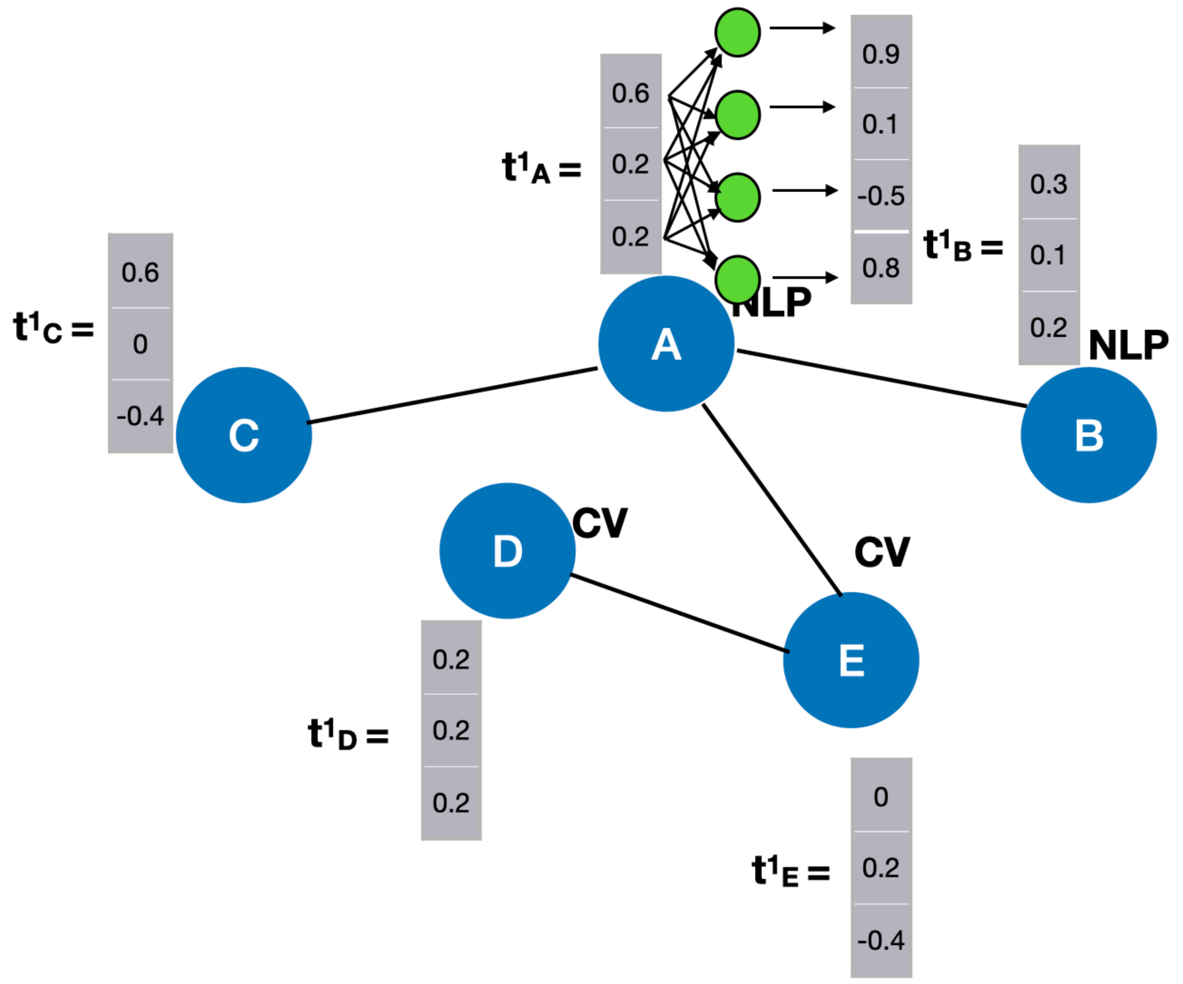
STEP -1 : MATRIX MULTIPLICATION

STEP-2 : CONTINUE FOR EACH PIXEL
WITH SHARED WEIGHTS

## GCN

STEP -1 : AGGREGATION

STEP-2: MATRIX MULTIPLICATION



# CNN VS GCN

## CNN

STEP -1 : MATRIX MULTIPLICATION

STEP-2 : CONTINUE FOR EACH PIXEL WITH SHARED WEIGHTS

## GCN

STEP -1 : AGGREGATION

STEP-2: MATRIX MULTIPLICATION

STEP-3: CONTINUE FOR EACH PIXEL WITH SHARED WEIGHTS